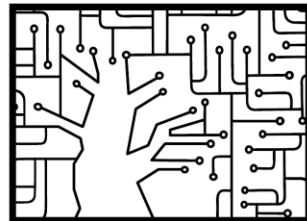


Training with Multiple GPUs

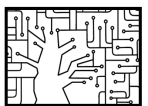
Shai Bagon



WAIC

Why?

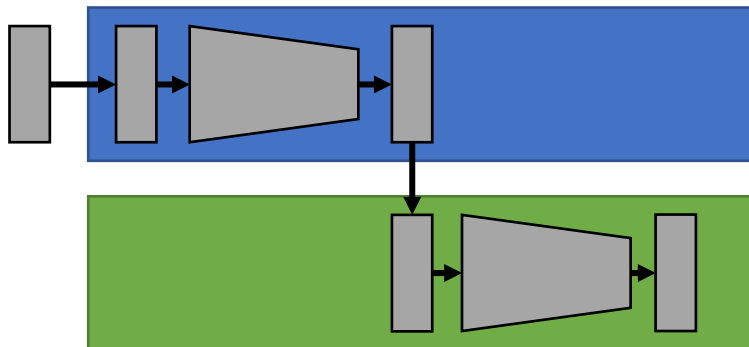
- More compute power
- More GPU memory



How to parallel?

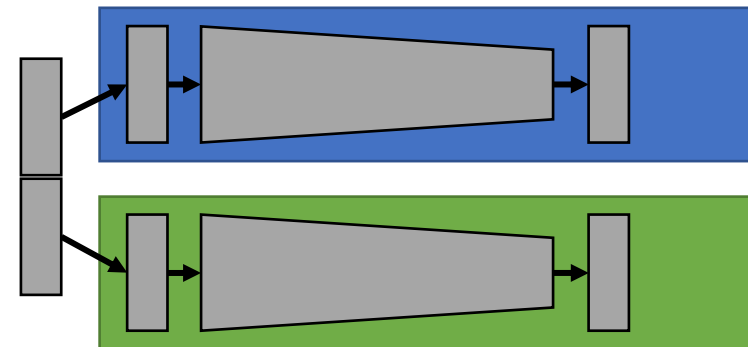
Model parallel

- Different parts of the **model** on parallel GPUs
- When GPU mem is the issue



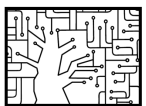
Data parallel

- Different parts of the **data** on parallel GPUs
- When GPU compute is the issue



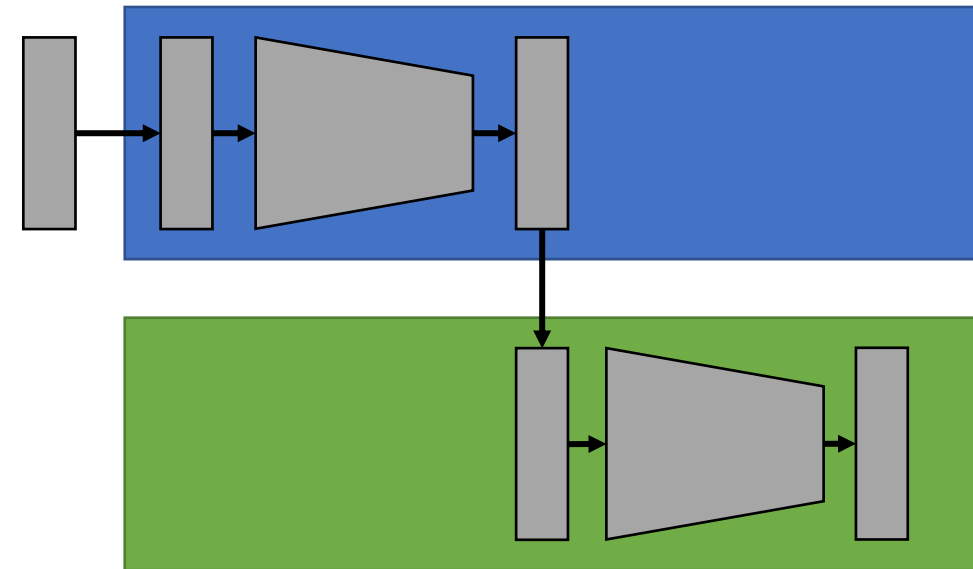
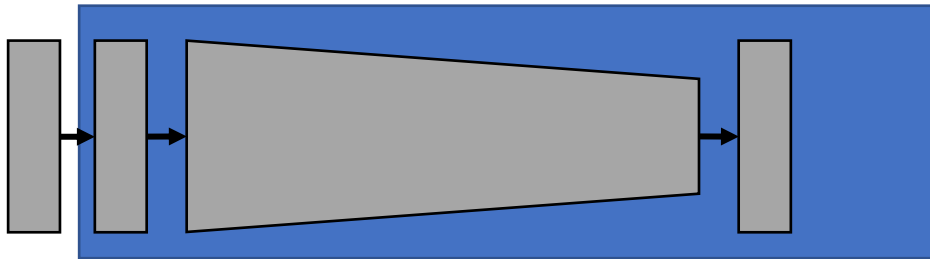
Model Parallel

Bottleneck is GPU mem



Model Parallel

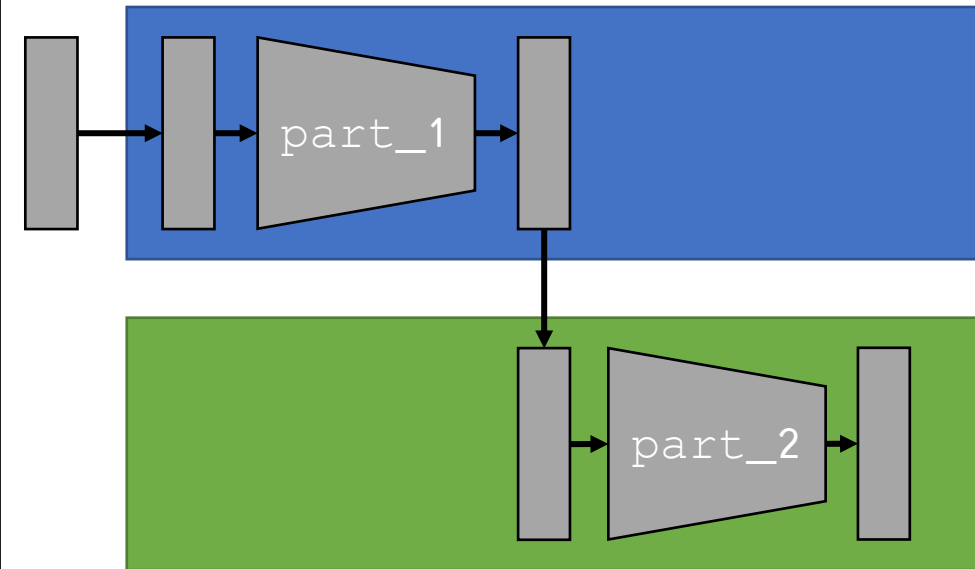
GPU memory is not enough
Split processing between GPUs



