

# DSC 102

# Systems for Scalable Analytics

Winter 2022

Arun Kumar

# About Myself



2009: Bachelors in CSE from IIT Madras, India

Summers: 110F!



2009–16: MS and PhD in CS from UW-Madison  
PhD thesis area: Data systems for ML workloads

Winters: -40F!



2016-: Asst. Prof. at UC San Diego CSE  
2019-: + Asst. Prof. at UC San Diego HDSI  
2021: Assoc. Prof. at CSE & HDSI

Ahh! :)

# My Current Research

New abstractions, algorithms, and software systems  
to “***democratize***” ML-based data analytics from  
a data management/systems standpoint

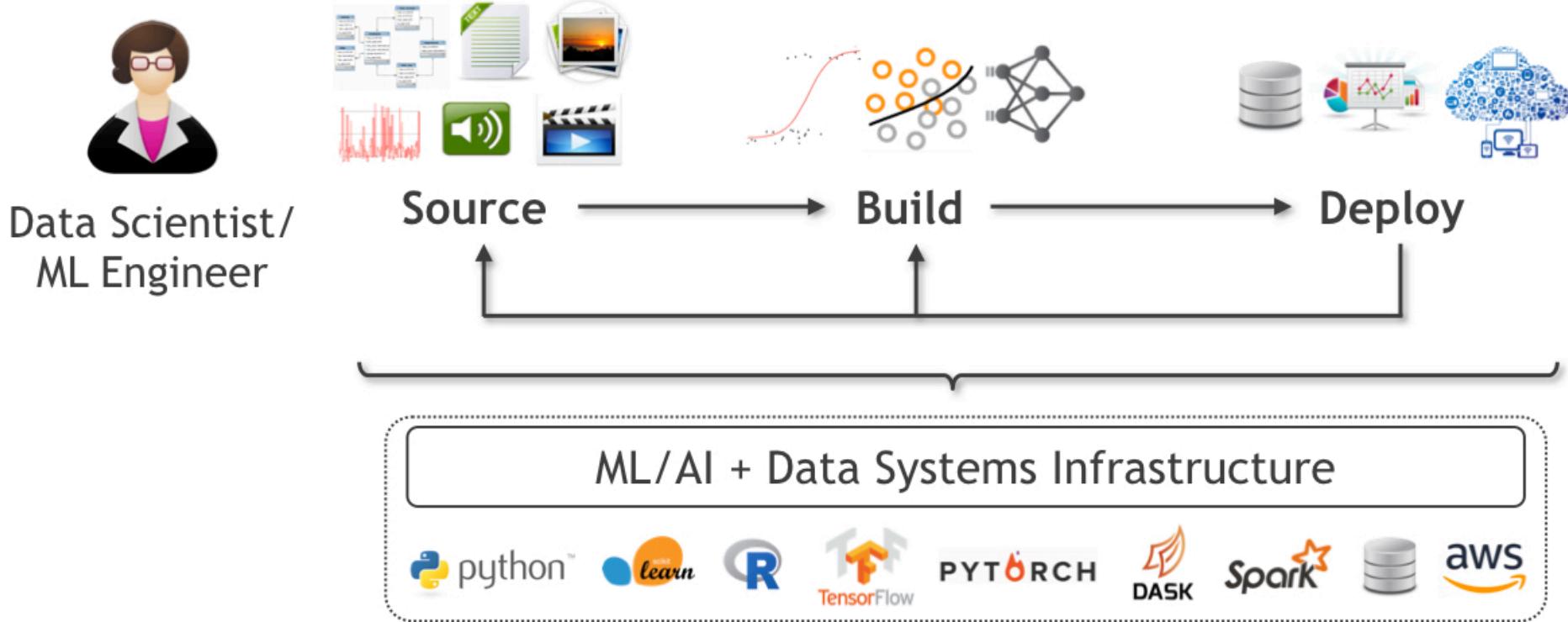
$$\text{Democratization} = \frac{\text{System Efficiency}}{(\text{Lower resource costs})} + \frac{\text{Human Efficiency}}{(\text{Higher productivity})}$$

# Practical and scalable data systems for ML analytics

**Inspired by *relational database systems* principles**

Exploit insights from *learning theory* and *optimization theory*

# My Current Research



**Research Approach :** *Abstract* key steps + *Formalize* computation + *Automate* grunt work + *Optimize* execution

What is this course about? Why take it?

# 1. Netflix's “spot-on” recommendations

## NETFLIX ORIGINAL **STRANGER THINGS**

95% Match 2017 2 Seasons 4K Ultra HD 5.1

When a young boy vanishes, a small town uncovers a mystery involving secret experiments, terrifying supernatural forces and one strange little girl.

*Winona Ryder, David Harbour, Matthew Modine*  
TV Shows, TV Sci-Fi & Fantasy, Teen TV Shows



### Popular on Netflix



### Recently Watched



# How does Netflix know that?

# Large datasets + Machine learning!

**Everything is a Recommendation**



**Over 80% of what people watch comes from our recommendations**

**Recommendations are driven by Machine Learning**

6

Log all user behavior (views, clicks, pauses, searches, etc.)  
Recommender systems apply ML to TBs of data from all users and movies to deliver a tailored experience

# 2. Structured data with search results

Google pradeep khosla

All News Images Videos Maps More Settings Tools

About 274,000 results (0.51 seconds)

[Pradeep Khosla - UC San Diego Office of the Chancellor - University ...](#)  
chancellor.ucsd.edu/chancellor-khosla ▾  
Pradeep K. Khosla became UC San Diego's eighth Chancellor on August 1, 2012. As UC San Diego's chief executive officer, he leads a campus with more than ...

[Pradeep K. Khosla - UC San Diego Office of the Chancellor](#)  
chancellor.ucsd.edu/chancellor-khosla/khosla-biography ▾  
Chancellor, University of California San Diego. Pradeep K. Khosla, an internationally renowned electrical and computer engineer, is the eighth Chancellor of the ...

[Pradeep Khosla - Wikipedia](#)  
[https://en.wikipedia.org/wiki/Pradeep\\_Khosla](https://en.wikipedia.org/wiki/Pradeep_Khosla) ▾  
Pradeep K. Khosla is an academic computer scientist and university administrator. He is the current chancellor of the University of California, San Diego. He was ...

[Pradeep Khosla | LinkedIn](#)  
<https://www.linkedin.com/in/pradeepkholst> ▾  
Greater San Diego Area - Chancellor, UC San Diego - Avigilon  
View Pradeep Khosla's professional profile on LinkedIn. LinkedIn is the world's largest business network, helping professionals like Pradeep Khosla discover ...

[Robotics Institute: Pradeep Khosla](#)  
[www.ri.cmu.edu](http://www.ri.cmu.edu) > people ▾



[More images](#)

**Pradeep Khosla**

Chancellor of the University of California, San Diego

Pradeep K. Khosla is an academic computer scientist and university administrator. He is the current chancellor of the University of California, San Diego. [Wikipedia](#)

**Born:** March 13, 1957 (age 60 years), Mumbai, India

**Spouse:** Thespine Kavoulakis

**Education:** Indian Institute of Technology Kharagpur, Carnegie Mellon University

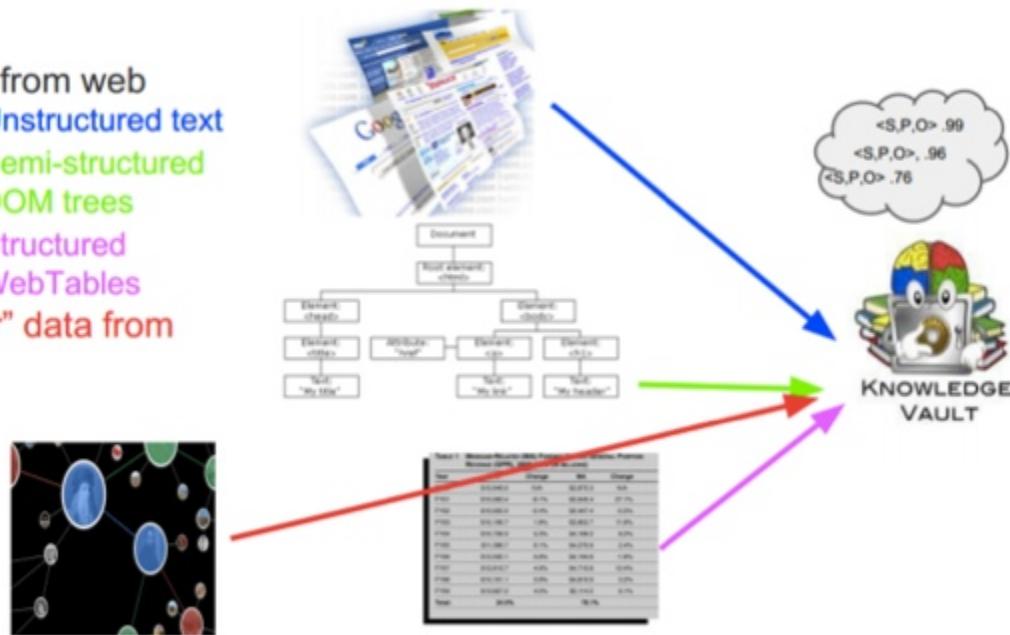
**Residence:** Audrey Geisel University House, La Jolla, CA

# How does Google know that?

# Large datasets + Machine learning!

Knowledge Vault\* fuses all these signals together

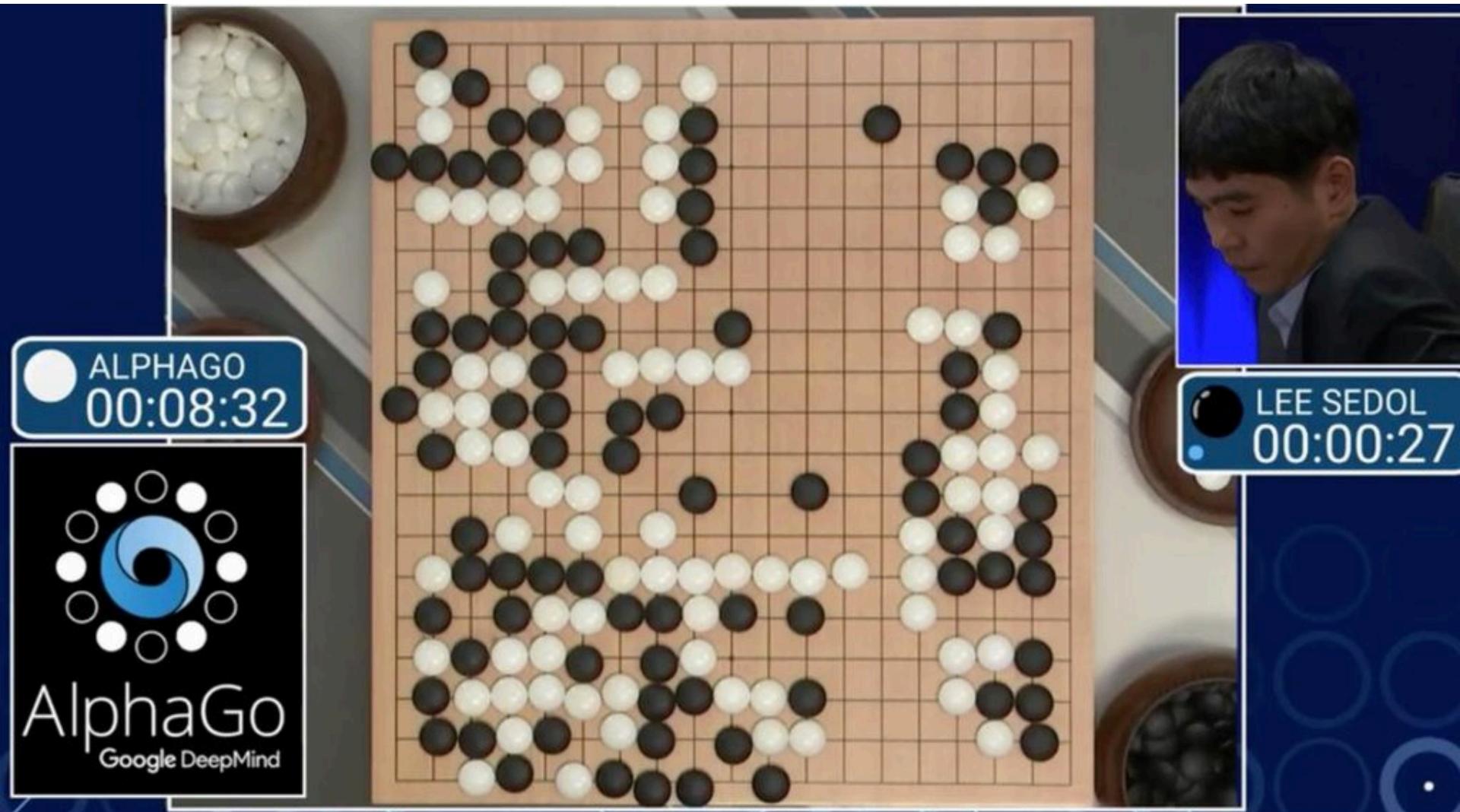
- Data from web
  - Unstructured text
  - Semi-structured DOM trees
  - Structured WebTables
- “Prior” data from FB



\* Details in a paper submitted to WWW'14 (Dong et al)

Knowledge Base Construction (KBC) process extracts tabular/relational data from large amounts of text data

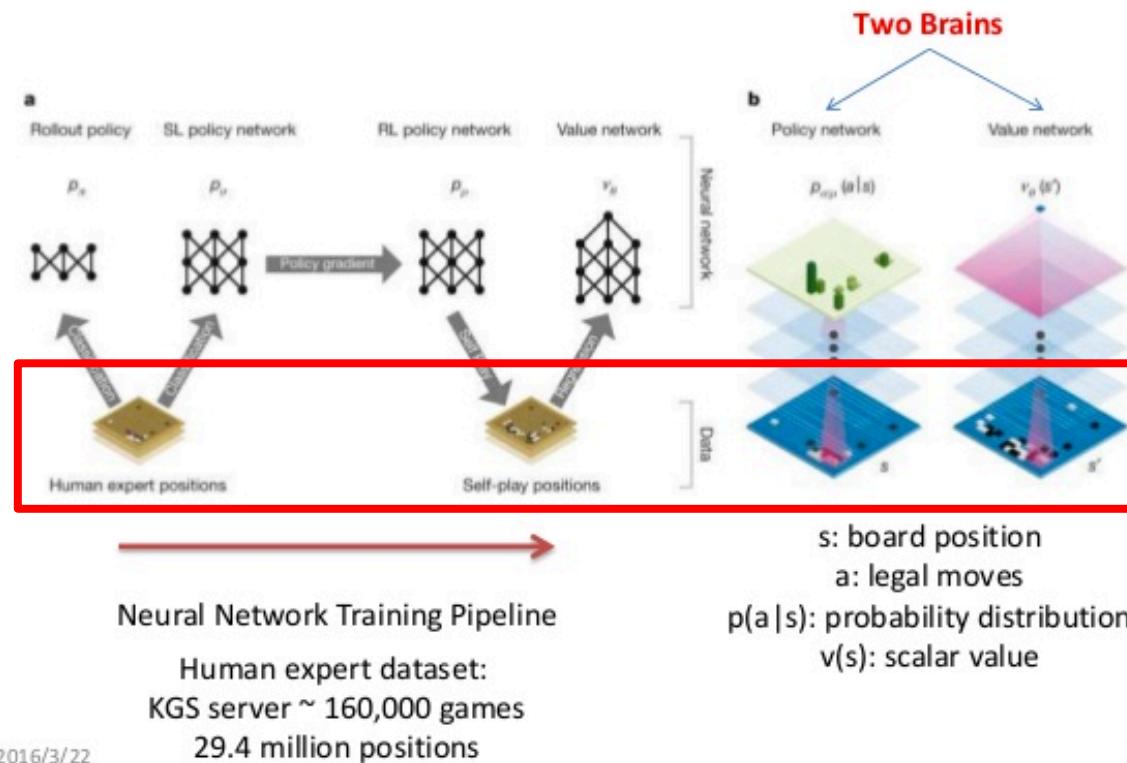
### 3. AlphaGo defeats human champion!



# How did AlphaGo achieve that?

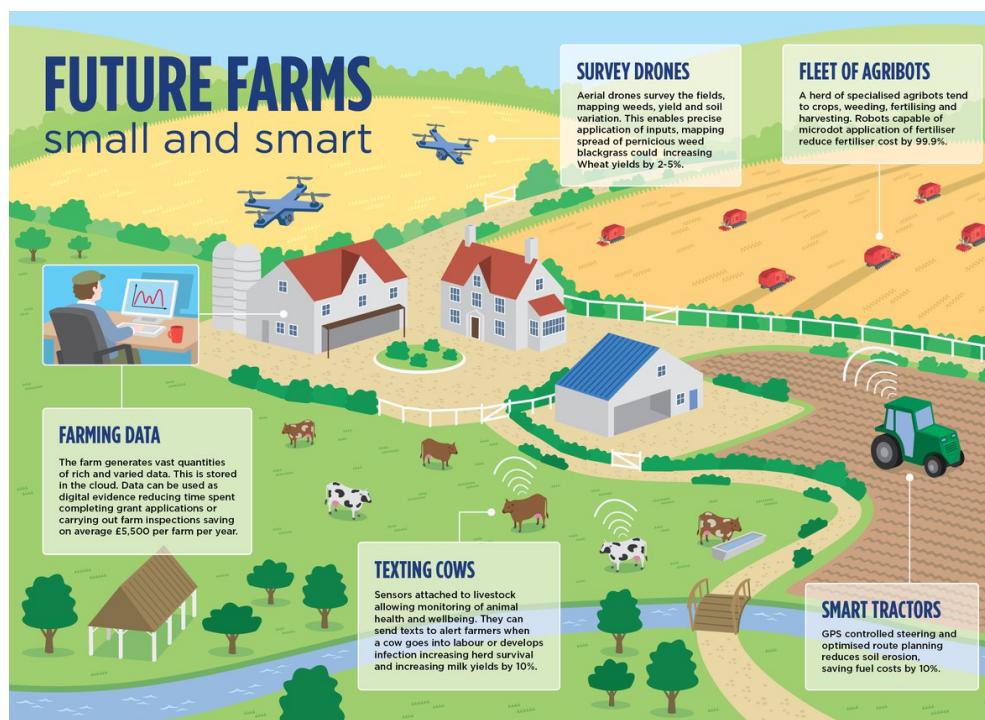
# Breakthrough powered by deep learning!

## Architecture of AlphaGo

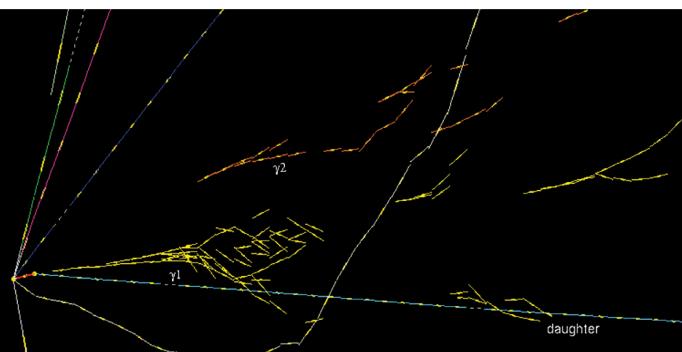
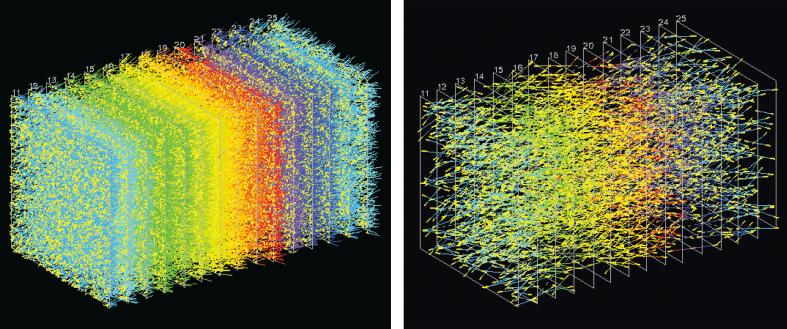


Deep CNNs to visually process board status in plays

# Innumerable “enterprise” applications

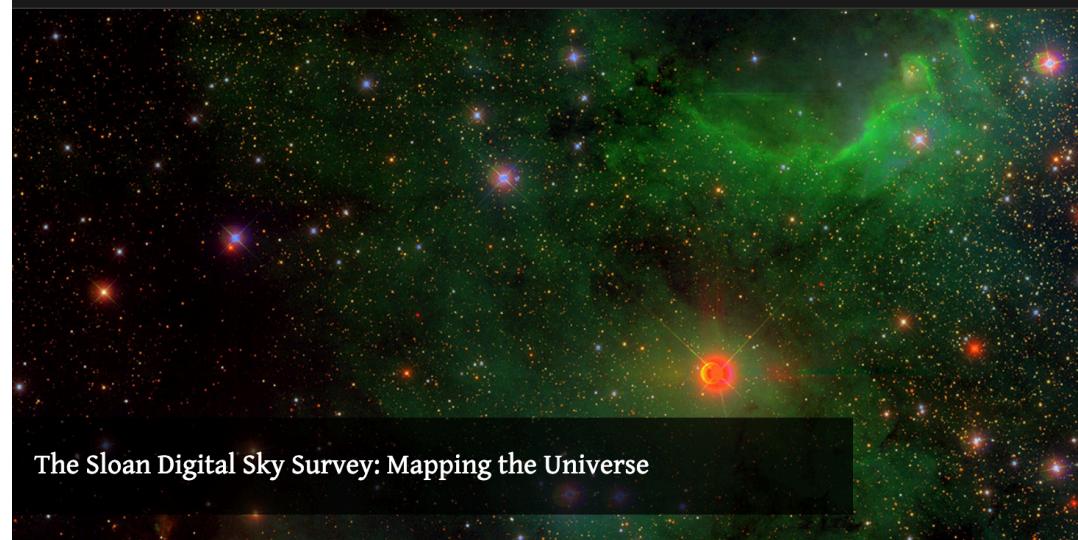


“Domain sciences” and healthcare tech  
are also becoming data+ML intensive

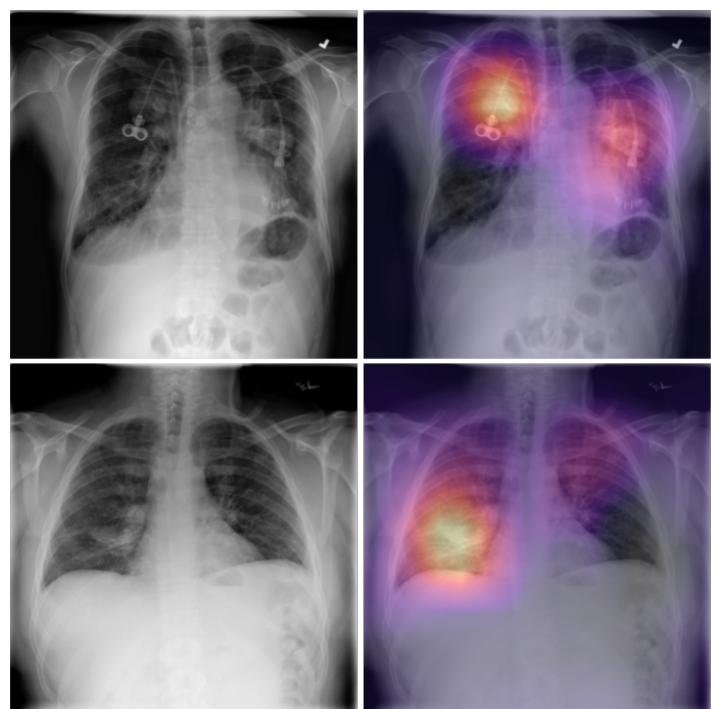


This is Data Release 16.

Data Surveys Instruments



The Sloan Digital Sky Survey: Mapping the Universe



**Software systems for data analytics and  
ML over large and complex datasets are  
now critical for digital applications in  
many domains**

# The Age of “Big Data”/“Data Science”

## The New York Times

SundayReview | NEWS ANALYSIS

The Age **Forbes** / Entrepreneurs

By STEVE LOHR F

MAR 25, 2015 @ 7:33 PM 4,407 VIEWS



Email



Share



Tweet



Save

Josh Steimle, CON

For roughly a decade, information about Big Data. The IDC industry will experience by 2018. What this

# Forbes

## Drowning In Big Data - Finding Insight In A Digital DATA Josh Steimle, CON by Thomas H. Davenport and D.J. Patil FROM THE OCTOBER 2012 ISSUE

SUMMARY   SAVE   SHARE   COMMENT 5   TEXT SIZE   PRINT   BUY COPIES \$8.95



**Harvard  
Business  
Review**

**W**hen Jonathan Goldman arrived for work in June 2006 at LinkedIn, the business networking site, the place still felt like a start-up. The company had just under 8 million accounts, and the number was growing quickly as existing members invited their friends and colleagues to join. But users weren't seeking out connections with the people who were already on the site at the rate executives had expected. Something was apparently missing in the social experience. As one LinkedIn manager put it, "It was like arriving at a conference reception and realizing you don't know anyone. So you just stand in the corner sipping your drink—

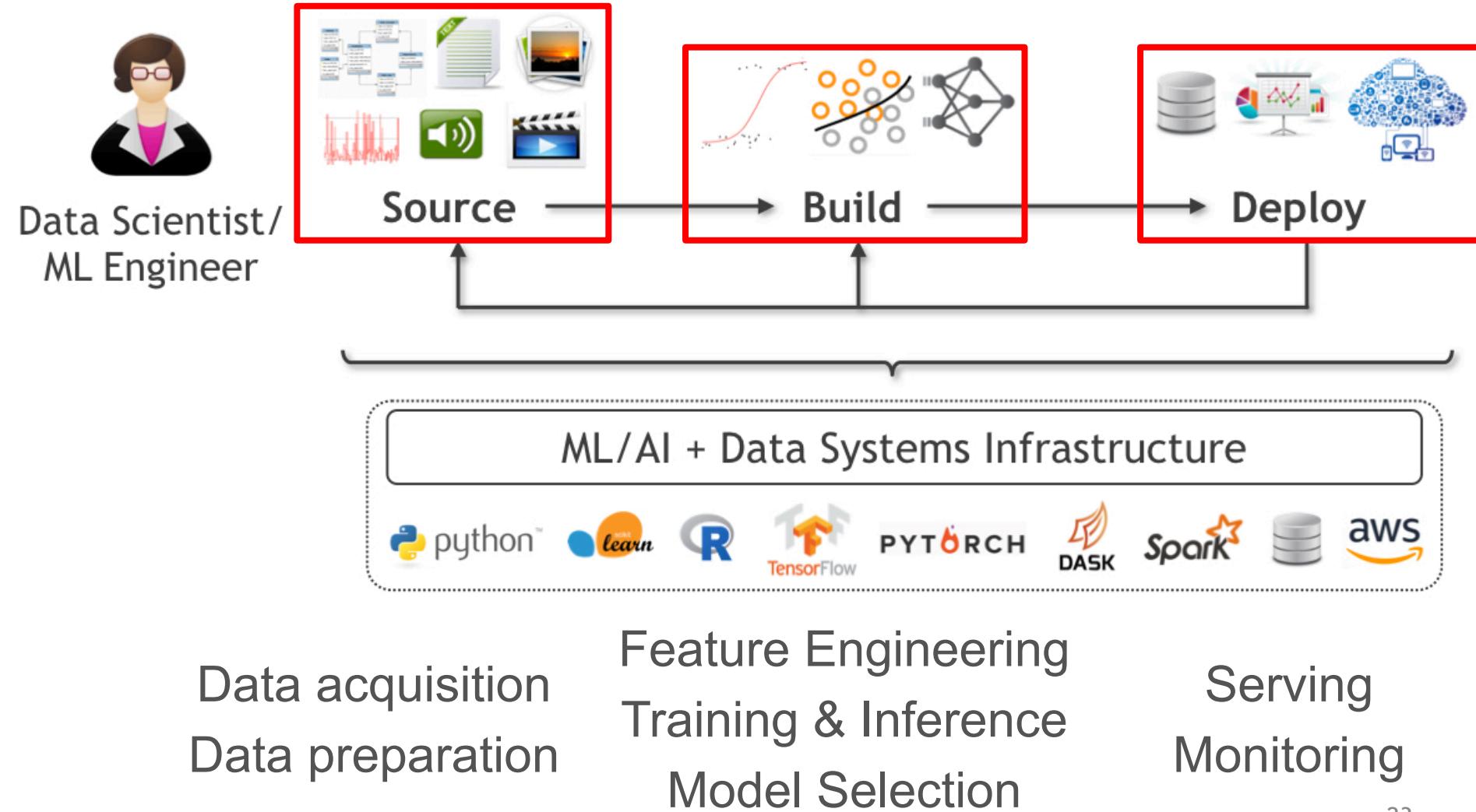
*Data data everywhere,  
All the wallets did shrink!*

*Data data everywhere,  
Nor any moment to think?*

# DSC 102 will get you thinking about the fundamentals of scalable analytics systems

1. “**Systems**”: What resources does a computer have?  
How to store and efficiently compute over large data?  
What is cloud?
2. “**Scalability**”: How to scale and parallelize data-intensive computations?
3. For “**Analytics**”:
  - 3.1. **Source**: Data acquisition & preparation for ML
  - 3.2. **Build**: Model selection & deep learning systems
  - 3.3. **Deploying** ML models
4. Hands-on experience with scalable analytics tools

# The Lifecycle of ML-based Analytics



# ML Systems

*Q: What is a Machine Learning (ML) System?*

- ❖ A data processing system (aka *data system*) for mathematically advanced data analysis operations (inferential or predictive):
  - ❖ Statistical analysis; ML, deep learning (DL); data mining (domain-specific applied ML + feature eng.)
  - ❖ *High-level APIs* to express ML computations over (large) datasets
  - ❖ *Execution engine* to run ML computations efficiently

# Categorizing ML Systems

## ❖ Orthogonal Dimensions of Categorization:

1. **Scalability:** In-memory libraries vs Scalable ML system (works on larger-than-memory datasets)
2. **Target Workloads:** General ML library vs Decision tree-oriented vs Deep learning, etc.
3. **Implementation Reuse:** Layered on top of scalable data system vs Custom from-scratch framework

# Major Existing ML Systems

## General ML libraries:

In-memory:



Disk-based files:



Layered on RDBMS/Spark:



Cloud-native:



Azure Machine Learning



Amazon SageMaker

“AutoML” platforms:



DataRobot



Decision tree-oriented:



Microsoft  
LightGBM

Deep learning-oriented:



TensorFlow



# Data Systems Concerns in ML

**Key concerns in ML:**

Q: How do “ML Systems” relate to ML?

Runtime efficiency (sometimes)

**Additional key *practical* concerns in ML Systems:**

ML Systems : ML :: Computer Systems : TCS  
Scalability (and efficiency at scale)

Usability

Manageability

Developability

*Long-standing  
concerns in the  
**DB systems**  
world!*

Q: ~~Q: What are the difficulties in building large-scale machine learning systems?~~

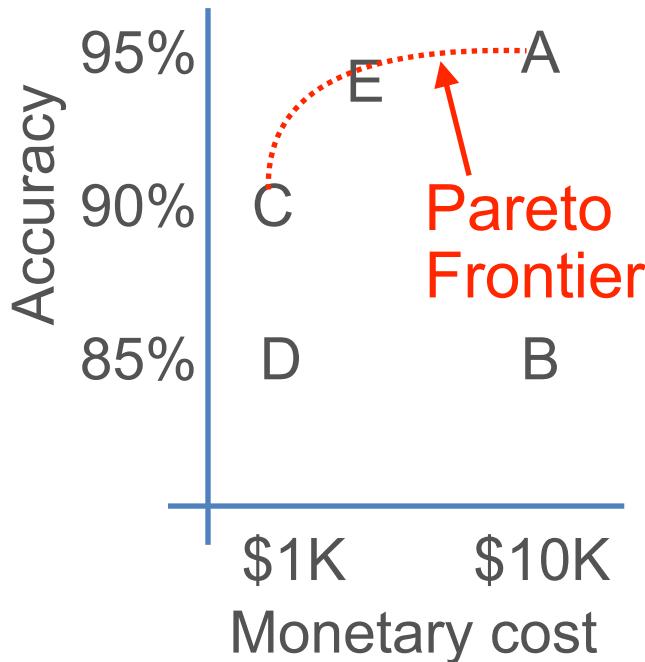
# Conceptual System Stack Analogy

	Relational DB Systems	ML Systems
Theory	First-Order Logic Complexity Theory	Learning Theory Optimization Theory
Program Formalism	Relational Algebra	Matrix Algebra Gradient Descent
Program Specification	SQL	TensorFlow? Scikit-learn?
Program Modification	Query Optimization	???
Execution Primitives	Parallel Relational Operator Dataflows	Depends on ML Algorithm
Hardware	CPU, GPU, FPGA, NVM, RDMA, etc.	

# Real-World ML: Pareto Surfaces

**Q:** Suppose you are given ad click-through prediction models A, B, C, and D with accuracies of 95%, 85%, 90%, and 85%, respectively. Which one will you pick?

**Q:** What about now?



- ❖ Real-world ML users must grapple with multi-dimensional *Pareto surfaces*: accuracy, monetary cost, training time, scalability, inference latency, tool availability, interpretability, fairness, etc.
- ❖ *Multi-objective optimization* criteria set by application needs / business policies.

# Learning Outcomes of this course

- ❖ *Explain* the basic principles of the memory hierarchy, parallelism paradigms, scalable data systems, cloud computing, and containerization.
- ❖ *Identify* the abstract data access patterns of, and opportunities for parallelism and efficiency gains in, data processing and ML algorithms at scale.
- ❖ *Outline* how to use cluster and cloud services, dataflow (“Big Data”) programming with MapReduce and Spark, and deep learning inference with TensorFlow and Keras.
- ❖ *Apply* the above programming skills to create end-to-end pipelines for data preparation, feature engineering, and model selection on large-scale datasets.
- ❖ *Reason* critically about practical tradeoffs between accuracy, efficiency, scalability, usability, and total cost.

# What this course is NOT about

- ❖ NOT a course on databases, relational model, or SQL
  - ❖ Take DSC 100 instead (pre-requisite)
- ❖ NOT a course on internal details of RDBMSs
  - ❖ Take CSE 132C instead
- ❖ NOT a training module for how to use Spark
- ❖ NOT a course on ML or data mining *algorithmics*;  
instead, we focus on ML *systems*

Now for the (boring) logistics ...

# Prerequisites

- ❖ **DSC 100** (or equivalent) is necessary
- ❖ Transitively **DSC 80**; a mainstream ML algorithmics course is necessary
- ❖ Proficiency in Python programming
- ❖ For all other cases, email the instructor with proper justification; a waiver can be considered

[http://cseweb.ucsd.edu/~arunkk/dsc102\\_winter22](http://cseweb.ucsd.edu/~arunkk/dsc102_winter22)

# Components and Grading

- ❖ **3 Programming Assignments: 35% (7% + 14% + 14%)**
  - ❖ No late days! Plan your work well ahead.
- ❖ **Best 5 of 6 Surprise Quizzes: 15%; in-class using iClicker**
- ❖ **Midterm Exam: 15%**
  - ❖ **Wed, Feb 9; in-class (50min)**
- ❖ **Cumulative Final Exam: 35%**
  - ❖ **Mon, Mar 14; 180min**
- ❖ All quizzes and exams are *in-person* only; plan accordingly
- ❖ LMK ahead of time if you need makeup exam slot

# Grading Scheme

Hybrid of relative and absolute; grade is better of the two

Grade	Relative Bin (Use strictest)	Absolute Cutoff (>=)
A+	Highest 5%	95
A	Next 10% (5-15)	90
A-	Next 15% (15-30)	85
B+	Next 15% (30-45)	80
B	Next 15% (45-60)	75
B-	Next 15% (60-75)	70
C+	Next 5% (75-80)	65
C	Next 5% (80-85)	60
C-	Next 5% (85-90)	55
D	Next 5% (90-95)	50
F	Lowest 5%	< 50

Example: Score 82 but 33%ile; Rel.: B-; Abs.: B+; so, B+

# Tentative Course Schedule

Week	Topic
1-3	Systems Principles; Basics of Computer Org. and Operating Systems
4	Basics of Cloud Computing
5-6	Parallel and Scalable Data Processing: Parallelism
Scalability Principles	
7-8	Midterm Exam on Wed, Feb 9 Parallel and Scalable Data Processing: Scalability
8	Dataflow Systems
9-10	ML Data Sourcing
10	ML Model Building Systems
10	ML Deployment
11	Final Exam on Wed, Mar 17

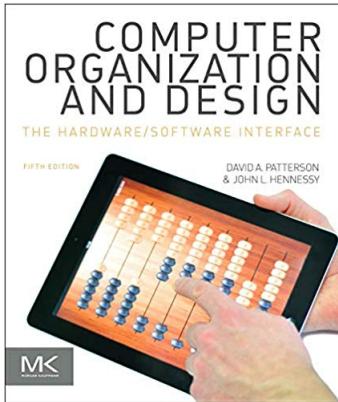
There will likely be 2 industry guest lectures; speakers TBD<sup>6</sup>

# Tentative Schedule for Prog. Assignments

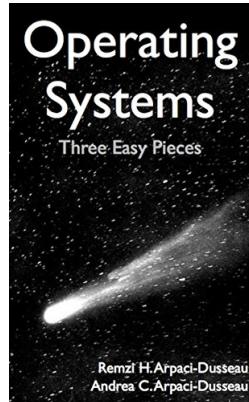
Date	Agenda
Jan 12	PA 0 released
Jan 14	Discussion on PA 0 by TA
Jan 25	<b>PA 0 due</b>
Jan 26	PA 1 released
Jan 28	Discussion on PA 1 by TA
Feb 16	<b>PA 1 due</b>
Feb 17	PA 2 released
Feb 18	Discussion on PA 2 by TA
Mar 8	<b>PA 2 due</b>

**No late days!** Plan your work upfront!  
Try to join the Discussion slot talks by the TAs.

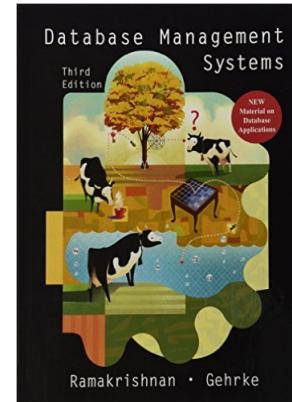
# Suggested Textbooks



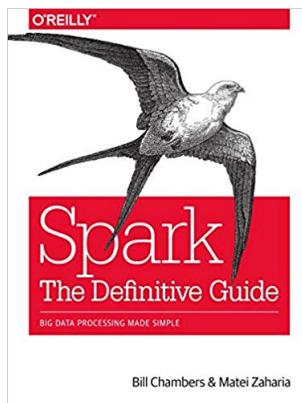
Aka “CompOrg Book”



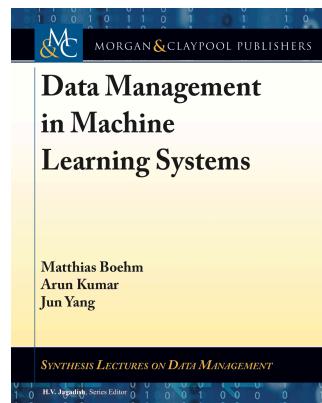
Aka “Comet Book”



Aka “Cow Book”



Aka “Spark Book”

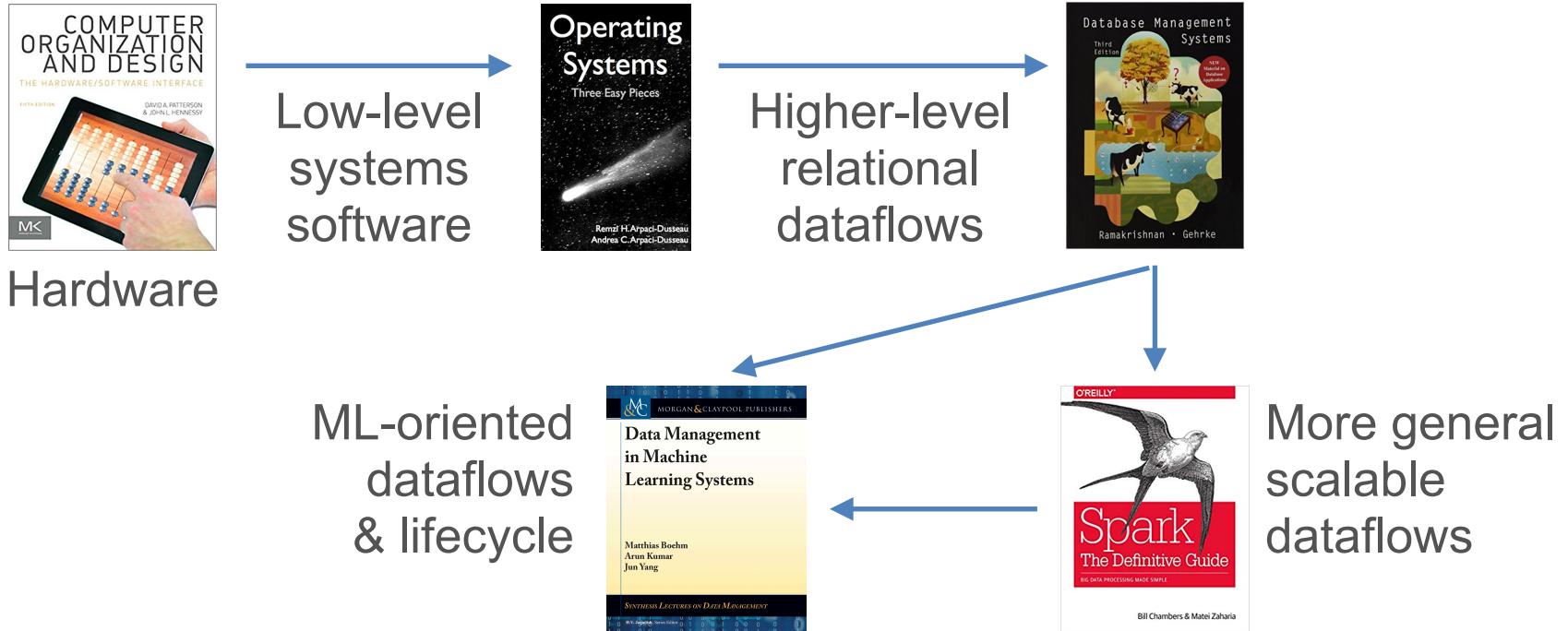


Aka “MLSys Book”

(Free PDFs available online; also check out our library)

# Why so many textbooks?!

1. Computer systems are about carefully layering *levels of abstraction*.



2. Analytics/ML Systems is a recent/emerging area of research.
3. Also, DSC 102 is the first UG course of its kind in the world!

# Course Administrivia

- ❖ **Lectures:** MWF 1:00pm-1:50pm PT
  - ❖ Virtual-only for weeks 1 and 2; in-person only afterward
  - ❖ Will take live Q&A throughout
- ❖ **Instructor:** Arun Kumar; arunkk [at] eng.ucsd.edu
  - ❖ Office hours: Wed 2:00-3:00pm PT
- ❖ See **Piazza** (or Canvas) for all Zoom meeting links, logistical announcements; see Canvas for gradebook
- ❖ **TAs:** Umesh Singla, Vignesh Nandakumar, and Pradyumna Sridhara (see webpage for details)

[http://cseweb.ucsd.edu/~arunkk/dsc102\\_winter22](http://cseweb.ucsd.edu/~arunkk/dsc102_winter22)

# General Dos and Do NOTs

## ***Do:***

- ❖ Follow all announcements on Piazza/Canvas
- ❖ Try to join the lectures/discussions live
- ❖ View/review videos asynchronously by yourself
- ❖ Participate in discussions in class / on Piazza
- ❖ Raise your hand before speaking
- ❖ Use “DSC102:” as subject prefix for all emails to me/TA

## ***Do NOT:***

- ❖ Record lectures on your side without permission of instructor and other students
- ❖ Harass, intimidate, or intentionally talk over others
- ❖ Violate academic integrity on the graded quizzes, exams, or programming assignments; I am *very strict* on this matter!

# DSC 102

# Systems for Scalable Analytics

Arun Kumar

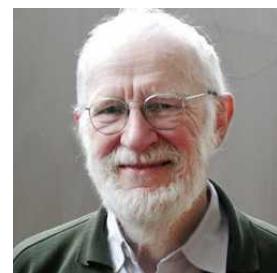
Topic 1: Basics of Machine Resources  
Part 1: Computer Organization

Ch. 1, 2.1-2.3, 2.12, 4.1, and 5.1-5.5 of CompOrg Book

***Q: What is a computer?***

A programmable electronic device that can store, retrieve, and process digital **data**.

Computer science aka “Datalogy”

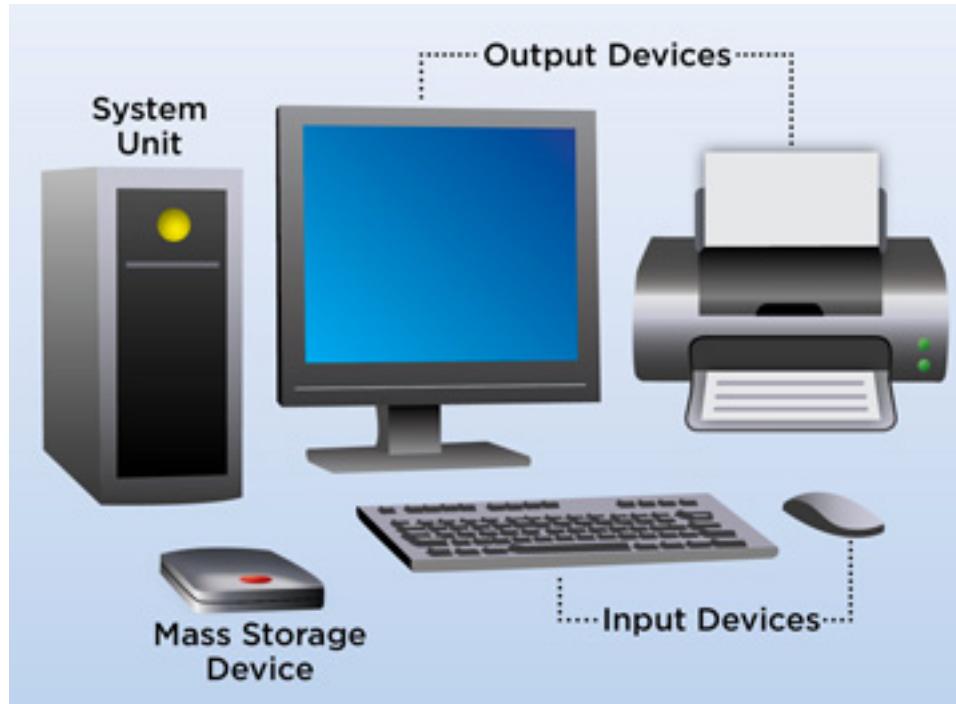


Peter Naur

# Outline

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

# Parts of a Computer



## **Hardware:**

The electronic machinery  
(wires, circuits, transistors,  
capacitors, devices, etc.)

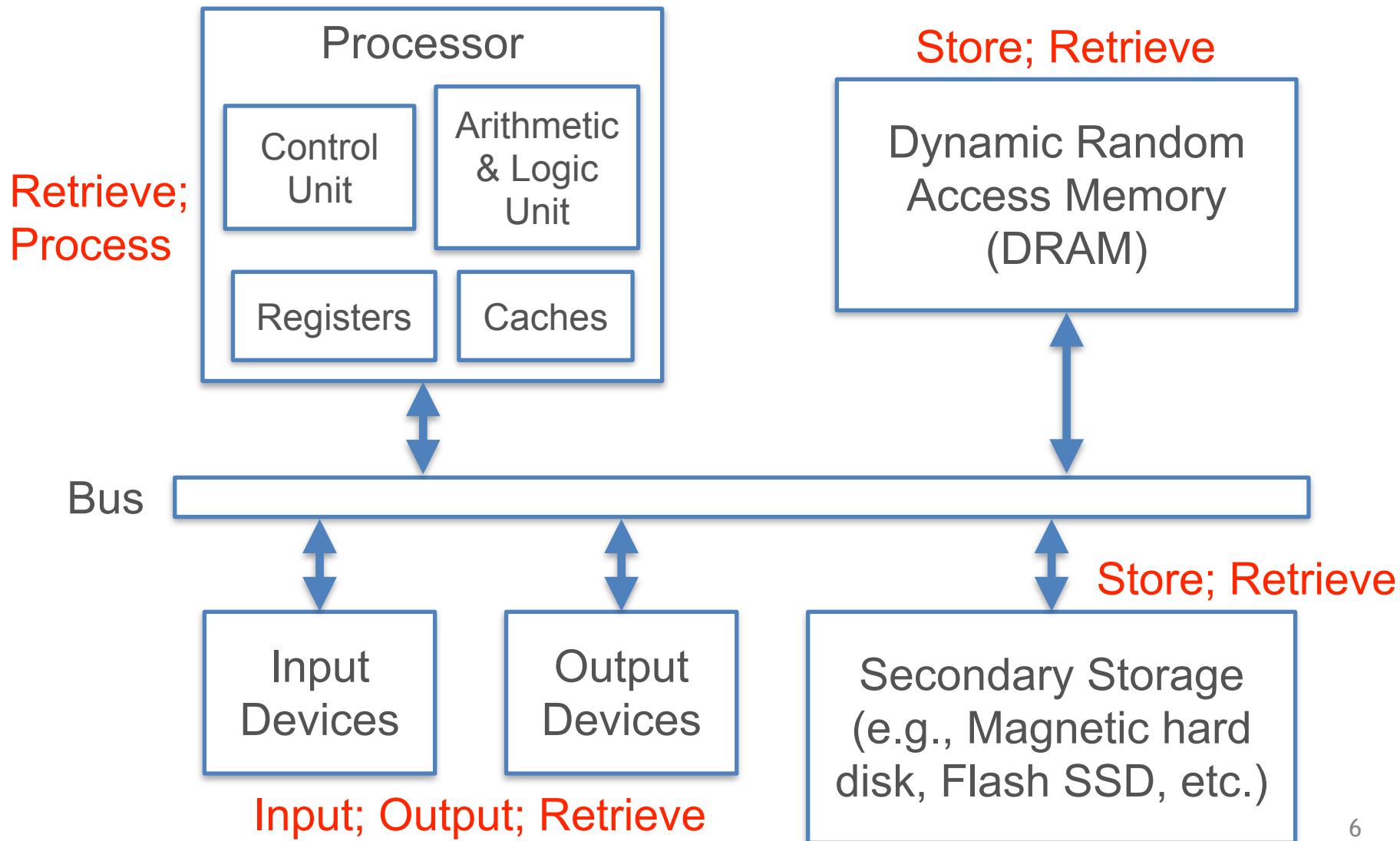
## **Software:**

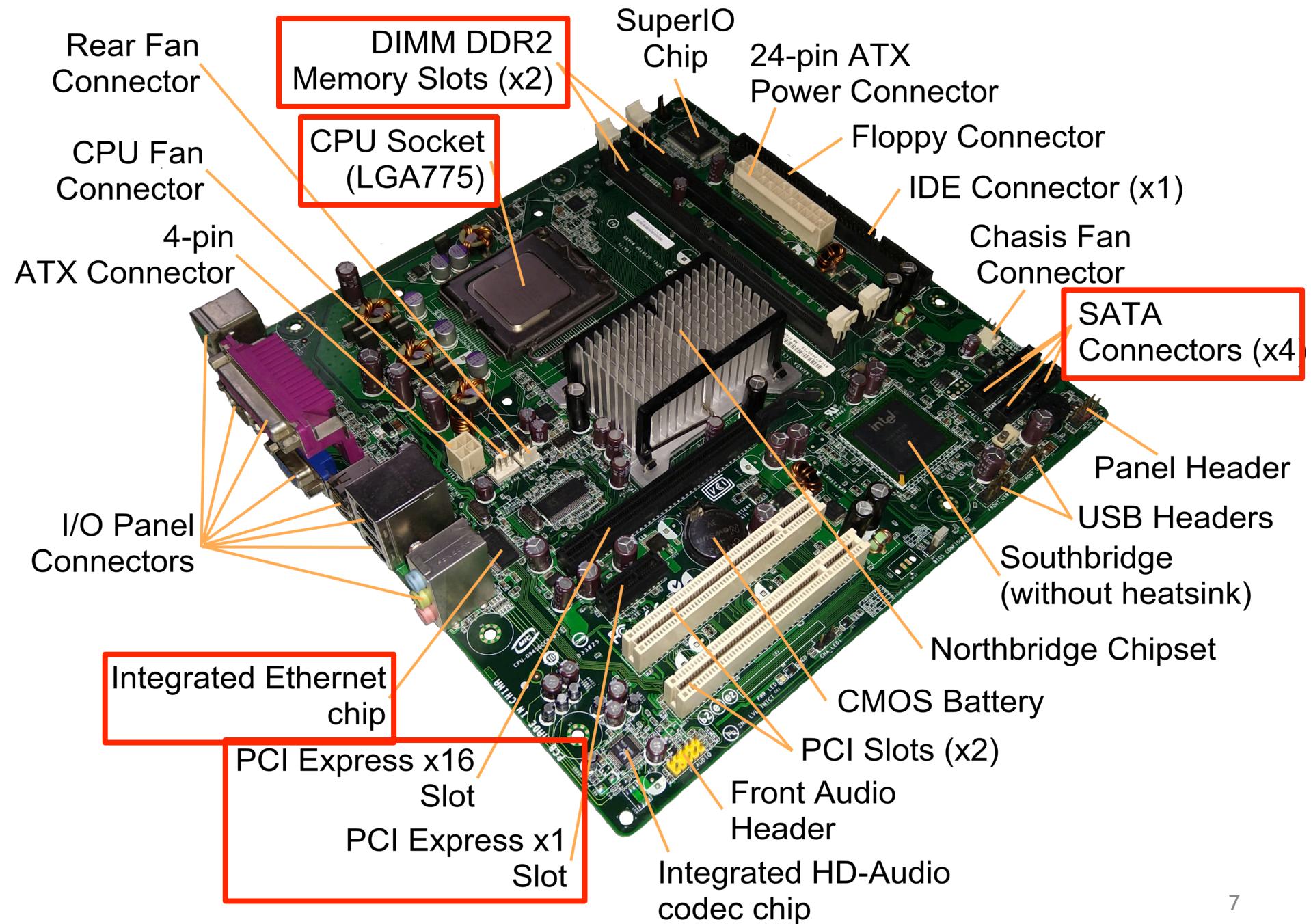
Programs (instructions)  
and data

# Key Parts of Computer Hardware

- ❖ **Processor** (CPU, GPU, etc.)
  - ❖ Hardware to orchestrate and execute *instructions* to manipulate *data* as specified by a *program*
- ❖ **Main Memory** (aka Dynamic Random Access Memory)
  - ❖ Hardware to store *data* and *programs* that allows very fast location/retrieval; byte-level *addressing* scheme
- ❖ **Disk** (aka secondary/persistent storage)
  - ❖ Similar to memory but *persistent*, *slower*, and higher capacity / cost ratio; various addressing schemes
- ❖ **Network interface controller (NIC)**
  - ❖ Hardware to send data to / retrieve data over network of interconnected computers/devices

# Abstract Computer Parts and Data





# Key Aspects of Software

- ❖ **Instruction**
  - ❖ A command understood by hardware; finite vocabulary for a processor: Instruction Set Architecture (ISA); bridge between hardware and software
- ❖ **Program (aka code)**
  - ❖ A collection of instructions for hardware to execute
- ❖ **Programming Language (PL)**
  - ❖ A human-readable *formal* language to write programs; at a much higher level of *abstraction* than ISA
- ❖ **Application Programming Interface (API)**
  - ❖ A set of functions (“interface”) exposed by a program/set of programs for use by humans/other programs
- ❖ **Data**
  - ❖ Digital representation of *information* that is stored, processed, displayed, retrieved, or sent by a program

# Main Kinds of Software

- ❖ **Firmware**

- ❖ Read-only programs “baked into” a device to offer basic hardware control functionalities

- ❖ **Operating System (OS)**

- ❖ Collection of interrelated programs that work as an intermediary platform/service to enable application software to use hardware more effectively/easily
  - ❖ Examples: Linux, Windows, MacOS, etc.

- ❖ **Application Software**

- ❖ A program or a collection of interrelated programs to manipulate data, typically designed for human use
  - ❖ Examples: Excel, Chrome, PostgreSQL, etc.

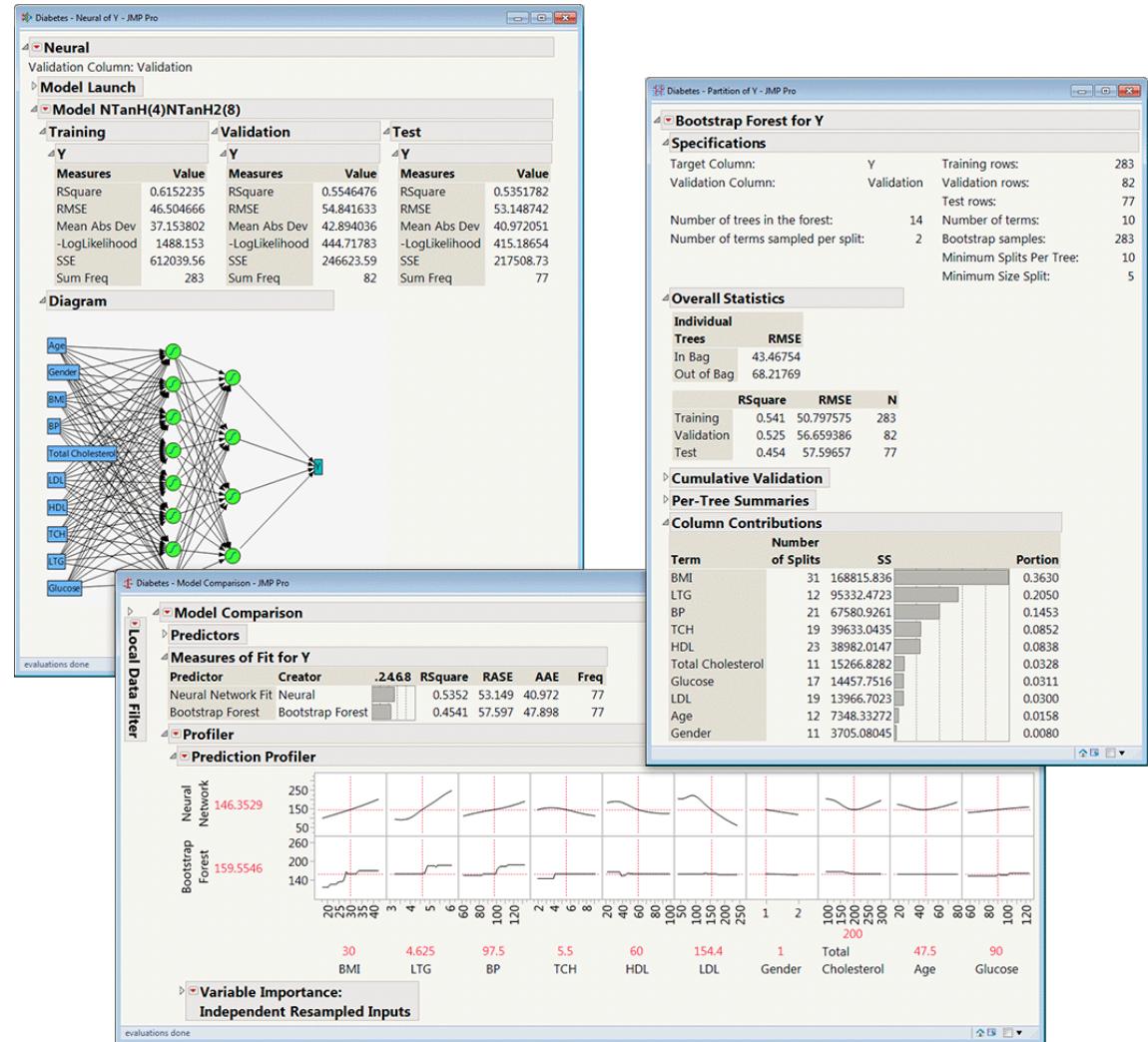
# Outline

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

*Q: But why bother learning such low-level computer sciencey stuff in Data Science?*

# Luxury of “Statisticians”/“Analysts” of Yore

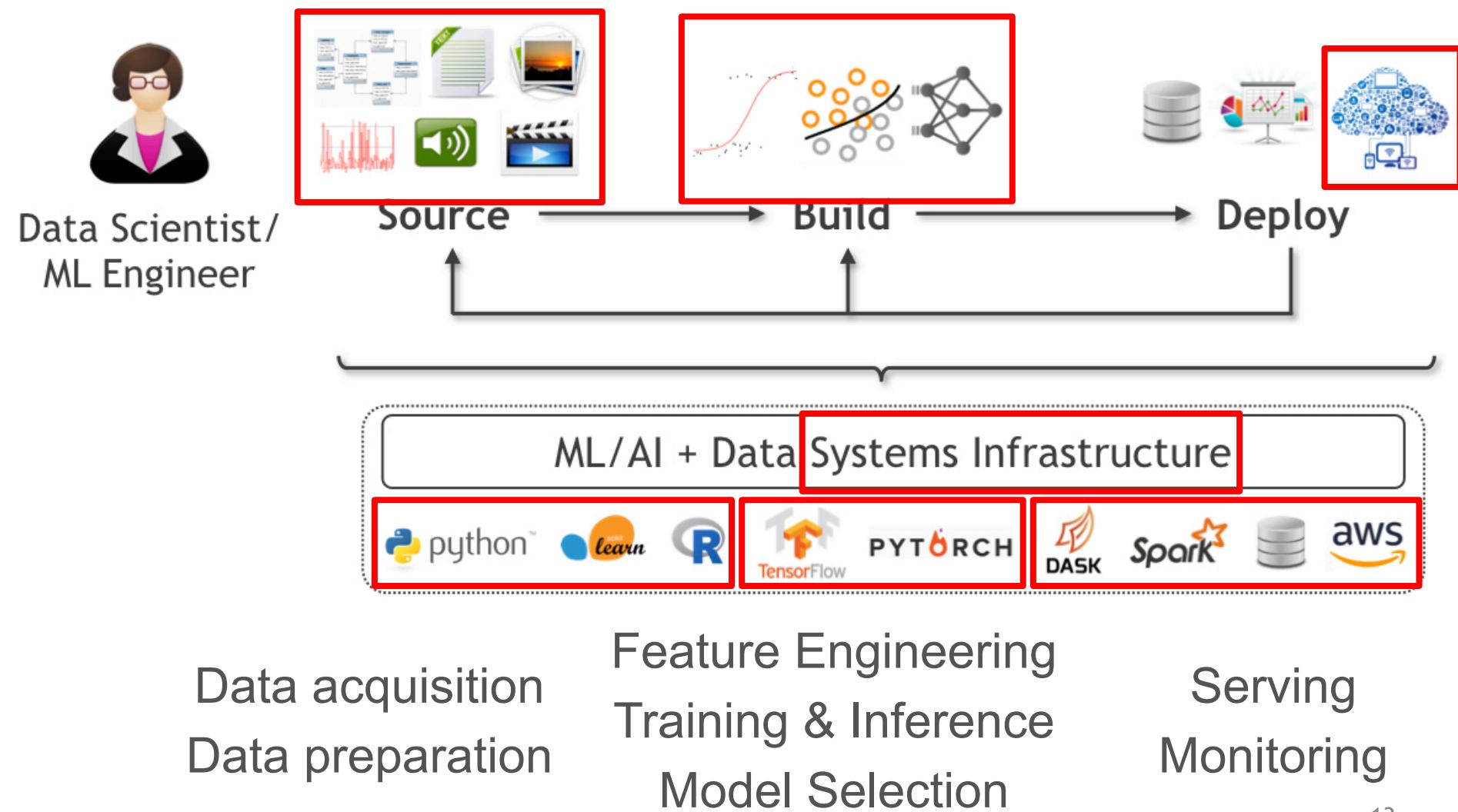
- ❖ **Methods:** Sufficed to learn just math/stats, maybe some SQL
- ❖ **Types:** Mostly tabular (relational), maybe some time series
- ❖ **Scale:** Mostly small (KBs to few GBs)
- ❖ **Tools:** Simple GUIs for both analysis and deployment; maybe an R-like console



[https://www.jmp.com/en\\_au/offers/jmp-pro-for-academic-research.html](https://www.jmp.com/en_au/offers/jmp-pro-for-academic-research.html)

<https://www.technologymagazine.com/data-and-data-analytics/sas-tops-worldwide-advanced-and-predictive-analytics-market-share>

# Reality of Today's “Data Scientists”



# Why bother with these in Data Science?

- ❖ Basics of Computer Organization
    - ❖ Digital Representation of Data
    - ❖ Processors and Memory Hierarchy
  - ❖ Basics of Operating Systems
    - ❖ 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 is evolving fast



statistician

Location



## Statistician Salaries United States ▾

Overview

Salaries

Interviews

Insights

Career Path

### How much does a Statistician make?

Updated Jan 4, 2022

Industry

Employer Size

Experience

All industries

All company sizes

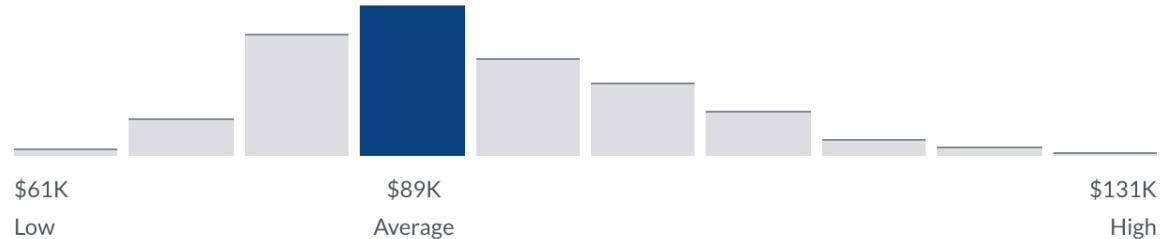
All years of Experience

Very High Confidence

**\$88,989** /yr

Average Base Pay

2,398 salaries





## Data Scientist Salaries United States ▾

Overview

Salaries

Interviews

Insights

Career Path

## How much does a Data Scientist make?

Updated Jan 4, 2022

Industry

All industries

Employer Size

All company sizes

Experience

All years of Experience

To filter salaries for Data Scientist, [Sign In](#) or [Register](#).

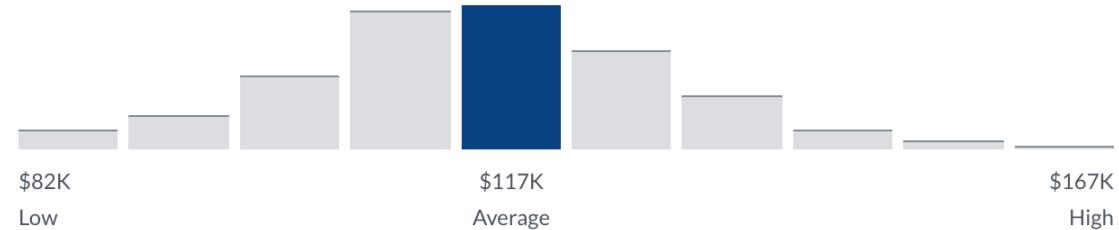


Very High Confidence

**\$117,212** /yr

Average Base Pay

18,354 salaries



— 88,989

= 28,223!

# Outline

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

***Q: What is data?***

本工事は、主に土木工事と機械工事の二種類に分かれます。

土木工事は、河川改修工事や道路工事など、地盤改良や構造物の建設を行います。

機械工事は、機械設備の設置や修理、運転管理などを担当します。

また、本工事では、機械の運転技術や作業効率を向上させるため、最新の機械や技術を導入する取り組みも実施されています。

このように、本工事は、多方面からの取り組みで、複数の専門知識が求められる重要な工事です。

本工事は、主に土木工事と機械工事の二種類に分かれます。

土木工事は、河川改修工事や道路工事など、地盤改良や構造物の建設を行います。

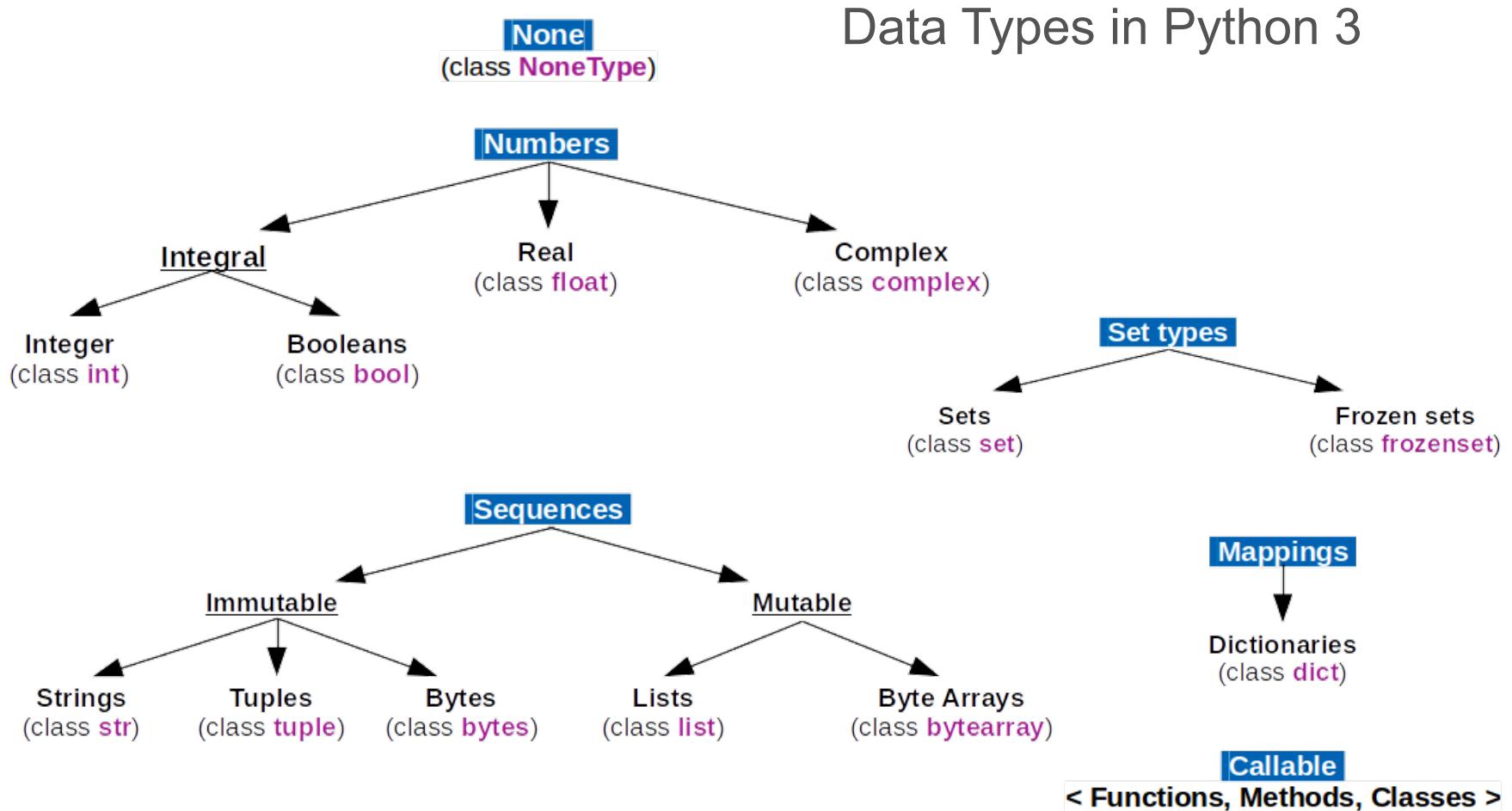
機械工事は、機械設備の設置や修理、運転管理などを担当します。

また、本工事では、機械の運転技術や作業効率を向上させるため、最新の機械や技術を導入する取り組みも実施されています。

# Digital Representation of Data

- ❖ **Bits:** All digital data are sequences of 0 & 1 (binary digits)
  - ❖ Amenable to high-low/off-on electromagnetism
  - ❖ Layers of *abstraction* to interpret bit sequences
- ❖ **Data type:** First layer of abstraction to interpret a bit sequence with a human-understandable category of information; interpretation fixed by the PL
  - ❖ Example common datatypes: Boolean, Byte, Integer, “floating point” number (Float), Character, and String
- ❖ **Data structure:** A second layer of abstraction to *organize* multiple instances of same or varied data types as a more complex object with specified properties
  - ❖ Examples: Array, Linked list, Tuple, Graph, etc.

# Digital Representation of Data



# Digital Representation of Data

- ❖ The size and *interpretation* of a data type depends on PL
- ❖ A **Byte** (B; 8 bits) is typically the basic unit of data types
- ❖ **Boolean:**
  - ❖ Examples in data sci.: Y/N or T/F responses
  - ❖ Just 1 bit needed but actual size is almost always 1B, i.e., 7 bits are wasted! (**Q: Why?**)
- ❖ **Integer:**
  - ❖ Examples in data science: #friends, age, #likes
  - ❖ Typically 4 bytes; many variants (short, unsigned, etc.)
  - ❖ Java *int* can represent  $-2^{31}$  to  $(2^{31} - 1)$ ; C *unsigned int* can represent 0 to  $(2^{32} - 1)$ ; Python3 *int* is effectively unlimited length (PL magic!)

# Digital Representation of Data

*Q: How many unique data items can be represented by 3 bytes?*

- ❖ Given  $k$  bits, we can represent  $2^k$  unique data items
- ❖ 3 bytes = 24 bits  $\Rightarrow 2^{24}$  items, i.e., 16,777,216 items
- ❖ Common approximation:  $2^{10}$  (i.e., 1024)  $\sim 10^3$  (i.e., 1000); recall kibibyte (KiB) vs kilobyte (KB) and so on

*Q: How many bits are needed to distinguish 97 data items?*

- ❖ For  $k$  unique items, invert the exponent to get  $\log_2(k)$
- ❖ But #bits is an integer! So, we only need  $\lceil \log_2(k) \rceil$
- ❖ So, we only need the next higher power of 2
- ❖ 97  $\rightarrow 128 = 2^7$ ; so, 7 bits

# Digital Representation of Data

**Q:** How to convert from decimal to binary representation?

- Given decimal n, if power of 2 (say,  $2^k$ ), put 1 at bit position k; if  $k=0$ , stop; else pad with trailing 0s till position 0
- If n is not power of 2, identify the power of 2 just below n (say,  $2^k$ ); #bits is then k; put 1 at position k
- Reset n as  $n - 2^k$ ; return to Steps 1-2
- Fill remaining positions in between with 0s

	7	6	5	4	3	2	1	0	Position/Exponent of 2
Decimal	128	64	32	16	8	4	2	1	Power of 2
$5_{10}$						1	0	1	
$47_{10}$				1	0	1	1	1	
$163_{10}$	1	0	1	0	0	0	1	1	
$16_{10}$				1	0	0	0	0	

**Q:** Binary to decimal?

# Digital Representation of Data

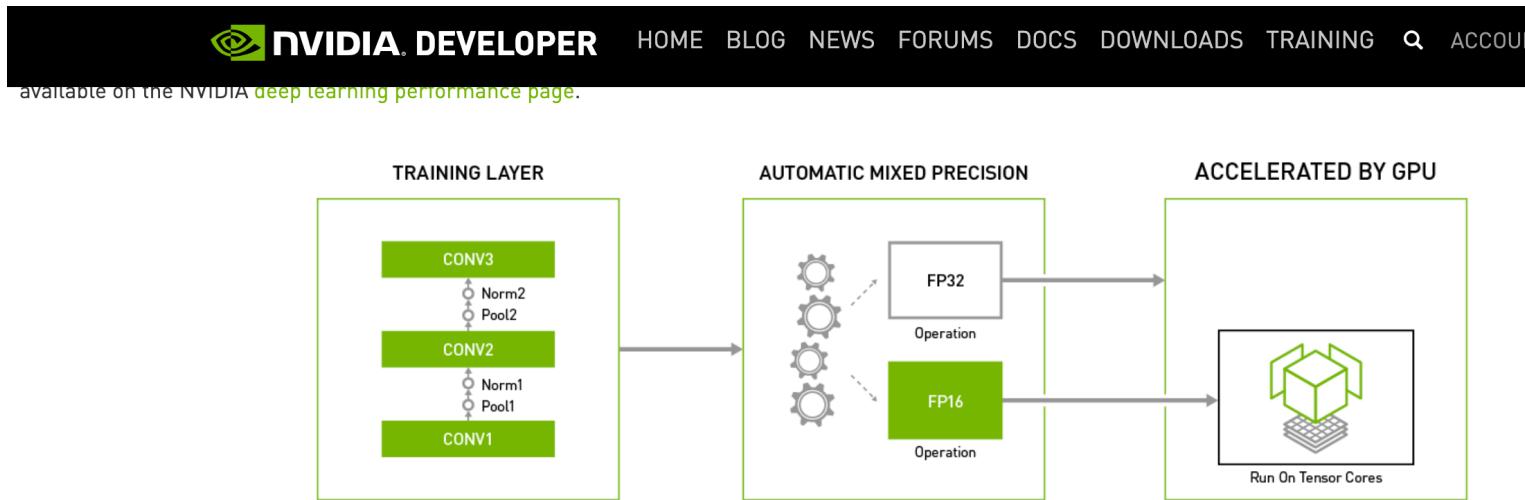
- ❖ *Hexadecimal* representation is a common stand-in for binary representation; more succinct and readable
  - ❖ Base 16 instead of base 2 cuts display length by ~4x
  - ❖ Digits are 0, 1, ... 9, A ( $10_{10}$ ), B, ... F ( $15_{10}$ )
  - ❖ From binary: combine 4 bits at a time from lowest

Decimal	Binary	Hexadecimal	
$5_{10}$	$101_2$	$5_{16}$	Alternative notations
$47_{10}$	$10\ 1111_2$	$2F_{16}$	
$163_{10}$	$1010\ 0011_2$	$A3_{16}$	$0xA3$ or $A3_H$
$16_{10}$	$1\ 0000_2$	$1\ 0_{16}$	

# Digital Representation of Data

- ❖ **Float:**

- ❖ Examples in data sci.: salary, scores, model weights
- ❖ IEEE-754 single-precision format is 4B long; double-precision format is 8B long
- ❖ Java and C *float* is single; Python *float* is double!

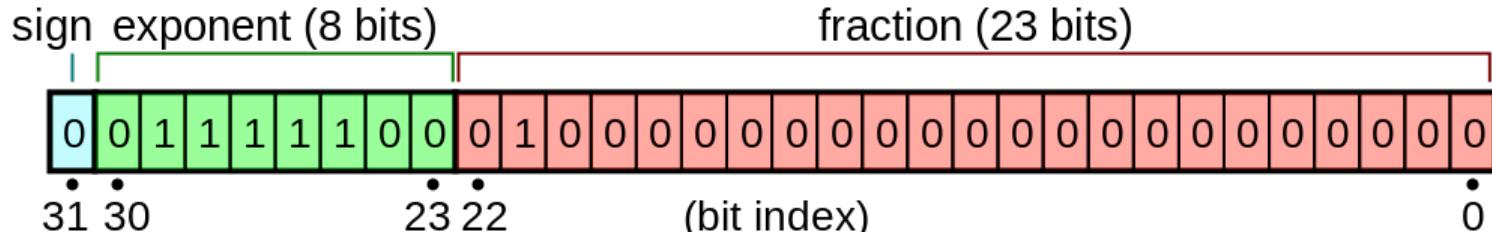


Using Automatic Mixed Precision for Major Deep Learning Frameworks

# Digital Representation of Data

- ❖ **Float:**

- ❖ Standard IEEE format for single (aka binary32):



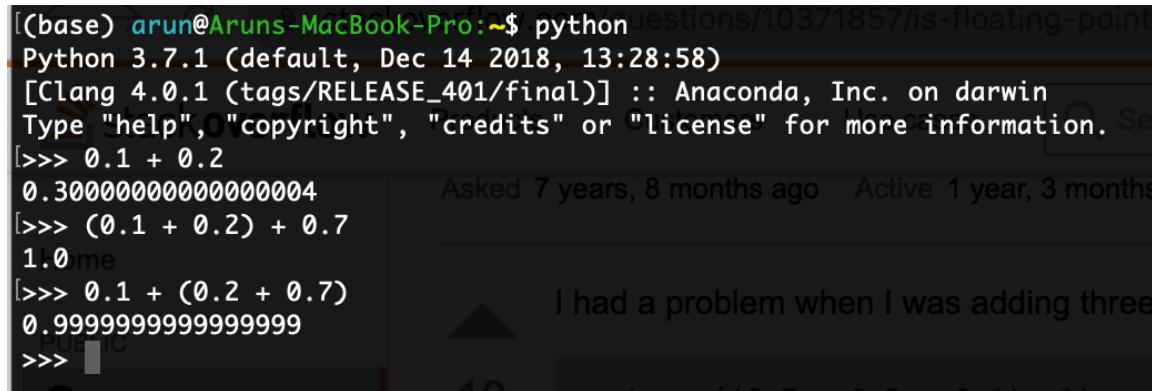
$$(-1)^{sign} \times 2^{exponent-127} \times \left(1 + \sum_{i=1}^{23} b_{23-i} 2^{-i}\right)$$

$$(-1)^0 \times 2^{124-127} \times \left(1 + 1 \cdot 2^{-2}\right) = (1/8) \times (1 + (1/4)) = 0.15625$$

(NB: Converting decimal reals/fractions to float is NOT in syllabus!)  
27

# Digital Representation of Data

- ❖ Due to representation imprecision issues, floating point arithmetic (addition and multiplication) is not associative!



The screenshot shows a terminal window running Python 3.7.1. The user types three expressions: 0.1 + 0.2, (0.1 + 0.2) + 0.7, and 0.1 + (0.2 + 0.7). The first expression returns 0.3000000000000004, while the second and third return 1.0. This illustrates that floating-point addition is not associative.

[(base) arun@Arun's-MacBook-Pro:~\$ python tests/10371857/is-floating-point  
Python 3.7.1 (default, Dec 14 2018, 13:28:58)  
[Clang 4.0.1 (tags/RELEASE\_401/final)] :: Anaconda, Inc. on darwin  
Type "help", "copyright", "credits" or "license" for more information. Se  
[>>> 0.1 + 0.2  
0.3000000000000004  
[>>> (0.1 + 0.2) + 0.7  
1.0me  
[>>> 0.1 + (0.2 + 0.7)  
0.9999999999999999  
>>>

- ❖ In binary32, special encodings recognized:
  - ❖ Exponent 0xFF and fraction 0 is +/- “Infinity”
  - ❖ Exponent 0xFF and fraction <> 0 is “NaN”
  - ❖ Max is ~  $3.4 \times 10^{38}$ ; min +ve is ~  $1.4 \times 10^{-45}$

# Digital Representation of Data

- ❖ More float standards: double-precision (float64; 8B) and half-precision (float16; 2B); different #bits for exponent, fraction
- ❖ Float16 is now common for *deep learning* parameters:
  - ❖ Native support in PyTorch, TensorFlow, etc.; APIs also exist for weight quantization/rounding post training
  - ❖ NVIDIA Deep Learning SDK support mixed-precision training; 2-3x speedup with similar accuracy!
- ❖ New processor hardware (FPGAs, ASICs, etc.) enable arbitrary precision, even 1-bit (!), but accuracy is lower

# Digital Representation of Data

- ❖ Representing **Character (char)** and **String**:
  - ❖ Represents letters, numerals, punctuations, etc.
  - ❖ A string is typically just a variable-sized array of char
  - ❖ C *char* is 1 byte; Java char is 2 bytes; Python does not have a char type (use *str* or *bytes*)
  - ❖ American Standard Code for Information Interchange (**ASCII**) for encoding characters; initially 7-bit; later extended to 8-bit
    - ❖ Examples: ‘A’ is 65, ‘a’ is 97, ‘@’ is 64, ‘!’ is 33, etc.
  - ❖ *Unicode UTF-8* is now most common; subsumes ASCII; 4 bytes for ~1.1 million “code points” incl. many other language scripts, math symbols, emojis, etc. ☺

# Digital Representation of Data

- ❖ All digital objects are *collections* of basic data types (bytes, integers, floats, and characters)
  - ❖ SQL dates/timestamp: string (w/ known format)
  - ❖ ML feature vector: *array* of floats (w/ known length)
  - ❖ Neural network weights: *set* of multi-dimensional *arrays* (matrices or tensors) of floats (w/ known dimensions)
  - ❖ Graph: an *abstract data type* (ADT) with *set* of vertices (say, integers) and *set* of edges (*pair* of integers)
  - ❖ Program in PL, SQL query: string (w/ grammar)
  - ❖ DRAM addresses: *array* of bytes (w/ known length)
  - ❖ Instruction in machine code: *array* of bytes (w/ ISA)
  - ❖ Other data structures or digital objects?

# Digital Representation of Data

## ❖ **Serialization and Deserialization:**

- ❖ A data structure often needs to be persisted (stored in a file) or transmitted over a network
- ❖ Serialization is the process of converting a data structure (or program objects in general) into a neat sequence of bytes that can be exactly recovered; deserialization is the reverse, i.e., bytes to data structure
- ❖ Serializing bytes and characters/strings is trivial
- ❖ 2 alternatives for serializing integers/floats:
  - ❖ As *byte stream* (aka “binary type” in SQL)
  - ❖ As *string*, e.g., 4B integer 5 -> 2B string as “5”
  - ❖ String ser. common in data science (CSV, TSV, etc.)

