

DSC 102

Systems for Scalable Analytics

Haojian Jin

Topic 1: Basics of Machine Resources
Part 2: Operating Systems

Ch. 2, 4.1-4.2, 6, 7, 13, 14.1, 18.1, 21, 22, 26, 36, 37, 39, and
40.1-40.2 of Comet Book

Outline

- ❖ Basics of Computer Organization
 - ❖ Digital Representation of Data
 - ❖ Processors and Memory Hierarchy
- ❖ Basics of Operating Systems (OS)
 - ❖ Process Management: Virtualization; Concurrency
 - ❖ Filesystem and Data Files
 - ❖ Main Memory Management
- ❖ Persistent Data Storage

App Stores

*"changed how software development worked, and expanded the **number of people who could comfortably, safely use a computer** from a few hundred million to a few billion."*

[1] Benedict Evans, App stores, trust and anti-trust

Putting apps in a sandbox

Apps can only do things that Apple allows and cannot ask (or persuade, or trick) the user for permission to do 'dangerous' things.



- Would this break my phone?
- Would this run my battery down?
- Steal my bank details?

Q: What is an OS? Why do we need it?



legislature



judiciary

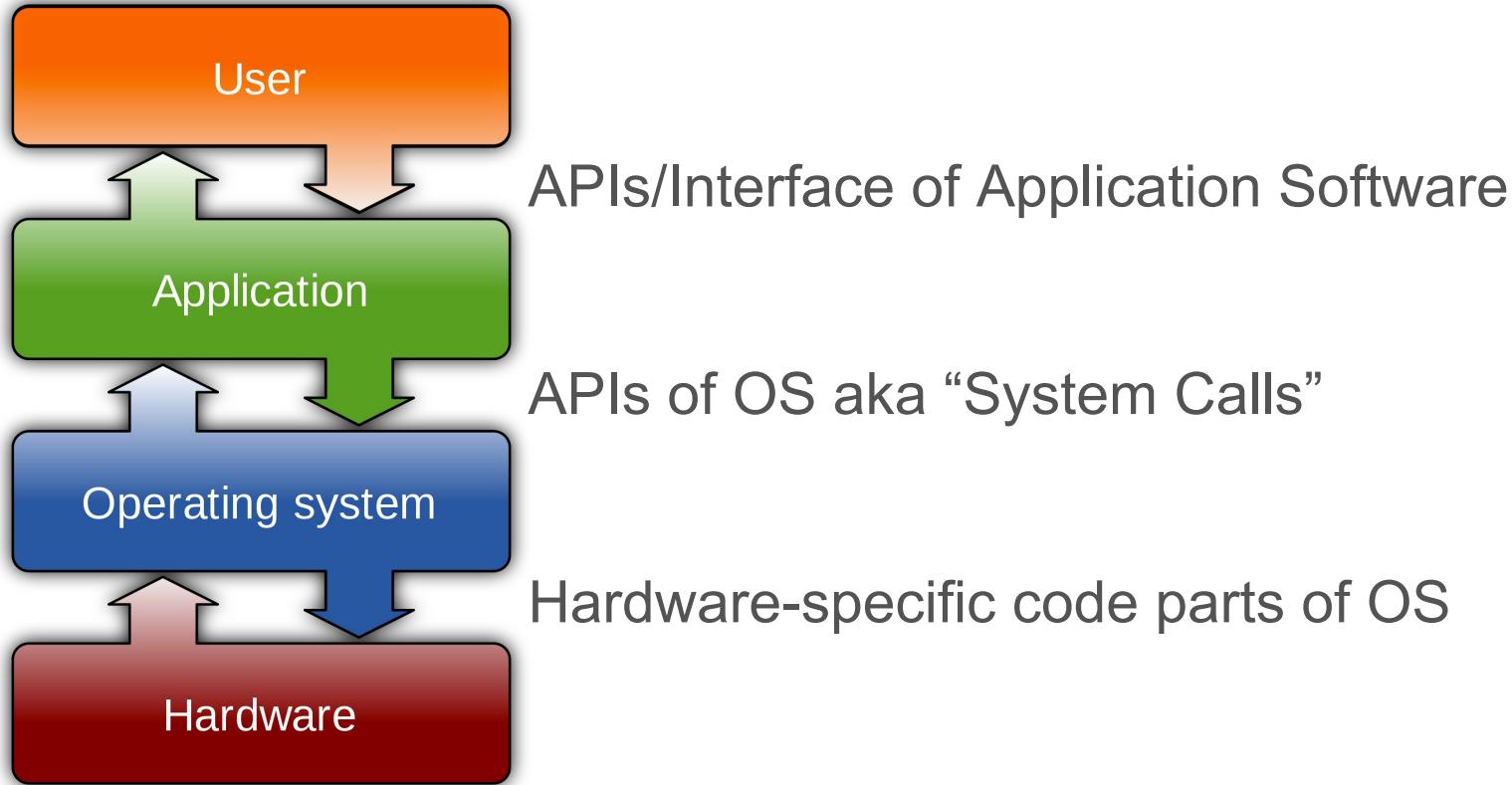


executive

Role of an OS in a Computer

- ❖ An OS is a large set of interrelated programs that *make it easier* for applications and user-written programs to use computer hardware *effectively, efficiently, and securely*
 - ❖ Akin to a government's role in a country
- ❖ Without OS, computer users must speak machine code!
- ❖ 2 key principles in OS (any system) design & impl.:
 - ❖ **Modularity:** Divide system into *functionally cohesive components* that each do their jobs well
 - ❖ Akin to executive-legislature-judiciary split
 - ❖ **Abstraction:** *Layers of functionalities* from low-level (close to hardware) to high level (close to user)
 - ❖ Akin to local-city-county-state-federal levels?

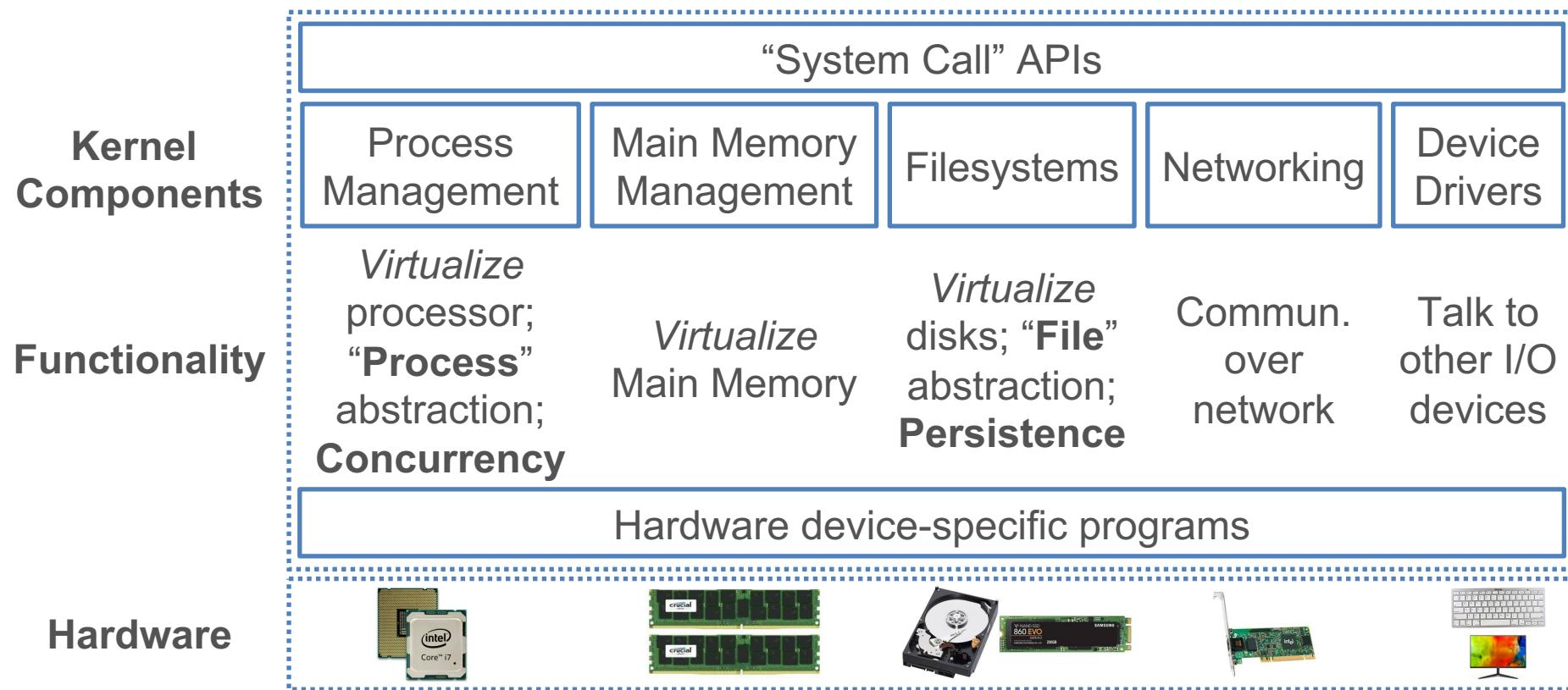
Role of an OS in a Computer



“Application Software” notion is now more complex due to multiple tiers of abstraction; “Platform Software” or “Software Framework” is a new tier between “Application” and OS

Key Components of OS

- ❖ **Kernel:** The core of an OS with modules to abstract the hardware and APIs for programs to use
- ❖ Auxiliary parts of OS include shell/terminal, file browser for usability, extra programs installed by I/O devices, etc.



Outline

- ❖ Basics of Computer Organization
 - ❖ Digital Representation of Data
 - ❖ Processors and Memory Hierarchies
 - ❖ Basics of Operating Systems (OS)
 - ❖ Process Management: Virtualization; Concurrency
 - ❖ Filesystem and Data Files
 - ❖ Main Memory Management
 - ❖ Persistent Data Storage
- You will face myriad
and new data types
- Compute hardware
is evolving fast
- You will need to use new
methods on evolving data file
formats on clusters / cloud
- Storage hardware
are evolving fast

The Abstraction of a Process

- ❖ **Process:** A *running* program, the central abstraction in OS
 - ❖ Started by OS when a program is executed by user
 - ❖ OS keeps inventory of “alive” processes (**Process List**) and handles apportioning of hardware among processes

Q: *Why bother knowing process management in Data Science?*

- ❖ A *query* is a program that becomes a process
- ❖ A data system typically *abstracts* away process management because user specifies the queries / processes in system’s API



- ❖ But in the cloud era, things are up in the air! Will help to know a bit of how they handle data-intensive computations under the hood

Logistics

- ❖ **Midterm: Nov. 1st, Wednesday (Tentative).**
- ❖ **Assignment due: 10/22/2023 23:59.**
- ❖ **No late days for submitting the PAs.**

The Abstraction of a Process

- ❖ High-level steps OS takes to get a process going:
 1. **Create** a process (get Process ID; add to Process List)
 2. Assign part of DRAM to process, aka its **Address Space**
 3. Load code and static data (if applicable) to that space
 4. Set up the inputs needed to run program's *main()*
 5. Update process' **State** to *Ready*
 6. When process is **scheduled** (*Running*), OS temporarily hands off control to process to run the show!
 7. Eventually, process finishes or run **Destroy**

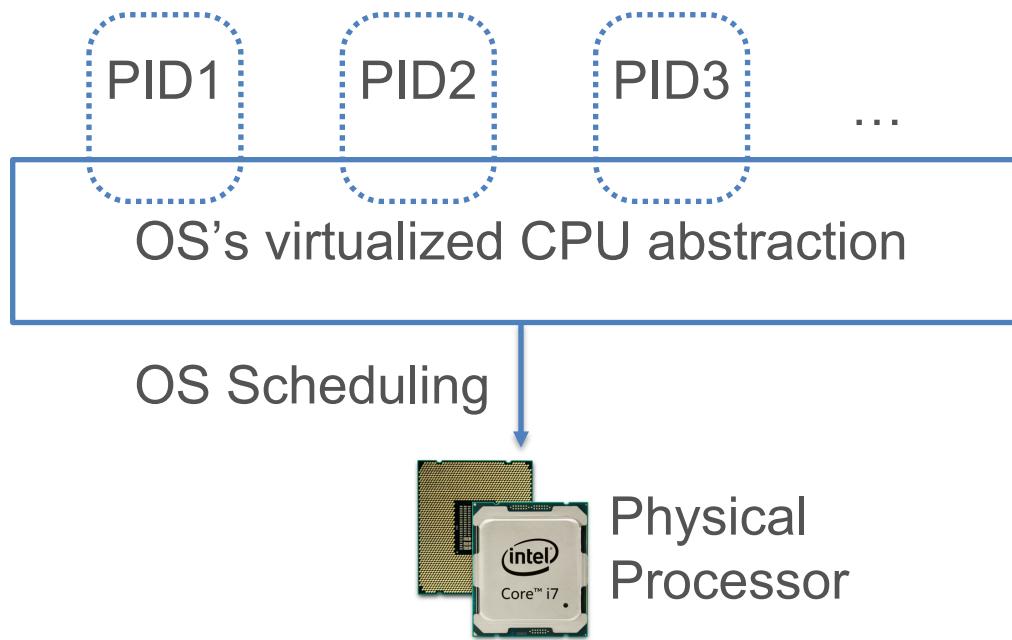
Virtualization of Hardware Resources

Q: *But is it not risky/foolish for OS to hand off control of hardware to a process (random user-written program)??!*