Model Parallel

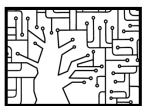
```
class ModelParallel(nn.Module):
    def __init__(self, ...):
        super(ModelParallel, self).__init__()
        self.part_1 = nn.Sequential(
            ...
        )
        self.part_2 = nn.Sequential(
            ...
        )
        # put each part on a different device
        self.part_1.to(torch.device('cuda:0'))
        self.part_2.to(torch.device('cuda:1'))

    def forward(self, x):
        x = x.to(torch.device('cuda:0'))
        x1 = self.part_1(x)
        # move to second device
        x1 = x1.to(torch.device('cuda:1'))
        y = self.part_2(x1)
        return y
```



Model Parallel

- Takes advantage of GPU mem of several devices
- Tailored specific to each model
- Requires “bookkeeping” of tensors/devices



Gradient Accumulation

Regular training

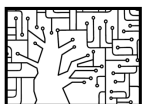
```
for x, y_gt in data:  
    opt.zero_grad()  
    y_pred = model(x)  
    loss = criterion(y, y_gt)  
    loss.backward()  
    opt.step()
```

Gradient accumulation

```
for x, y_gt in data:  
    opt.zero_grad()  
    for sub_x, sub_y_gt in split(x, y_gt):  
        sub_y_pred = model(sub_x)  
        loss = criterion(sub_y_pred, sub_y_gt)  
        loss.backward()  
    opt.step()
```

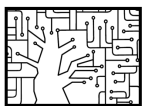
Minimal code change

Serial processing of sub-batches



Data Parallel

Bottleneck is GPU compute



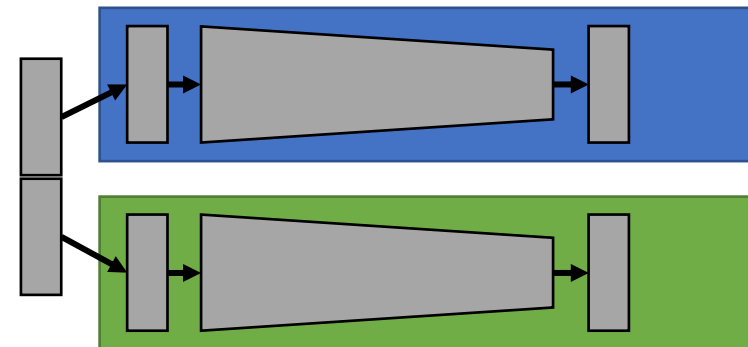
Data Parallel

When?

Process large batches **in parallel**

Key components

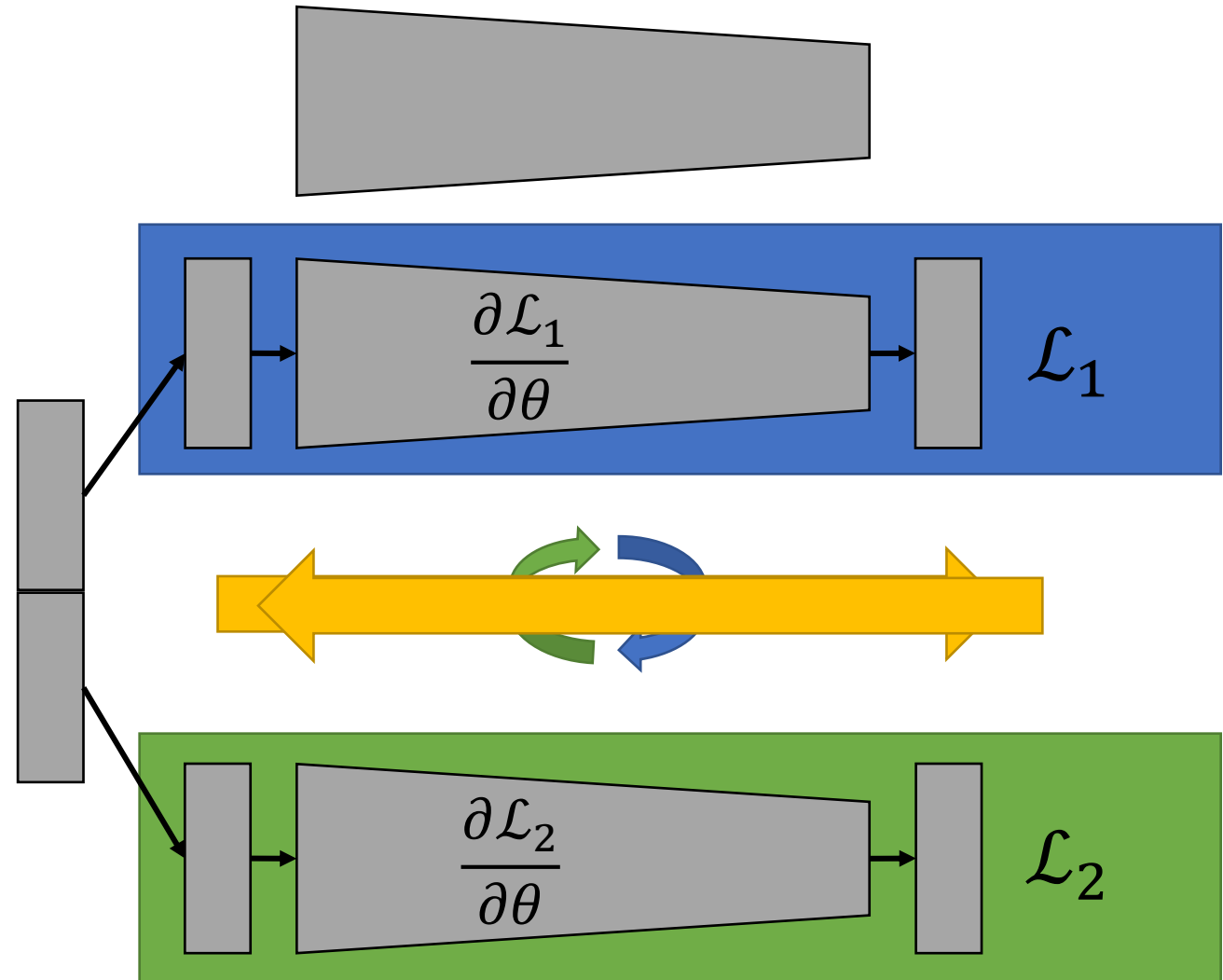
- **Identical** copies of the model
- **Different** sub-batches
- **Synchronized** updates