# Review Questions

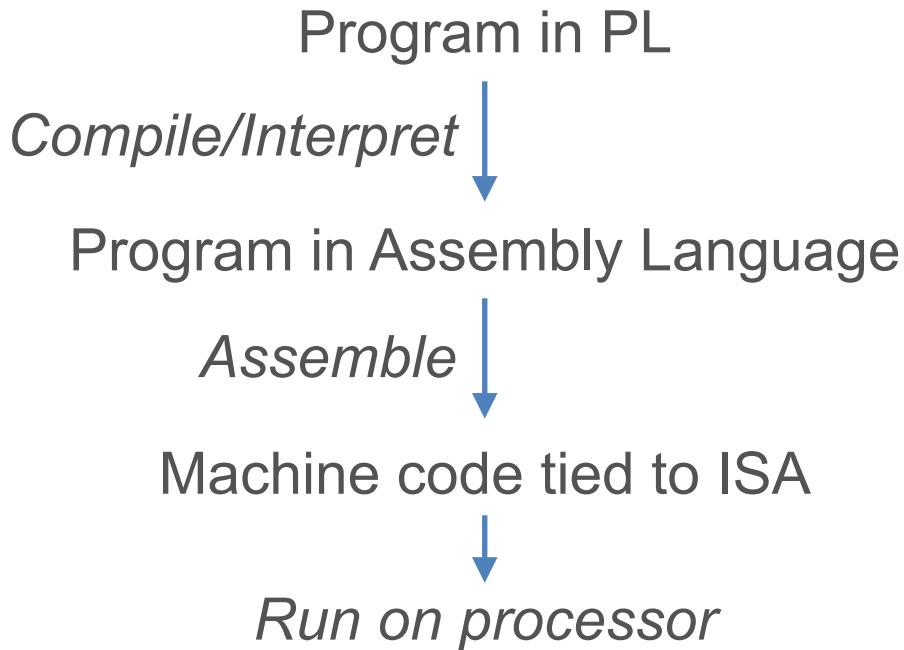
- ❖ What is the difference between data and code?
- ❖ What kind of software is TensorFlow? Linux?
- ❖ Why do computers use binary numbers?
- ❖ What is a byte?
- ❖ How many integers can you represent with 5 bits?
- ❖ How many bits do you need to represent 5 integers?
- ❖ What is the hexadecimal representation of  $20_{10}$ ?
- ❖ Why is a floating point standard needed?
- ❖ Why should a data scientist know about float formats?
- ❖ What does “lower precision” mean for a float weight in DL?
- ❖ Why is serialization needed on a computer?
- ❖ Is code a string? Is a string code?
- ❖ Is reality a computer simulation? :)

# Outline

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

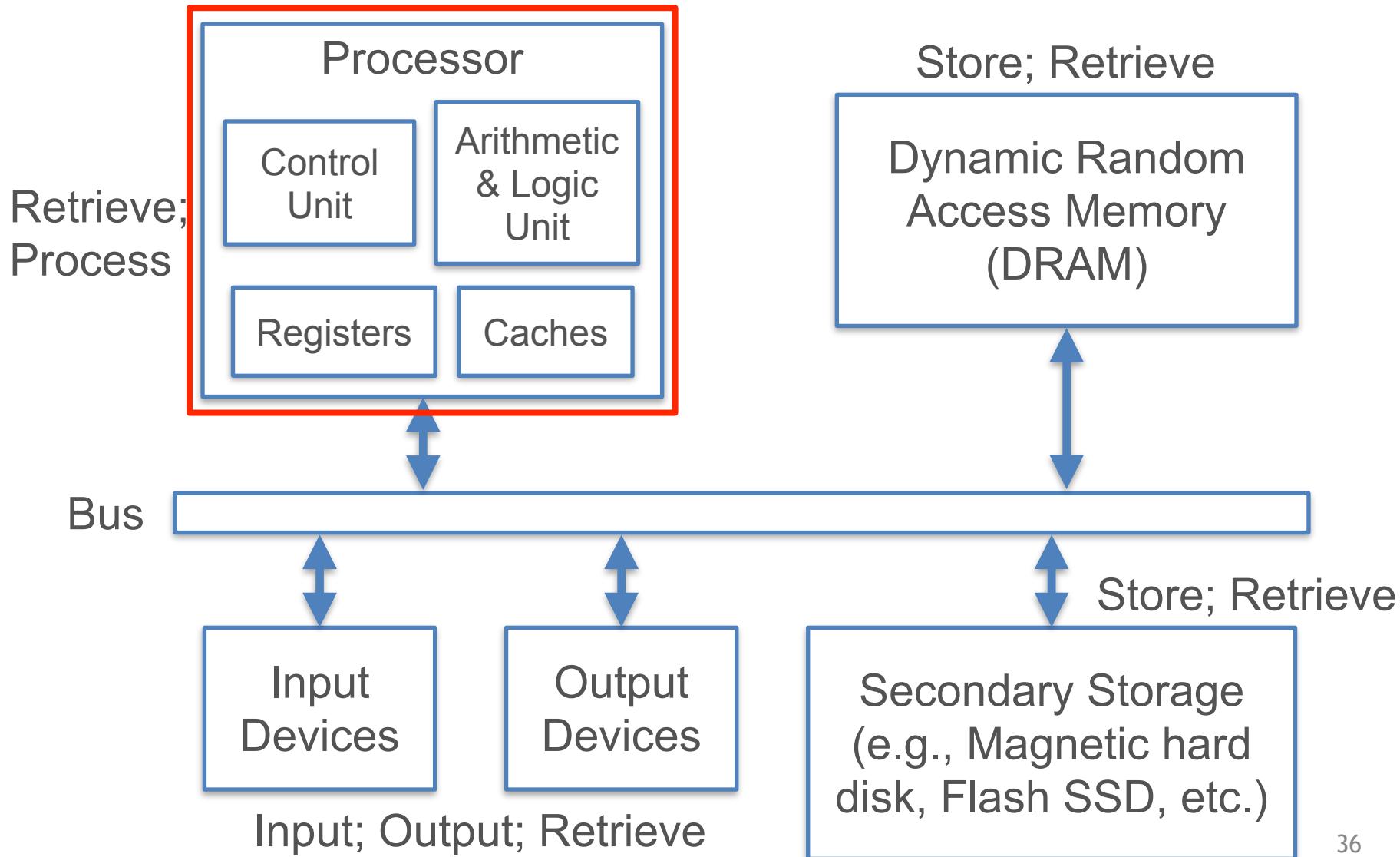
# Basics of Processors

- ❖ **Processor:** Hardware to orchestrate and *execute instructions to manipulate data* as specified by a program
  - ❖ Examples: CPU, GPU, FPGA, TPU, embedded, etc.
- ❖ **ISA:** The vocabulary of commands of a processor



```
80483b4: 55 push %ebp
80483b5: 89 e5 mov %esp,%ebp
80483b7: 83 e4 f0 and $0xffffffff,%esp
80483ba: 83 ec 20 sub $0x20,%esp
80483bd: c7 44 24 1c 00 00 00 movl $0x0,0x1c(%esp)
80483c4: 00
80483c5: eb 11 jmp 80483d8 <main+0x24>
80483c7: c7 04 24 b0 84 04 08 movl $0x80484b0,(%esp)
80483ce: e8 1d ff ff ff call 80482f0 <puts@plt>
80483d3: 83 44 24 1c 01 addl $0x1,0x1c(%esp)
80483d8: 83 7c 24 1c 09 cmpl $0x9,0x1c(%esp)
80483dd: 7e e8 jle 80483c7 <main+0x13>
80483df: b8 00 00 00 00 mov $0x0,%eax
80483e4: c9 leave
80483e5: c3 ret
80483e6: 90 nop
80483e7: 90 nop
80483e8: 90 nop
80483e9: 90 nop
80483ea: 90 nop
```

# Abstract Computer Parts and Data



# Basics of Processors

**Q:** *How does a processor execute machine code?*

- ❖ Most common approach: **load-store architecture**
- ❖ **Registers:** Tiny local memory (“scratch space”) on proc. into which instructions and data are copied
- ❖ ISA specifies bit length/format of machine code commands
- ❖ ISA has several commands to manipulate register contents

# Basics of Processors

*Q: How does a processor execute machine code?*

- ❖ Types of ISA commands to manipulate register contents:
  - ❖ **Memory access:** **load** (copy bytes from DRAM address to register); **store** (reverse); put constant
  - ❖ **Arithmetic & logic** on data items in registers: add/multiply/etc.; bitwise ops; compare, etc.
  - ❖ **Control flow** (branch, call, etc.)
- ❖ **Caches:** Small local memory to buffer instructions/data

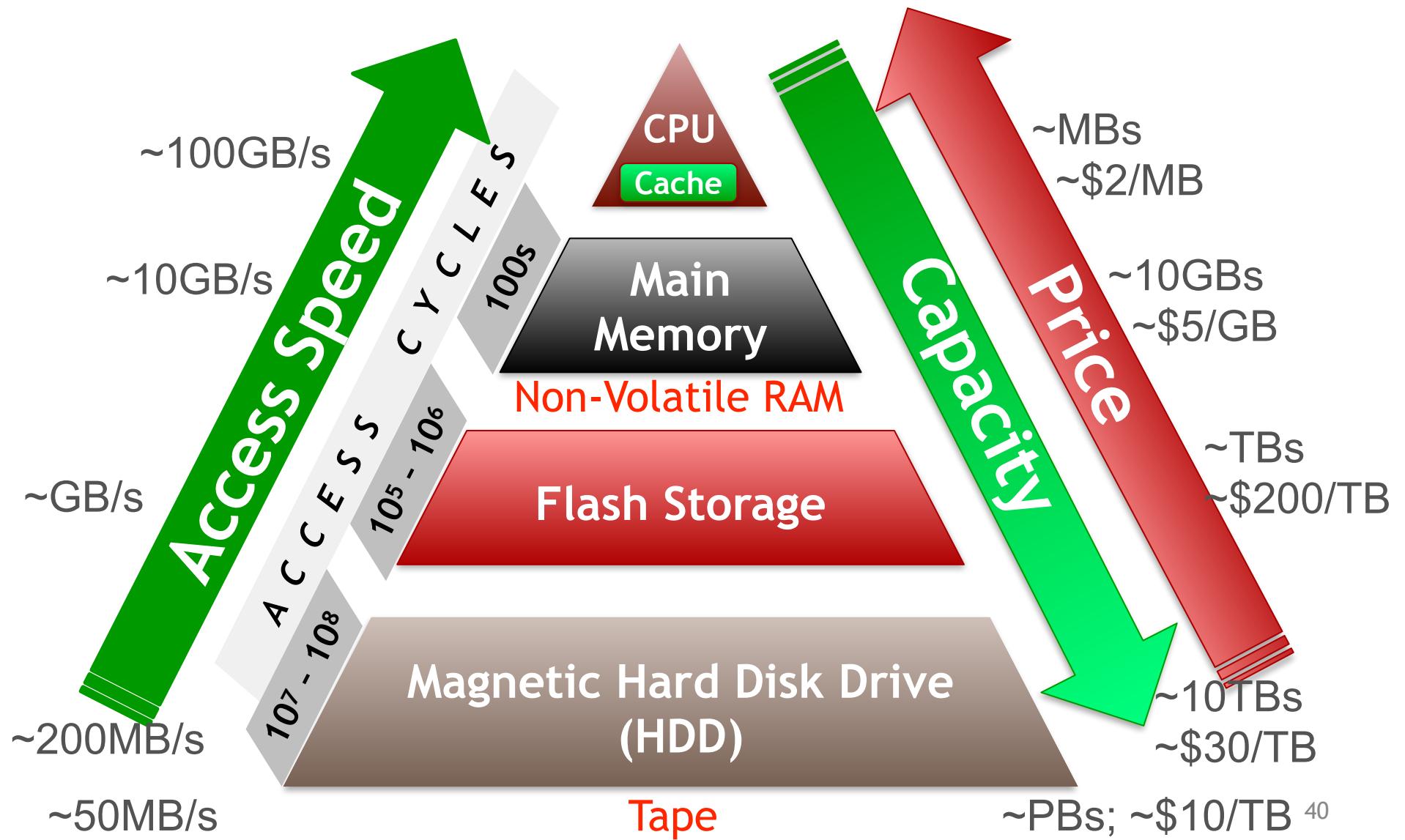
# Processor Performance

*Q: How fast can a processor process a program?*

- ❖ Modern CPUs can run millions of instructions per second!
  - ❖ ISA tells us **#clock cycles** each instruction needs
  - ❖ CPU's **clock rate** lets us convert that to runtime (ns)
- ❖ Alas, most programs do not keep CPU always busy
  - ❖ Memory access commands **stall** the processor; ALU and CU are *idle* during memory-register transfer
  - ❖ Worse, data may not be in DRAM—wait for disk I/O!
  - ❖ So, actual *execution runtime* of program may be OOM higher than what clock rate calculation suggests

**Key Principle:** Optimizing access to main memory and use of processor cache is critical for processor performance!

# Memory/Storage Hierarchy



# Memory/Storage Hierarchy

- ❖ Typical desktop computer today (\$700):
  - ❖ 1 TB magnetic hard disk (SATA HDD); 32 GB DRAM
  - ❖ 3.4 GHz CPU; 4 cores; 8MB cache
- ❖ High-end enterprise rack server for RDBMSs (\$8,000):
  - ❖ 12 TB Persistent memory; 6 TB DRAM
  - ❖ 3.8 GHz CPU; 28-core per proc.; 38MB cache
- ❖ Renting on Amazon Web Services (AWS):
  - ❖ EC2 m5.large: 2-core, 8GiB: \$0.115 / hour
  - ❖ EC2 m5.24xlarge: 96-core, 384 GiB, \$5.53 per hour
  - ❖ EBS general SSD: \$0.12 per GB-month
  - ❖ S3 store / read: \$0.023 / 0.05-0.09 per GB-month

# Key Principle: Locality of Reference

Carefully handling/optimizing access to main memory and use of processor cache is critical for processor performance!



Due to OOM access latency differences across memory hierarchy, optimizing access to lower levels and careful use of higher levels is critical for overall system performance!

- ❖ **Locality of Reference:** Many programs tends to access memory locations in a somewhat *predictable* manner
  - ❖ **Spatial:** Nearby locations will be accessed soon
  - ❖ **Temporal:** Same locations accessed again soon
- ❖ Locality can be exploited to reduce runtimes using **caching** and/or **prefetching** across all levels in the hierarchy

# Concepts of Memory Management

- ❖ **Caching:** Buffering a copy of bytes (instructions and/or data) from a lower level at a higher level to exploit locality
- ❖ **Prefetching:** Preemptively retrieving bytes (typically data) from addresses not explicitly asked yet by program
- ❖ **Spill/Miss/Fault:** Data needed for program is not yet available at a higher level; need to get it from lower level
  - ❖ **Register Spill** (register to cache); **Cache Miss** (cache to main memory); “**Page**” **Fault** (main memory to disk)
- ❖ **Hit:** Data needed is already available at higher level
- ❖ **Cache Replacement Policy:** When new data needs to be loaded to higher level, which old data to evict to make room? Many policies exist with different properties

# Memory Hierarchy in Action

**Q:** *What does this program do when run with ‘python’?  
(Assume tmp.csv is in current working directory)*

tmp.py

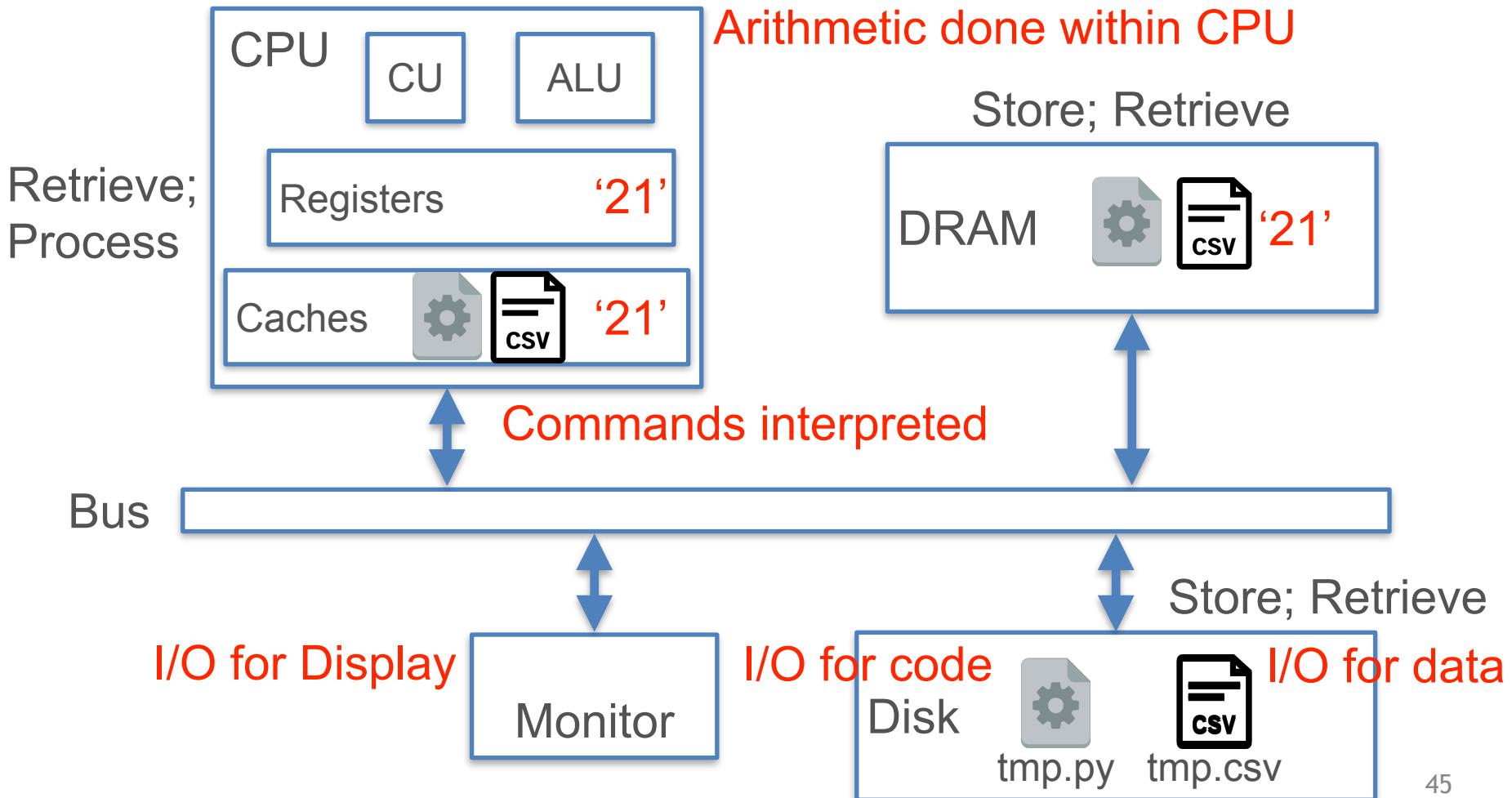
```
import pandas as p  
m = p.read_csv('tmp.csv',header=None)  
s = m.sum().sum()  
print(s)
```

tmp.csv

1,2,3
4,5,6

# Memory Hierarchy in Action

Rough sequence of events when program is executed



# Locality of Reference for Data

- ❖ **Data Layout:**
  - ❖ The *order* in which data items of a complex data structure/ADT are laid out in memory/disk
- ❖ **Data Access Pattern** (of a program on a data object):
  - ❖ The *order* in which a program has to access items of a complex data structure/ADT in memory
- ❖ **Hardware Efficiency** (of a program):
  - ❖ How close *actual execution runtime* is to best possible runtime given the proc. clock rate and ISA
  - ❖ Improved with careful data layout of all data objects used by a program based on its data access patterns
  - ❖ **Key Principle:** Raise cache hits; reduce memory stalls!

# Locality of Reference in Data Science

- ❖ Common example: matrix multiplication (>1m cells each)
- ❖ Suppose data layout in DRAM is in **row-major** order

$$C_{n \times m} = A_{n \times p} B_{p \times m}$$

DRAM    A[1:] A[2:] A[3:] ... B[1:] B[2:] ...

Caches

```
for i = 1 to n  
  for j = 1 to m  
    for k = 1 to p  
      C[i][j] += A[i][k] * B[k][j]
```

- ❖ Not too hardware-efficient
- ❖ Prefetching+caching means full row based on innermost loop is brought to proc. cache
- ❖ A[i][.] Hits but B[k][j] Misses
- ❖ So each \* op is a stall! :(

# Locality of Reference in Data Science

- ❖ Common example: matrix multiplication (>1m cells each)
- ❖ Suppose data layout in DRAM is in **row-major** order

$$C_{n \times m} = A_{n \times p} B_{p \times m}$$

DRAM    A[1:] A[2:] A[3:] ... B[1:] B[2:] ...

Caches

```
for i = 1 to n  
  for k = 1 to p  
    for j = 1 to m  
      C[i][j] += A[i][k] * B[k][j]
```

- ❖ *Logically equivalent* computation but different order of ops!
- ❖ C[i][.] and B[k][.] Hits
- ❖ A[i][k] also Hit (unaffected by j)
- ❖ Orders of magnitude fewer stalls!
- ❖ Lot more hardware-efficient

# Locality of Reference in Data Science

- ❖ Common example: matrix multiplication (>1m cells each)
- ❖ Suppose data layout in DRAM is in ***row-major*** order

$$C_{n \times m} = A_{n \times p} B_{p \times m}$$

```
for i = 1 to n  
  for j = 1 to m  
    for k = 1 to p  
      C[i][j] += A[i][k] * B[k][j]
```

Rewrite 

```
for i = 1 to n  
  for k = 1 to p  
    for j = 1 to m  
      C[i][j] += A[i][k] * B[k][j]
```

- ❖ Although the math is the same and gives the same results (“logically equivalent”), the physical properties of program execution are vastly different
- ❖ Commonly used in *compiler optimization* and later on, also in *query optimization*

# Locality of Reference in Data Science

- ❖ Matrices/tensors are ubiquitous in statistics/ML/DL programs

*Q: Would you like to write ML code in a cache-aware manner? :)*

- ❖ Decades of optimized hardware-efficient libraries exist for matrix/tensor arithmetic (*linear algebra*) that reduce memory stalls and increase parallelism (more on parallelism later) for you
  - ❖ **Multi-core CPUs:** BLAS/LAPACK (C), Eigen (C++), la4j (Java), NumPy/SciPy (Python; can wrap BLAS)
  - ❖ **GPUs:** cuBLAS, cuSPARSE, cuDNN, cuDF, cuGraph

If interested, some benchmark empirical comparisons:

<https://medium.com/datathings/benchmarking-blas-libraries-b57fb1c6dc7>

<https://github.com/andre-wojtowicz/blas-benchmarks>

<https://eigen.tuxfamily.org/index.php?title=Benchmark>

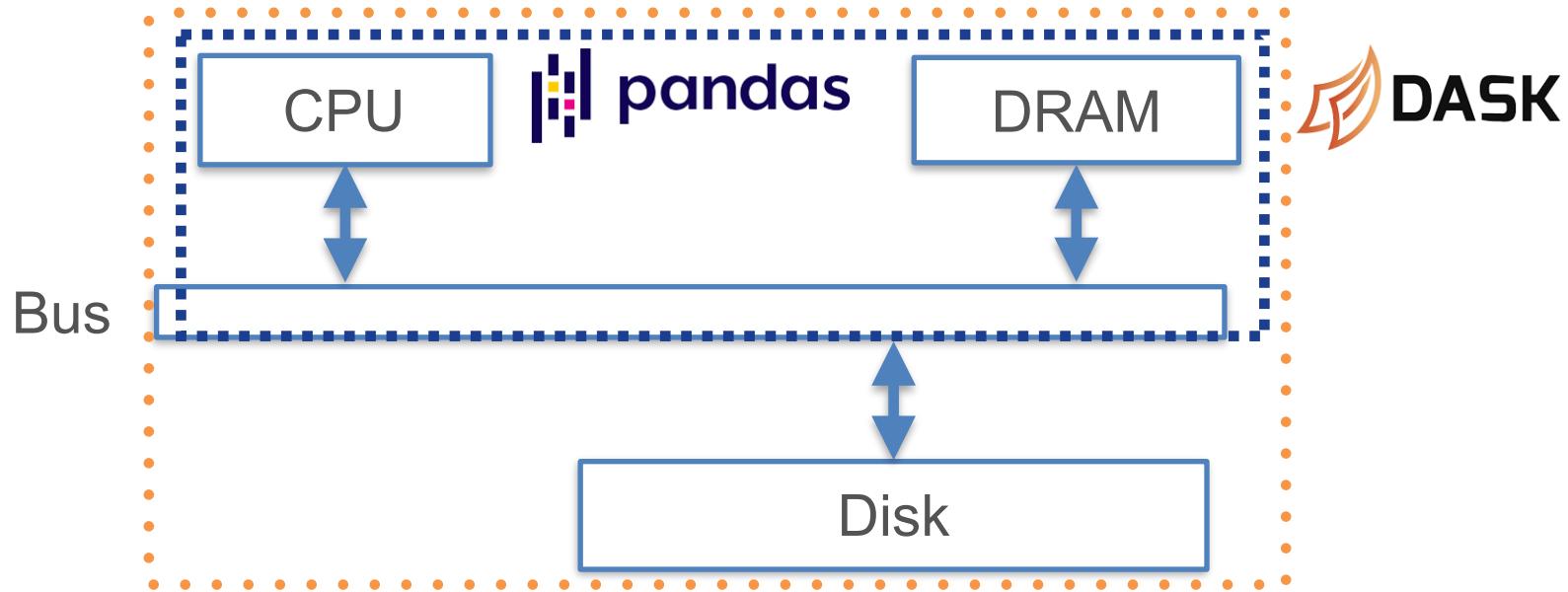
# Breakout Rooms Activity (8min)

Pick a specific data science application domain. It could be anything: natural sciences, social sciences, enterprise industry, Web industry, nonprofits, arts, etc.

1. Name a prediction task where ML is useful and why.
2. Describe dataset(s) for that prediction task (inputs, outputs).
3. What is the rough scale of those datasets? MBs? GBs? TBs?  
Justify why you expect this scale.
4. Describe a prudent memory hierarchy allocation for your task.  
Explain why it is prudent in terms of practical Pareto tradeoffs of data access latency and total cost.

# 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. :)

# Review Questions

- ❖ What is an ISA?
- ❖ What are the three main kinds of commands in an ISA?
- ❖ Why do CPUs have both registers and caches?
- ❖ Why is it typically impossible for data processing programs to achieve 100% processor utilization?
- ❖ Which of these layers in the memory hierarchy is the costliest: CPU cache, DRAM, flash disks, or magnetic hard disks?
- ❖ Which of the above layers is the slowest for data access?
- ❖ What is spatial locality of reference? Briefly describe a simple program that exhibits that.
- ❖ Why does data layout matter for a program's hardware efficiency?
- ❖ Which is more important for optimizing a program's hardware efficiency: data layout or data access pattern?

# 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

# DSC 102

# Systems for Scalable Analytics

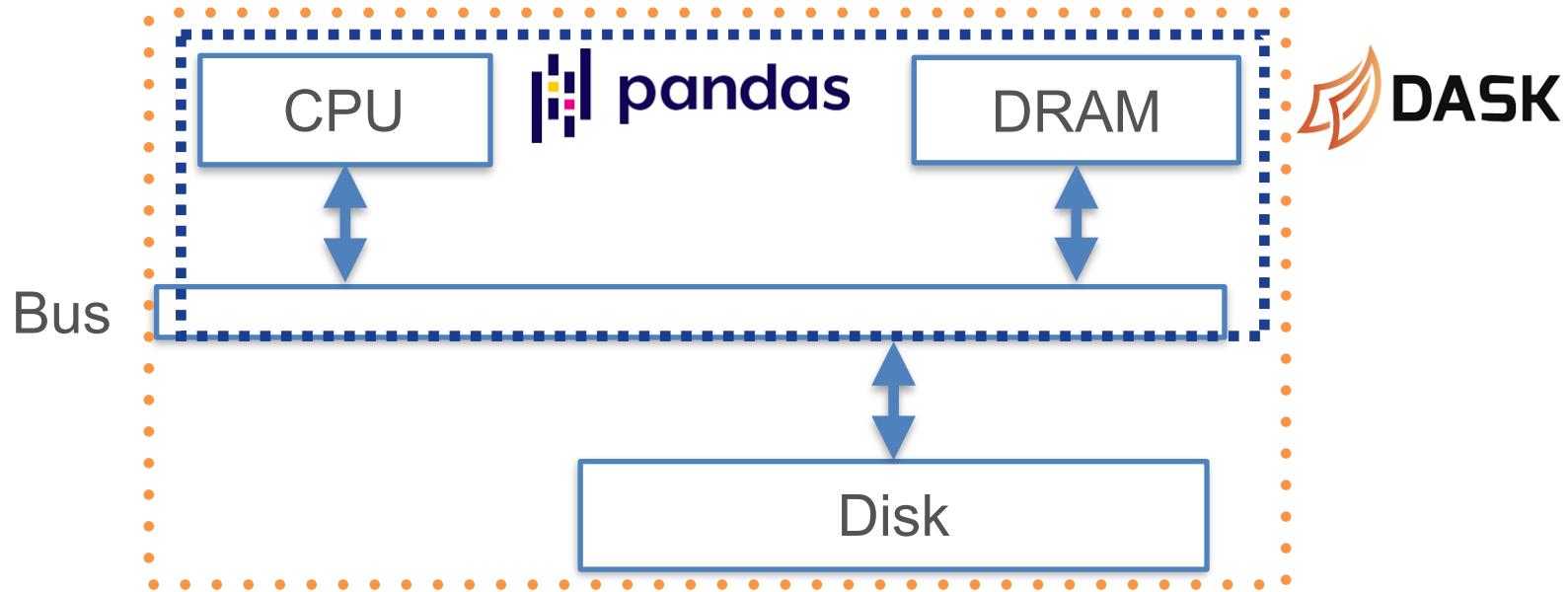
Arun Kumar

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

# 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

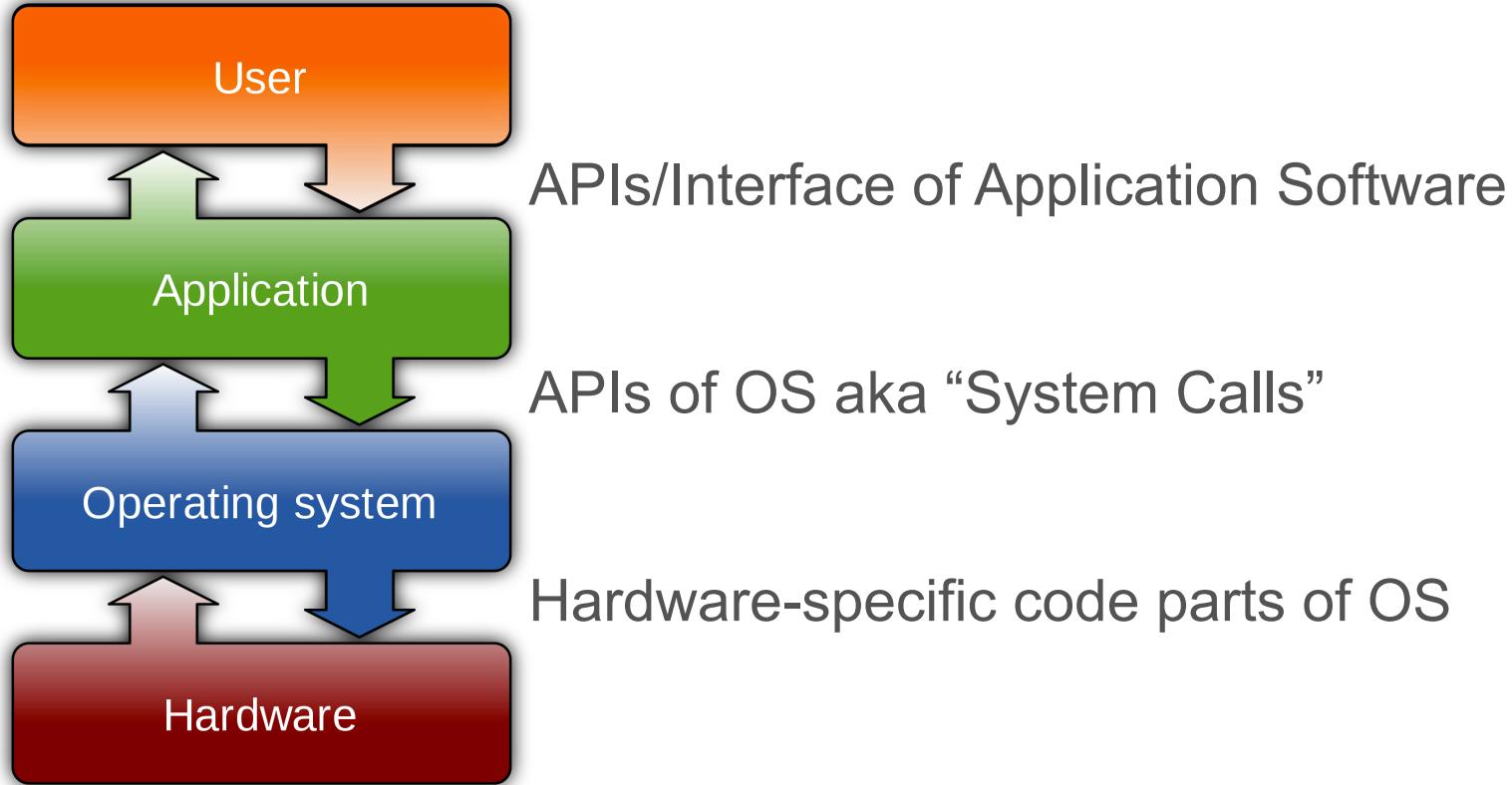
***Q: What is an OS? Why do we need it?***



# 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*
  - ❖ Kinda like the 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
    - ❖ Kinda like executive-legislature-judiciary split
  - ❖ **Abstraction:** *Layers of functionalities* from low-level (close to hardware) to high level (close to user)
    - ❖ Kinda like local-city-county-state-federal levels?

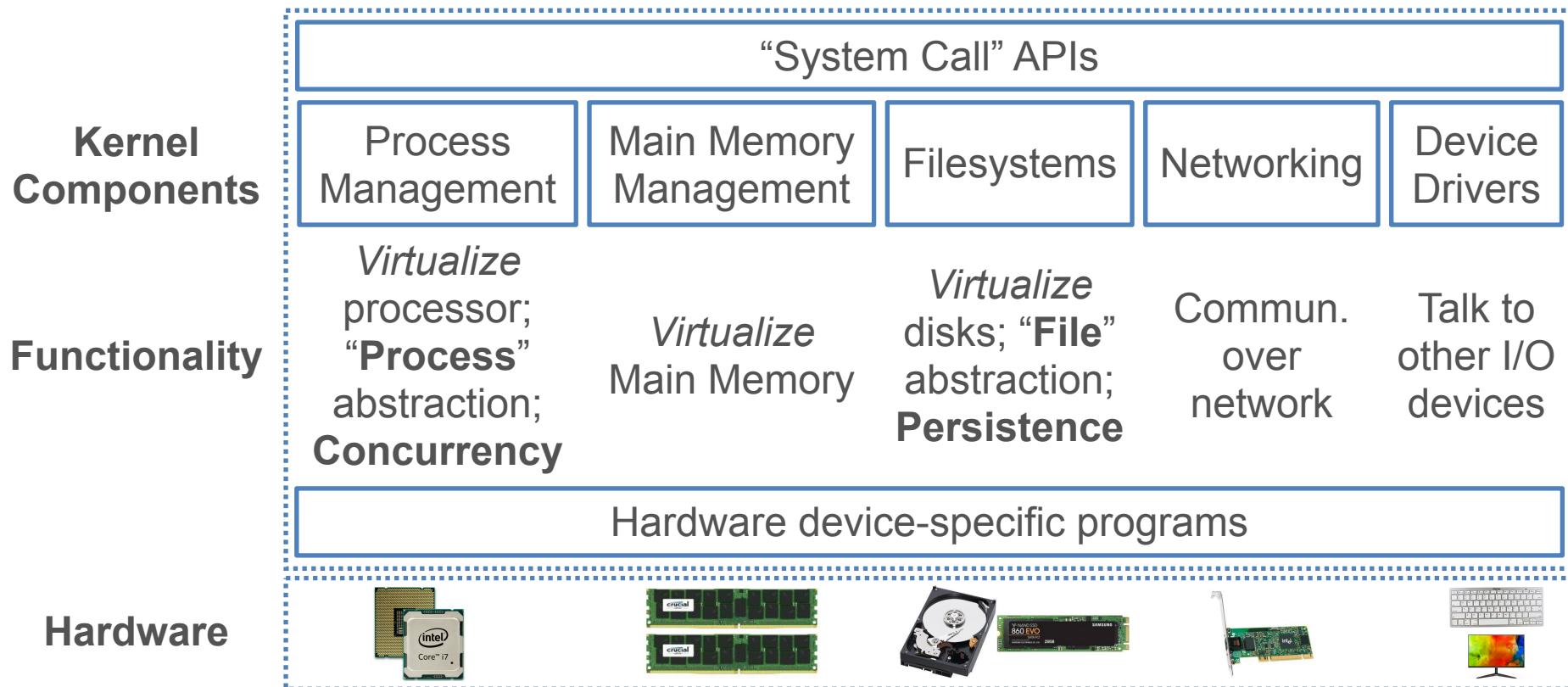
# Role of an OS in a Computer



“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<sub>10</sub>

# 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**

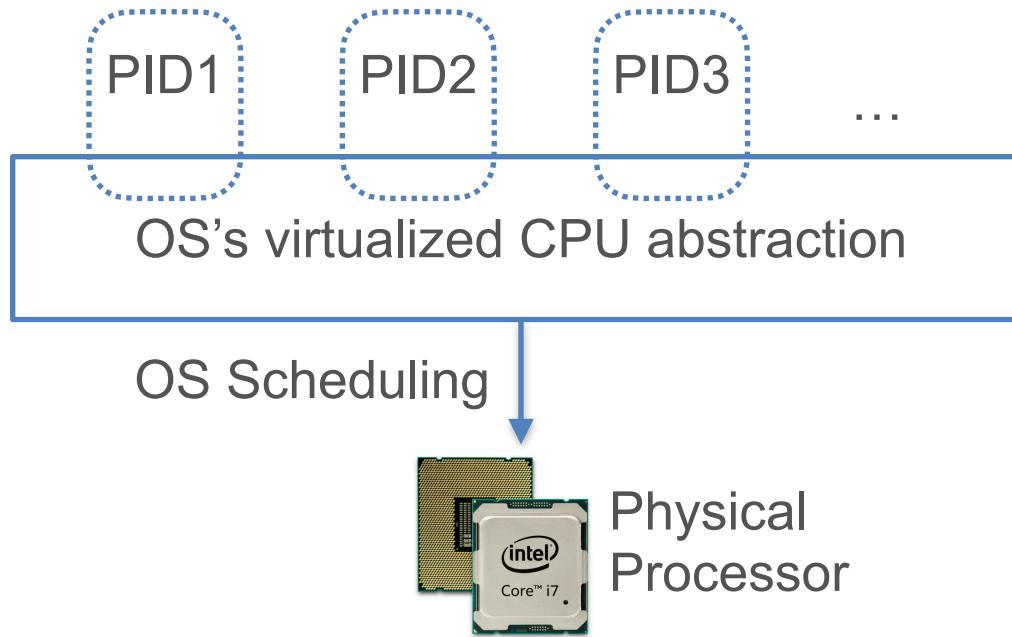
# Virtualization of Hardware Resources

*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

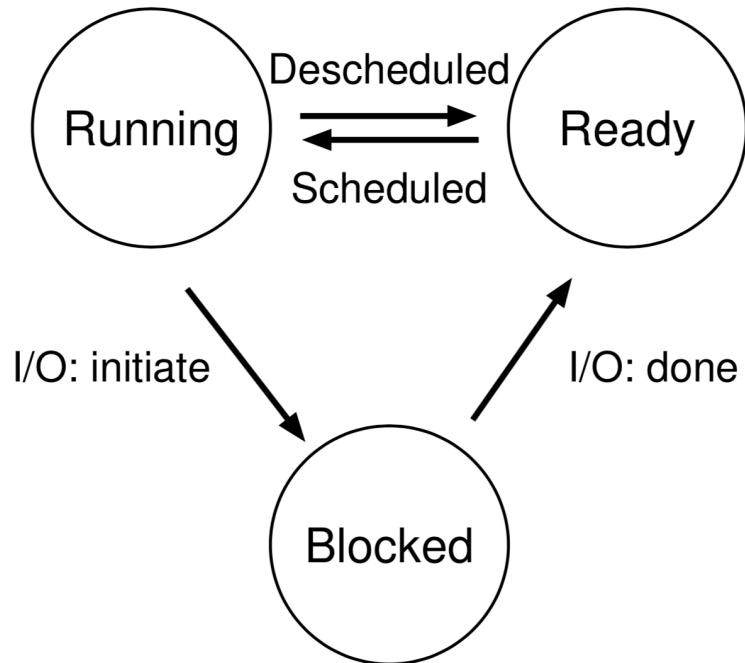
- ❖ Virtualization of processor enables process **isolation**, i.e., each process given an “illusion” that it alone runs



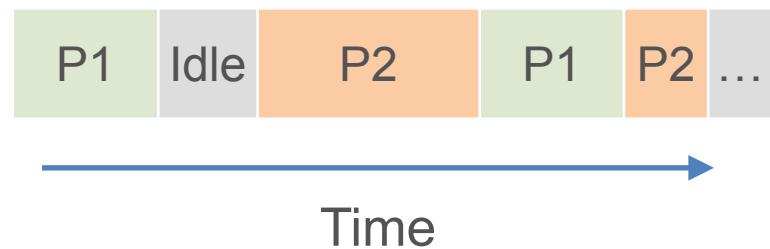
- ❖ Inter-process communication possible in System Calls API
- ❖ Later: Generalize to **Thread** abstraction for **concurrency**

# 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

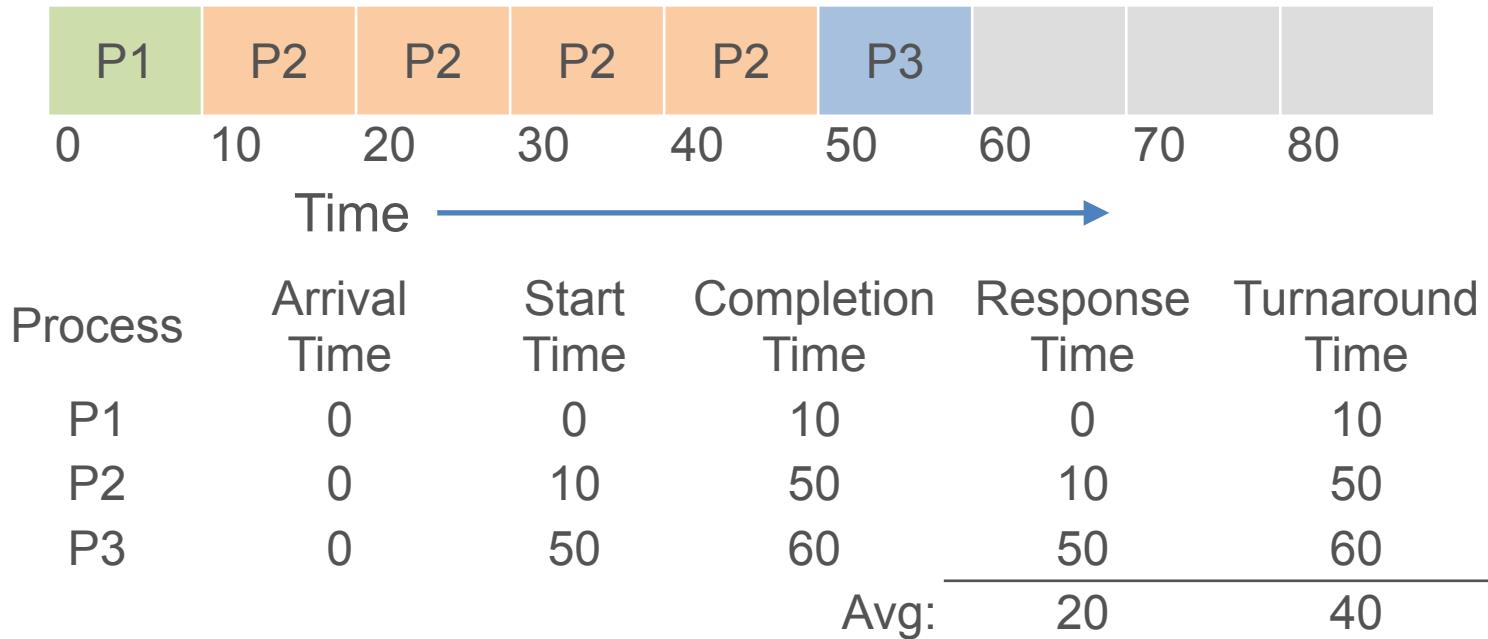
# Scheduling Policies/Algorithms

- ❖ 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*
  - ❖ 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

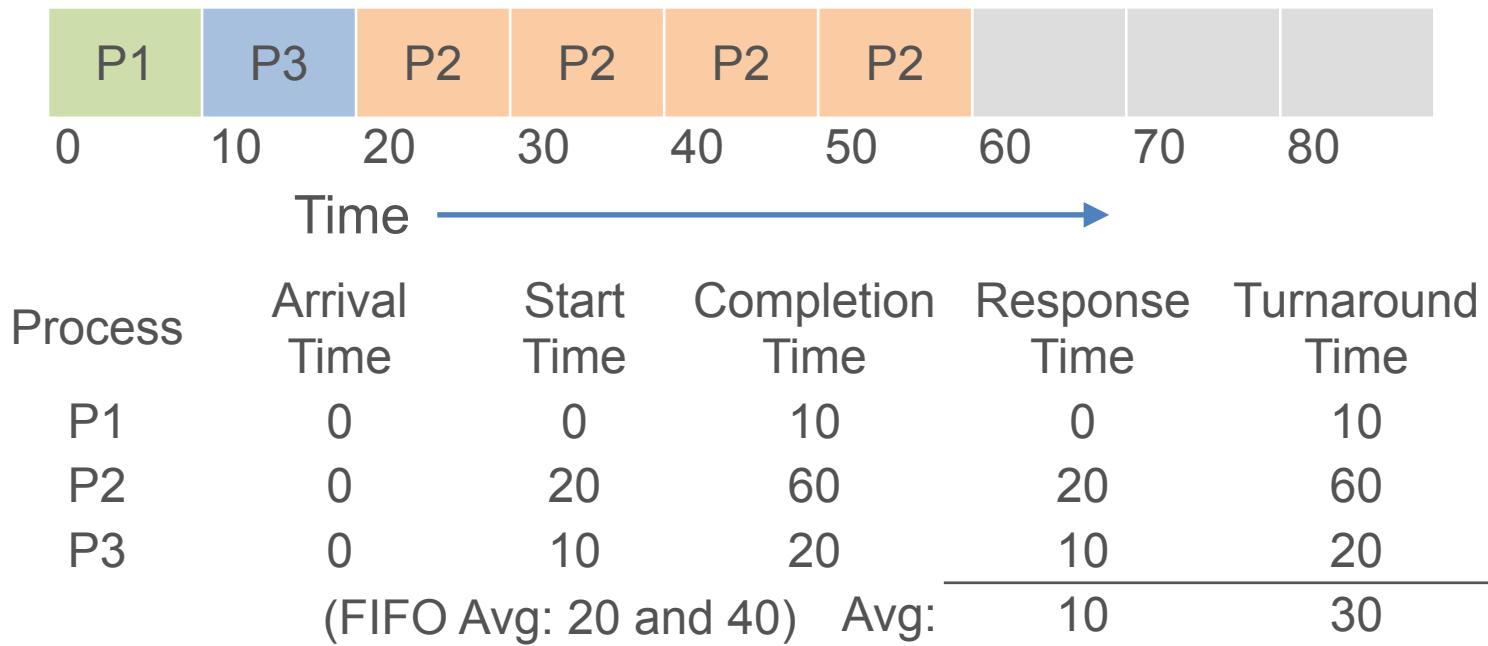


- ❖ Main con: Short jobs may wait a lot, aka “Convoy Effect”

# 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

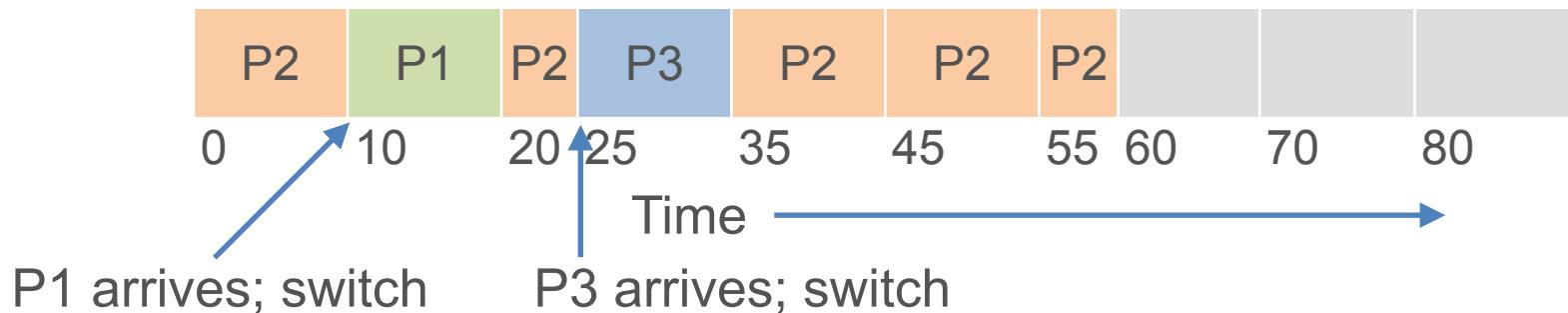


- ❖ Main con: Not all Job Lengths might be known beforehand

# 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



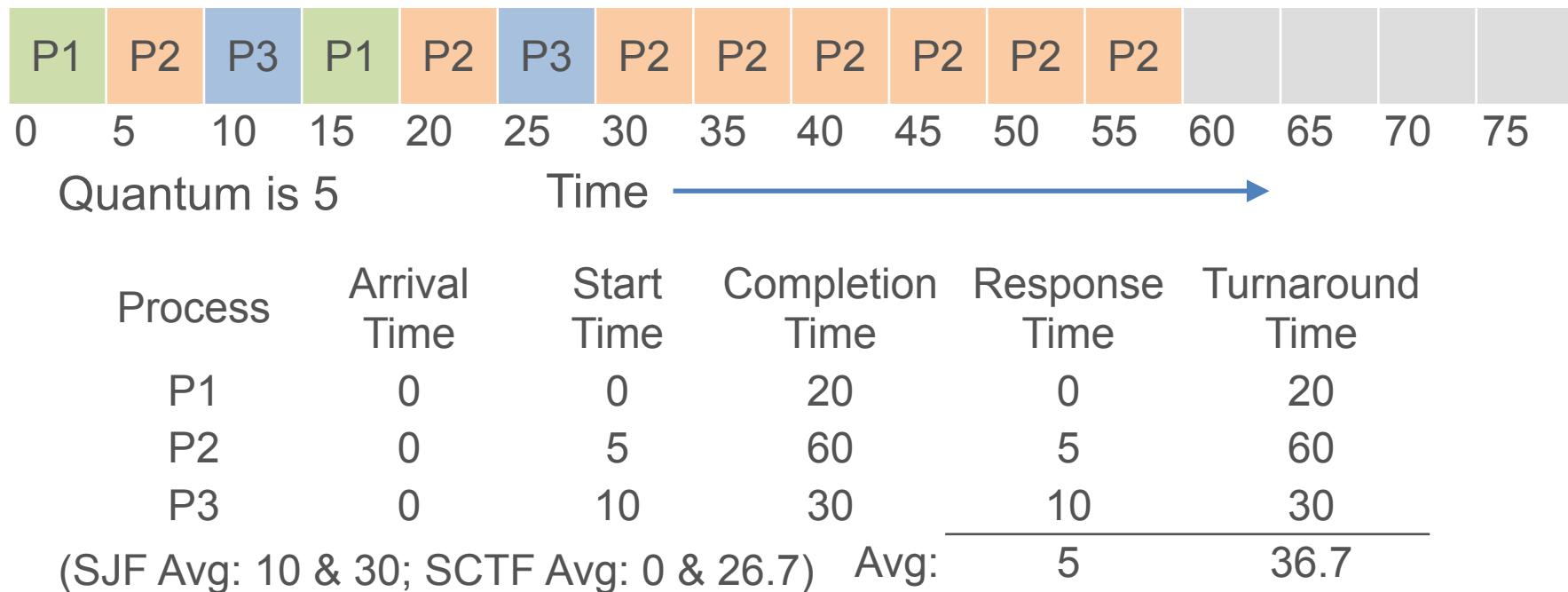
Process	Arrival Time	Start Time	Completion Time	Response Time	Turnaround Time
P1	10	10	20	0	10
P2	0	0	60	0	60
P3	25	25	35	0	10
(SJF Avg: 10 and 30)			Avg:	0	26.7

- ❖ 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

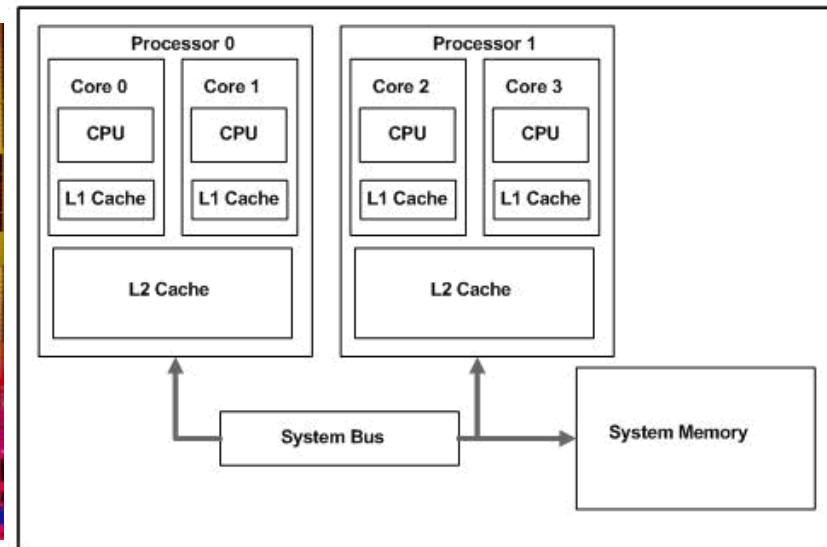
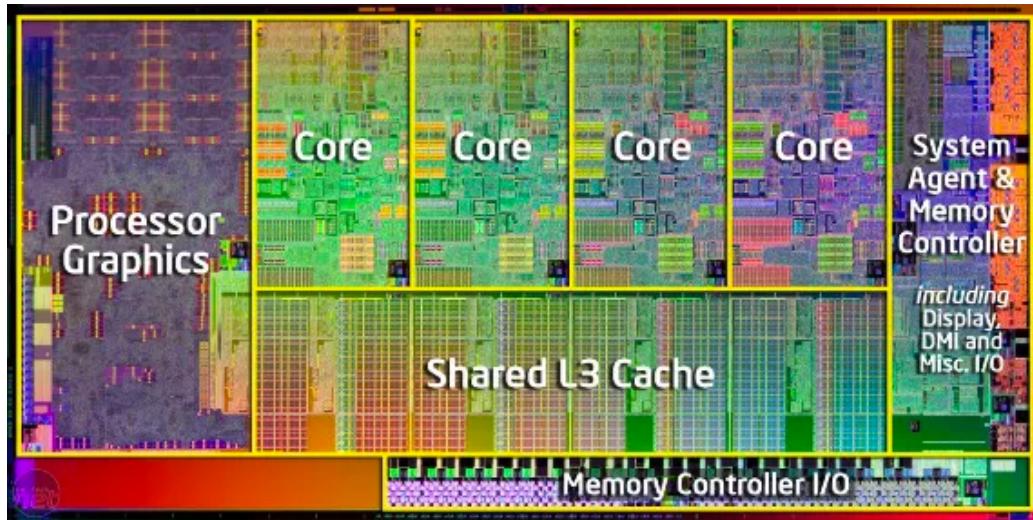
**Example:** P1, P2, P3 of lengths 10,40,10 units arrive closely in that order



- ❖ 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

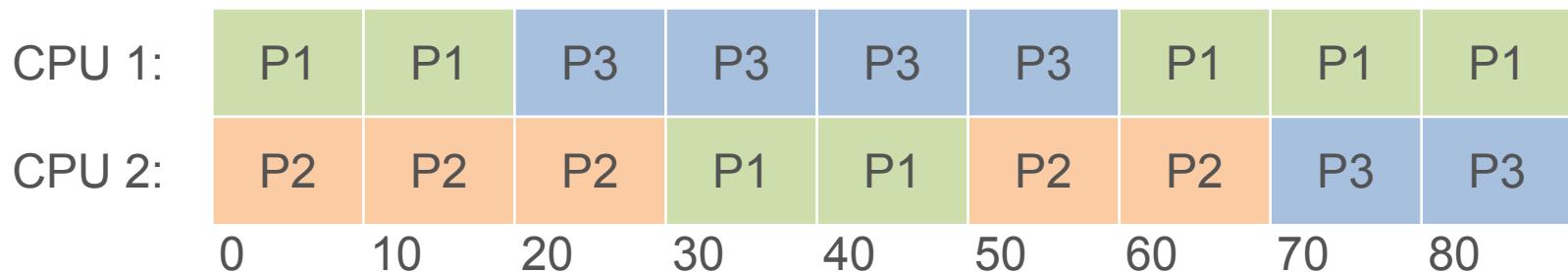
- ❖ **Multiprocessing:** Different processes run on different cores (or entire CPUs) simultaneously
- ❖ **Thread:** Generalization of OS's Process abstraction
  - ❖ A program *spawns* many threads; each run parts of the program's computations simultaneously
  - ❖ **Multithreading:** Same core used by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

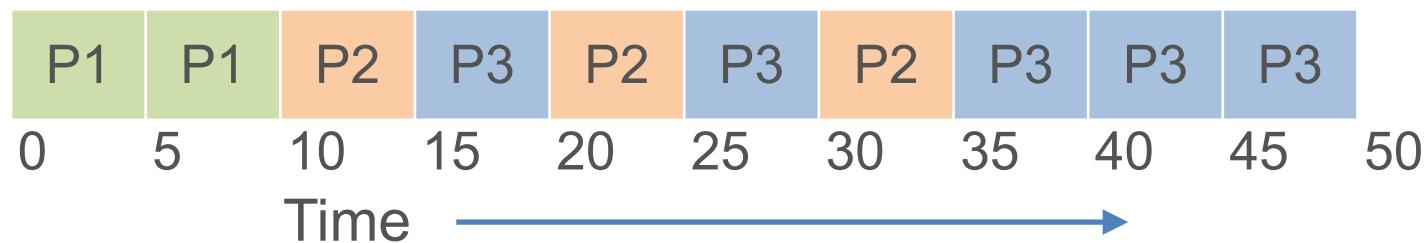
1. Briefly explain two differences between DRAM and disk.
2. If you can afford infinite DRAM, is there any reason not to do so?
3. Why is it important to align data access pattern and data layout?
4. What is the purpose of an OS?
5. Why is the design of an OS so modular?
6. Why does an OS need to use a scheduling policy?
7. Which quantity captures latency of a process starting: Response Time or Turnaround Time?
8. What gives rise to different scheduling policies?
9. Which scheduling policy is the fairest among the ones we covered?
10. What is the Convoy Effect? Which sched. policy has that issue?
11. Explain one pro and one con of Round Robin over SJF.

# Review Questions

Here is a Gantt Chart for 3 processes of the given lengths that arrive at times 0, 5, and 10, resp.

- A) What is the rough *average response time*?
- B) What is the rough *average turnaround time*?

P1, P2, and P3 are of lengths 10, 15, and 25 units, resp.



Process	Arrival Time	Start Time	Completion Time	Response Time	Turnaround Time
P1	0	0	10	0	10
P2	5	10	35	5	30
P3	10	15	50	5	40

Avg: 3.3

26.7

# 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

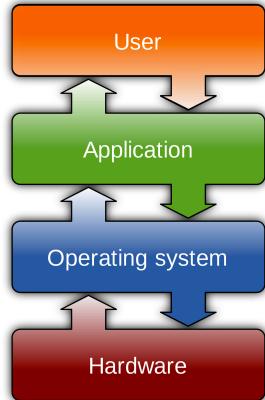
# Filesystem

- ❖ 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

# Virtualization of File on Disk

- ❖ OS abstracts a file on disk as a virtual object for processes
- ❖ **File Descriptor:** An OS-assigned +ve 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
- ❖ **Iseek()**: 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?*

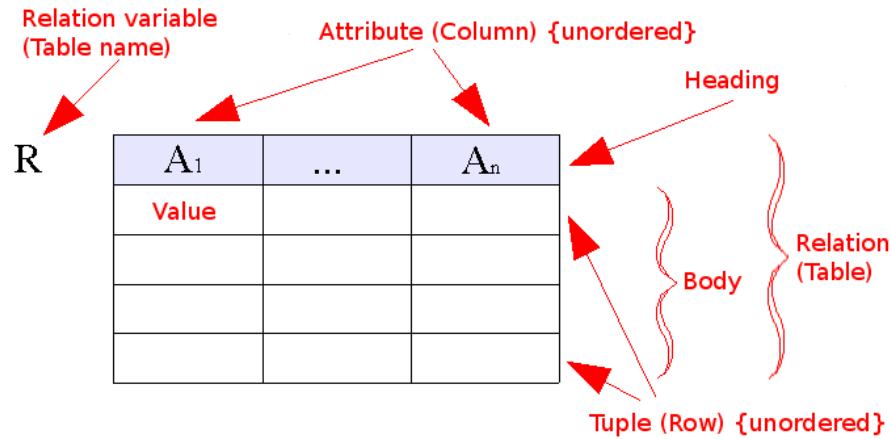
# Files Vs Databases: Data Model

- ❖ **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

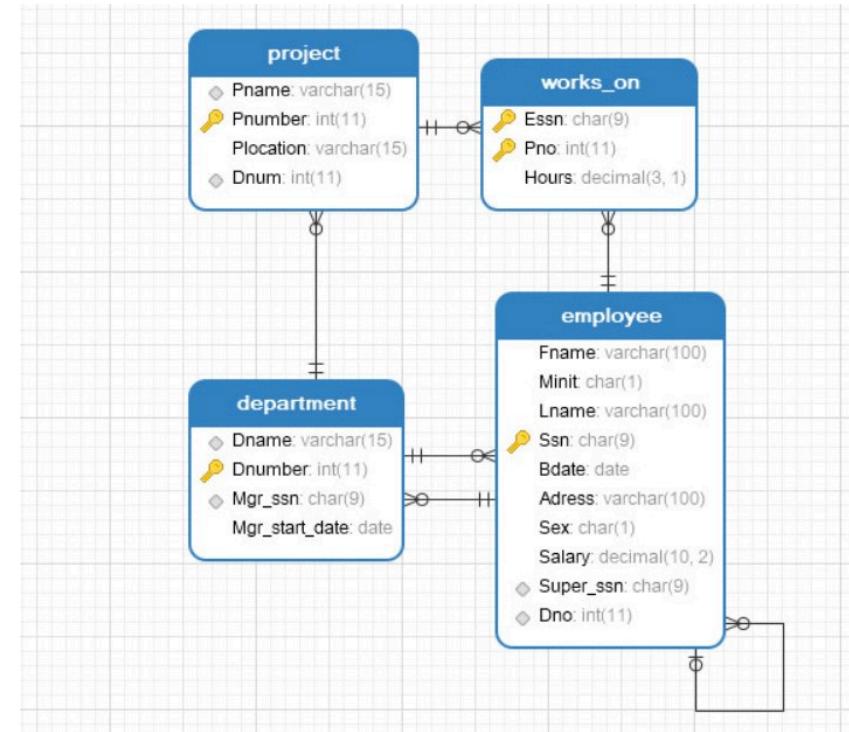
# Data as File: Structured

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

## Relation



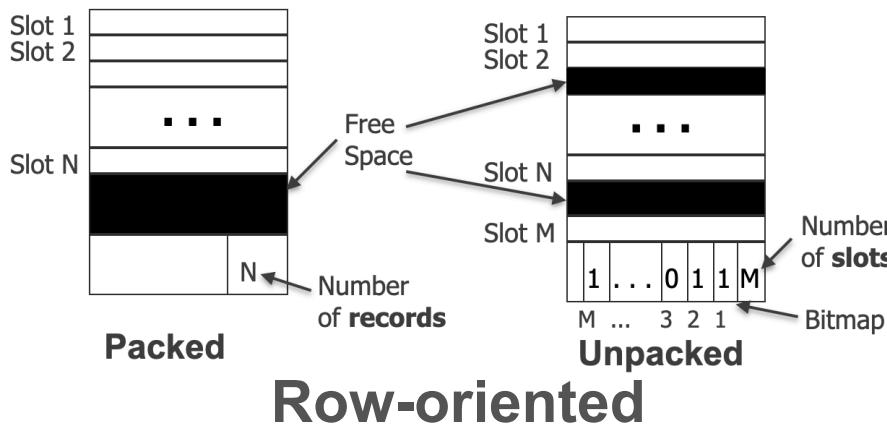
## Relational Database



- ❖ 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



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

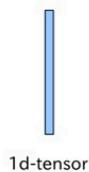
# Data as File: Structured

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

**Matrix**

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

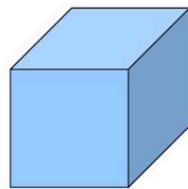
**Tensor**



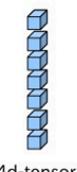
1d-tensor



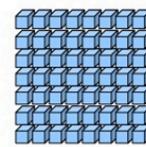
2d-tensor



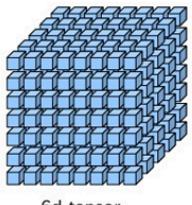
3d-tensor



4d-tensor

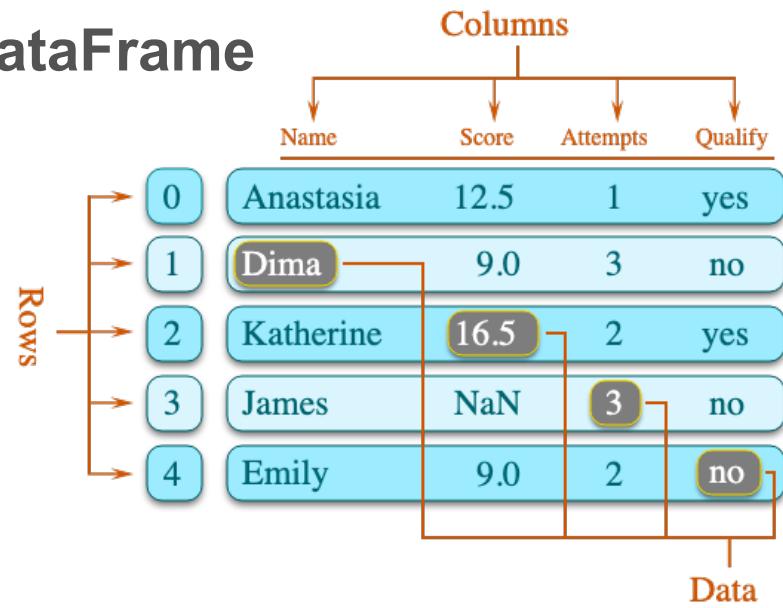


5d-tensor



6d-tensor

**DataFrame**

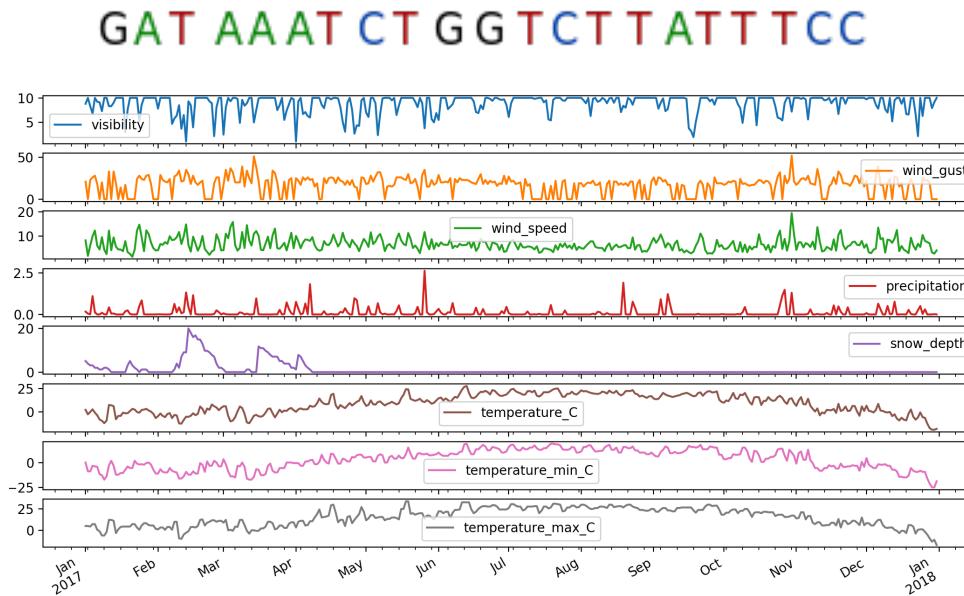


- ❖ Typically serialized as restricted ASCII text file (TSV, CSV, etc.)
- ❖ Matrix/tensor as binary too
- ❖ Can layer on Relations too!

# Data as File: Structured

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

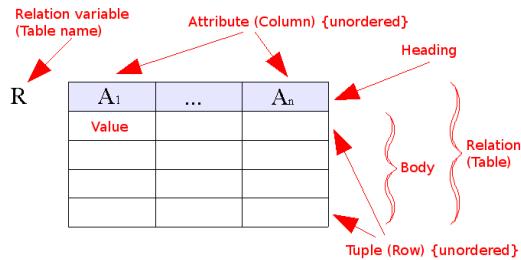
**Sequence  
(Includes  
Time-series)**



- ❖ 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'. Rows are indexed from 0 to 4. Red annotations explain the components: 'Columns' points to the column headers, 'Rows' points to the row indices, and 'Data' points to the individual cells. Specific cells are highlighted in orange, such as 'Dima' at index 1, '16.5' at index 2, 'NaN' at index 3, and 'no' at index 4.

	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

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 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>
```

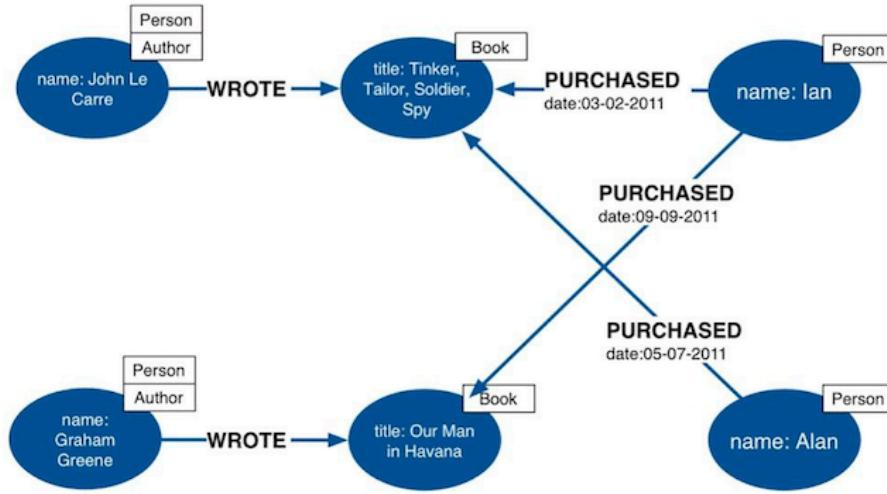
```
[ {
  {
    orderId: 1,
    date: '1/1/2014',
    orderItems: [
      {itemId: 1, qty: 3, price: 23.4},
      {itemId: 23, qty: 2, price: 3.3},
      {itemId: 7, qty: 5, price: 5.3}
    ]
  },
  {
    orderId: 2,
    date: '1/2/2014',
    orderItems: [
      {itemId: 31, qty: 7, price: 3.8},
      {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

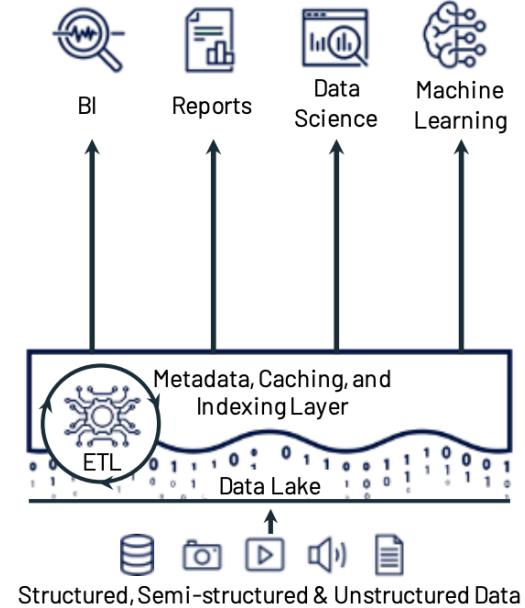
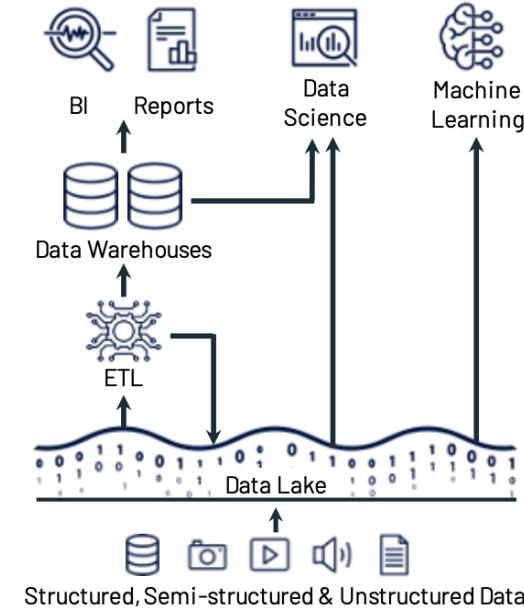
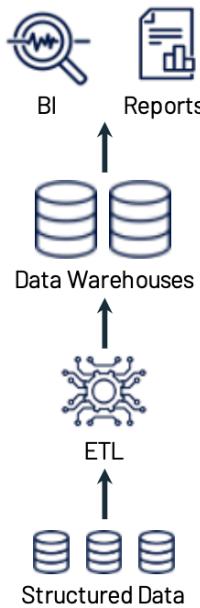


- ❖ Typically serialized with JSON or similar textual formats
- ❖ Some data systems also offer binary file formats
- ❖ Again, can layer on Relations too

**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](http://cidrdb.org/cidr2021/papers/cidr2021_paper17.pdf)

# Data Lake File Format Tradeoffs

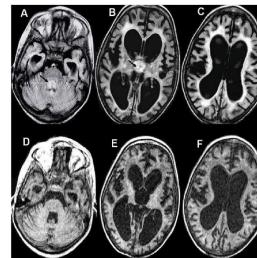
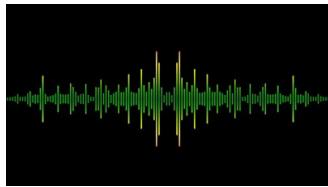
- ❖ Pros and cons of Parquet vs text-based files (CSV, JSON, etc.):
  - ❖ **Less storage:** Parquet stores in **compressed** form; can be much smaller (even 10x); less I/O to read
  - ❖ **Column pruning:** Enables app to read only columns needed to DRAM; even less I/O now!
  - ❖ **Schema on file:** Rich 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

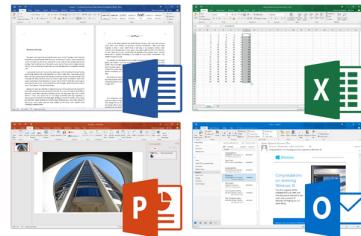
Dataset	Size on Amazon S3	Query Run Time	Data Scanned	Cost
Data stored as CSV files	1 TB	236 seconds	1.15 TB	\$5.75
Data stored in Apache Parquet Format	130 GB	6.78 seconds	2.51 GB	\$0.01
Savings	87% less when using Parquet	34x faster	99% less data scanned	99.7% savings

# 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



# Breakout Rooms Activity (8min)

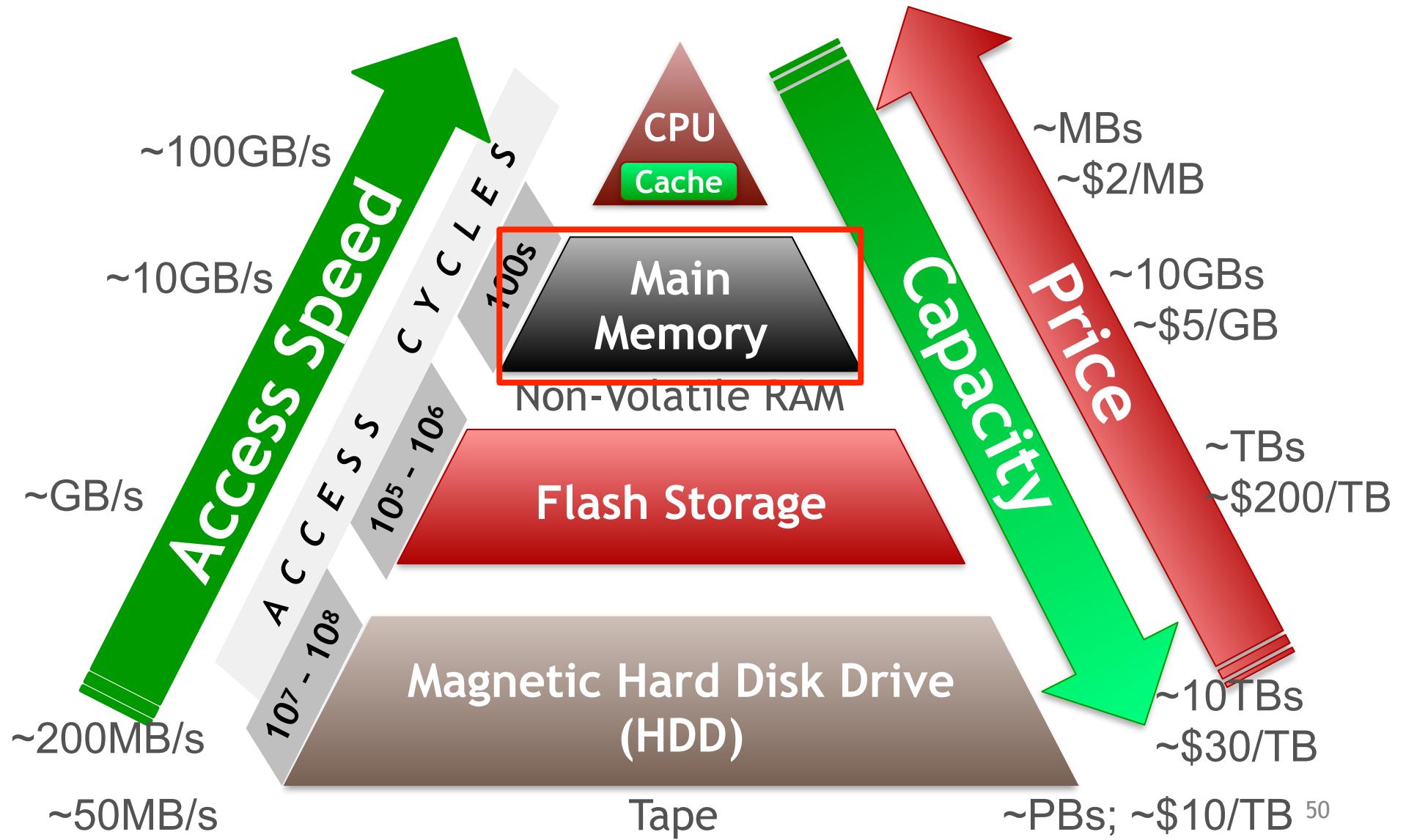
In the real world, Data Science is a team sport. The ability to learn from your wider community of peers/colleagues at a higher level than your specific implementation goals is a crucial skill you must pick up. Given this, answer the following about PA0:

1. Briefly describe one specific good programming/software engineering practice or tool that helped you with PA0 and how.
2. Briefly describe one specific system/software error you faced with AWS or Dask in PA0 and how you went about resolving it.
3. Briefly explain one general lesson PA0 taught you on handling large-scale data with Dask vs small-scale data with Pandas.

