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.

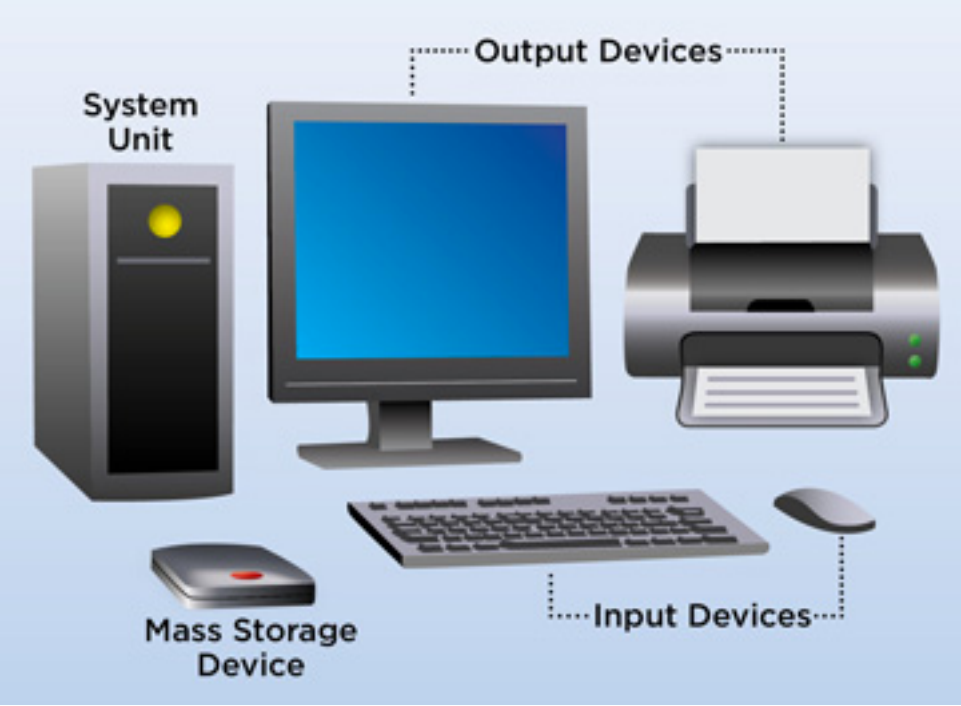
Computer science aka “Datalogy”

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.)

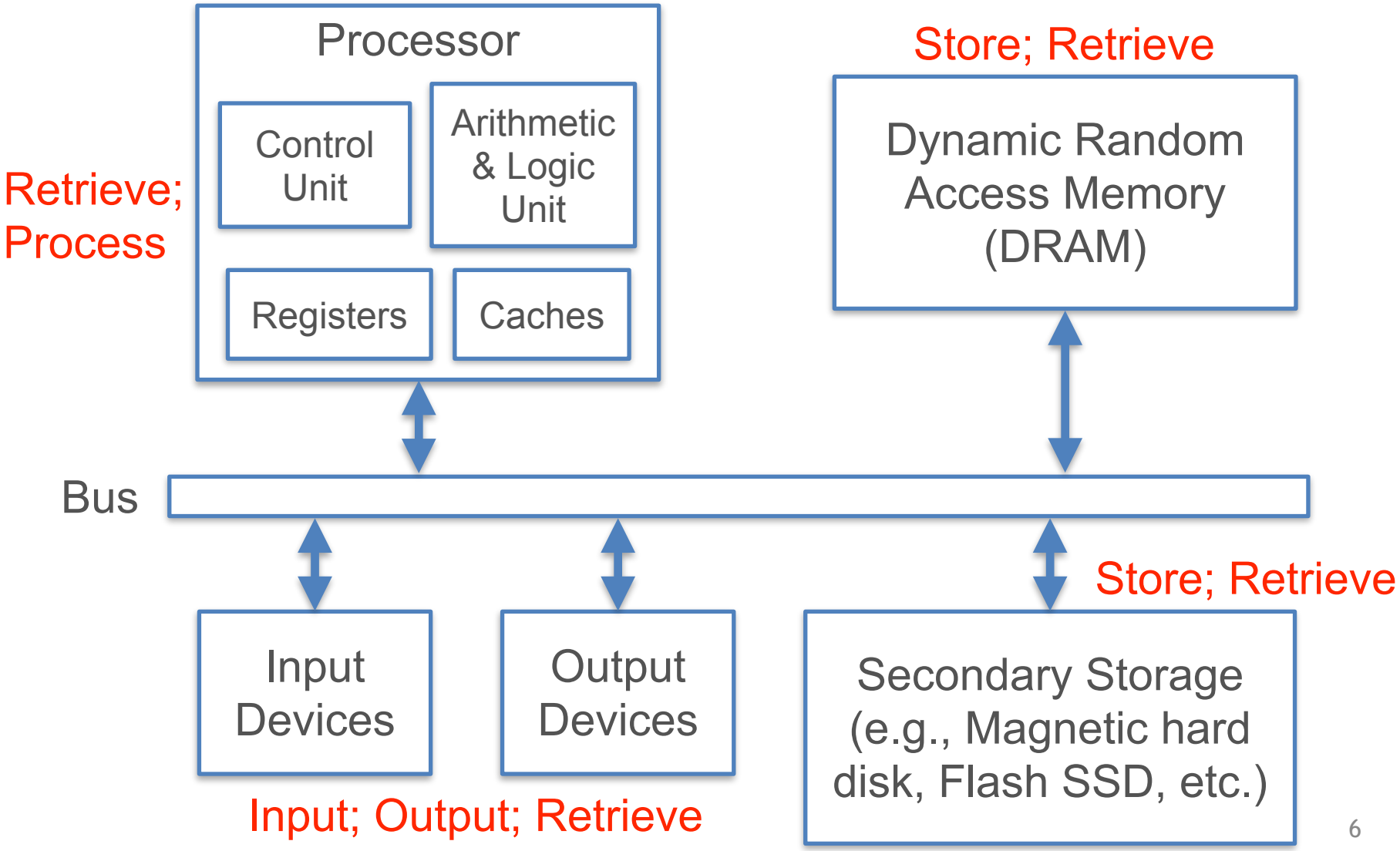
Software:
Programs (instructions) and data

