IPADS INSTITUTE OF PARALLEL AND DISTRIBUTED SYSTEMS

上海交通大學 SHANGHAI JIAO TONG UNIVERSITY

Distributed Computing frameworks

From MapReduce to computation graph to distributed training

Xingda Wei, Yubin Xia

IPADS, Shanghai Jiao Tong University

https://www.sjtu.edu.cn

Credits: Rong Chen@IPADS, CMU15-418

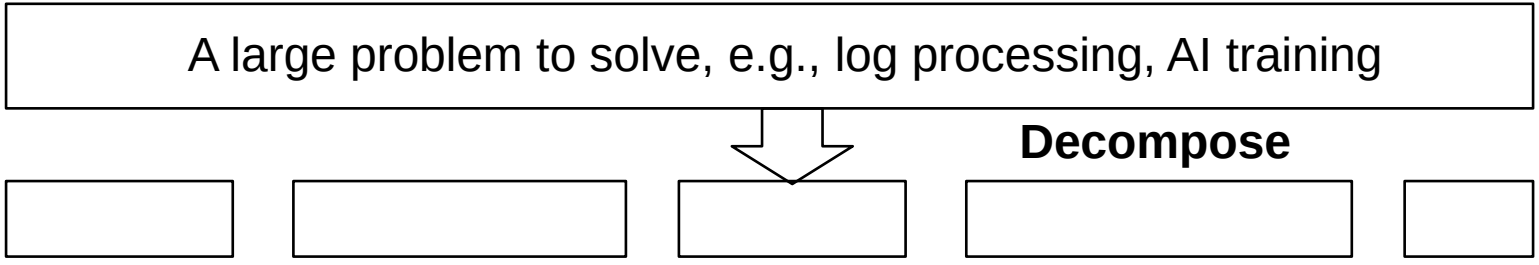
Review: simple command-line solution doesn’t scale

Example: log processing

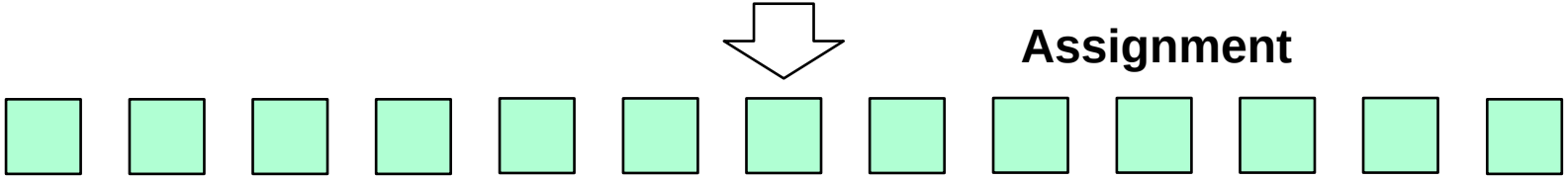
- Process large data
- Many computations can be paralleled processed

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

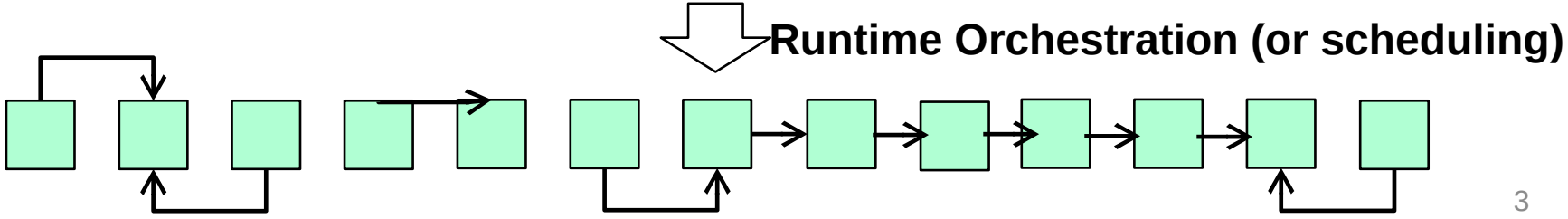
Review: process to distributed computing



Sub-problems. Many alias, e.g., sub-tasks, sub-jobs (or simply jobs)



A set of physical programs that can run on parallel units, e.g., a thread or a GPU kernel



Recall, common challenges of Distributed Computing

1. Sending **data** to/from nodes
2. **Coordinating** among nodes
3. Recovering from node **failure**
4. Optimizing for **locality**
5. **Partition** the data to enable more parallelism

Same for all problems

Review: we build a framework to hide the above tasks

Goal

- Reduce programmer’s effort in solving the above challenges

Challenges

- What are the abstraction? Thread & RPC is insufficient
- More general, harder to provide the above property
- E.g., if a thread fails, how can we know its progress?
    - It’s a very hard problem, we will come back to it later

MapReduce: a distributed batch processing system

Created by **Google** (OSDI'04)

- Jeffrey Dean and Sanjay Ghemawat

Inspired by LISP (function language)

- **Map** (function, set of values)
  - Applies function to each value in the set

(**map** #'length' (() (a) (a b) (a b c))) ✉ (0 1 2 3)

- **Reduce** (function, set of values)
  - Combines all the values using a function (e.g., +)

(**reduce** #'+' (1 2 3 4 5)) ✉ 15

Map & Reduce is slightly different in MapReduce

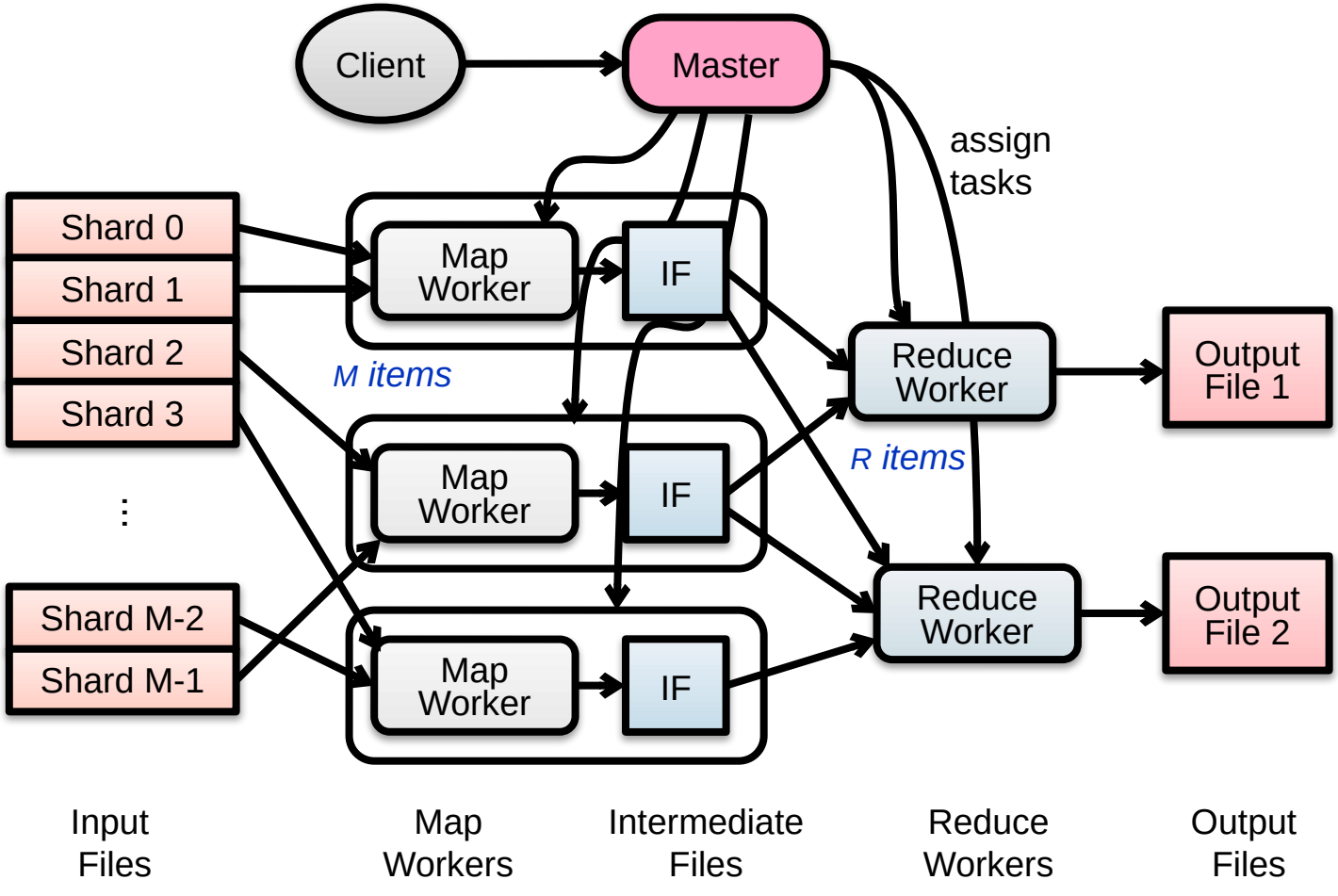
Map(input) -> [k,v]

- No f: the f is inside the Map itself (no high order function)
- A map is the same within the lifecycle of a MapReduce calculation

Reduce(k, values)

- Reduce on different keys can do parallelly

Review: The Complete Picture of MapReduce



Recall, common challenges of Distributed Computing

1. Sending **data** to/from nodes
2. **Coordinating** among nodes
3. Recovering from node **failure**
4. Optimizing for **locality**
5. **Partition** the data to enable more parallelism

Same for all problems

How does MapReduce handle these challenges?

Fault tolerance of MapReduce

Machine failures are common in datacenters

- A MapReduce job can possibly run on thousands of machines

MapReduce simplifies fault tolerance due to the following two choices:

1. Programming model simplifies fault recovery

   e.g., **no side-effect**: a map or a reduce can simply re-execute the computation to recover from failures

2. Builds on **a reliable service** (i.e., GFS)

Fault tolerance of MapReduce

Worker failure

- Master pings each worker periodically via heartbeat
- If no response is received within a certain time (**timeout**), the worker is marked as failed
- Map or reduce tasks given to this worker are reset back to the initial state and rescheduled for other worker (**re-execution**)
- Robust: lost 1,600 of 1,800 machines once, but finished fine

Question: why no side-effect?

- Simplify ensuring correctness

Fault tolerance of MapReduce

Master failure

- Can we re-execute?
- Master's state is persisted to **GFS**
- Recover master from GFS and continue

For each mapper & reducer

- Executing **state**: idle, in-progress or completed
- Locations of intermediate files

The state is checkpointed to GFS for fault tolerance

- Yet, master is unlikely to fail, since there is only one!

Fault tolerance of MapReduce

Skipping Bad Records

- Map/Reduce functions sometimes fail for some inputs
- Best solution is to debug & fix, but not always possible
- On segmentation fault:
    - Send UDP packet to master from signal handler
    - Include sequence number of record being processed
- If master sees two failures for same record:
    - Next worker is told to *skip* the record

Effect: Can work around bugs in third-party libraries

Optimization for Locality

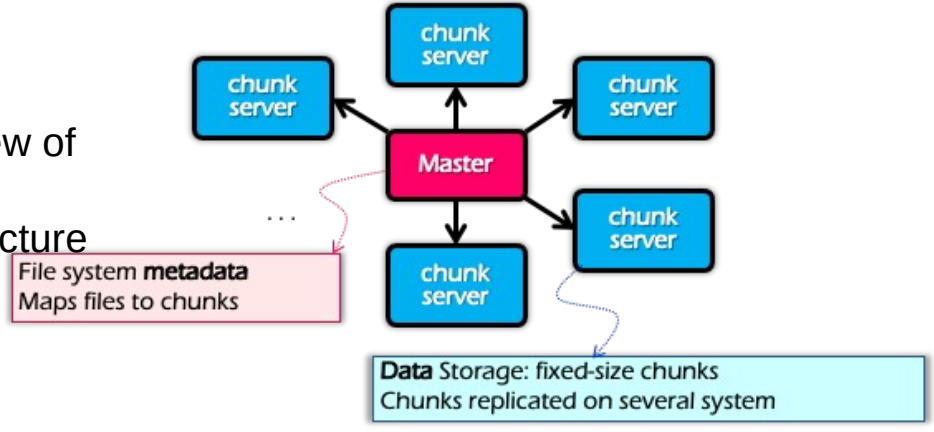
Problem: the bandwidth of datacenter network (even today) is scarce

- If a Map worker reads all the data from the network, it would be slow

Google: Input and Output files are stored on GFS

- Google File System (SOSP'03), see previous lectures
- Each chunk is replicated on multiple servers (3)

A review of GFS architecture



Optimization for Locality

Problem: the bandwidth of datacenter network (even today) is scarce

- If a Map worker reads all the data from the network, it would be slow

Google: Input and Output files are stored on GFS

- Google File System (SOSP'03), see previous lectures
- Each chunk is replicated on multiple servers (3)

MapReduce runs on **GFS chunkservers**

- Collocate computation and storage

Master tries to **schedule** map workers on one of the nodes that has a copy of the input chunk it needs

- A case of system co-design

Refinement: redundant execution

Some workers can be slower than others, aka, *stragglers*

- Other jobs consuming resources on machine
- Bad disks with soft errors transfer data slowly
- Weird things: processor caches disabled ⏸

Near end of phase, MapReduce spawn backup copies of tasks

- Whichever one finishes **first** “wins” ⏪
- Dramatically shortens job completion time:“*significantly reduces the time to complete large MapReduce operations* ”(check the paper)

Summary of MapReduce

Get a lot of data from **input** files

Map

- Parse & extract items of interest

Partition & Sort

Reduce

- Aggregate results

Write results to **output** files

All the other issues, scheduling, fault-tolerance, data partitioning, are handled by the framework

Summary of MapReduce

The user does not need to care ~~concurrency~~, ~~fault-tolerant~~, ~~data transfer~~, etc.

Reduce the user code burden and quality:

- Sort: only needs 50 LoC
- Indexing: reduce the code lines from 3800 LoC to 700LoC

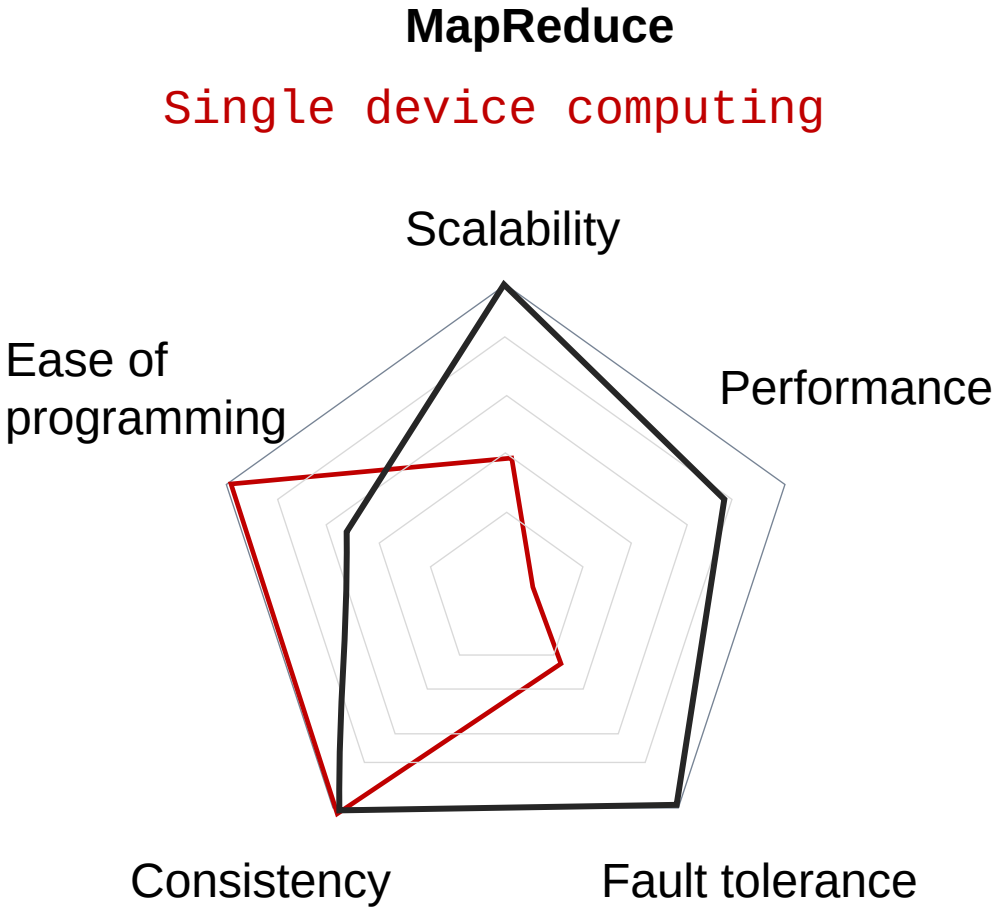
Summary of MapReduce

Pros:

- Easy to scale
- Fault tolerant
- Good performance (depends!)
    - Good for tasks that suits MapReduce, e.g., wordcount

Cons

- Limited programming abstraction



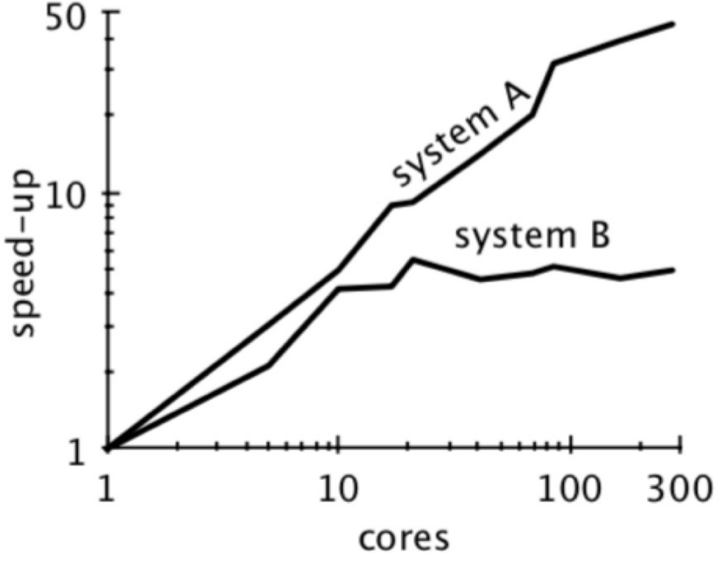
Question: is a scale system (like MapReduce) always better than a traditional system?

