CSCD43: Database Systems Technology

Lecture 9

Wael Aboulsaadat

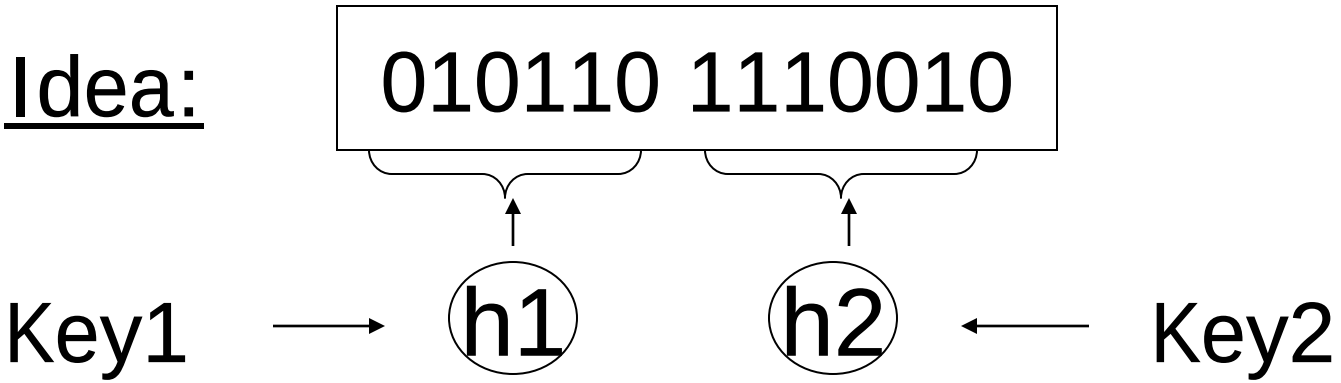
Acknowledgment: these slides are based on Prof. Garcia-Molina & Prof. Ullman slides accompanying the course's textbook.

