DSC 102
Systems for Scalable Analytics

Rod Albuyeh

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

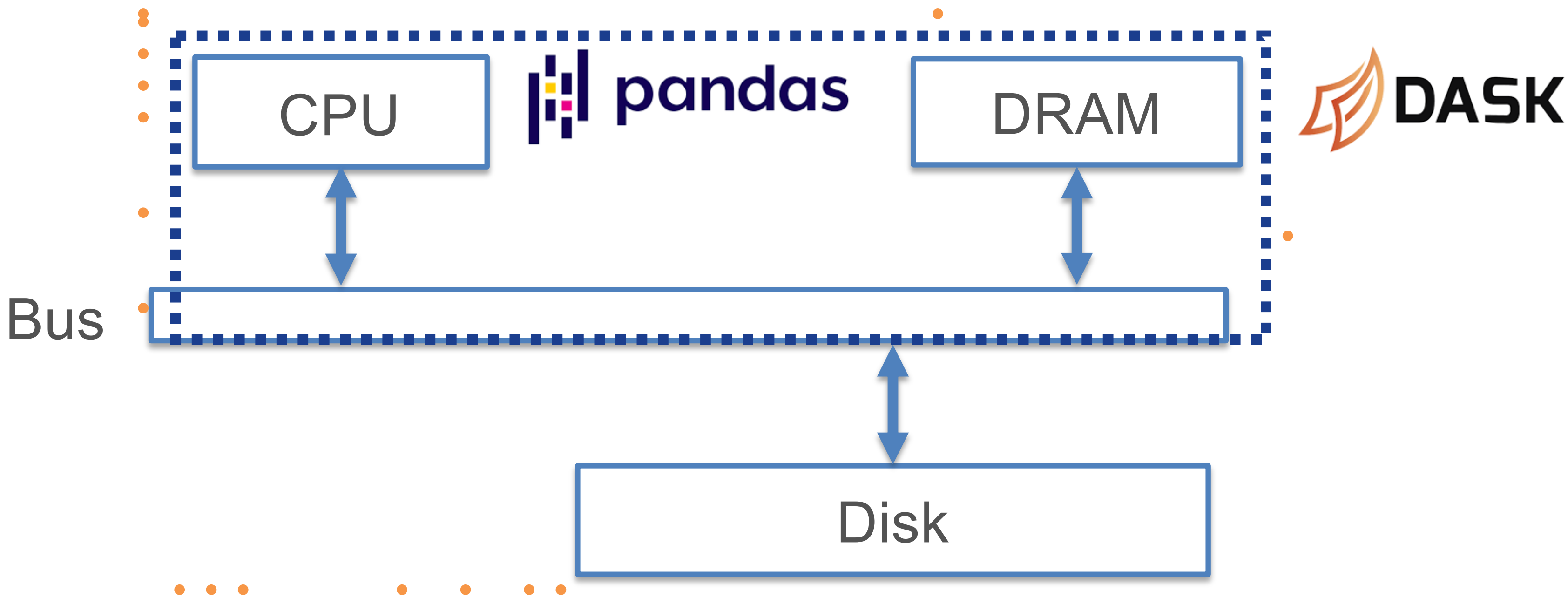
Ch. 2, 4.1-4.2, 6, 7, 13, 14.1, 18.1, 21, 22, 26, 36, 37, 39, and 40.1-40.2 of Comet Book

PA0 Released!

- If you did not declare a PA group already, you will be assigned to an individual group.
- Two students have mentioned issues accessing AWS portal, anybody else? Look again.
- UTF-8 vs. ASCII friendly group names, group names with potential delimiters
- Let’s glance over the release...

Memory Hierarchy in PA0

- Pandas DataFrame needs data to fit entirely in DRAM
- **Dask DataFrame** automatically manages Disk vs DRAM for you
  - Full data sits on Disk, brought to DRAM upon compute()
  - Dask stages out computations using Pandas



- **Tradeoff:** Dask may throw memory configuration issues. :)

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 an OS? Why do we need it?*

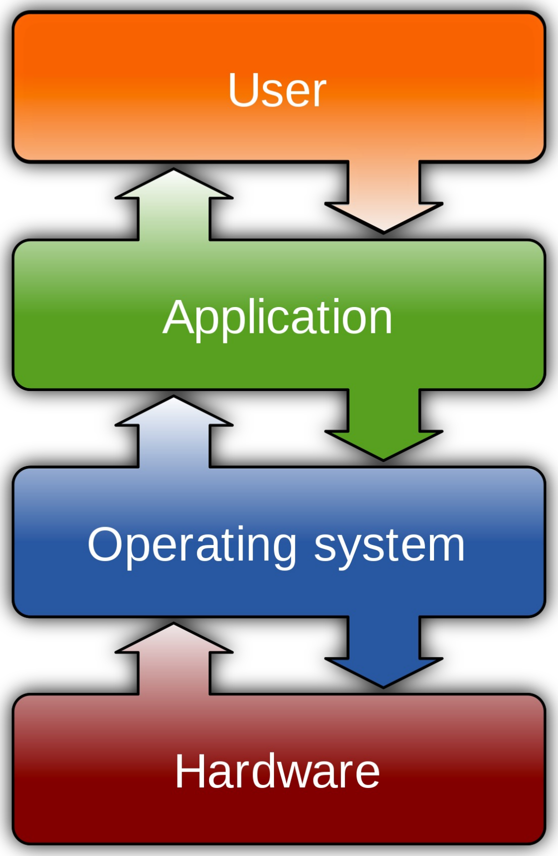

Master


105

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*
- Without OS, computer users must speak machine code!
- 2 key principles in OS (any system) design & implementation:
  - **Modularity:** Divide system into *functionally cohesive components* that each do their jobs well
    - Orchestra example: Consider a conductor orchestrating different sections
  - **Abstraction:** *Layers of functionalities* from low-level (close to hardware) to high level (close to user)
    - Car example: A pedal to transmission to engine to wheels

Role of an OS in a Computer



APIs/Interface of Application Software

APIs of OS aka “System Calls”

Hardware-specific code parts of OS

“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.