- ❖ OS has *mechanisms* and *policies* to regain control
- ❖ **Virtualization:**
 - ❖ Each hardware resource is treated as a virtual entity that OS can divvy up among processes in a controlled way
- ❖ **Limited Direct Execution:**
 - ❖ OS mechanism to time-share CPU and preempt a process to run a different one, aka “context switch”
 - ❖ **A Scheduling policy** tells OS what time-sharing to use
 - ❖ Processes also must transfer control to OS for “privileged” operations (e.g., I/O); **System Calls API**

Virtualization of Processors

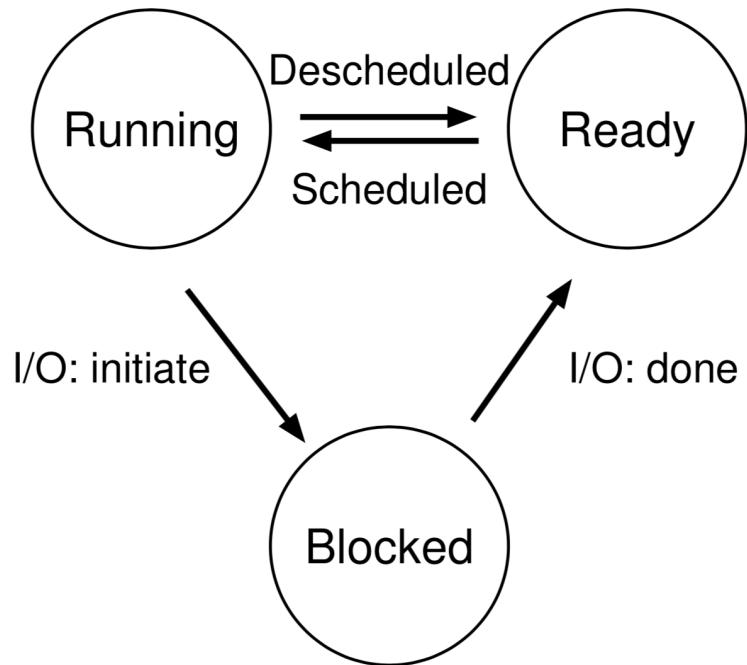
- ❖ Virtualization of processor enables process **isolation**, i.e., each process given an “illusion” that it alone runs



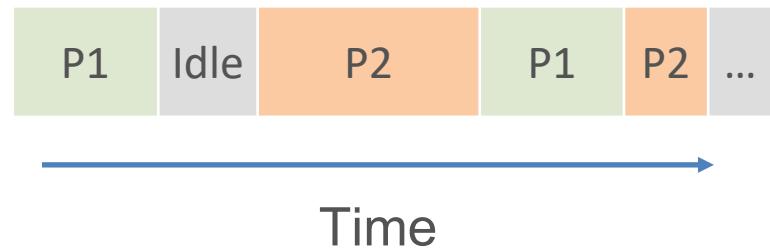
- ❖ Inter-process communication possible in System Calls API
- ❖ Later: Generalize to **Thread** abstraction for **concurrency**

Process Management by OS

- ❖ OS keeps moving processes between 3 states:



- ❖ Gantt Chart: A viz. to show what process runs when (on processor)



- ❖ Sometimes, if a process gets “stuck” and OS did not schedule something else, system **hangs**; need to reboot!

Scheduling Policies/Algorithms

- ❖ **Schedule:** Record of what process runs on each CPU when
- ❖ Policy controls how OS time-shares CPUs among processes
- ❖ Key terms for a process (aka **job**):
 - ❖ **Arrival Time:** Time when process gets created
 - ❖ **Job Length:** Duration of time needed for process
 - ❖ **Start Time:** Times when process first starts on processor
 - ❖ **Completion Time:** Time when process finishes/killed
 - ❖ **Response Time** = Start Time — Arrival Time
 - ❖ **Turnaround Time** = Completion Time — Arrival Time
- ❖ **Workload:** Set of processes, arrival times, and job lengths that OS Scheduler has to handle

APART





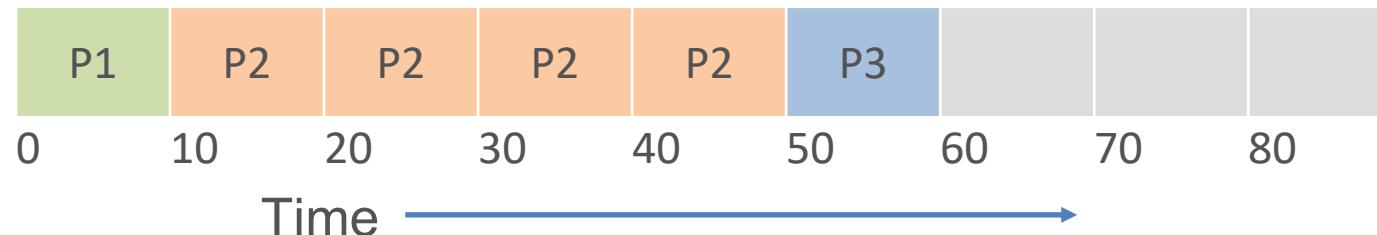
Scheduling Policies/Algorithms

- ❖ In general, not all Arrival Times and Job Lengths will be known beforehand. But **preemption** is possible.
- ❖ **Key Principle:** Inherent tension in scheduling between overall workload *performance* and allocation *fairness*
 - ❖ Performance metric is usually *Average Turnaround Time*
 - ❖ Many fairness metrics exist, e.g., Jain's fairness index
- ❖ 100s of scheduling policies studied! Well-known ones: FIFO, SJF, STCF, Round Robin, Random, etc.
 - ❖ Different criteria for ranking; preemptive vs not
 - ❖ Complex “multi-level feedback queue” schedulers
 - ❖ ML-based schedulers are “hot” nowadays!

Scheduling Policy: FIFO

- ❖ First-In-First-Out aka First-Come-First-Serve (FCFS)
- ❖ Ranking criterion: Arrival Time; no preemption allowed

Example: P1, P2, P3 of lengths 10,40,10 units arrive closely in that order



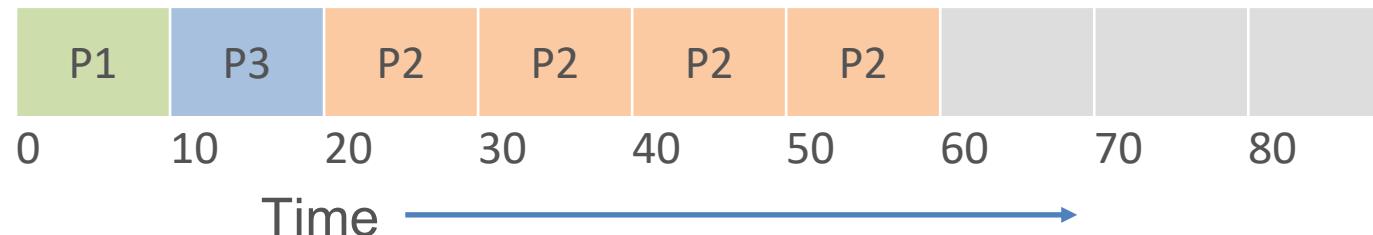
Process	Arrival Time	Start Time	Completion Time	Response Time	Turnaround Time
P1	0	0	10	0	10
P2	0	10	50	10	50
P3	0	50	60	50	60
Avg:			20	40	

- ❖ Main con: Short jobs may wait a lot, aka “Convoy Effect”

Scheduling Policy: SJF

- ❖ Shortest Job First
- ❖ Ranking criterion: Job Length; no preemption allowed

Example: P1, P2, P3 of lengths 10,40,10 units arrive closely in that order



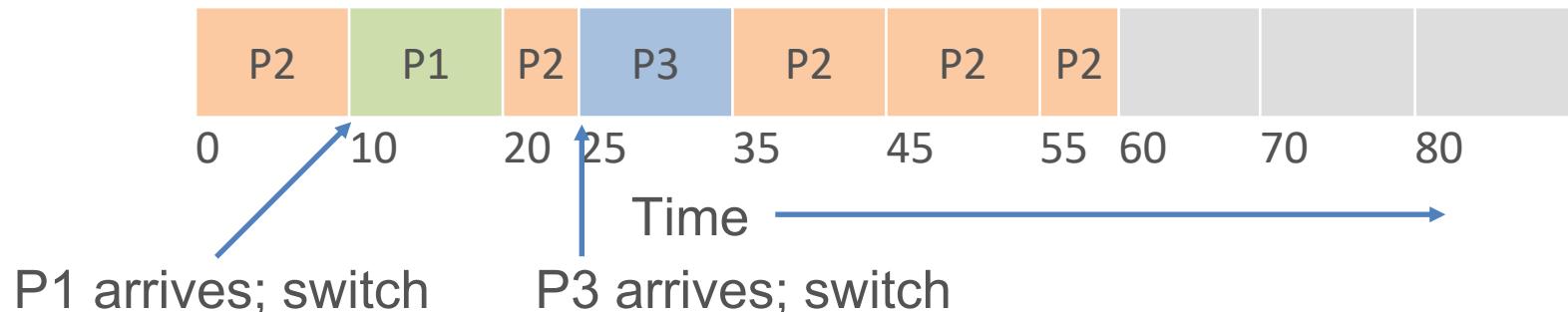
Process	Arrival Time	Start Time	Completion Time	Response Time	Turnaround Time
P1	0	0	10	0	10
P2	0	20	60	20	60
P3	0	10	20	10	20
(FIFO Avg: 20 and 40)			Avg:	10	30

- ❖ Main con: Not all Job Lengths might be known beforehand

Scheduling Policy: SCTF

- ❖ Shortest Completion Time First
- ❖ Jobs might not all arrive at same time; preemption possible

Example: P1, P2, P3 of lengths 10,40,10 units arrive at different times



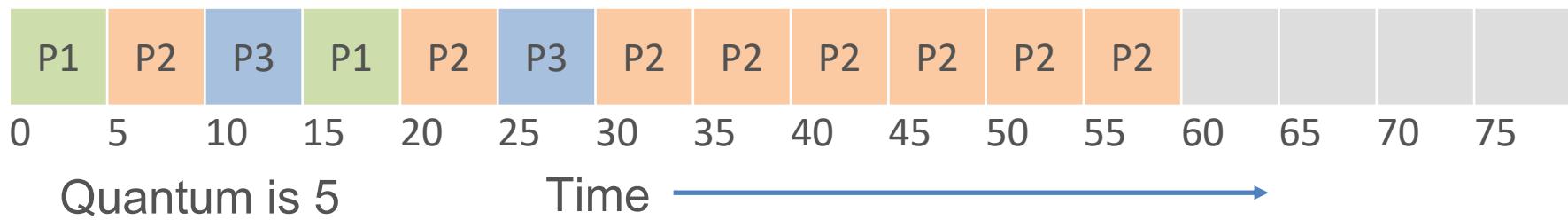
Process	Arrival Time	Start Time	Completion Time	Response Time	Turnaround Time
P1	10	10	20	0	10
P2	0	0	60	0	60
P3	25	25	35	0	10
(SJF Avg: 10 and 30)			Avg:	0	26.7

- ❖ Main con same as SJF; Job Lengths might not be known

Scheduling Policy: Round Robin

- ❖ RR does not need to know job lengths
- ❖ Fixed time *quantum* given to each job; cycle through jobs

Example: P1, P2, P3 of lengths 10,40,10 units arrive closely in that order



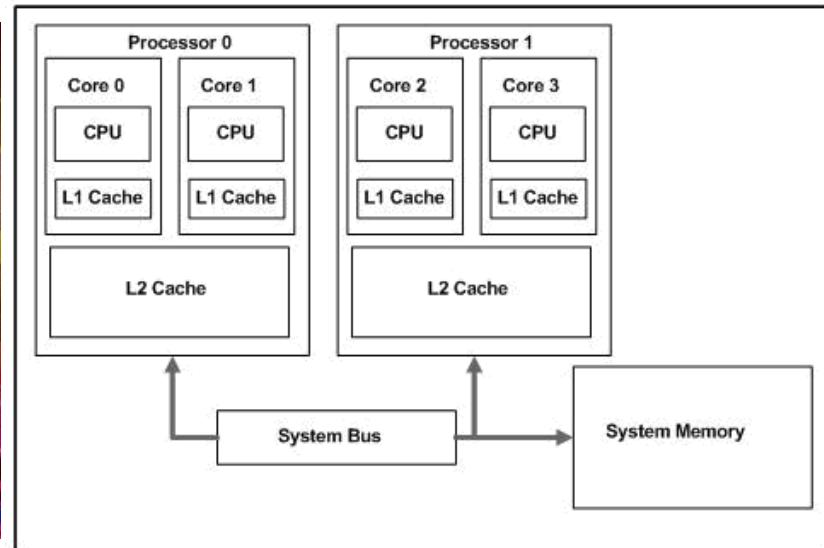
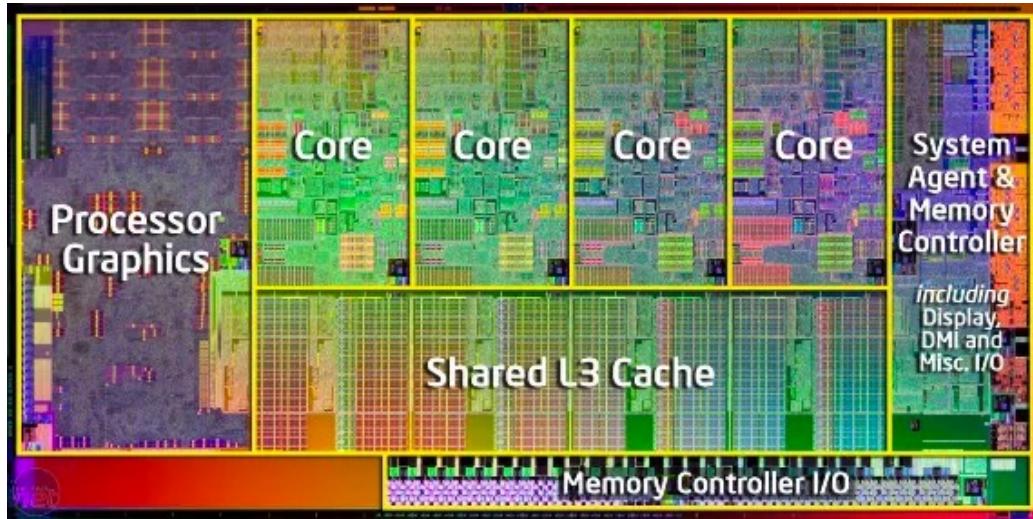
Process	Arrival Time	Start Time	Completion Time	Response Time	Turnaround Time
P1	0	0	20	0	20
P2	0	5	60	5	60
P3	0	10	30	10	30

(SJF Avg: 10 & 30; SCTF Avg: 0 & 26.7) Avg: 5 36.7

- ❖ RR is often very fair, but Avg Turnaround Time goes up!

