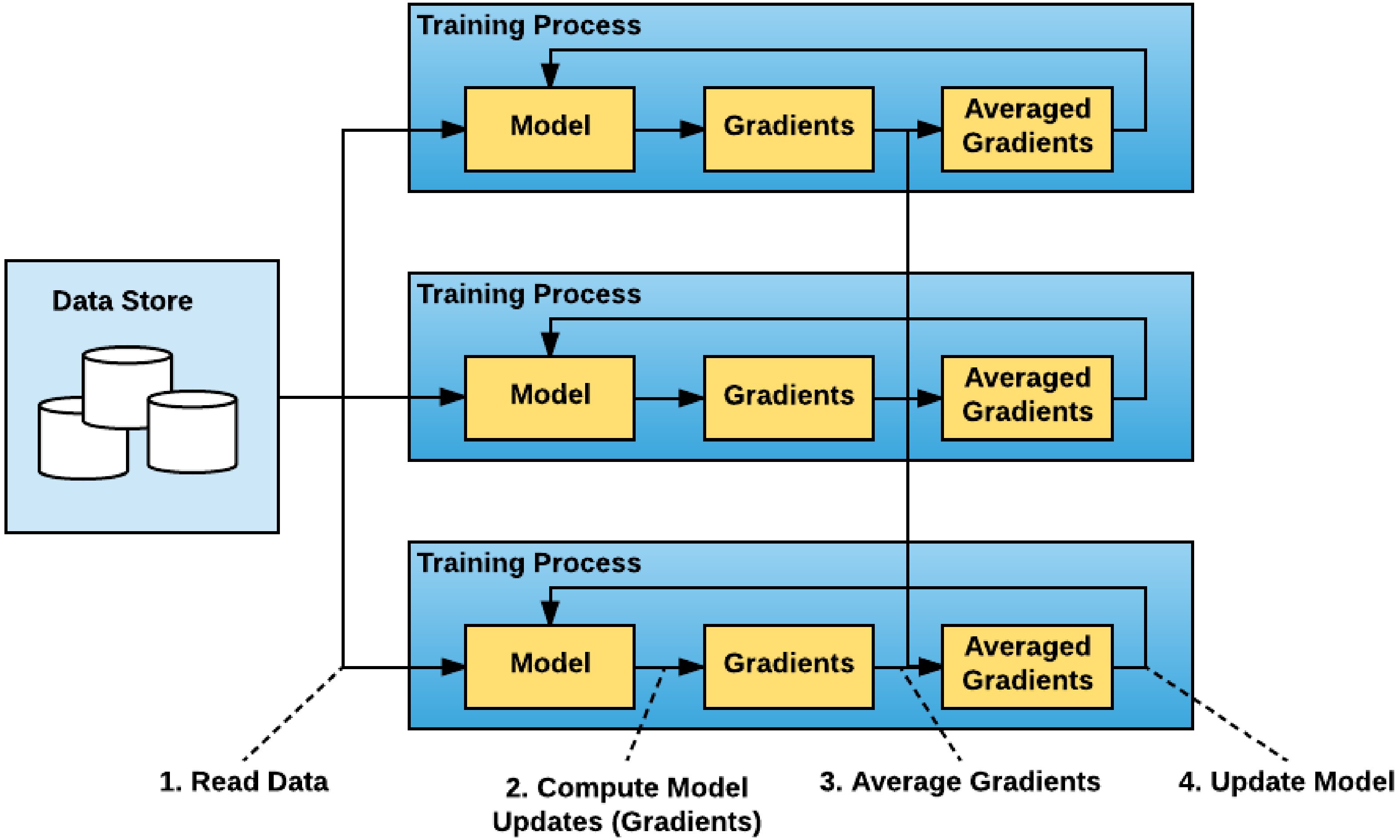
Multi-GPU

Why Multi-GPU Jobs?

