

# CS 541 Hashing

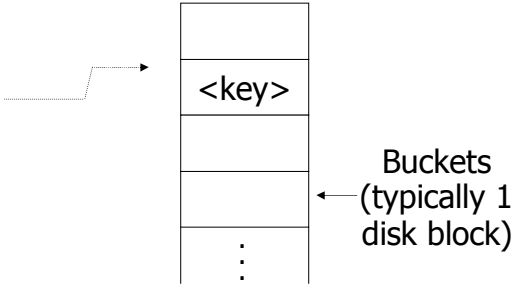
October 11, 2002

Chris Clifton - CS541

1

Hashing

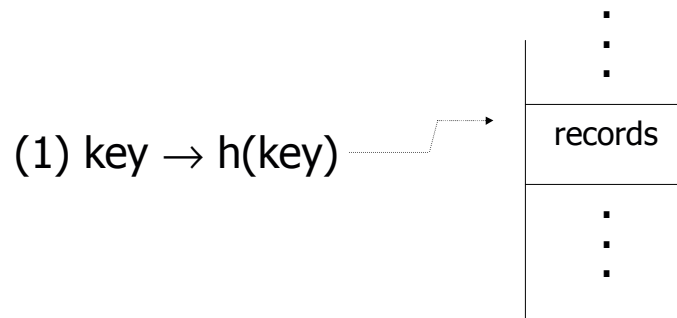
$key \rightarrow h(key)$



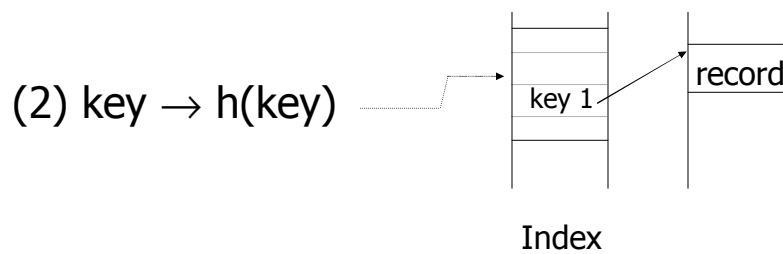
Buckets  
(typically 1  
disk block)

Fall 2002 Chris Clifton - CS541 2

Two alternatives



Two alternatives



- Alt (2) for “secondary” search key

## Example hash function

- Key = 'x<sub>1</sub> x<sub>2</sub> ... x<sub>n</sub>'  $n$  byte character string
- Have  $b$  buckets
- $h$ : add  $x_1 + x_2 + \dots + x_n$ 
  - ◆ compute sum modulo  $b$

Fall 2002

Chris Clifton - CS541

5

- This may not be best function ...
- Read Knuth Vol. 3 if you really need to select a good function.

Good hash  $\Rightarrow$  Expected number of  
function: keys/bucket is the  
same for all buckets

Fall 2002

Chris Clifton - CS541

6

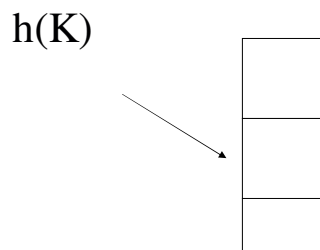
Within a bucket:

- Do we keep keys sorted?
- Yes, if CPU time critical  
& Inserts/Deletes not too frequent

Fall 2002

Chris Clifton - CS541

7

Next: example to illustrate  
inserts, overflows, deletes

Fall 2002

Chris Clifton - CS541

8

EXAMPLE 2 records/bucket

INSERT:

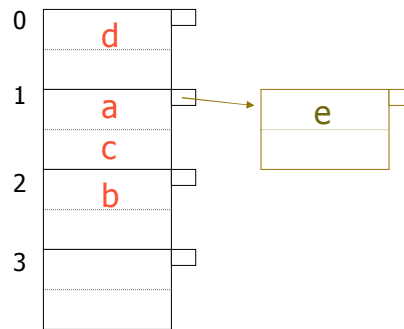
$h(a) = 1$

$h(b) = 2$

$h(c) = 1$

$h(d) = 0$

$h(e) = 1$



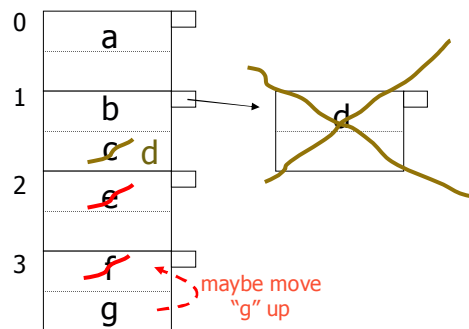
EXAMPLE: deletion

Delete:

e

f

c



Rule of thumb:

