Systems for Machine Learning (CMSC828G)

Parallel Training

Abhinav Bhatele, Daniel Nichols

UNIVERSITY OF MARYLAND

Announcements

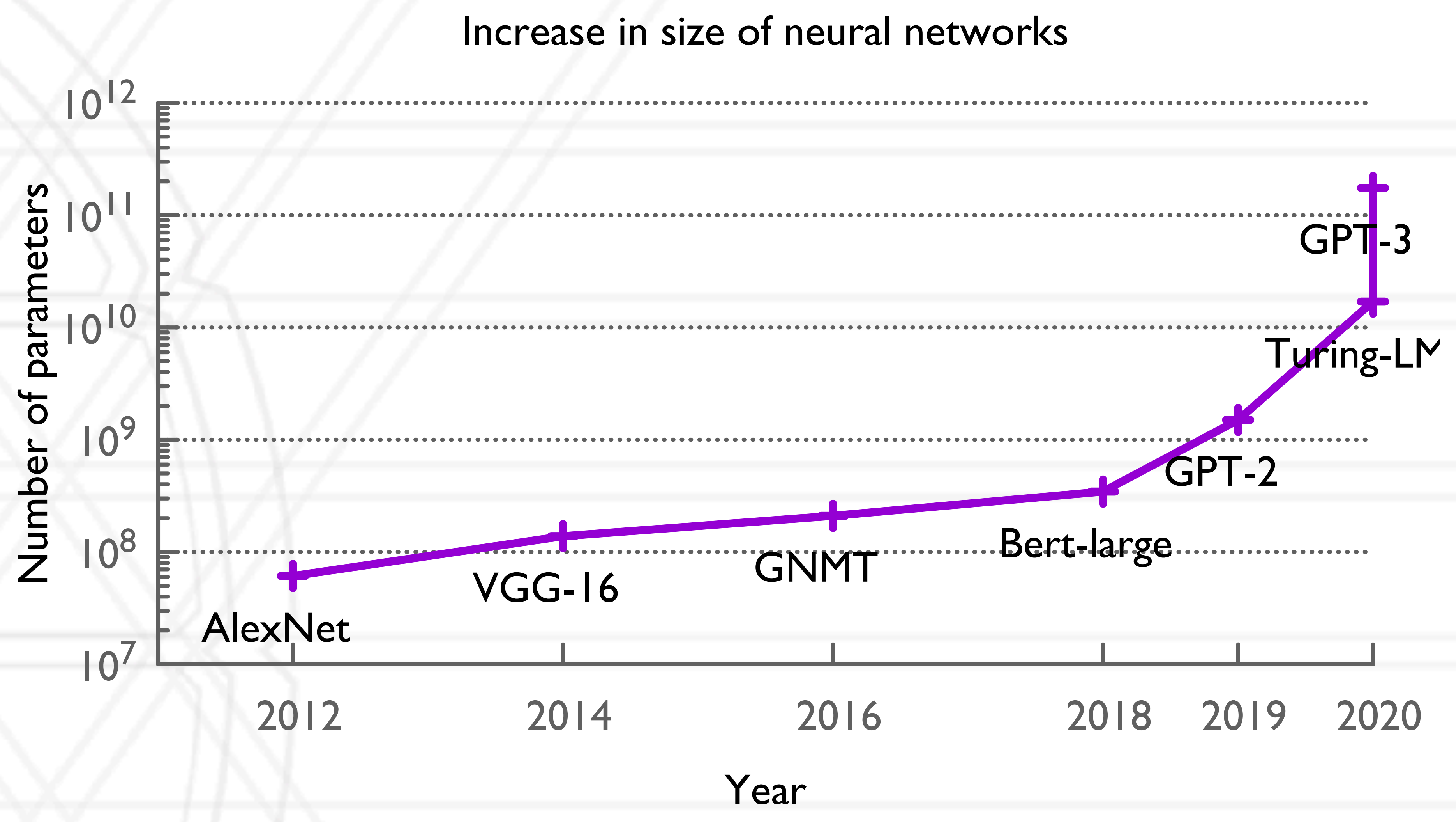
- Reminder: project proposals due on March 7 (extended)

Parallel/distributed training

- Many opportunities for exploiting parallelism
- Iterative process of training (epochs)
- Many iterations per epoch (mini-batches)
- Many layers in DNNs