- increased compute power
- increased GPU memory
- increased I/O bandwidth (GPUdirect)

Data Parallel (Batches)



(source: Horovod)

Data Parallel in PyTorch

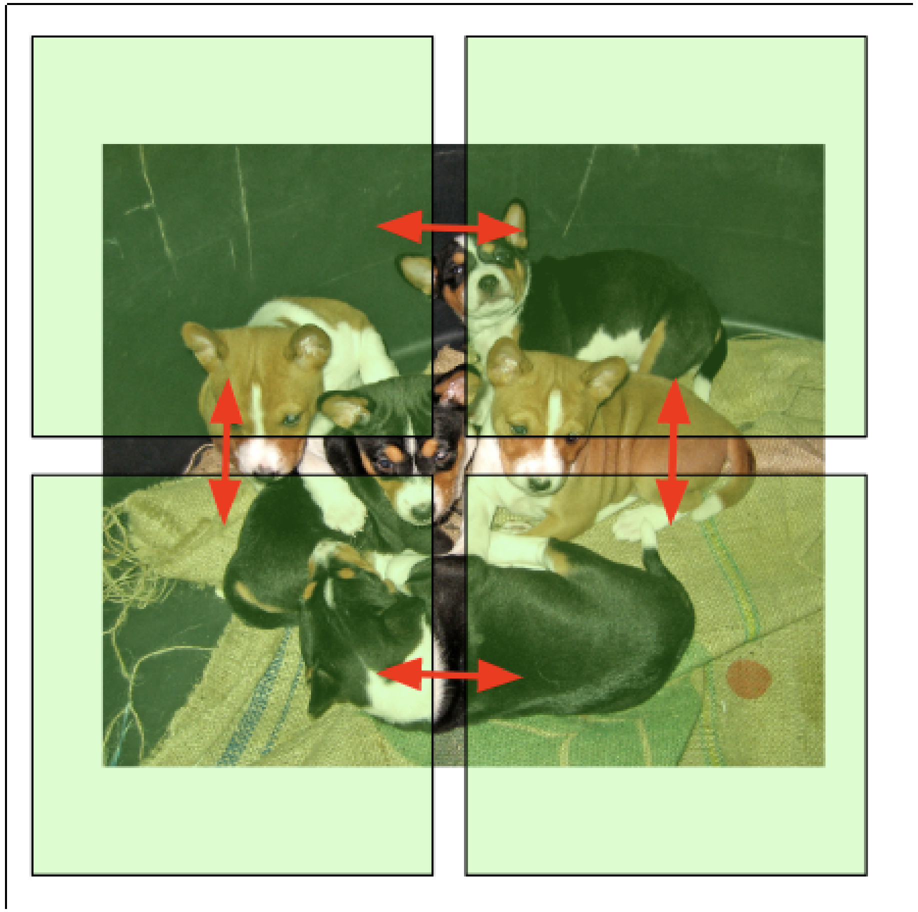
```
model = make_model()

if torch.cuda.device_count() > 1:
    model = nn.DataParallel(model)

model.to(device)
```

- Parameters are *tied* across GPUs (all parameters on all GPUs)

Tiled Processing



Tile Parallelism for Images

```
class ImageParallelExample(Module):
    def __init__(self, *args, **kwargs):
        l = Sequential(
            Conv2d(16), BatchNorm2d(), ReLU(), MaxPool2d(),
            Conv2d(32), BatchNorm2d(), ReLU(), MaxPool2d(),
        )
        layers = [l.to("cuda:%d"%i) for i in range(4)]

    def forward(self, x):
        b, d, h, w = x.shape
        parts = [ x[:, :, 0:h//2, 0:w//2], x[:, :, h//2:h, 0:w//2],
                  x[:, :, 0:h//2, w//2:w], x[:, :, h//2:h, w//2:w] ]
        y = [f(x) for f, x in zip(self.layers, parts)]
        return vstack([hstack([y[0], y[2]), hstack([y[1], y[3]])])
```

- parameters are *tied* across GPUs (all parameters on all GPUs)
- need to worry about boundary conditions

