UC San Diego

DSC 102
Systems for Scalable Analytics

Winter 2022

Arun Kumar

About Myself

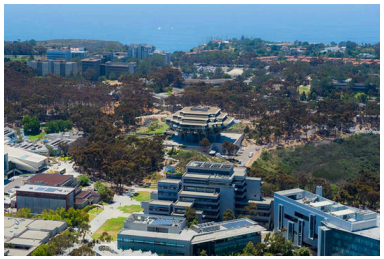
Summers: 110F!

2009: Bachelors in CSE from IIT Madras, India

Winters: —40F!

2009—16: MS and PhD in CS from UW-Madison

PhD thesis area: Data systems for ML workloads



Ahh! :)

2016-: Asst. Prof. at UC San Diego CSE

2019-: + Asst. Prof. at UC San Diego HDSI

2021: Assoc. Prof. at CSE & HDSI

My Current Research

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

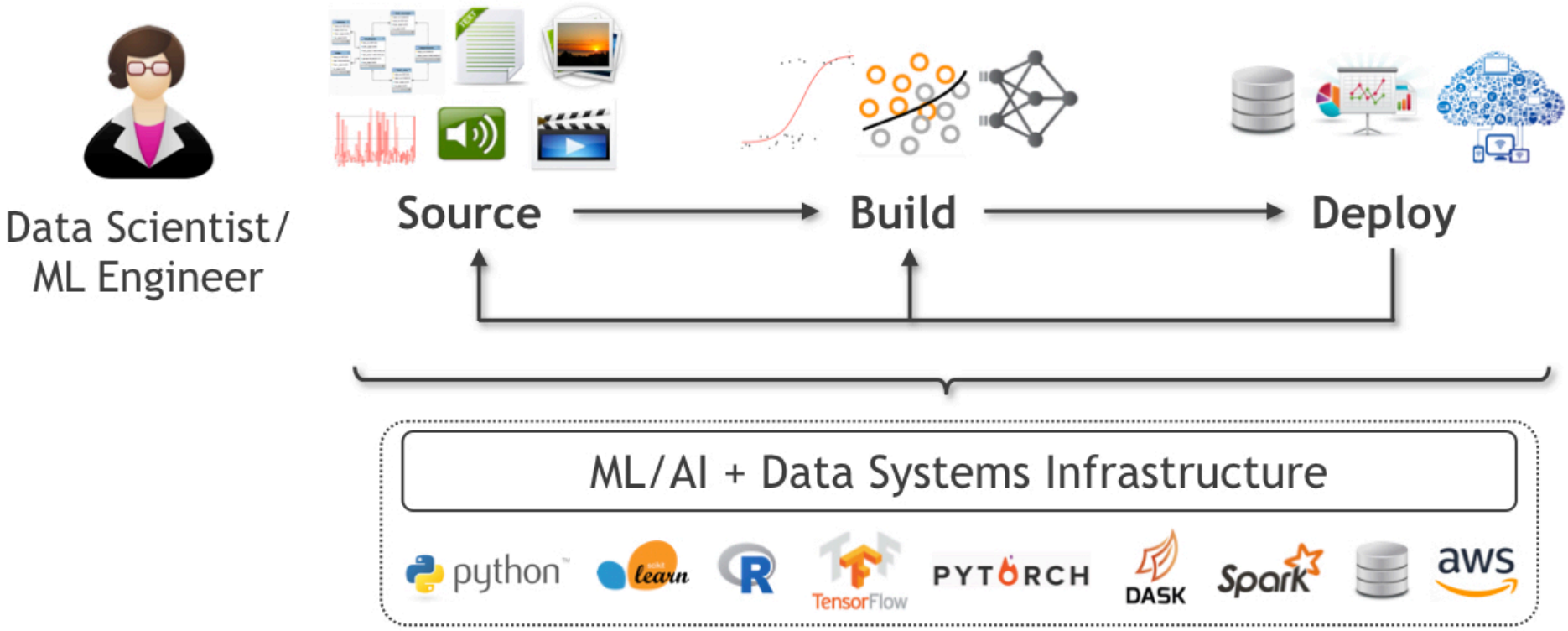
Democratization = **System Efficiency (Lower resource costs)** **+** **Human Efficiency (Higher productivity)**

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

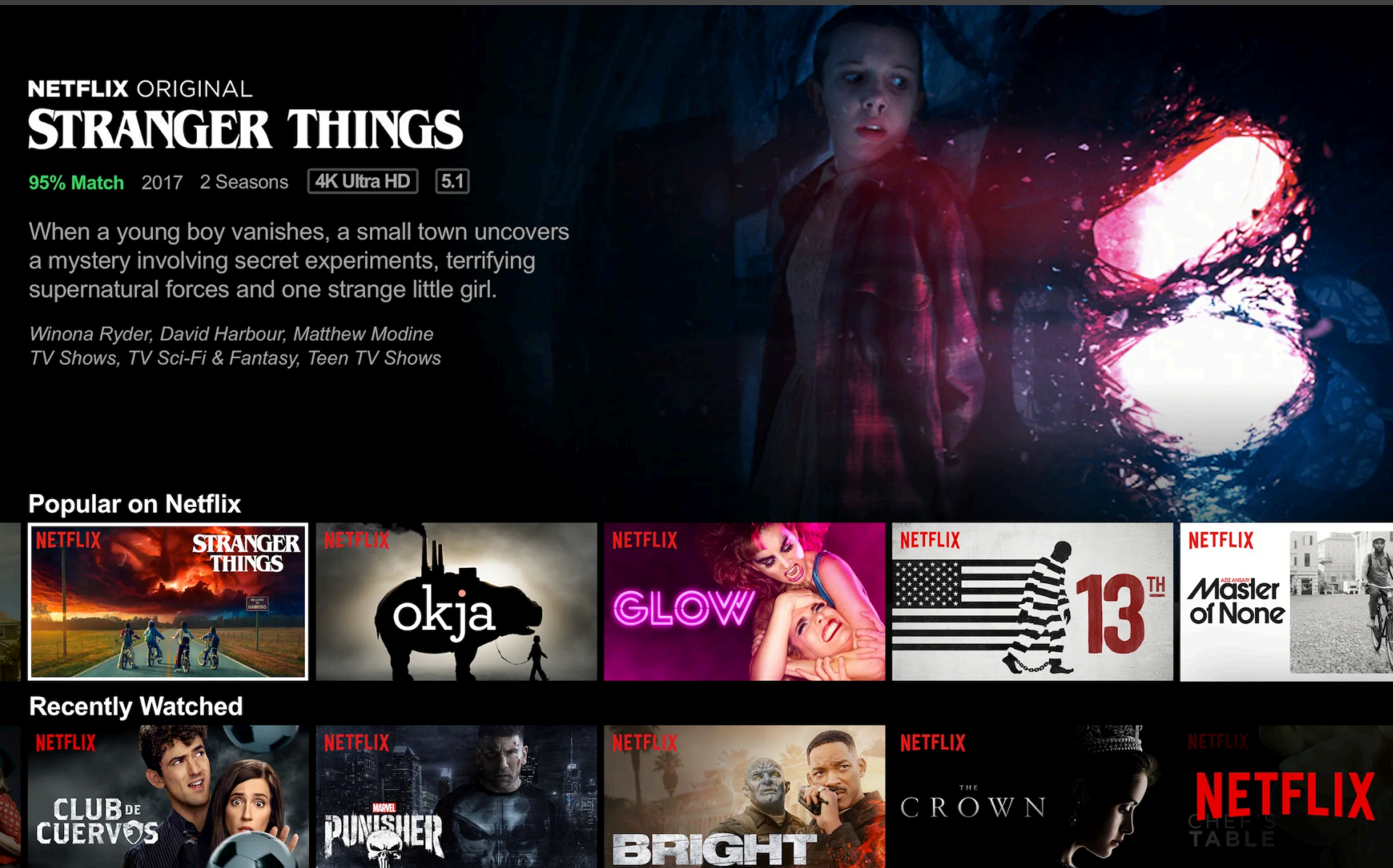


Research Approach: ***Abstract*** key steps + ***Formalize*** computation + ***Automate*** grunt work + ***Optimize*** execution

https://adalabucsd.github.io/

What is this course about? Why take it?

1. Netflix's "spot-on" recommendations

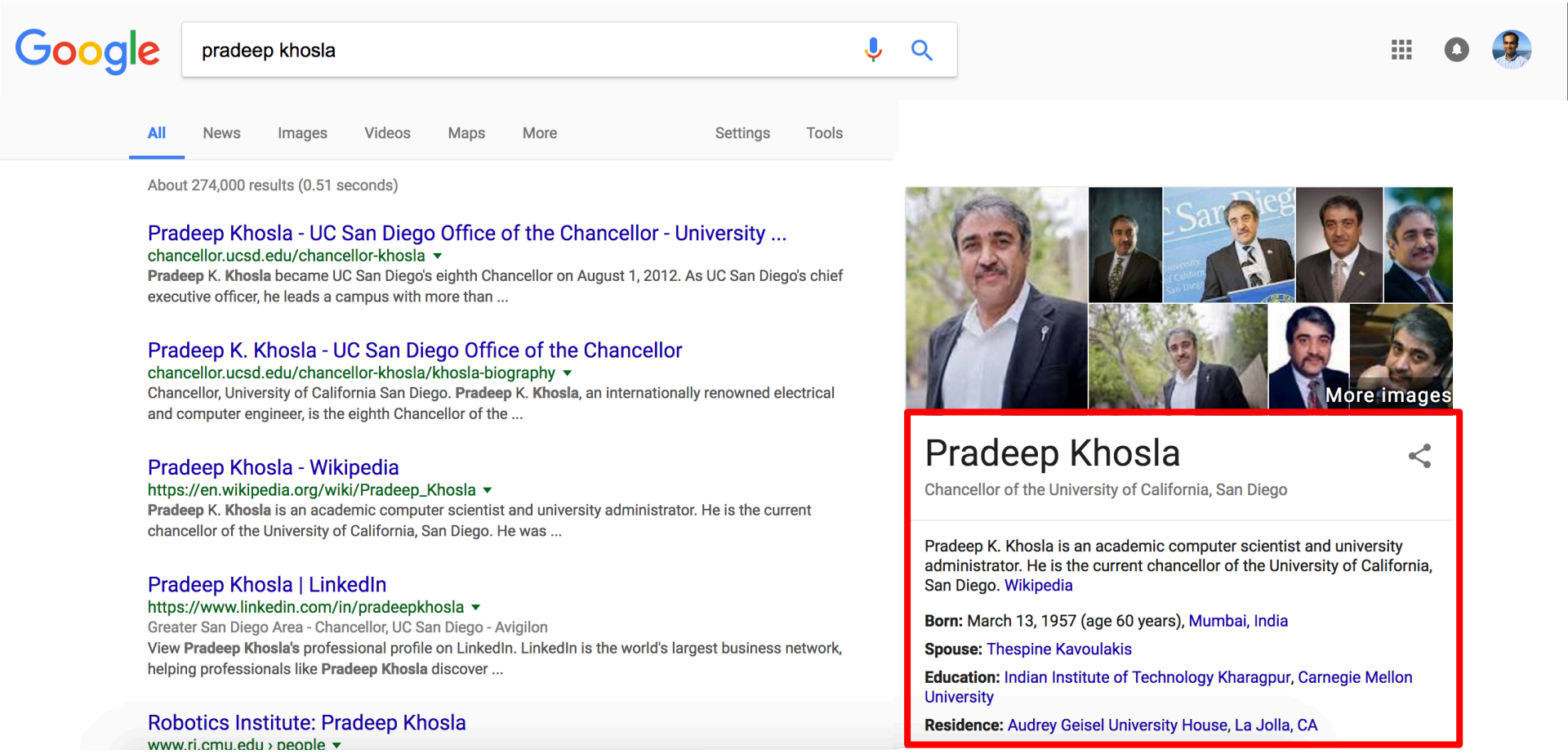


How does Netflix know that?

Large datasets + Machine learning!

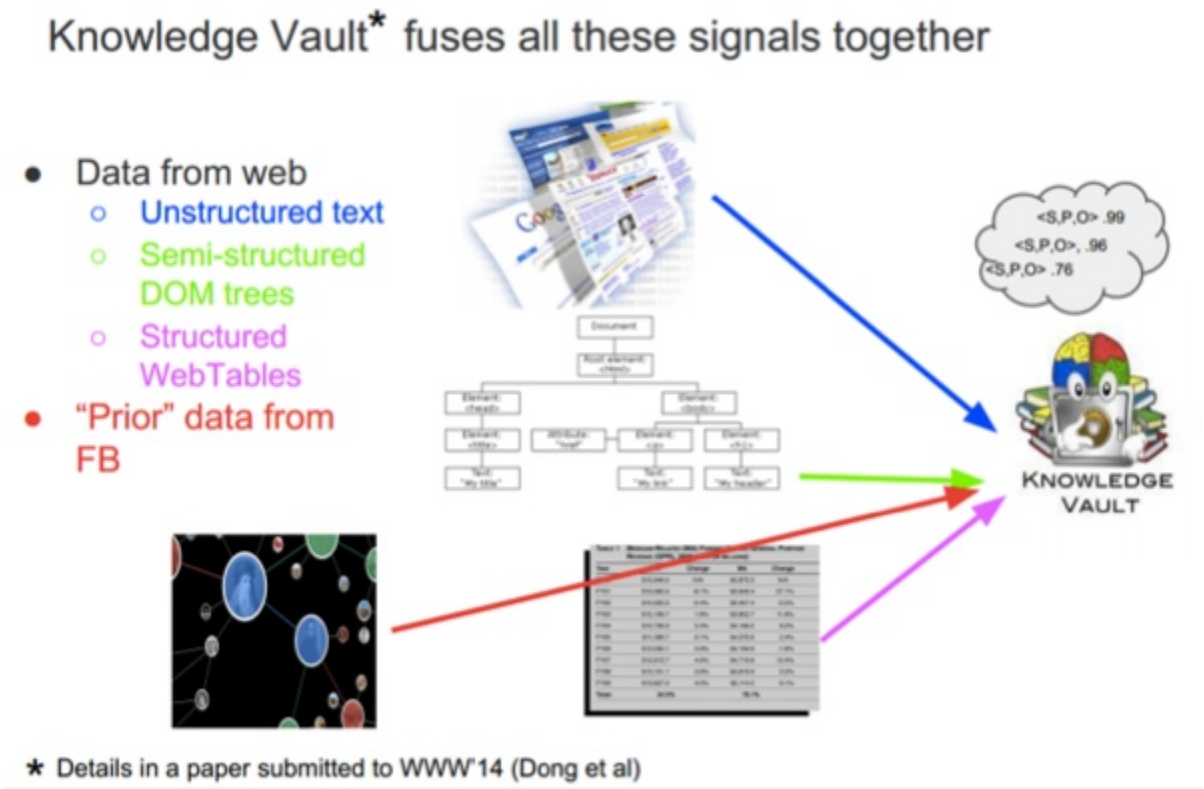
Log all user behavior (views, clicks, pauses, searches, etc.)
Recommender systems apply ML to TBs of data from all users and movies to deliver a tailored experience

2. Structured data with search results



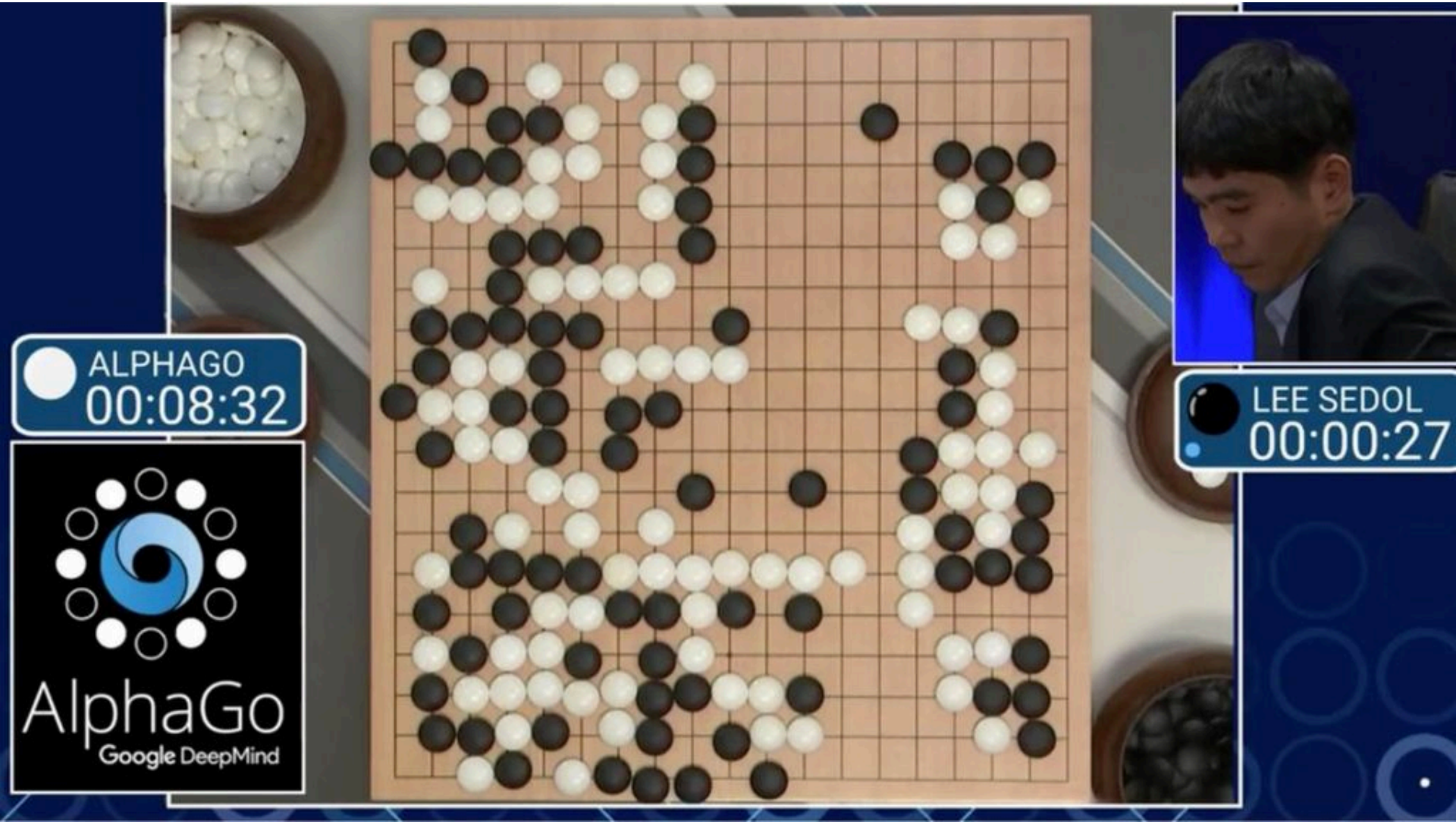
How does Google know that?

Large datasets + Machine learning!



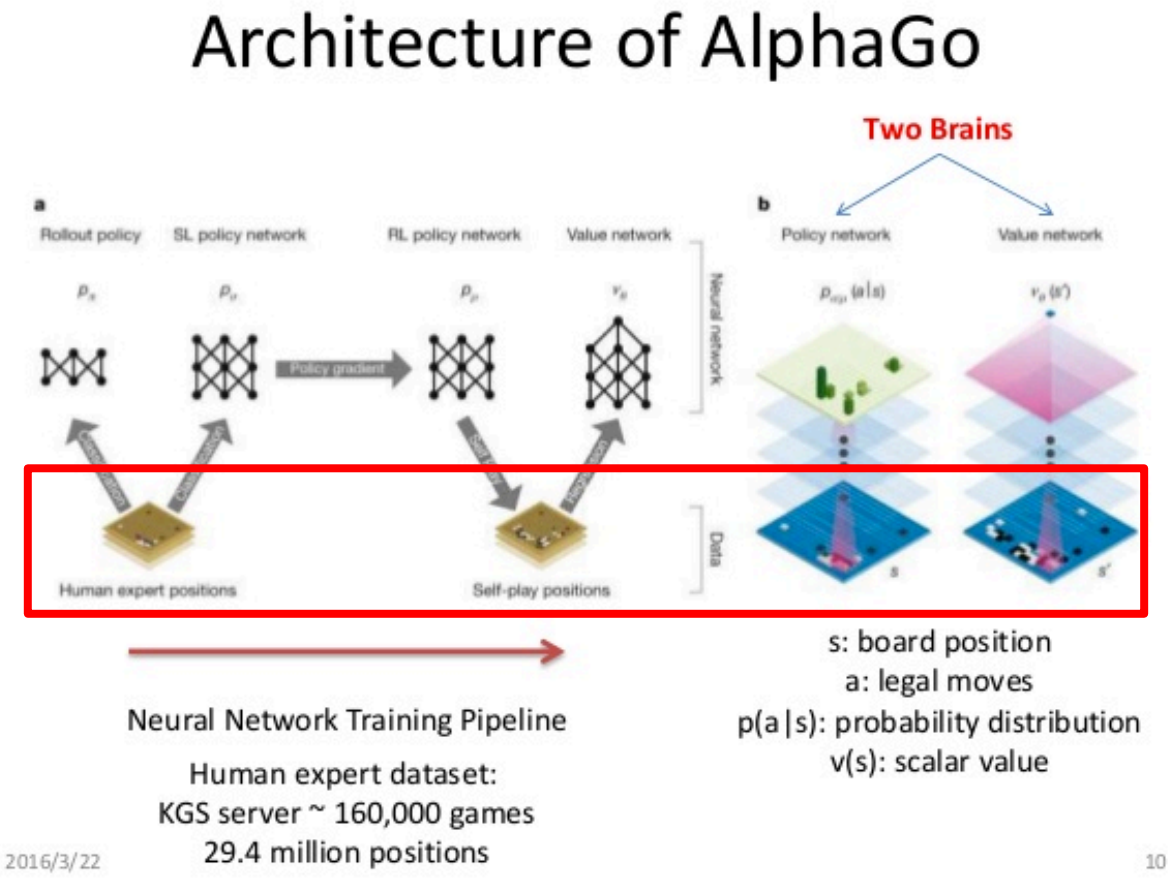
Knowledge Base Construction (KBC) process extracts tabular/relational data from large amounts of text data

3. AlphaGo defeats human champion!



How did AlphaGo achieve that?

Breakthrough powered by deep learning!



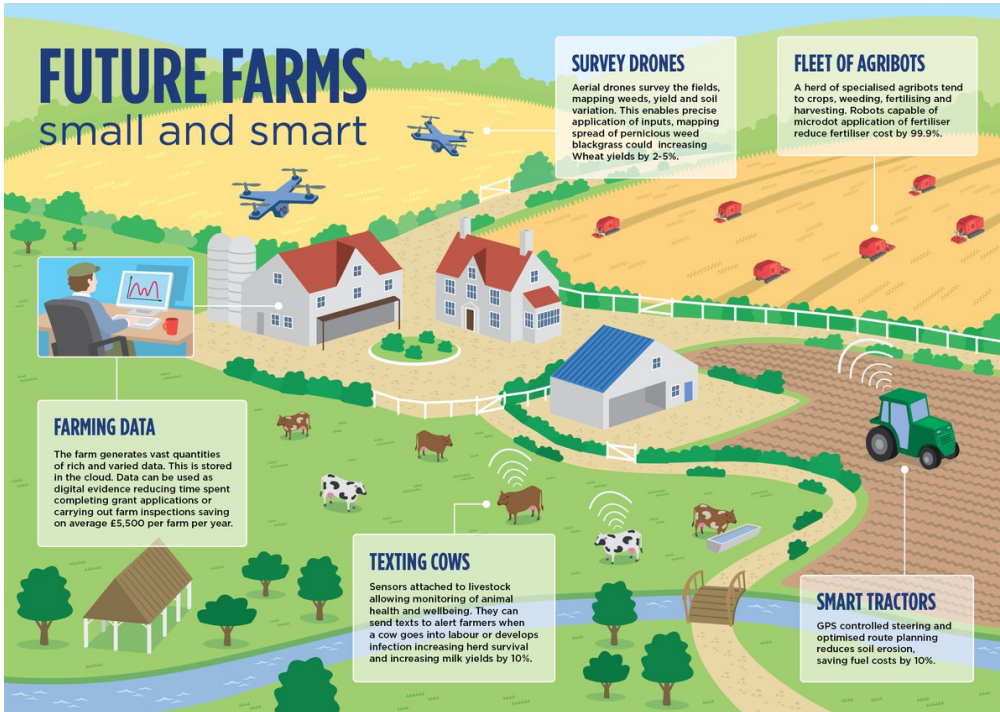
Deep CNNs to visually process board status in plays

https://www.slideshare.net/SanFengChang/mastering-the-game-of-go-with-deep-neural-networks-and-tree-search

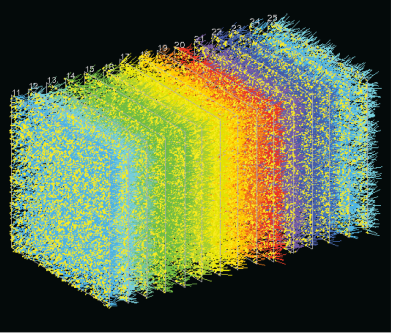
Innumerable “enterprise” applications

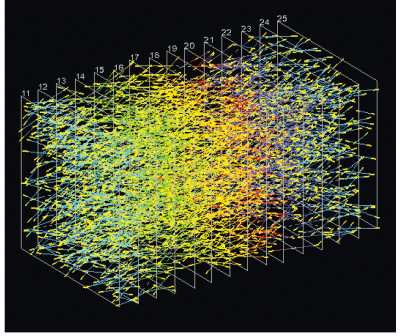

Enter

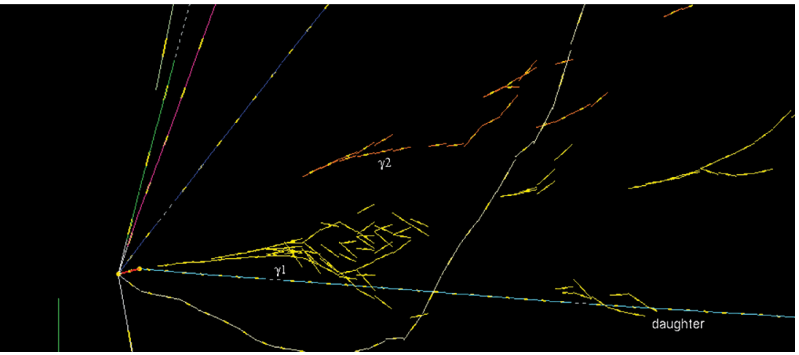

I ♥ INSURANCE
FLO

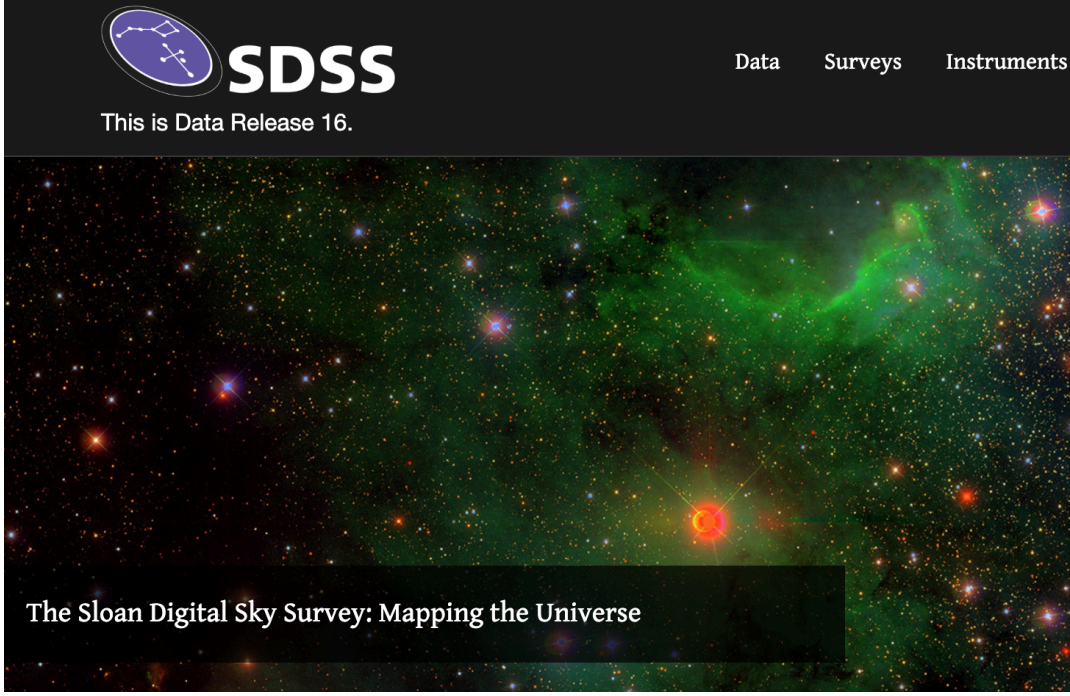

FUTURE FARMS
small and smart
SURVEY DRONES
Aerial drones survey the fields, mapping weeds, yield and soil variation. This enables precise application of inputs, mapping spread of pernicious weed blackgrass could increasing Wheat yields by 2-5%.
FLEET OF AGRIBOTS
A herd of specialised agribots tend to crops, weeding, fertilising and harvesting. Robots capable of microdot application of fertiliser reduce fertiliser cost by 99.9%.
FARMING DATA
The farm generates vast quantities of rich and varied data. This is stored in the cloud. Data can be used as digital evidence reducing time spent completing grant applications or carrying out farm inspections saving on average £5,500 per farm per year.
TEXTING COWS
Sensors attached to livestock allowing monitoring of animal health and wellbeing. They can send texts to alert farmers when a cow goes into labour or develops infection increasing herd survival and increasing milk yields by 10%.
SMART TRACTORS
GPS controlled steering and optimised route planning reduces soil erosion, saving fuel costs by 10%.

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

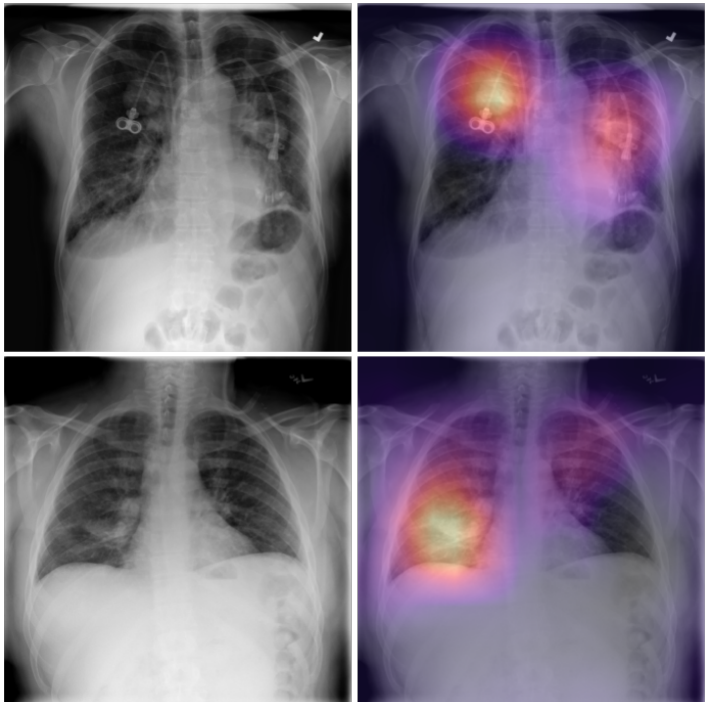





γ2
γ1
daughter


SDSS
Data
Surveys
Instruments
This is Data Release 16.
The Sloan Digital Sky Survey: Mapping the Universe


Gel.000718A
Channel: 37.50 Scan: 2343 Lanes Used: 36
2361
2295
2230
2165
2095
2030
1965
1895



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

By STEVE LOHR

Email

Share

Tweet

Save

Forbes / Entrepreneurs

Forbes

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

Drowning In Big Data - Finding Insight In A Digital

Josh Steimle, CONT

For roughly a deca
information about
Big Data. The IDC
industry will exper
by 2018. What this

DATA

Data Scientist: The Sexiest Job of the 21st Century

by Thomas H. Davenport and D.J. Patil

FROM THE OCTOBER 2012 ISSUE

Harvard Business Review

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

When 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!

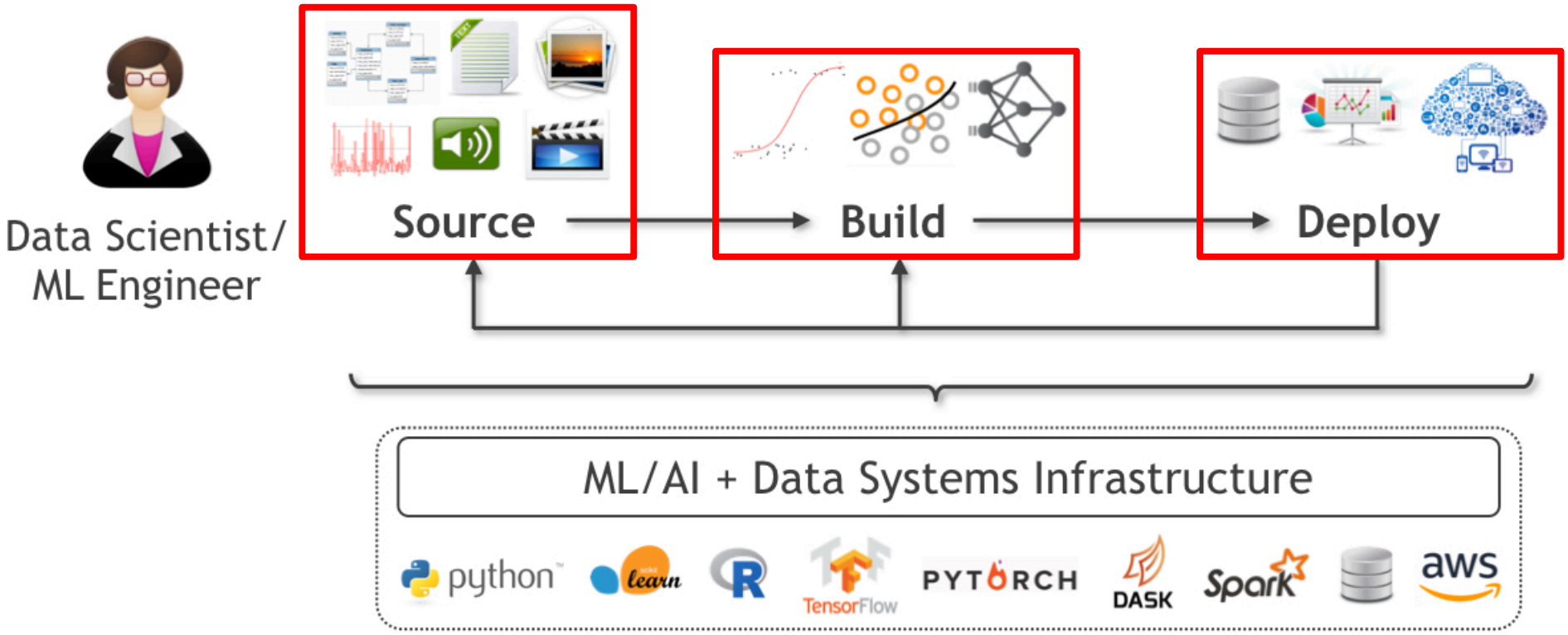
Data data everywhere,

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



Data acquisition
Data preparation

Feature Engineering
Training & Inference
Model Selection

Serving
Monitoring

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:

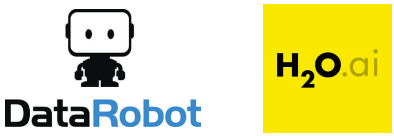
In-memory:

Disk-based files:

Layered on RDBMS/Spark:

Cloud-native:

"AutoML" platforms:



Decision tree-oriented:

Deep learning-oriented:

Data Systems Concerns in ML

Key concerns in ML:

Accuracy

Runtime efficiency (sometimes)

Additional key *practical* concerns in ML Systems:

Scalability (and efficiency at scale)

Usability

Manageability

Developability

Q: How do "ML Systems" relate to ML?

ML Systems : ML :: Computer Systems : TCS

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

Q: [illegible] ML systems?

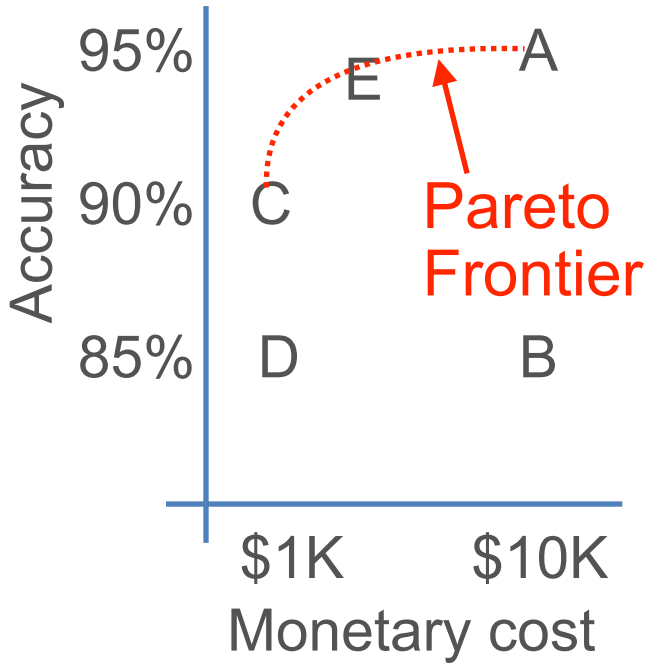
Conceptual System Stack Analogy

	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

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

http://cseweb.ucsd.edu/~arunkk/dsc102_winter22

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%le; Rel.: B-; Abs.: B+; so, B+

Tentative Course Schedule

Week	Topic
1-3	Basics of Computer Org. and Operating Systems
4	Basics of Cloud Computing
5-6	Parallel and Scalable Data Processing: Parallelism
	Midterm Exam on Wed, Feb 9
7-8	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**

Systems Principles

Scalability Principles

Scalable Analytics Systems

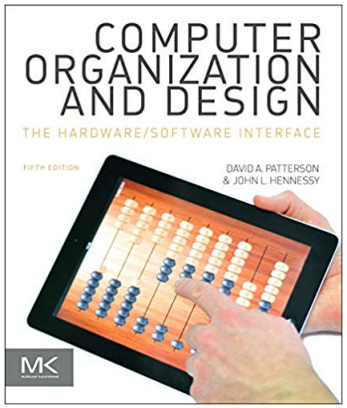
There will likely be 2 industry guest lectures; speakers TBD

Tentative Schedule for Prog. Assignments

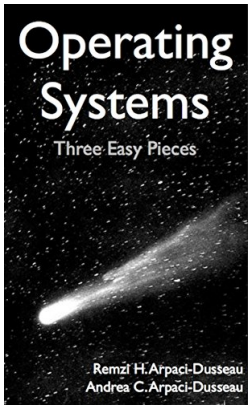
Date	Agenda
Jan 12	PA 0 released
Jan 14	Discussion on PA 0 by TA
Jan 25	**PA 0 due**
Jan 26	PA 1 released
Jan 28	Discussion on PA 1 by TA
Feb 16	**PA 1 due**
Feb 17	PA 2 released
Feb 18	Discussion on PA 2 by TA
Mar 8	**PA 2 due**

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

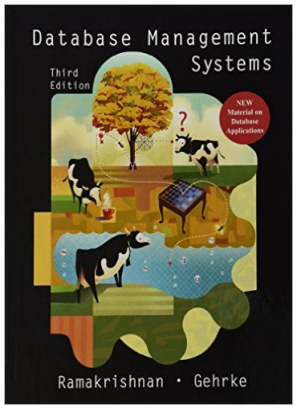
Suggested Textbooks



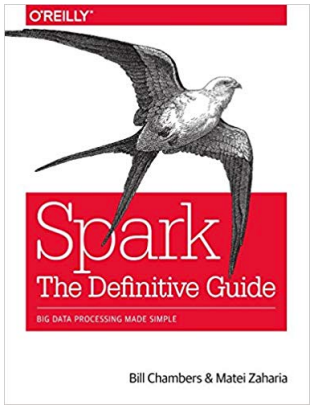
Aka “CompOrg Book”



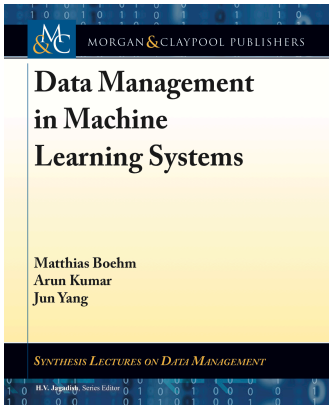
Aka “Comet Book”



Aka “Cow Book”



Aka “Spark Book”

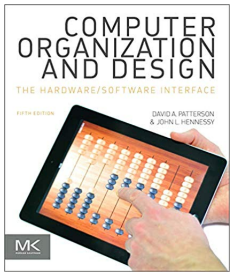


Aka “MLSys Book”

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

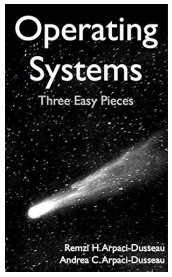
Why so many textbooks?!

