CSE 132C
Database System Implementation

Arun Kumar

Topic 1: Data Storage Management

Chapters 8 and 9 (except 8.5.4 and 9.2) of Cow Book

Slide ACKs: Jignesh Patel, Paris Koutris
Lifecycle of an SQL Query

```
SELECT R.text 
FROM Report R, Weather W 
WHERE W.rain() 
AND W.city = R.city 
AND W.date = R.date 
AND R.text.matches("insurance claims")
```
Lifecycle of an SQL Query

Query

Database Server

Parser

Syntax Tree

Parser

Query

Query Result
Lifecycle of an SQL Query

Database Server

Parser → Syntax Tree

Optimizer → Query Plan

Query Result
```
Select R.text from Report R, Weather W
where W.image.rain()
and W.city = R.city
and W.date = R.date
and R.text.
matches("insurance claims")
```
```
SELECT R.text FROM Report R, Weather W
WHERE W.image.rain() AND W.city = R.city AND W.date = R.date
AND R.text.
```

Lifecycle of an SQL Query
```
SELECT R.text FROM Report R, Weather W
WHERE W.image.rain()
AND W.city = R.city
AND W.date = R.date
AND R.text.
matches("insurance claims")
```
RDBMS Architecture
RDBMS Architecture

Storage Management Subsystem
Another View of Storage Manager

Access Methods
- Sorted File
- Heap File
- Hash Index
- B+-tree Index

Buffer Manager

I/O Manager

Concurrency Control Manager

Recovery Manager

I/O Accesses
Outline

❖ Data Storage (Disks)
❖ Buffer Management
❖ File Organization
❖ New Storage Hardware
Outline

❖ Data Storage (Disks)
❖ Buffer Management
❖ File Organization
❖ New Storage Hardware
Memory/Storage Hierarchy

- CPU
- Cache
- Main Memory
- Magnetic Hard Disk Drive (HDD)
Memory/Storage Hierarchy

- CPU
- Cache
- Main Memory
- Magnetic Hard Disk Drive (HDD)
Memory/Storage Hierarchy

CPU
Cache

Main Memory

Magnetic Hard Disk Drive (HDD)

Access Speed

10^7 - 10^8

100s

Access Cycles

ACC E S S

C Y C L E S
Memory/Storage Hierarchy

- **CPU**
  - Cache
  - Access Speed: ~100GB/s
  - Access Cycles: 100s

- **Main Memory**
  - Access Speed: ~10GB/s
  - Access Cycles: 100s

- **Magnetic Hard Disk Drive (HDD)**
  - Access Speed: ~200MB/s
  - Access Cycles: 10^7 - 10^8
Memory/Storage Hierarchy

CPU

Cache

Main Memory

Magnetic Hard Disk Drive (HDD)

Access Speed

~100GB/s

~10GB/s

~200MB/s

Capacity

10^7 - 10^8

100s

ACCESS CYCLES
Memory/Storage Hierarchy

- Access Speed:
  - ~100GB/s
  - ~10GB/s
  - ~200MB/s

- Capacity:
  - 10^7 - 10^8 cycles

- Price:
  - 100s

- Main Memory

- CPU Cache

- Magnetic Hard Disk Drive (HDD)
Memory/Storage Hierarchy

- **Flash Storage**
  - Access Speed: ~100GB/s
  - Access Cycles: 100s
  - Price: ~$2/MB
  - Capacity: ~10GBs

- **Main Memory**
  - Access Speed: ~10GB/s
  - Access Cycles: 10^5 - 10^6
  - Price: ~$5/GB
  - Capacity: ~1TBs

- **Magnetic Hard Disk Drive (HDD)**
  - Access Speed: ~200MB/s
  - Access Cycles: 10^7 - 10^8
  - Price: ~$200/TB
  - Capacity: ~10TBs
Memory/Storage Hierarchy

- **CPU Cache**
  - Access Speed: ~100GB/s
  - Capacity: ~1TB
  - Price: ~$5/GB

- **Main Memory**
  - Access Speed: ~10GB/s
  - Capacity: ~100GB
  - Price: ~$2/GB

- **Flash Storage**
  - Access Speed: ~200MB/s
  - Capacity: ~10TB
  - Price: ~$200/GB

- **Magnetic Hard Disk Drive (HDD)**
  - Access Speed: ~50MB/s
  - Capacity: ~PB
  - Price: ~$10/PB

- **Tape**
  - Access Speed: ~1GB/s
  - Capacity: ~PB
  - Price: ~$1/TB
Disks
Disks

- Widely used secondary storage device
- Data storage/retrieval units: **disk blocks** or **pages**
Disks

- Widely used secondary storage device
- Data storage/retrieval units: **disk blocks** or **pages**
- Unlike RAM, different disk pages have different retrieval times based on location!
- Need to optimize layout of data on disk pages
- Orders of magnitude performance gaps possible!
Components of a Disk

- Base Casting
- Spindle
- Slider (and Head)
- Actuator Arm
- Actuator Axis
- Actuator
- Cover Mounting Holes (Cover not shown)
- Case Mounting Holes
- Platters
- Ribbon Cable (attaches heads to Logic Board)
- SCSI Interface Connector
- Jumper Pins
- Power Connector
- Jumper
- Tape Seal
Components of a Disk

Anatomy of a regular hard disk
Components of a Disk

1 block = n contiguous sectors (n fixed during disk configuration)
How does a Disk Work?

❖ Magnetic changes on platters to store bits
❖ Spindle rotates platters
How does a Disk Work?

- Magnetic changes on platters to store bits
- Spindle rotates platters
- 7200 to 15000 RPM (Rotations Per Minute)

![Anatomy of a regular hard disk](image-url)
How does a Disk Work?

- Magnetic changes on platters to store bits
- Spindle rotates platters 7200 to 15000 RPM (Rotations Per Minute)
- Head reads/writes track
- Exactly 1 head can read/write at a time
- Arm moves radially to position head on track
How is the Disk Integrated?
How is the Disk Integrated?

OS interfaces with the Disk Controller
Disk Access Times

Access time = Rotational delay + Seek time + Transfer time
Disk Access Times

Access time = Rotational delay + Seek time + Transfer time

❖ Rotational delay
❖ Waiting for sector to come under disk head
❖ Function of RPM; typically, 0-10ms (avg v worst)
Disk Access Times

Access time = Rotational delay + Seek time + Transfer time

- Rotational delay
  - Waiting for sector to come under disk head
  - Function of RPM; typically, 0-10ms (avg v worst)
- Seek time
  - Moving disk head to correct track
  - Typically, 1-20ms (high-end disks: avg is 4ms)
Disk Access Times

Access time = Rotational delay + Seek time + Transfer time

❖ Rotational delay
❖ Waiting for sector to come under disk head
❖ Function of RPM; typically, 0-10ms (avg v worst)

❖ Seek time
❖ Moving disk head to correct track
❖ Typically, 1-20ms (high-end disks: avg is 4ms)

❖ Transfer time
❖ Moving data from/to disk surface
❖ Typically, hundreds of MB/s!
## Typical Modern Disk Spec

**Western Digital Blue WD10EZEX (from Amazon)**

<table>
<thead>
<tr>
<th>Specification</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Capacity</strong></td>
<td>1TB</td>
</tr>
<tr>
<td><strong>RPM</strong></td>
<td>7200</td>
</tr>
<tr>
<td><strong>Transfer</strong></td>
<td>6 Gb/s</td>
</tr>
<tr>
<td><strong>#Platters</strong></td>
<td>Just 1!</td>
</tr>
<tr>
<td><strong>Avg Seek</strong></td>
<td>9ms</td>
</tr>
<tr>
<td><strong>Price</strong></td>
<td>USD 50</td>
</tr>
</tbody>
</table>
Data Organization on Disk
Data Organization on Disk

- Disk space is organized into files; a relation is stored as a file!
Data Organization on Disk

- Disk space is organized into **files**; a relation is stored as a file!
- Files are made up of **pages**. Each page is physically laid out on 1 or more disk **blocks**.
  - Typical block size: 512 B; page size: 4KB or 8KB
  - OS/RAM page is not the same as disk block; page is basic unit of reads/writes to disk by OS
  - Page size is always a multiple of disk block size but can be much higher, e.g., even 1 MB
Disk space is organized into **files**; a relation is stored as a file!