# Outline

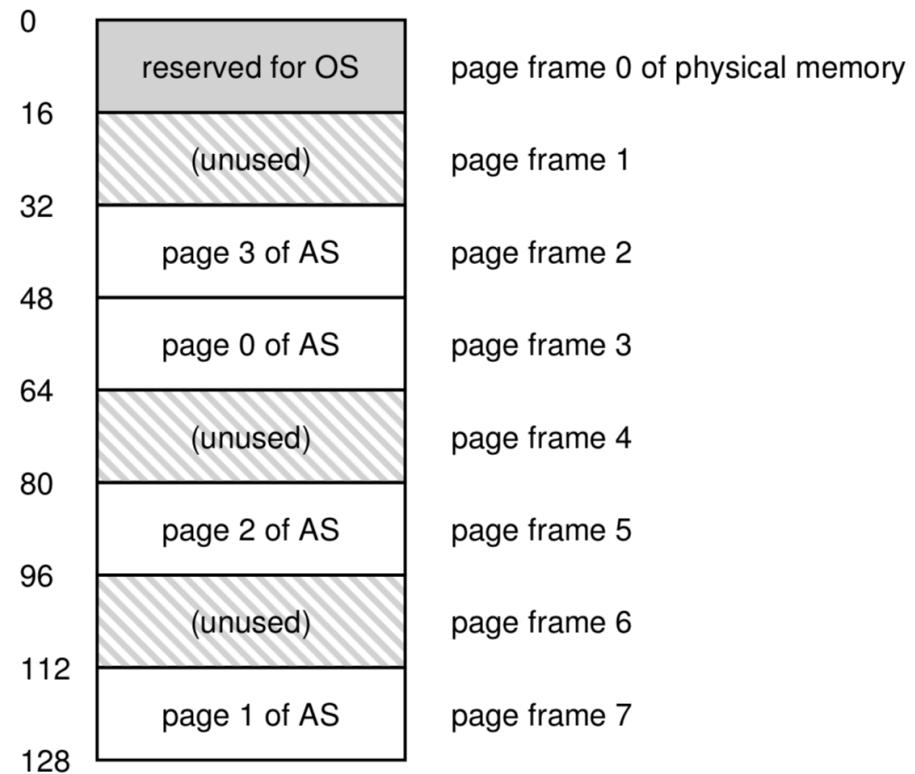
- ❖ Basics of Computer Organization
  - ❖ Digital Representation of Data
  - ❖ Processors and Memory Hierarchy
- ❖ Basics of Operating Systems (OS)
  - ❖ Process Management: Virtualization; Concurrency
  - ❖ Filesystem and Data Files
-  ❖ Main Memory Management
- ❖ Persistent Data Storage

# 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



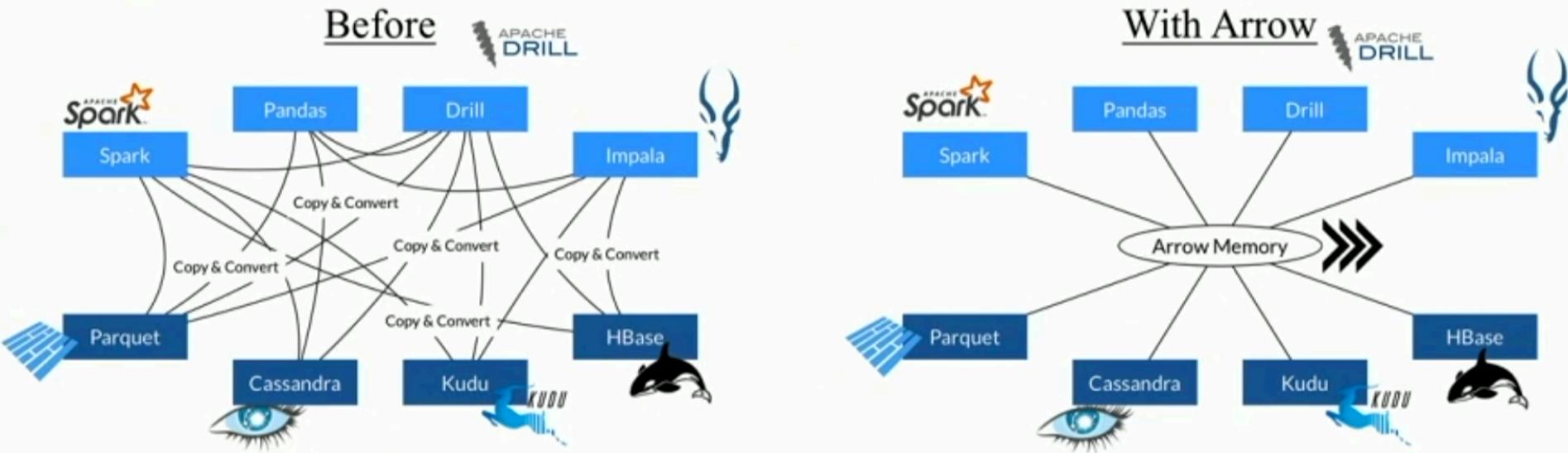
# Apportioning of DRAM

- ❖ 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 <sup>52</sup>

# 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.



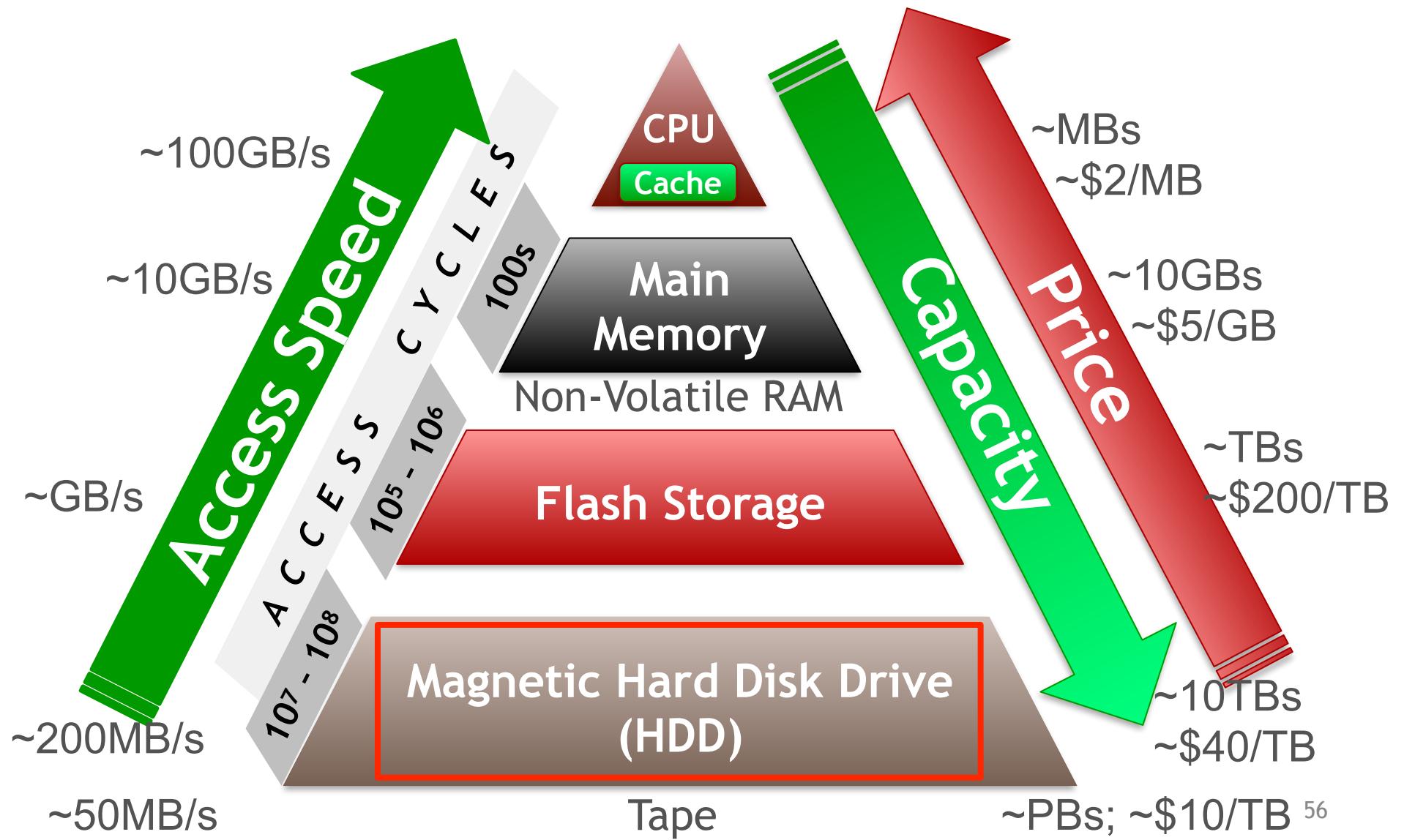
# Outline

- ❖ 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* layout of data on disk pages
  - ❖ Orders of magnitude performance gaps possible

# Data Organization on Disk

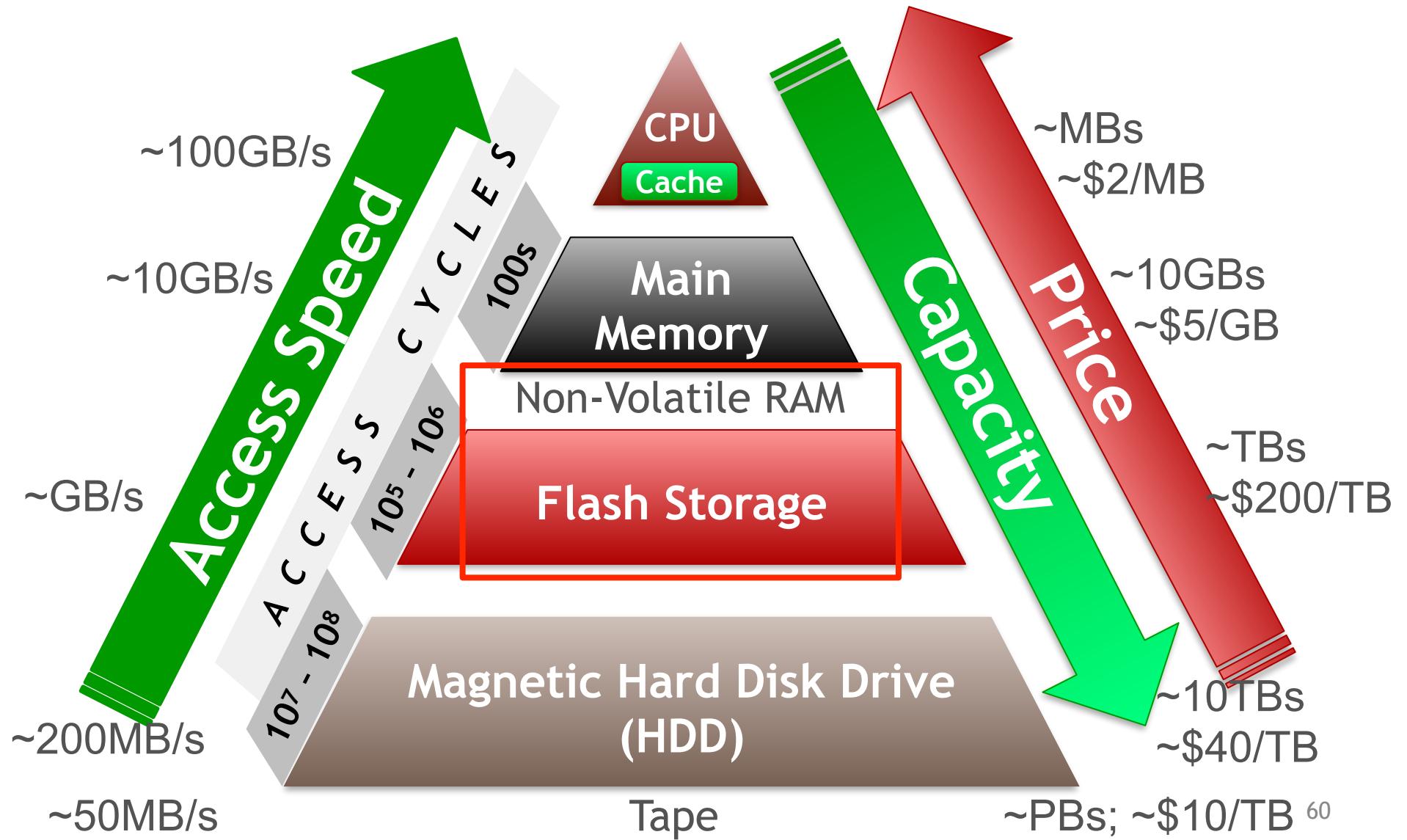
- ❖ Disk space is organized into **files**
- ❖ Files are made up of disk **pages** aka **blocks**
- ❖ Typical disk block/page size: 4KB or 8KB
  - ❖ Basic unit of reads/writes for a disk
  - ❖ OS/RAM page is *not* the same as disk page!
  - ❖ Typically, OS/RAM page size = disk page size but not always; 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

- ❖ What are the 2 levels of a filesystem? Why the dichotomy?
- ❖ How is a database different from a file?
- ❖ What are the 2 levels of a database? Why the dichotomy?
- ❖ Name 3 forms of structured, 2 forms of semistructured, and 2 forms of unstructured data models.
- ❖ Describe 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? Why are both gaining popularity in practice?
- ❖ Name 3 forms of persistent storage devices. Which one has a key latency dichotomy for random vs sequential data access?

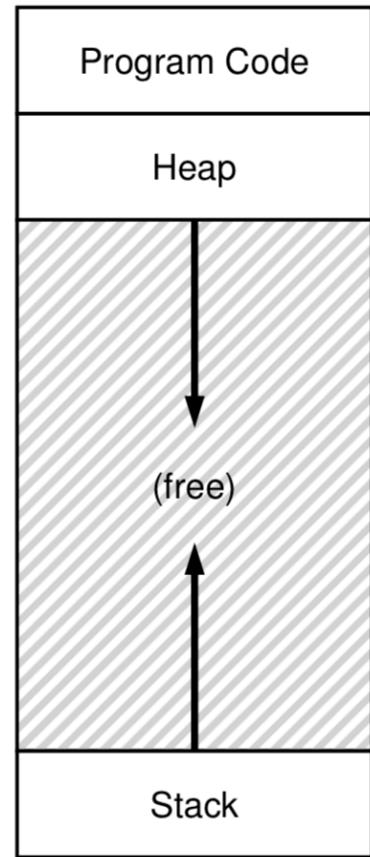
# Outline

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

Optional: More on Memory Management  
Not included in syllabus

# Address Space

- ❖ Chunk(s) of memory assigned by OS to a process
  - ❖ Helps virtualizes and apportion physical memory
- ❖ Split into 3 **segments**: Code, Stack, and Heap
  - ❖ Stack stores mostly statically known data (function arguments, return values, etc.)
  - ❖ Heap is for dynamically created data structures (**malloc()** system call)
  - ❖ Stack/Heap can grow/shrink on the fly when a process is running
  - ❖ **Segmentation fault**: illegal address access
  - ❖ **Memory leak**: program failed to **free()** dynamic space

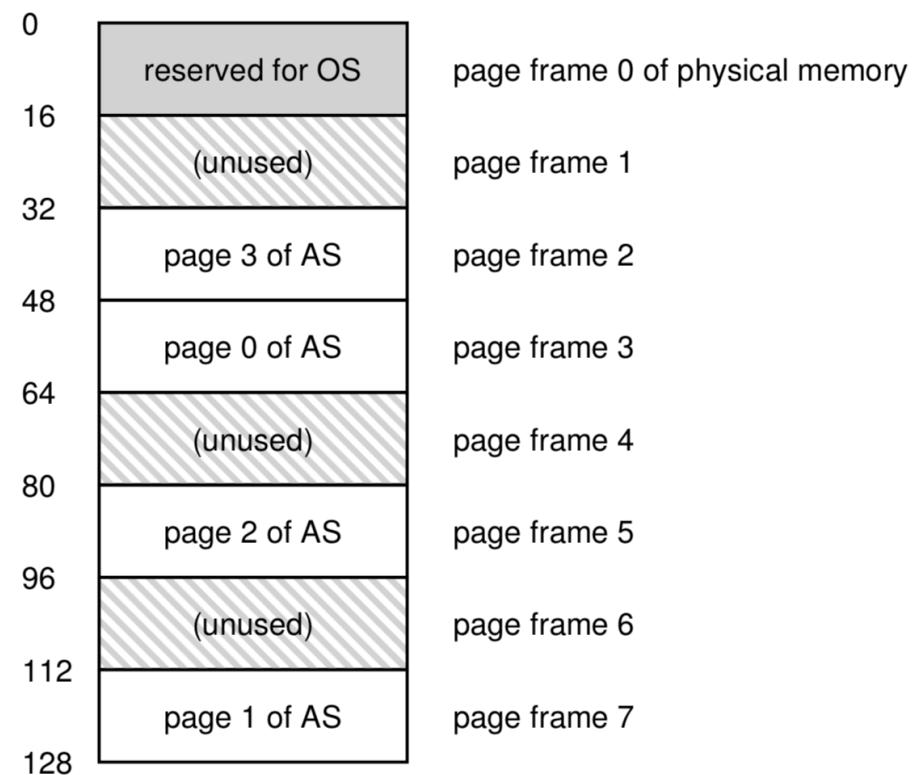


# Virtual Memory

- ❖ **Virtual Address vs Physical Address:**
  - ❖ Physical is tricky and not flexible for programs
  - ❖ Virtual gives “isolation” illusion when using DRAM
  - ❖ OS and hardware work together to quickly perform **address translation**
- ❖ OS maintains **free space list** to tell which chunks of DRAM are available for new processes, avoid conflicts, etc.
  - ❖ Variable-sized
  - ❖ **Fragmentation** possible; algorithms exist to tackle it
- ❖ If DRAM space not enough, OS can map virtual address to disk (lower level in memory hierarchy)

# Abstraction of Page in Memory

- ❖ **Page:** An abstraction of *fixed* size chunks of storage
  - ❖ Makes it easier to manage memory virtualization
- ❖ **Page Frame:** A virtual “slot” in DRAM to hold a page
- ❖ Frame numbers; virtual vs physical page numbers
- ❖ OS has **page table** data structure per process to map virtual to physical
- ❖ Overall, DRAM chopped up by OS neatly into frames



# Swap Space and Paging

- ❖ Sometimes, DRAM may not be enough for process(es)
  - ❖ OS expands virtual memory idea to disk-resident data
- ❖ **Swap Space:** OS reserved space on disk to *swap* pages in and out of DRAM (physical memory)
  - ❖ OS should know **disk address** of pages and translate
  - ❖ Later: how data is laid out on disks

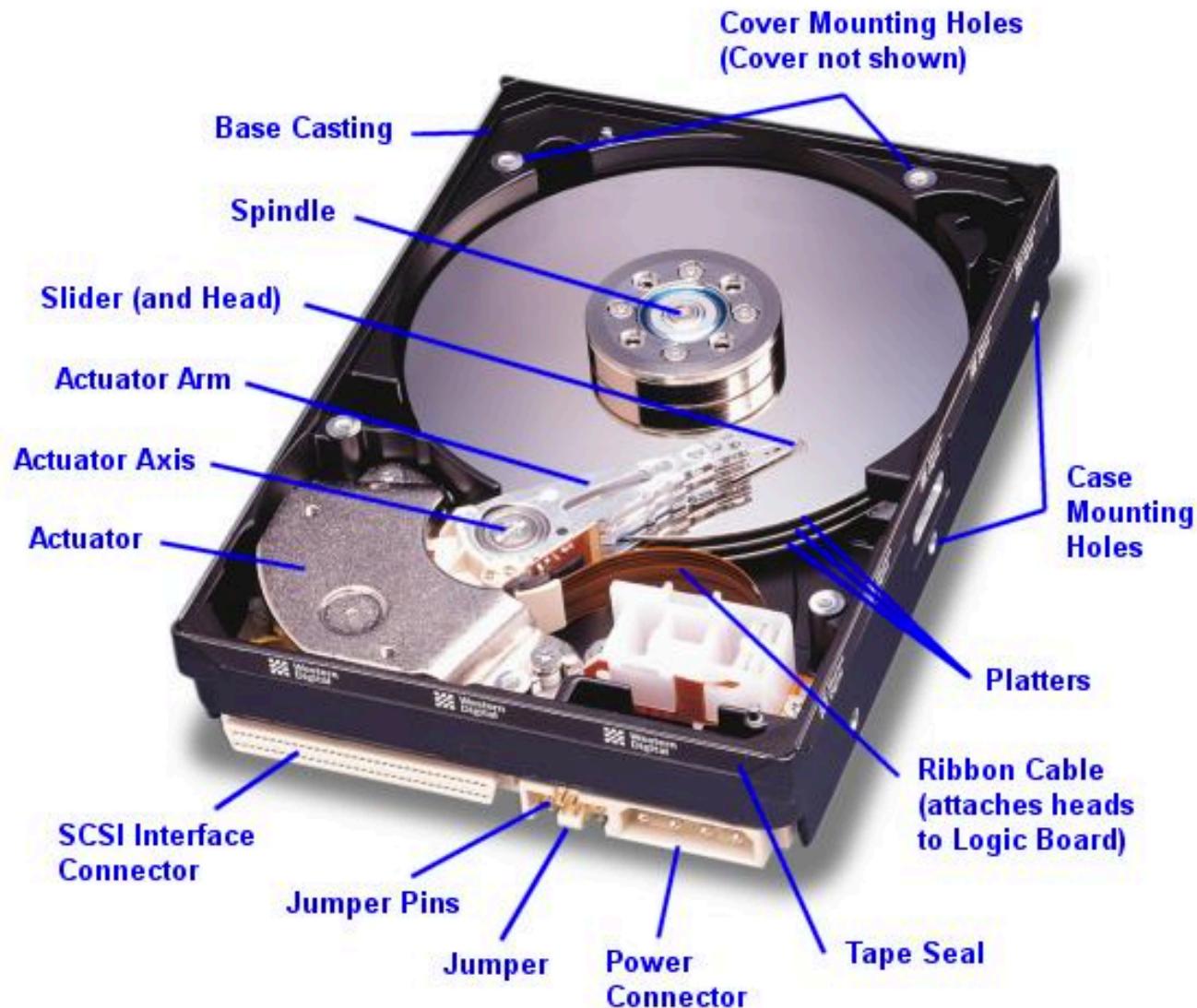
	PFN 0	PFN 1	PFN 2	PFN 3				
Physical Memory	Proc 0 [VPN 0]	Proc 1 [VPN 2]	Proc 1 [VPN 3]	Proc 2 [VPN 0]				
	Block 0	Block 1	Block 2	Block 3	Block 4	Block 5	Block 6	Block 7
Swap Space	Proc 0 [VPN 1]	Proc 0 [VPN 2]	[Free]	Proc 1 [VPN 0]	Proc 1 [VPN 1]	Proc 3 [VPN 0]	Proc 2 [VPN 1]	Proc 3 [VPN 1]

# Page Replacement

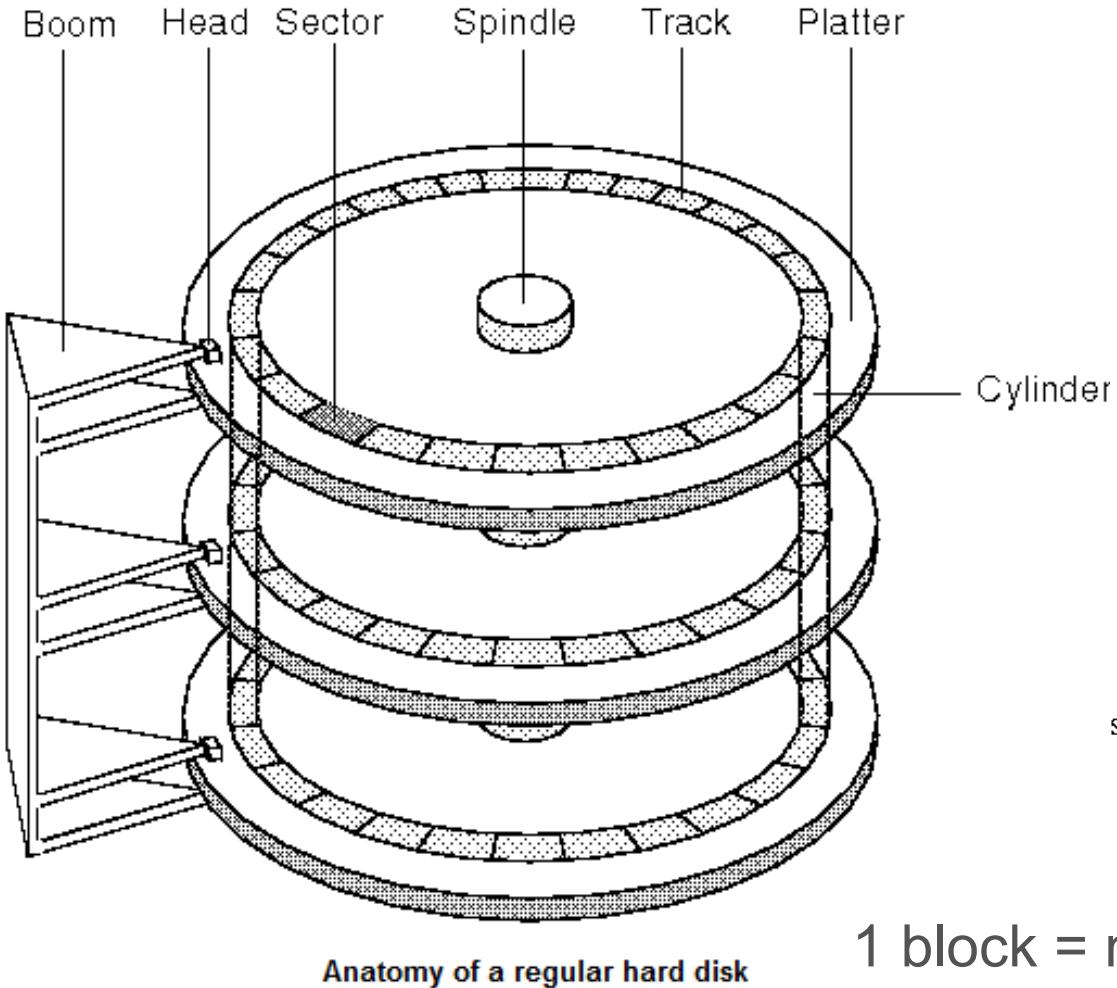
- ❖ Recall DRAM has page frames to hold page content; a process's address space may only have so many frames
- ❖ **Page Fault:** A page required by process is not in DRAM
  - ❖ OS intervenes to read page from disk to DRAM
  - ❖ If free page frame available in DRAM, all good
- ❖ **Page Replacement:** If no frame is free when page fault happens, OS must evict some occupied frame's page!
  - ❖ **Page Replacement Policy** (aka cache repl. policy): Algorithm that OS uses to tell what page to evict
  - ❖ Various policies exist with different *performance* and *complexity* tradeoffs: FIFO, MRU, LRU, etc. (later topic)

Optional: More on Magnetic Hard Disks  
Not included in syllabus

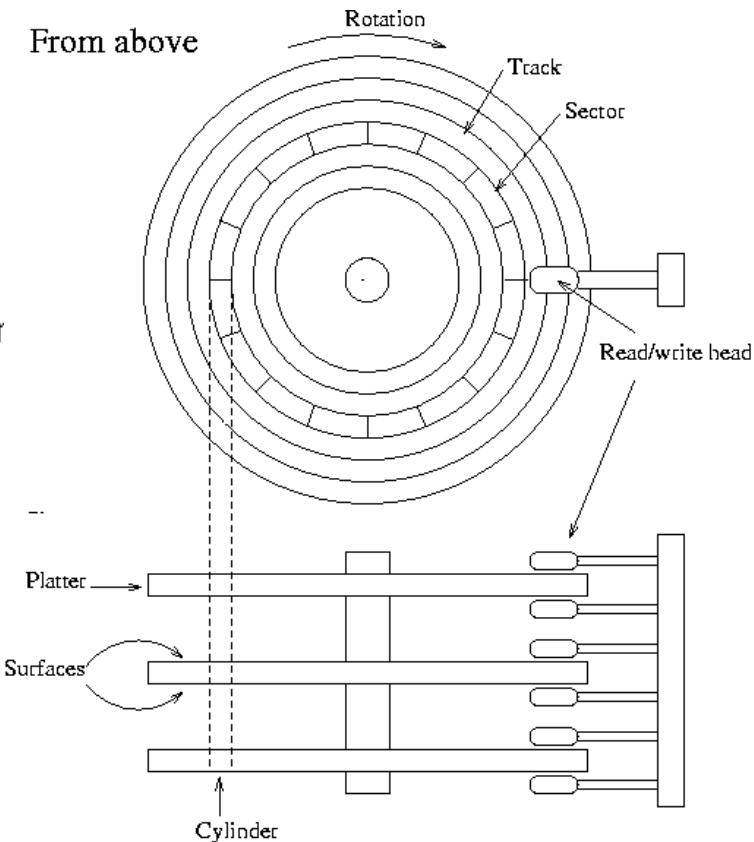
# Components of a Disk



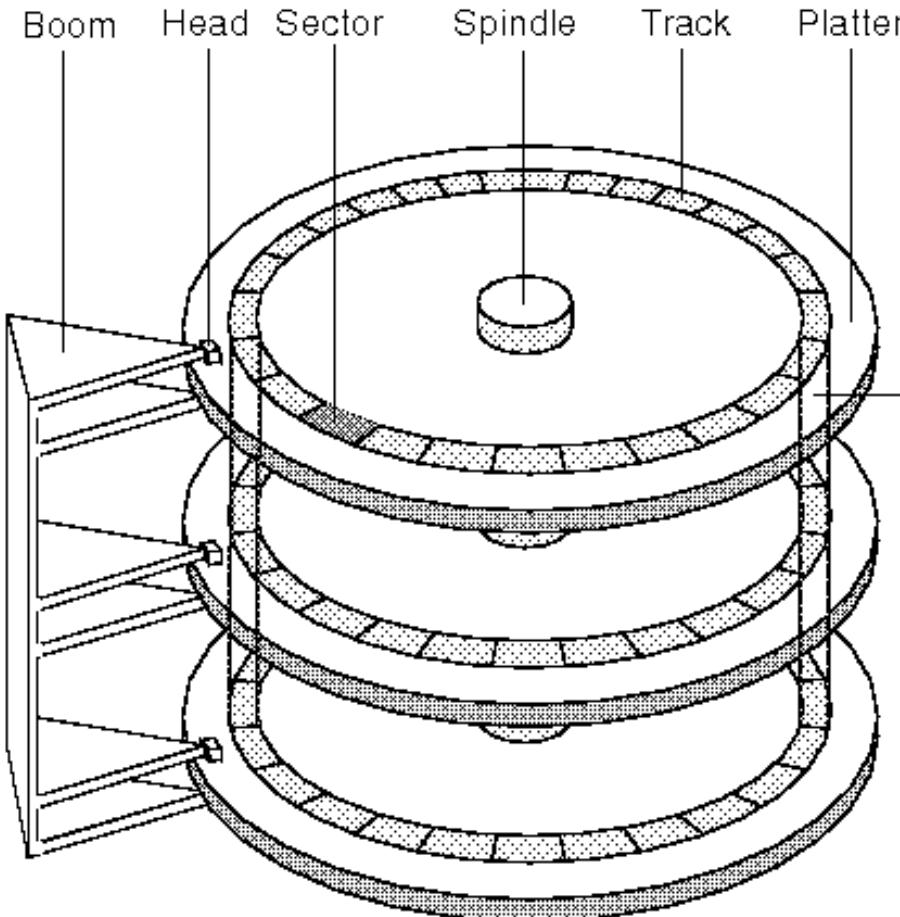
# Components of a Disk



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



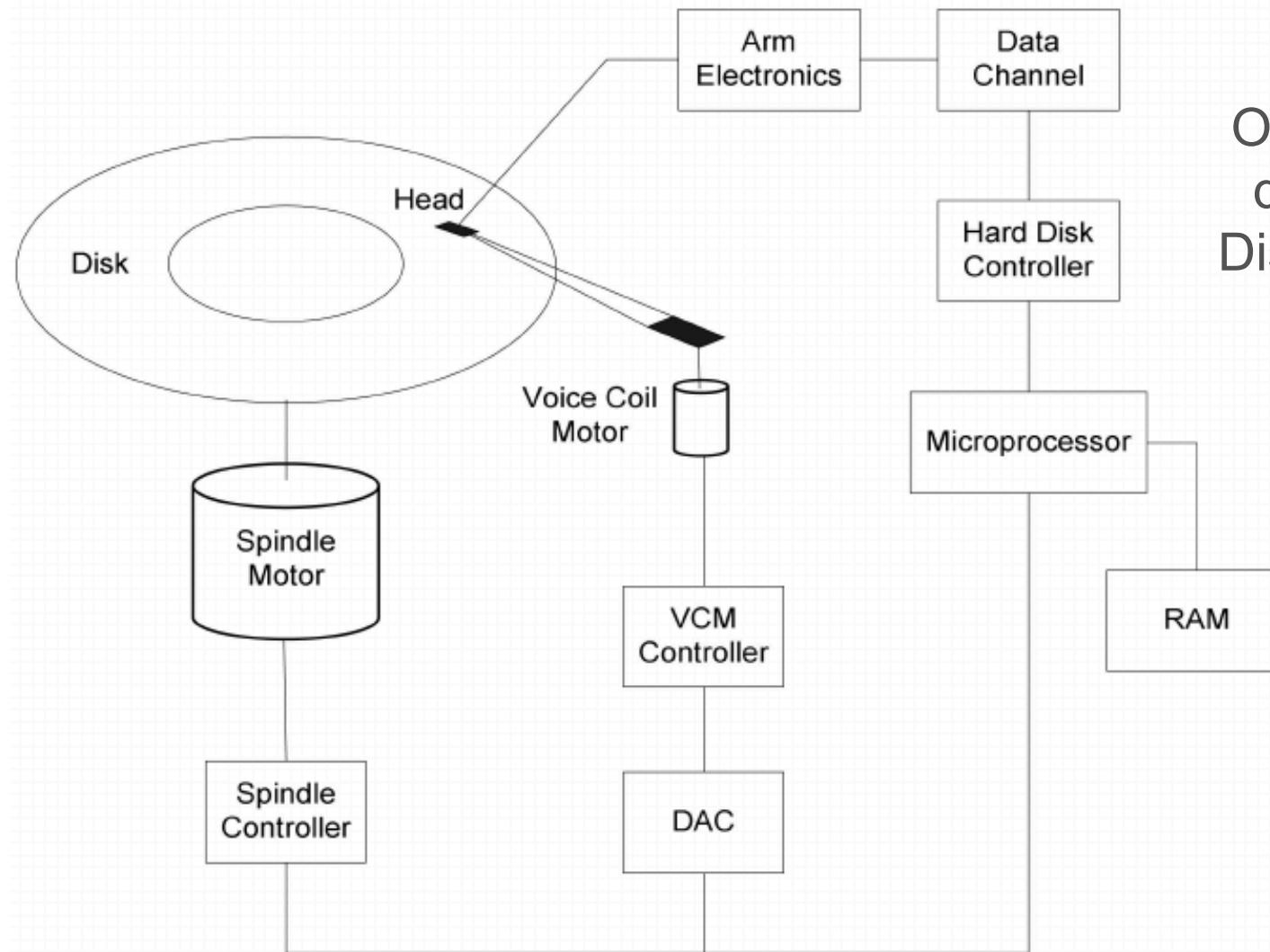
# How does a Disk Work?



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  
directly with  
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	\$50

# DSC 102

# Systems for Scalable Analytics

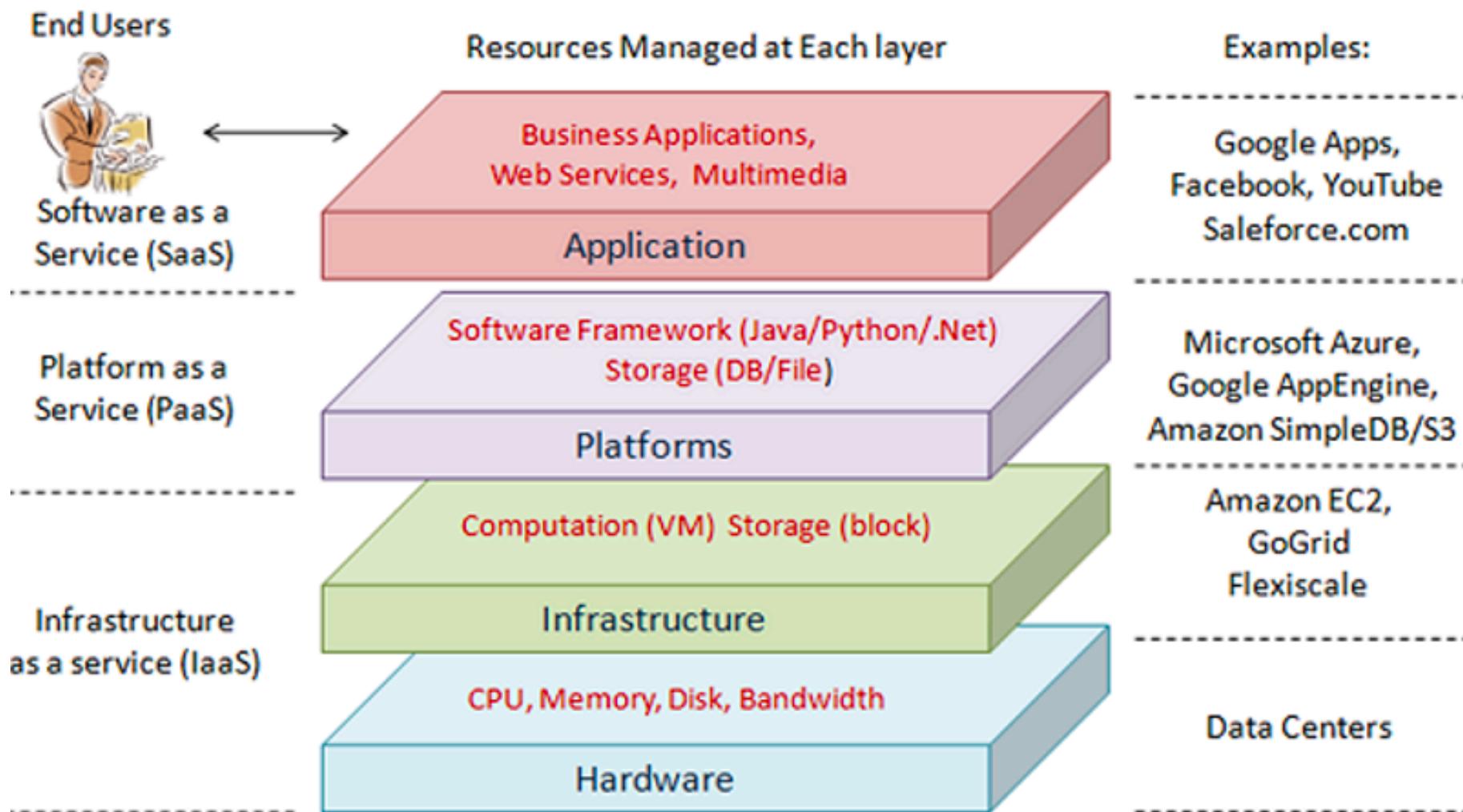
Arun Kumar

Topic 2: Basics of Cloud Computing

# Cloud Computing

- ❖ Compute, storage, memory, networking, etc. are virtualized and exist on *remote servers*; *rented* by application users
- ❖ Main pros of cloud vs on-premise clusters:
  - ❖ **Manageability:** Managing hardware is not user's problem
  - ❖ **Pay-as-you-go:** Fine-grained pricing economics based on actual usage (granularity: seconds to years!)
  - ❖ **Elasticity:** Can dynamically add or reduce capacity based on actual workload's demand
- ❖ Infrastructure-as-a-Service (IaaS); Platform-as-a-Service (PaaS); Software-as-a-Service (SaaS)

# Cloud Computing



# Examples of AWS Cloud Services

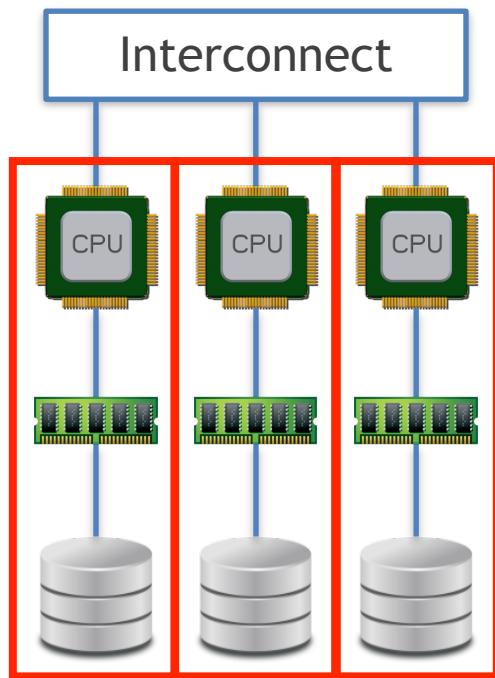
- ❖ **IaaS:**
  - ❖ **Compute:** EC2, ECS, Fargate, Lambda
  - ❖ **Storage:** S3, EBS, EFS, Glacier
  - ❖ **Networking:** CloudFront, VPC
- ❖ **PaaS:**
  - ❖ **Database/Analytics Systems:** Aurora, Redshift, Neptune, ElastiCache, DynamoDB, Timestream, EMR, Athena
  - ❖ **Blockchain:** QLDB; **IoT:** Greengrass
- ❖ **SaaS:**
  - ❖ **ML/AI:** SageMaker, Elastic Inference, Lex, Polly, Translate, Transcribe, Textract, Rekognition, Ground Truth
  - ❖ **Business Apps:** Chime, WorkDocs, WorkMail

# Evolution of Cloud Infrastructure

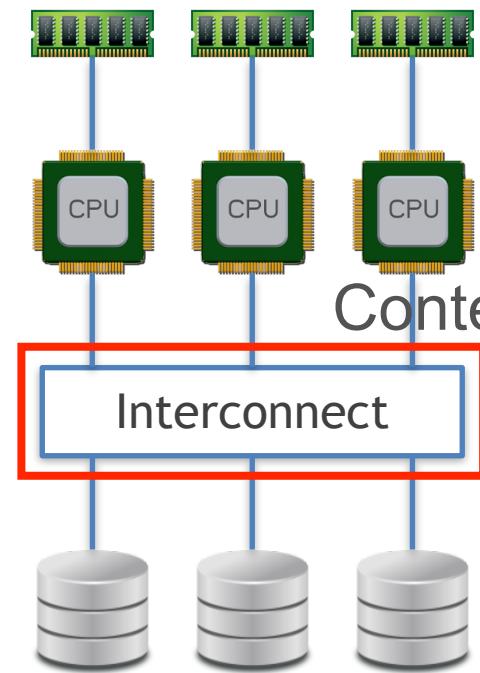
- ❖ **Data Center:** Physical space from which a cloud is operated
- ❖ **3 generations of data centers/clouds:**
  - ❖ **Cloud 1.0 (Past):** Networked servers; user rents servers (time-sliced access) needed for data/software
  - ❖ **Cloud 2.0 (Current):** “Virtualization” of networked servers; user rents amount of resource capacity; cloud provider has a lot more flexibility on provisioning (multi-tenancy, load balancing, more elasticity, etc.)
  - ❖ **Cloud 3.0 (Ongoing Research):** “Serverless” and disaggregated resources all connected to fast networks

# 3 Paradigms of Multi-Node Parallelism

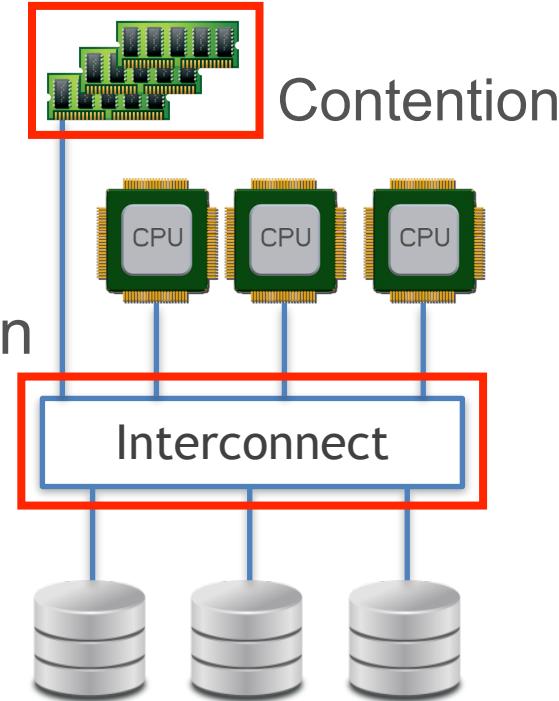
Independent Workers



Shared-Nothing  
Parallelism



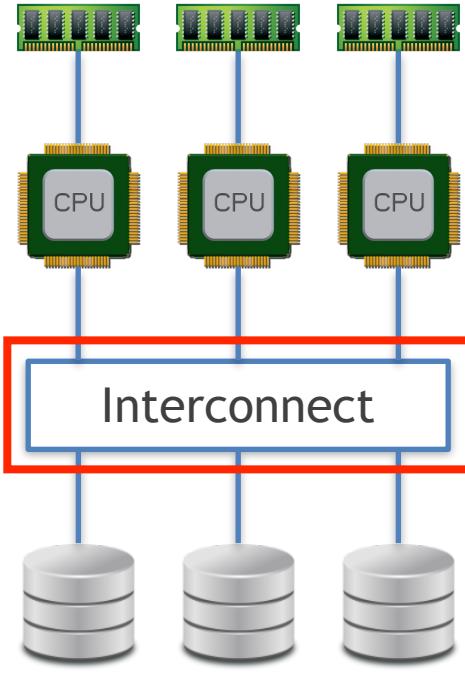
Shared-Disk  
Parallelism



Shared-Memory  
Parallelism

Most parallel RDBMSs (Teradata, Greenplum, Redshift),  
Hadoop, and Spark use shared-nothing parallelism

# Revisiting Parallelism in the Cloud

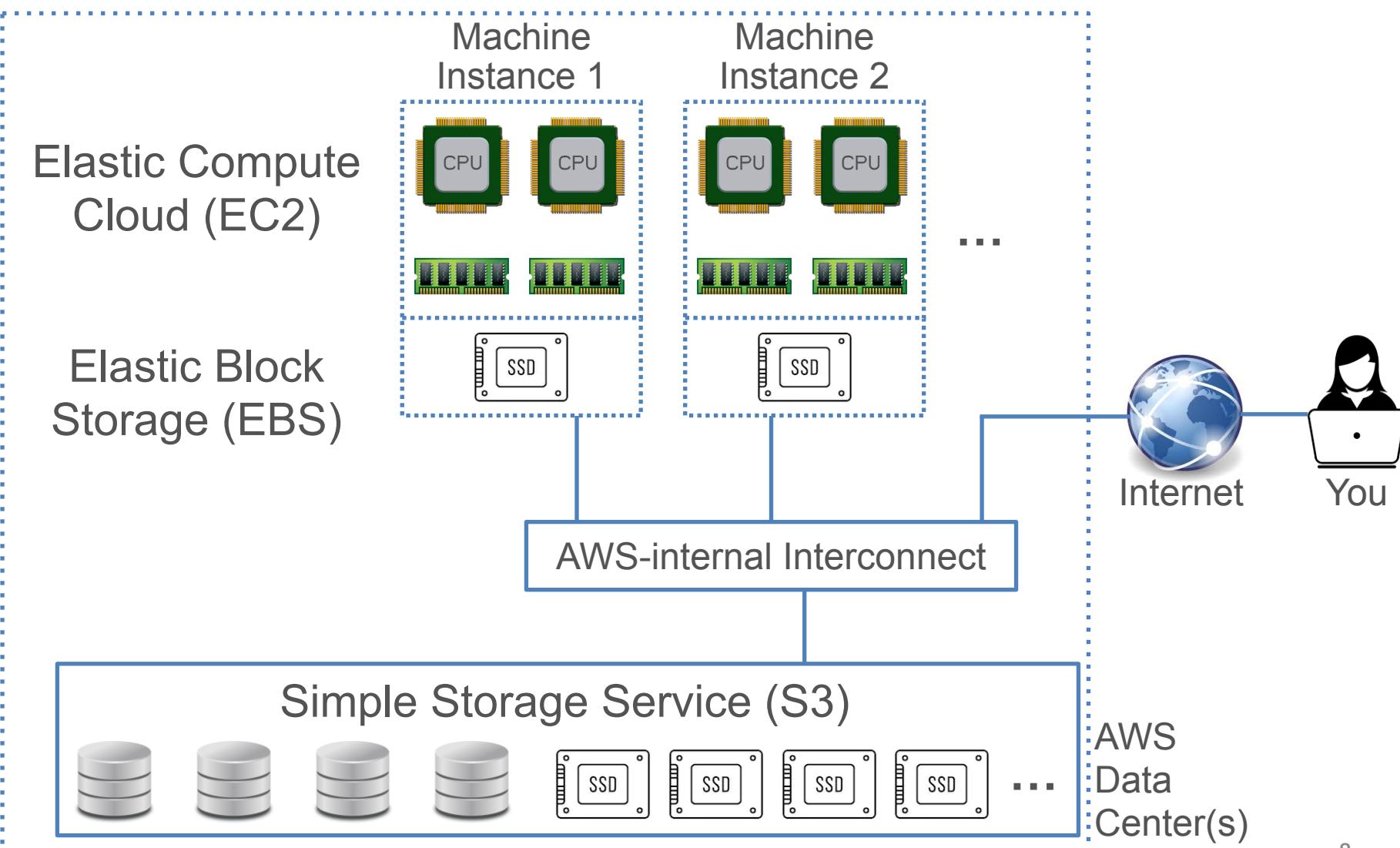


Shared-Disk  
Parallelism

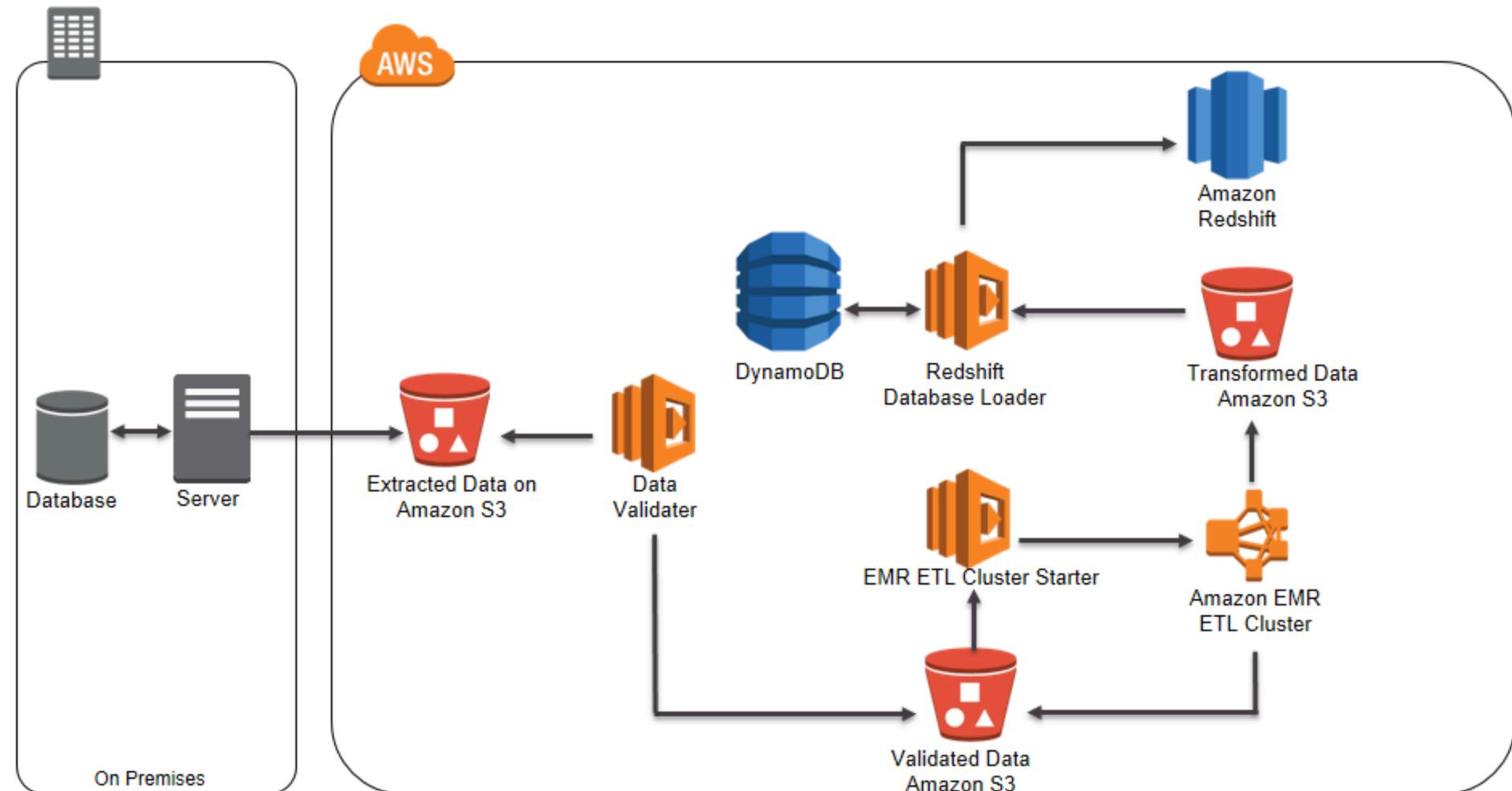
Modern networks in data centers have become much faster: 100GbE to even TbE!

- ❖ **Decoupling** of compute+memory from storage is common in cloud
  - ❖ *Hybrids* of shared-disk parallelism + shared-nothing parallelism
  - ❖ E.g, store datasets on S3 and read as needed to local EBS

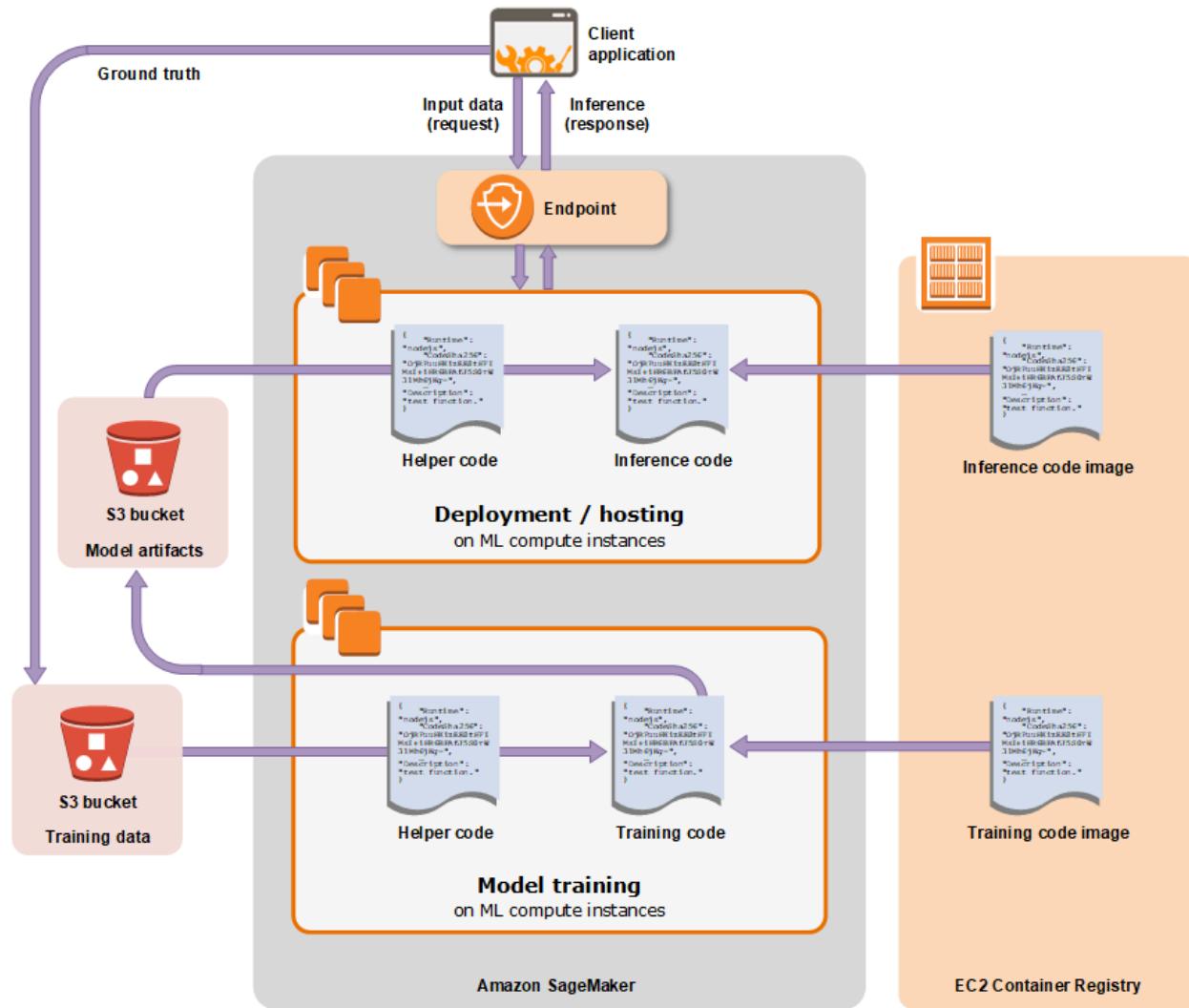
# Example: AWS Services for PA1



# Example: AWS DB/Analytics Services



# Example: AWS ML Services



# New Cloud Renting Paradigms

- ❖ Cloud 2.0's flexibility enables radically different paradigms
- ❖ AWS example below; Azure and GCP have similar gradations

## AWS EC2 Consumption Models

### On-Demand

Pay for compute capacity by the second or hour with no long-term commitments

For spiky workloads or to define needs initially



### Reserved

Significant discount compared to On-Demand instance pricing

Steady state applications or predictable usage, databases



### Spot

Spare EC2 capacity for up to 90% off the On-Demand price.

For fault tolerant, instance flexible or time-insensitive workloads



© 2018, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

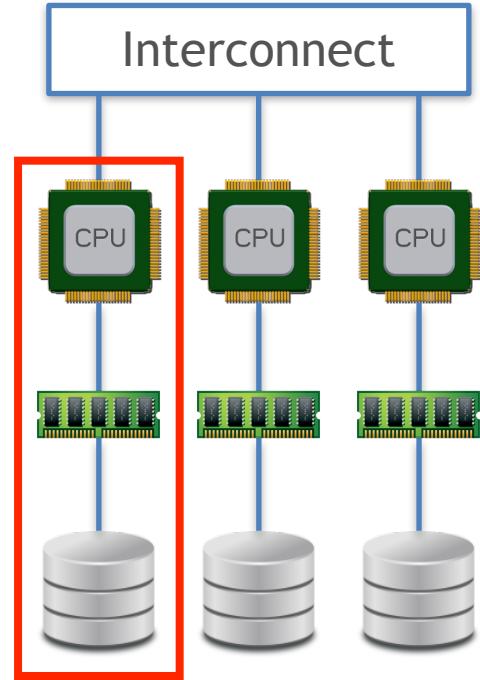
# More on Spot vs On-Demand

	<b>Spot Instances</b>	<b>On-Demand Instances</b>
Launch time	Can only be launched immediately if the Spot Request is active and capacity is available.	Can only be launched immediately if you make a manual launch request and capacity is available.
Available capacity	If capacity is not available, the Spot Request continues to automatically make the launch request until capacity becomes available.	If capacity is not available when you make a launch request, you get an insufficient capacity error (ICE).
Hourly price	The hourly price for Spot Instances varies based on demand.	The hourly price for On-Demand Instances is static.
Rebalance recommendation	The signal that Amazon EC2 emits for a running Spot Instance when the instance is at an elevated risk of interruption.	You determine when an On-Demand Instance is interrupted (stopped, hibernated, or terminated).
Instance interruption	You can stop and start an Amazon EBS-backed Spot Instance. In addition, the Amazon EC2 Spot service can <a href="#">interrupt</a> an individual Spot Instance if capacity is no longer available, the Spot price exceeds your maximum price, or demand for Spot Instances increases.	You determine when an On-Demand Instance is interrupted (stopped, hibernated, or terminated).

# New Cloud Renting Paradigms

Such bundling means some applications might under-utilize some resources!

- ❖ **Serverless** paradigm gaining traction for some applications, e.g., online ML prediction serving on websites
- ❖ User gives a program (function) to run and specifies CPU and DRAM needed
- ❖ Cloud provider abstracts away all resource provisioning entirely
- ❖ Higher resource efficiency; much cheaper, often by 10x vs Spot instances
- ❖ Aka *Function-as-a-Service* (FaaS)



Shared-Nothing  
Parallelism

# Car Analogy for Serverless Cloud



**Own a car**  
(Bare metal servers)



**Rent a car**  
(VPS)

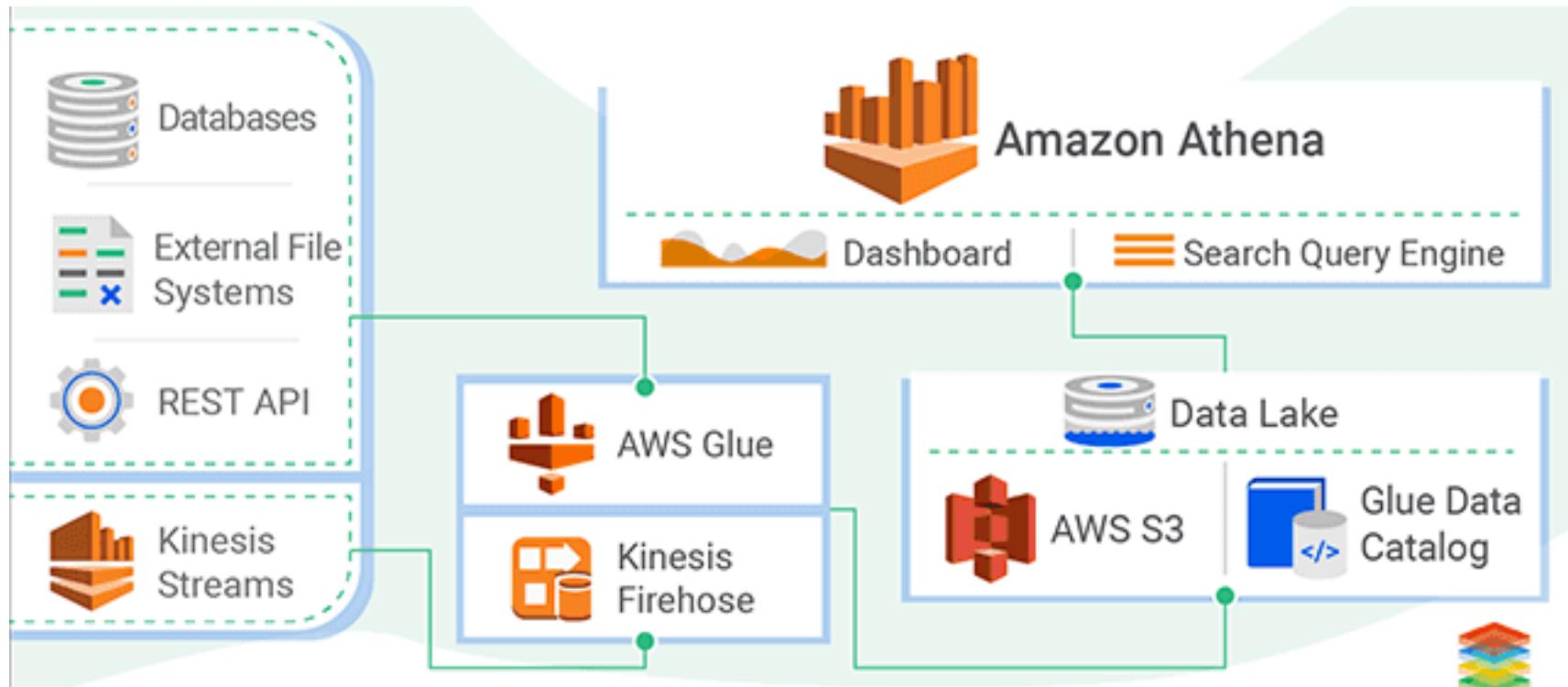


**City car-sharing**  
(Serverless)

Cars are parked **95%** of the time ([loige.link/car-parked-95](https://loige.link/car-parked-95))

**How much do you use the car?**

# Example: Serverless RDBMS on AWS

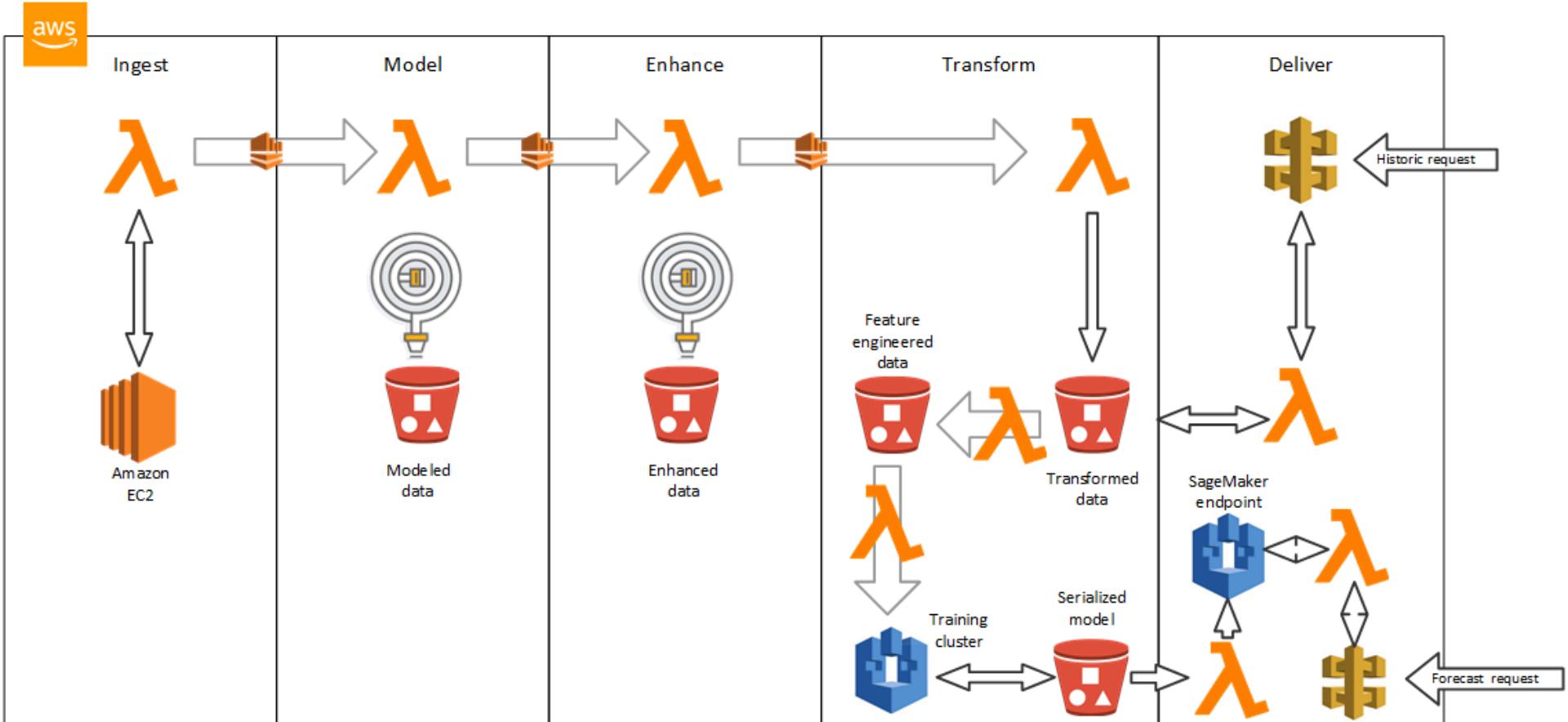


Remote read of  
data from S3

Schema-on-read  
Many data formats

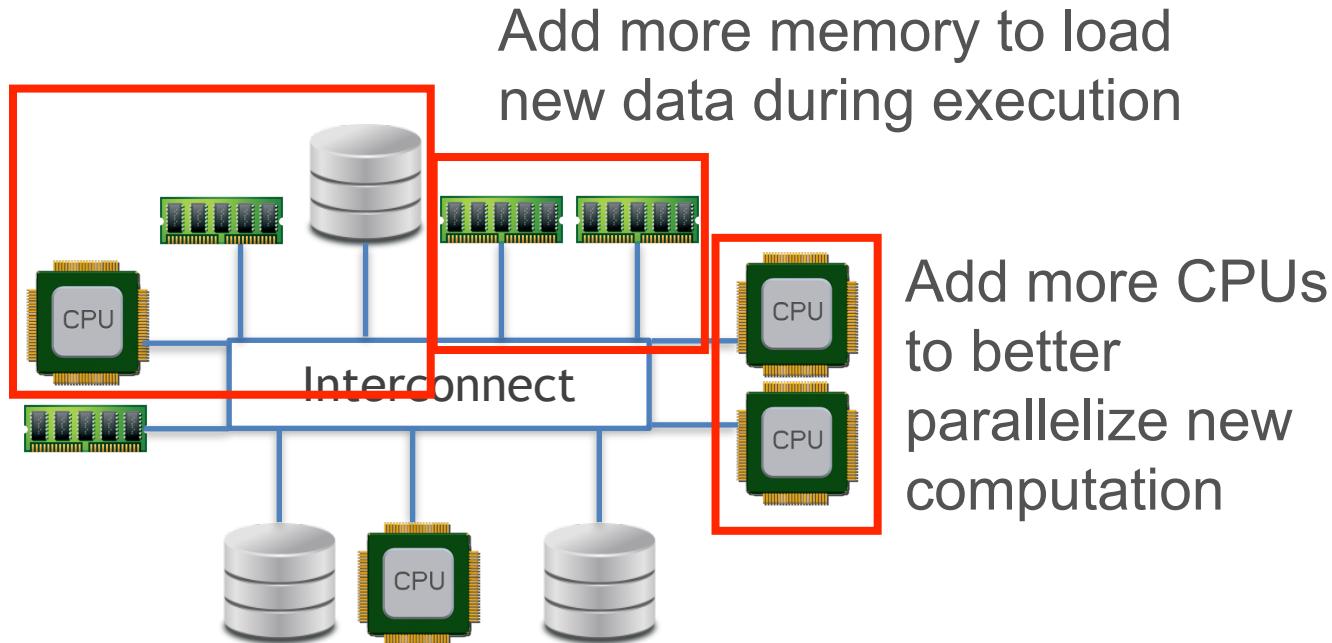
Simple interactive  
queries

# Example: Serverless ML app. on AWS



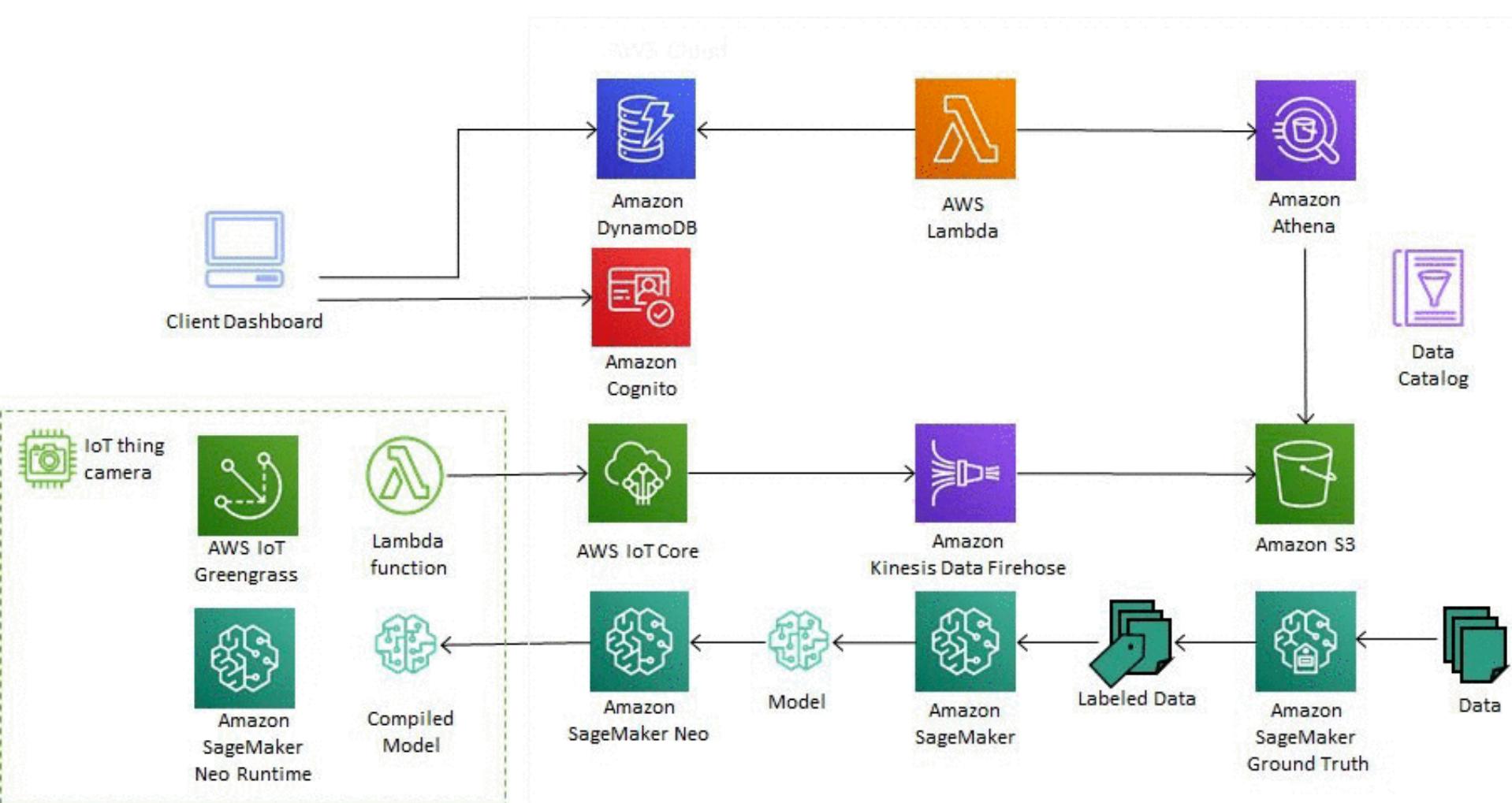
# Disaggregation: Glimpse into the Future?

- ❖ Logical next step in serverless direction: full **resource disaggregation**! That is, compute, memory, storage, etc. are all network-attached and elastically added/removed



**Ongoing Research:** Fulfill this promise with low latency!

# Example: AWS services for IoT app.



# OMG, is all this complexity worth it?!

- ❖ Depends on user's/application's Pareto tradeoffs! :)
- ❖ **On-premise** cluster are still common in large enterprises, healthcare, and academia; “hybrid clouds” too
- ❖ Recall main pros of cloud: manageability, cost, and elasticity
- ❖ Some main cons of cloud (vs on-premise):
  - ❖ **Complexity** of composing cloud APIs and licenses; data scientists must keep relearning; “CloudOps” teams
  - ❖ **Cost** over time can *crossover* and make it costlier!
  - ❖ Easier to **waste money** accidentally on the fly
  - ❖ “**Lock-in**” by cloud vendor
  - ❖ **Privacy, security, and governance** concerns
  - ❖ **Internet disruption or unplanned downtime**, e.g., AWS outage in 2015 made Netflix, Tinder, etc. unavailable! :)

# OMG, is all this complexity worth it?!



U.S. Department of Defense

News ▾ Spotlights ▾ About ▾

Release

IMMEDIATE RELEASE

## Future of the Joint Enterprise Defense Infrastructure Cloud Contract

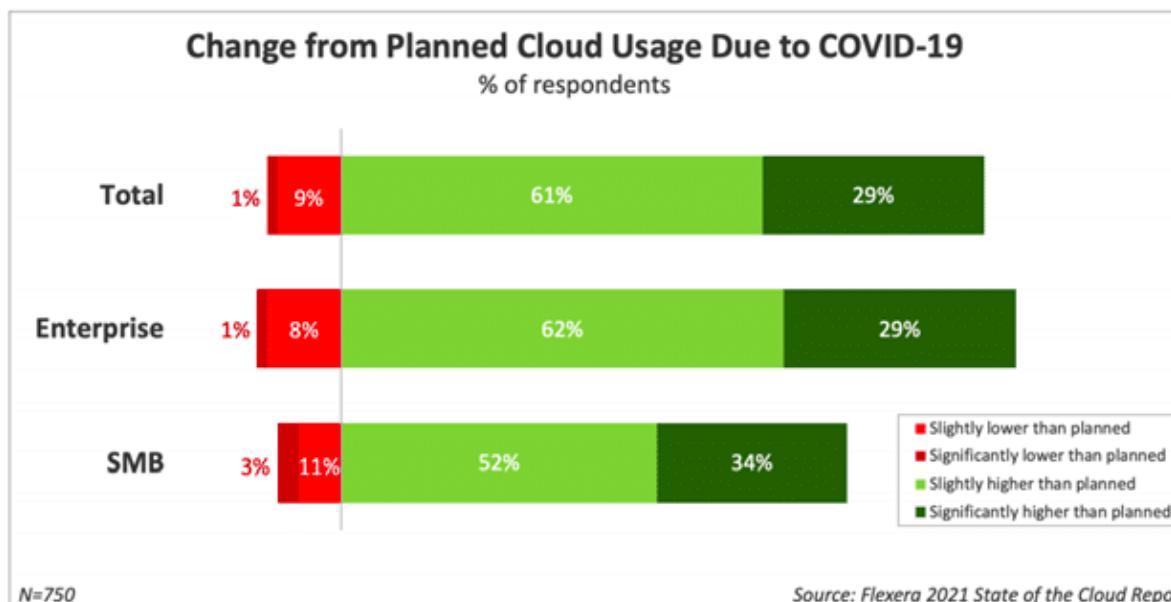
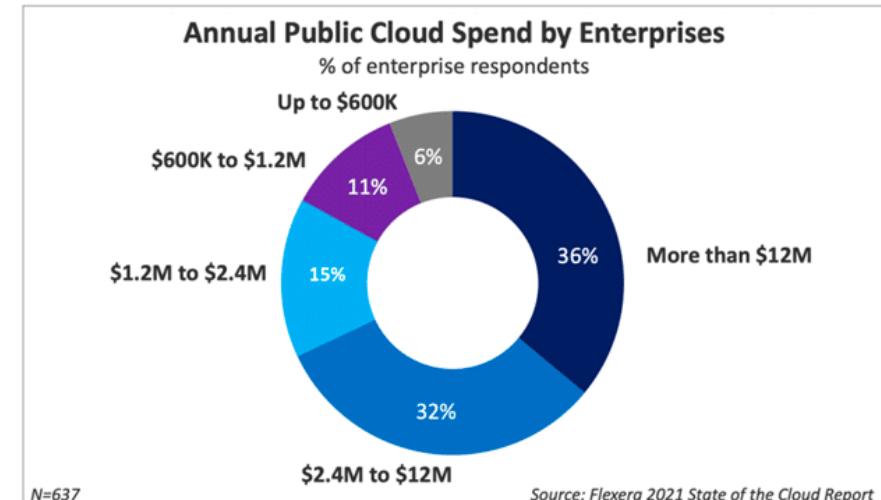
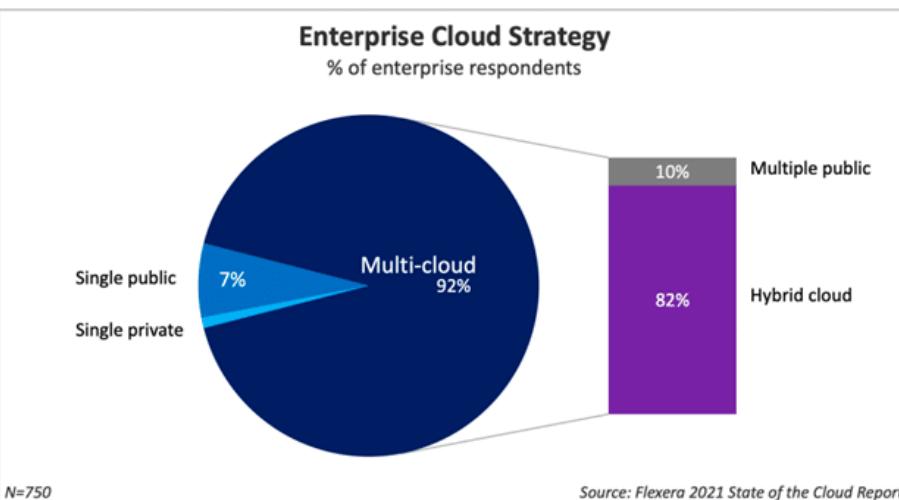
JULY 6, 2021

f  t 

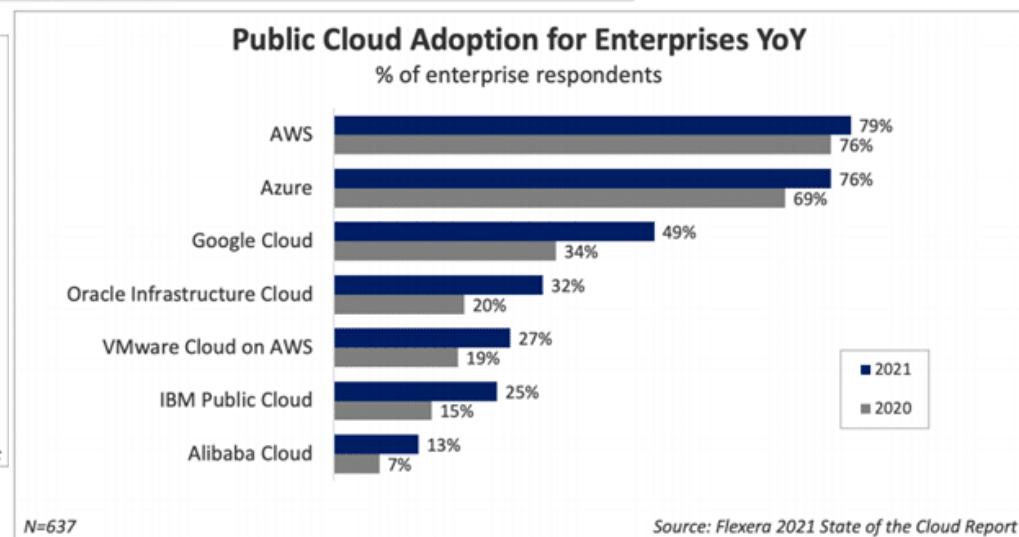
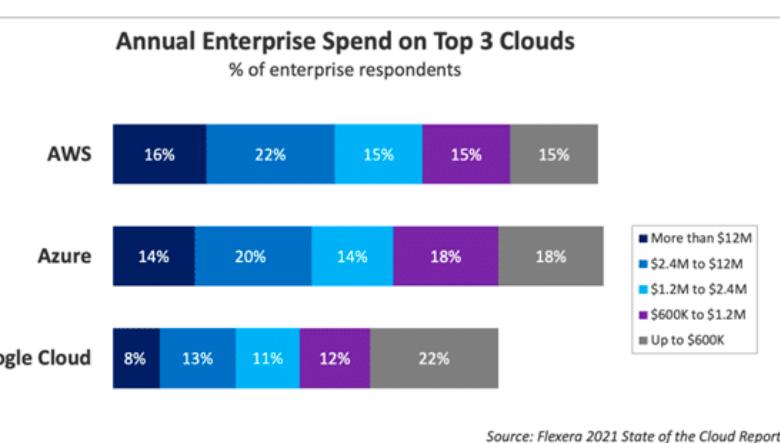
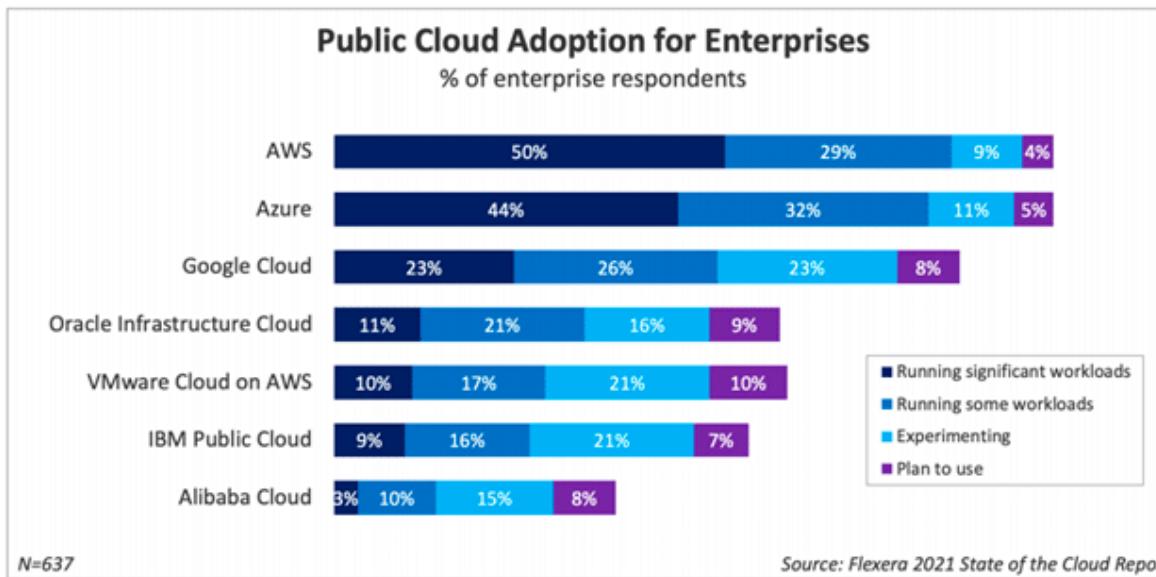
---

Today, the Department of Defense (DoD) canceled the Joint Enterprise Defense Infrastructure (JEDI) Cloud solicitation and initiated contract termination procedures. The Department has determined that, due to evolving requirements, increased cloud conversancy, and industry advances, the JEDI Cloud contract no longer meets its needs. The Department continues to have unmet cloud capability gaps for enterprise-wide, commercial cloud services at all three classification levels that work at the tactical edge, at scale -- these

# The State of the Cloud Survey

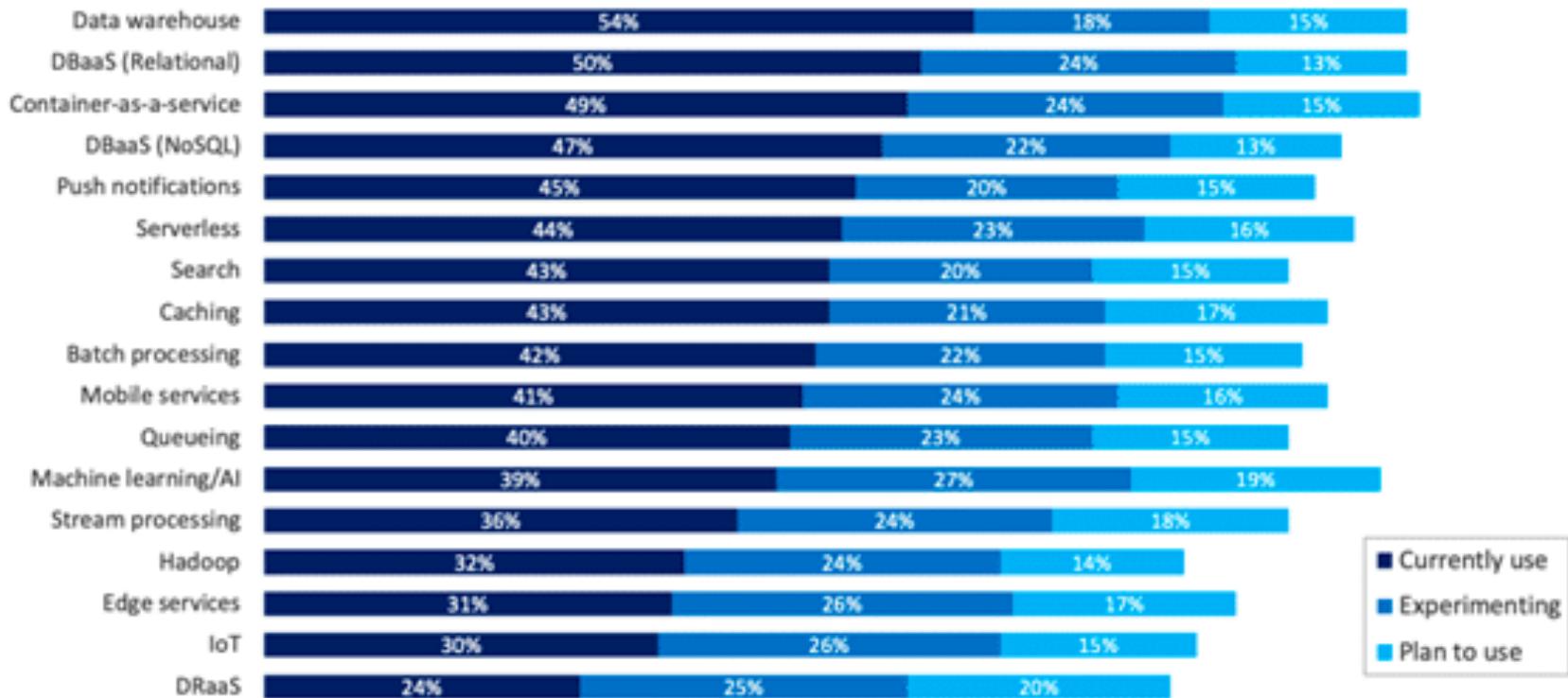


# The State of the Cloud Survey



# The State of the Cloud Survey

**Public Cloud Services Used**  
% of all respondents



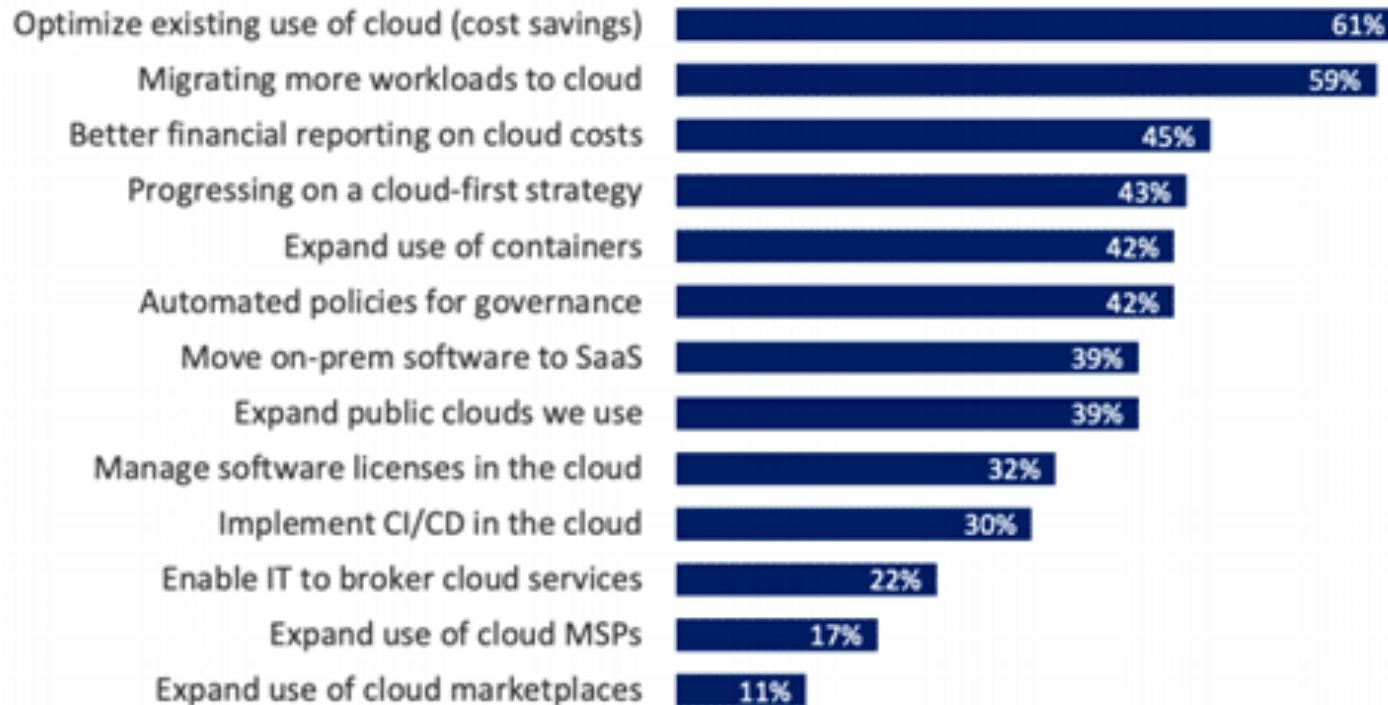
N=750

Source: Flexera 2021 State of the Cloud Report

# The State of the Cloud Survey

## Top Cloud Initiatives for 2021

% of all respondents



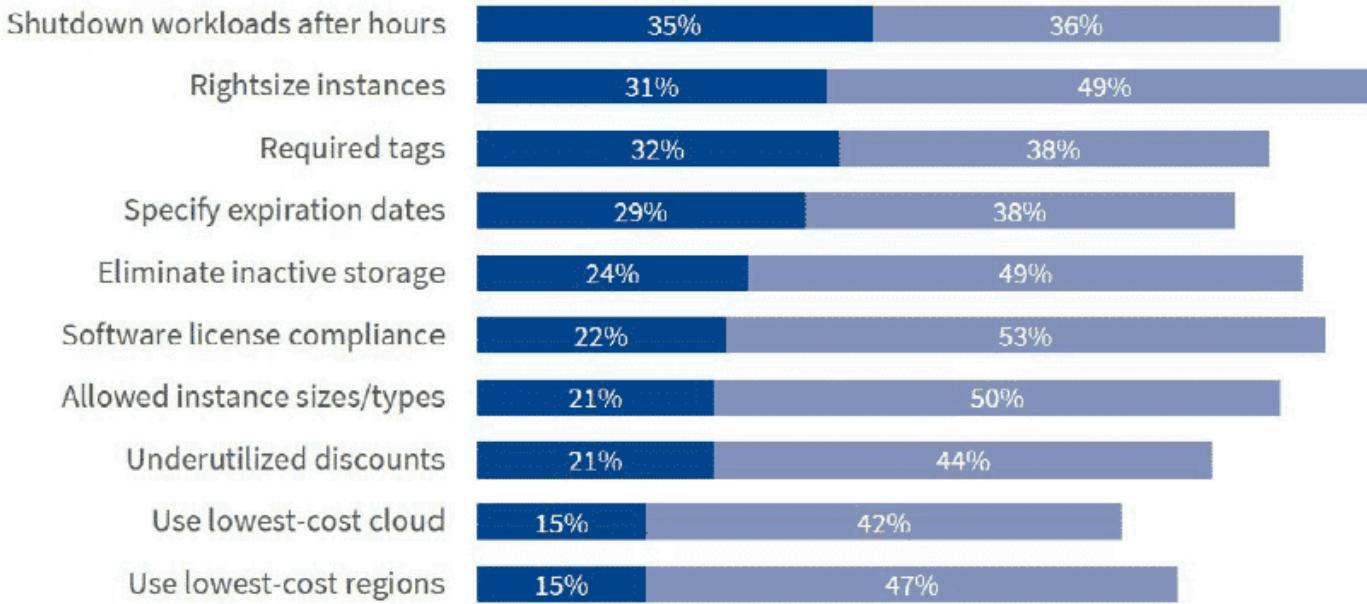
N=750

Source: Flexera 2021 State of the Cloud Report

# The State of the Cloud Survey

## Policies to Optimize Cloud Costs

% of Respondents



Source: RightScale 2019 State of the Cloud Report from Flexera

# Review Questions

1. What are the 3 main layers of a typical cloud? Give examples of AWS services in each layer. Which ones do your PAs use?
2. What is a benefit of separating PaaS from SaaS in cloud?
3. Briefly explain 1 pro and 1 con of Shared Disk Parallelism vs Shared Nothing Parallelism.
4. Briefly explain 1 pro and 1 con of On-Demand vs Spot instances on AWS.
5. What is so “great” about the serverless cloud anyway?
6. What is so great about “resource disaggregation” in future clouds?
7. Briefly explain 2 pros and 2 cons of cloud vs on-premise clusters.

