Efficient LLMs

Yatin Nandwani
Research Scientist, IBM Research

Large Language Models: Introduction and Recent Advances

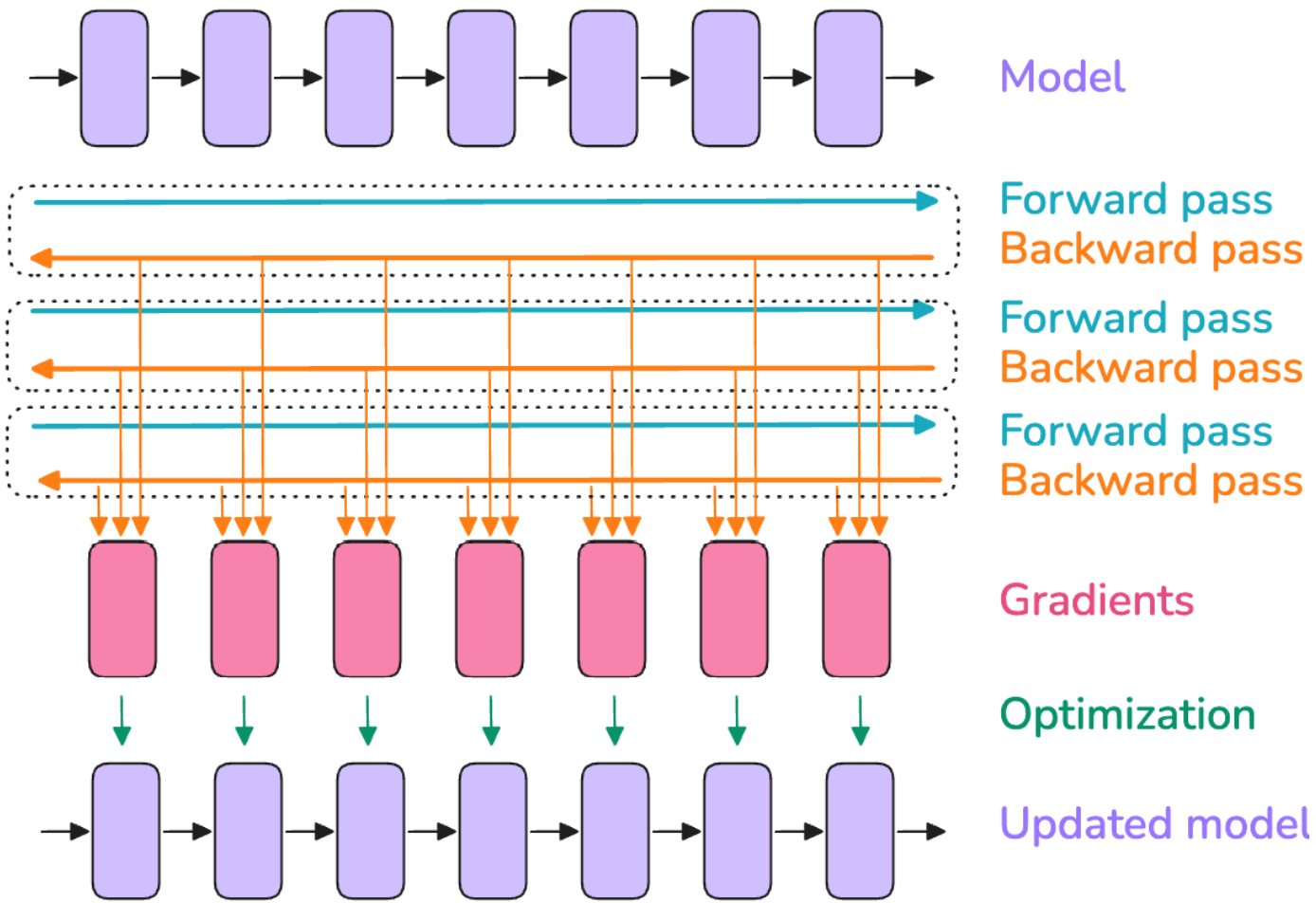
Semester 1,
2025-2026

ELL881 · AIL821

Recap

- How to train big models on big data?
- What's in the GPU memory during training?
  - Model params
  - Param gradients
  - Optim states
  - Activations
- What is the size of params / grads / optim states?
- What is the size of activations?
- How to reduce activation memory?
  - Activation re-computation *aka* Gradient checkpointing
- How to increase batch size?
  - Gradient accumulation (run fwd / bwd *k* times before optim.step())
- Can we parallelize grad. Accumulation?

How to increase batch size? – *Gradient Accumulation*



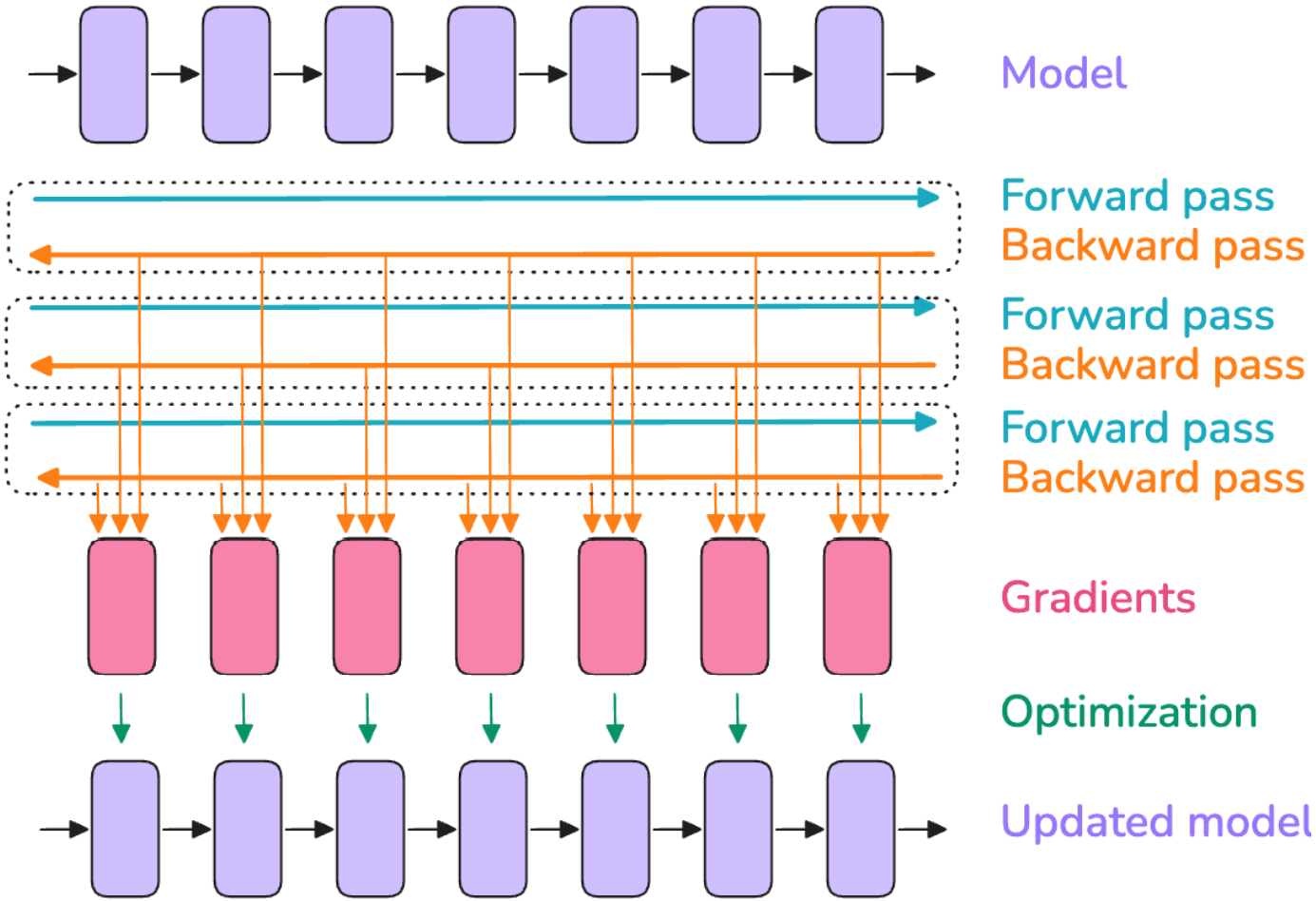
- Process smaller micro-batches sequentially
- $bs = gbs = mbs * grad\_acc$

Only one micro-batch's worth of activations needs to be kept in memory at a time

Requires multiple consecutive forward/backward passes per optimization step

How to speedup?

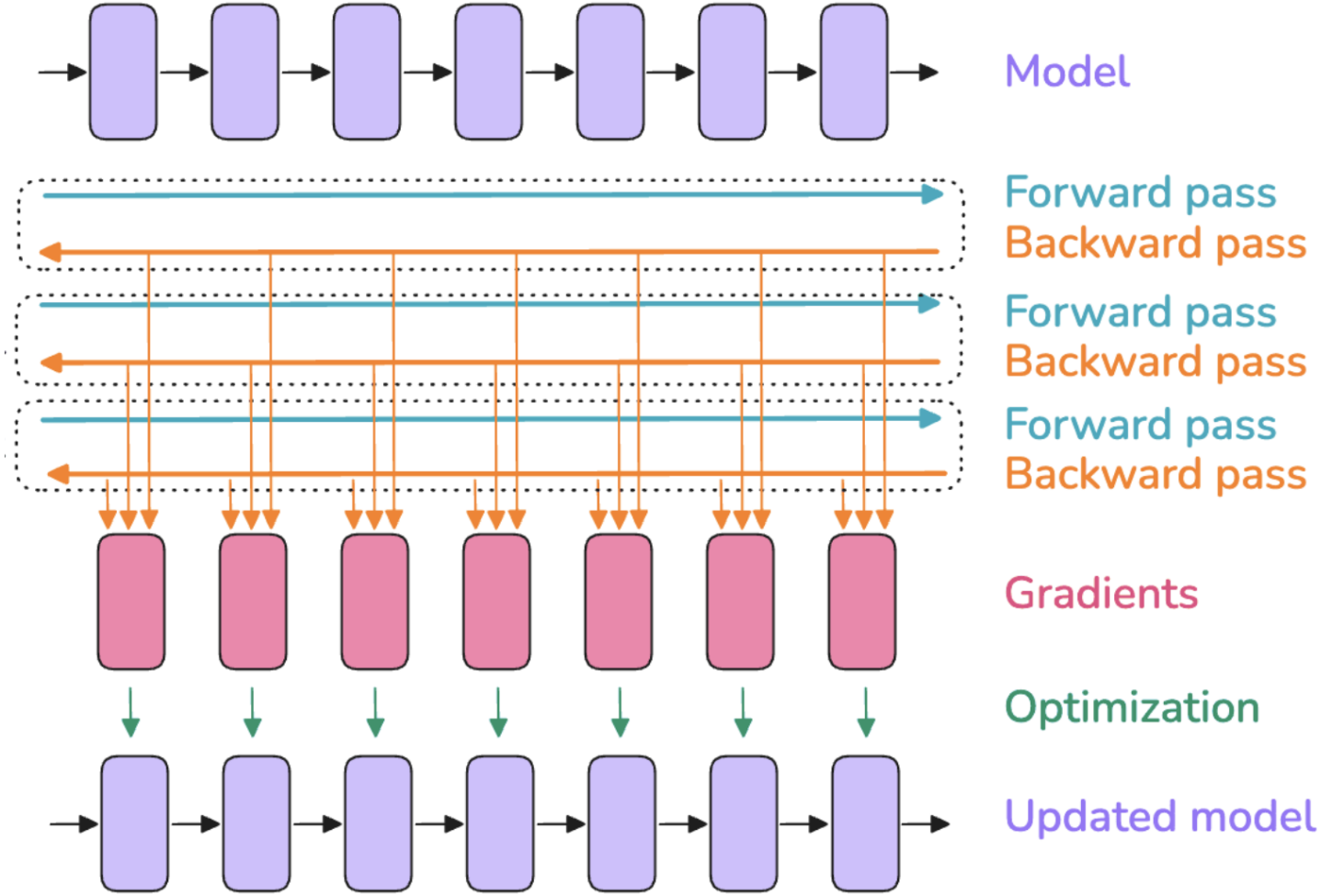
How to increase batch size? – *Gradient Accumulation*



How to speedup?

- Each forward / backward pass is on a different micro-batch.
- Independent of each other
- Can we run them in parallel?

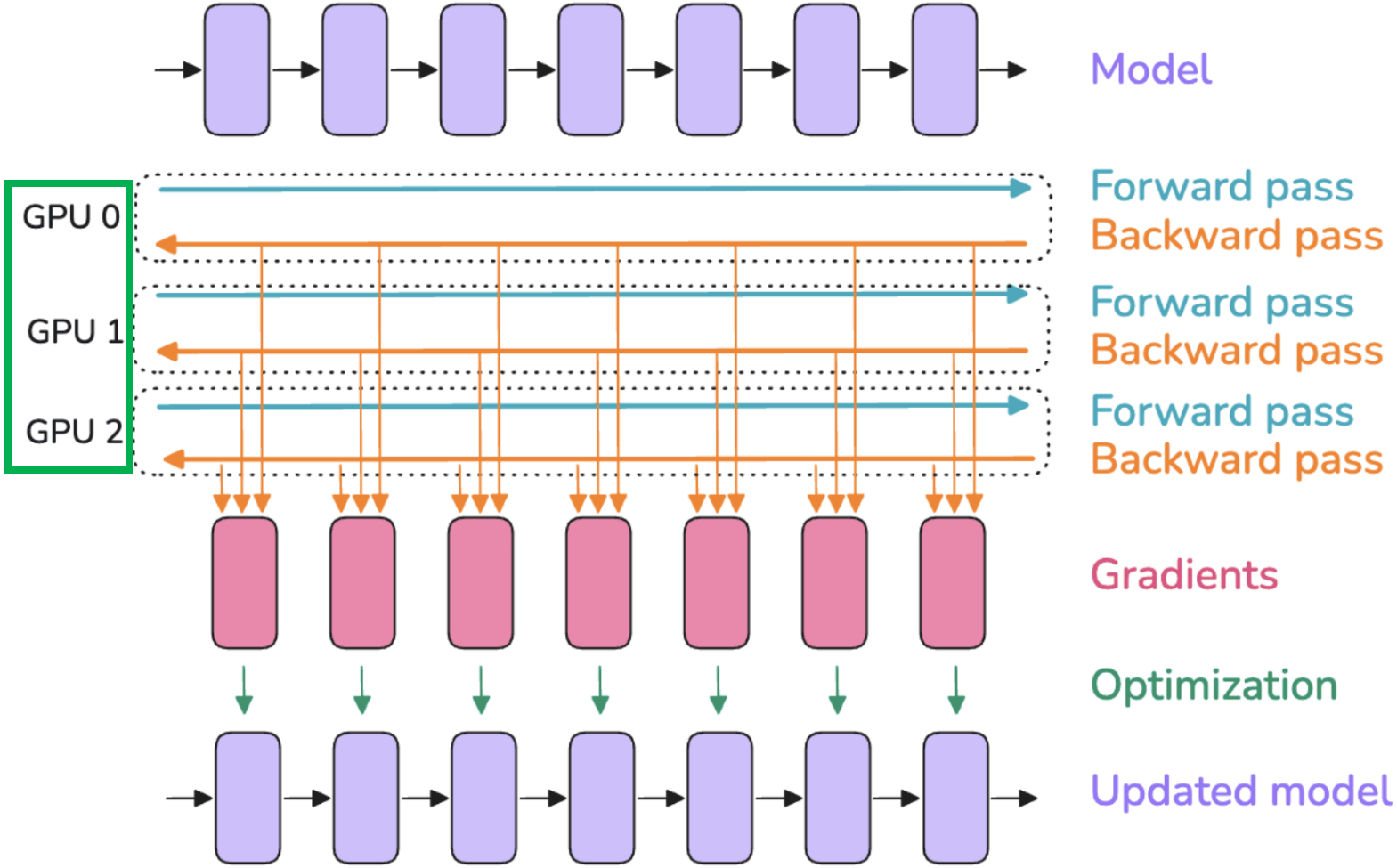
How to increase batch size? – *Gradient Accumulation*



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

- Process smaller micro-batches sequentially

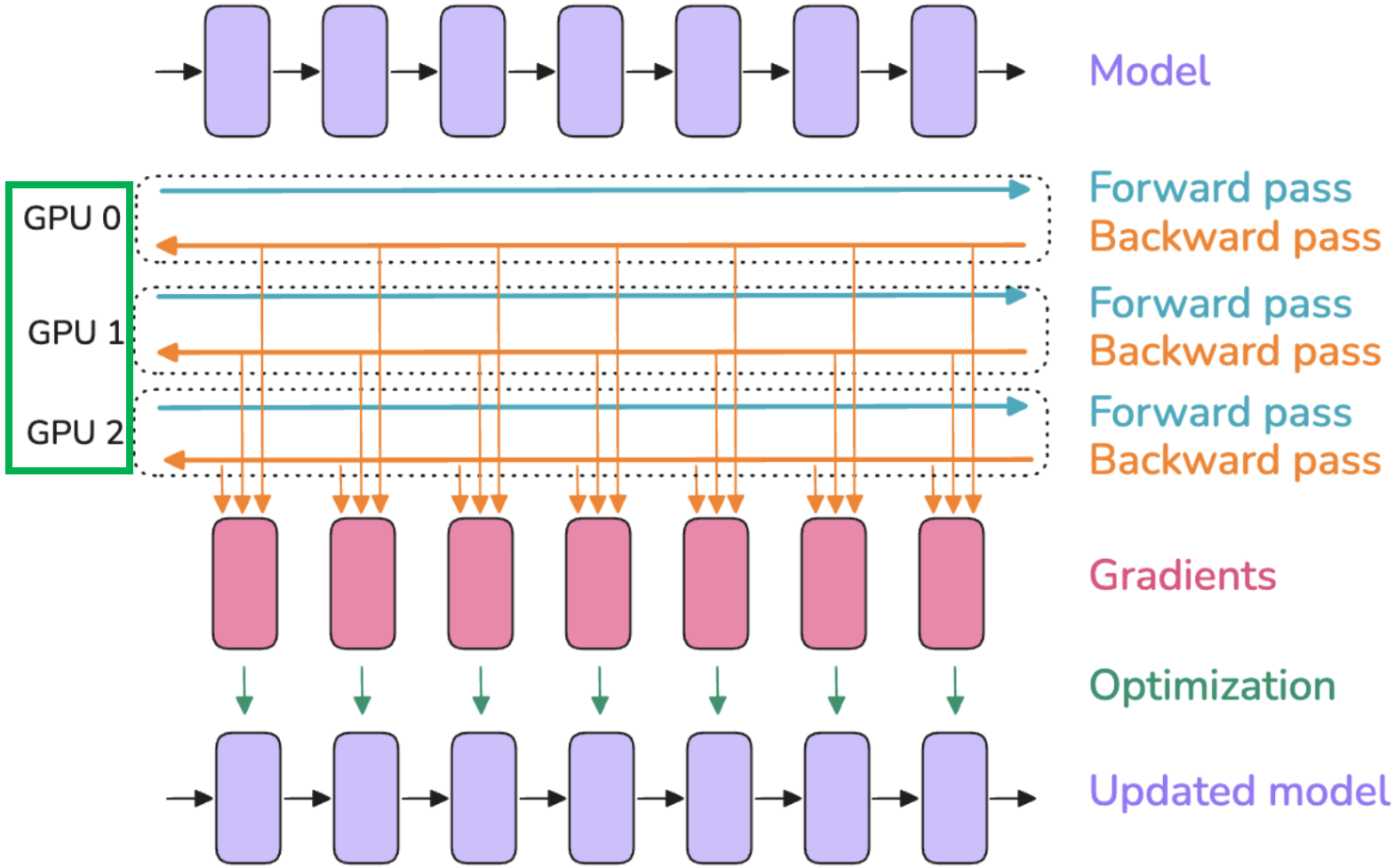
How to increase batch size? – *Data Parallelism*



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

- Process smaller micro-batches ~~sequentially~~ **Parallelly**

How to increase batch size? – *Data Parallelism*

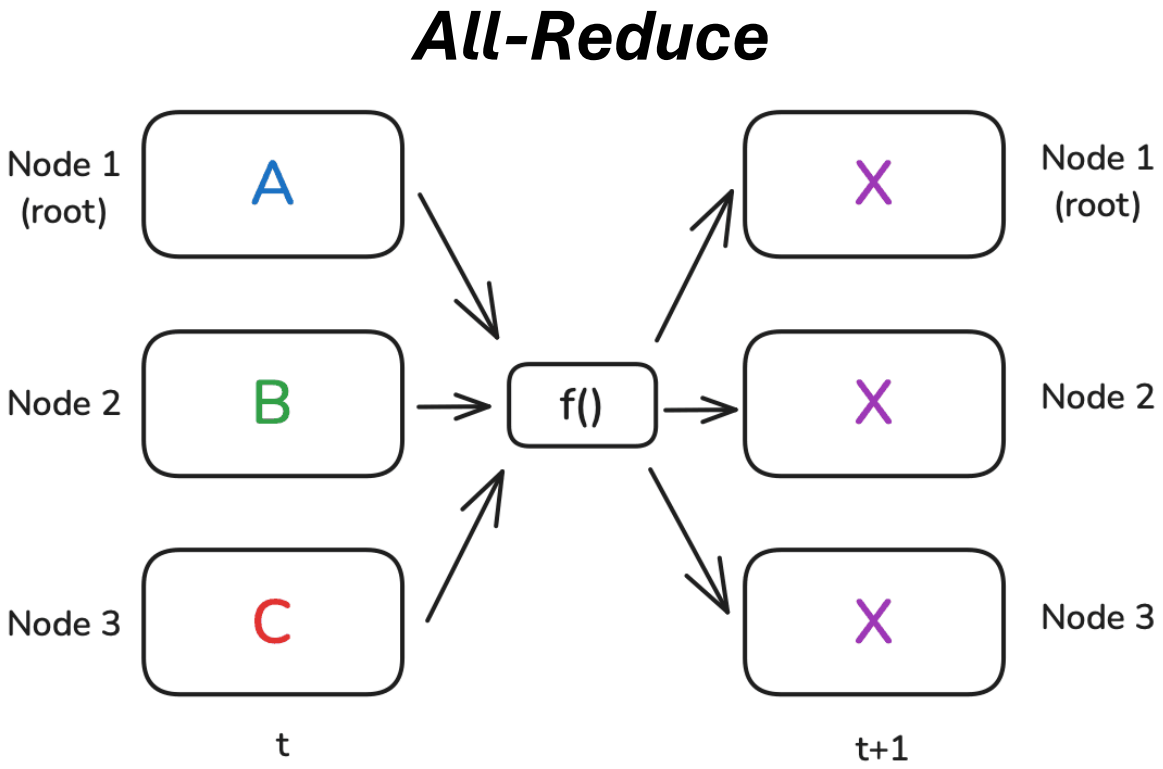


https://siboehm.com/articles/22/data-parallel-training

- Process smaller micro-batches ~~sequentially~~ **Parallelly**
- Keep a replica of params, grads, and optim states on each GPU
- Activations and grads computed locally and **independently** for each micro batch on each GPU
- **Communicate** the gradients to each other
- Independently update optim states & params on each GPU.

