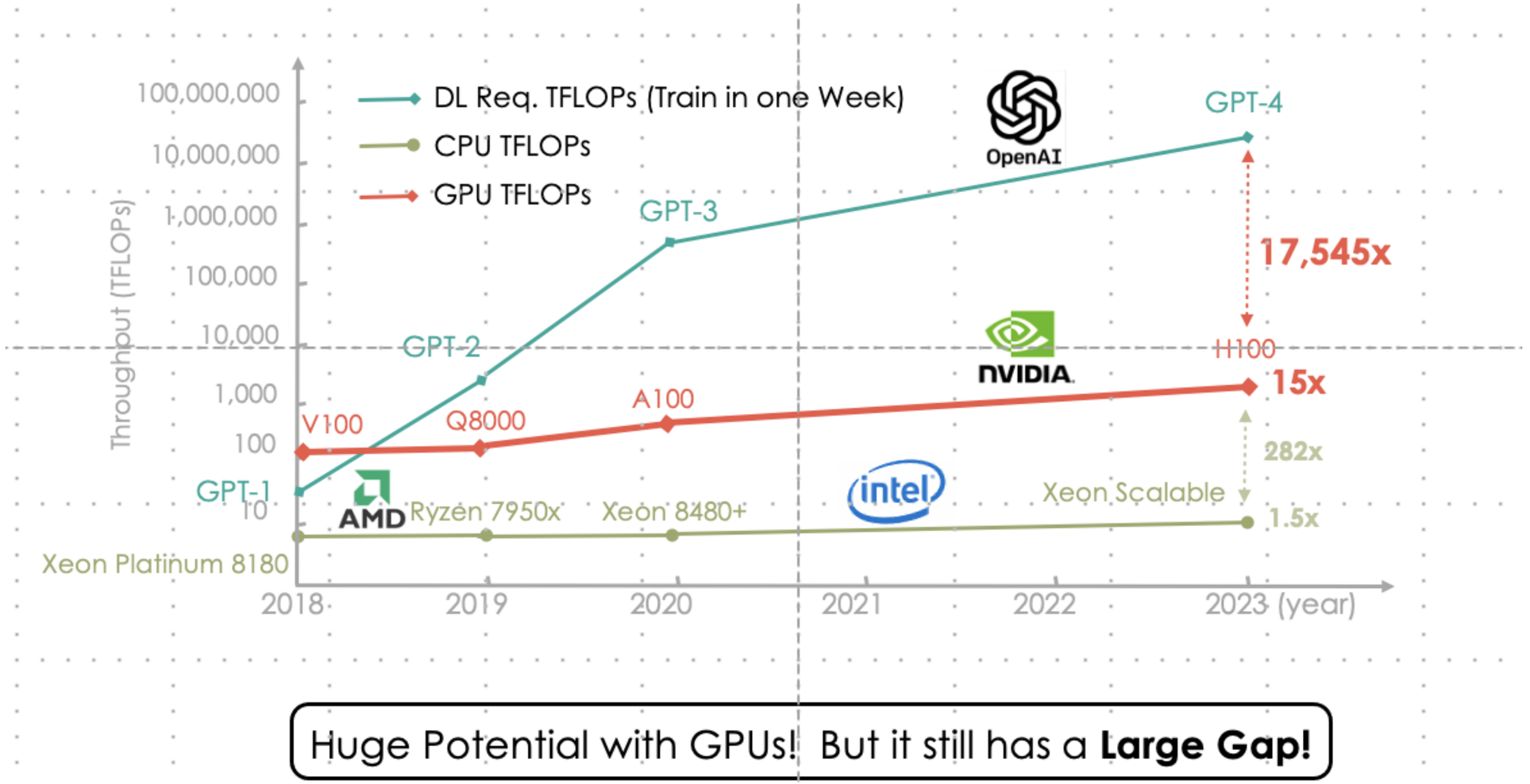
RICE UNIVERSITY

Week-8: Distributed Training

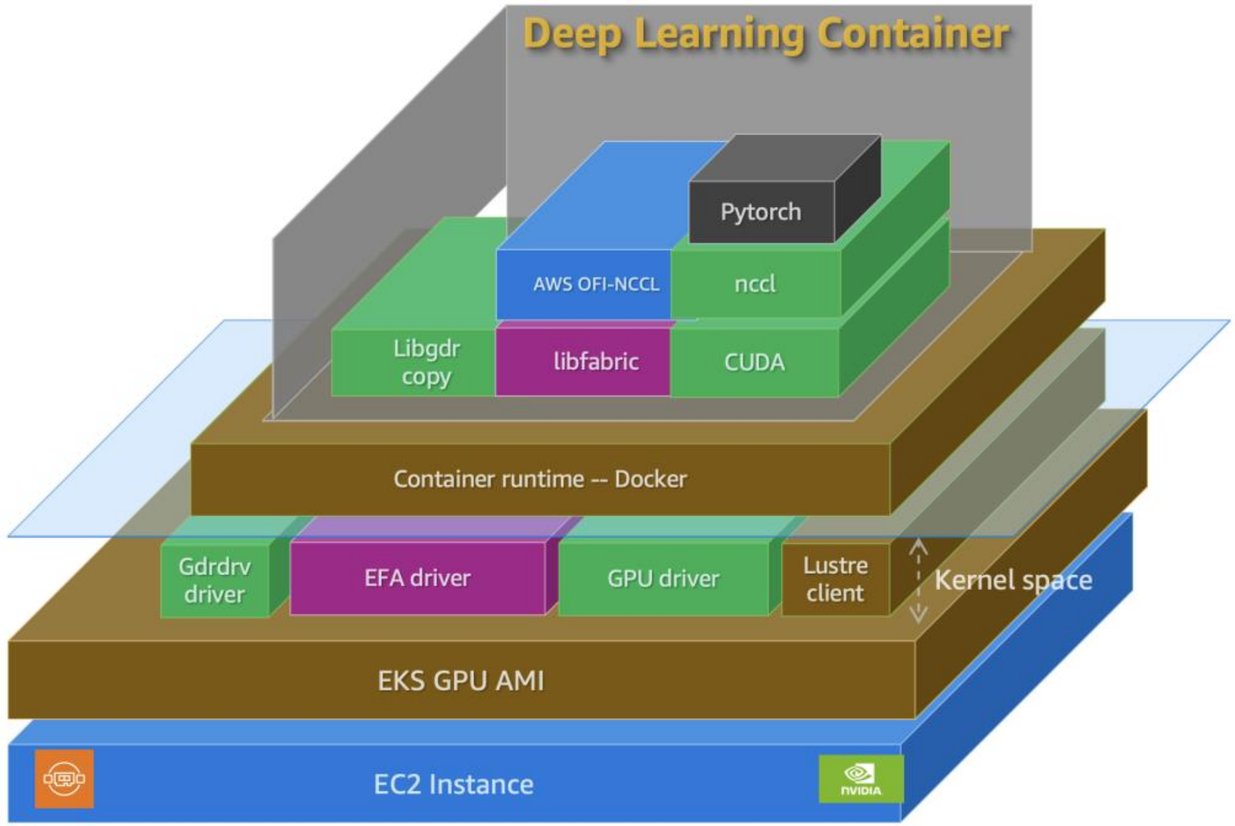
Yuke Wang

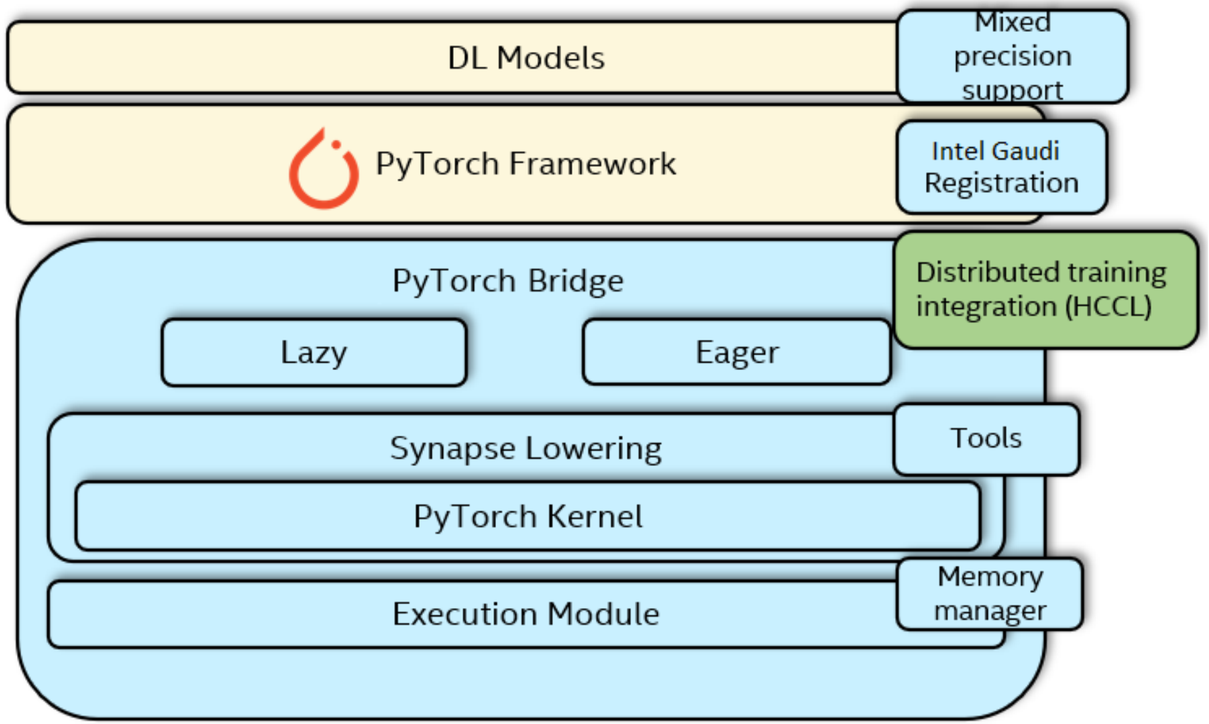
Rice University

Why Distributed Training

