https://hao-ai-lab.github.io/dsc204a-f25/

DSC 204A: Scalable Data Systems
Fall 2025

Staff
Instructor: Hao Zhang
TAs: Mingjia Huo, Yuxuan Zhang

@haozhangml @haoailab
haozhang@ucsd.edu

Where We Are

Motivations, Economics, Ecosystems, Trends

Networking

Storage

Part3: Compute

~~Datacenter networking~~

~~Collective communication~~

(Distributed) File Systems / Database

Cloud storage

Distributed Computing

Big data processing

Parallelism

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

- **assuming there will be coordination:**

func \ data	**Shared**	**Replicated**	**Partitioned**
Replicated	N/A (rare cases)		Data parallelism
Partitioned	Task parallelism		Hybrid parallelism

Terms are confusing

- Different domains term them differently in different contexts
- Architecture/parallel computing: single-node multi-cores
  - SIMD, MIMD, SIMT
- Distributed system: multiple-node multi-cores
  - SPMD vs. MPMD
- Machine learning community
  - Data parallelism vs. Model parallelism
  - Inter-operator parallelism vs. Intra-operator parallelism

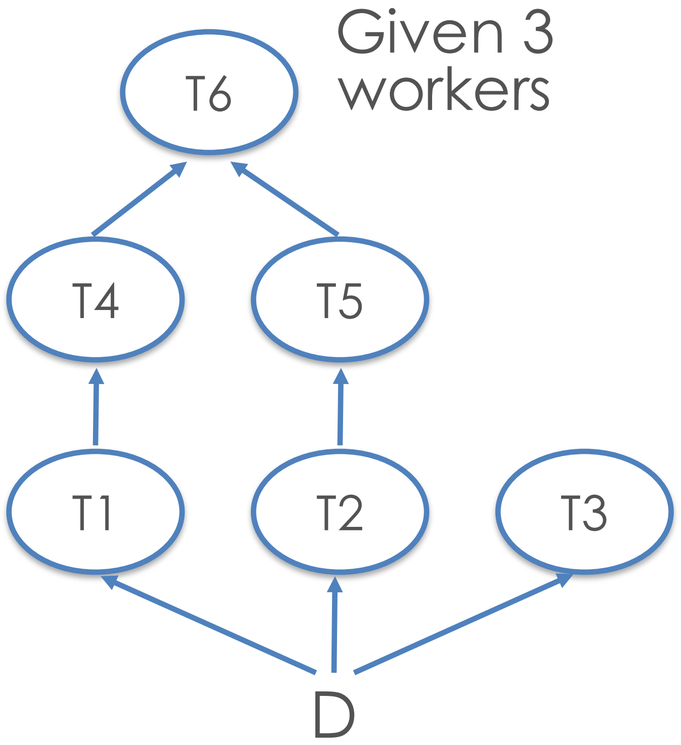
Today's topic: Parallelism

- Express data processing in abstraction
- Parallelisms
  - **Task parallelism**
  - Data parallelism
  - Terms: SIMD, SIMT, SPMD, MPMD

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:



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

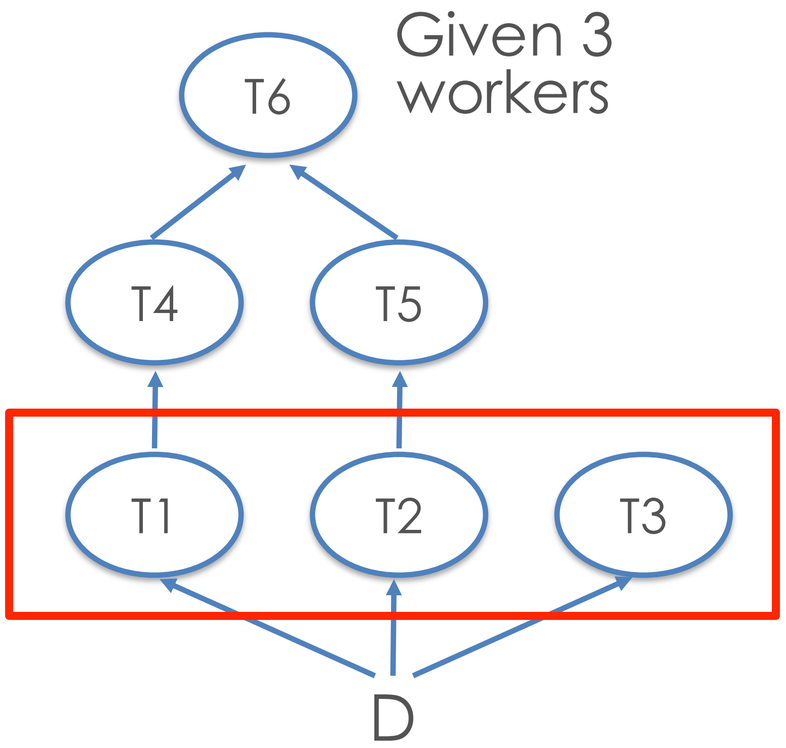
Task Parallelism

- 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., Ray: 4 worker nodes x 4 cores = 16 workers total
- Main pros of task parallelism:
  - Simple to understand
  - Independence of workers => low software complexity
- Main cons of task parallelism:
  - Can be difficult to implement
  - 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:



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

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

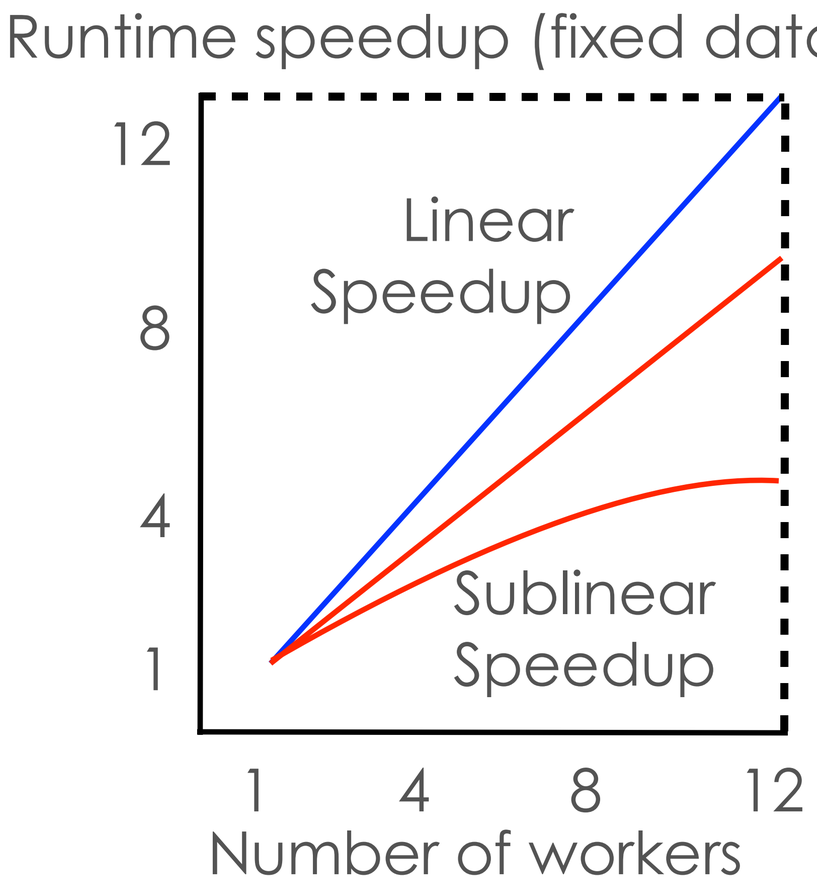
Q: *But given n workers, can we get a speedup of n?*

It depends!

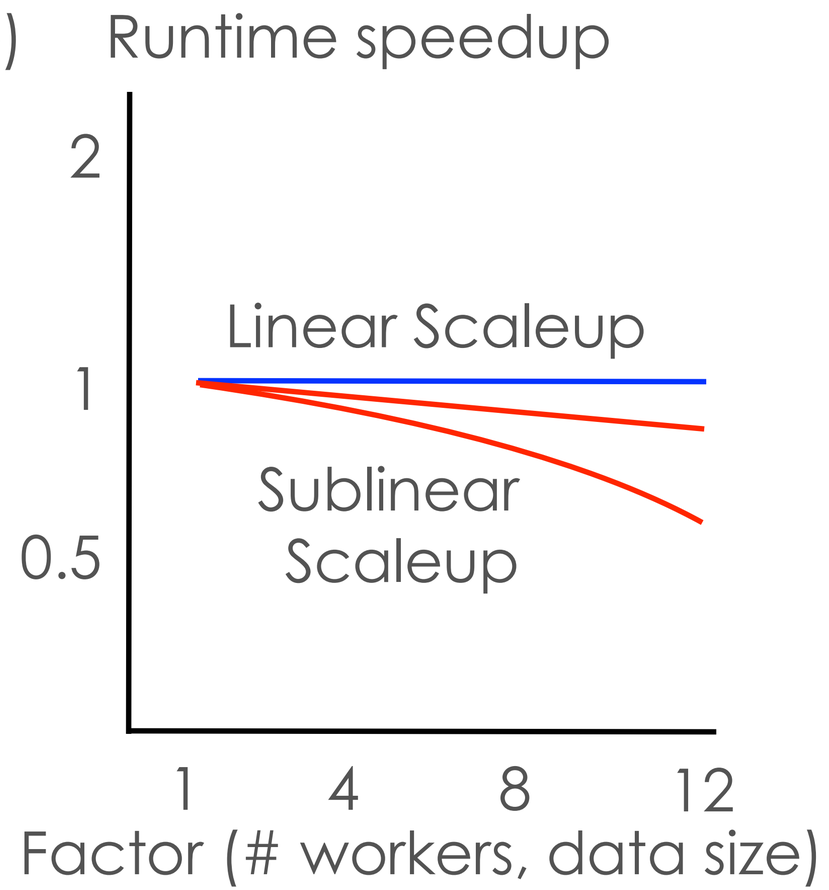
(On degree of parallelism, task dependency graph structure, intermediate data sizes, etc.)

Q: what kind of graphs can give a speedup of n?

Weak and Strong Scaling

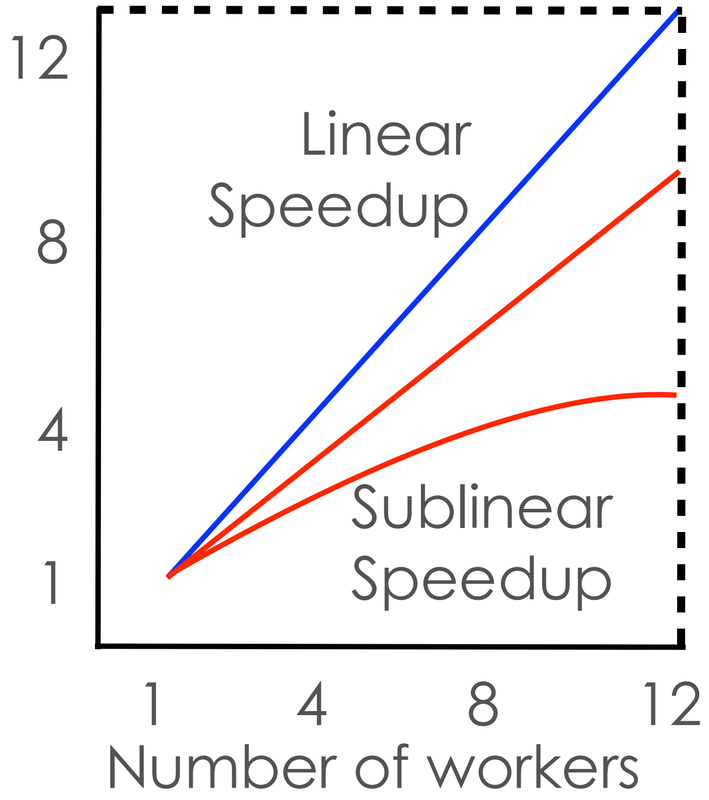


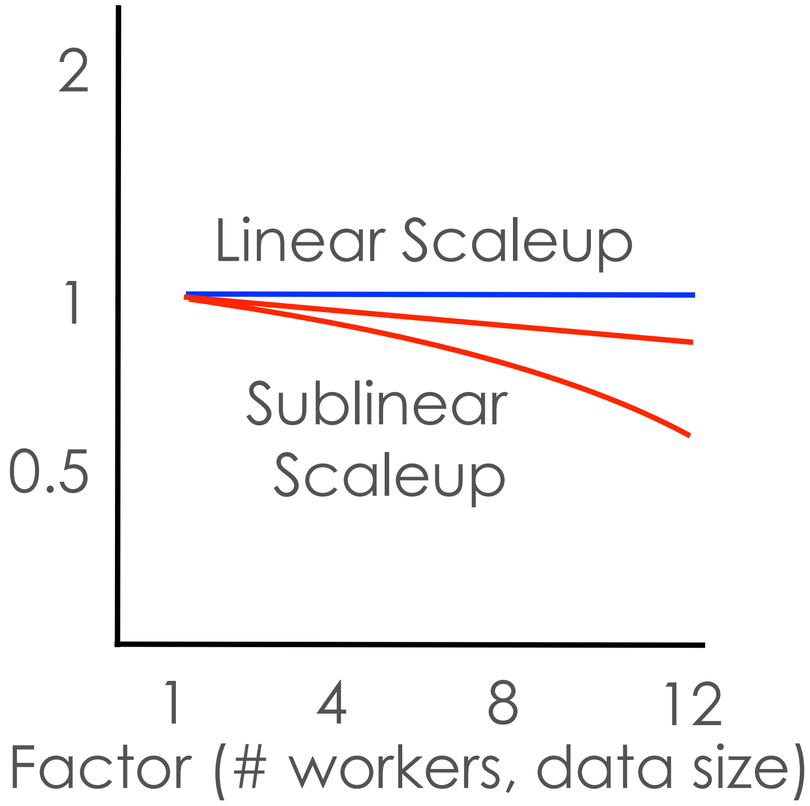
Speedup plot / Strong scaling



Scaleup plot / Weak scaling

Discussion: Is *superlinear* speedup/scaleup possible?





Some Clarifications on Terms

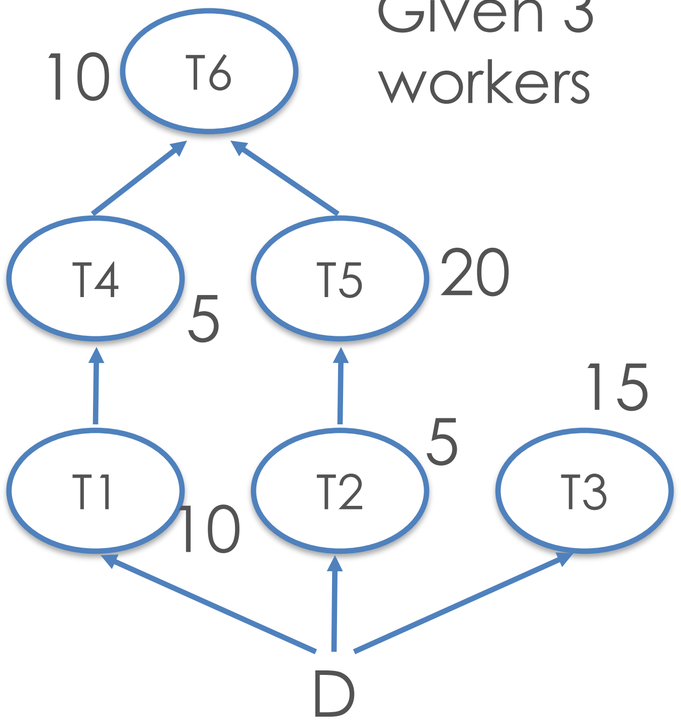
$$\textbf{Speedup} = \frac{\text{Completion time given only 1 worker}}{\text{Completion time given n (>1) workers}}$$

- 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

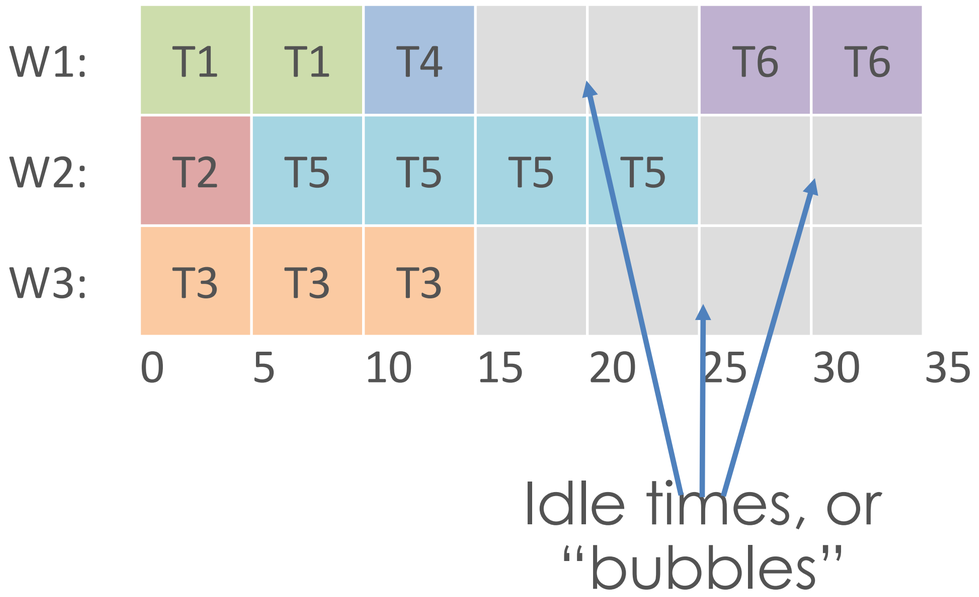
Idle Times in Task Parallelism

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

Example:



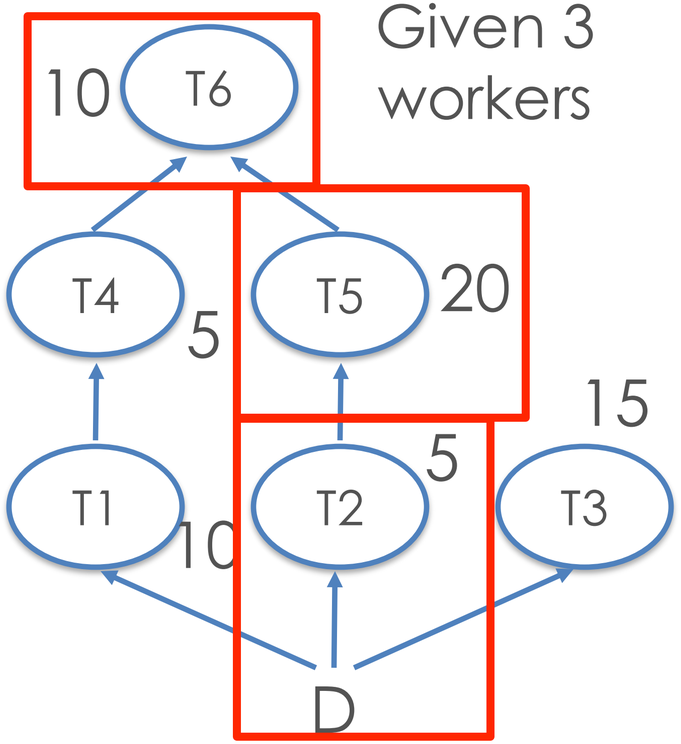
Gantt Chart visualization of schedule:



Idle Times in Task Parallelism

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

Example:

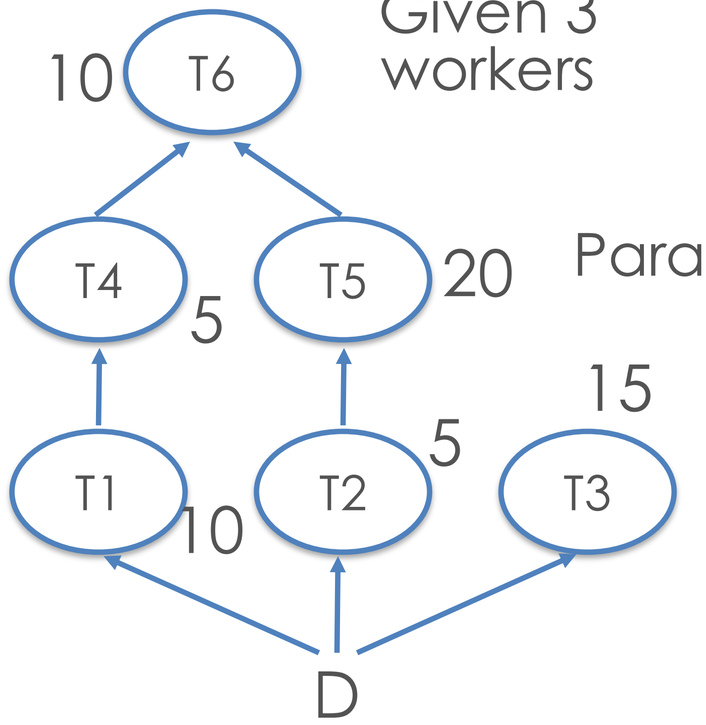


- In general, overall workload's completion time on task-parallel setup is always *lower bounded* by the **longest path** in the task graph
- Possibility: A task-parallel scheduler can "release" a worker if it knows that will be idle till the end
- Can saves costs in cloud

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

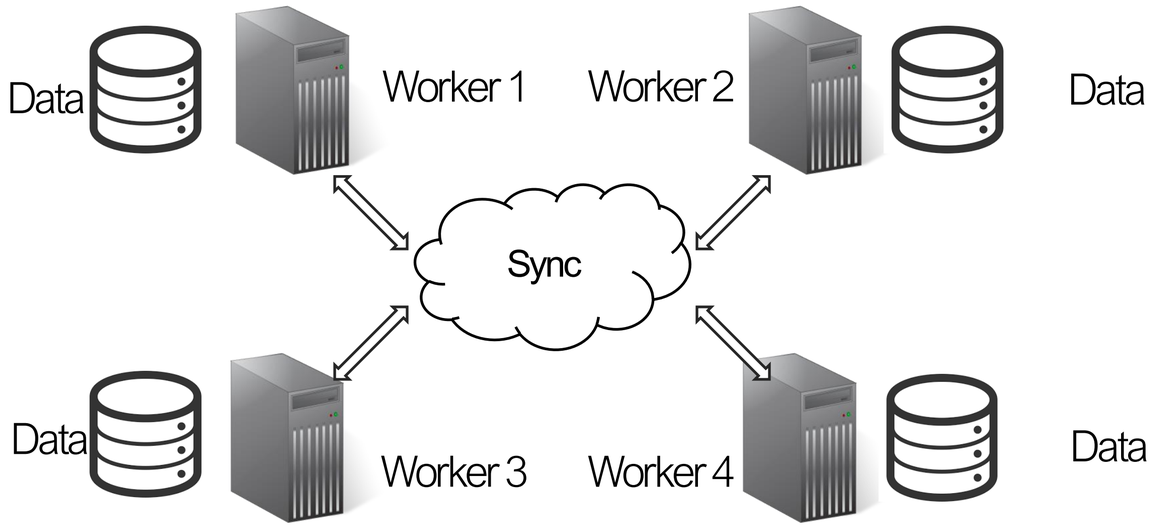
Ideal/linear speedup is 3x

Q: *Why is it only 1.9x?*

Today's topic: Parallelism

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

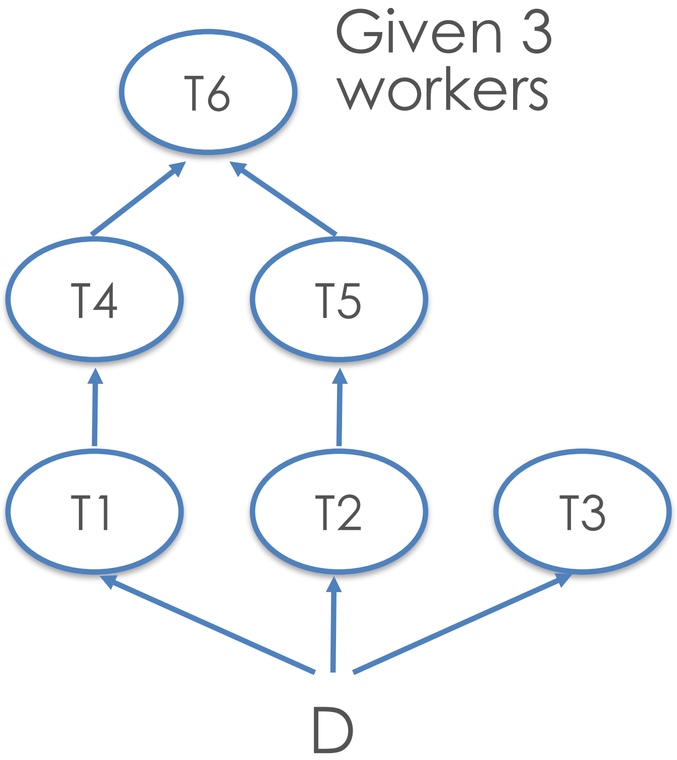
Recall: Data parallelism in ML

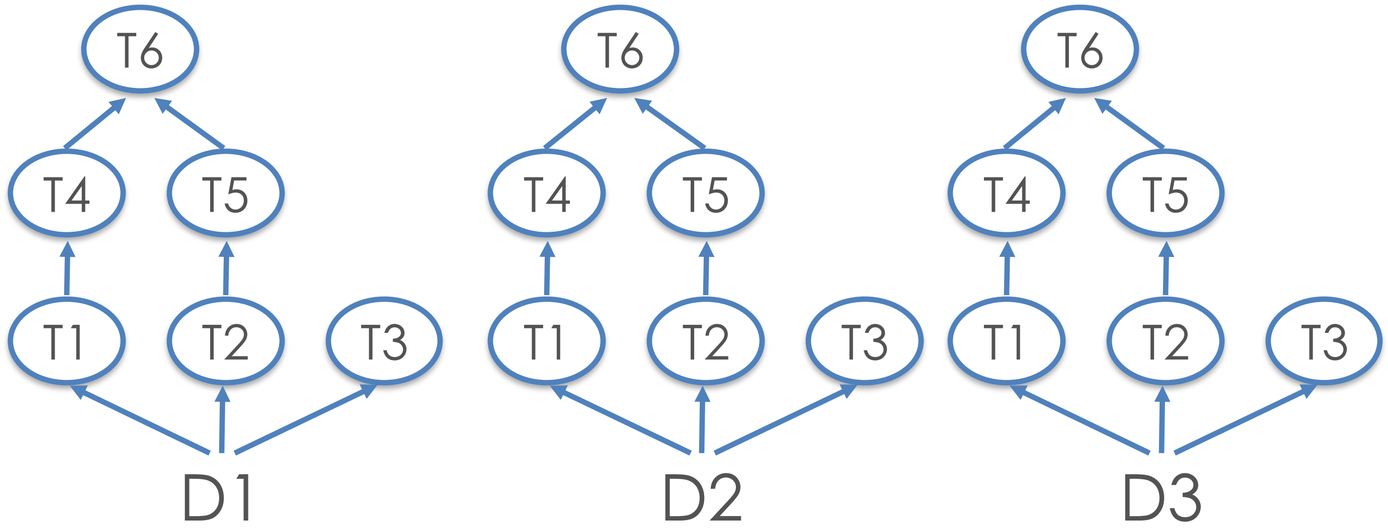


$$\theta^{(t+1)} = \theta^{(t)} + \varepsilon \sum_{p=1}^{P} \nabla_{\mathcal{L}}(\theta^{(t)}, D_p^{(t)})$$

Data parallelism: abstraction of SIMD/SIMT/SPMD

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





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:

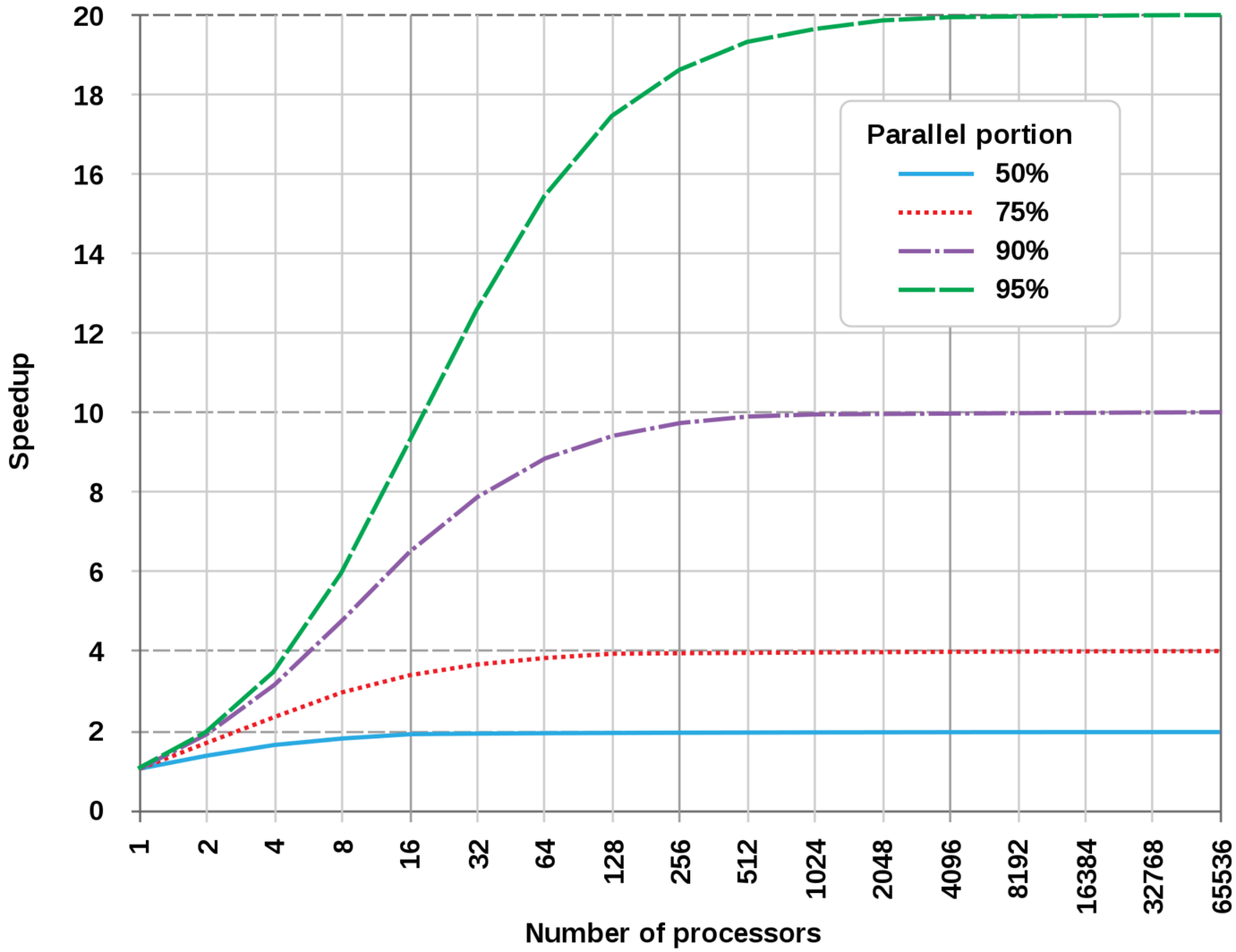
$T_{yes} \longrightarrow T_{yes}/n$

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

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

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

Amdahl's Law:



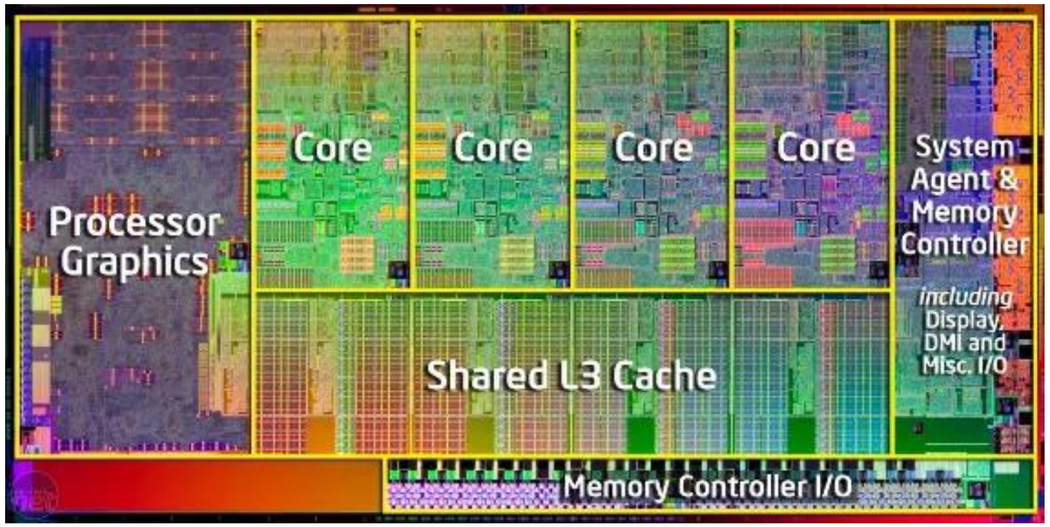
$f = T_{yes}/T_{no}$

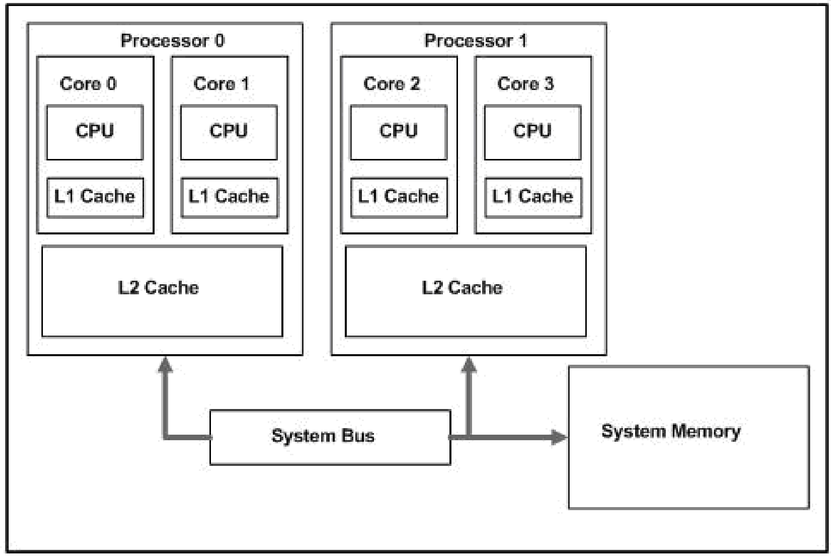
Parallel portion = $f / (1 + f)$

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

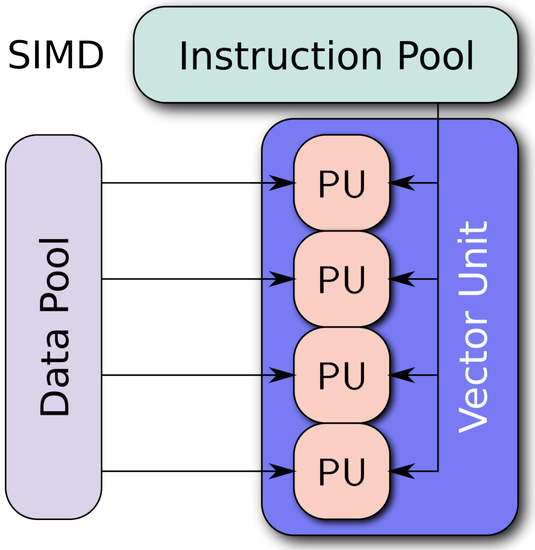
Data parallelism is built in with today's Processors

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

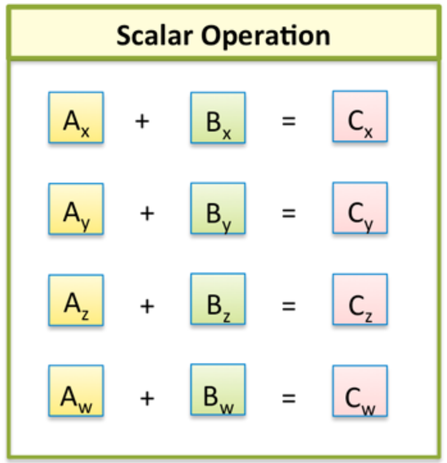


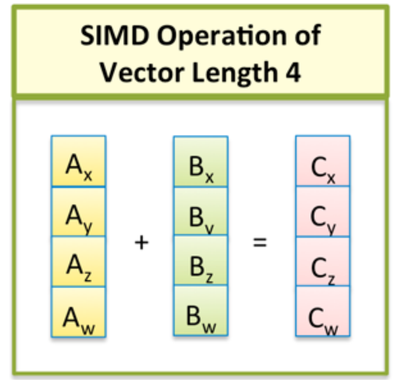


Single-Instruction Multiple-Data



Example for SIMD in data science:





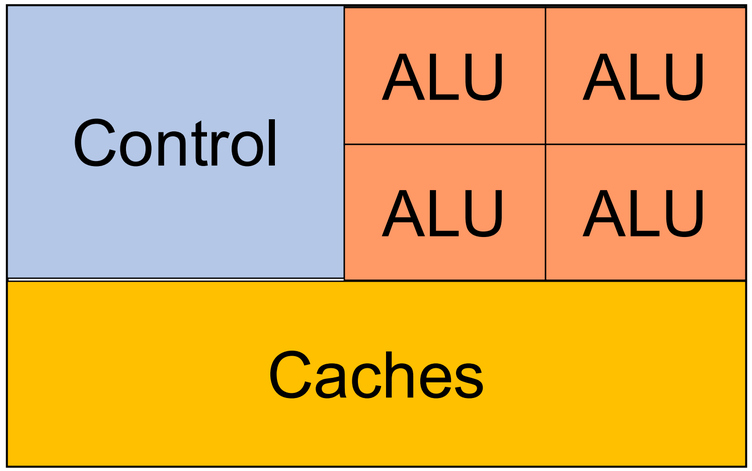
Intel® Architecture currently has SIMD operations of vector length 4, 8, 16

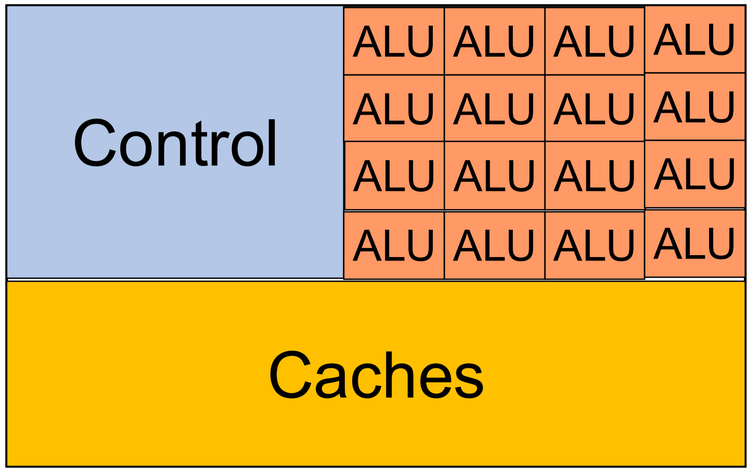
SIMD Generalizations

- 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

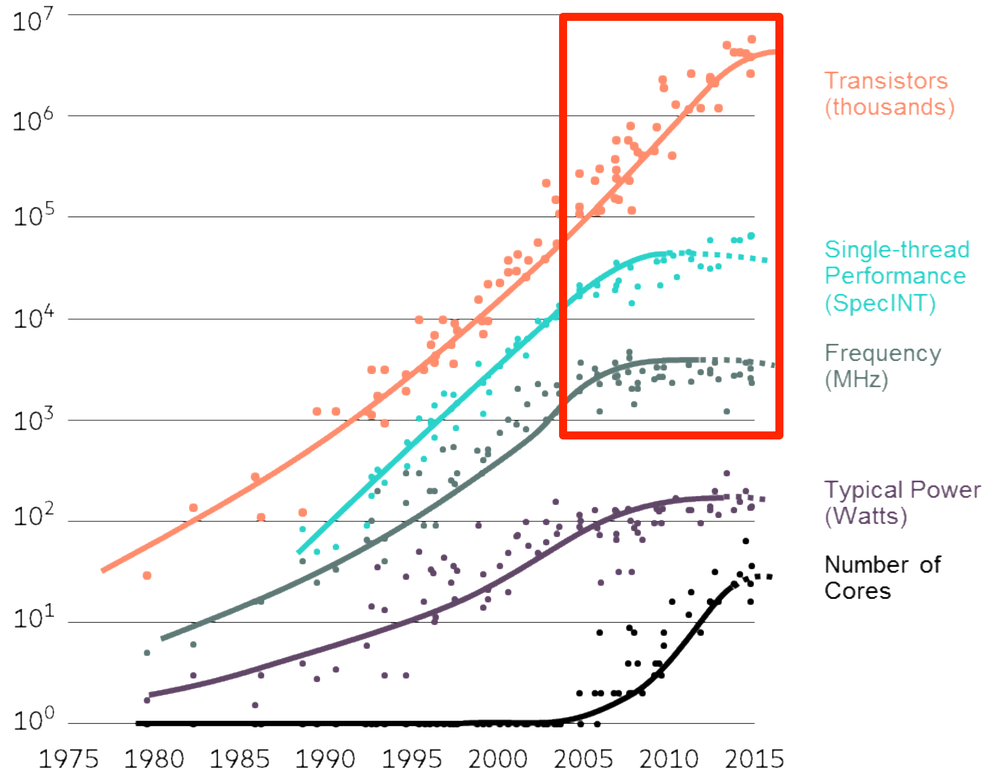
The Problem of Chip Design

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?

