15-884: Machine Learning Systems

Distributed Training and Communication Primitives

Instructor: Tianqi Chen

Carnegie Mellon University
School of Computer Science

catalyst

A Typical Deep Learning System Stack

Programming Abstraction

Automatic Differentiation

Graph IR Optimizations and Transformations

Runtime and Parallel Scheduling

Optimized Device Code, Libraries

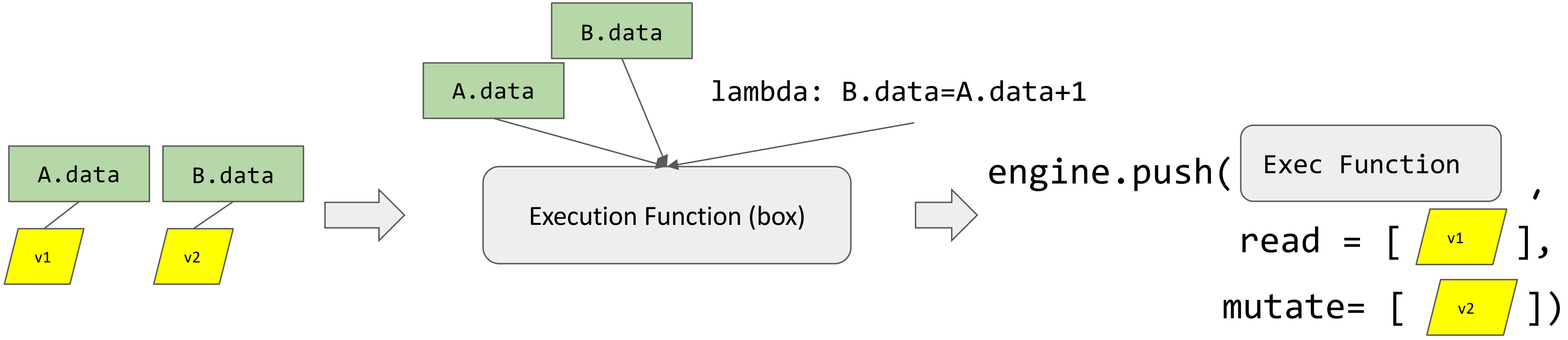
Accelerators and Hardware Backends

Recap: Parallel Scheduling Engine

The Tagged Data

Pack Reference to Related Things into Execution Function (via Closure)

Push the Operation to Engine

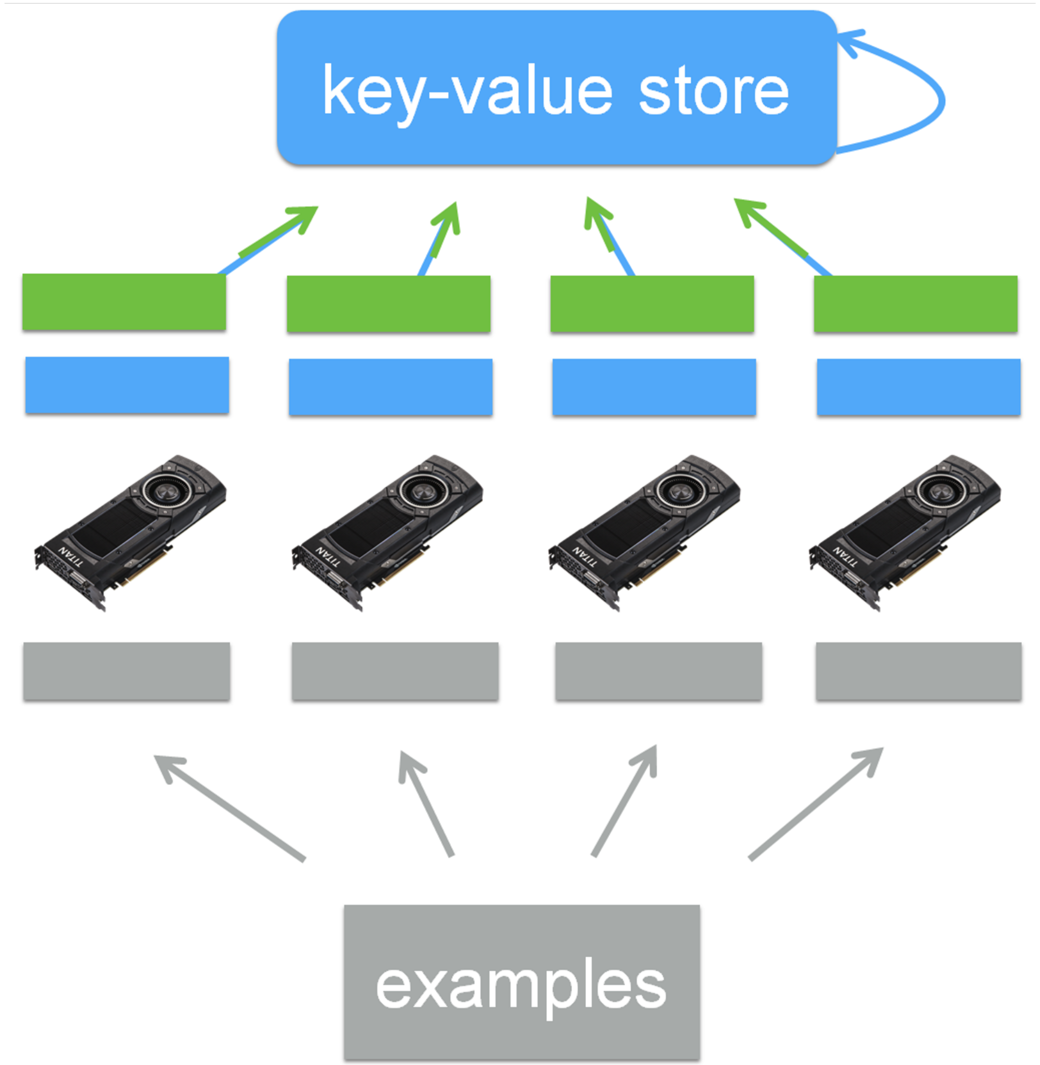


Recap: Example Scheduling

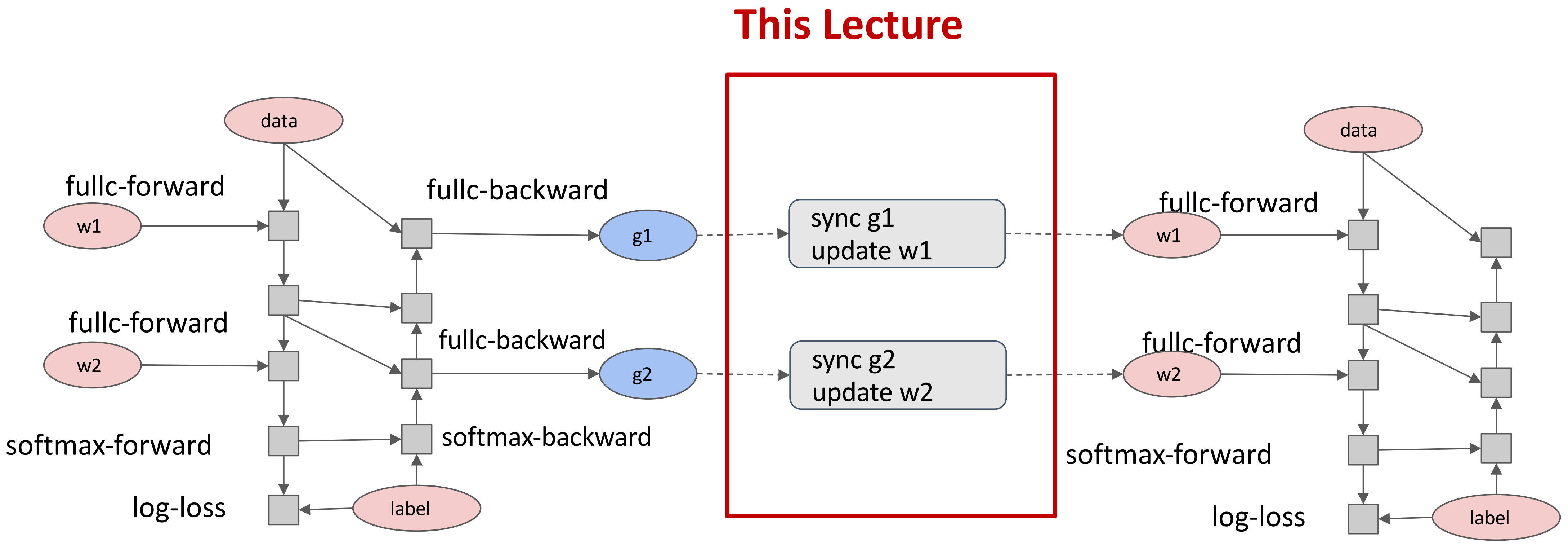
```
A = 2          engine.push(lambda: A.data=2,
                                 read=[], mutate= [A.var])

B = A + 1      engine.push(lambda: B.data=A.data+1,
                                read=[A.var], mutate= [B.var])

D = A * B      engine.push(lambda: D.data=A.data * B.data,
                                read=[A.var, B.var], mutate=[D.var])
```

