CS 498: Machine Learning System
Spring 2025

Minjia Zhang

The Grainger College of Engineering

Today

- **Distributed Training Preliminary**
  - Problem
  - Challenge
  - History
  - Parameter Server based DP
- **Data Parallelism**
  - **Allreduce-based DP**
  - **Communication terminologies**
- **Hardware**

Data Parallelism: DDP

```python
import torch
import torch.nn as nn
import torch.optim as optim

# initialize torch.distributed properly
# with init_process_group

# setup model and optimizer
net = nn.Linear(10, 10)
opt = optim.SGD(net.parameters(), lr=0.01)

# run forward pass
inp = torch.randn(20, 10)
exp = torch.randn(20, 10)
out = net(inp)

# run backward pass
nn.MSELoss()(out, exp).backward()

# update parameters
opt.step()
```

Data Parallelism: DDP

```python
import torch
import torch.nn as nn
import torch.optim as optim

# initialize torch.distributed properly
# with init_process_group

# setup model and optimizer
net = nn.Linear(10, 10)
opt = optim.SGD(net.parameters(), lr=0.01)

# run forward pass
inp = torch.randn(20, 10)
exp = torch.randn(20, 10)
out = net(inp)

# run backward pass
nn.MSELoss()(out, exp).backward()

# update parameters
opt.step()
```

```python
import torch
import torch.nn as nn
import torch.nn.parallel as par
import torch.optim as optim

# initialize torch.distributed properly
# with init_process_group

# setup model and optimizer
net = nn.Linear(10, 10)
net = par.DistributedDataParallel(net)
opt = optim.SGD(net.parameters(), lr=0.01)

# run forward pass
inp = torch.randn(20, 10)
exp = torch.randn(20, 10)
out = net(inp)

# run backward pass
nn.MSELoss()(out, exp).backward()

# update parameters
opt.step()
```

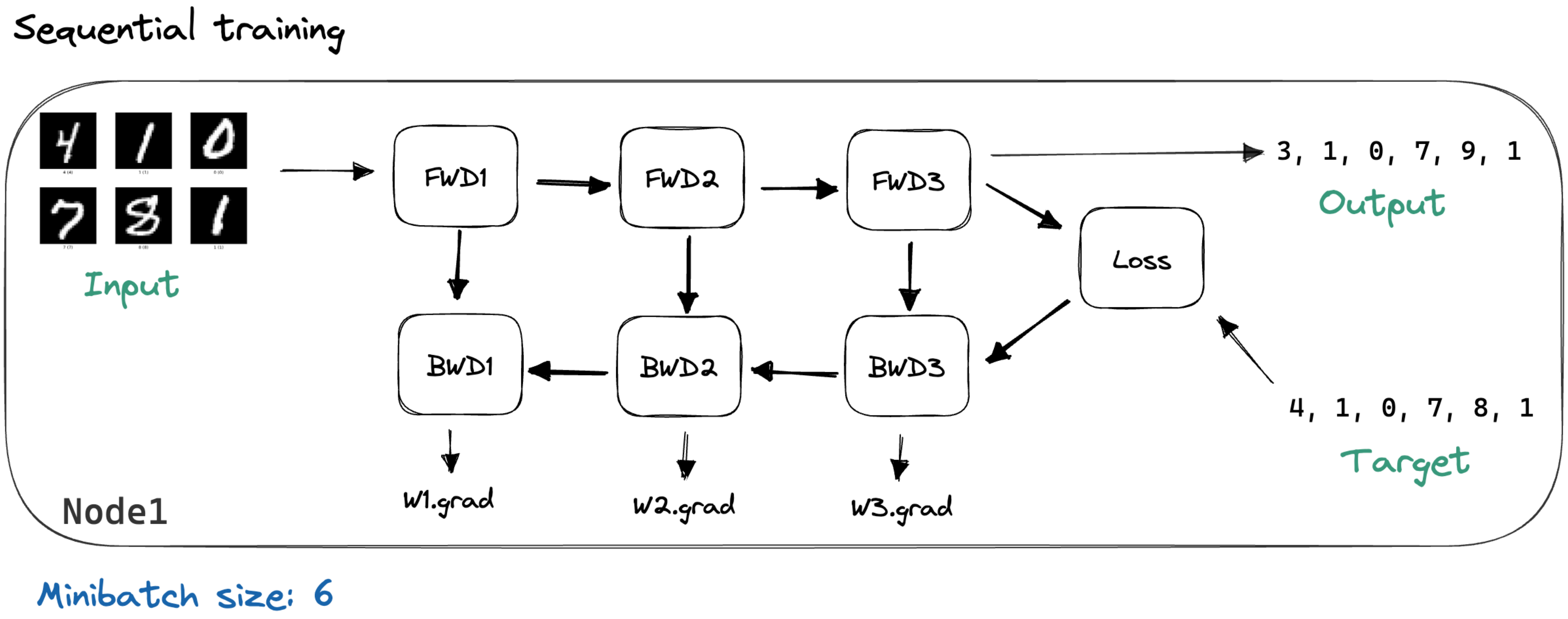
Data Parallelism: DDP

```python
import torch
import torch.nn as nn
import torch.optim as optim

# initialize torch.distributed properly
# with init_process_group

# setup model and optimizer
net = nn.Linear(10, 10)
opt = optim.SGD(net.parameters(), lr=0.01)

# run forward pass
inp = torch.randn(20, 10)
exp = torch.randn(20, 10)
out = net(inp)

# run backward pass
nn.MSELoss()(out, exp).backward()

# update parameters
opt.step()
```

```python
import torch
import torch.nn as nn
import torch.nn.parallel as par
import torch.optim as optim

# initialize torch.distributed properly
# with init_process_group

# setup model and optimizer
net = nn.Linear(10, 10)
net = par.DistributedDataParallel(net)
opt = optim.SGD(net.parameters(), lr=0.01)

# run forward pass
inp = torch.randn(20, 10)
exp = torch.randn(20, 10)
out = net(inp)

# run backward pass
nn.MSELoss()(out, exp).backward()

# update parameters
opt.step()
```

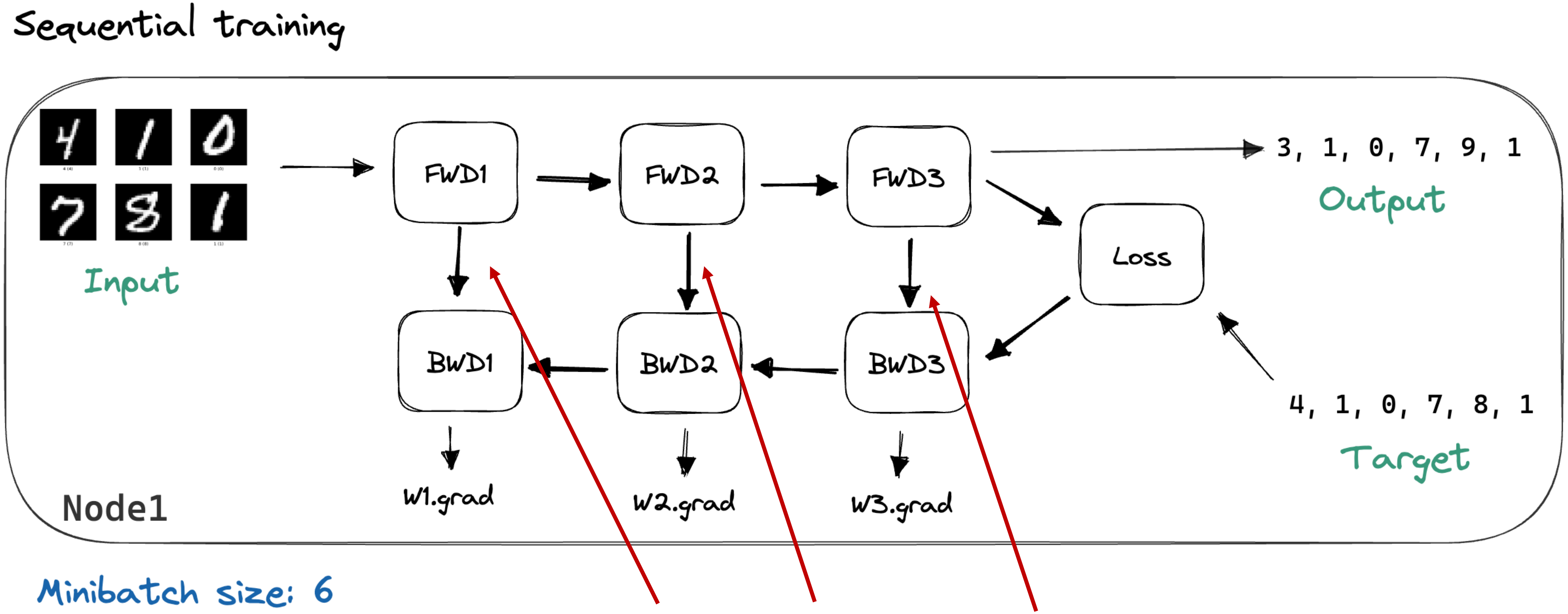
Gradient synchronization

Data Parallelism: AllReduce-based



https://siboehm.com/articles/22/data-parallel-training

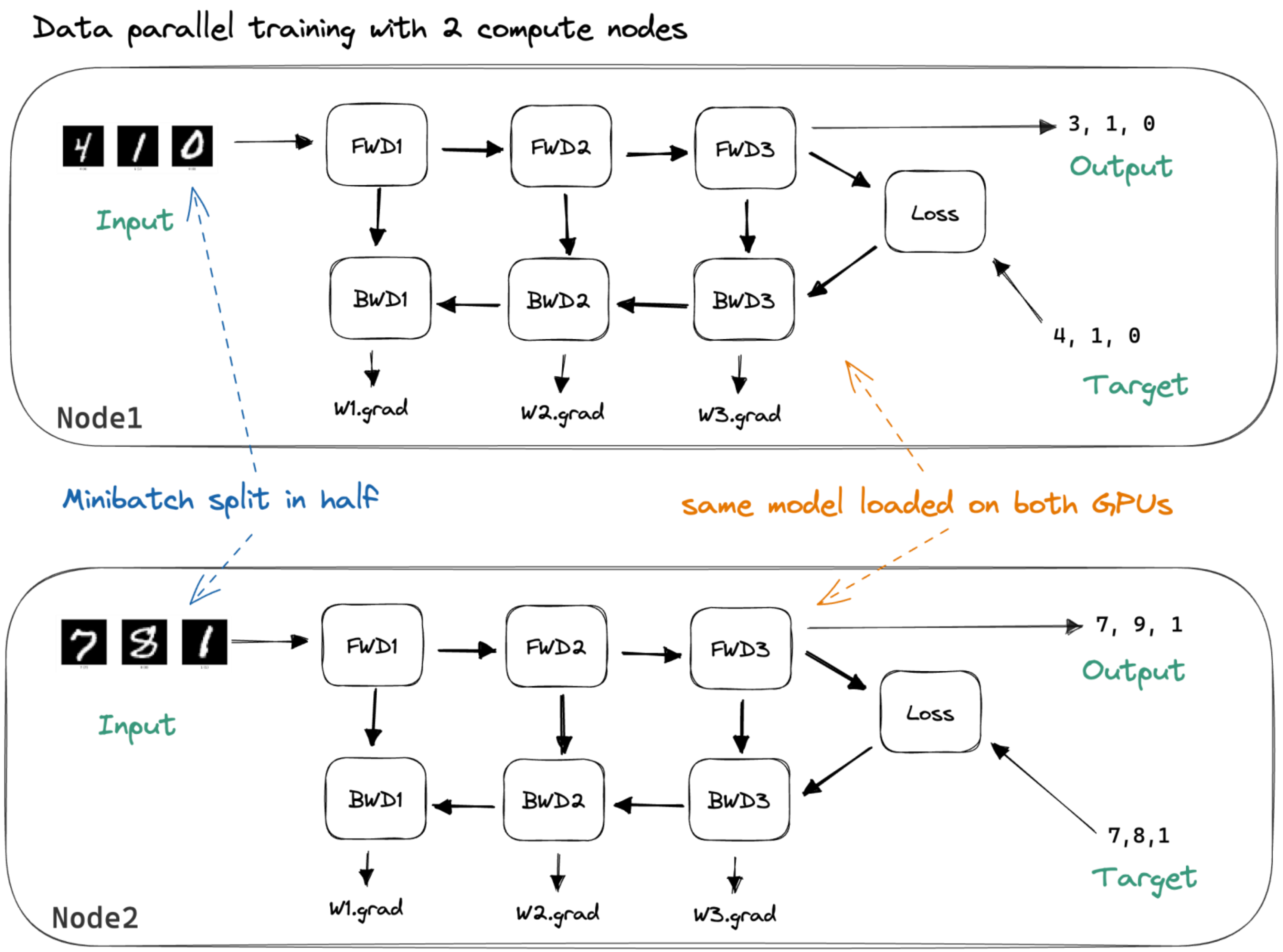
Data Parallelism: AllReduce-based



Question: why do we have these data dependency edges?

