Innovative R&D by NTT

Case Study: Large Scale Deployment for Machine Learning with High Speed Storage

Kota Tsuyuzaki
Takeharu Eda

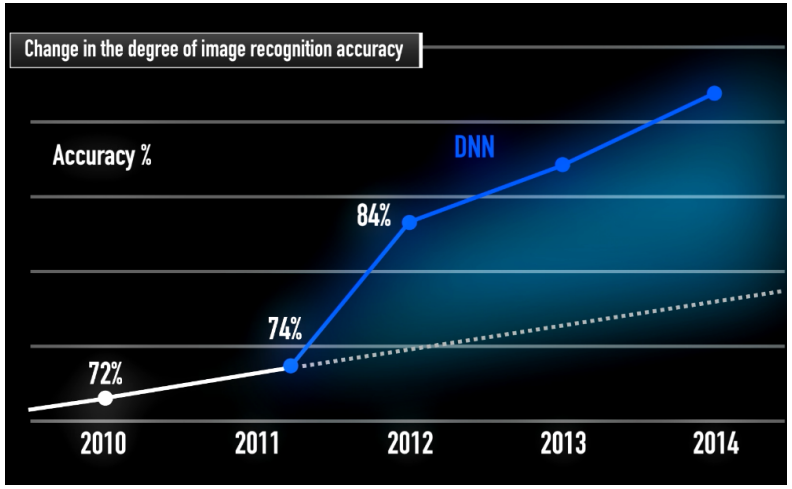
NTT Software Innovation Center

Presentation Outline

1. Introduction
2. Artificial Intelligence(AI) and Deep Learning
3. Deployments
4. Summary, Lessons, and Learns

1. Intro: AI - Artificial Intelligence

- **Business and Market is growing**
- **Machine Learning ⇒ Deep Learning**

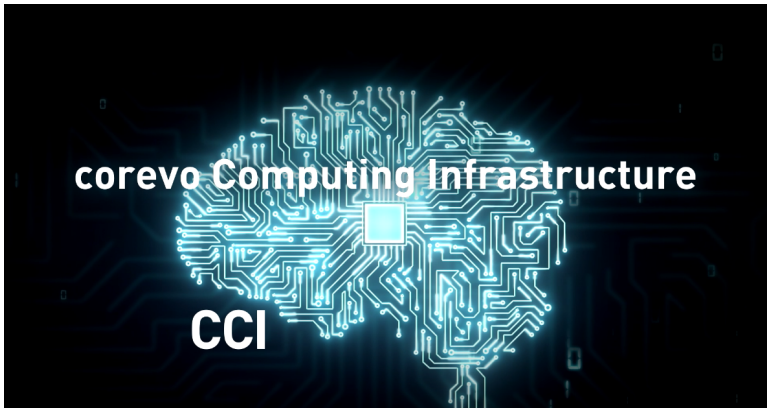


1. Intro: corevo

- **corevo is the unified brand name the NTT group has given to all of its initiatives to utilize artificial intelligence (AI) to bring about a revolution in collaboration with external enterprises, research institutes and municipalities.**