20xPR	cores	twitter_rv	uk_2007_05
Spark	128	857s	1759s
Giraph	128	596s	1235s
GraphLab	128	249s	833s
GraphX	128	419s	462s

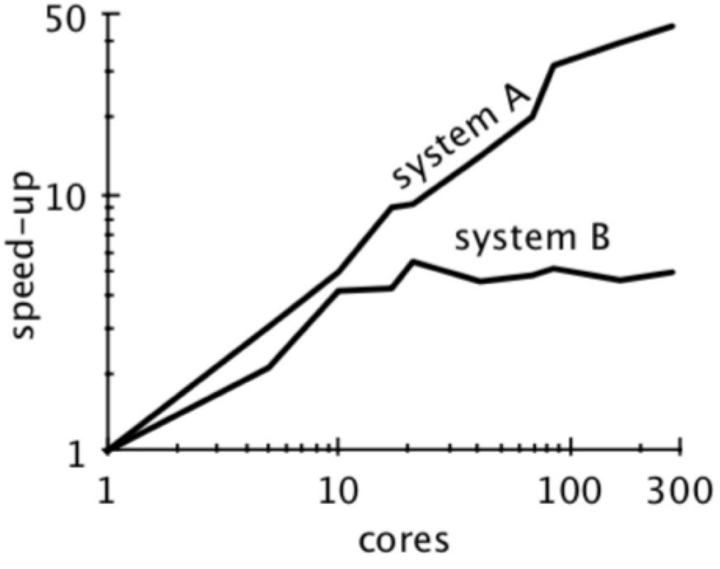
from Gonzalez *et al.*, OSDI 2014

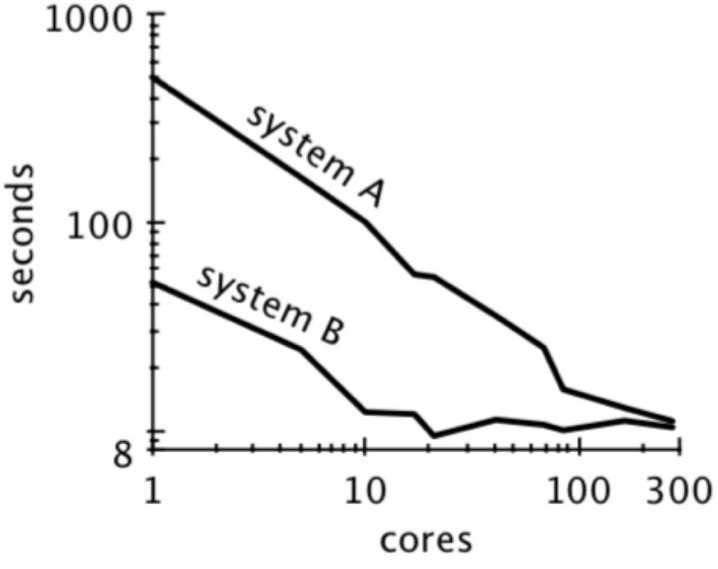
Connectivity	cores	twitter_rv	uk_2007_05
Spark	128	1784s	8000s+
Giraph	128	200s	8000s+
GraphLab	128	242s	714s
GraphX	128	251s	800s
Laptop	1	~~153s~~ 15s	~~417s~~ 30s

Question: which system is better?

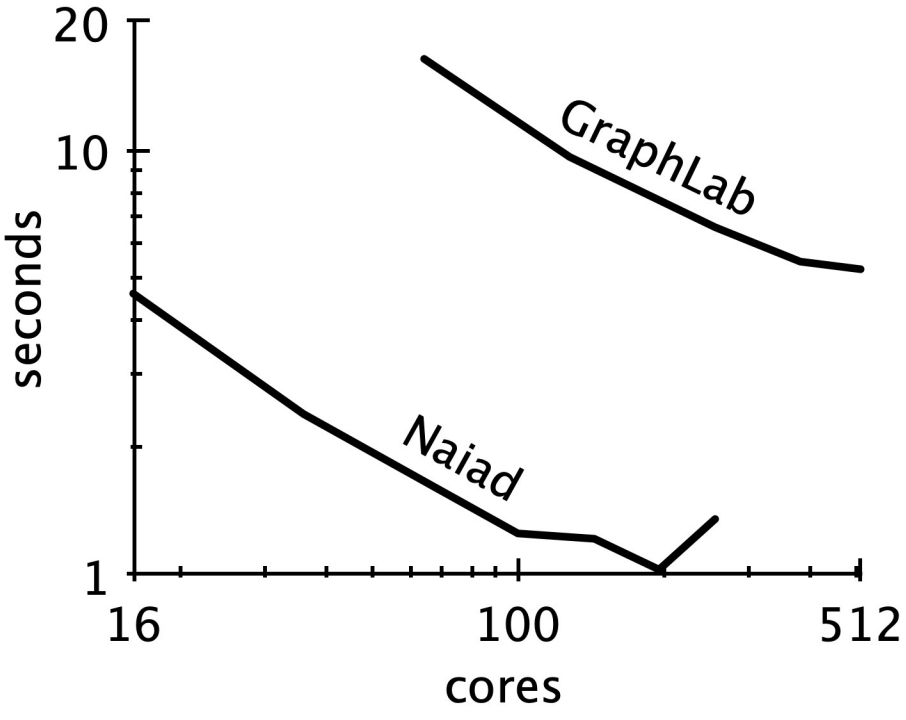


Question: which system is better?

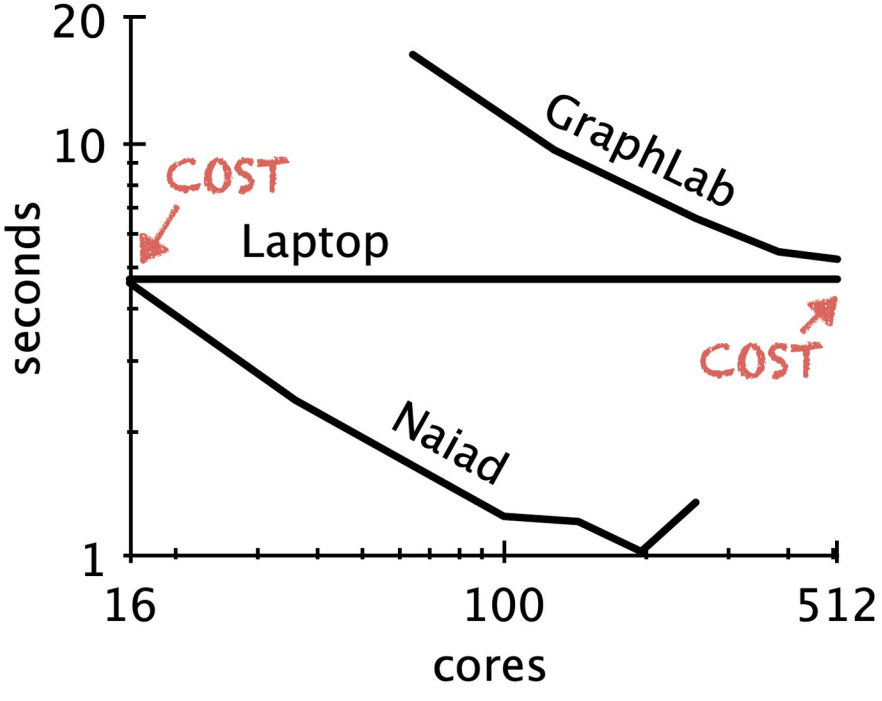




Workload: graph processing



Workload: graph processing



What can go wrong?

Scalability has the cost

- E.g. network communication is much slower than local accesses
- Synchronizations overhead to ensure correctness
- Fault tolerance mechanisms
    - Components in a distributed system can easily fail!

“You can have a second computer once you’ve shown you know how to use the first one.”

–Paul Barham

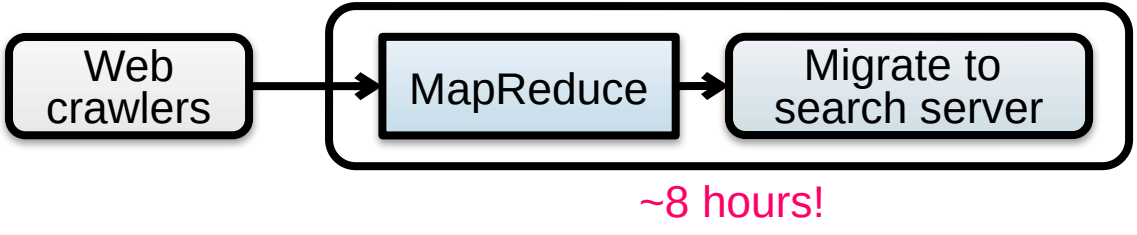
MapReduce cannot address all the issues

MapReduce was used to process webpage data collected by Google's crawlers

- It would extract the links and metadata needed to search the pages
- Determine the site's PageRank

The process took around **eight** hours

- Results were moved to search servers
- This was done **continuously**

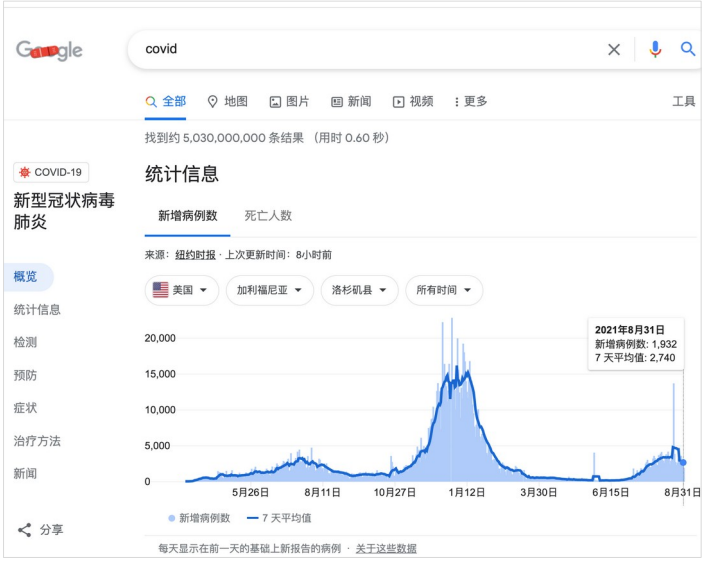


MapReduce cannot address all the issues

Web has become more **dynamic**

- An 8+ hour delay is a lot for some sites

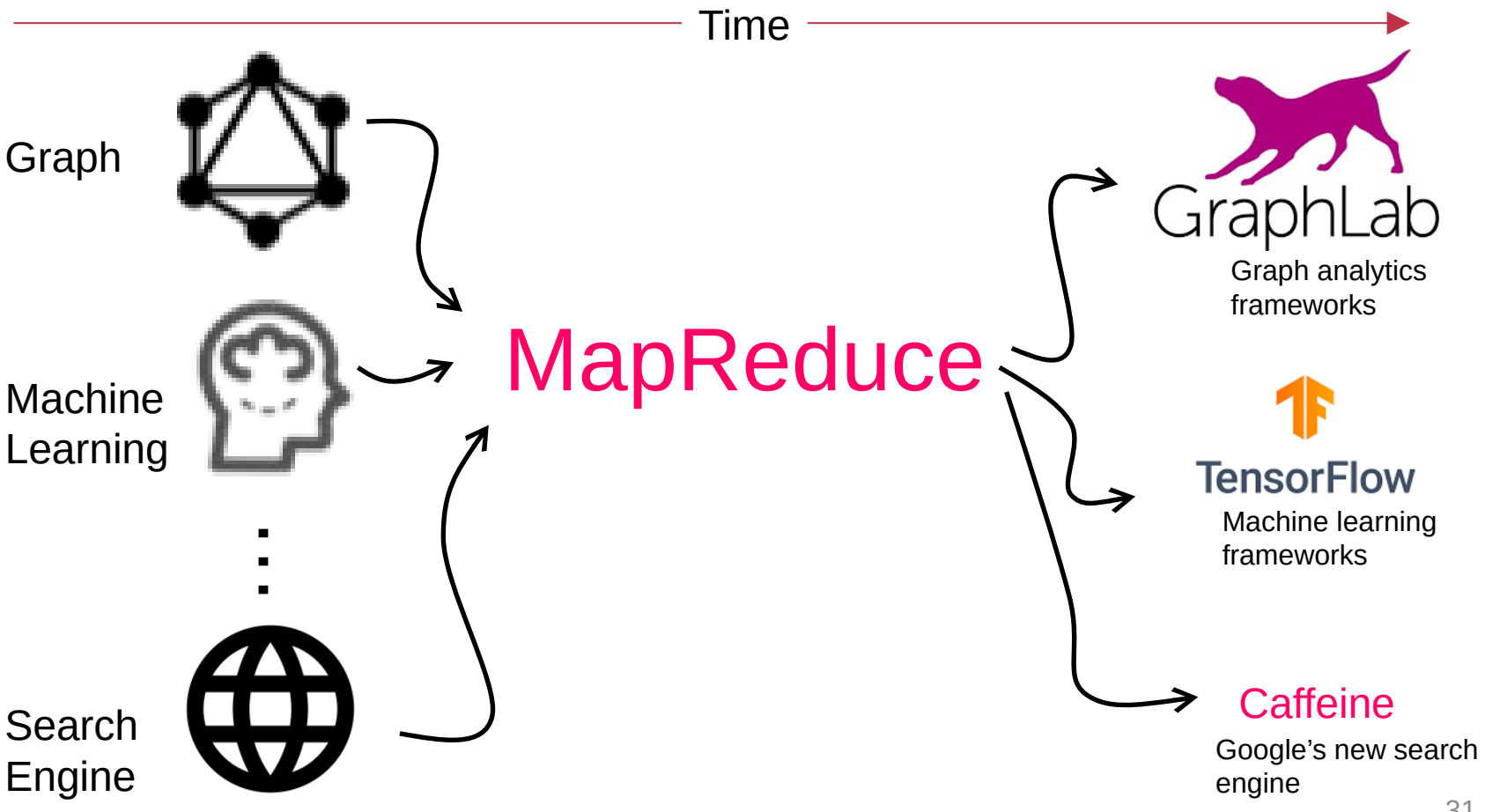
Goal: refresh certain pages within seconds



MapReduce

- Batch-oriented
- Huge performance overhead
- Not suited for **near-real-time** processes
- Cannot start a new phase util previous completed
- Not optimized for specific tasks (e.g., Graph, ML)

MapReduce cannot address all the issues



Restrictiveness of MapReduce Programming Model

Question#1

- How can we use MapReduce to implement “find the five most popular pages”?

