UC San Diego

DSC 102
Systems for Scalable Analytics

Arun Kumar

Topic 3: Parallel and Scalable Data Processing

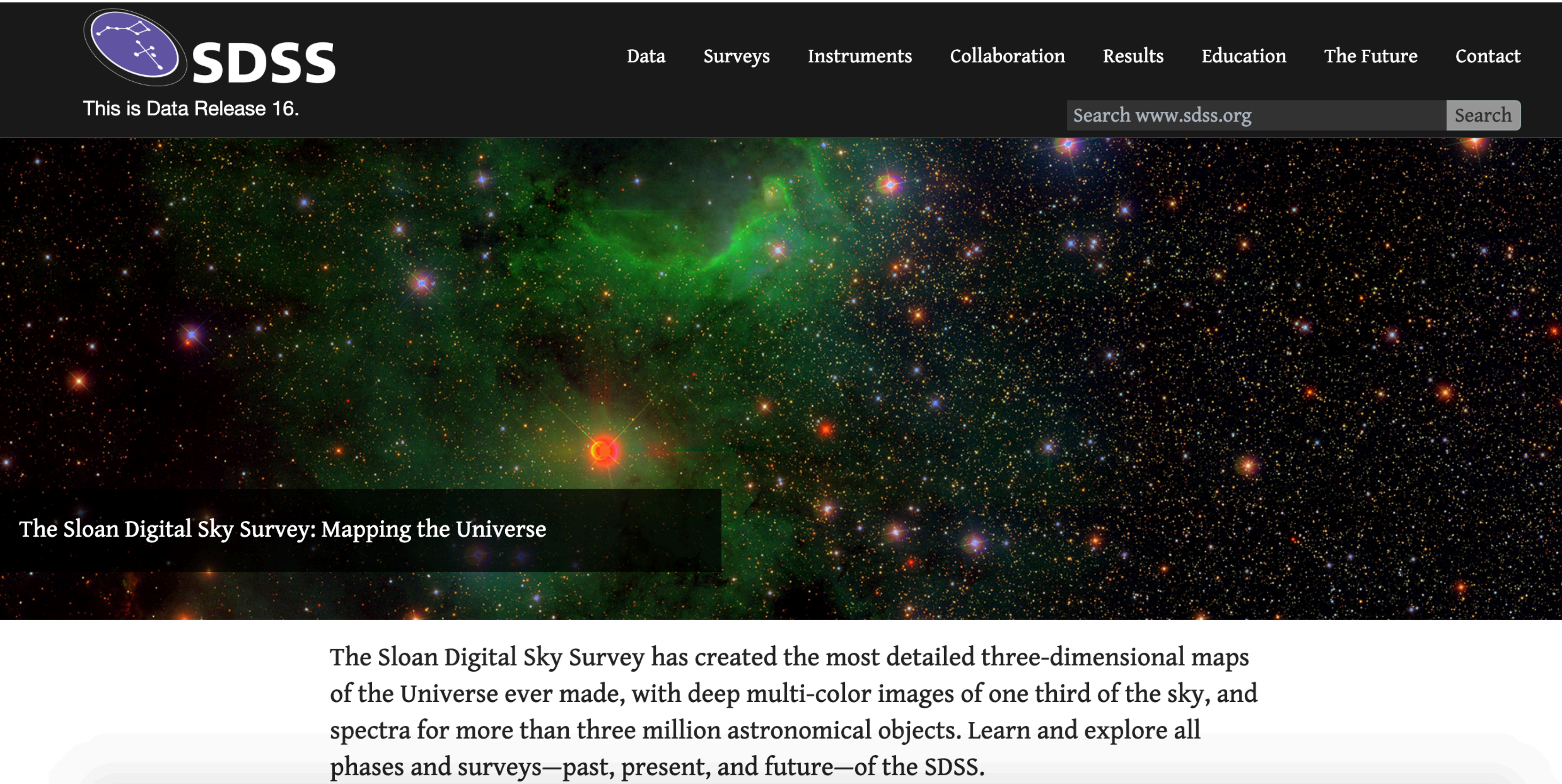
Part 1: Parallelism Basics

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

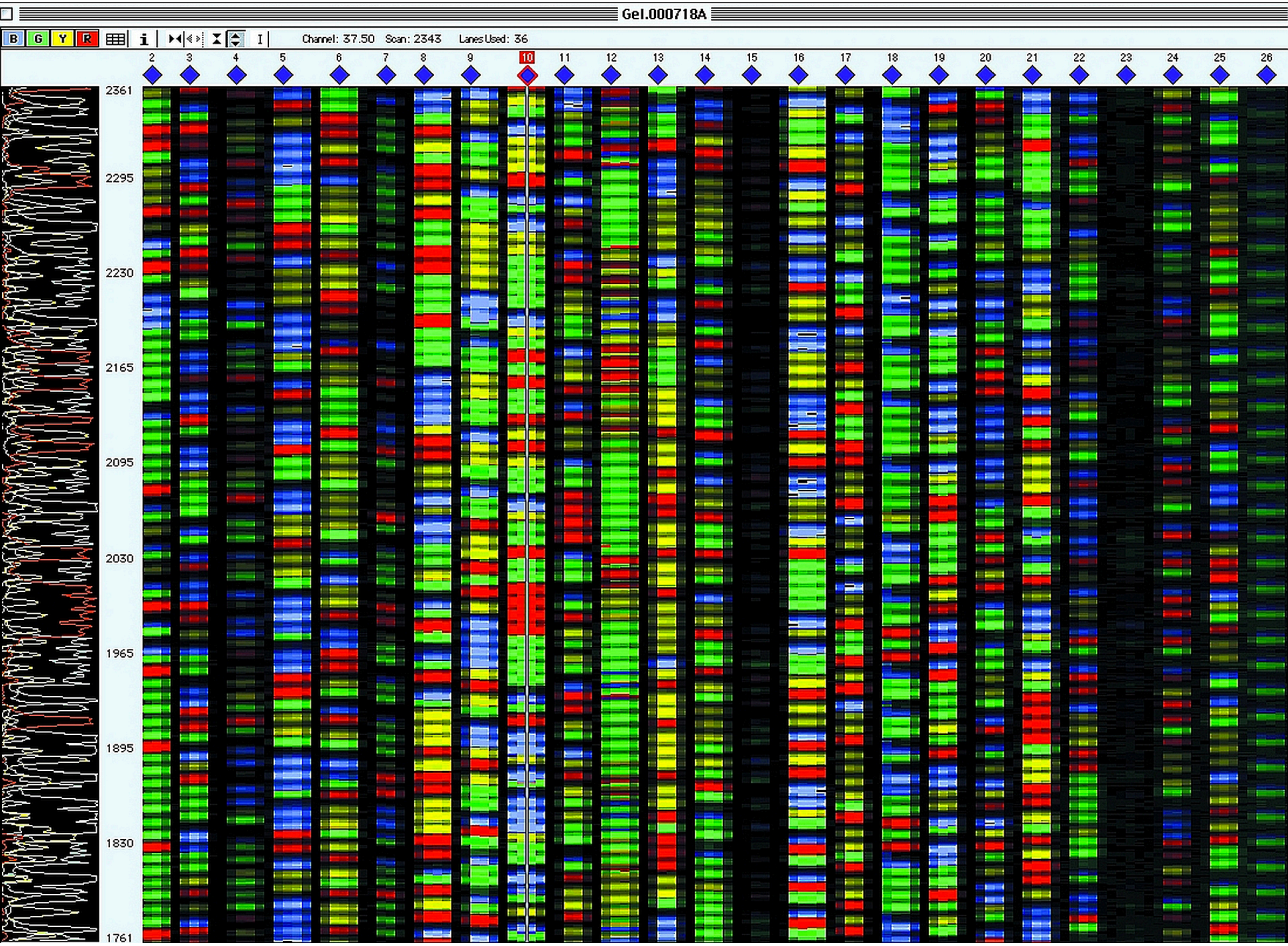
Q: *Why bother with large-scale data?*
Why does sampling not suffice?

Large-Scale Data in Astronomy

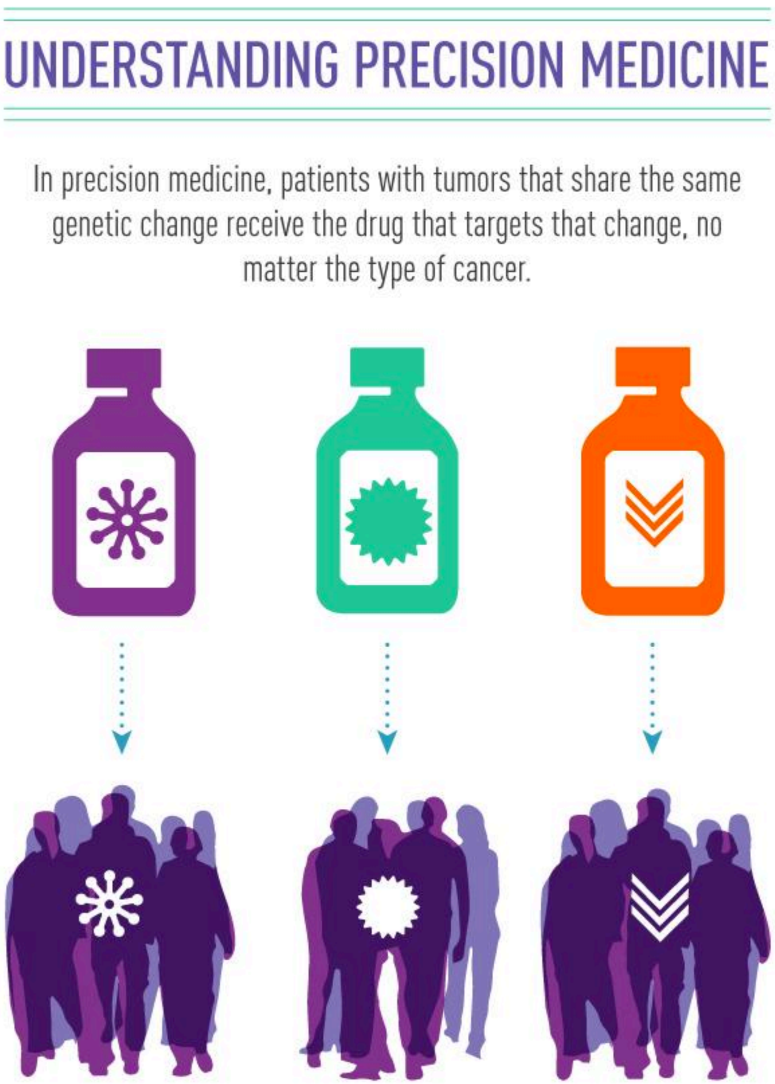


High-res. images: ~200 GB per day since 2000 (1PB+)

