AI For Science at Scale : Part 2

Sajal Dash

dashs@ornl.gov

Overview (Hour 1)

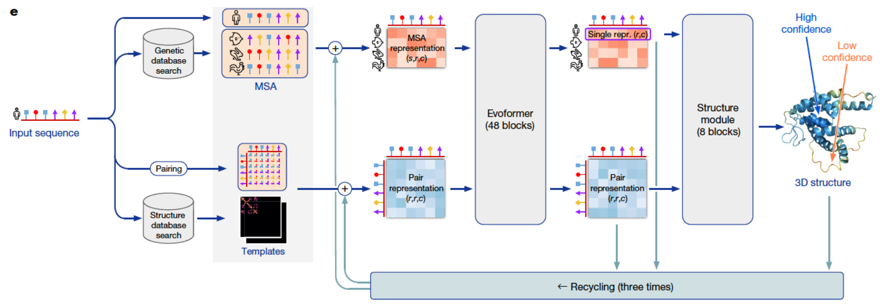
- AI For Science
- Deep Learning and Large Language Model training
- Data parallel training
- Model parallel training
  - Fully Sharded Data Parallel (FSDP)
  - Tensor Parallel (TP)
  - Pipeline Parallel (PP)
  - Hybrid Parallelism
- Case Study: Forge

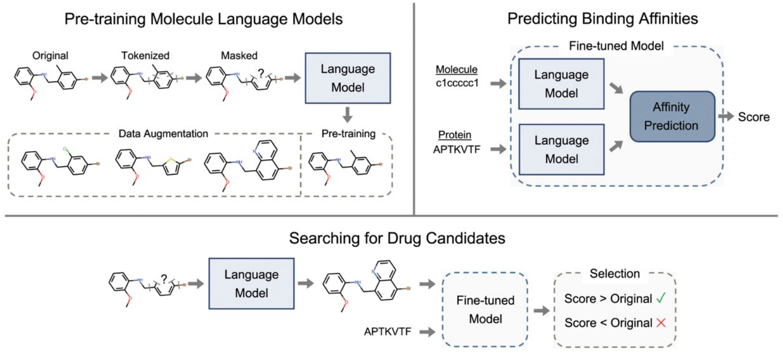
Overview (Hour 2)

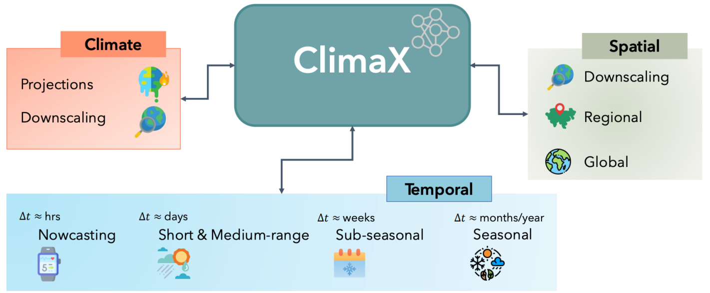
- Slurm
- Data Parallel Training
  - DDP Example
- Sharded Data Parallel Training
  - DeepSpeed ZeRO example
  - FSDP example
- Tensor, Pipeline, and Hybrid Parallelism
  - Megatron-DeepSpeed Example

AI For Science

- Applications of AI methods accelerated and enabled new discoveries
- AlphaFold "solved" decades-old protein folding problem
- Language models were used to predict new drug targets
- ClimaX, a Transformer model predicts climate and weather



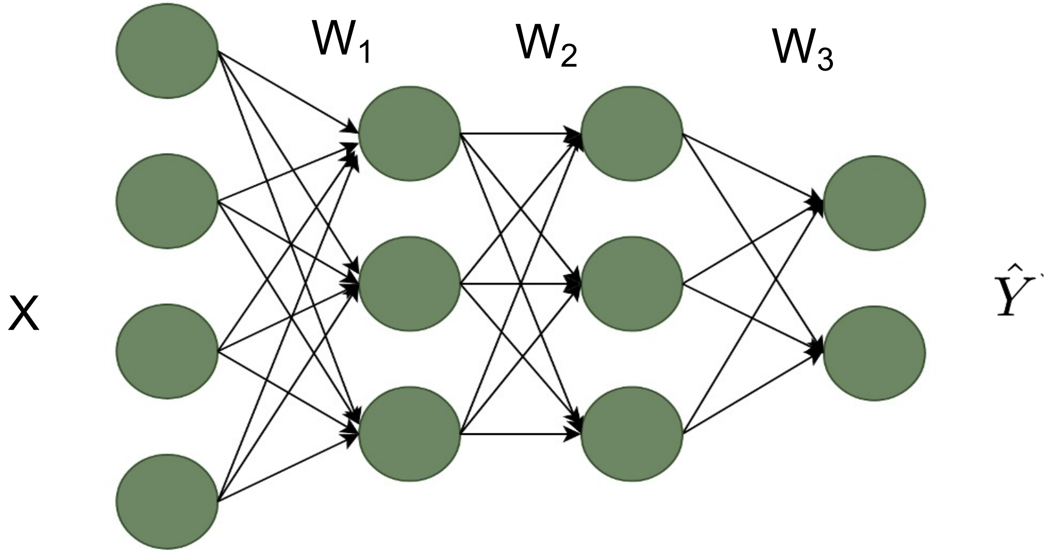




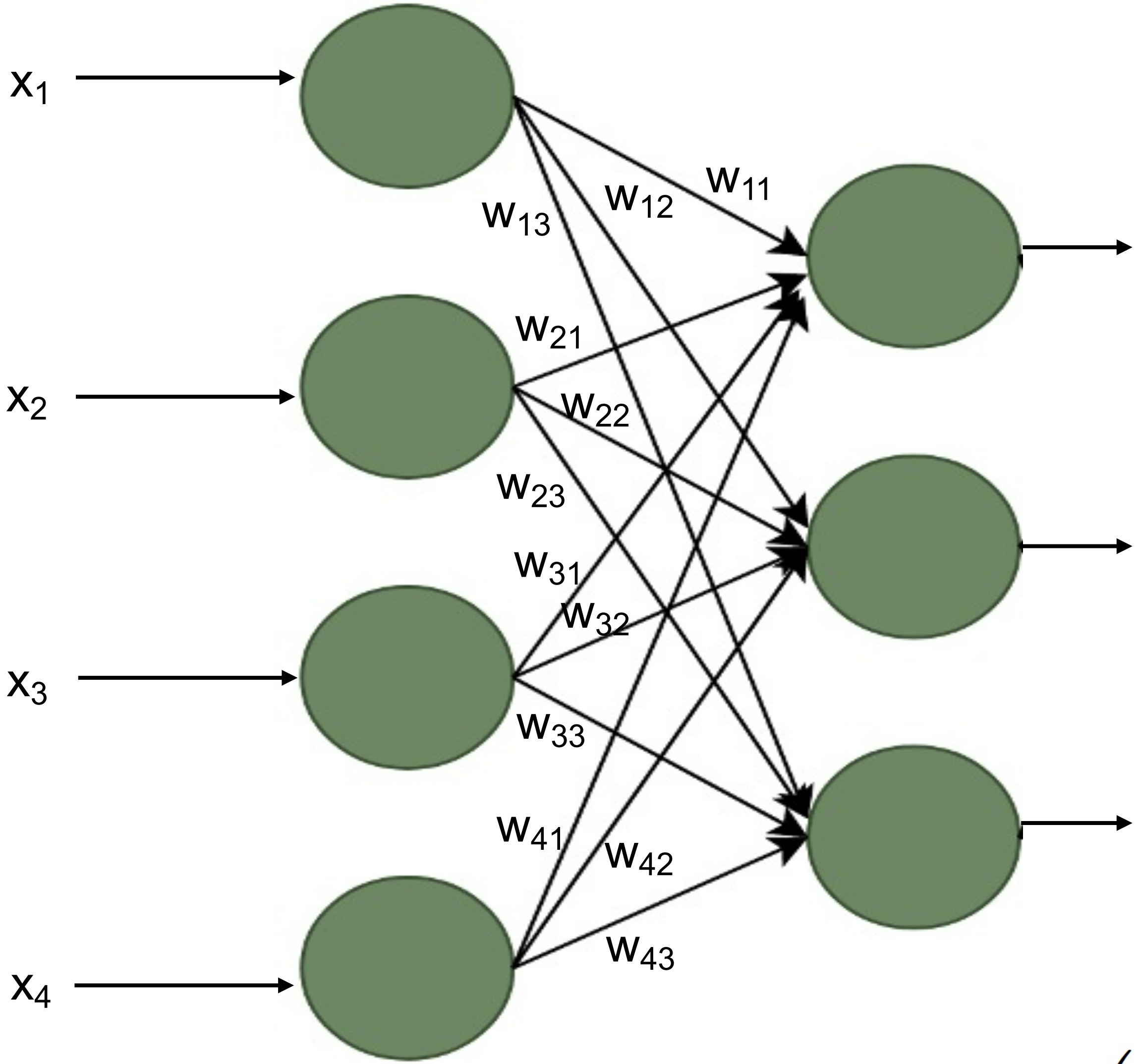
Overview (Hour 1)

- AI For Science
- Deep Learning and Large Language Model training
- Data parallel training
- Model parallel training
  - Fully Sharded Data Parallel (FSDP)
  - Tensor Parallel (TP)
  - Pipeline Parallel (PP)
  - Hybrid Parallelism
- Case Study: Forge

Internals of an Artificial Neural Network



$$X = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix}, W_1 = \begin{bmatrix} w_{11}^1 & w_{12}^1 & w_{13}^1 \\ w_{21}^1 & w_{22}^1 & w_{23}^1 \\ w_{31}^1 & w_{32}^1 & w_{33}^1 \\ w_{41}^1 & w_{42}^1 & w_{43}^1 \end{bmatrix}, W_2 = \begin{bmatrix} w_{11}^2 & w_{12}^2 & w_{13}^2 \\ w_{21}^2 & w_{22}^2 & w_{23}^2 \\ w_{31}^2 & w_{32}^2 & w_{33}^2 \end{bmatrix}, W_3 = \begin{bmatrix} w_{11}^3 & w_{12}^3 \\ w_{21}^3 & w_{22}^3 \\ w_{31}^3 & w_{32}^3 \end{bmatrix}$$



$$\sigma(x_1 w_{11} + x_2 w_{21} + x_3 w_{31} + x_4 w_{41})$$

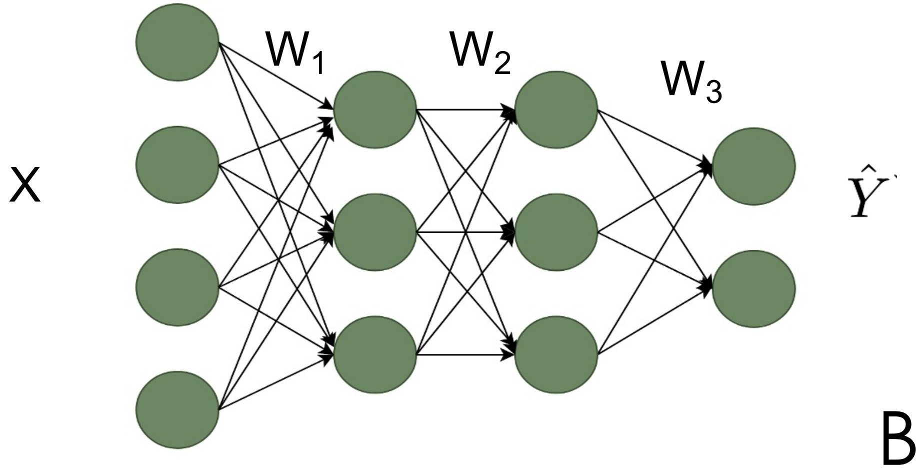
$$\sigma(x_1 w_{12} + x_2 w_{22} + x_3 w_{32} + x_4 w_{42})$$

$$\sigma(x_1 w_{13} + x_2 w_{23} + x_3 w_{33} + x_4 w_{43})$$

$$O_1 = \sigma\left(\begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix} \right) \times \begin{bmatrix} w_{11}^1 & w_{12}^1 & w_{13}^1 \\ w_{21}^1 & w_{22}^1 & w_{23}^1 \\ w_{31}^1 & w_{32}^1 & w_{33}^1 \\ w_{41}^1 & w_{42}^1 & w_{43}^1 \end{bmatrix}$$

