

CS150A Database

Lu Sun

School of Information Science and Technology

ShanghaiTech University

Sept. 13, 2022

Today:

- DBMS's Architecture
- Disks and Buffers
 - Storage Media: Disk&SSD
 - Disk Space Management

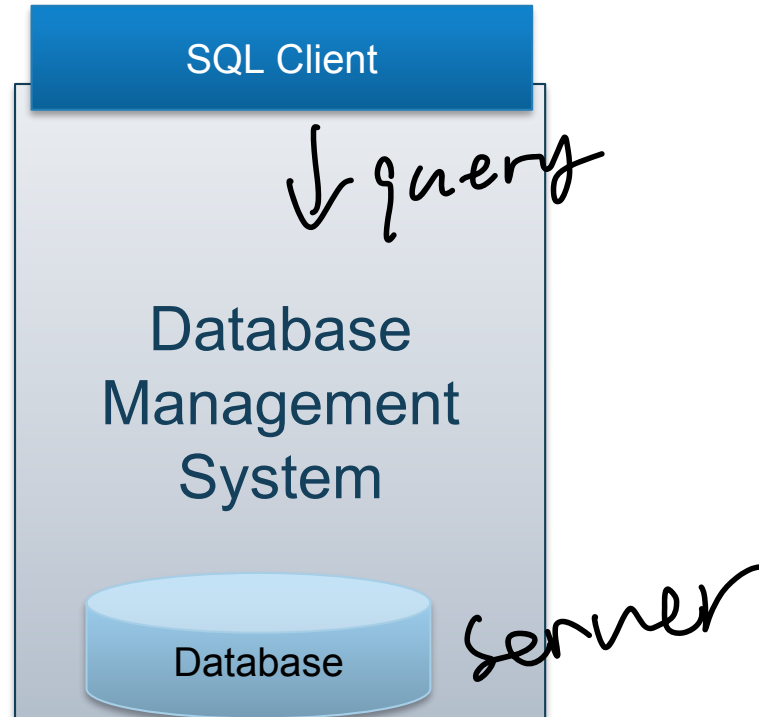
Readings:

- Database Management Systems (DBMS), Chapter 9
- Lecture note Disk Files

BIG PICTURE: ARCHITECTURE OF A DBMS

Architecture of a DBMS: SQL Client

- Last few lectures: SQL
- Next:
 - How is a SQL query executed?



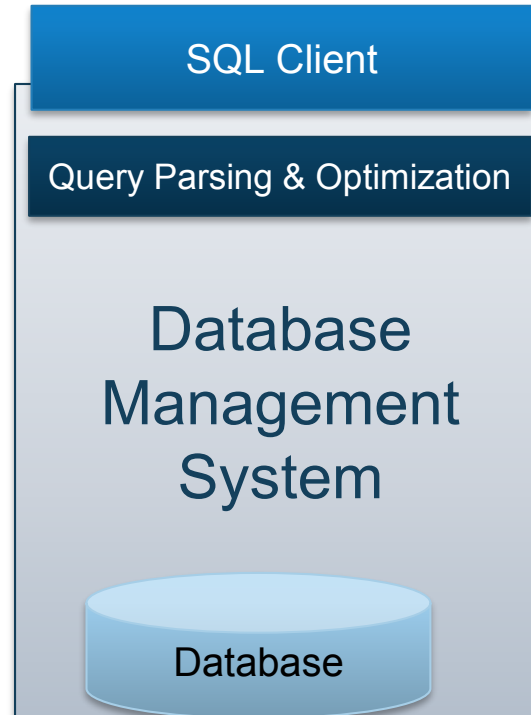
DBMS: Parsing & Optimization

Purpose:

Parse, check, and verify the SQL

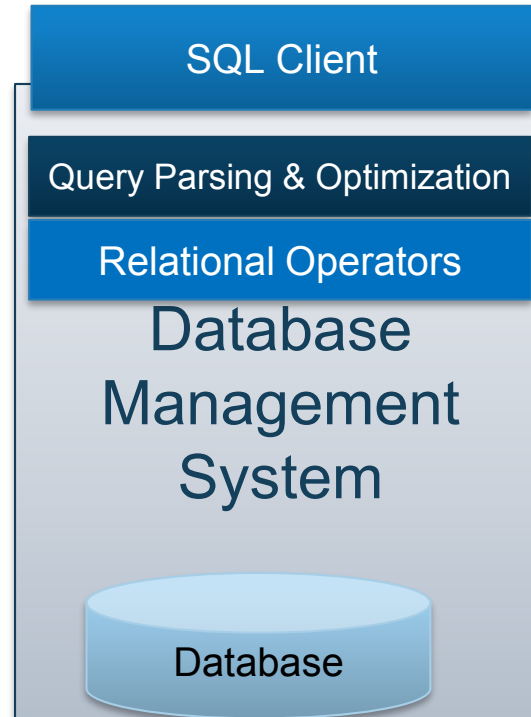
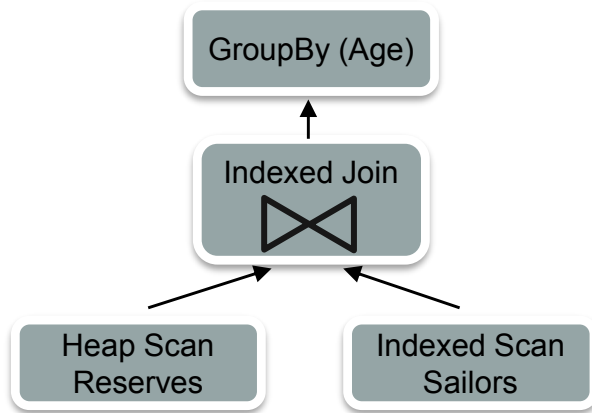
```
SELECT S.sid, S.sname, R.bid  
FROM Sailors R, Reserves R  
WHERE S.sid = R.sid and S.age > 30  
GROUP BY age
```

And translate into an efficient
relational query plan



DBMS: Relational Operators

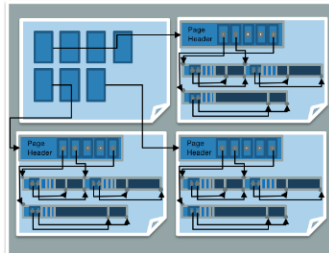
Purpose: Execute a dataflow by operating on **records** and **files**



