AI SWEDEN

Building LLMs in Practice

Part 2

Felix Stollenwerk
2025-05-26

AGENDA

I. Model Training Basics

II. Basic strategies:
- Gradient accumulation
- Activation recomputation
- Data parallelism
- Mixed Precision

III. Benchmarking, finding a training configuration

Many examples and illustrations taken from the excellent ultra-scale playbook:
https://huggingface.co/spaces/nanotron/ultrascale-playbook

AGENDA

IV. Multi-GPU Training

- Data Parallelism (DP)
- Fully Sharded Data Parallelism (FSDP)
- Tensor Parallelism (TP)
- Pipeline Parallelism (PP)

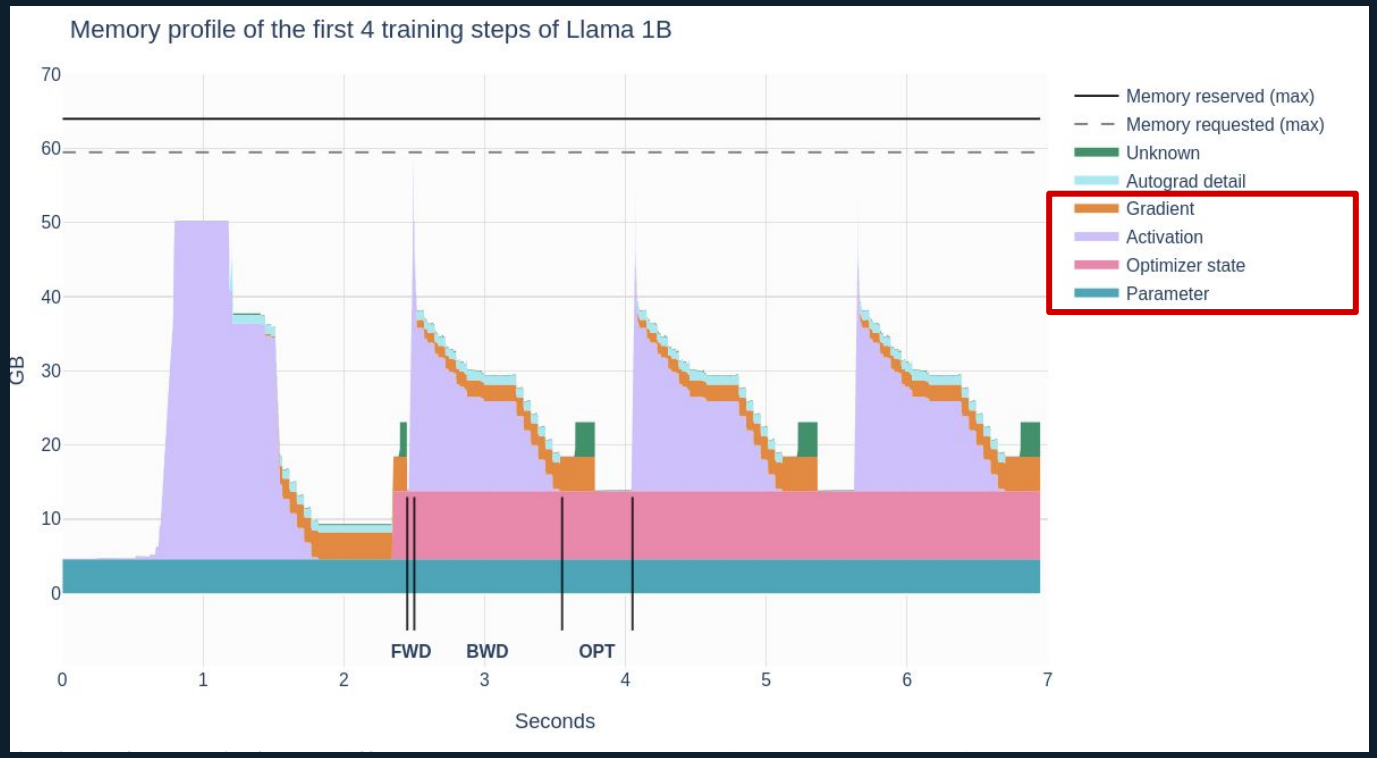
V. HPC in Europe

VI. Training Frameworks

Part IV

Multi-GPU Training

Recap: Memory Footprint



Mitigation Strategies (Single GPU)

- Activation Checkpointing
- Gradient Accumulation

Acceleration Strategies (Multi-GPU)

- Data Parallelism

Recap: Data Parallelism

- Gradient Accumulation

  B = b x grad_acc

Recap: Data Parallelism

AI SWEDEN

- Gradient Accumulation Parallelized = Data Parallelism

  B = b x grad_acc x DP

Recap: Data Parallelism

AI SWEDEN

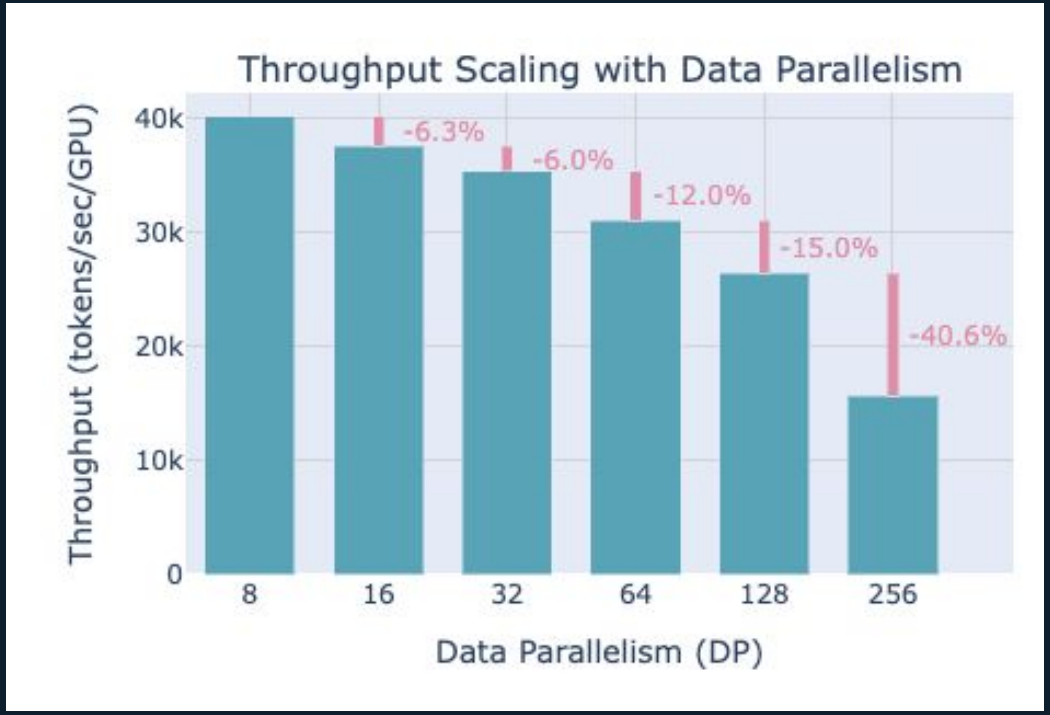
- Gradient Accumulation Parallelized = Data Parallelism

  $B = b \times grad\_acc \times DP$

- How to find the optimal DP setup in practice

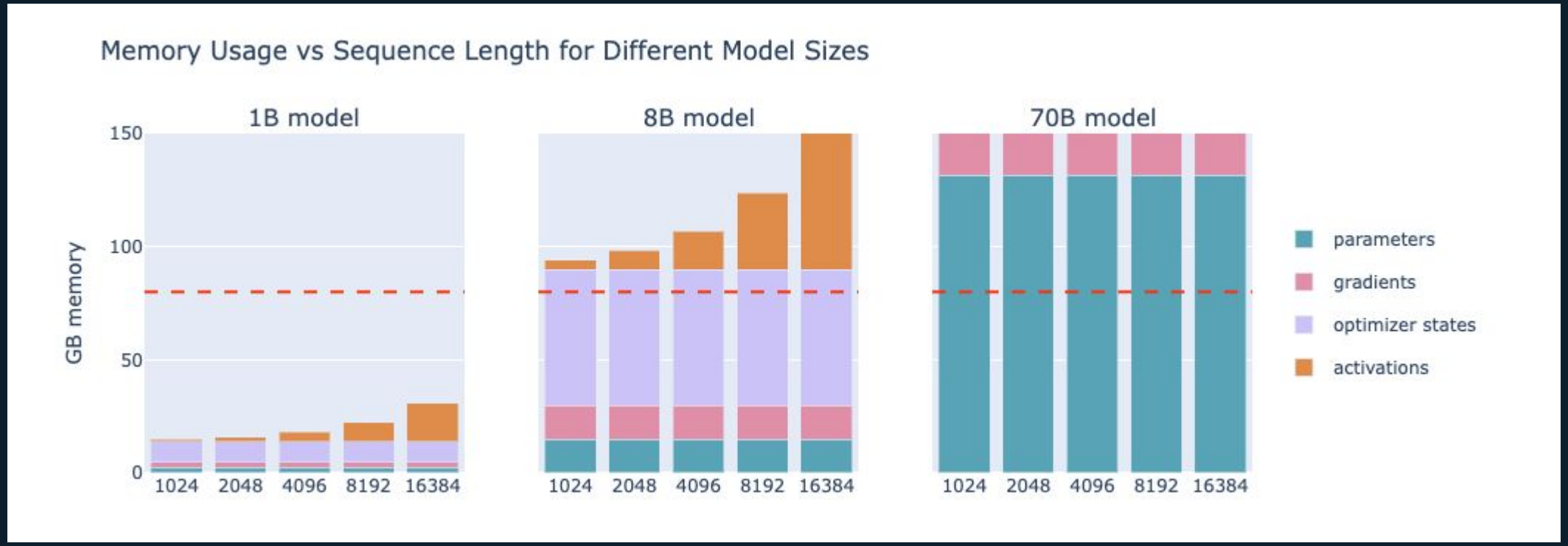
#	Step	Restrictions	Example
1	Fix global batch size B		1024 (samples)
2	Maximize micro batch size b	GPU memory, divide B	2 (samples)
3	Maximize data parallelism DP	#GPUs, divide B / b	128
4	Determine grad_acc = B / (b * DP)		4

DP Limitations



1) DP provides a communication overhead
   ⇒ becomes inefficient if #GPUs is large