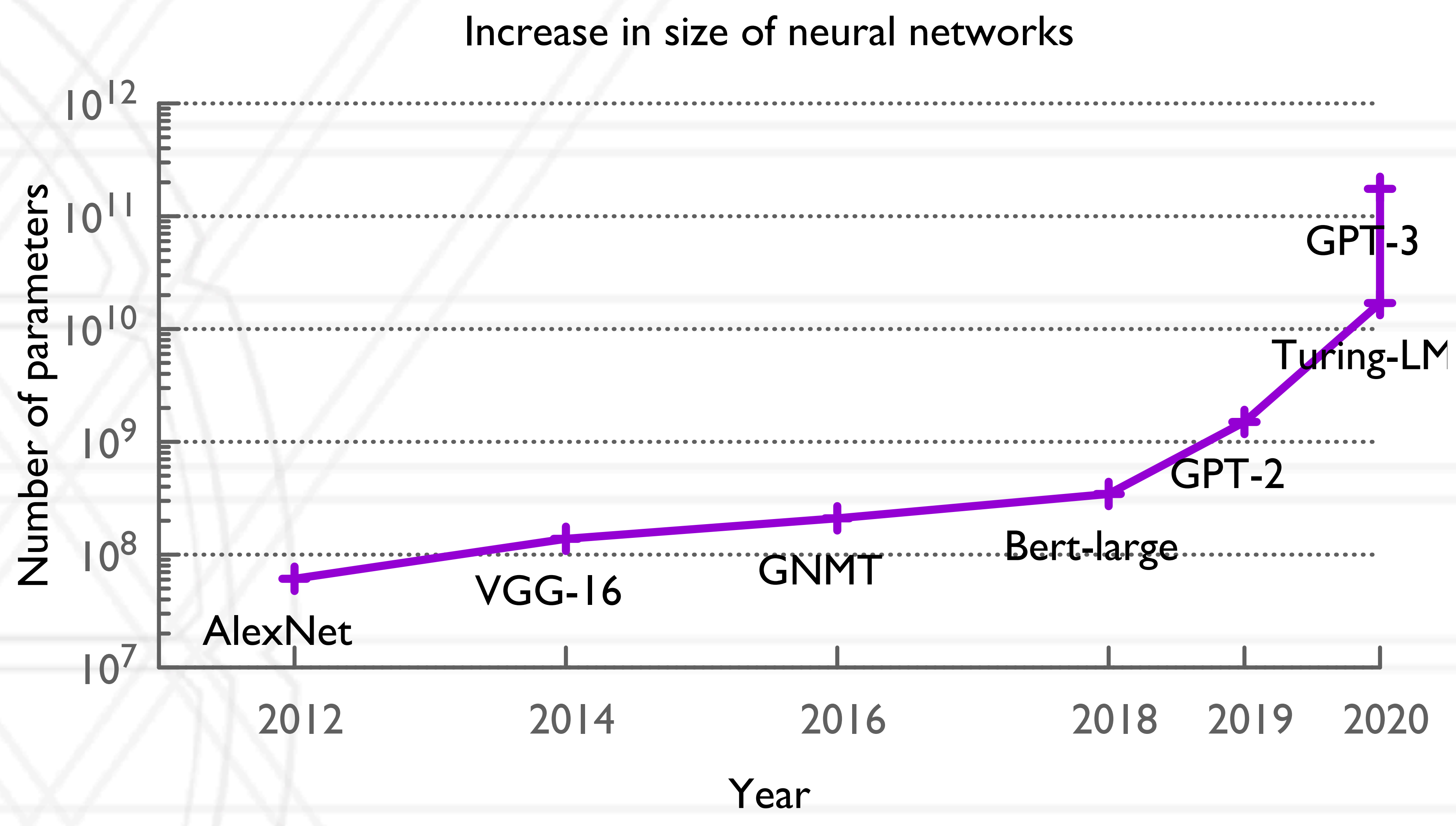
Parallel/distributed training

- Many opportunities for exploiting parallelism
- Iterative process of training (epochs)
- Many iterations per epoch (mini-batches)
- Many layers in DNNs



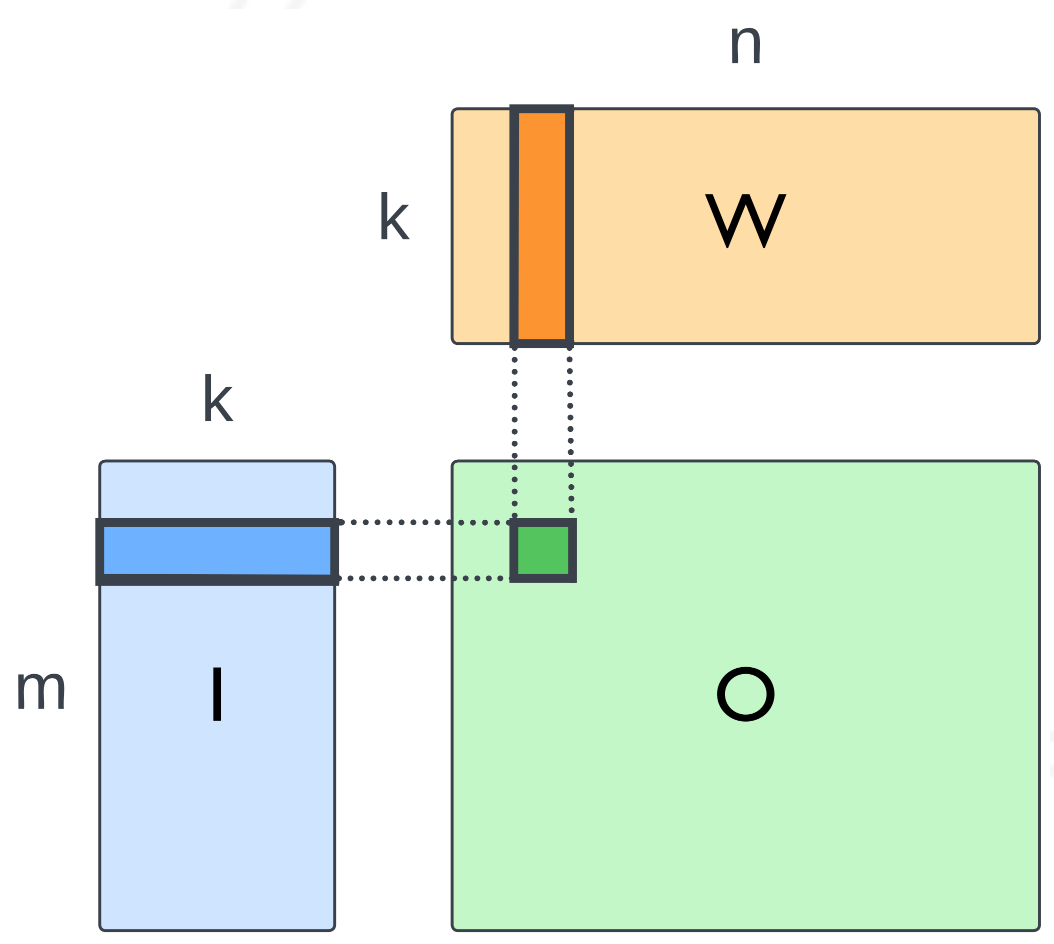
Parallel/distributed training

- Many opportunities for exploiting parallelism
- Iterative process of training (epochs)
- Many iterations per epoch (mini-batches)
- Many layers in DNNs



