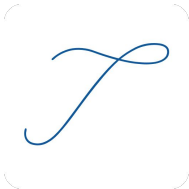
Multi-GPU fine-tuning

With DDP and FSDP

Trelis Research

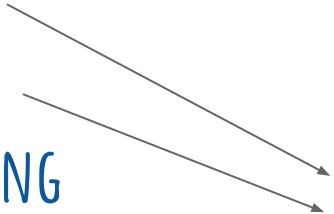
Overview

1. What kind of multi-GPU training should I do?
2. Figuring out VRAM requirements.
3. Training types:
   a. Model parallel (MP)
   b. Distributed data parallel (DDP)
   c. Fully sharded data parallel (FSDP)
4. Code modifications for each approach.
5. Demo for a) tinyllama in DDP and b) code llama 34B in FSDP



What GPU training to do?

1. What is my model size?
2. What accuracy do I want?
   A. Best - Full-fine tuning
   B. Good - Lora
   C. Ok - Quantized Lora
3. Do I want to:
   A. Minimize number of gpus / cost
   B. Maximise training speed



VRAM Requirements

GPU setup

(Mp, dDP, FSDP)

Effect of *model size* on VRAM requirements!

8 bits = 1 byte

16 bit number => 2 bytes

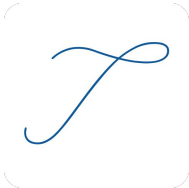
VRAM requirements!

= Model parameters (e.g. 7B)

\+ Gradients

\+ Optimizer states

\+ activations

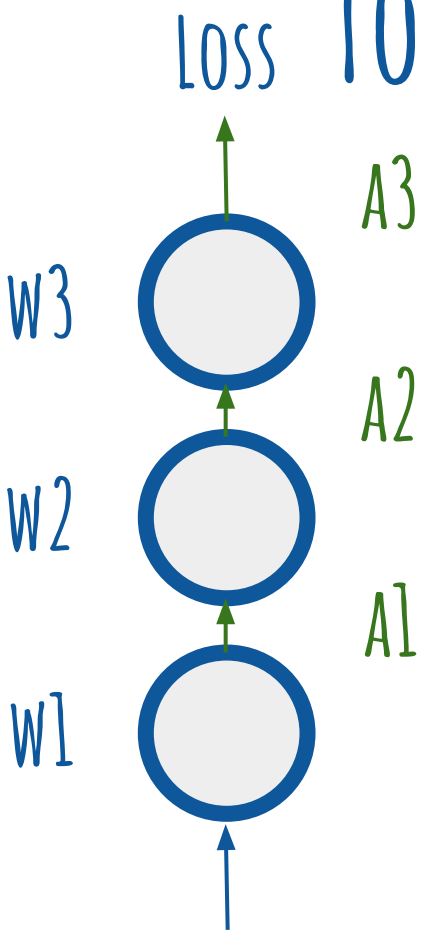


Toy model (forward)

Three "layers"

One weight per layer!

Toy model (forward)



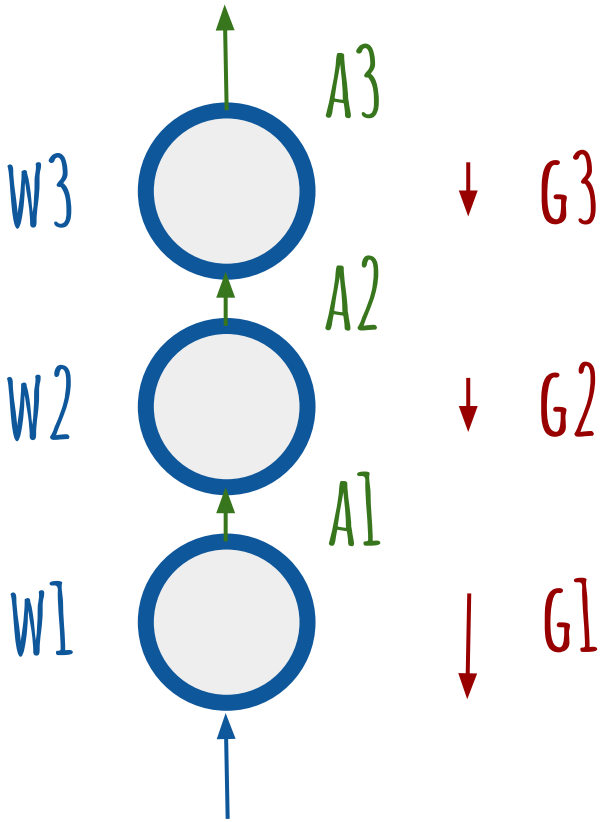
Three "layers"

One weight per layer!

output/Activation = weight * input

Loss (A3) = output - reference

Toy model (backward)



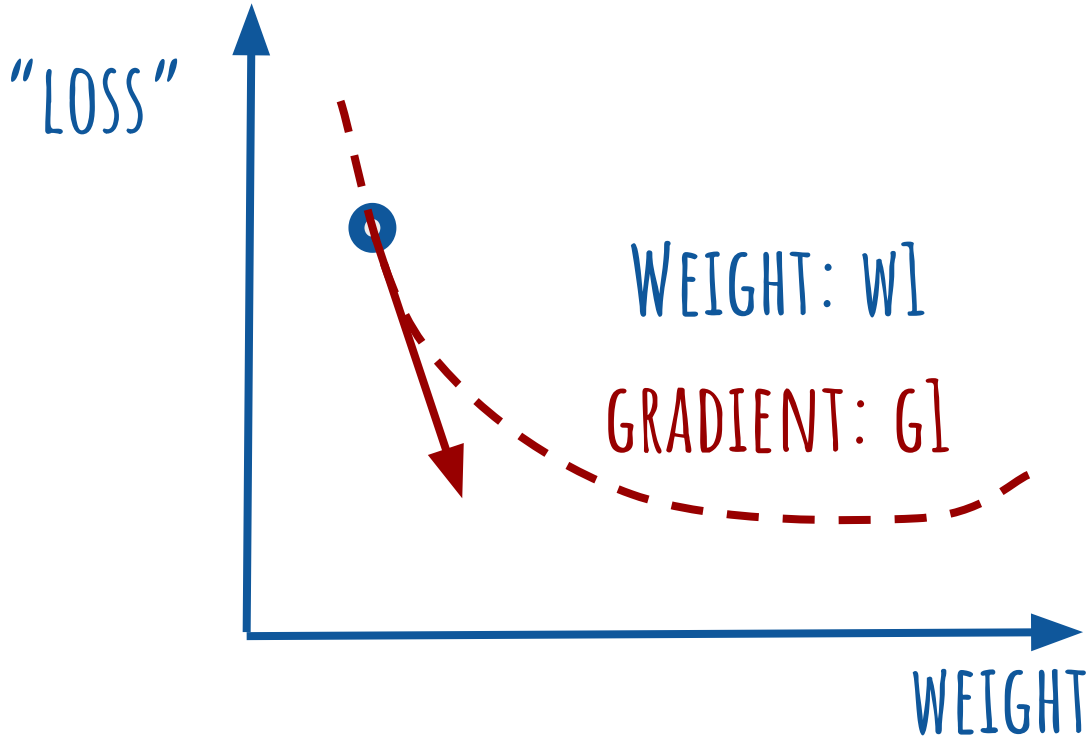
Three "layers"

One weight per layer!

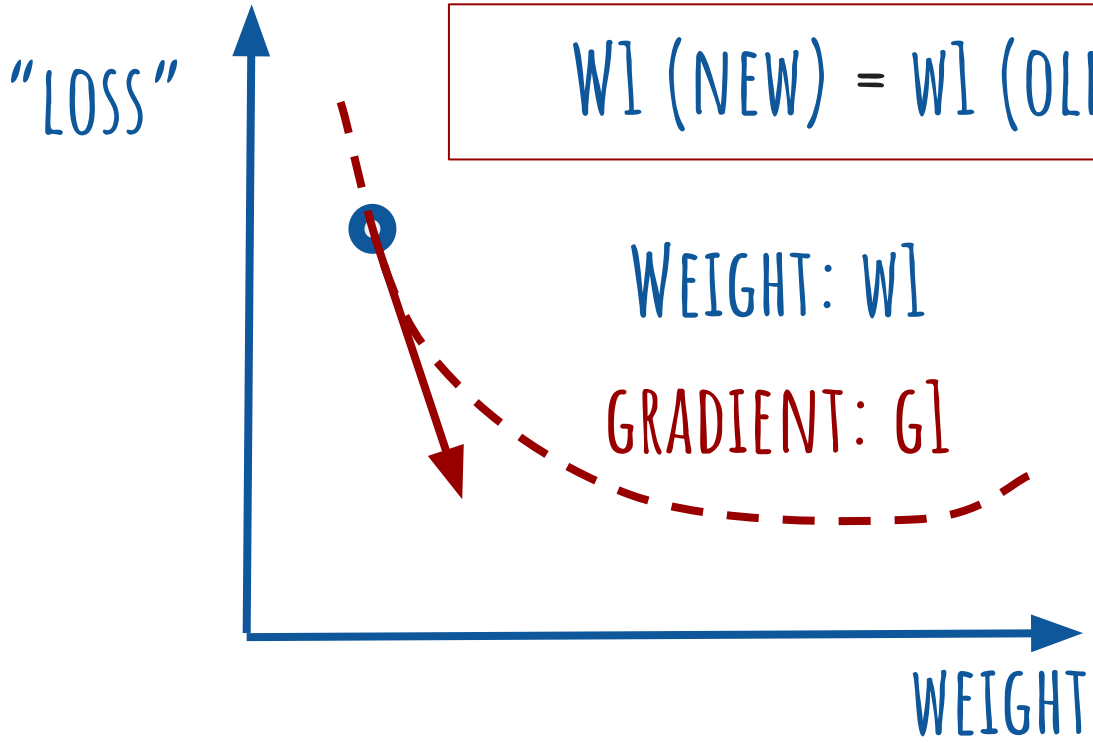
Activation = weight * input

Gradients (chain rule)

