Distributed Training with PyTorch

@shagunsodhani

Toronto Machine Learning Summit, 2022

About Me

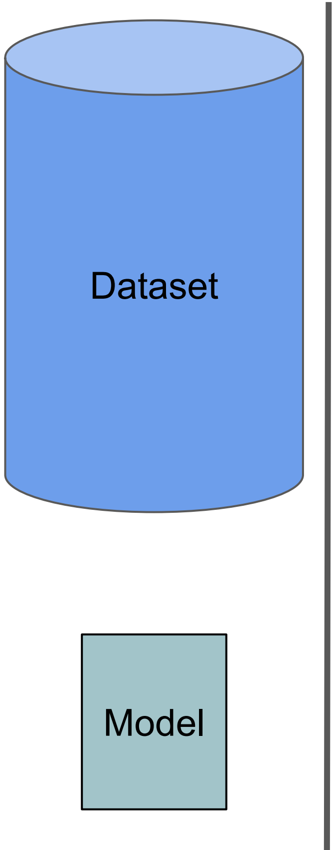
1. Research Engineer @ Meta AI

2. Focusing on building AI agents that can:
    a. interact with and learn from the physical world
    b. consistently improve as they do so without forgetting the previous knowledge

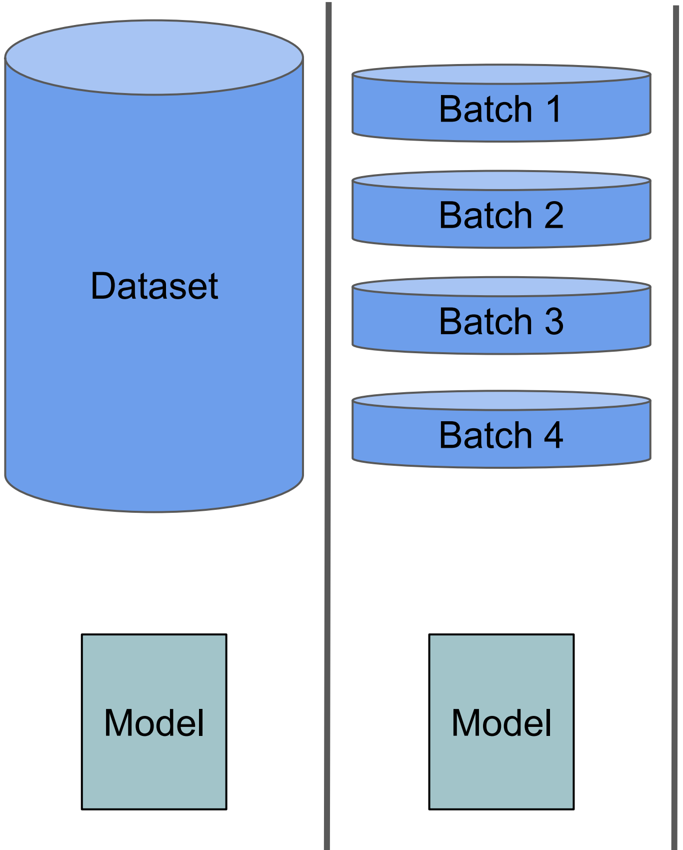
Agenda

1. Torch Distributed
2. Data Parallel (DP)
3. Distributed Data Parallel (DDP)
4. Fully Sharded Distributed Data Parallel (FSDP)

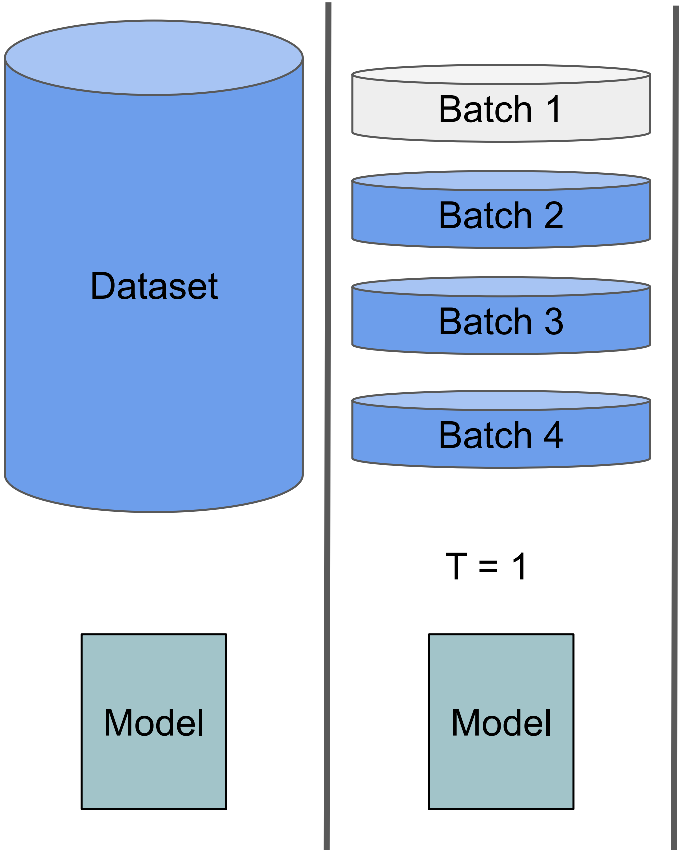
Single GPU | Data Parallel (DP)