Files are made up of **pages**. Each page is physically laid out on 1 or more disk **blocks**.

- Typical block size: 512 B; page size: 4KB or 8KB
- OS/RAM page is not the same as disk block; page is basic unit of reads/writes to disk by OS
- Page size is always a multiple of disk block size but can be much higher, e.g., even 1 MB
- File data (de-)allocated in increments of pages on disk
Data Organization on Disk

- Disk space is organized into **files**; a relation is stored as a file!
- Files are made up of **pages**. Each page is physically laid out on 1 or more disk **blocks**.
  - Typical block size: 512 B; page size: 4KB or 8KB
  - OS/RAM page is not the same as disk block; page is basic unit of reads/writes to disk by OS
  - Page size is always a multiple of disk block size but can be much higher, e.g., even 1 MB
- File data (de-)allocated in increments of pages on disk
- Pages contain **records** (tuples)
Disk Data Layout Principles
Disk Data Layout Principles

- **Sequential access v Random access**
  - Reading contiguous blocks together *amortizes* seek time and rotational delay!
  - For a transfer rate of 200MB/s, sequential reads can be ~200MB/s, but random reads ~0.3MB/s
  - Better to lay out pages of a file contiguously on disk
Disk Data Layout Principles

- **Sequential** access v **Random** access
- Reading contiguous blocks together *amortizes* seek time and rotational delay!
- For a transfer rate of 200MB/s, sequential reads can be ~200MB/s, but random reads ~0.3MB/s
- Better to lay out pages of a file contiguously on disk
- “Next” block concept:
  - On same track (in rotation order), then same cylinder, and then adjacent cylinder!
Review
Review

1. Why does hard disk have seek time?
Review

1. Why does hard disk have seek time?
2. If RPM of a given disk is 6000, what is the average rotational delay?
Review

1. Why does hard disk have seek time?
2. If RPM of a given disk is 6000, what is the average rotational delay?
3. What causes the random-sequential access dichotomy on hard disk?
Review

1. Why does hard disk have seek time?
2. If RPM of a given disk is 6000, what is the average rotational delay?
3. What causes the random-sequential access dichotomy on hard disk?
4. Which part(s) of the hard disk access latency does not change between random vs sequential access?
Is it possible to exploit RAM better and avoid going to disk all the time?
Outline

❖ Data Storage (Disks)
❖ Buffer Management
❖ File Organization
❖ New Storage Hardware
Outline

❖ Data Storage (Disks)
❖ Buffer Management
❖ File Organization
❖ New Storage Hardware
Buffer Management
Buffer Management

- Pages should be in RAM for DBMS query processing
- But not all pages of a database might fit in RAM!
Buffer Management

- Pages should be in RAM for DBMS query processing
- But not all pages of a database might fit in RAM!

Buffer Pool
- A part of main memory that DBMS manages
- Divided into buffer frames (slots for pages)
Buffer Management

❖ Pages should be in RAM for DBMS query processing
   ❖ But not all pages of a database might fit in RAM!

❖ Buffer Pool
   ❖ A part of main memory that DBMS manages
   ❖ Divided into buffer frames (slots for pages)

❖ Buffer Manager
   ❖ Subsystem of DBMS to read pages from disk to buffer pool and write “dirty” pages back to disk
Buffer Management

Page Requests from Higher Levels of DBMS

Buffer Pool

RAM

Disk

DB
Buffer Management

Page Requests from Higher Levels of DBMS

Buffer Pool

RAM

Disk

Page in an occupied frame
Buffer Management

Page Requests from Higher Levels of DBMS

Buffer Pool

RAM

Disk

Page in an occupied frame

Free frames
Buffer Management

**Page Requests** from Higher Levels of DBMS

Buffer Pool

RAM

Disk

Buffer Replacement Policy decides which frame to evict

Page in an occupied frame

Free frames
Page Requests to Buffer Manager

- **Request** a page for query processing (read or write)
- **Release** a page when no longer needed
- **Notify** if a page is modified (a write op happened)
Buffer Manager’s Bookkeeping
Buffer Manager’s Bookkeeping

- 2 variables per buffer frame maintained
Buffer Manager’s Bookkeeping

- 2 variables per buffer frame maintained
- Pin Count
  - Current number of “users” of the page in the frame
  - “Pinning” means PinCount++; page “requested”
  - “Unpinning” means PinCount is 0; page “released”
Buffer Manager’s Bookkeeping

- 2 variables per buffer frame maintained
- Pin Count
  - Current number of “users” of the page in the frame
  - “Pinning” means PinCount++; page “requested”
  - “Unpinning” means PinCount is 0; page “released”
- Dirty Bit
  - Set when a user “notifies” that page was modified
  - Must write this page back to disk in due course!
Buffer Manager’s Bookkeeping

- 2 variables per buffer frame maintained
- **Pin Count**
  - Current number of “users” of the page in the frame
  - “Pinning” means PinCount++; page “requested”
  - “Unpinning” means PinCount is 0; page “released”
- **Dirty Bit**
  - Set when a user “notifies” that page was modified
  - Must write this page back to disk in due course!

**Q:** What if 2 users pin and modify the same page?!
Handling Page Requests
Handling Page Requests

Is page in buffer pool?
Handling Page Requests

- Is page in buffer pool?
  - Yes
Handling Page Requests

Is page in buffer pool?

Yes

- Return address of the frame in the pool
Handling Page Requests

- Return address of the frame in the pool
- Increment Pin Count
Handling Page Requests

- Is page in buffer pool?
  - Yes
    - Return address of the frame in the pool
    - Increment Pin Count
  - No
Handling Page Requests

Is page in buffer pool?

- Yes
  - Return address of the frame in the pool
  - Increment Pin Count

- No
  - Choose a frame for replacement (buffer replacement policy); it should have Pin Count 0!
Handling Page Requests

- Return address of the frame in the pool
- Increment Pin Count

Is page in buffer pool?

- No
  - Choose a frame for replacement (buffer replacement policy); it should have Pin Count 0!
  - If chosen frame has Dirty Bit set, “flush” it to disk

- Yes
Handling Page Requests

- Is page in buffer pool?
  - Yes
    - Return address of the frame in the pool
    - Increment Pin Count
  - No
    - Choose a frame for replacement (buffer replacement policy); it should have Pin Count 0!
    - If chosen frame has Dirty Bit set, “flush” it to disk
    - Read requested page from disk into chosen frame
Handling Page Requests

Is page in buffer pool?

Yes

_return address of the frame in the pool
_increment Pin Count

No

Choose a frame for replacement (**buffer replacement policy**); it should have Pin Count 0!