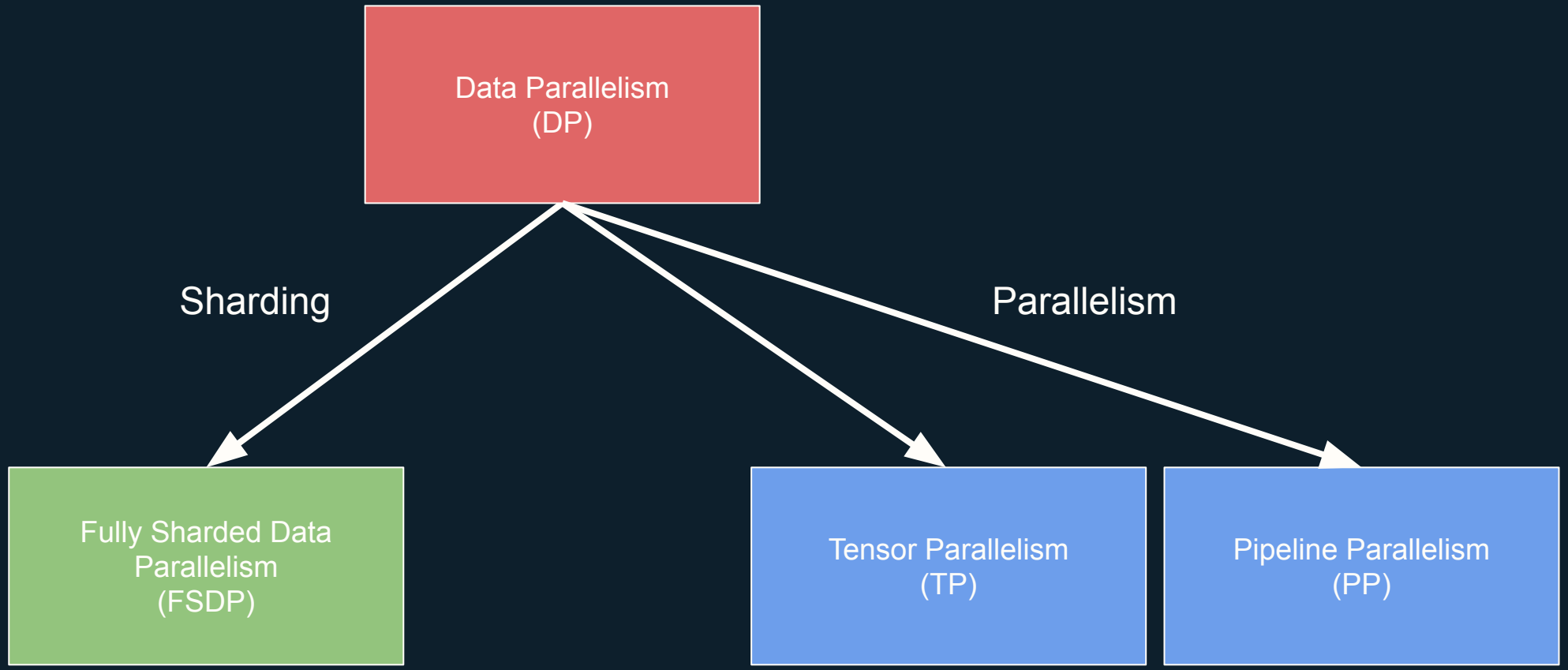
DP Limitations



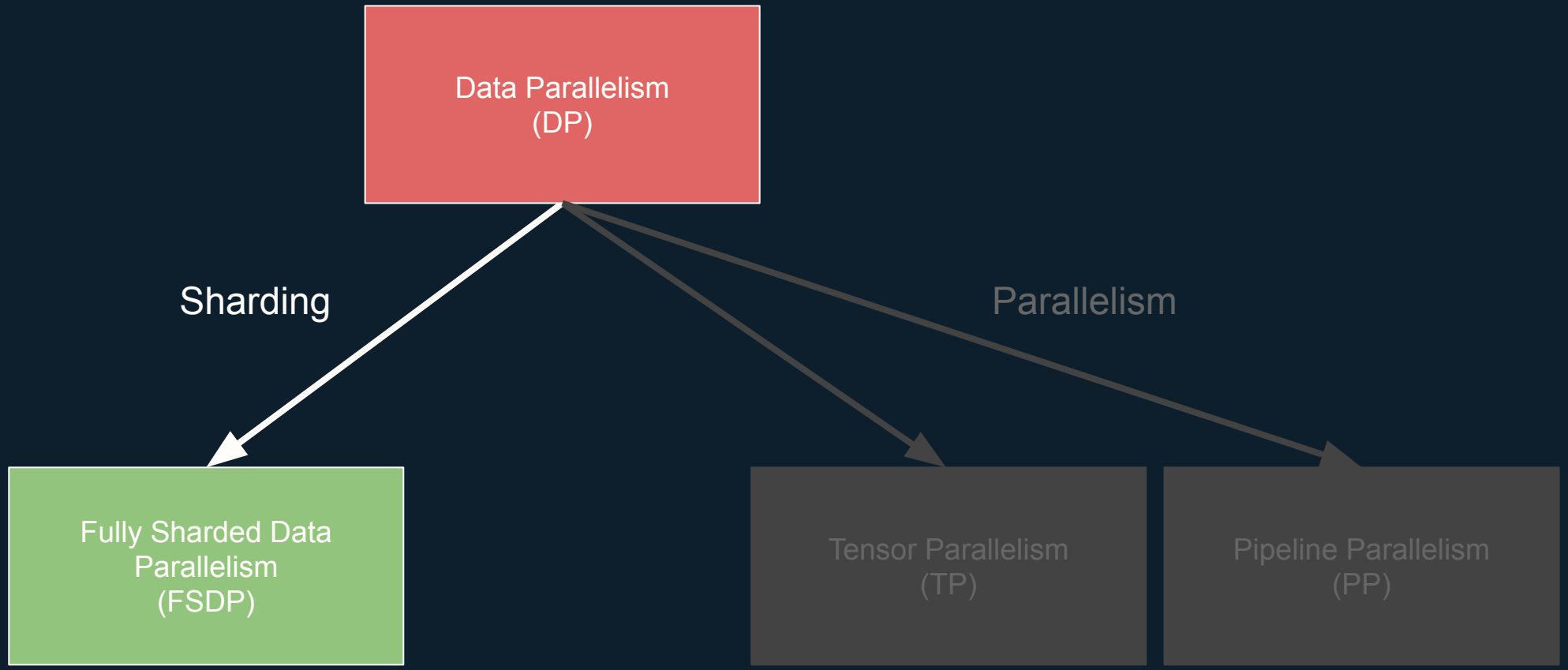
2) DP reduces the memory footprint for activations, not model-related tensors

