# Lecture 12: Distributed Training and Communication Protocols

CSE599G1: Spring 2017

# Where are we

High level Packages

**User API**

- Programming API
- Gradient Calculation (Differentiation API)

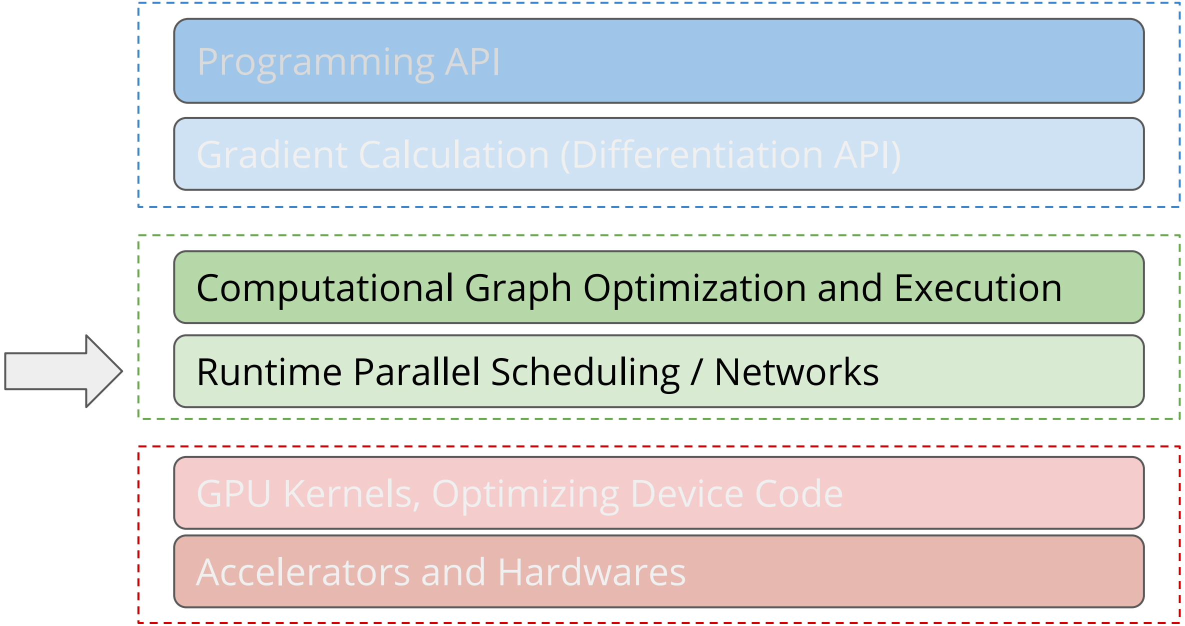
**System Components**

- Computational Graph Optimization and Execution
- Runtime Parallel Scheduling

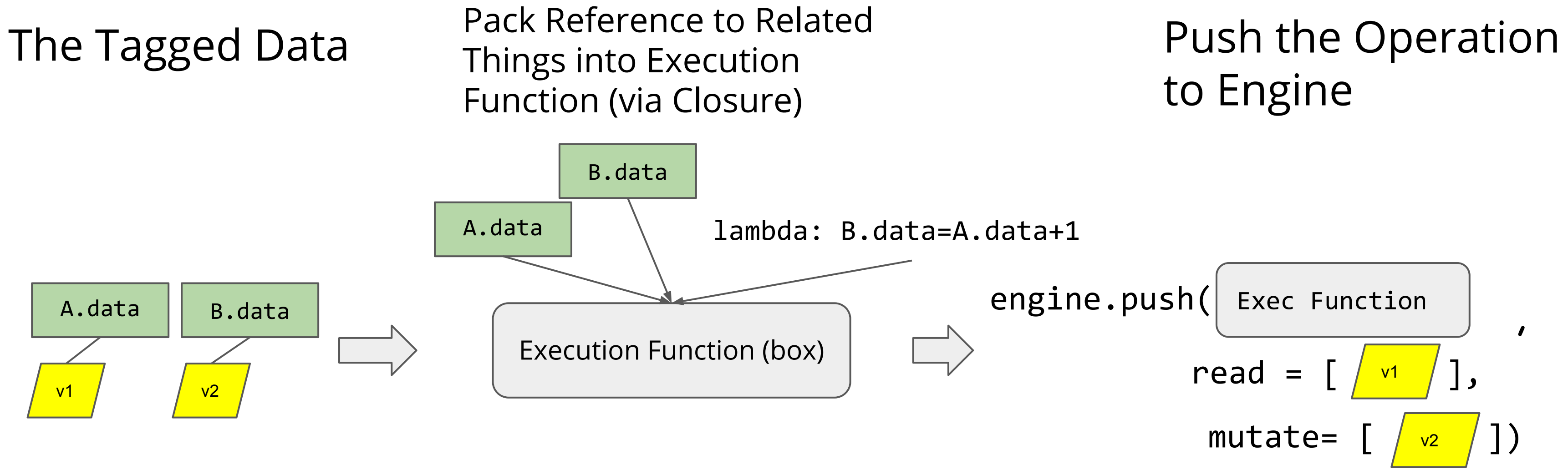
**Architecture**

- GPU Kernels, Optimizing Device Code
- Accelerators and Hardwares

# Where are we


Programming API
Gradient Calculation (Differentiation API)
Computational Graph Optimization and Execution
Runtime Parallel Scheduling / Networks
GPU Kernels, Optimizing Device Code
Accelerators and Hardwares

# Recap: Parallel Scheduling Engine



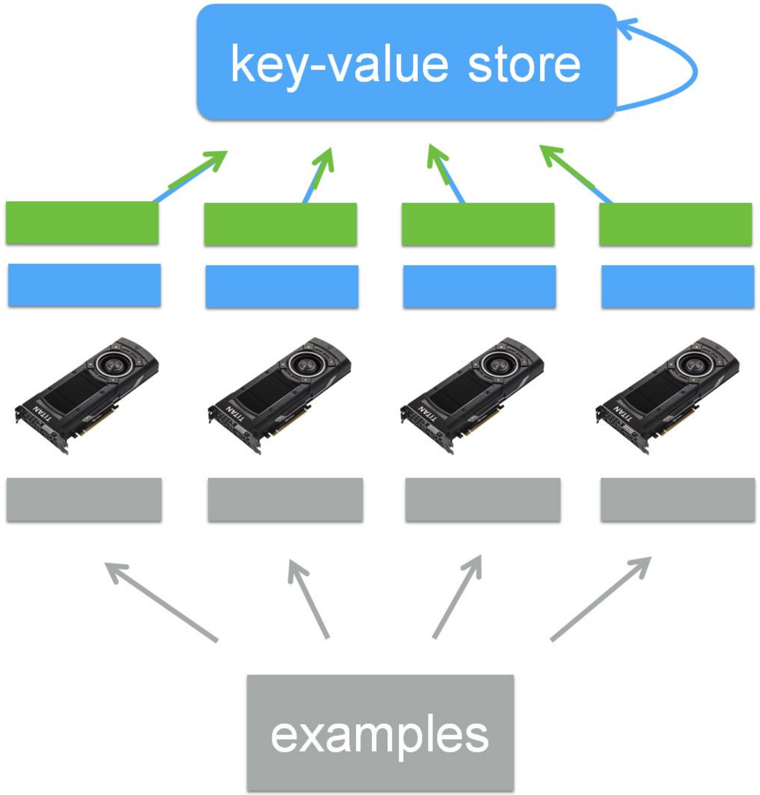
# Recap: Example Scheduling

```
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])
```