Framework	Type of Parallelism	Largest Accelerator Count	Largest Trained Network (No. of Parameters)
FlexFlow	Hybrid	64 GPUs	24M*
PipeDream	Inter-Layer	16 GPUs	138M
DDP	Data	256 GPUs	345M
GPipe	Inter-Layer	8 GPUs	557M
MeshTensorFlow	Intra-Layer	512-core TPUv2	4.9B
Megatron	Intra-Layer	512 GPUs	8.3B
TorchGPipe	Inter-Layer	8 GPUs	15.8B
KARMA	Data	2048 GPUs	17B
LBANN	Data	3072 CPUs	78.6B
ZeRO	Data	400 GPUs	100B

Sequential training




```
while (remaining_batches) {
    Read a single batch

    Forward pass: perform matrix multiplies to compute
        output activations

    Compute loss on this batch

    Backward pass: matrix multiplies to compute gradients of
        the loss w.r.t. parameters via backpropagation

    Optimizer step: use gradients to update the weights or
        parameters such that loss is gradually reduced
}
```

Data parallelism

- Divide training data (input batch) among workers (GPUs)
- Each worker has a full copy of the entire NN and processes different mini-batches
- All reduce operations to synchronize gradients
- Example: PyTorch's DDP, ZeRO

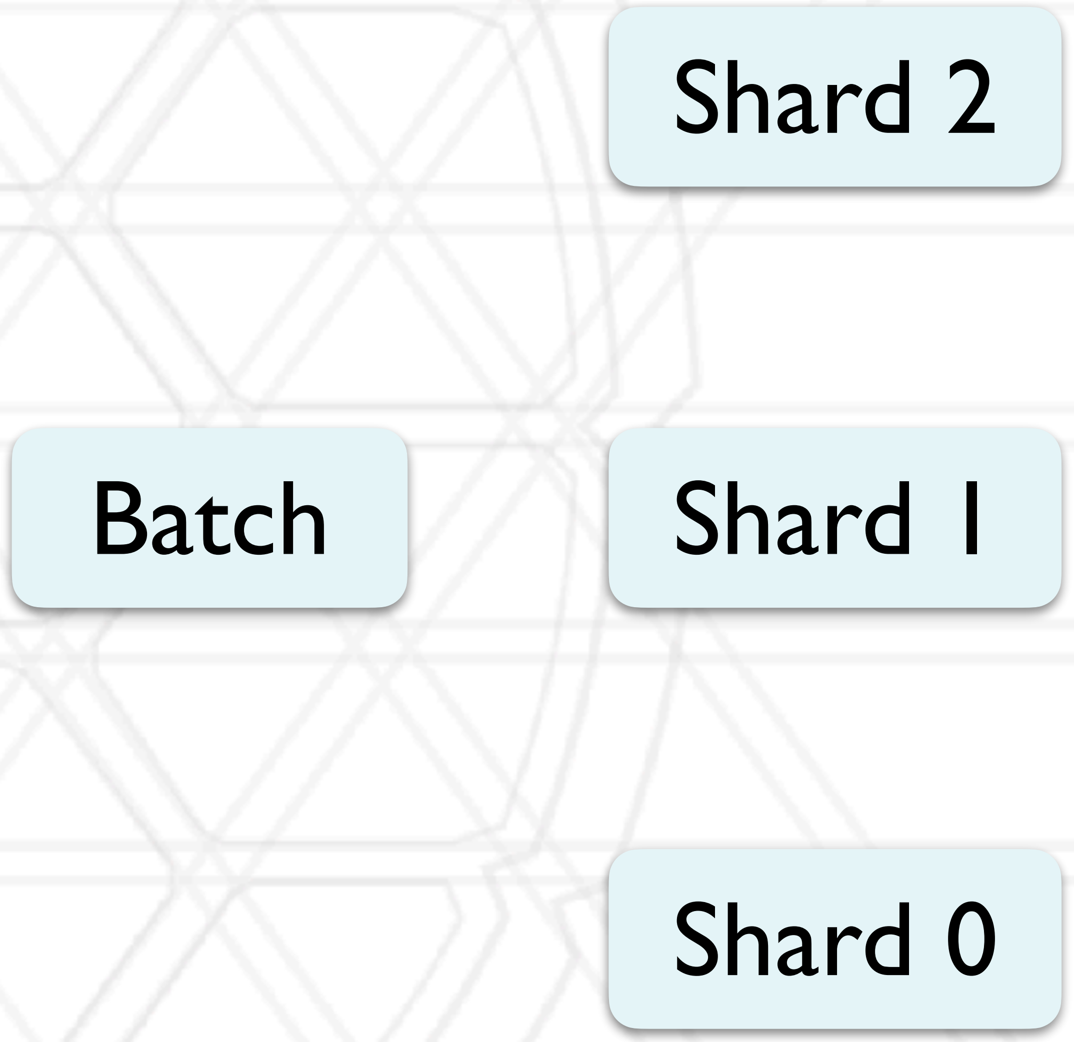
Data parallelism

- Divide training data (input batch) among workers (GPUs)
- Each worker has a full copy of the entire NN and processes different mini-batches
- All reduce operations to synchronize gradients
- Example: PyTorch's DDP, ZeRO