⇒ becomes infeasible if model size is large

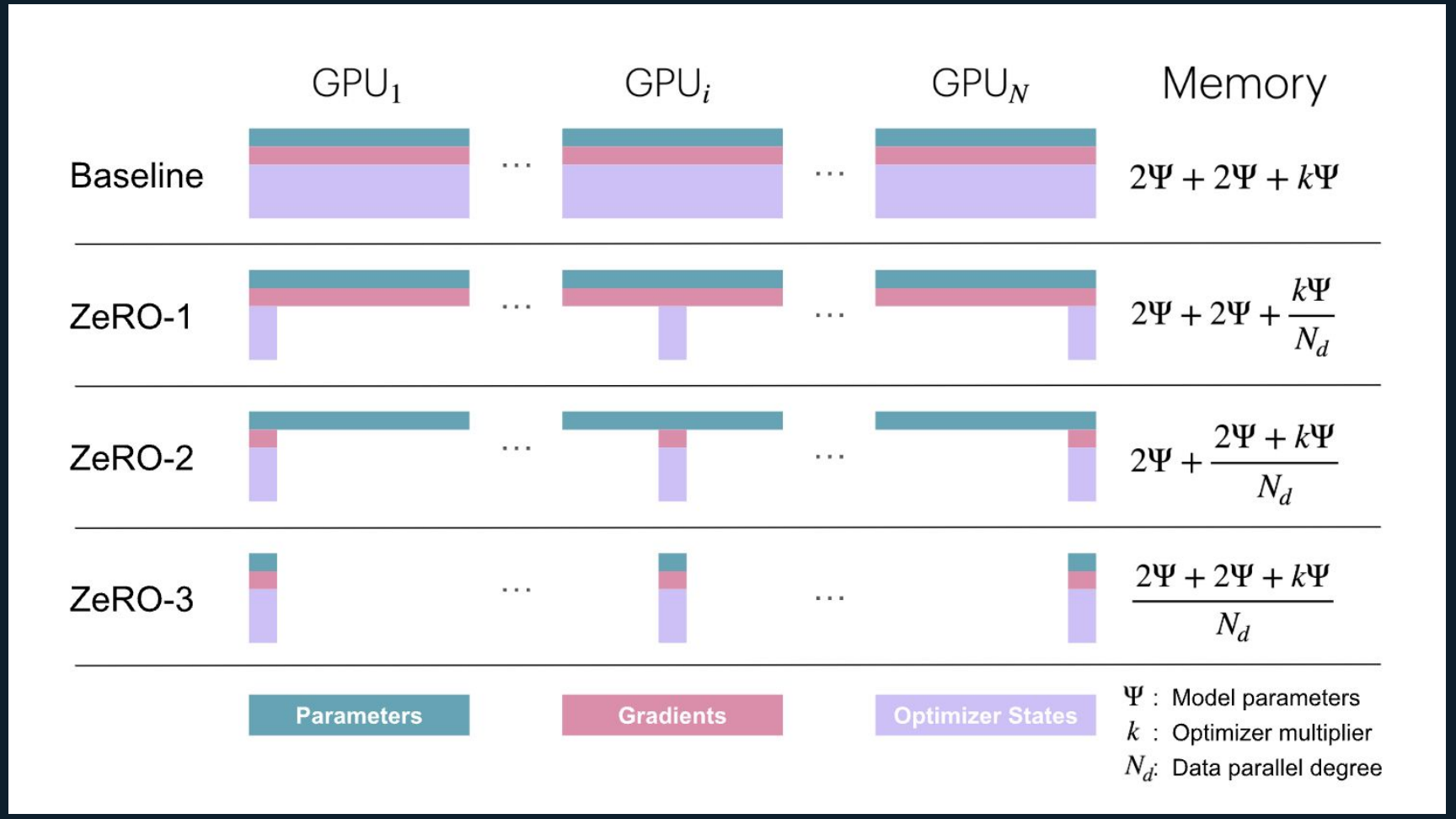
Sharding & Parallelism



Sharding & Parallelism



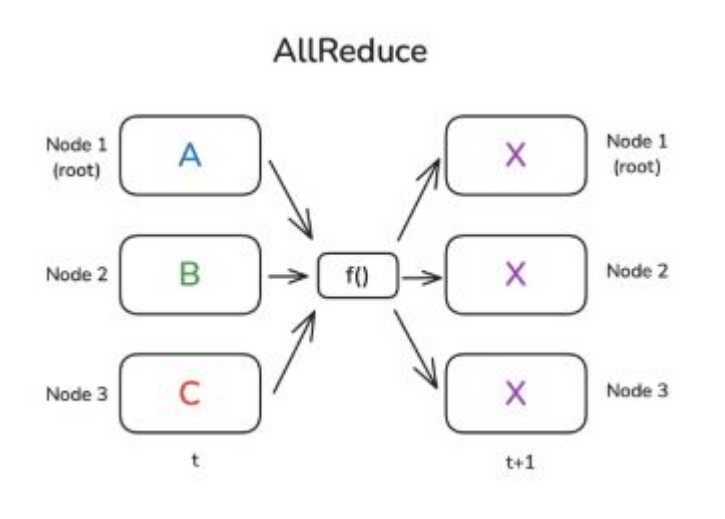
Sharding Overview

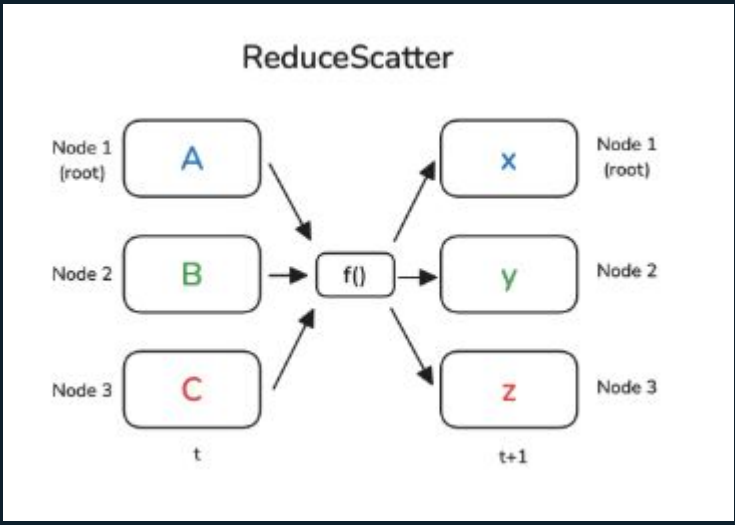


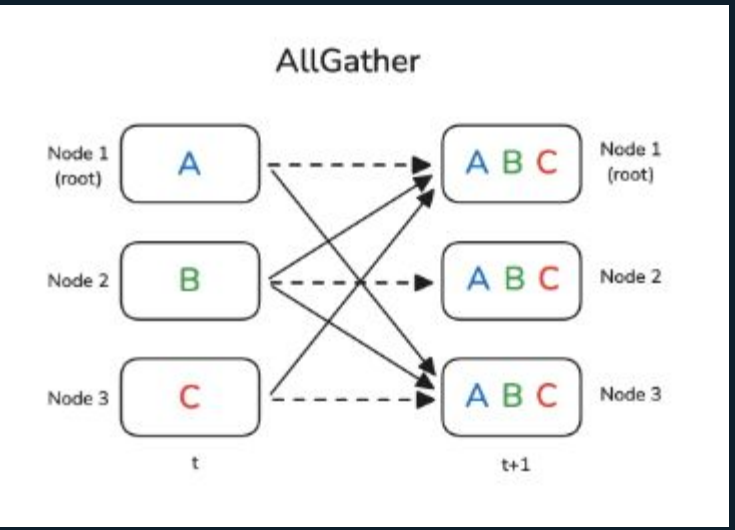
FSDP

Fully Sharded Data Parallel

Interlude: Collective Operations

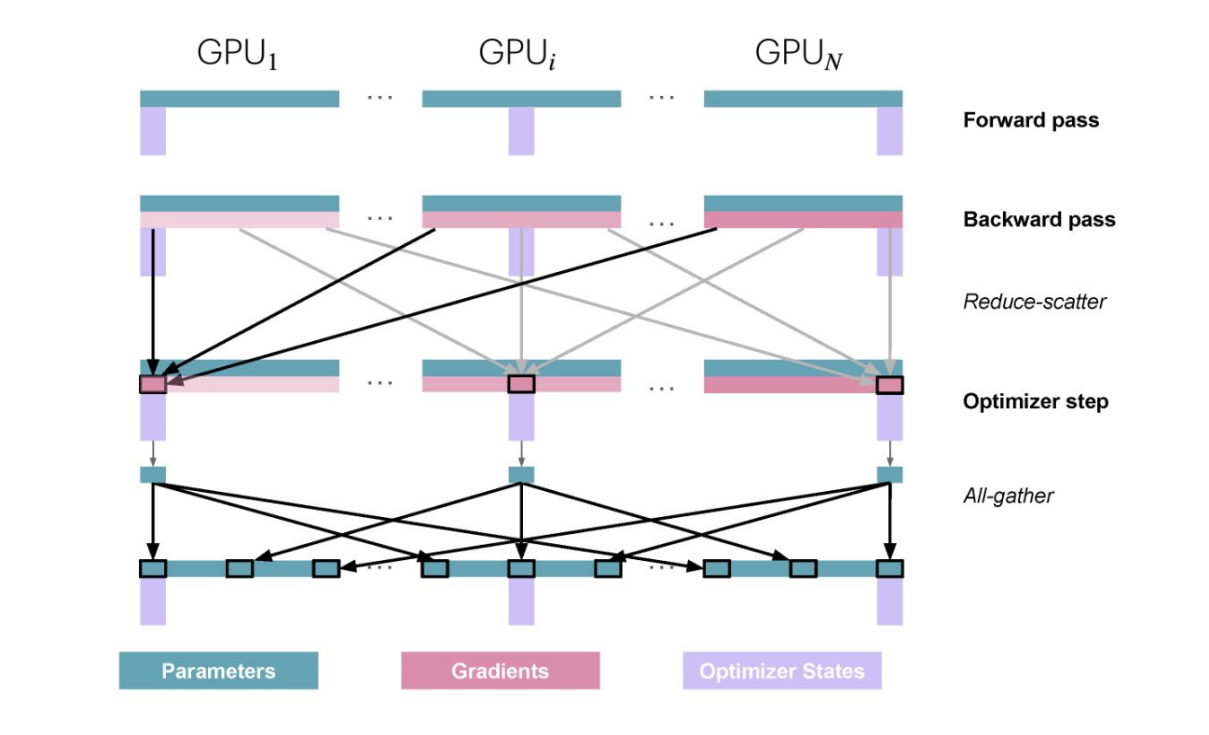






ZeRO-1

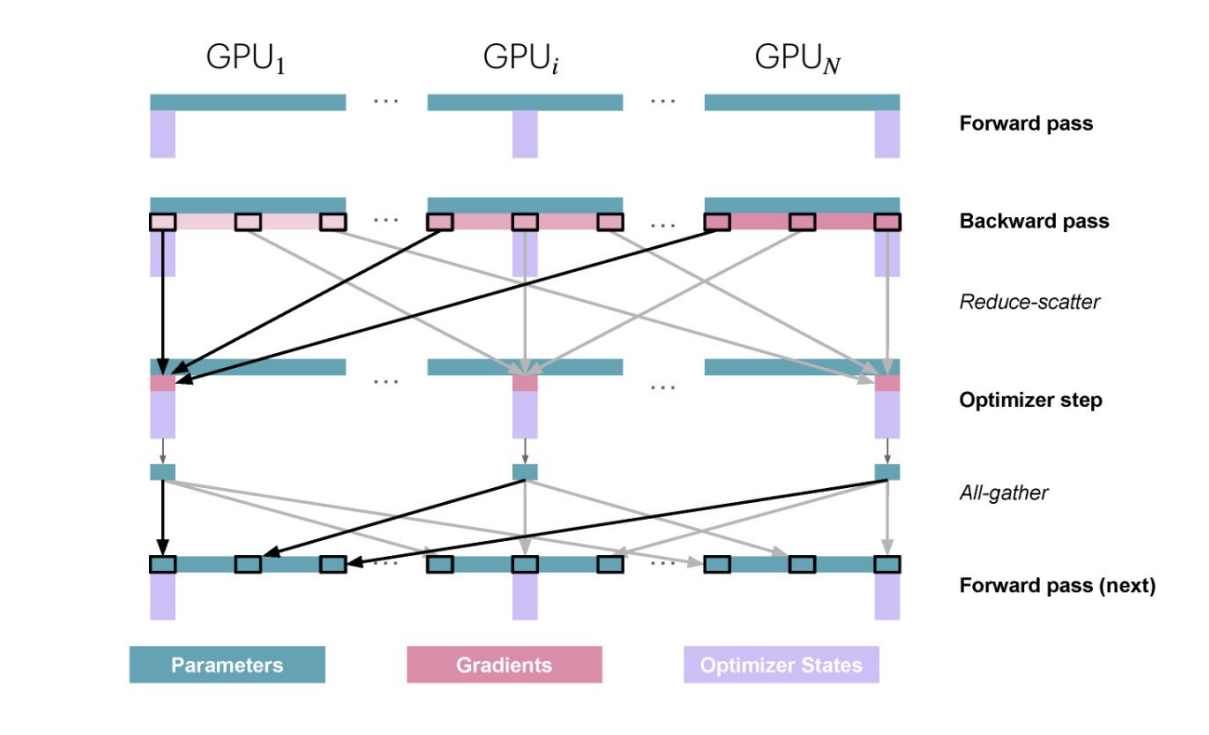
Optimizer State Sharding



1. Perform a **forward pass** with the same full set of parameters on each replica, but different micro-batches across replicas.
2. Perform a **backward pass** with the same full set of gradients on each replica, but different micro-batches across replicas.
3. Perform a **reduce-scatter** on the gradients.
4. Each replica performs an **optimizer step** on its local optimizer states.
5. Perform an **all-gather** on the parameters to send the missing slices back to each replica. This is a new operation in ZeRO and is not used in vanilla DP.

ZeRO-2

Gradient Sharding



ZeRO-1

Each rank only has its shard of the optimizer states in memory, **at all times**

ZeRO-2

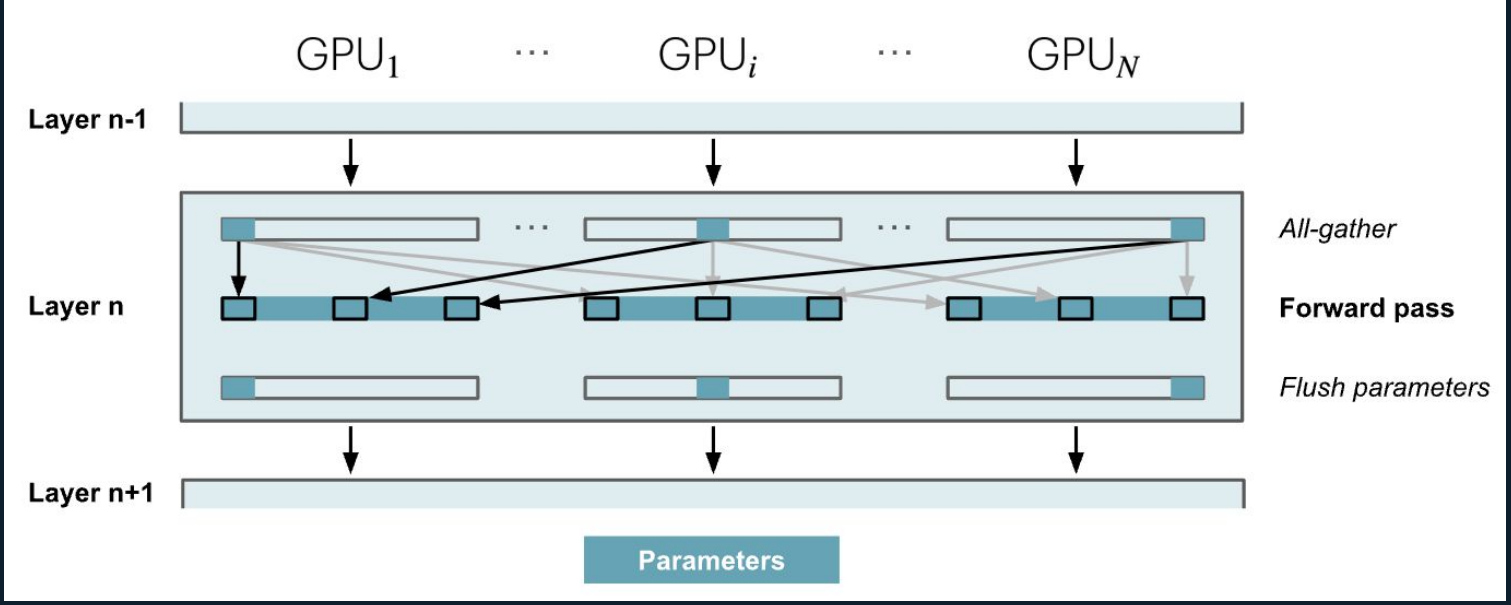
Each rank only has its shard of the gradients in memory, **most of the time**