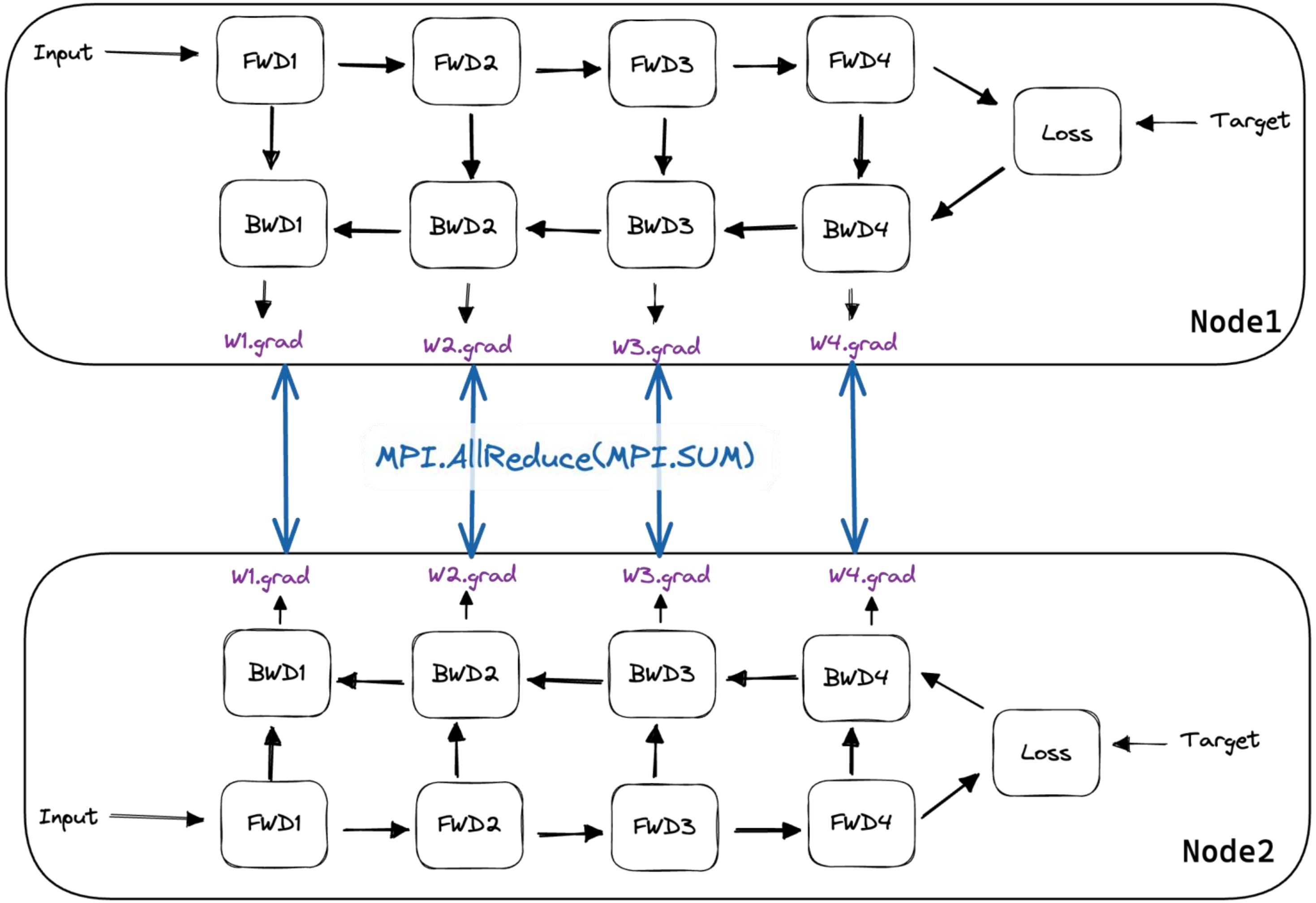
https://siboehm.com/articles/22/data-parallel-training

AllReduce based Data Parallelism



https://siboehm.com/articles/22/data-parallel-training

AllReduce based Data Parallelism

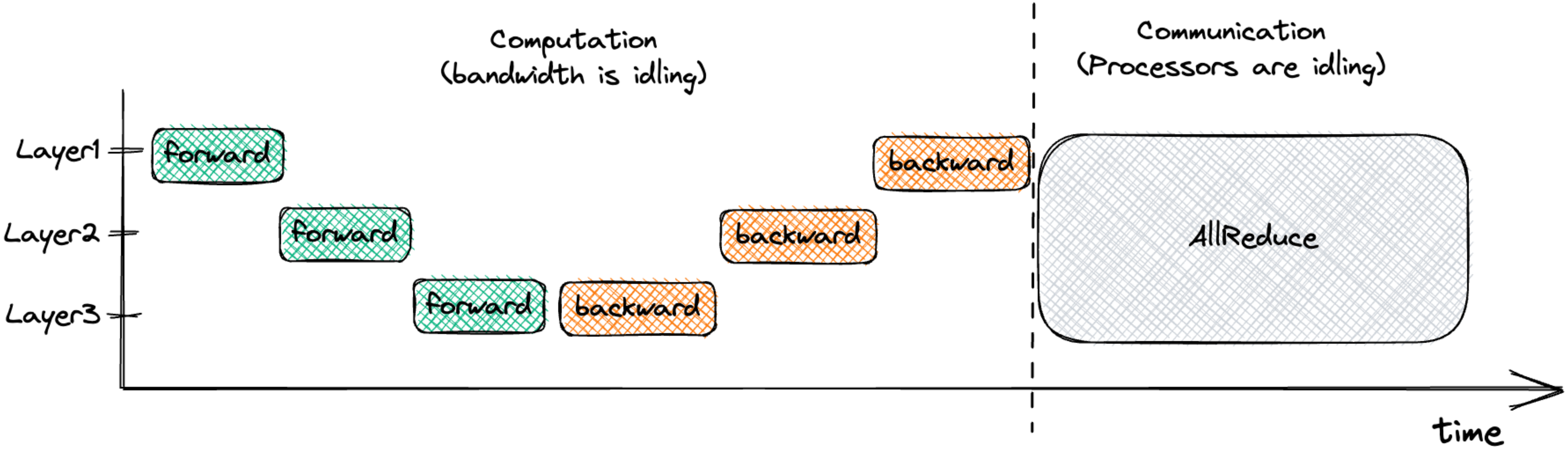


https://siboehm.com/articles/22/data-parallel-training

When Shall we Synchronize Gradients?

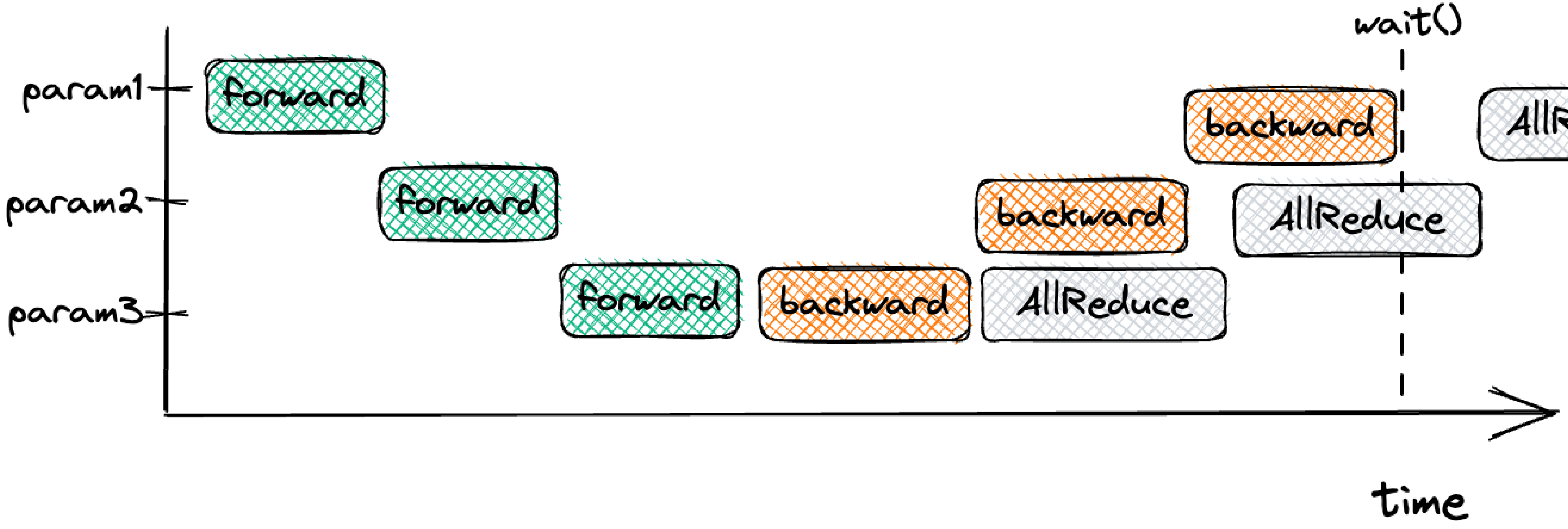
```
grad = gradient(net, w)

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

  w -= lr * gsum / num_workers
```

AllReduce based Data Parallelism: Option 1



https://siboehm.com/articles/22/data-parallel-training

AllReduce based Data Parallelism: Option 2



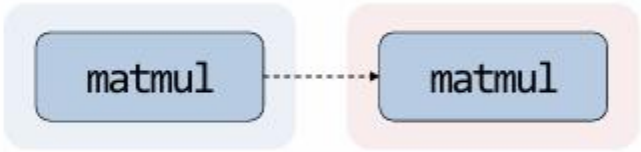
https://siboehm.com/articles/22/data-parallel-training

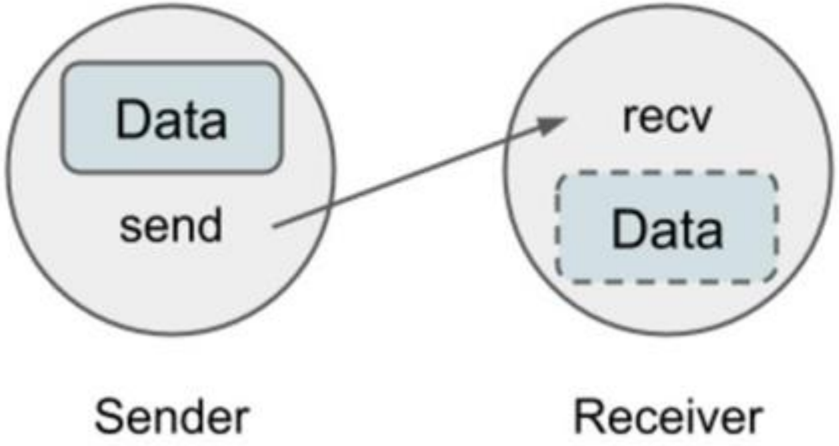
Today

- **Distributed Training Preliminary**
  - **Problem**
  - **Challenge**
  - **History**
  - **Parameter Server based DP**

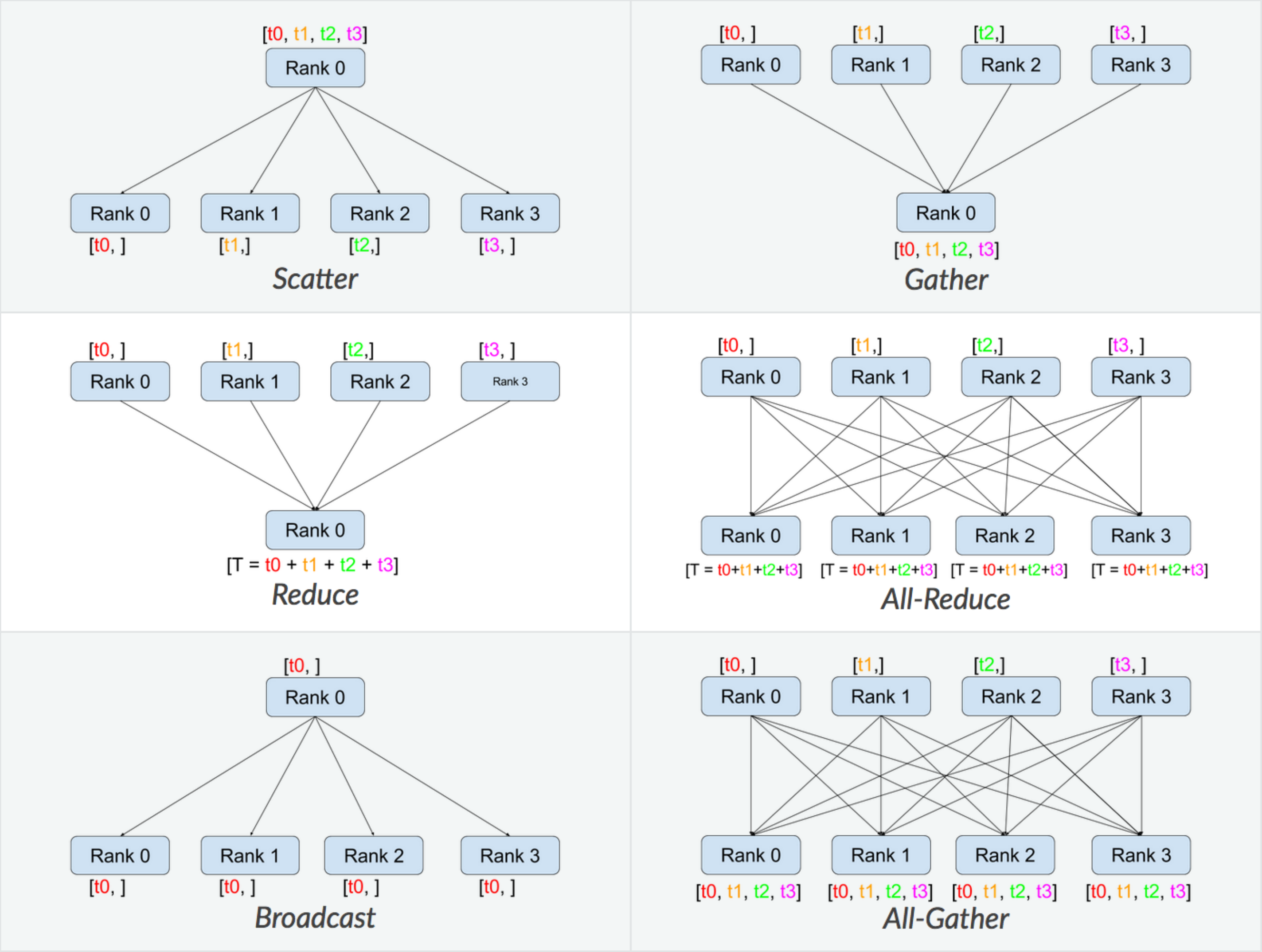
- **Data Parallelism**
  - **Allreduce-based DP**
  - **Communication terminologies**
- **Hardware**