# DSC 102

# Systems for Scalable Analytics

Arun Kumar

Topic 3: Parallel and Scalable Data Processing

Part 1: Parallelism Basics

Ch. 9.4, 12.2, 14.1.1, 14.6, 22.1-22.3, 22.4.1, 22.8 of Cow Book

Ch. 5, 6.1, 6.3, 6.4 of MLSys Book

*Q: Why bother with large-scale data?  
Why does sampling not suffice?*



# Large-Scale Data in Astronomy

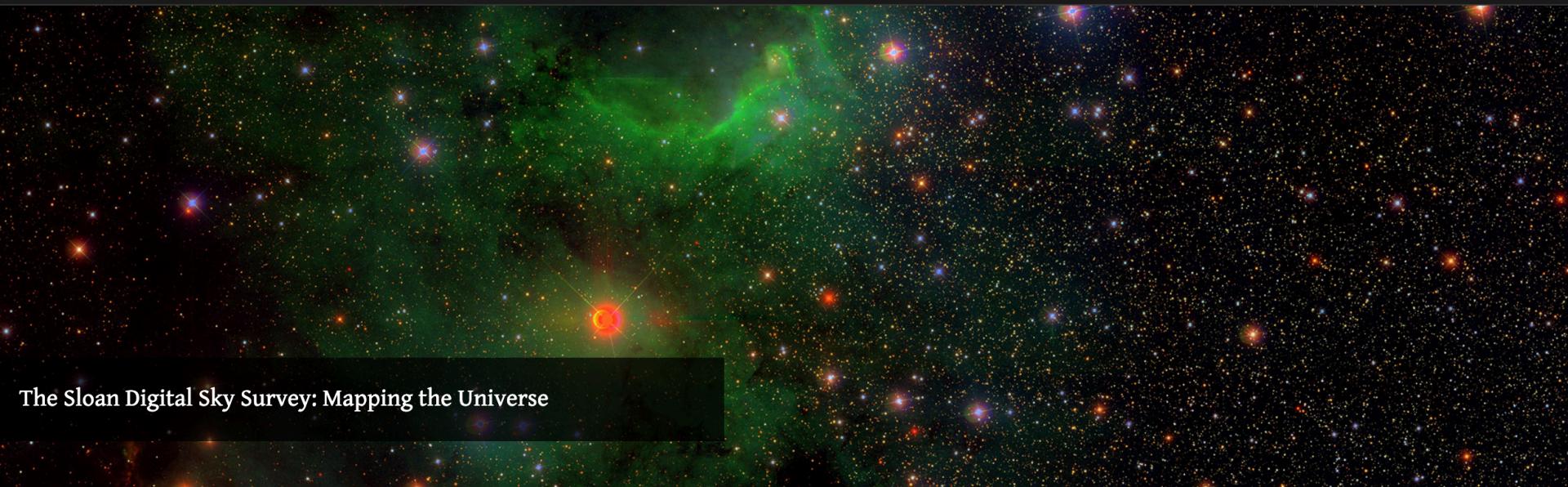


This is Data Release 16.

Data Surveys Instruments Collaboration Results Education The Future Contact

Search www.sdss.org

Search



The Sloan Digital Sky Survey: Mapping the Universe

The Sloan Digital Sky Survey has created the most detailed three-dimensional maps of the Universe ever made, with deep multi-color images of one third of the sky, and spectra for more than three million astronomical objects. Learn and explore all phases and surveys—past, present, and future—of the SDSS.

High-res. images: ~200 GB per day since 2000 (1PB+)  
Astronomers can study complex galactic evolution behaviors



# Large-Scale Data in Genomics

## UNDERSTANDING PRECISION MEDICINE

In precision medicine, patients with tumors that share the same genetic change receive the drug that targets that change, no matter the type of cancer.



Precision Medicine is becoming a reality

Analyze genomes across cohorts and prescribe targeted drugs and treatments

~3GB genome per human  
~1EB for USA

NETFLIX ORIGINAL

# STRANGER THINGS

95% Match 2017 2 Seasons 4K Ultra HD 5.1

When a young boy vanishes, a small town uncovers a mystery involving secret experiments, terrifying supernatural forces and one strange little girl.

*Winona Ryder, David Harbour, Matthew Modine*  
TV Shows, TV Sci-Fi & Fantasy, Teen TV Shows



## Popular on Netflix



## Recently Watched



# Large-Scale Data in E-commerce

**Everything is a Recommendation**



**Over 80% of what people watch comes from our recommendations**

**Recommendations are driven by Machine Learning**

6

Log all user behavior (views, clicks, pauses, searches, etc.)

Recommender systems combine TBs of data from all users and movies to deliver a tailored experience

8

# Large-Scale Data in Computer Vision

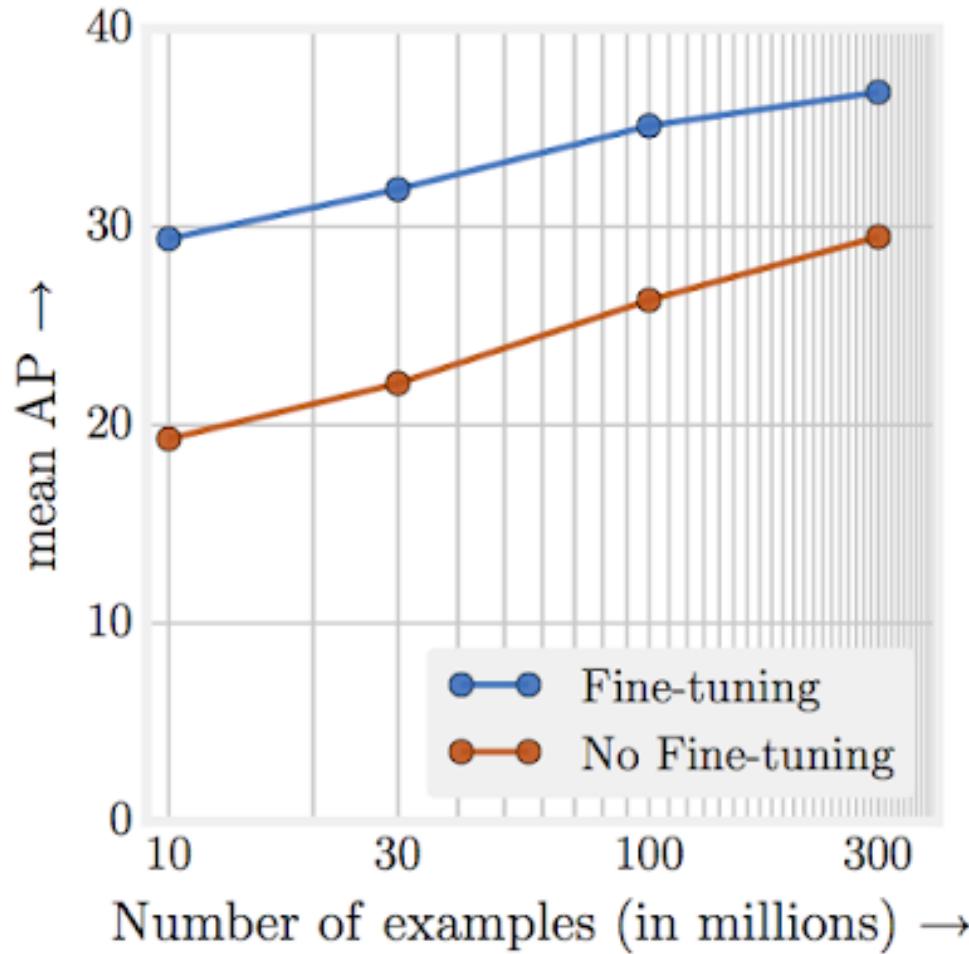


10million+ images labeled (20,000 classes) by crowdsourcing

>500GB uncompressed as tensors

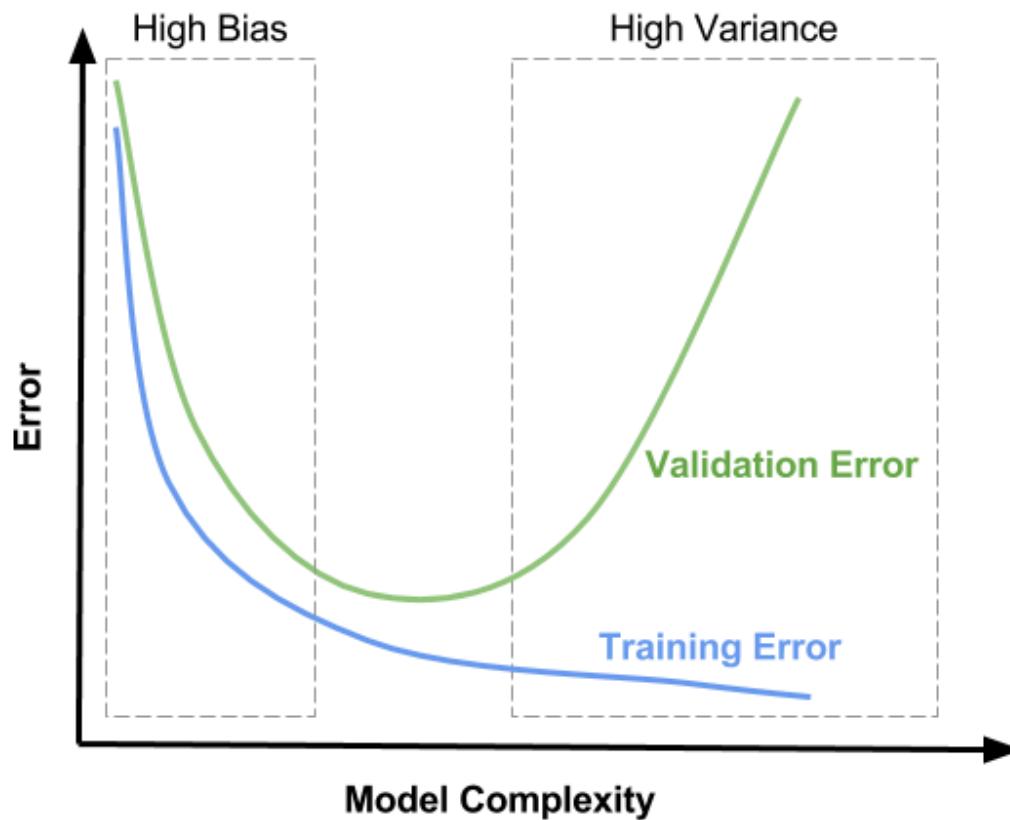
Harbinger of deep learning revolution

# “The Unreasonable Effectiveness of Data”



When **prediction target complexity** is high, more training data coupled with more complex models yield higher accuracy as number of training examples grows

# Bias-Variance Tradeoff of ML



**High Bias:** Roughly, model is not rich enough to represent data

**High Variance:** Model *overfits* to given data; poor *generalization*

**Large-scale training data lowers variance and raises accuracy!**

# Why Large-Scale Data?

- ❖ Large-scale data is a game changer in data science:
  - ❖ Enables **study of granular phenomena** in sciences, businesses, etc. not possible before
  - ❖ Enables **new applications** and personalization/customization
  - ❖ Enables more **complex ML prediction targets** and mitigates variance to offer **high accuracy**
- ❖ Hardware has kept pace to power the above:
  - ❖ Storage capacity has exploded (PB clusters)
  - ❖ Compute capacity has grown (multi-core, GPUs, etc.)
  - ❖ DRAM capacity has grown (10GBs to TBs)
  - ❖ Cloud computing is “democratizing” access to hardware; SaaS

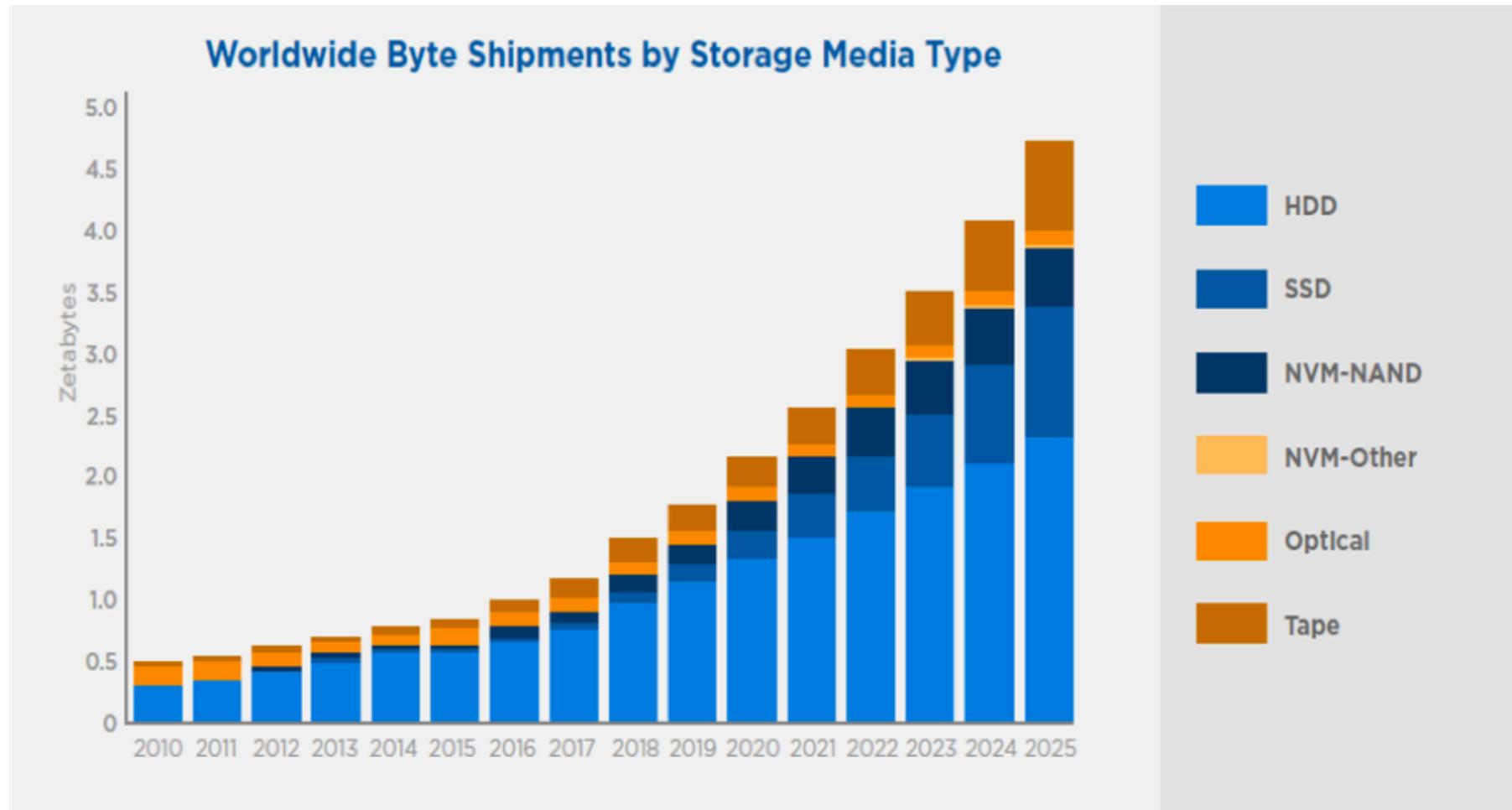
# “Big Data”

- ❖ Marketing term; think “Big” as in “Big Oil” or “Big Government” or “Big Tech”, not “big building”
  - ❖ Became popular in late 2000s to early 2010s
  - ❖ Wikipedia says: “Data that is so large and complex that existing toolkits [read RDBMSs!] are not adequate”
- ❖ Typical characterization by 3 Vs:
  - ❖ **Volume**: larger than single-node DRAM
  - ❖ **Variety**: relations, docs, tweets, multimedia, etc.
  - ❖ **Velocity**: high generation rate, e.g., sensors, surveillance

# Why “Big Data” now? 1. Applications

- ❖ New “data-driven mentality” in almost all human endeavors:
- ❖ **Web**: search, e-commerce, e-mails, social media
- ❖ **Science**: satellite imagery, CERN’s LHC, document corpora
- ❖ **Medicine**: pharmacogenomics, precision medicine
- ❖ **Logistics**: sensors, GPS, “Internet of Things”
- ❖ **Finance**: high-throughput trading, monitoring
- ❖ **Humanities**: digitized books/literature, social media
- ❖ **Governance**: e-voting, targeted campaigns, NSA ☺
- ❖ ...

# Why “Big Data” now? 2. Storage



Source: Data Age 2025, sponsored by Seagate with data from IDC Global DataSphere, Nov 2018

*To analyze large-scale data, parallel and scalable data systems are indispensable!*

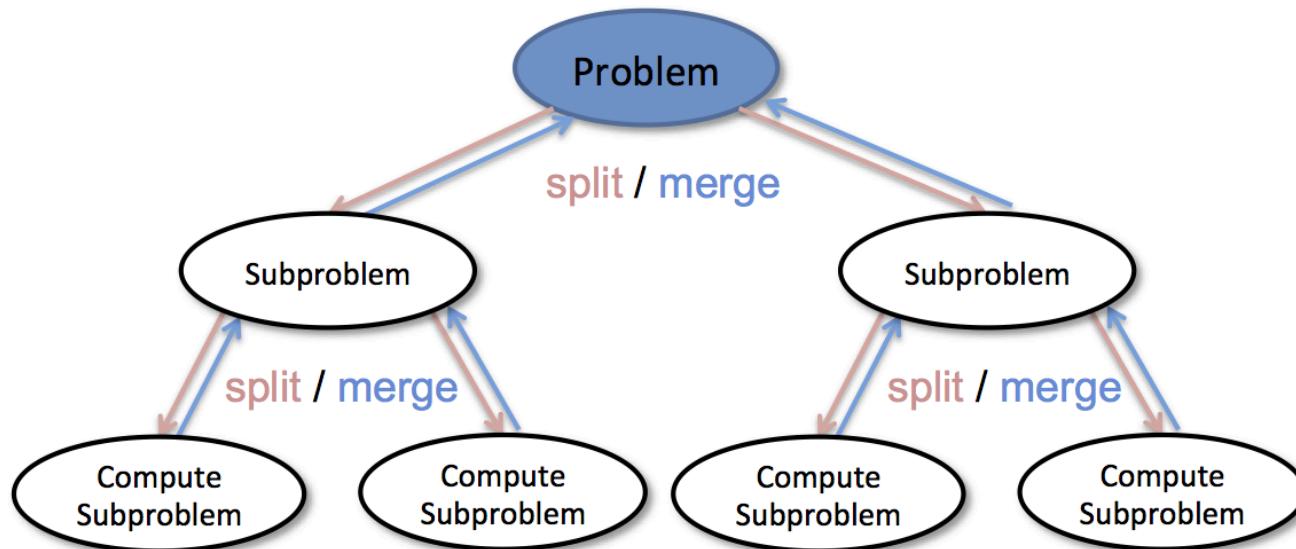
# Outline

- ➔ Basics of Parallelism
  - ❖ Task Parallelism; Dask
  - ❖ Single-Node Multi-Core; SIMD; Accelerators
- ❖ Basics of Scalable Data Access
  - ❖ Paged Access; I/O Costs; Layouts/Access Patterns
  - ❖ Scaling Data Science Operations
- ❖ Data Parallelism: Parallelism + Scalability
  - ❖ Data-Parallel Data Science Operations
  - ❖ Optimizations and Hybrid Parallelism

# Parallel Data Processing

**Central Issue:** Workload takes too long for one processor!

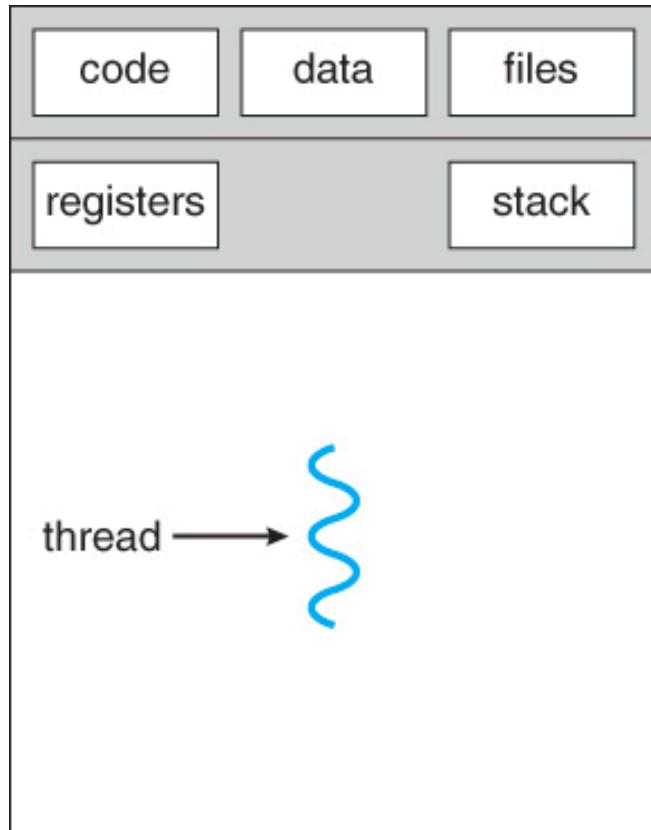
**Basic Idea:** Split up workload across processors and perhaps also across machines/workers (aka “Divide and Conquer”)



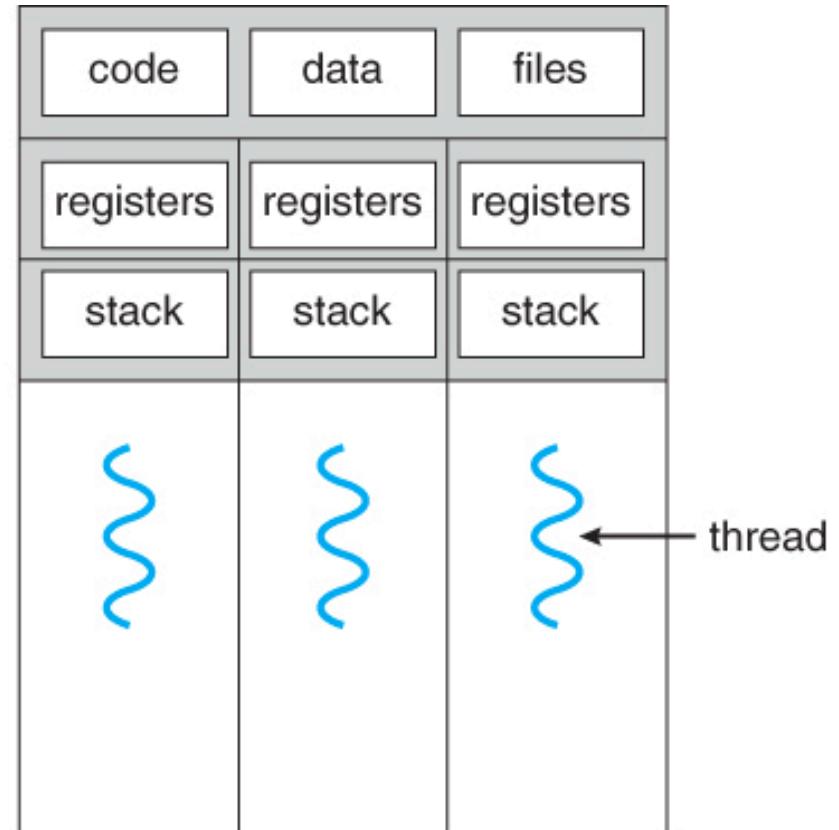
# New Parallelism Concept: Threads

- ❖ Common in parallel data processing: “**threads**”
  - ❖ Generalization of **process** abstraction of OS
- ❖ A program/process can *spawn* many threads
  - ❖ Each runs its part of program’s computations simultaneously
  - ❖ All threads share address space (so, data too)
- ❖ In multi-core CPUs, a thread uses up 1 core
  - ❖ “**Hyper-threading**”: Virtualizes a core to run 2 threads!

# Multiple Threads in a Process



single-threaded process



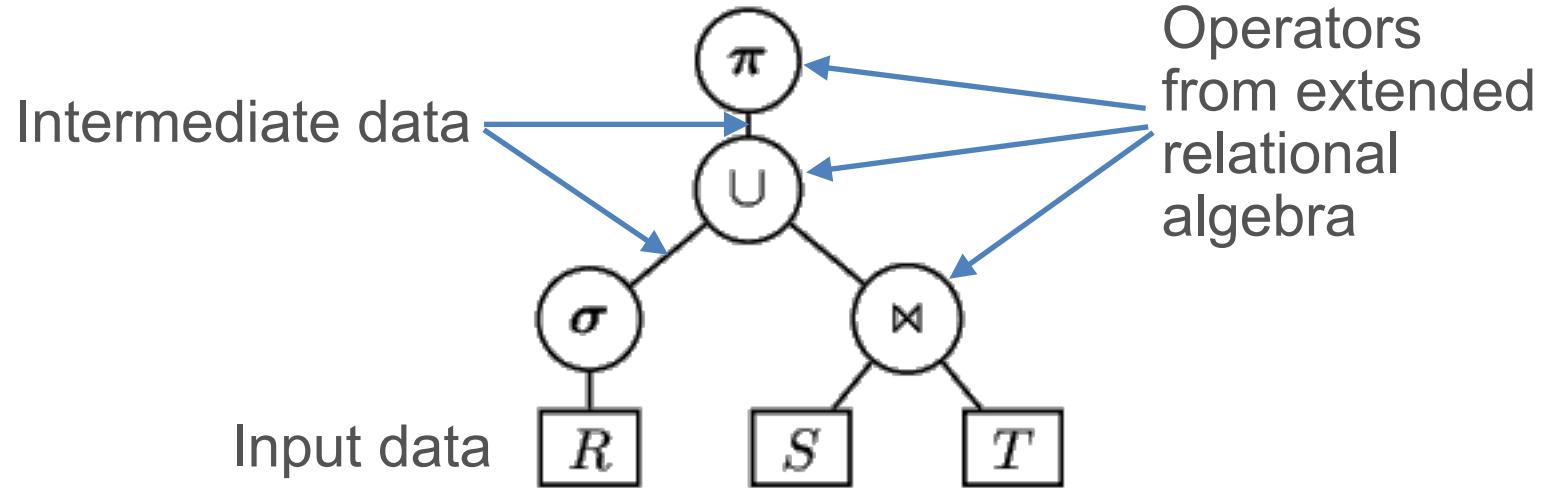
multithreaded process

# New Parallelism Concept: Dataflow

- ❖ Common in parallel data processing: “**Dataflow Graph**”:
  - ❖ A *directed graph* representation of a program with vertices being *abstract operations* from a restricted set of computational primitives:
  - ❖ Extended relational dataflows: RDBMS, Pandas, Modin
  - ❖ Matrix/tensor dataflows: NumPy, PyTorch, TensorFlow
- ❖ Enables us to reason about data-intensive programs at a higher level (logical level?)
- ❖ **Task Graph**: Similar but coarse-grained; vertex is a process

# Example Relational Dataflow Graph

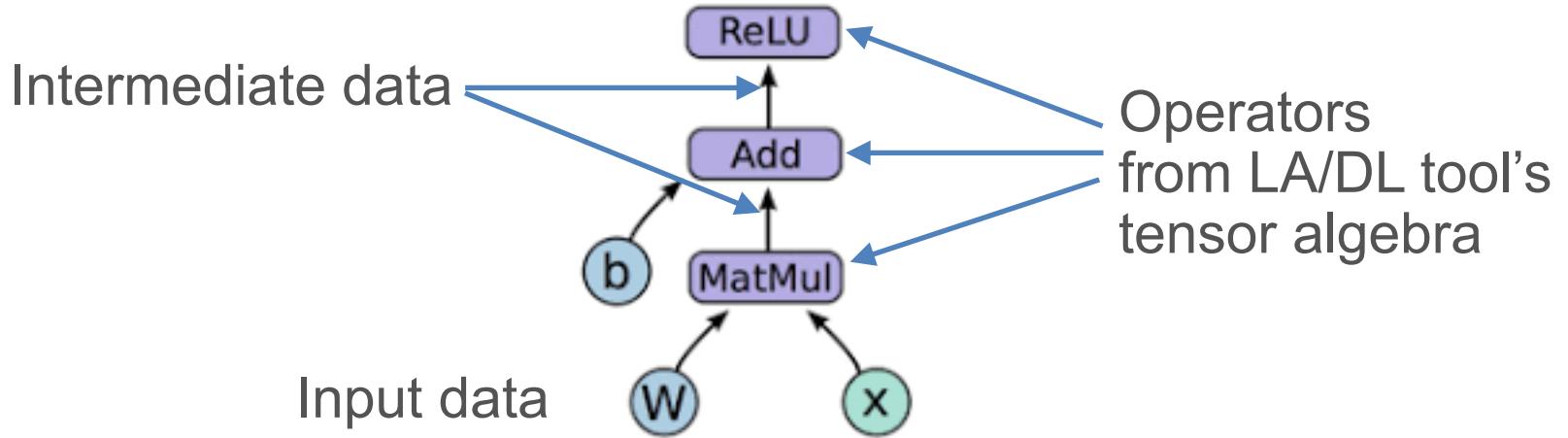
$$\pi(\sigma(R) \cup S \bowtie T)$$



Aka **Logical Query Plan** in the DB systems world

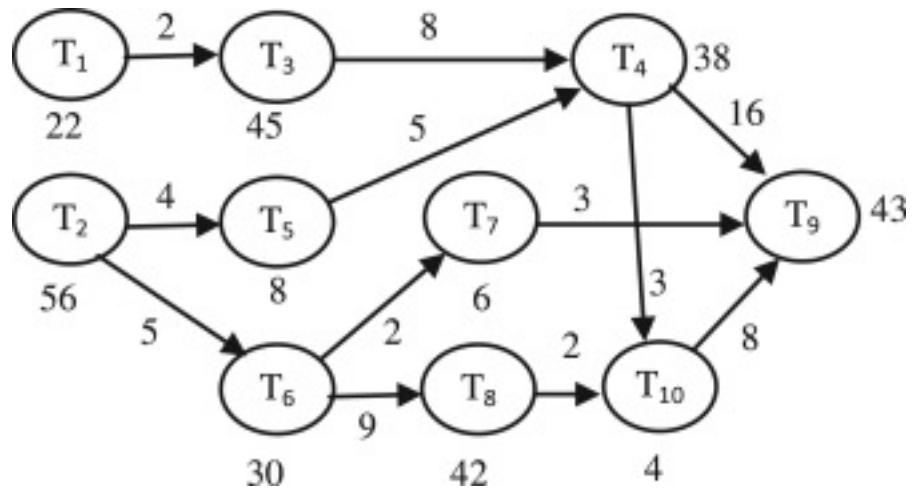
# Example Tensor Dataflow Graph

$$ReLU(WX + b)$$



Aka **Neural Computational Graph** in the ML systems world

# Example Task Graph



- ❖ More coarse-grained than operator-level dataflows
- ❖ Vertex: A full task/process
- ❖ Edge: A dependency between tasks
- ❖ Directed Acyclic Graph model (DAG) common; cycles?
- ❖ Data may not be shown

**NB:** Dask conflates the concepts of Dataflow and Task graphs because an “operation” on a Dask DataFrame becomes its own separate process/program under the hood!

<https://docs.dask.org/en/latest/graphviz.html>

# Parallel Data Processing

**Central Issue:** Workload takes too long for one processor!

**Basic Idea:** Split up workload across processors and perhaps also across machines/workers (aka “Divide and Conquer”)

**Key parallelism paradigms in data systems:**

Dataset is:	Shared	Replicated	Partitioned
Within a node:	“SIMD” “Pipelining”	“Task Parallel” Systems	“Data Parallel” Systems
Across nodes:	N/A	 DASK	 APACHE Spark™

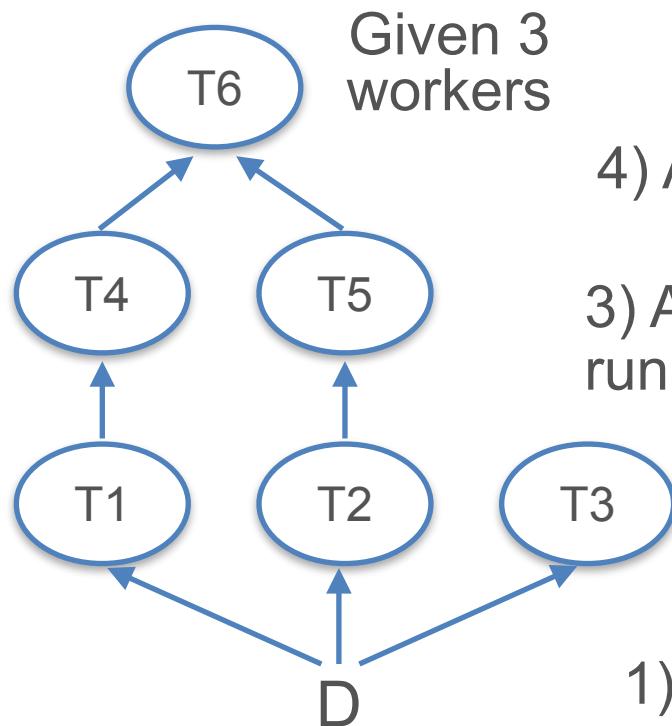
# Outline

- ❖ Basics of Parallelism
- ➔❖ Task Parallelism; Dask
  - ❖ Single-Node Multi-Core; SIMD; Accelerators
- ❖ Basics of Scalable Data Access
  - ❖ Paged Access; I/O Costs; Layouts/Access Patterns
  - ❖ Scaling Data Science Operations
- ❖ Data Parallelism: Parallelism + Scalability
  - ❖ Data-Parallel Data Science Operations
  - ❖ Optimizations and Hybrid Parallelism

# Task Parallelism

**Basic Idea:** Split up *tasks* across workers; if there is a common dataset that they read, just make copies of it (aka *replication*)

**Example:**



*This is your PA1 setup! Except, Task Scheduler puts tasks on workers for you.*

- 1) Copy whole D to all workers
- 2) Put T1 on worker 1 (W1), T2 on W2, T3 on W3; run all 3 in parallel
- 3) After T1 ends, run T4 on W1; after T2 ends, run T5 on W2; after T3 ends, W3 is *idle*
- 4) After T4 & T5 end, run T6 on W1; W2 is *idle*

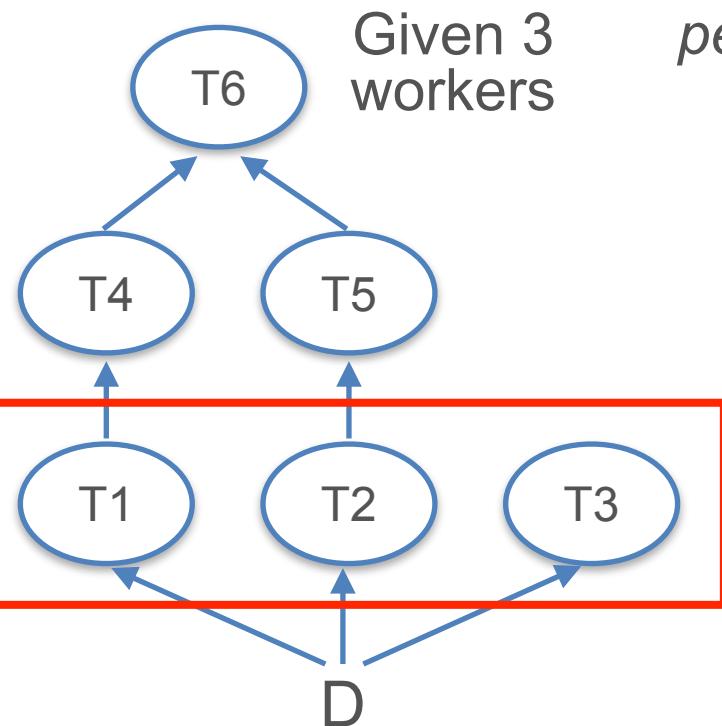
# Task Parallelism

- ❖ **Topological sort** of tasks in task graph for scheduling
- ❖ Notion of a “worker” can be at processor/core level, not just at node/server level
  - ❖ *Thread-level* parallelism possible instead of process-level
  - ❖ E.g., Dask: 4 worker nodes x 4 cores = 16 workers total
- ❖ **Main pros** of task parallelism:
  - ❖ **Simple** to understand; easy to implement
  - ❖ **Independence** of workers => low software complexity
- ❖ **Main cons** of task parallelism:
  - ❖ Data replication across nodes; **wastes memory/storage**
  - ❖ **Idle times** possible on workers

# Degree of Parallelism

- The largest amount of *concurrency* possible in the task graph, i.e., how many tasks can be run simultaneously

## Example:



*Q: How do we quantify the runtime performance benefits of task parallelism?*

But over time, degree of parallelism keeps dropping in this example

Degree of parallelism is only 3

So, more than 3 workers is not useful for this workload!

# Quantifying Benefit of Parallelism: Speedup

$$\text{Speedup} = \frac{\text{Completion time given only 1 worker}}{\text{Completion time given } n (>1) \text{ workers}}$$

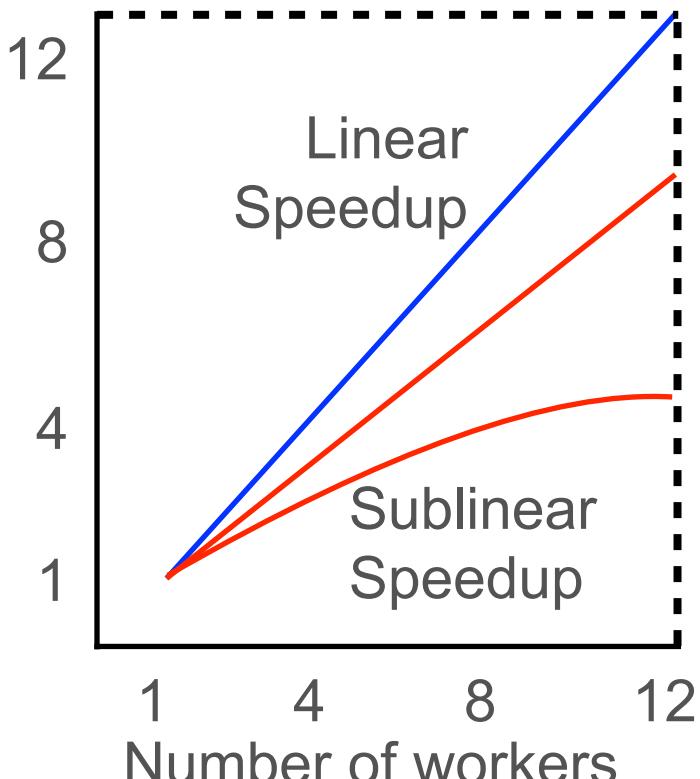
*Q: But given  $n$  workers, can we get a speedup of  $n$ ?*

It depends!

(On degree of parallelism, task dependency graph structure,  
intermediate data sizes, etc.)

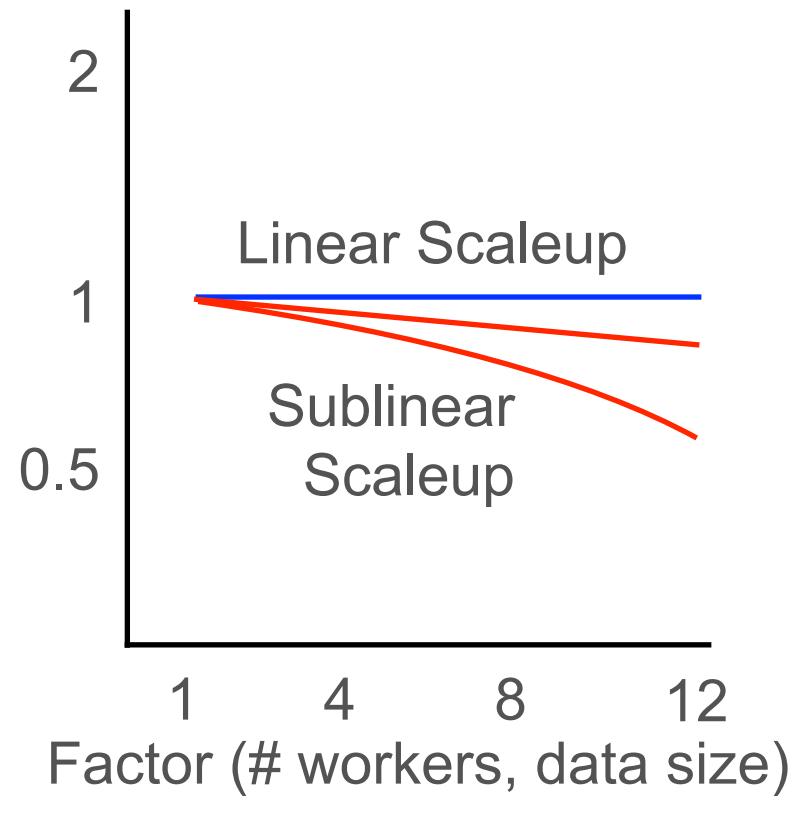
# Quantifying Benefit of Parallelism

Runtime speedup (fixed data size)



**Speedup** plot / Strong scaling

Runtime speedup



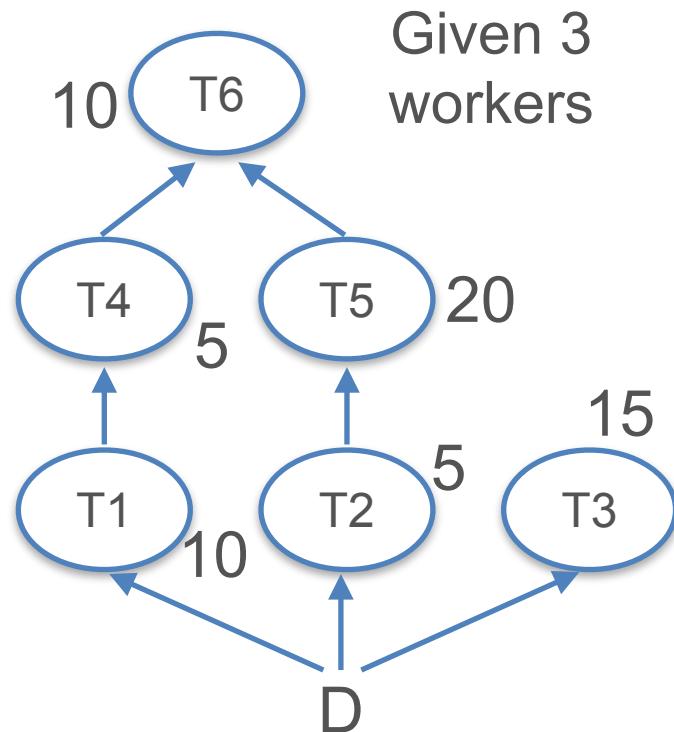
**Scaleup** plot / Weak scaling

**Q:** Is superlinear speedup/scaleup ever possible?

# Idle Times in Task Parallelism

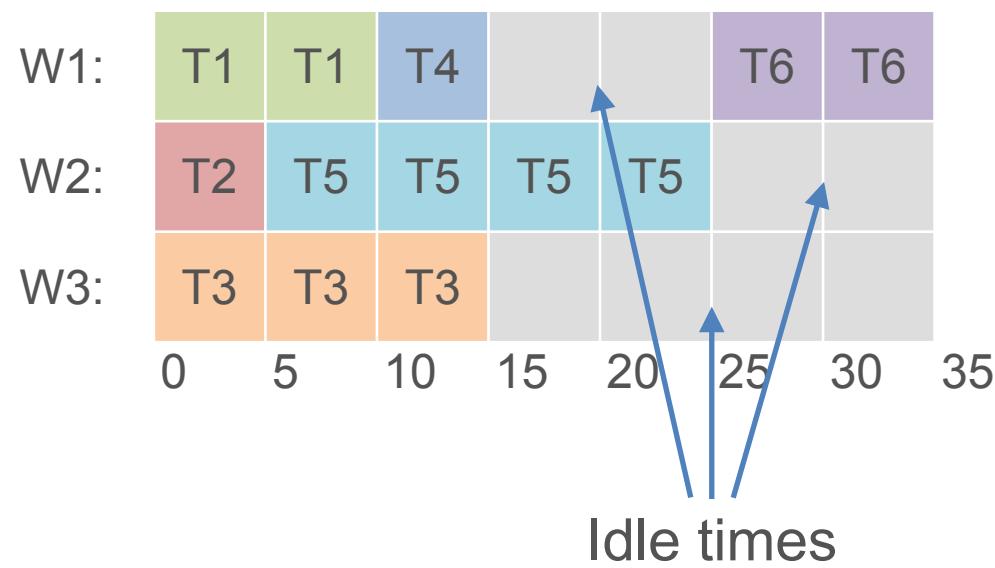
- Due to varying task completion times and varying degrees of parallelism in workload, idle workers waste resources

**Example:**



Given 3  
workers

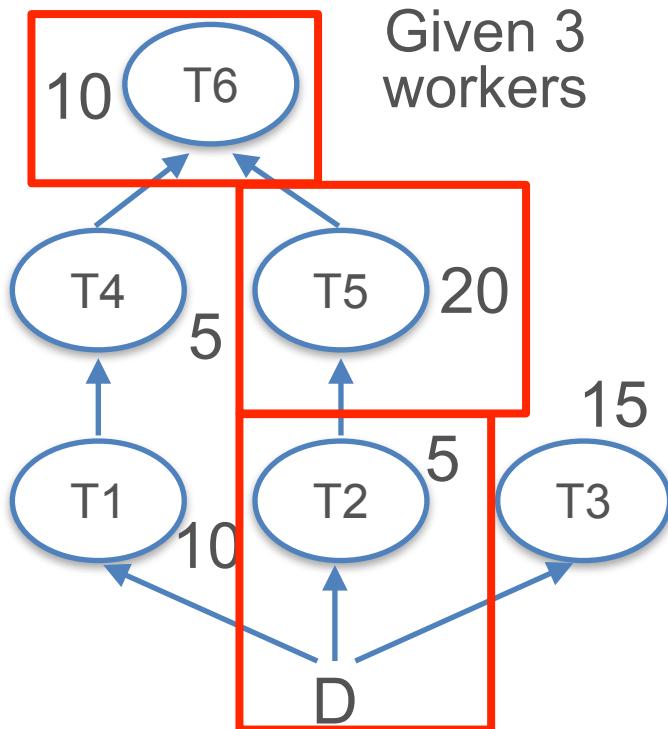
Gantt Chart visualization of schedule:



# Idle Times in Task Parallelism

- ❖ Due to varying task completion times and varying degrees of parallelism in workload, idle workers waste resources

## Example:

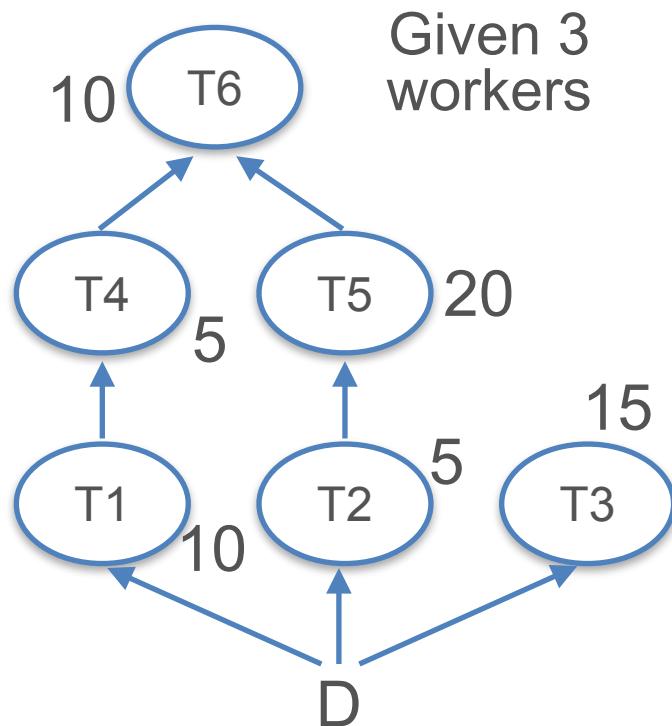


- ❖ In general, overall workload's completion time on task-parallel setup is always *lower bounded* by the **longest path** in the task graph
- ❖ Possibility: A task-parallel scheduler can “release” a worker if it knows that will be idle till the end
  - ❖ Can saves costs in cloud

# Calculating Task Parallelism Speedup

- Due to varying task completion times and varying degrees of parallelism in workload, idle workers waste resources

## Example:



Completion time  
with 1 worker       $10+5+15+5+20+10 = 65$

Parallel  
completion time      35

Speedup =  $65/35 = 1.9x$

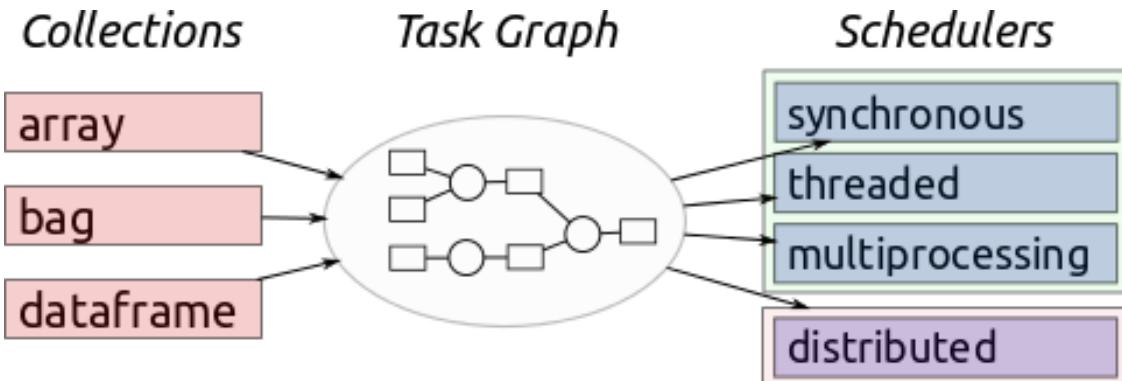
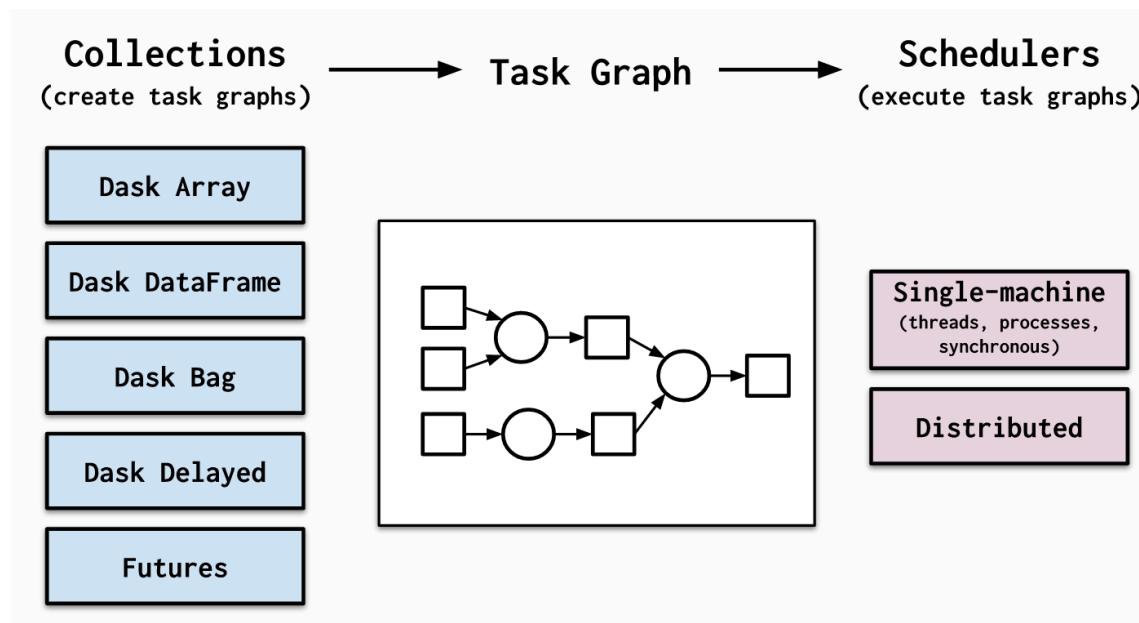
Ideal/linear speedup is 3x

*Q: Why is it only 1.9x?*

# Task Parallelism in Dask

- ❖ “*Dask is a flexible library for parallel computing in Python*”
- ❖ **2 key components:**
  - ❖ APIs for data sci. ops on large data
  - ❖ Dynamic task scheduling on multi-core/multi-node
- ❖ **Design desiderata:**
  - ❖ *Pythonic*: Stay within PyData stack (e.g., no JVM)
  - ❖ *Familiarity*: Retain APIs of NumPy, Pandas, etc.
  - ❖ *Scaling Up*: Seamlessly exploit all cores
  - ❖ *Scaling Out*: Easily exploit cluster (needs setup)
  - ❖ *Flexibility*: Can schedule custom tasks too
  - ❖ *Fast?*: “Optimized” implementations under APIs

# Task Parallelism in Dask



# Dask's Workflow

## ❖ “Lazy Evaluation”:

- ❖ Ops on data struct. are NOT executed immediately
- ❖ Triggered manually, e.g., *compute()*
- ❖ Dataflow graph / task graph is built under the hood

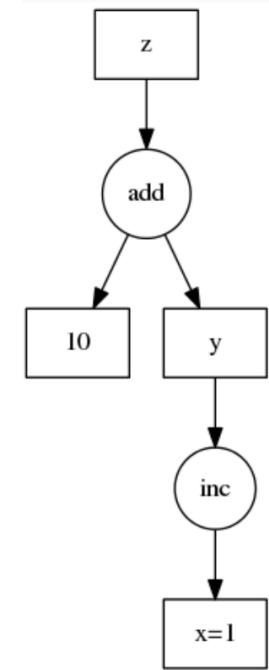
```
def inc(i):
    return i + 1

def add(a, b):
    return a + b

x = 1
y = inc(x)
z = add(y, 10)
```



```
d = {'x': 1,
      'y': (inc, 'x'),
      'z': (add, 'y', 10)}
```

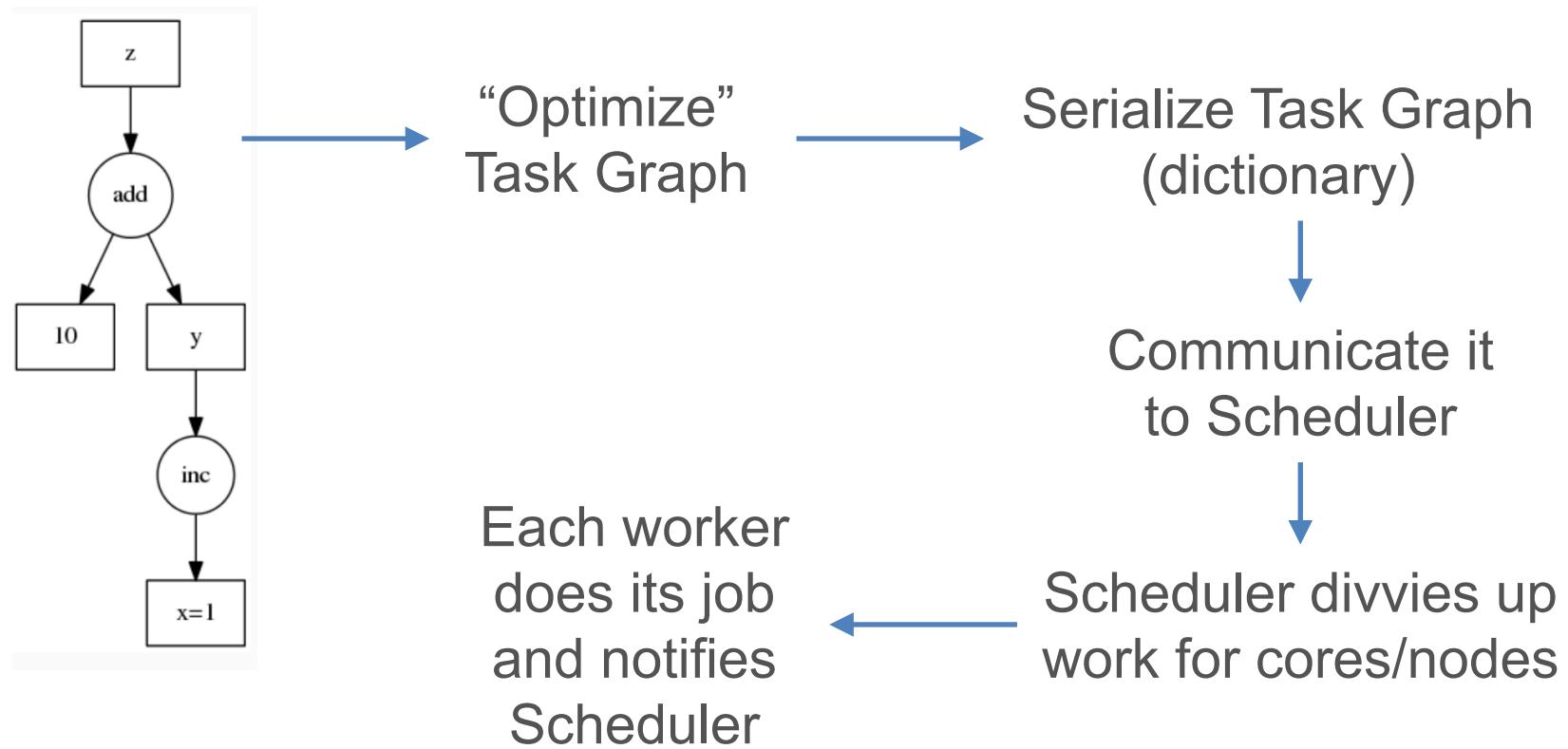


User code  
using their API

Internal dictionary with key-value pair representation of dataflow/task graph

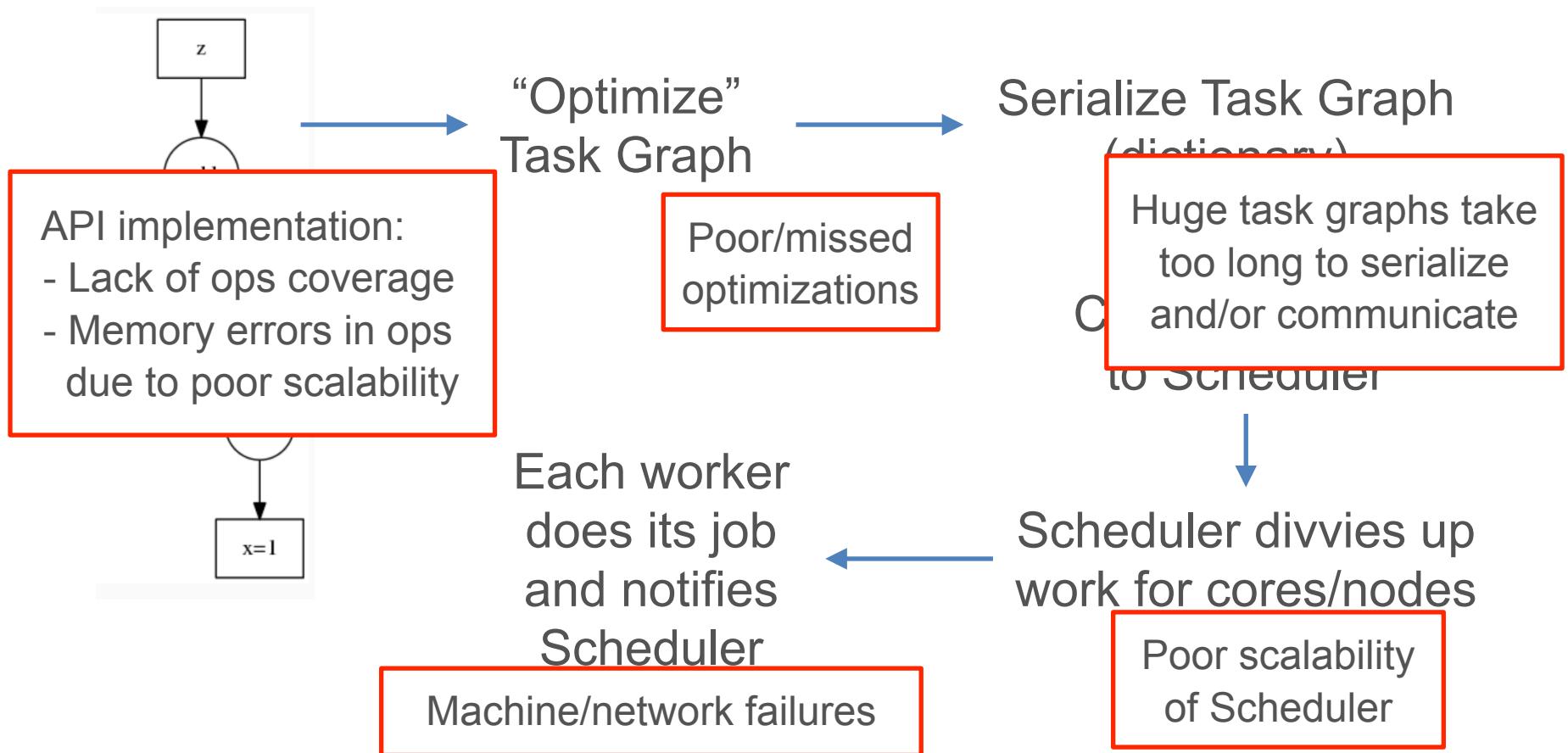
# Dask's Workflow

- ❖ Rest of the Dask's workflow for distributing computations:



# Possible Bottlenecks/Issues in Dask

- ❖ Rest of the Dask's workflow for distributing computations:



# Best Practices for Task-Par. Dask

- ❖ Is Dask even needed? Will single-node in-memory tool suffice?
- ❖ **Data Partition sizes:**
  - ❖ Avoid too few chunks (low degree of par.)
  - ❖ Avoid too many chunks (task graph overhead)
  - ❖ Be mindful of available DRAM
  - ❖ Rough guidelines they give:
    - ❖ # data chunks  $\sim$  3x-10x # cores, but
    - ❖ # cores x chunk size must be < machine DRAM, but
    - ❖ chunk size shouldn't be too small ( $\sim$ 1 GB is OK)

**Q:** *Do you tune any of these when using an RDBMS? :)*

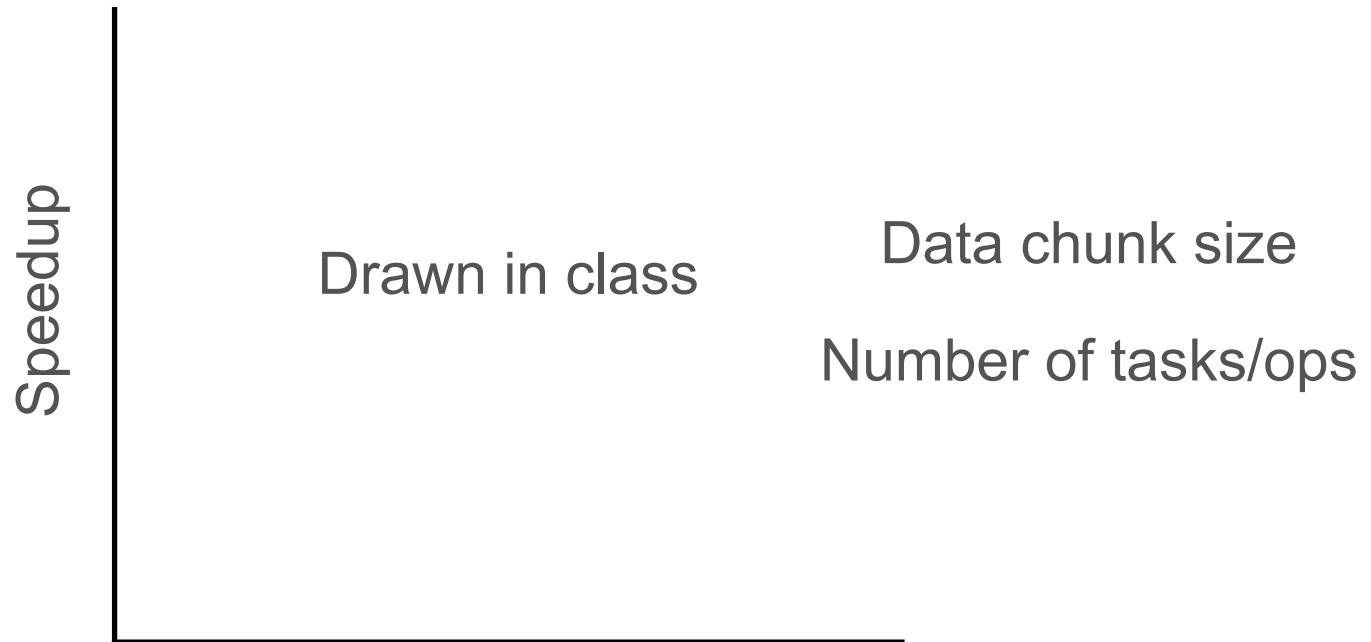
**Dask still lacks “physical data independence”!**

# Best Practices for Task-Par. Dask

- ❖ **Use the Diagnostics dashboard:**
  - ❖ Monitor # tasks, core/node usage, task completion
- ❖ **Task Graph sizes:**
  - ❖ Too large: Ser./comm./sched. bottlenecks
  - ❖ Too small: Under-utilization of cores/nodes
  - ❖ Rough guidelines they give:
    - ❖ Tune data chunk size to adjust # task (prev. point)
    - ❖ Break up a task/computation
    - ❖ Fuse tasks/computations aka “batching”

# Execution Optimization Tradeoffs

- ❖ Be judicious in tuning data chunk sizes
- ❖ Be judicious in batching vs breaking up tasks



# The WRATH of Codd?

## Information Retrieval

### A Relational Model of Data for Large Shared Data Banks

E. F. CODD

*IBM Research Laboratory, San Jose, California*

Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation). A prompting service which supplies such information is not a satisfactory solution. Activities of users at terminals and most application programs should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation are changed. Changes in data representation will often be needed as a result of changes in query, update, and report traffic and natural growth in the types of stored information.

 Arun Kumar  
@TweetAtAKK

...

<rant>

PSA for people building "scalable" ML/data sci. systems: TAKE A DB SYSTEMS IMPL. CLASS & get at least a PASS grade! 😞

It is 2021. It is ATROCIOUS how data scientists are still forced to tune low-level stuff like chunk sizes, loading, deg. of parallelism, etc. 😩

</rant>

## Information Retrieval

### A Relational Model of Data for Large Shared Data Banks

E. F. CODD

*IBM Research Laboratory, San Jose, California*

Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation). A prompting service which supplies such information is not a satisfactory solution. Activities of users at terminals and most application programs should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation are changed. Changes in data representation will often be needed as a result of changes in query, update, and report traffic and natural growth in the types of stored information.

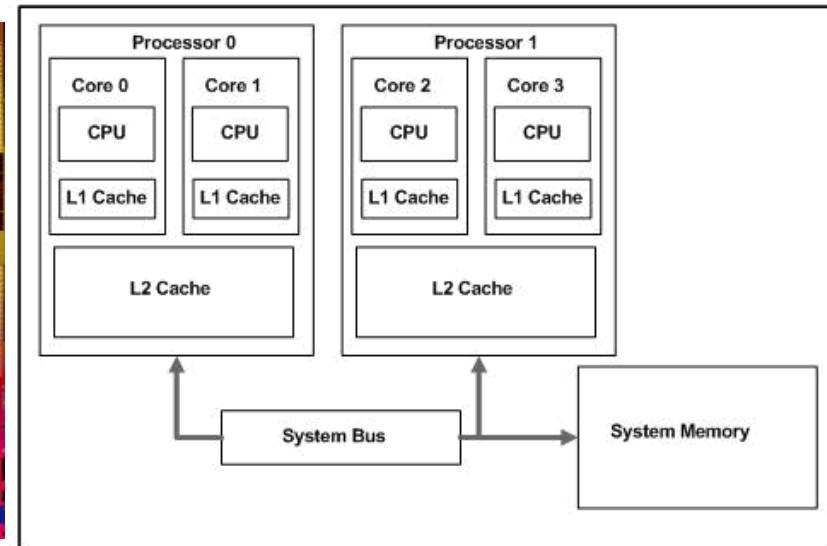
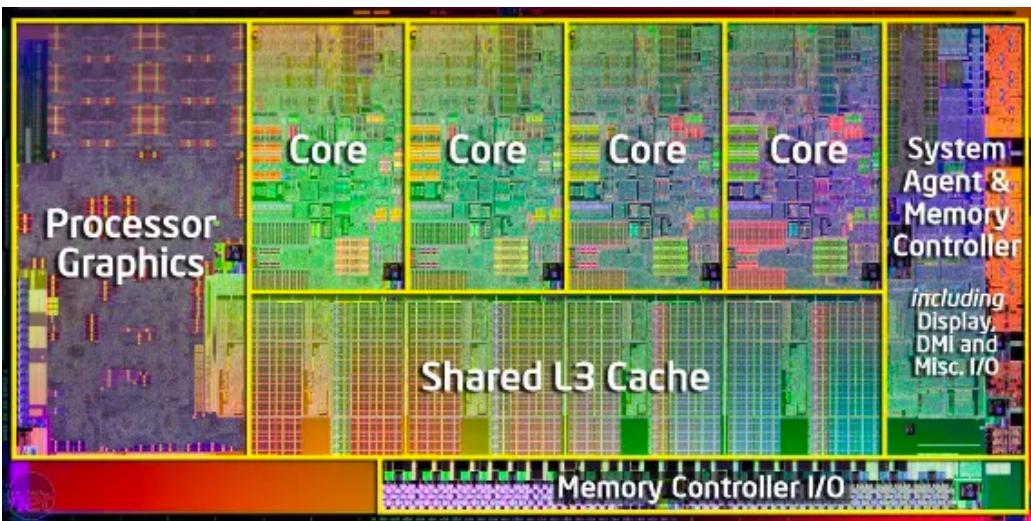
ALT

# Outline

- ❖ Basics of Parallelism
  - ❖ Task Parallelism; Dask
  - ➔❖ Single-Node Multi-Core; SIMD; Accelerators
- ❖ Basics of Scalable Data Access
  - ❖ Paged Access; I/O Costs; Layouts/Access Patterns
  - ❖ Scaling Data Science Operations
- ❖ Data Parallelism: Parallelism + Scalability
  - ❖ Data-Parallel Data Science Operations
  - ❖ Optimizations and Hybrid Parallelism

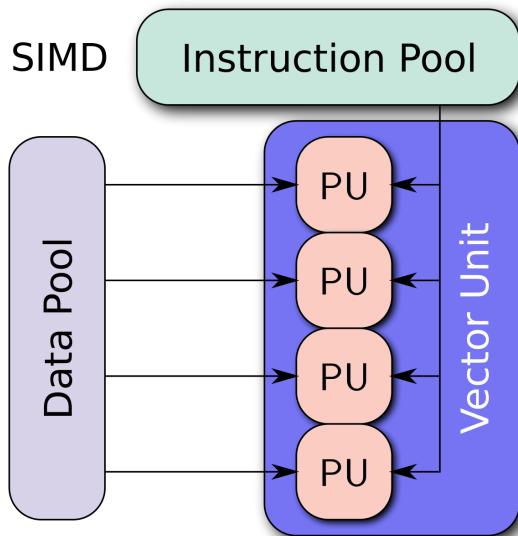
# Multi-core CPUs

- ❖ Modern machines often have multiple processors and multiple cores per processor; hierarchy of shared caches
- ❖ OS Scheduler now controls what cores/processors assigned to what processes/threads when

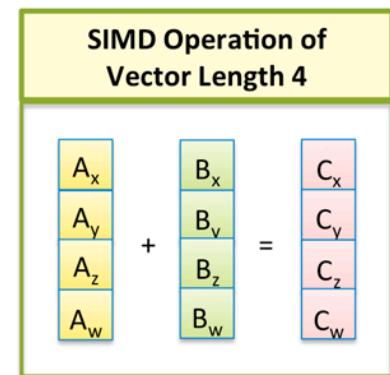
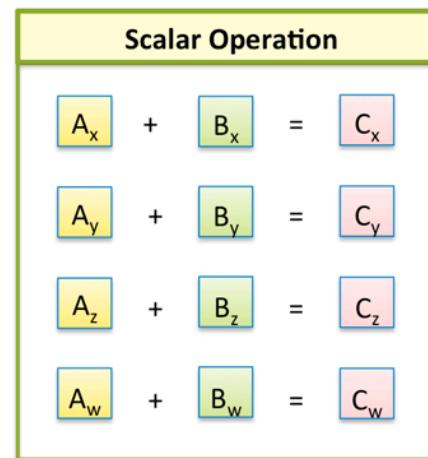


# Single-Instruction Multiple-Data

- ❖ **Single-Instruction Multiple-Data (SIMD):** A fundamental form of parallel processing in which *different chunks of data* are processed by the “*same*” set of *instructions* shared by multiple processing units (PUs)
- ❖ Aka “vectorized” instruction processing (vs “scalar”)
- ❖ Data science workloads are very amenable to SIMD



## Example for SIMD in data science:

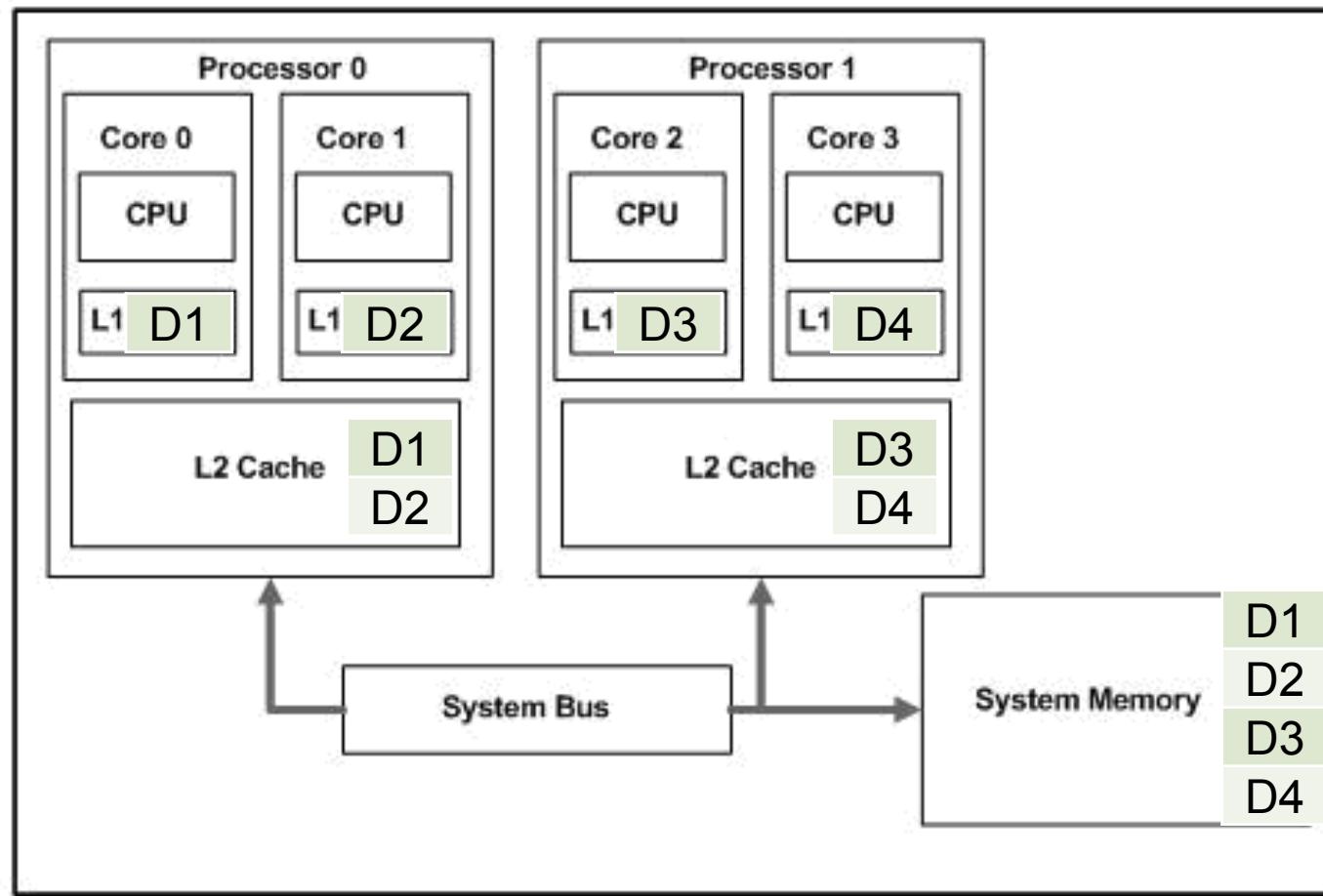


Intel® Architecture currently has SIMD operations of vector length 4, 8, 16

# SIMD Generalizations

- ❖ **Single-Instruction Multiple Thread (SIMT):** Generalizes notion of SIMD to *different threads* concurrently doing so
  - ❖ Each thread may be assigned a core or a whole PU
- ❖ **Single-Program Multiple Data (SPMD):** A higher level of abstraction generalizing SIMD operations or programs
  - ❖ Under the hood, may use multiple processes or threads
  - ❖ Each chunk of data processed by one core/PU
  - ❖ Applicable to any CPU, not just vectorized PUs
  - ❖ Most common form of parallel programming

# “Data Parallel” Multi-core Execution



# Quantifying Efficiency: Speedup

**Q:** How do we quantify the runtime performance benefits of multi-core parallelism?

- ❖ As with task parallelism, we measure the speedup:

$$\text{Speedup} = \frac{\text{Completion time given only 1 core}}{\text{Completion time given } n (>1) \text{ core}}$$

- ❖ In data science computations, an often useful surrogate for completion time is the instruction throughput **FLOP/s**, i.e., *number of floating point operations per second*
- ❖ Modern data processing programs, especially deep learning (DL) may have billions of FLOPs aka GFLOPs!

# Amdahl's Law

*Q: But given  $n$  cores, can we get a speedup of  $n$ ?*

It depends! (Just like it did with task parallelism)

- ❖ **Amdahl's Law:** Formula to upper bound possible speedup
  - ❖ A program has 2 parts: one that benefits from multi-core parallelism and one that does not
  - ❖ Non-parallel part could be for control, memory stalls, etc.