Toy model

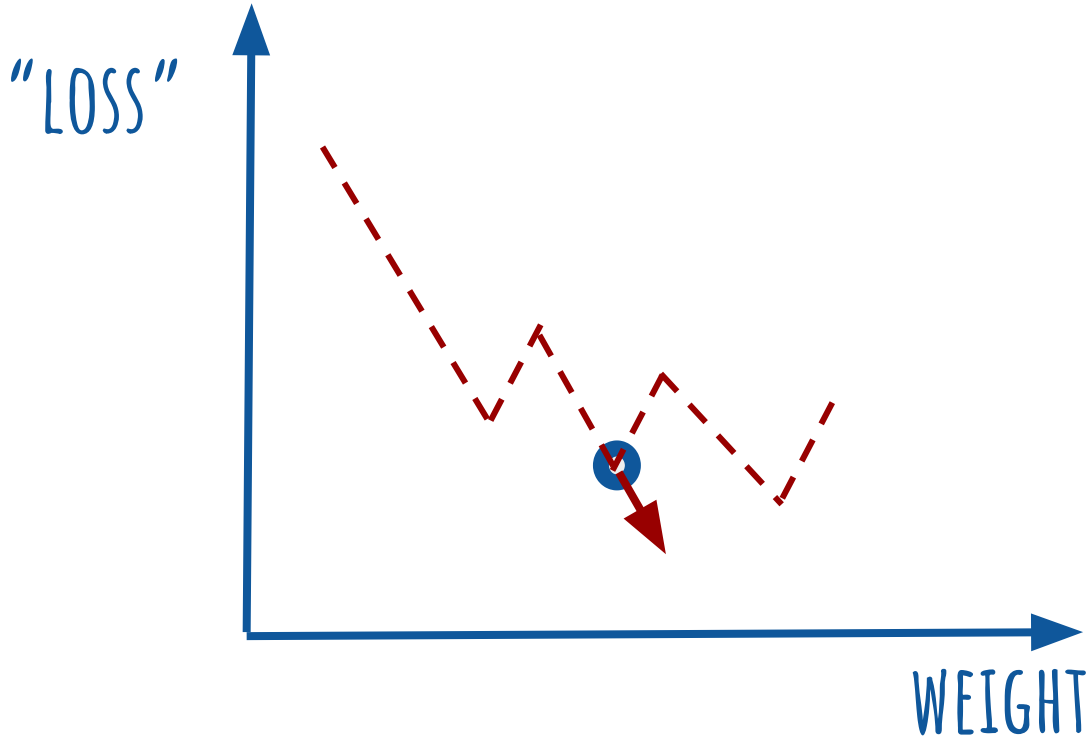


TOY OPTIMIZER

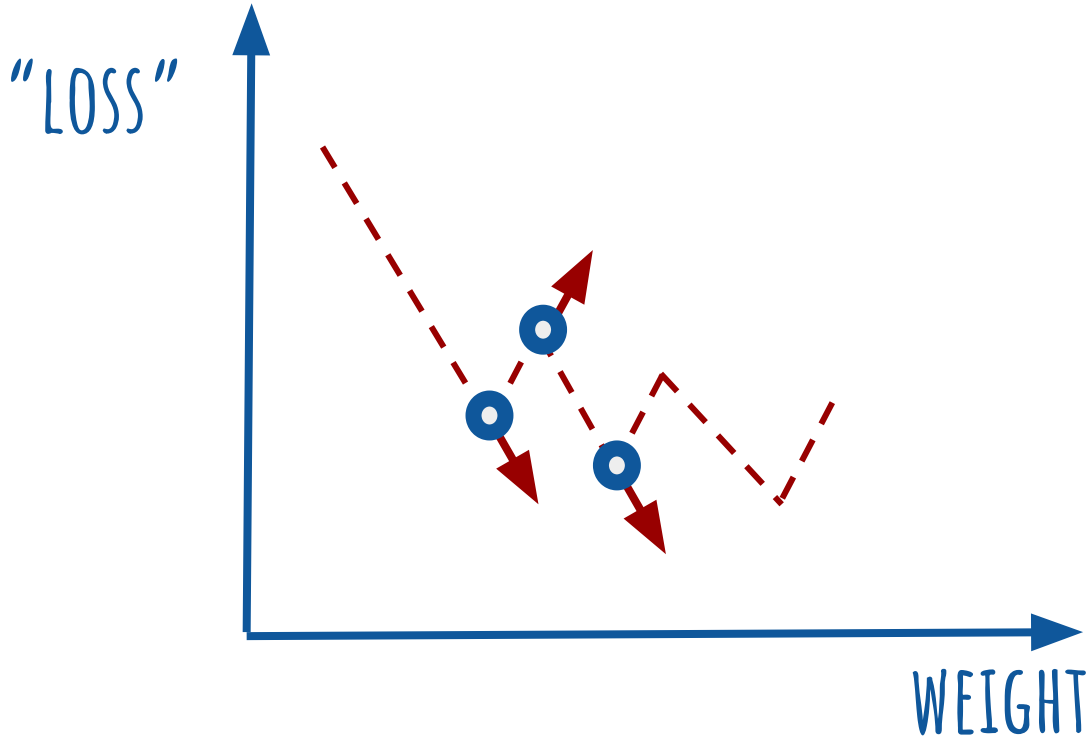


$$W1\ (NEW) = W1\ (OLD) + DELTA * G1$$

Don't just use the gradient

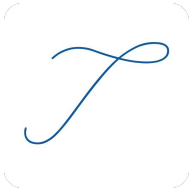


Use a historical average

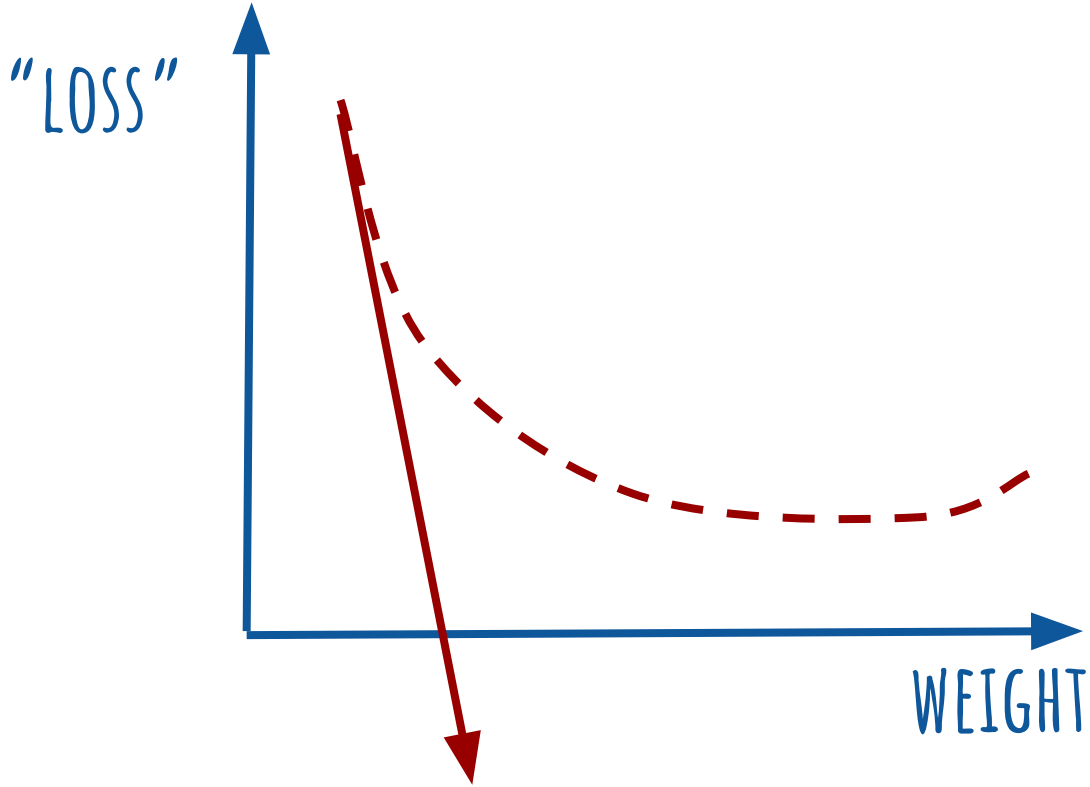


Consider a historical average

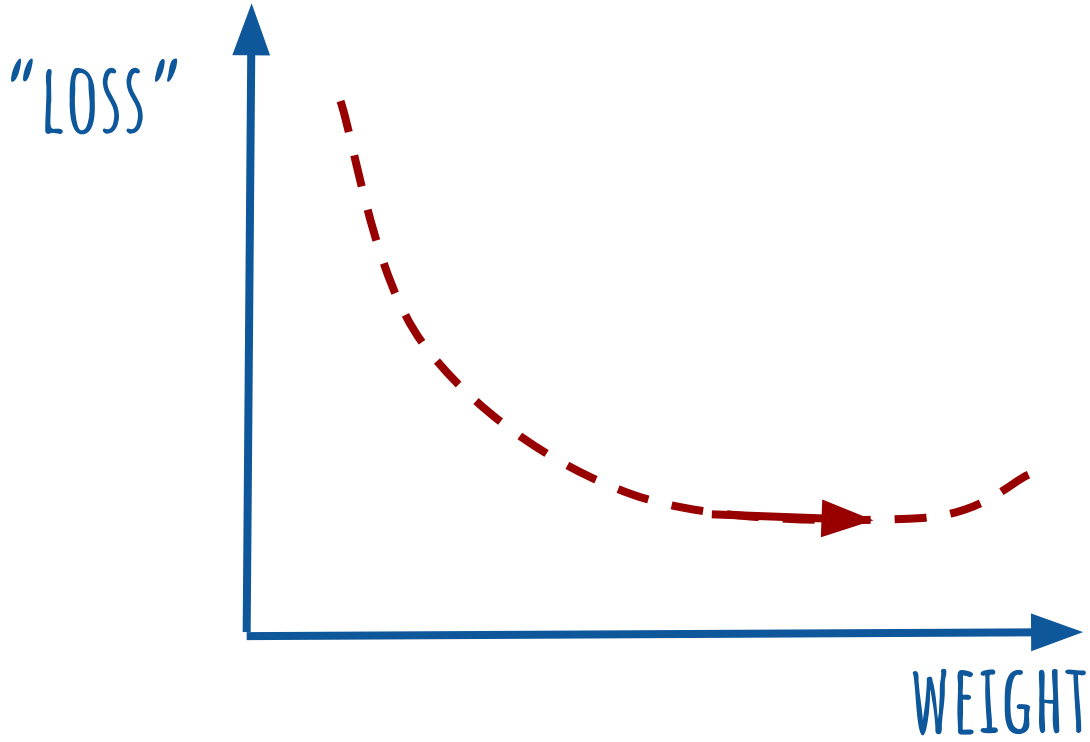
$$G = 1\% \cdot G(\text{new}) + 99\% \cdot G(\text{old}))$$



Dampen large gradients



AMPLIFY SMALL GRADIENTS



Amplify small, dampen big

Consider the historical variance

$$G^2 = 0.01\% \cdot G^2(\text{new}) + 99.9\% \cdot G^2(\text{old})$$

Big variance -> dampen

Small variance -> accelerate

Actual optimizer (Adam)

1. Average the gradients

$$G_M = 1\% \cdot G(\text{new}) + 99\% \cdot G(\text{old}).$$

2. Use the variance to dampen/amplify

$$G_V^2 = 0.01\% \cdot G^2(\text{new}) + 99.9\% \cdot G^2(\text{old}).$$

Actual optimizer (Adam)

For every trainable model parameter:

- "Gradient history" (momentum term)
- "Gradient Variance history" (adaptive term)

Model + grads + optimizer = 16 bytes

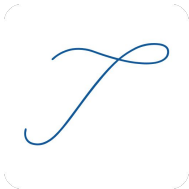
For every trainable model parameter:

- Parameter
- Gradient
- momentum term
- adaptive term

Model + Grads + Optimizer = 16 bytes

For every trainable model parameter:

- Parameter [16 bits = 2 bytes]
- Gradient [16 bits = 2 bytes]
- momentum term [32 bits = 4 bytes]
- adaptive term [32 bits = 4 bytes]
- Parameter in 32 bits [4 bytes]



Model (16bit)

Parameter [16 bits = 2 bytes]

+Gradient [16 bits = 2 bytes] x 1/64

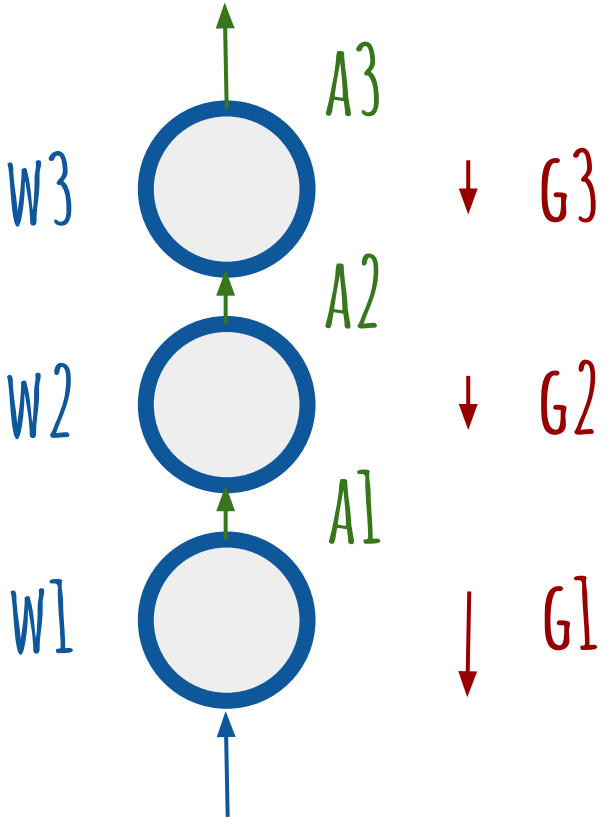
+momentum term [32 bits = 4 bytes] x 1/64

+adaptive term [32 bits = 4 bytes] x 1/64

+Parameter in 32 bits [4 bytes] x 1/64

+ activations!!! (scales with context and batch size)

Activations size...



Activation = weight * input

(Input = batch size x seq length)

GPU setup principle

Your GPUs must fit the model + grads + optimizer

\+ At least one batch!

Tip:

1. Start with enough GPUs to fit model + grads + optimizer
2. Start with batch = 1 and increase til OOM.

What GPU training to do?

1. What is my model size?
2. What accuracy do I want?
   a. Best - Full-fine tuning
   b. Good - Lora
   c. Ok - Quantized Lora
3. Do I want to:
   a. Minimize number of gpus / cost
   b. Maximise training speed

VRAM Requirements

GPU setup

(Mp, dDP, FSDP)

Model (4bit)

Parameter [4 bits = 0.5 bytes]

+Gradient [16 bits = 2 bytes] x 1/64

+Momentum term [32 bits = 4 bytes] x 1/64

+Adaptive term [32 bits = 4 bytes] x 1/64

+Parameter in 32 bits [4 bytes] x 1/64

+ activations!!! (scales with context and batch size)

Freezing the core model + Training Low rank adapters (LoRA)

LoRA savings

$$(2 * 8 * 1024) / (1024 * 1024)$$

= 1/64 parameters to train!!!

Model + grads + optimizer w/ LoRA

Parameter [16 bits = 2 bytes]

+Gradient [16 bits = 2 bytes] x 1/64

+momentum term [32 bits = 4 bytes] x 1/64

+adaptive term [32 bits = 4 bytes] x 1/64

+Parameter in 32 bits [4 bytes] x 1/64

+ activations

Model + optimizer w/ LoRA = 2 bytes

Parameter [16 bits = 2 bytes]

.

.

.

.

+ activations

Model(4bit)+optim.+LoRA = 0.5 bytes

Parameter [5 bits = 0.5 bytes]

.

.

.

.

\+ activations

What GPU training to do?

1. What is my model size?
2. What accuracy do I want?
   a. Best - Full-fine tuning
   b. Good - Lora
   c. Ok - Quantized Lora
3. Do I want to:
   a. Minimize number of gpus / cost
   b. Maximise training speed

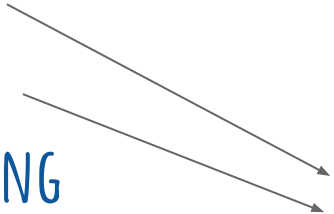
VRAM Requirements

GPU setup

(Mp, dDP, FSDP)

What GPU training to do?