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

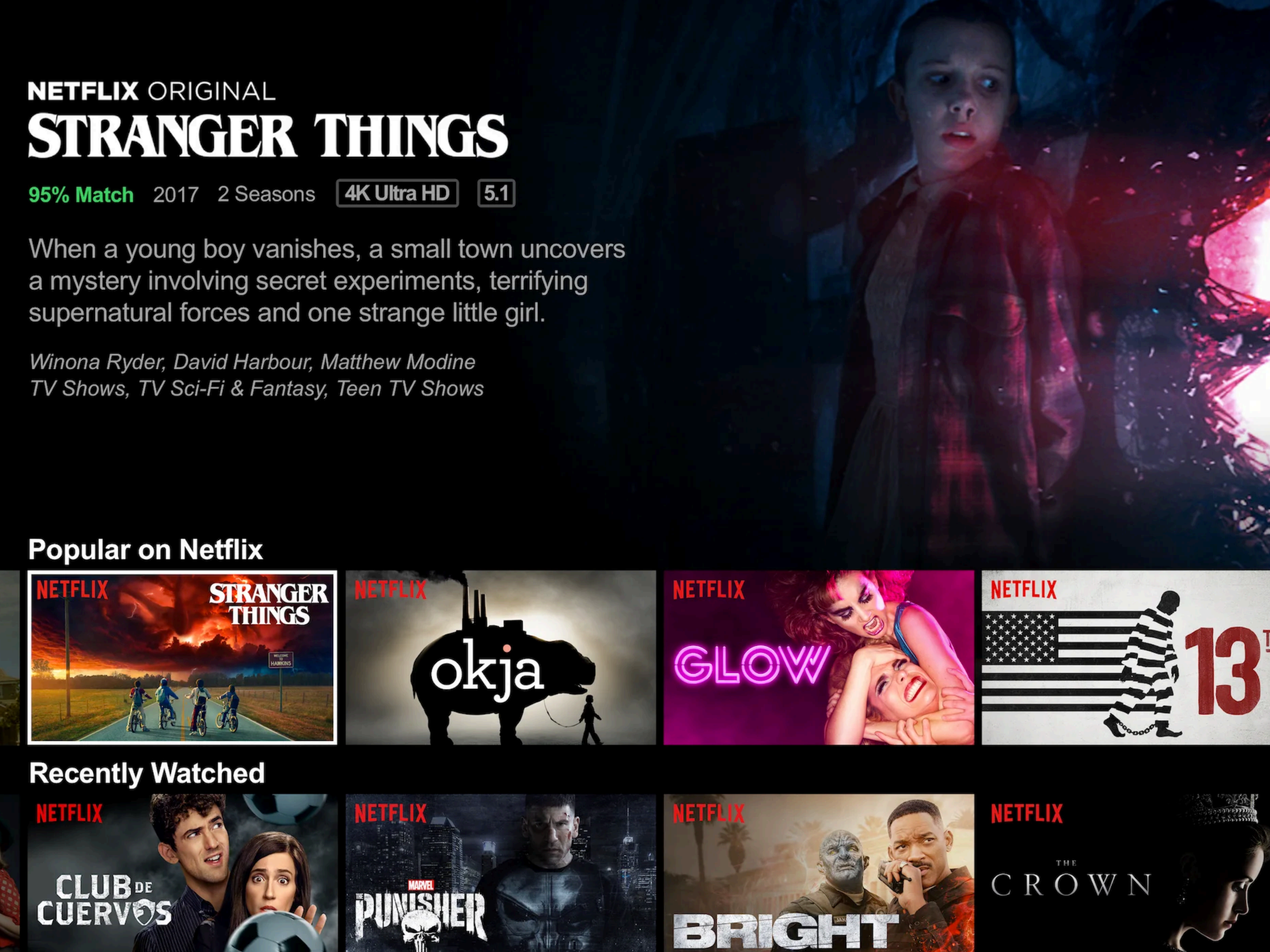
Large-Scale Data in Genomics



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
NETFLIX
STRANGER THINGS
NETFLIX
okja
NETFLIX
GLOW
NETFLIX
Recently Watched
NETFLIX
CLUB DE CUERVOS
NETFLIX
MARVEL
PUNISHER
NETFLIX
BRIGHT
NETFLIX
THE CROWN

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

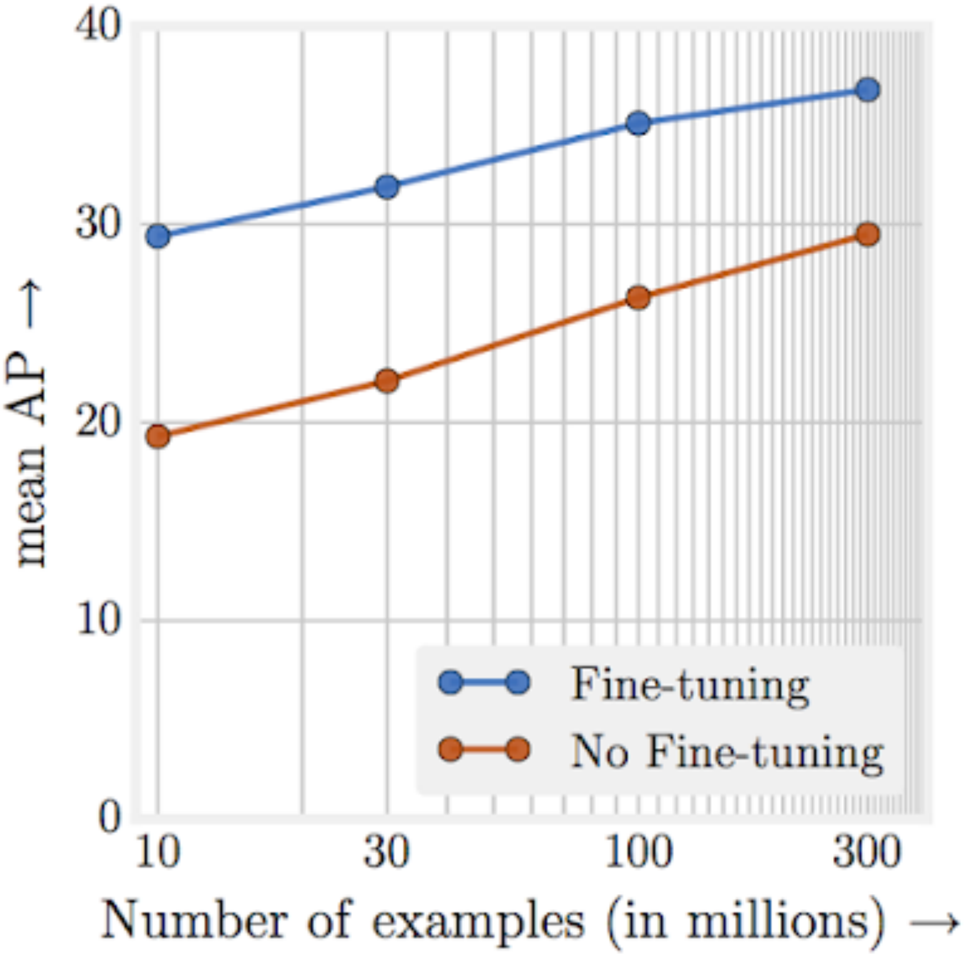
Large-Scale Data in Computer Vision

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

>500GB uncompressed as tensors

Harbinger of deep learning revolution

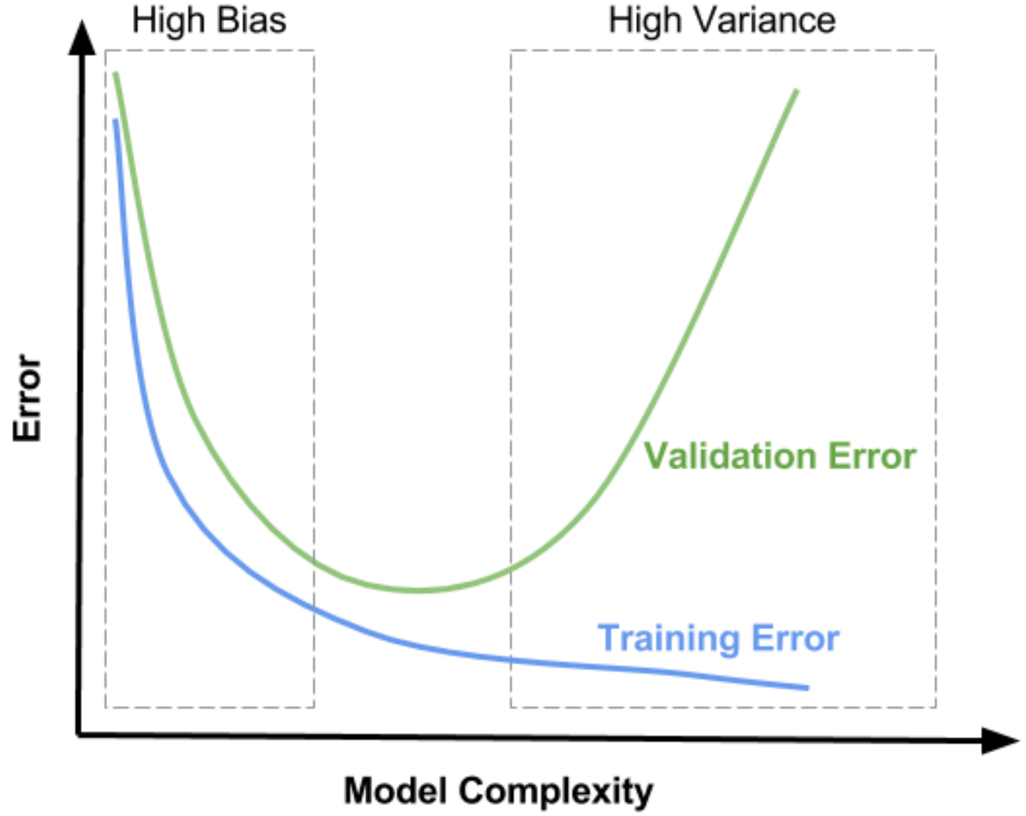
"The Unreasonable Effectiveness of Data"