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

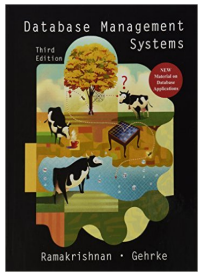


Hardware

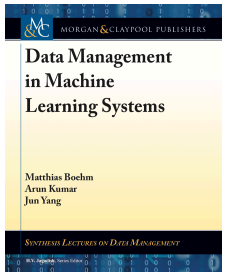
Low-level systems software

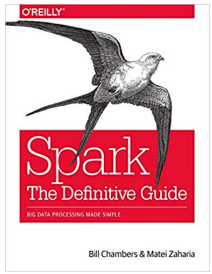


Higher-level relational dataflows



ML-oriented dataflows & lifecycle





More general scalable dataflows

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

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!

UC San Diego

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.

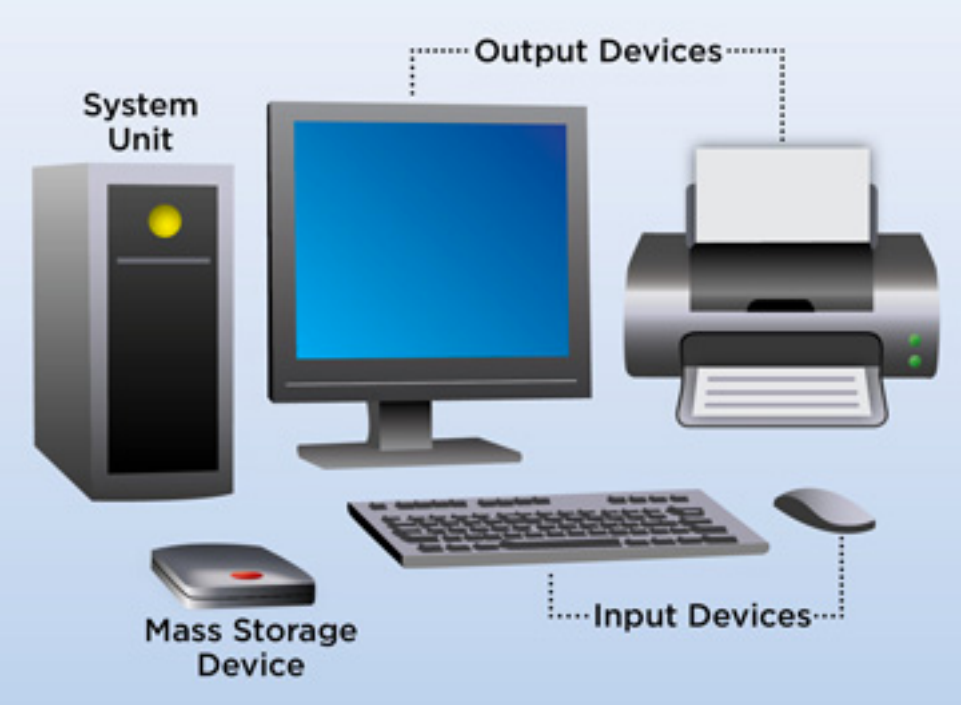
Computer science aka “Datalogy”

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.)

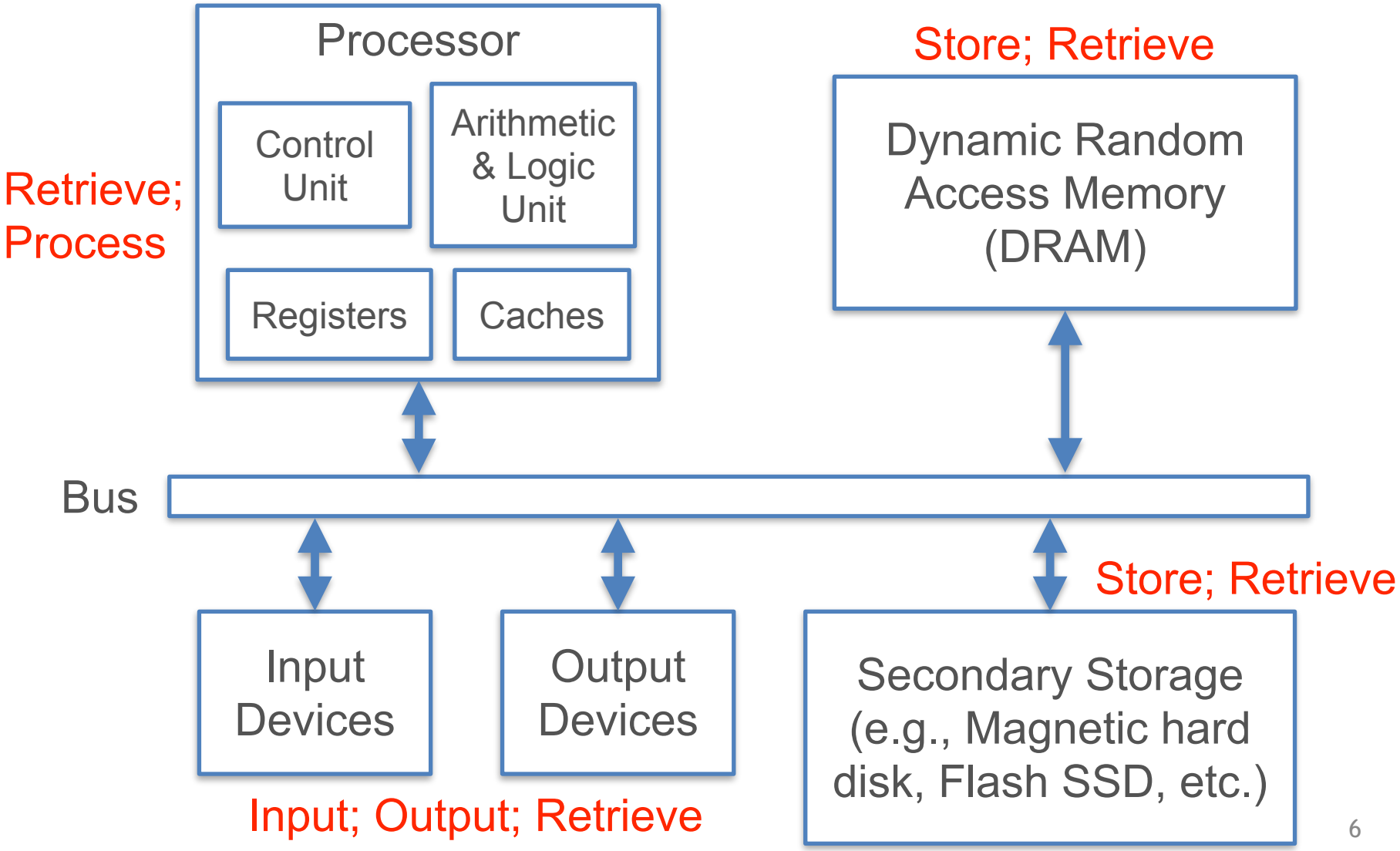
Software:
Programs (instructions) and data

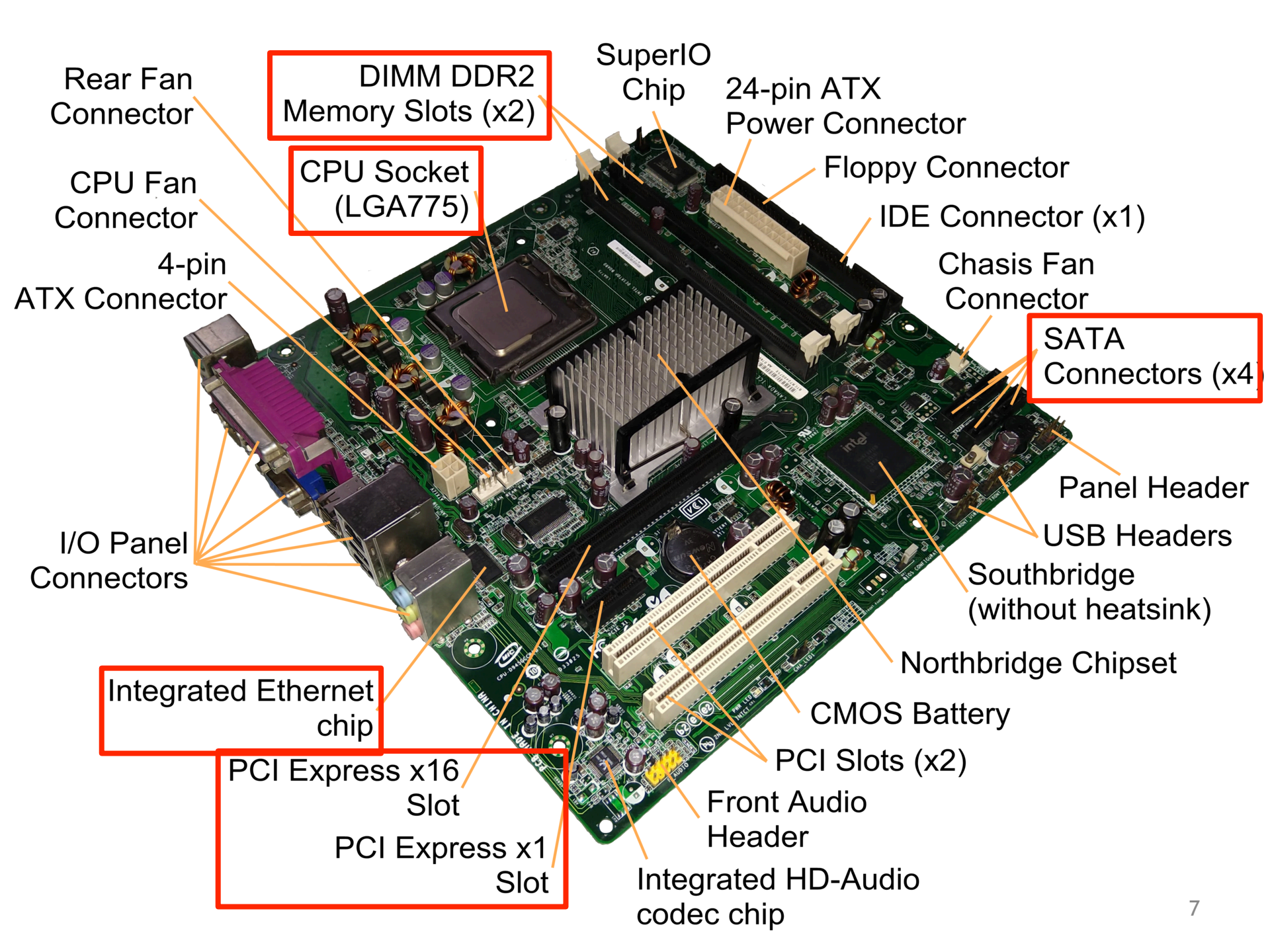
https://www.webopedia.com/TERM/C/computer.html

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




Rear Fan Connector
DIMM DDR2 Memory Slots (x2)
SuperIO Chip
24-pin ATX Power Connector
CPU Fan Connector
CPU Socket (LGA775)
Floppy Connector
IDE Connector (x1)
4-pin ATX Connector
Chasis Fan Connector
SATA Connectors (x4)
Panel Header
USB Headers
I/O Panel Connectors
Southbridge (without heatsink)
Northbridge Chipset
Integrated Ethernet chip
CMOS Battery
PCI Slots (x2)
PCI Express x16 Slot
Front Audio Header
PCI Express x1 Slot
Integrated HD-Audio codec chip

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.

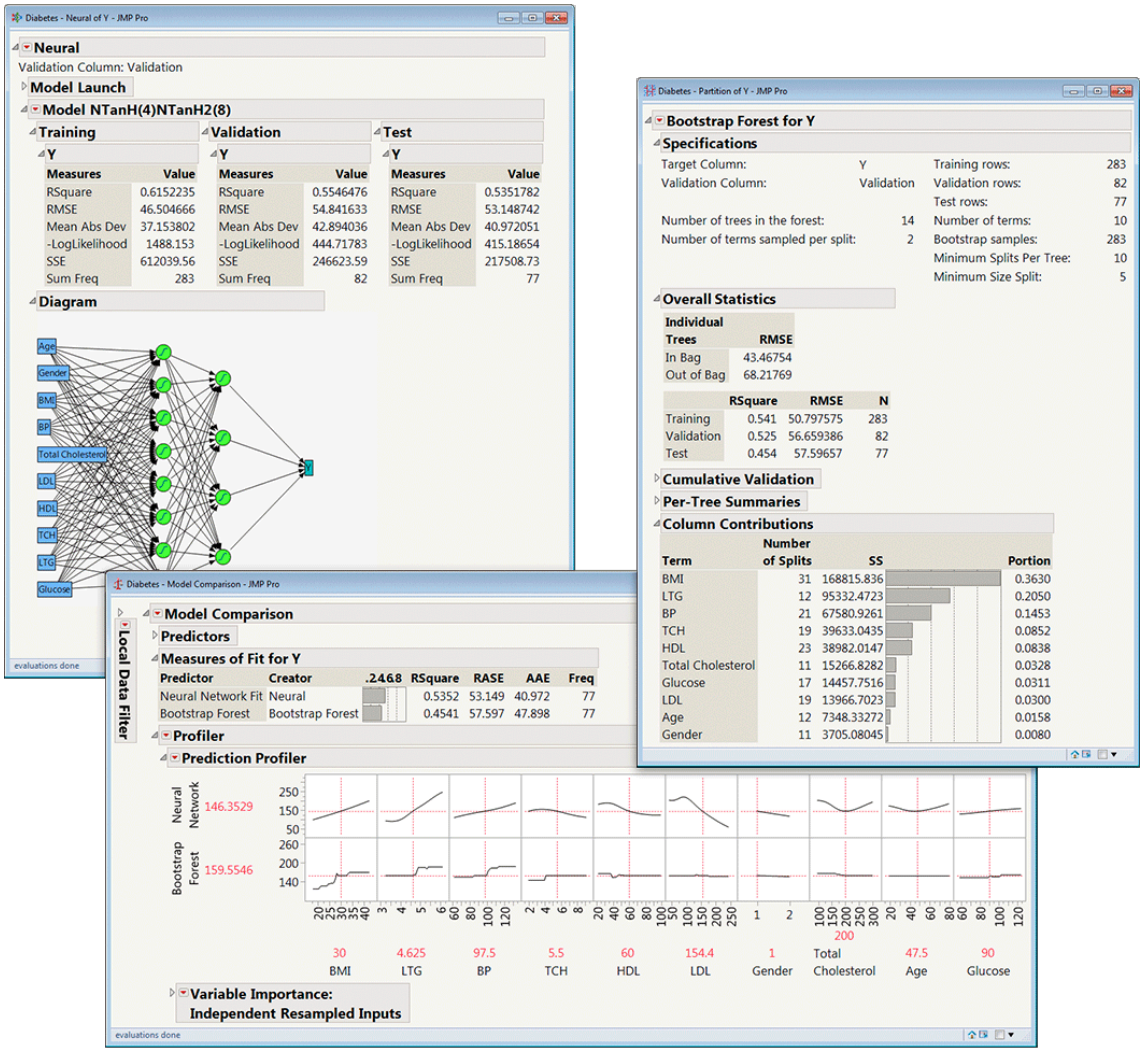
Outline

- 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?*

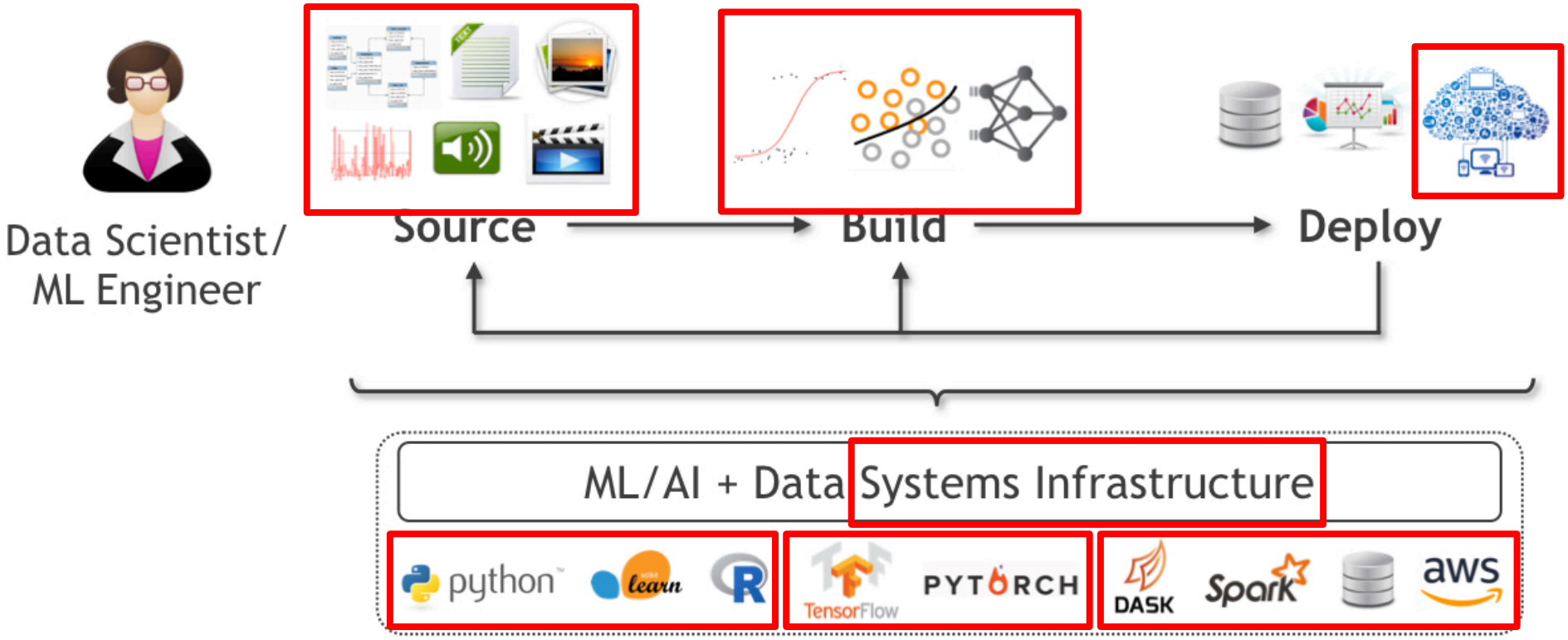
Luxury of “Statisticians”/“Analysts” of Yore

- **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.technologymagazine.com/data-and-data-analytics/sas-tops-worldwide-advanced-and-predictive-analytics-market-share

Reality of Today's "Data Scientists"



Data acquisition
Data preparation

Feature Engineering
Training & Inference
Model Selection

Serving
Monitoring

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

glassdoor

statistician | Location

Statistician Salaries United States

Overview | **Salaries** | Interviews | Insights | Career Path

How much does a Statistician make?

Updated Jan 4, 2022

Industry: All industries

Employer Size: All company sizes

Experience: All years of Experience

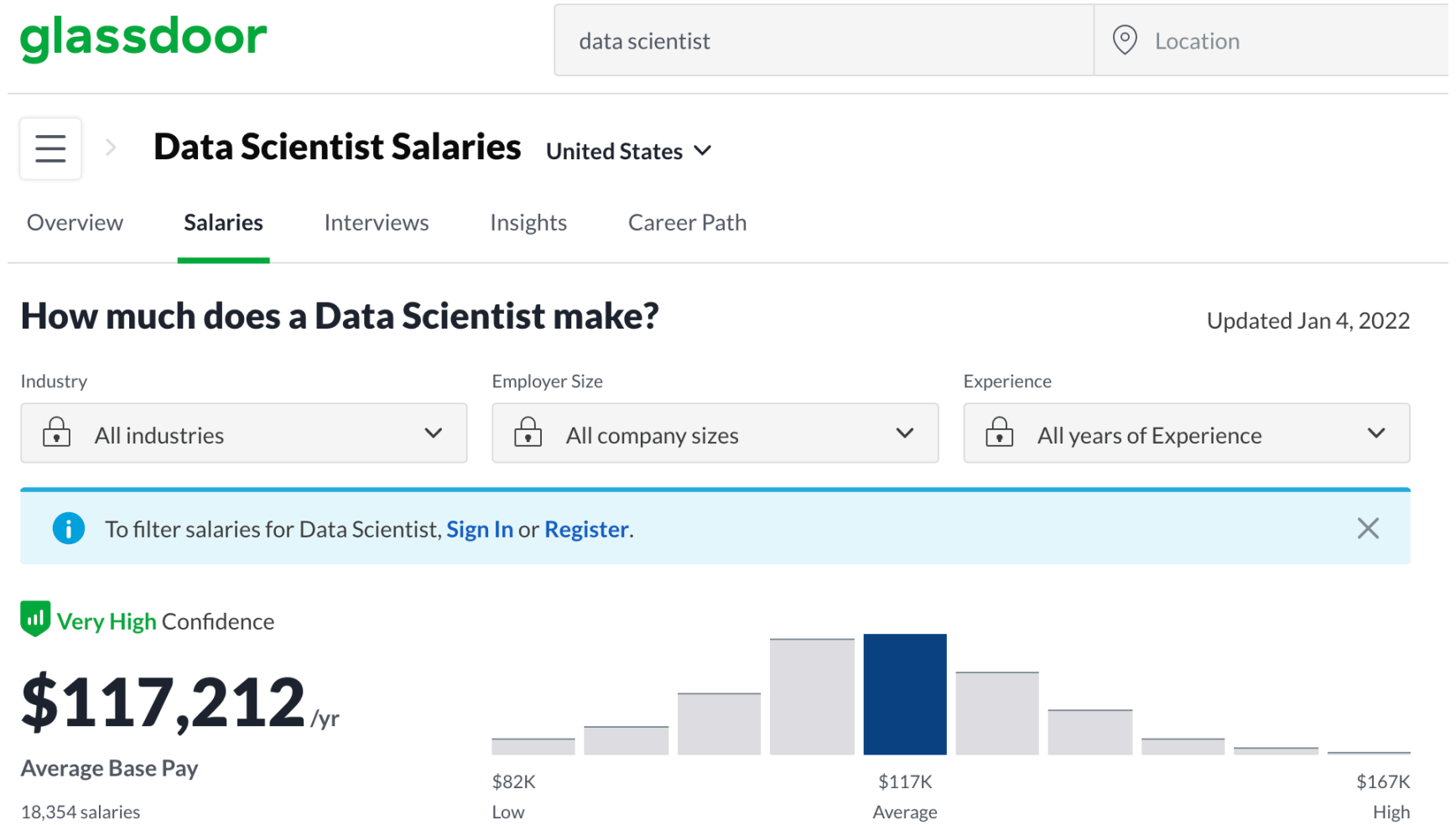
Very High Confidence

$88,989 /yr

Average Base Pay

2,398 salaries

$61K Low | $89K Average | $131K High


glassdoor
data scientist
Location
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
$82K
Low
$117K
Average
$167K
High

— 88,989

= 28,223!

Outline

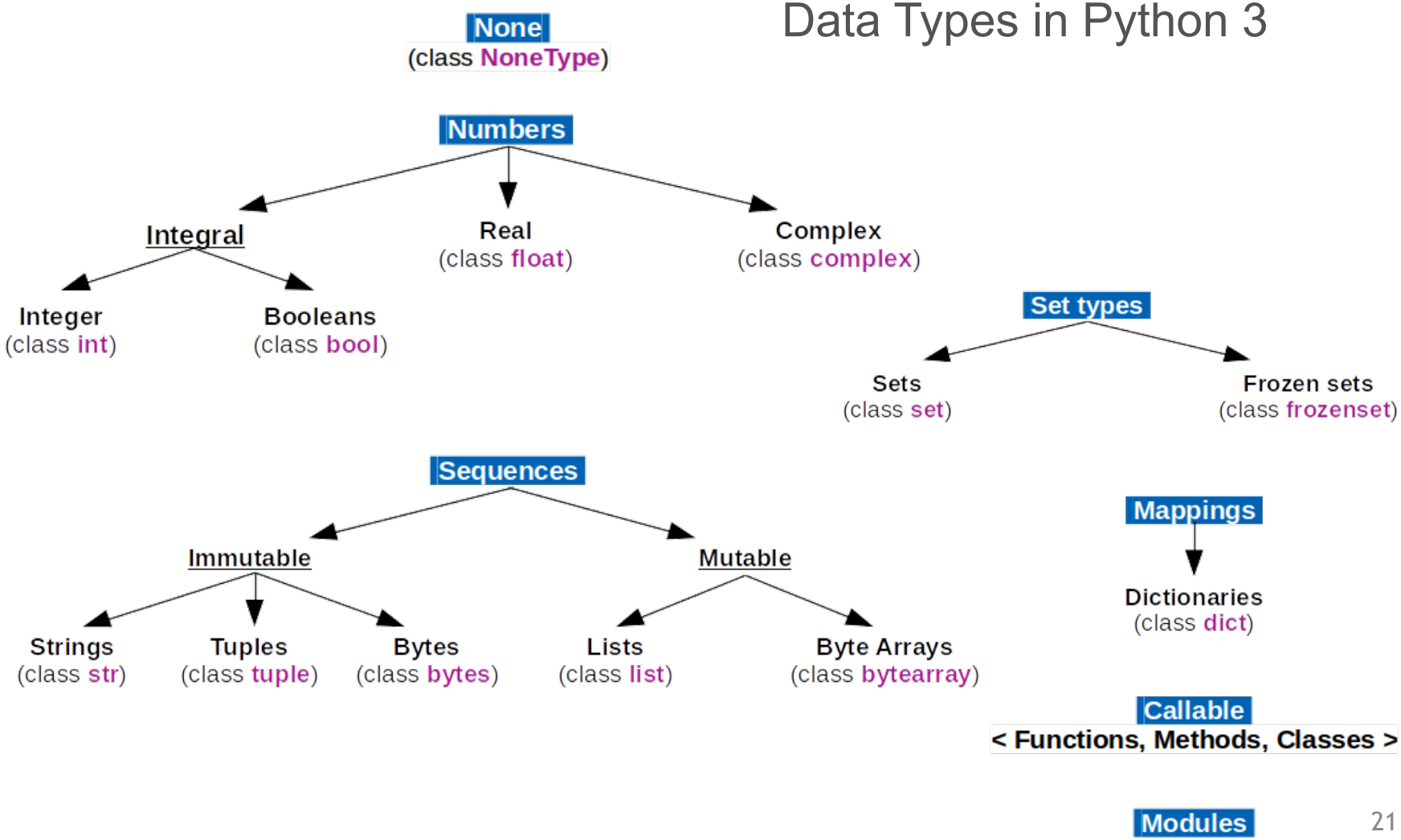
- Basics of Computer Organization
  - Digital Representation of Data
  - Processors and Memory Hierarchy
- Basics of Operating Systems
  - Process Management: Virtualization; Concurrency
  - Filesystem and Data Files
  - Main Memory Management
- Persistent Data Storage

Q: *What is data?*

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 => 2^{24} items, i.e., 16,777,216 items
- Common approximation: 2^{10} (i.e., 1024) ~ 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 ->128 = 2^7; so, 7 bits

Digital Representation of Data

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

1. 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
2. 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
3. Reset n as n - 2^k; return to Steps 1-2
4. 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	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

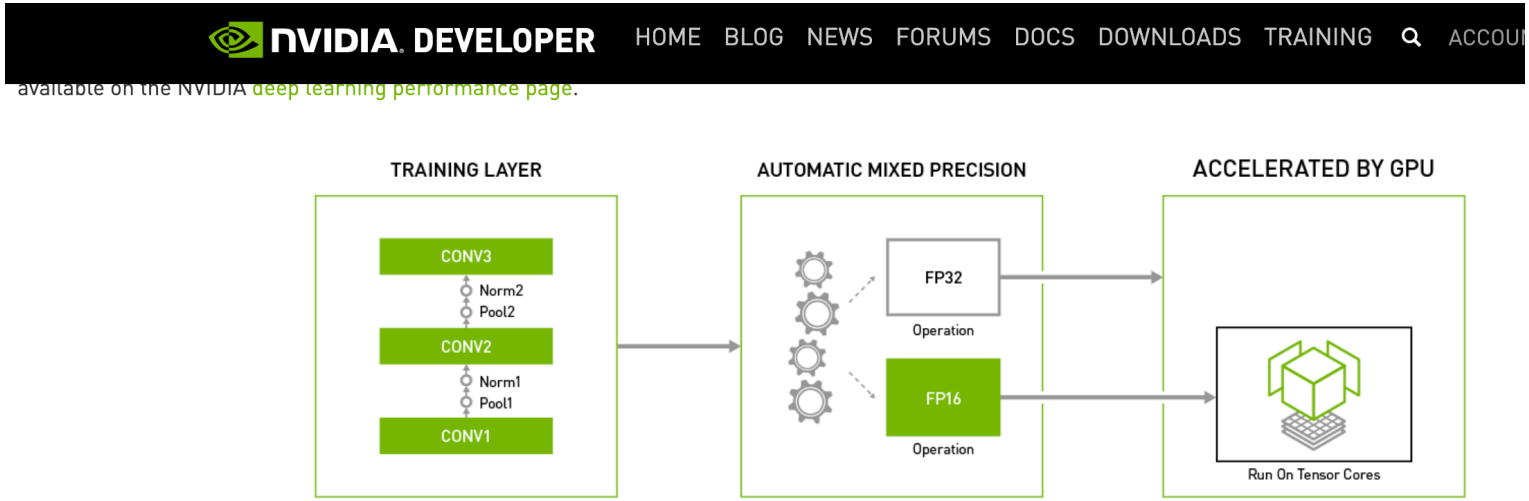
Decimal	Binary	Hexadecimal
5_{10}	101_2	5_{16}
47_{10}	$10\ 1111_2$	$2F_{16}$
163_{10}	$1010\ 0011_2$	$A3_{16}$
16_{10}	$1\ 0000_2$	10_{16}

Alternative notations

0xA3 or $A3_H$

Digital Representation of Data

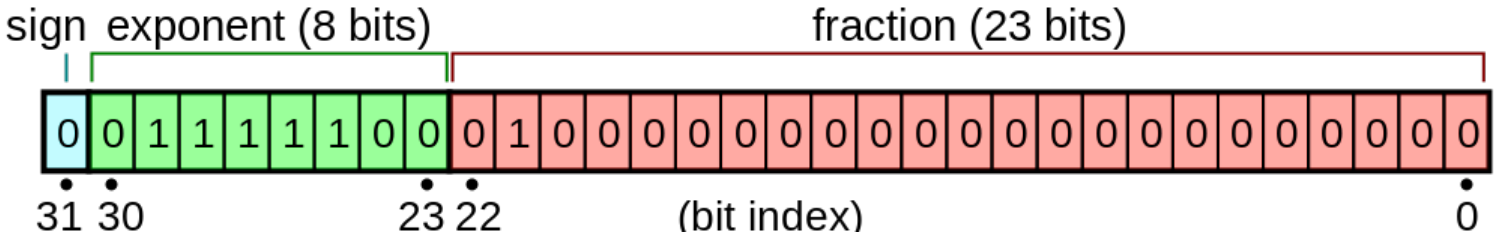
- **Float**:
  - Examples in data sci.: salary, scores, model weights
  - IEEE-754 single-precision format is 4B long; double-precision format is 8B long
  - Java and C *float* is single; Python *float* is double!



Using Automatic Mixed Precision for Major Deep Learning Frameworks

Digital Representation of Data

- **Float**:
  - Standard IEEE format for single (aka binary32):



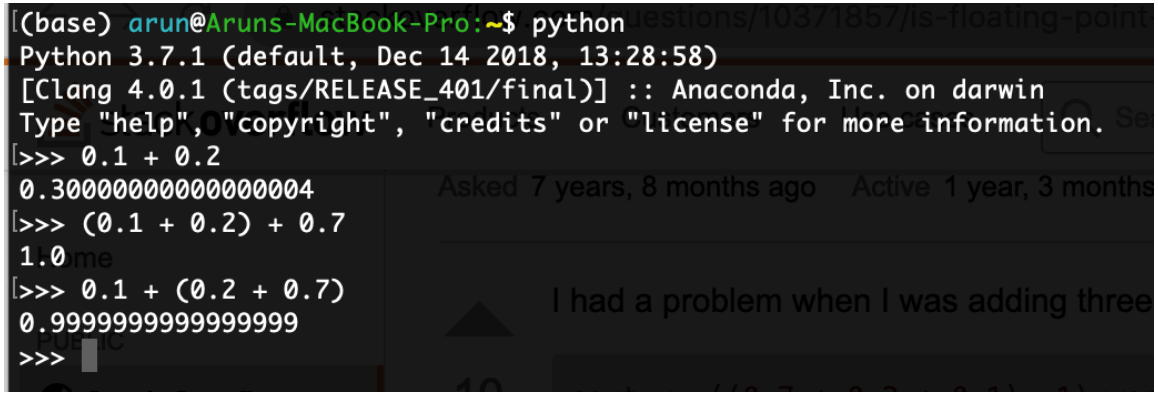
$$(-1)^{sign} \times 2^{exponent-127} \times (1 + \sum_{i=1}^{23} b_{23-i} 2^{-i})$$

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

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

Digital Representation of Data

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



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

https://docs.nvidia.com/deeplearning/sdk/mixed-precision-training/index.html

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 61, '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