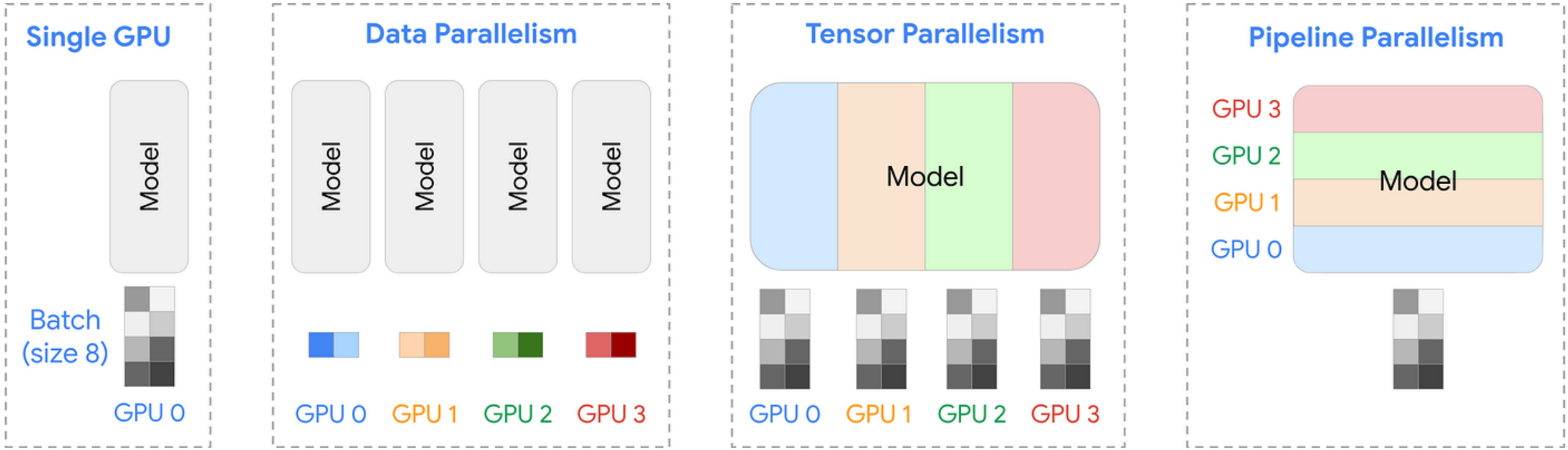


Overview of Distributed Training Frameworks

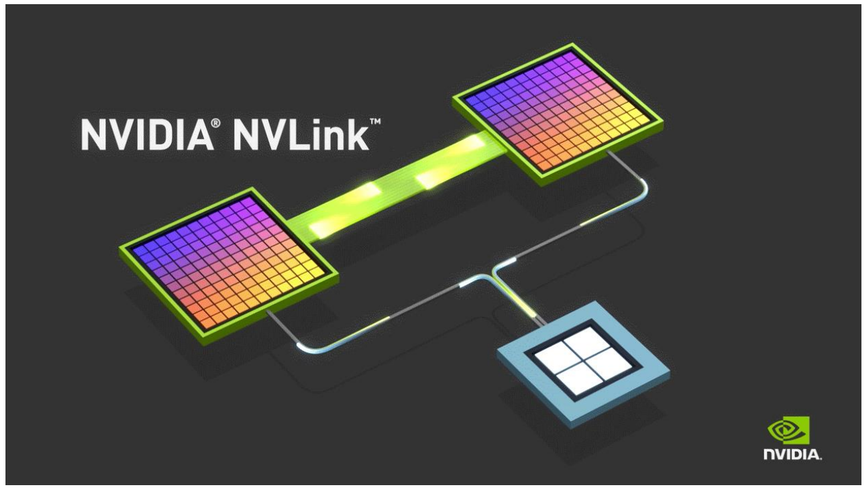


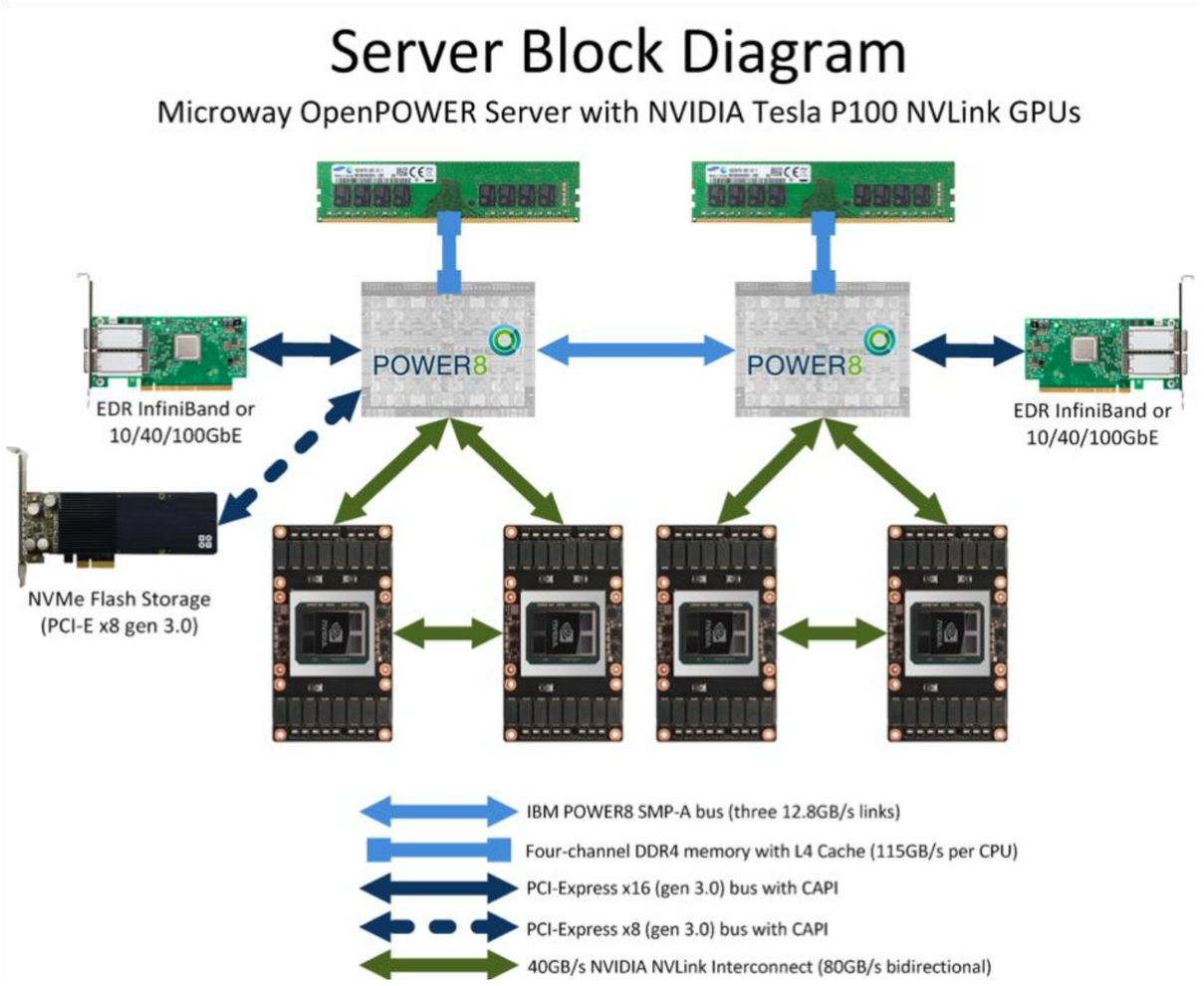


Patterns of Parallelism in Deep Learning

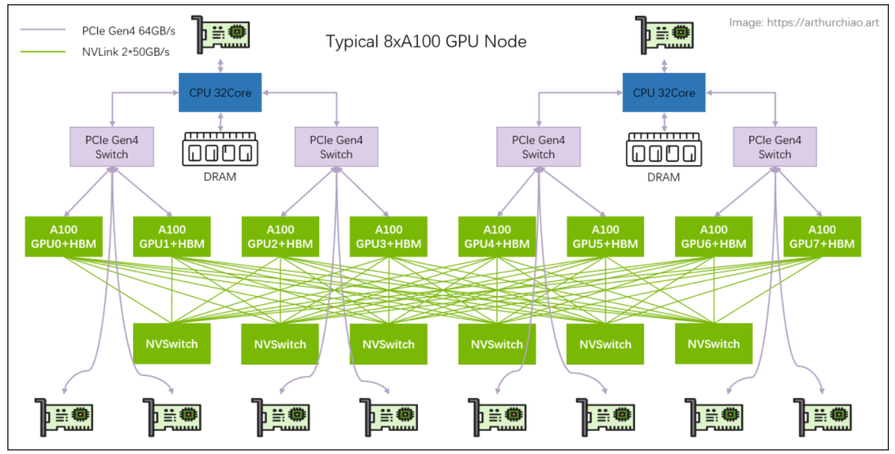