If chosen frame has Dirty Bit set, “flush” it to disk

Read requested page from disk into chosen frame

Pin the page and return the frame address
Buffer Replacement Policy

- Policy to pick the frame for replacement
Buffer Replacement Policy

❖ Policy to pick the frame for replacement
❖ Has a major impact on I/O cost (number of disk I/Os) of a query based on its data access pattern
Buffer Replacement Policy

❖ Policy to pick the frame for replacement
❖ Has a major impact on I/O cost (number of disk I/Os) of a query based on its data access pattern
❖ Popular policies:
  ❖ Least Recently Used (LRU)
  ❖ Most Recently Used (MRU)
  ❖ “Clock” (LRU variant with lower overhead)
  ❖ First In First Out (FIFO), Random, etc.
Least Recently Used (LRU)

- Queue of pointers to frames with PinCount of 0
- Add newly unpinned frame to end of queue
- For replacement, grab frame from front of queue
Least Recently Used (LRU)

❖ Queue of pointers to frames with PinCount of 0
❖ Add newly unpinned frame to end of queue
❖ For replacement, grab frame from front of queue

Example:

3 frames in pool
5 pages on disk: A, B, C, D, E
Least Recently Used (LRU)

- Queue of pointers to frames with PinCount of 0
- Add newly unpinned frame to end of queue
- For replacement, grab frame from front of queue

**Example:**

3 frames in pool

5 pages on disk: A, B, C, D, E

*Page request sequence:*
Request A, Request B, Modify A, Request D,
Release B, Release A, Request E, Request C,
Release C, Release D, Release E
Least Recently Used (LRU)
## Least Recently Used (LRU)

<table>
<thead>
<tr>
<th>Page</th>
<th>Buf. Pool</th>
<th>PC</th>
<th>DB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
**Least Recently Used (LRU)**

<table>
<thead>
<tr>
<th>Page</th>
<th>PC</th>
<th>DB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Buf. Pool

Request A
Least Recently Used (LRU)

<table>
<thead>
<tr>
<th>Buf. Pool</th>
<th>Page</th>
<th>PC</th>
<th>DB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Request A</td>
<td>A</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
# Least Recently Used (LRU)

<table>
<thead>
<tr>
<th>Buf. Pool</th>
<th>Page</th>
<th>PC</th>
<th>DB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Request A</td>
<td>A</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Request B</td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Contains a table for managing page allocation in a buffer pool using LRU algorithm.
**Least Recently Used (LRU)**

<table>
<thead>
<tr>
<th>Buf. Pool</th>
<th>Page</th>
<th>PC</th>
<th>DB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Request A</td>
<td>A</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Request B</td>
<td>A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
## Least Recently Used (LRU)

<table>
<thead>
<tr>
<th>Buf. Pool</th>
<th>Page</th>
<th>PC</th>
<th>DB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Request A</td>
<td>A</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Request B</td>
<td>A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Modify A

### Request A
- A
- 1
- 0
- 0

### Request B
- A
- B
- 1
- 1
- 0
- 0
- 0
## Least Recently Used (LRU)

<table>
<thead>
<tr>
<th>Buf. Pool</th>
<th>Page</th>
<th>PC</th>
<th>DB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Request A</th>
<th>Request B</th>
<th>Modify A</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A - -</td>
<td>A B -</td>
<td>A B -</td>
</tr>
<tr>
<td></td>
<td>1 0 0</td>
<td>1 1 0</td>
<td>1 1 0</td>
</tr>
<tr>
<td></td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>1 0 0</td>
</tr>
</tbody>
</table>
### Least Recently Used (LRU)

<table>
<thead>
<tr>
<th>Buf. Pool</th>
<th>Page</th>
<th>PC</th>
<th>DB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Request A</td>
<td>A</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Request B</td>
<td>A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Request D</td>
<td>A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Modify A</td>
<td>A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
## Least Recently Used (LRU)

<table>
<thead>
<tr>
<th>Page</th>
<th>PC</th>
<th>DB</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### Buf. Pool

<table>
<thead>
<tr>
<th>Request A</th>
<th>Request D</th>
</tr>
</thead>
<tbody>
<tr>
<td>A - -</td>
<td>A B D</td>
</tr>
<tr>
<td>1 0 0</td>
<td>1 1 1</td>
</tr>
<tr>
<td>0 0 0</td>
<td>1 0 0</td>
</tr>
</tbody>
</table>

### Request B

<table>
<thead>
<tr>
<th>Modify A</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B -</td>
</tr>
<tr>
<td>1 1 0</td>
</tr>
<tr>
<td>0 0 0</td>
</tr>
<tr>
<td>1 0 0</td>
</tr>
</tbody>
</table>
# Least Recently Used (LRU)

<table>
<thead>
<tr>
<th>Page</th>
<th>PC</th>
<th>DB</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Request A</th>
<th>Request D</th>
</tr>
</thead>
<tbody>
<tr>
<td>A - -</td>
<td>A B D</td>
</tr>
<tr>
<td>1 0 0</td>
<td>1 1 1</td>
</tr>
<tr>
<td>0 0 0</td>
<td>1 0 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Request B</th>
<th>Modify A</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B -</td>
<td>A B -</td>
</tr>
<tr>
<td>1 1 0</td>
<td>1 1 0</td>
</tr>
<tr>
<td>0 0 0</td>
<td>1 0 0</td>
</tr>
</tbody>
</table>
## Least Recently Used (LRU)

<table>
<thead>
<tr>
<th>Buf. Pool</th>
<th>Release B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page</td>
<td>PC</td>
</tr>
<tr>
<td>- - -</td>
<td>0 0 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Request A</th>
<th>Request D</th>
</tr>
</thead>
<tbody>
<tr>
<td>A - -</td>
<td>A B D</td>
</tr>
<tr>
<td>1 0 0</td>
<td>1 0 1</td>
</tr>
<tr>
<td>0 0 0</td>
<td>1 0 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Request B</th>
<th>Modify A</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B -</td>
<td>A B -</td>
</tr>
<tr>
<td>1 1 0</td>
<td>1 1 0</td>
</tr>
<tr>
<td>0 0 0</td>
<td>1 0 0</td>
</tr>
</tbody>
</table>
## Least Recently Used (LRU)

<table>
<thead>
<tr>
<th>Page</th>
<th>PC</th>
<th>DB</th>
<th>Request A</th>
<th>Request B</th>
<th>Request D</th>
<th>Modify A</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>0</td>
<td>0</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1 0 0</td>
<td>0 0 0</td>
<td>1 1 1</td>
<td>1 1 0</td>
</tr>
</tbody>
</table>

**Queue of pointers:** F1
### Least Recently Used (LRU)

<table>
<thead>
<tr>
<th>Page</th>
<th>PC</th>
<th>DB</th>
<th>Buf. Pool</th>
<th>Release B</th>
<th>Release A</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>- - -</td>
<td>A B D</td>
<td>1 0 1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0 0 0</td>
<td>1 0 0</td>
<td>1 0 0</td>
</tr>
</tbody>
</table>

- **Request A**
  - A - -
  - 1 0 0
  - 0 0 0

- **Request B**
  - A B -
  - 1 1 0
  - 0 0 0

- **Request D**
  - A B D
  - 1 1 1
  - 1 0 0

- **Modify A**
  - A B -
  - 1 1 0
  - 1 0 0

Queue of pointers: F1
## Least Recently Used (LRU)