Concurrency

- ❖ Modern computers often have multiple processors and multiple cores per processor
- ❖ **Concurrency:** Multiple processors/cores run different/same set of instructions simultaneously on different/shared data
- ❖ New levels of shared caches are added



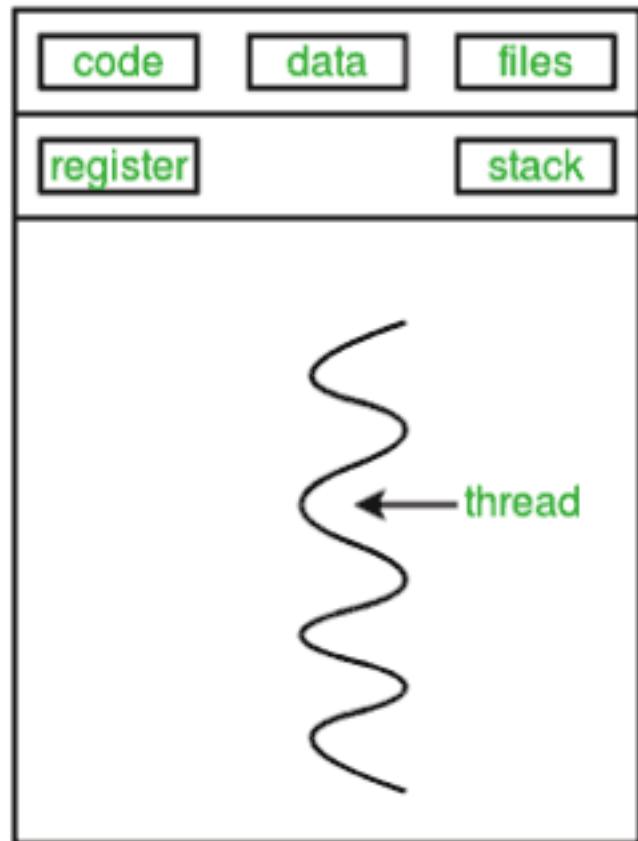
Concurrency

- ❖ **Multiprocessing:** Different processes run on different cores (or entire CPUs) simultaneously
- ❖ **Thread:** Generalization of OS's Process abstraction
 - ❖ A program *spawns* many threads; each run parts of the program's computations simultaneously
 - ❖ **Multithreading:** Same core used by many threads

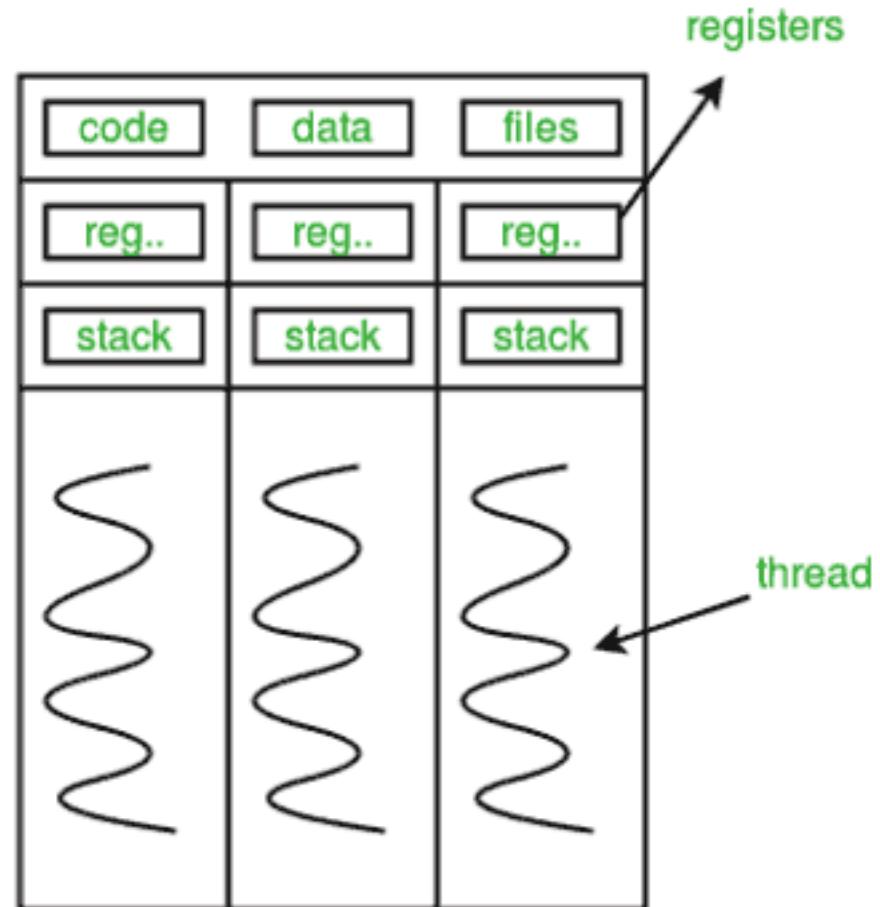


- ❖ Issues in dealing with multithreaded programs that *write shared data*:
 - ❖ Cache coherence
 - ❖ Locking; deadlocks
 - ❖ Complex scheduling

Thread vs process



single-threaded process



multithreaded process

Example thread code

```
# Python program to illustrate the concept
# of threading
# importing the threading module
import threading

def print_cube(num):
    # function to print cube of given num
    print("Cube: {}" .format(num * num * num))

def print_square(num):
    # function to print square of given num
    print("Square: {}" .format(num * num))

if __name__ == "__main__":
    # creating thread
    t1 = threading.Thread(target=print_square, args=(10,))
    t2 = threading.Thread(target=print_cube, args=(10,))

    # starting thread 1
    t1.start()
    # starting thread 2
    t2.start()

    # wait until thread 1 is completely executed
    t1.join()
    # wait until thread 2 is completely executed
    t2.join()

    # both threads completely executed
    print("Done!")
```

Inter-thread communication

```
from queue import Queue
from threading import Thread

# A thread that produces data
def producer(out_q):
    while True:
        # Produce some data
        ...
        out_q.put(data)

# A thread that consumes data
def consumer(in_q):
    while True:
        # Get some data
        data = in_q.get()
        # Process the data
        ...

# Create the shared queue and launch both threads
q = Queue()
t1 = Thread(target = consumer, args =(q, ))
t2 = Thread(target = producer, args =(q, ))
t1.start()
t2.start()
```

Inter-process communication

Your server could listen to receive python objects:

```
from multiprocessing.connection import Listener

address = ('localhost', 6000)      # family is deduced to be 'AF_INET'
listener = Listener(address, authkey=b'secret password')
conn = listener.accept()
print 'connection accepted from', listener.last_accepted
while True:
    msg = conn.recv()
    # do something with msg
    if msg == 'close':
        conn.close()
        break
listener.close()
```

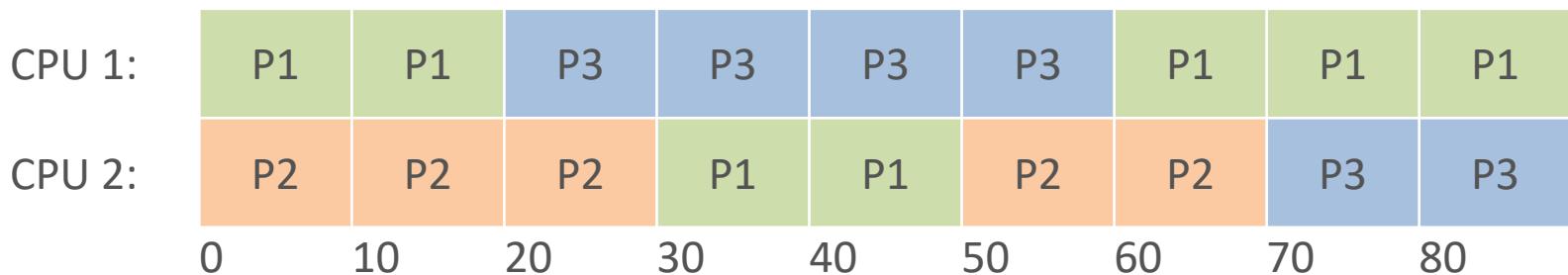
Your client could send commands as objects:

```
from multiprocessing.connection import Client

address = ('localhost', 6000)
conn = Client(address, authkey=b'secret password')
conn.send('close')
# can also send arbitrary objects:
# conn.send(['a', 2.5, None, int, sum])
conn.close()
```

Concurrency

- ❖ Scheduling for multiprocessing/multicore is more complex
- ❖ **Load Balancing:** Ensuring different cores/proc. are kept roughly equally busy, i.e., reduce **idle times**
- ❖ Multi-queue multiprocessor scheduling (MQMS) is common
 - ❖ Each proc./core has its own job queue
 - ❖ OS moves jobs across queues based on load
 - ❖ Example Gantt chart for MQMS:



Concurrency in Data Science

- ❖ Thankfully, most data-intensive computations in data science do not need concurrent writes on shared data!
 - ❖ Concurrent low-level ops abstracted away by libraries/APIs
 - ❖ **Partitioning / replication** of data simplifies concurrency
- ❖ Later topic (Parallelism Paradigms) will cover parallelism in depth:
 - ❖ Multi-core, multi-node, etc.
 - ❖ Task parallelism, Partitioned data parallelism, etc.

Review Questions

- ❖ If you can afford infinite DRAM, is there any reason not to use it?
- ❖ What is the purpose of an OS?
- ❖ Why is the design of an OS so modular?
- ❖ Why does an OS need to use a scheduling policy?
- ❖ Which quantity captures latency of a process starting: Response Time or Turnaround Time?
- ❖ What gives rise to different scheduling policies?
- ❖ Which scheduling policy is the fairest among the ones we covered?
- ❖ What is the Convoy Effect? Which sched. policy has that issue?
- ❖ Explain one pro and one con of Round Robin over SJF.

Peer Instruction Activity

(Switch slides)

Logistics

- ❖ PA0 assignments (45 teams).
- ❖ 2 teams have not made any submissions. Both are single person teams. One hasn't opened AWS. One started working on AWS on 22 Oct.
- ❖ Course content & assignments.

Outline

- ❖ Basics of Computer Organization
 - ❖ Digital Representation of Data
 - ❖ Processors and Memory Hierarchy
- ❖ Basics of Operating Systems (OS)
 - ❖ Process Management: Virtualization; Concurrency
-  ❖ Filesystem and Data Files
 - ❖ Main Memory Management
- ❖ Persistent Data Storage

Q: What is a file?



INVESTMENTS

Abstractions: File and Directory

- ❖ **File:** A persistent sequence of bytes that stores a logically coherent digital object for an application
 - ❖ **File Format:** An application-specific standard that dictates how to interpret and process a file's bytes
 - ❖ 100s of file formats exist (e.g., TXT, DOC, GIF, MPEG); varying data models/types, domain-specific, etc.
 - ❖ **Metadata:** Summary or organizing info. about file content (aka *payload*) stored with file itself; format-dependent
- ❖ **Directory:** A cataloging structure with a list of references to files and/or (recursively) other directories
 - ❖ Typically treated as a special kind of file
 - ❖ Sub dir., Parent dir., Root dir.