How to Communicate gradients?

- Use "***collective operations"*** **–** natively provided by PyTorch
- Called "communication *primitives*" - defined in `torch.distributed` API
- Each node performs some computation, e.g. grad. computation
- We communicate the result to other nodes for next computation step (t+1)



How to Communicate gradients?

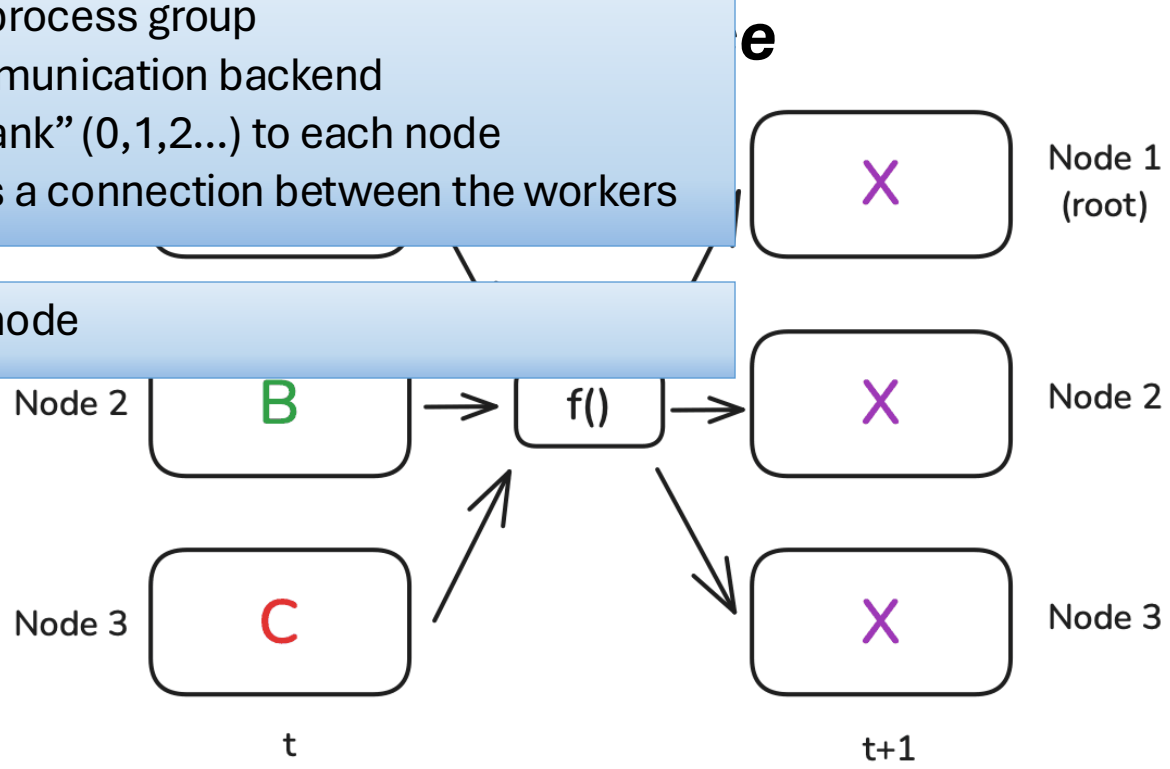
```
import torch
import torch.distributed as dist

def init_process():
  dist.init_process_group(backend='nccl')
  torch.cuda.set_device(dist.get_rank())

def example_all_reduce():
  tensor = torch.tensor([dist.get_rank()+1] * 5,
                dtype=torch.float32
  ).cuda()
  print(f"Before all_reduce on rank\
          {dist.get_rank()}: {tensor}")
  dist.all_reduce(tensor, op=dist.ReduceOp.SUM)
  print(f"After all_reduce on rank\
          {dist.get_rank()}: {tensor}")
```

- Initialize a process group
- Setup communication backend
- Assign a "rank" (0,1,2…) to each node
- establishes a connection between the workers

Rank of the node




```
torchrun --nproc_per_node=3 dist_op.py
```

How to Communicate gradients?

```python
import torch
import torch.distributed as dist

def init_process():
  dist.init_process_group(backend='nccl’)
  torch.cuda.set_device(dist.get_rank())

def example_all_reduce():
  tensor = torch.tensor([dist.get_rank()+1] * 5,
                dtype=torch.float32
  ).cuda()
  print(f“Before all_reduce on rank\
          {dist.get_rank()}: {tensor}")
  dist.all_reduce(tensor, op=dist.ReduceOp.SUM)
  print(f“After all_reduce on rank\
          {dist.get_rank()}: {tensor}")
```

All-Reduce

```
Before all_reduce on rank 0: tensor([1., 1., 1., 1., 1.], device='cuda:0')
Before all_reduce on rank 1: tensor([2., 2., 2., 2., 2.], device='cuda:1')
Before all_reduce on rank 2: tensor([3., 3., 3., 3., 3.], device='cuda:2')
```

```
torchrun --nproc_per_node=3 dist_op.py
```

https://docs.pytorch.org/docs/stable/distributed.html
https://docs.pytorch.org/tutorials/beginner/dist_overview.html

How to Communicate gradients?

```python
import torch
import torch.distributed as dist

def init_process():
  dist.init_process_group(backend='nccl’)
  torch.cuda.set_device(dist.get_rank())

def example_all_reduce():
  tensor = torch.tensor([dist.get_rank()+1] * 5,
                dtype=torch.float32
  ).cuda()
  print(f“Before all_reduce on rank\
          {dist.get_rank()}: {tensor}")
  dist.all_reduce(tensor, op=dist.ReduceOp.SUM)
  print(f“After all_reduce on rank\
          {dist.get_rank()}: {tensor}")
```

All-Reduce

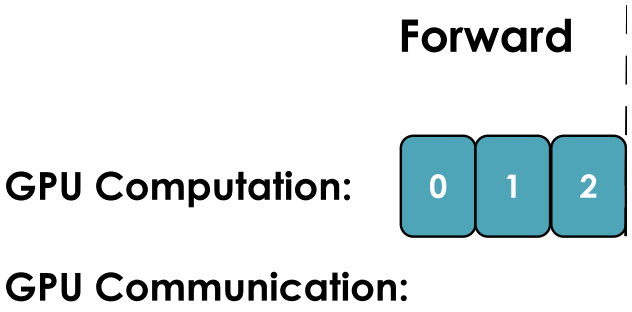
```
Before all_reduce on rank 0: tensor([1., 1., 1., 1., 1.], device='cuda:0')
Before all_reduce on rank 1: tensor([2., 2., 2., 2., 2.], device='cuda:1')
Before all_reduce on rank 2: tensor([3., 3., 3., 3., 3.], device='cuda:2')

After all_reduce on rank 0: tensor([6., 6., 6., 6., 6.], device='cuda:0')
After all_reduce on rank 1: tensor([6., 6., 6., 6., 6.], device='cuda:1')
After all_reduce on rank 2: tensor([6., 6., 6., 6., 6.], device='cuda:2')
```

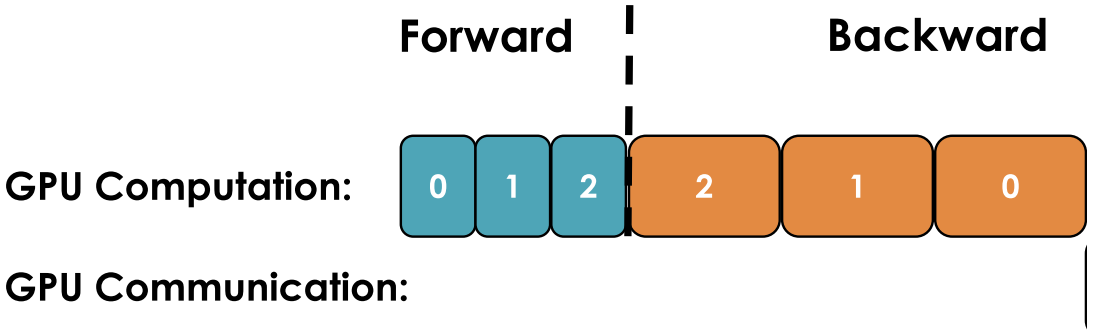
```
torchrun --nproc_per_node=3 dist_op.py
```

https://docs.pytorch.org/docs/stable/distributed.html
https://docs.pytorch.org/tutorials/beginner/dist_overview.html

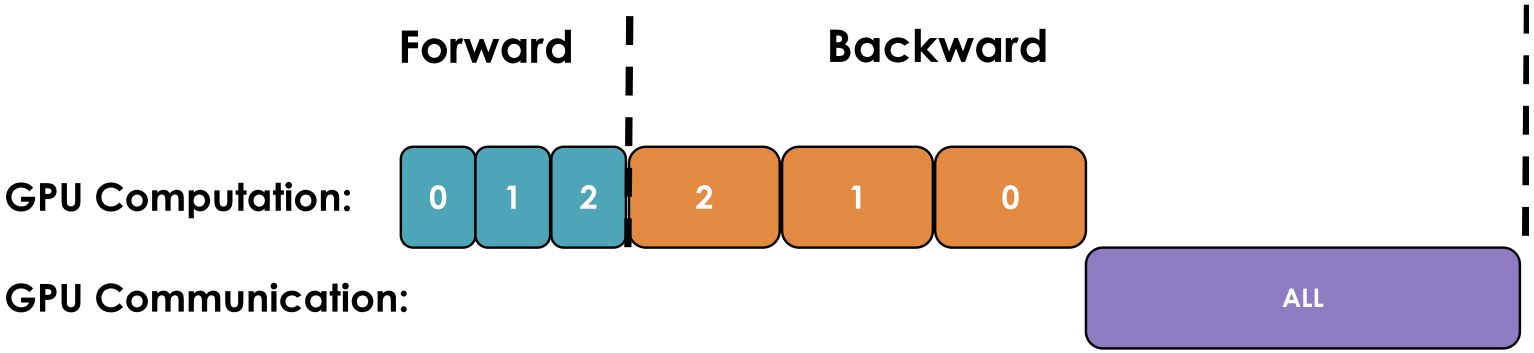
DP: Computation and Communication timeline



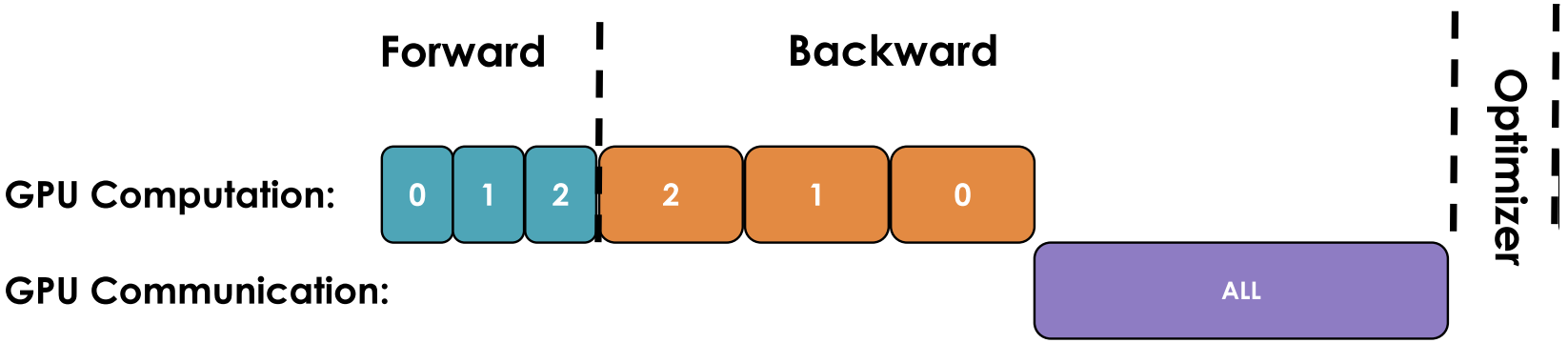
DP: Computation and Communication timeline



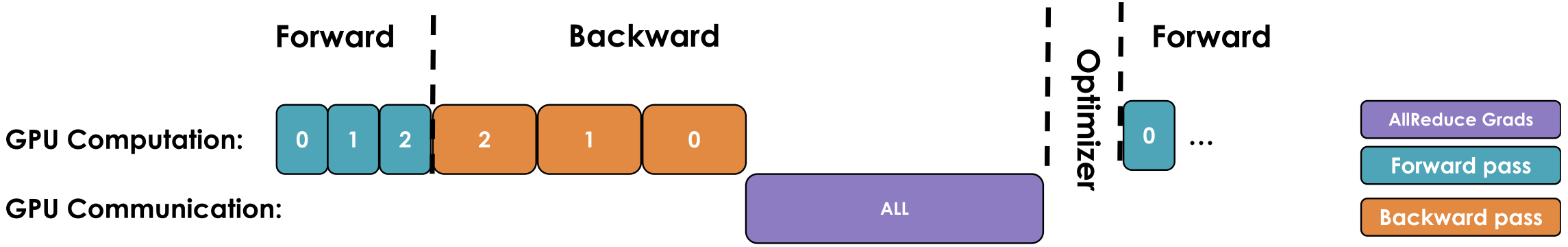
DP: Computation and Communication timeline



DP: Computation and Communication timeline

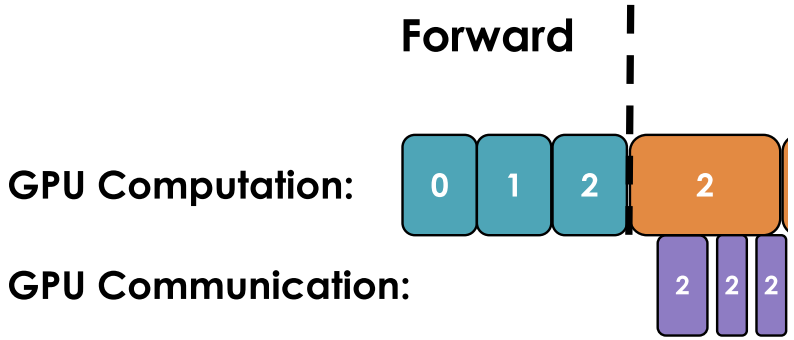


DP: Computation and Communication timeline



- GPUs are sitting idle while grads are synced.
- Can we avoid it?
- **OVERLAP GRADIENT SYNCHRONIZATION WITH BACKWARD PASS**

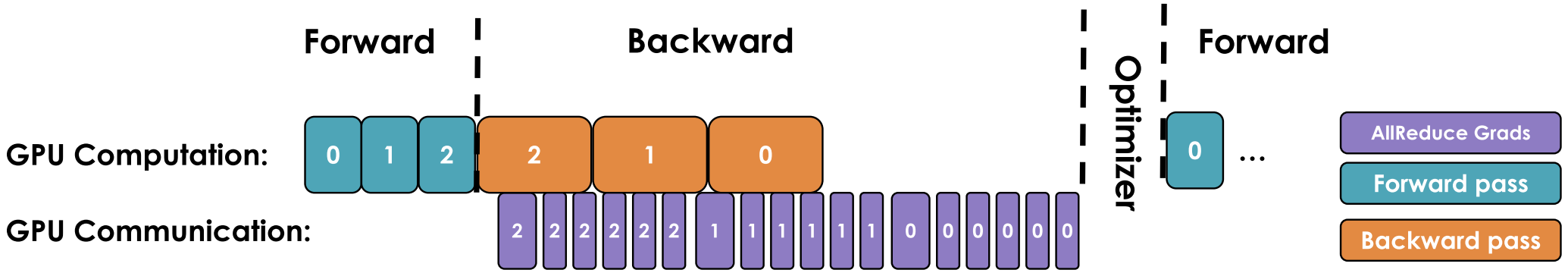
DP: Overlap grad. sync. with backward pass



- Attach an *all-reduce hook function* to each parameter

```
def register_backward_hook(self, hook):
""" Registers a backward hook for all parameters
of the model that require gradients. """
  for p in self.module.parameters():
    if p.requires_grad is True:
      p.register_post_accumulate_grad_hook(hook)
```

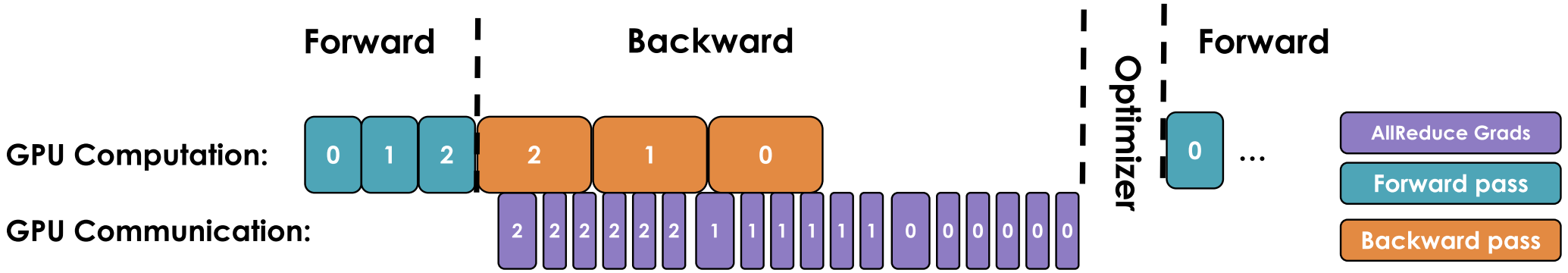
DP: Overlap grad. sync. with backward pass



- Attach an *all-reduce hook function* to each parameter

```
def register_backward_hook(self, hook):
""" Registers a backward hook for all parameters
of the model that require gradients. """
  for p in self.module.parameters():
    if p.requires_grad is True:
      p.register_post_accumulate_grad_hook(hook)
```

DP: Overlap grad. sync. with backward pass

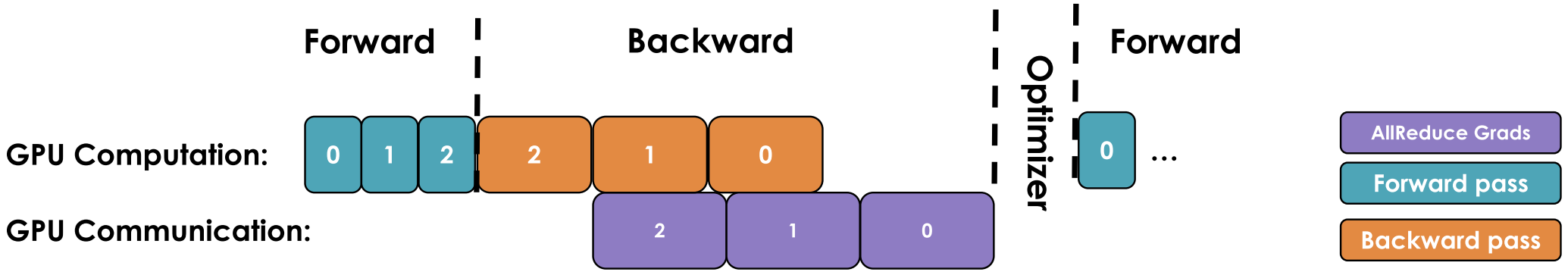


- Attach an *all-reduce hook function* to each parameter

Frequent communication with small packets.

```
def register_backward_hook(self, hook):
""" Registers a backward hook for all parameters
of the model that require gradients. """
  for p in self.module.parameters():
    if p.requires_grad is True:
      p.register_post_accumulate_grad_hook(hook)
```

DP: Bucketing Gradients



- Communication: more efficient when performed on large tensors
- Group gradients into “buckets” and launch a single all-reduce for all the gradients within the same bucket

- Like packing items into boxes before shipping
- More efficient to send a few big boxes than many small ones.
- Significantly reduce the communication overhead and speed up the communication operation.

Combining *Data Parallelism* with *Grad. Accumulation*

With Data Parallelism

$$bs = gbs = mbs * dp$$

With gradient accumulation:

$$bs = gbs = mbs * grad\_acc$$

With Data Parallelism and grad. acc.

$$bs = gbs = mbs * grad\_acc * dp$$

- Given a target gbs : **1024**

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

Combining *Data Parallelism* with *Grad. Accumulation*

With Data Parallelism

$$bs = gbs = mbs * dp$$

With gradient accumulation:

$$bs = gbs = mbs * grad\_acc$$

With Data Parallelism and grad. acc.

$$bs = gbs = mbs * grad\_acc * dp$$

- Given a target gbs :

1024
2

  - Assign mbs according to memory of 1GPU.

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

Combining *Data Parallelism* with *Grad. Accumulation*

With Data Parallelism

$$bs = gbs = mbs * dp$$

With gradient accumulation:

$$bs = gbs = mbs * grad\_acc$$

With Data Parallelism and grad. acc.

$$bs = gbs = mbs * grad\_acc * dp$$

- Given a target gbs :
  - Assign mbs according to memory of 1GPU.
  - Assign dp according to available #GPUs.

1024
2
128

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

Combining *Data Parallelism* with *Grad. Accumulation*

With Data Parallelism

$$bs = gbs = mbs * dp$$

With gradient accumulation:

$$bs = gbs = mbs * grad\_acc$$

With Data Parallelism and grad. acc.

$$bs = gbs = mbs * grad\_acc * dp$$

- Given a target gbs :
  - Assign mbs according to memory of 1GPU.
  - Assign dp according to available #GPUs.
  - Accordingly set $grad\_acc$

1024
2
128

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

Combining *Data Parallelism* with *Grad. Accumulation*

With Data Parallelism

$$bs = gbs = mbs * dp$$

With gradient accumulation:

$$bs = gbs = mbs * grad\_acc$$

With Data Parallelism and grad. acc.

