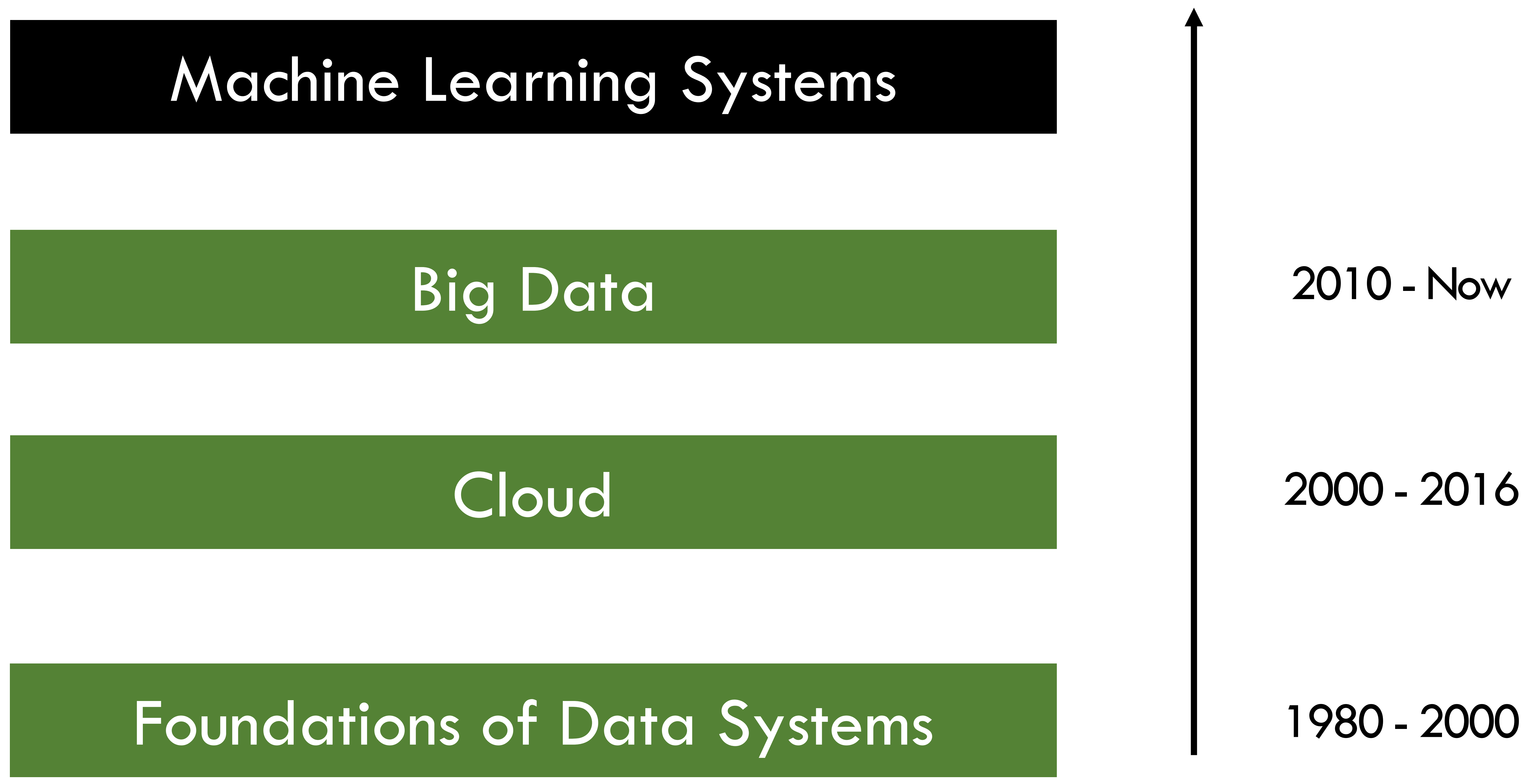
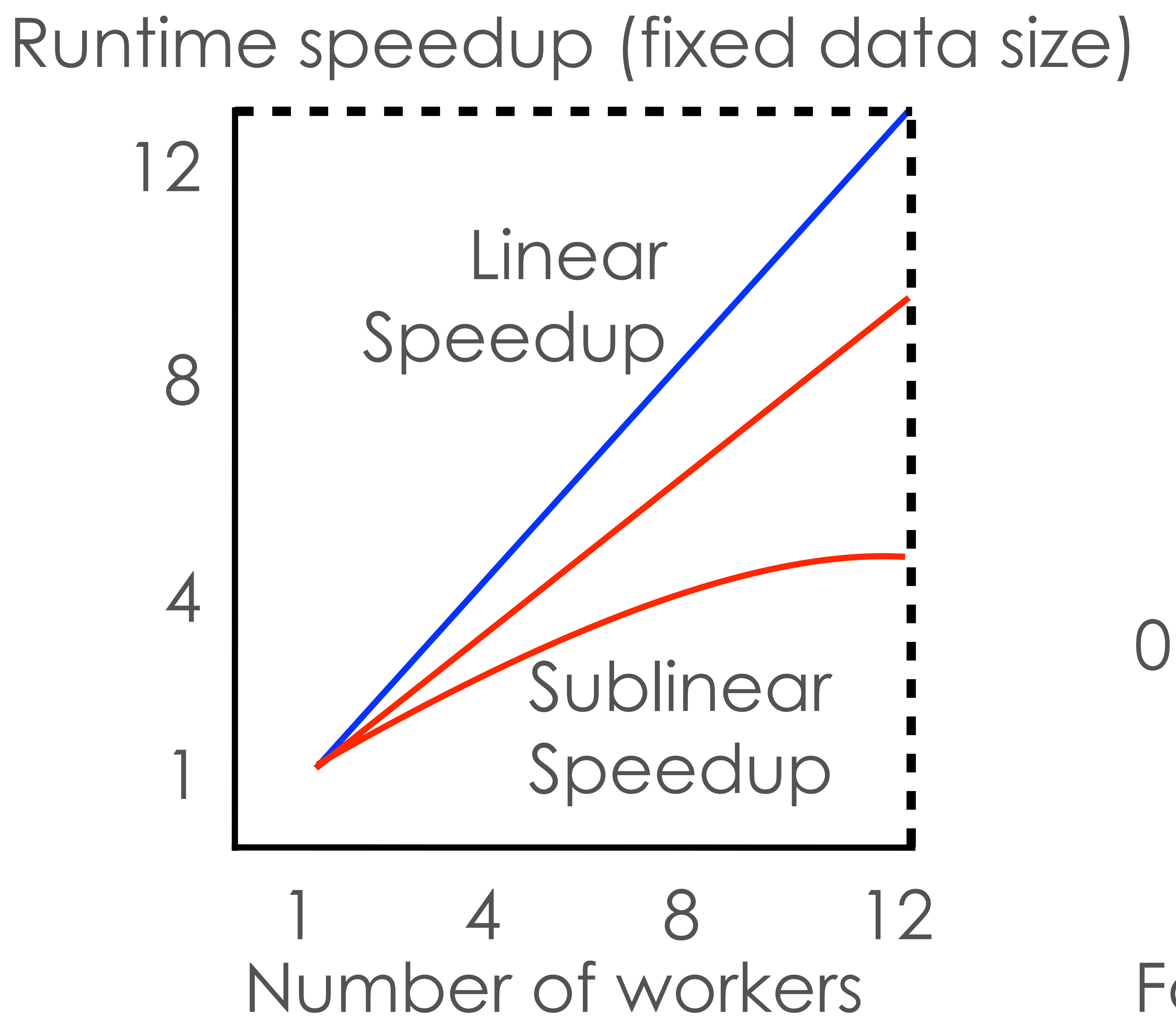
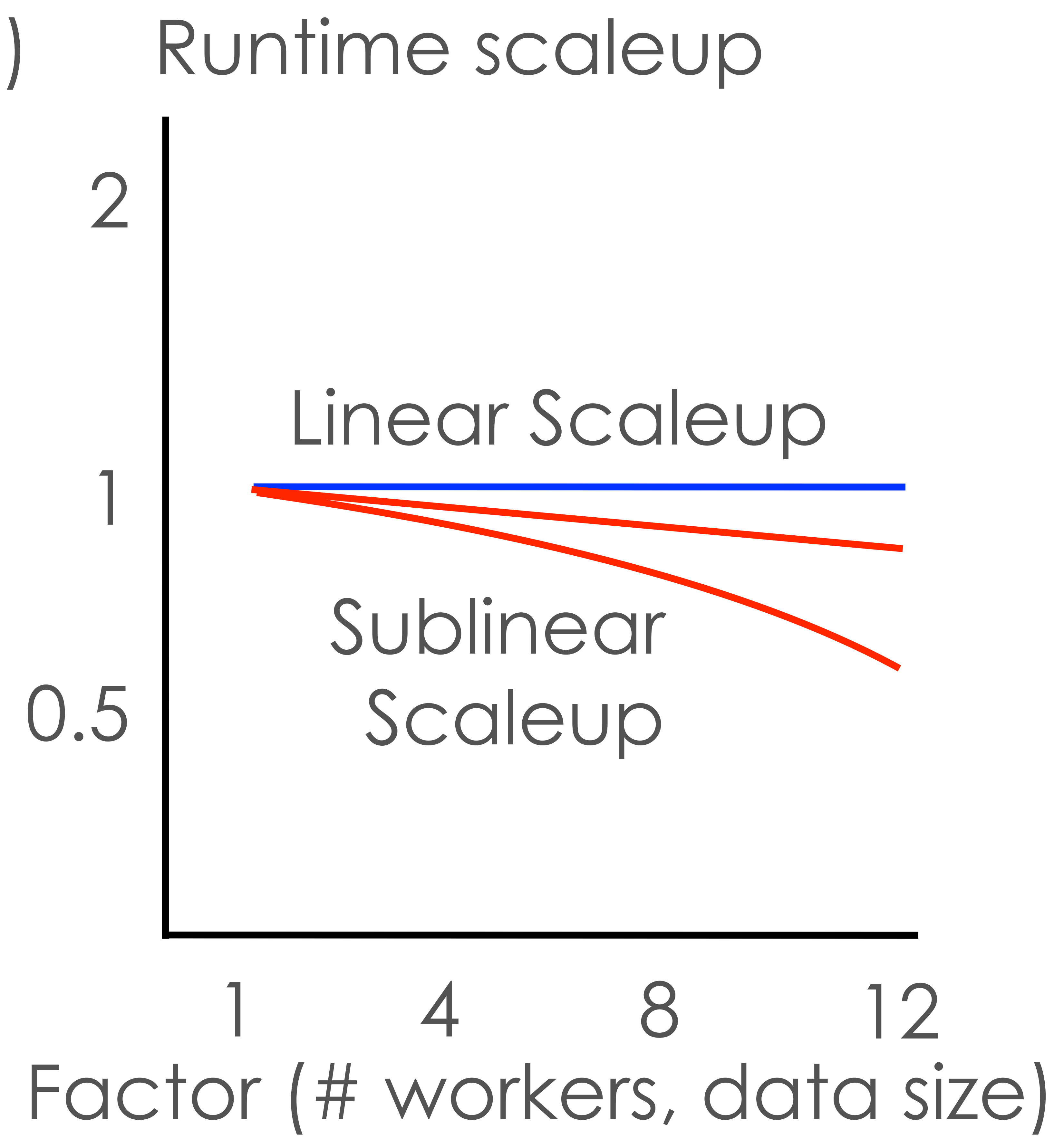
Where We Are


Machine Learning Systems
Big Data
2010 - Now
Cloud
2000 - 2016
Foundations of Data Systems
1980 - 2000

Weak and Strong Scaling



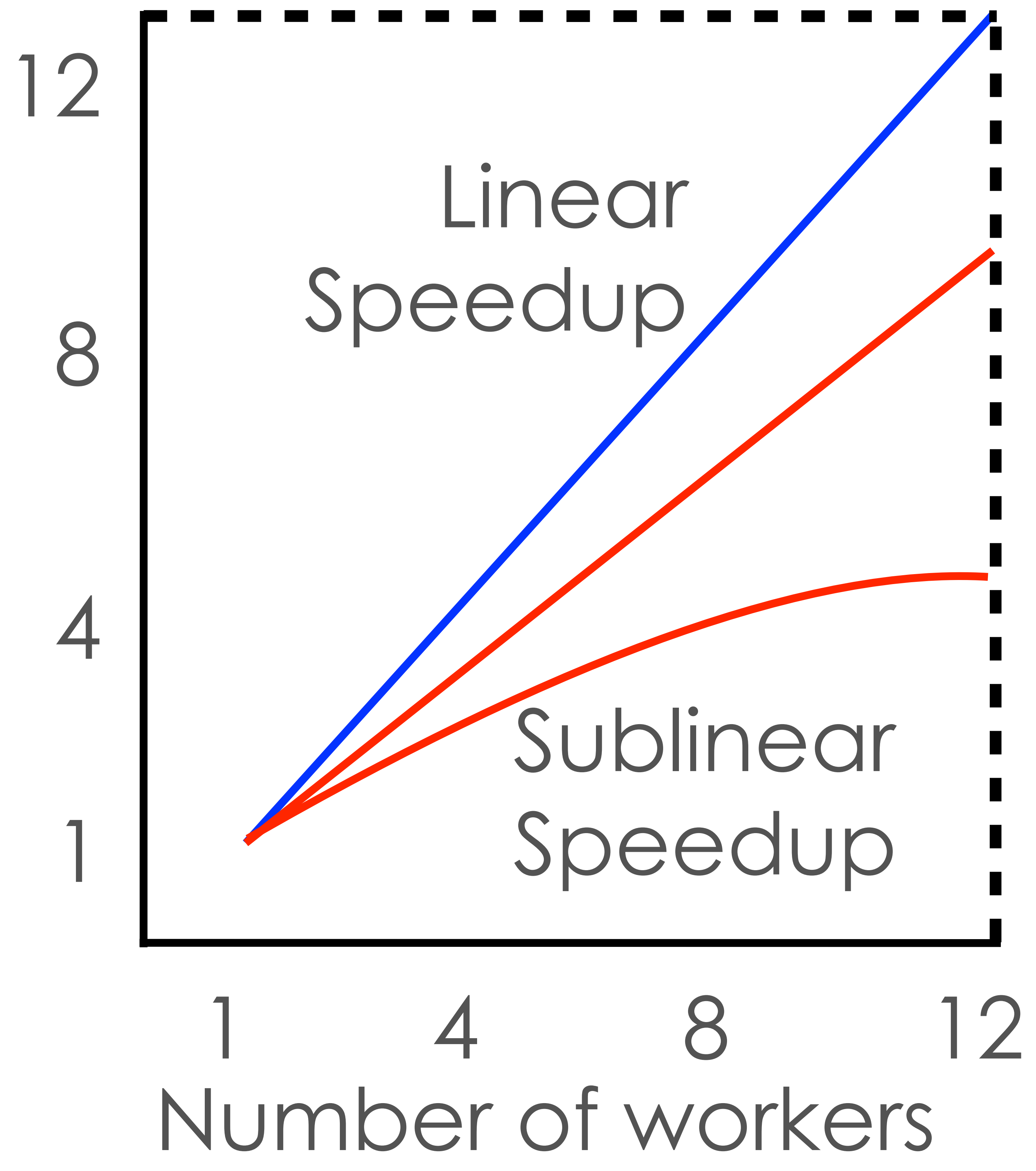
Speedup plot / Strong scaling

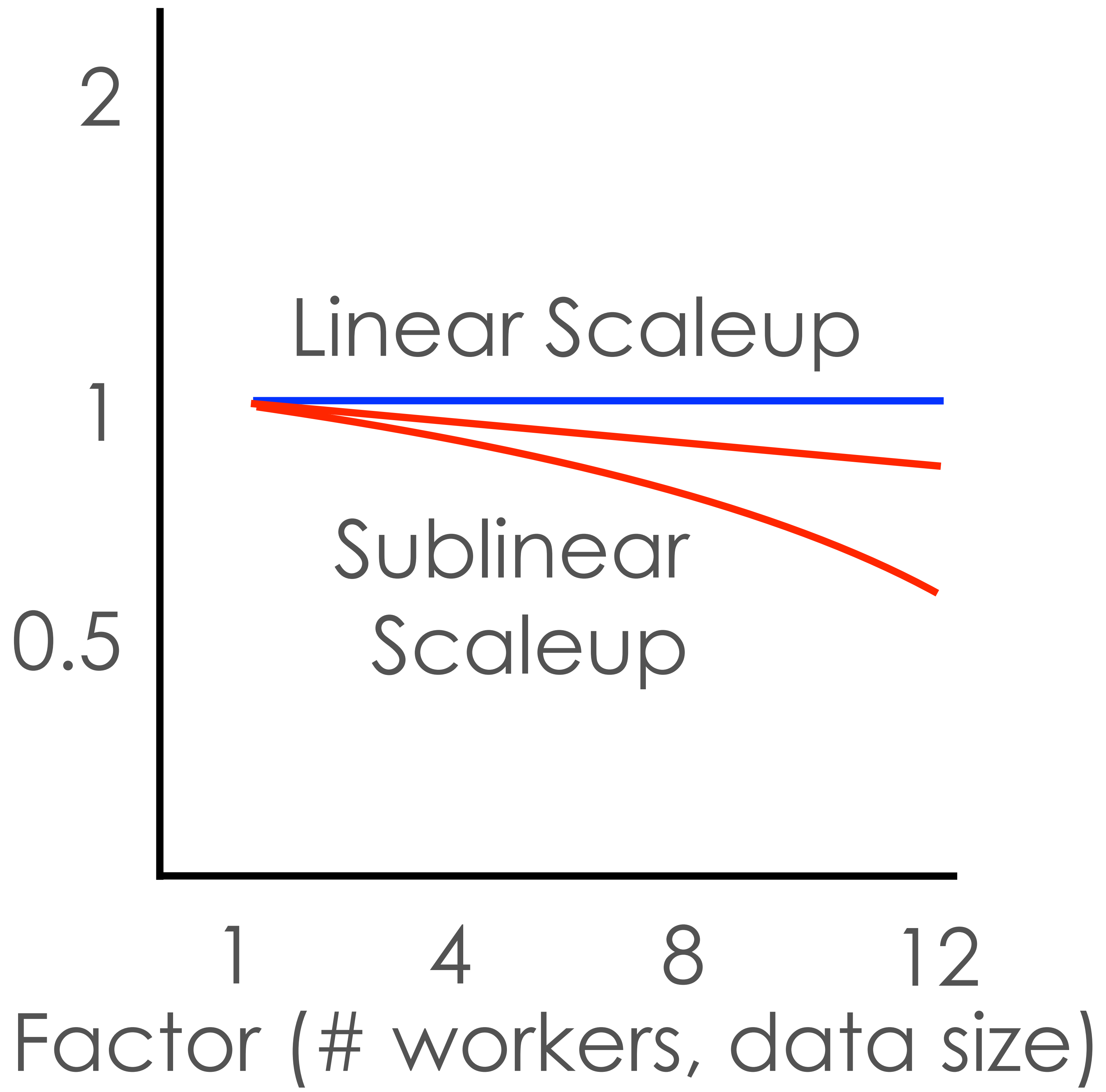


Scaleup plot / Weak scaling

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

Discussion: Is *superlinear* speedup/scaleup possible?