Data Parallelism

- Train replicated version of model in each machine
- Synchronize the gradient

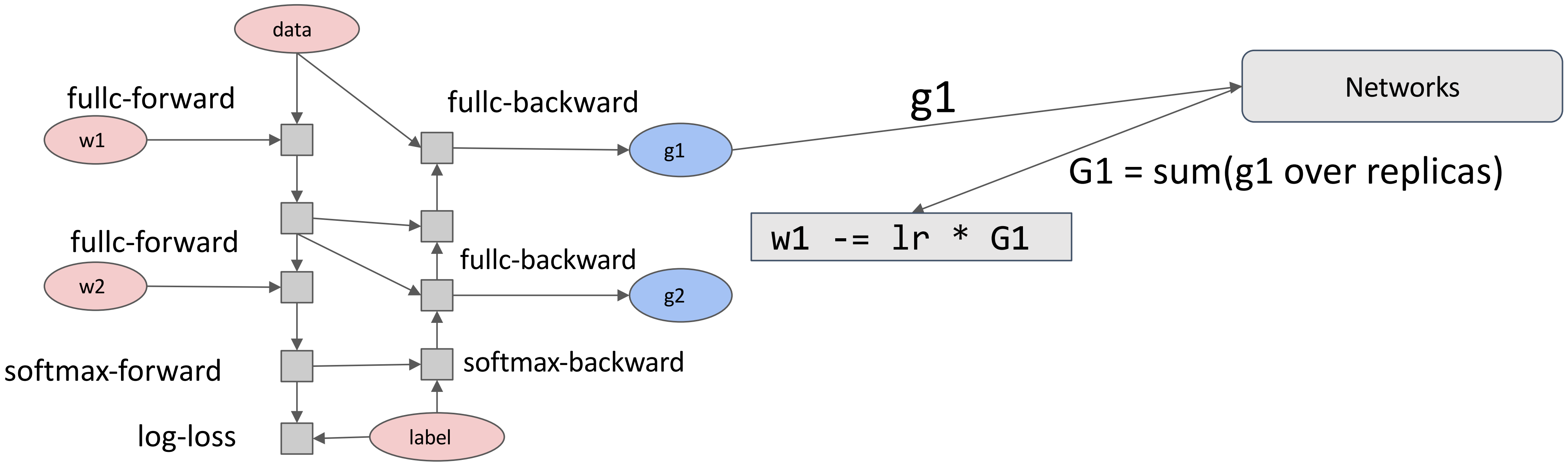


How to do Synchronization over Network



Distributed Gradient Aggregation, Local Update

Many replicas of the same graph run in parallel



Allreduce: Collective Reduction

Interface

```
result = allreduce(float buffer[size])
```

Running Example

Machine 1

```
comm = communicator.create()
a = [1, 2, 3]
b = comm.allreduce(a, op=sum)
```

Machine 2

```
comm = communicator.create()
a = [1, 0, 1]
b = comm.allreduce(a, op=sum)
```

Machine 1

```
assert b == [2, 2, 4]
```

Machine 2

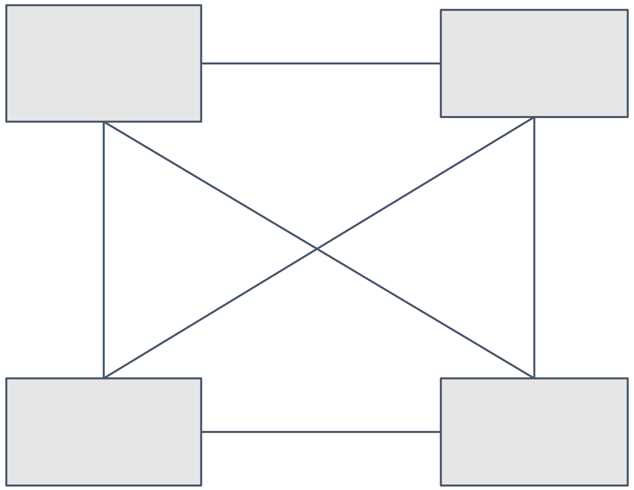
```
assert b == [2, 2, 4]
```

Use Allreduce for Data Parallel Training

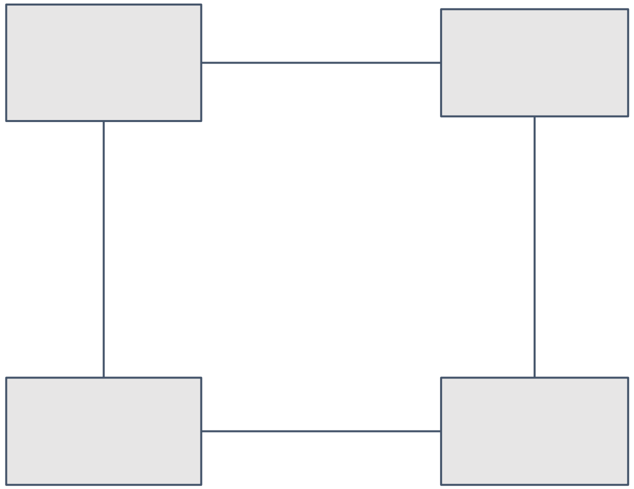
```
grad = gradient(net, w)

for epoch, data in enumerate(dataset):
  g = net.run(grad, in=data)
  gsum = comm.allreduce(g, op=sum)

  w -= lr * gsum / num_workers
```

Common Connection Topologies

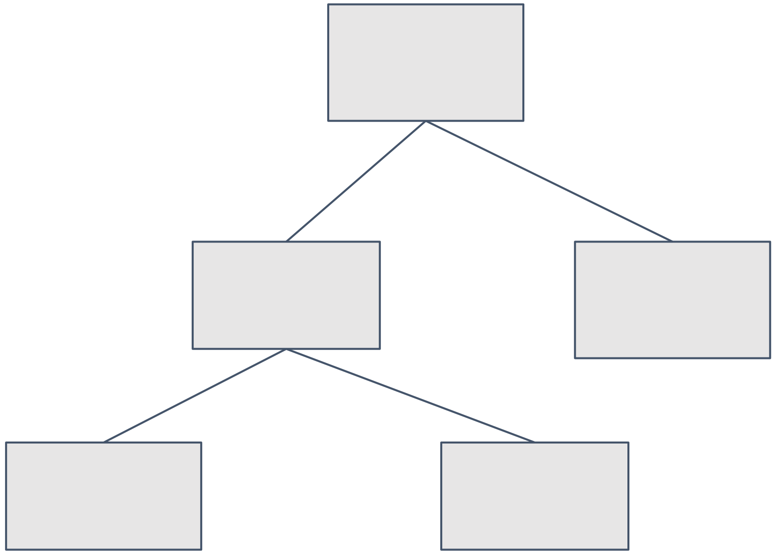
All-to-all:
(plugged to same switch)



Ring (NVLink)



Tree-Shape

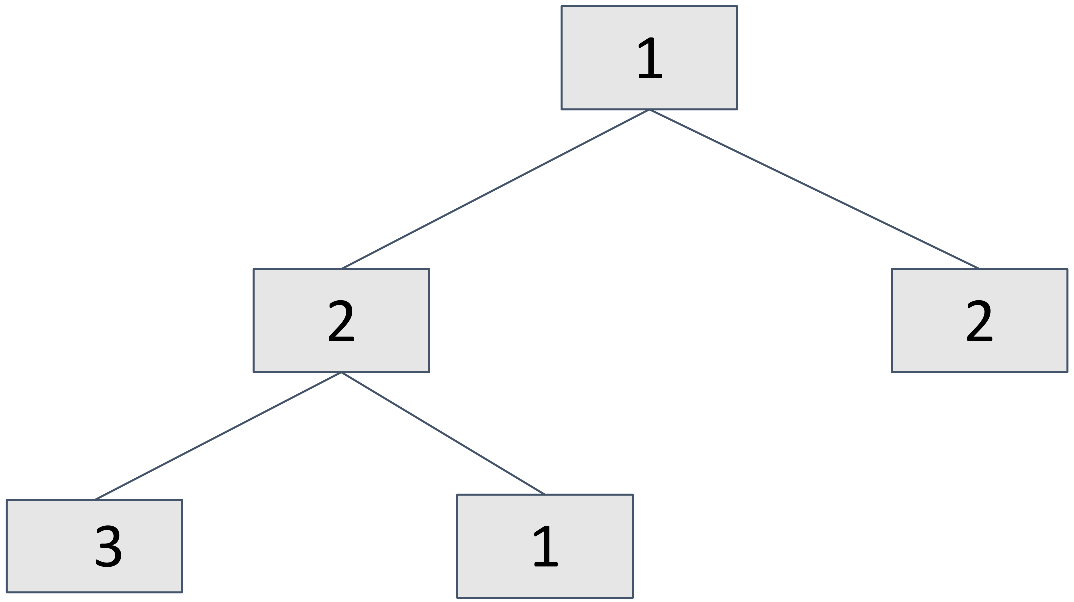


Discussion

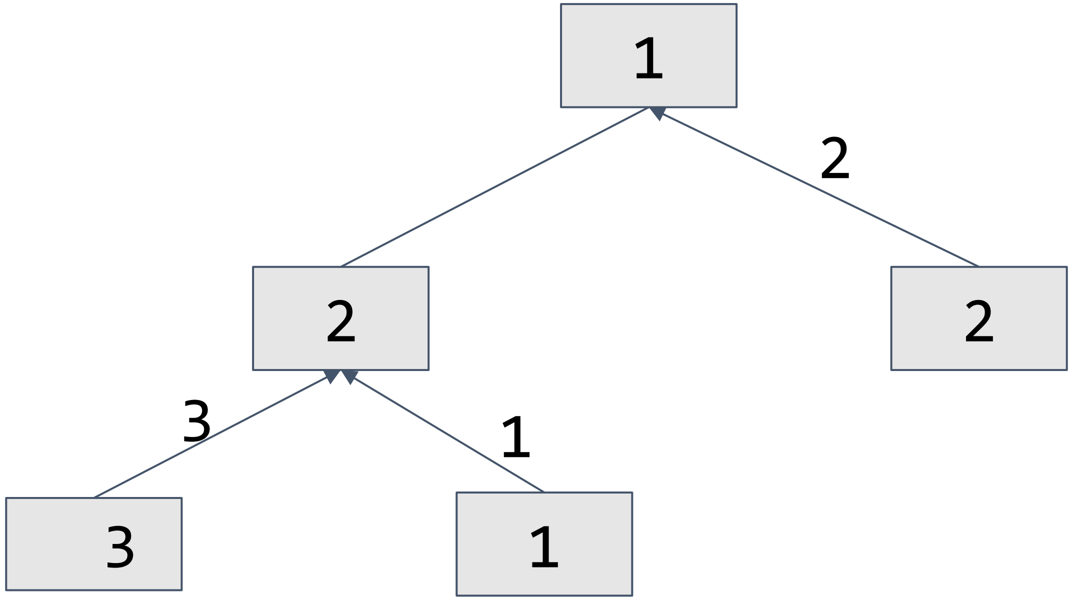
- How to Implement Allreduce over Network
- What is impact of network topology on this