<table>
<thead>
<tr>
<th>Page</th>
<th>PC</th>
<th>DB</th>
<th>Buf. Pool</th>
<th>Release B</th>
<th>Release A</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 0 0 0</td>
<td>1 0 0 0</td>
<td>1 0 0 0</td>
<td>0 0 1</td>
<td>1 0 0 0</td>
</tr>
<tr>
<td>Request A</td>
<td>A - -</td>
<td>A B D</td>
<td>A B D</td>
<td>A B D</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>1 0 0</td>
<td>A</td>
<td>1 0 0</td>
<td>1</td>
<td>0 0 0</td>
</tr>
<tr>
<td>Request B</td>
<td>A B -</td>
<td>A B D</td>
<td>A B D</td>
<td>A B D</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>1 1 0</td>
<td>A</td>
<td>1 1 0</td>
<td>1</td>
<td>0 0 0</td>
</tr>
<tr>
<td>Modify A</td>
<td>A B -</td>
<td>A B D</td>
<td>A B D</td>
<td>A B D</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>1 1 0</td>
<td>A</td>
<td>1 1 0</td>
<td>1</td>
<td>0 0 0</td>
</tr>
</tbody>
</table>

Queue of pointers: F1
Least Recently Used (LRU)

<table>
<thead>
<tr>
<th>Page</th>
<th>Buf. Pool</th>
<th>Release B</th>
<th>Release A</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC</td>
<td>0 0 0 0</td>
<td>1 0 1</td>
<td>0 0 1</td>
</tr>
<tr>
<td>DB</td>
<td>0 0 0 0</td>
<td>1 0 0</td>
<td>1 0 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Request A</th>
<th>Request D</th>
</tr>
</thead>
<tbody>
<tr>
<td>A - - -</td>
<td>A B D</td>
</tr>
<tr>
<td>1 0 0</td>
<td>1 1 1</td>
</tr>
<tr>
<td>0 0 0</td>
<td>1 0 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Request B</th>
<th>Modify A</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B -</td>
<td>A B -</td>
</tr>
<tr>
<td>1 1 0</td>
<td>1 1 0</td>
</tr>
<tr>
<td>0 0 0</td>
<td>1 0 0</td>
</tr>
</tbody>
</table>

Queue of pointers: F1 F0
Least Recently Used (LRU)

<table>
<thead>
<tr>
<th>Page</th>
<th>PCB</th>
<th>DB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Buf. Pool**

- Request A
  - A
  - 1 0 0
  - 0 0 0

- Request B
  - A
  - 1 1 0
  - 0 0 0

- Request D
  - A
  - 1 1 1
  - 1 0 0

- Modify A
  - A
  - 1 1 0
  - 1 0 0

**Release B**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Release A**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Queue of pointers:** F1 F0
# Least Recently Used (LRU)

<table>
<thead>
<tr>
<th>Buffer Pool</th>
<th>Release B</th>
<th>Release A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page</td>
<td>Request A</td>
<td>Request B</td>
</tr>
<tr>
<td>PC</td>
<td>A B D</td>
<td>A B D</td>
</tr>
<tr>
<td>DB</td>
<td>1 0 1</td>
<td>1 0 0</td>
</tr>
</tbody>
</table>

Queue of pointers: F1 F0
## Least Recently Used (LRU)

<table>
<thead>
<tr>
<th>Buf. Pool</th>
<th>Release B</th>
<th>Release A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page</td>
<td>PC</td>
<td>DB</td>
</tr>
<tr>
<td>- - -</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>1 0 1</td>
<td>0 0 1</td>
</tr>
<tr>
<td>0 0 0</td>
<td>1 0 0</td>
<td>1 0 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Request A</th>
<th>Request D</th>
<th>Request E</th>
</tr>
</thead>
<tbody>
<tr>
<td>A - -</td>
<td>A B D</td>
<td>A B D</td>
</tr>
<tr>
<td>1 0 0</td>
<td>1 1 1</td>
<td>1 0 0</td>
</tr>
<tr>
<td>0 0 0</td>
<td>1 0 0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Request B</th>
<th>Modify A</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B -</td>
<td>A B -</td>
</tr>
<tr>
<td>1 1 0</td>
<td>1 1 0</td>
</tr>
<tr>
<td>0 0 0</td>
<td>1 0 0</td>
</tr>
</tbody>
</table>

Queue of pointers: F0 F1
Least Recently Used (LRU)

<table>
<thead>
<tr>
<th>Buf. Pool</th>
<th>Release B</th>
<th>Release A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page</td>
<td>Release B</td>
<td>Release A</td>
</tr>
<tr>
<td>PC</td>
<td>A B D</td>
<td>A B D</td>
</tr>
<tr>
<td>DB</td>
<td>0 0 0</td>
<td>0 0 1</td>
</tr>
<tr>
<td></td>
<td>1 0 0</td>
<td>1 0 0</td>
</tr>
<tr>
<td></td>
<td>1 0 0</td>
<td>1 0 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Request A</th>
<th>Request D</th>
<th>Request E</th>
</tr>
</thead>
<tbody>
<tr>
<td>A - - -</td>
<td>A B D</td>
<td>A E D</td>
</tr>
<tr>
<td>1 0 0</td>
<td>1 1 1</td>
<td>0 1 1</td>
</tr>
<tr>
<td>0 0 0</td>
<td>1 0 0</td>
<td>1 0 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Request B</th>
<th>Modify A</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B -</td>
<td>A B -</td>
</tr>
<tr>
<td>1 1 0</td>
<td>1 1 0</td>
</tr>
<tr>
<td>0 0 0</td>
<td>1 0 0</td>
</tr>
</tbody>
</table>

Queue of pointers: F1 F0
# Least Recently Used (LRU)

<table>
<thead>
<tr>
<th>Page</th>
<th>PC</th>
<th>DB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Request A</th>
<th>Request B</th>
<th>Modify A</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1 0 0</td>
<td>1 1 0</td>
<td>1 1 0</td>
</tr>
<tr>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Request D</th>
<th>Release B</th>
<th>Release A</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>1 0 1</td>
<td>1 0 0</td>
<td>0 0 1</td>
</tr>
<tr>
<td>1 0 0</td>
<td>1 0 0</td>
<td>0 0 0</td>
</tr>
</tbody>
</table>

Queue of pointers: F0
### Least Recently Used (LRU)

<table>
<thead>
<tr>
<th>Page</th>
<th>PC</th>
<th>DB</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Request A</th>
<th>A</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request B</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Request D</th>
<th>A</th>
<th>B</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modify A</td>
<td>A</td>
<td>B</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Request E</th>
<th>A</th>
<th>E</th>
<th>D</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Release B</th>
<th>A</th>
<th>B</th>
<th>D</th>
</tr>
</thead>
</table>

| Release A |  A  |  B  |  D  |

Queue of pointers: F0
## Least Recently Used (LRU)

<table>
<thead>
<tr>
<th>Page</th>
<th>PC</th>
<th>DB</th>
<th>Buf. Pool</th>
<th>Release B</th>
<th>Release A</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>- - -</td>
<td>A B D</td>
<td>A B D</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 0 0</td>
<td>1 0 1</td>
<td>0 0 1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 0 0</td>
<td>1 0 0</td>
<td>1 0 0</td>
</tr>
</tbody>
</table>

**Request A**

<table>
<thead>
<tr>
<th>A</th>
<th>-</th>
<th>-</th>
<th>1 0 0</th>
<th>1 0 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 0 0</td>
<td>0 0 0</td>
</tr>
</tbody>
</table>

**Request B**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>-</th>
<th>1 1 0</th>
<th>1 0 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 0 0</td>
<td>0 0 0</td>
</tr>
</tbody>
</table>

**Request D**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>D</th>
<th>1 1 1</th>
<th>1 0 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1 0 0</td>
<td>1 0 0</td>
</tr>
</tbody>
</table>

**Request E**

<table>
<thead>
<tr>
<th>A</th>
<th>E</th>
<th>D</th>
<th>0 1 1</th>
<th>1 0 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1 0 0</td>
<td>1 0 0</td>
</tr>
</tbody>
</table>

**Modify A**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>-</th>
<th>1 1 0</th>
<th>F0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1 0 0</td>
<td>F0</td>
</tr>
</tbody>
</table>

**Queue of pointers:**

F0
### Least Recently Used (LRU)

<table>
<thead>
<tr>
<th>Buf. Pool</th>
<th>Release B</th>
<th>Release A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page</td>
<td>PC</td>
<td>DB</td>
</tr>
<tr>
<td>-</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

#### Request A

- **A**
- 1 0 0
- 0 0 0

#### Request B

- **A** **B**
- 1 1 0
- 0 0 0

#### Request D

- **A** **B** **D**
- 1 1 1
- 1 0 0

#### Request E

- **A** **E** **D**
- 0 1 1
- 1 0 0

#### Modify A

- **A** **B**
- 1 1 0
- 1 0 0

#### Request C

Flush A!