$$= \sigma(X \times W_1)$$

Training a Neural Network



- Forward Pass

$$O_1 = \sigma(X \times W_1)$$

$$O_2 = \sigma(O_1 \times W_2)$$

$$\hat{Y} = O_3 = \sigma(O_2 \times W_3)$$

Backward Pass

$$L = Loss = CrossEntropyLoss(Y, \hat{Y})$$

$$G = Gradient = \frac{\delta L}{\delta W}$$

$$W' = W - \eta \times \frac{\delta L}{\delta W}$$

Transformer Model Architecture

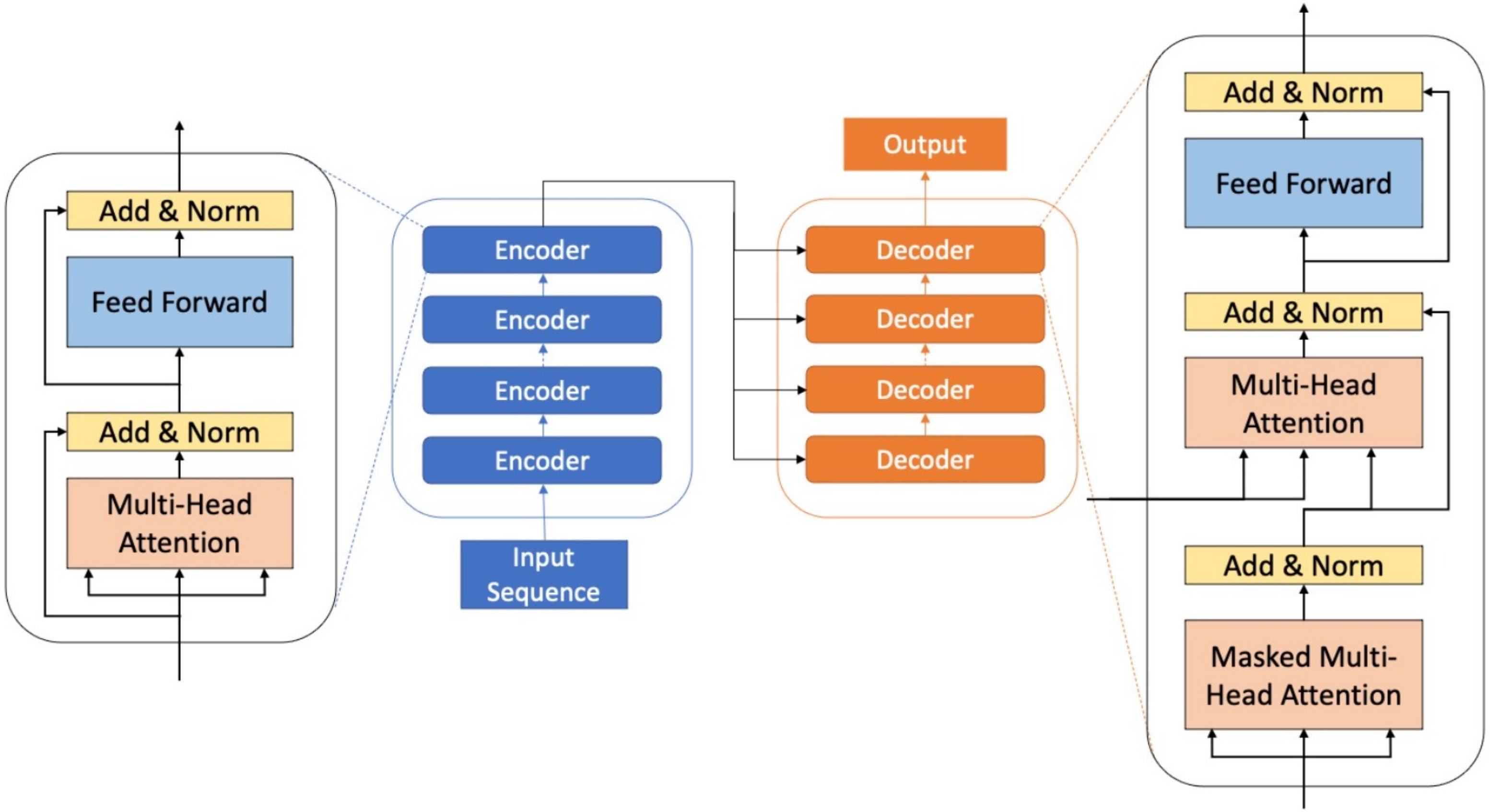
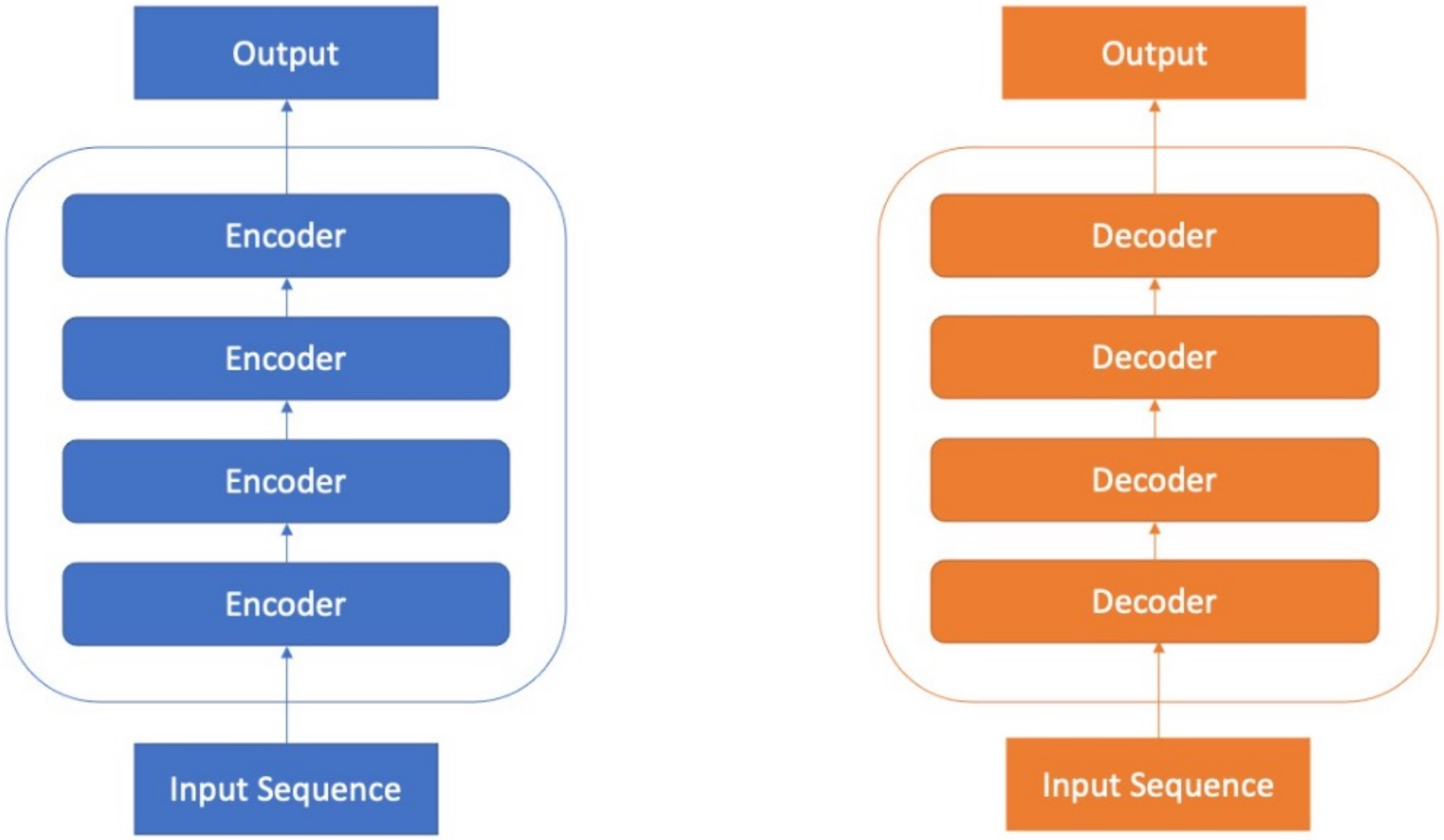
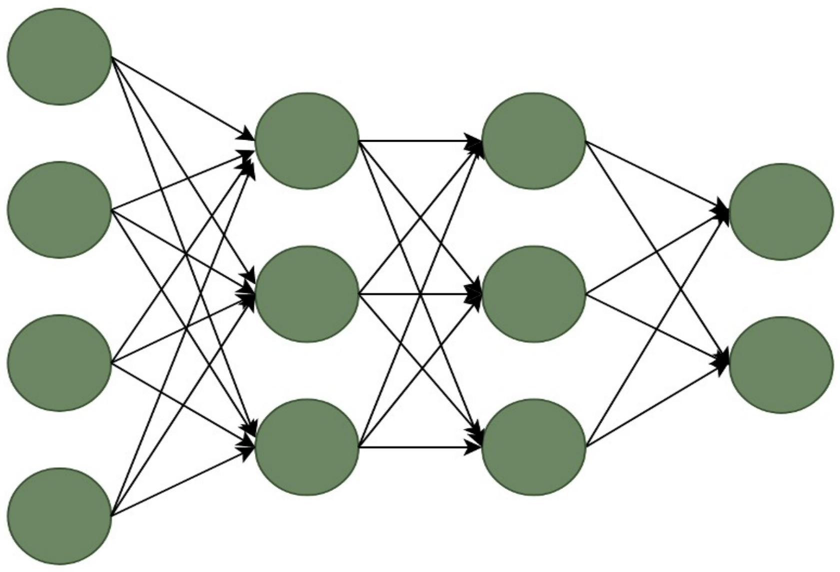


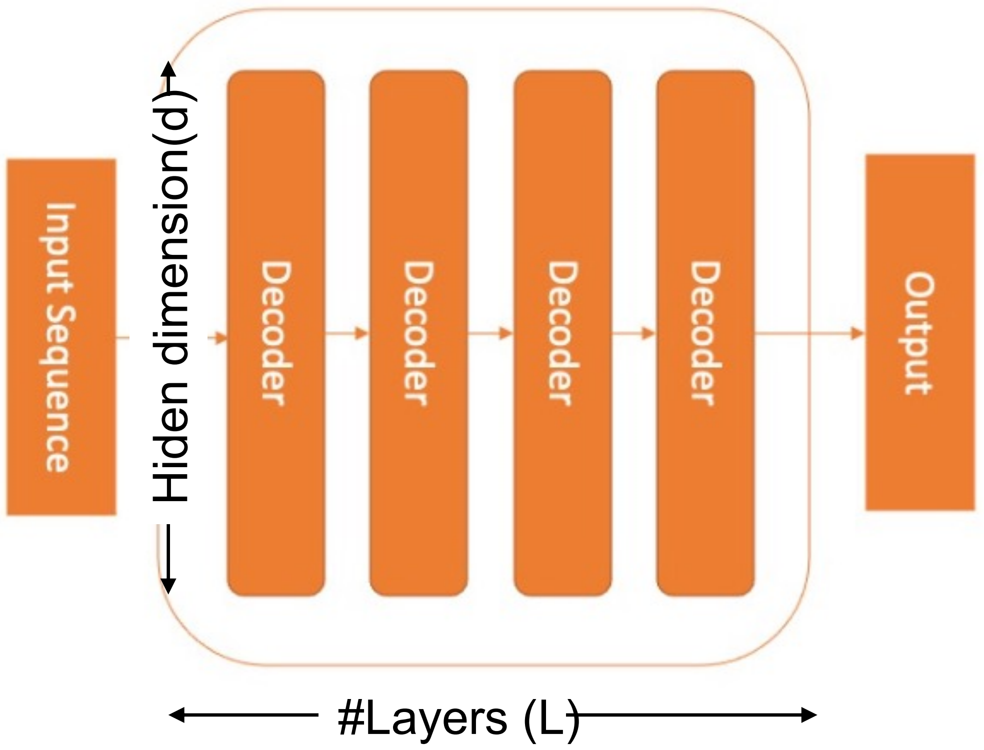
Fig. 1 Transformer architecture. A Transformer consists of stacked blocks of encoder followed by a stacked blocks of decoders. The leftmost block is the detailed structure of an encoder block, and the rightmost block is the detailed structure of a decoder block.

Large Language Models (LLM): BERT vs GPT



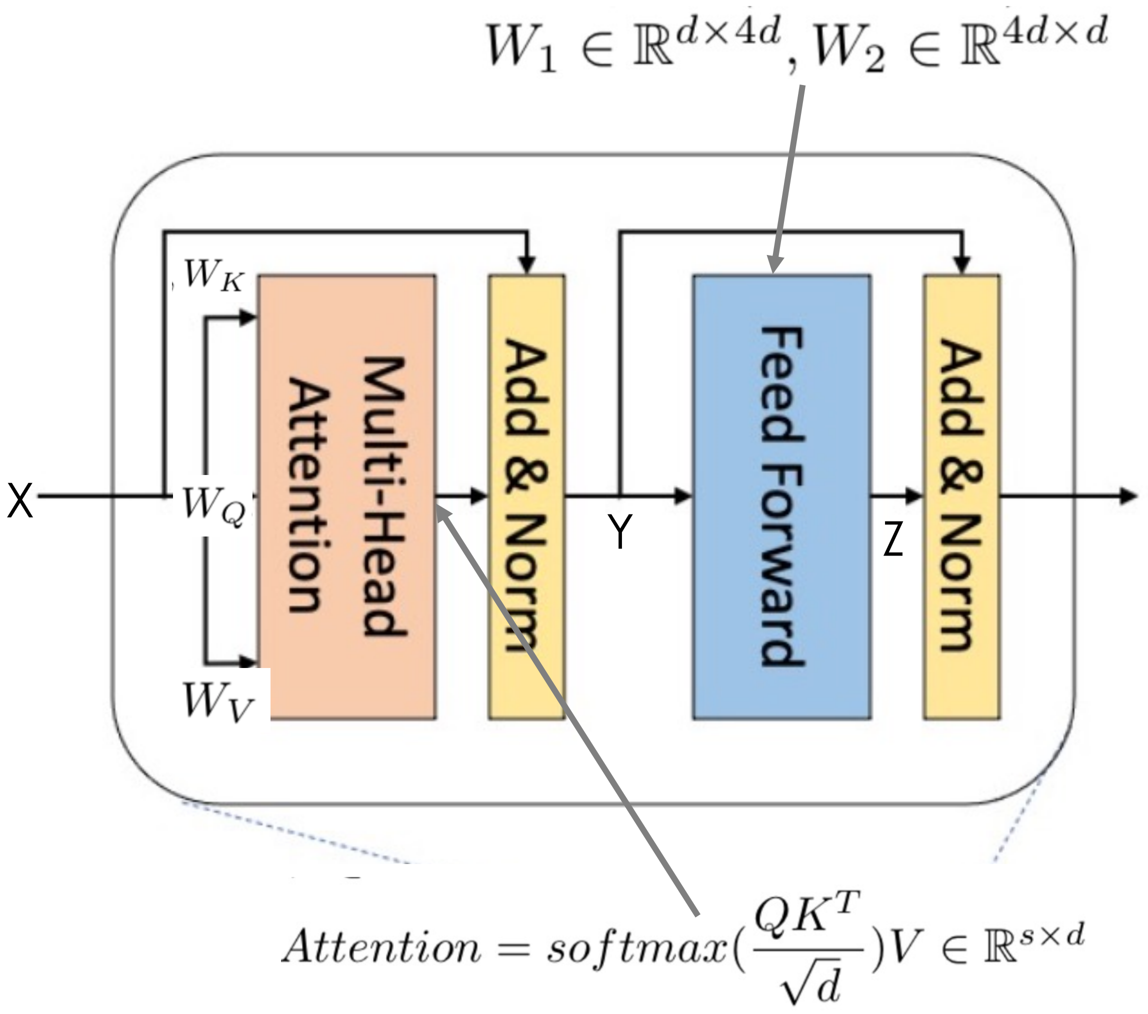
Transformers are Neural Networks





Forward Pass

Weights to update: W_K, W_Q, W_V, W_1, W_2



Let, $X \in \mathbb{R}^{s\times d}, W_K, W_Q, W_V \in \mathbb{R}^{d\times d}$.
Then, $K = XW_K, Q = XW_Q, V = XW_V$,
$K, Q, V \in \mathbb{R}^{s\times d}$.

And, $Attention = softmax(\frac{QK^T}{\sqrt{d}})V \in \mathbb{R}^{s\times d}$

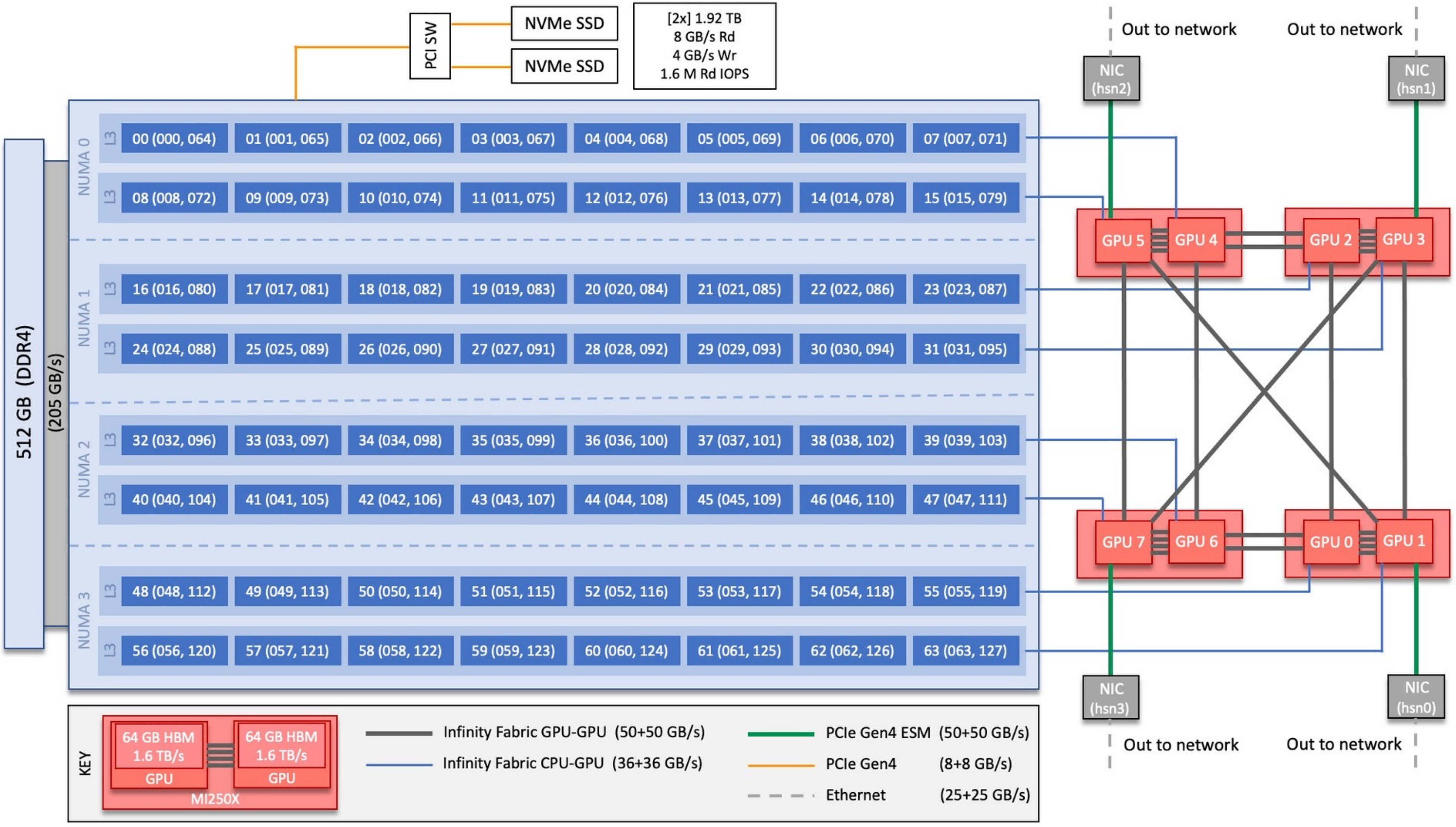
...