Program in PL

Compile/Interpret ↓

Program in Assembly Language

Assemble ↓

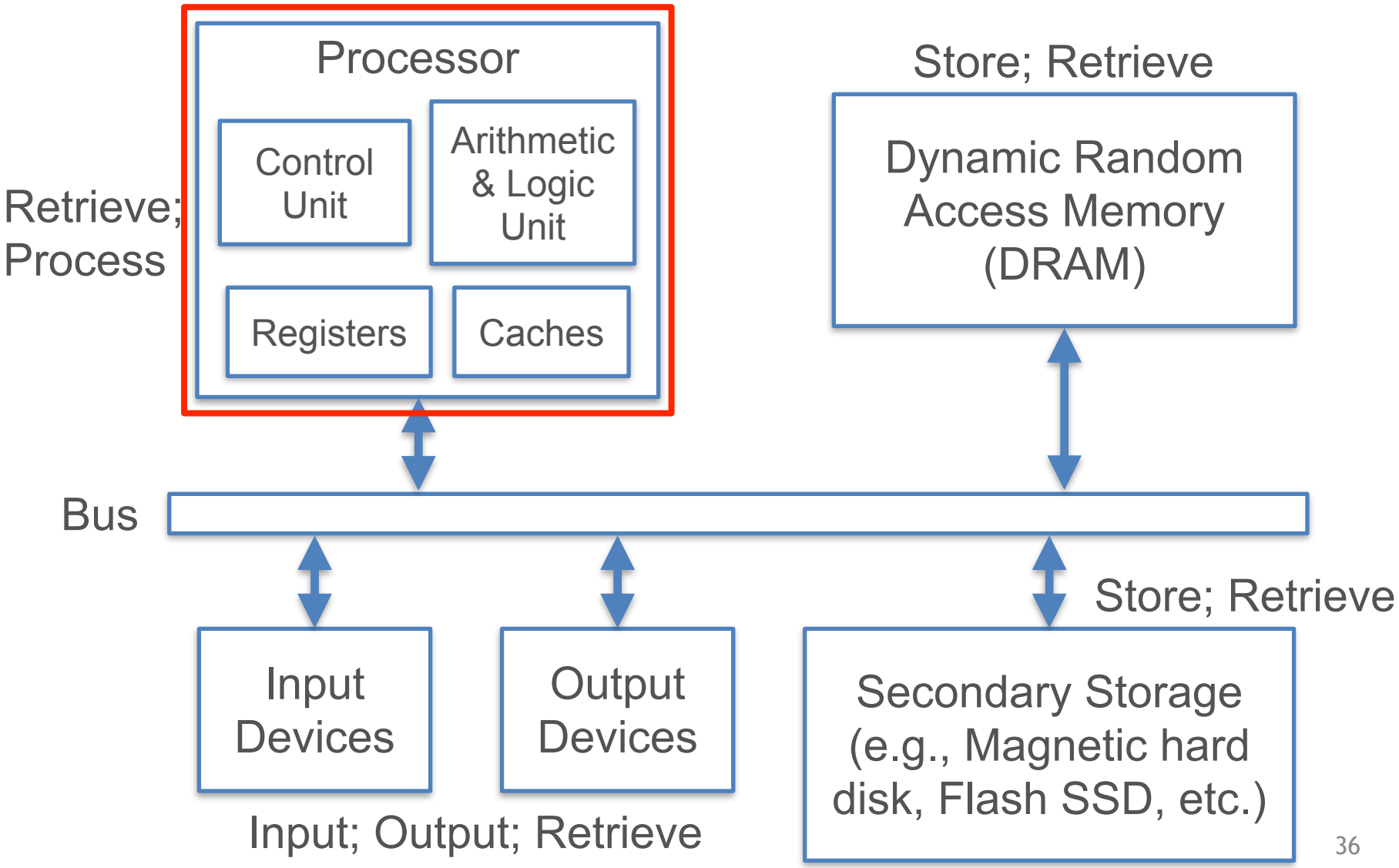
Machine code tied to ISA

↓

Run on processor

```
80483b4:    55                      push   %ebp
80483b5:    89 e5                   mov    %esp,%ebp
80483b7:    83 e4 f0                and    $0xfffffff0,%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

If interested in more details: https://www.youtube.com/watch?v=cNN_tTXABUA

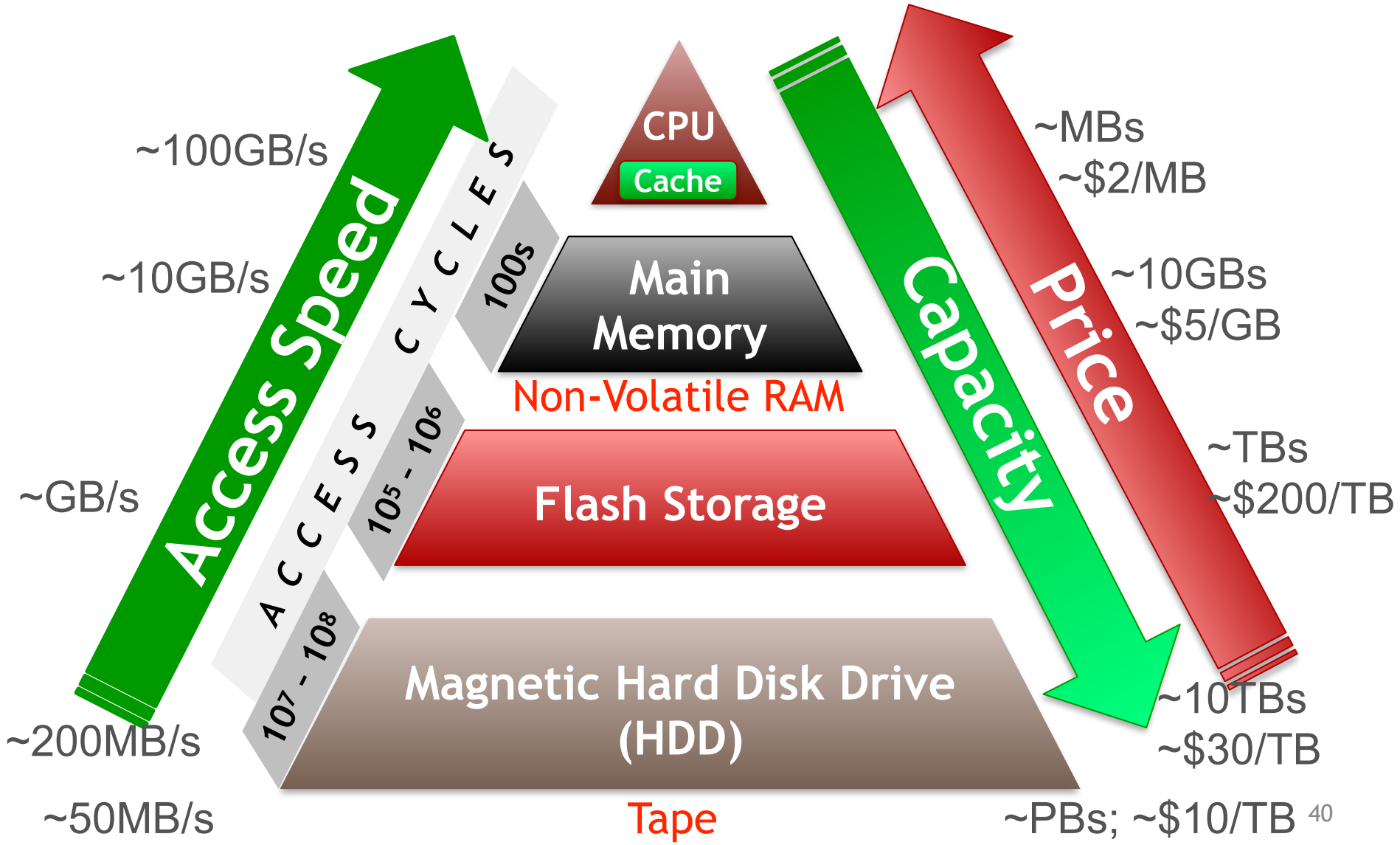
Processor Performance

***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

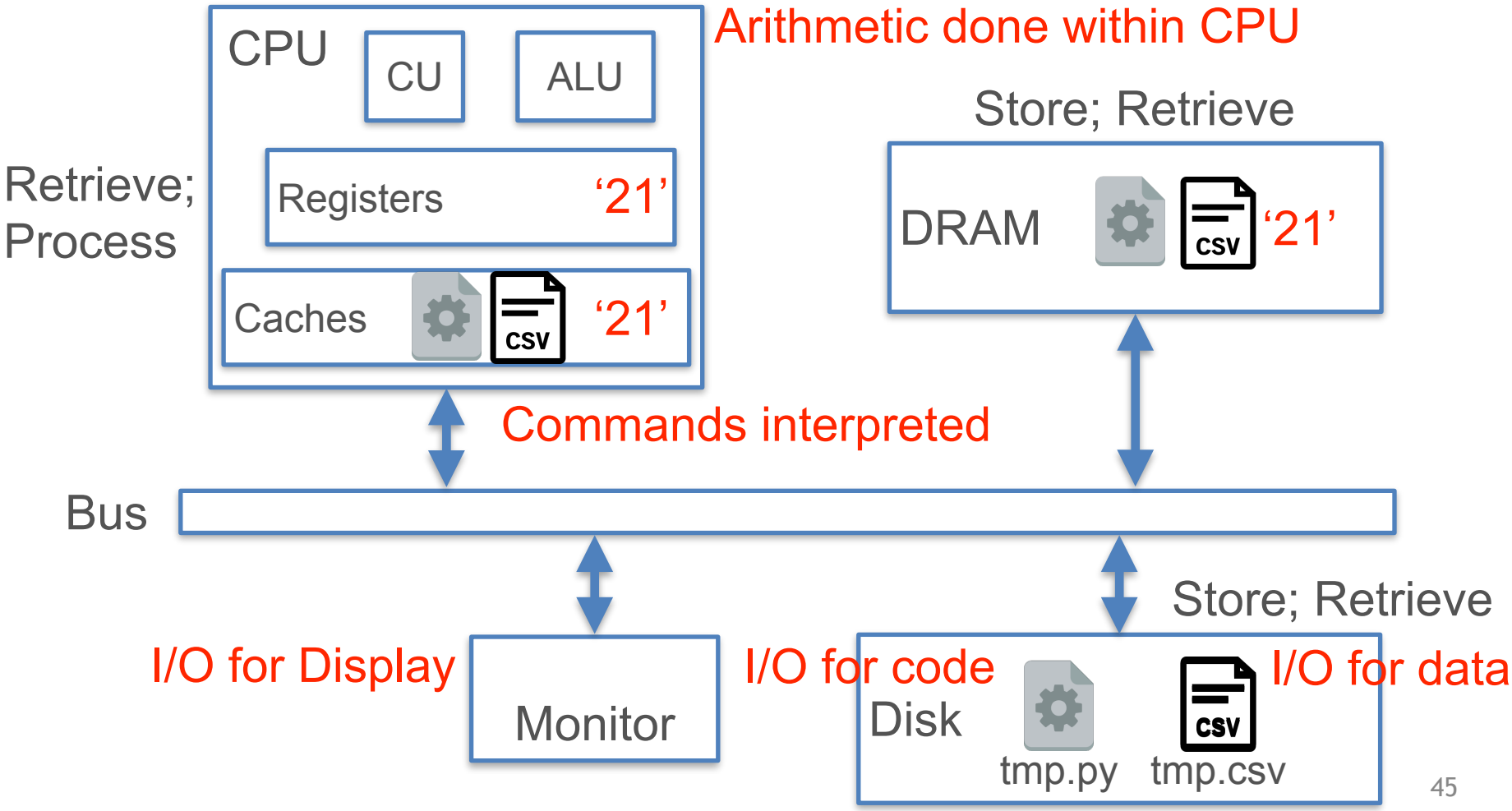
```
import pandas as p
m = p.read_csv('tmp.csv',header=None)
s = m.sum().sum()
print(s)
```

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

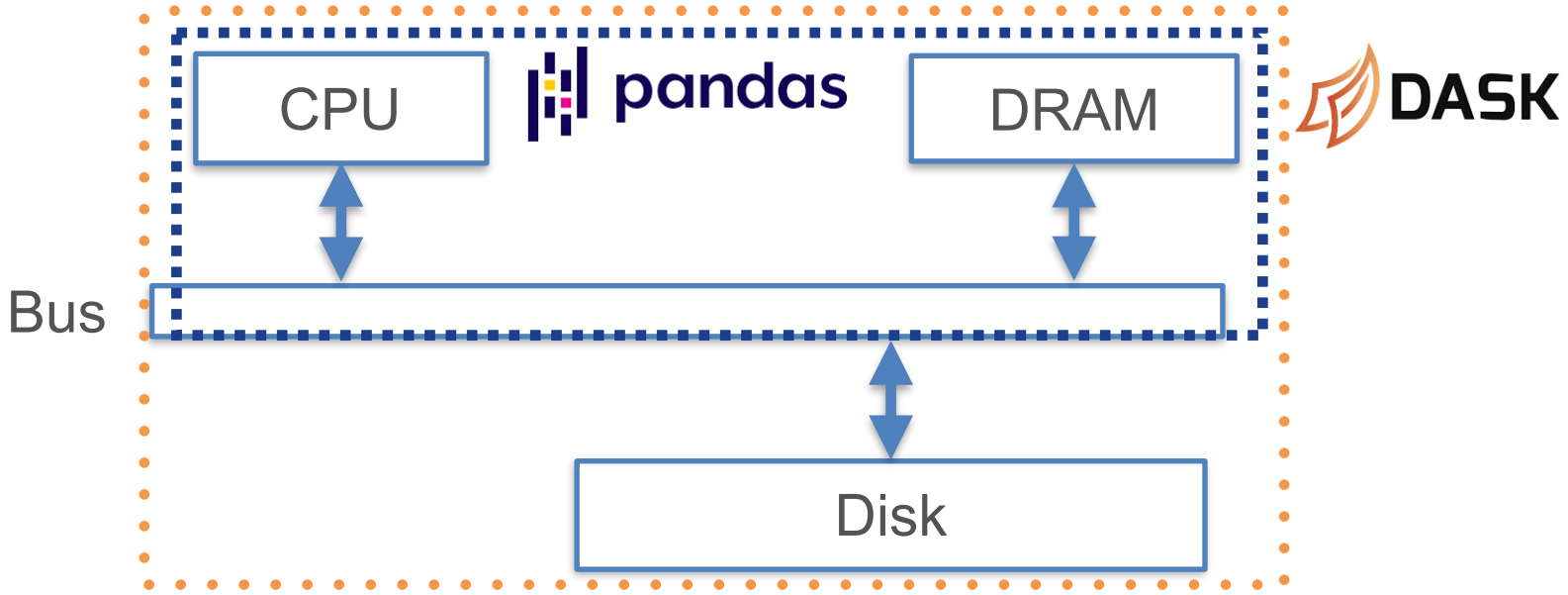
Breakout Rooms Activity (8min)

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

UC San Diego

DSC 102
Systems for Scalable Analytics

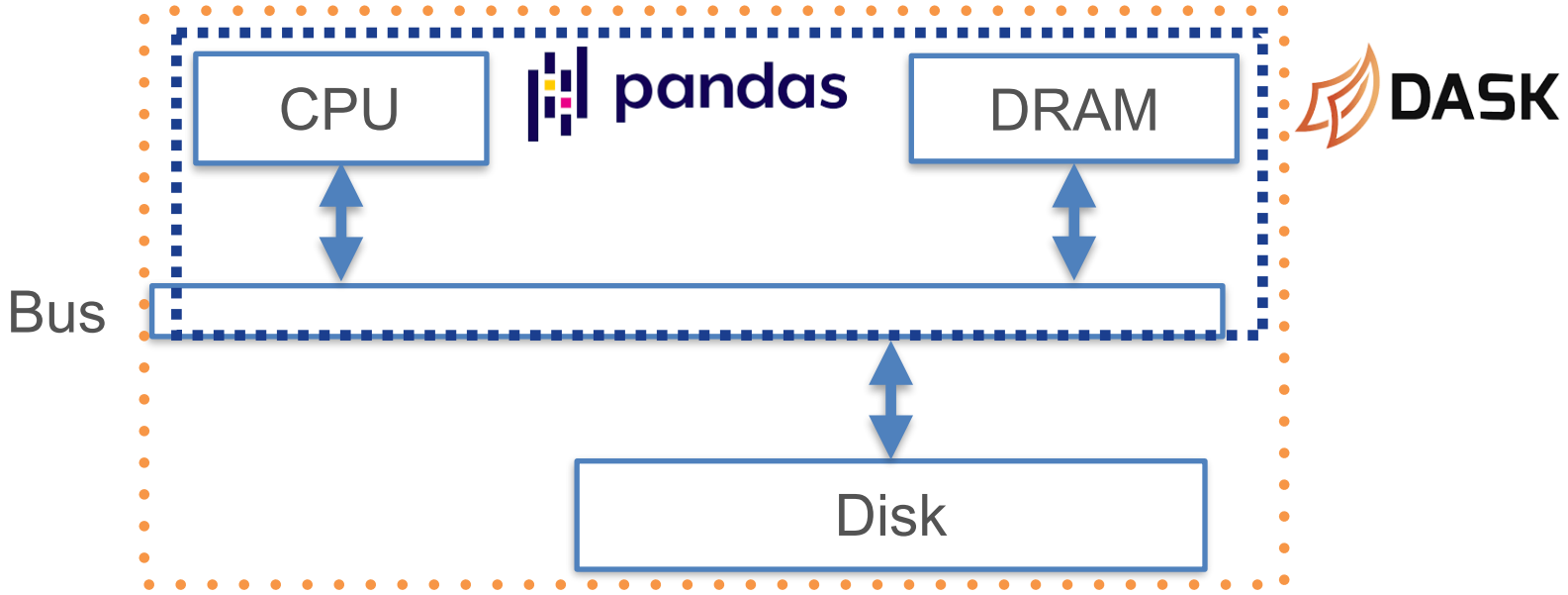
Arun Kumar

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

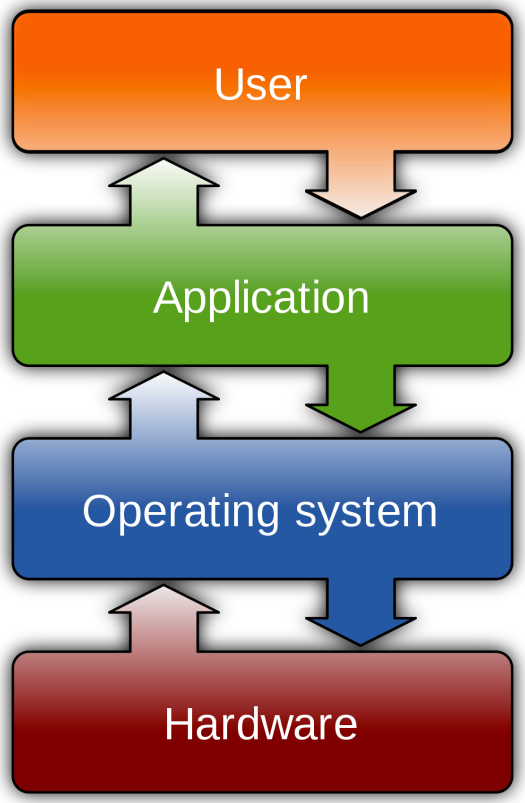
- Basics of Computer Organization
  - Digital Representation of Data
  - Processors and Memory Hierarchy
- Basics of Operating Systems (OS)
  - Process Management: Virtualization; Concurrency
  - Filesystem and Data Files
  - Main Memory Management
- Persistent Data Storage

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



APIs/Interface of Application Software

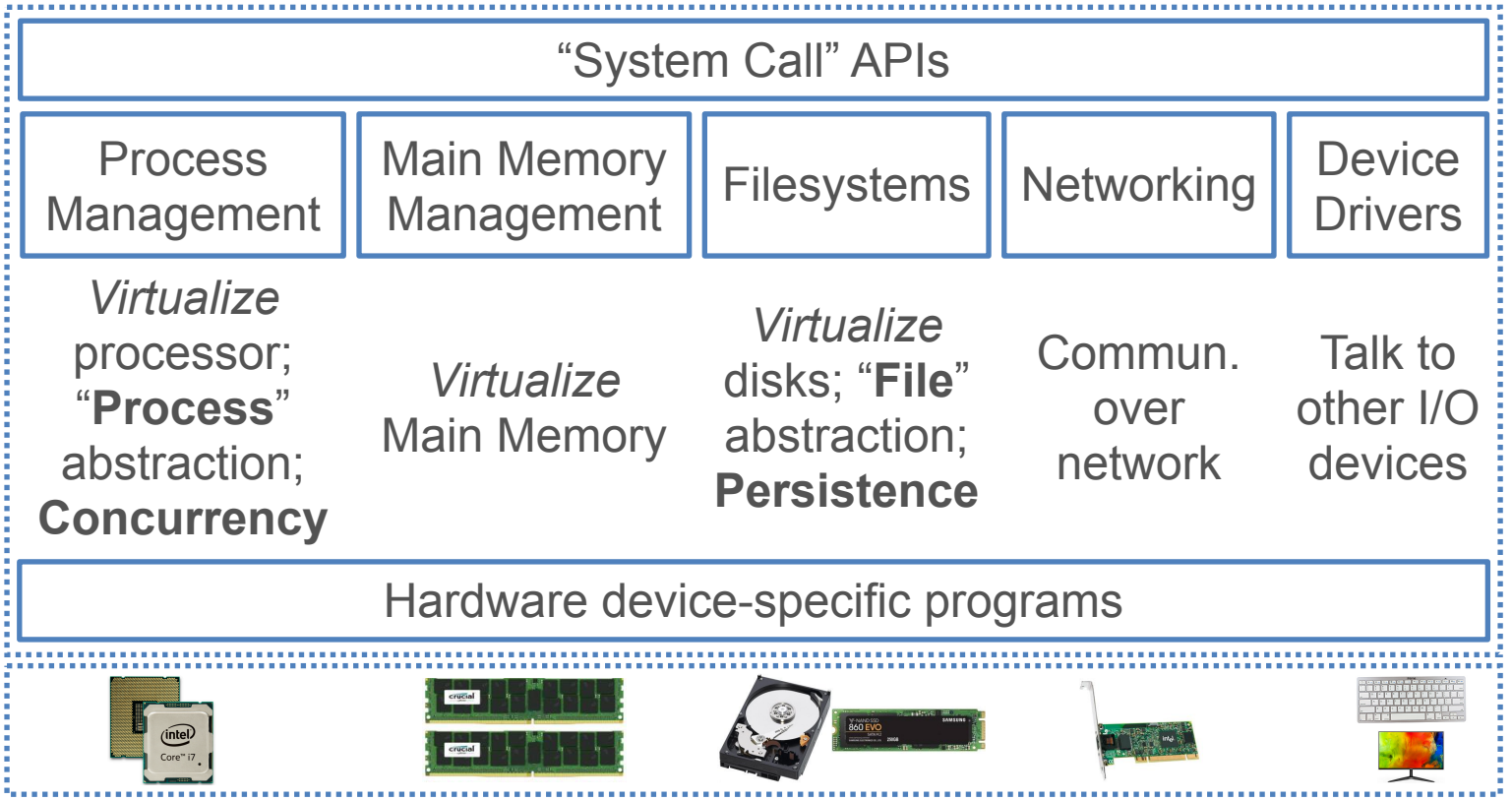
APIs of OS aka “System Calls”

Hardware-specific code parts of OS

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

Key Components of OS

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



Outline

- Basics of Computer Organizati
  - Digital Representation of Data
  - Processors and Memory Hierarc
- Basics of Operating Systems (OS
  - Process Management: Virtualization; Concurrency
  - Filesystem and Data Files
  - Main Memory Managemer
- Persistent Data Storag

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

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**

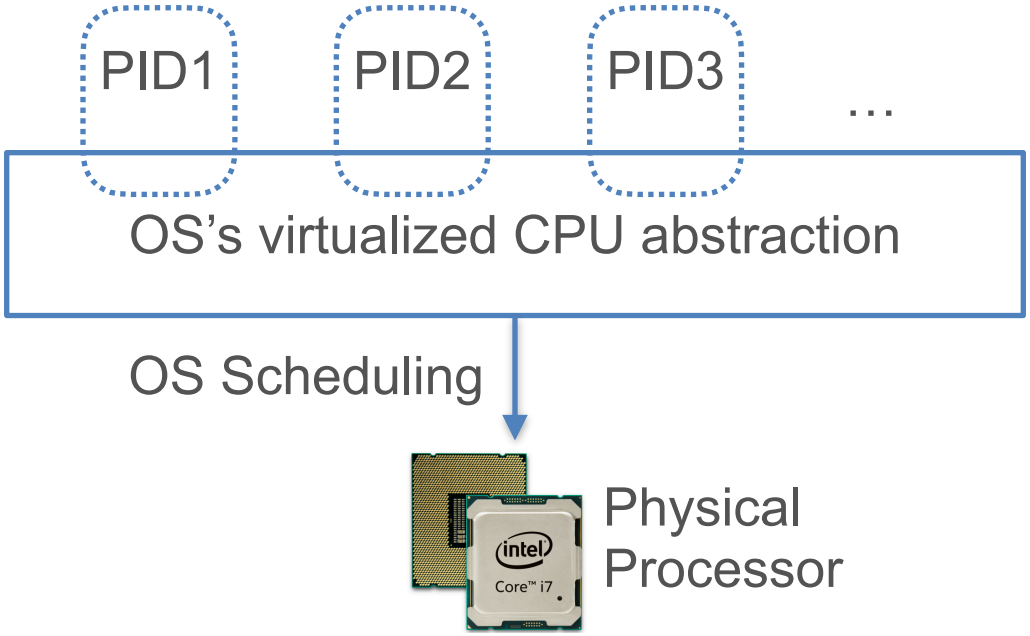
Virtualization of Hardware Resources

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

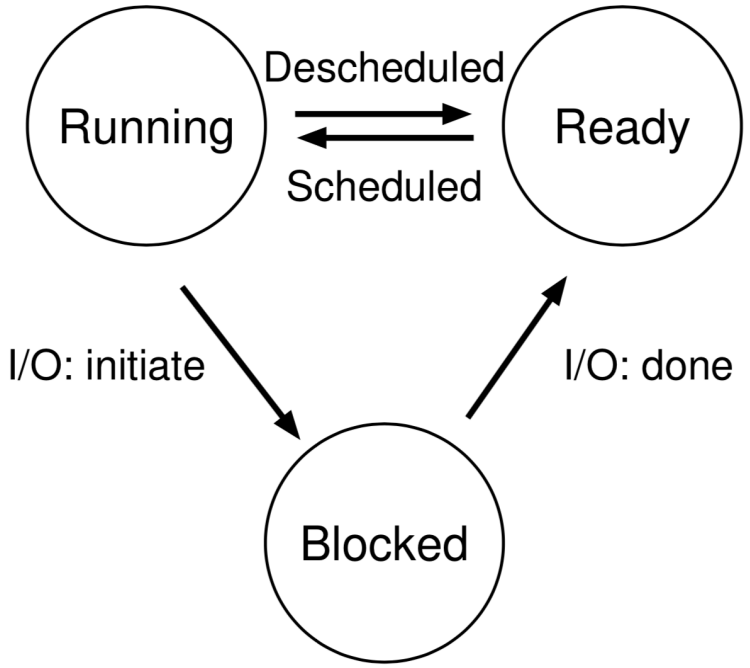
- Virtualization of processor enables process **isolation**, i.e., each process given an “illusion” that it alone runs



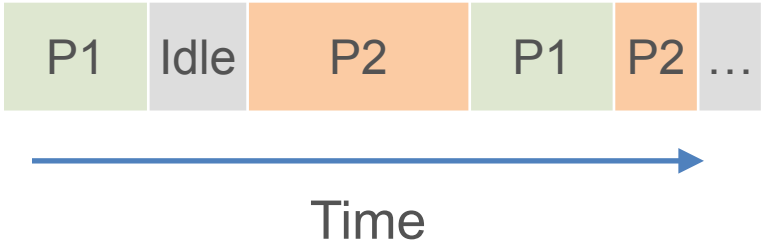
- Inter-process communication possible in System Calls API
- Later: Generalize to **Thread** abstraction for **concurrency**

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

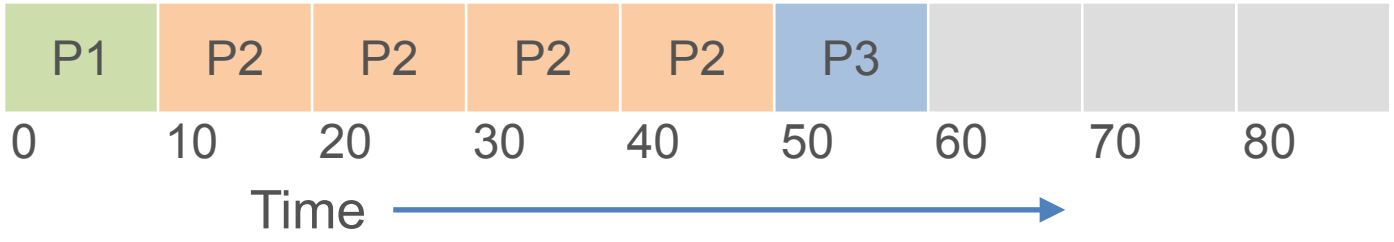
Scheduling Policies/Algorithms

- 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



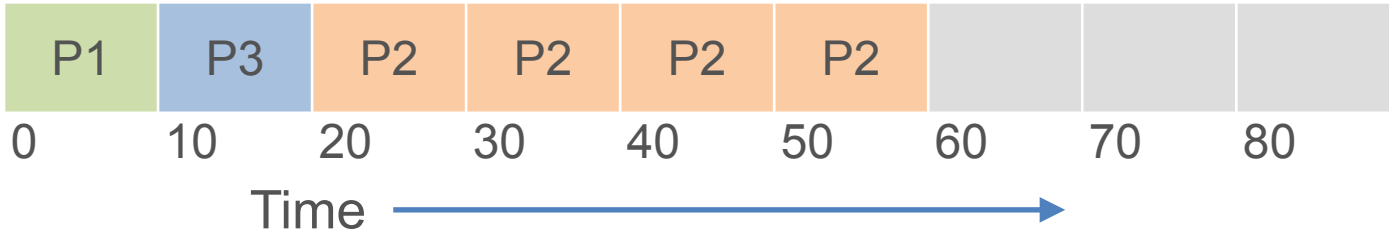
Process	Arrival Time	Start Time	Completion Time	Response Time	Turnaround Time
P1	0	0	10	0	10
P2	0	10	50	10	50
P3	0	50	60	50	60
			Avg:	20	40

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

Scheduling Policy: SJF

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

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



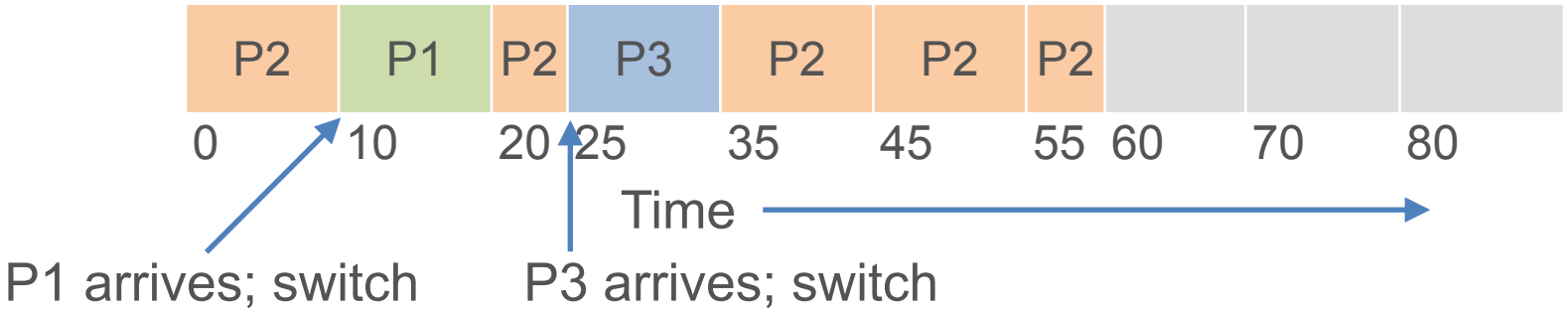
Process	Arrival Time	Start Time	Completion Time	Response Time	Turnaround Time
P1	0	0	10	0	10
P2	0	20	60	20	60
P3	0	10	20	10	20
	(FIFO Avg: 20 and 40)		Avg:	10	30

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

Scheduling Policy: SCTF

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

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



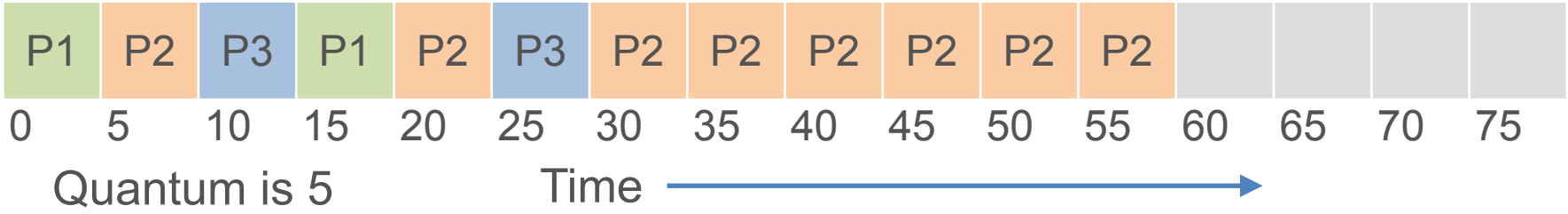
Process	Arrival Time	Start Time	Completion Time	Response Time	Turnaround Time
P1	10	10	20	0	10
P2	0	0	60	0	60
P3	25	25	35	0	10
	(SJF Avg: 10 and 30)		Avg:	0	26.7

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

Scheduling Policy: Round Robin

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

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



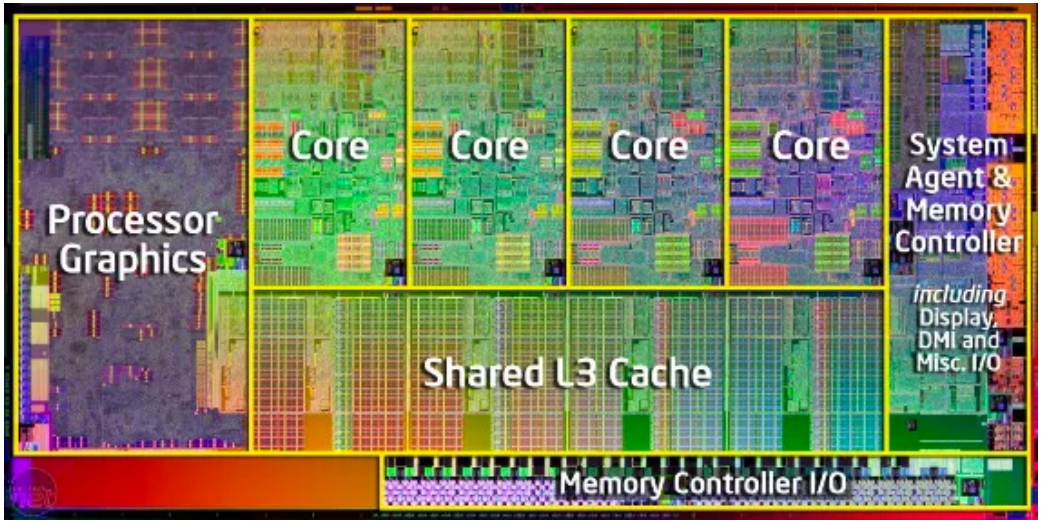
Process	Arrival Time	Start Time	Completion Time	Response Time	Turnaround Time
P1	0	0	20	0	20
P2	0	5	60	5	60
P3	0	10	30	10	30
			Avg:	5	36.7

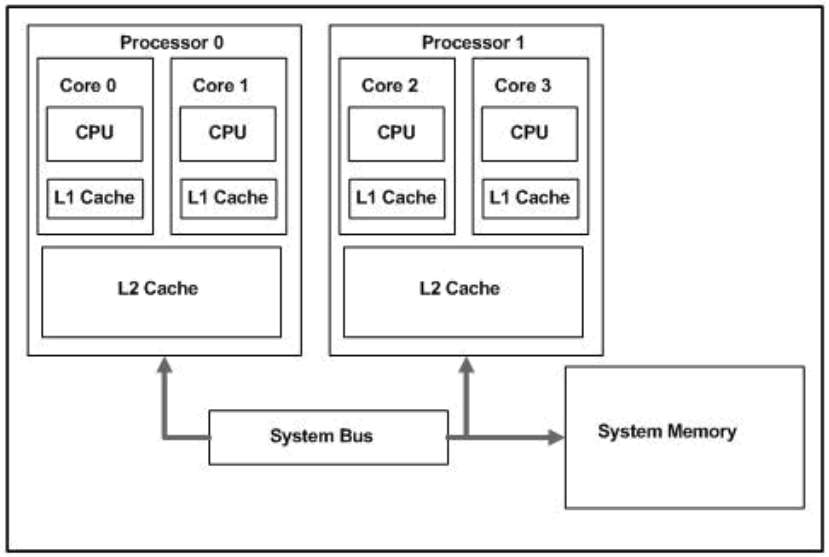
(SJF Avg: 10 & 30; SCTF Avg: 0 & 26.7)

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

Concurrency

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





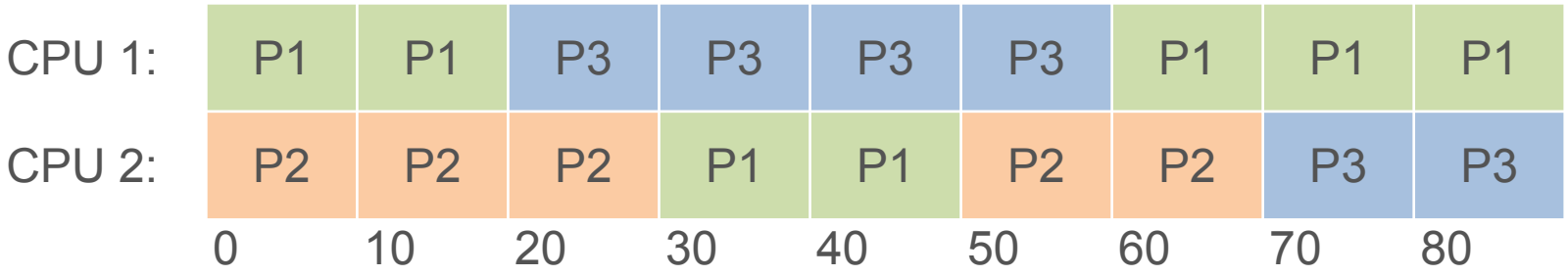
Concurrency

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

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

Concurrency

- Scheduling for multiprocessing/multicore is more complex
- **Load Balancing:** Ensuring different cores/proc. are kept roughly equally busy, i.e., reduce **idle times**
- Multi-queue multiprocessor scheduling (MQMS) is common
  - Each 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.

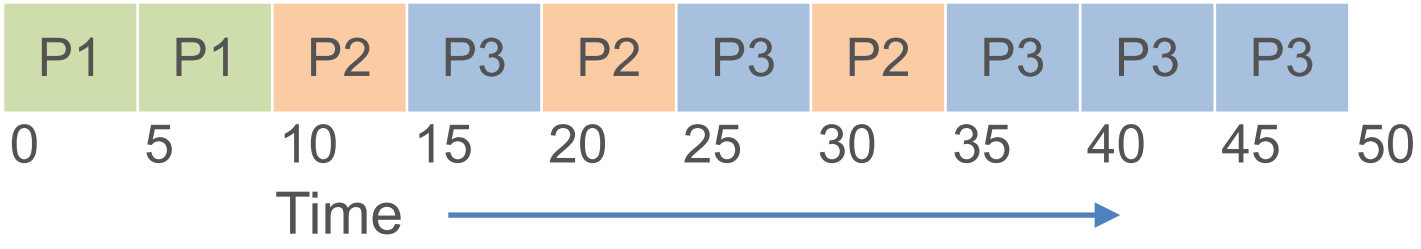
Review Questions

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
SAVINGS

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

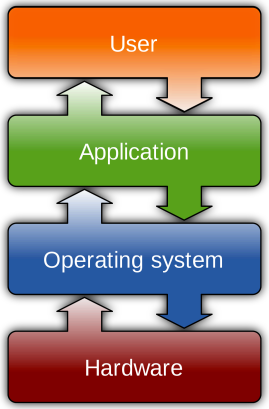
Filesystem

- 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
- **lseek**(): Position offset in file’s fd (for random R/W later)
- Dozens more (rename, mkdir, chmod, etc.)

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

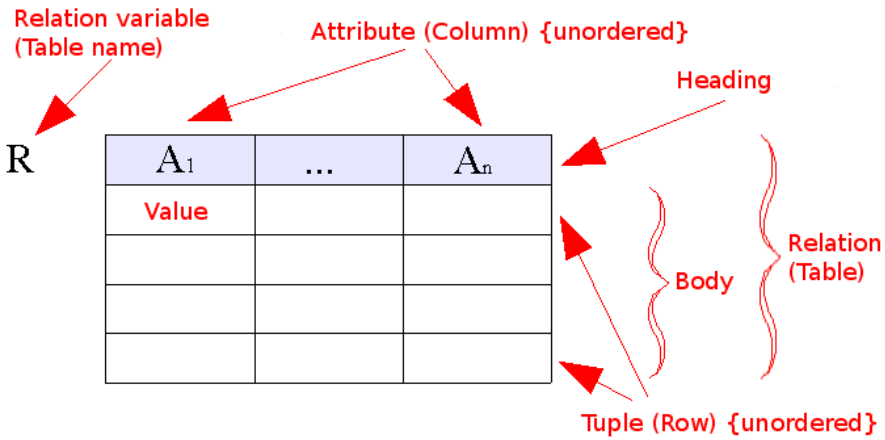
Files Vs Databases: Data Model

- **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

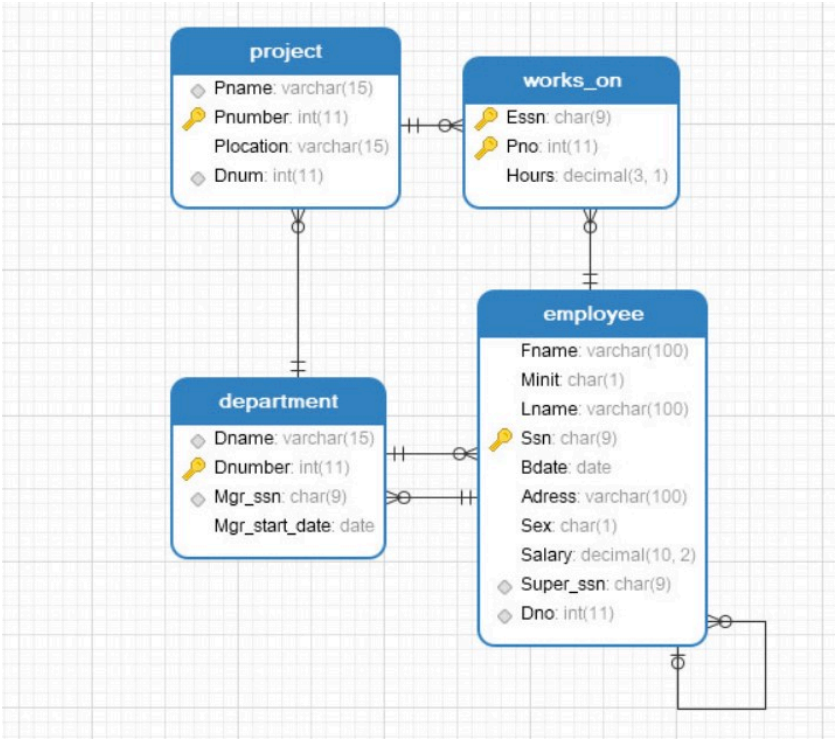
Data as File: Structured

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

Relation



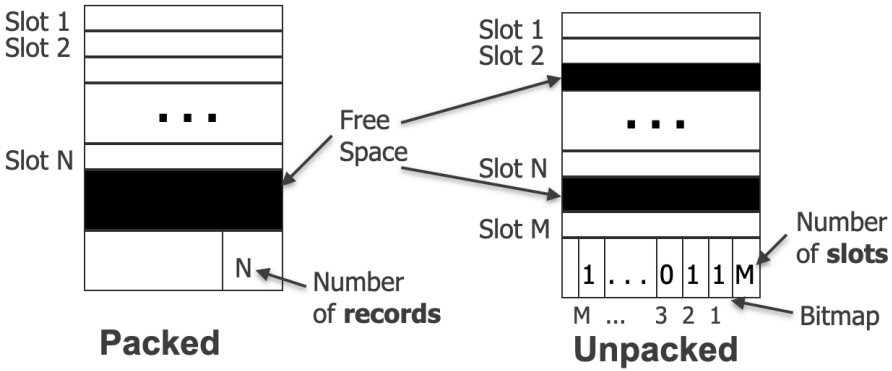
Relational Database

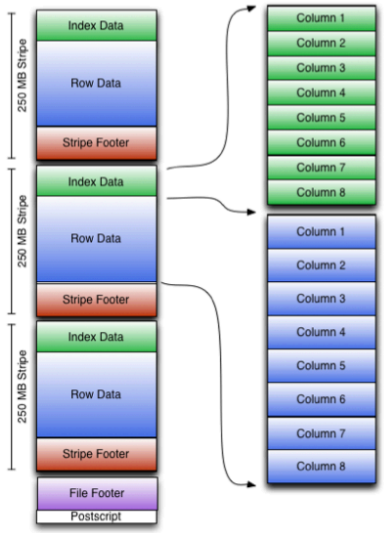


- Most RDBMSs and Spark serialize a relation as *binary* file(s), often compressed

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

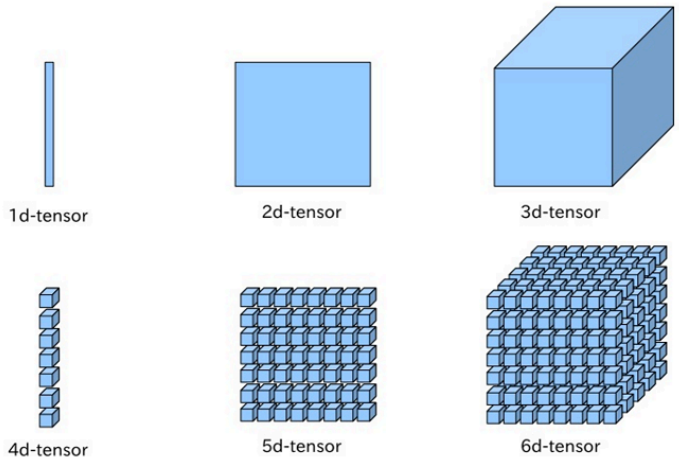
Data as File: Structured

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

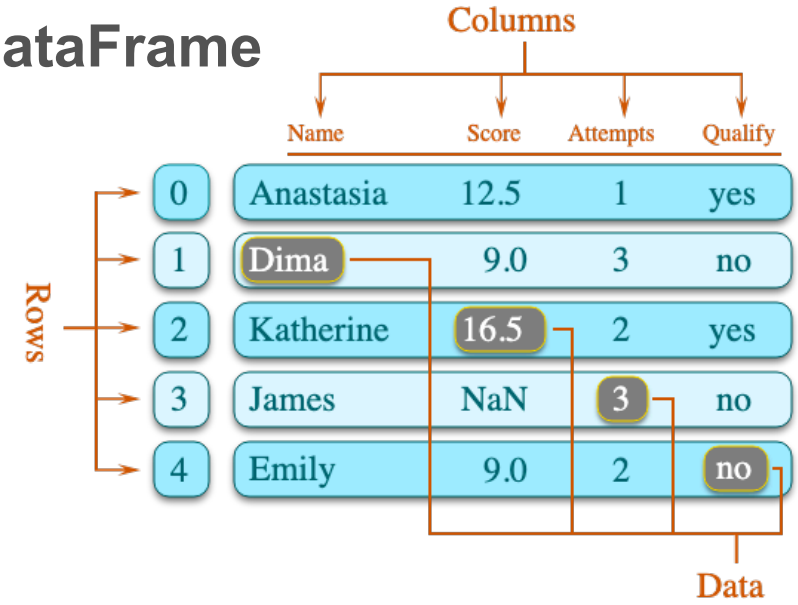
Matrix

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

Tensor



DataFrame

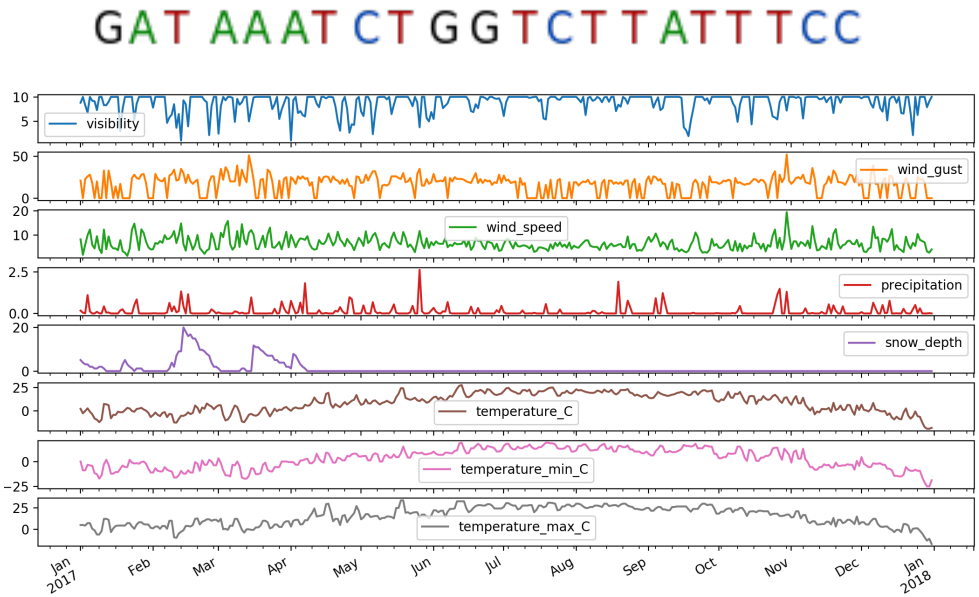


- Typically serialized as restricted ASCII text file (TSV, CSV, etc.)
- Matrix/tensor as binary too
- Can layer on Relations too!

Data as File: Structured

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

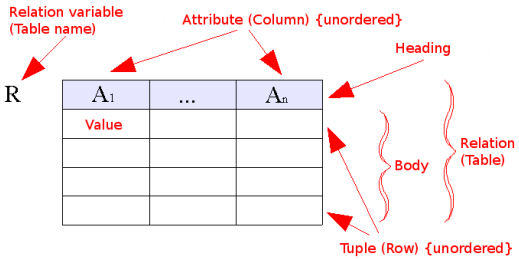
Sequence (Includes Time-series)

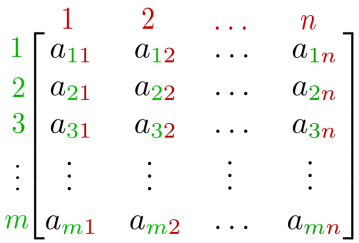


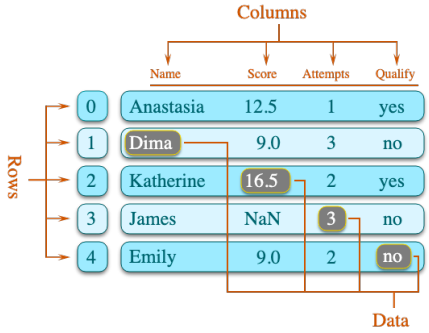
- Can layer on Relations, Matrices, or DataFrames, or be treated as first-class data model
- Inherits flexibility in file formats (text, binary, etc.)

Comparing Struct. Data Models

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







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

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

Data as File: Semistructured

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

Tree-Structured

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

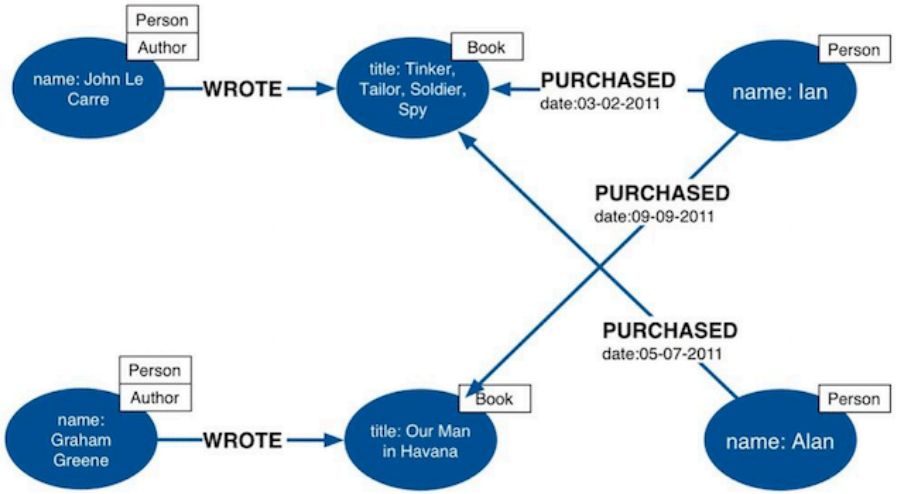
```
[
    {
        orderId: 1,
        date: '1/1/2014',
        orderItems: [
                {itemId: 1, qty: 3, price: 23.4},
                {itemId: 23, qty: 2, price: 3.3},
                {itemId: 7, qty: 5, price: 5.3}
            ]
    },
    {
        orderId: 2,
        date: '1/2/2014',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