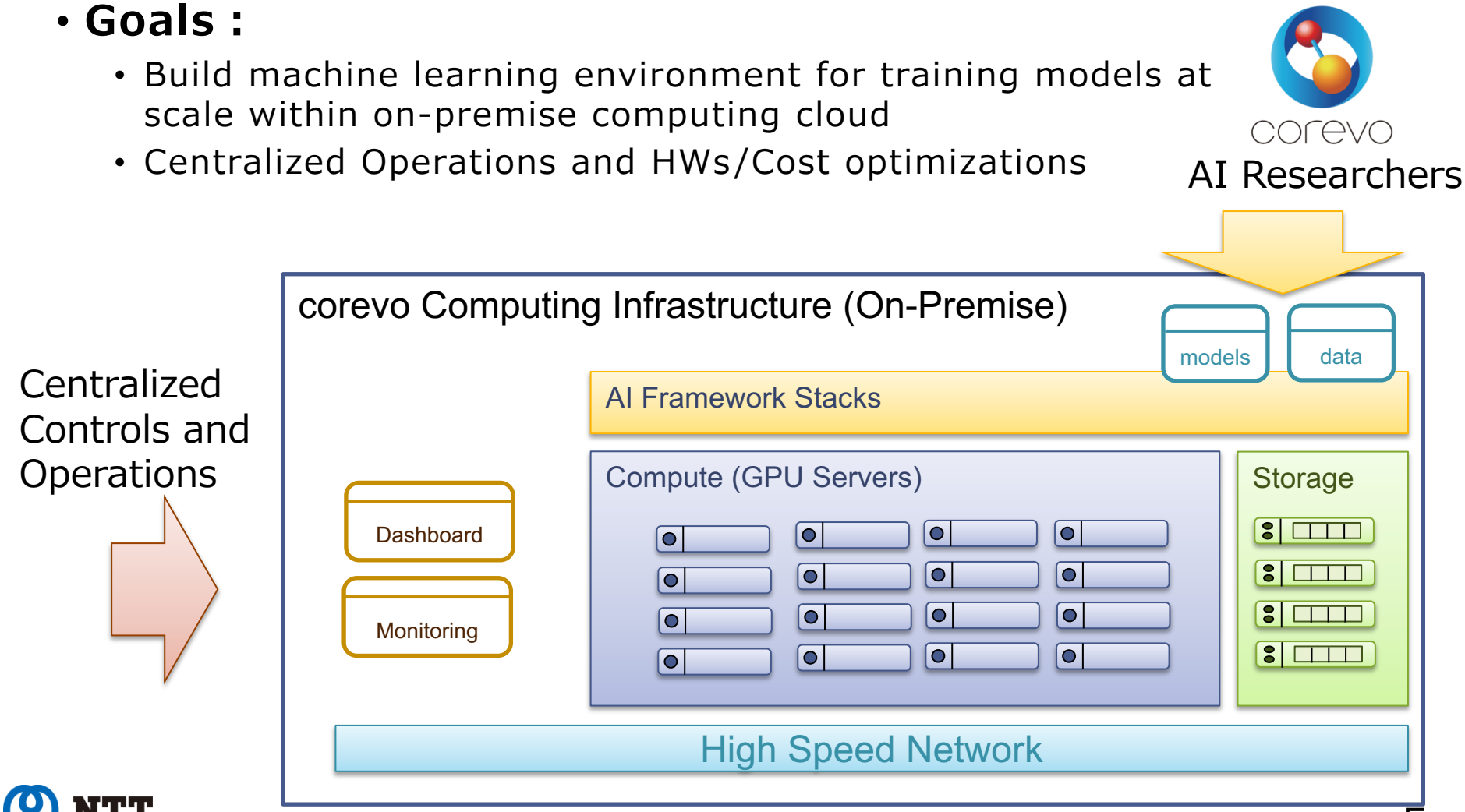
  [http://www.ntt.co.jp/corevo/e/]

corevo



1. Intro: cci - corevo Computing Infrastructure

- **R&D environment for corevo AI Researchers**
- **Goals :**
  - Build machine learning environment for training models at scale within on-premise computing cloud
  - Centralized Operations and HWs/Cost optimizations