- Try to keep space utilization between 50% and 80%

$$\text{Utilization} = \frac{\text{\# keys used}}{\text{total \# keys that fit}}$$

- If < 50%, wasting space
- If > 80%, overflows significant depends on how good hash function is & on # keys/bucket

Fall 2002

Chris Clifton - CS541

11

How do we cope with growth?

- Overflows and reorganizations
- Dynamic hashing

- 
- Extensible
  - Linear

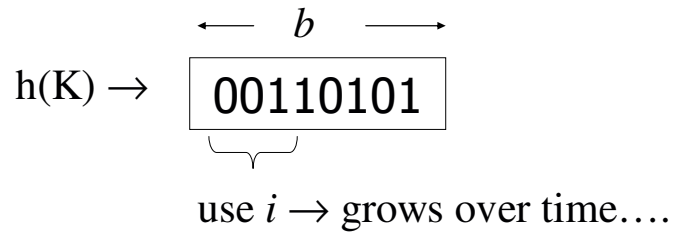
Fall 2002

Chris Clifton - CS541

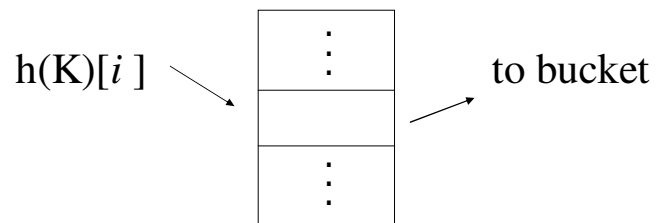
12

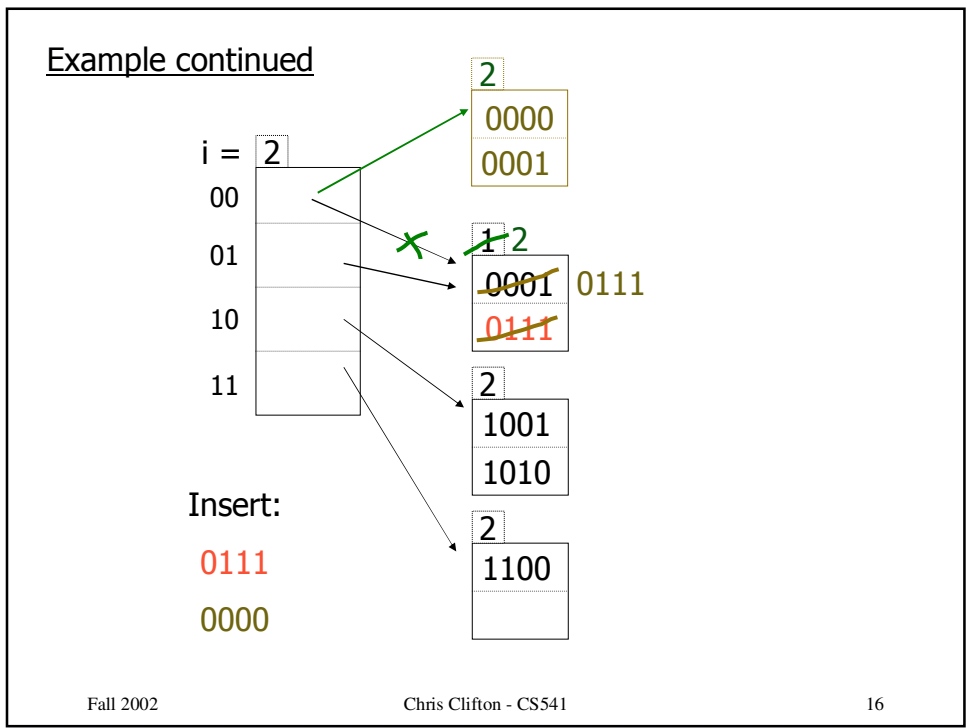
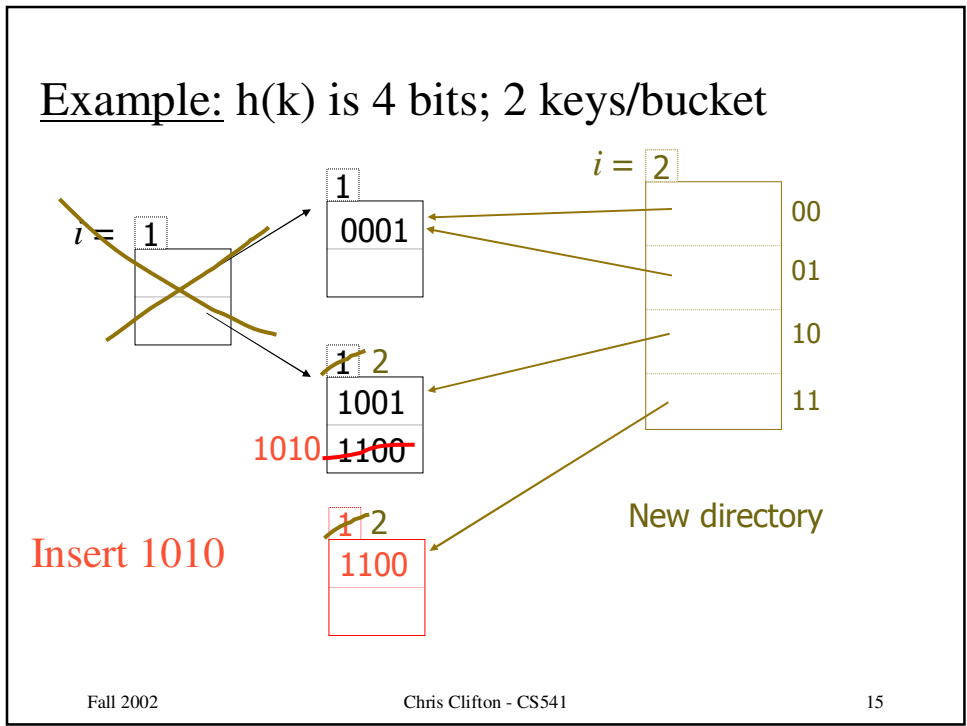
Extensible hashing: two ideas

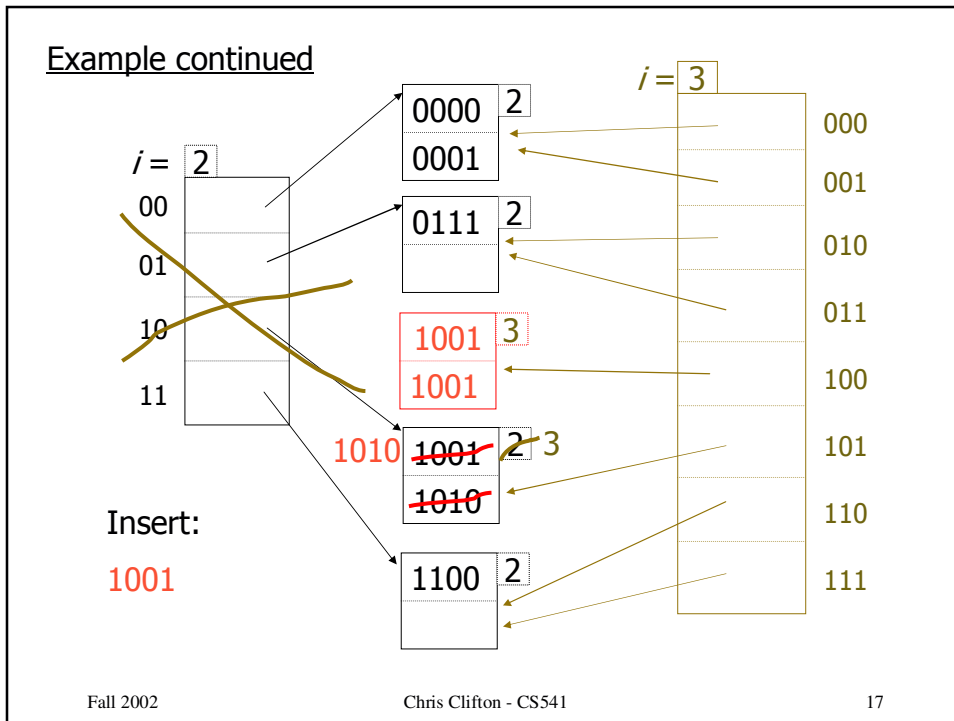
(a) Use  $i$  of  $b$  bits output by hash function