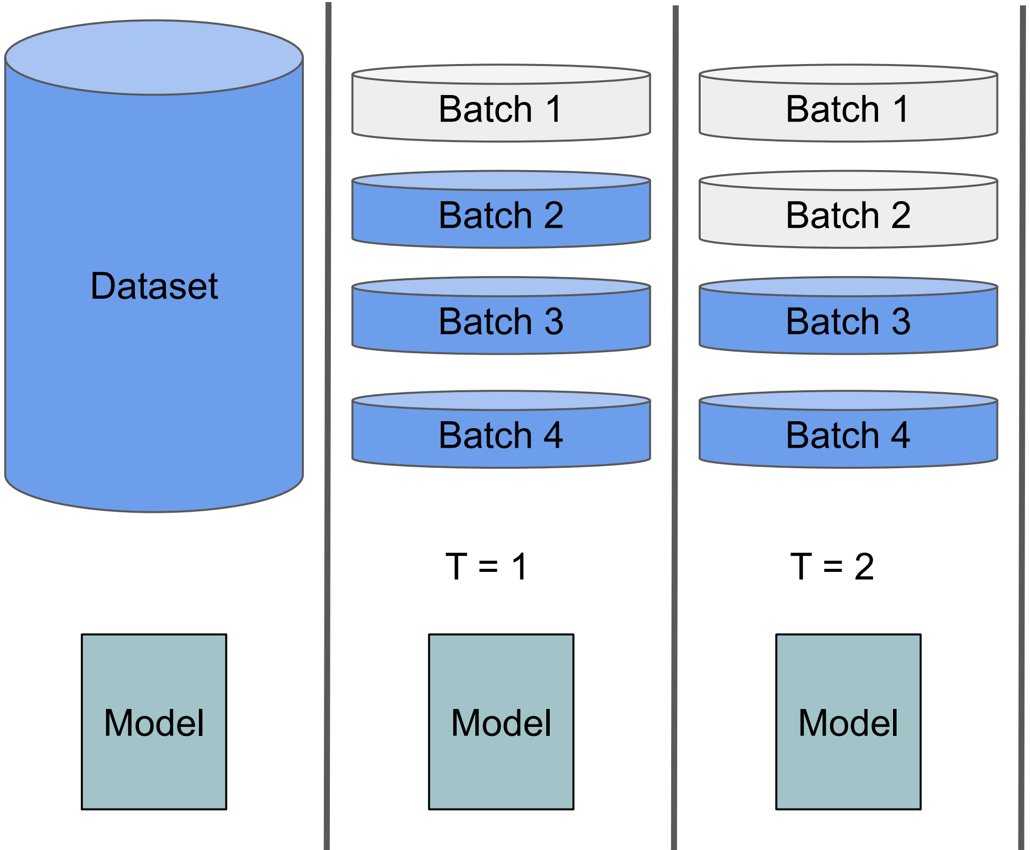
Single GPU | Data Parallel (DP)



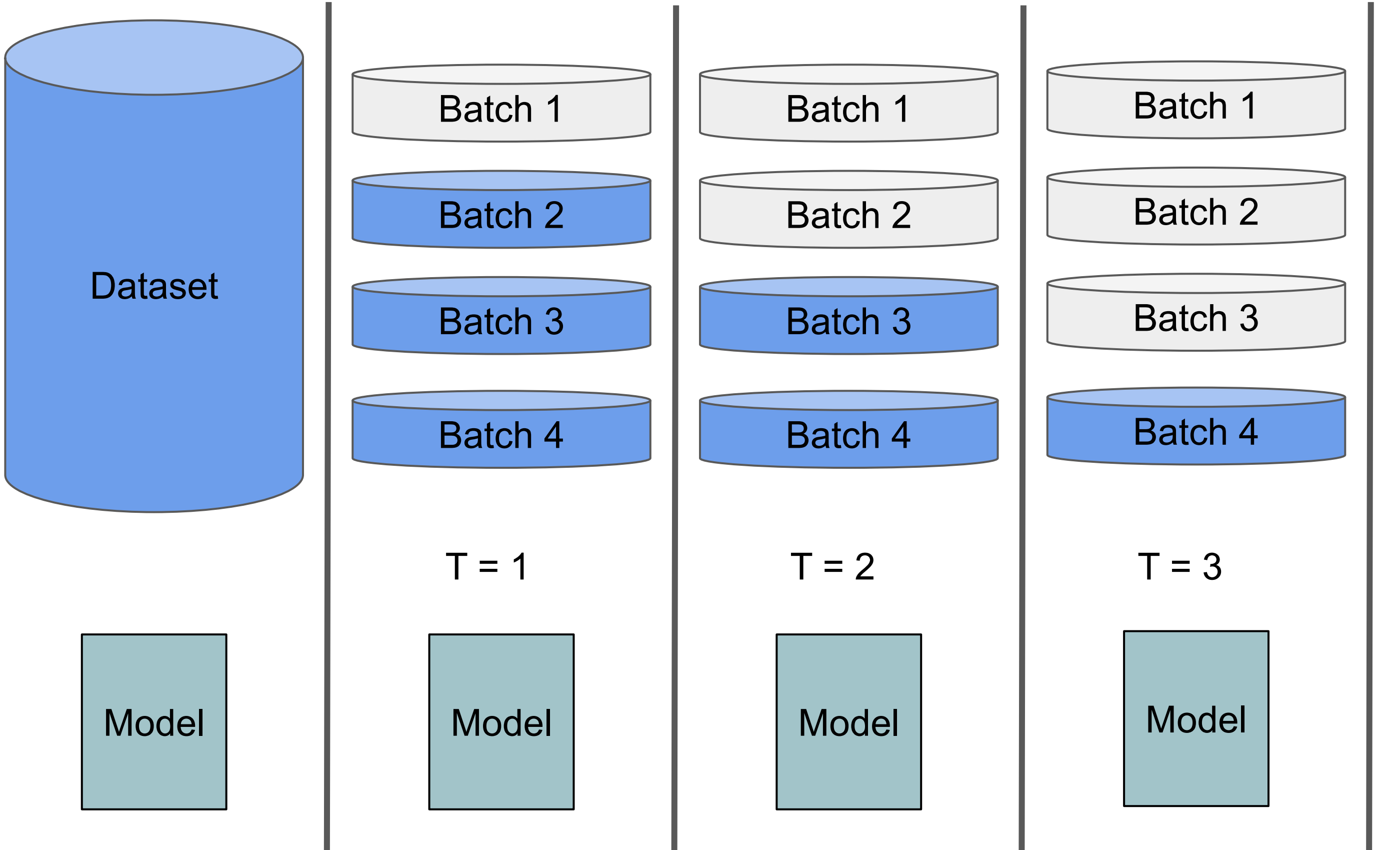
Single GPU | Data Parallel (DP)