ZeRO-3

Each rank only has its shard of the parameters in memory, **most of the time**

ZeRO-3

Parameter Sharding



ZeRO-1

Each rank only has its shard of the optimizer states in memory, **at all times**

ZeRO-2

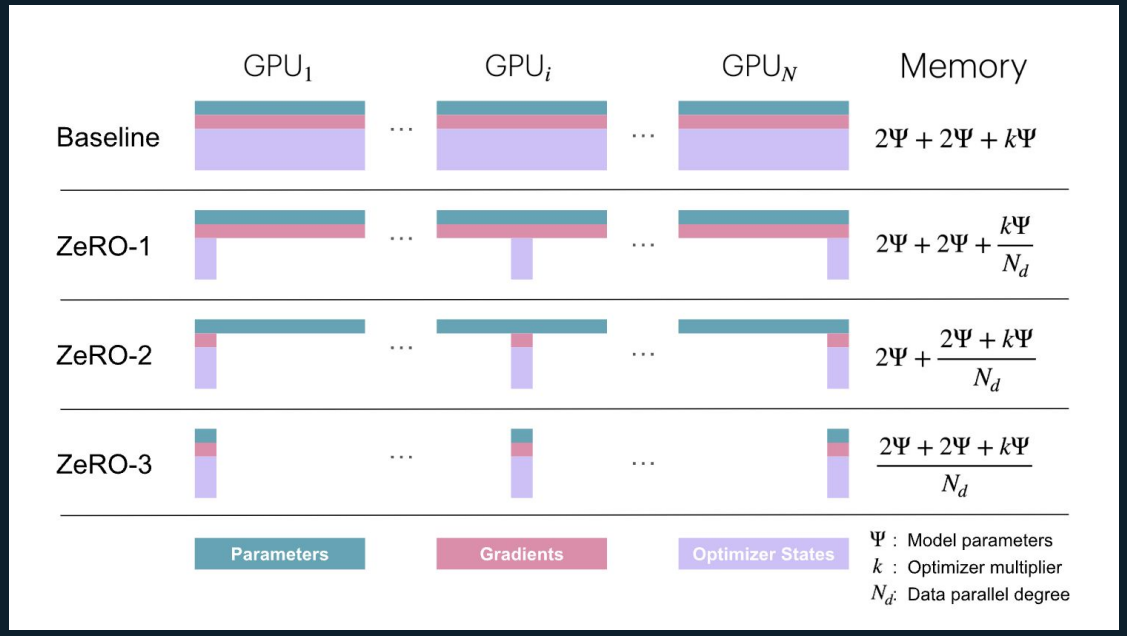
Each rank only has its shard of the gradients in memory, **most of the time**

ZeRO-3

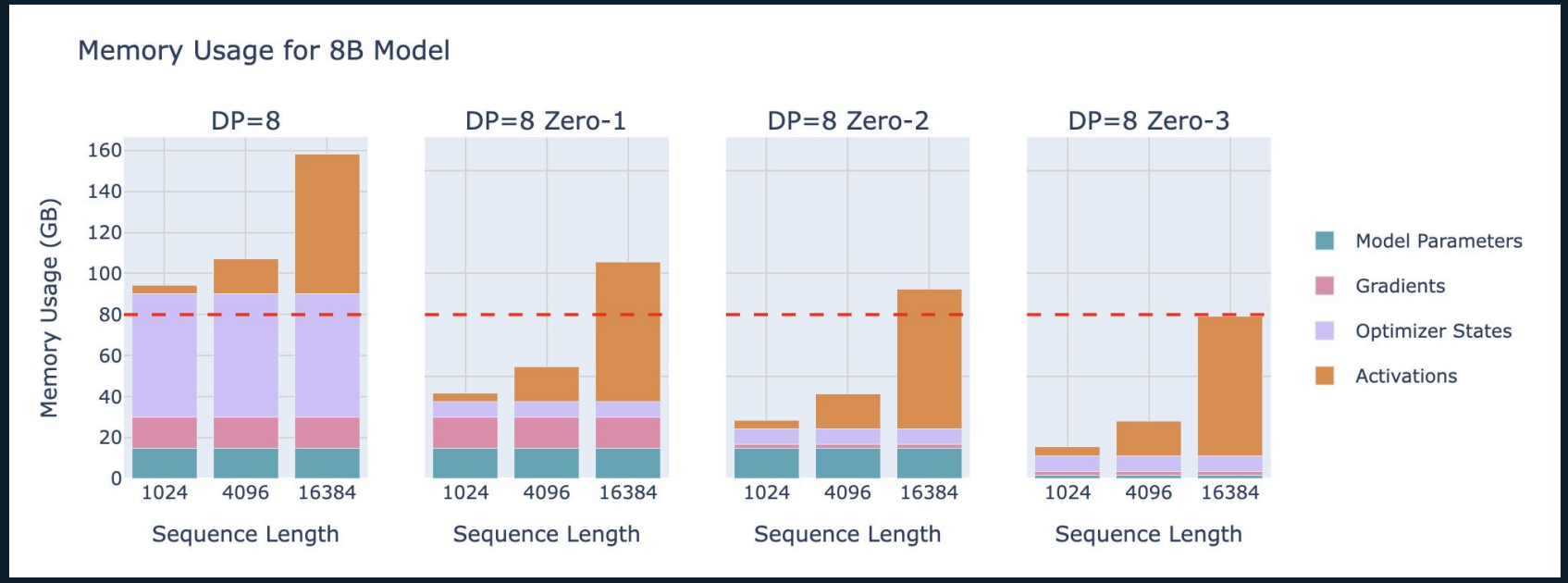
Each rank only has its shard of the parameters in memory, **most of the time**

Sharding Principles

- Shard optimizer states, gradients and parameters across DP ranks (⇒ avoid redundancy)
- All-Reduce → Reduce-Scatter + All-Gather
- Free memory asap (after each Reduce-Scatter)
- Trade-off: **lower memory footprint ↔ slightly more communication** (overlapped with computation)



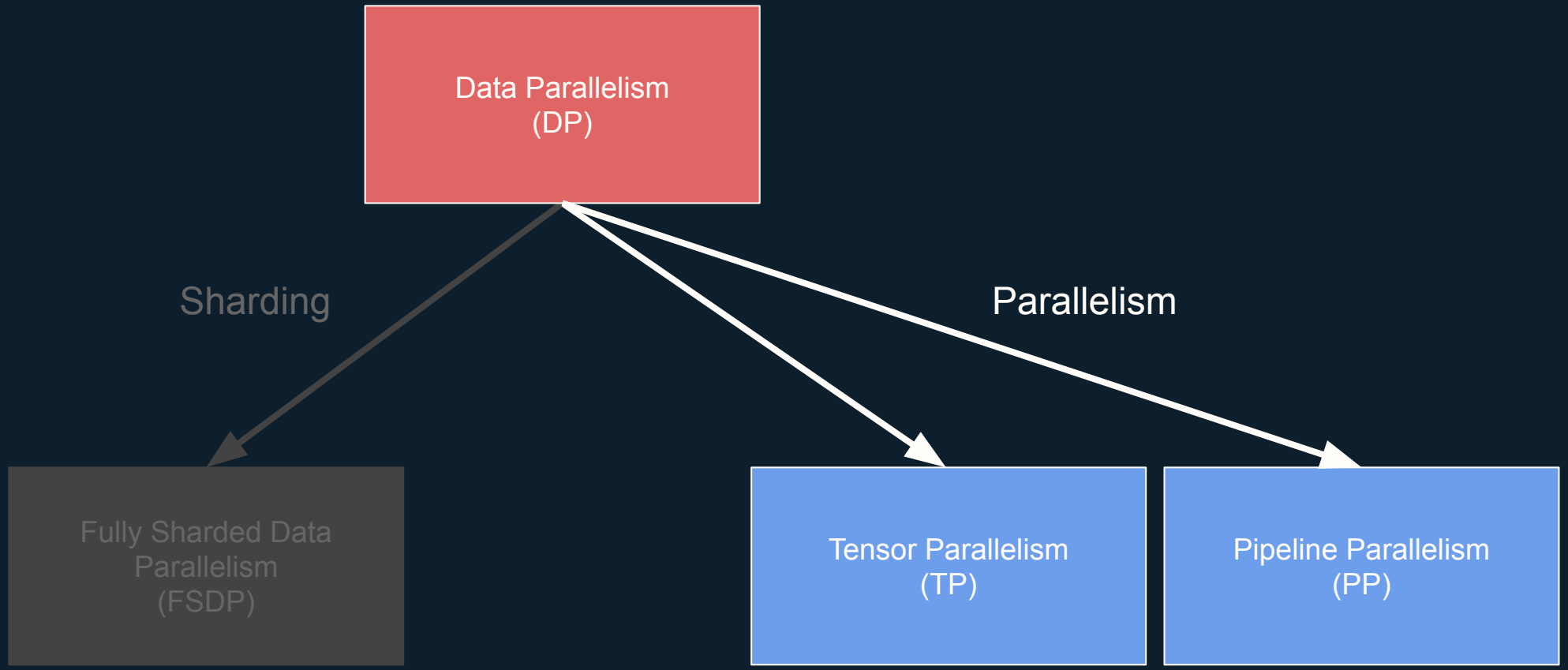
Sharding Summary



- Reduces memory footprint for optimizer states, gradients and parameters
- Allows to train larger models
- Allows to use larger micro batch sizes (⇒ less GPUs / more throughput)

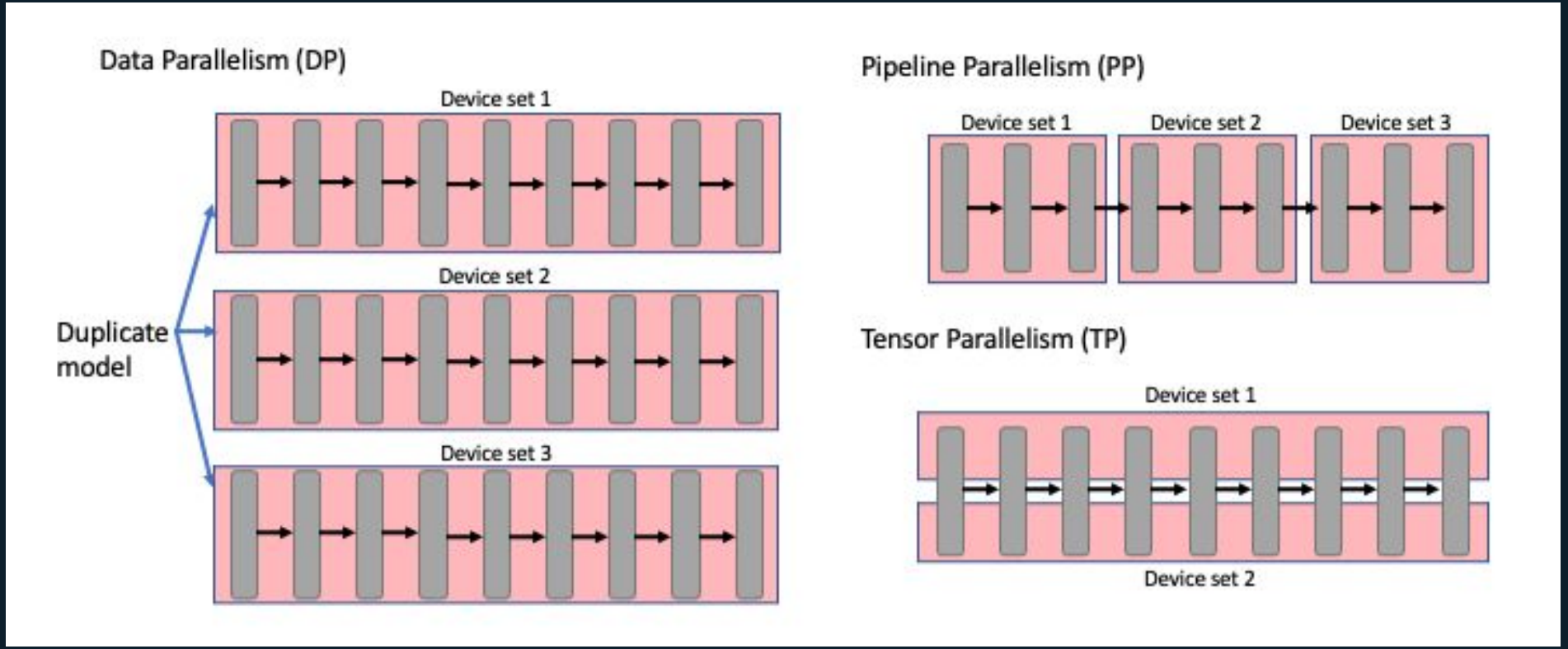
- Limitations: single model layer still needs to fit into memory