Batch

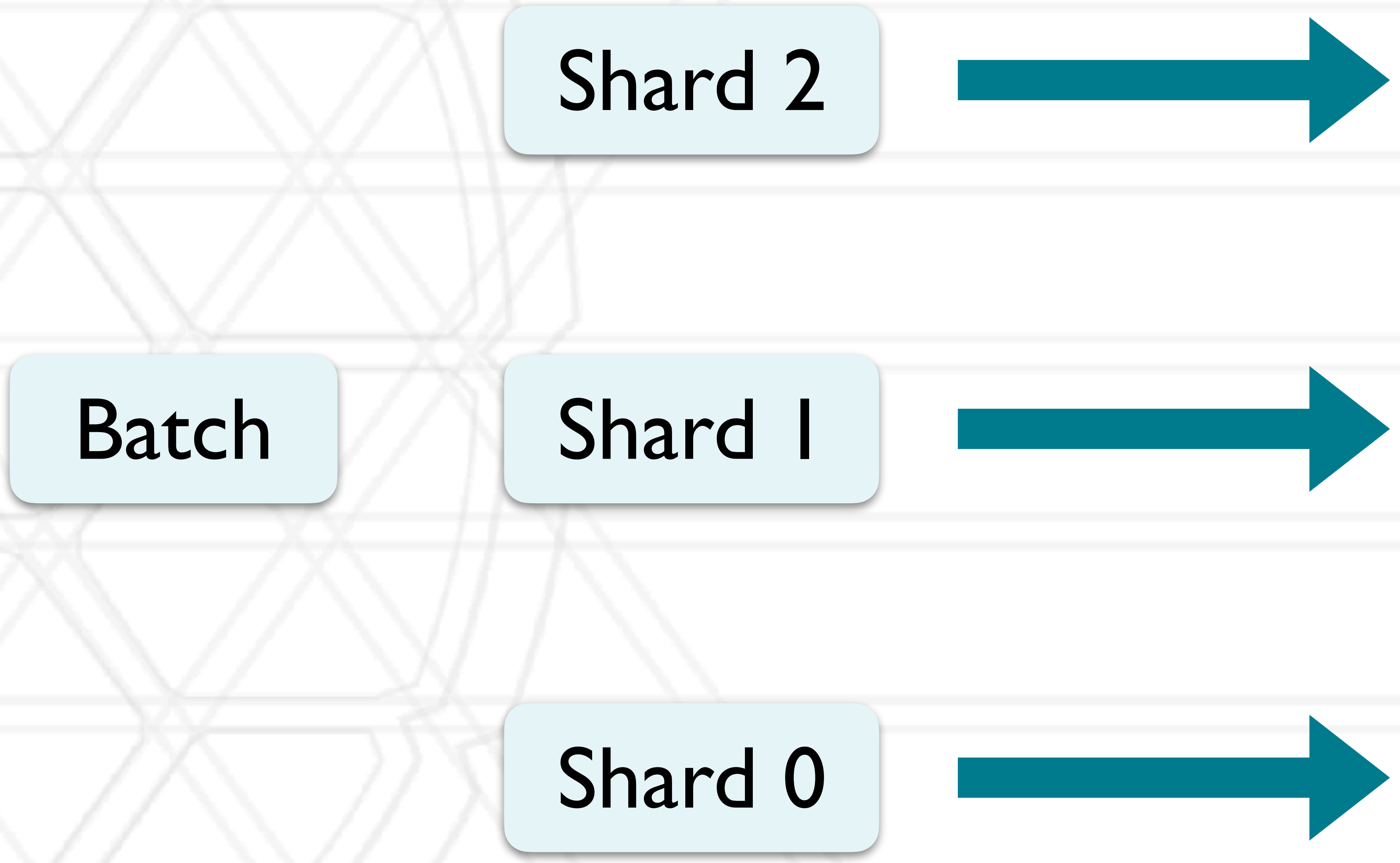
Data parallelism

- Divide training data (input batch) among workers (GPUs)
- Each worker has a full copy of the entire NN and processes different mini-batches
- All reduce operations to synchronize gradients
- Example: PyTorch's DDP, ZeRO