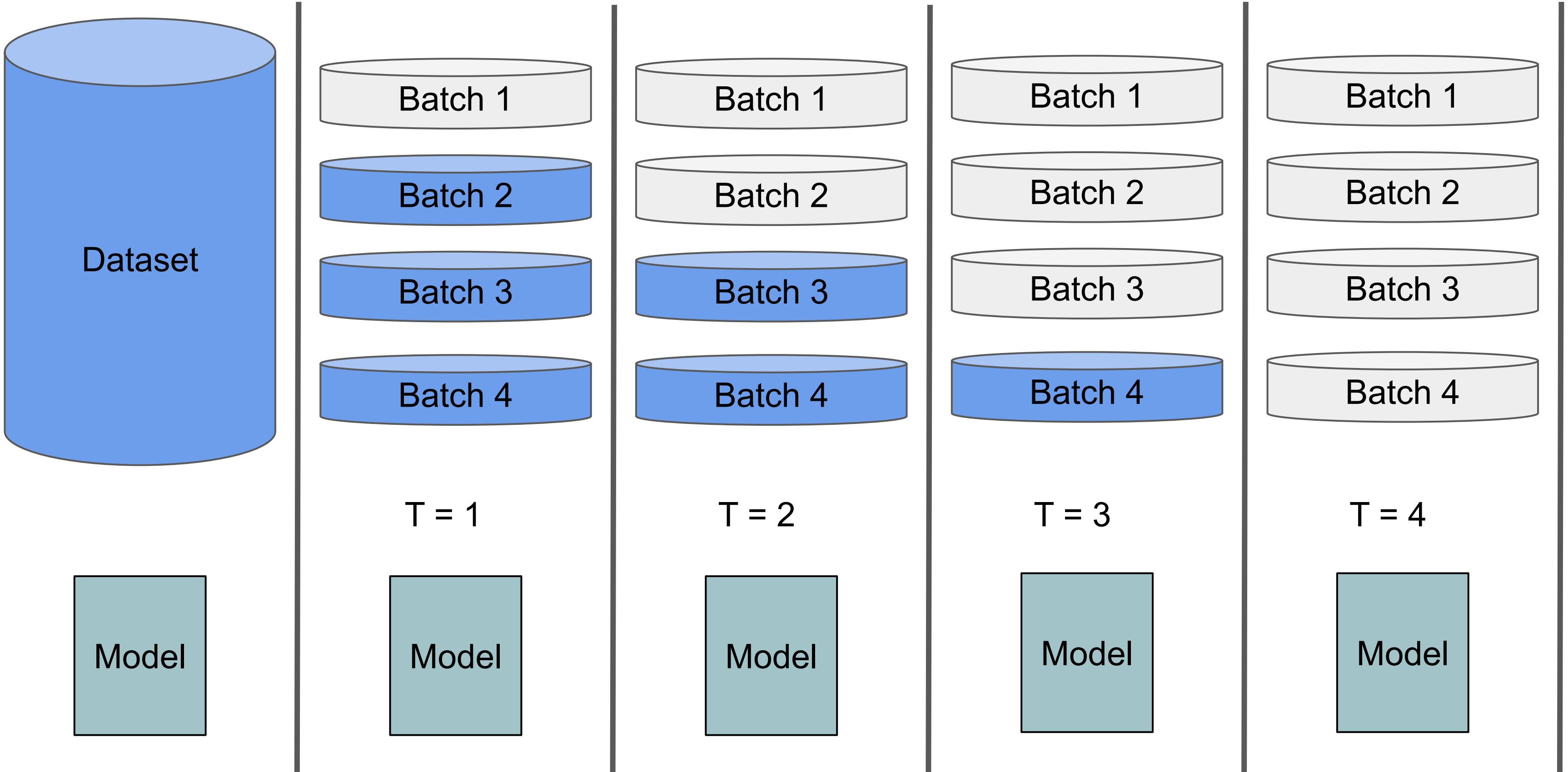
Single GPU | Data Parallel (DP)



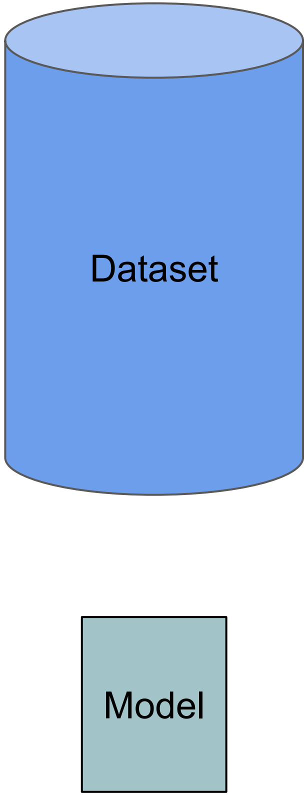
Single GPU | Data Parallel (DP)



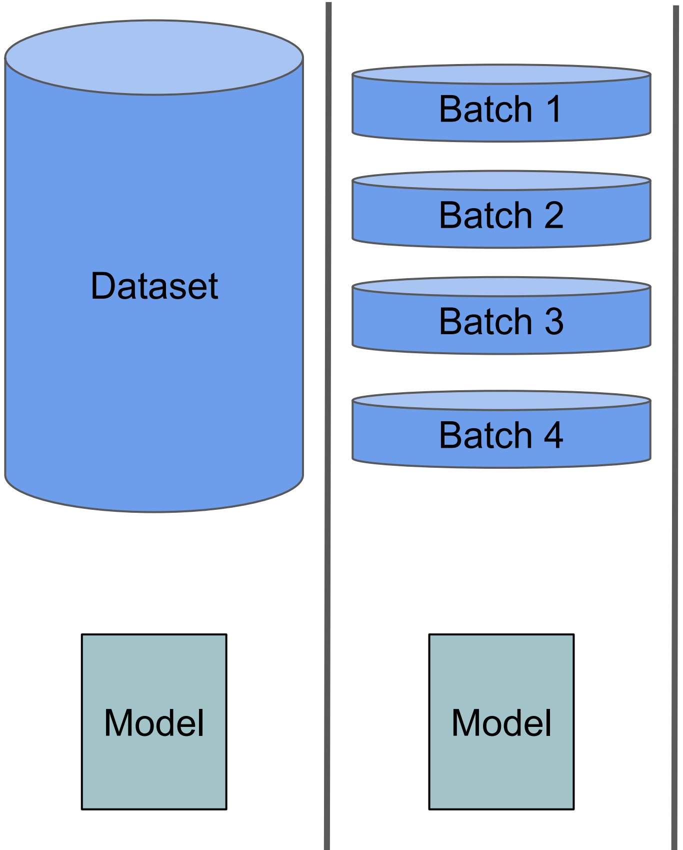
Single GPU | Data Parallel (DP)



