CASE SCHOOL OF ENGINEERING
CASE WESTERN RESERVE UNIVERSITY

CSDS 451: Designing High Performant Systems for AI

Lecture 23

11/20/2025

Sanmukh Kuppannagari

sanmukh.kuppannagari@case.edu

https://sanmukh.research.st/

Case Western Reserve University

Outline

- Data Parallel Distributed Training
- Model Parallel Distributed Training

Announcements

- WA 4 was out
  - Due December 4

- Finals next Tuesday
  - Transformer Models
  - Clusters
    - How to program clusters with MPI
    - Communication collectives with complexity
    - Distributed training – very simple question to check knowledge, no application/analysis questions

Midterm Logistics

- 60 minute exam, on paper
- 15~20 Fill in the blank/True-False questions
  - With space for explanation
  - 1 point for answer
  - 2 for explanation - wrong answer with a reasonable explanation will receive partial credits
- 2 big questions

Midterm Logistics

- Open Notes
  - You can use slides
  - I suggest you create a cheat sheet: You won't have enough time to search for concepts.
- No questions during exam (Gets too distractive)
  - If a questions is ambiguous, write your assumption and solve
  - The assumption will be taken into consideration when grading
- You should not need a calculator
  - You can use your phone if you do need it

Midterm Logistics

- Exam may be lengthy (intentionally)
  - If you know the concepts well, you should be able to solve in time
  - Searching your notes or slides will incur additional time

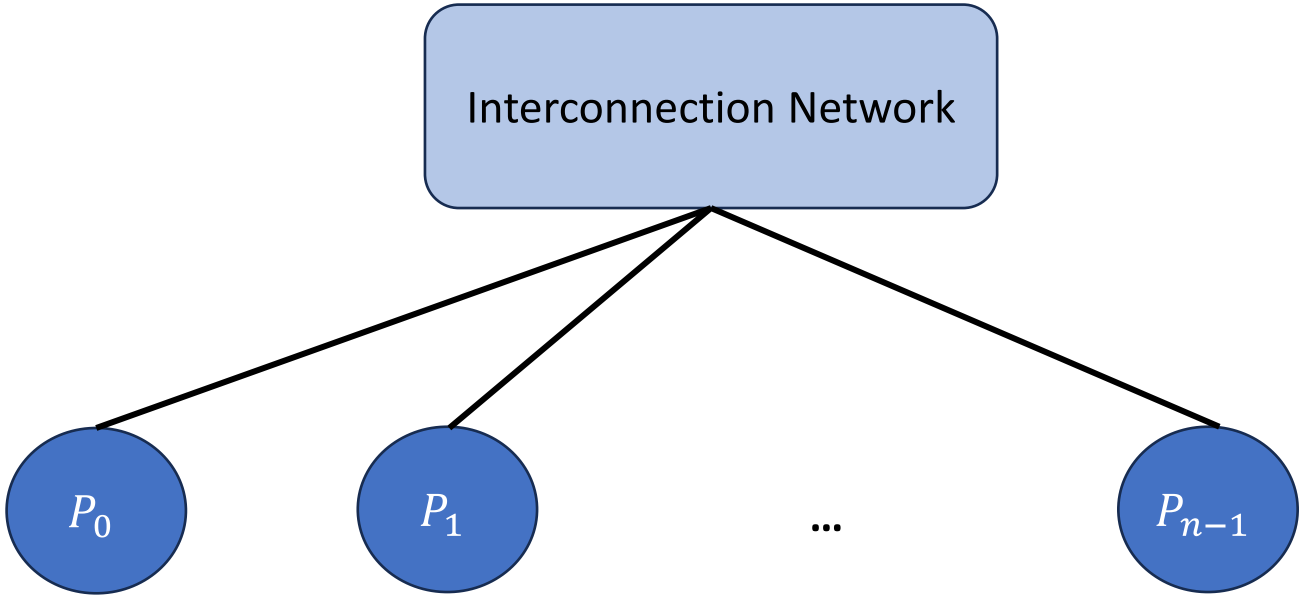
- Plan the questions you want to solve first accordingly
  - Suggestion: Get the big questions out of the way.
  - Don’t get stuck on a question. If you can’t solve go ahead and come back
  - If you don’t know an answer, write your explanation. We will consider that while grading

Outline

- Data Parallel Distributed Training
- Model Parallel Distributed Training

Cluster of Accelerators

- N processors (memory + accelerator)
  - Local compute
  - Local memory
- Connected using an Interconnection Network
- Communication through Message Passing



Anatomy of an MPI Program

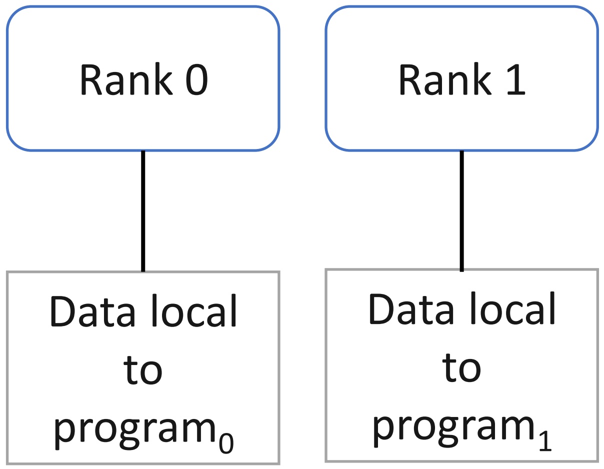
- MPI_Init(…)
- MPI_Comm_rank(MPI_COMM_WORLD, &my_rank)
- MPI_Comm_size(MPI_COMM_WORLD, &num_procs);

- Do rank specific work

Inter-Process Communication

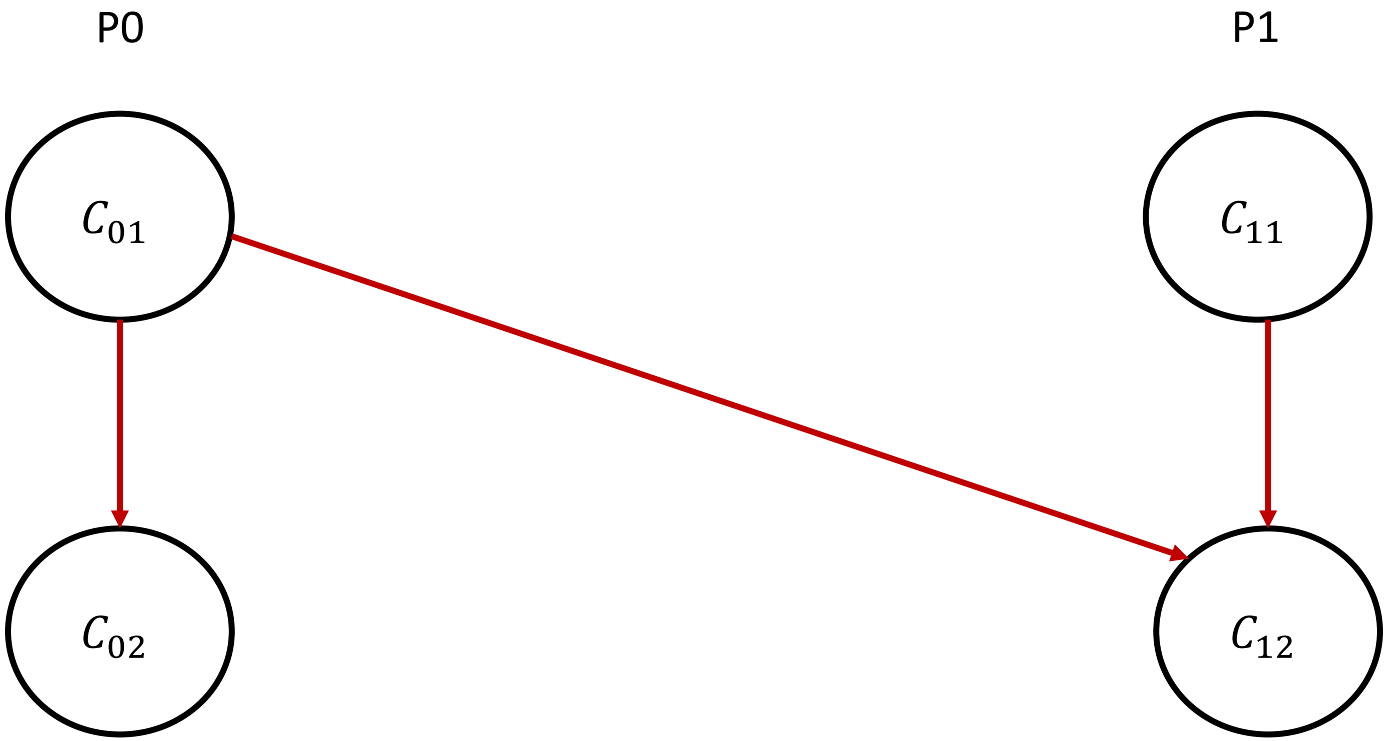
MPI_Init(…)
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank)
MPI_Comm_size(MPI_COMM_WORLD, &num_procs);

If (rank == 0) {
    D_0 = C_{00};
    Send(D_0, size, P1);
    C_{01} ; }
Else {
    C_{11};
    Receive(D_1, size, P1);
    $C_{12}(D_1)$; }



Two processor world

Inter-Process Communication

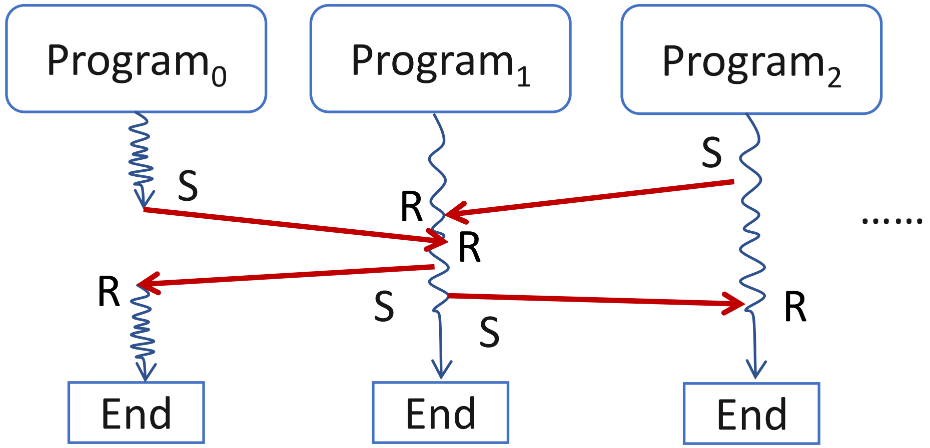


Dependency across processors due to communication operation

Message Passing Program

CASE SCHOOL OF ENGINEERING
CASE WESTERN RESERVE UNIVERSITY

Most General Model: Asynchronous



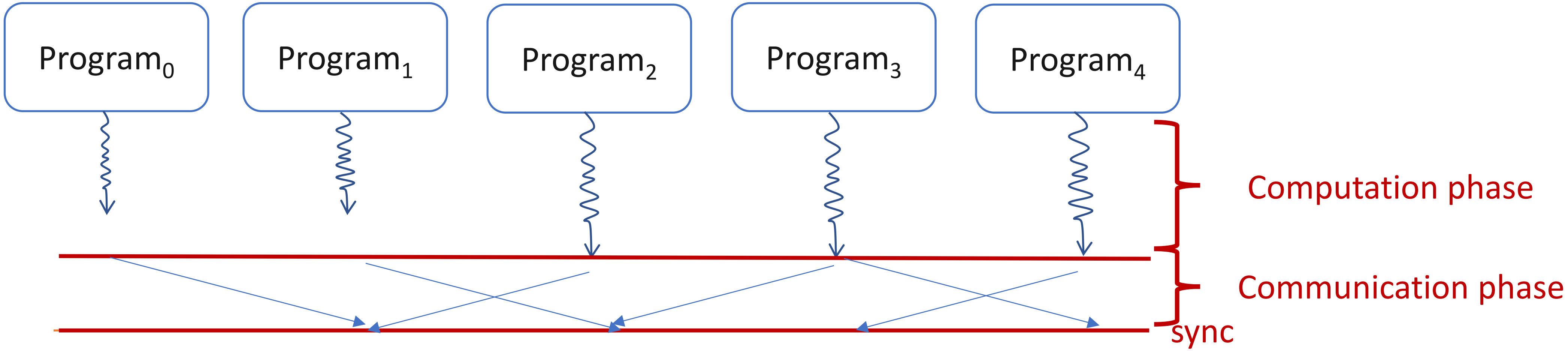
S = send
R = receive

- No structure with respect to instructions, interactions
- No global clock
- Execution is asynchronous
- Programs $0,1,\ldots,p-1$ can be all distinct
- Hard to write/debug

Message Passing Program

CASE SCHOOL OF ENGINEERING
CASE WESTERN RESERVE UNIVERSITY

Bulk synchronous



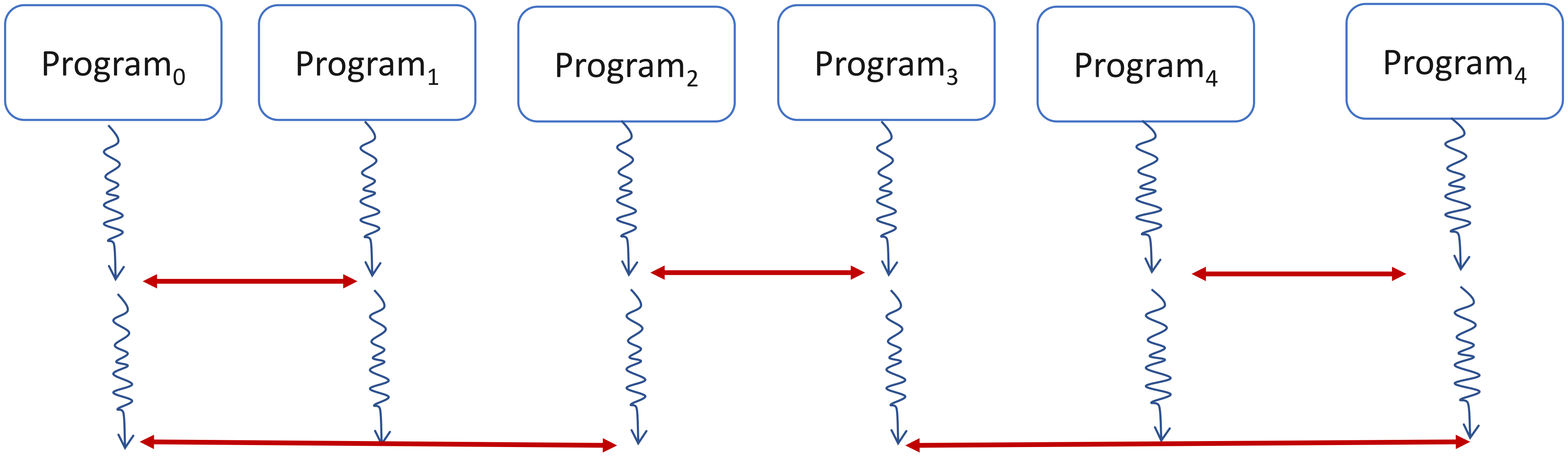
Two phases to the Program

- Computation Phase: Each process executes independently. No communication
- Communication Phase: Processes communicate with each other