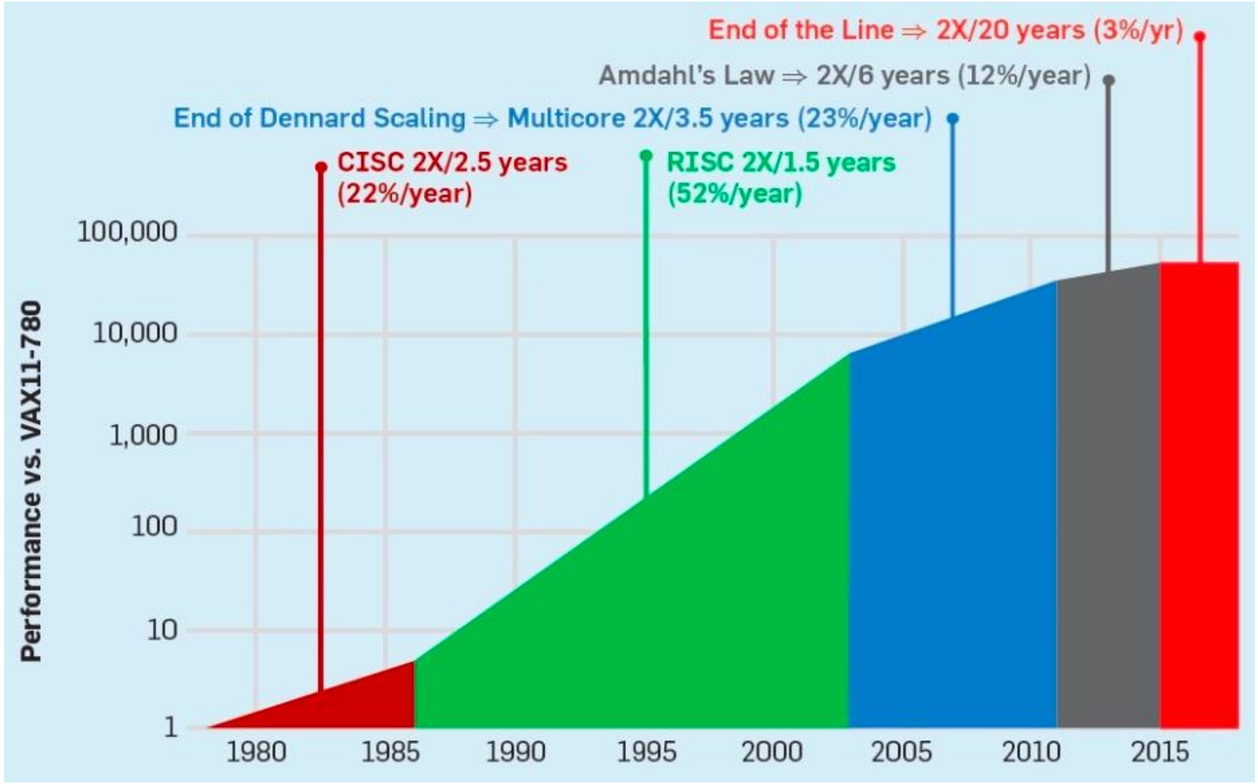


- That's why you see trends: 70nm -> 60nm -> 50nm -> … -> what is the best now?
- Takeaway from hardware trends: it is hard for general-purpose CPUs to sustain FLOPs-heavy programs like deep nets
- Motivated the rise of "accelerators" for some classes of programs

Chip Industry: 70nm -> 60nm -> 50nm -> … ->?

- Problem: this is not substantiable; there are also power/heat issues when you put more ALUs in a limited area (s.t. physics limitations)

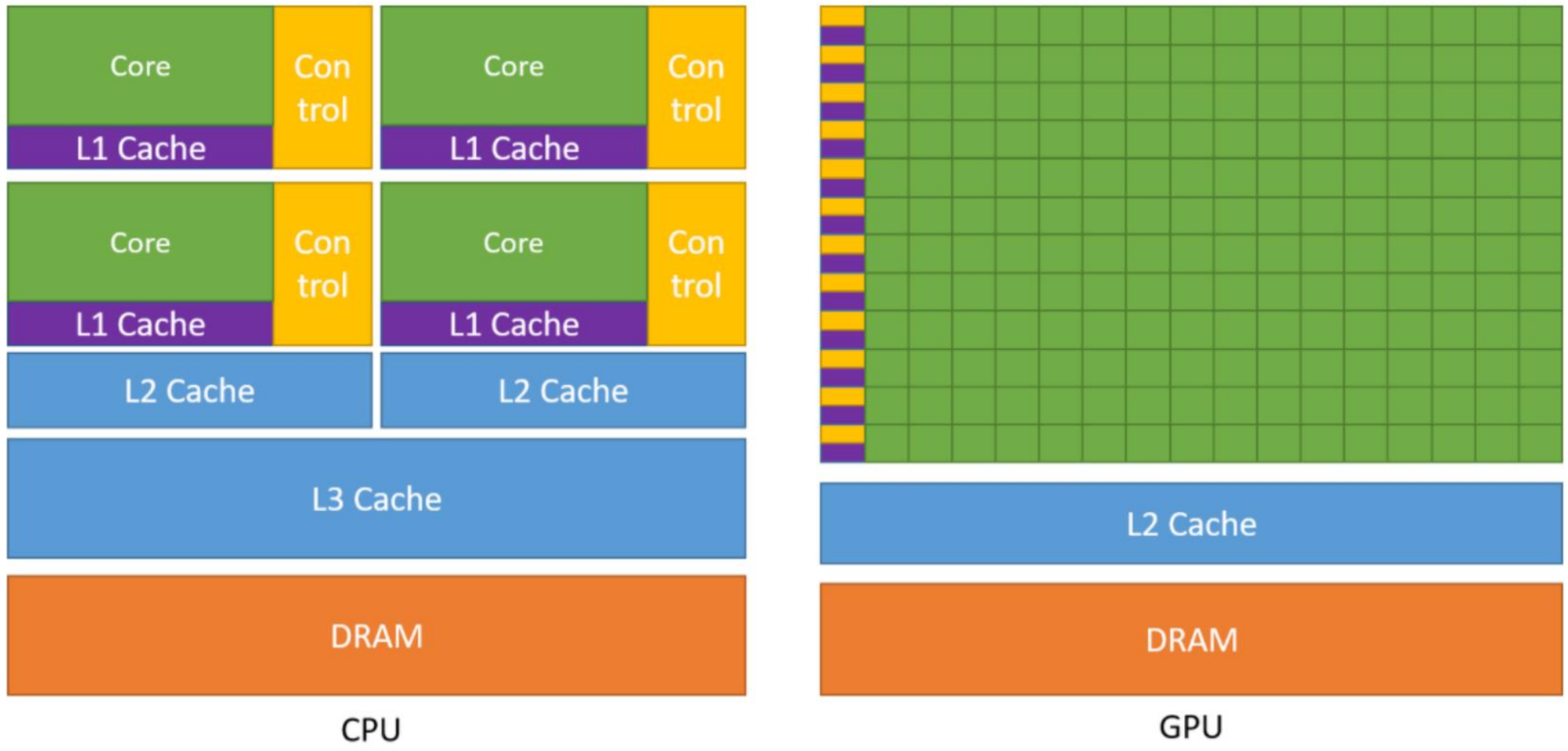
Chip Industry: Moore's Law Comes to an End



Option 1: Go to the quantum world

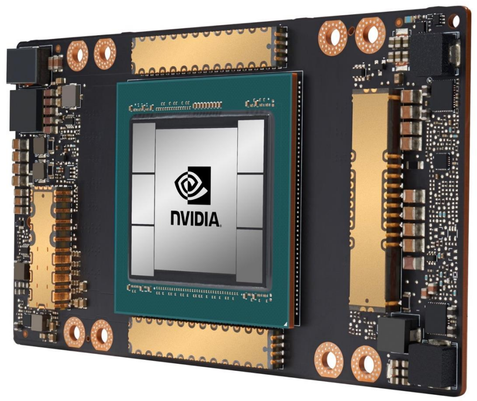
Option 2: Specialized hardware

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

- E.g.
  - Tensor Processing Unit (TPU)
  - An “application-specific integrated circuit” (ASIC) created by Google in mid 2010s; used for AlphaGo
- E.g.
  - B200 (projected release 2025): fp4 / fp8 Tensorcore
- E.g.
  - M3 max: mixing tensorcore and normal core

What Does It Mean by “Specialized” In accelerator world

In General:

- Functionality-specialized:
  - Can only compute certain computations: matmul, w/ sparsity
  - Mixing specialized cores with versatile cores
- Reduce precision
  - Floating point operations: fp32, fp16, fp8, int8, int4, …
- Tune the distribution of different components for specific workloads
  - SRAM, cache, registers, etc.

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

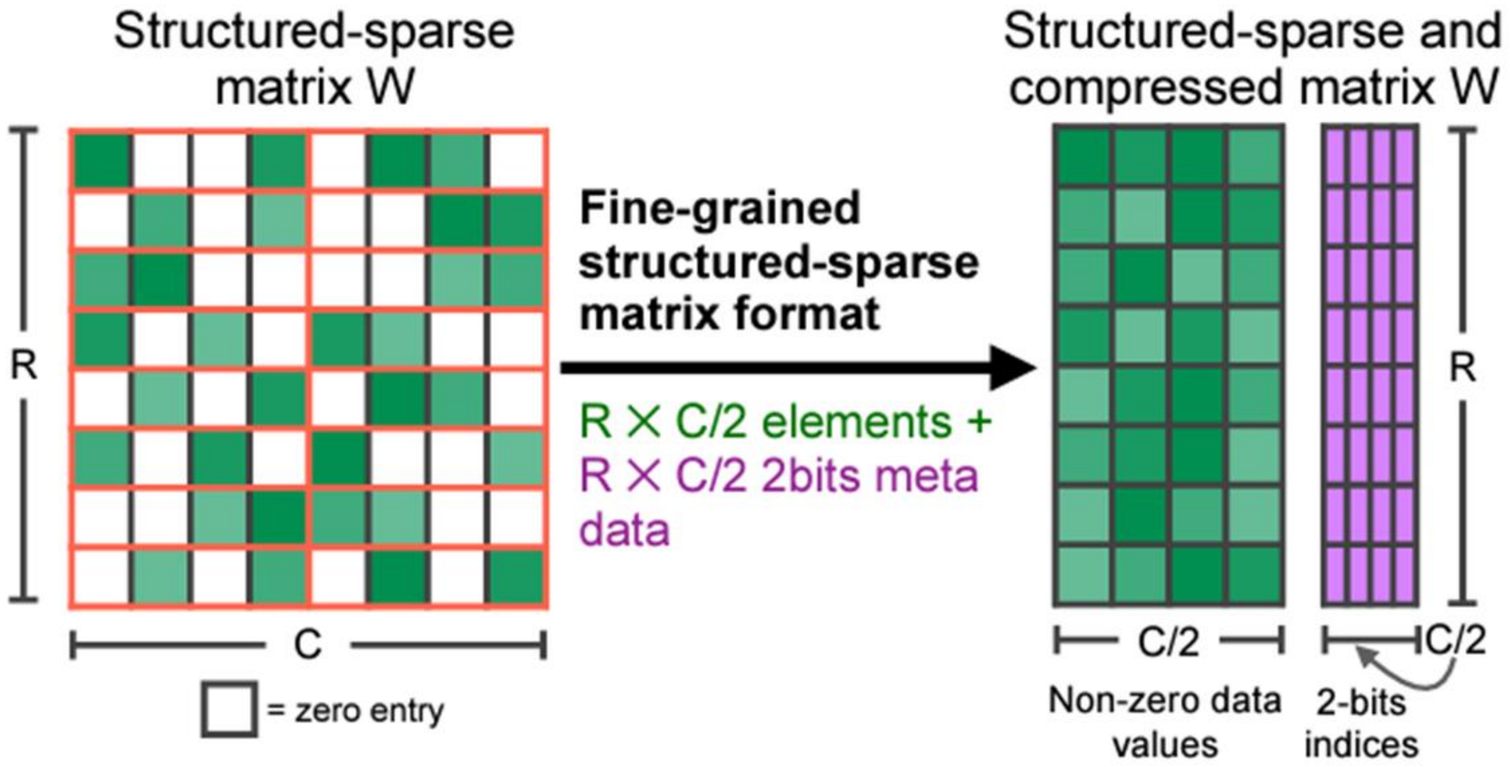
https://www.embedded.com/leveraging-fpgas-for-deep-learning/

Case Study 1: Nvidia GPU Specification

FP64 Tensor Core	67 teraFLOPS
FP32	67 teraFLOPS
TF32 Tensor Core*	989 teraFLOPS
BFLOAT16 Tensor Core*	1,979 teraFLOPS
FP16 Tensor Core*	1,979 teraFLOPS
FP8 Tensor Core*	3,958 teraFLOPS
INT8 Tensor Core*	3,958 TOPS
GPU Memory	80GB
GPU Memory Bandwidth	3.35TB/s
Decoders	7 NVDEC 7 JPEG
Max Thermal Design Power (TDP)	Up to 700W (configurable)
Multi-Instance GPUs	Up to 7 MIGS @ 10GB each
Form Factor	SXM
Interconnect	NVIDIA NVLink™: 900GB/s PCIe Gen5: 128GB/s
Server Options	NVIDIA HGX H100 Partner and NVIDIA-Certified Systems™ with 4 or 8 GPUs NVIDIA DGX H100 with 8 GPUs
NVIDIA AI Enterprise	Add-on

* With sparsity

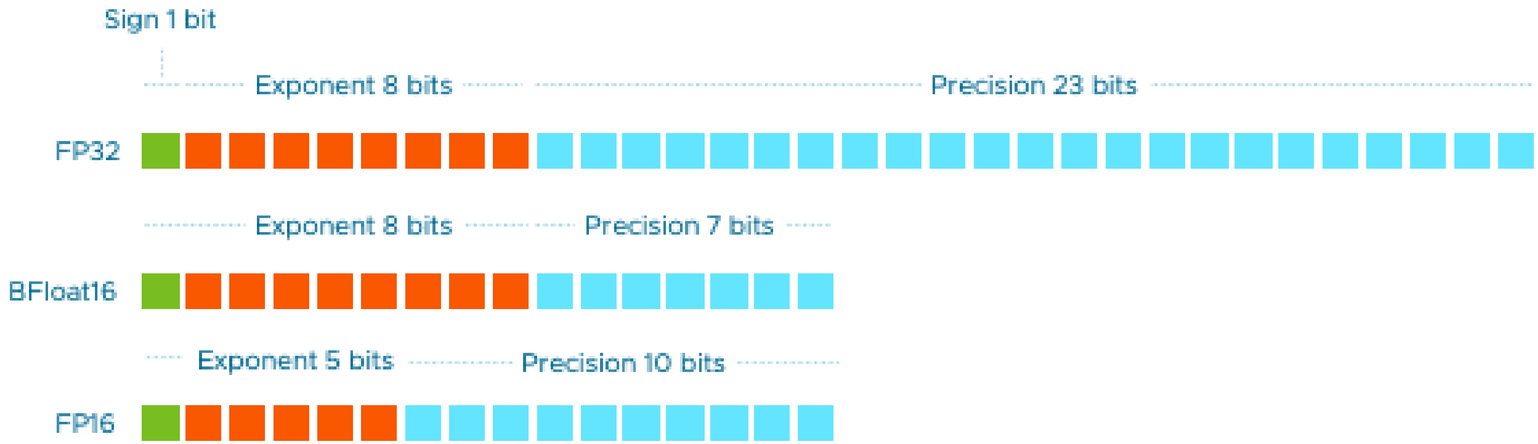
Case Study 1: Nvidia GPU Specification



Case Study 1: Nvidia GPU Specification

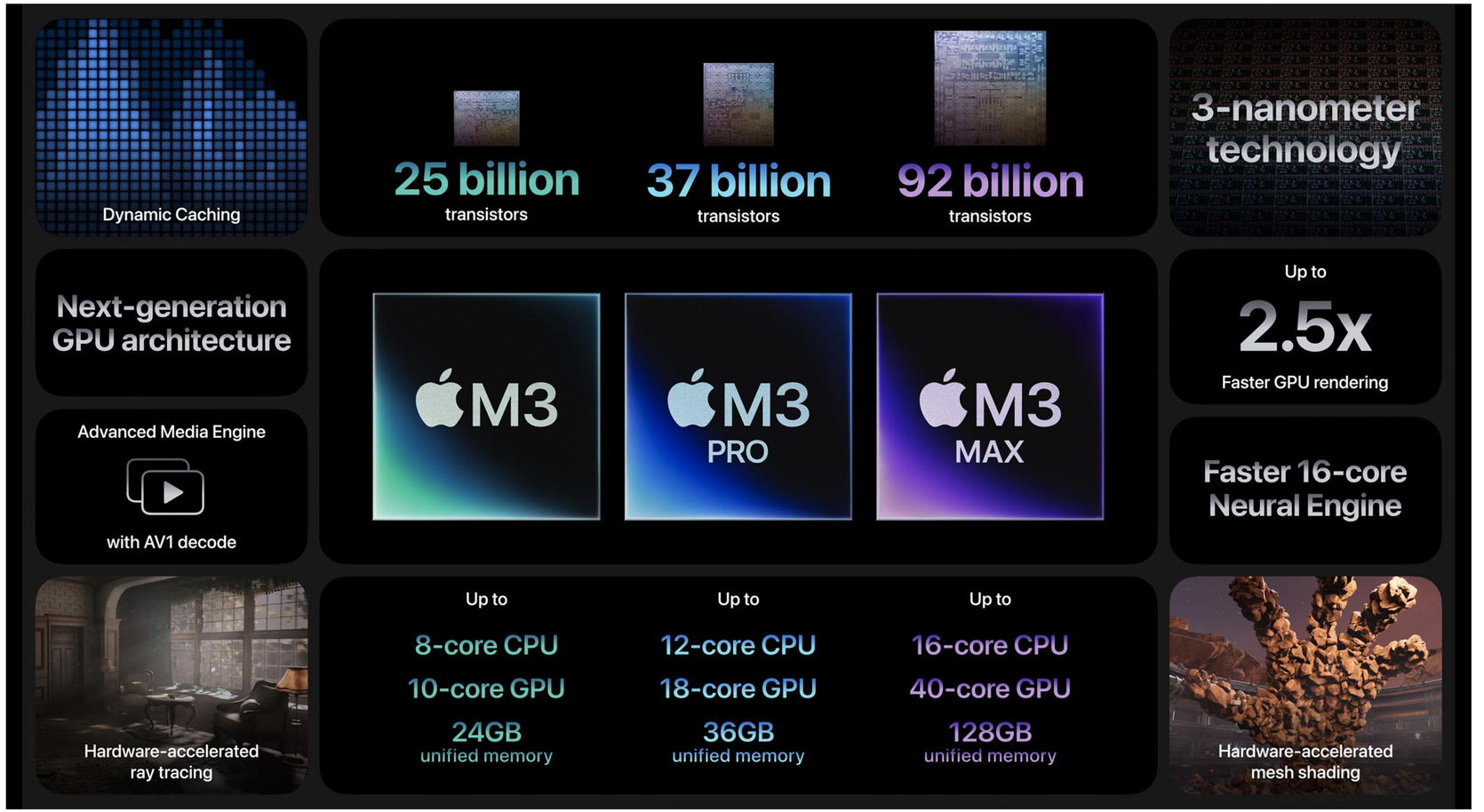
FP64 Tensor Core	67 teraFLOPS
FP32	67 teraFLOPS
TF32 Tensor Core*	989 teraFLOPS
BFLOAT16 Tensor Core*	1,979 teraFLOPS
FP16 Tensor Core*	1,979 teraFLOPS
FP8 Tensor Core*	3,958 teraFLOPS
INT8 Tensor Core*	3,958 TOPS
GPU Memory	80GB
GPU Memory Bandwidth	3.35TB/s
Decoders	7 NVDEC 7 JPEG
Max Thermal Design Power (TDP)	Up to 700W (configurable)
Multi-Instance GPUs	Up to 7 MIGS @ 10GB each
Form Factor	SXM
Interconnect	NVIDIA NVLink™: 900GB/s PCIe Gen5: 128GB/s
Server Options	NVIDIA HGX H100 Partner and NVIDIA-Certified Systems™ with 4 or 8 GPUs NVIDIA DGX H100 with 8 GPUs
NVIDIA AI Enterprise	Add-on