Terminologies: Point-to-Point Primitive Communication

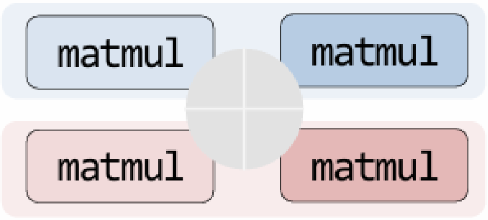




Terminologies: Collectives

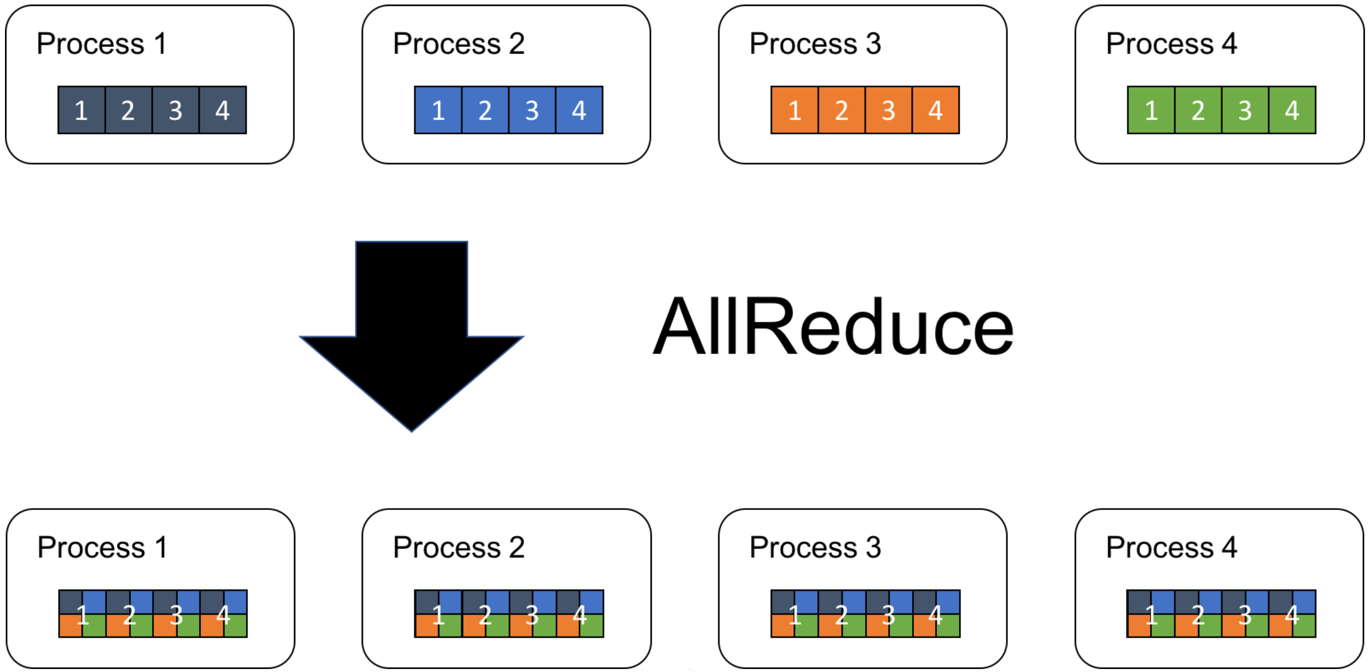


Terminologies: Communication Collective



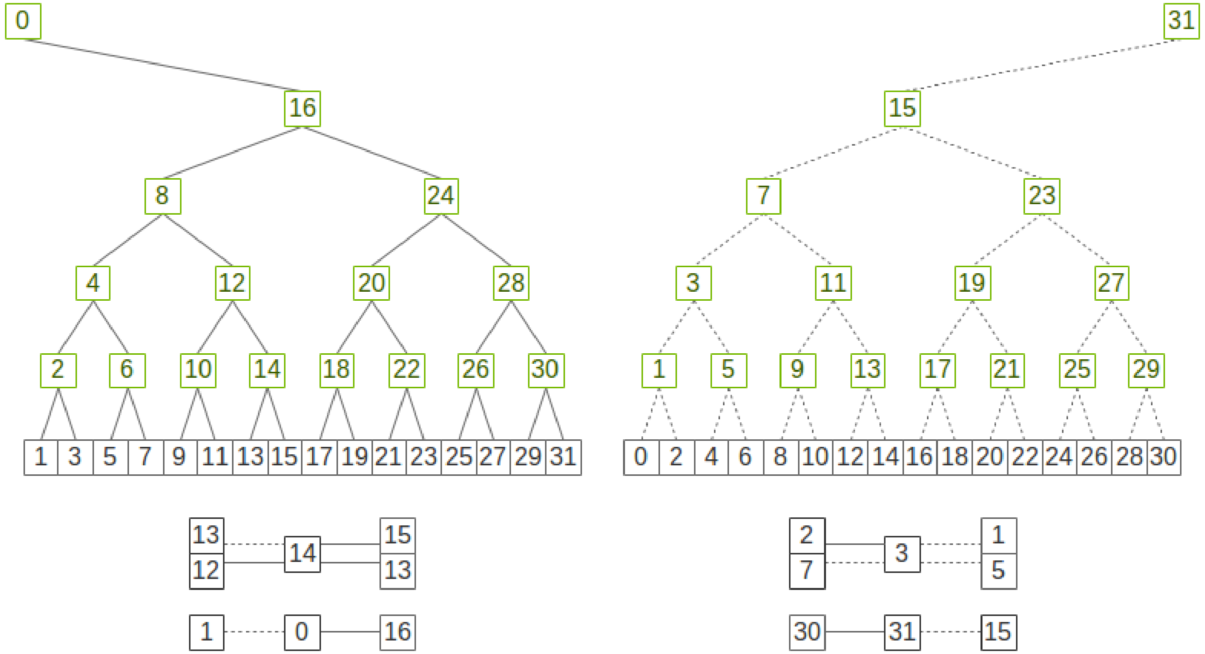

```python
ddp_model = DDP(Model(), device_ids=[rank])
for batch in data_loader:
    loss = train_step(ddp_model, batch)
```

Implicit allreduce here

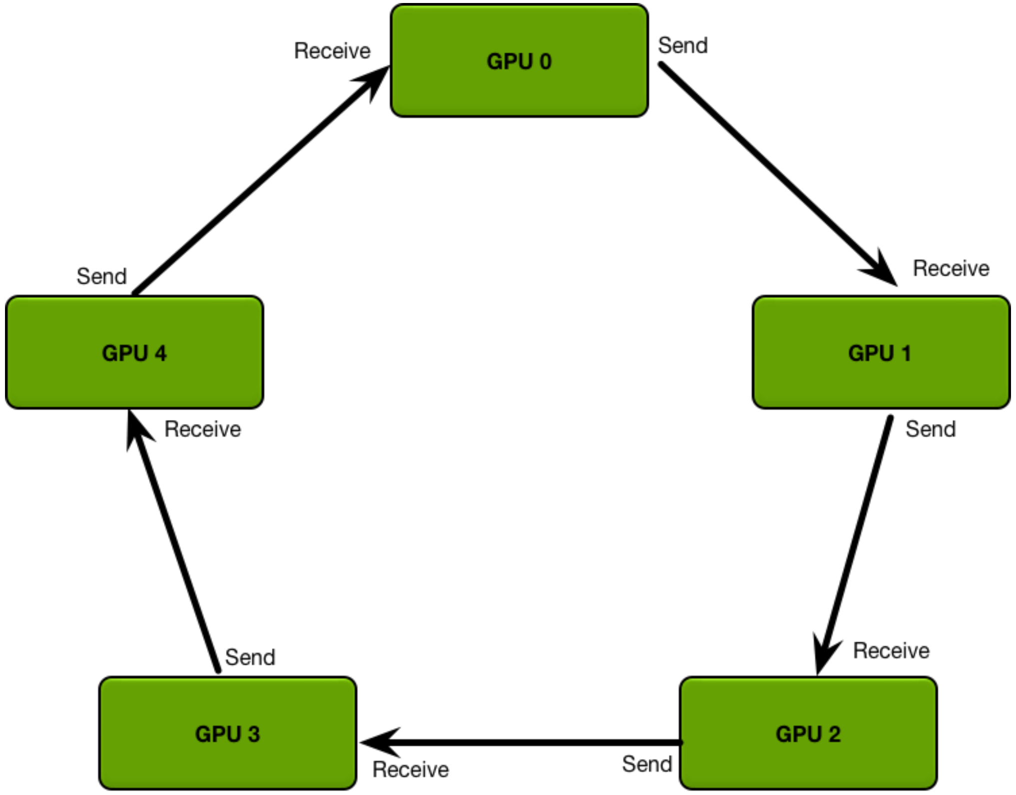


Different from PS, no centralized server

How is AllReduce Implemented?