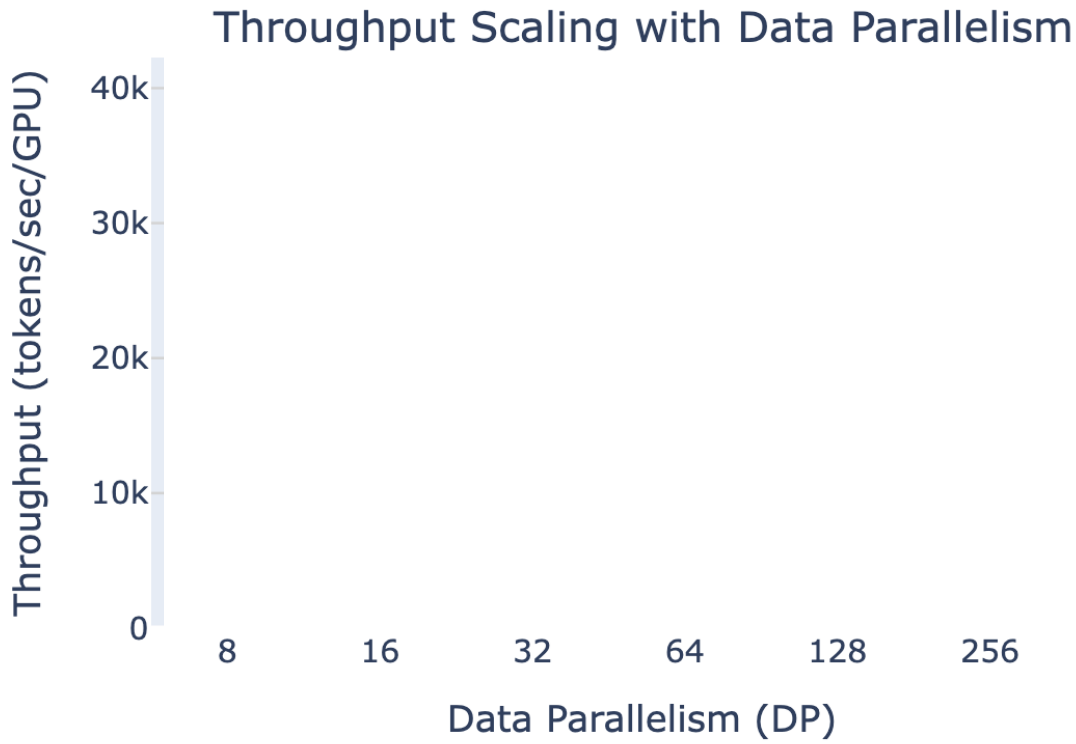
$$bs = gbs = mbs * grad\_acc * dp$$

- Given a target gbs :
  - Assign mbs according to memory of 1GPU.
  - Assign dp according to available #GPUs.
  - Accordingly set $grad\_acc$

1024
2
128
4

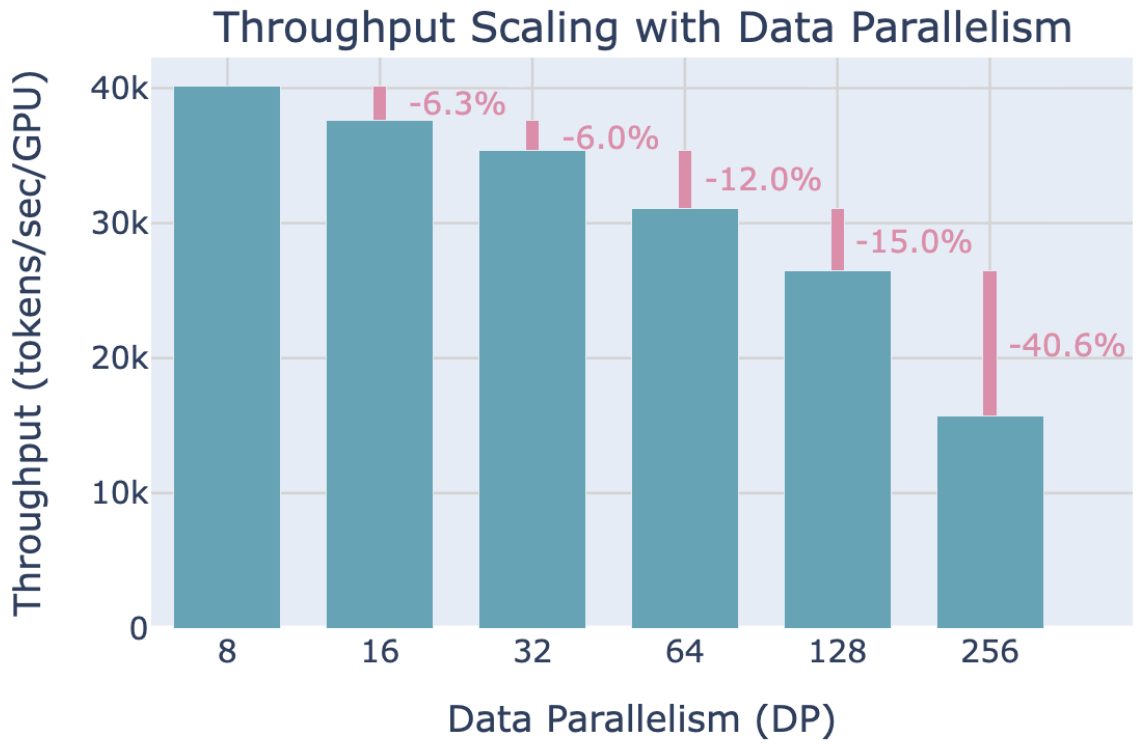
https://siboehm.com/articles/22/data-parallel-training

Can we scale DP as much as we want?



Can we scale DP as much as we want?

- DP benefit starts to break down at large scales.
- As we add more and more GPUs (hundreds or thousands), the overhead of coordinating between them grows significantly.
- The network requirements start to become too large for the benefits.
- As a result, our setup will become less and less efficient with each additional GPU we add to the system.

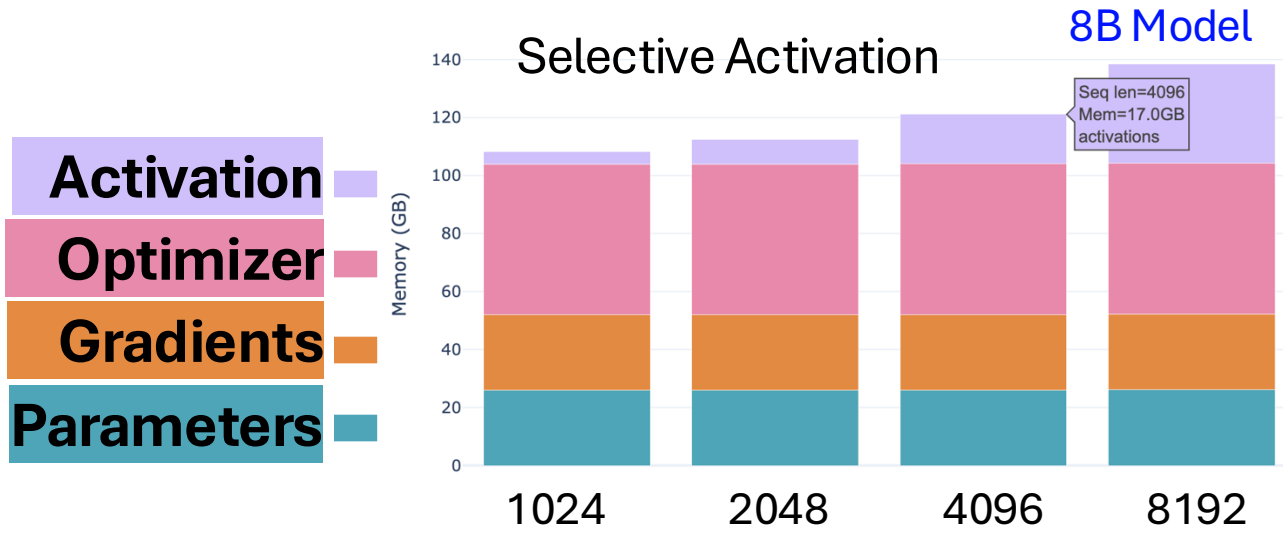


Assumptions of DP

- Model fits on 1 GPU

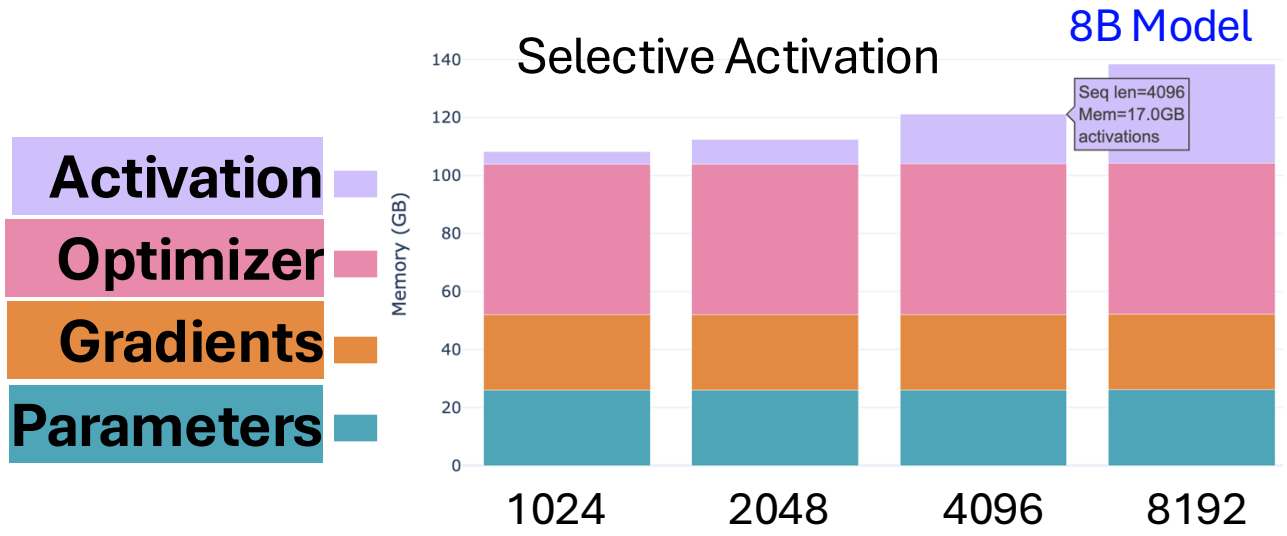
Assumptions of DP

- Model fits on 1 GPU



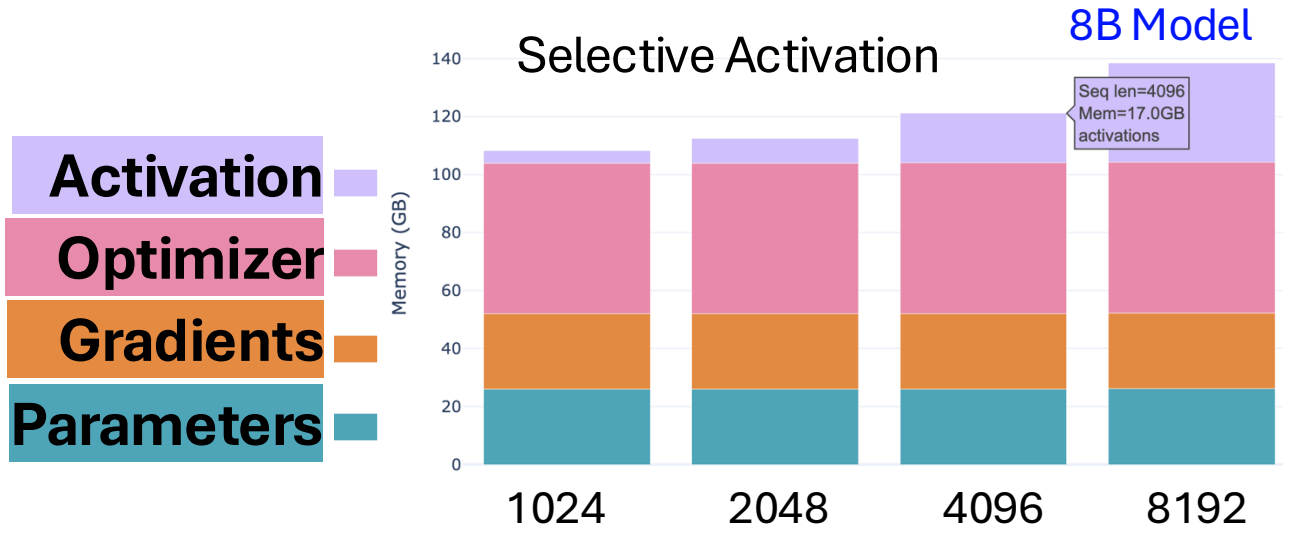
Assumptions of DP

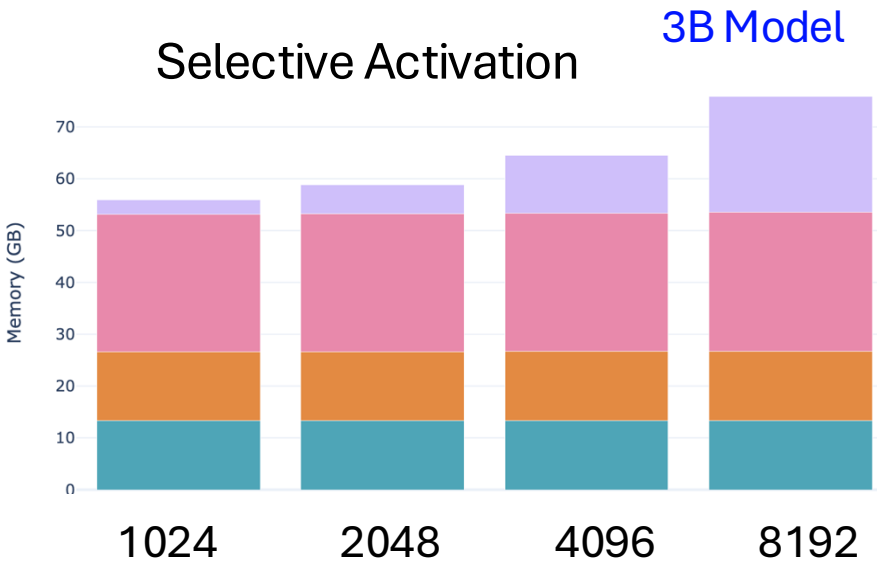
- Model fits on 1 GPU
- One sequence fits on 1 GPU



Assumptions of DP

- Model fits on 1 GPU
- One sequence fits on 1 GPU



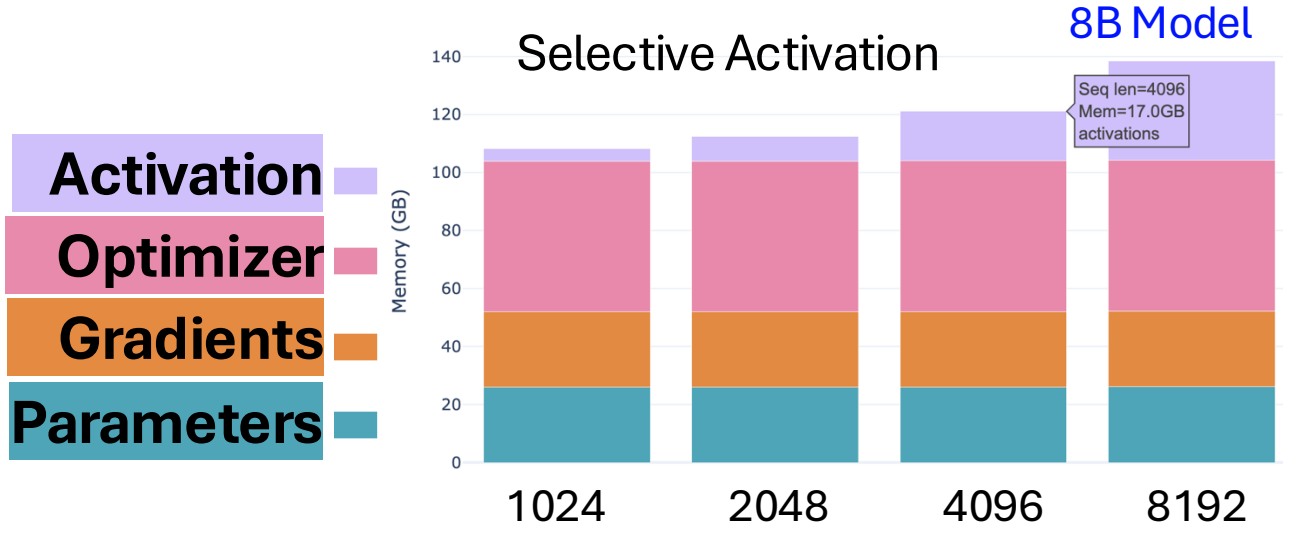


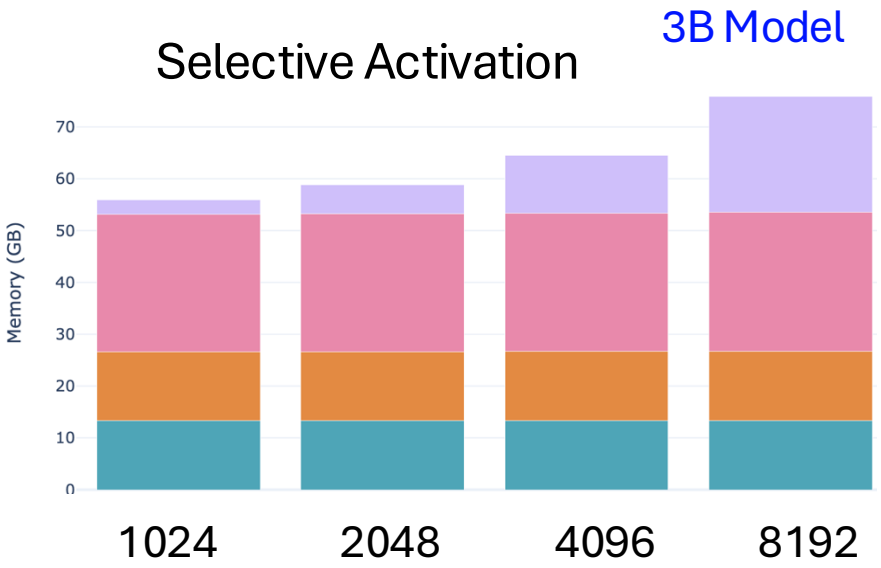
What can we do to improve it further?

Is there any redundancy?

Assumptions of DP

- Model fits on 1 GPU
- One sequence fits on 1 GPU





Recall weight utilization in **mixed precision training**

Memory for Weights, Grads, and Optim States

- **Mixed precision: computation in bf16 (2 bytes); storage in FP32**
  - $m_{params} = 2 * \Psi$
  - $m_{grad} = 2 * \Psi$
  - $m_{opt} = (4 + 4) * \Psi$
  - $m_{params\_fp32} = 4 * \Psi$ (master weights)

Total: $16 * \Psi bytes$

Why use mixed precision if total memory is same?

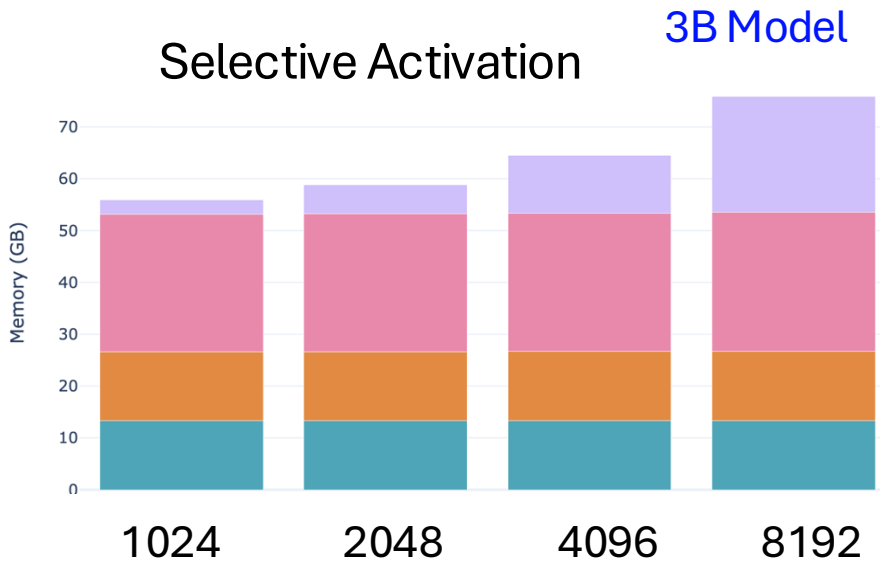
1. Allows us to use optimized lower precision operations on the GPU, which are faster
2. Reduces the activation memory requirements during the forward pass

Assumptions of DP

- Model fits on 1 GPU
- One sequence fits on 1 GPU

In **mixed precision training,**

- We keep params and grads in low precision (2 bytes)
- A copy of params in full precision FP32 (4 bytes)
- Optim states (1st and 2nd moment of grad.) in FP32

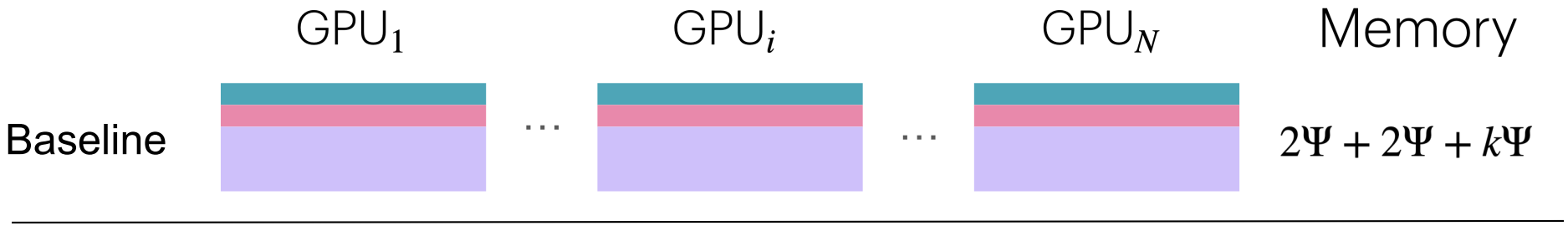


What can we do to improve it further?

Is there any redundancy?

Do we need ***all*** optim states on ***all*** GPUs?

Can we ***shard*** optim states on GPUs?



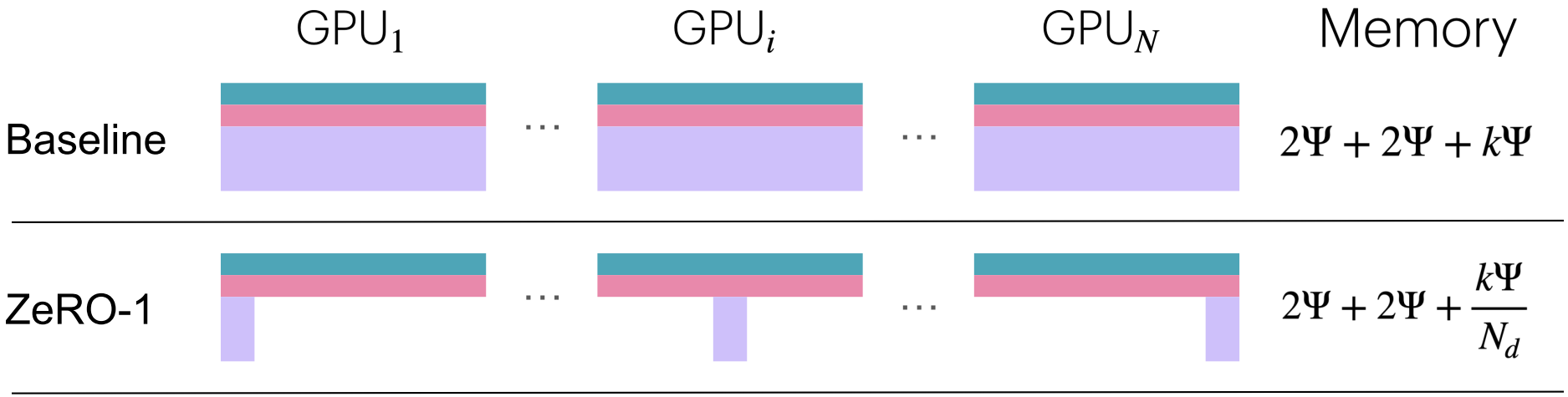
Ψ : Model parameters

k : Optimizer multiplier

N_d: Data parallel degree

Optimizer **Gradients** **Parameters**

Can we ***shard*** optim states on GPUs?



