

# 15-442/15-642: Machine Learning Systems

## Memory Optimizations

Spring 2026

Tianqi Chen and Zhihao Jia  
Carnegie Mellon University

# Outline

Activation Checkpointing and Rematerialization

Mixed Precision

Fully Sharded Data Parallelism

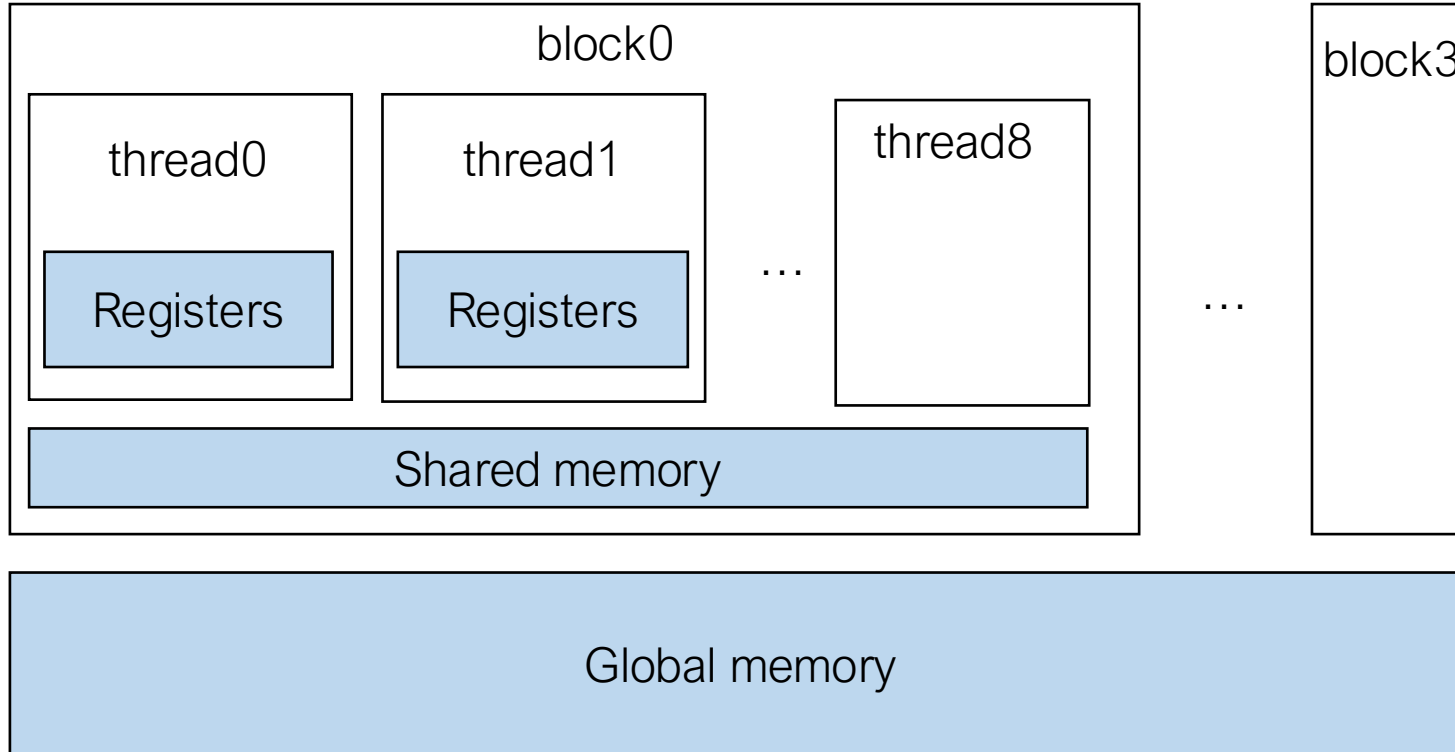
# Outline

Activation Checkpointing and Rematerialization

Mixed Precision

Fully Sharded Data Parallelism

# Recap: GPU memory hierarchy



Shared memory: 64 KB per core

GPU memory(Global memory):

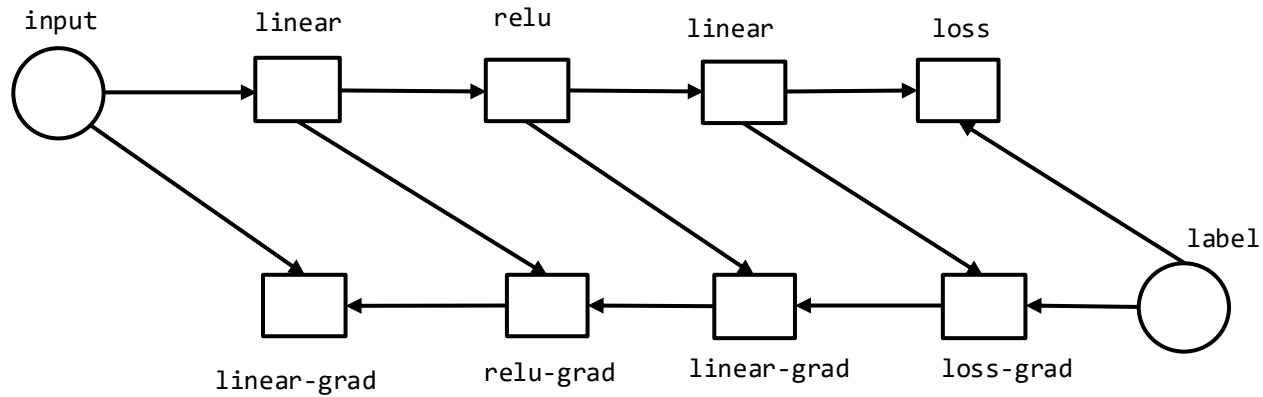
RTX3080 10GB

RTX3090 24GB

A100 40/80 GB

# Sources of memory consumption

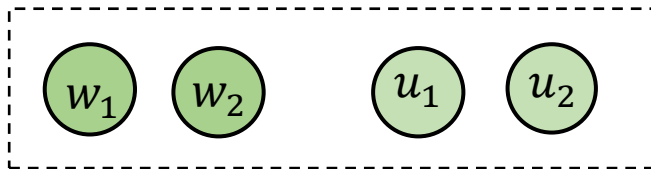
A simplified view of a typical computational graph for training, weights are omitted and implied in the grad steps.