(b) Use directory







Extensible hashing: deletion

- No merging of blocks
- Merge blocks and cut directory if possible (Reverse insert procedure)

Fall 2002 Chris Clifton - CS541 18

### Deletion example:

- Run thru insert example in reverse!

### Summary      Extensible hashing

- ⊕ Can handle growing files
  - with less wasted space
  - with no full reorganizations
- ⊖ Indirection  
(Not bad if directory in memory)
- ⊖ Directory doubles in size  
(Now it fits, now it does not)

### Linear hashing

- Another dynamic hashing scheme

Two ideas:

(a) Use  $i$  low order bits of hash

$\leftarrow b \rightarrow$   
01110101  
 grows  $\leftarrow$   $\underbrace{\hspace{2em}}_i$

(b) File grows linearly

Fall 2002
Chris Clifton - CS541
21

Example  $b=4$  bits,  $i=2$ , 2 keys/bucket

- insert 0101
- can have overflow chains!

0000	0101		
1010	1111		
00	01	10	11

$m = 01$  (max used block)

← Future growth buckets

**Rule** If  $h(k)[i] \leq m$ , then  
 look at bucket  $h(k)[i]$   
 else, look at bucket  $h(k)[i] - 2^{i-1}$

Fall 2002
Chris Clifton - CS541
22

Example  $b=4$  bits,  $i=2$ , 2 keys/bucket

• insert 0101

0000  
~~1010~~

0101  
~~1111~~

1010

1111

00 01 10 11

$m = \cancel{01}$  (max used block)

~~10~~

11

Future growth buckets

Fall 2002 Chris Clifton - CS541 23

Example Continued: How to grow beyond this?

$i = \cancel{2} 3$

0000  
~~1010~~

~~0101~~  
~~0101~~

1010

1111

100

0101  
0101

000 01 100 011 100 101 ...

~~100~~

~~101~~

$m = \cancel{11}$  (max used block)

~~100~~

101

Fall 2002 Chris Clifton - CS541 24

☞ When do we expand file?

- Keep track of:  $\frac{\text{\# used slots}}{\text{total \# of slots}} = U$
- If  $U > \text{threshold}$  then increase  $m$   
(and maybe  $i$ )

Fall 2002

Chris Clifton - CS541

25

## Summary Linear Hashing

- ⊕ Can handle growing files
  - with less wasted space
  - with no full reorganizations
- ⊕ No indirection like extensible hashing
- ⊖ Can still have overflow chains

Fall 2002

Chris Clifton - CS541

26



Next:

- Indexing vs Hashing
- Index definition in SQL
- Multiple key access

**Indexing vs Hashing**

- Hashing good for probes given key

e.g.,  
`SELECT ...`  
`FROM R`  
`WHERE R.A = 5`

## Indexing vs Hashing

- INDEXING (Including B Trees) good for Range Searches:

e.g., SELECT

FROM R

WHERE R.A > 5

Fall 2002

Chris Clifton - CS541

31

## Index definition in SQL

- Create index name on rel (attr)
- Create unique index name on rel (attr)

└──→ defines candidate key

- Drop INDEX name

Fall 2002

Chris Clifton - CS541

32

**Note** CANNOT SPECIFY TYPE OF  
INDEX

(e.g. B-tree, Hashing, ...)

OR PARAMETERS

(e.g. Load Factor, Size of Hash,...)

... at least in SQL...