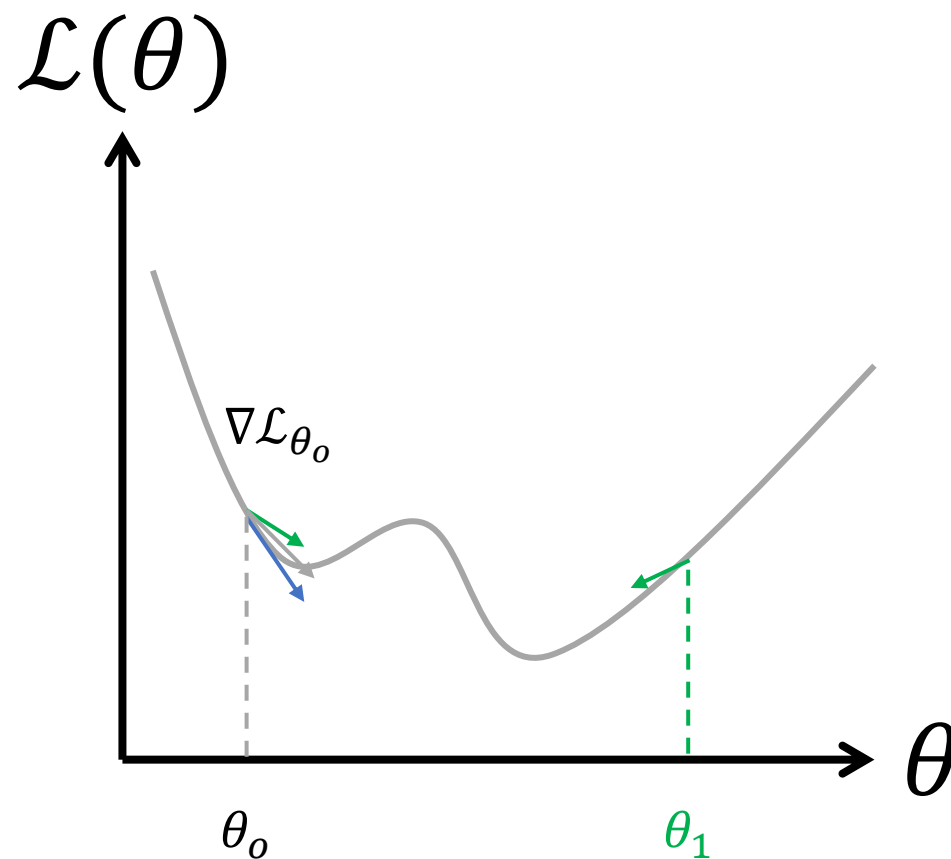
Data Parallel – How does it work?

Key components

- **Identical** copies of the model
- **Different** sub-batches
- **Synchronized** updates



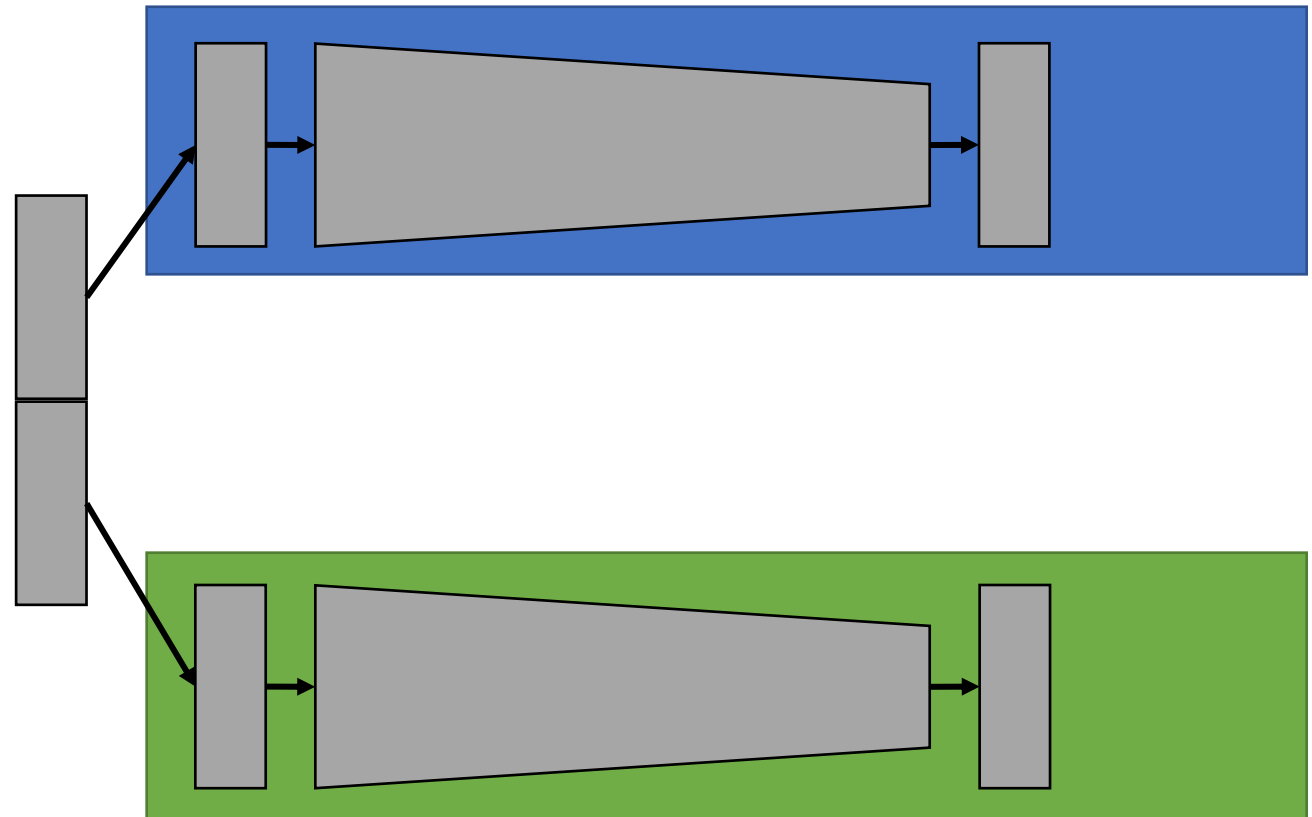
Why Identical?