1. Intro: First Step and Members

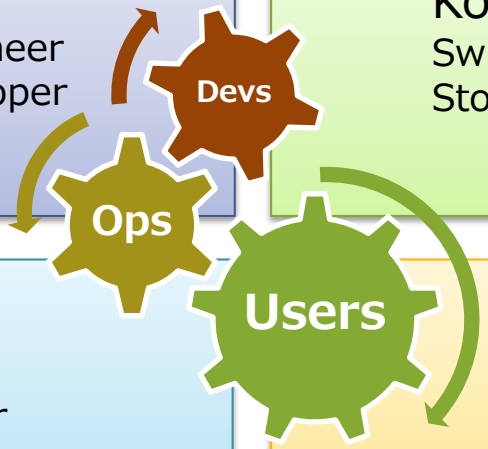
- **Starts from Scrum**

AI Researchers, Framework Design

Takeharu Eda
AI Research Engineer
Application Developer
1 more member

Scale-Out Storage Design

Kota Tsuyuzaki
Swift Core Developer
Storlets PTL



Facilities, Network Design

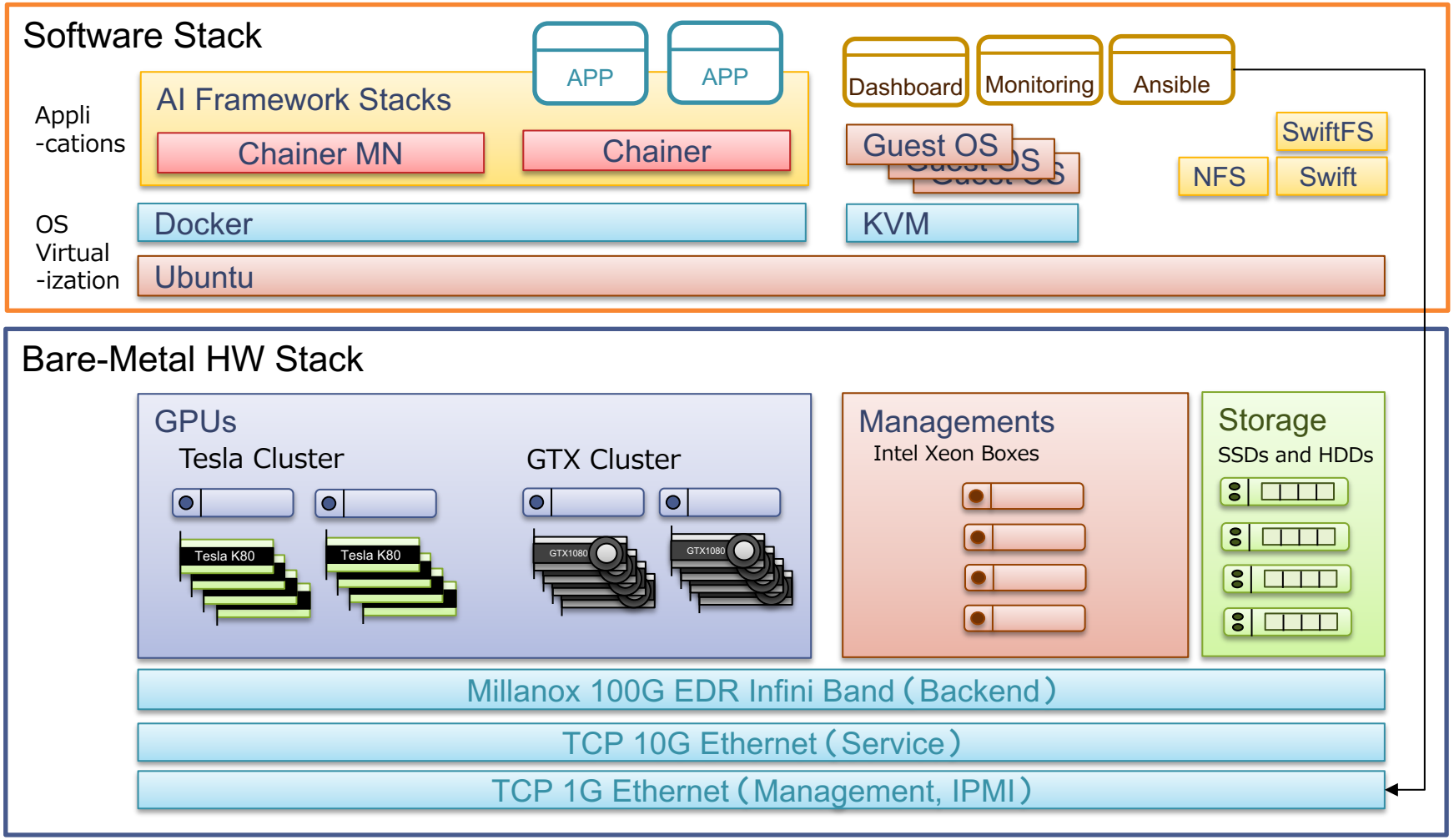
Kengo Okitsu
Research Engineer

1 more member

Monitoring/ and Operation Design

2 more members

1. Intro: Stack Design Overview

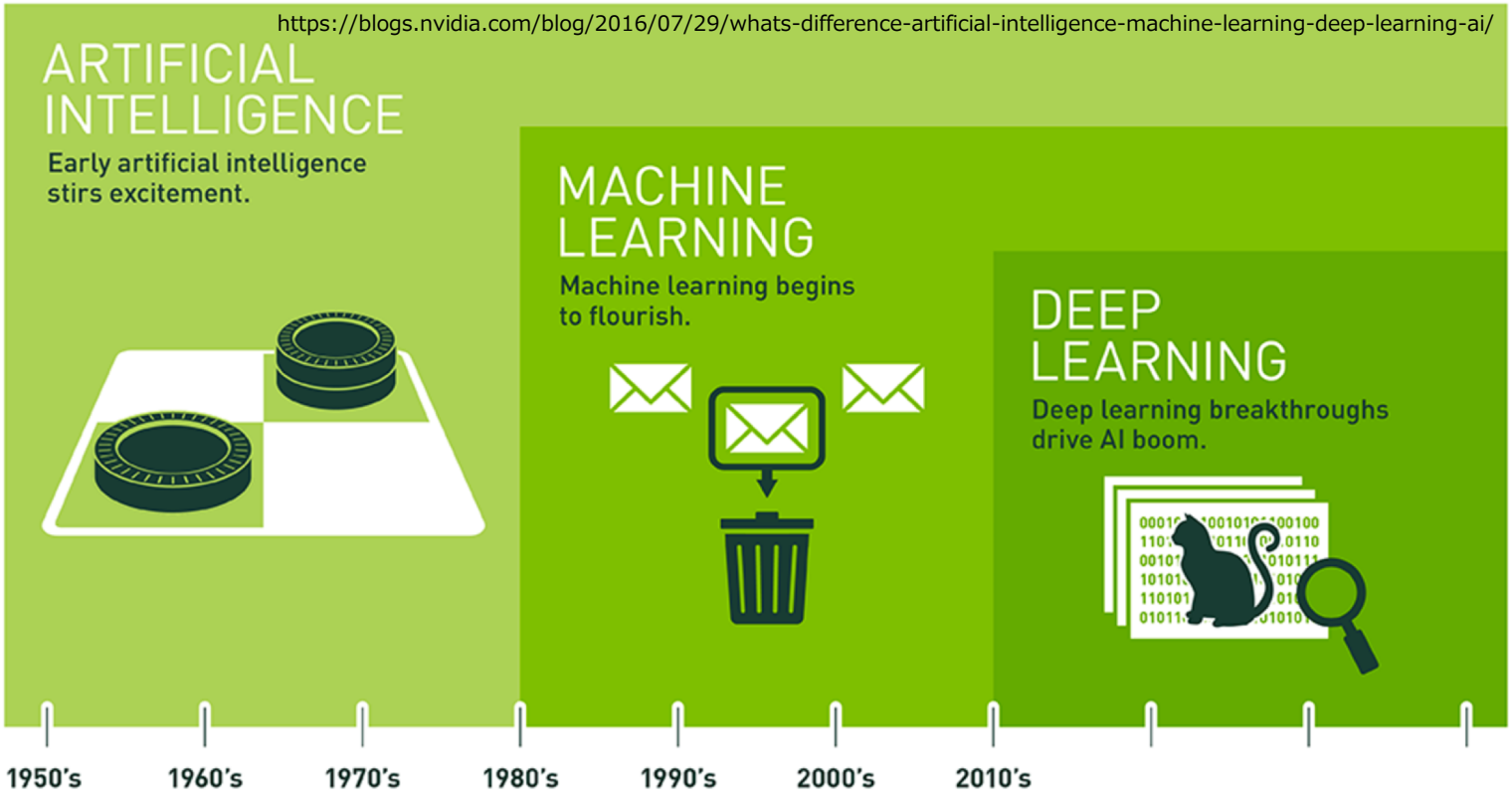


