

CSE 132C

Database System Implementation

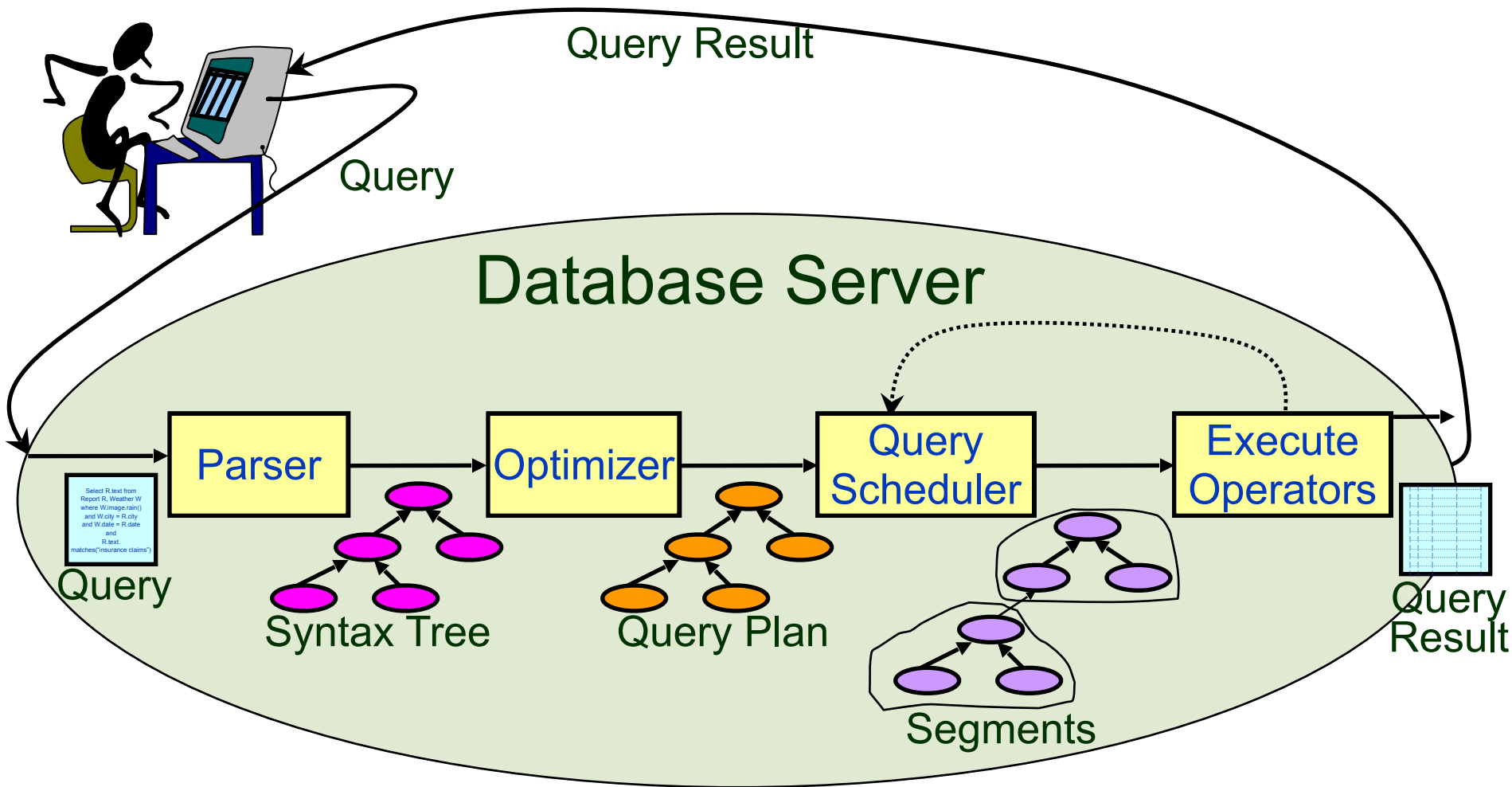
Arun Kumar

Topic 1: Data Storage Management

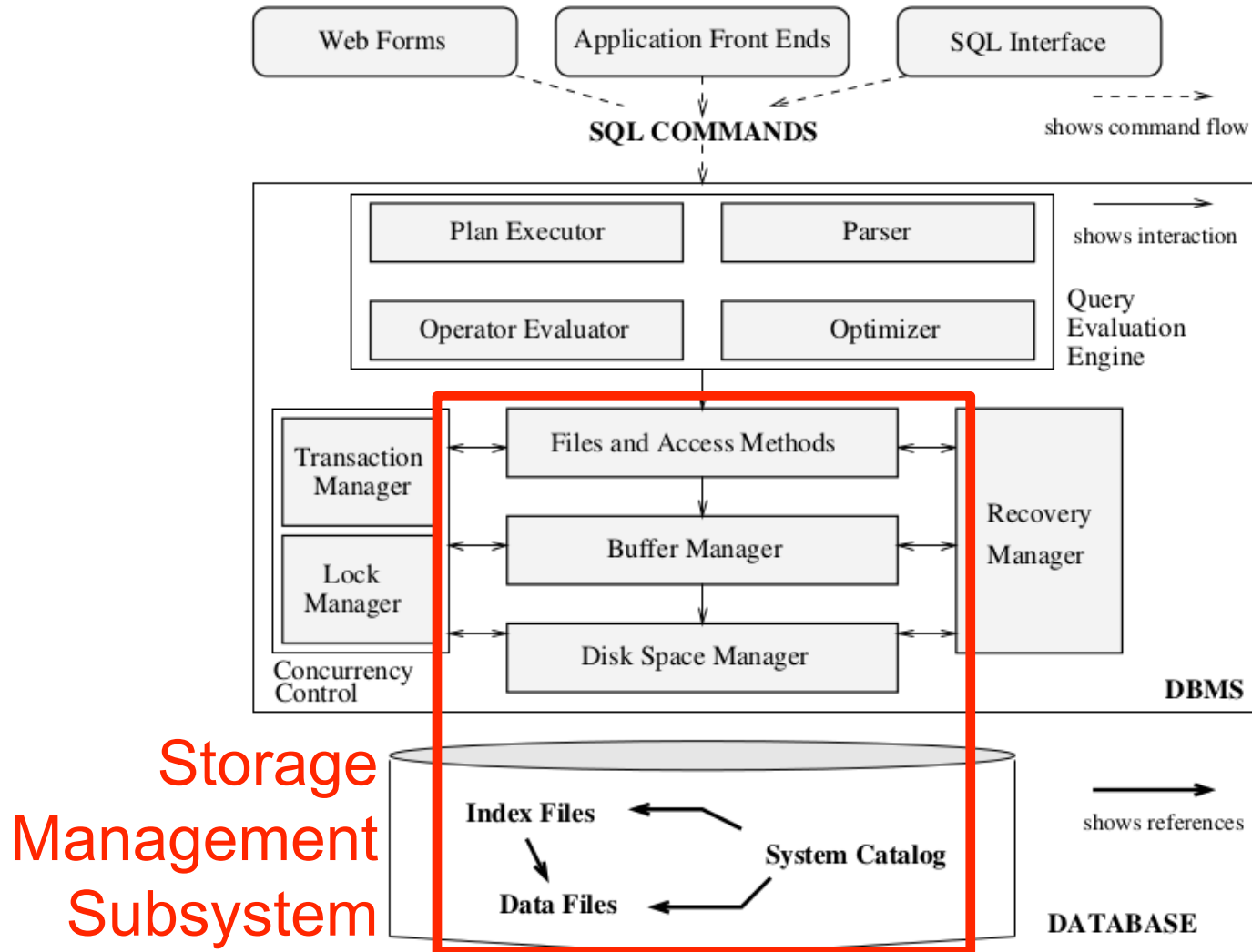
Chapters 8 and 9 (except 8.5.4 and 9.2) of Cow Book