Data Parallel – How does it work?

Key components

- **Identical** copies of the model
- **Different** sub-batches
- **Synchronized** updates



Data Parallel `nn.DataParallel`

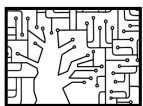
```
model = MyModel()  
# make it parallel  
model = nn.DataParallel(model)
```

At each iteration:

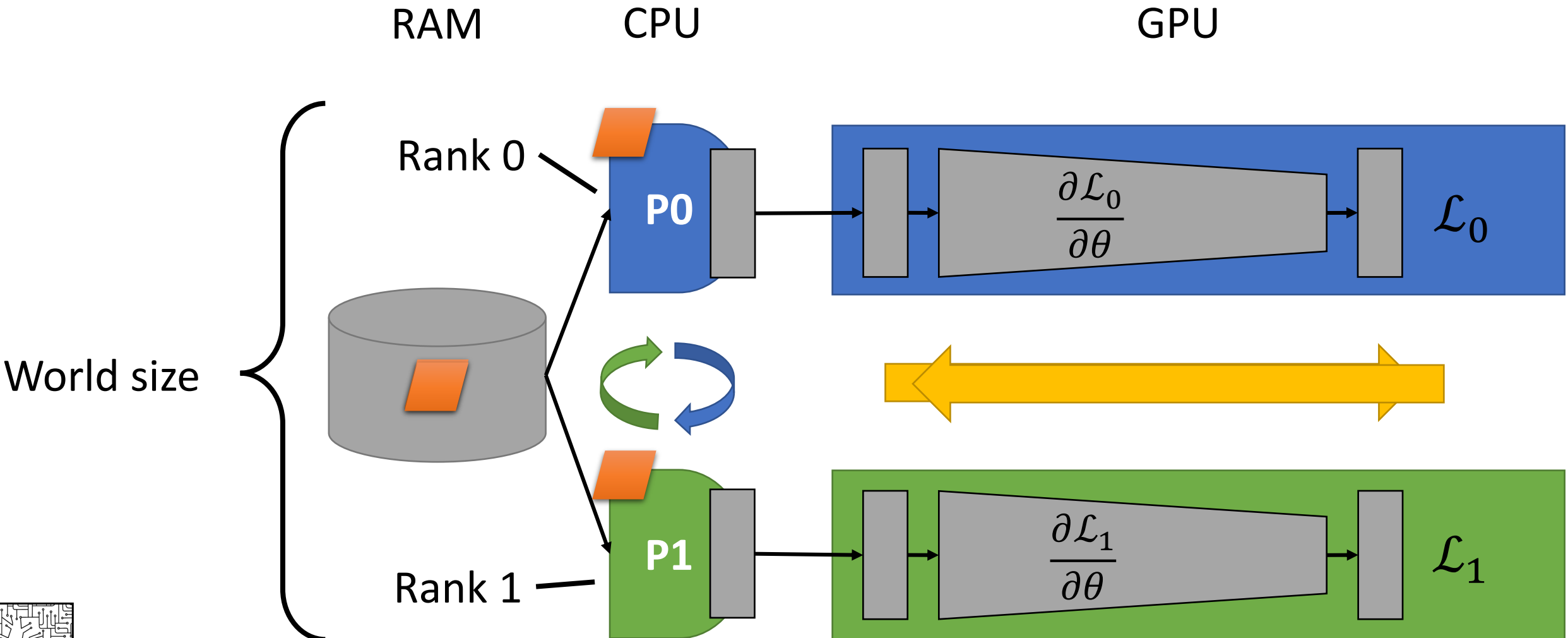
1. Replicates model to all GPUs
2. Splits the batch to the GPUs
3. Sync updates

Handles everything for you

Increased overhead



Data Parallel DistributedDataParallel

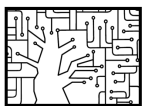


Data Parallel DistributedDataParallel

```
def train(rank, args):
    # init the process in context
    torch.distributed.init_process_group(backend='nccl', init_method='tcp://127.0.0.1:54263', world_size, rank)
    ...
    # wrap the model
    model = nn.parallel.DistributedDataParallel(model, device_ids=[rank])

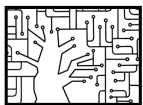
    # use "distributed aware" sampler
    sampler = torch.utils.data.distributed.DistributedSampler(
        train_dataset,
        num_replicas=world_size,
        rank=rank)

    loader = DataLoader(train_dataset, batch_size,
                        shuffle=False, sampler=train_sampler)
```



Data Parallel DistributedDataParallel

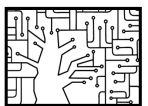
```
def main():  
    ...  
    torch.multiprocessing.spawn(train,  
                                nproc=world_size,  
                                args=args)  
  
def train(rank, args):  
    # init the process in context
```