Sharding & Parallelism



Parallelism Overview

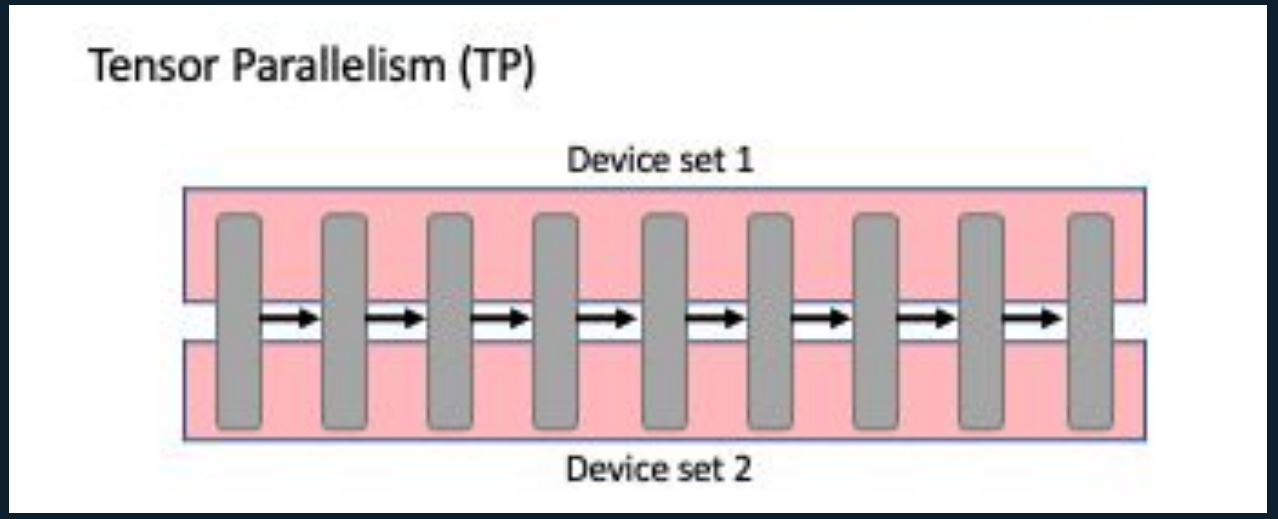
- Traditional 3D parallelism: DP + TP + PP



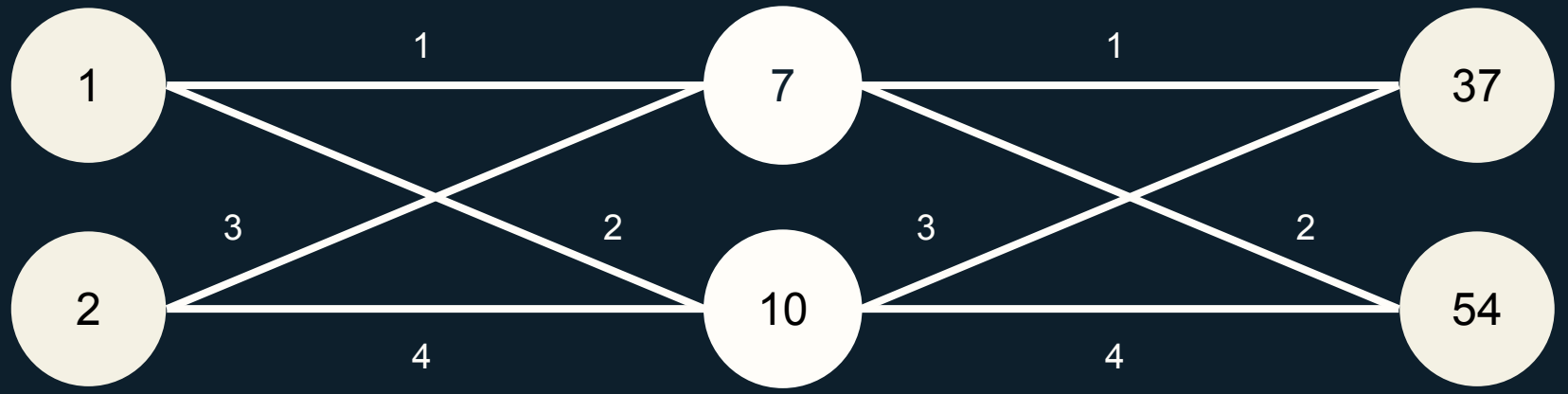
https://robotchinwag.com/posts/demystifying-tensor-parallelism/

Tensor Parallelism (TP)

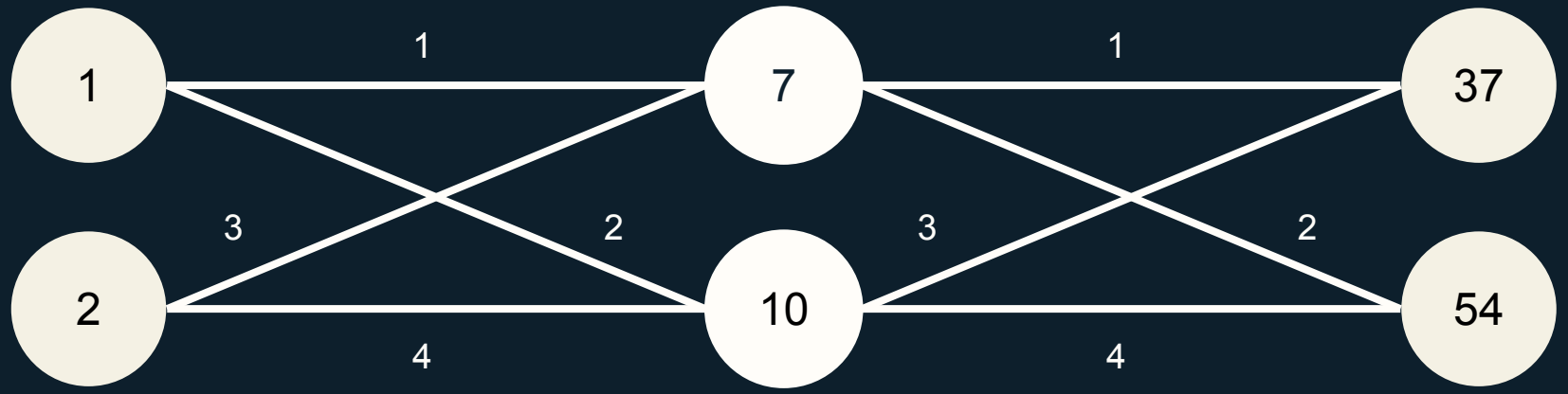
- Parallelizes optimizer states, gradients, parameters and (some) activations
- Parallelization happens within layers



Tensor Parallelism (TP)

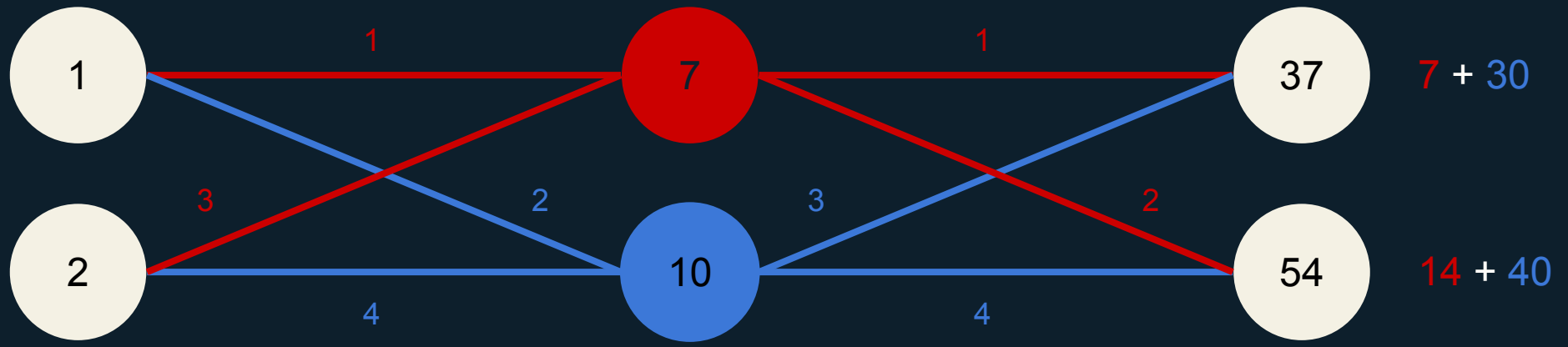


Tensor Parallelism (TP)



$$\begin{pmatrix} 1 & 2 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 7 & 10 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 37 & 54 \end{pmatrix}$$

Tensor Parallelism (TP)



$$\begin{pmatrix} 1 & 2 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 7 & 10 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 7+30 & 14+40 \end{pmatrix}$$

Tensor split (column-wise)

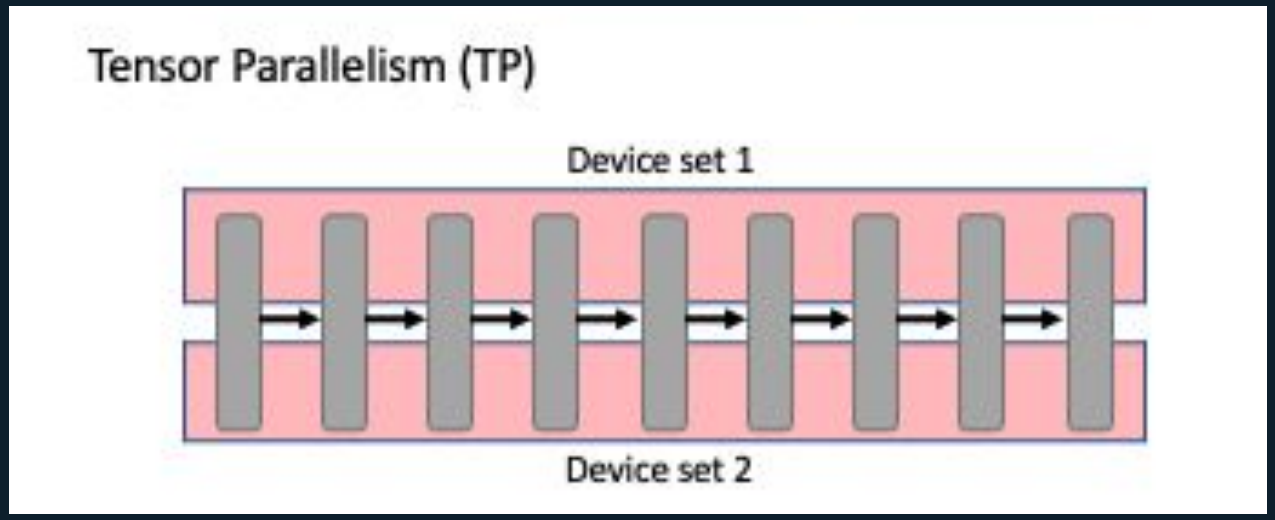
Tensor split (row-wise)

All-reduce

Tensor Parallelism (TP)

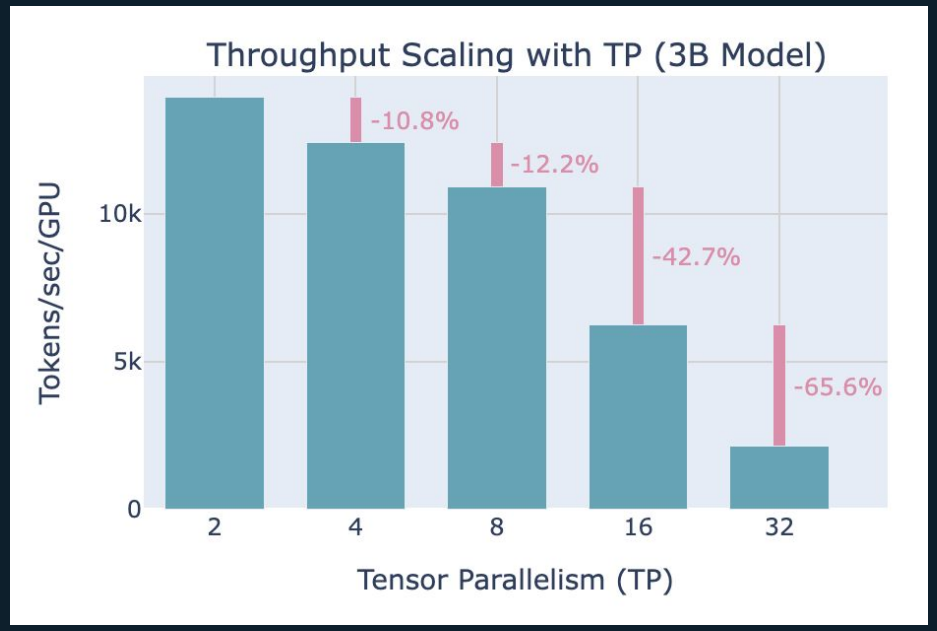
- Tensor split
  - FFN: along hidden dimension
  - Attention: along attention heads