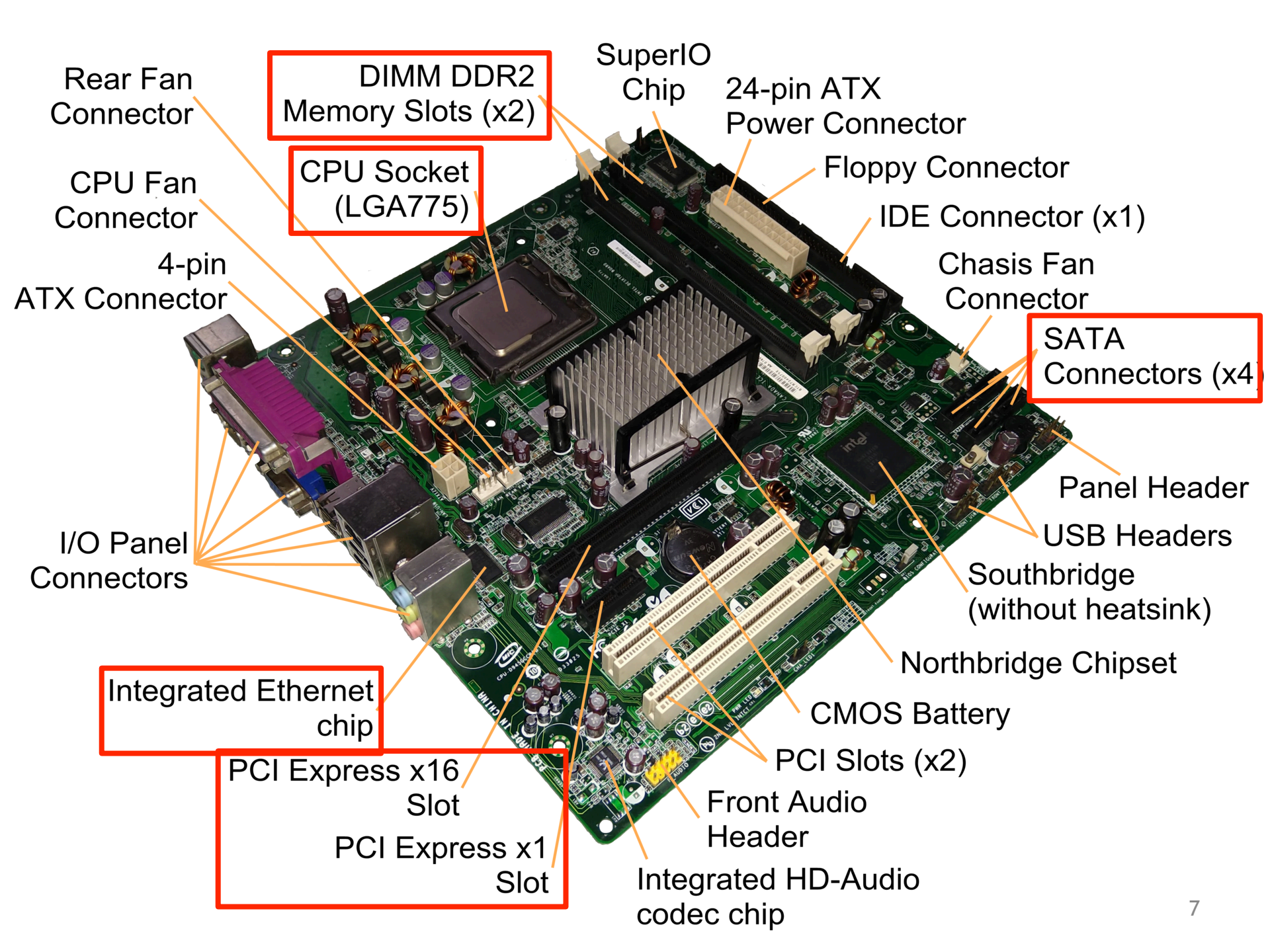
https://www.webopedia.com/TERM/C/computer.html

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.