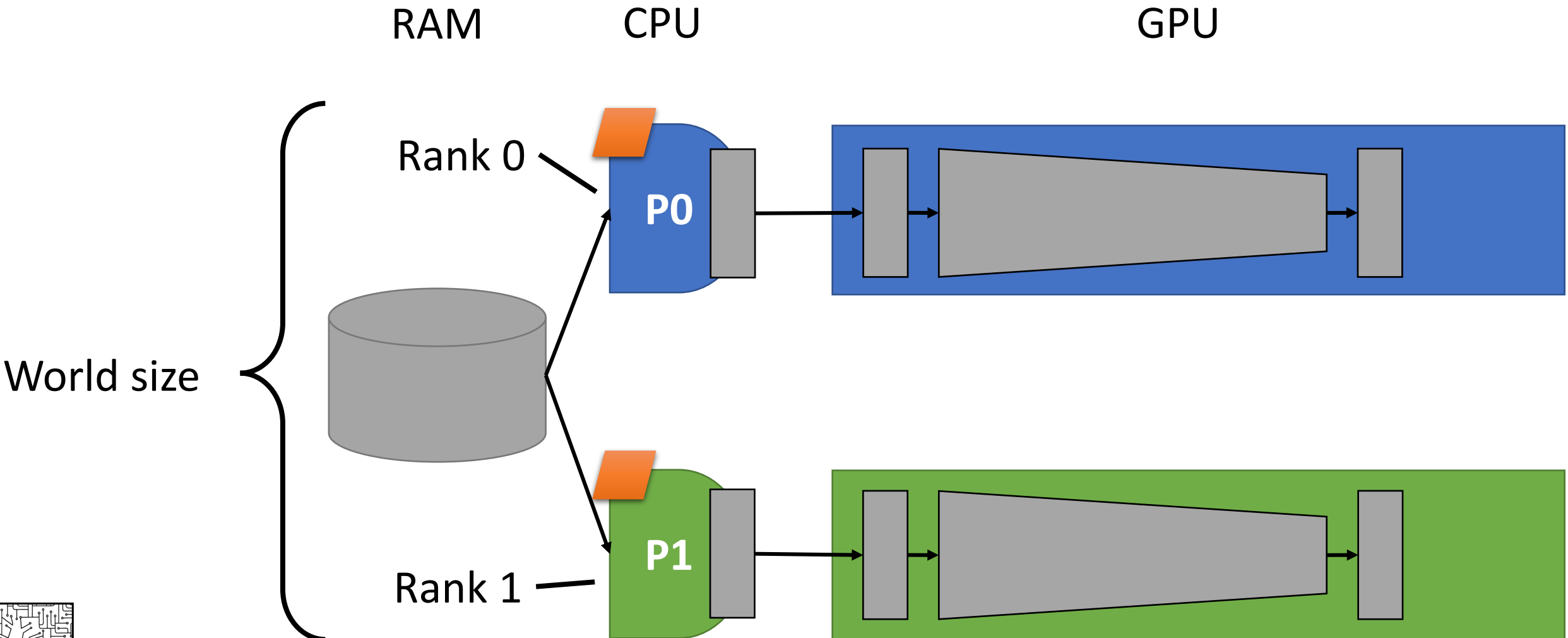
Data Parallel DistributedDataParallel

Same code – multiple processes

- Print/log only from rank 0
- Do eval only from rank 0



Data Parallel DistributedDataParallel



Data Parallel with PyTorch

DataParallel

```
model = MyModel()
# make it parallel
model = nn.DataParallel(model)
```

Handles everything for you

Increased overhead

DistributedDataParallel

```
def train(rank, args):
    # init the process in context
    torch.distributed.init_process_group(backend='nccl', init_method='tcp://127.0.0.1:54263', world_size, rank)
    ...
    # wrap the model
    model = nn.parallel.DistributedDataParallel(model, device_ids=[rank])

    # use "distributed aware" sampler
    sampler = torch.utils.data.distributed.DistributedSampler(
        train_dataset,
        num_replicas=world_size,
        rank=rank)

    loader = DataLoader(train_dataset, batch_size,
                        shuffle=False, sampler=train_sampler)
```

```
def main():
    torch.multiprocessing.spawn(train,
                                nproc=world_size,
                                args=args)

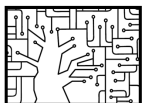
def train(rank, args):
    # init the process in context
```

Requires coding effort

Significant speedup

Scales to multiple machines

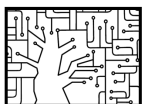
Flexible parallel scenarios



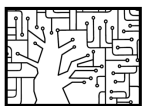
On WAIC

```
bsub -gpu num=2:j_exclusive=yes
-q waic-medium
-R rusage[mem=128000] -R affinity[thread*20]
-Is /bin/bash
```

```
NVIDIA-SMI 440.64.00    Driver Version: 440.64.00    CUDA Version: 10.2
+-----+-----+-----+-----+-----+-----+
| GPU  Name          Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf    Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute M. |
+-----+-----+-----+-----+-----+-----+-----+
|  0   Tesla V100-DGXS...  Off   | 00000000:08:00.0 Off  |             0MiB / 32508MiB |    0%      Default   |
| N/A   29C    P0     36W / 300W |             0MiB / 32508MiB |             |             |
+-----+-----+-----+-----+-----+-----+-----+
|  1   Tesla V100-DGXS...  Off   | 00000000:0E:00.0 Off  |             0MiB / 32508MiB |    0%      Default   |
| N/A   29C    P0     38W / 300W |             0MiB / 32508MiB |             |             |
+-----+-----+-----+-----+-----+-----+-----+
Processes:
GPU      PID    Type   Process name                      GPU Memory
Usage
+-----+-----+-----+-----+-----+-----+
No running processes found
```