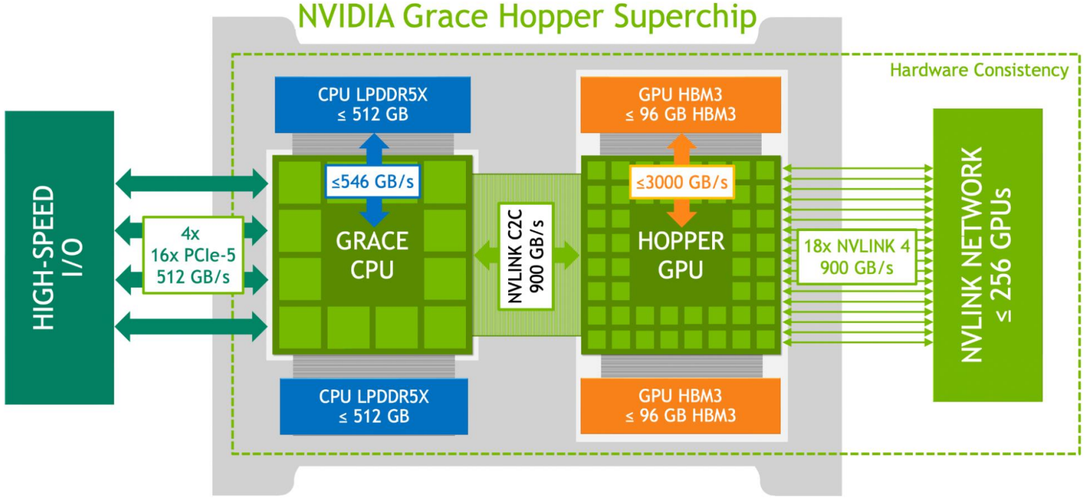
Hardware Interconnects

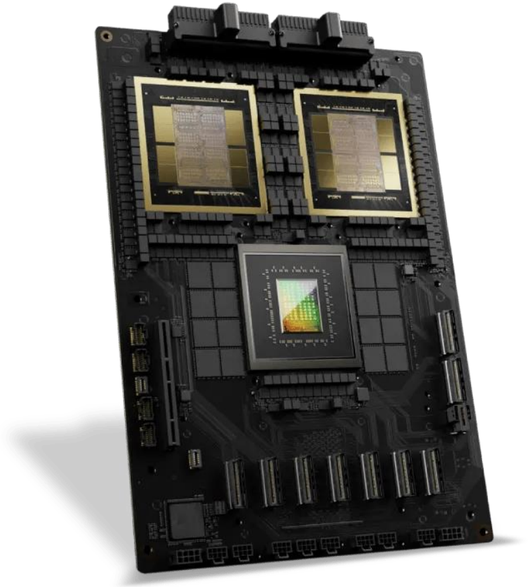




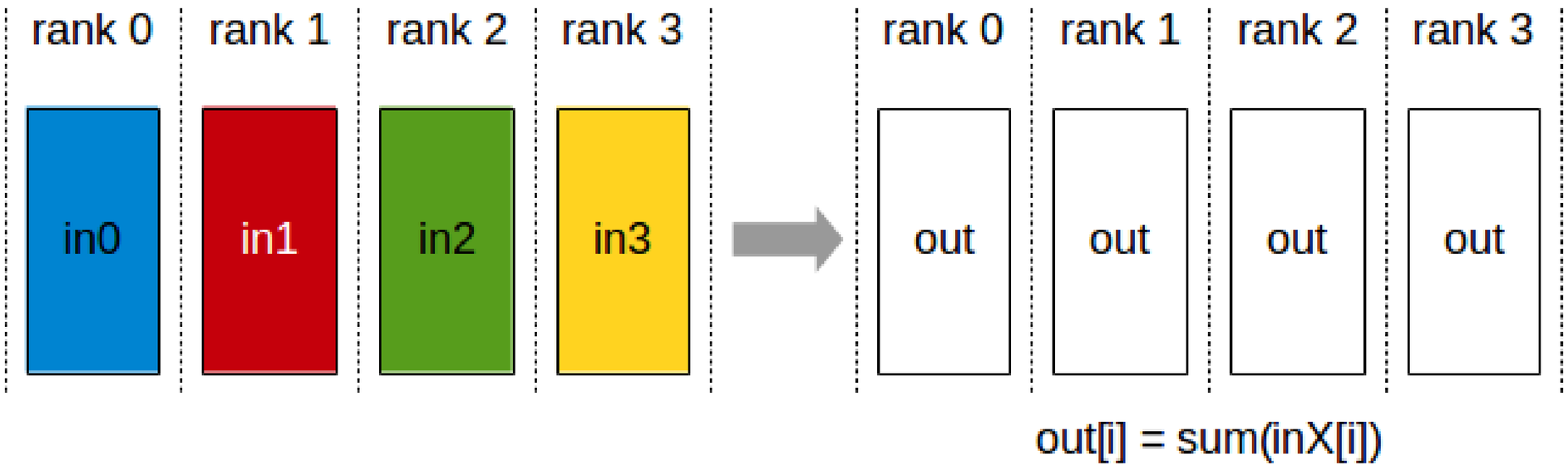
Hardware Interconnects



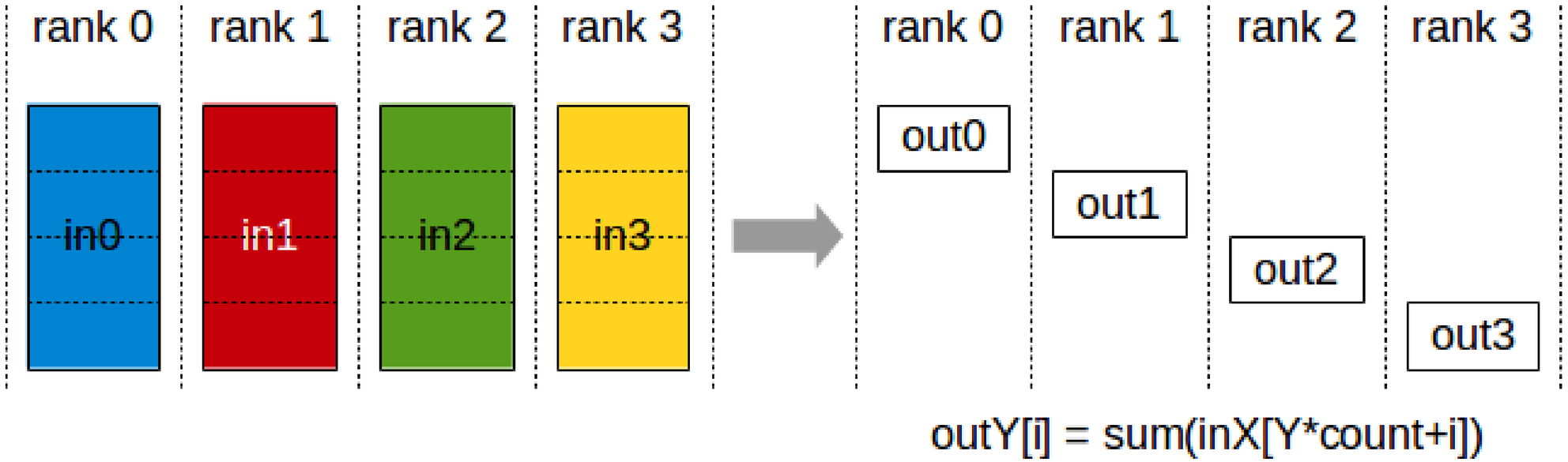