Ψ : Model parameters

k : Optimizer multiplier

N_d: Data parallel degree

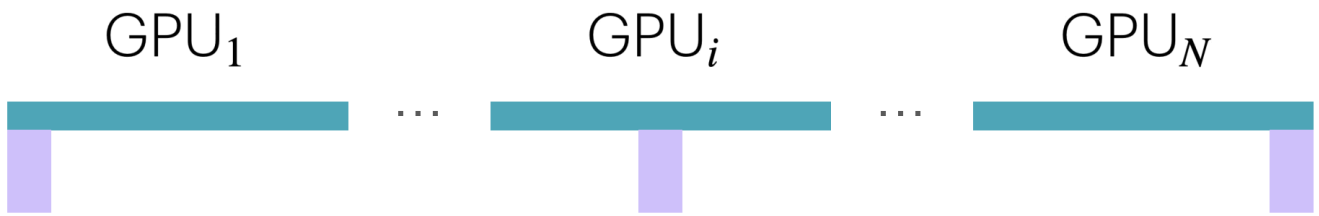
In DP,

1. We communicate the gradients via ***All-Reduce***
2. Update the optim states – 1st & 2nd moment, and FP32 copy of weights

Do we need any additional operation here?

Optimizer **Gradients** **Parameters**

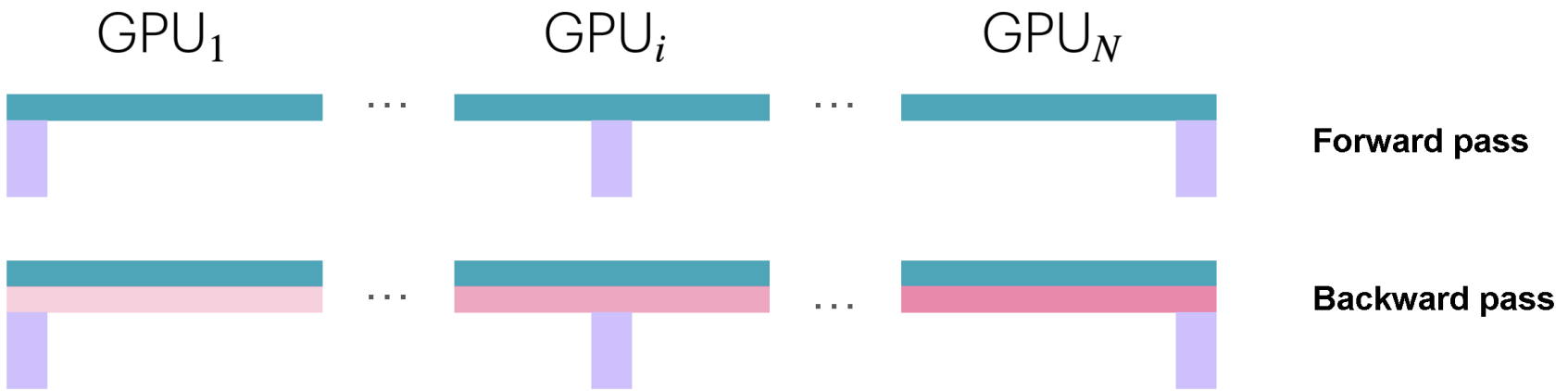
ZeRO-1



Forward pass

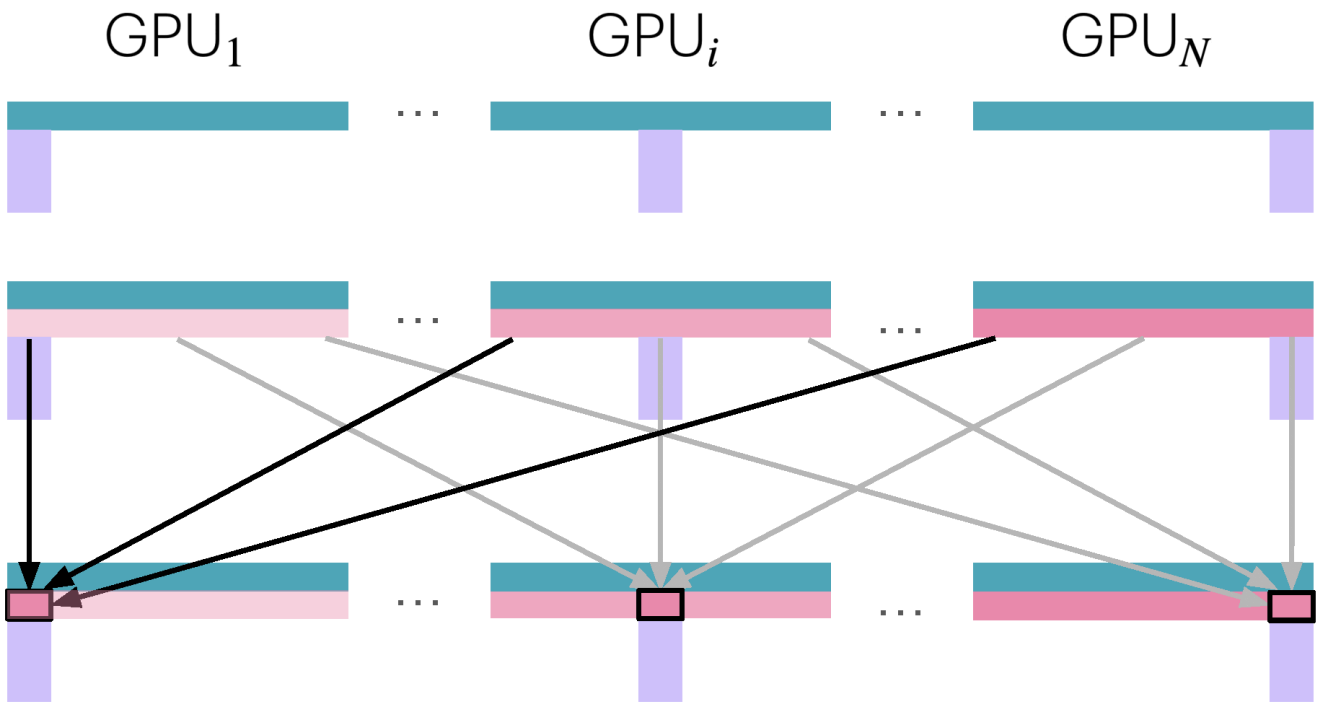
Parameters **Gradients** **Optimizer States**

ZeRO-1



Parameters | Gradients | Optimizer States

ZeRO-1



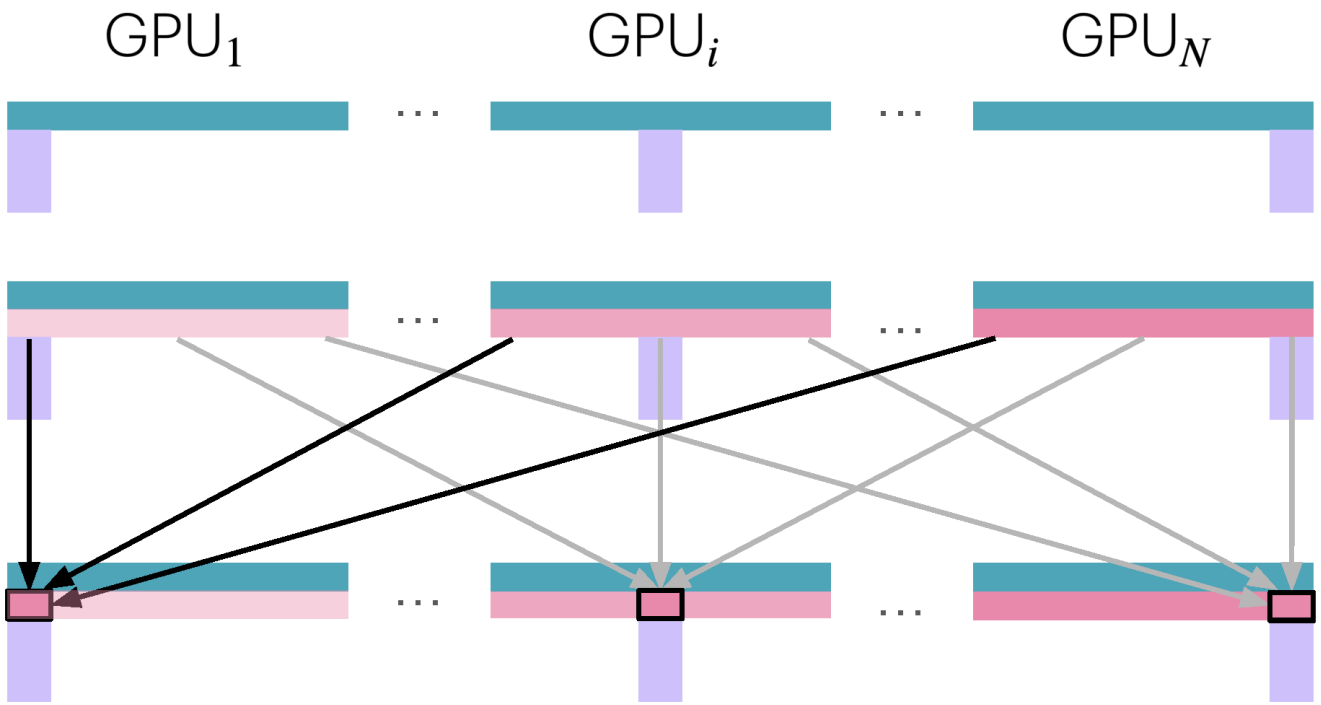
Forward pass

Backward pass

All-Reduce

Parameters **Gradients** **Optimizer States**

ZeRO-1



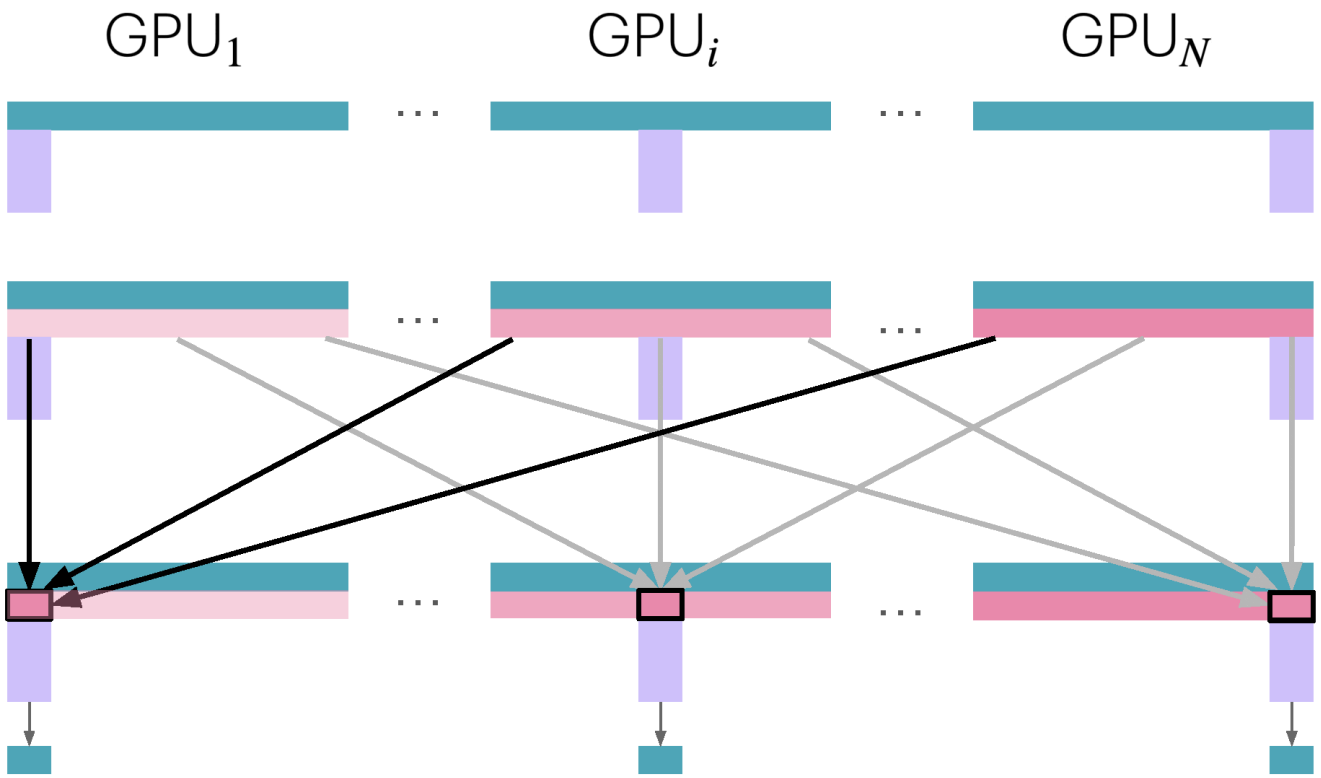
Forward pass

Backward pass

All-Reduce

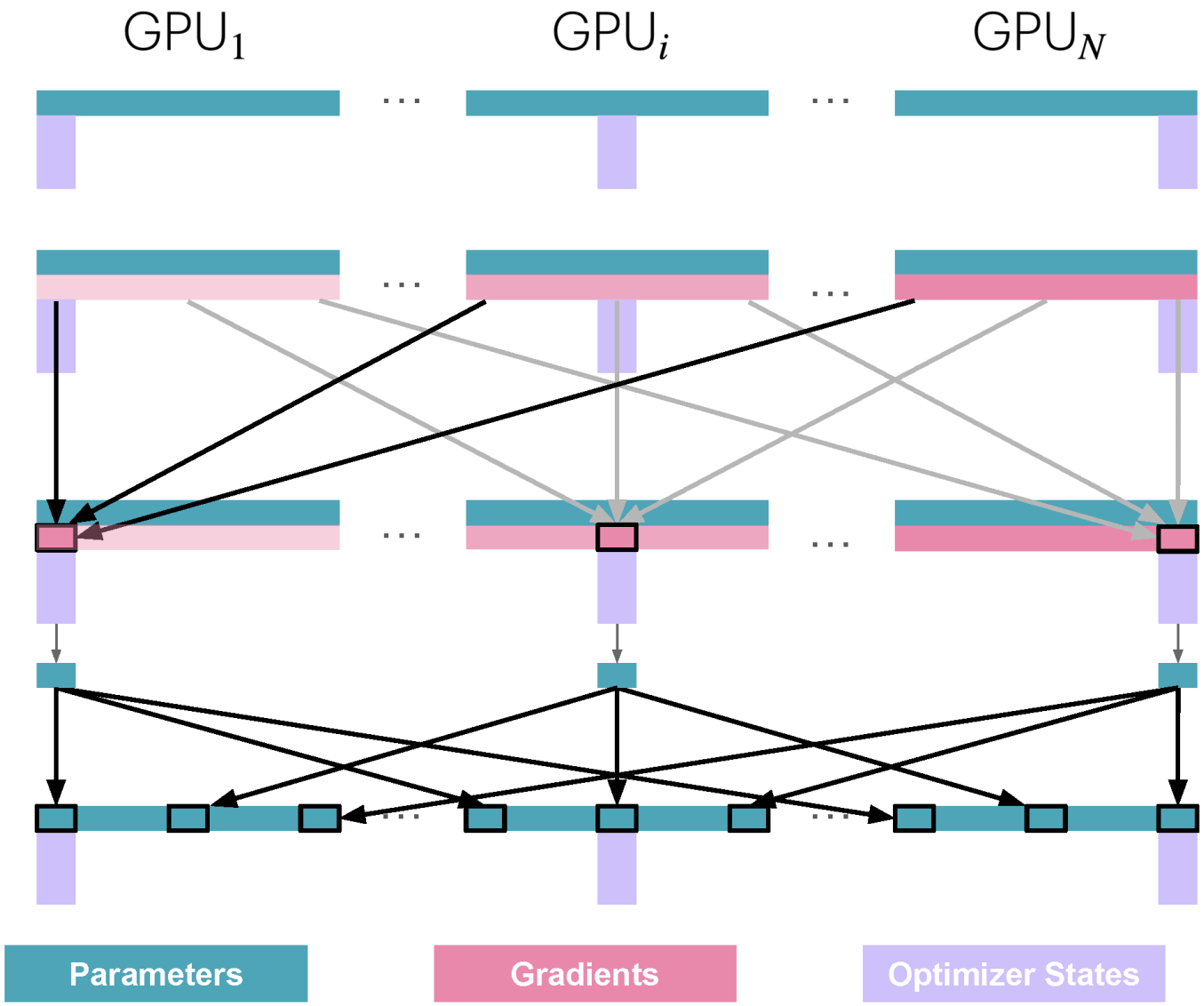
Parameters **Gradients** **Optimizer States**

ZeRO-1



Parameters | Gradients | Optimizer States

ZeRO-1



Forward pass

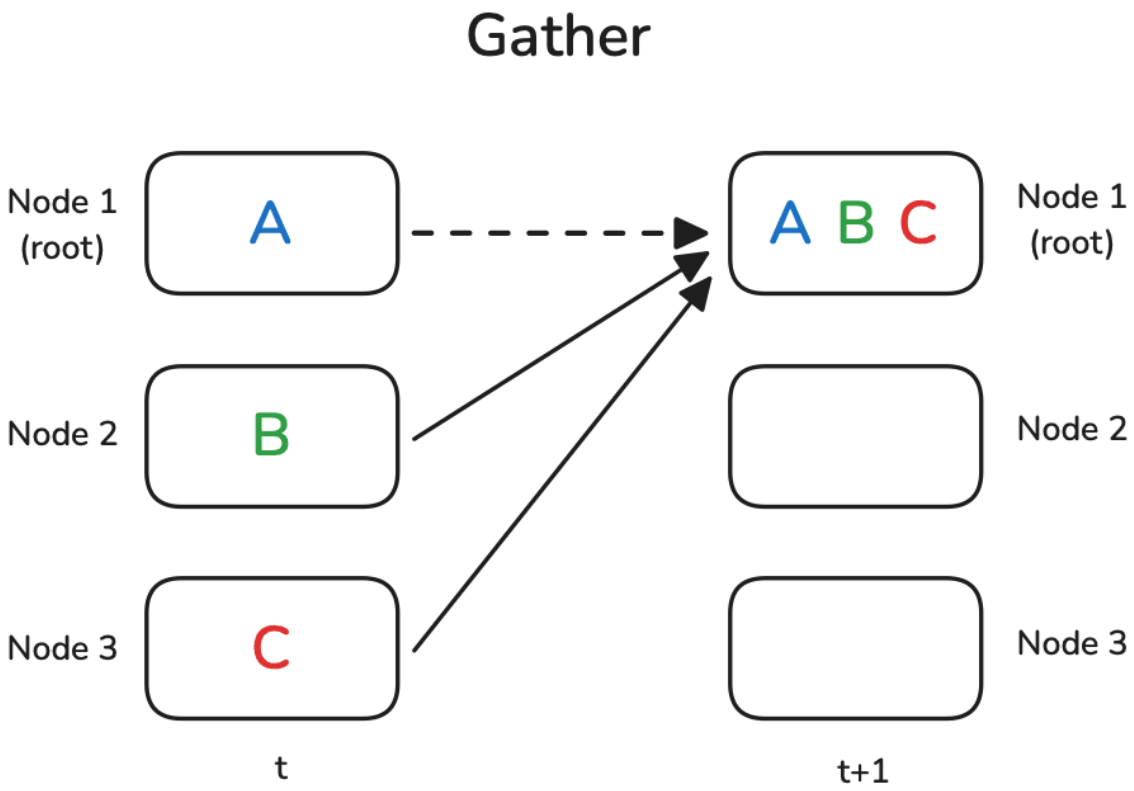
Backward pass

All-Reduce

Optimizer step

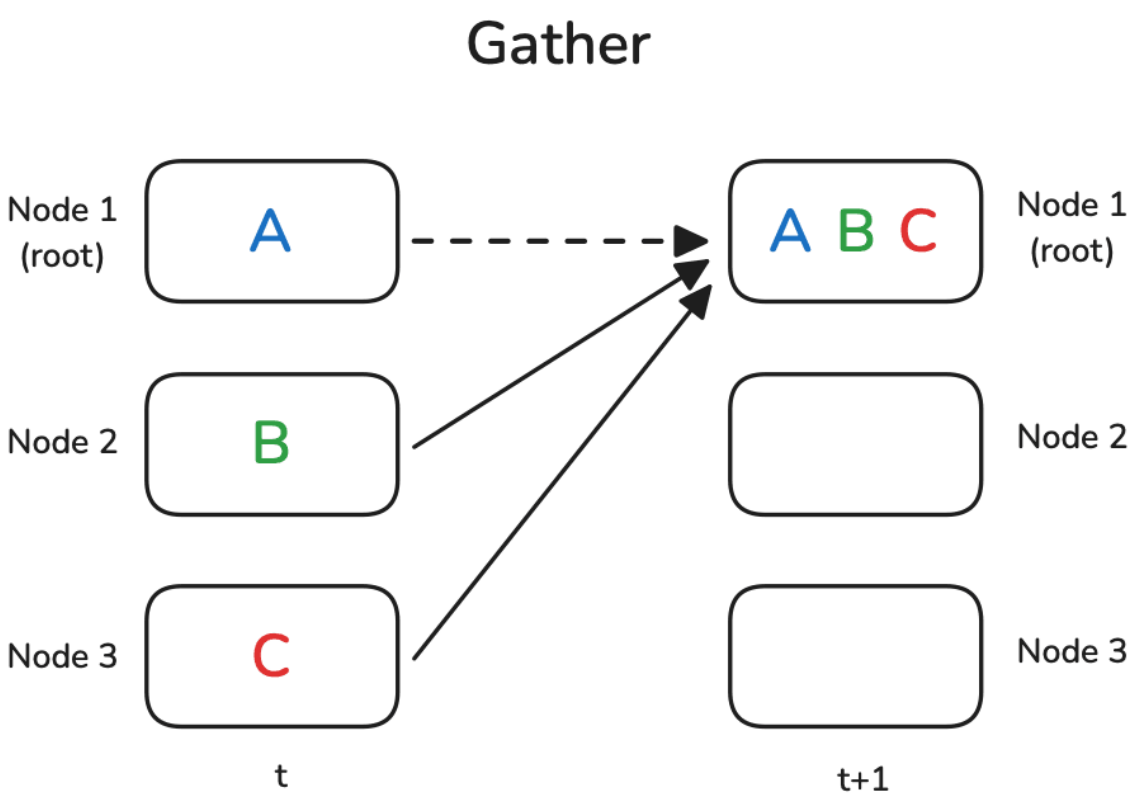
All-gather

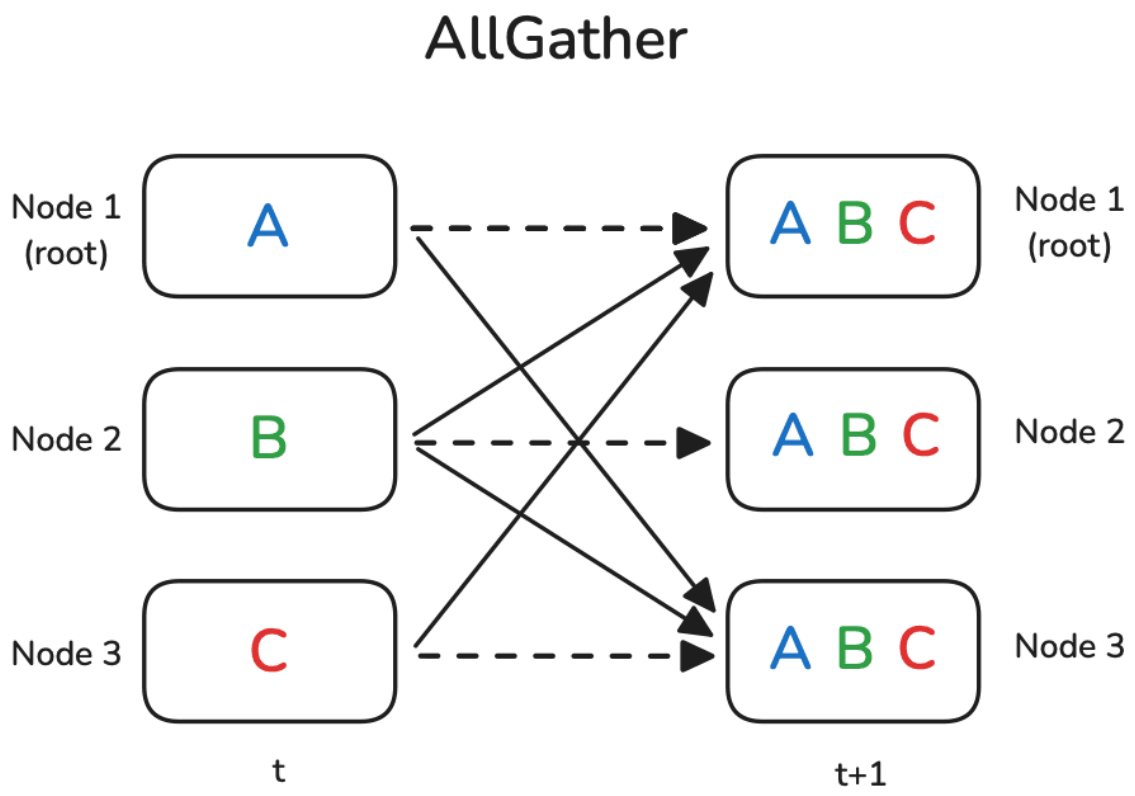
All-Gather – another communication primitive