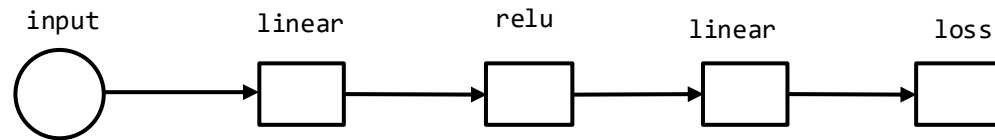
Sources of memory consumption

- Model weights
- Optimizer states
- Intermediate activation values

Optimizer states

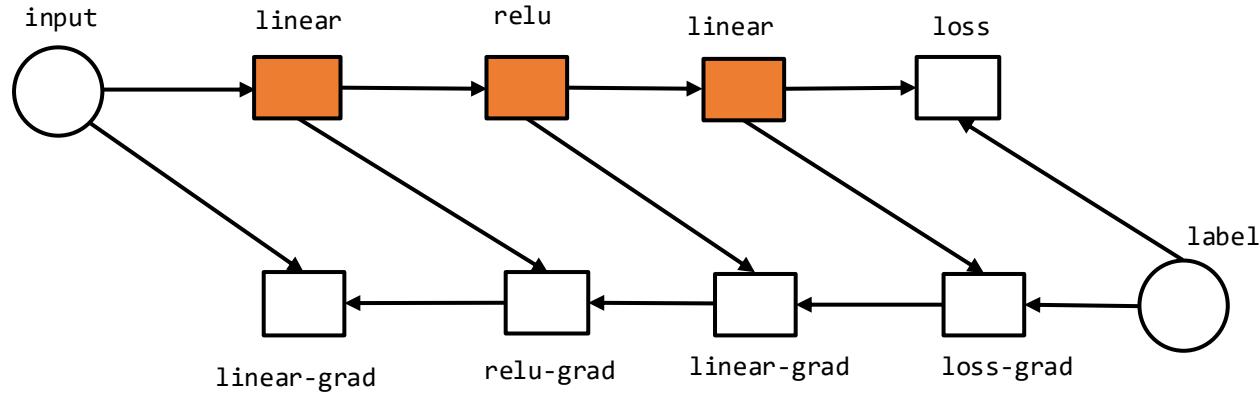


# Techniques for Memory Saving, Inference Only



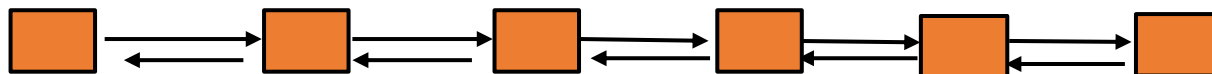
We only need  $O(1)$  memory for computing the final output of a  $N$  layer deep network by cycling through two buffers

# Activation Memory Cost for Training



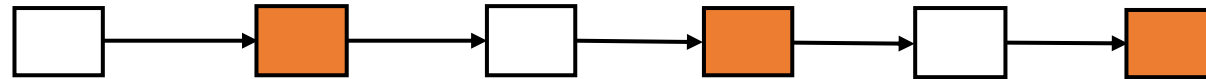
Because the need to keep intermediate value around (checkpoint) for the gradient steps. Training a  $N$ -layer neural network would require  $O(N)$  memory.

We will use the following simplified view to combine gradient and forward computation



# Checkpointing Techniques in AD

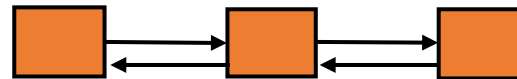
Step 0:



Step 1:



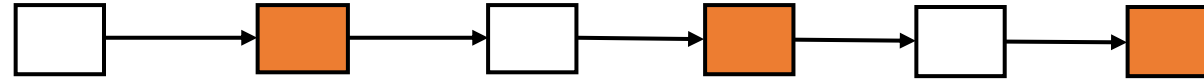
Step 2:



- Only checkpoint colored nodes (step 0)
- Recompute the missing intermediate nodes in small segments (step 1, 2)