Some Clarifications on Terms

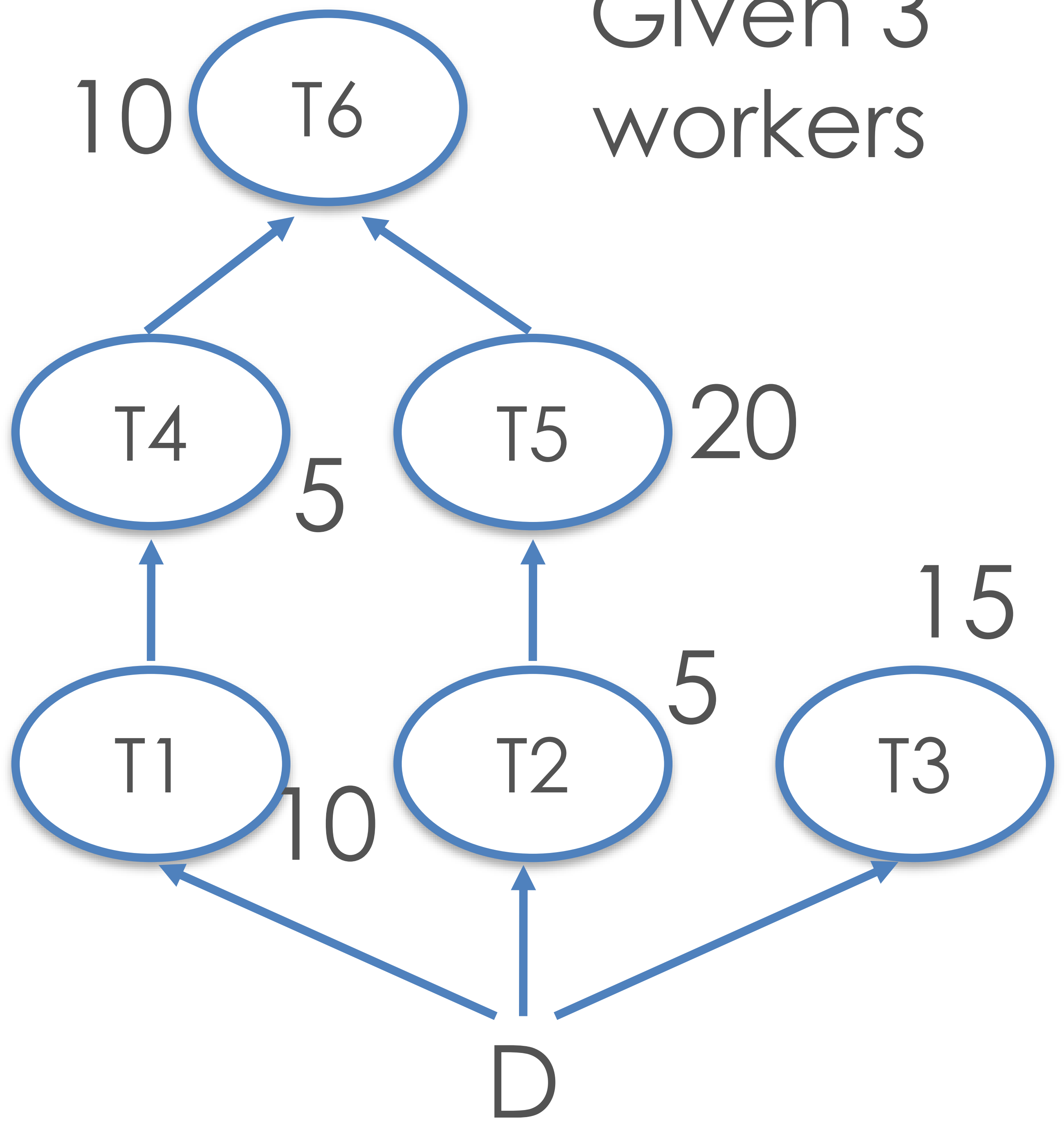
$$\textbf{Speedup} = \frac{\text{Completion time given only 1 worker}}{\text{Completion time given n (>1) workers}}$$

- These terms almost all refer to the above, but they are slightly different
  - Speedup, acceleration -> strong scaling
  - Scaling, scale-up -> weak scaling
  - Scalability -> both
- “system A is very scalable”
  - When you add 1 more workers, the speedup increase by ~1
- “system A is more scalable than system B”
  - When you add 1 more worker, the speedup of system A is larger than that of system B

Idle Times in Task Parallelism

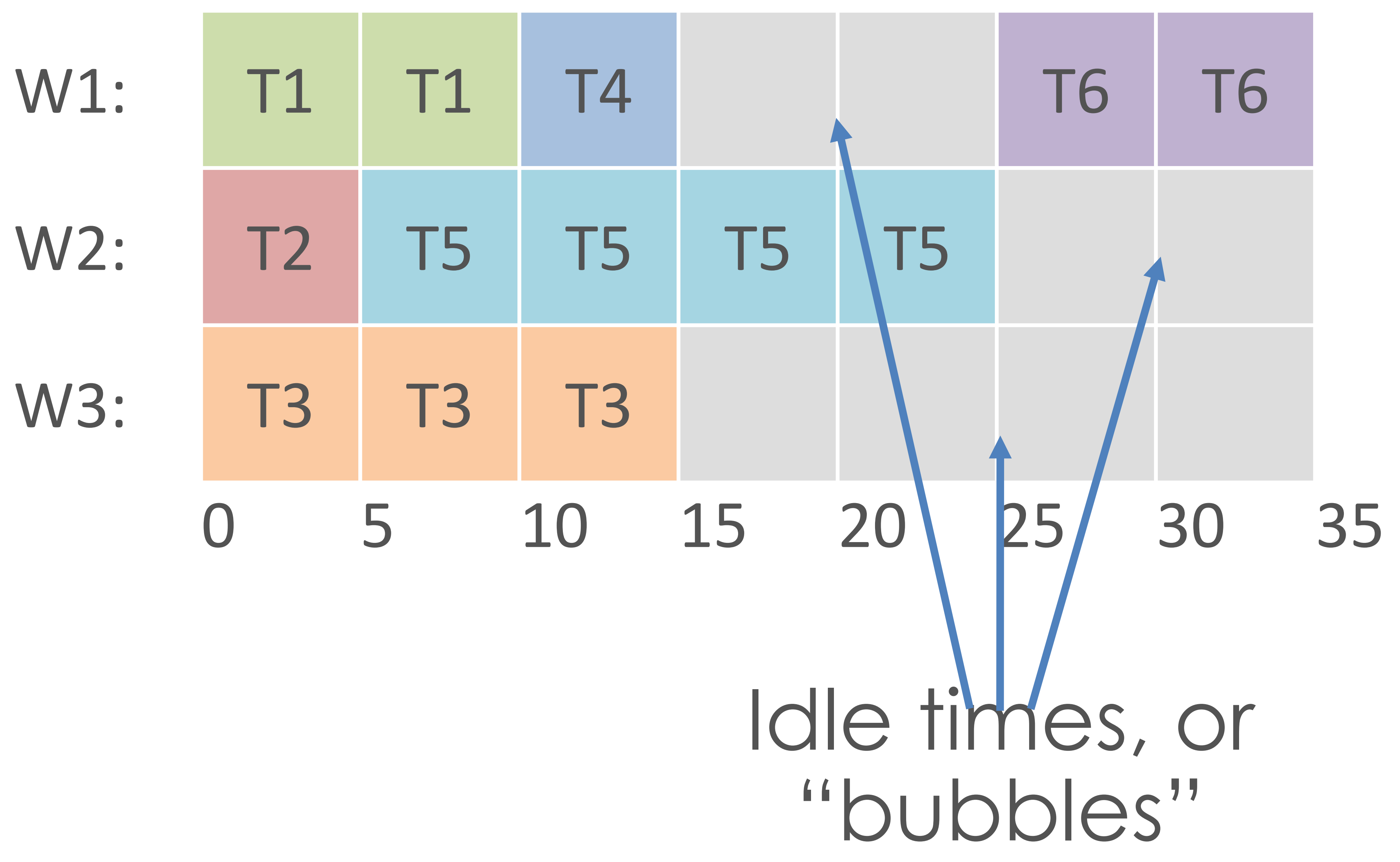
Due to varying task completion times and varying degrees of parallelism in workload, idle workers waste resources

Example:



Given 3 workers

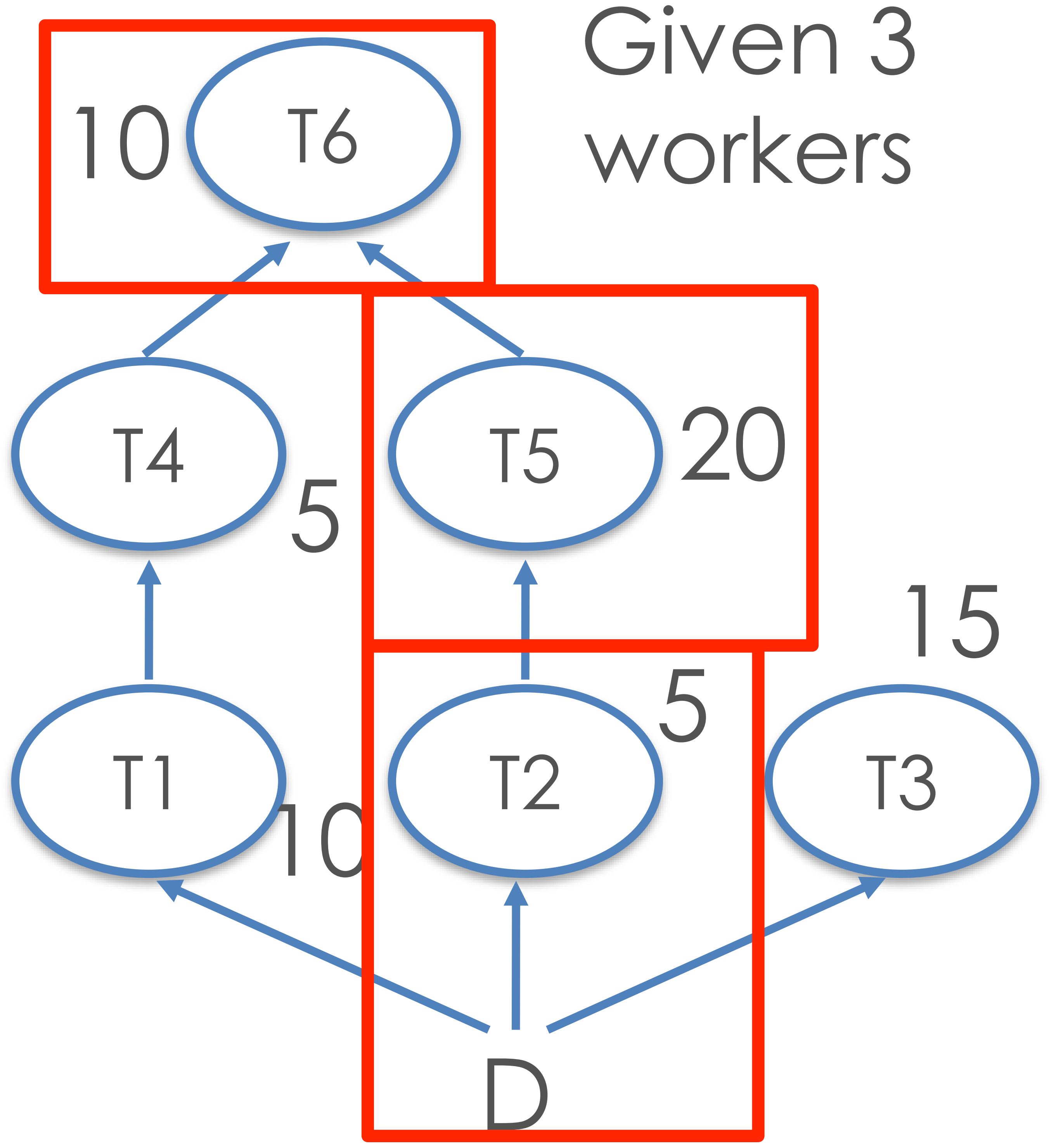
Gantt Chart visualization of schedule:



Idle Times in Task Parallelism

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

Example:



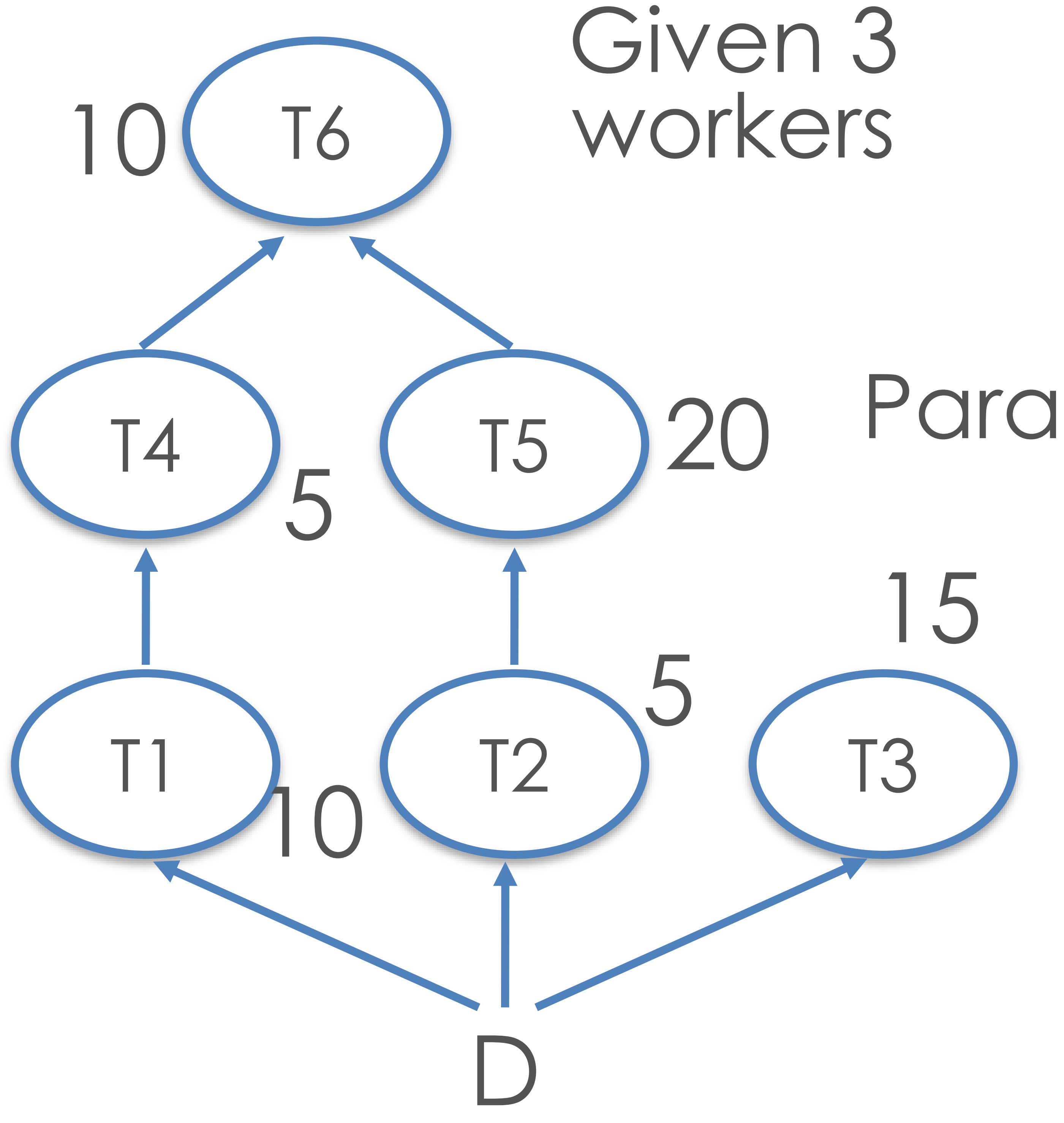
Given 3 workers

- 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

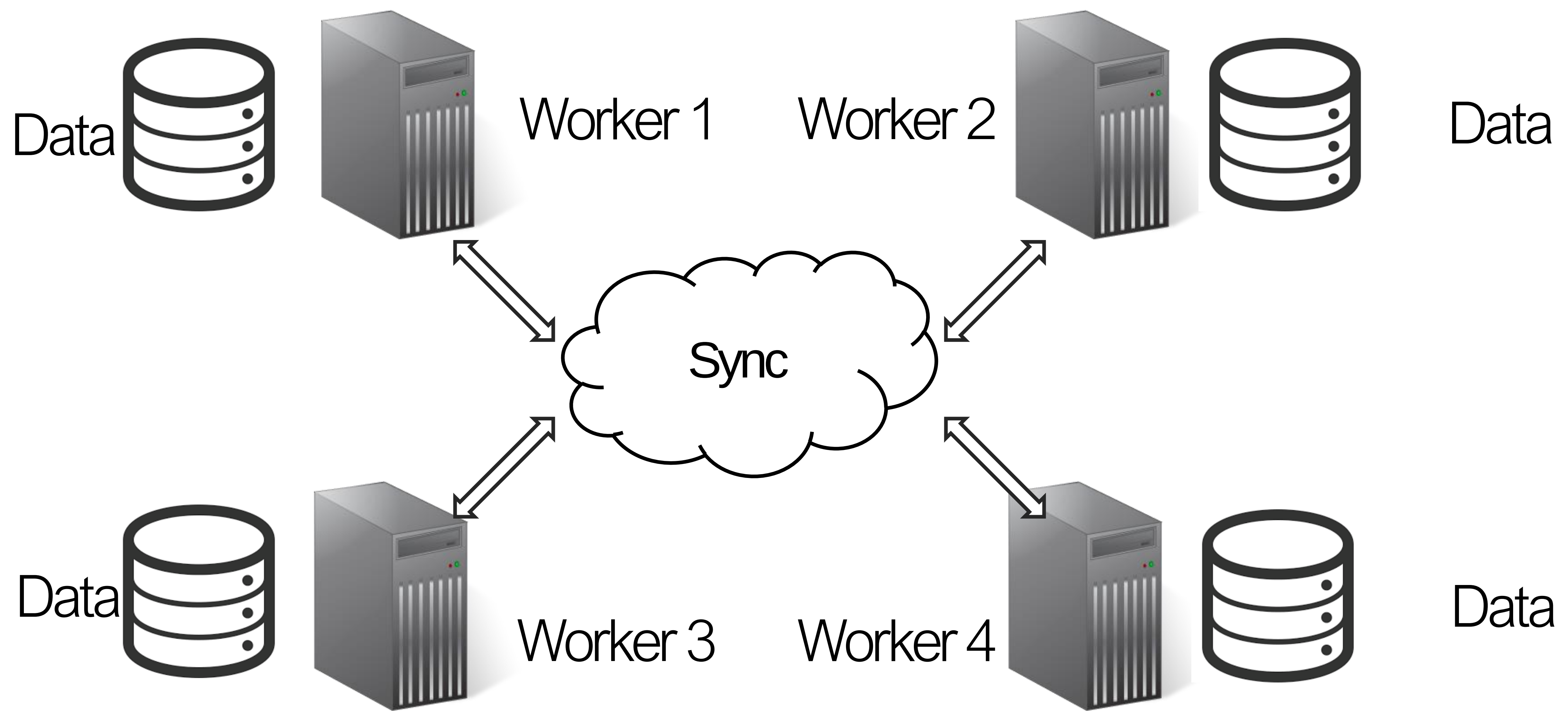
Ideal/linear speedup is 3x

Q: *Why is it only 1.9x?*

Today’s topic: Parallelism

- Express data processing in abstraction
- Parallelisms
  - Task parallelism
  - **Data parallelism**
  - Parallel Processing Chips

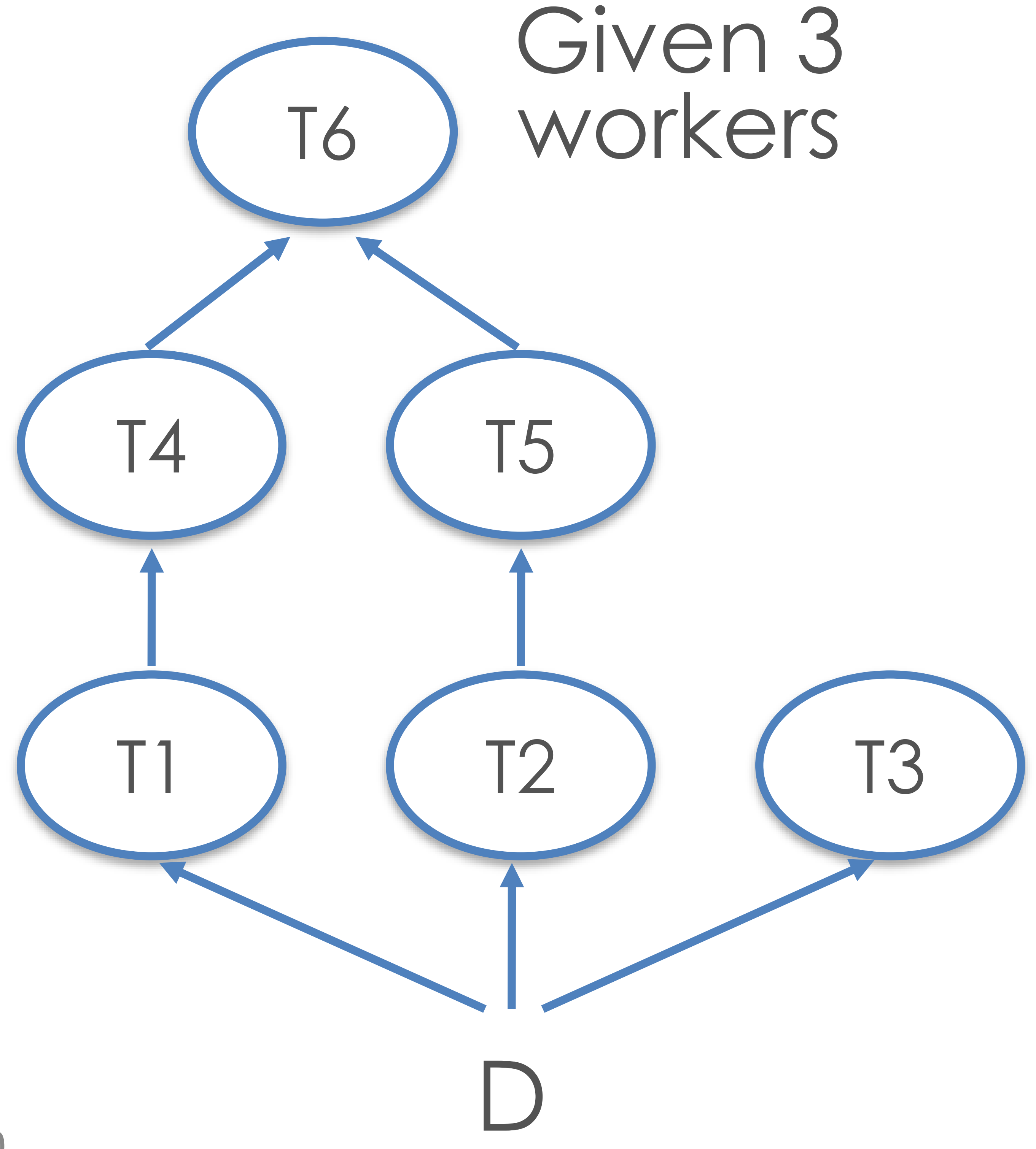
Recall: Data parallelism in ML (Lecture 10)



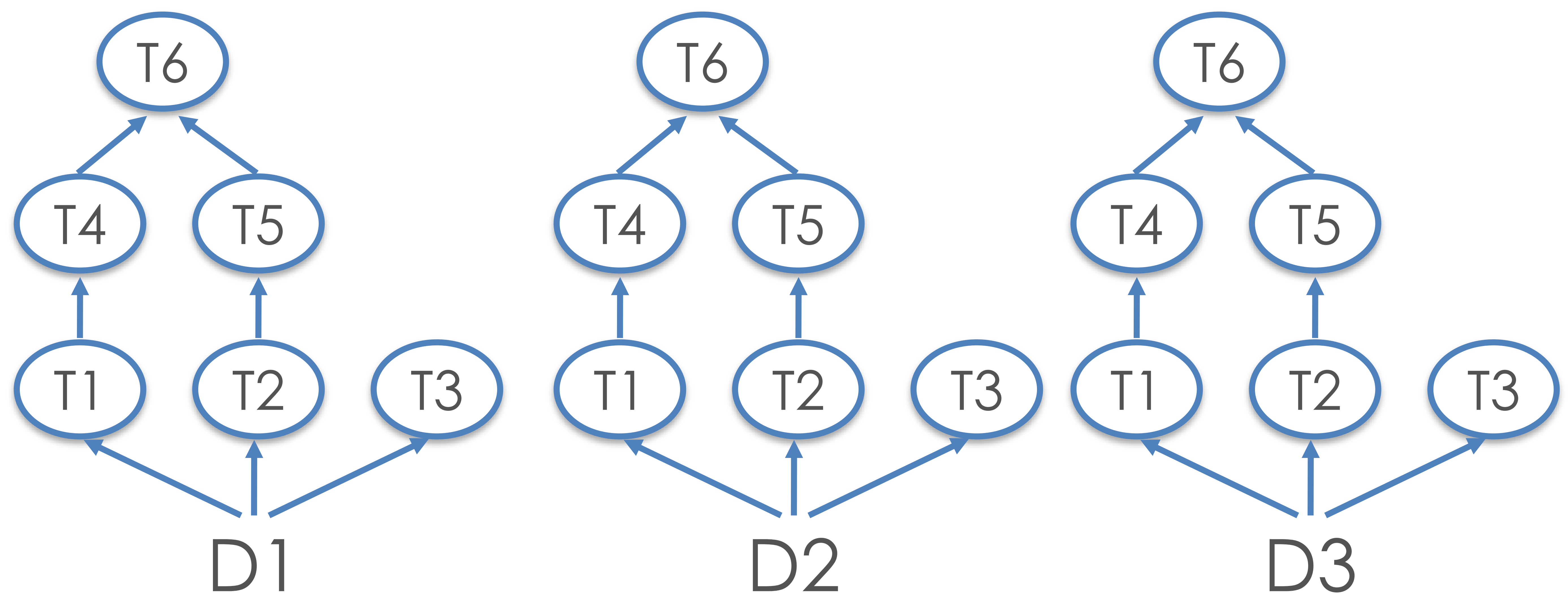
$$\theta^{(t+1)} = \theta^{(t)} + \varepsilon \sum_{p=1}^{P} \nabla_{\mathcal{L}}(\theta^{(t)}, D_p^{(t)})$$

Data parallelism: abstraction of SIMD/SIMT/SPMD

Q: How to represent data parallelism in dataflow graph notions?