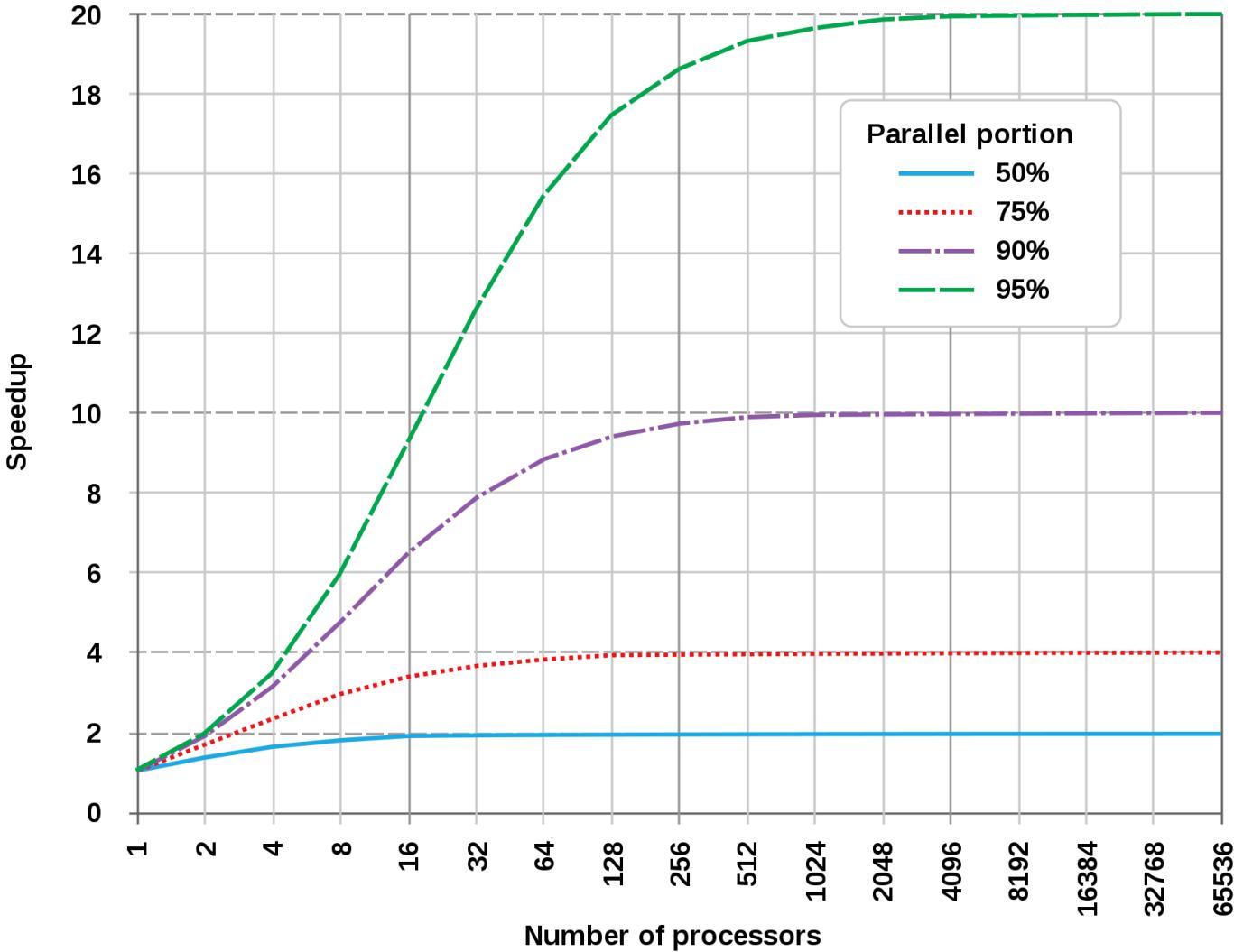
1 core:      n cores:

$$\begin{array}{ccc} T_{\text{yes}} & \xrightarrow{\hspace{1cm}} & T_{\text{yes}}/n \\ T_{\text{no}} & \xrightarrow{\hspace{1cm}} & T_{\text{no}} \end{array}$$

$$\text{Speedup} = \frac{T_{\text{yes}} + T_{\text{no}}}{T_{\text{yes}}/n + T_{\text{no}}} = \frac{n(1 + f)}{n + f}$$

Denote  $T_{\text{yes}}/T_{\text{no}} = f$

# Amdahl's Law



Speedup =

$$\frac{n(1 + f)}{n + f}$$

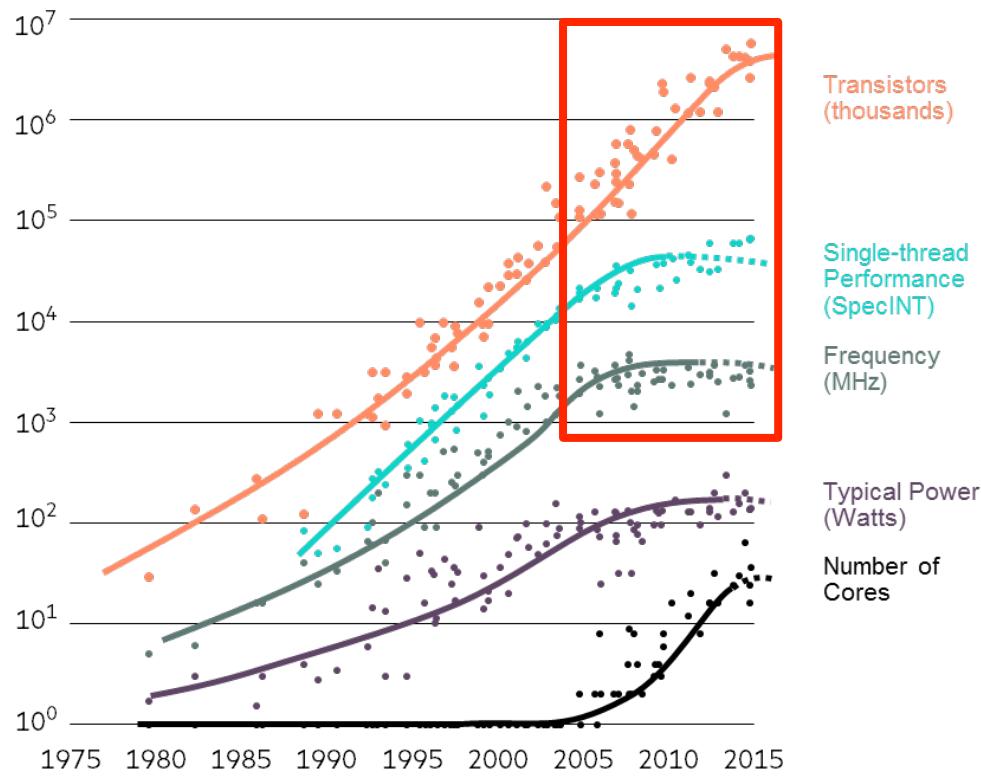
$$f = T_{yes}/T_{no}$$

Parallel portion =

$$f / (1 + f)$$

# Hardware Trends on Parallelism

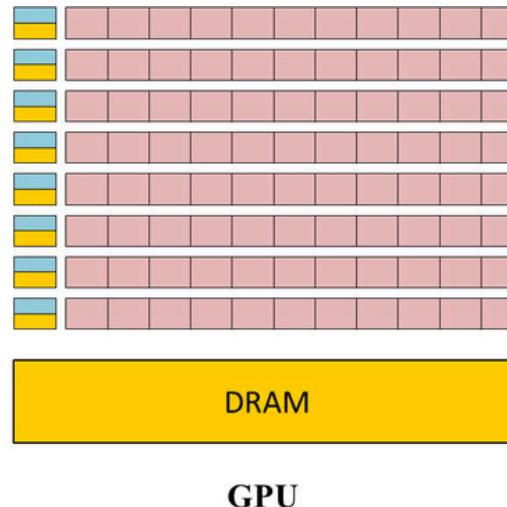
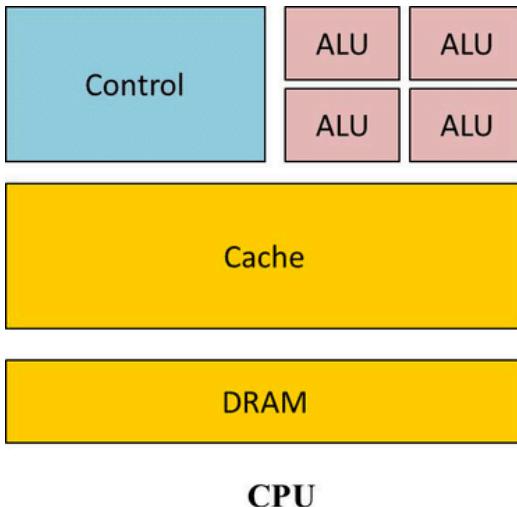
- ❖ Multi-core processors grew rapidly in early 2000s but hit physical limits due to packing efficiency and power issues
- ❖ End of “Moore’s Law” and End of “Dennard Scaling”



- ❖ Takeaway from hardware trends: it is hard for general-purpose CPUs to sustain FLOP-heavy programs like deep nets
- ❖ Motivated the rise of “accelerators” for some classes of programs

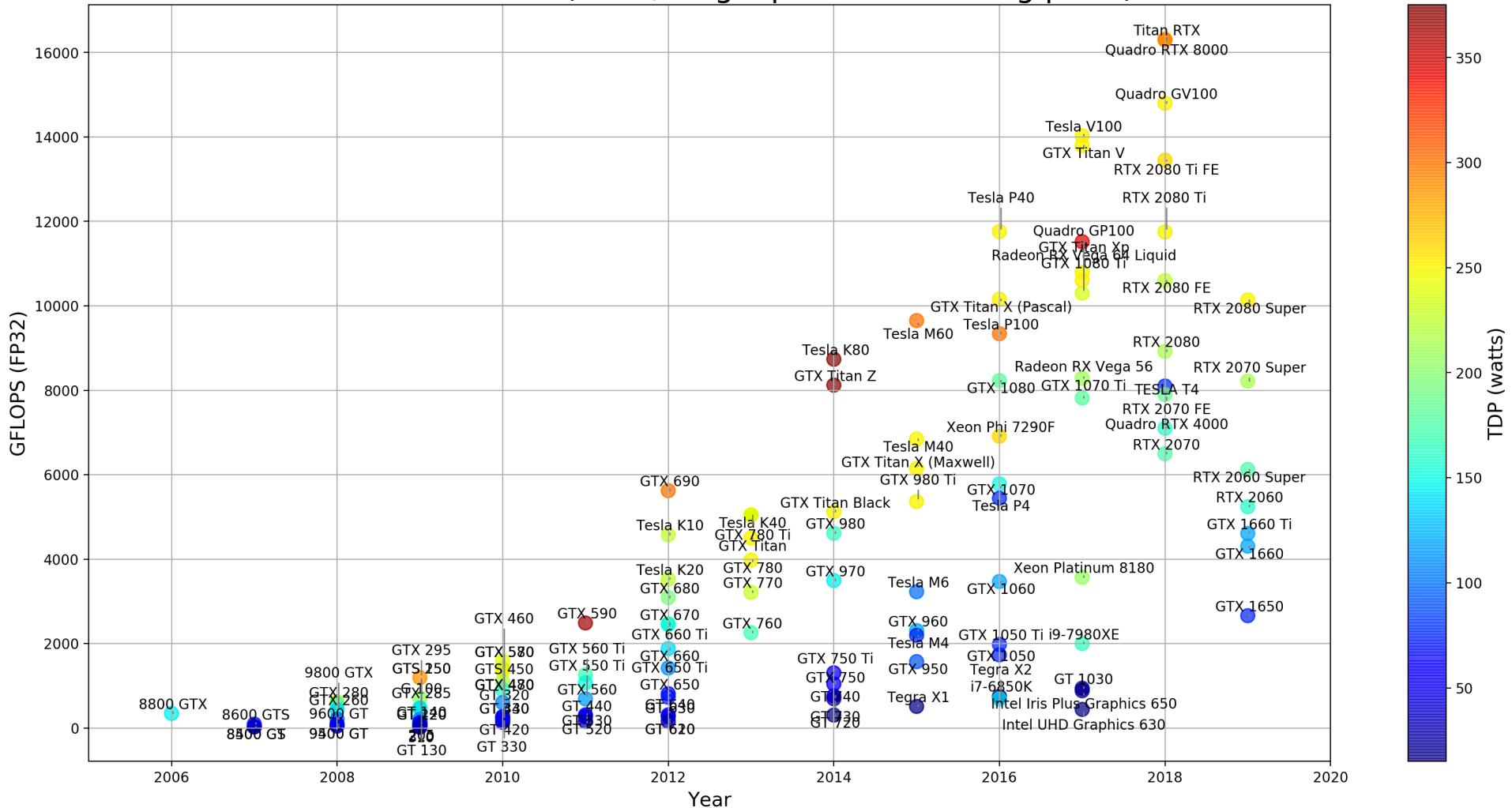
# Hardware Accelerators: GPUs

- ❖ **Graphics Processing Unit (GPU)**: Tailored for matrix/tensor ops
- ❖ Basic idea: use tons of ALUs; massive data parallelism (SIMD on steroids); Titan X offers ~11 TFLOP/s!
- ❖ Popularized by NVIDIA in early 2000s for video games, graphics, and video/multimedia; now ubiquitous in deep learning
- ❖ CUDA released in 2007; later wrapper APIs on top: CuDNN, CuSparse, CuDF (RapidsAI), etc.



# GPUs on the Market

GPU Performance (FP32, single precision floating point)



# Other Hardware Accelerators

- ❖ **Tensor Processing Unit (TPU)**: Even more specialized tensor ops in DL inference; ~45 TFLOP/s!
  - ❖ An “application-specific integrated circuit” (ASIC) created by Google in mid 2010s; used for AlphaGo!
- ❖ **Field-Programmable Gate Array (FPGA)**: Configurable for any class of programs; ~0.5-3 TFLOPs/s
  - ❖ Cheaper; new h/w-s/w stacks for ML/DL; Azure/AWS support



# Comparing Modern Parallel Hardware

	Multi-core CPU	GPU	FPGA	ASICs (e.g., TPUs)
Peak FLOPS/s	Moderate	High	High	Very High
Power Consumption	High	Very High	Very Low	Low-Very Low
Cost	Low	High	Very High	Highest
Generality / Flexibility	Highest	Medium	Very High	Lowest
“Fitness” for DL Training?	Poor Fit	Best Fit	Low Fit	Potential exists but yet unrealized
“Fitness” for DL Inference?	Moderate	Moderate	Good Fit	Best Fit
Cloud Vendor Support	All	All	AWS, Azure	GCP

# Review Questions

1. Briefly explain 3 benefits of large-scale data in Data Science.
2. What is a dataflow graph? Give an example from a data system.
3. How does a task graph differ from a dataflow graph?
4. Briefly explain 1 benefit and 1 drawback of task parallelism.
5. Briefly explain 1 scalability bottleneck that Dask still faces.
6. What is the lower bound on completion time when running a task graph in a task-parallel manner?
7. What is the degree of parallelism of a task graph?
8. What is speedup? How is it different from scaleup?
9. Is linear speedup always possible with task parallelism?
10. What is SIMD? Why is “vectorized” data processing critical?
11. What is the point of Amdahl’s Law?
12. Briefly 1 pro and 1 con of TPU vs GPU.

# Outline

- ❖ Basics of Parallelism
  - ❖ Task Parallelism; Dask
  - ❖ Single-Node Multi-Core; SIMD; Accelerators
- ❖ Basics of Scalable Data Access
  - ❖ Paged Access; I/O Costs; Layouts/Access Patterns
  - ❖ Scaling Data Science Operations
- ❖ Data Parallelism: Parallelism + Scalability
  - ❖ Data-Parallel Data Science Operations
  - ❖ Optimizations and Hybrid Parallelism

# DSC 102

# Systems for Scalable Analytics

Arun Kumar

Topic 3: Parallel and Scalable Data Processing

Part 2: Scalable Data Access

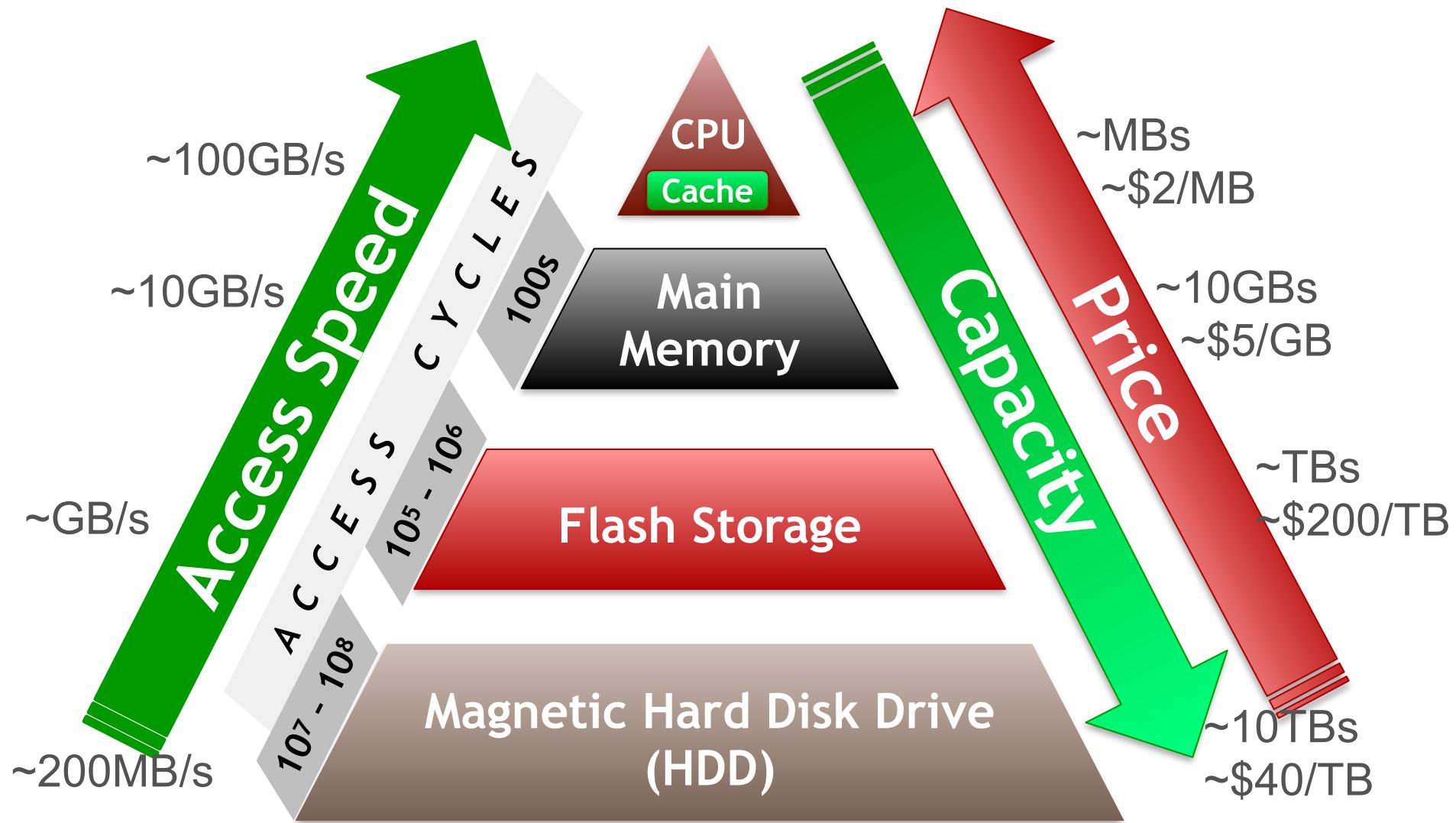
Ch. 9.4, 12.2, 14.1.1, 14.6, 22.1-22.3, 22.4.1, 22.8 of Cow Book

Ch. 5, 6.1, 6.3, 6.4 of MLSys Book

# Outline

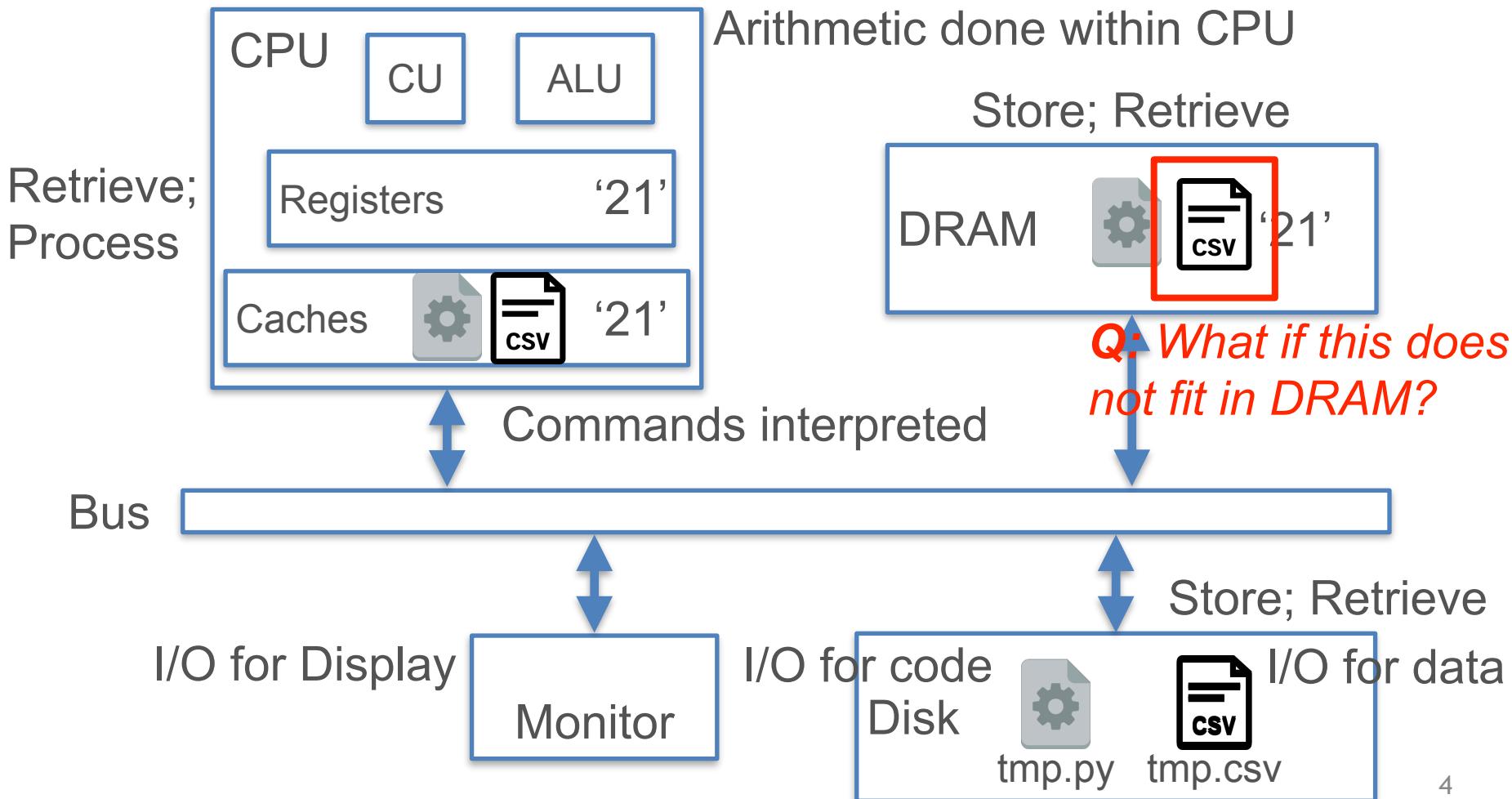
- ❖ Basics of Parallelism
  - ❖ Task Parallelism; Dask
  - ❖ Single-Node Multi-Core; SIMD; Accelerators
- ❖ Basics of Scalable Data Access
  - ❖ Paged Access; I/O Costs; Layouts/Access Patterns
  - ❖ Scaling Data Science Operations
- ❖ Data Parallelism: Parallelism + Scalability
  - ❖ Data-Parallel Data Science Operations
  - ❖ Optimizations and Hybrid Parallelism

# Recap: Memory Hierarchy



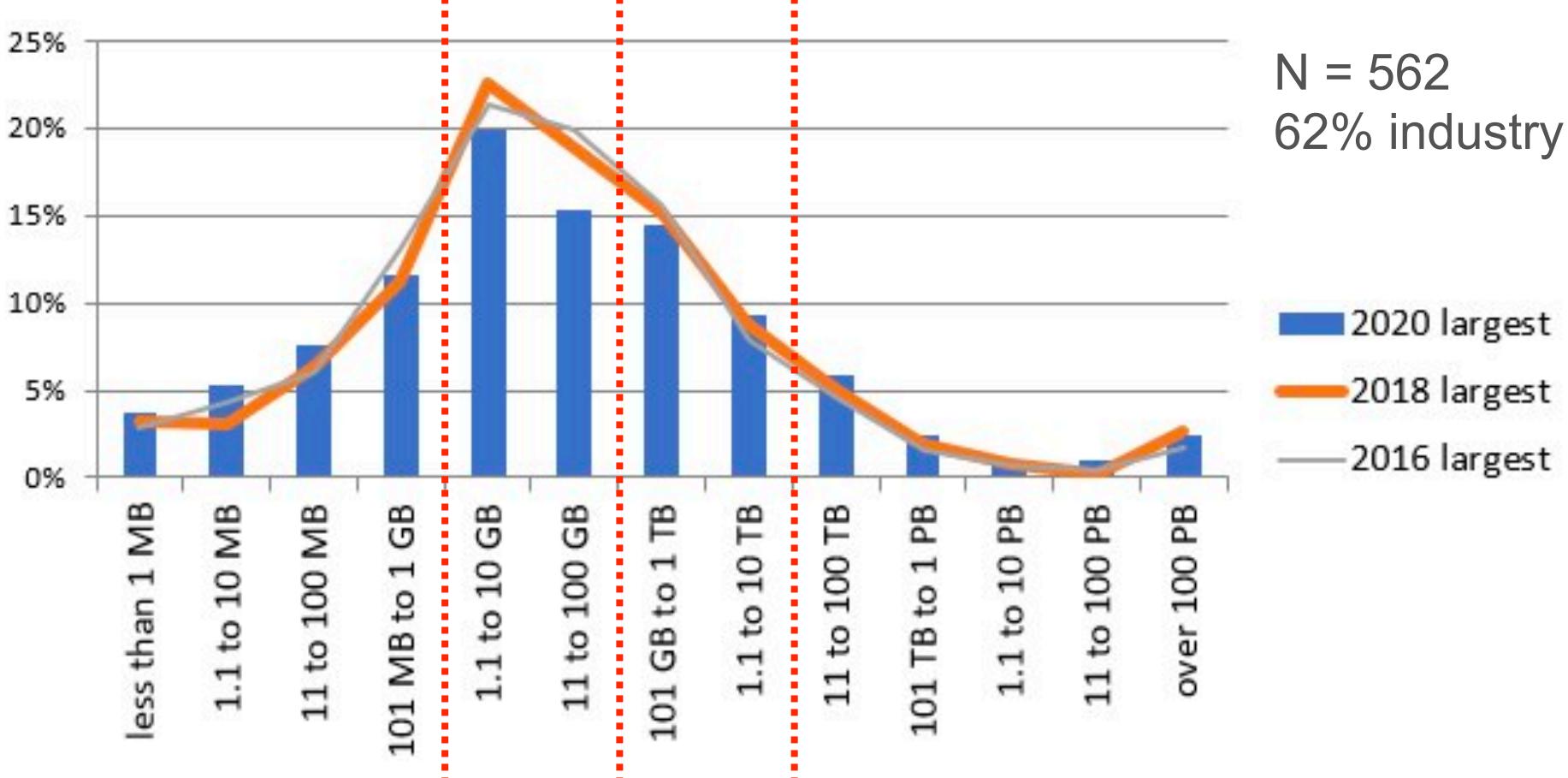
# Memory Hierarchy in Action

Rough sequence of events when program is executed



# Scale of Datasets in Practice

## KDnuggets 2020 Poll: Largest Dataset Analyzed



# Scalable Data Access

**Central Issue:** Large data file does not fit entirely in DRAM

**Basic Idea:** Divide-and-conquer again!

“Split” data file (virtually or physically) and stage reads of its pages from disk to DRAM; vice versa for writes

4 key regimes of scalability / staging reads:

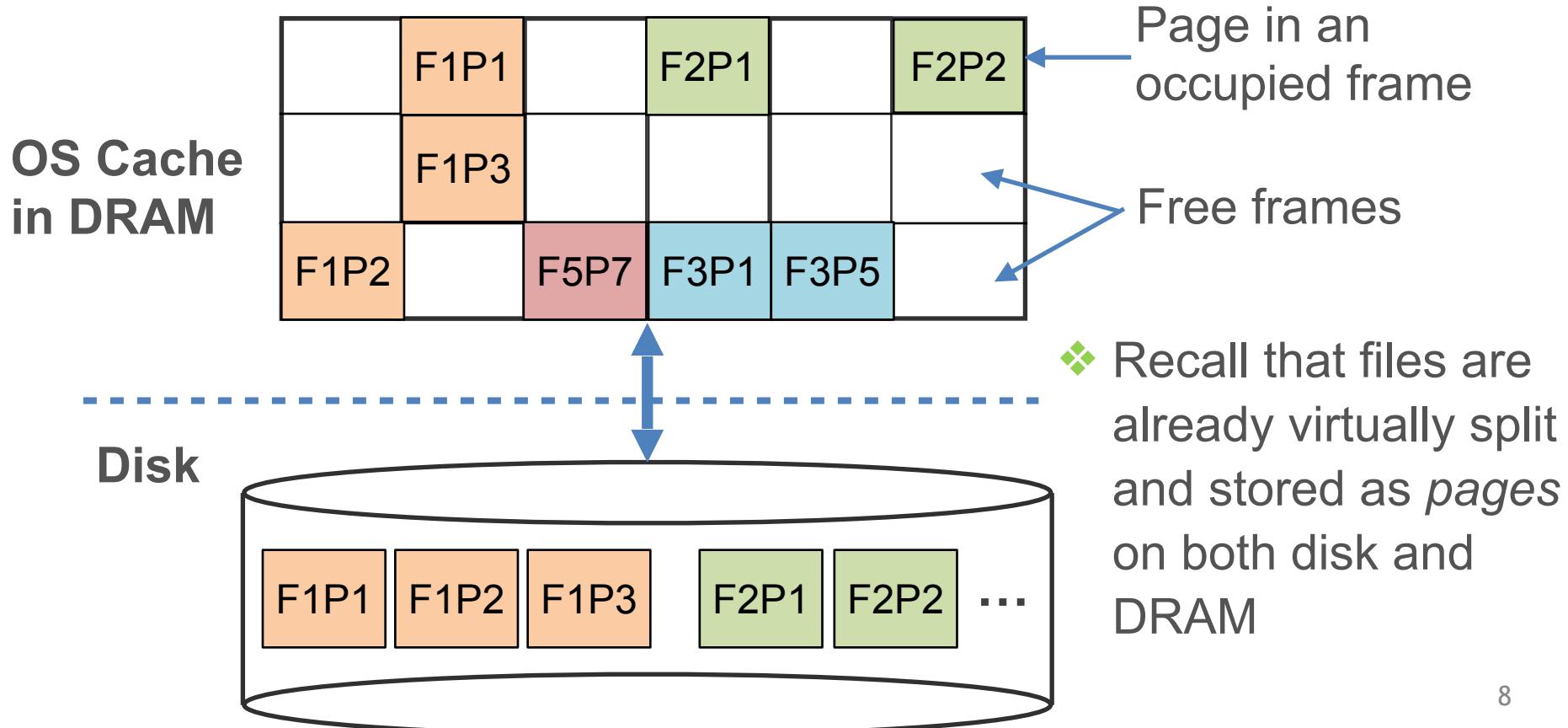
- ❖ **Single-node disk:** Paged access from file on local disk
- ❖ **Remote read:** Paged access from disk(s) over a network
- ❖ **Distributed memory:** Data fits on a cluster’s total DRAM
- ❖ **Distributed disk:** Use entire memory hierarchy of cluster

# Outline

- ❖ Basics of Parallelism
  - ❖ Task Parallelism; Dask
  - ❖ Single-Node Multi-Core; SIMD; Accelerators
- ❖ Basics of Scalable Data Access
  - ❖ Paged Access; I/O Costs; Layouts/Access Patterns
  - ❖ Scaling Data Science Operations
- ❖ Data Parallelism: Parallelism + Scalability
  - ❖ Data-Parallel Data Science Operations
  - ❖ Optimizations and Hybrid Parallelism

# Paged Data Access to DRAM

**Basic Idea:** “Split” data file (virtually or physically) and stage reads of its pages from disk to DRAM (vice versa for writes)



# Page Management in DRAM Cache

- ❖ **Caching:** Retaining pages read from disk in DRAM
- ❖ **Eviction:** Removing a page frame's content in DRAM
- ❖ **Spilling:** Writing out pages from DRAM to disk
  - ❖ If a page in DRAM is “**dirty**” (i.e., some bytes were written), eviction requires a spill; o/w, ignore that page
- ❖ The set of DRAM-resident pages typically changes over the lifetime of a process
- ❖ **Cache Replacement Policy:** The algorithm that chooses which page frame(s) to evict when a new page has to be cached but the OS cache in DRAM is full
  - ❖ Popular policies include Least Recently Used, Most Recently Used, etc. (more shortly)

# Quantifying I/O: Disk and Network

- ❖ Page reads/writes to/from DRAM from/to disk incur latency
- ❖ **Disk I/O Cost:** Abstract counting of number of page I/Os; can map to bytes given page size
- ❖ Sometimes, programs read/write data over network
- ❖ **Communication/Network I/O Cost:** Abstract counting of number of pages/bytes sent/received over network
- ❖ I/O cost is *abstract*; mapping to latency is *hardware-specific*

**Example:** Suppose a data file is 40GB; page size is 4KB

I/O cost to read file = 10 million page I/Os

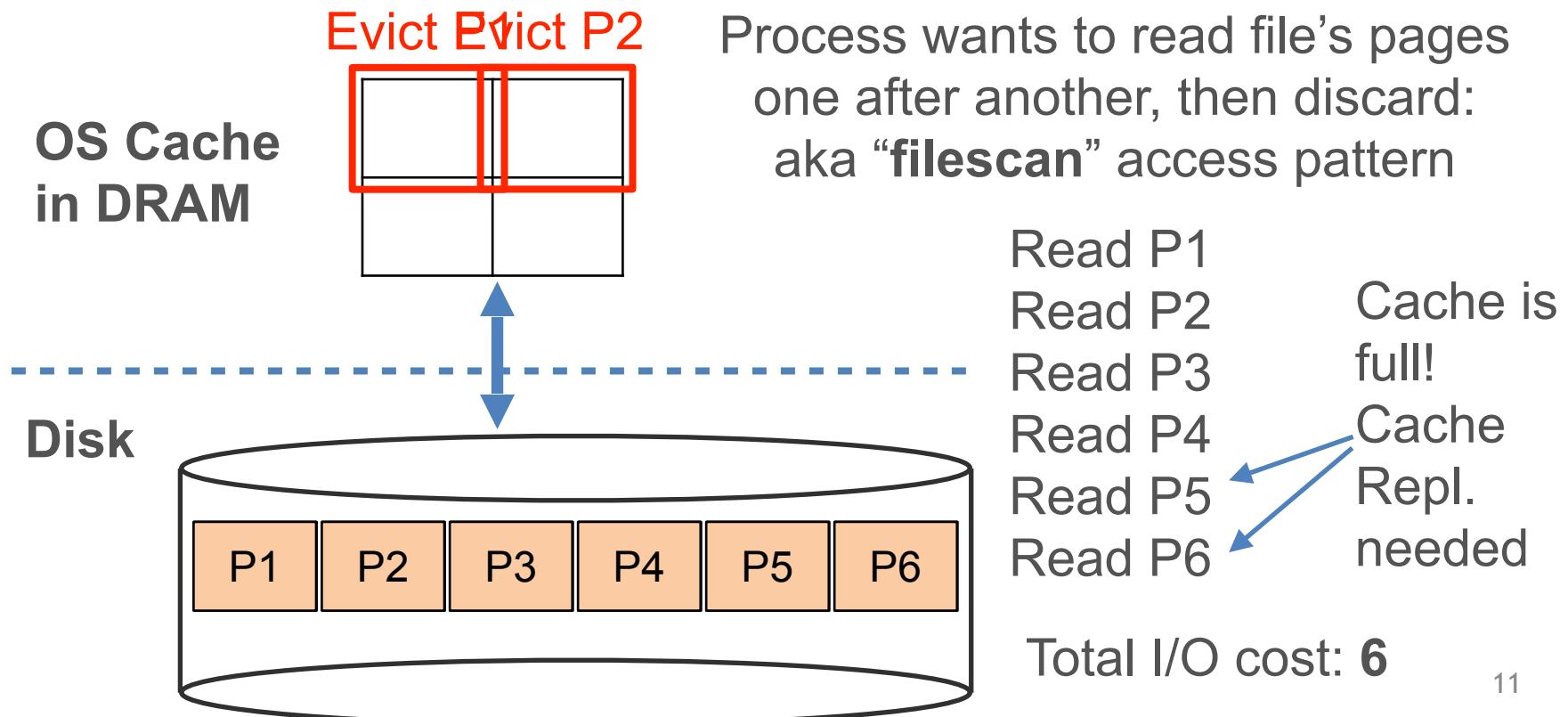
Disk with I/O throughput: 800 MB/s  $\longrightarrow$   $40\text{GB}/800\text{MBps} = 50\text{s}$

Network with speed: 200 MB/s  $\longrightarrow$   $40\text{GB}/200\text{MBps} = 200\text{s}$

# Scaling to (Local) Disk

**Basic Idea:** Split data file (virtually or physically) and stage reads of its pages from disk to DRAM (vice versa for writes)

Suppose OS Cache has only 4 frames; initially empty



# Scaling to (Local) Disk

- ❖ In general, scalable programs stage access to pages of file on disk and efficiently use available DRAM
  - ❖ Recall that typically DRAM size << Disk size
- ❖ Modern DRAM sizes can be 10s of GBs; so we read a “chunk”/“block” of file at a time (say, 1000s of pages)
  - ❖ On magnetic hard disks, such chunking leads to *more sequential I/Os*, raising throughput and lowering latency!
  - ❖ Similarly, write a chunk of dirtied pages at a time

# Generic Cache Replacement Policies

*Q: What to do if number of page frames is too few for file?*

- ❖ **Cache Replacement Policy:** Algorithm to decide which page frame(s) to evict to make space
  - ❖ Typical frame ranking criteria: recency of use, frequency of use, number of processes reading it
  - ❖ Typical optimization goal: Reduce total page I/O costs
- ❖ A few well-known policies:
  - ❖ **Least Recently Used (LRU):** Evict page that was used the longest time ago
  - ❖ **Most Recently Used (MRU):** Opposite of LRU
  - ❖ ML-based caching policies are “hot” nowadays! :)

**Ad:** Take CSE 132C for more on cache replacement policies

# Data Layouts and Access Patterns

- ❖ Recall that *data layouts* and *data access patterns* affect what data subset gets cached in higher level of memory hierarchy
  - ❖ Recall matrix multiplication example and CPU caches
- ❖ **Key Principle:** Optimizing layout of data file on disk based on data access pattern can help reduce I/O costs
  - ❖ Applies to both magnetic hard disk and flash SSDs
  - ❖ But especially critical for former due to vast differences in latency of random vs sequential access!

# Row-store vs Column-store Layouts

- ❖ A common dichotomy when serializing 2-D structured data (relations, matrices, DataFrames) to file on disk

A	B	C	D
1a	1b	1c	1d
2a	2b	2c	2d
3a	3b	3c	3d
4a	4b	4c	4d
5a	5b	5c	5d
6a	6b	6c	6d

Say, a page can fit only 4 cell values

Row-store:

1a,1b,1 c,1d	2a,2b,2 c,2d	3a,3b,3 c,3d
...	...	...

Col-store:

1a,2a,3 a,4a	5a,6a	1b,2b,3 b,4b
...	...	...

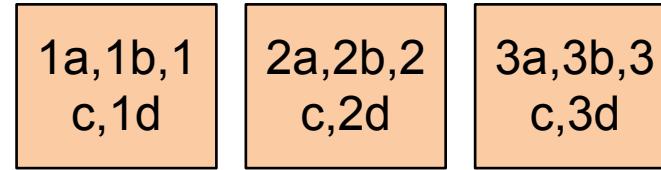
- ❖ Based on data access pattern of program, I/O costs with row- vs col-store can be orders of magnitude apart!

# Row-store vs Column-store Layouts

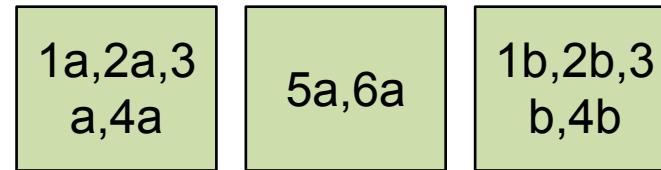
A	B	C	D
1a	1b	1c	1d
2a	2b	2c	2d
3a	3b	3c	3d
4a	4b	4c	4d
5a	5b	5c	5d
6a	6b	6c	6d

Say, a page can fit only 4 cell values

Row-store:



Col-store:



**Q:** *What is the I/O cost with each to compute, say, a sum over B?*

- ❖ With row-store: need to fetch *all* pages; I/O cost: 6 pages
- ❖ With col-store: need to fetch only B's pages; I/O cost: 2 pages
- ❖ This difference generalizes to higher dim. for tensors

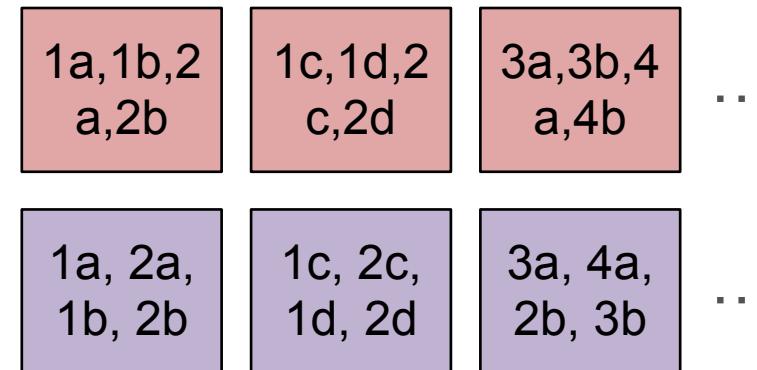
# Hybrid/Tiled/“Blocked” Layouts

- ❖ Sometimes, it is beneficial to do a hybrid, especially for analytical RDBMSs and matrix/tensor processing systems

A	B	C	D
1a	1b	1c	1d
2a	2b	2c	2d
3a	3b	3c	3d
4a	4b	4c	4d
5a	5b	5c	5d
6a	6b	6c	6d

Say, a page can fit only 4 cell values

Hybrid stores with 2x2 tiled layout:

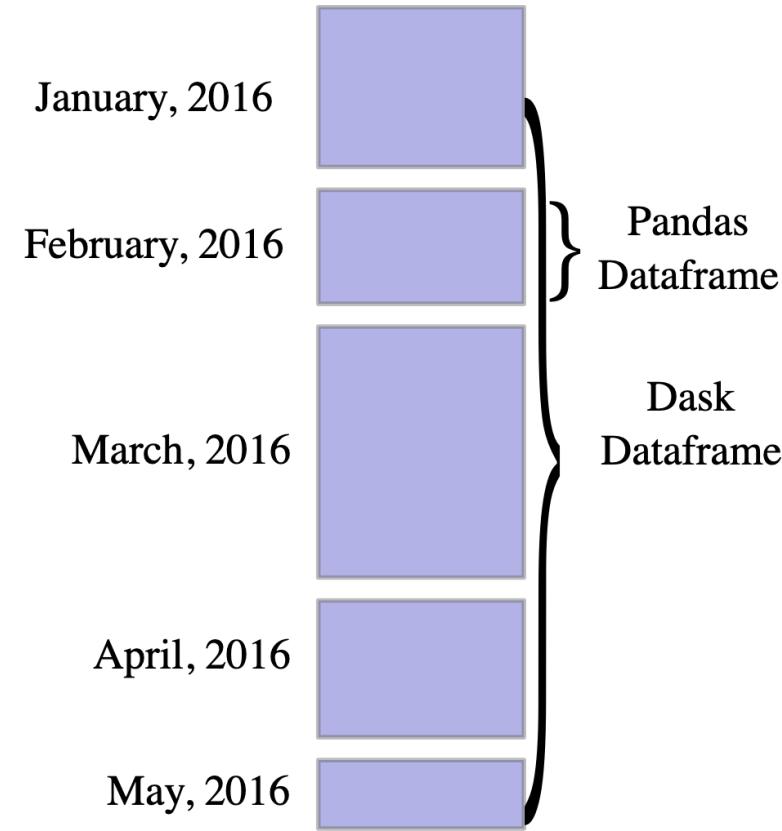


**Key Principle:** What data layout will yield lower I/O costs (row vs col vs tiled) depends on data access pattern of the program!

# Example: Dask's DataFrame

**Basic Idea:** Split data file (virtually or physically) and stage reads of its pages from disk to DRAM (vice versa for writes)

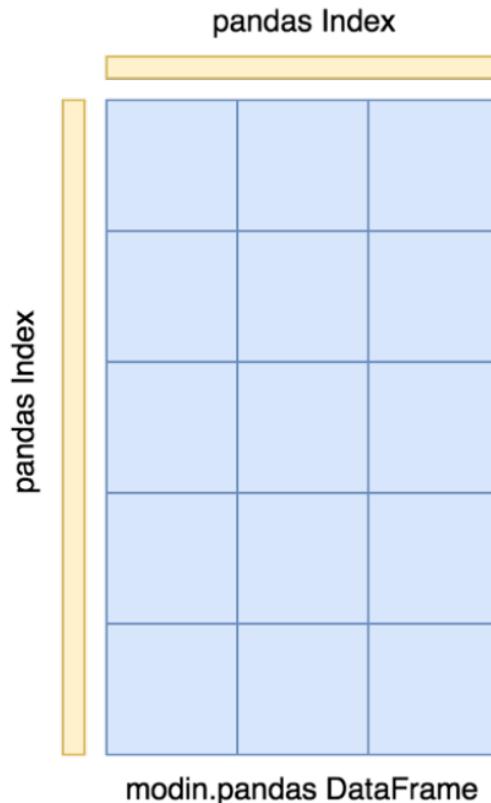
- ❖ Dask DF scales to disk-resident data via a row-store
- ❖ “Virtual” split: each split is a Pandas DF under the hood
- ❖ Dask API is a “wrapper” around Pandas API to scale ops to splits and put all results together
- ❖ If file is too large for DRAM, need manual *repartition()* to get physically smaller splits (< ~1GB)



# Example: Modin's DataFrame

**Basic Idea:** Split data file (virtually or physically) and stage reads of its pages from disk to DRAM (vice versa for writes)

- ❖ Modin's DF aims to scale to disk-resident data via a tiled store
- ❖ Enables seamless scaling along *both* dimensions
- ❖ Easier use of multi-core parallelism
- ❖ Many in-memory RDBMSs had this, e.g., SAP HANA, Oracle TimesTen
- ❖ ScaLAPACK had this for matrices



# Scaling with Remote Reads

**Basic Idea:** Split data file (virtually or physically) and stage reads of its pages from disk to DRAM (vice versa for writes)

- ❖ Similar to scaling to local disk but not “local”:
  - ❖ Stage page reads from remote disk/disks over the network (e.g., from S3)
- ❖ *More restrictive* than scaling with local disk, since spilling is not possible or requires costly network I/Os
  - ❖ OK for a *one-shot* filescan access pattern
  - ❖ Use DRAM to cache; repl. policies
  - ❖ Can also use smaller local disk as cache; you did this in PA1

# Outline

- ❖ Basics of Parallelism
  - ❖ Task Parallelism; Dask
  - ❖ Single-Node Multi-Core; SIMD; Accelerators
- ❖ Basics of Scalable Data Access
  - ❖ Paged Access; I/O Costs; Layouts/Access Patterns
- ➡ ❖ Scaling Data Science Operations
- ❖ Data Parallelism: Parallelism + Scalability
  - ❖ Data-Parallel Data Science Operations
  - ❖ Optimizations and Hybrid Parallelism

# Scaling Data Science Operations

- ❖ Scalable data access for key representative examples of programs/operations that are ubiquitous in data science:
  - ❖ DB systems:
    - ❖ Non-deduplicating project
    - ❖ Simple SQL aggregates
    - ❖ SQL GROUP BY aggregates
  - ❖ ML systems:
    - ❖ Matrix sum/norms
    - ❖ (Stochastic) Gradient Descent

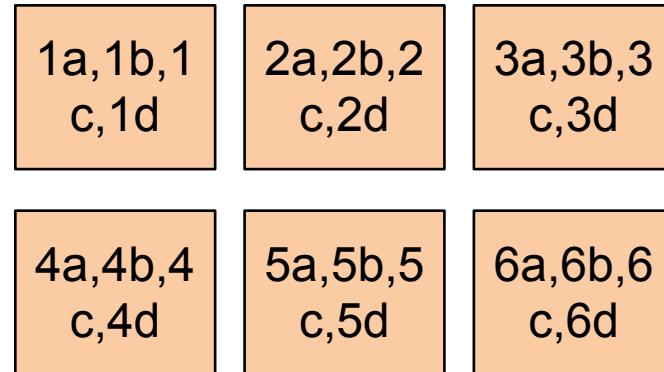
# Scaling to Disk: Non-dedup. Project

A	B	C	D
1a	1b	1c	1d
2a	2b	2c	2d
3a	3b	3c	3d
4a	4b	4c	4d
5a	5b	5c	5d
6a	6b	6c	6d

R

SELECT C FROM R

Row-store:



- ❖ Straightforward **filescan** data access pattern
  - ❖ Read one page at a time into DRAM; may need cache repl.
  - ❖ Drop unneeded columns from tuples on the fly
- ❖ I/O cost: **6** (read) + output # pages (write)

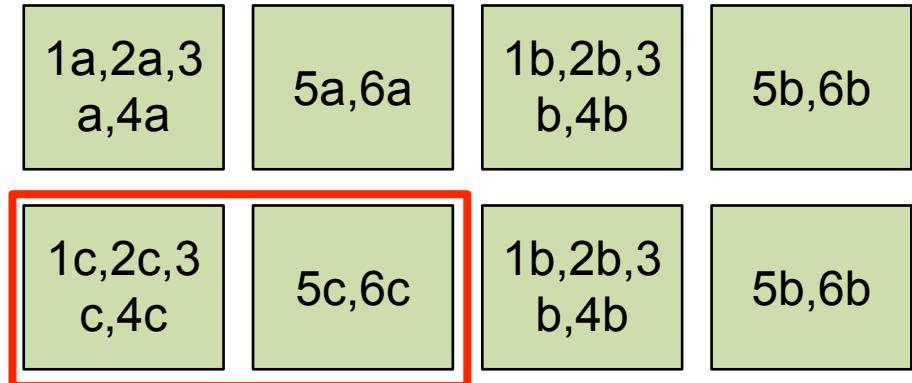
# Scaling to Disk: Non-dedup. Project

A	B	C	D
1a	1b	1c	1d
2a	2b	2c	2d
3a	3b	3c	3d
4a	4b	4c	4d
5a	5b	5c	5d
6a	6b	6c	6d

R

SELECT C FROM R

Col-store:



- ❖ Since we only need col C, no need to read other pages!
- ❖ I/O cost: 2 (read) + output # pages (write)
- ❖ Big advantage for col-stores over row-stores for SQL analytics queries (projects, aggregates, etc.), aka “OLAP”
- ❖ Rationale for col-store RDBMS (e.g., Vertica) and Parquet

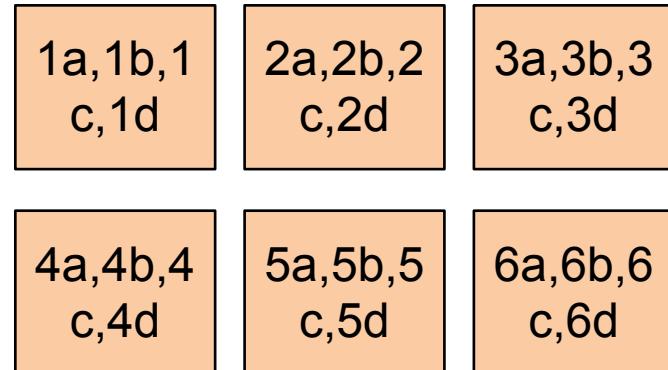
# Scaling to Disk: Simple Aggregates

A	B	C	D
1a	1b	1c	1d
2a	2b	2c	2d
3a	3b	3c	3d
4a	4b	4c	4d
5a	5b	5c	5d
6a	6b	6c	6d

R

SELECT MAX(A) FROM R

Row-store:



- ❖ Again, straightforward **filescan** data access pattern
  - ❖ Similar I/O behavior as non-deduplicating project
- ❖ I/O cost: **6** (read) + output # pages (write)

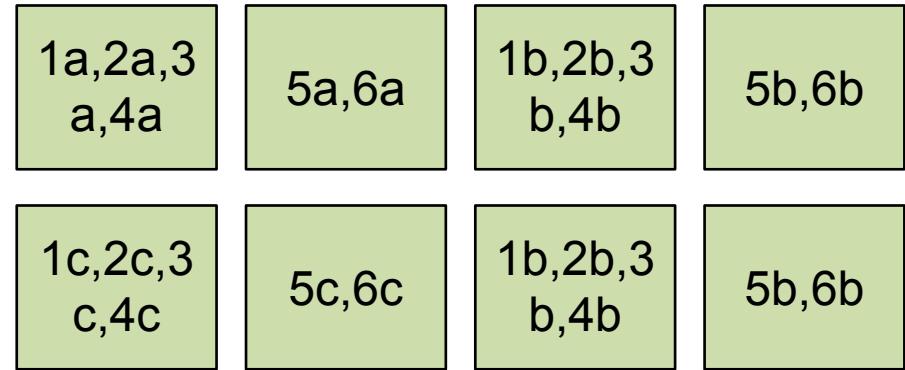
# Scaling to Disk: Simple Aggregates

A	B	C	D
1a	1b	1c	1d
2a	2b	2c	2d
3a	3b	3c	3d
4a	4b	4c	4d
5a	5b	5c	5d
6a	6b	6c	6d

R

SELECT MAX(A) FROM R

Col-store:



- ❖ Similar to the non-dedup. project, we only need col A; no need to read other pages!
- ❖ I/O cost: **2** (read) + output # pages (write)

# Scaling to Disk: Group By Aggregate

A	B	C	D
a1	1b	1c	4
a2	2b	2c	3
a1	3b	3c	5
a3	4b	4c	1
a2	5b	5c	10
a1	6b	6c	8

R

SELECT A, SUM(D)  
FROM R GROUP BY A

- ❖ Now it is not straightforward due to the GROUP BY!
- ❖ Need to “collect” all tuples in a group and apply agg. func. to each
- ❖ Typically done with a **hash table** maintained in DRAM
- ❖ Has 1 record per group and maintains “running information” for that group’s agg. func.
- ❖ Built on the fly during filescan of R; holds the output in the end

Hash table (output)

A	Running Info.
a1	17
a2	13
a3	1

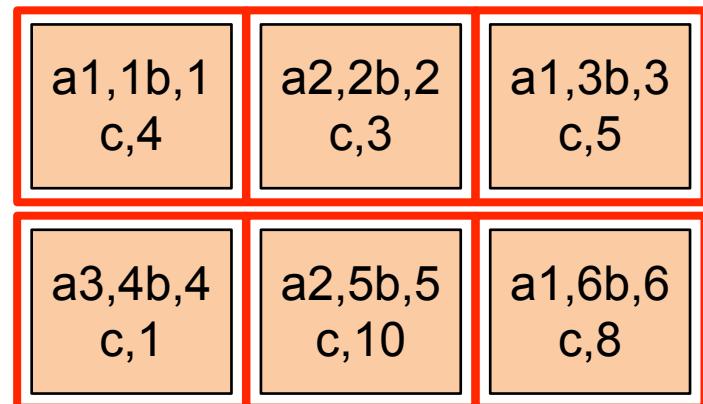
# Scaling to Disk: Group By Aggregate

A	B	C	D
a1	1b	1c	4
a2	2b	2c	3
a1	3b	3c	5
a3	4b	4c	1
a2	5b	5c	10
a1	6b	6c	8

R

SELECT A, SUM(D)  
FROM R GROUP BY A

Row-store:



Hash table in DRAM

A	Running Info.
a1	4 -> 9 -> 17
a2	3 -> 13
a3	1

- ❖ Note that the sum for each group is constructed *incrementally*
- ❖ I/O cost: 6 (read) + output # pages (write); just one filescan again!

**Q: But what if hash table > DRAM size?**

# Scaling to Disk: Group By Aggregate

SELECT A, SUM(D) FROM R GROUP BY A

*Q: But what if hash table > DRAM size?*

- ❖ Program will likely just crash! OS may keep swapping pages of hash table to/from disk; aka “thrashing”

*Q: How to scale to large number of groups?*

- ❖ Divide and conquer!  
Split up R based on values of A
- ❖ HT for each split may fit in DRAM alone
- ❖ Reduce running info. size if possible

The screenshot shows the DASK website's homepage. The navigation bar includes links for Get Started, Algorithms, Setup, and Community. On the left, there's a sidebar with sections for BASIC EXAMPLES (Dask Arrays, Dask Bags, Dask DataFrames, Custom Workloads with Dask Delayed, Custom Workloads with Futures, Dask for Machine Learning, Xarray with Dask Arrays) and DATAFRAMES (DataFrames: Read and Write Data, DataFrame: GroupBy). The main content area has a heading "Many groups". It explains that by default groupby-aggregations return results as single-partition Dask dataframes. It then discusses cases where many groups might not fit into a single Pandas dataframe, requiring multiple partitions. It shows two examples of computational graphs: one for a small number of groups (df.groupby('name').x.mean().visualize(node\_attr={'penwidth': '6'})) and another for a larger number of groups (df.groupby('id').x.mean(split\_out=4).visualize(node\_attr={'penwidth': '6'})).

**Ad:** Take CSE 132C for more on how GROUP BY is scaled

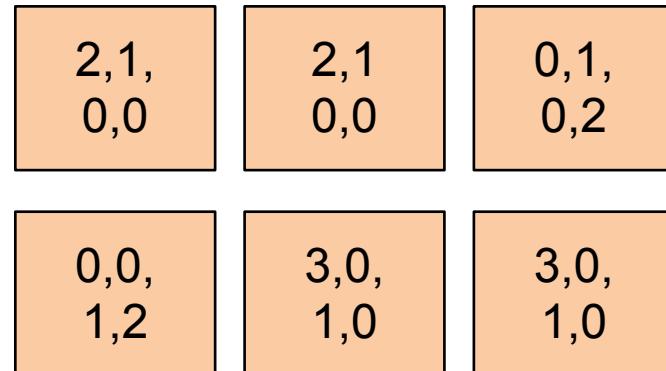
# Scaling to Disk: Matrix Sum/Norms

2	1	0	0
2	1	0	0
0	1	0	2
0	0	1	2
3	0	1	0
3	0	1	0

$M_{6 \times 4}$

$$\|M\|_2^2$$

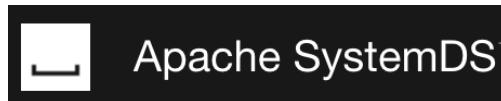
Row-store:



- ❖ Again, straightforward **filescan** data access pattern
  - ❖ Very similar to relational simple aggregate
  - ❖ Running info. in DRAM for sum of squares of cells
    - ❖  $0 \rightarrow 5 \rightarrow 10 \rightarrow 15 \rightarrow 20 \rightarrow 30 \rightarrow 40$
- ❖ I/O cost: **6** (read) + output # pages (write)
- ❖ Col-store and tiled-store also have I/O cost 6; why?

# Scalable Matrix/Tensor Algebra

- ❖ In general, tiled partitioning is more common for matrix/tensor ops
  - ❖ More DRAM/cache-efficient implementations
  
- ❖ DRAM-to-disk scaling:
  - ❖ pBDR, SystemDS, and Dask Arrays for matrices
  - ❖ SciDB, Xarray for n-d arrays
- ❖ CUDA for DRAM-GPU caches scaling of matrix/tensor ops



The screenshot shows a web browser window with the URL [examples.dask.org/array.html](http://examples.dask.org/array.html). The page is titled "DASK" and features a sidebar with links to various machine learning and data processing examples. On the right, there is a section titled "Create Random array" which describes generating a 10000x10000 array of random numbers. Below this is a code snippet:

```
import dask.array as da
x = da.random.random((10000, 10000), chunks=(1000, 1000))
x
```

At the bottom, there is a table showing array metadata and a visual representation of the array's chunking:

	Array	Chunk
Bytes	800.00 MB	8.00 MB
Shape	(10000, 10000)	(1000, 1000)
Count	100 Tasks	100 Chunks
Type	float64	numpy.ndarray

A 10x10 grid of squares represents the 1000x1000 chunked array, with the number "10000" indicating the total size.

# Numerical Optimization in ML

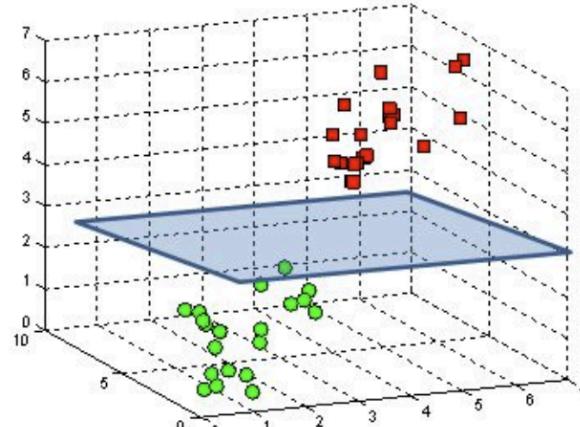
- ❖ Many regression and classification models in ML are formulated as a (constrained) *minimization* problem
  - ❖ E.g., logistic and linear regression, linear SVM, etc.
  - ❖ Aka “Empirical Risk Minimization” (ERM) approach
  - ❖ Computes “loss” of predictions over labeled examples

$$\mathbf{w}^* = \operatorname{argmin}_{\mathbf{w}} \sum_{i=1}^n l(y_i, f(\mathbf{w}, x_i))$$

- ❖ Hyperplane-based models aka Generalized Linear Models (GLMs) use  $f()$  that is a scalar function of distances:

$$\mathbf{w}^T x_i$$

A hyperplane in  $\mathbb{R}^3$  is a plane

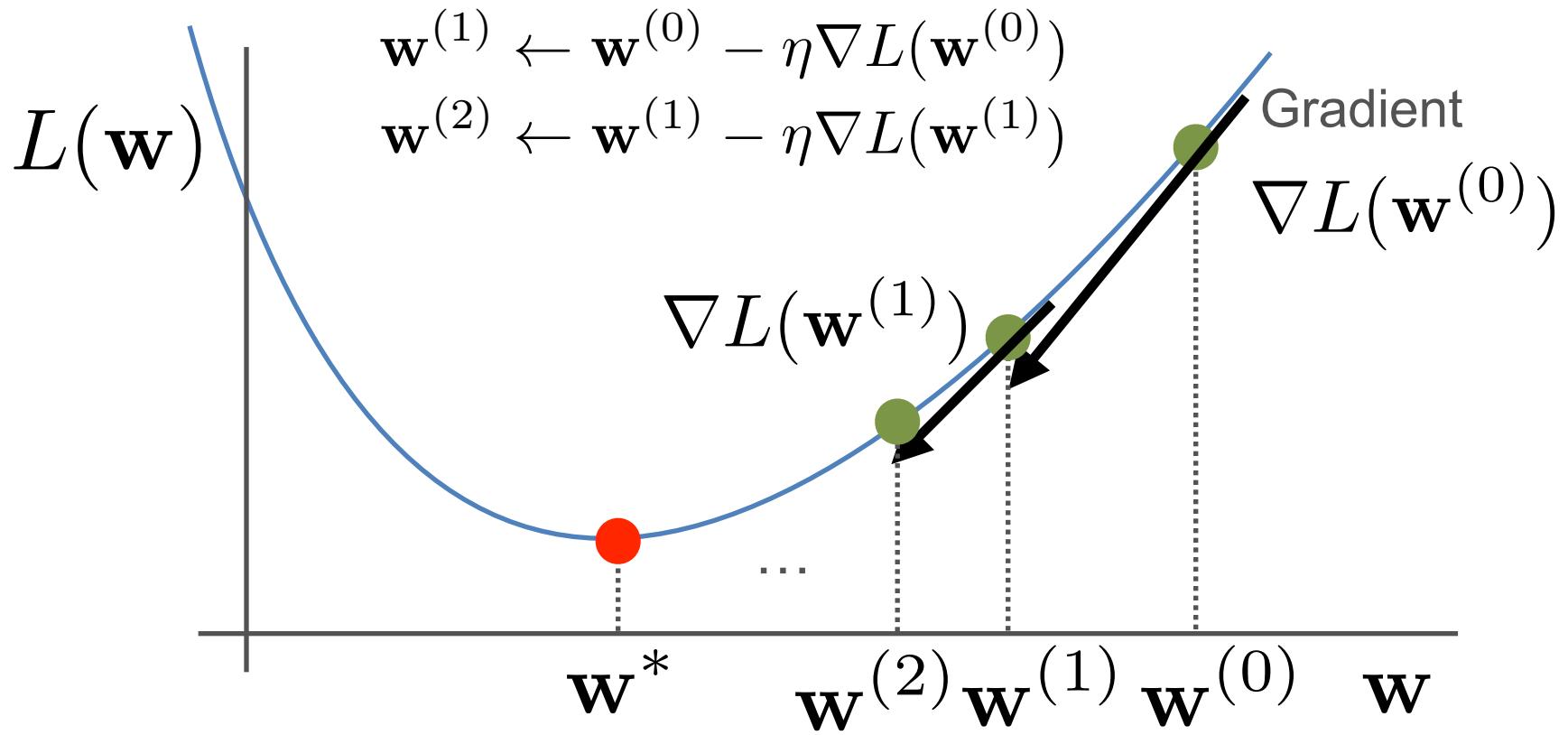


# Batch Gradient Descent for ML

$$L(\mathbf{w}) = \sum_{i=1}^n l(y_i, f(\mathbf{w}, x_i))$$

- ❖ In many cases, **loss function**  $l()$  is **convex**; so is  $L()$
- ❖ Closed-form minimization typically infeasible
- ❖ **Batch Gradient Descent:**
  - ❖ Iterative numerical procedure to find an optimal  $\mathbf{w}$
  - ❖ Initialize  $\mathbf{w}$  to some value  $\mathbf{w}^{(0)}$
  - ❖ Compute **gradient**:  
$$\nabla L(\mathbf{w}^{(k)}) = \sum_{i=1}^n \nabla l(y_i, f(\mathbf{w}^{(k)}, x_i))$$
  - ❖ Descend along gradient:  
(Aka **Update Rule**)  
$$\mathbf{w}^{(k+1)} \leftarrow \mathbf{w}^{(k)} - \eta \nabla L(\mathbf{w}^{(k)})$$
  - ❖ Repeat until we get close to  $\mathbf{w}^*$ , aka **convergence**

# Batch Gradient Descent for ML



- ❖ Learning rate is a **hyper-parameter** selected by user or “AutoML” tuning procedures
- ❖ Number of iterations/epochs of BGD also hyper-parameter

# Data Access Pattern of BGD at Scale

- ❖ The data-intensive computation in BGD is the gradient
  - ❖ In scalable ML, dataset D may not fit in DRAM
  - ❖ Model  $\mathbf{w}$  is typically small and DRAM-resident

$$\nabla L(\mathbf{w}^{(k)}) = \sum_{i=1}^n \nabla l(y_i, f(\mathbf{w}^{(k)}, x_i))$$

**Q:** *What SQL op is this reminiscent of?*

- ❖ Gradient is like SQL SUM over vectors (one per example)
- ❖ At each epoch, 1 **filescan** over D to get gradient
- ❖ Update of  $\mathbf{w}$  happens normally in DRAM
- ❖ Monitoring across epochs for convergence needed
- ❖ Loss function  $L()$  is also just a SUM in a similar manner

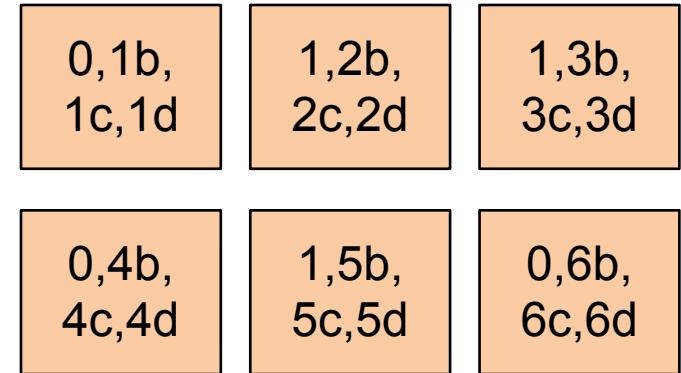
# I/O Cost of Scalable BGD

D

Y	X1	X2	X3
0	1b	1c	1d
1	2b	2c	2d
1	3b	3c	3d
0	4b	4c	4d
1	5b	5c	5d
0	6b	6c	6d

$$\nabla L(\mathbf{w}^{(k)}) = \sum_{i=1}^n \nabla l(y_i, f(\mathbf{w}^{(k)}, x_i))$$

Row-store:



- ❖ Straightforward **filescan** data access pattern for SUM
  - ❖ Similar I/O behavior as non-dedup. project and simple SQL aggregates
- ❖ I/O cost: 6 (read) + output # pages (write for final  $\mathbf{w}$ )

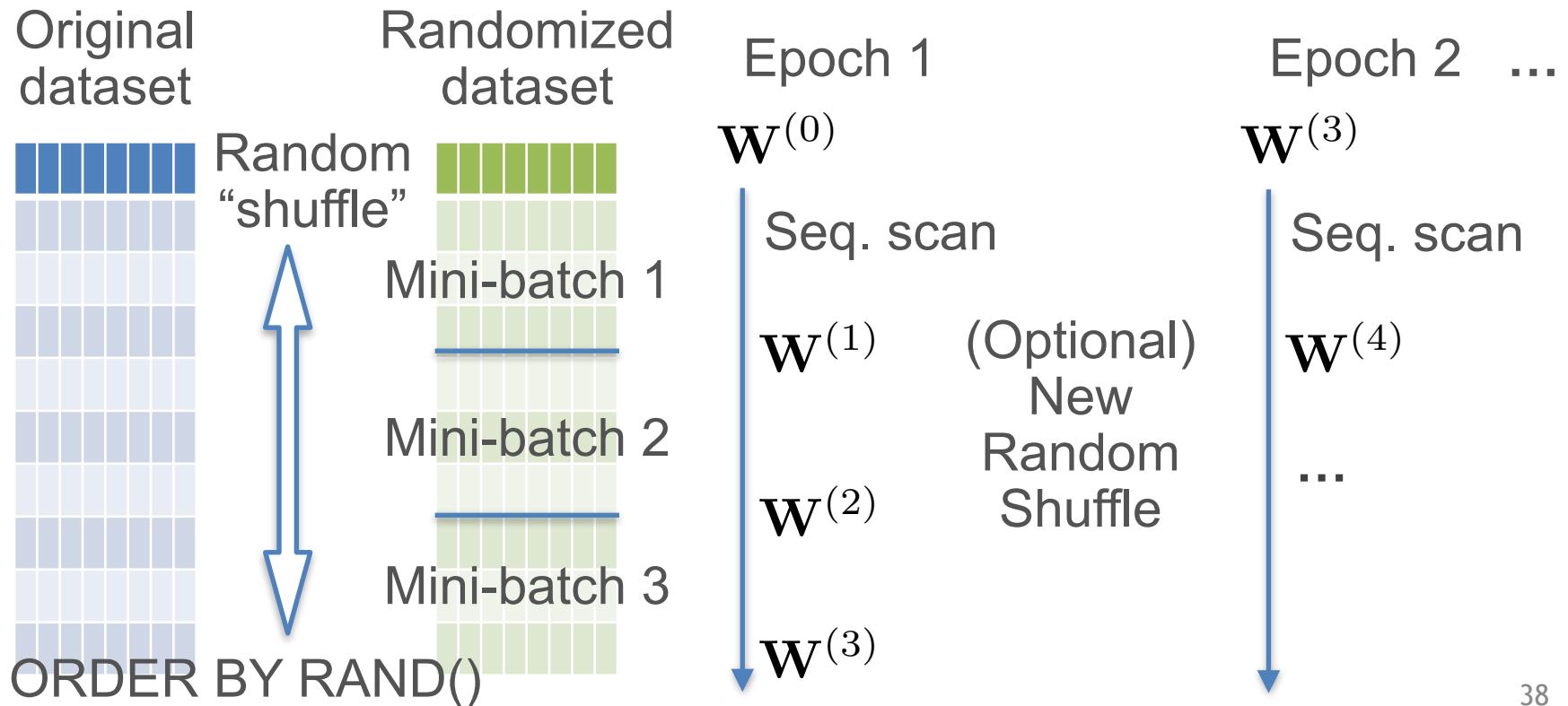
# Stochastic Gradient Descent for ML

- ❖ Two key cons of BGD:
  - ❖ Often, too many epochs to reach optimal
  - ❖ Each update of  $w$  needs full scan: costly I/Os
- ❖ Stochastic GD (SGD) mitigates both cons
- ❖ **Basic Idea:** Use a *sample (mini-batch)* of  $D$  to approximate gradient instead of “full batch” gradient
  - ❖ Done *without replacement*
  - ❖ Randomly reorder/shuffle  $D$  before every epoch
  - ❖ Sequential pass: sequence of mini-batches
- ❖ Another big pro of SGD: works well for *non-convex* loss too, especially DL; BGD does not
- ❖ SGD often called the “workhorse” of modern ML/DL

# Access Pattern of Scalable SGD

$$\mathbf{W}^{(t+1)} \leftarrow \mathbf{W}^{(t)} - \eta \nabla \tilde{L}(\mathbf{W}^{(t)}) \quad \nabla \tilde{L}(\mathbf{W}) = \sum_{i \in B} \nabla l(y_i, f(\mathbf{W}, x_i))$$

Sample mini-batch from dataset without replacement



# I/O Cost of Scalable SGD

- ❖ I/O cost of random shuffle is non-trivial; need so-called “external merge sort” (skipped in this course)
  - ❖ Typically amounts to 1 or 2 passes over file
- ❖ Mini-batch gradient computations: 1 **filescan** per epoch:
  - ❖ As filescan proceeds, count # examples seen, accumulate per-example gradient for mini-batch
  - ❖ Typical mini-batch sizes: 10s to 1000s
  - ❖ Orders of magnitude more model updates than BGD!
- ❖ Total I/O cost per epoch: 1 shuffle cost + 1 filescan cost
  - ❖ Often, shuffling only once upfront suffices
- ❖ Loss function  $L()$  computation is same as before (for BGD)

# Scaling Data Science Operations

- ❖ Scalable data access for key representative examples of programs/operations that are ubiquitous in data science:
  - ❖ DB systems:
    - ❖ Relational select
    - ❖ Non-deduplicating project
    - ❖ Simple SQL aggregates
    - ❖ SQL GROUP BY aggregates
  - ❖ ML systems:
    - ❖ Matrix sum/norms
    - ❖ (Stochastic) Gradient Descent

# Outline

- ❖ Basics of Parallelism
  - ❖ Task Parallelism; Dask
  - ❖ Single-Node Multi-Core; SIMD; Accelerators
- ❖ Basics of Scalable Data Access
  - ❖ Paged Access; I/O Costs; Layouts/Access Patterns
  - ❖ Scaling Data Science Operations
- ❖ Data Parallelism: Parallelism + Scalability
  - ❖ Data-Parallel Data Science Operations
  - ❖ Optimizations and Hybrid Parallelism

# Review Questions

1. What are the 4 main regimes of scalable data access?
2. Briefly explain 1 pro and 1 con of scaling with local disk vs scaling with remote reads.
3. You are given a DataFrame serialized as a 100 GB Parquet columnar file. All 20 columns are of same fixed-length data type. You compute a sum over 4 columns. What is the I/O cost (in GB)?
4. Which is the most flexible layout format for 2-D structured data?
5. You layout a 1 TB matrix in tile format with a shape 2000x500. What is the I/O cost (in GB) of computing its full matrix sum?
6. Briefly explain 1 pro and 1 con of SGD vs BGD.
7. Suppose you use scalable SGD to train a DL model. The dataset has 100 mil examples. You use a mini-batch size of 50. How many iterations (number of model update steps) will SGD finish in 20 epochs?
8. What is the precise runtime tradeoff involved in shuffle-once-upfront vs shuffle-every-epoch for SGD?

Optional: More Examples of Scaling Data  
Science Operations  
Not included in syllabus

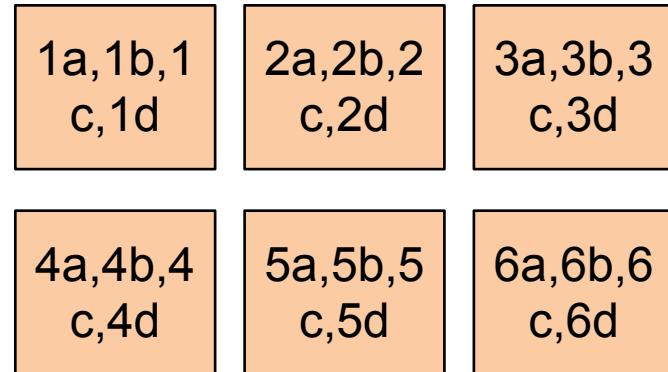
# Scaling to Disk: Relational Select

A	B	C	D
1a	1b	1c	1d
2a	2b	2c	2d
3a	3b	3c	3d
4a	4b	4c	4d
5a	5b	5c	5d
6a	6b	6c	6d

R

$$\sigma_{B=\text{"3b"}(R)}$$

Row-store:

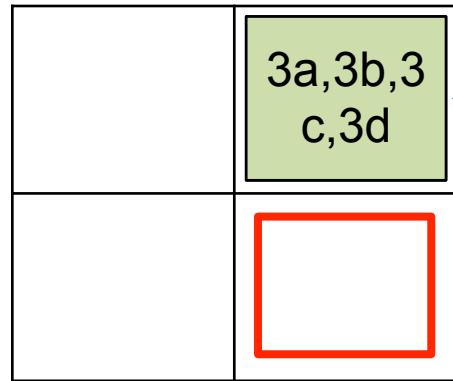


- ❖ Straightforward **filescan** data access pattern
  - ❖ Read pages/chunks from disk to DRAM one by one
  - ❖ CPU applies predicate to tuples in pages in DRAM
  - ❖ Copy satisfying tuples to temporary output pages
  - ❖ Use LRU for cache replacement, if needed
- ❖ I/O cost: 6 (read) + output # pages (write)

# Scaling to Disk: Relational Select

$$\sigma_{B=\text{"3b"}(R)}$$

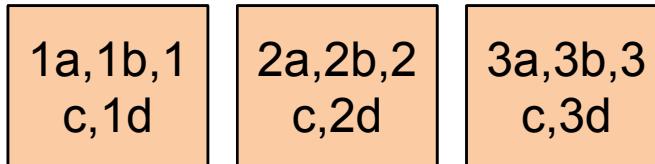
**OS Cache  
in DRAM**



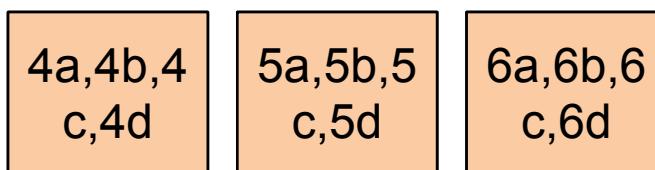
Reserved for writing output data of program (may be spilled to a temp. file)

CPU finds a matching tuple!  
Copies that to output page

**Disk**



Need to evict some page  
LRU says kick out page 1  
Then page 2 and so on



...

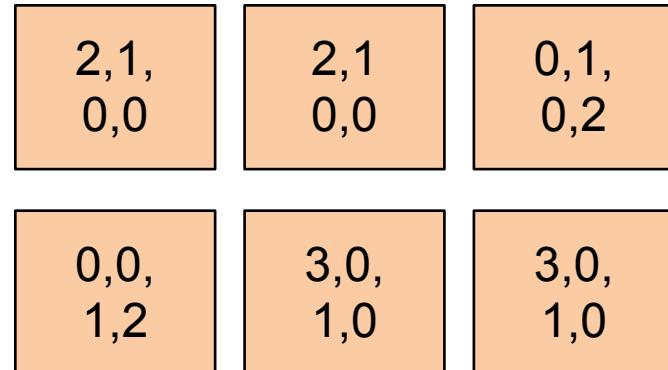
# Scaling to Disk: Gramian Matrix

2	1	0	0
2	1	0	0
0	1	0	2
0	0	1	2
3	0	1	0
3	0	1	0

$M_{6 \times 4}$

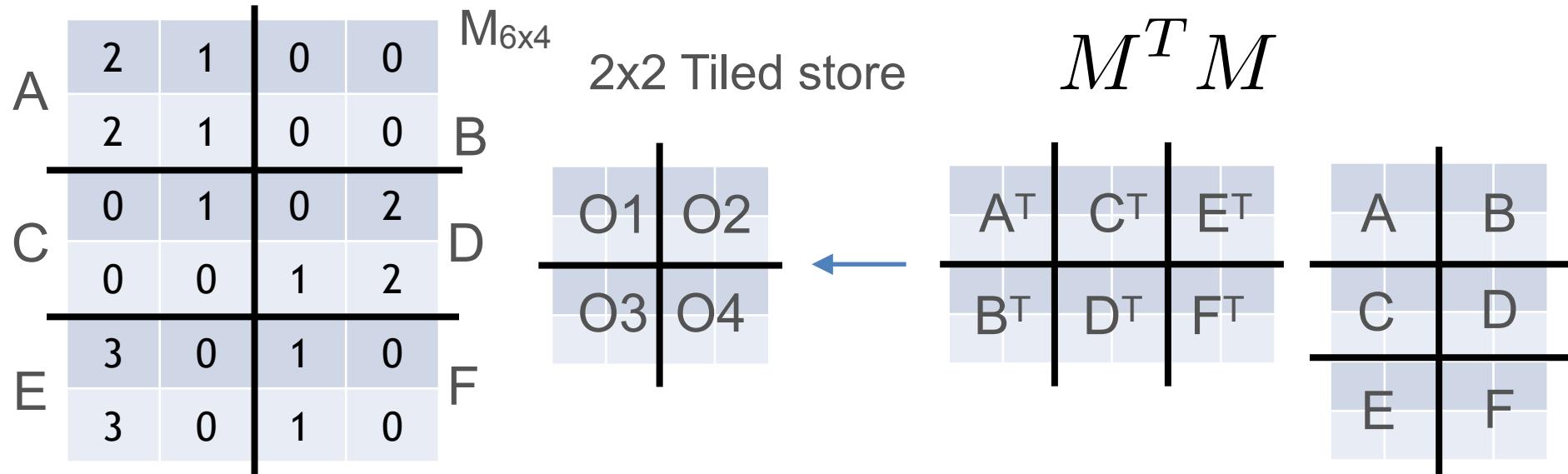
$$M^T M$$

Row-store:



- ❖ A bit tricky, since output may not fit entirely in DRAM
  - ❖ Similar to GROUP BY difficult case
- ❖ Output here is 4x4, i.e., 4 pages; only 3 can be in DRAM!
  - ❖ Each row will need to update entire output matrix
  - ❖ Row-store can be a poor fit for such matrix algebra
- ❖ What about col-store or tiled-store?