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

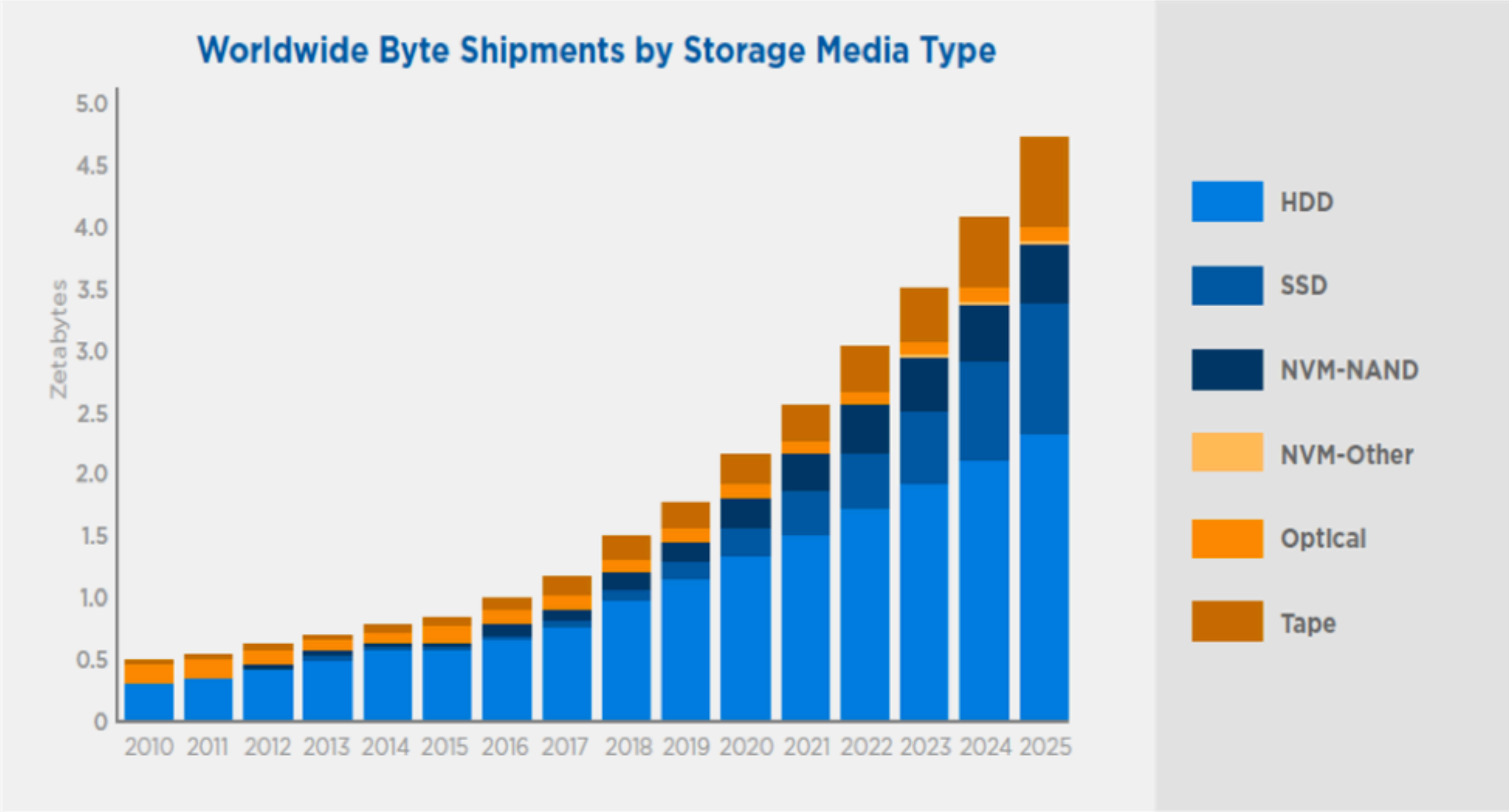
“Big Data”

- 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!

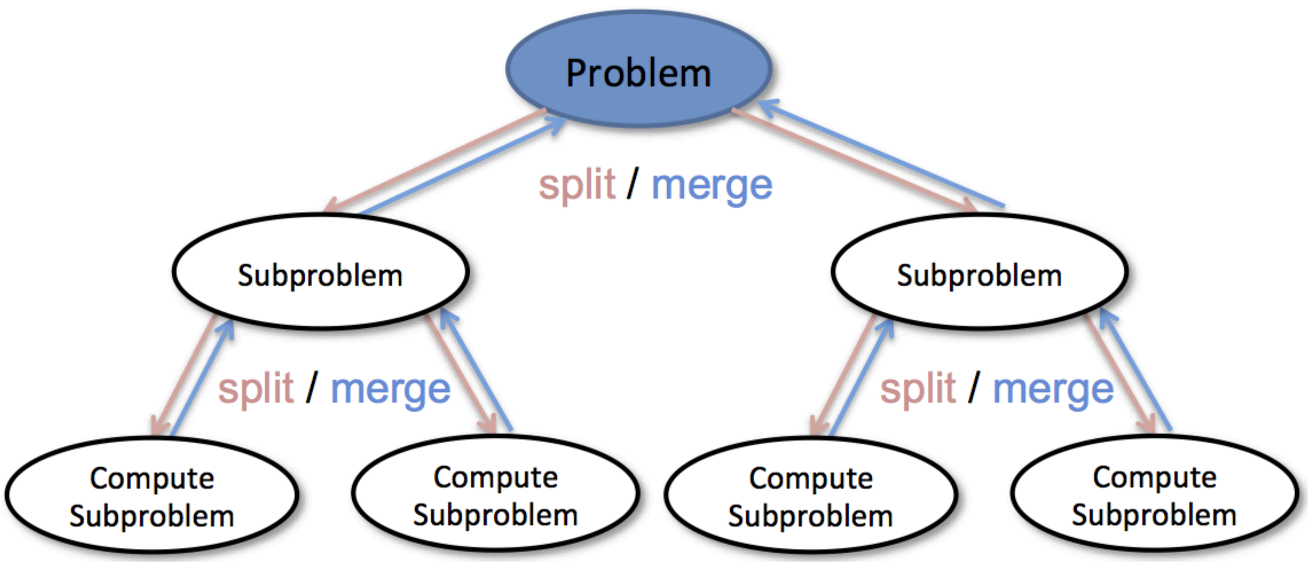
Outline

- 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”)

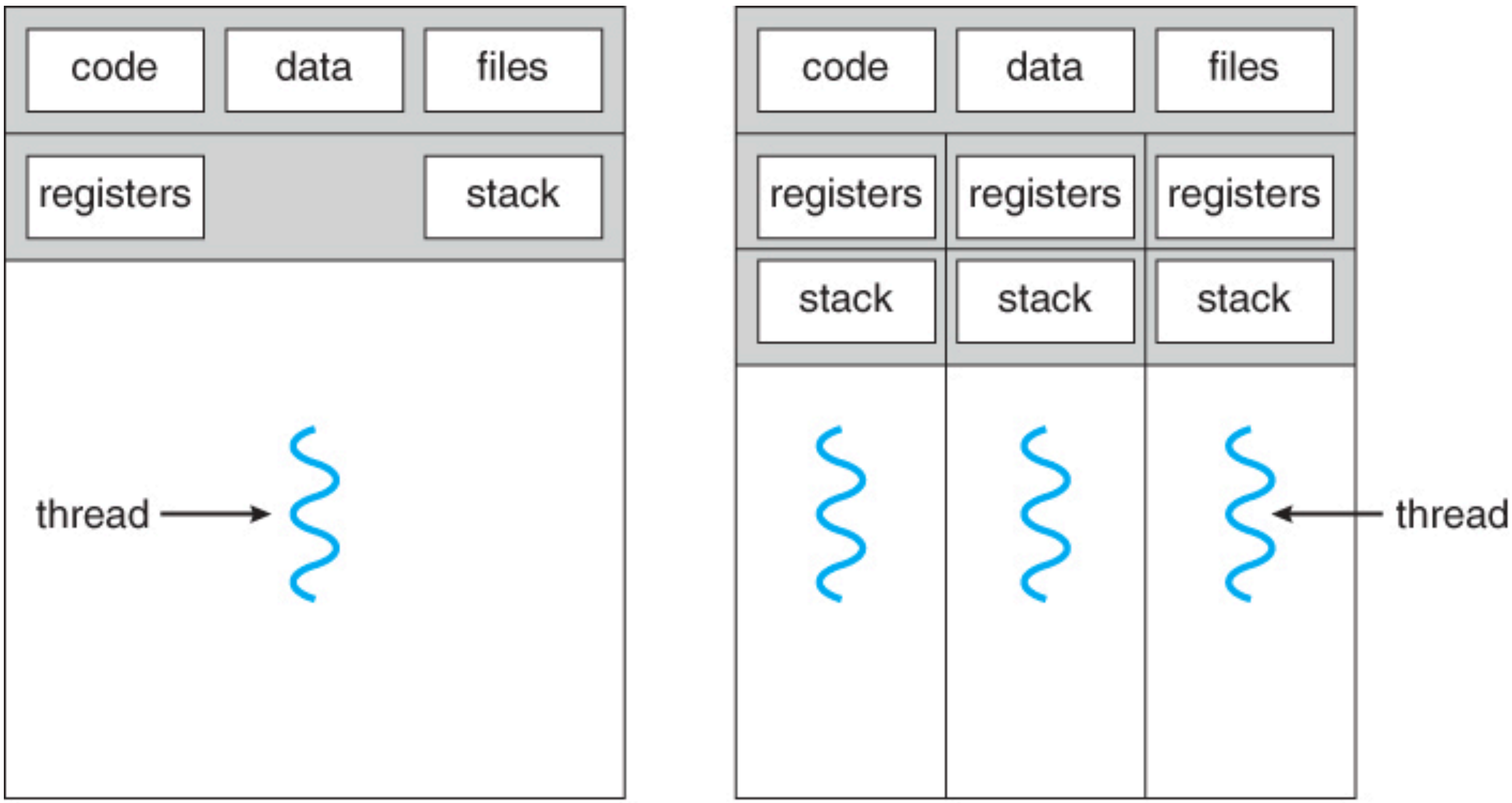


https://medium.com/cracking-the-data-science-interview/divide-and-conquer-algorithms-b135681d08fc