#### Queue of pointers:

F0
### Least Recently Used (LRU)

<table>
<thead>
<tr>
<th>Page</th>
<th>Buf. Pool</th>
<th>Release B</th>
<th>Release A</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC</td>
<td>0 0 0 0</td>
<td>A B D</td>
<td>A B D</td>
</tr>
<tr>
<td>DB</td>
<td>0 0 0 0</td>
<td>1 0 1</td>
<td>0 0 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Request A</th>
<th>Request D</th>
<th>Request E</th>
</tr>
</thead>
<tbody>
<tr>
<td>A - - -</td>
<td>A B D</td>
<td>A E D</td>
</tr>
<tr>
<td>1 0 0</td>
<td>1 1 1</td>
<td>0 1 1</td>
</tr>
<tr>
<td>0 0 0</td>
<td>1 0 0</td>
<td>1 0 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Request B</th>
<th>Modify A</th>
<th>Request C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B -</td>
<td>A B -</td>
<td>C E D</td>
</tr>
<tr>
<td>1 1 0</td>
<td>1 1 0</td>
<td>1 1 1</td>
</tr>
<tr>
<td>0 0 0</td>
<td>1 0 0</td>
<td>0 0 0</td>
</tr>
</tbody>
</table>

Flush A!

Queue of pointers: F0
### Least Recently Used (LRU)

<table>
<thead>
<tr>
<th>Queue of pointers:</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Page</th>
<th>PC</th>
<th>DB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 0 0 0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Request A</th>
<th>Request B</th>
<th>Request C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A - - -</td>
<td>A B -</td>
<td>C E D</td>
</tr>
<tr>
<td>1 0 0</td>
<td>1 1 0</td>
<td>1 1 1</td>
</tr>
<tr>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Request D</th>
<th>Modify A</th>
<th>Request E</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B D</td>
<td>A B -</td>
<td>A E D</td>
</tr>
<tr>
<td>1 1 1</td>
<td>1 1 0</td>
<td>0 1 1</td>
</tr>
<tr>
<td>1 0 0</td>
<td>1 0 0</td>
<td>1 0 0</td>
</tr>
</tbody>
</table>

- **Request A**: 
  - PC: 0 0 0 0
  - DB: 0 0 0 0
  - Page: A - - -
  - Modify A

- **Request B**: 
  - PC: 0 0 0 0
  - DB: 0 0 0 0
  - Page: A B -

- **Request D**: 
  - PC: 0 0 0 0
  - DB: 0 0 0 0
  - Page: A B D

- **Request E**: 
  - PC: 0 0 0 0
  - DB: 0 0 0 0
  - Page: A E D

- **Flush A!**

- **Flush B**: 
  - PC: 0 0 0 0
  - DB: 0 0 0 0
  - Page: A B D

- **Flush C**: 
  - PC: 0 0 0 0
  - DB: 0 0 0 0
  - Page: C E D

- **Flush D**: 
  - PC: 0 0 0 0
  - DB: 0 0 0 0
  - Page: A B D
# Least Recently Used (LRU)

<table>
<thead>
<tr>
<th>Page</th>
<th>PC</th>
<th>DB</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Buf. Pool**

<table>
<thead>
<tr>
<th>Request A</th>
<th>Request B</th>
<th>Request C</th>
<th>Request D</th>
<th>Request E</th>
<th>Release A</th>
<th>Release B</th>
<th>Release C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
<td>C</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>1 0 0</td>
<td>1 0 0</td>
<td>0 0 0</td>
<td>1 1 1</td>
<td>0 1 1</td>
<td>0 0 0</td>
<td>0 0 1</td>
<td></td>
</tr>
<tr>
<td>0 0 0</td>
<td>1 0 0</td>
<td>1 0 0</td>
<td>1 0 0</td>
<td>1 0 0</td>
<td>1 0 0</td>
<td>1 0 0</td>
<td></td>
</tr>
</tbody>
</table>

**Queue of pointers:**

Flush A!
## Least Recently Used (LRU)

<table>
<thead>
<tr>
<th>Buf. Pool</th>
<th>Release B</th>
<th>Release A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page</td>
<td>A B D</td>
<td>A B D</td>
</tr>
<tr>
<td>PC</td>
<td>1 0 1</td>
<td>0 0 1</td>
</tr>
<tr>
<td>DB</td>
<td>1 0 0</td>
<td>1 0 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Request A</th>
<th>Request D</th>
<th>Request E</th>
<th>Release C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A - -</td>
<td>A B D</td>
<td>A E D</td>
<td>C E D</td>
</tr>
<tr>
<td>1 0 0</td>
<td>1 1 1</td>
<td>0 1 1</td>
<td>0 1 1</td>
</tr>
<tr>
<td>0 0 0</td>
<td>1 0 0</td>
<td>1 0 0</td>
<td>0 0 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Request B</th>
<th>Modify A</th>
<th>Request C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B -</td>
<td>A B -</td>
<td>C E D</td>
</tr>
<tr>
<td>1 1 0</td>
<td>1 1 0</td>
<td>1 1 1</td>
</tr>
<tr>
<td>0 0 0</td>
<td>1 0 0</td>
<td>0 0 0</td>
</tr>
</tbody>
</table>

Queue of pointers:

Flush A!
## Least Recently Used (LRU)

<table>
<thead>
<tr>
<th>Page</th>
<th>PC</th>
<th>DB</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Request A</th>
<th>Request D</th>
<th>Release B</th>
<th>Release A</th>
<th>Request E</th>
<th>Release C</th>
<th>Request C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
<td>A B D</td>
<td>A B D</td>
<td>A E D</td>
<td>C E D</td>
<td>C E D</td>
</tr>
<tr>
<td>A - - -</td>
<td>A B D</td>
<td>1 0 1</td>
<td>0 0 1</td>
<td>0 1 1</td>
<td>0 1 1</td>
<td>0 1 1</td>
</tr>
<tr>
<td>1 0 0</td>
<td>1 1 1</td>
<td>1 0 0</td>
<td>1 0 0</td>
<td>1 1 1</td>
<td>0 0 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>0 0 0</td>
<td>1 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Modify A</th>
<th>Request C</th>
<th>Queue of pointers:</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B -</td>
<td>C E D</td>
<td></td>
</tr>
<tr>
<td>1 1 0</td>
<td>1 1 1</td>
<td></td>
</tr>
<tr>
<td>0 0 0</td>
<td>0 0 0</td>
<td></td>
</tr>
</tbody>
</table>