Tree Shape Reduction

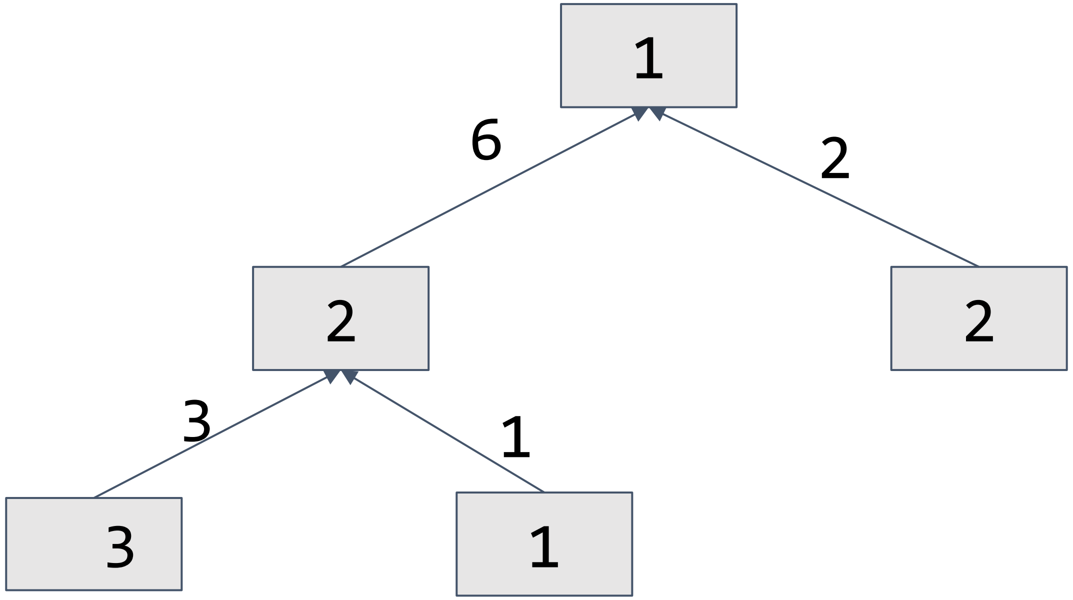
- Logically form a reduction tree between nodes
- Aggregate to root then broadcast



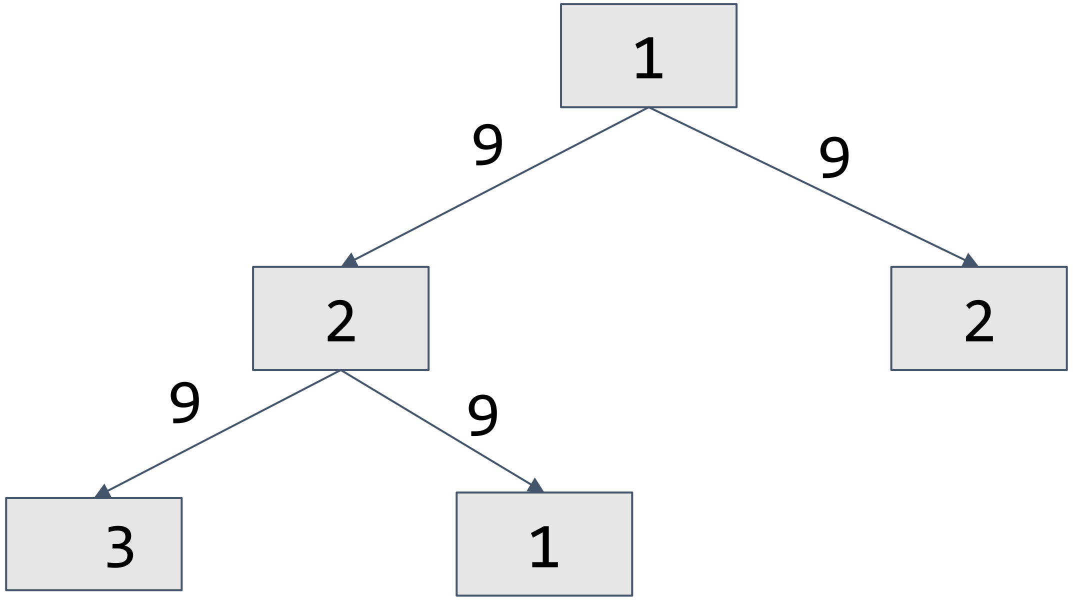
Tree Shape Reduction



Tree Shape Reduction



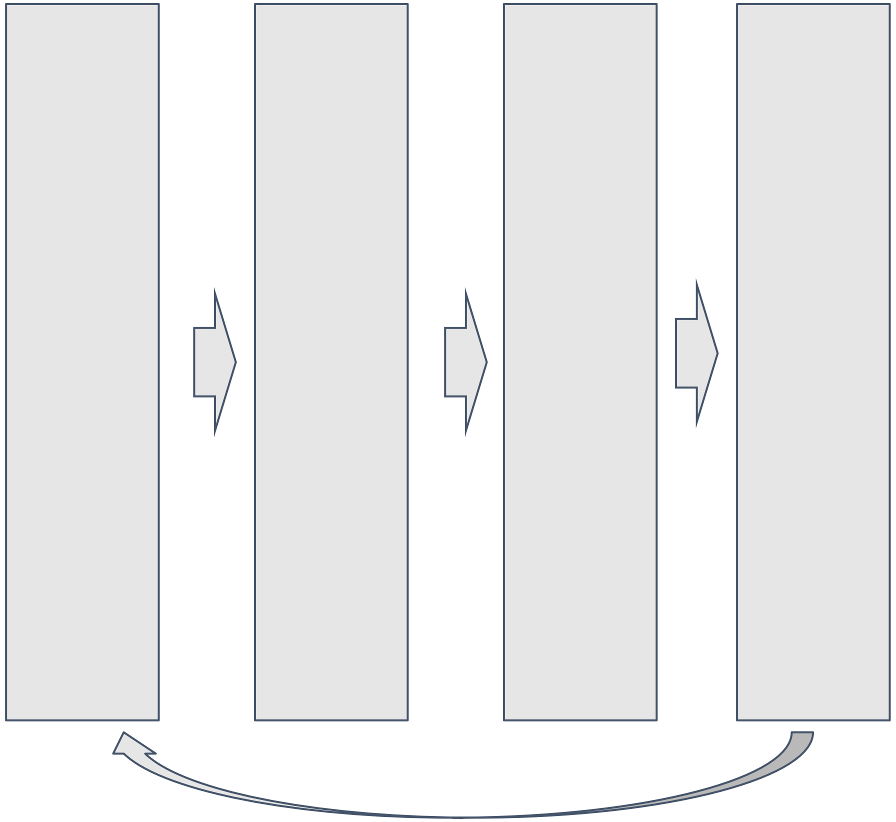
Tree Shape Reduction



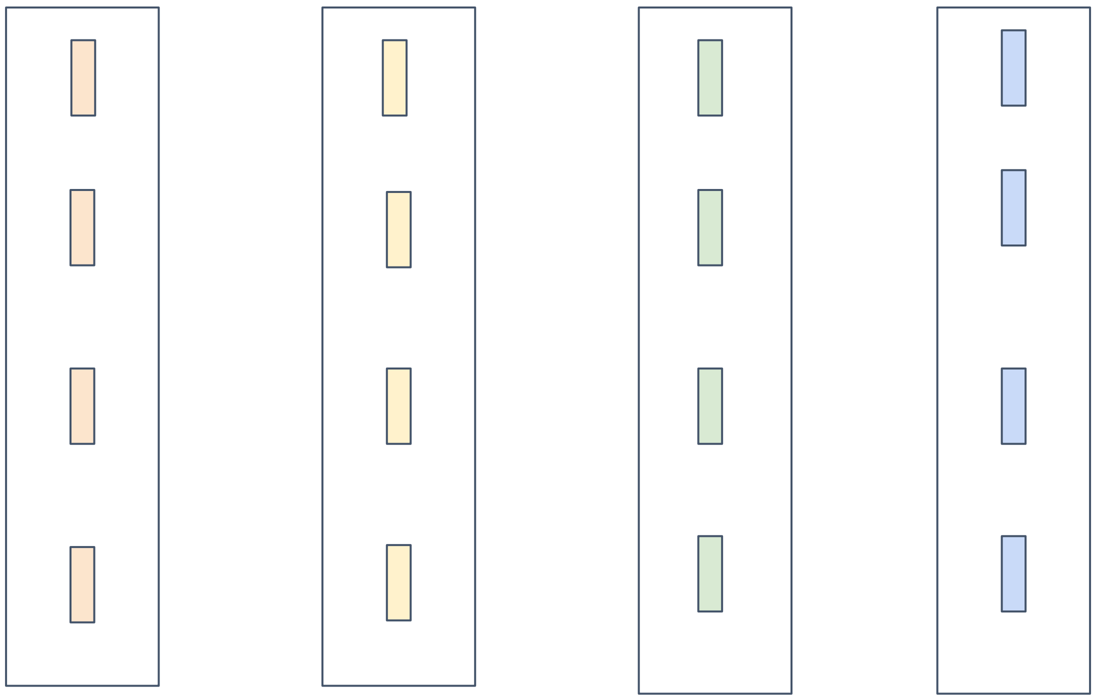
Question: What is Time Complexity of Tree Shape Reduction