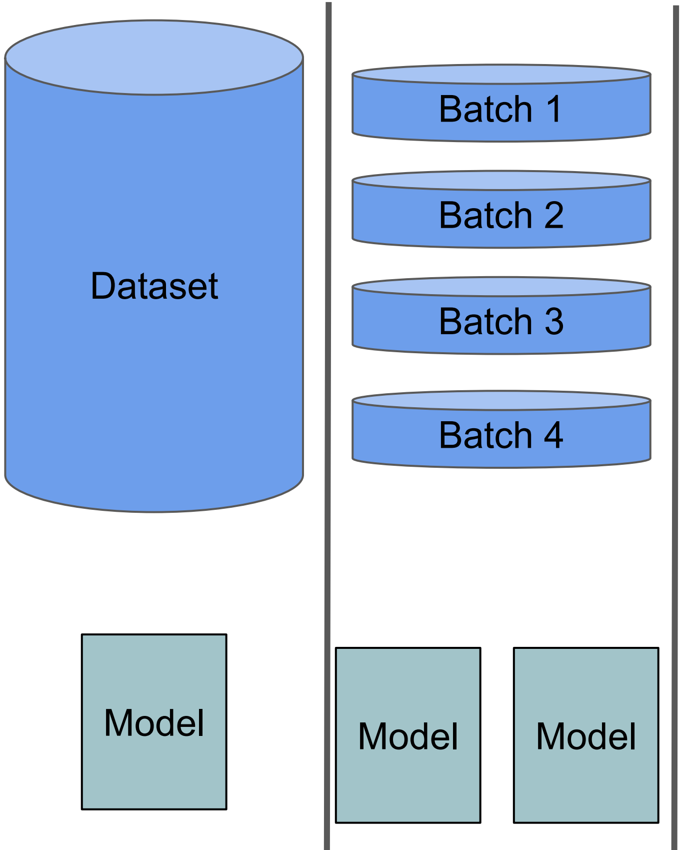
Multi GPU | Data Parallel (DP)



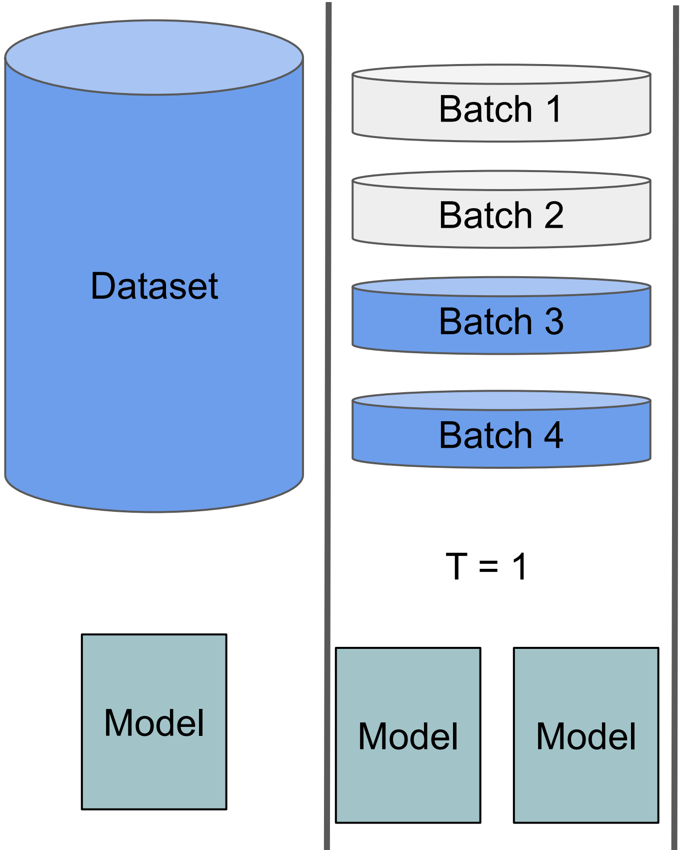
Multi GPU | Data Parallel (DP)