Flush A!
Clock Algorithm
Clock Algorithm

❖ Variant of LRU with lower overhead (no queue)
Clock Algorithm

- Variant of LRU with lower overhead (no queue)
- N buffer frames treated *logically* as a circle: the “clock”
Clock Algorithm

- Variant of LRU with lower overhead (no queue)
- N buffer frames treated *logically* as a circle: the “clock”
- **Current** variable points to a frame: the “hand”
Clock Algorithm

❖ Variant of LRU with lower overhead (no queue)
❖ N buffer frames treated *logically* as a circle: the “clock”
❖ **Current** variable points to a frame: the “hand”
❖ Each frame has **Referenced Bit** (along with PC, DB)
Clock Algorithm

- Variant of LRU with lower overhead (no queue)
- N buffer frames treated *logically* as a circle: the “clock”
- **Current** variable points to a frame: the “hand”
- Each frame has **Referenced Bit** (along with PC, DB)
- Finding a frame to replace:
Clock Algorithm

❖ Variant of LRU with lower overhead (no queue)
❖ N buffer frames treated *logically* as a circle: the “clock”
❖ **Current** variable points to a frame: the “hand”
❖ Each frame has **Referenced Bit** (along with PC, DB)
❖ Finding a frame to replace:
   ❖ If PC > 0, increment Current
Clock Algorithm

❖ Variant of LRU with lower overhead (no queue)
❖ N buffer frames treated \textit{logically} as a circle: the “clock”
❖ \textbf{Current} variable points to a frame: the “hand”
❖ Each frame has \textbf{Referenced Bit} (along with PC, DB)
❖ Finding a frame to replace:
  ❖ If PC > 0, increment Current
  ❖ If PC == 0:
Clock Algorithm

- Variant of LRU with lower overhead (no queue)
- N buffer frames treated *logically* as a circle: the “clock”
- **Current** variable points to a frame: the “hand”
- Each frame has **Referenced Bit** (along with PC, DB)
- Finding a frame to replace:
  - If PC > 0, increment Current
  - If PC == 0:
    - If RB == 1, set its RB = 0
    - and increment Current
Clock Algorithm

- Variant of LRU with lower overhead (no queue)
- N buffer frames treated \textit{logically} as a circle: the “clock”
- \textbf{Current} variable points to a frame: the “hand”
- Each frame has \textbf{Referenced Bit} (along with PC, DB)

Finding a frame to replace:
- If PC > 0, increment Current
- If PC == 0:
  - If RB == 1, set its RB = 0 and increment Current
  - If RB == 0, pick this frame!
Clock Algorithm

- Variant of LRU with lower overhead (no queue)
- N buffer frames treated *logically* as a circle: the “clock”
- **Current** variable points to a frame: the “hand”
- Each frame has **Referenced Bit** (along with PC, DB)

Finding a frame to replace:
- If PC > 0, increment Current
- If PC == 0:
  - If RB == 1, set its RB = 0 and increment Current
  - If RB == 0, pick this frame!
Sequential Flooding
Sequential Flooding

- LRU performs poorly when file is repeatedly scanned
  - Given: Num. buffer frames < Num. pages in file
  - Then, every page request causes a disk I/O!
Sequential Flooding

- LRU performs poorly when file is repeatedly scanned
  - Given: Num. buffer frames < Num. pages in file
  - Then, every page request causes a disk I/O!

**Q:** *Which other replacement policy is better for this case? LRU, Clock, MRU, FIFO, or Random?*
DBMS vs OS Filesystem
Q: DBMS sits on top of OS filesystem; so, why not just let OS handle database file layout and buffer management?
**DBMS vs OS Filesystem**

**Q:** DBMS sits on top of OS filesystem; so, why not just let OS handle database file layout and buffer management?

- DBMS knows fine-grained information of data access patterns of this “application” compared to OS!
- Can pre-fetch pages as per query semantics
- Can better interleave I/Os and computations
- Can exploit multiple disks more effectively (RAID)
Q: DBMS sits on top of OS filesystem; so, why not just let OS handle database file layout and buffer management?

❖ DBMS knows fine-grained information of data access patterns of this “application” compared to OS!
❖ Can pre-fetch pages as per query semantics
❖ Can better interleave I/Os and computations
❖ Can exploit multiple disks more effectively (RAID)
❖ Own buffer pool lets DBMS adjust buffer replacement policy, pin pages to memory, and flush dirty pages
Outline

❖ Data Storage (Disks)
❖ Buffer Management
❖ File Organization
❖ New Storage Hardware
Outline

- Data Storage (Disks)
- Buffer Management
- File Organization
- New Storage Hardware
Data Organization Basics: Recap
Data Organization Basics: Recap

- Disk space is organized into **files** (a relation is a file!)
Data Organization Basics: Recap

- Disk space is organized into **files** (a relation is a file!)
- Files are made up of **pages**
- File data (de-)allocated in increments of disk pages
Data Organization Basics: Recap

- Disk space is organized into **files** (a relation is a file!)
- Files are made up of **pages**
  - File data (de-)allocated in increments of disk pages
- Pages contain **records** (tuples)
  - Higher levels operate on (sets of) records!
Data Organization Basics: Recap

- Disk space is organized into **files** (a relation is a file!)
- Files are made up of **pages**
  - File data (de-)allocated in increments of disk pages
- Pages contain **records** (tuples)
  - Higher levels operate on (sets of) records!

How pages are organized in a file: **Page Layout**
Data Organization Basics: Recap

❖ Disk space is organized into **files** (a relation is a file!)
❖ Files are made up of **pages**
  ❖ File data (de-)allocated in increments of disk pages
❖ Pages contain **records** (tuples)
  ❖ Higher levels operate on (sets of) records!

How pages are organized in a file: **Page Layout**
How records are organized in a page: **Record Layout**
Data Organization Basics: Recap

- Disk space is organized into files (a relation is a file!)
- Files are made up of pages
  - File data (de-)allocated in increments of disk pages
- Pages contain records (tuples)
  - Higher levels operate on (sets of) records!

How pages are organized in a file: Page Layout
How records are organized in a page: Record Layout
Unordered (Heap) Files
Unordered (Heap) Files

- Simplest structure; records/pages in no particular order
Unordered (Heap) Files

- Simplest structure; records/pages in no particular order
- Pages added/deleted table grows/shrinks
Unordered (Heap) Files

❖ Simplest structure; records/pages in no particular order
❖ Pages added/deleted table grows/shrinks
❖ Metadata tracked to enable record-level access:
  ❖ Pages in the file (PageID)
  ❖ Records in a page (RecordID)
  ❖ Free space in a page
Unordered (Heap) Files

- Simplest structure; records/pages in no particular order
- Pages added/deleted table grows/shrinks
- Metadata tracked to enable record-level access:
  - Pages in the file (PageID)
  - Records in a page (RecordID)
  - Free space in a page
- Operations on the file: insert/delete file, read a record with a given RID, scan records (maybe with predicate), add/delete record(s)
Heap File as Linked Lists
Heap File as Linked Lists

Header Page

Full Pages

Pages with Free Space
Heap File as Linked Lists

- (Filename, Header PageID) stored in known catalog
Heap File as Linked Lists

- (Filename, Header PageID) stored in known catalog
- Each page has 2 pointers (PageIDs) and data records
Heap File as Linked Lists

- (Filename, Header PageID) stored in known catalog
- Each page has 2 pointers (PageIDs) and data records
- Pages in second list have some free space
Heap File as Linked Lists

- (Filename, Header PageID) stored in known catalog
- Each page has 2 pointers (PageIDs) and data records
- Pages in second list have some free space

Q: Why would free space arise in pages?
Heap File as Page Directory
Heap File as Page Directory

Header Page

Directory

Data Page 1

Data Page 2

Data Page N
Entry in directory for each page:

- Is it free or full?
- How many bytes of free space?
Entry in directory for each page:
- Is it free or full?
- How many bytes of free space?
- Faster to identify page with free space to add records
Data Organization Basics: Recap
Data Organization Basics: Recap

- Disk space is organized into files (a relation is a file!)
Data Organization Basics: Recap

❖ Disk space is organized into files (a relation is a file!)
❖ Files are made up of pages
❖ File data (de-)allocated in increments of disk pages
Data Organization Basics: Recap

- Disk space is organized into files (a relation is a file!)
- Files are made up of pages
  - File data (de-)allocated in increments of disk pages
- Pages contain records (tuples)
  - Higher levels operate on (sets of) records!
Data Organization Basics: Recap

- Disk space is organized into files (a relation is a file!)
- Files are made up of pages
  - File data (de-)allocated in increments of disk pages
- Pages contain records (tuples)
  - Higher levels operate on (sets of) records!