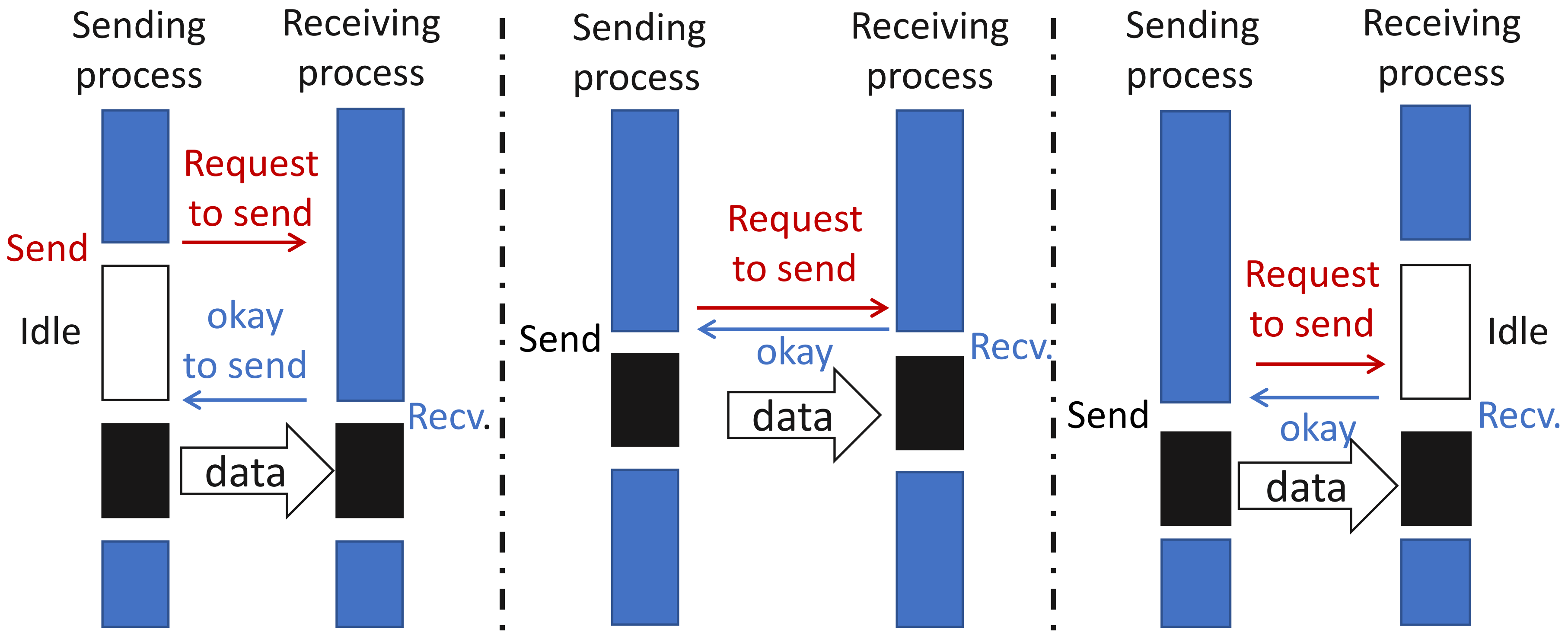
Message Passing Program

SPMD (Single Program Multiple Data)

- Code is same in all the processes except for initialization
- Restrictive model, easy to write and debug
- Easy to do performance analysis
- Widely used in Machine Learning Training



Blocking Non-Buffered Send/Receive



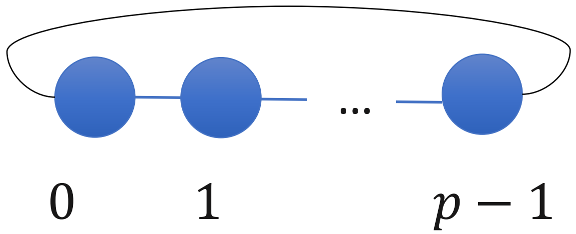
(a) Sender comes first; idling at sender

(b) Sender and Receiver come at about the same time; idling minimized

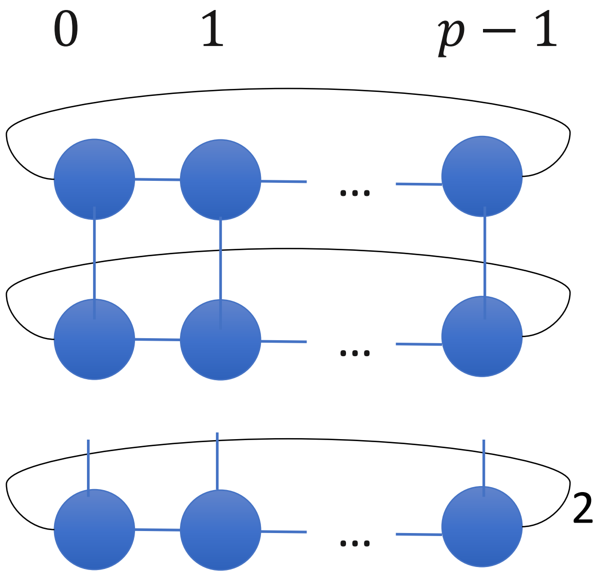
(c) Receiver comes first; idling at Receiver

Virtual Topology

- Define the "connectivity pattern" of the processors
- Helps us in developing more intuitive notions of message passing algorithms
- Think of it as 1D versus 2D arrays. It will be hard to visualize matrix multiplication if we write algorithms using 1D arrays



1D Mesh



2D Mesh

Message Passing Programming Paradigm

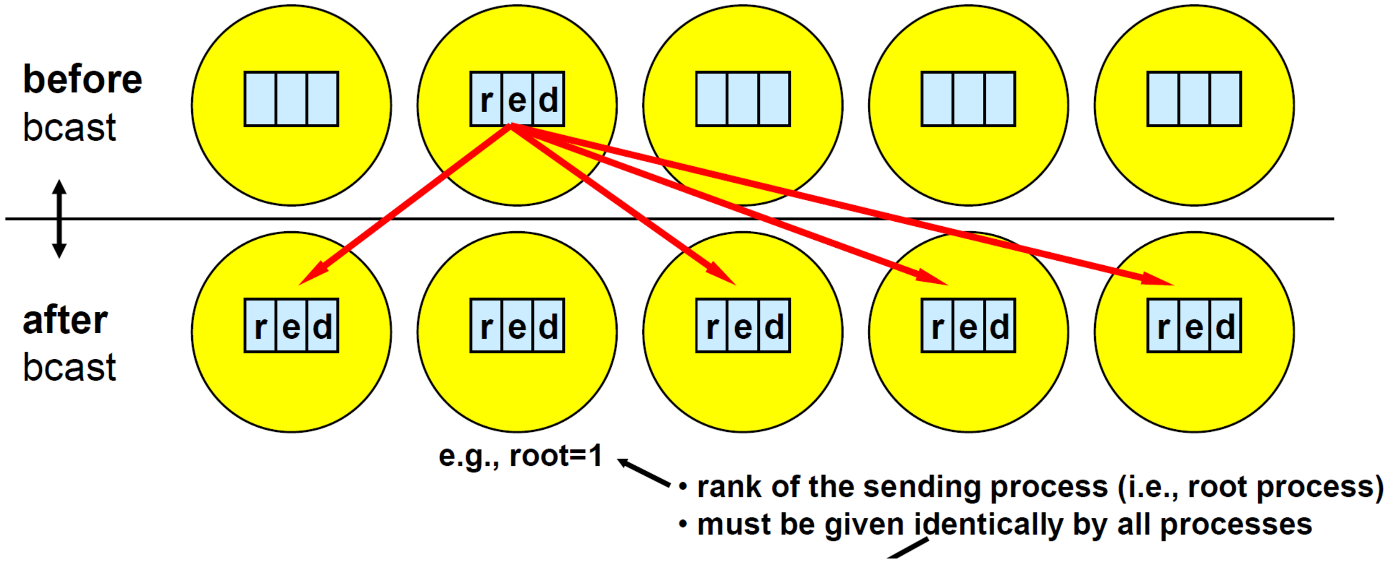
- User Specifies the following
  - Concurrency Model (BSP, SPMD, None/Asynchronous)
    - Processes: The number of processes and the work performed for each process
    - Send, receive that enable data (we will only use blocking non-buffered semantics)
  - A virtual topology of the processes

- We will discuss it at algorithmic level.
  - Skip initialization, rank calculation, etc.

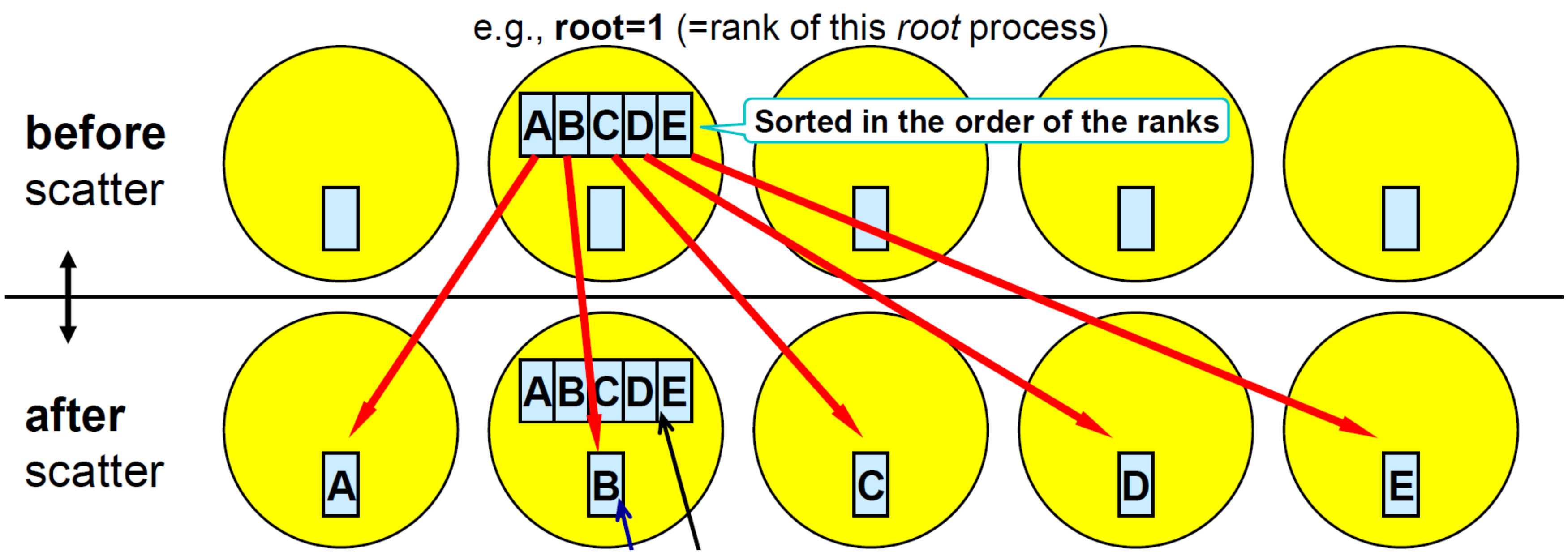
MPI_Broadcast

- MPI_Broadcast(buffer, size, root)

- Send the data stored in **buffer** at the **root** processor to all the processors

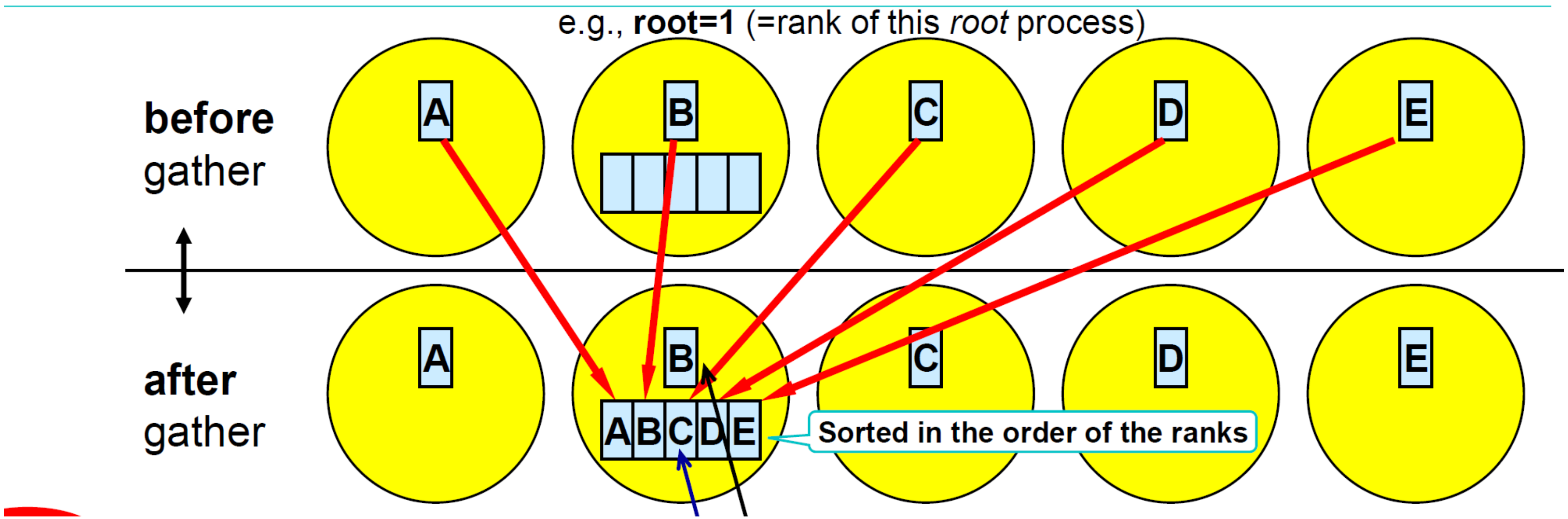


MPI_Scatter

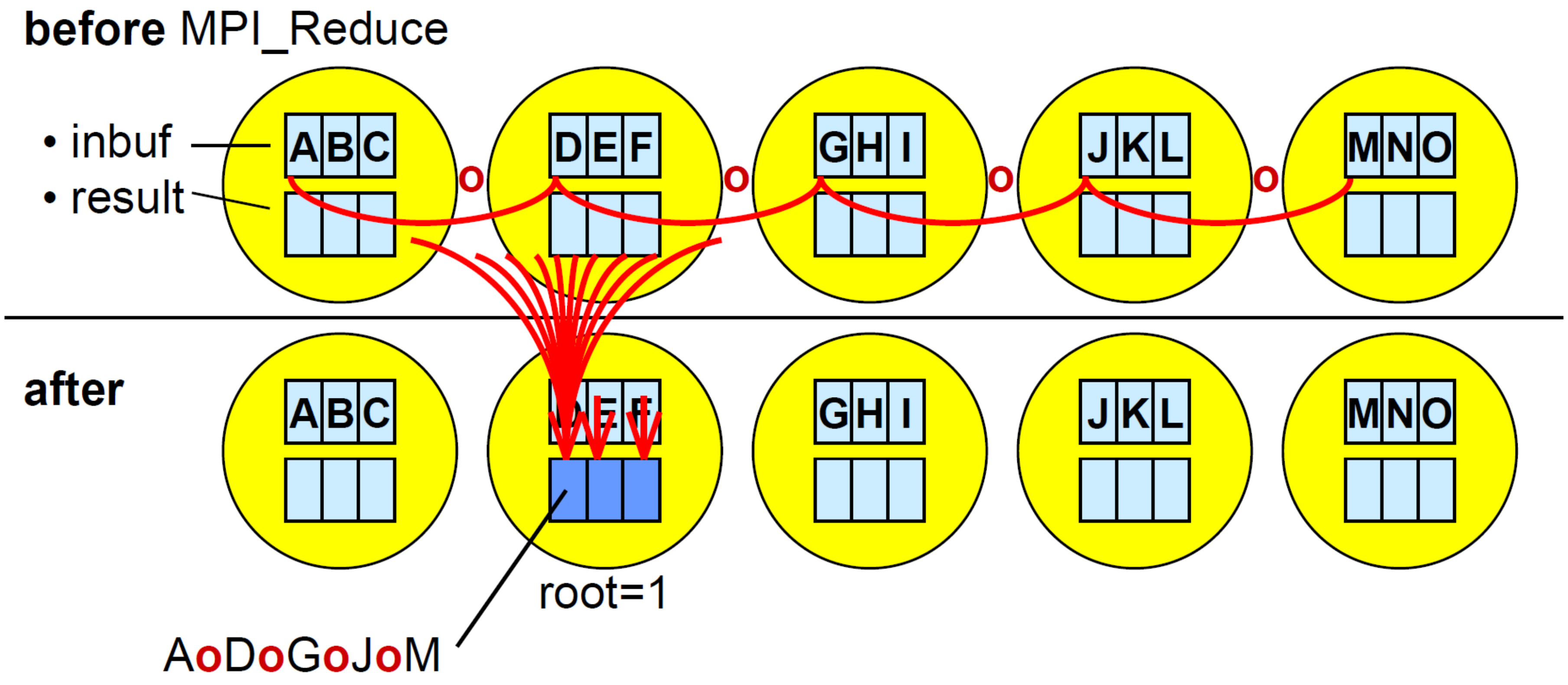


e.g., **root=1** (=rank of this *root* process)
before scatter
after scatter
ABCDE
Sorted in the order of the ranks
ABCDE
A
B
C
D
E