1. What is my model size?
2. What accuracy do I want?
   A. Best - Full-fine tuning
   B. Good - Lora
   C. Ok - Quantized Lora
3. GPU Setup: Do I want to:
   A. Minimize number of gpus / cost
   B. Maximise training speed



VRAM Requirements

GPU setup

(Mp, dDP, FSDP)

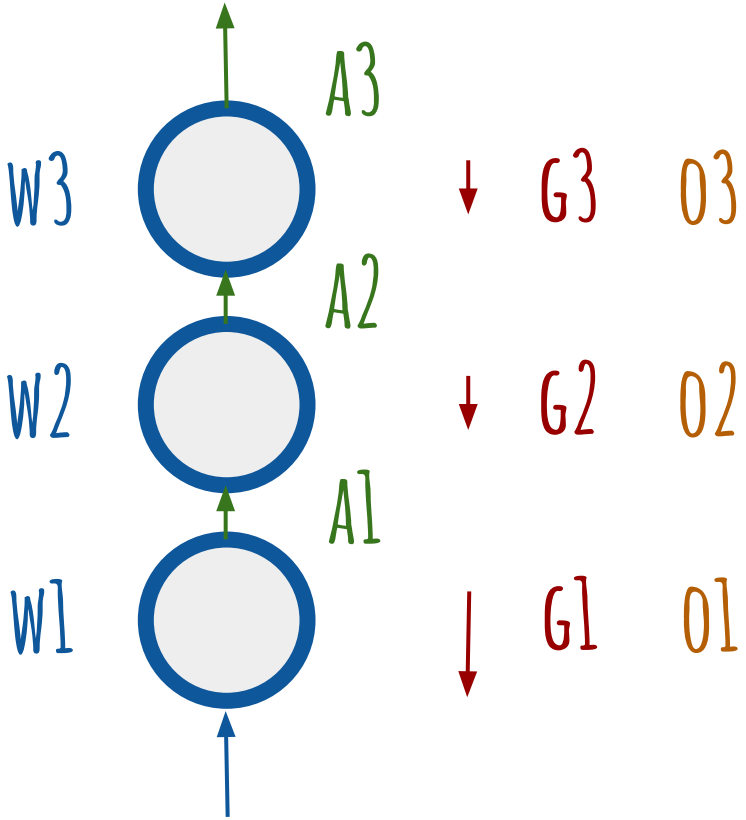
GPU Setup

1. You can always do *model parallel (MP)*

2. Does model fit on a single GPU?

Yes (single GPU)	No (need multiple GPUs)
More speed with	More speed with
Distributed data parallel (DDP)	*Fully sharded data parallel (FSDP)*