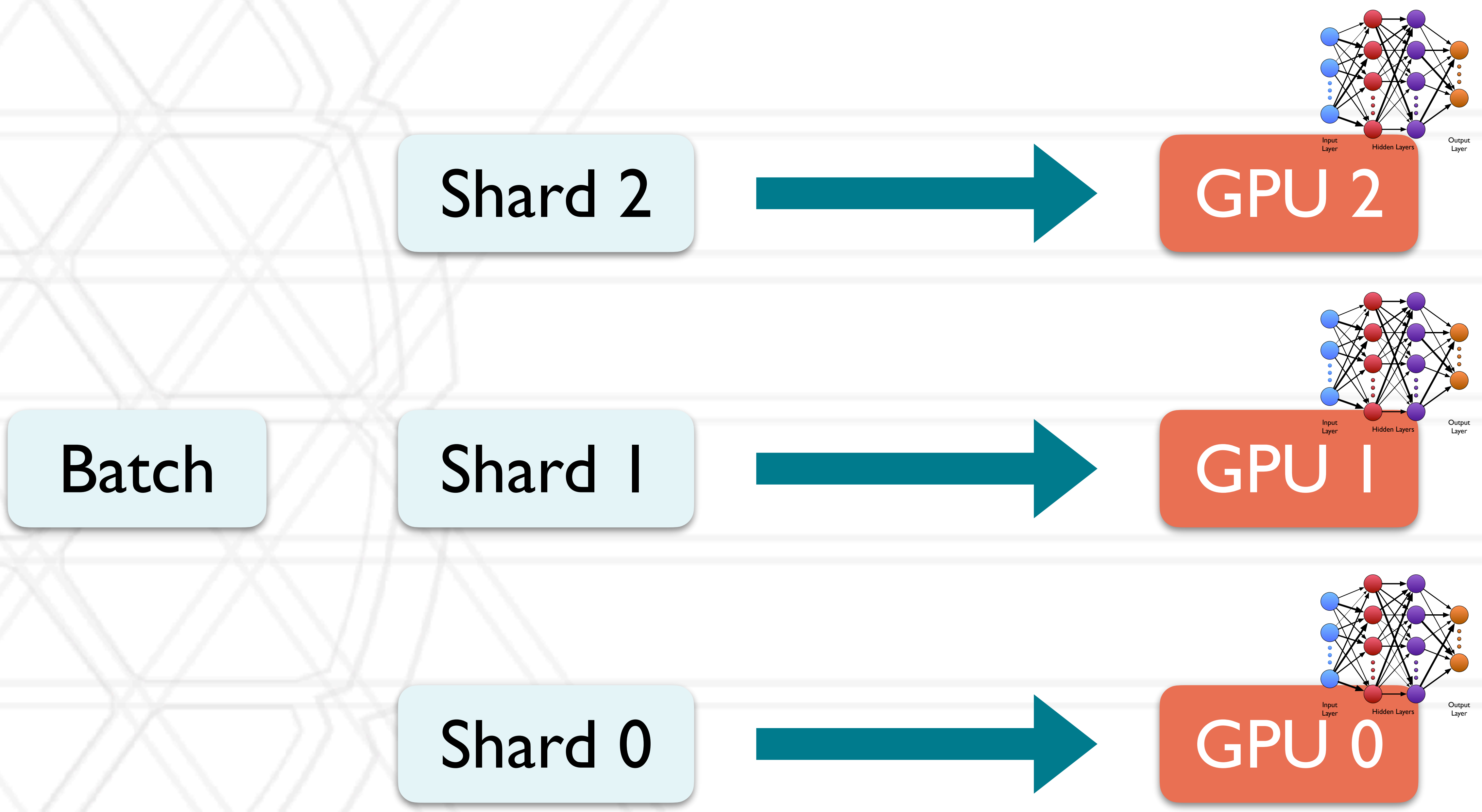
Data parallelism

- Divide training data (input batch) among workers (GPUs)
- Each worker has a full copy of the entire NN and processes different mini-batches
- All reduce operations to synchronize gradients
- Example: PyTorch's DDP, ZeRO



Data parallelism

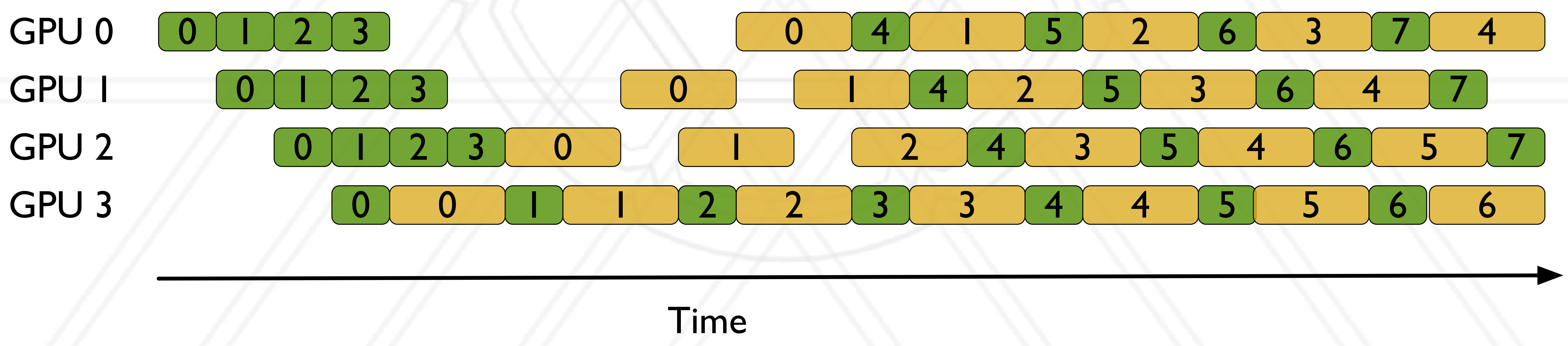
- Divide training data (input batch) among workers (GPUs)
- Each worker has a full copy of the entire NN and processes different mini-batches
- All reduce operations to synchronize gradients
- Example: PyTorch's DDP, ZeRO



Data Parallelism

Inter-layer parallelism

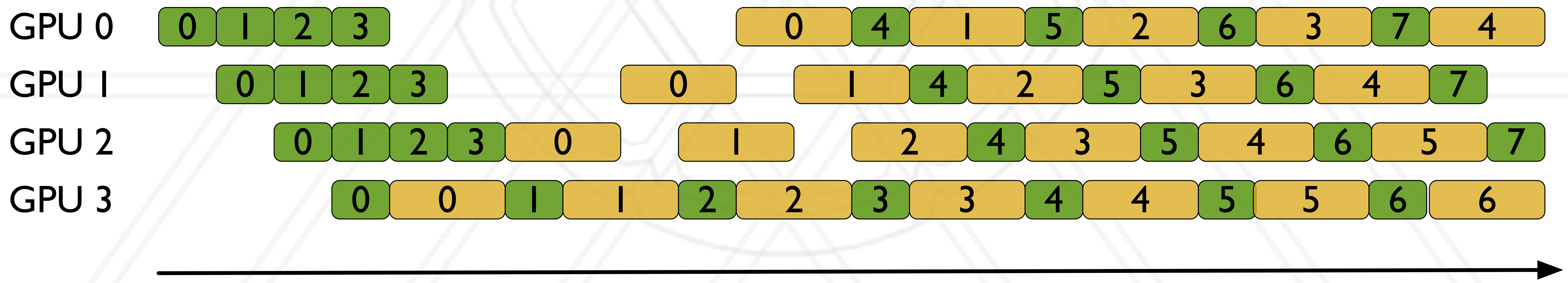
- Assign entire layers to different processes/GPUs
  - Ideally map contiguous subsets of layers
- Point-to-point communication (activations and gradients) between processes/GPUs managing different layers
- Use a pipeline of mini-batches to enable concurrent execution