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



single-threaded process

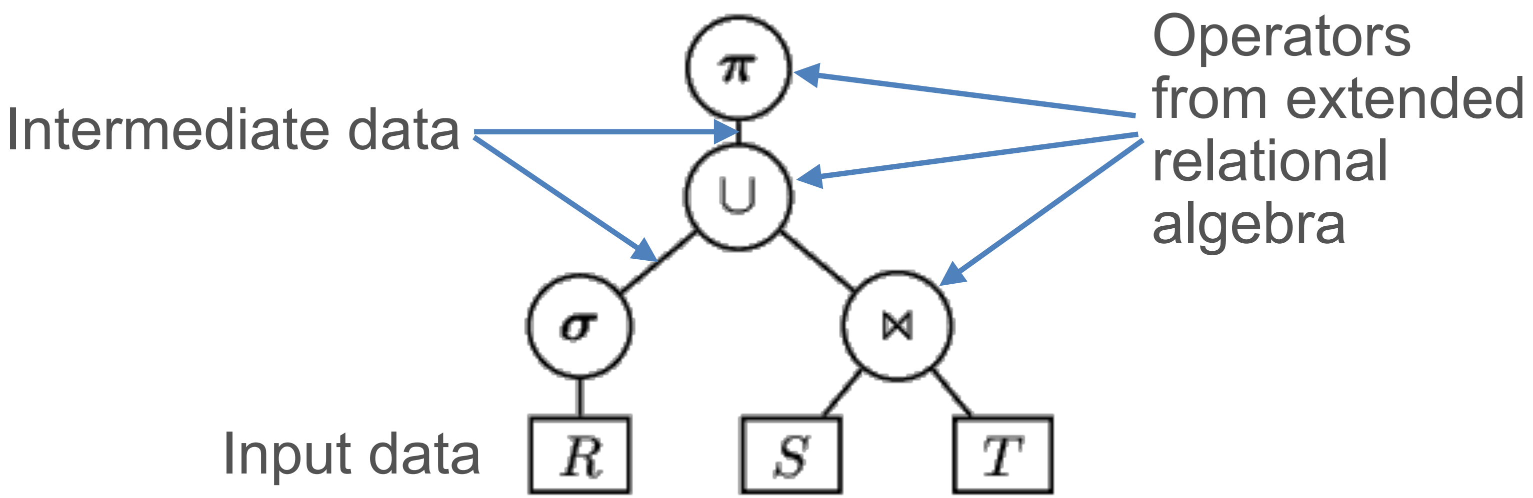
multithreaded process

New Parallelism Concept: Dataflow

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

Example Relational Dataflow Graph

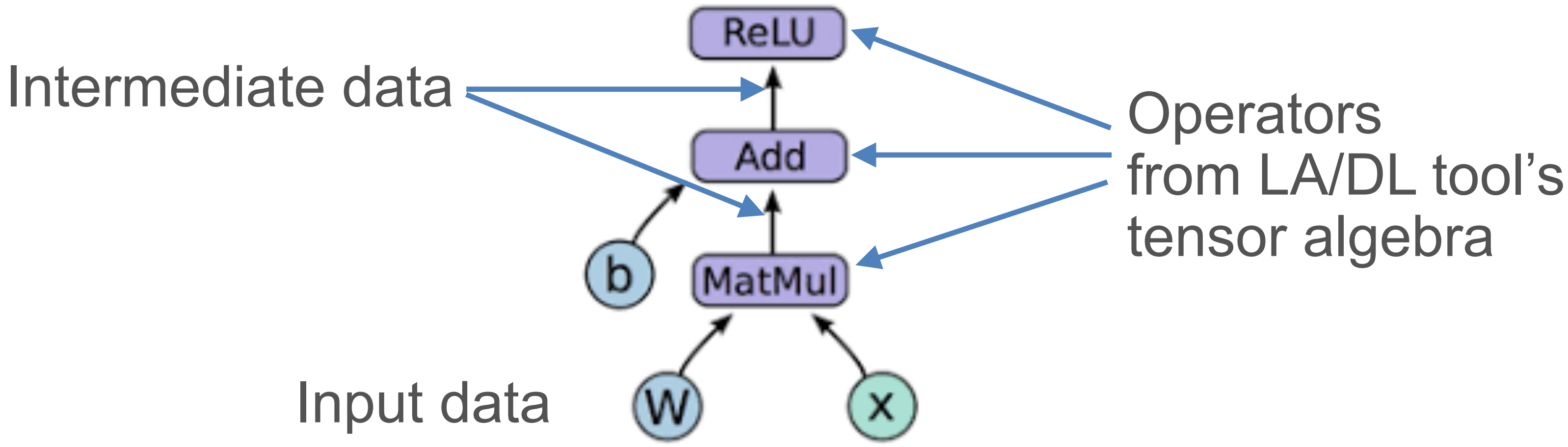
$$\pi(\sigma(R) \cup S \bowtie T)$$



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

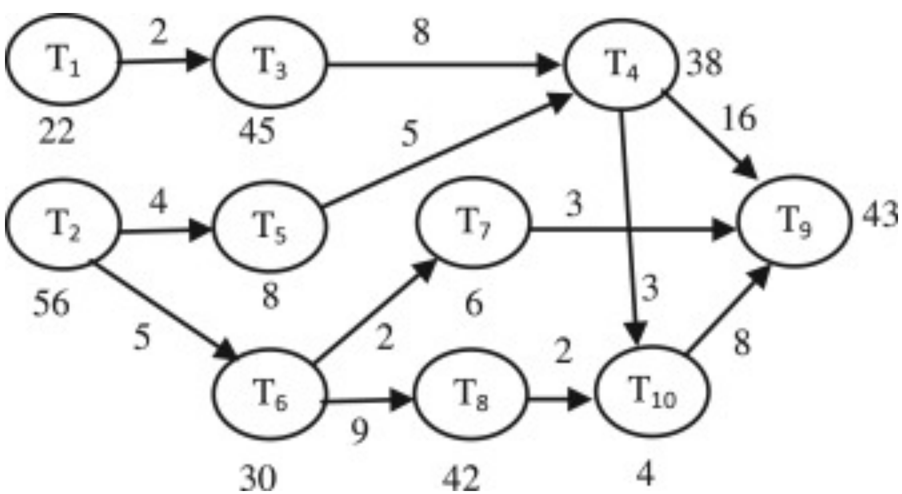
Example Tensor Dataflow Graph

$$ReLU(WX + b)$$



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

Example Task Graph



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

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

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

Parallel Data Processing

Central Issue: Workload takes too long for one processor!

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

Key parallelism paradigms in data systems:

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

Outline

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

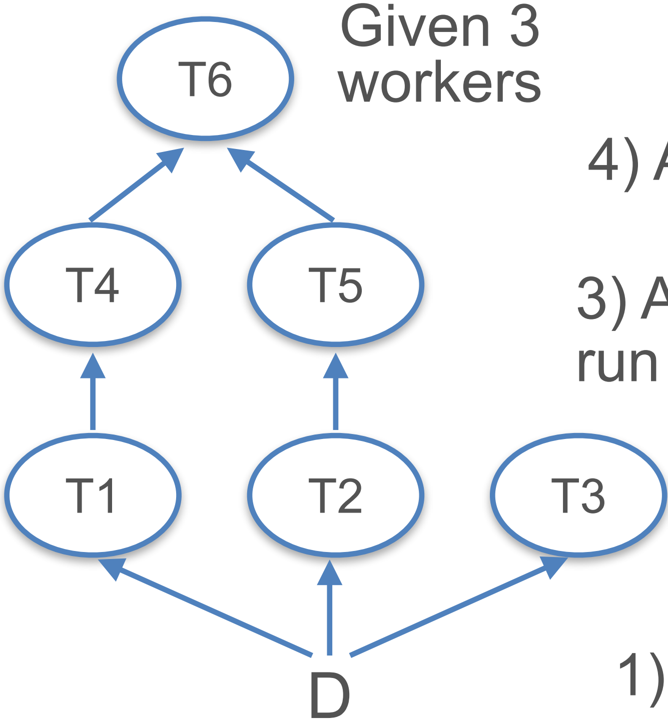
Task Parallelism

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

Example:

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

Given 3 workers



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

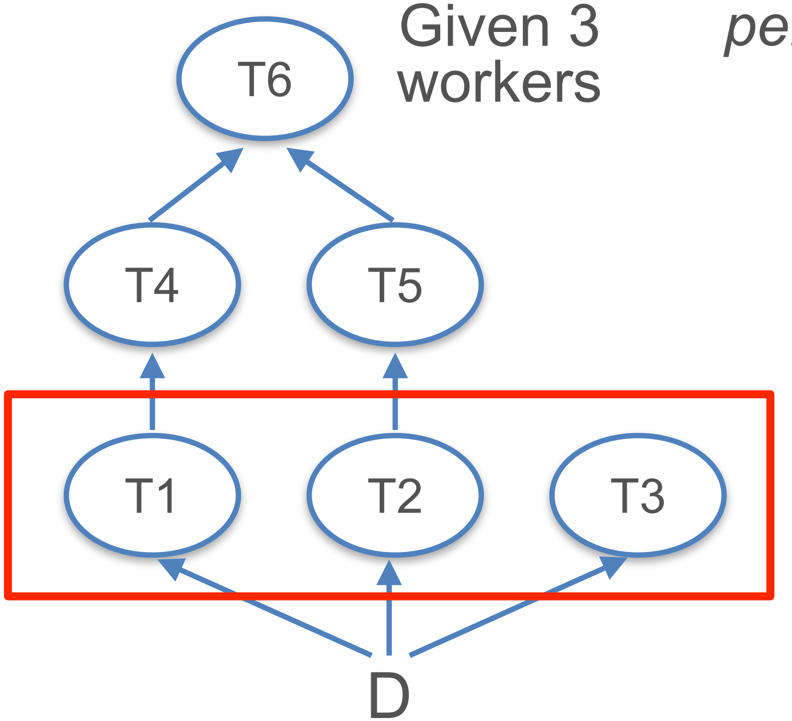
- **Topological sort** of tasks in task graph for scheduling
- Notion of a "worker" can be at processor/core level, not just at node/server level
  - *Thread-level* parallelism possible instead of process-level
  - E.g., Dask: 4 worker nodes x 4 cores = 16 workers total
- **Main pros** of task parallelism: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

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

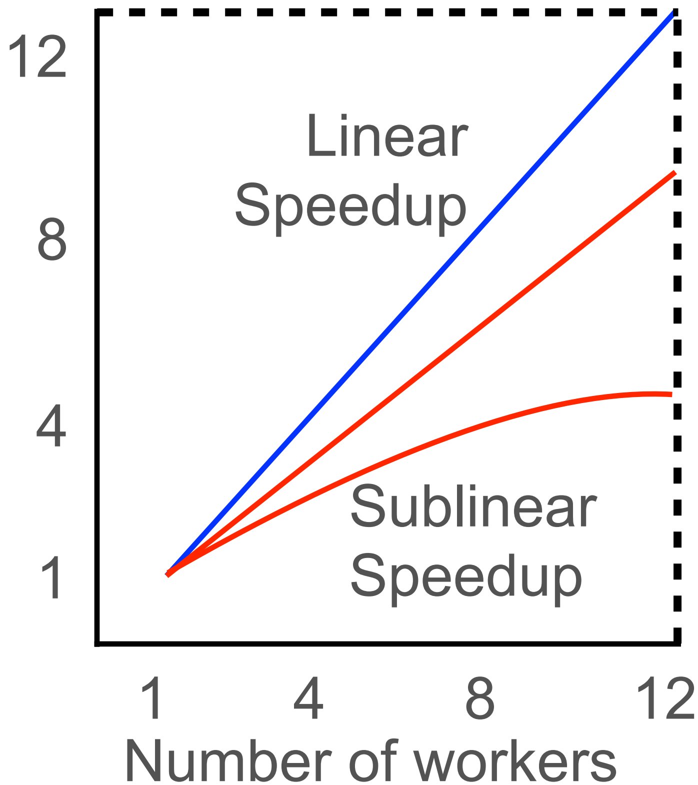
Q: *But given n workers, can we get a speedup of n?*

It depends!