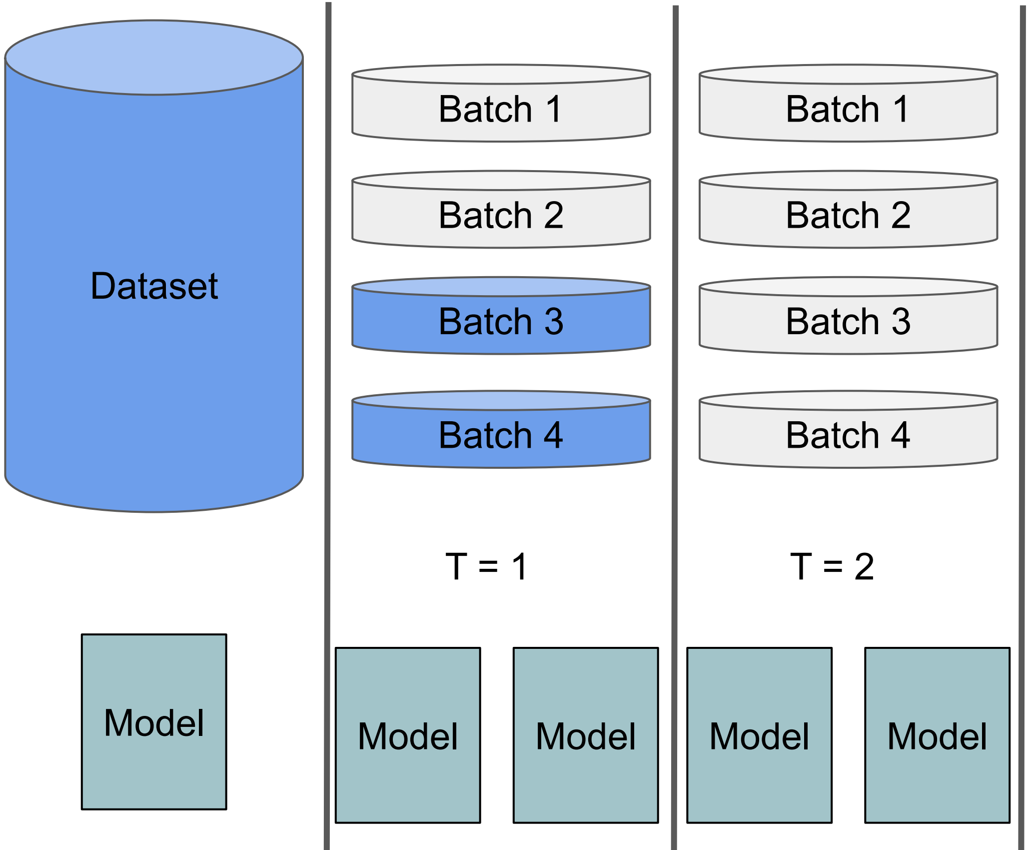
Multi GPU | Data Parallel (DP)



Multi GPU | Data Parallel (DP)



Multi GPU | Data Parallel (DP)

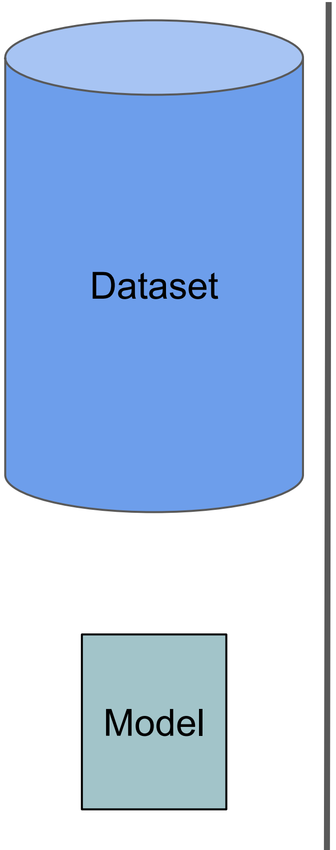


Multi GPU | Data Parallel (DP)

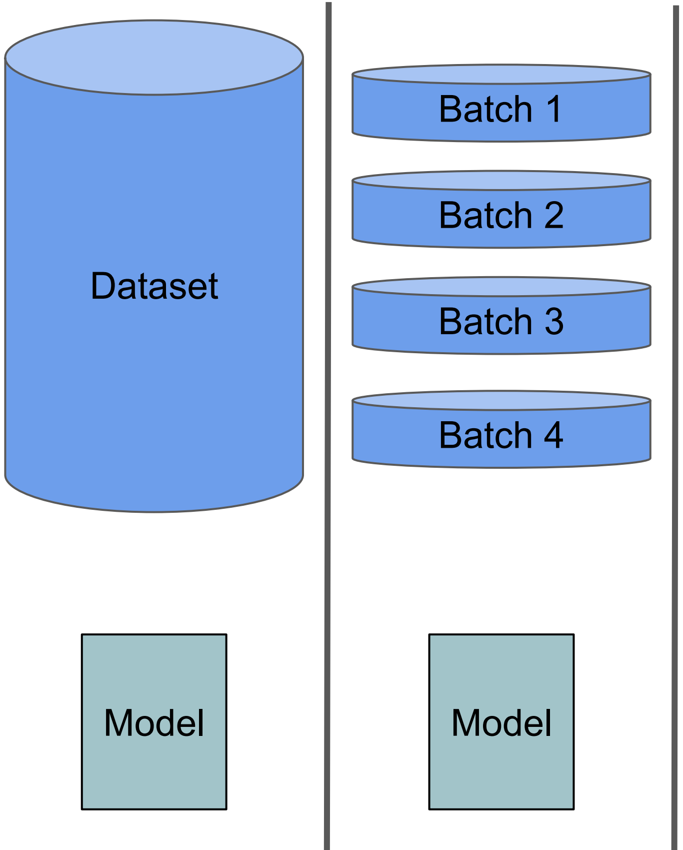
```
model: torch.nn.Module
input: torch.Tensor
output = model(input)
```

```
model: torch.nn.Module
input: torch.Tensor
model = torch.nn.DataParallel(model)
output = model(input)
```

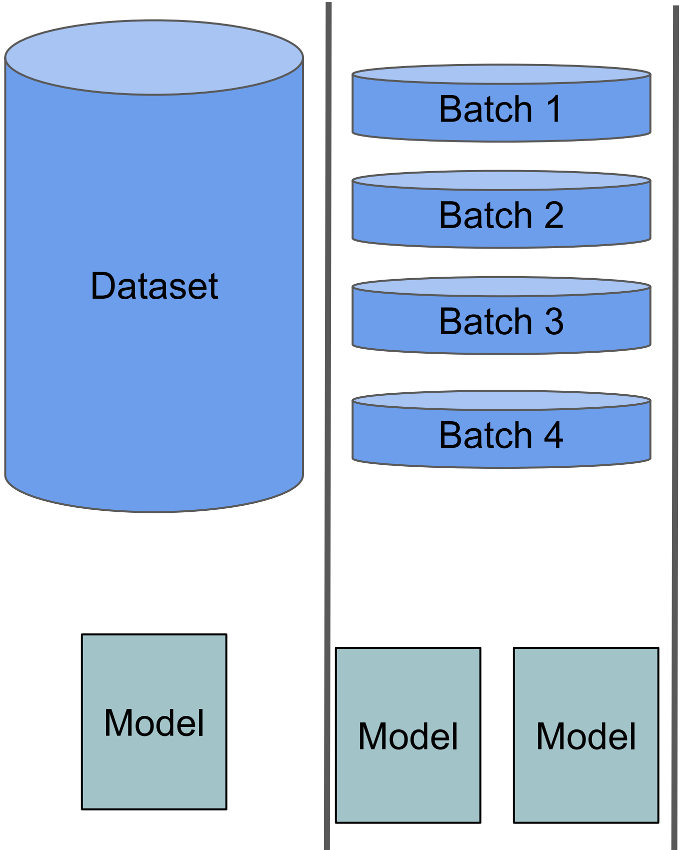
Multi GPU | Distributed Data Parallel (DDP)