Inter-layer parallelism

- Assign entire layers to different processes/GPUs
  - Ideally map contiguous subsets of layers
- Point-to-point communication (activations and gradients) between processes/GPUs managing different layers
- Use a pipeline of mini-batches to enable concurrent execution

Pipeline parallelism



Intra-layer parallelism

- Enables training neural networks that would not fit on a single GPU
- Distribute the work within each layer to multiple processes/GPUs
  - Essentially parallelize matrix operations such as matmuls across multiple GPUs
- Example: Megatron-LM

Intra-layer parallelism

Tensor parallelism

- Enables training neural networks that would not fit on a single GPU
- Distribute the work within each layer to multiple processes/GPUs
  - Essentially parallelize matrix operations such as matmuls across multiple GPUs
- Example: Megatron-LM

Hybrid parallelism

- Using two or more approaches together in the same parallel framework
- 3D parallelism: use all three
- Popular serial frameworks: pytorch, tensorflow
- Popular parallel frameworks: DDP, MeshTensorFlow, Megatron-LM, ZeRO, AxoNN

DDP: Distributed Data Parallelism

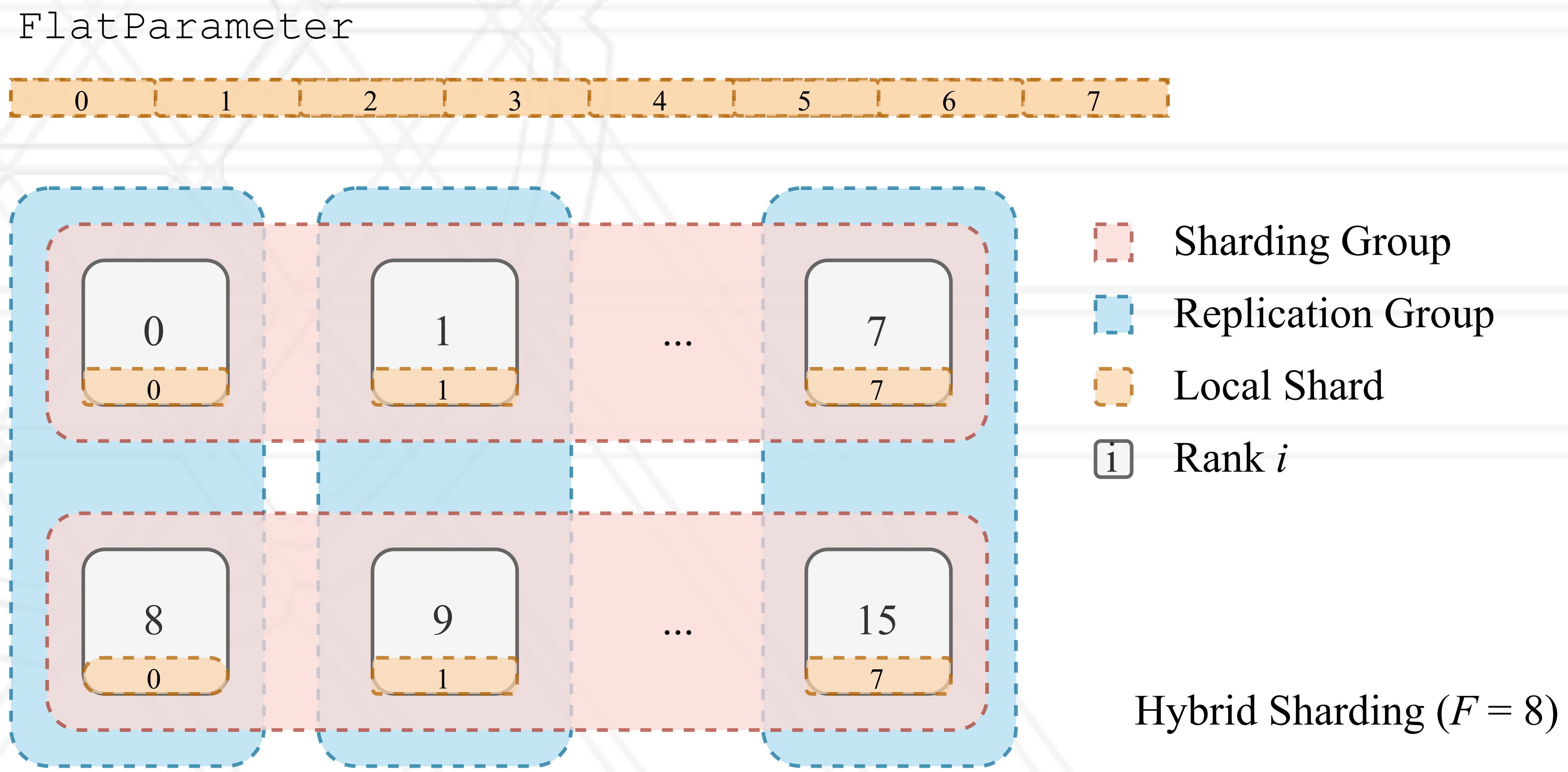
- Naive solution: wait for the entire backward pass to complete before issuing an all-reduce
- Improvement: issues all-reduces as gradient tensors become ready
- Even better: combine multiple all-reduces into a single operation — "buckets"

FSDP: Fully Sharded Data Parallelism

- "Sharding": Distribute model parameters within a layer or "FSDP unit" across all GPUs
- All-gather the parameters "before" computation starts
- Why does this work?