User’s perspective on the machine learning environment

2. AI and DL: AI Section Outline

- **Motivation – why deep learning?**
  - AI, Deep learning, GPU
- **Framework/Method selection**
  - Model or Data parallelism
  - Frameworks choices: TensorFlow, Chainer, Pytorch, etc.
- **Training script/Dockerfile**
  - MPI-based data parallelism is easy to implement
- **Evaluation**
  - Single node vs. Multi nodes
  - Bare metal vs. over Docker

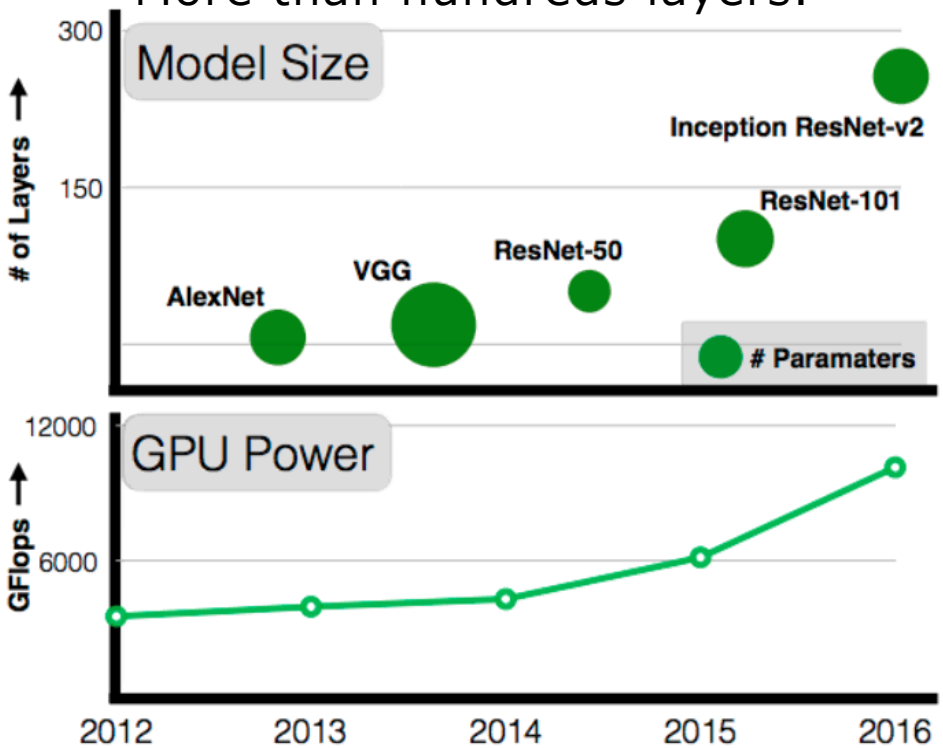
2. AI and DL: Why deep learning?