DBMS: Files and Index Management

Purpose: Organize tables and Records as groups of pages in a logical file

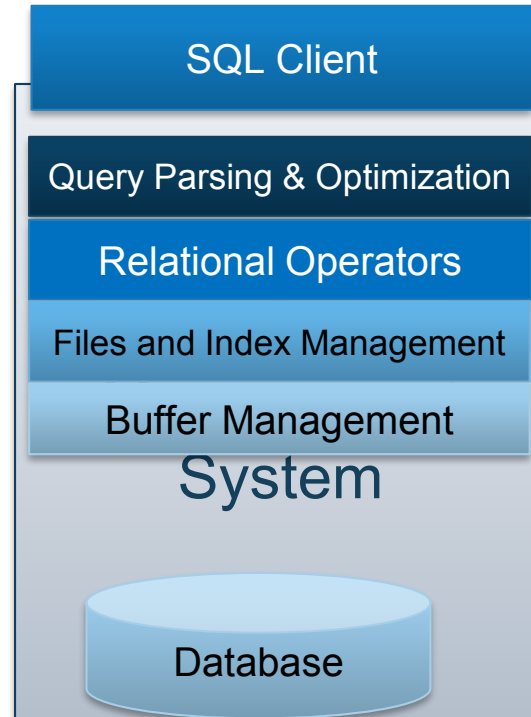
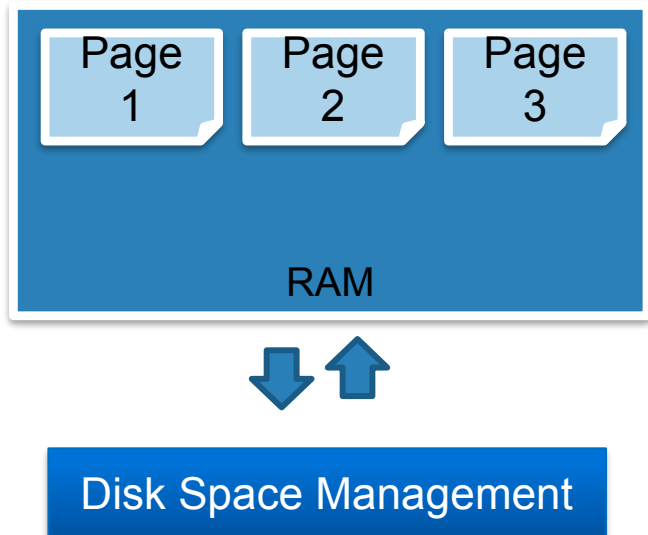
SSN	Last Name	First Name	Age	Salary
123	Adams	Elmo	31	\$400
443	Grouch	Oscar	32	\$300
244	Oz	Bert	55	\$140
134	Sanders	Ernie	55	\$400



DBMS: Buffer Management

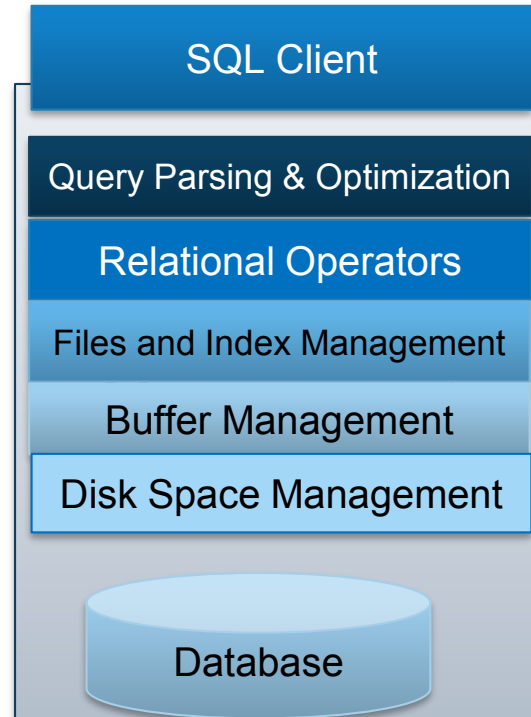
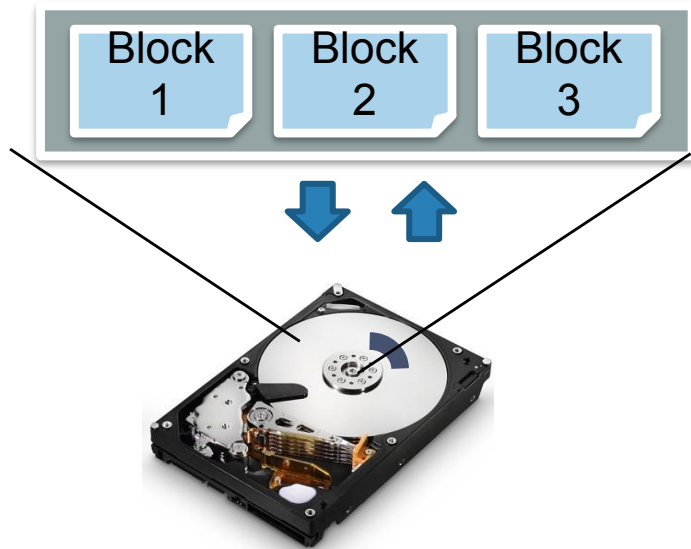
Purpose:

Provide the illusion of operating in memory



DBMS: Disk Space Management

Purpose: Translate page requests into **physical bytes** on one or more device(s) *hardware*



Architecture of a DBMS

- Organized in layers
- Each layer abstracts the layer below
 - Manage complexity
 - Performance assumptions
- Example of good systems design



DBMS: Concurrency & Recovery

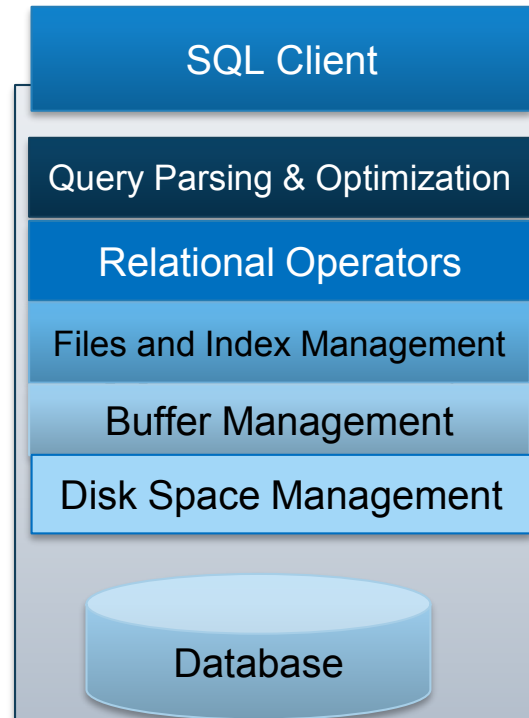
Two cross-cutting issues related to storage and memory management:

Multi user at same time

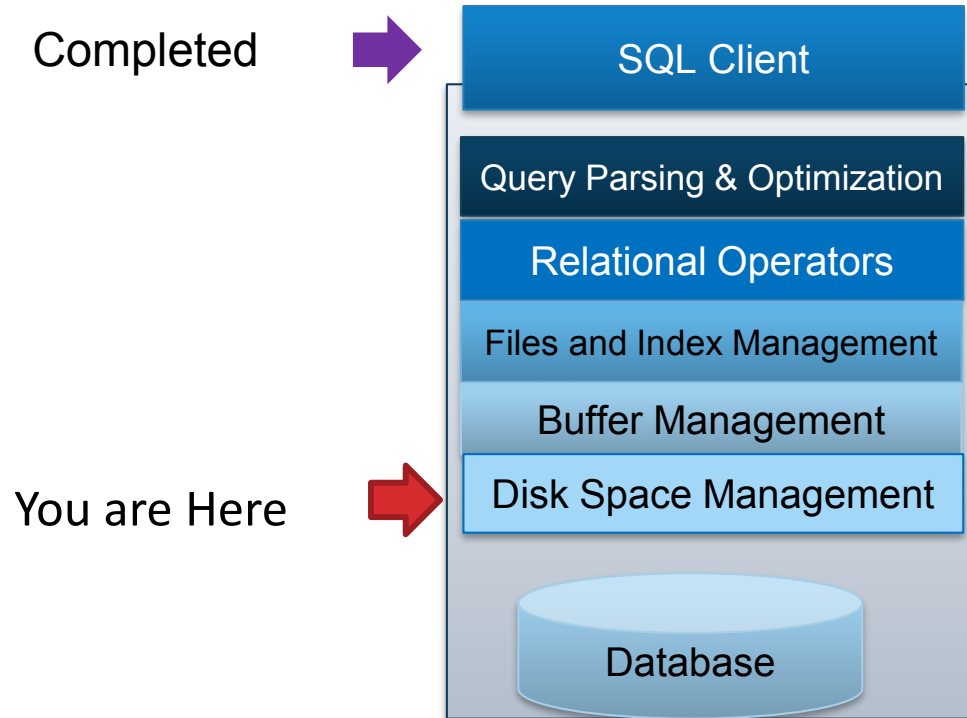


if crash recovery

single user



Context



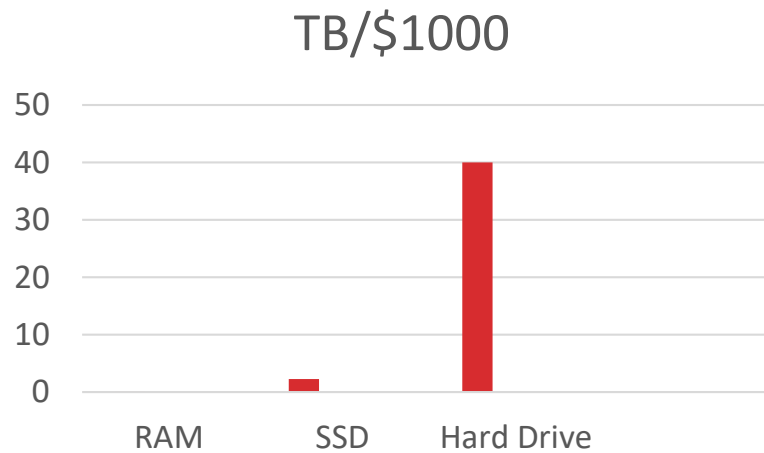
BEFORE WE BEGIN: **STORAGE MEDIA** (Hardware)

Disks

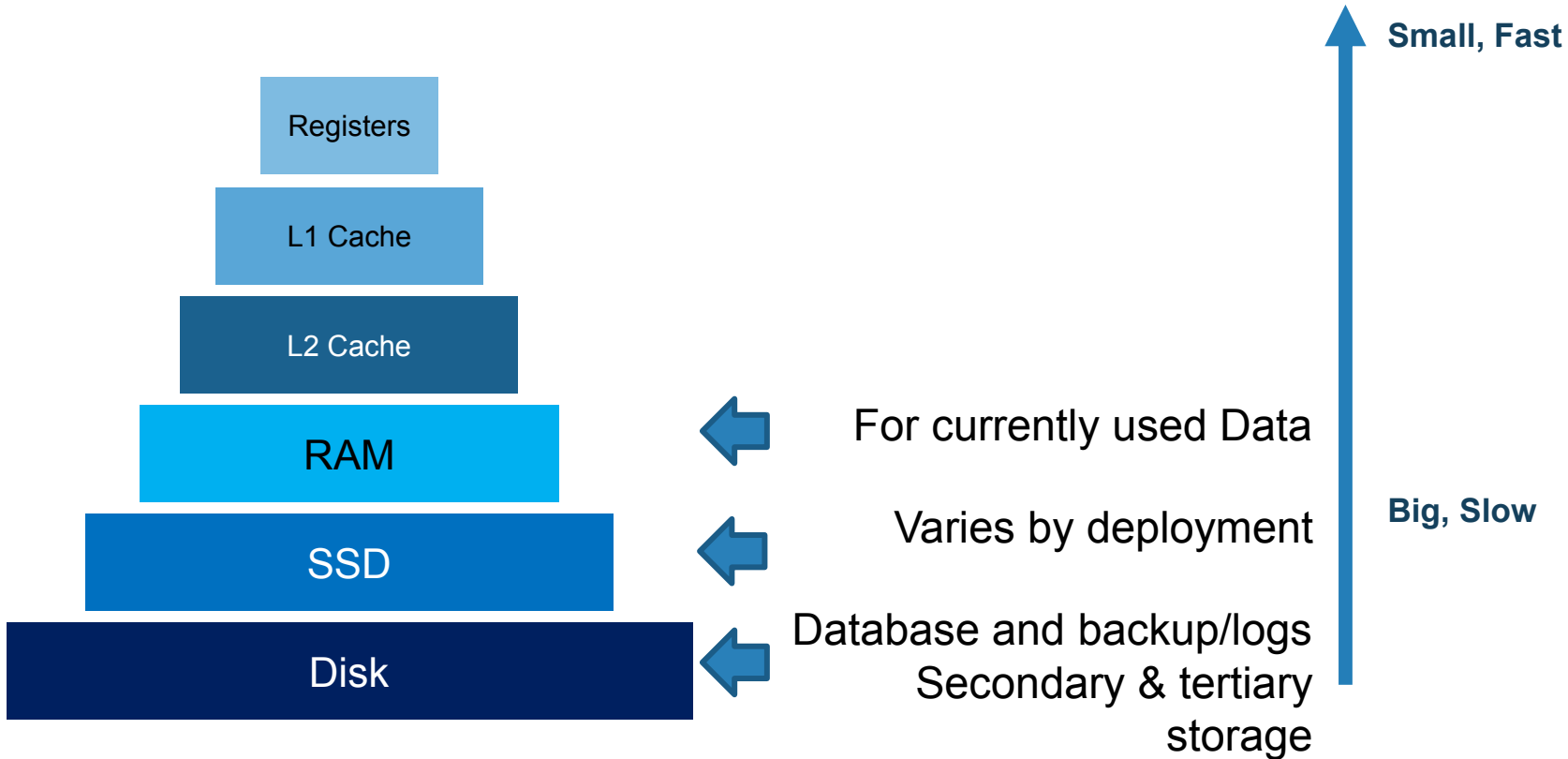
- Most database systems were originally designed for magnetic disks
 - Disk are a mechanical anachronism!
 - Instilled design ideas that apply to using solid state disks as well
- Major implications! **API**
 - **No “pointer derefs”**. Instead, an API:
 - READ: transfer “page” of data from disk to RAM.
 - WRITE: transfer “page” of data from RAM to disk.
 - Both API calls are very, **very slow!**
 - Plan carefully!
 - An explicit API can be a good thing
 - **Minimizes the kind of pointer errors** you see in C