# Sublinear Memory Cost

Forward computation



Gradient per segment  
with re-computation



For a  $N$  layer neural network,  
if we checkpoint every  $K$  layers

$$\text{Memory cost} = O\left(\frac{N}{K}\right) + O(K)$$

Checkpoint cost

Re-computation cost

Pick  $K = \sqrt{N}$

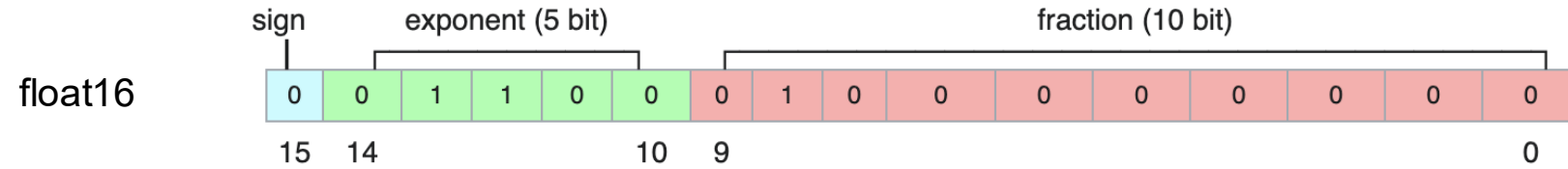
# Outline

Rematerialization

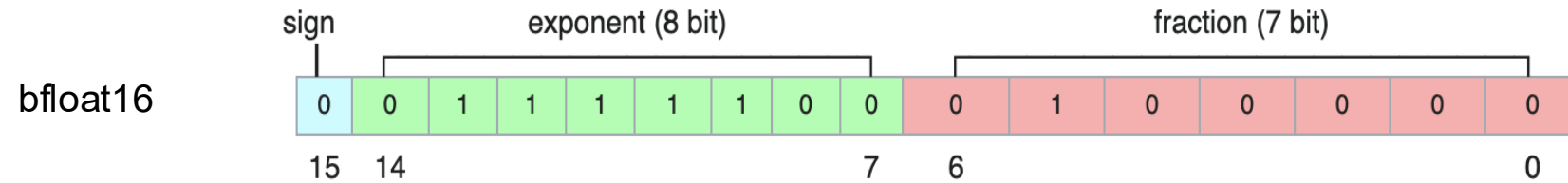
Mixed Precision

Fully Sharded Data Parallelism

# 16bit Floating Points



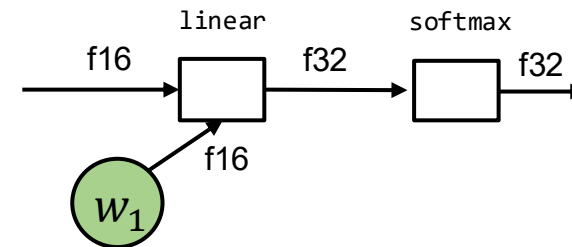
More fraction bits



Less easy to overflow

# Mixed Precision

- Some layers are more sensitive to dynamic range
- Common issues: aggregation of a lot of entries
- Mixed precision: different input/output/accumulation types



# Outline

Activation Checkpointing and Rematerialization

Mixed Precision

**Fully Sharded Data Parallelism**

# Recap: AllReduce Abstraction

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

## Running Example

### Worker 0

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

---

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

### Worker 1

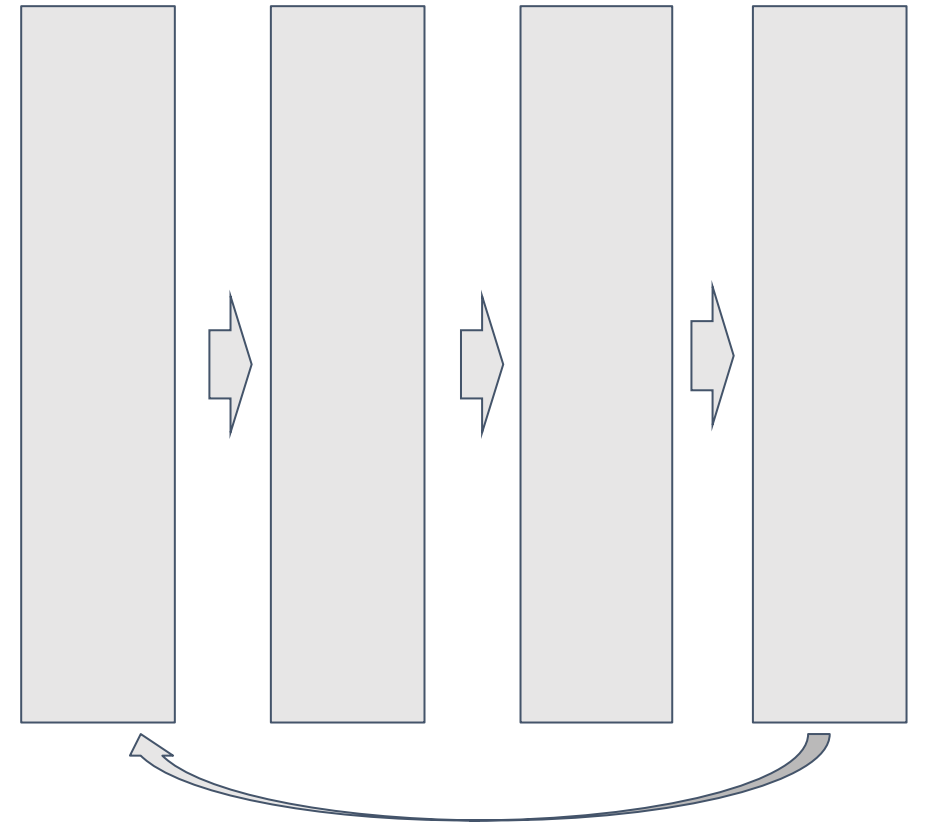
```
comm = communicator.create()  
a = [1, 0, 1]  
b = comm.allreduce(a, op=sum)
```