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

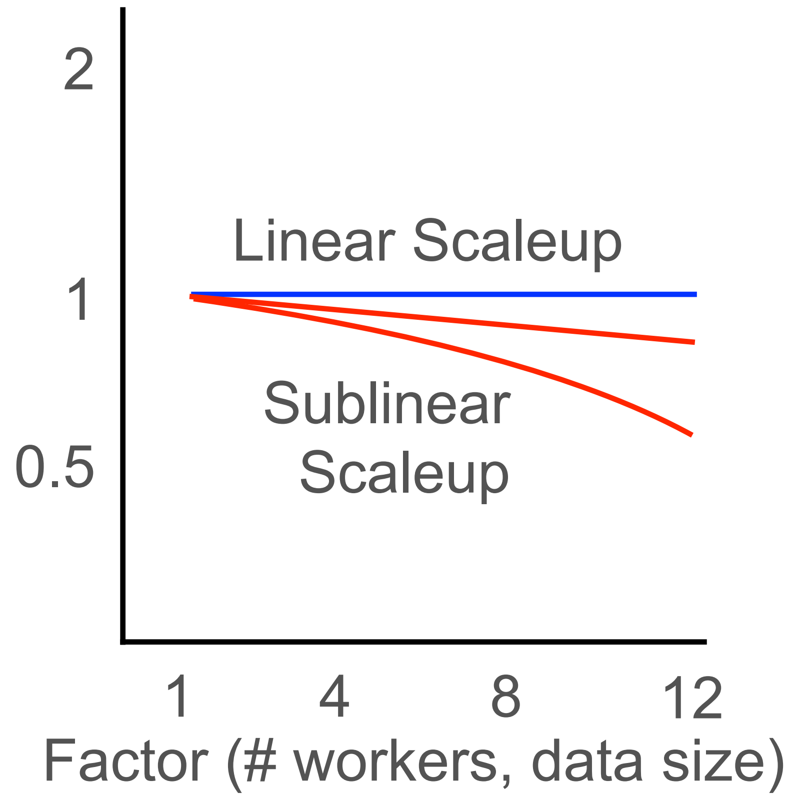
Quantifying Benefit of Parallelism

Runtime speedup (fixed data size)



Speedup plot / Strong scaling

Runtime speedup



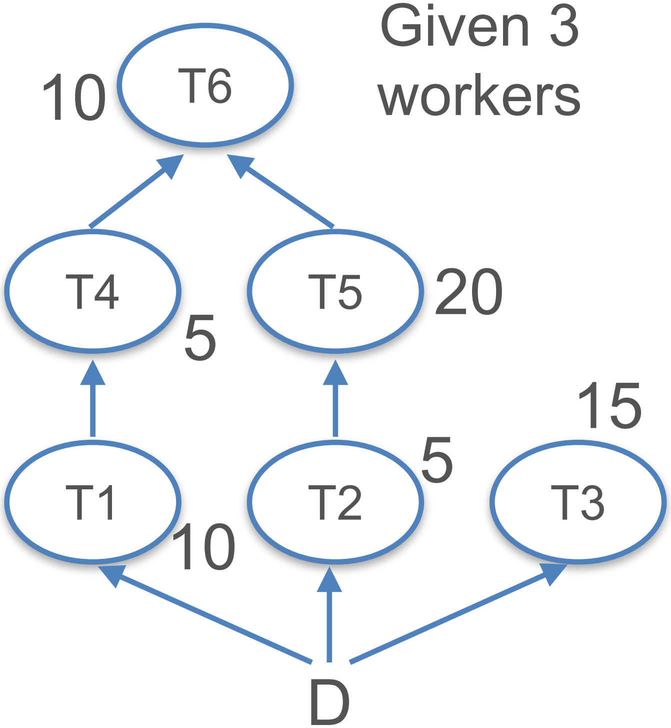
Scaleup plot / Weak scaling

Q: *Is <u>superlinear</u> speedup/scaleup ever possible?*

Idle Times in Task Parallelism

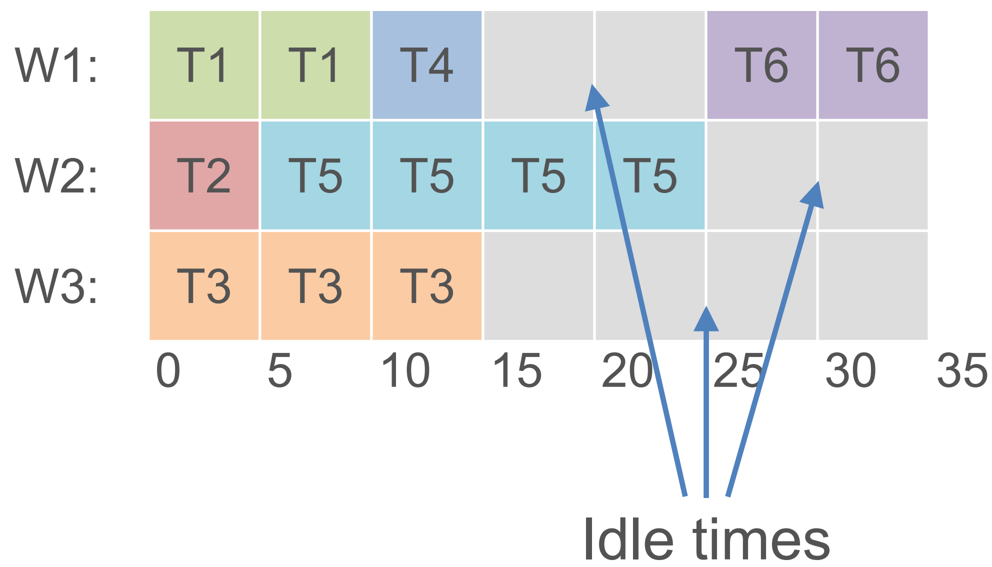
- Due to varying task completion times and varying degrees of parallelism in workload, idle workers waste resources

Example:



Given 3 workers

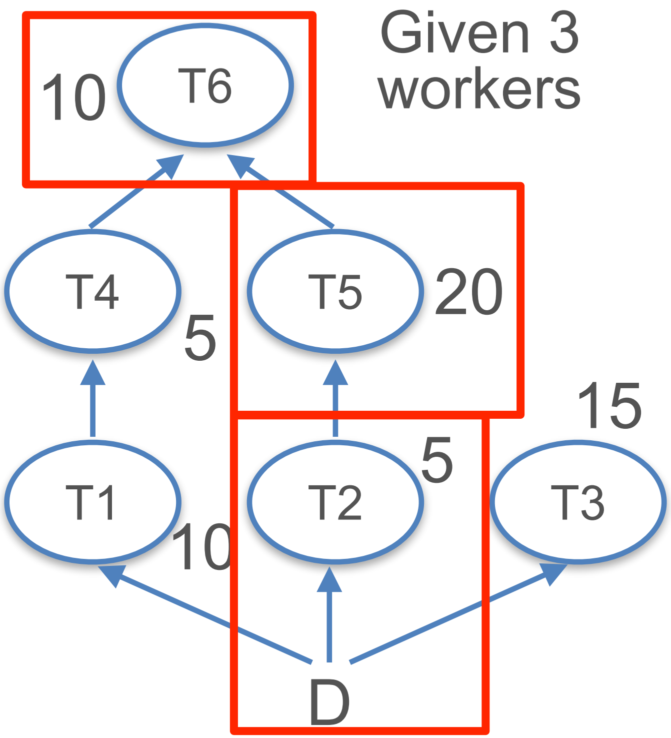
Gantt Chart visualization of schedule:



Idle Times in Task Parallelism

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

Example:

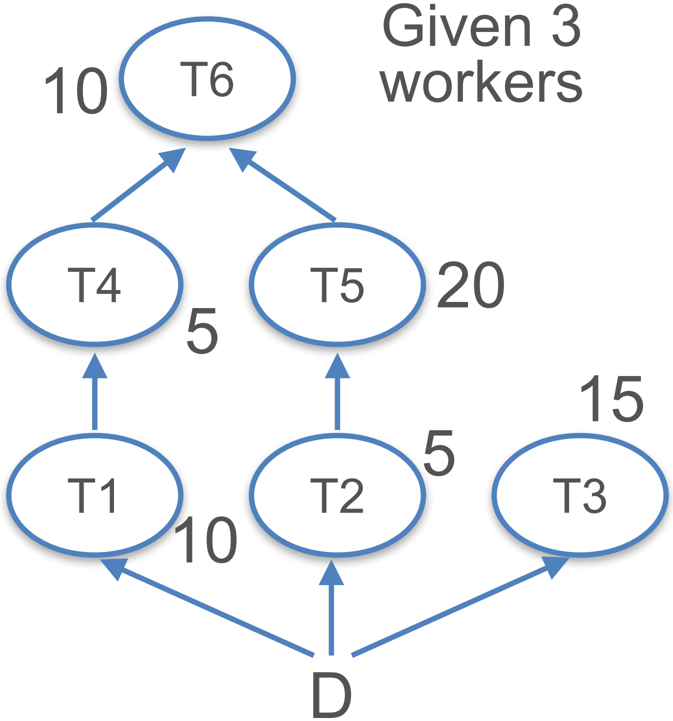


- In general, overall workload's completion time on task-parallel setup is always *lower bounded* by the **longest path** in the task graph
- Possibility: A task-parallel scheduler can "release" a worker if it knows that will be idle till the end
  - Can saves costs in cloud

Calculating Task Parallelism Speedup

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

Example:



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

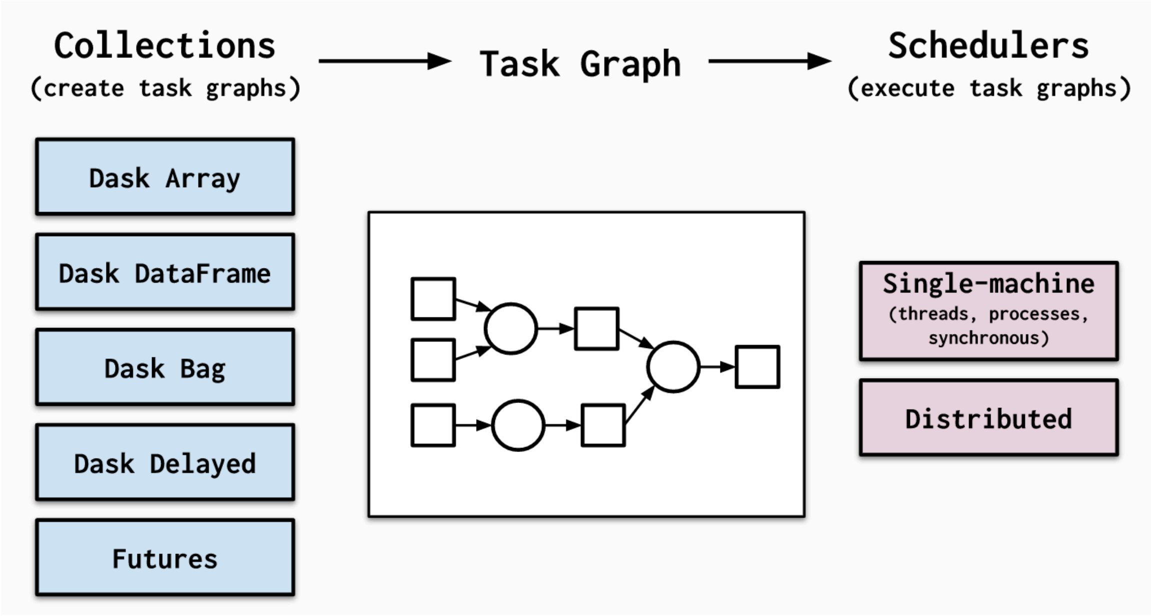
***Q:** Why is it only 1.9x?*

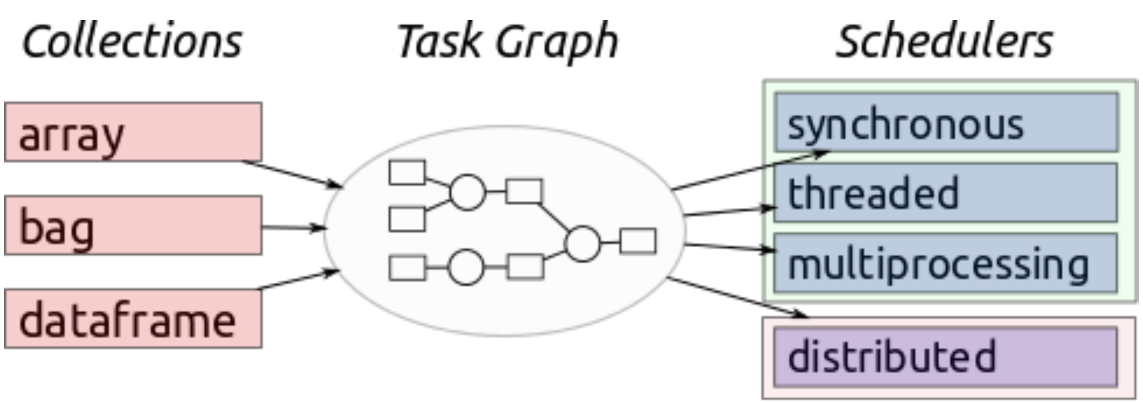
Task Parallelism in Dask

- *“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

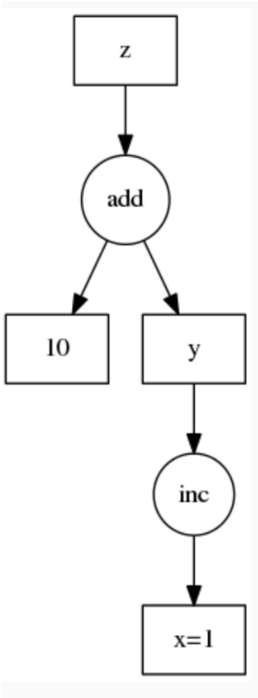
- **"Lazy Evaluation":**
  - Ops on data struct. are NOT executed immediately
  - Triggered manually, e.g., *compute()*
  - Dataflow graph / task graph is built under the hood

```python
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

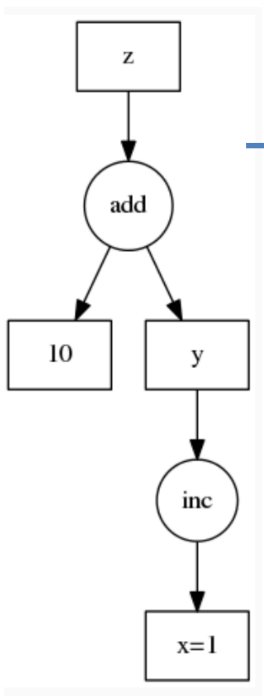
```python
d = {'x': 1,
     'y': (inc, 'x'),
     'z': (add, 'y', 10)}
```

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

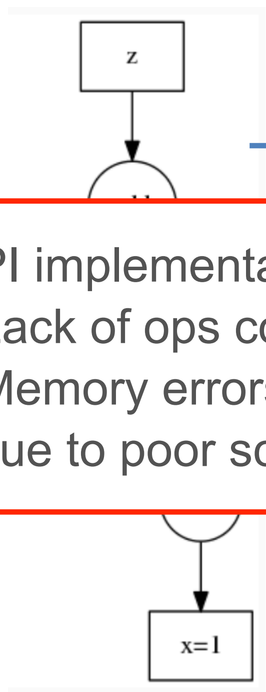
Scheduler divvies up work for cores/nodes

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 (dictionary)

C

to Scheduler

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

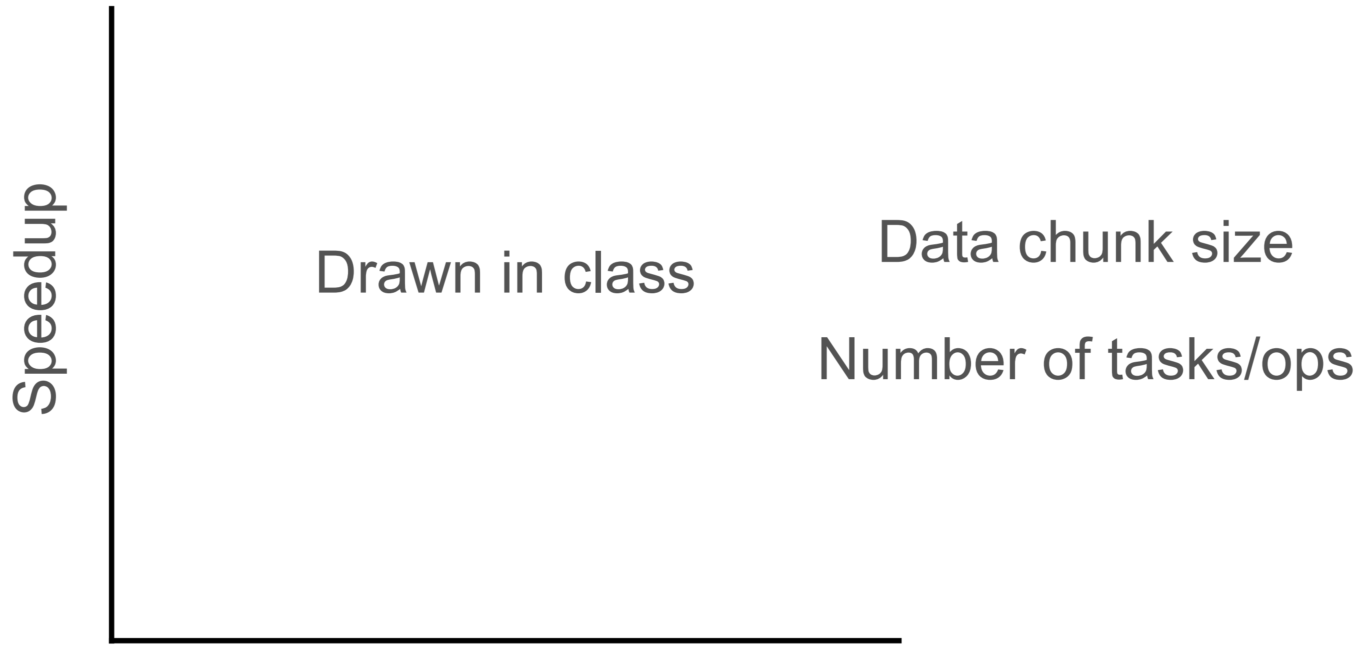
Best Practices for Task-Par. Dask

- **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

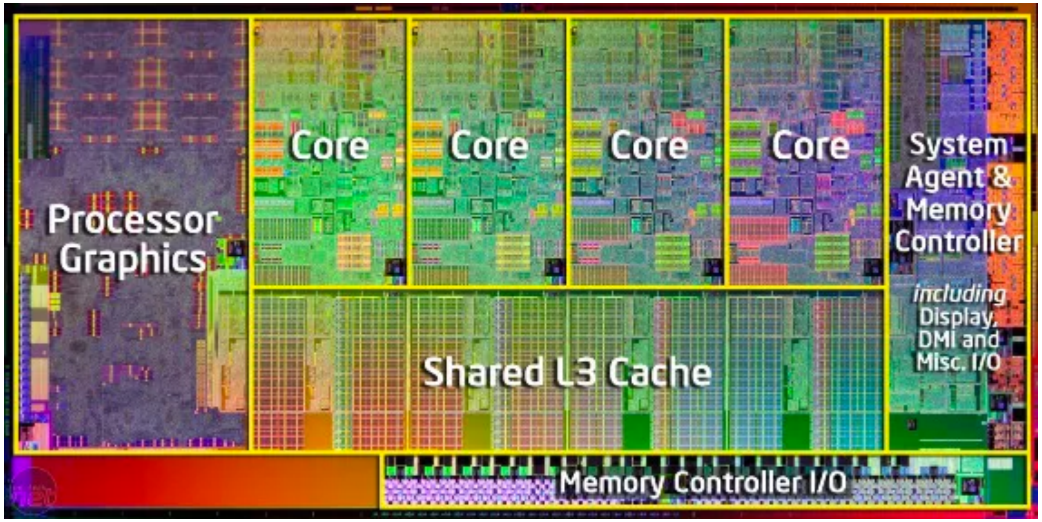