Slide ACKs: Jignesh Patel, Paris Koutris

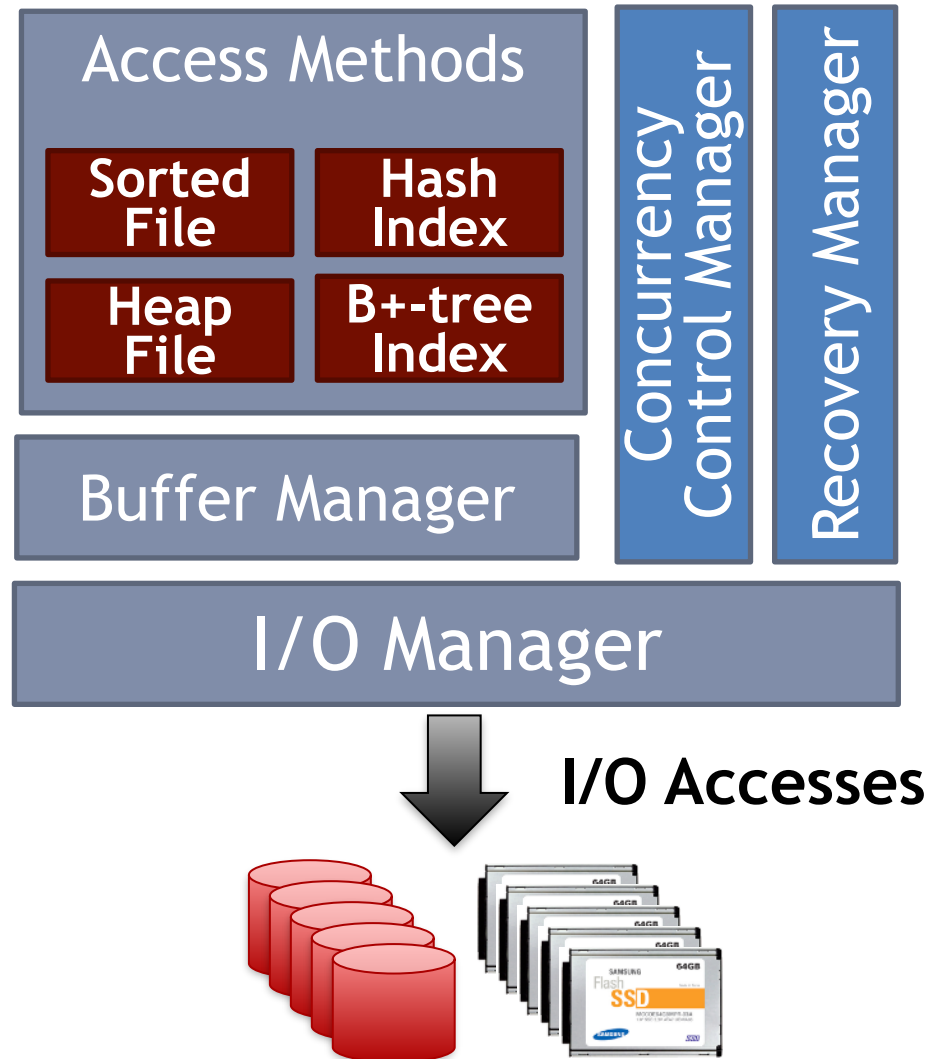
Lifecycle of an SQL Query



RDBMS Architecture



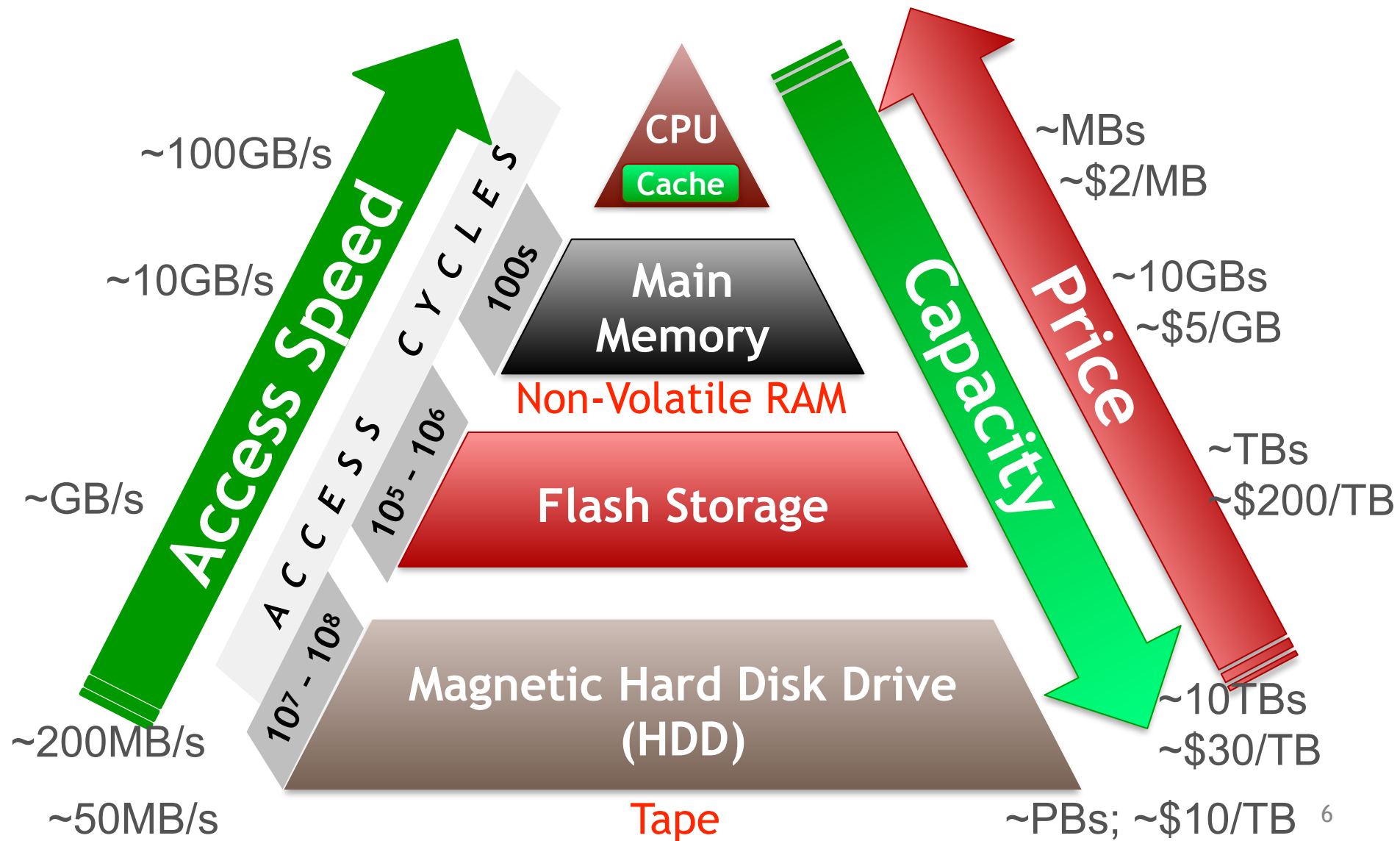
Another View of Storage Manager



Outline

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

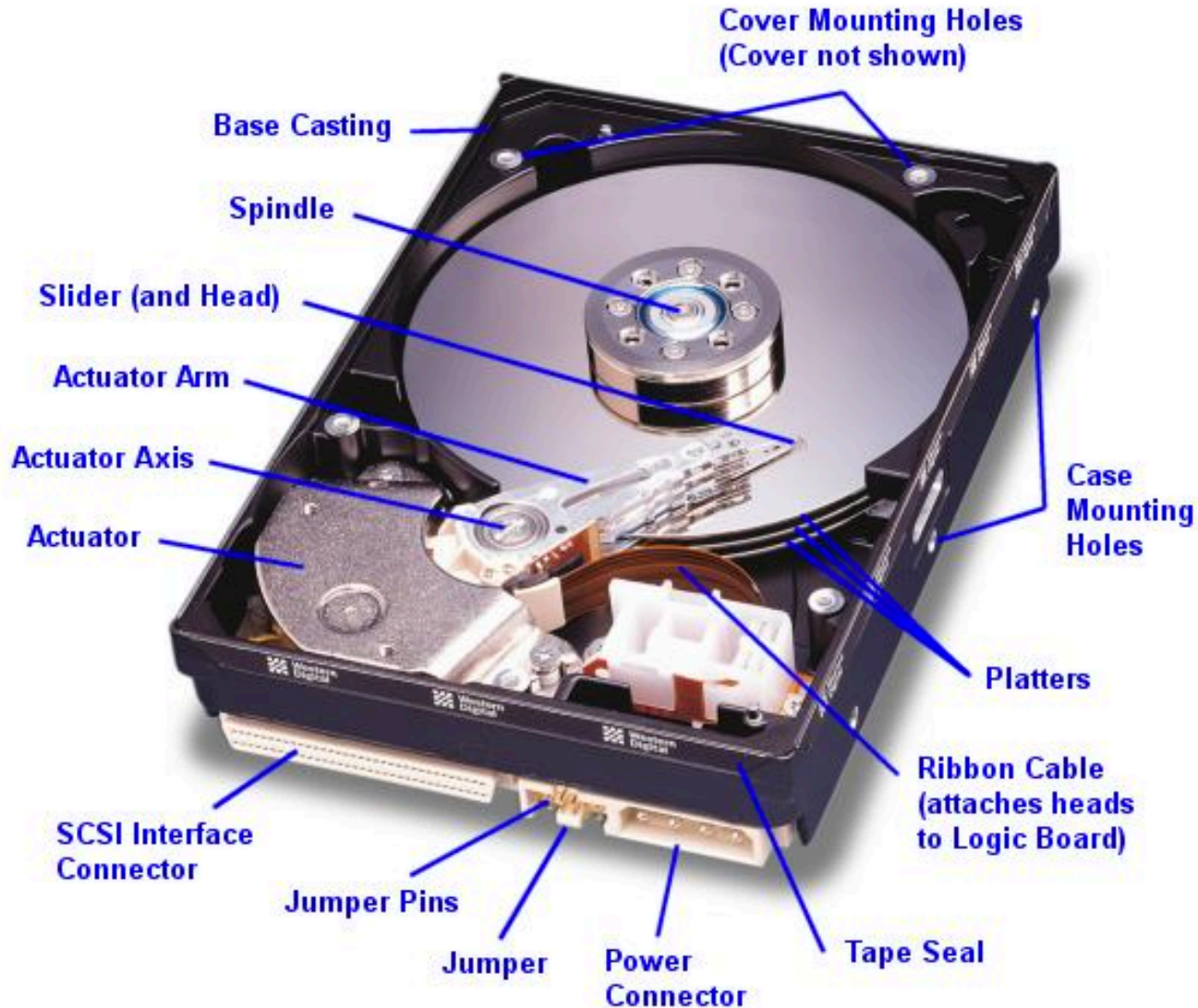