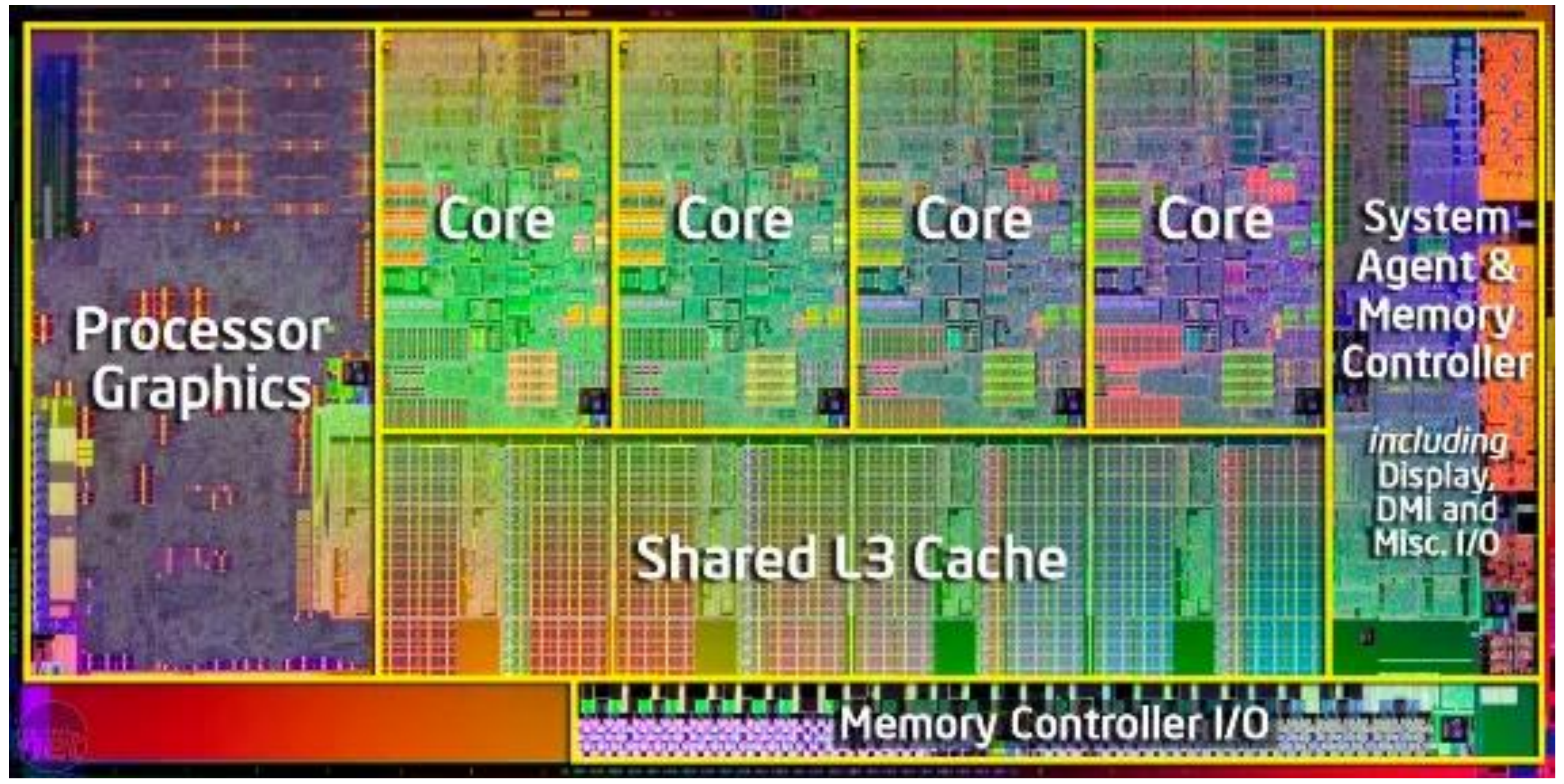


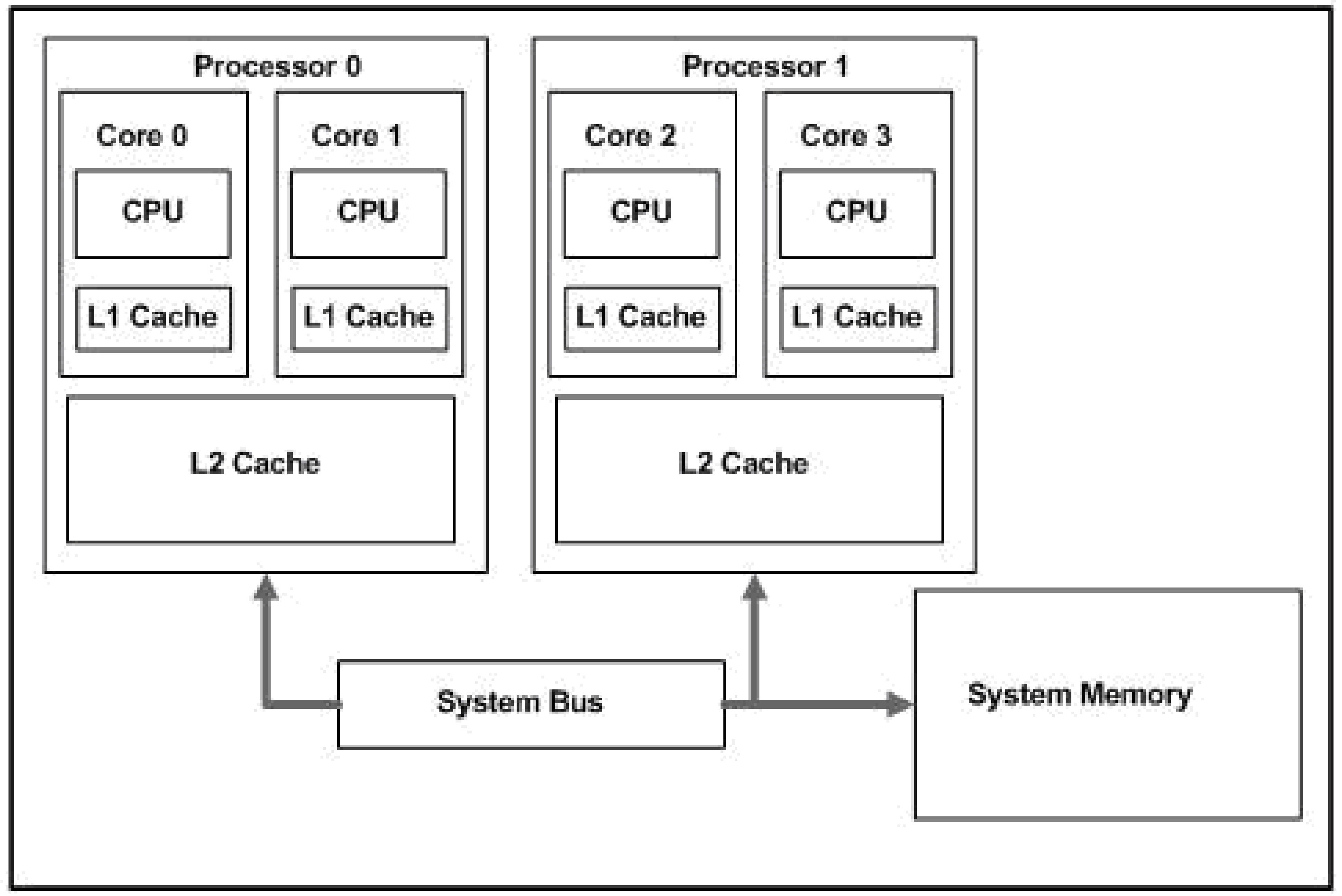
Given 3 workers



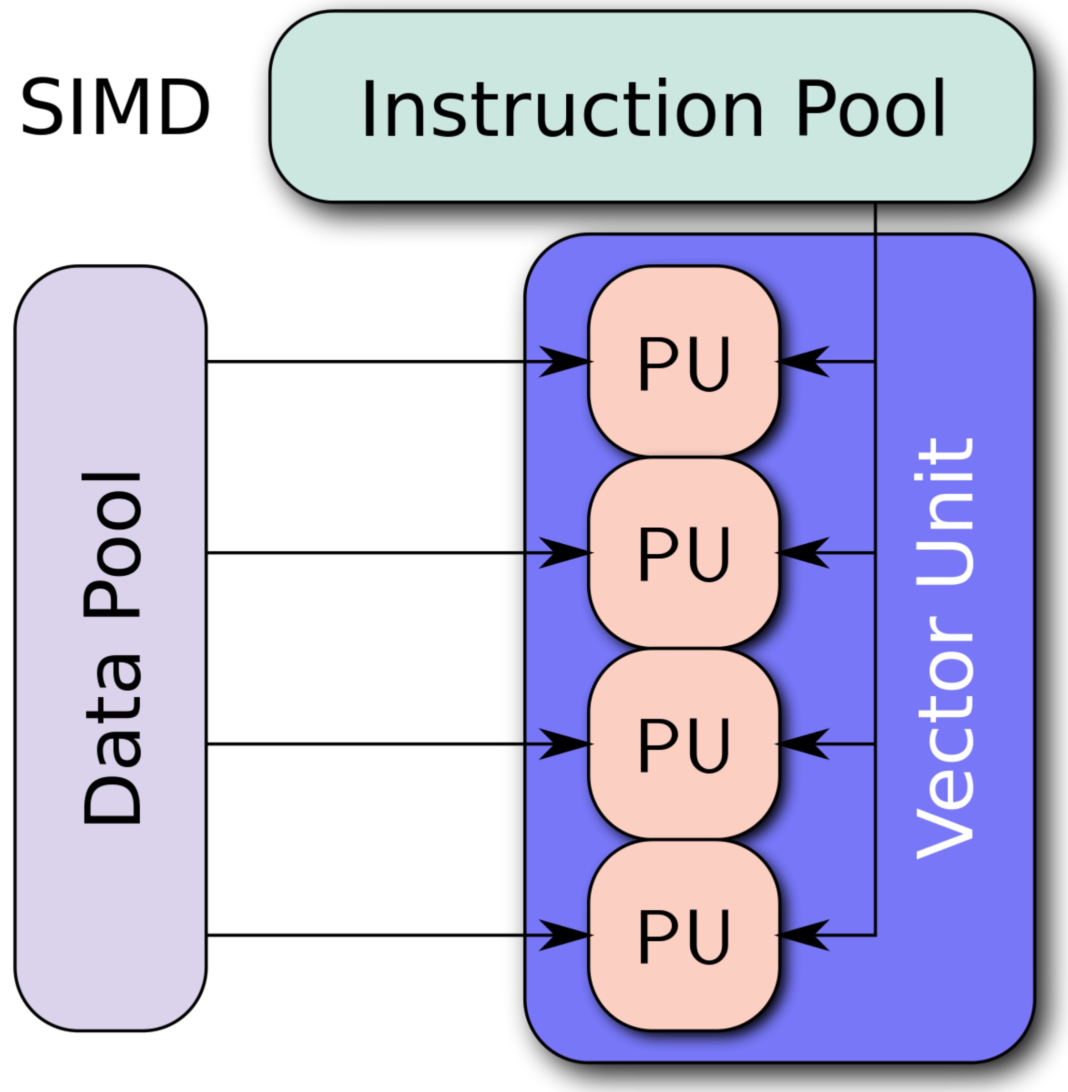
Data parallelism is built in with today's Processors

- **Recall Lecture 2:**
  - Modern computers often have multiple processors and multiple *cores* per processor, with a hierarchy of shared caches

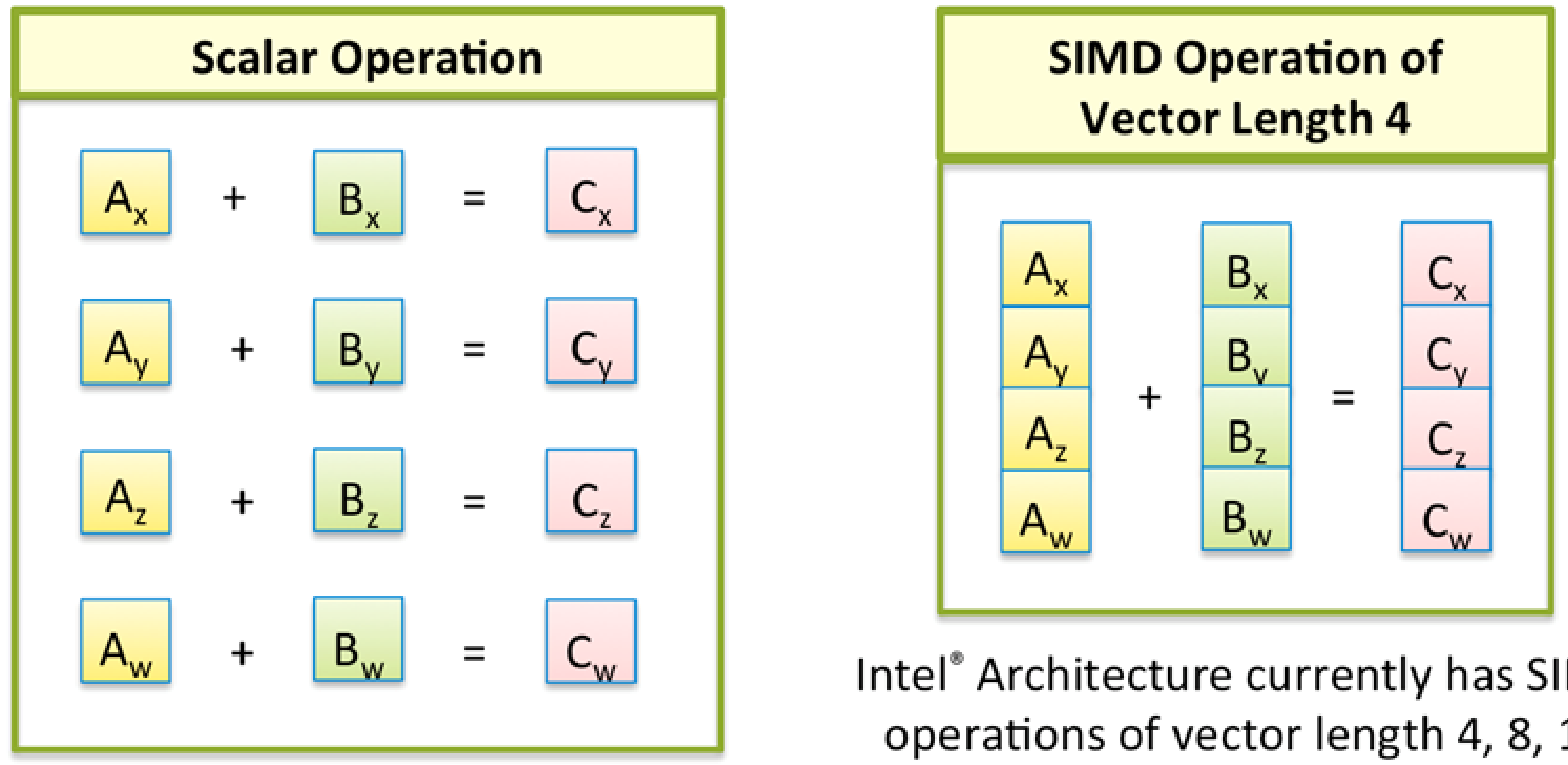




Single-Instruction Multiple-Data



Example for SIMD in data science:


Scalar Operation
A_x + B_x = C_x
A_y + B_y = C_y
A_z + B_z = C_z
A_w + B_w = C_w
SIMD Operation of Vector Length 4
A_x A_y A_z A_w + B_x B_y B_z B_w = C_x C_y C_z C_w

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

Quantifying Efficiency of Data Parallelism

- As with task parallelism, we measure the speedup:

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

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:

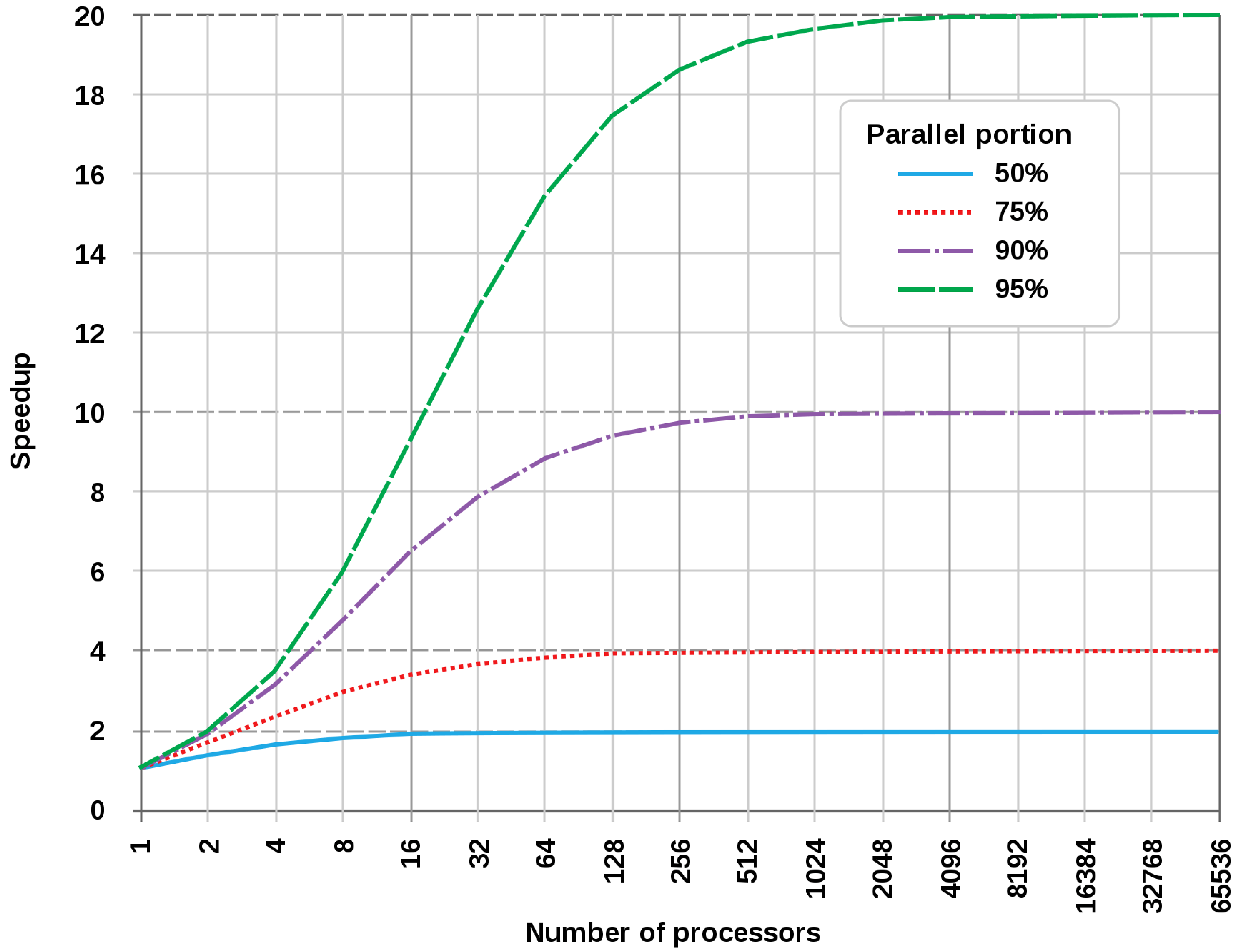
$$T_{yes} \longrightarrow T_{yes}/n$$

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

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

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

Amdahl's Law:



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

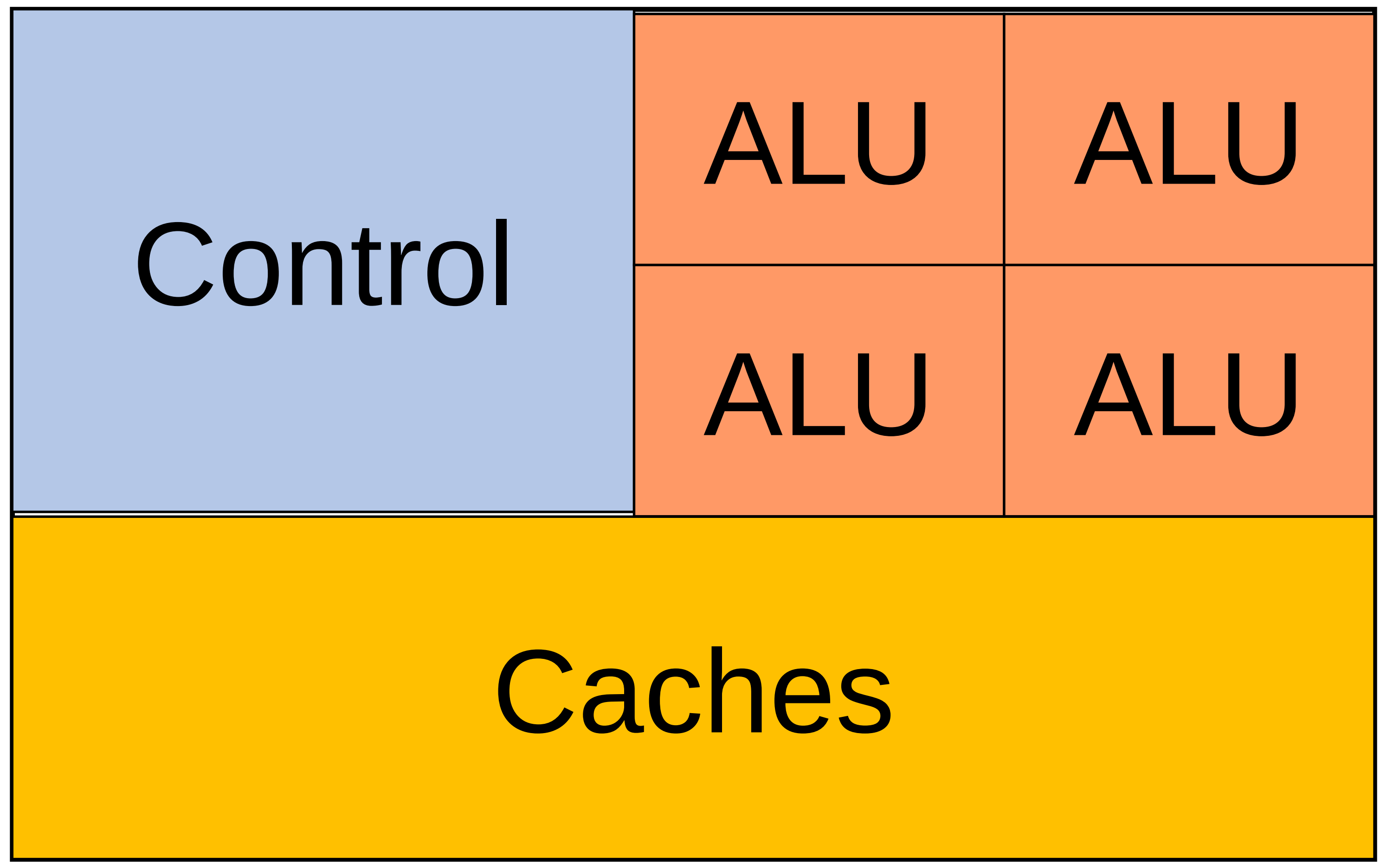
Parallel portion =

$f / (1 + f)$

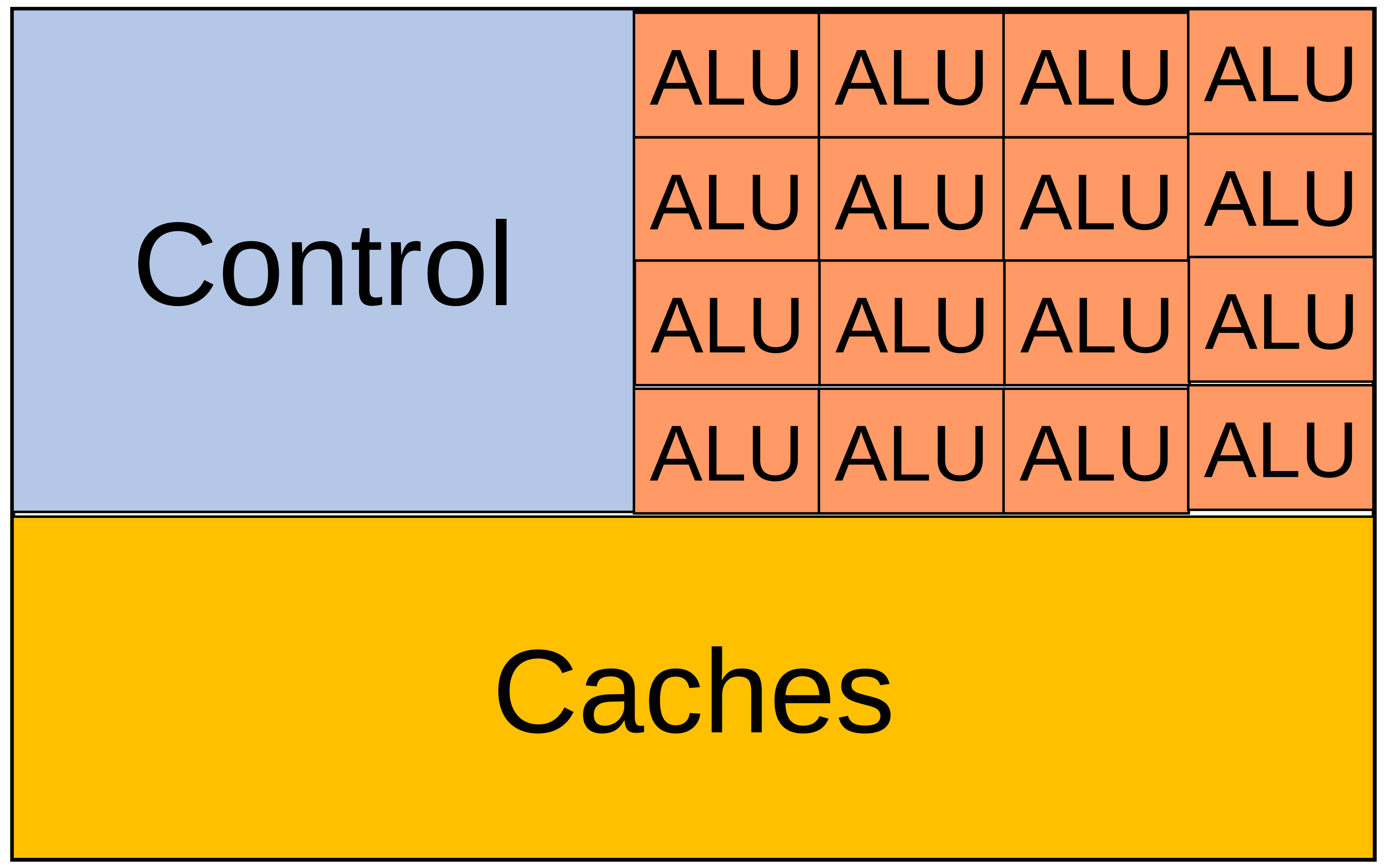
Speedup =

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

The Problem of Chip Design



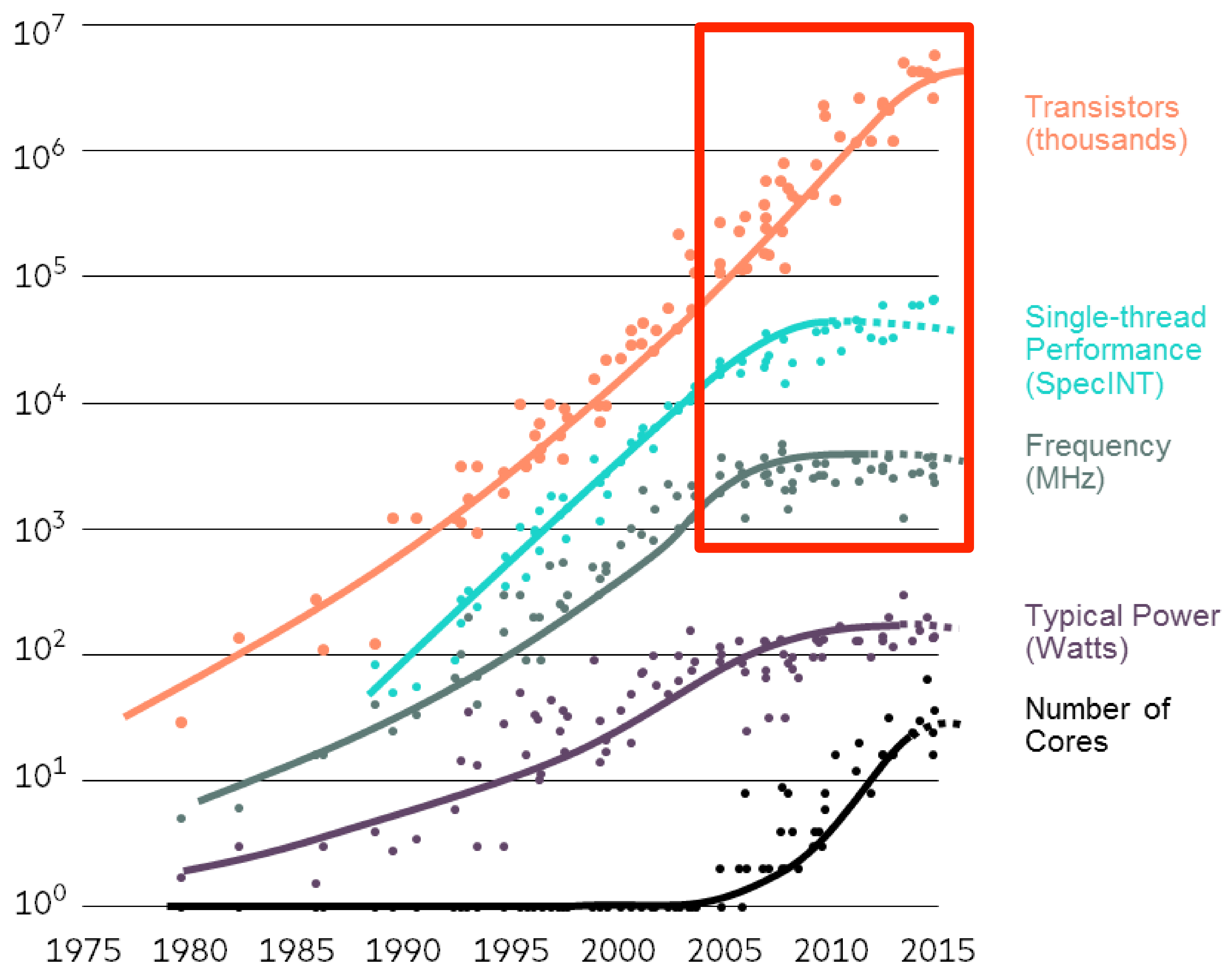
If we're able to reduce to size
of ALU while keeping its power



That's why you see trends: 70nm -> 60nm -> 50nm -> … -> what is the best now?

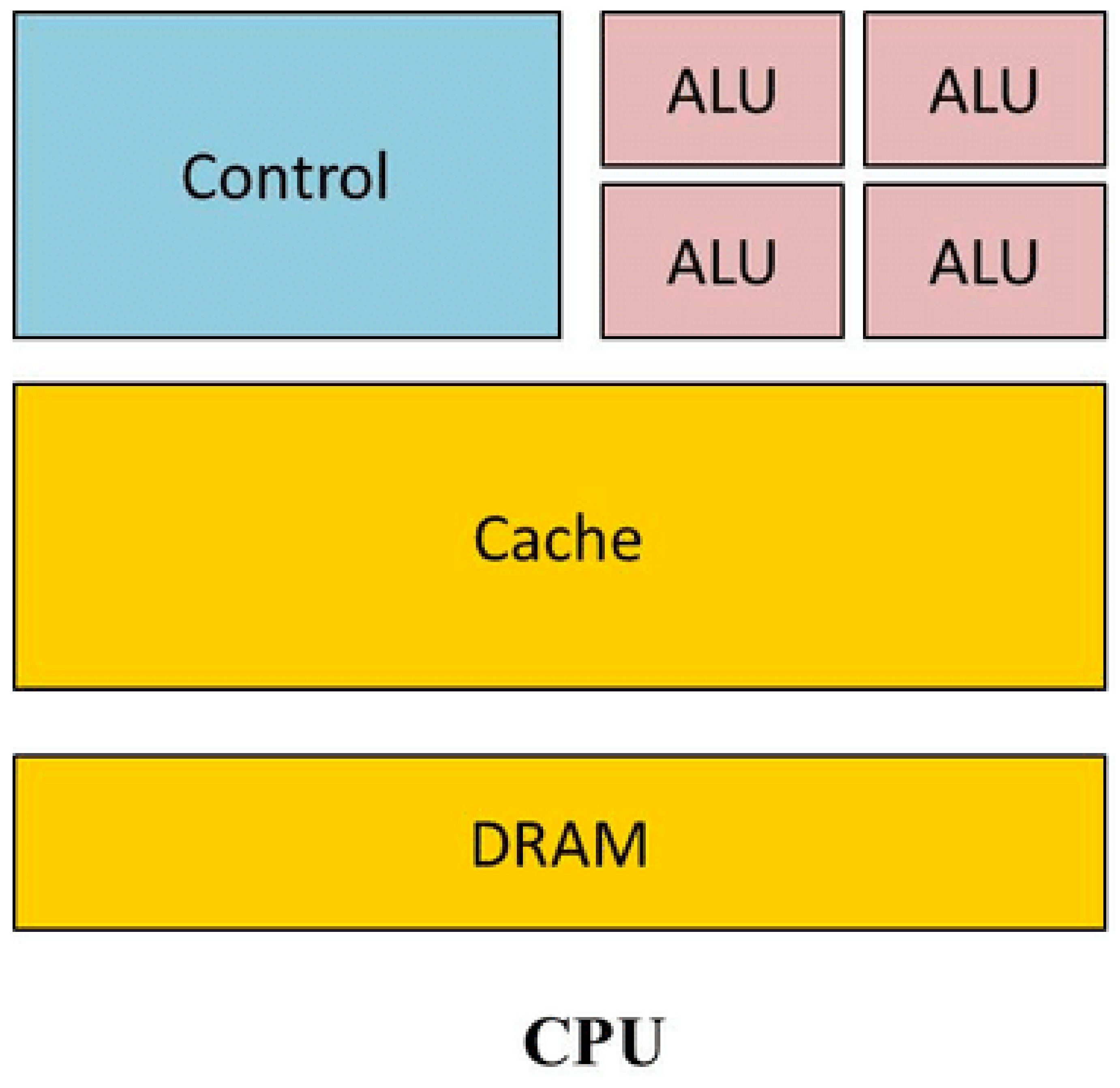
Problem: this is not substantiable; there are also power/heat issues when you put more ALUs in

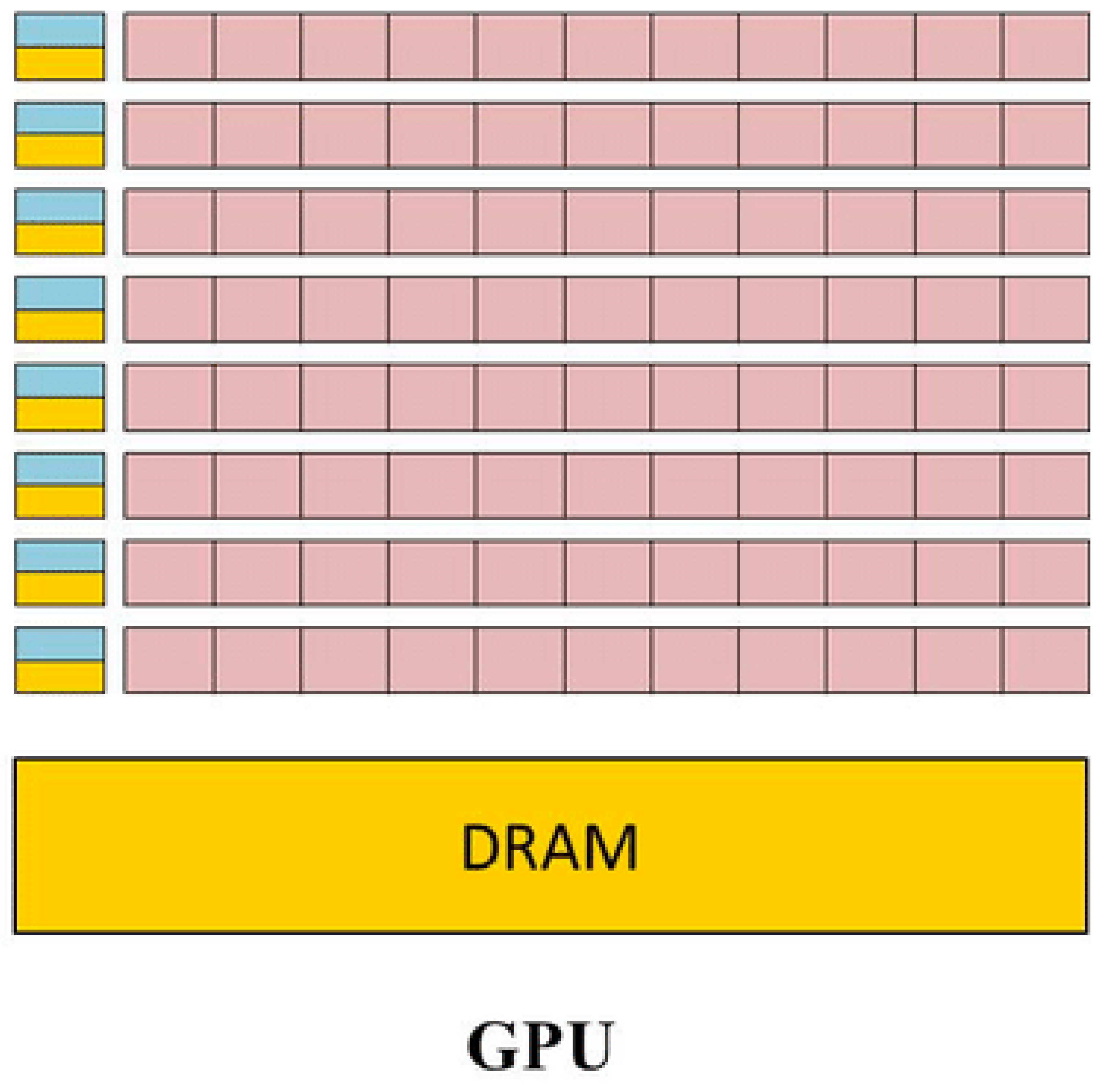
- That's why you see trends: 70nm -> 60nm -> 50nm -> … -> what is the best now?