Different sharding strategies

- Fully replicated: data parallelism
- Fully sharded
- Hybrid of fully replicated (data) + fully sharded

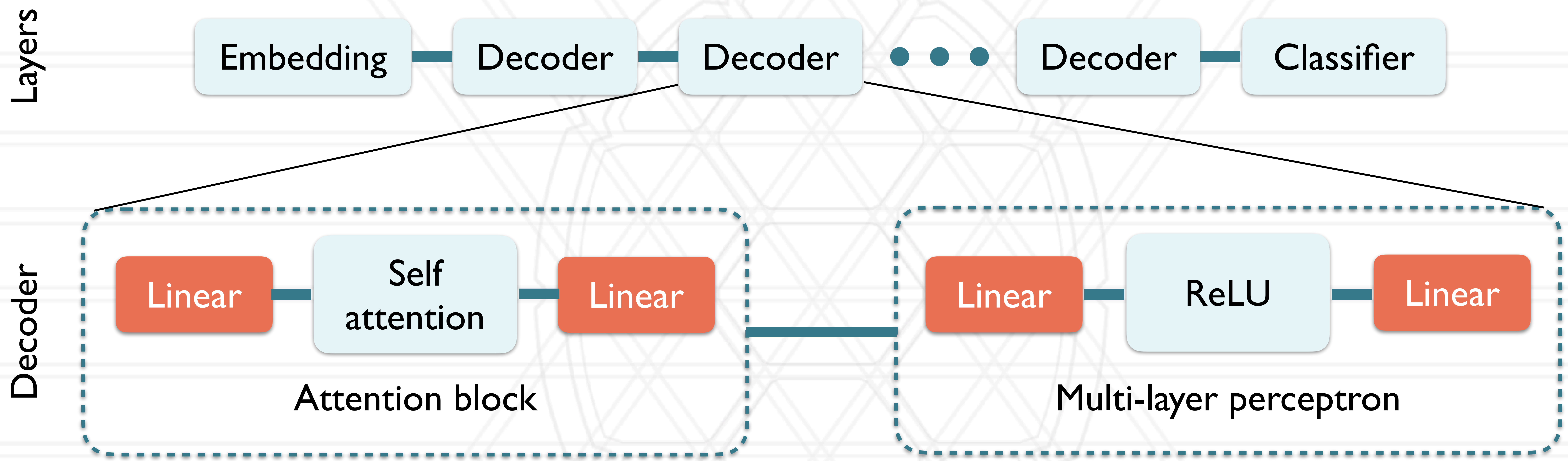


Intra-layer (tensor) parallelism

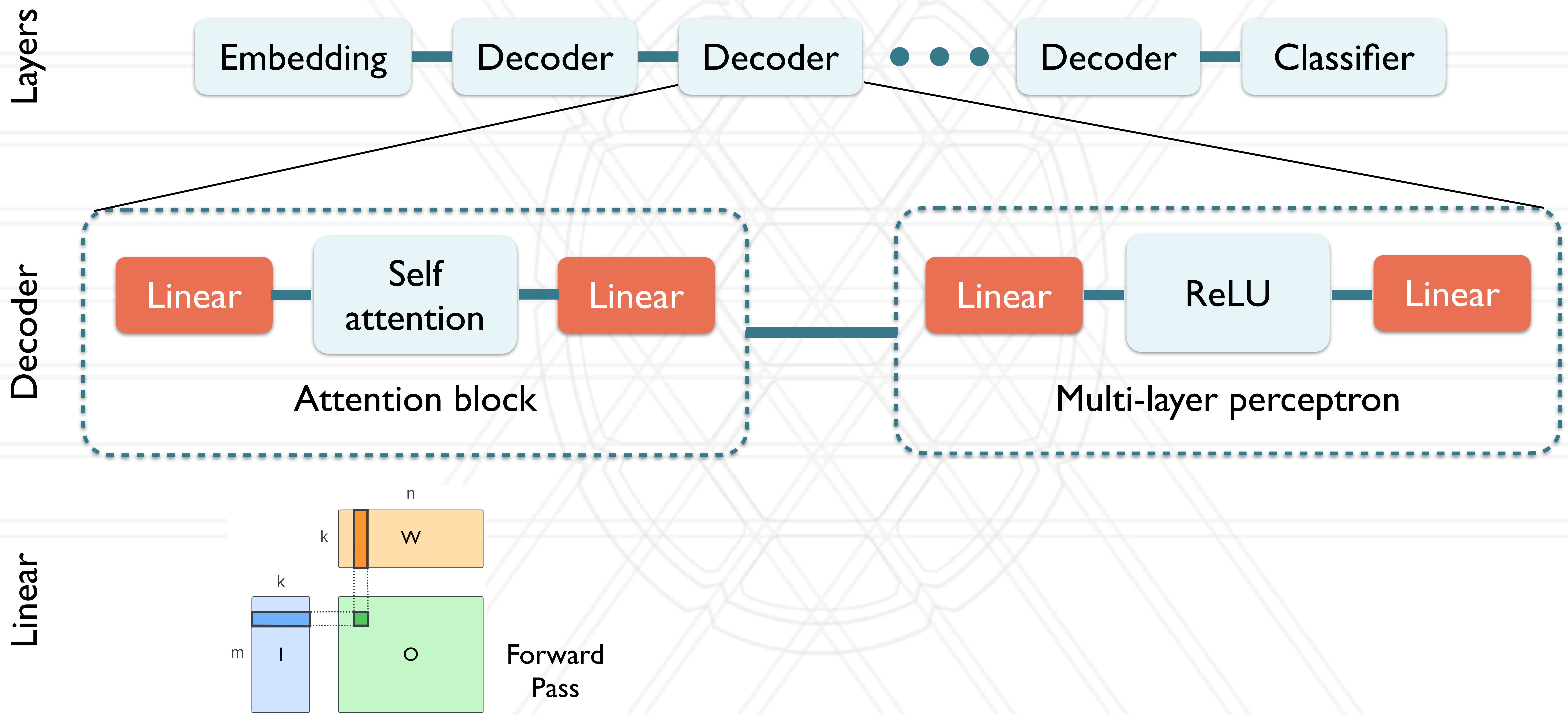
- Enables training neural networks that would not fit on a single GPU
- Distribute the work within each layer to multiple processes/GPUs
  - Essentially parallelize matrix operations such as matrix multiplies across multiple GPUs