Filesystem

- ❖ **Filesystem:** The part of OS that helps programs create, manage, and delete files on disk (sec. storage)
- ❖ Roughly split into *logical level* and *physical level*
 - ❖ Logical level exposes file and dir. abstractions and offers System Call APIs for file handling
 - ❖ Physical level works with disk firmware and moves bytes to/from disk to DRAM

Filesystem

- ❖ Dozens of filesystems exist, e.g., ext2, ext3, NTFS, etc.
 - ❖ Differ on how they layer file and dir. abstractions as bytes, what metadata is stored, etc.
 - ❖ Differ on how data integrity/reliability is assured, support for editing/resizing, compression/encryption, etc.
 - ❖ Some can work with (“**mounted**” by) multiple OSs

Filesystem

Question Copy/Paste speed difference: macOS vs. Windows PC

Endre · May 8, 2019 · Apple windows

Forums > Software > macOS

Not open for further replies.



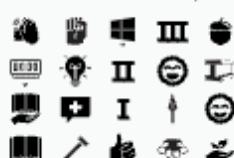
Endre
Reputable

Apr 30, 2019

1,102

214

5,840



May 8, 2019

#1

A friend of mine owns a macOS laptop (CPU: Intel i5, 16GB DDR4, OSX on SSD). He copied a 33GB file from his internal drive to the desktop in about 1 second!

I own a Desktop PC (CPU: Intel i7-9700K, 16GB DDR4, Windows 10 x64 on SATA-SSD). I copied a 16GB file from Local Disk C to the Desktop in... A LOT LONGER TIME THAN 1 second!

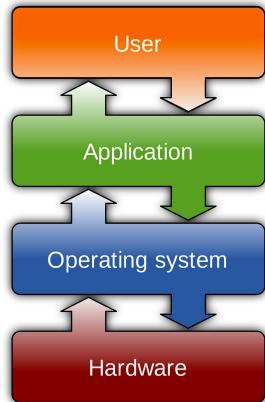
QUESTION#1: Why is Windows copying files so slow?

QUESTION#2: Would an M.2 NVMe SSD or an Intel Optane Memory match the speed of the macOS?

Virtualization of File on Disk

- ❖ OS abstracts a file on disk as a virtual object for processes
- ❖ **File Descriptor:** An OS-assigned integer identifier/reference for a file's virtual object that a process can use
 - ❖ 0/1/2 reserved for STDIN/STDOUT/STDERR
 - ❖ **File Handle:** A PL's abstraction on top of a file descr. (fd)

System Call API for File Handling:



API of OS called “System Calls”

- ❖ **open()**: Create a file; assign fd; optionally overwrite
- ❖ **read()**: Copy file's bytes on disk to in-mem. buffer; sized
- ❖ **write()**: Copy bytes from in-mem. buffer to file on disk
- ❖ **fsync()**: “Flush” (force write) “dirty” data to disk
- ❖ **close()**: Free up the fd and other OS state info on it
- ❖ **lseek()**: Position offset in file's fd (for random R/W later)
- ❖ Dozens more (rename, mkdir, chmod, etc.)

Q: What is a database? How is it different from just a bunch of files?

Collection of files?

Virtualization of Files

**Binary Representation on
Disk storage**

- Maintenance
- Performance
- Usability
- Security & privacy
- ...

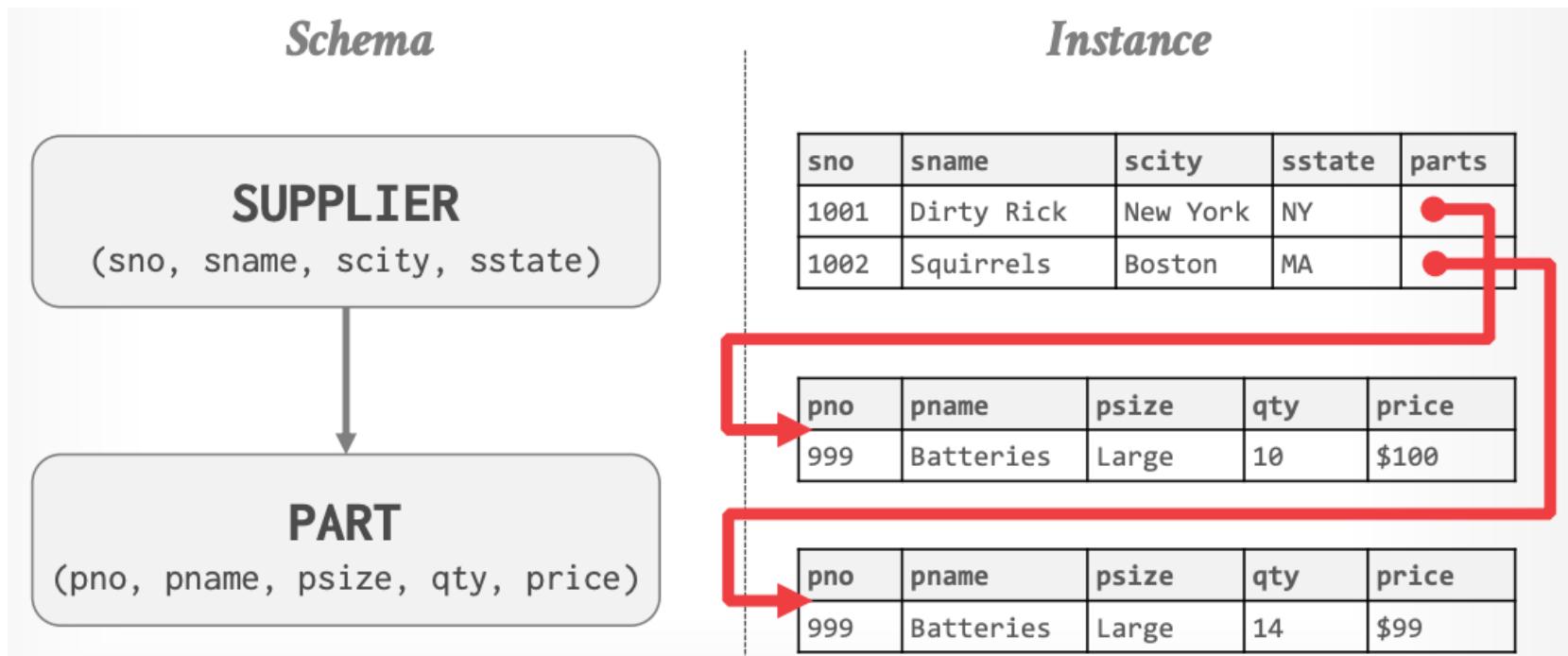
Files Vs Databases: Data Model

- ❖ **Database:** An *organized* collection of interrelated data
 - ❖ **Data Model:** An abstract model to define organization of data in a formal (mathematically precise) way
 - ❖ E.g., Relations, XML, Matrices, DataFrames
- ❖ Every database is just an *abstraction* on top of data files!
 - ❖ **Logical level:** Data model for higher-level reasoning
 - ❖ **Physical level:** How bytes are layered on top of files
 - ❖ All data systems (RDBMSs, Dask, Spark, TensorFlow, etc.) are application/platform software that use OS System Call API for handling data files

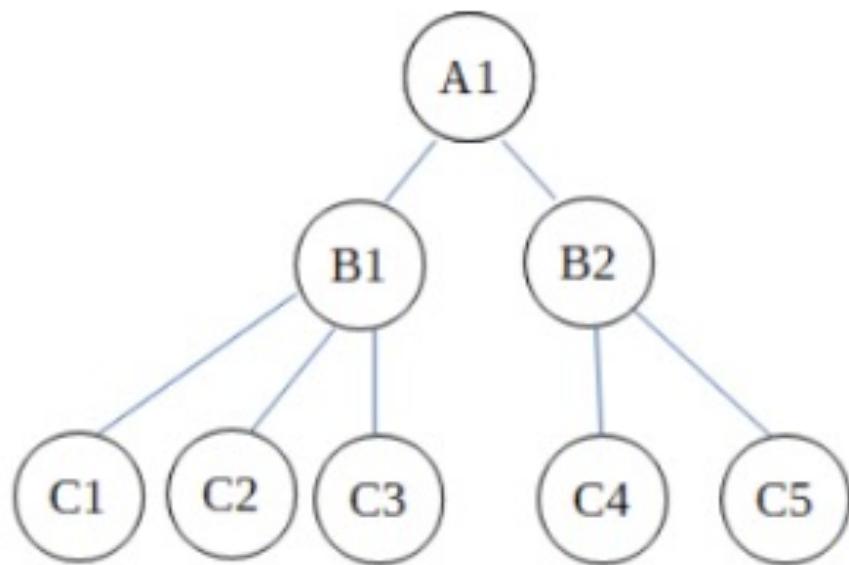
Principles and classical ideas (will discuss more later)

- Old ideas are still relevant today.
 - Most of the ideas are not new.
- People re-invent tools based on the classical ideas every decade.
 - It then fails and the later tools absorbs the key ideas.
- Tools change constantly, since we have new tasks and new hardware.