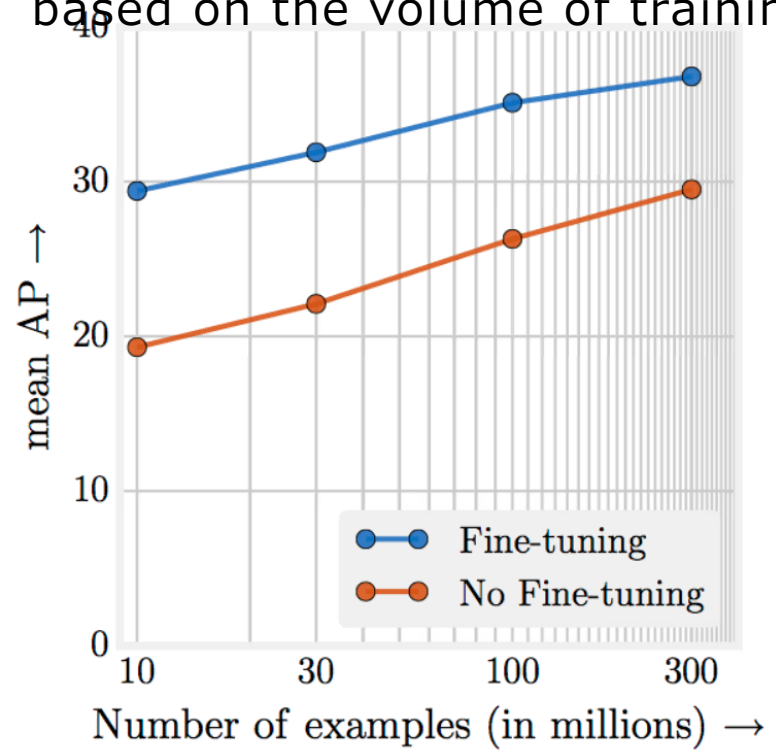
What make a big leap from traditional machine learning to deep learning are big data and computing resources.

2. AI and DL: Training deep models with big data

More than hundreds layers.



Accuracy increases logarithmically based on the volume of training data.

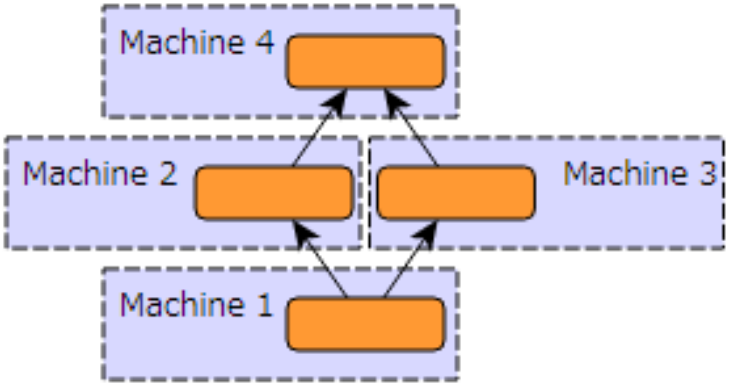


How to train deep models in parallel with many computing resources (GPUs) ?

https://ai.googleblog.com/2017/07/revisiting-unreasonable-effectiveness.html

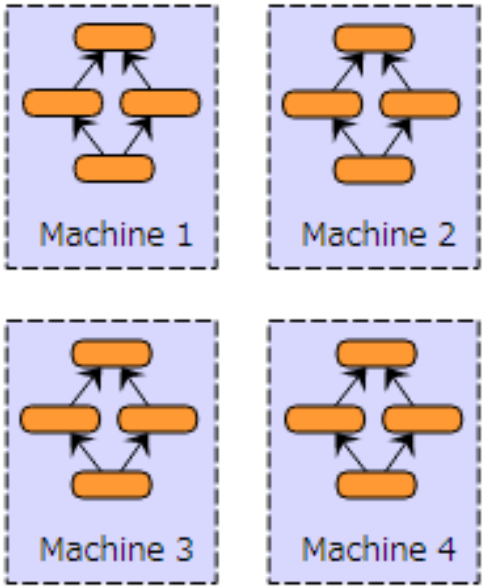
2. AI and DL: Model vs. data parallelism

Model Parallelism



- Divide a model into multiple components and put them in several machines/gpus.
- It was popular when the RAM of GPU is small.