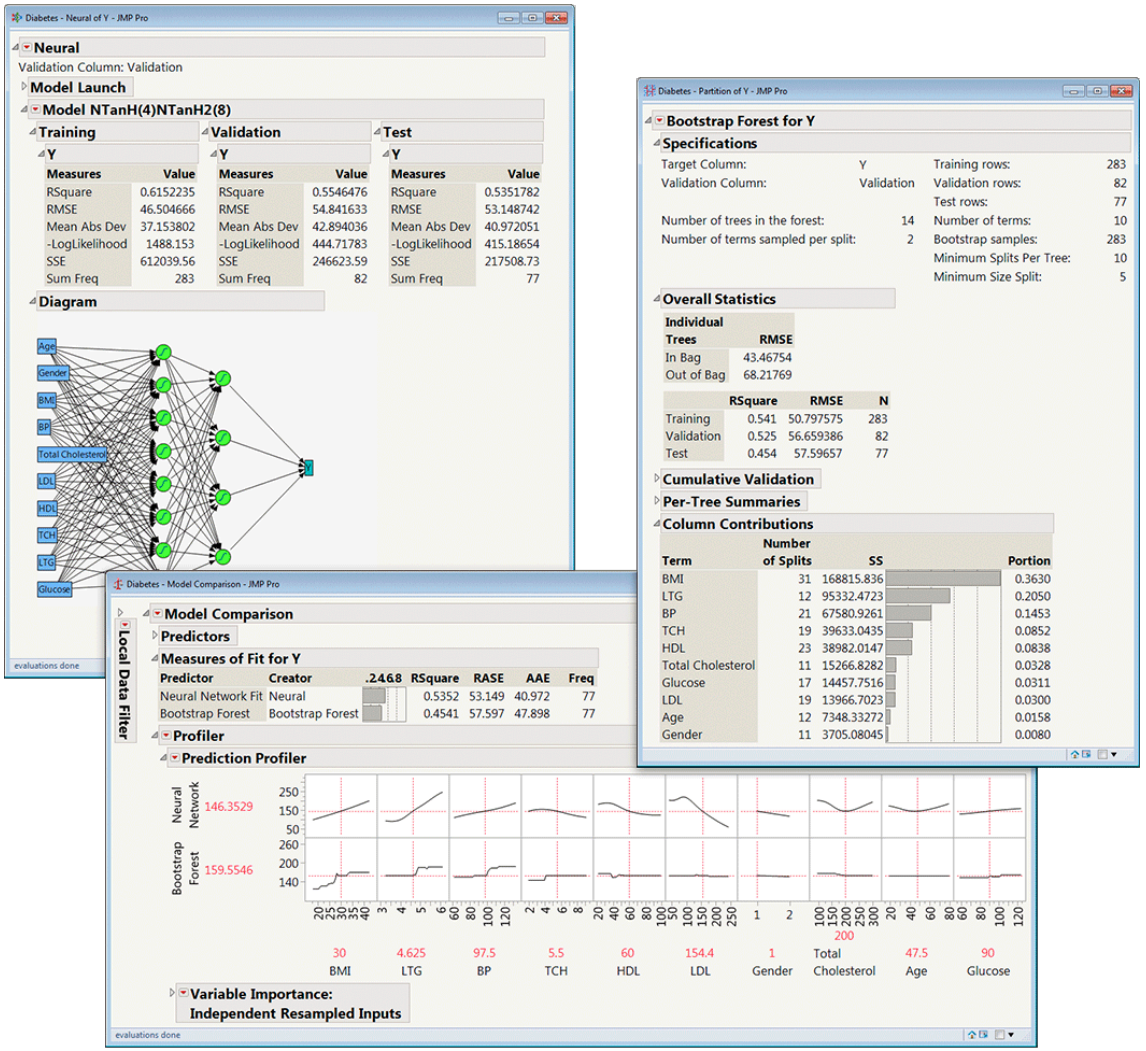
Outline

- 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?*

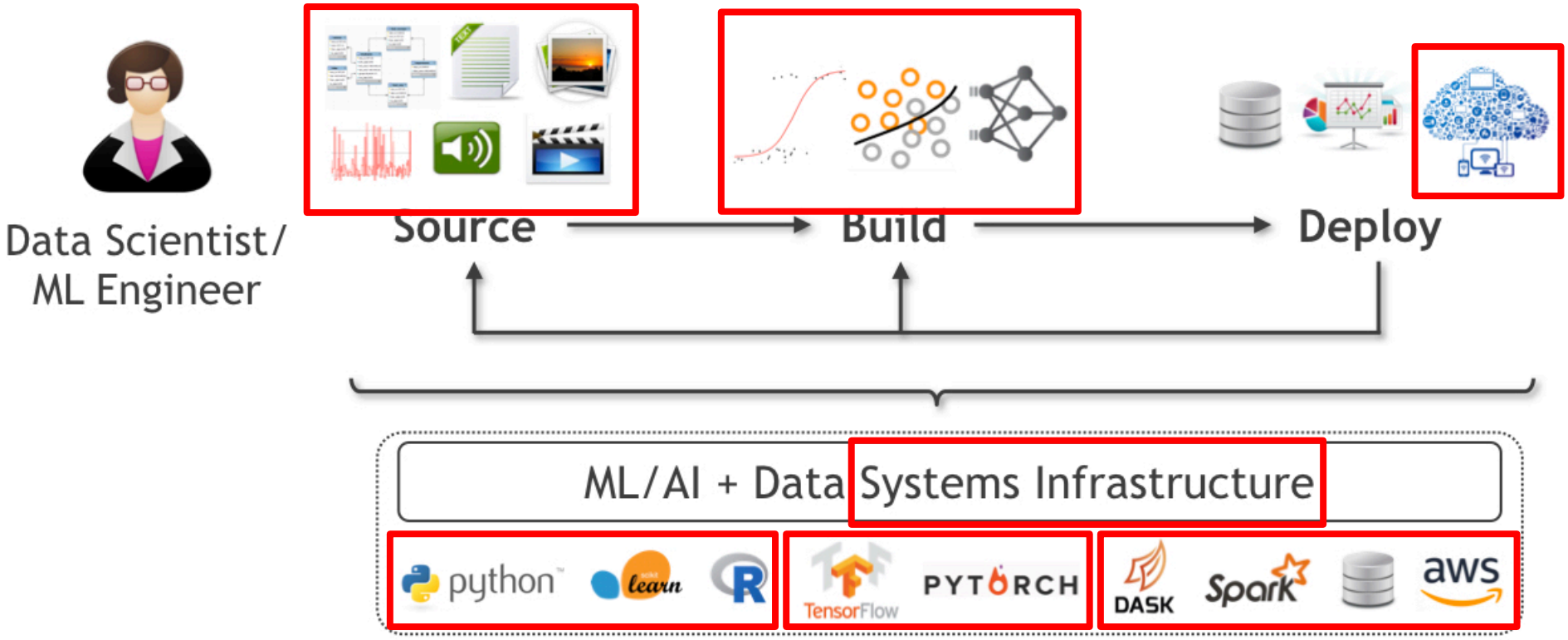
Luxury of “Statisticians”/“Analysts” of Yore