Economics

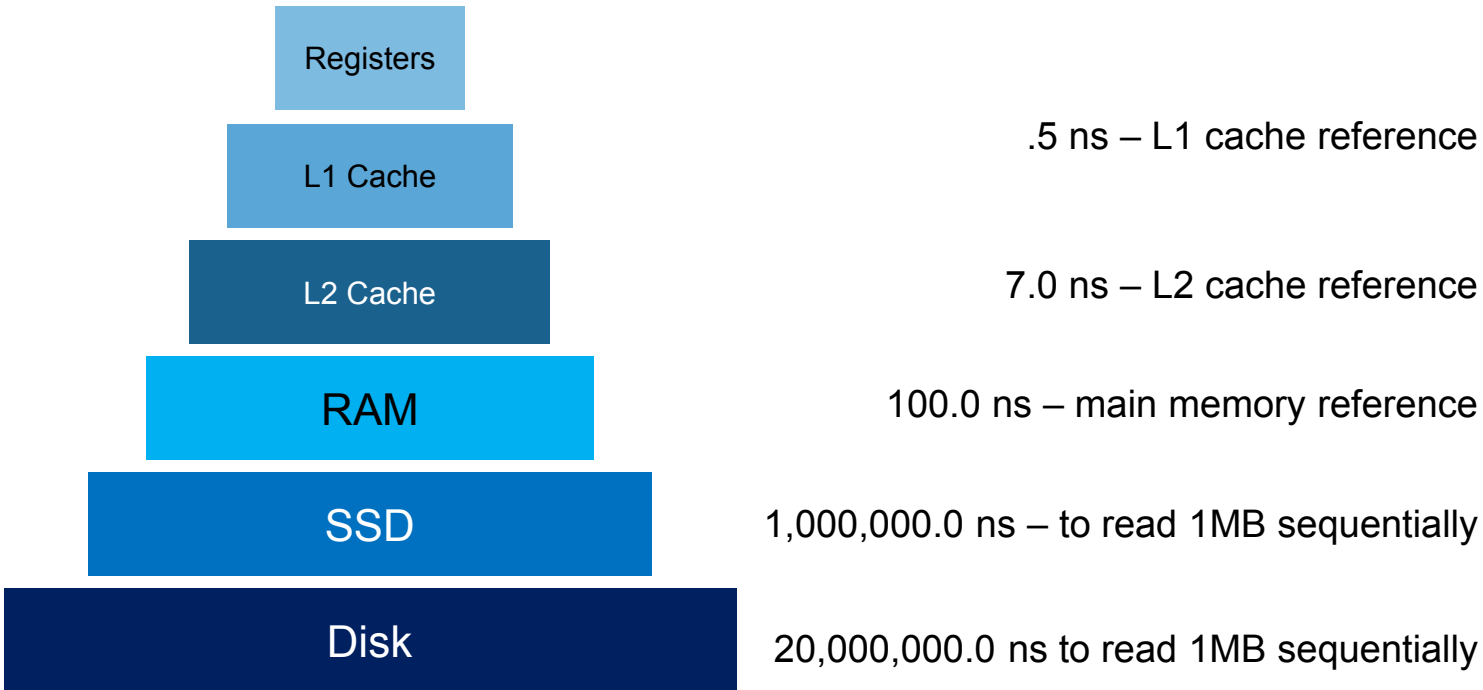
- \$1000 at NewEgg 2018:
 - Mag Disk: ~40TB for \$1000
 - SSD: ~2.3TB for \$1000
 - RAM: 80GB for \$1000



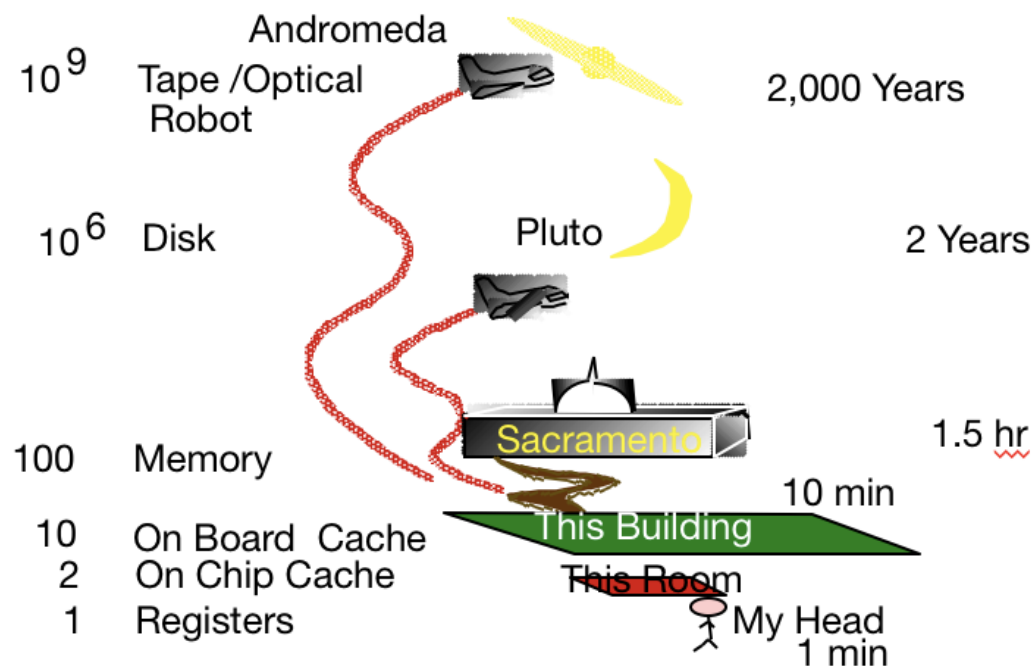
Storage Hierarchy



Hierarchy - Storage Latencies

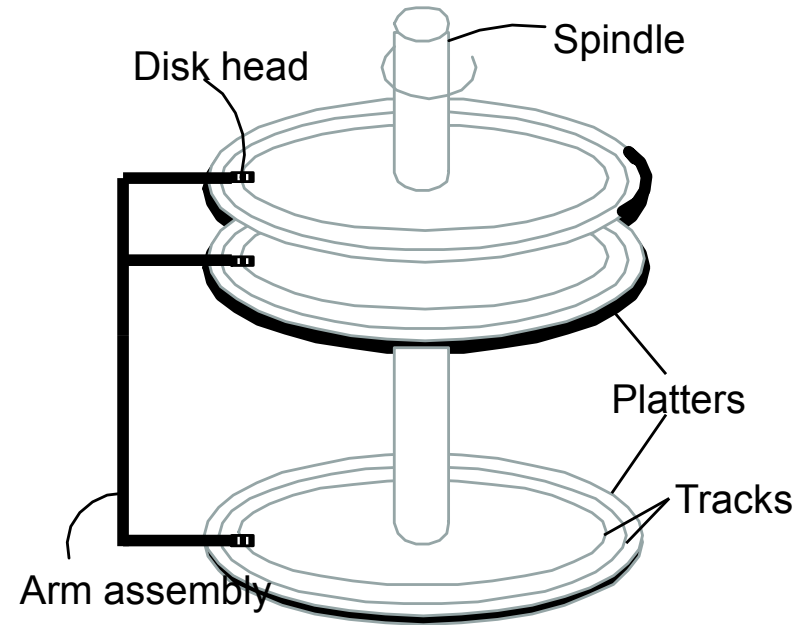


How Far Away is the Data?