Topics

- Conventional Indexes
- B-trees
- Hashing Schemes

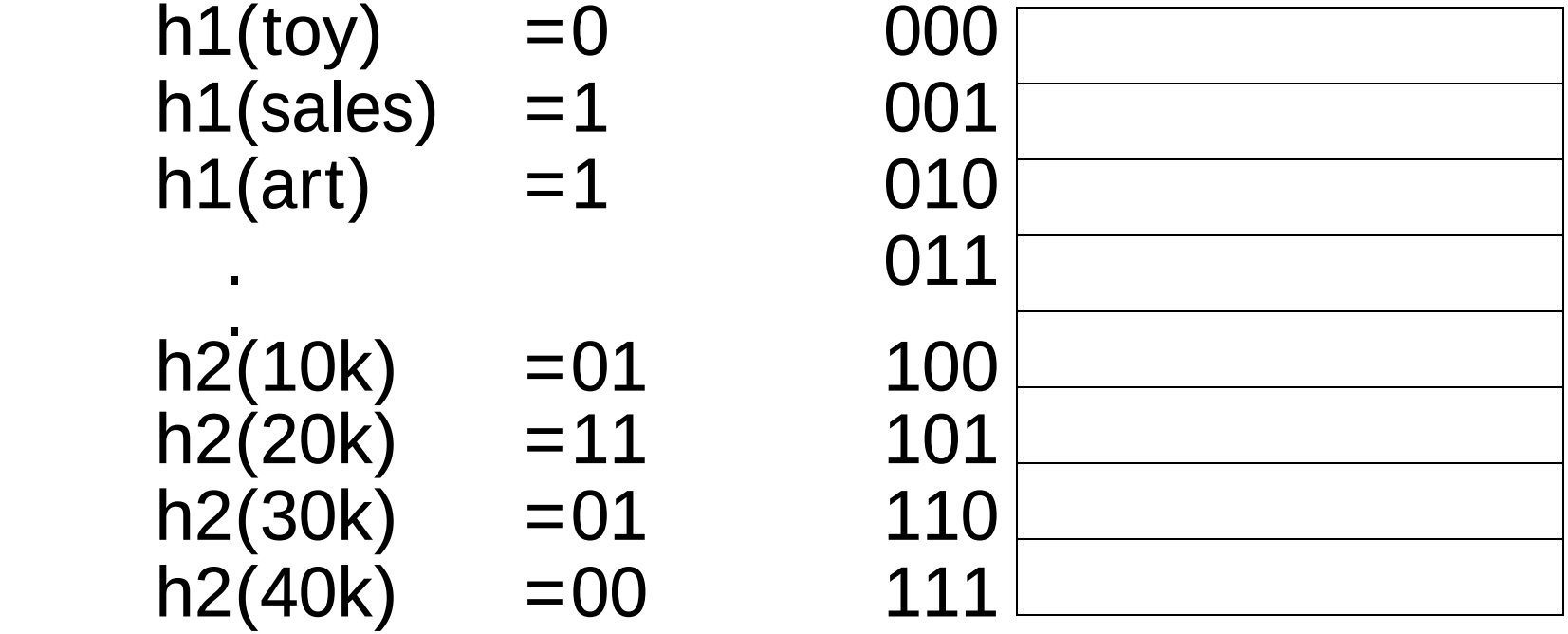
→ Multidimensional Indexes

2) Partitioned hash function



Partitioned hash function - insertion

h1(toy) =0
h1(sales) =1
h1(art) =1
.
.
.
h2(10k) =01
h2(20k) =11
h2(30k) =01
h2(40k) =00
.
.
.



Insert

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

Partitioned hash function - insertion

h1(toy) =0
h1(sales) =1
h1(art) =1
.
.
.
h2(10k) =01
h2(20k) =11
h2(30k) =01
h2(40k) =00
.
.
.

000	
001	<Fred>
010	
011	
100	
101	<Joe><Sally>
110	
111	

Insert

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

Partitioned hash function - lookup

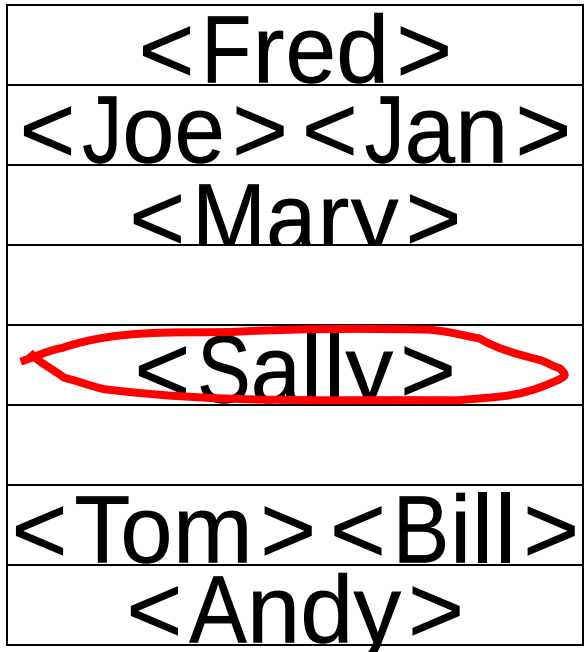
h1(toy) =0
h1(sales) =1
h1(art) =1
.
.
h2(10k) =01
h2(20k) =11
h2(30k) =01
h2(40k) =00
.
.

000	<Fred>
001	<Joe><Jan>
010	<Mary>
011	
100	<Sally>
101	
110	<Tom><Bill>
111	<Andy>

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

Partitioned hash function - lookup

h1(toy) =0
h1(sales) =1
h1(art) =1
.
.
h2(10k) =01
h2(20k) =11
h2(30k) =01
h2(40k) =00
.
.



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

Partitioned hash function - lookup

h1(toy) =0
h1(sales) =1
h1(art) =1
.
.
h2(10k) =01
h2(20k) =11
h2(30k) =01
h2(40k) =00
.
.