- That's why you see trends: 70nm -> 60nm -> 50nm -> … -> what is the best now?
- 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

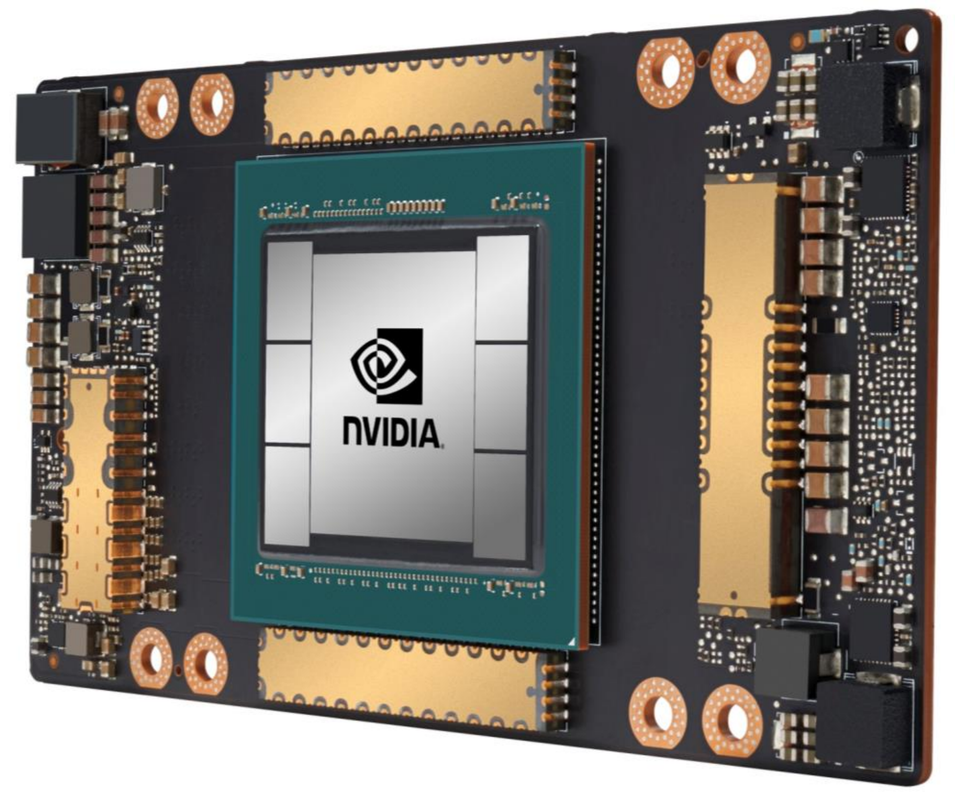
Idea: How about we use a lot of weak/specialized cores





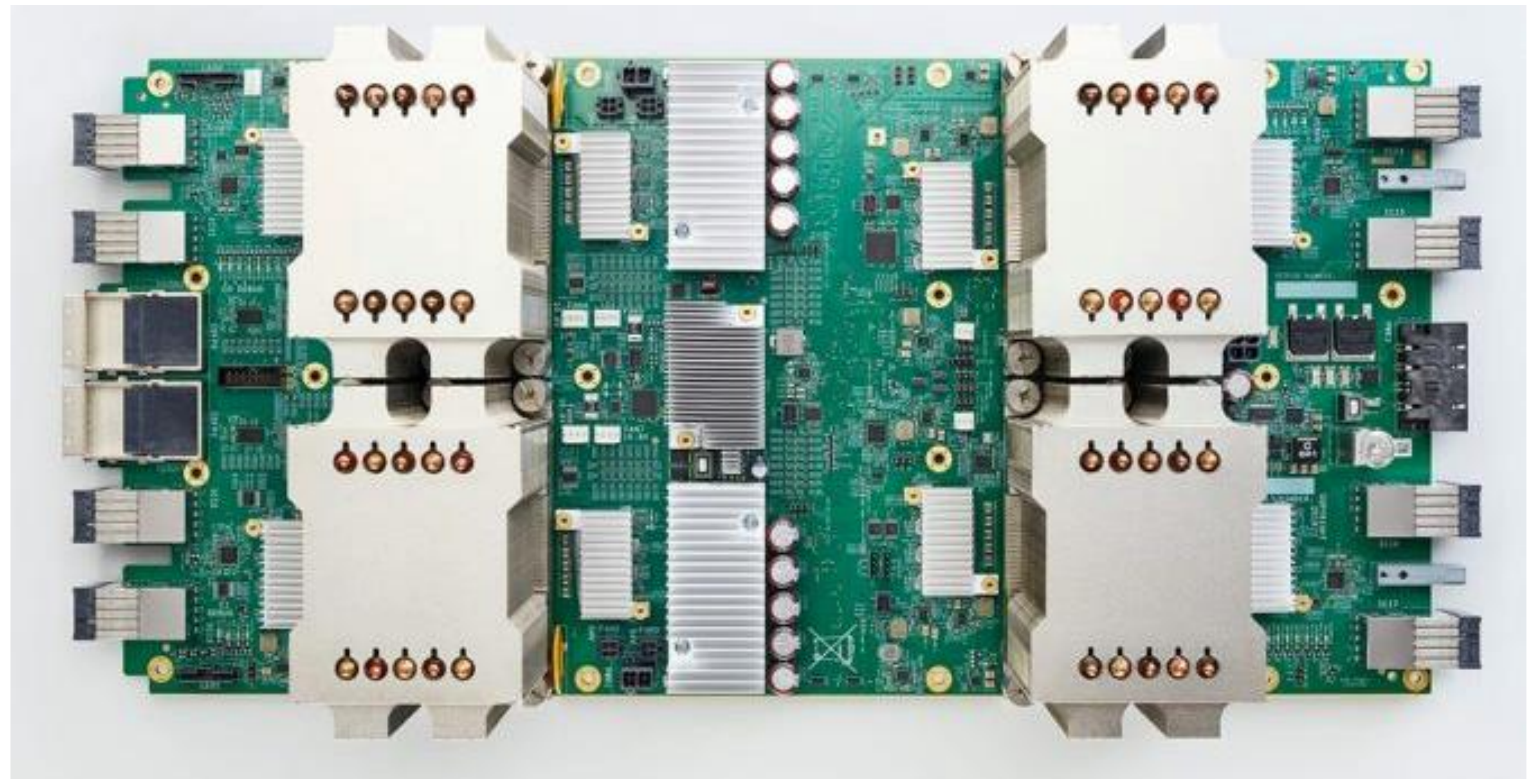
Hardware Accelerators: GPUs

- **Graphics Processing Unit (GPU)**: Tailored for matrix/tensor ops
- Basic idea: Use tons of ALUs (but weak and more specialized); massive data parallelism (SIMD on steroids); now H100 offers ~980 TFLOPS for FP16!
- 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), NCCL, etc.



Other Hardware Accelerators

- **Tensor Processing Unit (TPU):**
- **An "application-specific integrated circuit" (ASIC) created by Google in mid 2010s; used for AlphaGo**



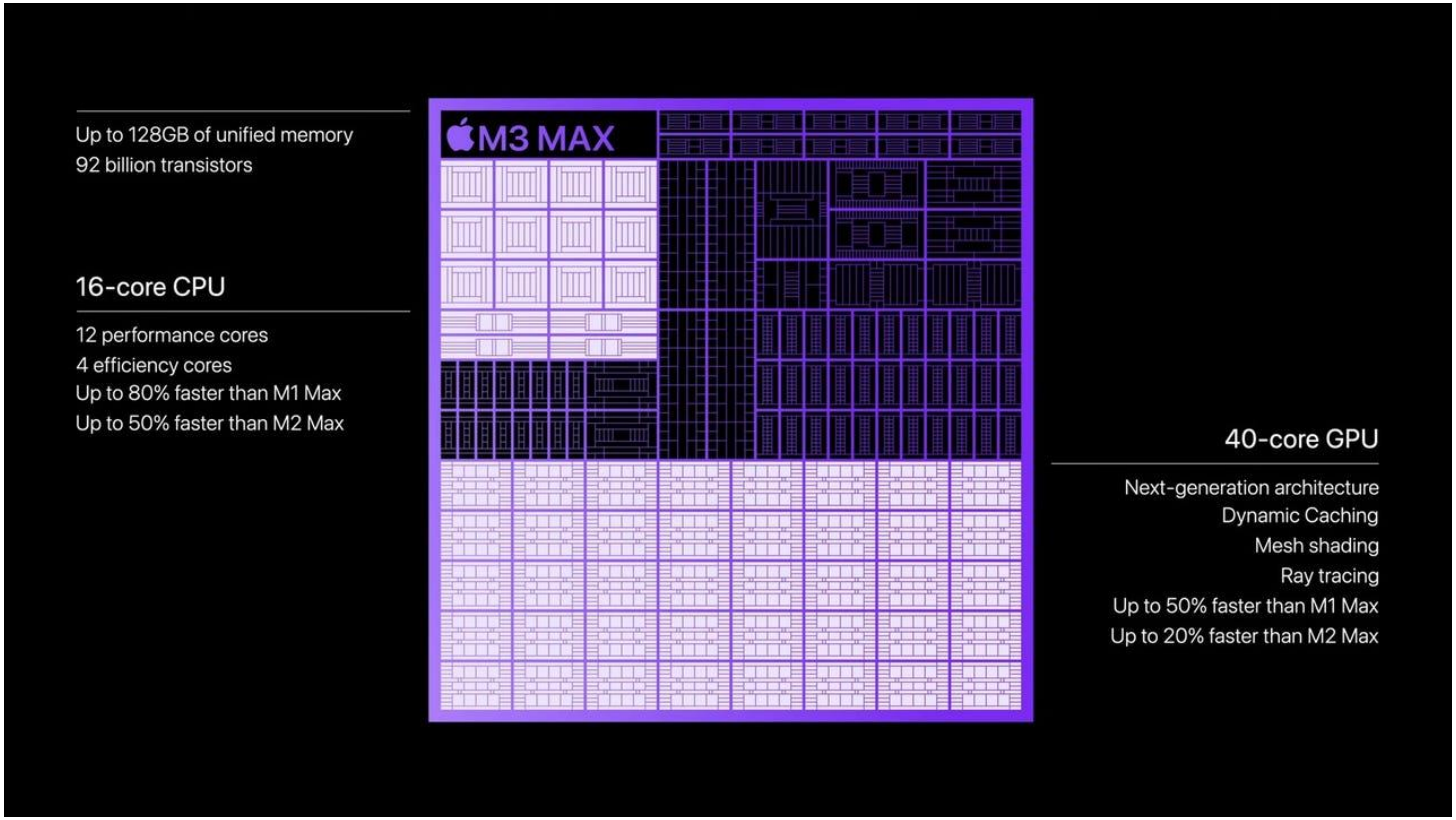
- **Field-Programmable Gate Array (FPGA):**
- Configurable for any class of programs; ~0.5-3 TFLOPS but very low power consumption
- Cheaper; new hardware-software integrated stacks for ML/DL

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

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

Practice: Let’s try reading the Latest Apple M3 Max



Up to 128GB of unified memory
92 billion transistors

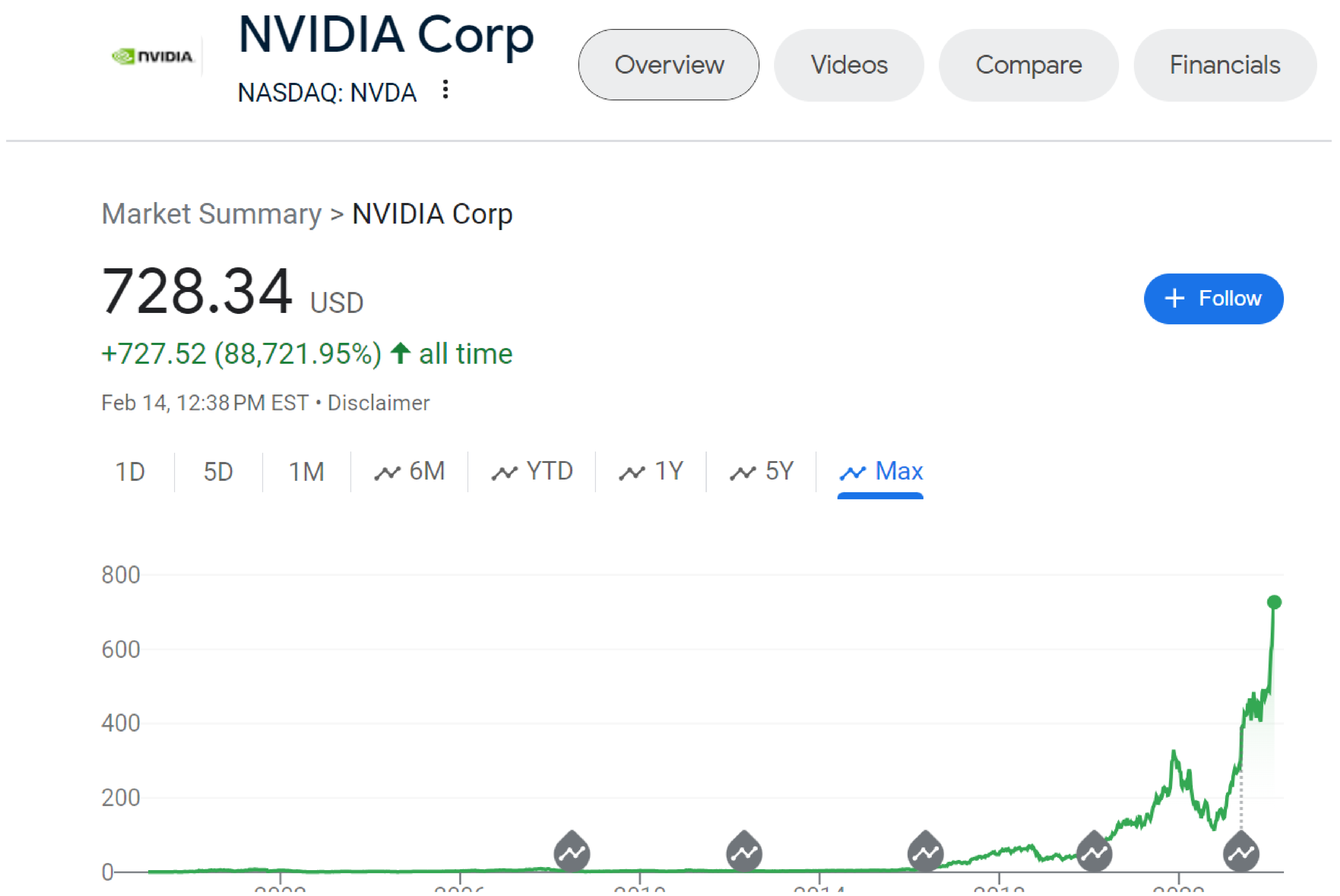
16-core CPU

12 performance cores
4 efficiency cores
Up to 80% faster than M1 Max
Up to 50% faster than M2 Max

40-core GPU

Next-generation architecture
Dynamic Caching
Mesh shading
Ray tracing
Up to 50% faster than M1 Max
Up to 20% faster than M2 Max