Ring based Reduction

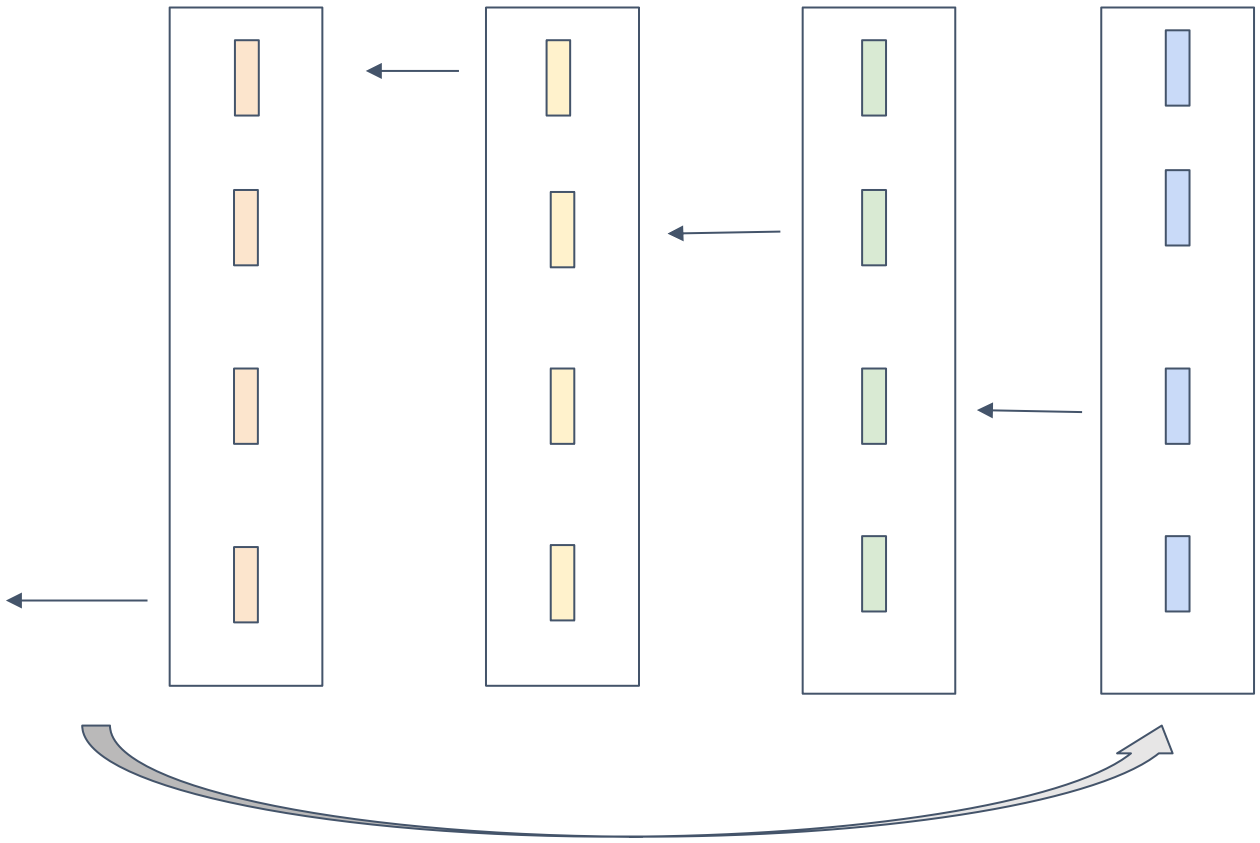
- Form a logical ring between nodes
- Streaming aggregation