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

Data as File: Semistructured

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

Graph-Structured

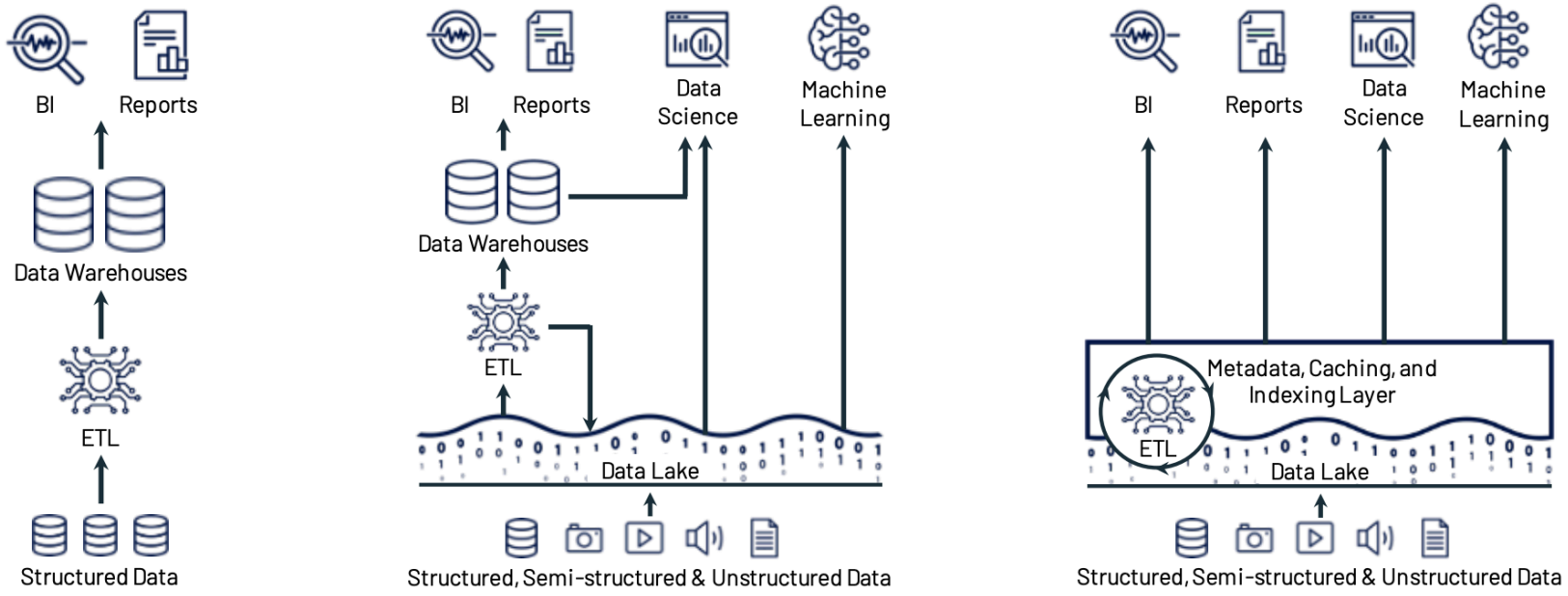


- Typically serialized with JSON or similar textual formats
- Some data systems also offer binary file formats
- Again, can layer on Relations too

Ad: Take DSC 104 for more on semistructured data

Data Files on Data "Lakes"

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



(a) First-generation platforms. **(b) Current two-tier architectures.** **(c) Lakehouse platforms.**

If interested, check out this vision paper on the future of data lakes:
http://cidrdb.org/cidr2021/papers/cidr2021_paper17.pdf

Data Lake File Format Tradeoffs

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

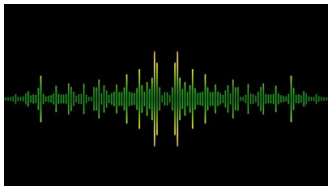
Data Lake File Format Tradeoffs

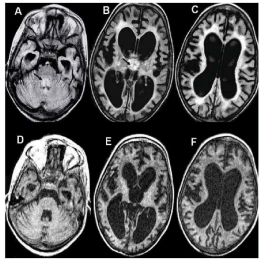
Dataset	Size on Amazon S3	Query Run Time	Data Scanned	Cost
Data stored as CSV files	1 TB	236 seconds	1.15 TB	$5.75
Data stored in Apache Parquet Format	130 GB	6.78 seconds	2.51 GB	$0.01
Savings	87% less when using Parquet	34x faster	99% less data scanned	99.7% savings

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

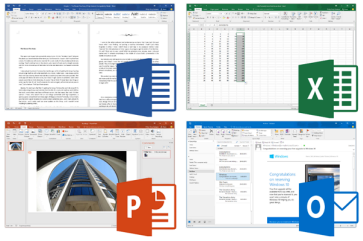
Data as File: Other Common Formats

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





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



Breakout Rooms Activity (8min)

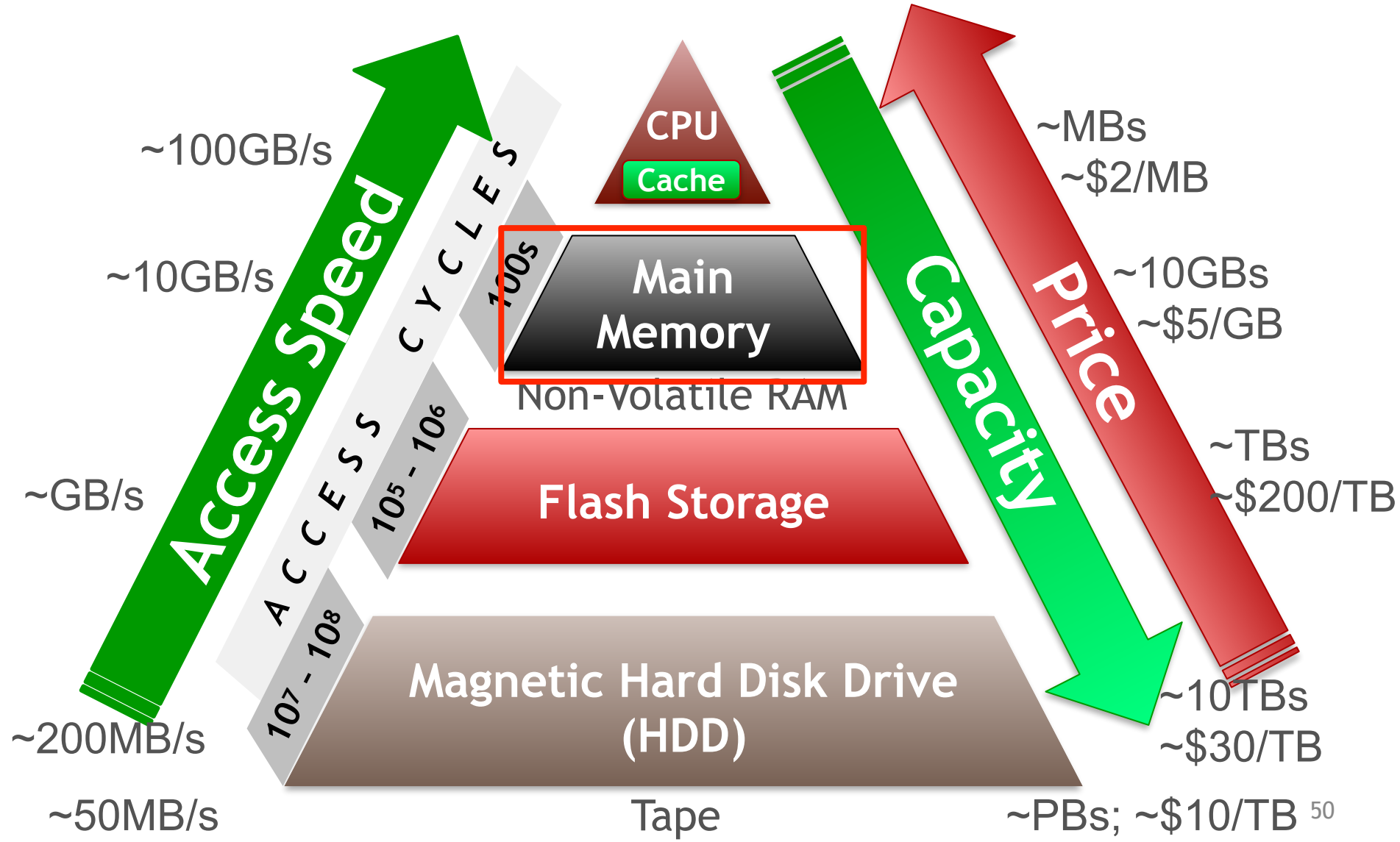
In the real world, Data Science is a team sport. The ability to learn from your wider community of peers/colleagues at a higher level than your specific implementation goals is a crucial skill you must pick up. Given this, answer the following about PA0:

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

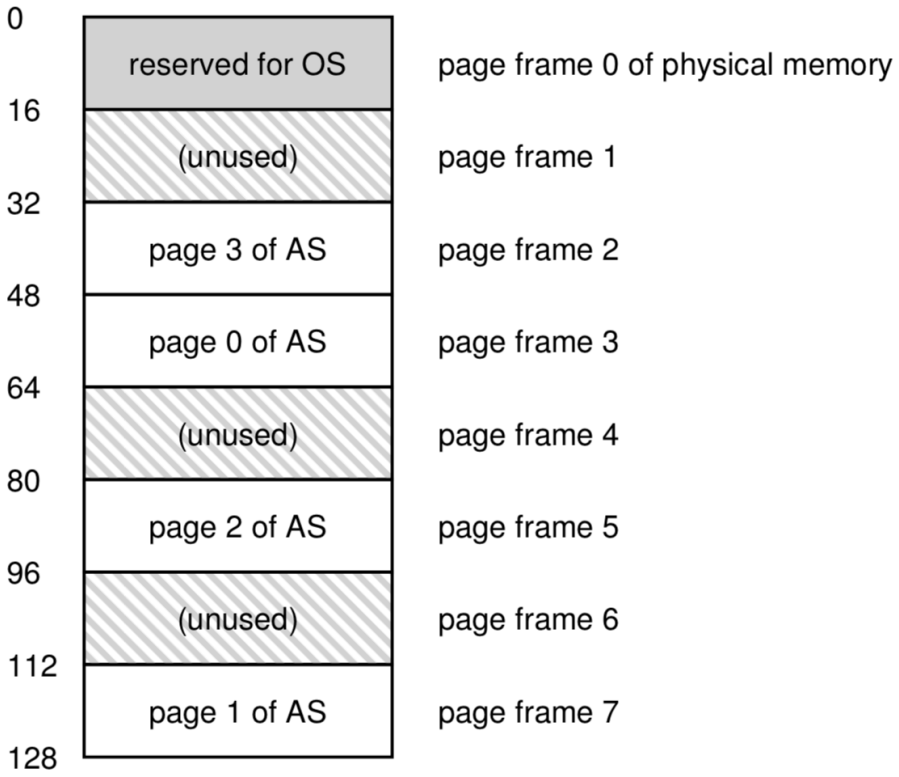
Memory/Storage Hierarchy



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



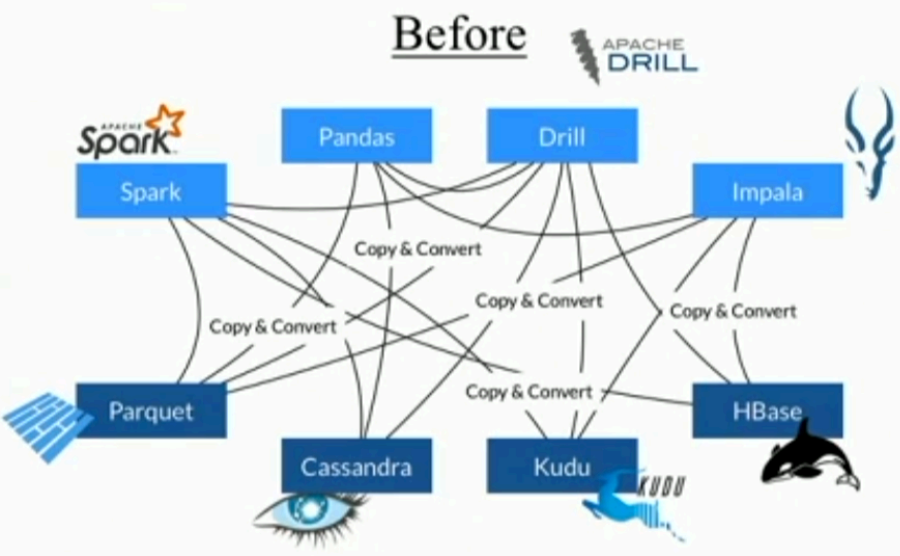
Apportioning of DRAM

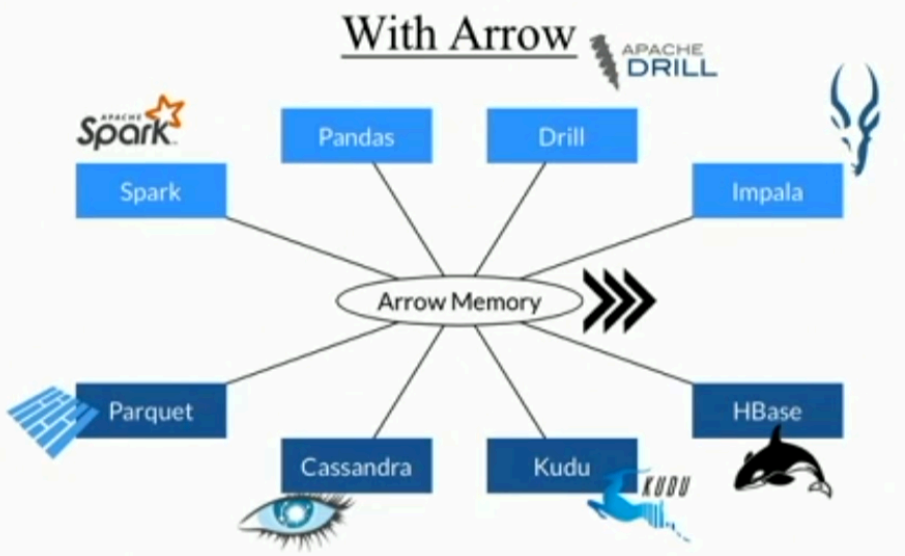
- A process' **Address Space:**
  - Slice of virtualize DRAM assigned to it alone!
  - OS "translates" DRAM vs disk address
- **Page Replacement Policy:**
  - When DRAM fills up, which cached page to evict?
  - Many policies in OS literature
- **Memory Leaks:**
  - Process forgot to "free" pages used 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

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.





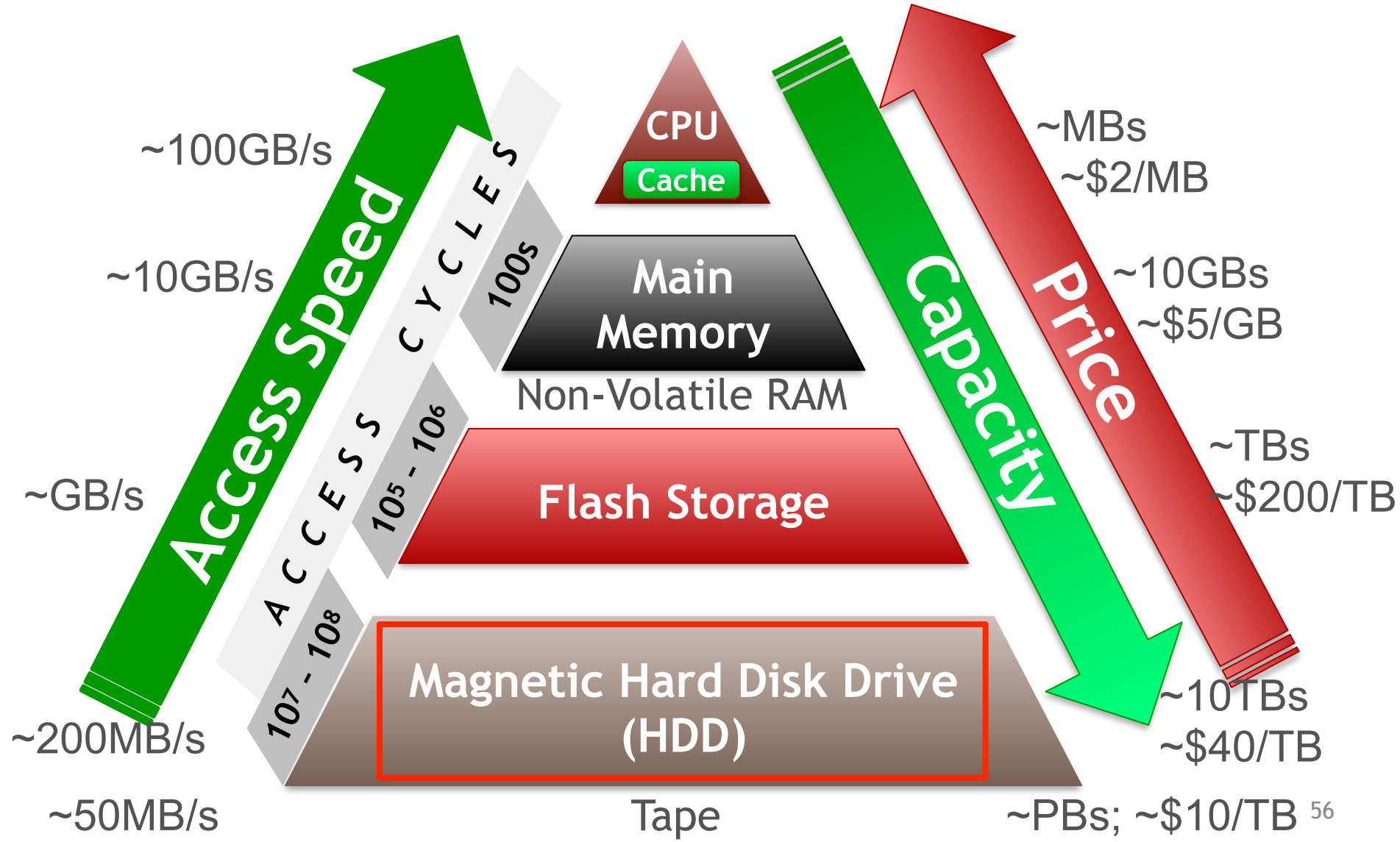
Outline

- 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

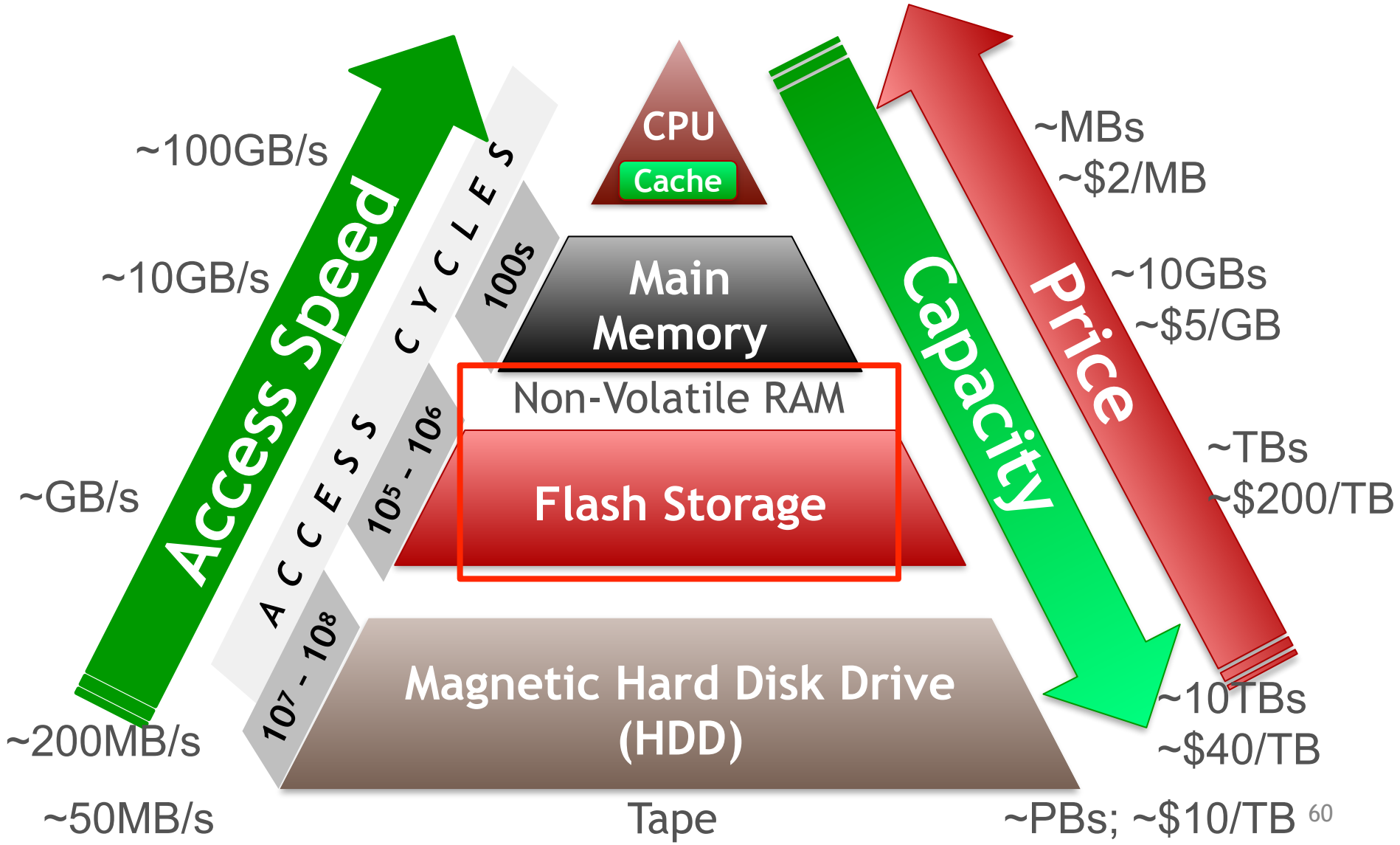
- Disk space is organized into **files**
- Files are made up of disk **pages** aka **blocks**
- Typical disk block/page size: 4KB or 8KB
  - Basic unit of reads/writes for a disk
  - OS/RAM page is *not* the same as disk page!
  - Typically, OS/RAM page size = disk page size but not always; disk page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

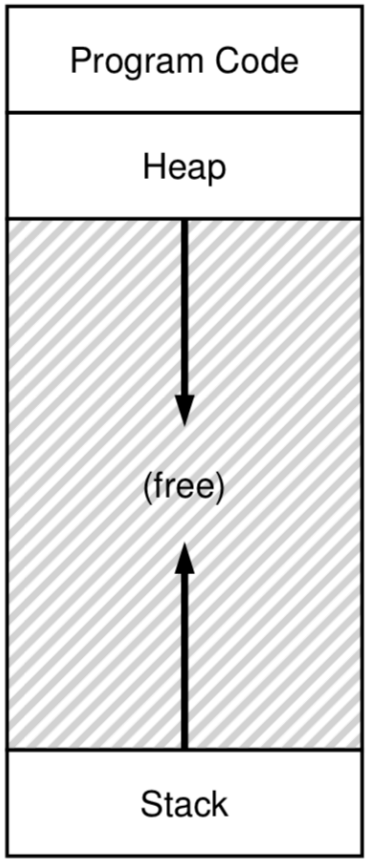
Outline

- 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



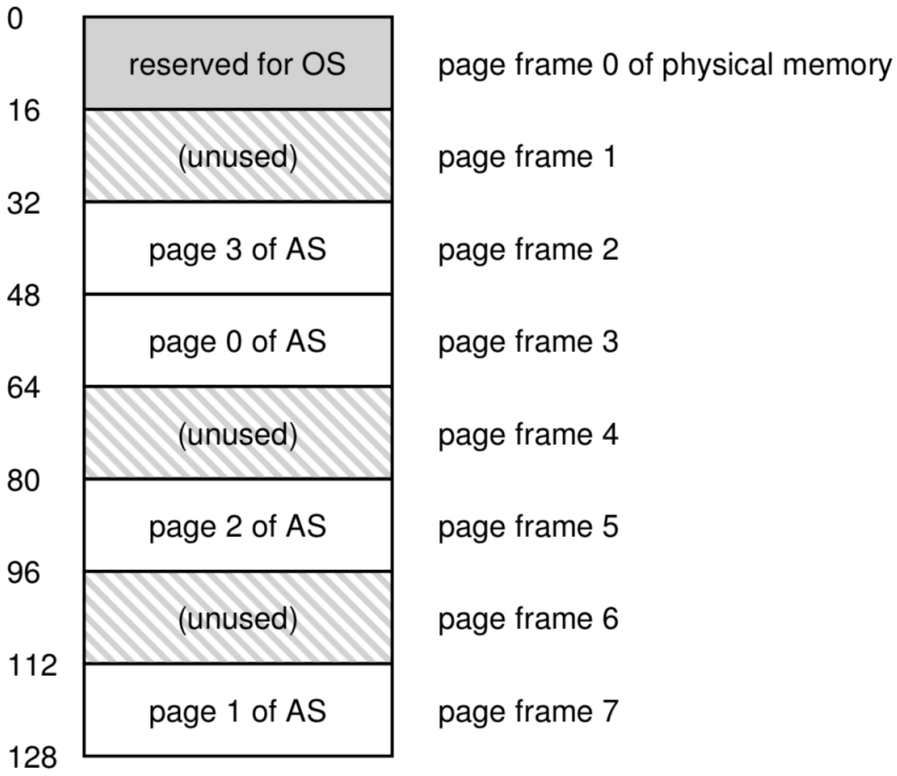
Virtual Memory

- **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

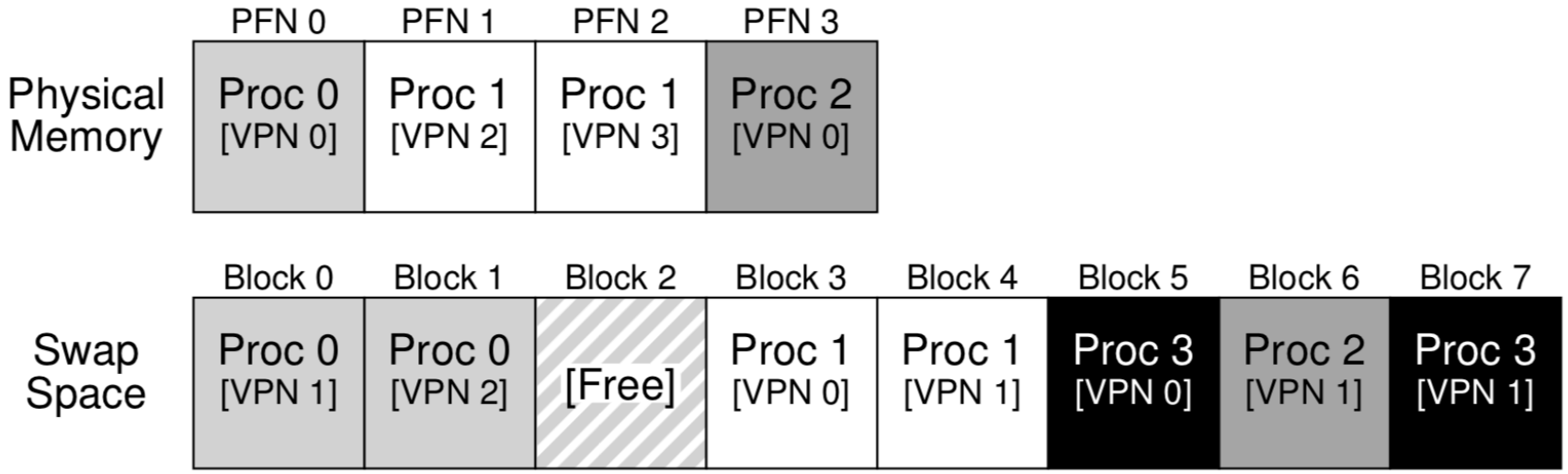
- **Page**: An abstraction of *fixed* size chunks of storage
  - Makes it easier to manage memory virtualization
- **Page Frame**: A virtual "slot" in DRAM to hold a page