* With sparsity

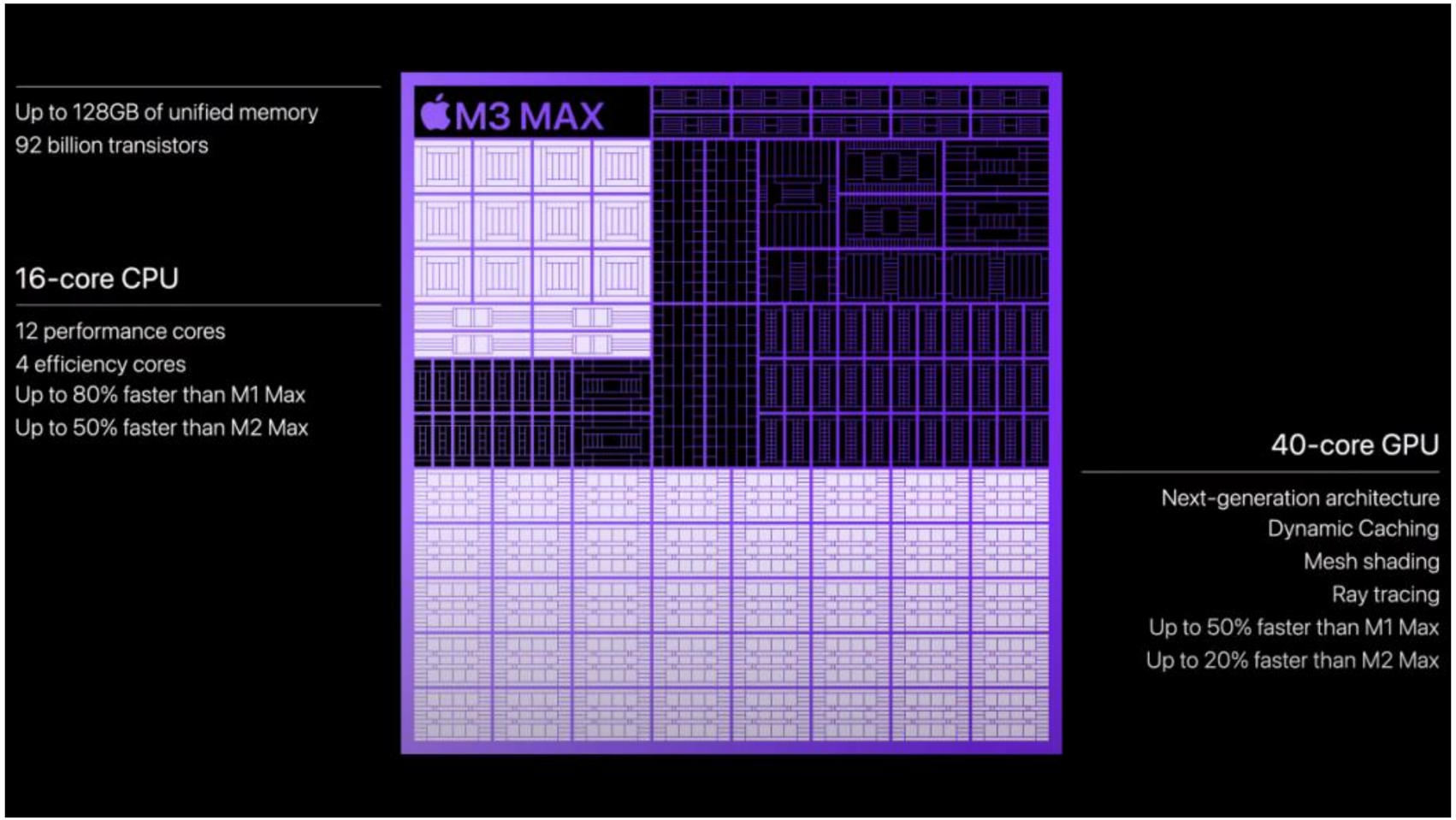


Question: why this could work in ML programs?

Case Study 2: Apple Silicon



Case Study 2: Apple Silicon Revealed



Specialized Hardware for DS/ML is a really good business

Market Summary > NVIDIA Corp

180.28 USD

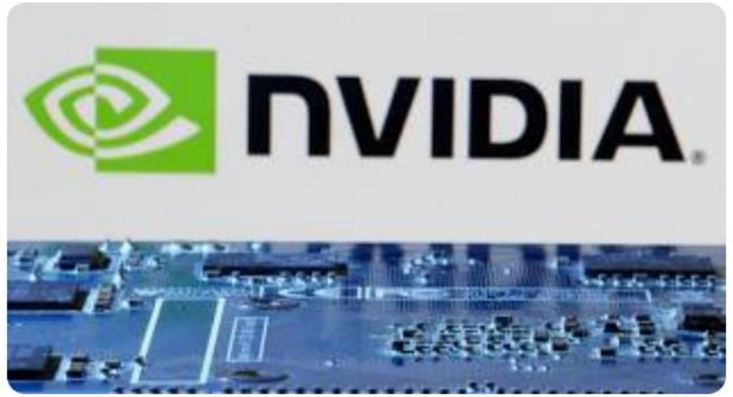
+180.24 (450,600.00%) ↑ all time

Closed: Oct 22, 7:59 PM EDT • Disclaimer

After hours 179.70 −0.58 (0.32%)

1D | 5D | 1M | 6M | YTD | 1Y | 5Y | Max

Open	181.14	Mkt cap	4.38T	52-wk high	195.62
High	183.44	P/E ratio	51.31	52-wk low	86.63
Low	176.76	Div yield	0.022%	Qtrly Div Amt	0.010



Reuters

Nvidia outstrips Alphabet as third largest US company by market value

1 hour ago

Case Study 3: Leading Chip Startups

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.

Take-home Exercise

- Study B100 specification and compare it to H100
  - How nvidia claims another 2x from H100 -> B100?
- Study Apple M5 and compare it to M3

Let's Focus On: Multi-node Distributed Systems

We have two primary problems to solve in real systems

1. How to Distribute Data (we'll not cover, also not in exam)
   - Read DDIA
   - Read GFS paper
   - Read BigTable paper
2. How to Distribute Compute (we'll cover in lectures)
   - Batching Processing
   - Streaming Processing

Metric Aspects of Distributed Big Data Systems

- Scalability
  - Data volume
  - Read/Write/Compute load

- Consistency and correctness
  - Read / Write sees consistency data
  - Compute produce correct results

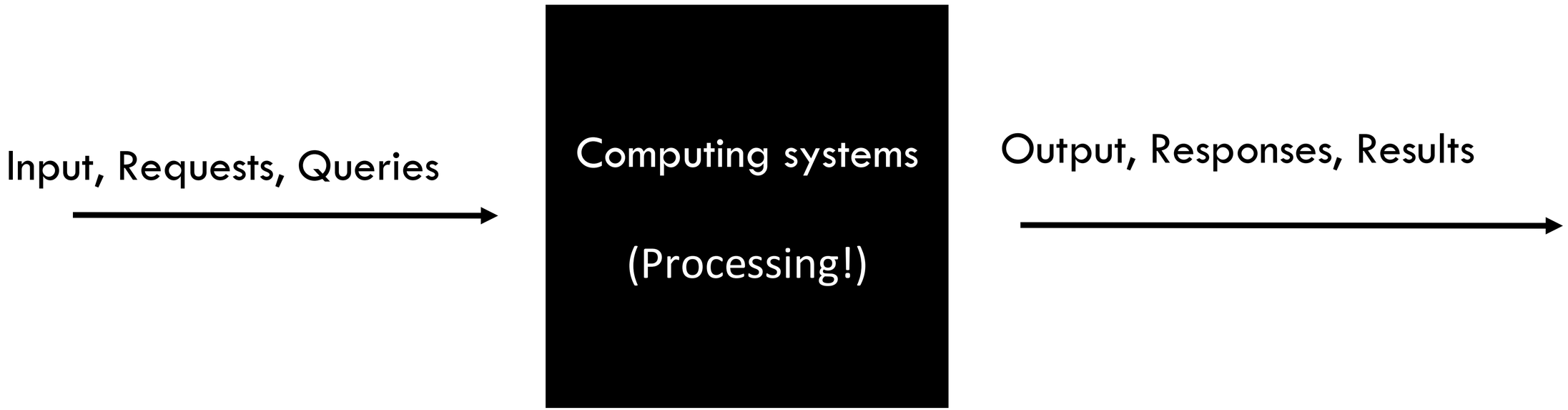
- Fault tolerance / high availability
  - When one fails, another can take over.

- Latency
  - Distribute machines worldwide.
  - Reduce network latency.

Today's topic: Batch Processing

- Overview
- IO & Unix pipes
- MapReduce
- Beyond MapReduce

Basic Computing System Paradigm



Processing latency

feedback cycle time

direct manipulation
no visible lag

Interactive data science!

Online system: handle request ASAP

turn-taking
minutes to seconds

Stream processing systems
(near-real-time systems)

batch-processing
hours or overnight

Batch processing systems
(Offline systems)

Figure from https://publishup.uni-potsdam.de/opus4-ubp/frontdoor/deliver/index/docId/10090/file/mueller_diss.pdf

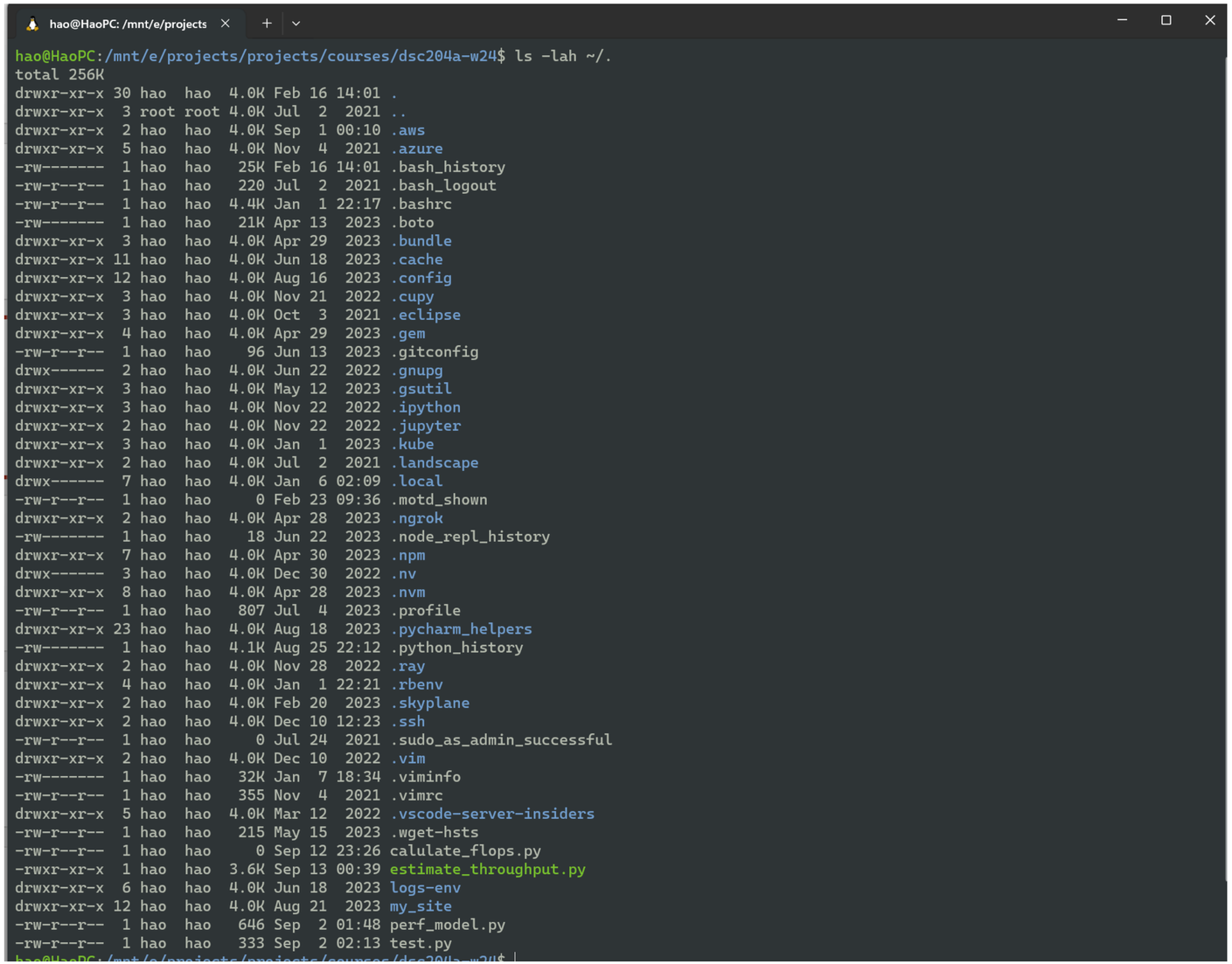
Today's topic: Batch Processing

- Overview
- IO & Unix pipes: the first batching processing system
- MapReduce
- Beyond MapReduce

Shell example

“.****rc”

Run commands



Useful shell commands

- Shell already has a collection of rich commands
  - Some Useful commands
    - uptime, cut, date, cat, finger, hexdump, man, md5sum, quota,
    - mkdir, rmdir, rm, mv, du, df, find, cp, chmod, cd
    - uname, zip, unzip, gzip, tar
    - tr, sed, sort, uniq, ascii
    - Type “man command” to read about shell commands

What do these shell commands do?

- cat dups.txt | sort | uniq
- cat dups.txt | sort -V | uniq
- cat dups.txt | sort -V | uniq > outfile.txt
- tr "a" "e" < z.txt
- cat z.txt | tr a e

Batch processing with Unix Tools

```
cat /var/log/nginx/access.log |  ❶
  awk '{print $7}' |             ❷
  sort             |             ❸
  uniq -c          |             ❹
  sort -r -n       |             ❺
  head -n 5                      ❻
```

```
4189 /favicon.ico
3631 /2013/05/24/improving-security-of-ssh-private-keys.html
2124 /2012/12/05/schema-evolution-in-avro-protocol-buffers-thrift.html
1369 /
 915 /css/typography.css
```

- Read the log file.
- Split each line into fields by white space, output only the 7th element (requested URL).
- Alphabetically sort
- Filter out repeated lines.
- Sort it again based on the line number (-n)
- Out put the first five lines.

Transparency and experimentation

- The input files to Unix commands are normally treated as immutable.
  - Run most commands without damaging the input files.
- You can end the pipeline at any point, pipe the output into less, and look at it to see if it has the expected form.
  - Great for debugging.
- You can write the output of one pipeline stage to a file and use that file as input to the next stage.
  - Restart process.

The biggest limitation of Unix tools is that they run only on a single machine — and that's where tools like Hadoop come in.

Today's topic: Batch Processing

- Overview
- IO & Unix pipes
- MapReduce
  - HDFS - infrastructure
  - Job execution
  - Programming models
  - Workflow
- Beyond MapReduce

Google Data Centers

- Dalles, Oregon
- Hydroelectric power @ 2¢ / KW Hr
- 50 Megawatts
- Enough to power 60,000 homes

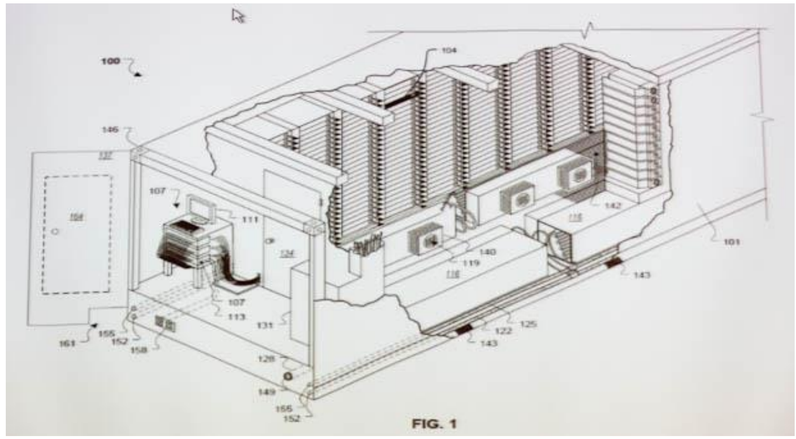
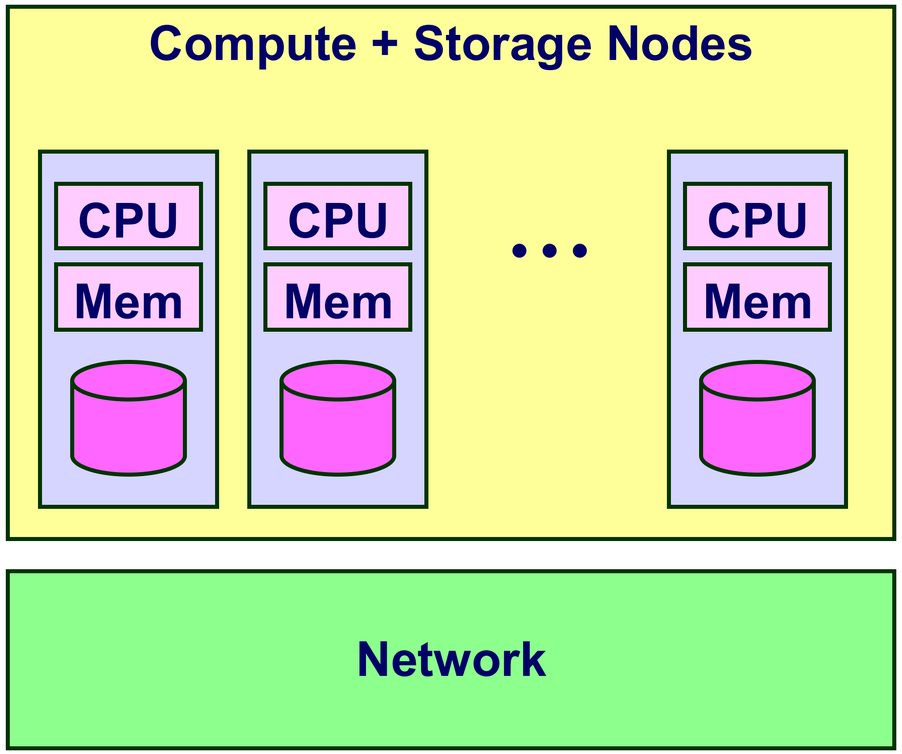