---

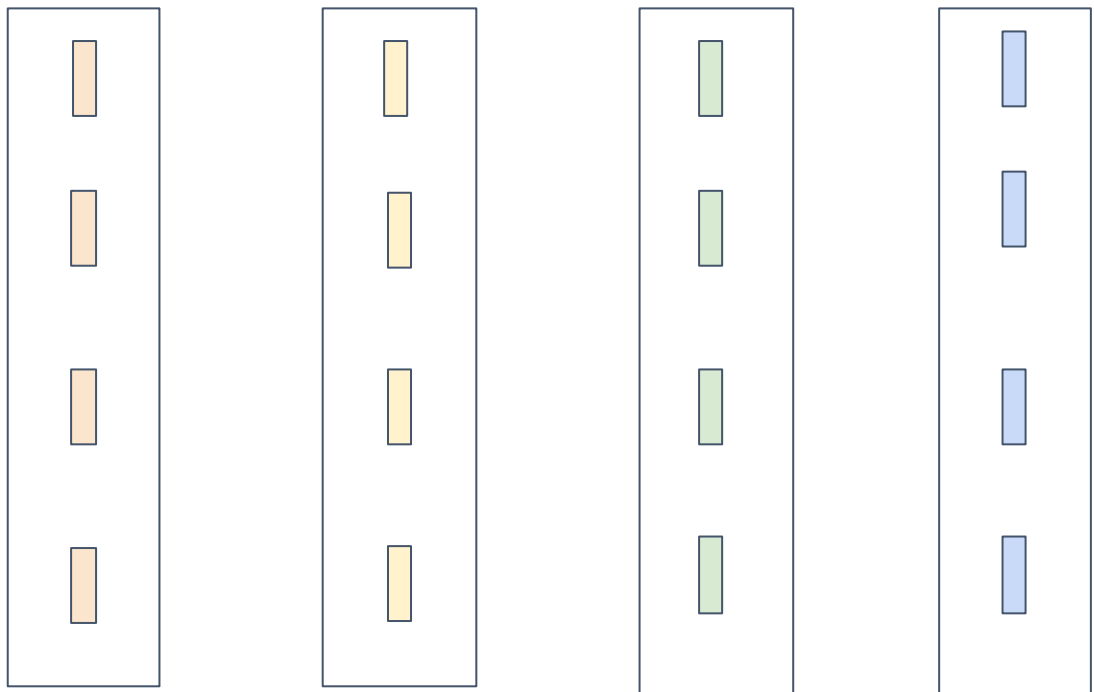
```
assert b == [2, 2, 4]
```

# Ring based Reduction

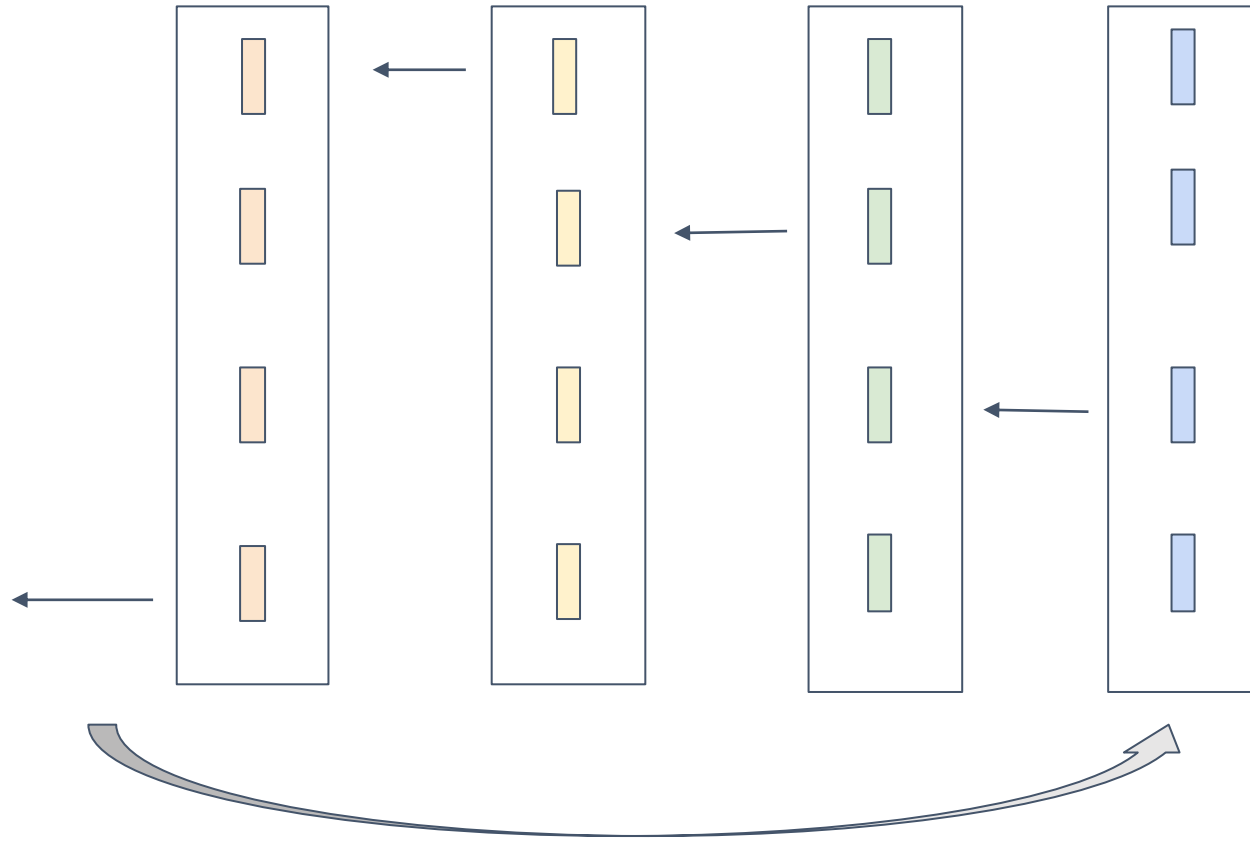
- Form a logical ring between nodes
- Streaming aggregation