- Memory Footprint
  - optimizer states, gradients and parameters are parallelized
  - intermediate activations are parallelized
  - block-output activations are not parallelized (need to be gathered for LayerNorm)



Tensor Parallelism (TP)

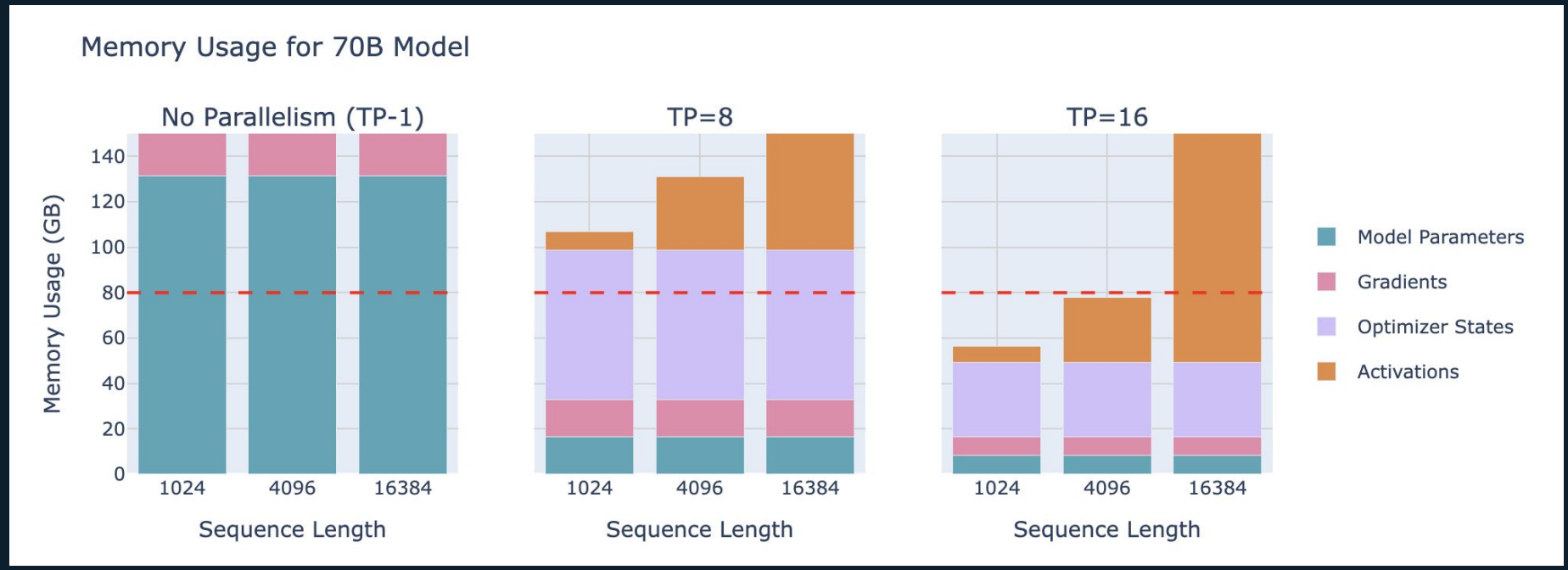
- All-reduce produces irreducible communication overhead



In practice, use

- intra-node TP (high-speed bandwidth)
- modest amount of TP = 2, 4, 8

Tensor Parallelism (TP)

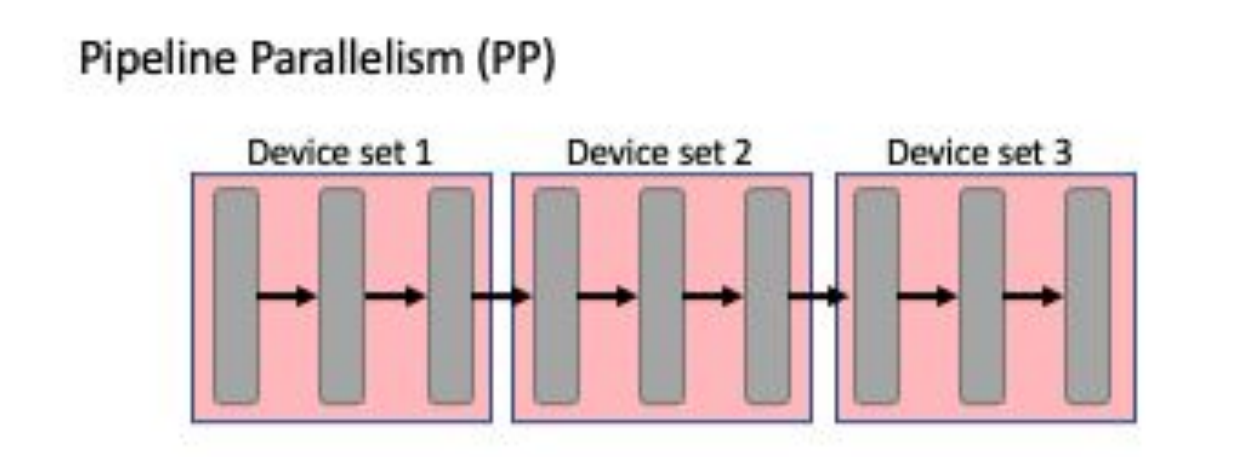


- Reduces memory footprint for optimizer states, gradients, parameters and (some) activations
- Allows to train larger models
- Allows to use larger micro batch sizes (⇒ less GPUs / more throughput)

- Limitations: block-output activations not parallelized, irreducible communication overhead, intra-node

Pipeline Parallelism (PP)

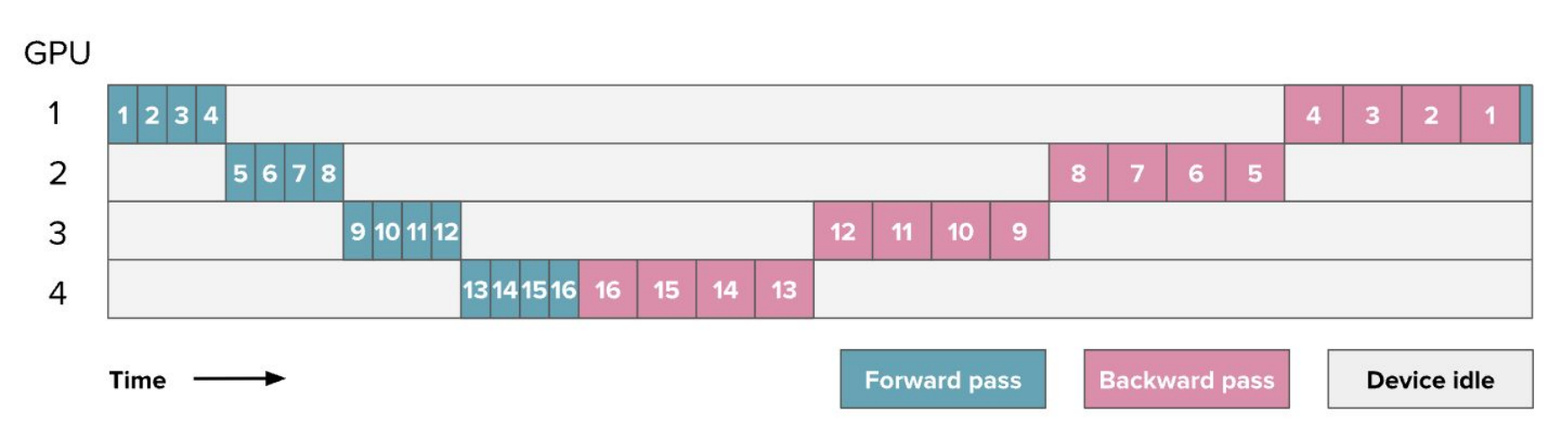
- Parallelizes optimizer states, gradients, parameters
- Parallelization happens across layers



Only few, moderate-sized activations are communicated between GPUs

Pipeline Parallelism (PP)

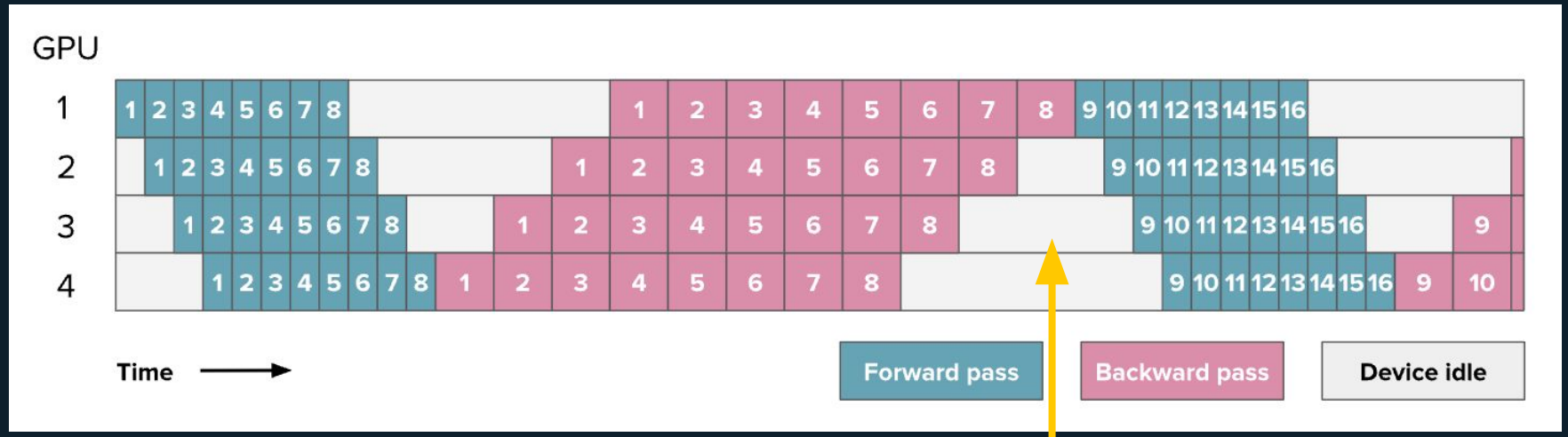
- Naively, this is embarrassingly sequential..