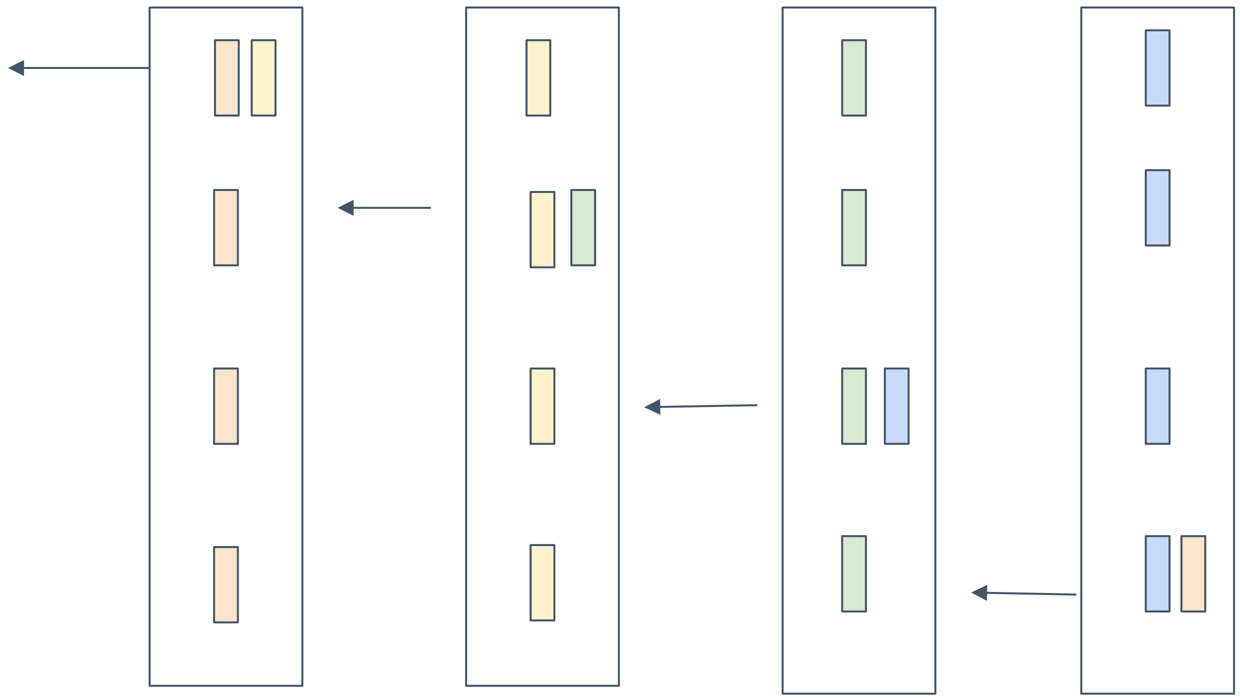
Ring based Reduction



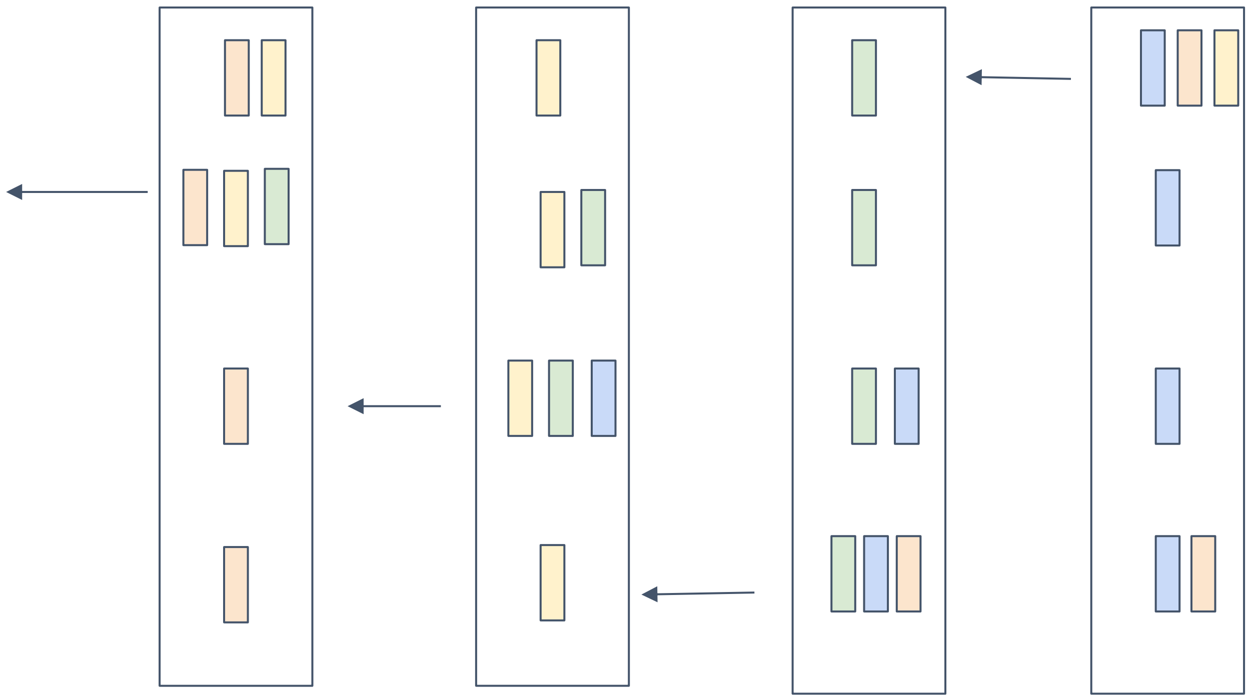
Ring based Reduction



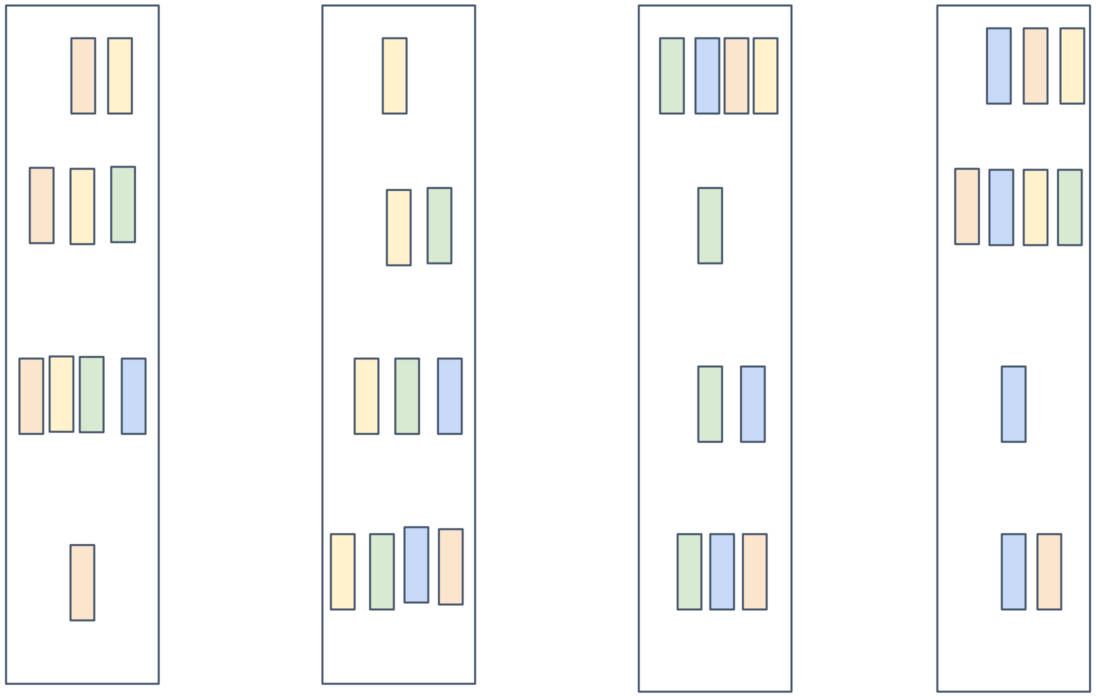
Ring based Reduction



Ring based Reduction



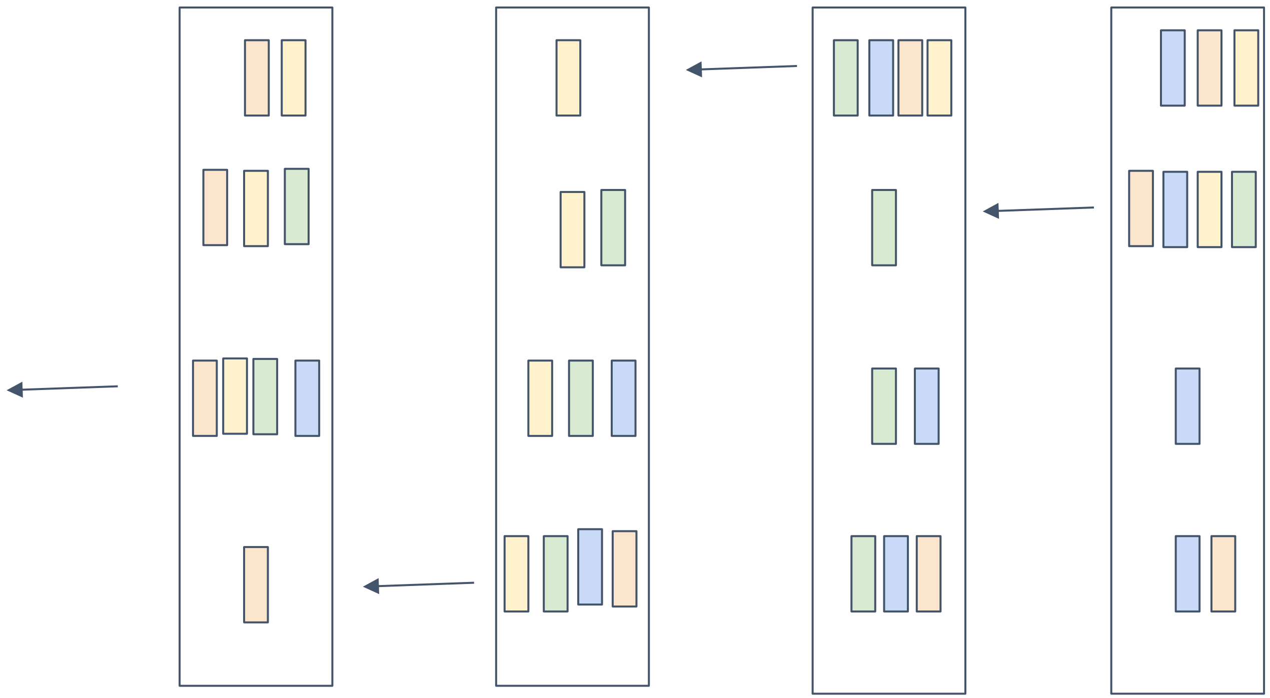
Ring based Reduction



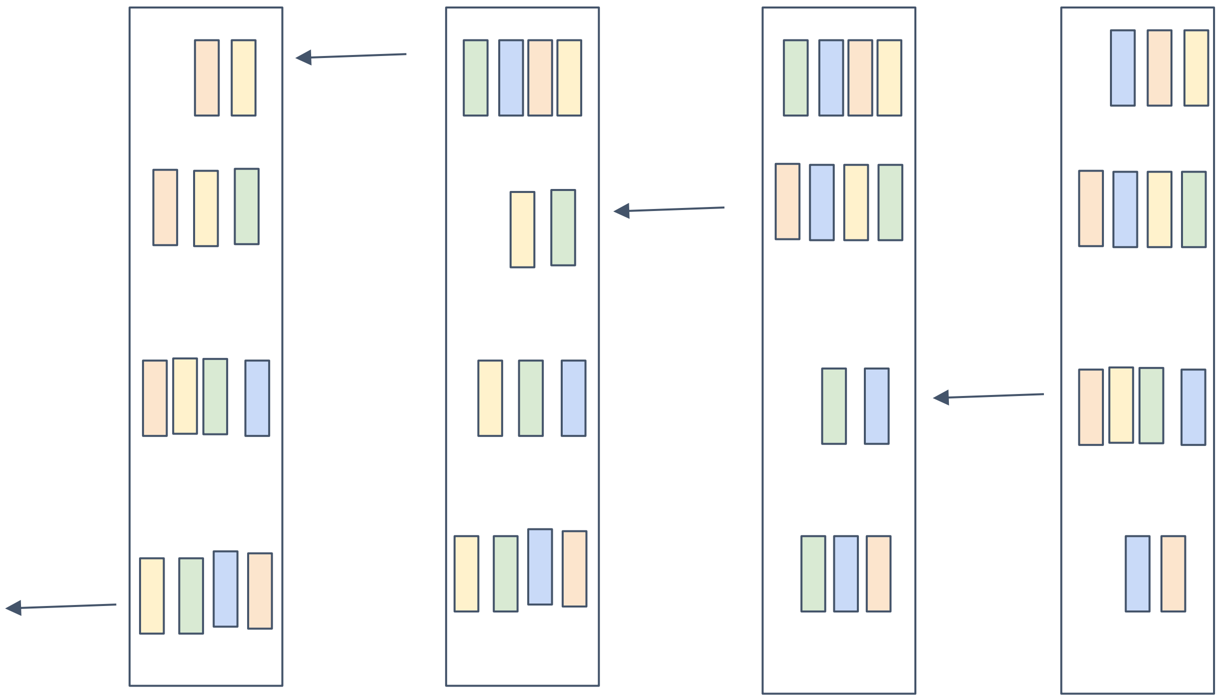
Each node have correctly reduced result of one segment!