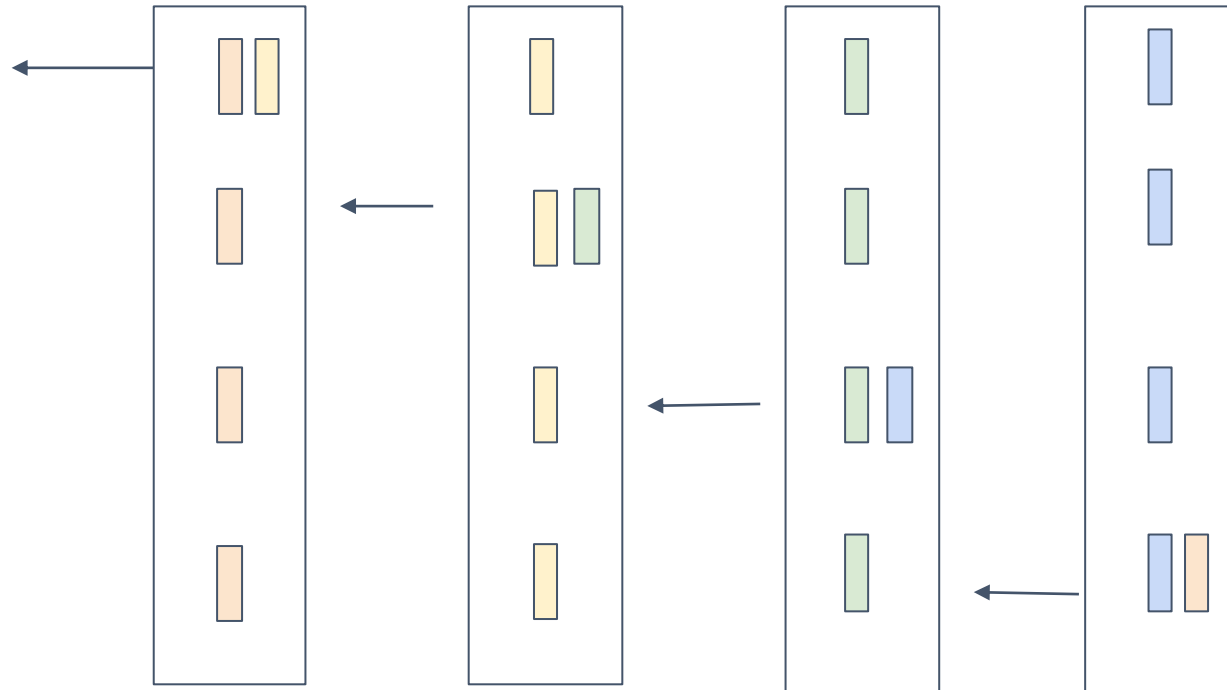
# Ring based Reduction



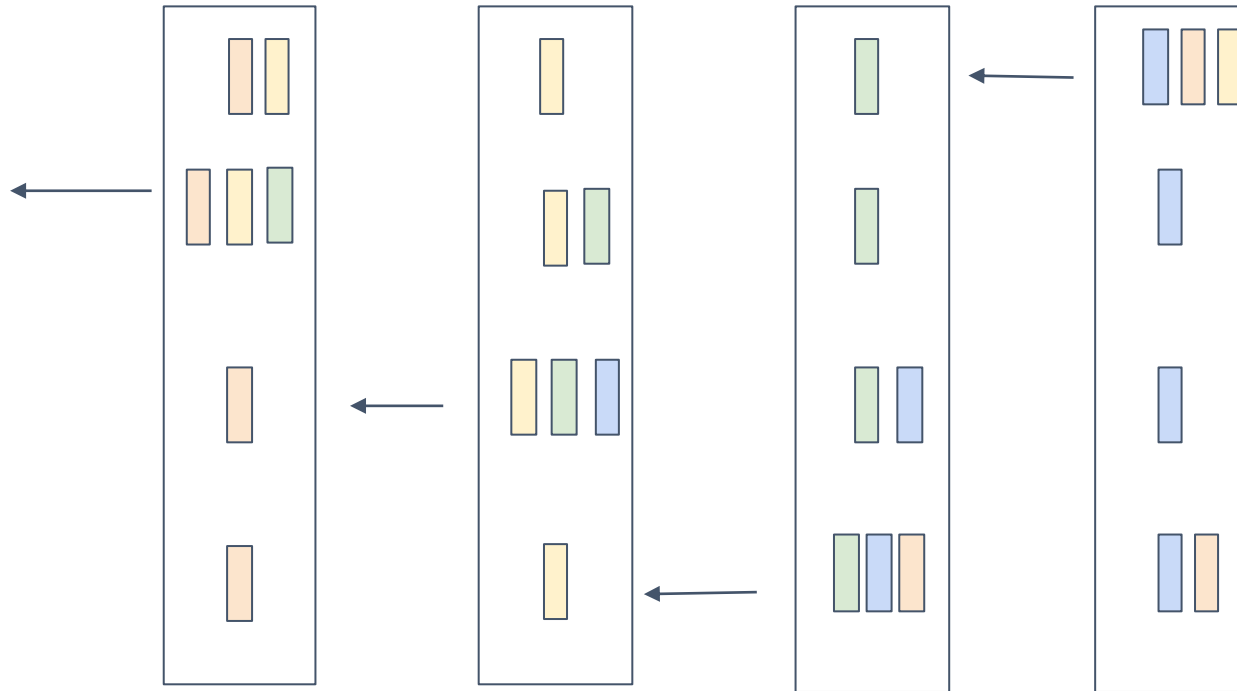
# Ring based Reduction



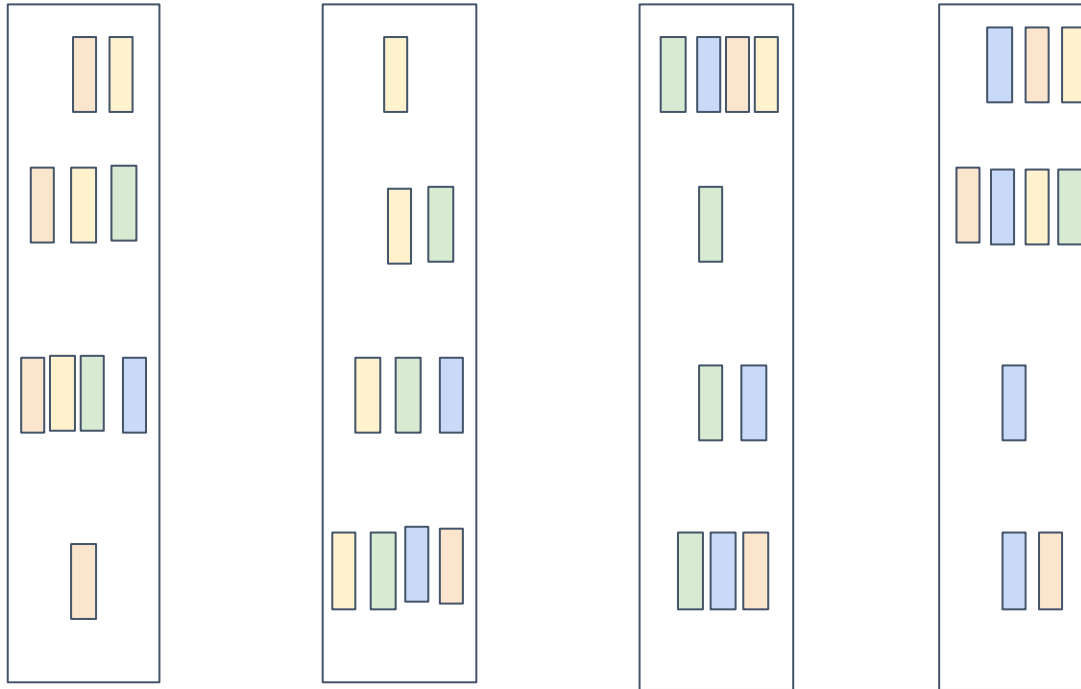
# Ring based Reduction



# Ring based Reduction



# Ring based Reduction



Each node has correctly reduced result of one segment!