$Y = func(Attention) \in \mathbb{R}^{s\times d}$
$W_1 \in \mathbb{R}^{d\times 4d}, W_2 \in \mathbb{R}^{4d\times d}$
$Z = GeLU(YW_1) \times W_2 \in \mathbb{R}^{s\times d}$

Overview (Hour 1)

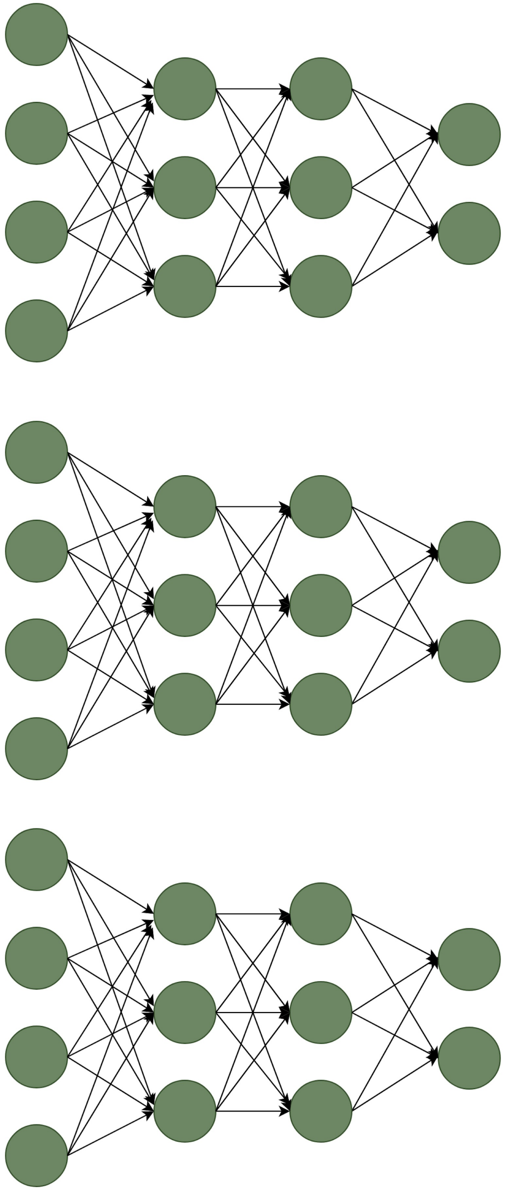
- AI For Science
- Deep Learning and Large Language Model training
- Data parallel training
- Model parallel training
  - Fully Sharded Data Parallel (FSDP)
  - Tensor Parallel (TP)
  - Pipeline Parallel (PP)
  - Hybrid Parallelism
- Case Study: Forge

Frontier Node Architecture



Training DL Models with Large Data

Batch 1.1
Batch 1.2
Batch 1.3



[Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour]

Data Parallel Training With PyTorch DDP

- DDP == Distributed Data Parallel
- A group of processes will work together to perform data parallel training
- One process will be designated as the “Master” and every process needs to know the IP address (MASTER_ADDR) of it
- Identifying the “MASTER_ADDR” varies from Summit to Frontier
- With this, you initialize DDP and wrap your model with model = DDP(model)

Setup DDP on Frontier

- There are a few ways. For this tutorial, we will pass the MASTER_ADDR as argument

```
scontrol show hostnames $SLURM_NODELIST > job.node.list
input="./job.node.list"
readarray -t arr <"$input"
first=${arr[0]}
ips=`ssh $first hostname -I`
read -ra arr <<< ${ips}
export MASTER_ADDR=${arr[0]}
```

```python
def setup_distributed_env(init_method=None, rank = 0, world_size=16):
    from mpi4py import MPI
    comm = MPI.COMM_WORLD
    world_size = comm.Get_size()
    world_rank = rank = comm.Get_rank()
    backend = None
    os.environ['MASTER_ADDR'] = master_addr
    os.environ['MASTER_PORT'] = master_port
    os.environ['WORLD_SIZE'] = str(world_size)
    os.environ['RANK'] = str(world_rank)
    os.environ['LOCAL_RANK'] = "0"#str(world_rank % 8)
    print("initialization parameters:", init_method, backend, rank, world_size)
    torch.distributed.init_process_group(backend,
                                         timeout=default_pg_timeout,
                                         init_method=init_method,
                                         rank=rank,
                                         world_size=world_size)
```

```python
train_dataset = NPZDataset(train_data_dir)
test_dataset = NPZDataset(test_data_dir)
train_dataloader = torch.utils.data.DataLoader(train_dataset, …)
test_dataloader = torch.utils.data.DataLoader(test_dataset, …)
```

```python
class CNN(nn.Module):
    def __init__(self):
        …
    def forward(self, x):
        …
model = CNN()
```

```python
num_classes = 231
model = models.resnet50(pretrained=True)
num_ftrs = model.fc.in_features
model.fc = nn.Linear(num_ftrs,
num_classes)
```

```python
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
```

SCALING

```python
for epoch in range(1):
    for i, data in enumerate(train_dataloader):
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        optimizer.step()
```

Scaling *Up* and *Out* a model

- Move the model to GPU

```
device = "cuda"
model.to(device)
```

- Run on multiple GPU

```
model = DataParallel(model, device_ids =
[0, 1, 2])
```

- Run on multiple Nodes

```
setup_DDP(backend="nccl")
model = DDP(model, device_ids = [0, 1,
2])
```

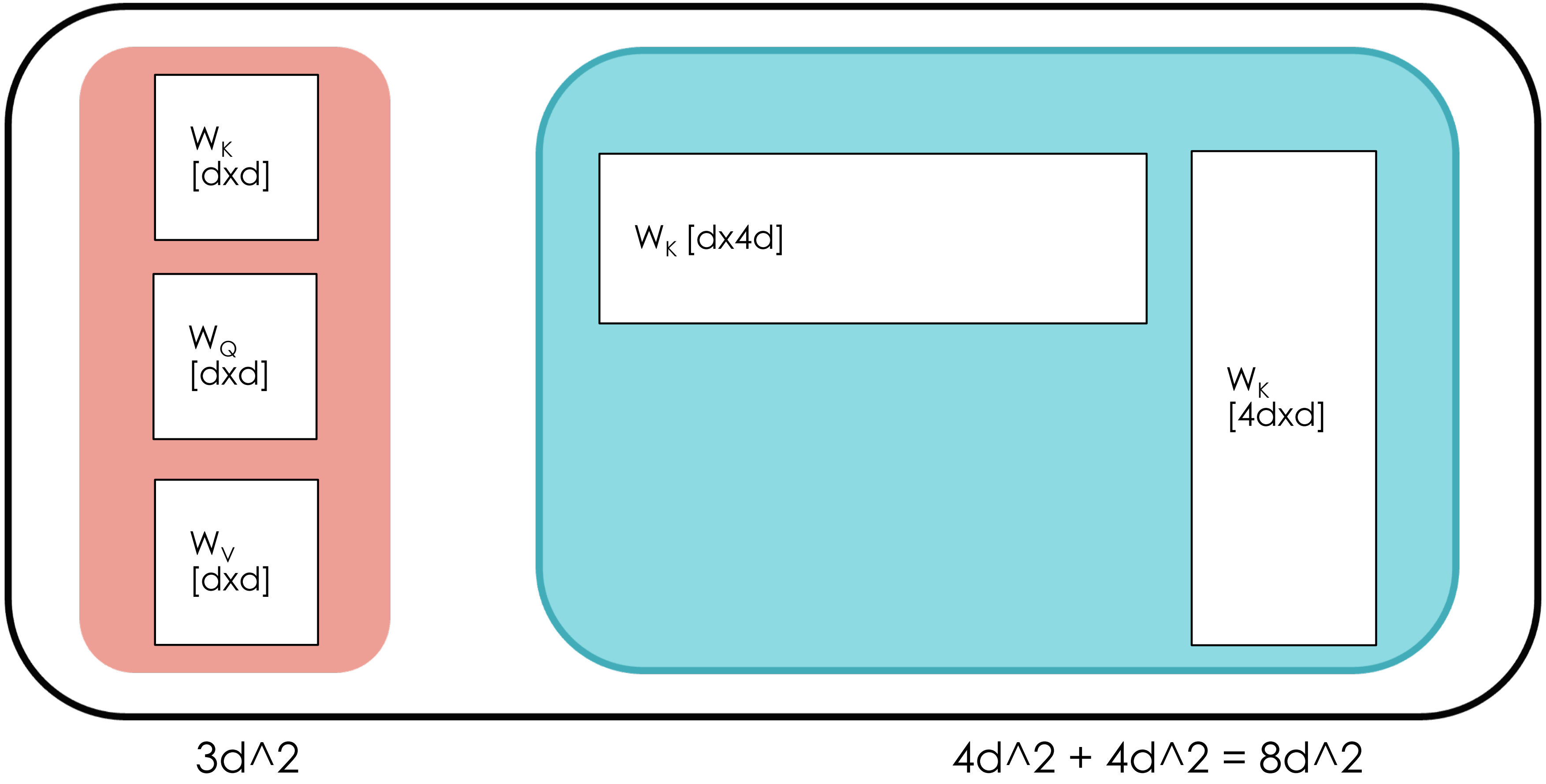
Overview (Hour 1)

- AI For Science
- Deep Learning and Large Language Model training
- Data parallel training
- Model parallel training
  - Fully Sharded Data Parallel (FSDP)
  - Tensor Parallel (TP)
  - Pipeline Parallel (PP)
  - Hybrid Parallelism
- Case Study: Forge

Overview (Hour 1)

- AI For Science
- Deep Learning and Large Language Model training
- Data parallel training
- Model parallel training
  - Distributed Data Parallel (DDP)
  - Fully Sharded Data Parallel (FSDP)
  - Tensor Parallel (TP)
  - Pipeline Parallel (PP)

Inside a Transformer Layer



Number of parameters
One layer: 11d^2
L layers: 12Ld^2

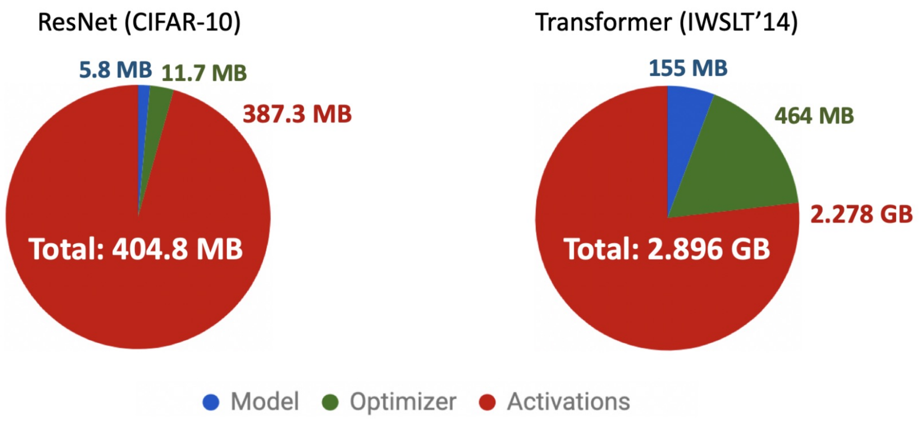
Memory Requirement During Training an LLM

Model Weights: Number of parameters: $12Ld^2$

- 4 bytes * number of parameters for fp32 training
- 6 bytes * number of parameters for mixed precision training (maintains a model in fp32 and one in fp16 in memory)

Optimizer States: Adam Optimizer uses two extra parameters (mean and variance)

- 8 bytes * number of parameters for normal AdamW (maintains 2 states)
- 2 bytes * number of parameters for 8-bit AdamW optimizers like bitsandbytes
- 4 bytes * number of parameters for optimizers like SGD with momentum (maintains only 1 state)



Gradients Same as number of parameters

- 4 bytes * number of parameters for either fp32 or mixed precision training (gradients are always kept in fp32)

Forward Activations Batch-size x output-nodes?

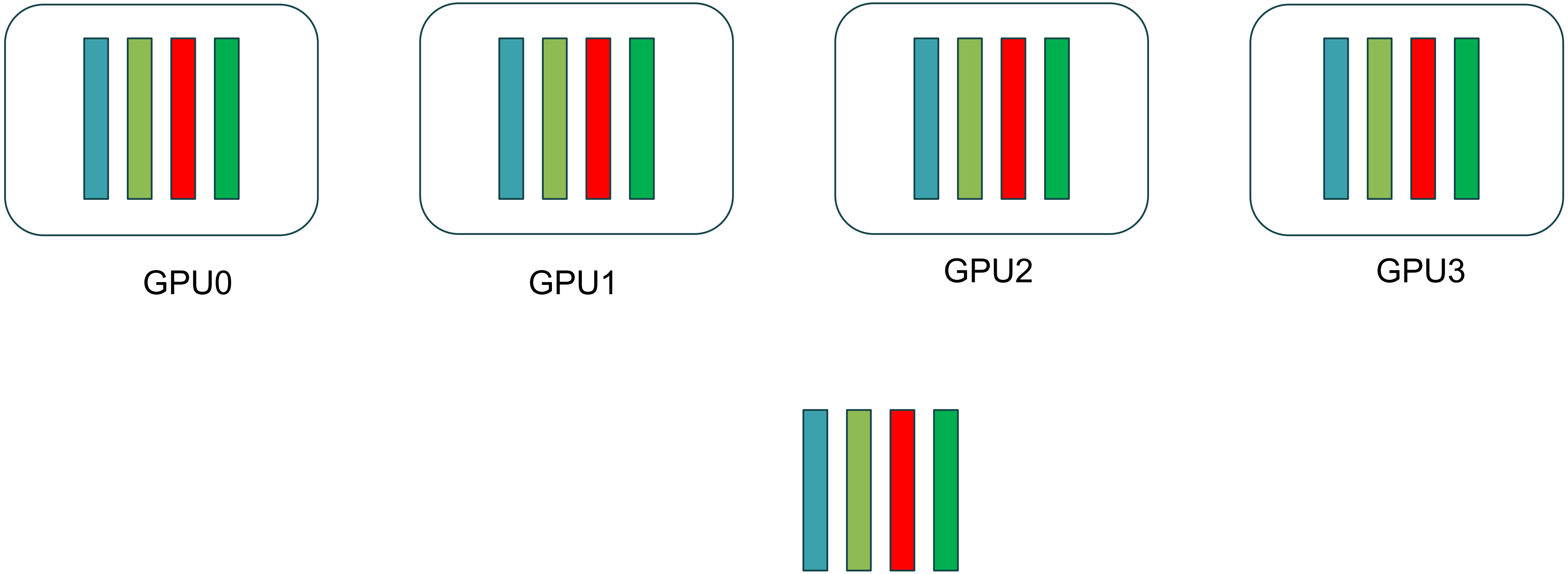
- size depends on many factors, the key ones being sequence length, hidden size and batch size.

