Centralized databases
- One to a few cores, shared memory

Client-server,
- One server machine executes work on behalf of multiple client machines.

Parallel databases
- Many core shared memory
- Shared disk
- Shared nothing

Distributed databases
- Geographical distribution
- Schema/data heterogeneity
### Storage Hierarchy (Cont.)

- **primary storage**: Fastest media but volatile (cache, main memory).
- **secondary storage**: next level in hierarchy, non-volatile, moderately fast access time
  - Also called **on-line storage**
  - E.g., flash memory, magnetic disks
- **tertiary storage**: lowest level in hierarchy, non-volatile, slow access time
  - also called **off-line storage** and used for **archival storage**
  - e.g., magnetic tape, optical storage
  - Magnetic tape
    - Sequential access, 1 to 12 TB capacity
    - A few drives with many tapes
    - Juke boxes with petabytes (1000’s of TB) of storage
### Storage Interfaces

- Disk interface standards families
  - SATA (Serial ATA)
    - SATA 3 supports data transfer speeds of up to 6 gigabits/sec
  - SAS (Serial Attached SCSI)
    - SAS Version 3 supports 12 gigabits/sec
  - NVMe (Non-Volatile Memory Express) interface
    - Works with PCIe connectors to support lower latency and higher transfer rates
    - Supports data transfer rates of up to 24 gigabits/sec

- Disks usually connected directly to computer system
- In Storage Area Networks (SAN), a large number of disks are connected by a high-speed network to a number of servers
- In Network Attached Storage (NAS) networked storage provides a file system interface using networked file system protocol, instead of providing a disk system interface

### Flash Storage

- NOR flash vs NAND flash
- NAND flash
  - used widely for storage, cheaper than NOR flash
  - requires page-at-a-time read (page: 512 bytes to 4 KB)
    - 20 to 100 microseconds for a page read
    - Not much difference between sequential and random read
  - Page can only be written once
    - Must be erased to allow rewrite

- Solid state disks
  - Use standard block-oriented disk interfaces, but store data on multiple flash storage devices internally
  - Transfer rate of up to 500 MB/sec using SATA, and up to 3 GB/sec using NVMe PCIe
Flash Storage (Cont.)

- Erase happens in units of **erase block**
  - Takes 2 to 5 milliseconds
  - Erase block typically 256 KB to 1 MB (128 to 256 pages)
- **Remapping** of logical page addresses to physical page addresses avoids waiting for erase
- **Flash translation table** tracks mapping
  - also stored in a label field of flash page
  - remapping carried out by **flash translation layer**

- After 100,000 to 1,000,000 erases, erase block becomes unreliable and cannot be used
  - **wear leveling**

RAID

- **RAID: Redundant Arrays of Independent Disks**
  - disk organization techniques that manage a large numbers of disks, providing a view of a single disk of
    - **high capacity** and **high speed** by using multiple disks in parallel,
    - **high reliability** by storing data redundantly, so that data can be recovered even if a disk fails
  - The chance that some disk out of a set of $N$ disks will fail is much higher than the chance that a specific single disk will fail.
    - E.g., a system with 100 disks, each with MTTF of 100,000 hours (approx. 11 years), will have a system MTTF of 1000 hours (approx. 41 days)
    - Techniques for using redundancy to avoid data loss are critical with large numbers of disks
Hardware: Key Takeaways

- Database must reside on non-volatile storage
  - Can cache in faster storage
- Non-volatile storage slow
  - But accessing a lot not much different than accessing a little
  - Therefore we read/write as large blocks (typically 4kb)
- Abstract performance as: $\alpha + \beta b$
  - $\alpha$ is seek time (abstraction of read/write setup overhead)
  - $\beta$ is transfer rate
  - $b$ is block size
- Rotating media: seek can dominate (but caching, sequential reads reduce this)
- Solid state: transfer dominates
  - but erasure, protocol overheads make “seek” more than you’d expect
- Writes typically worse than reads
  - Not “done” until safe in non-volatile storage, so reduces caching benefits

Storage / Buffer Management

10 February 2021
File Organization

- The database is stored as a collection of files. Each file is a sequence of records. A record is a sequence of fields.
- One approach
  - Assume record size is fixed
  - Each file has records of one particular type only
  - Different files are used for different relations
    This case is easiest to implement; will consider variable length records later
- We assume that records are smaller than a disk block.

Fixed-Length Records

- Simple approach:
  - Store record \( i \) starting from byte \( n \times (i - 1) \), where \( n \) is the size of each record.
  - Record access is simple but records may cross blocks
    - Modification: do not allow records to cross block boundaries

| record 0 | 10101 | Srinivasan | Comp. Sci. | 65000 |
| record 1 | 12121 | Wu         | Finance    | 90000 |
| record 2 | 15151 | Mozart     | Music      | 40000 |
| record 3 | 22222 | Einstein   | Physics    | 95000 |
| record 4 | 32343 | El Said    | History    | 60000 |
| record 5 | 33456 | Gold       | Physics    | 87000 |
| record 6 | 45565 | Katz       | Comp. Sci. | 75000 |
| record 7 | 58583 | Califiri   | History    | 62000 |
| record 8 | 76543 | Singh      | Finance    | 80000 |
| record 9 | 76766 | Crick      | Biology    | 72000 |
| record 10| 83821 | Brandt     | Comp. Sci. | 92000 |
| record 11| 98345 | Kim        | Elec. Eng. | 80000 |
Fixed-Length Records

- Deletion of record $i$: alternatives:
  - move records $i + 1, \ldots, n$ to $i, \ldots, n - 1$
  - move record $n$ to $i$
  - do not move records, but link all free records on a free list