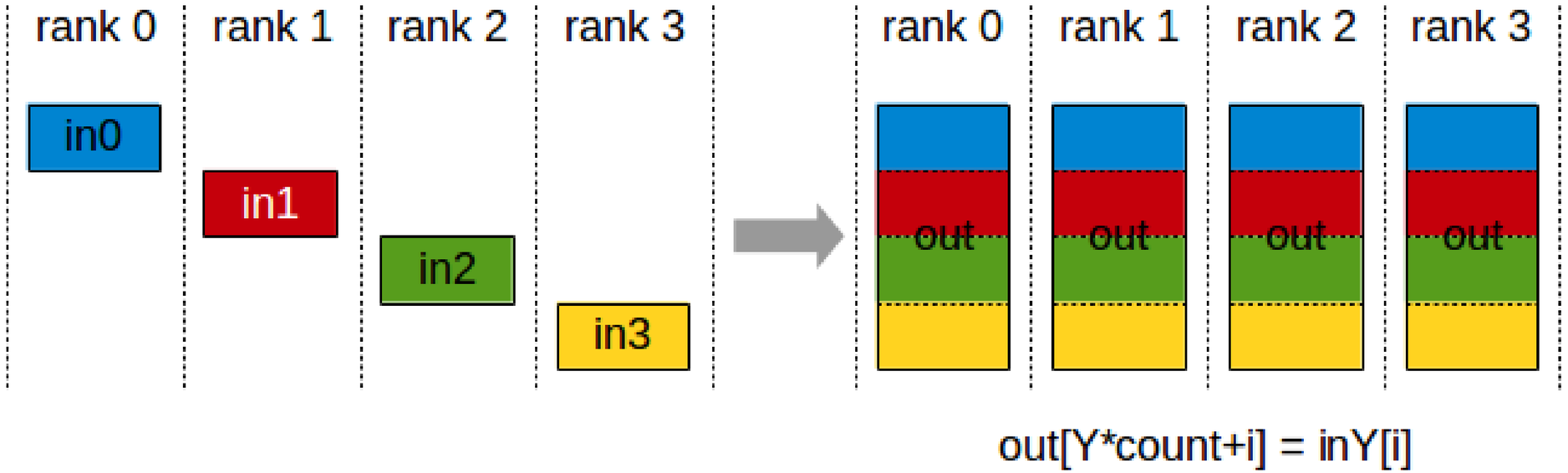
Collective Communication



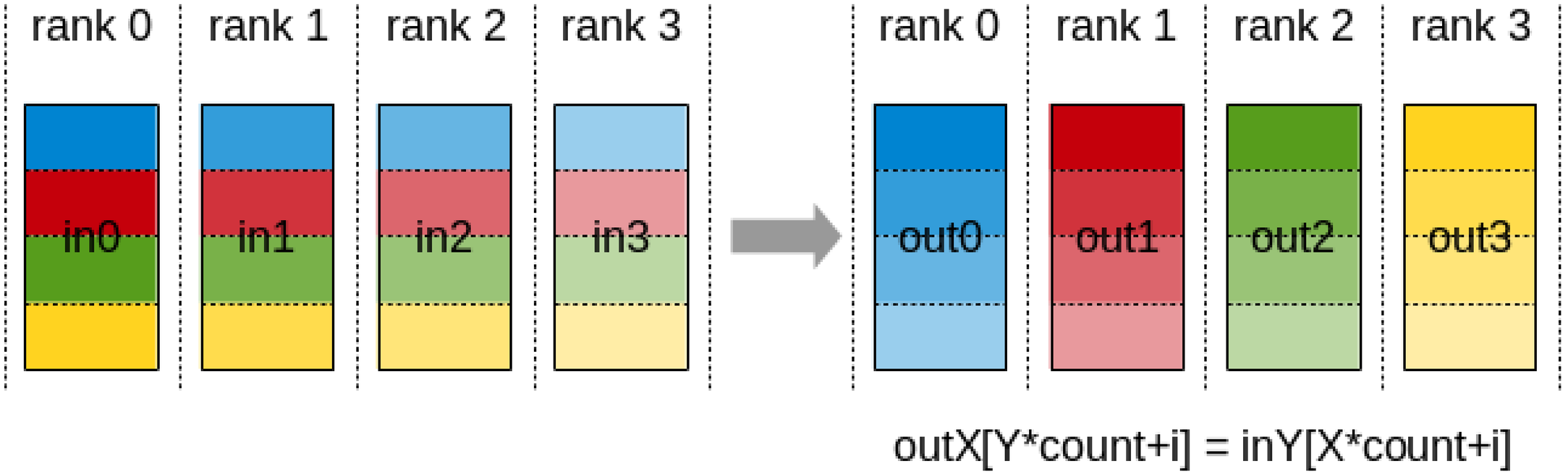
Reduced-Scatter



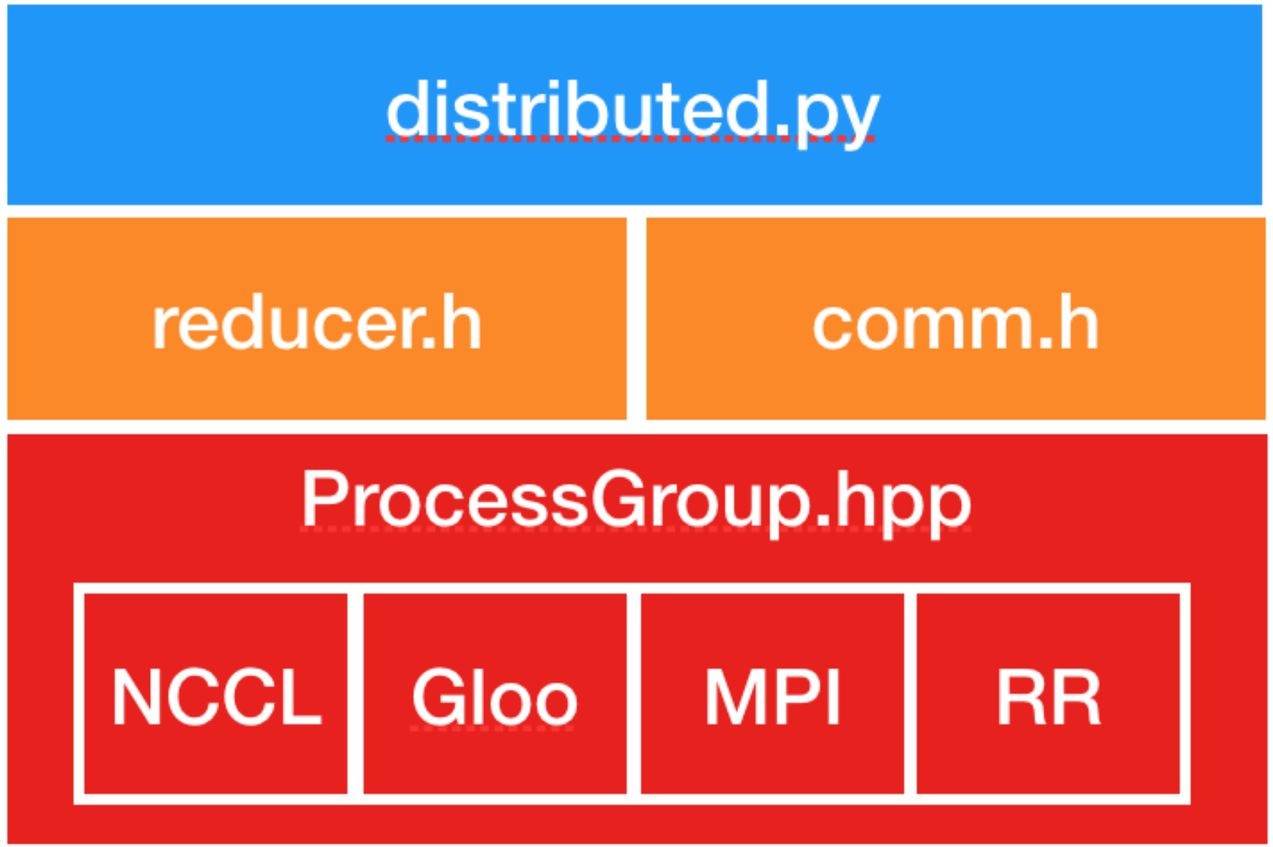
All-Gather


rank 0
rank 1
rank 2
rank 3
in0
in1
in2
in3
rank 0
rank 1