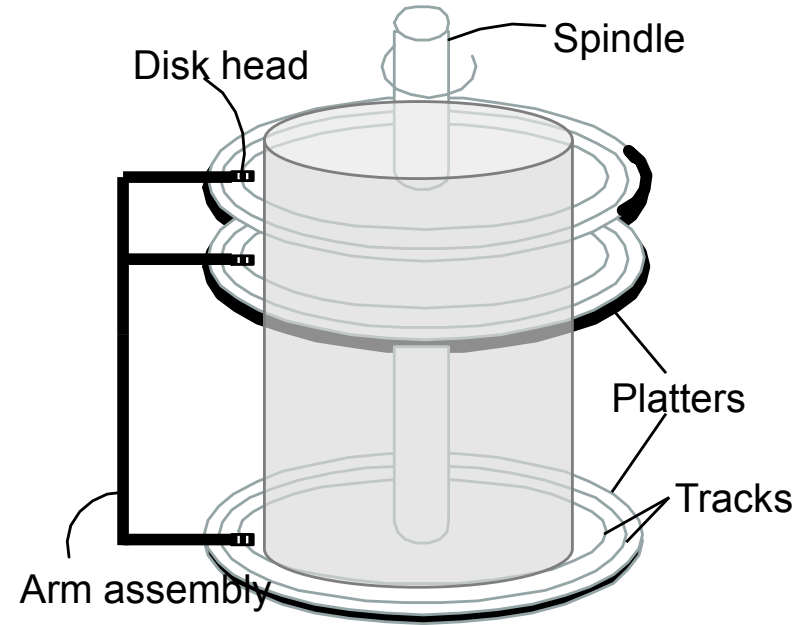
Components of a Disk, Pt. 1

- **Platters** spin (say 15000 rpm)
- **Arm assembly** moved in or out to position a **head** on a desired **track**
 - Tracks under heads make a “cylinder”



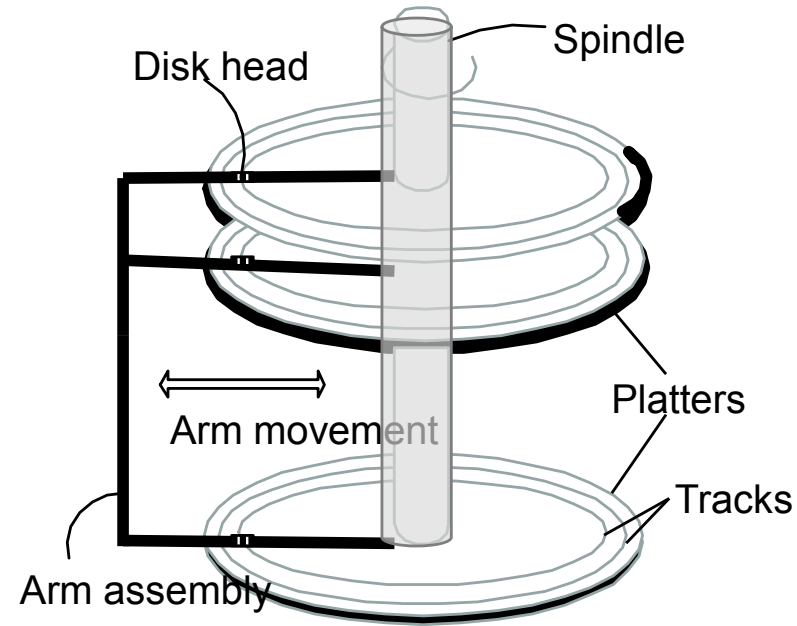
Components of a Disk, Pt. 2

- **Platters** spin (say 15000 rpm)
- **Arm assembly** moved in or out to position a **head** on a desired **track**
 - Tracks under heads make a “cylinder”



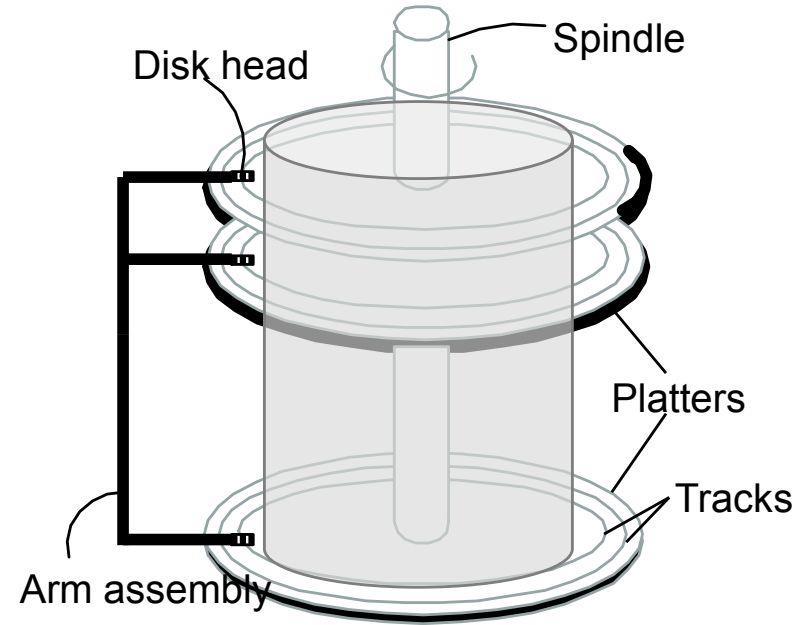
Components of a Disk, Pt. 3

- **Platters** spin (say 15000 rpm)
- **Arm assembly** moved in or out to position a **head** on a desired **track**
 - Tracks under heads make a “cylinder”