Model Parallelism: Why and How

- Models (or Memory needed for their training) are too big to fit in a single GPU
- So, we need to break the model into pieces
- What's broken needs to be rebuilt from the pieces
- It's like Kintsugi, but with a more practical concern such as price of gold (comm. Latency)

Kintsugi

Data Parallelism

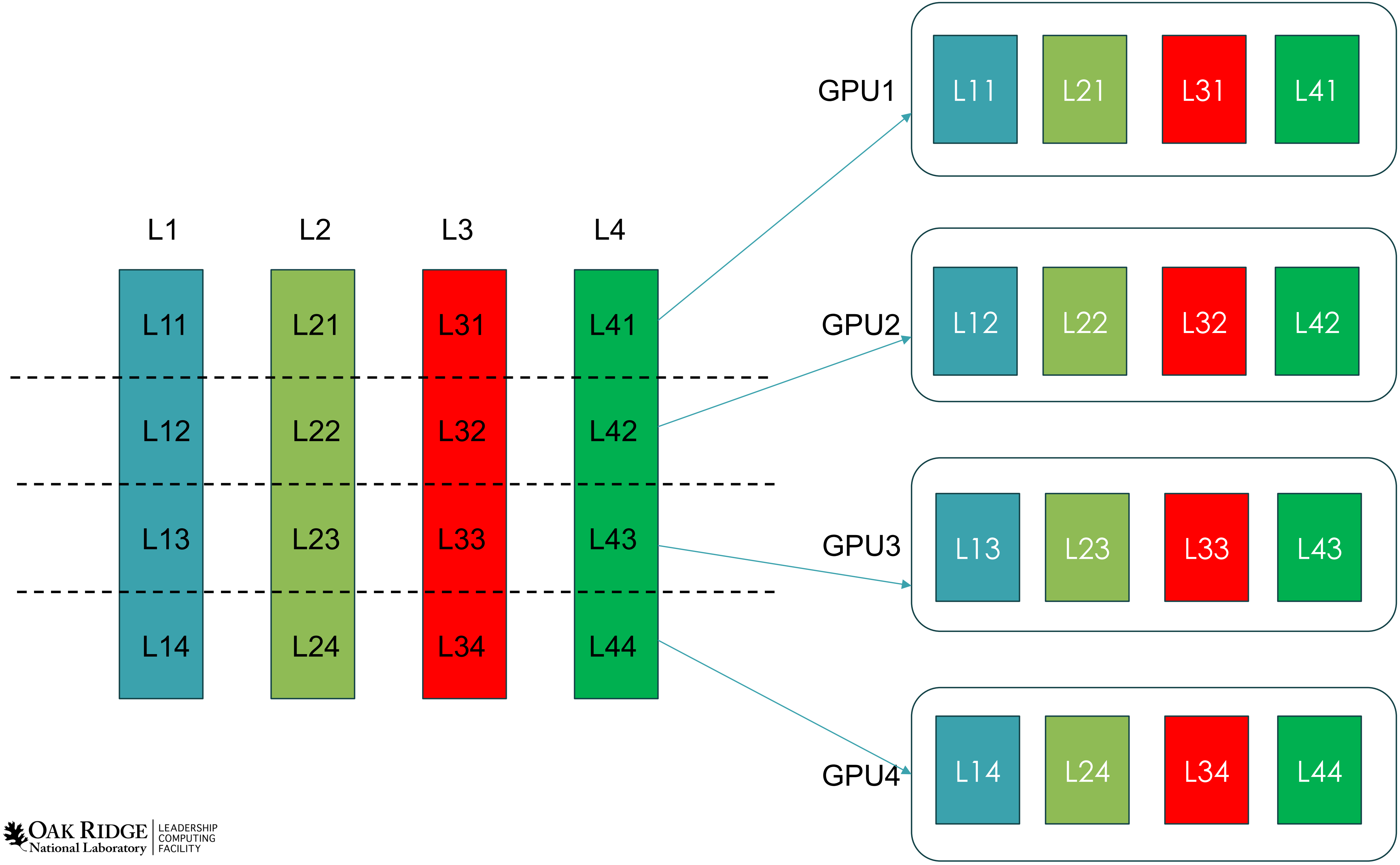


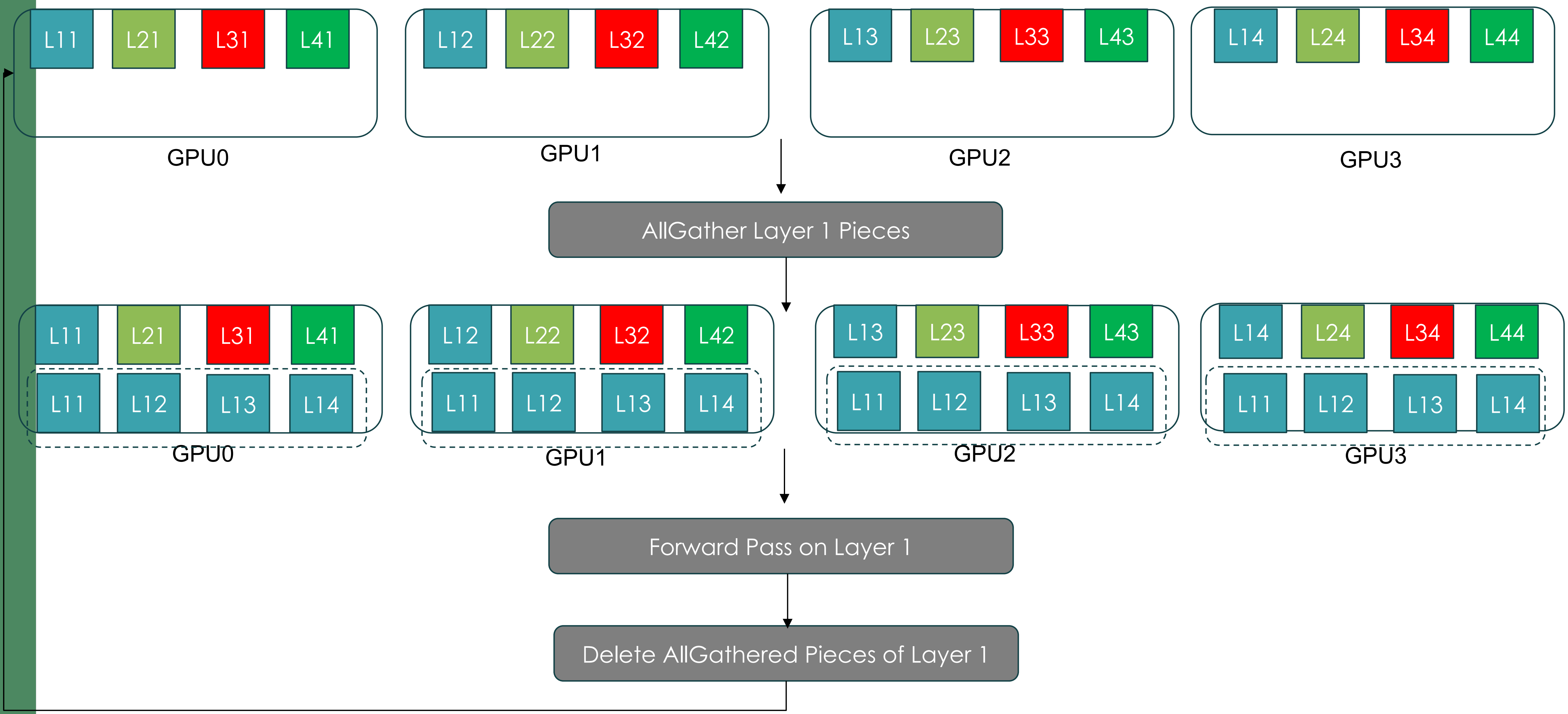
Overview (Hour 1)

- AI For Science
- Deep Learning and Large Language Model training
- Data parallel training
- Model parallel training
  - Sharded Data Parallel (ZeRO and FSDP)
  - Tensor Parallel (TP)
  - Pipeline Parallel (PP)
  - Hybrid Parallelism
- Case Study: Forge

Sharded Data Parallelism

- The model is too big to fit in a single GPU's memory
- One GPU has enough memory to fit a fraction of the model
- Slice the model with horizontal lines and place one horizontal slice in one GPU
- But, when a layer is being evaluated, each GPU should have the copy of that layer
- The GPU has enough memory to fit one full layer on top of its slice


L1
L2
L3
L4
L11
L21
L31
L41
L12
L22
L32
L42
L13
L23
L33
L43
L14
L24
L34
L44
GPU1
L11 L21 L31 L41
GPU2
L12 L22 L32 L42
GPU3
L13 L23 L33 L43
GPU4
L14 L24 L34 L44



L11
L21
L31
L41
GPU0
L12
L22
L32
L42
GPU1
L13
L23
L33
L43
GPU2
L14
L24
L34
L44
GPU3
AllGather Layer 1 Pieces
L11
L21
L31
L41
L11
L12
L13
L14
GPU0
L12
L22
L32
L42
L11
L12
L13
L14
GPU1
L13
L23
L33
L43
L11
L12
L13
L14
GPU2
L14
L24
L34
L44
L11
L12
L13
L14
GPU3
Forward Pass on Layer 1
Delete AllGathered Pieces of Layer 1

Two Methods to do Sharded Data Parallelism

- DeepSpeed ZeRO (by Microsoft) and PyTorch FSDP (by Meta)

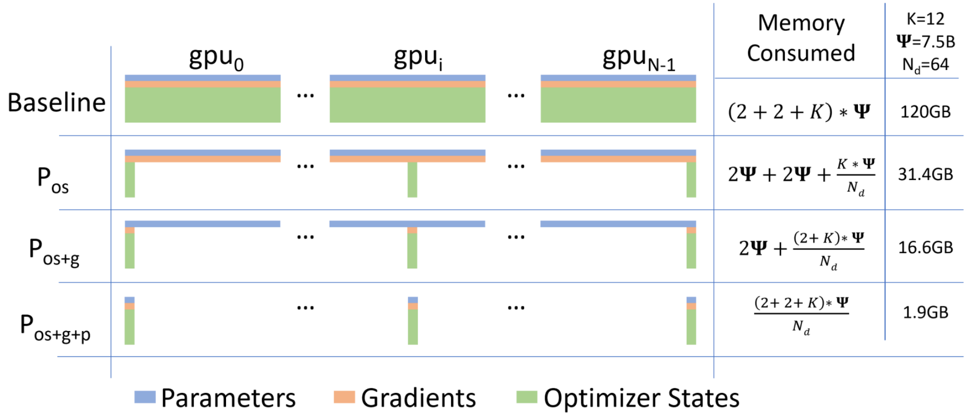
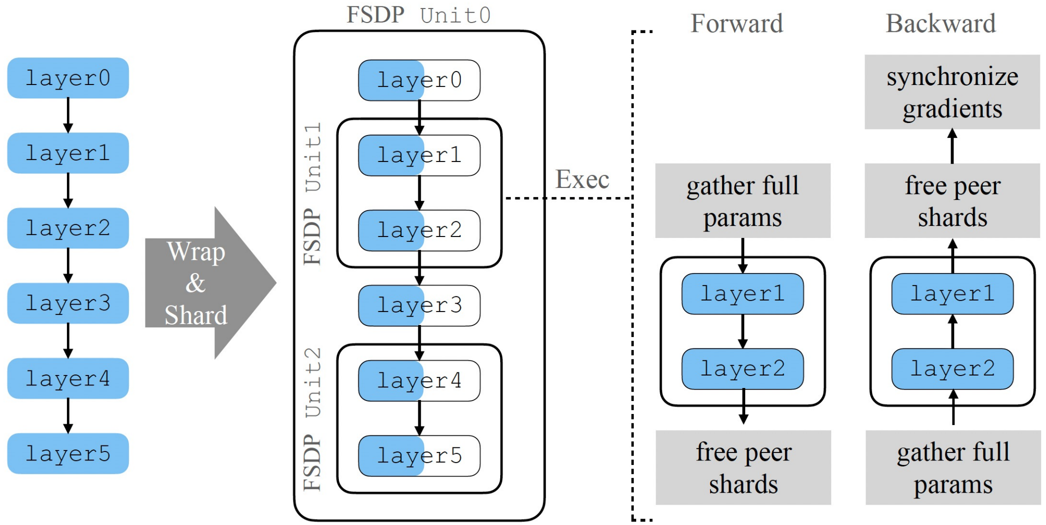


Figure 1: Comparing the per-device memory consumption of model states, with three stages of *ZeRO*-DP optimizations. Ψ denotes model size (number of parameters), K denotes the memory multiplier of optimizer states, and N_d denotes DP degree. In the example, we assume a model size of $\Psi = 7.5B$ and DP of $N_d = 64$ with $K = 12$ based on mixed-precision training with Adam optimizer.



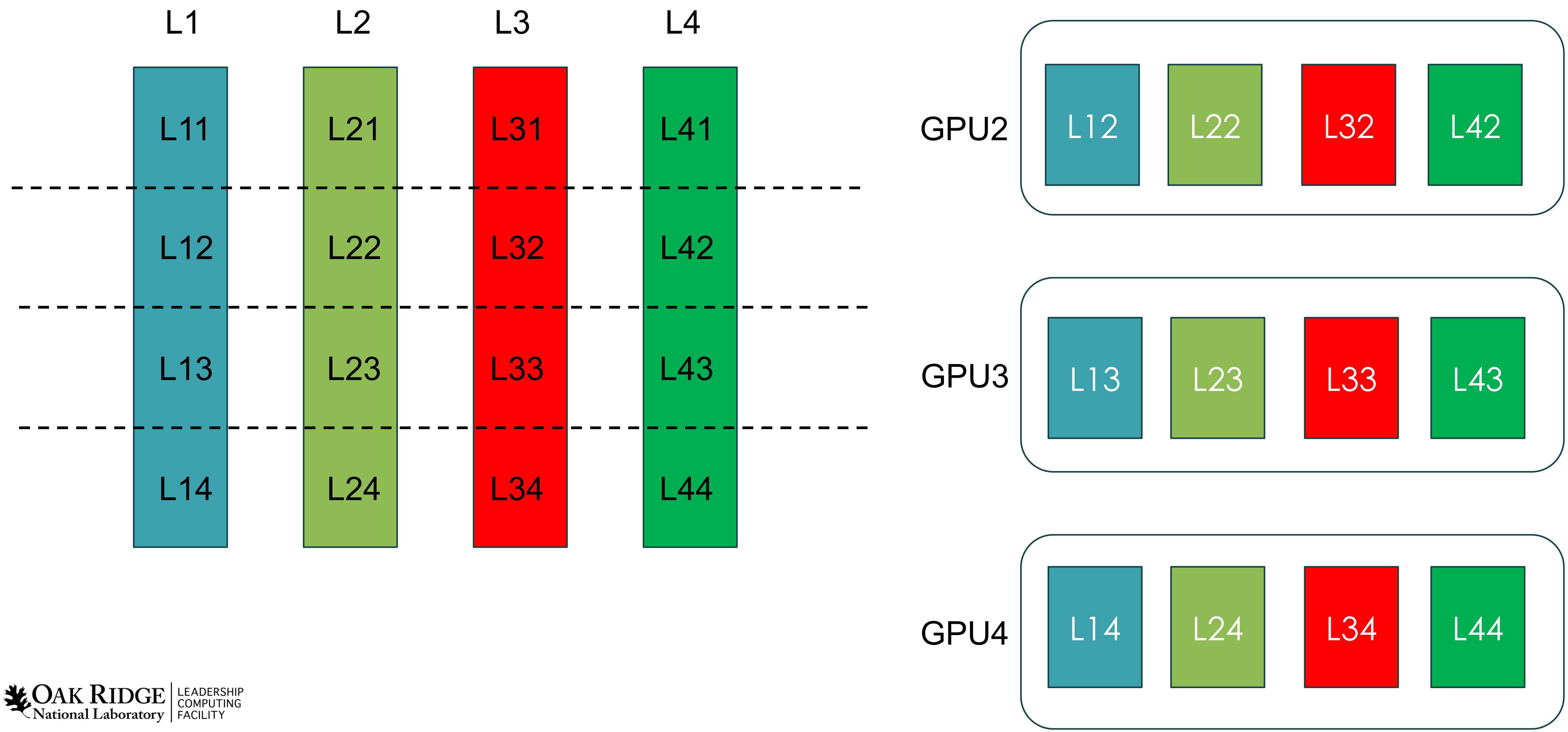
Overview (Hour 1)

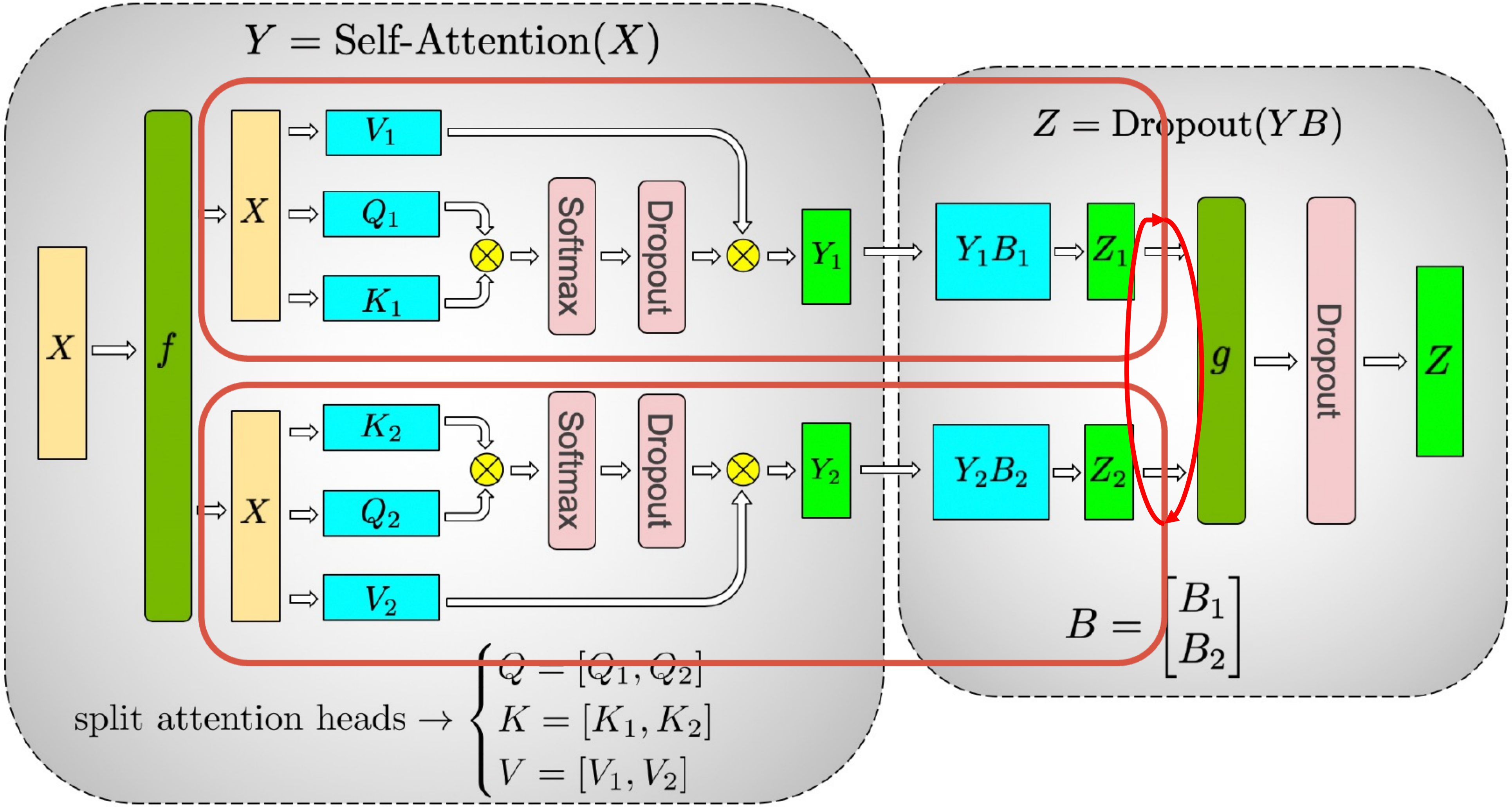
- AI For Science
- Deep Learning and Large Language Model training
- Data parallel training
- Model parallel training
  - Fully Sharded Data Parallel (FSDP)
  - Tensor Parallel (TP)
  - Pipeline Parallel (PP)
  - Hybrid Parallelism
- Case Study: Forge

