

# DSC 102

# Systems for Scalable Analytics

Haojian Jin

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

# Course Schedule

Week	Topic
Systems Principles 4	Basics of Machine Resources: Computer Organization
4-5	Basics of Machine Resources: Operating Systems Basics of Cloud Computing
Scalability Principles 5-7	Parallel and Scalable Data Processing: Parallelism Basics <b>Midterm Exam on Friday, Nov 3</b>
7-8	Parallel and Scalable Data Processing: Scalable Data Access Parallel and Scalable Data Processing: Data Parallelism
9	Dataflow Systems
10	ML Model Building Systems
11-12	<b>Final Exam on Dec 15</b>

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



# Large-Scale Data in Astronomy

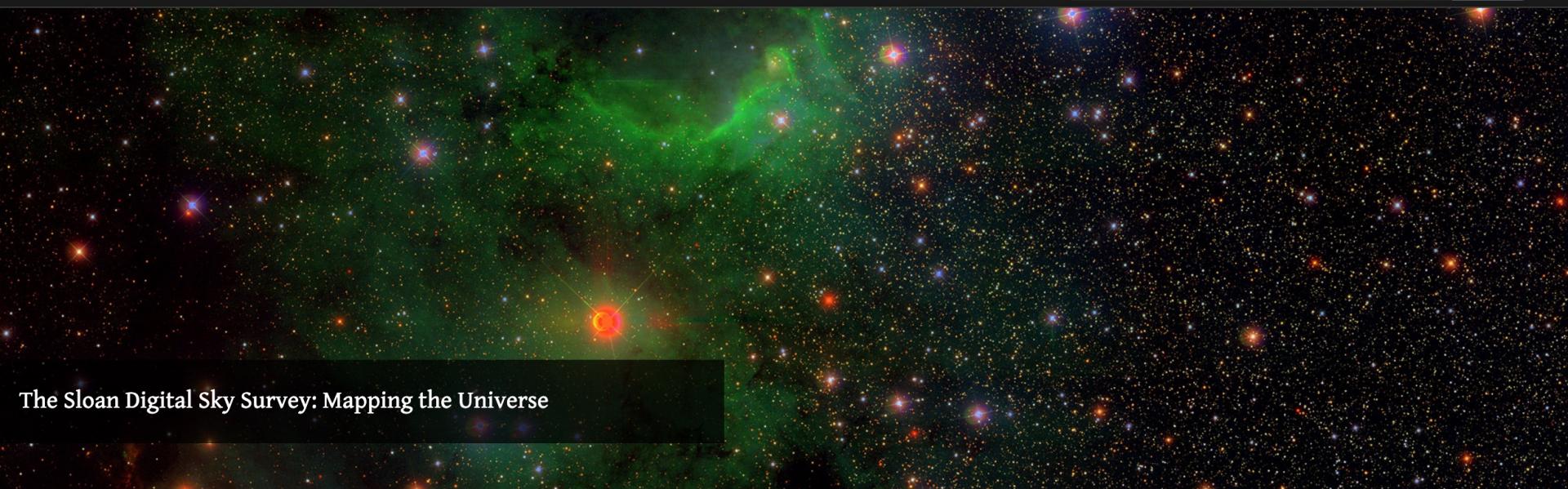


This is Data Release 16.

Data Surveys Instruments Collaboration Results Education The Future Contact

Search www.sdss.org

Search



The Sloan Digital Sky Survey: Mapping the Universe

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

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



# Large-Scale Data in Genomics

## UNDERSTANDING PRECISION MEDICINE

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



Precision Medicine is becoming a reality

Analyze genomes across cohorts and prescribe targeted drugs and treatments

~3GB genome per human  
~1EB for USA

NETFLIX ORIGINAL

# STRANGER THINGS

95% Match 2017 2 Seasons 4K Ultra HD 5.1

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

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



## Popular on Netflix



## Recently Watched



# Large-Scale Data in E-commerce



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

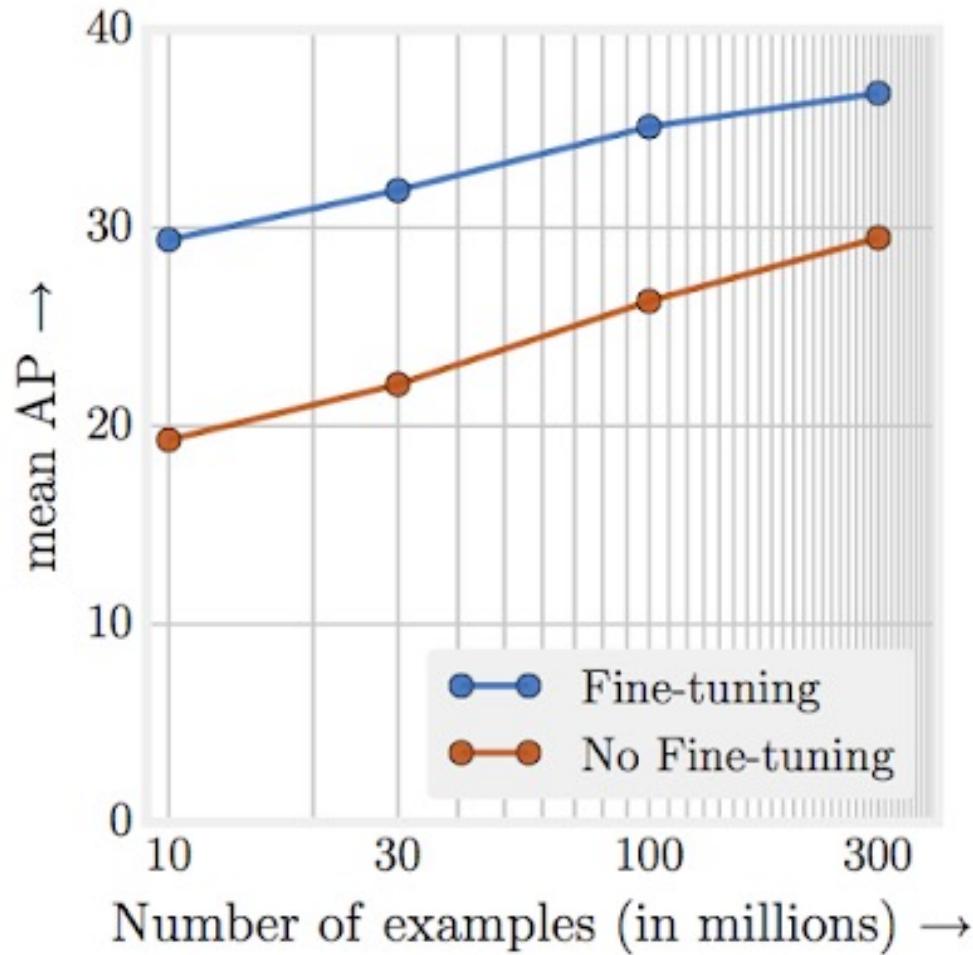


10million+ images labeled (20,000 classes) by crowdsourcing

>500GB uncompressed as tensors

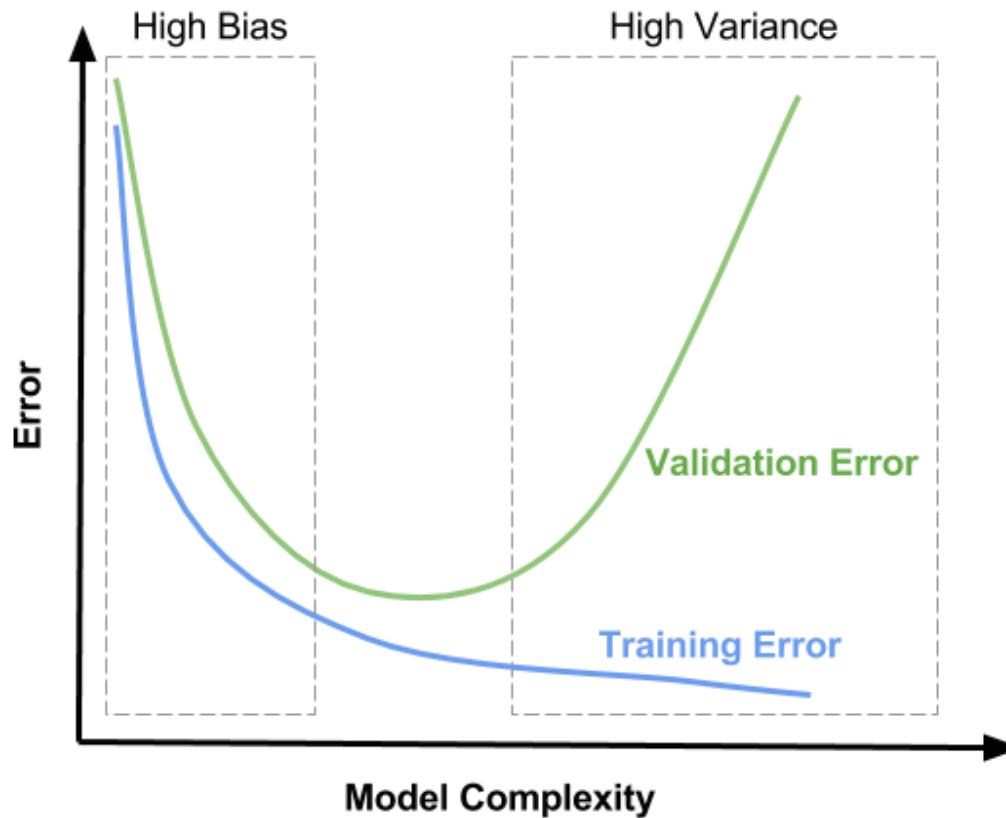
Harbinger of deep learning revolution

# “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

# 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)
  - ❖ Public cloud IaaS democratizing access to hardware
- ❖ Open source and SaaS are democratizing software