Memory/Storage Hierarchy



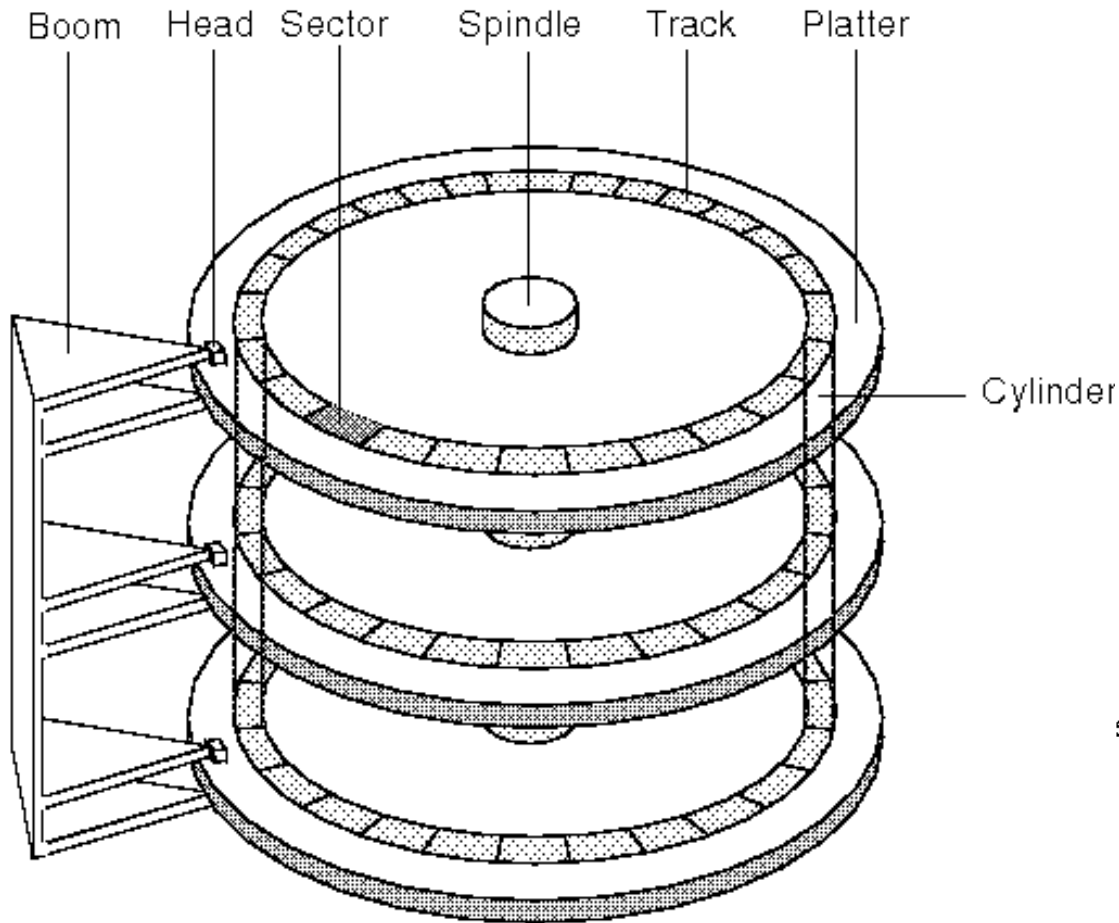
Magnetic Hard Disk (Disk for short)

- ❖ Widely used secondary storage device
- ❖ Data storage/retrieval units: **disk blocks** or **pages**
- ❖ Unlike DRAM, 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!