**Note** ATTRIBUTE LIST  $\Rightarrow$  MULTIKEY INDEX

(next)

e.g., CREATE INDEX foo ON R(A,B,C)

## Multi-key Index

Motivation: Find records where

DEPT = “Toy” AND SAL > 50k

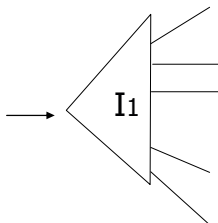
Fall 2002

Chris Clifton - CS541

35

### Strategy I:

- Use one index, say Dept.
- Get all Dept = “Toy” records  
and check their salary



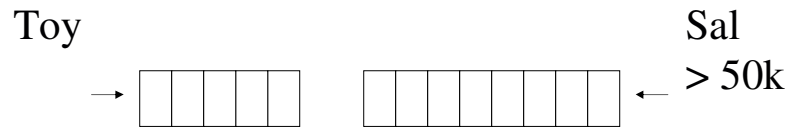
Fall 2002

Chris Clifton - CS541

36

Strategy II:

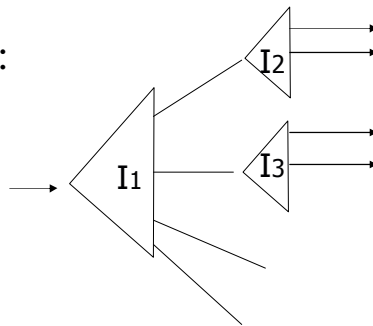
- Use 2 Indexes; Manipulate Pointers

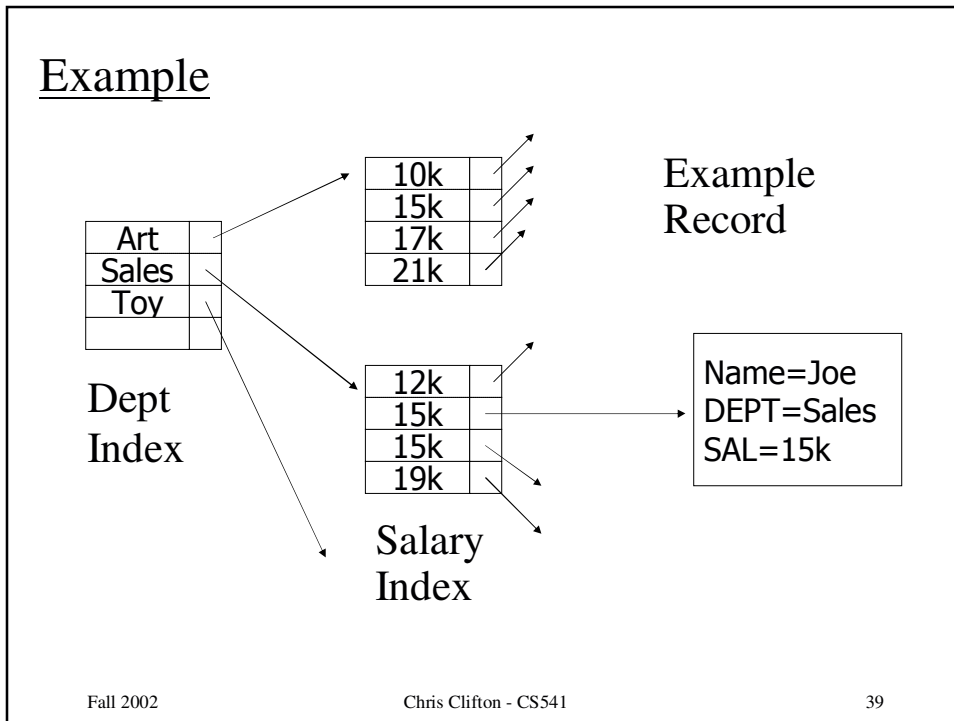


Strategy III:

- Multiple Key Index

One idea:





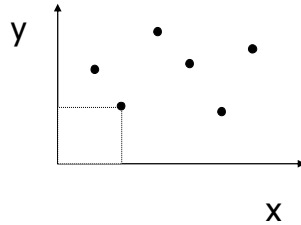
For which queries is this index good?