- 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

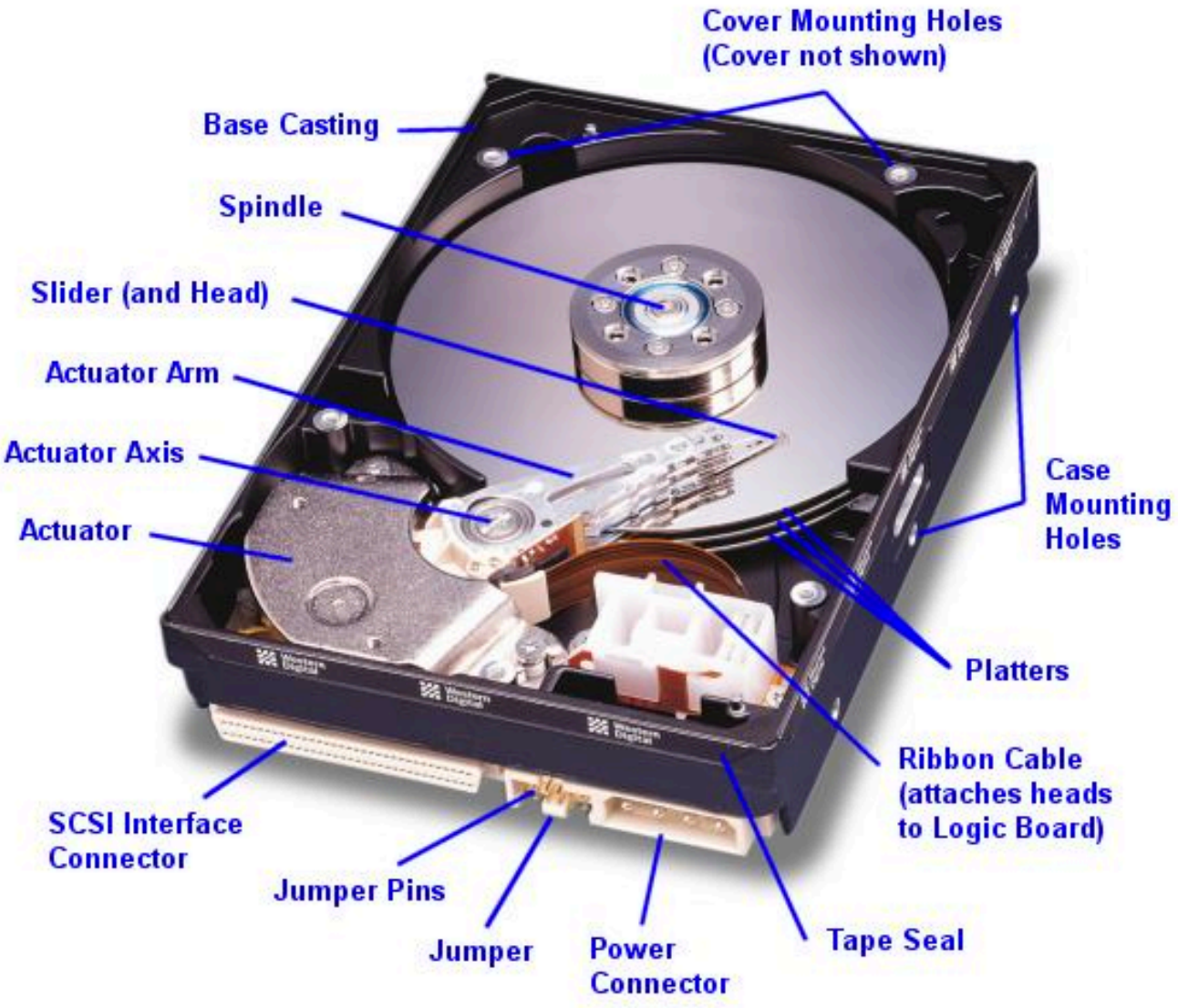


Page Replacement

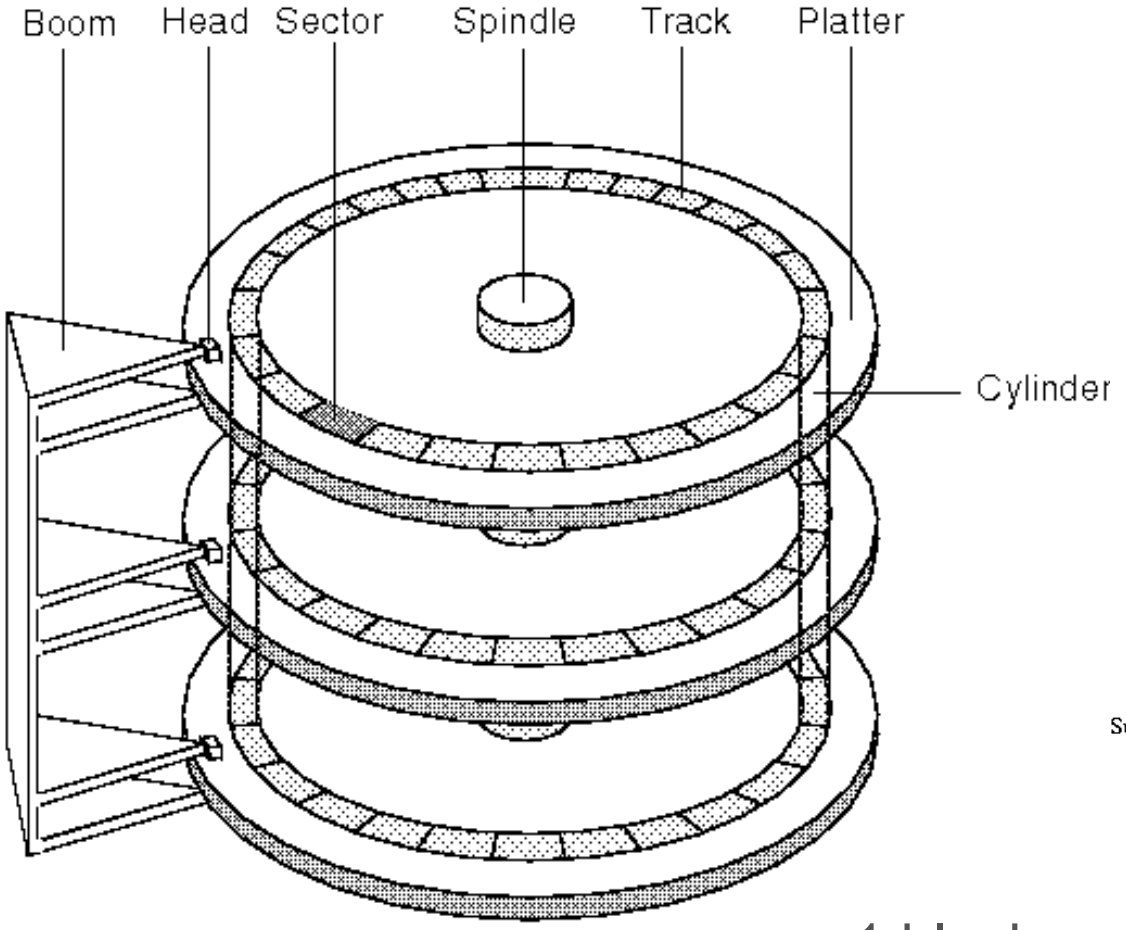
- Recall DRAM has page frames to hold page content; a process's address space may only have so many frames
- **Page Fault**: A page required by process is not in DRAM
  - OS intervenes to read page from disk to DRAM
  - If free page frame available in DRAM, all good
- **Page Replacement**: If no frame 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* tadeoffs: FIFO, MRU, LRU, etc. (later topic)

Optional: More on Magnetic Hard Disks
Not included in syllabus

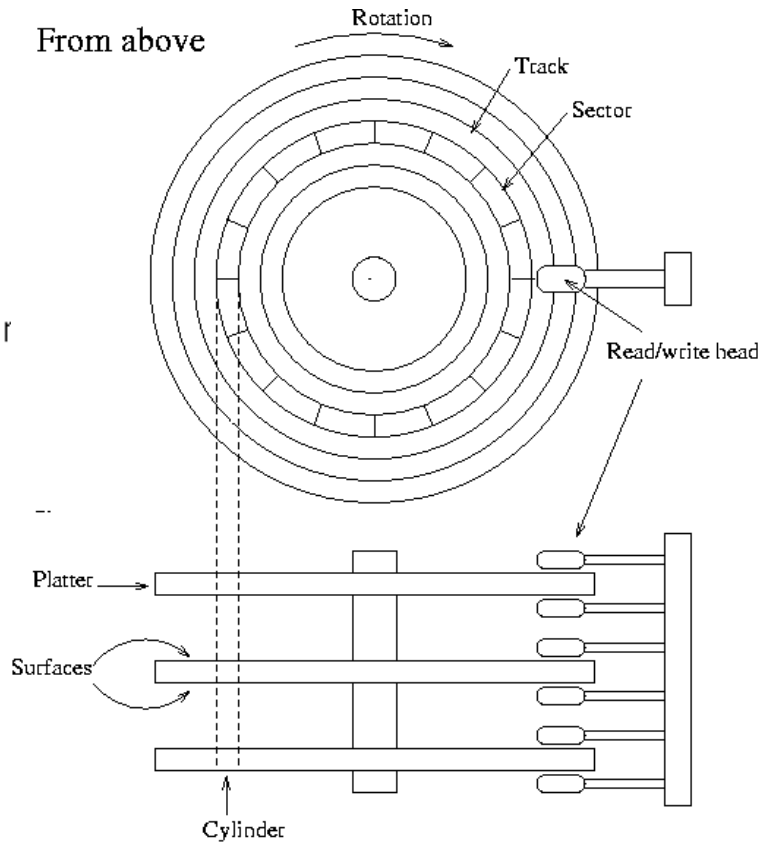
Components of a Disk



Components of a Disk

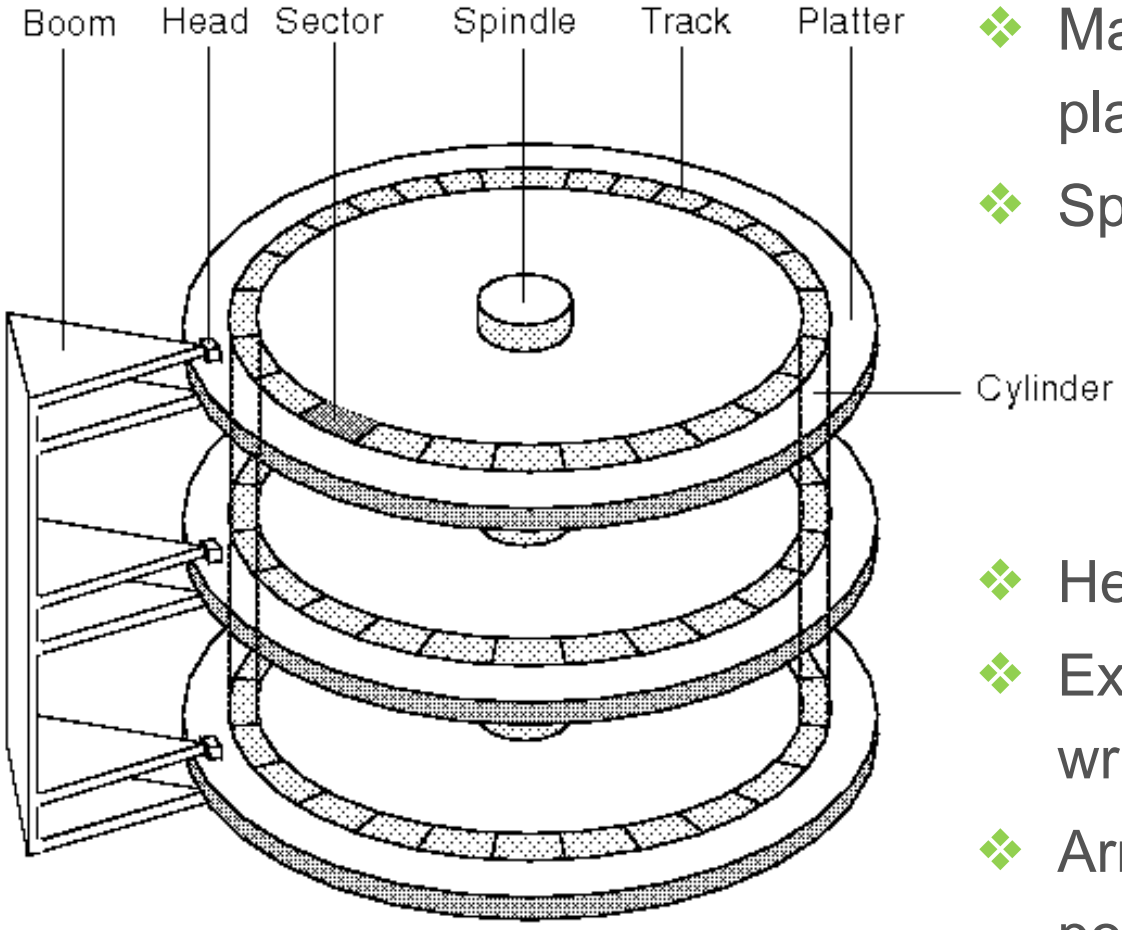


Anatomy of a regular hard disk



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

How does a Disk Work?



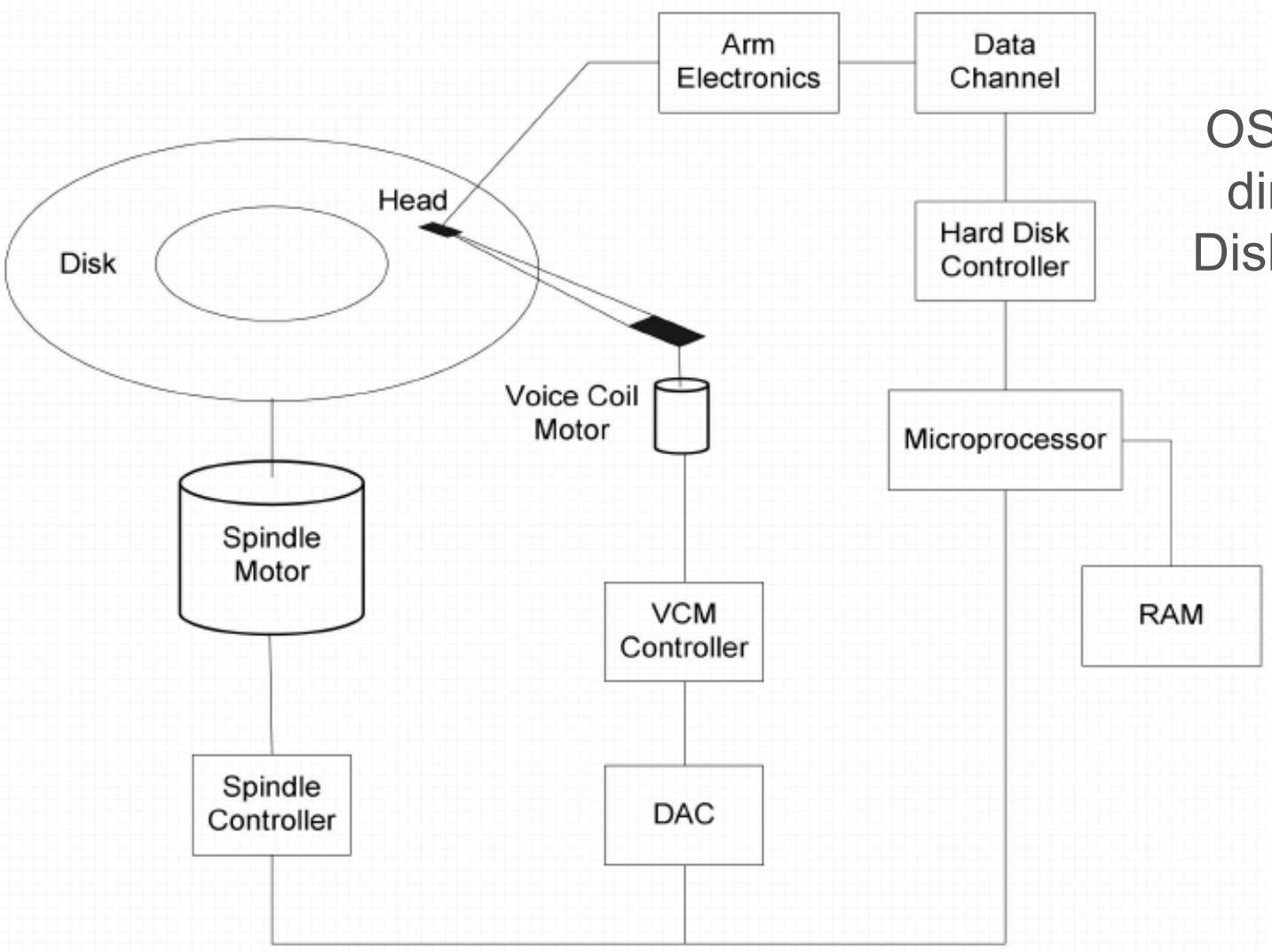
Anatomy of a regular hard disk

- 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

UC San Diego

DSC 102
Systems for Scalable Analytics

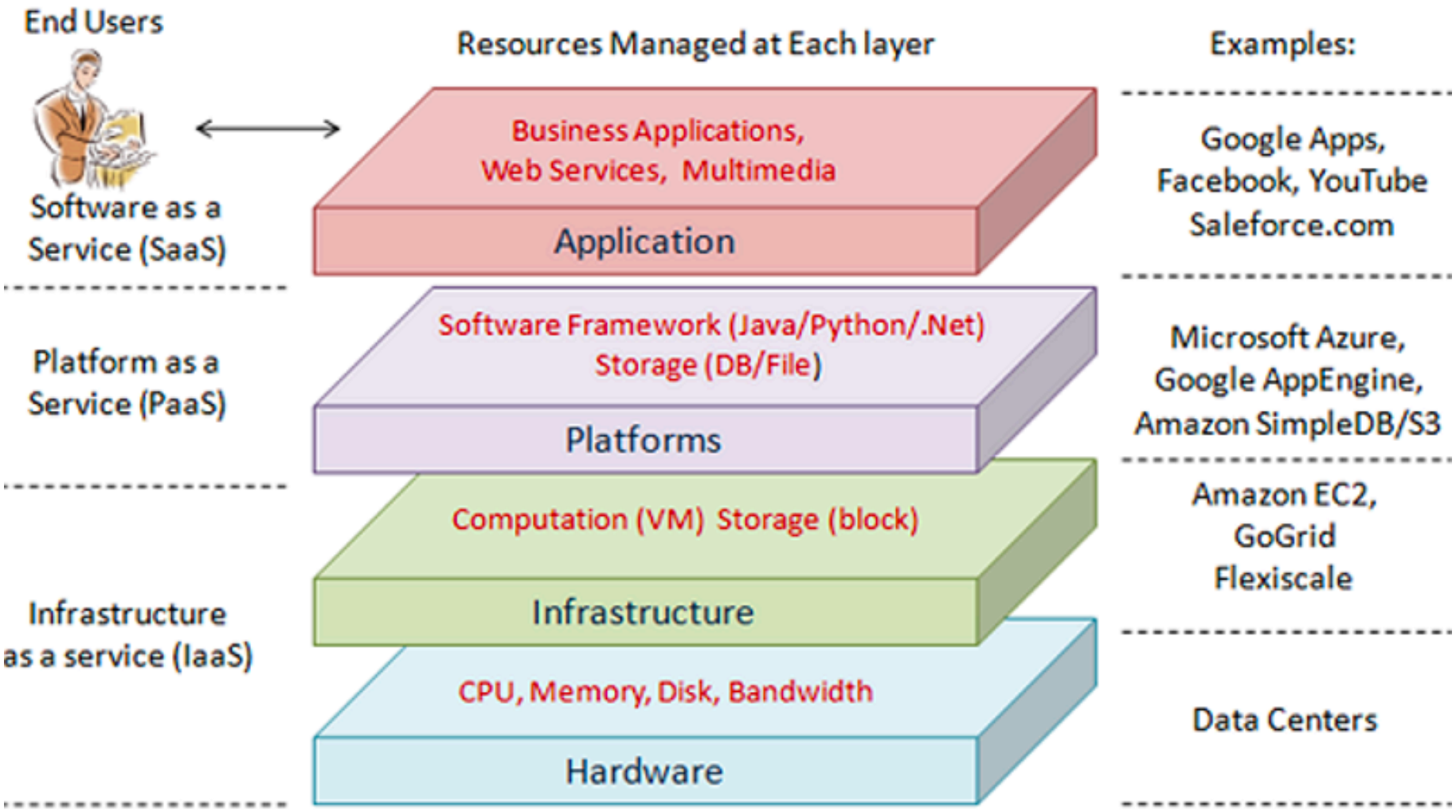
Arun Kumar

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



https://benchpartner.com/layered-model-of-cloud-computing

Examples of AWS Cloud Services

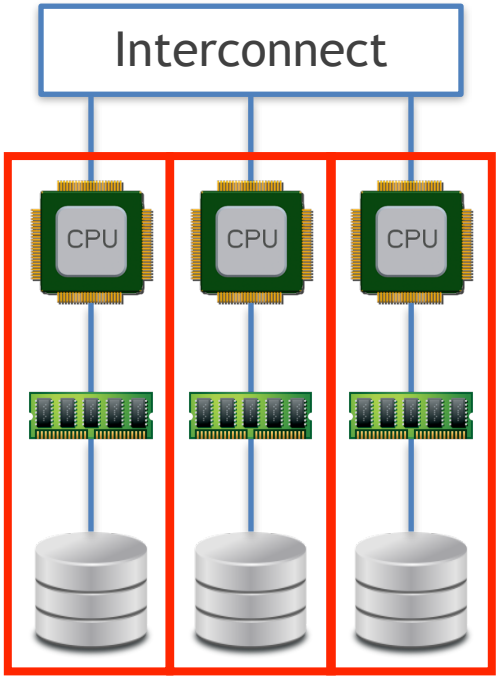
- **IaaS**:
  - **Compute**: EC2, ECS, Fargate, Lambda
  - **Storage**: S3, EBS, EFS, Glacier
  - **Networking**: CloudFront, VPC
- **PaaS**:
  - **Database/Analytics Systems**: Aurora, Redshift, Neptune, ElastiCache, DynamoDB, Timestream, EMR, Athena
  - **Blockchain**: QLDB; **IoT**: Greengrass
- **SaaS**: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

Evolution of Cloud Infrastructure

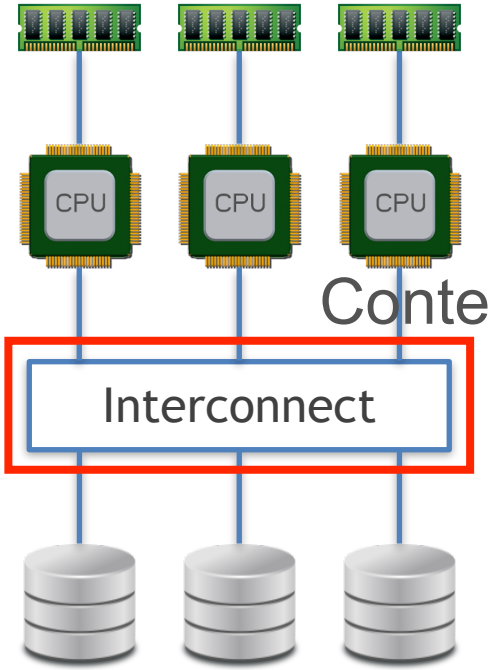
- **Data Center**: Physical space from which a cloud is operated
- **3 generations of data centers/clouds:**
  - **Cloud 1.0 (Past)**: Networked servers; user rents servers (time-sliced access) needed for data/software
  - **Cloud 2.0 (Current)**: “Virtualization” of networked servers; user rents amount 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

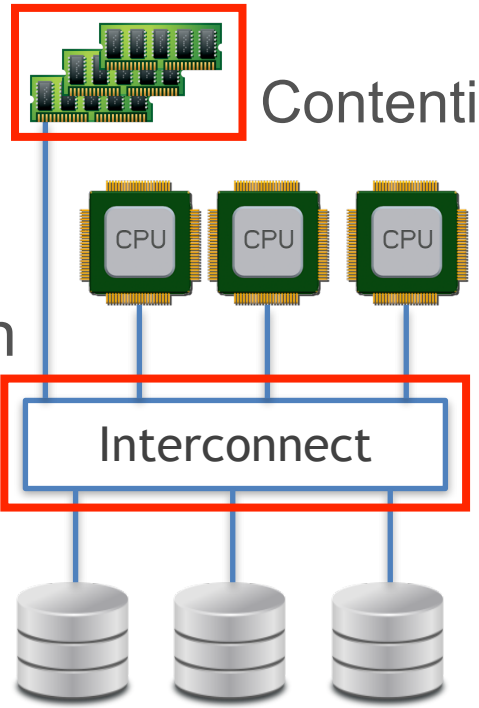


Shared-Nothing Parallelism



Contention

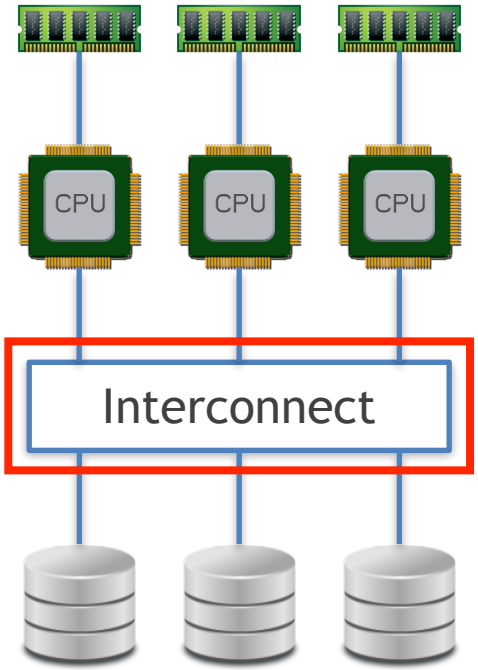
Shared-Disk Parallelism



Shared-Memory Parallelism

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

Revisiting Parallelism in the Cloud

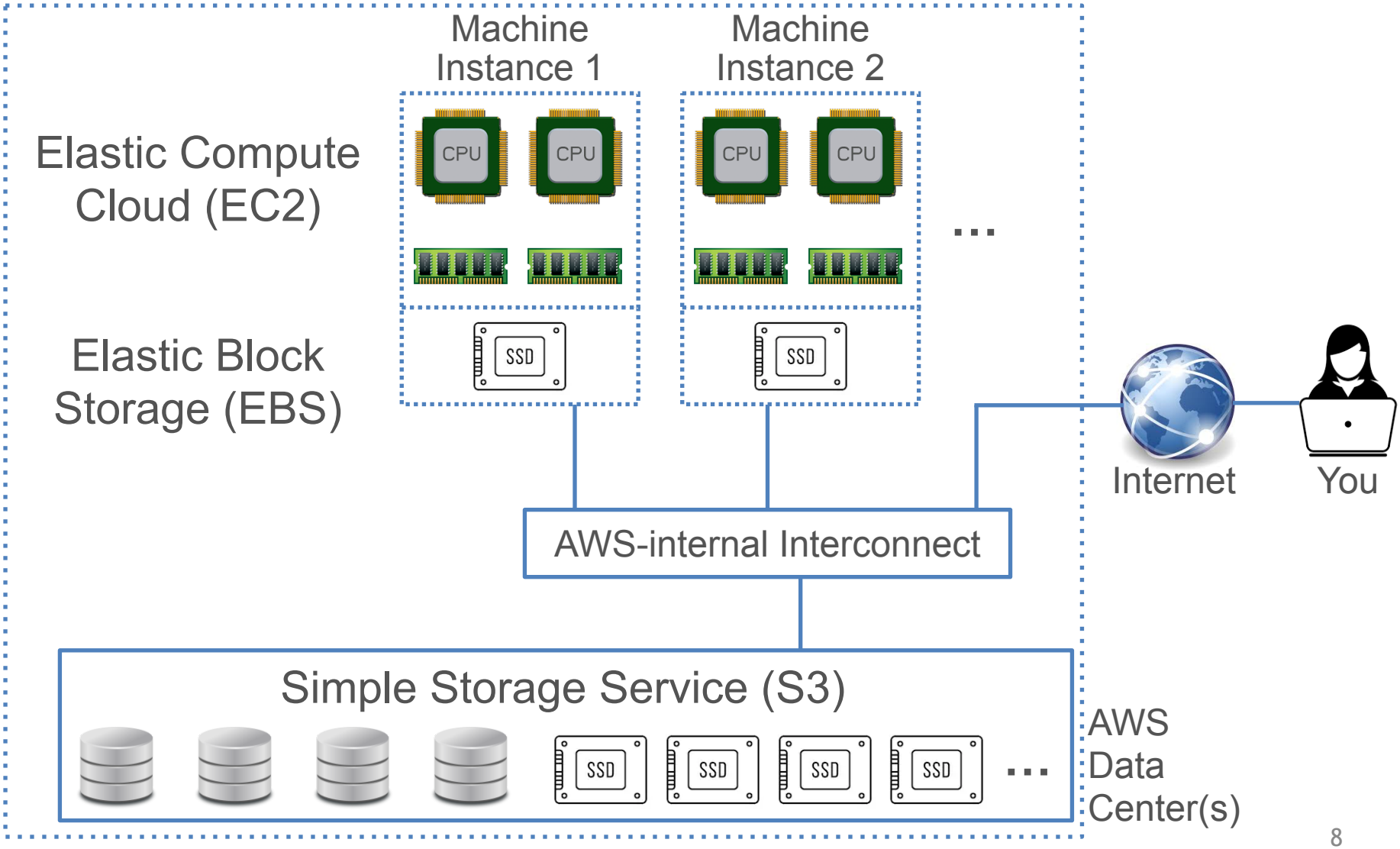


Shared-Disk Parallelism

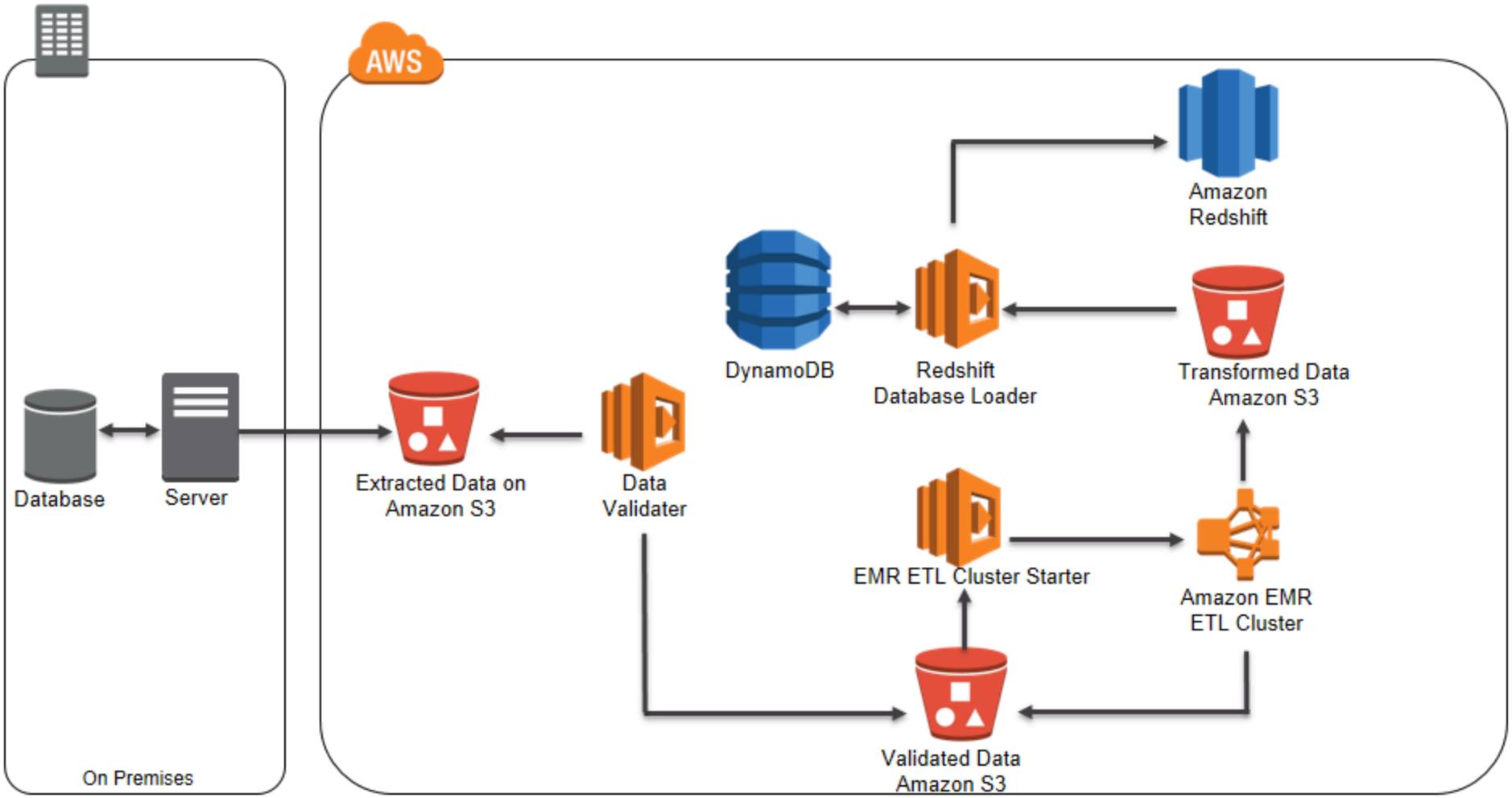
Modern networks in data centers have become much faster: 100GbE to even TbE!

- **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

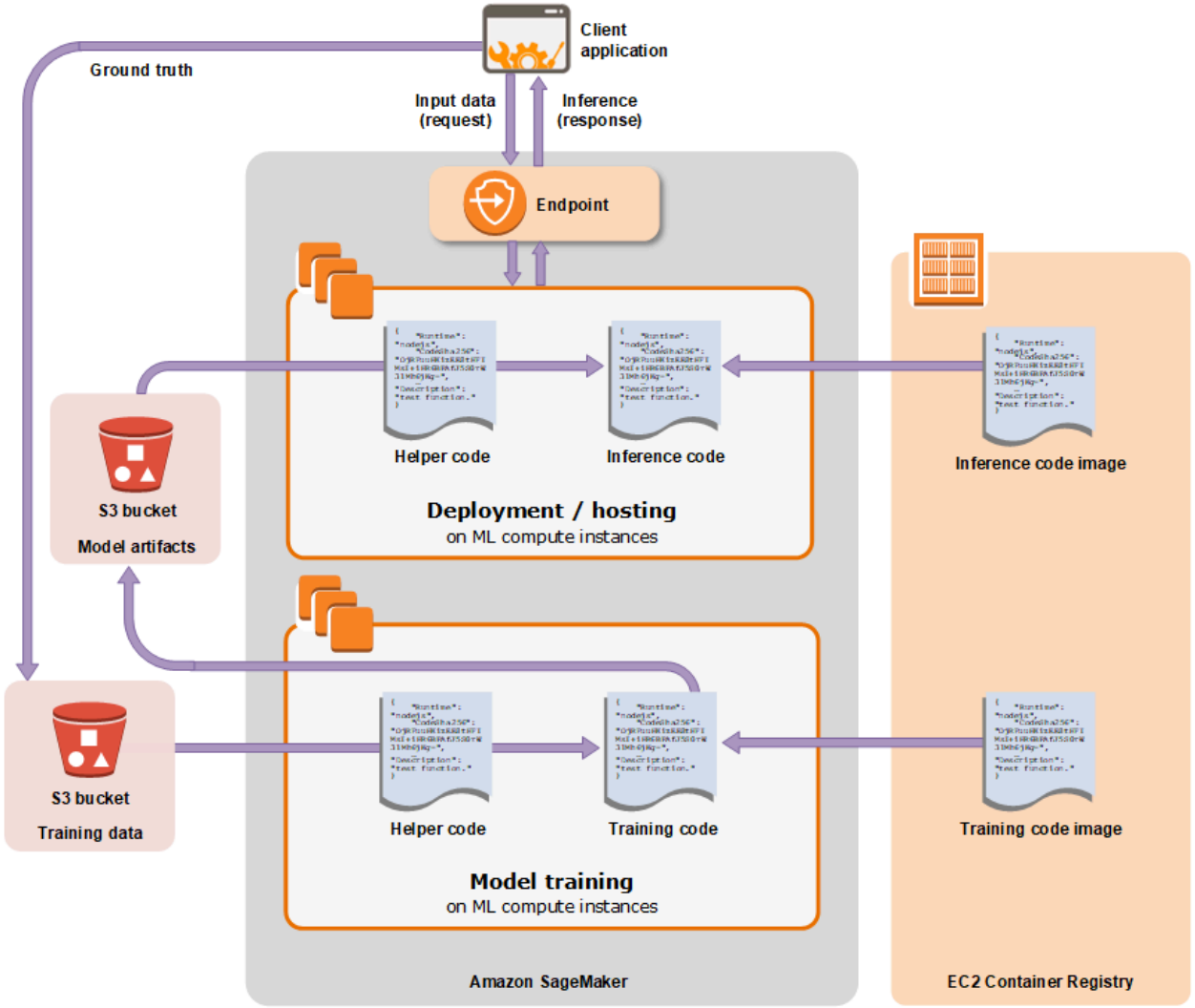


Example: AWS DB/Analytics Services



https://aws.amazon.com/blogs/big-data/automating-analytic-workflows-on-aws/

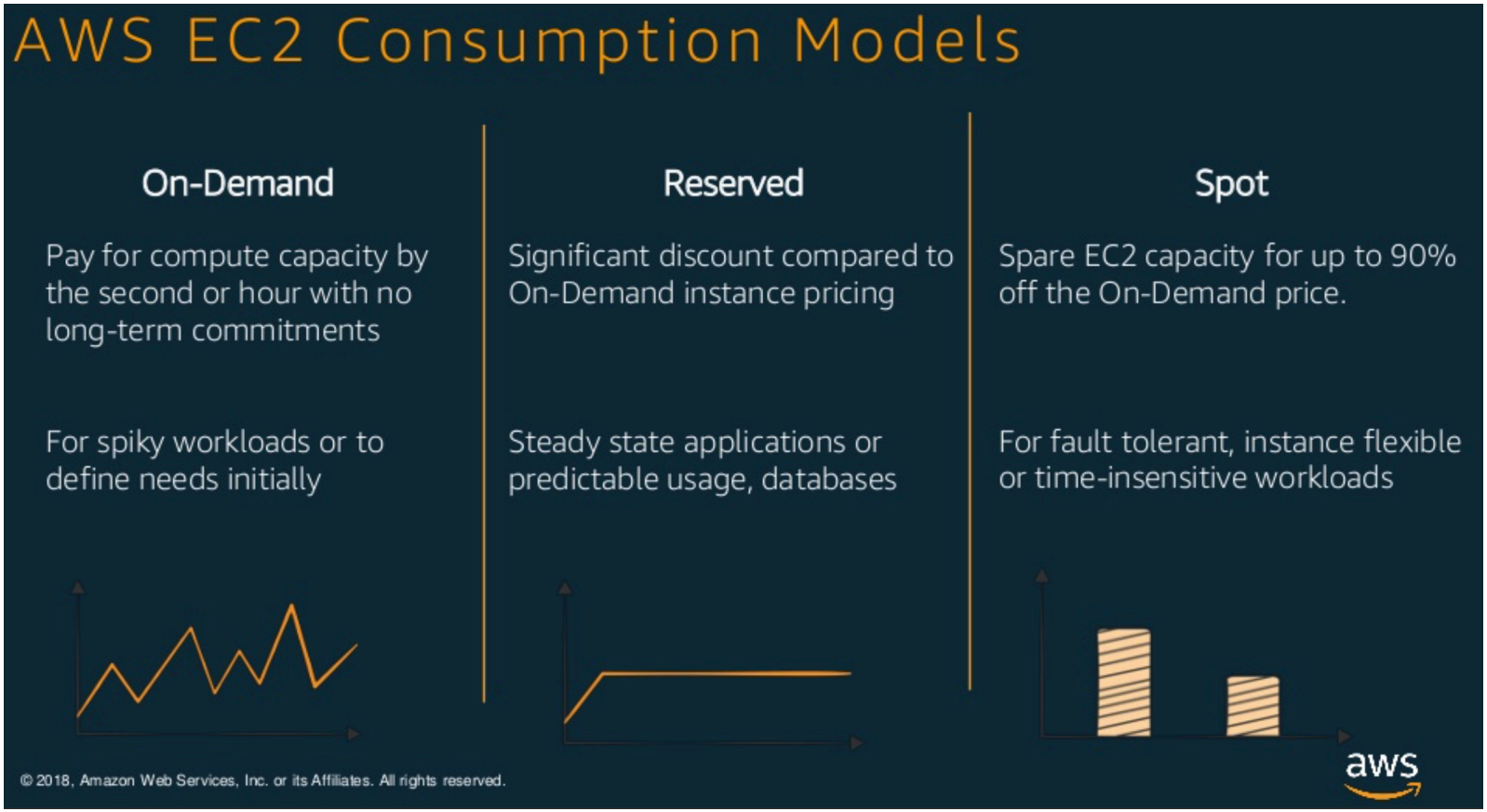
Example: AWS ML Services



https://docs.aws.amazon.com/sagemaker/latest/dg/how-it-works-training.html

New Cloud Renting Paradigms

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



https://www.slideshare.net/AWSUsersGroupBengalu/amazon-ec2-spot-instances

More on Spot vs On-Demand

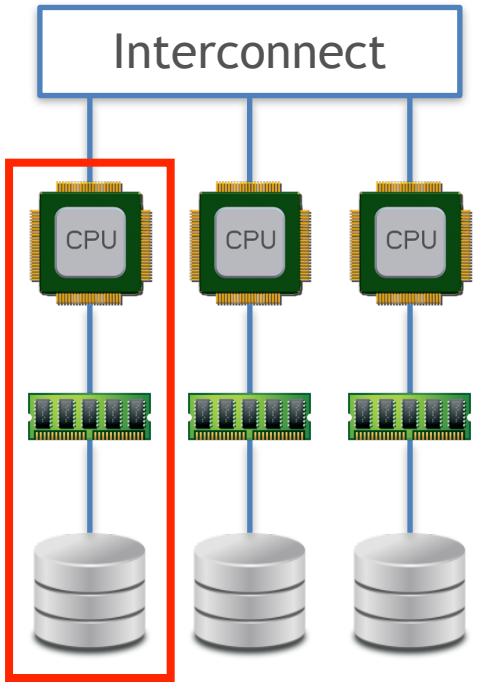
	Spot Instances	On-Demand Instances
Launch time	Can only be launched immediately if the Spot Request is active and capacity is available.	Can only be launched immediately if you make a manual launch request and capacity is available.
Available capacity	If capacity is not available, the Spot Request continues 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 interrupt 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).

https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-spot-instances.html

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)