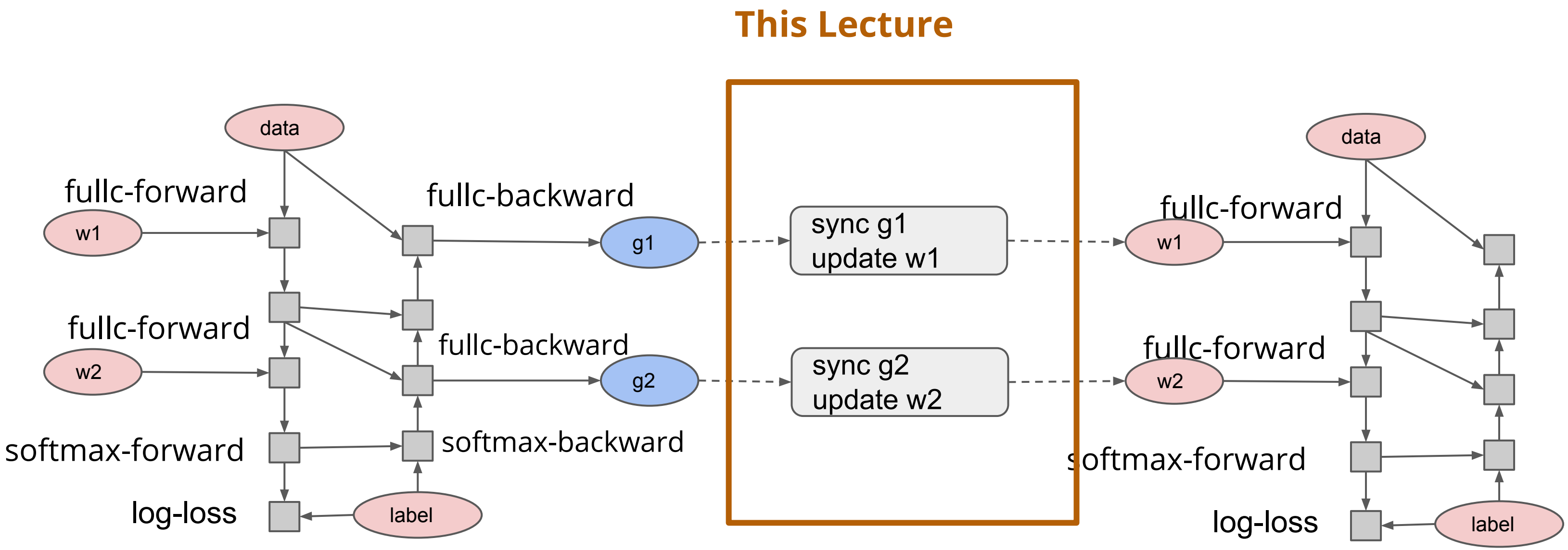
# Data Parallelism

- Train replicated version of model in each machine

- Synchronize the gradient

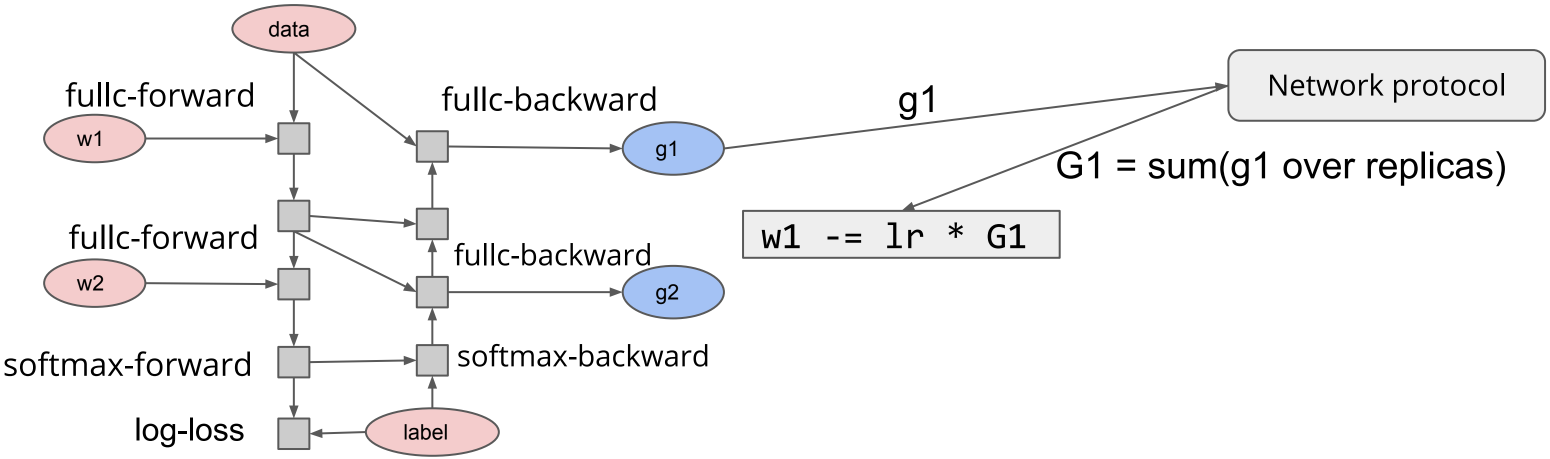


# How to do Synchronization over Network



# 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)
-----------------------------
assert b == [2, 2, 4]
```

Machine 2

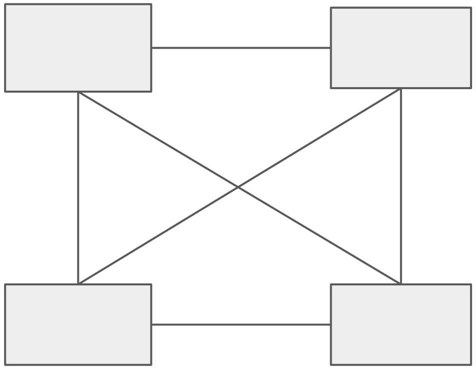
```
comm = communicator.create()
a = [1, 0, 1]
b = comm.allreduce(a, op=sum)
-----------------------------
assert b == [2, 2, 4]
```

# Use Allreduce for Data Parallel Training

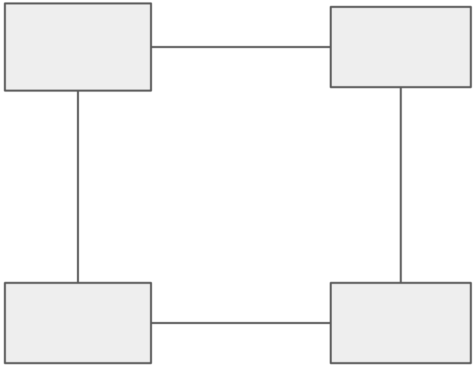
```
grad = gradient(net, w)