1-16: Layers

Pipeline Parallelism (PP)

- ..but micro-batch forward/backward passes can be overlapped and processed in parallel

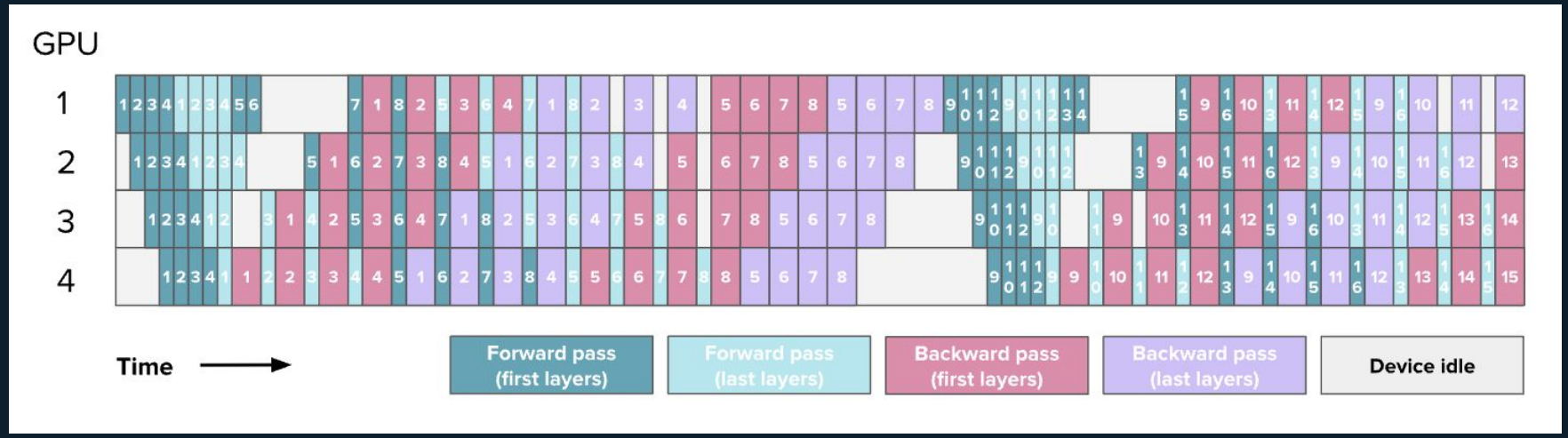


1-16:
Micro-batches

Bubble

Pipeline Parallelism (PP)

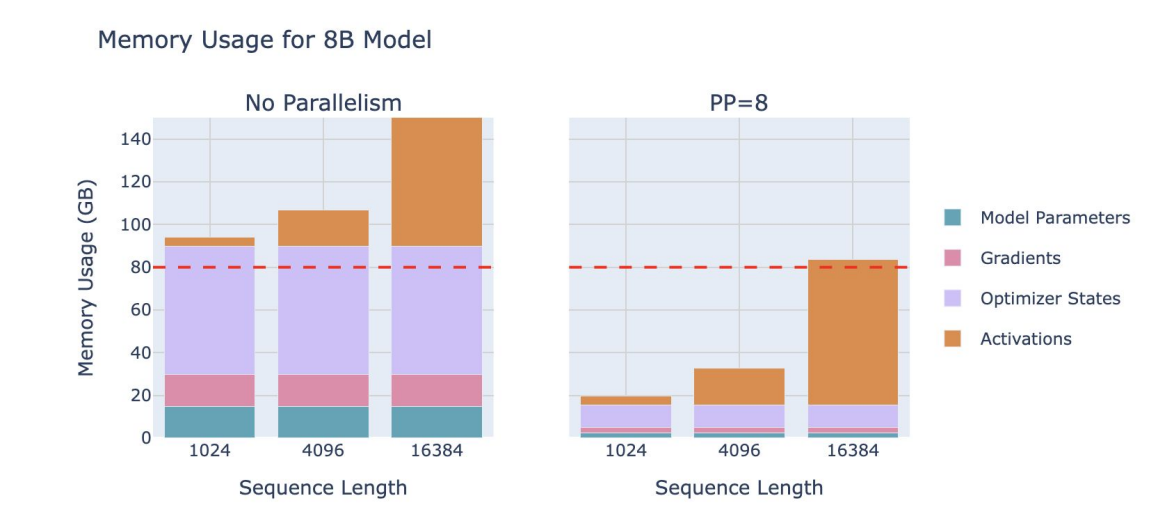
- The bubble can be reduced further using advanced techniques



1-16:
Micro-batches

- Recent work (Deep-Seek-V3/R1) reduces the bubble to virtually zero

Pipeline Parallelism (PP)



- Reduces memory footprint for optimizer states, gradients, parameters
- Allows to train larger models
- Allows to use larger micro batch sizes (⇒ less GPUs / more throughput)

- Typically used inter-node

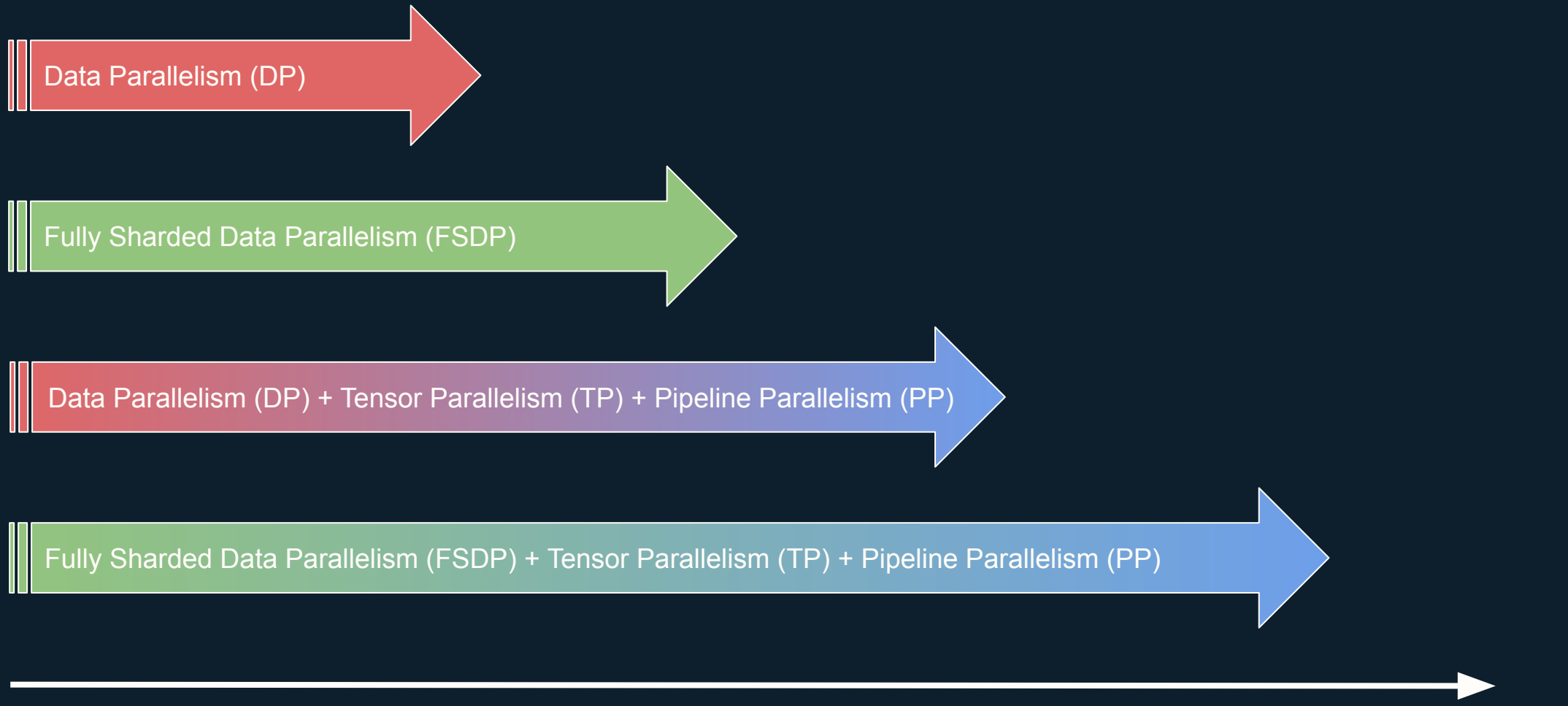
Multi-GPU Training: Summary

AI SWEDEN

	optimizer states	gradients	parameters	activations	layer split
DP	-	-	-	parallelized	-
FSDP	sharded	sharded	sharded	parallelized (DP)	within
TP	parallelized	parallelized	parallelized	partly parallelized	within
PP	parallelized	parallelized	parallelized	-	across