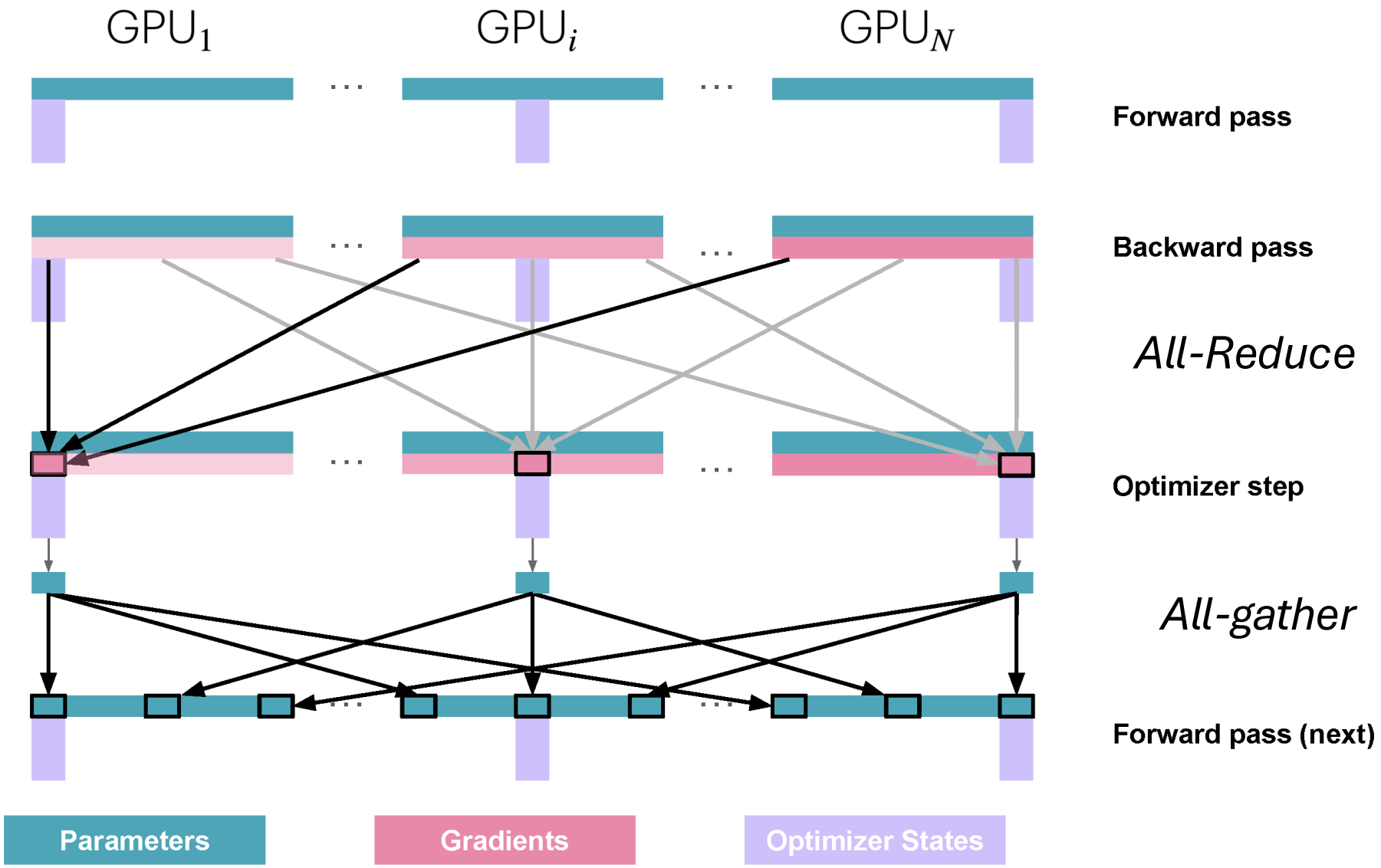
Any guesses on what would all-gather look like?

All-Gather – another communication primitive

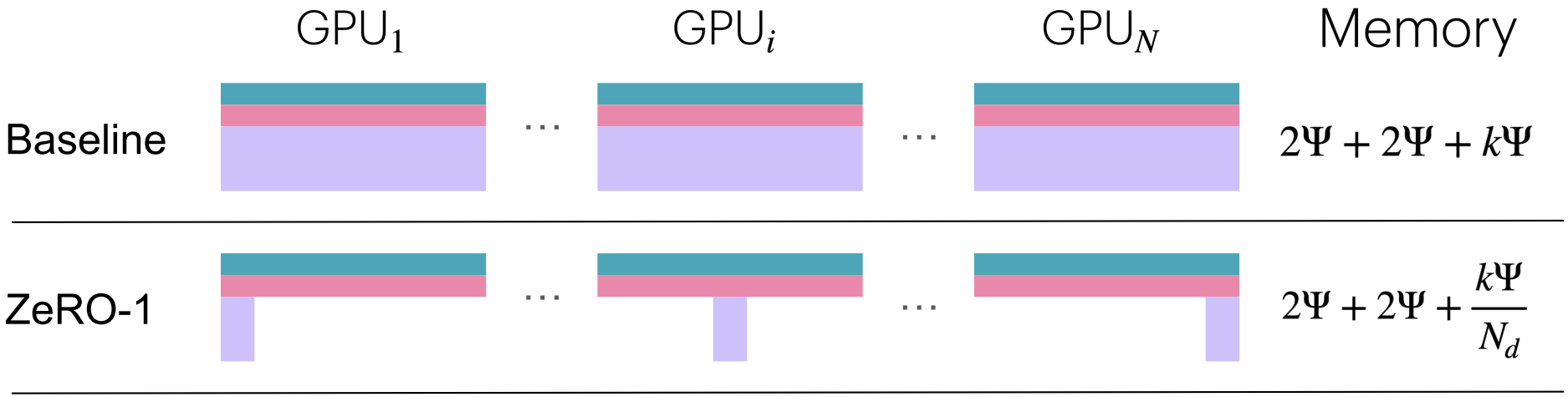




ZeRO-1


GPU_1
GPU_i
GPU_N
Forward pass
Backward pass
All-Reduce
Optimizer step
All-gather
Forward pass (next)
Parameters
Gradients
Optimizer States

Can we ***shard*** gradients on GPUs?



Ψ : Model parameters

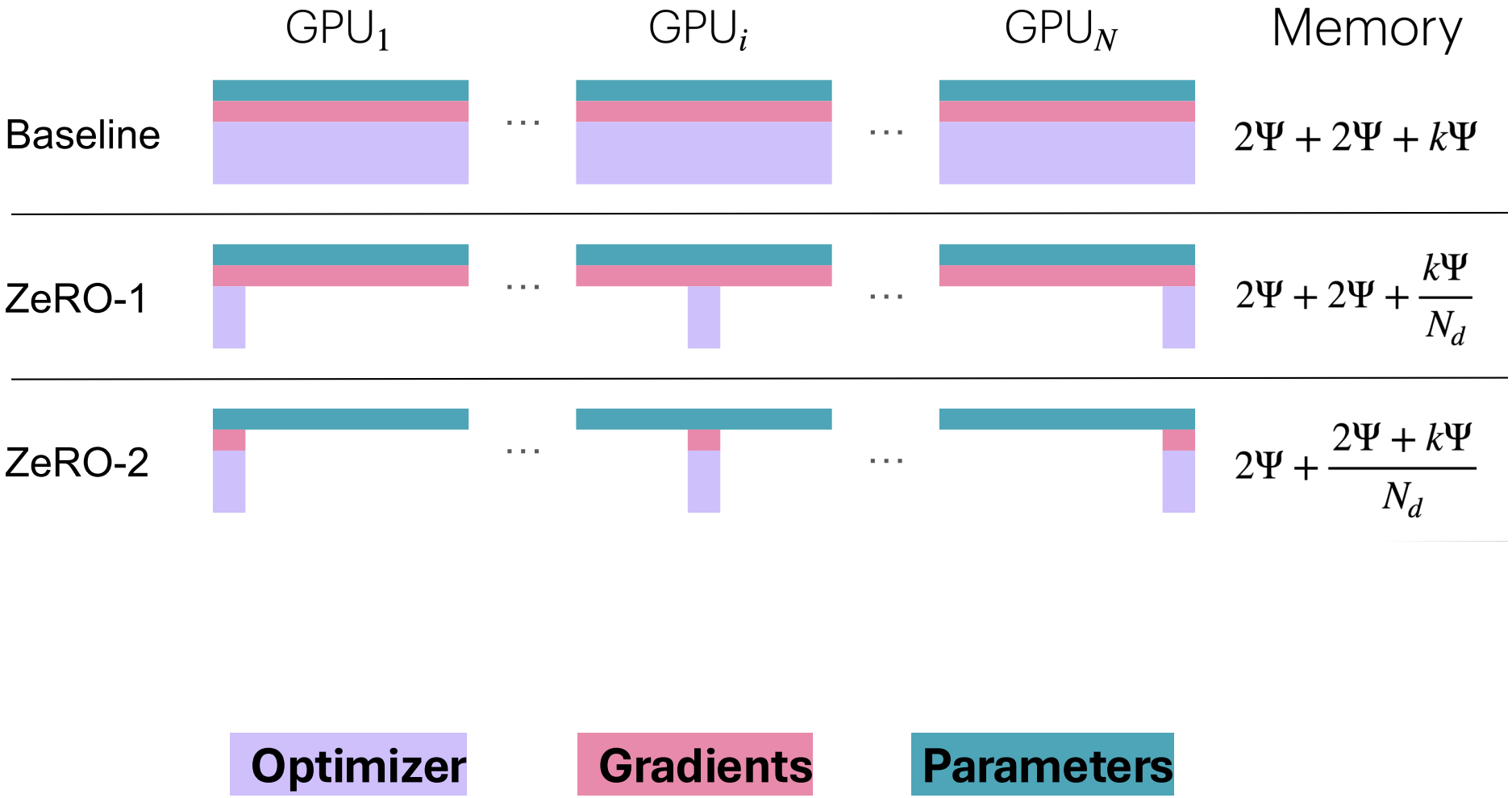
k : Optimizer multiplier

N_d: Data parallel degree

Do we need to store ***all*** gradients on ***all*** GPUs?

Optimizer **Gradients** **Parameters**

Can we ***shard*** gradients on GPUs?

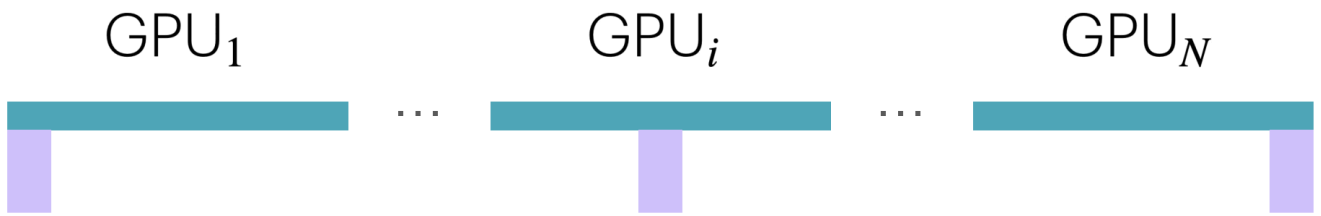


Ψ : Model parameters

k : Optimizer multiplier

N_d: Data parallel degree

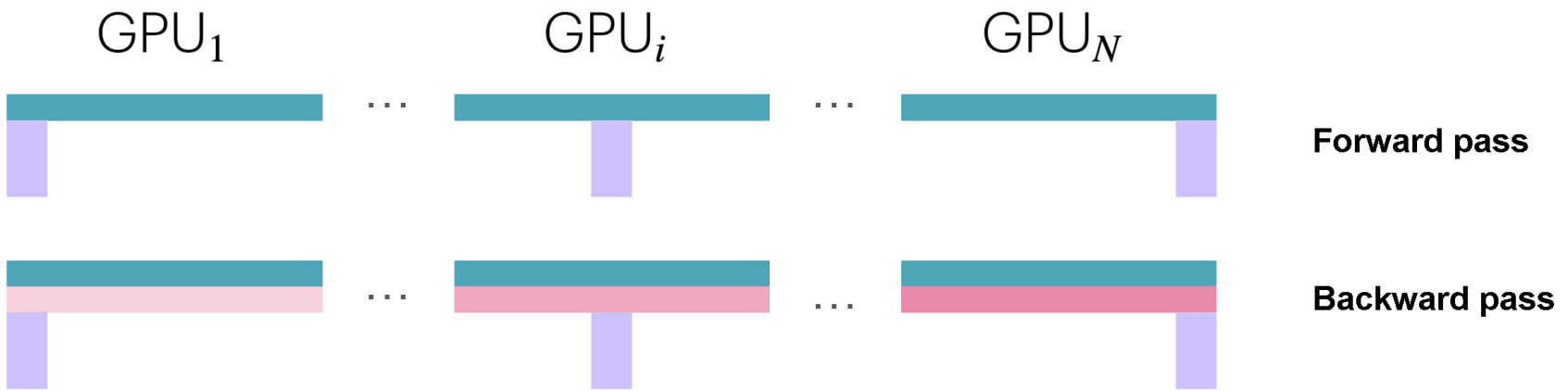
ZeRO-2



Forward pass

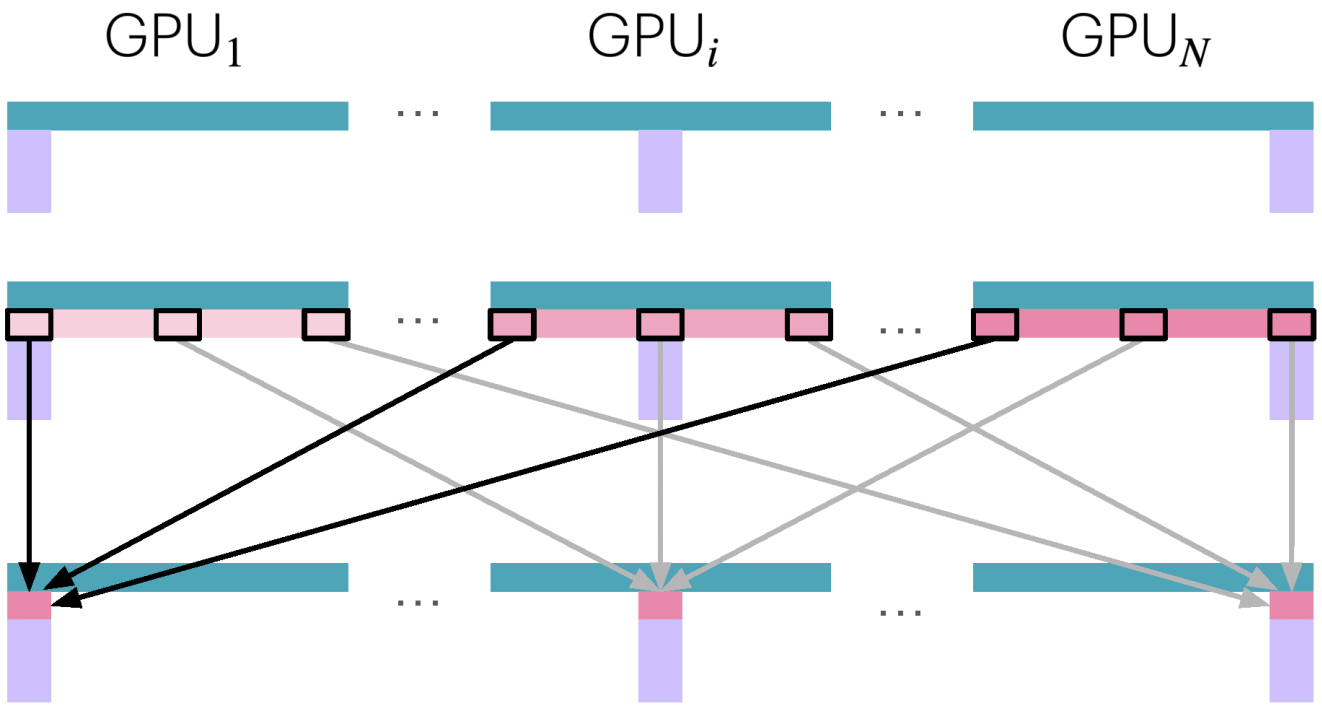
Parameters **Gradients** **Optimizer States**

ZeRO-2



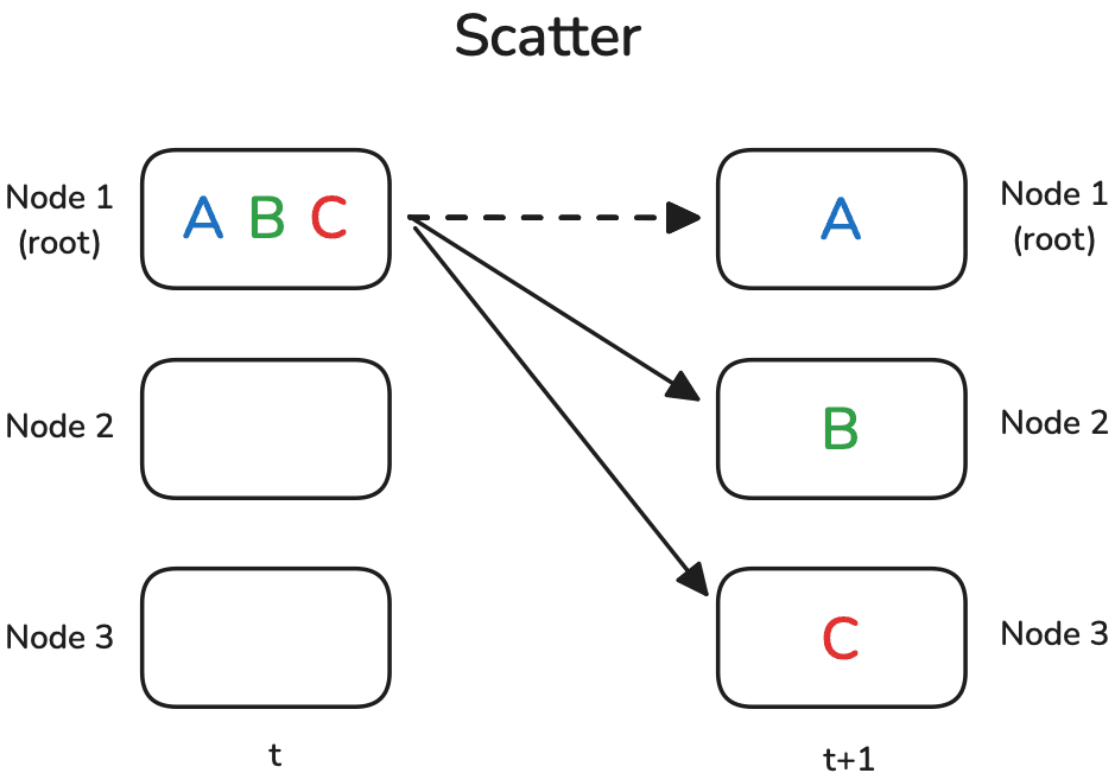
Parameters | Gradients | Optimizer States

ZeRO-2



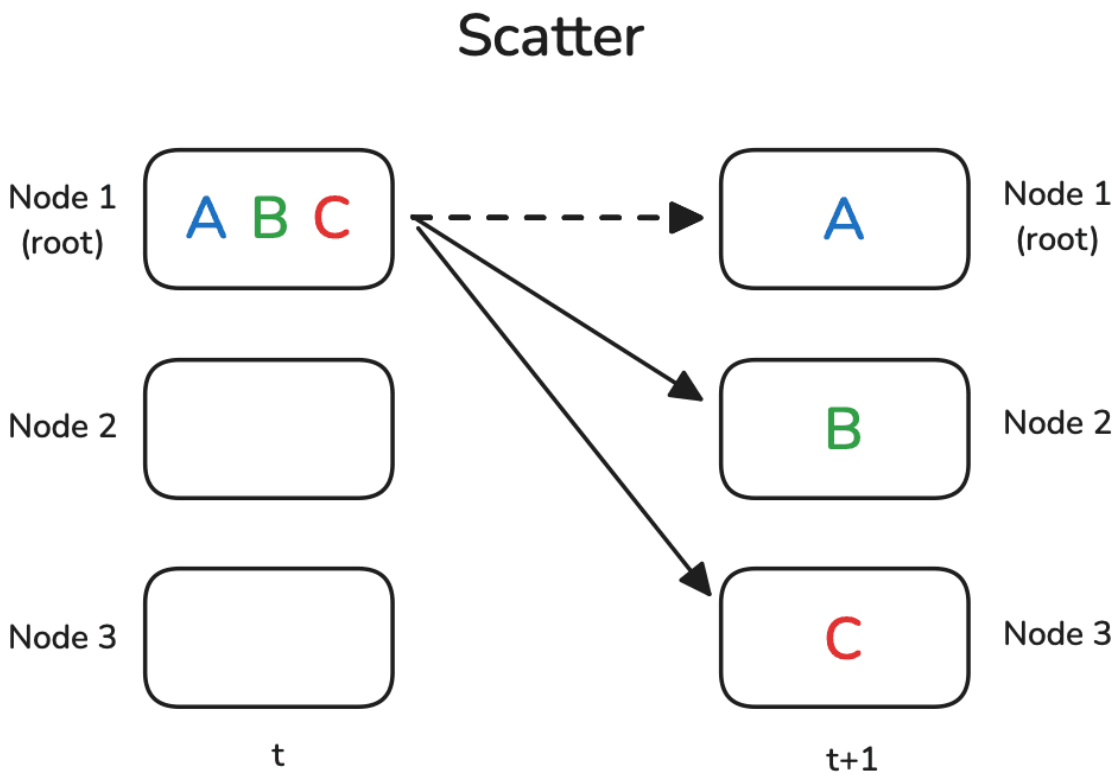
Parameters | Gradients | Optimizer States