- **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

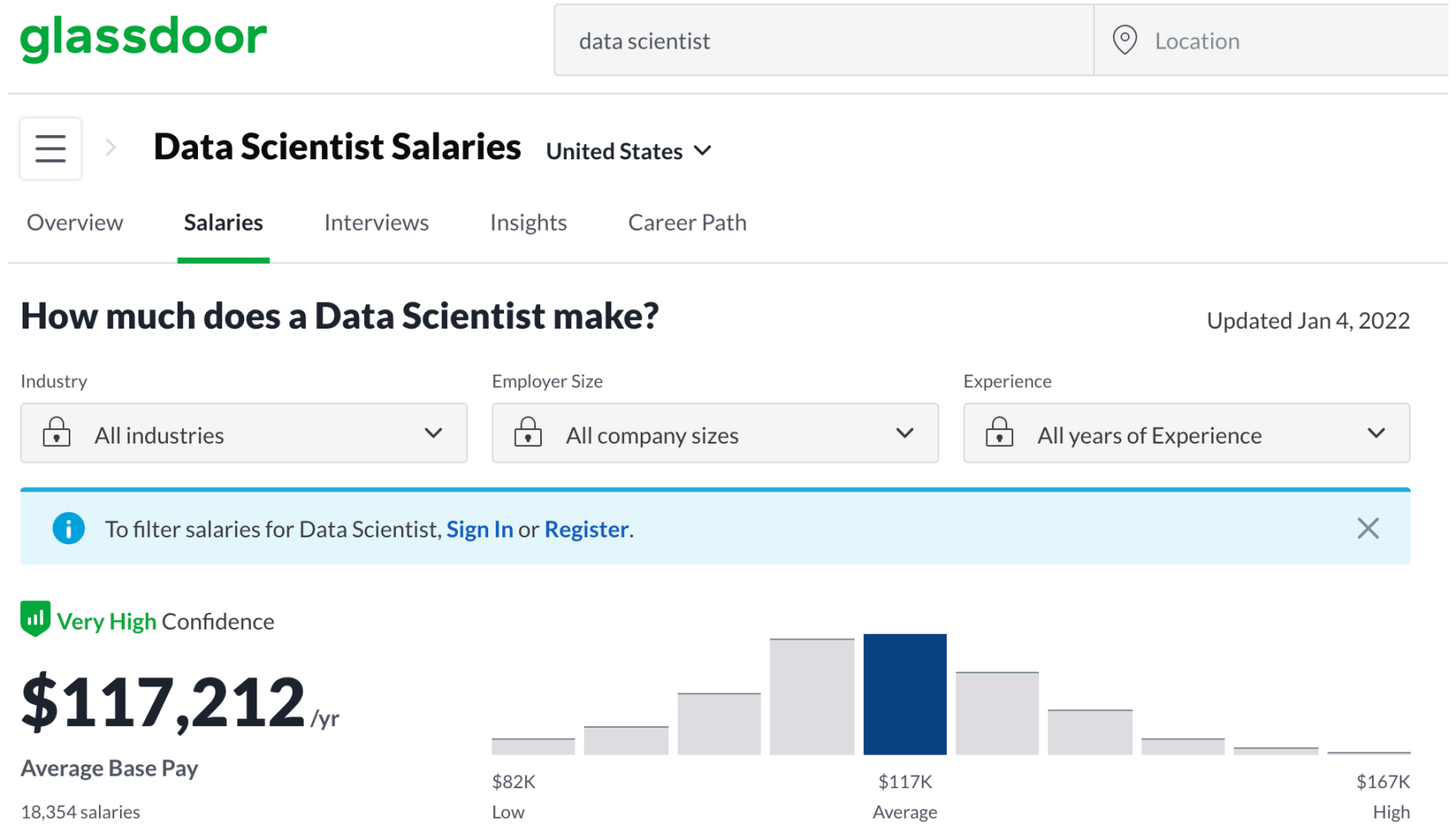
Very High Confidence

$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

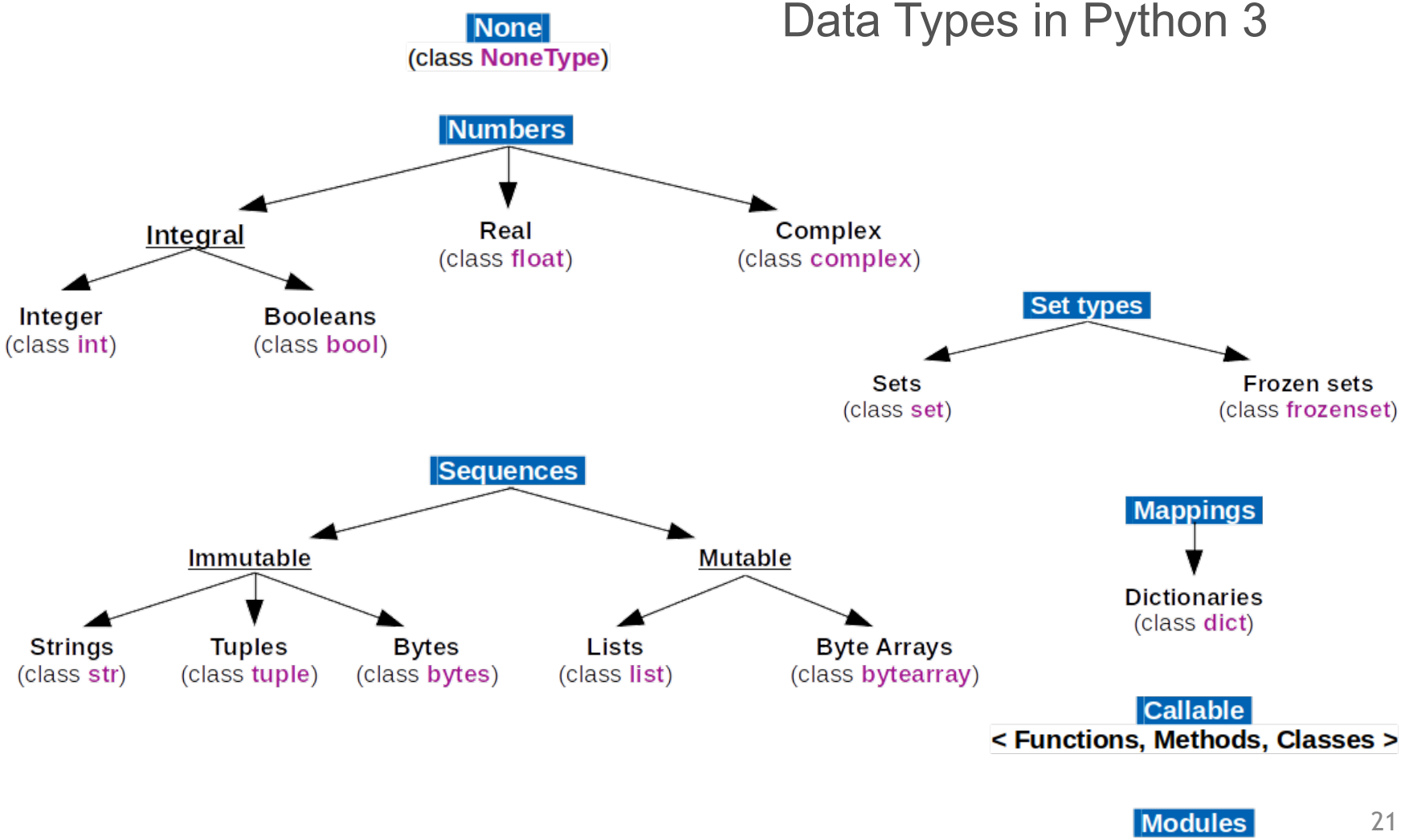
- Basics of Computer Organization
  - Digital Representation of Data
  - Processors and Memory Hierarchy
- Basics of Operating Systems
  - Process Management: Virtualization; Concurrency
  - Filesystem and Data Files
  - Main Memory Management
- Persistent Data Storage

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