Hierarchical Data Model

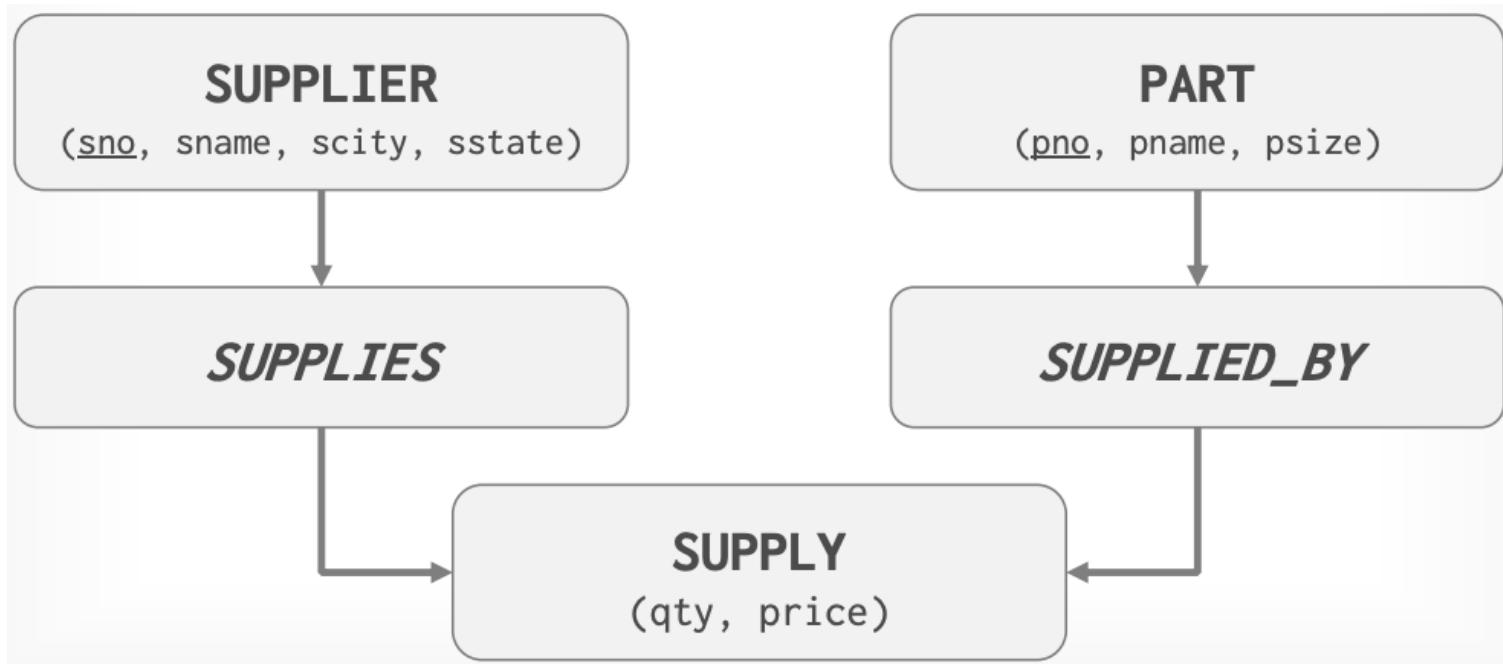


Hierarchical Data Model

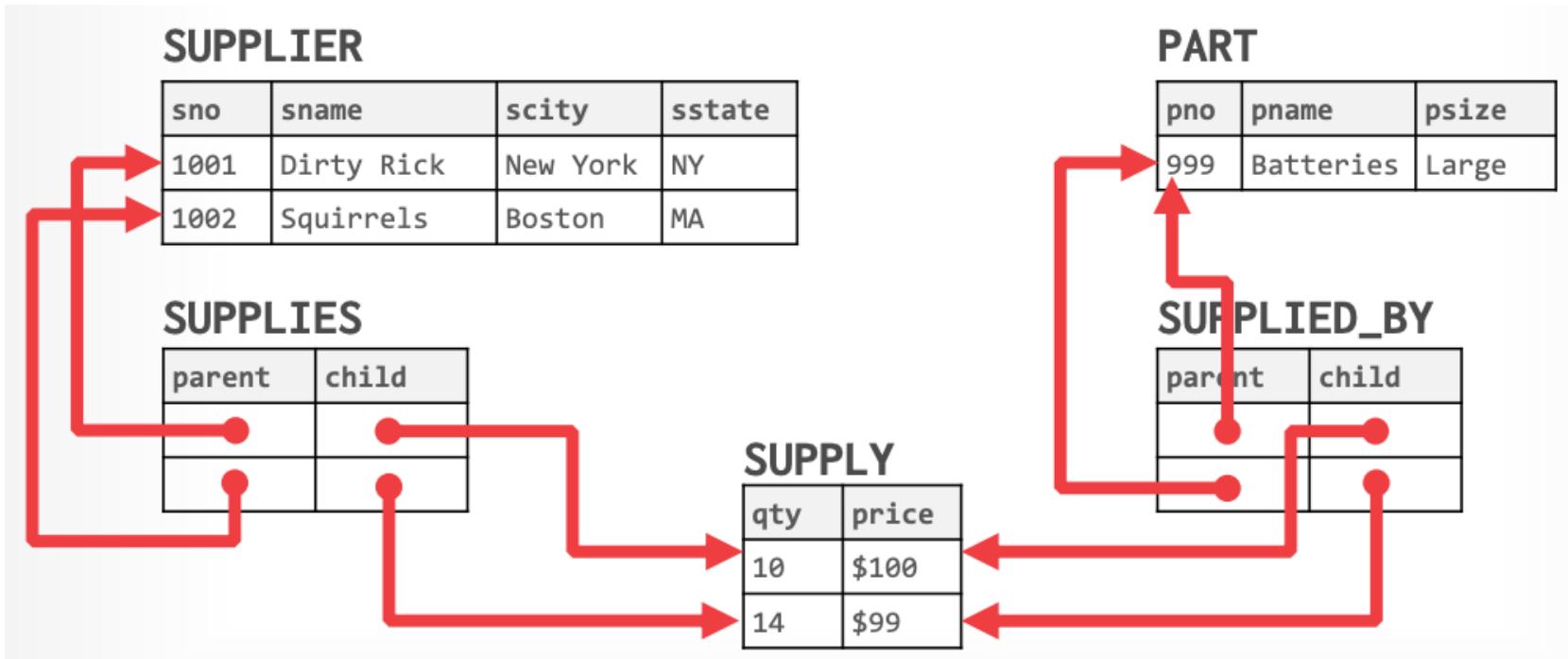


- Initially file-based model.
- Good for one-to-many relationships.
- Bad for many-to-many relationships.

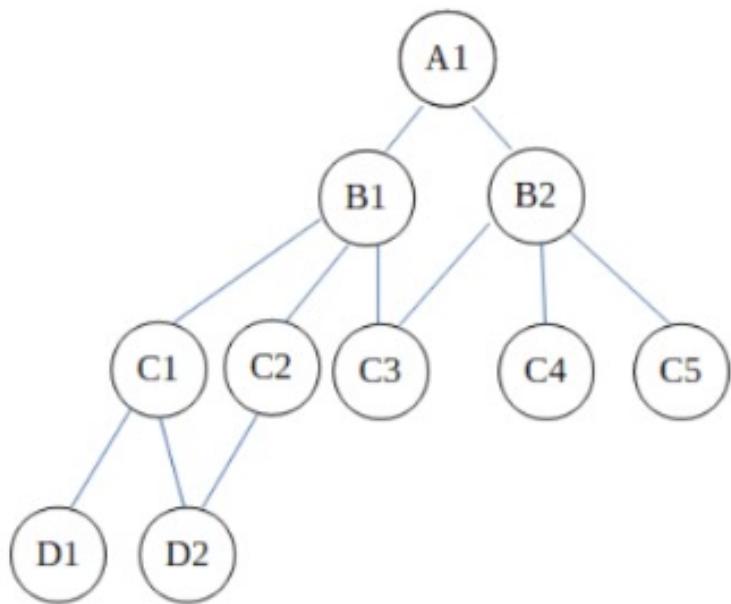
Network Data Model (schema)



Network Data Model (Instance)

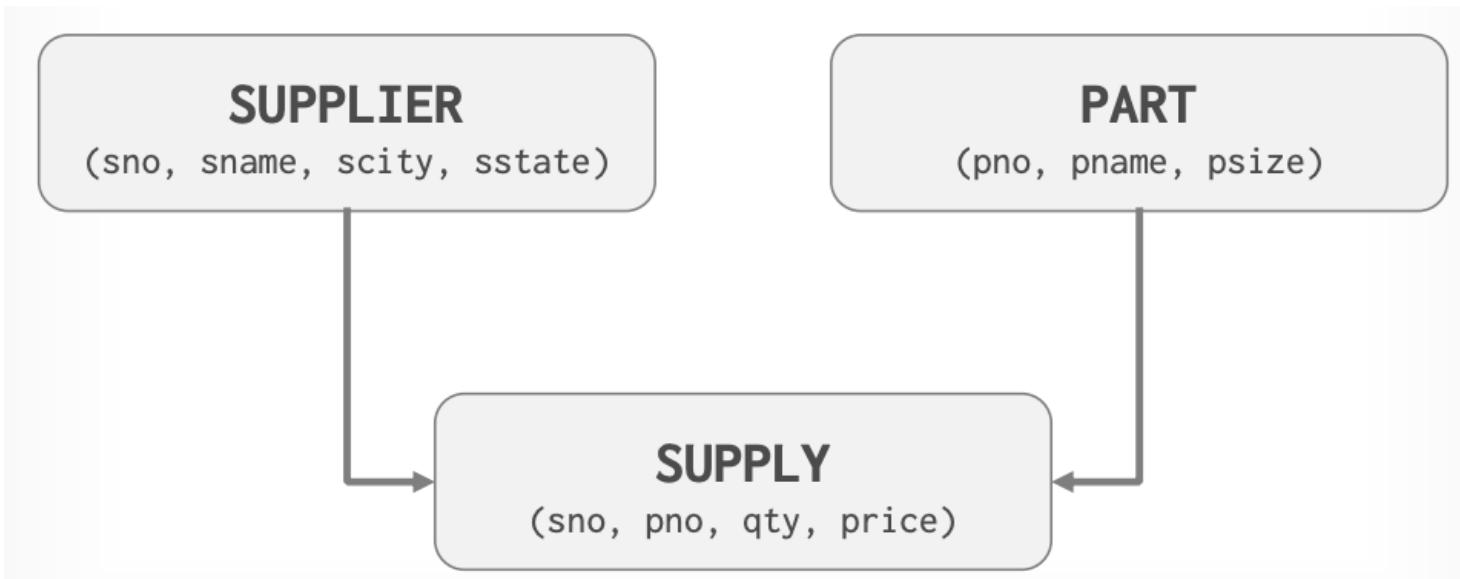


Network Data Model

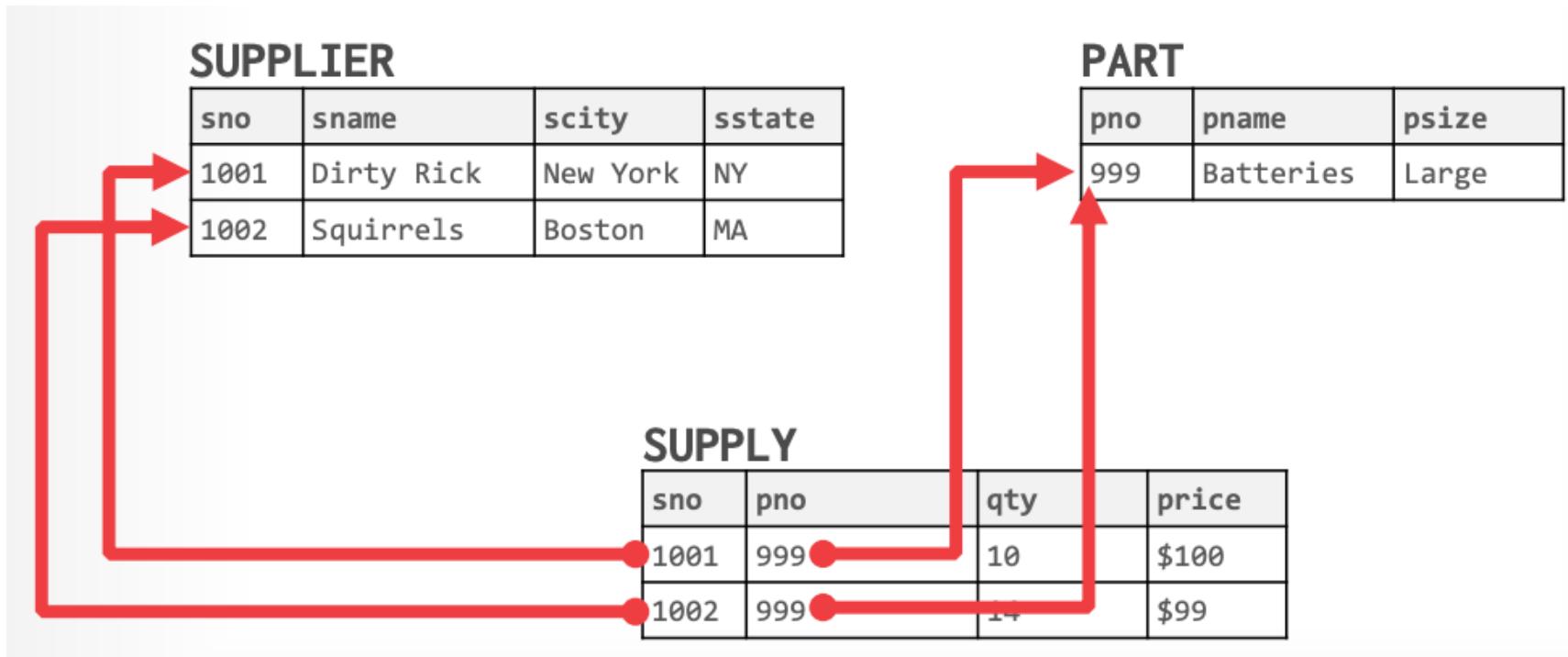


- Support many-to-many relationships.
- Bad for many-to-many relationships.

Relational Data Model (Schema)



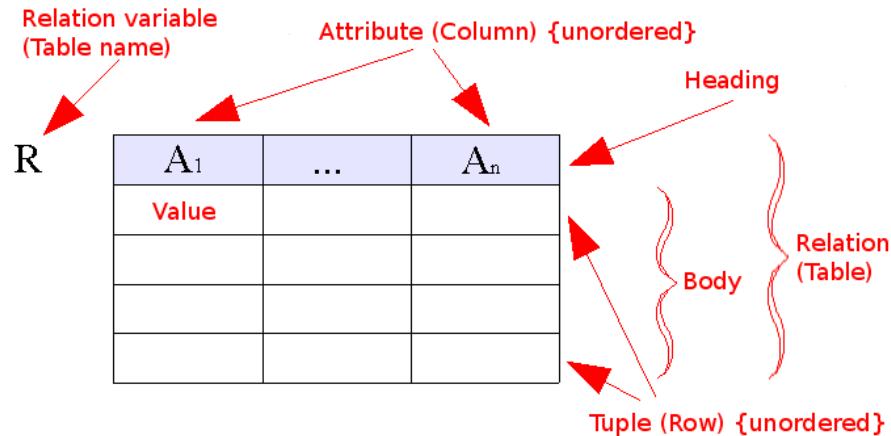
Relational Data Model (Instance)