MPI_Gather


e.g., root=1 (=rank of this root process)
before
gather
after
gather
A
B
C
D
E
A
B
C
D
E
ABCDE
Sorted in the order of the ranks

MPI_Reduce



before MPI_Reduce
• inbuf
• result
A B C
o
D E F
o
G H I
o
J K L
o
M N O
after
A B C
D E F
G H I
J K L
M N O
root=1
AoDoGoJoM

MPI_AllReduce

- MPI_AllReduce(inbuf, resultbuf, size)

- Same as MPI_Reduce() but stores the results in all the processors

- Equivalent to:
  - MPI_Reduce(inbuf, ibsize, resultbuf, rbsize, Aggr_op, root)
  - MPI_Broadcast(resultbuf, rbsize, root)

Training

$$E(W)\text{: } \min_{W} \sum (y_i - F(x_i : W))\text{^}2$$

$$W_{t+1} \leftarrow W_t - \alpha \frac{\delta E(W)}{\delta W}$$

Iteratively

$$\frac{\delta E(W)}{\delta W} = 2\sum \underbrace{\left(y_i - F(x_i : W)\right)}_{\text{Error}} \times \frac{\delta F(x_i : W)}{\delta W}$$

For all samples, in each iteration

CASE SCHOOL OF ENGINEERING
CASE WESTERN RESERVE UNIVERSITY

Training

L layers in CNN

Weights (Filters/Kernels)

$$W_1, W_2, W_3, \dots, W_L$$

$$O_1, O_2, O_3, \dots, O_L$$

Output of layer

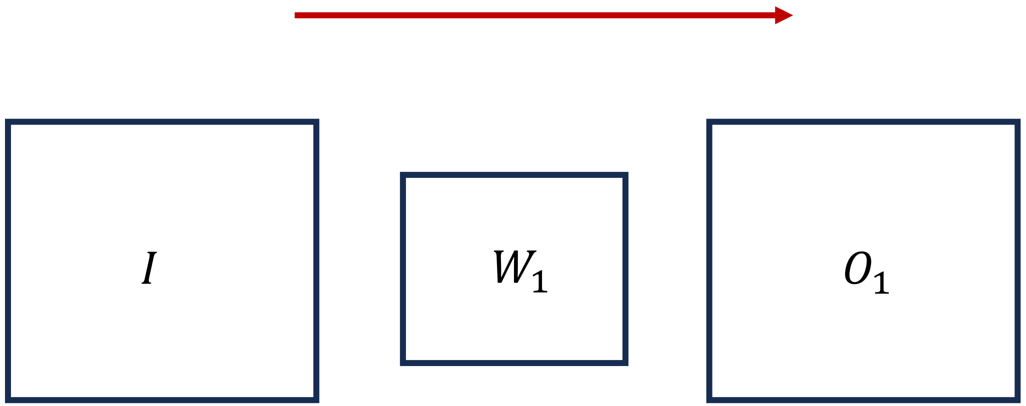
$$2 * Error \; * \; \frac{\delta F(x_i : W_{1:L})}{\delta W_1}, \frac{\delta F(x_i : W_{1:L})}{\delta W_2}, \dots, \frac{\delta F(x_i : W_{1:L})}{\delta W_L}$$

CNN Training: Forward Propagation

I

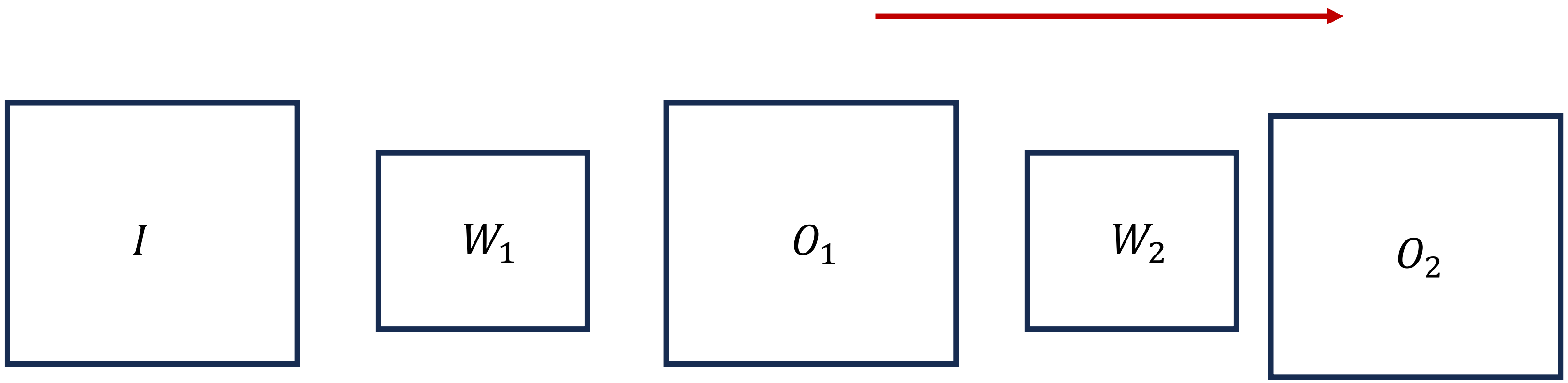
For a single image

CNN Training: Forward Propagation



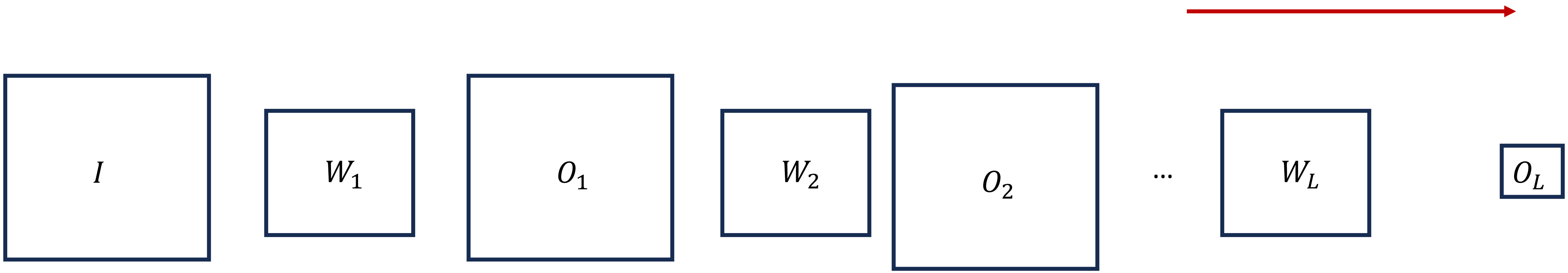
For a single image

CNN Training: Forward Propagation



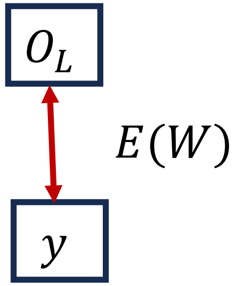
For a single image

CNN Training: Forward Propagation



For a single image

CNN Training: Error Calculation



For a single image

CNN Training: Backward Propagation

W_L

O_L

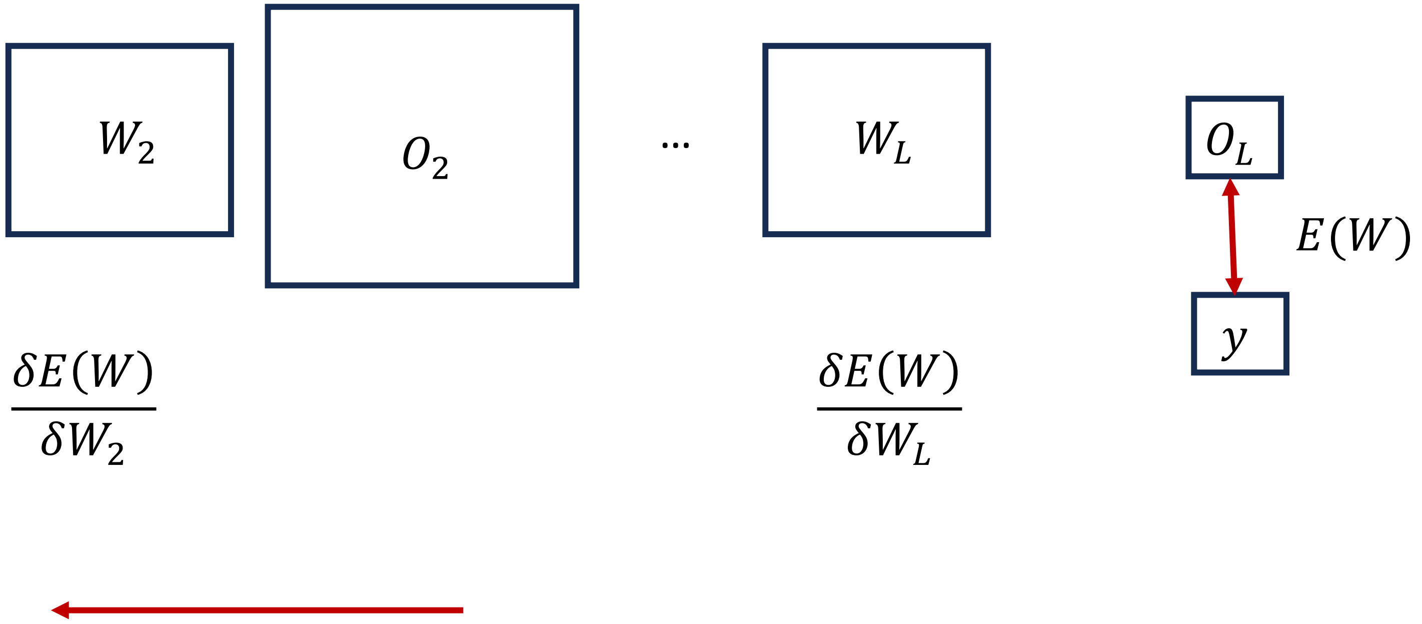
$E(W)$

y

$$\frac{\delta E(W)}{\delta W_L}$$

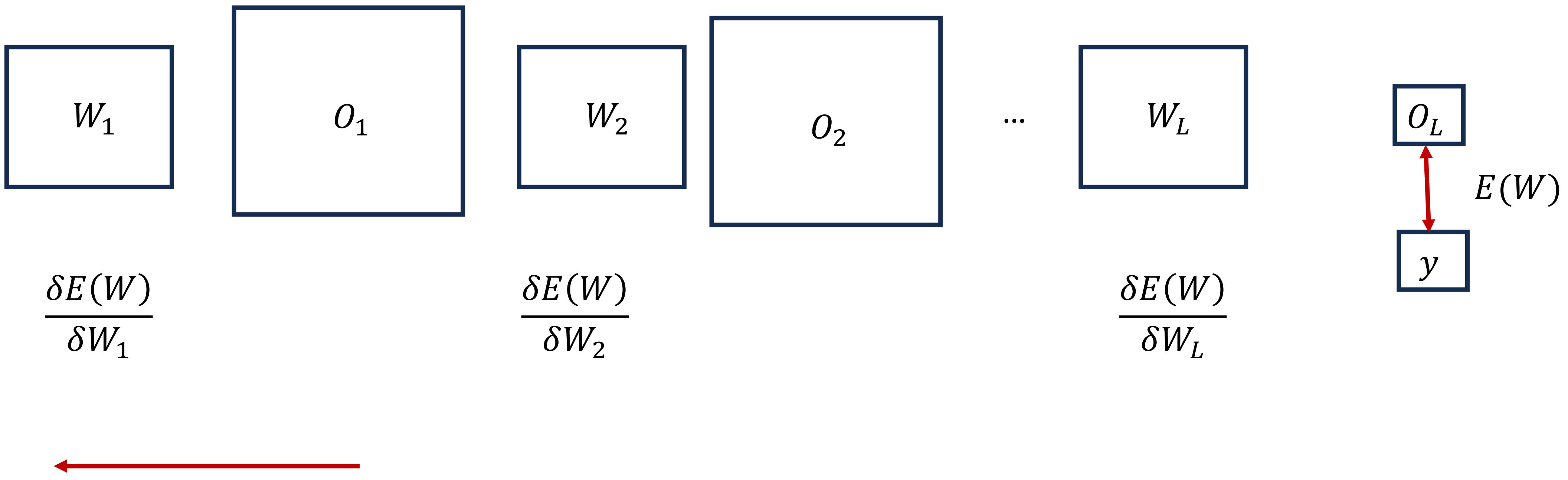
For a single image

CNN Training: Backward Propagation



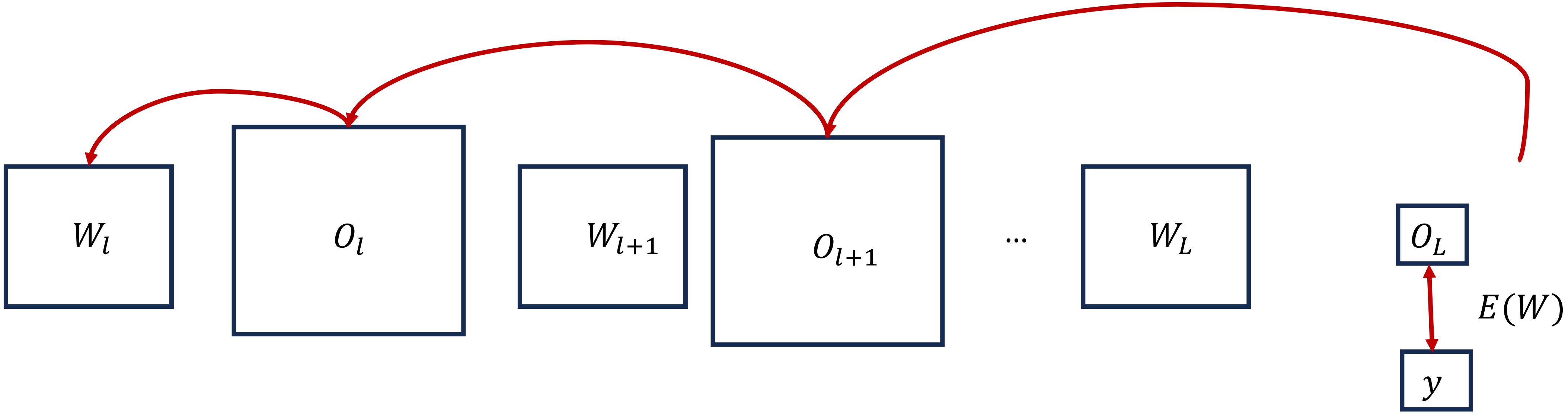
For a single image

CNN Training: Backward Propagation



For a single image

CNN Training: Backward Propagation



$$\frac{\delta E(W)}{\delta W_l} = \frac{\delta O_l}{\delta W_l} \times \frac{\delta O_{l+1}}{\delta O_l} \times \frac{\delta O_{l+2}}{\delta O_{l+1}} \times \cdots \times \frac{\delta O_L}{\delta O_{L-1}} \times \frac{\delta E(W)}{\delta O_L}$$

For a single image

Training

$$E(W):\ \min_{W} \sum (y_i\ - F(x_i : W))\hat{}2$$

$$W_{t+1} \leftarrow W_t - \alpha \frac{\delta E(W)}{\delta W}$$

Iteratively

$$\frac{\delta E(W)}{\delta W} = 2\sum \underbrace{\left(y_i\ - F(x_i : W)\right)} \times \frac{\delta F(x_i{:}W)}{\delta W}$$

For all samples, in each iteration

Error

For multiple images in a batch: Simply sum up the gradients

$$\frac{\delta E(W)}{\delta W_l} = \boxed{\frac{\delta O_l}{\delta W_l} \times \frac{\delta O_{l+1}}{\delta O_l} \times \frac{\delta O_{l+2}}{\delta O_{l+1}} \times \cdots \times \frac{\delta O_L}{\delta O_{L-1}}} \times \boxed{\frac{\delta E(W)}{\delta O_L}}$$