“System Call” APIs

Kernel Components	Process Management	Main Memory Management	Filesystems	Networking	Device Drivers
Functionality	*Virtualize* processor; “**Process**” abstraction; **Concurrency**	*Virtualize* Main Memory	*Virtualize* disks; “**File**” abstraction; **Persistence**	Commun. over network	Talk to other I/O devices

Hardware device-specific programs

Hardware

Outline

- Basics of Computer Organizati
  - Digital Representation of Data
  - Processors and Memory Hierarc
- Basics of Operating Systems (OS
  - Process Management: Virtualization; Concurrency
  - Filesystem and Data Files
  - Main Memory Manageme
- 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 how data-intensive computations are handled under the hood.

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**

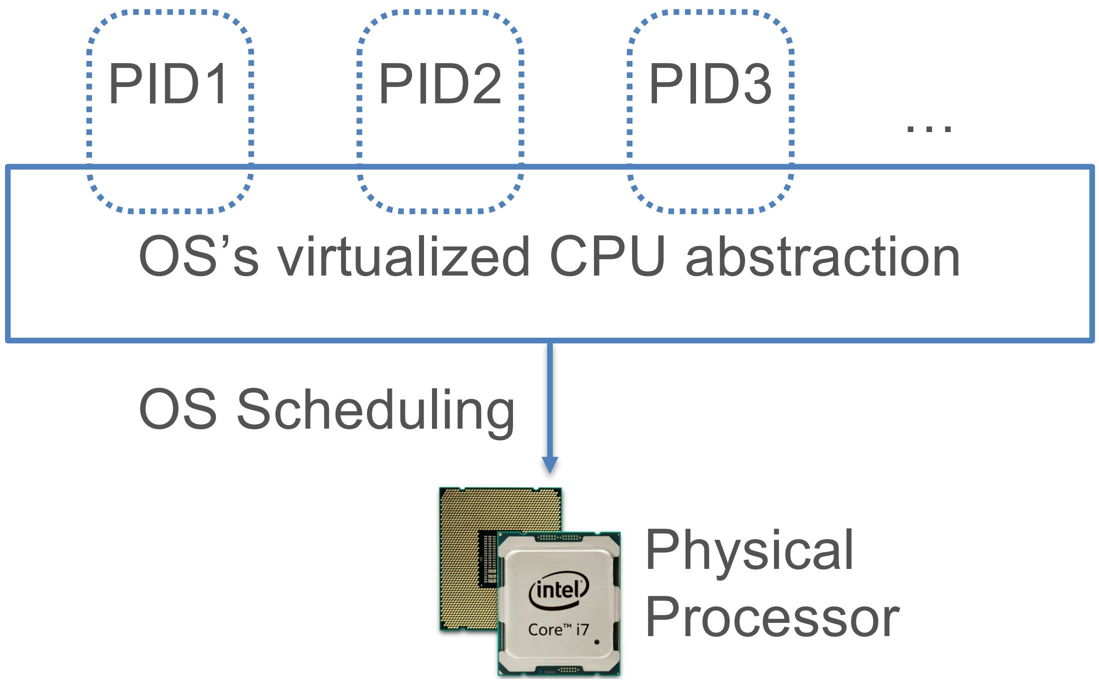
Virtualization of Hardware Resources

***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 divide up and share 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

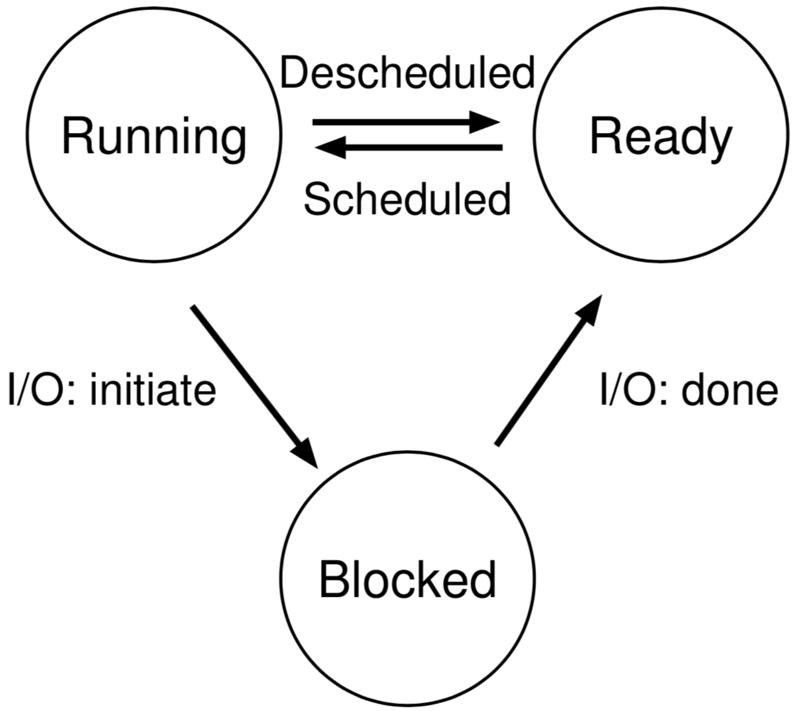
- Virtualization of processor enables process **isolation**, i.e., each process given an “illusion” that it alone runs