Tensor Parallelism

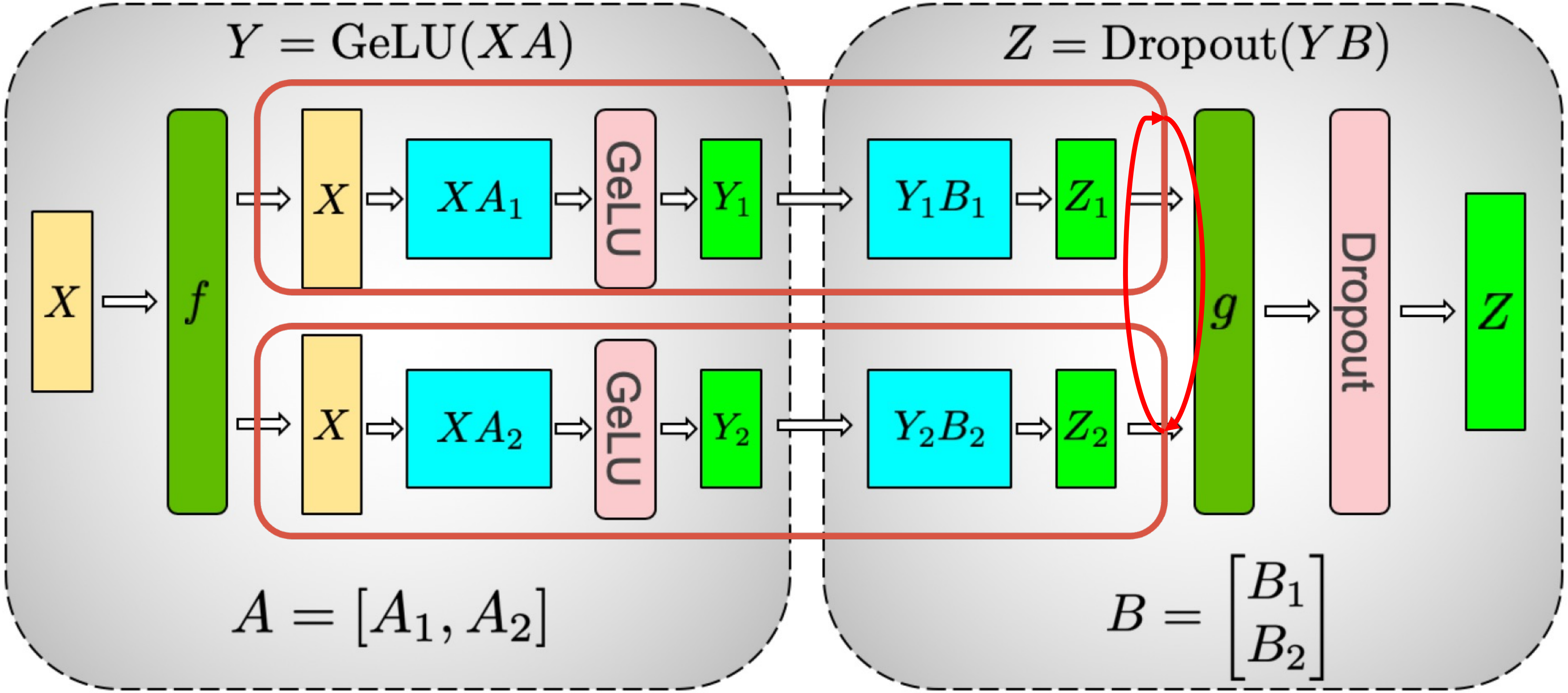
- Model is too large to fit in a GPU's memory
- We slice the model with horizontal line, and the GPU memory is large enough to fit one slice.
- Unlike sharded data parallelism, this is not data parallelism, the same data gets evaluated by different part of the same layer, and the output gets combined.

Tensor Parallelism (TP=4)





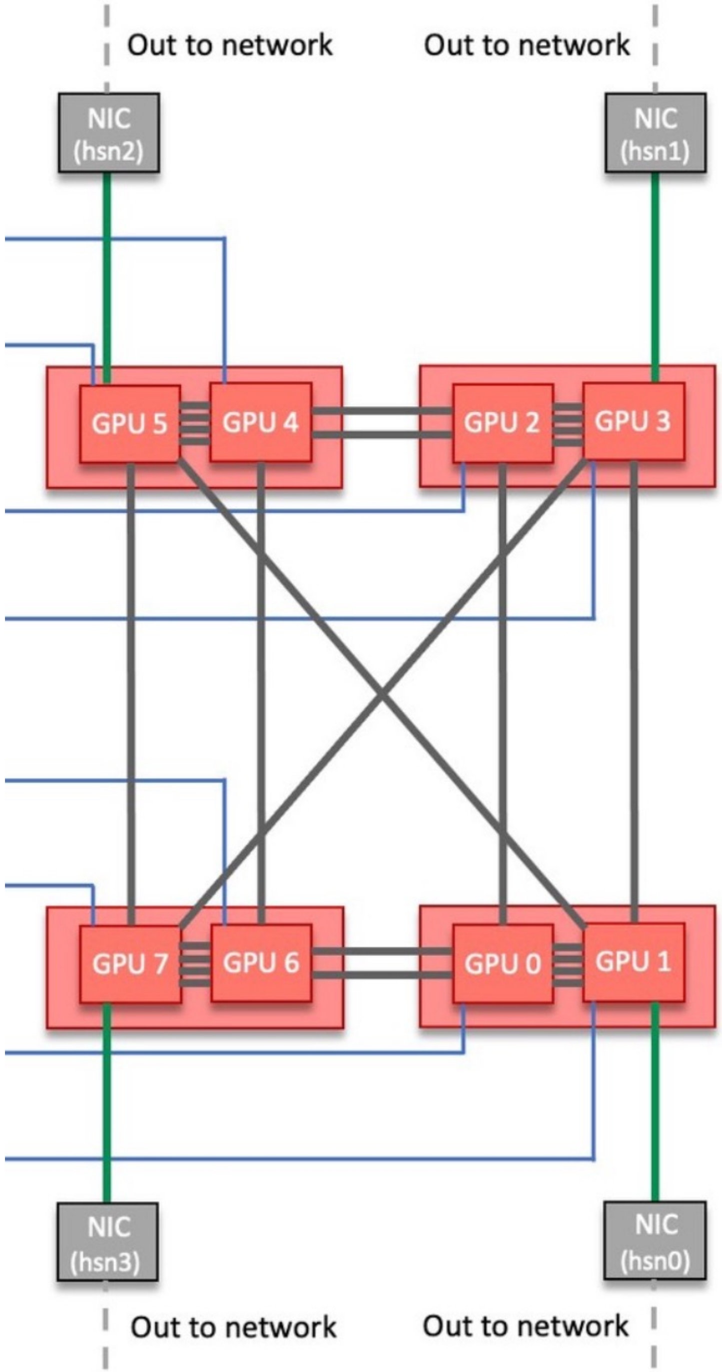
$Y = \text{Self-Attention}(X)$
$Z = \text{Dropout}(YB)$
X
f
X
V_1
Q_1
K_1
Softmax
Dropout
Y_1
Y_1B_1
Z_1
X
K_2
Q_2
V_2
Softmax
Dropout
Y_2
Y_2B_2
Z_2
g
Dropout
Z
$B = \begin{bmatrix} B_1 \\ B_2 \end{bmatrix}$
split attention heads $\rightarrow \begin{cases} Q = [Q_1, Q_2] \\ K = [K_1, K_2] \\ V = [V_1, V_2] \end{cases}$



$Y = \mathrm{GeLU}(XA)$
$Z = \mathrm{Dropout}(YB)$
X
f
X
XA_1
GeLU
Y_1
Y_1B_1
Z_1
X
XA_2
GeLU
Y_2
Y_2B_2
Z_2
g
Dropout
Z
$A = [A_1, A_2]$
$B = \begin{bmatrix} B_1 \\ B_2 \end{bmatrix}$

Limitations of Tensor Parallelism

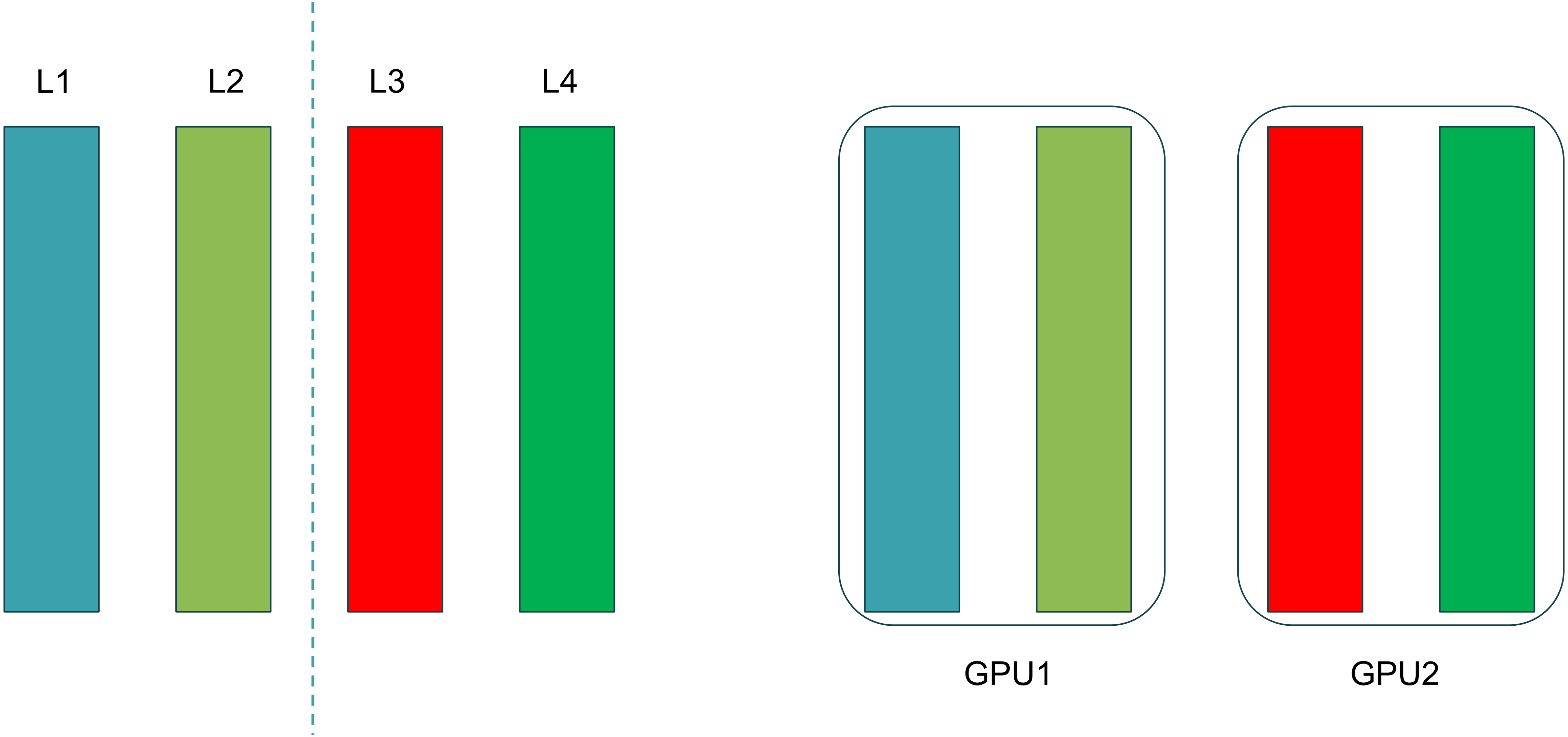
- Like Sharded data parallelism, Tensor Parallelism requires frequent AllReduce communication after every layer
- Instead of model weights, the intermediate outputs get AllReduced
- Tensor Parallel (TP) size is limited by the number of GPUs in a node (6 for Summit, 8 for Frontier)
- TP > 6/8 works, but the communication requires crossing node boundary, that introduces larger communication latency



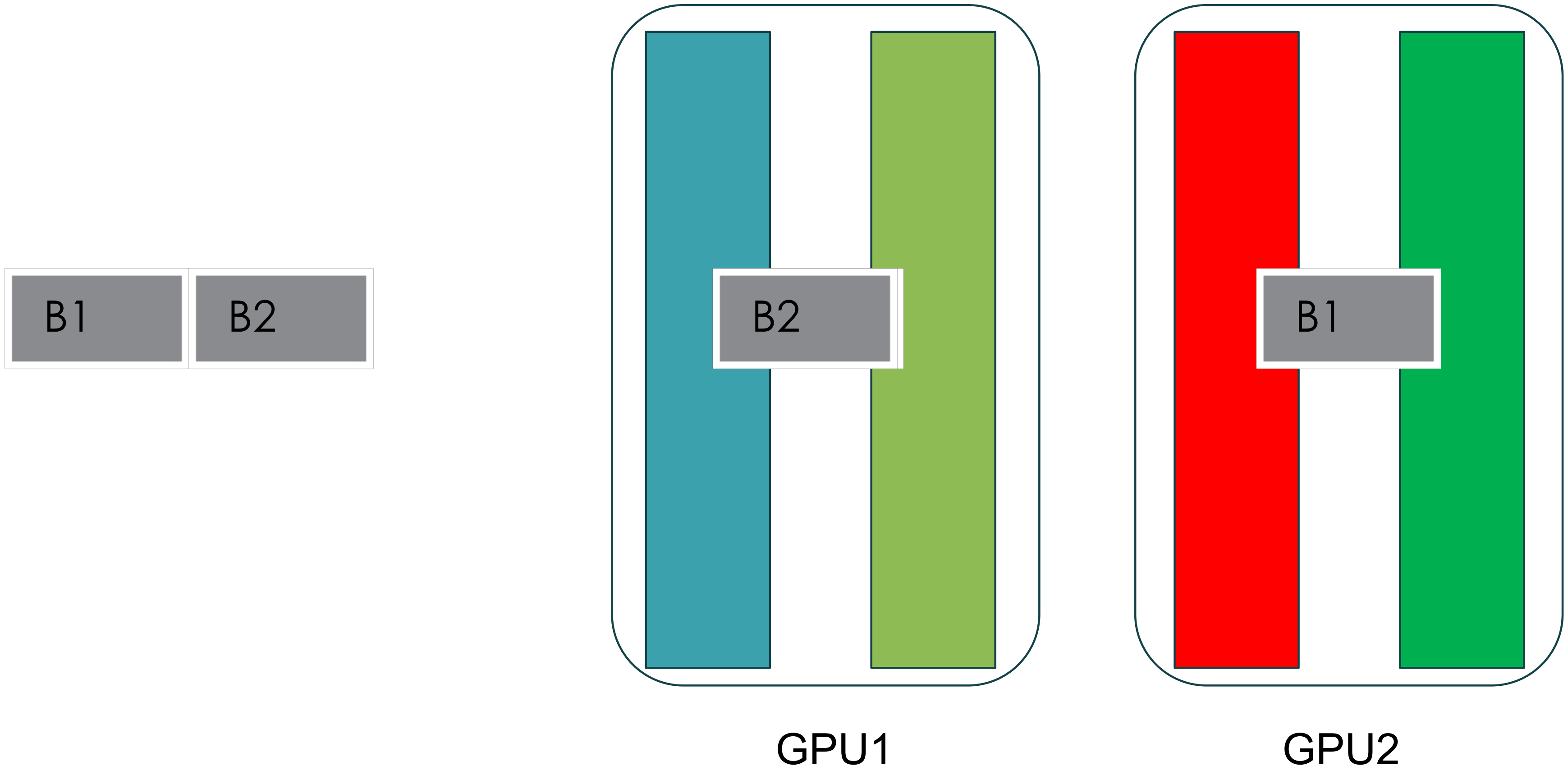
Overview (Hour 1)