rank 2
rank 3
out
out
out
out
out[Y*count+i] = inY[i]

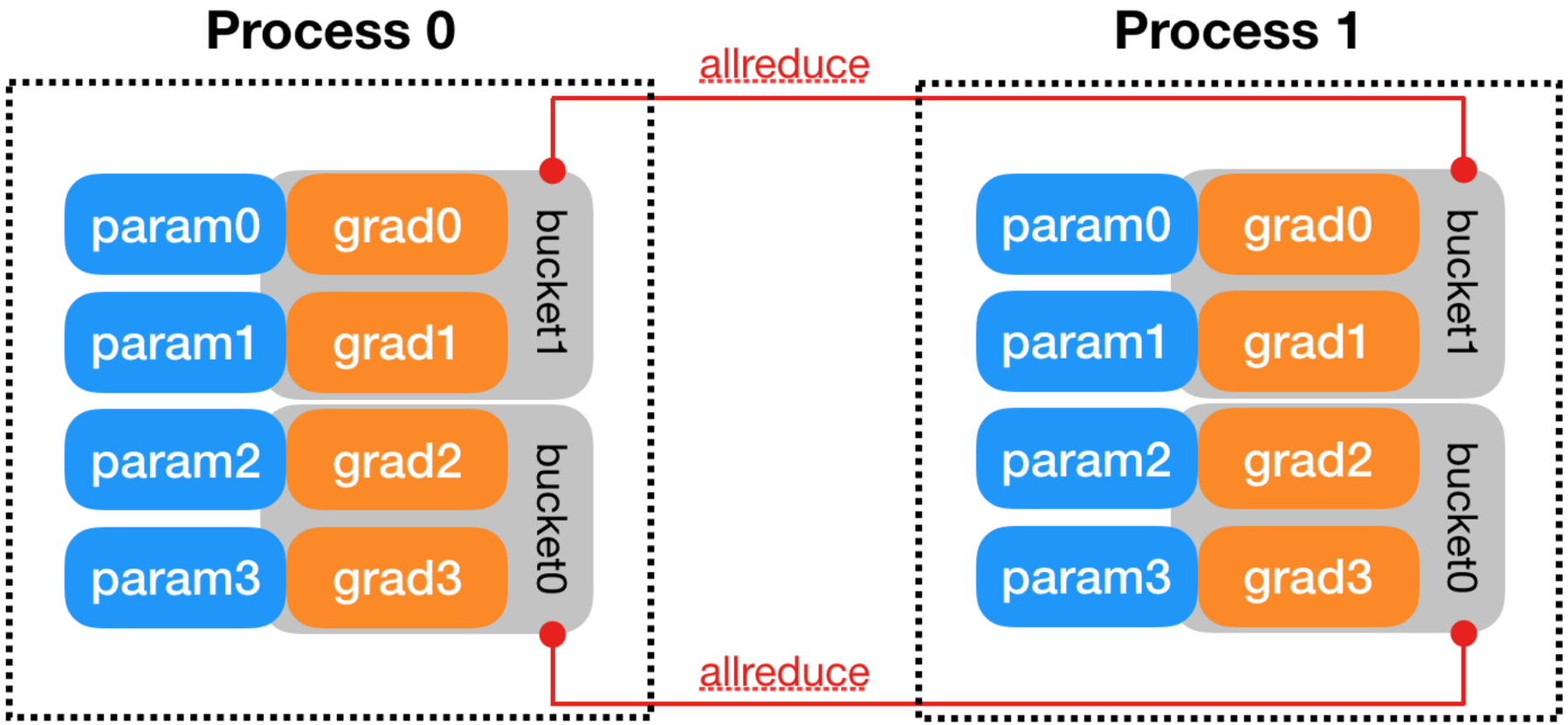
All-to-All


rank 0
rank 1
rank 2
rank 3
in0
in1
in2
in3
rank 0
rank 1
rank 2
rank 3
out0
out1
out2
out3
outX[Y*count+i] = inY[X*count+i]