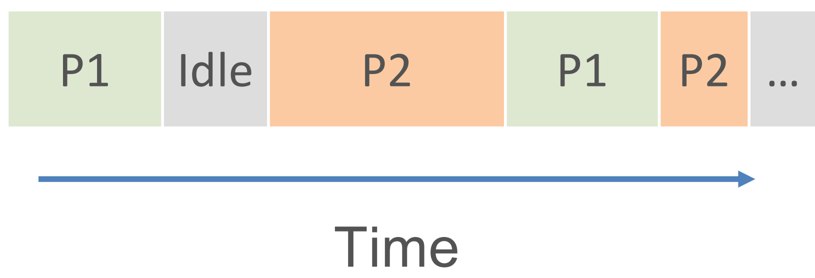
- Inter-process communication possible in System Calls API
- Later: Generalize to **Thread** abstraction for **concurrency**

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

Scheduling Policies/Algorithms

- In general, OS may not know all Arrival Times and Job Lengths 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*
  - Fairness: Many metrics exist (e.g., Jain's fairness index)

Scheduling Policies/Algorithms

- 100s of scheduling policies studied!

  We will be overviewing some well-known ones:

  FIFO (First-In-First-Out)

  SJF (Shortest Job First)

  SCTF (Shortest Completion Time First)

  Round Robin

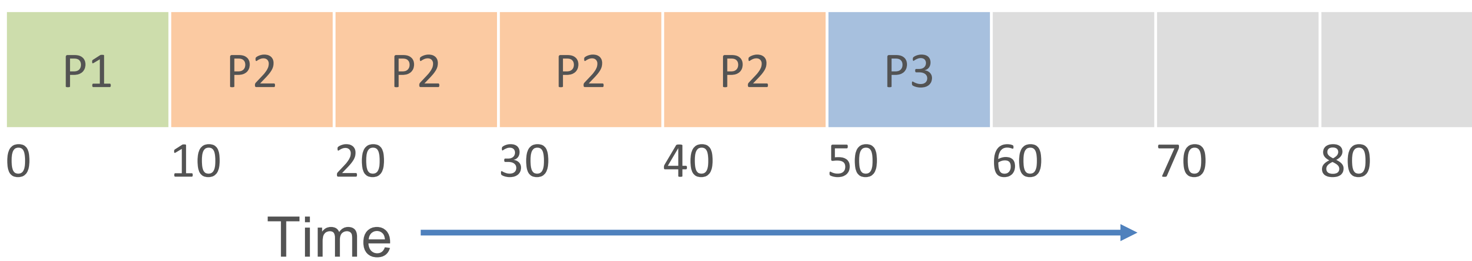
  Random, etc.

  - 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-Served (FCFS)
- Ranking criterion: Arrival Time; no preemption allowed

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



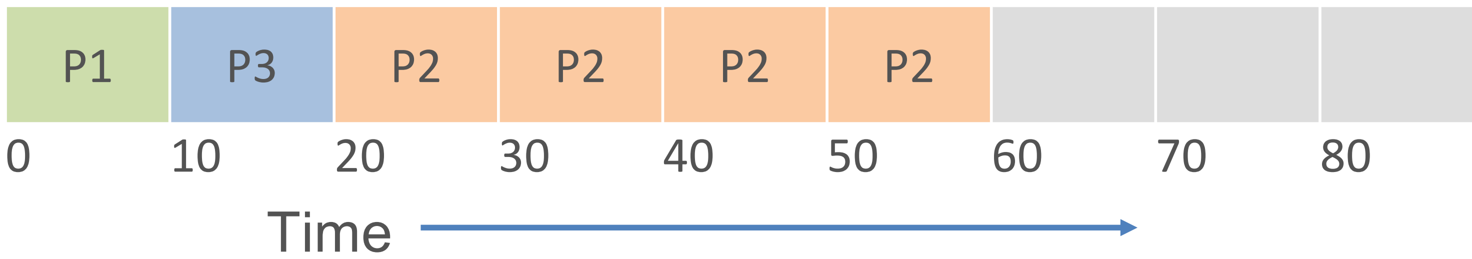
Process	Arrival t1	Start t2	Completion t3	Response t4 = t2-t1	Turnaround t5 = t3-t1
P1	0	0	10	0	10
P2	0	10	50	10	50
P3	0	50	60	50	60
			Avg:	20	40

- 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



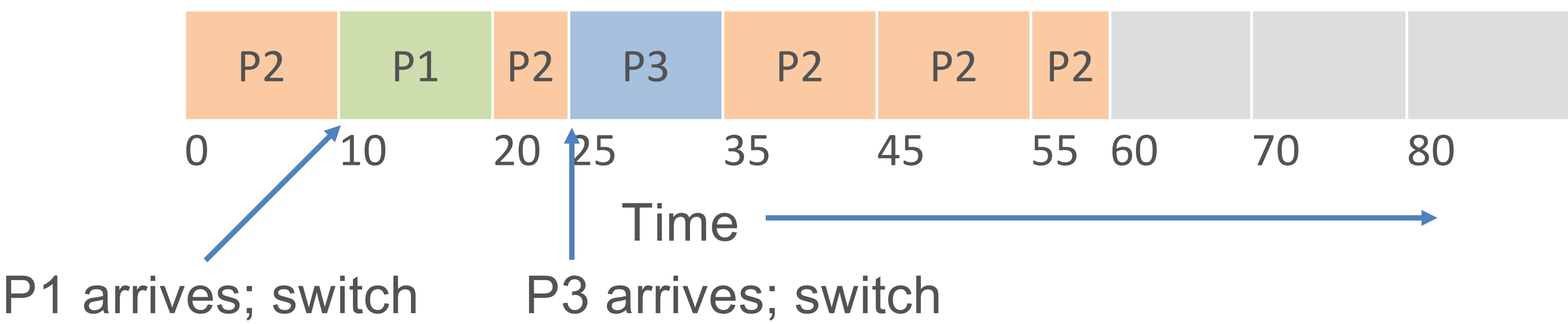
Process	Arrival t1	Start t2	Completion t3	Response t4 = t2-t1	Turnaround t5 = t3-t1
P1	0	0	10	0	10
P2	0	20	60	20	60
P3	0	10	20	10	20
	(FIFO Avg: 20 and 40)		Avg:	10	30