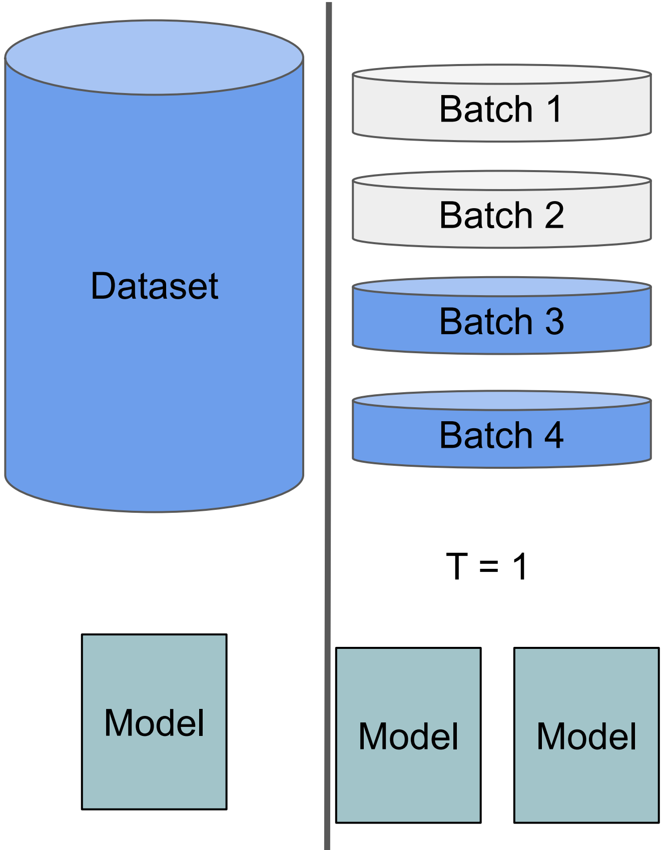
Multi GPU | Distributed Data Parallel (DDP)



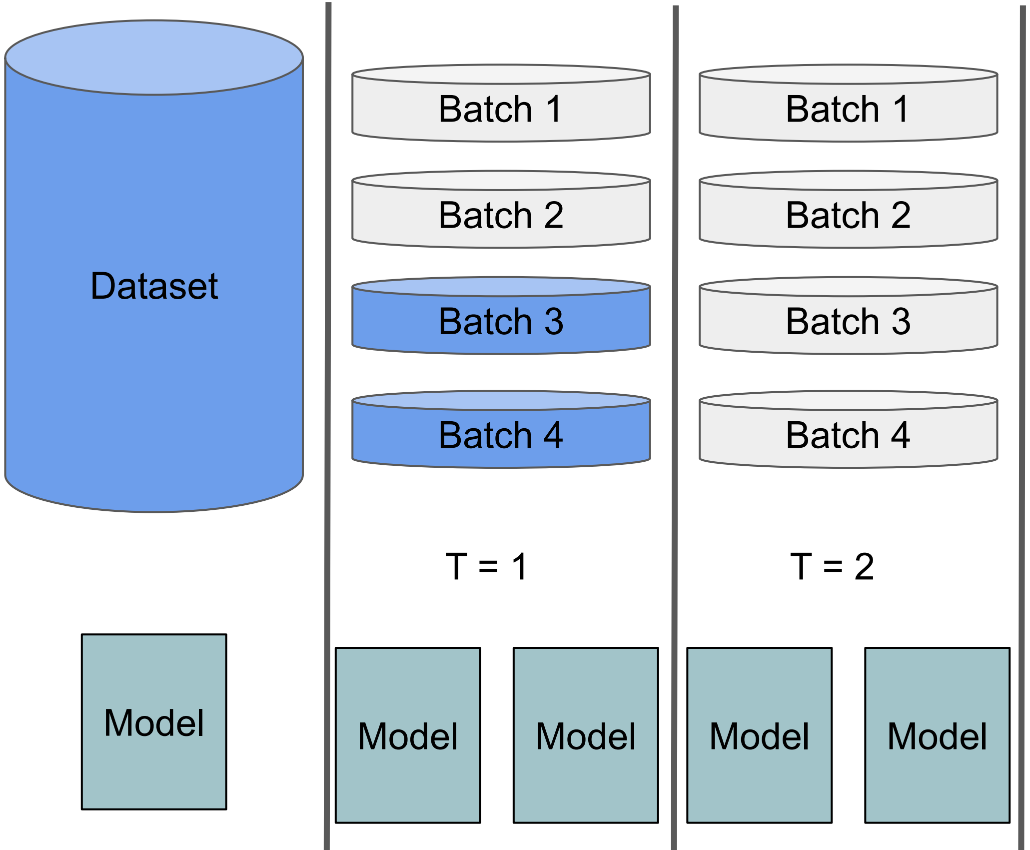
Multi GPU | Distributed Data Parallel (DDP)



Multi GPU | Distributed Data Parallel (DDP)



Multi GPU | Distributed Data Parallel (DDP)



Multi GPU | Distributed Data Parallel (DDP)

1. Start multiple processes, one process per gpu

2. For each process, initialize process groups

3. Update dataloader to use DistributedSampler

4. For each process, destroy the process group

Start multiple processes | DDP

```
world_size = torch.cuda.device_count()
mp.spawn(main, args=(world_size, ...), nprocs=world_size)
```