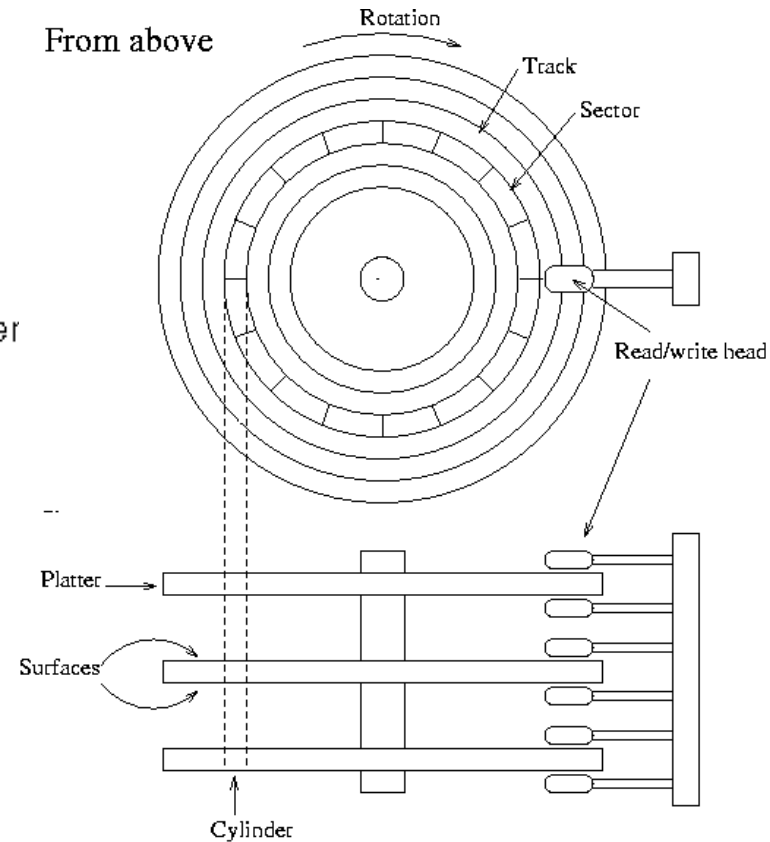
Components of a Disk



Components of a Disk

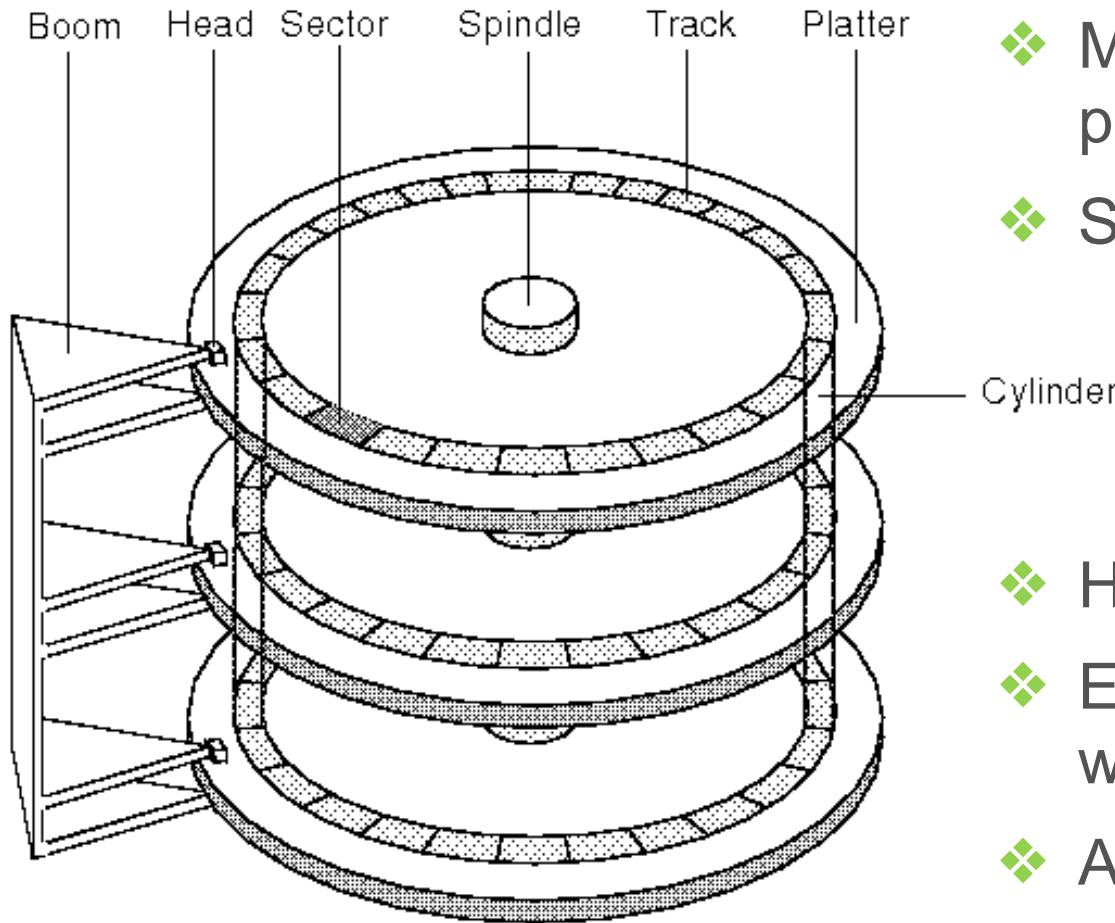


Anatomy of a regular hard disk



1 block = n contiguous sectors
(n fixed during disk configuration) 9

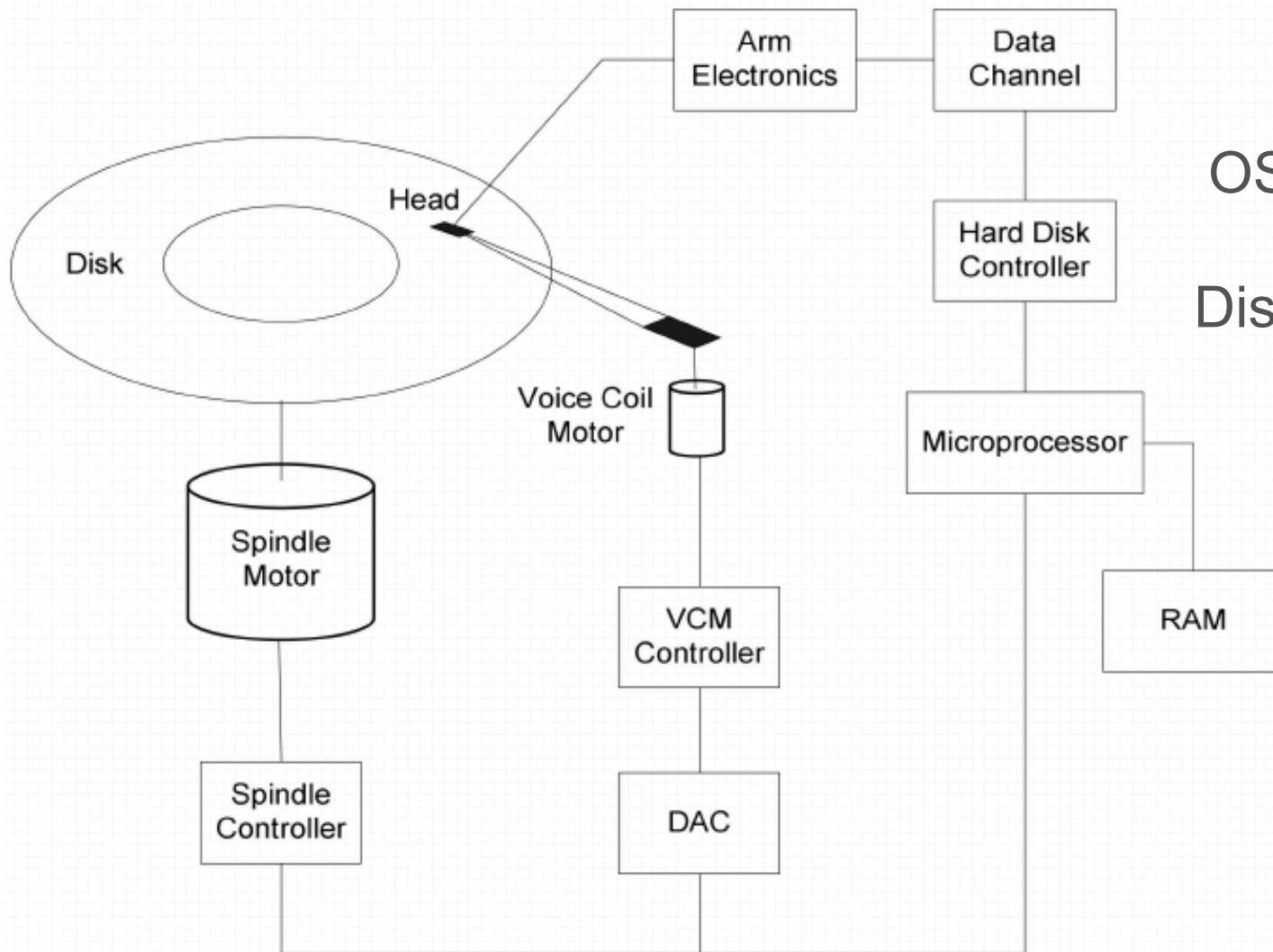
How does a Disk Work?



Anatomy of a regular hard disk

- ❖ 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?



OS interfaces
with the
Disk Controller

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)

Capacity	1TB
RPM	7200
Transfer	6 Gb/s
#Platters	Just 1!
Avg Seek	9ms
Price	USD 33!

Data Organization on Disk

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

Disk Data Layout Principles

- ❖ **Sequential** access vs. **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

1. Why does a magnetic hard disk have seek time?
2. If RPM of a given magnetic hard disk is 6000, what is the average rotational delay?
3. What causes the random vs. sequential access latency dichotomy on a magnetic hard disk?
4. Which part(s) of the 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?