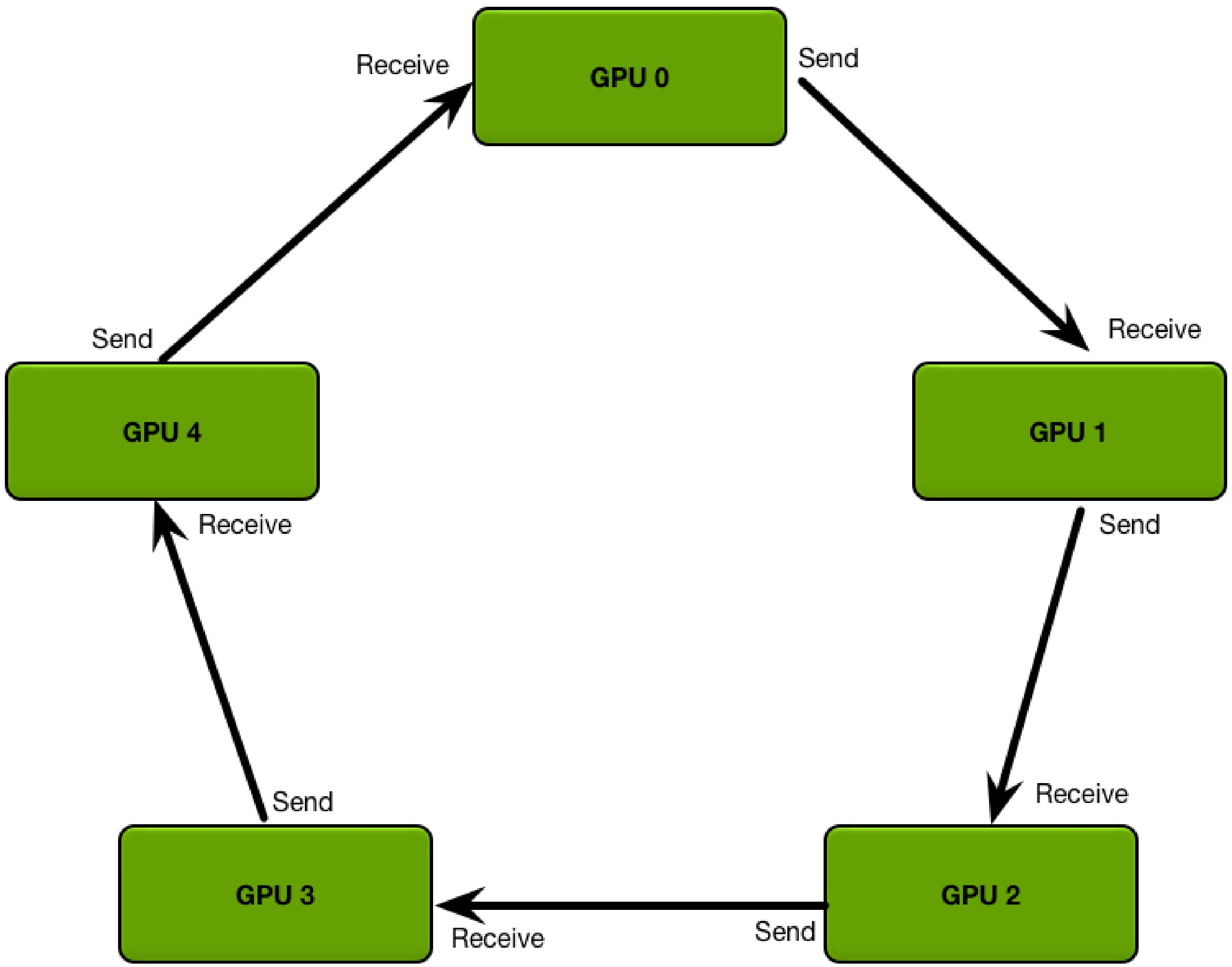


Tree-based



Ring-based

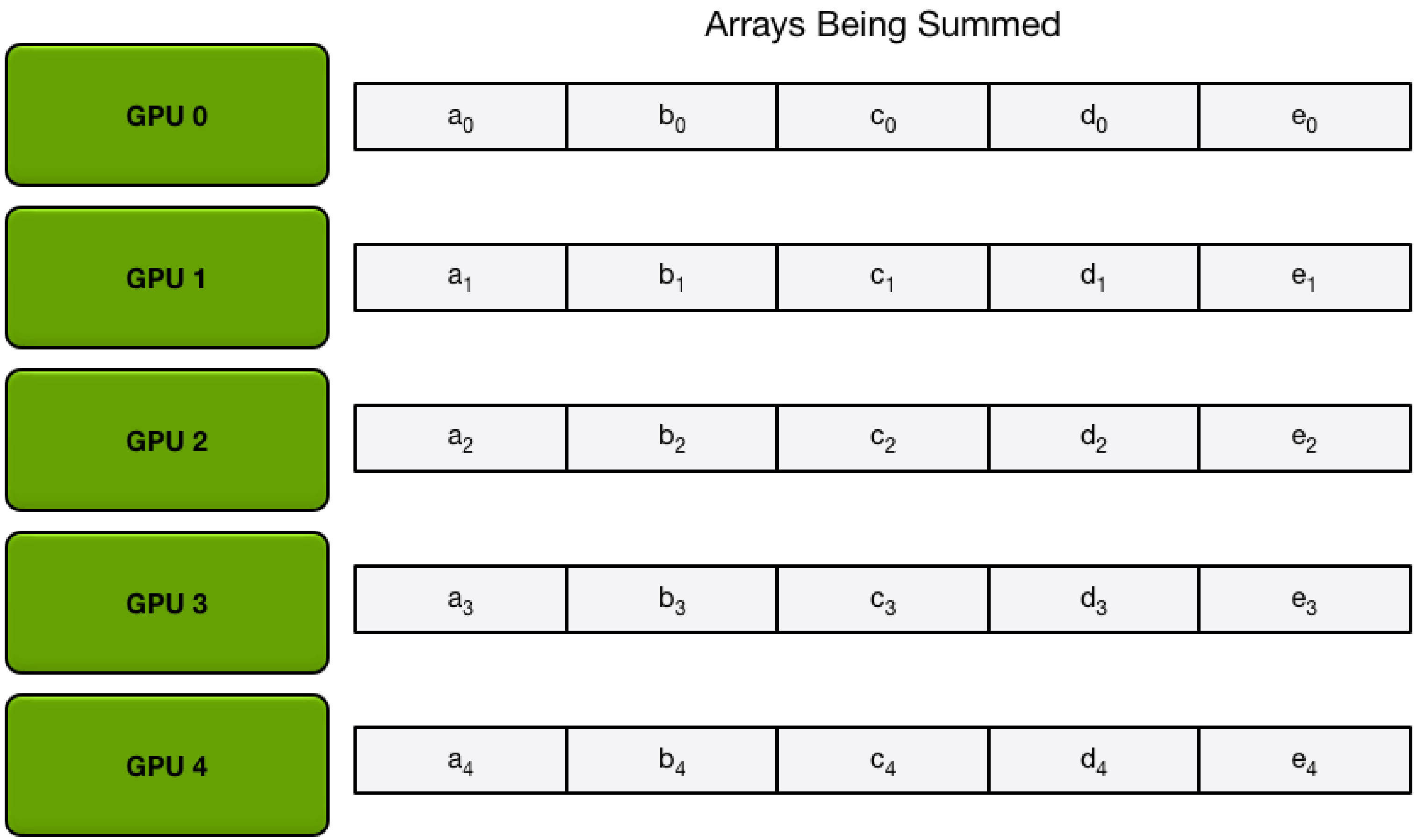
Ring-based AllReduce



GPUs arranged in a logical ring

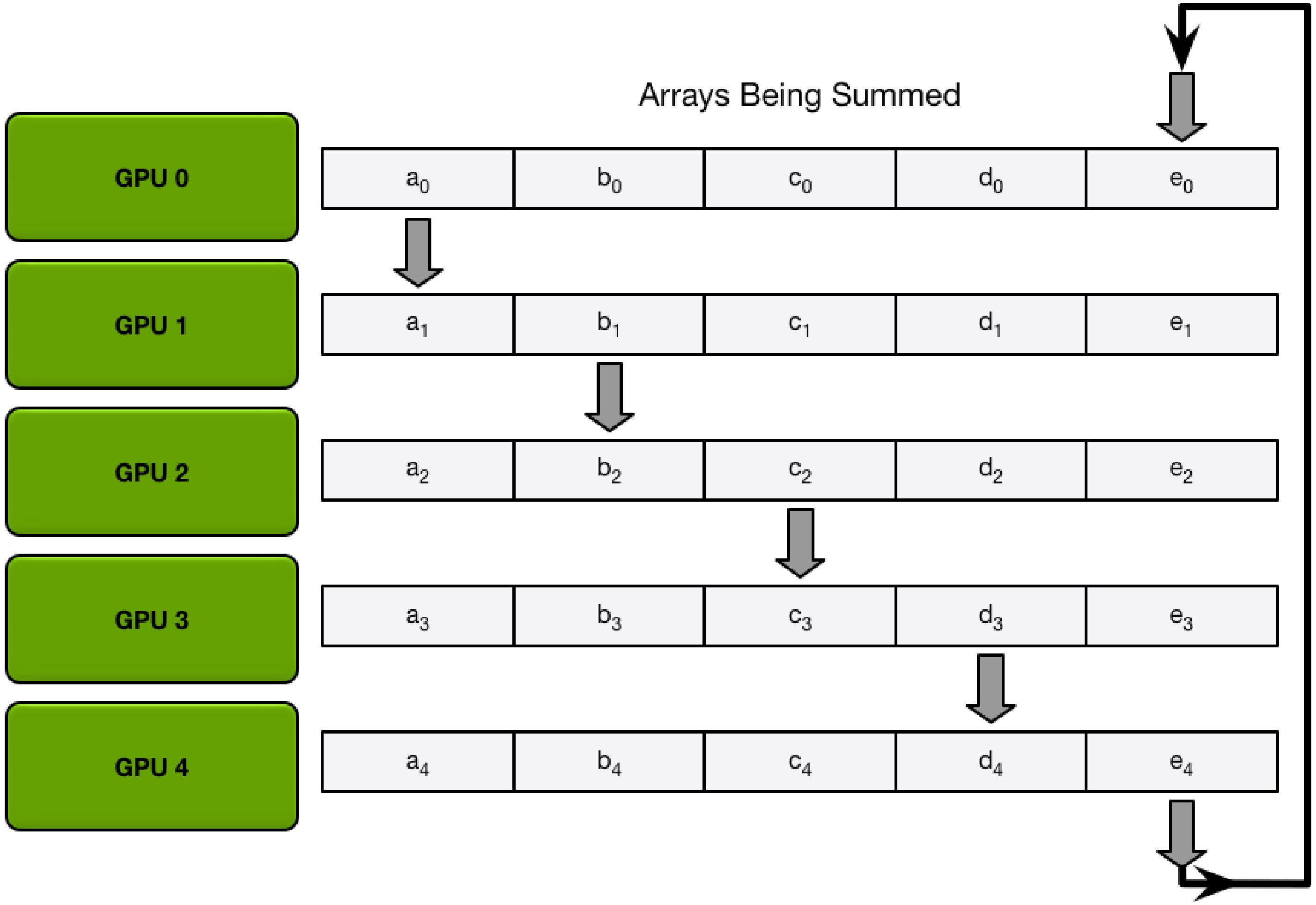
Two phases:
Scatter-reduce
All-gather

Phase 1: Reduce-Scatter



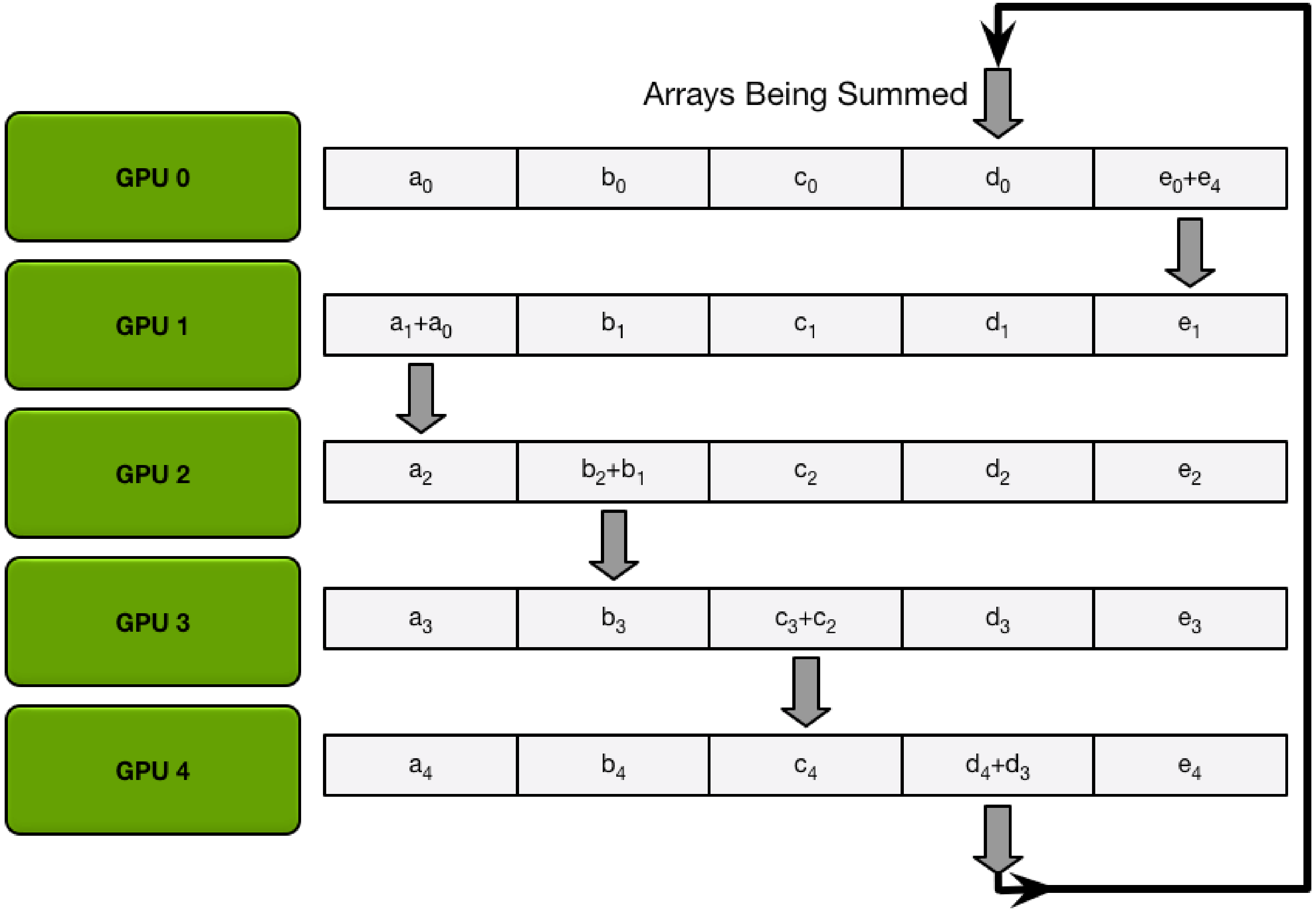
Partitioning of an array into N chunks

Phase 1: Reduce-Scatter



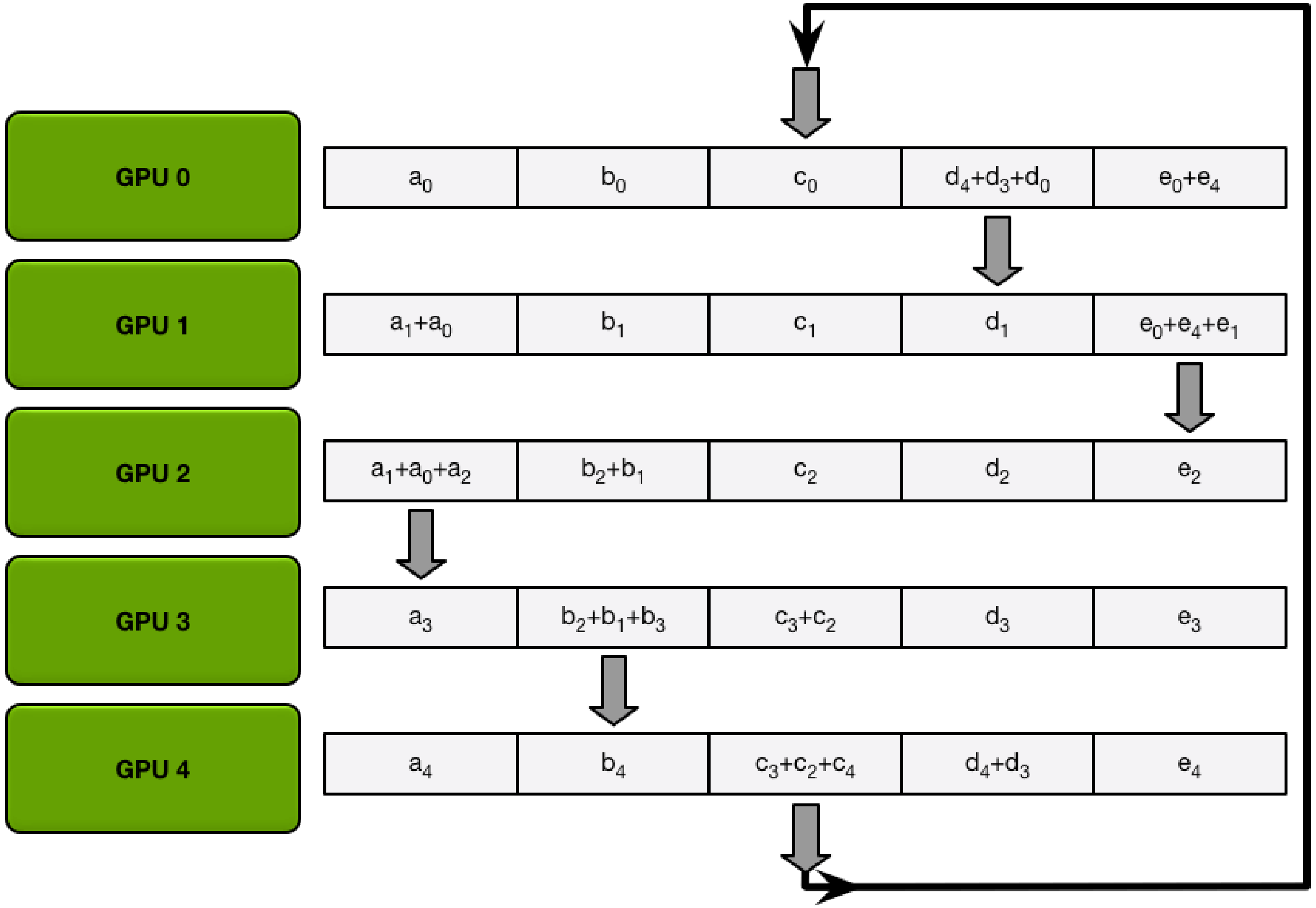
Data transfers in the 1st iteration of scatter-reduce

Phase 1: Reduce-Scatter



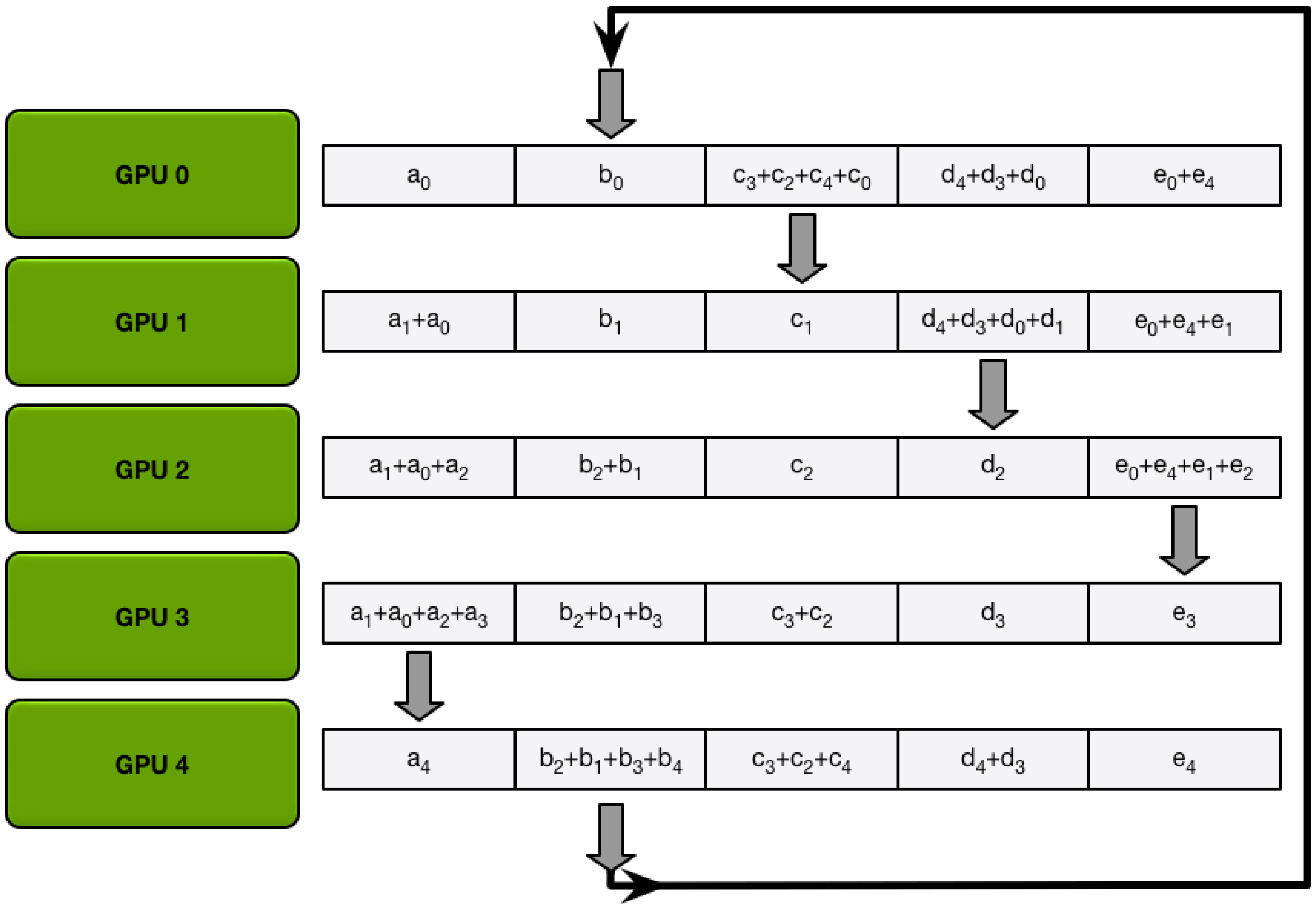
Intermediate sums after the first iteration of scatter-reduce is complete