This is called *reduce_scatter*

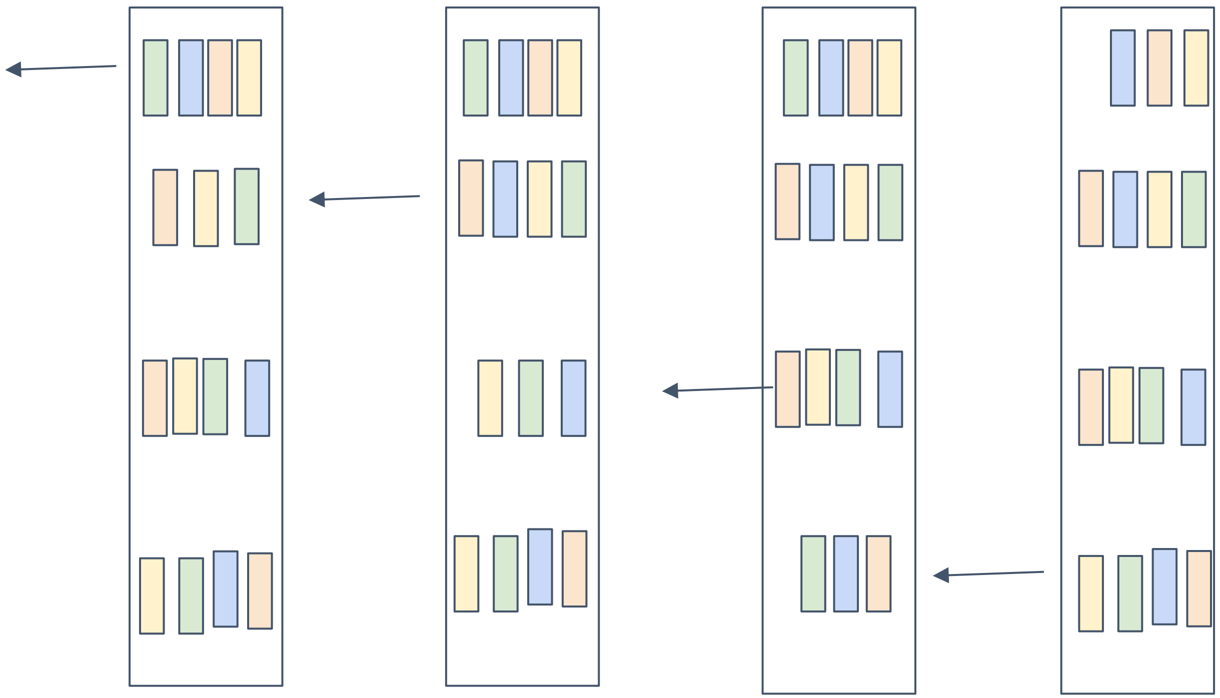
Ring based Reduction: Allgather phase



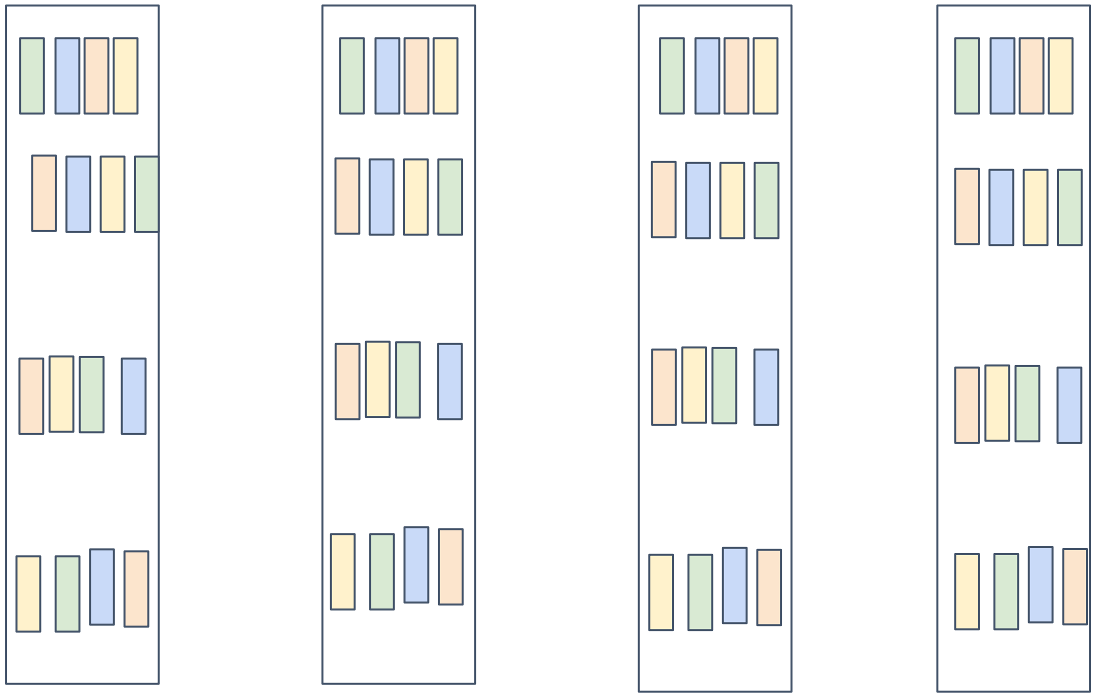
Ring based Reduction: Allgather phase



Ring based Reduction: Allgather phase



Ring based Reduction: Allgather phase

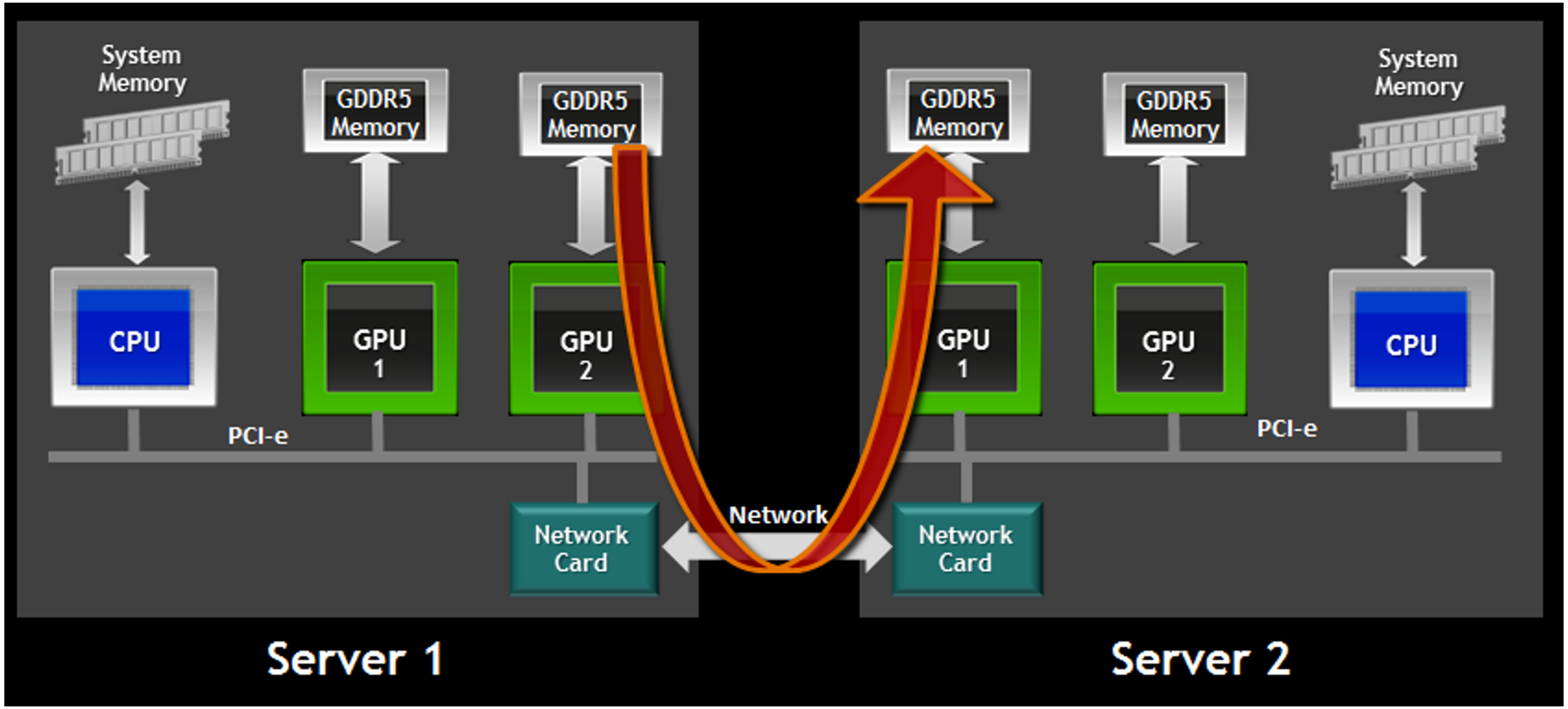


Question: What is Time Complexity of Ring based Reduction

Allreduce Libraries

- MPI offers efficient CPU allreduce
- dmlc/rabit: fault tolerant variant
- Horovod.ai
- NCCL: Nvidia’ efficient multiGPU collective

GPUDirect and RMDA



Source: nvidia

NCCL: Nvidia’s Efficient Multi-GPU Collective

- Uses unified GPU direct memory accessing
- Each GPU launch a working kernel, cooperate with each other to do ring-based reduction
- A single C++ kernel implements intra GPU synchronization and Reduction

Discussion

- What are pros and cons of Allreduce primitives
- How to integrate allreduce with a task scheduler

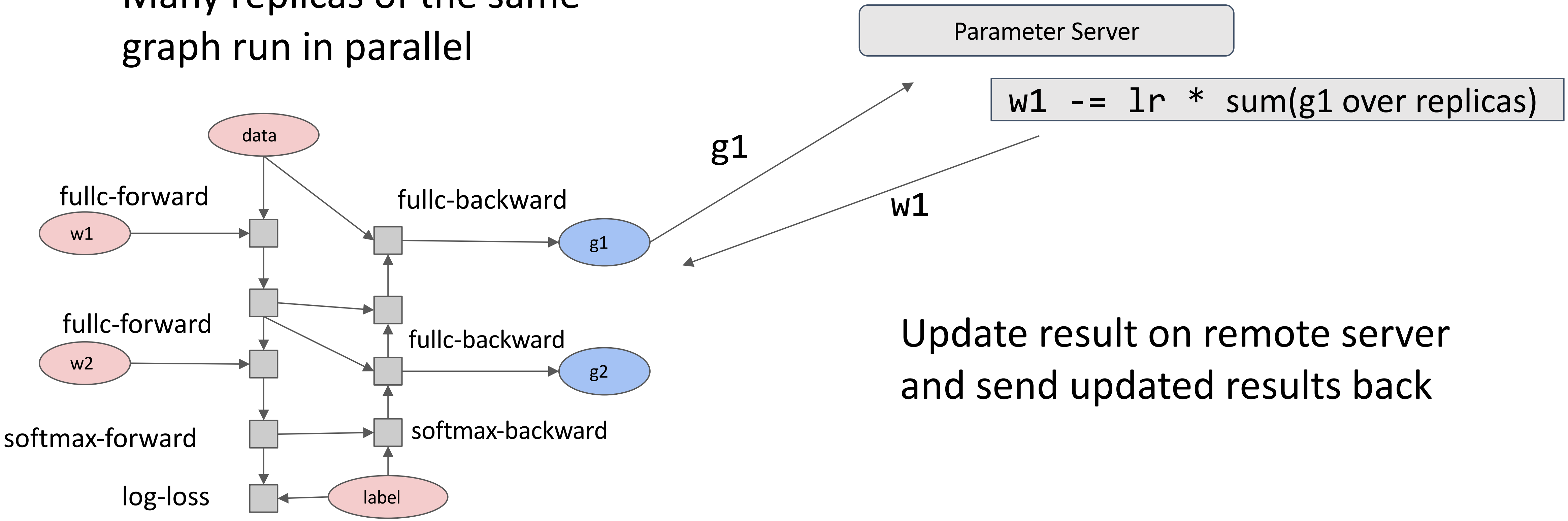
Schedule Allreduce Asynchronously

Make use of mutation semantics!

```
A = 2                    ⇨   engine.push(
                               lambda: A.data=2,
                               read=[], mutate= [A.var])

B = comm.allreduce(A)    ⇨   engine.push(
                                 lambda: B.data=allreduce(A.data),
                                 read=[A.var], mutate=[B.var, comm.var])

D = A * B                ⇨   engine.push(
                                 lambda: D.data=A.data * B.data,
                                 read=[A.var, B.var], mutate=[D.var])
```