# Scaling to Disk: Gramian Matrix



- ❖ Read  $A, C, E$  one by one to get  $O_1 = A^T A + C^T C + E^T E$ ;  $O_1$  is incrementally computed; write  $O_1$  out; I/O: 3 (r) + 1 (w)
- ❖ Likewise with  $B, D, F$  for  $O_4$ ; I/O: 3 (r) + 1 (w)
- ❖ Read  $A, B$  and put  $A^T B$  in  $O_2$ ; read  $C, D$  to add  $C^T D$  to  $O_2$ ; read  $E, F$  to add  $E^T F$  to  $O_2$ ; write  $O_2$  out; I/O: 6 + 1
- ❖ Likewise with  $B, A; D, C; F, E$  for  $O_3$ ; I/O: 6 + 1
- ❖ Max I/O cost: **18 (r) + 4(w)**; *scalable on both dimensions!*

# DSC 102

# Systems for Scalable Analytics

Arun Kumar

Topic 3: Parallel and Scalable Data Processing

Part 3: Data Parallelism

Ch. 9.4, 12.2, 14.1.1, 14.6, 22.1-22.3, 22.4.1, 22.8 of Cow Book

Ch. 5, 6.1, 6.3, 6.4 of MLSys Book

# Outline

- ❖ Basics of Parallelism
  - ❖ Task Parallelism; Dask
  - ❖ Single-Node Multi-Core; SIMD; Accelerators
- ❖ Basics of Scalable Data Access
  - ❖ Paged Access; I/O Costs; Layouts/Access Patterns
  - ❖ Scaling Data Science Operations
- ❖ Data Parallelism: Parallelism + Scalability
  - ❖ Data-Parallel Data Science Operations
  - ❖ Optimizations and Hybrid Parallelism

# Introducing Data Parallelism

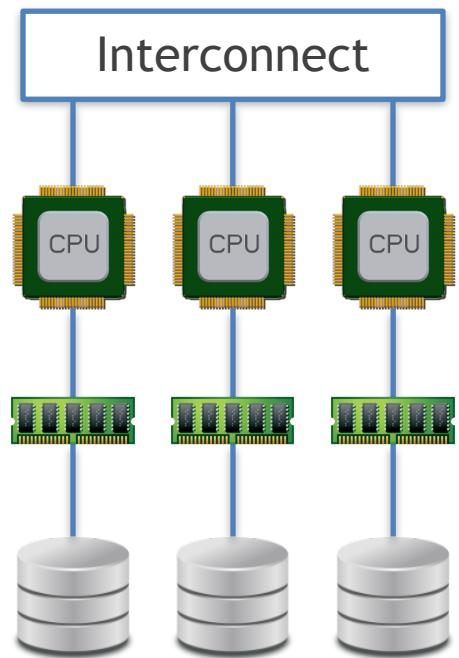
**Basic Idea of Scalability:** Split data file (virtually or physically) and stage reads/writes of its pages between disk and DRAM

**Q:** *What is “data parallelism”?*

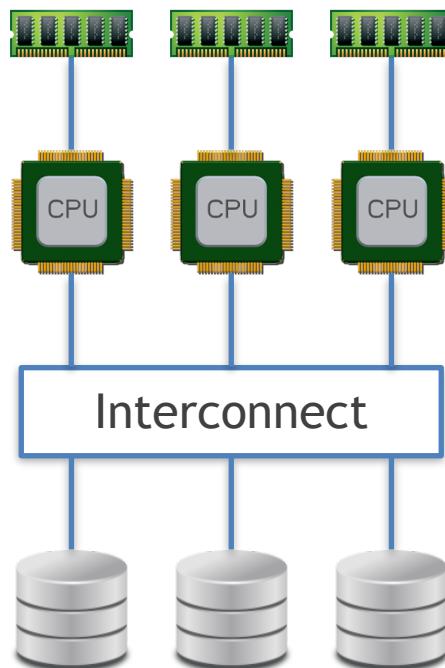
**Data Parallelism:** Partition large data file *physically* across nodes/workers; within worker: DRAM-based or disk-based

- ❖ The most common approach to marrying *parallelism* and *scalability* in data systems
- ❖ Generalization of SIMD and SPMD idea from parallel processors to large-scale data and multi-worker/multi-node setting
- ❖ Distributed-memory vs Distributed-disk

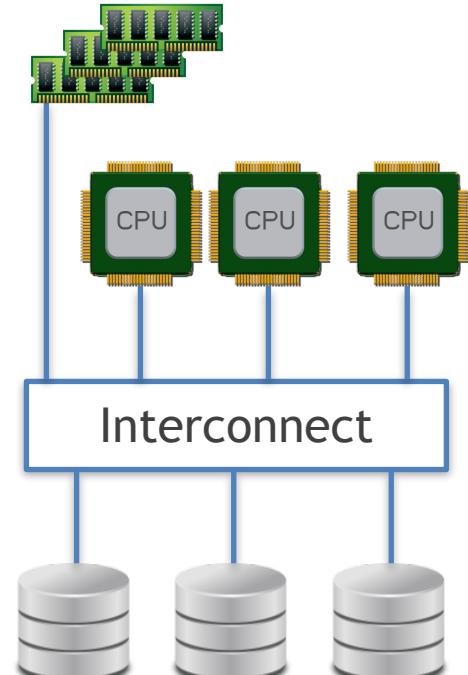
# 3 Paradigms of Multi-Node Parallelism



Shared-Nothing  
Parallelism



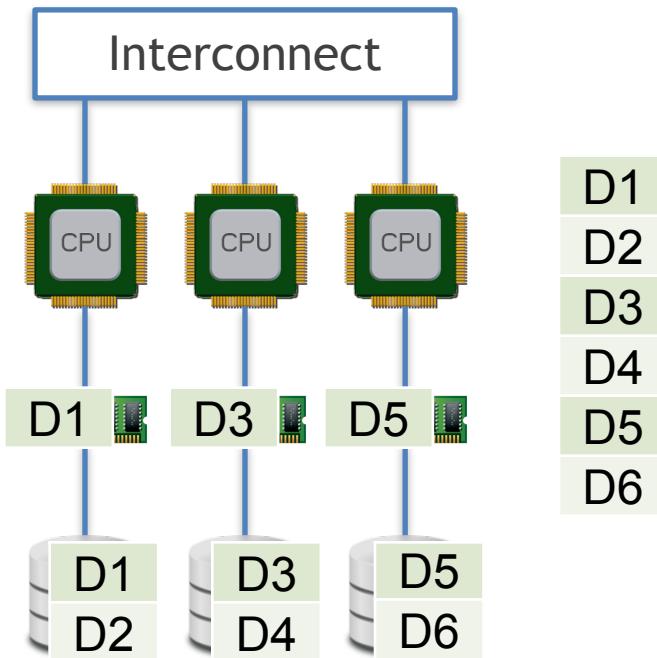
Shared-Disk  
Parallelism



Shared-Memory  
Parallelism

Data parallelism is technically *orthogonal* to these 3 paradigms  
but most commonly paired with shared-nothing

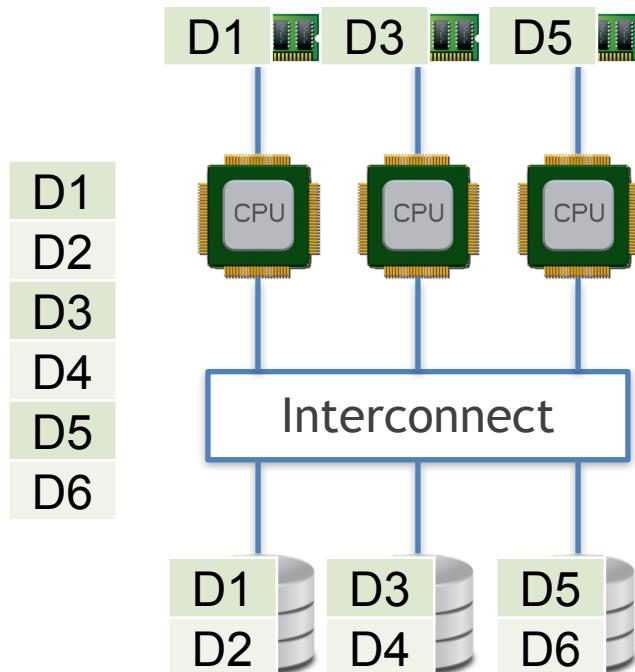
# Shared-Nothing Data Parallelism



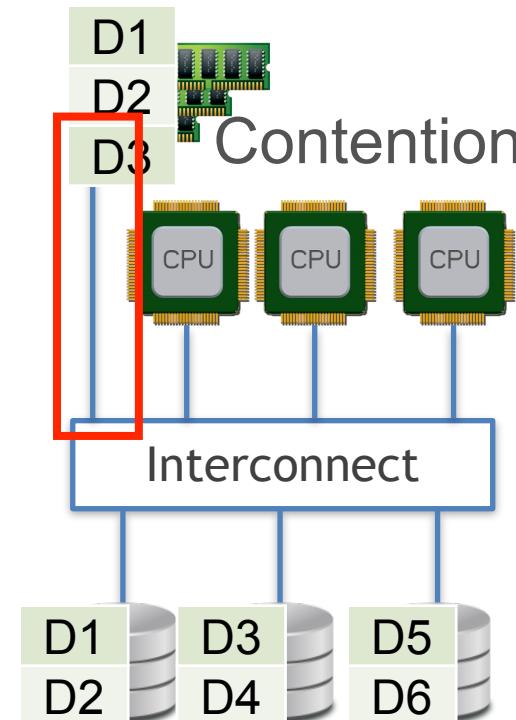
Shared-Nothing  
Parallel Cluster

- ❖ Partitioning a data file across nodes is aka **sharding**
- ❖ Part of a stage in data processing workflows called **Extract-Transform-Load (ETL)**
- ❖ ETL is an umbrella term for all kinds of processing done to the data file before it is ready for users to query, analyze, etc.
- ❖ Sharding, compression, file format conversions, etc.

# Data Parallelism in Other Paradigms?



Shared-Disk  
Parallel Cluster



Shared-Memory  
Parallel Cluster

# Data Partitioning Strategies

- ❖ Row-wise/*horizontal* partitioning is most common (sharding)
- ❖ 3 common schemes (given k nodes):
  - ❖ **Round-robin**: assign tuple i to node  $i \text{ MOD } k$
  - ❖ **Hashing-based**: needs hash partitioning attribute(s)
  - ❖ **Range-based**: needs ordinal partitioning attribute(s)
- ❖ **Tradeoffs:**
  - ❖ Hashing-based most common in practice for RA/SQL
  - ❖ Range-based often good for range predicates in RA/SQL
  - ❖ But all 3 are often OK for many ML workloads (why?)
- ❖ **Replication** of partition across nodes (e.g., 3x) is common to enable “*fault tolerance*” and better parallel *runtime performance*

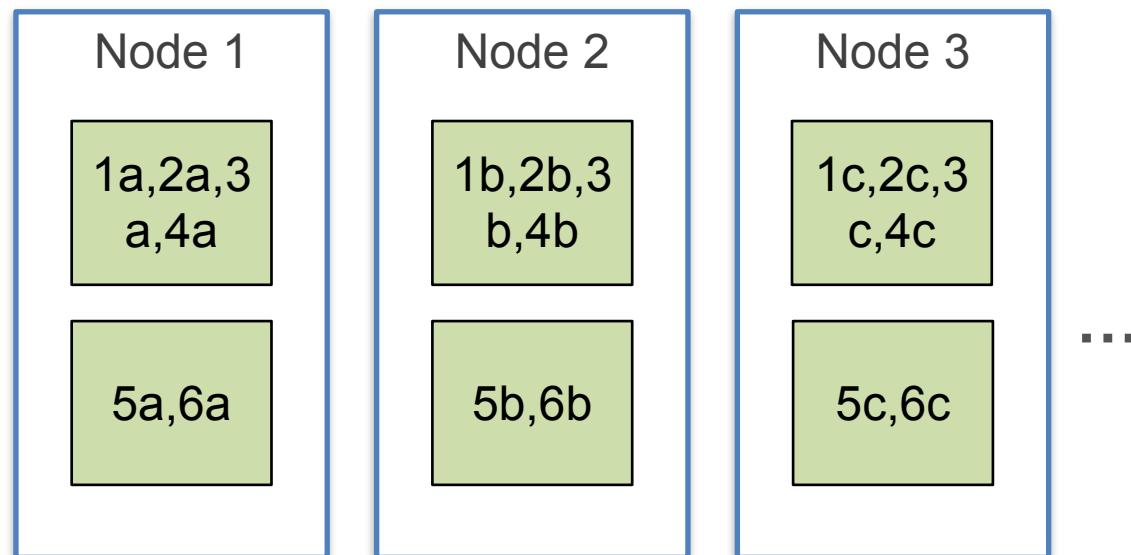
# Other Forms of Data Partitioning

- ❖ Just like with disk-aware data layout on single-node, we can partition a large data file across workers in other ways too:

R

A	B	C	D
1a	1b	1c	1d
2a	2b	2c	2d
3a	3b	3c	3d
4a	4b	4c	4d
5a	5b	5c	5d
6a	6b	6c	6d

## Columnar Partitioning



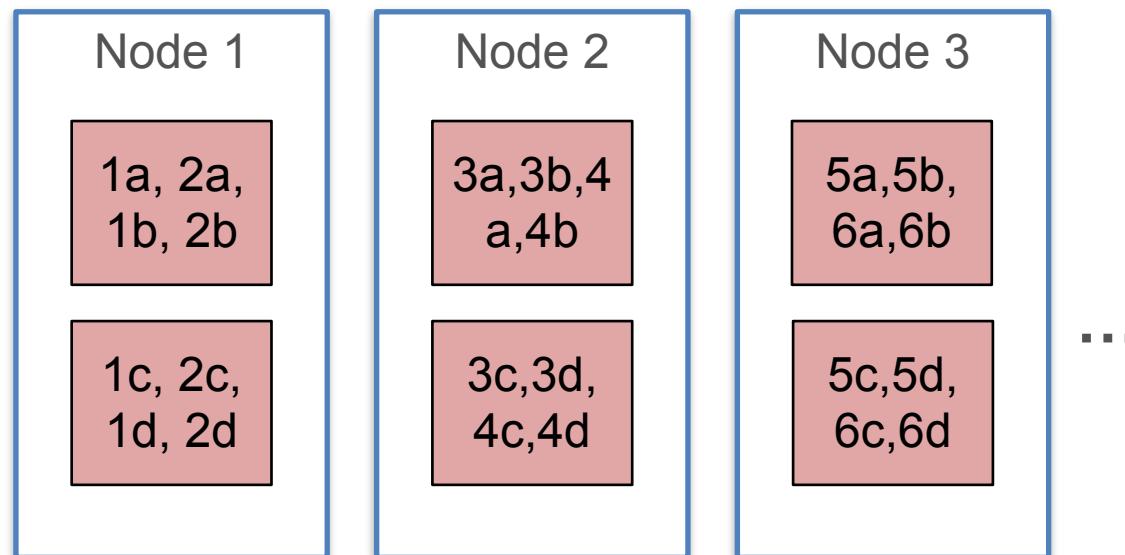
# Other Forms of Data Partitioning

- ❖ Just like with disk-aware data layout on single-node, we can partition a large data file across workers in other ways too:

R

A	B	C	D
1a	1b	1c	1d
2a	2b	2c	2d
3a	3b	3c	3d
4a	4b	4c	4d
5a	5b	5c	5d
6a	6b	6c	6d

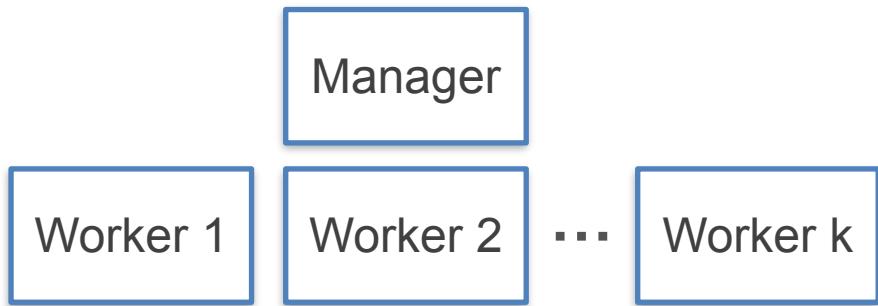
Hybrid/Tiled Partitioning



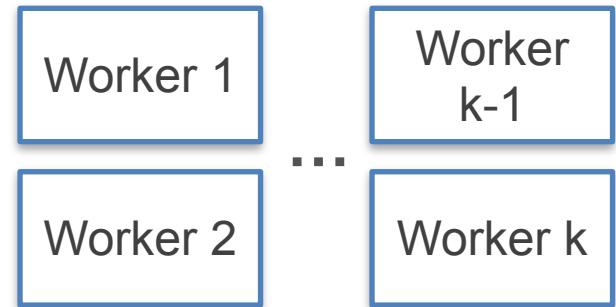
# Cluster Architectures

**Q:** *What is the protocol for cluster nodes to talk to each other?*

## Manager-Worker Architecture



## Peer-to-Peer Architecture

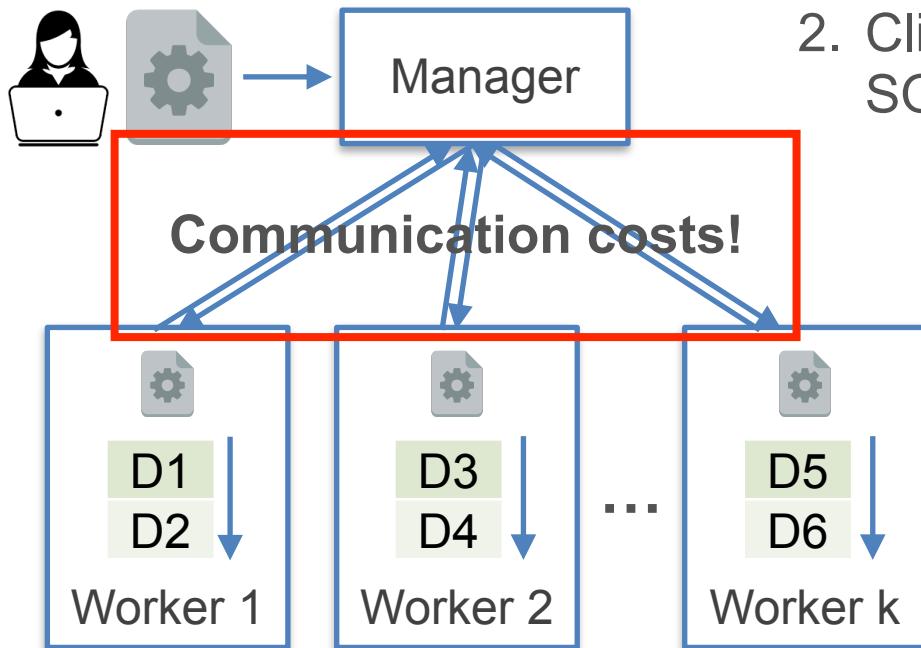


- ❖ 1 (or few) special node called **Manager** (aka “Server” or archaic “Master”); 1 or more **Workers**
- ❖ Manager tells workers what to do and when to talk to other nodes
- ❖ Most common in data systems (e.g., Dask, Spark, par. RDBMS, etc.)

- ❖ No special manager
- ❖ Workers talk to each other directly
- ❖ E.g., Horovod
- ❖ Aka Decentralized (vs Centralized)

# Bulk Synchronous Parallelism (BSP)

- ❖ Most common protocol of data parallelism in data systems (e.g., in parallel RDBMSs, Hadoop, Spark)
- ❖ Shared-nothing sharding + manager-worker architecture



Aka (Barrier) Synchronization

1. Sharded data file on workers
2. Client gives program to manager (e.g., SQL query, ML training, etc.)
3. Manager *divides* first piece of work among workers
4. Workers work *independently* on self's data partition (cross-talk can happen if Manager asks)
5. Worker sends partial results to Manager after one
6. Manager **waits** till all k done
7. Go to step 3 for next piece

# Speedup Analysis/Limits of of BSP

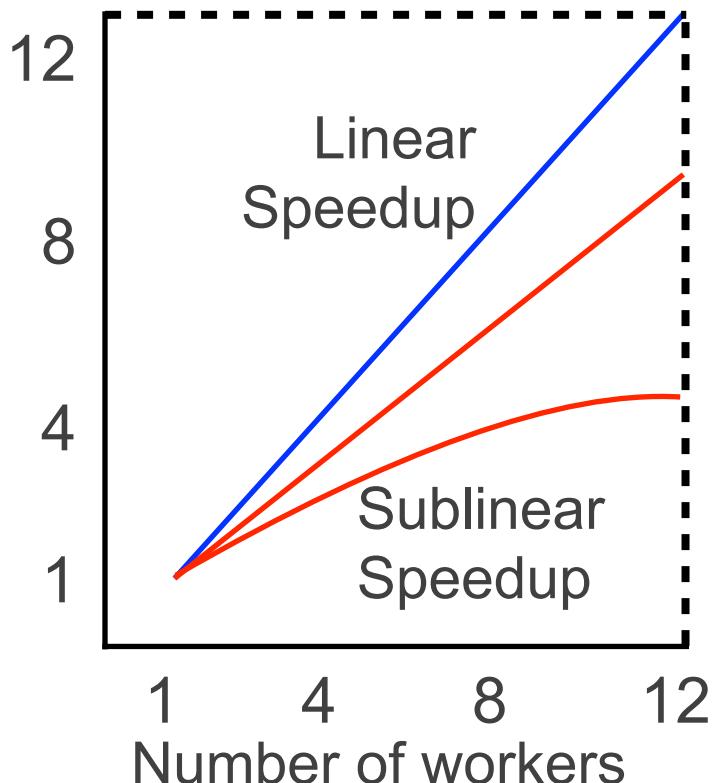
*Q: What is the speedup yielded by BSP?*

$$\text{Speedup} = \frac{\text{Completion time given only 1 worker}}{\text{Completion time given } k (>1) \text{ workers}}$$

- ❖ Cluster overhead factors that hurt speedup:
  - ❖ **Per-worker:** startup cost; tear-down cost
  - ❖ **On manager:** dividing up the work; collecting/unifying partial partial results from workers
  - ❖ **Communication costs:** talk between manager-worker and across workers (when asked by manager)
  - ❖ Barrier synchronization suffers from “**stragglers**” due to skews in shard sizes and/or worker capacities

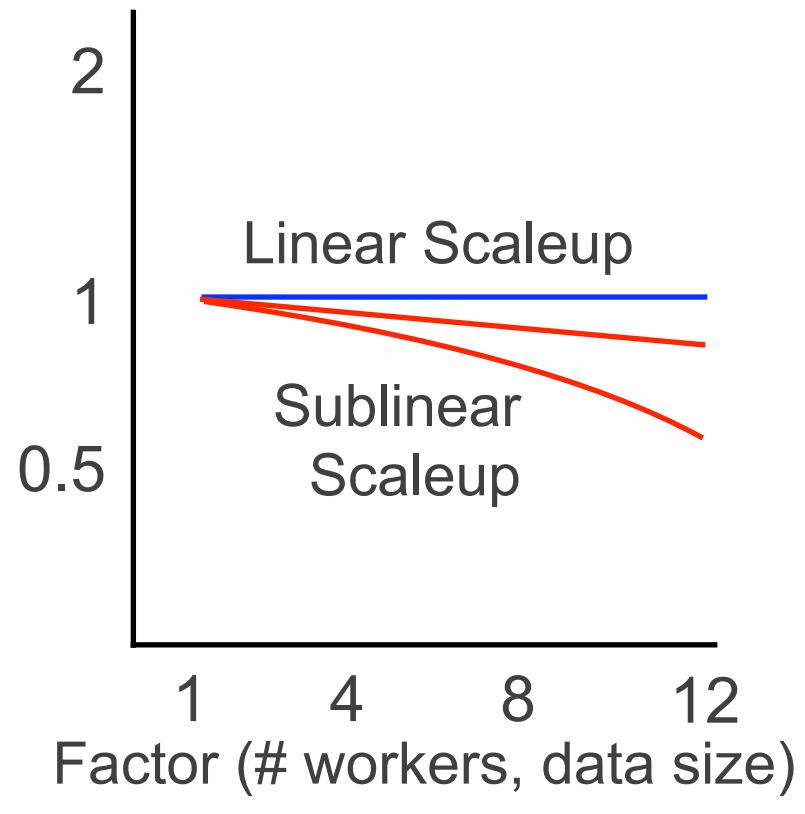
# Quantifying Benefit of Parallelism

Runtime speedup (fixed data size)



**Speedup** plot / Strong scaling

Runtime speedup



**Scaleup** plot / Weak scaling

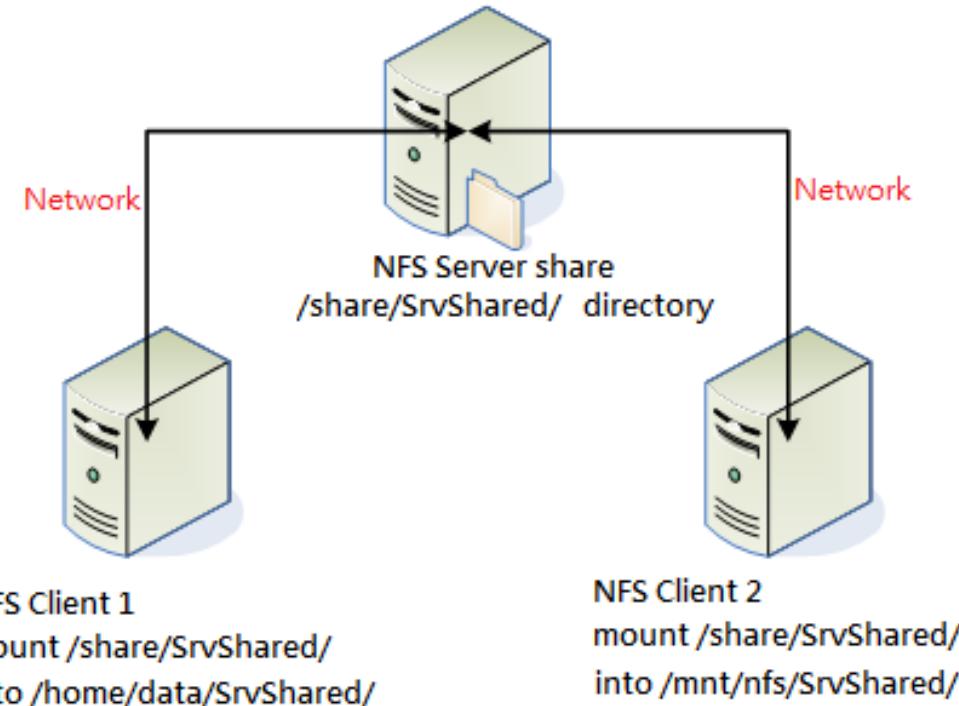
**Q:** Is superlinear speedup/scaleup ever possible?

# Distributed Filesystems

- ❖ Recall definition of file; *distributed file* generalizes it to a cluster of networked disks and OSs
- ❖ **Distributed filesystem (DFS)** is a cluster-resident filesystem to manage distributed files
  - ❖ A *layer of abstraction* on top of local filesystems
  - ❖ Nodes manage local data as if they are local files
  - ❖ *Illusion of a one global file*: DFS APIs let nodes access data sitting on other nodes
  - ❖ 2 main variants: Remote DFS vs In-Situ DFS
    - ❖ **Remote DFS**: Files reside elsewhere and read/written on demand by workers
    - ❖ **In-Situ DFS**: Files resides on cluster where workers exist

# Network Filesystem (NFS)

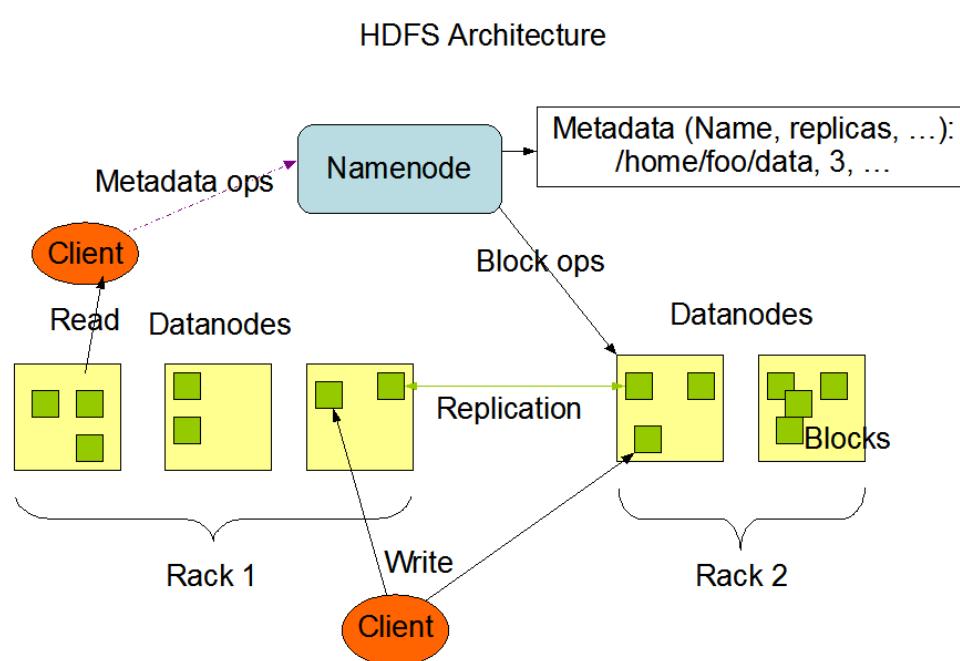
- ❖ An old remote DFS (c. 1980s) with simple client-server architecture for *replicating* files over the network



- ❖ Main pro: *simplicity* of setup and usage
- ❖ But many cons:
  - ❖ Not scalable to *very* large files
  - ❖ Full data replication
  - ❖ High contention for concurrent reads/writes
  - ❖ Single-point of failure

# Hadoop Distributed File System (HDFS)

- ❖ Most popular in-situ DFS (c. late 2000s); part of Hadoop; open source spinoff of Google File system (GFS)
- ❖ *Highly scalable*; scales to 10s of 1000s of nodes, PB files



- ❖ Designed for clusters of cheap commodity nodes
- ❖ *Parallel* reads/writes of sharded data “blocks”
- ❖ Replication of blocks to improve *fault tolerance*
- ❖ Cons: Read-only + batch-append (no fine-grained updates/writes)

# Hadoop Distributed File System (HDFS)

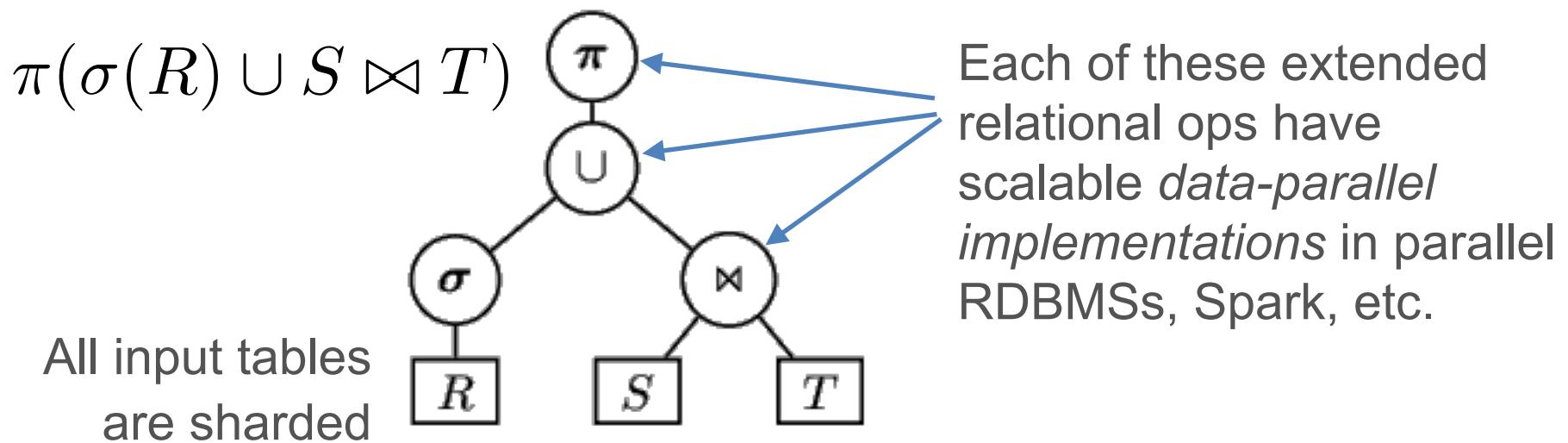
- ❖ NameNode's roster maps data blocks to DataNodes/IPs
- ❖ A distributed file on HDFS is just a directory (!) with individual filenames for each data block and metadata files

```
 ${dfs.datanode.data.dir}/
   +-- current
   |   +-- BP-526805057-127.0.0.1-1411980876842
   |   |   +-- current
   |   |   +-- VERSION
   |   |   +-- finalized
   |   |   |   +-- blk_1073741825
   |   |   |   +-- blk_1073741825_1001.meta
   |   |   |   +-- blk_1073741826
   |   |   |   +-- blk_1073741826_1002.meta
   |   |   +-- rbw
   |   +-- VERSION
   +-- in_use.lock
```

- ❖ HDFS *data block size* and *replication factor* are configurable parameters; default are 128 MB and 3x

# Data-Parallel Dataflow/Workflow

- ❖ **Data-Parallel Dataflow:** A dataflow graph with ops wherein each operation is executed in a data-parallel manner
- ❖ **Data-Parallel Workflow:** A generalization; each vertex a whole task/process that is run in a data-parallel manner



**Q: So how do we run data sci. ops in data-parallel manner?**

# Outline

- ❖ Basics of Parallelism
  - ❖ Task Parallelism; Dask
  - ❖ Single-Node Multi-Core; SIMD; Accelerators
- ❖ Basics of Scalable Data Access
  - ❖ Paged Access; I/O Costs; Layouts/Access Patterns
  - ❖ Scaling Data Science Operations
- ❖ Data Parallelism: Parallelism + Scalability
- ❖ Data-Parallel Data Science Operations
  - ❖ Optimizations and Hybrid Parallelism

# Data-Parallel Data Science Ops

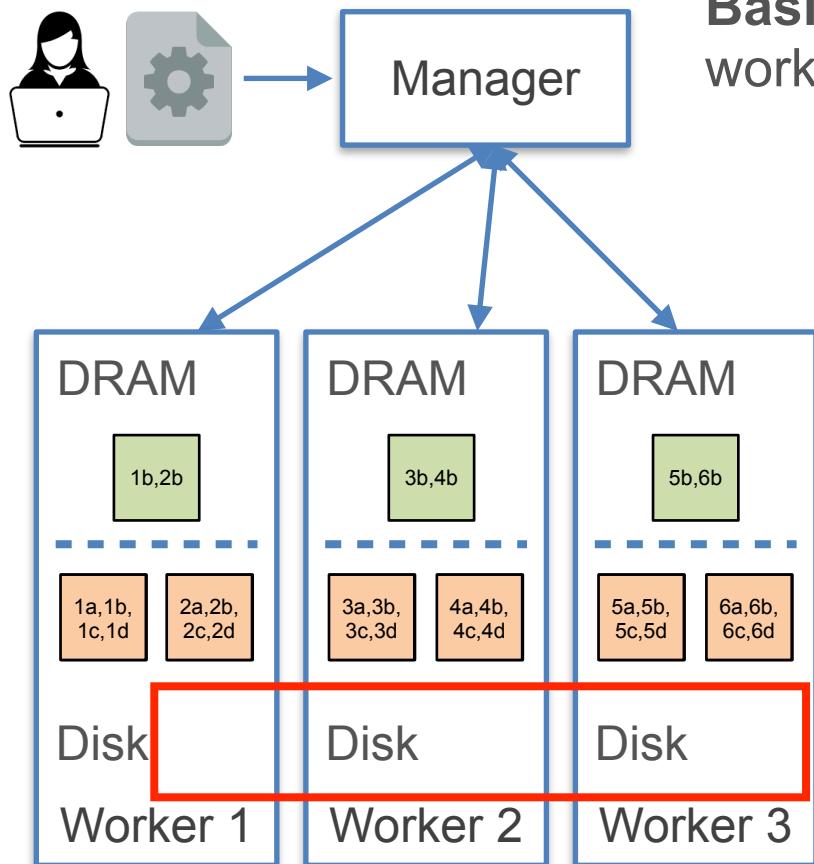
- ❖ Data parallelism for key representative examples of programs/operations that are ubiquitous in data science:
  - ❖ DB systems:
    - ❖ Non-deduplicating project
    - ❖ Simple SQL aggregates
    - ❖ SQL GROUP BY aggregates
  - ❖ ML systems:
    - ❖ Matrix sum/norms
    - ❖ Stochastic Gradient Descent

R	A	B	C	D
1a	1b	1c	1d	
2a	2b	2c	2d	
3a	3b	3c	3d	
4a	4b	4c	4d	
5a	5b	5c	5d	
6a	6b	6c	6d	

# Data-Parallel Non-dedup. Project

SELECT C FROM R

We focus on BSP data-parallel



**Basic Idea:** Manager splits work -> node-local work -> manager unifies results

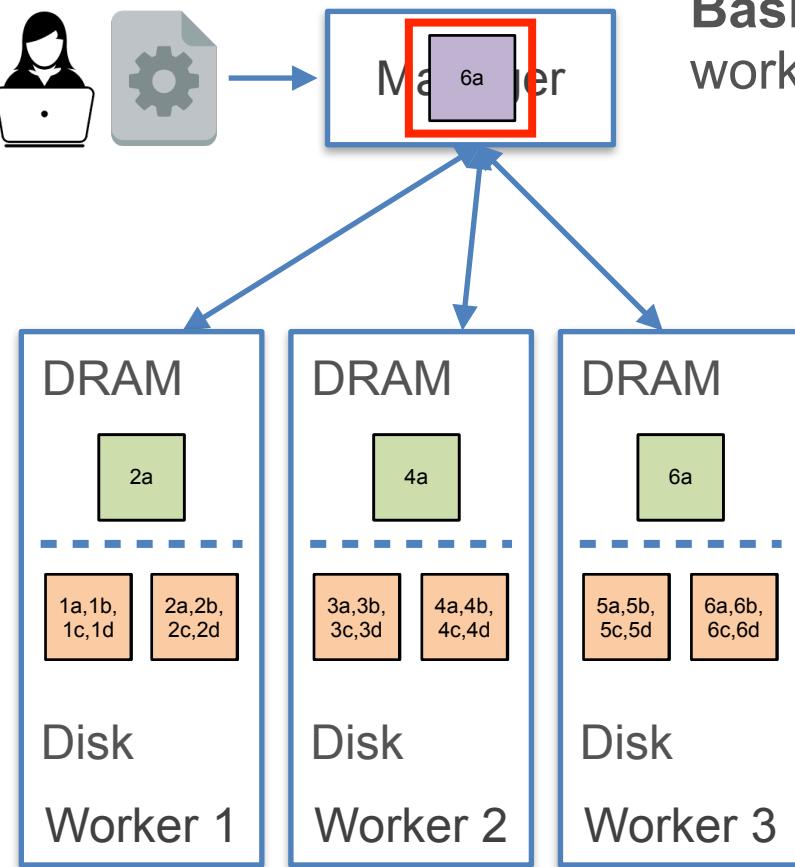
1. After ETL, sharded large input file sits cluster's disks
2. When query/program given, Manager broadcasts it as such
3. Each worker does node-local Non-dedup Project as explained before and writes local output to local file
4. Manager reports union of local files as global output file

I/O costs: Disk: 6 (pages) + output; Network: 0

# Data-Parallel Simple Aggregates

SELECT MAX(A) FROM R

We focus on BSP data-parallel



**Basic Idea:** Manager splits work -> node-local work -> manager unifies results

1. After ETL, sharded large input file sits cluster's disks
2. When query/program given, Manager broadcasts it as such
3. Each worker does node-local simple **partial aggregate** as explained before and *sends it to Manager* for unification
4. Manager unifies partial results based on op semantics

I/O costs: Disk: **6** (pages) + output; Network: **3** (#workers)

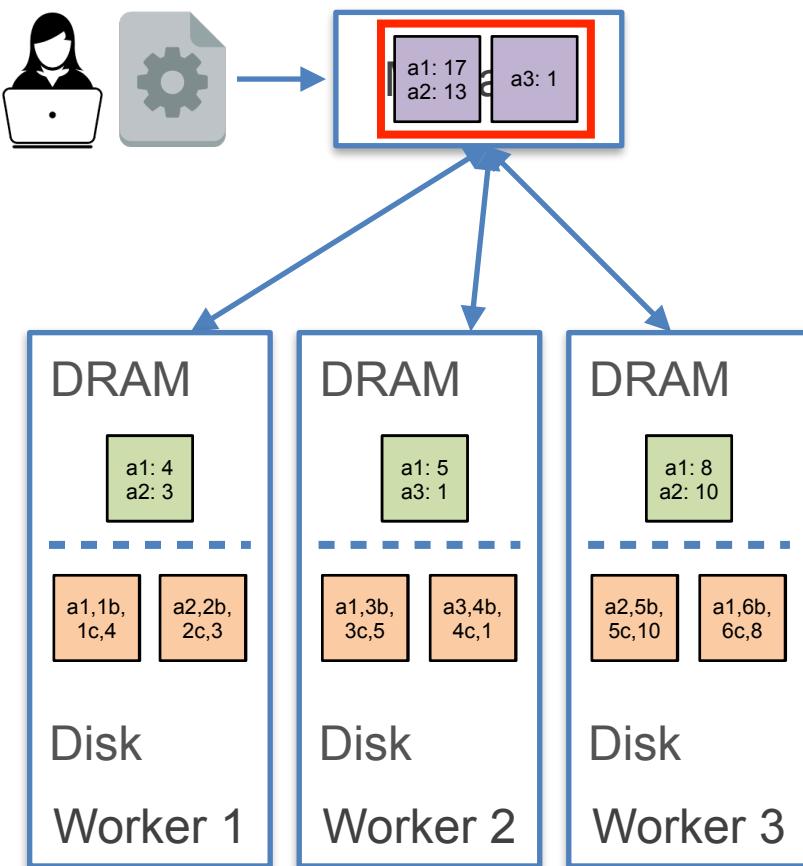
# Data-Parallel Simple Aggregates

*Q: Are all SQL aggregates easy to split up on sharded data?*

- ❖ Based on how easy it is to split up on shards, SQL aggs (aka descriptive stats) are categorized into 2/3 types:
- ❖ **Distributive Aggs:** A shard sends only 1 datum to manager
  - ❖ MIN, MAX, COUNT, SUM
- ❖ **Algebraic Aggs:** A shard sends  $O(1)$  size stats to manager
  - ❖ AVG (send SUM, COUNT separately); VARIANCE and STDEV (send SUM, SUM of squares, COUNT); etc.
- ❖ **Holistic Aggs:** Just  $O(1)$  size stats not enough in general; may need larger intermediate stats
  - ❖ MEDIAN, MODE, PERCENTILES, etc.

# Data-Parallel Group By Aggregate

SELECT A, SUM(D) FROM R GROUP BY A



R	A	B	C	D
a1	1b	1c	4	
a2	2b	2c	3	
a1	3b	3c	5	
a3	4b	4c	1	
a2	5b	5c	10	
a1	6b	6c	8	

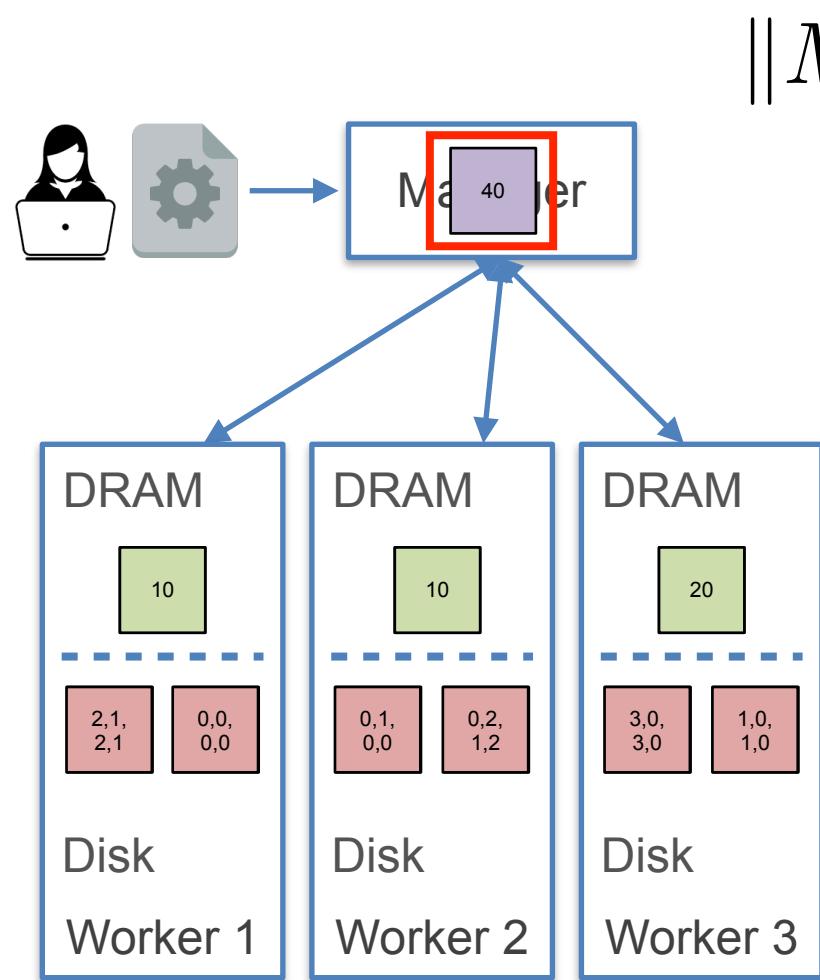
A	Running Info.
a1	17
a2	13
a3	1

Output

Similar to data-parallel simple agg  
Workers send **partial hash table** to manager based on local shards  
Manager collects and unifies local hash tables into global output  
Network I/O cost depends on data stats (domain size of A)

**Q:** What if Manager DRAM not enough to cache all hash tables?!

# Data-Parallel Matrix Sum/Norm



$$\|M\|_2^2$$

2	1	0	0
2	1	0	0
0	1	0	2
0	0	1	2
3	0	1	0
3	0	1	0

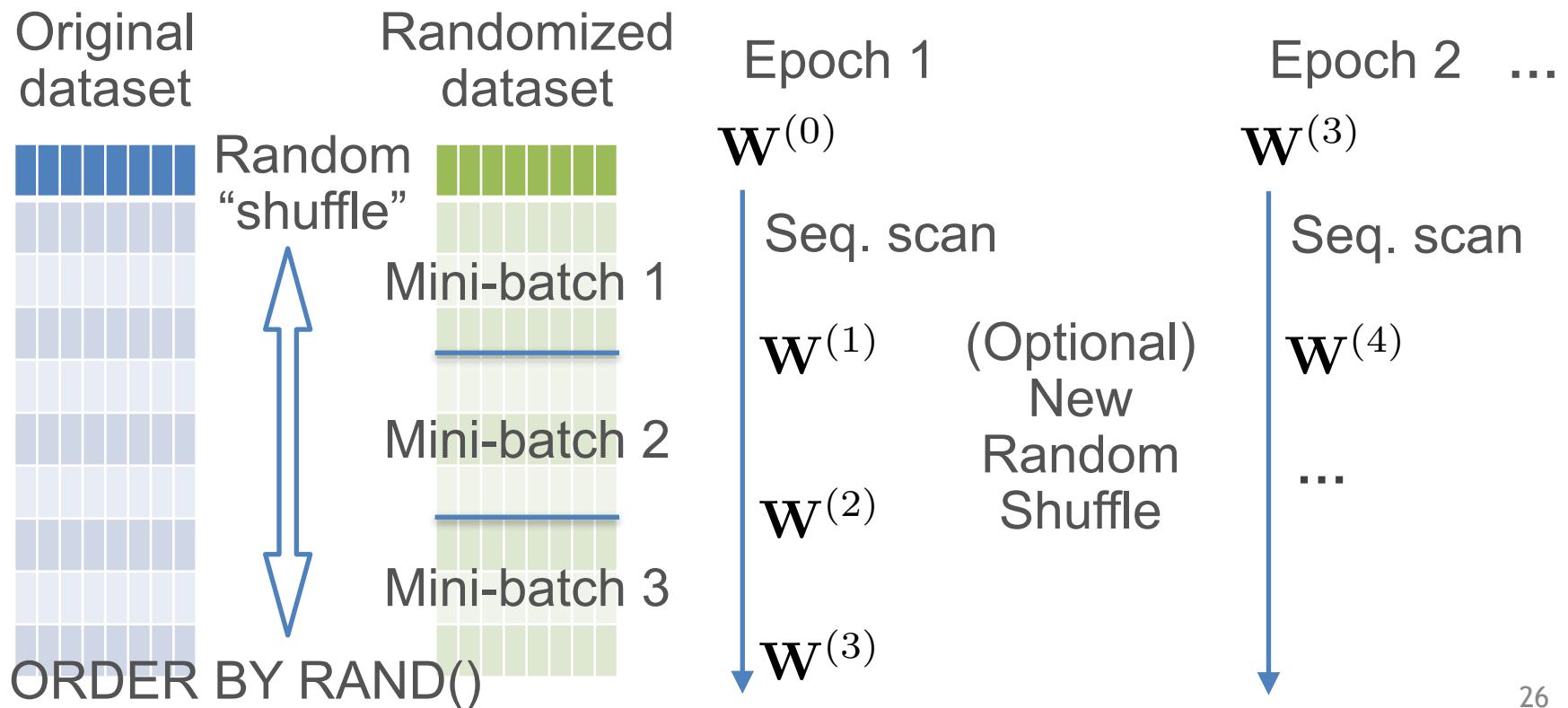
Say 2x2 tiled layout+partitioning

Similar to data-parallel simple agg  
Disk I/O cost: **6** (pages)  
Network I/O cost: **3** (#workers)

# Data Access Pattern of Scalable SGD

$$\mathbf{W}^{(t+1)} \leftarrow \mathbf{W}^{(t)} - \eta \nabla \tilde{L}(\mathbf{W}^{(t)}) \quad \nabla \tilde{L}(\mathbf{W}) = \sum_{i \in B} \nabla l(y_i, f(\mathbf{W}, x_i))$$

Sample mini-batch from dataset without replacement



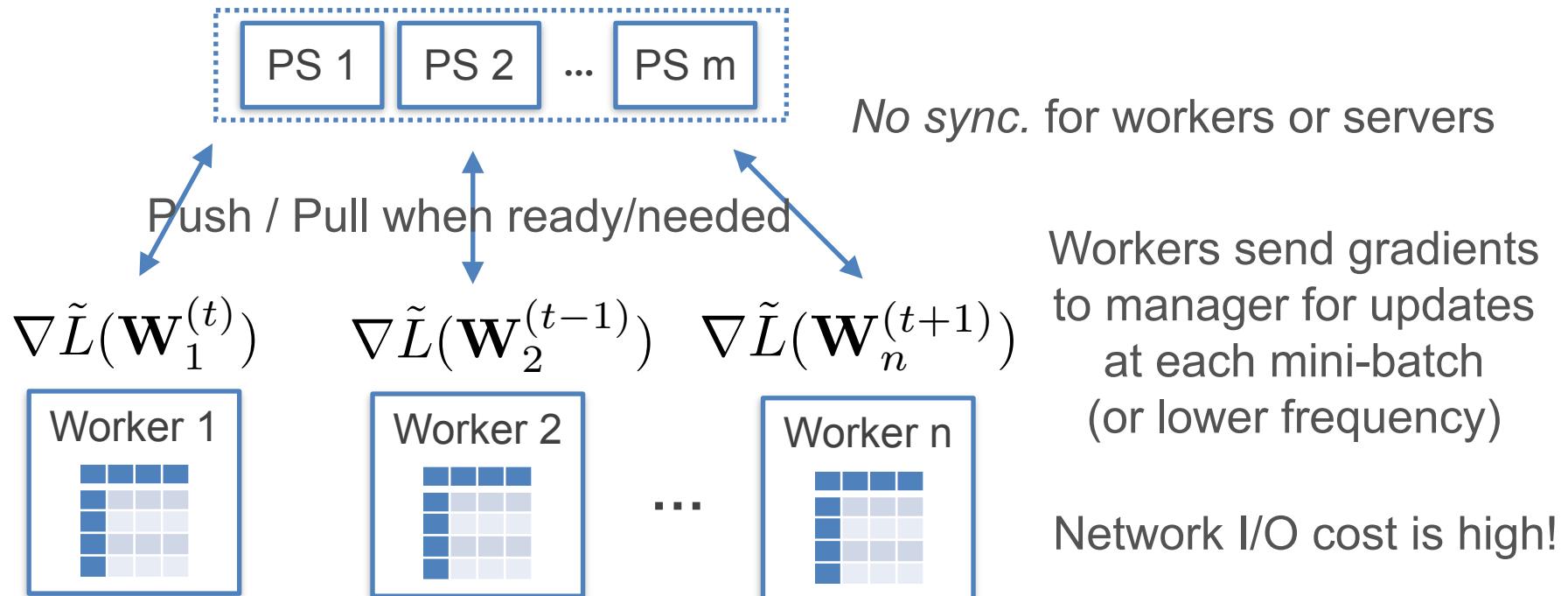
# Data Access Pattern of Scalable SGD

- ❖ An SGD epoch is similar to SQL aggs but also different:
  - ❖ More complex agg. state (running info): model param.  $\mathbf{W}^{(t)}$
  - ❖ Multiple mini-batch updates to model param. within a pass
  - ❖ Sequential dependency across mini-batches in a pass
  - ❖ Keep track of model param. across epochs
  - ❖ Not an *algebraic aggregate*; hard to parallelize!
  - ❖ Not *commutative*: different random shuffle orders give different results (very unlike relational ops)
  - ❖ (Optional) New random shuffling before each epoch

**Q:** How to execute SGD in a data-parallel manner?

# ParameterServer for Scalable SGD

Multi-server manager; each server manages a part of  $\mathbf{W}^{(t)}$



- ❖ Model params may get out-of-sync or stale; but SGD turns out to be robust; multiple updates/epoch helps



# Data-Parallel Data Science Ops

- ❖ Data parallelism for key representative examples of programs/operations that are ubiquitous in data science:
  - ❖ DB systems:
    - ❖ Non-deduplicating project
    - ❖ Simple SQL aggregates
    - ❖ SQL GROUP BY aggregates
  - ❖ ML systems:
    - ❖ Matrix sum/norms
    - ❖ Stochastic Gradient Descent

# Outline

- ❖ Basics of Parallelism
  - ❖ Task Parallelism; Dask
  - ❖ Single-Node Multi-Core; SIMD; Accelerators
- ❖ Basics of Scalable Data Access
  - ❖ Paged Access; I/O Costs; Layouts/Access Patterns
  - ❖ Scaling Data Science Operations
- ❖ Data Parallelism: Parallelism + Scalability
  - ❖ Data-Parallel Data Science Operations
- ➡❖ Optimizations and Hybrid Parallelism

# Execution Optimization Tradeoffs

- ❖ Some common optimizations in data-parallel systems:
  - ❖ **Replication:** Put a shard on >1 worker; more parallelism possible for execution
  - ❖ **Caching:** Store as much data as possible on worker DRAM and/or disk
  - ❖ **Asynchrony:** Less common in DB systems; more common in ML systems (e.g., ParameterServer)
  - ❖ **Approximation:** Carefully exploit data subsampling
- ❖ Using ML for data placement, caching, tiered storage across memory hierarchy is now a hot topic in “ML for systems” world

# Hybrid Parallelism

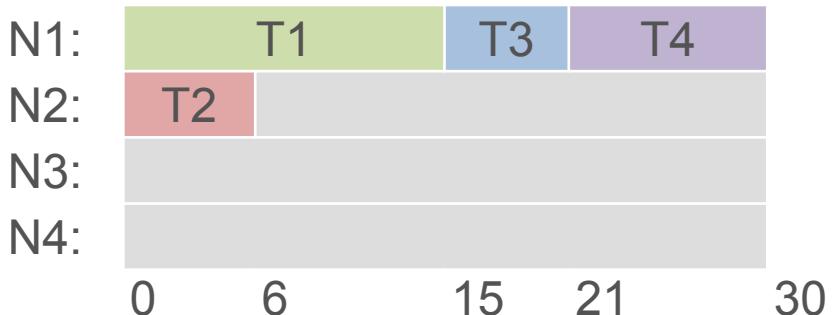
- ❖ Task- vs Data-Parallelism have pros and cons:
  - ❖ Task-par. wastes memory/storage due to replication; remote reads waste network; but easy to implement
  - ❖ Data-par. is painful to implement at op level; but scales w/o wasting memory/storage; more network costs

*Q: Is it possible to get the best of both these worlds?*

- ❖ Yes, often we can run task-par. on sharded data!
- ❖ Examples: Different SQL queries or different ML training routines run on top of same sharded data setup
  - ❖ Aka “**Multi-Query Execution**” in the DB world

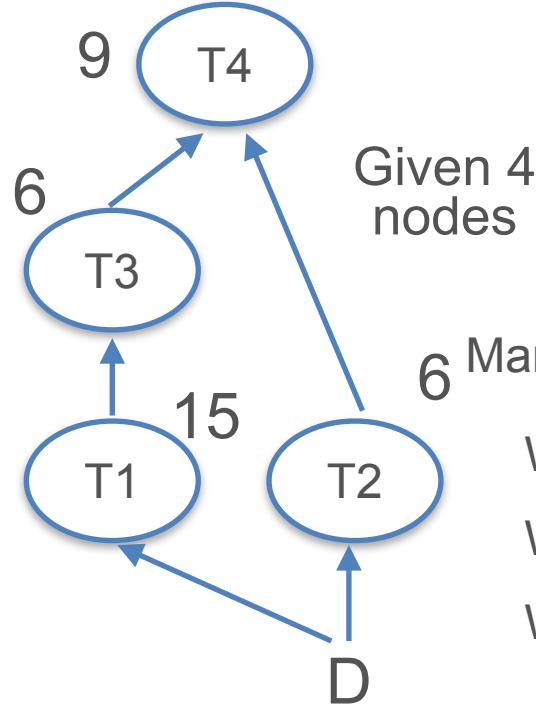
# Task Par. vs BSP Data Par.

Fully task-par schedule:



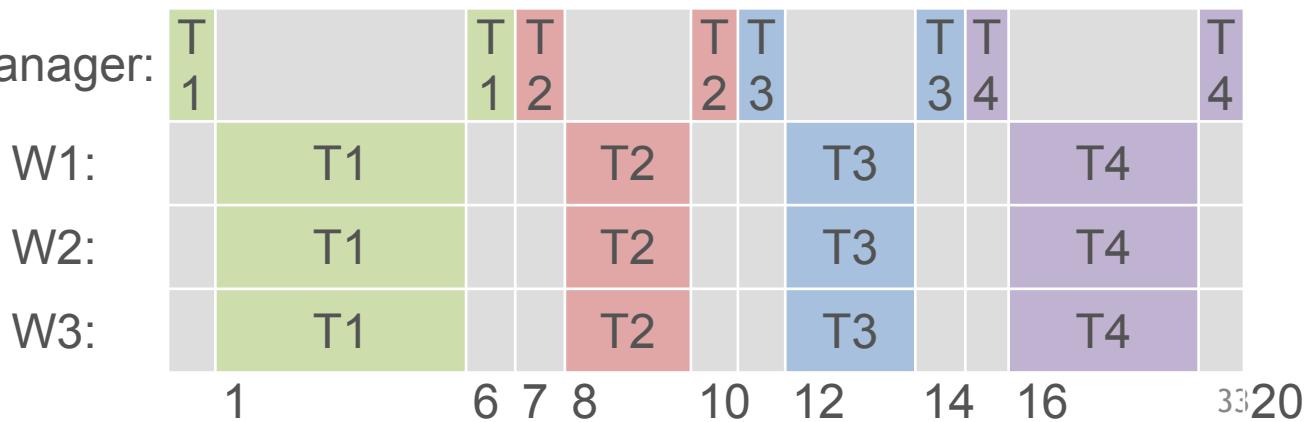
N3, N4 are both useless. Why?  
N2 has idle times too. Why?

Example:



Suppose each task gets perfect linear speedup on its useful work on BSP; manager overhead is, say, 1 unit each before/ after

Fully BSP data-par schedule:

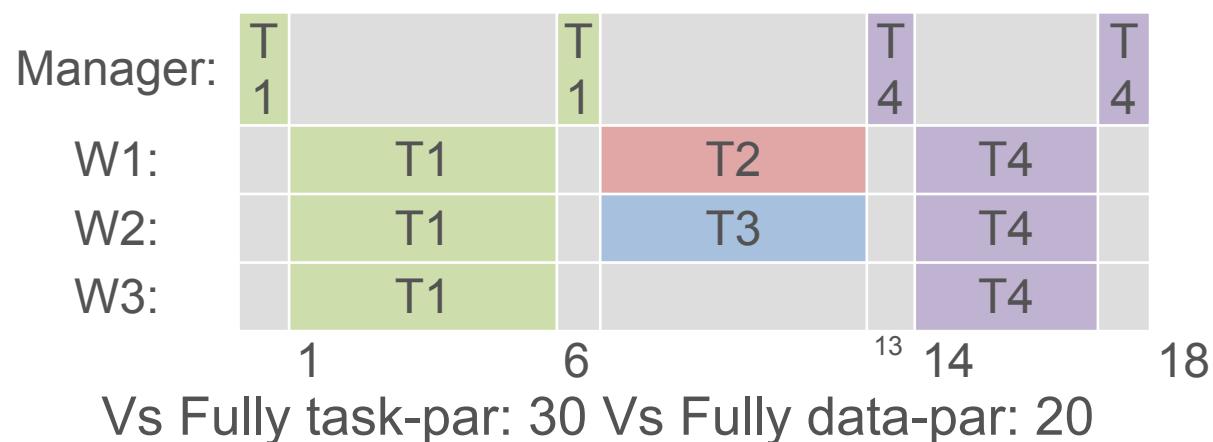
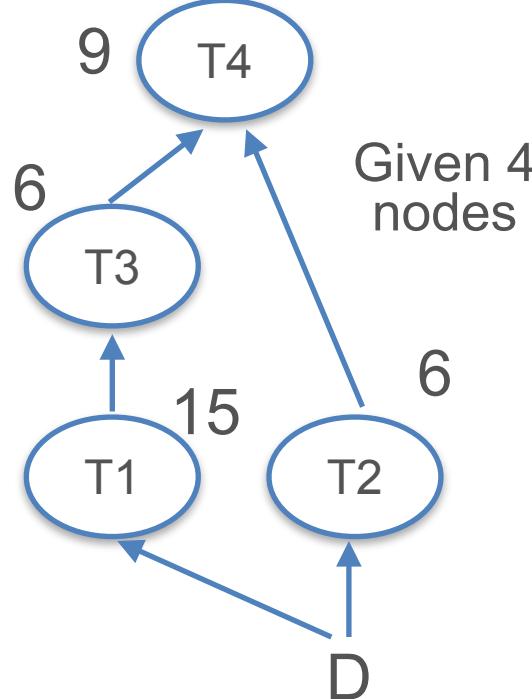


# Hybrid of Task and Data Parallelism

*Q: Can we go faster if we hybridize task and data par?*

One possible hybrid schedule:

**Example:**

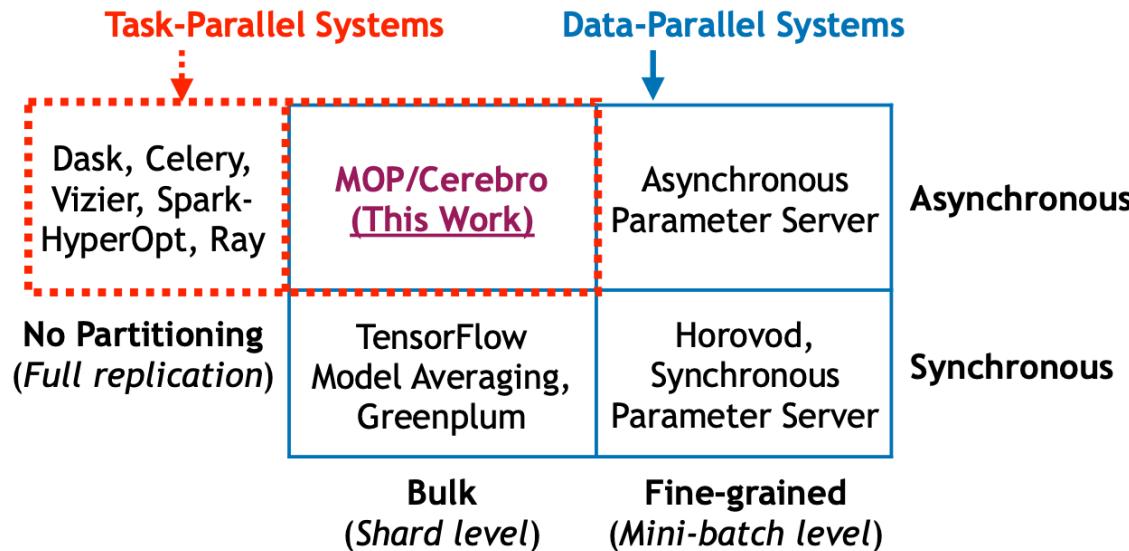


- ❖ Most scalable data systems today support only full task-par. (e.g., Dask) or full data-par. (e.g., RDBMS); hybrid software complexity is high
- ❖ Some RDBMSs do internally exploit hybrid-par. for relational dataflows
- ❖ Spark is beginning to support task-par. too

# Hybrid of Task and Data Parallelism

**Q:** *Can we go faster if we hybridize task and data par?*

- ❖ A key recent example from research: **Cerebro** for parallel DL model selection on clusters



- ❖ First known form of “Bulk Asynchronous Parallelism”
- ❖ Resource-optimal when compute, memory/storage, and network considered holistically

<https://adalabucsd.github.io/cerebro.html> (Start with the CIDR'21 paper and talk video)

**Ad:** Take CSE 234 for more on Cerebro, model selection systems

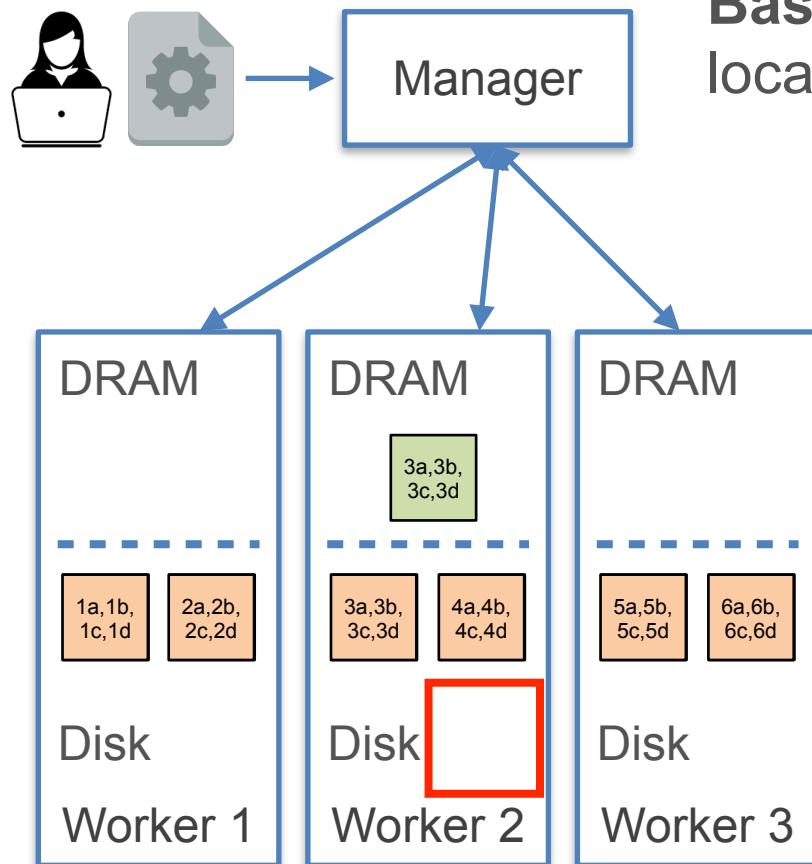
# Review Discussion

1. To which multi-node parallelism paradigm (Shared Nothing/Memory/Disk) does data parallelism apply?
2. What are the two most common types of cluster communication protocols in parallel data systems?
3. Is it possible to pair up columnar partitioning with row store? Vice versa?
4. What exactly is the “synchrony” in BSP?
5. Name 2 common sources of overhead in data-parallel systems that can lead to sub-linear speedups.
6. Name 2 SQL aggregates that are NOT algebraic.
7. Why is SGD not amenable to parallelization like algebraic aggregates?
8. Why does Parameter Server have high communication costs when executing data-parallel SGD on a cluster?
9. Briefly explain 2 systems-level optimizations in data-parallel systems and how they can benefit data science workloads.
10. Name 1 pro and 1 con of BSP over task parallelism. Why do most parallel data systems today employ only one or the other?

Optional: More Examples of Data-Parallel  
Data Science Operations  
Not included in syllabus

# Data-Parallel Relational Select

$$\sigma_B = \text{"3b"}(R)$$



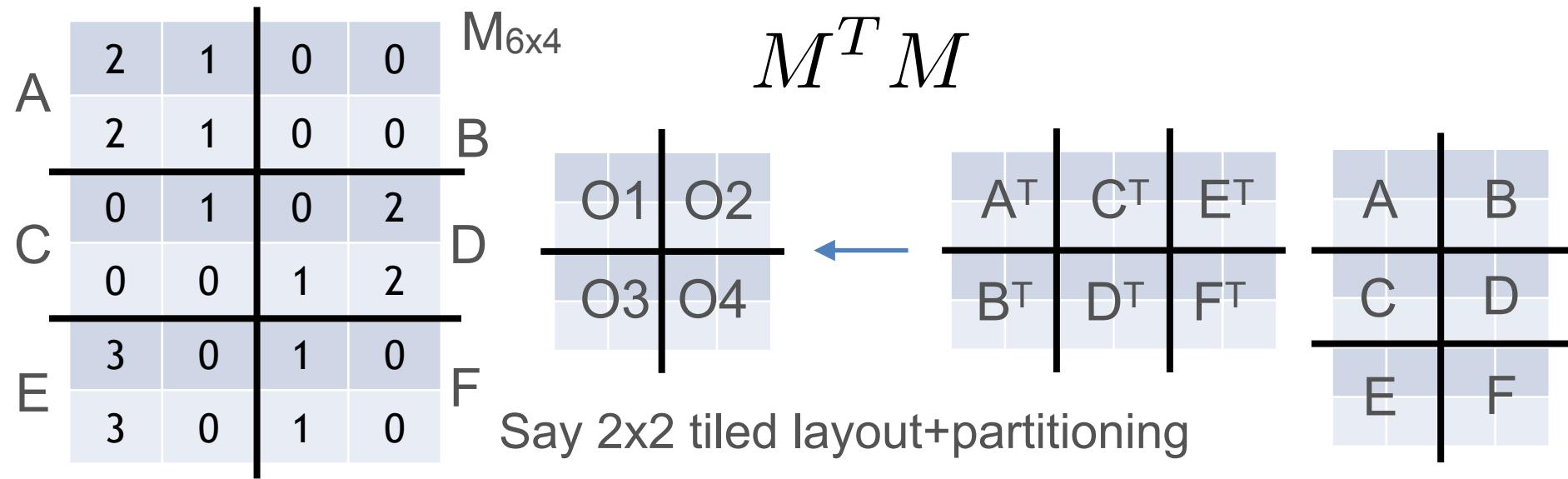
We focus on BSP data-parallel

**Basic Idea:** Manager splits work -> node-local work -> manager unifies results

1. After ETL, sharded large input file sits cluster's disks
2. When query/program given, manager broadcasts it as such
3. Each worker does node-local Select as explained before and writes local output to local file
4. Manager reports union of local files as global output file; note that output is also sharded file!

I/O costs: Disk: 6 (pages) + output; Network: 0

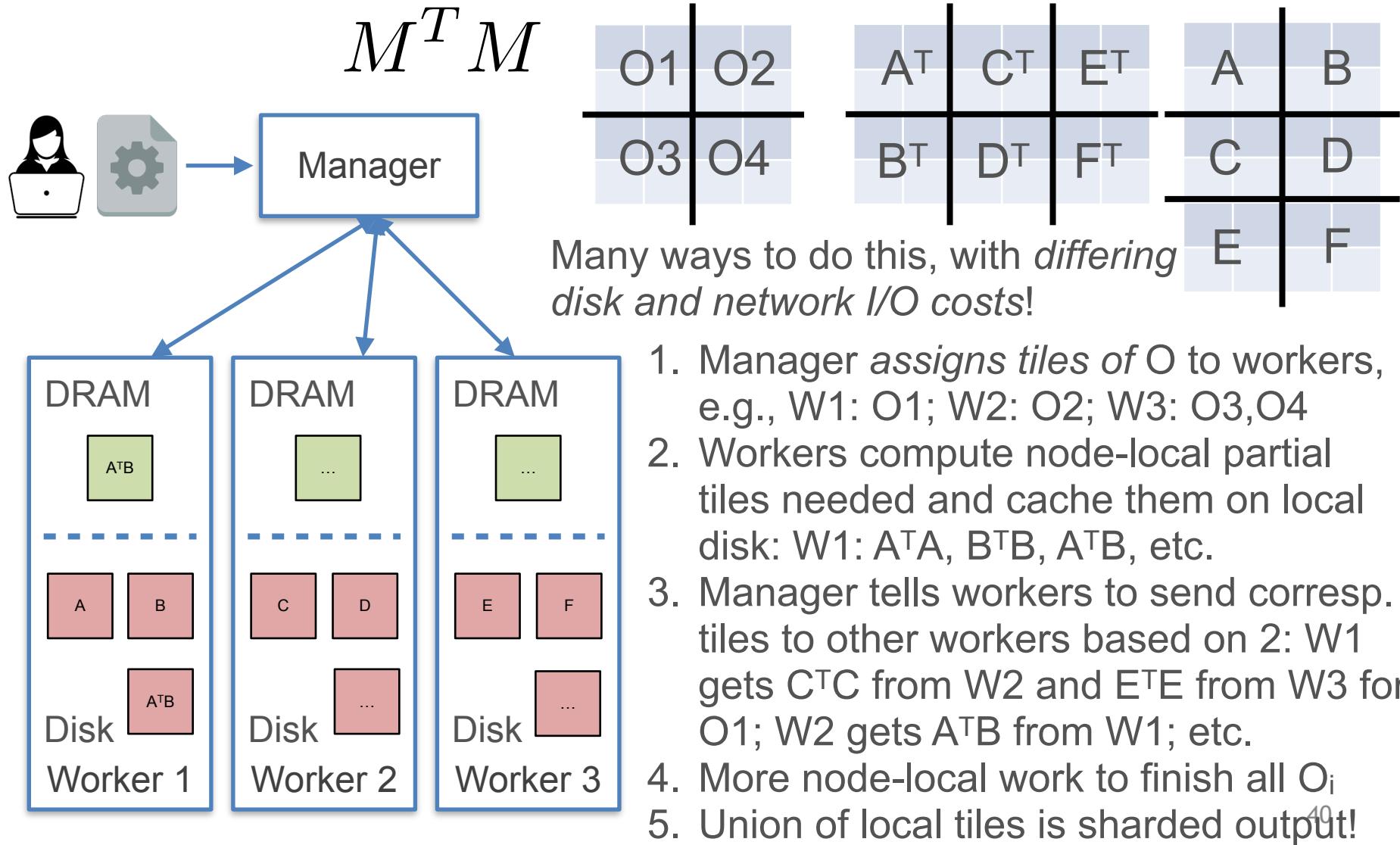
# Data-Parallel Gramian Matrix



More complex in the data-parallel setting, since we may need to *communicate data shards across workers!*

**Basic Idea:** Manager splits work -> node-local work -> *manager commands workers to talk to others as needed* -> more node-local work -> manager unifies results

# Data-Parallel Gramian Matrix



# Data-Parallel Gramian Matrix

- ❖ Not straightforward to determine I/O costs (both disk I/O and network I/O) of matrix mult., even simple Gramian!
  - ❖ CPU costs can also differ based on whether workers repeat redundant work vs cache it to file
  - ❖ Runtime is a complex function combining disk I/O cost, network I/O cost, and CPU/compute cost
- ❖ Different **operator implementations** exist in the parallel data systems literature: crossproduct-based multiply, replication-based multiply, etc.

# Reproducible ML At Scale

Lavanya Shukla, Head of Growth

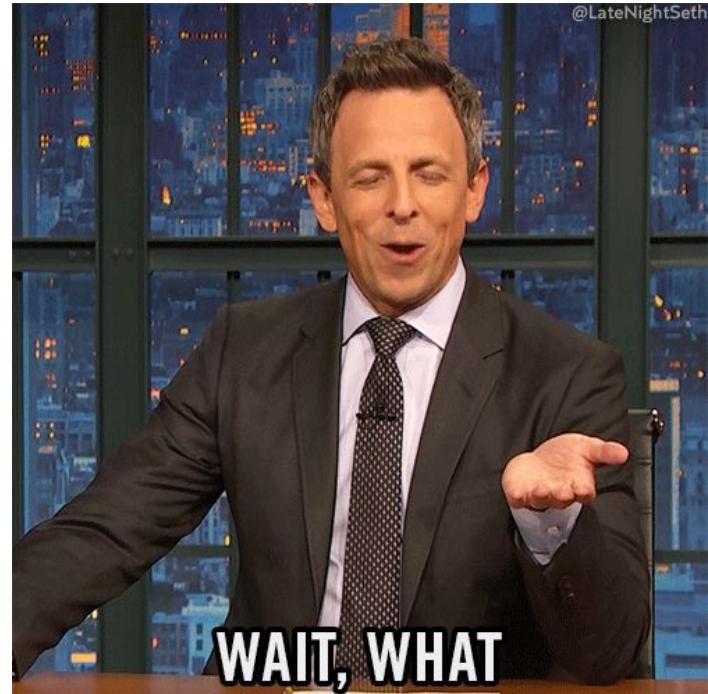


# Agenda

- 1. Why do we need ML reproducibility?**
- 2. Model understanding through reproducibility**
- 3. Interactive model visualization**

Why do we need ML  
reproducibility?

**Training the same model,  
with the same hyperparameters  
... doesn't always lead to the same results**



**Training the same model,  
with the same hyperparameters  
... doesn't always lead to the same results**

- Datasets – order of training examples
- Differences in hardware, GPUs
- Differences in ML library versions
- Initialization of weights
- Model architecture randomness – dropout

# **The more our ML models impact the real world, the more we need them to be reproducible.**

- Who gets a loan?
- Who gets indicted?
- How long is someone's sentence?
- Who's resume is good enough to be given an interview?
- Who is trustworthy? Responsible?

# ML in regulated industries – An example



# Reproducibility gap

- **Lack of access** to the same training data / **differences in data distribution**;
- Lack of availability of the code necessary to run the experiments, or **errors in the code**;
- **Under-specification of the metrics** used to report results;
- **Improper use of statistics to analyze results**, such as claiming significance without
- **Selective reporting of results** and ignoring the danger of adaptive overfitting;
- **Over-claiming of the results**, by drawing conclusions that go beyond the evidence

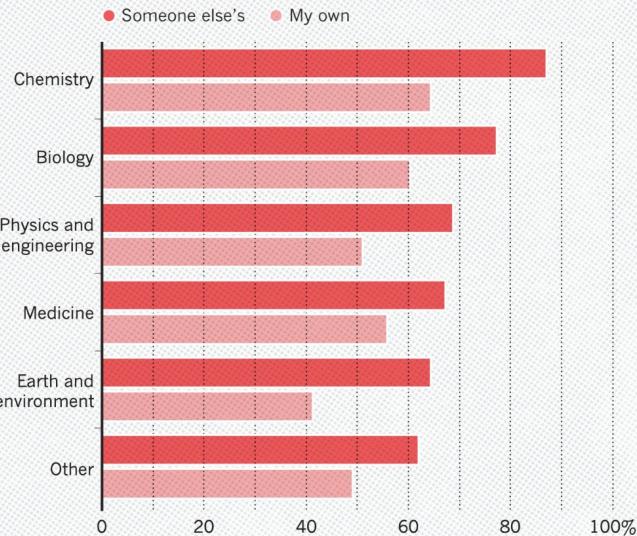
# **Is there a reproducibility crisis?**

**Have tried and failed to reproduce an experiment?**

# How many of you have tried and failed to reproduce an experiment?

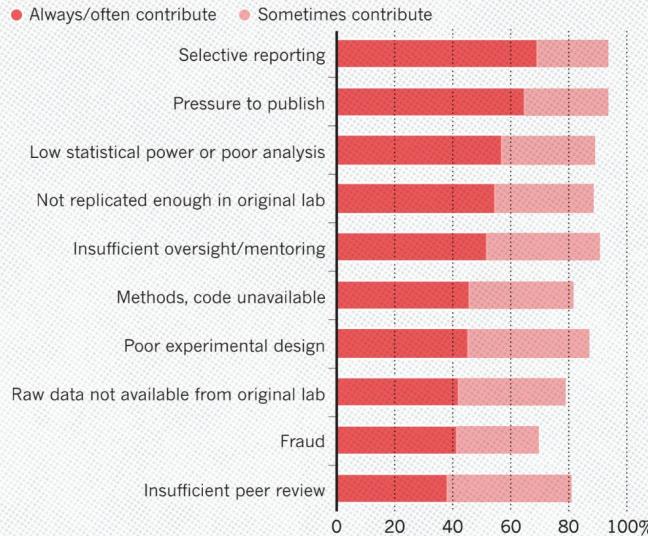
## HAVE YOU FAILED TO REPRODUCE AN EXPERIMENT?

Most scientists have experienced failure to reproduce results.



## WHAT FACTORS CONTRIBUTE TO IRREPRODUCIBLE RESEARCH?

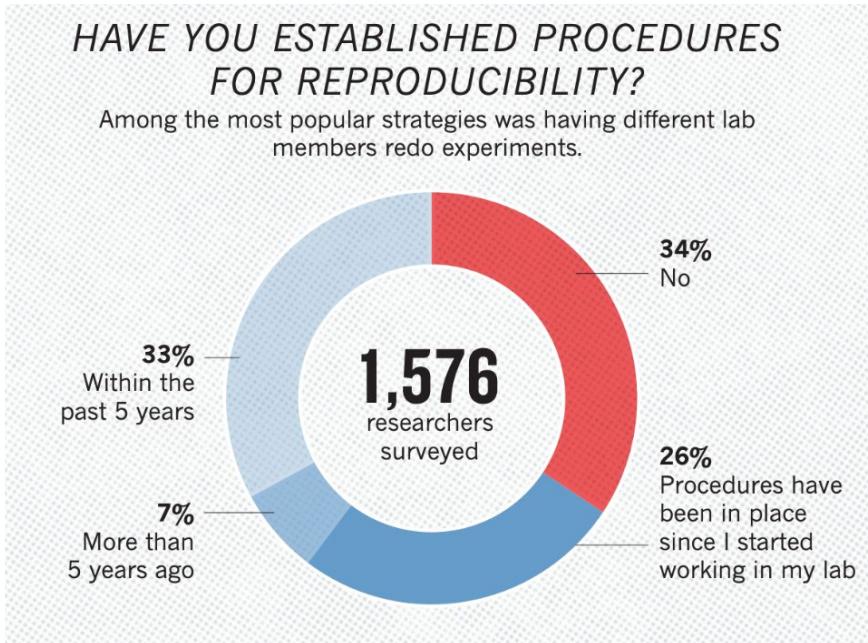
Many top-rated factors relate to intense competition and time pressure.



Source: [Nature](#)

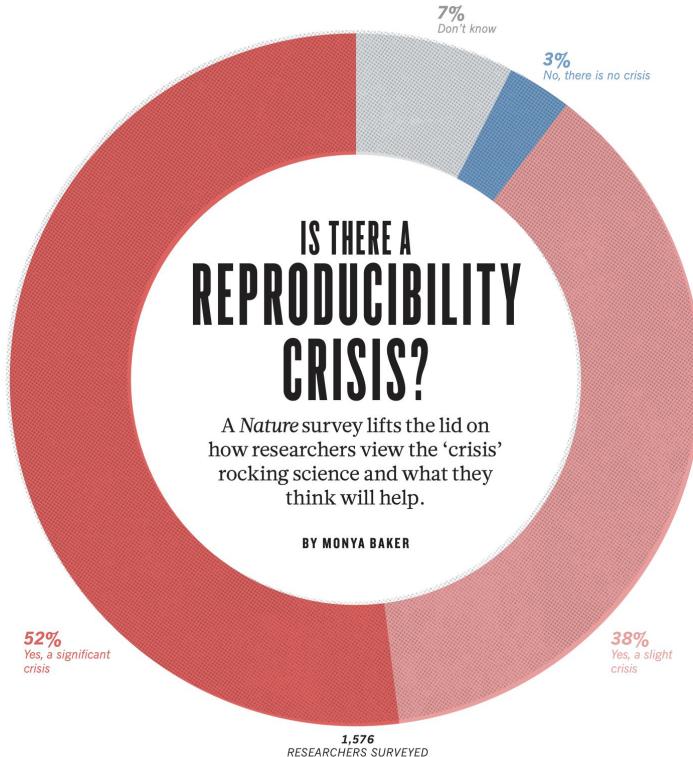
# **Do you have a process for making your models reproducible?**

# Do you have a process for making your models reproducible?



Source: [Nature](#)

# Is there a reproducibility crisis?



Source: [Nature](#)

**“REPRODUCIBILITY  
IS LIKE BRUSHING  
YOUR TEETH. ONCE  
YOU LEARN IT, IT  
BECOMES A HABIT.”**

What does it mean to be  
reproducible?

For all models and algorithms presented, check if you include:

- A clear description of the mathematical setting, algorithm, and/or model.
- A clear explanation of any assumptions.
- An analysis of the complexity (time, space, sample size) of any algorithm.

For any theoretical claim, check if you include:

- A clear statement of the claim.
- A complete proof of the claim.

For all datasets used, check if you include:

- The relevant statistics, such as number of examples.
- The details of train / validation / test splits.
- An explanation of any data that were excluded, and all pre-processing steps.
- A link to a downloadable version of the dataset or simulation environment.
- For new data collected, a complete description of the data collection process, such as instructions to annotators and methods for quality control.

For all shared code related to this work, check if you include:

- Specification of dependencies.
- Training code.
- Evaluation code.
- (Pre-)trained model(s).
- README file includes table of results accompanied by precise command to run to produce those results.

For all reported experimental results, check if you include:

- The range of hyper-parameters considered, method to select the best hyper-parameter configuration, and specification of all hyper-parameters used to generate results.
- The exact number of training and evaluation runs.
- A clear definition of the specific measure or statistics used to report results.
- A description of results with central tendency (e.g. mean) & variation (e.g. error bars).
- The average runtime for each result, or estimated energy cost.
- A description of the computing infrastructure used.

# Model understanding through reproducibility



## PROBLEM

# Massive developer tools gap for ML

### SOFTWARE 1.0 - WRITE CODE

Design	Version Code	Code & Collaborate	Deploy to Infra	CI/CD	Production Monitoring
Figma	GitHub	Jira	HashiCorp	circleci	PagerDuty
Sketch	GitLab	VS Code	puppet	Jenkins	DATADOG

### SOFTWARE 2.0 - TRAIN MODELS

Prep & Visualize Data	Version Data & Models	Experiment Tracking	Manage Model Pipeline	Model CI/CD	Production monitoring
Custom apps Notebooks	Files in S3	Text files Screenshots	Text files	Custom scripts	Nothing

# BCP + W&B INTEGRATED SOLUTION

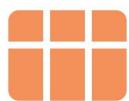
ML Ops Platform with Interoperable Applications

Version data &  
pipelines



Artifacts

Prep &  
visualize data



Tables

Track model  
development



Experiments

Optimize  
models



Sweeps

Collaborative  
analysis



Reports

Production  
monitoring



Coming Soon

ML LIBRARIES

Hugging Face

SpaCy

Detectron

Yolo V5

...

FRAMEWORKS

PyTorch

Keras

TensorFlow

XGBoost

...