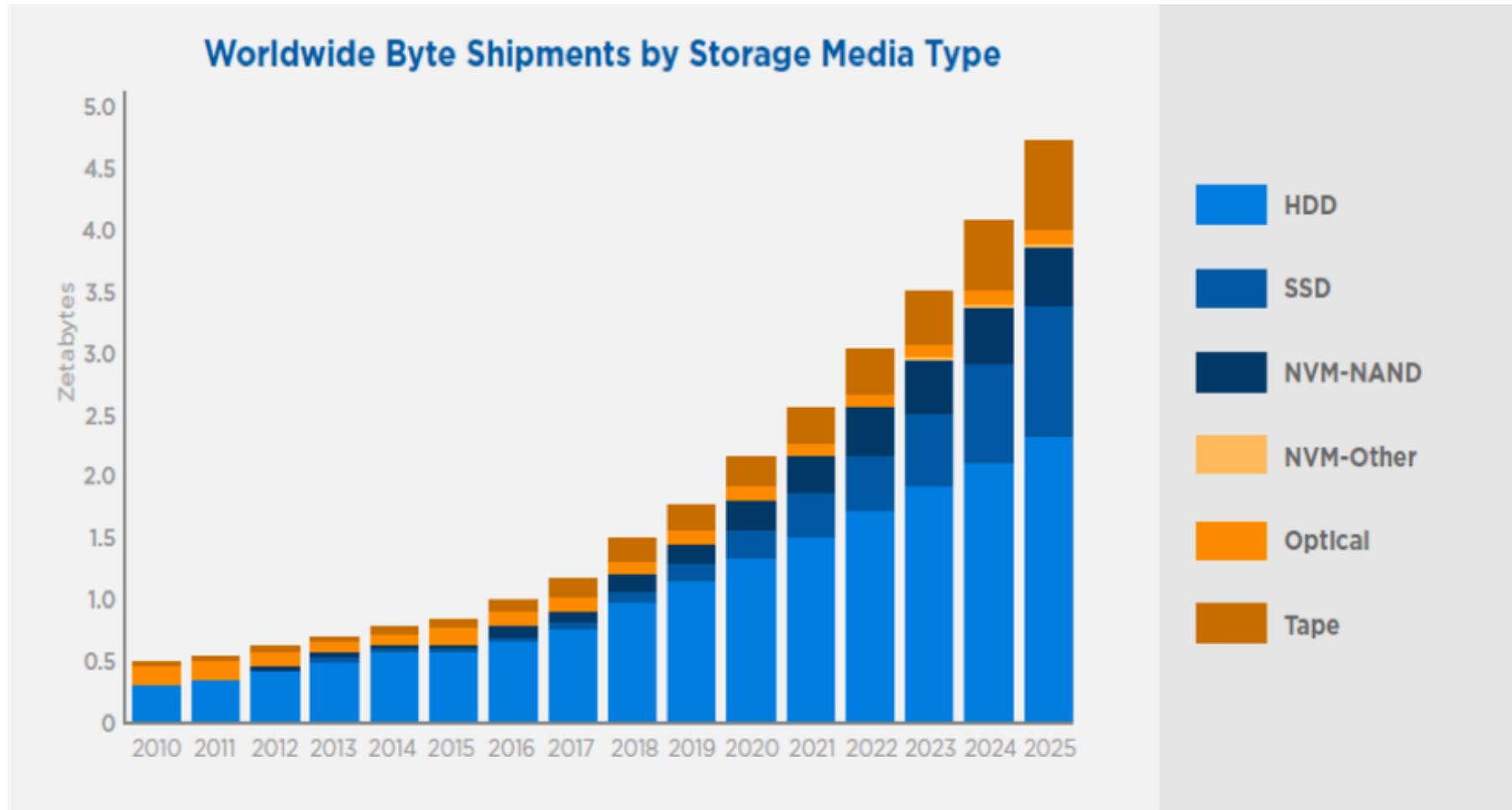
# “Big Data”

- ❖ 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



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

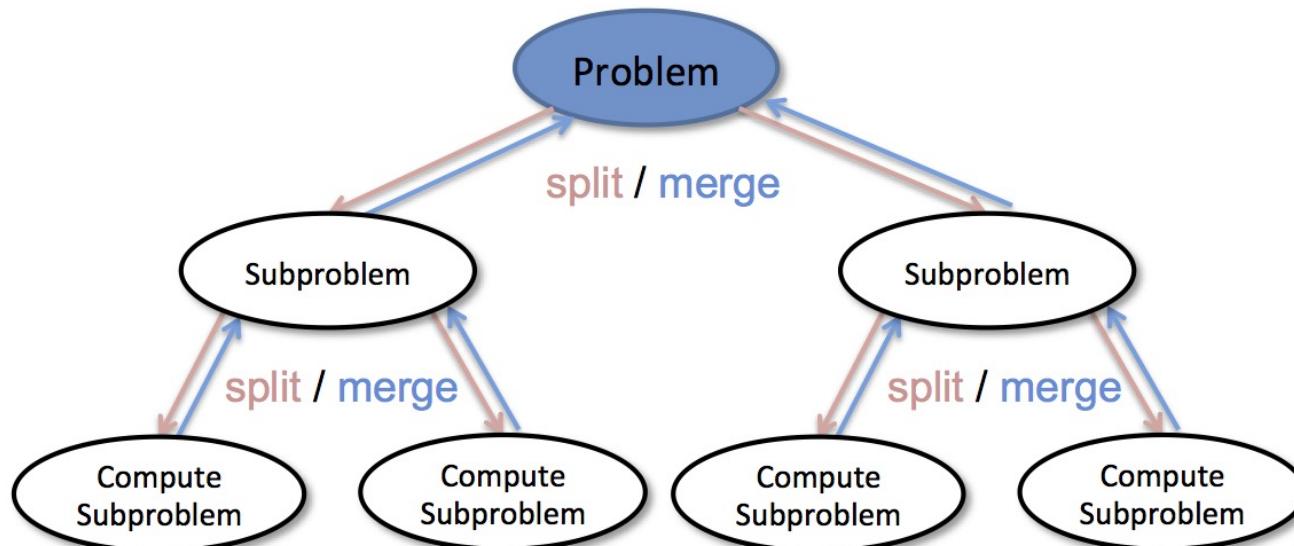
# Outline

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

# Parallel Data Processing

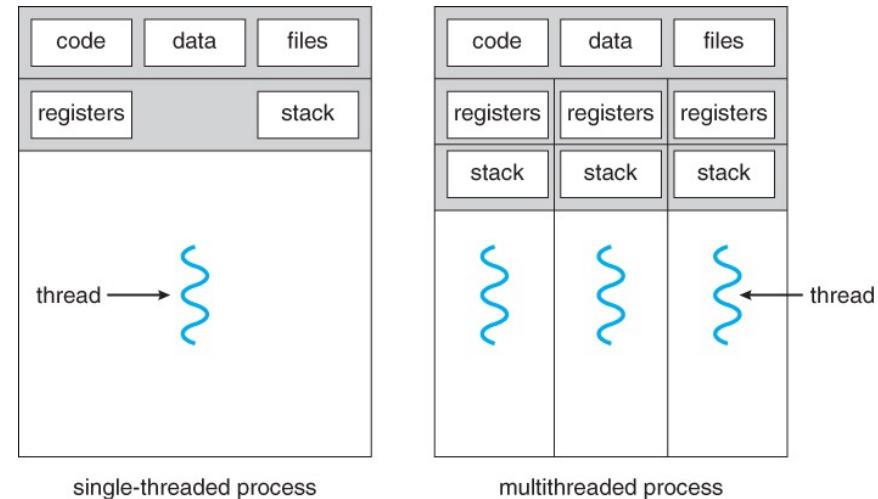
**Central Issue:** Workload takes too long for one processor!

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



# New Parallelism Concept: Threads

- ❖ **Threads:** Generalization of process abstraction of OS
  - ❖ Common in parallel data processing
- ❖ A multi-threaded program/process *spawns* 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