- AI For Science
- Deep Learning and Large Language Model training
- Data parallel training
- Model parallel training
  - Sharded Data Parallel (FSDP)
  - Tensor Parallel (TP)
  - Pipeline Parallel (PP)
  - Hybrid Parallelism
- Case Study: Forge

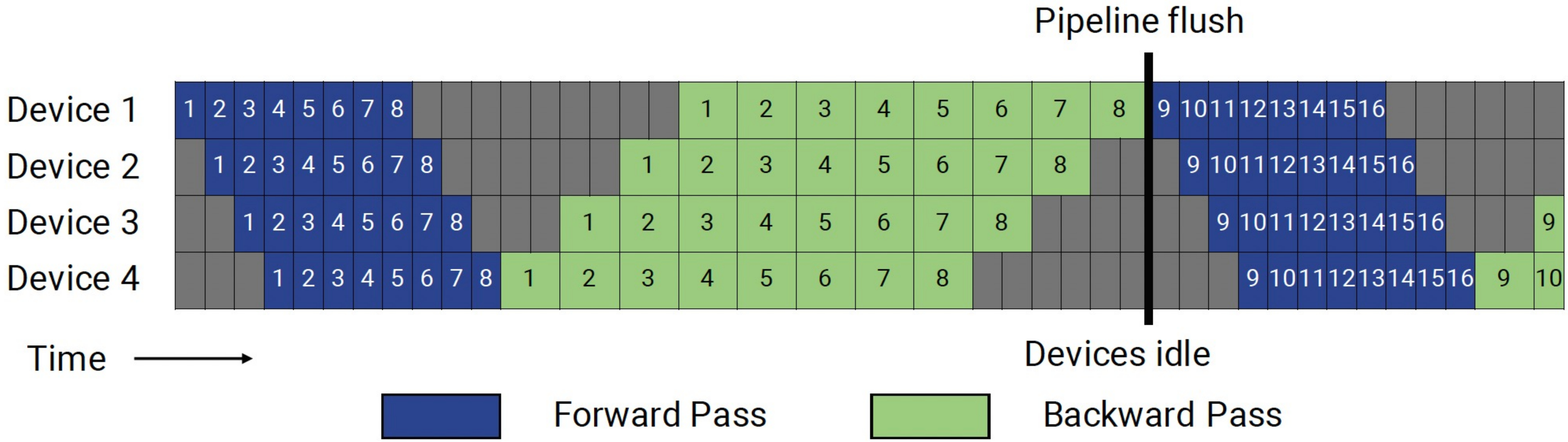
Pipeline Parallelism (PP = 2)



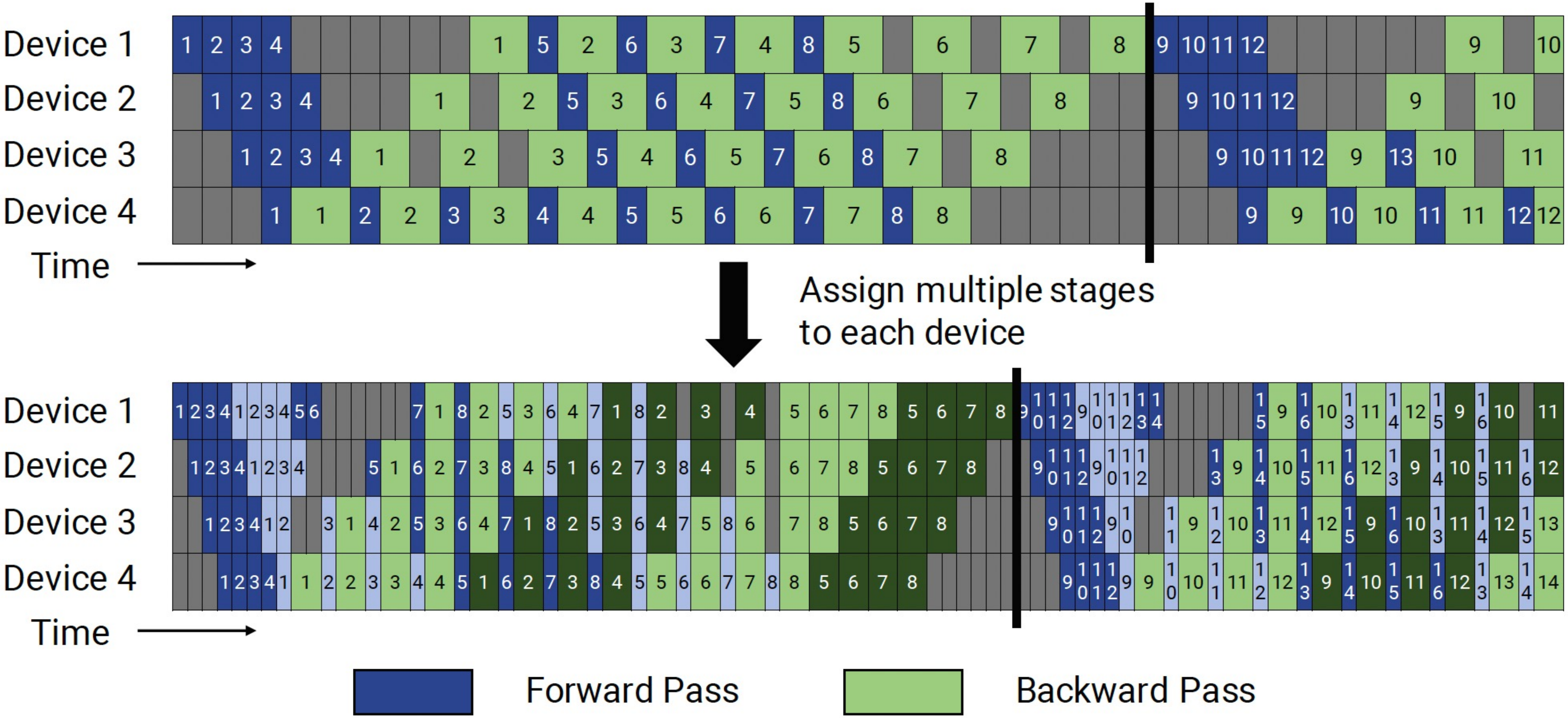
Pipeline Parallelism



Pipeline Parallelism (Gpipe)



Pipeline Parallelism



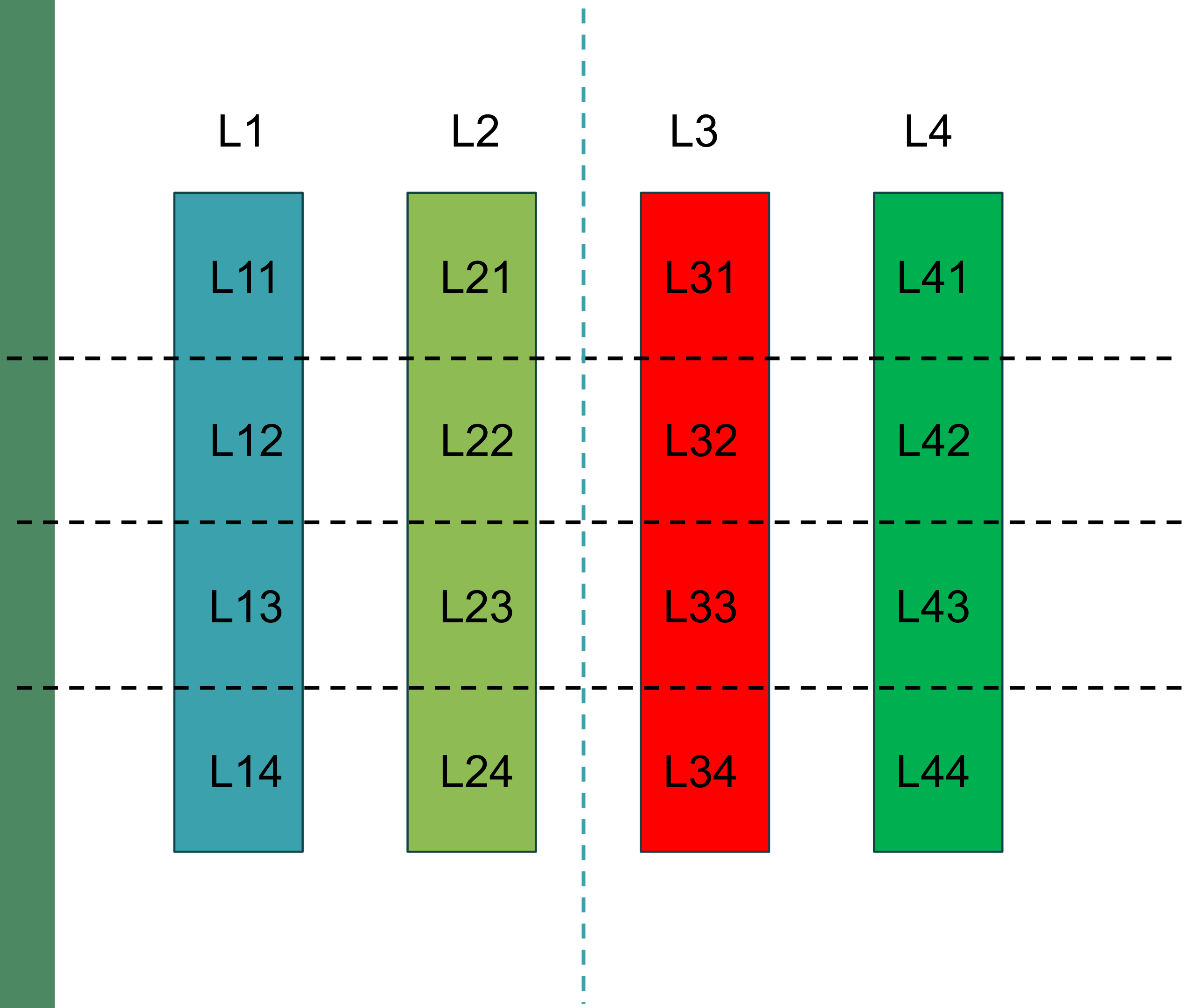
Data vs. Sharded vs. Tensor vs. Pipeline Parallelism

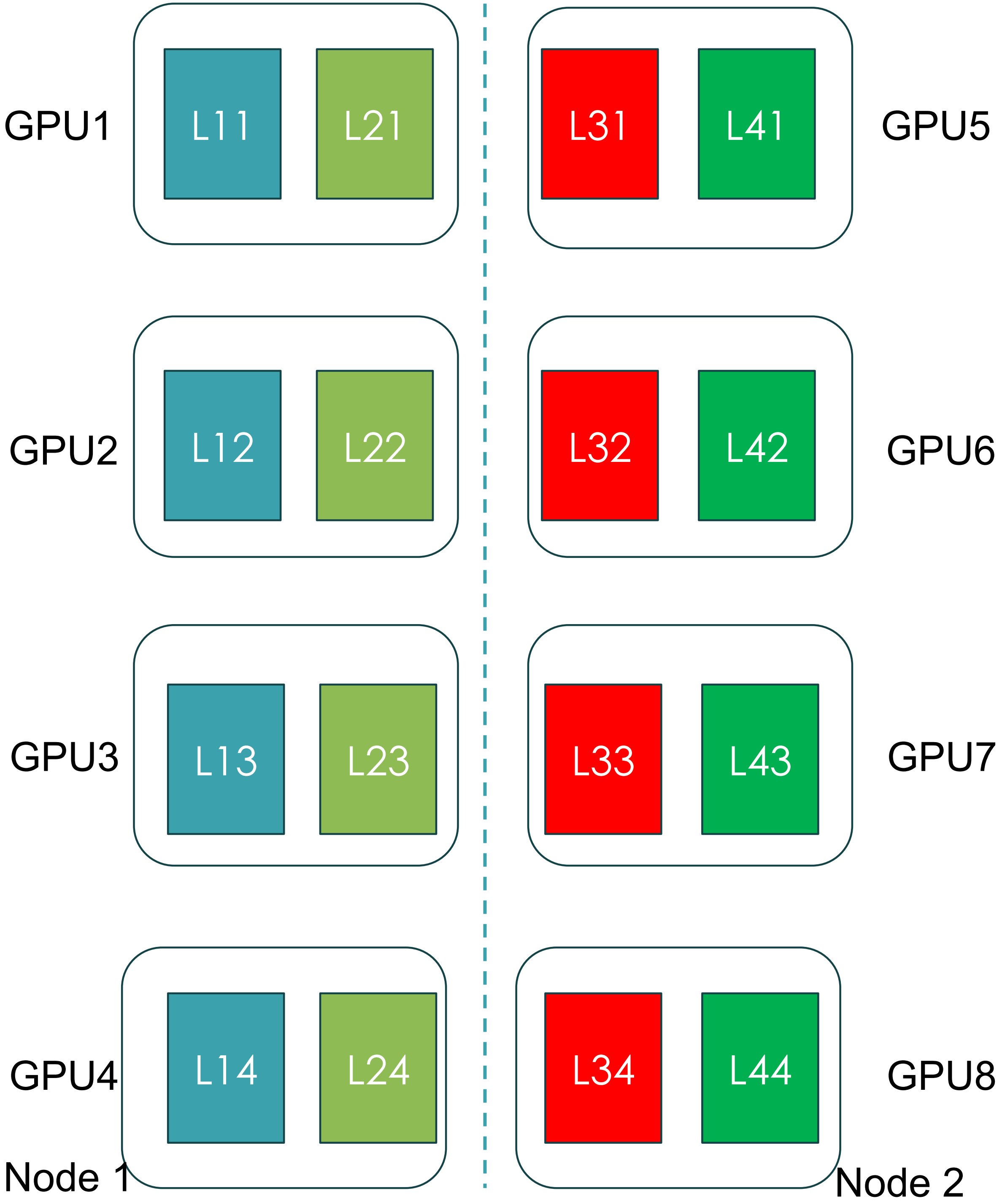
	Data Parallelism	**Sharded Data Parallelism**	**Tensor Parallelism**	**Pipeline Parallelism**
Dimension of distribution	Full Model gets replicated	Horizontal Slices + one layer replicated	Horizontal Slices	Vertical slices
Model/Output	Output	Model	Output	Output
Communication frequency	Least frequent	Most frequent	Most frequent	Intermediate
Communication volume	Large volume	Small	Small	Small
Limitation	High-volume communication	Frequent low-volume communication	Frequent low-volume communication	Bubbles from insufficient overlap
Advantage	Necessary for consuming large data	Allows large model training	Allows large model training	Can hide latency with smaller bubbles

Overview (Hour 1)