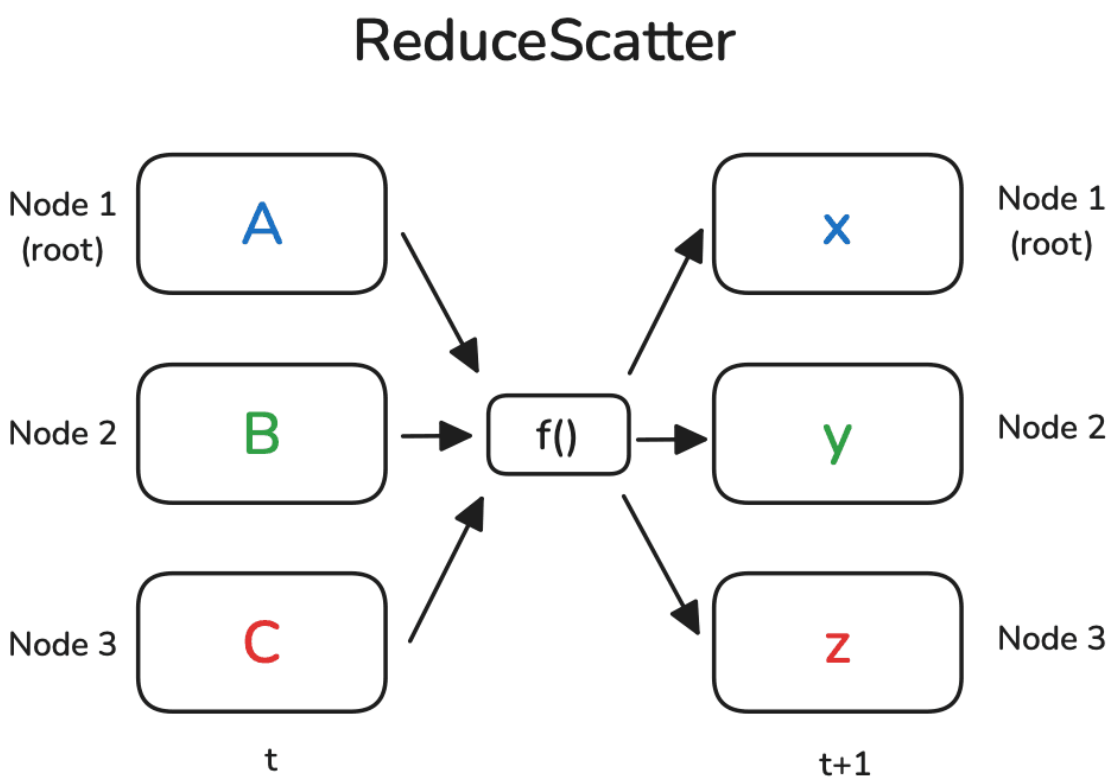
ReduceScatter – another communication primitive



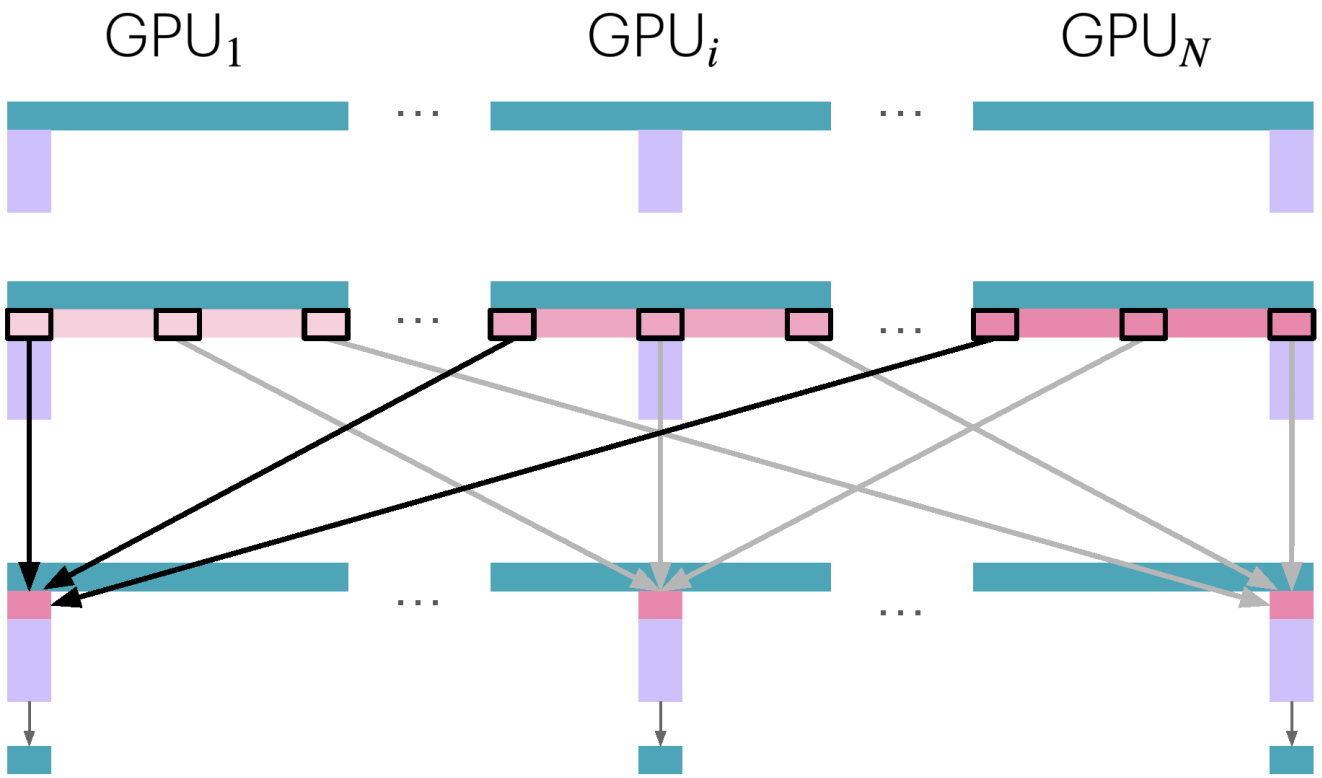
Combination of ***Reduce*** and ***Scatter***

ReduceScatter – another communication primitive



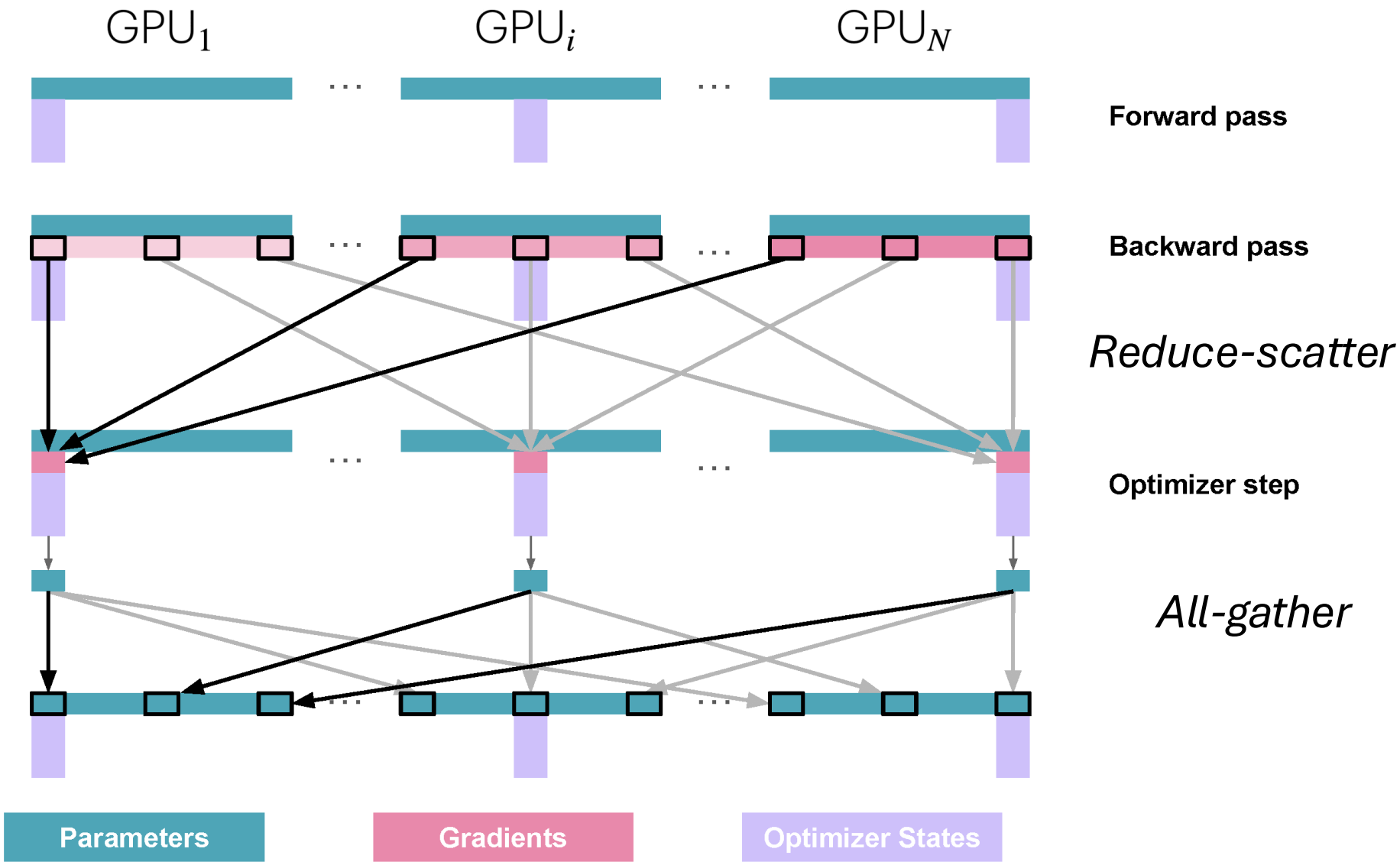


ZeRO-2

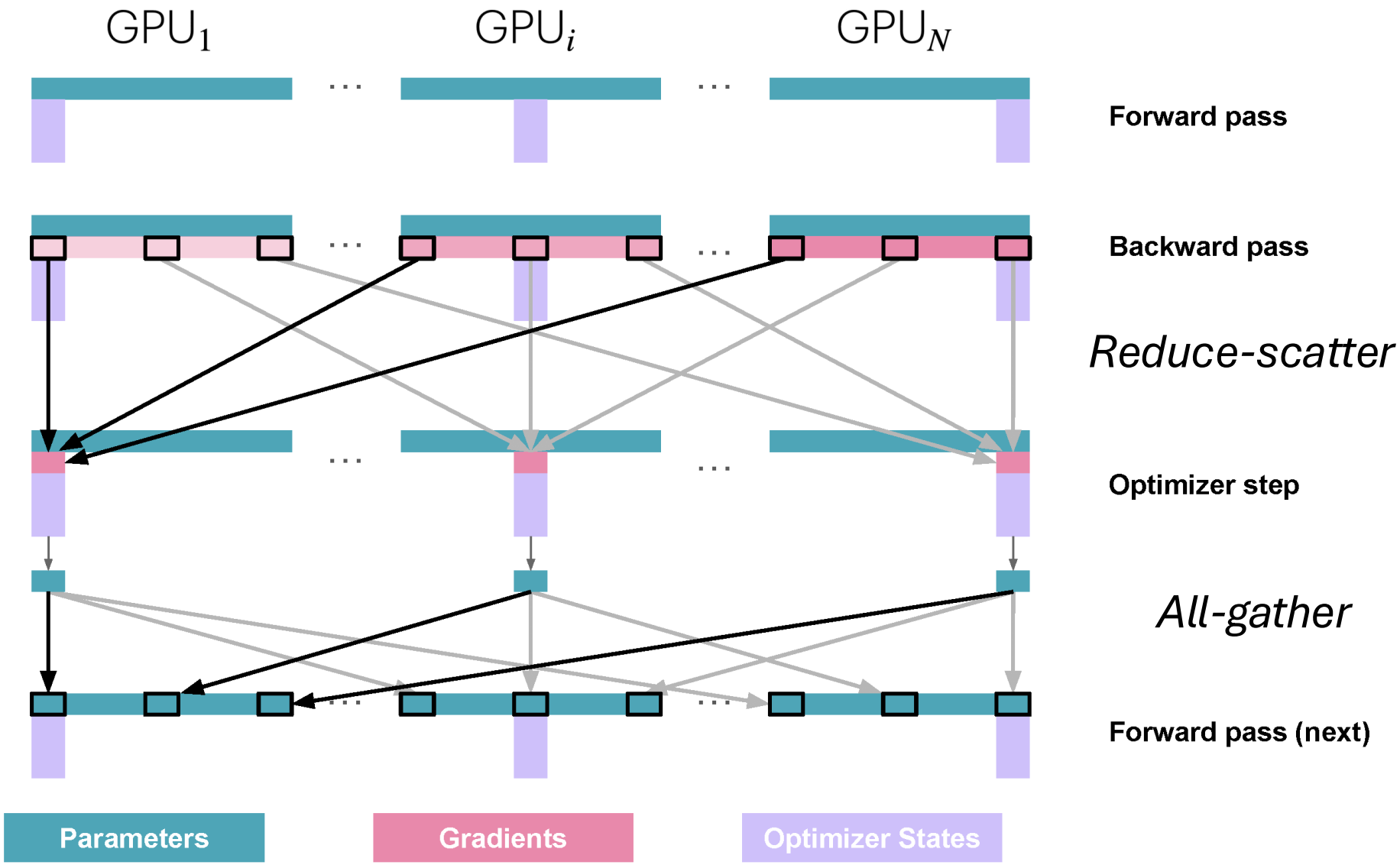


Parameters | Gradients | Optimizer States

ZeRO-2

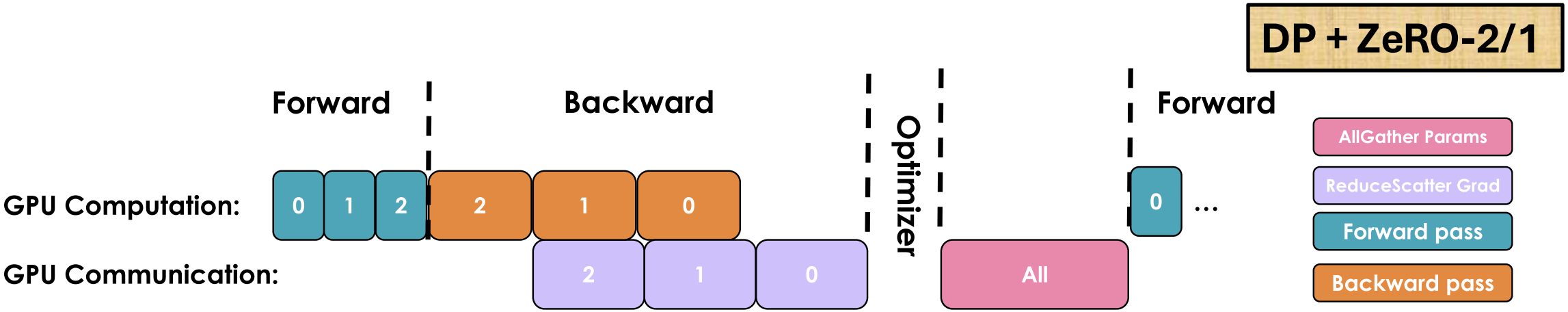

GPU_1
GPU_i
GPU_N
Forward pass
Backward pass
Reduce-scatter
Optimizer step
All-gather
Parameters
Gradients
Optimizer States

ZeRO-2

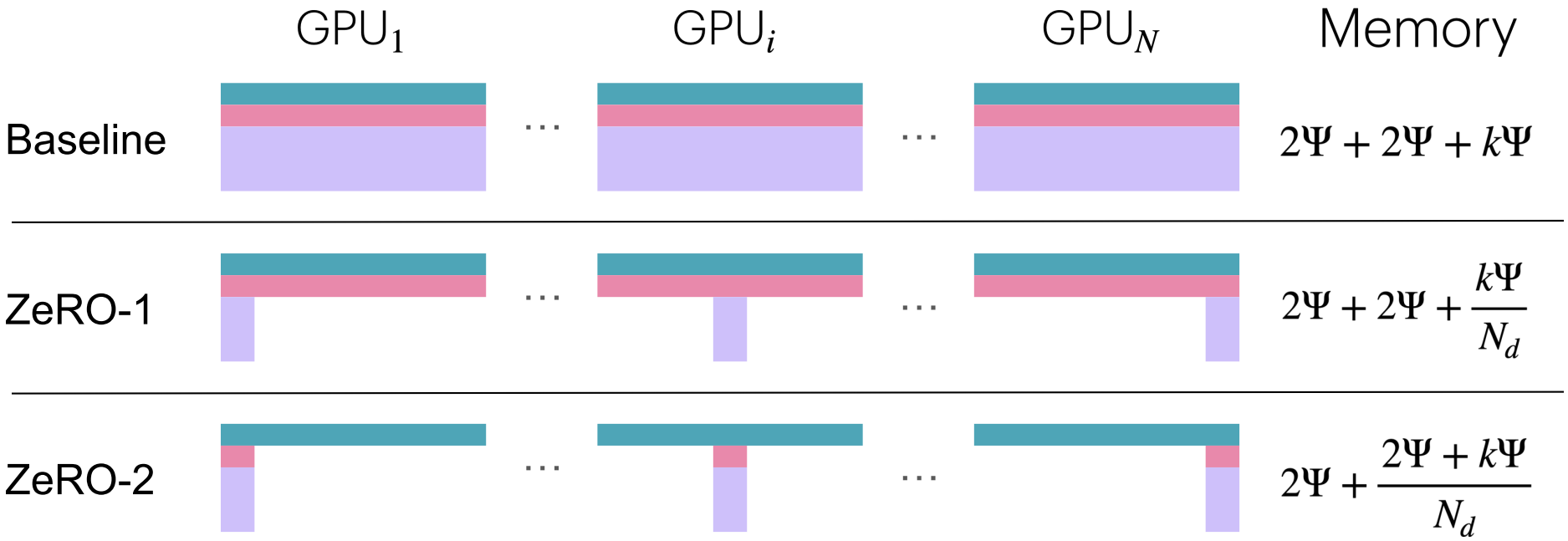

GPU_1
GPU_i
GPU_N
Forward pass
Backward pass
Reduce-scatter
Optimizer step
All-gather
Forward pass (next)
Parameters
Gradients
Optimizer States

Computation-communication timeline

- ZeRO-1: We keep a copy of all gradients
- ZeRO-2: communicate and release the gradients on the fly
- In practice, both use '*reduce-scatter*' for gradients and '*all-gather*' for FP32 copy of params
- There is no real overhead to using ZeRO-2 over ZeRO-1 besides implementation complexity, and indeed ZeRO-2 is usually the better option.



Can we ***shard*** gradients on GPUs?



Ψ : Model parameters

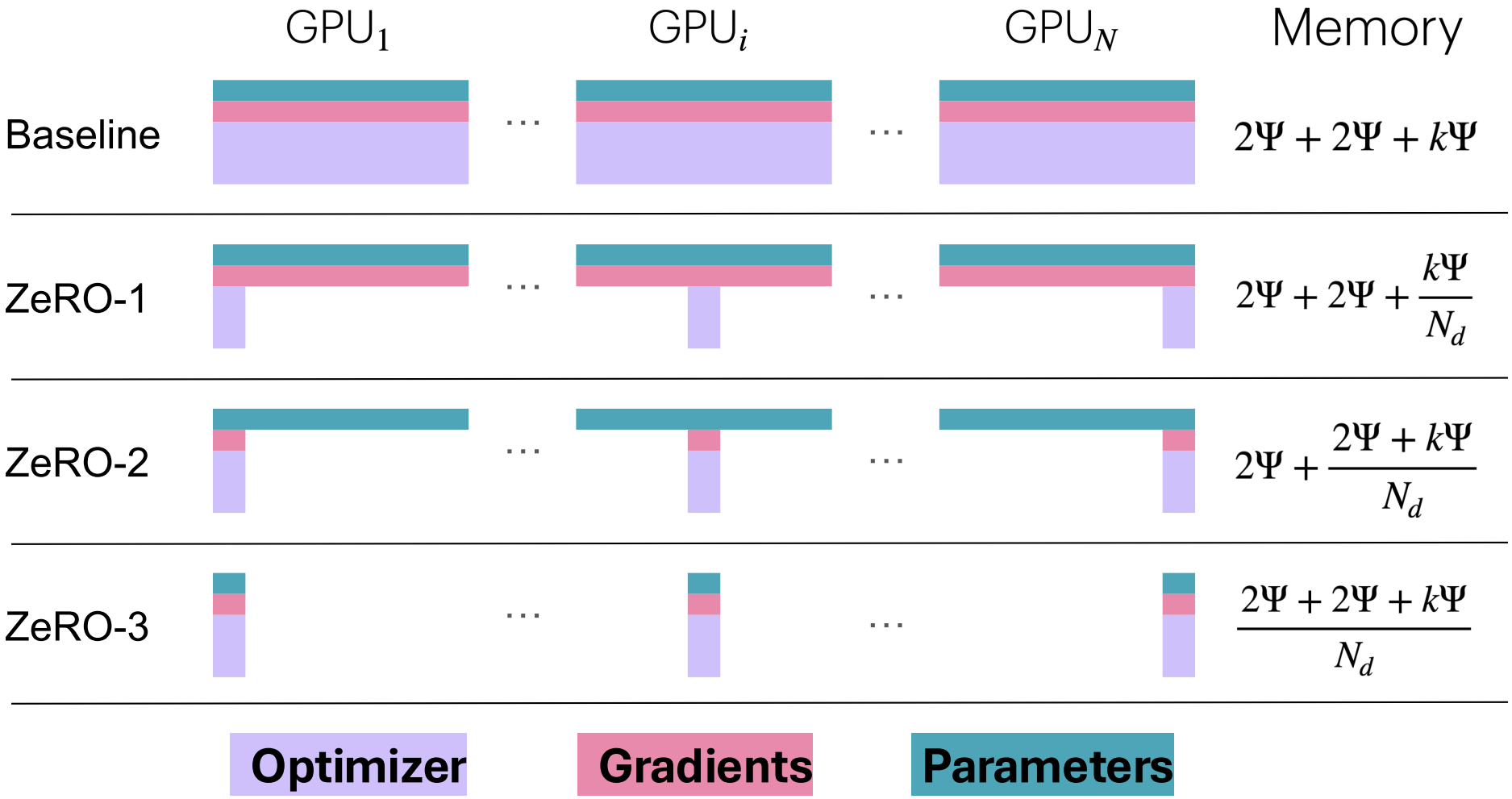
k : Optimizer multiplier

N_d: Data parallel degree

- Can we shard params as well?
- How would we run fwd/bwd passes if we don't have all the model weights?