Pytorch DDP

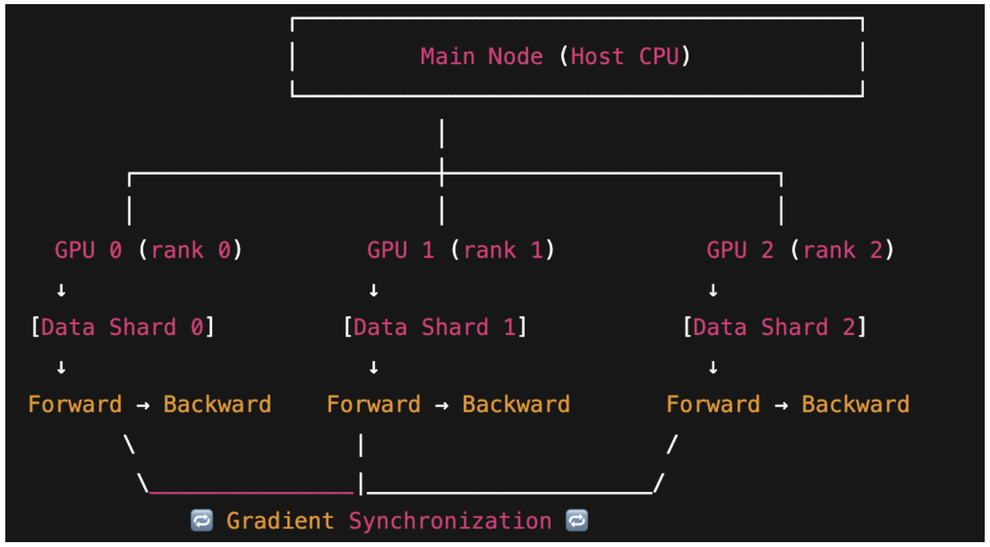

distributed.py
reducer.h
comm.h
ProcessGroup.hpp
NCCL
Gloo
MPI
RR

Pytorch DDP


Process 0
Process 1
allreduce
param0
grad0
param1
grad1
param2
grad2
param3
grad3
bucket1
bucket0
allreduce

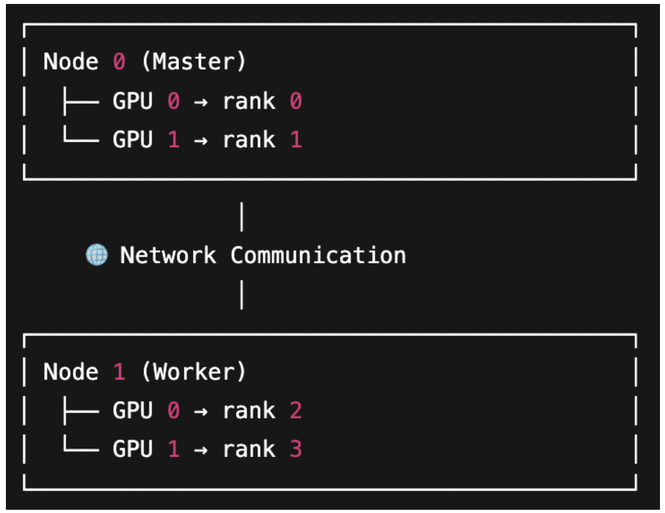
Lunch Single—node Multi-GPU training

```
+-----------------------------+
| Machine with 4 GPUs         |
|                             |
|  Process 0 → GPU 0          |
|  Process 1 → GPU 1          |
|  Process 2 → GPU 2          |
|  Process 3 → GPU 3          |
+-----------------------------+
```




```python
# Step 1: Setup distributed environment
init_process_group(backend="nccl", rank=rank, world_size=4)

# Step 2: Create model and move it to this GPU
model = ResNet18().to(rank)
model = DDP(model, device_ids=[rank])

# Step 3: Prepare data for this GPU
sampler = DistributedSampler(dataset, num_replicas=4, rank=rank)
loader = DataLoader(dataset, sampler=sampler, batch_size=64)

# Step 4: Training loop
for data, target in loader:
    data, target = data.to(rank), target.to(rank)
    output = model(data)
    loss = loss_fn(output, target)
    loss.backward()      # gradients synced automatically
    optimizer.step()
    optimizer.zero_grad()
```

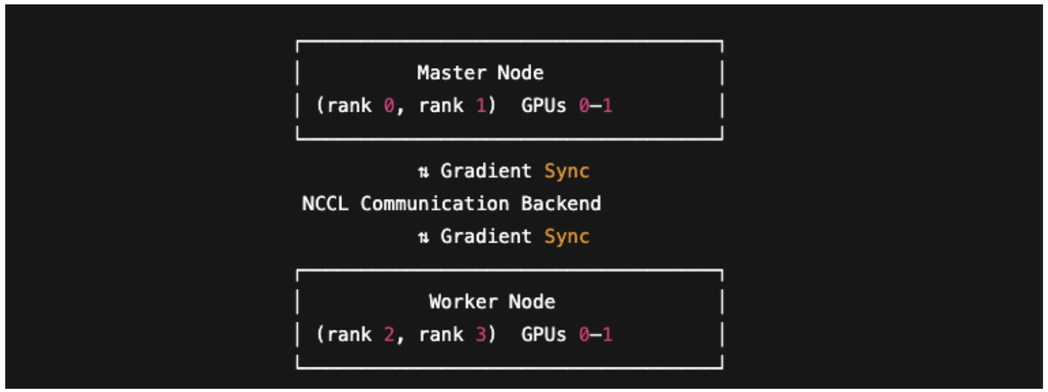
Lunch Multi—node Multi-GPU training




```python
# multi_node_ddp_example.py
import os
import torch
import torch.distributed as dist
from torch.nn.parallel import DistributedDataParallel as DDP
from torch.utils.data import DataLoader, DistributedSampler
from torchvision import datasets, transforms, models
import torch.multiprocessing as mp
import torch.nn as nn
import torch.optim as optim

def setup(rank, world_size, master_addr, master_port):
    os.environ["MASTER_ADDR"] = master_addr
    os.environ["MASTER_PORT"] = master_port
    dist.init_process_group("nccl", rank=rank, world_size=world_size)
    torch.cuda.set_device(rank % torch.cuda.device_count())

def cleanup():
    dist.destroy_process_group()
```

```python
def train(rank, world_size, master_addr, master_port):
    setup(rank, world_size, master_addr, master_port)

    model = models.resnet18(num_classes=10).to(rank % torch.cuda.device_count())
    model = DDP(model, device_ids=[rank % torch.cuda.device_count()])

    transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.5,), (0.5,))
    ])
    dataset = datasets.CIFAR10(root="./data", train=True, download=True, transform=transf
    sampler = DistributedSampler(dataset, num_replicas=world_size, rank=rank)
    dataloader = DataLoader(dataset, batch_size=64, sampler=sampler)

    optimizer = optim.Adam(model.parameters(), lr=0.001)
    criterion = nn.CrossEntropyLoss().to(rank % torch.cuda.device_count())

    for epoch in range(2):
        sampler.set_epoch(epoch)
        for batch, (data, target) in enumerate(dataloader):
            data, target = data.to(rank % torch.cuda.device_count()), target.to(rank % to
            optimizer.zero_grad()
            output = model(data)
            loss = criterion(output, target)
            loss.backward()
            optimizer.step()

            if rank == 0 and batch % 100 == 0:
                print(f"[Epoch {epoch}] [Batch {batch}] Loss = {loss.item():.4f}")

    cleanup()

def main():
    world_size = 4  # 2 nodes × 2 GPUs each
    master_addr = "192.168.0.1"  # IP address of node 0
    master_port = "12355"
    mp.spawn(train, args=(world_size, master_addr, master_port), nprocs=2)

if __name__ == "__main__":
    main()
```