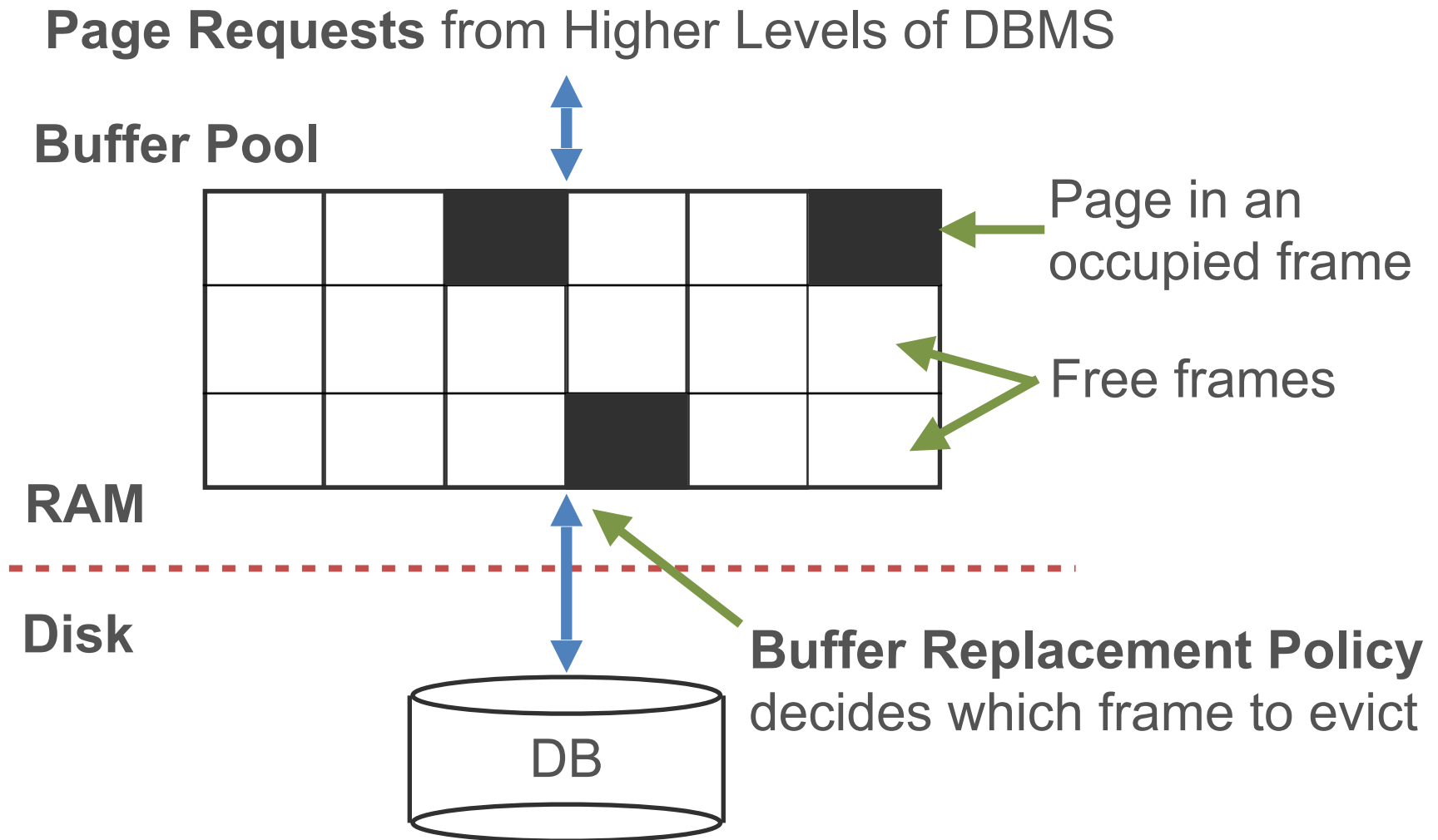
Outline

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

Buffer Management

- ❖ Pages should be in DRAM for DBMS query processor
 - ❖ But not all pages of a database might fit in DRAM!
- ❖ **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 to Buffer Manager

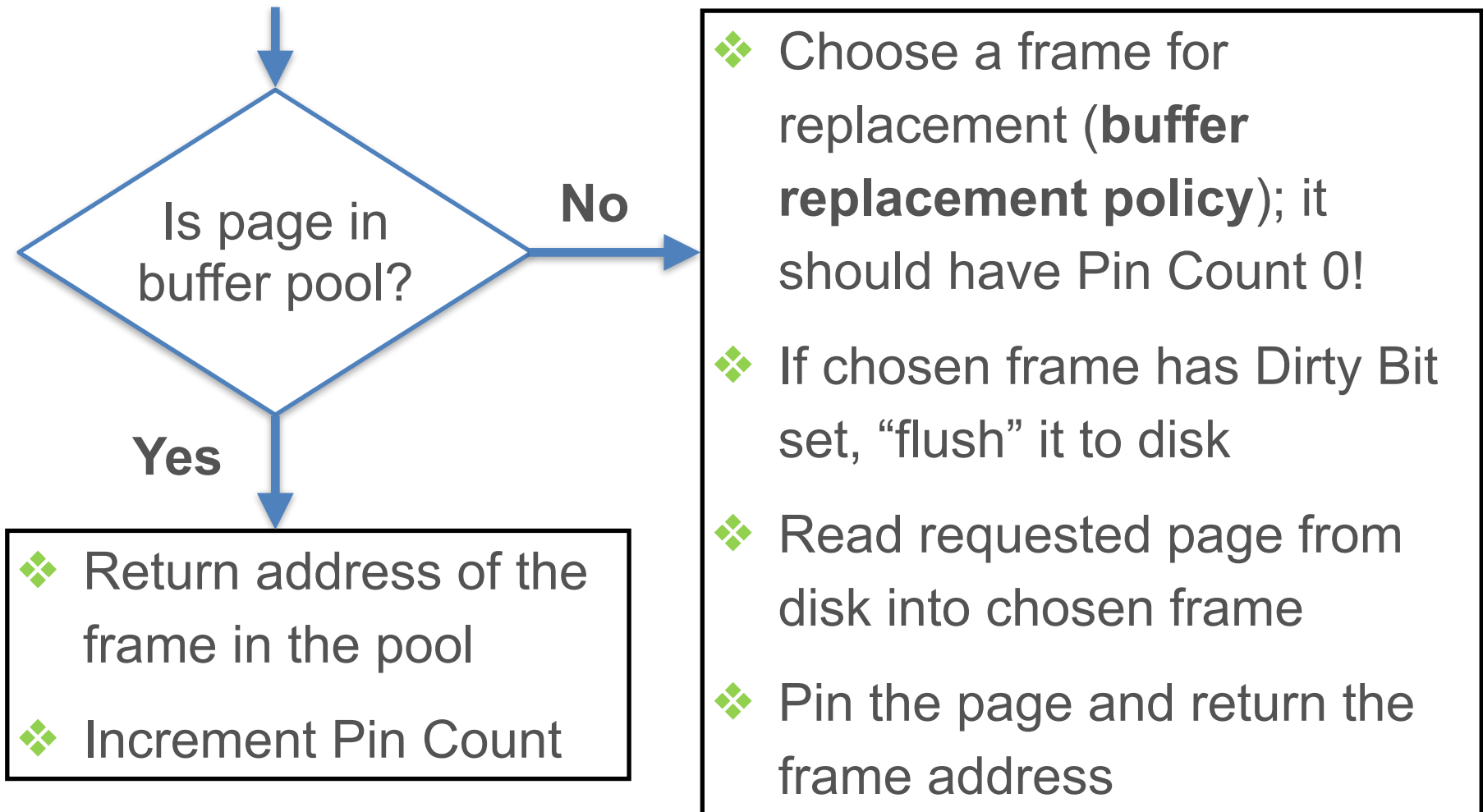
- ❖ **Request** a page for query processing (read or write)
- ❖ **Release** a page when no longer needed
- ❖ **Notify** if a page is modified (a write op happened)

Buffer Manager's Bookkeeping

- ❖ 2 variables per buffer frame maintained
- ❖ **Pin Count**
 - ❖ Current number of “users” of the page in the frame
 - ❖ “Pinning” means $\text{PinCount}++$; 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



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

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)

	Buf. Pool			Release B			Release A			
Page	-	-	-	A	B	D	A	B	D	
PC	0	0	0	1	0	1	0	0	1	
DB	0	0	0	1	0	0	1	0	0	and so on

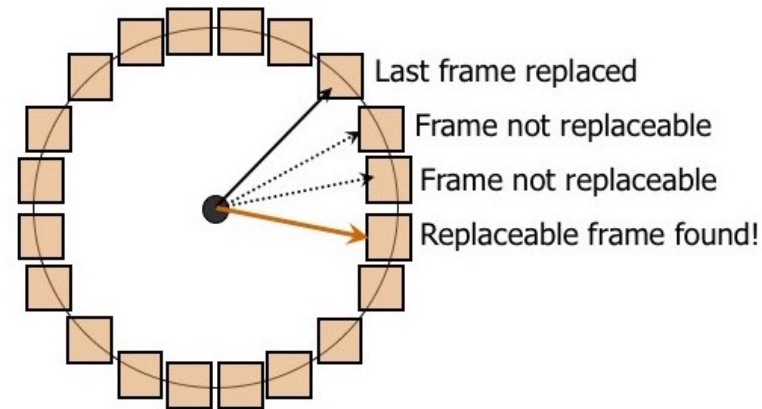
Request A			Request D			Request E			Release C		
A	-	-	A	B	D	A	E	D	C	E	D
1	0	0	1	1	1	0	1	1	0	1	1
0	0	0	1	0	0	1	0	0	0	0	0