- Find RECs Dept = “Sales”  $\wedge$  SAL=20k
- Find RECs Dept = “Sales”  $\wedge$  SAL  $\geq$  20k
- Find RECs Dept = “Sales”
- Find RECs SAL = 20k

Fall 2002 Chris Clifton - CS541 40

## Interesting application:

- Geographic Data



DATA:

$\langle X_1, Y_1, \text{Attributes} \rangle$

$\langle X_2, Y_2, \text{Attributes} \rangle$

⋮

Fall 2002

Chris Clifton - CS541

41

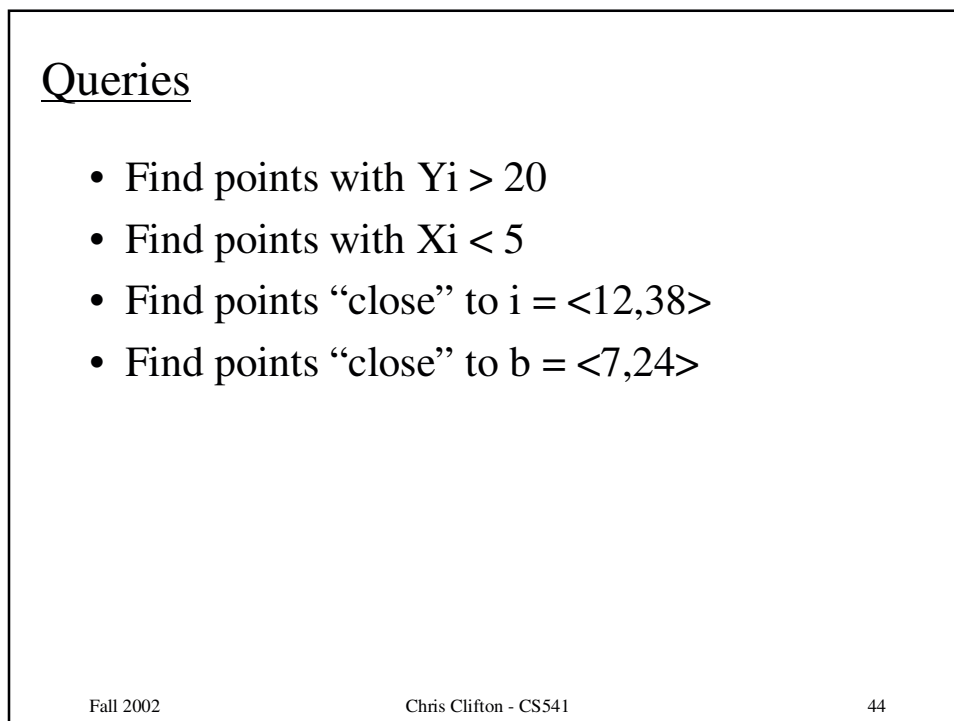
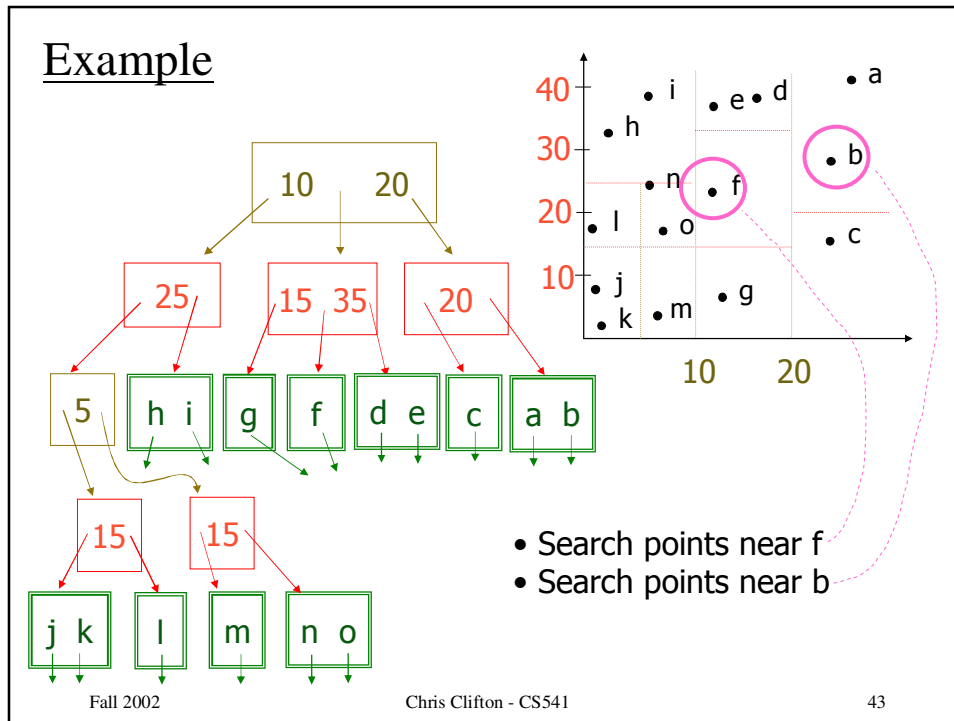
## Queries:

- What city is at  $\langle X_i, Y_i \rangle$ ?
- What is within 5 miles from  $\langle X_i, Y_i \rangle$ ?
- Which is closest point to  $\langle X_i, Y_i \rangle$ ?