FIG. 1

- Engineered for maximum modularity & power efficiency
- Container: 1160 servers, 250KW
- Server: 2 disks, 2 processors

History of MapReduce/Hadoop



Compute + Storage Nodes

- Medium-performance processors
- Modest memory
- 1-2 disks

Network

- Conventional Ethernet switches
  - 10 Gb/s within rack
  - 100 Gb/s across racks

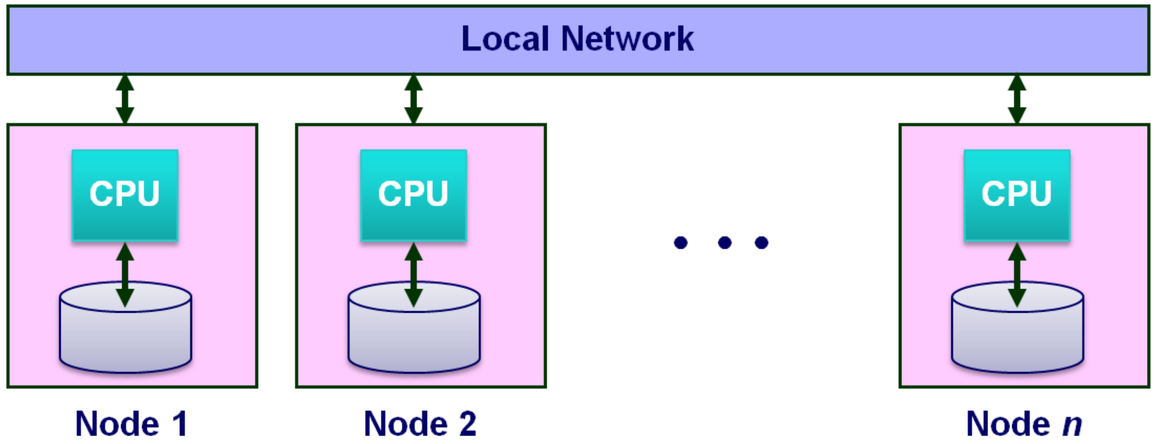
Data-Intensive System Challenge

For Computation That Accesses 1 TB in 5 minutes

- Data distributed over 100+ disks
  - Assuming uniform data partitioning
- Compute using 100+ processors
- Connected by gigabit Ethernet (or equivalent)

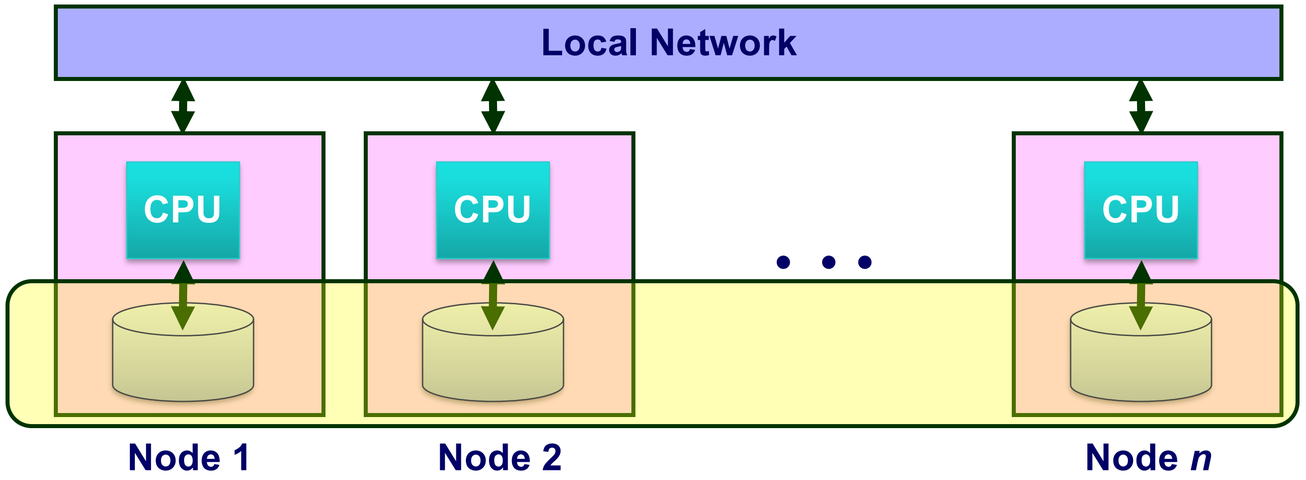
System Requirements

- Lots of disks
- Lots of processors
- Located in close proximity
  - Within reach of fast, local-area network



Hadoop Project

File system with files distributed across nodes

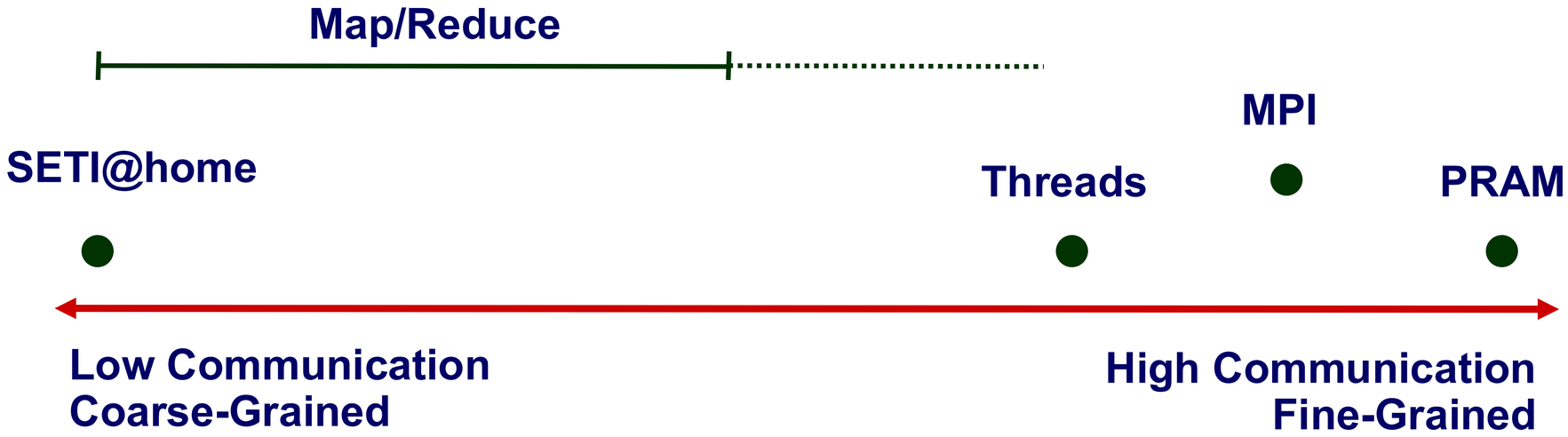


- Store multiple (typically 3 copies of each file)
  - If one node fails, data still available
- Logically, any node has access to any file
  - May need to fetch across network (ideally, leverage locality for perf.)

Map / Reduce programming environment

- Software manages execution of tasks on nodes

Exploring Parallel Computation Models



Map/Reduce Provides Coarse-Grained Parallelism

- Computation done by independent processes
- File-based communication

Observations

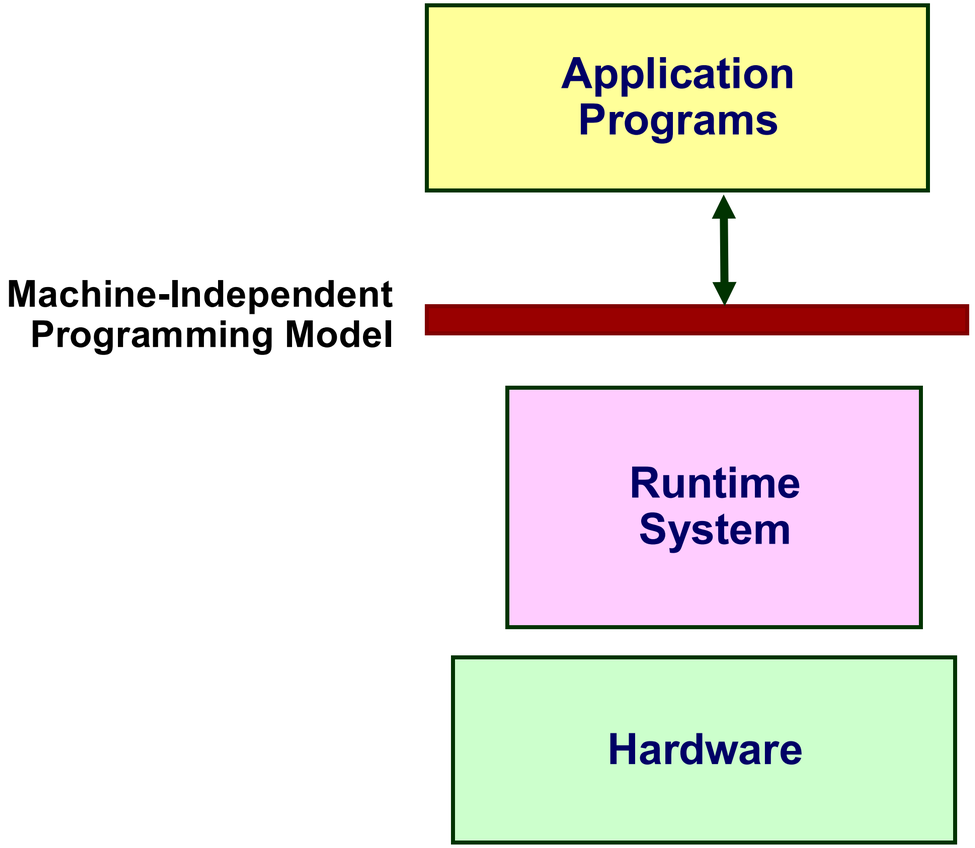
- Relatively "natural" programming model
- Research issue to explore full potential and limits

Today's topic: Batch Processing

- Overview
- IO & Unix pipes
- MapReduce
  - HDFS - infrastructure
  - Programming models (API)
  - Job execution (runtime)
  - Workflow
- Beyond MapReduce

Ideal Cluster Programming Model

- Application programs written in terms of high-level operations on data
- Runtime system controls scheduling, load balancing, …



MapReduce: Simplified Data Processing on Large Clusters

by Jeffrey Dean and Sanjay Ghemawat

Abstract

MapReduce is a programming model and an associated implementation for processing and generating large datasets that is amenable to a broad variety of real-world tasks. Users specify the computation in terms of a *map* and a *reduce* function, and the underlying runtime system automatically parallelizes the computation across large-scale clusters of machines, handles machine failures, and schedules inter-machine communication to make efficient use of the network and disks. Programmers find the system easy to use: more than ten thousand distinct MapReduce programs have been implemented internally at Google over the past four years, and an average of one hundred thousand MapReduce jobs are executed on Google's clusters every day, processing a total of more than twenty petabytes of data per day.

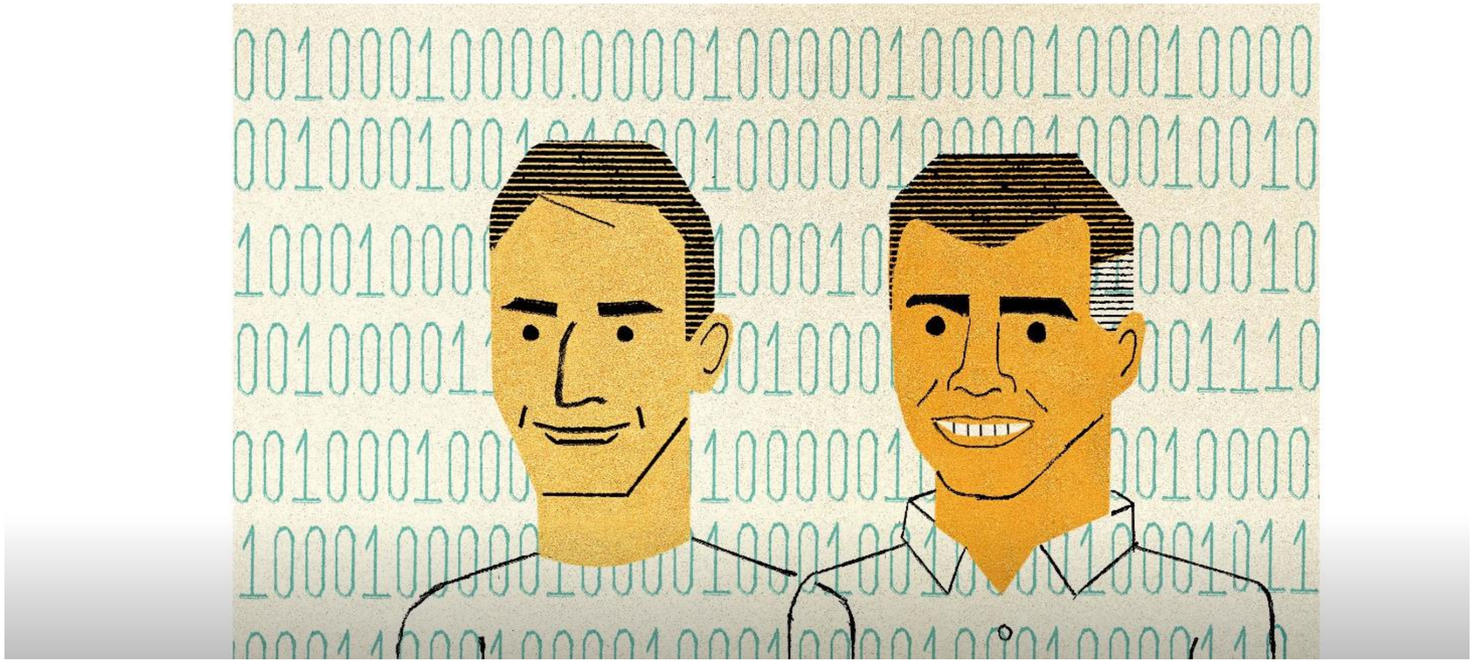
ANNALS OF TECHNOLOGY

THE FRIENDSHIP THAT MADE GOOGLE HUGE

Coding together at the same computer, Jeff Dean and Sanjay Ghemawat changed the course of the company—and the Internet.

By James Somers

December 3, 2018



https://www.newyorker.com/magazine/2018/12/10/the-friendship-that-made-google-huge

TF-IDF

TF-IDF is a measure of originality of a word by comparing the number of times a word appears in a doc with the number of docs the word appears in.

$$\text{TF-IDF} = \text{TF}(t, d) \times \text{IDF}(t)$$

Term frequency: Number of times term t appears in a doc, d

Inverse document frequency

$$\log \frac{1 + n}{1 + \text{df}(d, t)} + 1$$

n ← # of documents

df(d, t) ← Document frequency of the term t

Count the number of occurrences of word in a large collection of documents

```
map(String key, String value):
    // key: document name
    // value: document contents
    for each word w in value:
        EmitIntermediate(w, "1");

reduce(String key, Iterator values):
    // key: a word
    // values: a list of counts
    int result = 0;
    for each v in values:
        result += ParseInt(v);
    Emit(AsString(result));
```

- Functional programming
- Functions are stateless
- They takes an input, processes and output a result.
- Pros and Cons?

Data models

```
map(String key, String value):
    // key: document name
    // value: document contents
    for each word w in value:
        EmitIntermediate(w, "1");

reduce(String key, Iterator values):
    // key: a word
    // values: a list of counts
    int result = 0;
    for each v in values:
        result += ParseInt(v);
    Emit(AsString(result));
```

map	(k1,v1)	→ list(k2,v2)
reduce	(k2,list(v2))	→ list(v2)

MapReduce Example

- Create a word index of set of documents

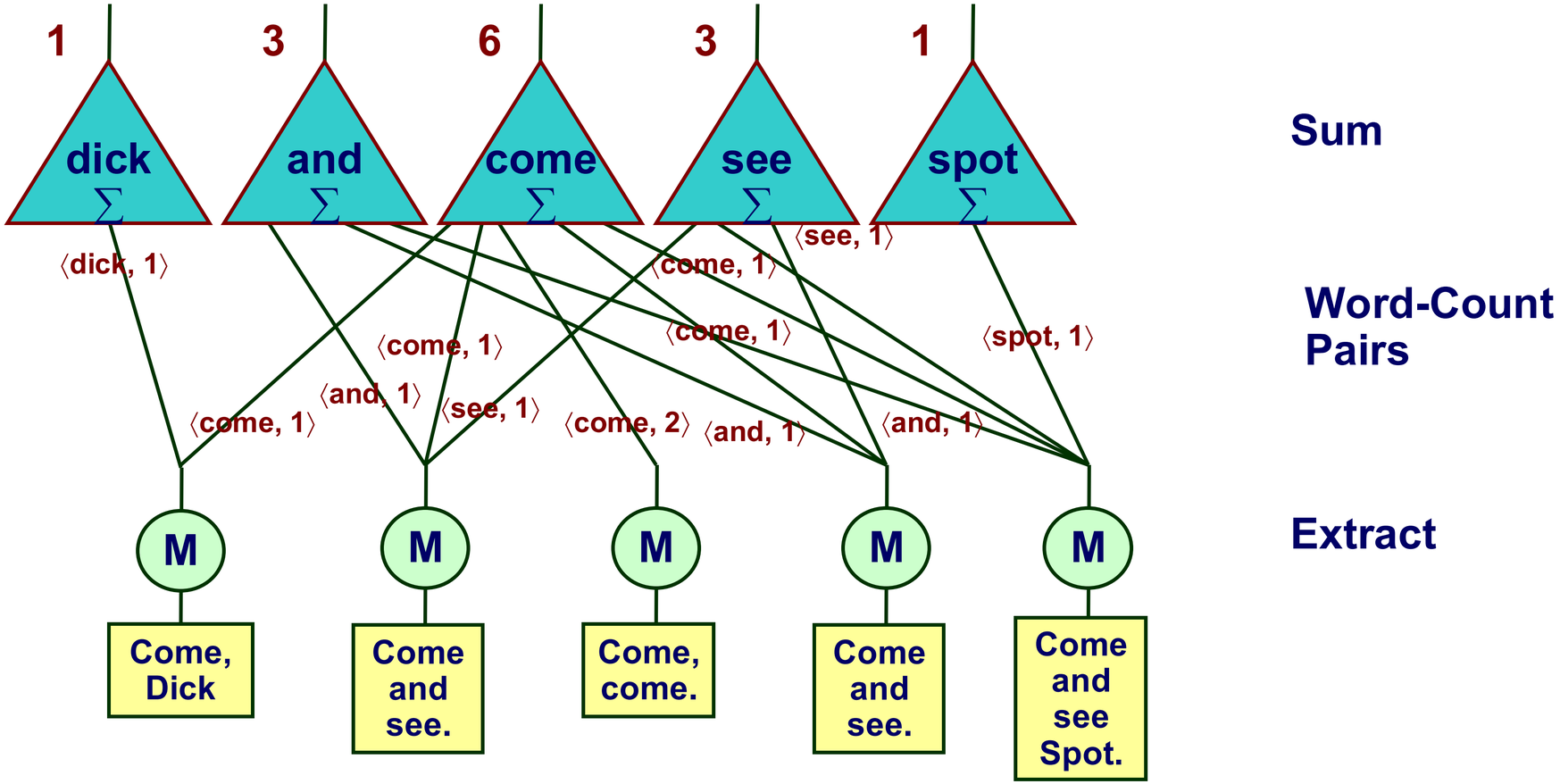
Come, Dick

Come and see.

Come, come.

Come and see.

Come and see Spot.



