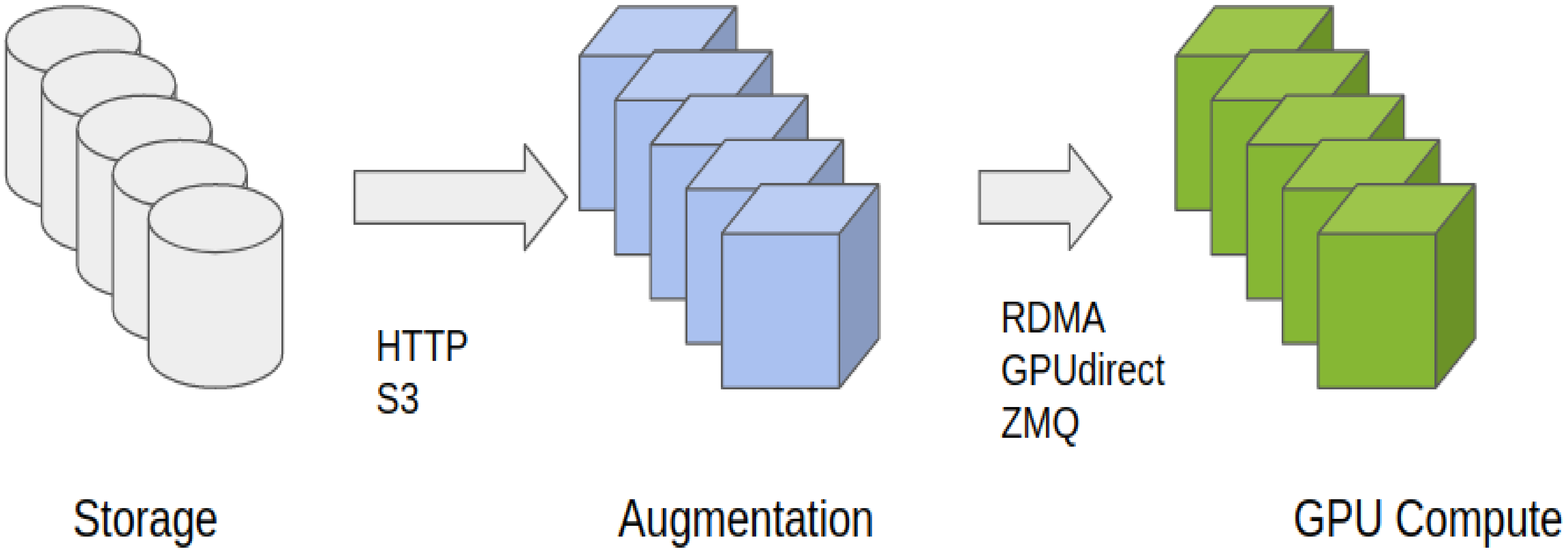
Distributed Loading and Agumentation

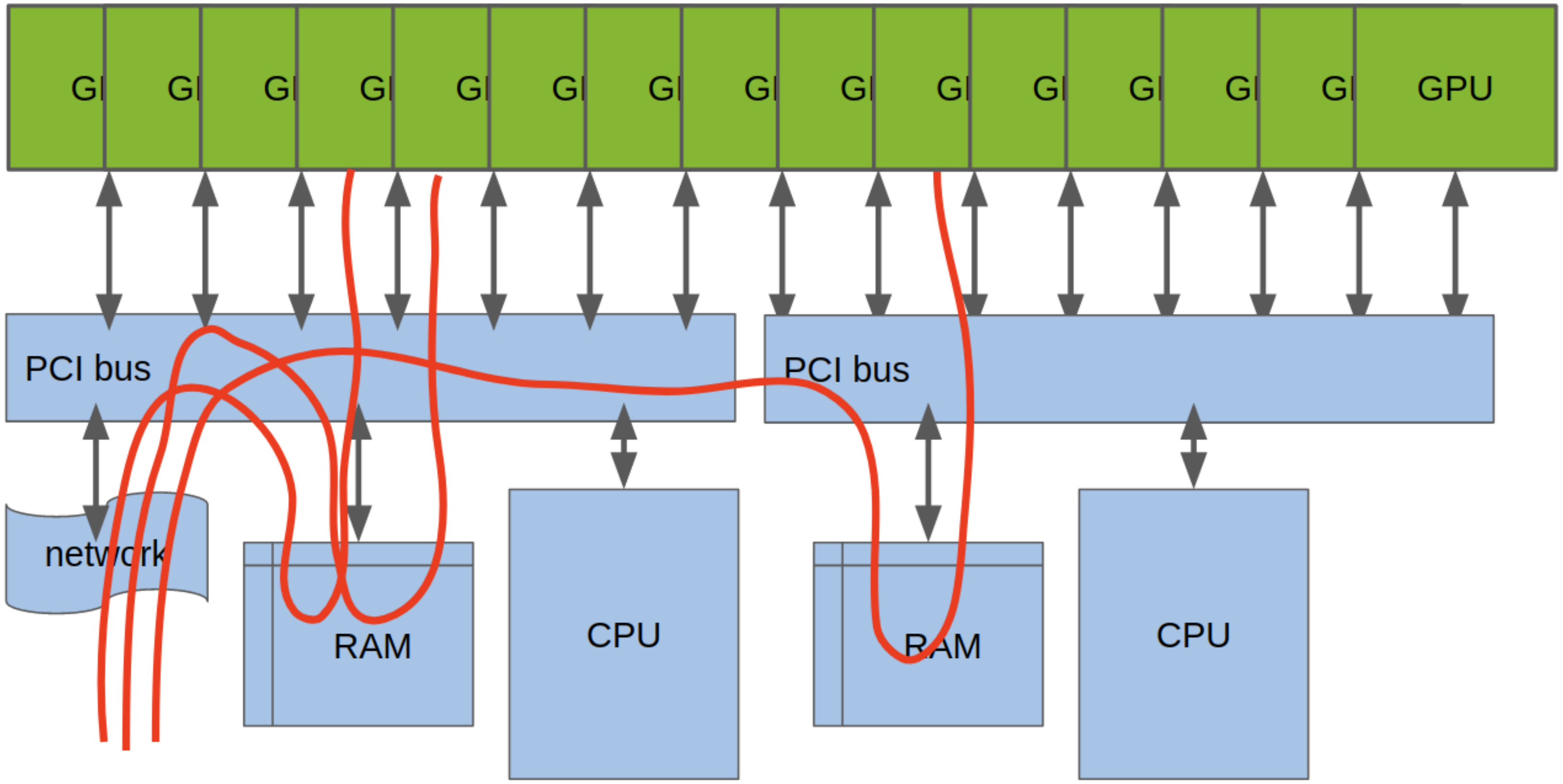
Architectures

- Two-Tier: Storage $\rightarrow$ GPU server
  - storage-to-GPU ratio infinitely scalable
  - CPU-cores-to-GPU ratio limited by GPU server
- Three-Tier: Storage $\rightarrow$ CPU server $\rightarrow$ GPU server
  - both storage-to-GPU and CPU-cores-to-GPU ratio independently scalable
  - also eliminates PCI bus bottlenecks via RDMA