Fall 2002

Chris Clifton - CS541

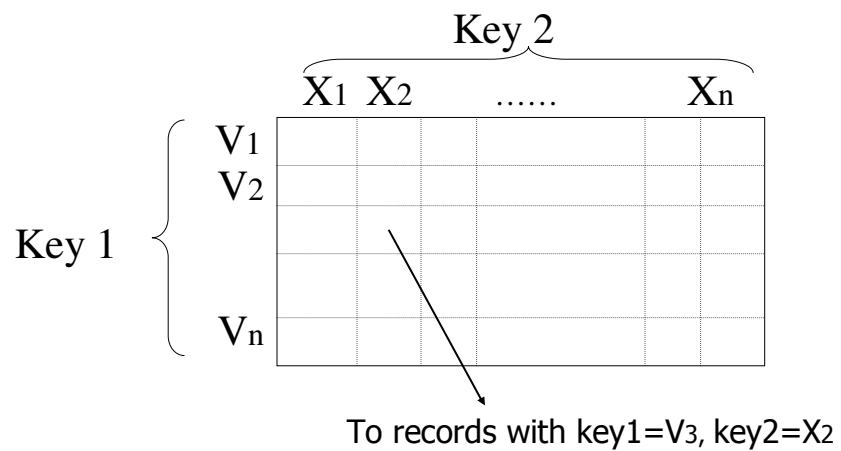
42



- Many types of geographic index structures have been suggested
  - Quad Trees
  - R Trees

### Two more types of multi key indexes

- Grid
- Partitioned hash

Grid Index

Fall 2002

Chris Clifton - CS541

47

CLAIM

- Can quickly find records with
  - ◆ key 1 =  $V_i \wedge$  Key 2 =  $X_j$
  - ◆ key 1 =  $V_i$
  - ◆ key 2 =  $X_j$
- And also ranges....
  - ◆ E.g., key 1  $\geq V_i \wedge$  key 2  $< X_j$

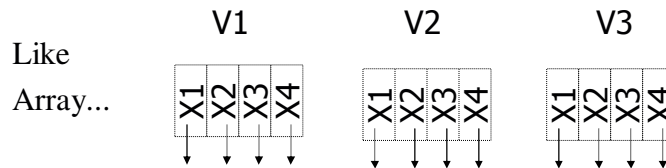
Fall 2002

Chris Clifton - CS541

48

☞ But there is a catch with Grid Indexes!

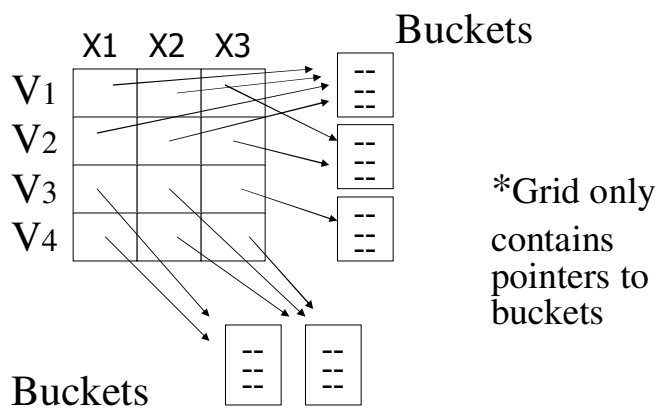
- How is Grid Index stored on disk?