Gradient Update

$$W \leftarrow W \ - \alpha \frac{\delta E(W)}{\delta W}$$

- Lets dig deeper into Gradient Update

Gradient Update

- For a mini-batch with M samples and mean squared error

- $E(W) = \sum_{i=1}^{M} E_i(W)$

- $E_i(w) = (\mathrm{O}_{\mathrm{Li}} - \mathrm{y}_{\mathrm{i}})^2$

- $\frac{\delta E(W)}{\delta W} = \sum_{i=1}^{M} \frac{\delta E_i(W)}{\delta W}$

Gradient Update

$$\frac{\delta E(W)}{\delta W} = \frac{\delta E_1(W)}{\delta W} + \frac{\delta E_2(W)}{\delta W} + \dots + \frac{\delta E_M(W)}{\delta W}$$

$$= \underbrace{2(\mathrm{O_{L1}} - \mathrm{y_1})\frac{\delta O_{L1}}{\delta W}} + \underbrace{2(\mathrm{O_{L2}} - \mathrm{y_2})\frac{\delta O_{L2}}{\delta W}} + \dots + \underbrace{2(\mathrm{O_{LM}} - \mathrm{y_M})\frac{\delta O_{LM}}{\delta W}}$$

Can be calculated in parallel

Gradient Update

$$\frac{\delta E(W)}{\delta W} = \frac{\delta E_1(W)}{\delta W} + \frac{\delta E_2(W)}{\delta W} + \ \ldots + \frac{\delta E_M(W)}{\delta W}$$

$$= 2(\mathrm{O_{L1}} - \mathrm{y_1})\frac{\delta O_{L1}}{\delta W} + 2(\mathrm{O_{L2}} - \mathrm{y_2})\frac{\delta O_{L2}}{\delta W} + \ \ldots + 2(\mathrm{O_{LM}} - \mathrm{y_M})\frac{\delta O_{LM}}{\delta W}$$

Forward propagation in parallel – Need multiple copies of model

Gradient Update

$$\frac{\delta E(W)}{\delta W} = \frac{\delta E_1(W)}{\delta W} + \frac{\delta E_2(W)}{\delta W} + \ldots + \frac{\delta E_M(W)}{\delta W}$$

$$= 2\underbrace{(\mathrm{O}_{\mathrm{L1}} - \mathrm{y}_1)}\frac{\delta O_{L1}}{\delta W} + 2\underbrace{(\mathrm{O}_{\mathrm{L2}} - \mathrm{y}_2)}\frac{\delta O_{L2}}{\delta W} + \ldots + 2\underbrace{(\mathrm{O}_{\mathrm{LM}} - \mathrm{y}_{\mathrm{M}})}\frac{\delta O_{LM}}{\delta W}$$

Error Calculation in parallel

Gradient Update

$$\frac{\delta E(W)}{\delta W} = \frac{\delta E_1(W)}{\delta W} + \frac{\delta E_2(W)}{\delta W} + \ \ldots + \frac{\delta E_M(W)}{\delta W}$$

$$= 2(\mathrm{O_{L1}} \ - \mathrm{y}_1)\frac{\delta O_{L1}}{\delta W} + 2(\mathrm{O_{L2}} \ - \mathrm{y}_2)\frac{\delta O_{L2}}{\delta W} + \ \ldots + 2(\mathrm{O_{LM}} \ - \mathrm{y_M})\frac{\delta O_{LM}}{\delta W}$$

Backward Propagation in parallel

Gradient Update

- If using p processes (or GPUs) for a mini-batch of size M
- Each process (or GPU) keeps a copy of the model W
- Each process performs the computations of M/p samples independently using its own model
- Note: No synchronization was needed till now

Gradient Update

- Weight Update Step:

$$W \leftarrow W - \alpha \frac{\delta E(W)}{\delta W}$$

- Do we need synchronization now???

Gradient Update

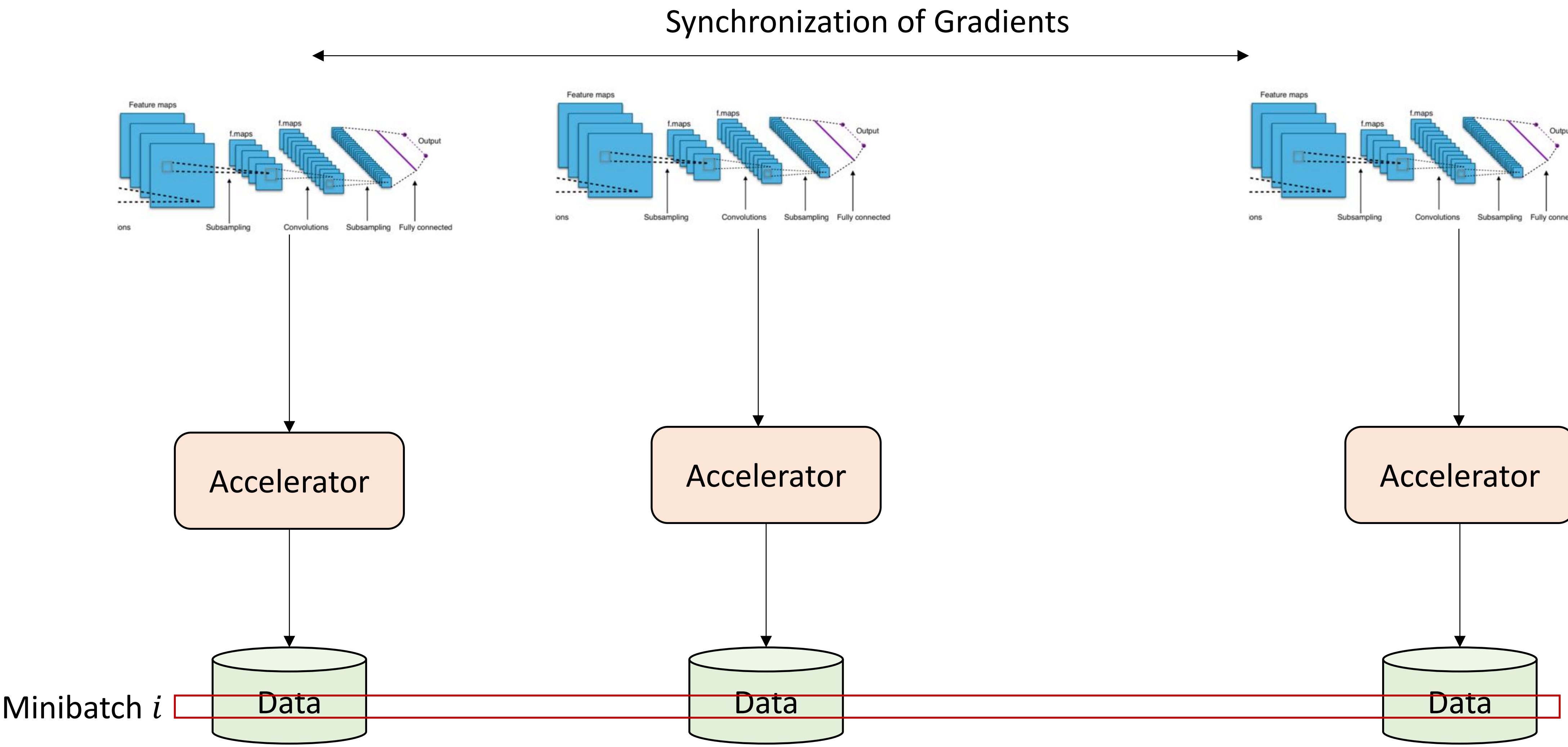
- Weight Update Step:

$$W \leftarrow W - \alpha \frac{\delta E(W)}{\delta W}$$

- Do we need synchronization now??? Yes.

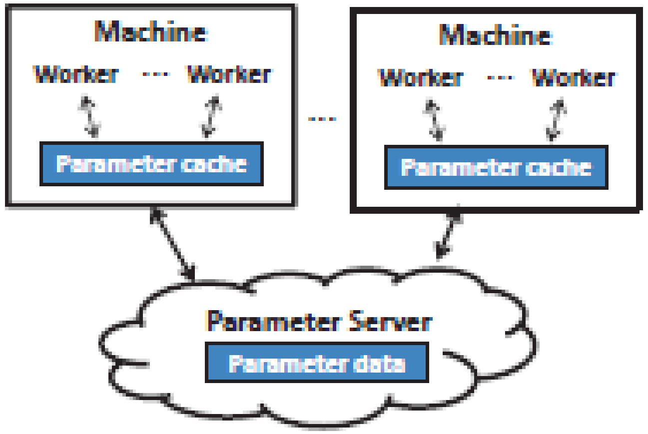
- Need to make sure that all GPUs have the same weights before they start processing the next mini-batch

Data Parallel Training of Neural Networks



Data Parallel Training of Neural Networks

- How do we synchronize?
- Model #1: Parameter Server Model
- A dedicated server performs the weight updates and distributes the models to all the workers
- Resource: Li, Mu, et al. "Scaling distributed machine learning with the parameter server." *11th USENIX Symposium on operating systems design and implementation (OSDI 14)*. 2014.

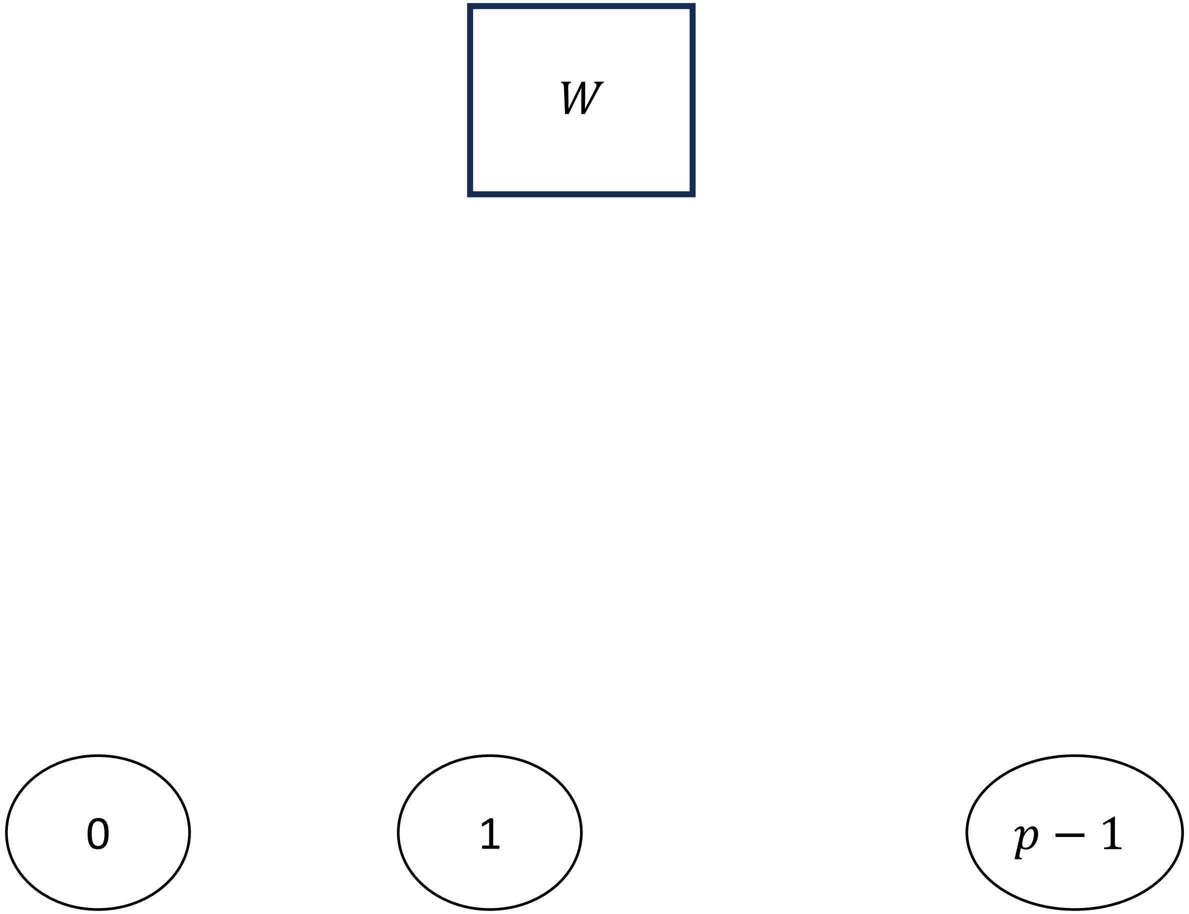


Cui, H., Zhang, H., Ganger, G. R., Gibbons, P. B., & Xing, E. P. (2016, April). Geeps: Scalable deep learning on distributed gpus with a gpu-specialized parameter server. In *Proceedings of the eleventh european conference on computer systems* (pp. 1-16).

