Introduction: Multimedia Databases
Outline of Today’s Lecture

- What is multimedia data?
- What are multimedia data applications?
- How should multimedia applications access different media types?
- What kinds of operations do users wish to perform on multimedia data?
- What infrastructure is needed to support these functionalities:
  - **Content**: Extracting media content. Indexing it. Querying it.
  - **Physical Storage**: Storing media data on primary, secondary, and tertiary storage devices. Building media servers.
  - **Creating Presentations**: How should a media presentation be created (perhaps in response to a query)? How should it be delivered to users at remote sites?
- Overview of the course
Today’s Media Types

- Text/Document
- Image
- Video
- Audio
- “Classical” Data (e.g. relations, flat files, object bases, etc.).

Video and audio differ from the other media types listed above because of their temporal nature. In particular:

- Video/Audio retrievals must appear to be continuous, hiccup-free presentations.
- Video/Audio support operations like fast-forward, rewind, and pause, that were not supported by classical data types.
What is an MM_DBMS

A multimedia database management system (MM_DBMS) is a framework that manages different types of data potentially represented in a wide diversity of formats on a wide array of media sources. An MM_DBMS must:

- Have the ability to uniformly query data (media data, textual data) represented in different formats.
- Have the ability to simultaneously query different media sources and conduct classical database operations across them.
- Have the ability to retrieve media objects from a local storage device in a smooth jitter free (i.e. continuous) manner.
- Have the ability to take the answer generated by a query (the notion of “answer to a query” may be mathematical structure of some sort) and develop a presentation of that answer in terms of audio-visual media.
- Have the ability to deliver this presentation in a way that satisfies various Quality of Service requirements.
Sample Multimedia Scenario

Consider a police investigation of a large-scale drug operation. This investigation may generate the following types of data:

- **Video** data captured by surveillance cameras that record the activities taking place at various locations.

- **Audio** data captured by legally authorized telephone wire-taps.

- **Image** data consisting of still photographs taken by investigators.

- **Document** data seized by the police when raiding one or more places.

- **Structured relational data** containing background information, bank records, etc., of the suspects involved.

- **Geographic information systems** data containing geographic data relevant to the drug investigation being conducted.
Data sources used in Sample Multimedia Scenario

- Surveillance Video
- Audio (phone) data
- Geographic Info.
- Still Image Data
- Relational Data
- Document Data

Police Application
Example Image Queries for Multimedia Scenario

Query 1:

- Police officer John Macho-Dude has a photograph in front of him.
- He wants to find the identity of the person in the picture.
- Query: “Retrieve all images from the image library in which the person appearing in the (currently displayed) photograph appears.”

Query 2:

- Police officer John Macho-Dude wants to examine pictures of Denis Dopeman.
- Query: “Retrieve all images from the image library in which Denis Dopeman appears.”
Issues Raised by Example Image Queries

- Two basic kinds of queries:
  - Image-based queries
  - Keyword-based queries
- In the first, police officer John Macho-Dude, gives a image as input (query image). He wants as output, a ranked list of images that are “similar” to the query image.
- To support this, we need to know what “similarity” means.
- We need to know what “ranking” means.
- We need to be able to efficiently support these operations.
- In the second, police officer John Macho-Dude, gives a keyword as input (name of suspect – Denis Dopeman). He wants as output, those photographs that are known to contain an image-object whose name attribute is Denis Dopeman.
- To support this, we need to know how to associate different attributes with images (or parts of images).
- We need to know how to effectively index and retrieve images based on such attributed.
Example Audio Query in Sample Multimedia Scenario

Query 1:

- Police officer John Macho-Dude is listening to an audio surveillance tape.
- Example: Tape contains a conversation between individual A (person under surveillance) and individual B (somebody meeting person A).
- Query: “Find the identity of individual B, given that individual A is Denis Dopeman.”

Query 2:

- Police officer John Macho-Dude wants to review all audios that Denis Dopeman participated in during some specified time period.
- Query: “Find all audio tapes in which Denis Dopeman was a participant.”
Example Text/Video Query

Text Query:

- Police officer John Macho-Dude is browsing an archive of text documents – these include old newspaper archives, police department files on old, unsolved murder cases, witness statements, etc.

- Query: “Find all documents (from the corpora of text documents) that deal with the Cali drug cartel’s financial transactions with ABC Corp.”

Video Query:

- Police officer John Macho-Dude is examining a surveillance video of a particular person being fatally assaulted by an assailant. However, the assailant’s face is occluded and image processing algorithms return very poor matches. John Macho-Dude thinks the assault was by someone known to the victim.

- Query: “Find all video segments in which the victim of the assault appears.”

- By examining the answer to the above query, John Macho-Dude hopes to find other people who have previously interacted with the victim.
Simple Heterogeneous Query

- All queries discussed thus far involve one media type.
- For example, each query accesses only image or audio or video data, but does not access a mix of these media types.
- Complex queries will “mix and match” data from these different media sources.
- Such “mixing and matching” is difficult, even for purely textual data sources.
- TEXTUAL EXAMPLE: Find all individuals
  - who have been convicted of attempted murder in North America and
  - who have recently had electronic fund transfers made into their bank accounts from ABC Corp.
- Answering this query is problematic because:
  - Determining all people convicted of different crimes may require accessing a wide variety of databases belonging to different police jurisdictions and courts
  - ABC Corp. may have accounts in hundreds of banks worldwide each of which uses different formats and different database systems
Heterogeneous Multimedia Query

- Find all individuals who have been photographed with Jose Orojuelo and who have been convicted of attempted murder in North America and who have recently had electronic fund transfers made into their bank accounts from ABC Corp.

- This query requires that:
  - We find all people satisfying the conditions in the "Simple" Heterogeneous Query and
  - We access a mugshot-database, **mugshotdb**, containing the names and pictures of various individuals
  - We access a surveillance photograph database of still images
  - We access a **surveillance video** database to see if a meeting between the suspect and Jose Orojuelo was recorded on video
  - We access **image processing algorithms** to determine who occurs in which video/still photograph.
Multimedia Research Issues: Queries

Let us suppose (for now) that we already have a pre-existing body of multimedia data that we wish to access.

- Need a single language within which multimedia data of different types can be accessed.
- In classical database theory, there are binary operations that combine different relations in different ways (e.g. join, union, difference, intersection, Cartesian Product). In the same way, this language must be able to specify combination operations across different media types (rather than just across different relations).
- This language must be able to access:
  - “metadata” describing the content of different media sources and
  - “raw” data supported by the different media sources.
- This language must be able to merge, manipulate and “join” together, results from different media sources.
- Once such languages are devised, we need techniques to:
  - optimize a single query (through the notion of a query plan
  - develop servers that can optimize processing of a set of queries.
Multimedia Research Issues: Content

- What is content of a media source? Under what conditions can content be described textually and under what conditions must it be described directly through the original media type?
- How should we extract the content of:
  - an image
  - a video-clip
  - an audio-clip
  - a free/structured text document.
- How should we index the results of this extracted content?
- What is retrieval by similarity?
- What algorithms can be used to efficiently retrieve media data on the basis of similarity?
- If one has to design a multimedia database using legacy media sources as well as to-be-organized media data, what is the best way of creating such a database?
- How should queries be relaxed so that not only the originally stated query, but also “similar” queries get processed?
- What are efficient algorithms for processing such queries?
Multimedia Research Issues: Storage

- How do the following (standard) storage devices work?
  - disk systems
  - CD-ROM systems
  - tape systems and tape libraries
- How is data laid out on such devices?
- How do we design disk/CD-ROM/tape servers so as to optimally satisfy different clients concurrently, when these clients execute the following operations
  - playback
  - rewind
  - fast forward
  - pause
Multimedia Research Issues: Presentations and Delivery

- How do we specify the content of multimedia presentations?
- How do we specify the form (temporal/spatial layout) of this content?
- How do we create a presentation schedule that satisfies these temporal/spatial presentation requirements?
- How can we deliver a multimedia presentation to users when there is:
  - a need to interact with other remote servers to assemble the presentation (or parts of it)
  - a bound on the buffer, bandwidth, load, and other resources available on the system
  - a mismatch between the host server’s capabilities and the customer’s machine capabilities?
- How can such presentations optimize Quality of Service?
Overview of the Textbook

• Part I: Preliminaries
  – Introduction to Relational DBMSs
  – Introduction to Object Oriented Database Management Systems

• Part II: Organizing Multimedia Content
  – Specialized Indexing Structures
  – Image Databases
  – Text/Document Databases
  – Video Databases
  – Audio Databases
  – Multimedia Databases (consisting of a mix of all the above, as well as other unlisted types).

• Part III: Physical Storage and Retrieval
  – Retrieving Multimedia Data from Disks
  – Retrieving Multimedia Data from CD-ROMS
  – Retrieving Multimedia Data from Tapes

• Part IV: Creating and Delivering Multimedia Presentations
  – Creating Distributed Multimedia Presentations
  – Distributed Media Servers
  – Future Directions
# Plan for this Course (Undergraduate Level)

<table>
<thead>
<tr>
<th>Week</th>
<th>Chapter/Sections</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ch. 1-2</td>
<td>Half a lecture on Introduction, rest of Chapter 2.</td>
</tr>
<tr>
<td>2</td>
<td>Ch. 3</td>
<td>One lecture on Secs. 3.1-3.3, another on Sec. 3.4-3.5</td>
</tr>
<tr>
<td>3</td>
<td>Secs. 4.1-4.2</td>
<td>One lecture on k-d trees, one on point quadrees.</td>
</tr>
<tr>
<td>4</td>
<td>Secs. 4.3-4.5</td>
<td>One lecture on MX-quadrees, one on R-trees.</td>
</tr>
<tr>
<td>5</td>
<td>Secs. 5.1-5.6,5.8</td>
<td>Skip section 5.7</td>
</tr>
<tr>
<td>6</td>
<td>Ch. 6</td>
<td>Use lots of examples, drop mathematical details on Sec. 6.4</td>
</tr>
<tr>
<td>7</td>
<td>Ch. 7</td>
<td>One lecture on Secs. 7.1, 7.2.1, one lecture on Secs. 7.2.3, 7.3</td>
</tr>
<tr>
<td>8</td>
<td>Secs. 9.1-9.4</td>
<td>One lecture on Secs. 9.1-9.2, one on Secs. 9.3-9.4</td>
</tr>
<tr>
<td>9</td>
<td>Secs. 9.1-9.4</td>
<td>One lecture on Secs. 9.1-9.2, one on Secs. 9.3-9.4</td>
</tr>
<tr>
<td>10</td>
<td>Secs. 9.5-9.6</td>
<td>One lecture on Sec. 9.5, one on Sec. 9.6</td>
</tr>
<tr>
<td>11</td>
<td>Sec. 10.1-10.4</td>
<td>Explain Sec. 10.4 by example.</td>
</tr>
<tr>
<td>12</td>
<td>Secs. 11.1-11.3</td>
<td>One lecture on CD-Roms, skim mathematics</td>
</tr>
<tr>
<td></td>
<td>Sec. 12.1-12.2</td>
<td>One lecture on tapes, skim mathematics.</td>
</tr>
<tr>
<td>13</td>
<td>Ch. 13</td>
<td>One lecture on Secs. 13.1-13.2, one on Secs. 13.3-13.5</td>
</tr>
<tr>
<td>14</td>
<td>Ch. 13</td>
<td>One lecture on Secs. 14.1-14.4, one on Sec. 14.5</td>
</tr>
</tbody>
</table>
Plan for this Course (Graduate Level)

<table>
<thead>
<tr>
<th>Week</th>
<th>Chapter/Sections</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ch. 1-3</td>
<td>Brief Introduction. What is a multimedia database?</td>
</tr>
<tr>
<td>2</td>
<td>Sec. 4.1,4.2</td>
<td>1 lecture each on k-d trees and Point Quadtrees. Explain why these are useful for images.</td>
</tr>
<tr>
<td>3</td>
<td>Sec 4.3,4.4</td>
<td>1 lecture each on MX-quadtrees and R-trees.</td>
</tr>
<tr>
<td>4</td>
<td>Ch. 5</td>
<td>one lecture on Secs. 5.1-5.4, rest in second lecture.</td>
</tr>
<tr>
<td>5</td>
<td>Ch. 6</td>
<td>one lecture of Secs. 6.1-6.5, one lecture on Secs. 6.6-6.7</td>
</tr>
<tr>
<td>6</td>
<td>Ch. 7</td>
<td>one lecture on Secs. 7.1, 7.2.1, one lecture on Secs. 7.2.3, 7.3</td>
</tr>
<tr>
<td>7</td>
<td>Ch. 8 + midterm exam</td>
<td>One lecture on Ch. 7,</td>
</tr>
<tr>
<td>8</td>
<td>Secs. 9.1-9.4</td>
<td>One lecture on Secs. 9.1-9.2, one on Secs. 9.3-9.4</td>
</tr>
<tr>
<td>9</td>
<td>Secs. 9.5-9.6, 10.1</td>
<td>One lecture on sec. 9.5-9.6, one on Secs. 10.1</td>
</tr>
<tr>
<td>10</td>
<td>Secs.10.2-10.4</td>
<td>One lecture on Sec. 10.2-10.3, one on sec. 10.4</td>
</tr>
<tr>
<td>11</td>
<td>Ch. 11</td>
<td>Use plenty of examples contained in Overheads.</td>
</tr>
<tr>
<td>12</td>
<td>Ch. 12</td>
<td>Use plenty of examples contained in Overheads.</td>
</tr>
<tr>
<td>13</td>
<td>Ch. 13</td>
<td>One lecture on Secs. 13.1-13.2, one on Secs. 13.3-13.5</td>
</tr>
<tr>
<td>14</td>
<td>Ch. 14</td>
<td>One lecture on Secs. 14.1-14.4, one on Sec. 14.5</td>
</tr>
</tbody>
</table>
Relational DBMSs
Relational DBMSs

- A relational DBMS manipulates *tables*.
- A table consists of *rows* and *columns*.
- Each row is called a *tuple*.
- Each column represents an *attribute*.
- Every attribute has an associated set, called the *domain* of the attribute.
- The *i*’th entry in a given tuple is in the *i*’th column of the table.
- The *i*’th entry in a given tuple is drawn from the domain of the attribute associated with the *i*’th column of the table.
Example

- Consider a relation called **emp**.
- The schema of this relation may be

\[(\text{COMP}, \text{SSN}, \text{FNAME}, \text{LNAME}, \text{STREETNUM}, \text{STREETNAME}, \text{CITY}, \text{STATE}, \text{ZIP})\]

where:

1. **COMP, FNAME, LNAME, STREETNAME, CITY** all denote the set of alphanumeric strings,
2. **STATE** denotes the set of all two-letter strings denoting US states (e.g. MD, VA, etc.) – this represents an enumerated type,
3. **SSN** denotes the set of nine-digit positive integers,
4. **ZIP** denotes the set of five-digit positive integers.
5. **STREETNUM** denotes the set of positive integers.

An instance of the **emp** relation may be:

<table>
<thead>
<tr>
<th>COMP</th>
<th>SSN</th>
<th>FNAME</th>
<th>LNAME</th>
<th>STREETNUM</th>
<th>STREETNAME</th>
<th>CITY</th>
<th>STATE</th>
<th>ZIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABC Corp</td>
<td>992466009</td>
<td>John</td>
<td>Smith</td>
<td>27</td>
<td>Casal St.</td>
<td>Fairfax</td>
<td>VA</td>
<td>22087</td>
</tr>
<tr>
<td>ABC Corp</td>
<td>247890123</td>
<td>Donna</td>
<td>Jones</td>
<td>789</td>
<td>Baker St.</td>
<td>McLean</td>
<td>VA</td>
<td>22185</td>
</tr>
<tr>
<td>ABC Corp</td>
<td>567890123</td>
<td>Anne</td>
<td>Cox</td>
<td>12345</td>
<td>Sunny Dr.</td>
<td>Herndon</td>
<td>MD</td>
<td>20089</td>
</tr>
<tr>
<td>ABC Corp</td>
<td>987654321</td>
<td>Lisa</td>
<td>Martinez</td>
<td>17</td>
<td>Edgar St.</td>
<td>Rockville</td>
<td>MD</td>
<td>20085</td>
</tr>
<tr>
<td>ABC Corp</td>
<td>189146590</td>
<td>Bill</td>
<td>Davis</td>
<td>26</td>
<td>Casal St.</td>
<td>Parkers</td>
<td>VA</td>
<td>22089</td>
</tr>
<tr>
<td>XYZ Corp</td>
<td>876543210</td>
<td>Bob</td>
<td>Brown</td>
<td>11</td>
<td>Lake Dr.</td>
<td>Richmond</td>
<td>VA</td>
<td>20876</td>
</tr>
<tr>
<td>XYZ Corp</td>
<td>897654231</td>
<td>Bill</td>
<td>Davis</td>
<td>26</td>
<td>Casal St.</td>
<td>Parkers</td>
<td>VA</td>
<td>22087</td>
</tr>
<tr>
<td>XYZ Corp</td>
<td>897654231</td>
<td>Bob</td>
<td>Davis</td>
<td>26</td>
<td>Casal St.</td>
<td>Parkers</td>
<td>VA</td>
<td>22087</td>
</tr>
<tr>
<td>XYZ Corp</td>
<td>897654231</td>
<td>Bob</td>
<td>Davis</td>
<td>26</td>
<td>Casal St.</td>
<td>Parkers</td>
<td>VA</td>
<td>22087</td>
</tr>
</tbody>
</table>
Relational Algebra

The relational algebra consists of a set of operations that can be used to manipulate tables. Each operation in the algebra takes one or more tables as input, and produces a table as output. The standard operations in RA are listed below.

- Selection
- Projection
- Cartesian Product
- Union
- Difference
- Join
- Intersection
Selection

- Takes as input, a relation $R$ and a condition $C$.
- $\sigma_C(R)$ says: “Select all tuples in relation $R$ that satisfy condition $C$.”
- The result of executing $\sigma_C(R)$ is a set of tuples drawn from $R$.
- EX: Suppose we wish to select all tuples in the $\mathsf{emp}$ relation that have $\mathsf{ABC Corp.}$ as its $\mathsf{COMP}$ field. This can be expressed as the selection:
  $$\sigma_{\mathsf{COMP}=\mathsf{ABC Corp.}}(\mathsf{emp})$$
- Result:

<table>
<thead>
<tr>
<th>COMP</th>
<th>SSN</th>
<th>FNAME</th>
<th>LNAME</th>
<th>STREET</th>
<th>STREET NAME</th>
<th>CITY</th>
<th>STATE</th>
<th>ZIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABC Corp.</td>
<td>992786589</td>
<td>John</td>
<td>Smith</td>
<td>29</td>
<td>Canal St.</td>
<td>Fairfax</td>
<td>VA</td>
<td>22037</td>
</tr>
<tr>
<td>ABC Corp.</td>
<td>287606724</td>
<td>Dennis</td>
<td>Jones</td>
<td>120</td>
<td>Baker St.</td>
<td>Manassas</td>
<td>VA</td>
<td>22400</td>
</tr>
<tr>
<td>ABC Corp.</td>
<td>489221704</td>
<td>Jane</td>
<td>Fox</td>
<td>1234</td>
<td>Cooper Dr.</td>
<td>Rockville</td>
<td>MD</td>
<td>20887</td>
</tr>
<tr>
<td>ABC Corp.</td>
<td>983764470</td>
<td>Lisa</td>
<td>Barnes</td>
<td>17</td>
<td>Edgar Ct.</td>
<td>Rockville</td>
<td>MD</td>
<td>20887</td>
</tr>
<tr>
<td>ABC Corp.</td>
<td>169466394</td>
<td>Sue</td>
<td>Davis</td>
<td>26</td>
<td>Canal St.</td>
<td>Fairfax</td>
<td>VA</td>
<td>22037</td>
</tr>
</tbody>
</table>

---

V.S. Subrahmanian — All Rights Reserved

These slides may not be duplicated without explicit written permission from Morgan Kaufmann Press.
Principles of Multimedia Database Systems Morgan Kaufmann Copyright ©1997
Projection

- We specify a relation and one or more columns from that relation that we are interested in.
- The result of a projection is a table that is identical to the original relation, except that all columns not explicitly named are eliminated.
- EX: Suppose we wish to project out the COMP and CITY fields of the emp relation.
- This is specified in the relational algebra as
  \[\pi_{\text{COMP}, \text{CITY}}(\text{emp})\]
- Result:

<table>
<thead>
<tr>
<th>COMP</th>
<th>CITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABC Corp.</td>
<td>Fairfax</td>
</tr>
<tr>
<td>ABC Corp.</td>
<td>Manassas</td>
</tr>
<tr>
<td>ABC Corp.</td>
<td>Bethesda</td>
</tr>
<tr>
<td>ABC Corp.</td>
<td>Rockville</td>
</tr>
<tr>
<td>XYZ Corp.</td>
<td>Richmond</td>
</tr>
<tr>
<td>XYZ Corp.</td>
<td>Baltimore</td>
</tr>
<tr>
<td>XYZ Corp.</td>
<td>Rockville</td>
</tr>
<tr>
<td>XYZ Corp.</td>
<td>Fairfax</td>
</tr>
<tr>
<td>XYZ Corp.</td>
<td>Bethesda</td>
</tr>
</tbody>
</table>
• Note: The tuple (ABC Corp., Fairfax) could be placed in the result of the projection in two ways – either via the record associated with John Smith, or via the record associated with Jill Davis. As both these (different) tuples in the original emp relation agree on the fields being projected, we place only one copy of the resulting projected tuple, (ABC Corp., Fairfax), instead of both.
Cartesian Product

- Let relation $R_1$ have the schema $(A_1, \ldots, A_n)$.
- Let $R_2$ have the schema $(A'_1, \ldots, A'_m)$.
- The Cartesian Product, $R_1 \times R_2$, has the scheme $(A_1, \ldots, A_n, A'_1, \ldots, A'_m)$ and consists of all tuples $(t_1, \ldots, t_n, t'_1, \ldots, t'_m)$ such that:
  - $(t_1, \ldots, t_n)$ is a tuple in $R_1$ and $(t'_1, \ldots, t'_m)$ is a tuple in $R_2$.
- **EX:** Let $R_1 = \pi_{\text{COMP,CITY}}(\text{emp})$ and let $R_2$ be

<table>
<thead>
<tr>
<th>COMPANY</th>
<th>EMPLOYEES</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABC Corp.</td>
<td>30000</td>
</tr>
<tr>
<td>XYZ Corp.</td>
<td>15000</td>
</tr>
</tbody>
</table>

- Then $R_1 \times R_2$ is:
<table>
<thead>
<tr>
<th>COMP</th>
<th>CITY</th>
<th>COMPANY</th>
<th>EMPLOYEES</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABC Corp.</td>
<td>Fairfax</td>
<td>ABC Corp.</td>
<td>30000</td>
</tr>
<tr>
<td>ABC Corp.</td>
<td>Manassas</td>
<td>ABC Corp.</td>
<td>30000</td>
</tr>
<tr>
<td>ABC Corp.</td>
<td>Bethesda</td>
<td>ABC Corp.</td>
<td>30000</td>
</tr>
<tr>
<td>ABC Corp.</td>
<td>Rockville</td>
<td>ABC Corp.</td>
<td>30000</td>
</tr>
<tr>
<td>ABC Corp.</td>
<td>Fairfax</td>
<td>XYZ Corp.</td>
<td>15000</td>
</tr>
<tr>
<td>ABC Corp.</td>
<td>Manassas</td>
<td>XYZ Corp.</td>
<td>15000</td>
</tr>
<tr>
<td>ABC Corp.</td>
<td>Bethesda</td>
<td>XYZ Corp.</td>
<td>15000</td>
</tr>
<tr>
<td>ABC Corp.</td>
<td>Rockville</td>
<td>XYZ Corp.</td>
<td>15000</td>
</tr>
<tr>
<td>XYZ Corp.</td>
<td>Richmond</td>
<td>ABC Corp.</td>
<td>30000</td>
</tr>
<tr>
<td>XYZ Corp.</td>
<td>Baltimore</td>
<td>ABC Corp.</td>
<td>30000</td>
</tr>
<tr>
<td>XYZ Corp.</td>
<td>Rockville</td>
<td>ABC Corp.</td>
<td>30000</td>
</tr>
<tr>
<td>XYZ Corp.</td>
<td>Fairfax</td>
<td>ABC Corp.</td>
<td>30000</td>
</tr>
<tr>
<td>XYZ Corp.</td>
<td>Bethesda</td>
<td>XYZ Corp.</td>
<td>15000</td>
</tr>
<tr>
<td>XYZ Corp.</td>
<td>Richmond</td>
<td>XYZ Corp.</td>
<td>15000</td>
</tr>
<tr>
<td>XYZ Corp.</td>
<td>Baltimore</td>
<td>XYZ Corp.</td>
<td>15000</td>
</tr>
<tr>
<td>XYZ Corp.</td>
<td>Rockville</td>
<td>XYZ Corp.</td>
<td>15000</td>
</tr>
<tr>
<td>XYZ Corp.</td>
<td>Fairfax</td>
<td>XYZ Corp.</td>
<td>15000</td>
</tr>
<tr>
<td>XYZ Corp.</td>
<td>Bethesda</td>
<td>XYZ Corp.</td>
<td>15000</td>
</tr>
</tbody>
</table>
Union

- Relations $R_1$ and $R_2$ are union-compatible if they have the same scheme.
- Union of relations $R_1$ and $R_2$ in this case, denoted $R_1 \cup R_2$, has the same scheme as $R_1$ (and $R_2$).
- $R_1 \cup R_2$ contains a tuple $t$ iff $t$ is either in relation $R_1$ or in relation $R_2$.
- Let $R_1$ be the relation $R_1 = \pi_{\text{FNAME, LNAME}}(\text{emp})$ and suppose $R_2$ is the relation given by:

<table>
<thead>
<tr>
<th>FNAME</th>
<th>LNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jack</td>
<td>Arnold</td>
</tr>
<tr>
<td>John</td>
<td>Smith</td>
</tr>
<tr>
<td>Ted</td>
<td>Garroway</td>
</tr>
</tbody>
</table>

- Then $R_1 \cup R_2$ is:
<table>
<thead>
<tr>
<th>FNAME</th>
<th>LNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jack</td>
<td>Arnold</td>
</tr>
<tr>
<td>John</td>
<td>Smith</td>
</tr>
<tr>
<td>Ted</td>
<td>Garroway</td>
</tr>
<tr>
<td>Denis</td>
<td>Jones</td>
</tr>
<tr>
<td>Jane</td>
<td>Fox</td>
</tr>
<tr>
<td>Lisa</td>
<td>Barnes</td>
</tr>
<tr>
<td>Jill</td>
<td>Davis</td>
</tr>
<tr>
<td>Bill</td>
<td>Bosco</td>
</tr>
<tr>
<td>Bill</td>
<td>Dashell</td>
</tr>
<tr>
<td>David</td>
<td>Johns</td>
</tr>
<tr>
<td>Jim</td>
<td>Hatch</td>
</tr>
<tr>
<td>Tina</td>
<td>Budge</td>
</tr>
</tbody>
</table>
Difference

- $R_1$ and $R_2$ are *difference-compatible* if they have the same scheme.
- Difference of $R_1$ and $R_2$ is denoted $R_1 - R_2$.
- $R_1 - R_2$ contains a tuple $t$ iff $t$ is in $R_1$, but is not in $R_2$.
- With the previous example, we see that $R_1 - R_2$ is

<table>
<thead>
<tr>
<th>FNAME</th>
<th>LNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>Denis</td>
<td>Jones</td>
</tr>
<tr>
<td>Jane</td>
<td>Fox</td>
</tr>
<tr>
<td>Lisa</td>
<td>Barnes</td>
</tr>
<tr>
<td>Jill</td>
<td>Davis</td>
</tr>
<tr>
<td>Bill</td>
<td>Bosco</td>
</tr>
<tr>
<td>Bill</td>
<td>Dashell</td>
</tr>
<tr>
<td>David</td>
<td>Johns</td>
</tr>
<tr>
<td>Jim</td>
<td>Hatch</td>
</tr>
<tr>
<td>Tina</td>
<td>Budge</td>
</tr>
</tbody>
</table>
Join

- The **join** operation takes as input, two tables and a boolean condition, \( C \).

- Typically, the boolean condition \( C \) involves a condition linking a given attribute in the first table with an attribute in the second.

- **EX:** Suppose we have a relation called **crime** of the form shown below:

<table>
<thead>
<tr>
<th>SSN</th>
<th>FIRST</th>
<th>LAST</th>
<th>CONVICTION</th>
<th>DAY</th>
<th>MONTH</th>
<th>YEAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>992786589</td>
<td>John</td>
<td>Smith</td>
<td>drug</td>
<td>17</td>
<td>may</td>
<td>1990</td>
</tr>
<tr>
<td>992786589</td>
<td>John</td>
<td>Smith</td>
<td>assault</td>
<td>5</td>
<td>aug</td>
<td>1986</td>
</tr>
<tr>
<td>983744470</td>
<td>Lisa</td>
<td>Barnes</td>
<td>tax fraud</td>
<td>14</td>
<td>june</td>
<td>1987</td>
</tr>
<tr>
<td>837464632</td>
<td>Bill</td>
<td>Dashell</td>
<td>drug</td>
<td>27</td>
<td>sep</td>
<td>1990</td>
</tr>
<tr>
<td>837464632</td>
<td>Bill</td>
<td>Dashell</td>
<td>theft</td>
<td>11</td>
<td>jul</td>
<td>1986</td>
</tr>
<tr>
<td>193284646</td>
<td>Jim</td>
<td>Hatch</td>
<td>mail fraud</td>
<td>21</td>
<td>aug</td>
<td>1984</td>
</tr>
<tr>
<td>193284646</td>
<td>Jim</td>
<td>Hatch</td>
<td>drug</td>
<td>14</td>
<td>feb</td>
<td>1989</td>
</tr>
</tbody>
</table>

Suppose we wish to create a table that has the scheme \((\text{LNAME}, \text{FNAME}, \text{CITY}, \text{CONVICTION})\) – find all people (and the cities they live in) in the **emp** relation who have been convicted.

- This requires that we:

  1. match social security numbers from the two relations **emp** and **crime** to “match” a person in the **emp** relation with a person in the **crime** relation, and

  2. then he needs to project out the relevant fields (i.e. \( \text{LNAME}, \text{FNAME} \) and \( \text{CONVICTION} \)).
• First operation can be done by \\

$$\text{emp \bowtie}_{\text{emp.SSN}=\text{crime.SSN}} \text{crime.}$$

• Both operations can be done by

$$\pi_{FNAME,LNAME,CITY,CONVICTION}(\text{emp \bowtie}_{\text{emp.SSN}=\text{crime.SSN}} \text{crime}).$$

• Result:

<table>
<thead>
<tr>
<th>FNAME</th>
<th>LNAME</th>
<th>CITY</th>
<th>CONVICTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Smith</td>
<td>Fairfax</td>
<td>drug</td>
</tr>
<tr>
<td>John</td>
<td>Smith</td>
<td>Fairfax</td>
<td>assault</td>
</tr>
<tr>
<td>Lisa</td>
<td>Barnes</td>
<td>Rockville</td>
<td>tax fraud</td>
</tr>
<tr>
<td>Bill</td>
<td>Dashell</td>
<td>Baltimore</td>
<td>drug</td>
</tr>
<tr>
<td>Bill</td>
<td>Dashell</td>
<td>Baltimore</td>
<td>theft</td>
</tr>
<tr>
<td>Jim</td>
<td>Hatch</td>
<td>Fairfax</td>
<td>mail fraud</td>
</tr>
<tr>
<td>Jim</td>
<td>Hatch</td>
<td>Fairfax</td>
<td>drug</td>
</tr>
</tbody>
</table>
Join (Contd.)

- Suppose $R_1$ and $R_2$ are two relations whose attribute names are distinct.

- Then $R_1 \bowtie_C R_2$ is given by:

$$R_1 \bowtie_C R_2 \overset{def}{=} (\sigma_C(R_1 \times R_2)).$$
Intersection

• Intersection of two relations $R_1, R_2$ is defined if $R_1, R_2$ are union compatible.

• Denoted $R_1 \cap R_2$.

• Definition:

$$R_1 \cap R_2 \overset{def}{=} R_1 - (R_1 - R_2).$$
Relational Calculus

• When a user specifies a query in the relational algebra, he specifies a *sequence* of operations to be performed.

• Example: Consider the relational algebra query:

\[ \pi_{\text{FNAME}, \text{LNAME}, \text{CITY}, \text{CONVICTION}}(\text{emp} \bowtie_{\text{emp.SSN} = \text{crime.SSN}} \text{crime}). \]

This query explicitly says: “First perform the specified join and then project out the specified columns.”

• Relation Calculus is declarative.

• A Relational Calculus Query specifies *what* to compute, but not *how* to compute it.

• Given a relational calculus query \( Q \), there may be many ways to represent \( Q \) by an equivalent relational algebra query.

\[ Q \text{ realize/implement } \]

\[ \text{CALCULUS} \quad \text{ALGEBRA} \]
Relational Calculus Syntax

- $R_1, \ldots, R_k$ – relations.
- Each relation $R_i$ has a schema $Sch_i$.
- **Constant Symbols:** For each attribute symbol $A \in \bigcup_{i=1}^{k} Sch_i$, and each member $o \in \text{dom}(A)$, $o$ is a constant symbol.
- **Variable Symbols:** There exists an infinite set of variable symbols $t_1, \ldots, t_r, \ldots$ ranging over tuples.
- **Predicate Symbols:** $\in, =, <, >, \leq, \geq, \neq$ are all predicate symbols.
- **Relational calculus atom:**
  - If $t$ is a tuple variable, and $R_i$ is a relation, then $(t \in R_i)$ is an atom. The (sole) occurrence of $t$ in this atom is said to be *free*. $t$ is said to range over $R_i$.
  - If $t$ is a tuple variable ranging over relation $R_i$ and $A$ is an attribute in $Sch_j$, then $t.A$ is said to be a *tuple-term*. If $t.A$ is a tuple term, and $\tau$ is either a tuple term or a constant symbol, and $\Theta$ is any of the comparison operations $=, <, >, \leq, \geq, \neq$, then $(t.A \Theta \tau)$ and $(\tau \Theta t.A)$ are atoms. $t$ is said to be free in this atom.
- **Example:** $(t \in \text{bank}) \& (t.LNAME = Smith) \& (t.AMOUNT > 5, 200)$.
• Example: \((t_1 \in \text{bank}) \& (t_2 \in \text{emp}) \& (t_1.FNAME = t_2.FNAME) \& (t_1.LNAME = t_2.LNAME) \& (t_1.TRANS = \text{deposit})\).
Relational Calculus Queries

- **Relational Calculus Query:**

  \[ \{ t \mid F \} \]

  where \( F \) is a relational calculus formula in which \( t \) appears free.

- Says: Find all tuples \( t \) such that condition \( F \) is true.

- EX: Find all tuples in the \texttt{bank} relation where the amount field of the tuple exceeds \$5,200.
  \[
  \{ t \mid (t \in \texttt{bank}) \land (t.AMOUNT > 5,200) \}
  \]

- Find all tuples \( t_1 \) such that there exists a tuple \( t_2 \in \texttt{emp} \) such that these two tuples have the same \texttt{FNAME} and \texttt{LNAME} fields and such that tuple \( t_1 \) reflects a deposit of over \$5,200.
  \[
  \{ t_1 \mid (\exists t_2)((t_1 \in \texttt{bank}) \land (t_2 \in \texttt{emp}) \land (t_1.FNAME = t_2.FNAME) \land (t_1.LNAME = t_2.LNAME) \land (t_1.TRANS = \text{deposit}) \land (t_2.AMOUNT > 5,200)) \}
  \]
Linking the Relational Calculus and Relational Algebra

- Relational calculus expresses declarative queries;
- Relational algebra is used to implement these queries.
- Question: Can all relational calculus queries be implemented via equivalent relational algebra queries?
- Answer: No. Only queries that satisfy a property called safety can be implemented.
Suppose $F$ is a relational calculus formula. The *space* of $F$ is defined as follows.