Hint:

- We can chain multiple MapReduce tasks together

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

Restrictiveness of MapReduce Programming Model

Question#2:

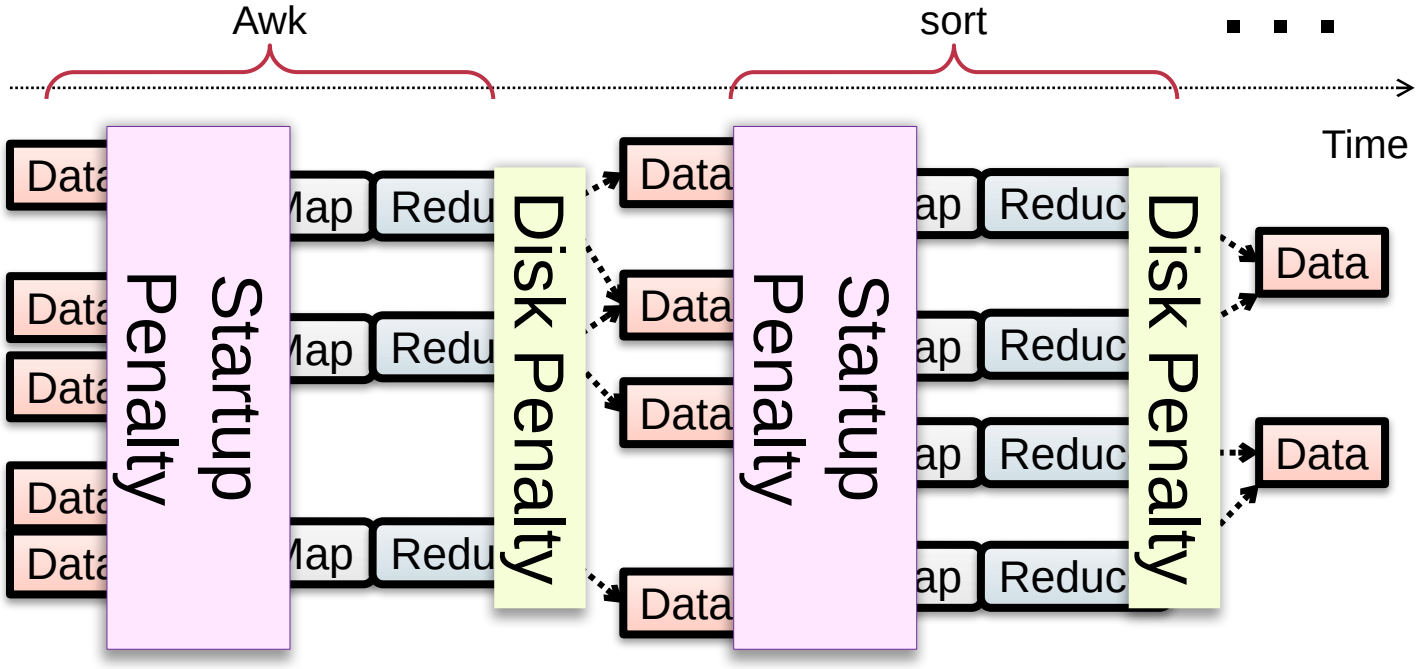
- Is chaining multiple MapReduce tasks a good solution?
  - E.g., programming is not easy
  - Fault tolerance of multiple map-reduce tasks is not supported, should be handled by the users
  - Performance cost due to chaining

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

Performance issues of multiple-sages execution

MapReduce runtime is not optimized for **iteration**

– Persistent I/O (e.g., GFS)

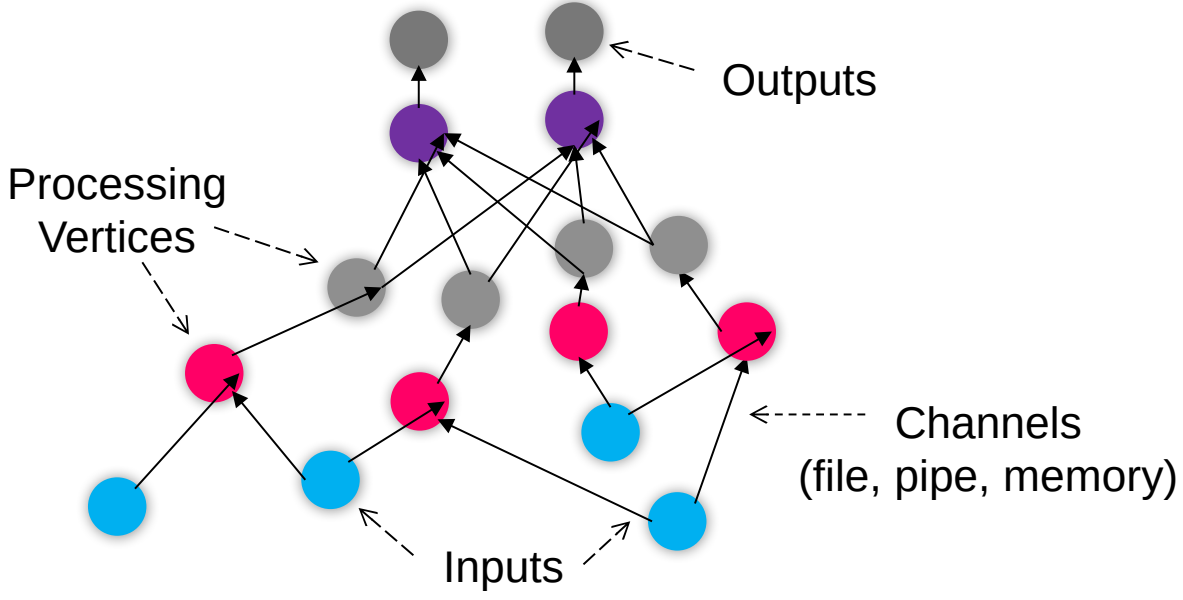


From MapReduce to DAG (Computation graph)

The computation graph abstraction

Computations are expressed as a graph (Directly acrylic graph)

- Vertices are computations (or data)
- Edges are communication channels
- Each vertex has several input and output edges

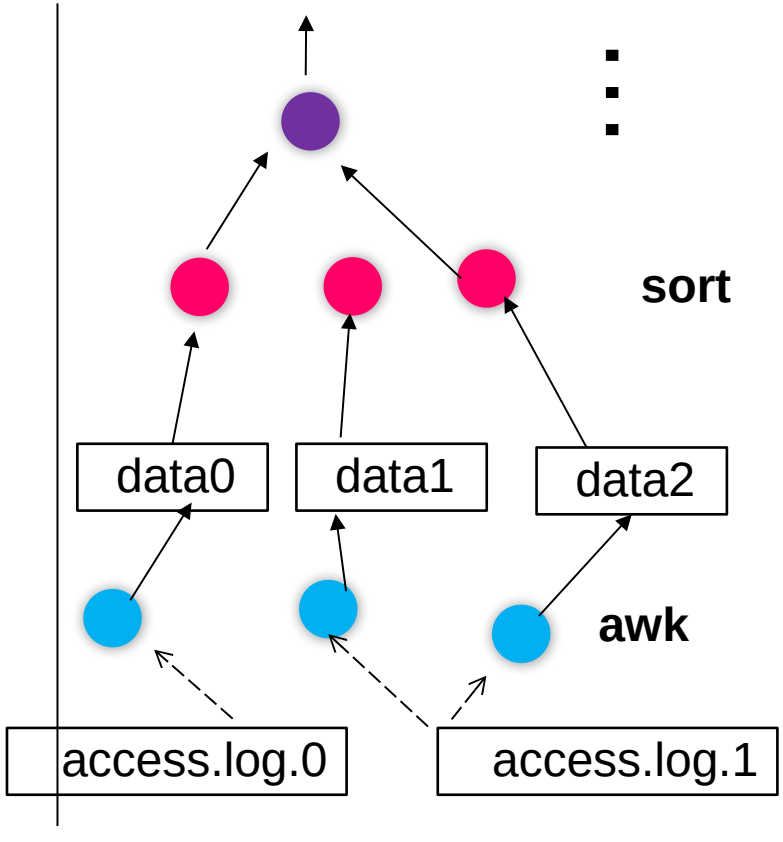


Dataflow graph can support a wide-range of jobs

Distributed computing Job = Directed Acyclic Graph (DAG)

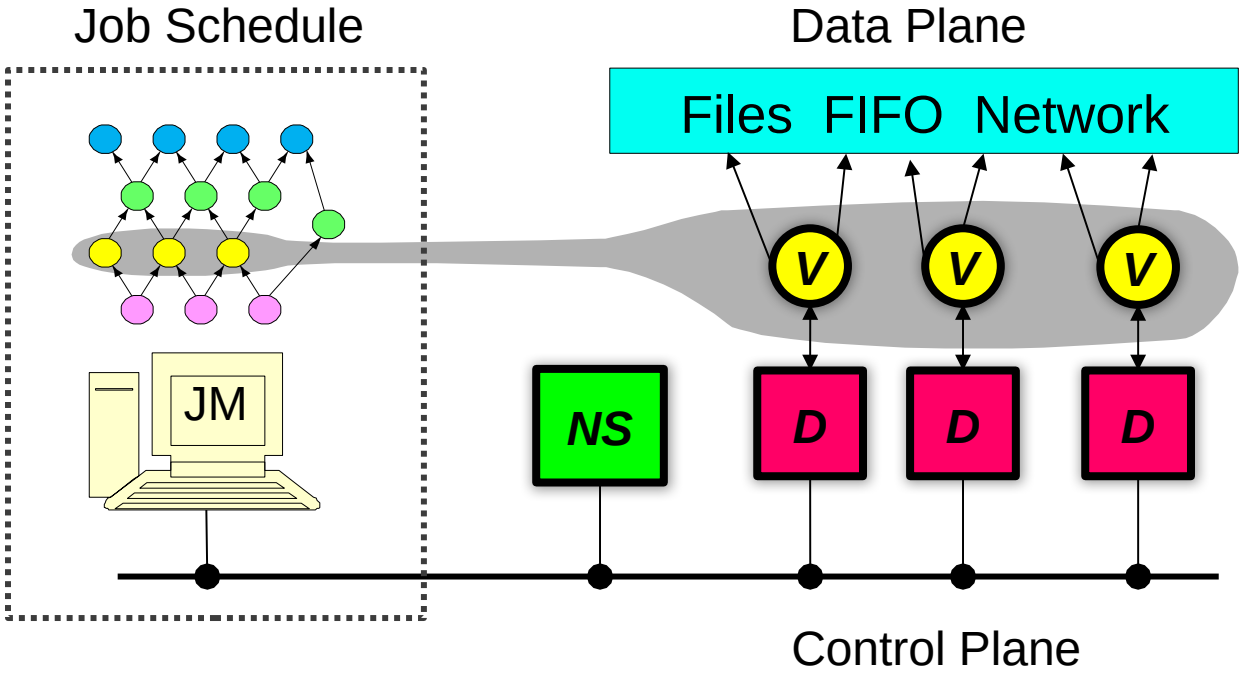
Back to our log analysis example

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



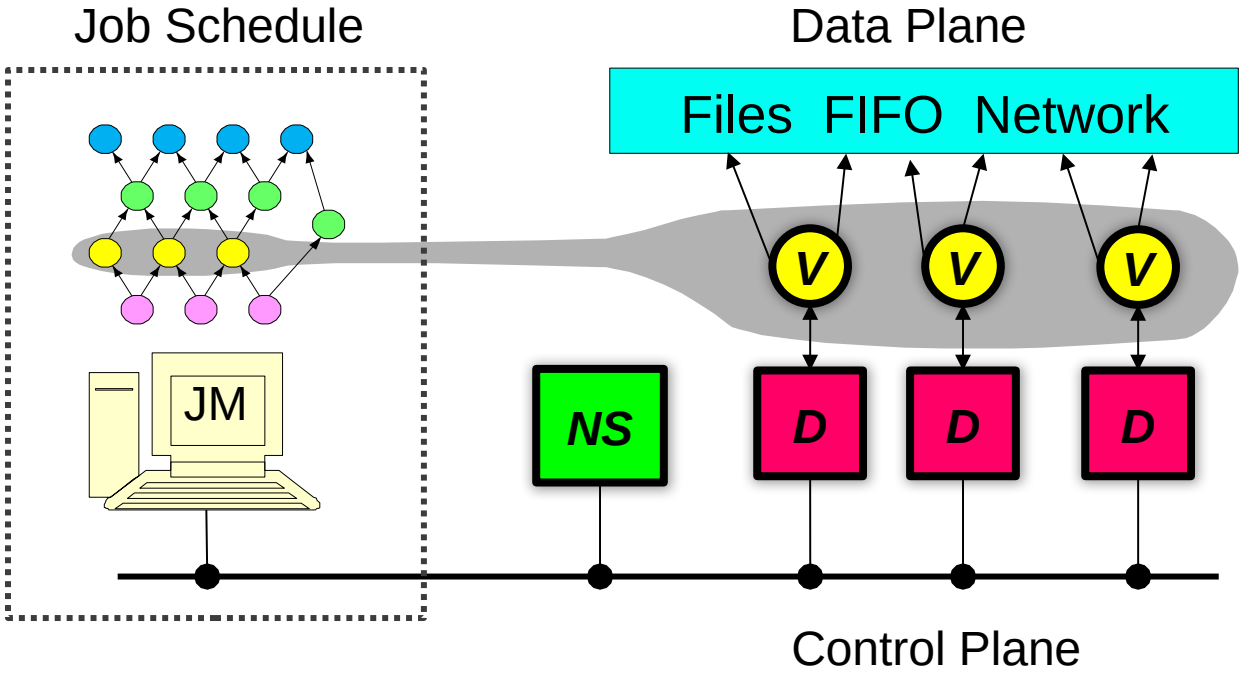
A typical DAG runtime

Vertices (V) run arbitrary app code, exchange data through TCP pipes etc., and communicate with JM to report status



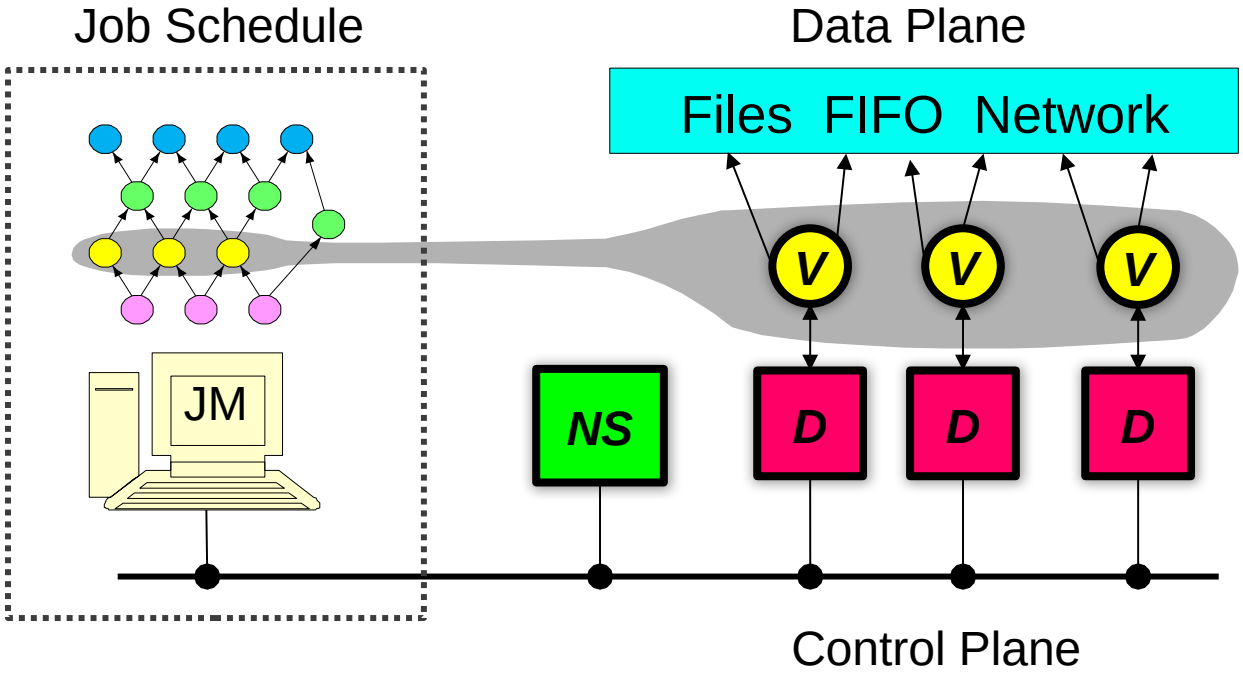
A typical DAG runtime

Job Manager (JM) consults name server (NS) to discover available nodes, and maintains job graph and schedules vertices



A typical DAG runtime

Daemon process (D) executes vertices



Scheduling in job manager

General scheduling rules

- Vertex can run anywhere once all its inputs are ready
  - Prefer executing a vertex near its inputs **(locality)**

Fault tolerance

- Vertex **fails** 🗹 run it again
- Vertex's inputs are gone 🗹 run upstream vertices again **(recursively)**
- Vertex is **slow** 🗹 run another copy elsewhere and use output from whichever finishes **first**

What if the vertex execution is **non-idempotent** ?

Dryad proposes the DAG as distributed computing abstraction

Created by Microsoft (EuroSys'07)

- Authors: Michael Isard, Andrew Birrell, et al.