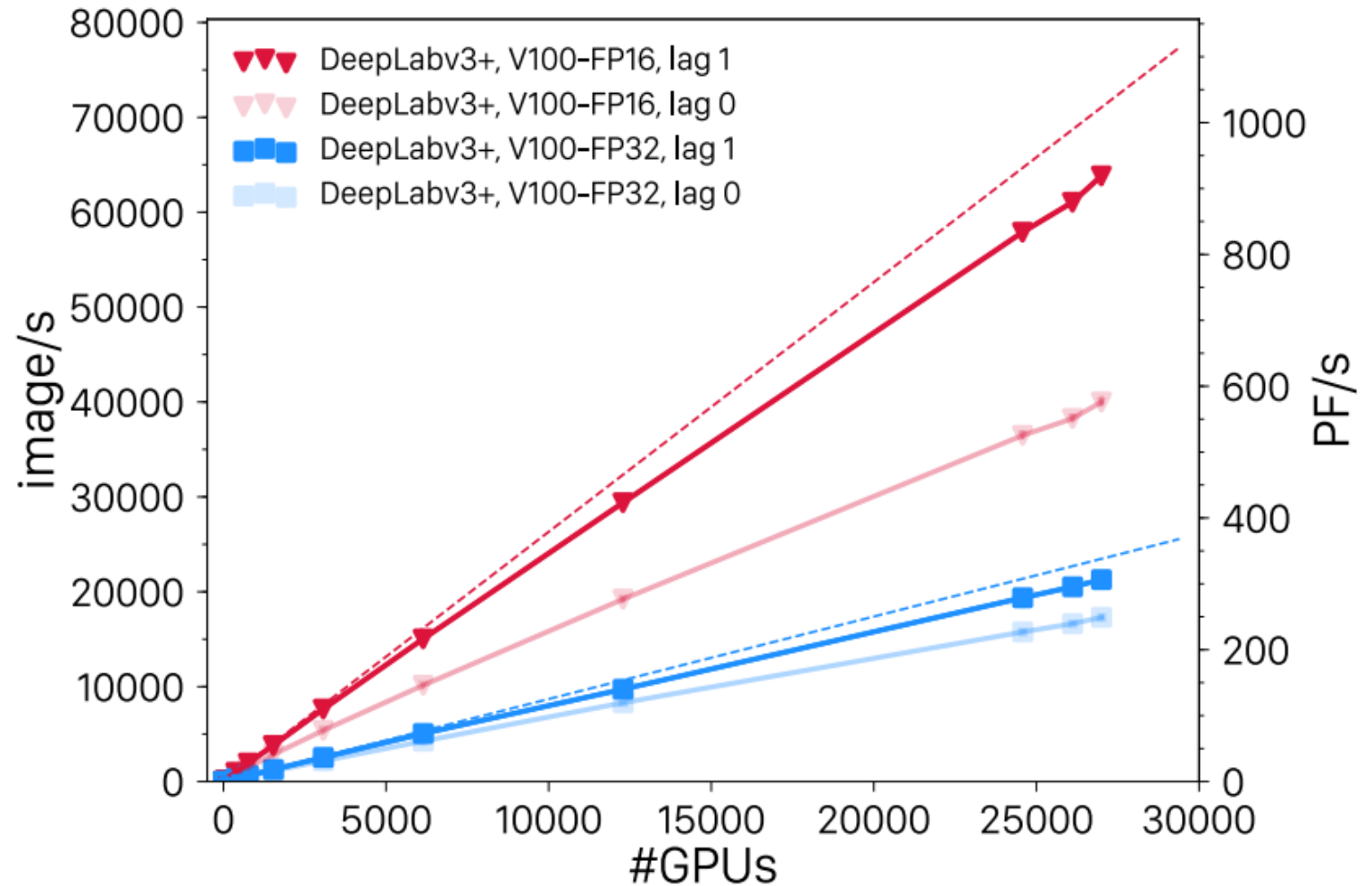


Before you leave...



Do we always want larger batches?

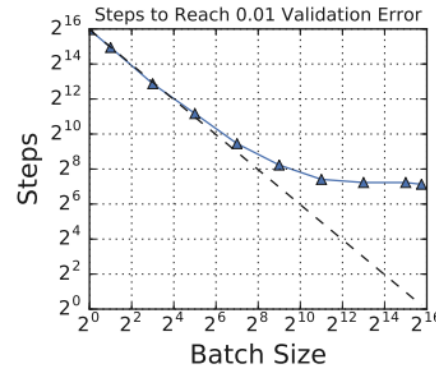
Images/sec **YES!**



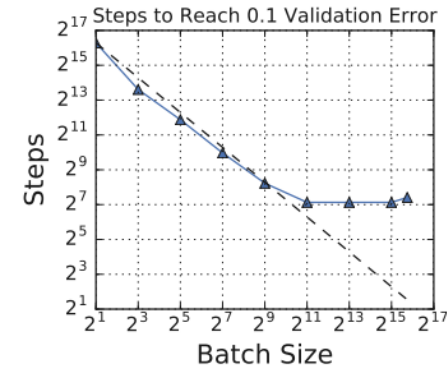
Kurth, Thorsten, Sean Treichler, Joshua Romero, Mayur Mudigonda, Nathan Luehr, Everett Phillips, Ankur Mahesh et al. "[Exascale deep learning for climate analytics](#)" (2018)

Do we always want larger batches?