- The space of a relation calculus atom $t \in R$ is the set of all tuples in $R$.
- The space of a relation calculus atom $(t.A_1\Theta t)$ is the set of all tuples over $\text{dom}(A_1) \times \text{dom}(A_n)$ that satisfy this condition. Here, $(A_1, \ldots, A_n)$ is the scheme of the relation $R$ over which tuple $t$ ranges.
- The space of the relational calculus formula $(F_1 \& F_2)$ is the intersection of the spaces of $F_1$ and $F_2$.
- The space of the relational calculus formula $(F_1 \lor F_2)$ is the union of the spaces of $F_1$ and $F_2$.
- The space of the relational calculus formula $\neg F$ involving free variables $t_1, \ldots, t_n$ over relations $R_1, \ldots, R_n$ is the complement of the space of $F$ w.r.t. the appropriate attribute domains of $R_1, \ldots, R_n$.
- The space of the relational calculus formulas $(\forall t \in R)F$ and $(\exists t \in R)F$ coincide with the space of $F$.

A relational calculus query $\{t \mid F\}$ is said to be *safe* if the space of $F$ is finite.
THEOREM: Consider the relational algebra and the relational calculus.

1. If \( A \) is an expression in the relational algebra, then there exists a (safe) relational calculus query \( Q \) such that the set of answers to \( A \) coincides with the set of answers to \( Q \).

2. If \( Q \) is a query in the safe relational calculus, then there exists an algebraic expression \( A \) such that the set of answers to \( A \) coincides with the set of answers to \( Q \).
The most basic SQL query is of the form:

```
SELECT attr_1, attr_2, ..., attr_n
FROM R_1 < V_1 >, R_2 < V_2 >, ..., R_k < V_k >
<WHERE F >.
```

Here,

- $R_1, \ldots, R_k$ are relation names,
- $V_1, \ldots, V_k$ are tuple variables that range over the relations $R_1, \ldots, R_k$, respectively.
- The $V_i$’s are optional (denoted by their being enclosed within pointed brackets).
- The `WHERE` clause is optional.
- The above SQL query corresponds to the relational calculus query:

$$\{ t \mid (V_1 \in R_1) \land \ldots \land (V_n \in R_n) \land F \}$$

where $t$ has the scheme $(attr_1, \ldots, attr_n)$ where each $attr_i$ is associated with one of the relations $R_1, \ldots, R_k$.

- The algebraic version of this query may be expressed as:

$$\pi_{attr_1, \ldots, attr_n} (\sigma_F (R_1 \times \cdots \times R_n))$$
where the relations $R_i$ are (for the sake of simplicity) assumed to have attributes with mutually distinct names and $G$ is obtained from $F$ by replacing all expressions of the form $V_i.attr$ in $F$ are replaced by $attr$. 
SQL Example

- Algebra: $\pi_{\text{COMP}, \text{CITY}}(\text{emp})$
- SQL:

```
SELECT COMP, CITY
FROM emp
```

- Algebra:

$$\pi_{\text{TRANS}, \text{DAY}, \text{MTH}, \text{YR}}(\sigma_{\text{FNAME}=\text{John} \land \text{LNAME}=\text{Smith} \land \text{AMOUNT}>6000}(\text{bank}))$$

- SQL:

```
SELECT TRANS, DAY, MTH, YR
FROM emp
WHERE FNAME = John \land LNAME = Smith \land AMOUNT > 6000.
```

- Algebra:

$$\text{emp} \bowtie_{\text{emp.SSN}=\text{crime.SSN}} \text{crime}$$

- SQL:

```
SELECT FNAME, LNAME
FROM emp E, crime C
WHERE E.SSN = C.SSN.
```
Object Oriented Databases
Object Oriented Databases

Relational model of data has lots of drawbacks. Some of these are:

- Data is organized in the form of relatively “flat” tuples, and there is little scope for tuples with fields that reflect complex data structures.

- The schemes of relations are relatively static, and seamlessly extending them to handle temporary variations in data formats (e.g. through the addition of one or more columns, or the deletion of one or more columns) is not well supported.

- The fact that certain relationships might exist between the content of (part of) one table and (part of) another relational table, must be explicitly encoded through the use of constructs such as integrity constraints.

The basic idea behind the OO paradigms is that data-items being manipulated by an application are “organized” as follows:

- objects being manipulated by the application;

- classes that are collections of objects, either implicitly or explicitly specified

- a hierarchy that imposes an acyclic graph structure on the set of classes.
Example Hierarchy

Suppose $d_1, \ldots, d_k$ is the set of all documents that the police officer has access to.
Another Example Hierarchy

Suppose we consider an application involving museums. Each museum in this case, may be thought of as an object. Museums may then be grouped into art museums, science museums, natural history museums, and so on.

```
  museums
    /   \   \
  Art   Science   Natl.History
     /     \       \       \       \       \       \       \       \       \       \
  Smith sonian Met, NY Louvre MOMA, NY Baltimore Aquarium
```

Art Science Natl.History

Smithsonian Met, Louvre MOMA, Baltimore

NY NY Aquarium

Object Alphabet

An object alphabet consists of:

- A set, called Oid_Set, whose elements are called object-ids;
- A set, called Cid_Set, whose elements are called class-ids;
- A set, Attr_Set, of elements called attributes. Associated with each attribute \( a \in \text{Attr}_{\text{Set}} \) is a set, \( \text{dom}(a) \), called the domain of the attribute.

Example:

- The set of oids may consist of all strings of the form \( \{bi \mid i \text{ is a positive integer} \} \).
- The set of class-id’s may consist of all strings of the form \( \{#i \mid i \text{ is an alphanumeric string} \} \).
- Attributes:
  - author, whose domain is all alphabetic strings;
  - date-created, date-last-modified: whose domain consists of all valid (day,mth,year) triples;
  - admission-fee: whose domain consists of all non-negative reals having at most 2 decimal places.
Values

- Suppose $\Sigma = (\text{Oid Set}, \text{Cid Set}, \text{Attr Set})$ is an object alphabet.
- Suppose $\text{Attr Core} \subseteq \text{Attr Set}$ is some designated set of “core” attributes.

The set of values generated by $\Sigma$ and $\text{Attr Core}$, denoted $\text{Values}(\Sigma, \text{Attr Core})$, is defined inductively as follows:

- Every member of $\text{Oid Set} \cup \bigcup_{A \in A} \text{dom}(A)$ is a value.
- $\text{nil}$ is a special value.
- If $A_1, \ldots, A_n \in \text{Attr Ncore}$, and $c_i \in \text{dom}(A_i)$ for all $1 \leq i \leq n$, then $[a_1 = c_1; \ldots; a_n = c_n]$ is a value.
- If $v_1, \ldots, v_m$ are values, then $<v_1, \ldots, v_m>$ is a value called a tuple value.
- If $v_1, \ldots, v_m$ are values, then $\{v_1, \ldots, v_m\}$ is a value called a set value.
Example (of Values)

Let us return to our Police example. Here, our object alphabet $\Sigma$ may be given by:

- **Oid_Set** = \{b1, b2, b3, b4, b5, b6\}.
- **Cid_Set** = \{html, image, rawtext, word_{mac}, word_{pc}\}.
- **Attr_Set** may be given by:
  - **Attr_Core**: \{real, bool, int, string\}. The domain of these attributes is given in the obvious way.
  - **Attr_Ncore**: \{author, date – created, date – last – modified related – docs\}. The attribute, related-docs, may have as its domain, the set $2^{\text{Oid_Set}}$, i.e. the power set of the space of object-ids.

The space of values, $\text{Values}(\Sigma)$ associated with this example, includes:

- [author = John Smith];
- [author = John Smith; date – created = (15, Jan, 1996)];
- \{[author = John Smith], author = John Smith; date – created = (15, Jan, 1996)]\}.
- < [author = John Smith], {[author = Lisa Adams], author = John Smith; date – created = (15, Jan, 1996)] >.
• [author = Lisa Adams, related – doc = \{b2, b3\}].
**Object**

**Definition:** An object $o$ w.r.t. an object alphabet $\Sigma$ and a core set $Attr\_Core$ of attributes, is a pair $(id_o, val_o)$ where $id_o \in Oid\_Set$ is called the *identity* of the object, and $val_o$ is called the (property) value of $o$.

This definition is nice because:

- It says that an object has an associated identity (which tells us how to refer to the object) and
- An object has an associated set of properties (with corresponding values).

**EXAMPLE:**

- Take an whose id is $b59878$ (corresponding to a person like John Smith)
- Take another object whose id is $b818932$ corresponding to Jill Smith
- They may have different properties – e.g. Jill Smith may have a property called *gynecologist*, with value Donna Manson, but John Smith doesn’t have such a property.
Declaring Objects

declare oid
values value.

EXAMPLE:

declare b2
values [author = John Smith,
        url = http://www.somewhere.com/index.html
        date – created = (15,Jan, 1996)
        date – last – modified = (19,Nov,1996)].

Says that a document, with object id b2 exists, and that this
document has four attributes, viz. author, url, date-created,
and date-last-modified, with the values specified above.

EXAMPLE:

declare b3
values [author = John Smith,
        url = http://www.somewhereelse.com/index.html ,
        {[link = b1], [link = b2]}].

This HTML document that the police investigator in our sam-
ple multimedia scenario might be investigating, is different from
the preceding one in a few significant ways. First, it contains
links to two objects, viz. \( b_2 \) and \( b_1 \), but it also contains a set of
link properties. Furthermore, the attributes, \texttt{date-created} and
\texttt{date-last-modified}, that were associated with the (seemingly
similar) document \( b_2 \) are not associated with this document.
Types and Classes

- Each type Attr_Core is a type. (Assume Attr_Core contains the “standard” data types).
- Each member of Cid_Set is a type.
- **Record types** that have fields, each with an associated type. \([f_1 : \tau_1; \ldots; f_n : \tau_n]\) denotes a record type having fields \(f_1, \ldots, f_n\) of type \(\tau_1, \ldots, \tau_n\), respectively.
- **Set types** that are of the form \(\{\tau\}\) which denotes a set of data items, each of type \(\tau\).
- **List types** that are of the form \(<\tau>\) which denotes a list of data items, each of type \(\tau\).

**EXAMPLE:**

```plaintext
[  author: string;
  url: urltype;
  date-created: datatype;
  date-last-modified: datatype ]
```

**EXAMPLE:**
address: string;
director: string;
special-exhibits: { string };
affiliated-museums: { Oid_Set } ]
Class Hierarchy

**DEF:** A class hierarchy is a triple, $(G, \leq, \text{types})$ where:

- $G$ is a set of objects and classes;
- $\leq$ is a partial ordering on $G$;
- $\text{types}$ is a map that associates a type with each $g \in G$ such that the following condition holds:

  $$(\forall g_1, g_2 \in G) g_1 \leq g_2 \rightarrow \text{types}(g_1) \text{ subtype } \text{types}(g_2).$$

**DEF:** The **subtype** relation is defined as follows:

- $[f_1 : \tau_1, \ldots, f_{n+k} : \tau_{n+k}]$ is a subtype of $[f_1 : \tau_1, \ldots, f_n : \tau_n]$.
- If $\tau_1$ is a subtype of $\tau_2$, then $\{\tau_1\}$ is a subtype of $\{\tau_2\}$.
- If $\tau_1$ is a subtype of $\tau_2$, then $< \tau_1 >$ is a subtype of $< \tau_2 >$.

The above relation is closed under reflexivity and transitivity.
Example Types

Type of museum might be:

[ address: string
director: string
departments: { string }
budget: real ]

Type of art (museum) might be:

[ address: string
director: string
departments: { string }
budget: real
old-master-collection: { string };
modern-art-collection: { string };
lithograph-collection: { string };
cartographic-collection: { string }]

Note that the type of art is a subtype of the type associated with museum.
Methods

**DEF:** *Methods* are programs associated with any object/class $g \in G$ that manipulate the structures declared in $g$’s type definition.

- Each class $g$ has an associated set of programs, $\text{methods}(g)$, that apply to that class.
- Provides *encapsulation*.
- Objects in a class can be manipulated only by methods applicable to that class, no others.
- Methods provide an elegant interface via which other classes/objects and/or other third party programs, may access an object or a class.

**EXAMPLE:**

- The class *documents* may have a method called $\text{FindDocs}_d$ which takes as input, any name (type *string*) and returns as output, a set of documents.
- The class *image* may have a method called $\text{FindDocs}$ (referred to by us as $\text{FindDocs}_i$) of which takes as input, any name (type *string*) and returns as output, a set of images.
• The class \texttt{Word} may have a method called \texttt{FindDocs} (referred to by us as \texttt{FindDocs}_w) which takes as input, any name (type \texttt{string}) and returns as output, a set of Word documents.

Suppose the function \texttt{FindDocs} is invoked by the class \texttt{image}. Then the code associated with the function \texttt{FindDocs}_i is executed, as there is a version of \texttt{FindDocs} that is \textit{directly associated} with this class.

On the other hand, consider invoking the method \texttt{FindDocs} in the class \texttt{PC\_Word}. This class has no method called \texttt{FindDocs} \textit{explicitly} associated with it. Thus, we must examine if one of its ancestors has \texttt{FindDocs} associated with it. The answer is yes, both the class \texttt{Word}, and the class \texttt{Documents}, have \texttt{FindDocs} associated with it. However, the class \texttt{Word} is \textit{more specific} and applies “better” (informally speaking) to the case of PC-Word documents, so we apply the function \texttt{FindDocs}_w to objects in the class \texttt{PC\_Word}. 
Object Definition Language (ODL)

- Provide a simple language within which both objects and object interfaces can be defined.
- External programs wishing to access/manipulate the object must do so using the methods provided by the object.
- To do so, however, they must have access to the signatures (I/O types) of those objects.
- ODL provides a formal syntax for this purpose
ODL Example

\begin{verbatim}
interface html:documents
  ( extent html_documents
    keys url:persistent
      {<properties>
       <operations>}
  );
\end{verbatim}

- Line (1) above states that the type, \texttt{html} is a subtype of the bigger type, \texttt{documents}.
- Line (2) above states that the type \texttt{html} applies to all HTML documents, i.e. it specifies that this type definition applies to all HTML documents in the object oriented database.
- Line (3) specifies that the type HTML has a key attribute, called \texttt{url}, which is persistent. This means that memory allocated to such persistent objects does not cause the object to “disappear” when the process that creates it terminates.
- In addition, the ODL definition of the object \texttt{html} contains two additional features, viz. a list of \texttt{properties} and a list of \texttt{relationships}.
- These may be “filled in” as shown on the next slide.
ODL Example

\texttt{interface html:documents} \hspace{1cm} (1) \\
\texttt{( extent html_documents} \hspace{1cm} (2) \\
\texttt{  keys url:persistent} \hspace{1cm} (3) \\
\texttt{    \{} \hspace{1cm} (4) \\
\texttt{       attribute string author;} \hspace{1cm} (4) \\
\texttt{       attribute date date\_created;} \hspace{1cm} (5) \\
\texttt{       attribute date date\_last\_modified;} \hspace{1cm} (6) \\
\texttt{    \}} \hspace{1cm} (7) \\
\texttt{relationship Set<Persons> author inverse Persons:written\_work;} \hspace{1cm} (7) \\
\texttt{relationship Set<Persons> reader inverse Persons:works\_read;} \hspace{1cm} (8) \\
\texttt{)}
Object Query Language (OQL)

- Introduced by the ODMG group.
- If \( Q \) is a query expressed in SQL, then \( Q \) is also a valid OQL query.
- SQL queries can only access “flat” relational tables.
- In contrast, objects may have a nested structure, as well as include fields that contain the so-called collection types – sets, lists, and bags. OQL provides facilities to access such data types as well.

**EXAMPLE**

```
SELECT struct(field1:x.url, field2:x.link)
FROM Word x
WHERE x.author=“John Smith”.
```

- This query first finds all objects \( x \) that are \texttt{Word} documents, and then identifies those \texttt{Word} documents that satisfy the conditions \( x\.author=“John Smith” \).
- For each such \( x \), it returns a \texttt{structure} containing two fields: the \texttt{url} field of \( x \), and the \texttt{link} field of \( x \). Note that the link field of \( x \) may be a linked list. Thus, an answer to this query may look like:
|----------------------------|-------------------------------
| www.somewhere.com/file2.html | www.abc.com/file1.html,         
| ...                         | www.somewhere.com/file1.html  

More OQL Examples

```
SELECT       y.author
FROM
(SELECT    x.link
FROM      Word x
WHERE    x.author="John Smith") y.
```

- Query is complicated for two reasons: first, it contains a nested query, and second, it contains this new variable \( y \).

- This query says: *first find all Word documents authored by John Smith and let \( y \) refer to any such object* — this is accomplished by the inner `SELECT` statement.

- Then return the `y.author` field for all such documents.
Object-Relational Systems

- Relational databases have proved very useful in querying “flat” data.

- Is it possible to extend the relational model of data to handle complex data, rather than flat data?

- The answer is yes, and the resulting paradigm is typically called an object-relational database.
Example Object-Relational Systems

- Consider a bank relation with schema
  
  \[ (\text{FNAME}, \text{LNAME}, \text{ACCTYPE}, \text{TRANS}, \text{AMOUNT}, \text{DAY}, \text{MTH}, \text{YR}) \]
  
  and a \textit{crime} relation with schema
  
  \[ (\text{SSN}, \text{FIRST}, \text{LAST}, \text{CONVICTION}, \text{DAY}, \text{MTH}, \text{YR}). \]

- Suppose we wished to extend both these schemes to also include an image.

- Adequate to merely extend the schemes of the two relations involved to include a \textit{PIC} field as follows.

  \[ (\text{FNAME}, \text{LNAME}, \text{ACCTYPE}, \text{TRANS}, \text{AMOUNT}, \text{DAY}, \text{MTH}, \text{YR}, \text{PIC}). \]

  \[ (\text{SSN}, \text{FIRST}, \text{LAST}, \text{CONVICTION}, \text{DAY}, \text{MTH}, \text{YR}, \text{PIC}). \]

- Suppose that we have a tuple in the \textit{bank} relation of the form:

  \[
  \begin{array}{cccccccc}
  \text{FNAME} & \text{LNAME} & \text{ACCTYPE} & \text{TRANS} & \text{AMOUNT} & \text{DAY} & \text{MTH} & \text{YR} & \text{PIC} \\
  \text{Jill} & \text{Davis} & \text{savings} & \text{withdrawal} & 1400 & 1 & \text{jan} & 1993 & \text{image1} \\
  \end{array}
  \]

- Suppose furthermore, that Jill Davis goes to the bank and reports that her bankcard was stolen from her, and that she did not make this withdrawal.

- In this case, an obvious check to run is to examine the surveillance image, stored in the file \textit{image1}, to see if it is in fact Jill Davis or not. Suppose Ms. Davis’ assertion is correct and it is apparent that the person depicted in \textit{image1} is not her.
• next logical step for the police to perform is to attempt to match the contents of image1 against the images contents of the crime database.

• This requires execution of a query that says: Select all tuples in the crime relation that “match” (using face recognition techniques) the image depicted in image1.

• Relational database query languages such as SQL cannot support this query directly because the selection condition requires a comparison operator (match) that is not typically supported by a relational database.

• An object database query language cannot be used either because the data is not stored in an object oriented system.
Object-Relational Schemes

- Suppose we have a set $O$ of objects, each with associated properties and methods.

- An object relational scheme is of the form

$$ (A_1 : T_1, \ldots, A_n : T_n) $$

where the $A_i$'s are attribute names, and the $T_i$'s are objects.

- The scheme of the extended crime relation can be written as:

$$ (\text{FNAME} : \text{str}, \text{LNAME} : \text{str}, \text{ACCTYPE} : \text{int}, \text{TRANS} : \text{tt}, \text{AMOUNT} : \text{real}, \text{DAY} : \text{dom}, \text{MTH} : \text{mths}, \text{YR} : \text{int}, \text{PIC} : \text{image}). $$

$$ (\text{SSN} : \text{ssntyp}, \text{FIRST} : \text{str}, \text{LAST} : \text{str}, \text{CONVICTIO\text{N}} : \text{str}, \text{DAY} : \text{dom}, \text{MTH} : \text{mths}, \text{YR} : \text{int}, \text{PIC} : \text{image}). $$

- Examples of object conditions:

  - $\text{match(image}_1, \text{image}_2) > 0.7$;

  - $\text{match(image}_1, \text{image}_2) > \text{match(image}_3, \text{image}_2)$;

  - $\text{match(image}_1, \text{image}_2) < \text{t.SAL}$. (Though this condition may not make much sense from an intuitive point of view, it is a syntactically valid condition if $\text{t.SAL}$ is a real valued field).

- We may now extend SQL to access the functions associated with objects and express our desired query as follows:
SELECT FNAME, LNAME
FROM crime C, bank B
WHERE match(image1,B.PIC) > 0.9.

- This query says “Select the FNAME and LNAME fields of all tuples t in the crime relation such that the match function defined in the object image returns a value of over 90% when it matches image1 and the PIC field of t.”

- In general, if \( m(arg1, \ldots, argn) \) is a valid method invocation, and \( x \) is either another method invocation or a value of the same type as the output type of method \( m \), then \( m(arg1, \ldots, argn) = x \) is an object condition. If \( x \) is a numeric value and the above condition holds, then \( m(arg1, \ldots, argn) > x \), \( m(arg1, \ldots, argn) \geq x \), \( m(arg1, \ldots, argn) \leq x \), are also object conditions.
Multidimensional Data Structures
Multidimensional Data Structures

- An important source of media data is geographic data.

- A geographic information system (GIS) stores information about some physical region of the world.

- A map is just viewed as a 2-dimensional image, and certain “points” on the map are considered to be of interest.

- These points are then stored in one of many specialized data structures.
  - $k$-d Trees
  - Point Quadtrees
  - MX-Quadtrees

- Alternatively, we may wish to store certain rectangular regions of the map.

- We will study one data structure – the R-tree – that is used to store such rectangular data.
Example Maps

(a) Map with Marked Points

(b) Map with Marked Regions
$k$-D Trees

- Used to store $k$ dimensional point data.
- It is not used to store region data.
- A 2-d tree (i.e. for $k = 2$) stores 2-dimensional point data while a 3-d tree stores 3-dimensional point data, and so on.
Node Structure

```pascal
nodetype = record
    INFO: infotype;
    XVAL: real;
    YVAL: real;
    LLINK: ↑nodetype
    RLINK: ↑nodetype
end
```

<table>
<thead>
<tr>
<th>INFO</th>
<th>XVAL</th>
<th>YVAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>LLINK</td>
<td>RLINK</td>
<td></td>
</tr>
</tbody>
</table>

- INFO field is any user-defined type whatsoever.
- XVAL and YVAL denote the coordinates of a point associated with the node.
- LLINK and RLINK fields point to two children.
2-d trees, formally

Level of nodes is defined in the usual way (with root at level 0).

Def: A 2-d tree is any binary tree satisfying the following condition:

1. If $N$ is a node in the tree such that $\text{level}(N)$ is even, then every node $M$ in the subtree rooted at $N.LLINK$ has the property that $M.XVAL < N.XVAL$ and every node $P$ in the subtree rooted at $N.RLINK$ has the property that $P.XVAL \geq N.XVAL$.

2. If $N$ is a node in the tree such that $\text{level}(N)$ is odd, then every node $M$ in the subtree rooted at $N.LLINK$ has the property that $M.YVAL < N.YVAL$ and every node $P$ in the subtree rooted at $N.RLINK$ has the property that $P.YVAL \geq N.YVAL$. 
Example 2-d Trees
Insertion/Search in 2-d Trees

To insert a node $N$ into the tree pointed to by $T$, do as follows:

- Check to see if $N$ and $T$ agree on their XVAL and YVAL fields.
- If so, just overwrite node $T$ and we are done.
- Else, branch left if $N.XVAL < T.XVAL$ and branch right otherwise.
- Suppose $P$ denotes the child we are examining. If $N$ and $P$ agree on their XVAL and YVAL fields, just overwrite node $P$ and we are done, else branch left if $N.YVAL < P.YVAL$ and branch right otherwise.
- Repeat this procedure, branching on XVAL’s when we are at even levels in the tree, and on YVALs when we are at odd levels in the tree.
Example of Insertion

Suppose we wish to insert the following points.

<table>
<thead>
<tr>
<th>City</th>
<th>(XVAL,YVAL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Banja Luka</td>
<td>(19,45)</td>
</tr>
<tr>
<td>Derventa</td>
<td>(40,50)</td>
</tr>
<tr>
<td>Toslic</td>
<td>(38,38)</td>
</tr>
<tr>
<td>Tuzla</td>
<td>(54,35)</td>
</tr>
<tr>
<td>Sinj</td>
<td>(4,4)</td>
</tr>
</tbody>
</table>
Example of Insertion

(a) Splitting of region by Banja Luka
(b) Splitting of region by Derventa
(c) Splitting of region by Toslic
(d) Splitting of region by Sinj
Deletion in 2-d Trees

Suppose $T$ is a 2-d tree, and $(x, y)$ refers to a point that we wish to delete from the tree.

- Search for the node $N$ in $T$ that has $N.XVAL = x$ and $N.YVAL = y$.
- If $N$ is a leaf node, then set the appropriate field (LLINK or RLINK) of $N$’s parent to NIL and return $N$ to available storage.
- Otherwise, either the subtree rooted at $N.LLINK$ (which we will denote by $T_\ell$) or the subtree rooted at $N.RLINK$ (which we will denote by $T_r$) is non-empty.

(Step 1) Find a “candidate replacement” node $R$ that occurs either in $T_i$ for $i \in \{\ell, r\}$.

(Step 2) Replace all of $N$’s non-link fields by those of $R$.

(Step 3) Recursively delete $R$ from $T_i$.

- The above recursion is guaranteed to terminate as $T_i$ for $i \in \{\ell, r\}$ has strictly smaller height than the original tree $T$. 
Finding Candidate Replacement Nodes for Deletion

- The desired replacement node $R$ must bear the same spatial relation to all nodes $P$ in both $T_L$ and $T_R$ that $N$ bore to $P$
- I.e. if $P$ is to the southwest of $N$, then $P$ must be to the southwest of $R$, if $P$ is to the northwest of $N$, then $P$ must be to the northwest of $R$, and so on.

This means that the desired replacement node $R$ must satisfy the property that:

1. Every node $M$ in $T_L$ is such that: $M.XVAL < R.XVAL$ if $\text{level}(N)$ is even and $M.YVAL < R.YVAL$ if $\text{level}(N)$ is odd.

2. Every node $M$ in $T_R$ is such that: $M.XVAL \geq R.XVAL$ if $\text{level}(N)$ is even and $M.YVAL \geq R.YVAL$ if $\text{level}(N)$ is odd.

- If $T_R$ is not empty, and $\text{level}(N)$ is even, then any node in $T_R$ that has the smallest possible $XVAL$ field in $T_R$ is a candidate replacement node.
- But if $T_R$ is empty, then we might not be able to find a candidate replacement node from $T_L$ (why?).
• In this case, find the node $R'$ in $T_\ell$ with the smallest possible XVAL field. Replace $N$ with this.

• Set $N.RLINK = N.LLINK$ and set $N.LLINK = NIL$.

• Recursively delete $R'$. 
Range Queries in 2-d Trees

- A range query with respect to a 2-d tree $T$ is a query that specifies a point $(x_c, y_c)$, and a distance $r$.

- The answer to such a query is the set of all points $(x, y)$ in the tree $T$ such that $(x, y)$ lies within distance $d$ of $(x_c, y_c)$.

- I.e. A range query defines a circle of radius $r$ centered at location $(x_c, y_c)$, and expects to find all points in the 2-d tree that lie within the circle.

- Recall that each node $N$ in a 2-d tree implicitly represents a region $R_N$.

- If the circle specified in a query has no intersection with $R_N$, then there is no point searching the subtree rooted at node $N$. 
Example Range Query
Point Quadtrees

- Point quadtrees always split regions into four parts.
- In a 2-d tree, node $N$ splits a region into two by drawing one line through the point ($N.XVAL$, $N.YVAL$).
- In a point quadtree, node $N$ splits the region it represents by drawing both horizontal and a vertical line through the point ($N.XVAL$, $N.YVAL$).
- These four parts are called the NW (northwest), SW (southwest), NE (northeast) and SE (southeast) quadrants determined by node $N$.
- Each of these quadrants corresponds to a child of node $N$. Thus, quadtree nodes may have up to 4 children each.

Node structure in a point quadtree:

```plaintext
qtnodetype = record
  INFO: infotype;
  XVAL: real;
  YVAL: real;
  NW,SW,NE,SE: ↑qtnodetype
end
```
Nodes in Point Quadtrees Implicitly Represent Regions
### Insertion into Point Quadtrees

<table>
<thead>
<tr>
<th>City</th>
<th>(XVAL, YVAL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Banja Luka</td>
<td>(19, 45)</td>
</tr>
<tr>
<td>Derventa</td>
<td>(40, 50)</td>
</tr>
<tr>
<td>Toslic</td>
<td>(38, 38)</td>
</tr>
<tr>
<td>Tuzla</td>
<td>(54, 35)</td>
</tr>
<tr>
<td>Sinj</td>
<td>(4, 4)</td>
</tr>
</tbody>
</table>
Insertion into Point Quadtrees
(a) Splitting of region by Banja Luka

(b) Splitting of region by Derventa

(c) Splitting of region by Toslic

(d) Splitting of region by Tuzla
Insertion into Point Quadtrees

(a) BanjaLuka (19,45)
(b) BanjaLuka (19,45)
(c) BanjaLuka (19,45)

(d) Tuzla (54,35)
(e) Tuzla (54,35)
Deletion in Point Quadtrees

- If the node being deleted is a leaf node, deletion is completely trivial: we just set the appropriate link field of node N’s parent to NIL and return the node to available storage.

- As in the case of deletion in 2-d trees, we need to find an appropriate replacement node for non-leaf nodes being deleted.

- Is this easy?

- No. Why? Return to Previous slide.
Expanded Node Type

- Expand the node structure `qtnodetype` to a new node structure `newqtnodetype`
- `qtnodetype = record`
  INFO: `infotype`;
  XVAL,YVAL: `real`;
  XLB,YLB,XUB,YUB: `real` $\cup \{-\infty, +\infty\}$
  NW,SW,NE,SE: $\uparrow$qtnodetype
end
- When inserting a node $N$ into the tree $T$, we need to ensure that:
  - If $N$ is the root of tree $T$, then $N.XLB = -\infty$, $N.YLB = -\infty$, $N.XUB = +\infty$, $N.YUB = +\infty$.
  - If $P$ is the parent of $N$ then the following table describes what $N$’s XLB, YLB, XUB, YUB fields should be, depending upon whether $N$ is the NW, SW, NE, SE child of $P$. We use the notation $w = (P.XUB - P.XLB)$ and $h = (P.YUB - Y.YLB)$.

<table>
<thead>
<tr>
<th>Case</th>
<th>N.XLB</th>
<th>N.XUB</th>
<th>N.YLB</th>
<th>N.YUB</th>
</tr>
</thead>
<tbody>
<tr>
<td>N=P,NW</td>
<td>P.XLB</td>
<td>P.XLB + $w \times 0.5$</td>
<td>P.YLB + $h \times 0.5$</td>
<td>P.YUB</td>
</tr>
<tr>
<td>N=P,SW</td>
<td>P.XLB</td>
<td>P.XLB + $w \times 0.5$</td>
<td>P.YLB</td>
<td>P.YLB + $h \times 0.5$</td>
</tr>
<tr>
<td>N=P,NE</td>
<td>P.XLB + $w \times 0.5$</td>
<td>P.XUB</td>
<td>P.YLB + $h \times 0.5$</td>
<td>P.YUB</td>
</tr>
<tr>
<td>N=P,SE</td>
<td>P.XLB + $w \times 0.5$</td>
<td>P.XUB</td>
<td>P.YLB</td>
<td>P.YLB + $h \times 0.5$</td>
</tr>
</tbody>
</table>
Deletion in Point Quadtrees, Continued

– When deleting an interior node $N$, we must find a replacement node $R$ in one of the subtrees of $N$ (i.e. in one of $N.NW,N.SW,N.NE,N.SE$) such that:

* every other node $R_1$ in $N.NW$ is to the north west of $R$,
* every other node $R_2$ in $N.SW$ is to the south west of $R$,
* every other node $R_3$ in $N.NE$ is to the north east of $R$ and
* every other node $R_4$ in $N.SE$ is to the south east of $R$.

– Consider the figure on the next page.

– Suppose we wish to delete Banja Luka from this quadtree. In this case, one such replacement node can in fact be found, viz. Toslic.

– However, in general, it may not always be possible to find such a replacement node. See the figure in the page after next.
Deletion of Banja Luka
(a) Splitting of region by Banja Luka  
(b) Splitting of region by Dervent

(c) Splitting of region by Toslic  
(d) Splitting of region by Tuzla
Impossibility of finding Replacement Candidates

Thus, in general, deletion of an interior node \( N \) may require reinsertion of all nodes in the subtrees pointed to by \( N.NE, N.SE, N.NW \) and \( N.SW \). In the worst case, this may require almost all nodes to be reinserted.
Range Searches in Point Quadtrees

- Each node in a point quadtree represents a region.
- Do not search regions that do not intersect the circle defined by the query.

proc RangeQueryPointQuadtree(T:newqtnodetype, C:circle);

1. If $\text{region}(T) \cap C = \emptyset$ then Halt
2. else
   (a) If $(T.XVAL, T.YVAL) \in C$ then print $(T.XVAL, T.YVAL)$;
   (b) RangeQueryPointQuadtree(T.NW, C);
   (c) RangeQueryPointQuadtree(T.SW, C);
   (d) RangeQueryPointQuadtree(T.NE, C);
   (e) RangeQueryPointQuadtree(T.SE, C);
end proc
The MX-Quadtree

- For both 2-d trees as well as point quadtrees, the “shape” of the tree depends upon the order in which objects are inserted into the tree.

- In addition, both 2-d trees and point quadtrees split regions into 2 (for 2-trees) or 4 (for point quadtrees) sub-regions – however, the split may be uneven depending upon exactly where the point \((N.XVAL, N.YVAL)\) is located inside the region represented by node \(N\).

- MX-quadtrees attempt to: ensure that the shape (and height) of the tree are independent of the number of nodes present in the tree, as well as the order of insertion of these nodes.

- MX-quadtrees also attempt to provide efficient deletion and search algorithms.
The MX-Quadtree

- Assume that the map being represented is “split up” into a grid of size \( (2^k \times 2^k) \) for some \( k \).
- The application developer is free to choose \( k \) as s/he likes to reflect the desired granularity, but once s/he chooses \( k \), s/he is required to keep it fixed.

Example:
The MX-Quadtree

- **Node Structure**: Exactly the same as for point quadtrees, except that the root of an MX-quadtree represents the region specified by $XLB = 0$, $XUB = 2^k$, $YLB = 0$, $YUB = 2^k$.

- When a region gets “split”, it gets split down the middle. Thus, if $N$ is a node, then the regions represented by the four children of $N$ are described by the following table.