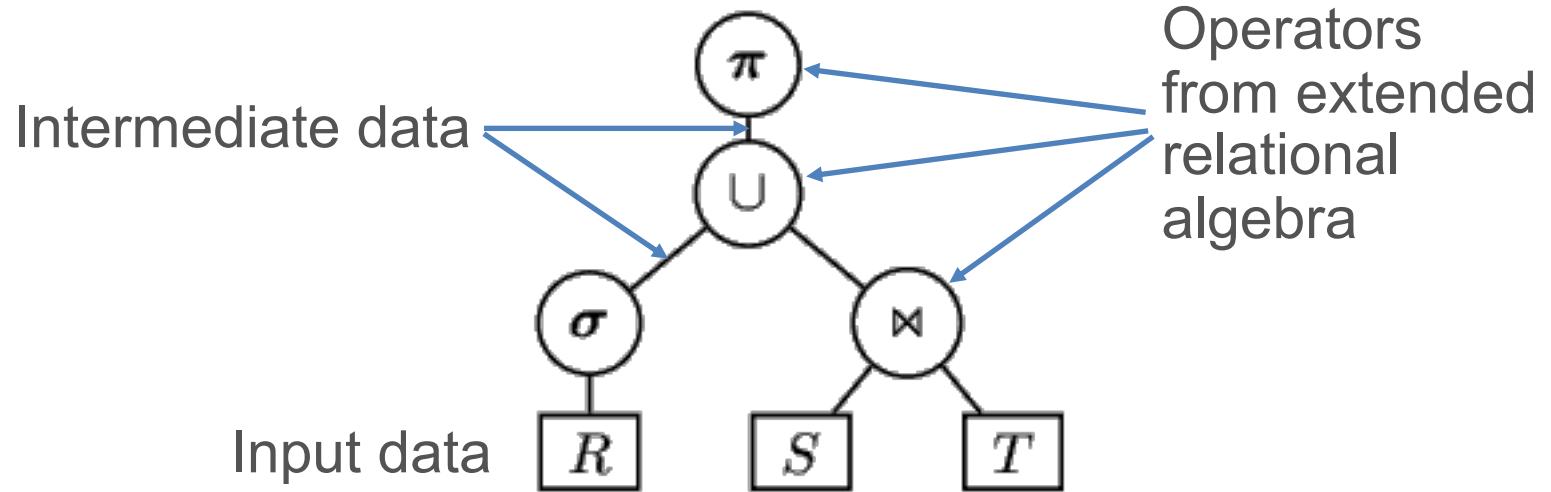


# New Parallelism Concept: Dataflow

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

# Example Relational Dataflow Graph

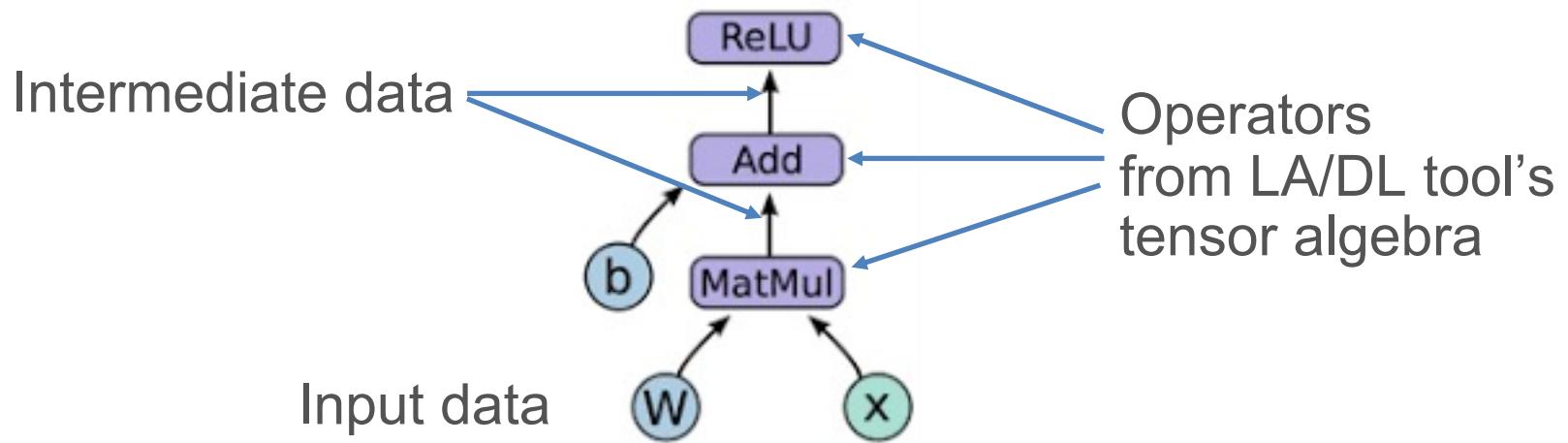
$$\pi(\sigma(R) \cup S \bowtie T)$$



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

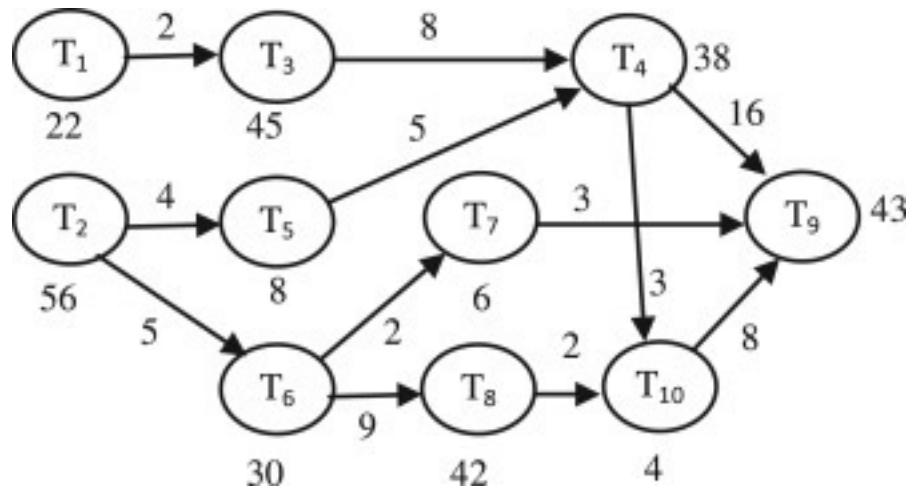
# Example Tensor Dataflow Graph

$$ReLU(WX + b)$$



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

# Example Task Graph



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

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

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

# Parallel Data Processing

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

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

**Key parallelism paradigms in data systems:**

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

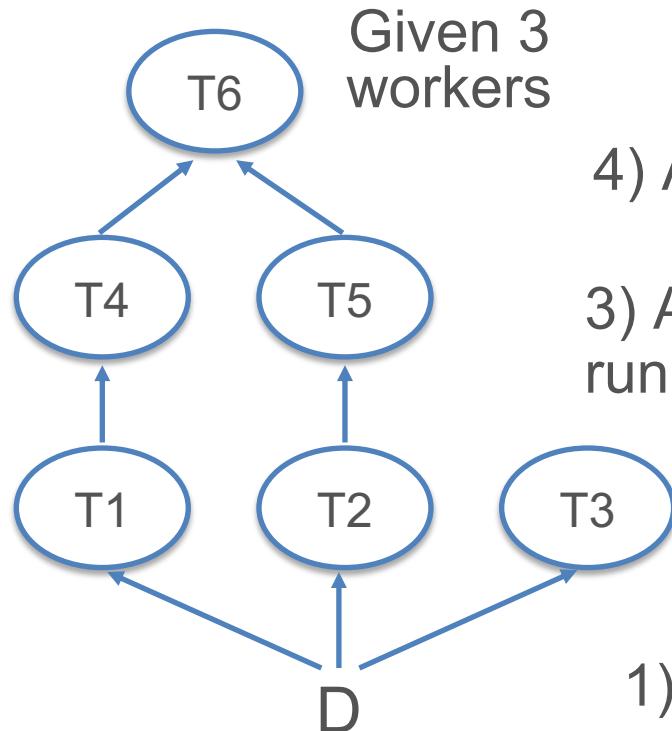
# Outline

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

# Task Parallelism

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

**Example:**



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

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

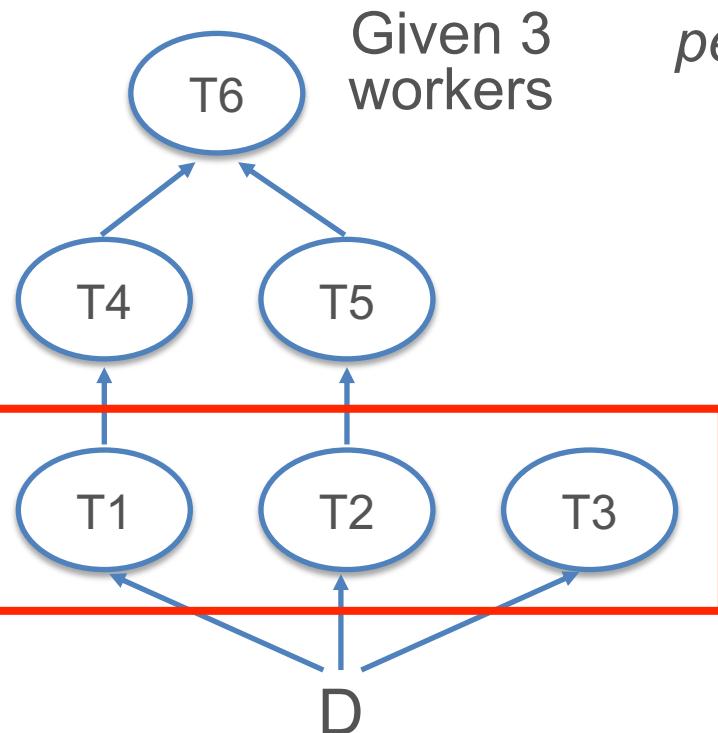
# Task Parallelism

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

# Degree of Parallelism

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

## Example:



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

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

Degree of parallelism is only 3

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

# Quantifying Benefit of Parallelism: Speedup

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

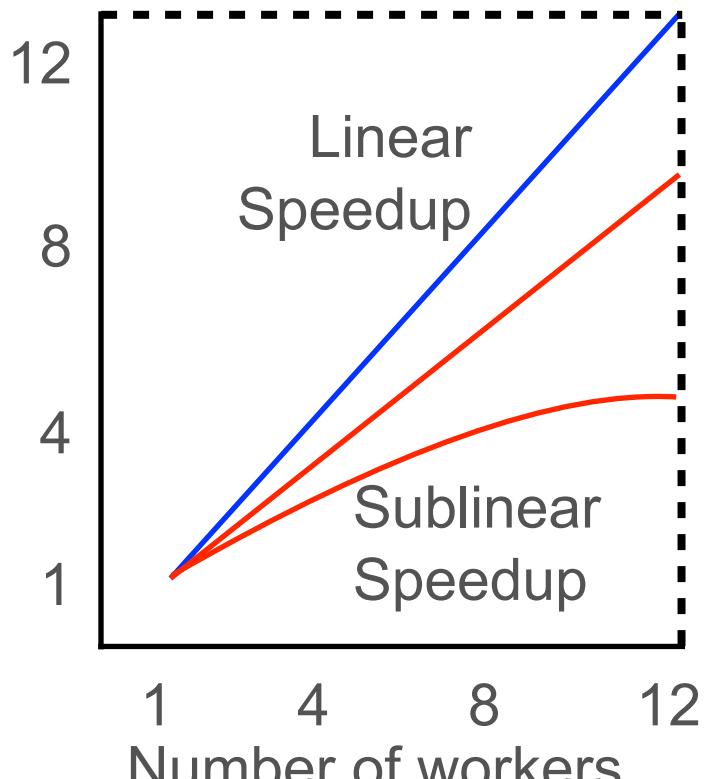
*Q: But given  $n$  workers, can we get a speedup of  $n$ ?*

It depends!

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

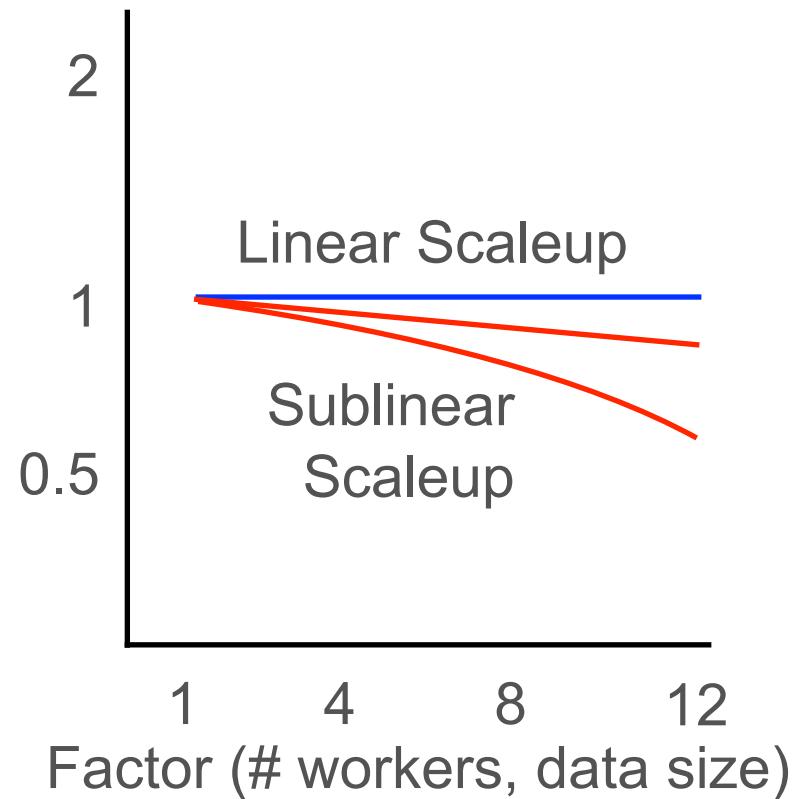
# Quantifying Benefit of Parallelism

Runtime speedup (fixed data size)