Request B			Modify A			Request C			Flush A!		
A	B	-	A	B	-	C	E	D			
1	1	0	1	1	0	1	1	1			
0	0	0	1	0	0	0	0	0			

Queue of pointers: **F1** F0

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

- ❖ 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 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**

Data Organization Basics: Recap

- ❖ Disk space organized into **files**; relations are stored as files
- ❖ Files are made up of **pages**
 - ❖ File data (de-)allocated in increments of disk pages
- ❖ Pages contain **records** (physical version of tuples)
 - ❖ Higher levels operate on sets of records

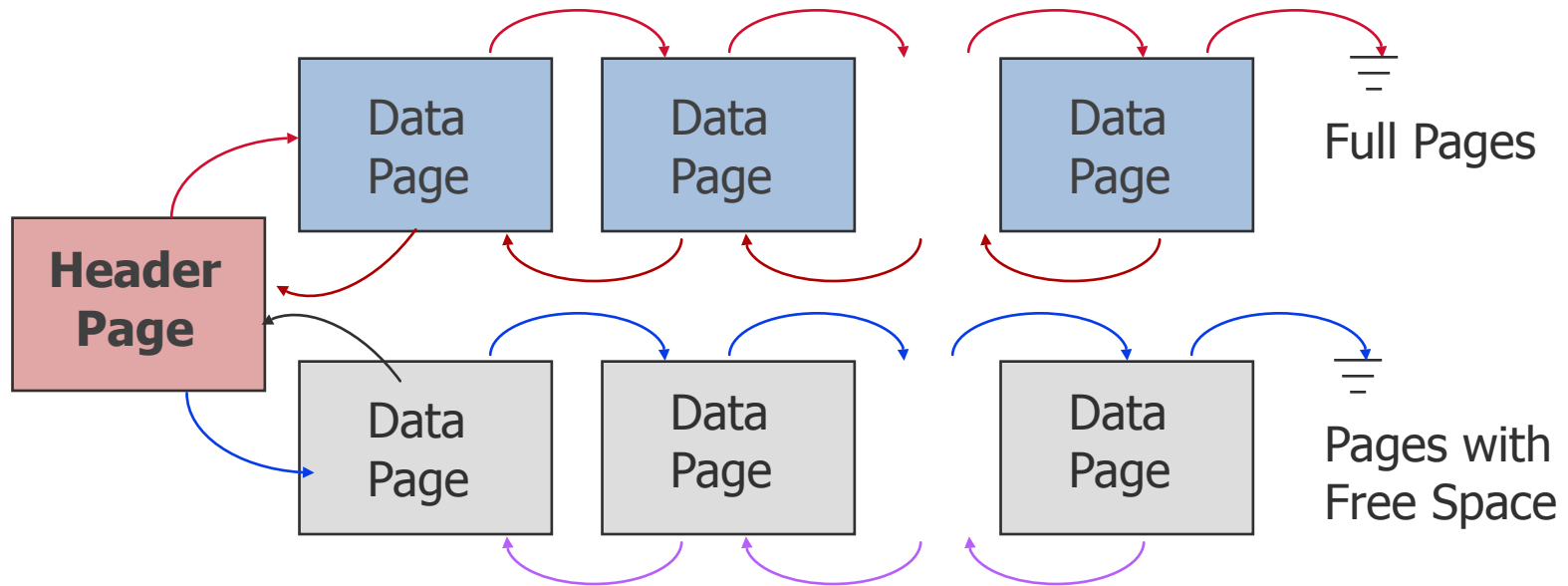
How pages are organized in a file: **Page Layout**

How records are organized in a page: **Record Layout**

Unordered (Heap) Files

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

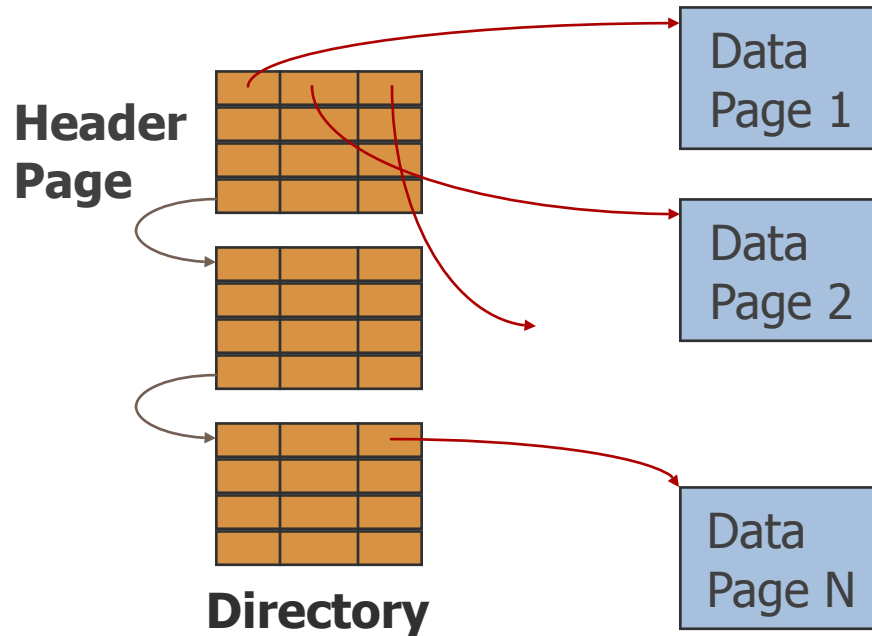
Heap File as Linked Lists



- ❖ (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



- ❖ 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

- ❖ Disk space organized into **files**; relations are stored as files
- ❖ Files are made up of **pages**
 - ❖ File data (de-)allocated in increments of disk pages
- ❖ **Pages contain records** (physical version of 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**

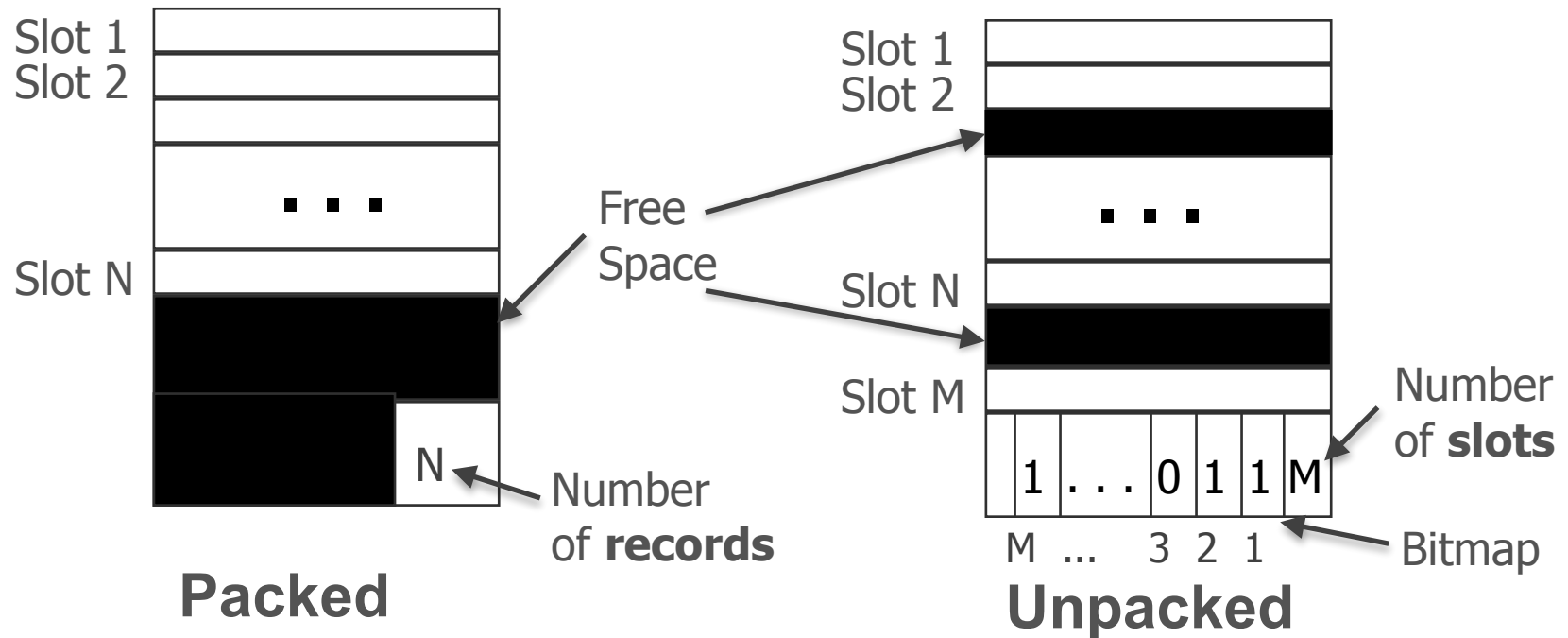
Record Layout Desiderata

- ❖ Higher levels (queries) operate on sets of records
- ❖ 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