- Map: generate ⟨word, count⟩ pairs for all words in document
- Reduce: sum word counts across documents

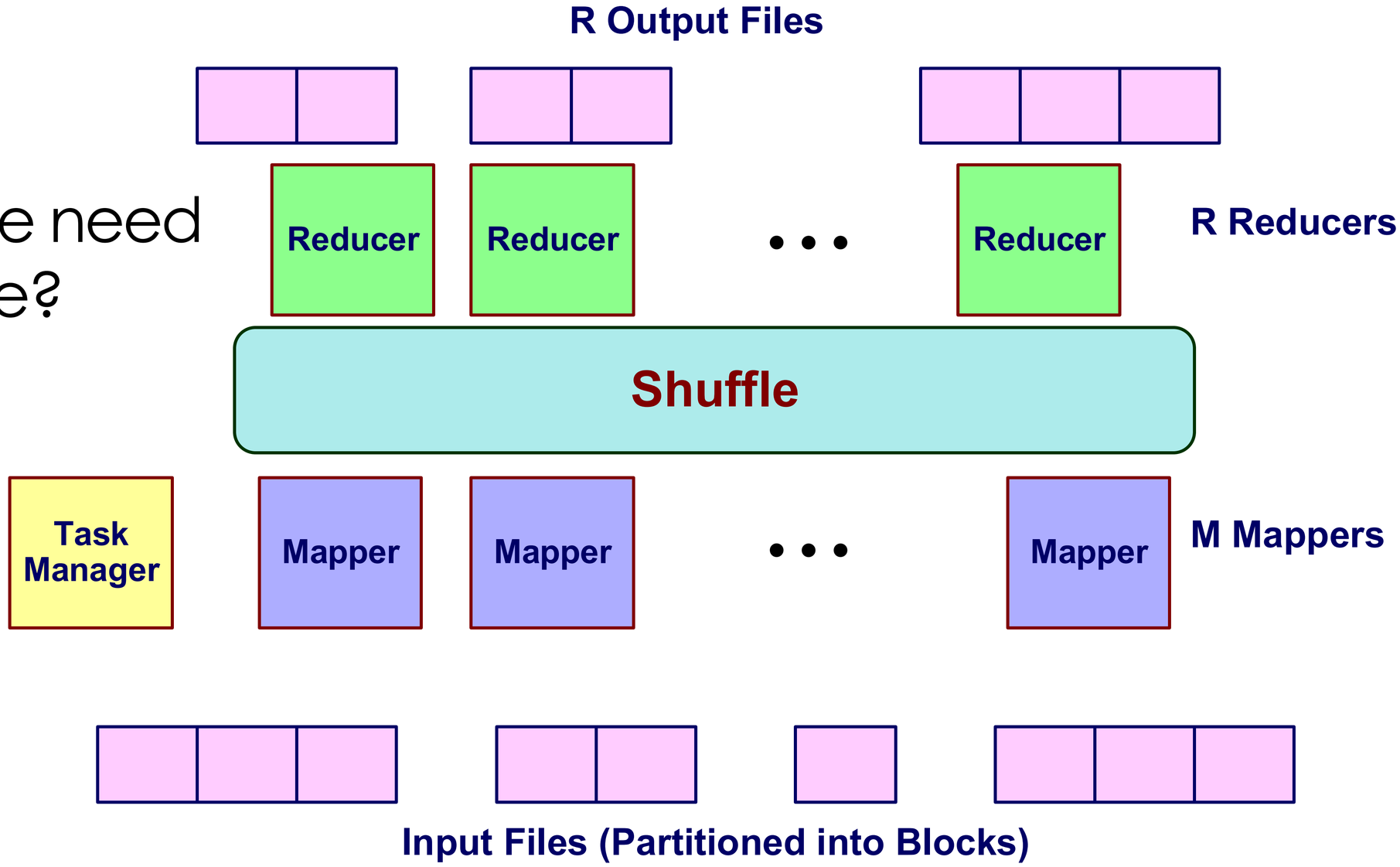
Discussion:
Other possible way to implement this using map-reduce?

Today's topic: Batch Processing

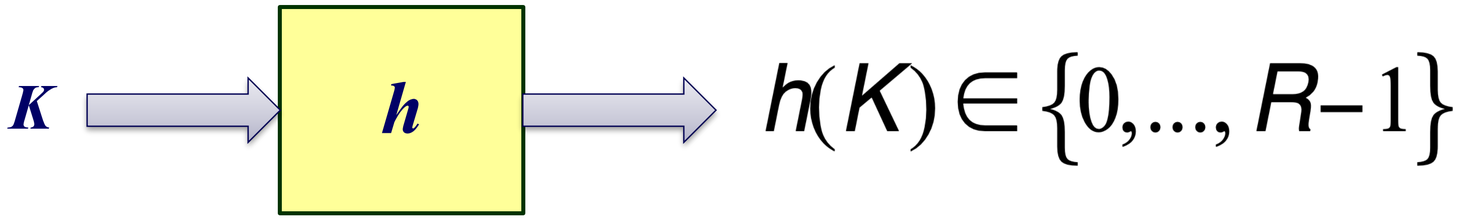
- Overview
- IO & Unix pipes
- MapReduce
  - HDFS - infrastructure
  - Programming models
  - Job execution
  - Workflow
- Beyond MapReduce

MapReduce Execution (Runtime)

Why do we need shuffle?



Single Mapper

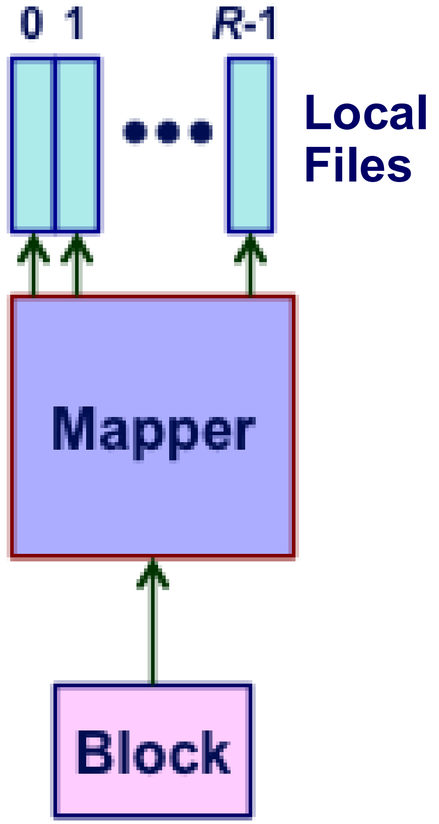


Hash Function *h*

- Maps each key *K* to integer *i* such that $0 \le i < R$

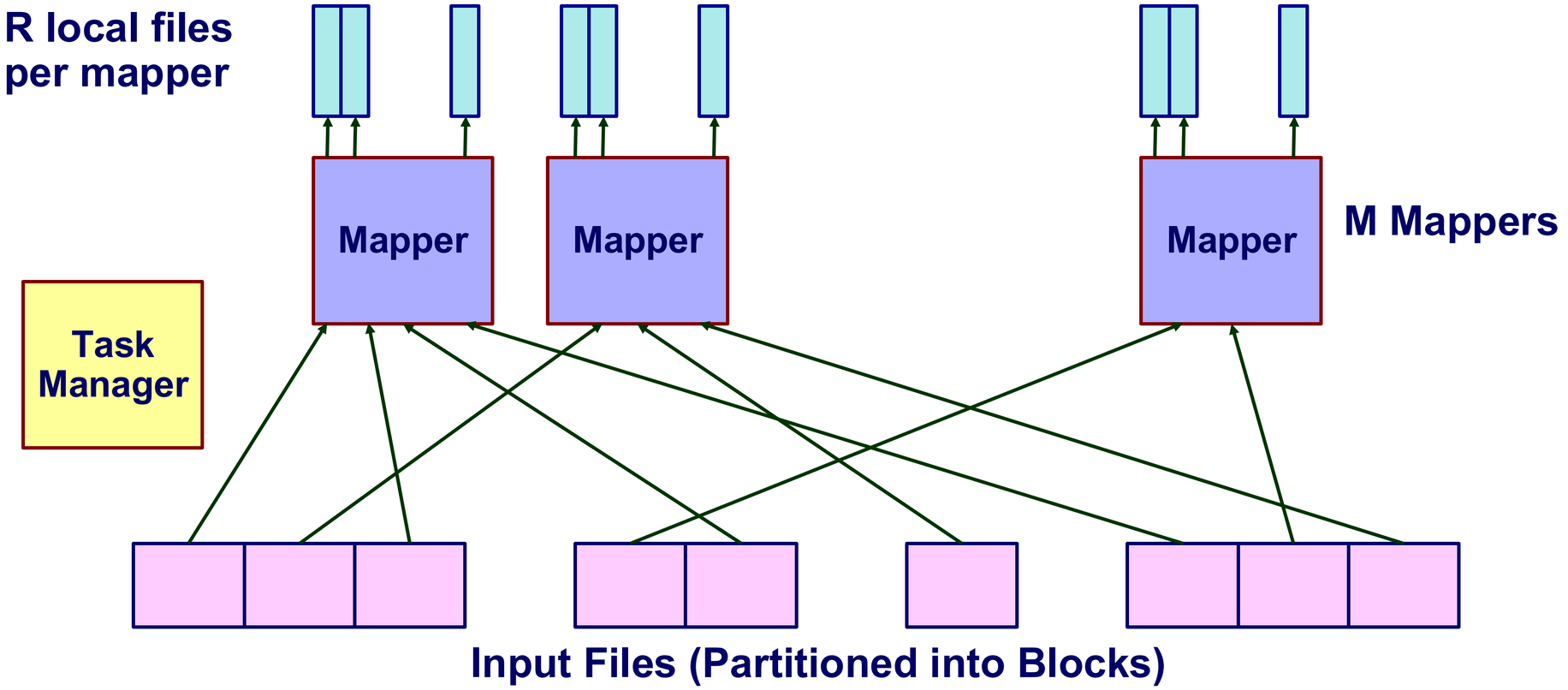
Mapper Operation

- Reads input file blocks
- Generates pairs ⟨*K*, *V*⟩
- Writes to local file *h*(*K*)



Distributed Mapper

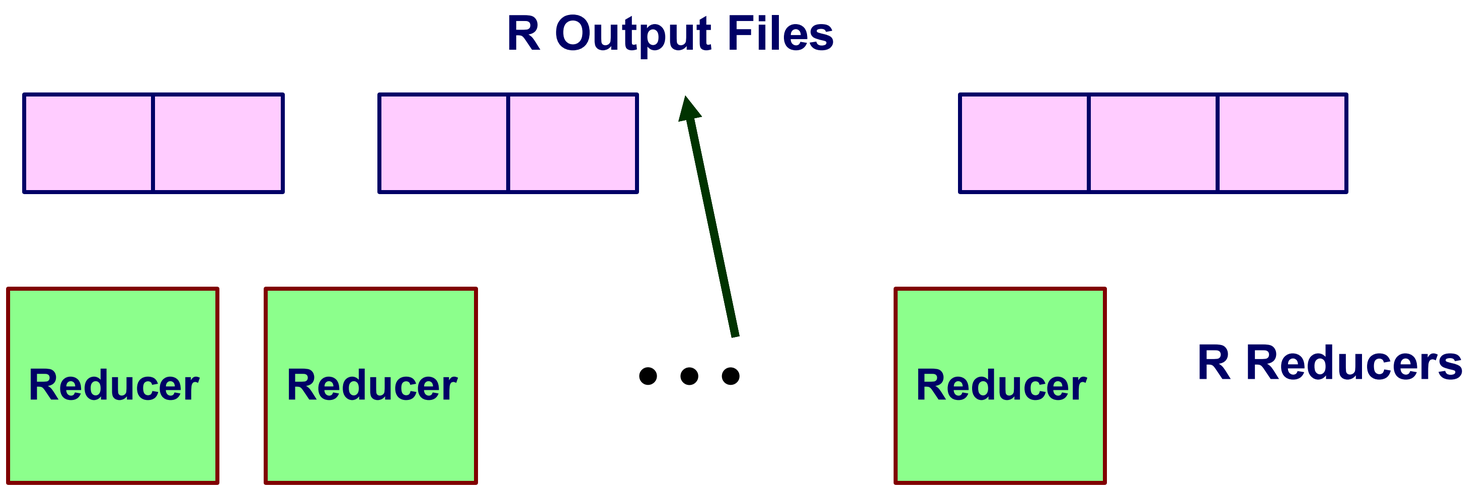
- Dynamically map input file blocks onto mappers
- Each generates key/value pairs from its blocks
- Each writes R files on local file system



Reducer

Each Reducer:

- Executes reducer function for each key
- Writes output values to parallel file system

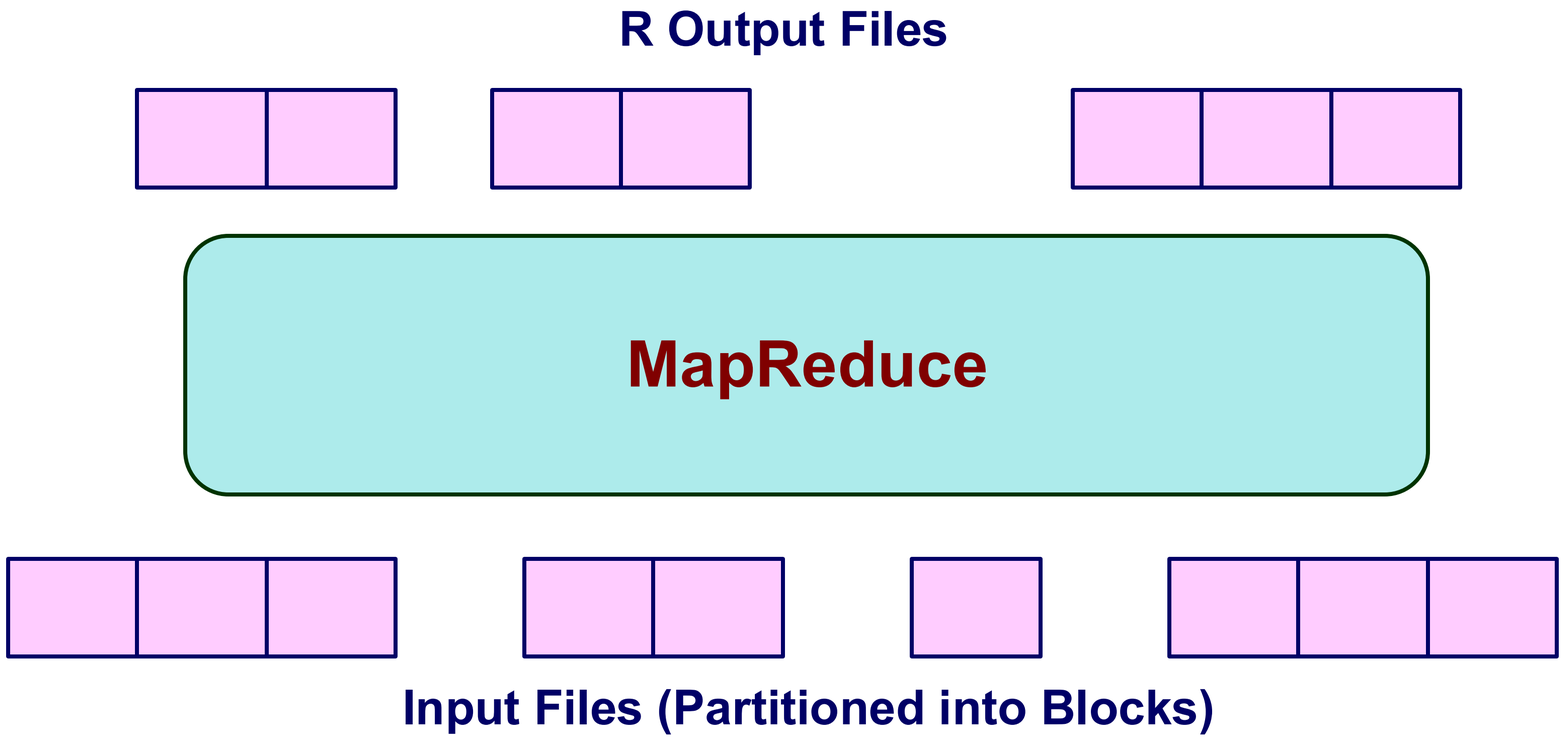


MapReduce Effect

MapReduce Step

- Reads set of files from file system
- Generates new set of files

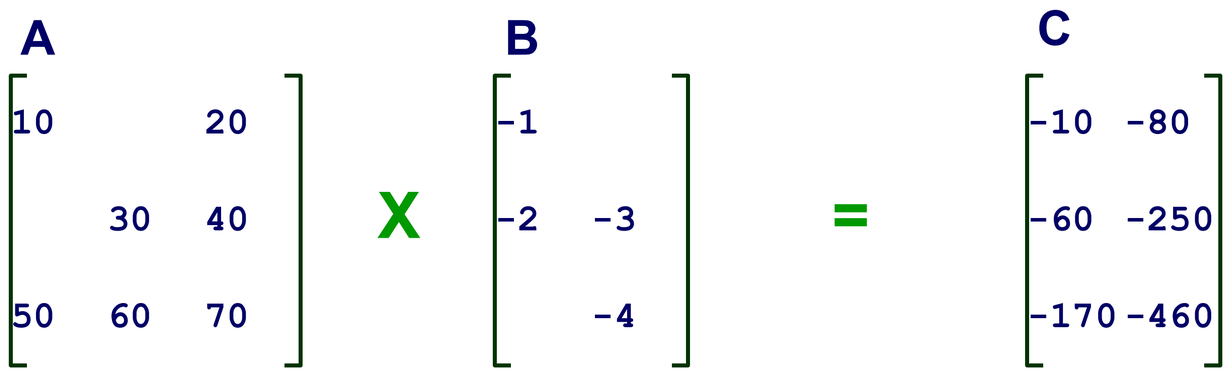
Can iterate to do more complex processing



Today's topic: Batch Processing

- Overview
- IO & Unix pipes
- MapReduce
  - HDFS - infrastructure
  - Programming models (API)
  - Job execution (runtime)
  - MapReduce dataflow
- Beyond MapReduce