**Speedup** plot / Strong scaling

Runtime speedup



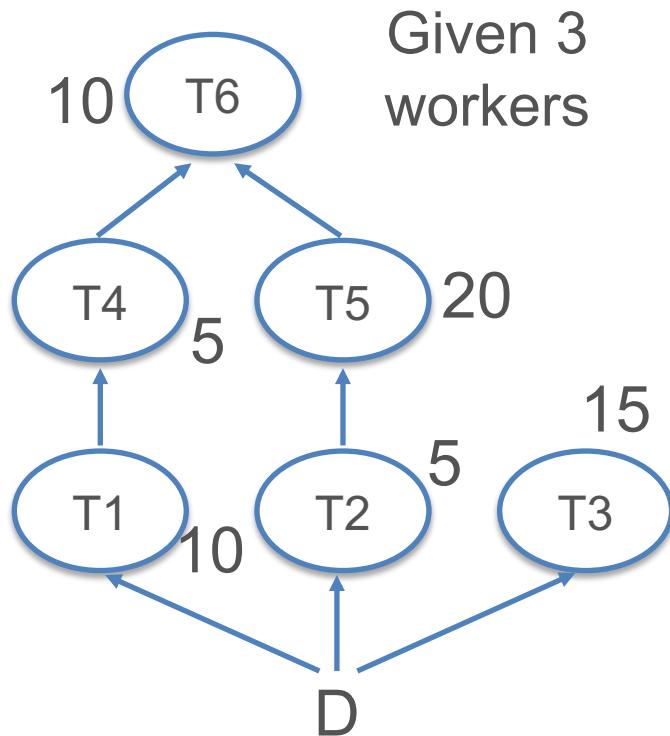
**Scaleup** plot / Weak scaling

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

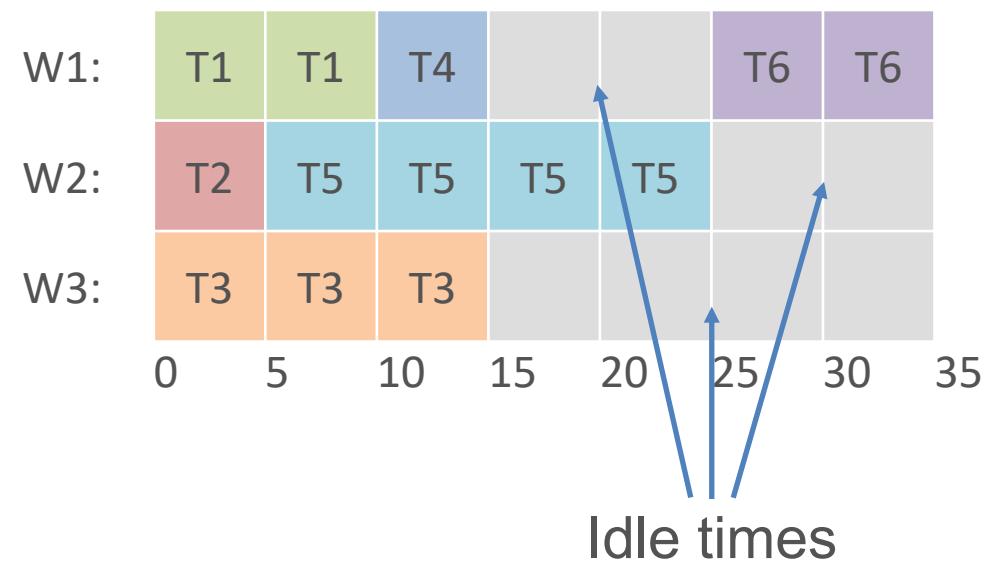
# Idle Times in Task Parallelism

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

**Example:**



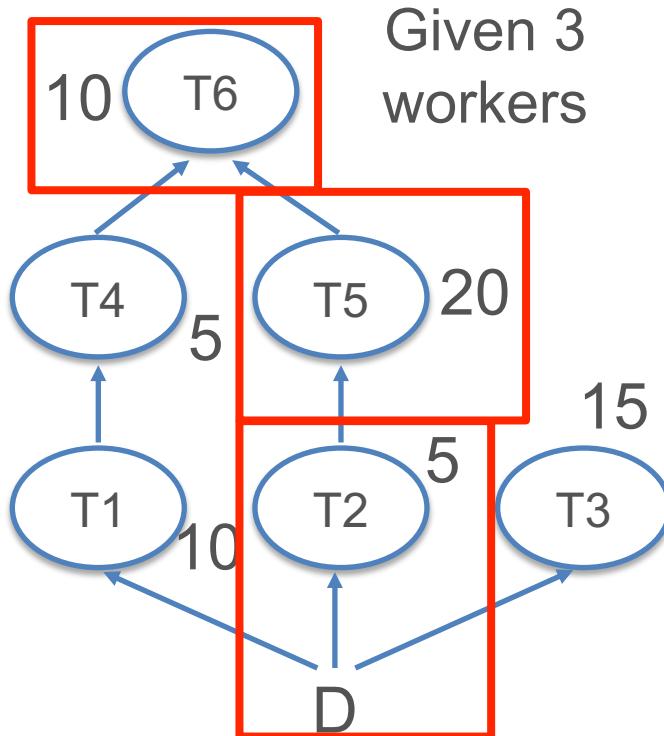
Gantt Chart visualization of schedule:



# Idle Times in Task Parallelism

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

## Example:

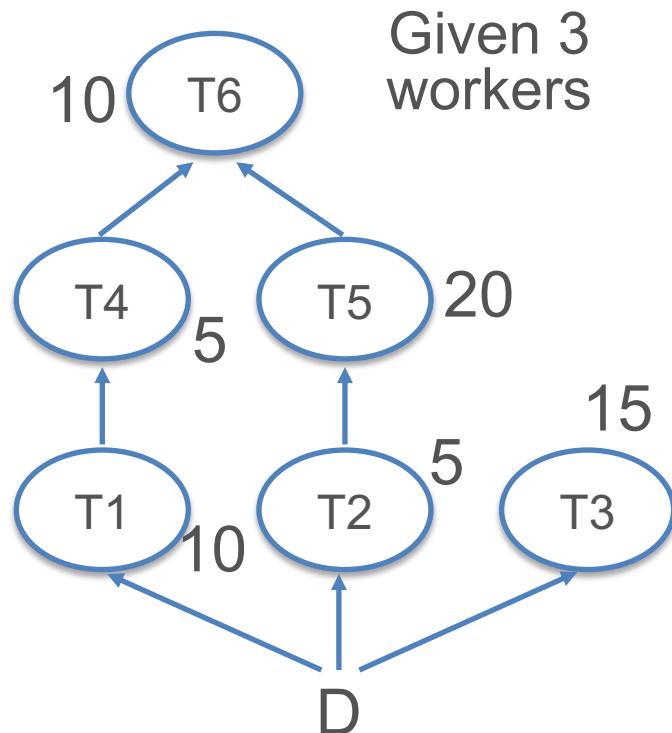


- ❖ In general, overall workload's completion time on task-parallel setup is always *lower bounded* by the **longest path** in the task graph
- ❖ Possibility: A task-parallel scheduler can “release” a worker if it knows that will be idle till the end
  - ❖ Can saves costs in cloud

# Calculating Task Parallelism Speedup

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

## Example:



Given 3 workers

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  $1.9x$ ?

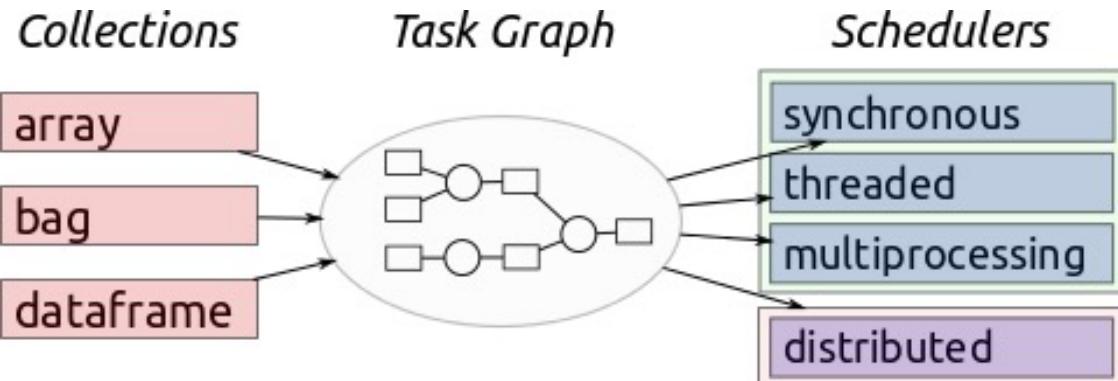
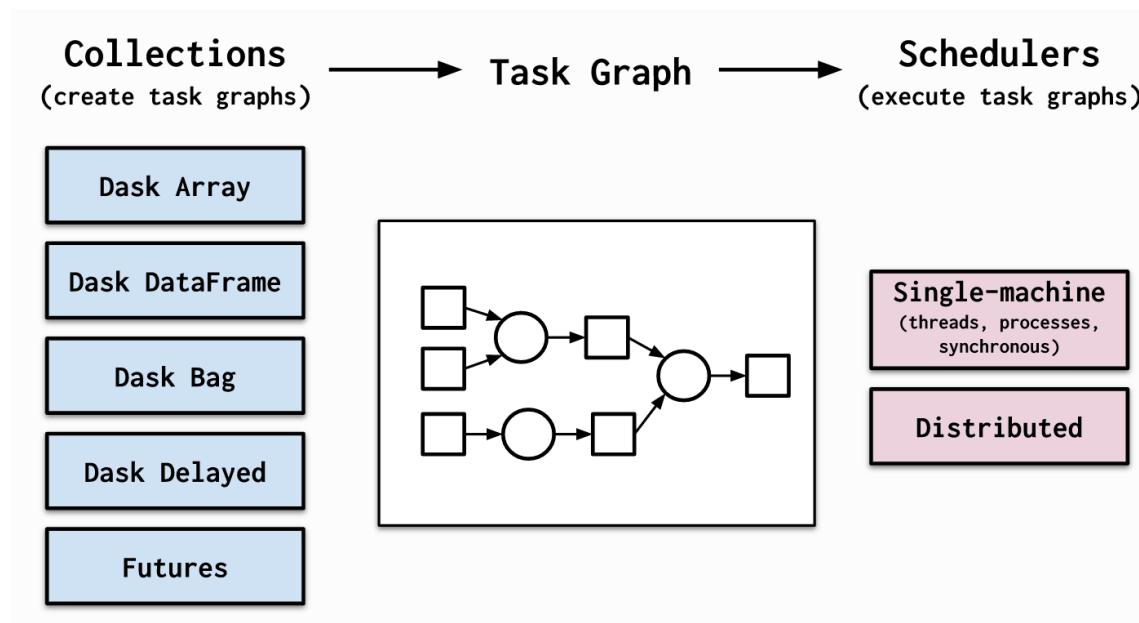
# Peer Instruction Activity

(Switch slides)