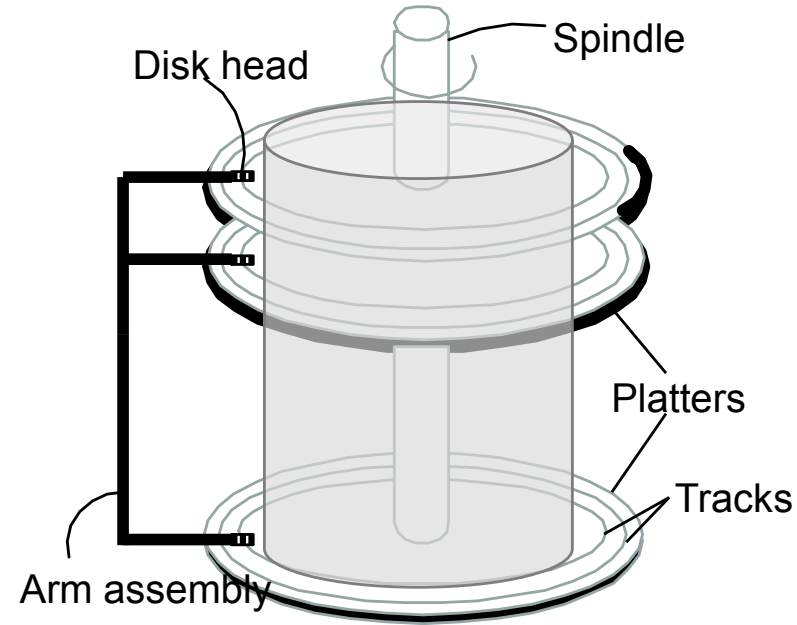
Components of a Disk, Pt. 4

- **Platters** spin (say 15000 rpm)
- **Arm assembly** moved in or out to position a **head** on a desired **track**
 - Tracks under heads make a “cylinder”



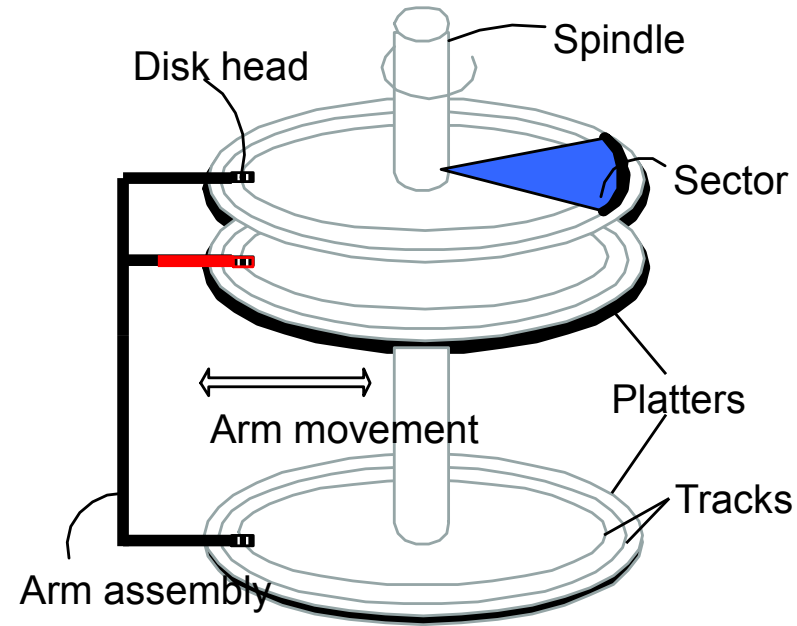
Components of a Disk, Pt. 5

- **Platters** spin (say 15000 rpm)
- **Arm assembly** moved in or out to position a **head** on a desired **track**
 - Tracks under heads make a “cylinder”
- Only one head reads/writes at any one time



Components of a Disk, Pt. 6

- **Platters** spin (say 15000 rpm)
- **Arm assembly** moved in or out to position a **head** on a desired **track**
 - Tracks under heads make a “cylinder”
- Only one head reads/writes at any one time
- Block/page size is a multiple of (fixed) **sector** size

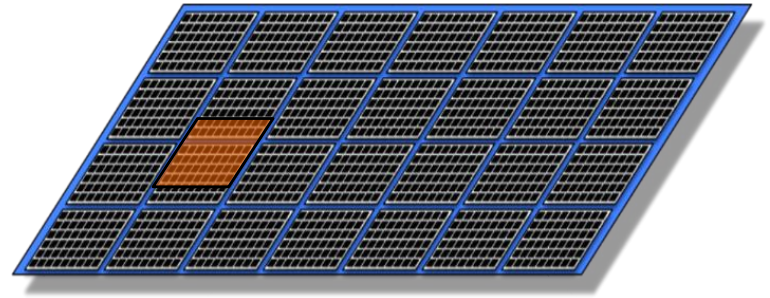


Accessing a Disk page

- Time to access (read/write) a disk block:
 - **seek time** (moving arms to position disk head on track)
 - ~2-3 ms on average
 - **rotational delay** (waiting for block to rotate under head)
 - ~0-4 ms (15000 RPM)
 - **transfer time** (actually moving data to/from disk surface)
 - ~0.25 ms per 64KB page
- Key to lower I/O cost: reduce seek/rotational delays

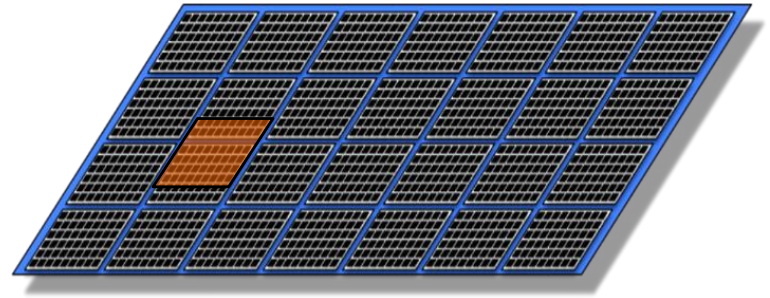
Notes on Flash (SSD)

- Issues in current generation (NAND)
 - Fine-grain reads (4-8K reads), coarse-grain writes (1-2 MB writes)
 - Only 2k-3k erasures before failure, so keep moving hot write units around (“wear leveling”)
 - Write amplification: big units, need to reorg for wear & garbage collection



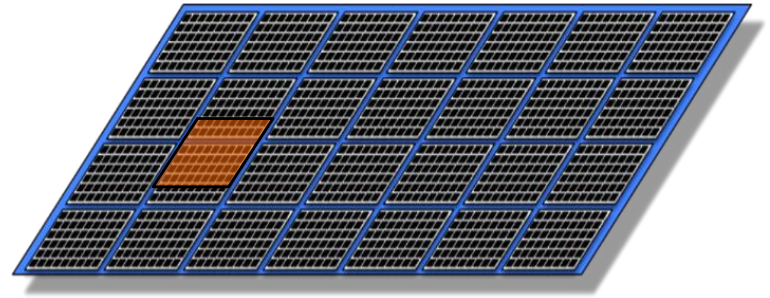
Notes on Flash (SSD), Pt. 2

- So... read is fast and predictable
 - Single read access time: 0.03 ms
 - 4KB random reads: ~500MB/sec
 - Sequential reads: ~525MB/sec
 - 64K: 0.48 ms

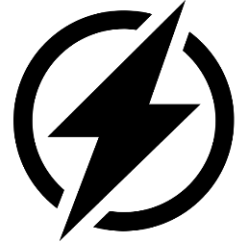


Notes on Flash (SSD), cont

- But... write is not! Slower for random
 - Single write access time: 0.03 ms
 - 4KB random writes: ~120 MB/sec
 - Sequential writes: ~480 MB/sec



Is Flash Faster than Disk?