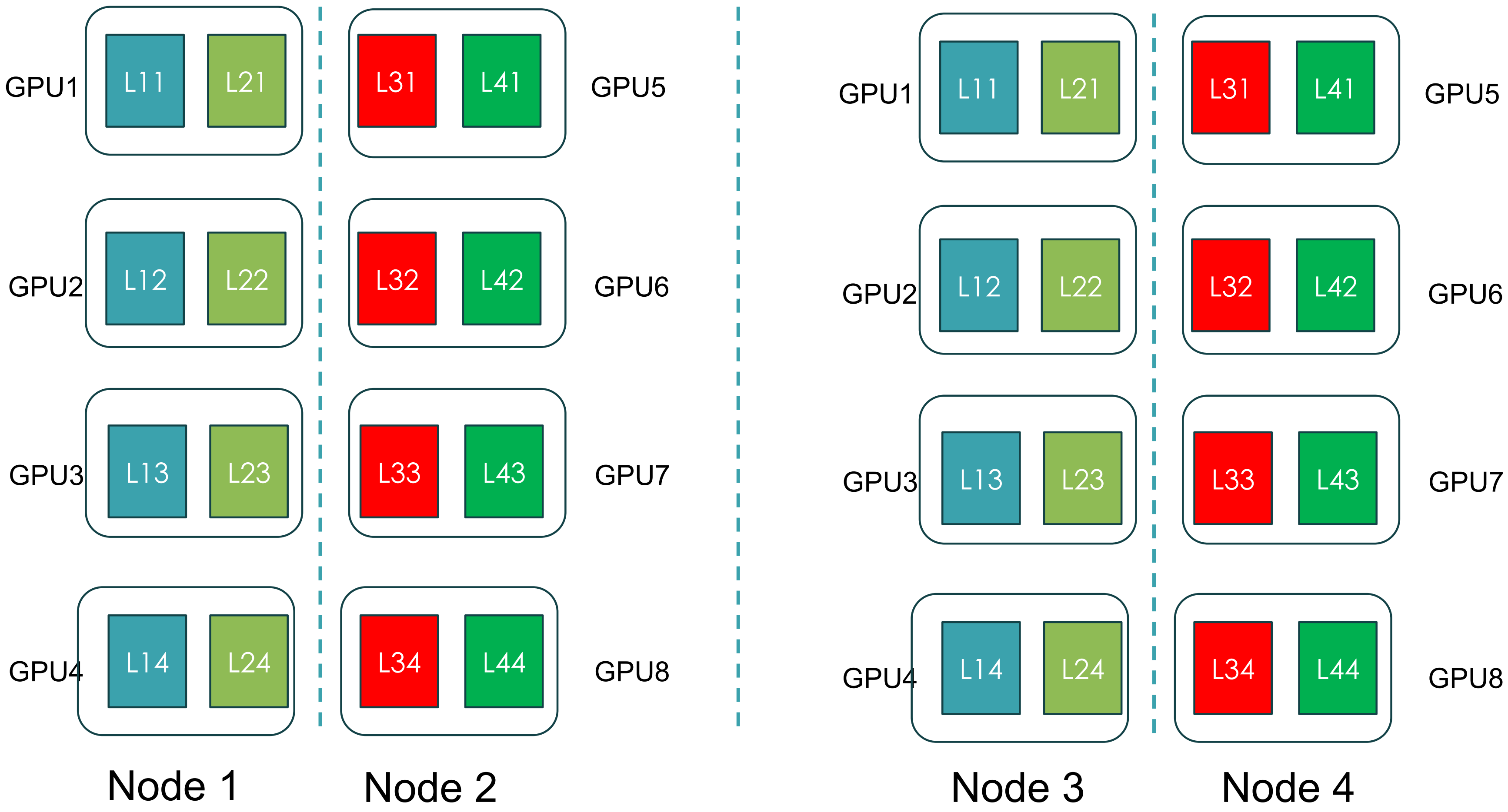
- AI For Science
- Deep Learning and Large Language Model training
- Data parallel training
- Model parallel training
  - Sharded Data Parallel (ZeRO and FSDP)
  - Tensor Parallel (TP)
  - Pipeline Parallel (PP)
  - Hybrid Parallelism
- Case Study: Forge

Hybrid (TP=4, PP=2)





Hybrid (TP=4, PP=2, DP=2)



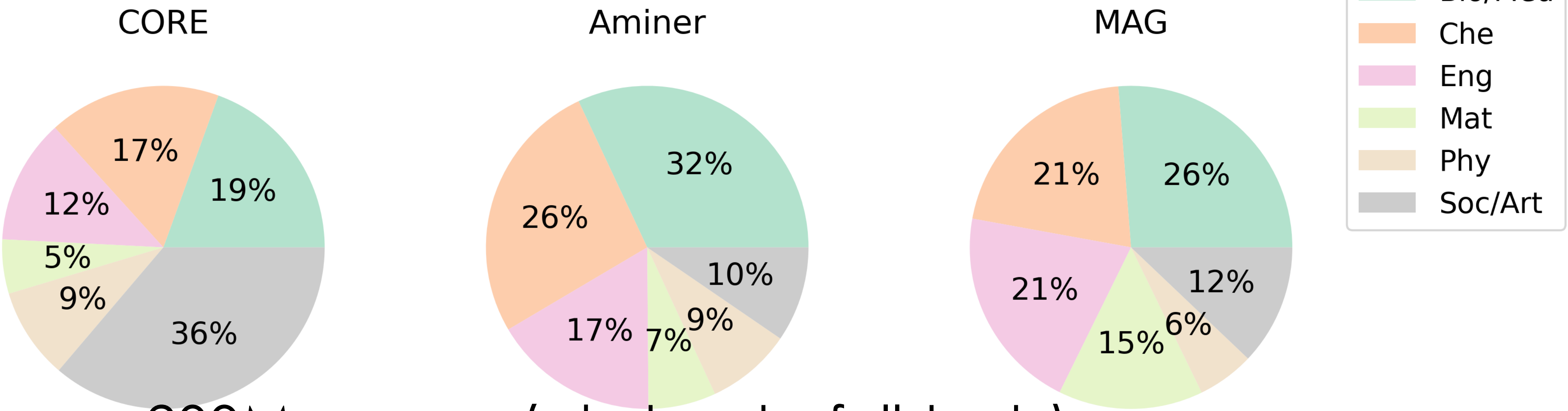
Overview (Hour 1)

- AI For Science
- Deep Learning and Large Language Model training
- Data parallel training
- Model parallel training
  - Sharded Data Parallel (ZeRO and FSDP)
  - Tensor Parallel (TP)
  - Pipeline Parallel (PP)
  - Hybrid Parallelism
- Case Study: Forge

Case Study: Forge

- FORGE: Pre-Training Open Foundation Models for Science
  - Junqi Yin, Sajal Dash, Feiyi Wang, Mallikarjun Shankar
- An open foundation model by AAIMS team trained on scientific research articles to perform science tasks
- Will be presented at SuperComputing Conference, 2023
- Will need volunteers for fine-tuning its “Chat” capability

Scientific Texts



- Data sources: 200M papers (abstracts, full-texts)

Source	#Docs (M)	# Tokens (B)	Size (GB)
CORE	52.3	225	764
Aminer	47.1	10	55
MAG	103.8	20	108
SCOPUS	6.3	1.5	6.5
ArXiv	2.2	0.8	3.4
Total	212	257	937

Domain	#Docs (M)	#Tokens (B)	Size (GB)
Bio/Med	52.5	38	155
Che	43.2	41	158
Eng	35.7	29	113
Mat	27.4	15	67
Phy	14.7	32	108
Soc/Art	36	90	336

FORGE Models

- GPT-NeoX architecture

Model	#Params	d model	n layers	n heads	d head
FORGE-S & FORGE-domain	1.44B	2064	24	24	86
FORGE-M	13B	5120	40	40	128
FORGE-L	25.6B	6144	48	48	128

FORGE	#Params	#Tokens
Biology/Med	1.44B	38B
Chemistry	1.44B	41B
Engineering	1.44B	29B
Material	1.44B	15B
Physics	1.44B	32B
Social/Art	1.44B	90B
All	1.44B/13 B/ 25.6B	257B

J. Yin, S. Dash, F. Wang, and M. Shankar, SC'23

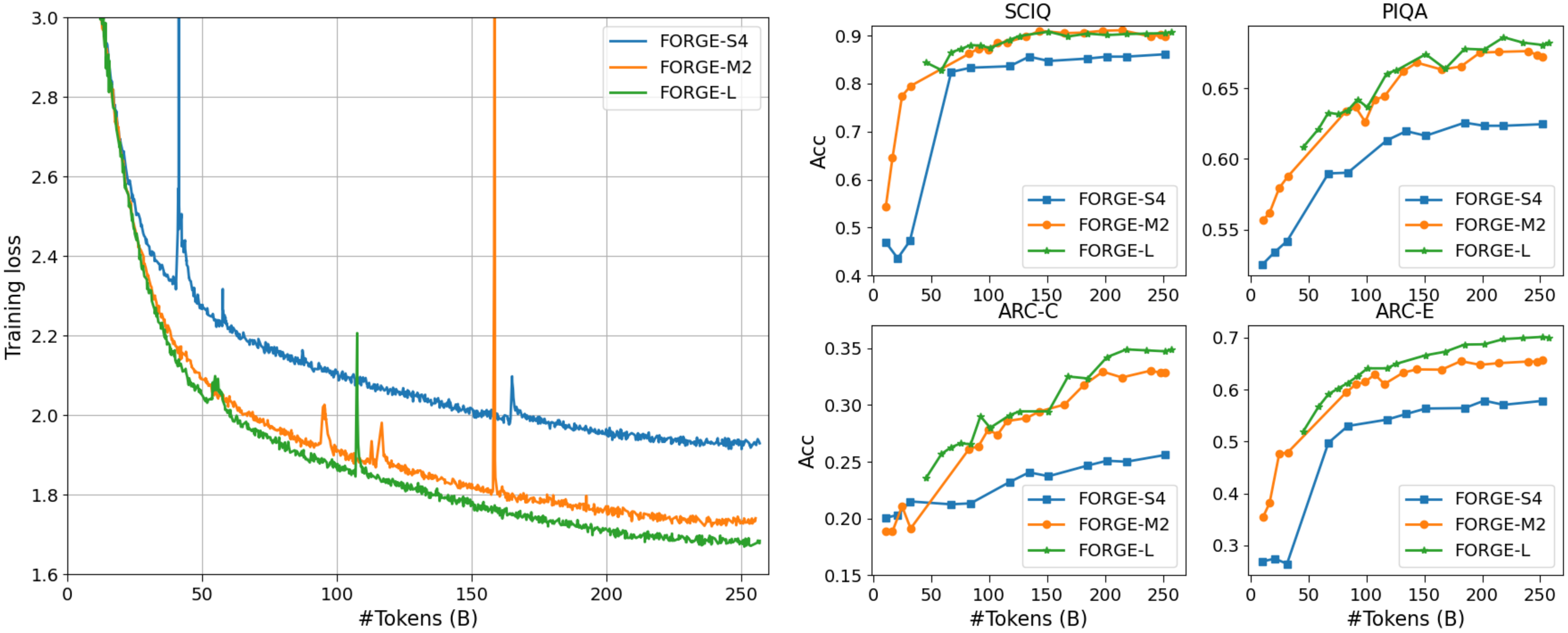
Generic Language Benchmarks

- Lm-eval: zero-shot performance

Model\Test	#Params:#Tokens	ARC-E	ARC-C	HT-CC	HT-CCS	HT-CM	HT-CP	HT-S	OBQA	PIQA	SCIQ
GPT2	124M:1.5B	0.44	0.19	0.25	0.28	0.19	0.23	0.31	0.16	0.63	0.75
GPT2-Large	774M:45B	0.53	0.22	0.28	0.29	0.24	0.26	0.32	0.19	0.70	0.80
GPT2-XL	1.5B:150B	0.58	0.25	0.26	0.26	0.21	0.23	0.31	0.22	**0.71**	0.83
Ref[14]	1.47B:201B	0.55	0.24	0.25	n/a	n/a	n/a	n/a	0.17	n/a	0.84
GPT-NeoX-20B	20B:472B	**0.72**	**0.38**	0.17	0.25	0.24	**0.30**	0.29	**0.29**	**0.77**	**0.93**
FORGE-Bio	1.44B:38B	0.47	0.24	0.24	0.31	0.20	0.22	0.24	0.17	0.61	0.79
FORGE-Che	1.44B:41B	0.53	0.24	0.30	0.29	0.22	0.21	0.29	0.2	0.60	0.82
FORGE-Eng	1.44B:29B	0.45	0.22	0.23	0.31	0.19	0.18	0.31	0.16	0.58	0.80
FORGE-Mat	1.44B:15B	0.47	0.23	0.29	**0.34**	0.19	0.20	0.30	0.17	0.60	0.78
FORGE-Phy	1.44B:32B	0.42	0.21	0.26	0.28	**0.24**	0.21	0.30	0.14	0.55	0.76
FORGE-Soc	1.44B:90B	0.5	0.21	**0.32**	**0.34**	0.22	0.23	**0.34**	0.2	0.61	0.82
FORGE-S4	1.44B:257B	0.58	0.26	**0.36**	**0.39**	**0.25**	0.21	**0.35**	0.2	0.62	0.86
FORGE-M2*	13B:143B	0.64	**0.29**	0.24	0.3	0.21	**0.28**	0.34	0.2	0.67	**0.91**
FORGE-L*	25.6B:125B	**0.65**	**0.29**	0.27	0.27	0.24	0.22	0.28	**0.24**	0.66	0.90