- Main con: Not all Job lengths might be unknown 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



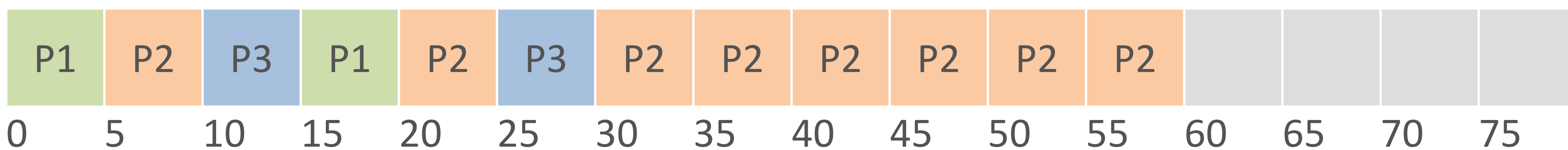
Process	Arrival t1	Start t2	Completion t3	Response t4 = t2-t1	Turnaround t5 = t3-t1
P1	10	10	20	0	10
P2	0	0	60	0	60
P3	25	25	35	0	10
	(SJF Avg: 10 and 30)		Avg:	0	26.7

- Main con same as SJF: Job lengths might be unknown beforehand.

Scheduling Policy: Round Robin

- In Round Robin job lengths need not be known
- 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



Quantum is 5 Time →

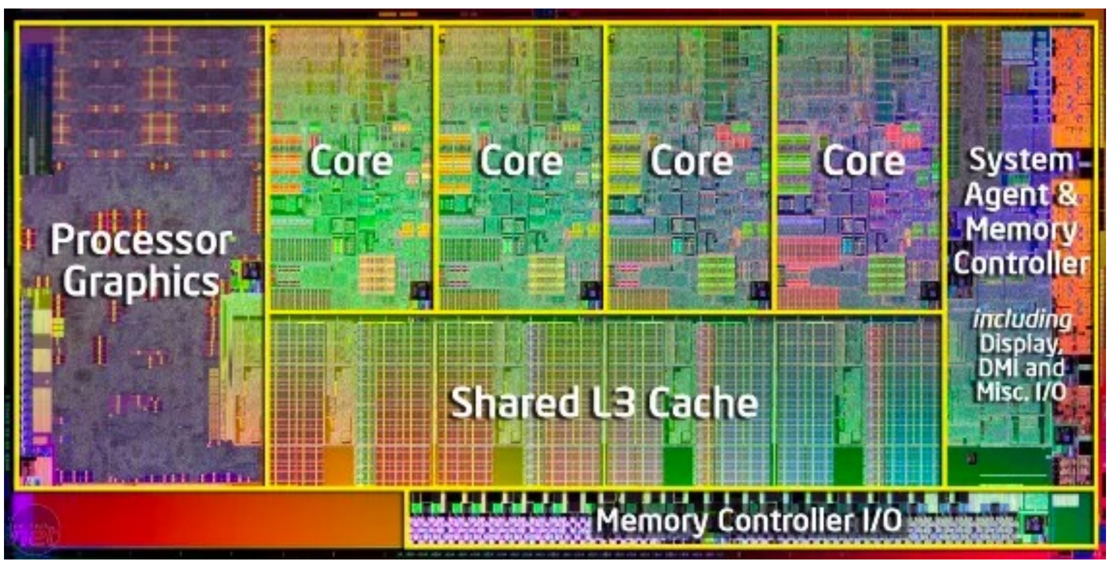
Process	Arrival t1	Start t2	Completion t3	Response t4 = t2-t1	Turnaround t5 = t3-t1
P1	0	0	20	0	20
P2	0	5	60	5	60
P3	0	10	30	10	30
			Avg:	5	36.7

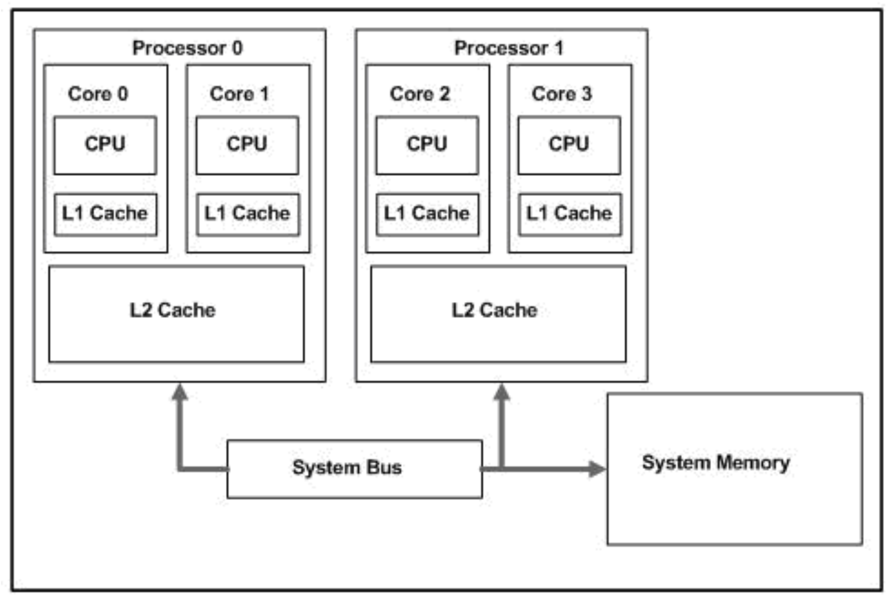
(SJF Avg: 10 & 30; SCTF Avg: 0 & 26.7)