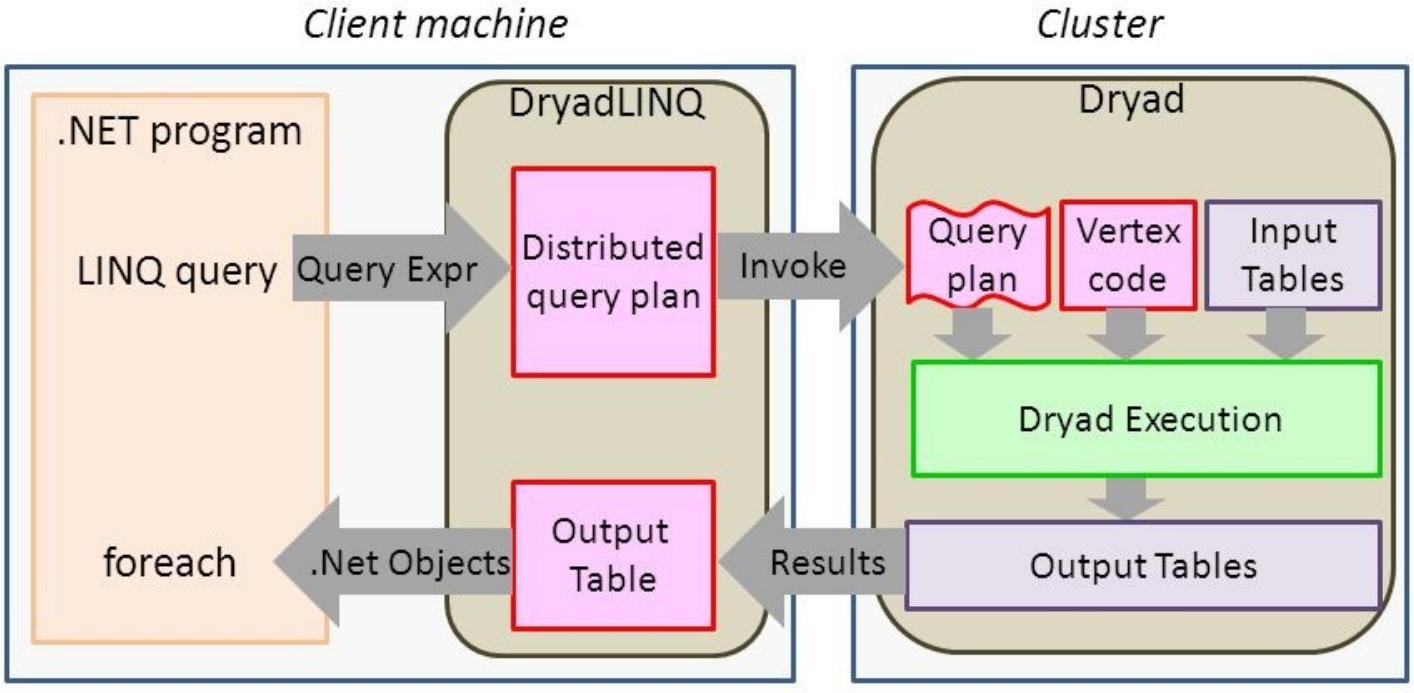
Similar goals as MapReduce

- A **general-purpose** distributed execution engine for coarse-grain data-parallel applications
- Automatic management of scheduling, distribution, and fault tolerance

But needs application-specific semantic to split the nodes

- Otherwise, the graph fallback to a chain, so there is no parallelism
- A little harder than MapReduce, but also hides the distributed execution details

DryadLINQ: Dryad + LINQ (SQL like)



More friendly to traditional data programmers: use LINQ (SQL-like language)

- Hide the generation of a DAG that has high concurrency

Summary

> Dryad lets developers easily create large-scale distributed apps without requiring them to master any concurrency techniques beyond being able to draw a graph of the data dependencies of their algorithms.
>
> *-- Michael Isard*

Sacrifice some architectural **simplicity** compared with MapReduce system design

Provide more **flexible** abstract to developers expressing their code as a **DAG**

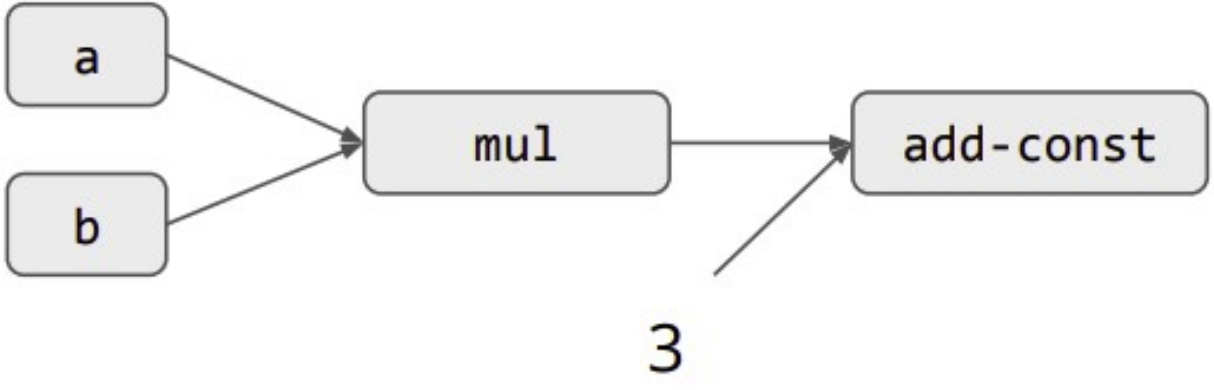
How to express AI program to distributed framework?
Dataflow graph!

The Language of AI: Computation Graph

Nodes represents the computation (operation)

- E.g., Matrix multiplications, softmax operator, activation functions, variables

Edge represents the data dependency between operations

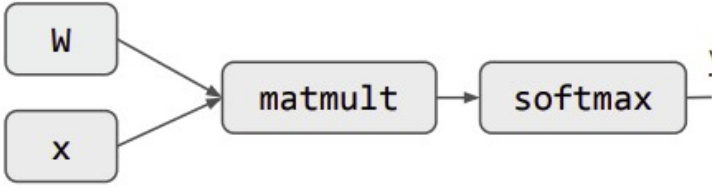


Example: computational Graph for a * b +3

Computational Graph Construction by Step

Example NN: y = softmax(XW) (assuming using Tensorflow-like framework)

```
4  # Create the model
5  x = tf.placeholder(tf.float32, [None, 784])
6  W = tf.Variable(tf.zeros([784, 10]))
7  y = tf.nn.softmax(tf.matmul(x, W))
```



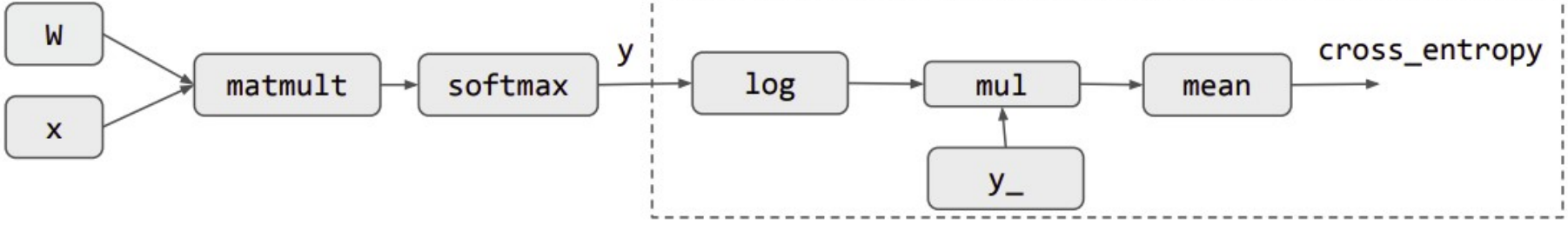
Computational Graph Construction by Step

Cross-entropy loss: #labels (10)

$$\text{Cross-Entropy Loss} = -\frac{1}{N}\sum_{j=1}^{N}\sum_{i=1}^{C} y_{ji} \cdot \log(\hat{y}_{ji})$$

```
4  # Create the model
5  x = tf.placeholder(tf.float32, [None, 784])
6  W = tf.Variable(tf.zeros([784, 10]))
7  y = tf.nn.softmax(tf.matmul(x, W))

9  # Define loss and optimizer
10 y_ = tf.placeholder(tf.float32, [None, 10])
11 cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), reduction_indices=[1]))
```

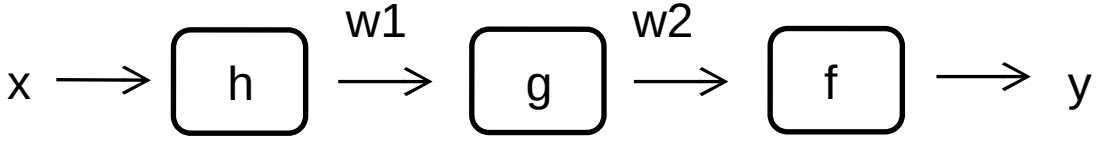


Benefit of computation graph: Automatic Differentiation

Basically, based on the chain rules

$$y = f(g(h(x))) = f(g(h(w_0))) = f(g(w_1)) = f(w_2) = w_3$$

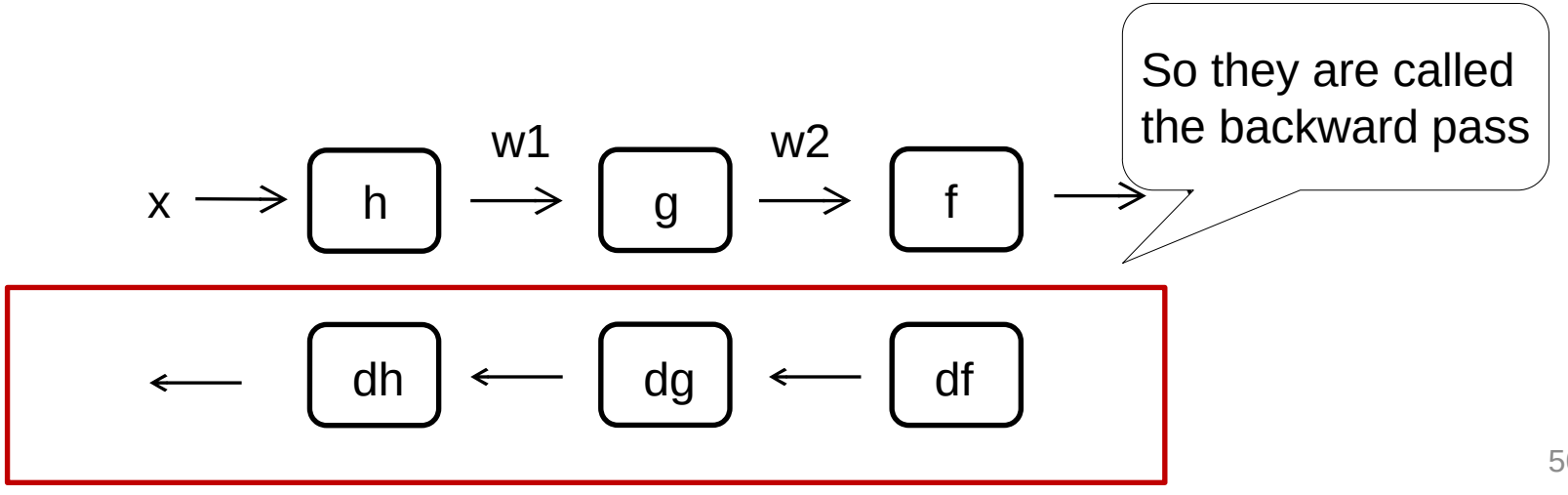
$$w_0 = x$$
$$w_1 = h(w_0)$$
$$w_2 = g(w_1)$$
$$w_3 = f(w_2) = y$$



Benefit of computation graph: Automatic Differentiation

Basically, based on the chain rules

$$\frac{dy}{dx} = \frac{dy}{dw_2}\frac{dw_2}{dw_1}\frac{dw_1}{dx} = \frac{df(w_2)}{dw_2}\frac{dg(w_1)}{dw_1}\frac{dh(w_0)}{dx}$$

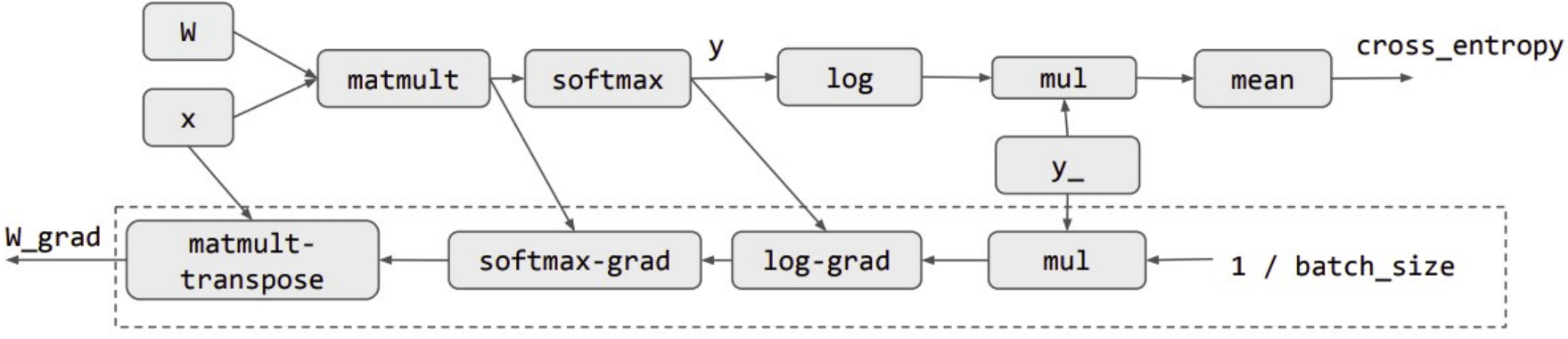


Benefit of computation graph: Automatic Differentiation

Based on the computation graph, we can easily compute the derivative

- Assumes each operator (node) has a corresponding derivative implementation
- We can trivially combine them together with the chain rule

```
15 W_grad = tf.gradients(cross_entropy, [W])[0]
```



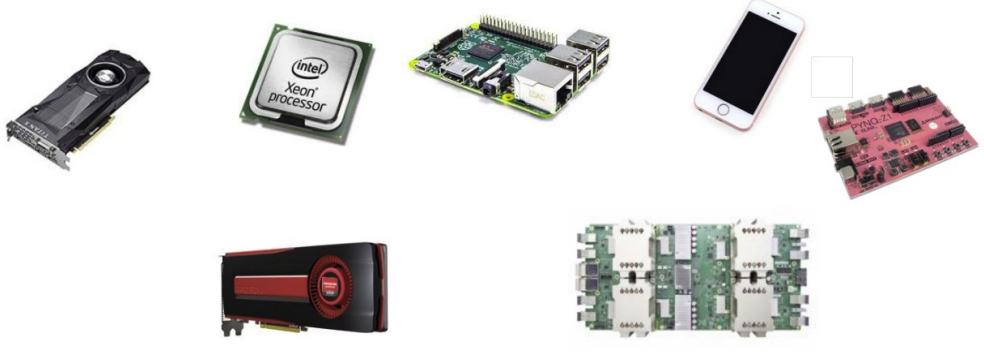
Another benefit: tame heterogeneous hardware

Hardware accelerators to accelerate AI computations

- E.g., GPU, TPU, NPU
- Non-trivial to program! (Recall the previous lecture, e.g., CUDA)

What we have an operator implemented in a specific hardware?

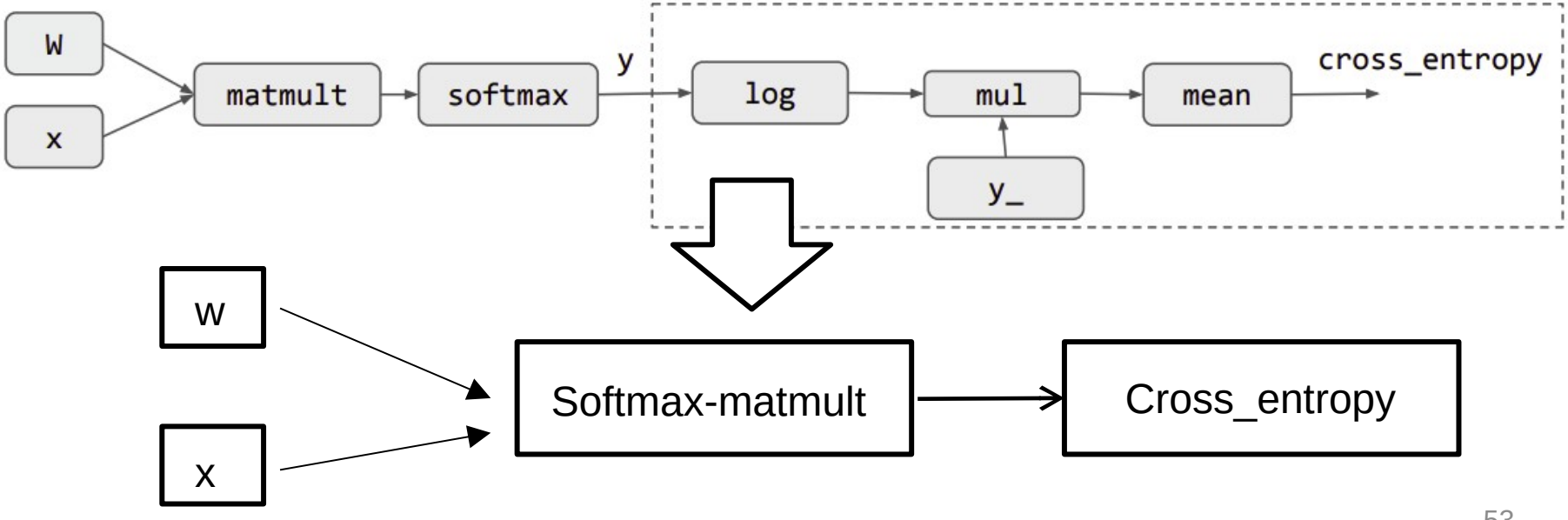
- Can trivially run our computation graph on these hardware
    - E.g., using a compiler such as TVM



One thing to note: graph fusion

We can fuse multiple graph nodes to form a larger node

- Benefit: reduce communications
  - Both internal (within a device) & cross-node (distributed)