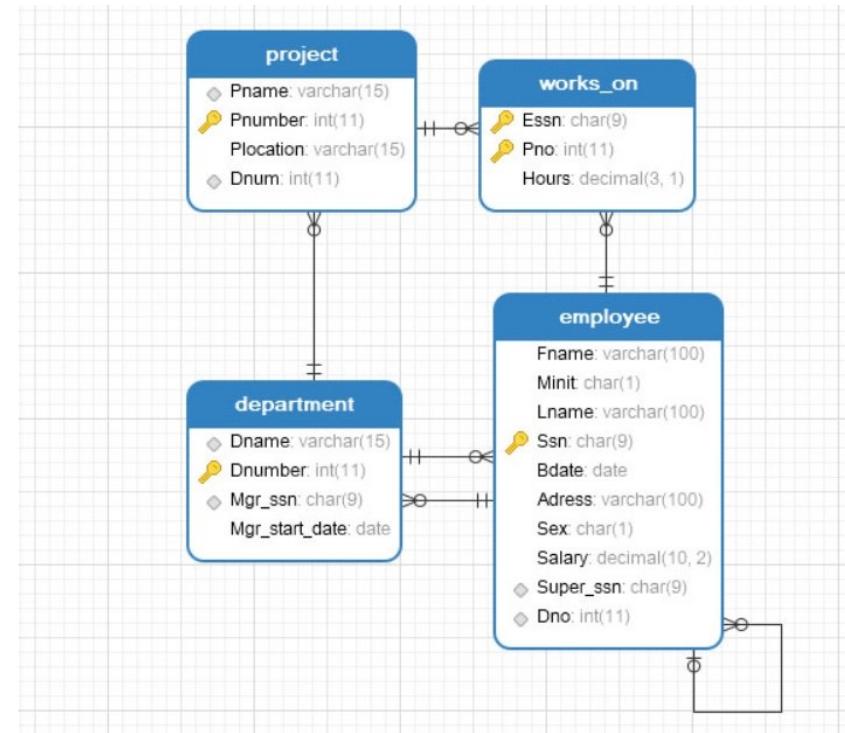
Data as File: Structured

- ❖ **Structured Data:** A form of data with regular substructure

Relation



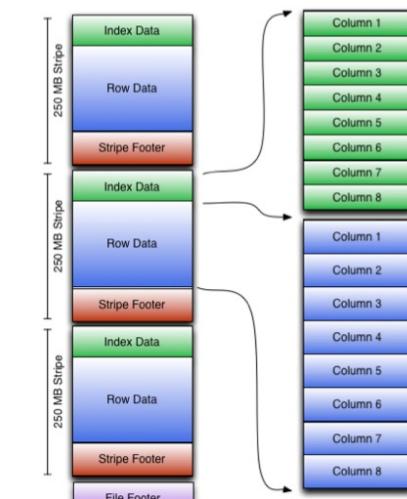
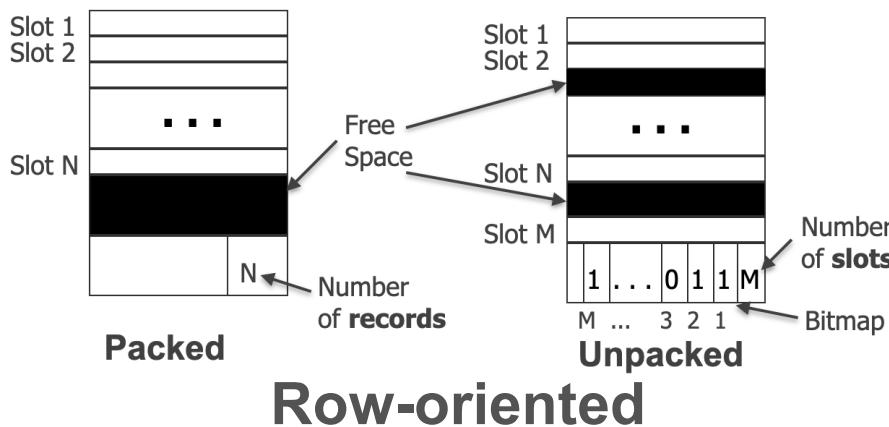
Relational Database



- ❖ Most RDBMSs and Spark serialize a relation as *binary* file(s), often compressed

Aside: Relational File Formats

- ❖ Different RDBMSs and Spark/HDFS-based tools serialize relation/tabular data in different binary formats, often compressed
 - ❖ One file per relation; *data layout* can be row vs columnar (e.g., ORC, Parquet) vs hybrid formats
 - ❖ RDBMS vendor-specific vs open Apache
 - ❖ Parquet becoming especially popular



Ad: Take CSE 132C for more on relational file formats

Data as File: Structured

- ❖ **Structured Data:** A form of data with regular substructure

Matrix

$$\begin{matrix} & 1 & 2 & \dots & n \\ 1 & a_{11} & a_{12} & \dots & a_{1n} \\ 2 & a_{21} & a_{22} & \dots & a_{2n} \\ 3 & a_{31} & a_{32} & \dots & a_{3n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ m & a_{m1} & a_{m2} & \dots & a_{mn} \end{matrix}$$

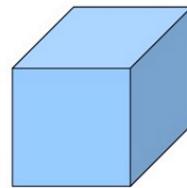
Tensor



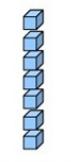
1d-tensor



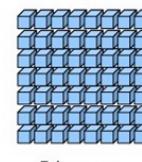
2d-tensor



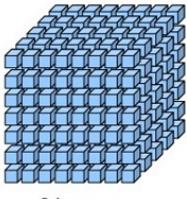
3d-tensor



4d-tensor



5d-tensor



6d-tensor

DataFrame

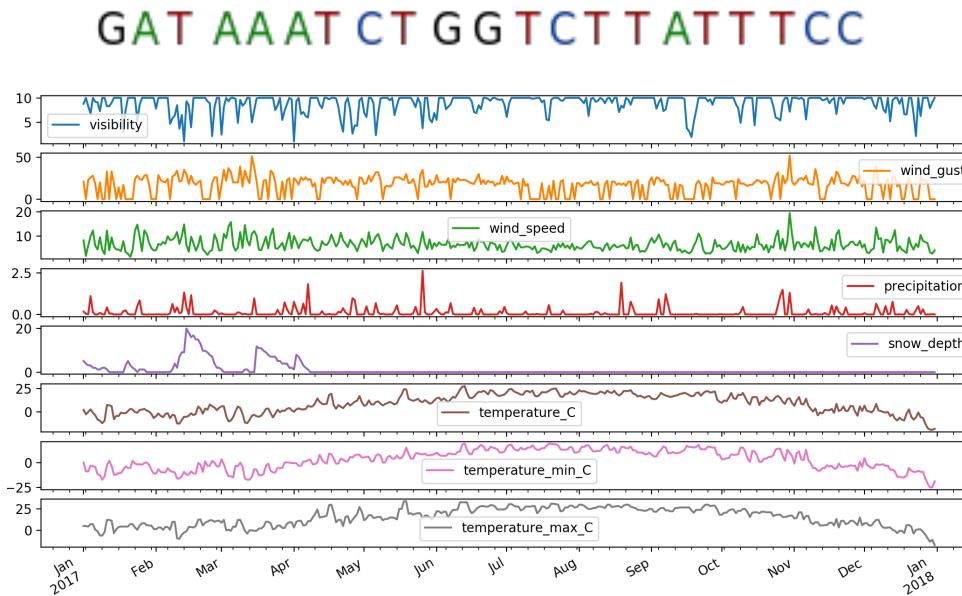
Columns				
	Name	Score	Attempts	Qualify
Rows	0	Anastasia	12.5	1 yes
1	Dima	9.0	3	no
2	Katherine	16.5	2	yes
3	James	NaN	3	no
4	Emily	9.0	2	no

- ❖ Typically serialized as restricted ASCII text file (TSV, CSV, etc.)
- ❖ Matrix/tensor as binary too
- ❖ Can layer on Relations too!

Data as File: Structured

- ❖ **Structured Data:** A form of data with regular substructure

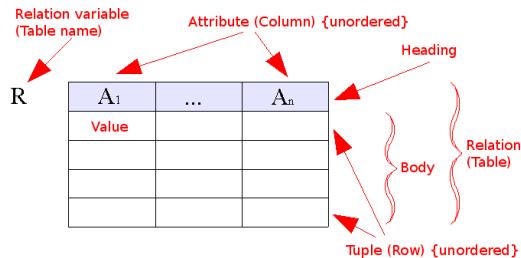
**Sequence
(Includes
Time-series)**



- ❖ Can layer on Relations, Matrices, or DataFrames, or be treated as first-class data model
- ❖ Inherits flexibility in file formats (text, binary, etc.)

Comparing Struct. Data Models

Q: What is the difference between Relation, Matrix, and DataFrame?



$$\begin{matrix} & 1 & 2 & \dots & n \\ 1 & a_{11} & a_{12} & \dots & a_{1n} \\ 2 & a_{21} & a_{22} & \dots & a_{2n} \\ 3 & a_{31} & a_{32} & \dots & a_{3n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ m & a_{m1} & a_{m2} & \dots & a_{mn} \end{matrix}$$

A diagram illustrating a DataFrame. It shows a table with columns labeled "Name", "Score", "Attempts", and "Qualify". The rows are indexed from 0 to 4. Red arrows point from labels to specific parts of the table: "Columns" points to the column headers; "Rows" points to the row indices; and "Data" points to the actual values in the table cells.

	Name	Score	Attempts	Qualify
0	Anastasia	12.5	1	yes
1	Dima	9.0	3	no
2	Katherine	16.5	2	yes
3	James	NaN	3	no
4	Emily	9.0	2	no

- ❖ **Ordering:** Matrix and DataFrame have row/col numbers; Relation is orderless on both axes!
- ❖ **Schema Flexibility:** Matrix cells are numbers. Relation tuples conform to pre-defined schema. DataFrame has no pre-defined schema but all rows/cols can have names; col cells can be mixed types!
- ❖ **Transpose:** Supported by Matrix & DataFrame, not Relation

Data as File: Semistructured

- ❖ **Semistructured Data:** A form of data with less regular / more flexible substructure than structured data

Tree-Structured

```
[  
  {  
    orderId: 1,  
    date: '1/1/2014',  
    orderItems: [  
      {itemId: 1, qty: 3, price: 23.4},  
      {itemId: 23, qty: 2, price: 3.3},  
      {itemId: 7, qty: 5, price: 5.3}  
    ]  
  },  
  {  
    orderId: 2,  
    date: '1/2/2014',  
    orderItems: [  
      {itemId: 31, qty: 7, price: 3.8},  
      {itemId: 17, qty: 4, price: 9.2}  
    ]  
  },  
  {  
    orderId: 3,  
    date: '1/5/2014',  
    orderItems: [  
      {itemId: 11, qty: 9, price: 13.3},  
      {itemId: 27, qty: 2, price: 19.2},  
      {itemId: 6, qty: 19, price: 3.6},  
      {itemId: 7, qty: 22, price: 9.1}  
    ]  
  }  
]
```

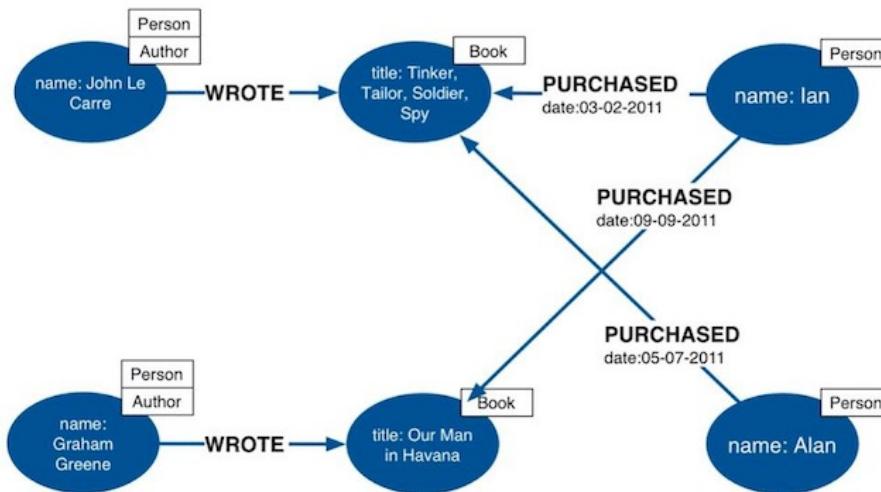
XML Example:

```
<?xml version="1.0" encoding="UTF-8"?>  
<customers>  
  <customer>  
    <customer_id>1</customer_id>  
    <first_name>John</first_name>  
    <last_name>Doe</last_name>  
    <email>john.doe@example.com</email>  
  </customer>  
  <customer>  
    <customer_id>2</customer_id>  
    <first_name>Sam</first_name>  
    <last_name>Smith</last_name>  
    <email>sam.smith@example.com</email>  
  </customer>  
  <customer>  
    <customer_id>3</customer_id>  
    <first_name>Jane</first_name>  
    <last_name>Doe</last_name>  
    <email>jane.doe@example.com</email>  
  </customer>  
</customers>
```

- ❖ Typically serialized as restricted ASCII text file (extensions XML, JSON, YML, etc.)
- ❖ Some data systems also offer binary file formats
- ❖ Can layer on Relations too

Data as File: Semistructured

- ❖ **Semistructured Data:** A form of data with less regular / more flexible substructure than structured data



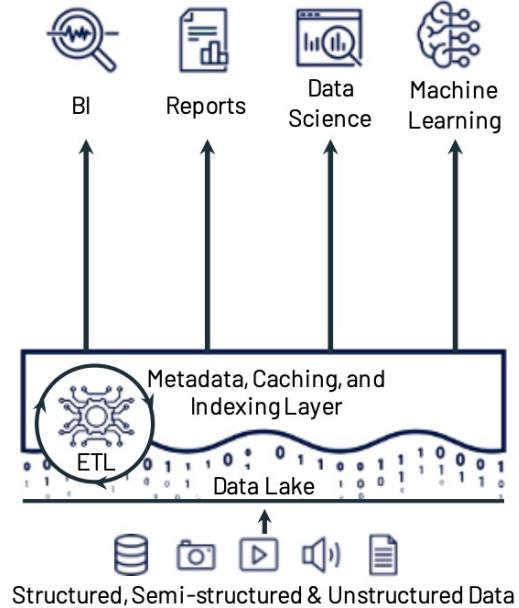
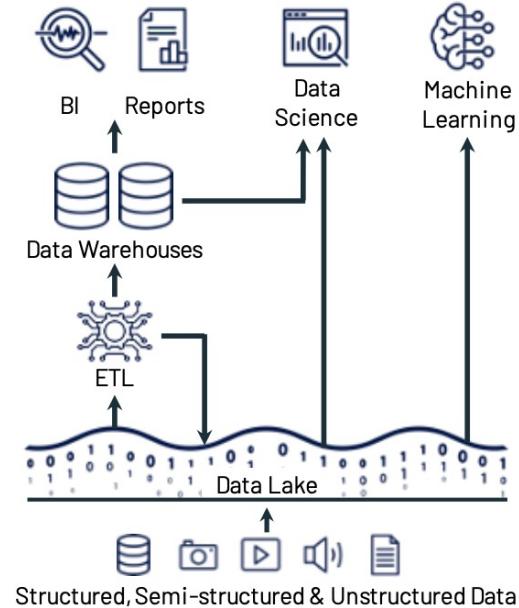
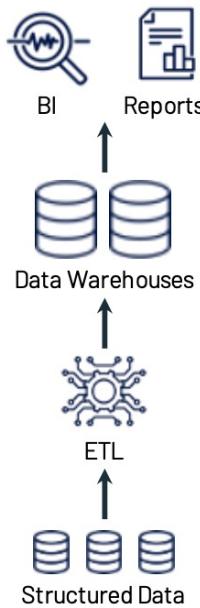
Graph-Structured

- ❖ Typically serialized with JSON or similar textual formats
- ❖ Some data systems also offer binary file formats
- ❖ Again, can layer on Relations too

Ad: Take DSC 104 for more on semistructured data

Data Files on Data “Lakes”

- ❖ **Data “Lake”:** *Loose coupling* of data file format for storage and data/query processing stack (vs RDBMS’s tight coupling)
 - ❖ JSON for raw data; Parquet processed is common



(a) First-generation platforms.