Toy model

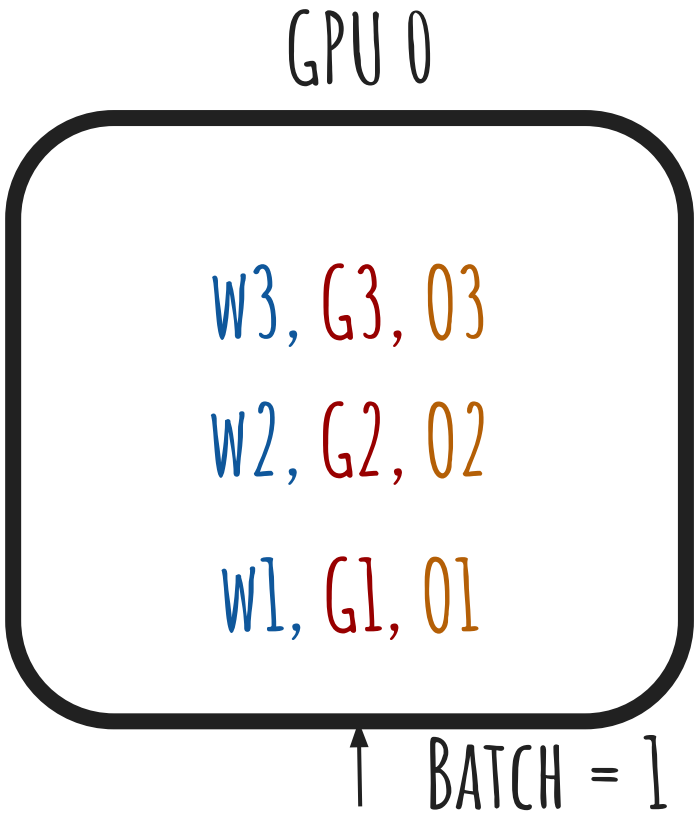


Weights

gradients

Optimizer states

If Model fits on single GPU



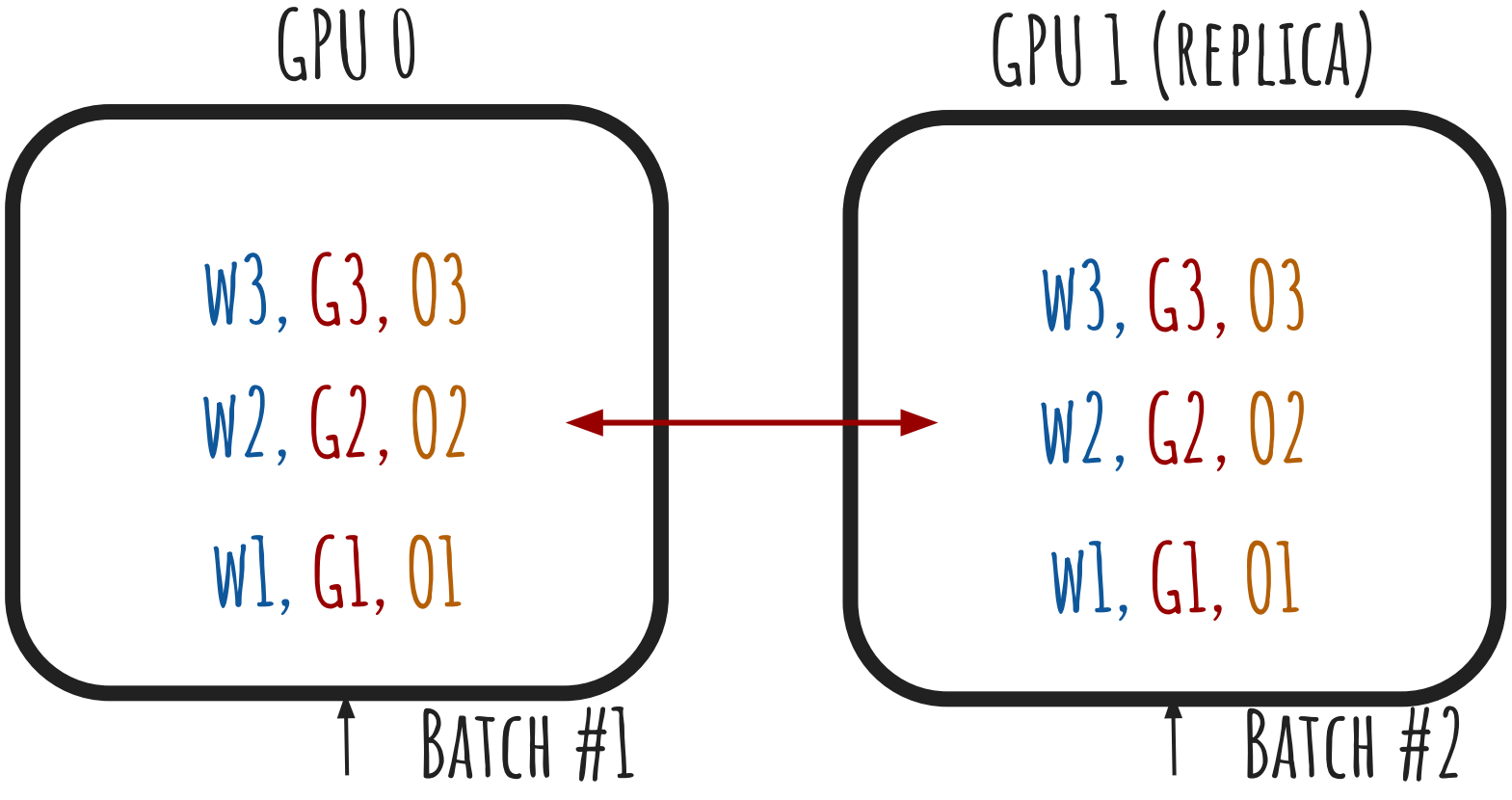
Weights

Gradients

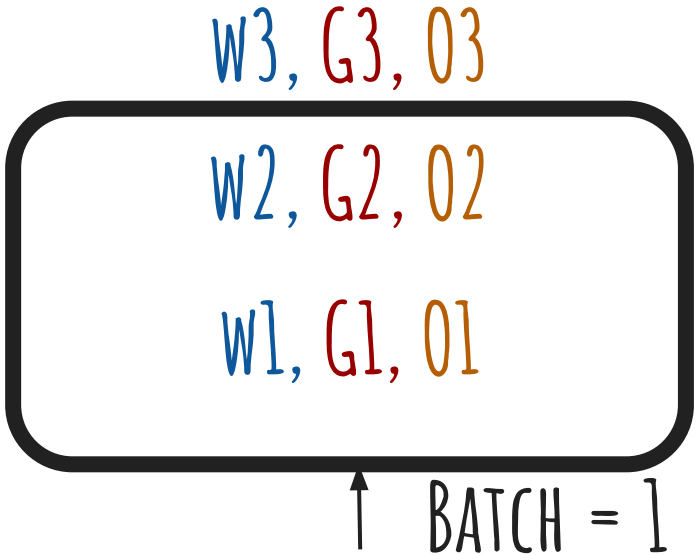
Optimizer states

https://pytorch.org/tutorials/intermediate/ddp_tutorial.html

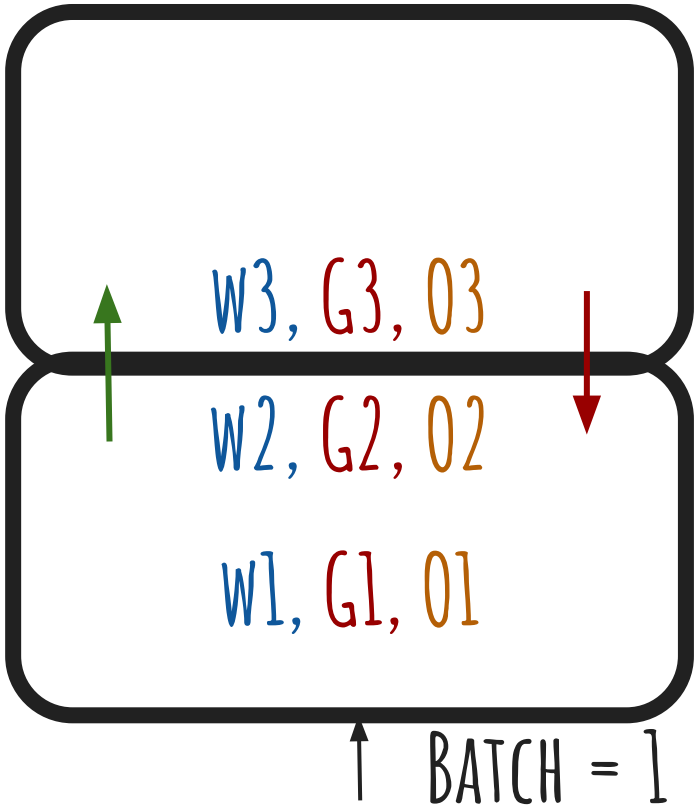
SINGLE GPU -> DATA PARALLEL