Data Parallelism



- All machines/gpus train the same model with different data, and the models are averaged and synced.
- Fast and efficient for training very large dataset.

We chose data parallelism considering the dataset volume and model size.

http://engineering.skymind.io/distributed-deep-learning-part-1-an-introduction-to-distributed-training-of-neural-networks

2. AI and DL: Deep learning frameworks

-	TensorFlow	MXNet	ChainerMN (Chainer)	pytorch	PaddlePaddle
Distribution method	Data / Model	Data / Model	Data / Model	Data / Model	Data
Synchronization	Sync / Async	Sync / Async	Sync	Sync	Sync/Async
Parameter server	✓	✓	×	×	✓
Interconnect	gRPC (MPI), NCCL(horovod)	MPI, SSH, SGE, YARN	MPI (NCCL)	TCP, gloo, MPI	OpenMPI, Kubernetes, Fabric
GPU Direct	P2P RDMA	P2P RDMA	P2P (MPI) RDMA (MPI)	P2P RDMA	P2P (MPI) RDMA (MPI)
Distributed training	○	○	○	○	○

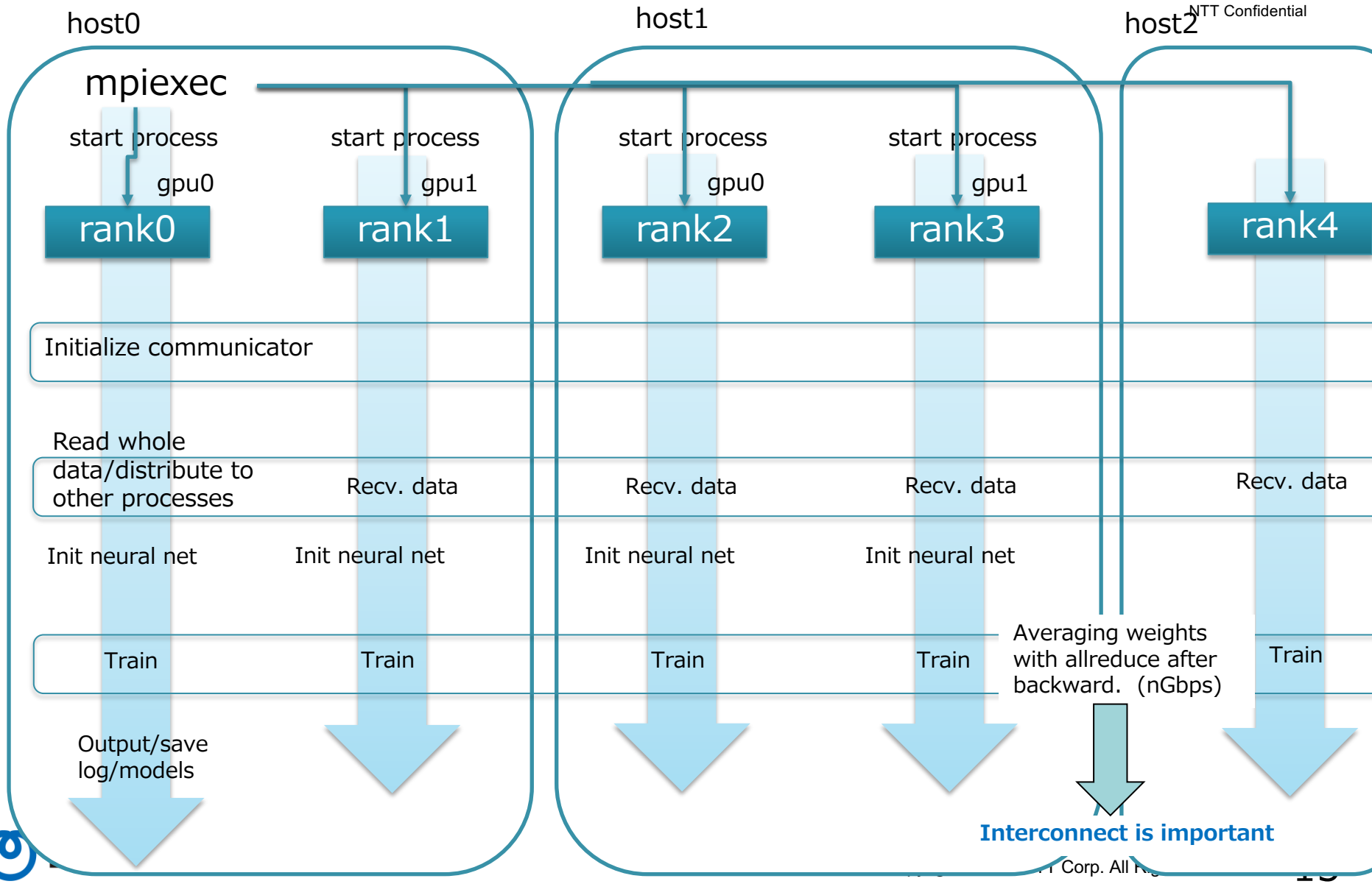
The progress of frameworks is very high.
ChainerMN supported data parallelism with MPI at an early date over high-speed interconnect (infiniband, and NCCL2).