<table>
<thead>
<tr>
<th>Child</th>
<th>$XLB$</th>
<th>$XUB$</th>
<th>$YLB$</th>
<th>$YUB$</th>
</tr>
</thead>
<tbody>
<tr>
<td>NW</td>
<td>N.$XLB$</td>
<td>N.$XLB + \frac{w}{2}$</td>
<td>N.$YLB + \frac{w}{2}$</td>
<td>N.$YLB + w$</td>
</tr>
<tr>
<td>SW</td>
<td>N.$XLB$</td>
<td>N.$XLB + \frac{w}{2}$</td>
<td>N.$YLB$</td>
<td>N.$YLB + \frac{w}{2}$</td>
</tr>
<tr>
<td>NE</td>
<td>N.$XLB + \frac{w}{2}$</td>
<td>N.$XLB + w$</td>
<td>N.$YLB + \frac{w}{2}$</td>
<td>N.$YLB + w$</td>
</tr>
<tr>
<td>SE</td>
<td>N.$XLB + \frac{w}{2}$</td>
<td>N.$XLB + w$</td>
<td>N.$YLB$</td>
<td>N.$YLB + \frac{w}{2}$</td>
</tr>
</tbody>
</table>

Here, $w$ denotes the width of the region represented by $N$. 
Insertion in MX-Quadtrees

(a) (b)
Insertion in MX-Quadtrees

After insertion of A

After insertion of C

After Insertion of B

After Insertion of D
Deletion in MX-Quadtrees

- Deletion in an MX-quadtree is a fairly simple operation, because all points are represented at the leaf level.
- If \( N \) is an interior (i.e. non-leaf) node in an MX-quadtree whose root is pointed to by \( T \), then the region implicitly represented by node \( N \) contains at least one point that is explicitly contained in the tree.
- If we wish to delete a point \((x, y)\) from tree \( T \), we try to preserve this property.
- This can be done as follows.
  * First, we set the appropriate link of \( N \)'s parent to NIL.
  * We then check if all the four link fields of \( M \) are NIL.
  * If so, we examine \( M \)'s parent (let us call it \( P \) for now). As \( M \) is \( P \)'s child, we find a link field \( \text{dir1} \) such that \( P.\text{dir1} = M \). We then set \( P.\text{dir1} = \text{NIL} \) and then (as before) check to see if \( P \)'s four link fields are all NIL.
  * If so, we continue this process.
- Total time required for deletion is \( O(k) \).
Range Queries in MX-Quadtrees

Handled in exactly the same way as for point quadtrees. But there are two differences:

- The *content* of the XLB, XUB, YLB, YUB fields is different from that in the case of point quadtrees.
- As points are stored at the leaf level, checking to see if a point is in the circle defined by the range query needs to be performed only at the leaf level.
R-Trees

- Used to store *rectangular regions* of an image or a map such as those shown below.
- R-trees are particularly useful in storing very large amounts of data on disk.
- They provide a convenient way of minimizing the number of disk accesses.
R-Trees

- Each R-tree has an associated order, which is an integer $K$.
- Each nonleaf R-tree node contains a set of at most $K$ rectangles and at least $\lfloor K/2 \rfloor$ rectangles (with the possible exception of the root).
- Intuitively, this says that each nonleaf node in the R-tree, with the exception of the root, must be at least “half” full.
- This feature makes R-trees appropriate for disk based retrieval because each disk access brings back a page containing several (i.e. at least $\frac{K}{2}$ rectangles).

R-trees manipulate two kinds of rectangles:

- “Real” rectangles (such as those shown in the map on the previous slide) or
- “Group” rectangles such as those shown below.
Example R-Tree

This is an R-tree of order 4, associated with the rectangles shown earlier.

R-tree nodes have the following structure:

\[
\text{rtnodetype} = \text{record} \\
Rec_1, \ldots, Rec_K: \text{rectangle} ; \\
P_1, \ldots, P_K: \uparrow\text{rtnodetype} \\
\text{end}
\]
Insertion into an R-Tree
Insertion into an R-Tree
An Incorrect Insertion into an R-Tree
Deletion in R-Trees

- Deletion of objects from R-trees may cause a node in the R-tree to “underflow” because an R-tree of order \( K \) must contain at least \( \lceil K/2 \rceil \) rectangles (real or group) in it.

- When we delete a rectangle from an R-tree, we must ensure that that node is not “underfull.”

**Example:** Delete R9 from the following R-tree.
Deletion in R-Trees

- If we delete R9, then the node containing rectangle R9 would have only one node in it.
- In this case, we must create a new logical grouping.
- One possibility is to reallocate the groups as follows:

<table>
<thead>
<tr>
<th>Group</th>
<th>Rectangles</th>
</tr>
</thead>
<tbody>
<tr>
<td>G1</td>
<td>R1, R2, R3</td>
</tr>
<tr>
<td>G2</td>
<td>R4, R6, R7</td>
</tr>
<tr>
<td>G3</td>
<td>R5, R8</td>
</tr>
</tbody>
</table>

- The new R-tree is:

![Diagram of R-tree](image-url)
Image Databases
Image Databases

- In standard relational databases, the user types in a query, and obtains an answer in response.

- In image databases, things are different: for example, a police investigator may have in front of him/her, a surveillance photograph of someone, whose identity s/he may not know, but wishes to determine. Thus, s/he may wish to ask a query of the form: *Here’s a picture of a person. Can you retrieve all pictures from the image database that are “similar” to this person and tell me the identities of the people in the pictures you return to me?*

- This query is fundamentally different from ordinary queries for two reasons:

  1. First, the query includes a picture as part of the query.

  2. Second, the query asks about “similar” pictures and hence, uses a notion of “imprecise match” whose definition needs to be precisely articulated (it is possible to reason precisely about imprecise data!)

---

_V.S. Subrahmanian — All Rights Reserved_

These slides may not be duplicated without explicit written permission from Morgan Kaufmann Press,

Principles of Multimedia Database Systems  Morgan Kaufmann  Copyright ©1997
Example Face Database
Raw Images

The content of an image consists of all “interesting” objects in that image. Each object is characterized by:

- a *shape descriptor* that describes the shape/location of the region within which the object is located inside a given image.

- a *property descriptor* that describes the properties of the individual pixels (or groups of pixels) in the given image. Examples of such properties include: red-green-blue (RGB) values of the pixel (or aggregated over a group of pixels), grayscale levels in the case of black and white images, etc. In general, it will be infeasible to associate properties with individual pixels, and hence, cells (rectangular “groups” of pixels) will be used most of the time.

- We assume the existence of a set $\text{Prop}$ of properties. A property consists of two components –
  1. A *property name* – e.g. “Red”, “Green”, “Blue” and
  2. a *property domain* which specifies the range of values that the property can assume – e.g. $\{0, \ldots, 8\}$. 

---

V.S. Subrahmanian — All Rights Reserved

These slides may not be duplicated without explicit written permission from Morgan Kaufmann Press.

Principles of Multimedia Database Systems    Morgan Kaufmann    Copyright ©1997
Example: Consider the image file *pic1.gif* on the preceding slide. This image has two objects of interest - let call these two objects $o_1$ and $o_2$.

- The shapes of these objects are captured by rectangles shown. Formally, object $o_1$'s shape may be specified by:
  
  $\text{rectangle : } XLB = 10; XUB = 60, YLB = 5; YUB = 50.$

- The *property descriptor* associated with an individual cell (group of pixels) may look like this:
  
  1. Red = 5;
  2. Green = 1;
  3. Blue = 3.

Other properties like texture may also be included.
Definitions

- Every image $I$ has an associated pair of positive integers $(m, n)$, called the grid-resolution of the image. This divides the image into $(m \times n)$ cells of equal size, called the image grid.

- Each cell in a given gridded $(m \times n)$ image $I$ consists of a collection of pixels.

- A cell property is a triple $(\text{Name}, \text{Values}, \text{Method})$ where $\text{Name}$ is a string denoting the property’s name, $\text{Values}$ is a set of values that that property may assume, and $\text{Method}$ is an algorithm that tells us how to compute the property involved.

Example: Consider black and white images.

$$(\text{bwcolor}, \{\text{b, w}\}, \text{bwalgo})$$

could be a cell property with property name $\text{bwcolor}$ and the possible values are $\text{b}$ (black) and $\text{w}$ (white), respectively. $\text{bwalgo}$ then is an algorithm which may take a cell as input, and return as output, either black or white, by somehow combining the black/white levels of the pixels in the cell.

Example: Consider gray scale images (with 0 =white and 1 =black), we may have the cell property

$$(\text{graylevel}, [0, 1], \text{grayalgo})$$
where our property name is named \texttt{graylevel}, and its possible values are real numbers in the $[0, 1]$ interval, and the associated method \texttt{grayalgo} takes as input, a cell, and computes its gray level.
Image Definitions

- An object shape is any set $P$ of points such that if $p, q \in P$, then there exists a sequence of points $p_1, \ldots, p_n$ all in $P$ such that:

  1. $p = p_1$ and $q = p_n$ and
  2. for all $1 \leq i < n$, $p_{i+1}$ is a neighbor of $p_i$, i.e. if $p_i = (x_i, y_i)$ and $p_{i+1} = (x_{i+1}, y_{i+1})$, then $(x_{i+1}, y_{i+1})$ satisfies one of the following conditions:

$$
(x_{i+1}, y_{i+1}) = (x_i + 1, y_i) \quad (x_{i+1}, y_{i+1}) = (x_i - 1, y_i)
$$
$$
(x_{i+1}, y_{i+1}) = (x_i, y_i + 1) \quad (x_{i+1}, y_{i+1}) = (x_i, y_i - 1)
$$
$$
(x_{i+1}, y_{i+1}) = (x_i - 1, y_i + 1) \quad (x_{i+1}, y_{i+1}) = (x_i + 1, y_i - 1)
$$

- A rectangle is an object shape, $P$, such that there exist integers $XLB, XUB, YLB, YUB$ such that

$$
P = \{(x, y) \mid XLB \leq x < XUB \& YLB \leq y < YUB\}.$$
Image Database

**Def:** An *image database*, $\text{IDB}$, consists of a triple ($\text{Gl}$, $\text{Prop}$, $\text{Rec}$) where:

1. $\text{Gl}$ is a set of gridded images of the form ($\text{Image}, m, n$) and
2. $\text{Prop}$ is a set of cell properties, and
3. $\text{Rec}$ is a mapping that associates with each image, a set of rectangles denoting objects.
Issues in Image Databases

- First and foremost, images are often very large objects consisting of a \((p_1 \times p_2)\) pixel array. Explicitly storing properties on a pixel by pixel basis is usually infeasible. This has led to a family of *image compression* algorithms that attempt to compress the image into one containing fewer pixels.

- Given an image \(I\) (compressed or raw), there is a critical need to determine what “features” appear in the image. This is typically done by breaking up the image into a set of homogeneous (w.r.t. some property) rectangular regions, each of which is called a *segment*. The process of finding these segments is called *segmentation*.

- Once image data has been segmented, we need to support “match” operations that map either a whole image or a segmented portion of an image against another whole/segmented image.
Compressed Image Representations

- Consider a 2-dimensional image $I$ consisting of $(p_1 \times p_2)$ pixels.

- Let $I(x, y)$ be a number denoting one or more attributes of the pixel.

- The creation of the compressed representation, $cr(I)$, of image $I$ consists of two parts:

1. **Size Selection**: The larger the size, the greater is the fidelity of the representation. However, as the size increases, so does the complexity of creating an index for manipulating such representations, and searching this index. Let $cr(I)$ be of size $h_1 \times h_2$ where $h_i \leq p_i$.

2. **Transform Selection**: The user must select a transformation, which, given the image $I$, and any pairs of number $1 \leq i \leq h_1$, and $1 \leq j \leq h_2$ will determine what the value of $cr(i, j)$ is.

3. There are many such transforms.
The Discrete Fourier Transform (DFT)

\[
\text{DFT}(i, j) = \frac{1}{\sqrt{p_1}} \times \frac{1}{\sqrt{p_2}} \times \sum_{a=0}^{p_1} \sum_{b=0}^{p_2} I(a, b) \times \exp \left( \frac{-2\pi ja \times i}{p_1} \right) \times \frac{-2\pi jb \times i}{p_2}.
\]

where: \( j \) is the well known complex number, \( \sqrt{-1} \).

DFT has many nice properties.

- **Invertibility:** It is possible to “get back” the original image \( I \) from its DFT representation. Useful for decompression.

- **Note that practical realizations of DFT ave often sacrificed this property by applying the DFT together with certain other non-invertible operations.**

- **Distance Preservation:** DFT preserves Euclidean distance. This is important in image matching applications where we may wish to use distance measures to represent similarity levels.
The Discrete Cosine Transform

\[ \text{DCT}(i, j) = \frac{2}{\sqrt{p_1 \times p_2}} \alpha(i) \times \alpha(j) \]

\[ = \sum_{r=0}^{p_1-1} \sum_{s=0}^{p_2-1} \cos \left( \frac{(2r + 1) \times \pi i}{2r} \right) \times \cos \left( \frac{(2s + 1) \times \pi j}{2s} \right) \]

where:

\[ \alpha(i), \alpha(j) = \begin{cases} \frac{1}{\sqrt{2}} & \text{when } u, v = 0 \\ 1 & \text{otherwise.} \end{cases} \]

- DCT also is easily invertible
- DCT can be computed quite fast.

Other compression techniques include Wavelets.
Image Processing: Segmentation

- This is the process of taking as input, an image, and producing as output, a way of “cutting up” the image into disjoint regions such that each region is “homogeneous”.

- Suppose $I$ is an image containing $(m \times n)$ cells.

- A connected region, $R$, in image $I$, is a set of cells such that if cells $(x_1, y_1), (x_2, y_2) \in R$, there there exists a sequence of cells $C_1, \ldots, C_n$ in $R$ such that:

  1. $C_1 = (x_1, y_1)$ and
  2. $C_n = (x_2, y_2)$ and
  3. The Euclidean distance between cells $C_i$ and $C_{i+1}$ for all $i < n$ is 1.
Example: Connected Regions

- Each of $R_1$, $R_2$, $R_3$ is a connected region.
- $(R_1 \cup R_2)$ is a connected region;
- $(R_2 \cup R_3)$ is a connected region;
- $(R_1 \cup R_2 \cup R_3)$ is a connected region;
- But $(R_1 \cup R_3)$ is not a connected region. The reason for this is that the Euclidean distance between the cell $(2, 3)$ which represents the rightmost of the two cells of $R_1$ and the cell $(3, 4)$ which represents the only cell of $R_3$ is $\sqrt{2} > 1$. 

Homogeneity Predicates

- A *homogeneity predicate* associated with an image $I$ is a function $H$ that takes as input, any connected region $R$ in image $I$, and returns either “true” or “false.”

- **Example:** Suppose $\delta$ is some real number between 0 and 1, inclusive, and we are considering black and white images. We may define a simple homogeneity predicate, $H_{\delta}^{bw}$ as follows: $H_{\delta}^{bw}(R)$ returns “true” if over $(100 \times \delta)\%$ of the cells in region $R$ have the same color.

  - Consider three regions now, as described in the following table:

<table>
<thead>
<tr>
<th>Region</th>
<th>Num. of Black Pixels</th>
<th>Num. of White Pixels</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_1$</td>
<td>800</td>
<td>200</td>
</tr>
<tr>
<td>$R_2$</td>
<td>900</td>
<td>100</td>
</tr>
<tr>
<td>$R_3$</td>
<td>100</td>
<td>900</td>
</tr>
</tbody>
</table>

  - Suppose we consider some different predicates, $H_{0.8}^{bw}$, $H_{0.89}^{bw}$ and $H_{0.92}^{bw}$.

  - The following table shows us the results returned by these three homogeneity predicates on the above table $R$.

<table>
<thead>
<tr>
<th>Region</th>
<th>$H_{0.8}^{bw}$</th>
<th>$H_{0.89}^{bw}$</th>
<th>$H_{0.92}^{bw}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_1$</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>$R_2$</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>$R_3$</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
</tbody>
</table>
Another Homogeneity Predicate Example

- Suppose each pixel has a real value between 0 and 1, inclusive.
- This value is called the bw-level.
- 0 denotes “white”, 1 denotes “black”, and everything in between denotes a shade somewhere between black and white.
- Suppose $f$ assigns numbers between 0 and 1 (inclusive) to each cell. In addition, you have a “noise factor” $0 \leq \eta \leq 1$, and a threshold $\delta$ as in the preceding case.
- $H^{f,\eta,\delta}(R)$ is now “true” iff
  \[
  \left\{ (x, y) \mid \frac{|\text{bwlevel}(x, y) - f(x, y)|}{(m \times n)} < \eta \right\} > \delta.
  \]
- What this homogeneity predicate does is to use a “baseline” function $f$, and a maximal permissible noise level $\eta$. It considers the bw-level of cell $(x, y)$ to be sufficiently similar to that predicted by $f$ if
  \[
  |\text{bwlevel}(x, y) - f(x, y)| < \eta,
  \]
i.e. if the two differ by no more than $\eta$.
- It then checks to see if sufficiently many cells (which is determined by the factor $\delta$) in the region “match” the predictions made by $f$. If so, it considers the region $R$ to be homogeneous, and returns “true.” Otherwise, it returns “false.”
Segmentation

**Def:** Given an image $I$ represented as a set of $(m \times n)$ pixels, we define a segmentation of image $I$ w.r.t. a homogeneity predicate $P$ to be a set $R_1, \ldots, R_k$ of regions such that:

1. $R_i \cap R_j = \emptyset$ for all $1 \leq i \neq j \leq k$ and
2. $I = R_1 \cup \cdots \cup R_k$;
3. $H(R_i) =$ “true” for all $1 \leq i \leq k$;
4. For all distinct $i, j$, $1 \leq i, j \leq n$ such that $R_i \cup R_j$ is a connected region, it is the case that $H(R_i \cup R_j) =$ “false.”

**Example:** For example, consider a simple $(4 \times 4)$ region containing the bw-levels shown in the table below.

<table>
<thead>
<tr>
<th>Row/Col</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.1</td>
<td>0.25</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>2</td>
<td>0.05</td>
<td>0.30</td>
<td>0.6</td>
<td>0.6</td>
</tr>
<tr>
<td>3</td>
<td>0.35</td>
<td>0.30</td>
<td>0.55</td>
<td>0.8</td>
</tr>
<tr>
<td>4</td>
<td>0.6</td>
<td>0.63</td>
<td>0.85</td>
<td>0.9</td>
</tr>
</tbody>
</table>

Consider now, the homogeneity predicate $H_{1}^{\text{dyn}, 0.03}$. This homogeneity predicate says that a region $R$ is to be considered homogeneous iff there exists an $r$ such that each and every cell in the
region has a bw-level \( v \) such that

\[ |v - r| \leq 0.03. \]

According to this classification, it is easy to see that the following five regions constitute a valid segmentation of the above image w.r.t. \( H_1^{dy,0.03} \).

\[
R_1 = \{(1, 1), (1, 2)\}.
R_2 = \{(1, 3), (2, 1), (2, 2), (2, 3)\}.
R_3 = \{(3, 1), (3, 2), (3, 3), (4, 1), (4, 2)\}.
R_4 = \{(3, 4), (4, 3), (4, 4)\}.
R_5 = \{(1, 4), (2, 4)\}.
\]
Segmentation Algorithm Sketch

- **Split**: In this method, we start with the whole image. If it is homogeneous, then we are done, and the image is a valid segmentation of itself. Otherwise, we split the image into two parts, and recursively repeat this process, till we find a set \( R_1, \ldots, R_n \) of regions that are homogeneous, and satisfy all conditions, except the fourth condition in the definition of “homogeneity” predicates.

- **Merge**: We now check which of the \( R_i \)’s can be merged together. At the end of this step, we will obtain a valid segmentation \( R'_1, \ldots, R'_k \) of the image, where \( k \leq n \) and where each \( R'_i \) is the union of some of the \( R_j \)’s.
Segmentation Algorithm

function segment(I:image);
    SOL = ∅;
    check_split(I);
    merge(SOL);
end function

function check_split(R);
    if $H(R) = 'true'$ then addsol(R)
    else
        { X = split(R);
        check_split(X.part1);
        check_split(X.part2);
        }
end function

procedure addsol(R);
    SOL = SOL ∪ \{R\}
end procedure
Segmentation Algorithm (Contd.)

function merge(S);
    while S ≠ ∅ do {
        Pick some Cand in S;
        merged = false;
        S = S − {Cand};
        Enumerate S as C₁,...,Cₖ;
        while i ≤ k do
            { if adjacent(Cand, Cᵢ) then
                { Cand = Cand ∪ Cᵢ;
                  S = S − {Cᵢ};
                  merged = true;
                }
            else { i = i + 1;
                if merged then S = S ∪ {Cand};
                merged = false
              }
            }
    }
end function
Similarity Based Retrieval

Which of these images is similar to the other?

(a) One Monkey (chimp)  (b) Another Monkey (orangutan)

(a) One Shark (tiger)  (b) Another Shark (grayreef)
Similarity Based Retrieval

Two approaches:

- **(The Metric Approach)** Assume there is a distance metric \( d \) that can compare any two image objects. The closer two objects are in distance, the more similar they are considered to be. *Given an input image \( i \), find the “nearest” neighbor of \( i \) in the image archive.* This is the most widely followed approach in the database world.

- **(The Transformation Approach)** The metric approach assumes that the notion of similarity is “fixed”, i.e. in any given application only one notion of similarity is used to index the data (though different applications may use different notions of similarity). Computes the “cost” of transforming one image into another based on user-specified cost functions that may vary from one query to another.
Metric Approach

- Suppose we consider a set \( \text{Obj} \) of objects, having pixel properties \( p_1, \ldots, p_n \), as described earlier in this chapter. Thus, each object \( o \) may be viewed as a set \( S(o) \).

- A function \( d \) from some set \( X \) to the unit interval \([0, 1]\) is said to be a distance function if it satisfies the following axioms for all \( x, y, z \in X \):

\[
\begin{align*}
    d(x, y) &= d(y, x). \\
    d(x, z) &\leq d(x, z) + d(z, y). \\
    d(x, x) &= 0.
\end{align*}
\]

- Let \( d_{\text{Obj}} \) be a distance function on the space of all objects in our domain, i.e. \( d_{\text{Obj}} \) is a distance function on a \( k = (n + 2) \) dimensional space.

- Example: \( \text{Obj} \) to consist of \((256 \times 256)\) images having three attributes (red, green, blue) each of which assumes a value from the set \( \{0, \ldots, 7\} \). Could have:

\[
\begin{align*}
    d_i(o_1, o_2) &= \sqrt{\sum_{i=1}^{256} \sum_{j=1}^{256} (\text{diff}_{r}[i, j] + \text{diff}_{g}[i, j] + \text{diff}_{b}[i, j])} \\
    \text{diff}_{r}[i, j] &= (o_1[i, j].\text{red} - o_2[i, j].\text{red})^2 \\
    \text{diff}_{g}[i, j] &= (o_1[i, j].\text{green} - o_2[i, j].\text{green})^2 \\
    \text{diff}_{b}[i, j] &= (o_1[i, j].\text{blue} - o_2[i, j].\text{blue})^2
\end{align*}
\]
Such computations can be cumbersome because the double summation leads to 65536 expressions being computed inside the sum.
**Metric Approach**

- How can this massive similarity computation be avoided?
- Suppose we have a “good” feature extraction function \( \text{fe} \).
- Use \( \text{fe} \) to map objects into single points in a \( s \)-dimensional space, where \( s \) would typically be pretty small compared to \( n + 2 \).
- This leads to two reductions:
  1. First, recall that an object \( o \) is a *set of points* in an \( (n+2) \) dimensional space. In contrast, \( \text{fe}(o) \) is a *single point*.
  2. Second, \( \text{fe}(o) \) is a point in an \( s \)-dimensional space where \( s \ll (n + 2) \).

- Mapping must preserve distance, i.e. if \( o_1, o_2, o_3 \) are objects such that the distance \( d(o_1, o_2) \leq d(o_1, o_3) \), then \( d'(\text{fe}(o_1), \text{fe}(o_2)) \leq d'(\text{fe}(o_1), \text{fe}(o_3)) \) where \( d \) is a metric on the original \( (n + 2) \) dimensional space, and \( d' \) is a metric on the new, \( s \)-dimensional space. In other words, the feature extraction map should preserve the distance relationships in the original space.
Reducing Dimensionality of Feature Space

(n+2) dim. space → Map FE → s-dimensional space → Indexing Algorithm → INDEX → Object Repository
Index Creation Algorithm

Input: \textit{Obj}, a set of objects.

1. \( T = NIL \). (* \( T \) is an empty quadtree, or R-tree for \( s \)-dimensional data *)

2. \textbf{if} \( \textit{Obj} = \emptyset \) \textbf{then return} \( T \) \textbf{and} \textit{halt}.

3. \textbf{else}

   (a) Compute \( \text{fe}(o) \),
   (b) Insert \( \text{fe}(o) \) into \( T \).
   (c) \( \textit{Obj} = \textit{Obj} - \{o\} \).
   (d) Goto 2
Finding the Best Matches

FindMostSimilarObject Algorithm

Input: a tree $T$ of the above type. An object $o$.

1. $\text{bestnode} = \text{NIL}$;
2. if $T = \text{NIL}$ then return $\text{bestnode}$. Halt
3. else
   
   - find the nearest neighbors of $\text{fe}(o)$ in $T$ using a nearest neighbor search technique. If multiple such neighbors exist, return them all.

---

V.S. Subrahmanian — All Rights Reserved

These slides may not be duplicated without explicit written permission from Morgan Kaufmann Press.

Principles of Multimedia Database Systems    Morgan Kaufmann    Copyright ©1997
Finding “Sufficiently” Similar Objects

FindSimilarObjects Algorithm

Input: a tree $T$ of the above type. An object $o$. A tolerance $0 < \epsilon \leq 1$.

1. Execute a range query on tree $T$ with center $fe(o)$ and radius $\epsilon$.
2. Let $o_1, \ldots, p_r$ be all the points returned.
3. for $i = 1$ to $r$ do
   (a) if $d(o, fe^{-1}(o_i)) \leq \epsilon$ then print $fe^{-1}(o_i)$.

The above algorithm works only if the distance metric in the space of small dimensionality (i.e. dimension $s$) consistently overestimates the distance metric $d$. 
The Transformation Approach

- Based on the principle that given two objects $o_1, o_2$, the level of *dis-similarity* between $o_1, o_2$ is proportional to the (minimum) cost of transforming object $o_1$ into object $o_2$, or vice-versa.

- We start with a set of transformation operators, $to_1, \ldots, to_r$, e.g.

  - translation
  - rotation
  - scaling – uniform and nonuniform
  - excision (that culls out a part of an image)

- The transformation of object $o$ into object $o'$ is a *sequence* of transformation operations $to_1, \ldots, to_r$ and a sequence of objects $o_1, \ldots, o_r$ such that:

  1. $to_1(o) = o_1$ and
  2. $to_i(o_{i-1}) = o_i$ and
  3. $to(o_r) = o'$.

The cost of the above transformation sequence, $TS$ is given by:

\[
\text{cost}(TS) = \sum_{i=1}^{r} \text{cost}(to_i).
\]
The Transformation Approach

- Suppose $\text{TSeq}(o, o')$ is the set of all transformation sequences that convert $o$ into $o'$.

- The dissimilarity between $o$ and $o'$, denoted $\text{dis}(o, o')$ w.r.t. a set $TR$ of transformation operators, and a set $CF$ of cost functions is given by:

$$\text{dis}(o, o') = \min\{\text{cost}(TS) \mid TS \in \text{TSeq}(o, o') \cup \text{TSeq}(o', o)\}.$$
Example

$TS_1$: This transformation sequence consists of a

- non-uniform scaling operation (scale the blue part of $o_1$ 50% in the vertical upward direction, leaving the horizontal unchanged),

- a non-uniform scaling operation (scale the green part of object $o_1$ by a 100% increase in the vertical, downward direction, with no change in the horizontal).

- The third operation applies the \textit{paint} operation, painting the two pixels colored magenta to green.

- The Figure below depicts the intermediate steps.
(a) non-uniform scaling

(b) paint
Example

(a)  (b)

$TS_2$: This transformation sequence consists of a

- non-uniform scaling operation (scale the blue part of $o_1$ 50% in the vertical upward direction, leaving the horizontal unchanged).
- apply the $\text{paint}$ operation, painting the green object magenta.
- apply the non-uniform scaling operation (scale the magenta part of object $o_1$ by a 100% increase in the vertical, downward direction, with no change in the horizontal).
- The following figure depicts the intermediate steps.
- If we assume that the cost functions associated with non-uniform scaling are independent of color, and the paint operation merely counts the number of pixels being painted, then it is easy to see that transformation $TS_2$ accomplishes the desired transformation at a cheaper cost (as it paints one less pixel than transformation $TS_1$.
Transformation Model vs. the Metric Model

Advantages of the Transformation Model over the Metric Model

- First and foremost, the user can “set up” his own notion of similarity by specifying that certain transformation operators may/may not be used.

- Second, the user may associate, with each transformation operator, a cost function that assesses a cost to each application of the operation, depending upon the arguments to the transformation operator. This allows the user to personalize the notion of similarity for his/her needs.

Advantages of the Metric Model over the Transformation Model

By forcing the user to use one and only one (dis)similarity metric, the system can facilitate the indexing of data so as to optimize the one operation of finding the “nearest” neighbor (i.e. least dis-similar) object w.r.t. the query object specified by the user.
Alternative Image DB Paradigms

- Representing IDBs as Relations
- Representing IDBs with Spatial Data Structures
- Representing IDBs using Image Transformations

Two different photographs of the same person may vary, depending upon a variety of factors such as:

1. the time of the day at which the two photographs were taken;
2. the lighting conditions under which the photographs were taken;
3. the camera used;
4. the exact position of the subject’s head and his/her facial expression.
5. etc.
Representing Image DBs with Relations

Suppose $\text{IDB} = (\text{Gl}, \text{Prop}, \text{Rec})$.

1. Create a relation called $\text{images}$ having the scheme:

   $$(\text{Image}, \text{ObjID}, \text{XLB}, \text{XUB}, \text{YLB}, \text{YUB})$$

   where $\text{Image}$ is the name of an image file, $\text{ObjID}$ is a dummy name created for an object contained in the image, and $\text{XLB}, \text{XUB}, \text{YLB}, \text{YUB}$ describe the rectangle in question. If $R$ is a rectangle specified by $\text{XLB}, \text{XUB}, \text{YLB}, \text{YUB}$ and $R$ is in $\text{Rec}(I)$, then there exists a tuple

   $$(I, \text{newid}, \text{XLB}, \text{XUB}, \text{YLB}, \text{YUB})$$

   in the relation $\text{images}$.

2. For each property $p \in \text{Prop}$, create a relation $R_p$ having the scheme:

   $$(\text{Image}, \text{XLB}, \text{XUB}, \text{YLB}, \text{YUB}, \text{Value})$$

   Here, $\text{Image}$ is the name of an image file. Unlike the preceding case though, $\text{XLB, XUB, YLB, YUB}$ denote a rectangular cell in the image, and $\text{Value}$ specifies the value of the property $p$.

Example:
<table>
<thead>
<tr>
<th>Image</th>
<th>ObjId</th>
<th>XLB</th>
<th>XUB</th>
<th>YLB</th>
<th>YUB</th>
</tr>
</thead>
<tbody>
<tr>
<td>pic1.gif</td>
<td>$o_1$</td>
<td>10</td>
<td>60</td>
<td>5</td>
<td>50</td>
</tr>
<tr>
<td>pic1.gif</td>
<td>$o_2$</td>
<td>80</td>
<td>120</td>
<td>20</td>
<td>55</td>
</tr>
<tr>
<td>pic2.gif</td>
<td>$o_3$</td>
<td>20</td>
<td>65</td>
<td>20</td>
<td>75</td>
</tr>
<tr>
<td>pic3.gif</td>
<td>$o_4$</td>
<td>25</td>
<td>75</td>
<td>10</td>
<td>60</td>
</tr>
<tr>
<td>pic4.gif</td>
<td>$o_5$</td>
<td>20</td>
<td>60</td>
<td>30</td>
<td>80</td>
</tr>
<tr>
<td>pic5.gif</td>
<td>$o_6$</td>
<td>0</td>
<td>40</td>
<td>15</td>
<td>50</td>
</tr>
<tr>
<td>pic6.gif</td>
<td>$o_7$</td>
<td>20</td>
<td>75</td>
<td>15</td>
<td>80</td>
</tr>
<tr>
<td>pic6.gif</td>
<td>$o_8$</td>
<td>20</td>
<td>70</td>
<td>130</td>
<td>185</td>
</tr>
<tr>
<td>pic7.gif</td>
<td>$o_9$</td>
<td>15</td>
<td>70</td>
<td>15</td>
<td>75</td>
</tr>
</tbody>
</table>
Image Properties

- Pixel Level Properties: e.g. RGB values
- Object/Region Level Properties: e.g. NAME, AGE
- Image Level Properties: e.g. when image was captured, where, and by whom
Querying (Relational Representations of) Image DBs

- Eliciting the contents of an image is done using image processing algorithms.
- Image processing algorithms are usually only partially accurate.
- This implies that tuples placed in a relation by an image processing program has certain associated probabilistic attributes.
- Each has object has a value associated with each property $p \in \text{Prop}$.
- Each relation $R_p$ associated with object/region level properties as well as image level properties has just two attributes: an object (or region) id, and a value for the property.
- Example:

  Relation name
<table>
<thead>
<tr>
<th>ObjId</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>(o_1)</td>
<td>Jim Hatch</td>
</tr>
<tr>
<td>(o_2)</td>
<td>John Lee</td>
</tr>
<tr>
<td>(o_3)</td>
<td>John Lee</td>
</tr>
<tr>
<td>(o_4)</td>
<td>Jim Hatch</td>
</tr>
<tr>
<td>(o_5)</td>
<td>Bill Bosco</td>
</tr>
<tr>
<td>(o_6)</td>
<td>Dave Dashell</td>
</tr>
<tr>
<td>(o_7)</td>
<td>Ken Yip</td>
</tr>
<tr>
<td>(o_8)</td>
<td>Bill Bosco</td>
</tr>
<tr>
<td>(o_7)</td>
<td>Ken Yip</td>
</tr>
</tbody>
</table>

- This description must be enhanced to include probabilistic object identification.
Probabilistic Version of the name relation

<table>
<thead>
<tr>
<th>ObjId</th>
<th>Name</th>
<th>Prob</th>
</tr>
</thead>
<tbody>
<tr>
<td>o₁</td>
<td>Jim Hatch</td>
<td>0.8</td>
</tr>
<tr>
<td>o₁</td>
<td>Dave Fox</td>
<td>0.2</td>
</tr>
<tr>
<td>o₂</td>
<td>John Lee</td>
<td>0.75</td>
</tr>
<tr>
<td>o₂</td>
<td>Ken Yip</td>
<td>0.15</td>
</tr>
<tr>
<td>o₃</td>
<td>John Lee</td>
<td>1</td>
</tr>
<tr>
<td>o₄</td>
<td>Jim Hatch</td>
<td>1</td>
</tr>
<tr>
<td>o₅</td>
<td>Bill Bosco</td>
<td>1</td>
</tr>
<tr>
<td>o₆</td>
<td>Dave Dashell</td>
<td>1</td>
</tr>
<tr>
<td>o₇</td>
<td>Ken Yip</td>
<td>0.7</td>
</tr>
<tr>
<td>o₇</td>
<td>John Lee</td>
<td>0.3</td>
</tr>
<tr>
<td>o₈</td>
<td>Bill Bosco</td>
<td>0.6</td>
</tr>
<tr>
<td>o₈</td>
<td>Dave Dashell</td>
<td>0.2</td>
</tr>
<tr>
<td>o₉</td>
<td>Jim Hatch</td>
<td>0.10</td>
</tr>
<tr>
<td>o₉</td>
<td>Ken Yip</td>
<td>1</td>
</tr>
</tbody>
</table>

Reading:

1. The probability that “John Lee” is the name attribute of o₂ is 0.75.

2. The probability that “Ken Yip” is the name attribute of o₂ is 0.15.

3. There is, in this case, a 10% missing probability.
Complex Queries

- Suppose we ask the query \textit{What is the probability that \texttt{pic1.gif} contains both Jim Hatch and Ken Yip?} Is the answer the product of the two probabilities, i.e. is it $(0.8 \times 0.15) = 0.12$?

- \textit{What is the probability that \texttt{pic6.gif} contains both Jim Hatch and Ken Yip?} Is the answer $(0.7 \times 0.1) = 0.07$?

- In general, the answer is NO.

\textbf{Example:} Consider a hypothetical image \texttt{pic8.gif} with two objects $o_{10}, o_{11}$ in it, and suppose our table above is expanded by the insertion of the following new tuples identified by the image processing algorithm.

<table>
<thead>
<tr>
<th>ObjId</th>
<th>Name</th>
<th>Prob</th>
</tr>
</thead>
<tbody>
<tr>
<td>$o_{10}$</td>
<td>Ken Yip</td>
<td>0.5</td>
</tr>
<tr>
<td>$o_{10}$</td>
<td>Jim Hatch</td>
<td>0.4</td>
</tr>
<tr>
<td>$o_{11}$</td>
<td>Jim Hatch</td>
<td>0.8</td>
</tr>
<tr>
<td>$o_{11}$</td>
<td>John Lee</td>
<td>0.1</td>
</tr>
</tbody>
</table>

If we are ignorant about the dependencies between different events (as we are in the above case, then we are forced to confront \textit{four possibilities}:}
Possibility 1 \( o_{10} \) is Ken Yip and \( o_{11} \) is John Hatch.