- 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

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 processor/core has its own job queue
  - OS moves jobs across queues based on load
  - Example Gantt chart for MQMS:

CPU 1:	P1	P1	P3	P3	P3	P3	P1	P1	P1
CPU 2:	P2	P2	P2	P1	P1	P2	P2	P3	P3
	0	10	20	30	40	50	60	70	80

Concurrency in Data Science

- Thankfully, most data-intensive computations in data science do not need concurrent writes on shared data! Although we often need concurrent reads.
  - 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

1. Briefly explain two differences between DRAM and disk.
2. Why is it important to align data access pattern and data layout?
3. What is the purpose of an OS?
4. Why is the design of an OS so modular?
5. Why does an OS need to use a scheduling policy?
6. Which quantity captures latency of a process starting: Response Time or Turnaround Time?
7. What gives rise to different scheduling policies?
8. Which scheduling policy is the fairest among the ones we covered?
9. What is the Convoy Effect? Which sched. policy has that issue?

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?*

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
  - 1000s 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.

Q: *Are files stored contiguously on disk?*

Filesystem

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

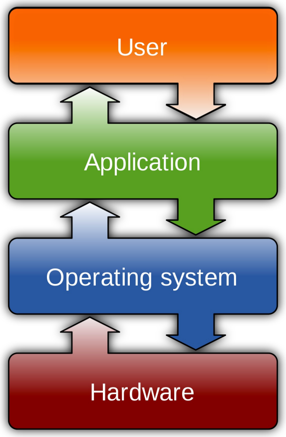
Filesystem

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

Virtualization of File on Disk

- OS abstracts a file on disk as a virtual object for processes
- **File Descriptor:** An OS-assigned positive 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 descriptor (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
- **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 read/write later)
- Dozens more (rename, mkdir, chmod, etc.)

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

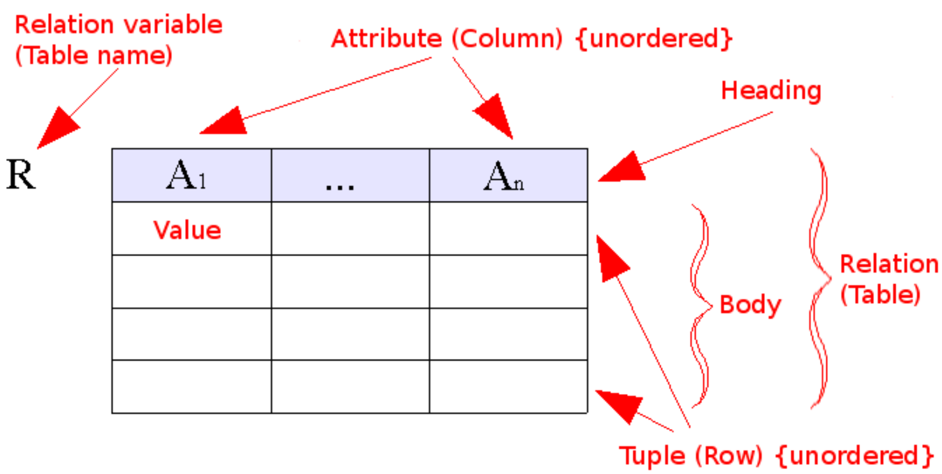
Files Vs Databases: Data Model

- **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, PyTorch, etc.) are application/platform software that use OS System Call API for handling data files