Start multiple processes | DDP

```
world_size = torch.cuda.device_count()
mp.spawn(main, args=(world_size, ...), nprocs=world_size)
```

```
def main(rank: int, world_size: int, *args, **kwargs):
    ddp_setup(rank, world_size)
```

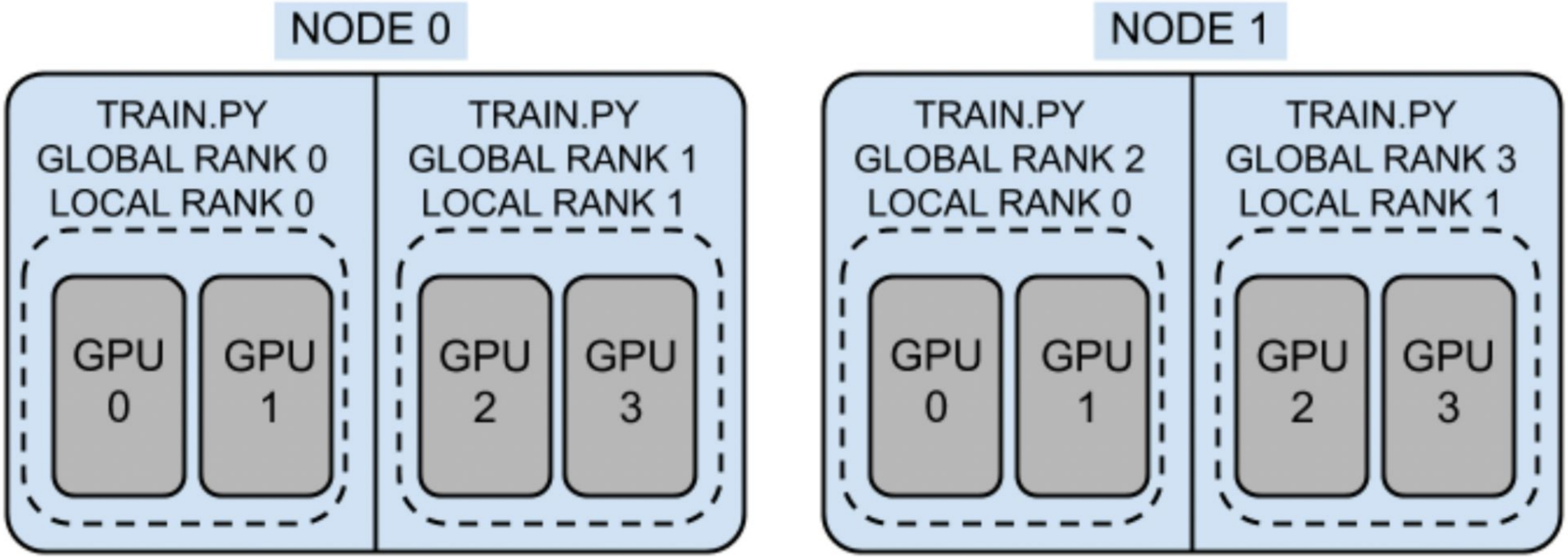
Start multiple processes | DDP

```
world_size = torch.cuda.device_count()
mp.spawn(main, args=(world_size, ...), nprocs=world_size)
```

```
def main(rank: int, world_size: int, *args, **kwargs):
    ddp_setup(rank, world_size)
```

```
def ddp_setup(rank, world_size):
    os.environ["MASTER_ADDR"] = "localhost"
    os.environ["MASTER_PORT"] = "12355"
    init_process_group(backend="nccl", rank=rank, world_size=world_size)
```

World Size and Rank | DDP



Taken from https://github.com/pytorch/examples/blob/main/distributed/ddp/README.md

Initialize Process Groups | DDP

Backend	`gloo`	`mpi`	`nccl`

Taken from https://github.com/pytorch/examples/blob/main/distributed/ddp/README.md

Initialize Process Groups | DDP

Backend	`gloo`	`mpi`	`nccl`

Taken from https://github.com/pytorch/examples/blob/main/distributed/ddp/README.md

Initialize Process Groups | DDP

Backend	gloo		mpi		nccl	
Device	CPU	GPU	CPU	GPU	CPU	GPU
send	✓	✘	✓	?	✘	✓
recv	✓	✘	✓	?	✘	✓
broadcast	✓	✓	✓	?	✘	✓
all_reduce	✓	✓	✓	?	✘	✓
reduce	✓	✘	✓	?	✘	✓

Taken from https://github.com/pytorch/examples/blob/main/distributed/ddp/README.md

Which backend to use | DDP

- Rule of thumb
  - Use the NCCL backend for distributed **GPU** training
  - Use the Gloo backend for distributed **CPU** training.
- GPU hosts with InfiniBand interconnect
  - Use NCCL, since it's the only backend that currently supports InfiniBand and GPUDirect.
- GPU hosts with Ethernet interconnect
  - Use NCCL, since it currently provides the best distributed GPU training performance, especially for multiprocess single-node or multi-node distributed training. If you encounter any problem with NCCL, use Gloo as the fallback option. (Note that Gloo currently runs slower than NCCL for GPUs.)
- CPU hosts with InfiniBand interconnect
  - If your InfiniBand has enabled IP over IB, use Gloo, otherwise, use MPI instead. We are planning on adding InfiniBand support for Gloo in the upcoming releases.
- CPU hosts with Ethernet interconnect
  - Use Gloo, unless you have specific reasons to use MPI.