This is called *reduce\_scatter*

# Reduce Scatter Abstraction

Interface     `result = reduce_scatter(float buffer[size])`

## Running Example

### Worker 0

```
comm = communicator.create()  
a = [1, 2, 3, 4]  
b = comm.allreduce(a, op=sum)
```

---

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

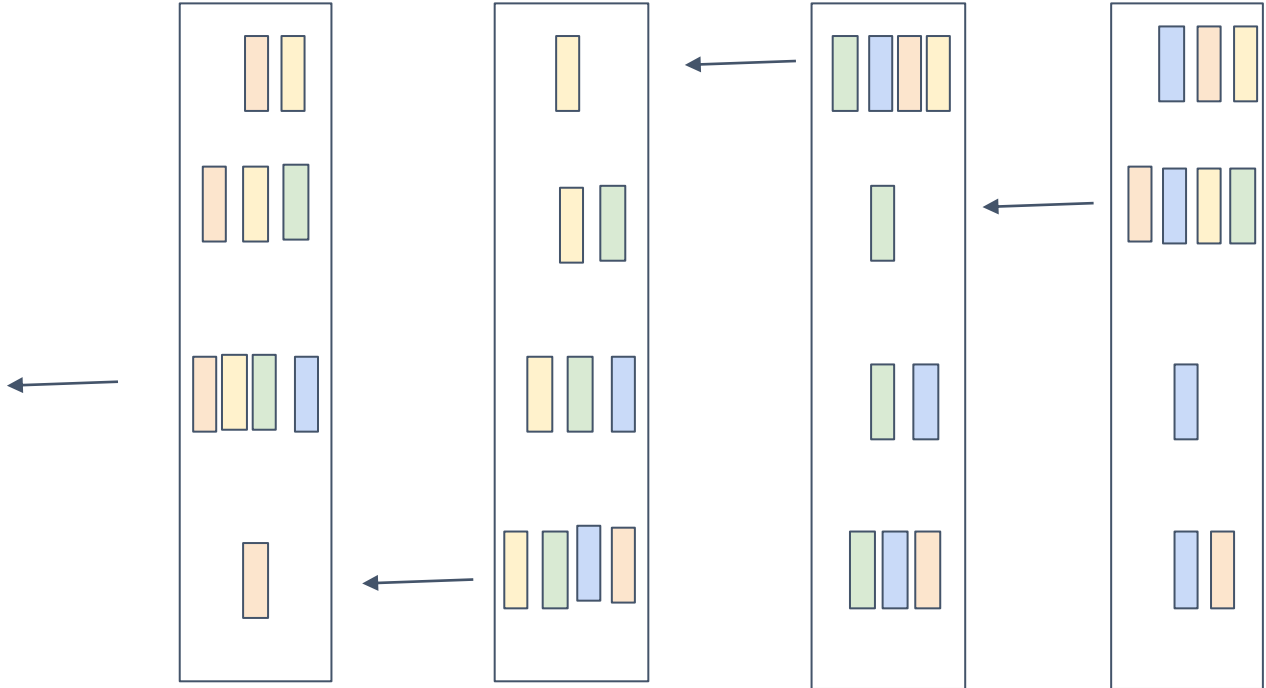
### Worker 1

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

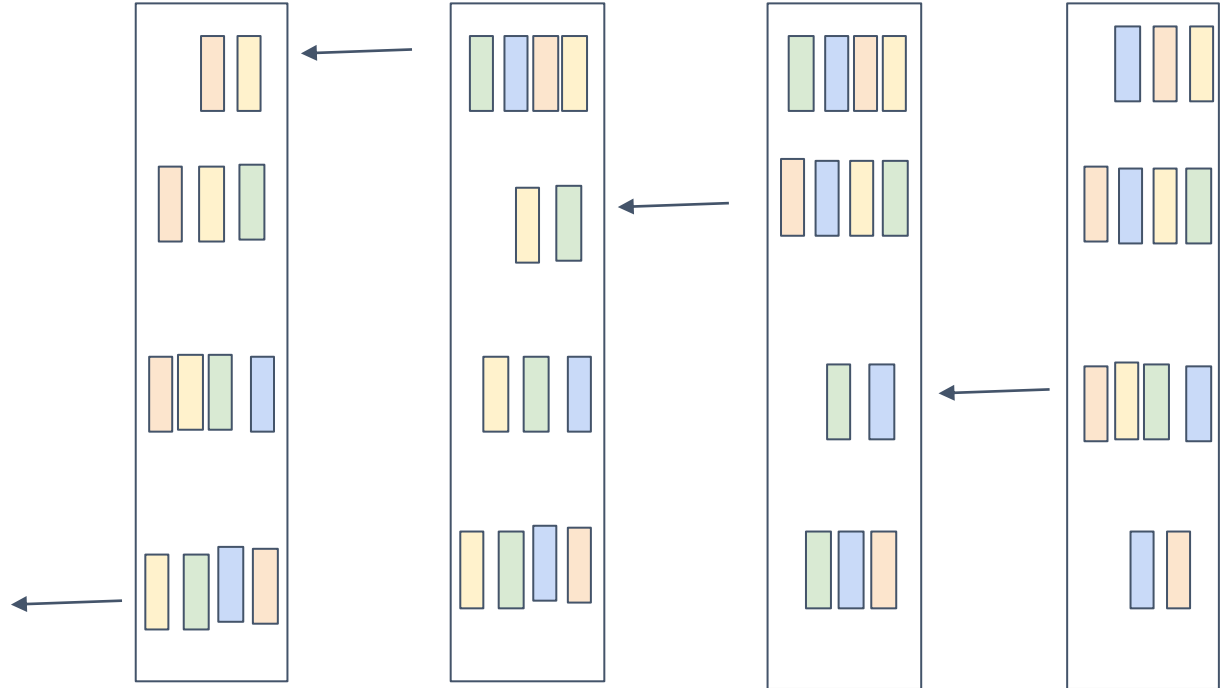