How pages are organized in a file: Page Layout
Data Organization Basics: Recap

- Disk space is organized into **files** (a relation is a file!)
- Files are made up of **pages**
  - File data (de-)allocated in increments of disk pages
- Pages contain **records** (tuples)
  - Higher levels operate on (sets of) records!

How pages are organized in a file: **Page Layout**
How records are organized in a page: **Record Layout**
Data Organization Basics: Recap

- Disk space is organized into **files** (a relation is a file!)
- Files are made up of **pages**
  - File data (de-)allocated in increments of disk pages
- Pages contain **records** (tuples)
  - Higher levels operate on (sets of) records!

How pages are organized in a file: **Page Layout**
How records are organized in a page: **Record Layout**
Record Layout Desiderata
Higher levels (queries) operate on sets of records
Record Layout Desiderata

❖ Higher levels (queries) operate on sets of records
❖ Records are stored in slotted pages
❖ Page is a collection of slots; one record per slot
Higher levels (queries) operate on sets of records

Records are stored in slotted pages

Page is a collection of slots; one record per slot

Physically, RecordID = <PageID, SlotNumber>!
Record Layout Desiderata

- Higher levels (queries) operate on sets of records
- Records are stored in slotted pages
  - Page is a collection of slots; one record per slot
- Physically, RecordID = <PageID, SlotNumber>
- Many record layouts possible
Higher levels (queries) operate on sets of records

Records are stored in **slotted pages**

- Page is a collection of slots; one record per slot
- Physically, **RecordID = <PageID, SlotNumber>**!

Many record layouts possible

Need to support record-level operations efficiently

- Insert a record or multiple records
- Read/update/delete a record given its RecordID
- Scan all records (possibly applying a predicate)
Record Layout and Format Outline

❖ Layout of fixed-length records:
  ❖ Packed layout
  ❖ Unpacked layout
❖ Layout of variable-length records
❖ Record format for fixed-length records
❖ Record formats for variable-length records
  ❖ Delimiter-based
  ❖ Pointer-based
Layout of Fixed-length Records
Layout of Fixed-length Records

Slot 1
Slot 2
Slot N

Packed

Free Space

Number of records

N
Layout of Fixed-length Records

Packed

Unpacked

Number of records

Free Space

Number of slots

Bitmap
Recall that \textbf{RecordID} = \langle \text{PageID}, \text{SlotNumber} \rangle

- Con for Packed: moving/deleting records alter RecIDs
- Con for Unpacked: extra space used by bitmap
Layout of Variable-length Records

Start
Layout of Variable-length Records
Layout of Variable-length Records

Start
Layout of Variable-length Records

Start

Book-keeping
Layout of Variable-length Records

Start

Free Space

Pointer

Book-keeping
Layout of Variable-length Records

Start

Free Space Pointer

Slot directory

Book-keeping

[Diagram showing the layout of variable-length records with slots, pointers, and free space]
Layout of Variable-length Records

Start

Free Space Pointer

Slot directory

Slot entry: offset, length

Book-keeping

70, 50
6

5 4 3 2 1 0
Layout of Variable-length Records

Start

Free Space Pointer

Slot directory

Slot num

Slot entry: offset, length

Book-keeping

70, 50

6

...
Layout of Variable-length Records

Start

Slot directory

Slot entry: offset, length

Free Space Pointer

Pointer

Slot num

Book-keeping

120, 40
-1, 0
560, 90
-1, 0
0, 70
70, 50
6

0, 70
0
1
2
3
4
5
Layout of Variable-length Records

Free Space Pointer

Slot directory

Slot num

5 4 3 2 1 0

Dir. grows backwards!

Slot entry: offset, length

Book-keeping

Slot num

Slot directory

Start

120, -1, 0 560, -1, 0 0, 0 70, 70, 50 6

Book-keeping
Layout of Variable-length Records

Start

Page num = 11

Rid = ?

Slot directory

Free Space Pointer

Slot num

Slot entry: offset, length

Dir. grows backwards!

Book-keeping

120, 40
-1, 0
560, 90
-1, 0
0, 70
70, 50
6

1 0

2 1

3 2

4 3

5

Book-keeping
Layout of Variable-length Records

- **Slot directory**
- **Rid=?(11, 1)**
- **Page num = 11**
- **Dir. grows backwards!**
- **Slot entry: offset, length**
- **Start**
- **Free Space Pointer**
- **Slot num**
Pros: moving records on page does not alter RID!
- Good for fixed-length records too
- Deleting a record: offset is set to -1
- Inserting a new record:
  - Any available slot can be used (incl. in free space)
  - If not enough free space, reorganize
Fixed-length Record Format

- All records in a file are same “type” and length
- System catalog contains attribute data type lengths

![Diagram showing fixed-length record format with fields F1, F2, F3, F4, and base address (B). Address = B + L1 + L2]
Variable-length Record Formats
Variable-length Record Formats

Field Count

4 | F1 | $ | F2 | $ | F3 | $ | F4 | $
Variable-length Record Formats

Field Count

Array of Integer Offsets

Delimiter symbol

F1 $ F2 $ F3 $ F4 $
Variable-length Record Formats

- Both store fields consecutively; count fixed by schema!
- Con of delimiter-based: need to scan record from start to retrieve even a single field (attribute)
- Cons of pointer-based: small dir. overhead; growing records require maintaining dir.; records larger than a page!
Column Store Layout
Consider the following SQL query:

```
SELECT COUNT(DISTINCT Year) FROM Movies M
```
Consider the following SQL query:

```
SELECT COUNT(DISTINCT Year) FROM Movies M
```

Q: *Why bother reading other attributes?*
Consider the following SQL query:

```sql
SELECT COUNT(DISTINCT Year) FROM Movies M
```

**Q:** Why bother reading other attributes?

Often, “analytical” queries read only one or a few attributes; reading other attributes wastes I/O time!
Consider the following SQL query:

```
SELECT COUNT(DISTINCT Year) FROM Movies M
```

Q: Why bother reading other attributes?