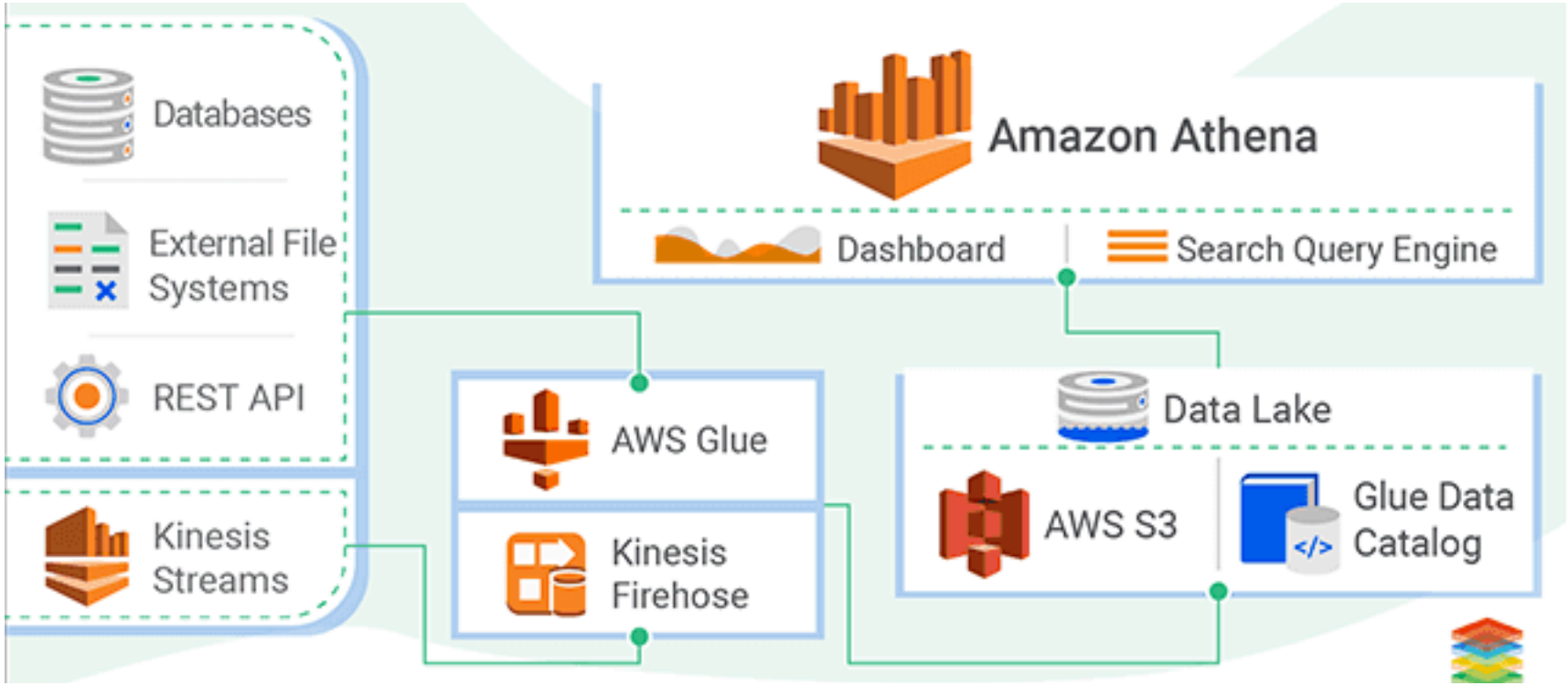
City car-sharing
(Serverless)

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

How much do you use the car?

https://www.slideshare.net/loige/building-a-serverless-company-with-nodejs-react-and-the-serverless-framework-jsday-2017-verona

Example: Serverless RDBMS on AWS



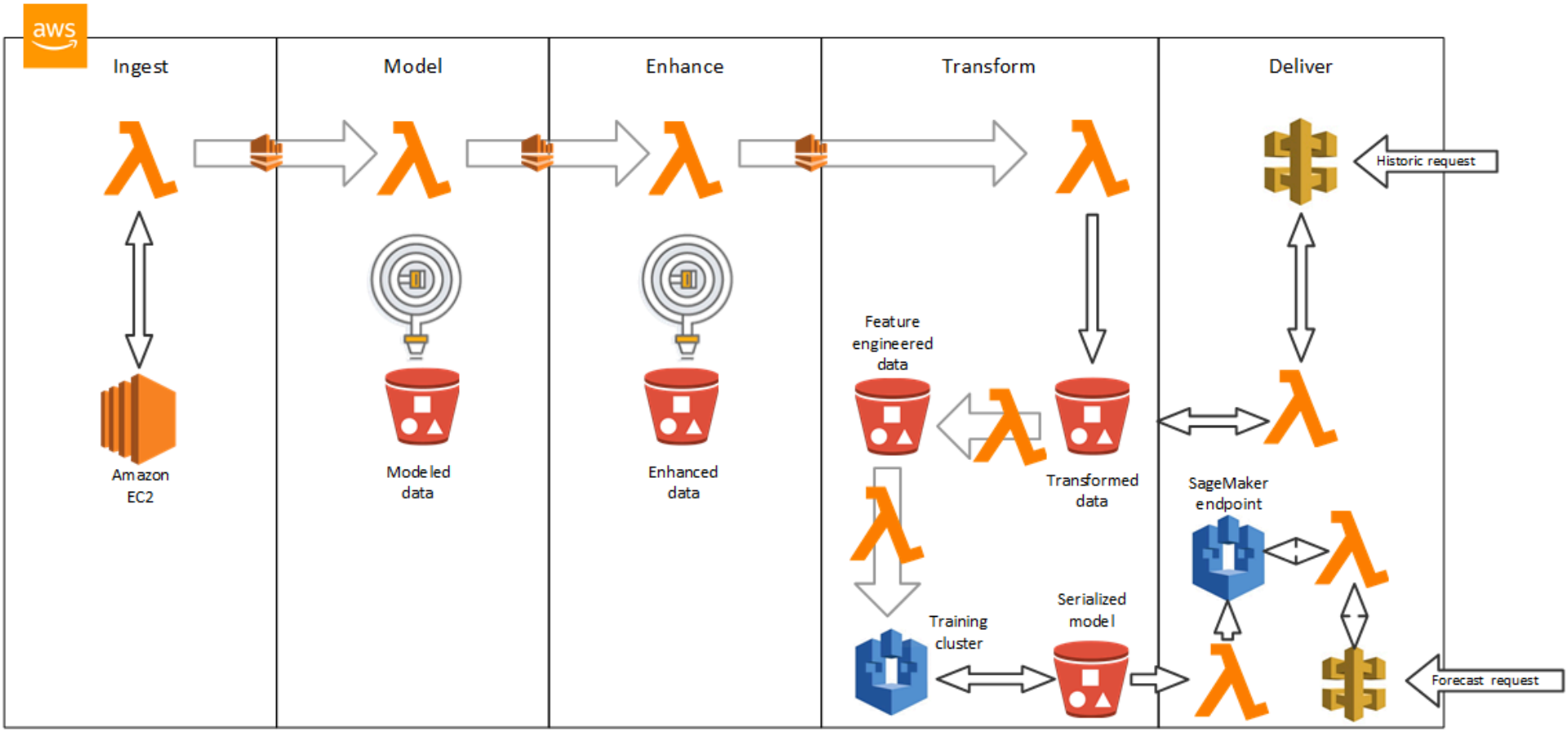
Remote read of data from S3

Schema-on-read
Many data formats

Simple interactive queries

https://www.xenonstack.com/blog/amazon-athena-quicksight/

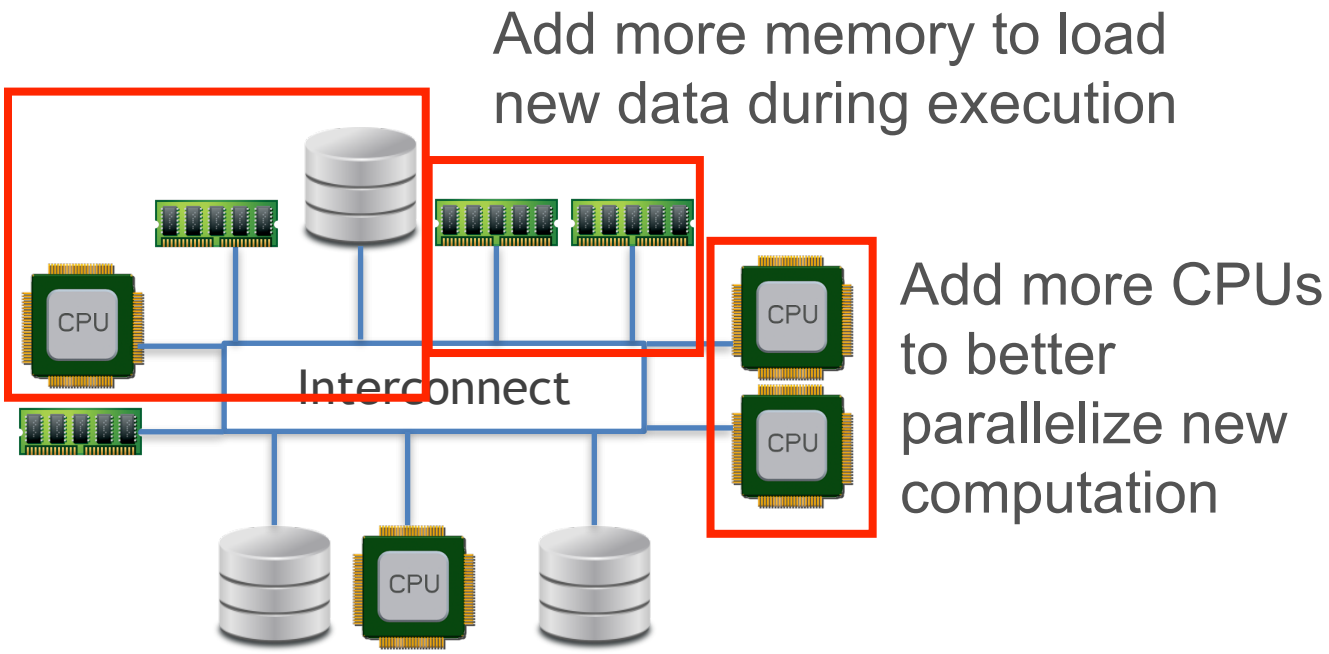
Example: Serverless ML app. on AWS



https://aws.amazon.com/quickstart/architecture/predictive-data-science-sagemaker-and-data-lake/

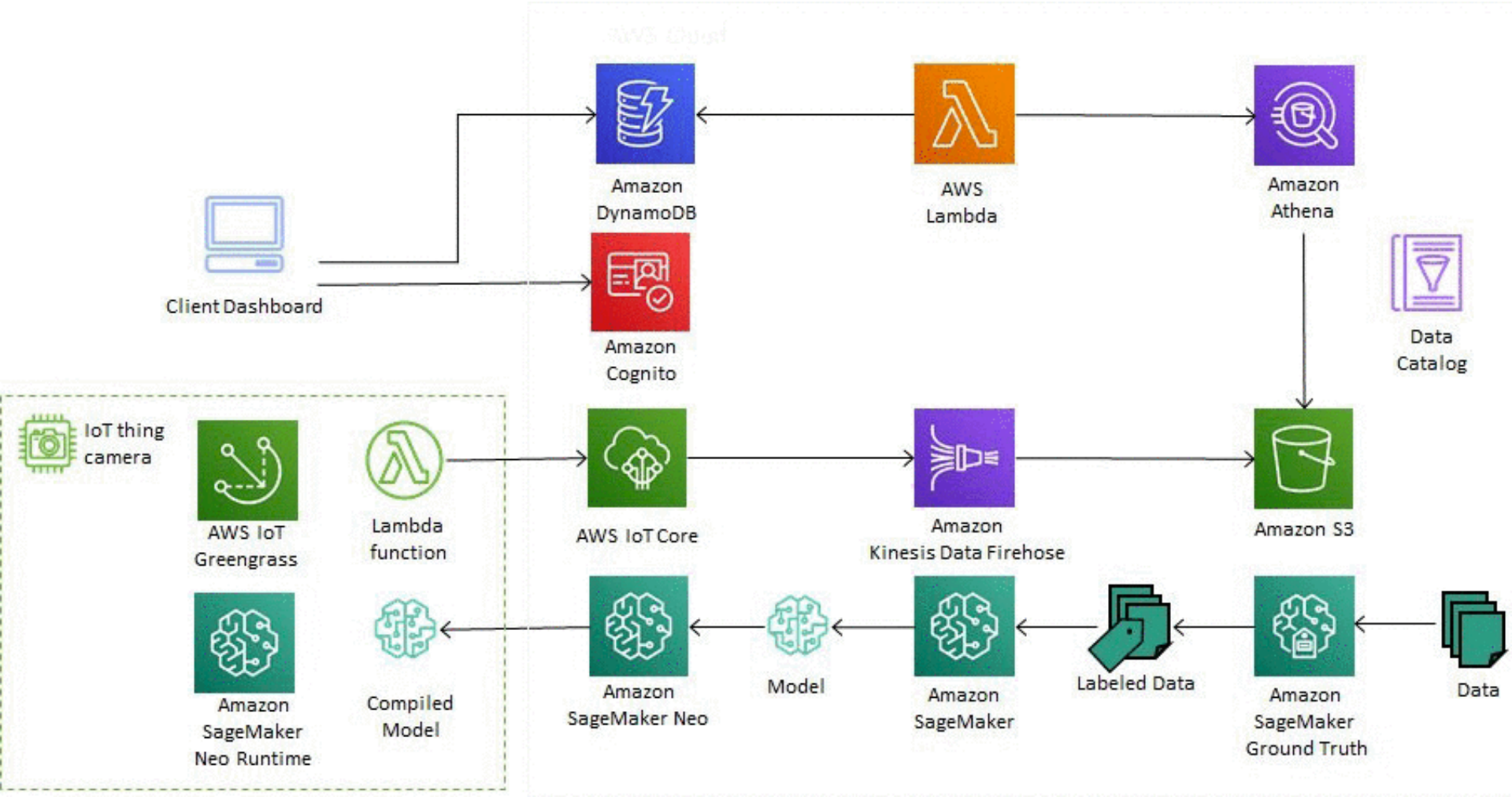
Disaggregation: Glimpse into the Future?

- 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.



https://aws.amazon.com/blogs/machine-learning/amazon-sagemaker-neo-helps-detect-objects-and-classify-images-on-edge-devices/

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

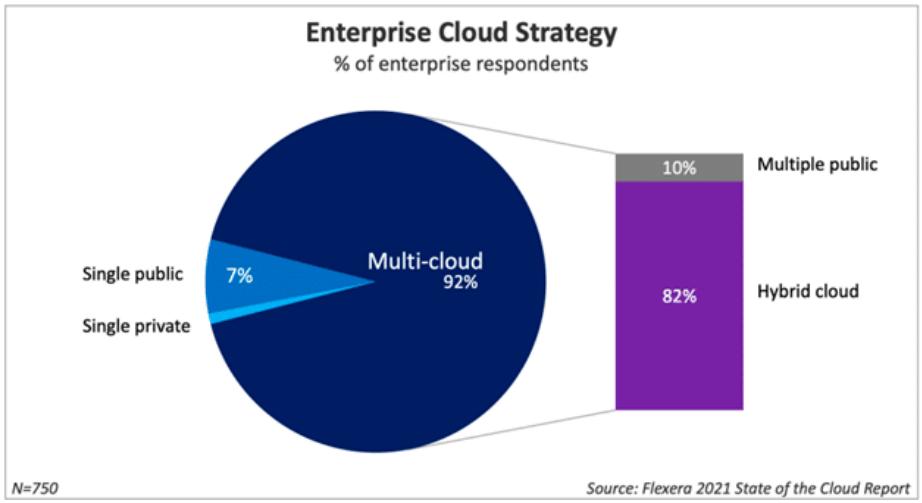
IMMEDIATE RELEASE

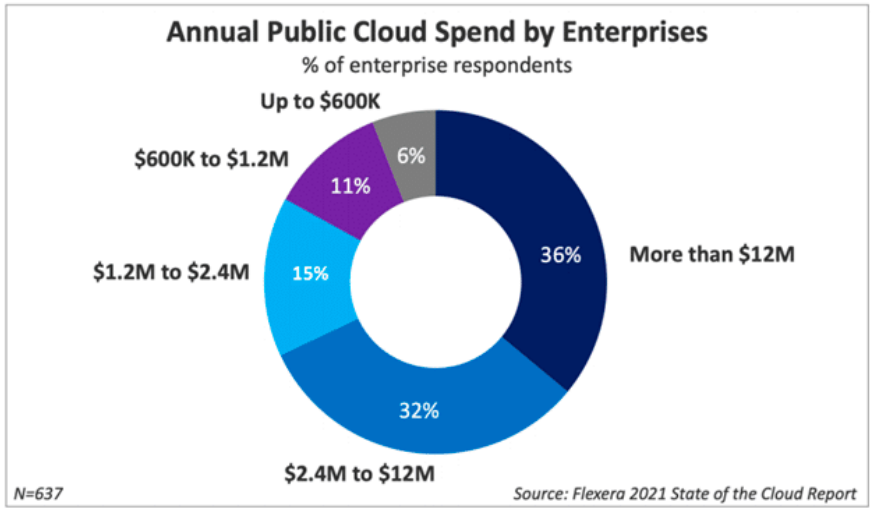
Future of the Joint Enterprise Defense Infrastructure Cloud Contract

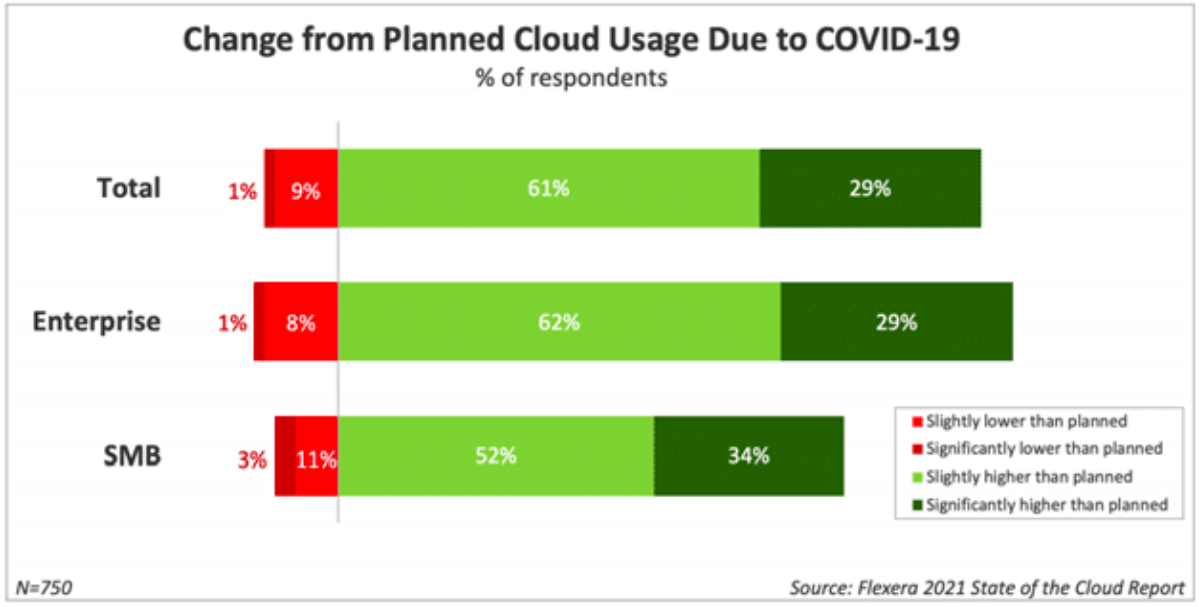
JULY 6, 2021

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

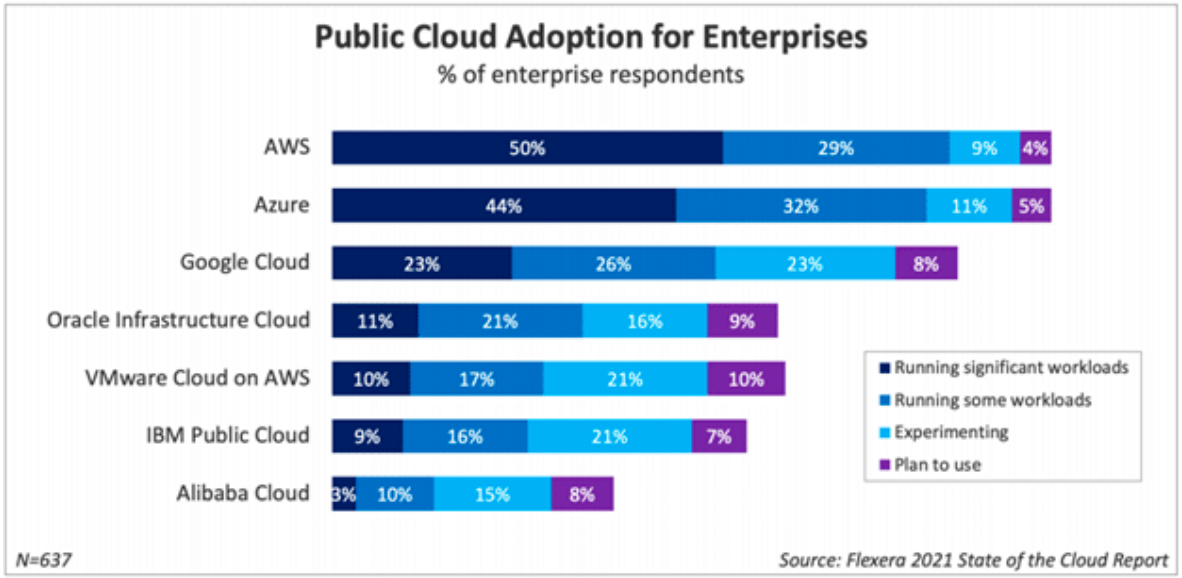


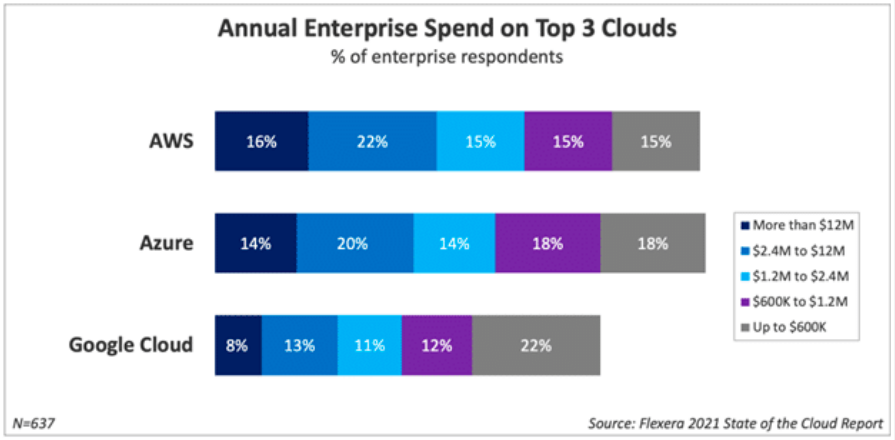


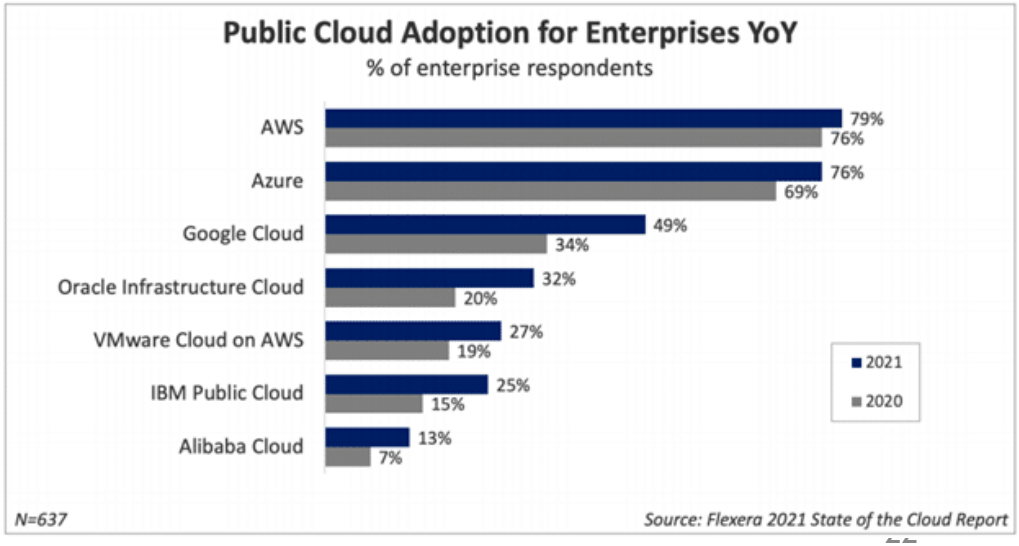


https://www.flexera.com/blog/cloud/cloud-computing-trends-2021-state-of-the-cloud-report/

The State of the Cloud Survey

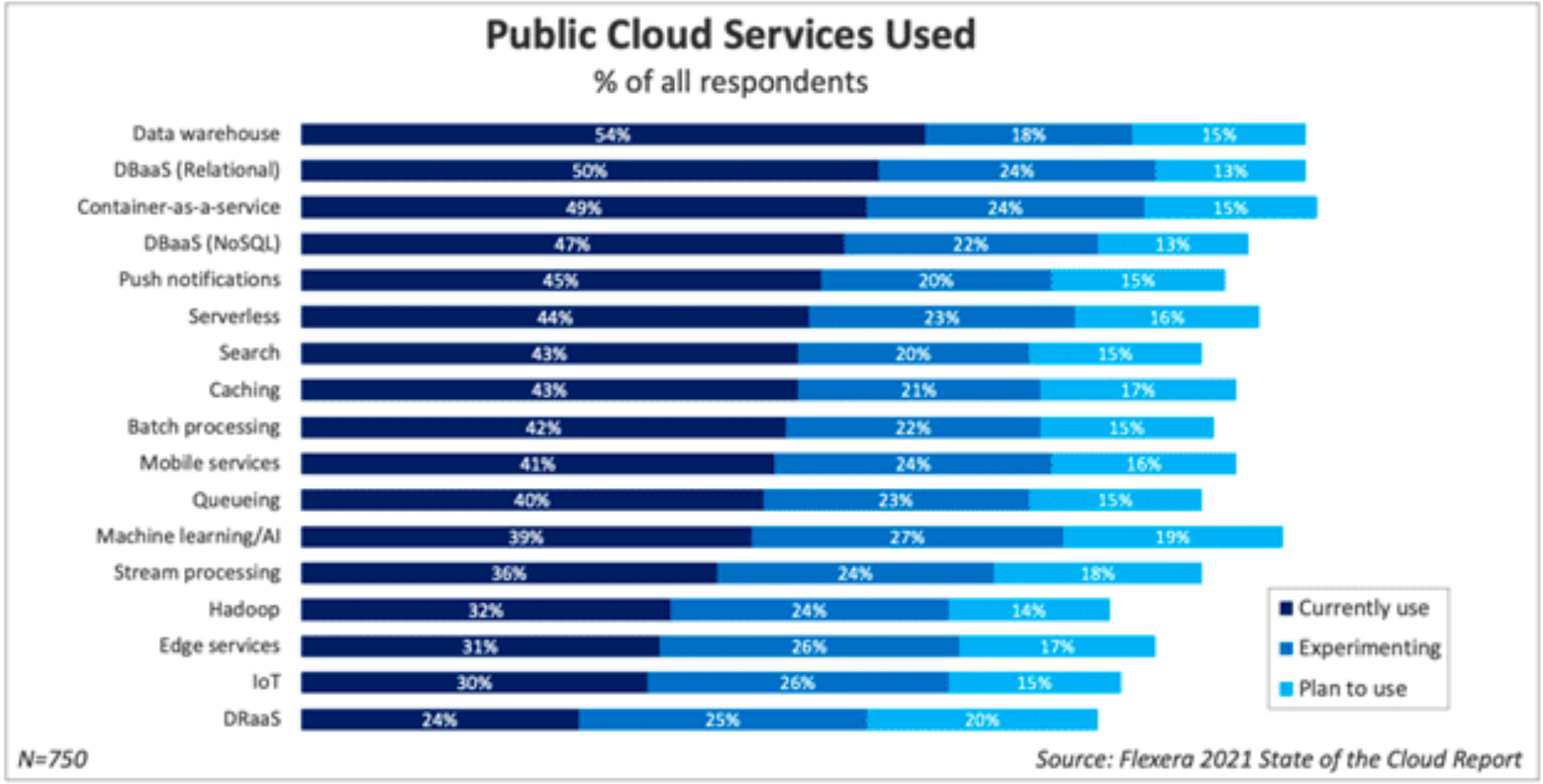






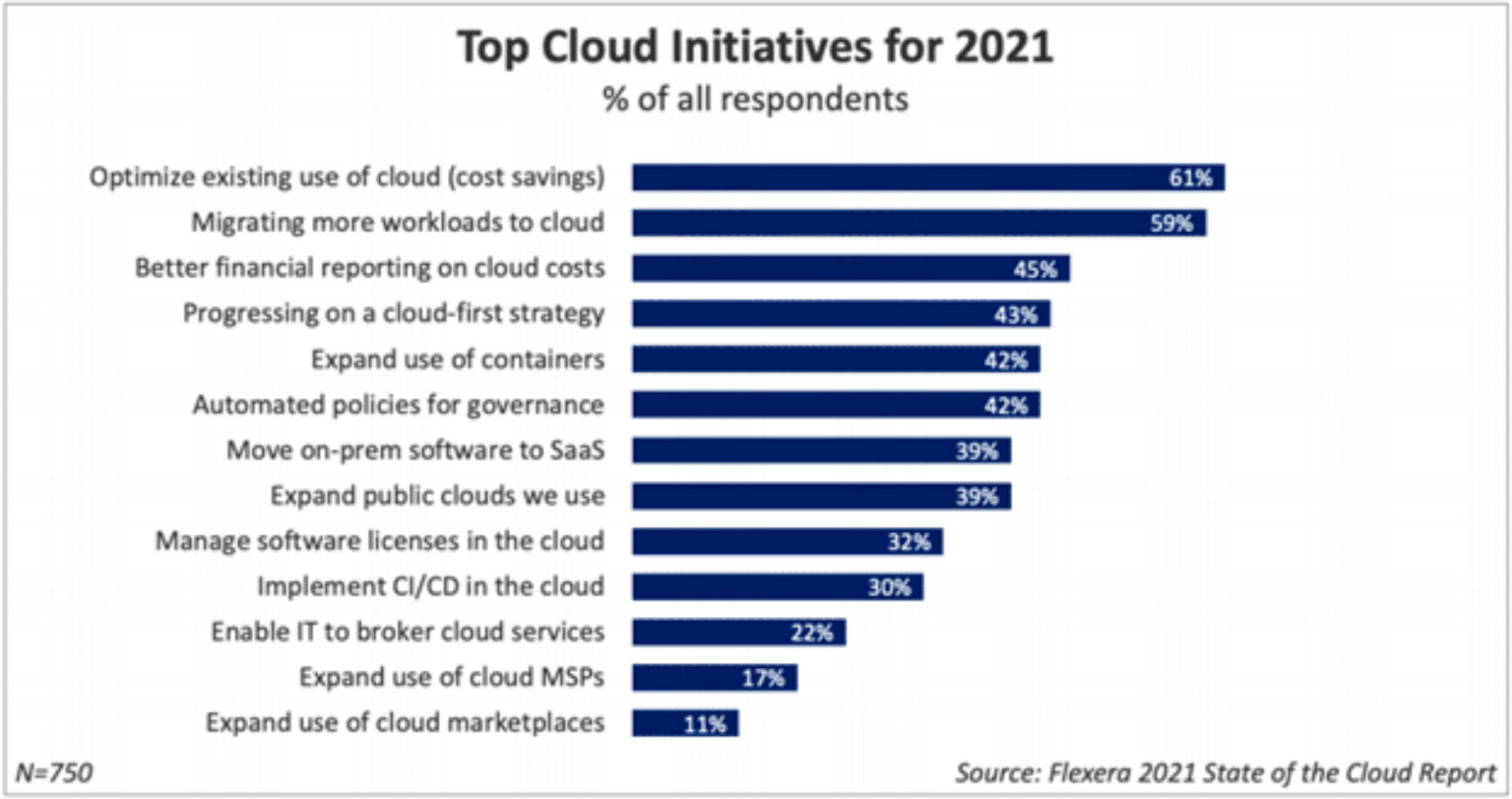
https://www.flexera.com/blog/cloud/cloud-computing-trends-2021-state-of-the-cloud-report/

The State of the Cloud Survey



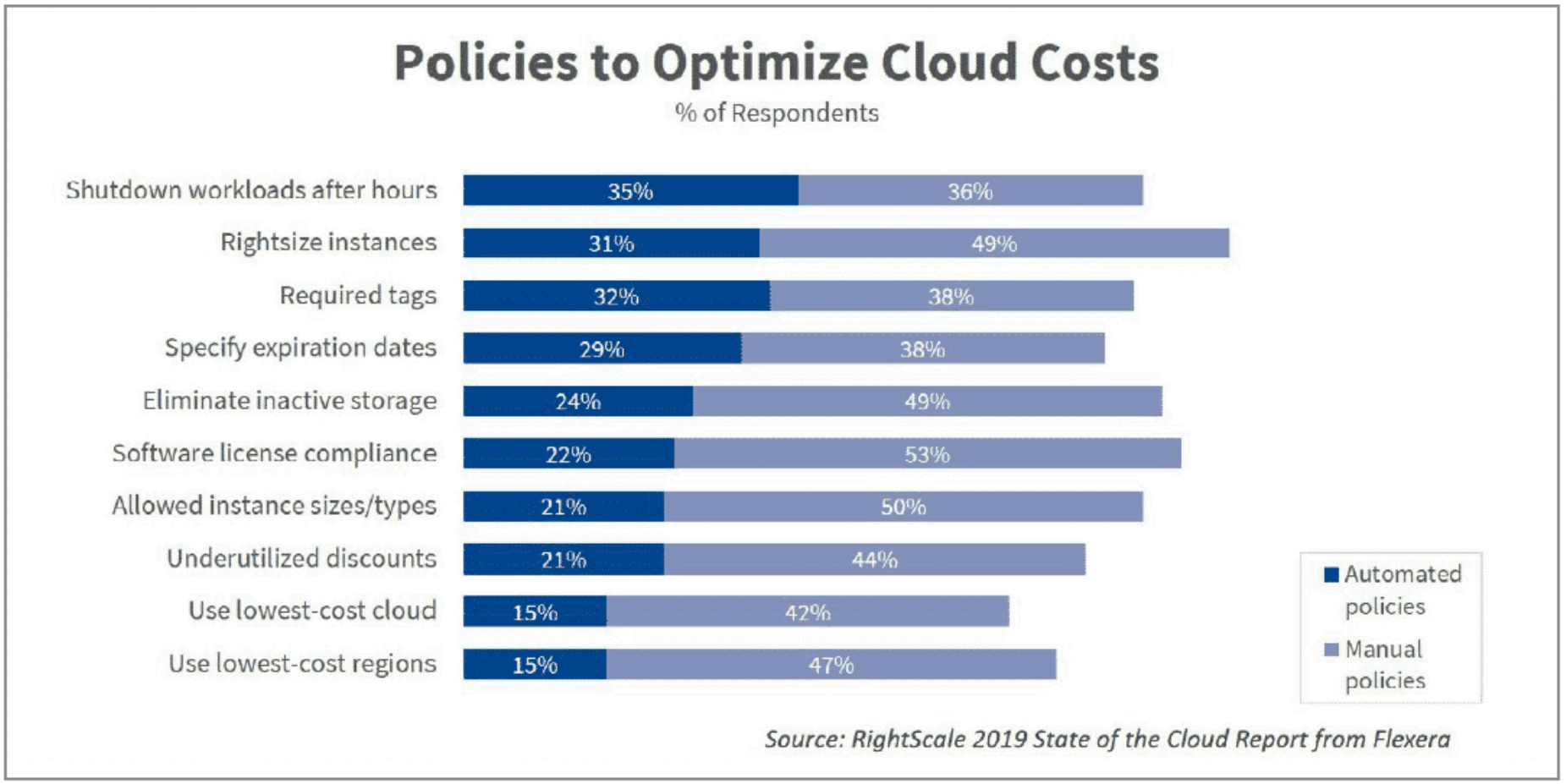
https://www.flexera.com/blog/cloud/cloud-computing-trends-2021-state-of-the-cloud-report/

The State of the Cloud Survey



https://www.flexera.com/blog/cloud/cloud-computing-trends-2021-state-of-the-cloud-report/

The State of the Cloud Survey



https://www.flexera.com/blog/cloud/2019/02/cloud-computing-trends-2019-state-of-the-cloud-survey/

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.