Parameter Server Model

Parameter Server

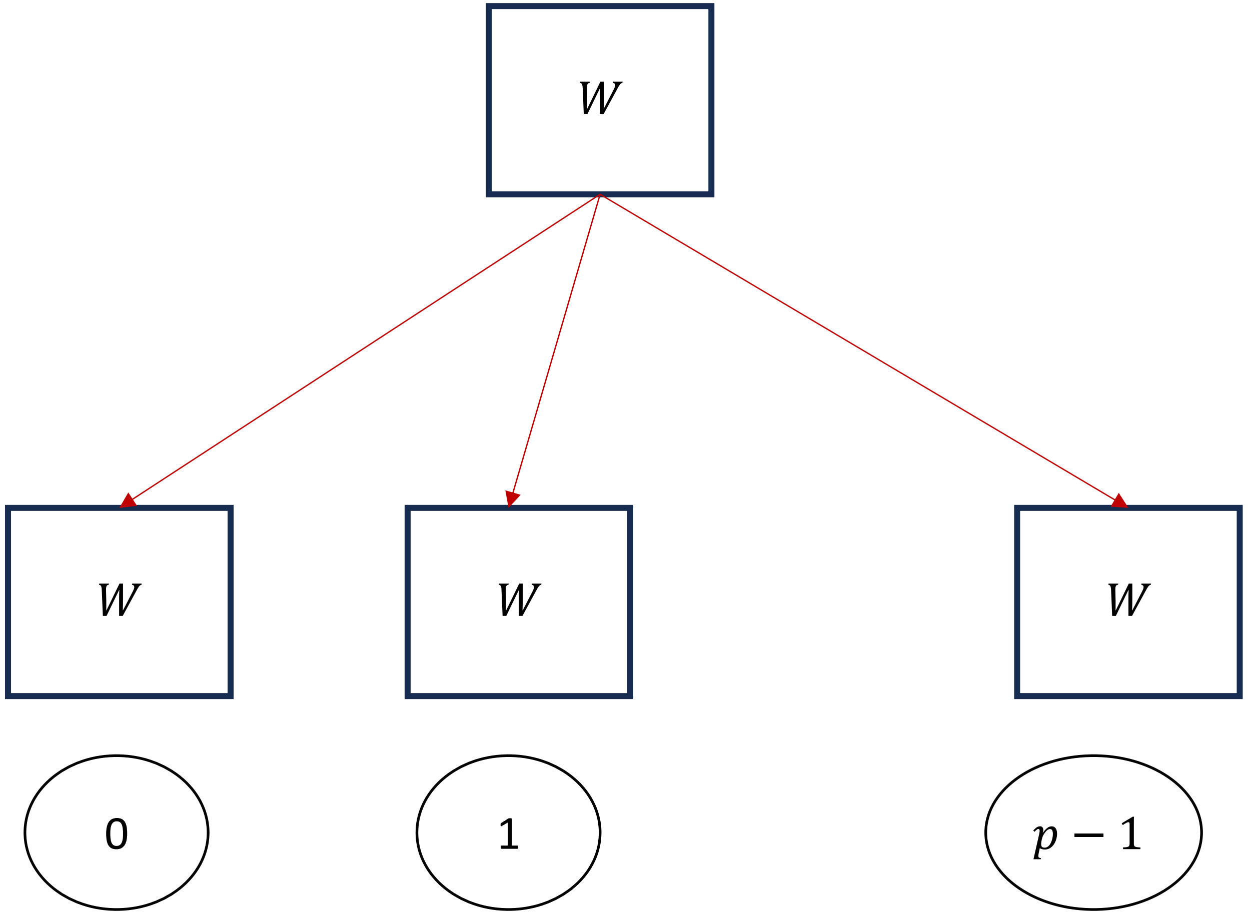
Data Parallel GPUs



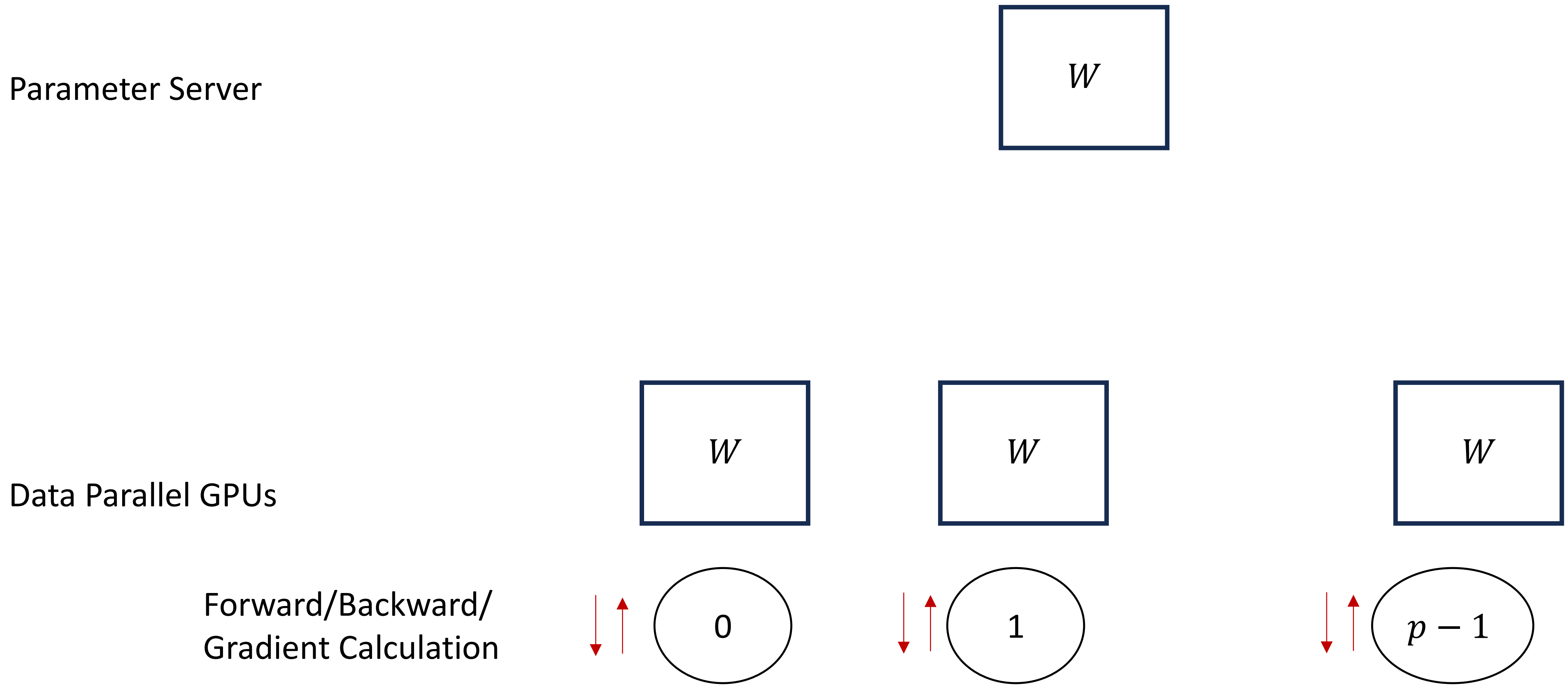
Parameter Server Model

Parameter Server

Data Parallel GPUs



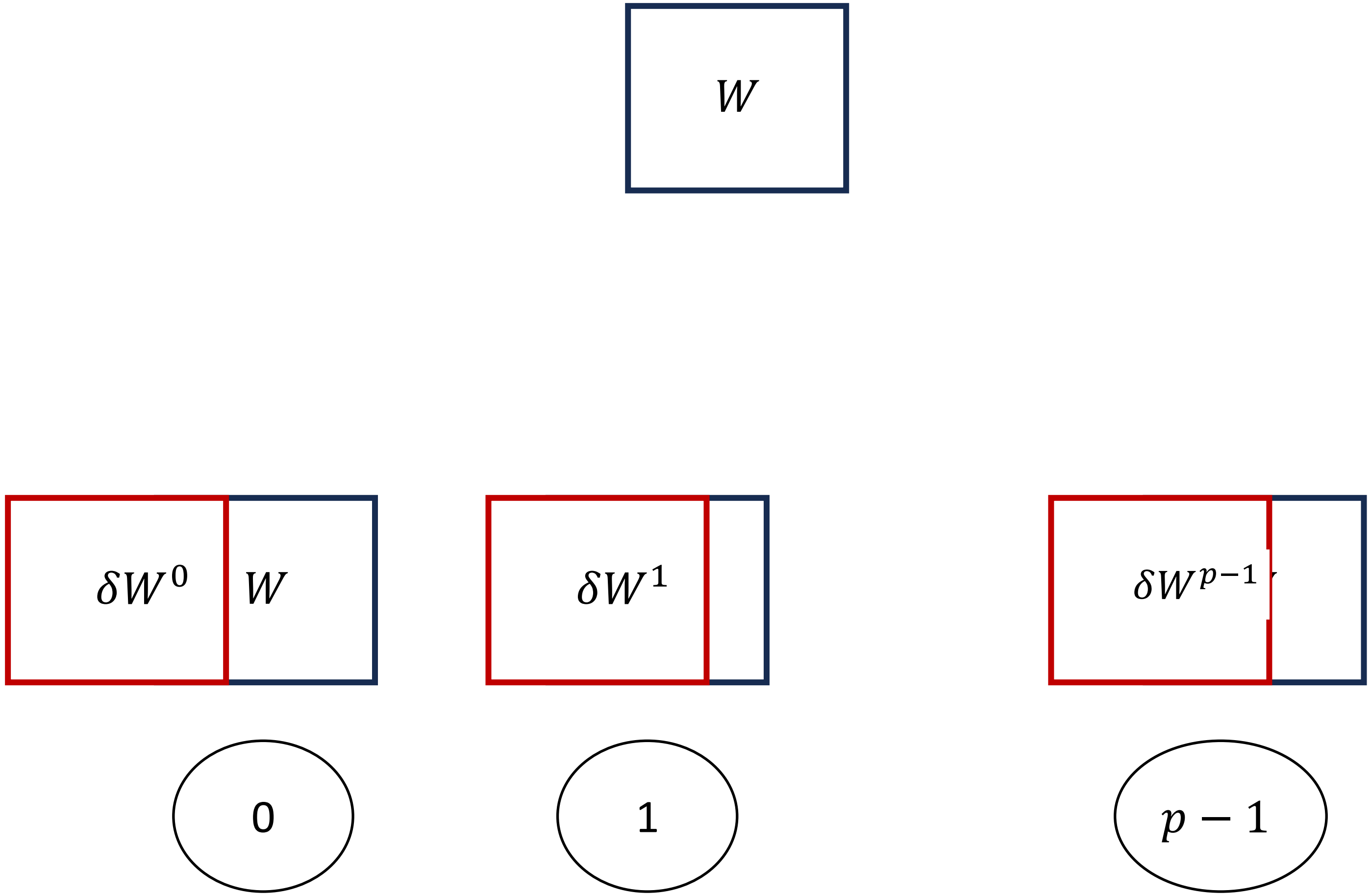
Parameter Server Model



Parameter Server Model

Parameter Server

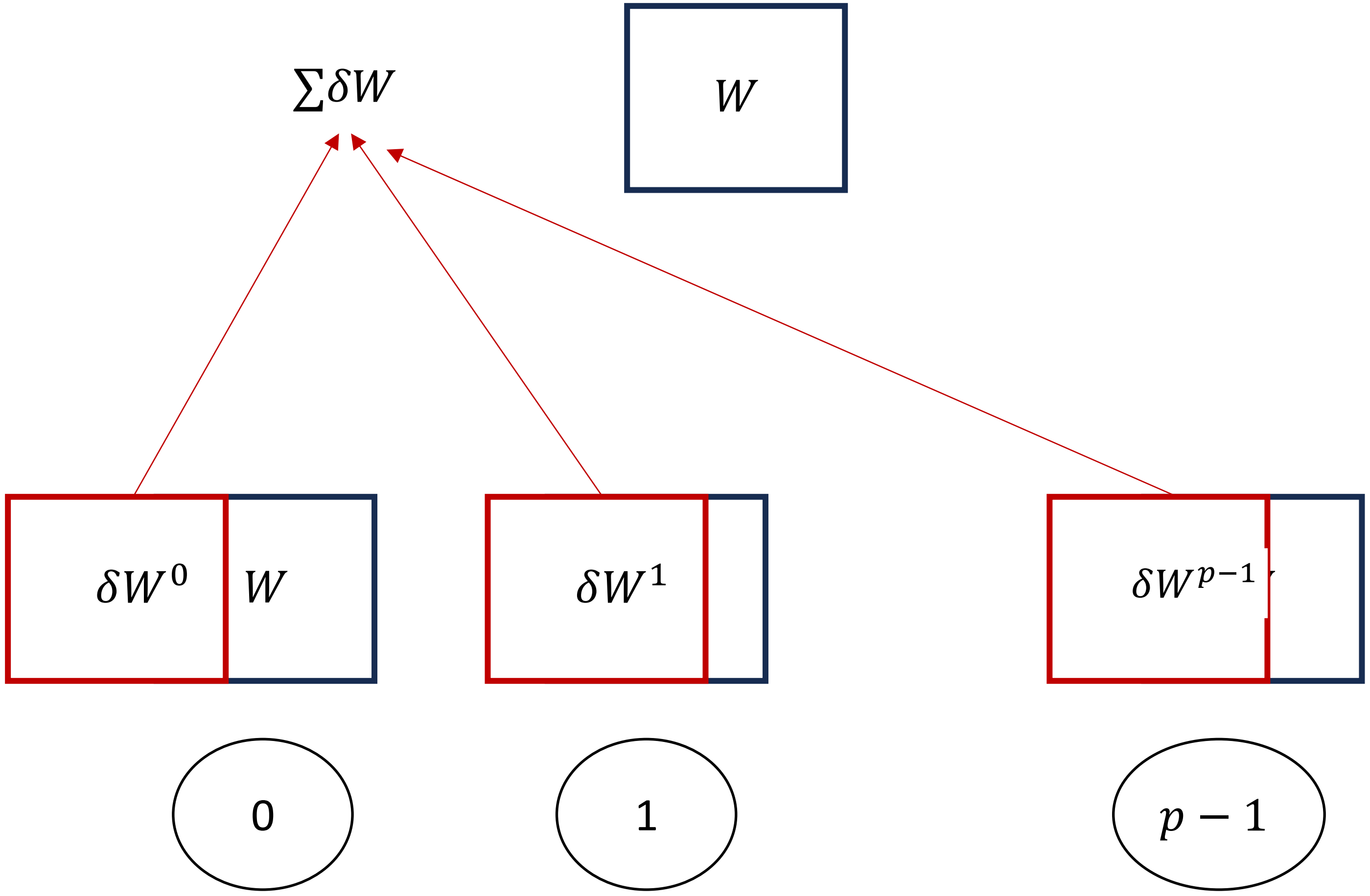
Data Parallel GPUs



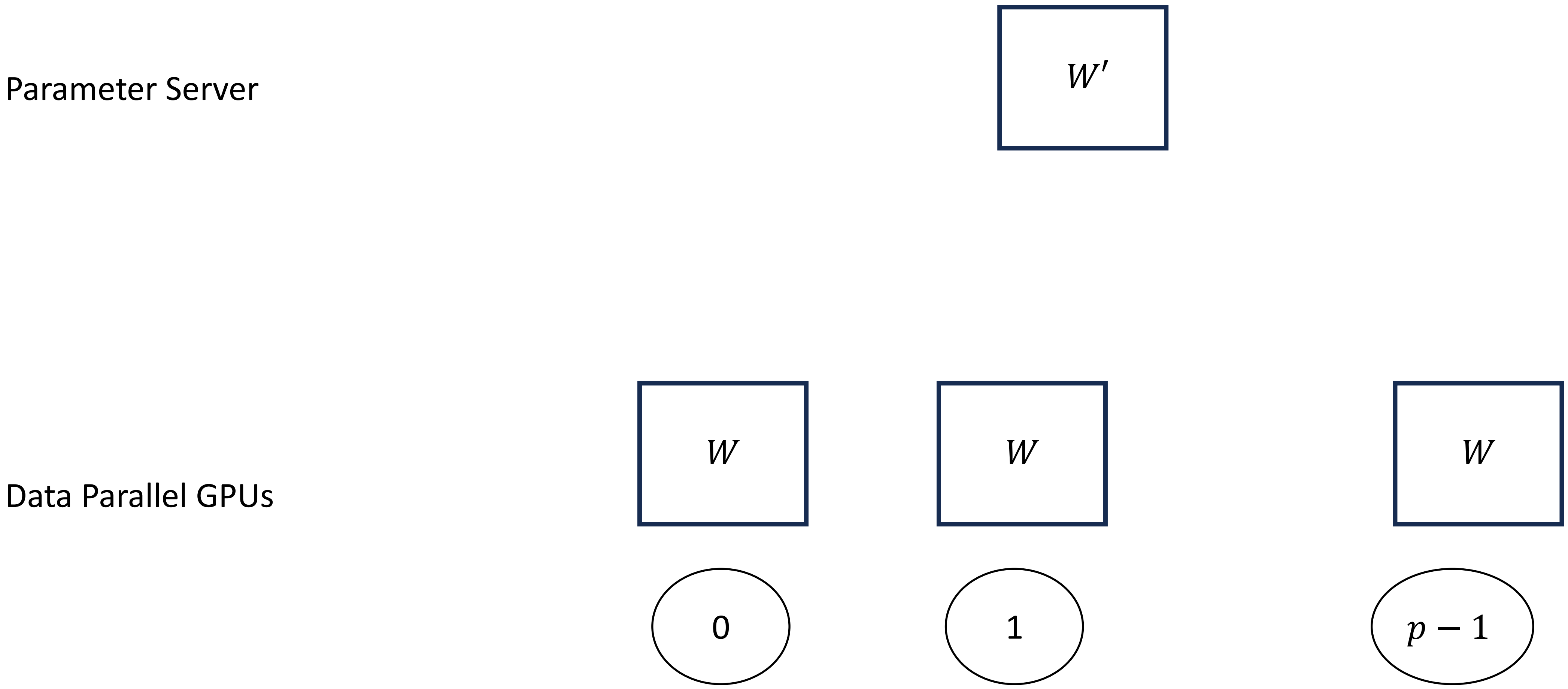
Parameter Server Model

Parameter Server

Data Parallel GPUs



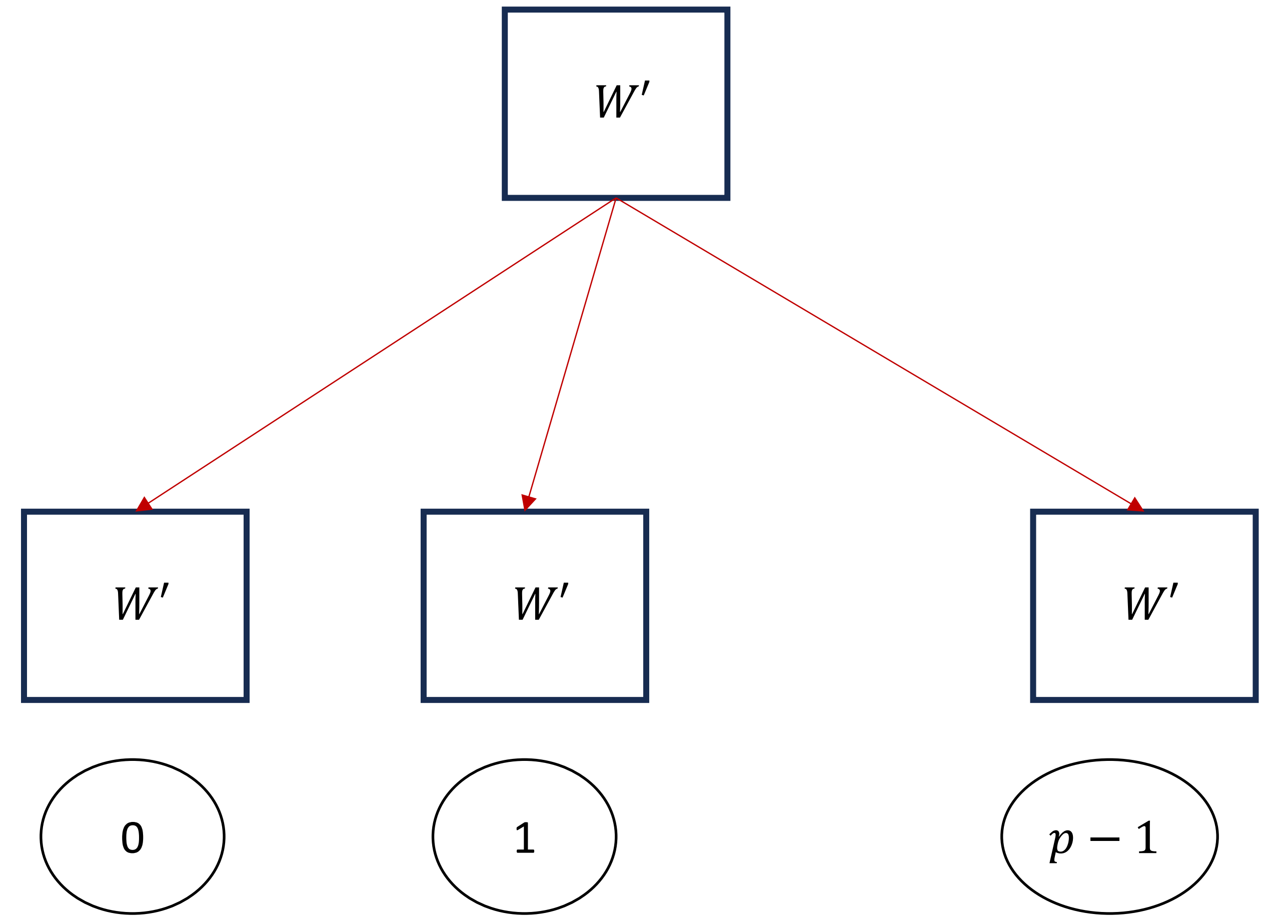
Parameter Server Model



Parameter Server Model

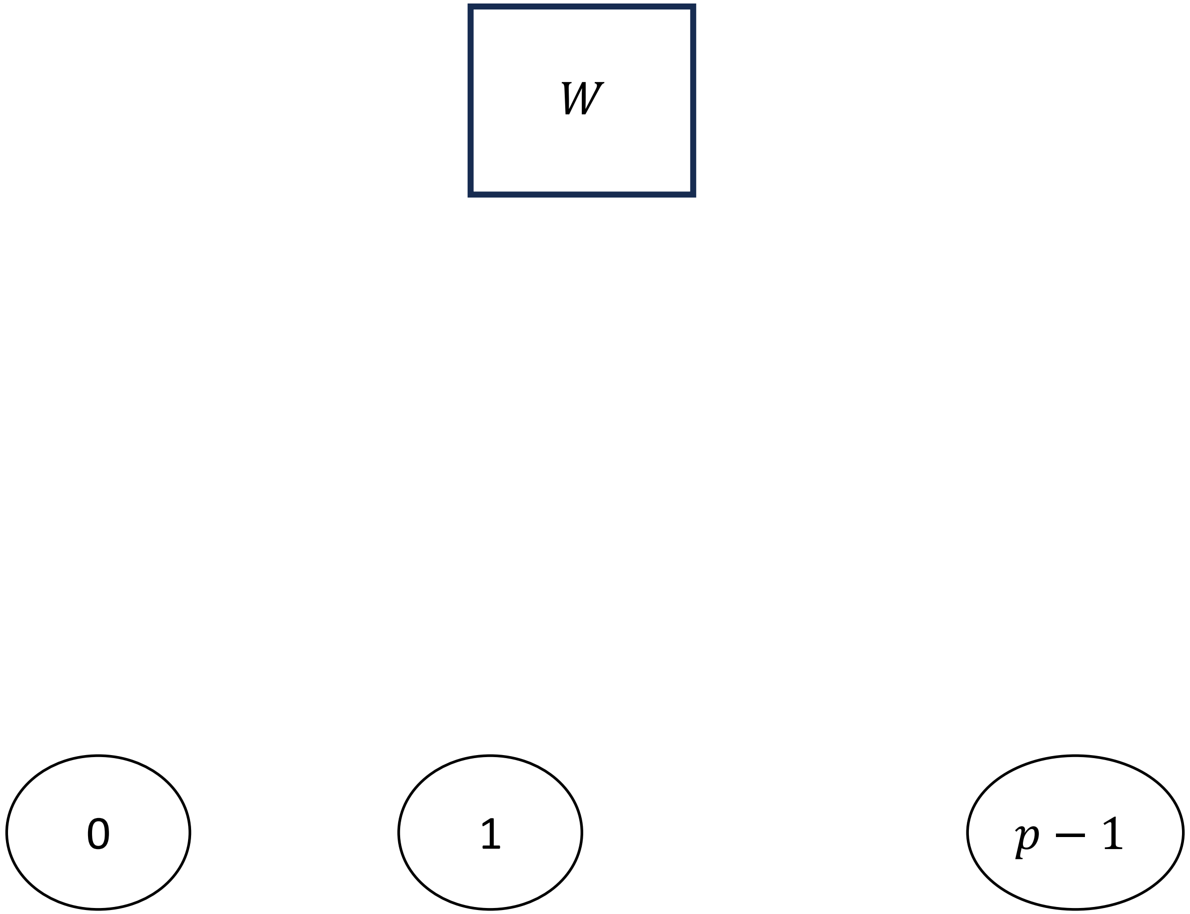
Parameter Server

Data Parallel GPUs



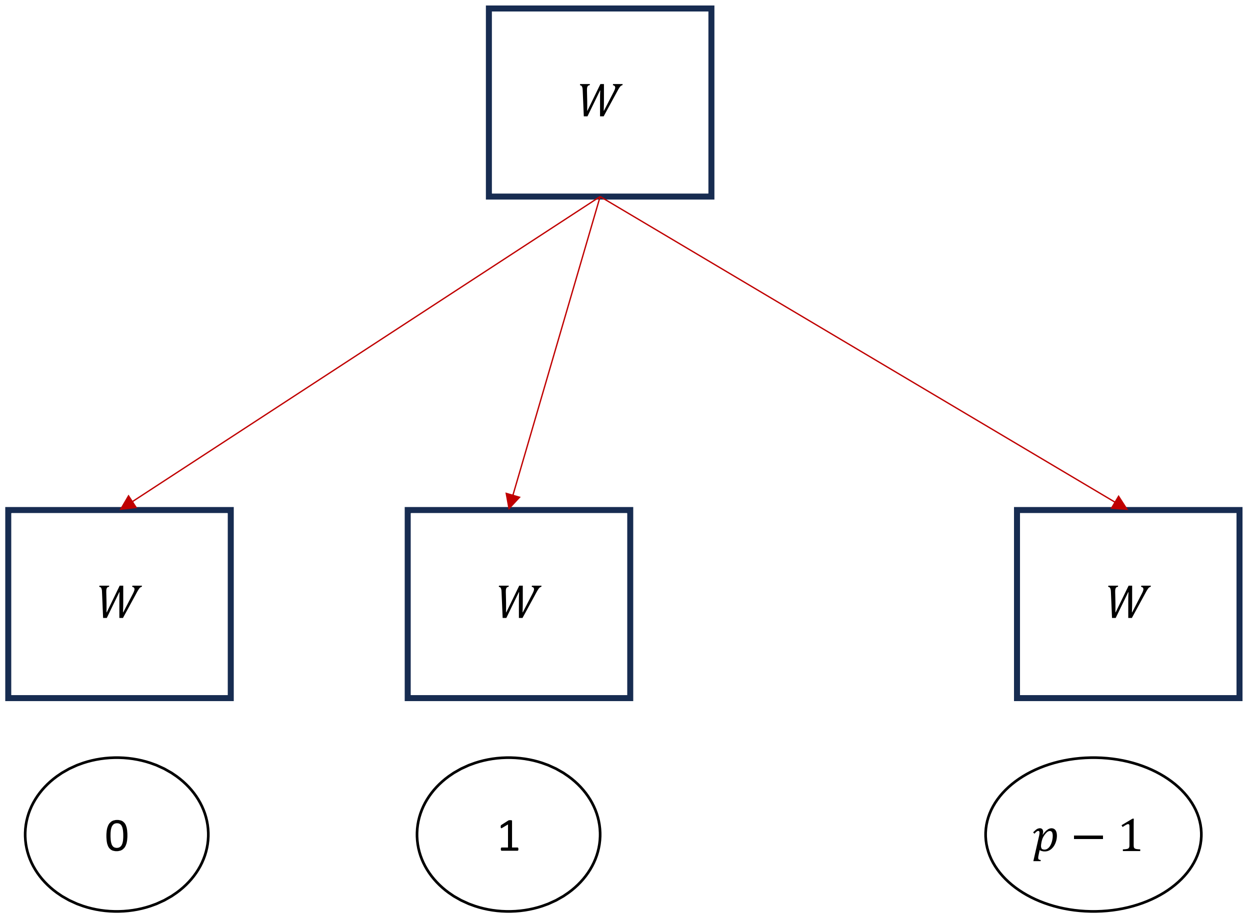
Parameter Server Using Communication Collectives

- In each Iteration