PARTNERSHIP

 **NVIDIA.** Base Command Platform

Making ML workflows reproducible, one step at a time

# Before W&B

Layer (type)	Output Shape	Param #	Connected to
<hr/>			
conv2d_1_input (InputLayer)	(None, 299, 299, 3)	0	
lambda_1 (Lambda)	(None, 299, 299, 3)	0	conv2d_1_input[0][0]
lambda_2 (Lambda)	(None, 299, 299, 3)	0	conv2d_1_input[0][0]
sequential_1 (Sequential)	(None, 10)	910922	lambda_1[0][0] lambda_2[0][0]
activation_7 (Concatenate)	(None, 10)	0	sequential_1[1][0] sequential_1[2][0]
<hr/>			
Total params:	910,922		
Trainable params:	910,922		
Non-trainable params:	0		
<hr/>			
None			
Found 49999 images belonging to 10 classes.			
Found 8000 images belonging to 10 classes.			
Epoch 1/50			
39/39 [=====] - 270s 7s/step - loss: 2.3153 - acc: 0.1178 - val_loss: 2.2428 - val_acc: 0.1589			
Epoch 2/50			
39/39 [=====] - 263s 7s/step - loss: 2.2588 - acc: 0.1538 - val_loss: 2.1890 - val_acc: 0.2174			
Epoch 3/50			
39/39 [=====] - 262s 7s/step - loss: 2.2060 - acc: 0.1827 - val_loss: 2.1767 - val_acc: 0.2096			
Epoch 4/50			
39/39 [=====] - 263s 7s/step - loss: 2.1640 - acc: 0.2107 - val_loss: 2.1133 - val_acc: 0.2357			
Epoch 5/50			
39/39 [=====] - 261s 7s/step - loss: 2.0827 - acc: 0.2432 - val_loss: 2.1499 - val_acc: 0.2344			
Epoch 6/50			
39/39 [=====] - 262s 7s/step - loss: 2.0810 - acc: 0.2508 - val_loss: 2.0289 - val_acc: 0.2604			
Epoch 7/50			
39/39 [=====] - 262s 7s/step - loss: 2.0575 - acc: 0.2602 - val_loss: 1.9912 - val_acc: 0.2865			
Epoch 8/50			
39/39 [=====] - 261s 7s/step - loss: 2.0355 - acc: 0.2734 - val_loss: 2.0319 - val_acc: 0.2409			
Epoch 9/50			
39/39 [=====] - 263s 7s/step - loss: 2.0254 - acc: 0.2829 - val_loss: 1.9364 - val_acc: 0.3307			
Epoch 10/50			
39/39 [=====] - 262s 7s/step - loss: 2.0056 - acc: 0.2804 - val_loss: 1.9934 - val_acc: 0.2773			
Epoch 11/50			
39/39 [=====] - 256s 7s/step - loss: 1.9678 - acc: 0.3048 - val_loss: 1.9048 - val_acc: 0.3352			
Epoch 12/50			
39/39 [=====] - 260s 7s/step - loss: 1.9634 - acc: 0.3109 - val_loss: 1.8758 - val_acc: 0.3568			

Table 3: Detection results on PASCAL VOC 2007 test set. The detector is Fast R-CNN and VGG-16. Training data: “07”: VOC 2007 trainval, “07+12”: union set of VOC 2007 trainval and VOC 2012 trainval. For RPN, the train-time proposals for Fast R-CNN are 2000. <sup>†</sup>: this number was reported in [2]; using the repository provided by this paper, this result is higher (68.1).

method	# proposals	data	mAP (%)
SS	2000	07	66.9 <sup>†</sup>
SS	2000	07+12	70.0
RPN+VGG, unshared	300	07	68.5
RPN+VGG, shared	300	07	69.9
RPN+VGG, shared	300	07+12	73.2
RPN+VGG, shared	300	COCO+07+12	78.8

Table 4: Detection results on PASCAL VOC 2012 test set. The detector is Fast R-CNN and VGG-16. Training data: “07”: VOC 2007 trainval, “07+12”: union set of VOC 2007 trainval+test and VOC 2012 trainval. For RPN, the train-time proposals for Fast R-CNN are 2000. <sup>†</sup>: http://host.robots.ox.ac.uk:8080/anonymous/YNPLXB.html; <sup>‡</sup>: http://host.robots.ox.ac.uk:8080/anonymous/HZJ1QA.html; <sup>§</sup>: http://host.robots.ox.ac.uk:8080/anonymous/XEDH10.html.

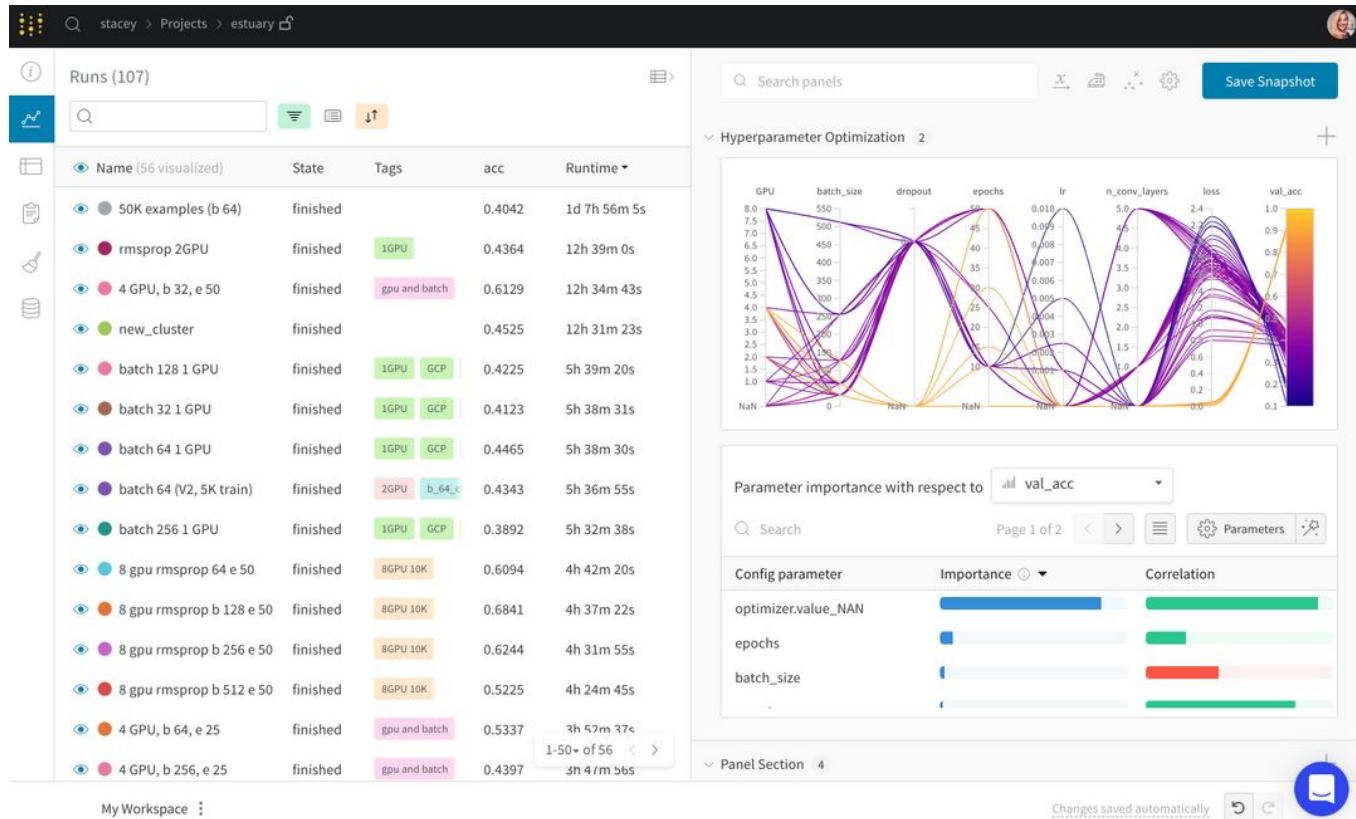
method	# proposals	data	mAP (%)
SS	2000	12	65.7
SS	2000	07+12	68.4
RPN+VGG, shared <sup>†</sup>	300	12	67.0
RPN+VGG, shared <sup>‡</sup>	300	07+12	70.4
RPN+VGG, shared <sup>§</sup>	300	COCO+07+12	75.9

Table 5: Timing (ms) on a K40 GPU, except SS proposal is evaluated in a CPU. “Region-wise” includes NMS, pooling, fully-connected, and softmax layers. See our released code for the profiling of running time.

model	system	conv	proposal	region-wise	total	rate
VGG	SS + Fast R-CNN	146	1510	174	1830	0.5 fps
VGG	RPN + Fast R-CNN	141	10	47	198	5 fps
ZF	RPN + Fast R-CNN	31	3	25	59	17 fps

	backbone	AP	AP <sub>50</sub>	AP <sub>75</sub>	AP <sub>S</sub>	AP <sub>M</sub>	AP <sub>L</sub>
<i>Two-stage methods</i>							
Faster R-CNN++ [5]	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN [8]	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI [6]	Inception-ResNet-v2 [21]	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM [20]	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	<b>52.1</b>
<i>One-stage methods</i>							
YOLOv2 [15]	DarkNet-19 [15]	21.6	44.0	19.2	5.0	22.4	35.5
SSD513 [11, 3]	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513 [3]	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet [9]	ResNet-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
RetinaNet [9]	ResNeXt-101-FPN	<b>40.8</b>	<b>61.1</b>	<b>44.1</b>	<b>24.1</b>	<b>44.2</b>	51.2
YOLOv3 608 × 608	Darknet-53	33.0	57.9	34.4	18.3	35.4	41.9

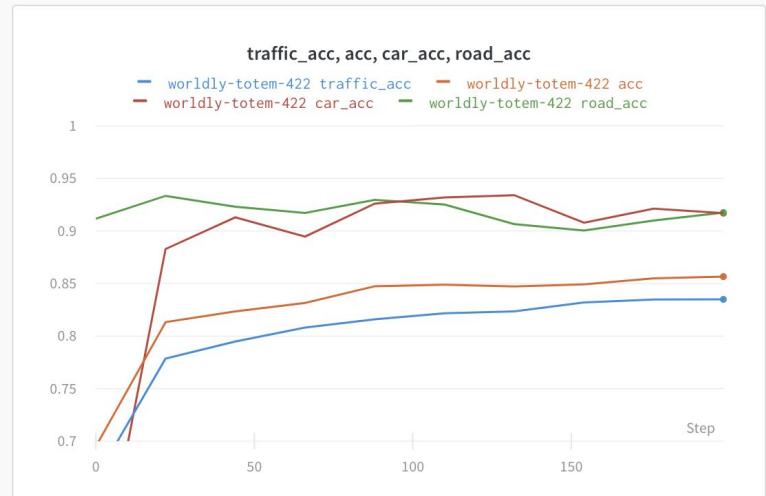
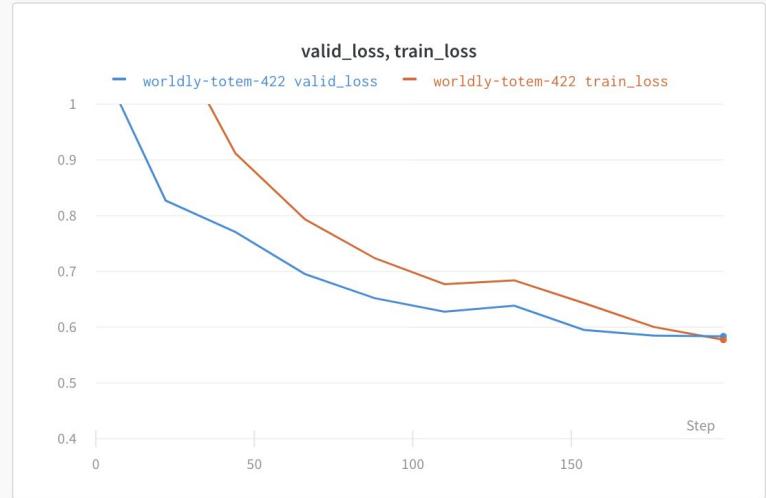
# Track experiments – After W&B



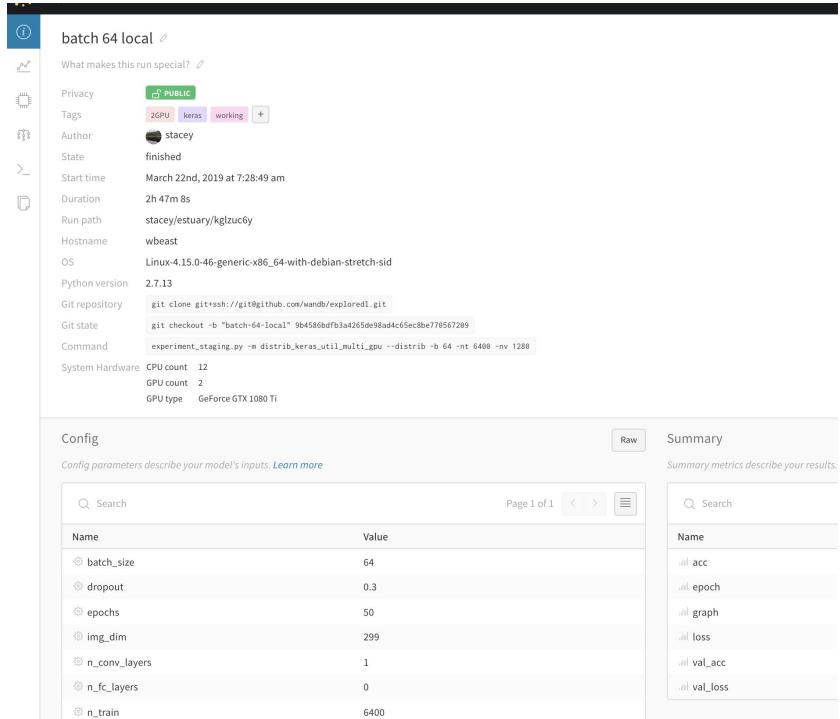
[bit.ly/demo-run](https://bit.ly/demo-run)

# Reproduce a model

See live updates on model performance, check for overfitting, and visualize how a model performs on different classes.



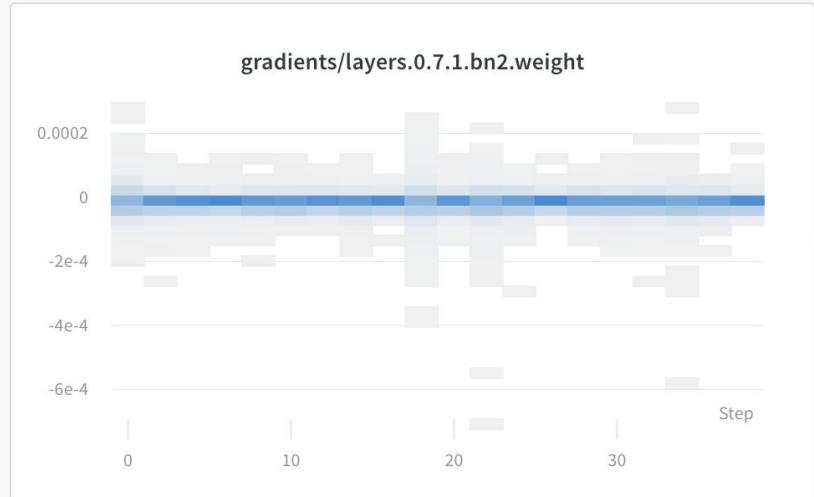
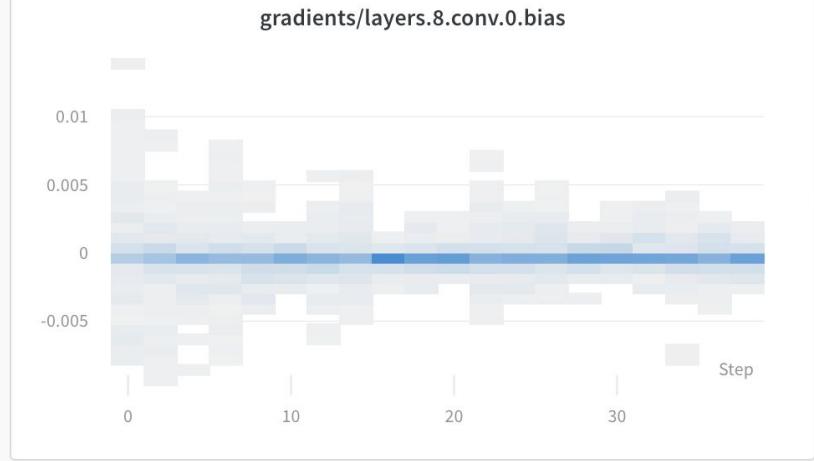
# Track experiments – After W&B



[bit.ly/demo-run](https://bit.ly/demo-run)

# Visualize gradients

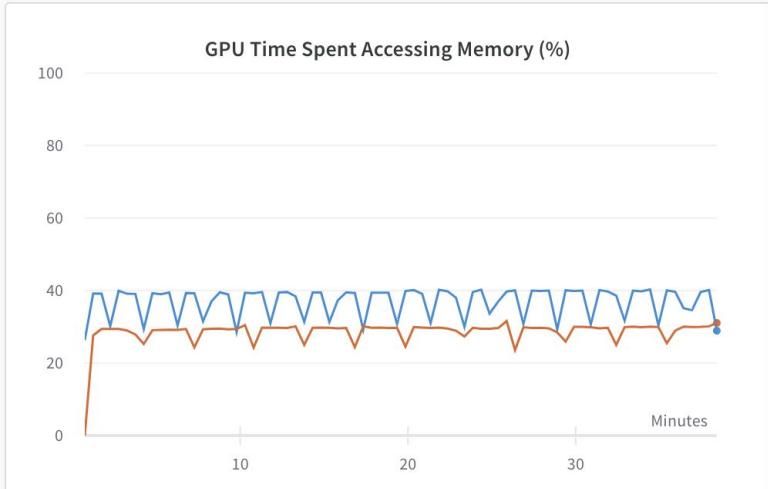
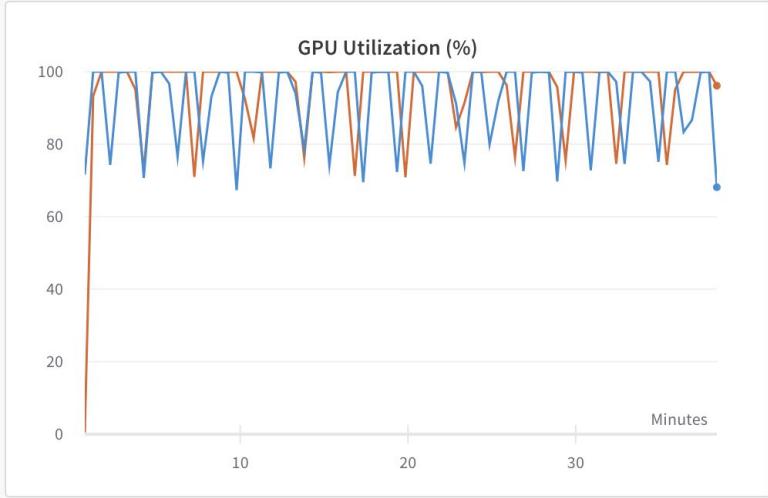
See the inner workings of your model, look for convergence during training, and check for exploding gradients.



[bit.ly/demo-run](https://bit.ly/demo-run)

# System metrics

Get the most out of your GPUs  
and identify opportunities for  
optimizing hardware utilization.



[bit.ly/demo-run](https://bit.ly/demo-run)

# Capture the code

Save the most recent git commit,  
the command and args, and  
system hardware setup.

W&B also saves a patch file with  
uncommitted changes so you  
can reproduce the exact code  
that trained the model.

dutiful-dragon-174 

What makes this run special? 

Privacy	 PUBLIC
Tags	 +
Author	stacey
State	finished
Start time	January 24th, 2020 at 1:30:16 pm
Duration	38m 27s
Run path	stacey/deep-drive/6zsn8ltb
Hostname	wbrave
OS	Linux-5.0.0-37-generic-x86_64-with-debian-buster-sid
Python version	3.7.2
Python executable	/home/stacey/.pyenv/versions/sd/bin/python
Git repository	<code>git clone https://github.com/borisdayma/semantic-segmentation.git</code>
Git state	<code>git checkout -b "dutiful-dragon-174" f0a9494a5d152663d226e28e279c65</code>
Command	<code>train.py</code>
CPU count	20
System Hardware	GPU count 2
	GPU type GeForce RTX 2080 Ti
W&B CLI Version	0.8.21



# Code tab with versioning – spot changes

Find matching artifacts

Overview API Metadata Files **Files** Graph view

> root / training / env\_factory.py File Diff

CODE

▼ safelife\_training

v17 latest

v16  
v15  
v14  
v13  
v12  
v11  
v10  
v9  
v8  
v7  
v6  
v5

v4  
v3  
v2  
v1  
v0

► safelife\_core

EPISODE\_DATA

► episode\_data

VIDEO

► videos\_append-spa...  
► video

Expand 163 lines ...

```
164     Probability of picking level 2 instead of level 1.  
165     """  
166     def __init__(self, level1, level2, p_switch, **kwargs):  
167 -         super().__init__(level1, level2, **kwargs)  
168         self.p_switch = p_switch  
169  
170     def get_next_parameters(self):  
171         """  
172         Expand 38 lines ...  
173         """  
174         'navigate': {  
175             'iter_class': SafeLifeLevelIterator,  
176             # The navigation levels take a *long* time to generate, so  
177             # use a fixed set of 10k levels instead.  
178             'train_levels': ['training/navigation'],  
179             'validation_levels': ['random/navigation'],  
180             'benchmark_levels': 'benchmarks/v1.0/navigation.npz',  
181         },  
182         # Multi-agent tasks:  
183         """  
184         Expand 116 lines ...  
185         """  
186         iter_class = task_data.get('iter_class', SafeLifeLevelIterator)  
187         iter_args = {'seed': training_seed, 'repeat_levels': True}  
188  
189         if iter_class is CurricularLevelIterator:  
190             iter_args['logger'] = training_logger  
191             iter_args['curriculum_params'] = {  
192                 'curriculum_distribution': config.setdefault(  
193                     'curriculum_distribution', 'uniform')
```

164 Probability of picking level 2 instead of level 1.  
165 """  
166 def \_\_init\_\_(self, level1, level2, p\_switch, \*\*kwargs):  
167 + super().\_\_init\_\_(level1, level2, repeat\_levels=True, \*\*kwargs)  
168 self.p\_switch = p\_switch  
169  
170 def get\_next\_parameters(self):  
171 """  
172 Expand 38 lines ...  
173 """  
174 'navigate': {  
175 'iter\_class': SafeLifeLevelIterator,  
176 'train\_levels': ['random/navigation'],  
177 'validation\_levels': ['random/navigation'],  
178 'benchmark\_levels': 'benchmarks/v1.0/navigation.npz',  
179 },  
180 # Multi-agent tasks:  
181 """  
182 Expand 116 lines ...  
183 """  
184 iter\_class = task\_data.get('iter\_class', SafeLifeLevelIterator)  
185 iter\_args = {'seed': training\_seed}  
186  
187 if iter\_class is CurricularLevelIterator:  
188 iter\_args['logger'] = training\_logger  
189 iter\_args['curriculum\_params'] = {  
190 'curriculum\_distribution': config.setdefault(  
191 'curriculum\_distribution', 'uniform')}

Expand 69 lines ...



# P.S. Store seeds & randomization settings in wandb.config

```
138 def main(config):
139     config.name = config_desc(config)
140     if config.use_wandb:
141         run.save()
142
143     # set random seed
144     tf.random.set_seed(config.random_seed)
145     # also set numpy seed to control train/val dataset split
146     np.random.seed(config.random_seed)

204     def set_global_seed(config):
205         from safelife.random import set_rng
206
207         # Make sure the seed can be represented by floating point exactly.
208         # This is just because we want to pass it over the web, and javascript
209         # doesn't have 64 bit integers.
210         if config.get('seed') is None:
211             config['seed'] = np.random.randint(2**53)
212         seed = np.random.SeedSequence(config['seed'])
213         logger.info("SETTING GLOBAL SEED: %i", seed.entropy)
214         set_rng(np.random.default_rng(seed))
215         torch.manual_seed(seed.entropy & (2**31 - 1))

216
217         if config['deterministic']:
218             # Note that this may slow down performance
219             # See https://pytorch.org/docs/stable/notes/randomness.html#cudnn
220             torch.backends.cudnn.deterministic = True
```

# Save each script run

- Run overview tab: see all the details
- Save the most recent git commit, the command and args, and system hardware setup (+ requirements.txt!)
- W&B also saves a patch file with uncommitted changes so you can reproduce the exact code that trained the model
- [Example run](#)

dutiful-dragon-174 ↗

What makes this run special? ↗

Privacy	 PUBLIC
Tags	<a href="#">+</a>
Author	stacey
State	finished
Start time	January 24th, 2020 at 1:30:16 pm
Duration	38m 27s
Run path	stacey/deep-drive/6zsn8ltb
Hostname	wbrave
OS	Linux-5.0.0-37-generic-x86_64-with-debian-buster-sid
Python version	3.7.2
Python executable	/home/stacey/.pyenv/versions/sd/bin/python
Git repository	<code>git clone https://github.com/borisdayma/semantic-segmentation.git</code>
Git state	<code>git checkout -b "dutiful-dragon-174" f0a9494a5d152663d226e28e279c65</code>
Command	<code>train.py</code>
CPU count	20
System Hardware	GPU count 2
	GPU type GeForce RTX 2080 Ti
W&B CLI Version	0.8.21

[bit.ly/deep-drive](https://bit.ly/deep-drive)

# Persistent system of record

Query and filter across thousands of runs, and easily keep projects organized.

Runs (395)															
<input type="checkbox"/>	Name (228 visualized)	Tags	Runtime	batch_size	encoder	learning_rate	num_train	num_valid	weight_decay	iou	train_loss	valid_loss	acc	traffic_acc	road
-	best car acc (50% data)	seg_masks	47m 52s	6	resnet34	0.001311	3524	492	0.08173	0.7997	0.5375	0.4427	0.8823	0.8664	0.93
-	best traffic acc (50% data)	seg_masks	46m 42s	8	resnet18	0.001	3523	492	0.097	0.8073	0.4919	0.4203	0.888	0.8718	0.94
-	best human iou (50% data)	seg_masks	31m 34s	7	alexnet	0.0009084	1405	190	0.097	0.716	0.6222	0.6259	0.8334	0.8042	0.9
-	best overall IOU (20% data)	seg_masks	20m 23s	7	resnet34	0.001367	1376	205	0.06731	0.7948	0.5096	0.4659	0.8726	0.8593	0.93
-	vibrant-cherry-426		6m 12s	8	resnet34	0.001	347	42	0.097	0.752	0.7114	0.6055	0.8474	0.8282	0.95
-	worldly-totem-422		12m 54s	8	resnet34	0.001	682	97	0.097	0.7523	0.5774	0.5836	0.8566	0.8349	0.91
-	jumping-voice-421		11m 59s	8	resnet34	0.001	725	92	0.097	0.7449	0.5633	0.5334	0.8504	0.8296	0.91
-	logical-energy-420	test_only	2m 14s	8	resnet34	0.001	66	10	0.097	0.4297	1.459	1.221	0.626	0.5958	0.76

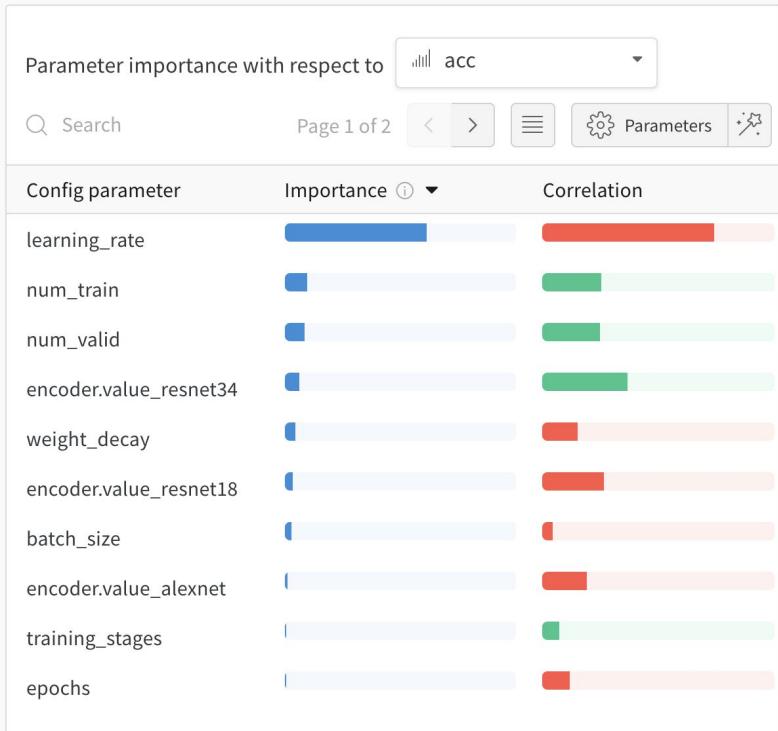
Understand models  
through interactive visualizations

[bit.ly/deep-drive](https://bit.ly/deep-drive)

# Compare runs

Visualize the relationships between hyperparameters and model metrics.

Explore the space of possible models quickly, without getting bogged down setting up manual visualizations.

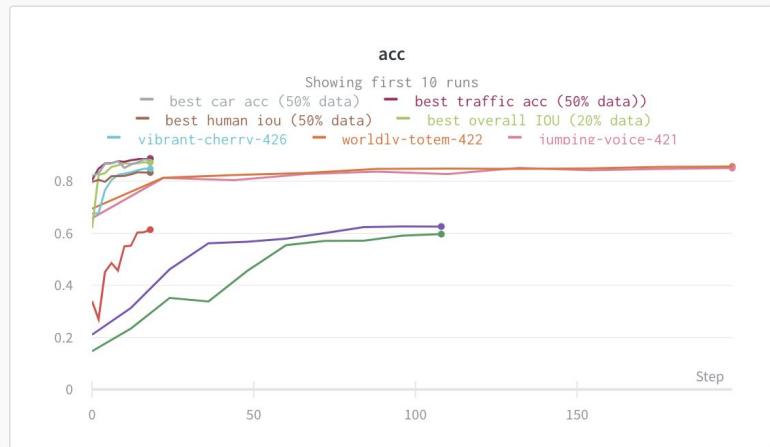
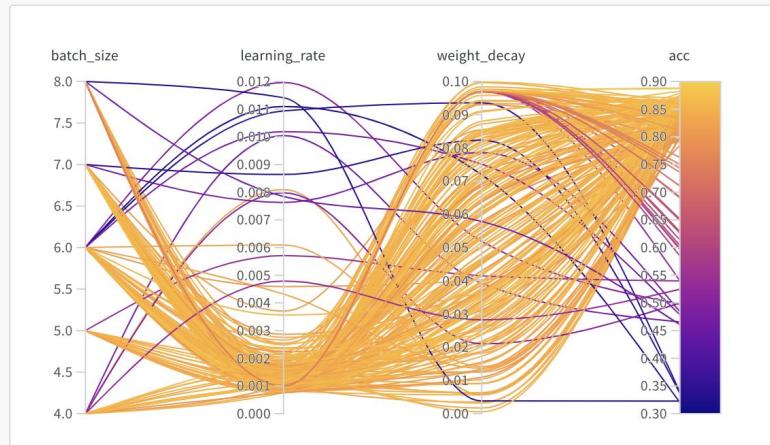


[bit.ly/deep-drive](https://bit.ly/deep-drive)

# Iterate quickly Accelerate time to insight

Use the interactive dashboard to  
spot issues in real time.

Stop underperforming runs early  
to optimize resource utilization.



[bit.ly/deep-drive-report](https://bit.ly/deep-drive-report)

## REPORTS

# Annotate progress

Use reports to keep a work log and quickly pick up where you left off.

## Reports

Last edited

Created by

### The View from the Driver's Seat

segmentation for scene  
Berkeley Deep Drive 100K



1 month ago

stacey

### asks for Semantic tation

; and explore semantic  
ion masks



1 month ago

stacey

### c Segmentation Masks

; and explore semantic  
ion masks



3 months ago

stacey

### c Segmentation Demo

e segmentation model using  
; car scenes. Click the Gear  
interact with the scene.



3 months ago

nbaryd

### [WIP] Semantic Segmentation from Dashcam

Ongoing notes, exploration, and  
development



4 months ago

stacey

[bit.ly/deep-drive-report](https://bit.ly/deep-drive-report)

REPORTS

# Collaborate easily

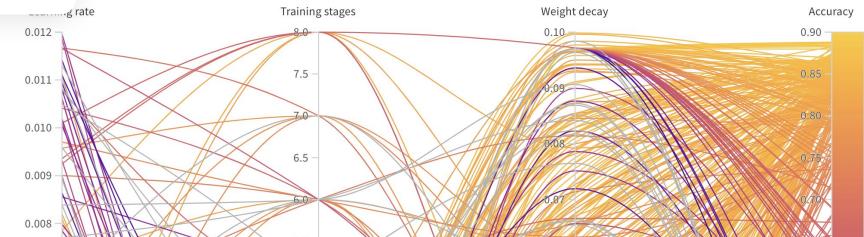
Give team members access to your exploratory results, sharing the context they need to quickly build on your work.

INSTALL • TRACK • COMPARE • OPTIMIZE • COLLABORATE

## Weight decay: inconclusive

Initially increasing the weight decay 5X improved the accuracy by 9%. Increasing by 200X causes the same amount of improvement though.

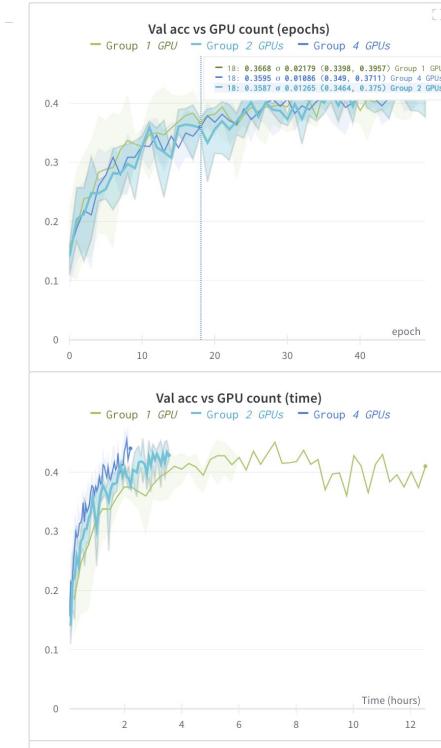
### Hyperparameter Sweep Insights





# Sharing model insights with W&B

## ▼ Scaling to 4 GPUs



Speed up: 4 GPUs: 2.5X, 2 GPUs: 1.6X

2.5X faster training on 4 GPUs vs 1 GPU

- model reaches a slightly higher validation accuracy 2.5X as fast when using 4 versus 1 GPU—this is the main advantage
- train/val acc/loss are not significantly affected by parallelizing the job across 1, 2, or 4 GPUs—this is expected and reassuring
- could continue tuning to improve relative speed-up—improvement is less than linear
- need better metrics (data throughput, batches per unit time, time to convergence) to quantify the added value of distributed training

### Experiment

Train a 7-layer convnet on main iNaturalist dataset (5000 train / 800 val) as a proof of concept for the Keras multi\_gpu\_model function.

### Notes

- initially no noticeable difference between 1 GPU and 2 GPUs—masked by one extremely slow run, batch 64\_2, which stalled for 2 hours during training for unknown reasons. Leaving it out of the average shows that 2 GPUs yield a 1.6x acceleration
- accuracy vs batch size: consider effect of both batch size and number of GPUs
- for 2 GPUs, batch 64 > batch 32 / 128 > batch 256, but the effect is not super clear. 64 seems to be the optimal choice for batch size
- some combinations might be slower—resource sharing on the GPUs



# 5 lines of code

```
!pip install wandb
import wandb
wandb.init(project="classify_photos")

# save any experiment configuration
wandb.config = {"learning_rate": 0.001, "epochs": 10}

# define & train your model

# log any metric from your training script
wandb.log({"acc": accuracy, "val_acc": val_accuracy})
```

Understand datasets  
through interactive visualizations

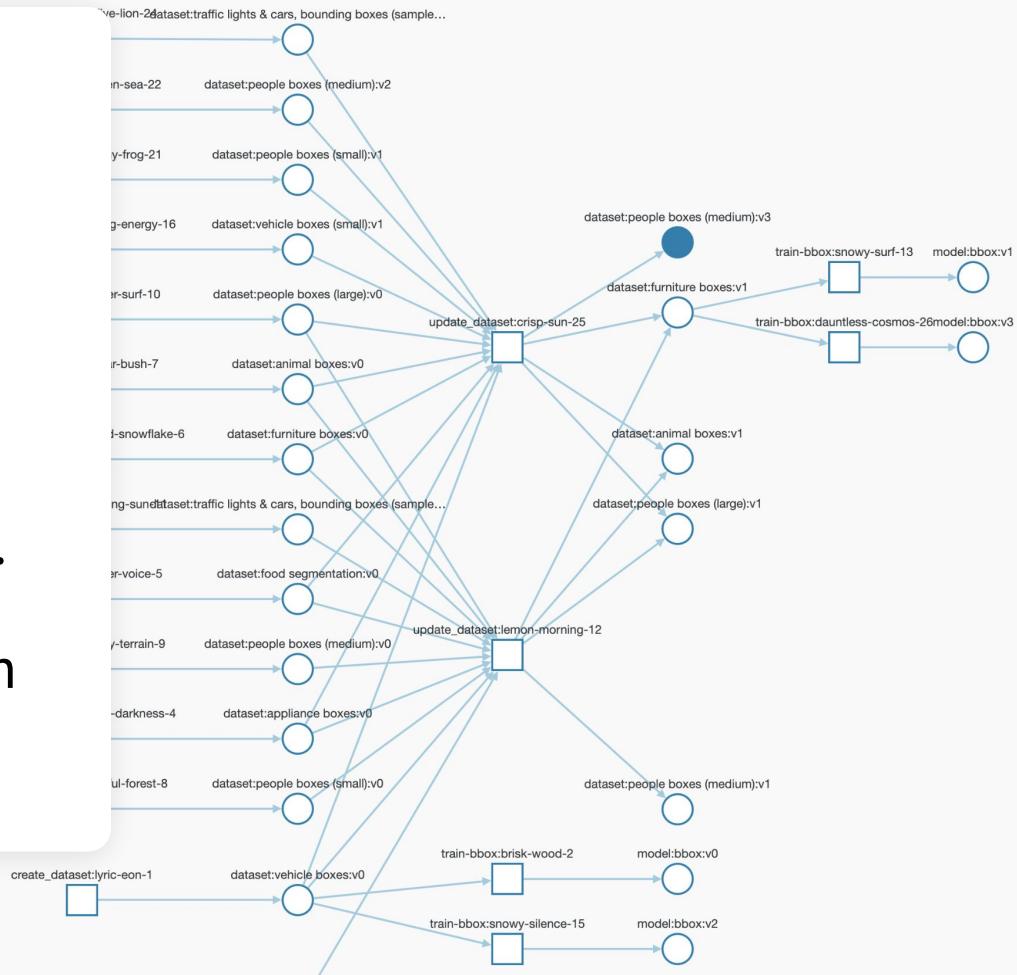


# Track artifacts

Get a bird's eye view of every step of model development.

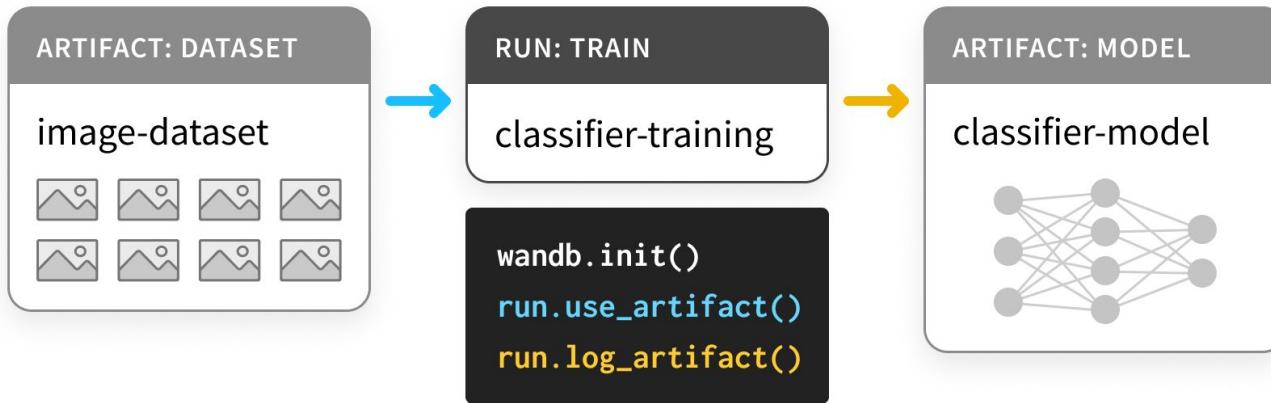
Automatically checksum and version datasets and models.

Host your files anywhere with our infra-agnostic tracking.



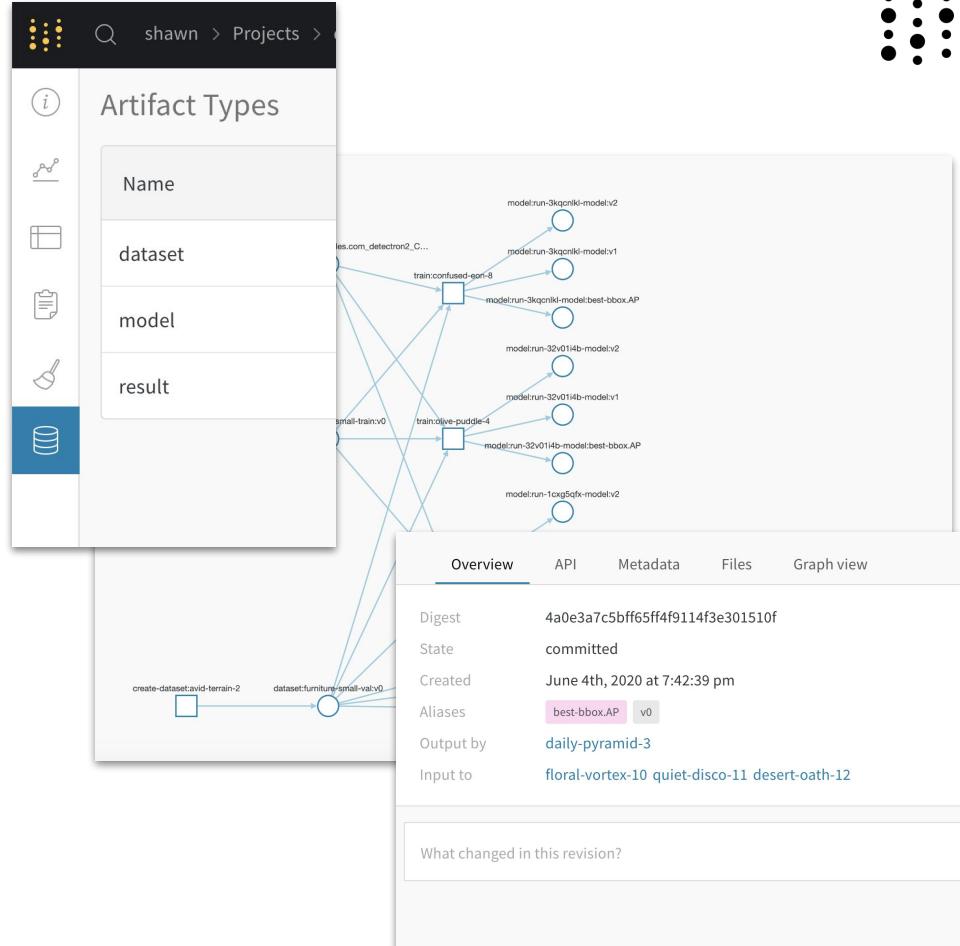


<https://docs.wandb.com/artifacts>



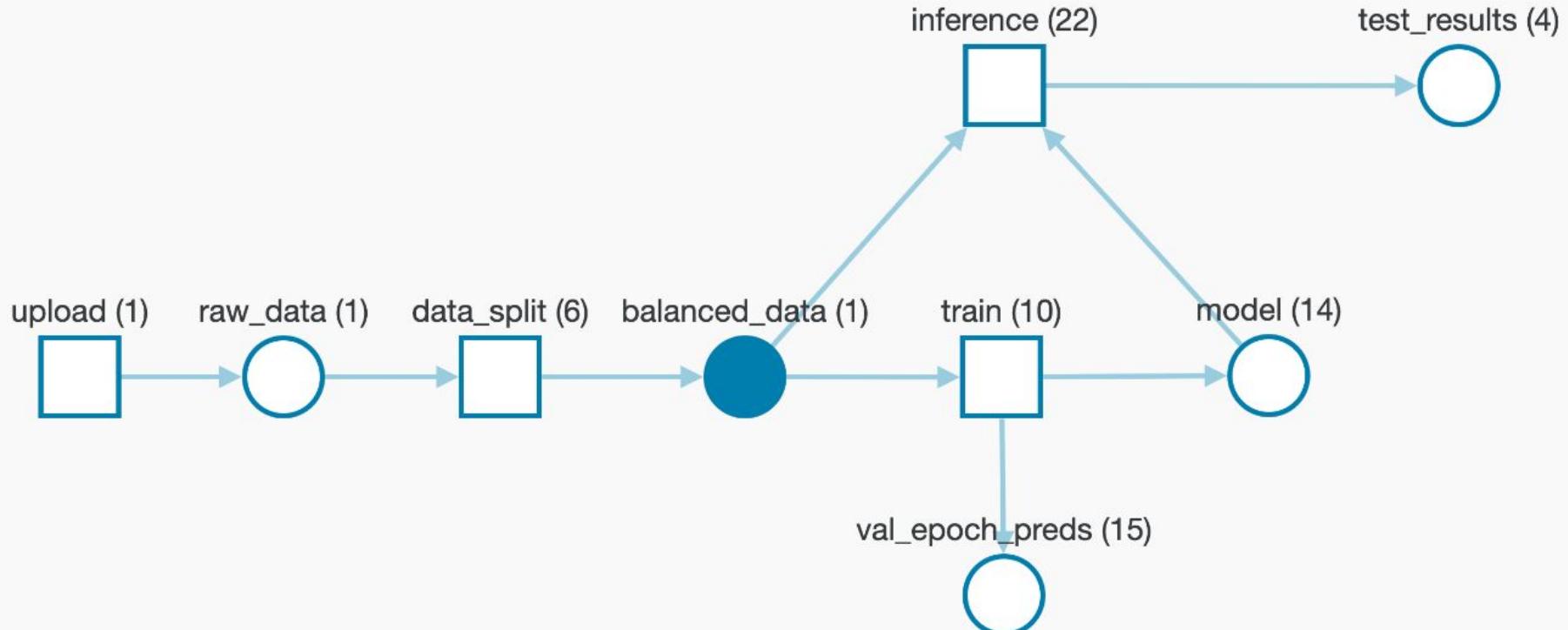
# Artifacts

- Integrated with the project
  - Get a clear understanding of what is consumed by and generated from runs and projects
- Can be consumed by any run with access to the project
  - Artifacts as a serving agent for downstream systems
- Automatically generates lineage graphs





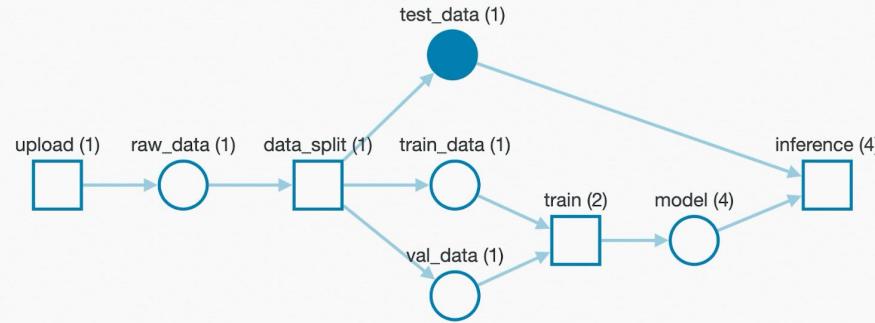
# Version datasets, models & any files



[Artifacts Quickstart →](#)



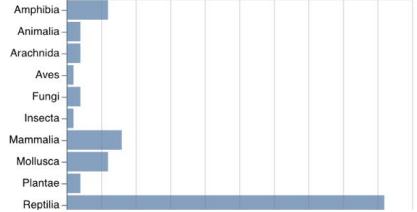
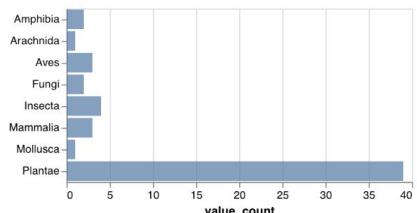
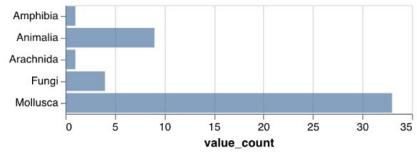
# Organize, visualize, and share workflows





# Images

Group by (guess) ↓      image      truth

Group	Image 1	Image 2	Image 3	Truth Distribution
Reptilia				
Plantae				
Mollusca				

1-3 of 76    < >

10-12 of 55    < >

1-3 of 48    < >

[Example →](#)

[Report →](#)



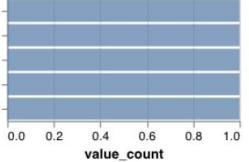
# Segmentation masks

id	prediction	ground truth	overall IOU	false positive	false negative	iou_road
2b9bc4f8-780bd01f			0.6061	171737	213406	0.5211
134f6849-00000000			0.4365	350147	61326	0.8978
382e37b6-139b8d9f			0.4425	298525	206063	0.8833
83a3ef4b-9e72764d			0.7372	157810	113137	0.7578

Example →



# Text

Group by (temperature)	prompt	response
0.3	<p>And this our life, exempt from O brave new world that hath such cre...</p> <p>Shall I compare thee to a-</p> <p>To be or not to be that is the-</p> <p>Tomorrow and tomorrow and tomorrow</p>	 <p>To be or not to be that is the dearty to the brocaty of the persing stands and be is the beard to the sease that that the last may</p> <p>1-3 of 5 &lt; &gt;</p>
0.6	<p>And this our life, exempt from O brave new world that hath such cre...</p> <p>Shall I compare thee to a-</p> <p>To be or not to be that is the-</p> <p>Tomorrow and tomorrow and tomorrow</p>	<p>To be or not to be that is the his visitition of prose, and may make to his tree the best kinous and it be herart. MERCUTIO: Why s</p> <p>1-3 of 5 &lt; &gt;</p>
0.8	<p>And this our life, exempt from O brave new world that hath such cre...</p> <p>Shall I compare thee to a-</p> <p>To be or not to be that is the-</p> <p>Tomorrow and tomorrow and tomorrow</p>	<p>To be or not to be that is the see and hence not it aradements, the stands all one. ATBON: As stay the strange and I am before gon</p> <p>1-3 of 5 &lt; &gt;</p>

[Example →](#)

[Report →](#)



# Videos

Type: video

▼ videos\_append-spawn

v1 latest

v0

▶ video

Overview API Metadata **Files** Graph view

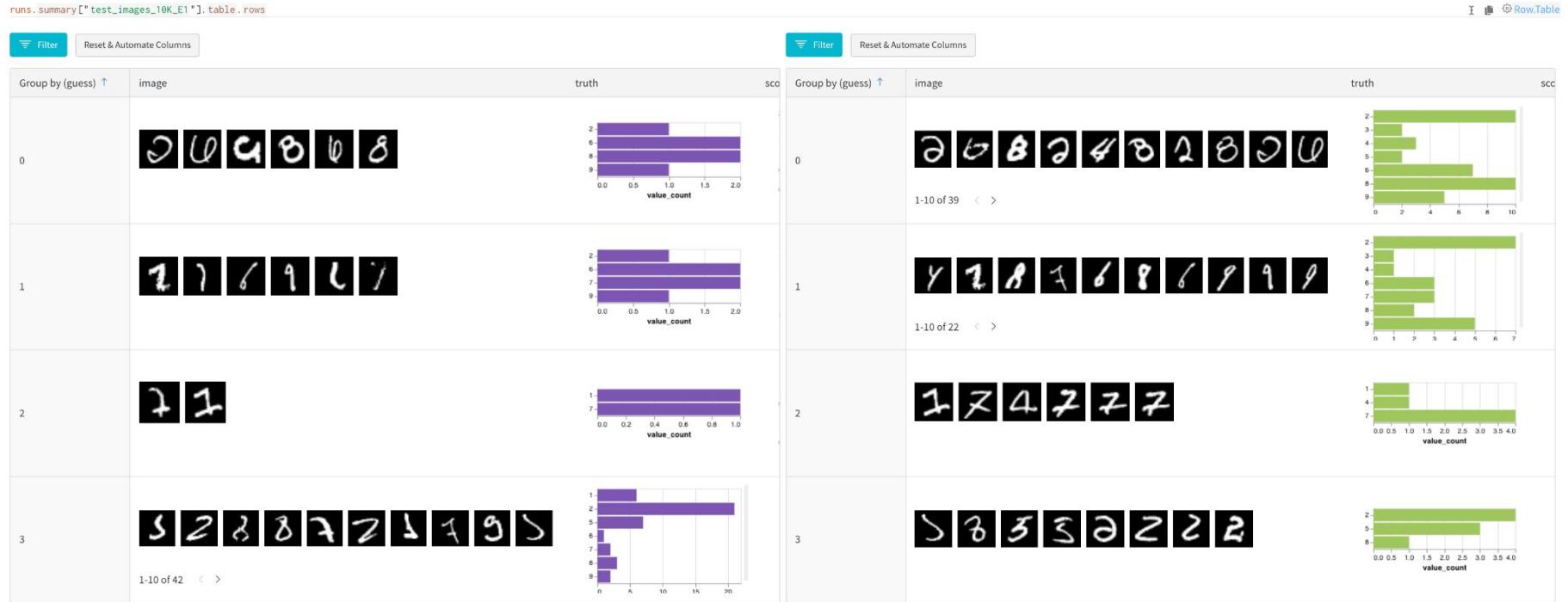
Filter 1-10 of 40002 < >

video	side_effects	length	reward	success	score ↓	steps	episodes
	0	50	0.9706	0	96.544	1279994	25597
	0	50	0.9706	0	96.544	1561417	31228
	0	50	0.9706	0	96.544	1893888	37877

Example →



# Interactive exploration & comparison



Single table →

Comparison →



# Tables Use Cases

- Log interactive media
- Manage data splits & iteratively refine datasets
- Visualize predictions
- Explore dynamically: filter, sort, group, add columns, & more
- Compare model variants
- Save Tables to workspace, report, etc.



# Example: why are two models different?

stacey > Projects > evalserver\_answers > Artifacts > results > eval\_Janet > ec2ff5fda > files > eval\_results.table.json

1

Type: results

eval\_Felix  
v0 latest  
eval\_Ivy  
eval\_Janet  
v0 latest  
eval\_Elon  
eval\_Daenerys  
eval\_Charlie  
eval\_Bob  
v0 latest  
eval\_Ada

Overview API Metadata Files Graph view

Sort [50 rows]

eval\_Bob eval\_Janet overall IOU false positive

0-overall IOU - 1-overall IOU -

0.0 0.2 0.4 0.6 0.8

0 50,000 100,000 150,000

0-false positive - 1-false positive -

0 50,000 100,000 150,000

0-prediction, 1-prediction 0-prediction, 1-prediction

0-overall IOU - 1-overall IOU -

0.0 0.2 0.4 0.6

0 100,000 200,000 300,000

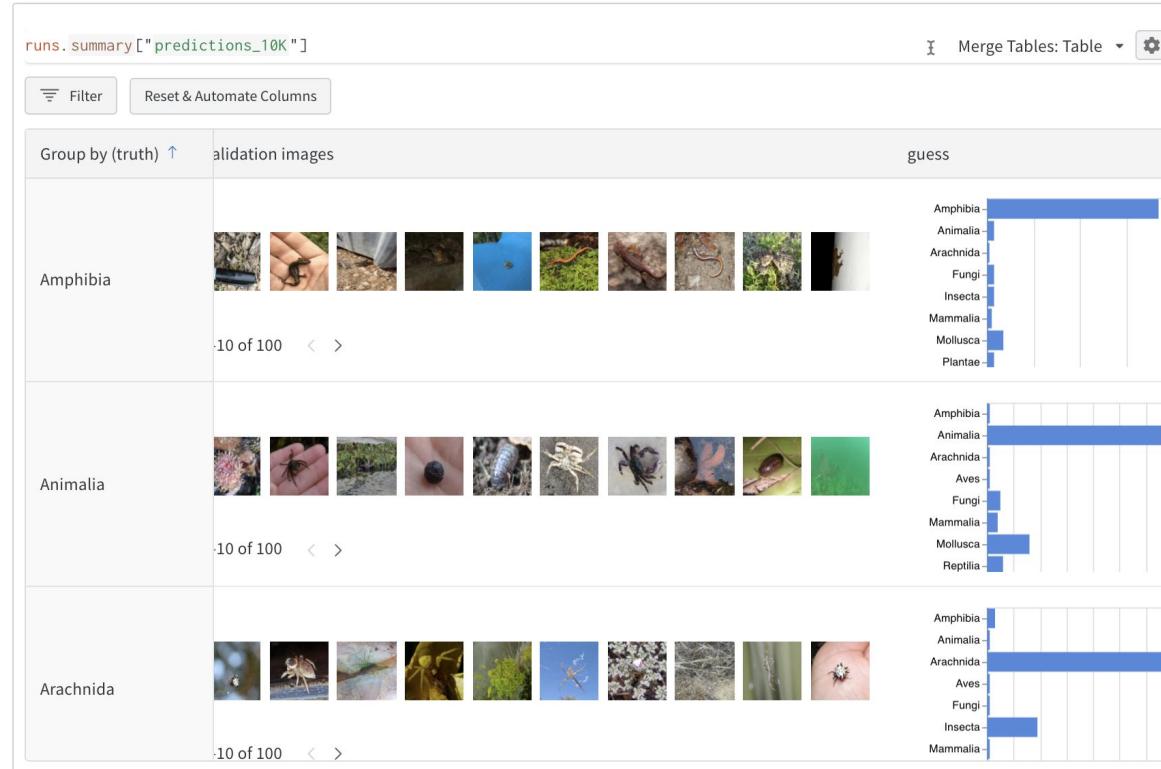
0-false positive - 1-false positive -

0 100,000 200,000 300,000

Chat icon



# Spot classes that confound the model





# Spot images that confound the model

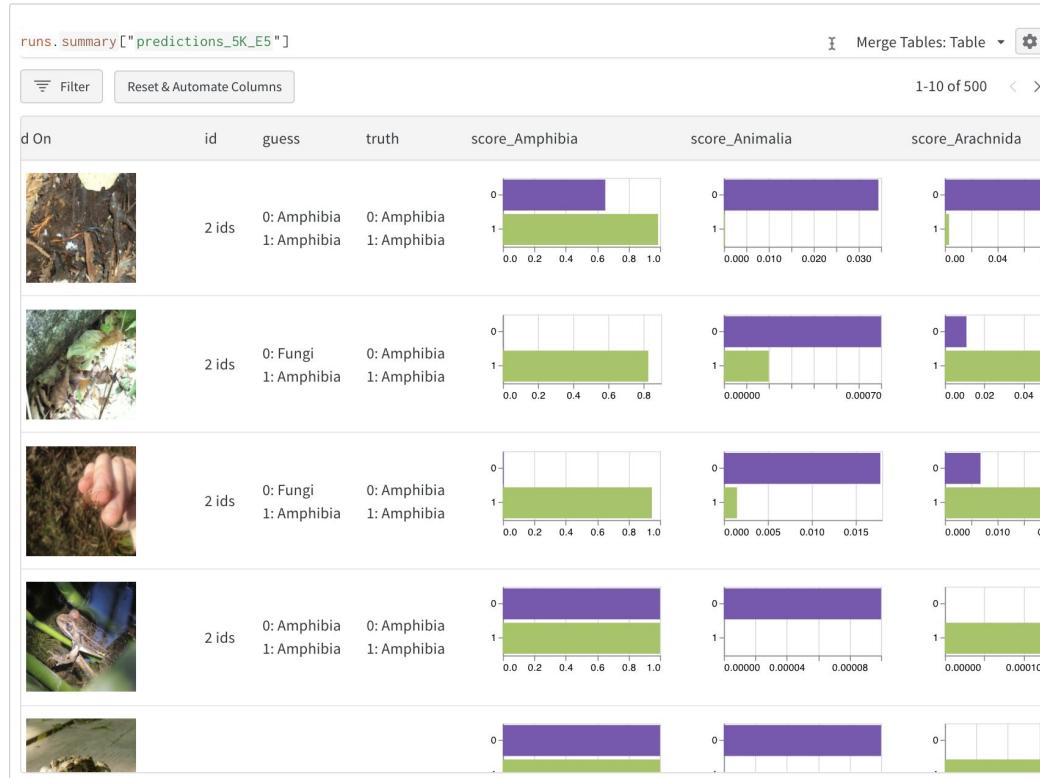
runs.summary ["predictions_10K"]				Merge Tables: Table	⚙️
Filter		Reset & Automate Columns		1-10 of 26 < >	
image	guess	truth	score_Amphib		
	Reptilia	Amphibia	0.401		
	Mollusca	Amphibia	0.3202		
	Fungi	Amphibia	0.3091		
	Mollusca	Amphibia	0.2878		

runs.summary ["predictions_10K"]			Merge Tables: Table	⚙️
Filter		Reset & Automate Columns	Group by (guess) ↑	
Group by (guess)	id	image		
Animalia	3 ids			
Arachnida	1 ids			
Fungi	3 ids			



# Compare performance across models



# Questions?

Docs      [docs.wandb.com](https://docs.wandb.com)

Community      [community.wandb.ai/](https://community.wandb.ai/)

Questions?      [twitter.com/lavanyaai](https://twitter.com/lavanyaai)  
[lavanya@wandb.com](mailto:lavanya@wandb.com)



# Thank You!

# DSC 102

# Systems for Scalable Analytics

Arun Kumar

Topic 4: Dataflow Systems

Chapter 2.2 of MLSys Book

*Q: How to shield users from needing to think about moving raw pages between disk/RAM/network to scale data-intensive programs?*

# Parallel RDBMSs

- ❖ Parallel RDBMSs are highly successful and widely used
- ❖ Typically shared-nothing data parallelism
- ❖ Optimized **runtime performance** + enterprise-grade features:
  - ❖ ANSI SQL & more
  - ❖ Business Intelligence (BI) dashboards/APIs
  - ❖ Transaction management; crash recovery
  - ❖ Indexes, auto-tuning, etc.

*Q: So, why did people need to go beyond parallel RDBMSs?*

**Ad:** Take my CSE 132C for more on parallel RDBMSs

# Beyond RDBMSs: A Brief History

- ❖ DB folks got blindsided by the rise of Web/Internet giants



- ❖ 4 new concerns of Web giants vs RDBMSs built for enterprises:
  - ❖ **Developability:** Custom data models and computations hard to program on SQL/RDBMSs; need for simpler APIs
  - ❖ **Fault Tolerance:** Need to scale to 1000s of machines; need for graceful handling of worker failure
  - ❖ **Elasticity:** Need to be able to easily upsize or downsize cluster size based on workload
  - ❖ **Cost:** Commercial RDBMSs licenses too costly; hired own software engineers to build custom new systems

A new breed of parallel data systems  
called **Dataflow Systems** jolted the DB  
folks from being smug and complacent!

# Outline

- ❖ Beyond RDBMSs: A Brief History
- ➔❖ MapReduce/Hadoop Craze
- ❖ Spark and Dataflow Programming
- ❖ Scalable BGD with MapReduce/Spark
- ❖ Dataflow Systems vs Task-Parallel Systems

# What is MapReduce?

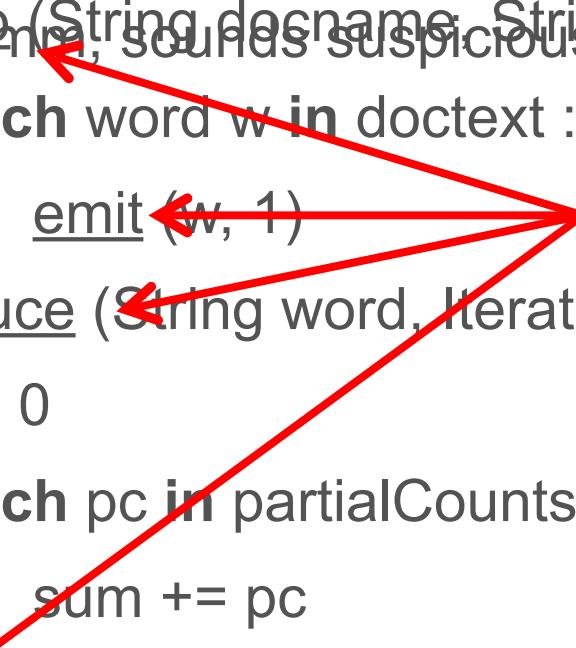
- ❖ A programming model for parallel programs on **sharded data + distributed system** architecture
- ❖ **Map** and **Reduce** are terms from functional PL; software/data/ML engineer implements logic of Map, Reduce
- ❖ System handles data distribution, parallelization, fault tolerance, etc. under the hood
- ❖ Created by Google to solve “simple” data workload: index, store, and search the Web!
- ❖ Google’s engineers started with MySQL! Abandoned it due to reasons listed earlier (developability, fault tolerance, elasticity, etc.)

# What is MapReduce?

- ❖ **Standard example:** count word occurrences in a doc corpus
- ❖ **Input:** A set of text documents (say, webpages)
- ❖ **Output:** A dictionary of unique words and their counts

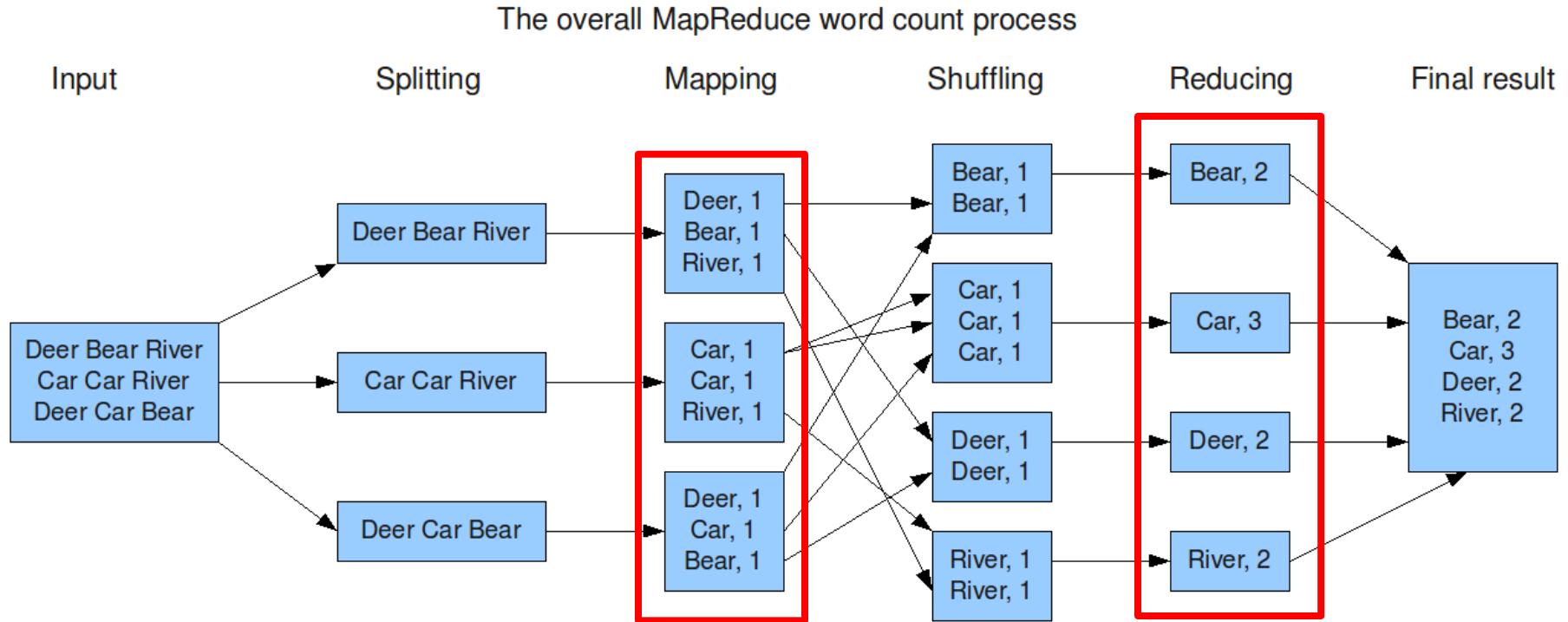
```
function map (String docname, String doctext) :  
    for each word w in doctext :  
        emit (w, 1)  
  
function reduce (String word, Iterator partialCounts) :  
    sum = 0  
    for each pc in partialCounts :  
        sum += pc  
    emit (word, sum)
```

*Part of MapReduce API*



# How MapReduce Works

Parallel flow of control and data during MapReduce execution:



Under the hood, each **Mapper** and **Reducer** is a separate process;  
Reducers face barrier synchronization (BSP)  
Fault tolerance achieved using **data replication**

# Benefits and Catch of MapReduce

- ❖ **Goal:** High-level *functional* ops to simplify data-intensive programs
- ❖ **Key Benefits:**
  - ❖ Map() and Reduce() are highly general; any data types/structures; great for ETL, text/multimedia
  - ❖ Native scalability, large cluster parallelism
  - ❖ System handles fault tolerance automatically
  - ❖ Decent FOSS stacks (Hadoop and later, Spark)
- ❖ **Catch:** Users must learn “art” of casting program as MapReduce
  - ❖ Map operates record-wise; Reduce aggregates globally
  - ❖ But MR libraries now available in many PLs: C/C++, Java, Python, R, Scala, etc.

# Abstract Semantics of MapReduce

- ❖ **Map():** Process one “record” at a time *independently*
  - ❖ A record can physically *batch* multiple data examples/tuples
  - ❖ Dependencies across Mappers *not allowed*
  - ❖ *Emit* 1 or more key-value pairs as output(s)
  - ❖ Data types of input vs. output can be different
- ❖ **Reduce():** Gather all Map outputs across workers sharing same key into an Iterator (list)
  - ❖ Apply *aggregation* function on Iterator to get final output(s)
- ❖ **Input Split:**
  - ❖ Physical-level shard to batch many records to one file “block” (HDFS default: 128MB?)
  - ❖ User/application can create *custom* Input Splits

# Emulate MapReduce in SQL?

**Q:** How would you do the word counting in RDBMS / in SQL?

❖ First step: **Transform** text docs into relations and load:

Part of the ETL stage

Suppose we pre-divide each doc into words w/ schema:

**DocWords** (DocName, Word)

❖ Second step: a single, simple SQL query!

SELECT           Word, COUNT (\*)

FROM             DocWords

GROUP BY       Word

[ORDER BY      Word]

Parallelism, scaling, etc. done  
by RDBMS under the hood

# More MR Examples: Select Operation

- ❖ **Input Split:**
  - ❖ Shard table tuple-wise
- ❖ **Map():**
  - ❖ On tuple, apply selection condition; if satisfies, emit KV pair with dummy key, entire tuple as value
- ❖ **Reduce():**
  - ❖ Not needed! No cross-shard aggregation here
- ❖ These kinds of MR jobs are called “**Map-only**” jobs

# More MR Examples: Simple Agg.

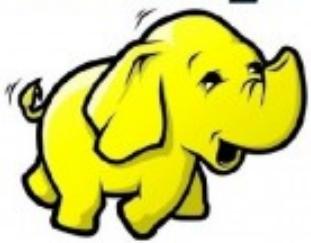
- ❖ Suppose it is *algebraic* aggregate (SUM, AVG, MAX, etc.)
- ❖ **Input Split:**
  - ❖ Shard table tuple-wise
- ❖ **Map():**
  - ❖ On agg. attribute, compute incr. stats; emit pair with single global dummy key and incr. stats as value
- ❖ **Reduce():**
  - ❖ Since only one global dummy key, Iterator has *all* suff. stats to unify into global agg.

# More MR Examples: GROUP BY Agg.

- ❖ Assume it is *algebraic* aggregate (SUM, AVG, MAX, etc.)
- ❖ **Input Split:**
  - ❖ Shard table tuple-wise
- ❖ **Map():**
  - ❖ On agg. attribute, compute incr. stats; emit pair with *grouping attribute* as key and stats as value
- ❖ **Reduce():**
  - ❖ Iterator has all suff. stats *for a single group*; unify those to get result for that group
  - ❖ Different reducers will output different groups' results

# More MR Examples: Matrix Norm

- ❖ Assume it is *algebraic* aggregate ( $L_{p,q}$  norm)
- ❖ Very similar to simple SQL aggregates
- ❖ **Input Split:**
  - ❖ Shard table tuple-wise
- ❖ **Map():**
  - ❖ On agg. attribute, compute incr. stats; emit pair with single global dummy key and stats as value
- ❖ **Reduce():**
  - ❖ Since only one global dummy key, Iterator has *all* suff. stats to unify into global agg.



# What is Hadoop then?

- ❖ FOSS system implementation with MapReduce as prog. model and HDFS as filesystem
- ❖ MR user API; input splits, data distribution, shuffling, fault tolerances handled by Hadoop under the hood
- ❖ Exploded in popularity in 2010s: 100s of papers, 10s of products
- ❖ A “revolution” in scalable+parallel data processing that took the DB world by surprise
- ❖ But nowadays Hadoop largely supplanted by Spark

**NB:** Do not confuse MR for Hadoop or vice versa!

# Outline

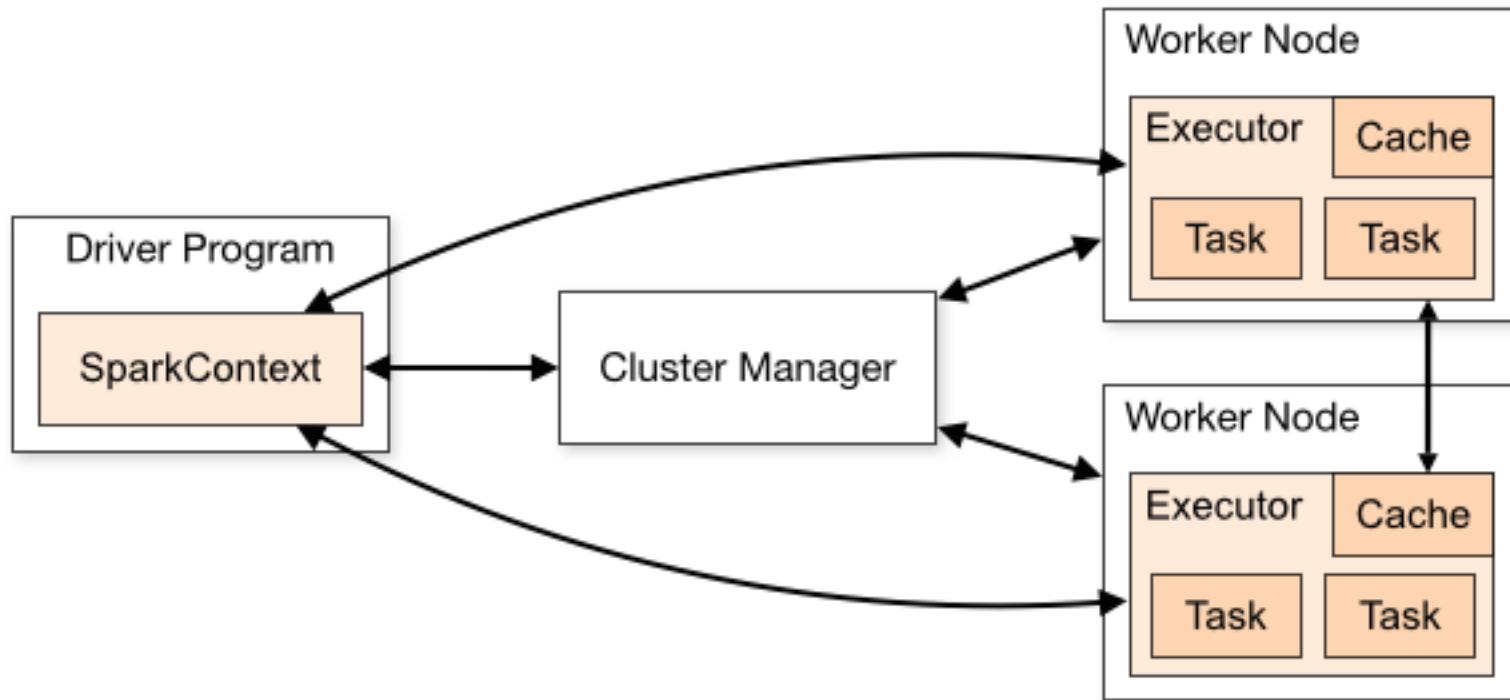
- ❖ Beyond RDBMSs: A Brief History
- ❖ MapReduce/Hadoop Craze
-  ❖ Spark and Dataflow Programming
- ❖ Scalable BGD with MapReduce/Spark
- ❖ Dataflow Systems vs Task-Parallel Systems

# Apache Spark



- ❖ **Dataflow programming** model (subsumes most of RA; MR)
  - ❖ Inspired by Python Pandas style of chaining functions
  - ❖ Unified storage of relations, text, etc.; custom programs
  - ❖ System impl. (re)designed from scratch
- ❖ Tons of sponsors, gazillion bucks, unbelievable hype!
- ❖ **Key idea vs Hadoop:** exploit distributed memory to cache data
- ❖ **Key novelty vs Hadoop:** lineage-based fault tolerance
- ❖ Open-sourced to Apache; commercialized as Databricks

# Distributed Architecture of Spark



# Spark's Dataflow Programming Model

**Transformations** are relational ops, MR, etc. as functions

**Actions** are what force computation; aka *lazy evaluation*

<b>Transformations</b>	<i>map</i> ( $f : T \Rightarrow U$ )	: $RDD[T] \Rightarrow RDD[U]$
	<i>filter</i> ( $f : T \Rightarrow \text{Bool}$ )	: $RDD[T] \Rightarrow RDD[T]$
	<i>flatMap</i> ( $f : T \Rightarrow Seq[U]$ )	: $RDD[T] \Rightarrow RDD[U]$
	<i>sample</i> ( $\text{fraction} : \text{Float}$ )	: $RDD[T] \Rightarrow RDD[T]$ (Deterministic sampling)
	<i>groupByKey()</i>	: $RDD[(K, V)] \Rightarrow RDD[(K, Seq[V])]$
	<i>reduceByKey</i> ( $f : (V, V) \Rightarrow V$ )	: $RDD[(K, V)] \Rightarrow RDD[(K, V)]$
	<i>union()</i>	: $(RDD[T], RDD[T]) \Rightarrow RDD[T]$
	<i>join()</i>	: $(RDD[(K, V)], RDD[(K, W)]) \Rightarrow RDD[(K, (V, W))]$
	<i>cogroup()</i>	: $(RDD[(K, V)], RDD[(K, W)]) \Rightarrow RDD[(K, (Seq[V], Seq[W]))]$
	<i>crossProduct()</i>	: $(RDD[T], RDD[U]) \Rightarrow RDD[(T, U)]$
	<i>mapValues</i> ( $f : V \Rightarrow W$ )	: $RDD[(K, V)] \Rightarrow RDD[(K, W)]$ (Preserves partitioning)
	<i>sort</i> ( $c : \text{Comparator}[K]$ )	: $RDD[(K, V)] \Rightarrow RDD[(K, V)]$
<b>Actions</b>	<i>partitionBy</i> ( $p : \text{Partitioner}[K]$ )	: $RDD[(K, V)] \Rightarrow RDD[(K, V)]$
	<i>count()</i>	: $RDD[T] \Rightarrow \text{Long}$
	<i>collect()</i>	: $RDD[T] \Rightarrow Seq[T]$
	<i>reduce</i> ( $f : (T, T) \Rightarrow T$ )	: $RDD[T] \Rightarrow T$
	<i>lookup</i> ( $k : K$ )	: $RDD[(K, V)] \Rightarrow Seq[V]$ (On hash/range partitioned RDDs)
	<i>save</i> ( $path : \text{String}$ )	: Outputs RDD to a storage system, e.g., HDFS

# Word Count Example in Spark

Spark RDD API available in Python, Scala, Java, and R

```
text_file = sc.textFile("hdfs://...")  
counts = text_file.flatMap(lambda line: line.split(" ")) \  
    .map(lambda word: (word, 1)) \  
    .reduceByKey(lambda a, b: a + b)  
counts.saveAsTextFile("hdfs://...")
```

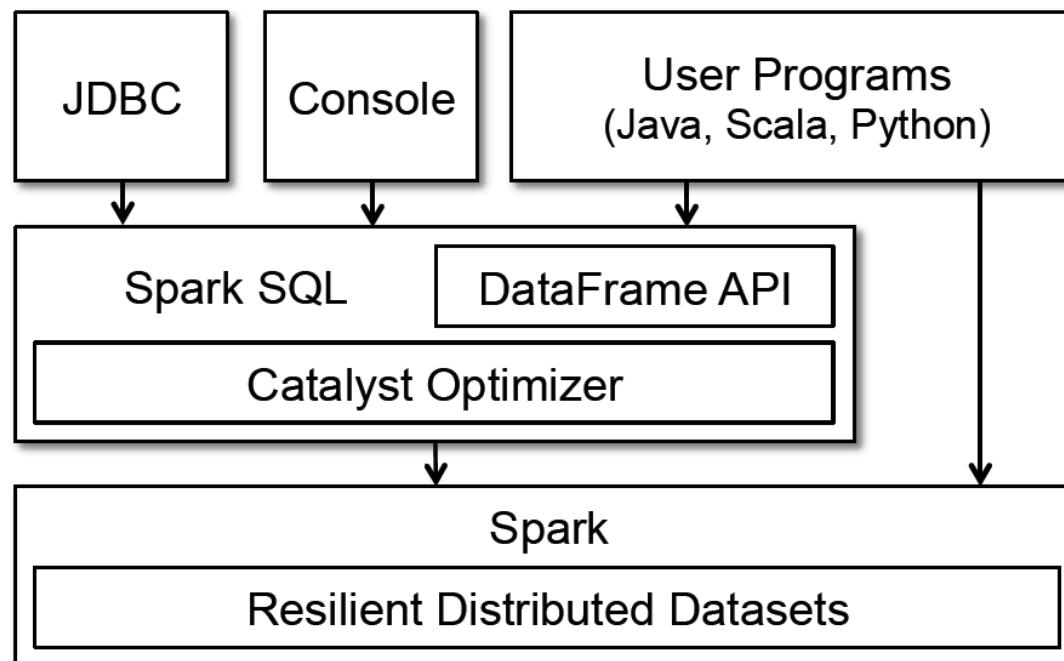
```
val textFile = sc.textFile("hdfs://...")  
val counts = textFile.flatMap(line => line.split(" "))  
    .map(word => (word, 1))  
    .reduceByKey(_ + _)  
counts.saveAsTextFile("hdfs://...")
```

```
JavaRDD<String> textFile = sc.textFile("hdfs://...");  
JavaPairRDD<String, Integer> counts = textFile  
    .flatMap(s -> Arrays.asList(s.split(" ")).iterator())  
    .mapToPair(word -> new Tuple2<>(word, 1))  
    .reduceByKey((a, b) -> a + b);  
counts.saveAsTextFile("hdfs://...");
```

Spark DataFrame API of SparkSQL offers an SQL interface  
Can also interleave SQL with DF-style function chaining!

# Spark DF API and SparkSQL

- ❖ Databricks now recommends SparkSQL/DataFrame API; avoid RDD API unless really needed!
- ❖ **Key Reason:** Automatic query optimization becomes more feasible
  - ❖ AKA (painfully) re-learn 40 years of database systems research! :)



# Query Optimization in Spark

- ❖ Common automatic query optimizations (from RDBMS world) are now performed in Spark's Catalyst optimizer:
- ❖ **Projection pushdown:**
  - ❖ Drop unneeded columns early on
- ❖ **Selection pushdown:**
  - ❖ Apply predicates close to base tables
- ❖ **Join order optimization:**
  - ❖ Not all joins are equally costly
- ❖ Fusing of aggregates
- ❖ ...

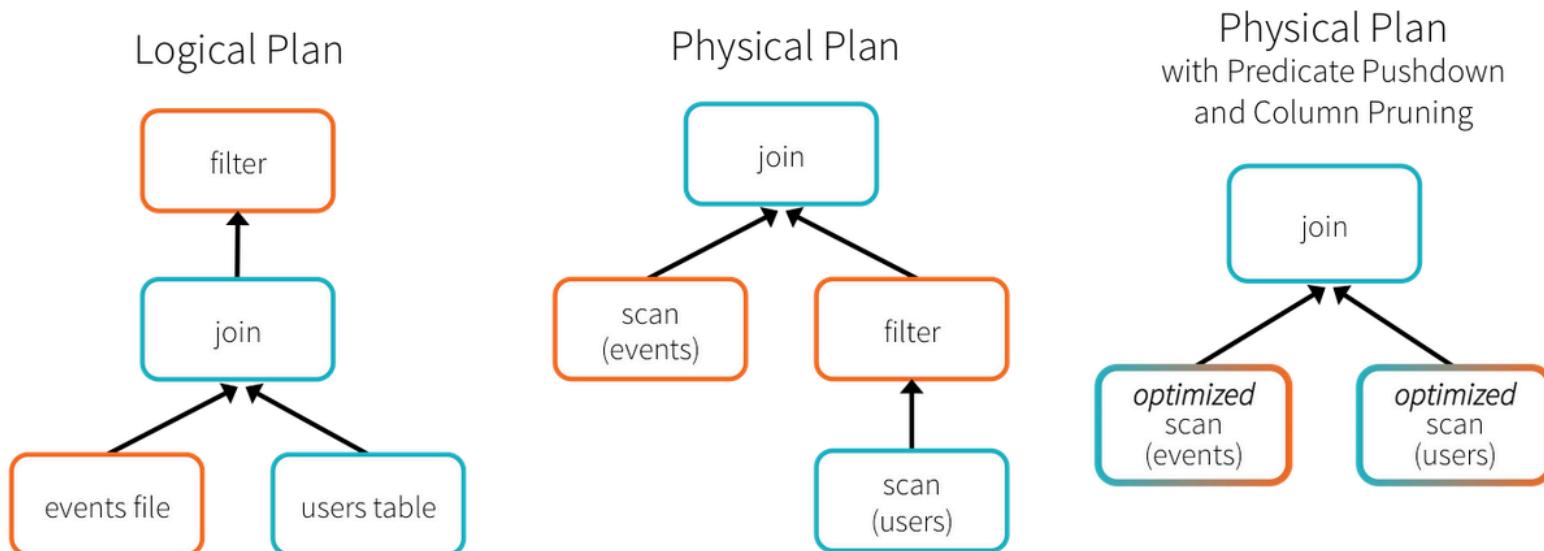
Spark SQL: Relational Data Processing in Spark. In SIGMOD 2015.