# Task Parallelism in Dask

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

# Task Parallelism in Dask



# Dask's Workflow

## ❖ “Lazy Evaluation”:

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

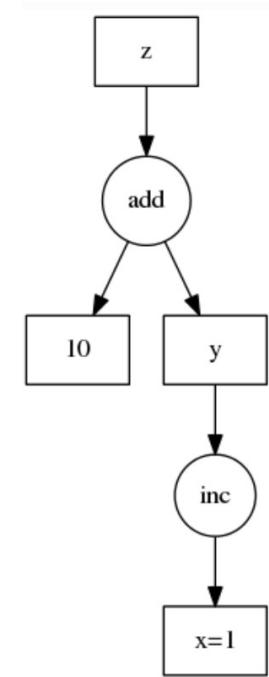
```
def inc(i):
    return i + 1

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

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



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

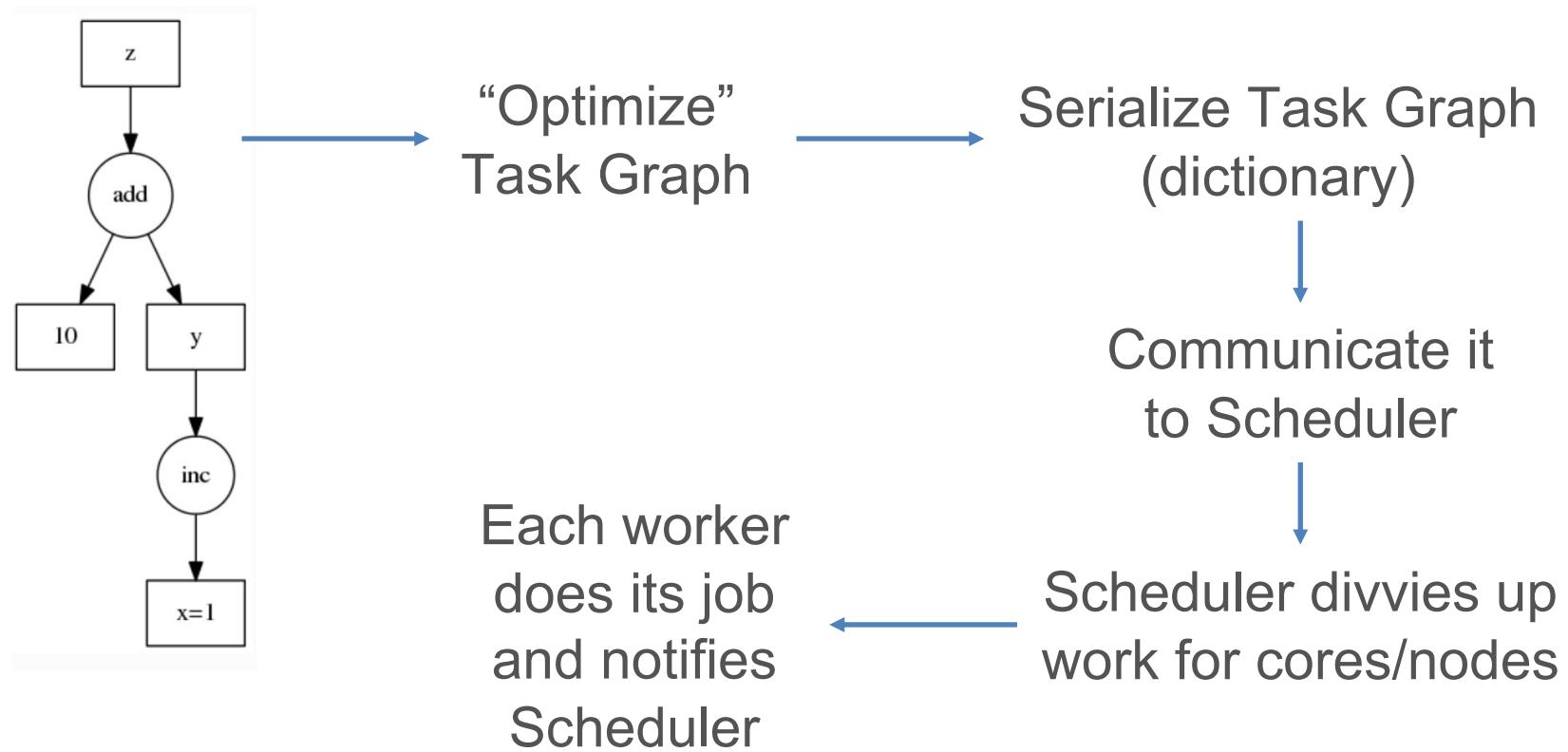


User code  
using their API

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

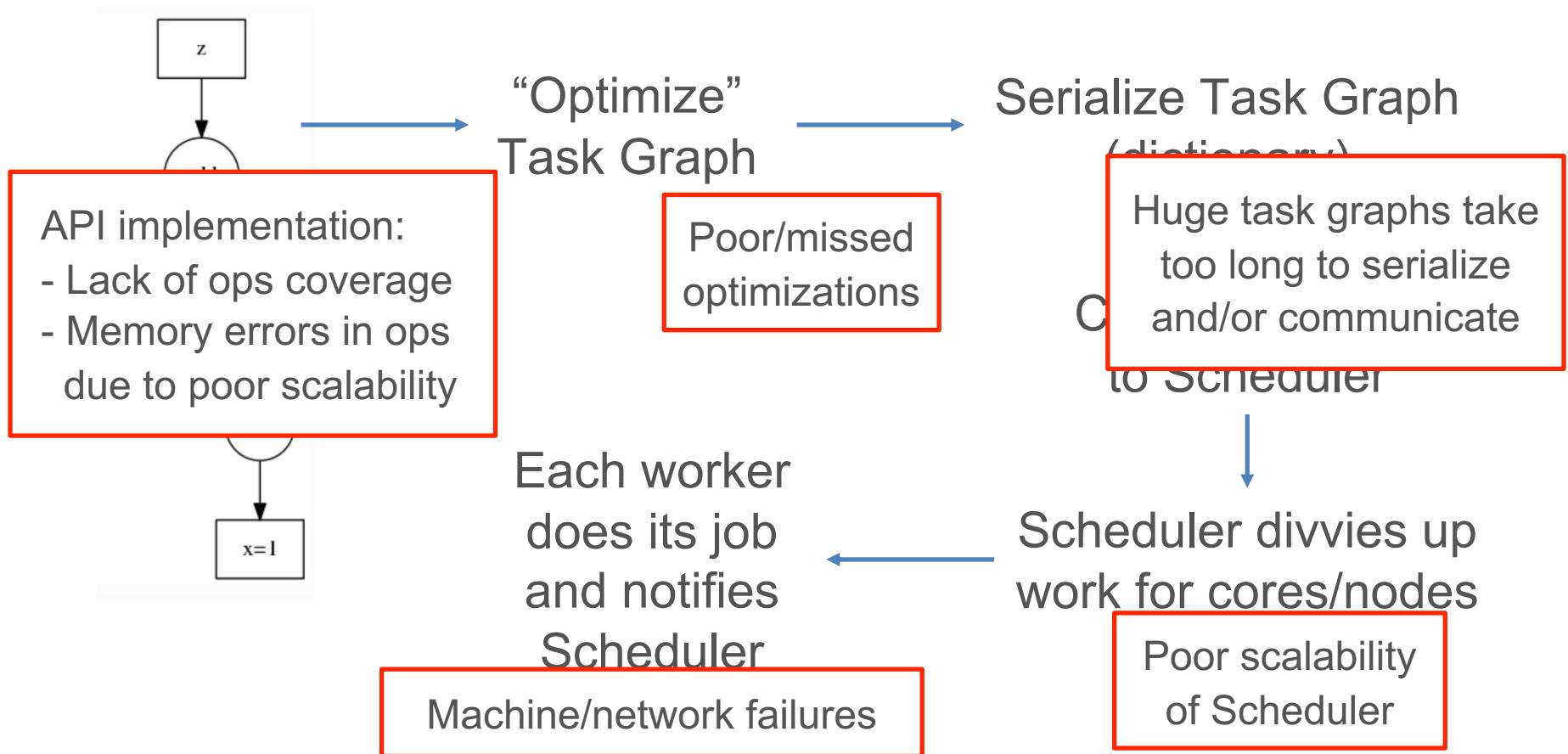
# Dask's Workflow

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



# Possible Bottlenecks/Issues in Dask

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



# Best Practices for Task-Par. Dask

- ❖ **Tuning task graph sizes:**
  - ❖ Avoid too many tasks (ser./comm./sched. bottlenecks) but also avoid too few tasks (under-utilization of cores/nodes)
  - ❖ Be mindful of available # cores/nodes
- ❖ Rough guidelines they give:
  - ❖ Adjust # tasks by fusing ops/tasks (aka “**batching**”) vs. breaking up ops/tasks
  - ❖ Adjust # tasks by tuning data chunk size (next slide)
- ❖ **Use the Diagnostics dashboard:**
  - ❖ Monitor # tasks, core/node usage, task completion

# Best Practices for Task-Par. Dask

- ❖ **Tuning data chunk sizes:**
  - ❖ Avoid too few chunks (low degree of par.) but also too many chunks (task graph overhead)
  - ❖ Be mindful of available DRAM
- ❖ Rough guidelines they give:
  - ❖ # data chunks  $\sim$  3x-10x # cores, but
  - ❖ # cores x chunk size must be  $<$  machine DRAM, but