Optimizer **Gradients** **Parameters**

Can we ***shard*** params on GPUs?

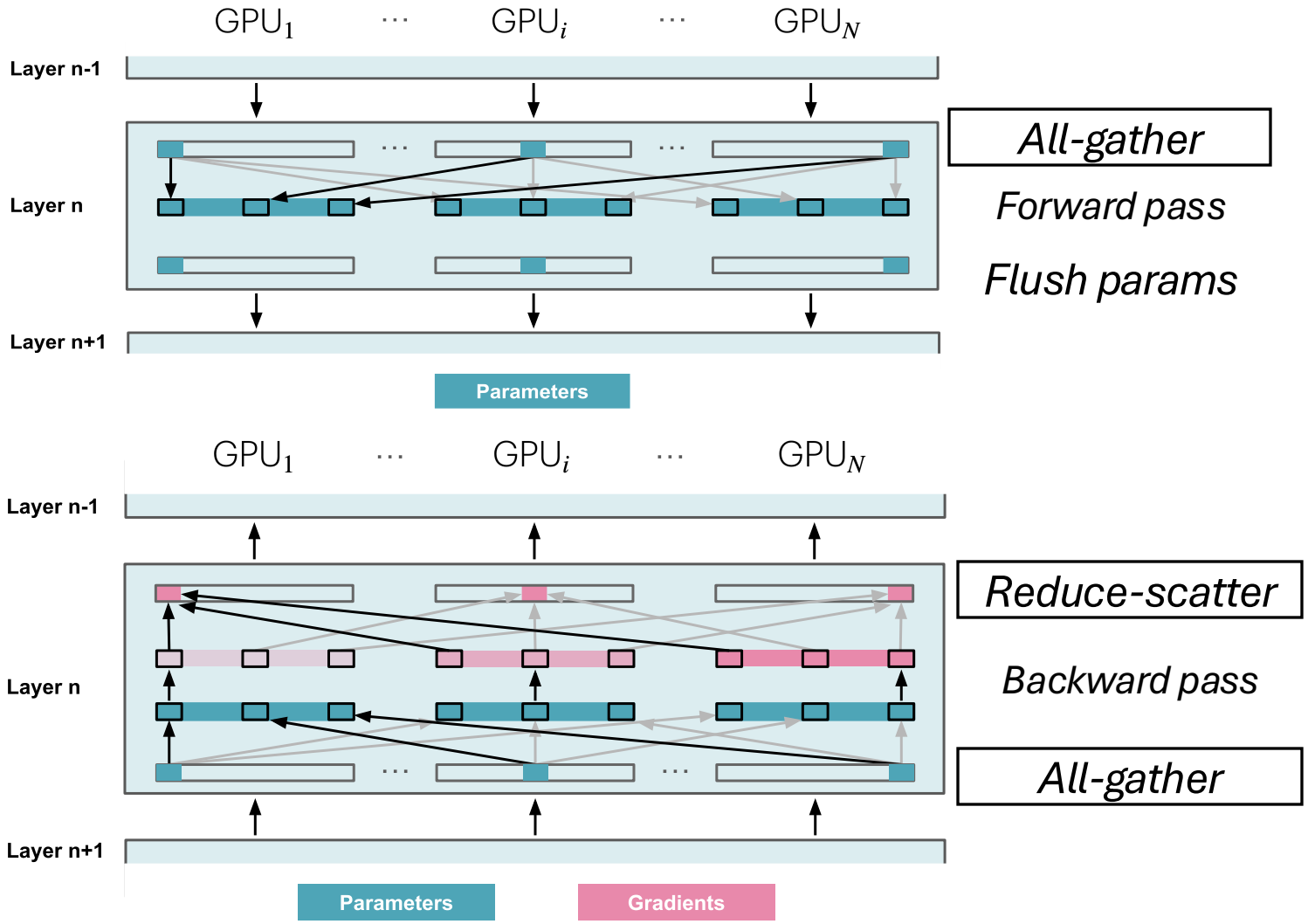


Ψ : Model parameters

k : Optimizer multiplier

N_d: Data parallel degree

ZeRO-3 / FSDP- *Fully Sharded Data Parallel*



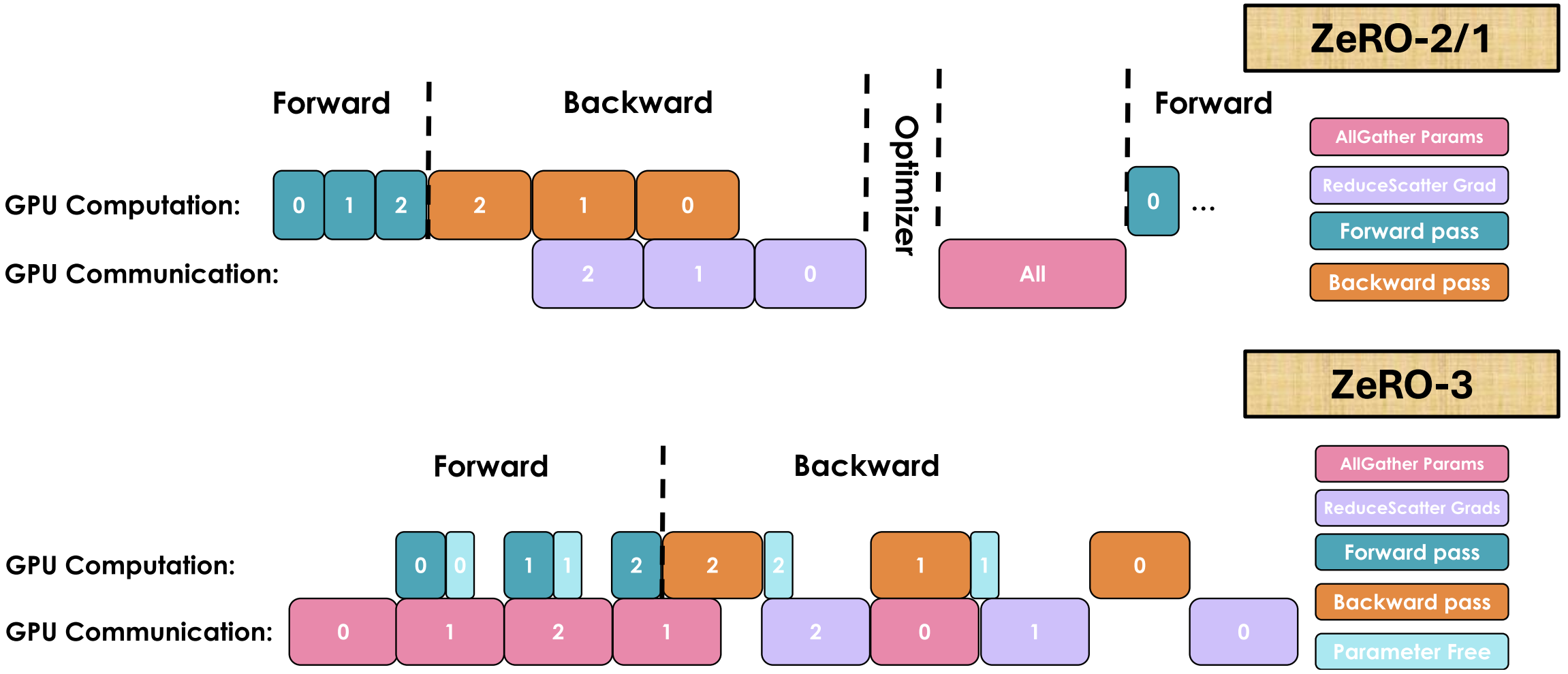
Forward Pass:

- Gather params on demand
- Flush them from memory when not needed

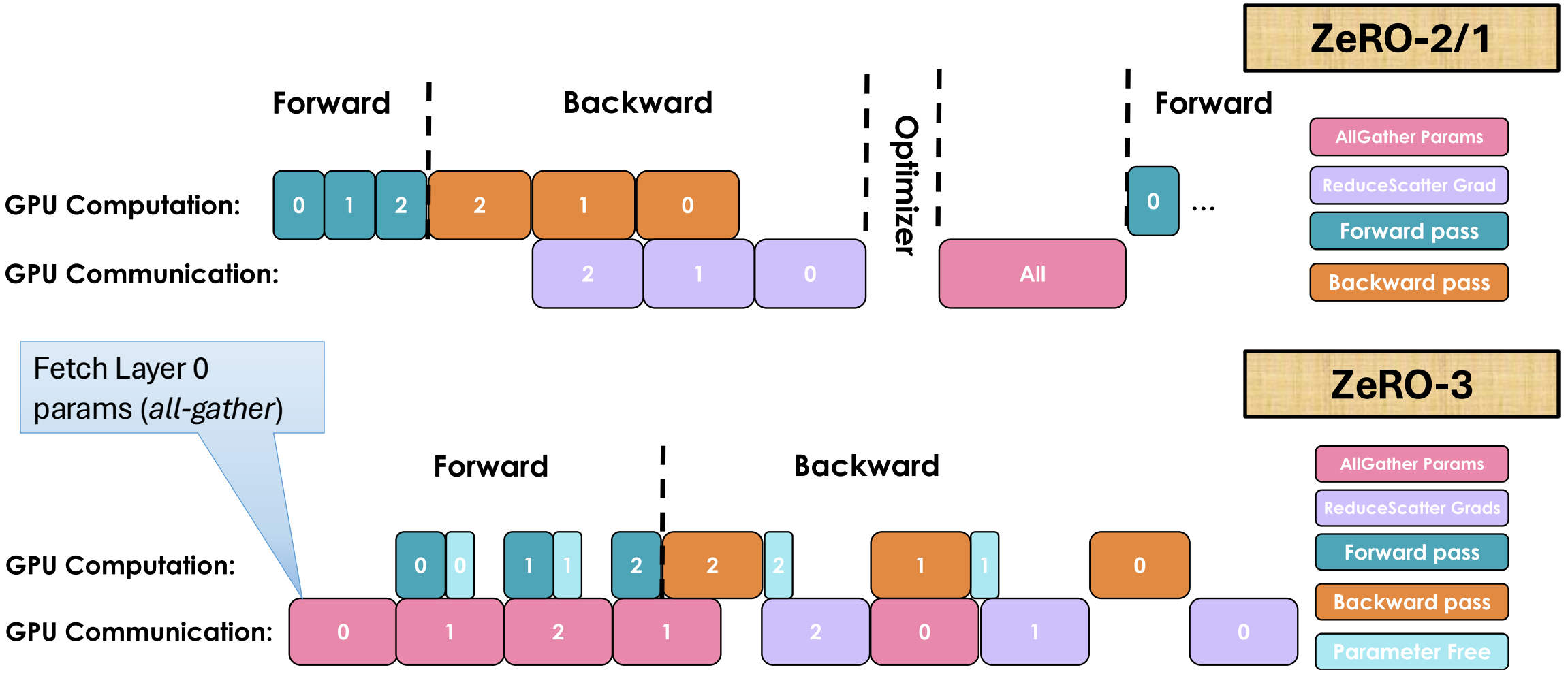
Backward Pass:

- Gather params on demand
- Reduce-scatter as in ZeRO-2

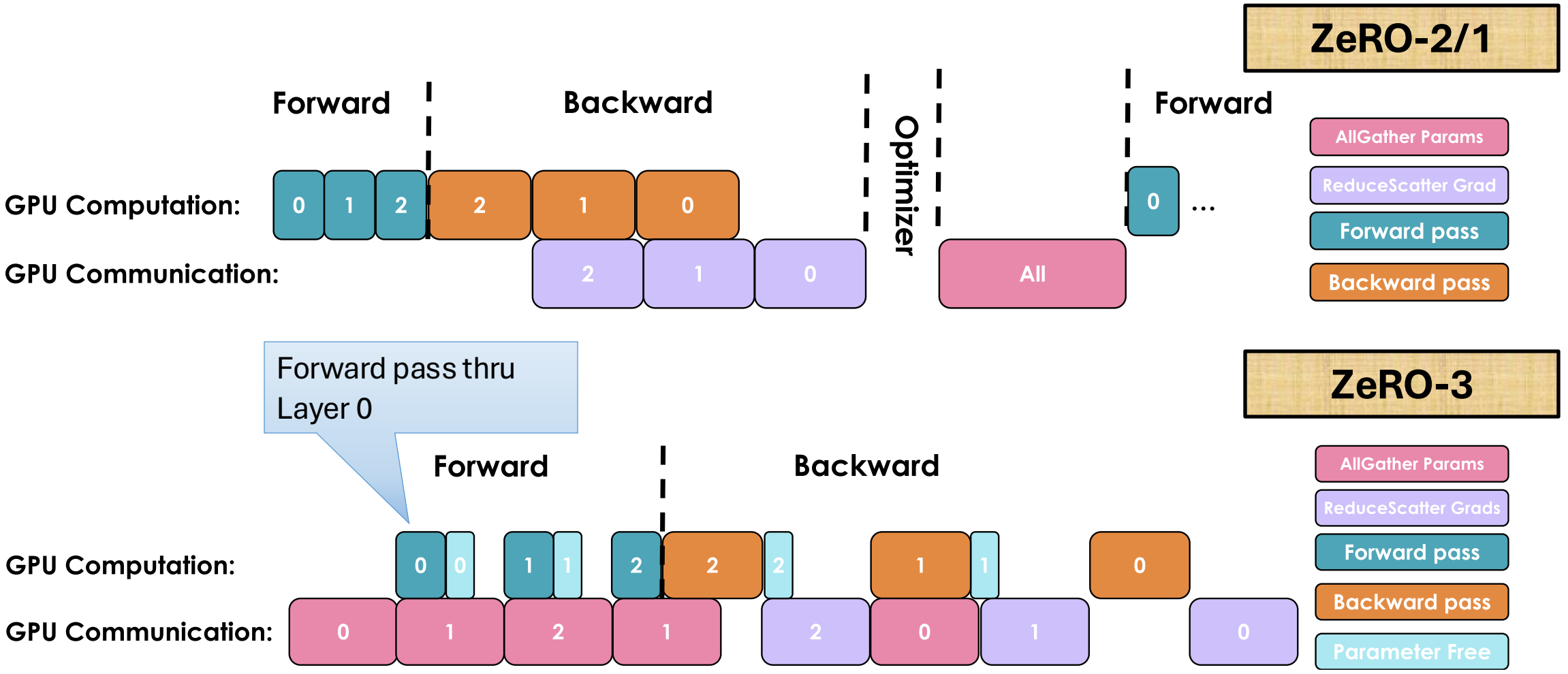
Increased communication cost in FSDP?



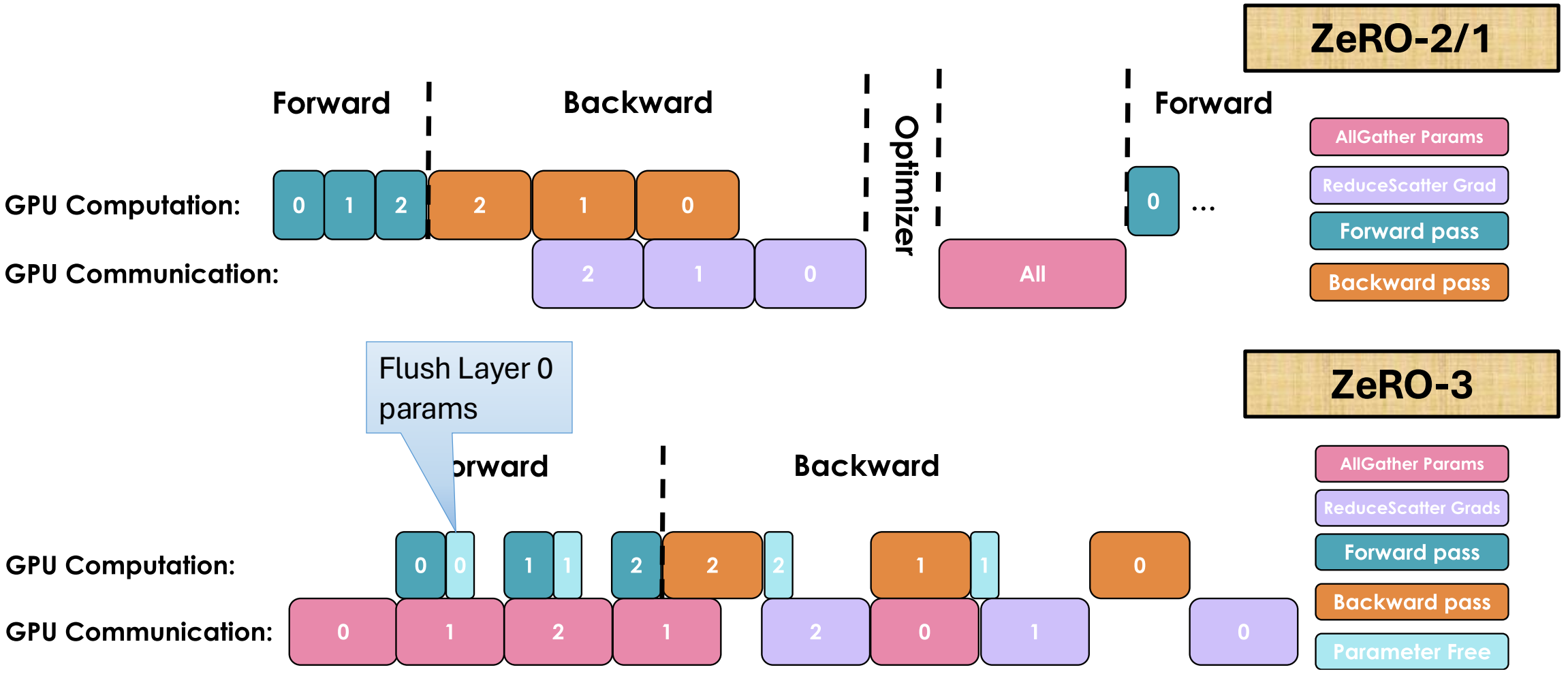
Increased communication cost in FSDP?



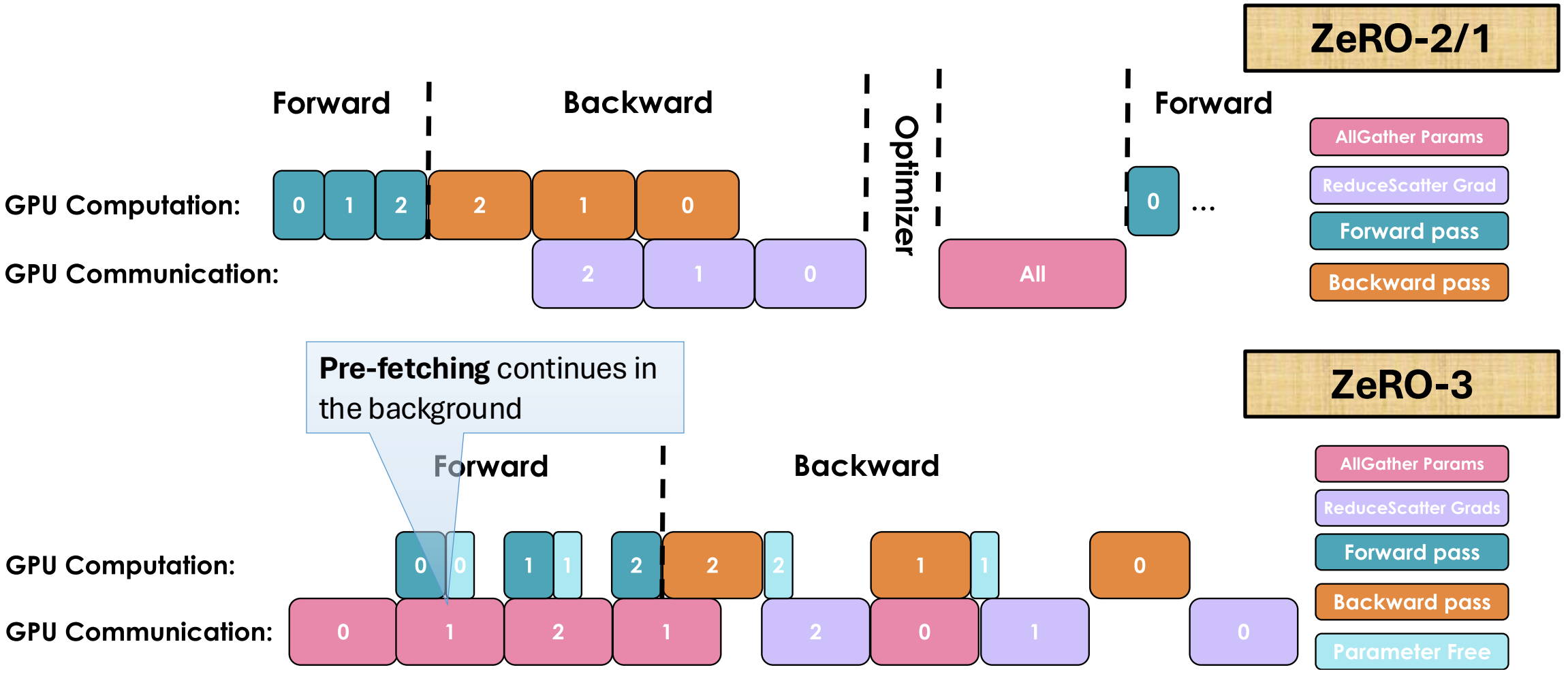
Increased communication cost in FSDP?