UC San Diego

DSC 102
Systems for Scalable Analytics

Arun Kumar

Topic 3: Parallel and Scalable Data Processing

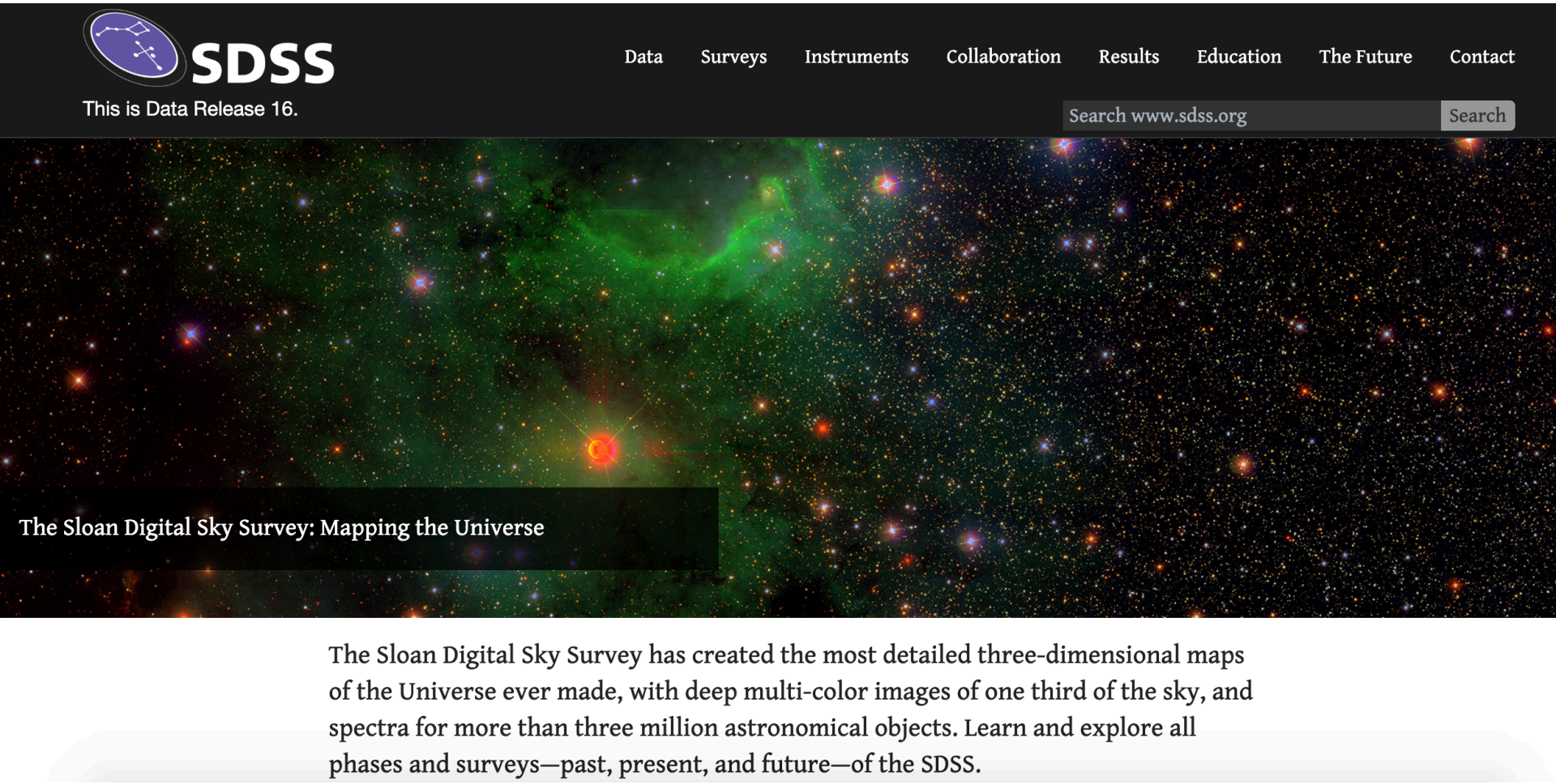
Part 1: Parallelism Basics

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

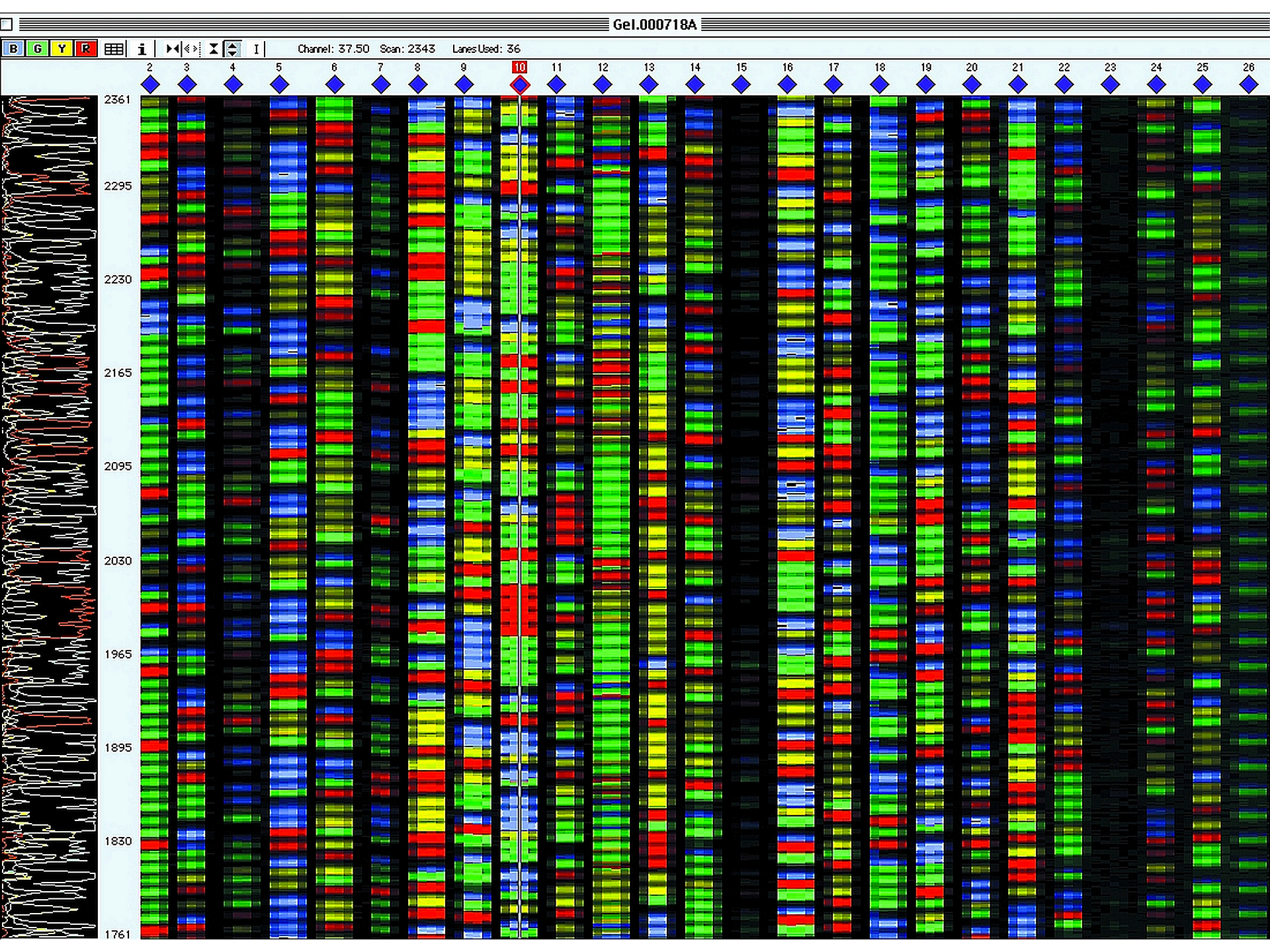
Q: *Why bother with large-scale data? Why does sampling not suffice?*

Large-Scale Data in Astronomy

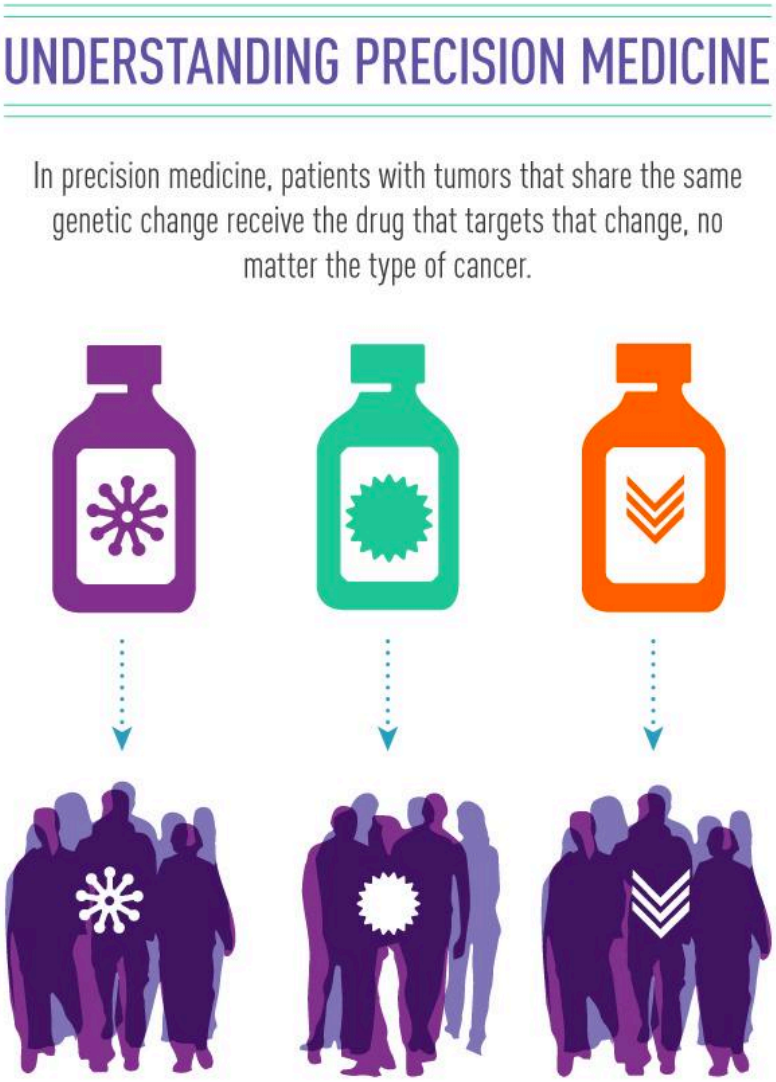


High-res. images: ~200 GB per day since 2000 (1PB+)

Astronomers can study complex galactic evolution behaviors


Gel.000718A
B G Y R
Channel: 37.50 Scan: 2343 Lanes Used: 36
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
2361
2295
2230
2165
2095
2030
1965
1895
1830
1761

Large-Scale Data in Genomics



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

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

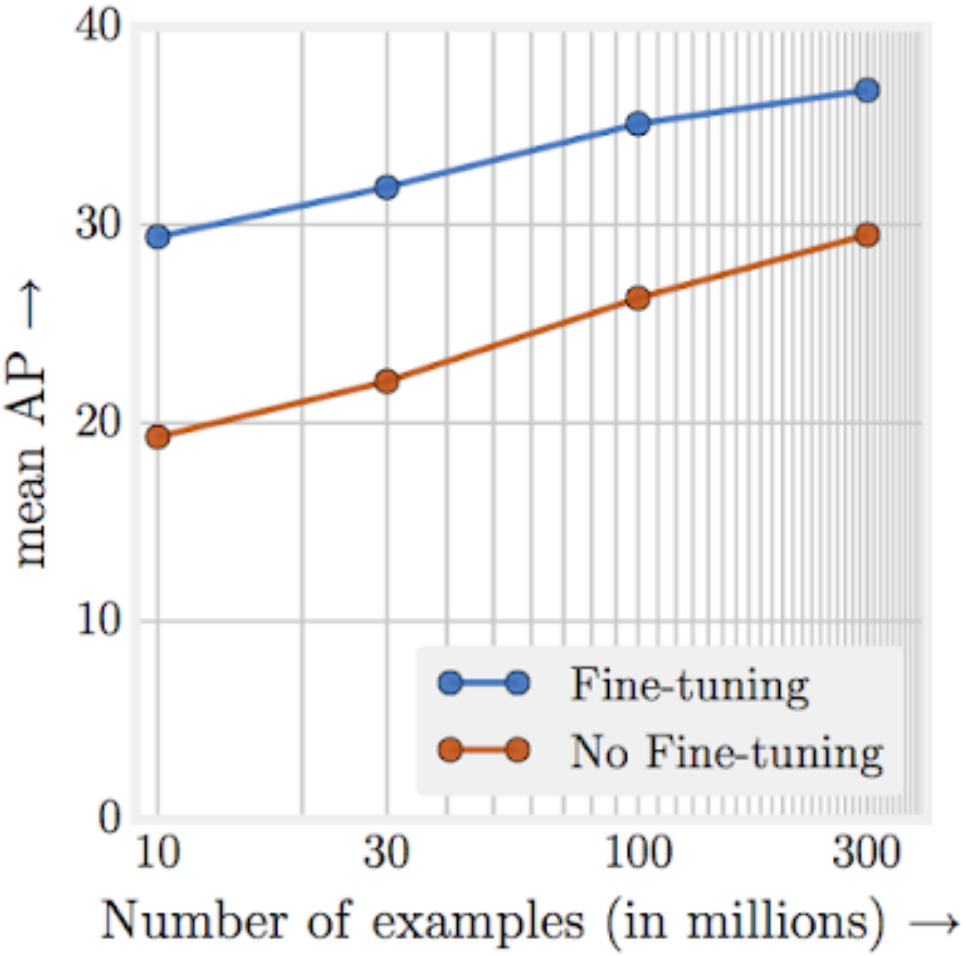
Large-Scale Data in Computer Vision

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

>500GB uncompressed as tensors

Harbinger of deep learning revolution

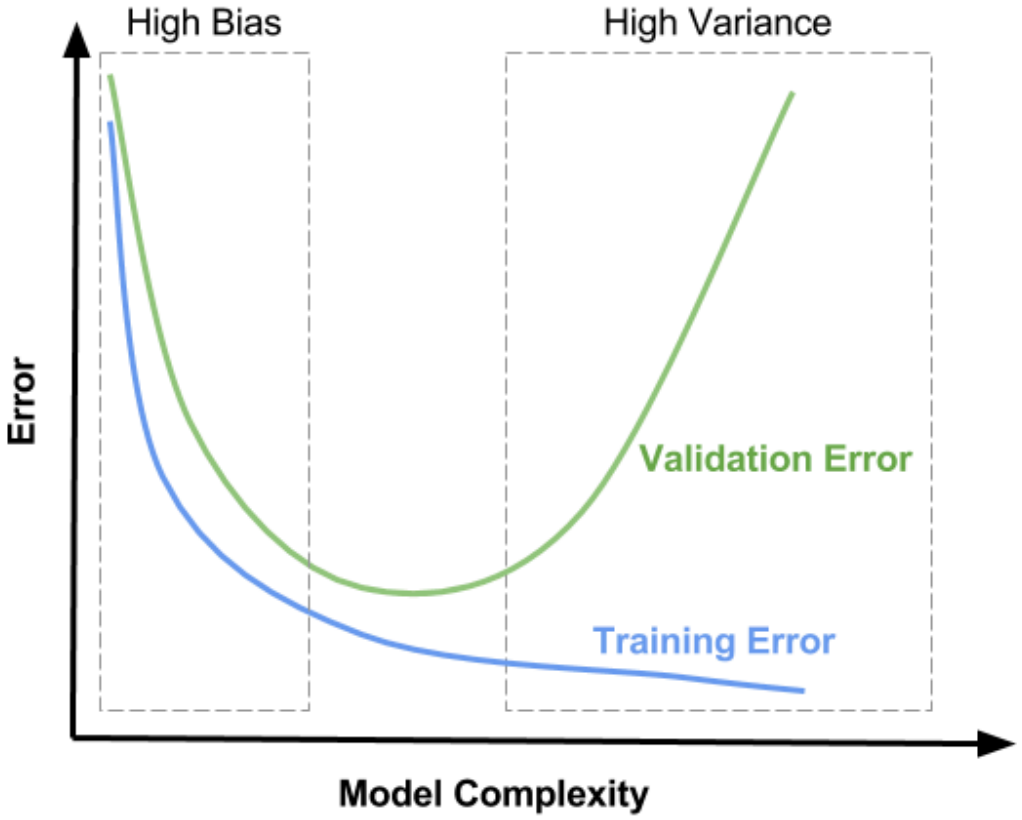
“The Unreasonable Effectiveness of Data”



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

https://ai.googleblog.com/2017/07/revisiting-unreasonable-effectiveness.html

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

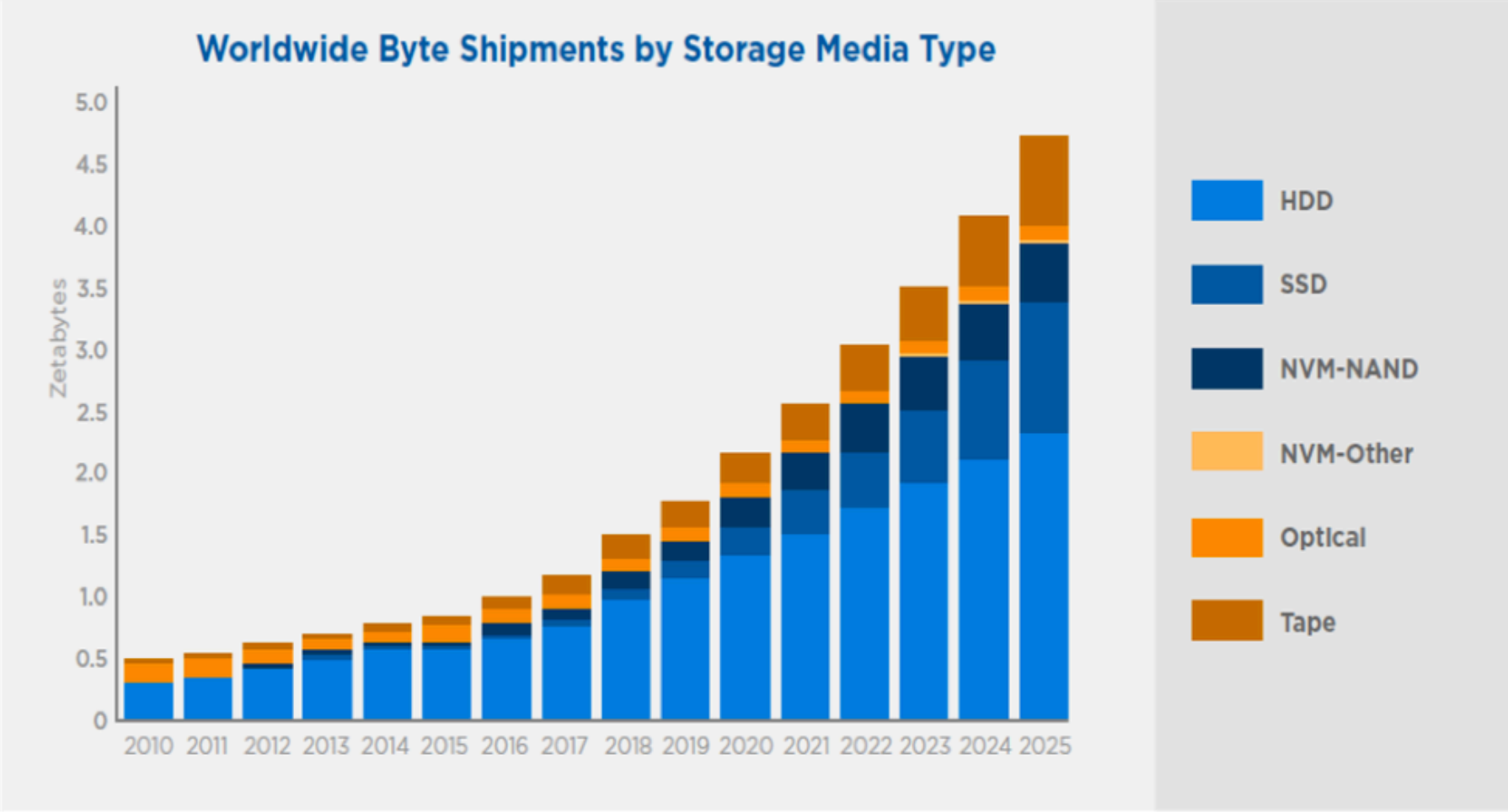
“Big Data”

- 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!

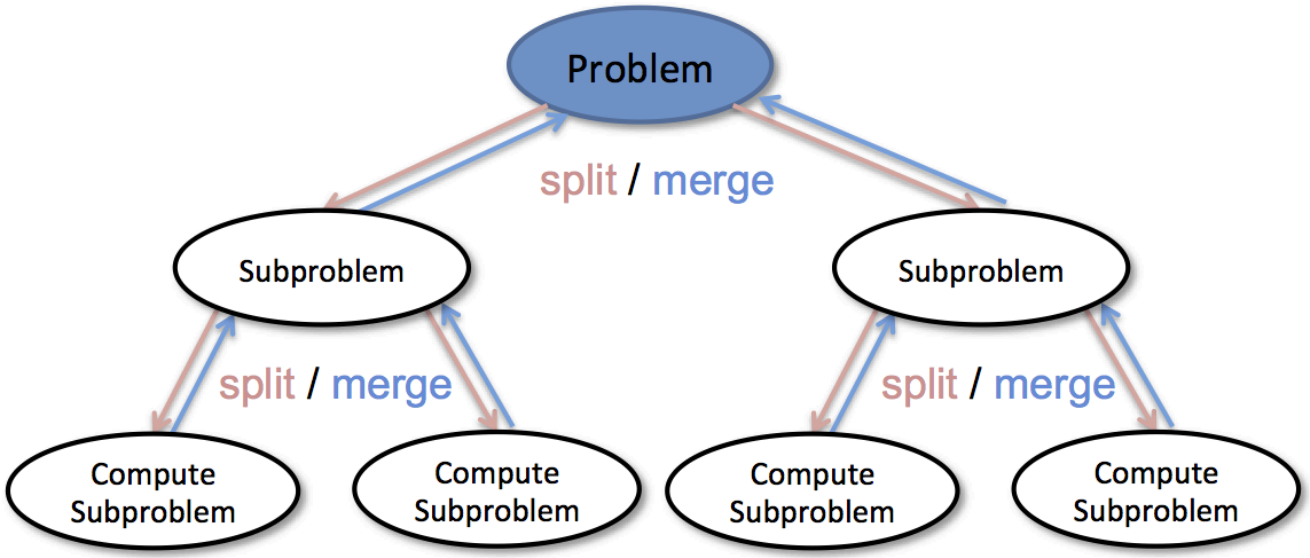
Outline

- 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”)

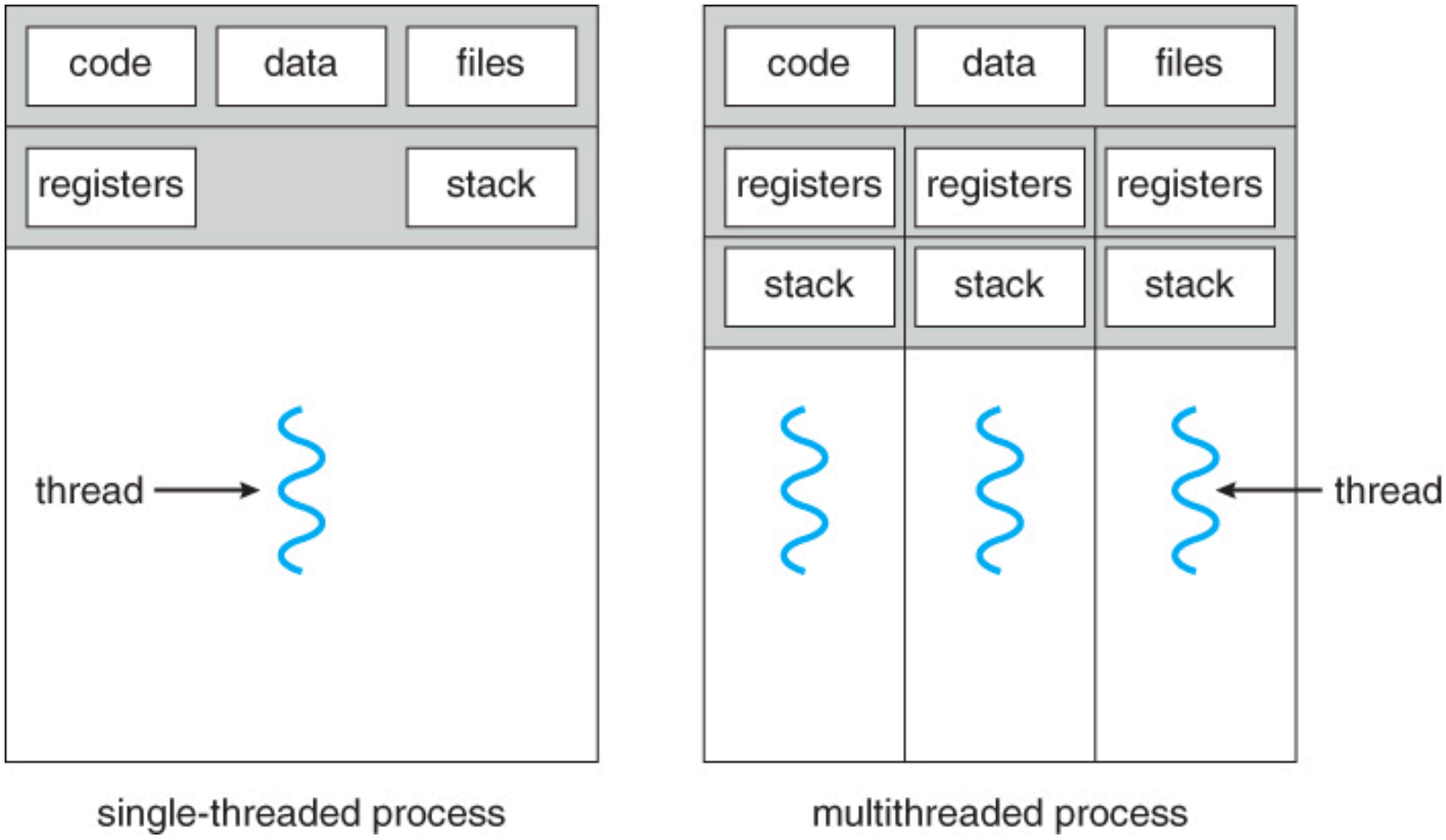


https://medium.com/cracking-the-data-science-interview/divide-and-conquer-algorithms-b135681d08fc

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

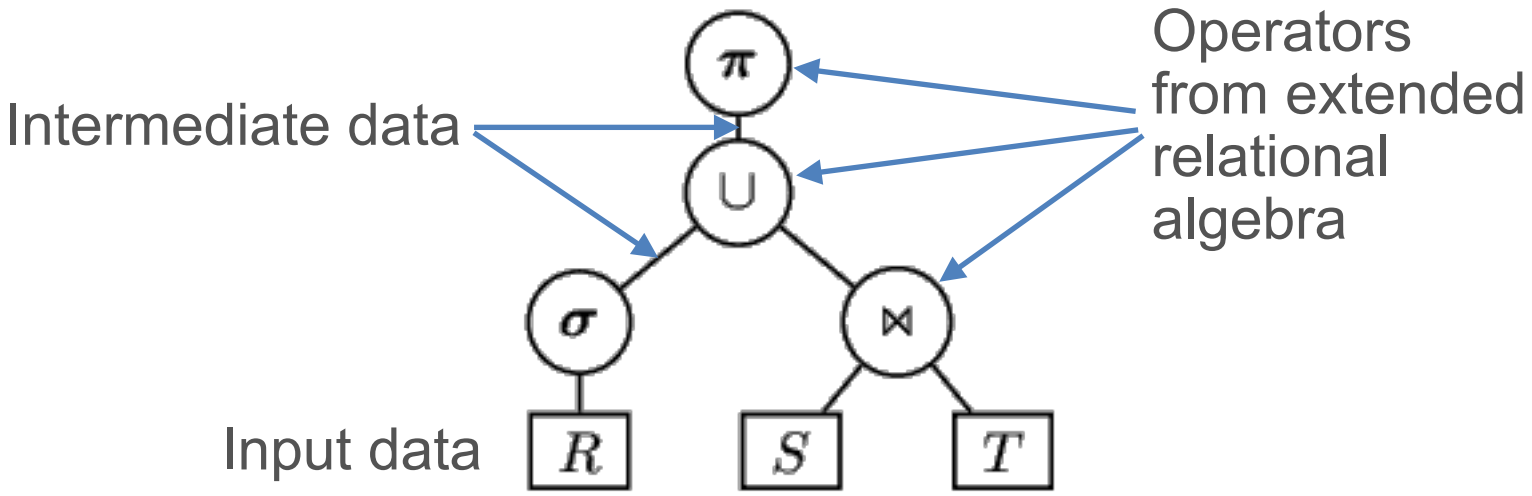


New Parallelism Concept: Dataflow

- 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

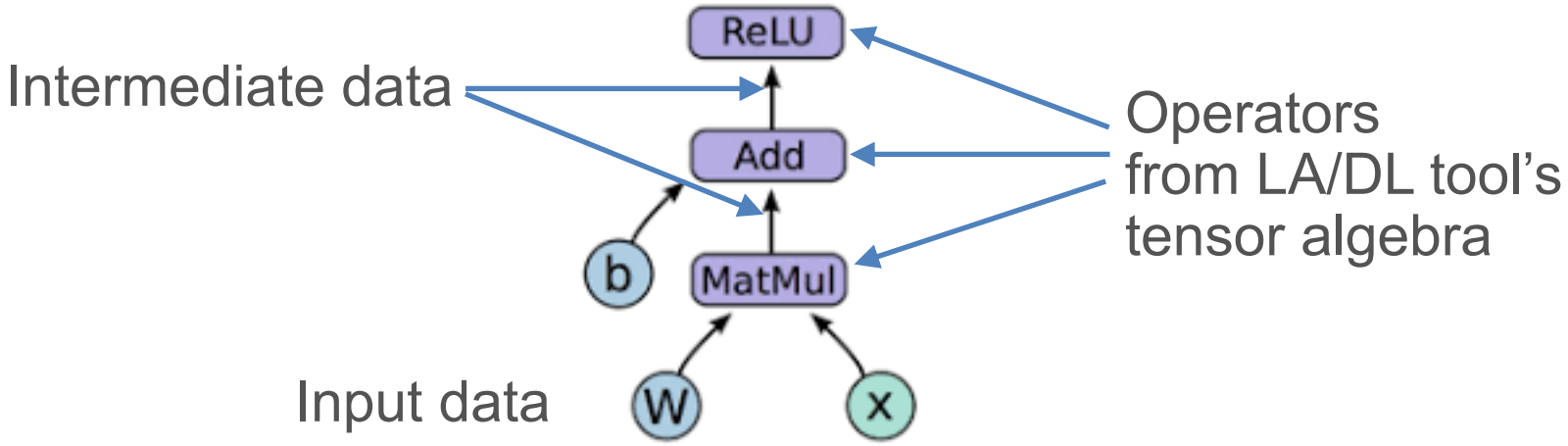
$$\pi(\sigma(R) \cup S \bowtie T)$$



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

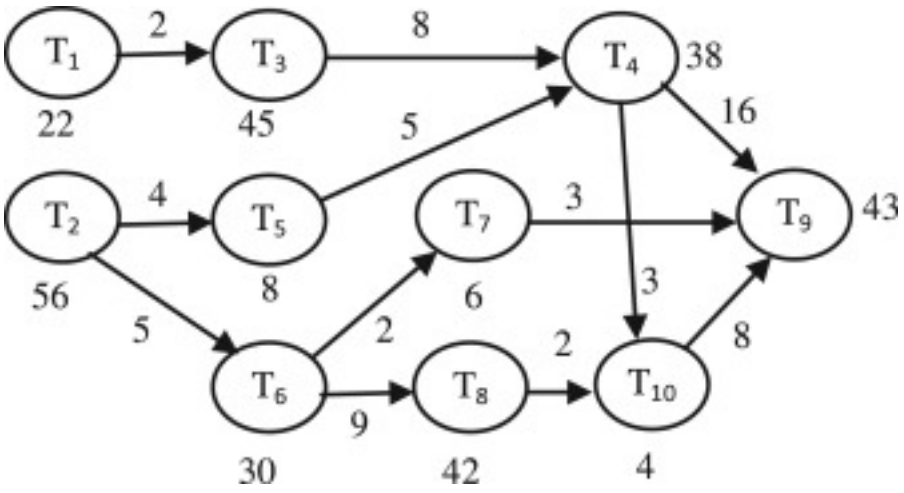
Example Tensor Dataflow Graph

$$ReLU(WX + b)$$



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

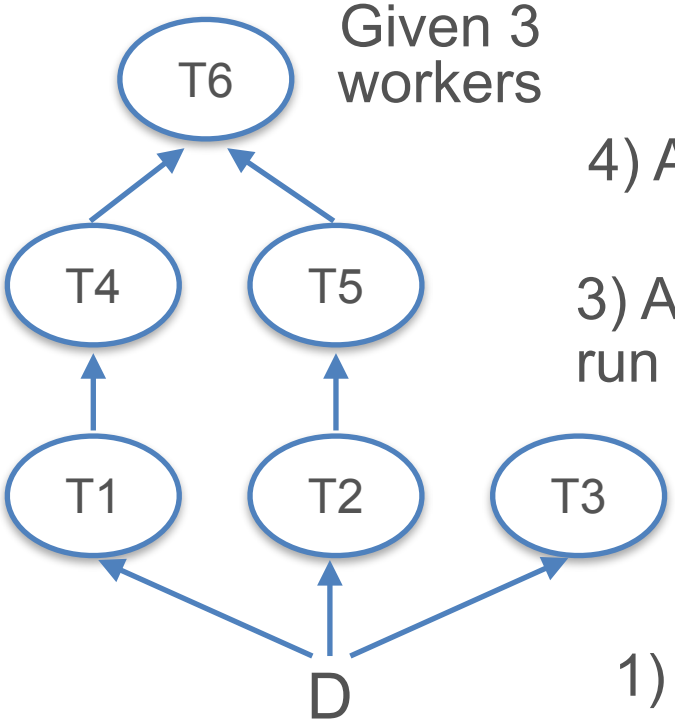
Outline

- 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, Dask Scheduler puts tasks on workers for you.

4) After T4 & T5 end, run T6 on W1; W2 is *idle*

3) After T1 ends, run T4 on W1; after T2 ends, run T5 on W2; after T3 ends, W3 is *idle*

2) Put T1 on worker 1 (W1), T2 on W2, T3 on W3; run all 3 in parallel

1) Copy whole D to all workers

Task Parallelism

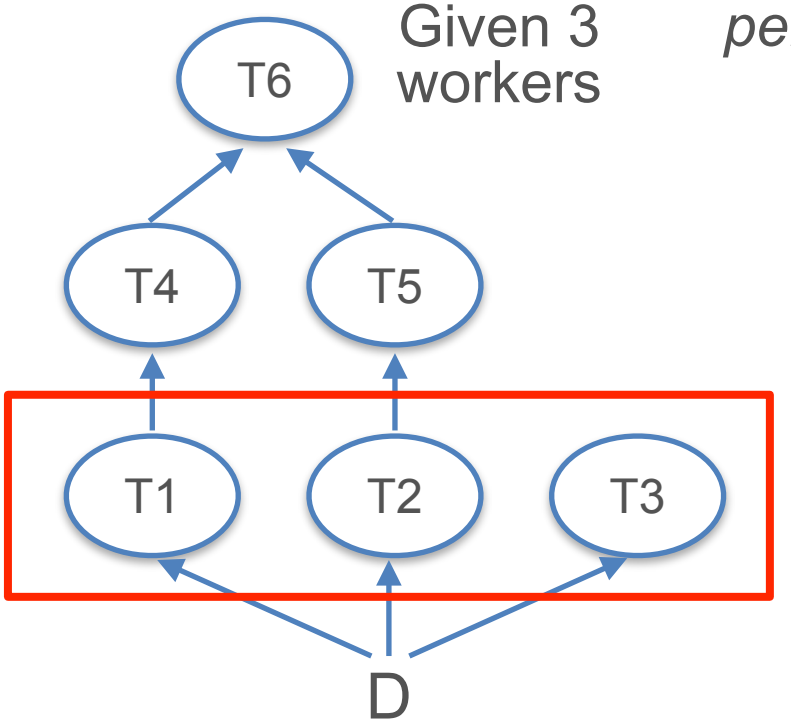
- **Topological sort** of tasks in task graph for scheduling
- Notion of a "worker" can be at processor/core level, not just at node/server level
  - *Thread-level* parallelism possible instead of process-level
  - E.g., Dask: 4 worker nodes x 4 cores = 16 workers total
- **Main pros** of task parallelism:
  - **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 task can be run simultaneously

Example:

Given 3 workers



***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

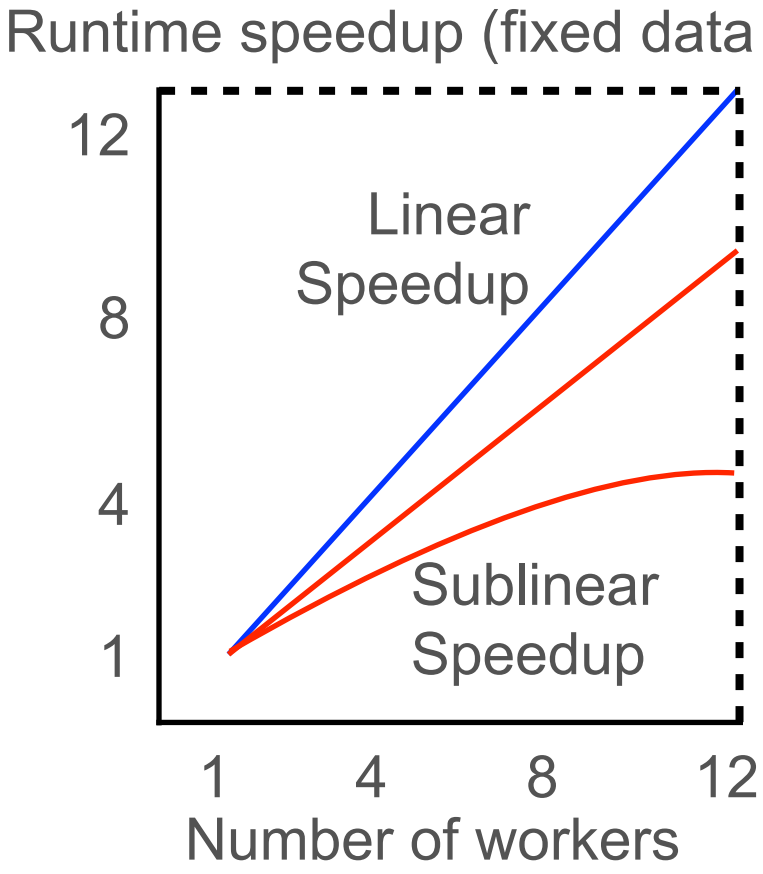
$$\textbf{Speedup} = \frac{\text{Completion time given only 1 worker}}{\text{Completion time given n (>1) workers}}$$

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

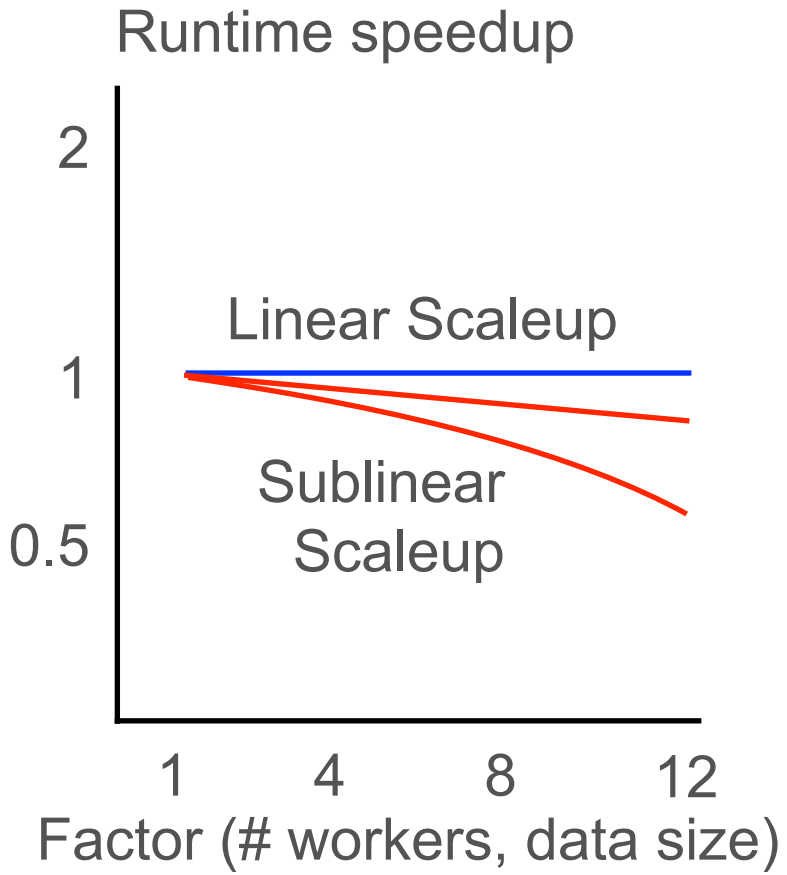
It depends!