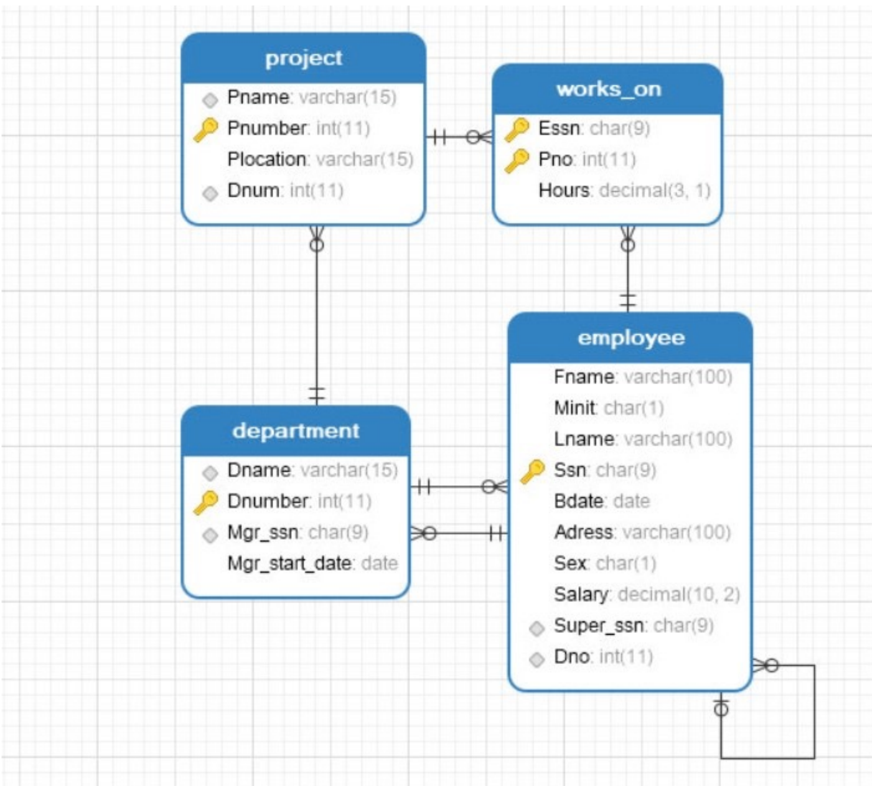
Data as File: Structured

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

Relation



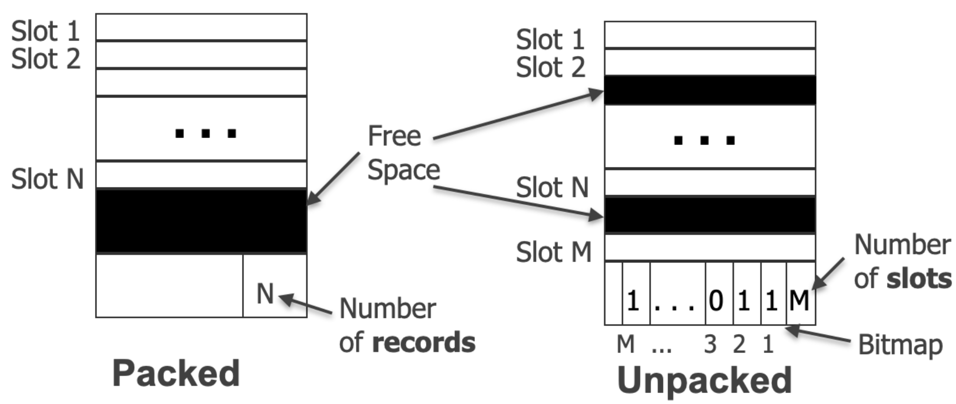
Relational Database



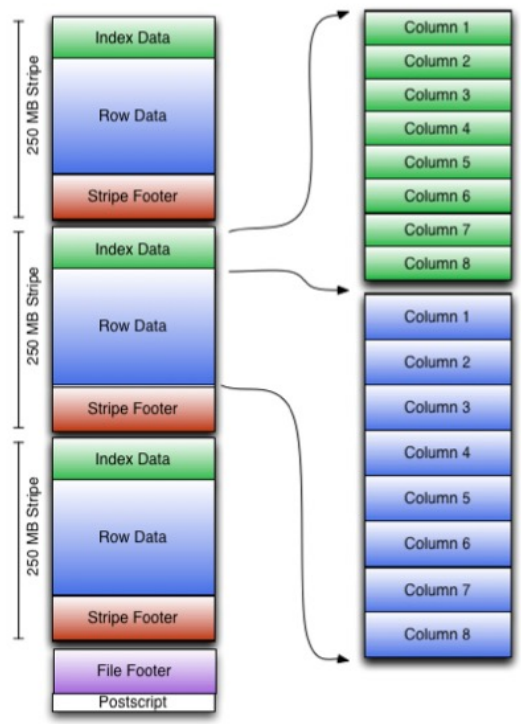
❖ Most RDBMSs and Spark serialize a relation as *binary* file(s), often compressed

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; row vs columnar (e.g., ORC, Parquet) vs hybrid formats
  - RDBMS vendor-specific vs open Apache
  - Parquet becoming especially popular



Row-oriented



Columnar

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

Q: *Suppose you have a dataset of 10,000 columns and only want to select 10 columns. Which format should you use? What do you gain from your selection?*