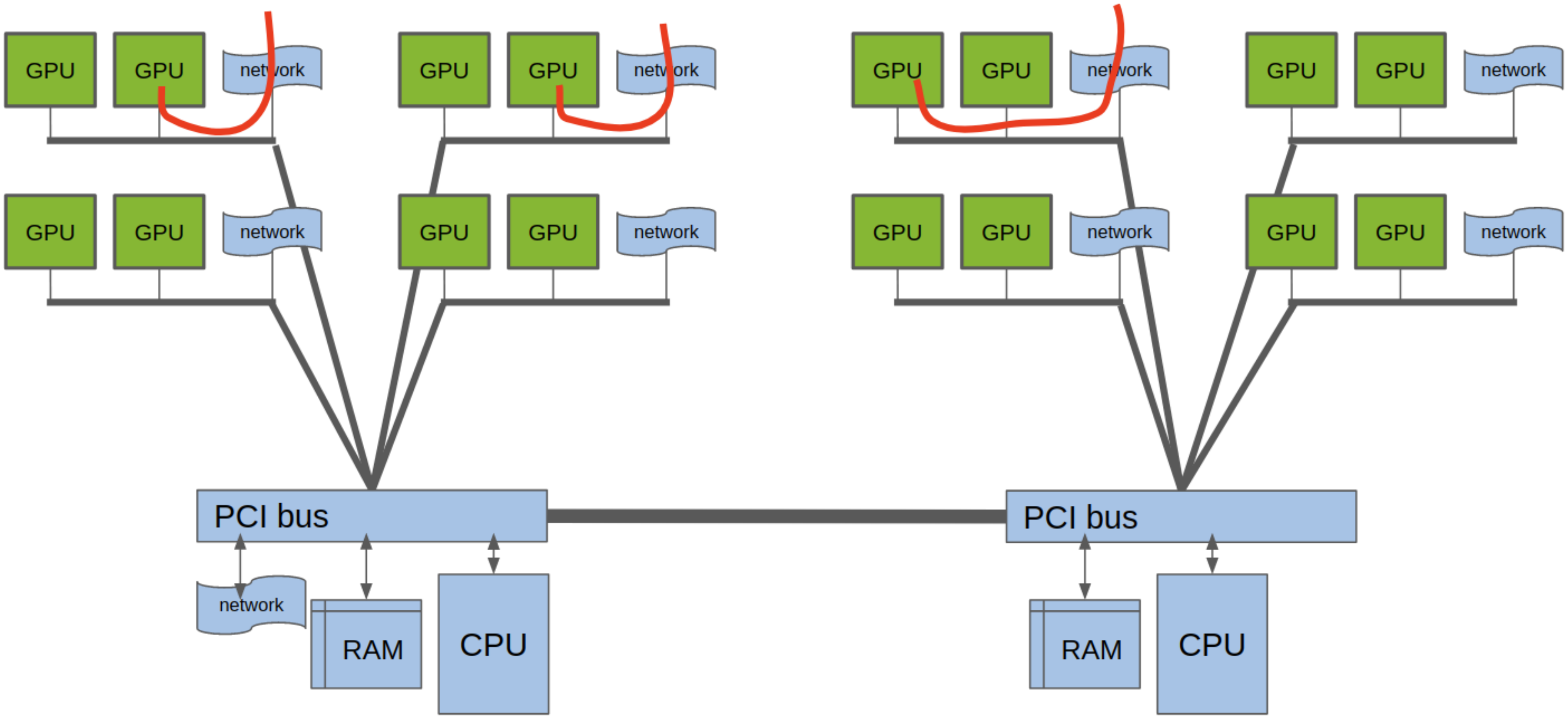
Three Tier Architecture



PCI-bus Bottleneck in Two Tier Architecture



GPUDirect Networking in Three Tier Architecture



The Tensorcom Library

The middle tier is implemented by the open source `Tensorcom` library:

- Tensorcom servers perform functions of PyTorch `DataLoader` (storage access, augmentation, batching)
- can use any PyTorch dataset or data loader, including `WebDataset`

Tensorcom-to-GPU protocols:

- ZMQ: standard networking hardware; provides high scalability and PUB/SUB
- RDMA: requires RoCE or Infiniband; provides very high data rates direct to GPU