Model doesn't fit



Naive model parallel



GPU 1

GPU 0

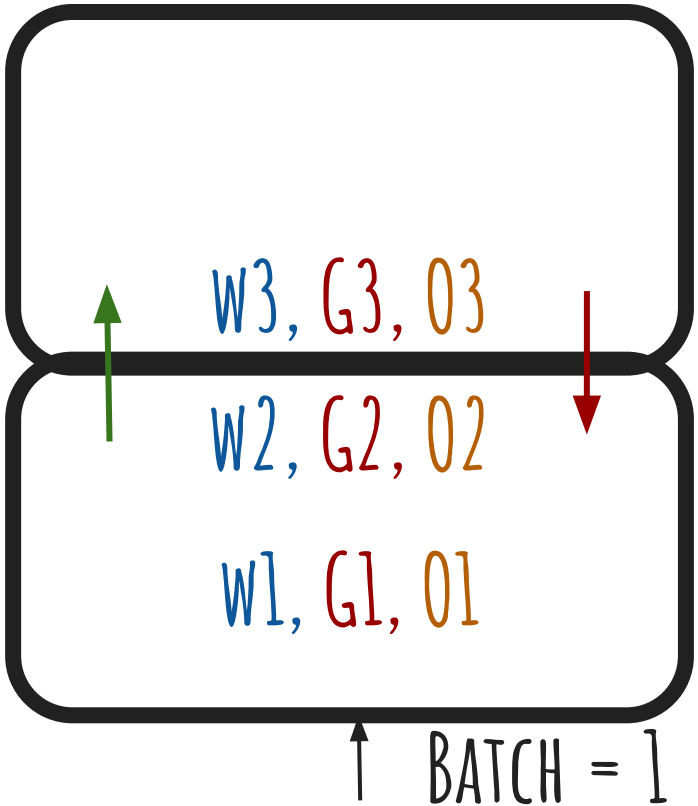
Forward pass:

pass activations

Back-prop:

pass gradient info

Naive model parallel



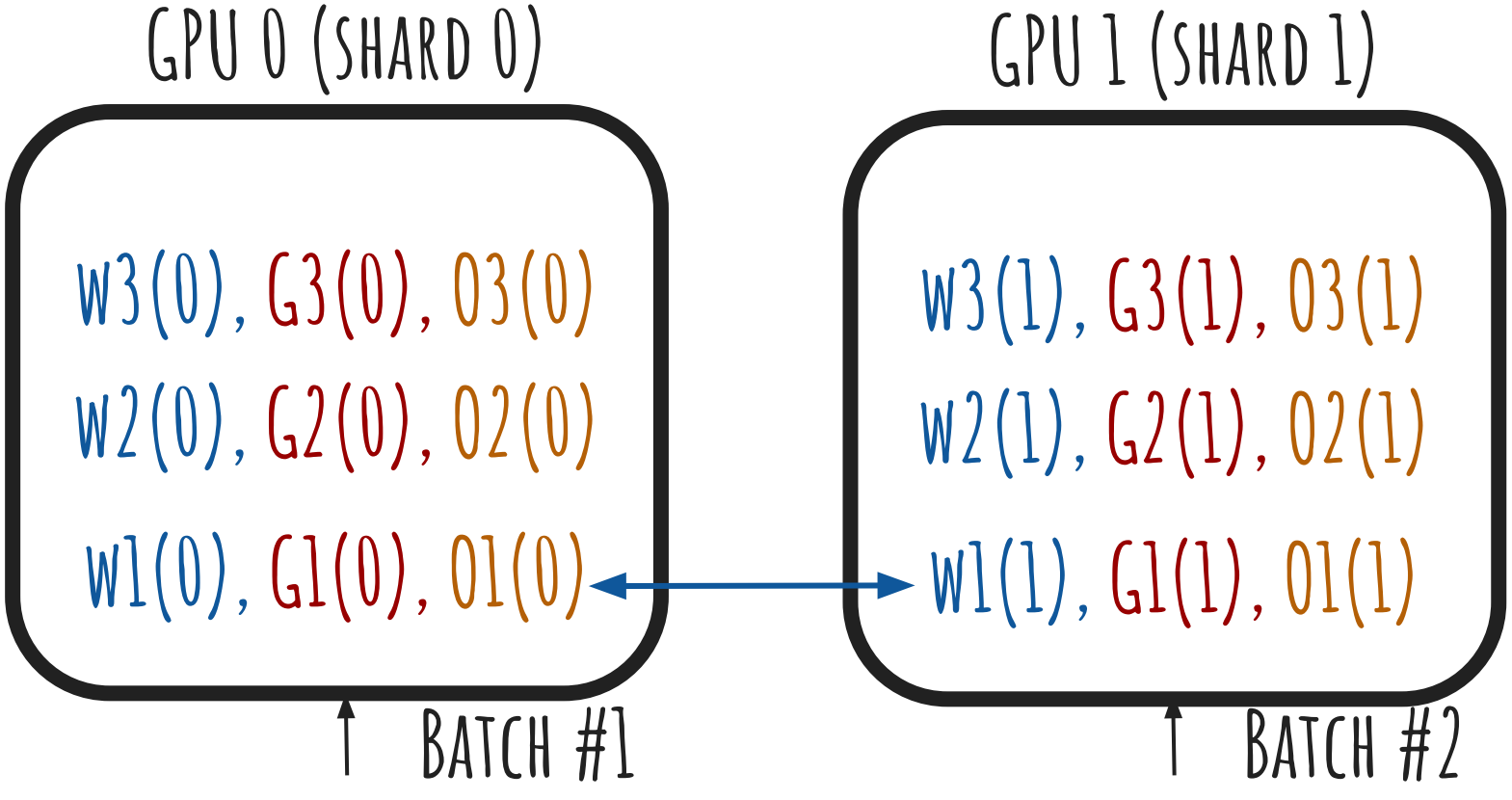
Problem:

~1 GPU active at a time

Bad "utilization"

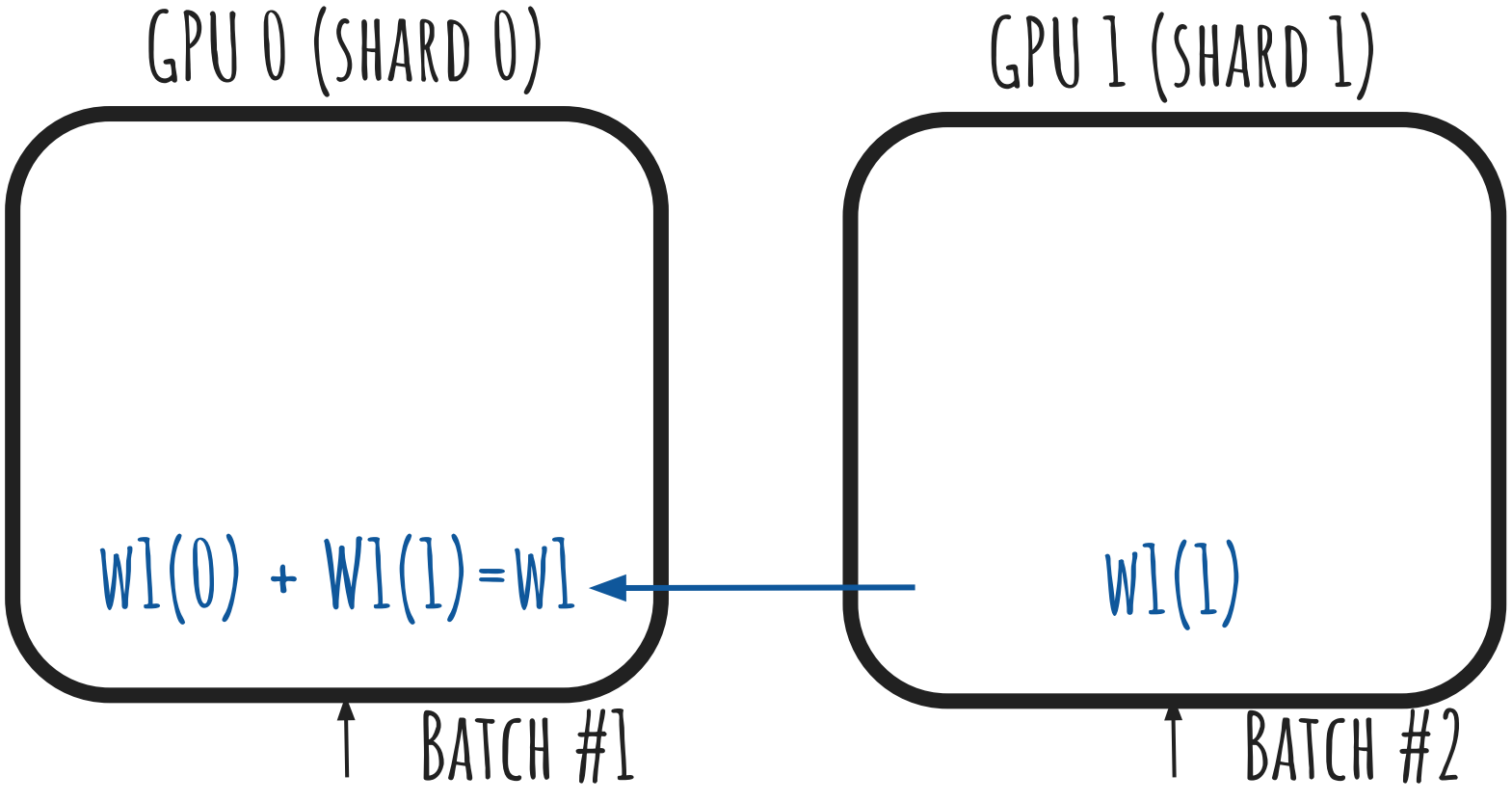
https://pytorch.org/tutorials/intermediate/FSDP_tutorial.html