Phase 1: Reduce-Scatter



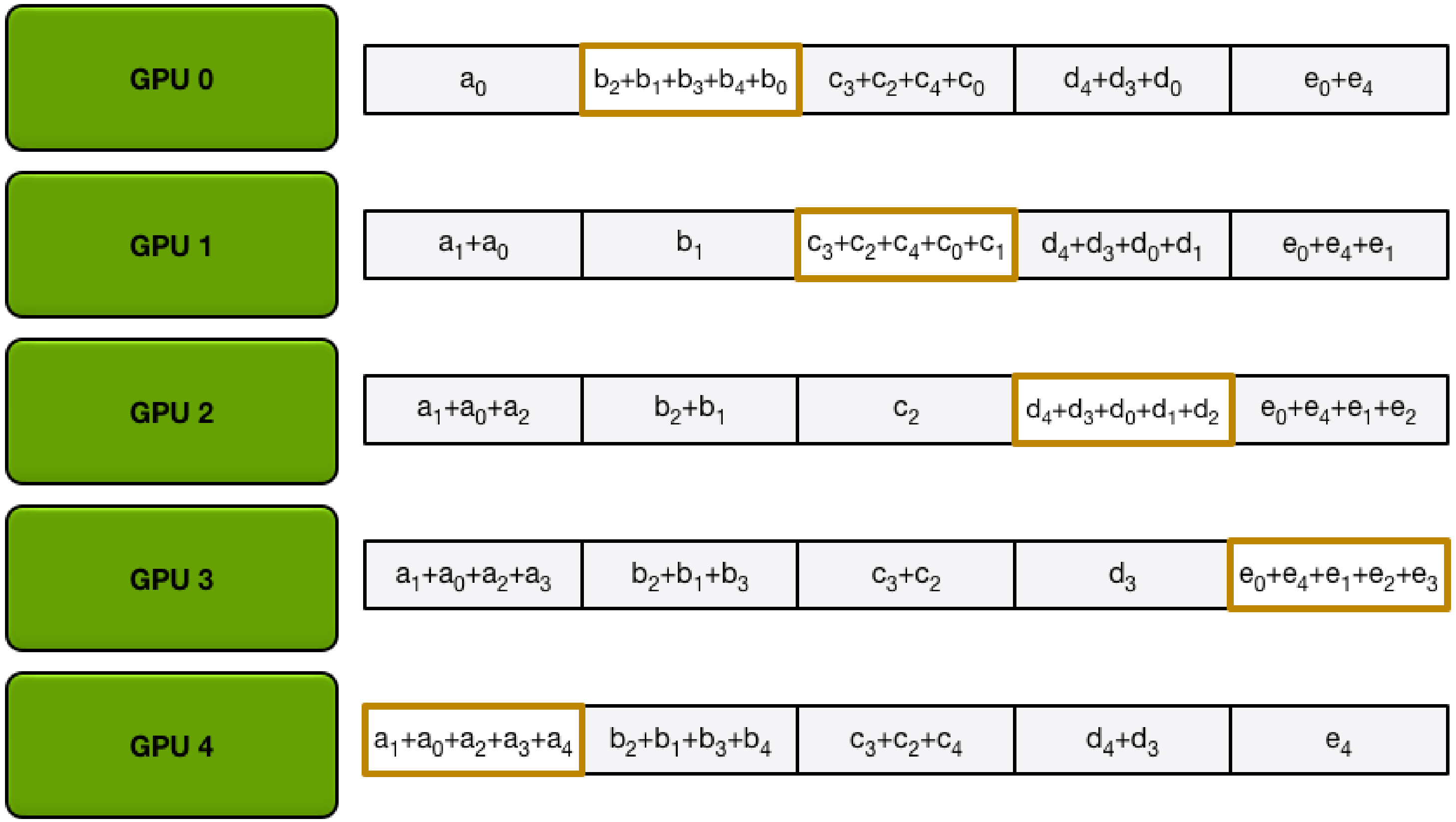
Reduce-scatter data transfer (iteration 3)

Phase 1: Reduce-Scatter



Reduce-scatter data transfer (iteration 4)

Phase 1: Reduce-Scatter



Reduce-scatter data transfer (after iteration 4)

Phase 1: Reduce-Scatter

GPU 0	a_0	$\boxed{b_2+b_1+b_3+b_4+b_0}$	$c_3+c_2+c_4+c_0$	$d_4+d_3+d_0$	e_0+e_4
GPU 1	a_1+a_0	b_1	$\boxed{c_3+c_2+c_4+c_0+c_1}$	$d_4+d_3+d_0+d_1$	$e_0+e_4+e_1$
GPU 2	$a_1+a_0+a_2$	b_2+b_1	c_2	$\boxed{d_4+d_3+d_0+d_1+d_2}$	$e_0+e_4+e_1+e_2$
GPU 3	$a_1+a_0+a_2+a_3$	$b_2+b_1+b_3$	c_3+c_2	d_3	$\boxed{e_0+e_4+e_1+e_2+e_3}$
GPU 4	$\boxed{a_1+a_0+a_2+a_3+a_4}$	$b_2+b_1+b_3+b_4$	$c_3+c_2+c_4$	d_4+d_3	e_4

Question: What is the complexity of reduce-scatter?

Reduce-scatter data transfer (after iteration 4)

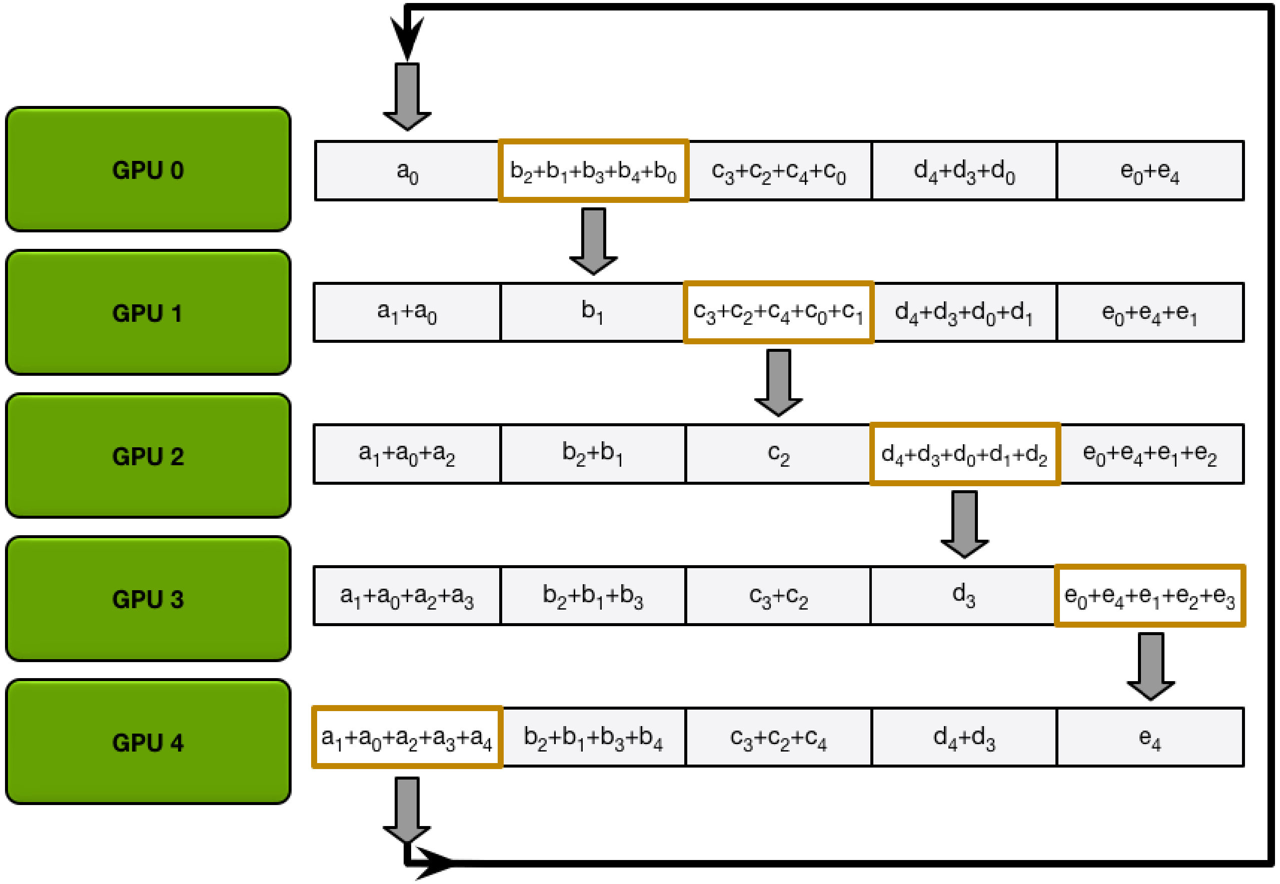
Phase 1: Reduce-Scatter

GPU 0	a_0	$\boxed{b_2+b_1+b_3+b_4+b_0}$	$c_3+c_2+c_4+c_0$	$d_4+d_3+d_0$	e_0+e_4
GPU 1	a_1+a_0	b_1	$\boxed{c_3+c_2+c_4+c_0+c_1}$	$d_4+d_3+d_0+d_1$	$e_0+e_4+e_1$
GPU 2	$a_1+a_0+a_2$	b_2+b_1	c_2	$\boxed{d_4+d_3+d_0+d_1+d_2}$	$e_0+e_4+e_1+e_2$
GPU 3	$a_1+a_0+a_2+a_3$	$b_2+b_1+b_3$	c_3+c_2	d_3	$\boxed{e_0+e_4+e_1+e_2+e_3}$
GPU 4	$\boxed{a_1+a_0+a_2+a_3+a_4}$	$b_2+b_1+b_3+b_4$	$c_3+c_2+c_4$	d_4+d_3	e_4

$O(N/P * (P - 1))$

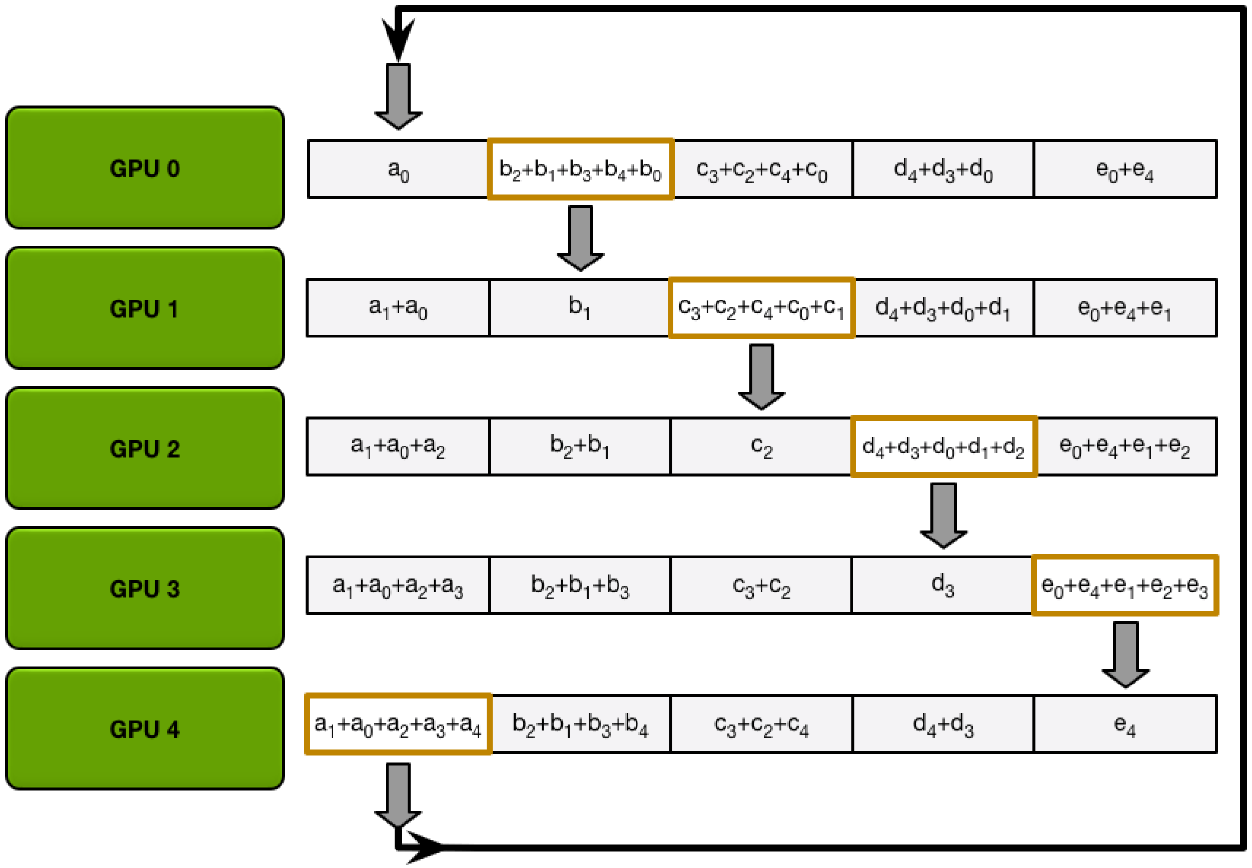
Reduce-scatter data transfer (iteration 4)