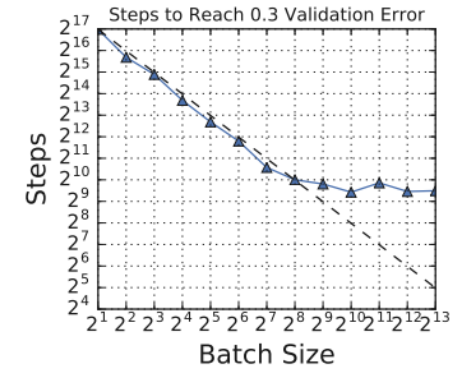
Steps to converge **NO!**



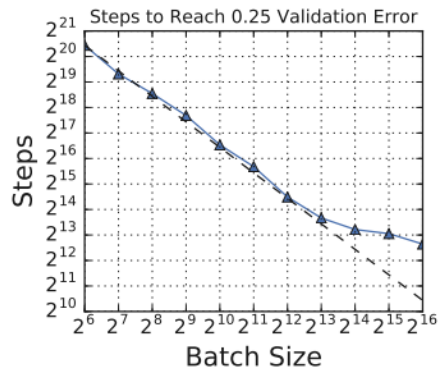
(a) Simple CNN on MNIST



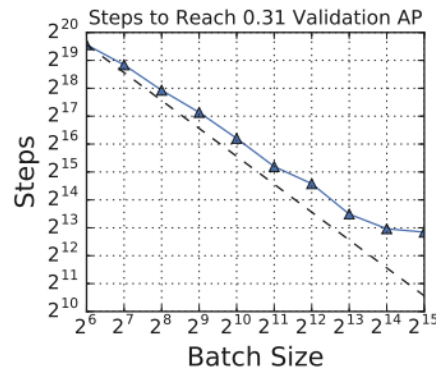
(b) Simple CNN on Fashion MNIST



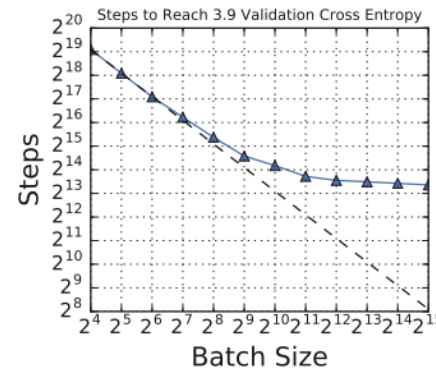
(c) ResNet-8 on CIFAR-10



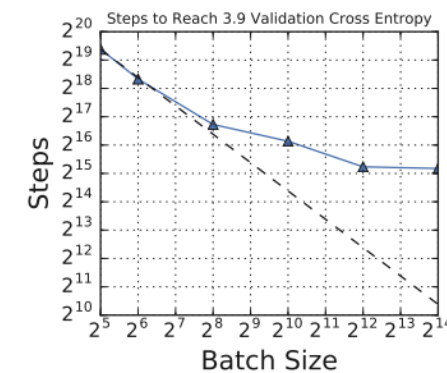
(d) ResNet-50 on ImageNet



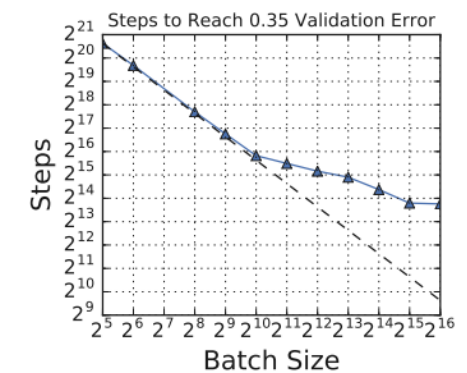
(e) ResNet-50 on Open Images



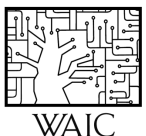
(f) Transformer on LM1B



(g) Transformer on Common Crawl

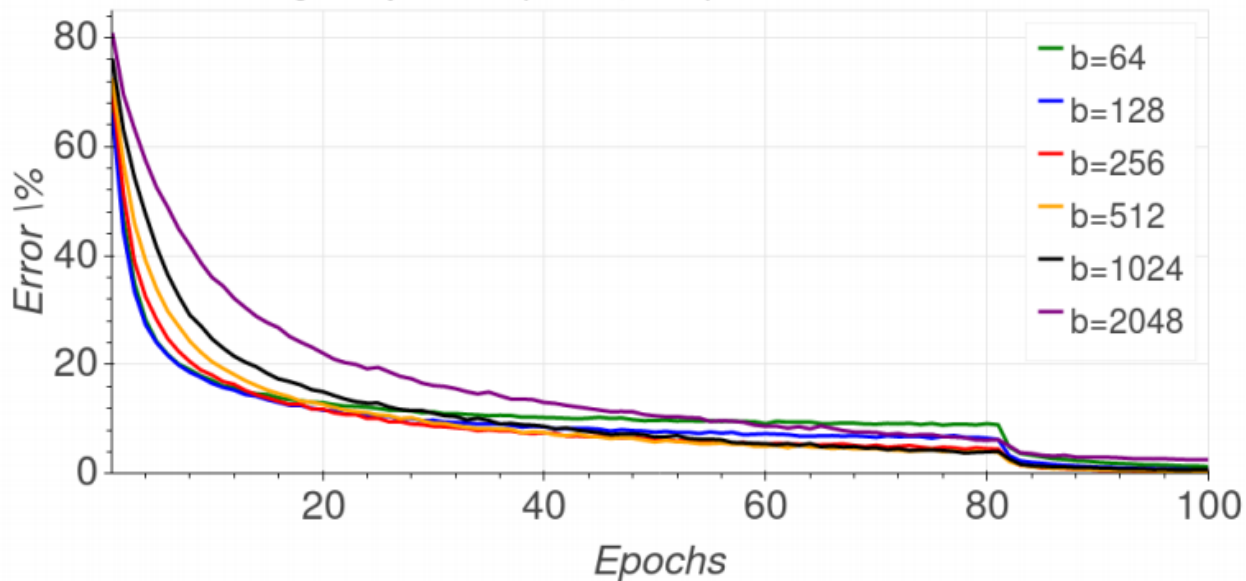


(h) VGG-11 on ImageNet

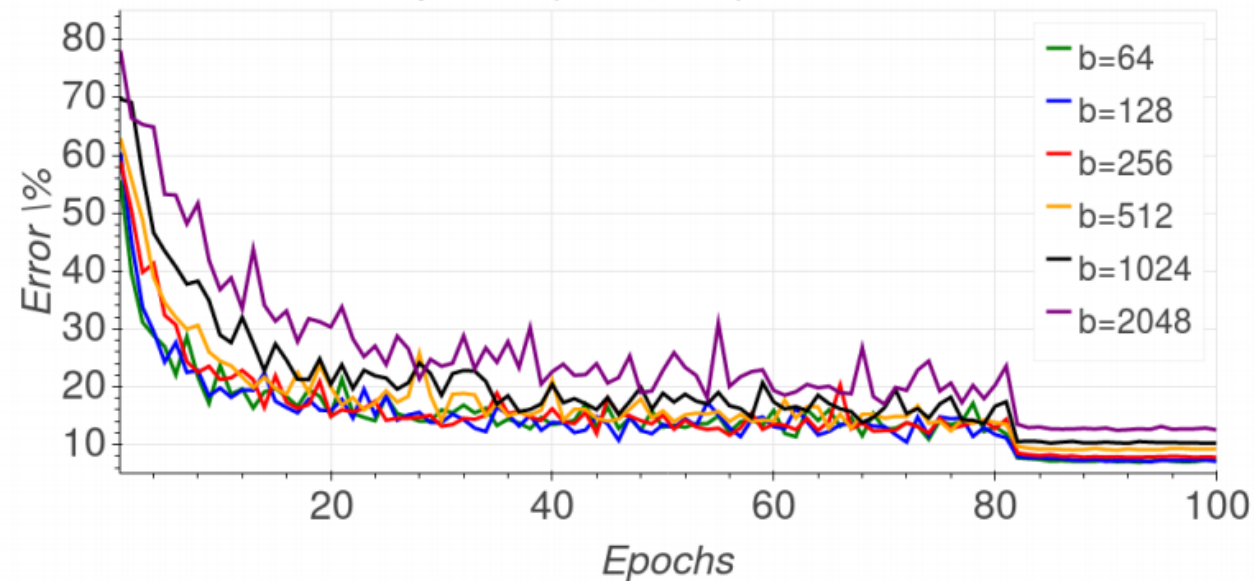


Do we want larger batches?