10:46 AM · Feb 2, 2021 · Twitter Web App

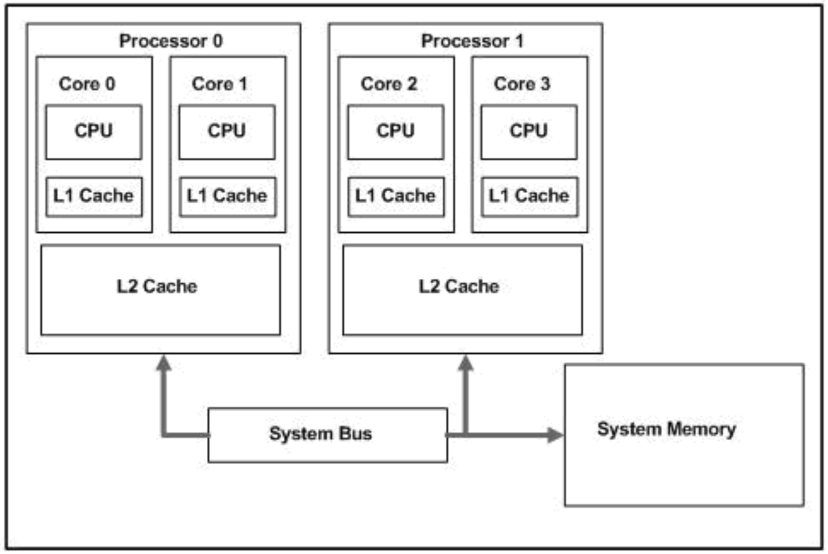
Outline

- 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

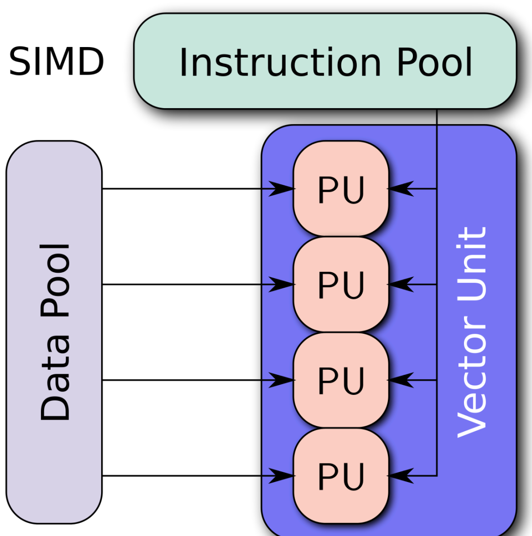
- Modern machines often have multiple processors and multiple *cores* per processor; hierarchy of shared caches
- OS Scheduler now controls what cores/processors assigned to what processes/threads when



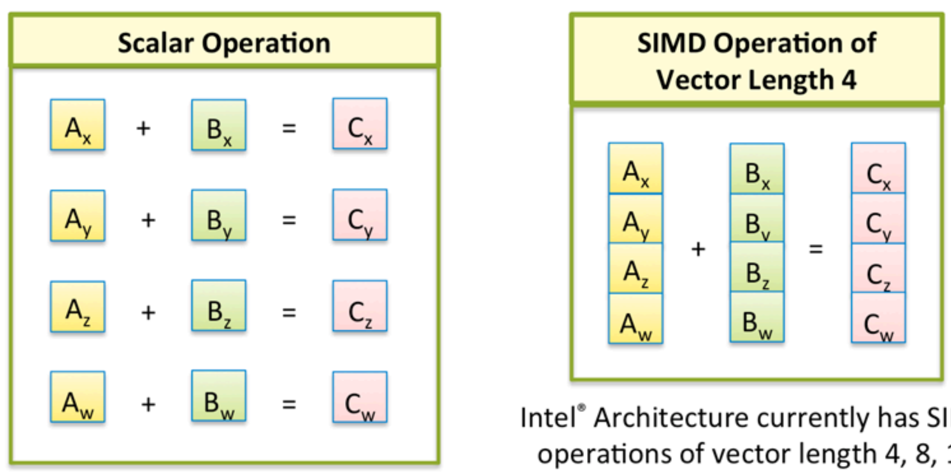


Single-Instruction Multiple-Data

- **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:

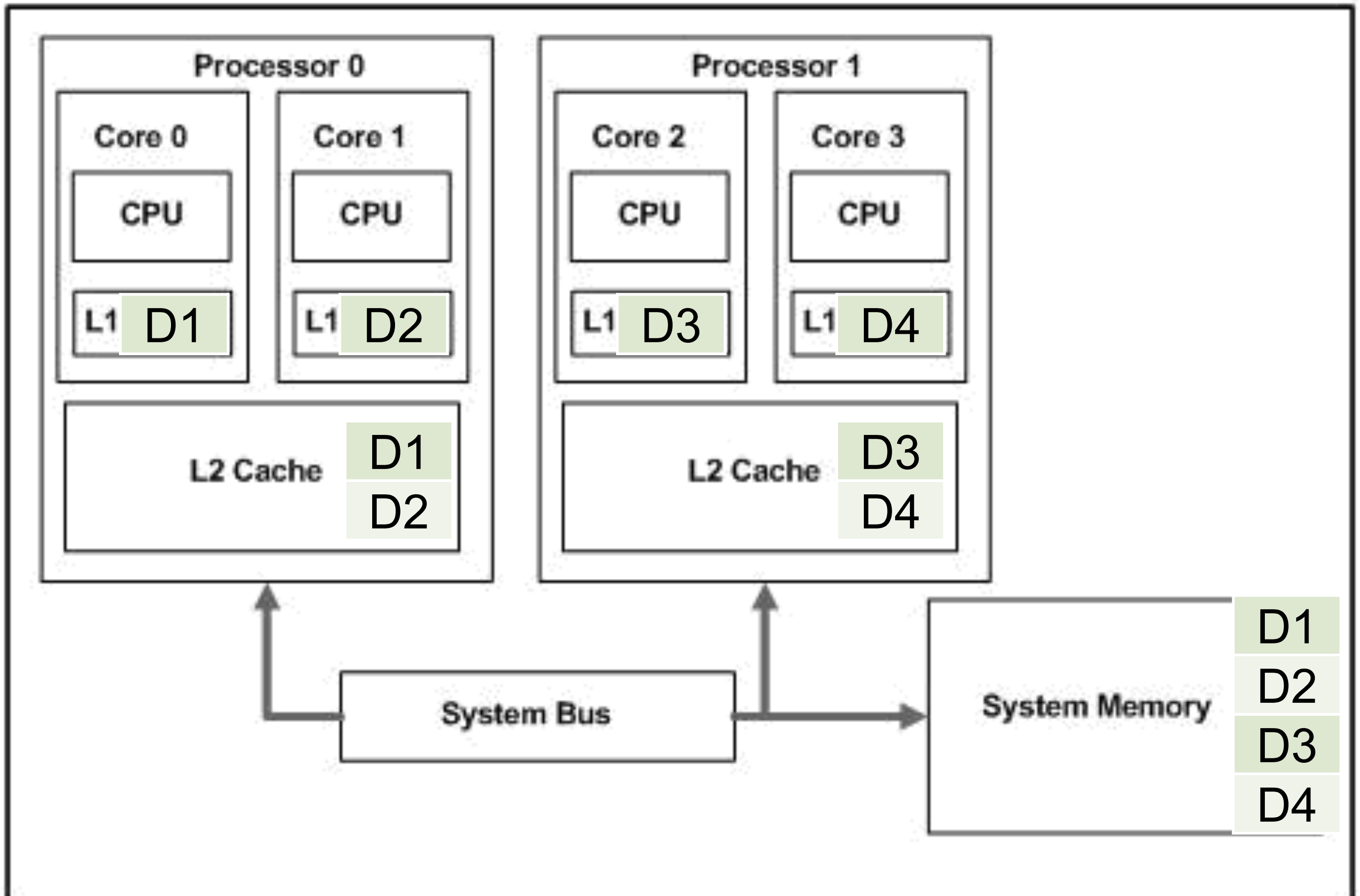


Intel® Architecture currently has SIMD operations of vector length 4, 8, 16

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:

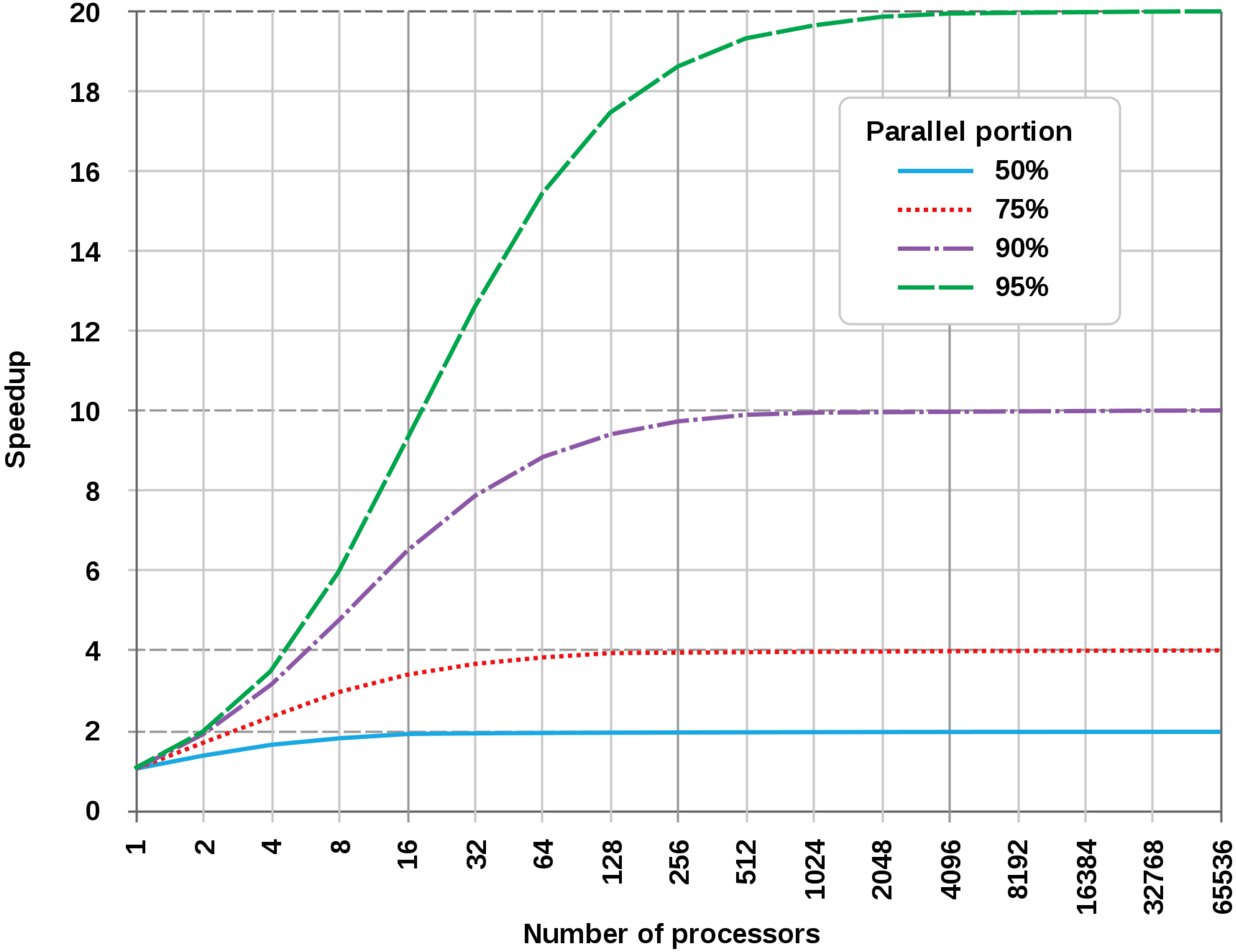
$T_{yes} \rightarrow T_{yes}/n$

$T_{no} \rightarrow T_{no}$

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

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

Amdahl's Law



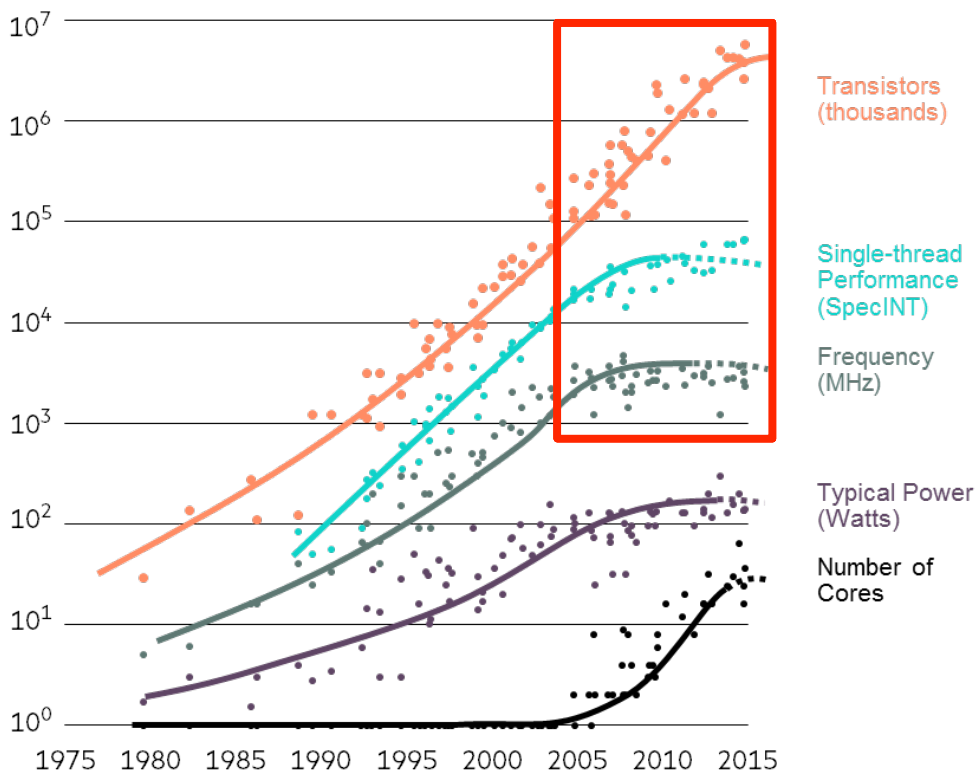
$$\text{Speedup} = \frac{n(1 + f)}{n + f}$$

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