---

```
assert b == [4, 5]
```

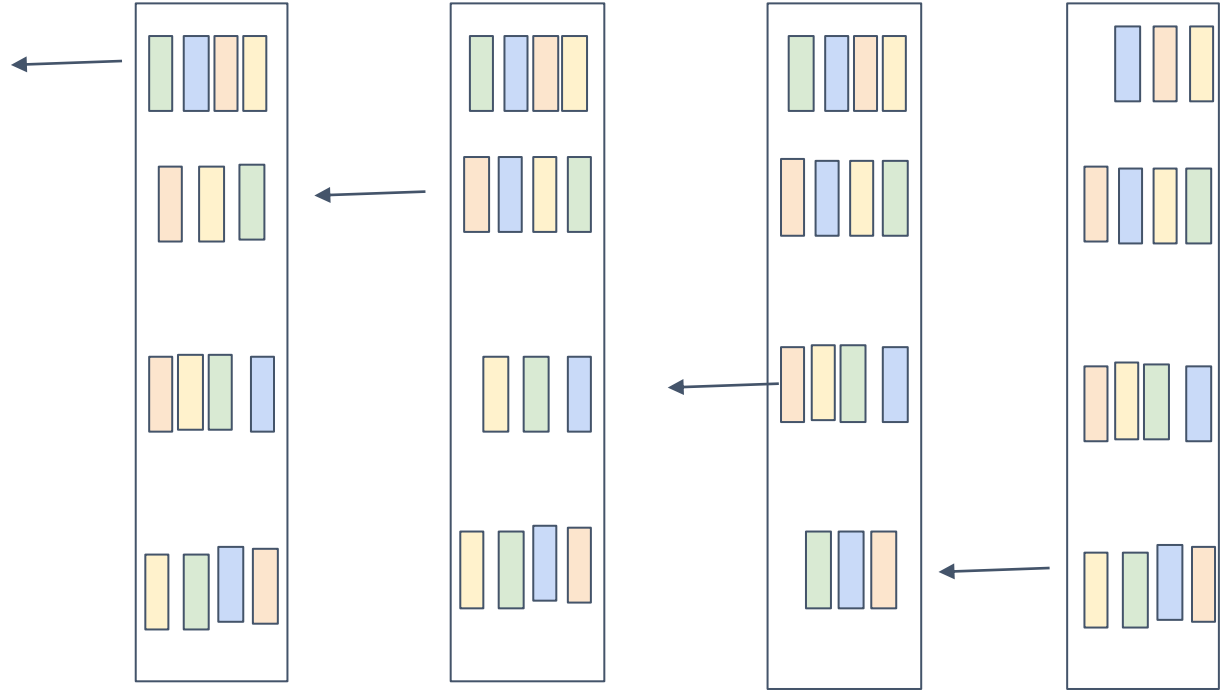
# Ring based Reduction: Allgather phase



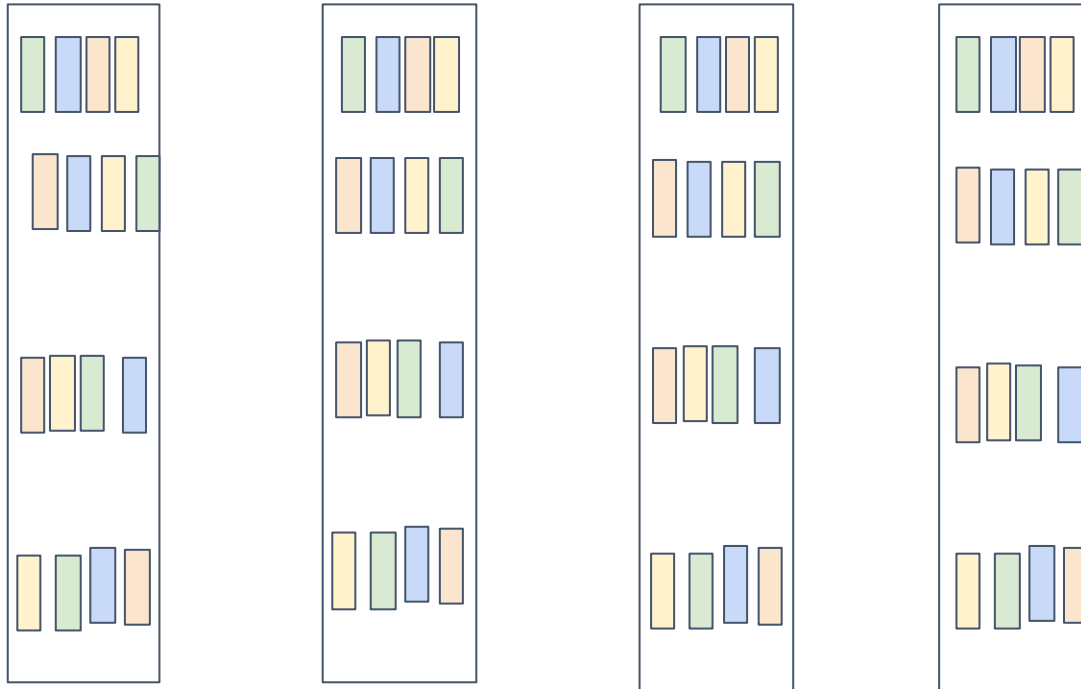
# Ring based Reduction: Allgather phase



# Ring based Reduction: Allgather phase



# Ring based Reduction: Allgather phase



Question: What is  
Time Complexity of  
Ring based Reduction

# Allgather abstraction

Interface     `result = allgather(float buffer[size])`