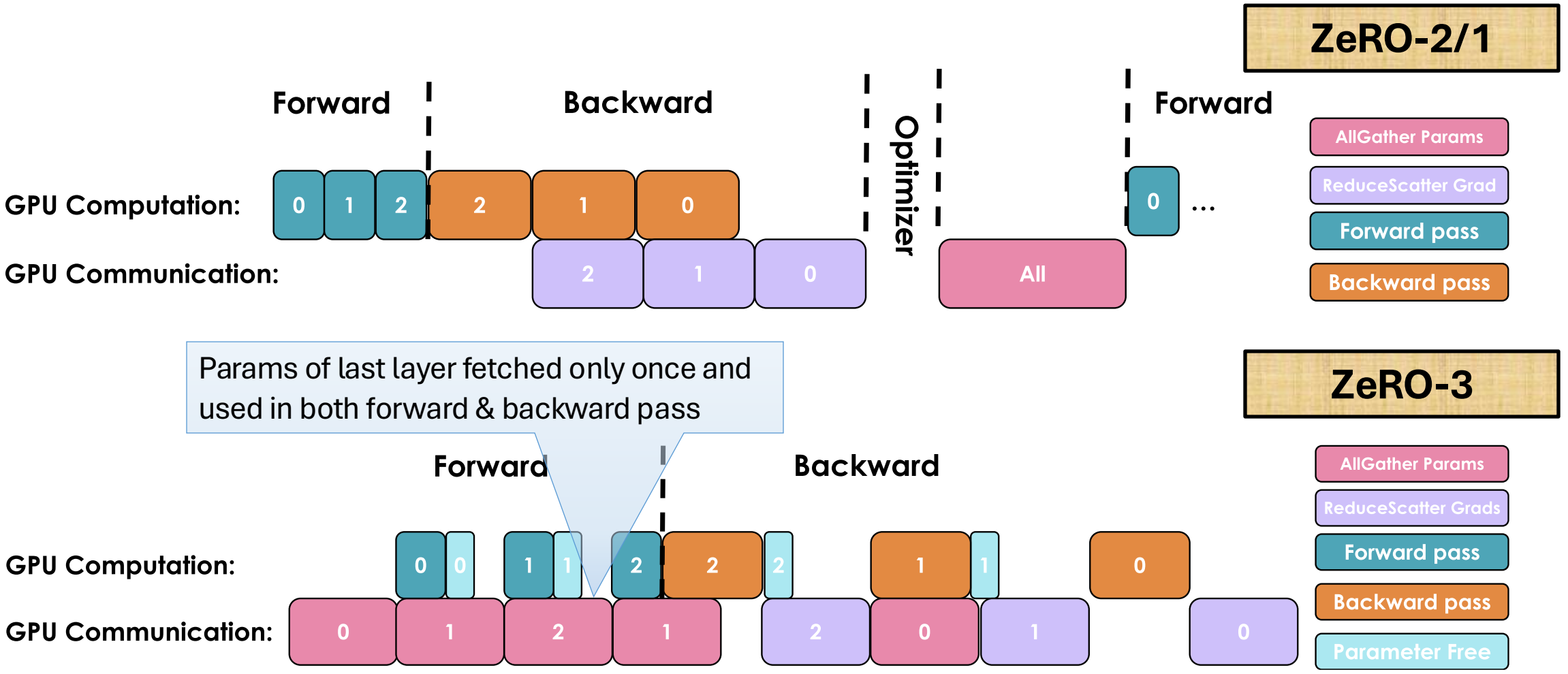
Increased communication cost in FSDP?



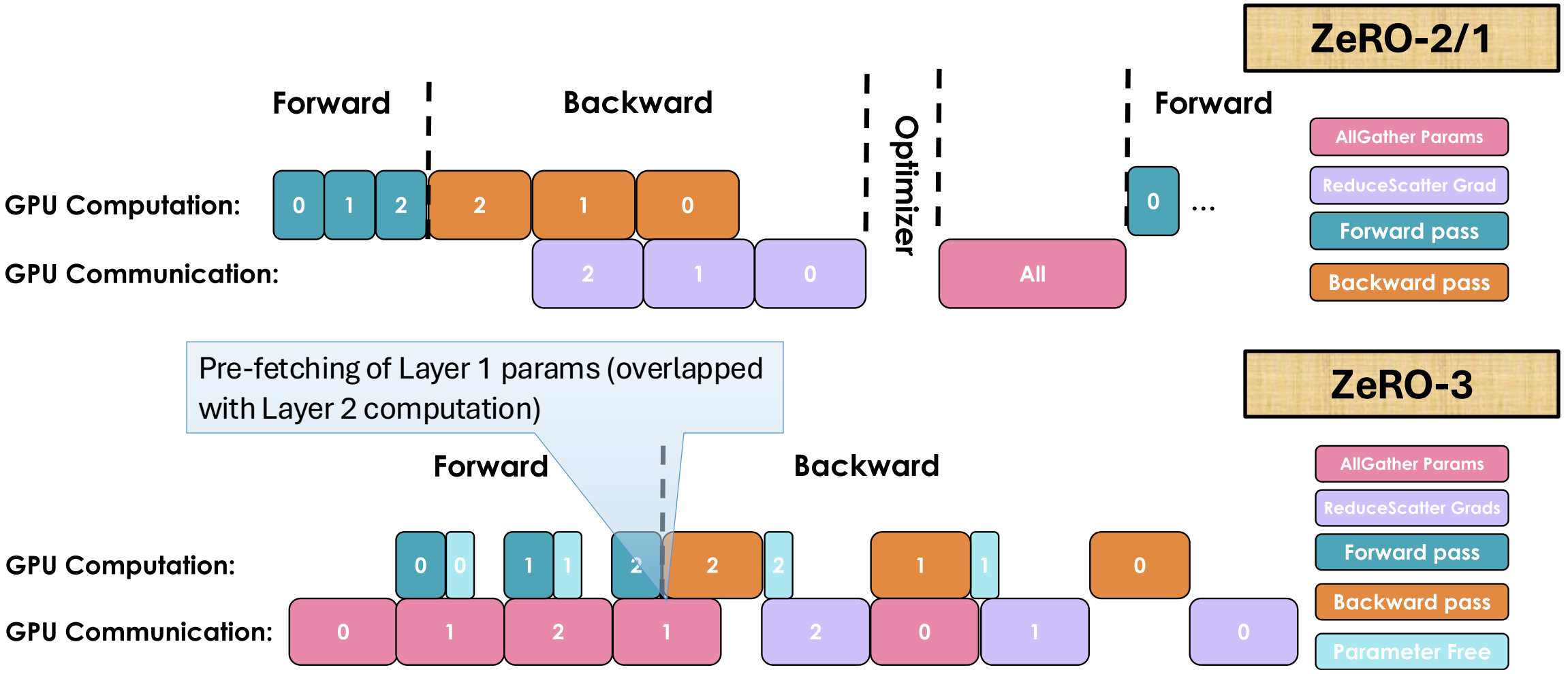
Increased communication cost in FSDP?



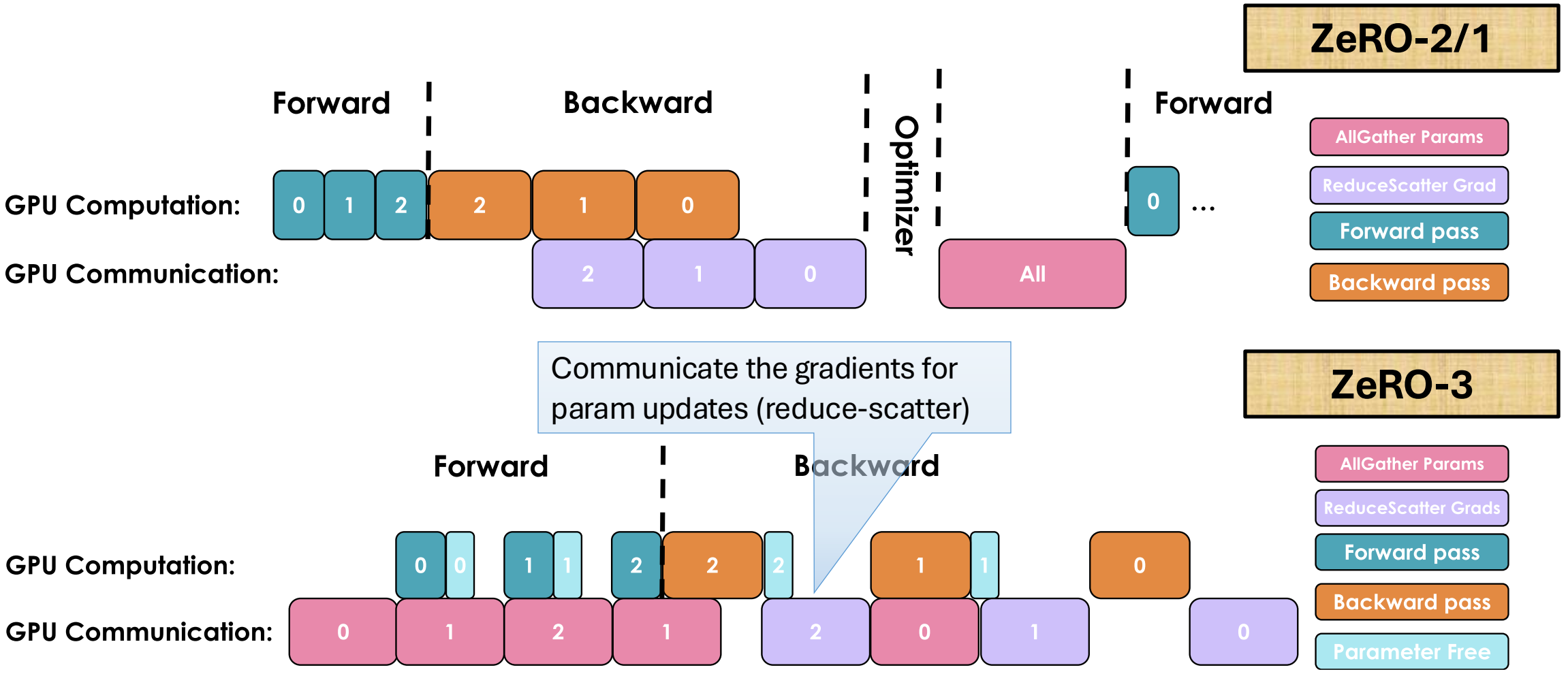
Increased communication cost in FSDP?



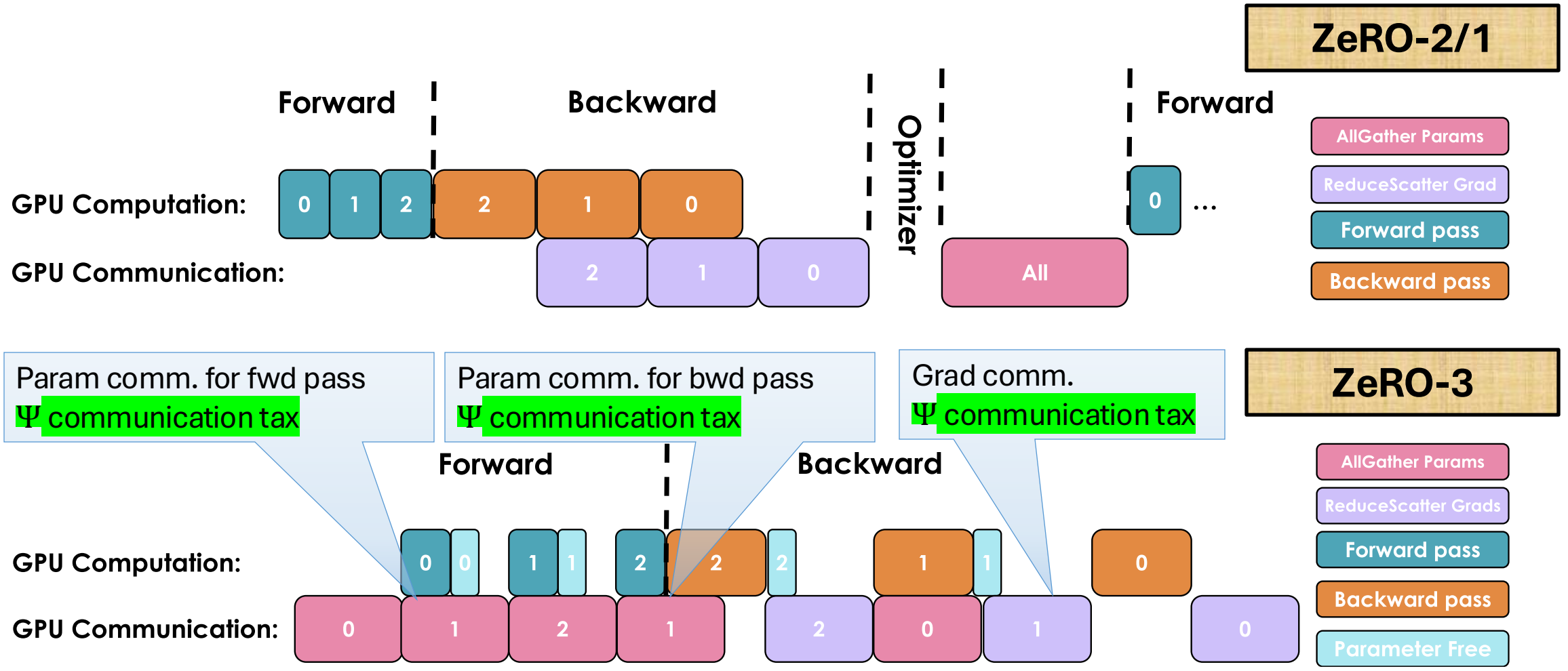
Increased communication cost in FSDP?



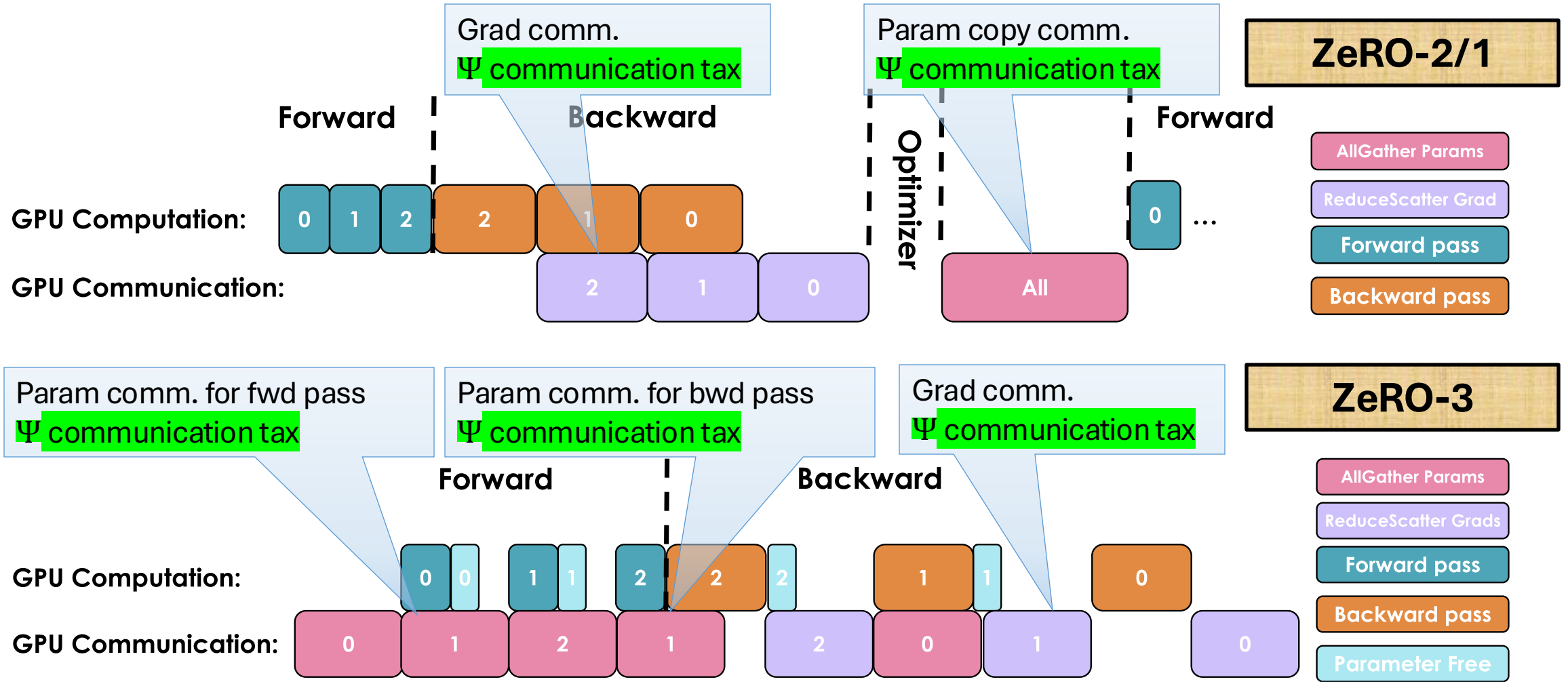
Increased communication cost in FSDP?



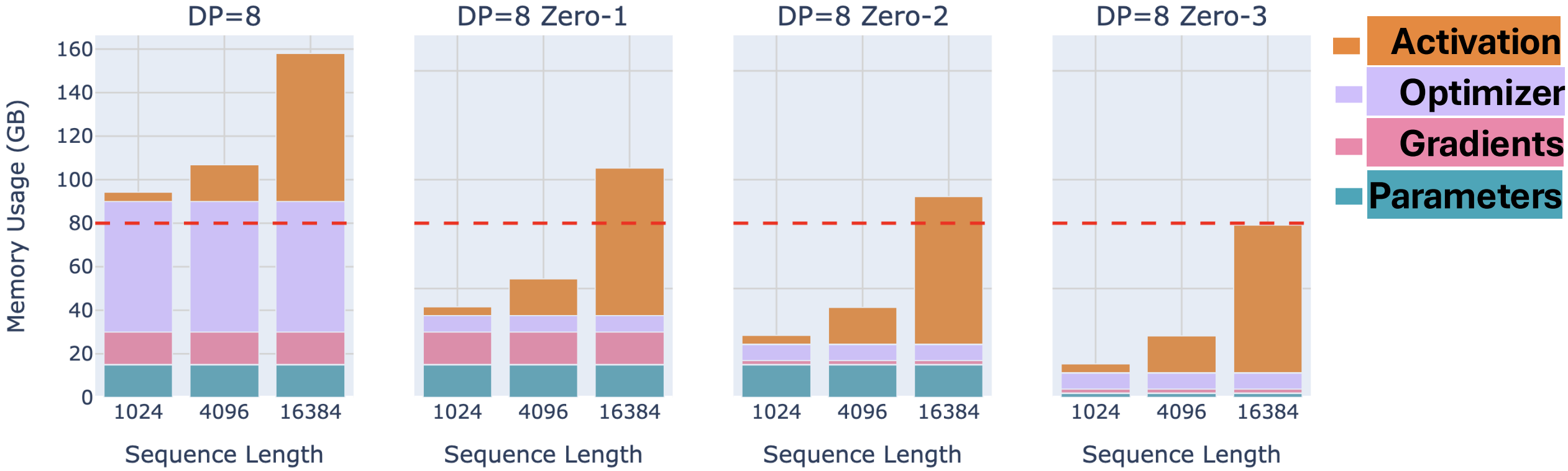
Increased communication cost in FSDP?



Increased communication cost in FSDP?



Comparing memory usage for 8B Model



Summary

	Vanilla DP	ZeRO-1	ZeRO-2	ZeRO-3
Assumptions				
Parallelizes /Shards				
Memory (excluding activations)				
Communication Tax				

Summary

	Vanilla DP	ZeRO-1	ZeRO-2	ZeRO-3
Assumptions	1 seq. act. + all params + all grad. + all optim. fit on 1 GPU	1 seq. act. + all params + all grad. + $(1/N_d)$ optim fit on 1 GPU	1 seq. act. + all params + $(1/N_d)$ grad. + $(1/N_d)$ optim fit on 1 GPU	1 seq. act. + $(1/N_d)$ params + $(1/N_d)$ grad. + $(1/N_d)$ optim fit on 1 GPU
Parallelizes /Shards				
Memory (excluding activations)				
Communication Tax				

Summary

	Vanilla DP	ZeRO-1	ZeRO-2	ZeRO-3
Assumptions	1 seq. act. + all params + all grad. + all optim. fit on 1 GPU	1 seq. act. + all params + all grad. + $(1/N_d)$ optim fit on 1 GPU	1 seq. act. + all params + $(1/N_d)$ grad. + $(1/N_d)$ optim fit on 1 GPU	1 seq. act. + $(1/N_d)$ params + $(1/N_d)$ grad. + $(1/N_d)$ optim fit on 1 GPU
Parallelizes /Shards	Batch of samples	Batch + Optim. states	Batch + Optim. + Grads.	Batch + Optim. + Grads. + Params
Memory (excluding activations)				
Communication Tax				

Summary

	Vanilla DP	ZeRO-1	ZeRO-2	ZeRO-3
Assumptions	1 seq. act. + all params + all grad. + all optim. fit on 1 GPU	1 seq. act. + all params + all grad. + $(1/N_d)$ optim fit on 1 GPU	1 seq. act. + all params + $(1/N_d)$ grad. + $(1/N_d)$ optim fit on 1 GPU	1 seq. act. + $(1/N_d)$ params + $(1/N_d)$ grad. + $(1/N_d)$ optim fit on 1 GPU
Parallelizes /Shards	Batch of samples	Batch + Optim. states	Batch + Optim. + Grads.	Batch + Optim. + Grads. + Params
Memory (excluding activations)	$2\Psi + 2\Psi + 12\Psi$	$2\Psi + 2\Psi + \frac{12\Psi}{N_d}$	$2\Psi + \frac{2\Psi + 12\Psi}{N_d}$	$\frac{2\Psi + 2\Psi + 12\Psi}{N_d}$
Communication Tax				

Summary

	Vanilla DP	ZeRO-1	ZeRO-2	ZeRO-3
Assumptions	1 seq. act. + all params + all grad. + all optim. fit on 1 GPU	1 seq. act. + all params + all grad. + $(1/N_d)$ optim fit on 1 GPU	1 seq. act. + all params + $(1/N_d)$ grad. + $(1/N_d)$ optim fit on 1 GPU	1 seq. act. + $(1/N_d)$ params + $(1/N_d)$ grad. + $(1/N_d)$ optim fit on 1 GPU
Parallelizes /Shards	Batch of samples	Batch + Optim. states	Batch + Optim. + Grads.	Batch + Optim. + Grads. + Params
Memory (excluding activations)	$2\Psi + 2\Psi + 12\Psi$	$2\Psi + 2\Psi + \frac{12\Psi}{N_d}$	$2\Psi + \frac{2\Psi + 12\Psi}{N_d}$	$\frac{2\Psi + 2\Psi + 12\Psi}{N_d}$
Communication Tax	Ψ (grad. all-reduce)	2Ψ (grad. reduce scatter + params all-gather)	2Ψ	3Ψ

Summary

What if activations for one sequence do not fit on one GPU?

	Vanilla DP	ZeRO-1	ZeRO-2	ZeRO-3
Assumptions	1 seq. act. + all params + all grad. + all optim. fit on 1 GPU	1 seq. act. + all params + all grad. + $(1/N_d)$ optim fit on 1 GPU	1 seq. act. + all params + $(1/N_d)$ grad. + $(1/N_d)$ optim fit on 1 GPU	1 seq. act. + $(1/N_d)$ params + $(1/N_d)$ grad. + $(1/N_d)$ optim fit on 1 GPU
Parallelizes /Shards	Batch of samples	Batch + Optim. states	Batch + Optim. + Grads.	Batch + Optim. + Grads. + Params
Memory (excluding activations)	$2\Psi + 2\Psi + 12\Psi$	$2\Psi + 2\Psi + \frac{12\Psi}{N_d}$	$2\Psi + \frac{2\Psi + 12\Psi}{N_d}$	$\frac{2\Psi + 2\Psi + 12\Psi}{N_d}$
Communication Tax	Ψ (grad. all-reduce)	2Ψ (grad. reduce scatter + params all-gather)	2Ψ	3Ψ