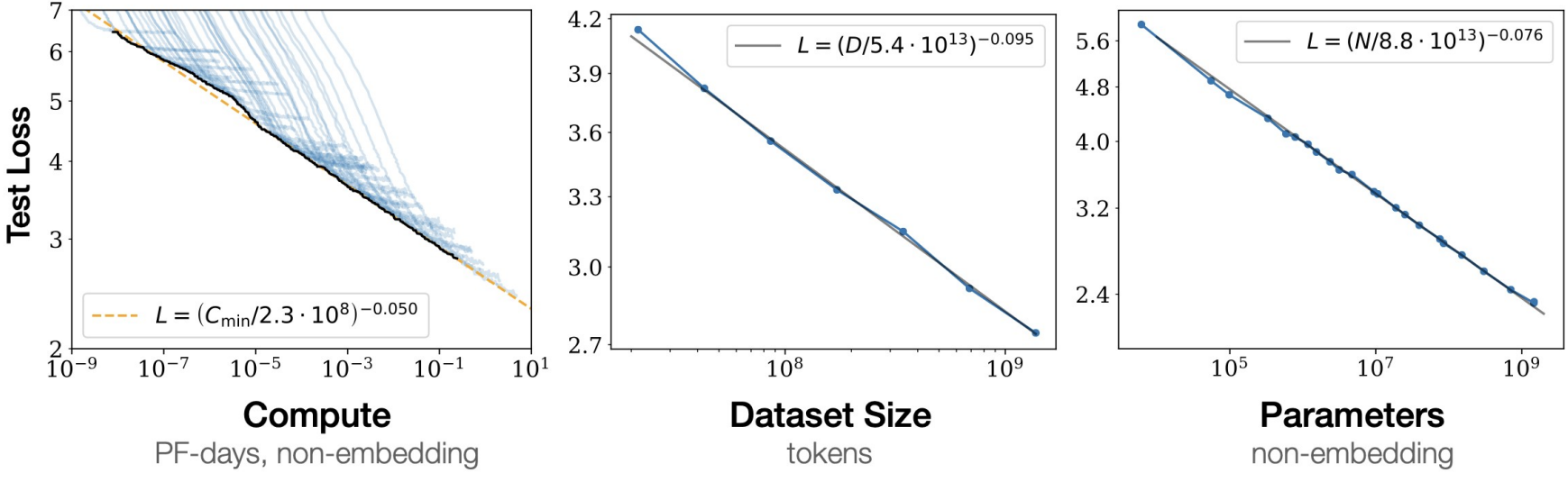
Distributed training

Why we need distributed training? Scaling low

Massive data needed (the larger, the better)

Massive model needed (the larger, the better)

Massive computation power needed



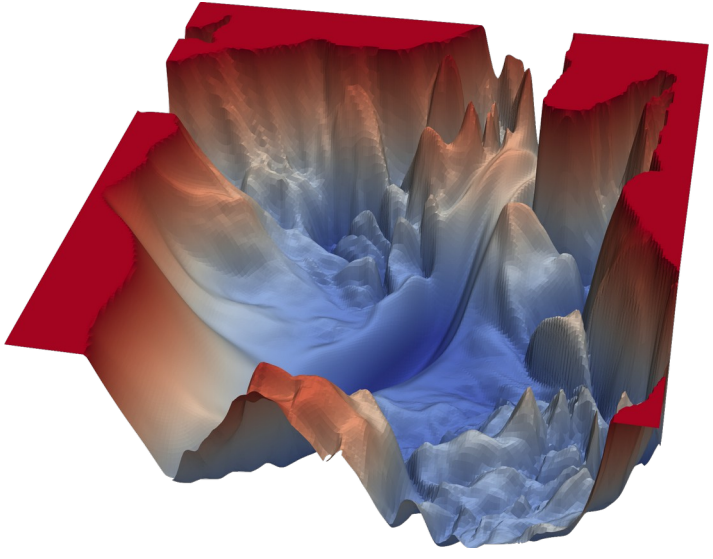
Ideal Metric of Success for Efficient Training

$$\left(\frac{\text{“Learning”}}{\text{Second}}\right) = \left(\frac{\text{“Learning”}}{\text{Record}}\right) \times \left(\frac{\text{Record}}{\text{Second}}\right)$$

Convergence
Machine Learning
Property

Throughput
System
Property

Review: Stochastic Gradient Descent



Learning rate

Two key elements:

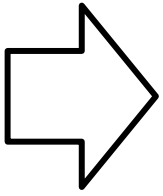
- The computed gradient: the direction
- The learning rate: how big a step do we take?

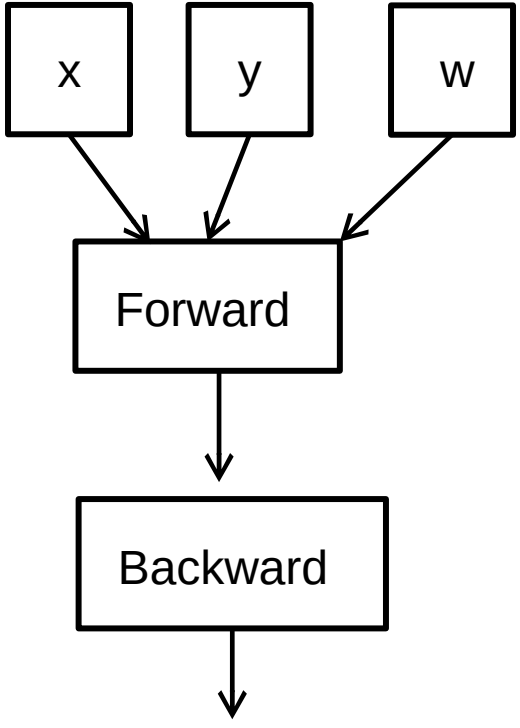
Stochastic Gradient Descent into computation graph

X: input features

Y: labels in supervised learning

Learning rate



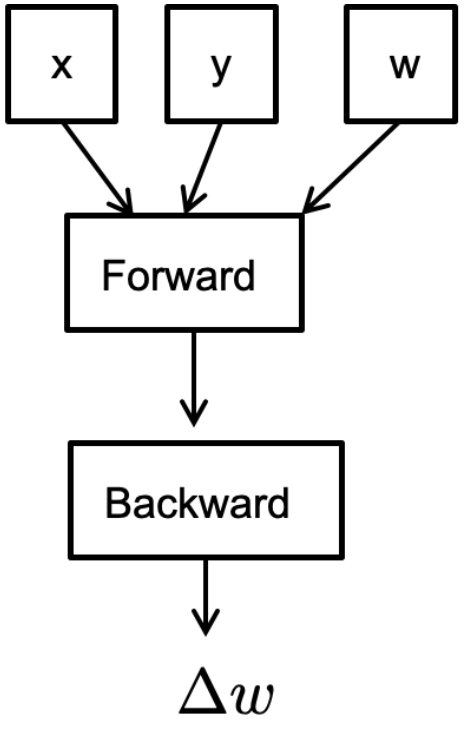


Pseudocode for Stochastic Gradient Descent

```
// execute on a master
for iter in num_iters:
```

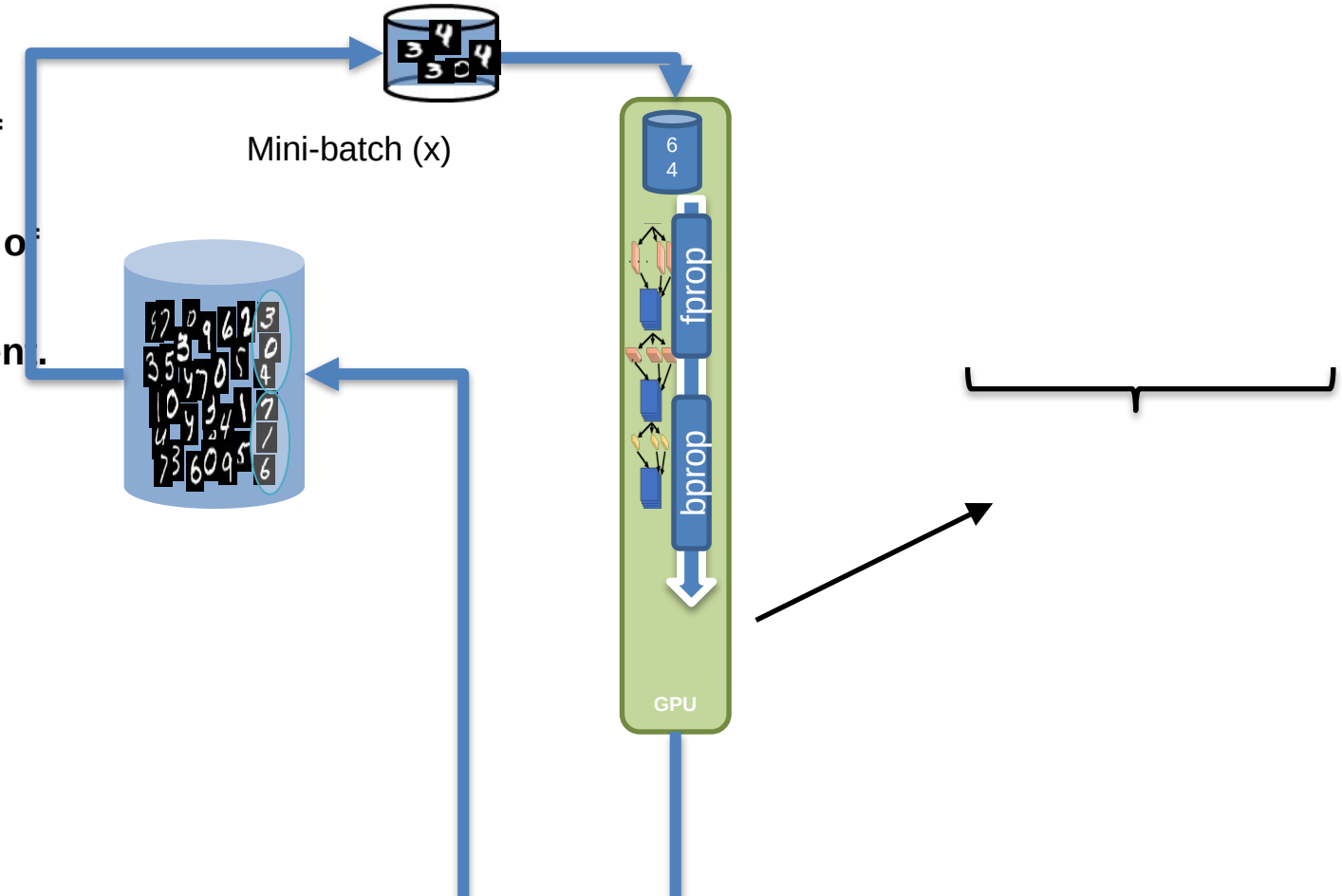
iter

iter+1 iter



Overview of Stochastic Gradient Descent in action

In every iteration of SGD we load a random mini-batch of training data and compute the gradient.

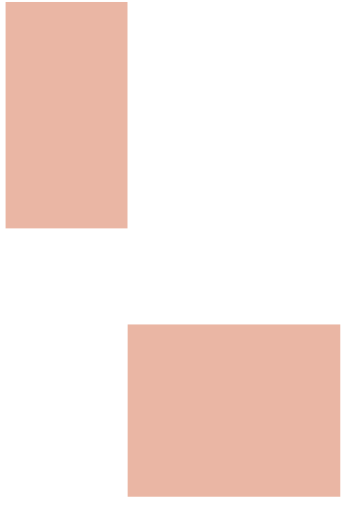


Parallelization Opportunities

Data Parallelism: *Distribute the processing of data to multiple PEs.*

Model Parallelism: *Break the model and distribute processing of every layer to multiple PEs*

For either approach it is also possible to use ***synchronous*** *or* ***asynchronous*** *updates*

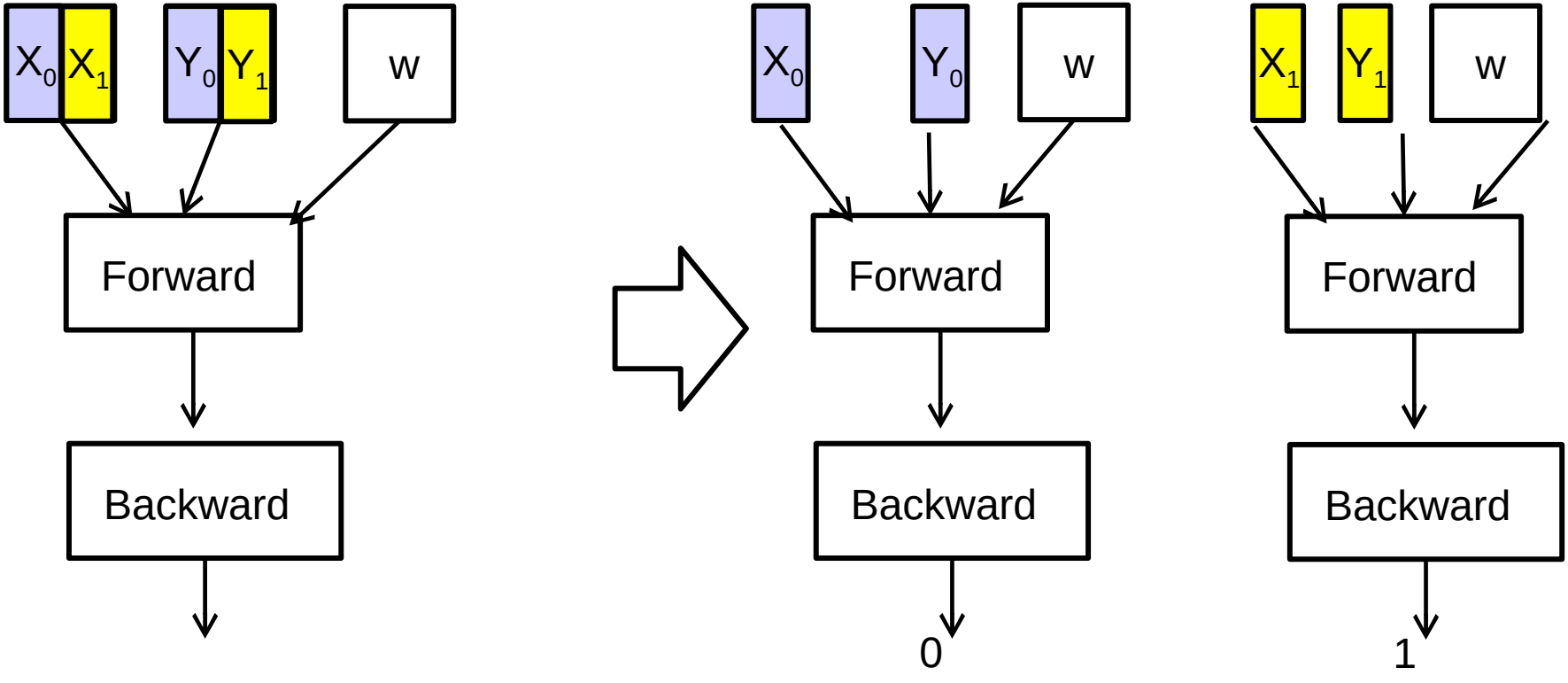


(Sync) data parallel

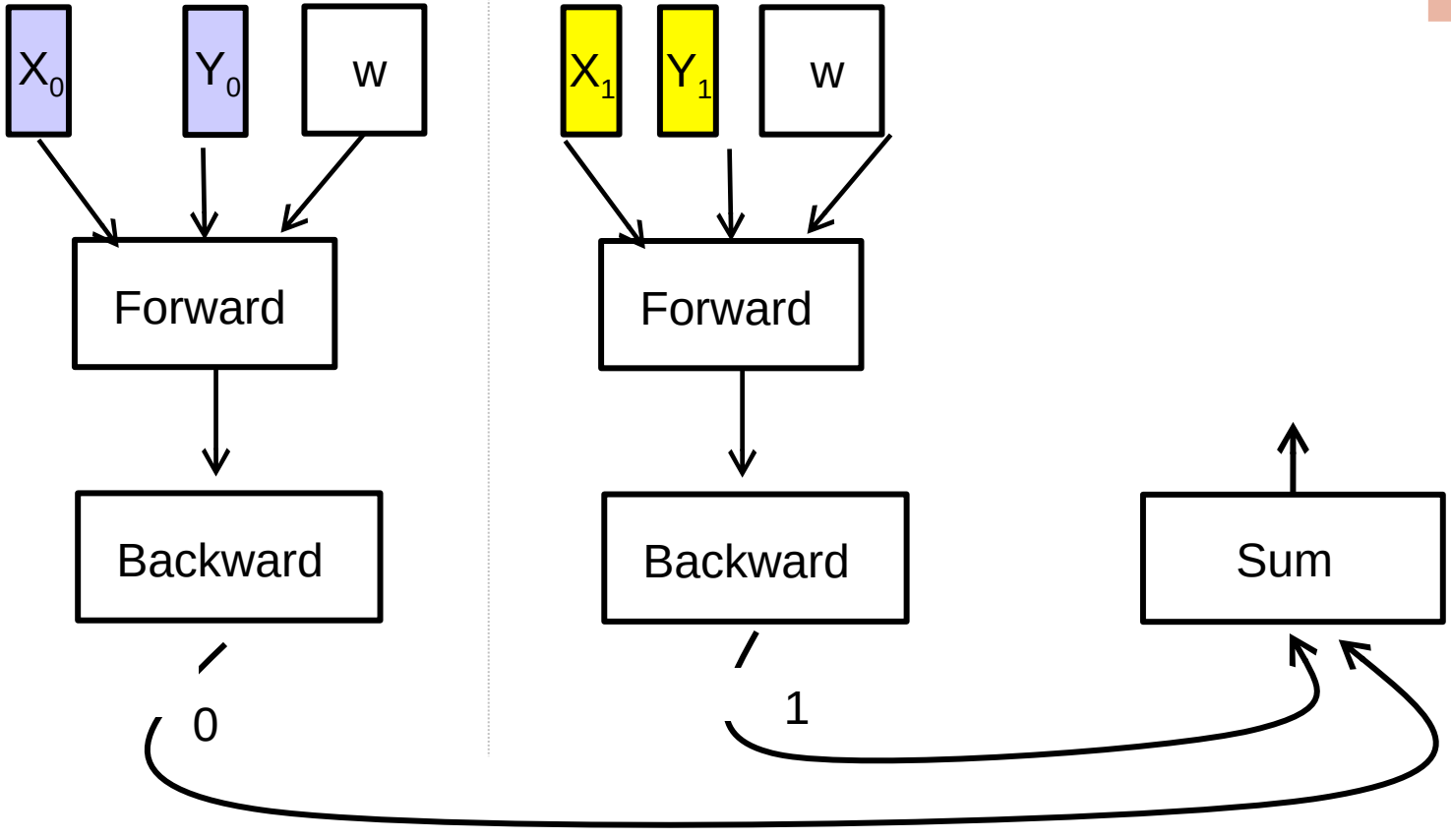
Synchronous Data Parallelism

Parallelize on the x (and y)