2. AI and DL: ChainerMN-based training script

- **Easy to run data parallel training script with mpiexec**
  - python3 train_mnist.py

  ⇒
  - mpiexec -n 4 python3 train_mnist_mn.py
- **Minimal modification on original script for multi GPU training**
  1. Init communicator and set device id
  2. Set multi node optimizer/evaluater
  3. Distribute data
  4. Start training

2. AI and DL: Processing flow of training script

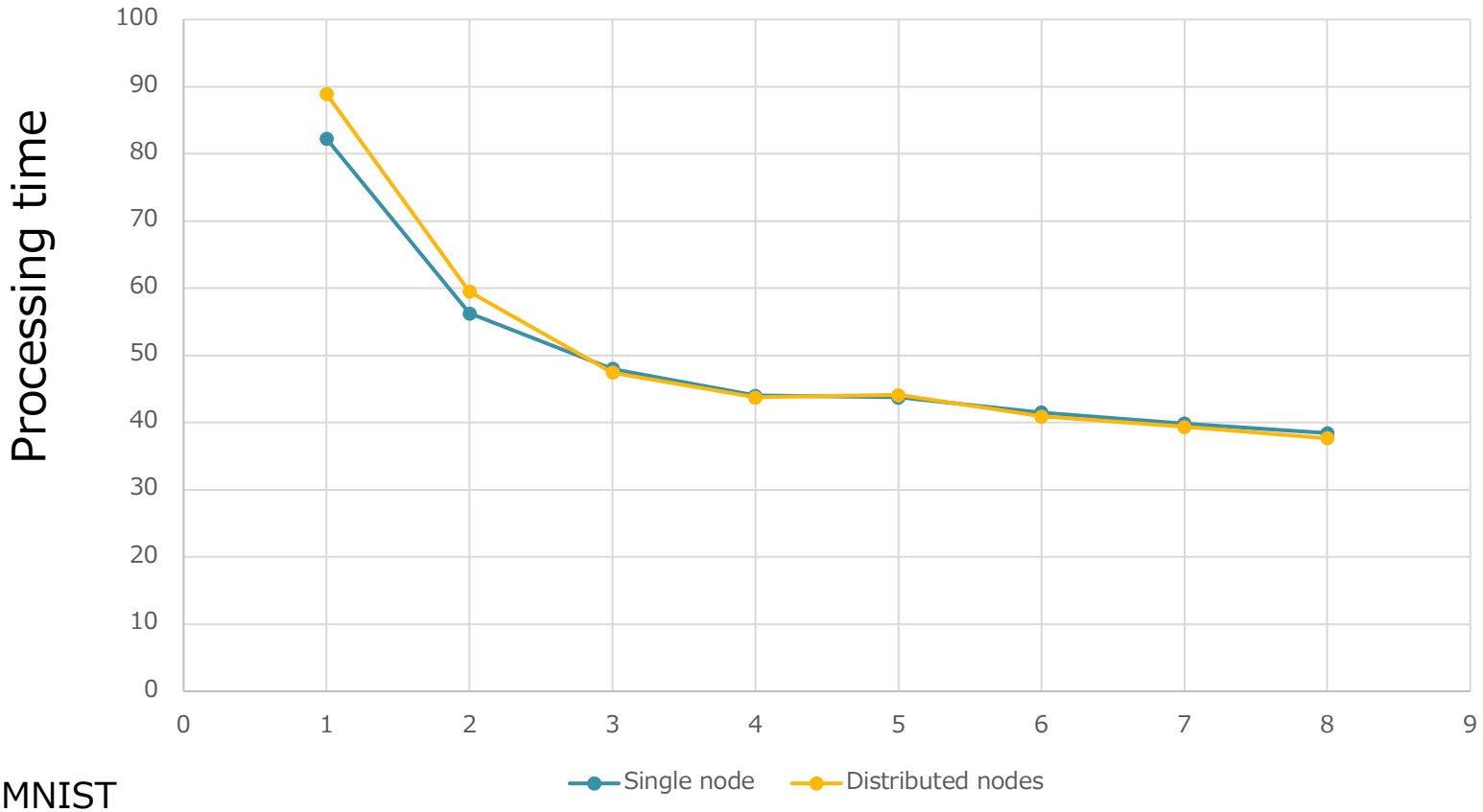


2. AI and DL: Containerization

- **Customized image based on official nvidia docker image**
  - Cuda:9.0, cuDNN:7
  - OpenMPI/SSH inside Docker container
  - NCCL:2.1.2
  - MLNX_OFED: 4.1-1.0.2.0

- **Steps of distributed training**
  1. Start all of docker containers
  2. Kick the training script at master node
     - Master node -> worker nodes
  3. Stops all of containers when training finished.