Taken from https://github.com/pytorch/examples/blob/main/distributed/ddp/README.md

Which backend to use | DDP

1. NCCL for distributed GPU training

2. Gloo for distributed CPU training

Dataloader | DDP

```python
def prepare_dataloader(dataset: Dataset, batch_size: int):
    return DataLoader(
        dataset,
        batch_size=batch_size,
        pin_memory=True,
        shuffle=True
    )
```

```python
def prepare_dataloader(dataset: Dataset, batch_size: int):
    return DataLoader(
        dataset,
        batch_size=batch_size,
        pin_memory=True,
        shuffle=False,
        sampler=DistributedSampler(dataset)
    )
```

Destroy the process group | DDP

```
from torch.distributed import destroy_process_group
destroy_process_group()
```

Distributed Data Parallel (DDP) vs Data Parallel (DP)

DDP	DP
Implements data parallelism at model level	
Uses multiprocessing	Uses multithreading
Preferred	Not Recommended
Requires writing more code	Requires a one-line change

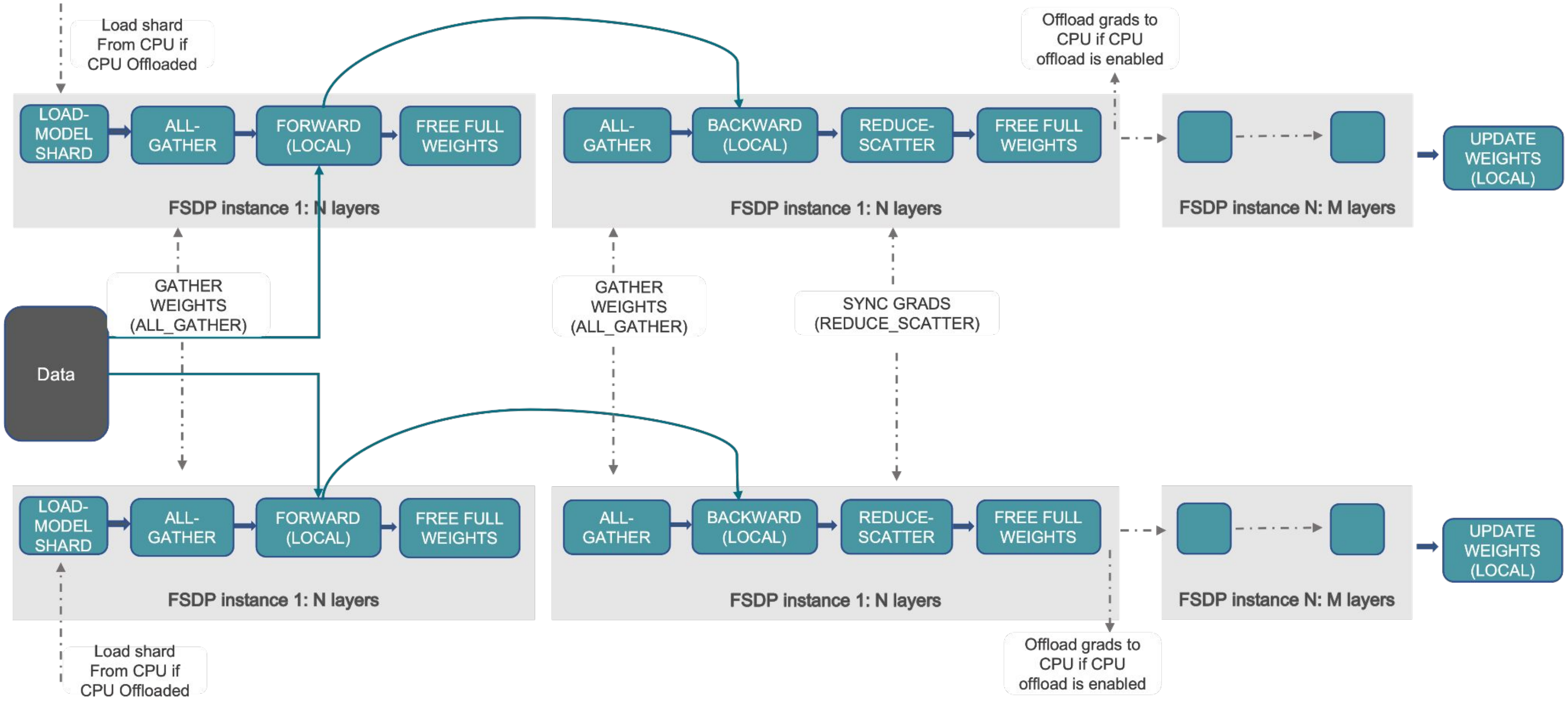
Fully Sharded Data Parallel (FSDP)

1. DDP (and DP) are useful when we have fit the model on one GPU.

2. What happens is the model is too big for one GPU?

3. FSDP to the rescue

Fully Sharded Data Parallel (FSDP)



Taken from https://pytorch.org/tutorials/intermediate/FSDP_tutorial.html

Fully Sharded Data Parallel (FSDP)

```
>>> import torch
>>> from torch.distributed.fsdp import FullyShardedDataParallel as FSDP
>>> torch.cuda.set_device(device_id)
>>> sharded_module = FSDP(my_module)
>>> optim = torch.optim.Adam(sharded_module.parameters(), lr=0.0001)
>>> x = sharded_module(x, y=3, z=torch.Tensor([1]))
>>> loss = x.sum()
>>> loss.backward()
>>> optim.step()
```

Taken from https://pytorch.org/docs/stable/fsdp.html

Motivation | Torch Distributed

1. Build custom workflows for training models

Overview | Torch Distributed

1. *torch.distributed* module

2. Provides communication primitives

3. *torch.distributed.send* or *torch.distributed.recv*

```
torch.distributed.send(tensor, dst, group=None, tag=0)
```

Taken from https://pytorch.org/docs/stable/distributed.html

Communication Primitives | Torch Distributed

1. Point-to-point communication
    a. *send, recv, isend, irecv*

2. Collective Operations
    a. *broadcast, reduce, gather…*

Taken from https://pytorch.org/docs/stable/distributed.html
For more details on Collective Operations, refer https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/usage/operations.html

References

1. https://pytorch.org/

2. https://pytorch.org/docs/stable/index.html

3. https://discuss.pytorch.org/

What did we not cover

1. Model Parallel
2. Distributed RPC
3. Distributed Optimizers
4. Distributed Elastic

Acknowledgements

Olivier Delalleau

Thank you!

@shagunsodhani