Problem:

- Need regularity so we can compute position of  $\langle V_i, X_j \rangle$  entry

Solution: Use Indirection



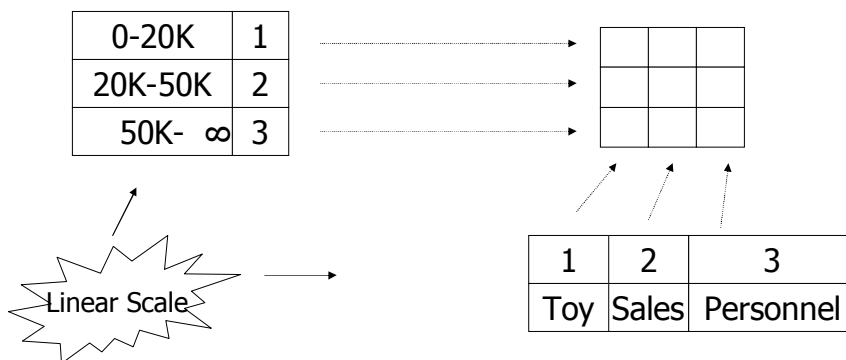
With indirection:

- Grid can be regular without wasting space
- We do have price of indirection

Can also index grid on value ranges

Salary

Grid



## Grid files

- ⊕ Good for multiple-key search
- ⊖ Space, management overhead  
(nothing is free)
- ⊖ Need partitioning ranges that evenly split keys

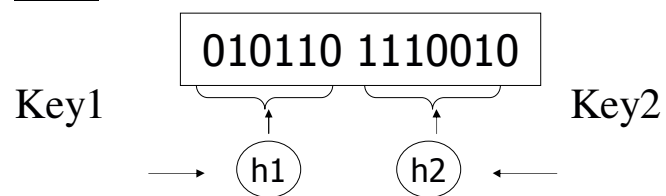
Fall 2002

Chris Clifton - CS541

53

## Partitioned hash function

Idea:



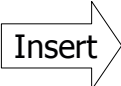
Fall 2002

Chris Clifton - CS541

54

**EX:**

h1(toy)	=0	000	
h1(sales)	=1	001	<Fred>
h1(art)	=1	010	
.		011	
.			
h2(10k)	=01	100	
h2(20k)	=11	101	<Joe><Sally>
h2(30k)	=01	110	
h2(40k)	=00	111	
.			


 <Fred,toy,10k>,<Joe,sales,10k>  
 <Sally,art,30k>

Fall 2002 Chris Clifton - CS541 55

h1(toy)	=0	000	<Fred>
h1(sales)	=1	001	<Joe><Jan>
h1(art)	=1	010	<Mary>
.		011	
.			
h2(10k)	=01	100	<Sally>
h2(20k)	=11	101	
h2(30k)	=01	110	<Tom><Bill>
h2(40k)	=00	111	<Andy>
.			

- Find Emp. with Dept. = Sales  $\wedge$  Sal=40k

Fall 2002 Chris Clifton - CS541 56

h1(toy)	=0	000	<Fred>
h1(sales)	=1	001	<Joe><Jan>
h1(art)	=1	010	<Mary>
.		011	
.		100	<Sally>
h2(10k)	=01	101	
h2(20k)	=11	110	<Tom><Bill>
h2(30k)	=01	111	<Andy>
h2(40k)	=00		
.			

• Find Emp. with Sal=30k

look here

Fall 2002 Chris Clifton - CS541 57

h1(toy)	=0	000	<Fred>
h1(sales)	=1	001	<Joe><Jan>
h1(art)	=1	010	<Mary>
.		011	
.		100	<Sally>
h2(10k)	=01	101	
h2(20k)	=11	110	<Tom><Bill>
h2(30k)	=01	111	<Andy>
h2(40k)	=00		
.			

• Find Emp. with Dept. = Sales

look here

Fall 2002 Chris Clifton - CS541 58

## Summary

### Post hashing discussion:

- Indexing vs. Hashing
- SQL Index Definition
- Multiple Key Access
  - Multi Key Index
- Variations: Grid, Geo Data
- Partitioned Hash

## Reading Chapter 5

- Skim the following sections:
  - ◆ 5.3.6, 5.3.7, 5.3.8
  - ◆ 5.4.2, 5.4.3, 5.4.4
- Read the rest

## The **BIG** picture....

- Chapters 2 & 3: Storage, records, blocks...
- Chapter 4 & 5: Access Mechanisms
  - Indexes
    - B trees
    - Hashing
    - Multi key
- Chapter 6 & 7: Query Processing