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

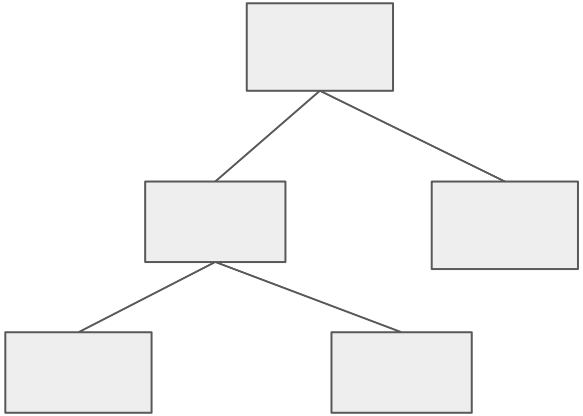
All-to-all:
(plugged to same switch)



Ring (NVLink)



Tree-Shape

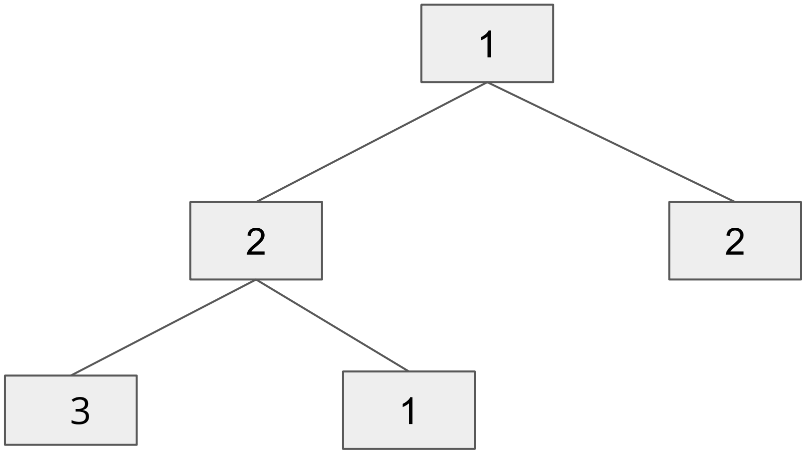


# Discussion: 3min

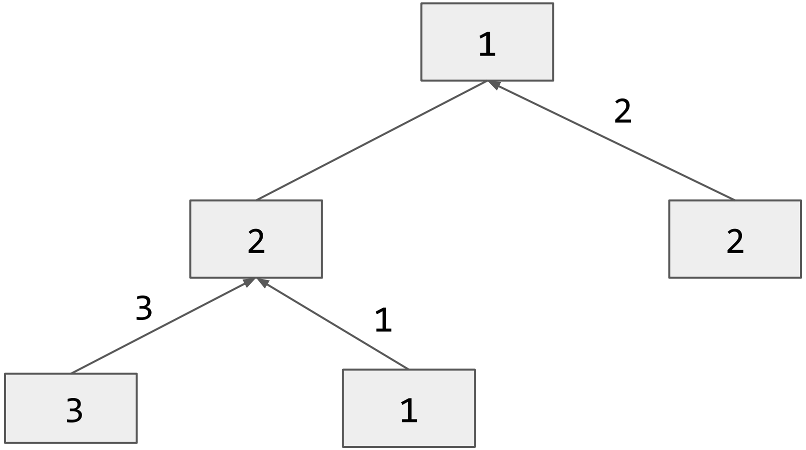
- How to Implement Allreduce over Network
- What is impact of network topology on this

# Tree Shape Reduction

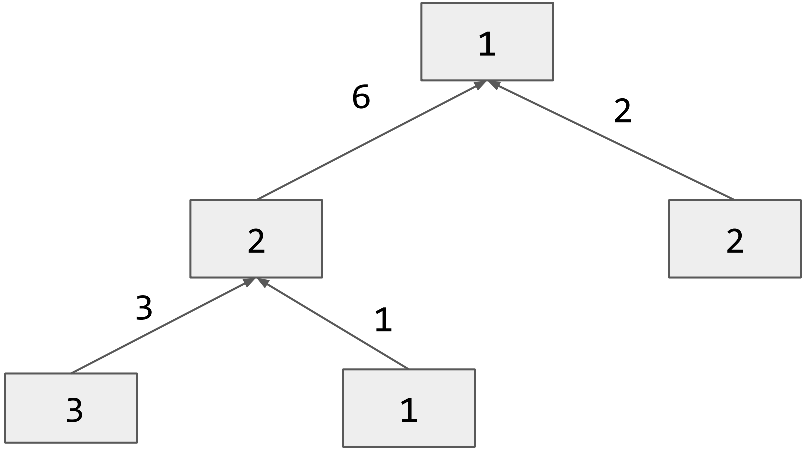
- Logically form a reduction tree between nodes
- Aggregate to root then broadcast