    - #1: Parameter server “sends” the weights to each processor
    - #2: Forward/Backward/gradient computation in each processor
    - #3: Each processor sends the gradient back to the parameter server which takes an average of the gradients
    - #4: Parameter server performs the gradient update step
- Repeat



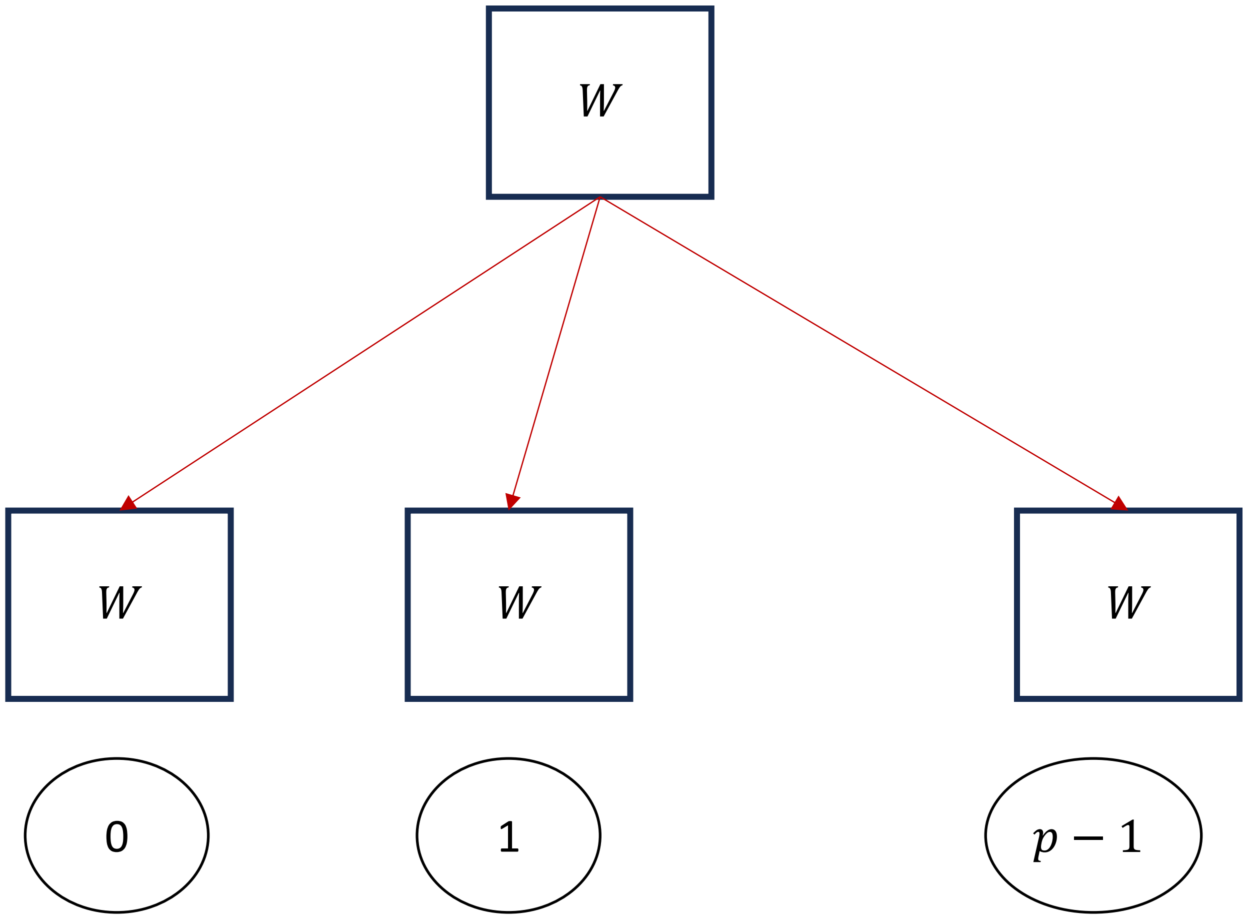
Parameter Server Using Communication Collectives

- In each Iteration
  - #1: Parameter server “sends” the weights to each processor
  - #2: Forward/Backward/gradient computation in each processor
  - #3: Each processor sends the gradient back to the parameter server which takes an average of the gradients
  - #4: Parameter server performs the gradient update step
- Repeat



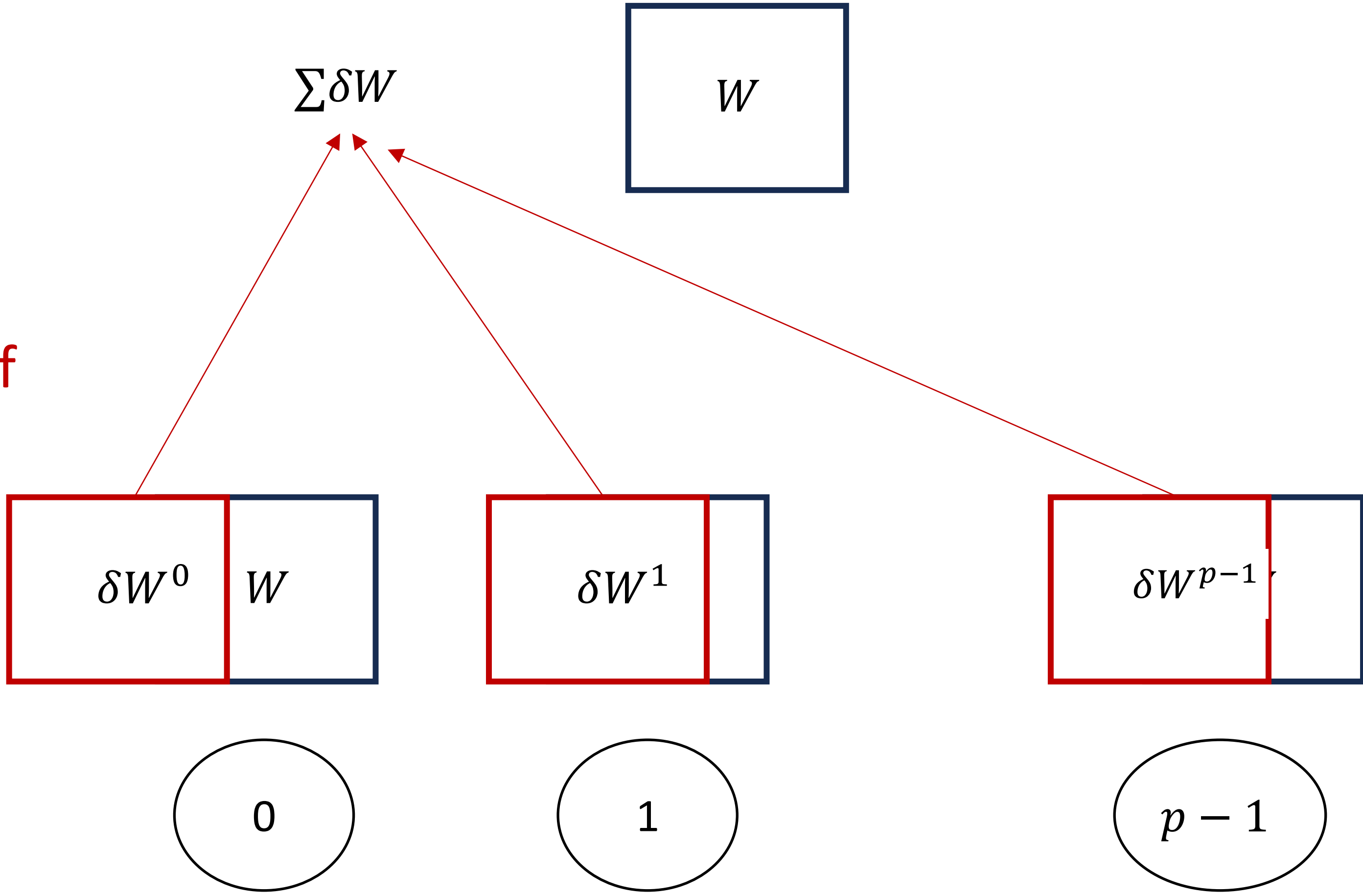
Parameter Server Using Communication Collectives