- Often, “analytical” queries read only one or a few attributes; reading other attributes wastes I/O time!
- “Column store” DBMSs lay out relations in column-major order to help speed up such queries
<table>
<thead>
<tr>
<th>MovieID</th>
<th>Name</th>
<th>Year</th>
<th>Director</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>Inception</td>
<td>2010</td>
<td>Christopher Nolan</td>
</tr>
<tr>
<td>16</td>
<td>Avatar</td>
<td>2009</td>
<td>Jim Cameron</td>
</tr>
<tr>
<td>53</td>
<td>Gravity</td>
<td>2013</td>
<td>Alfonso Cuaron</td>
</tr>
<tr>
<td>74</td>
<td>Blue Jasmine</td>
<td>2013</td>
<td>Woody Allen</td>
</tr>
</tbody>
</table>
## Column Store Layout

<table>
<thead>
<tr>
<th>MovieID</th>
<th>Name</th>
<th>Year</th>
<th>Director</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>Inception</td>
<td>2010</td>
<td>Christopher Nolan</td>
</tr>
<tr>
<td>16</td>
<td>Avatar</td>
<td>2009</td>
<td>Jim Cameron</td>
</tr>
<tr>
<td>53</td>
<td>Gravity</td>
<td>2013</td>
<td>Alfonso Cuaron</td>
</tr>
<tr>
<td>74</td>
<td>Blue Jasmine</td>
<td>2013</td>
<td>Woody Allen</td>
</tr>
</tbody>
</table>

Pages in a column store layout:
## Column Store Layout

<table>
<thead>
<tr>
<th>MovieID</th>
<th>Name</th>
<th>Year</th>
<th>Director</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>Inception</td>
<td>2010</td>
<td>Christopher Nolan</td>
</tr>
<tr>
<td>16</td>
<td>Avatar</td>
<td>2009</td>
<td>Jim Cameron</td>
</tr>
<tr>
<td>53</td>
<td>Gravity</td>
<td>2013</td>
<td>Alfonso Cuaron</td>
</tr>
<tr>
<td>74</td>
<td>Blue Jasmine</td>
<td>2013</td>
<td>Woody Allen</td>
</tr>
</tbody>
</table>

Pages in a column store layout:

- 20, 16, 53, 74
- Inception, Avatar
- Gravity, Blue Jasmine

...
## Column Store Layout

<table>
<thead>
<tr>
<th>MovieID</th>
<th>Name</th>
<th>Year</th>
<th>Director</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>Inception</td>
<td>2010</td>
<td>Christopher Nolan</td>
</tr>
<tr>
<td>16</td>
<td>Avatar</td>
<td>2009</td>
<td>Jim Cameron</td>
</tr>
<tr>
<td>53</td>
<td>Gravity</td>
<td>2013</td>
<td>Alfonso Cuaron</td>
</tr>
<tr>
<td>74</td>
<td>Blue Jasmine</td>
<td>2013</td>
<td>Woody Allen</td>
</tr>
</tbody>
</table>

Pages in a column store layout:

- 20, 16, 53, 74
- Inception, Avatar
- Gravity, Blue Jasmine
- 2010, 2009, 2013, 2013...

Column Store Layout

<table>
<thead>
<tr>
<th>MovieID</th>
<th>Name</th>
<th>Year</th>
<th>Director</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>Inception</td>
<td>2010</td>
<td>Christopher Nolan</td>
</tr>
<tr>
<td>16</td>
<td>Avatar</td>
<td>2009</td>
<td>Jim Cameron</td>
</tr>
<tr>
<td>53</td>
<td>Gravity</td>
<td>2013</td>
<td>Alfonso Cuaron</td>
</tr>
<tr>
<td>74</td>
<td>Blue Jasmine</td>
<td>2013</td>
<td>Woody Allen</td>
</tr>
</tbody>
</table>

Pages in a column store layout:

- 20, 16, 53, 74
- Inception, Avatar
- Gravity, Blue Jasmine

High potential for data compression!
## Column Store Layout

<table>
<thead>
<tr>
<th>MovieID</th>
<th>Name</th>
<th>Year</th>
<th>Director</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>Inception</td>
<td>2010</td>
<td>Christopher Nolan</td>
</tr>
<tr>
<td>16</td>
<td>Avatar</td>
<td>2009</td>
<td>Jim Cameron</td>
</tr>
<tr>
<td>53</td>
<td>Gravity</td>
<td>2013</td>
<td>Alfonso Cuaron</td>
</tr>
<tr>
<td>74</td>
<td>Blue Jasmine</td>
<td>2013</td>
<td>Woody Allen</td>
</tr>
</tbody>
</table>

Pages in a column store layout:

- 20, 16, 53, 74
- Inception, Avatar
- Gravity, Blue Jasmine

High potential for data compression!

**Q:** When is column store “bad” for performance?
Outline

❖ Data Storage (Disks)
❖ Buffer Management
❖ File Organization
❖ New Storage Hardware
Outline

❖ Data Storage (Disks)
❖ Buffer Management
❖ File Organization
❖ New Storage Hardware
Storage/Memory Hierarchy

- **Flash Storage**
  - Access Speed: $10^7 - 10^8$
  - Cycles: $10^5 - 10^6$
  - Capacity: 100s
  - Price: Low

- **Magnetic Hard Disk Drive (HDD)**
  - Access Speed: $10^7 - 10^8$
  - Cycles: $10^5 - 10^6$
  - Capacity: Magnetic
  - Price: Medium

- **Tape**
  - Access Speed: $10^7 - 10^8$
  - Cycles: $10^5 - 10^6$
  - Capacity: Tape
  - Price: High

- **Non-Volatile Memory?**

- **CPU**
  - Access Speed: Fast
  - Cycles: Few
  - Price: Expensive
Flash Solid State Drive vs Hard Disks
Roughly speaking, flash combines the speed benefits of RAM with persistence of disks.
Flash Solid State Drive vs Hard Disks

*Roughly speaking, flash combines the speed benefits of RAM with persistence of disks*

- Random reads/writes are not much worse
- “Locality of reference” different for data/file layout
- But still block-addressable like HDDs
- Data access latency: 100x faster!
- Data transfer throughout: Also 10-100x higher
- Parallel read/writes more feasible
- Cost per GB is 5-15x higher!
- Read-write impact asymmetry; much lower lifetimes
Flash SSDs in RDBMSs
Flash SSDs in RDBMSs

Q: How best to exploit flash SSDS for RDBMSs?
Flash SSDs in RDBMSs

**Q:** How best to exploit flash SSDs for RDBMSs?

- Various ideas explored in research:
  - Fully replace hard disks
  - Supplement HDDs; but to store which part of DB?
    - Just the “logs” of transactions
    - Index structures
    - “Hot” relations/data structures
    - …
  - Requires rethinking DBMS techniques!
Flash SSDs in RDBMSs

Q: How best to exploit flash SSDs for RDBMSs?

- Various ideas explored in research:
  - Fully replace hard disks
  - Supplement HDDs; but to store which part of DB?
    - Just the “logs” of transactions
    - Index structures
    - “Hot” relations/data structures
    - …
  - Requires rethinking DBMS techniques!

No consensus yet; “fully replace” becoming common
NVMs vs Disks
NVMs vs Disks

*Roughly speaking, NVMs are like persistent RAM, but with similar capacity as SSDs*
Roughly speaking, NVMs are like persistent RAM, but with similar capacity as SSDs

- Random R/W with less to no SSD-style wear and tear
- Byte-addressability (not blocks like SSDs/HDDs)
- “Locality of reference” concept radically changes
- Latency, throughput, parallelism, etc. similar to RAM
- Yet to see light of day in production settings
- Cost per GB: No one knows yet. :)
Recap

❖ Data Storage (Disks)
❖ Storage hierarchy
❖ Disk architecture and access times

❖ Buffer Management
❖ Buffer manager and buffer pool
❖ Buffer replacement algorithms (LRU, clock, etc.)

❖ File Organization
❖ Page layouts (heap files)
❖ Record layouts (fixed, variable, columnar)

❖ New Storage Hardware