Using the Tensorcom Library

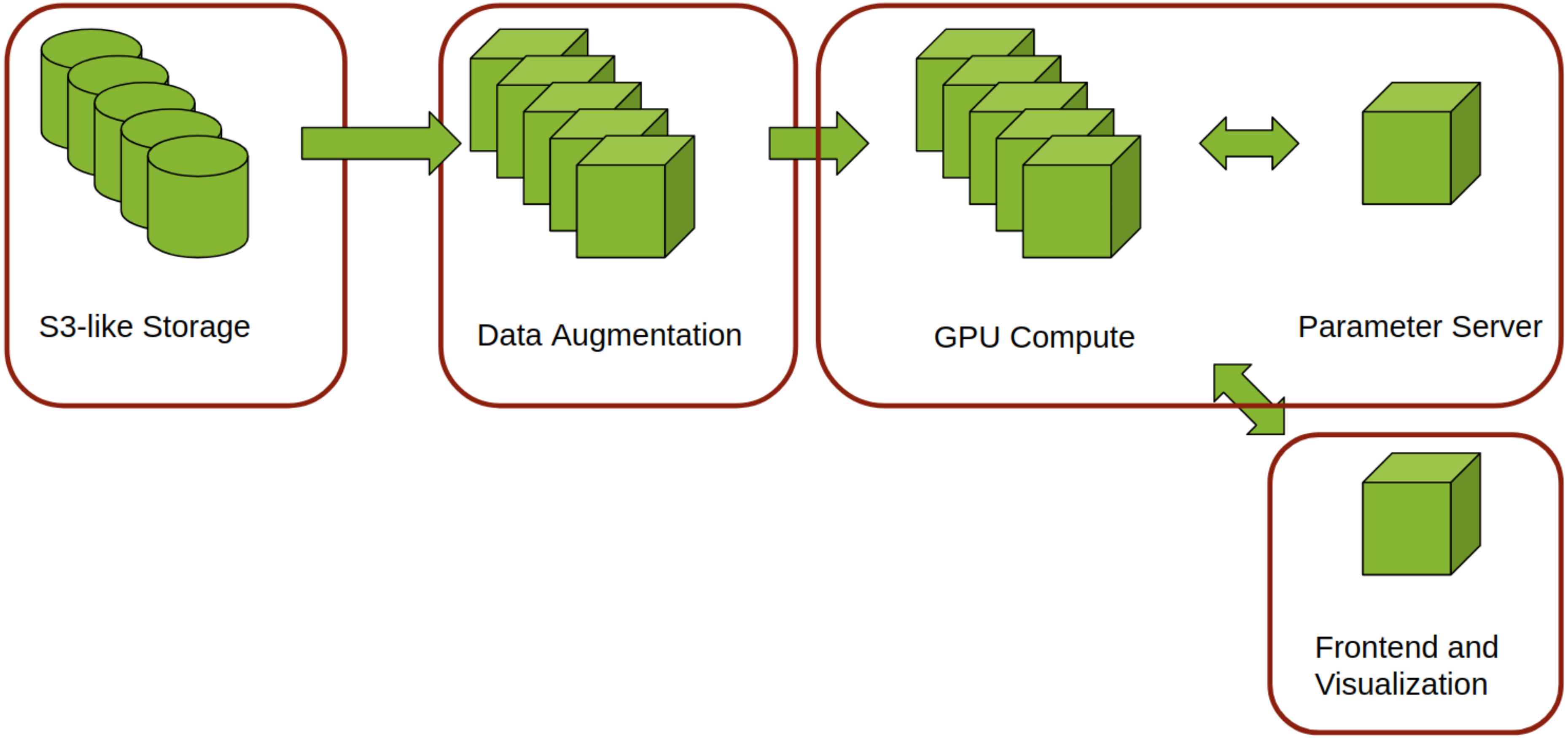
Sender:

```
dest = Connection(...)
for input_tensor, target_tensor in ...:
    dest.send(input_tensor, target_tensor)
```

Receiver:

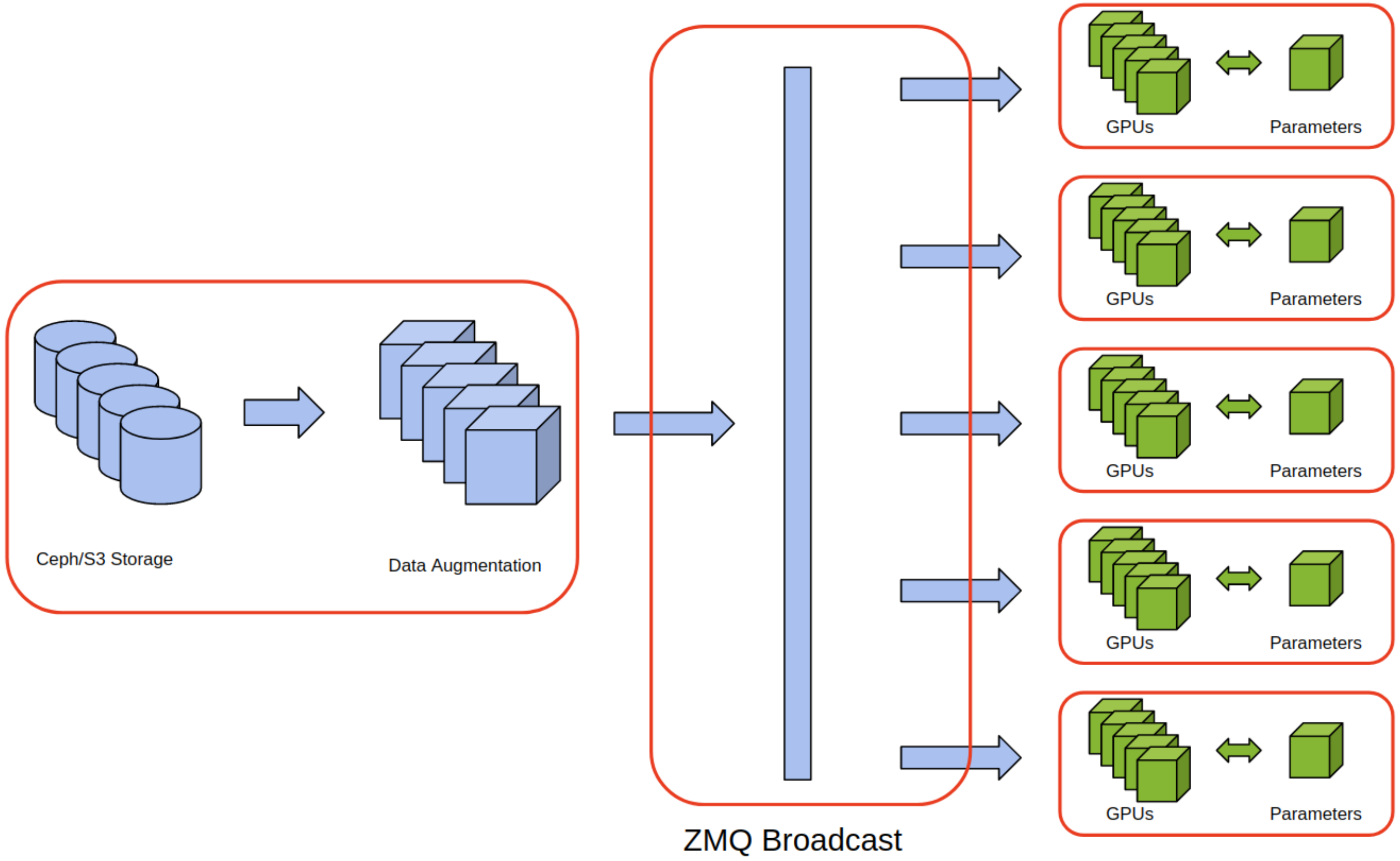
```
source = Connection(...)
for input_tensor, target_tensor in source:
    train_batch(input_tensor, target_tensor)
```

Application: Scalable CPU-Based Augmentation



Scale each component independently and dynamically (using K8s)

Application: Hyperparameter Search



Additional Benefit: Framework Independent Input Pipelines

Structure:

- file server $\longrightarrow$ tensorcom $\longrightarrow$ GPU nodes

Advantages:

- preprocessing/augmentation code becomes independent of framework
- framework-independent testing/monitoring tools
- framework integration of tensorcom connection is only a few lines of code
- ZMQ libraries + simple tensor decoder

Large Scale Training with Tensorcom

(notebook)