Why is LLM training well-suited for HPC?

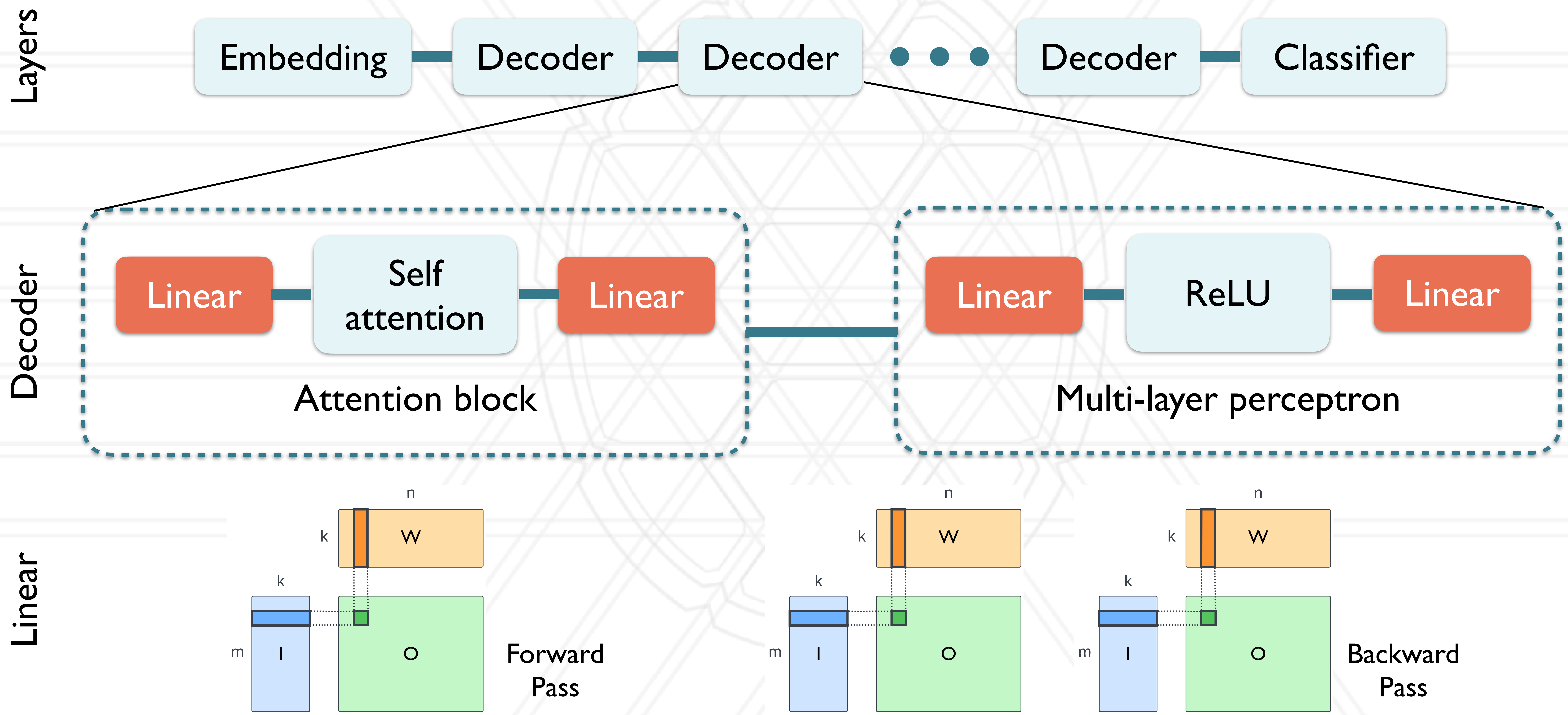
Why is LLM training well-suited for HPC?



Why is LLM training well-suited for HPC?



Why is LLM training well-suited for HPC?



Parallelizing a matrix multiply kernel

- Distribute matrices A and B
- Each process computes a portion of the result matrix, C
- Some communication is required depending on how you distribute the matrices and where you want the final output to be
- Choices:
  - How to divide the matrices: 1D or 2D
  - How to arrange the GPUs in a virtual grid: 1D, 2D or 3D

Frameworks

- 1D arrangement of GPUs: Megatron-LM
- 3D arrangement of GPUs: AxoNN

UNIVERSITY OF
MARYLAND