**Ad:** Take my CSE 132C for more on relational query optimization<sup>4</sup>

# Query Optimization in Spark

```
def add_demographics(events):
    u = sqlCtx.table("users")                      # Load partitioned Hive table
    events \
        .join(u, events.user_id == u.user_id) \
        .withColumn("city", zipToCity(df.zip))       # Run udf to add city column

events = add_demographics(sqlCtx.load("/data/events", "parquet"))
training_data = events.where(events.city == "New York").select(events.timestamp).collect()
```



Databricks is building yet another parallel RDBMS! :)

# Reinventing the Wheel?



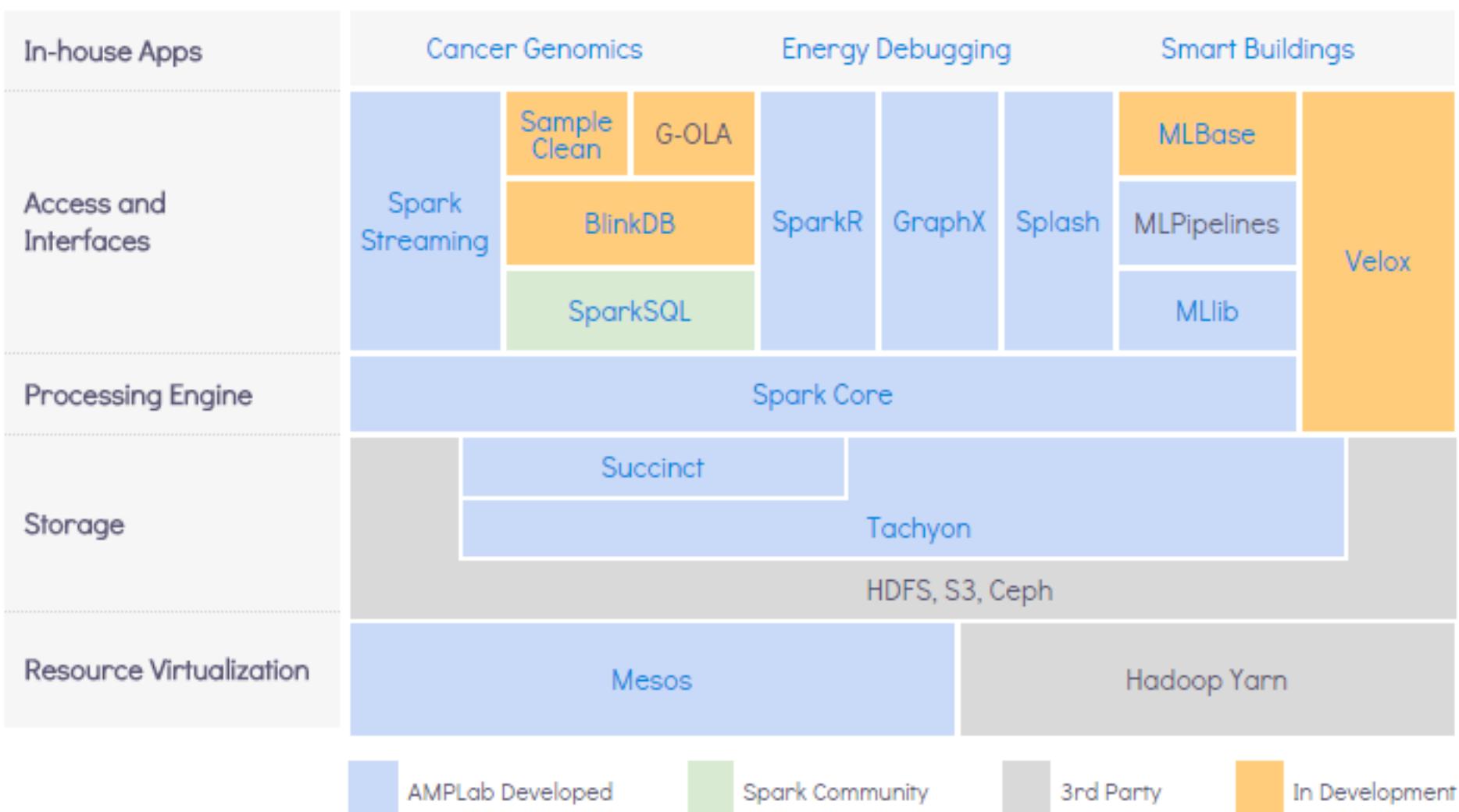
# Comparing Spark's APIs

Check out Pradyumna's PA 2 slides for more on Spark APIs

	RDD	DataFrame	Koalas
<b>Abstraction Level</b>	Low	High	High
<b>Named Columns</b>	No	Yes	Yes
<b>Support for Query Optimization</b>	No	Yes	Yes
<b>Programming Mode</b>	map-reduce	Dataflow, SQL	Pandas-like
<b>Best suited for</b>	Unstructured data Low-level ops Folks who like func. PLs and MapReduce	Structured data High-level ops Folks who know SQL, Python, R	Structured data Lower barrier to entry for folks who only know Pandas or Dask

**Ad:** Take Yoav's DSC 291 to learn more Spark programming

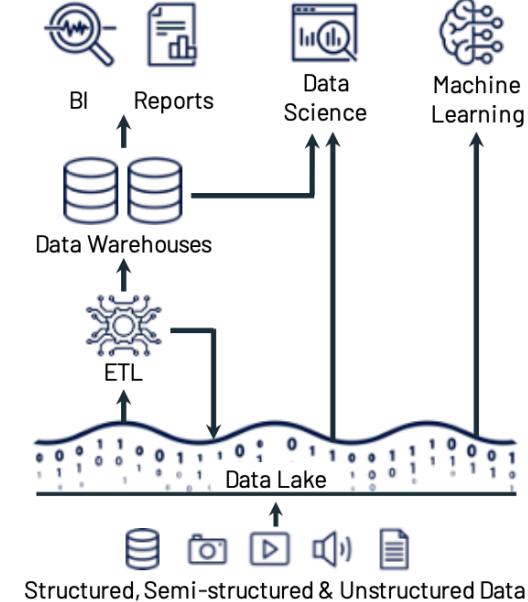
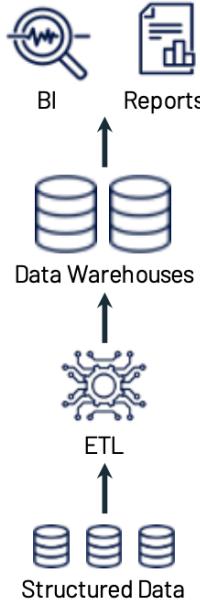
# Spark-based Ecosystem of Tools



*The Berkeley Data Analytics Stack (BDAS)*

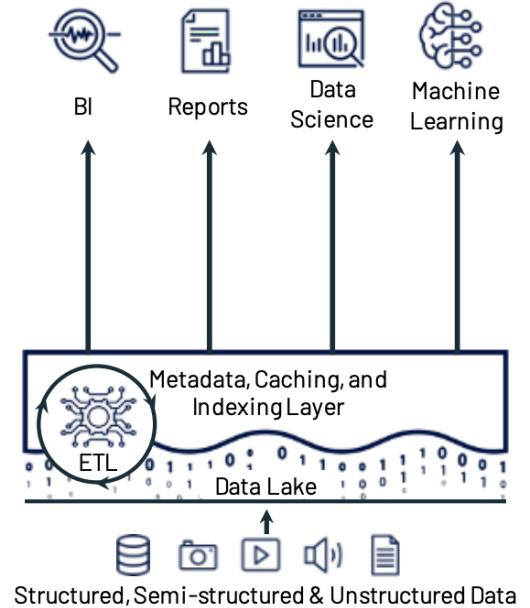
# New Paradigm of Data “Lakehouse”

- ❖ **Data “Lake”:** *Loose coupling* of data file format and data/query processing stack (vs RDBMS’s tight coupling); many frontends



(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 and data lakehouses:

[http://cidrdb.org/cidr2021/papers/cidr2021\\_paper17.pdf](http://cidrdb.org/cidr2021/papers/cidr2021_paper17.pdf)

# ... which too is a form of DBMS! :)

A  Reynold Xin  
@rxin

C Replying to @TweetAtAKK

While I agree with you, note that there's nothing wrong with "repeat and relearn". Many use cases' first order problems are not "data independence" (which is about change), but about being able to get the task done first.

Time fo   
<rant>  
A data  
A Data  
An ML  
A mod  
A featu

4:24 PM · Feb 12, 2022 · Twitter Web App

2 Likes

 Tweet your reply 

Those doome ... / are  
</rant>

Arun Kumar @TweetAtAKK · Feb 12  
Replies to @rxin  
Reynold! I was wondering who'd be the first to comment b/w you and @matei\_zaharia. 😊

Yes, agreed on your point. I put it more colorfully. 🌟 A wheel is a wheel but we don't use an Egyptian chariot wheels for a car nor a car's for a space shuttle. 😊

30

# References and More Material

## ❖ MapReduce/Hadoop:

- ❖ MapReduce: Simplified Data Processing on Large Clusters.  
Jeffrey Dean and Sanjay Ghemawat. In [OSDI 2004](#).
- ❖ More Examples: <http://bit.ly/2rkSRj8>
- ❖ Online Tutorial: <http://bit.ly/2rS2B5j>

## ❖ Spark:

- ❖ Resilient Distributed Datasets: A Fault-tolerant Abstraction for In-memory Cluster Computing. Matei Zaharia and others. In [NSDI 2012](#).
- ❖ More Examples: <http://bit.ly/2rhkhEp>, <http://bit.ly/2rkT8Tc>
- ❖ Online Guide: <https://spark.apache.org/docs/2.1.0/sql-programming-guide.html>

# Outline

- ❖ Beyond RDBMSs: A Brief History
- ❖ MapReduce/Hadoop Craze
- ❖ Spark and Dataflow Programming
-  ❖ Scalable BGD with MapReduce/Spark
- ❖ Dataflow Systems vs Task-Parallel Systems

# Example: Batch Gradient Descent

$$\nabla L(\mathbf{w}^{(k)}) = \sum_{i=1}^n \nabla l(y_i, f(\mathbf{w}^{(k)}, x_i))$$

- ❖ Very similar to algebraic SQL; vector addition
- ❖ **Input Split:** Shard table tuple-wise
- ❖ **Map():**
  - ❖ On tuple, compute per-example gradient; add these across examples in shard; emit partial sum with single dummy key
- ❖ **Reduce():**
  - ❖ Only one global dummy key, Iterator has partial gradients; just add all those to get full batch gradient

# Outline

- ❖ Beyond RDBMSs: A Brief History
- ❖ MapReduce/Hadoop Craze
- ❖ Spark and Dataflow Programming
- ❖ More Scalable ML with MapReduce/Spark
- ❖ Dataflow Systems vs Task-Parallel Systems

# Dataflow Systems vs Task-Par. Sys.

## ❖ Pros:

In-class activity

## ❖ Cons:

In-class activity

# Specific to Spark vs Dask?

- ❖ **Pros:**

- In-class activity

- ❖ **Cons:**

- In-class activity

Optional: More complex examples of  
MapReduce usage to scale ML  
Not included in syllabus

# Primer: K-Means Clustering

- ❖ **Basic Idea:** Identify clusters based on Euclidean distances; formulated as an optimization problem
  - ❖ **Lloyd's algorithm:** Most popular heuristic for K-Means
  - ❖ **Input:**  $n \times d$  examples/points
  - ❖ **Output:**  $k$  clusters and their centroids
1. Initialize  $k$  centroid vectors and point-cluster ID assignment
  2. **Assignment step:** Scan dataset and assign each point to a cluster ID based on which centroid is *nearest*
  3. **Update step:** Given new assignment, scan dataset again to recompute centroids for all clusters
  4. Repeat 2 and 3 until convergence or fixed # iterations

# K-Means Clustering in MapReduce

- ❖ **Input Split:** Shard the table tuple-wise
  - ❖ Assume each tuple/example/point has an *ExampleID*
- ❖ Need 2 jobs! 1 for Assignment step, 1 for Update step
- ❖ 2 external data structures needed for both jobs:
  - ❖ Dense matrix A:  $k \times d$  centroids; ultra-sparse matrix B:  $n \times k$  assignments
  - ❖ A and B first broadcast to all Mappers via HDFS; Mappers can read small data directly from HDFS files
  - ❖ Job 1 read A and creates new B
  - ❖ Job 2 reads B and creates new A

# K-Means Clustering in MapReduce

- ❖ A:  $k \times d$  centroid matrix; B:  $n \times k$  assignment matrix
- ❖ **Job 1 Map():** Read A from HDFS; compute point's distance to all  $k$  centroids; get nearest centroid; emit new assignment as output pair (PointID, ClusterID)
- ❖ No Reduce() for Job 1; new B now available on HDFS
- ❖ **Job 2 Map():** Read B from HDFS; look into B and see which cluster point got assigned to; emit point as output pair (ClusterID, point vector)
- ❖ **Job 2 Reduce():** Iterator has all point vectors of a given ClusterID; add them up and divide by count; got new centroid; emit output pair as (ClusterID, centroid vector)

# DSC 102

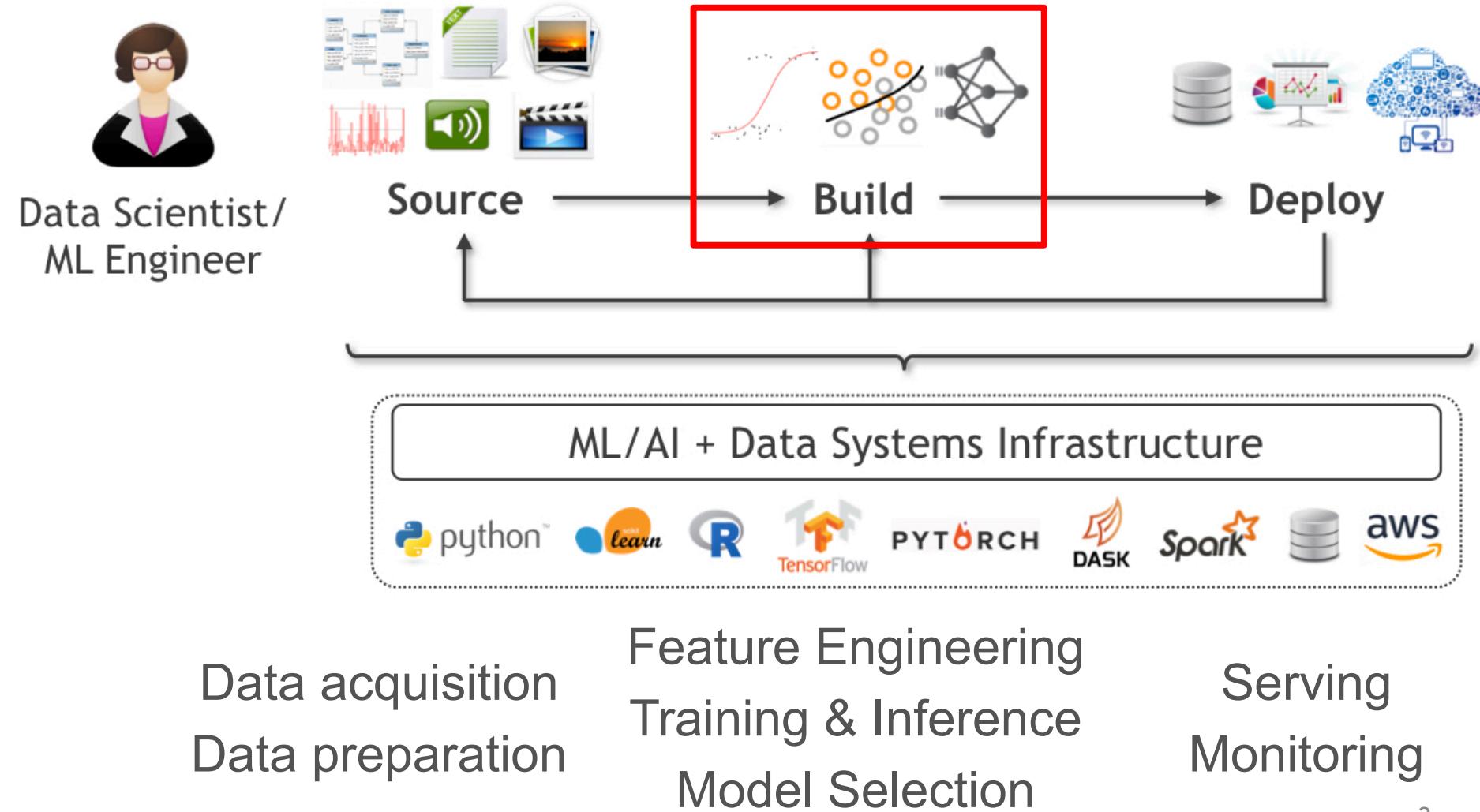
# Systems for Scalable Analytics

Arun Kumar

Topic 5: Model Building Systems

Chapter 8.1 and 8.3 of MLSys Book

# The Lifecycle of ML-based Analytics



# Building Stage of ML Lifecycle

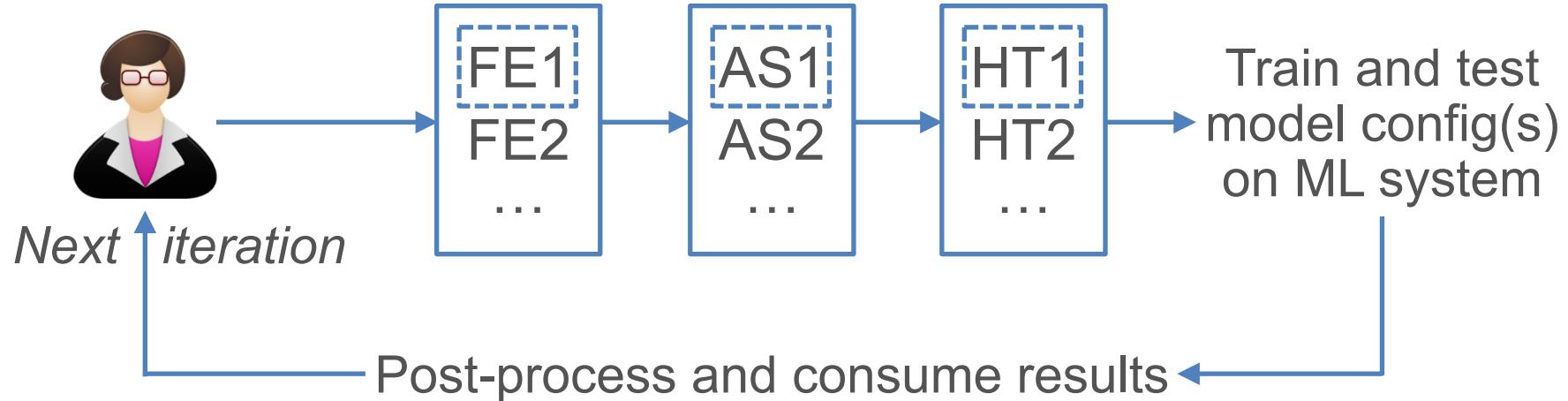
- ❖ Perform **model selection**, i.e., convert prepared ML-ready data to **prediction function(s)** and/or other analytics outputs
- ❖ What makes model building challenging/time-consuming?
  - ❖ **Heterogeneity** of data sources/formats/types
  - ❖ **Configuration complexity** of ML models
  - ❖ Large **scale** of data
  - ❖ **Long training runtimes** of some models
  - ❖ **Pareto optimization on criteria** for application
  - ❖ **Evolution** of data-generating process/application

# Building Stage of ML Lifecycle

- ❖ Perform **model selection**, i.e., convert prepared ML-ready data to **prediction function(s)** and/or other analytics outputs
- ❖ Data scientist / ML engineer must steer 3 key activities that invoke **ML training** and **inference** as sub-routines:
  1. **Feature Engineering (FE)**: How to represent signals appropriately for domain of prediction function?
  2. **Algorithm/Architecture Selection (AS)**: What class of prediction functions (incl. ANN architecture) to use?
  3. **Hyper-parameter Tuning (HT)**: How to improve accuracy/etc. by configuring ML “knobs” better?

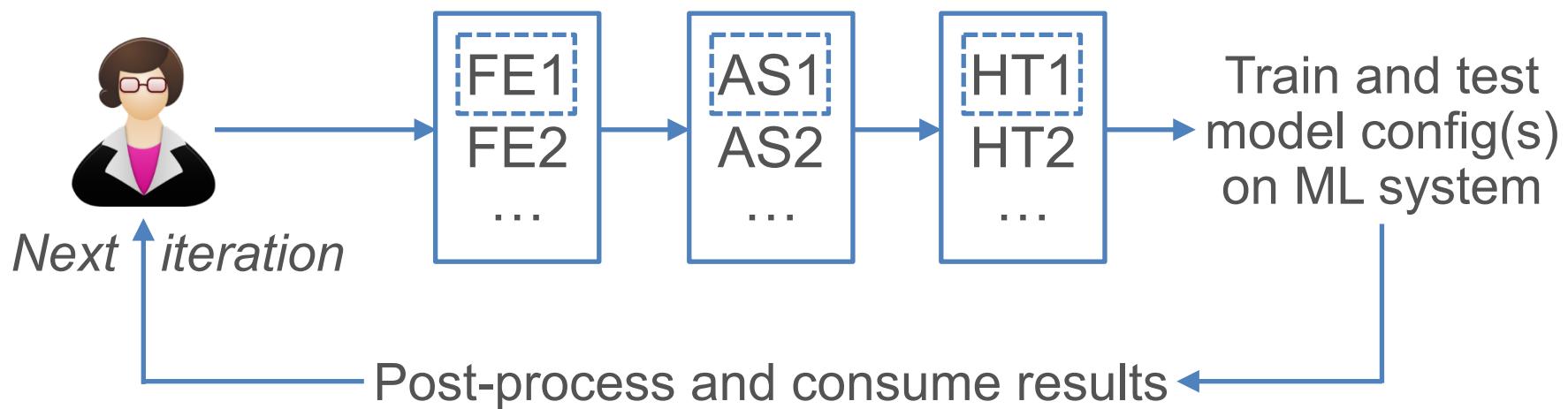
# Model Selection Process

- ❖ Model selection is usually an *iterative exploratory* process with human making decisions on FE, AS, and/or HT
- ❖ Increasingly, automation of some or all parts possible: **AutoML**

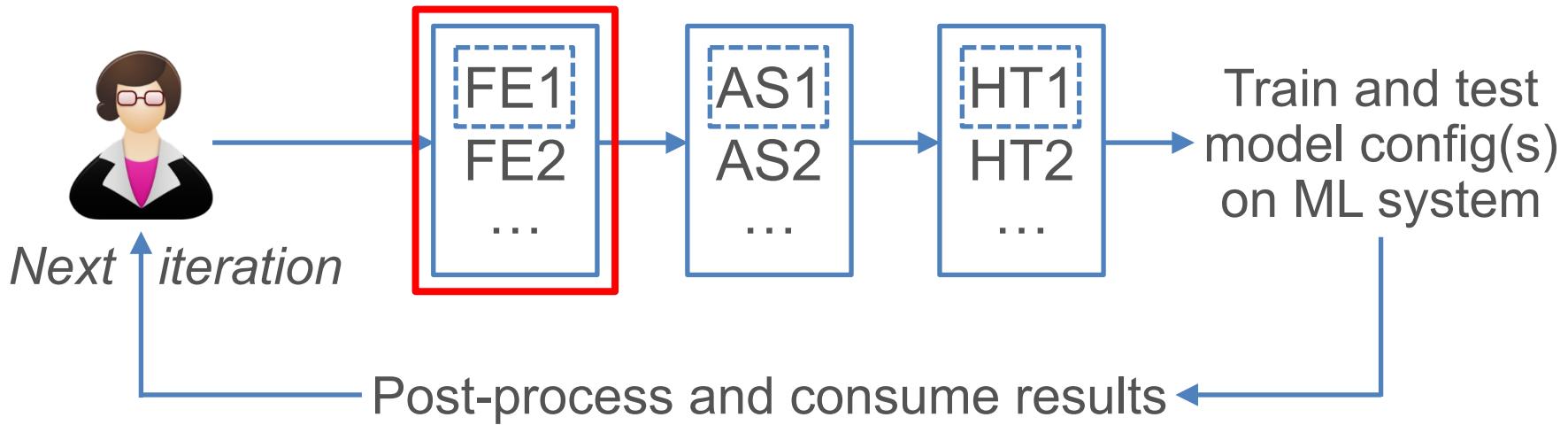


# Model Selection Process

- ❖ Decisions on FE, AS, HT guided by many constraints/metrics: prediction accuracy, data/feature types, interpretability, tool availability, scalability, runtimes, fairness, legal issues, etc.
- ❖ Decisions are typically application-specific and dataset-specific; recall Pareto surfaces and tradeoffs



# Feature Engineering



# Feature Engineering

- ❖ Converting prepared data into a *feature vector representation* for ML training and inference
  - ❖ Aka feature extraction, representation extraction, etc.
- ❖ Umbrella term for many tasks dep. on type of ML model trained:
  1. Recoding and value conversions
  2. Joins and/or aggregates
  3. Feature interactions
  4. Feature selection
  5. Dimensionality reduction
  6. Temporal feature extraction
  7. Textual feature extraction and embeddings
  8. Learned feature extraction in deep learning

# 1. Recoding and value conversions

- ❖ Common on relational/tabular data
- ❖ Typically needs some *global column stats* + code to *reconvert* each tuple (example's feature values)

UserID	State	Date	Upvotes	Comment	Label
143	CA	4/3/19	1539	“This restaurant is overrated”	-
337	NY	11/7/19	5020	“Not too bad!”	+
98	WI	2/8/20	402	“Pretty rad”	+
...	...	...	...	...	...

## Example:

Decision trees can use categorical features directly but GLMs support only numeric features; need **one-hot encoded** 0/1 vector

**Scaling global stats:** “SELECT DISTINCT State”?

**Reconversion:** Tuple-level function to look up domain hash table

# 1. Recoding and value conversions

- ❖ Common on relational/tabular data
- ❖ Typically needs some *global column stats* + code to *reconvert* each tuple (example's feature values)

UserID	State	Date	Upvotes	Comment	Label
143	CA	4/3/19	1539	"This restaurant is overrated"	-
337	NY	11/7/19	5020	"Not too bad!"	+
98	WI	2/8/20	402	"Pretty rad"	+
...	...	...	...	...	...

**Example:**

GLMs and ANNs need **whitening** of numeric features; dense: subtract mean and divide by stdev; sparse: divide by max-min

**Scaling global stats:** How to scale mean/stdev/max/min?

**Reconversion:** Tuple-level function to modify number using stats

# 1. Recoding and value conversions

- ❖ Common on relational/tabular data
- ❖ Typically needs some *global column stats* + code to *reconvert* each tuple (example's feature values)

UserID	State	Date	Upvotes	Comment	Label
143	CA	4/3/19	1539	"This restaurant is overrated"	-
337	NY	11/7/19	5020	"Not too bad!"	+
98	WI	2/8/20	402	"Pretty rad"	+
...	...	...	...	...	...

## Example:

Some models like Bayesian Networks or Markov Logic Networks benefit from (or even need) **binning/discretization** of numerics

**Scaling global stats:** How to scale histogram computations?

**Reconversion:** Tuple-level function to convert number to bin ID

# 2. Joins and Aggregates

- ❖ Common on relational/tabular data
- ❖ Most real-world relational datasets are multi-table; require key-foreign key joins, aggregation-and-key-key-joins, etc.

UserID	Age	Name	UserID	State	Date	Upvotes	Comment	Label
304	40	...	143	CA	...	...	...	-
23	25	...	337	NY	...	...	...	+
143	33	...	143	CA	...	...	...	+
...	...	...	...	...	...	...	...	...

## Example:

Join tables on UserID; concatenate user's info. as extra features!

What kind of join is this? How to scale this computation?

# 2. Joins and Aggregates

- ❖ Common on relational/tabular data
- ❖ Most real-world relational datasets are multi-table; require key-foreign key joins, aggregation-and-key-key-joins, etc.

UserID	State	Date	Upvotes	Comment	Label
143	CA	...	...	...	-
337	NY	...	...	...	+
143	CA	...	...	...	+
...	...	...	...	...	...

## Example:

Join table with itself on UserID to count #reviews and avg #upvotes for each user in a new temp. table and join that to get more features!

What kind of computation is this? How to scale it?

# 3. Feature Interactions

- ❖ Sometimes used on relational/tabular data, especially for high-bias models like GLMs
- ❖ Pairwise is common; ternary is not unheard of

F1	F2	F3	Label
3	2	...	-
4	20	...	+
5	10	...	+
...	...	...	...

F1	F2	F3	F11	F12	F13	F22	F23	F33	Label
3	2	...	9	6	...	4	...	...	-
4	20	...	16	80	...	400	...	...	+
5	10	...	25	50	...	100	...	...	+
...	...	...	...	...	...	...	...	...	...

- ❖ No global stats, just a tuple-level function
- ❖ **NB:** Popularity of this has reduced due to kernel SVMs; but so-called “factorization machines” still need this

# 4. Feature Selection

- ❖ Sometimes used on relational/tabular data
- ❖ **Basic Idea:** Instead of using whole feature set, use a subset

UserID	State	Date	Upvotes	Comment	Label
...	...	...	...	...	...

State	Upvotes	Comment	Label
...	...	...	...

...

- ❖ Formulated as a *discrete optimization* problem
  - ❖ NP-Hard in #features in general
  - ❖ Many heuristics exist in ML/data mining; typically rely on some *information theoretic criteria*
  - ❖ Typically scaled as “outer loops” over training/inference
- ❖ Some ML users also prefer human-in-the-loop approach

# 5. Dimensionality Reduction

- ❖ Often used on relational/structured/tabular data
- ❖ **Basic Idea:** Transforms features to a different latent space
- ❖ **Examples:** PCA, SVD, LDA, Matrix factorization

UserID	State	Date	Upvotes	Comment	Label
...	...	...	...	...	...

F1	F2	F3	Label
0.3	4.2	-29.2	...

**Q:** How is this different from “feature selection”?

- ❖ Feat. sel. *preserves* semantics of each feature but dim. red. typically does not—combines features in “nonsensical” ways
- ❖ Scaling this is non-trivial! Similar to scaling individual ML training algorithms (later)

# 6. Temporal Feature Extraction

- ❖ Many relational/tabular data have time/date
- ❖ Per-example reconversion to extract numerics/categoricals
- ❖ Sometimes global stats needed to calibrate time
- ❖ Complex temporal features studied in *time series mining*

UserID	State	Date	Upvotes	Comment	Label
143	CA	4/3/19	1539	“This restaurant is overrated”	-
337	NY	11/7/19	5020	“Not too bad!”	+
98	WI	2/8/20	402	“Pretty rad”	+
...	...	...	...	...	...

## Example:

Most classifiers cannot use Date directly; extract month (categorical), year (categorical?), day? (categorical), etc.

**Reconversion:** Tuple-level function to extract numbers/categories

# 7. Textual Feature Extraction

- ❖ Many relational/tabular data have text columns; in NLP, whole example is often just text
- ❖ Most classifiers cannot process text/strings directly
- ❖ Extracting numerics from text studied in *text mining*

...	Comment	Label
...	"This restaurant is sucks"	-
...	"Good good!"	+
...	"Pretty rad"	+
...	...	...

...	sucks	good	...	Label
...	1	0	...	-
...	0	2	...	+
...	0	0	...	+
...	...	...	...	...

**Example:**

**Bag-of-words** features: count number of times each word in a given vocabulary arises; need to know vocabulary first

**Scaling global stats:** How to get vocabulary?

**Reconversion:** Tuple-level function to count words; look up index

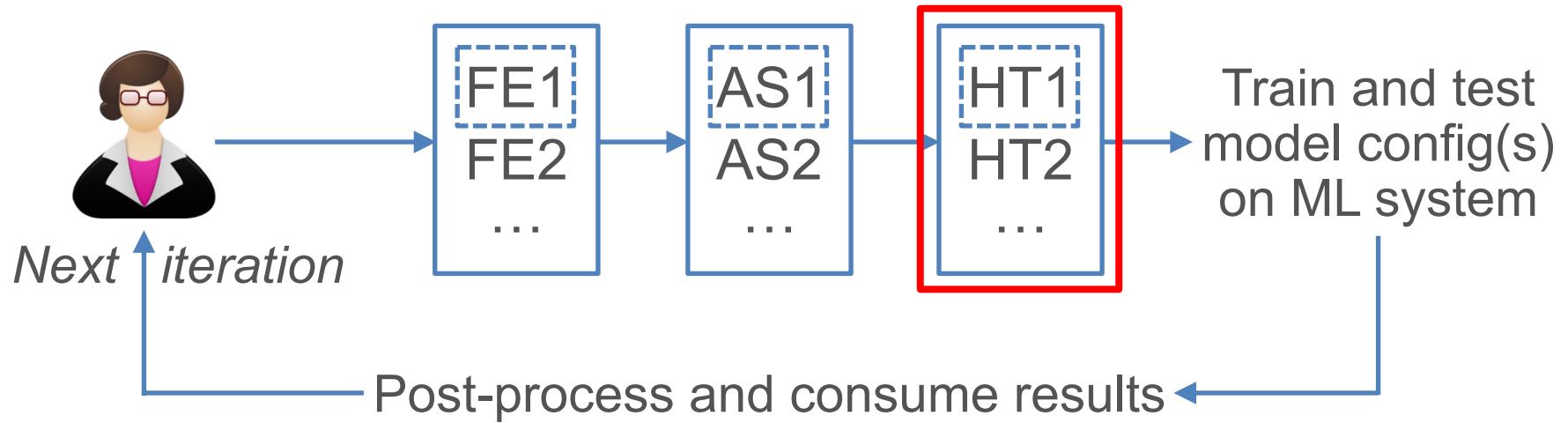
# 7. Textual Feature Extraction

- ❖ **Knowledge Base-based:** Domain-specific knowledge bases like entity dictionaries (e.g., celebrity or chemical names) help extract domain-specific features
- ❖ **Embedding-based:**
  - ❖ Numeric vector for a text token; popular in NLP
  - ❖ Offline training of function from string to numeric vector in self-supervised way on large text corpus (e.g., Wikipedia); embedding dimensionality is a hyper-parameter
  - ❖ *Pre-trained* word embeddings (Word2Vec and GloVe) and sentence embeddings (Doc2Vec) available off-the-shelf; to scale, just use a tuple-level conversion function

# 8. Learned Feature Extraction in DL

- ❖ A big win of DL is no manual feature eng. on *unstructured* data
  - ❖ **NB:** DL is *not* common on struct./tabular data!
- ❖ DL is very *versatile*: almost *any data type* as input and/or output:
  - ❖ Convolutional NNs (CNNs) over image tensors
  - ❖ Recurrent NNs (RNNs) and Transformers over text
  - ❖ Graph NNs (GNNs) over graph-structured data
- ❖ Neural architecture specifies how to extract and transform features internally with weights that are learned
- ❖ **Software 2.0:** Buzzword for such “learned feature extraction” programs vs old hand-crafted feature engineering

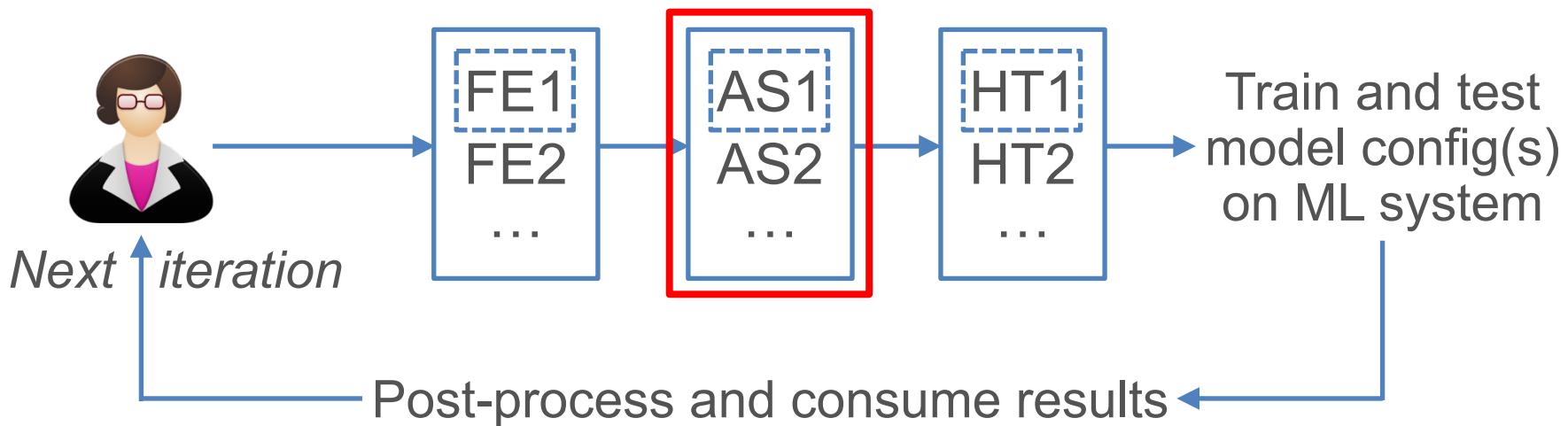
# Hyper-Parameter Tuning



# Hyper-Parameter Tuning

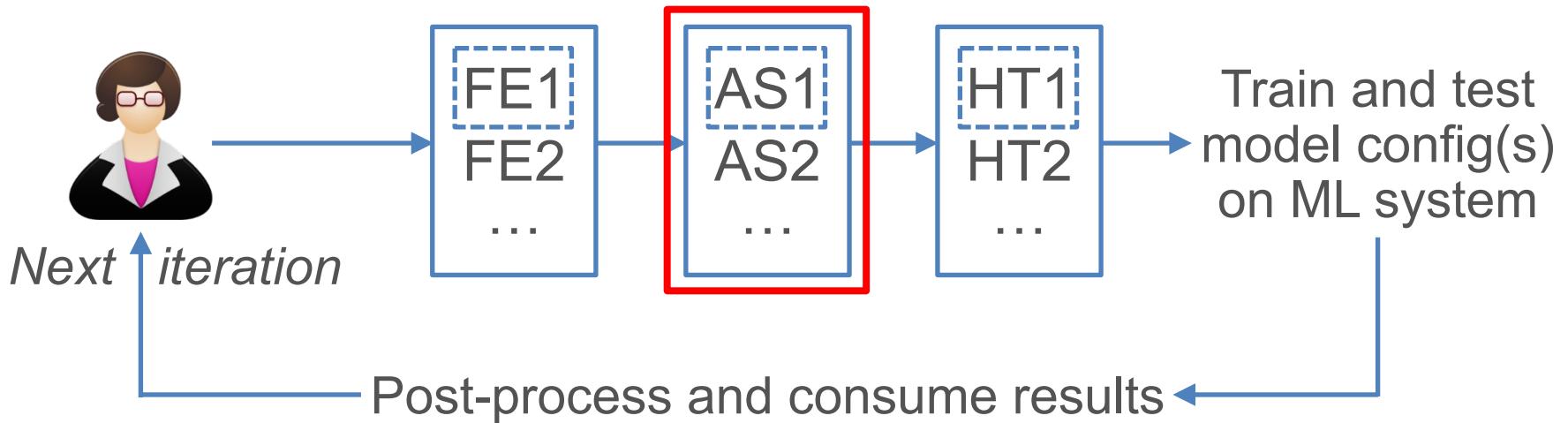
- ❖ **Hyper-parameters:** Knobs for an ML model or training algorithm to control *bias-variance* tradeoff in a dataset-specific manner to make learning effective
- ❖ **Examples:**
  - ❖ GLMs: L1 or L2 *regularizer* to constrain weights
  - ❖ All gradient methods: *learning rate*
  - ❖ Mini-batch SGD: *batch size*
- ❖ HT is an “outer loop” around training/inference
- ❖ Most common approach: **grid search**; pick set of values for each hyper-parameter and take cartesian product
- ❖ Also common: **random search** to subsample from grid
- ❖ Complex AutoML heuristics exist too for HT, e.g., HyperOpt

# Algorithm Selection



- ❖ Not much to say; ML user typically picks models/algorithms ab initio in **“classical” ML** (non-DL)
- ❖ Best practice: first train simple models (log. reg.) as baselines; then try complex models (XGBoost)
- ❖ **Ensembles:** Build diverse models and aggregate predictions

# Architecture Selection in DL



- ❖ More critical in DL; neural arch. is **inductive bias** in classical ML parlance; controls feature learning and bias-variance tradeoff
- ❖ Some applications: Many off-the-shelf pre-trained DL models to do “transfer learning,” e.g., HuggingFace Models
- ❖ Other applications: Swap pain of hand-crafted feature eng. for pain of neural arch. eng.! :)

# Automated Model Selection / AutoML

**Q:** Can we automate the whole model selection process?

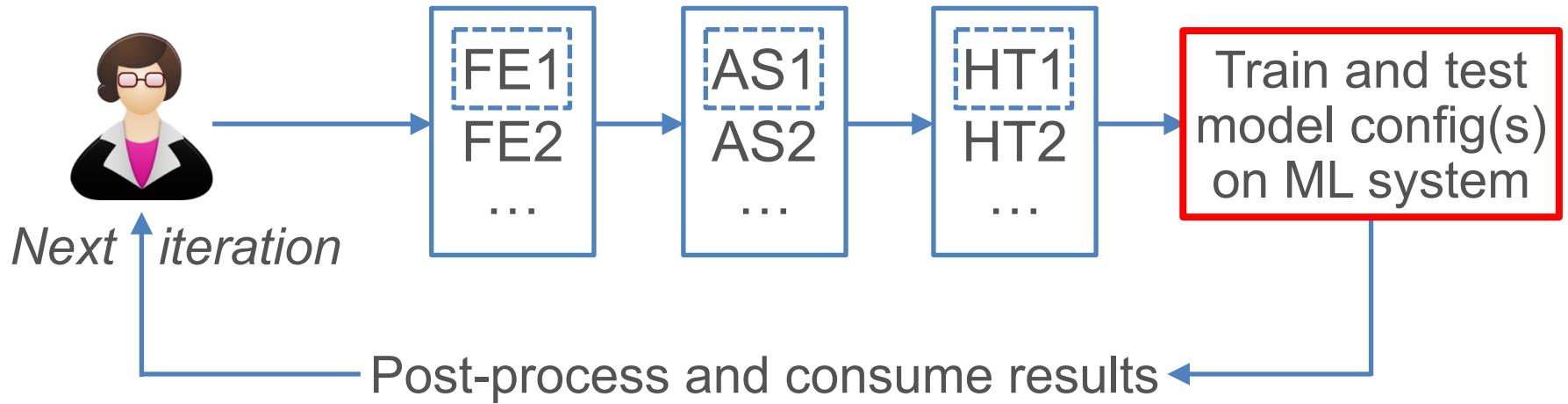
- ❖ It depends. HT and most of FE already automated mostly in practice; (neural) AS is often application-dictated
- ❖ AutoML tools/systems now aim to reduce data scientist's work; or even replace them?! ;)



- ❖ **Pros:** Ease of use; lower human cost; easier to audit; improves ML accessibility
- ❖ **Cons:** Higher resource cost; less user control; may waste domain knowledge
- ❖ Pareto-optima; hybrids possible

**But:** The Data Sourcing stage is still very hard to automate!

# Scalable ML Training and Inference



# Major ML Model Families/Types

**Generalized Linear Models** (GLMs); from statistics

**Bayesian Networks**; inspired by causal reasoning

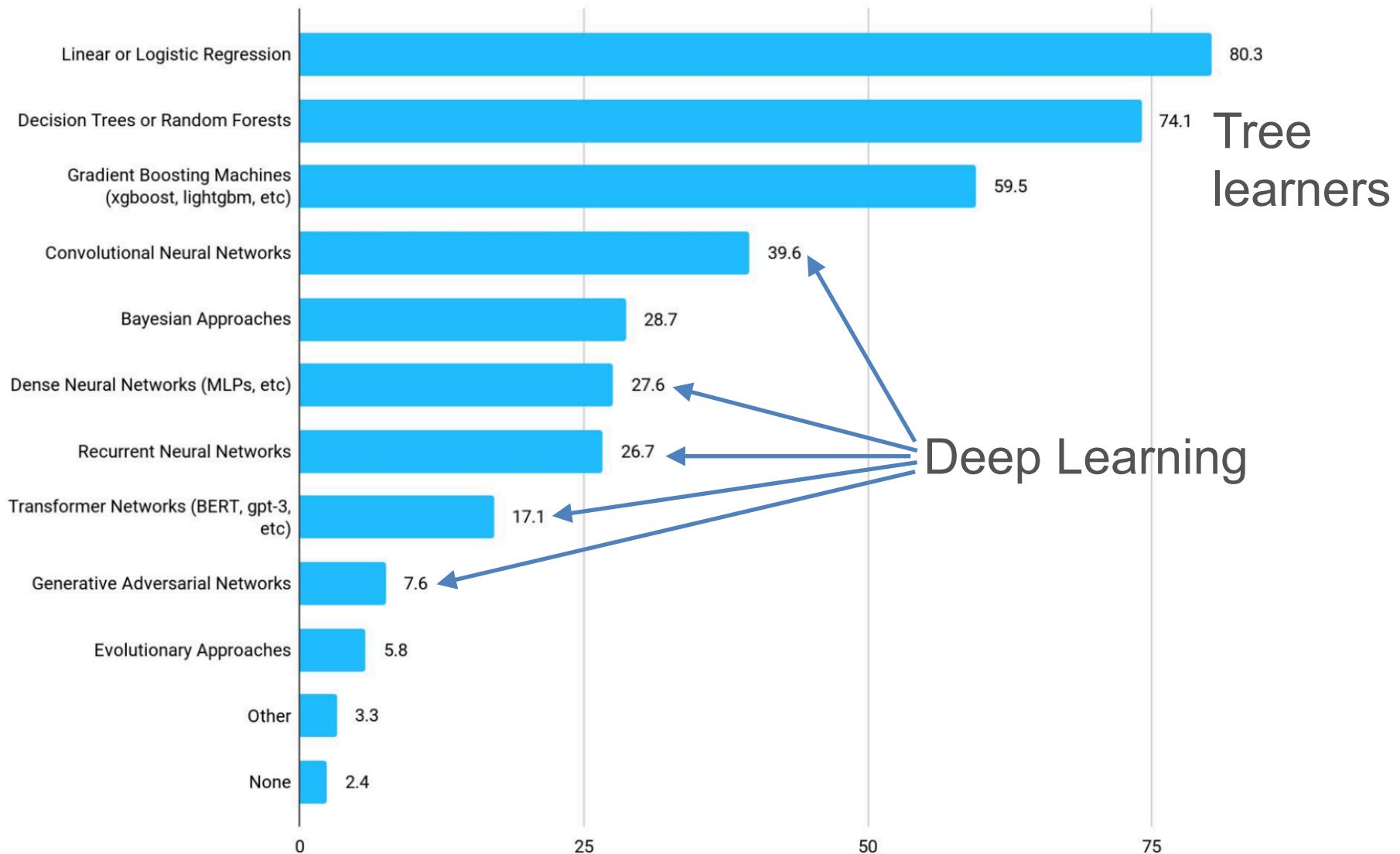
**Decision Tree-based**: CART, Random Forest, Gradient-Boosted Trees (GBT), etc.; inspired by symbolic logic

**Support Vector Machines** (SVMs); inspired by psychology

**Artificial Neural Networks** (ANNs): Multi-Layer Perceptrons (MLPs), Convolutional NNs (CNNs), Recurrent NNs (RNNs), Transformers, etc.; inspired by brain neuroscience

**Unsupervised**: Clustering (e.g., K-Means), Matrix Factorization, Latent Dirichlet Allocation (LDA), etc.

# ML Models in Kaggle 2021 Survey



# Scalable ML Training Systems

- ❖ Scaling ML training is involved and model type-dependent
- ❖ Orthogonal Dimensions of Categorization:
  1. **Scalability:** In-memory libraries vs Scalable ML system  
(works on larger-than-memory datasets)
  2. **Target Workloads:** General ML library vs Decision tree-oriented vs Deep learning, etc.
  3. **Implementation Reuse:** Layered on top of scalable data system vs Custom from-scratch framework

**Ad:** Take CSE 234 in Fall'22 for more on scalable ML systems <sup>29</sup>

# Major Existing ML Systems

## General ML libraries:

In-memory:



Disk-based files:



Layered on RDBMS/Spark:



Cloud-native:



Azure Machine Learning



Amazon SageMaker

“AutoML” platforms:



DataRobot



Decision tree-oriented:



Microsoft  
LightGBM

Deep learning-oriented:



TensorFlow



# Scalable ML Inference

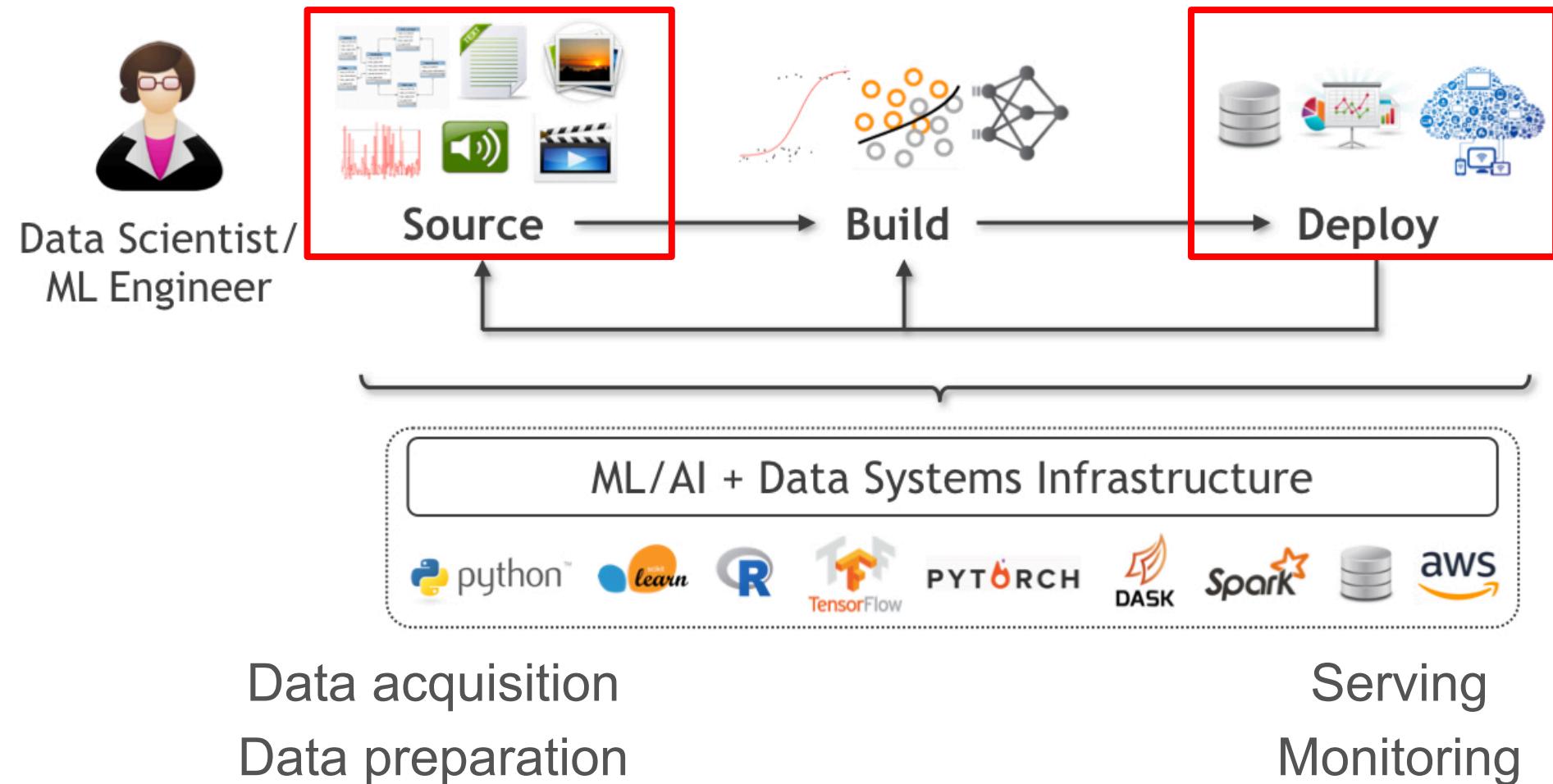
- ❖ A trained/learned ML model is just a prediction function:

$$f : \mathcal{D}_X \rightarrow \mathcal{D}_Y$$

*Q: Given large dataset of examples, how to scale inference?*

- ❖ Assumption 1: An example fits entirely in DRAM
- ❖ Assumption 2:  $f$  fits entirely in DRAM
- ❖ If both hold, trivial access pattern: single filescan, apply per-tuple function  $f$ , write output. How to do this with MapReduce?
- ❖ If either fails, access pattern becomes more complex and dependent on breaking up internals of  $f$  to stage access to data for partial computations

# The Lifecycle of ML-based Analytics



**Ad:** Take CSE 234 in Fall'22 for more on Source and Deploy stages

Week	Topic and Papers	Slides, Videos; Review Forms, Deadlines
0	<b>Introduction, ML Lifecycle Overview, and Basics</b>  Readings: SIGMOD tutorial 1, SIGMOD tutorial 2, Berkeley report	Slides: <a href="#">PDF</a> <a href="#">PPTX</a> <a href="#">Video 1</a> ; <a href="#">Video 2</a> ; <a href="#">Video 3</a>
1-2	<b>Topic 1: Classical ML Training at Scale</b>  For review: Parameter Server  For review: XGBoost  More readings: <a href="#">MADlib</a> , <a href="#">MLlib</a> , <a href="#">Mahout</a> , <a href="#">GraphLab</a> , <a href="#">AWS Sagemaker</a>	Slides: <a href="#">PDF</a> <a href="#">PPTX</a> <a href="#">Video 1</a> ; <a href="#">Video 2</a>  <a href="#">Review 1 Form</a> ; due 10/6 <a href="#">Review 2 Form</a> ; due 10/13  -
1	<b>No class on 10/8</b>	-
3	<b>Topic 2: Deep Learning Systems</b>  For review: TensorFlow (Talk slides)  More readings: Horovod, Distributed PyTorch, TVM	Slides: <a href="#">PDF</a> <a href="#">PPTX</a> <a href="#">Video 1</a> ; <a href="#">Video 2</a> ; <a href="#">Video 3</a>  <a href="#">Review 3 Form</a> ; due 10/20  -
4-5	<b>Topic 3: Feature Engineering and Model Selection Systems</b>  For review: Cerebro  More readings: <a href="#">MSMS</a> , <a href="#">Hyperband</a> , <a href="#">ASHA</a> , <a href="#">Vizier</a> , <a href="#">Columbus</a> , <a href="#">Vista</a>	Slides: <a href="#">PDF</a> <a href="#">PPTX</a> <a href="#">Video 1</a> <a href="#">Video 2</a>  <a href="#">Review 4 Form</a> ; due 10/27  -
5	<b>Review Session 1 on 11/3 (tentative)</b>	Slides: <a href="#">PDF</a>
5	<b>Exam 1 on 11/5</b>	-
6	<b>Topic 4: Data Sourcing and Organization for ML</b>  For review: TFDV  More readings: <a href="#">Deequ</a> , <a href="#">Snorkel</a> , <a href="#">Ground</a> , <a href="#">SortingHat</a> , <a href="#">Hamlet</a>	Slides: <a href="#">PDF</a> <a href="#">PPTX</a> <a href="#">Video 1</a> ; <a href="#">Video 2</a> ; <a href="#">Video 3</a>  <a href="#">Review 5 Form</a> ; due 11/3  -
7	<b>Guest Lecture by Matei Zaharia (Databricks and Stanford) on MLFlow on 11/17</b>	<a href="#">Video</a> ; <a href="#">Slides</a> <a href="#">PDF</a>
7-9	<b>Topic 5: ML Deployment</b>  For review: Clipper  More readings: <a href="#">TF Serving</a> , <a href="#">Uber PyML</a> , <a href="#">Hummingbird</a> , <a href="#">Federated ML</a>	Slides: <a href="#">PDF</a> <a href="#">PPTX</a> <a href="#">Video 1</a> ; <a href="#">Video 2</a>  <a href="#">Review 6 Form</a> ; due 11/12  -
8	<b>Guest Lecture by Angela Jiang (Determined AI) on Determined DL Platform on 11/24</b>	<a href="#">Video</a> ; <a href="#">Slides</a> <a href="#">PDF</a>
8	<b>Thanksgiving Holiday on 11/26</b>	-
9	<b>Guest Lecture by Joshua Patterson (NVIDIA) on RAPIDS on 12/1</b>	<a href="#">Video</a> ; <a href="#">Slides</a> <a href="#">PDF</a>
9-10	<b>Topic 6: ML Platforms and Feature Stores</b>  For review: ML systems technical debt  For review: TensorFlow Extended  More readings: <a href="#">MLFlow</a> , <a href="#">Michelangelo</a>	Slides: <a href="#">PDF</a> <a href="#">PPTX</a> <a href="#">Video 1</a> ; <a href="#">Video 2</a>  <a href="#">Review 7 Form</a> ; due 11/17  <a href="#">Review 8 Form</a> ; due 12/3  -

Ad: CSE 234/291 from Fall'20 with lecture videos on Youtube

<https://cseweb.ucsd.edu/classes/fa20/cse291-d/schedule.html>

Week	Topic	Textbook Chapters, Additional References	Slides
1	Introduction; Recap of Relational Algebra and SQL	Ch 1, 4, 5.1-5.6	PPTX PDF
1-2	Data Storage; Buffer Management; File Organization	Ch 8, except 8.5.4, Ch 9, except 9.2	PPTX PDF
2	Talk by the TA on <b>Project 1</b> on TBD	-	
3-4	Indexing (B+ Tree; Hash Index)	Ch 10, Ch 11, sections 11.1-11.2 only	PPTX PDF
4	Industry Guest Lecture on Tuesday, 4/20 by Andrew Lamb (Apache Arrow and InfluxDB)	-	Video
4-5	External Sorting	Ch 13	PPTX PDF
5	Talk by the TA on <b>Project 2</b> on TBD	-	
5	Review discussion on TBD	-	
6	Midterm Exam on Tuesday, 5/4	-	-
6-7	Relational Operator Implementations; Query Processing	Ch 12, sections 12.1-12.3, Ch 14	PPTX PDF
7-8	Query Optimization	Ch 12, sections 12.4 - 12.6	PPTX PDF
9	ML for RDBMSs	TBD	PPTX PDF
9	Industry Guest Lecture on Thursday, 5/27 by Andy Pavlo (OtterTune and CMU)	-	Video
10	Parallel DBMSs and Dataflow Systems	Ch 22, till 22.5	PPTX PDF
10	Review session on Thursday, 6/3	-	
11	Final Exam on Tuesday, 6/8	-	-
N/A	Optional: Key-value stores, Graph DBMSs, ML systems	Not in syllabus	PPTX PDF
N/A	Optional: Transaction Management	Not in syllabus	PPTX PDF

**Ad:** Take CSE 132C in Spring'22 or Fall'22 for more on RDBMSs, parallel data systems, and more advanced DBMS topics

Short (12min) overview: <https://www.youtube.com/watch?v=hotGs0afSWc> <sup>34</sup>

# DSC 102 will get you thinking about the fundamentals of scalable analytics systems

1. “**Systems**”: What resources does a computer have?  
How to store and efficiently compute over large data?  
What is cloud?
2. “**Scalability**”: How to scale and parallelize data-intensive computations?
3. For “**Analytics**”:
  - 3.1. **Source**: Data acquisition & preparation for ML
  - 3.2. **Build**: Model selection & deep learning systems
  - 3.3. **Deploying** ML models
4. Hands-on experience with scalable analytics tools

Week	Topic	Textbook Chapters, Additional References	Slides
1	Introduction and Administrivia	-	<a href="#">PDF PPTX</a>
1-2	Basics of Machine Resources: Computer Organization  No class on Mon, Jan 17 (MLK Day holiday)	Ch. 1, 2.1-2.3, 2.12, 4.1, and 5.1-5.5 of CompOrg Book	<a href="#">PDF PPTX</a>
3-4	Basics of Machine Resources: 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	<a href="#">PDF PPTX</a>
5	Basics of Cloud Computing	-	<a href="#">PDF PPTX</a>
5-6	Parallel and Scalable Data Processing: Parallelism Basics	Ch. 9.4, 12.2, 14.1.1, 14.6, 22.1-22.3, 22.4.1, 22.8 of Cow Book; Ch. 5, 6.1, 6.3, 6.4 of MLSys Book	<a href="#">PDF PPTX</a>
6	Review for Midterm Exam on Tue, Feb 8, 2-3pm PT	-	-
6	Midterm Exam on Wed, Feb 9	-	-
7	Parallel and Scalable Data Processing: Scalable Data Access  No class on Mon, Feb 21 (President's Day holiday)	Ch. 9.4, 12.2, 14.1.1, 14.6, 22.1-22.3, 22.4.1, 22.8 of Cow Book; Ch. 5, 6.1, 6.3, 6.4 of MLSys Book	<a href="#">PDF PPTX</a>
7-8	Parallel and Scalable Data Processing: Data Parallelism	Ch. 9.4, 12.2, 14.1.1, 14.6, 22.1-22.3, 22.4.1, 22.8 of Cow Book; Ch. 5, 6.1, 6.3, 6.4 of MLSys Book	<a href="#">PDF PPTX</a>
8	Industry Guest Lecture on Wed, Feb 23 by Lavanya Shukla (Weights & Biases)	-	
9	Dataflow Systems	Ch. 2.2 of MLSys Book	<a href="#">PDF PPTX</a>
10	ML Model Building Systems	Ch. 8-8.4 of MLSys Book	<a href="#">PDF PPTX</a>
10	Industry Guest Lecture on Wed, Mar 9 by Sarah Catanzaro (Amplify Partners)	-	
10	Review for Final Exam on Fri, Mar 11, 4-5pm PT	-	-
11	Final Exam on Fri, Mar 18	-	-

Thank you for taking DSC 102.

Please make sure to submit your CAPE if you have not done so already.

All the best for final exams week!



P A R T N E R S

## MLOops:

A review of the ML stack in industry and the problems that ML stakeholders and tool builders must solve to drive AI adoption

---

**SARAH CATANZARO**

*General Partner*  
Amplify Partners



**who am I?**

# Who am I and why am I here?

# Agenda

## A SNAPSHOT

**01**

Snapshot of ML in industry (beyond FAANG/MANGA)

## THE ML STACK

**02**

A taxonomy of applications, platforms and infrastructure

## KEY CHALLENGES

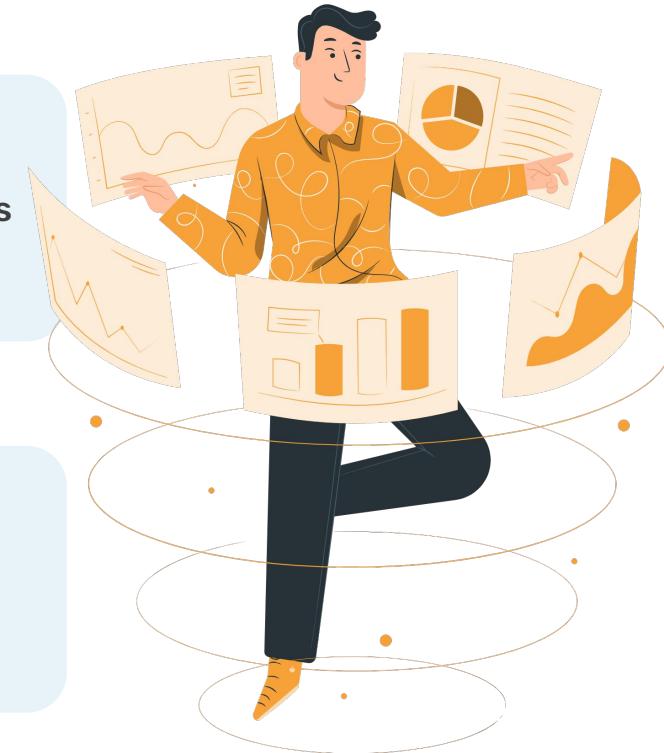
**03**

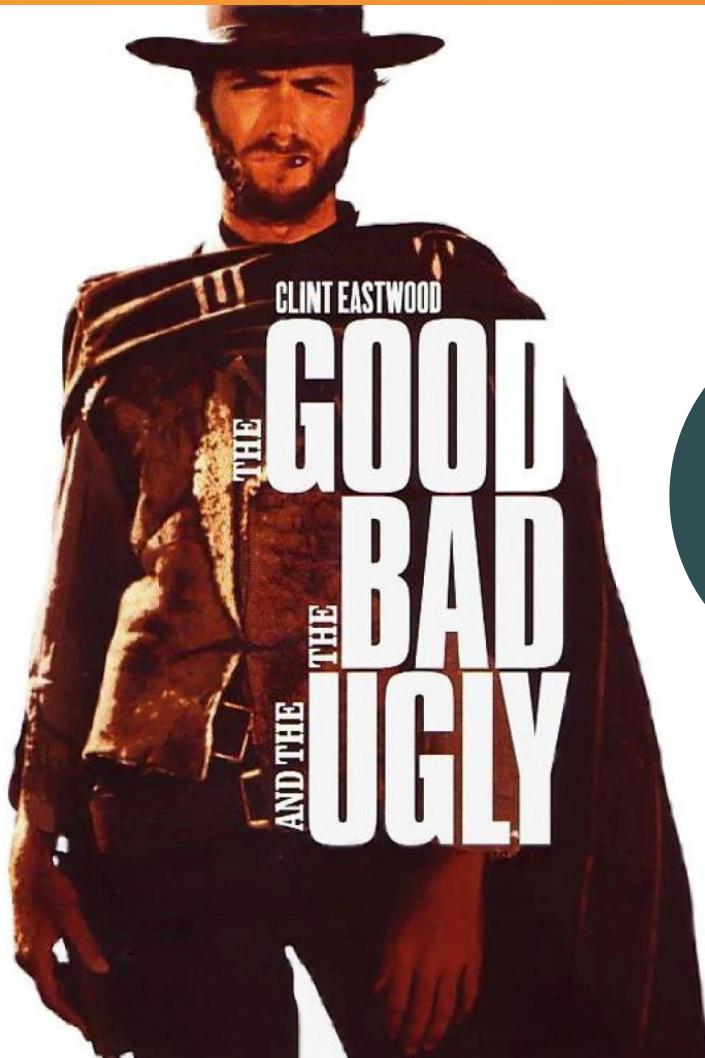
Key challenges ML practitioners face

## KEY CHALLENGES

**04**

Key challenges ML tool builders face





A snapshot of ML  
in industry

# A Snapshot of ML in Industry

High ROI use cases are emerging among mid-market and public companies



ML-driven SaaS applications

The image displays two side-by-side screenshots of software-as-a-service (SaaS) applications that utilize machine learning (ML) for their core functionality. The top screenshot shows a user interface for a service like 'Employee Recognition' or 'Facial Recognition'. It features a grid of numerous small, square profile pictures of people. Below this grid, there is a larger, more detailed view of a single person's face, possibly for identification or analysis. The bottom screenshot shows another SaaS application, likely related to business intelligence or predictive analytics. This interface includes a sidebar with navigation options, a main area with several large, vertically stacked data cards, and a bottom section with a timeline or sequence of events. The overall theme of the image is how ML is being integrated into various industry-specific SaaS tools to enhance efficiency and decision-making.

Compare Similar Items

Item	Price	Rating	Reviews	Action
Wayfair Basics Metal Bed Frame by Wayfair Basics®	\$48.99	4.5	8647	Add
Hiett Platform Bed Frame by Alwyn Home	\$76.99	4.5	10904	See Details
Blough Heavy Duty Bed Frame by Alwyn Home	\$85.99	4.5	4672	See Details
Higbee Steel Bed Frame by Alwyn Home	\$78.99	4.5	5116	See Details
Vela Heavy Duty Bed Frame by Alwyn Home	\$69.99	4.5	1651	See Details

## A SNAPSHOT OF ML IN INDUSTRY

**Structured Data**



**Embedding**

(structured/semi-structured data)



**Text and visual data**



## A SNAPSHOT OF THE ML INDUSTRY

**Mid-market and public companies have hired ML practitioners and plan to expand their team, but are still experimenting with different organizational models.**

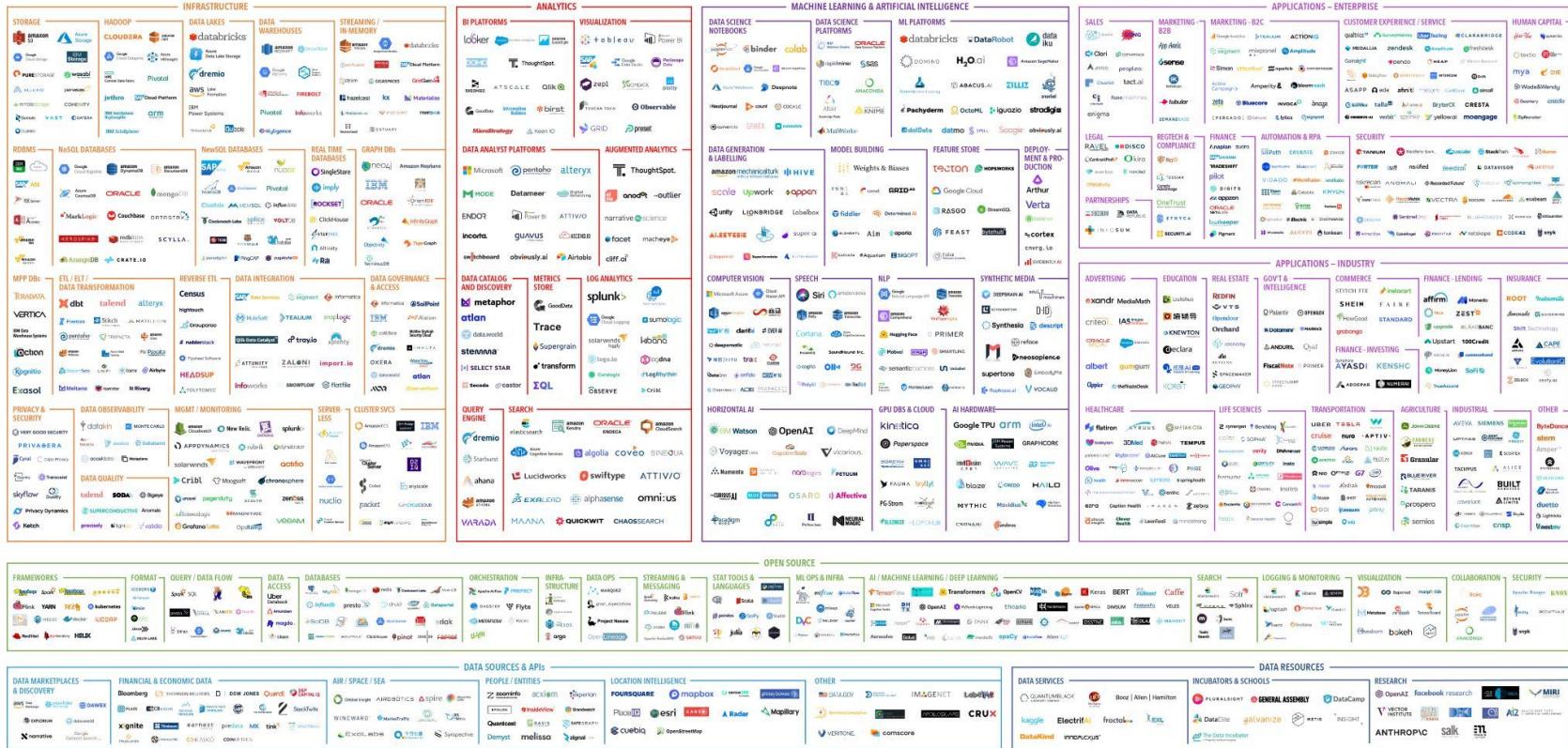
**DATA SCIENTIST AFTER WRITING A FLASK APP**

You know, I'm something  
of a software engineer myself

imgflip.com

# The ML stack; a taxonomy

MACHINE LEARNING, ARTIFICIAL INTELLIGENCE, AND DATA (MAD) LANDSCAPE 2021



Version 1.0 - September 2021

© Matt Turck (@mattturck), John Wu (@john\_d\_wu) & FirstMark (@firstmarkcap)

mattturck.com/data2021

**FIRSTMARK**  
EARLY STAGE VENTURE CAPITAL

Amplify Partners - Proprietary and Confidential

# The ML stack; a taxonomy

## MODEL OPERATIONS

MODEL MONITORING



MODEL OPTIMIZATION & DEPLOYMENT



MODEL COMPLIANCE



CONTINUOUS LEARNING



## BUILD & DEPLOY

DATA SCIENCE FRAMEWORK



EXPERIMENT TRACKING



VERSION CONTROL



AUTO ML



DISTRIBUTED TRAINING



## DATA MANAGEMENT

### DATABASES

CLOUD DATA WAREHOUSE

DATA LAKE

VECTOR DATABASE

### FEATURE MANAGEMENT

FEATURE STORE

FEATURE /METRICS LAYER

### DATA LABELING

LABELING SERVICE

WEAK SUPERVISION

# Data Management

## Databases

### DATA WAREHOUSE



### DATA LAKE



### VECTOR DATABASE

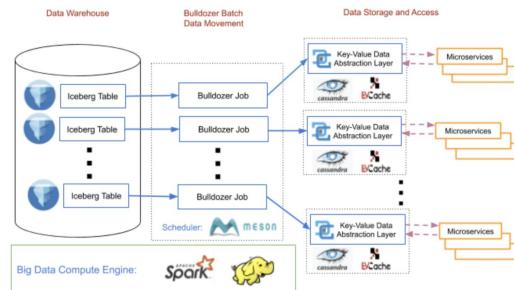


## Feature Management

### FEATURE STORE



### FEATURE LAYER/PROXY



## Data Labeling

### LABELING SERVICES



### WEAK SUPERVISION

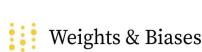


# Build and Deploy

## DATA SCIENCE FRAMEWORK



## EXPERIMENT TRACKING



## VERSION CONTROL



## AUTOML



## DISTRIBUTED TRAINING



# Model Operations

## Model Monitoring



## Model Optimization & Deployment

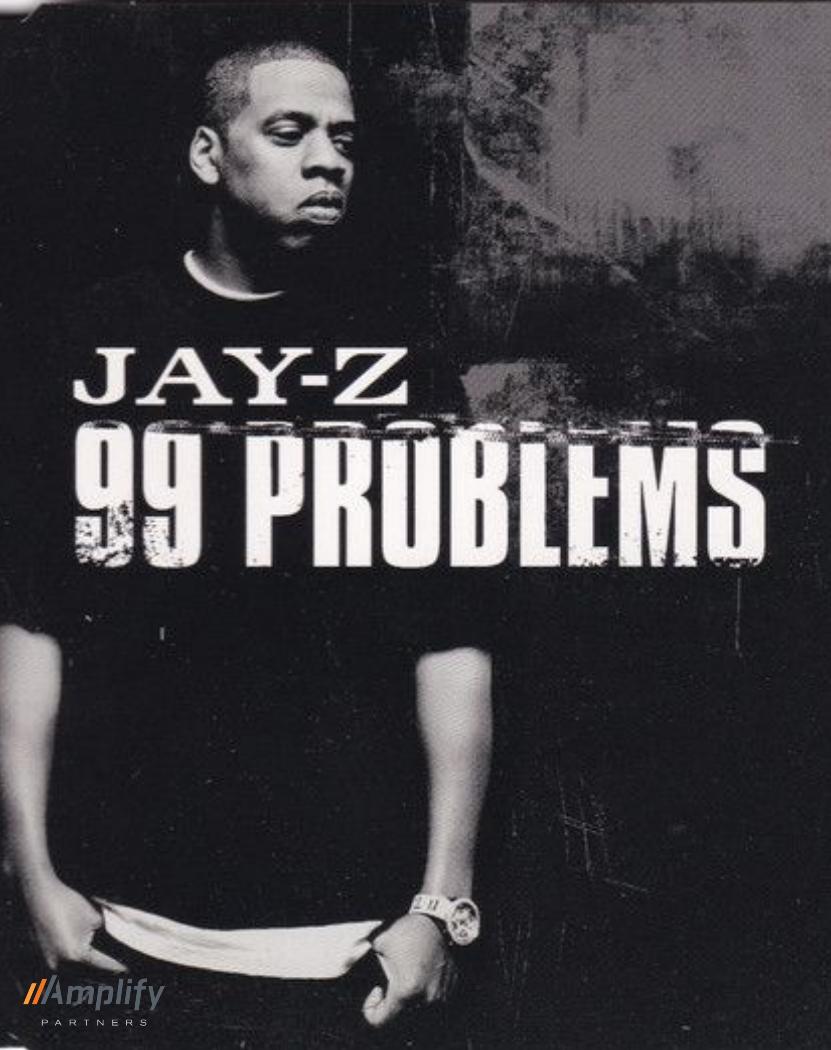


## Model Compliance



## Continuous Learning





# Challenges that ML practitioners face

# SCOPING ML PROJECTS

---

## HOW CAN YOU ANTICIPATE THE:

- Likelihood of success?
- Resources (data, compute, time) needed?

## HOW CAN YOU SCOPE ML EXPERIMENTS TO:

- Avoid expending unnecessary effort on projects that aren't feasible
- Leverage engineering resources effectively

Nice paper

---



GitHub  
Link

---



Written in  
your favourite  
framework

---



Runs smoothly on  
your system  
without error or  
dependency issues

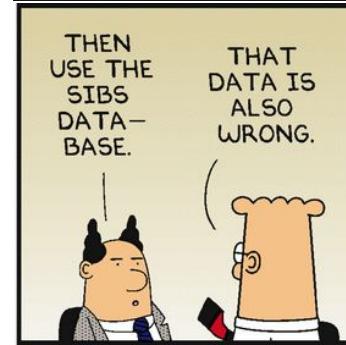
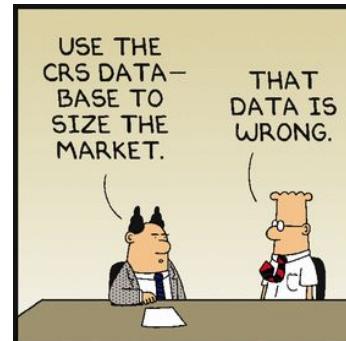
---



# ACCESSING AND ITERATING ON DATASETS

## PRACTITIONERS NEED:

- Access to high quality data
- Tools for data-centric ML
- Unstructured data wrangling

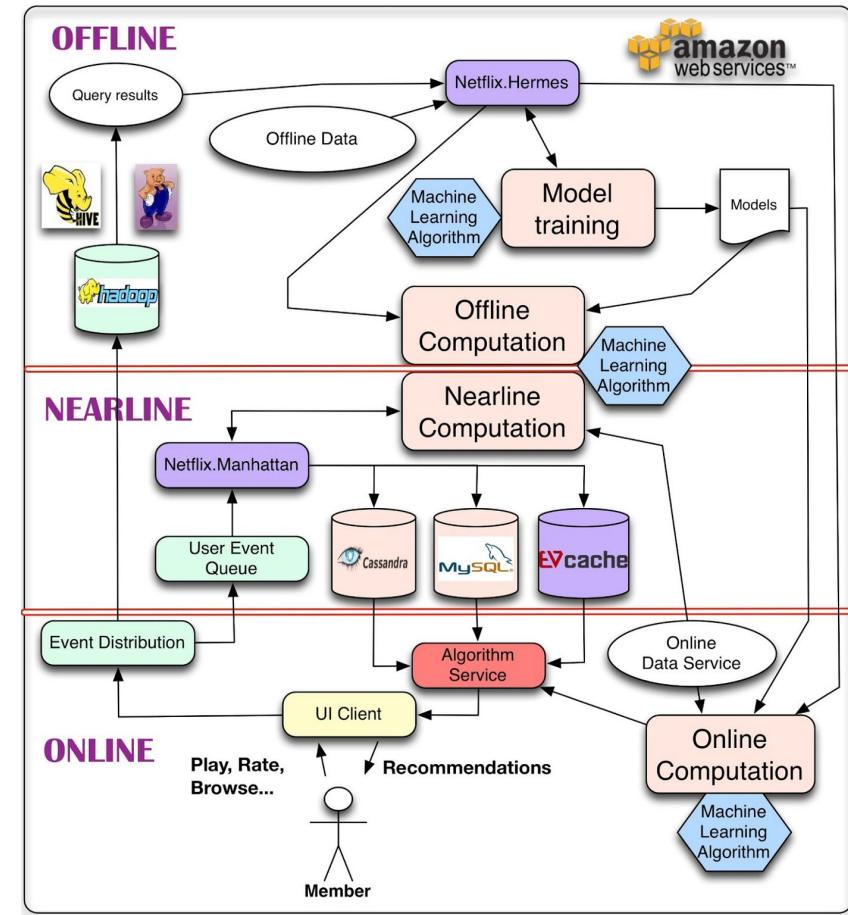


# BUILDING SOFTWARE APPLICATIONS WITH ML

Practitioners must build software applications for which a prediction service (ML model) is one component

## NEED TO INTEGRATE WITH:

- Data management systems
- Other [micro]services (e.g. authentication)
- Developer tools (e.g. CI/CD)



# MANAGING INFRASTRUCTURE

- DS should spend time on modeling and analyses NOT managing infrastructure and environments!





INTRODUCING

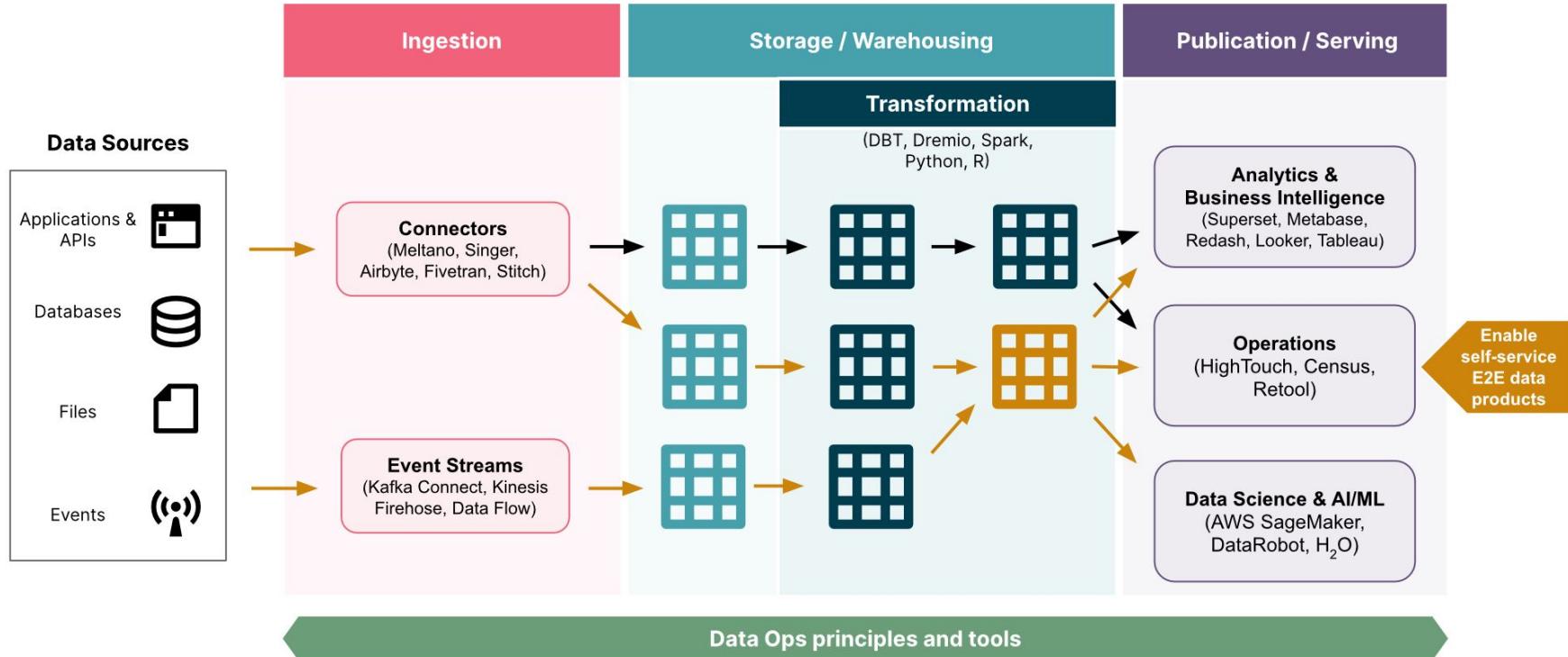
# Modal Labs





# Challenges ML Tool Builders Face

# Comparing the “Modern Data Stack” and ML Stack



**NO DOMINANT DESIGN  
FOR THE ML STACK**



**NO system of record**



**NO best practices or  
standards**



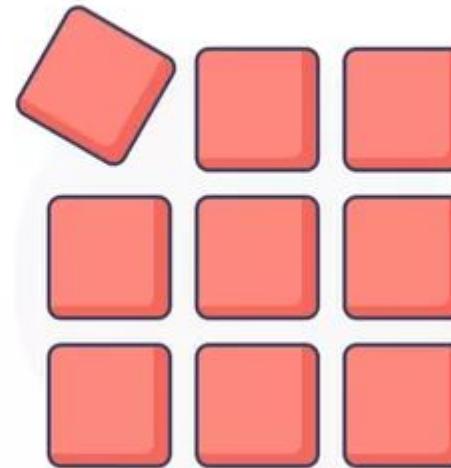
**NO clear roles and  
responsibilities**

# No partnership ecosystem: Why everything becomes an end-to-end platform

**BEST OF BREED  
STACK**



**ALL IN ONE SUITE**



# ONE ML STACK OR MANY?

- Different applications may have different requirements
- Different model architectures may have different requirements