Model Parallelism in Depth

```
class ModelParallelExample(Module):
    def __init__(self, *args, **kwargs):
        self.seq1 = Sequential(
            Conv2d(16), BatchNorm2d(), ReLU(), MaxPool2d(),
            Conv2d(32), BatchNorm2d(), ReLU(), MaxPool2d(),
        ).to('cuda:0')
        self.seq2 = Sequential(
            Conv2d(128), BatchNorm2d(), ReLU(), MaxPool2d(),
            Conv2d(256), BatchNorm2d(), ReLU(), MaxPool2d(),
            Conv2d(512), BatchNorm2d(), ReLU(), MaxPool2d(),
        ).to('cuda:1')
    def forward(self, x):
        x = self.seq2(self.seq1(x.to('cuda:0')).to('cuda:1'))
        return x
```

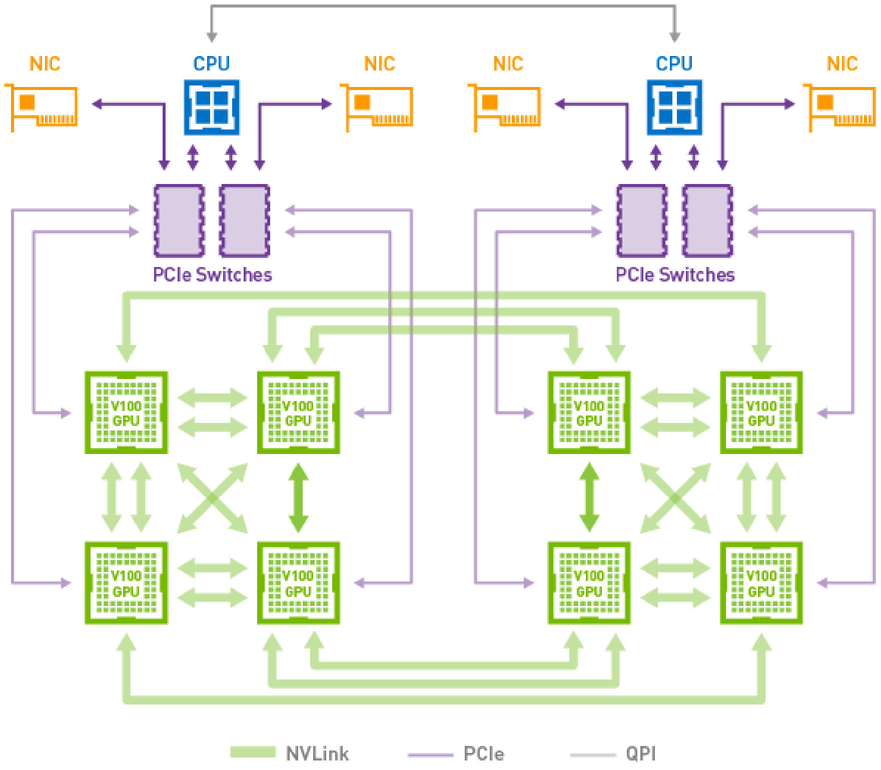
- each GPU has different parameters
- less opportunity for parallelism, but larger models possible (storage)

Latency vs. Throughput

- latency = time until a specific output arrives at output
- throughput = total number of outputs produced per unit time

For DL, we want high throughput, don't care much about latency.

CPU/GPU Communications



- GPU-to-GPU communications uses very fast NVLINK connections
- high-end GPU setups support GPUDirect without CPU involvement
- high performance communications abstracted by NCCL, PyTorch, TensorFlow

The DataParallel Bottleneck

- `DataParallel` is good for illustrating data parallelism
- the specific implementation is rather inefficient
  - multithreaded loader funnels data to single GPU
  - single GPU splits data to multiple GPUs
- better approach using `DistributedDataParallel` even on single node
  - each GPU gets its own I/O pipeline
  - only weights are synchronized
- still, `DataParallel` can work well for simple applications, or in-memory data

DataParallel Notebook

(notebook)