Example: Sparse Matrices with Map/Reduce

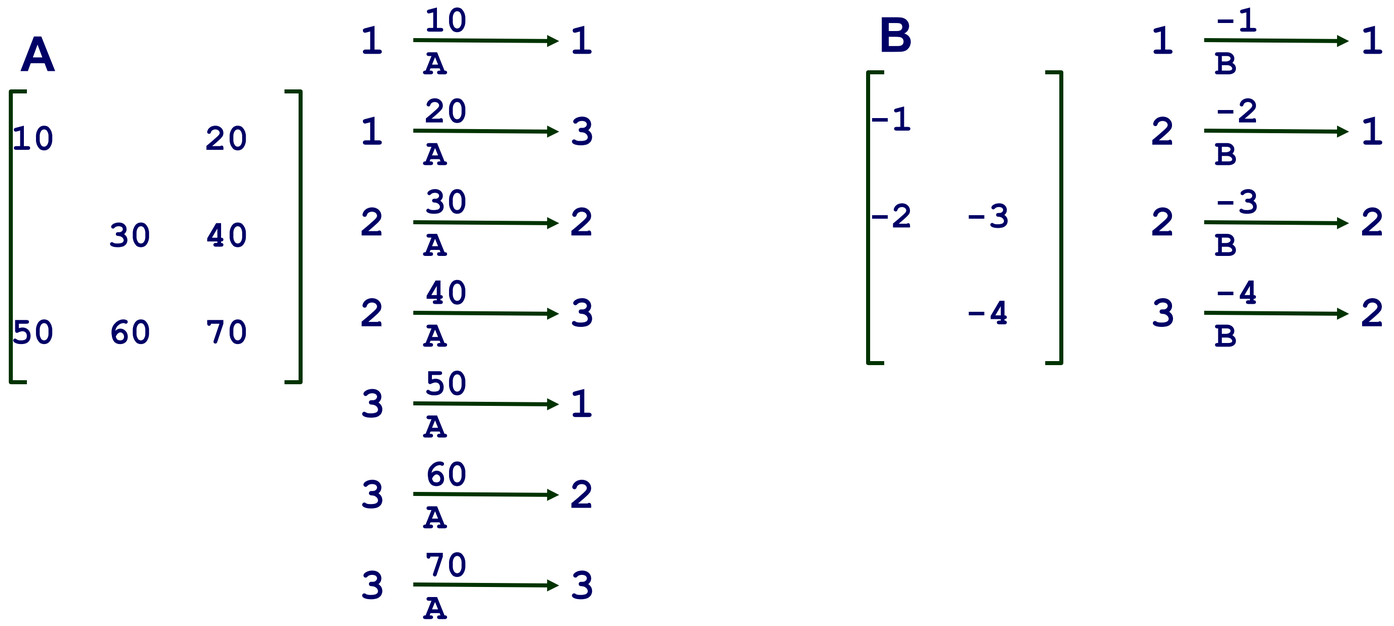


- Task: Compute product C = A·B
- Assume most matrix entries are 0

Motivation

- Core problem in scientific computing
- Challenging for parallel execution

Computing Sparse Matrix Product

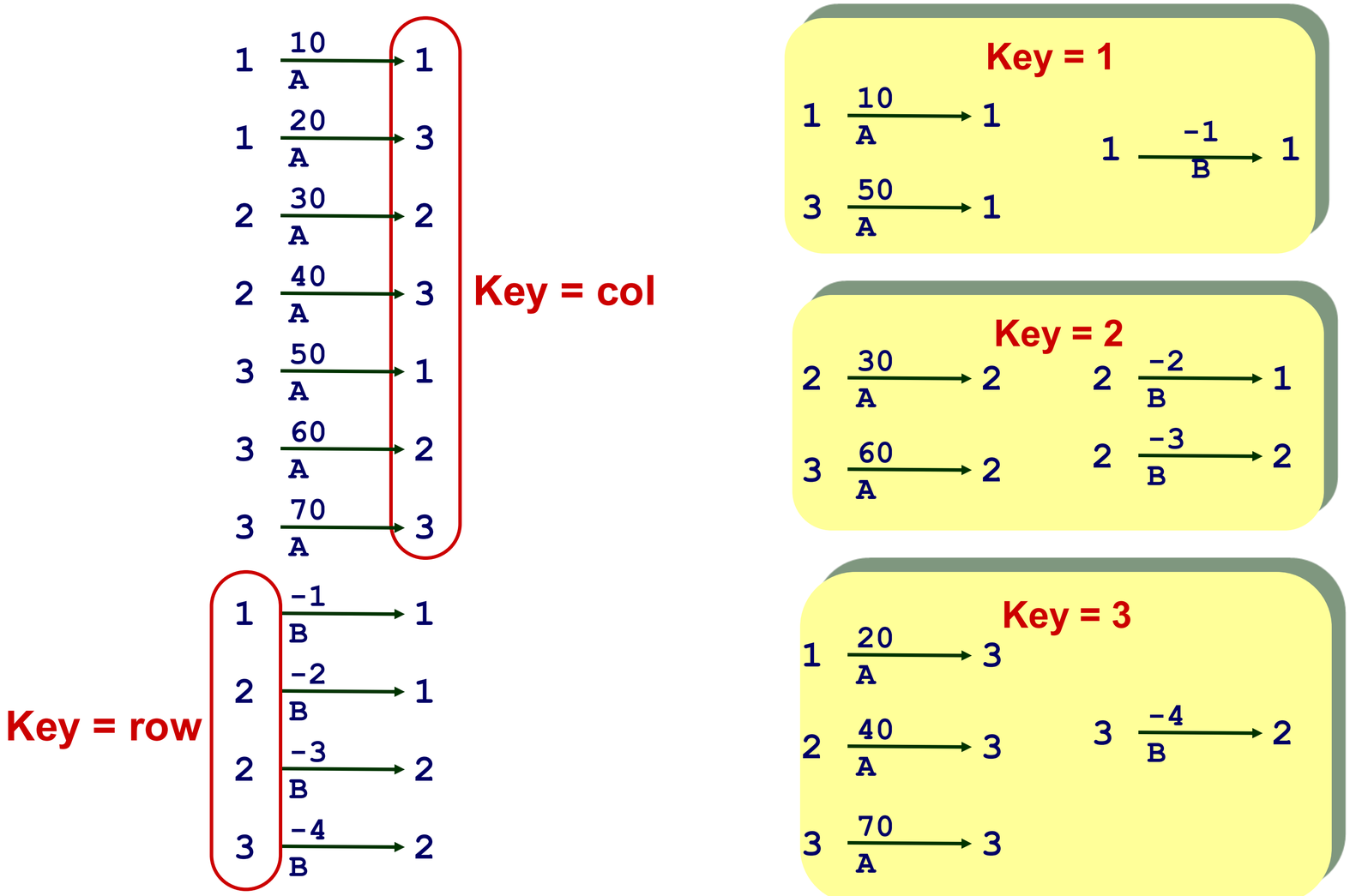


- Represent matrix as list of nonzero entries

  ⟨row, col, value, matrixID⟩

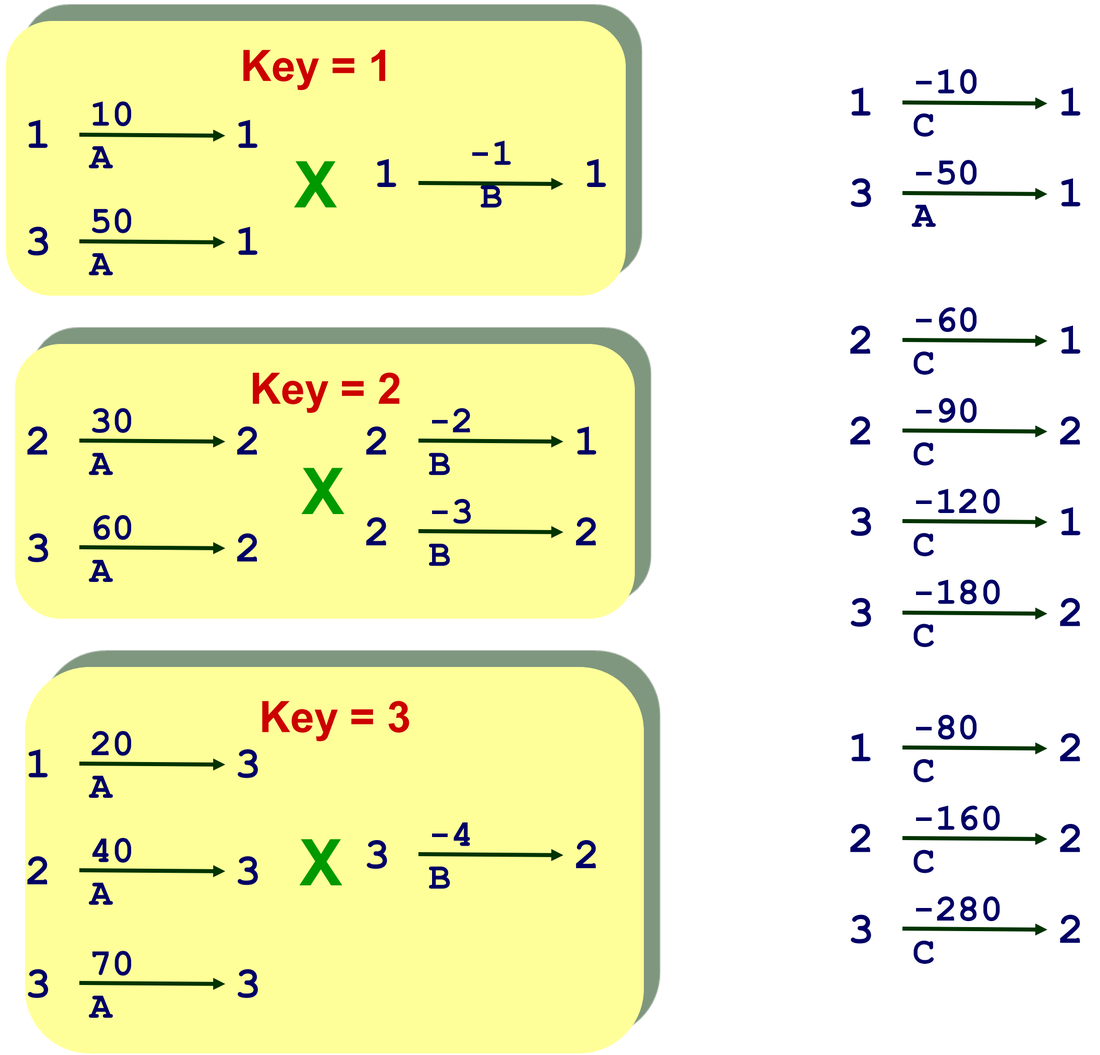
- How to represent the computation as map-reduce?
  - Phase 1: Compute all products $a_{i,k} \cdot b_{k,j}$
  - Phase 2: Sum products for each entry i,j
  - Each phase involves a Map/Reduce

Phase 1 Map of Matrix Multiply



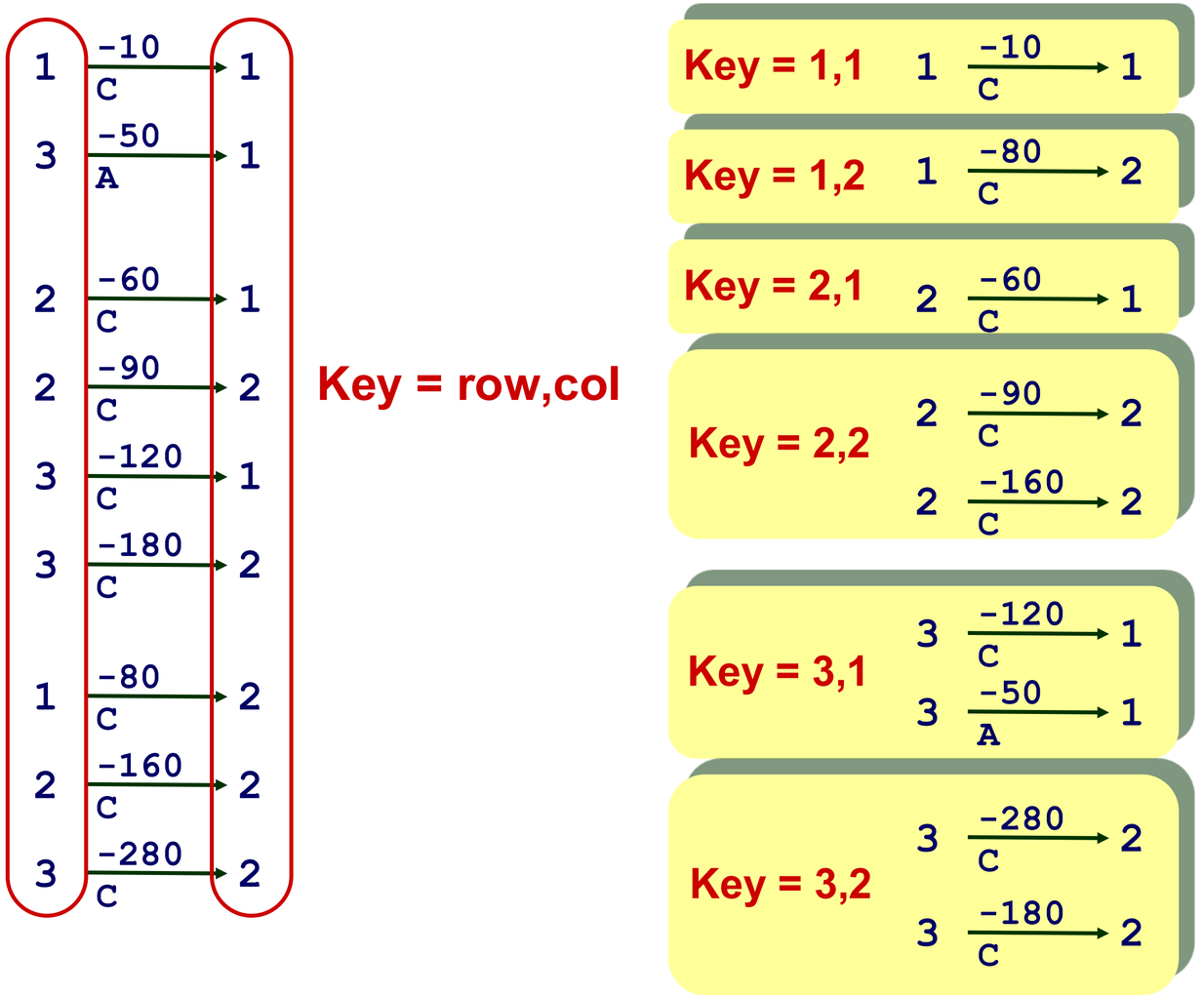
- Group values $a_{i,k}$ and $b_{k,j}$ according to key k

Phase 1 "Reduce" of Matrix Multiply



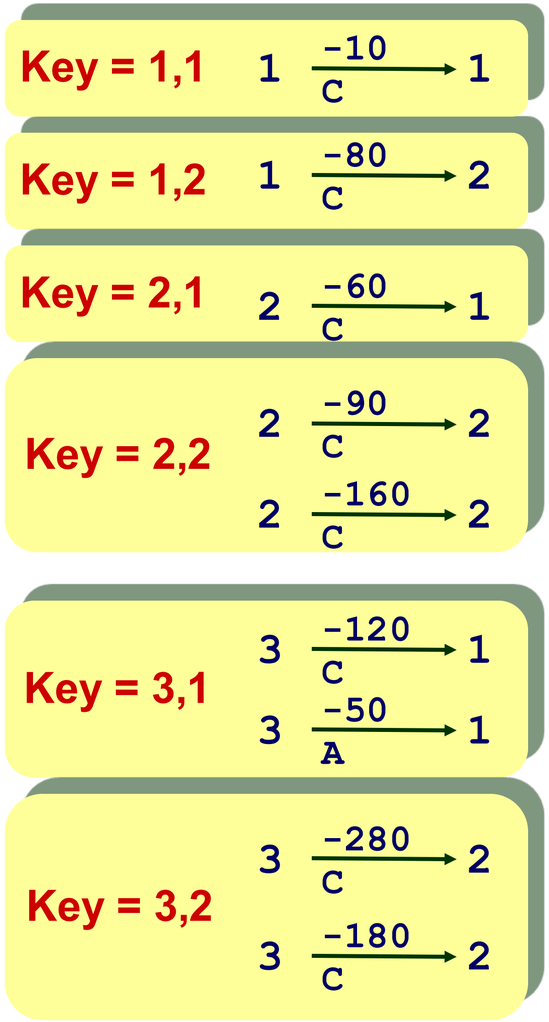
- Generate all products $a_{i,k} \cdot b_{k,j}$

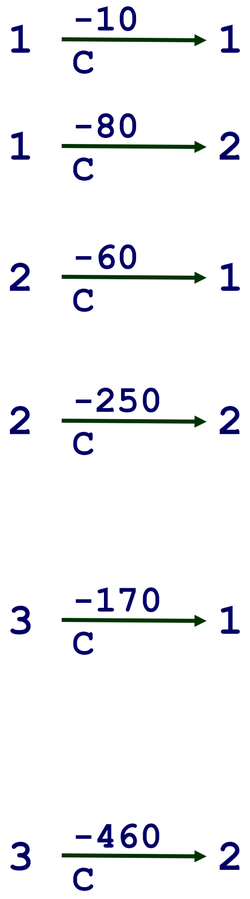
Phase 2 Map of Matrix Multiply



- Group products $a_{i,k} \cdot b_{k,j}$ with matching values of i and j

Phase 2 Reduce of Matrix Multiply





C

$$\begin{bmatrix} -10 & -80 \\ -60 & -250 \\ -170 & -460 \end{bmatrix}$$

- Sum products to get final entries

Recap: MapReduce Implementation

Built on Top of Parallel File System

- Google: GFS, Hadoop: HDFS
- Provides global naming
- Reliability via replication (typically 3 copies)

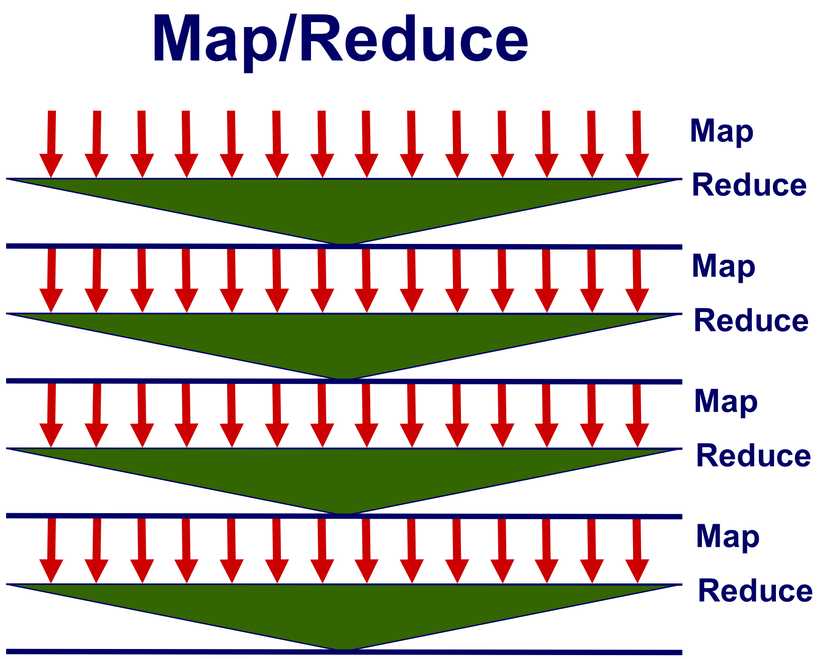
Breaks work into tasks

- Master schedules tasks on workers dynamically
- Typically #tasks >> #processors

Net Effect

- Input: Set of files in reliable file system
- Output: Set of files in reliable file system

Analyzing Pros and Cons of Map/Reduce



Characteristics

- Computation broken into many, short-lived tasks
  - Mapping, reducing
- Use disk storage to hold intermediate results

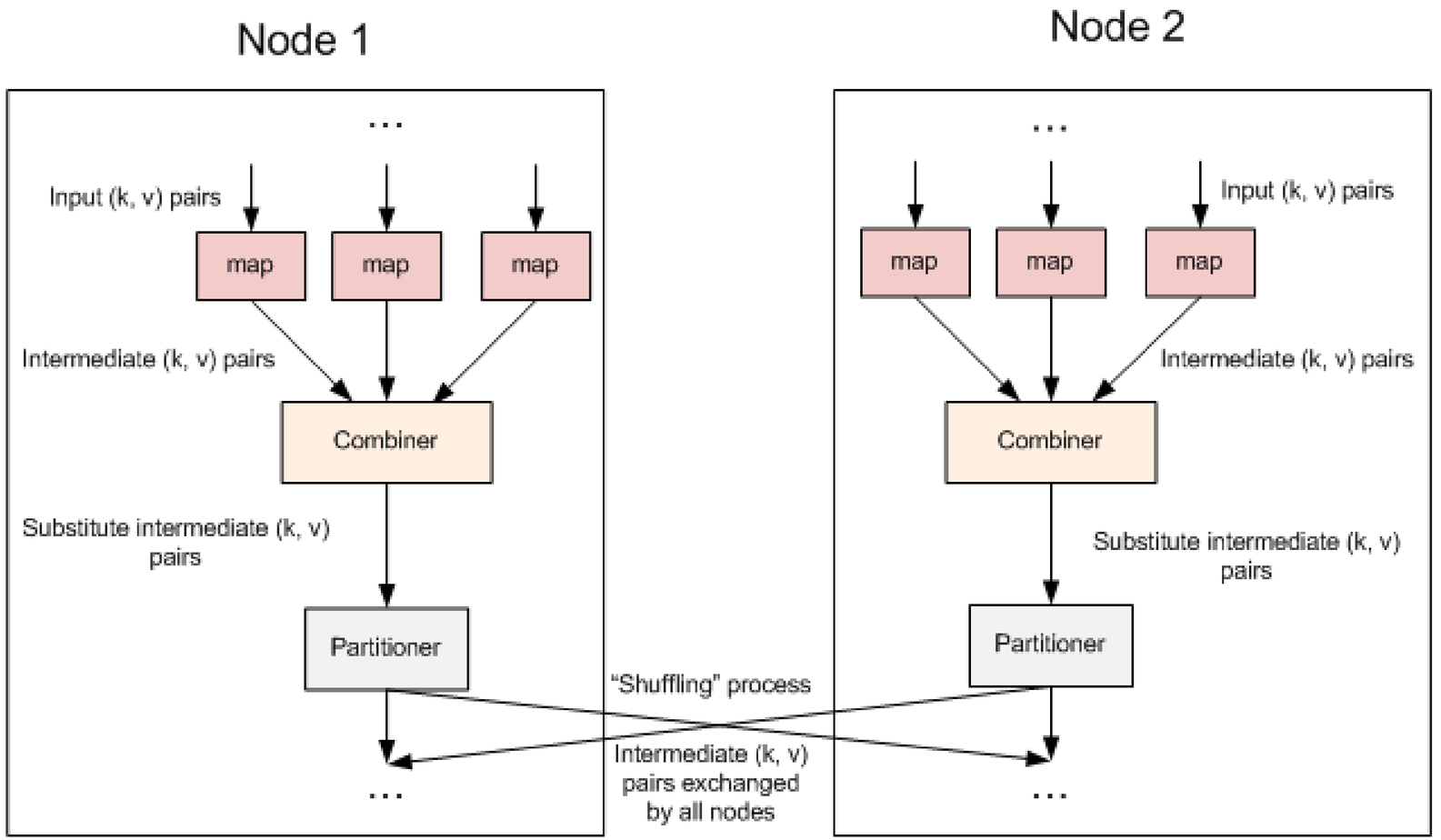
Strengths

- Great flexibility in placement, scheduling, and load balancing
- Can access large data sets