Phase 2: Allgather



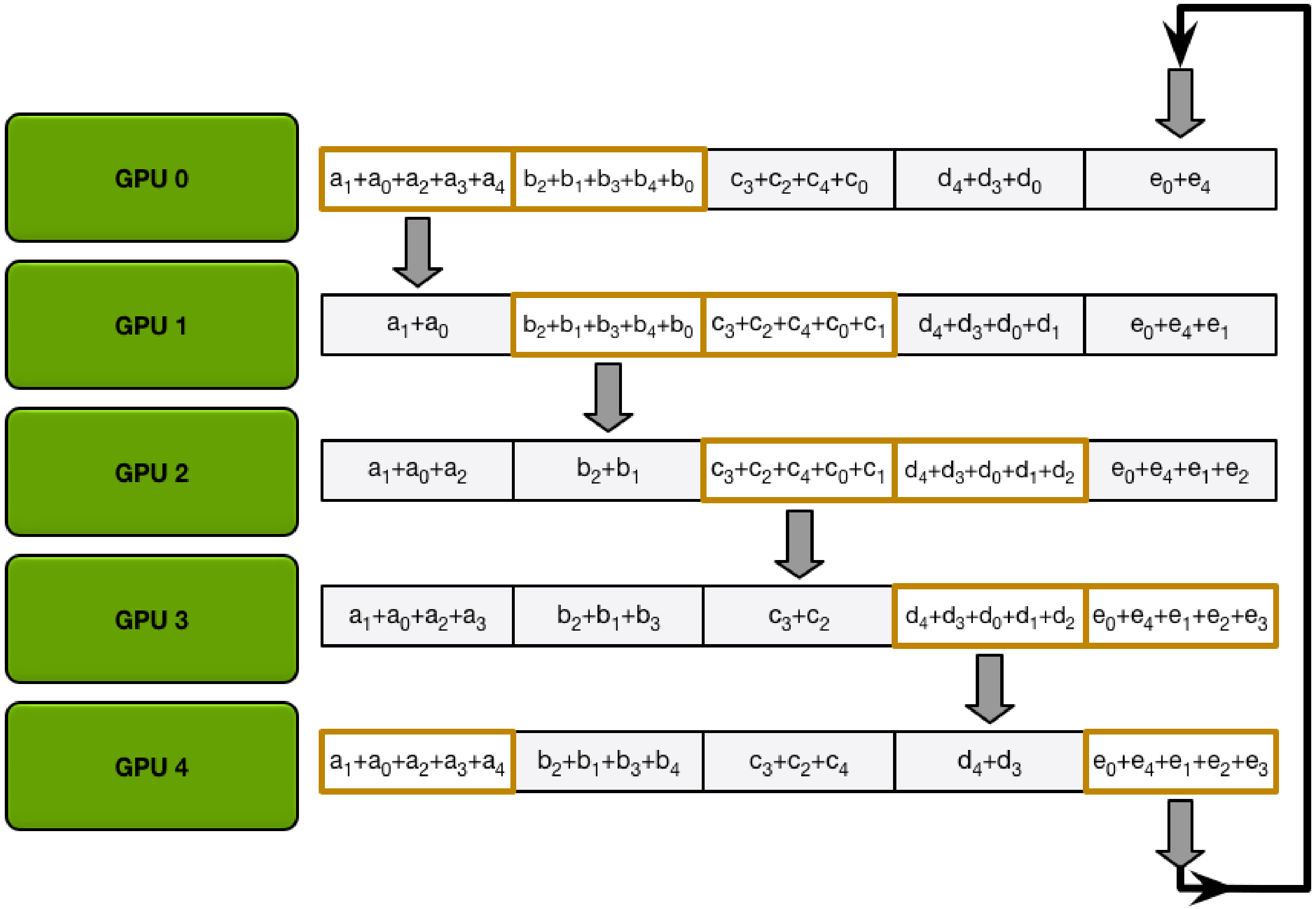
All processes can obtain the complete array by sharing the partial results from them. This can be done by circulating again without reduction operations.

Phase 2: Allgather



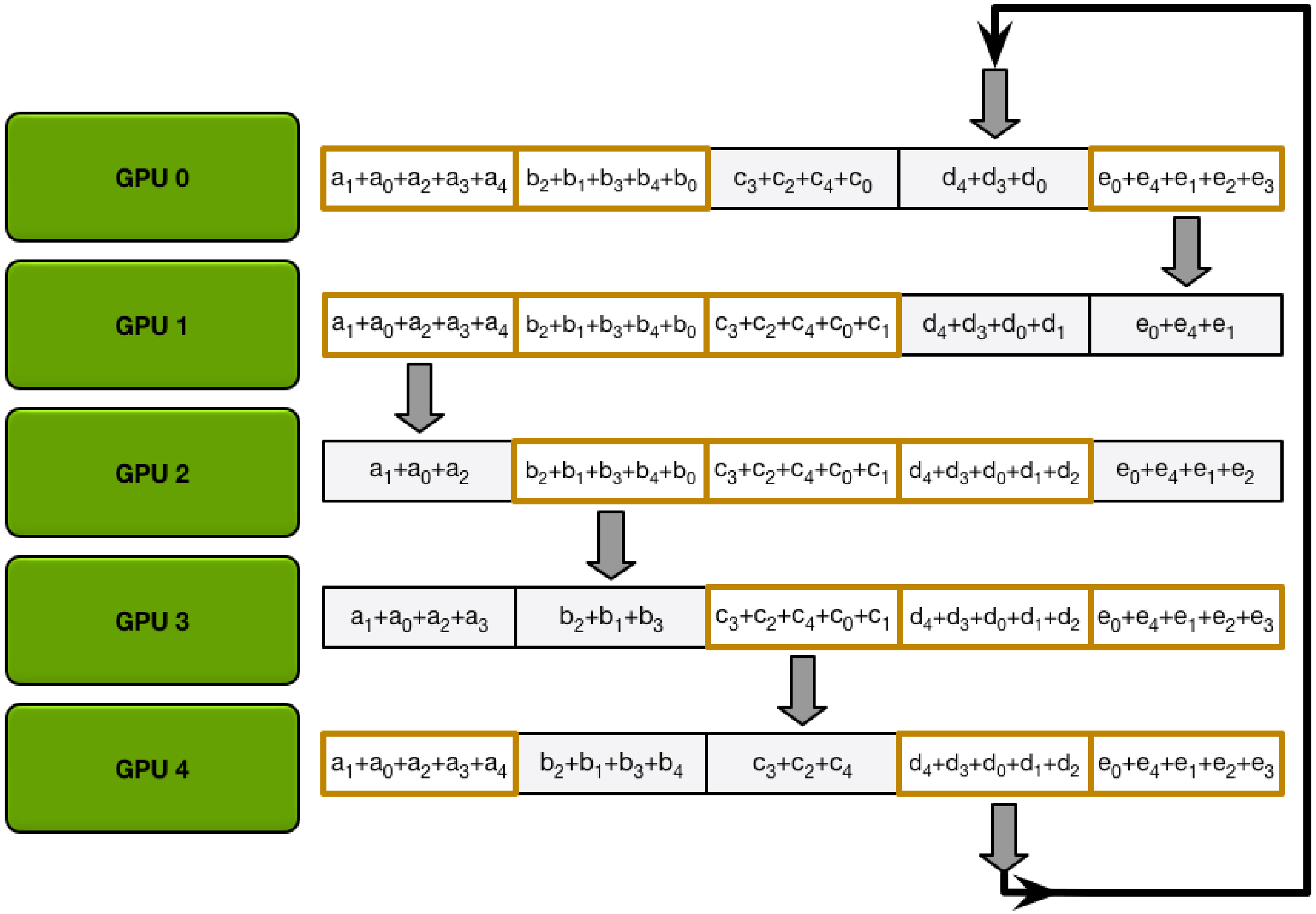
Allgather data transfer (iteration 1)

Phase 2: Allgather



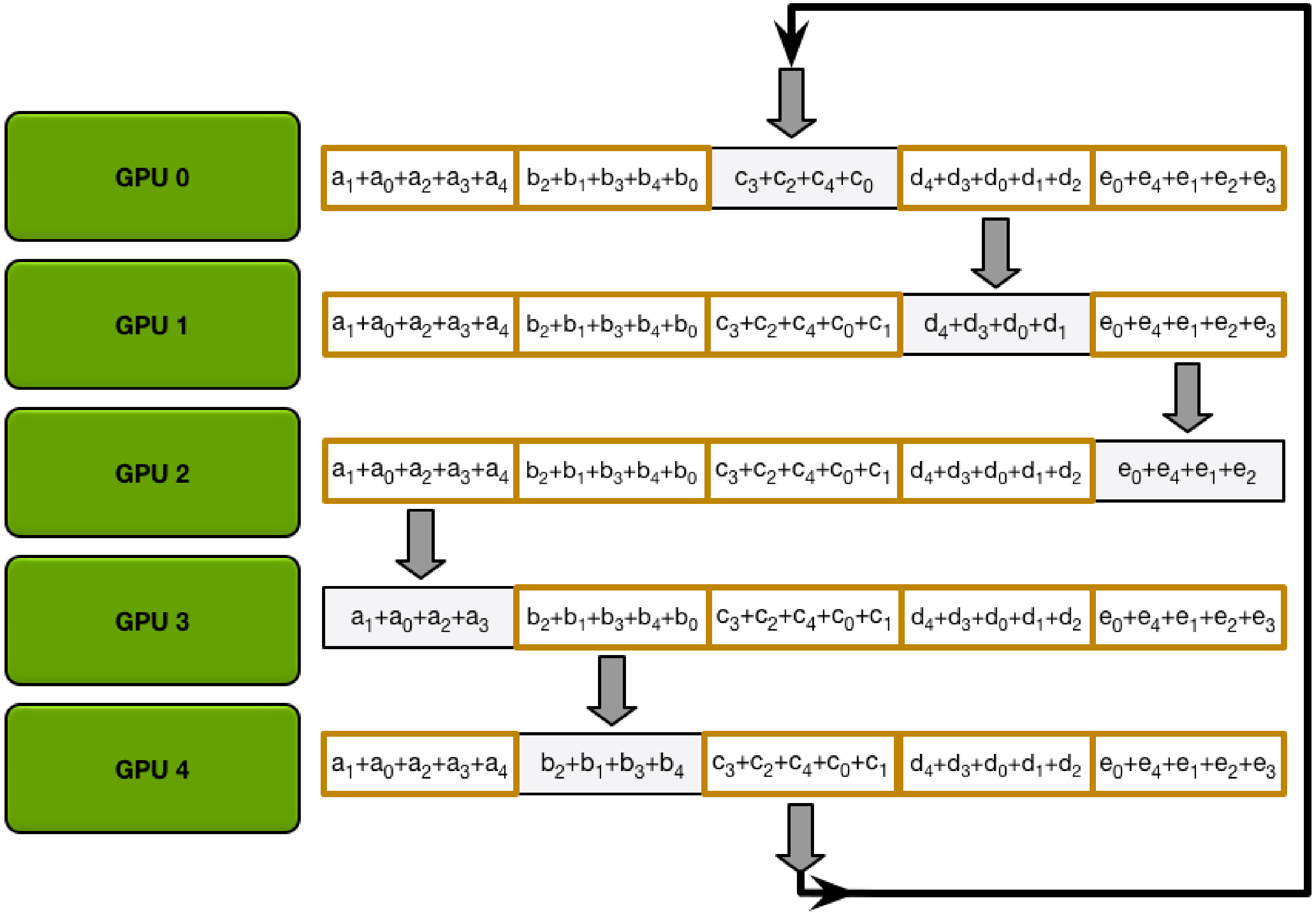
Allgather data transfer (iteration 2)

Phase 2: Allgather



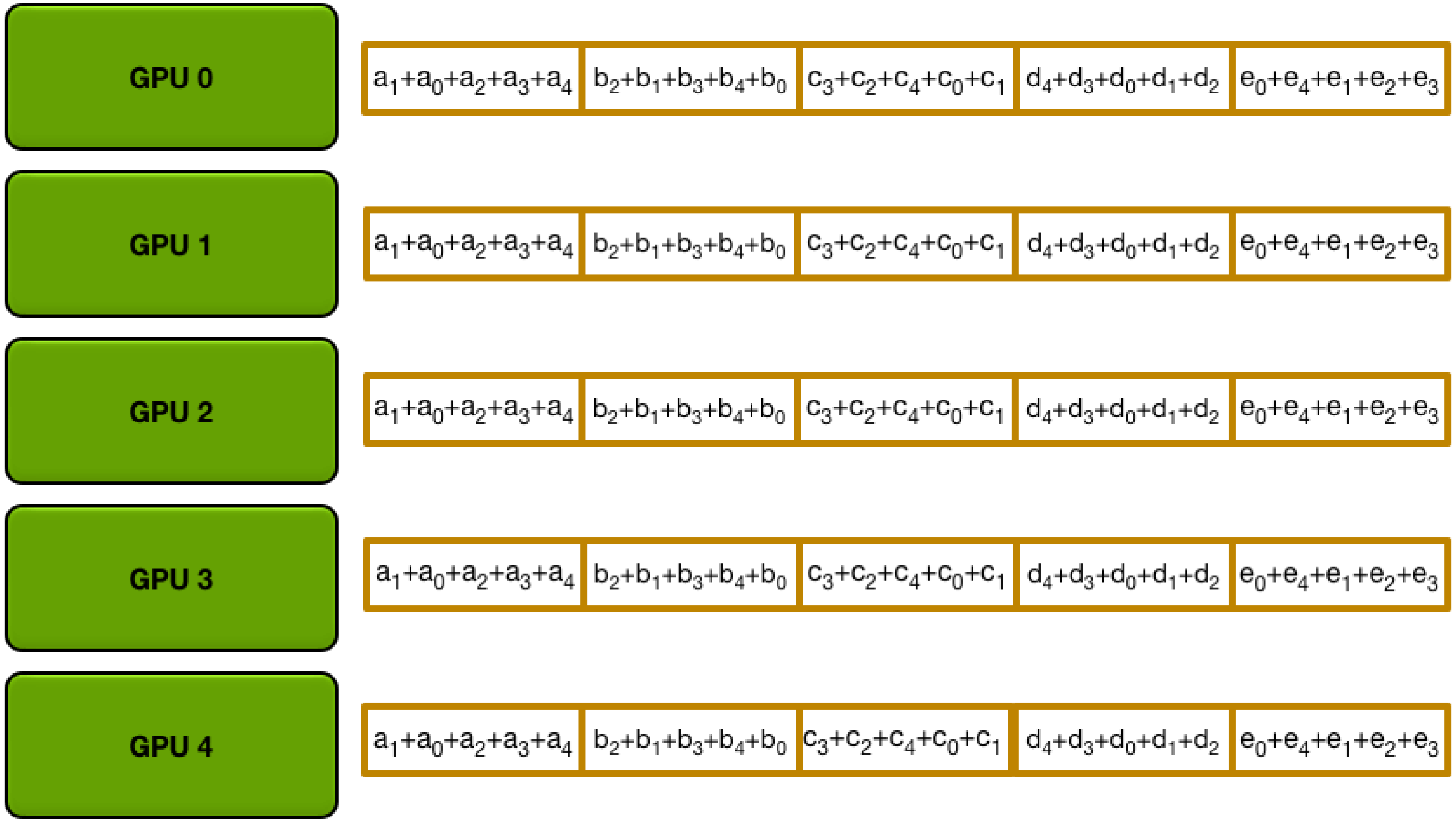
Allgather data transfer (iteration 3)

Phase 2: Allgather



Allgather data transfer (iteration 4)