Lunch Multi—node Multi-GPU training



Node 0 (master)

bash — Copy code

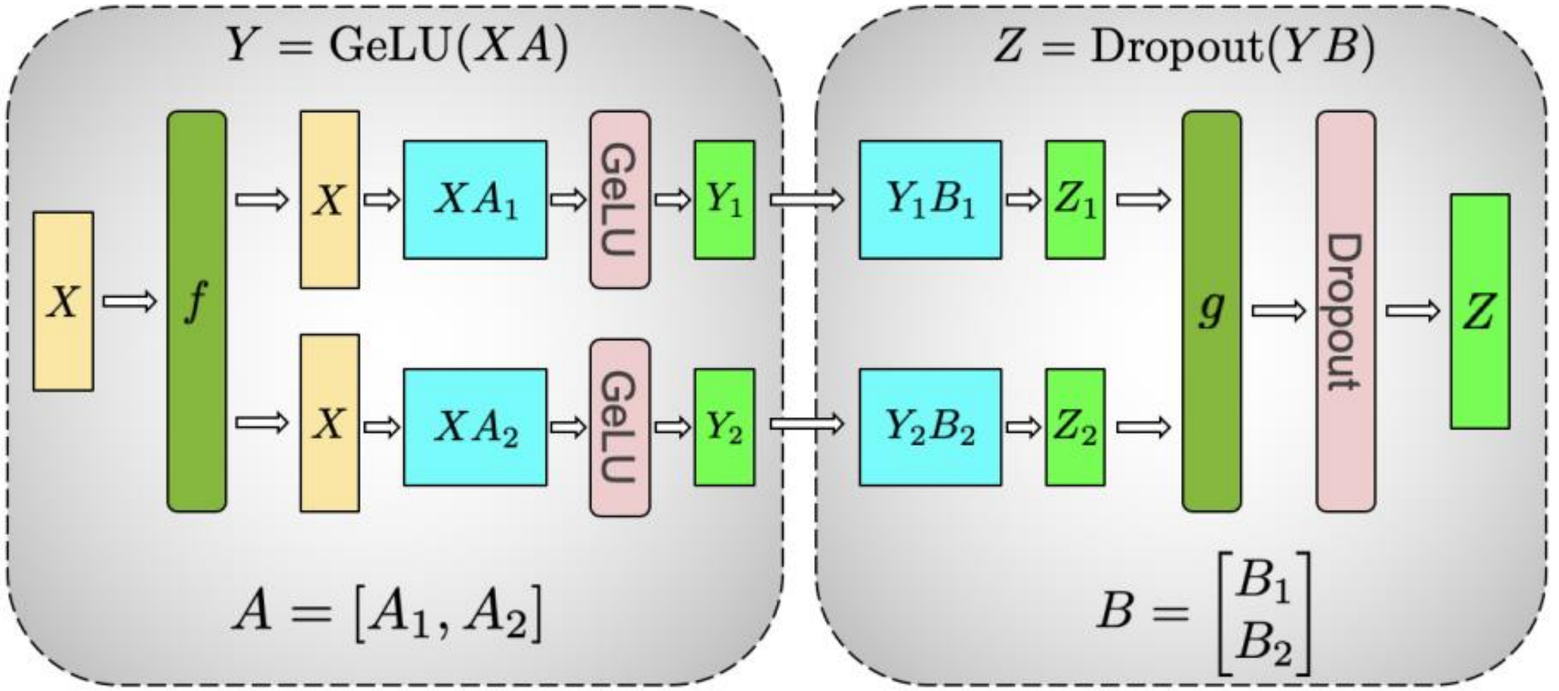
```
python -m torch.distributed.run \
  --nproc_per_node=2 \
  --nnodes=2 \
  --node_rank=0 \
  --master_addr="192.168.0.1" \
  --master_port=12355 \
  multi_node_ddp_example.py
```

Node 1 (worker)

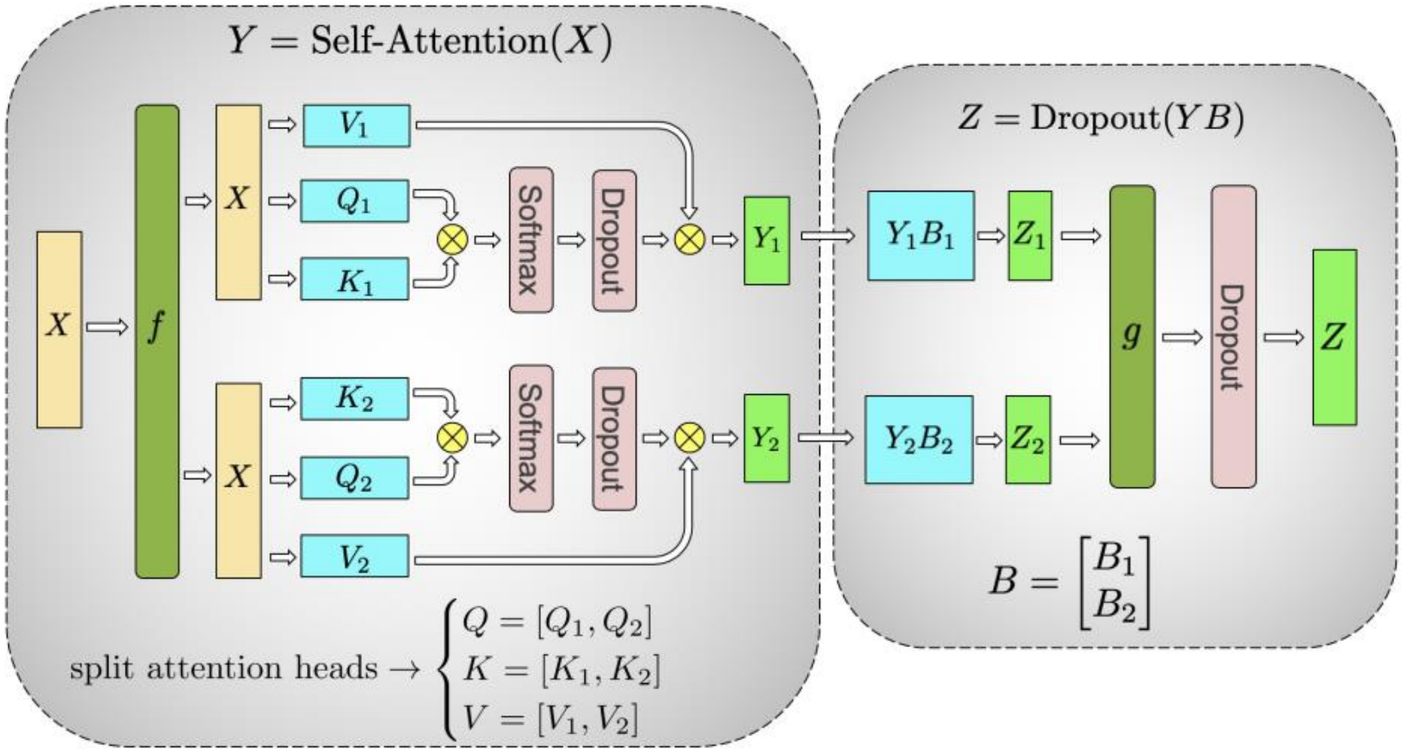
bash — Copy code

```
python -m torch.distributed.run \
  --nproc_per_node=2 \
  --nnodes=2 \
  --node_rank=1 \
  --master_addr="192.168.0.1" \
  --master_port=12355 \
  multi_node_ddp_example.py
```