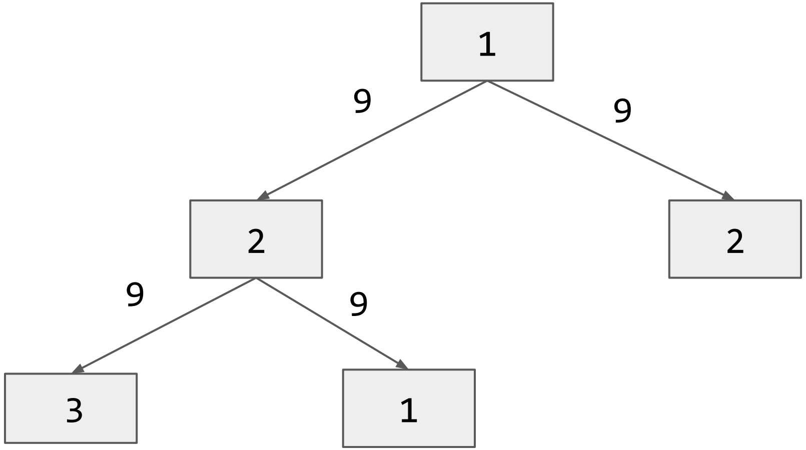
# Tree Shape Reduction



# Tree Shape Reduction



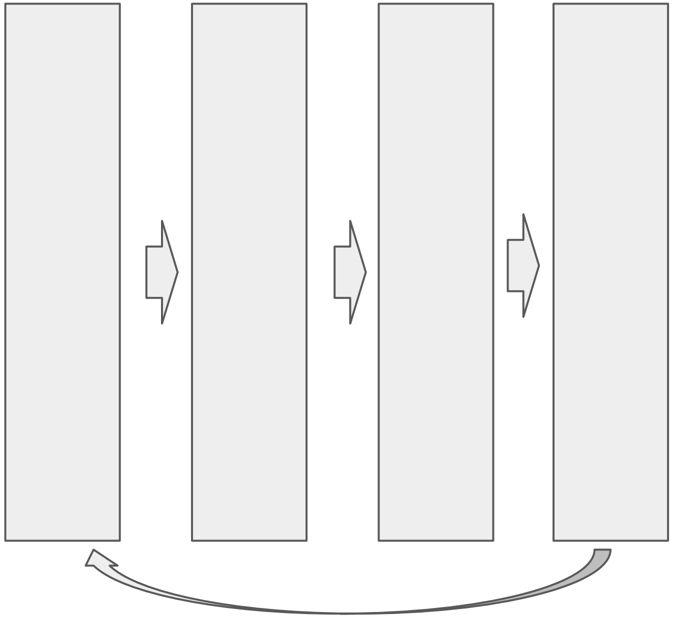
# Tree Shape Reduction



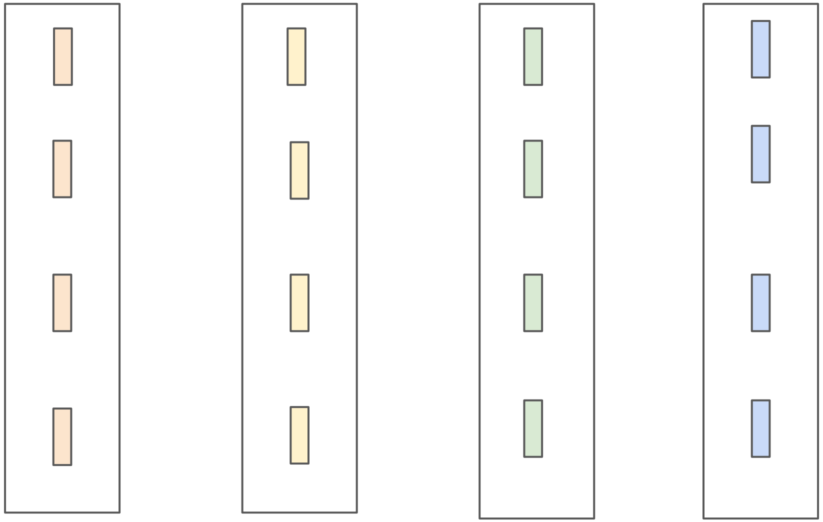

**Question: What is Time Complexity of Tree Shape Reduction**

# Ring based Reduction

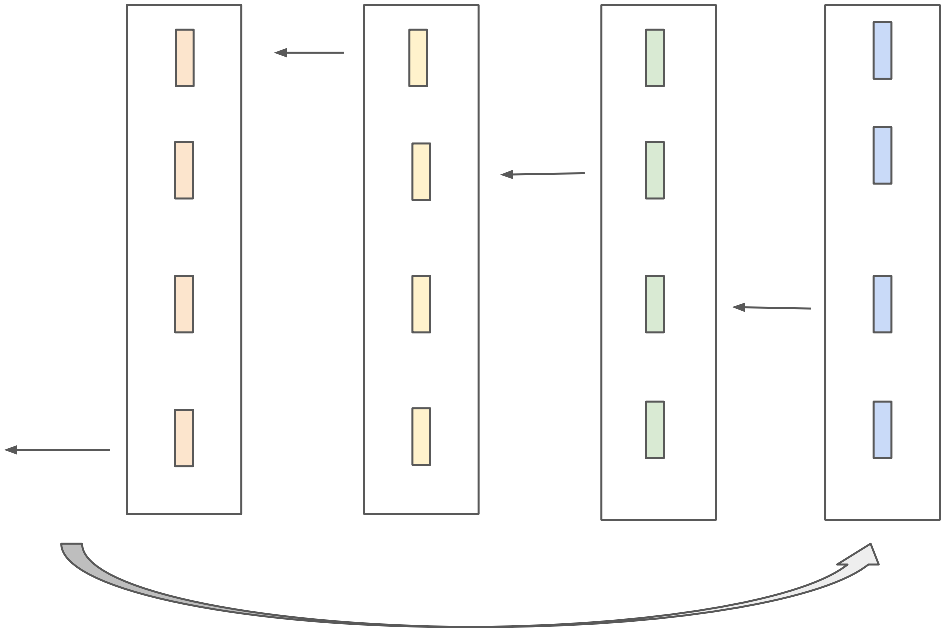
- Form a logical ring between nodes
- Streaming aggregation