Record 3 deleted

<table>
<thead>
<tr>
<th>Record</th>
<th>Student</th>
<th>Major</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10101</td>
<td>Srinivasan</td>
<td>65000</td>
</tr>
<tr>
<td>1</td>
<td>12121</td>
<td>Wu</td>
<td>90000</td>
</tr>
<tr>
<td>2</td>
<td>15151</td>
<td>Mozart</td>
<td>40000</td>
</tr>
<tr>
<td>4</td>
<td>32343</td>
<td>El Said</td>
<td>60000</td>
</tr>
<tr>
<td>5</td>
<td>33456</td>
<td>Gold</td>
<td>87000</td>
</tr>
<tr>
<td>6</td>
<td>45565</td>
<td>Katz</td>
<td>75000</td>
</tr>
<tr>
<td>7</td>
<td>58583</td>
<td>Califere</td>
<td>62000</td>
</tr>
<tr>
<td>8</td>
<td>76543</td>
<td>Singh</td>
<td>80000</td>
</tr>
<tr>
<td>9</td>
<td>76766</td>
<td>Crick</td>
<td>72000</td>
</tr>
<tr>
<td>10</td>
<td>83821</td>
<td>Brandt</td>
<td>92000</td>
</tr>
<tr>
<td>11</td>
<td>98345</td>
<td>Kim</td>
<td>80000</td>
</tr>
</tbody>
</table>

Fixed-Length Records

- Deletion of record $i$: alternatives:
  - move records $i + 1, \ldots, n$ to $i, \ldots, n - 1$
  - move record $n$ to $i$
  - do not move records, but link all free records on a free list

Record 3 deleted and replaced by record 11

<table>
<thead>
<tr>
<th>Record</th>
<th>Student</th>
<th>Major</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10101</td>
<td>Srinivasan</td>
<td>65000</td>
</tr>
<tr>
<td>1</td>
<td>12121</td>
<td>Wu</td>
<td>90000</td>
</tr>
<tr>
<td>2</td>
<td>15151</td>
<td>Mozart</td>
<td>40000</td>
</tr>
<tr>
<td>11</td>
<td>98345</td>
<td>Kim</td>
<td>80000</td>
</tr>
<tr>
<td>4</td>
<td>32343</td>
<td>El Said</td>
<td>60000</td>
</tr>
<tr>
<td>5</td>
<td>33456</td>
<td>Gold</td>
<td>87000</td>
</tr>
<tr>
<td>6</td>
<td>45565</td>
<td>Katz</td>
<td>75000</td>
</tr>
<tr>
<td>7</td>
<td>58583</td>
<td>Califere</td>
<td>62000</td>
</tr>
<tr>
<td>8</td>
<td>76543</td>
<td>Singh</td>
<td>80000</td>
</tr>
<tr>
<td>9</td>
<td>76766</td>
<td>Crick</td>
<td>72000</td>
</tr>
<tr>
<td>10</td>
<td>83821</td>
<td>Brandt</td>
<td>92000</td>
</tr>
</tbody>
</table>
Fixed-Length Records

- Deletion of record $i$: alternatives:
  - move records $i+1, \ldots, n$ to $i, \ldots, n-1$
  - move record $n$ to $i$
  - do not move records, but link all free records on a free list

Variable-Length Records

- Variable-length records arise in database systems in several ways:
  - Storage of multiple record types in a file.
  - Record types that allow variable lengths for one or more fields such as strings (varchar)
  - Record types that allow repeating fields (used in some older data models).

- Attributes are stored in order
- Variable length attributes represented by fixed size (offset, length), with actual data stored after all fixed length attributes
- Null values represented by null-value bitmap
Variable-Length Records: Slotted Page Structure

- Slotted page header contains:
  - number of record entries
  - end of free space in the block
  - location and size of each record
- Records can be moved around within a page to keep them contiguous with no empty space between them; entry in the header must be updated.
- Refer to a record by block and entry#

Storing Large Objects

- E.g., blob/clob types
- Records must be smaller than pages
- Alternatives:
  - Store as files in file systems
  - Store as files managed by database
  - Break into pieces and store in multiple tuples in separate relation
    - PostgreSQL TOAST
©Silberschatz, Korth and Sudarshan

Organization of Records in Files

- **Heap** – record can be placed anywhere in the file where there is space
- **Sequential** – store records in sequential order, based on the value of the search key of each record
- In a **multitable clustering file organization** records of several different relations can be stored in the same file
  - Motivation: store related records on the same block to minimize I/O
- **B*-tree file organization**
  - Ordered storage even with inserts/deletes
  - More on this in Chapter 14
- **Hashing** – a hash function computed on search key; the result specifies in which block of the file the record should be placed
  - More on this in Chapter 14

Heap File Organization

- Records can be placed anywhere in the file where there is free space
- Records usually do not move once allocated
- Important to be able to efficiently find free space within file
- **Free-space map**
  - Array with 1 entry per block. Each entry is a few bits to a byte, and records fraction of block that is free
  - In example below, 3 bits per block, value divided by 8 indicates fraction of block that is free

\[
\begin{array}{cccccccc}
4 & 2 & 1 & 4 & 7 & 3 & 6 & 5 \\
1 & 2 & 0 & 1 & 1 & 0 & 5 & 6 \\
\end{array}
\]

  - Can have second-level free-space map
  - In example below, each entry stores maximum from 4 entries of first-level free-space map

\[
\begin{array}{cccc}
4 & 7 & 2 & 6 \\
\end{array}
\]

- Free space map written to disk periodically, OK to have wrong (old) values for some entries (will be detected and fixed)