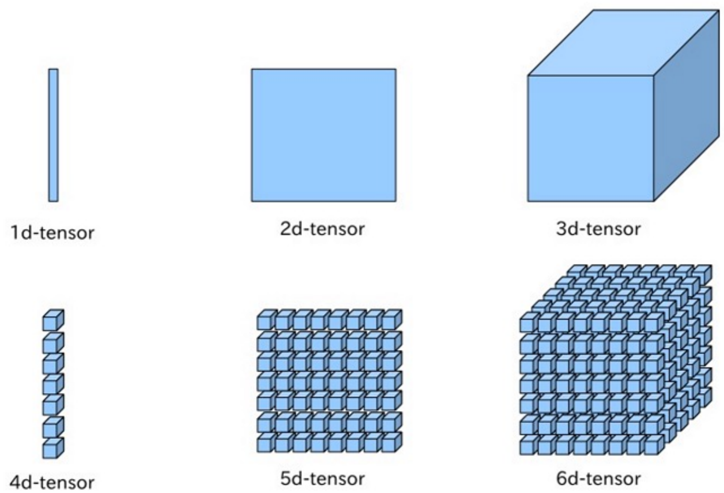
Data as File: Structured

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

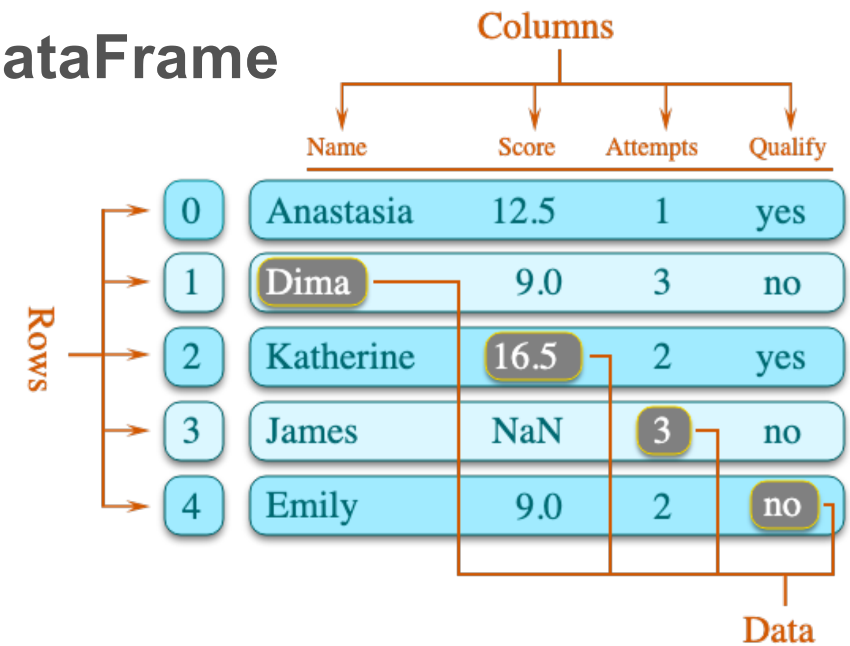
Matrix

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

Tensor



DataFrame



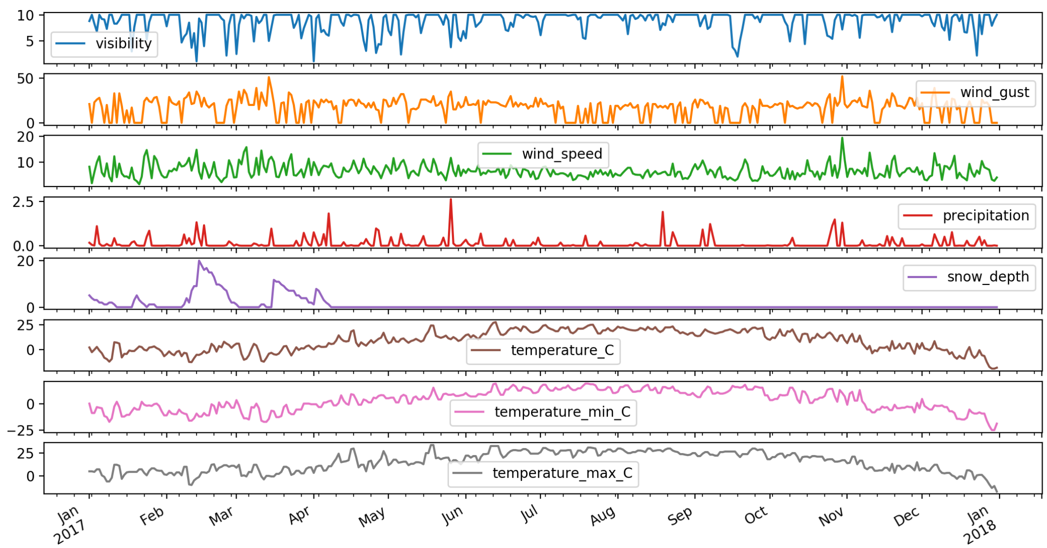
- Typically serialized as restricted ASCII text file (TSV, CSV, etc.)
- Matrix/tensor as binary too
- Can layer on Relations too!

Data as File: Structured

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

Sequence (Includes Time-series)

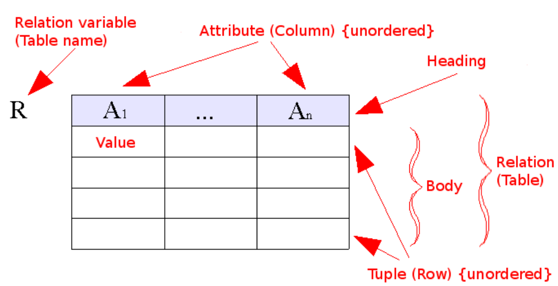
GAT AAAT CT GGTCTTATTTCC

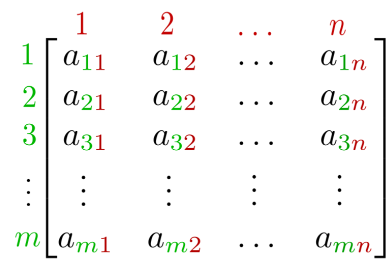


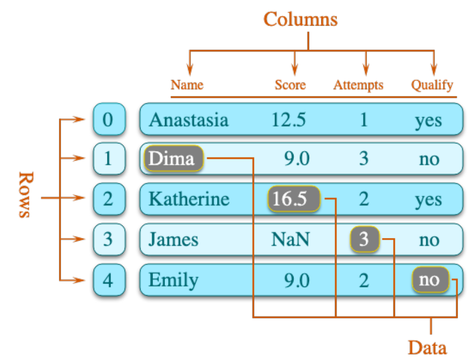
- Can layer on Relations, Matrices, or DataFrames, or be treated as first-class data model
- Inherits flexibility in file formats (text, binary, etc.)

Comparing Structured Data Models

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







- **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

If interested in reading more:
https://towardsdatascience.com/preventing-the-death-of-the-dataframe-8bca1c0f83c8

Data as File: Semistructured

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

Tree-Structured

```xml
<?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>
```