000	<Fred>
001	<Joe><Jan>
010	<Mary>
011	
100	<Sally>
101	
110	<Tom><Bill>
111	<Andy>

- Find Emp. with Sal=30k

Partitioned hash function - lookup

h1(toy) =0
h1(sales) =1
h1(art) =1
.
.
h2(10k) =01
h2(20k) =11
h2(30k) =01
h2(40k) =00
.
.

000	<Fred>
001	<Joe><Jan>
010	<Mary>
011	
100	<Sally>
101	
110	<Tom><Bill>
111	<Andy>

- Find Emp. with Sal=30k

look here

Partitioned hash function - lookup

h1(toy) =0
h1(sales) =1
h1(art) =1
.
.
h2(10k) =01
h2(20k) =11
h2(30k) =01
h2(40k) =00
.
.

000	<Fred>
001	<Joe><Jan>
010	<Mary>
011	
100	<Sally>
101	
110	<Tom><Bill>
111	<Andy>

- Find Emp. with Dept. = Sales

Partitioned hash function - lookup

h1(toy) =0
h1(sales) =1
h1(art) =1
.
.
h2(10k) =01
h2(20k) =11
h2(30k) =01
h2(40k) =00
.
.

000	<Fred>
001	<Joe><Jan>
010	<Mary>
011	
100	<Sally>
101	
110	<Tom><Bill>
111	<Andy>

- Find Emp. with Dept. = Sales

look here

Partitioned Hash function

⊕ Good for multiple-key search

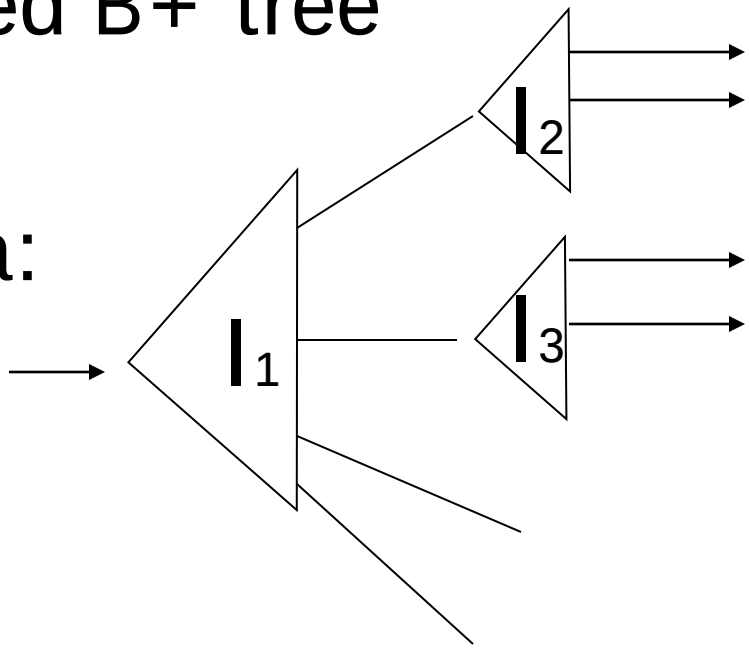
⊕ Easier to maintain than Grid files

⊖ Can not ask nearest neighbor queries!