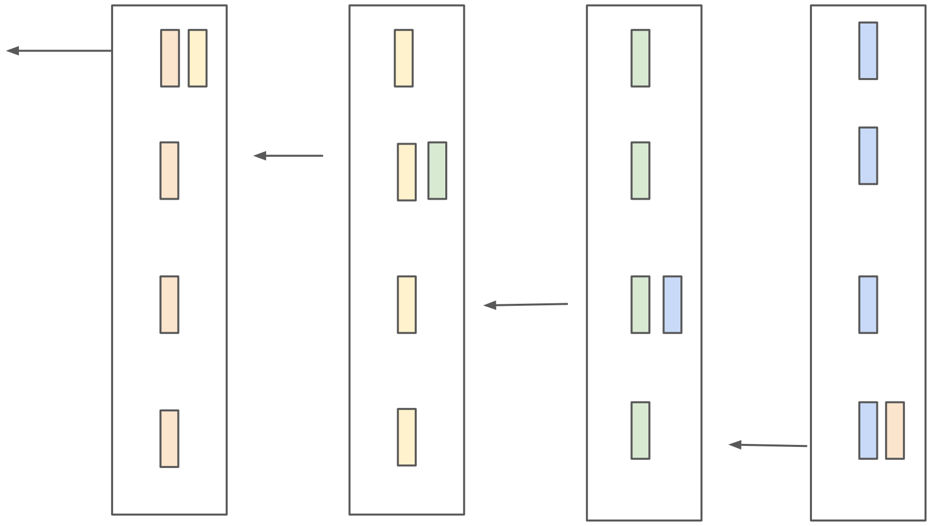
# Ring based Reduction



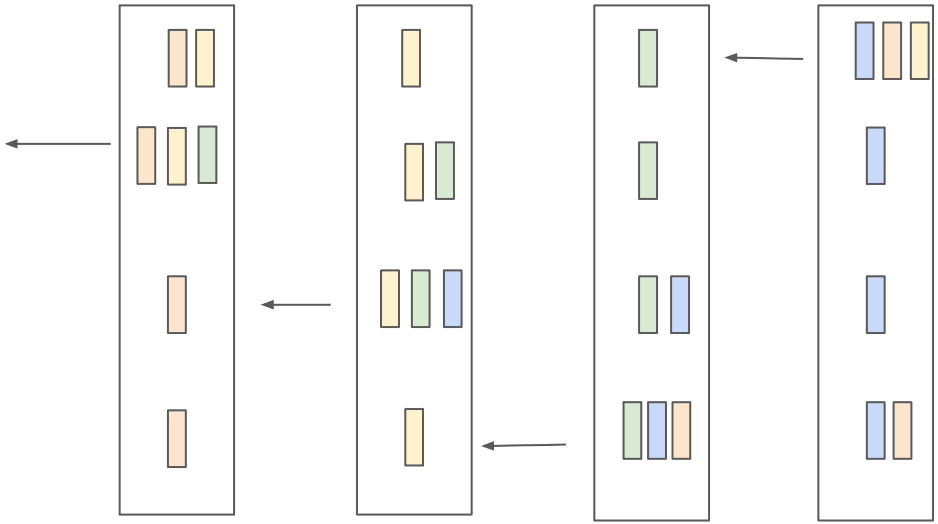
# Ring based Reduction



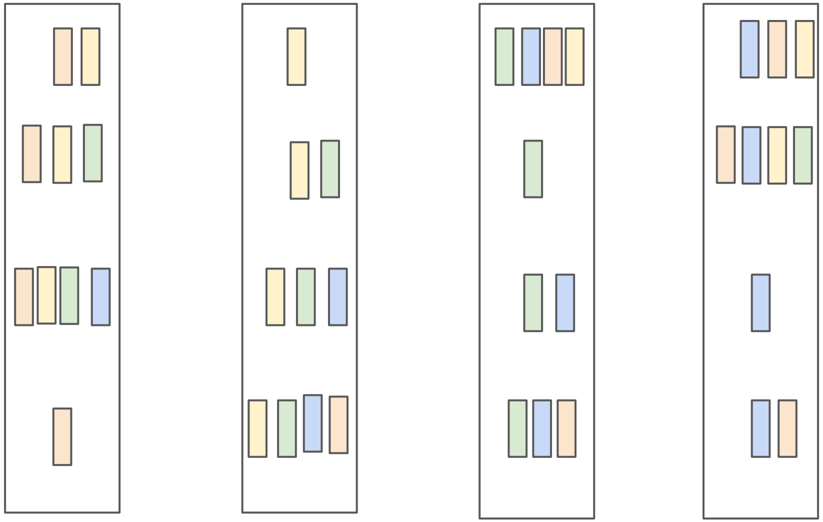
# Ring based Reduction



# Ring based Reduction



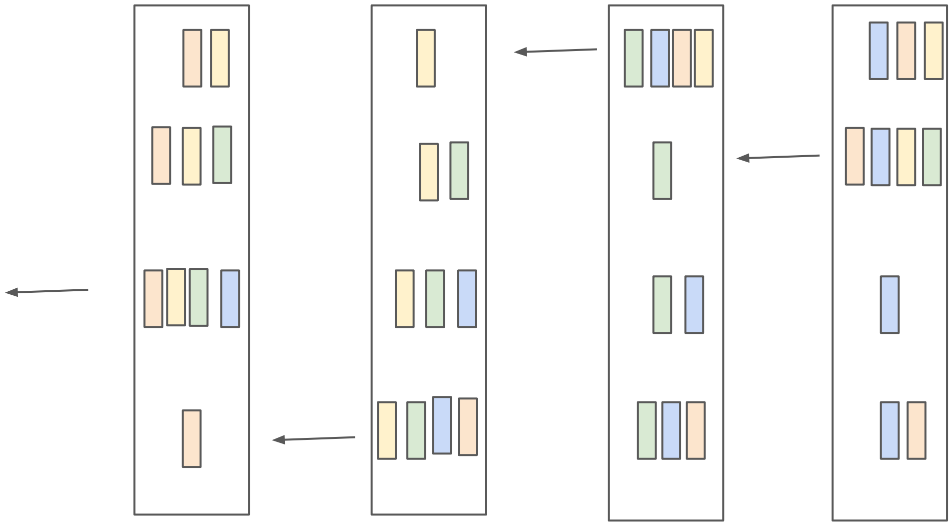
# Ring based Reduction



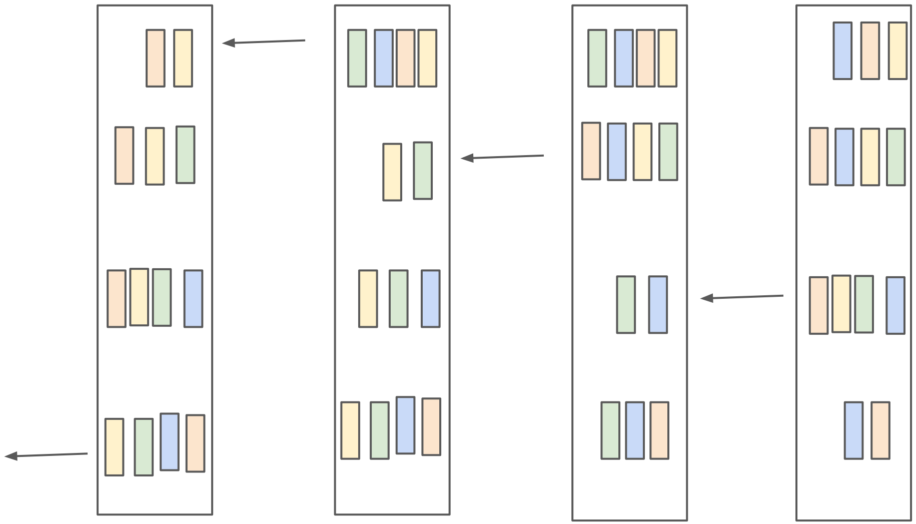
Each node have correctly reduced result of one segment!