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

Hardware Trends on Parallelism

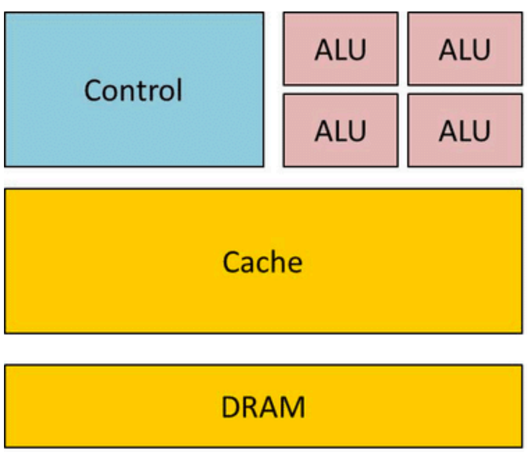
- Multi-core processors grew rapidly in early 2000s but hit physical limits due to packing efficiency and power issues
- End of "Moore's Law" and End of "Dennard Scaling"



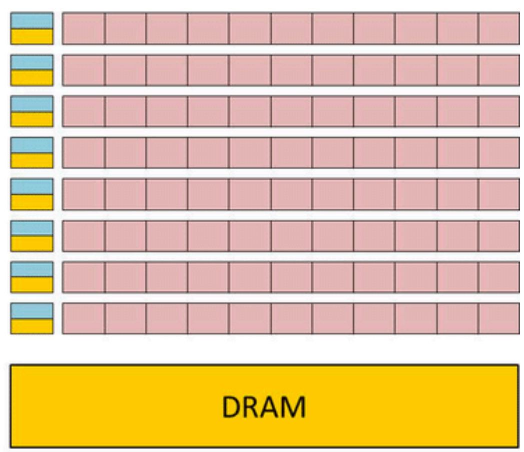
- Takeaway from hardware trends: it is hard for general-purpose CPUs to sustain FLOP-heavy programs like deep nets
- Motivated the rise of "accelerators" for some classes of programs

Hardware Accelerators: GPUs

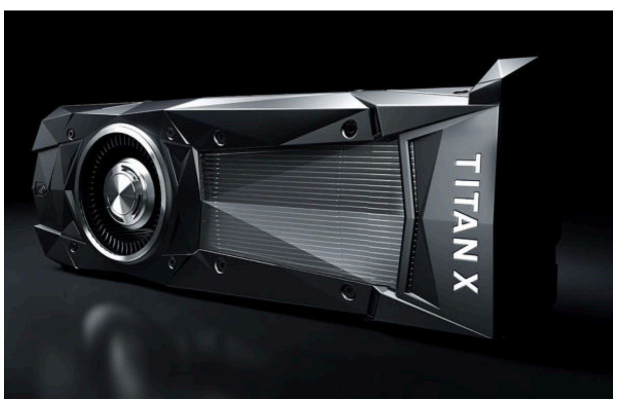
- **Graphics Processing Unit (GPU)**: Tailored for matrix/tensor ops
- Basic idea: use tons of ALUs; massive data parallelism (SIMD on steroids); Titan X offers ~11 TFLOP/s!
- Popularized by NVIDIA in early 2000s for video games, graphics, and video/multimedia; now ubiquitous in deep learning
- CUDA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



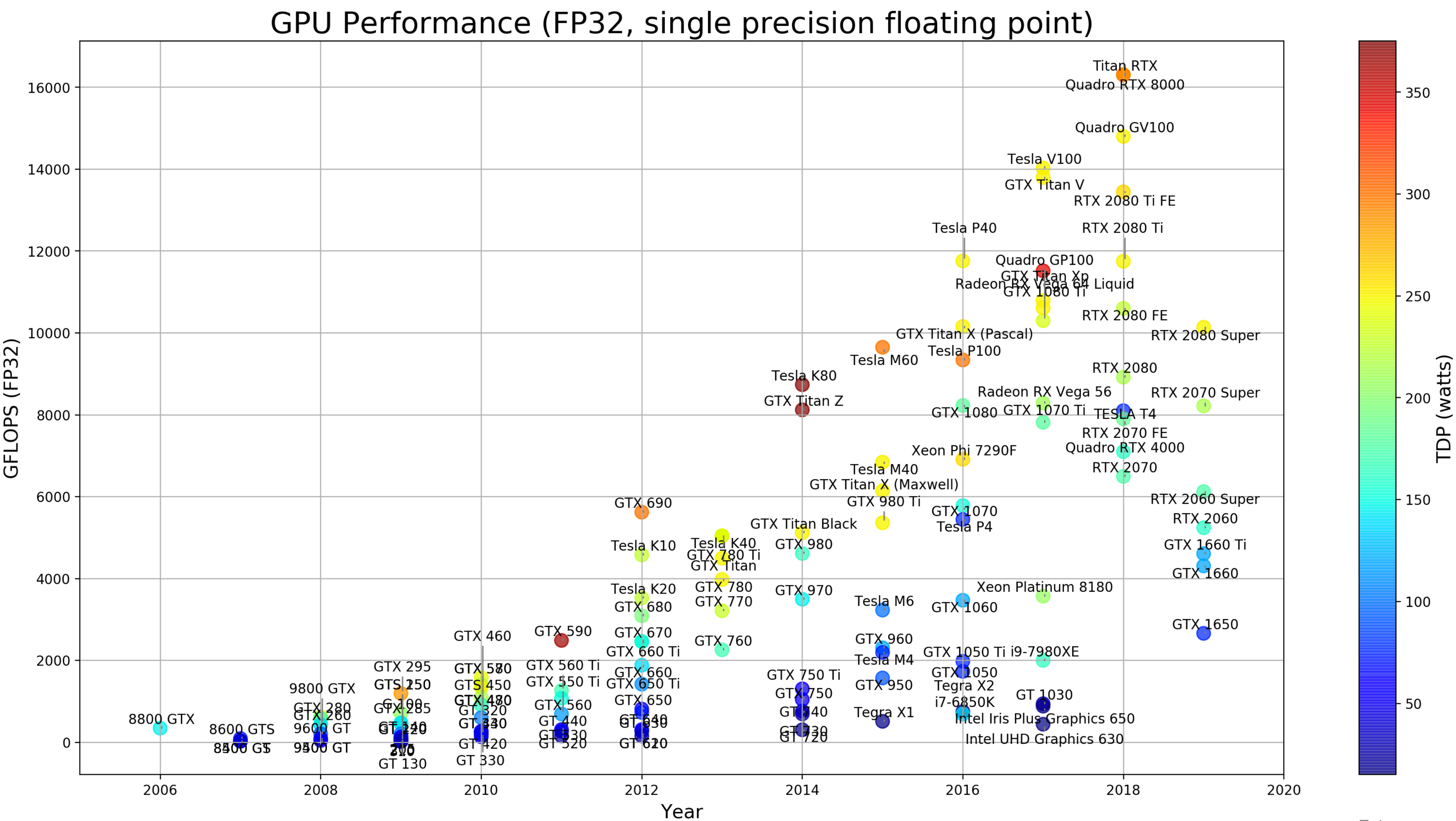
CPU



GPU

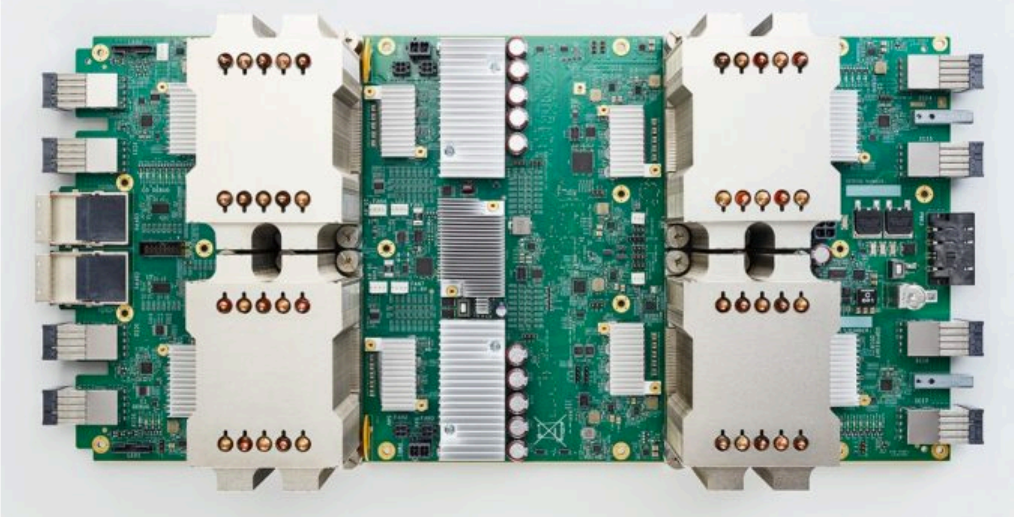


GPUs on the Market



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