– Observation: + can be easily parallelized



Synchronous Data Parallelism

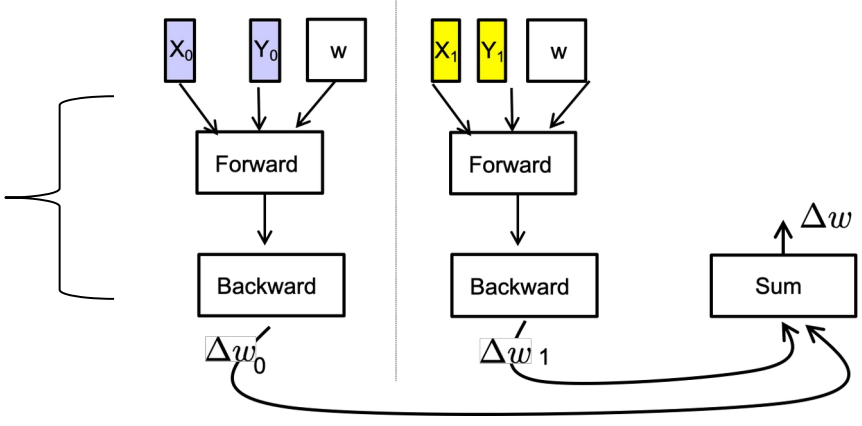


Pseudocode for (iterative) data parallel in action

```
// execute on a master
for iter in num_iters:
```

```
      iter+1 iter
```

iter



The master will do the coordination

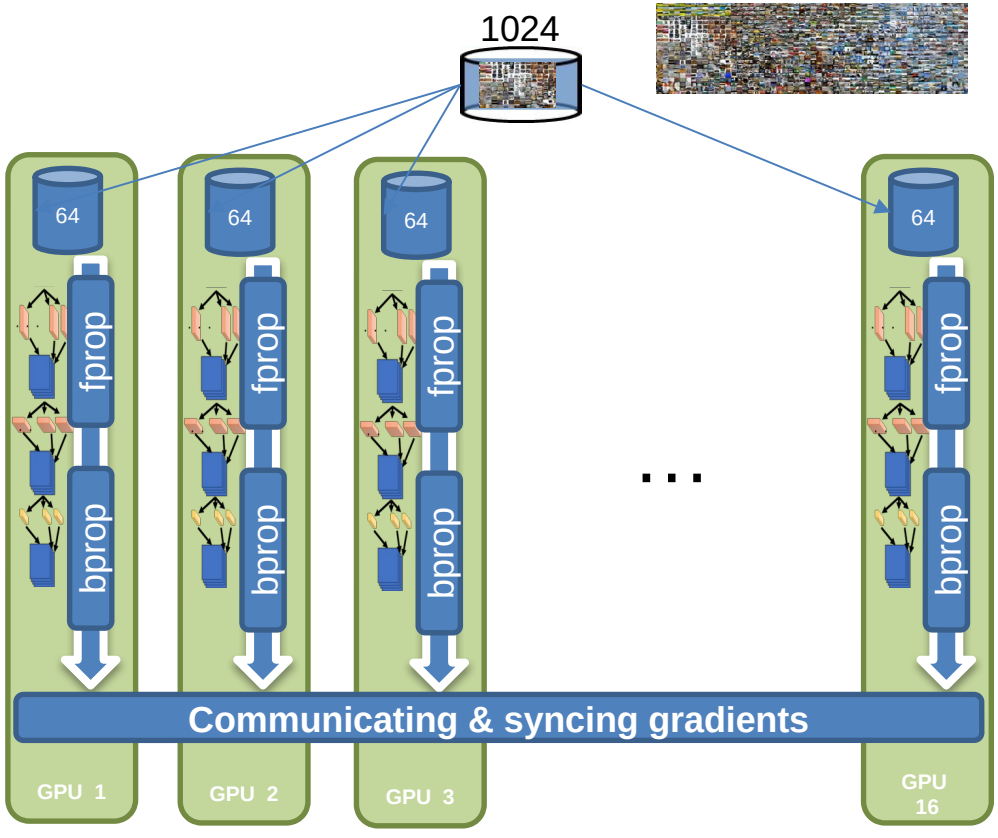
- Similar to the Job manager in Dryad

Synchronous Data Parallelism w/ BSP

Store the entire model (W) on each processor

- Then distribute the batch evenly across each processor
- E.g., 64 images per GPU

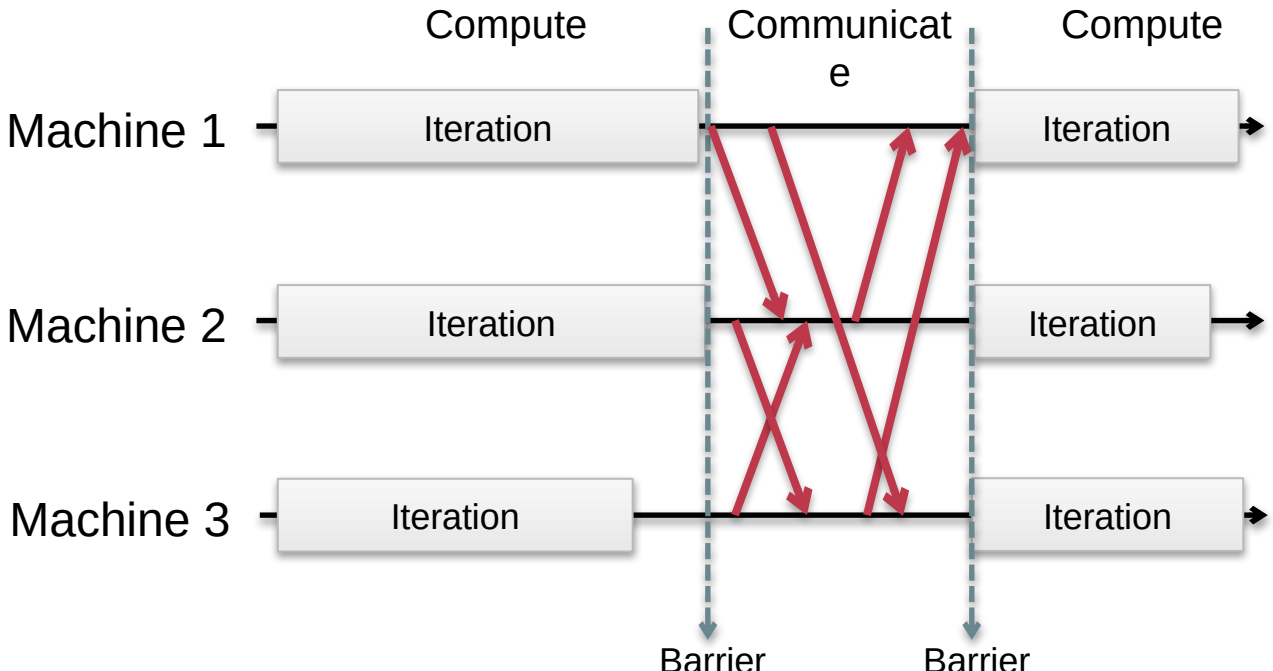
Question: how to coordinate between iterations?



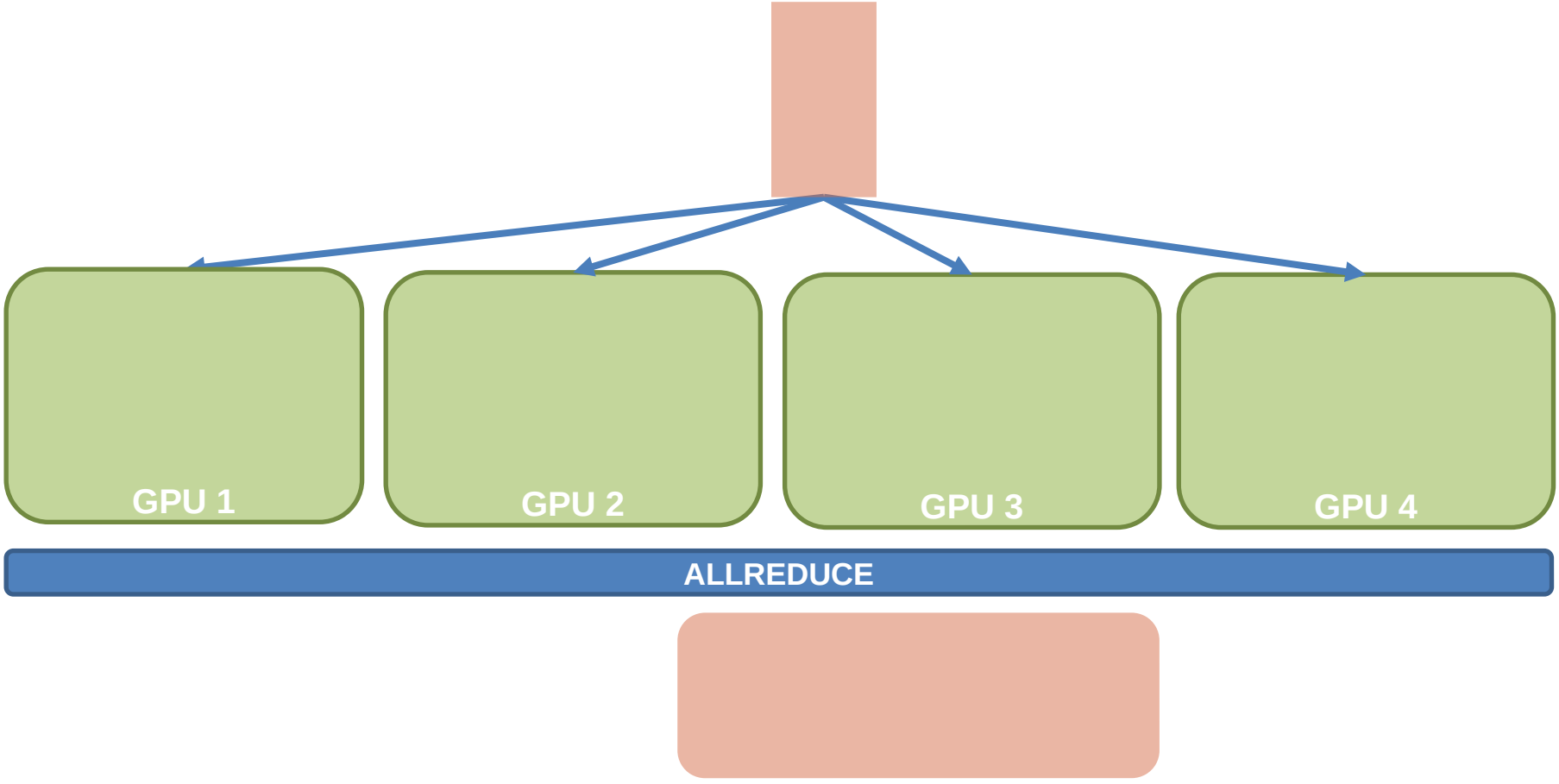
How to parallelize the partitioned graphs? BSP

Bulk Synchronous Parallel (BSP) Execution

- Using a strict barrier to coordinator processes in a distributed computing
- Barrier can be implemented in a centralized way (master) or decentralized



Key operator: allreduce



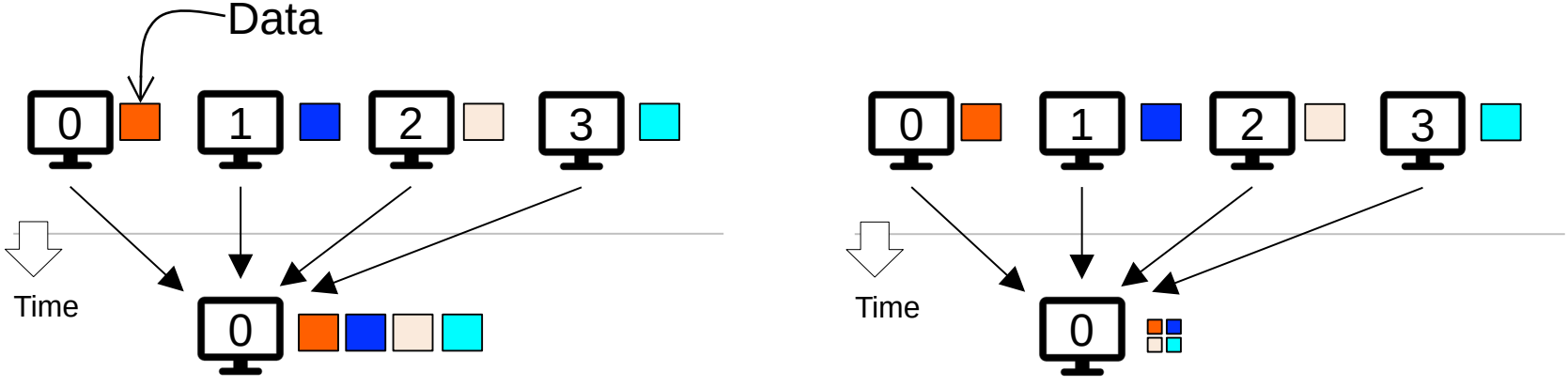
Communication primitives in distributed computing

Gather: Collect data from all the other processes

- Example: [1][2][3][4] -> (gather) -> [1,2,3,4]

Reduce: Gather and aggregate a piece of data from all the other processes

- Example: [1][2][3][4] -> (reduce) -> [1+2+3+4] = [10]



Gather vs. Reduce

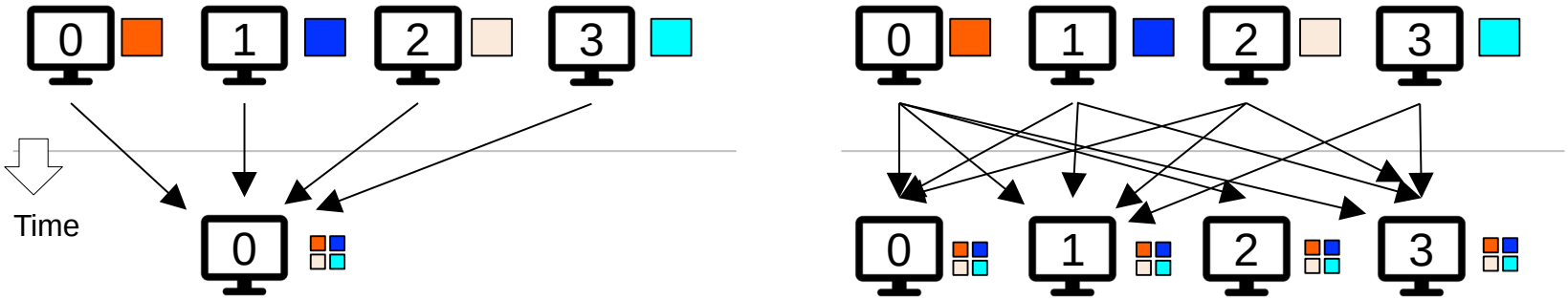
Communication primitives in distributed computing

Reduce: Gather and aggregate a piece of data from all the other processes

- Example: [1][2][3][4] -> (reduce) -> [1+2+3+4] = [10]

Allreduce = reduce on all processes

Question: how to efficiently implement allreduce?