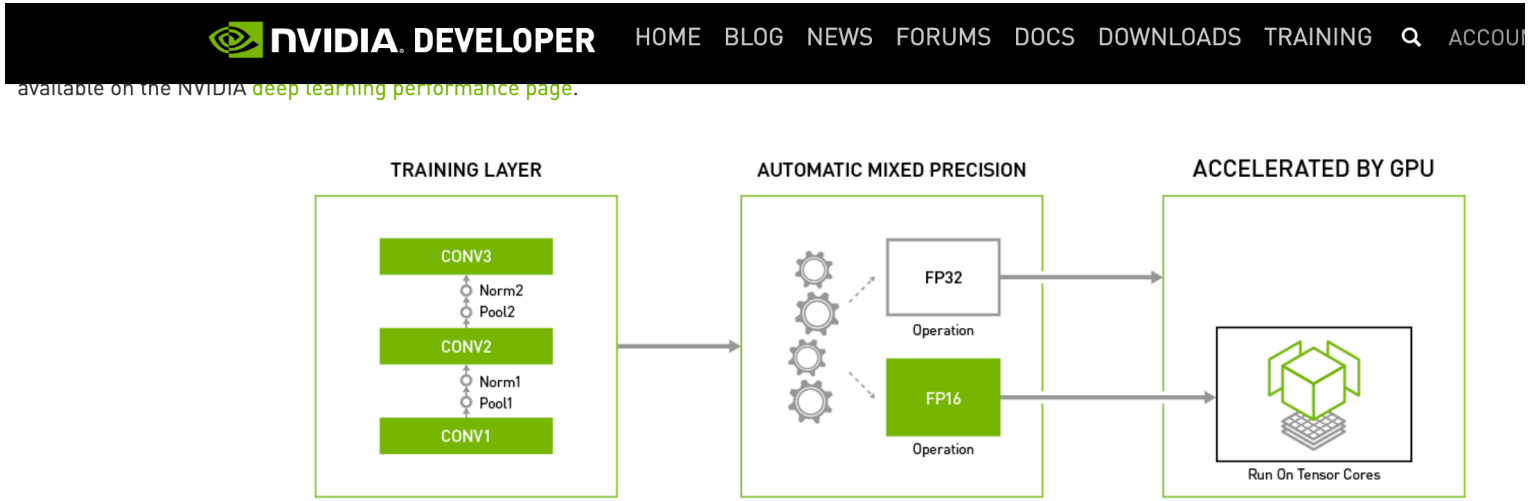
Decimal	Binary	Hexadecimal
5_{10}	101_2	5_{16}
47_{10}	$10\ 1111_2$	$2F_{16}$
163_{10}	$1010\ 0011_2$	$A3_{16}$
16_{10}	$1\ 0000_2$	10_{16}

Alternative notations

0xA3 or $A3_H$

Digital Representation of Data

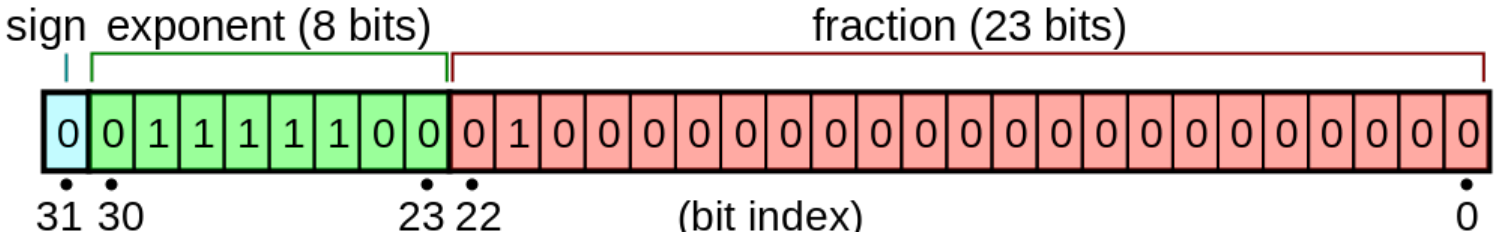
- **Float**:
  - Examples in data sci.: salary, scores, model weights
  - IEEE-754 single-precision format is 4B long; double-precision format is 8B long
  - Java and C *float* is single; Python *float* is double!