3) MultiKey Index

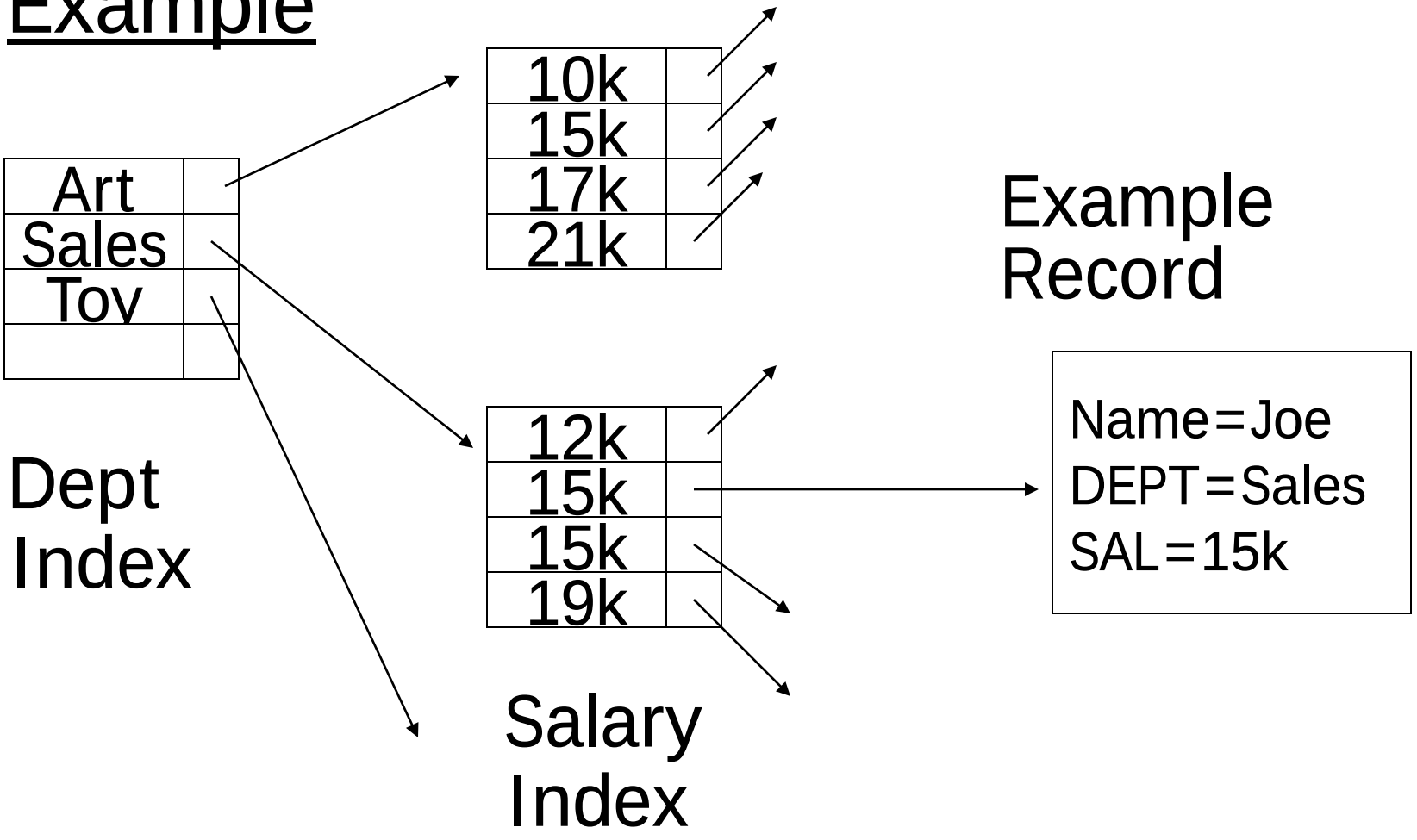
- Modified B+ tree

One idea:

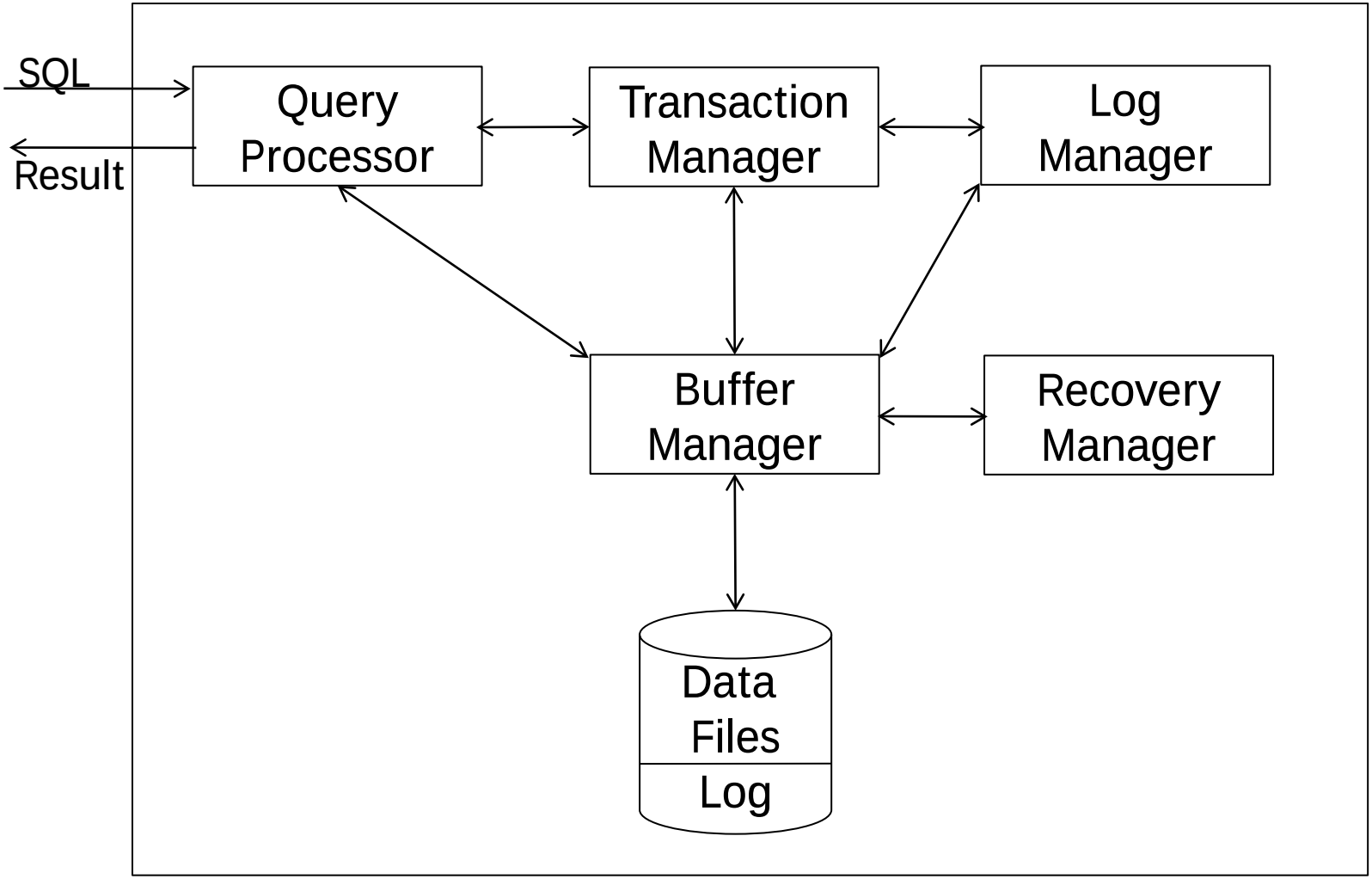


CREATE INDEX foo ON R(A,B,C)

Example

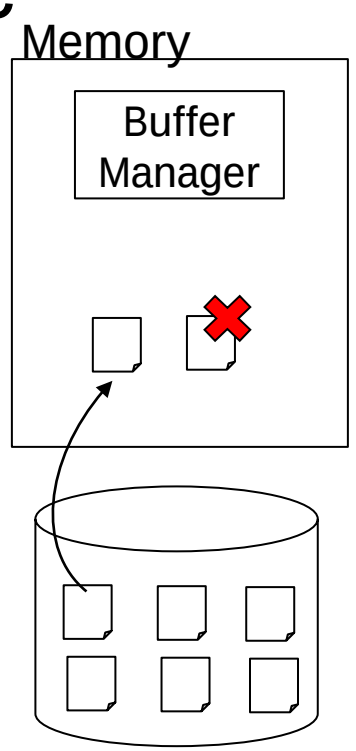

Art
Sales
Toy
10k
15k
17k
21k
12k
15k
15k
19k
Dept Index
Salary Index
Example Record
Name=Joe
DEPT=Sales
SAL=15k