Possibility 2 \( o_{10} \) is Ken Yip and \( o_{11} \) is not John Hatch.

Possibility 3 \( o_{10} \) is not Ken Yip but \( o_{11} \) is John Hatch.

Possibility 4 \( o_{10} \) is not Ken Yip and \( o_{11} \) is not John Hatch.
Complex Queries

- Suppose $p_i$ denotes the probability of Possibility $i$, $1 \leq i \leq 4$. Then, we can say that:

\[
\begin{align*}
    p_1 + p_2 &= 0.5. \\
    p_3 + p_4 &= 0.5. \\
    p_1 + p_3 &= 0.8. \\
    p_2 + p_4 &= 0.2 \\
    p_1 + p_2 + p_3 + p_4 &= 1.
\end{align*}
\]

- The first equation follows from the fact that $o_{10}$ is Ken Yip according to possibilities 1 and 2, and we know from the table, that the probability of $o_{10}$ being Ken Yip is 0.5.

- The second equation follows from the fact that $o_{10}$ is someone other than Ken Yip according to possibilities 3 and 4, and we know from the table, that the probability of $o_{10}$ not being Ken Yip is 0.5.

- The third equation follows from the fact that $o_{11}$ is Jim Hatch according to possibilities 1 and 3, and we know from the table, that the probability of $o_{11}$ being Jim Hatch is 0.8.

- Finally, the last equation follows from the fact that $o_{11}$ is someone other than Jim Hatch according to possibilities 2 and 4, and we know from the table, that the probability of $o_{11}$ not being John Hatch is 0.2.
• In order to determine the probability that pic8.gif contains both Ken Yip and John Hatch, we must attempt to solve the above system of linear equations for \( p_1 \), keeping in mind the fact that all scenarios possible are covered by our four possibilities. The result we obtain, using a linear programming engine, is that \( p_1 \)'s probability is not uniquely determinable. It could be as low as 0.3 or as high as 0.5, or anywhere in between. In particular, note that merely multiplying the probability of 0.5 associated with Ken Yip being object \( o_{10} \) and the probability value 0.8 of Hatch being object \( o_{11} \) leads to a probability of 0.4 which is certainly inside this interval, but does not accurately capture the four possibilities listed above.

• Requires the use of interval probabilities.
Interval Probability Model

Interval Probabilistic Version of the **name** relation with **pic8.gif** included

<table>
<thead>
<tr>
<th>ObjId</th>
<th>Name</th>
<th>Prob (Lower)</th>
<th>Prob (Upper)</th>
</tr>
</thead>
<tbody>
<tr>
<td>o₁</td>
<td>Jim Hatch</td>
<td>0.77</td>
<td>0.83</td>
</tr>
<tr>
<td>o₁</td>
<td>Dave Fox</td>
<td>0.17</td>
<td>0.23</td>
</tr>
<tr>
<td>o₂</td>
<td>John Lee</td>
<td>0.72</td>
<td>0.78</td>
</tr>
<tr>
<td>o₂</td>
<td>Ken Yip</td>
<td>0.12</td>
<td>0.18</td>
</tr>
<tr>
<td>o₃</td>
<td>John Lee</td>
<td>0.97</td>
<td>1.00</td>
</tr>
<tr>
<td>o₄</td>
<td>Jim Hatch</td>
<td>0.97</td>
<td>1.00</td>
</tr>
<tr>
<td>o₅</td>
<td>Bill Bosco</td>
<td>0.97</td>
<td>1.00</td>
</tr>
<tr>
<td>o₆</td>
<td>Dave Dashell</td>
<td>0.97</td>
<td>1.00</td>
</tr>
<tr>
<td>o₇</td>
<td>Ken Yip</td>
<td>0.67</td>
<td>0.73</td>
</tr>
<tr>
<td>o₇</td>
<td>John Lee</td>
<td>0.27</td>
<td>0.33</td>
</tr>
<tr>
<td>o₈</td>
<td>Bill Bosco</td>
<td>0.57</td>
<td>0.63</td>
</tr>
<tr>
<td>o₈</td>
<td>Dave Dashell</td>
<td>0.17</td>
<td>0.23</td>
</tr>
<tr>
<td>o₈</td>
<td>Jim Hatch</td>
<td>0.07</td>
<td>0.13</td>
</tr>
<tr>
<td>o₉</td>
<td>Ken Yip</td>
<td>0.97</td>
<td>1.00</td>
</tr>
<tr>
<td>o₁₀</td>
<td>Ken Yip</td>
<td>0.47</td>
<td>0.53</td>
</tr>
<tr>
<td>o₁₀</td>
<td>Jim Hatch</td>
<td>0.37</td>
<td>0.43</td>
</tr>
<tr>
<td>o₁₁</td>
<td>Jim Hatch</td>
<td>0.77</td>
<td>0.83</td>
</tr>
<tr>
<td>o₁₀</td>
<td>Hohn Lee</td>
<td>0.07</td>
<td>0.13</td>
</tr>
</tbody>
</table>

- "**Find an image that contains both Ken Yip and Jim Hatch.**"
• Let us re-examine the image pic8.gif and see what the probability of this image containing both Ken Yip and Jim Hatch is.

• Constraints generated:

\[
0.47 \leq p_1 + p_2 \leq 0.53.
\]

\[
0.47 \leq p_3 + p_4 \leq 0.53.
\]

\[
0.77 \leq p_1 + p_3 \leq 0.83.
\]

\[
0.17 \leq p_2 + p_4 \leq 0.23.
\]

\[
p_1 + p_2 + p_3 + p_4 = 1.
\]

• Solving the above linear program for minimal and maximal values of the variable \( p_1 \), we obtain 0.24 and 0.53, respectively.

• Beauty is that when implementing this, we can avoid solving the linear program altogether.
A General Approach

- A **probabilistic relation** over a scheme \((A_1, \ldots, A_n)\) is an ordinary relation over the scheme \((A_1, \ldots, A_n, LB, UB)\) where the domain of the \(LB\) and \(UB\) attributes is the unit interval \([0, 1]\) of real numbers.

- In particular, the relation **name** is a probabilistic relation that has three attributes:

\[
(ImageId, ObjectId, Name)
\]

of the sort we have already seen thus far.

- The **name** relation satisfies some integrity constraints:

\[
(\forall t_1, t_2) t_1. ObjId = t_2. ObjId \rightarrow t_1. ImageId = t_2. ImageId.
\]

This constraint states that an ObjectId can be associated with only one image, i.e. distinct images have distinct ObjectIds. The following constraint says that the LB field of any tuple is always smaller than the UB field.

\[
(\forall t) t. LB \leq t. UB.
\]

- An **image database** consists of a probabilistic relation called **name** of the above form, together with a set of ordinary (i.e. non-probabilistic) relations \(R_1, \ldots, R_k\) corresponding to image properties.
Membership Queries

- A membership query in an image database is a query of the form: Find all images in the image database that contain objects named \( s_1, \ldots, s_n \).

- SELECT ImageId
  FROM name \( T_1, \ldots, T_n \)
  WHERE \( T_1.Name = s_1 \) AND \( \cdots \) AND \( T_n.Name = s_n \) AND
  \( T_1.ImageId = T_2.ImageId \) AND \( \cdots \) AND
  \( T_1.ImageId = T_n.ImageId \).

The result of this membership query is a table containing three fields — the ImageId fields that is explicitly listed in the query, a LB field, and a UB field. \( (im, \ell, u) \) is in the result iff for each \( 1 \leq j \leq n \), there exists a tuple \( t_j \in \text{name} \) such that:

1. \( t_.ImageId = im \) and
2. \( t_.LB = \ell_i \) and \( t_.UB = u_i \) and
3. \( [\ell, u] = [\ell_1, u_1] \otimes [\ell_2, u_2] \otimes \cdots \otimes [\ell_n, u_n] \)

where

\[
[x, y] \otimes [x', y'] = [\max(0, x + x' - 1), \min(y, y')].
\]
Other Queries

- Find all people who have had deposits of over 9000 dollars, and who have been photographed with Denis Jones.

- SELECT I.ImageId
  FROM name I, bank B
  WHERE I CONTAINS B.name, Denis Jones AND B.trans=deposit AND B.amount > 9000 AND B.name = I.name.
Representing Image DBs with R-Trees

1. Create a relation called \textit{occursin} with two attributes

   \[(\text{ImageId, ObjId})\]

   specifying which objects appear in which images.

2. Create \textit{one} R-tree that stores all the rectangles. If the same
   rectangle (say with \(XLB = 5, XUB = 15, YLB = 20, YUB = 30\))
   appears in two images, then we have an overflow list associated
   with that node in the R-tree.

3. Each rectangle has an associated set of fields that specifies
   the object/region level properties of that rectangle.
Example
occurs in Relation

| pic1.gif   | o_1  |
| pic1.gif   | o_2  |
| pic2.gif   | o_3  |
| pic3.gif   | o_4  |
| pic4.gif   | o_5  |
| pic5.gif   | o_6  |
| pic6.gif   | o_7  |
| pic6.gif   | o_8  |
| pic7.gif   | o_9  |
R-tree representation

facenode = record
  Rec1, Rec2, Rec3: rectangle;
  P1, P2, P3: ↑rtnodetype
end

rectangle = record
  XLB, XUB, YLB, YUB: integer;
  objlist: ↑objnode;
  day, mth, yr: integer;
  camera_type: string;
  place: string
end

objnode = record
  objid: string;
  imageid: string;
  info: infotype
end

infotype = record
  objname: string;
  Lp, Up: real: (* lower and upper probability bounds *)
  Next: ↑objinfo
end
R-Tree Construction

- Get Rectangles
- Create R-tree
- Flesh out Objects
Get Rectangles

First and foremost, we may construct a small table describing all the rectangles that occur, and the images they occur in.

<table>
<thead>
<tr>
<th>ObjId</th>
<th>ImageId</th>
<th>XLB</th>
<th>XUB</th>
<th>YLB</th>
<th>YUB</th>
</tr>
</thead>
<tbody>
<tr>
<td>$o_1$</td>
<td>pic1.gif</td>
<td>10</td>
<td>60</td>
<td>5</td>
<td>50</td>
</tr>
<tr>
<td>$o_2$</td>
<td>pic1.gif</td>
<td>80</td>
<td>120</td>
<td>20</td>
<td>55</td>
</tr>
<tr>
<td>$o_3$</td>
<td>pic2.gif</td>
<td>20</td>
<td>65</td>
<td>20</td>
<td>75</td>
</tr>
<tr>
<td>$o_4$</td>
<td>pic3.gif</td>
<td>25</td>
<td>75</td>
<td>10</td>
<td>60</td>
</tr>
</tbody>
</table>
Create R-Tree

We then create an R-tree representing the above rectangles. At this stage the object nodes in the R-tree may still not be “filled in” completely.
Flesh Out Objects

We then “flesh out” and appropriately fill in the fields of the various objects stored in the R-tree.

LEGEND:

<table>
<thead>
<tr>
<th>objid</th>
<th>imageid</th>
<th>nxt</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>info</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>objname</th>
<th>Lp</th>
<th>Up</th>
</tr>
</thead>
</table>

Ch. 5 Image Databases

V.S. Subrahmanian — All Rights Reserved
These slides may not be duplicated without explicit written permission from Morgan Kaufmann Press.
Principles of Multimedia Database Systems  Morgan Kaufmann  Copyright ©1997
**Generalized R-trees**

- Previous representation does not provide an efficient way to perform nearest neighbor searches.
- Why? Only 2-attributes (bounding rectangles) are stored.
- In general, an object $o$ has an associated $(n + 2)$-dimensional vector.
- If each point in the 2-d rectangle has $n$ attributes, then we need to use a generalized rectangle.
- A *generalized rectangle* for a space of dimensionality $g$ (specifically consider $g = n + 2$) may be defined by a set of constraints of the form:
  \[
  \ell_1 \leq x_1 \leq u_1 \\
  \ell_2 \leq x_2 \leq u_2 \\
  \ldots \\
  \ell_g \leq x_g \leq u_g 
  \]
- When $g = 2$, we have $n = 0$, and in this case, an ordinary 2-dimensional rectangle is a special case of this definition.
- Sets of generalized rectangles are represented by generalized $R$-trees, or gR-trees.
gR-Trees

A generalized R-tree (gR-tree) of order $K$ is exactly like an R-tree except for the following factors:

- First, each node $N$ represents a generalized bounding rectangle $GBR(N)$ of dimensionality $(n + 2)$, which is represented by $2 \times (n + 2)$ real number fields, one for the lower bound and upper bound, respectively, of each dimension.

- When a node $N$ is split, the union of the generalized bounding rectangles associated with its children equals the generalized bounding rectangle associated with $N$.

- Each node (other than the root and the leaves) contains at most $K$ generalized bounding rectangles and at least $\lceil K/2 \rceil$ generalized rectangles.

- As usual, all $(n + 2)$-dimensional “data” rectangles are stored in leaves.
Nearest Rectangular Neighbors

- Suppose $R_Q$ is a query rectangle (which may represent an image object).
- Find all rectangles in a gR-tree $T$ that are as close to $R_Q$ as possible (where closeness is defined by a metric $d$ on points).
- We extend the metric $d$ to apply to rectangles as follows:
  \[ d(R, R') = \min\{d(p, p') \mid p \in R, p' \in R'\}. \]
Algorithm

\textbf{Algorithm 1} \textit{NN\_Search\_GR}(T, R_Q)
\begin{align*}
SOL &= NIL; (* \text{no solution so far} *); \\
Todo &= \text{List containing } T \text{ only}; \\
Bestdist &= \infty; (* \text{distance of best solution from } R_Q *); \\
\textbf{while} & \quad Todo \neq NIL \textbf{ do} \\
& \quad \{ \\
& \quad \quad F = \text{first element of } Todo; \\
& \quad \quad Todo = \text{delete } F \text{ from } Todo; \\
& \quad \quad \textbf{if} d(GBR(F), R_Q) < Bestdist \textbf{ then} \\
& \quad \quad \quad \{ \\
& \quad \quad \quad \quad \text{Compute children } N_1, \ldots, N_r \text{ of } F; \\
& \quad \quad \quad \quad \textbf{if} N_i \text{'s are leaves of } T \textbf{ then} \\
& \quad \quad \quad \quad \quad \{ \\
& \quad \quad \quad \quad \quad \quad N_{min} = \text{any } N_i \text{ at minimal distance from } R_Q; \\
& \quad \quad \quad \quad \quad \quad Ndist = d(GBR(N_i), R_Q)); \\
& \quad \quad \quad \quad \quad \quad \textbf{if} Ndist < Bestdist \textbf{ then} \\
& \quad \quad \quad \quad \quad \quad \quad \{ \\
& \quad \quad \quad \quad \quad \quad \quad \quad Bestdist = Ndist; SOL = N_{min}; \\
& \quad \quad \quad \quad \quad \quad \quad \} \\
& \quad \quad \quad \quad \textbf{else} \quad Todo = \text{insert all } N_i \text{'s into } Todo \text{ in order of distance from } R_Q; \\
& \quad \quad \quad \} \\
& \quad \} \\
\textbf{Return} & \quad SOL; \\
\textbf{end}
\end{align*}
Retrieving Images by Spatial Layout

Given an image $I$, and two objects (represented by rectangles) $o_1$, $o_2$ in $I$, a user may wish to ask queries of the form:

1. Is $o_1$ to the South of $o_2$ ?
2. Is $o_1$ to the South-East of $o_2$ ?
3. Are $o_1$ and $o_2$ overlapping ?
## One-dimensional Precedence Relations

<table>
<thead>
<tr>
<th>o1</th>
<th>o2</th>
<th>Description</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Image" /></td>
<td><img src="image2.png" alt="Image" /></td>
<td>o1 before o2</td>
<td>B(o1,o2)</td>
</tr>
<tr>
<td><img src="image3.png" alt="Image" /></td>
<td><img src="image4.png" alt="Image" /></td>
<td>o1 meets o2</td>
<td>M(o1,o2)</td>
</tr>
<tr>
<td><img src="image5.png" alt="Image" /></td>
<td><img src="image6.png" alt="Image" /></td>
<td>o1 overlaps o2</td>
<td>OV(o1,o2)</td>
</tr>
<tr>
<td><img src="image7.png" alt="Image" /></td>
<td><img src="image8.png" alt="Image" /></td>
<td>o1 during o2</td>
<td>D(o1,o2)</td>
</tr>
<tr>
<td><img src="image9.png" alt="Image" /></td>
<td><img src="image10.png" alt="Image" /></td>
<td>o1 starts o2</td>
<td>S(o1,o2)</td>
</tr>
<tr>
<td><img src="image11.png" alt="Image" /></td>
<td><img src="image12.png" alt="Image" /></td>
<td>o1 finishes o2</td>
<td>F(o1,o2)</td>
</tr>
<tr>
<td><img src="image13.png" alt="Image" /></td>
<td><img src="image14.png" alt="Image" /></td>
<td>o1 equal o2</td>
<td>EQ(O1,o2)</td>
</tr>
</tbody>
</table>
2-Dimensional Precedence Relations

It is easy to extend the precedence relation along one dimension, to the case of two dimensions, is straightforward. If we use the notation \( o[x] \) and \( o[y] \) to denote the projection of object \( o \) on the \( x \) and \( y \) axes, respectively, then it is easy to capture our spatial relationships as follows:

1. We say \( o_1 \) is \textbf{South} of \( o_2 \) iff \( B(o_1[y], o_2[y]) \) and either (i) \( D(o_1[x], o_2[x]) \) or (ii) \( D(o_2[x], o_1[x]) \) or (iii) \( S(o_1[x], o_2[x]) \) or (iv) \( S(o_2[x], o_1[x]) \) holds or (v) \( F(o_1[x], o_2[x]) \) or (vi) \( F(o_2[x], o_1[x]) \) holds or (vii) \( E(o_1[x], o_2[x]) \) holds.

2. Likewise, we say that \( o_1 \) is to the \textbf{Left} of \( o_2 \) iff either (i) \( B(o_1[x], o_2[x]) \) holds or (ii) \( M(o_1[x], o_2[x]) \) holds.
Text/Document Databases
What is a text database?

- User wants to find documents related to a topic \( T \).
- The search program tries to find the documents in the "document database" that contain the string \( T \).
- This has two problems:
  1. **Synonymy**: Given a word \( T \) (i.e. specifying a topic), the word \( T \) does not occur anywhere in a document \( D \), even though the document \( D \) is in fact closely related to the topic \( T \) in question.
  2. **Polysemy**: The same word may mean many different things in different contexts.

- Consider a text database that only indexes the following titles.

<table>
<thead>
<tr>
<th>DocumentID</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>( d_1 )</td>
<td>Jose Orojuelo’s Operations in Bosnia.</td>
</tr>
<tr>
<td>( d_2 )</td>
<td>The Medelin Cartel’s Financial Organization.</td>
</tr>
<tr>
<td>( d_3 )</td>
<td>The Cali Cartel’s Distribution Network.</td>
</tr>
<tr>
<td>( d_4 )</td>
<td>Banking Operations and Money Laundering.</td>
</tr>
<tr>
<td>( d_5 )</td>
<td>Profile of Hector Gomez.</td>
</tr>
<tr>
<td>( d_6 )</td>
<td>Connections between Terrorism and Asian Dope Operations.</td>
</tr>
<tr>
<td>( d_7 )</td>
<td>Hector Gomez’s: How he Gave Agents the Slip in Cali.</td>
</tr>
<tr>
<td>( d_8 )</td>
<td>Sex, Drugs, and Videotape.</td>
</tr>
<tr>
<td>( d_9 )</td>
<td>The Iranian Connection.</td>
</tr>
<tr>
<td>( d_{10} )</td>
<td>Boating and Drugs: Slips owned by the Cali Cartel.</td>
</tr>
</tbody>
</table>
Organization of this Topic

- Measures of performance of a text retrieval system.
- Latent Semantic Indexing
- Telescopic-Vector Trees for Document Retrieval.
• Suppose $D$ is a finite set of documents.

• Suppose $A$ is any algorithm that takes as input, a topic string $T$, returns as output, a set ($T$) of documents.

• **Precision** of algorithm $A$ w.r.t. the predicate $relevant$ and test set $D_{test}$ is $P\%$ for topic $t \in T_{test}$ iff:

$$P_t = 100 \times \frac{1 + \text{card} \{d \in D_{test} \mid d \in A(t) \land relevant(t, d) \text{ is true} \}}{1 + \text{card} \{d \in D_{test} \mid d \in A(t) \}}.$$  

(To avoid division by zero, we add one to both the numerator and denominator above). We say that the precision of algorithm $A$ w.r.t. the predicate $relevant$, the document test set $D_{test}$ and the topic test set $T_{test}$ is $P\%$ iff:

$$P = \frac{\sum_{t \in T_{test}} P_t}{\text{card}(T_{test})}.$$  

• Precision basically says: How many of the answers returned are in fact correct.

• **Recall** of an algorithm $A$ is a measure of “how many” of the right documents are in fact retrieved by the query.

• Recall $R_t$ associated with a topic $t$ is given by the formula:

$$R_t = 100 \times \frac{1 + \text{card} \{d \in D_{test} \mid d \in A(t) \land relevant(t, d) \text{ is true} \}}{1 + \text{card} \{d \in D_{test} \mid relevant(t, d) \text{ is true} \}}.$$
The overall recall rate $R$ associated with test sets $D_{test}$ of
documents, and $T_{test}$ of topics, is given by:

$$R = \frac{\sum_{t \in T_{test}} R_t}{\text{card}(T_{test})}.$$
Stop Lists, Word Stems, and Frequency Tables

- **Stop List:** This is a set of words that do not "discriminate" between the documents in a given archive.

- E.g. Cornell SMART system has about 440 words on its stop list.

- **Word Stems:** Many words are small syntactic variants of each other. For example, the words *drug*, *drugged*, *drugs*, are all similar in the sense that they share a common "stem", viz. the word *drug*.

- Most document retrieval systems first eliminate words on stop lists and they also reduce words to their stems, before creating a frequency table.

- **Frequency Tables:** Suppose
  
  - \( D \) is a set of \( N \) documents, and
  
  - \( T \) is a set of \( M \) words/terms occurring in the documents of \( D \).
  
  - Assume that no words on the stop list for \( D \) occur in \( T \), and that all words in \( T \) have been stemmed.

The frequency table, \( \text{Freq}_T \), associated with \( D \) and \( T \) is an \((M \times N)\) matrix such that \( \text{Freq}_T(i,j) \) equals the number of occurrences of the word \( t_i \) in document \( d_j \).
Example

<table>
<thead>
<tr>
<th>DocumentID</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d_1$</td>
<td>Jose Orojuelo’s Operations in Bosnia.</td>
</tr>
<tr>
<td>$d_2$</td>
<td>The Medelin Cartel’s Financial Organization.</td>
</tr>
<tr>
<td>$d_3$</td>
<td>The Cali Cartel’s Distribution Network.</td>
</tr>
<tr>
<td>$d_4$</td>
<td>Banking Operations and Money Laundering.</td>
</tr>
<tr>
<td>$d_5$</td>
<td>Profile of Hector Gomez.</td>
</tr>
<tr>
<td>$d_6$</td>
<td>Connections between Terrorism and Asian Dope Operations.</td>
</tr>
<tr>
<td>$d_7$</td>
<td>Hector Gomez’s: How he Gave Agents the Slip in Cali.</td>
</tr>
<tr>
<td>$d_8$</td>
<td>Sex, Drugs, and Videotape.</td>
</tr>
<tr>
<td>$d_9$</td>
<td>The Iranian Connection.</td>
</tr>
<tr>
<td>$d_{10}$</td>
<td>Boating and Drugs: Slips owned by the Cali Cartel.</td>
</tr>
</tbody>
</table>

The associated frequency table might be given by:

<table>
<thead>
<tr>
<th>Term/Doc</th>
<th>$d_8$</th>
<th>$d_9$</th>
<th>$d_{10}$</th>
<th>$d_{11}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>sex</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>drug</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>videotape</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>iran</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>connection</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>boat</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>slip</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>own</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>cali</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>cartel</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Another example

Consider the frequency table shown below.

<table>
<thead>
<tr>
<th>Term/Doc</th>
<th>$d_1$</th>
<th>$d_2$</th>
<th>$d_3$</th>
<th>$d_4$</th>
<th>$d_5$</th>
<th>$d_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>615</td>
<td>390</td>
<td>10</td>
<td>10</td>
<td>18</td>
<td>65</td>
</tr>
<tr>
<td>$t_2$</td>
<td>15</td>
<td>4</td>
<td>76</td>
<td>217</td>
<td>91</td>
<td>816</td>
</tr>
<tr>
<td>$t_3$</td>
<td>2</td>
<td>8</td>
<td>815</td>
<td>142</td>
<td>765</td>
<td>1</td>
</tr>
<tr>
<td>$t_4$</td>
<td>312</td>
<td>511</td>
<td>677</td>
<td>11</td>
<td>711</td>
<td>2</td>
</tr>
<tr>
<td>$t_5$</td>
<td>45</td>
<td>33</td>
<td>516</td>
<td>64</td>
<td>491</td>
<td>59</td>
</tr>
</tbody>
</table>

- $d_1, d_2$ are similar because the distribution of the words in $d_1$ “mirrors” the distribution of the words for $d_2$.
- Both contain lots of occurrences of $t_1, t_4$ and relatively few occurrences of $t_2, t_3$, and moderately many occurrences of $t_5$.
- $d_3$ and $d_5$ are also similar.
- But $d_4, d_6$ stand out as sharply different.
- But is this enough?
- Merely counting words does not indicate the importance of the words, in the document. What about document lengths?
- Usually, a frequency table represents not just the number of occurrences of a (stemmed) word in a document, but also its important.
Queries

- User wants to execute the query: *Find the 25 documents that are maximally relevant w.r.t. banking operations and drugs?*

- Query $Q$ is trying to retrieve documents relevant to two keywords, which after “stemming” are:

  
  \begin{align*}
  \text{drug, bank}.
  \end{align*}

- Think of the query $Q$ as a document. Thus, $Q$ is a column vector.

- We want to find the columns in $\mathbf{FreqT}$ that are “as close” as possible to the vector associated with $Q$.

- Example closeness metrics include:

1. **Term Distance:** Suppose $vec_Q(i)$ denotes the number of occurrences of term $t_i$ in $Q$. Then the term distance between $Q$ and document $d_r$ is given by:

   \[ \sqrt{\sum_{j=1}^{M} (vec_Q(j) - \operatorname{FreqT}(j, r))^2}. \]

   Of course, this is a rather arbitrary metric.

2. **Cosine Distance:** This metric is used extensively in the document database world, and it may be described as follows:

   \[ \frac{\sum_{j=1}^{M} (vec_Q(j) \times \operatorname{FreqT}(j, r))}{\sqrt{\sum_{j=1}^{M} vec_Q(j)^2 \times \sqrt{\sum_{j=1}^{M} \operatorname{FreqT}(j, r)^2}}}. \]
• Complexity of retrievals may be $O(N \times M)$ which could be staggering in size.
Latent Semantic Indexing: The Basic Idea

- The number of documents $M$ and the number of terms $N$ is very large. For example, $N$ could be over 10,000,000, as English words as well as proper nouns can be indexed.

- What LSI tries to do is to find a relatively small subset of $K$ words which discriminate between the $M$ documents in the archive.

- LSI is claimed to work effectively for around $K = 200$.

- Advantage: Each document is now a column vector of length 200, instead of length $N$.

- This is obviously a big plus.

- Bottom line: How do we find a relatively small subset of $K$ words which discriminate between the $M$ documents in the archive.

- Use a technique called singular valued decomposition.

- 4-step approach used by LSI:
  1. **(Table Creation)** Create frequency matrix $\text{Freq}_T$.
  2. **(SVD Construction)** Compute the singular valued decompositions, $(A, S, B)$ of $\text{Freq}_T$.
  3. **(Vector Identification)** For each document $d$, let $\text{vec}(d)$ be the set of all terms in $\text{Freq}_T$ whose corresponding rows have not been eliminated in the singular matrix $S$. 
4. **(Index Creation)** Store the set of all $vec(d)$’s, indexed by any one of a number of techniques (later we will discuss one such technique called a TV-tree).
A matrix $\mathcal{M}$ is said to be of order $(m \times n)$ if it has $m$ rows and $n$ columns.

If $\mathcal{M}_1, \mathcal{M}_2$ are matrices of order $(m_1 \times n_1)$ and $(m_2 \times n_2)$ respectively, then we say that the product, $(\mathcal{M}_1 \times \mathcal{M}_2)$, is well defined iff $n_1 = m_2$.

If:

$$
\mathcal{M}_1 = \begin{pmatrix}
a^1_1 & a^1_2 & \cdots & a^1_{m_1} \\
a^2_1 & a^2_2 & \cdots & a^2_{m_1} \\
\vdots & \vdots & \ddots & \vdots \\
a^{n_1}_1 & a^{n_1}_2 & \cdots & a^{n_1}_{m_1}
\end{pmatrix}; \mathcal{M}_2 = \begin{pmatrix}
b^1_1 & b^1_2 & \cdots & b^1_{m_2} \\
b^2_1 & b^2_2 & \cdots & b^2_{m_2} \\
\vdots & \vdots & \ddots & \vdots \\
b^{n_2}_1 & b^{n_2}_2 & \cdots & b^{n_2}_{m_2}
\end{pmatrix}
$$

then the product, $(\mathcal{M}_1 \times \mathcal{M}_2)$ is the matrix

$$
(\mathcal{M}_1 \times \mathcal{M}_2) = \begin{pmatrix}
c^1_1 & c^1_2 & \cdots & c^1_{m_1} \\
c^2_1 & c^2_2 & \cdots & c^2_{m_1} \\
\vdots & \vdots & \ddots & \vdots \\
c^{n_2}_1 & c^{n_2}_2 & \cdots & c^{n_2}_{m_1}
\end{pmatrix}
$$

where:

$$
c^i_j = \sum_{r=1}^{n_1} (a^i_r \times b^r_j).
$$

EX:

$$
\begin{pmatrix} 3 & 2 \\ 4 & 8 \end{pmatrix} \times \begin{pmatrix} 1 & 4 & 3 \\ 2 & 4 & 6 \end{pmatrix} = \begin{pmatrix} 7 & 20 & 21 \\ 20 & 48 & 60 \end{pmatrix}.
$$
SVD Continued

- Given a matrix $\mathcal{M}$ of order $(m \times n)$, the transpose of $\mathcal{M}$, denoted $\mathcal{M}^T$, is obtained by converting each row of $\mathcal{M}$ into a column of $\mathcal{M}^T$.

- EX:
  \[
  \begin{pmatrix}
  7 & 20 & 21 \\
  20 & 48 & 60 \\
  \end{pmatrix}^T = \begin{pmatrix}
  7 & 20 \\
  20 & 48 \\
  21 & 60 \\
  \end{pmatrix}.
  \]

- Vector = matrix of order $(1 \times m)$.

- Two vectors $x, y$ of the same order are said to be orthogonal iff $x^T y = 0$.

- EX:
  \[
  x = (10, 5, 20). \\
  y = (1, 2, -1).
  \]

These two vectors are orthogonal because

\[
  x^T y = \begin{pmatrix} 10 \\ 5 \\ 20 \end{pmatrix} \times \begin{pmatrix} 1 & 2 & -1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}.
\]
SVD Continued

- Matrix $\mathcal{M}$ is said to be **orthogonal** iff $(\mathcal{M}^T \times \mathcal{M})$ is the identity matrix (i.e. the matrix, all of whose entries are 1). For example, consider the matrix:

$$\mathcal{M} = \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}.$$ 

This matrix is orthogonal.

- Matrix $\mathcal{M}$ is said to be a **diagonal matrix** iff the order of $\mathcal{M}$ is $(m \times m)$ and for all $1 \leq i, j \leq m$, it is the case that:

$$i \neq j \rightarrow \mathcal{M}(i, j) = 0.$$ 

- **EX**: $A$ and $B$ below are diagonal matrices, but $C$ is not:

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 5 \end{pmatrix}; \quad B = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}; \quad C = \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}.$$ 

- Diagonal matrix $\mathcal{M}$ of order $(m \times m)$ is said to be **non-decreasing** iff for all $1 \leq i, j \leq m$,

$$i \leq j \rightarrow \mathcal{M}(i, i) \leq \mathcal{M}(j, j).$$ 

- Above, $A$ is a non-decreasing diagonal matrix, but $B$ is not.
SVD Continued

- A *Singular Value Decomposition* of $\text{Freq} T$ is a triple $(A, S, B)$ where:

  1. $\text{Freq} T = (A \times S \times B^T)$ and
  2. $A$ is an $(M \times M)$ orthogonal matrix such that $A^T A = I$ and
  3. $B$ is an $(N \times N)$ orthogonal matrix such that $B^T B = I$ and
  4. $S$ is a diagonal matrix called a *singular matrix*.

- Theorem: Given any matrix $\mathcal{M}$ of order $(m \times n)$, it is possible to find a singular value decomposition, $(A, S, B)$ of $\mathcal{M}$ such that $S$ is a *non-decreasing* diagonal matrix.

- EX: The SVD of the matrix

  \[
  \begin{pmatrix}
  1.44 & 0.52 \\
  0.92 & 1.44 
  \end{pmatrix}
  \]

  is given by:

  \[
  \begin{pmatrix}
  0.6 & -0.8 \\
  0.8 & 0.6 
  \end{pmatrix}
  \begin{pmatrix}
  5 & 0 \\
  0 & 2 
  \end{pmatrix}
  \begin{pmatrix}
  0.8 & 0.6 \\
  0.6 & -0.8 
  \end{pmatrix}.
  \]

  Here, the singular values are 5 and 2, and it is easy to see that the singular matrix is non-decreasing.
Returning to LSI

- Given a frequency matrix $\text{FreqT}$, we can decompose it into an SVD $\mathbf{TSD}^T$ where $\mathbf{S}$ is non-decreasing.
- If $\text{FreqT}$ is of size $(M \times N)$, then $\mathbf{T}$ is of size $(M \times M)$ and $\mathbf{S}$ is of order $(M \times R)$ where $R$ is the rank of $\text{FreqT}$, and $\mathbf{D}^T$ is of order $(R \times N)$.
- We can now “shrink” the problem substantially by eliminating the least significant singular values from the singular matrix $\mathbf{S}$.

![Diagram showing the decomposition $\text{FreqT} = \mathbf{T} \times \mathbf{S} \times \mathbf{D}^T$.](attachment:diagram.png)
LSI Continued

Shrinking the matrices is done as follows.

- Choose an integer $k$ that is substantially smaller than $R$.
- Replace $S$ by $S^*$, which is a $(k \times k)$ matrix, such that $S^*(i, j) = S(i, j)$ for $1 \leq i, j \leq k$.
- Replace the $(R \times N)$ matrix $D^T$ by the $(k \times N)$ matrix $D^{*T}$ where: $D^{*T}(i, j) = D^T(i, j)$ if $1 \leq i \leq k$ and $1 \leq j \leq N$.