Distributed Gradient Aggregation, Remote Update

Many replicas of the same graph run in parallel

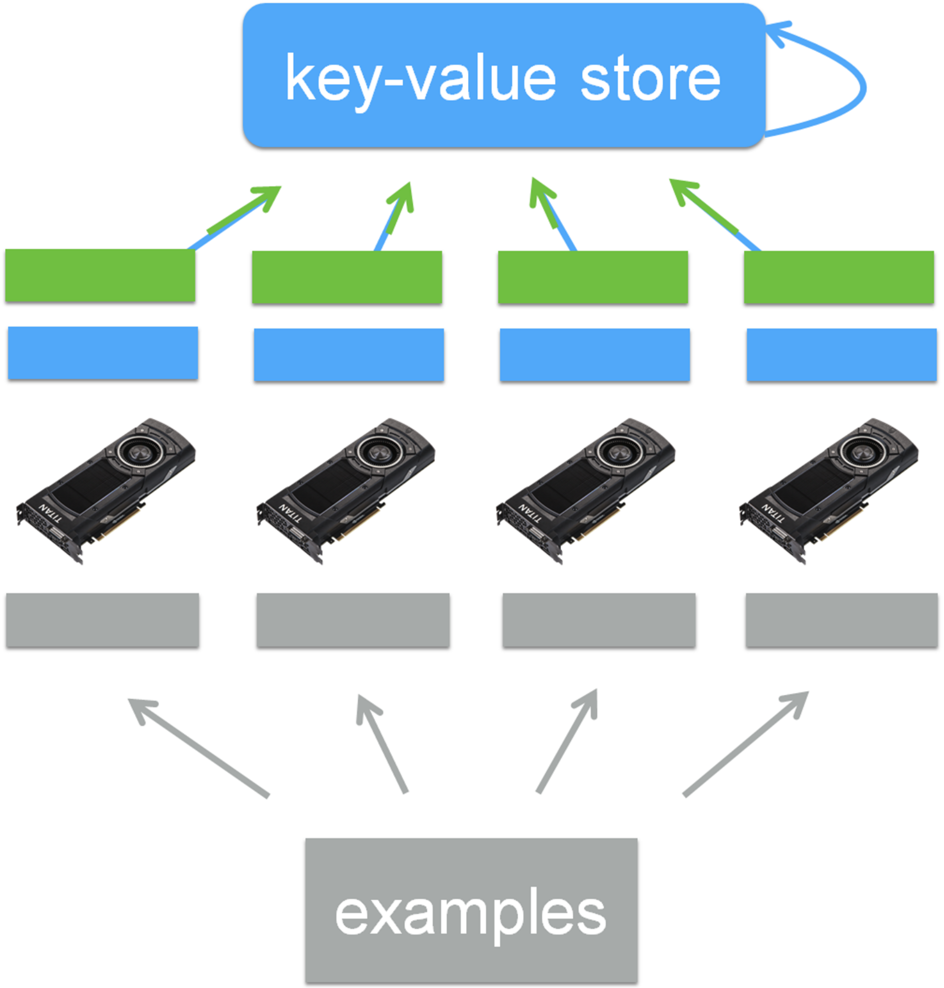


Update result on remote server and send updated results back

Parameter Server Abstraction

Interface

```
ps.push(index, gradient)

ps.pull(index)
```

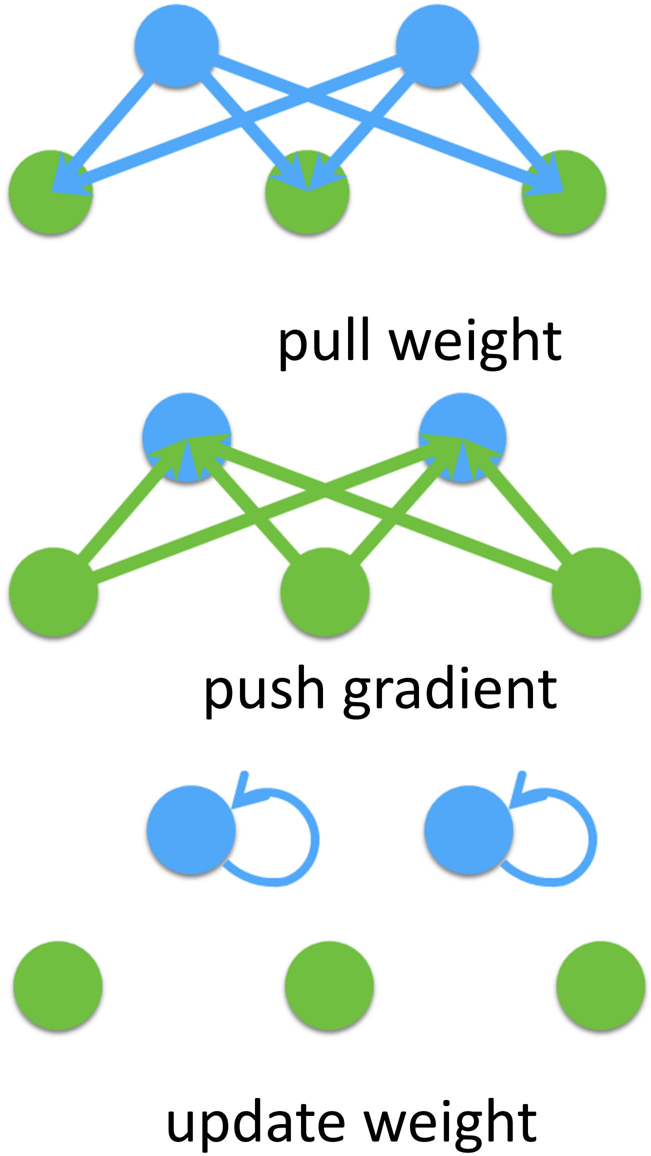


PS Interface for Data Parallel Training

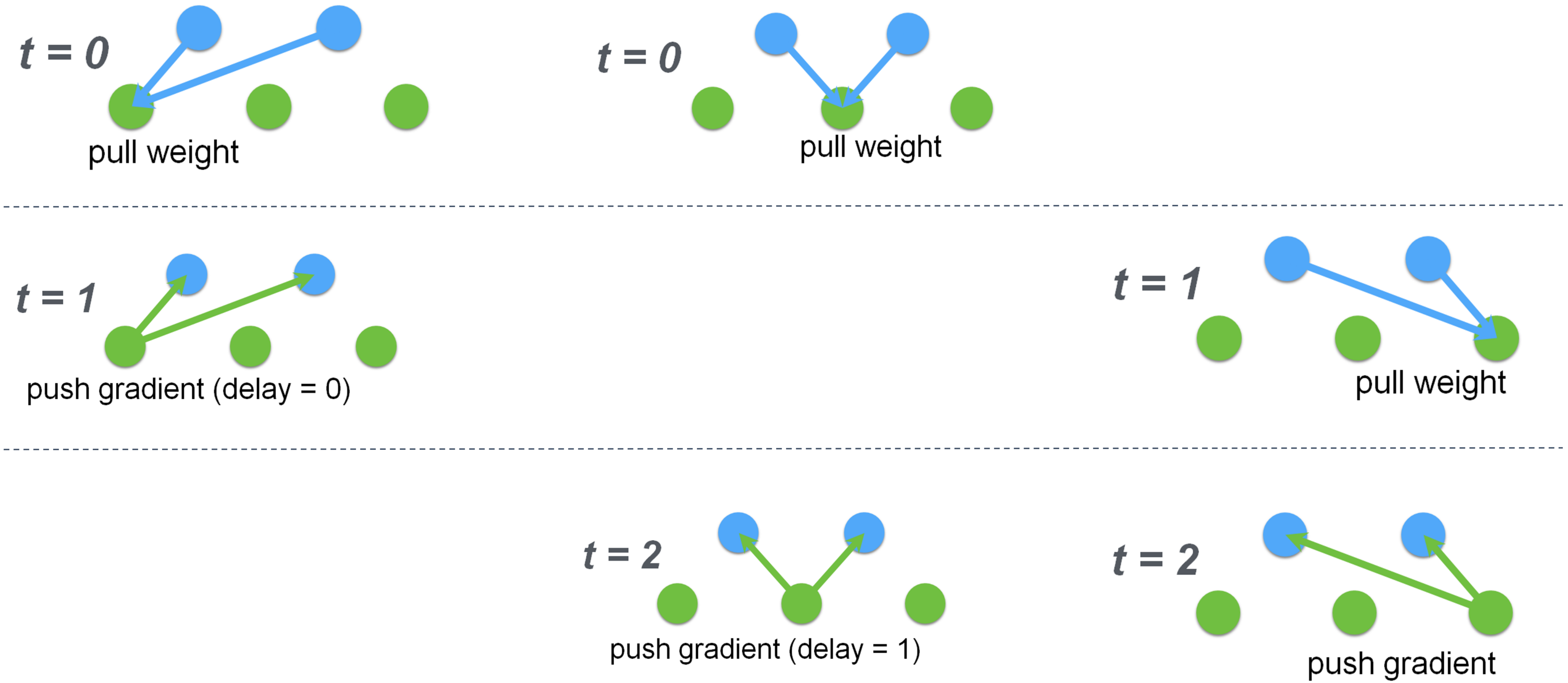
```
grad = gradient(net, w)

for epoch, data in enumerate(dataset):
  g = net.run(grad, in=data)

  ps.push(weight_index, g)
  w = ps.pull(weight_index)
```

PS Data Consistency: BSP

- "Synchronized"
    - Gradient aggregated over all works
    - All workers receives the same parameters
    - Give same result as single batch update
    - Brings challenges to synchronization

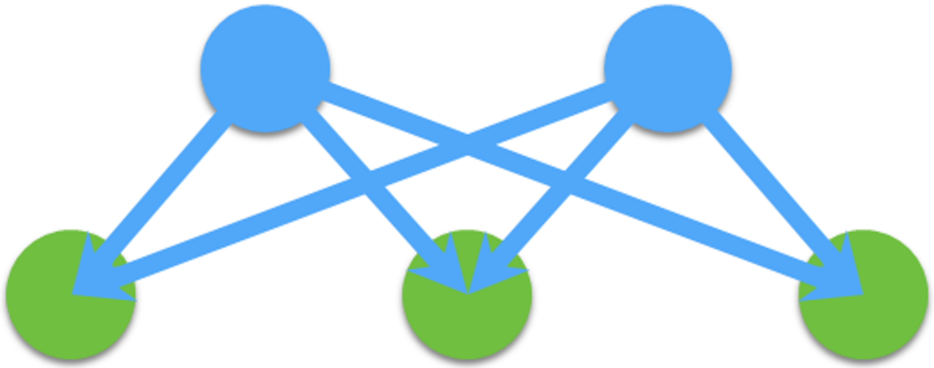


PS Consistency: Asynchronous



The Cost of PS Model: All to All Pattern

- Each worker talks to all servers
- Shard the parameters over different servers
- What is the time complexity of communication?