DBMS Architecture



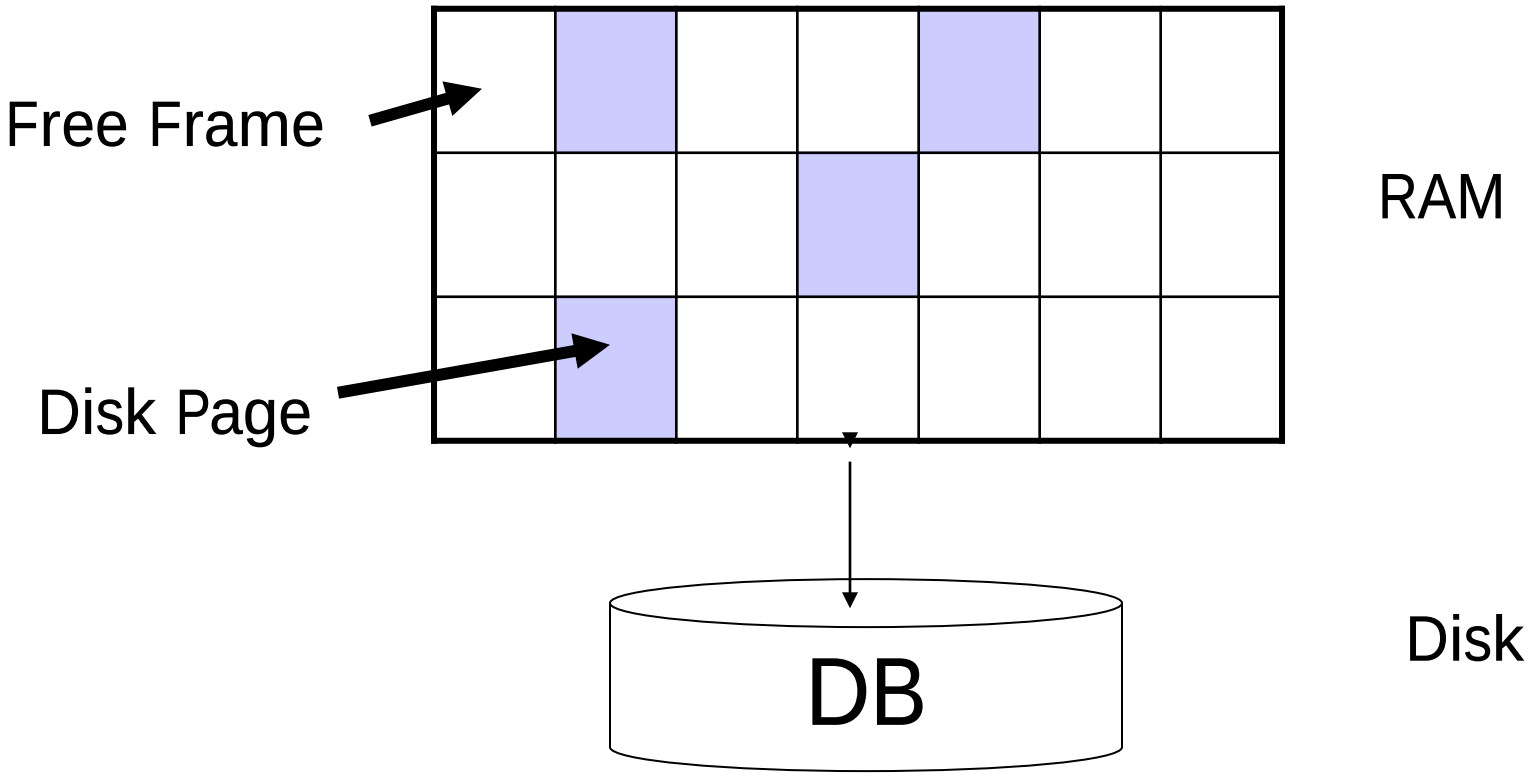
Buffer Manager

- Accessing data files with every insert/update/delete is unrealistic
- DBMS have a special module that virtualizes those pages!