Using Automatic Mixed Precision for Major Deep Learning Frameworks

Digital Representation of Data

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



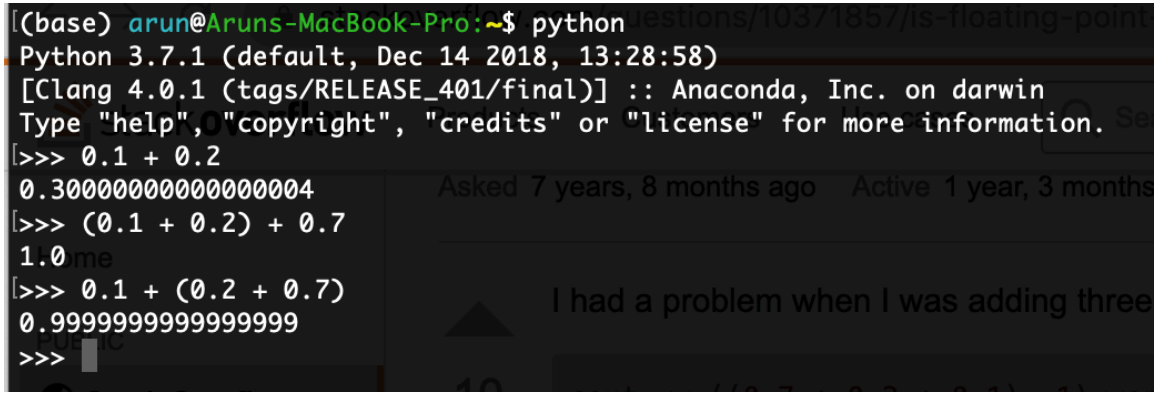
$$(-1)^{sign} \times 2^{exponent-127} \times (1 + \sum_{i=1}^{23} b_{23-i} 2^{-i})$$