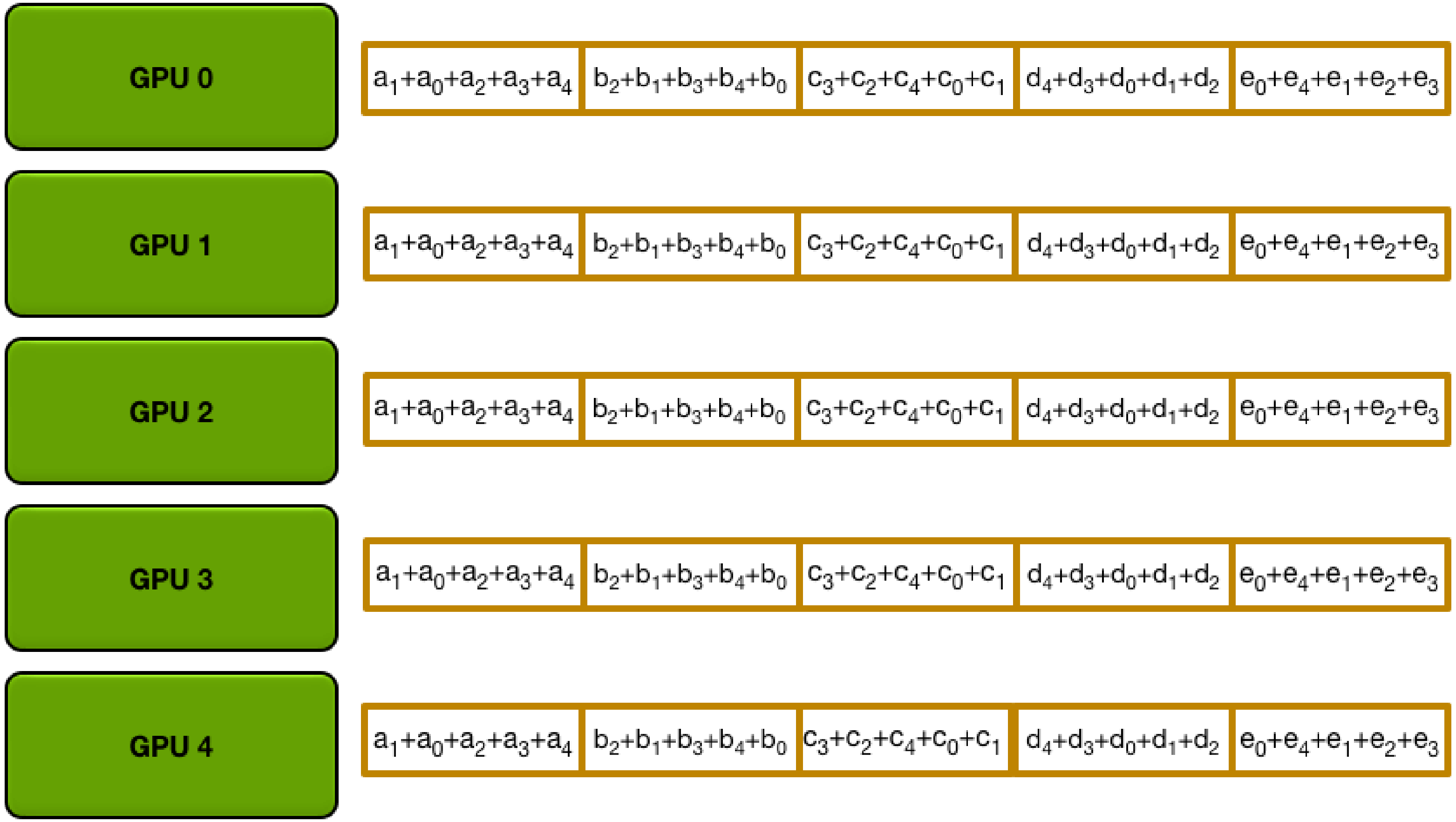
Phase 2: Allgather



Question: What is the time complexity of all-gather?

Final state after allgather transfer

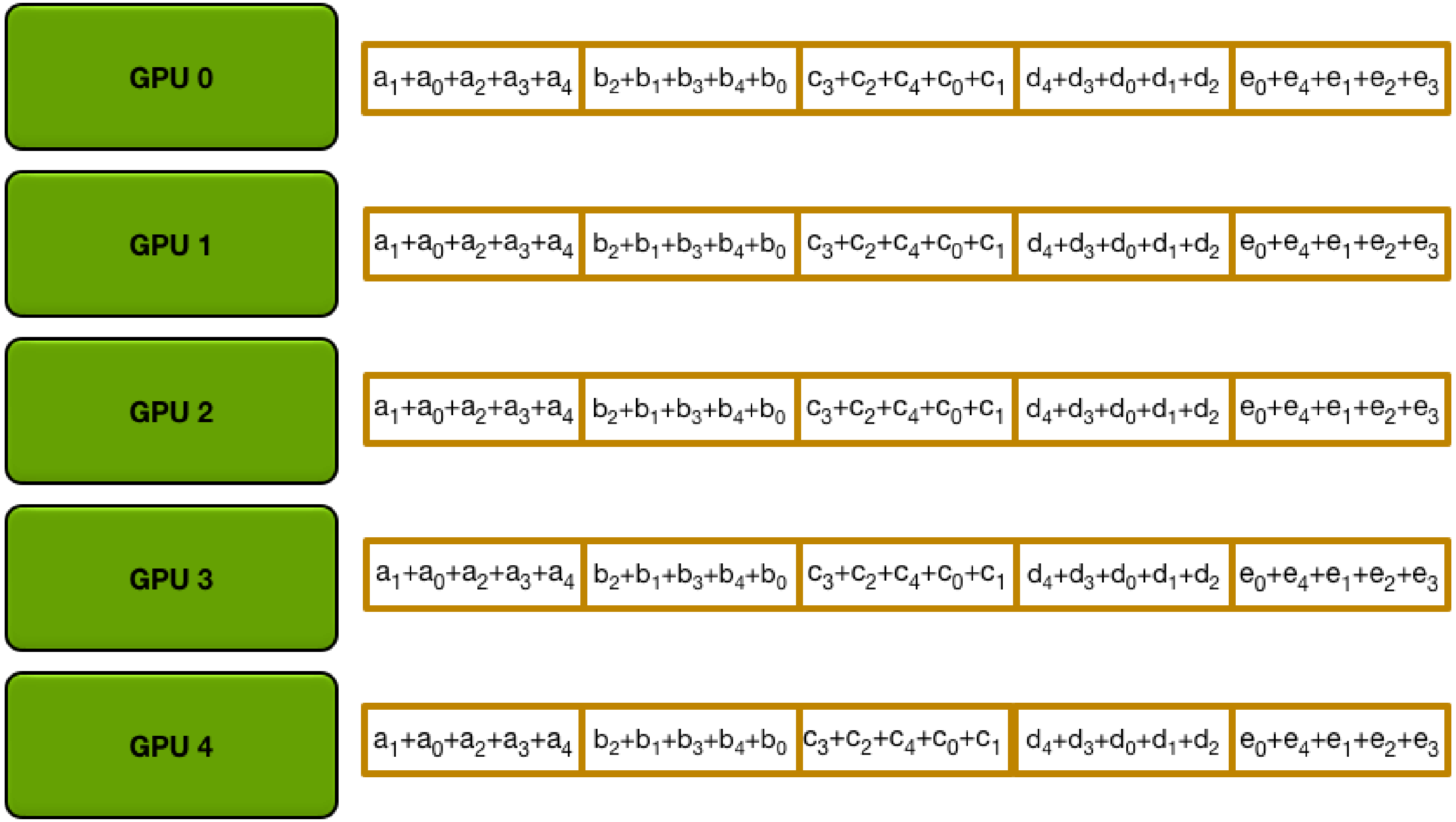
Phase 2: Allgather



O(N/P * (P -1))

Final state after allgather transfer

Phase 2: Allgather

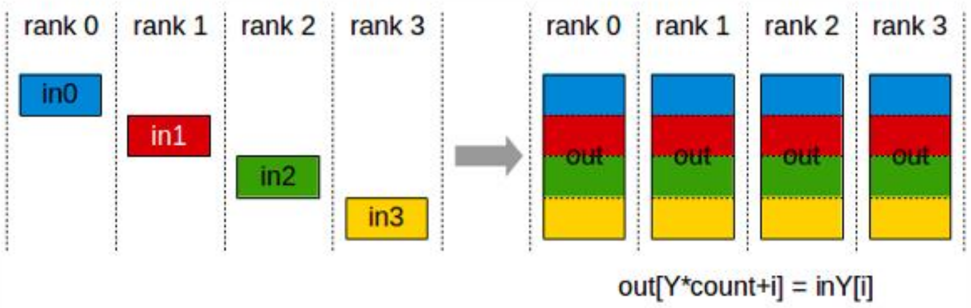


First phase: N/P * (P – 1)
Second phase: N/P * (P -1)
Total: 2N (P – 1) / P, largely independent of P

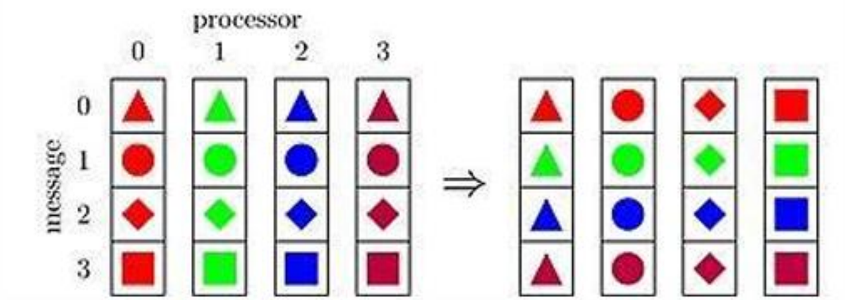
Final state after allgather transfer

Terminologies: Communication Collective

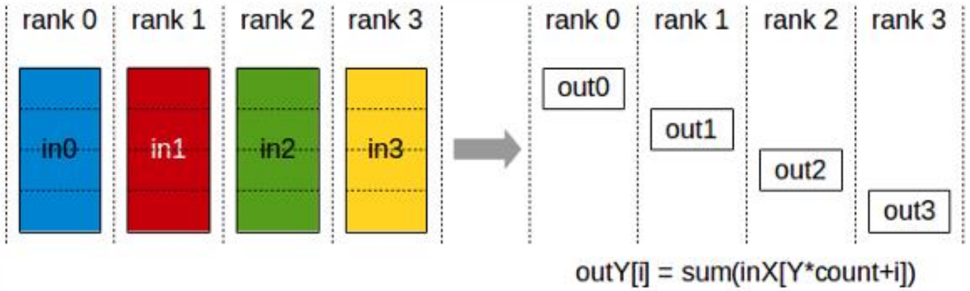
all-gather



all-to-all



Reduce-scatter



Figures from NCCL documentation

Some Transformations

Reduce-scatter

$$(p-1)\alpha + \frac{p-1}{p} n(\beta+\gamma)$$

Scatter

$$\log(p)\alpha + \frac{p-1}{p} n\beta$$

Gather

$$\log(p)\alpha + \frac{p-1}{p} n\beta$$

Allgather

$$(p-1)\alpha + \frac{p-1}{p} n\beta$$

Reduce(-to-one)

Allreduce

Broadcast

Some Transformations

Reduce-scatter

$(p-1)\alpha + \frac{p-1}{p} n(\beta+\gamma)$

Scatter

$\log(p)\alpha + \frac{p-1}{p} n\beta$

Gather

$\log(p)\alpha + \frac{p-1}{p} n\beta$

Allgather

$(p-1)\alpha + \frac{p-1}{p} n\beta$

Reduce(-to-one)

$(p-1+\log(p))\alpha + \frac{p-1}{p} n(2\beta+\gamma)$

Allreduce

$2(p-1)\alpha + \frac{p-1}{p} n(2\beta+\gamma)$

Broadcast

Some Transformations

Reduce-scatter

$$(p-1)\alpha+\frac{p-1}{p}n(\beta+\gamma)$$

Scatter

$$\log(p)\alpha+\frac{p-1}{p}n\beta$$

Gather

$$\log(p)\alpha+\frac{p-1}{p}n\beta$$

Allgather

$$(p-1)\alpha+\frac{p-1}{p}n\beta$$

Reduce(-to-one)

$$(p-1+\log(p))\alpha+\frac{p-1}{p}n(2\beta+\gamma)$$

Allreduce

$$2(p-1)\alpha+\frac{p-1}{p}n(2\beta+\gamma)$$

Broadcast

$$(\log(p)+p-1)\alpha+2\frac{p-1}{p}n\beta$$

Some Transformations

Reduce-scatter

$$(p-1)\alpha+\frac{p-1}{p}n(\beta+\gamma)$$

Scatter

$$\log(p)\alpha+\frac{p-1}{p}n\beta$$

Gather

$$\log(p)\alpha+\frac{p-1}{p}n\beta$$

Allgather

$$(p-1)\alpha+\frac{p-1}{p}n\beta$$

Reduce(-to-one)

$$(p-1+\log(p))\alpha+\frac{p-1}{p}n(2\beta+\gamma)$$

Allreduce

Broadcast

Communication Basics

- Collectives are implemented using many P2P
  - Collectives are much more expensive than P2P
- Collectives are highly optimized throughout the past 20 years
  - Look for “X”CCL libraries
  - NCCL, RCCL, MSCCL,…
- Collectives are not fault-tolerant
- Collectives can be implemented differently
  - Ring
  - Minimum Spanning Tree (MST)

NCCL: Nvidia's Efficient Multi-GPU Collective

- Use unified GPU direct memory accessing
- Each GPU launches a working kernel, cooperate with each other to do ring-based reduction
- A common backend for popular DL frameworks such as PyTorch

Today

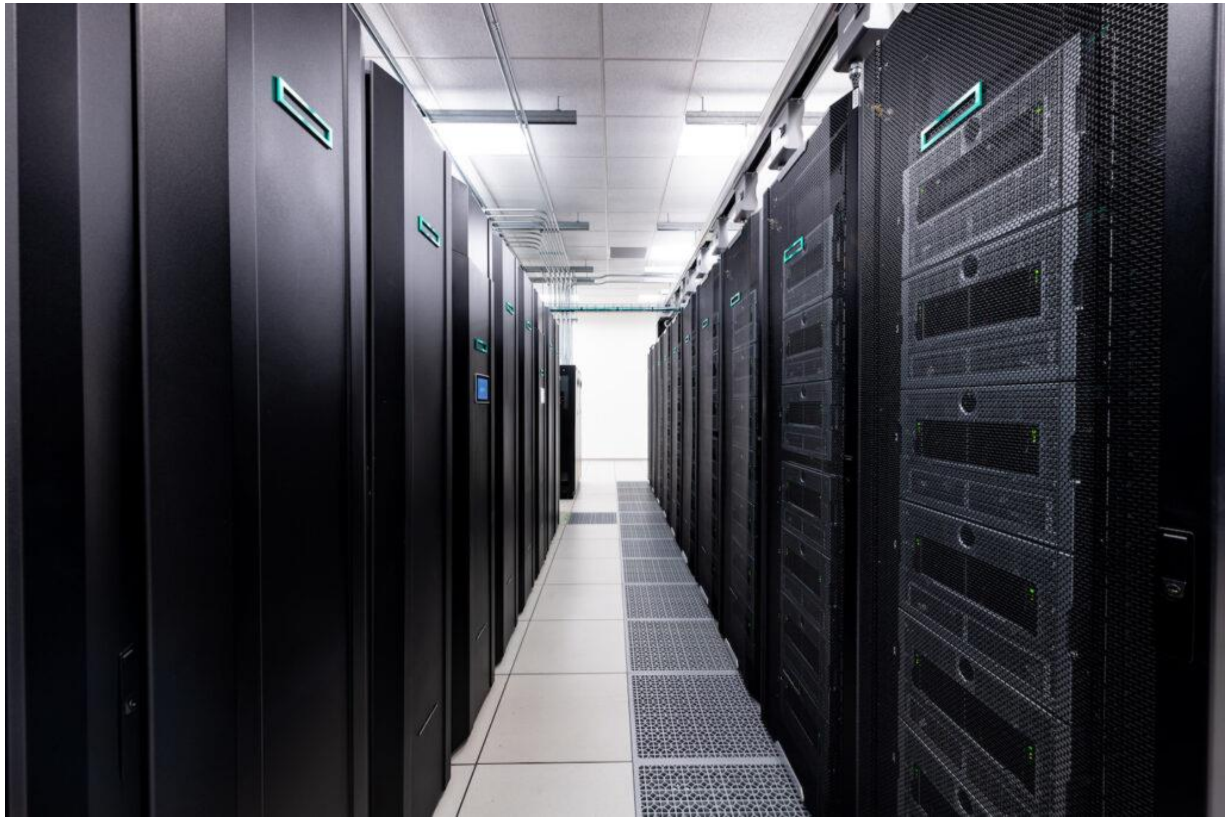
- **Distributed Training Preliminary**
  - **Problem**
  - **Challenge**
  - **History**
  - **Parameter Server based DP**