Specialized Hardware for DS/ML is a really good business


NVIDIA Corp

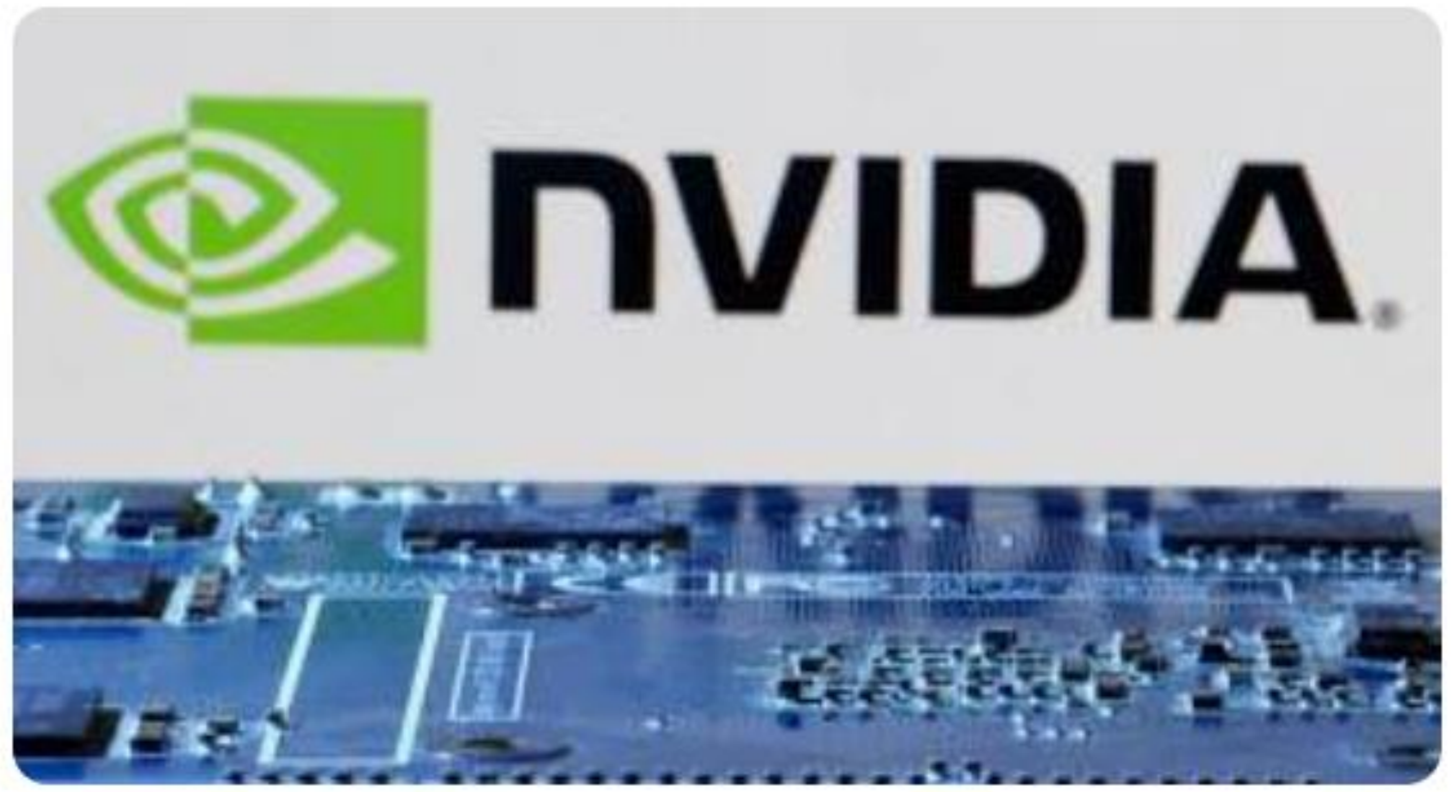
NASDAQ: NVDA

Market Summary > NVIDIA Corp

728.34 USD

+727.52 (88,721.95%) ↑ all time

Feb 14, 12:38 PM EST • Disclaimer



Reuters

Nvidia outstrips Alphabet as third largest US company by market value

1 hour ago

Emerging business (numbers are speculations)

cerebras

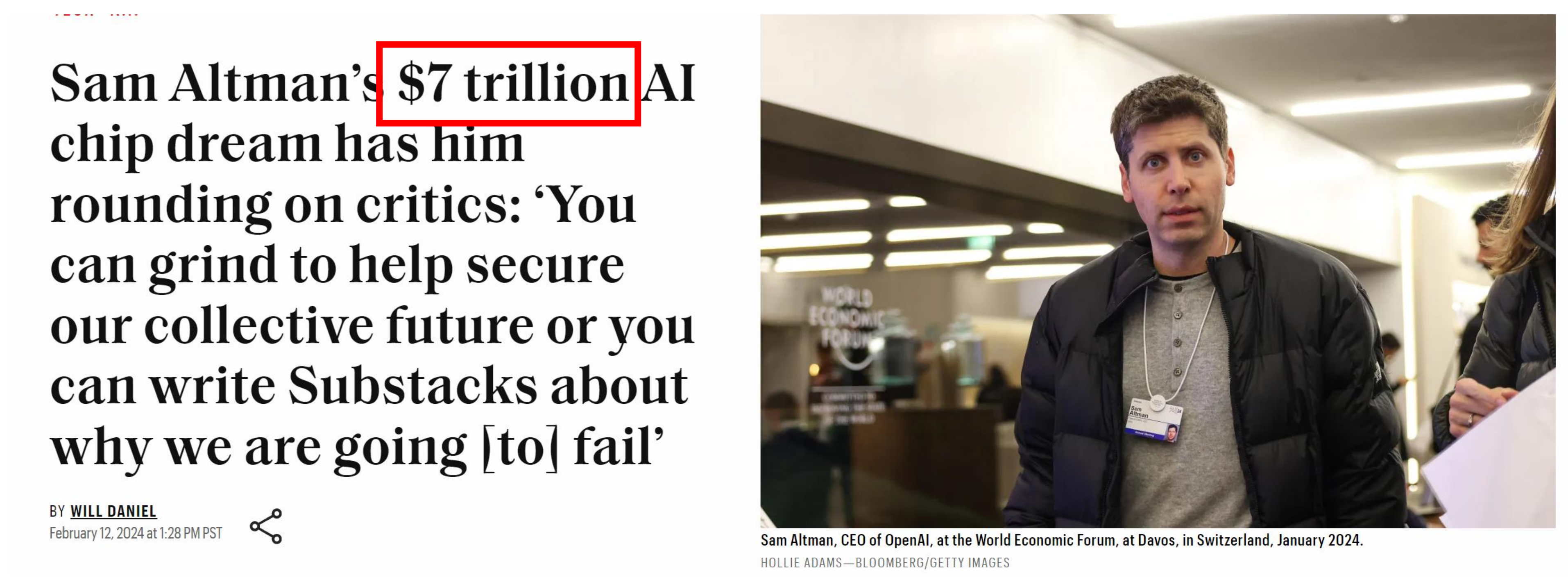
~4B

GRAPHCORE

~2.8B

groq™

~1B



Sam Altman's $7 trillion AI chip dream has him rounding on critics: 'You can grind to help secure our collective future or you can write Substacks about why we are going [to] fail'

BY WILL DANIEL
February 12, 2024 at 1:28 PM PST

Sam Altman, CEO of OpenAI, at the World Economic Forum, at Davos, in Switzerland, January 2024.
HOLLIE ADAMS—BLOOMBERG/GETTY IMAGES

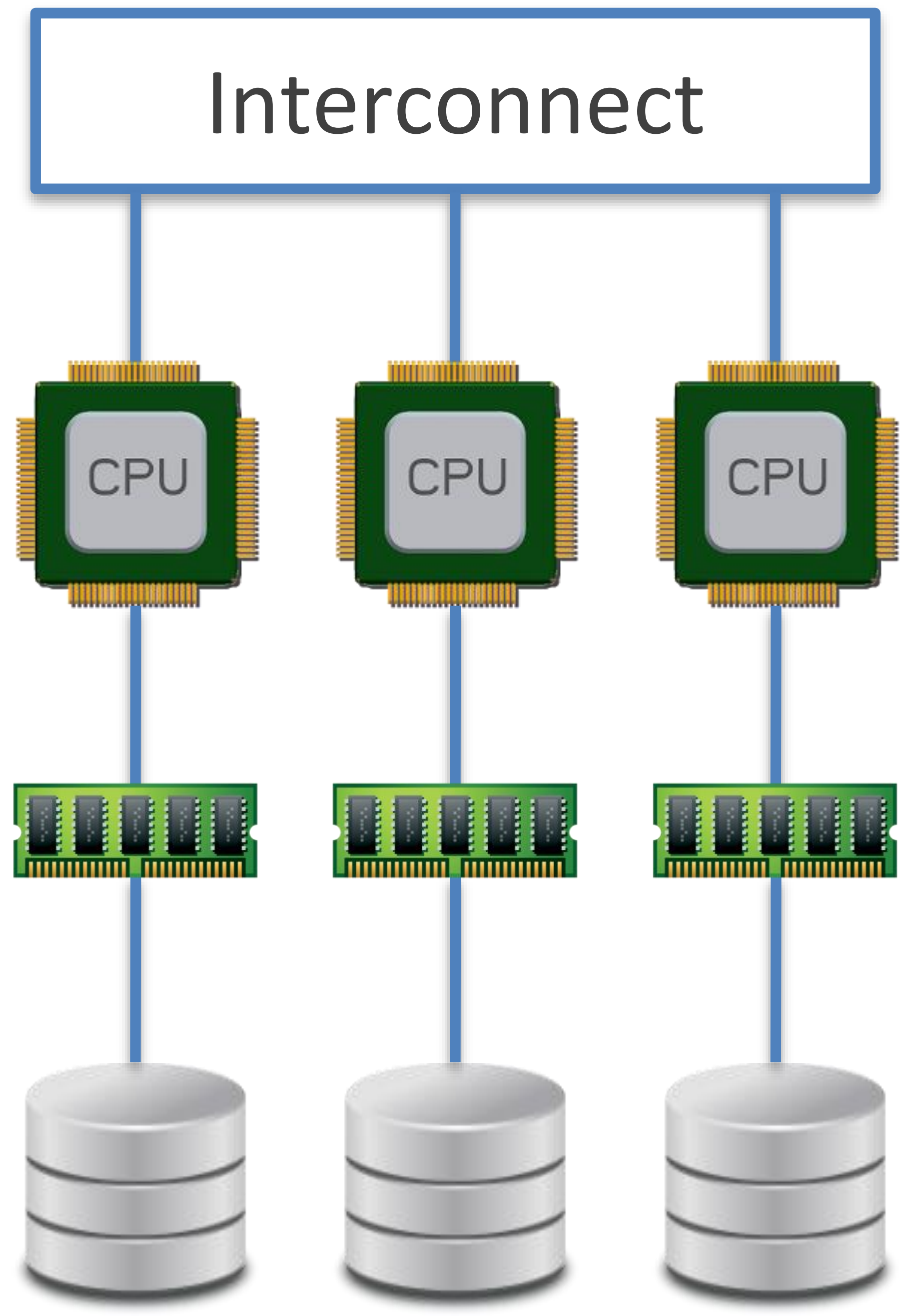
Summary

- Dataflow Graph
- Two major parallelisms:
  - Task parallelism -> partitioning the dataflow graph
  - Data parallel -> partitioning the data
- Data parallelism is ubiquitous, built in modern chips and everywhere.

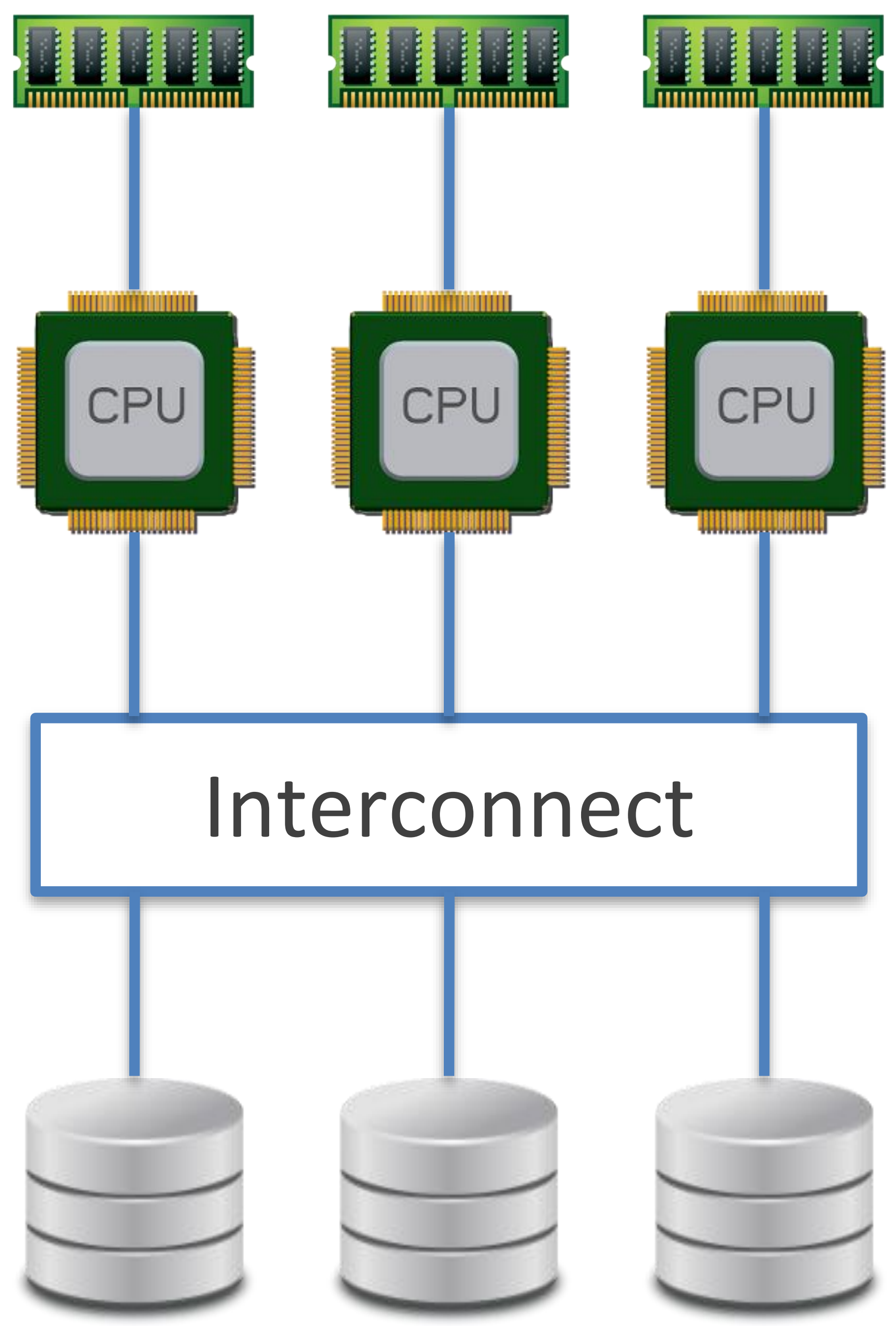
Let’s Focus On: Multi-node Distributed Systems

- Different ways of implementing parallelisms in distributed systems:
  - Shared-nothing parallelism
  - Shared-disk parallelism
  - Share-memory parallelism
- How to Distribute Data
- How to Coordinate
- How to Distribute Compute

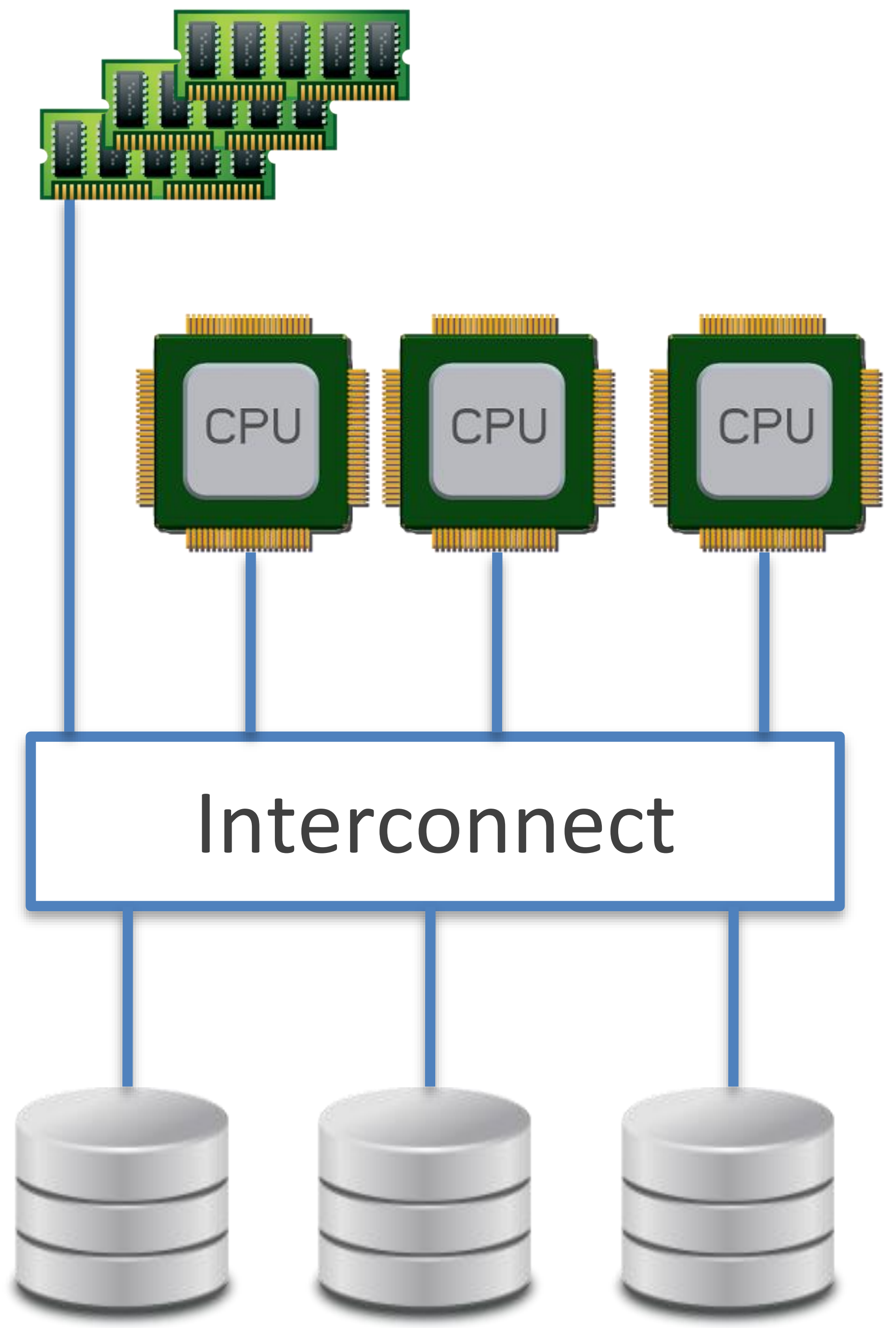
3 Paradigms of Multi-Node Parallelism Implementations