FORWARD PASS: FSDP



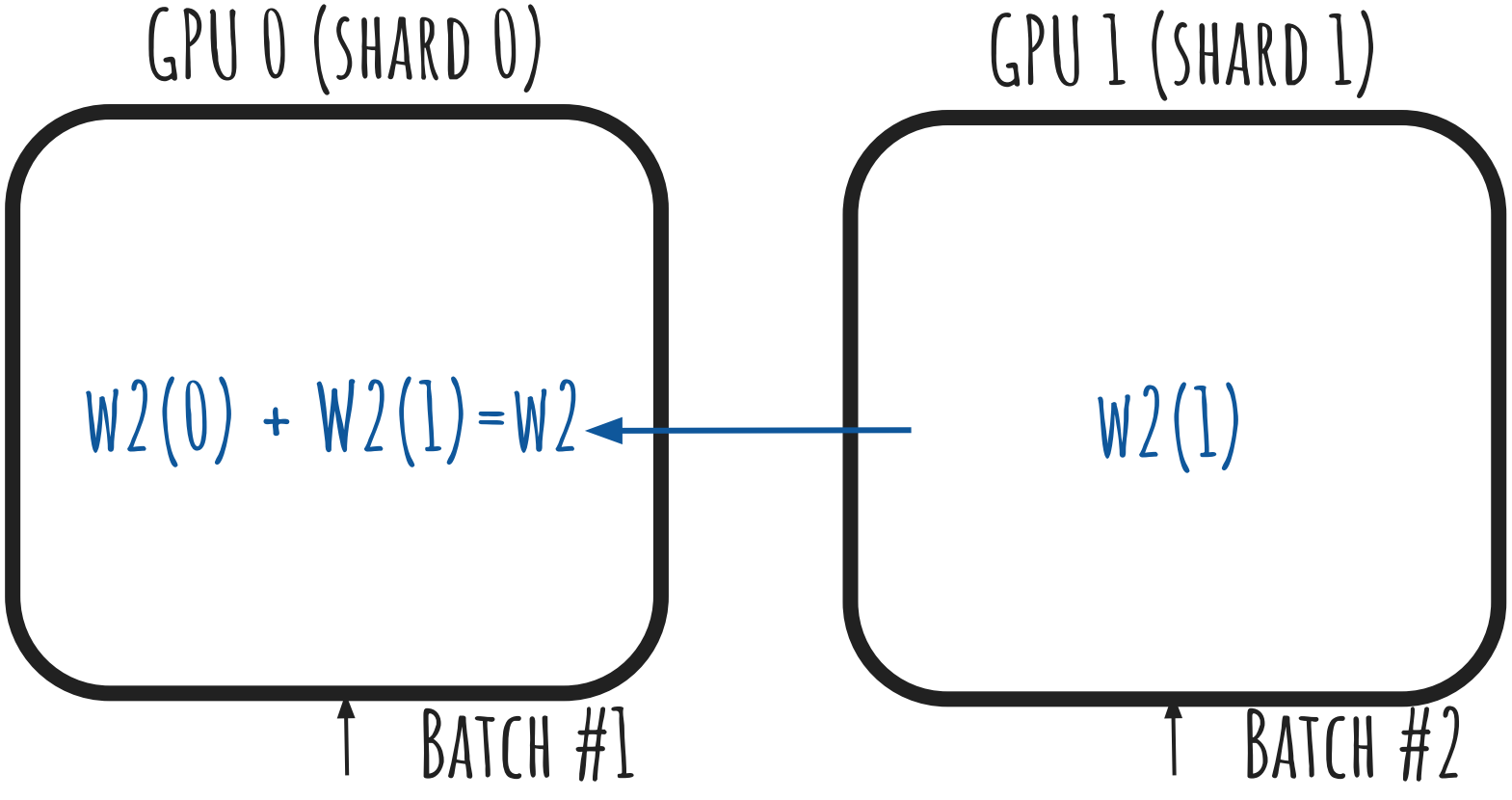
https://pytorch.org/tutorials/intermediate/FSDP_tutorial.html

FORWARD PASS: FSDP



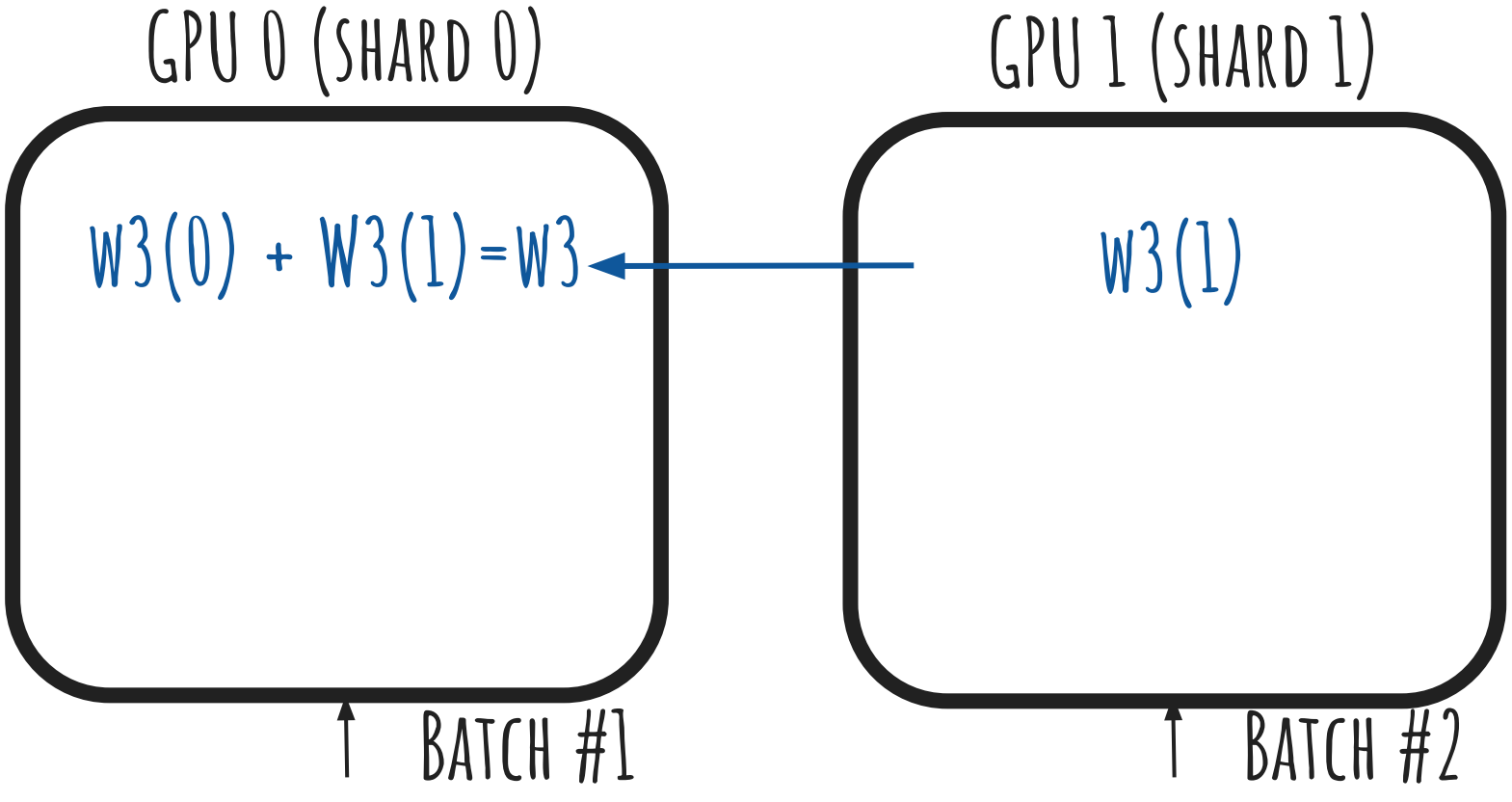
https://pytorch.org/tutorials/intermediate/FSDP_tutorial.html