Reduce vs. AllReduce

* The messages of allreduce may be missing due to space limitations

System requirements for AllReduce

Overhead analysis: computation + network

The overhead of computation is typically small

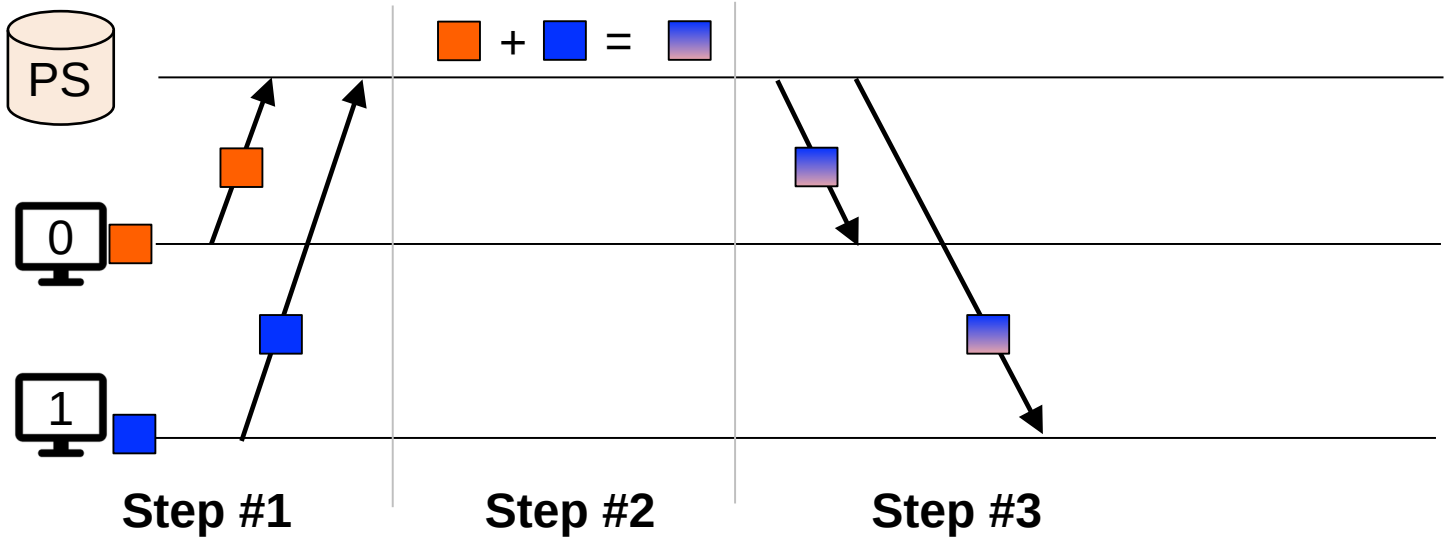
- Compute a sum operation on device is highly optimized, e.g., SIMD, GPU, TPU, etc.

Goal: reduce network overhead

- Reduce bytes sent per-message
- Reduce concurrent messages sent to a server

Try #1: parameter server

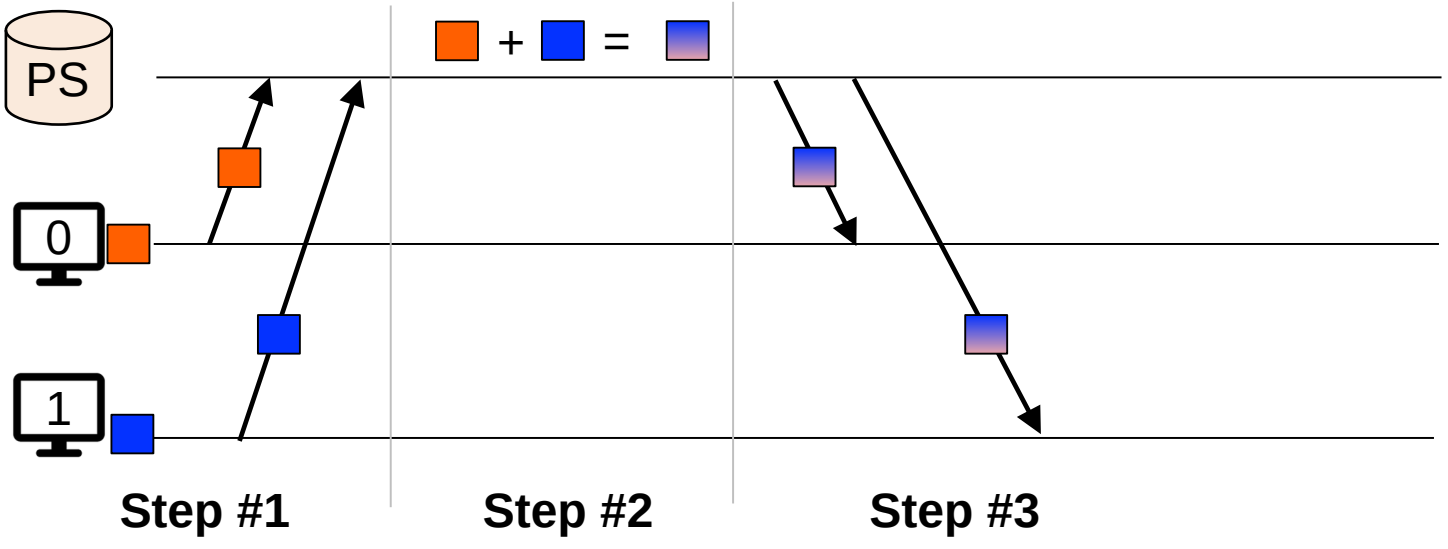
1. Each process pushes the data to the parameter server (PS)
2. After receiving all data, aggregate the data at the PS
3. The PS pushes the data back to the processes



Allreduce with a single parameter server (PS)

Question: what is the network requirements at each stage?

- Step #1: O(1) at each process, O(n) at the PS
- Step #3: O(1) at each process, O(n) at the PS



Allreduce with a single parameter server (PS)

Problem: huge bandwidth requirement at the PS (O(P * N))

- As well as huge contention at the master machine

Example: training ResNET50 on V100

- ResNET50: 24.4 M params ~= 97.5MB storage
- On V100, each node can train 3 iterations / second (forward + backward path), w/o communication
- #Processing Node: 256
- Total requirement bandwidth: 256 * 3 * 97.5 = 73.1GB/s

The state-of-the-art NIC: 400Gbps RDMA ~= 50GB/s

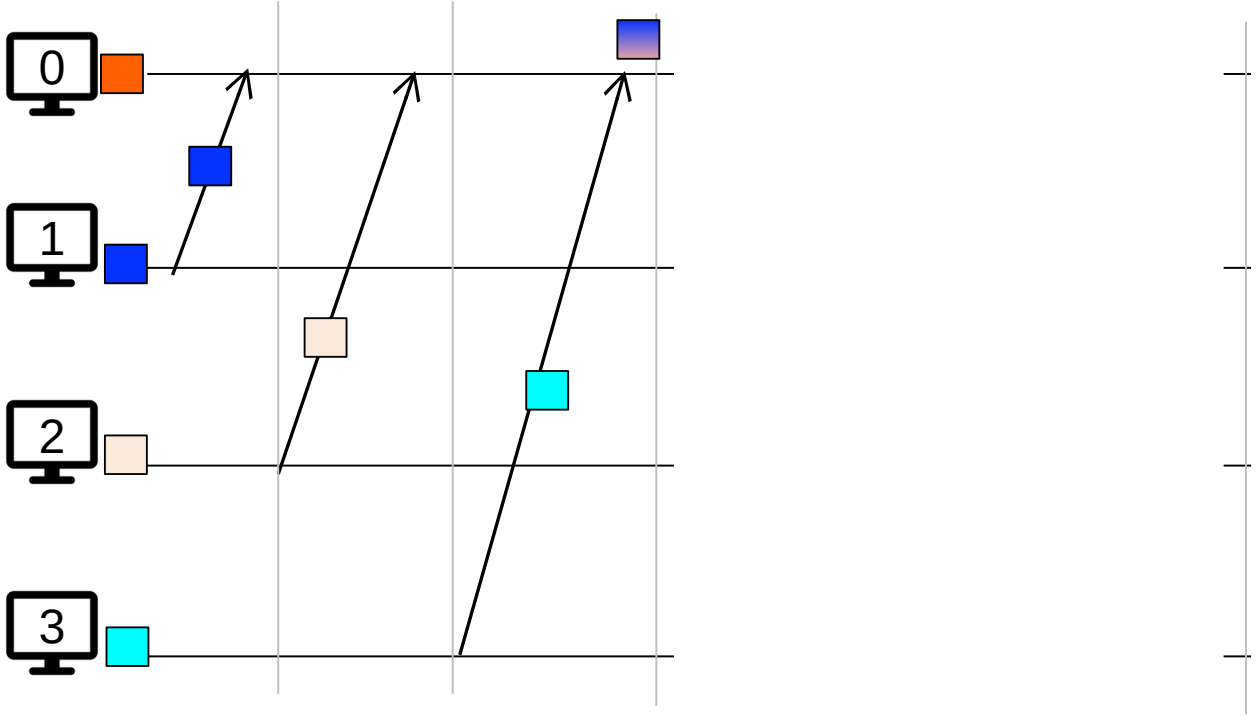
- The network would become the bottleneck!

Goal: reduce messages send to a centralized server

Try #2: De-centralized approach for allreduce

Each node finishes its allreduce, one-by-one

- E.g., with a careful designed scheduler



Analysis of the naïve decentralized reduce

N: total parameters, P: # processors

Total communication per-machine

- N * (P – 1) data (Bad …)
- O(1) fan-in (Good!)

Total communication rounds: O(P * P)

- Much higher than the parameter server approach (O(1))

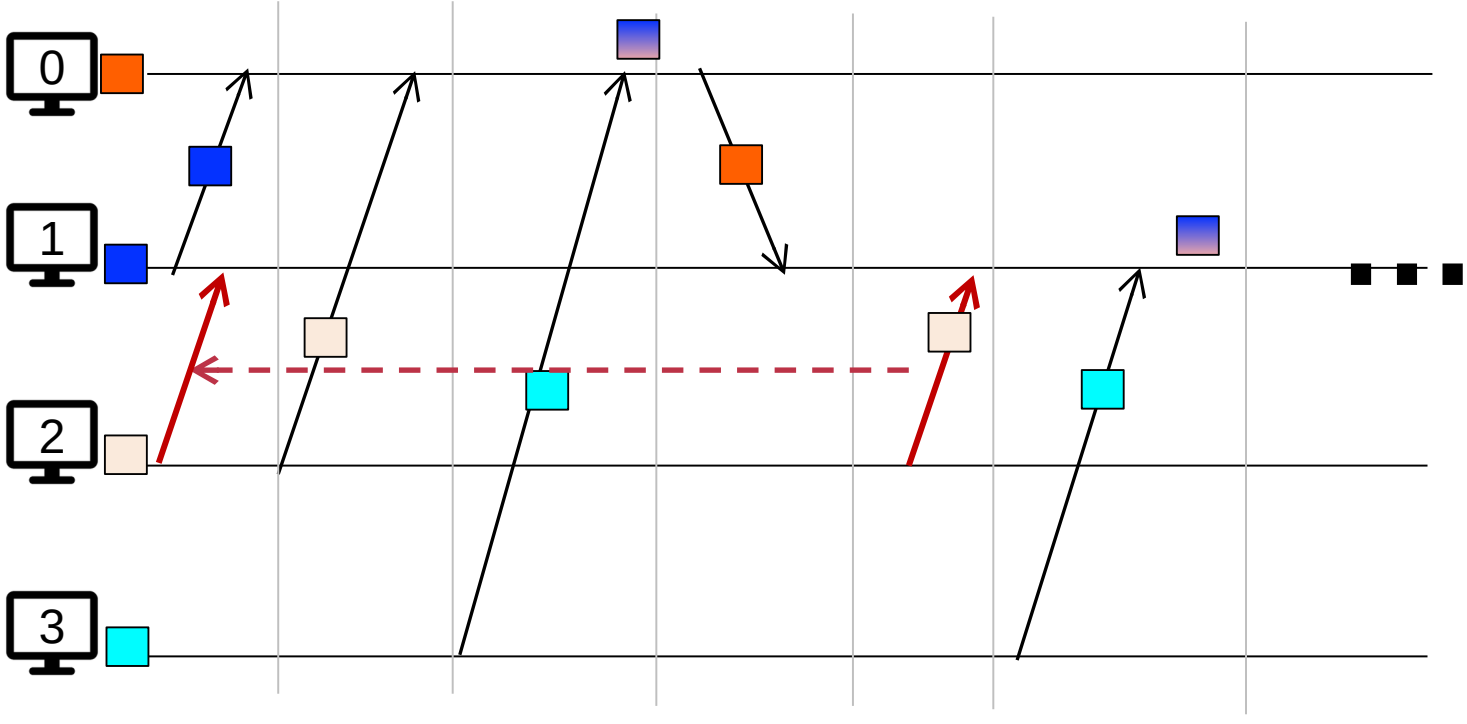
Question

- Can we do better?

Try#2: De-centralized approach for allreduce

Each node finishes its allreduce, one-by-one

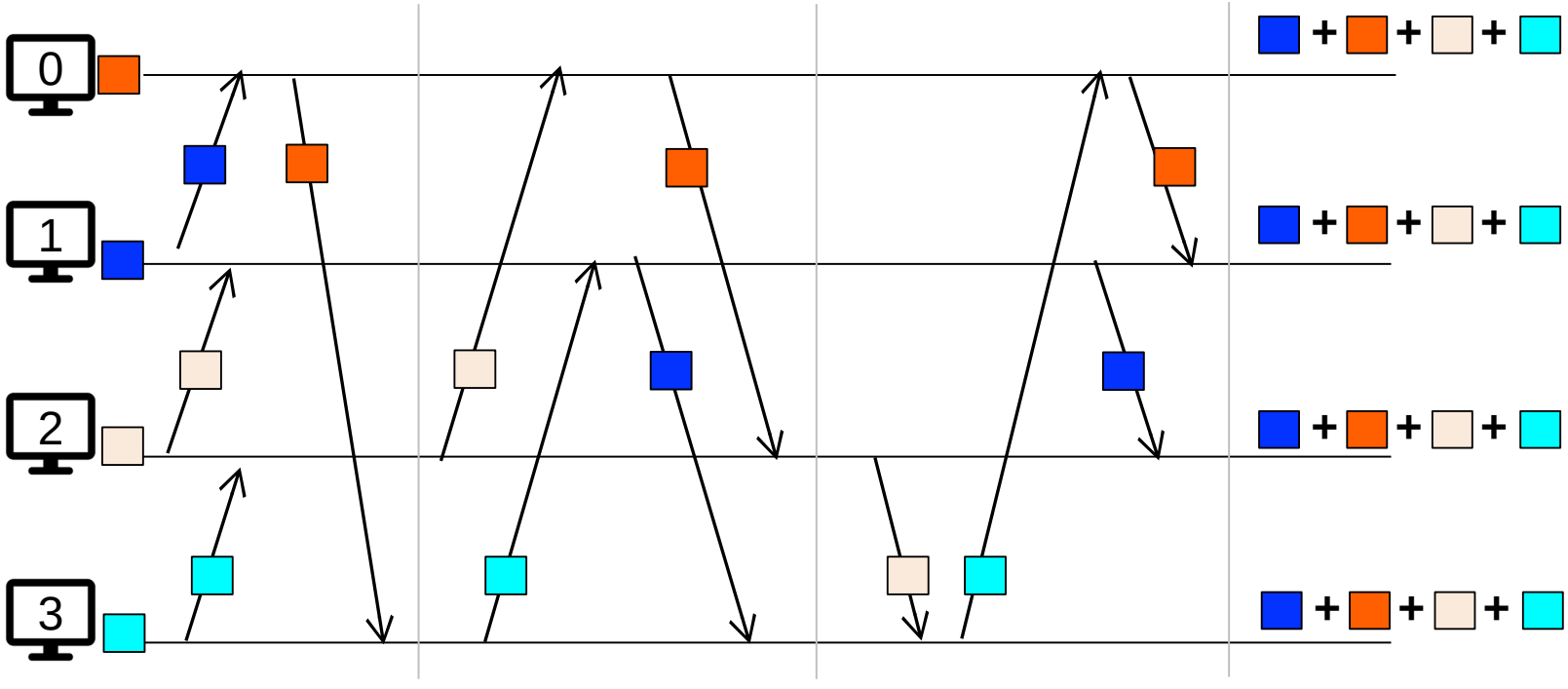
- Observation: communications from future rounds can be moved earlier if w/o contention



De-centralized approach for allreduce

Each node finishes its allreduce, one-by-one

- Communications from future rounds can be moved earlier if w/o contention



Analysis of the naïve decentralized reduce

N: total parameters, P: # processors

Total communication per-machine

- N * (P – 1) data (Bad …)
- O(1) fan-in (Good!)

Total communication rounds: ~~O(P * P)~~ =》 O(P)

- Higher than the parameter server approach (O(1)) , but w/o contention

Question

- Can we do better?