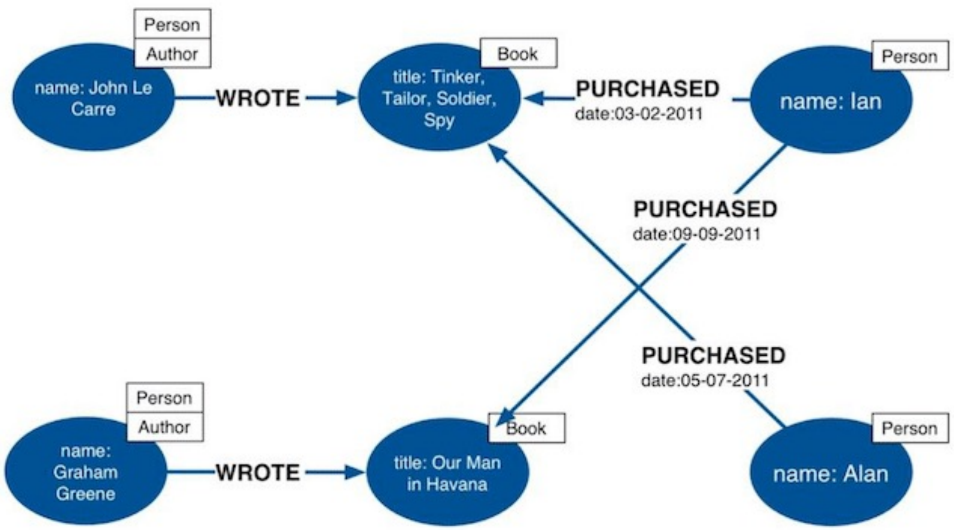
```
[
    {
        orderId: 1,
        date: '1/1/2014',
        orderItems: [
                {itemId: 1, qty: 3, price: 23.4},
                {itemId: 23, qty: 2, price: 3.3},
                {itemId: 7, qty: 5, price: 5.3}
            ]
    },
    {
        orderId: 2,
        date: '1/2/2014',
        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}
        ]
    }
]
```

- 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

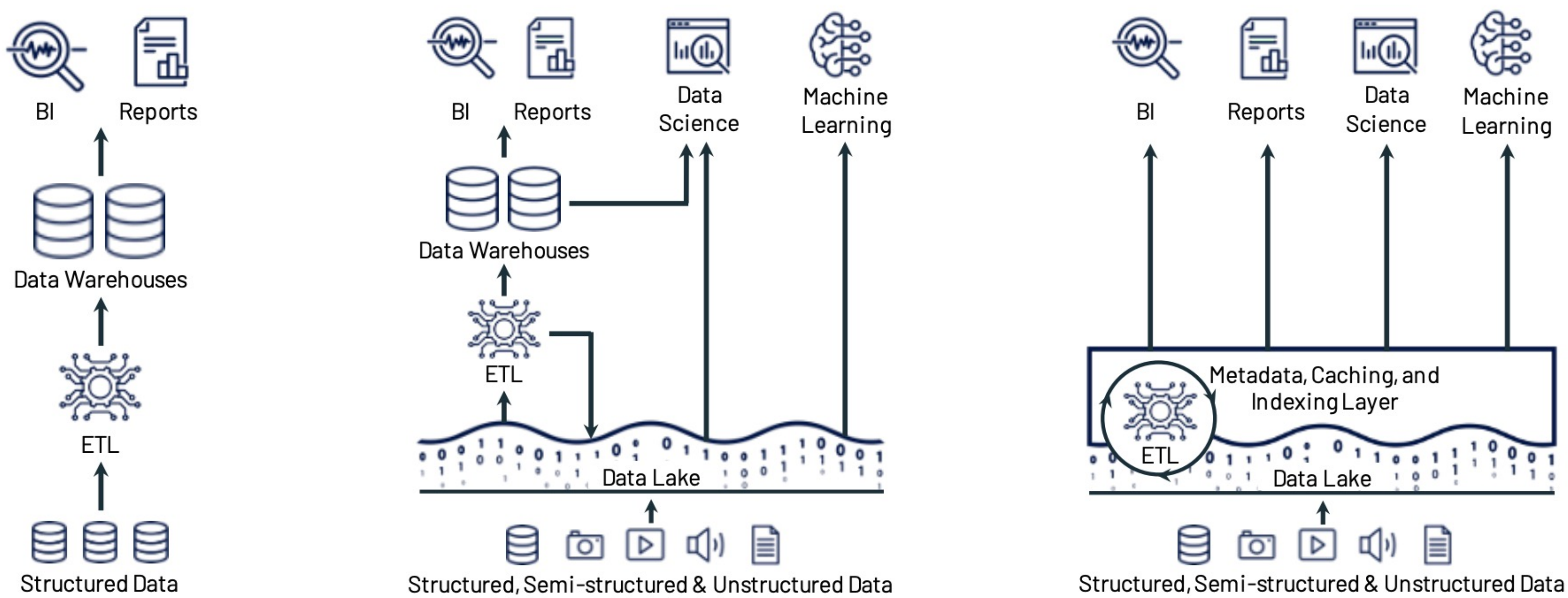
Graph-Structured



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

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.**

ETL: Extract, Transform, Load

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
  - **Adoption in practice**: CSV/JSON support more pervasive but Parquet is catching up, especially in enterprise “big data” situations

Data Lake File Format Tradeoffs

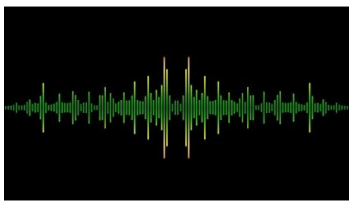
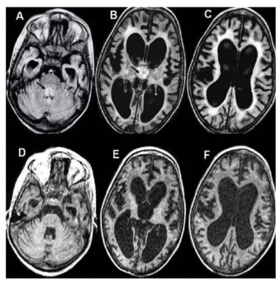
Hypothetical but realistic query performance

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

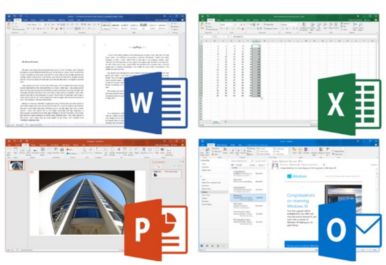
https://databricks.com/glossary/what-is-parquet

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



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