Bottom line:

- Throw away the least significant values, and retain the rest of the matrix involved.
- Key claim in LSI Is that if $k$ is chosen judiciously, then the $k$ rows appearing in the singular matrix $S^*$ represent the $k$ “most important” (from the point of view of retrieval) terms occurring in the entire document collection.
Example

Suppose $\text{FreqT}$ has the SVD

\[
\begin{pmatrix}
  a_1^1 & a_2^1 & a_3^1 & a_4^1 & a_5^1 \\
  a_1^2 & a_2^2 & a_3^2 & a_4^2 & a_5^2 \\
  \cdots & \cdots & \cdots & \cdots & \cdots \\
  a_1^M & a_2^M & a_3^M & a_4^M & a_5^M
\end{pmatrix}
\begin{pmatrix}
  20 & 0 & 0 & 0 & 0 \\
  0 & 16 & 0 & 0 & 0 \\
  0 & 0 & 12 & 0 & 0 \\
  0 & 0 & 0 & 0.08 & 0 \\
  0 & 0 & 0 & 0 & 0.004
\end{pmatrix}
\begin{pmatrix}
  b_1^1 & b_2^1 & b_3^1 & \cdots & b_N^1 \\
  b_1^2 & b_2^2 & b_3^2 & \cdots & b_N^2 \\
  \cdots & \cdots & \cdots & \cdots & \cdots \\
  b_1^5 & b_2^5 & b_3^5 & \cdots & b_N^5
\end{pmatrix}
\]

- If we set 3 as the threshold, then we obtain the following result (after eliminating the 4'th and 5'th singular values).

\[
\begin{pmatrix}
  a_1^1 & a_2^1 & a_3^1 \\
  a_1^2 & a_2^2 & a_3^2 \\
  \cdots & \cdots & \cdots \\
  a_1^M & a_2^M & a_3^M
\end{pmatrix}
\begin{pmatrix}
  20 & 0 & 0 \\
  0 & 16 & 0 \\
  0 & 0 & 12
\end{pmatrix}
\begin{pmatrix}
  b_1^1 & b_2^1 & b_3^1 & \cdots & b_N^1 \\
  b_1^2 & b_2^2 & b_3^2 & \cdots & b_N^2 \\
  b_1^3 & b_2^3 & b_3^3 & \cdots & b_N^3
\end{pmatrix}
\]
Analysis

- Usually, $R$ is taken to be 200.
- The size of the original frequency table is $(M \times N)$ where $M$ is the number of terms, and $N$ is the number of documents. We may easily have $M = 1,000,000$ and $N = 10,000$, even for just a small document database such as that consisting of the University of Maryland’s Computer Science technical reports.
- The size of the three matrices, after we have reduced the size of the singular matrix to, say 200, is:
  - The first matrix’s size is $M \times R$. With the above numbers, this is $1000000 \times 200 = 200,000,000$.
  - The singular matrix’s size is $200 \times 200 = 400,000$. (In fact, of these 400,000 entries, only 200 at most need to be stored, as all other entries are zero).
  - The last matrix’s size is $R \times N$. With the above numbers, this is $200 \times 10000$.

Adding up the above, we get a total of 202,000,200 entries in the tables after SVD’s are applied. This is approximately 200 million.

- In contrast, $(M \times N)$ is close to 10,000-million: in other words, the SVD trick reduced the space utilized to about $\frac{1}{50}$th of that required by the original frequency table.
LSI: Document Retrieval using SVDs

Two questions:

1. Given two documents $d_1, d_2$ in the archive, how “similar” are they?
2. Given a query string/document $Q$, what are the $n$ documents in the archive that are “most relevant” for the query?

- Suppose $\mathbf{x} = (x_1, \ldots, x_w)$ and $\mathbf{y} = (y_1, \ldots, y_w)$.
- Dot product of $\mathbf{x}$ and $\mathbf{y}$, denoted $\mathbf{x} \odot \mathbf{y}$ is given by
  \[
  \mathbf{x} \odot \mathbf{y} = \sum_{i=1}^{w} x_i \times y_i.
  \]
- The similarity of these two documents w.r.t. the SVD representation, $TS^* \times D^{*T}$, of a frequency table is given by computing the dot product of the two columns in the matrix $D^{*T}$ associated with these two documents.
  \[
  \sum_{z=1}^{R} D^{*T}[i, z] \times D^{*T}[j, z].
  \]
- When we are asked to find the top $p$ matches for $Q$, we are trying to find $p$ documents $d_{\alpha(1)}, \ldots, d_{\alpha(p)}$ such that:
  1. for all $1 \leq i \leq j \leq p$, the similarity between $\text{vec}_Q$ and $d_{\alpha(i)}$ is greater than or equal to the similarity between $\text{vec}_Q$ and $d_{\alpha(j)}$. 
2. There is no other document $d_z$ such that the similarity between $d_z$ and $vec_Q$ exceeds that of $d_{\alpha(p)}$.

- This can be done by using any generalized, high dimensional data structure that supports nearest neighbor searches.
Telescopic Vector (TV) Trees

- Access to point data in very large dimensional spaces should be highly efficient.
- A document \( d \) may be viewed as a vector \( \vec{d} \) of length \( k \), where the singular valued matrix, after decomposition, is of size \((k \times k)\).
- Thus, each document may be thought of as a point in a \( k \)-dimensional space.
- A document database may be thought of as a collection of such points, indexed appropriately.
- When a user \( u \) presents a query \( Q \), s/he is in effect specifying, a vector \( \text{vec}(Q) \) of length \( k \). We must find the \( p \) documents in the database that are maximally relevant to \( Q \).
- This boils down to attempting to find the \( k \)-nearest neighbors present in the document database, of the query \( Q \).
- The TV-tree is a data structure that borrows from R-trees in this effort.
- The TV-tree attempts to dynamically and flexibly decide how to branch, based on the data that is being examined. If lots of vectors all agree on certain attributes (e.g. if lots of documents all have many common terms), then we must organize our index by branching on those terms (i.e. fields of the vectors) that distinguish between these vectors/documents.
Organization of a TV-Tree

- **NumChild**: This is the maximal number of children that any node in the TV-tree is allowed to have.

- **α**: \( \alpha \) is a number, greater than 0 and less than \( k \), called the number of active dimensions.

- **TV(\( k \), NumChild, \( \alpha \))** denotes TV-tree used to store \( k \)-dimensional data with NumChild as the maximal number of children, and \( \alpha \) as the number of active dimensions.

- Each node in a TV-tree represents a region. For this purpose, each node \( N \) in a TV-tree contains three fields:
  1. \( N.Center \): This represents a point in \( k \)-dimensional space.
  2. \( N.Radius \): This is a real number greater than 0.
  3. \( N.ActiveDims \): This is a list of at most \( \alpha \) dimensions. Each of these dimensions is a number between 1 and \( k \). Thus, \( N.ActiveDims \) is a subset of \( \{1, \ldots, k\} \) of cardinality \( \alpha \) or less,
Region associated with a node $N$

- Suppose $x$ and $y$ are points in $k$-dimensional space, and $ActiveDims$ is some set of active dimensions. The active distance between $x$ and $y$, denoted $\text{act}_\text{dist}(x, y)$ is given by:

$$\text{act}_\text{dist}(x, y) = \sqrt{\sum_{i \in ActiveDims} x_i^2 - y_i^2}.$$ 

Here, $x_i$ and $y_i$ denote the value of the $i$'th dimension of $x$ and $y$, respectively.

- EX: $k = 200$ and $\alpha = 5$ and the set $ActiveDims = \{1, 2, 3, 4, 5\}$. Suppose:

$$x = (10, 5, 11, 13, 7, x_6, x_7, \ldots, x_{200})$$
$$y = (2, 4, 14, 8, 6, y_6, y_7, \ldots, y_{200})$$

Then the active distance between $x$ and $y$ is given by:

$$\text{act}_\text{dist}(x, y) = \sqrt{(10 - 2)^2 + (5 - 4)^2 + (11 - 14)^2 + (13 - 8)^2 + (7 -}$$

$$= \sqrt{100}$$
$$= 10.$$ 

- Node $N$ represents the region containing all points $x$ such that the active distance (w.r.t. the active dimensions in $N.ActiveDims$) between $x$ and $N.Center$ is less than or equal to $N.Radius$. 

---

*V.S. Subrahmanian  — All Rights Reserved*

These slides may not be duplicated without explicit written permission from Morgan Kaufmann Press.

Principles of Multimedia Database Systems  Morgan Kaufmann  Copyright ©1997
- EX: If we had a node \( N \) with its center at 

\[
N.Center = (10, 5, 11, 13, 7, 0, 0, 0, \ldots, 0)
\]

and \( N.ActiveDims = \{1, 2, 3, 4, 5\} \), then this node represents the region consisting of all points \( \mathbf{x} \) such that:

\[
\sqrt{(x_1 - 10)^2 + (x_2 - 5)^2 + (x - 11)^2 + (x_4 - 13)^2 + (x_5 - 7)^2} \leq N.Ra
\]

We use the notation \( Region(N) \) to denote the region represented by a node \( N \) in a TV-tree.

- \( N \) also contains an array, \( \text{Child} \) of \( \text{NumChild} \) pointers to other nodes of the same type.
Properties of TV-Trees

- All data is stored at the leaf nodes;
- Each node in a TV-tree (except for the root and the leaves) must be at least half full, i.e. at least half the Child pointers must be non-NIL.
- If $N$ is a node, and $N_1, \ldots, N_r$ are all its children, then

$$Region(N) = \bigcup_{i=1}^{r} Region(N_i).$$
There are three key steps.

1. **Branch Selection**: The first operation is called branch selection. When we insert a new vector into the TV-tree, and we are at node $N$ (with children $N_i$, for $1 \leq i \leq \text{NumChild}$), we need to determine which of these children to insert the key into.

2. **Splitting**: The second approach is what to do, when we are at a leaf node that is full and cannot accommodate the vector $v$ we are inserting. This causes a split in that node.

3. **Telescoping**: Suppose a node $N$ is split into subnodes $N_1, N_2$. In this case, it may well turn out that the vectors in $\text{Region}(N_1)$ all agree on not just the active dimensions of the parent $N$, but a few more as well. The addition of these extra dimensions is called *telescoping*. Telescoping may also involve the *removal* of some active dimensions, as we shall see later.
Example

- 5-dimensional space.
- $TV(5, 3, 2)$.
- Space is a sphere centered at $(0, 0, 0, 0, 0)$ with radius 50.
- Initially, tree is empty.
- Insert $(5, 3, 20, 1, 5)$. This is handled straightforwardly by the creation of a root node with
  1. $Root.Center = (0, 0, 0, 0, 0)$;
  3. In this case, the root is also a leaf, with a pointer to the information relevant to the point $v_1 = (5, 3, 20, 1, 5)$.
  4. Suppose $Root.ActiveDims = \{2, 3\}$.
   
   See (a).
- Insert $v_2 = (0, 0, 18, 42, 4)$.
- Insert $v_3 = (0, 0, 19, 39, 6)$. At this stage, the root is “full”.
- Insert $v_4 = (9, 10, 2, 0, 16)$.
  1. We must split the root.
  2. Take the four vectors involved and “group” them together into two groups, say. $v_1, v_4$ and $v_2, v_3$. See figure (d).
  3. Insert $v_5 = (18, 5, 27, 9, 9)$. Branch selection needed to determine how to branch. See Figure 2(a).
4. Insert $v_6 = (0, 0, 29, 0, 3)$. Again, we must perform branch selection, and this time, we may choose to branch right, as shown in Figure 2(b).
Figure 1
Figure 2
Branch Selection

- Consider node $N$ with $1 \leq j \leq \text{NumChild}$ children, denoted $N_1, \ldots, N_{\text{NumChild}}$.

- $\text{exp}_j(v)$ denotes the amount we must expand $N_j.\text{Radius}$ so that $v$'s active distance from $N_j.\text{Center}$ falls within this radius.

- First, select all $j$'s such that $\text{exp}_j(v)$ is minimized.
  EX: if we have nodes $N_1, \ldots, N_5$ with $\text{exp}$ values 10, 40, 19, 10, 32 respectively, the two candidates selected for possible insertion are $N_1$ and $N_4$. If a tie occurs, as in the above case, pick the node such that the distance from the center of that node to $v$ is minimized.
Splitting

- When we attempt to insert a vector $v$ into a leaf node $N$ that is already full, then we need to split the node.
- We must create subnodes $N_1, N_2$, and each vector in node $N$ must fall into one of the regions represented by these two subnodes.
- Split vectors in leaf $N$ into two groups $(G_1, H_1)$.
- We may be able to enclose all vectors in $G_1$ within a region with center $c_1$ and radius $r_1$ and all vectors in $H_1$ within a region with center $c_2$ and radius $r_2$.
- Many such splits are possible in general.
- Split $(G_1, H_1)$ is finer than split $(G_2, H_2)$ iff the sums of the radii, $(r_1 + c'_1)$ is smaller than the sum of the radii $(r_2 + r'_2)$.
- Still not enough to uniquely identify a “finest” split.
- If $(G_1, H_1)$ and $(G_2, H_2)$ are splits such that neither is finer than the other and no other split is finer than each of them, then we say that $(G_1, H_1)$ is more conservative than $(G_2, H_2)$ iff
  \[ r_1 + r'_1 - \text{act\_dist}(c_1, c'_1) \leq r_2 + r'_2 - \text{act\_dist}(c_2, c'_2). \]
- Split $(G, H)$ is the selected split iff:
  1. there is no split $(G', H')$ that is finer than $(G, H)$ and
2. there is no split \((G', H')\) that satisfies condition (1) (i.e. there exists no split \((G^*, H^*)\) finer than \((G', H')\)) and that is more conservative than \((G, H)\).
Telescoping

- Suppose $N$ is the node into which we are to insert a vector $v$.

- Insertion of $v$ may cause two types of changes to $N$: it may cause $N$ to be split into two subnodes $N_1, N_2$, or it may "modify" the set of active dimensions of node $N$ (e.g. if vector $v$ does not agree, on the active dimensions, with other vectors stored at node $N$).

- When node $N$ gets split into two sub-nodes $N_1, N_2$, the set of vectors at either node $N_1$ or node $N_2$ (but not both) must be a subset of the set of vectors at node $N$ before the insertion.

- Suppose $N_1$ has this property.

- The vectors in $N_1$ may agree not only on the active dimensions of $N$, but on some other dimensions as well. In this case, we can expand the set of active dimensions of node $N$, by adding these new dimensions. See figure below.

- The other case when telescoping occurs is when a vector is added to a node, $N$, but no split occurs. If $N$ originally
contained the vectors \( \mathbf{v}_1, \ldots, \mathbf{v}_r \) and \( \mathbf{v} \) is the vector being added, even though vectors \( \mathbf{v}_1, \ldots, \mathbf{v}_r \) originally agreed on the active dimensions \( d_1, \ldots, d_s \) of node \( N \), they only agree now on a subset (for example, \( d_2, \ldots, d_s \)) and hence, the set of active dimensions of node \( N \) must be \textit{contracted} to reflect this fact.
Searching in TV-Trees

Algorithm 2 \textit{Search}(T, v)

\begin{algorithm}
\begin{tabular}{l}
\textbf{if} \textit{Leaf}(T) \textbf{then} \textbf{Return} \ (T.\textit{Center} = v); \textbf{Halt} \\
\textbf{else} \\
\quad \{ \textbf{if} \ v \in \textit{Region}(T) \textbf{then} \\
\quad \quad \textbf{Return} \ \sqrt[N]{\text{NumChild}} \ \text{Search}(T.\text{Child}[i], v) \\
\quad \} \\
\end{tabular}
\end{algorithm}
Nearest Neighbor Retrievals in TV-Trees

\[
\begin{align*}
\min(N, v) &= \begin{cases} 
0 & \text{if } v \in \text{Region}(N) \\
\text{act}_\text{dist}(v, N.\text{Center}) - N.\text{Radius} & \text{otherwise}
\end{cases} \\
\max(N, v) &= \text{act}_\text{dist}(v, N.\text{Center}) + N.\text{Radius}.
\end{align*}
\]

- Maintain an array SOL of length \( p \), i.e. with indices running 1 through \( p \).
- The algorithm \( \text{NNSearch} \) uses a routine called \( \text{Insert} \) that takes as input, a vector \( \text{vec} \), and an array \( \text{SOL} \) maintained in non-descending order of active distance from \( \text{vec} \).
- \( \text{Insert} \) returns as output, the array \( \text{SOL} \) with \( \text{vec} \) inserted in it, at the right place, and with the \( p \)'th element of \( \text{SOL} \) eliminated.
Nearest Neighbor Retrievals in TV-Trees (Contd.)

Algorithm 3 \( \text{NNSearch}(T, v, p) \)

for \( i = 1 \) to \( p \) do \( \text{SOL}[i] = \infty; \)
\( \text{NNSearch1}(T, v, p); \)

end (* end of program NNSearch *)

procedure \( \text{NNSearch1}(T, v, p); \)

if \( \text{Leaf}(T) \& \text{act.dist}(T.val, v) < \text{SOL}[p] \) then
    Insert \( T.val \) into \( \text{SOL}; \)
else
    if \( \text{Leaf}(T) \) then \( r = 0; \)
    else \{ Let \( N_1, \ldots, N_r \) be the children of \( T; \)
        Order the \( N_i \)'s in ascending order w.r.t. \( \min(N_i, v) \);
        Let \( N_{\eta(1)}, \ldots, N_{\eta(r)} \) be the resulting order;
    \};
    done \( = \) false; \( i = 1; \)
    while \( ((i \leq r) \land \neg \text{done}) \) do
        \{ \( \text{NNSearch}(N_{\eta(i)}, v, p); \)
        if \( \text{SOL}[p] < \min(N_{\eta(i+1)}, v) \) then
            done \( = \) true;
            \( i = i + 1; \)
        \}; (* end of while *)
    \} (* end of else *)

Return \( \text{SOL}; \)
end proc (* end of subroutine NNSearch1 *)
Other Retrieval Techniques: Inverted Indices

- A document record contains two fields – a doc_id and a postings_list.
- Postings list is a list of terms (or pointers to terms) that occur in the document. Sorted using a suitable relevance measure.
- A term record consists of two similar fields: a term field (string), and a postings_list.
- Two hash tables are maintained: a DocTable and a TermTable. The DocTable is constructed by hashing on the doc_id key, while the TermTable is obtained by hashing on the term key.
- To find all documents associated with a term, we merely return the postings list.
- Used in many commercial systems such as MEDLARS and DIALOG.
Other Retrieval Techniques: Signature Files

• Associate a signature $s_d$, with each document $d$.

• A signature is a representation of an ordered list of terms that describe the document.

• The list of terms from which $s_d$ is derived is obtained after performing a frequency analysis, stemming, and usage of stop lists.

• If signature list consists of the ordered list of words $w_1, w_2, \ldots, w_r$.

• This means that word $w_1$ is most important when describing the document, word $w_2$ is second most important, and so forth.

• Signature of the document $d$ is a bit-representation of this list, usually obtained by encoding the list after using hashing, and then superimposing a coding scheme.
Video Databases
Introduction

Retrieval requests may take one of two forms.

- **Retrieving a Specified Video:** User specifies the video he wants to see, e.g. “Show me *The Sound of Music*”.

- **Identifying and Retrieving Video Segments:** User might express a query such as *Find all videos in which John Wayne appears with a gun*. This query requires that we:
  - identify the movies in which John Wayne appears with a gun and
  - identify the segments within those movies in which John Wayne appears with a gun.

- Once we can organize the content of a single video, we can organize the content of a set of videos.
Organizing Content of a Single Video

We must ask ourselves the following questions:

1. Which aspects of the video are likely to be of interest to the users who access the video archive?
2. How can these aspects of the video be stored efficiently, so as to minimize the time needed to answer user queries?
3. What should Query Languages for video data look like and how should the relational model of data be extended to handle video information?
4. Can the Content Extraction process be automated, and if so, how can the reliability of such content extraction techniques be taken into account when processing queries?
Video Content: Which Aspects of a Video To Store?

Example: An 8-hour, one day lecture of a short course given by a professor on the topic *Multimedia Databases*. In this case, the video contains a set of “items of interest.” These items of interest could include:

1. *People* such as the professor, any guest lecturer (or lecturers) who speak at selected times in the course, and any students who might ask questions and/or distinguish themselves in other ways; For instance, *Prof. Felix* might be one such person, while *Erica* might be a student.

2. *Activities* that occur in the class such as *lecturing* (on a particular topic, by a particular individual), or *questioning* (by a particular student), or *answering* a question posed by a particular student. Other activities could involve general group discussions, and/or coffee breaks.

In addition, activities have attributes, e.g. *lecturing*(quadtrees, *Prof. Felix*) indicates an activity involving *Prof. Felix* lecturing on quadtrees, and *questioning*(Erica, *Prof. Felix*) indicating that *Prof. Felix* was questioned by *Erica.*
Movie Example

• Consider the movie, *Sound of Music*.

• Items of interest include:

  1. *People* such as Maria, Count Von Trapp, and others;
  2. *Inanimate objects* such as the piano in Count Von Trapp’s house;
  3. *Animate objects* such as the ducks and birds in the pond;
  4. *Activities* such as singing and dancing, with their associated list of attributes. For example, the activity *singing* may have two attributes:

     (a) **Singer** specifying which person is singing and
     (b) **Song** specifying the name of the song and

• Certain common characteristics occur. Given any frame \( f \) in the video, the frame \( f \) has a set of associated objects and associated activities.

• Objects/activities have certain properties, and these properties may vary from one frame to another.

• **Creating a video database means we should be able to index all these associations.**
Properties

- **Property**: Consists of a pair \((\text{pname}, \text{Values})\) where:
  - \(\text{pname}\) is the *Name* of the property,
  - \(\text{Values}\) is a set.

- **Property Instance**: An expression of the form \(\text{pname} \equiv v\) where \(v \in \text{Values}\).

- Example properties:
  1. \((\text{height}, R^+)\) consists of the “height” property with real-values;
  2. \((\text{primarycolors}, \{\text{red, green, blue}\})\) consists of a property called *primarycolors* with values red, green, blue.
Object Scheme

- **Object Scheme**: A pair \((fd, fi)\) where:
  1. \(fd\) is a set of *frame-dependent* properties;
  2. \(fi\) is a set of *frame-independent* properties.
  3. \(fi\) and \(fd\) are disjoint sets.

\(fd\) and \(fi\) are disjoint.

- If \((pname, Values)\) is a property in \(fd\), then this means that the property named \(pname\) may assume different instances, depending upon the video frame being considered. E.g. the property *shirtcolor* varies from frame to frame.

- **Object Instance**: An *Object Instance* is a triple \((oid, os, ip)\) where:
  1. \(oid\) is a string called the object-id and
  2. \(os = (fd, fi)\) is an object structure and
  3. \(ip\) is a set of statements such that:
    (a) for each property \((pname, Values)\) in \(fi\), \(ip\) contains *at most* one property instance of \((pname, Values)\) and;
    (b) for each property \((pname, Values)\) in \(fd\), and each frame \(f\) of the video, \(ip\) contains *at most* one property instance of \((pname, Values)\) – this property instance is denoted by the expression \(pname = v\) IN \(f\).
Example

- Surveillance video of 5 frames.
- Show surveillance video of the house of Denis Dopeman.
- Frame 1: We see Jane Shady at the path leading to Mr. Dopeman’s door. She is carrying a briefcase.
- Frame 2: She is halfway on the path to the door. Door opens. Mr. Dopeman appears at the door.
- Frame 3: Jane Shady and Denis Dopeman are ext to each other at the door; Jane Shady is still carrying the briefcase.
- Frame 4: Jane Shady is walking back, and Denis Dopeman has the brief case.
- Frame 5: Jane Shady is at the beginning of the path to Denis Dopeman’s door. Door is shut.
Example, contd.
Frame-dependent properties

<table>
<thead>
<tr>
<th>Frame</th>
<th>Objects</th>
<th>Frame-dependent properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Jane Shady</td>
<td>has(briefcase), at(path_front)</td>
</tr>
<tr>
<td></td>
<td>dopeman_house</td>
<td>door(closed)</td>
</tr>
<tr>
<td></td>
<td>Briefcase</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Jane Shady</td>
<td>has(briefcase), at(path_middle)</td>
</tr>
<tr>
<td></td>
<td>Denis Dopeman</td>
<td>at(door)</td>
</tr>
<tr>
<td></td>
<td>dopeman_house</td>
<td>door(open)</td>
</tr>
<tr>
<td></td>
<td>Briefcase</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Jane Shady</td>
<td>has(briefcase), at(door)</td>
</tr>
<tr>
<td></td>
<td>Denis Dopeman</td>
<td>at(door)</td>
</tr>
<tr>
<td></td>
<td>dopeman_house</td>
<td>door(open)</td>
</tr>
<tr>
<td></td>
<td>Briefcase</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Jane Shady</td>
<td>at(door)</td>
</tr>
<tr>
<td></td>
<td>Denis Dopeman</td>
<td>has(briefcase), at(door)</td>
</tr>
<tr>
<td></td>
<td>dopeman_house</td>
<td>door(open)</td>
</tr>
<tr>
<td></td>
<td>Briefcase</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Jane Shady</td>
<td>at(path_middle)</td>
</tr>
<tr>
<td></td>
<td>dopeman_house</td>
<td>door(closed)</td>
</tr>
<tr>
<td></td>
<td>Briefcase</td>
<td></td>
</tr>
</tbody>
</table>
# Frame independent properties

<table>
<thead>
<tr>
<th>Object</th>
<th>Frame-independent property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jane Shady</td>
<td>age</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>height</td>
<td>170 (cms)</td>
</tr>
<tr>
<td>dopeman_house</td>
<td>address</td>
<td>6717 Pimmit Drive Falls Church, VA 22047.</td>
</tr>
<tr>
<td></td>
<td>type</td>
<td>brick</td>
</tr>
<tr>
<td></td>
<td>color</td>
<td>brown</td>
</tr>
<tr>
<td>Denis Dopeman</td>
<td>age</td>
<td>56</td>
</tr>
<tr>
<td></td>
<td>height</td>
<td>186</td>
</tr>
<tr>
<td>briefcase</td>
<td>color</td>
<td>black</td>
</tr>
<tr>
<td></td>
<td>length</td>
<td>40 (cms)</td>
</tr>
<tr>
<td></td>
<td>width</td>
<td>31 (cms)</td>
</tr>
</tbody>
</table>
Activity Schema

- An *Activity Scheme*, `ACT_SCH`, is a finite set of properties such that if `(pname, Values_1)` and `(pname, Values_2)` are both in `ACT_SCH`, then `Values_1 = Values_2`.

- **Example:** Consider the activity `ExchangeObject` such as the exchange of objects between Jane Shady and Denis Dopeman. his activity has the three-pair scheme consisting of the pairs:

  1. (Giver, Person): This pair specifies that the activity `ExchangeObject` has a property called `Giver` specifying who is transferring the object in question. This says that the property `Giver` is of type `Person`. `Person` is the set of all persons.

  2. (Receiver, Person) This pair specifies that the activity `ExchangeObject` has a property called `Receiver` specifying who is receiving the object in question.

  3. (Item, Thing): This pair specifies the item being exchanged. `Thing` is the set of all “exchange-able” items.

Thus, the exchange of the briefcase that occurred between Jane Shady and Denis Dopeman can be captured as an activity scheme with `Giver = Jane Shady`, `Receiver = Denis Dopeman`, and `Item = briefcase`. 
Activity/Event

- An *Activity* is a pair:
  1. **AclID**: the “name” of the activity of scheme **ACT_SCH** and
  2. for each pair \((pname, Values) \in ACT_SCH\), an equation of the form \(pname = v\) where \(v \in Values\).

- Any activity has an associated activity scheme, and each property of the activity has an associated value from its set of possible values.

- **Example:**

  1. The activity **Lecturing** may have the scheme
     
     \[
     \{(\text{Lecturer, Person}), (\text{Topic, String})\}
     \]
     
     and may contain the equations:
     
     \[
     \text{Lecturer} = \text{Prof. Felix}.
     \]
     \[
     \text{Topic} = \text{Video Databases}.
     \]

  2. Likewise, the activity **Questioning** may have the scheme
     
     \[
     \{(\text{Questioner, Person}), (\text{Questionee, Person}),
     (\text{Question, String}), (\text{Answer, String})\}
     \]
     
     and may contain the equations:
     
     \[
     \text{Questioner} = \text{Erica}.
     \]
     \[
     \text{Questionee} = \text{Prof. Felix}.
     \]
     \[
     \text{Question} = \text{How many children does a quadtree node have?}
     \]
     \[
     \text{Answer} = \text{At most 4.}
     \]
Video Content

• Suppose \( v \) is a video.

• Let \( \text{framenum}(v) \) specify the total number of frames of video \( v \).

• The content of \( v \) consists of a triple \((\text{OBJ}, \text{AC}, \lambda)\) where:
  1. \( \text{OBJ} = \{\text{oid}_1, \ldots, \text{oid}_n\} \) is a finite set of object instances;
  2. \( \text{AC} = \{\text{AcID}_1, \ldots, \text{AcID}_k\} \) is a finite set of activities/events and
  3. \( \lambda \) is a map from \( \{1, \ldots, \text{framenum}(v)\} \) to \( 2^{\text{OBJ} \cup \text{AC}} \).

• Intuitively,
  1. \( \text{OBJ} \) represents the set of objects of interest in the video and
  2. \( \text{AC} \) represents the set of activities of interest in the video and
  3. \( \lambda \) tells us which objects and which activities are associated with any given frame \( f \) of the video.

• Though this definition assumes that \( \lambda \) will be specified on a frame by frame basis, this is not required, as we will see later.
Video Library

- A video library, VidLib, consists of a finite set of 5-tuples (Vid_Id, VidContent, framenum, plm, R) where:
  1. Vid_Id is the Name of the video and
  2. VidContent is the Content of the video and
  3. framenum is the number of frames in the video and
  4. plm is a placement mapping that specifies the address of different parts of the video.
  5. R is a set of relations about videos “as a whole”.

---

V.S. Subrahmanian — All Rights Reserved

These slides may not be duplicated without explicit written permission from Morgan Kaufmann Press.

Principles of Multimedia Database Systems Morgan Kaufmann Copyright ©1997
## Organization of a simple video library

<table>
<thead>
<tr>
<th>Vid Content</th>
<th>Vid_Id</th>
<th>framenum</th>
<th>Relations</th>
<th>plm</th>
</tr>
</thead>
<tbody>
<tr>
<td>vid1.mpg</td>
<td>9999</td>
<td></td>
<td>date, place</td>
<td></td>
</tr>
<tr>
<td>vid2.mpg</td>
<td>4000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>vid3.mpg</td>
<td>16000</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Query Languages for Video Data

Querying video involves the following types of queries.

- **Segment Retrievals:** Find all segments, from one or more videos in the library, that satisfy a given condition.

- **Object Retrievals:** Given a video $v$ and a segment $[s, e]$ (start frame through end frame) of the video, find all objects that occurred in:
  
  - all frames between $s$ and $e$ (inclusive),
  - some frame between $s$ and $e$ (inclusive).

- **Activity Retrievals:** Given a video $v$ and a segment $[s, e]$ (start frame through end frame) of the video, find all activities occurred in:
  
  - all frames between $s$ and $e$ (inclusive),
  - some frame between $s$ and $e$ (inclusive).

- **Property-based Retrievals:** Find all videos, and video segments in which objects/activities with certain properties occur.
Video Functions

- **FindVideoWithObject(o):** Given the name of a data object \( o \), this function returns as output, a set of triples of the form:

\[
(\text{VideoId}, \text{Startframe}, \text{EndFrame})
\]

such that if \((v, s, e)\) is a triple returned in the output, then video \( v \)'s segment starting at frame \( s \) and ending at frame \( e \) has the object \( o \) in all frames between and including \( s, e \).

- **FindVideoWithActivity(a):** This does exactly the same as above, except that it returns all triples \((v, s, e)\) such that video \( v \)'s segment starting at frame \( s \) and ending at frame \( e \) has the activity \( a \) in it. For each property \( p \), the notation \( a.p \) specifies the value of that property.

- **FindVideoWithActivityandProp(a,p,z):** This does exactly the same as above, except that it returns all triples \((v, s, e)\) such that video \( v \)'s segment starting at frame \( s \) and ending at frame \( e \) has the activity \( a \) in it with \( z \) as the value of property \( p \).

- **FindVideoWithObjectandProp(o,p,z):** This does exactly the same as above, except that it returns all triples \((v, s, e)\) such that video \( v \)'s segment starting at frame \( s \) and ending at frame \( e \) has the object \( o \) in it with \( z \) as the value of property \( p \).
• **FindObjectsInVideo(v,s,e):** Given the name of a video, and a start and end frame, this returns all objects that appear in all segments of the video between \( s \) and \( e \) (inclusive).

• **FindActivitiesInVideo(v,s,e):** Identical to the above, except it applies to activities, not objects.

• **FindActivitiesAndPropsInVideo(v,s,e):** Given the name of a video, a start and end frame, this returns a set of records of the form

\[
\text{activityname} : \text{prop1} = \text{entity1}; \text{prop2} = \text{entity2}; \cdots
\]

\[
\text{propk} = \text{entityk}
\]

comprising all activities, and their associated roles, that occur in all times between \( s \) and \( e \) of video \( v \).

• **FindObjectsAndPropsInVideo(v,s,e):** Identical to the above, except that it applies to objects, not to activities.
Video Query Languages

• Standard SQL query has the form:

```
SELECT field1,...,fieldn
FROM relation1 (R1),
relation2 (R2),..., relationk (Rk)
WHERE Condition.
```

• Expand this so that:

1. The `SELECT` statement may contain entries of the form

   \[ \text{Vid.Id} : [s, e] \]

   denoting the selection of a video with id, `Vid.Id`, and with the relevant segment comprised of frames between \( s \) and \( e \) inclusive.

2. The `FROM` statement may contain entries of the form:

   \[ \text{video(source)(V)} \]

   which says that \( V \) is a variable ranging over videos from the source named.

3. The `WHERE` condition allows statements of the form

   \[ \text{term IN func-call} \]

   where:
(a) \textit{term} is either a variable or an object or an activity, 
or a property value and
(b) \textit{func\_call} is any of the eight video functions listed above.
Examples

- “Find all videos and their relevant segments from video library VidLib\textsubscript{1} that contain Denis Dopeman.”

\[
\text{SELECT} \quad \text{vid:}[s,e] \\
\text{FROM} \quad \text{video:VidLib\textsubscript{1}} \\
\text{WHERE} \quad (\text{vid}, s, e) \ \text{IN} \\
\quad \text{FindVideoWithObject}(\text{Denis Dopeman}).
\]

- “Find all videos and their relevant segments from video library VidLib\textsubscript{1} that contain Denis Dopeman and Jane Shady.”

\[
\text{SELECT} \quad \text{vid:}[s,e] \\
\text{FROM} \quad \text{video:VidLib\textsubscript{1}} \\
\text{WHERE} \quad (\text{vid}, s, e) \ \text{IN} \\
\quad \text{FindVideoWithObject}(\text{Denis Dopeman}) \ \text{AND} \\
\quad (\text{vid}, s, e) \ \text{IN} \\
\quad \text{FindVideoWithObject}(\text{Jane Shady}).
\]
Examples, Continued

- “Find all videos and their relevant segments from video library VidLib₁ that contain Denis Dopeman and Jane Shady exchanging a briefcase.”

```
SELECT vid:[s,e]
FROM video:VidLib₁
WHERE (vid,s,e) IN
    FindVideoWithObject(Denis Dopeman)
AND (vid,s,e) IN
    FindVideoWithObject(Jane Shady)
AND (vid,s,e) IN
    FindVideoWithActivityandProp
        (ExchangeObject, Item, Briefcase)
AND (vid,s,e) IN
    FindVideoWithActivityandProp
        (ExchangeObject, Giver, Jane Shady)
AND (vid,s,e) IN
    FindVideoWithActivityandProp
        (ExchangeObject, Receiver, Denis Dopeman)
```
Indexing Video Content

- Now that we have defined content, we need to index it.
- We have 8 types of video retrieval functions. Indexing must support efficient execution of these 8 function types.
- It is impossible to store video content on a frame by frame basis due to the fact that a single 90 minute video contains close to ten million frames.
- We need Compact Representations to store video content.
- Two such data structures:
  - Frame Segment Tree
  - R-Segment Tree
Frame Segment Trees

- **Frame-sequence** is a pair \([i, j]\) where \(1 \leq i \leq j \leq n\). \([i, j]\) represents the set of all frames between \(i\) (inclusive) and \(j\) (non-inclusive), i.e.

\[
[i, j] = \{k \mid i \leq k < j\}.
\]

- EX: \([6, 12]\) denotes the set of frames \(\{6, 7, 8, 9, 10, 11\}\).

- **Frame-sequence Ordering**: \([i_1, j_1]\) \(\subseteq [i_2, j_2]\) iff \(i_1 < j_1 \leq i_2 < j_2\).

- \([i_1, j_1]\) \(\subseteq [i_2, j_2]\) means that the sequence of frames denoted by \([i_1, j_1]\) precedes the sequence of frames denoted by \([i_2, j_2]\).

- EX: Cconsider frame-sequences \(f s_1 = [10, 15]\), \(f s_2 = [8, 10]\) and \(f s_3 = [11, 13]\).

\[
- f s_2 \subseteq f s_1 \\
- f s_2 \subseteq f s_3 \\
- f s_1 \nsubseteq f s_3.
\]

- **Well-Ordered Set of Frame-sequences**: A set, \(X\), of frame-sequences is said to be *well-ordered* iff:

1. \(X\) is finite, i.e. \(X = \{[i_1, j_1], \ldots, [i_r, j_r]\}\) for some integer \(r\), and
2. \([i_1, j_1] \subseteq [i_2, j_2] \subseteq \ldots \subseteq [i_r, j_r]\).