FORWARD PASS: FSDP



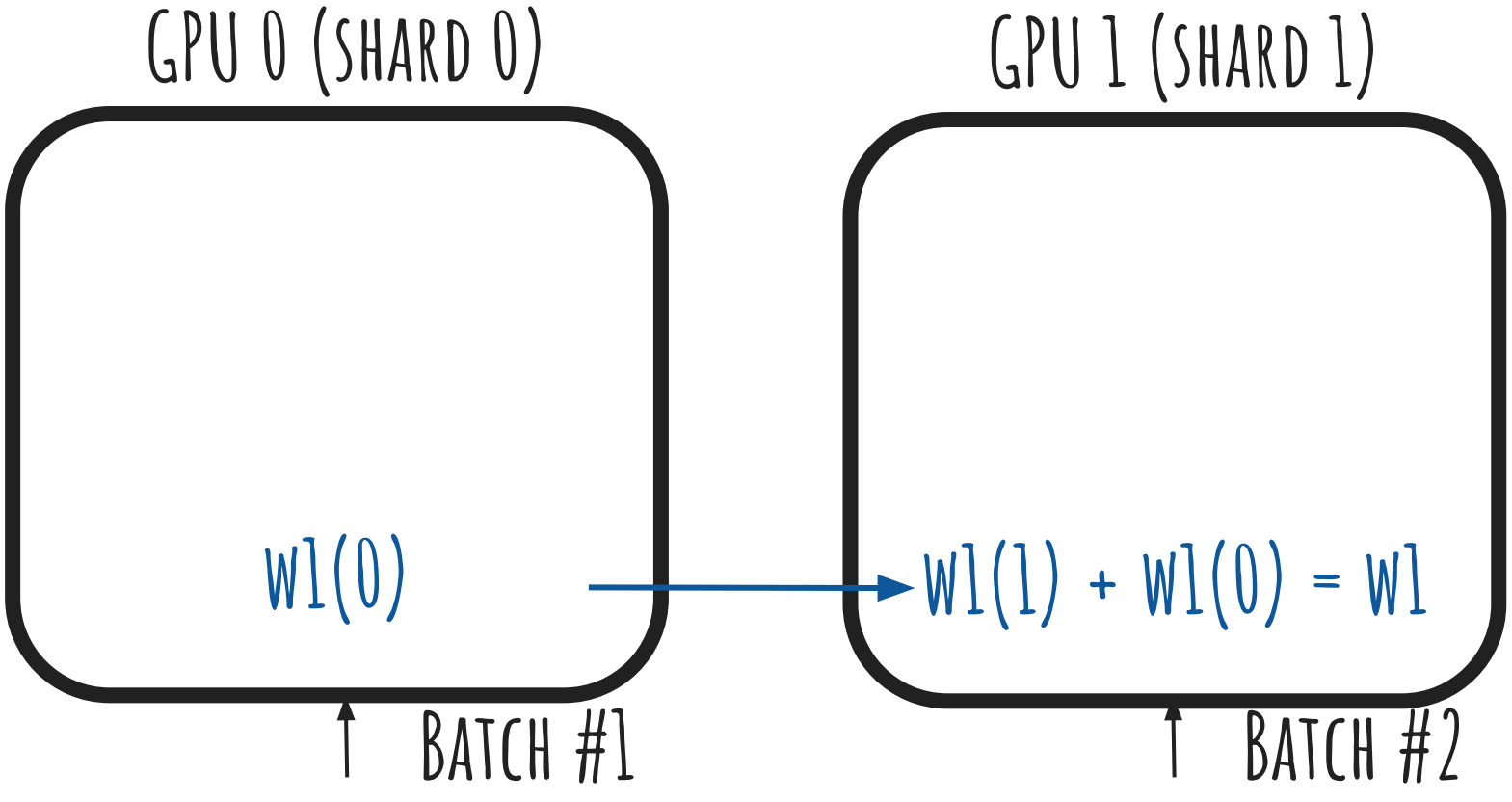
https://pytorch.org/tutorials/intermediate/FSDP_tutorial.html

FORWARD PASS: FSDP



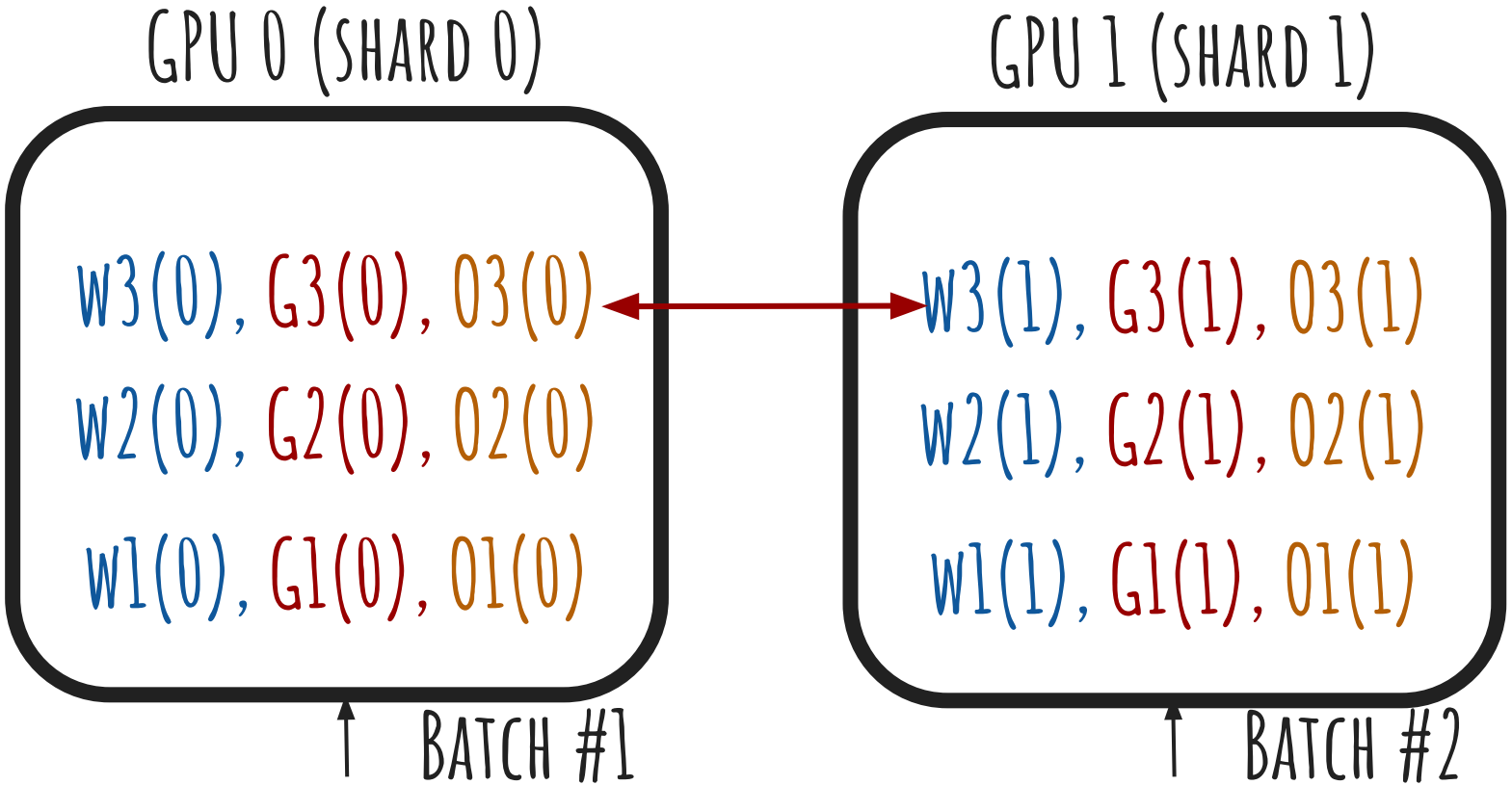
https://pytorch.org/tutorials/intermediate/FSDP_tutorial.html