Weaknesses

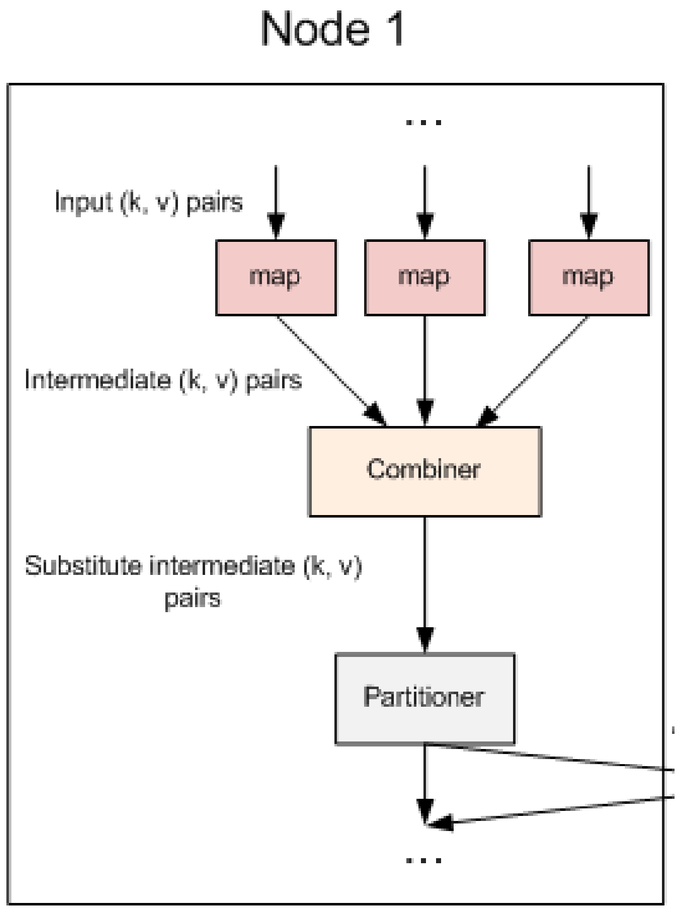
- Higher overhead due to disk read/write
- Lower raw performance (each map / reduce task takes long to invoke)
- Learning Functional programming is non-trivial!

Beyond Map/Reduce: Combiner and Partitioner



Combiners & Partitioners are optional.

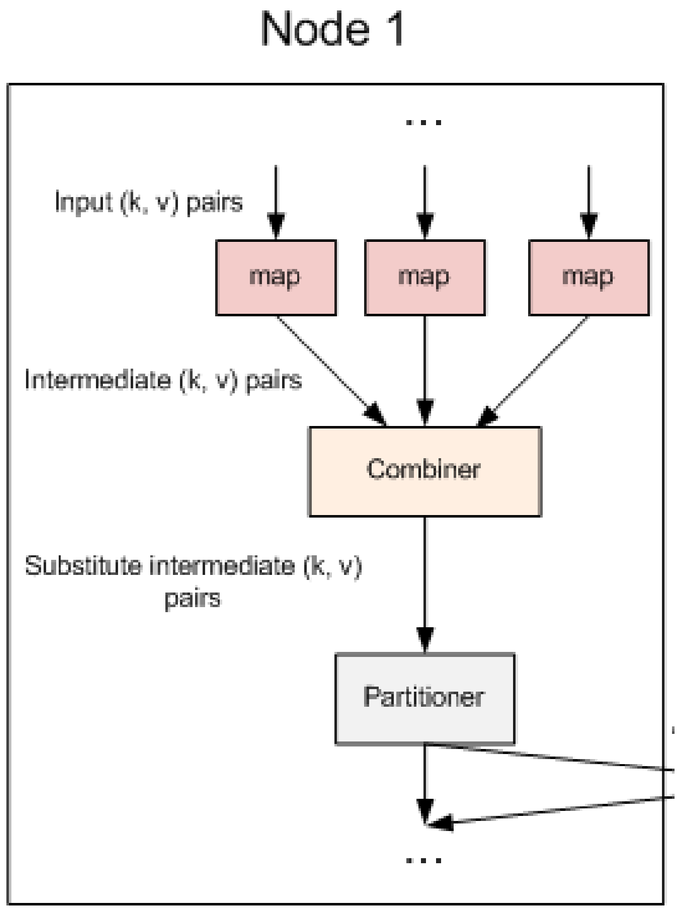
Input → Map → Combiner → Partitioner → Reducer → Output



Input:
What do you mean by Object
What do you know about Java
What is Java Virtual Machine
How Java enabled High Performance

Record reader:
<1, What do you mean by Object>
<2, What do you know about Java>
<3, What is Java Virtual Machine>
<4, How Java enabled High Performance>

Combiner (mini-reducer):
optional, to summarize the map output records with the same key



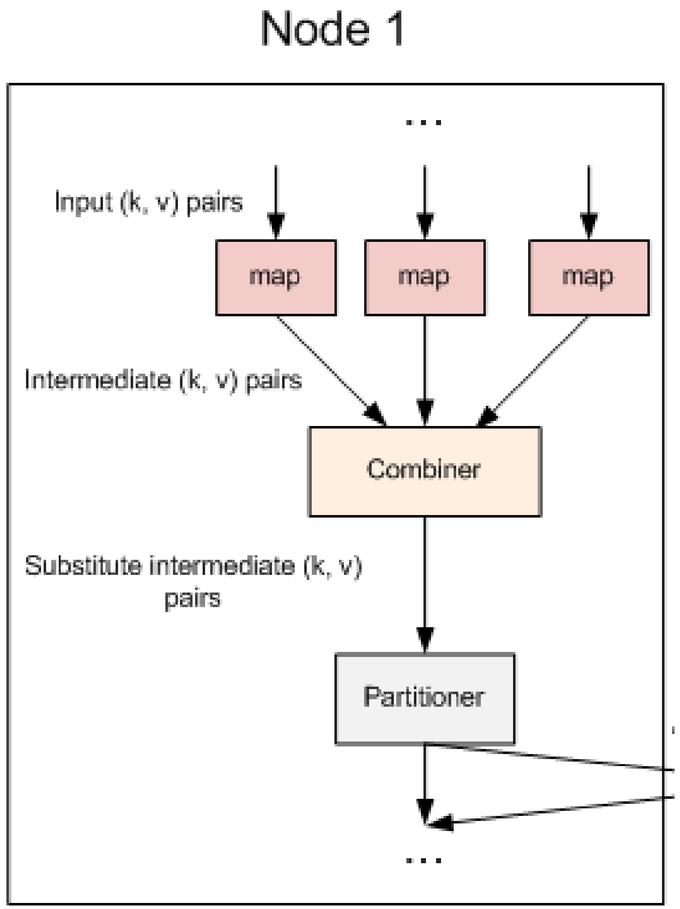
Map output:

<What,1> <do,1> <you,1> <mean,1> <by,1>
<Object,1> <What,1> <do,1> <you,1> <know,1>
<about,1> <Java,1> <What,1> <is,1> <Java,1>
<Virtual,1> <Machine,1> <How,1> <Java,1>
<enabled,1> <High,1> <Performance,1>

Combiner output:

<What,1,1,1> <do,1,1> <you,1,1> <mean,1> <by,1>
<Object,1> <know,1> <about,1> <Java,1,1,1> <is,1>
<Virtual,1> <Machine,1> <How,1> <enabled,1>
<High,1> <Performance,1>

Partitioner:
optional, a condition in processing an input dataset



Combiner output:
<What,1,1,1> <do,1,1> <you,1,1> <mean,1> <by,1>
<Object,1> <know,1> <about,1> <Java,1,1,1> <is,1>
<Virtual,1> <Machine,1> <How,1> <enabled,1>
<High,1> <Performance,1>

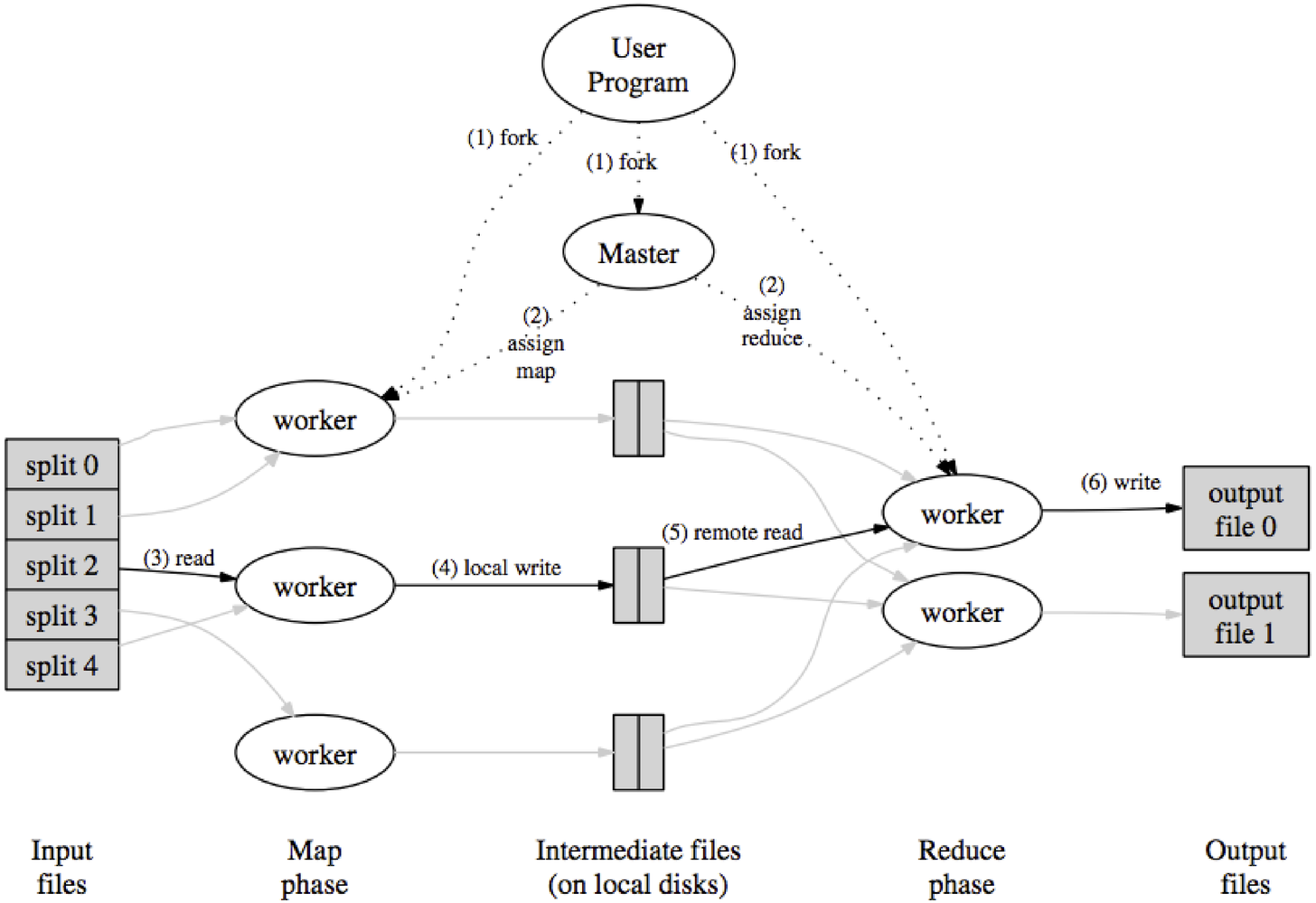
The number of partitioners is equal to the number of reducers.

Partitioner output:
<What,1,1,1>: long sentence,
<do,1,1>: long sentence,
……

Today's topic: Batch Processing

- Overview
- IO & Unix pipes
- MapReduce
  - HDFS - infrastructure
  - Programming models (API)
  - Job execution (runtime)
  - Workflow
  - MapReduce Recap
- Beyond MapReduce

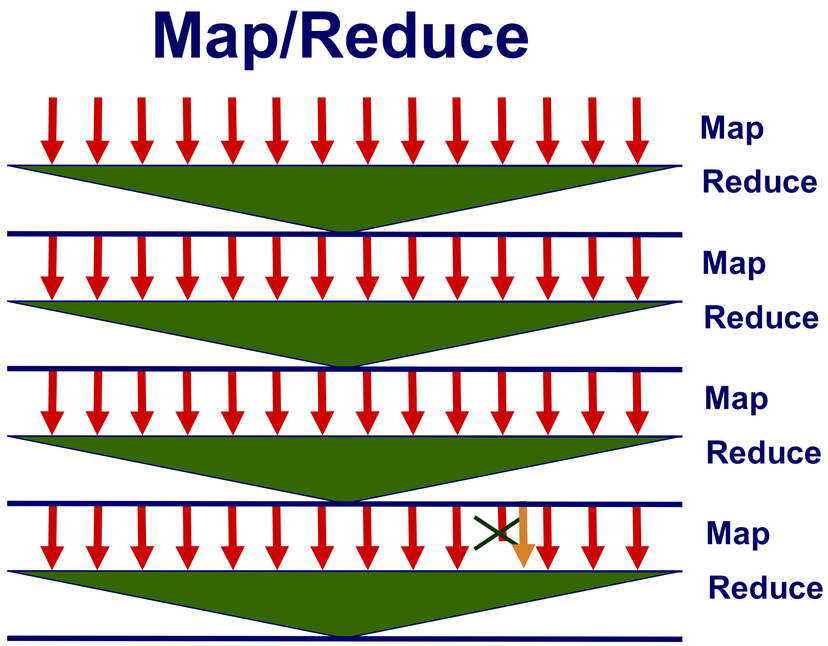
MapReduce System architecture (Paper)



Fault Tolerance and Straggler Mitigation

- Fault Tolerance
  - Assume reliable file system
  - Detect failed worker
    - Heartbeat mechanism
  - Reschedule failed task
- Dealing with Stragglers
  - Tasks that take long time to execute
  - Might be bug, flaky hardware, or poor partitioning
  - When done with most tasks, reschedule any remaining executing tasks
    - Keep track of redundant executions
    - Significantly reduces overall run time

Fault Tolerance



Data Integrity

- Store multiple copies of each file
- Including intermediate results of each Map / Reduc
  - Continuous checkpointing

Recovering from Failure

- Simply recompute lost result
  - Localized effect
- Dynamic scheduler keeps all processors busy

Map/Reduce Summary

Typical Map/Reduce Applications

- Sequence of steps, each requiring map & reduce
- Series of data transformations

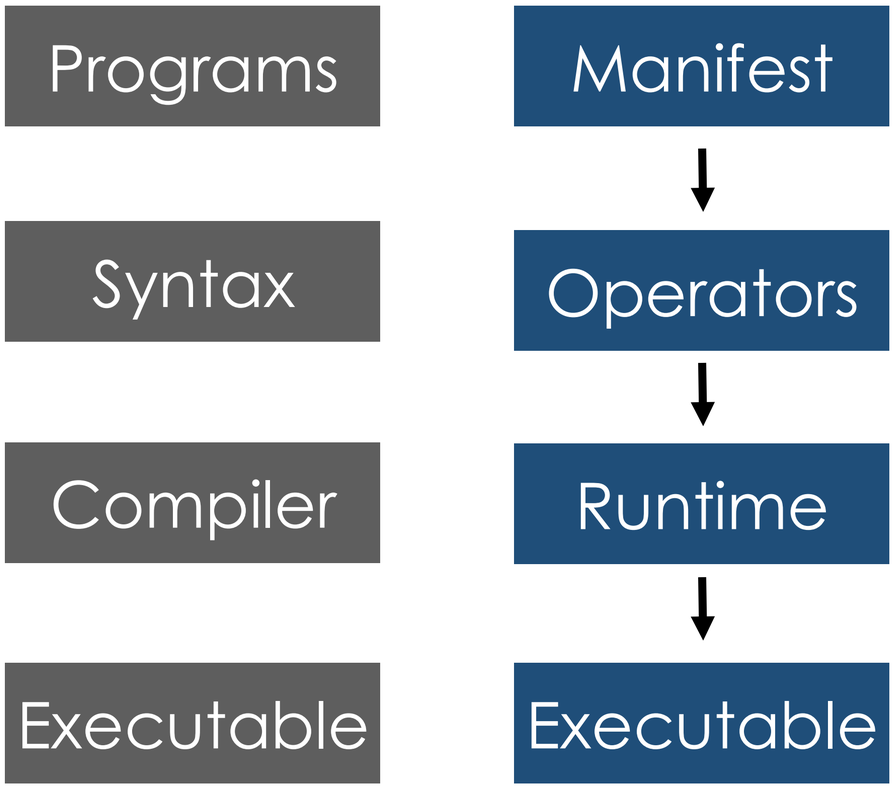
Strengths of Map/Reduce

- User writes simple functions, system manages complexities of mapping, synchronization, fault tolerance
- Very general
- Good for large-scale data analysis

Map Reduce Summary: Cons

- Disk I/O overhead is super high
- Not flexible enough: Each map/reduce step must complete before next begins
- Not suitable for workloads:
  - Iterative processing
  - Real-time processing
- Map-reduce is still difficult to program with

All Modern Data/ML Systems follow the following arch



A fixed set of operators

A trusted runtime with a small set of pre-loaded implementations

PageRank Computation

Initially

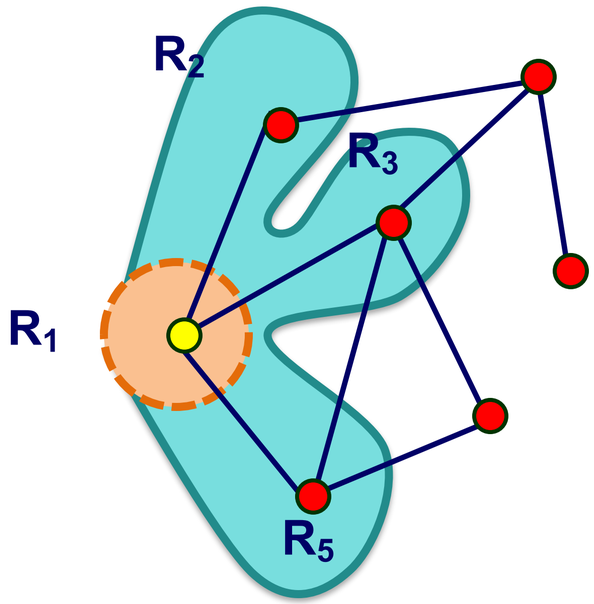
- Assign weight 1.0 to each page

Iteratively

- Select arbitrary node and update its value

Convergence

- Results unique, regardless of selection ordering



$$R_1 \leftarrow \quad 0.1 + 0.9 * (\tfrac{1}{2} R_2 + \tfrac{1}{4} R_3 + \tfrac{1}{3} R_5)$$

Q: how to express pagerank using map-reduce?