  - ❖ chunk size should not be too small either ( $\sim$ 1 GB is OK)

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

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

# 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

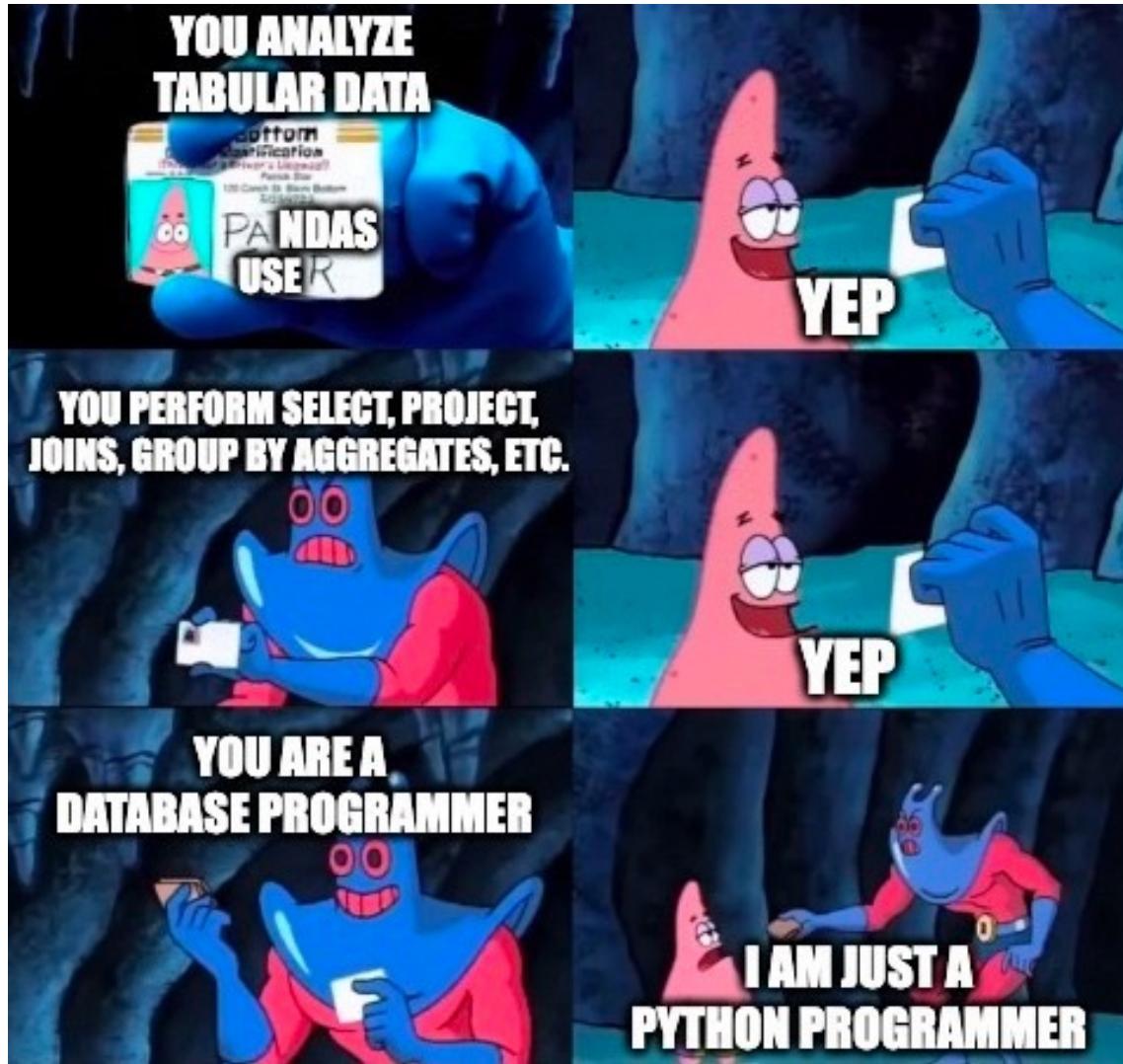
E. F. CODD

*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.

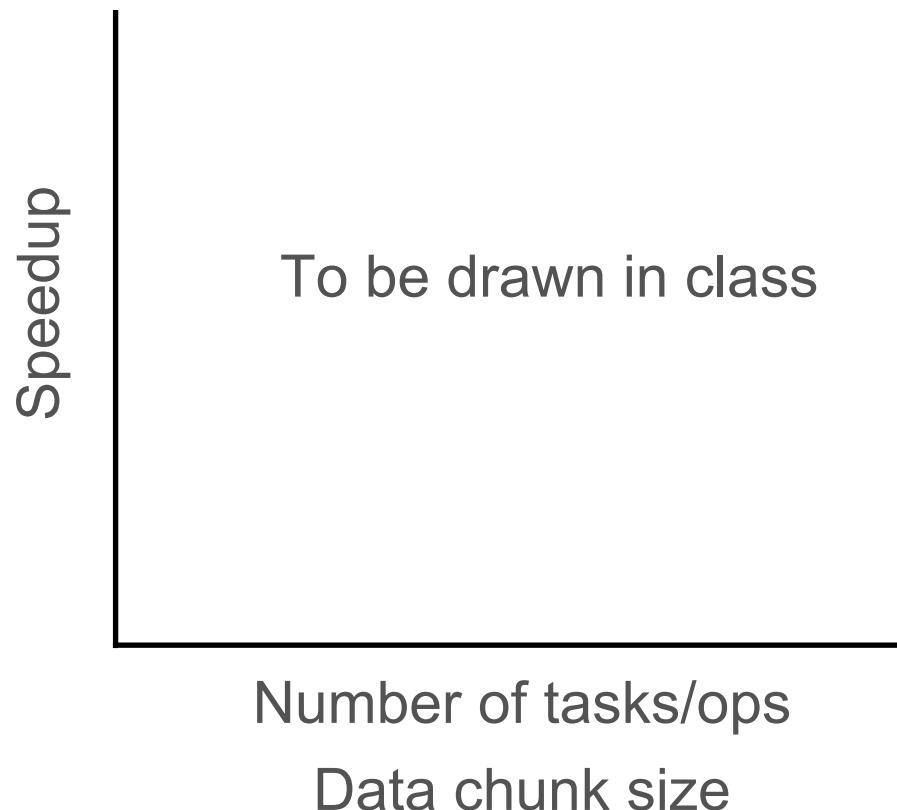
ALT

# Y'all are database programmers! :)



# Execution Optimization Tradeoffs

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

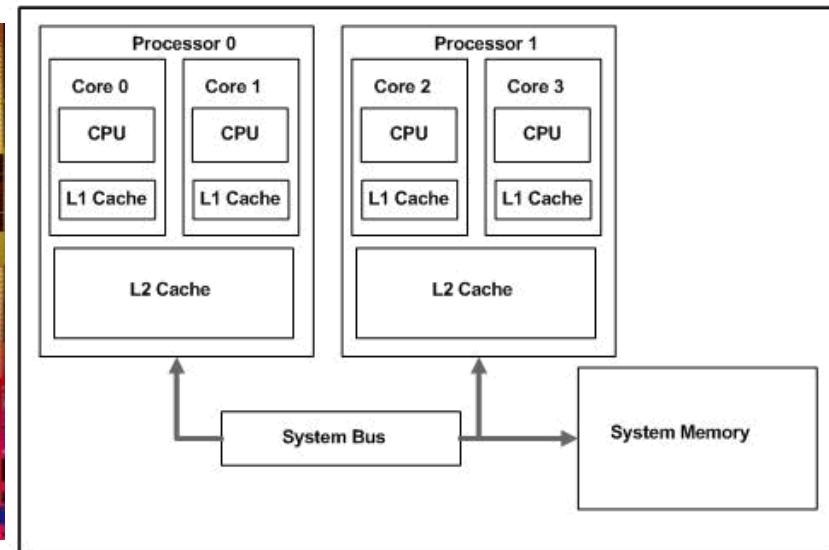
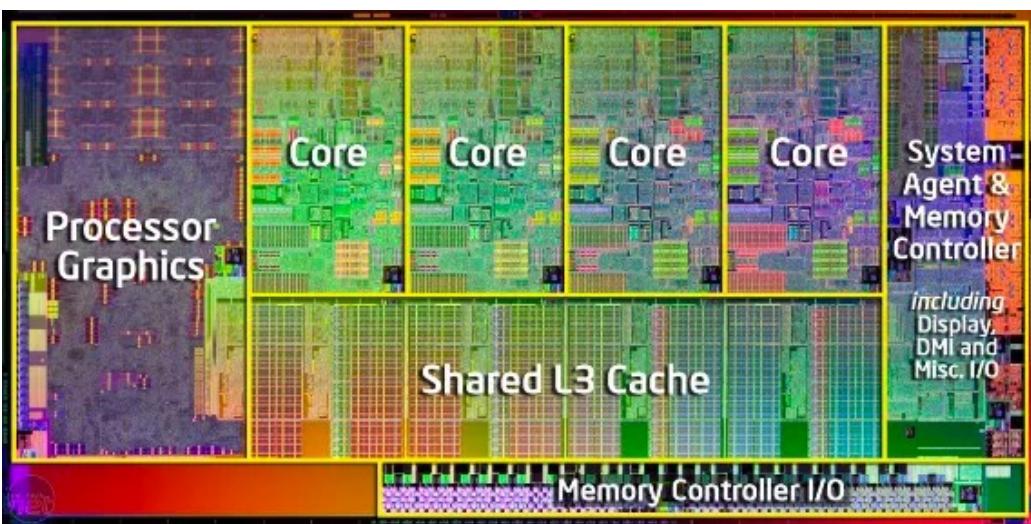


# Outline

- ❖ 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

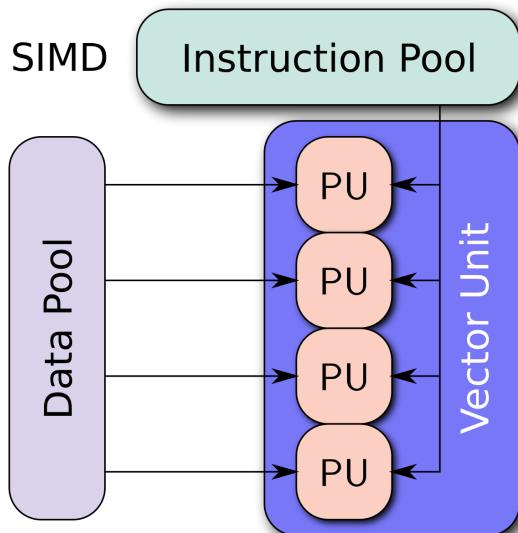
# Multi-core CPUs

- ❖ Modern computers often have multiple processors and multiple cores per processor, with a hierarchy of shared caches
- ❖ OS Scheduler controls what cores/processors assigned to what processes/threads when

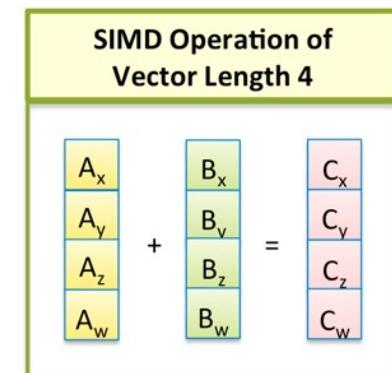
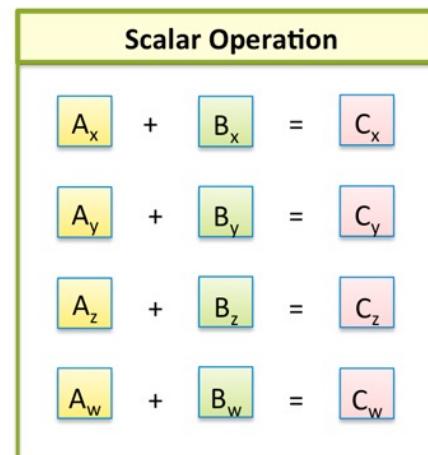


# Single-Instruction Multiple-Data

- ❖ **Single-Instruction Multiple-Data (SIMD):** A fundamental form of parallel data 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:

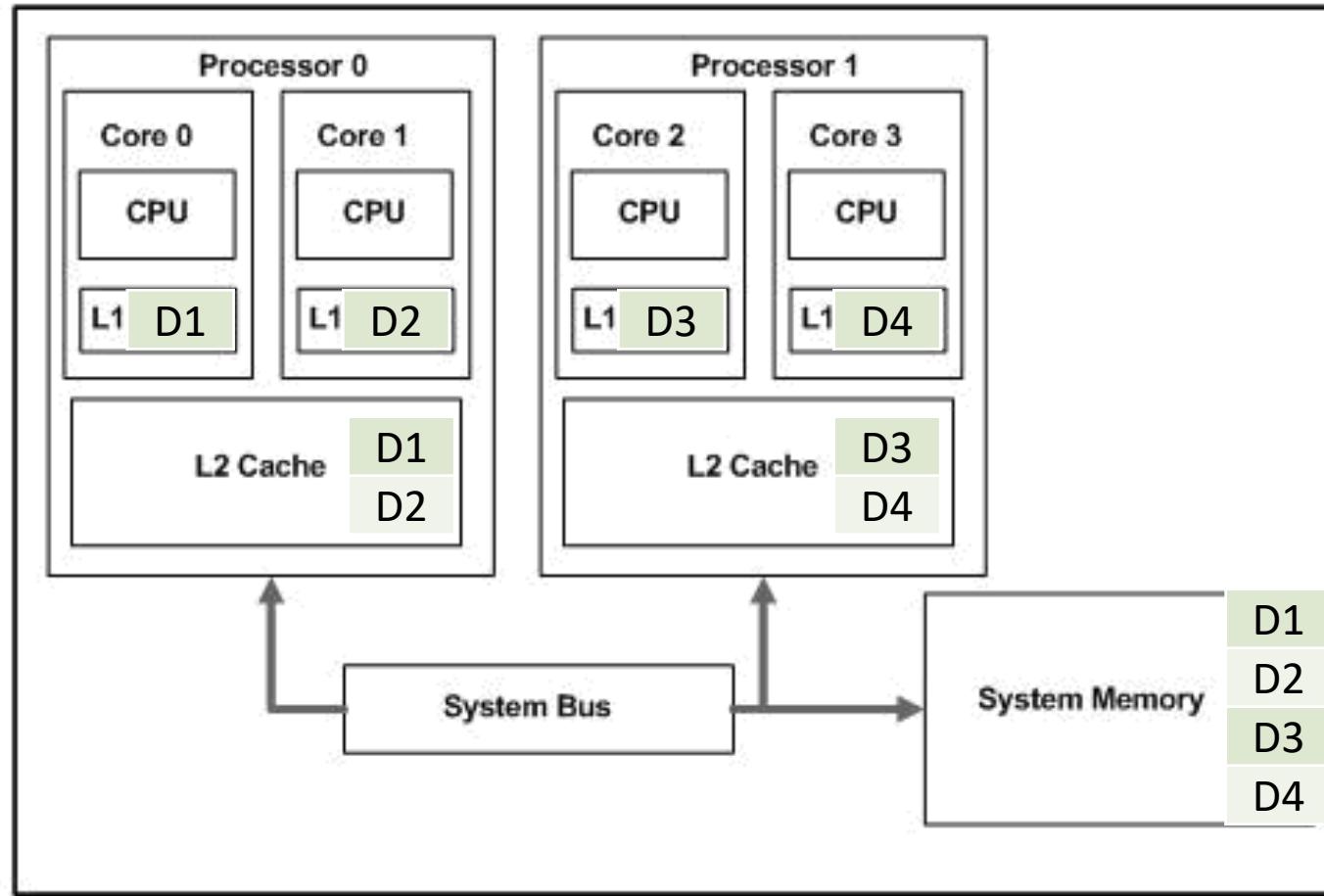


Intel® Architecture currently has SIMD operations of vector length 4, 8, 16

# 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 data processing at scale

# “Data Parallel” Multi-core Execution



# Quantifying Efficiency: Speedup

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

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

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