- **Data Parallelism**
  - **Allreduce-based DP**
  - **Communication terminologies**
- **Hardware**

Hardware: Delta

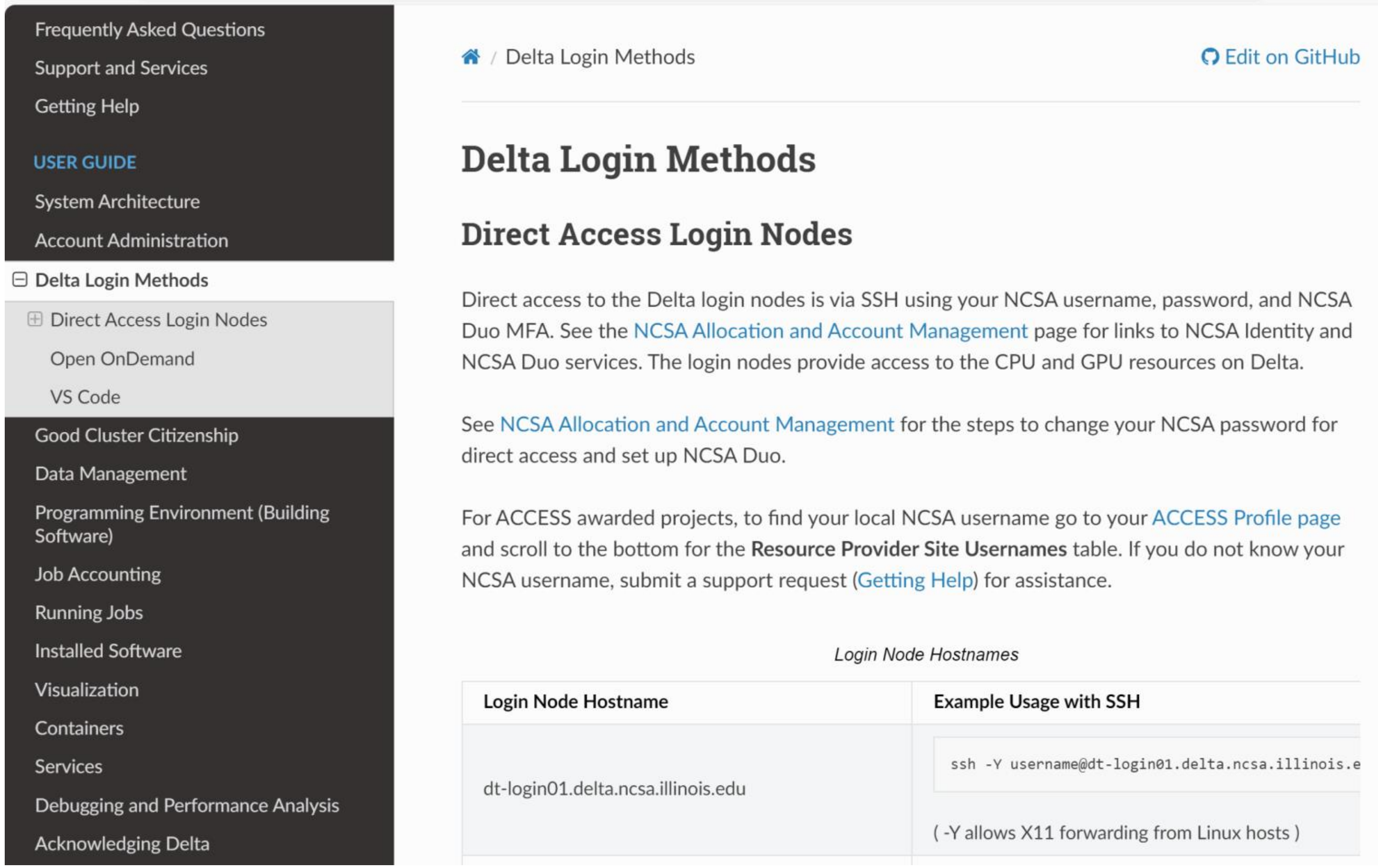
- Home page: https://www.ncsa.illinois.edu/research/project-highlights/delta/

- 100 quad A100 GPU node, each with 4 A100
- 100 quad A40 GPU node, each with 4 A40
- 5 8-way A100 GPU, each with 8 A100
- 1 MI100 node, 8 MI100



Delta Onboarding

- https://docs.ncsa.illinois.edu/systems/delta/en/latest/user_guide/accessing.html



Frequently Asked Questions
Support and Services
Getting Help
USER GUIDE
System Architecture
Account Administration
Delta Login Methods
Direct Access Login Nodes
Open OnDemand
VS Code
Good Cluster Citizenship
Data Management
Programming Environment (Building Software)
Job Accounting
Running Jobs
Installed Software
Visualization
Containers
Services
Debugging and Performance Analysis
Acknowledging Delta

/ Delta Login Methods

Edit on GitHub

Delta Login Methods

Direct Access Login Nodes

Direct access to the Delta login nodes is via SSH using your NCSA username, password, and NCSA Duo MFA. See the NCSA Allocation and Account Management page for links to NCSA Identity and NCSA Duo services. The login nodes provide access to the CPU and GPU resources on Delta.

See NCSA Allocation and Account Management for the steps to change your NCSA password for direct access and set up NCSA Duo.

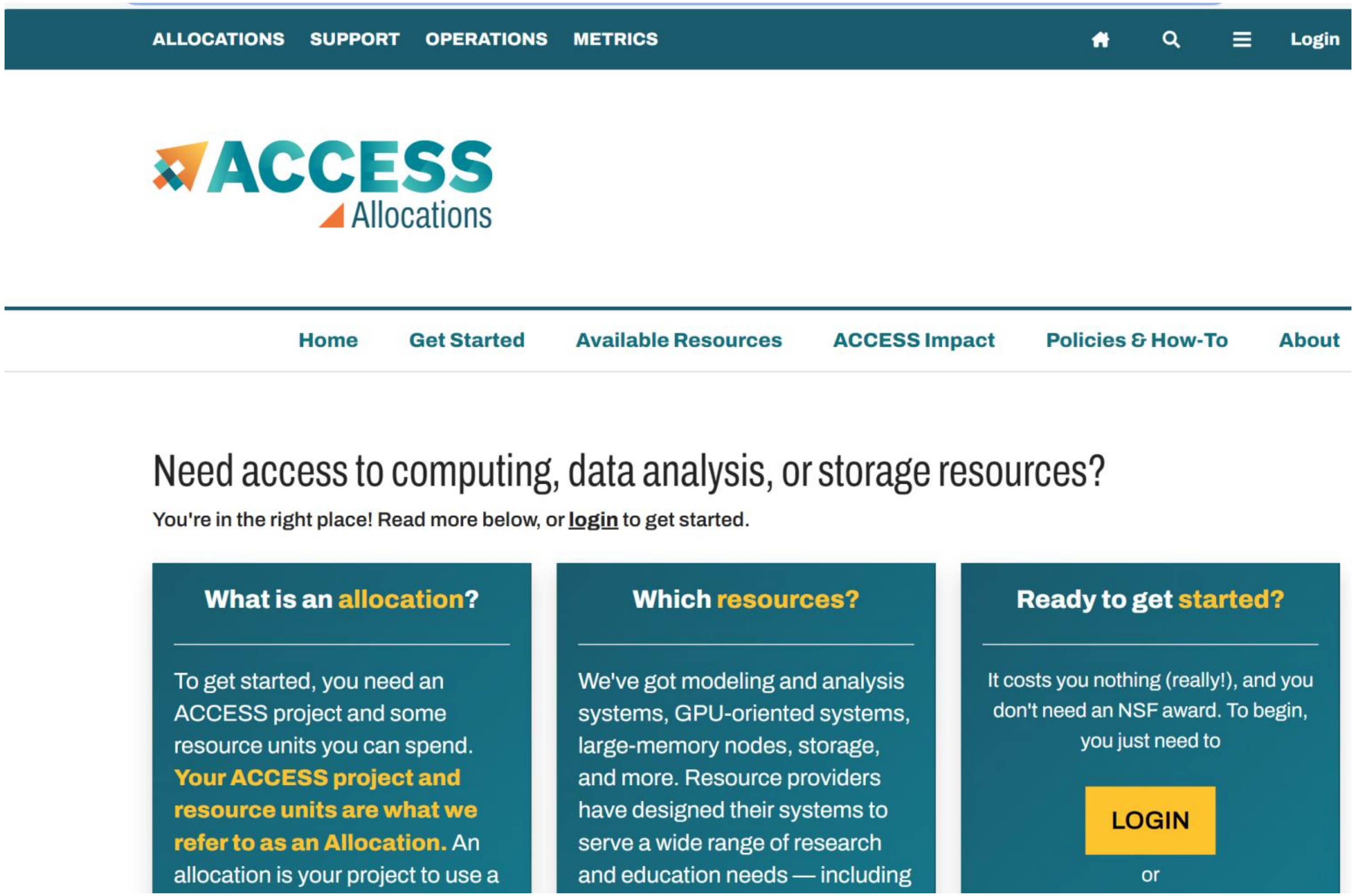
For ACCESS awarded projects, to find your local NCSA username go to your ACCESS Profile page and scroll to the bottom for the **Resource Provider Site Usernames** table. If you do not know your NCSA username, submit a support request (Getting Help) for assistance.

Login Node Hostnames

Login Node Hostname	Example Usage with SSH
dt-login01.delta.ncsa.illinois.edu	`ssh -Y username@dt-login01.delta.ncsa.illinois.e` (-Y allows X11 forwarding from Linux hosts)

Step 1: Create ACCESS ID

- Register an ACCESS id at: https://access-ci.org/ (top right-hand corner)
- After you register, send the instructor your ACCESS id. The instructor will add you to access to his GPU allocation.



Step 2: Get Started

- Delta uses Slurm to manage jobs/GPUs
- Please watch this tutorial video: Getting Started on NCSA's Delta Supercomputer.
- After that, you may want to check Delta User Documentation — UIUC NCSA Delta User Guide (illinois.edu).
- Please learn how to use slurm to get GPUs: Slurm Workload Manager - Quick Start User Guide (schedmd.com).

Step 3: SSH Login

- You shall use ssh to login to the node: [Delta Login Methods — UIUC NCSA Delta User Guide (illinois.edu)]().

- For instance, you can use commands such as "srun -A bcjw-delta-gpu --time=00:30:00 --nodes=1 --ntasks-per-node=16 --partition=gpuA100x4,gpuA40x4 --gpus=1 --mem=32g --pty /bin/bash"

- Maintaining Persistent Sessions: tmux

Delta Login Methods

Direct Access Login Nodes

Direct access to the Delta login nodes is via SSH using your NCSA username, password, and NCSA Duo MFA. See the NCSA Allocation and Account Management page for links to NCSA Identity and NCSA Duo services. The login nodes provide access to the CPU and GPU resources on Delta.

See NCSA Allocation and Account Management for the steps to change your NCSA password for direct access and set up NCSA Duo.

For ACCESS awarded projects, to find your local NCSA username go to your ACCESS Profile page and scroll to the bottom for the **Resource Provider Site Usernames** table. If you do not know your NCSA username, submit a support request (Getting Help) for assistance.

Login Node Hostnames

Login Node Hostname	Example Usage with SSH
dt-login01.delta.ncsa.illinois.edu	`ssh -Y username@dt-login01.delta.ncsa.illinois.e` (-Y allows X11 forwarding from Linux hosts)
dt-login02.delta.ncsa.illinois.edu	`ssh -l username dt-login02.delta.ncsa.illinois.e` (-l username alt. syntax for `user@host`)
login.delta.ncsa.illinois.edu (round robin DNS name for the set of login nodes)	`ssh username@login.delta.ncsa.illinois.edu`

Additional Info

- It is the instructor’s own research allocation, and it has a limit. So please be mindful when using GPU resources.
  - Avoid allocating too many GPUs at once
  - Turn off the job when you are not using the GPUs

- The allocation has 500 GB of storage in total (shared by the class and other students in the instructor’s lab)
  - Please avoid downloading large data files and super large model checkpoints, e.g., one llama7b checkpoint consumes roughly 14GB.

Course Project (Reproducibility Challenge)

4-credit undergraduate students, 3-credit graduate students
Some suggestions below, but it can be any machine learning system related papers that you are interested in

GPTCache: An Open-Source Semantic Cache for LLM Applications Enabling Faster Answers and Cost Savings	https://github.com/zilliztech/GPTCache
RouteLLM: Learning to Route LLMs with Preference Data	https://github.com/lm-sys/RouteLLM
LLM-QAT: Data-Free Quantization Aware Training for Large Language Models	https://github.com/facebookresearch/LLM-QAT
Speculative decoding in vLLM	https://docs.vllm.ai/en/v0.5.5/models/spec_decode.html
REST: Retrieval-Based Speculative Decoding	https://github.com/FasterDecoding/REST
MemGPT: Towards LLMs as Operating Systems	https://github.com/letta-ai/letta
…	…

Course Project (Open-End Research Project)

4-credit graduate students

- Benchmark and analyze important DL workloads to understand their performance gap and identify important angles to optimize their performance.
- Apply and evaluate how existing solutions work in the context of emerging AI/DL workloads.
- Design and implement new algorithms that are both theoretically and practically efficient.
- Design and implement system optimizations, e.g., parallelism, cache-locality, IO-efficiency, to improve the compute/memory/communication efficiency of AI/DL workloads.
- Offer customized optimization for critical DL workloads where latency is extremely tight.
- Build library/tool/framework to improve the efficiency of a class of problems.
- Integrate important optimizations into existing frameworks (e.g., DeepSpeed), providing fast and agile inference.
- Combine system optimization with modeling optimizations.
- Combine and leverage hardware resources (e.g., GPU/CPU, on-device memory/DRAM/NVMe/SSD) in a principled way.

Questions?