(b) Current two-tier architectures.

(c) Lakehouse platforms.

If interested, check out this vision paper on the future of data lakes:

http://cidrdb.org/cidr2021/papers/cidr2021_paper17.pdf

Data Lake File Format Tradeoffs

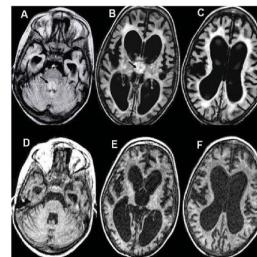
- ❖ Pros and cons of Parquet vs text-based files (CSV, JSON, etc.):
 - ❖ **Less storage:** Parquet stores in **compressed** form; can be much smaller (even 10x); less I/O to read
 - ❖ **Column pruning:** Enables app to read only columns needed to DRAM; even less I/O now!
 - ❖ **Schema on file:** Rich metadata, stats inside format itself
 - ❖ **Complex types:** Can store them in a column
 - ❖ **Human-readability:** Cannot open with text apps directly
 - ❖ **Mutability:** Parquet is immutable/read-only; no in-place edits
 - ❖ **Decompression/Deserialization overhead:** Depends on application tool; can go either way
 - ❖ **Adoption in practice:** CSV/JSON support more pervasive but Parquet is catching up

Data Lake File Format Tradeoffs

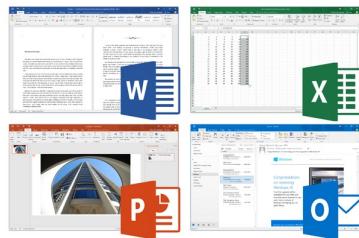
Dataset	Size on Amazon S3	Query Run Time	Data Scanned	Cost
Data stored as CSV files	1 TB	236 seconds	1.15 TB	\$5.75
Data stored in Apache Parquet Format	130 GB	6.78 seconds	2.51 GB	\$0.01
Savings	87% less when using Parquet	34x faster	99% less data scanned	99.7% savings

Data as File: Other Common Formats

- ❖ **Machine Perception** data layer on tensors and/or time-series
- ❖ Myriad binary formats, typically with (lossy) compression, e.g., WAV for audio, MP4 for video, etc.



- ❖ **Text File** (aka plaintext): Human-readable ASCII characters
- ❖ **Docs/Multimodal File**: Myriad app-specific rich binary formats



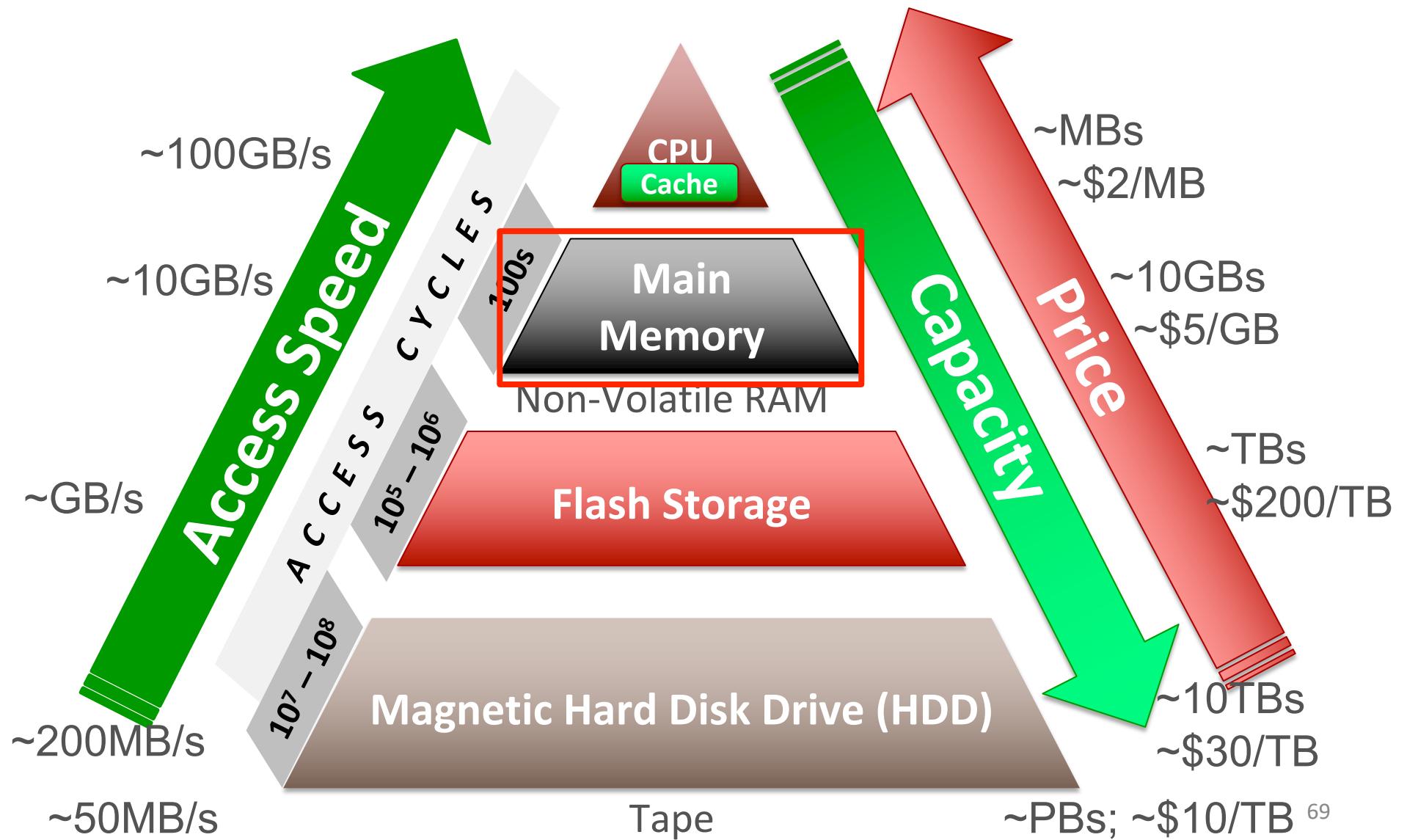
Peer Instruction Activity

(Switch slides)

Outline

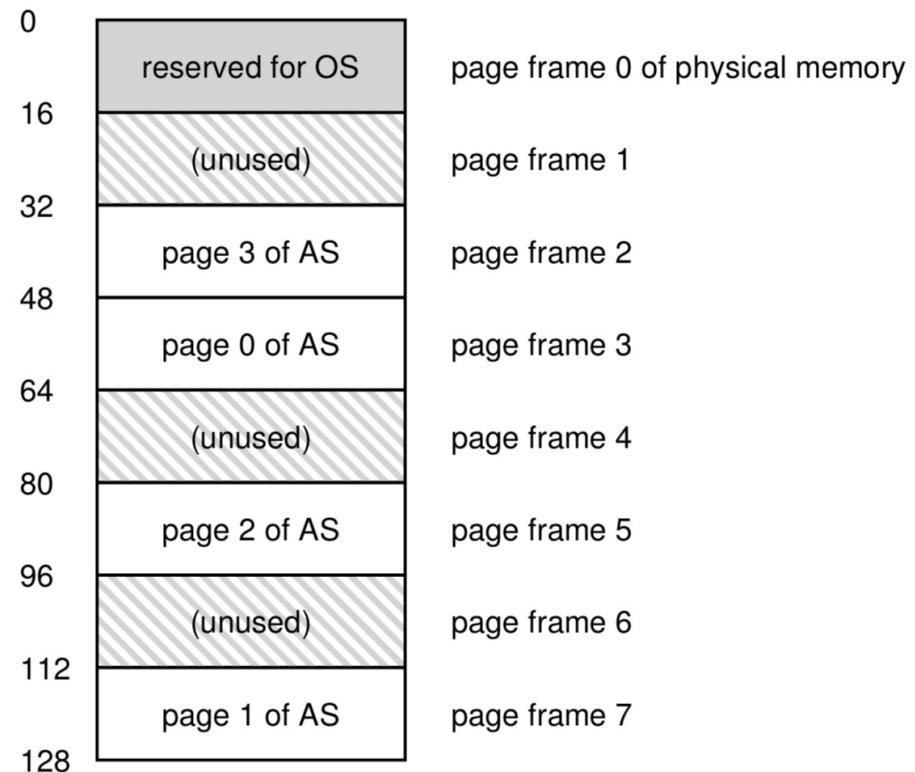
- ❖ Basics of Computer Organization
 - ❖ Digital Representation of Data
 - ❖ Processors and Memory Hierarchy
- ❖ Basics of Operating Systems (OS)
 - ❖ Process Management: Virtualization; Concurrency
 - ❖ Filesystem and Data Files
-  ❖ Main Memory Management
- ❖ Persistent Data Storage

Memory/Storage Hierarchy



Virtualization of DRAM with Pages

- ❖ **Page:** An abstraction of *fixed* size chunks of memory/storage
 - ❖ Makes it easier to virtualize and manage DRAM
- ❖ **Page Frame:** Virtual slot in DRAM to hold a page's content
- ❖ Page size is usually an OS configuration parameter
 - ❖ E.g., 4KB to 16KB
- ❖ **OS Memory Management** has mechanisms to:
 - ❖ Identify pages uniquely
 - ❖ Read/write page from/to disk when requested by a process



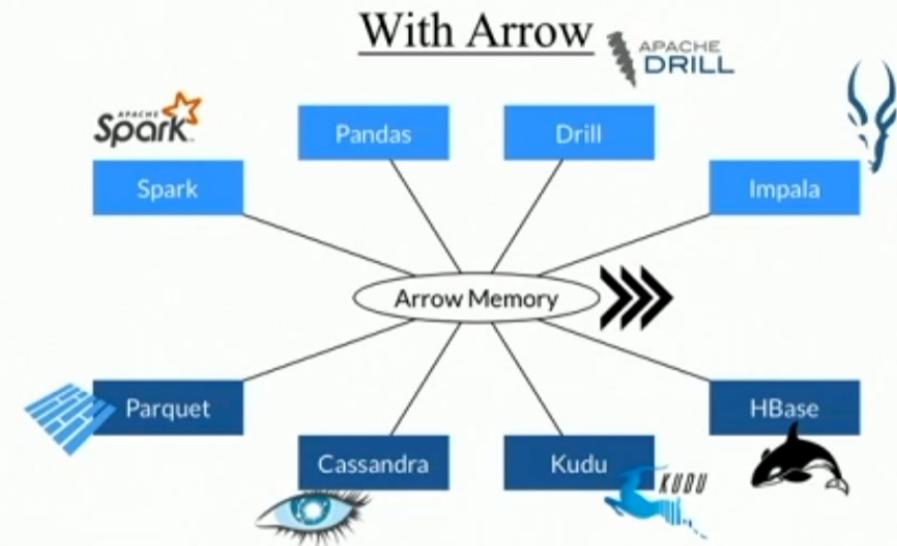
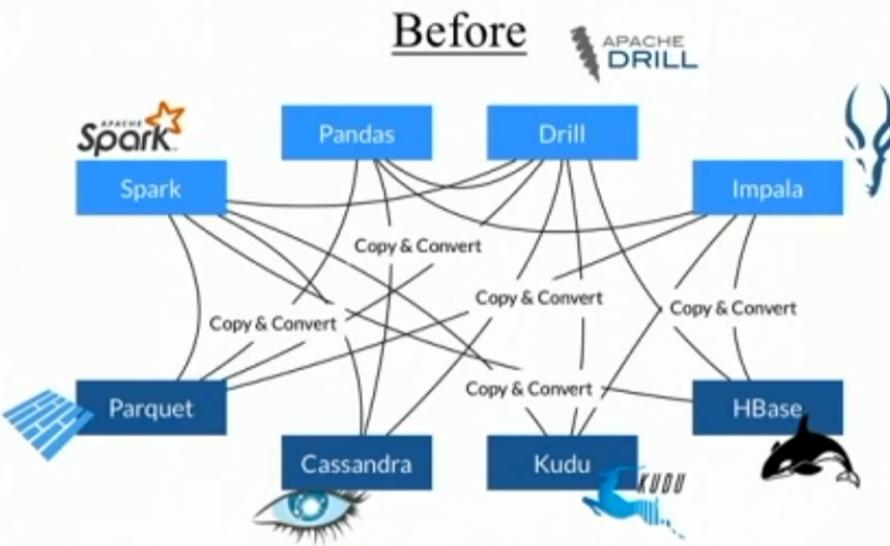
Apportioning of DRAM

- ❖ A process' **Address Space**:
 - ❖ Slice of virtualized DRAM assigned to it alone!
 - ❖ OS "translates" DRAM vs disk address
- ❖ **Page Replacement Policy**:
 - ❖ When DRAM fills up, which cached page to evict?
 - ❖ Many policies in OS literature
- ❖ **Memory Leaks**:
 - ❖ Process forgot to "free" pages used a while ago
 - ❖ Wastes DRAM and slows down system
- ❖ **Garbage Collection**:
 - ❖ Some PL impl. can auto-reclaim some wasted memory

Ad: Take CSE 120 or 132C for more on memory management 71

Storing Data In Memory

- ❖ Any data structure in memory is overlaid on pages
- ❖ Process can ask OS for more memory in System Call API
 - ❖ If OS denies, process may crash; your PA0 Dask crashes?
- ❖ **Apache Arrow:**
 - ❖ Emerging standard for columnar in-memory data layout
 - ❖ Compatible with Pandas, (Py)Spark, Parquet, etc.