This is called *reduce_scatter*

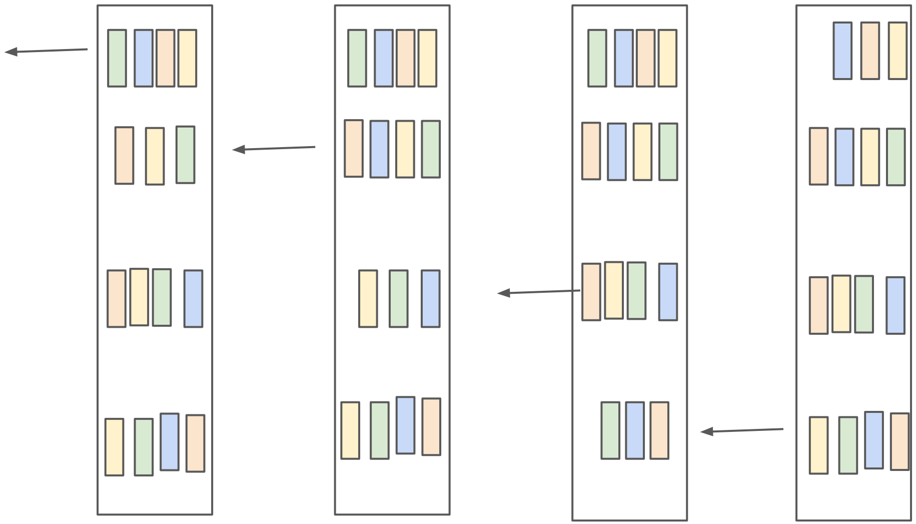
# Ring based Reduction: Allgather phase



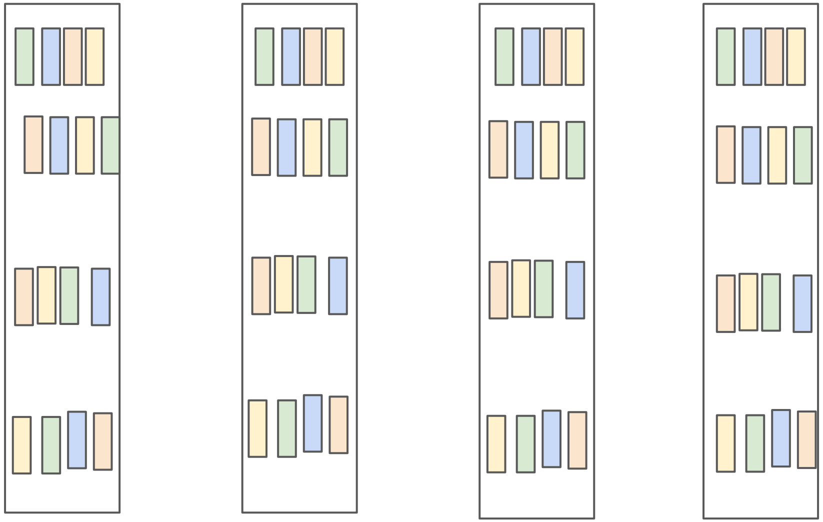
# Ring based Reduction: Allgather phase



# Ring based Reduction: Allgather phase



# Ring based Reduction: Allgather phase

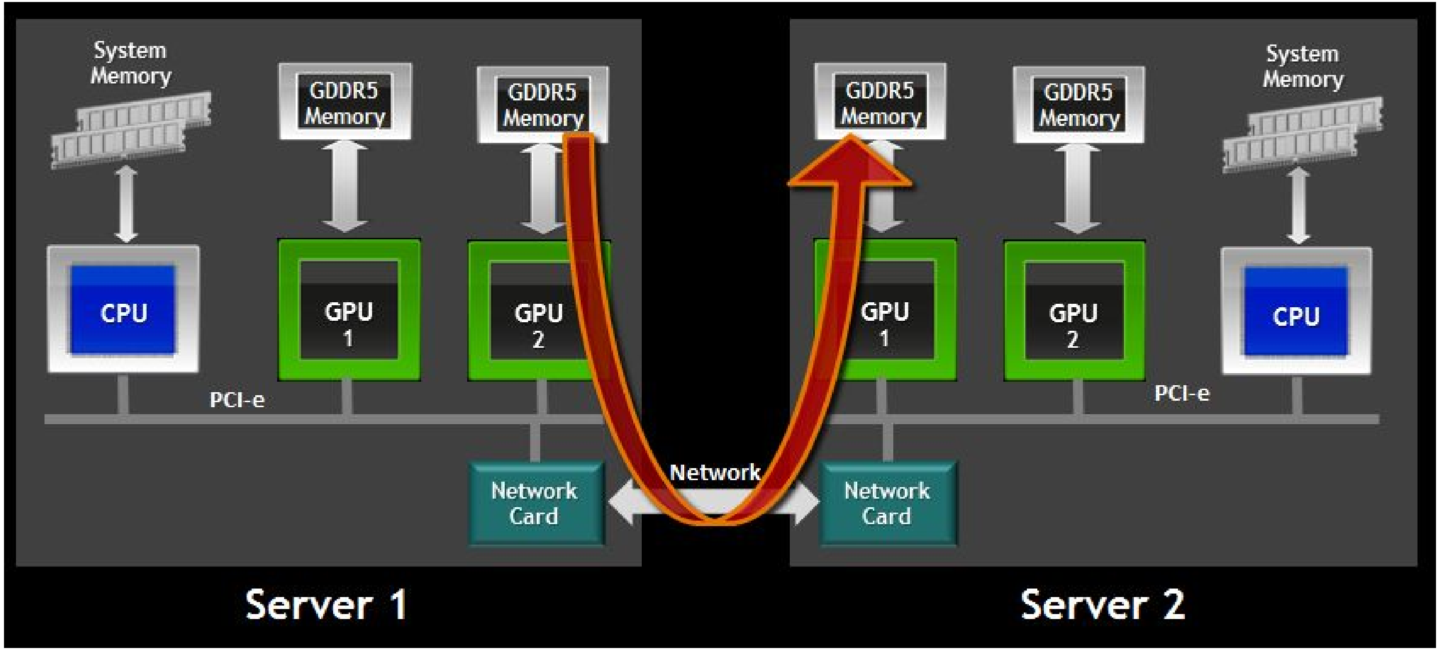


**Question: What is Time Complexity of Ring based Reduction**

# Allreduce Libraries

- MPI offers efficient CPU allreduce
- dmlc/rabit: fault tolerant variant
- facebookincubator/gloo
- NCCL: Nvidia' efficient multiGPU collective