- In each Iteration
  - #1: MPI_Broadcast(W)
  - #2: Forward/Backward/gradient computation in each processor
  - #3: Each processor sends the gradient back to the parameter server which takes an average of the gradients
  - #4: Parameter server performs the gradient update step
- Repeat



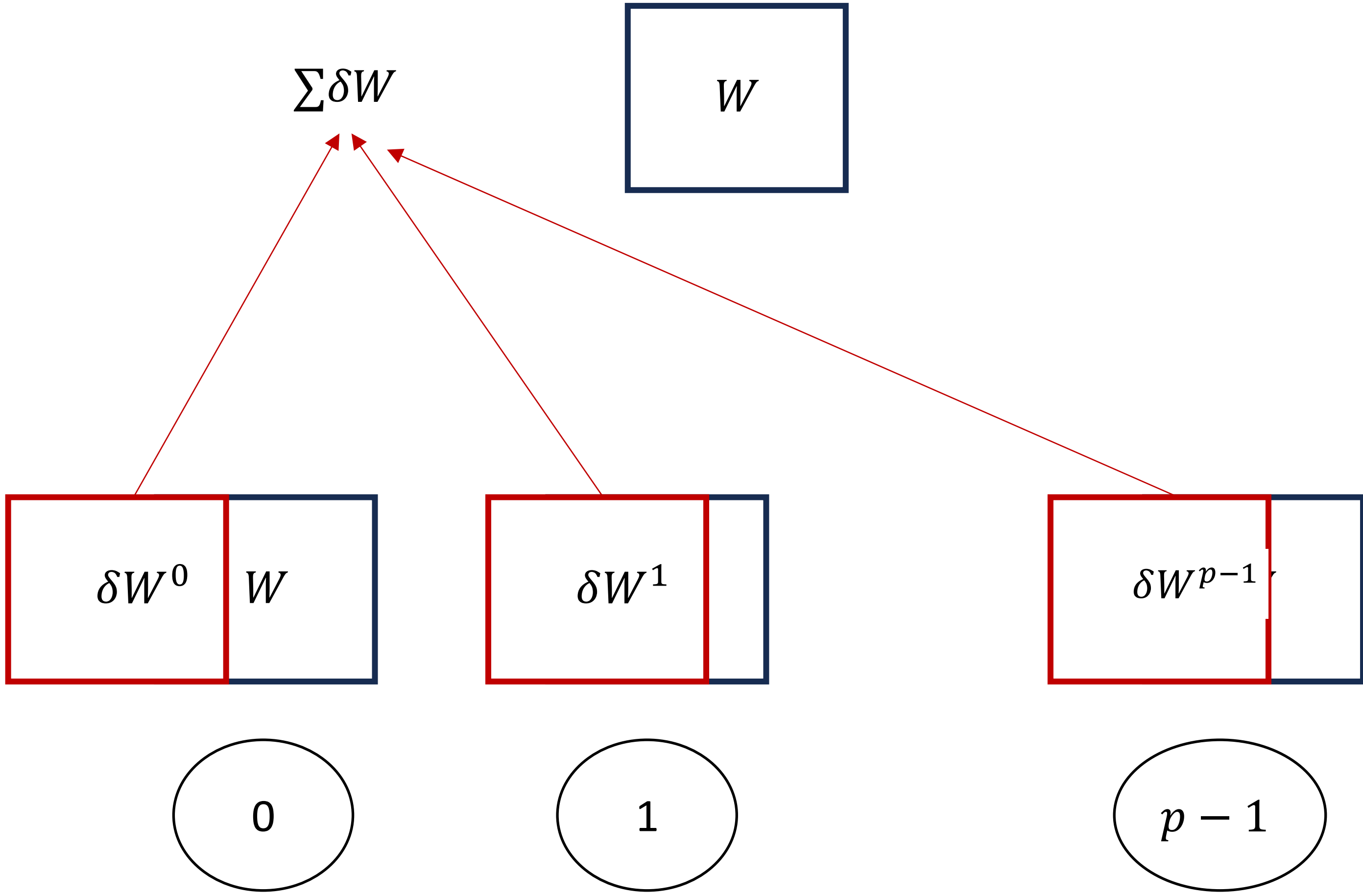
Parameter Server Using Communication Collectives

- In each Iteration
  - #1: MPI_Broadcast(W)
  - #2: Forward/Backward/gradient computation in each processor
  - #3: Each processor sends the gradient back to the parameter server which takes an average of the gradients
  - #4: Parameter server performs the gradient update step
- Repeat



Parameter Server Using Communication Collectives

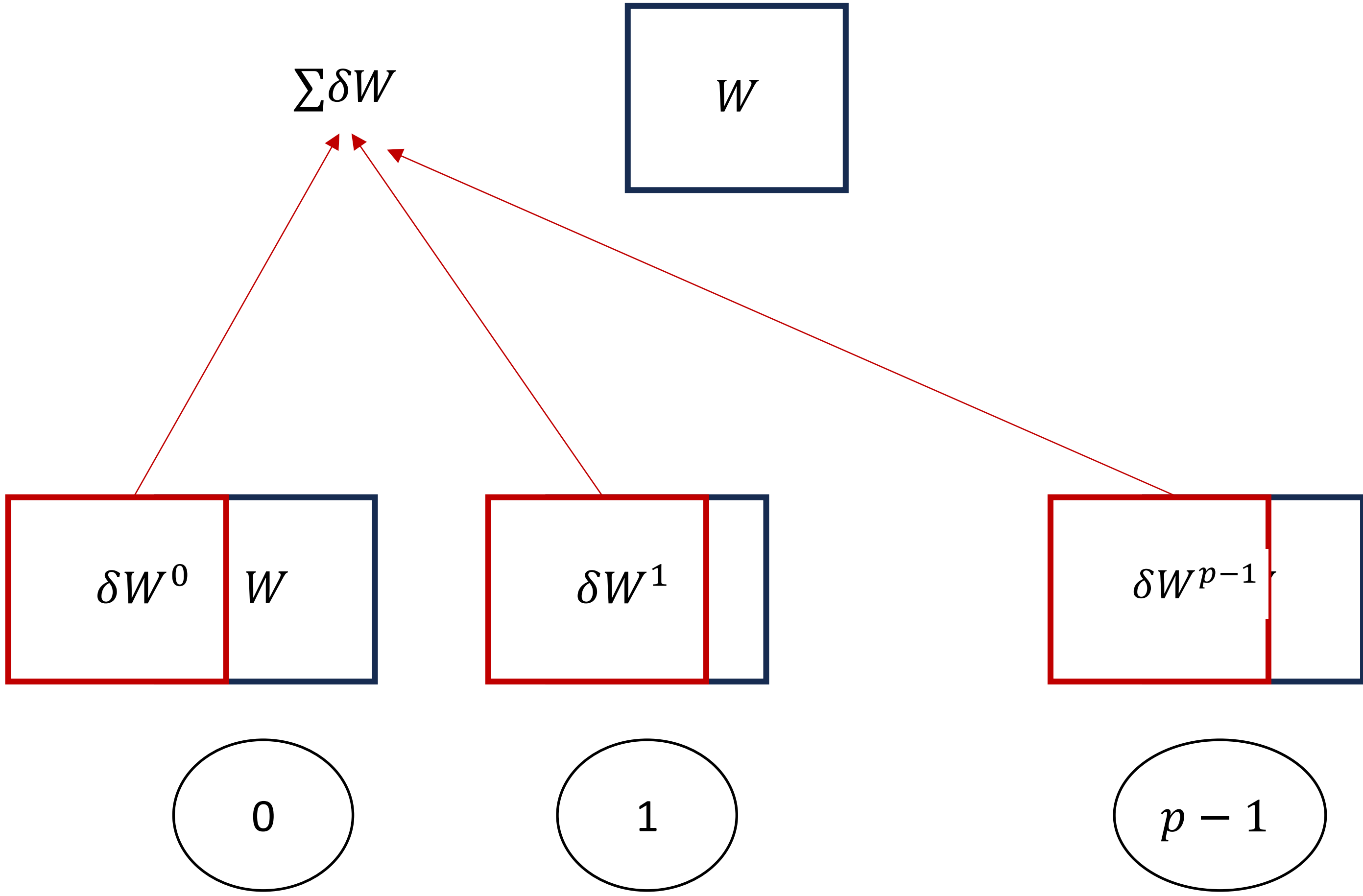
- In each Iteration
  - #1: MPI_Broadcast(W)
  - #2: Forward/Backward/gradient computation in each processor
  - #3: MPI_REDUCE(δW,AVG)
  - #4: Parameter server performs the gradient update step
- Repeat



Parameter Server Using Communication Collectives

- In each Iteration
  - #1: MPI_Broadcast(W)
  - #2: Forward/Backward/gradient computation in each processor
  - #3: MPI_REDUCE(δW,AVG)
  - #4: Parameter server performs the gradient update step
- Repeat

- Communication Complexity per Iteration (assume $|W| = N$): ??



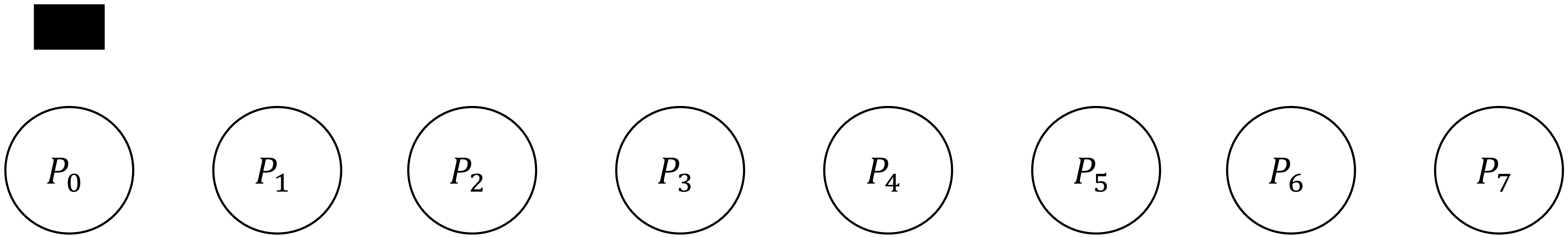
MPI_Broadcast

- MPI_Broadcast: $O(\mathrm{t_w} \times \log P)$
    - t_w: size of data to broadcast
    - P: Number of processors
    - Algorithm proceeds in $\log P$ steps, each step communicates t_w amount of data

MPI_Reduce

Can you calculate the communication complexity?

1 + 1 + … 1 $(\log P)$times = $O(\log P)$

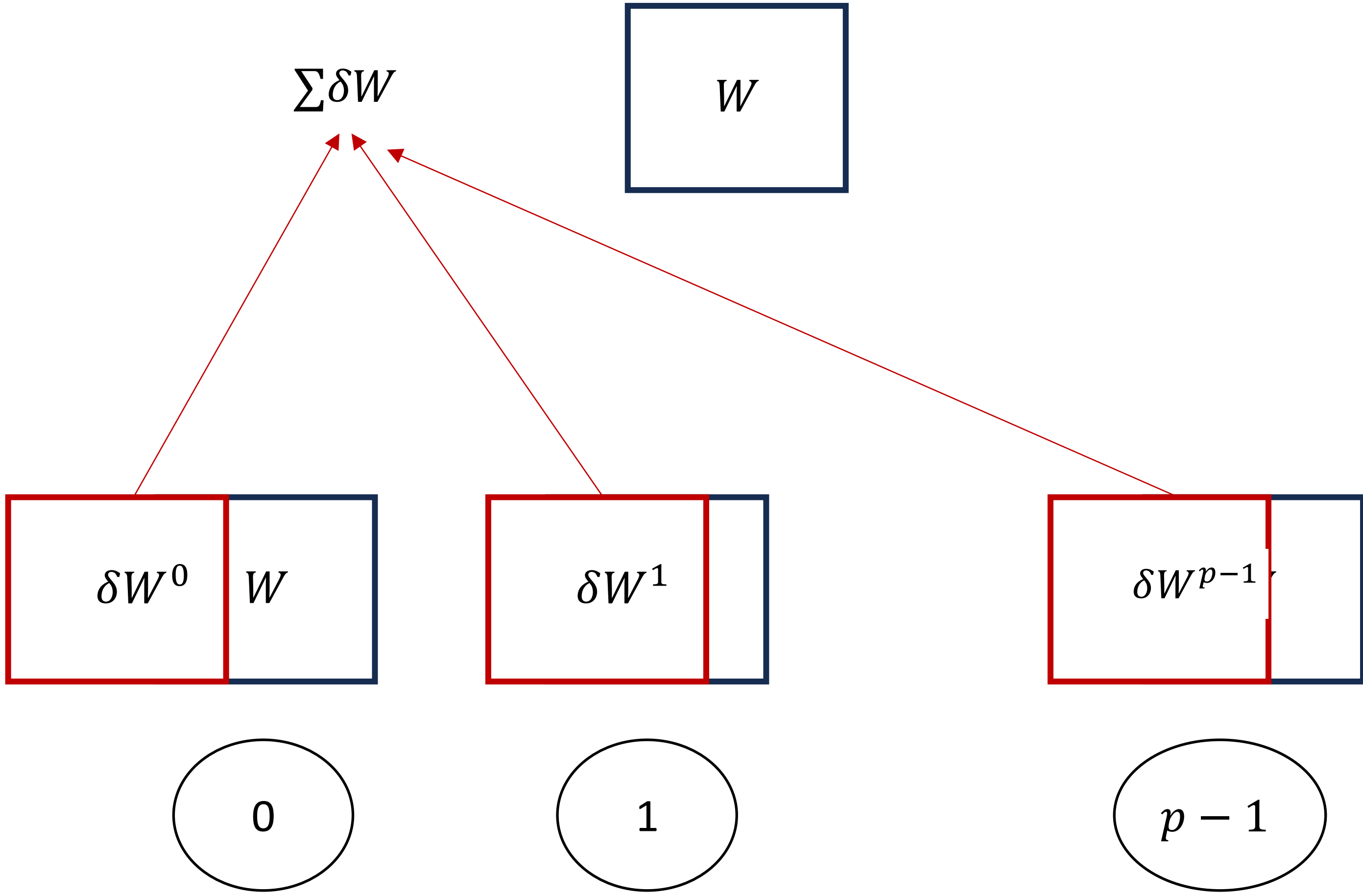


Step 3

Parameter Server Using Communication Collectives

- In each Iteration
  - #1: MPI_Broadcast(W)
  - #2: Forward/Backward/gradient computation in each processor
  - #3: MPI_REDUCE(δW,AVG)
  - #4: Parameter server performs the gradient update step
- Repeat

- Communication Complexity per Iteration (assume $|W| = N$): $O(N \log(p+1))$

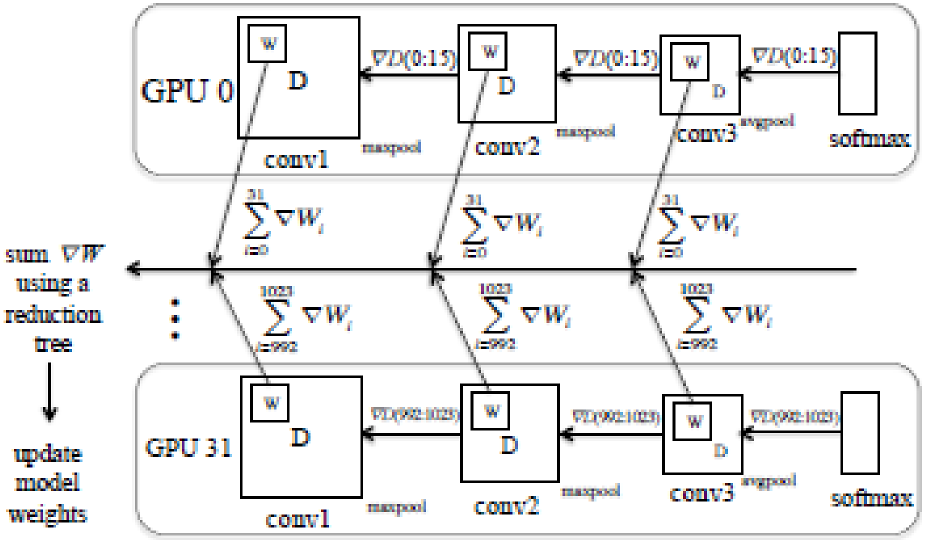


Data Parallel Training of Neural Networks

- How do we synchronize?