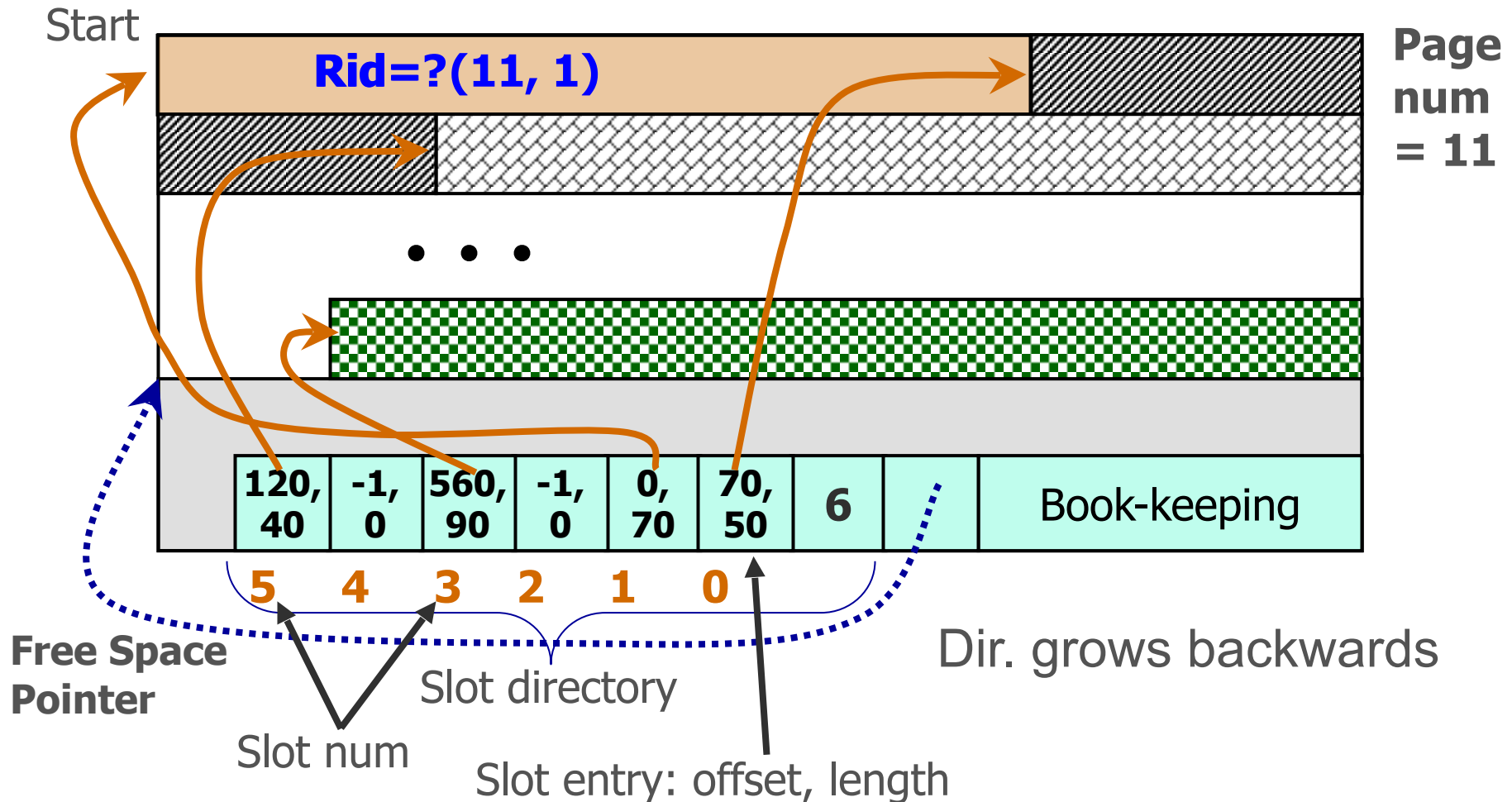
- ❖ Layout of fixed-length records:
 - ❖ Packed layout
 - ❖ Unpacked layout
- ❖ Layout of variable-length records
- ❖ Record format for fixed-length records
- ❖ Record formats for variable-length records
 - ❖ Delimiter-based
 - ❖ Pointer-based

Layout of Fixed-length Records



- ❖ Recall that **RecordID** = **<PageID, SlotNumber>**
- ❖ Con for Packed: moving/deleting records alter RIDs
- ❖ Con for Unpacked: extra space used by bitmap

Layout of Variable-length Records

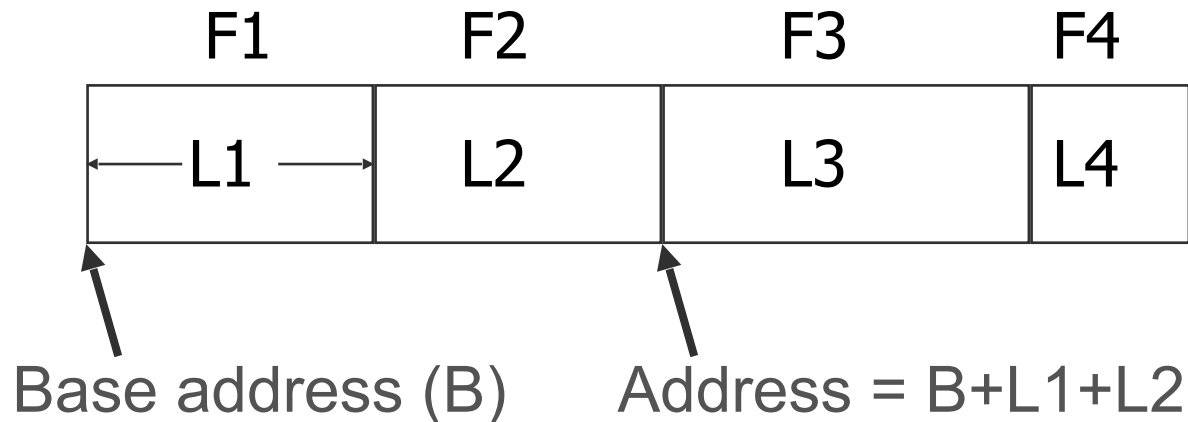


Layout of Variable-length Records

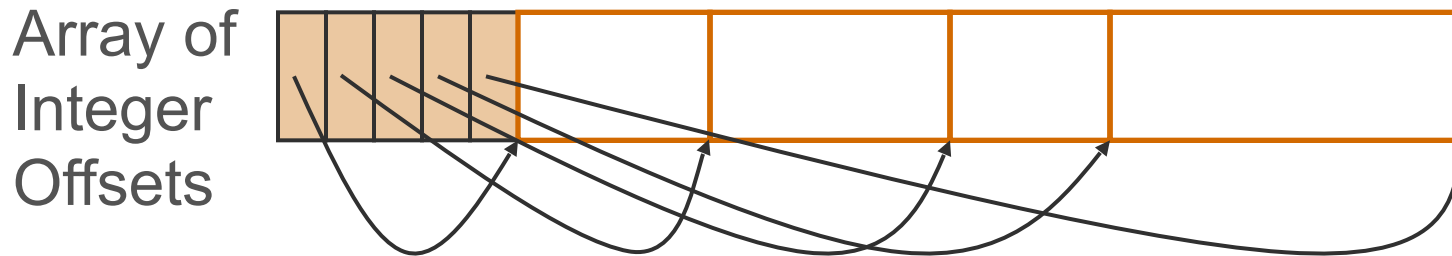
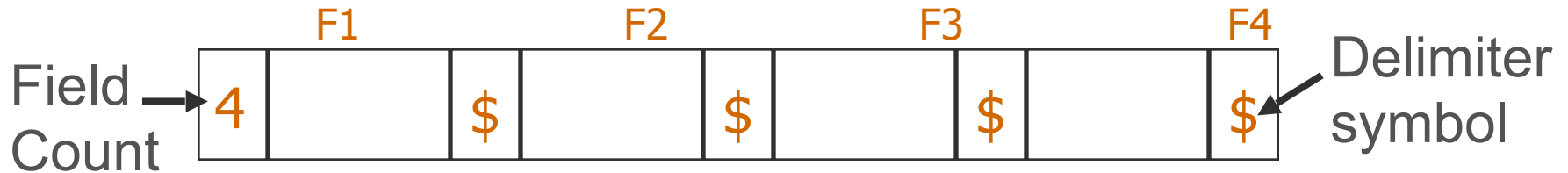
- ❖ A big pro: moving records on page does not alter RIDs
- ❖ Good for fixed-length records too
- ❖ Deleting a record: offset is set to -1
- ❖ Inserting a new record:
 - ❖ Any available slot can be used (incl. in free space)
 - ❖ If not enough free space, reorganize