Shared-Nothing Parallelism



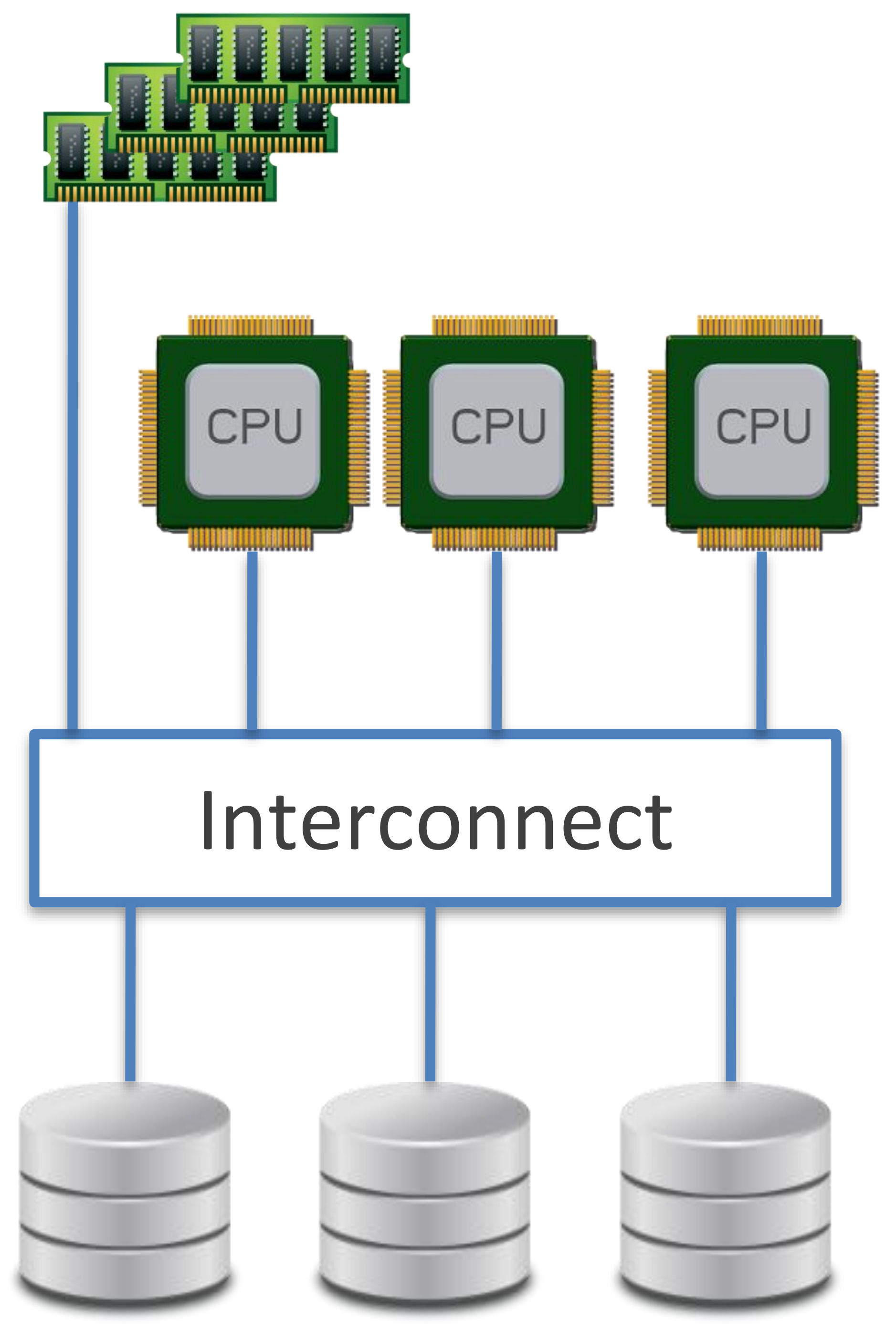
Shared-Disk Parallelism



Shared-Memory Parallelism

Shared-memory parallelism (vertical scaling)

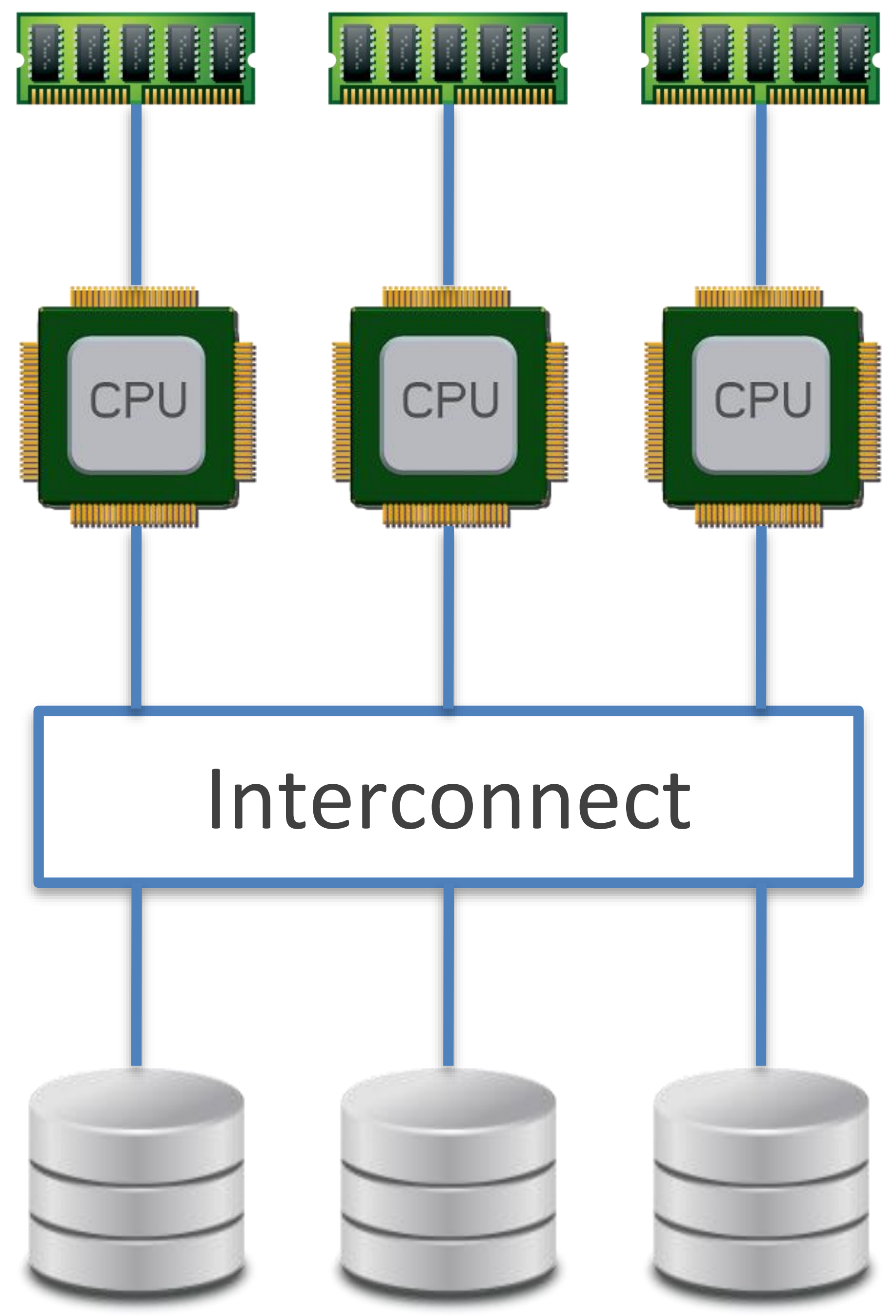
- CPUs, RAM chips, Disks are joined under one OS.
- Advantage:
  - Simple, High performance.
- Disadvantage:
  - Expensive.
    - Cost grows faster super-linearly.
    - Performance grows sub-linearly.
  - Limited fault tolerance
    - Note hot-swappable
  - Geolocation



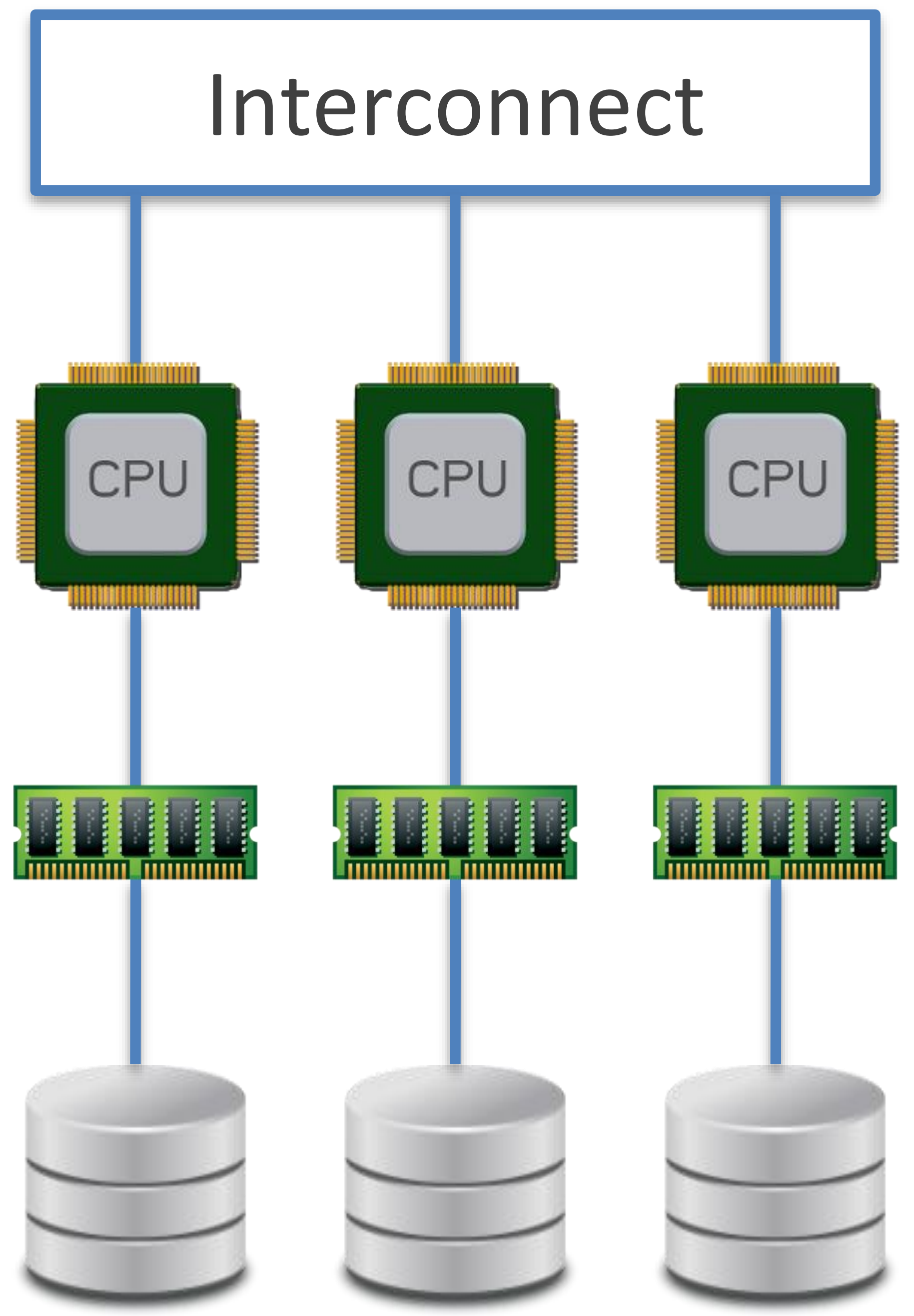
Shared-Memory Parallelism

Shared-Disk Parallelism (data warehouse)

- Machines with independent CPUs and RAM
  - But stores data on an array of disks
- Advantage:
  - Low-cost
- Disadvantage:
  - Overhead of locking limit the scalability

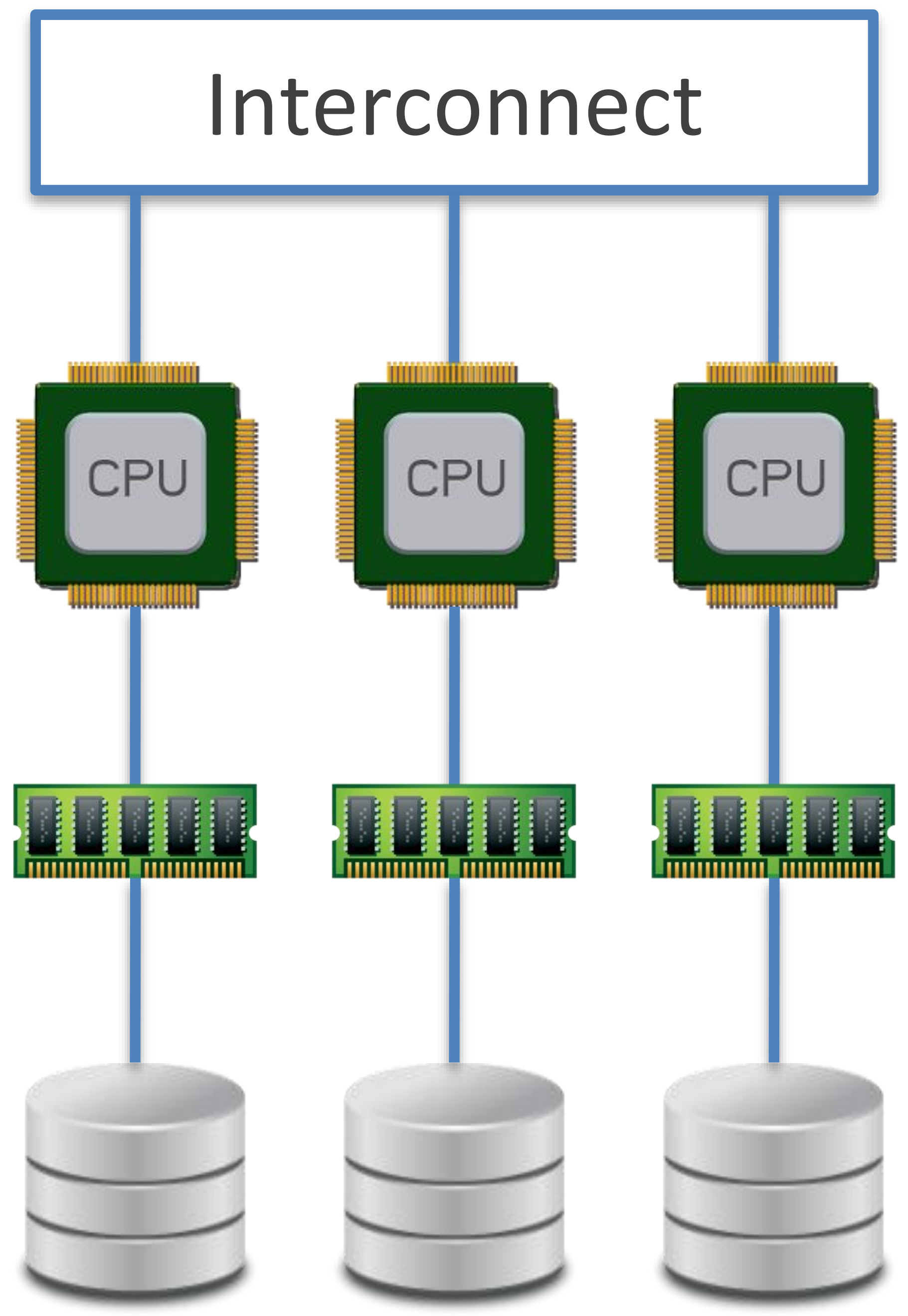


Shared-Nothing Parallelism (horizontal scaling)



- Most popular approach!
- Each node uses its CPUs, RAM, and disks independently.
  - Any coordination, software level, through a conventional network.
- The vanilla (and most complex!) distributed systems
  - Consistency
  - Communication
  - Coordination

Shared-Nothing Parallelism (horizontal scaling)



- Advantage:
  - Performance
  - Cost
- Disadvantage
  - Complexity.
  - Involves many constraints and trade-offs.
- Database cannot magically hide all these from you.