- Model #2: All-reduce based Synchronization

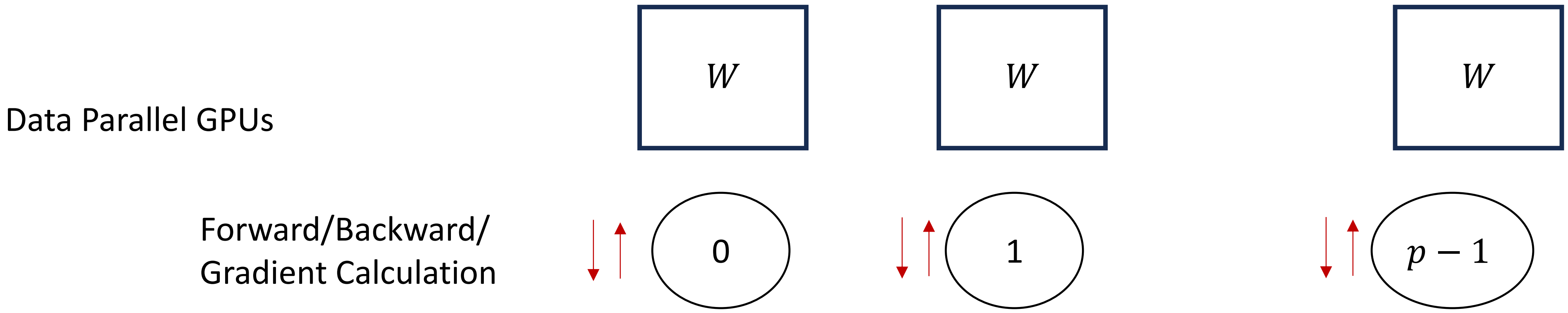
- No dedicated parameter server. Each worker calculates the gradients and broadcasts it to all other workers
- Each worker aggregates the gradients collected by all other workers and performs the update step



Iandola, F. N., Moskewicz, M. W., Ashraf, K., & Keutzer, K. (2016). Firecaffe: near-linear acceleration of deep neural network training on compute clusters. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 2592-2600).

All Reduce Model

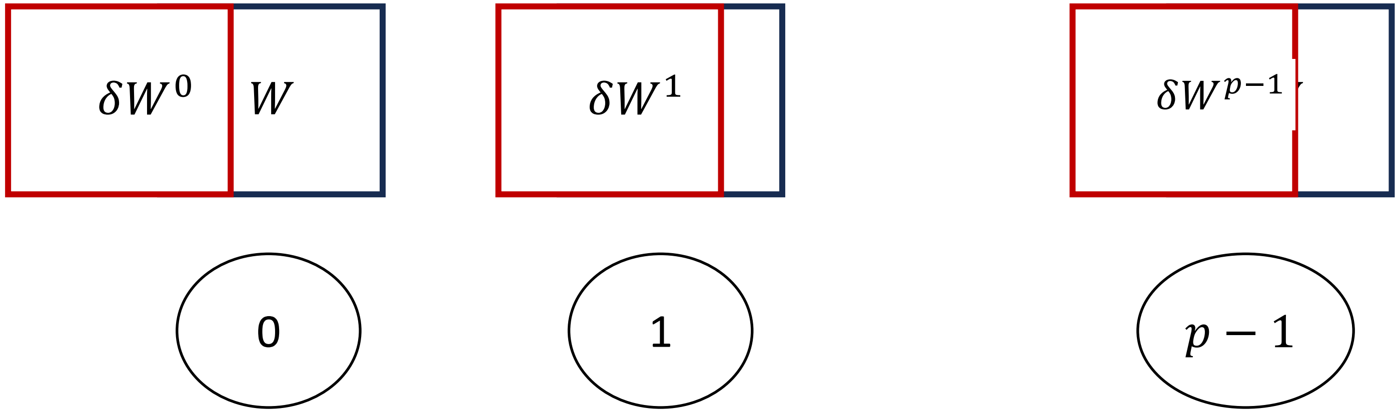
- In each Iteration:
  - Forward/Backward Propagation



All Reduce Model

- In each Iteration:
  - Forward/Backward Propagation
  - Gradient Calculation

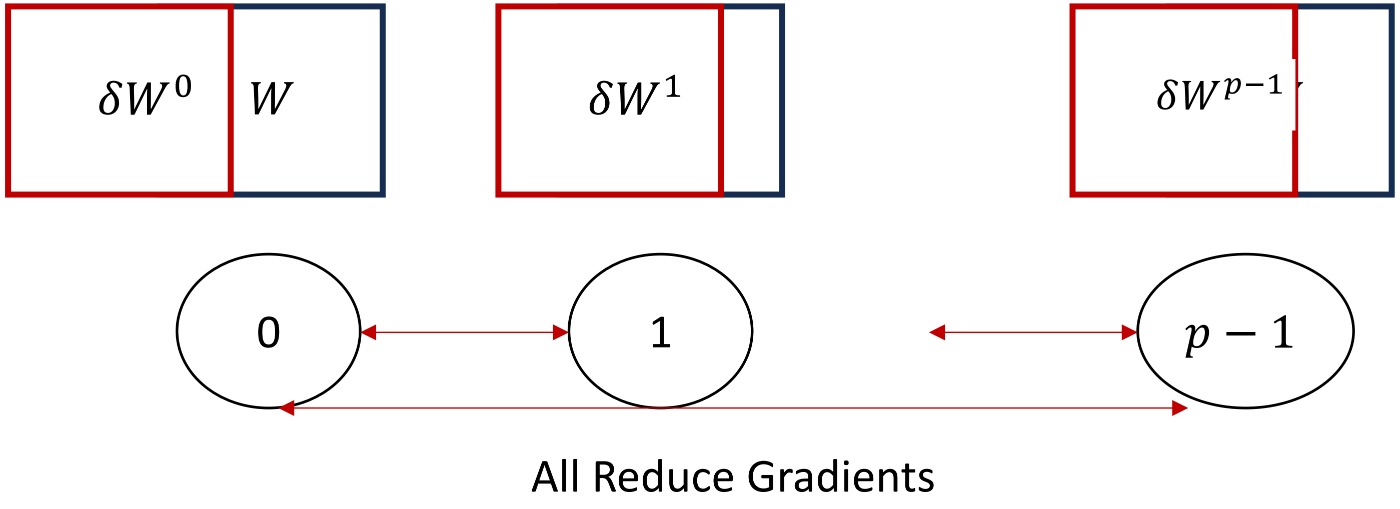
Data Parallel GPUs



All Reduce Model

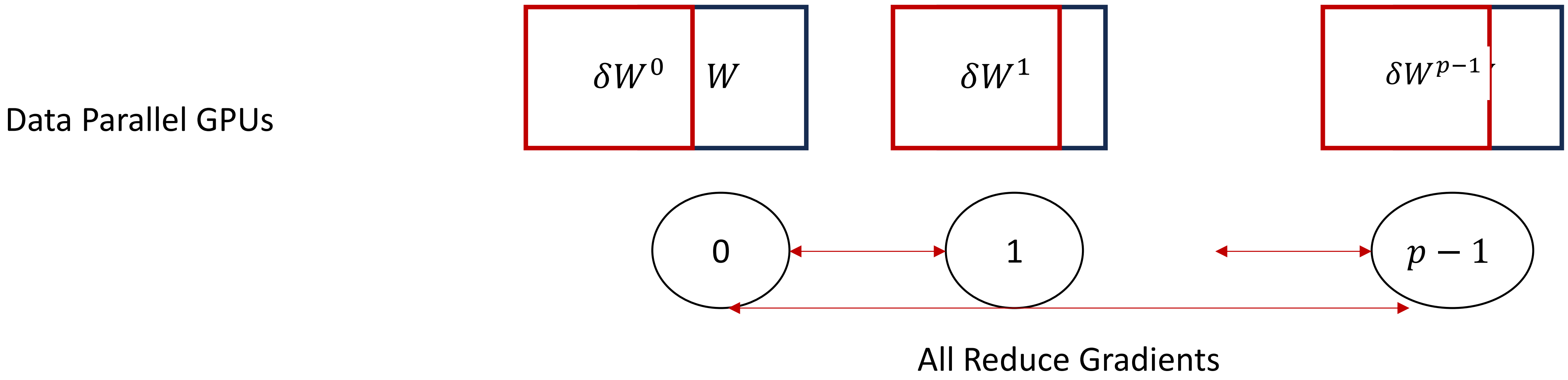
- In each Iteration:
  - Forward/Backward Propagation
  - Gradient Calculation
  - Each processor collects gradients from all the other processors.

Data Parallel GPUs



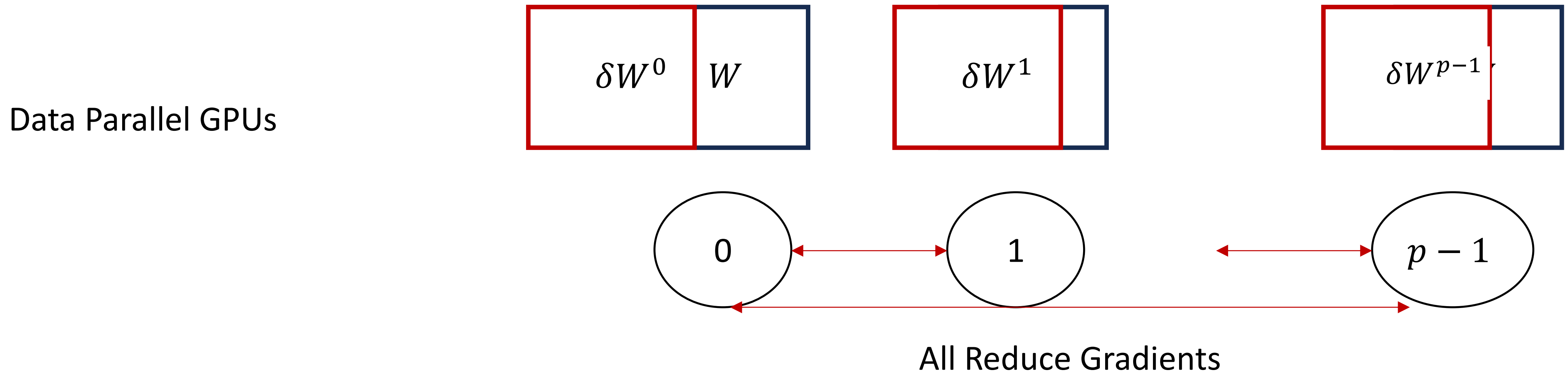
All Reduce Model

- In each Iteration:
  - Forward/Backward Propagation
  - Gradient Calculation
  - Each processor collects gradients from all the other processors.



All Reduce Model

- In each Iteration:
  - Forward/Backward Propagation
  - Gradient Calculation
  - MPI_ALLREDUCE(δW,AVG)



MPI_AllReduce

- MPI_AllReduce(inbuf, resultbuf, size)

- Same as MPI_Reduce() but stores the results in all the processors

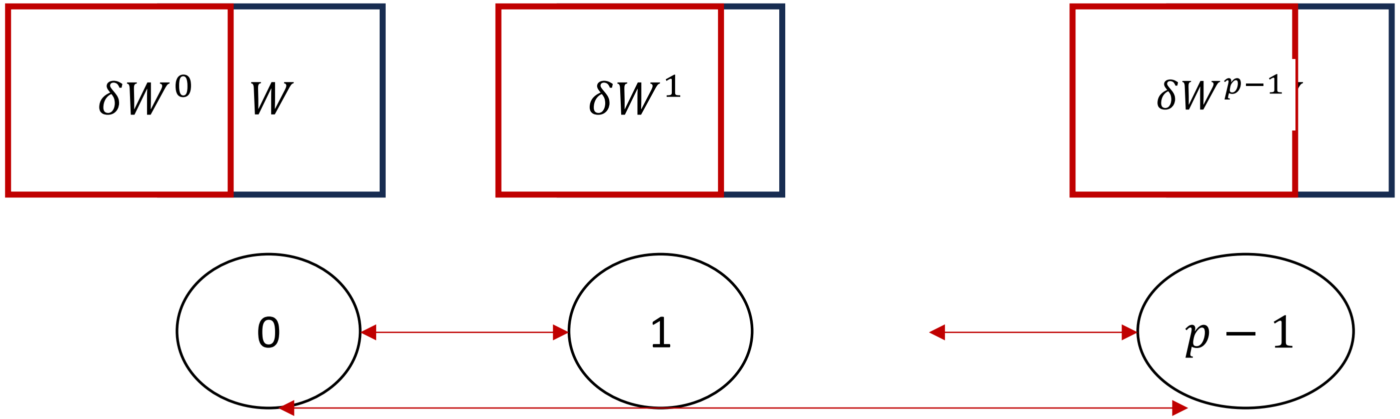
- Equivalent to:
  - MPI_Reduce(inbuf, ibsize, resultbuf, rbsize, Aggr_op, root)
  - MPI_Broadcast(resultbuf, rbsize, root)

All Reduce Model

- In each Iteration:
  - Forward/Backward Propagation
  - Gradient Calculation
  - MPI_ALLREDUCE(δW,AVG)

- Complexity: ??

Data Parallel GPUs

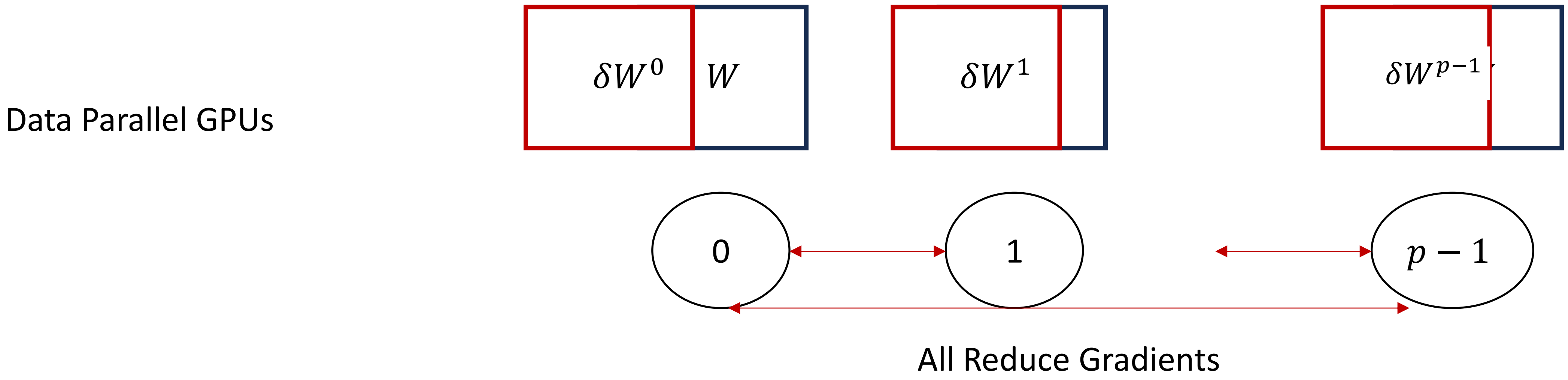


All Reduce Gradients

All Reduce Model

- In each Iteration:
  - Forward/Backward Propagation
  - Gradient Calculation
  - MPI_ALLREDUCE(δW,AVG)

- Complexity: $O(N \log p)$



3D Parallelism

- In practice, all three parallelisms are combined
- Data + Pipeline + Tensor = 3D parallelism
- DeepSpeed: https://github.com/microsoft/DeepSpeed/tree/master?tab=readme-ov-file

Next Class

- 11/25 Final exam

Thank You

- Questions?

- Email: sanmukh.kuppannagari@case.edu