- EX: \(X = \{(1, 4), [9, 13), [33, 90]\}\) is a well-ordered set of frame-sequences because \([1, 4] \subseteq [9, 13] \subseteq [33, 90]\).
Frame Segment Trees, Continued

- **Solid Set of Frame-sequences:** A set, \( X \), of frame-sequences is said to be *solid* iff

1. \( X \) is well-ordered, and
2. there is no pair of frame-sequences in \( X \) of the form \([i_1, i_2]\) and \([i_2, i_3]\).

- Take \( X = \{[1, 5], [5, 7], [9, 11]\}\).
- \( X \) is not solid. Why?
- Take \( Y = \{[1, 7], [9, 11]\}\). This is solid.

- **Segment Association Map:** Suppose \((\text{OBJ}, \text{AC}, \lambda)\) represents the content of a video \(v\). A Segment Association Map \(\sigma_v\) associated with video \(v\) is the map defined as follows:

1. \(\sigma_v\)'s domain is \(\text{OBJ} \cup \text{AC}\) and
2. \(\sigma_v\) returns, for each \(x \in \text{OBJ} \cup \text{AC}\), a *solid* set of frame-sequence, denoted \(\sigma_v(x)\) such that:
   
   a) if \([s, e) \in \sigma_v(x)\), then for all \(s \leq f < e\), it is the case that \(x \in \lambda(f)\) and
   
   b) for all frames \(f\) and all \(x \in \text{OBJ} \cup \text{AC}\), if \(x \in \lambda(f)\), then there exists a frame-sequence \([s, e) \in \sigma_v(x)\) such that \(f \in [s, e)\).
An example of a video’s content
Example, continued

- 5000 frames in example.
- The table below shows how many frames each object appears in:

<table>
<thead>
<tr>
<th>Object</th>
<th>Number of frames</th>
</tr>
</thead>
<tbody>
<tr>
<td>object1</td>
<td>1250</td>
</tr>
<tr>
<td>object2</td>
<td>1500</td>
</tr>
<tr>
<td>object3</td>
<td>3250</td>
</tr>
<tr>
<td>object4</td>
<td>1000</td>
</tr>
<tr>
<td>object5</td>
<td>2750</td>
</tr>
</tbody>
</table>

- To explicitly represent the mapping $\lambda$ associated with the content of this video, we would need to have a total of 9750 tuples.

- Instead, represent information with 16 tuples as shown below.

  **Segment Table:**
<table>
<thead>
<tr>
<th>Object</th>
<th>Segment</th>
</tr>
</thead>
<tbody>
<tr>
<td>object1</td>
<td>250–750</td>
</tr>
<tr>
<td>object1</td>
<td>1750–2500</td>
</tr>
<tr>
<td>object2</td>
<td>250–1000</td>
</tr>
<tr>
<td>object2</td>
<td>2250–2500</td>
</tr>
<tr>
<td>object2</td>
<td>2750–3250</td>
</tr>
<tr>
<td>object3</td>
<td>0–250</td>
</tr>
<tr>
<td>object3</td>
<td>500–750</td>
</tr>
<tr>
<td>object3</td>
<td>1000–1750</td>
</tr>
<tr>
<td>object3</td>
<td>2500–2750</td>
</tr>
<tr>
<td>object3</td>
<td>3250–5000</td>
</tr>
<tr>
<td>object4</td>
<td>1500–2250</td>
</tr>
<tr>
<td>object4</td>
<td>4500–5000</td>
</tr>
<tr>
<td>object5</td>
<td>250–750</td>
</tr>
<tr>
<td>object5</td>
<td>1250–2750</td>
</tr>
<tr>
<td>object5</td>
<td>3500–3750</td>
</tr>
<tr>
<td>object5</td>
<td>4500–5000</td>
</tr>
</tbody>
</table>
Frame-segment tree structure

- Suppose there are $n$ objects $o_1, \ldots, o_n$ in our video $v$ and $m$ activities $a_1, \ldots, a_m$.
- Then we have a total of:
  $$\sum_{i=1}^{n} (\text{card}(\sigma_v(o_i))) + \sum_{j=1}^{m} (\text{card}(\sigma_v(a_j)))$$
  entries in the table just for one single video.
- FS-trees use the following components:
  - **OBJECTARRAY**: specifies, for each object, an ordered linked list of pointers to nodes in the frame segment tree specifying which segments the object appears in.
  - **ACTIVITYARRAY**: specifies, for each activity, an ordered linked list of pointers to nodes in the frame segment tree specifying which segments the activity occurs in.
  - The FS-tree is now constructed from the segment table.
Frame-segment trees, continued

• **Step 1:** Let \([s_1, e_1), \ldots, [s_w, e_w]\) be all the intervals in the “Segment” column of the segment table. Let

\[ q_1, \ldots, q_z \]

be an enumeration, in ascending order, of all members of \(\{s_i, e_i \mid 1 \leq i \leq w\}\) with duplicates eliminated.

If \(z\) is not an exponent of 2, then do as follows: let \(r\) be the smallest integer such that \(z < 2^r\) and \(2^r > \text{framenum}(v)\). Add new elements \(q_{z+1}, \ldots, q_{2r}\) such that \(q_{2r} = \text{framenum}(v) + 1\) and \(q_{z+j} = q_z + j\) (for \(z + j < 2^r\)).

*By virtue of the above argument, we may proceed under the assumption that \(z\) is an exponent of 2, i.e. \(z = 2^r\) for some \(r\).*

• **Step 2:** The frame-segment tree is a binary tree constructed as follows.

1. Each node in the frame segment tree represents a frame-sequence \([x, y]\) starting at frame \(x\) and including all frames up to, but not including, frame \(y\).

2. Every leaf is at level \(r\). The leftmost leaf denotes the interval \([z_1, z_2]\), the second from left-most represents the interval \([z_2, z_3]\), the third from left-most represents the interval \([z_3, z_4]\) and so on. If \(N\) is a node with two children
representing the intervals \([p_1, p_2), [p_2, p_3)\), then \(N\) represents the interval \([p_1, p_3)\). Thus, the root of the segment tree represents the interval \([q_1, q_2)\) if \(q_z\) is an exponent of 2; otherwise it represents the interval \([q_1, \infty)\).

3. The number inside each node may be viewed as the address of that node.

4. The set of numbers placed next to a node denotes the id-numbers of video objects and activities that appear in the entire frame-sequence associated with that node. Thus, for example, if a node \(N\) represents the frame sequence \([i, j)\) and object \(o\) occurs in all frames in \([i, j)\), then object \(o\) labels node \(N\) (unless object \(o\) labels an ancestor of node \(N\) in the tree).
Example
Example, continued
Example, continued
Example, continued

<table>
<thead>
<tr>
<th>LB=250</th>
<th>OBJ</th>
</tr>
</thead>
<tbody>
<tr>
<td>UB=500</td>
<td>J</td>
</tr>
</tbody>
</table>

.locations in OBJECTARRAY

V.S. Subrahmanian — All Rights Reserved

These slides may not be duplicated without explicit written permission from Morgan Kaufmann Press.

Principles of Multimedia Database Systems    Morgan Kaufmann    Copyright ©1997
Example, continued

[Diagram of a frame-segment tree with nodes labeled 17, 18, 23, and 24.]

nodes in frame-segment tree
Video Library

- Suppose our video library, VidLib, contains videos \( v_1, \ldots, v_n \).
- Create a table called INTOBJECTARRAY having the scheme (VID_ID, OBJ, PTR).
- Tuple \( (v, o, ptr) \) is in INTOBJECTARRAY iff the pair \( (o, ptr) \) is in the OBJECTARRAY associated with video \( v \).
- Create a table called INTACTIVITYARRAY having the scheme (VID_ID, ACT, PTR).
- Tuple \( (v, a, ptr) \) is in INTOBJECTARRAY iff the pair \( (a, ptr) \) is in the ACTIVITYARRAY associated with video \( v \).
- For each \( v_i \), a frame segment tree, \( \text{fst}(v_i) \) is associated with video \( v_i \).
- Only difference from before is that pointers from the frame segment tree point to locations in INTOBJECTARRAY and INTACTIVITYARRAY rather than to OBJECTARRAY and ACTIVITYARRAY as described earlier.
Implementing Video Operations

- **FindVideoWithObject(o):**

  ```
  SELECT VIDEO_ID
  FROM INTOBJECTARRAY
  WHERE OBJ = o.
  ```

- **FindVideoWithActivityandProp(a,p,z):**

  ```
  SELECT VIDEO_ID
  FROM INTOBJECTARRAY t
  WHERE OBJ = o AND t.p = z.
  ```

- **FindObjectsInVideo(v,s,e):**

  ```
  Algorithm 4 FindObjectsInVideo(R,s,e) 
  S = NIL; (* no objects found so far *)
  if R=NIL then { Return S; Halt }
  else 
    { if [R.L.B, R.U.B] ⊆ [s, e) then S = append(S, preorder(R)) } 
    else 
      { if [R.L.B, R.U.B] ∩ [s, e) ≠ ∅ then 
        { S = append(S, R.obj); 
          S = append(S, FindObjectsInVideo(R.LINK, s,e)); 
          S = append(S, FindObjectsInVideo(R.RLINK, s,e)); 
        } 
      } 
  return(S); end
  ```
RS-Trees

- Very similar to the frame segment tree.
- The concepts of **OBJECTARRAY** and **ACTIVITYARRAY** remain the same as before.
- Instead of using a segment tree to represent the frame sequences (such as those shown in Figure ??), we take advantage of the fact that a sequence \([s, e)\) is a rectangle of length \(e - s\) and of width 0.
- We already know how to represent a set of rectangles using an R-tree.
- Example:
Video Segmentation

- We have assumed a logical delineation of video data – video is broken up into homogeneous segments.

- Usually, a video is created by taking a set of shots.

- These shots are then composed together using specified composition operators.

- Shots are usually taken with a fixed set of cameras each of which has a constant relative velocity.

- A shot composition operator, often referred to as an edit effect is an operation that takes as input, two shots, $S_1, S_2$, and a duration, $t$, and merges the two shots into a composite shot in within time $t$.

- Thus, for example, suppose we wish to compose together, two shots $S_1, S_2$ and suppose these two shots have durations $t_1, t_2$ respectively. If $f$ is a shot composition operator, then

  $$ f(S_1, S_2, t) $$

  creates a segment of video of length $(t_1 + t_2 + t)$. $S_1$ is first shown and then undergoes a continuous transformation over a time interval $t$ leading to the presentation of $S_2$ next.

- $f(S_1, S_2, t)$ then is a continuous sequence of video.

- In general, a video as a whole may be represented as:

  $$ (f_n(\ldots f_2(f_1(S_1, S_2, t_1), S_3, t_2)\ldots, S_n, t_n)). $$
Shot Composition Operators

• **Shot Concatenation**: Concatenates the two shots (even if the transition is not smooth). If `shotcat` is a shot concatenation operator, then `t` must be zero, i.e. whenever we invoke `shotcat(S_1, S_2, t)`, the third argument `t` must be set to 0.

• **Spatial Composition**: Examples include: a `translate` operation which causes two successive shots to be overlaid one on top of the other. For instance, suppose we want to show shot `S_1` first, followed by shot `S_2`. This is done by first overlaying shots `S_1` *on top* of shot `S_2` and then moving (i.e. translating) shot `S_1` away, thus exposing shot `S_2`.

• **Chromatic Composition**: `fades` and `dissolves`. Both these operations are chromatic scaling operations that try to continuously transform each pixel `(x, y)` in the first shot into the corresponding pixel in the second shot.
Video Segmentation Problem

- Given a video $V$, express the video $V$ in the form:

$$V = f_n(\ldots f_2(f_1(S_1, S_2, t_1), S_3, t_2)\ldots, S_n, t_n).$$

- That is, given video $V$, find $n$ and shots $S_1, \ldots, S_n$, times $t_1, \ldots, t_n$, and composition operations $f_1, \ldots, f_n$ such that the above equation holds.
Video Standards

- All video compression standards attempt to compress videos by performing an *intra-frame* analysis.
- Each frame is divided up into blocks. Compare different frames to see which data is “redundant” in the two frames.
- Drop redundant data to compress.
- Compression quality is measured by:
  - the fidelity of the color map – how many colors of the original video occur when the compressed video is decompressed?
  - the pixel resolution per frame – how many pixels per frame of the video have been dropped?
  - the number of frames per second – how many frames have been dropped?
- Compression Standards: MPEG-1,2,3, Cinepak, JPEG video etc.
MPEG-1

- Stores videos as a sequence of I, P and B frames.
- I frames are independent images called “intra frames”. Basically a still image.
- P-frame is computed from the closest I-frame preceding it by interpolation (using DCT).
- B-frames are computed by interpolating from the two closest P or I frames.
MPEG-1, Continued
Other Compression Standards

- MPEG-2 uses higher pixel resolution and a higher data rate, thus making it superior to MPEG-1 in terms of the quality of the video as seen by the user. However, it requires higher bandwidth thus making it feasible for some, but not all applications.

- MPEG-3 uses even higher sampling rates and frames per second that MPEG-2.
Audio Databases
Audio Databases

- Using metadata to represent audio content is done in a very similar way as we did for video.
- For example, consider an audio-recording of an opera.
- Given any interval of time in the opera, there are some activities that occur in that time interval, and some objects that occur in it.
- For example, we may have the following objects:

  1. **Singers**: a set valued field containing records having a **Role**, **SingerType** and **SingerName** field.
  2. **Score**: this may be a field of type `music_doc` which points to a relevant part of the music score associated with the time segment [5, 9).
  3. **Transcript**: This may be a field of type `document` that points to the relevant part of the libretto (words being sung) during the time segment [5, 9).

```
activity1
activity2
activity3
activity4
```

```
50 100 150 200
```
Signal-based Audio Content

- Audio data is considered as a signal, $\phi(x)$, over time $x$.
- Different features of the signal $\phi$ are extracted, indexed and stored for efficient retrieval.
- Example Signals:

  ![Diagram](a)

  ![Diagram](b)

- **Period of vibration** $T =$ time taken for a “particle” in the wave to return to its starting position.
- **Frequency of vibration** $f =$ number of vibrations per second. $f = \frac{1}{T}$.
• of the wave is the speed at which the crests and troughs move to the right. If if \( w \) denotes the wavelength of the wave, then

\[
v = \frac{w}{T}.
\]

\[
= w \times f.
\]

• *Amplitude* \( a \) of a wave is the maximum intensity of the signal associated with the wave.
Segmentation

- Split up the audio signal into relatively homogeneous “windows.” This may be done in one of two ways.
  - Application developer can specify, a priori, a window size $w$ (in seconds or milliseconds), and assume that the wave’s properties within that window are obtained by averaging.
  - Use a homogeneity predicate as in the case of images.
Feature Extraction

- After segmentation, the audio signal may be viewed as a sequence of \( n \) windows, \( w_1, \ldots, w_n \).
- For each window, we extract some features associated with the audio signal.
- If \( k \) features are extracted, then an audio signal may be considered to be a sequence of \( n \) points in a \( k \)-dimensional space.
- Example features:
  
  - **Intensity:**
    
    \[
    I = 2 \times \pi^2 \times f^2 \times \mu \times a^2 \times v
    \]
    
    where \( f \) is the frequency of the wave (in Hz), \( \mu \) is the density of the material through which the sound is being propagated (in kg/m\(^3\)), \( a \) is the amplitude of the wave in meters, and \( v \) is the velocity of the wave in meters per second. The intensity is in Watts per square meters.

  - **Loudness:** Suppose \( L_0 \) denotes the loudness associated with the lowest frequency (about 15 Hz) that a human ear can detect, and suppose we are looking at a wave of intensity \( I \). Then the loudness of \( I \), in decibels, is given by:
    
    \[
    L = 10 \times \log\left(\frac{I}{L_0}\right).
    \]
    
    Notice that when \( I = L_0 \), then \( L = 10 \times \log(1) = 0 \).
- **Pitch:** The pitch, $p(f, a)$ of an audio signal is computed as a derived quantity from the frequency $f$ and amplitude $a$ of the signal. Typically, given any window of the sort shown in Figure ??, the pitch is computed using a gcd algorithm in terms of the frequency and amplitude.

- **Brightness:** The brightness of a signal $\phi$ in a window $w$ is a measure of how “clear” the sound is. For instance, a muffled sound is less bright than the sound resulting from breaking glass.
Capturing Audio Content through Discrete Transformations

- Even a relatively short audio recording (say of about 10 minutes duration when played back), may have as many as 100,000 windows, if one assumes that each window represents a fairly “smooth” signal.

- Reduce the number of windows of homogeneous segments through the use of the Discrete Fourier Transform (DFT), and the Discrete Cosine Transform (DCT).

- Suppose we have a total of \( n \) windows after segmenting the audio signal, but we wish to store this in an array, \( A \), of size \( N \), where \( N \) is considerably smaller than \( n \).

- For each field \( f \), we may compute a value for \( A[i].f, 1 \leq i \leq n \), as follows, using the DFT:

\[
A[r].f = \sum_{j=0}^{N-1} \phi(j).f \exp\left(-\frac{2\pi \times i \times r \times j}{N}\right)
\]

- Here, \( \phi(j).f \) refers to the value of property \( f \) at time \( j \) of the signal \( \phi \). As usual, the symbol \( i \) denotes the complex number \( \sqrt{-1} \).
Indexing Audio Data

Algorithm 5 CreateAudioIndex(K, N)

Index = NIL; (* index is initially empty *)
for i = 1 to K do
    for j = 0 to (N - 1) do
        $A^i[j] = DFT(\phi_i)$;
        $A^i[N] = i$;
        (* insert vector $A^i[j]$ into TV-tree *)
        Index = insert($A^i[j], Index$)
    end for
end for
Multimedia Databases

- Thus, far we have shown how to represent and organize the content of different types of *individual* media data.
- What about databases containing a mix of media types?
- Such databases can arise in one of three ways:
  - All the media data is “legacy” data.
  - Some of the media data is “legacy” data, others are being specially crafted.
  - None of the media data is “legacy” data, all of it is being specially crafted.
(The Principle of Autonomy) Should we choose to group together all images, all videos, all documents, and index each of them in a way that is maximally efficient for the expected types of accesses we plan to make on those objects?

(The Principle of Uniformity) Should we try to find a single abstract structure $A$ that can be used to index all the above media types, and that can thus be used to create a “unified index” that can then be used to access the different media objects?

(The Principle of Hybrid Organization) A third possibility is to use a hybrid of the previous two principles. In effect, according to this principle, certain media types use their own indexes, while others use the “unified” index.
The Principle of Autonomy
The Principle of Uniformity
The Principle of Hybrid Organization
Organizing Multimedia Data based on the Principle of Uniformity

Suppose we consider the following statements about media data. These statements may be made by a human or may be produced as the output of an image/video/text content retrieval engine.

- The image `photo1.gif` shows Jane Shady, Denis Dopeman and an unidentified third person, in Medellin, Colombia. The picture was taken on January 5, 1997.

- The video-clip `video1.mpg` shows Jane Shady giving Daniel Dopeman a briefcase (in frames 50-100). The video was obtained from surveillance set up at Denis Dopeman’s house in Rockville, Maryland, in October, 1996.

- The document `dopeman.txt` contains background information on Denis Dopeman, obtained from Mr. Dopeman’s FBI file.

All the above statements are `Metadata` statements.

- Associate, with each media object $o_i$, some meta-data, $\text{md}(o_i)$.
- $\text{md}(o_i)$ may be produced by a human or a program.
- If our archive contains objects $o_1,\ldots,o_n$, then index the metadata $\text{md}(o_1),\ldots,\text{md}(o_n)$ in a way that provides efficient ways of implementing the expected accesses that users will make.
Media-Abstractions

- If we consider the content of media data of different types, what is it that is common to all these media types, and what is it that is different?

- We would like a single data structure that can represent different instances of this common “core”.

- Media abstractions are mathematical structures representing such media content.

- Media abstractions may be implemented through a single data structure.
Media-Abstractions continued

• A media-abstraction is an 8-tuple

\[(S, fe, ATTR, \lambda, R, F, Var_1, Var_2)\]

where:
- \(S\) is a set of objects called states, and
- \(fe\) is a set of objects called features, and
- \(ATTR\) is a set of objects called attribute values, and
- \(\lambda : S \rightarrow 2^{fe}\) is a map from states to sets of features, and
- \(R\) is a set of relations on \(fe^i \times ATTR^j \times S\) for \(i, j \geq 0\), and
- \(F\) is a set of relations of \(S\) and
- \(Var_1\) is a set of objects, called variables, ranging over \(S\), and
- \(Var_2\) is a set of variables ranging over \(fe\).
Media-Abstractions: Intuition

- A “state” is the smallest “chunk” of media data that we wish to reason about.

- A “feature” is any object in a state that is deemed to be of “interest.” Features can in principle include both objects and activities.

- A feature may have one or more attributes associated with it.

- The feature extraction map specifies what features occur in which states. In some cases, feature extraction maps are implemented as content extraction programs, while in other cases, they may involve humans manually specifying content.

- $\mathcal{R}$ is a set of state-dependent and state-independent relations, e.g. $\text{left-of}$ is state dependent, while $\text{age}$ is not.

- $\mathcal{F}$ contains inter-state relations, e.g $\text{before}(s_1,s_2)$ may say that state $s_1$ occurred before state $s_2$. 
Image Data Viewed as a Media-Abstraction

- The set of states consists of \{pic1.gif, \ldots, pic7.gif\}.

- **Features:** The set of features consists of the names of the people shown in the photographs. Let us call this list: Bob, Jim, Bill, Charlie, and Ed.

- **Extraction Map \( \lambda \):** This map tells us, for each state, which features occur in that state. The table below contains this description:

<table>
<thead>
<tr>
<th>State</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>pic1.gif</td>
<td>bob, jim</td>
</tr>
<tr>
<td>pic2.gif</td>
<td>jim</td>
</tr>
<tr>
<td>pic3.gif</td>
<td>bob</td>
</tr>
<tr>
<td>pic4.gif</td>
<td>bill</td>
</tr>
<tr>
<td>pic5.gif</td>
<td>charlie</td>
</tr>
<tr>
<td>pic6.gif</td>
<td>ed, bill</td>
</tr>
<tr>
<td>pic7.gif</td>
<td>ed</td>
</tr>
</tbody>
</table>

- The set of relations may contain just two relations: a state-dependent relation called **left_of** and a state-independent relation called **father**, with the obvious meanings.

- The set of inter-state relations may be empty.
Video Data Viewed as a Media-Abstraction

- **States:** The set of states consists just of frames 1 through 5.
- **Features:** The set of features consists of: Jane Shady, Denis Dopeman, Dopeman\_house and briefcase,
- **Extraction Map \( \lambda \):** The extraction map, \( \lambda \), is described by the following simple table:

<table>
<thead>
<tr>
<th>State</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>frame1</td>
<td>Dopeman_house, briefcase, Jane Shady</td>
</tr>
<tr>
<td>frame2</td>
<td>Dopeman_house, briefcase, Jane Shady, Denis Dopeman</td>
</tr>
<tr>
<td>frame3</td>
<td>Dopeman_house, briefcase, Jane Shady, Denis Dopeman</td>
</tr>
<tr>
<td>frame4</td>
<td>Dopeman_house, briefcase, Jane Shady, Denis Dopeman</td>
</tr>
<tr>
<td>frame5</td>
<td>Dopeman_house, Jane Shady</td>
</tr>
</tbody>
</table>

- **Relations:**

  1. **have** is a state-dependent relation specifying who has an object in a given state. This may be given by the following simple table:

<table>
<thead>
<tr>
<th>Person</th>
<th>Object</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jane Shady</td>
<td>briefcase</td>
<td>1</td>
</tr>
<tr>
<td>Jane Shady</td>
<td>briefcase</td>
<td>2</td>
</tr>
<tr>
<td>Jane Shady</td>
<td>briefcase</td>
<td>3</td>
</tr>
<tr>
<td>Denis Dopeman</td>
<td>briefcase</td>
<td>4</td>
</tr>
</tbody>
</table>
2. **spouse** is a state-independent relation specifying the name of the spouse of an individual. This may be given by the simple table:

<table>
<thead>
<tr>
<th>Person</th>
<th>Spouse</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jane Shady</td>
<td>Peter Shady</td>
</tr>
<tr>
<td>Jane Shady</td>
<td>Peter Shady</td>
</tr>
<tr>
<td>Denis Dopeman</td>
<td>Debra Dopewoman</td>
</tr>
</tbody>
</table>

3. **Inter-State Relations:** This may consist of just one relation called \texttt{before(s1,s2)} which holds iff state \texttt{s1} occurs before state \texttt{s2}. 
Simple Multimedia Database

- A simple multimedia database is a finite set, $\mathcal{M}$, of media-abstractions.

- Simple multimedia databases are naive.

- A media abstraction may list “Church” as a feature; however, when searching for “Cathedrals” or “Monuments”, we may not find the church, because the system does not know that cathedrals and churches are (more or less) synonymous, and that all churches are monuments (but not vice-versa).

- Also, users often search for media objects containing one or more features, and then “refine” the search later when they find that the media-objects returned by their query, though correct, do not correspond precisely to what they wanted.
Structured Multimedia Database

A structured multimedia database system, SMDS, is a 5-tuple \(\{M_1, \ldots, M_n\}, \equiv, \leq, \text{inh}, \text{subst}\) where:

- \(M_i = (S^i, f_\text{e}^i, \text{ATTR}^i, \lambda^i, \mathcal{R}^i, \mathcal{F}^i, \text{Var}_1^i, \text{Var}_2^i)\) is a media-abstraction, and

- \(\equiv\) is an equivalence relation on \(\mathcal{F} = \cup_{i=1}^n f_\text{e}^i\);

- \(\leq\) is a partial ordering on the set \(\mathcal{F}/\equiv\) of equivalence classes on \(\mathcal{F}\), and

- \(\text{inh} : \mathcal{F}/\equiv \rightarrow 2^{\mathcal{F}/\equiv}\) such that \([f_1] \in \text{inh}([f_2])\) implies that \([f_1] \leq [f_2]\). Thus, \(\text{inh}\) is a map that associates with each feature \(f\), a set of features that are "below" \(f\) according to the \(\leq\)-ordering on features.

- \(\text{subst}\) is a map from \(\cup_{i=1}^n \text{ATTR}^i\) to \(2^{\cup_{i=1}^n \text{ATTR}^i}\).
### Example SMDS

<table>
<thead>
<tr>
<th>Media</th>
<th>Object</th>
<th>Part/frame</th>
<th>Feature(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>image</td>
<td>photo1.gif</td>
<td>-</td>
<td>church, durnstein, danube, subrahmanian</td>
</tr>
<tr>
<td>image</td>
<td>photo2.gif</td>
<td>-</td>
<td>cathedral, melk, subrahmanian</td>
</tr>
<tr>
<td>image</td>
<td>photo3.gif</td>
<td>-</td>
<td>church, st. paul, rome</td>
</tr>
<tr>
<td>video</td>
<td>video1.mpg</td>
<td>1-5</td>
<td>church, durnstein, stream</td>
</tr>
<tr>
<td>video</td>
<td>video1.mpg</td>
<td>6-10</td>
<td>stream</td>
</tr>
<tr>
<td>audio</td>
<td>audiol.wav</td>
<td>1-20</td>
<td>st. peters, tiber, rome</td>
</tr>
</tbody>
</table>

- three media-abstractions, one each associated with image, video, and audio data.

- the set of features, \( F \) contains: church, durnstein, danube, subrahmanian, cathedral, melk, st. paul, rome, stream, restaurant, st. peters, tiber.

- \( \equiv \) says that:
  - church \( \equiv \) cathedral.
  - river \( \equiv \) stream.

- the \( \leq \) relation says that:

```
                church, cathedral
               /             
st. paul st. peter
```

```
                river, stream
               /             
  danube  tiber  melk  durnstein  rome  subrahmanian
```
Querying SMDSs (Uniform Representation)

Basic SMDS functions are:

- **FindType(Obj)**: This function takes as input, a media object Obj as input, and returns the output type of the object. For example,

\[
\text{FindType(im1.gif)} = \text{gif}.
\]
\[
\text{FindType(movie1.mpg)} = \text{mpg}.
\]

- **FindObjectWithFeature(f)**: This function takes as input, a feature f and returns as output, the set of all media-objects that contain that feature. For example,

\[
\text{FindObjectWithFeature(john)} = \{\text{im1.gif,im2.gif,im3.gif,video1.mpg:[1,5]}\}.
\]
\[
\text{FindObjectWithFeature(mary)} = \{\text{video1.mpg:[1,5], video1.mpg:[15,50]}\}.
\]

- **FindObjectWithFeatureandAttr(f,a,v)**: This function takes as input, a feature f, an attribute name a associated with that feature, and a value v. It returns as output, all objects o that contain the feature and such the value of the attribute a in object o is v. For example,

1. **FindObjectWithFeatureandAttr(Jane Shady,suit,blue)**: This query asks to find all media objects in which Jane Shady appears in a blue suit.
2. `FindObjWithFeatureandAttr(Elephant, bow, red)`: This query asks to find all media objects in which an elephant wearing a red bow appears.

- **FindFeaturesinObj(Obj)**: This query asks to find all features that occur within a given media object. It returns as output, the set of all such features. For example,
  1. **FindFeaturesinObj(im1.gif)**: This asks for all features within the image file `im1.gif`. It may return as output, the objects John, and Lisa.
  2. **FindFeaturesinObj(video1.mpg:[1,5])**: This asks for all features within the first 5 frames of the video file `video1.mpg`. The answer may include objects such as Mary and John.

- **FindFeaturesandAttrinObj(Obj)**: This query is exactly like the previous query except that it returns as output, a relation having the scheme:

  $$(\text{Feature}, \text{Attribute}, \text{Value})$$

  where the triple $(f, a, v)$ occurs in the output relation iff feature $f$ occurs in the query `FindFeaturesinObj(Obj)` and feature $f$’s attribute $a$ is defined and has value $v$. For example,

  `FindFeaturesandAttrinObj(im1.gif` may return as answer, the table:
<table>
<thead>
<tr>
<th>Feature</th>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>age</td>
<td>32</td>
</tr>
<tr>
<td>John</td>
<td>address</td>
<td>32 Pico Lane, Mclean, VA 22050.</td>
</tr>
<tr>
<td>Mary</td>
<td>age</td>
<td>46</td>
</tr>
<tr>
<td>Mary</td>
<td>address</td>
<td>16 Shaw Road, Dumfries, VA 22908.</td>
</tr>
<tr>
<td>Mary</td>
<td>employer</td>
<td>XYZ Corp.</td>
</tr>
<tr>
<td>Mary</td>
<td>boss</td>
<td>David</td>
</tr>
</tbody>
</table>
SMDS-SQL

- All ordinary SQL statements are SMDS-SQL statements. In addition:

- The **SELECT** statement may contain *media-entities*. A *media entity* is defined as follows:

  1. If \( m \) is a continuous media object, and \( i, j \) are integers, then \( m : [i, j] \) is a media-entity denoting the set of all frames of media object \( m \), that lie between (and inclusive of) segments \( i, j \).

  2. If \( m \) is not a continuous media-object, then \( m \) is a media entity.

  3. If \( m \) is a media-entity, and \( a \) is an attribute of \( m \), then \( m.a \) is a media-entity.

- The **FROM** statement may contain entries of the form

  \[
  \langle \text{media} \rangle \langle \text{source} \rangle \langle M \rangle
  \]

  which says that only all media-objects associate with the named media type and named data source are to be considered when processing the query, and that \( M \) is a variable ranging over such media objects.

- The **WHERE** statement allows (in addition to standard SQL constructs), expressions of the form:

  \[
  \text{term} \ \text{IN} \ \text{func\_call}
  \]

  where:
1. *term* is either a variable (in which case it ranges over the output type of *func_call*) or an object having the same output type as *func_call* and

2. *func_call* is any of the five function calls listed above.
SMDS-SQL Examples

- **Find all image/video objects containing both Jane Shady and Denis Dopeman.** This can be expressed as the SMDS-SQL query:

```sql
SELECT M
FROM  smds source1 M
WHERE  (FindType(M)=Video OR FindType(M)=Image)
       AND
       M IN FindObjWithFeature(Denis Dopeman)
       AND
       M IN FindObjWithFeature(Jane Shady).
```

- **Find all image/video objects containing Jane Shady wearing a purple suit.** This can be expressed as the SMDS-SQL query:

```sql
SELECT M
FROM  smds source1 M
WHERE  (FindType(M)=Video OR FindType(M)=Image)
       AND
       M IN FindObjWithFeatureandAttr(Jane Shady,suit, purple).
```

- **Find all images containing Jane Shady and a person who appears in a video with Denis Dopeman.** Unlike the preceding queries, this query involves computing a "join" like operations across different data domains. In order to do this,
we use existential variables such as the variable “Person” in
the query below, which is used to refer to the existence of an
“unknown” person whose identity is to be determined.

SELECT M,Person
FROM smds source1 M,M1
WHERE FindType(M)=Image) AND
    FindType(M1)=Video) AND
    M IN FindObjectWithFeature(Jane Shady) AND
    M1 IN FindObjectWithFeature(Denis Dopeman)
    AND
    Person IN FindFeaturesinObj(M)AND
    Person IN FindFeaturesinObj(M1)AND
    Person ≠ Jane Shady AND Person ≠ Denis Dopeman.
Querying Hybrid Representations of Multimedia Data

- SMDS-SQL may be used to query multimedia objects which are stored in the uniform representation.

- "What is it about the hybrid representation that causes our query language to change?"

- In the uniform representation, all the data sources being queried are SMDSs, while in the hybrid representation, different (non-SMDS) representations may be used.

- A hybrid media representation basically consists of two parts – a set of media objects that use the uniform representation (which we have already treated in the preceding section), and a set of media-types that use their own specialized access structures and query language.

- To extend SMDS-SQL to Hybrid-Multimedia SQL (HM-SQL for short), we need to do two things:
  
  - First, HM-SQL must have the ability to express queries in each of the specialized languages used by these non-SMDS sources.
  
  - Second, HM-SQL must have the ability to express "joins" and other similar binary algebraic operations between SMDS-sources, and non-SMDS sources.
**HM-SQL**

HM-SQL is exactly like SQL except that the `SELECT`, `FROM`, `WHERE` clauses are extended as follows:

- the `SELECT` and `FROM` clauses are treated in exactly the same way as in SMDS-SQL.
- The `WHERE` statement allows (in addition to standard SQL constructs), expressions of the form:

  $$\text{term IN MS : func\_call}$$

where:

1. `term` is either a variable (in which case it ranges over the output type of `func\_call`) or an object having the same output type as `func\_call` as defined in the media-source `MS` and
2. either `MS` = SMDS and `em func\_call` is one of the five SMDS functions described earlier, or
3. `MS` is not an SMDS-media source, and `func\_call` is a query in `QL(MS)`.

- Thus, there are 2 differences between HM-SQL and SMDS-SQL:

  1. `func\_calls` occurring in the `WHERE` clause must be explicitly annotated with the media-source involved, and
2. queries from the query languages of the individual (non-SMDS) media-source implementations may be embedded within an HM-SQL query. This latter feature makes HM-SQL very powerful indeed as it is, in principle, able to express queries in other, third-party, or legacy media implementations.
HM-SQL Examples

- **Find all video clips containing Denis Dopeman, from both the video sources, video1, and video2.**

  ```sql
  SELECT M
  FROM smds video1, videodb video2
  WHERE M IN smds:FindObjWithFeature(Denis Dopeman) OR
  M IN videodb:FindVideoWithObject(Denis Dopeman).
  ```

- **Find all people seen with Denis Dopeman in either video1, video2, or idb.**

  ```sql
  (SELECT P1
   FROM smds video1 V1
   WHERE V1 IN smds:FindObjWithFeature(Denis Dopeman) AND
   P1 IN smds:FindFeaturesInObj(V1) AND
   P1 != Denis Dopeman)
  UNION
  (SELECT P2
   FROM videodb video2 V2
   WHERE V2 IN smds:FindObjWithFeature(Denis Dopeman) AND
   P2 IN videodb:FindObjectsInVideo(V1) AND
   P2 != Denis Dopeman)
  UNION
  (SELECT *
   FROM imagedb idb I2
   WHERE V2 IN image:getpic(Denis Dopeman) AND
   P2 IN imagedb:getfeatures(V1) AND
   P2 != Denis Dopeman).
  ```
Indexing SMDSs with Enhanced Inverted Indexes

Suppose \( \{ \mathcal{M}_1, \ldots, \mathcal{M}_n \} \), \( \equiv, \leq, \text{inh, subst} \) where

\[
\mathcal{M}_i = (S^i, \text{fe}^i, \text{ATTR}^i, \lambda^i, \mathcal{R}^i, \mathcal{F}^i, \text{Var}_1^i, \text{Var}_2^i)
\]

is an SMDS.