# GPUDirect and RMDA




From 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: 4min

- What are advantages and limitations of Allreduce

- How to integrate allreduce with dependency 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)
```

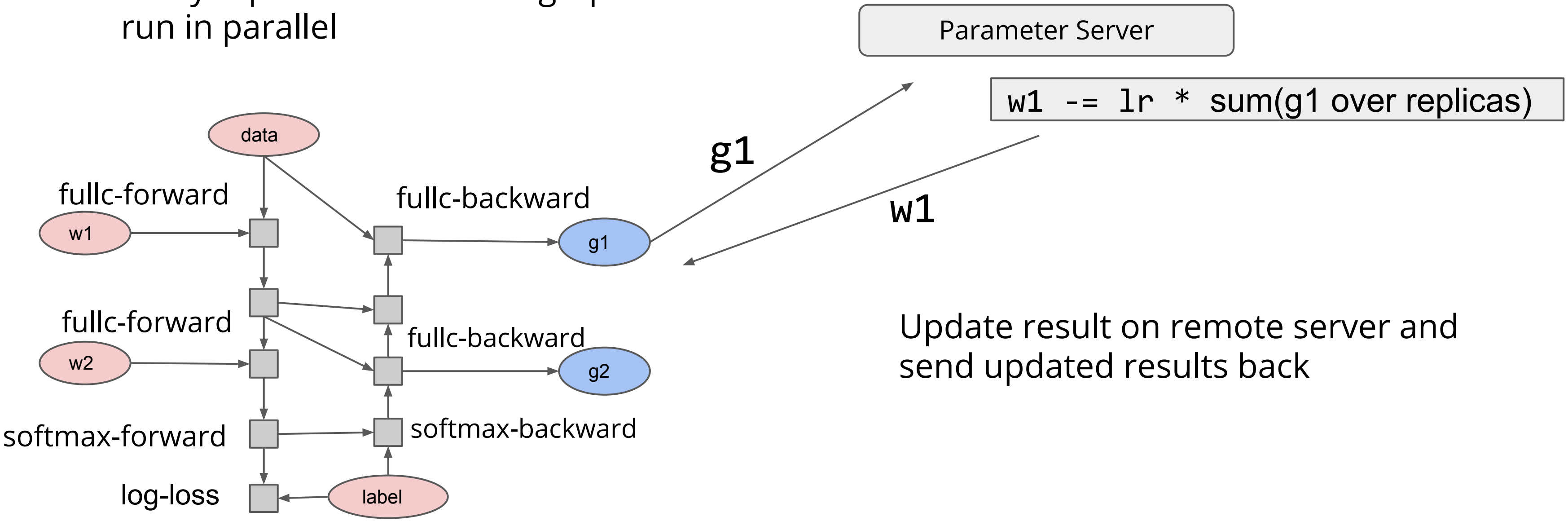
```
engine.push(
   lambda: B.data=A.data+1,
   read=[A.var], mutate=[B.var, comm.var])
```