## Running Example

### Worker 0

```
comm = communicator.create()  
a = [1, 2]  
b = comm.allgather(a)
```

---

```
assert b == [1, 2, 3, 4]
```

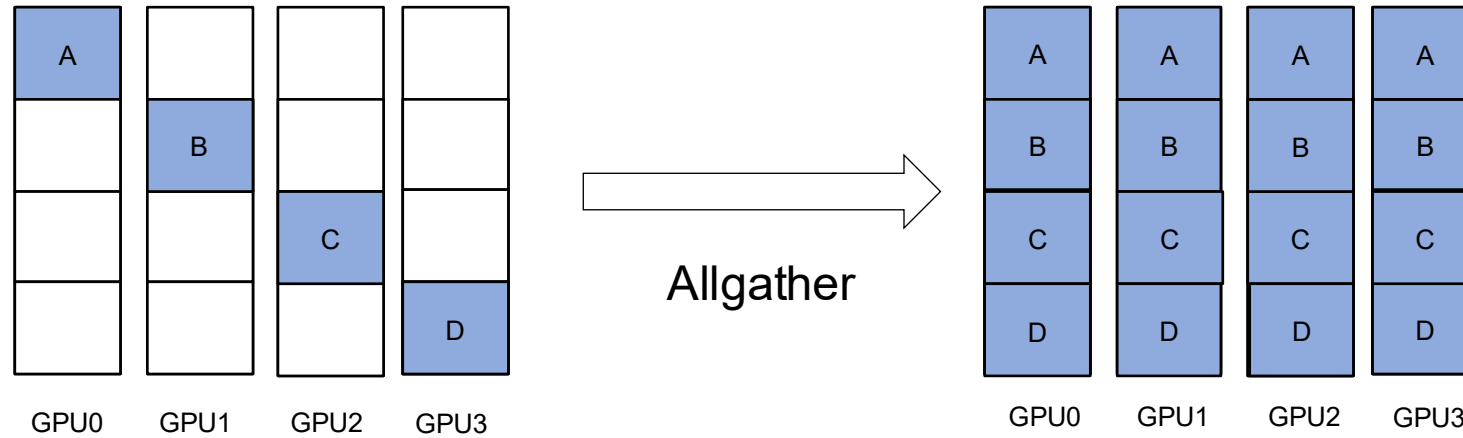
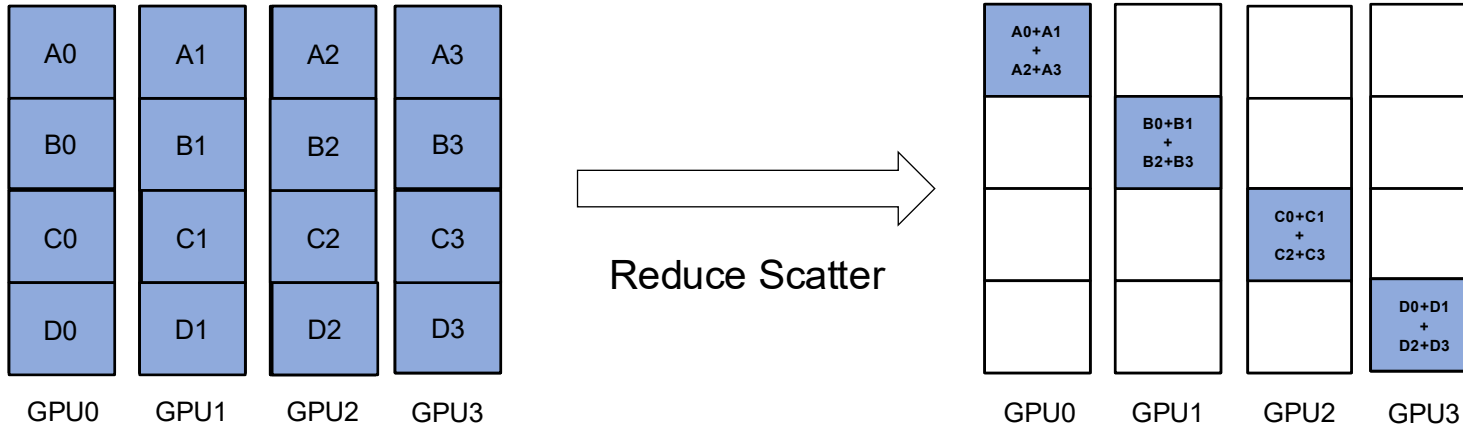
### Worker 1

```
comm = communicator.create()  
a = [3, 4]  
b = comm.allgather(a)
```

---

```
assert b == [1, 2, 3, 4]
```

# Overall Relations



Combine both we get Allreduce

# FSDP: Fully Sharded Data Parallel