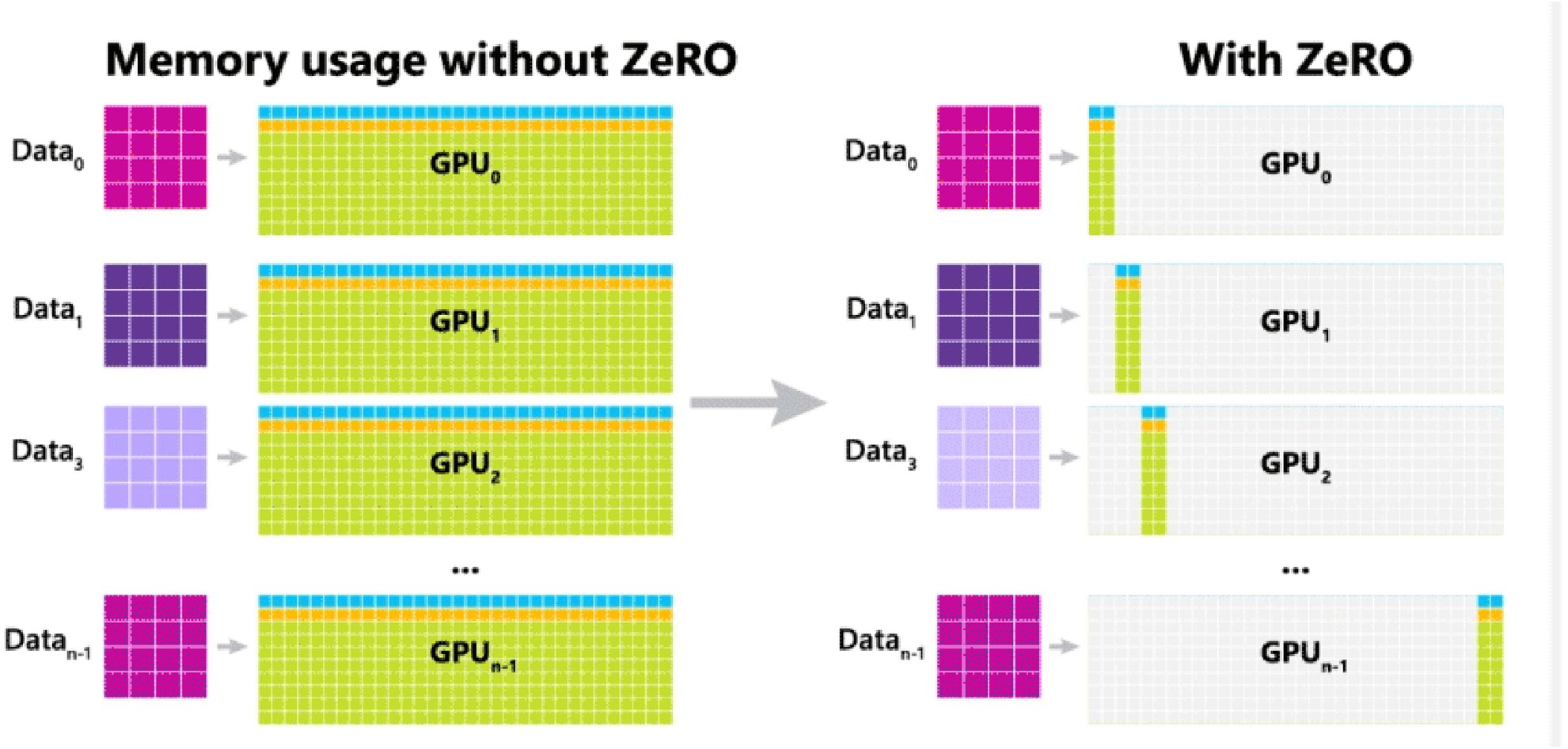
Case Study: Megatron LM



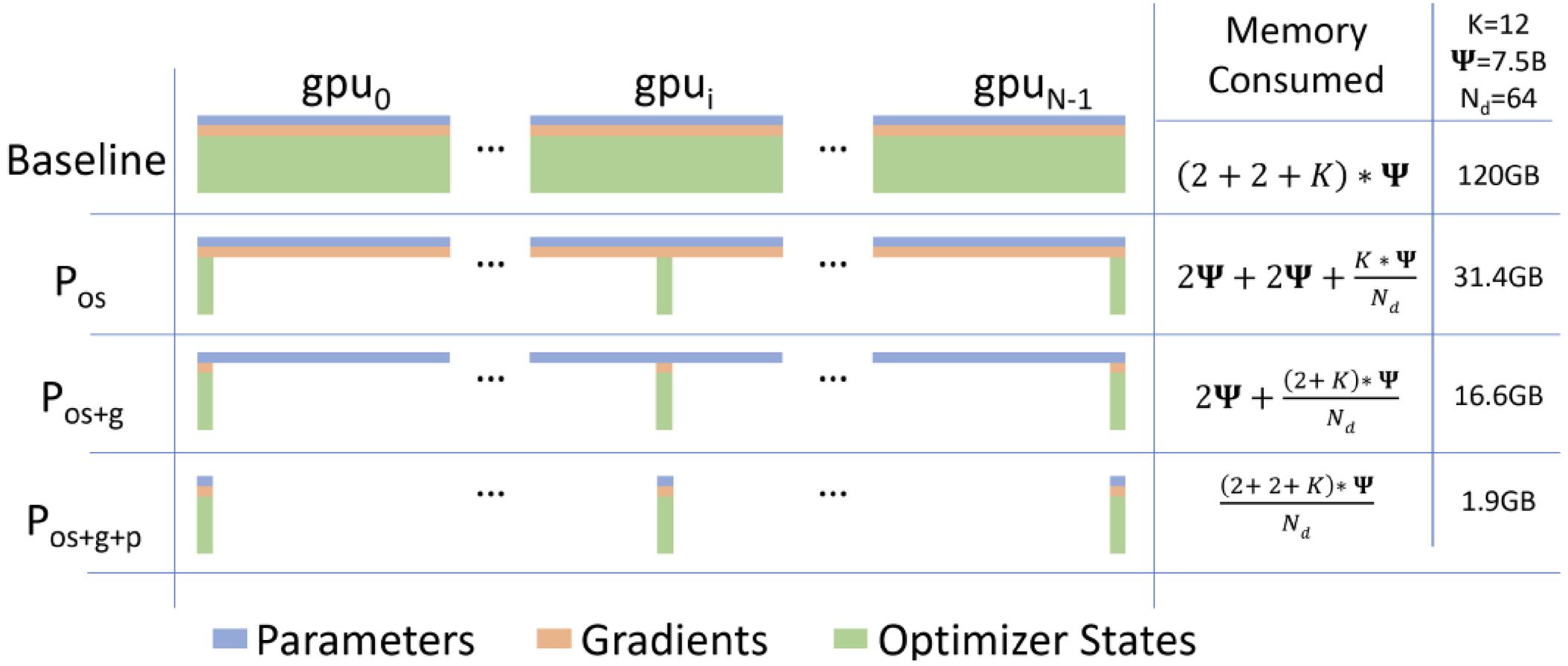
Case Study: Megatron LM



Case Study: ZeRO/FSDP



Case Study: ZeRO/FSDP



RICE UNIVERSITY

yuke.wang@rice.edu