1. FEATURETABLE: This is a hash table whose entries are features in \( \bigcup_{i=1}^n \text{fe}^i \). Each hash table location \( i \) contains a bucket of “featurenodes” that hash to location \( i \).

2. STATETABLE: This is a hash table whose entries are states in \( \bigcup_{i=1}^n S^i \). Like the FEATURETABLE, the STATETABLE contains a bucket of “statenodes” that hash to the specified location.

3. FEATURENODEs: Each featurenode contains:
   
   - the name of the feature (e.g. “Denis Dopeman”),
   - a list of children nodes (if \( f_1, f_2 \) are features in \( \bigcup_{i=1}^n \text{fe}^i \), we say that \( f_2 \) is a child of \( f_1 \) iff \( f_2 \leq f_1 \) and there is no other feature \( f_3 \in \bigcup_{i=1}^n \text{fe}^i \) such that \( f_2 < f_3 < f_1 \))
   - a list of pointers to statenodes (see below) that contain that feature (e.g. in the case of an image database, this would be a pointer to the nodes associated with images that contain the feature in question, e.g. Denis Dopeman)
• a list of pointers to other featurenodes that are appropriate substitutes for the featurenode in question. Specifically, if we consider a feature node associated with feature $f$, then a pointer to feature $g$ is in this list iff $g \in h(f)$.

4. STATENODEs: A statenode consists of just two components: a pointer to a file containing the media-object (image, video, audio, document, etc.) that the state in question refers to, and a linked list whose members point to featurenodes – intuitively, there is a pointer to a featurenode $f$ iff the feature in question is in the state.
Data Structure

type featurenode = record of /* nodes in feature graph */
    name : string; /* name of feature */
    children: ~node1; /* points to a list of pointers to the children */
    statelist: ~node2; /* points to a list of pointers to states that */
        /* contain this feature */
    replacelist: ~node3; /* points to a list of descendants whose */
        /* associated states can be deemed to have the */
        /* the feature associated with this node */
    end record;

type node1 record of
    element: ~featurenode; /* points to a child of a featurenode */
    next : ~node1; /* points to next child */
end record;

type node2 record of
    state: ~statenode; /* pointer to the list associated with a state */
    link: ~node2; /* next node */
end record;

type node3 record of
    feat: ~featurenode; /* pointer to a node that can be deemed to have */
        /* the feature associated with the current node */
    link1: ~node3;
end record;

type statenode record of
    rep: ~framerep;
    flist: ~node4;
end record;

type node4 record of
    f : ~featurenode;
    link2: ~node4;
end record;
Example

- Consider a very simple toy SMDS containing three media-abstractions: images, video, and audio, with following content:

<table>
<thead>
<tr>
<th>Media-Abstraction</th>
<th>State</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>image</td>
<td>im1</td>
<td>john, mary</td>
</tr>
<tr>
<td></td>
<td>im2</td>
<td>john, mary, liz</td>
</tr>
<tr>
<td></td>
<td>im3</td>
<td>liz, mary</td>
</tr>
<tr>
<td>video</td>
<td>vid1:[1,10]</td>
<td>john, singing</td>
</tr>
<tr>
<td></td>
<td>vid1:[11,20]</td>
<td>john, mary, dancing</td>
</tr>
<tr>
<td></td>
<td>vid1:[21,40]</td>
<td>john, mary, liz, singing, dancing</td>
</tr>
<tr>
<td></td>
<td>vid2:[1,20]</td>
<td>ed, speaking</td>
</tr>
<tr>
<td></td>
<td>vid2:[21,30]</td>
<td>man, speaking</td>
</tr>
<tr>
<td>audio</td>
<td>disk1:[1,20]</td>
<td>john, singing</td>
</tr>
<tr>
<td></td>
<td>disk1:[21,40]</td>
<td>woman, speaking</td>
</tr>
</tbody>
</table>

- Next slide shows one possible STATETABLE (without hashing) and a possible FEATURETABLE.

- Slide after that shows the $\leq$ ordering on features.

- We have shown the featurelist associated with the states im1 and disk1:[1,20]. Other feature lists are not shown for the sake of simplicity.

- Likewise, we have shown the statelists associated with the features man and liz.
• The relations associated with these media-abstractions may be stored within standard relational databases.
Example of STATETABLE
Example of $\leq$ Ordering

man

woman

john  ed  liz  mary  singing  dancing  speaking
Example Algorithm FindObjwithFeature(f)

Algorithm 6 $\text{FindObjwithFeature}(f)$;

1. $SOL = \emptyset$;
2. Hash feature $f$ and resolve collisions (if any) till feature $f$ is located (at location $i$ say) in the FEATURETABLE;
3. $SOL = \text{append}(SOL, \text{FeatureTable}[i].\text{Statelist})$;
4. For all children $f'$ of some member of $[f]$ do $SOL = \text{append}(SOL, \text{FindObjwithFeature}(f'))$
5. Return $SOL$.

This assumes that if $f'$ is a descendant of $f$ then $f'$ occurs in every state in which $f$ occurs. We may easily modify the above algorithm if this assumption is not valid.
Find features in state $s$

1. Hash state $s$ and resolve collisions (if any) till state $s$ is located (at location $i$ say) in the STATETABLE;

2. **Return** STATETABLE[i].Featurelist.
Check if state $s$ contains feature $f$

1. Hash state $s$ and resolve collisions (if any) till state $s$ is located (at location $i$ say) in the STATETABLE;

2. If $f$ is in STATETABLE[$i$] Featurelist, then Return true else Return false.
Query Relaxation/Expansion

- The inh and subst components of an SMDS are used to determine how queries must be relaxed.
- When a user poses a query $Q$, we somehow modify that query $Q$ into a set $\{Q_1, \ldots, Q_k\}$ of queries.
- This set is partially ordered and contains the original query $Q$ as the maximal element of the ordering.
- Intuitively, if $Q'$ is a child of $Q''$ then this means that $Q'$ is obtained from $Q''$ by making a modification or relaxation.
- Thus, query relaxation depends on two aspects:
  - what are the “modification” operations that are allowed to modify queries?
  - what is the ordering on the set of modified queries?
FindObjwithFeatureandInh(f,inh)

Algorithm 7 FindObjwithFeatureandInh(f,inh);
1. SOL = ∅;
2. Hash feature f and resolve collisions (if any) till feature f is located (at location i say) in the FEATURETABLE;
3. SOL = append(SOL,Featuretable[i].StateList);
4. Forall children f’ of some member of inh([f]) do SOL = append(SOL,FindObjwithFeatureandInh(f’, ∅))
5. Return SOL.
Query Relaxation, Continued

- $Q_1$ is a feature-relaxation of $Q_2$, denoted $Q_1 \subseteq Q_2$ iff there exist features $f_1, f_2 \in \bigcup_{i=1}^{n} \mathbf{f}^i$ such that
  
  * $Q_1 \preceq Q_2[f_1/f_2]$ (where the notation $Q_2[f_1/f_2]$ denotes the replacement of all occurrences of $f_1$ in $Q_2$ by $f_2$) and
  
  * $f_2 \in \text{inh}([f_1])$.

- **EX:** $Q_2$ is:

  ```sql
  SELECT M
  FROM smds video1, videodb video2
  WHERE M | N smds:FindObjectWithFeature(Denis Dopeman) OR
           M | N videodb:FindObjectWithObject(Denis Dopeman)
  ```

- Suppose David Johns in in $\text{inh}(\text{Denis Dopeman})$.

- $Q_1$ is:

  ```sql
  SELECT M
  FROM smds video1, videodb video2
  WHERE M | N smds:FindObjectWithFeature(David Johns) OR
           M | N videodb:FindObjectWithObject(David Johns)
  ```

- $Q_1$ is a feature relaxation of $Q_2$. 
Relaxation Examples

- $Q_4$:

```sql
SELECT M
FROM smds
WHERE M |N smds:FindObjWithFeature(Denis Dopeman) AND M |N smds:FindObjWithFeatureandAttr(briefcase, color, black).
```

- $Q_5$:

```sql
SELECT M
FROM smds
WHERE M |N smds:FindObjWithFeature(Denis Dopeman) AND M |N smds:FindObjWithFeatureandAttr(package, color, grey).
```

- $Q_6$:

```sql
SELECT M
FROM smds
WHERE M |N smds:FindObjWithFeature(Denis Dopeman) AND M |N smds:FindObjWithFeatureandAttr(briefcase, color, grey).
```

- (Feature Replacement) obtained $Q_4$ by replacing occurrences of “package” in $Q_3$ with “briefcase”

- (Attribute Replacement) obtained $Q_5$ by replacing occurrences of “black” by “grey”

- (Feature-cum-Attribute Replacement) obtained $Q_6$ by replacing occurrences of “black” by “grey” and occurrences of “package” with “briefcase.”
Relaxations, Continued

- $Q_1$ is said to be a attribute-relaxation of $Q_2$, denoted $Q_1 \equiv a Q_2$ iff there exist attributes $a_1, a_2 \in \bigcup_{i=1}^{n} \text{ATTR}^i$ such that $Q_1 = S_2[a_1/a_2]$ and $a_2 \in \text{subst}(a_1)$.

- Relaxation of queries is defined inductively as follows:
  
  - If $Q_1$ is a feature-relaxation of $Q_2$, then $Q_1$ is a relaxation of $Q_2$;
  - If $Q_1$ is an attribute-relaxation of $Q_2$, then $Q_1$ is a relaxation of $Q_2$;
  - If $Q_1$ is a relaxation of $Q_3$ and $Q_3$ is a relaxation of $Q_2$, then $Q_1$ is a relaxation of $Q_2$.

- Relax($Q$) denotes the set of all relaxations of query $Q$ (w.r.t. some SMDS).
Retrieving Multimedia Data from Disks
Overview of Disks

• Disk drive Side View.

• Disk drive Top View.

• Tracks: Each disk platter consists of a number of concentric “tracks.”

• Cylinders: Suppose we are intersected in track number \( i \) for some \( i \). Each platter in our disk device contains such a track. The set of such tracks, one from each platter, is called a cylinder.

• Regions: Each disk platter is divided into \( k \) regions (for some fixed \( k \)). Each region represents a wedge of the platter with angle \( \frac{360}{k} \).
• **Sectors:** That part of a track that intersects a wedge is called a sector. Clearly, if we have \( n \) tracks altogether, then we will have \( n \) sectors per wedge.
Disk Retrieval

- Associated with each disk platter is a disk arm that contains a read-write head to read/write from/to that platter.

- When a disk address is to be accessed, the disk controllers (the programs that control the position of the disk head relative to the disk) start by pursuing two steps:

  1. **Seek Operations:** First, find the track (and hence the cylinder) on which the address is located. Seek time has 4 phases:
     (a) acceleration phase
     (b) constant velocity/coast phase
     (c) deceleration phase
     (d) settle phase.

  2. **Rotational Operation:** Once the head is positioned over the right track, the disk spindle rotates so that the sector containing the desired physical address is located directly under the read/write head. The time taken for this is called *rotational latency*.

- **Read Head:** Associated with each disk arm is a read/write head that contains the necessary hardware to read data from a sector, or write data onto a sector.

- **Transfer Rate:** Rate at which data is read/written. Varies based on whether reading or writing.
Notation

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$tnum$</td>
<td>total number of tracks</td>
</tr>
<tr>
<td>$rnum$</td>
<td>total number of regions</td>
</tr>
<tr>
<td>$itd$</td>
<td>distance between two tracks</td>
</tr>
<tr>
<td>$ss$</td>
<td>spin speed of disk in rotations per minute</td>
</tr>
<tr>
<td>$rv$</td>
<td>average radial velocity = average movement of disk head along arm</td>
</tr>
<tr>
<td>$dtr$</td>
<td>The transfer rate of the disk</td>
</tr>
<tr>
<td>$rd$</td>
<td>Recording density in Mbytes per sector</td>
</tr>
</tbody>
</table>

- Suppose we wish to read sector $i$ (on track $t_i$) on a given platter, and the read head on that platter is currently over sector $j$ in track $t_j$.

  \[
  \text{Readtime}(i, j) = \frac{rd}{dtr} + \text{spintime}(i, j) + Sk(t_i, t_j)
  \]

  where $\text{spintime}(i, j)$ is the amount of time required to spin from sector $i$ to sector $j$ and $Sk(t_i, t_j)$ is the amount of time for the read head to move from track $t_i$ to track $t_j$.

- The seek time required to find track $i$ from track $j$ (assuming the head is currently positioned at track $j$) is given (crudely) by:

  \[
  Sk(t_i, t_j) = \frac{\text{abs}(t_i - t_j)}{rv}.
  \]
• Similarly,

\[ \text{spintime}(i, j) = \text{abs}((i - j) \mod rnum) \times \frac{360}{rnum} \times \frac{1}{av}. \]
Raid-0 Architecture

- We have a set of \( n \) disks, labeled 0, 1, \ldots, \((n - 1)\).
- A \( k \)-stripe is a set of \( k \) drives for some integer \( k < n \) which divides \( n \).
- When storing a set \( b_0, b_1, \ldots, b_{r-1} \) of contiguous blocks in terms of a \( k \) striped layout, what we do is the following. We store block \( b_0 \) on disk 0, block \( b_1 \) on disk 1, block \( b_2 \) on disk 2, and so on.
- Example striping: 2 movies. The blocks of the first movie are denoted by \( b_0, b_1, b_2, b_3, b_4 \). These are striped with \( k = 3 \) starting at disk 0. Second movie has six blocks, denoted by \( c_0, \ldots, c_5 \), and these are being striped with \( k = 4 \) and starting at disk 1, i.e. \( j = 1 \).
Raid-1 Architecture

- If there are $N$ disks available altogether, then $n = \frac{N}{2}$ disks are utilized.
- For each disk, there is a “mirror” disk.
- Raid-1 is predicated on the assumption that there is a very low probability that a disk and its mirror will fail simultaneously.
- When we wish to read from a disk, we read from the disk (if it is active) or we read from its mirrored disk (if the disk has crashed). When writing to disk $d$, we must write on both the disk and its mirror.
Raid-5 Architecture

- Each cluster of $k$ disks has one disk reserved as a “parity” disk.
- Suppose that $k = n$, i.e. we have only one cluster (RAID-5 also applies when $k \neq n$, this is just an assumption for illustrative purposes).
- Let us further assume that these disks are numbered $0, 1, \ldots, (n-1)$.
- EX:

```
Controller

Disk 0
b0 b3 c2
c5

Disk 1
b1 b4 c0 c3
c6

Disk 2
b2 c1 c4

Parity Disk
```

- In this case, movie blocks are striped across $(n-1)$ of the $n$ disks available, and disk number $(n-1)$ is reserved as a “parity” disk.
- Suppose we use $D_{i,j}$ to denote the value of the $j$'th bit of disk $i$ and suppose disk $n$ is the parity disk. If the symbol $\oplus$ denotes the exclusive-or operator, then

$$D_{n-1,j} = D_{0,j} \oplus \cdots \oplus D_{n-2,j}.$$
• I.e. the $j$’th bit of the parity disk is obtained by taking the exclusive-or of the $j$’th bits of all the other disks.

• This can be used to recover data if one disk crashes.
Example

- Consider a simple example, where \( n = 3 \).
- The truth table for exclusive or is:

<table>
<thead>
<tr>
<th>( D_1 )</th>
<th>( D_2 )</th>
<th>( D_3 )</th>
<th>((\text{Parity disk}) D_p = D_1 \oplus D_2 \oplus D_3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- Suppose disk \( D_2 \) crashes and we wish to find the value for a specific bit \( j \).
- Read the value of bit \( j \) in disks \( D_1, D_3 \) and the parity disk \( D_p \).
- For instance, these values might be 0, 1, 1 respectively.
- Then examine the above truth table to see which row has \( D_1 = 0, D_3 = 1 \) and \( D_p = 1 \).
- The second last row in the table satisfies this condition.
- From this, we can infer the value of this bit in disk \( D_2 \) to be 0.
- How can we do this in general?
Service Algorithms

- Given a set of clients each of whom wants to read some data from disk, how do we do schedule their reads?

- Several algorithms proposed for this task.

- Most such algorithms *must* execute very fast, i.e. it cannot take too long for the algorithm to determine order of reads.

- Some well known algorithms:
  - First Come First Serve (FCFS)
  - SCAN
  - SCAN Earliest Deadline First (SCAN-EDF)
First Come First Served

- Each client retrieval request has an associated time stamp.
- Clients are serviced in order of their associated time-stamp.
- EXAMPLE: Suppose the disk read head is currently over track \( i \), sector \( j \). The table below shows some numbers specifying client requests, when they were made, and how long it takes to reposition the disk head from its current location to the desired location.

<table>
<thead>
<tr>
<th>RequestID</th>
<th>ReqTime</th>
<th>Est. Seek</th>
<th>Est. Rotational Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>10</td>
<td>24</td>
<td>3</td>
</tr>
<tr>
<td>r2</td>
<td>8</td>
<td>12</td>
<td>5</td>
</tr>
<tr>
<td>r3</td>
<td>14</td>
<td>30</td>
<td>6</td>
</tr>
<tr>
<td>r4</td>
<td>11</td>
<td>18</td>
<td>4</td>
</tr>
</tbody>
</table>

- In this case, FCFS will serve the requests in the order: \( r2,r1,r4,r3 \).
- The last 2 columns are completely ignored by FCFS.
SCAN Algorithm

- Suppose the disk read head is currently over track $i$, sector $j$.
- In this case, we order requests in the order of the number of tracks to be traversed from track $i$, moving either outwards first and then inwards, or vice versa, but not both. (Of course, the further away the tracks
- We then service the requests in the order prescribed.
- EXAMPLE: Suppose we return to the last example, and each track requires 3 units of time to be traversed (estimate). Then

<table>
<thead>
<tr>
<th>RequestID</th>
<th>ReqTime</th>
<th>Est. Seek (Num of tracks)</th>
<th>Est. Rotational Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>10</td>
<td>24(8)</td>
<td>3</td>
</tr>
<tr>
<td>r2</td>
<td>8</td>
<td>12(4)</td>
<td>5</td>
</tr>
<tr>
<td>r3</td>
<td>14</td>
<td>30(10)</td>
<td>6</td>
</tr>
<tr>
<td>r4</td>
<td>11</td>
<td>18(6)</td>
<td>4</td>
</tr>
</tbody>
</table>

- If we assume that all of $r1$–$r4$ are in tracks beyond track $i$ (i.e. between track $i$ and the outer rim of the disk) then it is easy to see that we will provide service in the order $r2,r4,r1,r3$. 
SCAN-EDF Algorithm

- EDF stands for “Earliest Deadline First”.
- Here, we first group all requests in ascending order of their deadline.
- Suppose the resulting groups are G1,G2,...Gn.
- Each group is serviced using SCAN.
- EXAMPLE:

<table>
<thead>
<tr>
<th>RequestID</th>
<th>ReqTime</th>
<th>Est. Seek</th>
<th>Est. Rotational Delay</th>
<th>Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>(Num of tracks)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>r1</td>
<td>10</td>
<td>24(8)</td>
<td>3</td>
<td>100</td>
</tr>
<tr>
<td>r2</td>
<td>8</td>
<td>12(4)</td>
<td>5</td>
<td>120</td>
</tr>
<tr>
<td>r3</td>
<td>14</td>
<td>30(10)</td>
<td>6</td>
<td>120</td>
</tr>
<tr>
<td>r4</td>
<td>11</td>
<td>18(6)</td>
<td>4</td>
<td>100</td>
</tr>
</tbody>
</table>

- Here we have two groups, G1 which contains r1,r4, and G2 which contains r2,r3.
- G1 is serviced first using SCAN. By the same assumptions as on the previous slide, we first service r4, then r1.
- G2 is then serviced using SCAN. By the same assumptions as on the previous slide, we first service r2, then r3.
- Thus, the overall order of service is r4,r1,r2,r3.
Building Disk-based Media Servers

- Must service multiple clients simultaneously.
- Clients do not want just playback functionality, they also want to perform interactive operations like rewind, fast forward, pause, etc.
- For each client, the server must:
  - provide continuous playback
  - this requires filling his buffer at just the “right” rate.
  - Too fast → buffer might get overwritten.
  - Too slow → client might experience service interruption.
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>bnum($\mathcal{M}_i$)</td>
<td>Number of blocks in movie $\mathcal{M}_i$</td>
</tr>
<tr>
<td>buf($i$)</td>
<td>The total buffer space associated with disk server $i$</td>
</tr>
<tr>
<td>cyctime($i,t$)</td>
<td>the total cycle time for server $i$ at time $t$</td>
</tr>
<tr>
<td>dtr($i$)</td>
<td>The total disk bandwidth associated with disk server $i$</td>
</tr>
<tr>
<td>switchtime($i,t$)</td>
<td>the time required for disk server $i$ to switch from one client’s job to another client’s job at time $t$</td>
</tr>
<tr>
<td>cons($i,t$)</td>
<td>The consumption rate of client $C_i$ at time $t$</td>
</tr>
<tr>
<td>data($i,t$)</td>
<td>The event specification for the client $C_i$ at time $t$</td>
</tr>
<tr>
<td>timealloc($i,j,t$)</td>
<td>The time-slice allocated to client $j$ at time $t$</td>
</tr>
<tr>
<td>active($t$)</td>
<td>The set of all clients that are active at time $t$</td>
</tr>
<tr>
<td>d_active($i,t$)</td>
<td>The set of all clients that have been assigned a non-zero time-allocation by disk server $i$</td>
</tr>
<tr>
<td>$\varphi(\mathcal{M}_i,b)$</td>
<td>The set of servers that contain block $b$ of movie $\mathcal{M}_i$ according to placement mapping $\varphi$</td>
</tr>
<tr>
<td>$\mu_i(i)$</td>
<td>The set of servers handling requests by client $C_i$ at time $t$</td>
</tr>
<tr>
<td>bufreq($j,i,t$)</td>
<td>The buffer space needed at server $i$ to match the consumption rate of client $j$</td>
</tr>
<tr>
<td>$S(t)$</td>
<td>The state of a movie-on-demand system</td>
</tr>
</tbody>
</table>

Figure 1: Notation and terminology

## Parameters

- We assume that we have $n$ disk servers, $d_1, \ldots, d_n$ and $m$ movies $m_1, \ldots, m_k$.

- Each movie is a contiguous sequence of blocks. Block size can be fixed in any way.
## Client Request

- **data**\((i,t)\) is a set of pairs of the form \((m,b)\) where \(m\) is a movie, and \(b\) is a block in that movie.

- **Playback:**
  \[
  \text{data}(i,t) = \{(m,b),(m,b + 1),\ldots,m(b + r - 1)\}.
  \]
  Here \(r\) is the number of blocks the client watches per time unit.

- **Fast forward:** Suppose \(f\) is a positive integer called the "fast forward step."
  \[
  \text{data}(i,t) = \{(m,b + i \times f) | i < r \land (b + i \times f < bnum(m))\}.
  \]

- **Rewind:** Suppose \(r\) is the rewind step.
  \[
  \text{data}(i,t) = \{(m,b - i \times r) | i < r \land (b - i \times f) > 1\}.
  \]

- **Pause:** If when the user pauses, he was watching block \(b\) of movie \(m\), then:
  \[
  \text{data}(i,t) = \{(m,b)\}.
  \]
Client Request

- Reformulate $\text{data}(i, t)$ as a single quadruple
  \[ \text{data}(i, t) = (m, b, \text{len}, \text{step}) \]

- This means that client $C_i$ wishes to view the following blocks of movie $m$:
  \[ b, (b + \text{step}), (b + 2 \times \text{step}), \ldots, (b + (\text{len} - 1) \times \text{step}). \]

- **Play - normal viewing**: In this case, $\text{step} = 1$.

- **Pause**: In this case, $\text{step} = 0$.

- **Fast Forward with speed ffs**: In this case, $\text{step} = \text{ffs}$.

- **Rewind at speed rws**: In this case, $\text{step} = -\text{rws}$. 
Algorithm to Support VCR-Functionality (Sketch)

- When a set of events (transactions) occur, each of these transactions must have an associated priority.

- Initial priority assignments.

<table>
<thead>
<tr>
<th>Transaction</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exiting client</td>
<td>5</td>
</tr>
<tr>
<td>Continuing Client - normal viewing</td>
<td>4</td>
</tr>
<tr>
<td>Continuing Client - fast forwarding</td>
<td>4</td>
</tr>
<tr>
<td>Continuing Client - rewind</td>
<td>4</td>
</tr>
<tr>
<td>Continuing Client - pause</td>
<td>4</td>
</tr>
<tr>
<td>New (entering) client</td>
<td>2</td>
</tr>
</tbody>
</table>

- **Splitting**: This causes a user’s transaction to be split into two or more pieces (called **twins**). Transaction is satisfied iff both twins can be satisfied.

- **Switching**: Causes a user’s transaction (or its descendent subtransactions) to be switched from the server that was originally handling the request, to another server.
Algorithm to Support VCR-Functionality (Sketch)

- Algorithm considers clients in decreasing order of priority.
- Note that exiting clients have top priority (as they free up resources) and continuing clients have high priority too.
- Satisfy clients in decreasing order of priority.
- If a client request cannot be satisfied, and it cannot be switched to another server, then split it, reinsert into client list with slightly lower priority (3 for split containing clients).
- New client’s priority increased if not served in the loop, but can never reach 3.
Retrieving Multimedia Data from CD-ROMs
Retrieving Multimedia Data from CD-ROMs

- CD-ROM driver typically contains one platter.
- The CD-ROM contains a single spiral track, that is traversed by the read head.
- Spiral track is divided up into equal sized sectors.
- Thus, unlike a disk drive system where the disk head moves at a constant angular velocity, in the case of a CD-ROM based system, the disk head moves at a constant linear velocity across these tracks.
Example

- A particular CD-ROM contains 100 sectors.
- The read head is currently at location 58.
- A client wishes to read the sectors 10, 30, 50, 70, 90.
- This client has enough buffer to accommodate 3 sectors.
- **Possibility 1:** CD-ROM reads sector 70 first, then sector 90, then sector 10, then sector 30, and then sector 50.
- Most CD-ROM drivers do not allow this possibility.
- **Possibility 2:** Reset the disk head to point to zero, and then move the head so that sectors 10, 30, 50 get buffered. We then consume 10 and buffer 70. Next we consume sector 30 and buffer 90. Next, we consume sector 50, 70 and 90.
Reading from a CD-ROM

- Reading is done in *rounds*.
- Each round starts with the read head at location 1.
- In any given round, we attempt to read a sorted (in ascending order of sector number) set of sectors.

**Architecture of CD-ROM disk subsystem:**

```
prefetch buffer
```

![Diagram of CD-ROM architecture](drive -> prefetch buffer -> CLIENT)
Buffer Requirements

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>( ss )</td>
<td>sector size</td>
<td>bytes</td>
</tr>
<tr>
<td>( bw_d )</td>
<td>bandwidth of disk to prefetch buffer</td>
<td>bytes/sec</td>
</tr>
<tr>
<td>( dcr )</td>
<td>decompression rate</td>
<td>bytes/sec</td>
</tr>
<tr>
<td>( cr )</td>
<td>compression ratio</td>
<td>integer</td>
</tr>
<tr>
<td>( cons )</td>
<td>consumption rate of client</td>
<td>bytes/sec</td>
</tr>
<tr>
<td>( sk )</td>
<td>average seek time</td>
<td>sec</td>
</tr>
<tr>
<td>( t_{fill} )</td>
<td>buffer filling time</td>
<td>sec</td>
</tr>
</tbody>
</table>

- Need to ensure two properties:
  
  - **Continuity of playback**: The client should be able to read data from the buffer without any interruption.
  
  - **Buffer utilization**: At no time should the buffer get over-written.

- Suppose we know all the quantities in the above table, and want to determine buffer size.
Buffer Requirements

- If \( cr \) is greater than the rate at which the buffer is filled, then it is possible that at some time, the buffer is empty, but the client is trying to read something.

- If \( cr \) is less than the rate at which the buffer is filled, then it is possible that data is being written into the buffer too fast, leading to the possibility that the buffer is over-written.

- In \( t_{\text{fill}} \) seconds, the server can read
  \[
  (t_{\text{fill}} \times bw_d) \text{ bytes}
  \]
  of compressed data into the buffer.

- In one second, the client can consume
  
  \[
  cons \text{ bytes}
  \]
  of uncompressed data.

- One byte of compressed data is equivalent to \( cr \) bytes of uncompressed data.

- The maximal amount of uncompressed data consumable by the client in \( t_{\text{fill}} \) seconds is given by the equation:
  \[
  t_{\text{fill}} = \left( \delta \times t_{\text{fill}} \right) + \frac{\delta \times t_{\text{fill}} \times dcr \times cr}{cons}
  \]
  where \( \delta \) is a real number between 0 and 1, inclusive, denoting the fraction of time within the cycle of \( t_{\text{fill}} \) seconds, in which the client is decompressing compressed data.
• Solving the above equation to eliminate the variable \( \delta \), we get:

\[
t_{\text{fill}} = (\delta \times t_{\text{fill}}) + \frac{\delta \times t_{\text{fill}} \times dcr \times cr}{\text{cons}}, \quad \text{i.e.}
\]

\[
1 = \delta \times (1 + \frac{dcr \times cr}{\text{cons}}) \text{i.e.}
\]

\[
\delta = \frac{\text{cons}}{\text{cons} + dcr \times cr}.
\]

• Therefore, in \( t_{\text{fill}} + sk \) seconds, the server can write \((t_{\text{fill}} - sk) \times bw_d\) bytes (of uncompressed data) into the buffer, and the client can consume \(\frac{\delta \times t_{\text{fill}} \times dcr \times cr}{bw_c}\) bytes (of uncompressed data).

• We want

\[
(t_{\text{fill}} - sk) \times bw_d = \frac{\delta \times t_{\text{fill}} \times dcr \times cr}{bw_c}.
\]

\[
= \frac{\text{cons}}{\text{cons} + dcr \times cr} \times \frac{t_{\text{fill}} \times dcr \times cr}{\text{cons}}.
\]

\[
= \frac{t_{\text{fill}} \times dcr \times cr}{\text{cons} + dcr \times cr}.
\]

• This yields that the minimal buffer size needed is \((bw_d \times t_{\text{fill}})\). Hence, the minimal amount of buffer needed is given by the expression:

\[
bw_d \times \left( \frac{(sk - bw_d) \times (\text{cons} + dcr \times cr)}{(bw_d \times \text{cons}) + (dcr \times cr) \times (bw_d - 1)} \right).
\]
Scheduling Retrieval of Multiple Sectors from CD-ROMs

- Consider a CD-ROM server that receives requests for a set \( \{s_1, \ldots, s_k\} \) of sectors.
- Many different algorithms will be studied:
  - First Come First Serve (FCFS)
  - SCAN Algorithm
  - Scan EDF Algorithm
First Come First Serve (FCFS)

- Processes requests according to their arrival time.
- The total seek time taken to serve the entire set of $k$ requests is given by:

$$\sum_{i=1}^{k} \frac{\text{abs}(s_i - s_{i-1})}{lv}$$

where $s_0$ is defined to be sector 1, and $lv$ is the linear velocity of the read head, expressed in sectors per second.

- EX: If we consider an FCFS approach to serving requests for sectors 25, 5, 35, 15, 5, 10 in a case where the angular velocity is 2 sectors per millisecond, then the total time taken is:

$$\text{seek} = \frac{\text{abs}(25 - 1) + \text{abs}(5 - 25) + \text{abs}(35 - 5) + \text{abs}(15 - 35) + \text{abs}(5 - 15) + \text{abs}(10 - 5)}{2}$$

$$= \frac{24 + 20 + 30 + 20 + 10 + 5}{2}$$

$$= 54.5 \text{ milliseconds.}$$
SCAN Algorithm

- We first collect a set of requests, and then sort the sectors in increasing order of seek distance.
- If the read-head is initially not at the start location, then this might lead to a bidirectional sweep.
- If we reconsider the request for the sectors 25, 5, 35, 15, 5, 10 when the read head is positioned at sector 1, the SCAN algorithm would first sort these into the order: 5, 10, 15, 25, 35.
- 5 is read only once (as opposed to FCFS)
- Seek time:

\[
seek = \frac{(5 - 1) + (10 - 5) + (15 - 10) + (25 - 15) + (35 - 25)}{2}
= 17 \text{ milliseconds.}
\]
Scan EDF Algorithm

- The SCAN-EDF algorithm sorts incoming service requests on two keys: first it sorts them in ascending order of deadline; then in terms of sector number.
- If a set of service requests have the same deadline, it then processes those service requests using the SCAN algorithm.
- EX: Suppose the table below shows a set of sectors being requested, as well as the deadline by which those sectors need to be read.

<table>
<thead>
<tr>
<th>Job_Id</th>
<th>Sector</th>
<th>Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>35</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>50</td>
<td>5</td>
</tr>
</tbody>
</table>

- Two groups of requests: $G_1$ consists of jobs 2 and 5 (both of which have the same deadline, viz. 5) and $G_2$ contains jobs 1, 3, 4 (all of which have the same deadline, viz. 10).
- $G_1$ is serviced first, as it has the earlier deadline, and $G_2$ is serviced later.
- The servicing of $G_1$ causes sector 20 to be read first and sector 50 next (as the SCAN algorithm is applied within a group).
The servicing of $G_2$ causes sectors 10, 15, 35 to be read in that order. Thus, the order in which the read-head reads data is 20, 50, 10, 15, 35.
Placement of Files on a CD-ROM

- A real time file \( f \) is a triple \((\ell_f, b_f, p_f)\) where:
  - \( \ell_f \geq 0 \) is an integer called the length of the file. Intuitively, file \( f \) is broken down into \( \ell_f \) “blocks.” Different blocks in an RTF may be stored at dispersed locations on the CD-ROM.
  - \( b_f \geq 1 \) is an integer called the block size of the file. Intuitively, \( b_f \) is the number of sectors contained within a block. All these sectors are assumed to be contiguous, and in fact, all sectors of a given block are stored in contiguous sectors of the CD-ROM.
  - \( p_f \geq 0 \) is an integer called the period of file \( f \). Intuitively, \( p_f \) specifies the distance (in sectors) between the first sector of two consecutive blocks of a real time file.
Real-time file layout

- **EX:** Consider a real time file, $f_1$ described by the triple $(4, 2, 7)$. This means that file $f_1$ has 4 blocks in it, and that each block contains 2 sectors of data.

- Given a real time file $f$, we define the set of sectors occupied by the $i$'th block of file $f$ as:

$$\text{occ}_i(f) = \{j \mid st(f) + (i - 1) \times p_f \leq j \leq st(f) + (i - 1) \times p_f + b_f - 1\}.$$

As there are $\ell_f$ blocks in file $f$, it follows that the sectors occupied by file $f$ as a whole is given by the expression:

$$\text{occ}(f) = \bigcup_{i=1}^{\ell_f} \text{occ}_i(f) = \bigcup_{i=1}^{\ell_f} \{j \mid st(f) + (i - 1) \times p_f \leq j \leq st(f) + (i - 1) \times p_f + b_f - 1\}.$$

- If $st(f) = 3$ in our example, then

  $$\text{occ}_1(f_1) = \{j \mid 3 \leq j \leq 3 + 4 - 1\} = \{3, 4, 5, 6\}.$$

  $$\text{occ}_2(f_1) = \{j \mid 3 + 7 \leq j \leq 3 + 7 + 4 - 1\} = \{10, 11, 12, 13\}.$$

  $$\text{occ}_3 = \{j \mid 3 + 2 \times 7 \leq j \leq 3 + 3 \times 7 + 4 - 1\} = \{17, 18, 19, 20\}.$$
Real-time file layout

- Suppose we have a CD-ROM containing \( N \) sectors, numbered 1 through \( N \).
- Suppose we have a set \( \mathcal{F} \) of real time files.
- The start assignment problem is the problem of finding a function \( st : \mathcal{F} \rightarrow \{1, \ldots, N\} \) such that:

  **Non-Collision Axiom:** For all \( f_i, f_j \in \mathcal{F}, f_i \neq f_j \Rightarrow \text{occ}(f_i) \cap \text{occ}(f_j) = \emptyset \). If such a function \( st \) exists, then it is called a placement function.