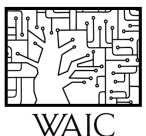
Generalization **NO!**



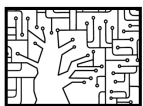
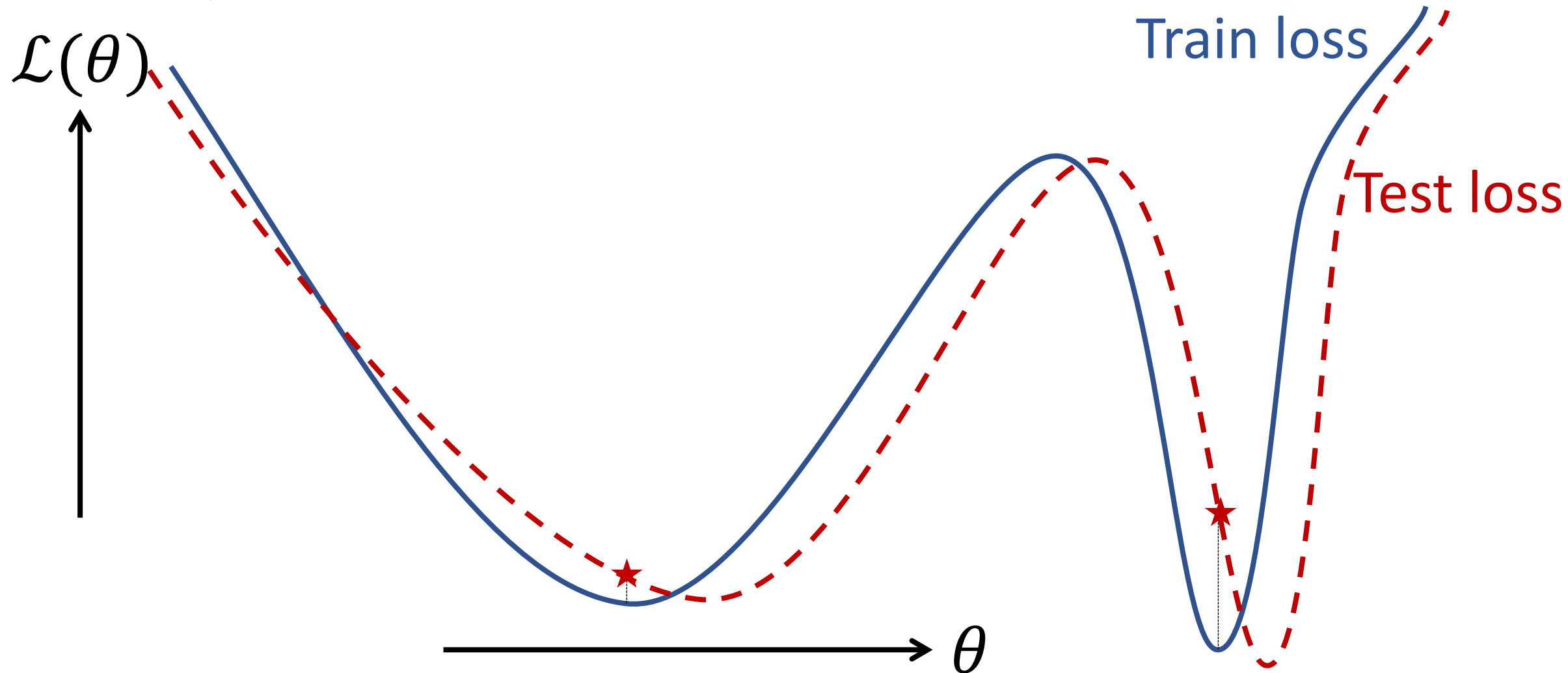
(a) Training error



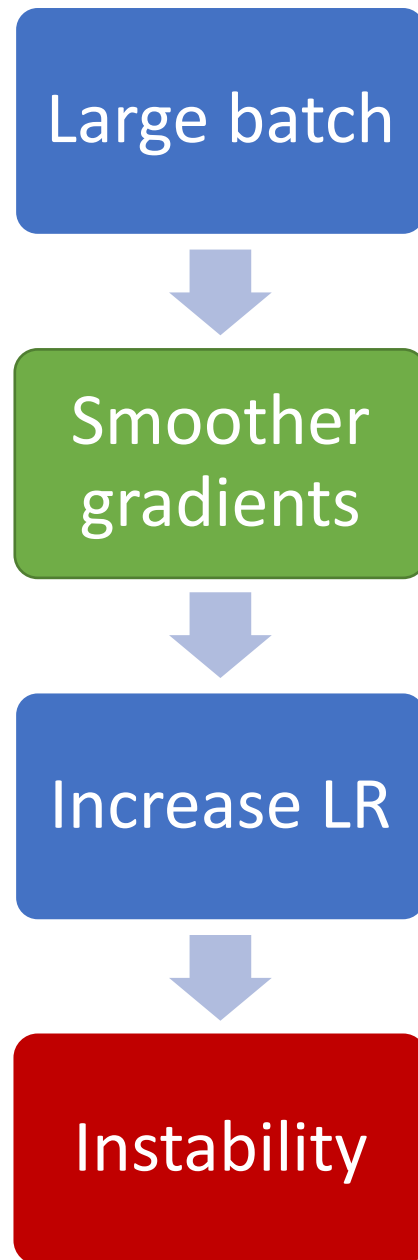
(b) Validation error



Why?

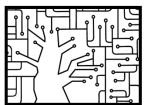


Do we want larger batches?



Multiple GPU - Summary

- Parallel GPUs – when more compute/memory is needed
- Model Parallel – when GPU mem is required
- Data Parallel – when GPU compute is required
- Data Parallel – ready made in pytorch
- Accumulating gradients – when many GPUs are not available



Multiple GPU - Summary