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

(**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 ↓

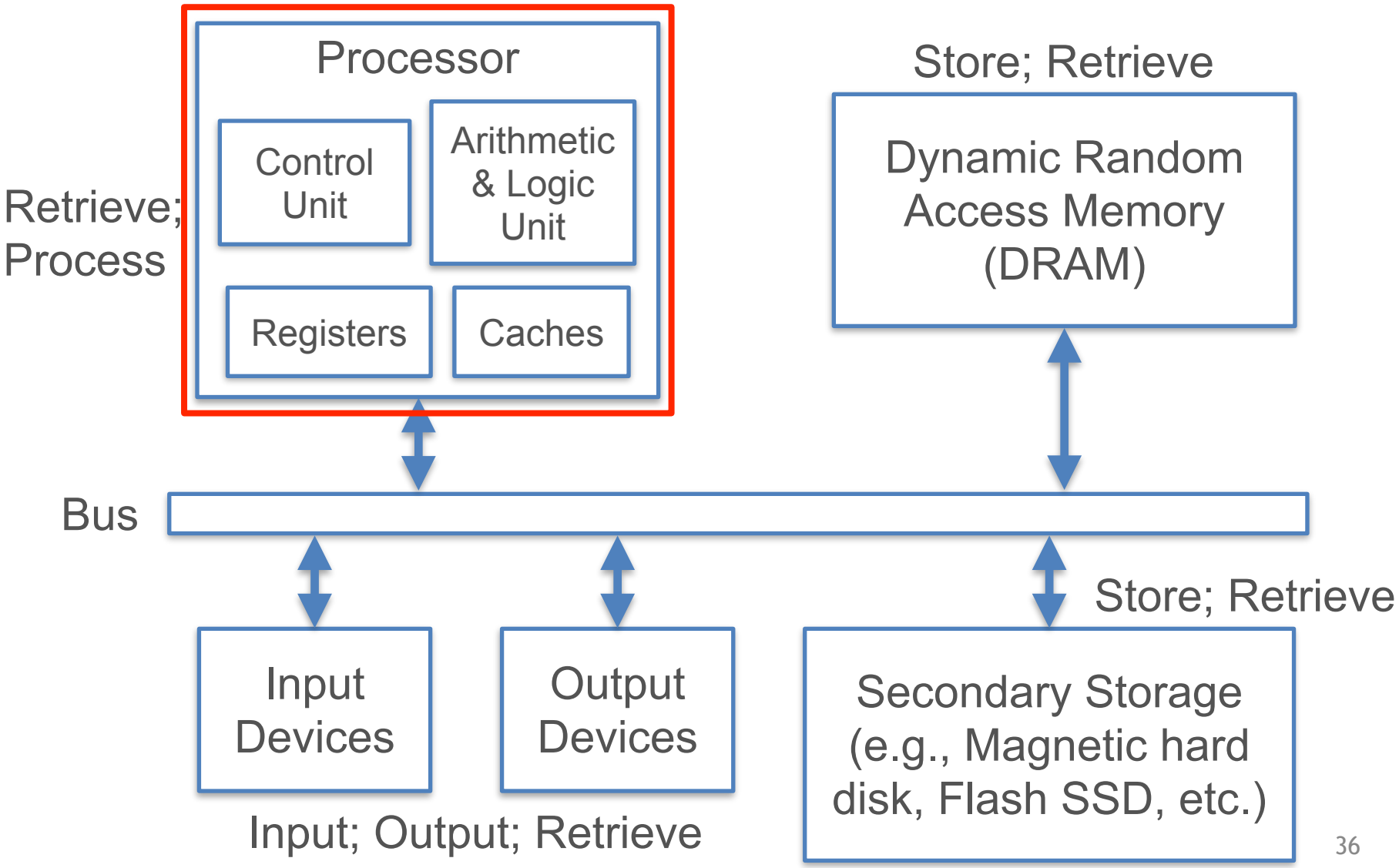
Machine code tied to ISA

↓

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

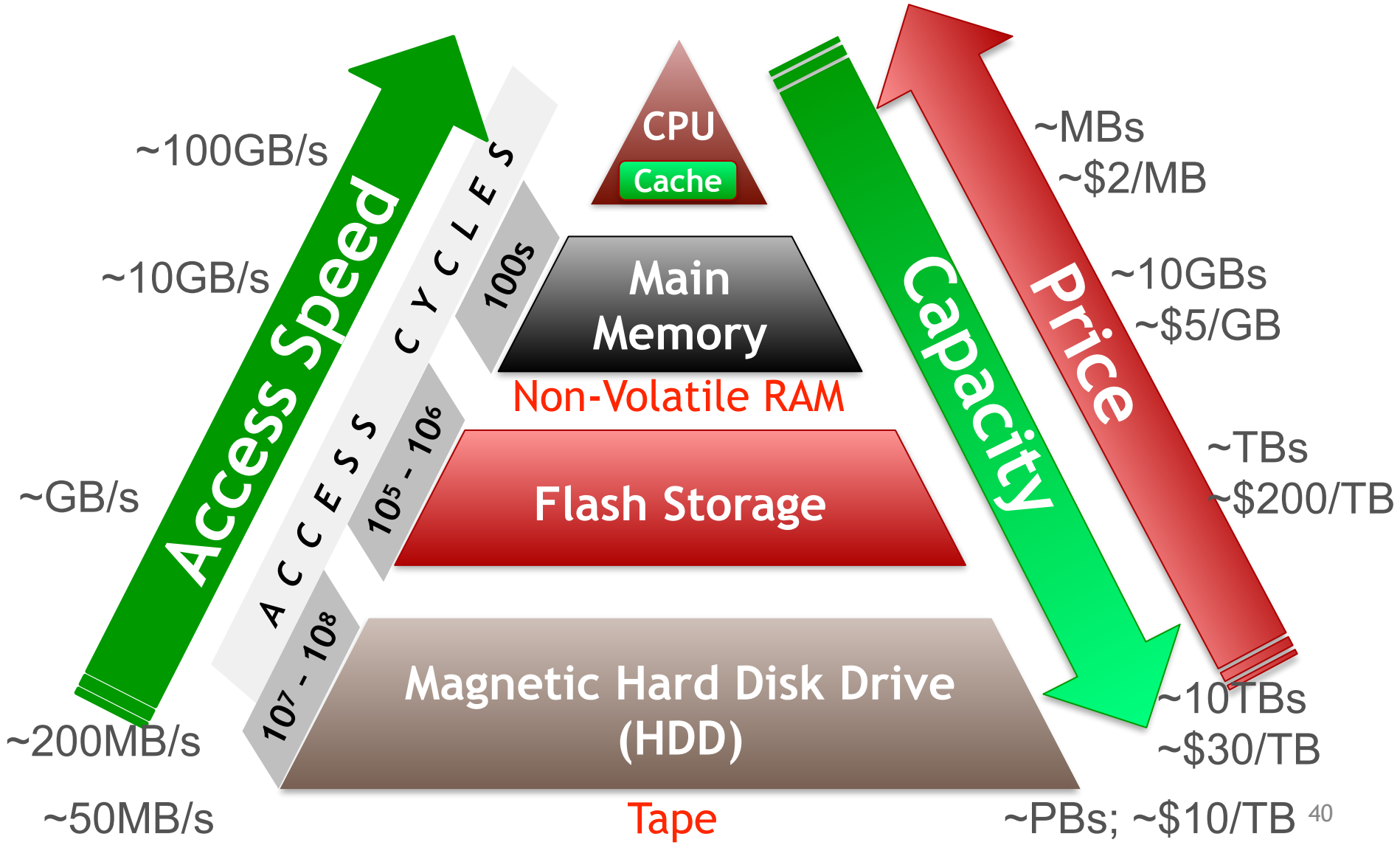
Processor Performance