- ❖ In data science computations, a useful surrogate for completion time is the instruction throughput: **FLOPS**, 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:

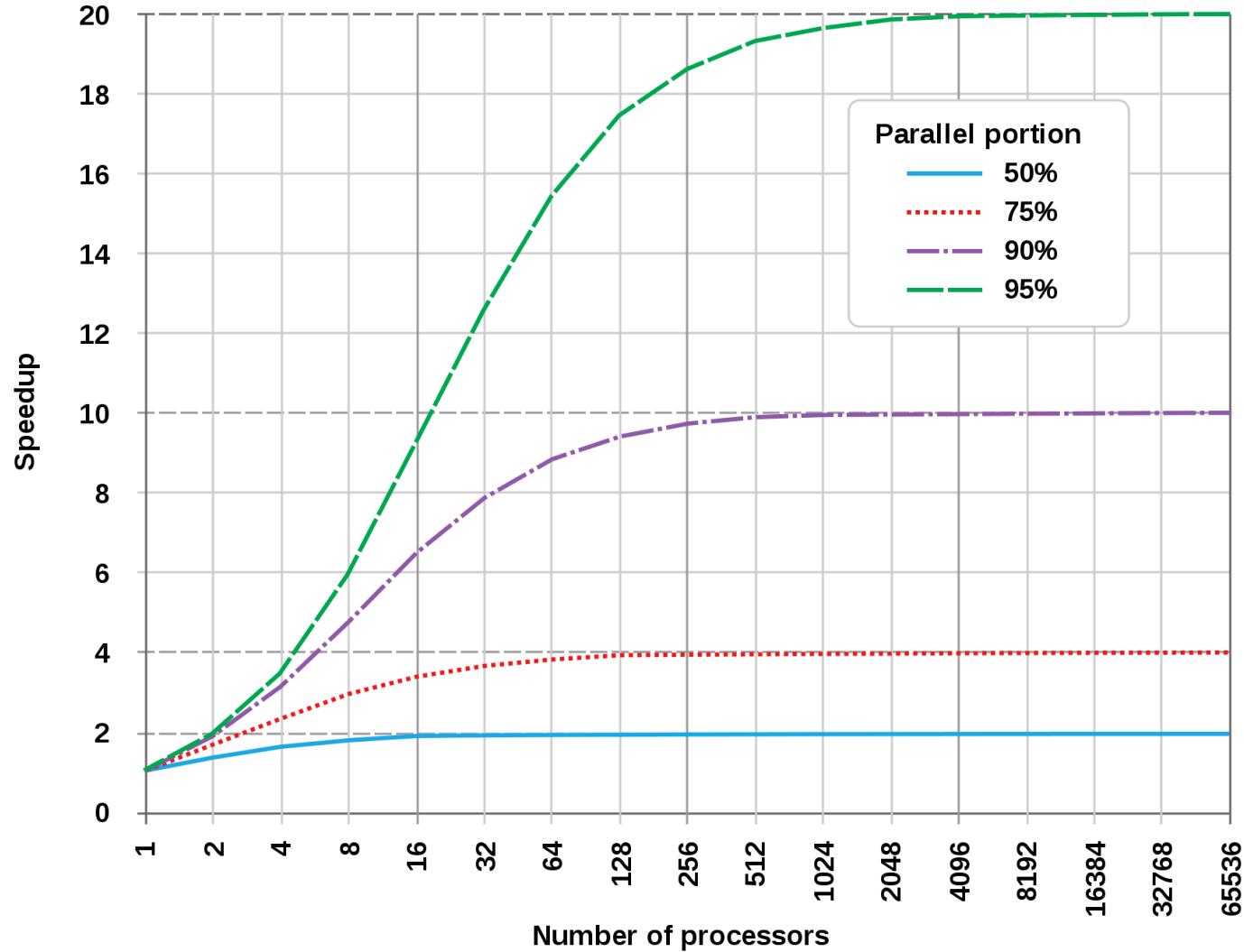
$$T_{\text{yes}} \longrightarrow T_{\text{yes}}/n$$

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

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

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

# Amdahl's Law



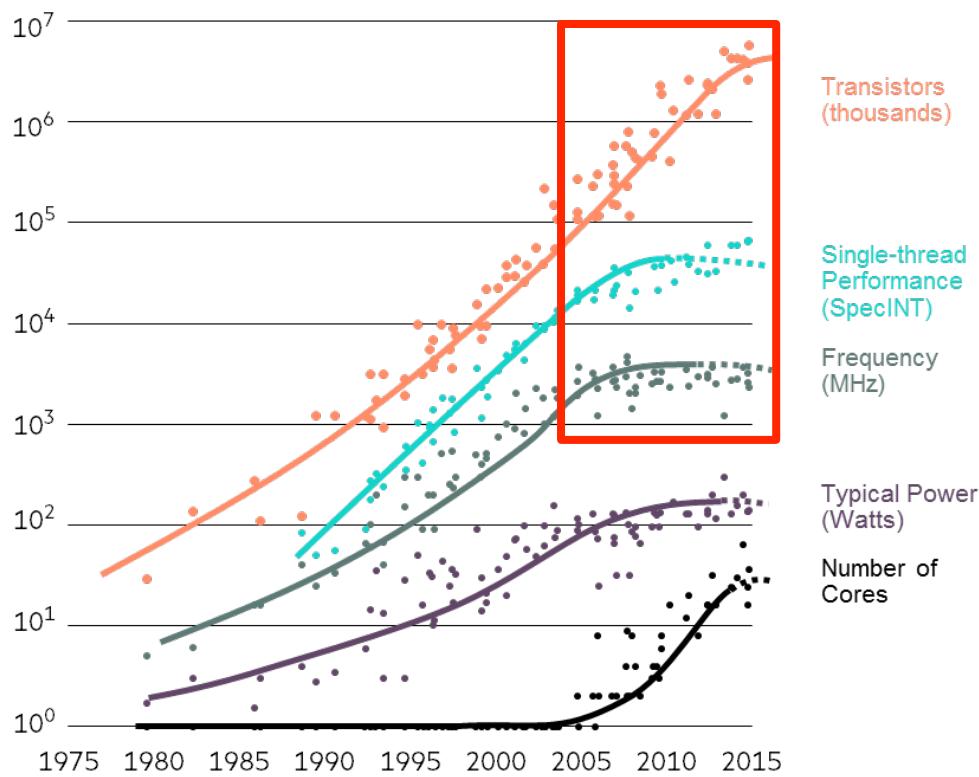
$$f = T_{yes}/T_{no}$$

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

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

# Hardware Trends on Parallelism

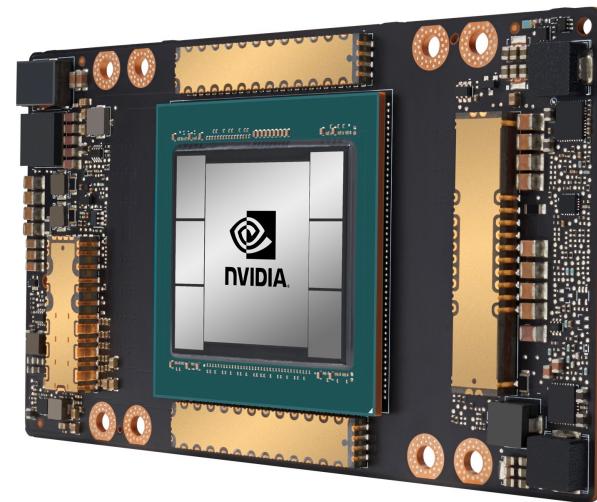
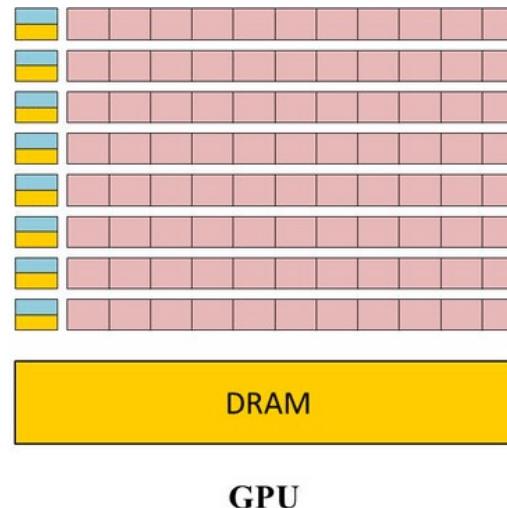
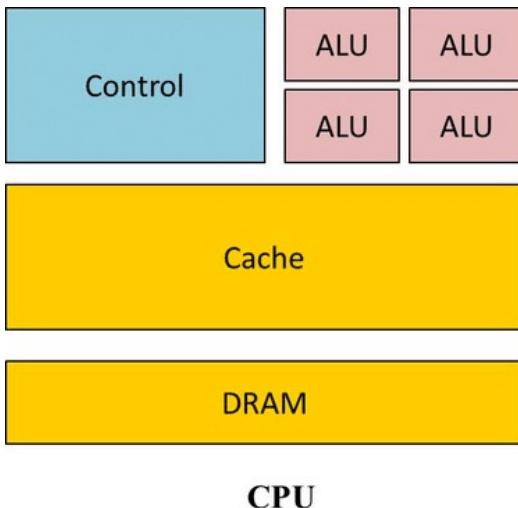
- ❖ Multi-core CPUs grew rapidly in early 2000s but hit physical limits due to packing efficiency and power issues
- ❖ End of “Moore’s Law” and End of “Dennard Scaling”



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

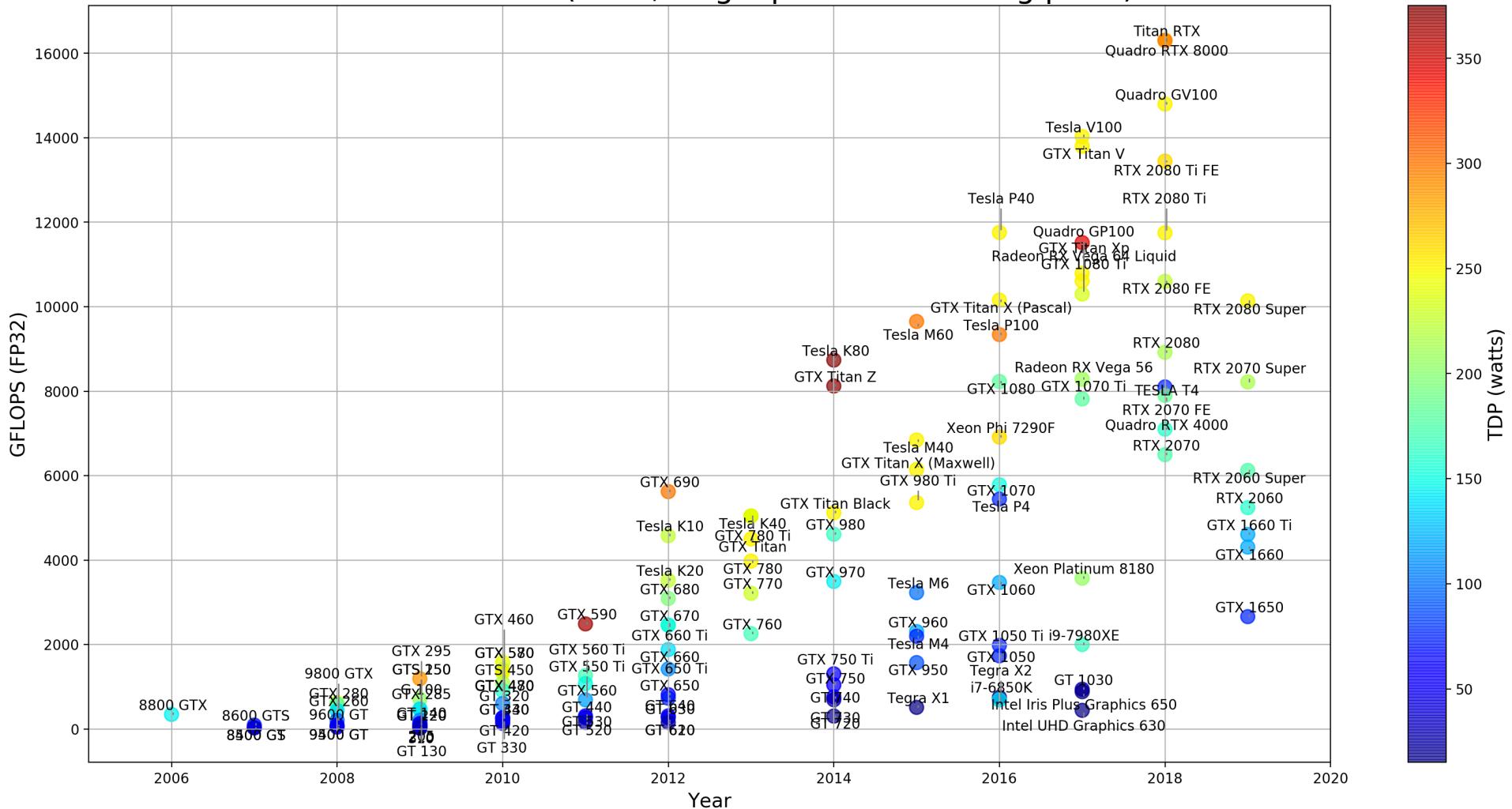
# Hardware Accelerators: GPUs

- ❖ **Graphics Processing Unit (GPU):** Tailored for matrix/tensor ops
- ❖ Basic idea: Use tons of ALUs; massive data parallelism (SIMD on steroids); now A100 offers ~20 TFLOPS for FP32!
- ❖ Popularized by NVIDIA in early 2000s for video games, graphics, and multimedia; now ubiquitous in DL
- ❖ CUDA released in 2007; later wrapper APIs on top: CuDNN, CuSparse, CuDF (RapidsAI), etc.



# GPUs on the Market

GPU Performance (FP32, single precision floating point)



# Other Hardware Accelerators

- ❖ **Tensor Processing Unit (TPU):**
  - ❖ An “application-specific integrated circuit” (ASIC) created by Google in mid 2010s; used for AlphaGo