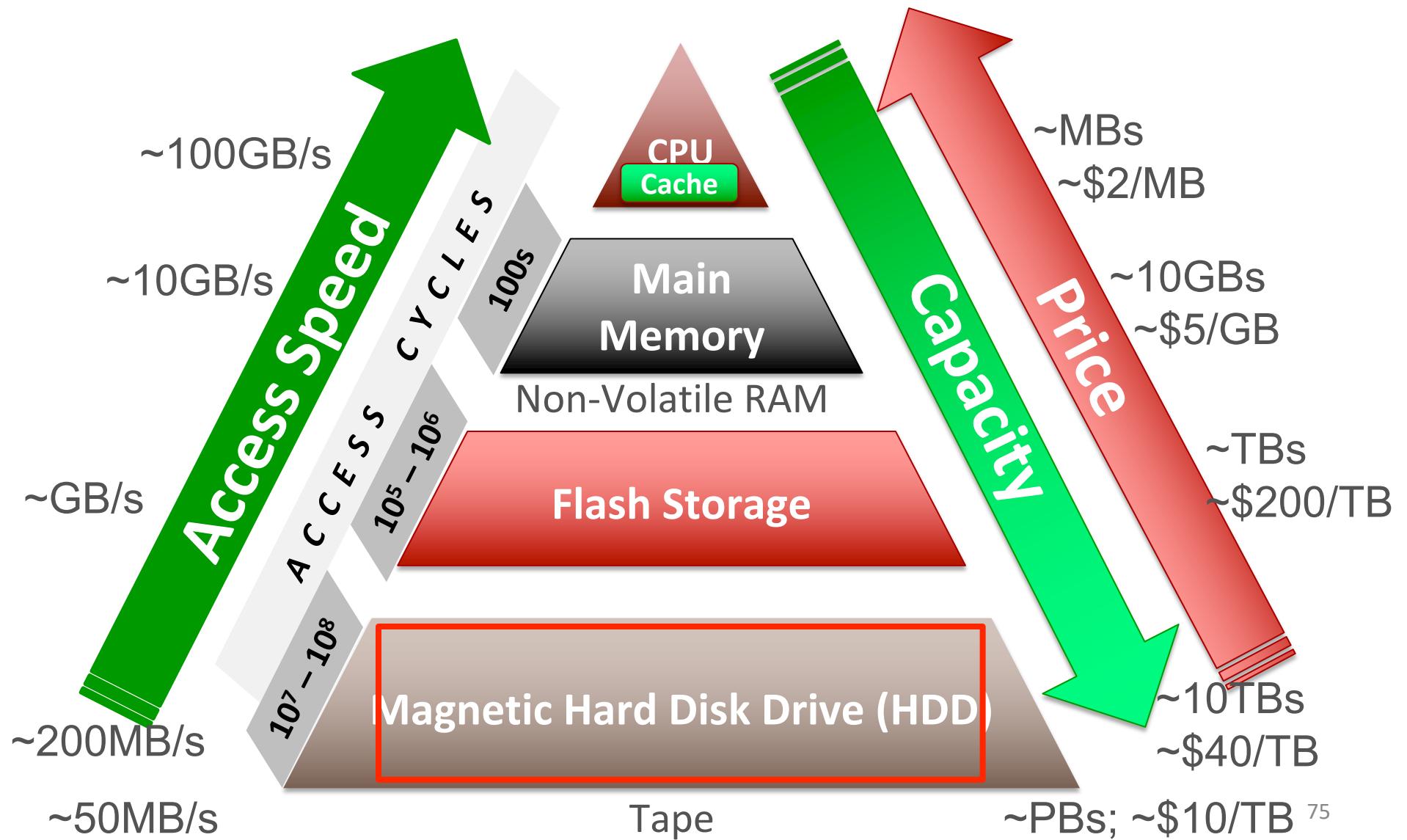
Outline

- ❖ Basics of Computer Organization
 - ❖ Digital Representation of Data
 - ❖ Processors and Memory Hierarchy
- ❖ Basics of Operating Systems (OS)
 - ❖ Process Management: Virtualization; Concurrency
 - ❖ Filesystem and Data Files
 - ❖ Main Memory Management
- ❖ Persistent Data Storage

Persistent Data Storage

- ❖ **Persistence:** Program state/data is available intact even after process finishes
- ❖ **Volatile Memory:** A data storage device that needs power/electricity to store bits; e.g., DRAM, CPU caches (SRAM)
- ❖ **Non-Volatile or Persistent mem./storage:** A data storage device that retains bits intact after power cycling
 - ❖ E.g., all levels below DRAM in memory hierarchy
 - ❖ “**Persistent Memory (PMEM)**”: Marketing term for large DRAM that is backed up by battery power!
 - ❖ **Non-Volatile RAM (NVRAM)**: Popular term for DRAM-like device that is genuinely non-volatile (no battery)

Memory/Storage Hierarchy



Disks

- ❖ Aka secondary storage; likely holds the vast majority of the world's day-to-day business-critical data!
- ❖ Data storage/retrieval units: **disk blocks or pages**
- ❖ Unlike RAM, different disk pages have different retrieval times based on location:
 - ❖ Need to optimize *data layout* on disk pages
 - ❖ Orders of magnitude performance gaps possible

Data Organization on Disk

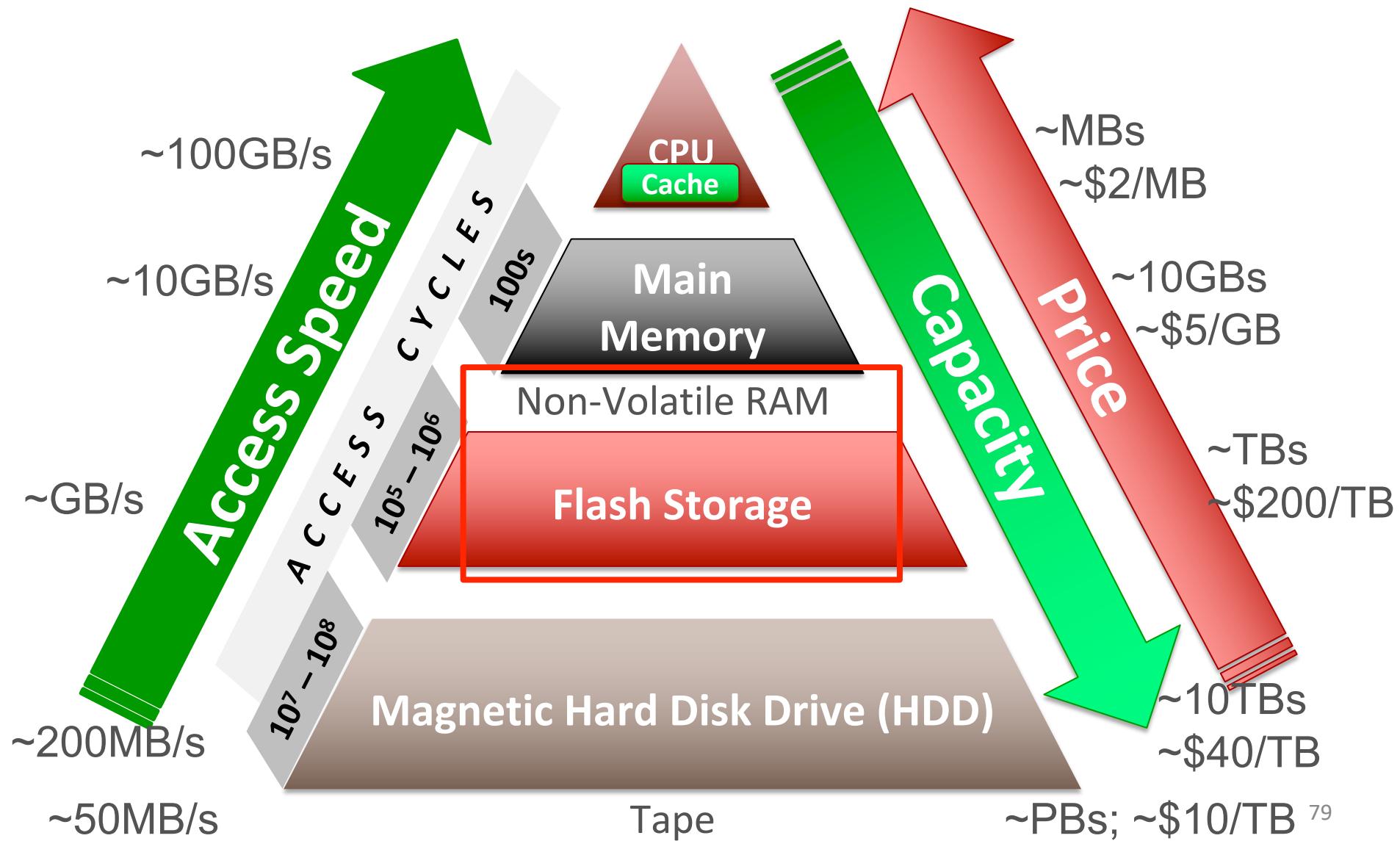
- ❖ Disk space is organized into **files**
- ❖ Files are made up of disk **pages** aka **blocks**
- ❖ Typical disk block/page size: 4KB or 8KB
 - ❖ Basic unit of reads/writes for a disk
 - ❖ OS/RAM page is *not* the same as disk page!
 - ❖ Typically, OS/RAM page size = disk page size but not always; disk page can be a multiple, e.g., 1MB
- ❖ File data (de-)allocated in increments of disk pages

Magnetic Disk Quirks

- ❖ **Key Principle:** Sequential v Random Access Dichotomy
- ❖ Accessing disk pages in sequential order gives *higher throughput*
 - ❖ Random reads/writes are OOM slower!
- ❖ Need to carefully lay out data pages on disk
- ❖ Abstracted away by data systems: Dask, Spark, RDBMSs, etc.

Ad: Take CSE 132C for more on quirks of magnetic disks

Memory/Storage Hierarchy



Flash SSD vs Magnetic Hard Disks

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

- ❖ Random reads/writes are not much worse
 - ❖ Different locality of reference for data/file layout
 - ❖ But still block-addressable like HDDs
- ❖ Data access latency: 100x faster!
- ❖ Data transfer throughput: Also 10-100x higher
- ❖ Parallel read/writes more feasible
- ❖ Cost per GB is 5-15x higher!
- ❖ Read-write impact asymmetry; much lower lifetimes

NVRAM vs Magnetic Hard Disks

Roughly speaking, NVRAM is like a non-volatile form of 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)
 - ❖ Spatial locality of reference like DRAM; radical change!
- ❖ Latency, throughput, parallelism, etc. similar to DRAM
- ❖ Alas, yet to see light of day in production settings
- ❖ Cost per GB: No one knows for sure yet!

Review Questions

- ❖ How is a database different from a file?
- ❖ What are the 2 levels of a database? Why the dichotomy?
- ❖ What type of data modality is JSON meant to capture?
- ❖ Explain 2 differences between a relation and a DataFrame.
- ❖ Can you store a relation as a DataFrame? Vice versa?
- ❖ Can you store a tensor as a relation? Vice versa?
- ❖ What is the address space of a process? What is a memory leak?
- ❖ What is Parquet? Explain 3 pros of Parquet over CSVs.
- ❖ What is Arrow? How is it different from Parquet?
- ❖ Which storage device has random-sequential access dichotomy?

Peer Instruction Activity

(Switch slides)

Outline

- ❖ Basics of Computer Organization
 - ❖ Digital Representation of Data
 - ❖ Processors and Memory Hierarchy
- ❖ Basics of Operating Systems (OS)
 - ❖ Process Management: Virtualization; Concurrency
 - ❖ Filesystem and Data Files
 - ❖ Main Memory Management
- ❖ Persistent Data Storage