Created by Dima Shio
from Noun Project

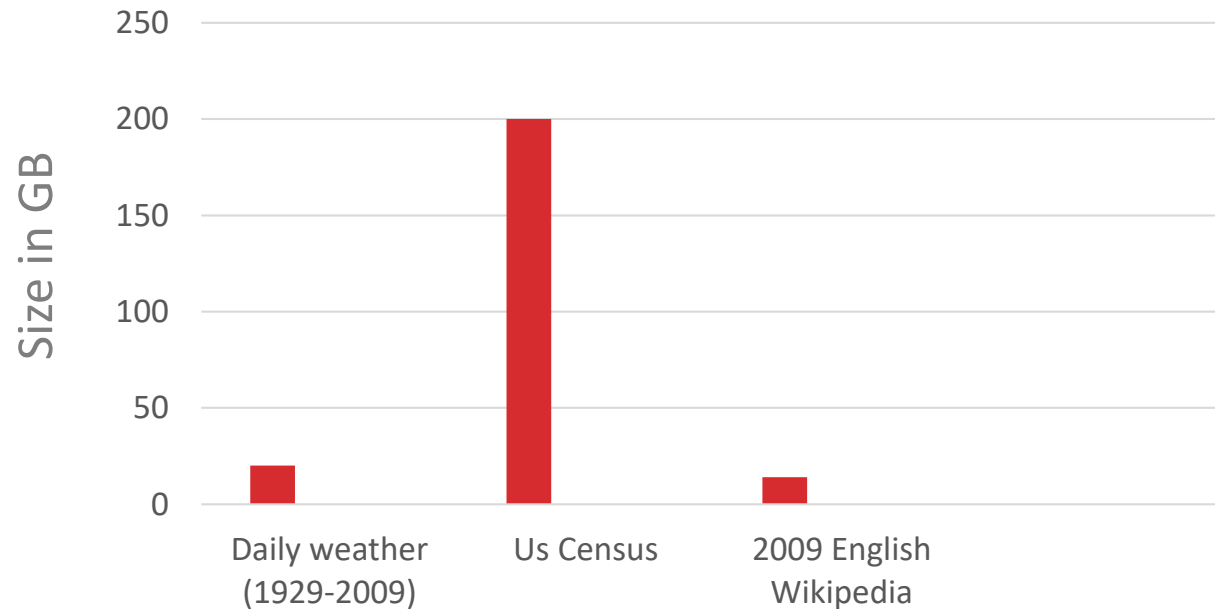
- Why of course it is...it's called "flash"!
 - Can be 1-10x the bandwidth (bytes/sec) of ideal HDD #s
 - Note: Ideal HDD #s hard to achieve.
 - Expect 10-100x bandwidth for non-sequential read.

Is Flash Faster Than Disk Pt 2.

- Locality” matters for both
 - Reading/writing to “far away” blocks on disk requires slow seek/rotation delay
 - Writing 2 “far away” blocks on SSD can require writing multiple much larger units
 - High-end flash drives are getting much better at this
- And don’t forget:
 - Disk offers about 10x the capacity per \$

Storage Pragmatics & Trends

- Many significant DBs are not big.



Storage Trends Pt. 2

- But data sizes grow faster than Moore's Law
 - “Big Data” is real
 - Boeing 787 generates ½ TB of data per flight
 - Walmart handles 1M transactions/hour,
 - maintains 2.5 PetaByte data warehouse



Created by Ralf Schmitzer
from Noun Project



Created by Ralf Schmitzer
from Noun Project

- So...what is the role of disk, flash, RAM
 - The subject of some debate!

[Airplane](#) [Cash Register](#)

Bottom Line (last few years)

- Very large DBs: relatively traditional
 - Disk still the best cost/MB by a lot
 - SSDs improve performance and performance variance
- Smaller DB story is changing quickly
 - Entry cost for disk is not cheap, so flash wins at the low end
 - Many interesting databases fit in RAM

Bottom Line Pt. 2

- Change brewing on the Hardware storage tech side
- Mixed answers on the Software/usage side
 - Big Data: Can generate and archive data cheaply and easily
 - Small Data: Many rich data sets have (small) fixed size
- People will continue to worry about magnetic disk for some time yet, typically at large scale

DISK SPACE MANAGEMENT

Disks and Files

- Recall, most DBMSs stores information on **Disks** and **SSDs**.
 - Disk are a mechanical anachronism (slow!)
 - SSDs faster, **slow relative to memory**, costly writes



Block Level Storage

- Read and Write **large chunks of sequential bytes**
- *Sequentially*: “Next” disk block is fastest
- Maximize usage of data per Read/Write
 - “Amortize” seek delays (HDDs) and writes (SSDs):
if you’re going all the way to Pluto, pack the spaceship full!
- Predict future behavior
 - Cache popular blocks
 - Pre-fetch likely-to-be-accessed blocks
 - Buffer writes to sequential blocks
 - More on these as we go

A Note on Terminology

64k usual

- **Block = Unit of transfer for disk read/write**
 - 64KB – 128KB is a good number today
 - Book says 4KB
- **Page: a common synonym for “block”**

same meaning

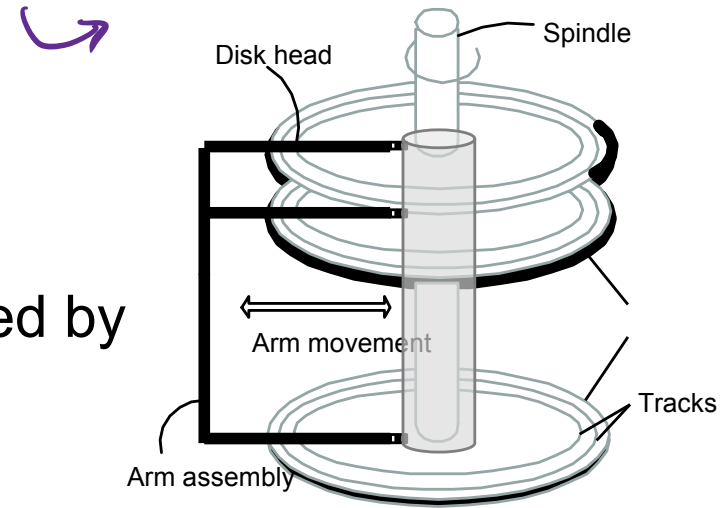
 - In some texts, “page” = a block-sized chunk of RAM
- We’ll treat “block” and “page” as synonyms

w

Arranging Blocks on Disk

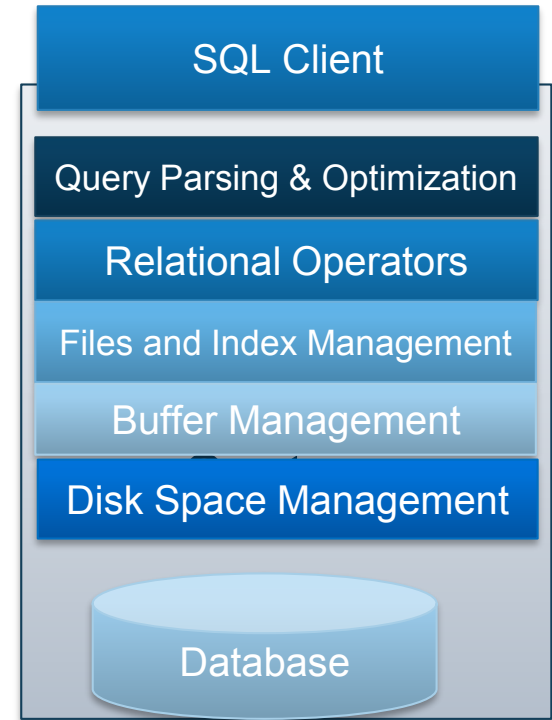
a - notion

- **'Next'** block concept:
 - sequential blocks on same track, followed by
 - blocks on same cylinder, followed by
 - blocks on adjacent cylinder
- Arrange file pages sequentially by 'next' on disk
 - minimize seek and rotational delay.
- For a **sequential scan**, *pre-fetch*
 - several blocks at a time!
- **Read large consecutive blocks**
scussive