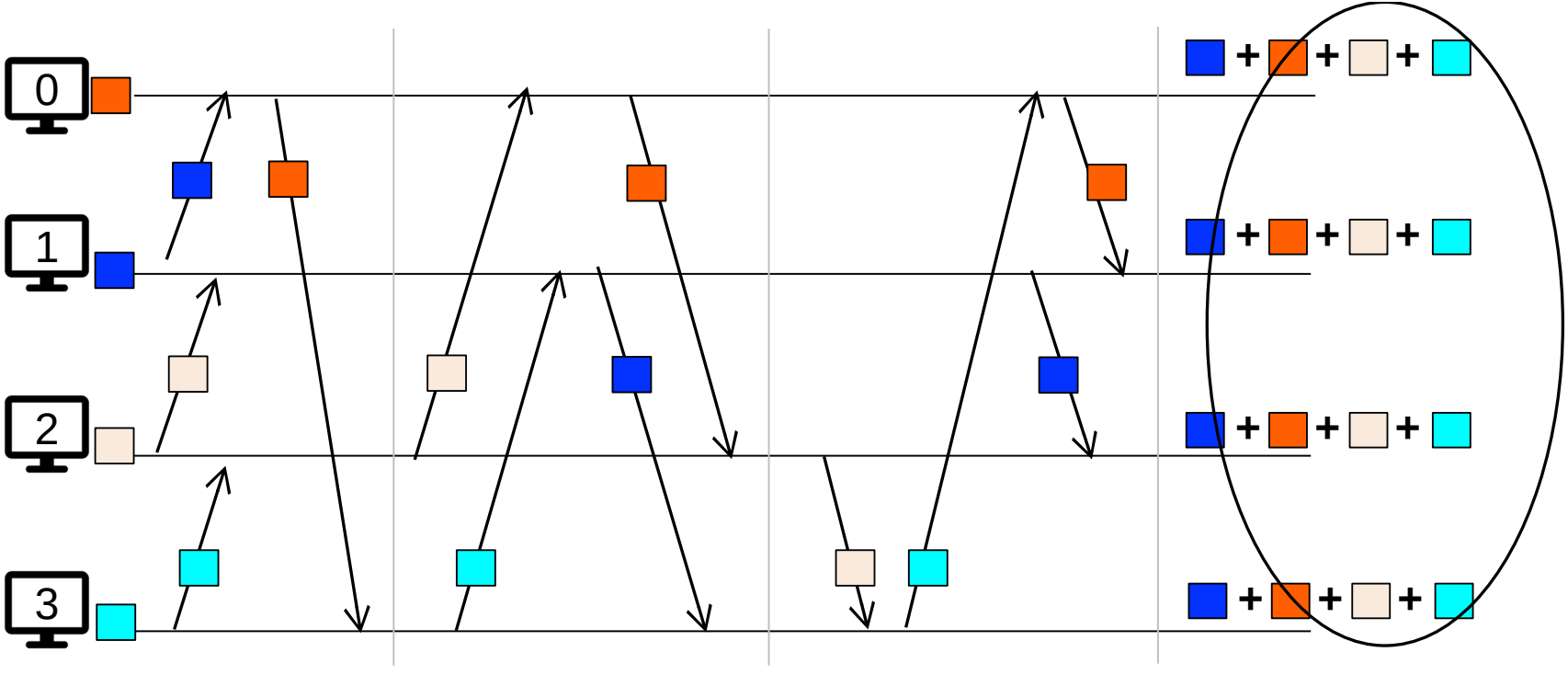
Goal: reduce the payload sent per-process

Idea: partition the data to reduce traffic

Each node finishes its (a partition of its) allreduce, one-by-one

- Finally, using a decentralized approach to gather all the data

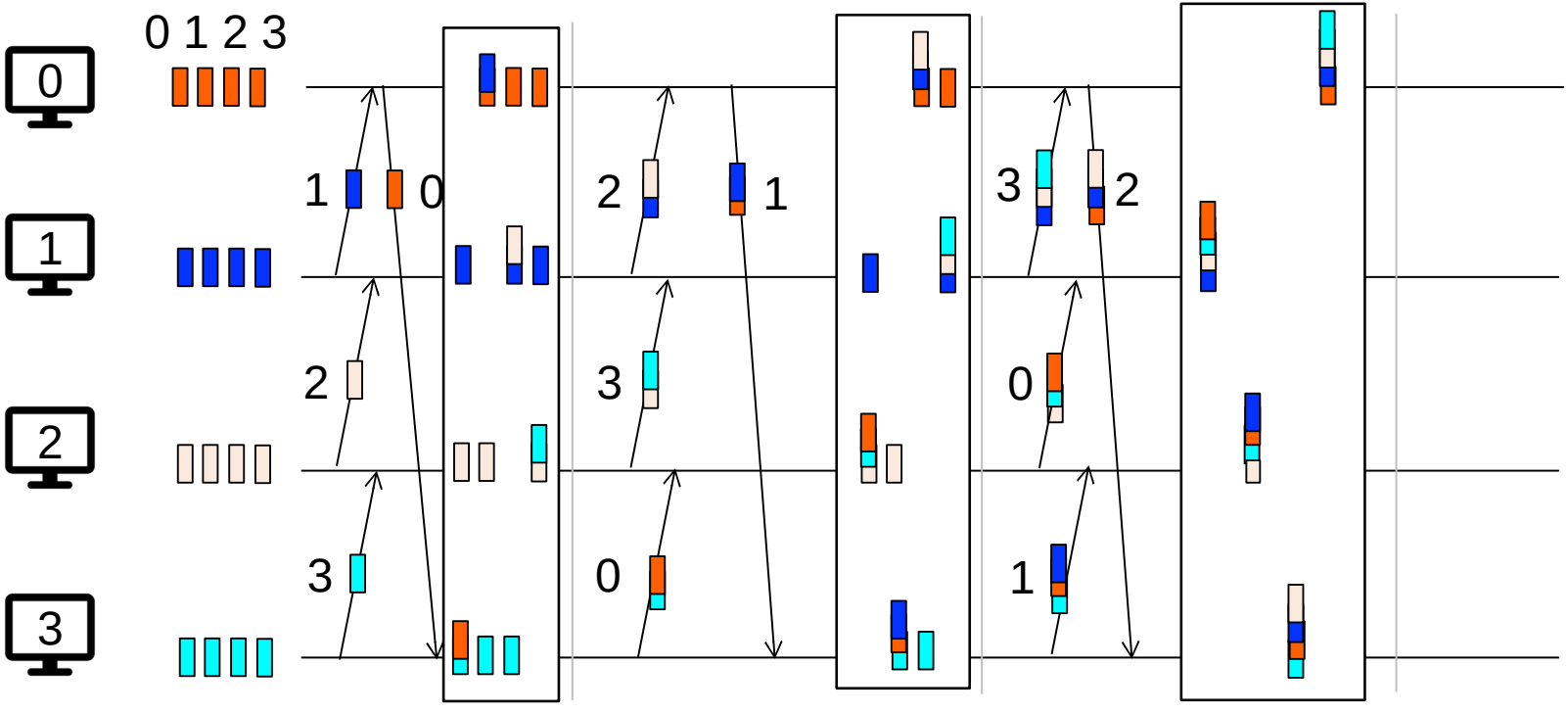
Redundant computations



Try #3: Ring allreduce

Decentralized reduce, each process reduce on a partitioned data (prev/next to it)

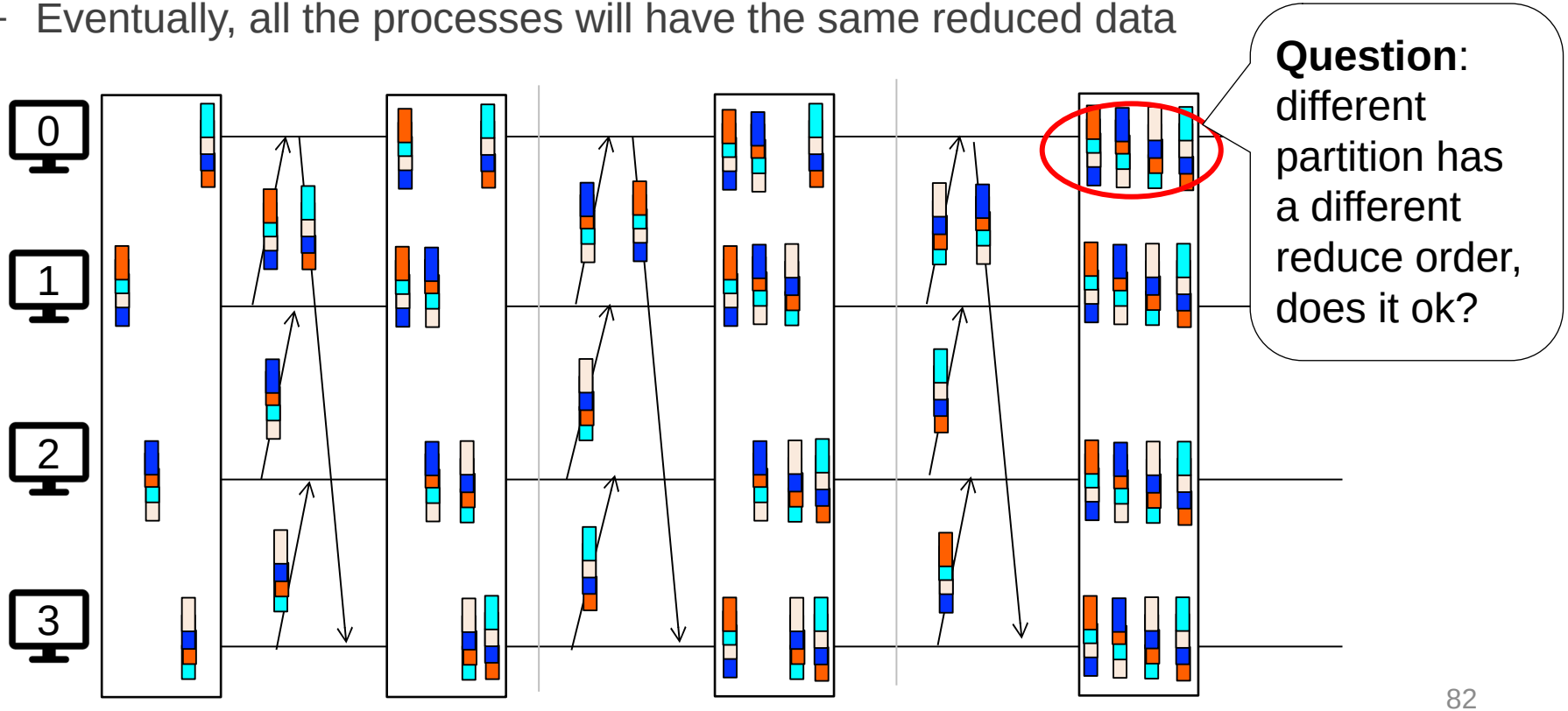
- i.e., each process send one partition data sequentially



Ring allreduce

After the individual reduce, each process can scatter to collect results

- Eventually, all the processes will have the same reduced data



Why ring allreduce can work?

Assumption: reduce op is associative & communicative

- So we can reorder them to reduce sudden loads at a node
- E.g., we don’t need to collect all the data to do the reduce

Otherwise, we cannot reorder the computation

Analysis of the communication cost of ring allreduce

N: total parameters, P: # processors

Total communication per-machine

- 2 * (P-1) * N / P: the same as partitioned
- O(1) fan-in

Total communication rounds: O(P)

- Much higher than the parameter server approach (O(1))
- A trade-off for reducing network contention due to high fan-in

What are the drawback?

- High latency due to extra round

Allreduce: put it all together

Trade-off between rounds vs. fan-in performance

	Round	Peak node bandwidth	Per-node fan-in
Parameter server (PS)	O(1)	O(N * P)	O(P)
Decentralized Allreduce	O(P)	O(N)	O(1)
Ring allreduce	O(2 * P)	O(N/P)	O(1)

More reduce methods exist

Key design goals

- Reduce payload sent per-node
- Reduce fan-in to avoid contention on network resource
- (new) Reduce rounds (grow linearly w/ the number of processors)
- (new) Better suits the underlying hardware (Cloud & HPC)

Still an active research field

- E.g., what if the hardware speed between different machines diff?
- What if the link speed changes overtime?

Case study: NVLink + RDMA

Setup

- Each machine has multiple GPUs for computation
- These GPUs essentially form a distributed (processing) system

NVLink

- Fast interconnects that connect different GPUs

RDMA

- Fast networking feature that connects different machines

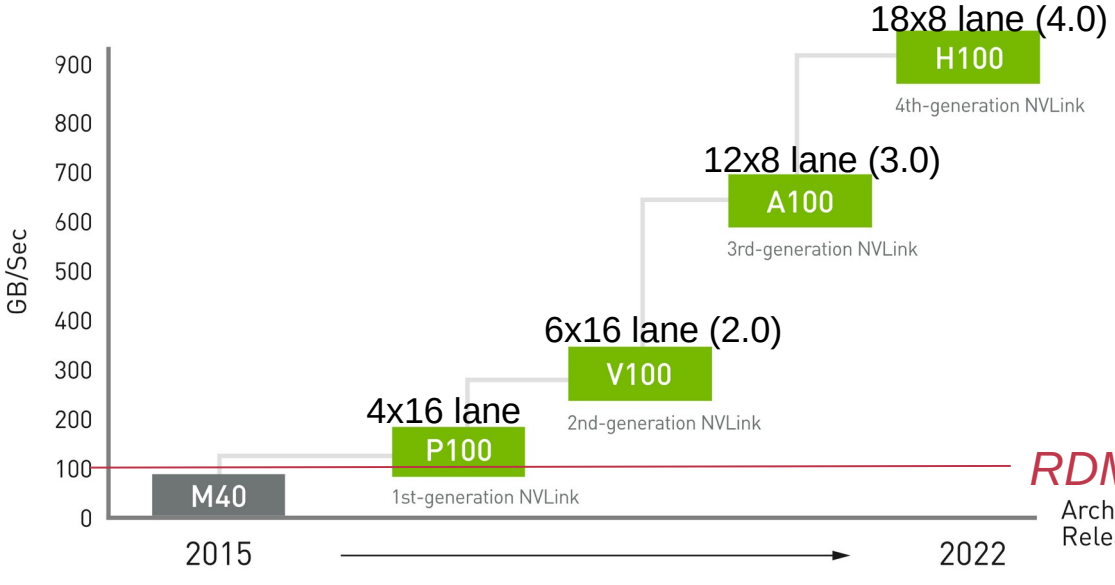
What does NVLink Look Like?

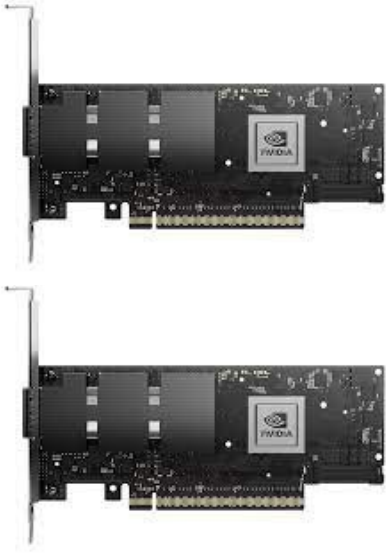
8-GPU on single board

Why is NVLink Fast?

Massive Parallelism (thanks to the close placement of hardware)

– GPUs are connected w/ *multiple lanes, w/ high frequencie*



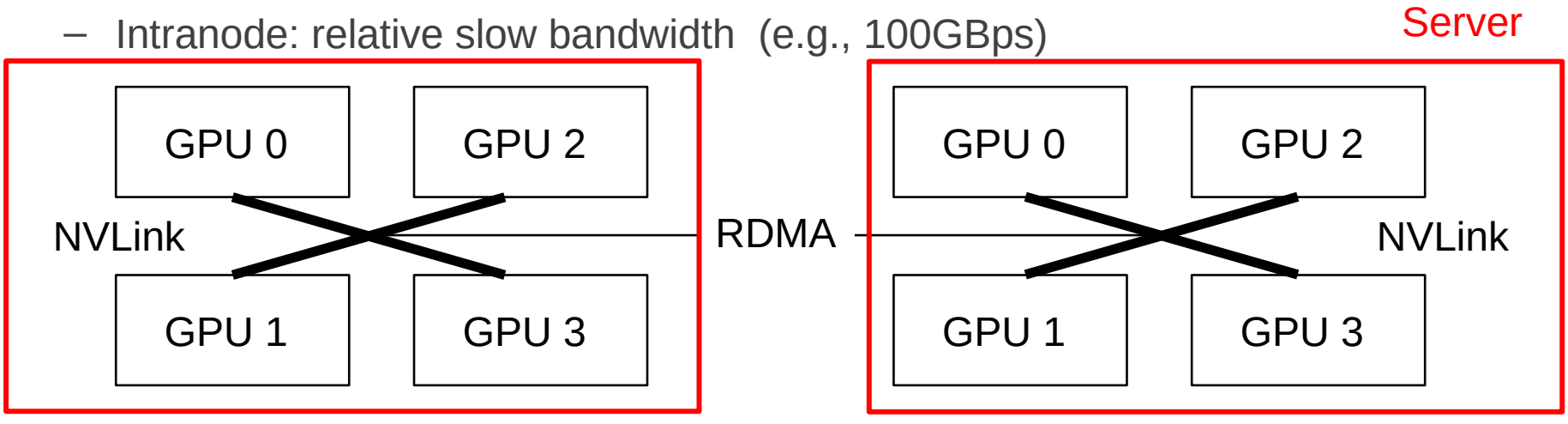


RDMA (2X400Gbps)

Communication heterogeneity

Different nodes' link capacity differs

- Internode: high bandwidth (e.g., 450GBps)
- Intranode: relative slow bandwidth (e.g., 100GBps)



Optimization opportunities

- We can do a parameter server within a server, and do rings across servers