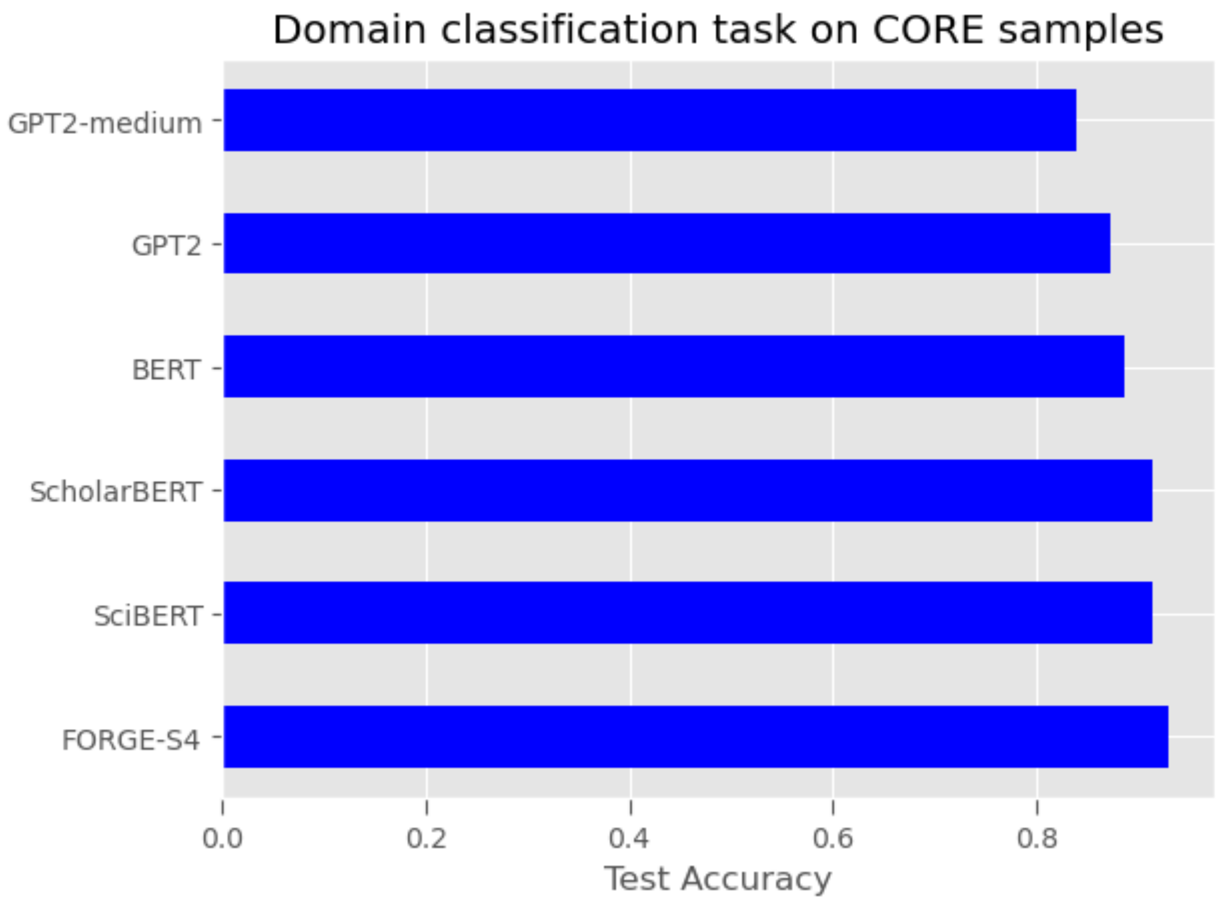
Generic Language Benchmarks

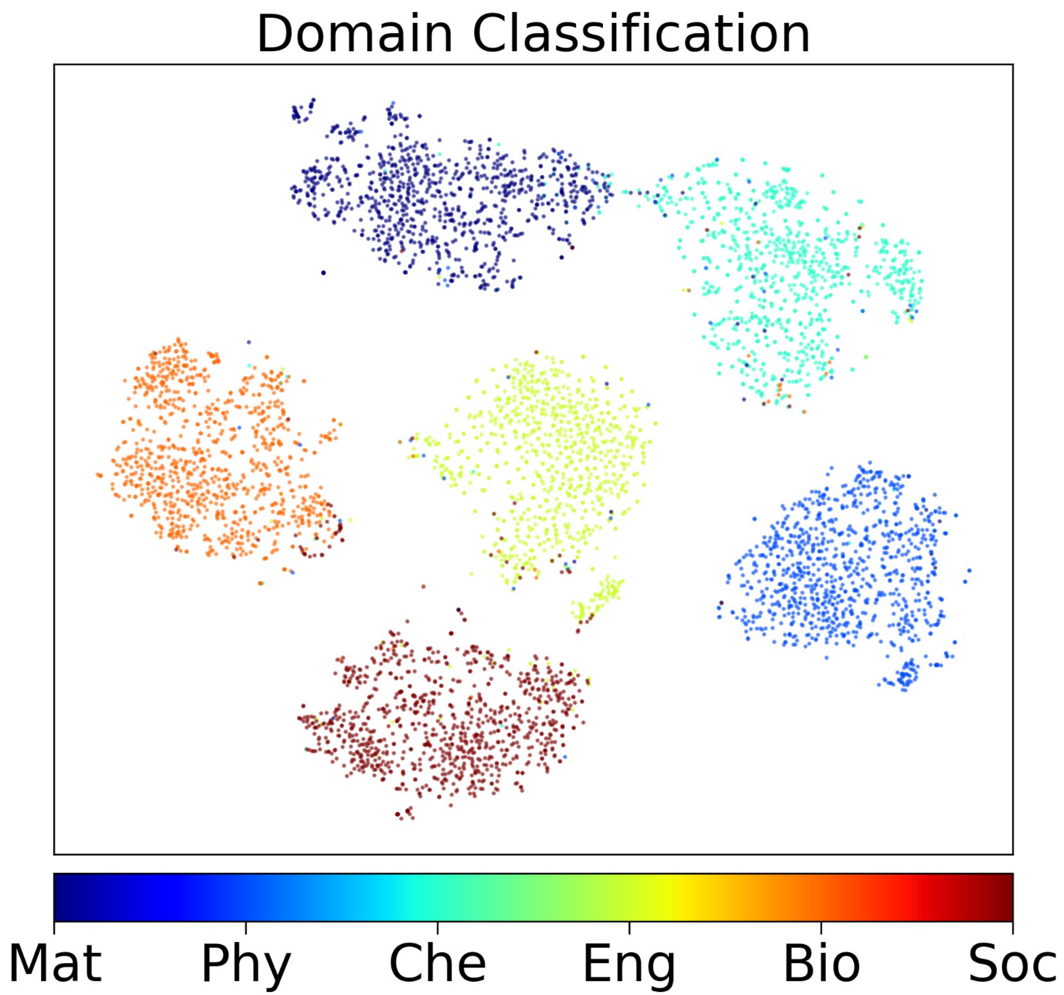
- Lm-eval: loss & accuracy progression



Scientific Downstream Tasks

- Domain subject classification

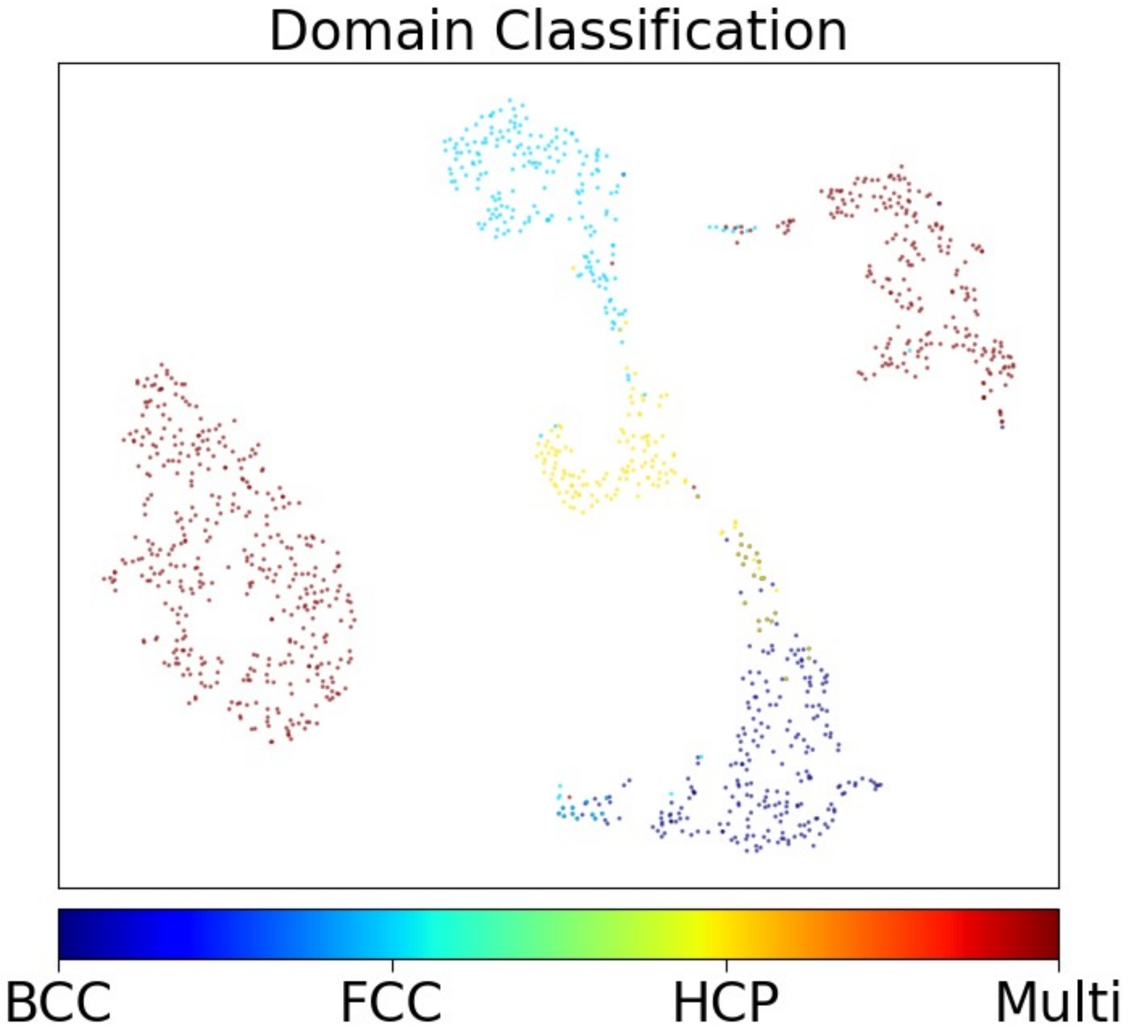




Scientific task: Alloy Phase Prediction

- Alloy phase dataset
  - npj Computational Materials 6, 50 (2020)
- FORGE-mat: LLM embeddings ~ 92%

```
[[-0.0032954  -0.0443229   0.03746693  0.00976305  0.01051364]
 [-0.00595172 -0.04582867  0.02879063  0.0108158  -0.0043186 ]
 [-0.0070161  -0.04830499  0.01686401  0.00728451  0.00023745]
 [-0.00519273 -0.03851099  0.02685266 -0.02516581 -0.00436113]
```



Scientific task: Energy Regression

- A sample DFT dataset on Fe_xPt_y systems (DOI:10.17188/1198813)
  - 10 different concentrations
  - [x, y]: [15, 1], [7, 1], [13, 3], [3, 1], [11, 5], [5, 3], [9, 7], [1, 1],[7, 9], [3, 5]

- Traditional model of effective pair interaction (DOI:10.1016/j.matdes.2019.108247):

$$E \approx N \sum_{j<i,s} V_{ij}^s P_{ij}^s + \text{const},$$

conc	shell1	shell2	shell3	shell4	shell5	shell6	enthalpy
0.625	0.484375	0.5625	0.4375	0.479167	0.5	0.208333	25.33232
0.625	0.5	0.520833	0.458333	0.5	0.375	0.166667	24.55154
0.625	0.484375	0.4375	0.5	0.479167	0.5625	0.166667	25.92364
0.625	0.5	0.458333	0.541667	0.484375	0.5	0.125	25.93012
0.625	0.453125	0.520833	0.5625	0.463542	0.375	0.166667	25.61216

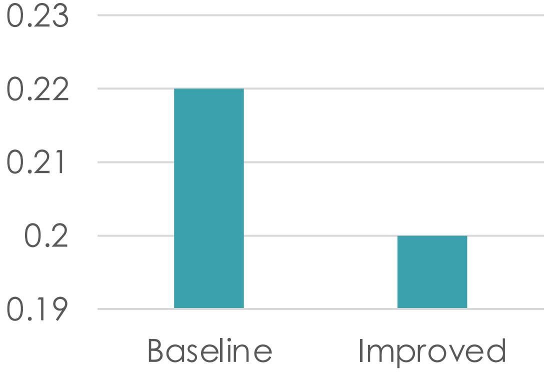
Scientific task: Energy Regression

- LLM embeddings on materials formula
  - Hidden dimension: [10, 2064]
  - Reduced via PCA: [10, 3]

```
[[ 0.31629855, -0.26957738,  0.23415331],
 [ 0.31629948, -0.24336472,  0.13244611],
 [ 0.31616915,  0.24951304,  0.11633574],
```

- Improve the regression with LLM embeddings
  - E ~ f(V^{s}_{ij} , embedding)

~ 9% improvement



Scientific Downstream Tasks

- Phase classification / energy regression

Model\Task	Classification		Regression
	Domain	Phase	
MatSciBERT	**0.91**	**0.88**	0.07
BioGPT	0.90	0.81	0.05
FORGE-Mat	0.90	**0.88**	**0.09**
FORGE-Bio	**0.91**	0.83	0.07
FORGE-Che	0.90	0.83	0.05
FORGE-Phy	0.89	0.86	0.06
FORGE-Eng	0.90	0.81	0.05
FORGE-Soc	**0.91**	**0.88**	0.07

Model\Task	Classification		Regression
	Domain	Phase	
BERT	0.88	0.90	0.05
SciBERT	0.91	0.90	0.05
ScholarBERT	0.91	0.90	0.05
GPT2	0.87	0.90	0.03
GPT2-Medium	0.90	0.85	0.04
GPT2-Large	0.91	0.85	0.06
FORGE-S1	0.92	0.91	0.06
FORGE-S2	0.91	0.89	0.06
FORGE-S3	0.91	0.87	0.05
FORGE-S4	**0.93**	**0.92**	**0.06**
FORGE-M1	0.92	0.90	0.06
FORGE-M2*	0.92	0.91	0.05
FORGE-L*	0.91	0.90	0.03

GPT Assistant training pipeline

Stage	Pretraining	Supervised Finetuning	Reward Modeling	Reinforcement Learning
Dataset	**Raw internet** text trillions of words low-quality, large quantity	**Demonstrations** Ideal Assistant responses, ~10-100K (prompt, response) written by contractors low quantity, high quality	**Comparisons** 100K –1M comparisons written by contractors low quantity, high quality	**Prompts** ~10K-100K prompts written by contractors low quantity, high quality
Algorithm	**Language modeling** predict the next token	**Language modeling** predict the next token	**Binary classification** predict rewards consistent w preferences	**Reinforcement Learning** generate tokens that maximize the reward
		init from	init from	init from SFT use RM
Model	**Base model**	**SFT model**	**RM model**	**RL model**
Notes	1000s of GPUs months of training ex: GPT, LLaMA, PaLM **can deploy this model**	1-100 GPUs days of training ex: Vicuna-13B **can deploy this model**	1-100 GPUs days of training	1-100 GPUs days of training ex: ChatGPT, Claude **can deploy this model**

Hands On

- Data and Model Preparation
- DDP Example with gpt-xl (1.5B)
- ZeRO Example with GPT-J (6B)
- “3D”-parallel with Megatron (22B)
  - Tensor Parallel
  - Data Parallel
  - Pipeline Parallel
  - Sharded data parallel (ZeRO-1)

Python Environment

Following lines can be used with all the scripts.

```
source /lustre/orion/world-shared/stf218/sajal/miniconda3/bin/activate

conda activate /lustre/orion/world-shared/stf218/sajal/TORCH2/env-py310-rccl-megatron-new

export LD_PRELOAD="/usr/lib64/libcrypto.so /usr/lib64/libssh.so.4 /usr/lib64/libssl.so.1.1"
```

Download Data and Models

- https://github.com/olcf/ai-training-series/tree/main/ai_at_scale_part_2/ddp_examples
- bash download_oscar.sh
- bash dl_models.sh

Training Progression

DDP (1.5B)
Data Parallelism

→

Sharded Data Parallelism (DeepSpeed ZeRO, FSDP)

→

Megatron-DeepSpeed (22B)
3D Parallelism

↓

Pipeline Parallelism
Tensor Parallelism
Sharded Data Parallelism
Data Parallelism

DDP Example

- https://github.com/olcf/ai-training-series/tree/main/ai_at_scale_part_2/ddp_examples
- sbatch launch_gpt_srun.frontier
- This launches gpt_oscar_srun.py on 2 Frontier nodes (16 MI250X GCDs)

DDP: Initializing DDP

```python
def setup_distributed_env(init_method=None, rank = 0, world_size=16):
    from mpi4py import MPI
    comm = MPI.COMM_WORLD
    world_size = comm.Get_size()
    world_rank = rank = comm.Get_rank()
    backend = None
    os.environ['MASTER_ADDR'] = master_addr
    os.environ['MASTER_PORT'] = master_port
    os.environ['WORLD_SIZE'] = str(world_size)
    os.environ['RANK'] = str(world_rank)
    os.environ['LOCAL_RANK'] = "0"
    torch.distributed.init_process_group(backend,
                                         init_method=init_method,
                                         rank=rank,
                                         world_size=world_size)

    using_mpi = torch.distributed.get_backend() == 'mpi'

setup_distributed_env()
```

Sharded Data Parallelism (ZeRO Optimizers)

- Add ZeRO Optimizer to ds_config.json

```
"zero_optimization": {
    "stage": 2
},
```

- bsub launch_gptJ_srun.frontier

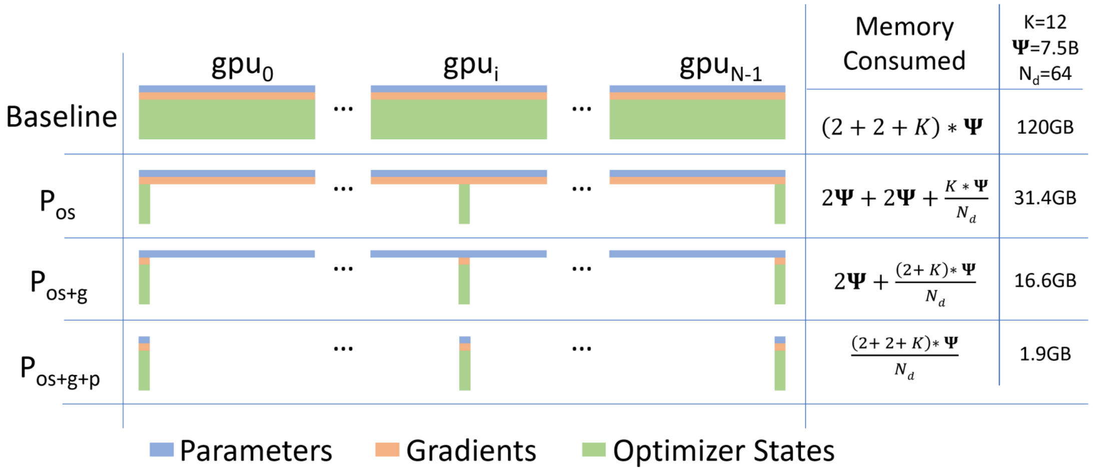


Figure 1: Comparing the per-device memory consumption of model states, with three stages of *ZeRO*-DP optimizations. Ψ denotes model size (number of parameters), K denotes the memory multiplier of optimizer states, and N_d denotes DP degree. In the example, we assume a model size of $\Psi = 7.5B$ and DP of $N_d = 64$ with $K = 12$ based on mixed-precision training with Adam optimizer.

FSDP Example

- Launch launch_gpt_fsdp.frontier
- Wrap the model with FSDP in the python script

```
model = AutoModelForCausalLM.from_pretrained("models/gpt2-xl")

from torch.distributed.fsdp import FullyShardedDataParallel as FSDP

model = FSDP(model)
```

- Specify “input_id” field of the dataset

```
trainer = Trainer(
    model=model,
    args=training_args,
    data_collator=data_collator,
    train_dataset=tokenized_dataset["input_ids"]
```

3D Parallelism

- sbatch launch_gpt22b_srun.frontier
- Enable 3D parallelism and change model-size in the launch script (setting to 1 disables parallelism in that dimension)

```
export GPT_ARGS="--tensor-model-parallel-size 4 \
                --pipeline-model-parallel-size 8 \
                --num-layers 48 \
                --hidden-size 6144 \
```