(FS)DP, TP & PP can be combined

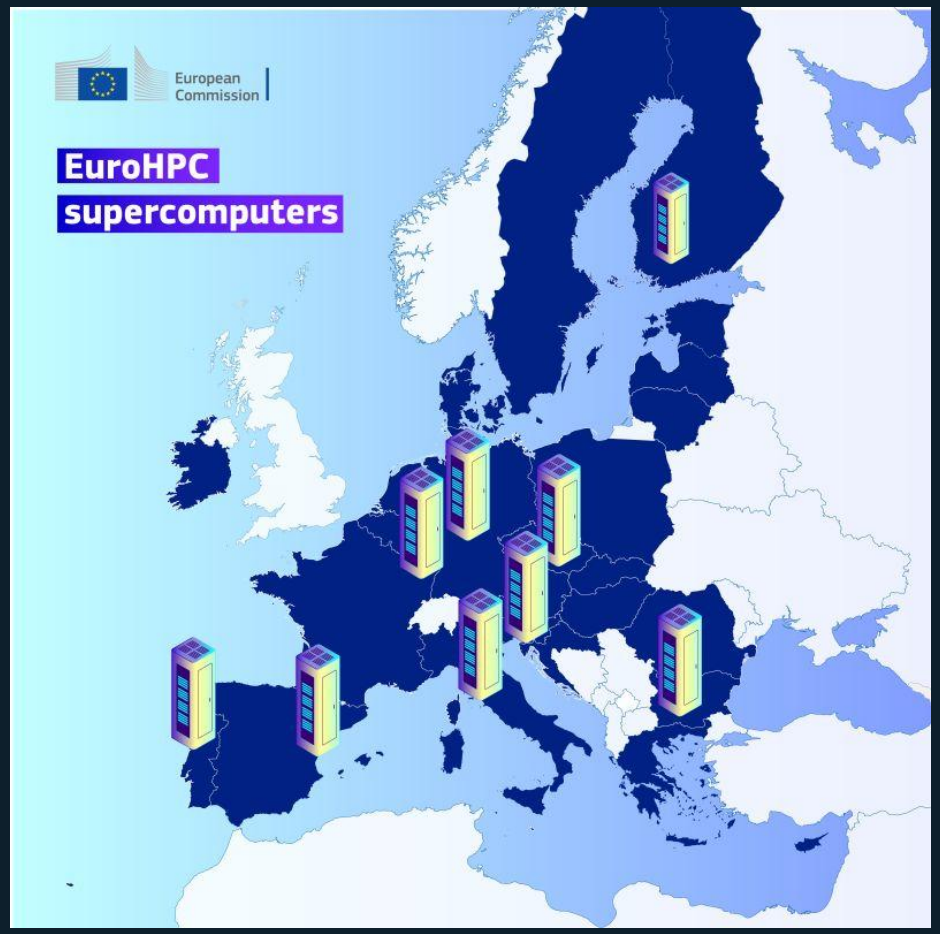
Multi-GPU Training: Summary



Part V

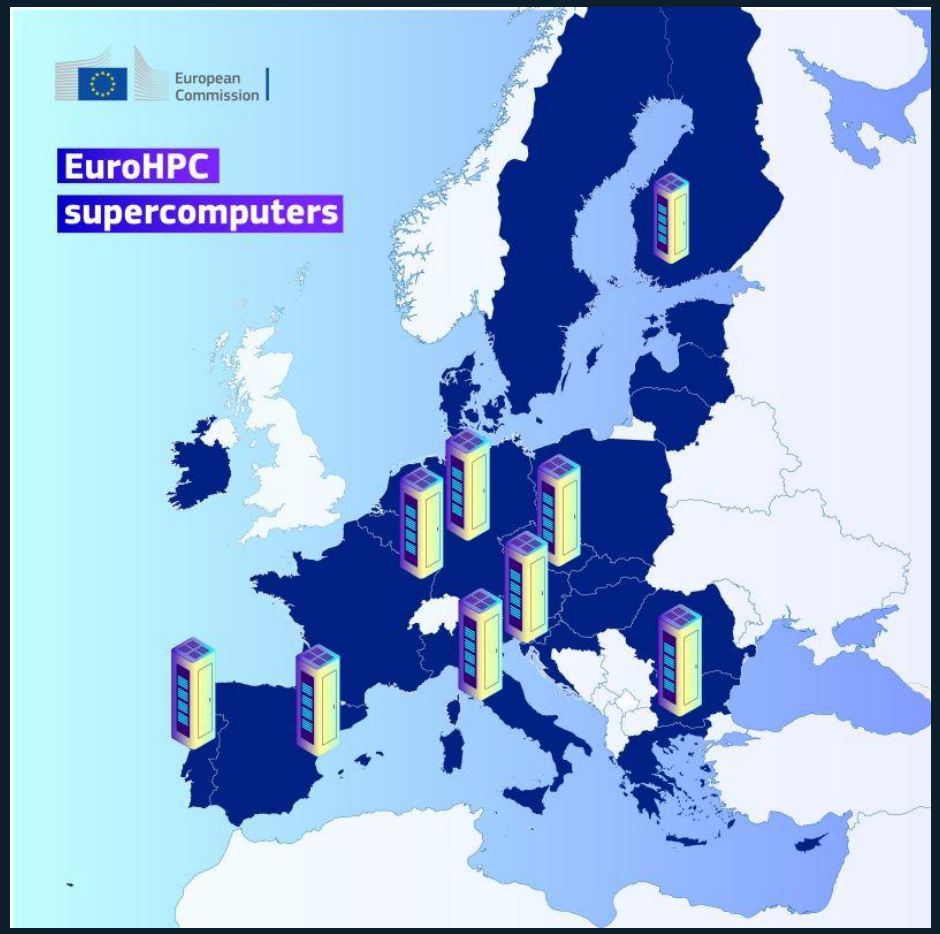
HPC in Europe

Compute



- Lumi (Finland)
- Leonardo (Italy)
- MareNostrum5 (Spain)
- MeluXina (Luxembourg)
- Vega (Slovenia)
- Discoverer (Bulgaria)
- Karolina (Czech Republic)
- Deucalion (Portugal)
- Jupiter (Germany)

Compute



Major Differences

- GPU type (A100, H100, H200, AMD)
- GPUs per node (4 or 8)

Minor Differences

- Partitions, queues & reservation time
- Software stack
- Disk usage (quota, backups, ..)
- Peculiarities (e.g. internet access)
- ..

Common Feature: Slurm job scheduler

Compute

AI SWEDEN

DEMO

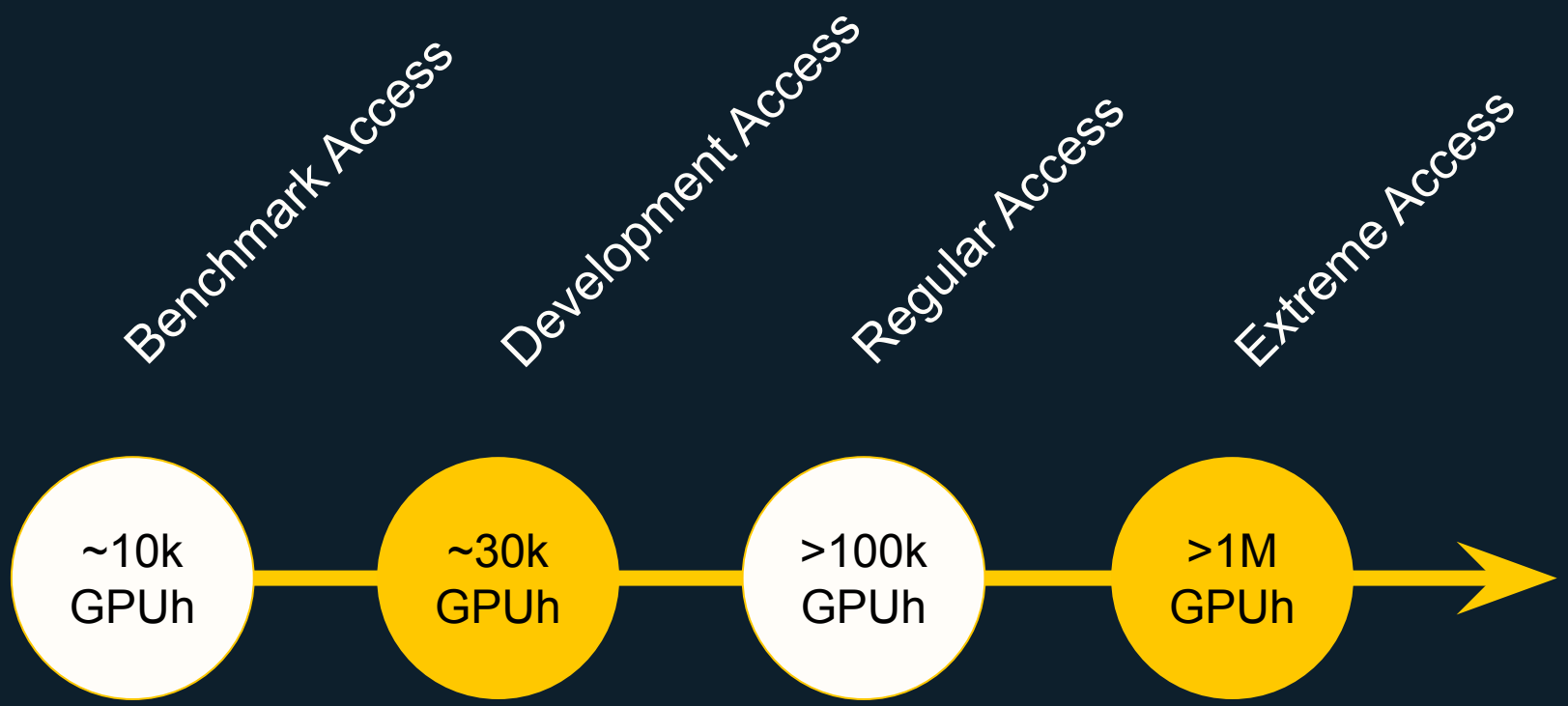
Compute

AI SWEDEN

A few things to consider

- jobs
  - can be looped (to handle limited reservation time)
  - waiting time is unpredictable + there will be downtime / failure
- checkpointing
  - choose a reasonable frequency that minimizes waste of compute
  - make sure there are backups
- versioning
  - save code version & training config (e.g. write automatically to W&B)
- responsibility
  - track and optimize GPU utilization
  - small test runs before scaling up
  - double-check & evaluate everything frequently

Compute Calls



Part VI

Training Frameworks

Framework

A training framework should fulfill the following requirements:

- Correctness
  it needs to do the right thing
- Performance
  it needs to be fast, efficient & parallelizable
- Usability
  it should be simple and easy to use
- SOTA
  it should support modern model architectures & features
- Adaptability
  it should be easy to implement own model architectures & features
- Maturity
  documentation, proven track record, community

Framework (Examples)

	Megatron	LLM Foundry	Modalities
Developer	Nvidia	Mosaic ML	Open Source (Fraunhofer & AI Sweden)
Basis	custom	transformers	pytorch
Parallelization	custom	custom	pytorch
Correctness			
Performance			
Usability			
SOTA			
Adaptability			
Maturity			

subjective
(should be taken
with a grain of salt)

Framework (Examples)

	Megatron	LLM Foundry	Modalities
Developer	Nvidia	Mosaic ML	Open Source (Fraunhofer & AI Sweden)
Basis	custom	transformers	pytorch
Parallelization	custom	custom	pytorch
Correctness	↑	↑	↑
Performance	↑	↗	↗
Usability	→	↗	↗
SOTA	↗	↗	↗
Adaptability	→	→	↑
Maturity	↗	↑	→

subjective (should be taken with a grain of salt)

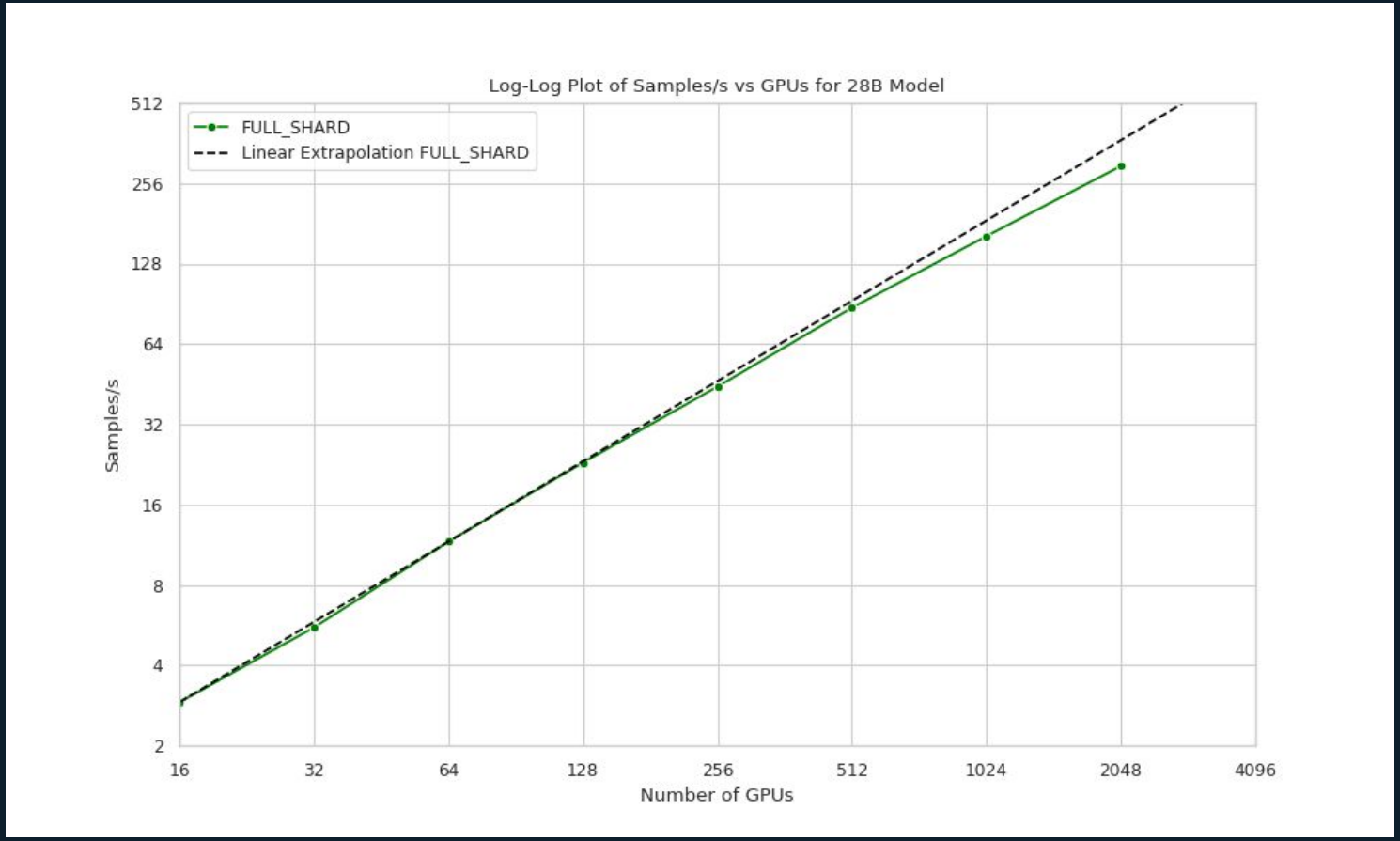
Framework (Modalities)

AI SWEDEN

DEMO

Framework (Modalities)

based on PyTorch FSDP



Thank you!

felix.stollenwerk@ai.se