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

Quantifying Benefit of Parallelism



Speedup plot / Strong scaling



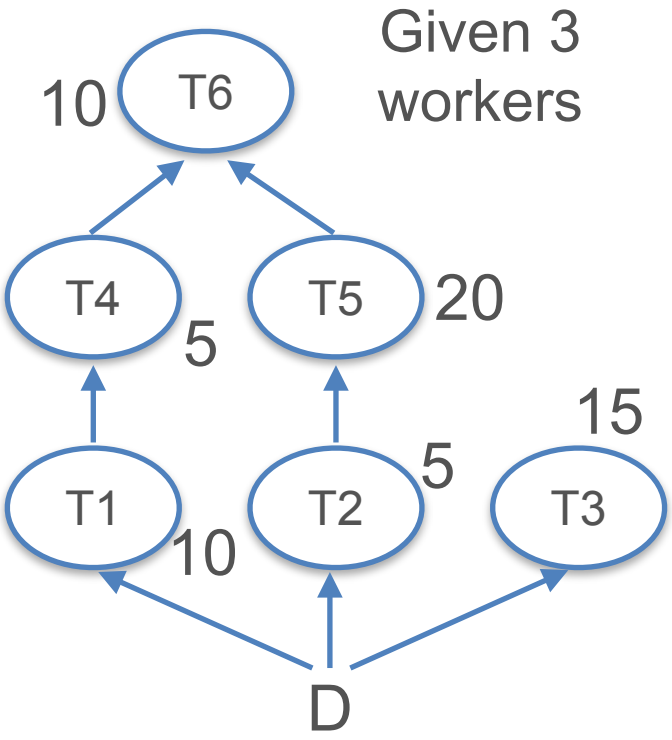
Scaleup plot / Weak scaling

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

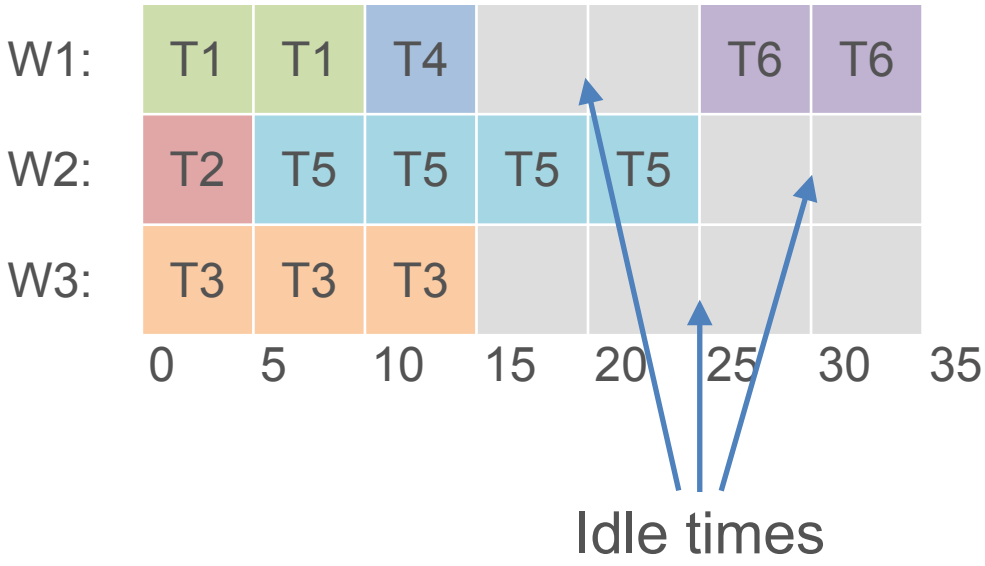
Idle Times in Task Parallelism

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

Example:



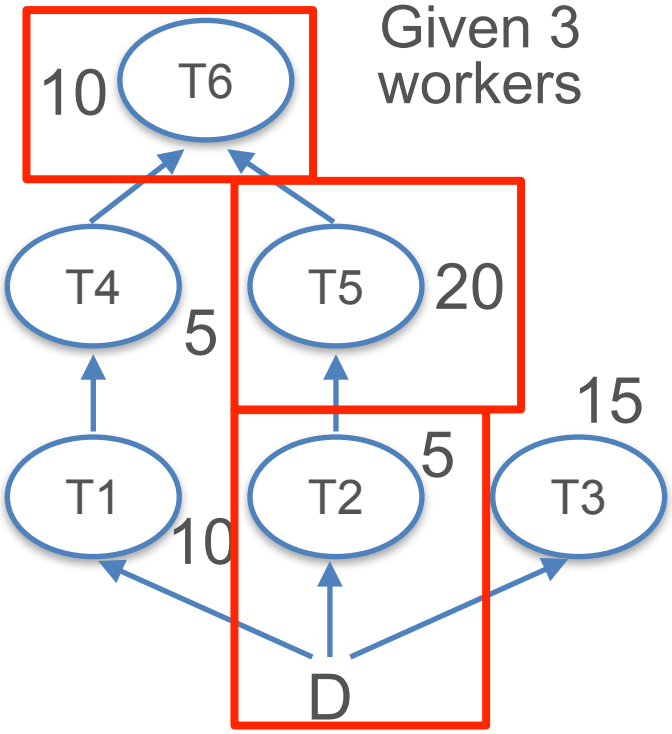
Gantt Chart visualization of schedule:



Idle Times in Task Parallelism

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

Example:

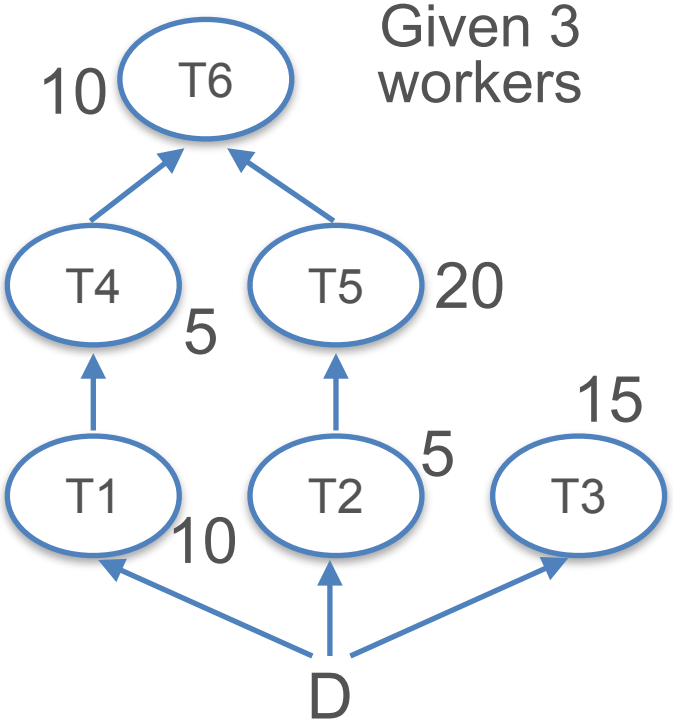


- In general, overall workload's completion time on task-parallel setup is always *lower bounded* by the **longest path** in the task graph
- Possibility: A task-parallel scheduler can "release" a worker if it knows that will be idle till the end
  - Can saves costs in cloud

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

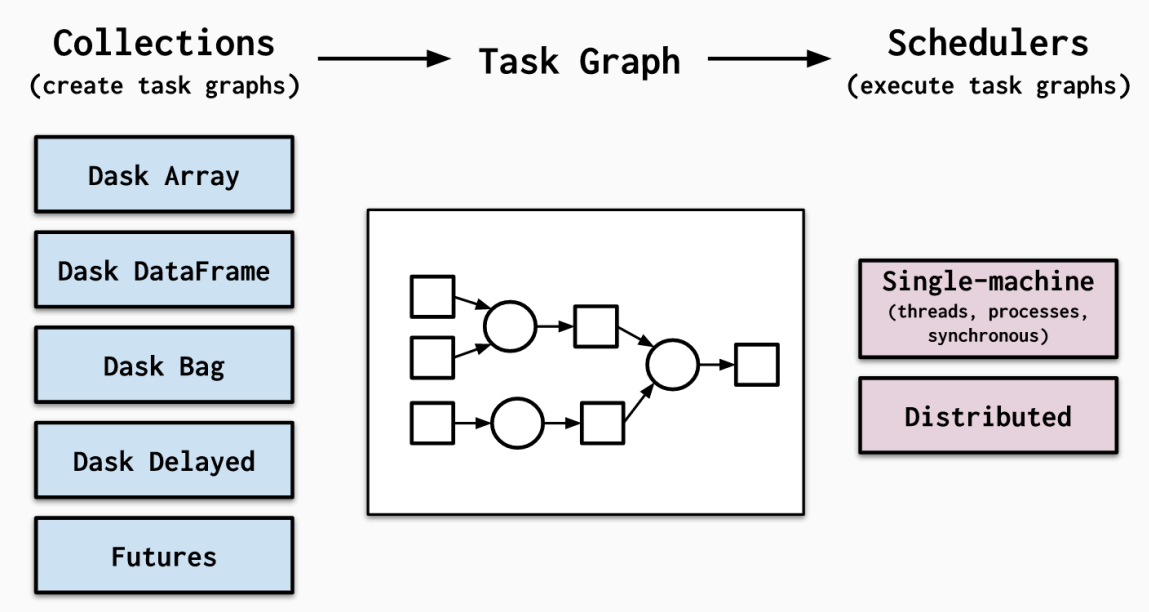
***Q:** Why is it only 1.9x?*

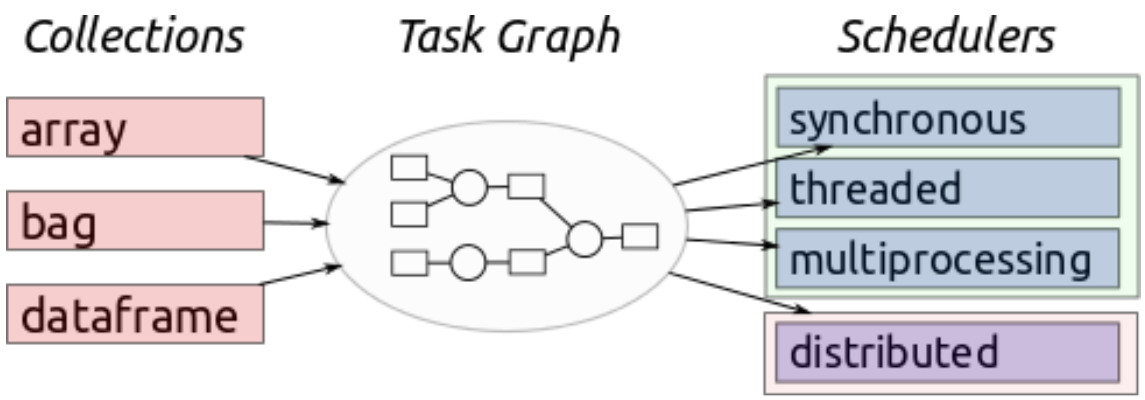
Task Parallelism in Dask

- *"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

https://docs.dask.org/en/latest/

Task Parallelism in Dask





https://docs.dask.org/en/latest/
https://docs.dask.org/en/latest/scheduling.html

Dask's Workflow

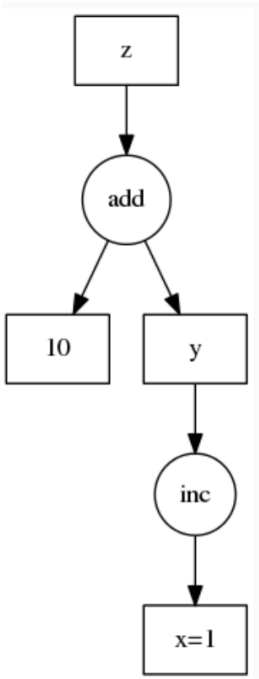
- **"Lazy Evaluation":**
  - Ops on data struct. are NOT executed immediately
  - Triggered manually, e.g., *compute()*
  - Dataflow graph / task graph is built under the hood

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

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

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

User code using their API

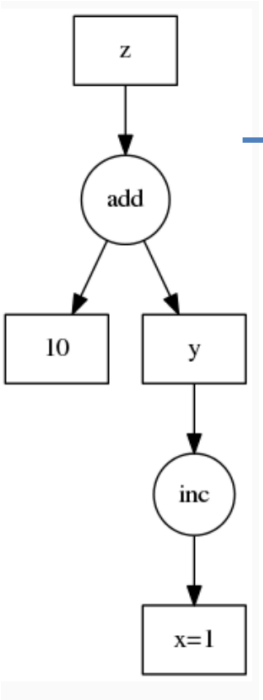
```
d = {'x': 1,
     'y': (inc, 'x'),
     'z': (add, 'y', 10)}
```

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



Dask's Workflow

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



"Optimize" Task Graph

Serialize Task Graph (dictionary)

Communicate it to Scheduler

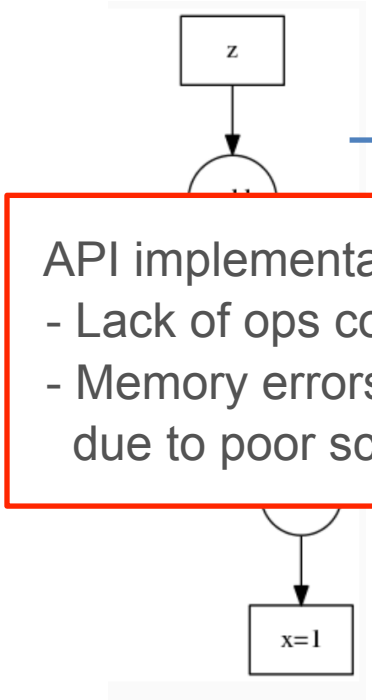
Scheduler divvies up work for cores/nodes

Each worker does its job and notifies Scheduler

https://docs.dask.org/en/latest/phases-of-computation.html

Possible Bottlenecks/Issues in Dask

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



API implementation:
- Lack of ops coverage
- Memory errors in ops due to poor scalability

"Optimize" Task Graph

Poor/missed optimizations

Serialize Task Graph

Huge task graphs take too long to serialize and/or communicate

Scheduler divvies up work for cores/nodes

Poor scalability of Scheduler

Each worker does its job and notifies Scheduler

Machine/network failures

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 ~ 3x-10x # cores, but
    - # cores x chunk size must be < machine DRAM, but
    - chunk size shouldn't be too small (~1 GB is OK)

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

Dask still lacks "physical data independence"!

https://docs.dask.org/en/latest/best-practices.html

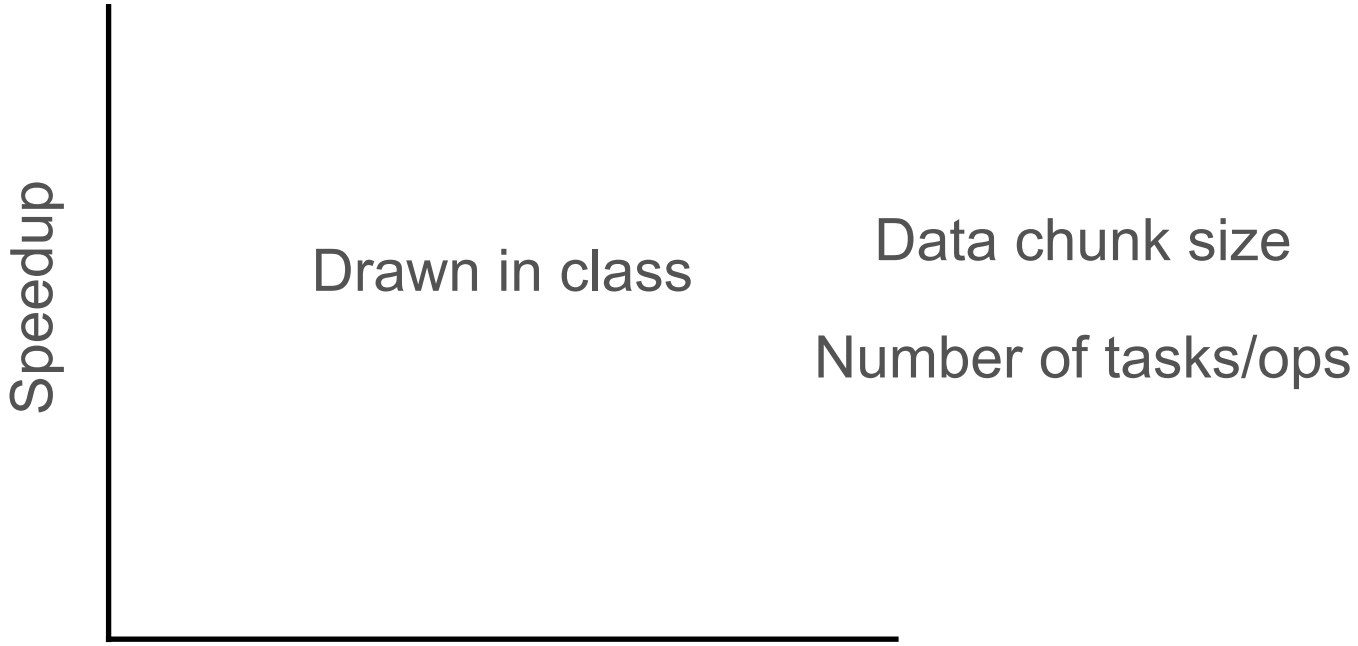
Best Practices for Task-Par. Dask

- **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”

https://docs.dask.org/en/latest/best-practices.html
https://docs.dask.org/en/latest/diagnostics-distributed.html

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

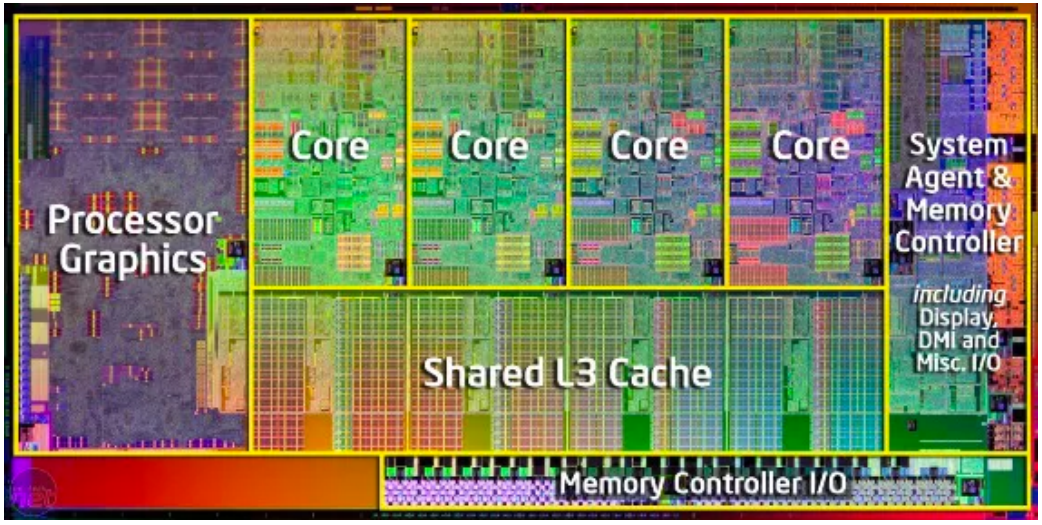
10:46 AM · Feb 2, 2021 · Twitter Web App

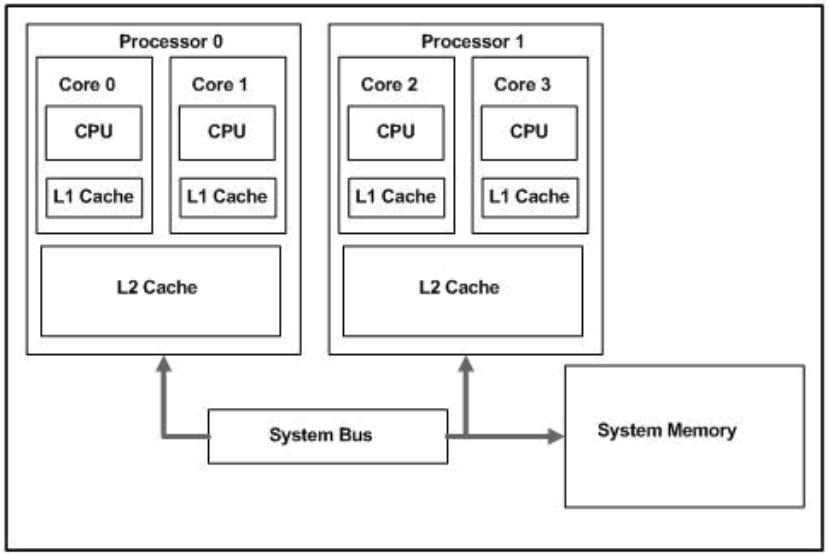
Outline

- 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

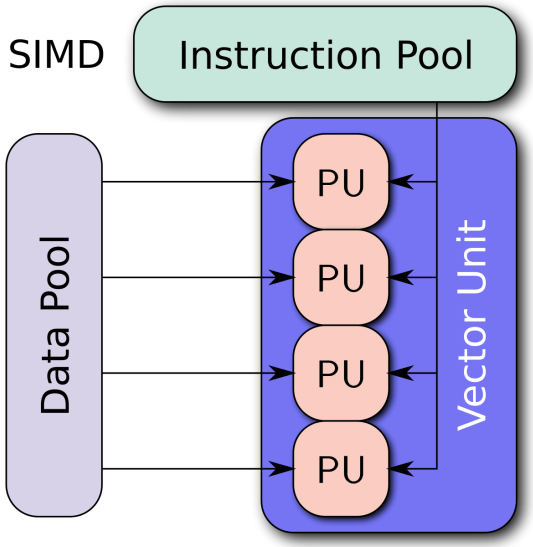
- Modern machines often have multiple processors and multiple *cores* per processor; hierarchy of shared caches
- OS Scheduler now controls what cores/processors assigned to what processes/threads when



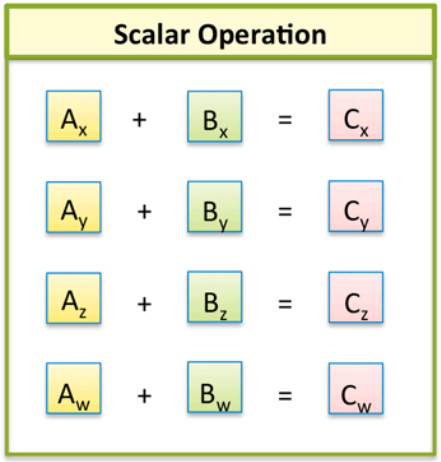


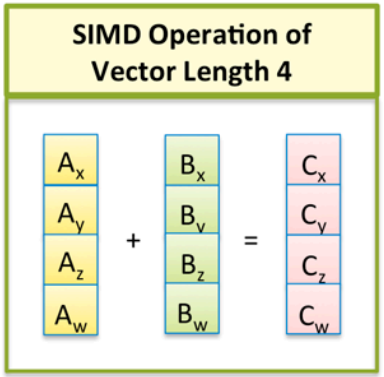
Single-Instruction Multiple-Data

- **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:



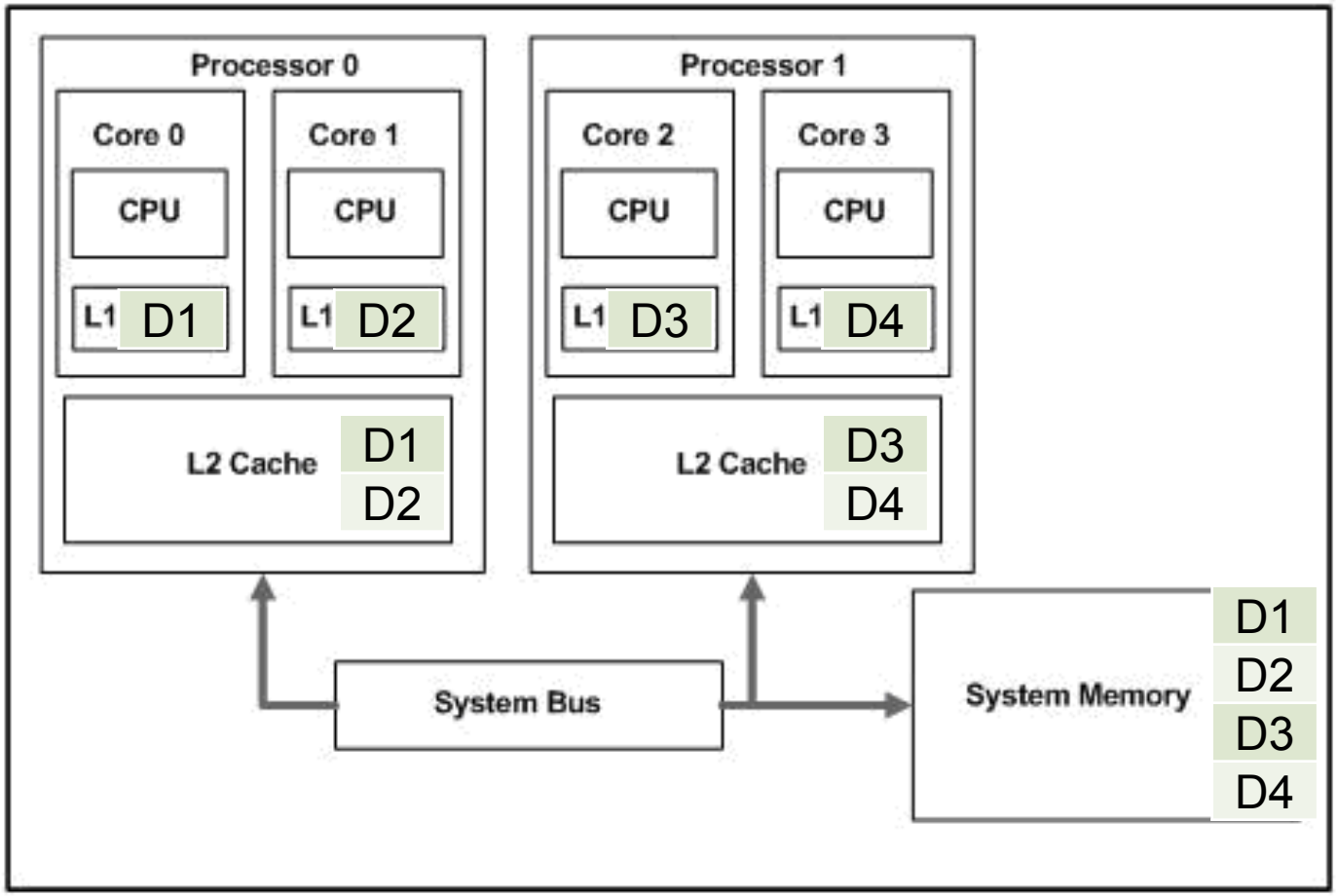


Intel® Architecture currently has SIMD operations of vector length 4, 8, 16

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:

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

- 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:

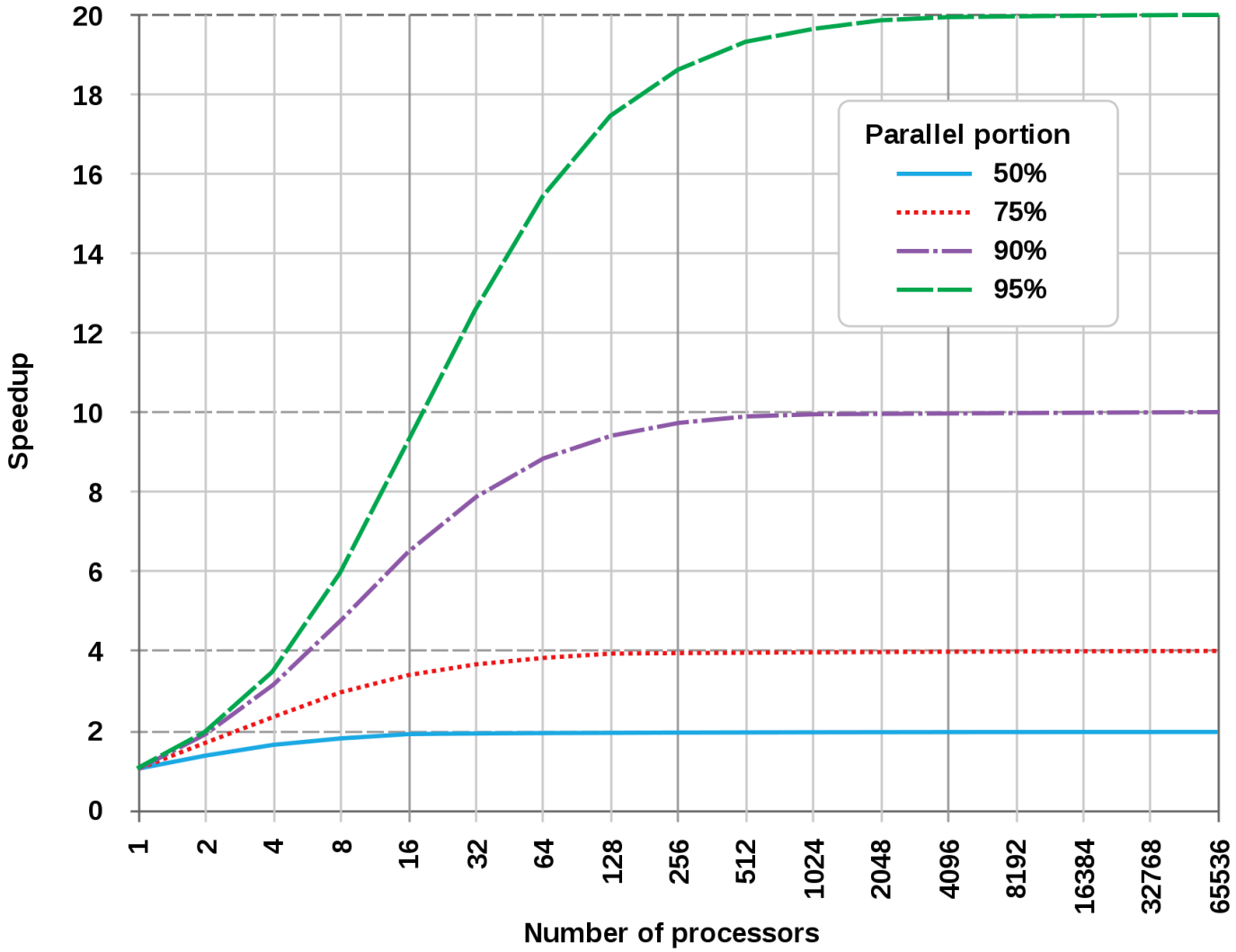
$T_{yes} \longrightarrow T_{yes}/n$

$T_{no} \longrightarrow T_{no}$

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

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

Amdahl's Law



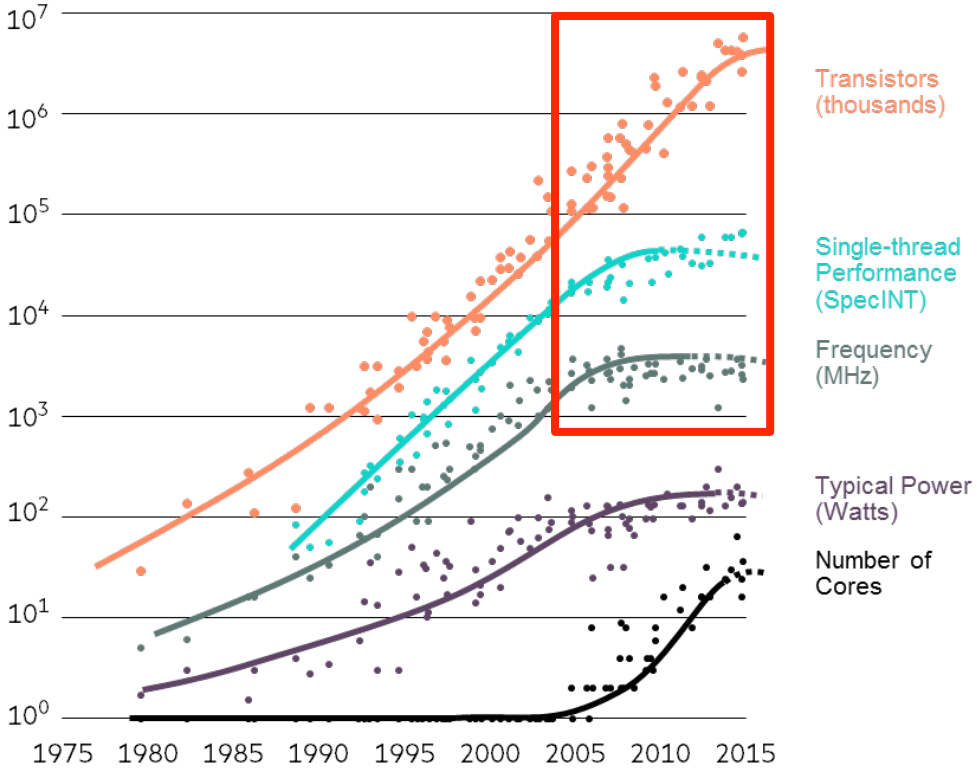
$$\text{Speedup} = \frac{n(1+f)}{n+f}$$

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

$$\text{Parallel portion} = f / (1+f)$$

Hardware Trends on Parallelism

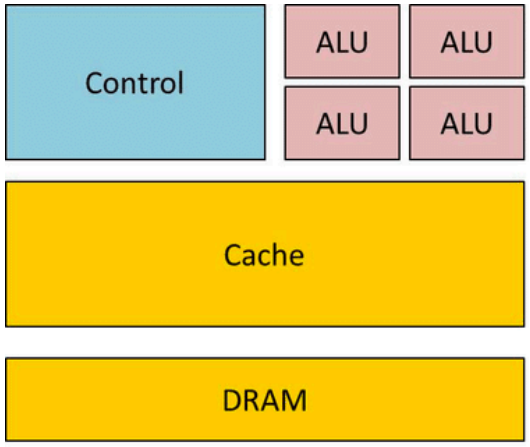
- Multi-core processors grew rapidly in early 2000s but hit physical limits due to packing efficiency and power issues
- End of “Moore’s Law” and End of “Dennard Scaling”



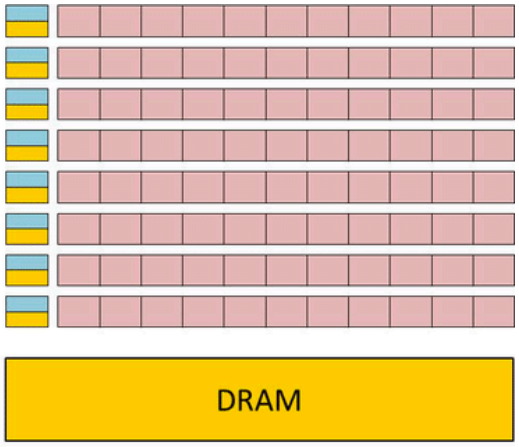
- Takeaway from hardware trends: it is hard for general-purpose CPUs to sustain FLOP-heavy programs like deep nets
- Motivated the rise of “accelerators” for some classes of programs

Hardware Accelerators: GPUs

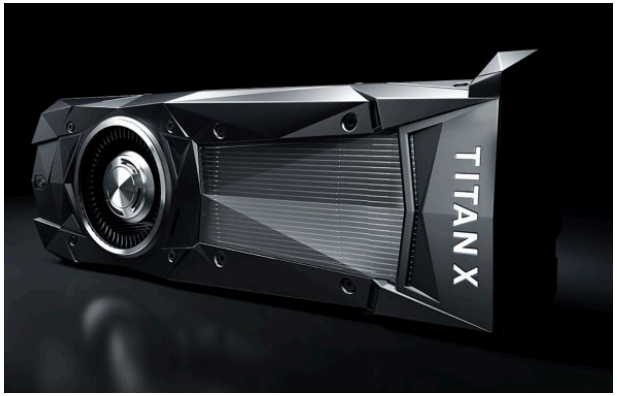
- **Graphics Processing Unit (GPU)**: Tailored for matrix/tensor ops
- Basic idea: use tons of ALUs; massive data parallelism (SIMD on steroids); Titan X offers ~11 TFLOP/s!
- Popularized by NVIDIA in early 2000s for video games, graphics, and video/multimedia; now ubiquitous in deep learning
- CUDA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



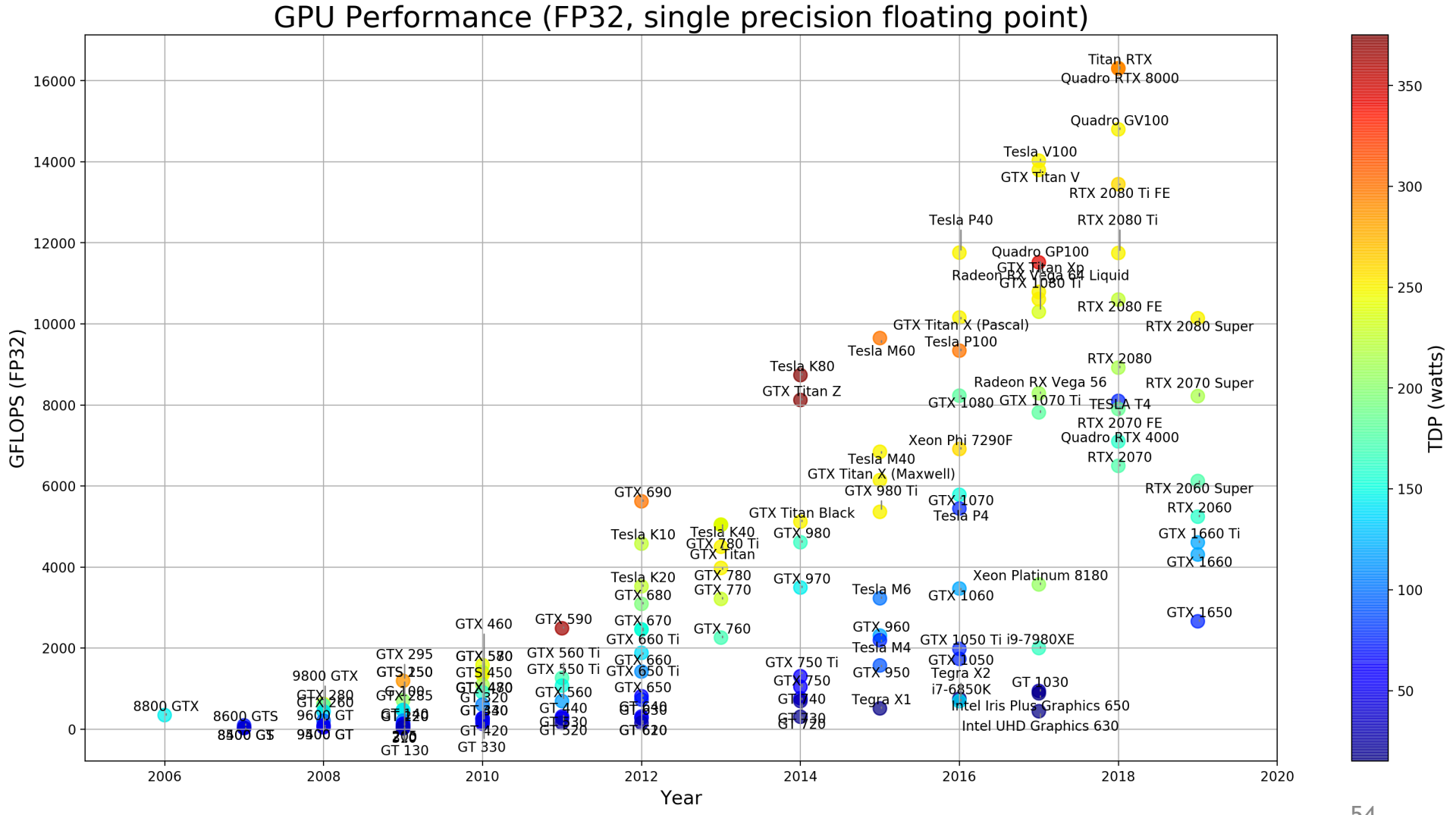
CPU



GPU

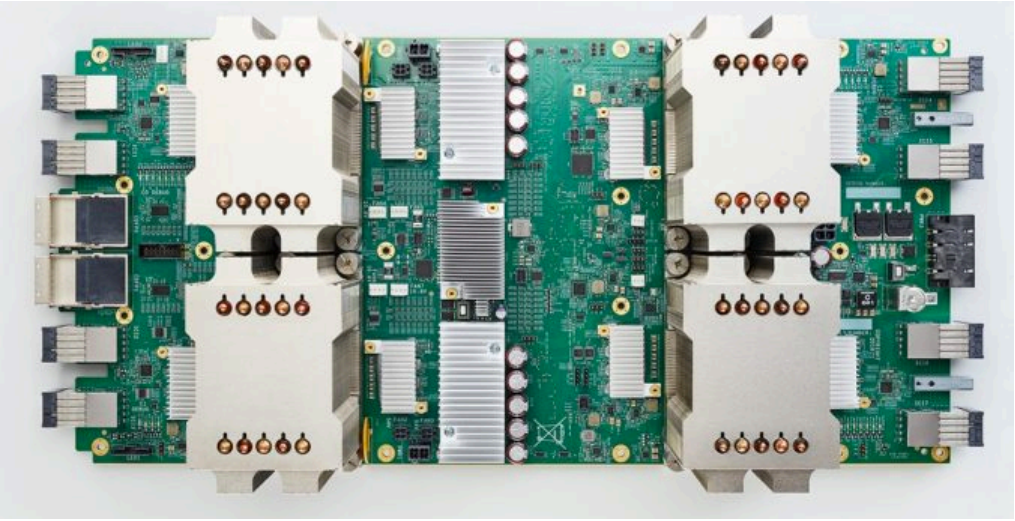


GPUs on the Market



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

https://www.embedded.com/leveraging-fpgas-for-deep-learning/

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