Fixed-length Record Format

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



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:

Movies (M)

<u>MovieID</u>	Name	Year	Director
----------------	------	------	----------

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

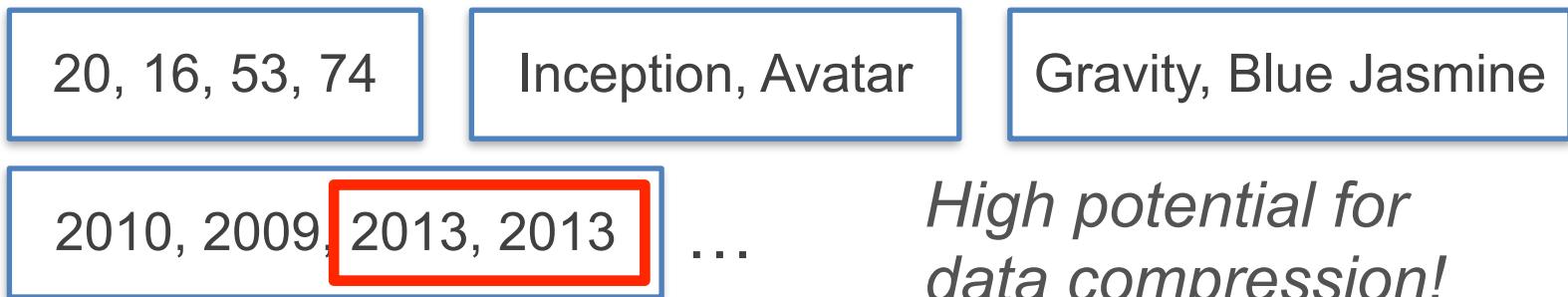
Q: Why bother reading other attributes?

- ❖ Most “analytics” queries read only one or a few attributes; reading other attributes wastes I/O time
- ❖ “Column store” RDBMSs lay out relations in column-major order to help speed up such queries

Column Store Layout

<u>MovieID</u>	Name	Year	Director
20	Inception	2010	Christopher Nolan
16	Avatar	2009	Jim Cameron
53	Gravity	2013	Alfonso Cuaron
74	Blue Jasmine	2013	Woody Allen

Pages in a column store layout:

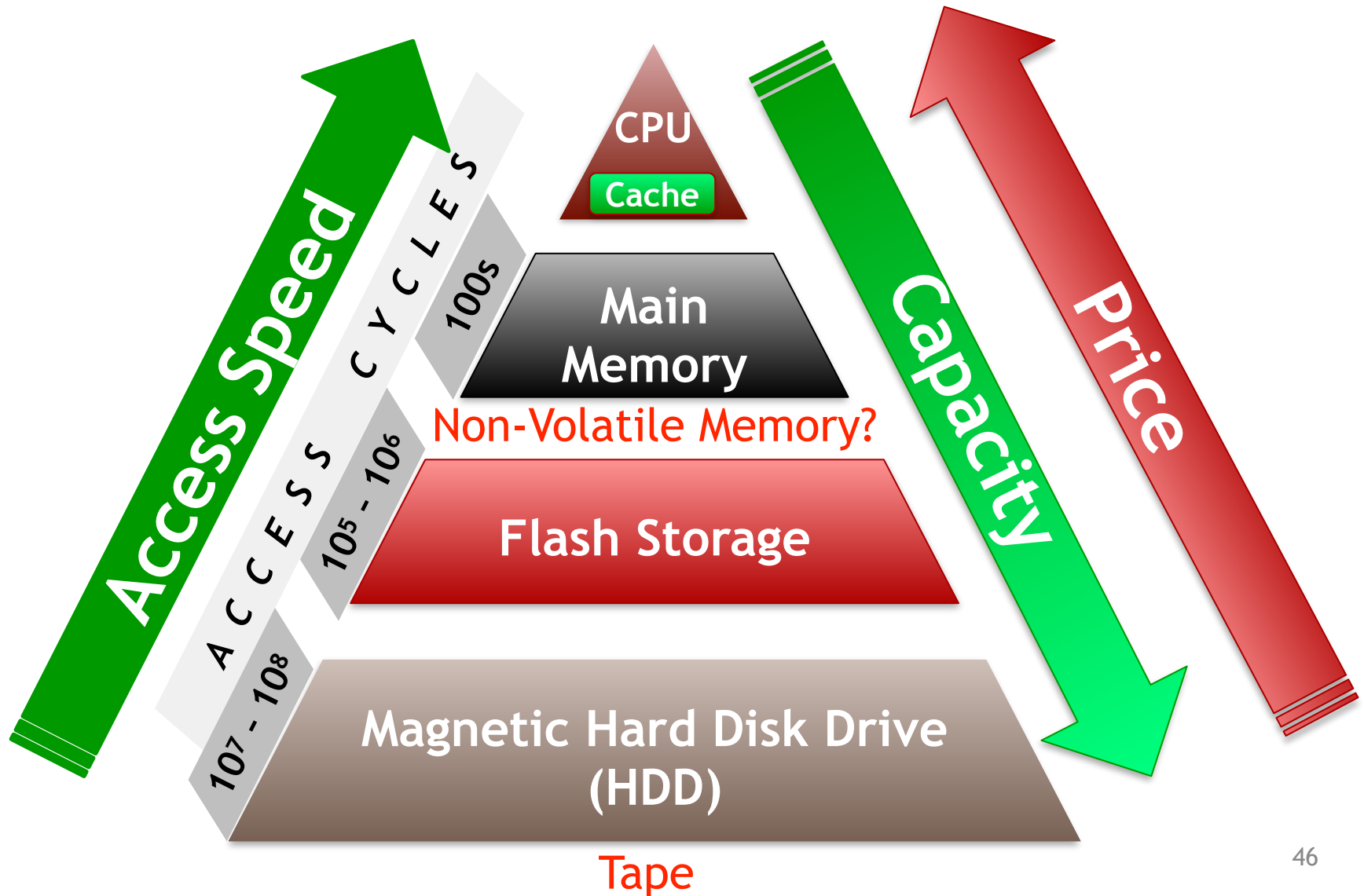


Q: When might a column store cause poorer performance?

Outline

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

Storage/Memory Hierarchy



Flash Solid State Drive vs. Magnetic Disk

- ❖ Roughly speaking, flash SSD combines persistence of magnetic disks with speed closer to DRAM
- ❖ Random reads/writes not much worse than sequential
 - ❖ “Locality of reference” different for data/file layout
 - ❖ But still block-addressable like magnetic disks
- ❖ Data access latency: 100x faster!
- ❖ Data transfer throughput: Also 10-100x higher
- ❖ Parallel read/writes more feasible
- ❖ But cost per GB can be 3x-5x higher
- ❖ Read-write impact asymmetry leads to lower lifetimes

Flash SSDs in RDBMSs

Q: How best to exploit flash SSDs for RDBMSs?

- ❖ Various ideas explored in research:
- ❖ Fully replace magnetic disks
- ❖ Supplement HDDs; but to store which part of DB?
 - ❖ Just the “logs” of transactions
 - ❖ Index structures
 - ❖ “Hot” relations/data structures
- ❖ Requires reimplementing RDBMS internals
- ❖ No consensus yet but “fully replace” becoming common

NVMs vs. Disks

- ❖ Roughly speaking, NVMs are like persistent DRAM, 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 DRAM
- ❖ Yet to see light of day in production settings
- ❖ Cost per GB: No one knows yet. :)
- ❖ Active area of research in comp. arch. + DB systems

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**