- SAP tries to assign a start location to each file in such a way that no sector on the CD-ROM contains data belonging to two or more files.
- EX: Suppose we consider two simple real time files, \( f_1, f_2 \) characterized by the triples \((2, 2, 5)\) and \((3, 1, 4)\), respectively, and our CD-ROM has 10 sectors labeled 1 through 10. Then the following is a valid start assignment:

  \[
  st(f_1) = 2.
  \]

  \[
  st(f_2) = 1.
  \]

  Note that the function \( st \) satisfies the non-collision axiom because

  \[
  \text{occ}(f_1) = \{2, 3, 7, 8\}.
  \]
\[ \text{occ}(f_2) = \{1, 5, 9\} \]

- Korst and Pronk prove that SAP is NP-complete.
Placing 2 Real-Time Files

Algorithm 8 Placement\( (f_1, f_2) \)

\( (* f_1 : (\ell_1, b_1, p_1), f_2 : (\ell_2, b_2, p_2) *) \)

maxstart1 = \( N - (b_1 - 1) \times p_1 - (b_1 - 1) \);
maxstart2 = \( N - (b_2 - 1) \times p_2 - (b_2 - 1) \);
nogo = maxstart1 \leq 0 \lor \text{maxstart2} \leq 0;

\text{if nogo then Return "No Solution". Halt.}

\text{for } i = 1 \text{ to maxstart1 do}

\{ 
  \text{st}(f_1) = i;
  \text{for } j = 1 \text{ to maxstart2 do}

  \{ 
    \text{st}(f_2) = j;
    \text{if } \text{occ}(f_1) \cap \text{occ}(f_2) = \emptyset \text{ then}
    \text{Return } \text{st}(f_1) = i, \text{st}(f_2) = j. \text{ Halt.}
  \}

\}

\text{Return "No Solution". Halt.}
end
Tape Recording Mechanisms

- Three basic methods are used to store data onto tapes:
  - Serpentine recordings,
  - Helical recordings,
  - Transverse recordings.

- In multimedia applications, there is often a *Robotic Tape Library* and a fixed set of *Players*. The robotic arm reaches into the library, retrieves a requested tape, and inserts it into an available player for playback.

- We will study all the above aspects.
Serpentine Tapes

- Tape contains several *tracks* that are parallel to the length of the tape.
- Each track has a “track number” and a linear set of tape “blocks.”
- When reading:
  - The tape is first “rolled” forward (i.e. in the left to right direction) and the read-head of the tape driver is positioned over track 1.
  - When we reach the end of track 1, the read head gets repositioned over track 2, and we read the contents of track 2 moving from right to left.
  - When we reach the end of track 2, the read head gets re-positioned over track 3, and we read from left to right.
  - The process continues till we reach the end of the tape.
Serpentine Tapes
Serpentine Tape Traversal

**Algorithm 9 Tape_traverse(n)**

\[ i = 1; block = 1; \]
\[ \text{while } i \leq n \text{ do} \]
\[ \{ \]
\[ \text{end_of_track} = \text{false}; \]
\[ \text{while } \neg \text{end_of_track} \text{ do} \]
\[ \{ \]
\[ \text{if } i \mod 2 = 1 \text{ then block} = \text{block} + 1 \]
\[ \text{else block} = \text{block} - 1; \]
\[ \text{Read block;} \]
\[ \} \]
\[ i = i + 1; (* \text{shift tracks} *) \]
\[ \text{Position read head over track } i; \]
\[ \} \]
\[ \text{end} \]
Example

- Suppose the read head is currently positioned over track 4, and we are reading block 90 (shown as A in the figure) on this track.

- Suppose we wish to read block B of track 1 now.

- Then the tape must be rewound (as shown by the dotted lines) to the beginning of track 4, then the read head must be switched to track 1 (jumping tracks 2 and 3) and finally, we move the tape ahead to block B. (Many systems do not support such jumps).
Alternative 1: Reading from a Serpentine Tape

1. Rewind tape to the left till the read head is positioned over block 1 of track \( t_1 \). This requires traversing \((b_1 - 1)\) blocks.

2. Then re-position the read head to track \( t_2 \), jumping \( \text{abs}(t_1 - t_2) \) tracks. This requires a jump over \( \text{abs}(t_1 - t_2) \) tracks.

3. Roll tape forward till the read head is positioned over block \( b_2 \). This requires traversing \((b_2 - 1)\) blocks.

- Let \( ff \) and \( rew \) denote the fast forward and rewind speeds (in blocks/sec)
- Let \( trkspeed \) denotes the number of tracks that can be jumped per second.
- Then the time \( \tau_1 \) taken for alternative 1 is given by:

\[
\tau_1 = \frac{(b_1 - 1)}{rew} + \frac{\text{abs}(t_1 - t_2)}{trkspeed} + \frac{(b_2 - 1)}{ff}.
\]
Alternative 2: Reading from a Serpentine Tape

1. Fast forward tape to the right till the read head is positioned over the last block (denoted $nblock$) of the track. This requires traversing $(nblock - b_1)$ blocks.

2. Then re-position the read head to track $t_2$, jumping $\text{abs}(t_1 - t_2)$ tracks. This requires a jump over $\text{abs}(t_1 - t_2)$ tracks.

3. Rewind the tape (moving left) till the read head is positioned over block $b_2$. This requires traversing $(nblock - b_2)$ blocks.

- The time $\tau_2$ required for alternative 2 is given by:

$$\tau_2 = \frac{(nblock - b_1)}{rew} + \frac{\text{abs}(t_1 - t_2)}{trkspeed} + \frac{(nblock - b_2)}{ff}.$$ 

- It is important to note that the above calculations change when jumps over multiple tracks are not allowed.
Helical Tape Recording

- Tracks are “diagonal” tracks.
- Tape winds around a cylinder in a spiral fashion.
- Read/write heads are embedded in the surface of the cylinder.
- The axis across which the cylinder rotates is somewhat tilted, relative to the tape itself.
- The heads “pass” the linear movement of the tape, different parts of the tape, corresponding to angular, diagonal tracks.
Example

(a) Reading of tape by read-heads  
(b) Reading of tape by read-heads after tape movement/cylinder rotation
Handling “Bad” Sectors

- When recording a tape, a two step procedure is followed when we are writing block b:
  - first we write it, and
  - then we immediately read it back.
- If the result of reading back is the same as what was written (this test may be efficiently computed via a “checksum” operation) then it means that the block is okay, otherwise it means there is a problem.
- If there is a problem, then we add this block to a list of “bad blocks” and try to re-write the information onto the next block.
- When reading the tape, the list of “bad blocks” must be taken into account.
Robotic Tape Library Architecture

- Relative cost of obtaining the tape from the “shelf” and loading it into a tape drive is a very expensive, time consuming operation.
- Thus, minimizing such accesses is a key requirement of tape-based storage and retrieval algorithms.
Retrieval from Tape Libraries

- Suppose that we have a set \( td_1, \ldots, td_r \) of tape drives, but only one robot arm.

- When client \( C \) requests a tape, the following steps are performed, once the robot arm is available to service the request:
  
  - check if there is a free drive into which the robot can insert the desired tape. If not we must wait.

  - Once such a drive is available, we
    
    * rewind the tape currently in the drive, then
    * eject it from the drive, then
    * return it to its correct location in the tape library,
    * pick up the requested tape,
    * insert it into the tape drive, and then
    * let the tape driver access the desired blocks for the client.
Algorithm\ns ape_retrieve(tape_id, Set_of_drives)\n
Algorithm 10 Tape_retrieve(tape_id, Set_of_drives)\n
\nindrive = false;
if all drives in Set_of_drives are busy then
  \{ wait for time \( \delta \);
  Call Tape_retrieve(tape_id, Set_of_drives) again
  \}\;
else
  \{ If there is a free tape drive with tape
tape_id in it, then set TD to this tape drive;
and set indrive to true; else set TD to a new free tape drive
  \}\;
  if (~indrive) then
  \{ \n  if there is a tape \( \tau \) in TD then
  \{ rewind \( \tau \);
eject \( \tau \);
pick up \( \tau \) with the robot arm;
return \( \tau \) to the tape library;
  \}\;
  Pick up the requested tape, tape_id;
  Insert tape_id into TD;
  \};
  \};
Fast forward tape_id to the desired starting block in drive TD;
Playback requested blocks.
end
Striping

- **Granule Size**: First, we must divide the object $o$ up into equal sized granules. *What is the impact of granule size on retrieval efficiency?*

- **Stripe Width**: Next, we must determine how many tapes the object $o$ will be striped across. *What is the impact of stripe width on retrieval efficiency?*

- Example:
  - media object $o$ of size 200 MB
  - granule size 20 MB
  - stripe width 3
Example

<table>
<thead>
<tr>
<th>Tape</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>g7 g10</td>
<td>g8</td>
<td>g9</td>
</tr>
<tr>
<td></td>
<td>g1 g4</td>
<td>g2 g5</td>
<td>g3 g6</td>
</tr>
</tbody>
</table>
Stripe Table

- associates with each object \( o \):
  - its size
  - its granule size
  - its stripe width
  - the set of tapes on which it (i.e. the object \( o \)) is stored.

- EXAMPLE:

<table>
<thead>
<tr>
<th>Object</th>
<th>Size</th>
<th>Granule Size</th>
<th>Stripe Width</th>
<th>List of Tapes</th>
</tr>
</thead>
<tbody>
<tr>
<td>( o_1 )</td>
<td>200</td>
<td>40</td>
<td>3</td>
<td>( t_1, t_2, t_3 )</td>
</tr>
<tr>
<td>( o_1 )</td>
<td>200</td>
<td>40</td>
<td>3</td>
<td>( t_4, t_5, t_6 )</td>
</tr>
<tr>
<td>( o_2 )</td>
<td>100</td>
<td>25</td>
<td>3</td>
<td>( t_1, t_2, t_3 )</td>
</tr>
<tr>
<td>( o_2 )</td>
<td>100</td>
<td>20</td>
<td>3</td>
<td>( t_4, t_5, t_6 )</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Algorithm 11  **Striped_tape_retrieve**(obj)

\[
SW = S\_Table\_Stripe\_width;
\]

\[
\text{Tapes}\_\text{available} = \{\text{tape} \mid t \text{ is not currently being used}\};
\]

\[
\text{Tape}\_\text{sets}\_\text{needed} = \{S\_Table\_List \mid S\_Table\_Object = \text{obj}\};
\]

\[\text{while} (\forall t \in \text{Tape}\_\text{sets}\_\text{needed}) t \notin \text{Tapes}\_\text{available} \text{ do wait;}\]

\[
\text{Sel}\_\text{list} = \ell \text{ for some } \ell \in \text{Tape}\_\text{sets}\_\text{needed} \text{ such that } \ell \subseteq \text{Tapes}\_\text{available};
\]

\[\text{while} \ SW \ \text{drives \ are \ not \ available} \ \text{do wait;}\]

**Let** \(FTD = \text{the \ set \ of \ all \ free \ tape \ drives;}\)

\[\text{while} \ FTD \neq \emptyset \ \text{do} \]

\[\{\]

\[\text{while} \ \ell \neq \emptyset \ \text{do} \]

\[\{\]

\[\text{select \ tape} \_i \in \ell; \ \ell = \ell - \{\text{tape} \_i\};\]

\[\text{if there is a \ tape \ drive \ in } FTD \text{ with tape} \]

\[\ell \_i \ \text{in it, \ then \ set } td \_i \ \text{to this \ tape \ drive,}\]

\[\text{set } \text{indrive} \ \text{to true, \ and \ set } FTD = FTD - \{td \_i\};\]

\[\};\]

\[\text{if (} \neg \text{ indrive) \ then}\]

\[\{\]

\[\text{select \ any \ tape \ drive } td \_i \in FTD;\]

\[\text{if \ there \ is \ a \ tape } \tau \ \text{in } td \_i \ \text{then}\]

\[\{ \text{rewind } \tau; \]

\[\text{eject } \tau; \]

\[\text{pick \ up } \tau \ \text{with \ the \ robot \ arm;}\]

\[\text{return } \tau \ \text{to \ the \ tape \ library;}\]

\[\};\]

\[\text{Pick \ up \ the \ requested \ tape, } t \_i;\]

\[\text{Insert } t \_i \ \text{into } td \_i;\]

\[FTD = FTD - \{td \_i\};\]

\[\};\]

\[\text{Fast \ forward } \text{tape} \_id \ \text{to \ the \ desired \ starting \ block;}\]

**Playback** requested blocks.

\[
\}\]

\text{end}
Creating and Delivering Networked Multimedia Presentations

- When creating a multimedia presentation, three basic questions must be answered:
  - *What* objects should be included in the presentation?
  - *When* should these objects be presented to the user?
  - *Where* should the objects appear on the screen?

These questions must be answered by the presentation author who may not be a computer scientist.

- Once these have been specified, the Presentation Server must generate a *Retrieval Plan* that allows it to retrieve the objects required for the presentation keeping in mind:
  - when the objects need to be presented
  - bandwidth limitations on the network
  - resource (load, buffer) limitations on the server
  - resource (load, buffer) limitations on the client
  - mismatches between delivery rate and client consumption rate.
Constraints

Pres. Constraint Solver

Constraint Relaxation

Retrieval Scheduler

Solution exists?

yes

no

find new presentation schedule

Constraint

yes

no

Output schedule

yes

no

schedule exists?

subject

find new presentation schedule

schedules
Creating a Presentation

- Suppose objects $o_1, \ldots, o_n$ are to be presented.
- Temporal constraints specify how the objects should be laid out in time. Example:
  - The presentation of objects 1 and 2 must start at the same time.
  - The termination of objects 2 and 3 must occur at the same time.
  - Object 3 must start at the time object 1 ends.
- Spatial constraints specify how the objects should be laid out in space. Example:
  - Object 1 must be to the left of object 2.
  - Object 1 must be above object 3.
Constraint Language

- **Constants:** Every integer (positive and negative) is a constant.

- **Variables:** Associated with each $o_i$ are two integer variables, $s_i$ (denoting the “start” of $o_i$) and $e_i$ (denoting the “end” of $o_i$).

- **Elementary Terms:** Elementary terms are defined inductively as follows:
  1. Every constant is an elementary term.
  2. Every variable is an elementary term.

- **Difference Constraint:** If $t_1, t_2$ are elementary terms, and $c$ is a constant, then

\[
    t_1 - t_2 \leq c
\]

is a difference constraint.
Example

- $e_1 - s_1 \leq 10$.
  This constraint says that object $o_1$ must end with 10 time units of the time its presentation starts.

- $s_2 - e_1 \leq 0; e_1 - s_2 \leq 0$.
  These two constraints jointly state that object $o_2$’s presentation starts as soon as object $o_1$’s presentation ends.

- $s_2 - e_1 \leq 3$.
  This constraint says that object $o_2$’s presentation starts within 3 time units of the end of the presentation of object $o_1$. 
Definitions

- A temporal presentation is a pair $TP = (O, DC)$ where $O$ is a finite set of objects, and $DC$ is a finite set of difference constraints in the constraint language generated by $O$.

- Suppose $DC$ is a set of difference constraints over $O = \{o_1, \ldots, o_n\}$ of virtual objects. A solution of $DC$ is an assignment of integers to each of the variables, $s_1, \ldots, s_n, e_1, \ldots, e_n$, which makes all the constraints true.

- EX:

$$s_1 - s_2 = 0.$$ $$e_1 - e_2 = 0.$$ $$s_3 - e_1 = 0.$$ $$s_3 - e_2 = 0.$$ $$e_3 - s_3 = 10.$$ $$s_4 - e_2 = 5.$$ $$e_4 - e_3 \leq 4.$$ $$e_3 - e_4 \leq -2.$$  

The figure on the next page shows these constraints.

- Sets of constraints can have 0, 1 or many solutions. EX:
<table>
<thead>
<tr>
<th>Solution</th>
<th>$s_1$</th>
<th>$e_1$</th>
<th>$s_2$</th>
<th>$e_2$</th>
<th>$s_3$</th>
<th>$e_3$</th>
<th>$s_4$</th>
<th>$e_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma_1$</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>10</td>
<td>10</td>
<td>20</td>
<td>15</td>
<td>22</td>
</tr>
<tr>
<td>$\sigma_2$</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>10</td>
<td>10</td>
<td>20</td>
<td>15</td>
<td>23</td>
</tr>
<tr>
<td>$\sigma_3$</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>10</td>
<td>10</td>
<td>20</td>
<td>15</td>
<td>24</td>
</tr>
<tr>
<td>$\sigma_4$</td>
<td>3</td>
<td>10</td>
<td>3</td>
<td>10</td>
<td>10</td>
<td>20</td>
<td>15</td>
<td>22</td>
</tr>
<tr>
<td>$\sigma_5$</td>
<td>3</td>
<td>10</td>
<td>3</td>
<td>10</td>
<td>10</td>
<td>20</td>
<td>15</td>
<td>23</td>
</tr>
<tr>
<td>$\sigma_6$</td>
<td>3</td>
<td>10</td>
<td>3</td>
<td>10</td>
<td>10</td>
<td>20</td>
<td>15</td>
<td>24</td>
</tr>
</tbody>
</table>

- A temporal presentation $TP = (O, DC)$ is feasible iff the set, $DC$, of difference constraints, has a solution $\sigma$. In this case, $\sigma$ is said to be a schedule for $TP$. 
Constraints Viewed Graphically
Definition

- The start and end of $\sigma$, denoted $\text{start}(\sigma)$ and $\text{end}(\sigma)$, are defined to be:

\[
\text{start}(\sigma) = \min(\{\sigma(s_i) | 1 \leq i \leq n\}).
\]
\[
\text{end}(\sigma) = \max(\{\sigma(e_i) | 1 \leq i \leq n\}).
\]

- We are assuming that the constraints $\{s_i-e_i \leq 0 | 1 \leq i \leq n\}$ are included in DC.

- Returning to our example:

<table>
<thead>
<tr>
<th>Solution</th>
<th>Start</th>
<th>End</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma_1$</td>
<td>0</td>
<td>22</td>
</tr>
<tr>
<td>$\sigma_2$</td>
<td>0</td>
<td>23</td>
</tr>
<tr>
<td>$\sigma_3$</td>
<td>0</td>
<td>24</td>
</tr>
<tr>
<td>$\sigma_4$</td>
<td>3</td>
<td>22</td>
</tr>
<tr>
<td>$\sigma_5$</td>
<td>3</td>
<td>23</td>
</tr>
<tr>
<td>$\sigma_6$</td>
<td>3</td>
<td>24</td>
</tr>
</tbody>
</table>
How should we create presentation schedules?

• There is a classic algorithm called the Bellman Ford Algorithm.
• Takes as input, a set S of difference constraints.
• Convert S to a graph.
• S has a solution iff the graph has no negative cycles.
• A negative cycle is a cycle whose edges sum up to a negative number.
Example

- Constraints:

\[
\begin{align*}
    s_1 - e_1 &\leq -3, \\
    e_1 - s_1 &\leq 5, \\
    s_2 - e_2 &\leq -4, \\
    e_2 - s_2 &\leq 6, \\
    e_1 - s_2 &\leq 0, \\
    s_2 - e_1 &\leq 2.
\end{align*}
\]

- Graph:

- Shortest path from the start node to each node:
### Table: Shortest Path (S.P.) from `start` to `N`

<table>
<thead>
<tr>
<th>Node</th>
<th>Shortest Path (S.P.) from <code>start</code> to <code>N</code></th>
<th>Cost of S.P.</th>
</tr>
</thead>
<tbody>
<tr>
<td>(s_1)</td>
<td><code>start \rightarrow e_2 \rightarrow s_2 \rightarrow e_1 \rightarrow s_1</code></td>
<td>-7.</td>
</tr>
<tr>
<td>(e_1)</td>
<td><code>start \rightarrow e_2 \rightarrow s_2 \rightarrow e_1</code></td>
<td>-4</td>
</tr>
<tr>
<td>(s_2)</td>
<td><code>start \rightarrow e_2 \rightarrow s_2</code></td>
<td>-4</td>
</tr>
<tr>
<td>(e_2)</td>
<td><code>start \rightarrow e_2</code></td>
<td>0</td>
</tr>
</tbody>
</table>

- No negative cycles.

![Diagram](attachment:image.png)
Theorems

- Let the cost of a path be the sum of the costs of edges along the path.
- A cycle in $G$ is a sequence of nodes $v_1, \ldots, v_k$ such that $v_k = v_1$ and for all $1 \leq j < k$, $(v_j, v_{j+1})$ is an edge in the graph.
- $v_1, \ldots, v_k$ is said to be a negative cycle in graph $G$ iff $\sum_{j=1}^{k-1} \varphi(v_j, v_{j+1})$ is a negative number.
- A set DC of difference constraints has no solution iff $G_{DC}$ has a negative cycle.
- Suppose a graph has no negative cycles. Then the cost of the shortest path from the start node to that node ($s_i$ or $e_i$) directly allows us to get a solution to the original set of difference constraints.
Approach

The Bellman Ford algorithm associates with each node $N$ in $G_{DC}$, the following two fields:

- \textit{Bestval}: This specifies the cheapest path from the start node to the node $N$, that has been discovered thus far.
- \textit{Bestpar}: $Bestpar(N)$ specifies the immediate predecessor of node $N$ along the best path from the start node to node $N$, that we have found thus far.
Initialization

Algorithm 12 Initialize($V, E$);

(* $n = \text{card}(V); m = \text{card}(E); *$
for $i = 1$ to $n$ do

{ 
    $\text{Bestval}(v_i) = \infty$;
    $\text{Bestpar}(v_i) = \text{NIL}$

} 
end
Refinement

Takes two nodes $v_i, v_j$ as input, and determines if the shortest path from the start node to $v_j$ is made cheaper by “going through” $v_i$.

**Algorithm 13** $\text{Refine}(v_i, v_j)$;

$\text{new} = \text{Bestval}(v_i) + \omega(v_i, v_j);$  
\textbf{if} $\text{new} < \text{Bestval}(v_j)$ \textbf{then} 
\{ 
  $\text{Bestval}(v_j) = \text{new};$
  $\text{Bestpar}(v_j) = v_i$
\} 
\textbf{end}
Main Algorithm

Algorithm 14  \textit{Bellman-Ford}(V, E, \omega);

\begin{algorithm*}
\begin{algorithmic}
  \State $n = \text{card}(V); m = \text{card}(E);$ \\
  \Comment\textit{we assume $V = \{v_1, \ldots, v_n\}$};  \\
  \Comment\textit{we assume $E = \{(v^1_j, v^2_j), \ldots, (v^1_m, v^2_m)\}$};  \\
  \textbf{Initialize}($V, E$);  \\
  \For{$i = 1$ \textbf{to} $(n - 1)$}  \\
    \For{$j = 1$ \textbf{to} $m$}  \\
      \textbf{Refine}($v^1_j, v^2_j$);  \\
    \EndFor  \\
    \EndFor  \\
    \If{$\text{Bestval}(v^1_j) > \text{Bestval}(v^2_j) + \omega(v^1_j, v^2_j)$}  \\
      \textbf{Return} \textit{error and Halt};  \\
    \EndIf  \\
  \EndFor  \\
  \For{$i = 1$ \textbf{to} $n$}  \\
    \textbf{Return} $v_i.\text{Name} = v_i.\text{Bestval}$;  \\
  \EndFor  \\
\end{algorithmic}
\end{algorithm*}

}\textit{algo *}

Examples

\begin{align*}
  s_1 - e_1 & \leq -3. \\
  e_1 - s_1 & \leq 5. \\
  s_2 - e_2 & \leq -4. \\
  e_2 - s_2 & \leq 6. \\
  e_1 - s_2 & \leq 0. \\
  s_2 - e_1 & \leq 2.
\end{align*}
Example, Continued
Example, Continued
Example, Continued
Example, Continued
Spatial Constraints

Given a set \( \{v_0, \ldots, v_n\} \) of virtual objects, we associate, with each object \( v_0 \), the following variables:

- \( W_i \): width of the window in which object \( v_0 \) is shown;
- \( H_i \): height of the window in which object \( v_0 \) is shown;
- \( X_i \): x-coordinate of the lower left corner of the window in which object \( v_0 \) is shown;
- \( Y_i \): y-coordinate of the lower left corner of the window in which object \( v_0 \) is shown.
- \( R_i \): This variable is equal to \( (W_i + X_i) \) and denotes the right vertical edge of the window containing \( v_0 \).
- \( U_i \): This variable is equal to \( (H_i + Y_i) \) and denotes the right vertical edge of the window containing \( v_0 \).

Difference constraints using these variables may be specified and solved in exactly that same way as for temporal constraints.
Example

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>$vo_i$ is to the left of $vo_j$</td>
<td>$R_i - R_j \leq 0$.</td>
</tr>
<tr>
<td>$vo_i$ is to the right of $vo_j$</td>
<td>$R_j - R_i \leq 0$.</td>
</tr>
<tr>
<td>$vo_i$ is above $vo_j$</td>
<td>$U_j - Y_i \leq 0$</td>
</tr>
<tr>
<td>$vo_i$ is below $vo_j$</td>
<td>$Y_i - U_j \leq 0$.</td>
</tr>
</tbody>
</table>
Distributed Media Servers

- We have seen how to create *Presentation Schedules*.
- A presentation schedule specifies *when* the display of media objects should start, and when they should end.
- A given set of presentation constraints may have zero, one or more presentation schedules that satisfy the presentation constraints.
- Not all presentation schedules are *deliverable*.
- To how an object \(o\) at time \(t\), the server retrieving \(o\) must obtain commitments of resources from several sources.
- These commitments include buffer resources, bandwidth resources, and load guarantees.
- Retrieval scheduling takes a presentation schedule as input, and tries to create a retrieval schedule that ensures that objects are retrieved into the client’s buffer before the client’s associated presentation schedule requires the object to be displayed.
Retrieval Schedules: Architecture
Distributed Multimedia Server Systems

- A *customer* interested in retrieving data at or by a given time contacts his/her local server.
- This local originating server is then charged with the responsibility of creating a Retrieval Schedule by interacting with other servers on the network.
- Each server on the network has an associated body of media objects stored on its local disk.
- A *placement mapping* is a mapping, \( \varphi \) that takes as input,
  1. a movie \( m_i \in \text{MOVIE} \) and
  2. a block number, \( 1 \leq b \leq \text{bnum}(m_i) \), and returns as output, a subset of \( V \). Here, \( V \) is the set of all servers available in the DMSS system.
## Example Placement Mapping

<table>
<thead>
<tr>
<th>Movie</th>
<th>Available Blocks</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
<th>S6</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>{B1–B5, B7, B9}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M2</td>
<td>{B1, B4–B12, B19}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M3</td>
<td>{B1–B15, B19–B25}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Movie</th>
<th>Available Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>{B1–B5, B8–B15}</td>
</tr>
<tr>
<td>M2</td>
<td>{B1–B10, B12–B20}</td>
</tr>
<tr>
<td>M3</td>
<td>{B1–B15, B19–B25}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Movie</th>
<th>Available Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>{B1–B5, B10–B14, B17, B19–B22}</td>
</tr>
<tr>
<td>M2</td>
<td>{B1–B4, B11–B14, B19–B22–B25}</td>
</tr>
<tr>
<td>M3</td>
<td>{B1–B6}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Movie</th>
<th>Available Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>{B1–B6, B11–B17, B19, B21–B25}</td>
</tr>
<tr>
<td>M2</td>
<td>{B1–B5, B8–B14, B20}</td>
</tr>
<tr>
<td>M3</td>
<td>{B1–B2, B10–B17, B20, B22}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Movie</th>
<th>Available Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>{B1–B6, B10–B14, B17, B19, B22}</td>
</tr>
<tr>
<td>M2</td>
<td>{B1–B4–B11, B14–B19, B22–B25}</td>
</tr>
<tr>
<td>M3</td>
<td>{B1–B6}</td>
</tr>
</tbody>
</table>
Server/client Parameters

- Server Buffer Size
- Customer Consumption Rate
- Customer Buffer Size
- Server-server Bandwidth
- Customer-Server Bandwidth
Data structures: Commitment Record List

- Commitment Record List: Each server maintains a commitment record list, specifying what commitments it has made. When a request for services is received by the server, the commitment record list is consulted to determine if the service request can be satisfied. If so (i.e. when a new commitment is made), the commitment record list is updated to reflect the new commitment.

| BegCom | This specifies the start time of a commitment. |
| FinCom | This specifies the finish time of a commitment. |
| Client | This could either be a customer, or another server to whom a commitment is being made. |
| Movie  | This specifies what movie forms part of the commitment. |
| BlockSt| This specifies the starting block of the movie. |
| BlockEnd| This specifies the ending block of the movie associated with this commitment. |
| BWCom | This specifies the amount of bandwidth committed to this commitment. |
Example Commitment Record List

- EX:

<table>
<thead>
<tr>
<th>BegCom</th>
<th>FinCom</th>
<th>Client</th>
<th>Movie</th>
<th>BlockSt</th>
<th>BlockEnd</th>
<th>BWCom</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>15</td>
<td>John</td>
<td>The Rope</td>
<td>B5</td>
<td>B35</td>
<td>0.5 MB/sec</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>s4</td>
<td>The Abyss</td>
<td>B25</td>
<td>B45</td>
<td>0.25 MB/sec</td>
</tr>
<tr>
<td>15</td>
<td>25</td>
<td>s5</td>
<td>Dracula</td>
<td>B50</td>
<td>B70</td>
<td>0.5 MB/sec</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Data structures: Retrieval Record

• Given any customer request \( r \), the originating server for that customer creates a set of retrieval records for it. The retrieval records specify the retrieval plan put together by the originating server.

• Retrieval record format:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>Orig</strong></td>
</tr>
<tr>
<td>2</td>
<td><strong>Target</strong></td>
</tr>
<tr>
<td>3</td>
<td><strong>Movie</strong></td>
</tr>
<tr>
<td>4</td>
<td><strong>Start</strong></td>
</tr>
<tr>
<td>5</td>
<td><strong>End</strong></td>
</tr>
<tr>
<td>6</td>
<td><strong>Reqtime</strong></td>
</tr>
<tr>
<td>7</td>
<td><strong>ConOK</strong></td>
</tr>
<tr>
<td>8</td>
<td><strong>BWAssign</strong></td>
</tr>
<tr>
<td>9</td>
<td><strong>DelivSt</strong></td>
</tr>
</tbody>
</table>

• (Continued on next page).
## Retrieval Record Format, Continued

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td><strong>DelivEnd</strong></td>
<td>For each block ( b_w ) where ( r.\text{Start} \leq w \leq r.\text{End} ),</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( r.\text{DelivEnd}[w] = r.\text{DelivSt} + \frac{(w-r.\text{Start}+1) \times \text{size}}{r.\text{BW Assign}} )</td>
</tr>
<tr>
<td>11</td>
<td><strong>CustShipSt</strong></td>
<td>For each block ( b_w ) where ( r.\text{Start} \leq w \leq r.\text{End} ),</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( r.\text{CustShipSt}[w] \geq r.\text{DelivEnd}[w] )</td>
</tr>
<tr>
<td>12</td>
<td><strong>CustShipEnd</strong></td>
<td>For each block ( b_w ) where ( r.\text{Start} \leq w \leq r.\text{End} ),</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( r.\text{CustShipEnd}[w] = r.\text{CustShipSt}[w] + \frac{\text{size}}{bw(r.\text{Orig}, C)} )</td>
</tr>
<tr>
<td>13</td>
<td><strong>CustConsStart</strong></td>
<td>For each block ( b_w ) where ( r.\text{Start} \leq w \leq r.\text{End} ),</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( r.\text{CustConsStart}[w] \geq r.\text{CustShipEnd}[w] )</td>
</tr>
<tr>
<td>14</td>
<td><strong>CustConsEnd</strong></td>
<td>For each block ( b_w ) where ( r.\text{Start} \leq w \leq r.\text{End} ),</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( r.\text{CustConsEnd}[w] = r.\text{CustConsStart}[w] + \frac{\text{size}}{ew(C)} )</td>
</tr>
</tbody>
</table>
Retrieval Plans

- A retrieval plan is a finite sequence $r_1, \ldots, r_n$ of retrieval records such that for all $1 \leq i < n$, $r_i.BlockEnd + 1 = r_{i+1}.BlockSt$.

- The retrieval plan must satisfy the following constraints:

  - **Continuity of Playback**: At all times $t$ such that $r_1.CustConsStart \leq t \leq r_n.CustConsEnd$, there must exist exactly one block $b_i$ of the movie such that $b_i$ is being viewed by the customer.

  - **Server-Server Bandwidth Constraint**: At all times $t$ such that $r_1.DelivSt \leq t \leq r_1.DelivEnd$, and for all network connections between 2 servers $(s_1, s_2)$, the total assignment of bandwidth to jobs using the channel $(s_1, s_2)$ should be less than or equal to the total physical bandwidth of the channel.

  - **Server-Customer Bandwidth Constraint**: At all times $t$ such that $r_1.CustShipSt \leq t \leq r_n.CustConsEnd$, the total assignment of bandwidth to jobs using the channel between the customer and the originating server should be less than or equal to the total physical bandwidth of the channel.

  - **Customer Buffer Constraint**: At all times $t$ such that $r_1.CustShipSt \leq t \leq r_n.CustConsEnd$, the total number of blocks that have been shipped by the originating server to the customer, but that have not yet
been consumed by the customer, should occupy space less than or equal to the total buffer space available at the customer’s machine.

- **Originating Server Buffer Constraint:** At all times $t$ such that $r_1.\text{DelivEnd} \leq t \leq r_n.\text{CustShipEnd}$, the total number of blocks that have been shipped by a remote server to the originating server, but that have not yet been shipped to the customer, should occupy space less than or equal to the total buffer space available at the customer’s machine.
Optimal Retrieval Plans

Many possible objective functions, but we consider two.

- **Minimizing customer wait time** $RP$ is said to be *wait-minimal* iff there is no other retrieval plan $RP'$ that can satisfy the request such that $RP'$ has a strictly smaller wait time than $RP$.

- **Minimizing the access bandwidth** $RP$ is said to be *access-bandwidth* minimal iff the sum of the number of disk accesses and the number of network accesses is minimal, i.e. there is no other retrieval plan $RP'$ such that the sum of the number of disk accesses and the number of network accesses made by $RP'$ is strictly less than the sum of the number of disk accesses and the number of network accesses made by $RP$.

- Many other objective functions may also be used, and the algorithm we provide below takes an objective function as input, and always produces an optimal retrieval plan w.r.t. that objective function as output.
Basic idea for creating Retrieval Plans

- Initial client retrieval request is a triple, \( \text{Req}(C') = (m, b_1, b_2) \). Intuitively, \( \text{Req}(C') = (m, b_1, b_2) \) denotes the request, by user \( C \), for all blocks \( b \) of movie \( m \) such that \( b_1 \leq b \leq b_2 \).

- The \textit{Problem} of creating retrieval plans has an IMPLICIT RETRIEVAL TREE associated with it.

- Each node of the tree is labeled with a segment of blocks to be retrieved. Thus, the root of the tree is labeled with \( (b_1, b_2) \).

- If node \( N \) is attempting to get blocks \( (b'_1, b'_2) \) and there is a server that can deliver \( (b'_1, b'_3) \) for \( b'_3 \leq b'_2 \), then there is an edge labeled with the server’s name (plus some other parameters) and the child is labeled with \( (b'_3+1, b'_2) \) indicating that these are yet to be accomplished.

- The edge is also labeled with a set of time-bandwidth pairs – intuitively, if \( (t, bw) \) is in this set, it means that it is possible to deliver the set of blocks starting at time \( t \) and bandwidth \( bw \). Bandwidth and buffer constraints taken into account here.

- Details of the algorithm are in the book, pages 384–385.