```
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

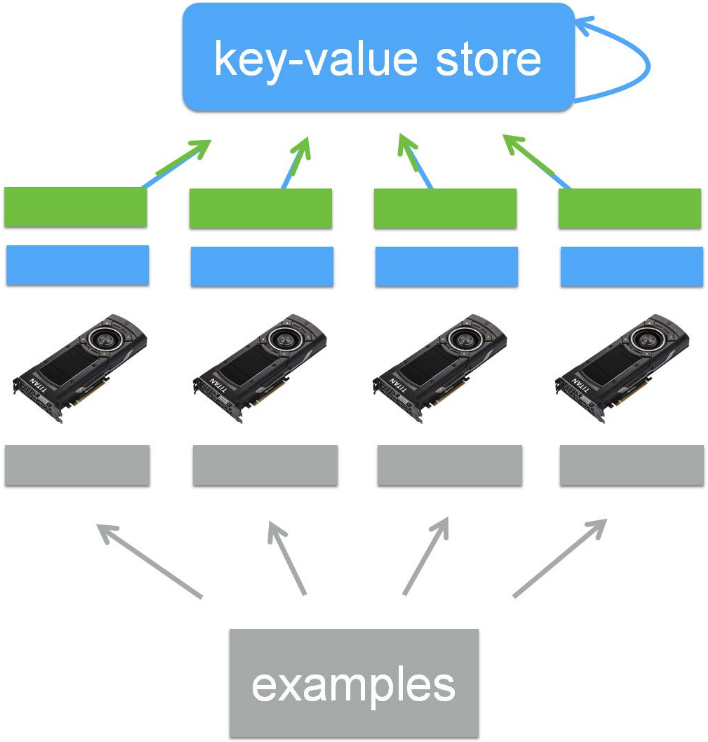



Update result on remote server and
send updated results back

# Parameter Server Abstraction

## Interface

```
ps.push(index, gradient)

ps.pull(index)
```

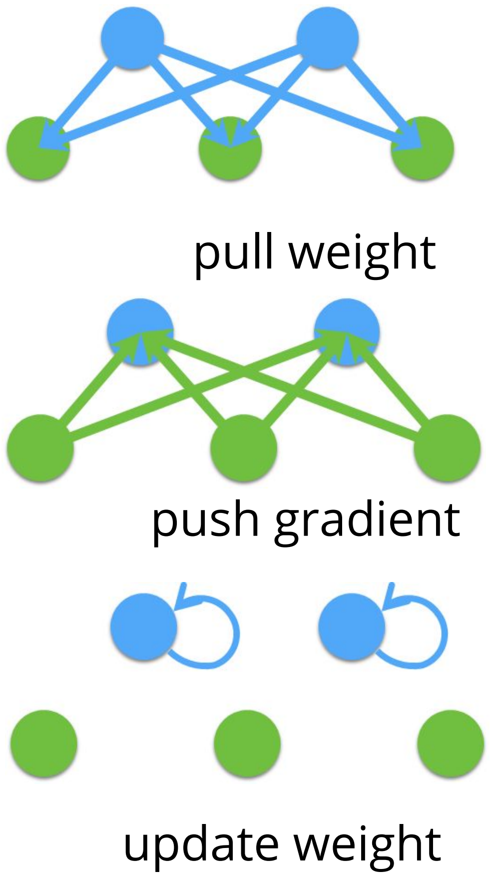


# PS Interface for Data Parallel Training

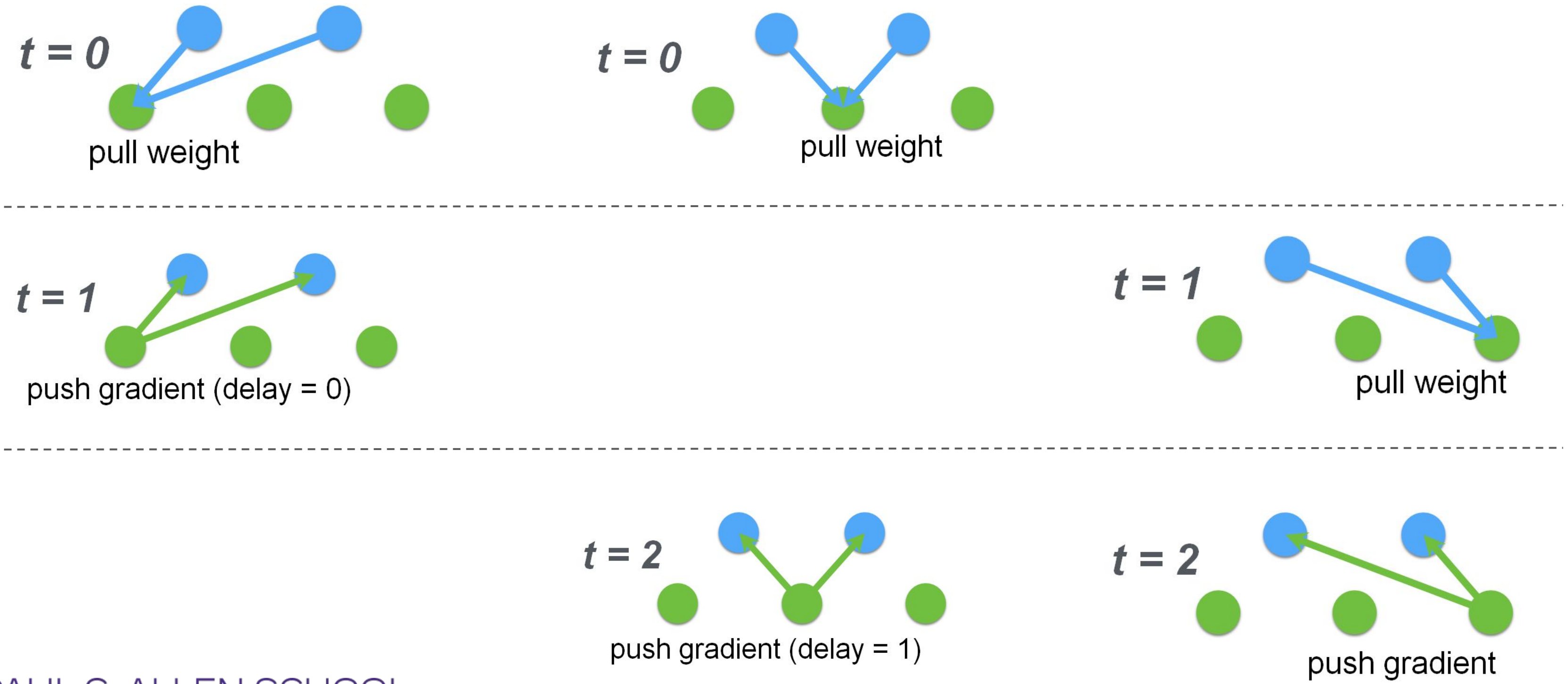
```
grad = gradient(net, w)

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



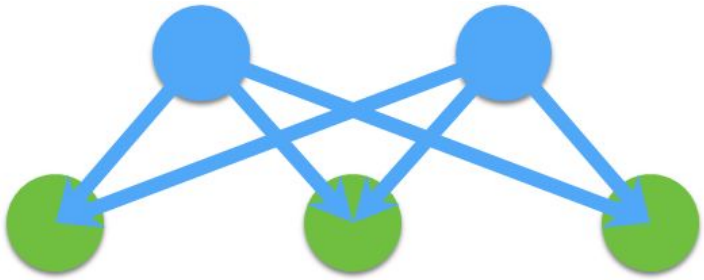
# PS Consistency: Asynchronous




PAUL G. ALLEN SCHOOL
OF COMPUTER SCIENCE & ENGINEERING

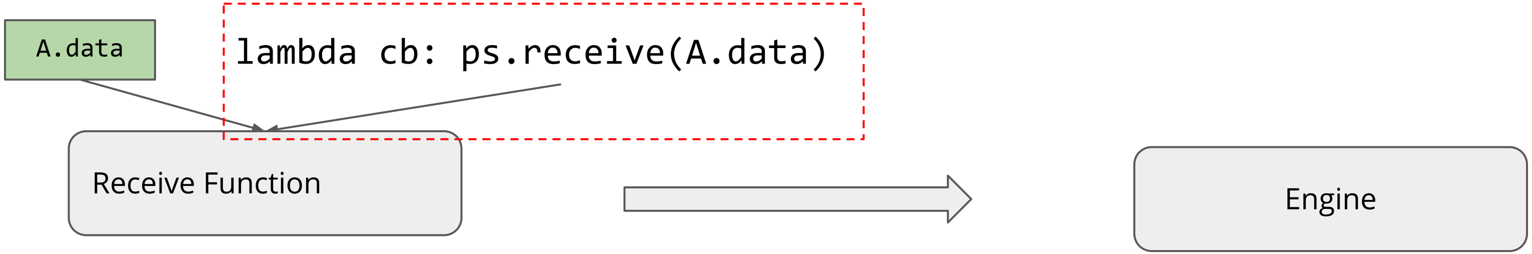
# The Cost of PS Model: All to All Pattern

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



# Integrate Schedule with Networking using Events

**Asynchronous function that takes a callback from engine**

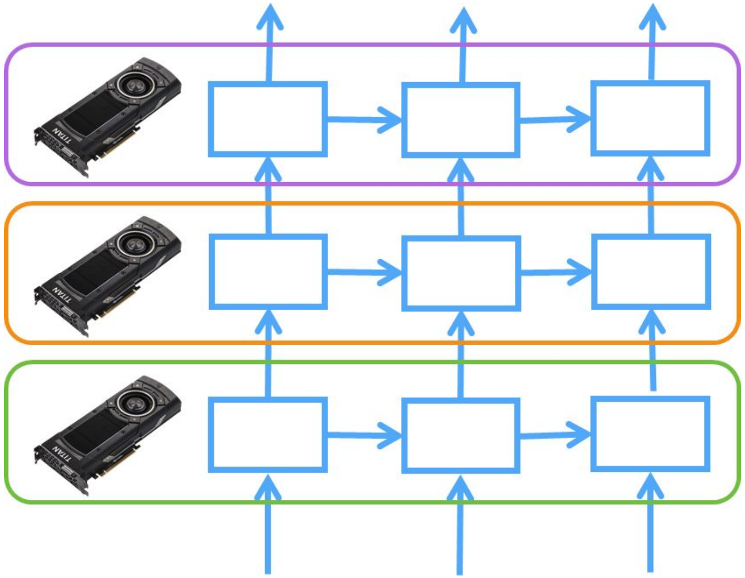



```
def event.on_data_received():
   #notify engine receive complete
   cb();
```

**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

# Discussion: 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