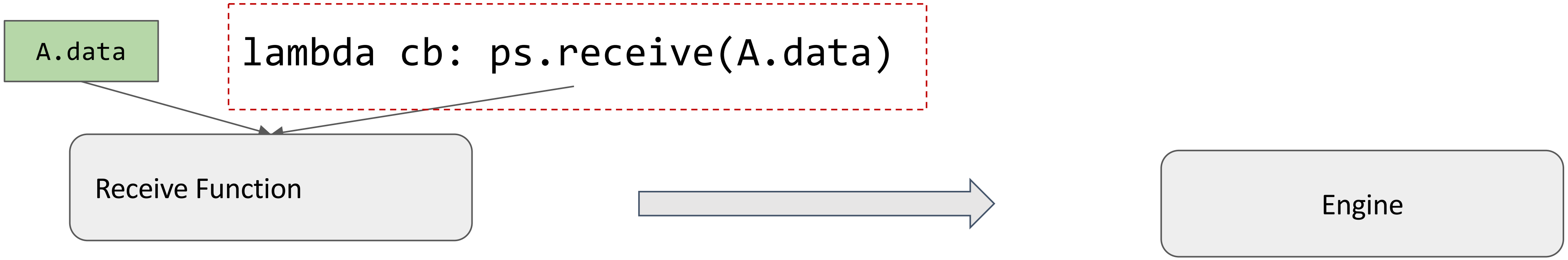


Discussion

- What are pros and cons of parameter server
- How can we handle fault tolerance/straggler in both allreduce or PS

Integrate Schedule with Networking using Events

Asynchronous function that takes a callback from engine

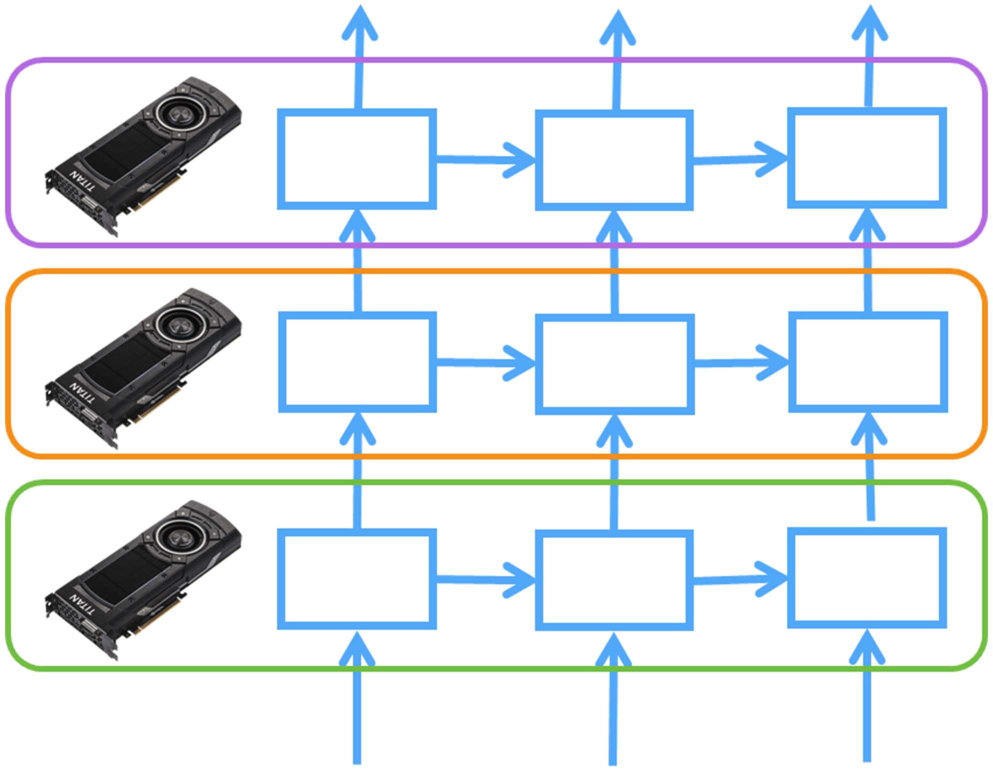



```
def event.on_data_received():
   # notify engine receive
complete
   cb();
```

Use the callback to notify engine that data receive is finished

Model Parallel Training

- Map parts of workload to different devices
- Require special dependency patterns (wave style)
  - e.g. LSTM

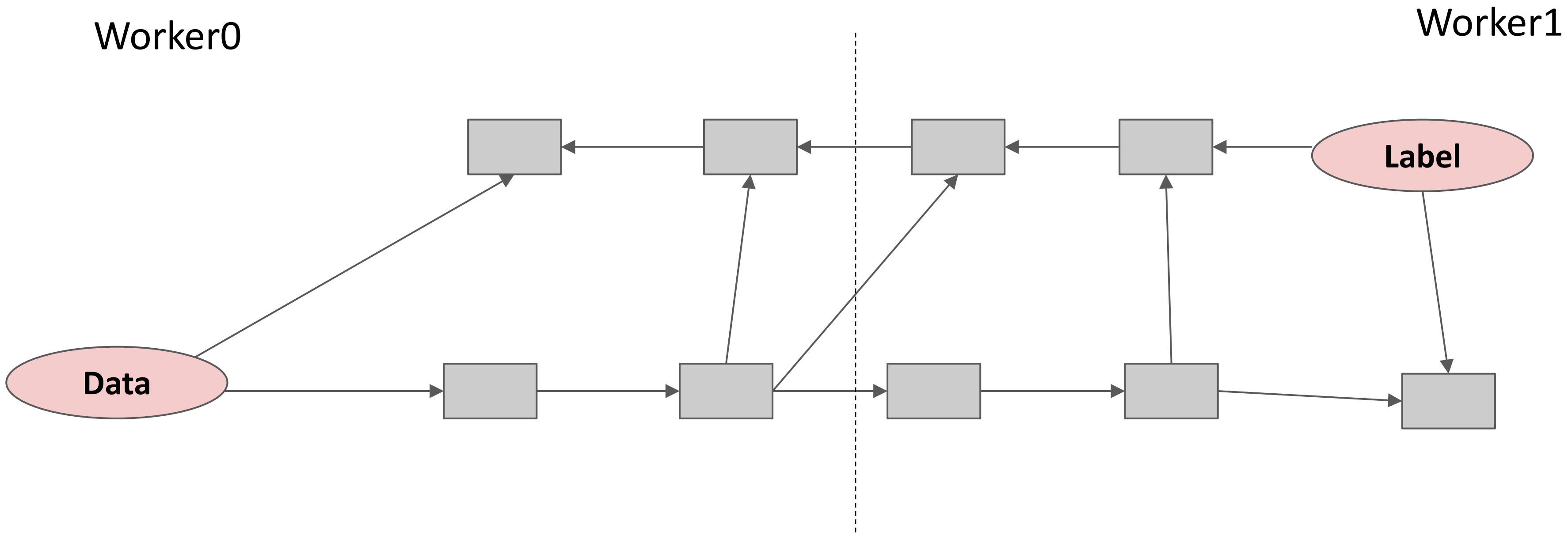


Question: How to Write Model Parallel Program?

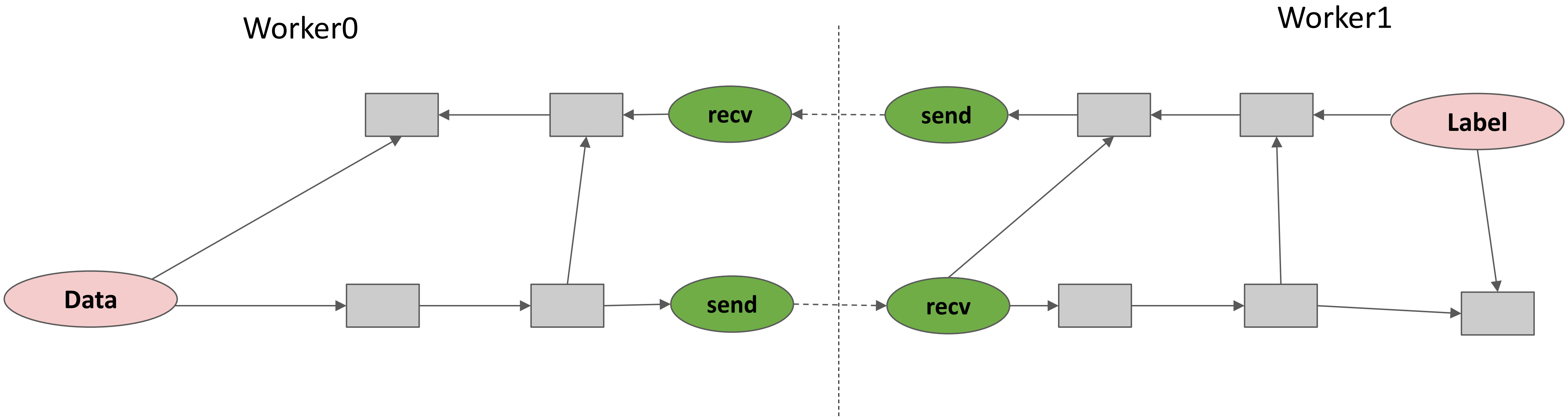
```
for i in range(num_layers):
  for t in range(num_time_stamp):
    out, state = layer[i].forward(data[i][t], state)
    data[i+1][t] = out.copyto(device[i])
```

Scheduler tracks these dependencies
we only talked about single host case

Breaking up the Computation for Model Parallelism



Breaking up the Computation for Model Parallelism



Partition the graph, put send/recv pairs in the boundary

Discussion

- How to represent pipeline model parallelism
- How can we handle fault tolerance/straggler issues

Summary: What's Special about Communication

Requirements

- Track dependency correctly
- Resolve resource contention and allocation
- Some special requirement on channel
    - Allreduce: ordered call

Most of them are simplified by a scheduler