***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

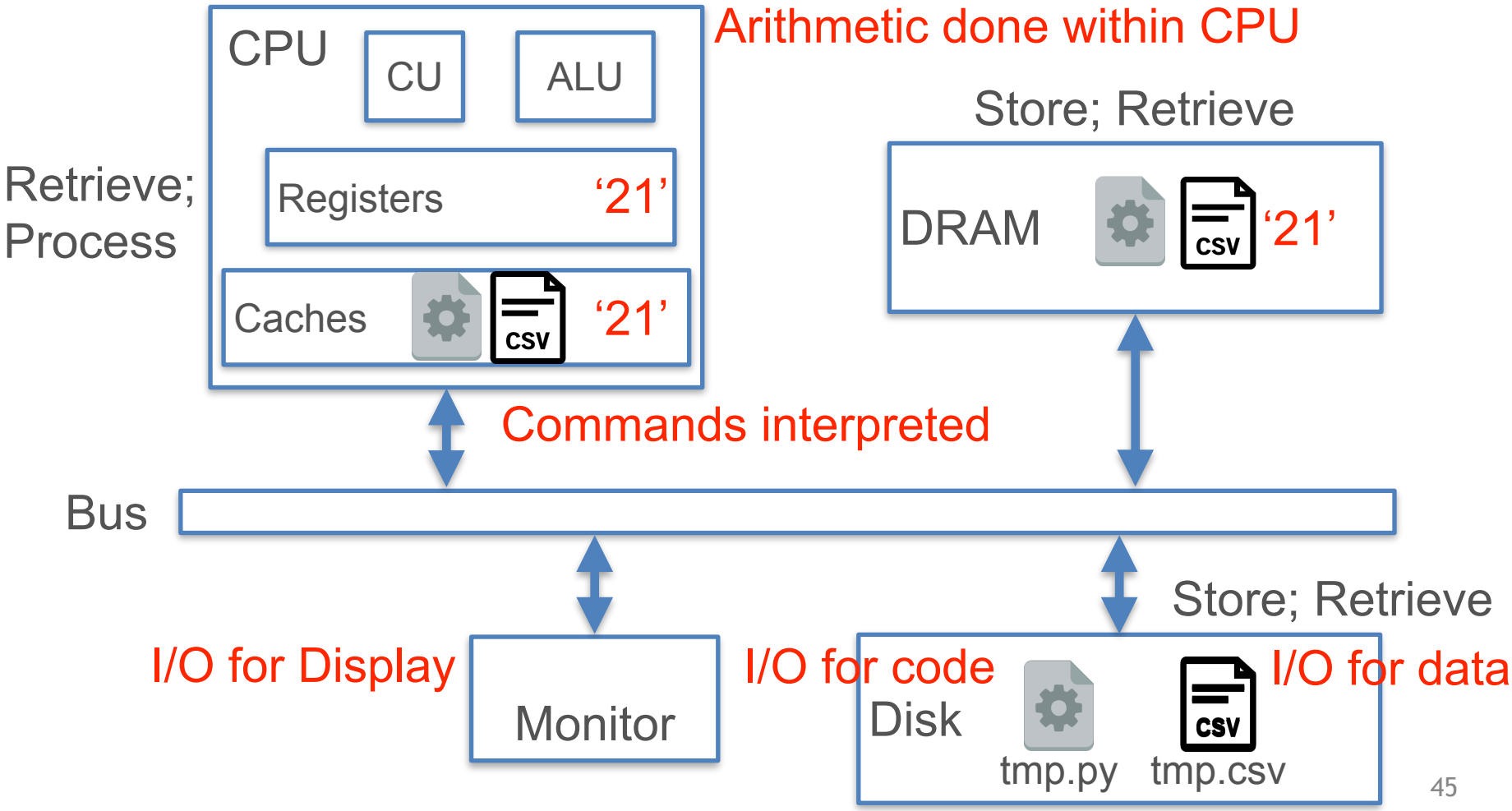
```
import pandas as p
m = p.read_csv('tmp.csv',header=None)
s = m.sum().sum()
print(s)
```

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

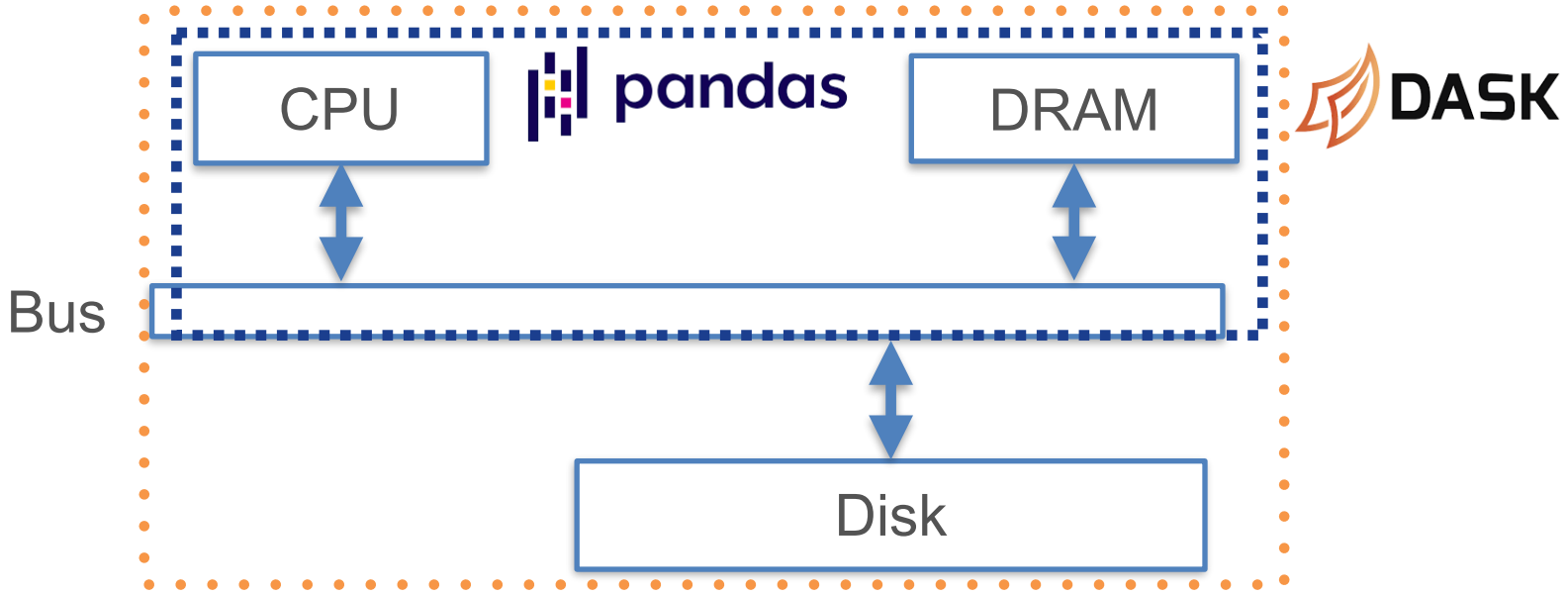
Breakout Rooms Activity (8min)

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