  - ❖ Even more specialized tensor ops in DL inference; ~45 TFLOPS!
- ❖ **Field-Programmable Gate Array (FPGA):**
  - ❖ Configurable for *any* class of programs; ~0.5-3 TFLOPS but very low power consumption
  - ❖ Cheaper; new hardware-software integrated stacks for ML/DL



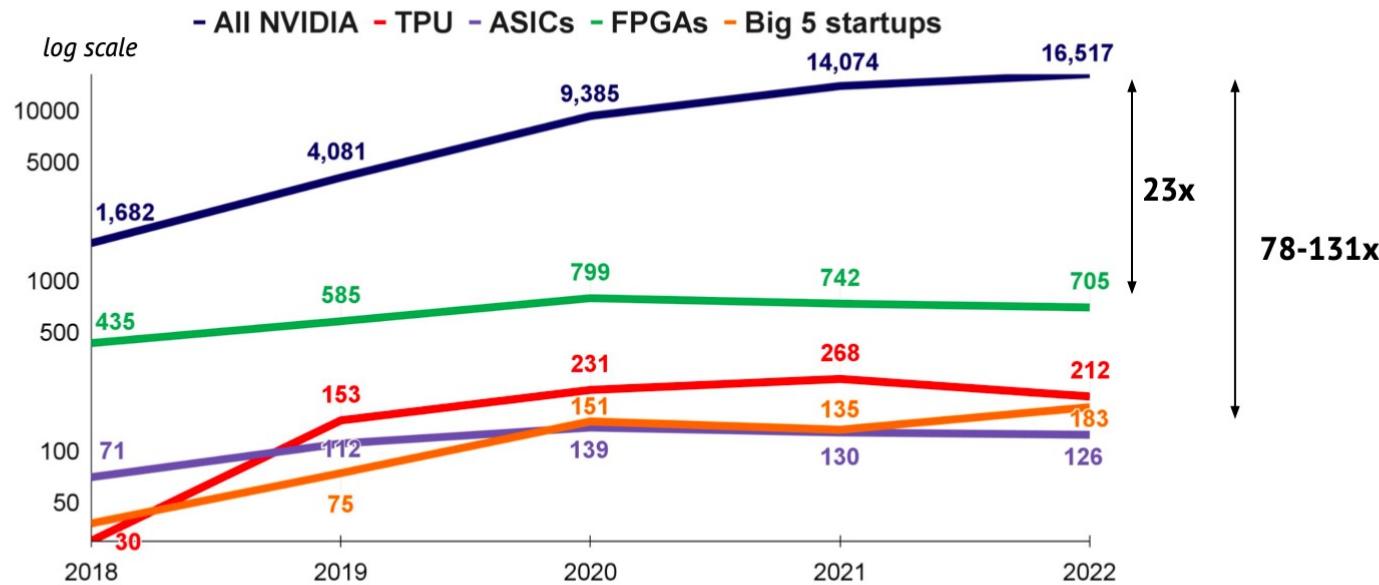
# Comparing Modern Parallel Hardware

	Multi-core CPU	GPU	FPGA	ASICs (e.g., TPUs)
Peak FLOPS	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	Poor Fit	Potential exists but not mass market
Fitness for DL Inference?	Moderate	Moderate	Good Fit	Best Fit
Cloud Vendor Support	All	All	All	GCP

# NVIDIA's monopoly remains strong!

**NVIDIA's chips are the most popular in AI research papers...and by a massive margin**

► GPUs are 131x more commonly used than ASICs, 90x more than chips from Graphcore, Habana, Cerebras, SambaNova and Cambricon combined, 78x more than Google's TPU, and 23x more than FPGAs.



# Review Questions

- ❖ Briefly explain 3 benefits of large-scale data in Data Science.
- ❖ What is a dataflow graph? Give an example from a data system.
- ❖ How does a task graph differ from a dataflow graph?
- ❖ Briefly explain 1 pro and 1 con of task parallelism.
- ❖ Briefly explain 1 scalability bottleneck that Dask still faces.
- ❖ What is the degree of parallelism of a task graph?
- ❖ What is speedup? How is it different from scaleup?
- ❖ Is linear speedup always possible with task parallelism?
- ❖ What is SIMD? Why is “vectorized” data processing critical?
- ❖ What is the point of Amdahl’s Law?
- ❖ Briefly 1 pro and 1 con of TPU vs. GPU.

# Peer Instruction Activity

(Switch slides)

# 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