Disk Space Management, cont

- **Lowest layer of DBMS, manages space on disk**
- **Purpose:** *memory* *disk*
 - Map pages to locations on disk
 - Load pages from disk to memory
 - Save pages back to disk & ensuring writes
- Higher levels call upon this layer to:
 - Read/write a page
 - Allocate/de-allocate logical pages



Disk Space Management: Requesting Pages

- Request for a *sequence* of pages best satisfied by pages stored sequentially on disk
 - Physical details hidden from higher levels of system
 - Higher levels may “safely” **assume** **Next Page** is fast, so they will simply expect sequential runs of pages to be quick to scan.

Disk Space Management: Implementation

low-level

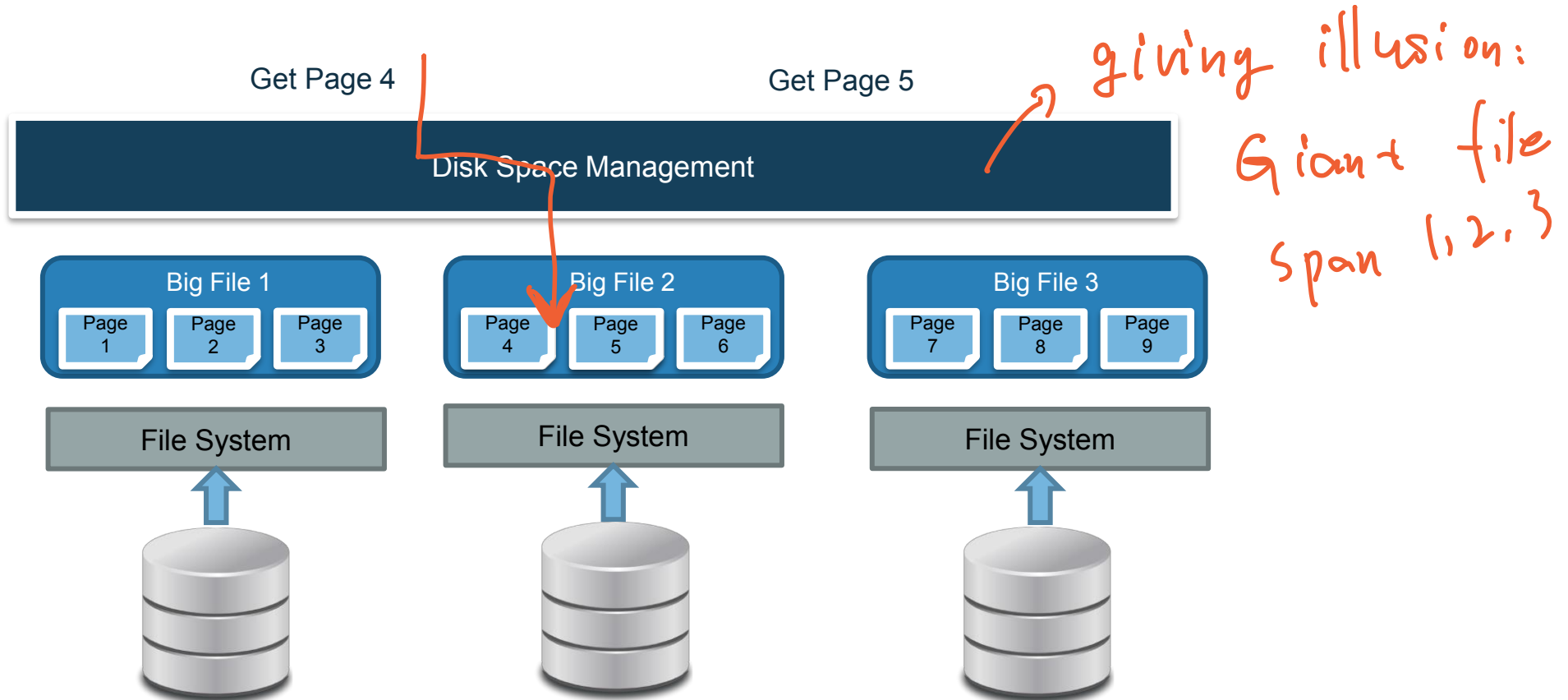
- **Proposal 1:** Talk to the storage device directly ✗
 - Could be very fast if you knew the device well
 - What happens when devices change?

Disk Space Management: Implementation 2

- **Proposal 2:** Run over filesystem (FS)
 - Allocate single large “contiguous” file on a nice empty disk, and assume sequential/nearby byte access are fast
 - Most FS optimize disk layout for sequential access
 - Gives us more or less what we want if we start with an empty disk
 - DBMS “file” may span multiple FS files on multiple disks/machines

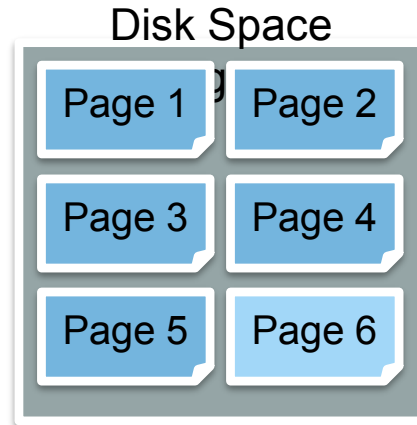
DBMS { FS 1
FS 2

Using Local Filesystem



Summary: Disk Space Management

- Provide API to read and write pages to device
- Pages: block level organization of bytes on disk
- Provides “next” locality and abstracts FS/device details



Disks and Files: Summary

- Magnetic (hard) disks and SSDs
 - Basic HDD mechanics
 - SSD write amplification
 - Concept of “near” pages and how it relates to cost of access
 - Relative cost of
 - Random vs. sequential disk access (10x)
 - Disk (pluto) vs RAM (sacramento) vs. registers (your head)
 - Big, big differences!

Files: Summary Pt 2

- DB File storage
 - Typically over FS file(s)
- Disk space manager loads and stores pages
 - Block level reasoning
 - Abstracts device and file system; provides fast “next”