FORWARD PASS: FSDP



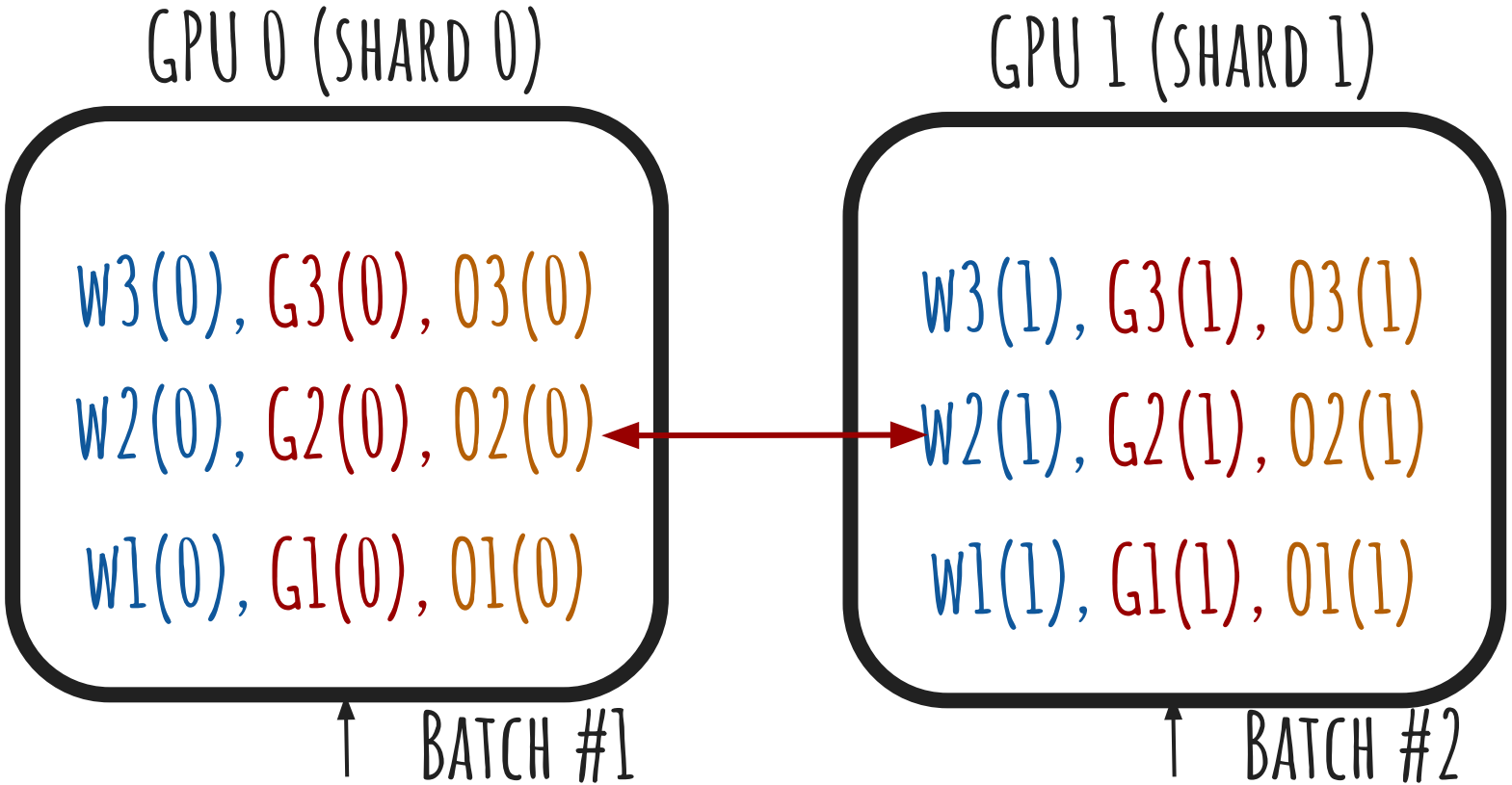
https://pytorch.org/tutorials/intermediate/FSDP_tutorial.html

BACKWARD PROPAGATION: FSDP



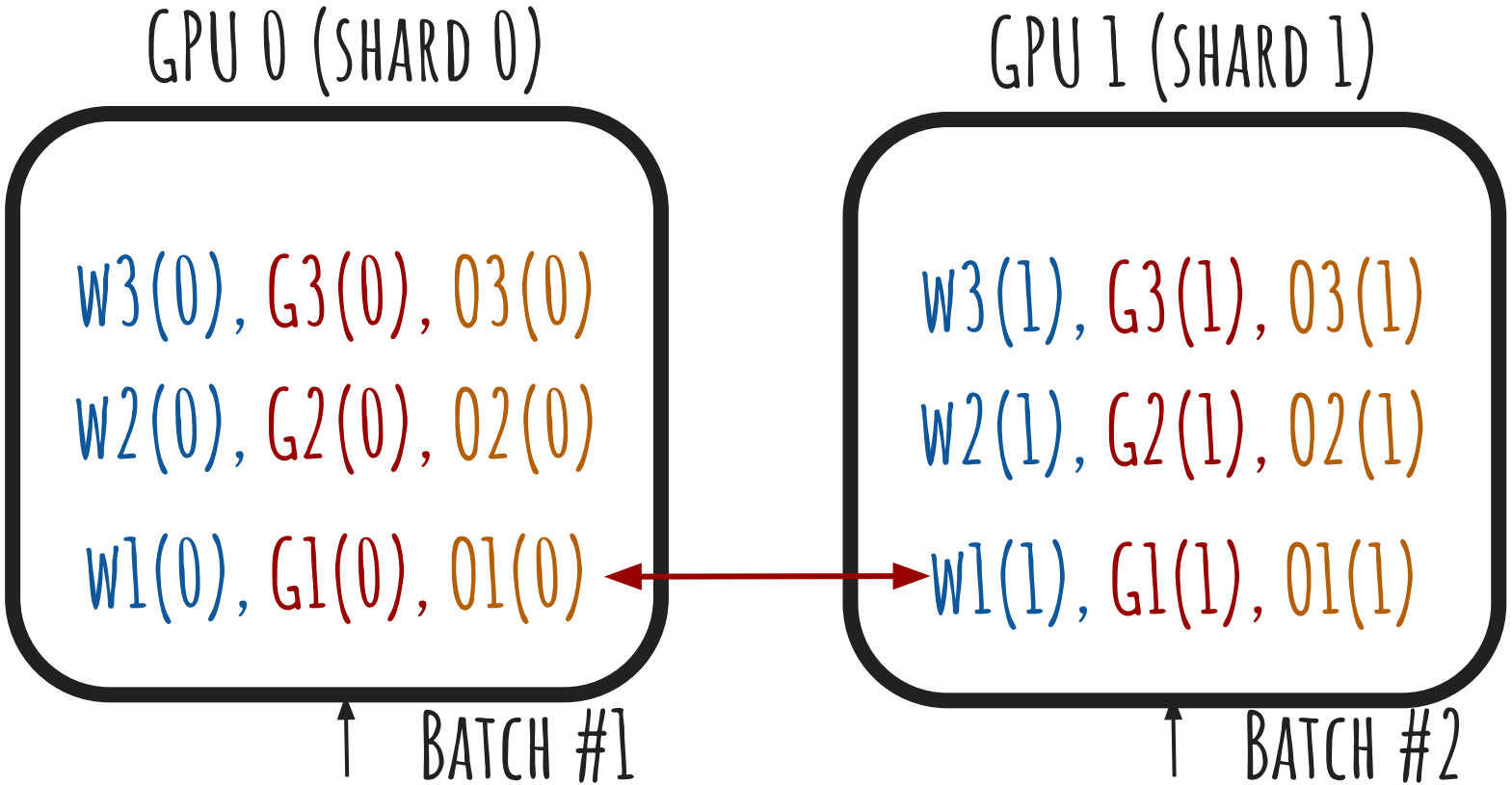
https://pytorch.org/tutorials/intermediate/FSDP_tutorial.html

BACKWARD PROPAGATION: FSDP



https://pytorch.org/tutorials/intermediate/FSDP_tutorial.html

BACKWARD PROPAGATION: FSDP



Comparing approaches

Model Parallel (MP)

Low communication overhead

Low utilization for multi-GPU

Distributed data parallel (DDP)

Modest communication overhead

(gradients)

Fully sharded data parallel (FSDP)

High communication overhead

(weights + gradients)

GPU Setup

1. You can always do *model parallel (MP)*

2. Does model fit on a single GPU?

Yes (single GPU)	No (need multiple GPUs)
More speed with	More speed with
Distributed data parallel (DDP)	*Fully sharded data parallel (FSDP)*

A note on deepspeed (microsoft)

Deepspeed stage 1 <-> pytorch FSDP

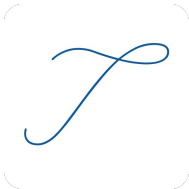
shard O1

Deepspeed stage 2 <-> pytorch FSDP

shard G1, O1

Deepspeed stage 3 <-> pytorch FSDP

shard W1, G1, O1

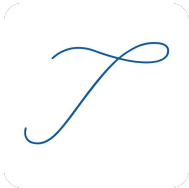


ACCELERATE

A WRAPPER AROUND LIBRARIES:

- SUPPORTS PYTORCH DDP + FSDP
- SUPPORTS DEEPSPEED

CONFIGURE WITH 'ACCELERATE CONFIG'



Results - TinyLlama 1.1B w/ LoRA

4x A6000 Ada (48GB)

	GPU Utilization	Training Time
Model Parallel	24%	3m53s
DDP	90%	59s

1024 rows, 1024 context, batch 16, GA 2

Results - CodeLlama 34B w/ LoRA

4x A6000 Ada (48GB)

	GPU Utilization	Training Time
Model Parallel	~20%	47 mins
FSDP	~100%	30 mins

1024 rows, 1024 context, batch 4, GA 4

K V CACHE

	key(“The”) . query (“quick”)	**The**
The	**quick**	

K V CACHE

	key(“The”) · query (“quick”)	key(“The”) · query (“brown”)	**The**
		key(“quick”) · query (“brown”)	**quick**
The	**quick**	**brown**	