Buffer Manager

- Replacement Strategy



Goals

- Main goal: minimize the physical I/O for a given buffer size (minimize the **page faults**).

- Maximize the **hit rate** (average ratio of the number of times requested pages are found in the buffer over the total number of requests made in a unit of time).

Goals - cont'd

- To be effective, the buffering scheme must keep pages with high frequency of access in the buffer.
  - E.g. Five-minute rule

    pages whose access frequency is 5 minutes or less should be kept in buffer.
- Generally, Buffer Manager does not know in advance which data will be active and which will be passive.
- The likelihood of access to a page in a near future is usually determined by observing the actual request pattern made in the past.

Goals - cont'd

- *Locality of reference (can be stated in two ways):*

1. *Locality of active data* - pages that have been requested recently will likely be requested again in the near future.

2. *Locality of passive data* - pages that have not been requested recently will likely not be requested soon again.

Role of Buffer Manager

- **Sharing**. Pages in buffer are accessible to all threads, thus avoiding redundant read and copy operations.
- **Synchronization**. Each request results in latching the page. But, the actual synchronization is responsibility of the access modules.
- **Asynchronous writes.** The access modules inform BM if their page access will result in an update of the page. Actual writing to disk is done by BM, often at a time when the update transaction is long gone.
- **Durable storage**. BM must coordinate writing of pages to disk with Log Manager

Operational mode

- All requested data pages must first be placed into the buffer pool.
- pin_count is used to keep track of number of transactions that are using the page
  - 0 means no body is using it
- dirty is used a flag (dirty bit) to indicate that a page has been modified since read from disk
  - Need to flush it to disk if the page is to be evicted from pool
- Page is an array of bytes where the actual is located
  - Need to interpret this bytes as the int, char, Date data types supported by SQL
    - This is very complex and tricky! (you are not doing that)

Buffer replacement

- If we need to bring a page from disk, we need to find a frame in the buffer to hold it
- Buffer pool keeps track on the number of frames in use
  - List of frames that are free (Linked list of free frame nums)
- If there is a free frame, we use it
  - Remove from list of free frame
  - Increment the pin_count
  - Store the data page into the byte array (page field)
- If the buffer is full, we need a policy to decide which page will be evicted

Buffer access & replacement algorithm

- Upon request of page X do
  - Look for page X in buffer pool
  - If found, ++pin_count, then return it
  - Else, determine if there is a free frame Y in the pool
  - If frame Y is found
    - Increment its pin_count
    - Read page from disk into the frame's byte array
    - Return it
  - Else, use a replacement policy to find a frame Z to replace
    - Z must have pin_count == 0
  - Increment the pin_count in Z
  - If dirty bit is set, write data currently in Z to disk
  - Read the new page into the byte array in the frame Z
  - Return it, else wait or abort transaction (insufficient resources)

Some issues

- Need to make sure pin_count is 0
  - Nobody is using the frame
- Need to write the data to disk in dirty bit is true
- This latter approach is called Lazy update
  - Write to disk only when you have to!!!
  - Careful, if power fails, you are in trouble.
  - DBMS need to periodically flush pages to disk
    - Force write
- If no page is found with pin_count equal to 0, then either:
  - Wait until one is freed
  - Abort the transaction (insufficient resources)

Buffer Replacement Policies

- LRU – Least Recently Used
  - Evicts the page that is least recently used page in the pool.
  - Can be implemented by having a queue with the frame numbers.
  - Head of the queue is the LRU
  - Each time a page is used it must be removed from current queue position and put back at the end
    - This queue need a method erase() that can erase stuff from the middle of the queue
- LRU is the most widely used policy for buffer replacement
  - Most cache managers also use it

Buffer Replacement Policies – cont'd

- Most Recently Used
  - Evicts the page that was most recently accessed
  - Can be implemented with a priority queue
- FIFO
  - Pages are replaced in a strict First-In-First Out
  - Can be implemented with a FIFO List (queue in the strict sense)
- Random
  - Pick any page at random for replacement