2. AI and DL: Evaluation - Single node vs. Multi nodes

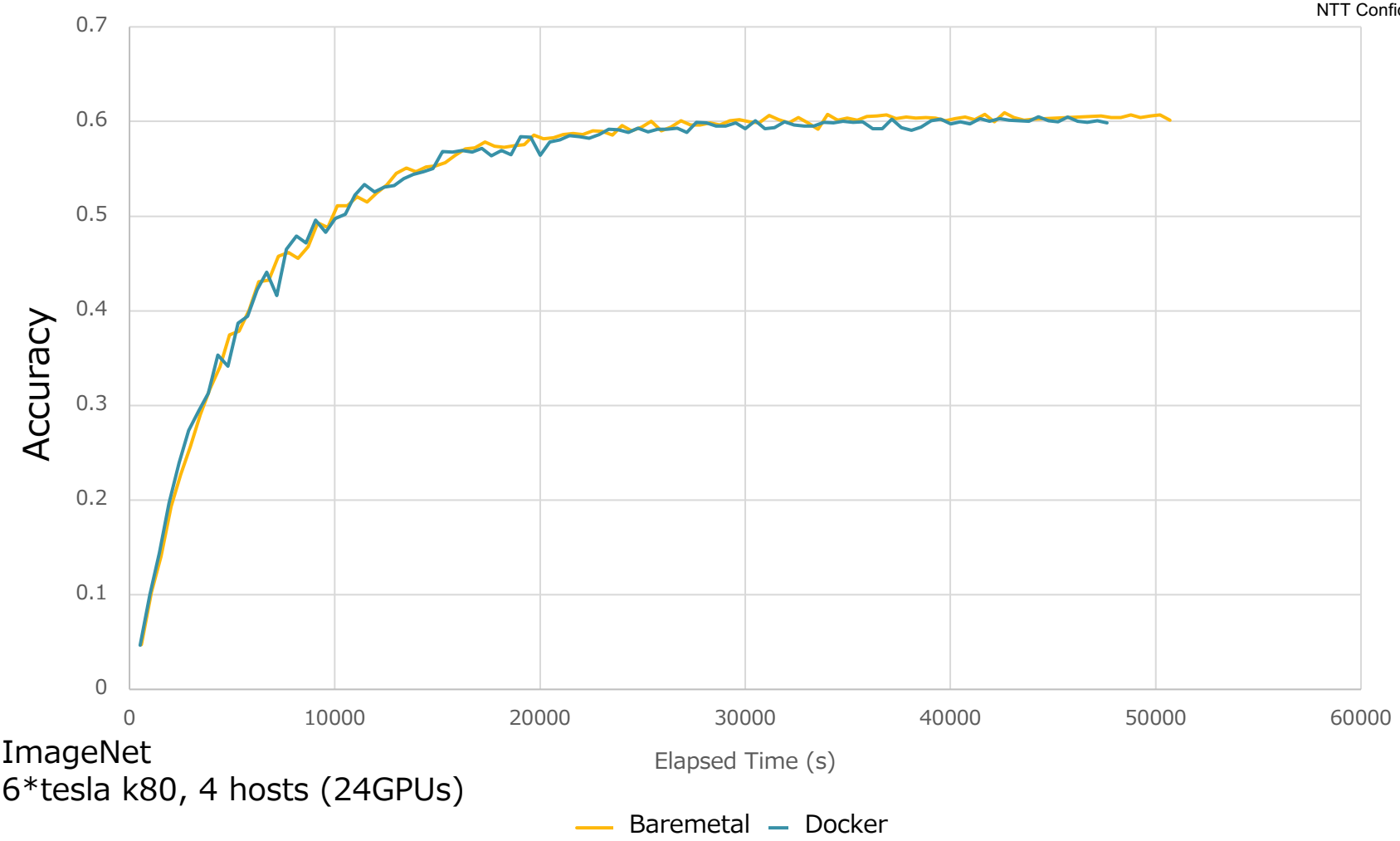


MNIST
8*tesla K80
2 hosts

The more GPUs(=jobs), the less overhead in multi nodes

2. AI and DL: Evaluation - Bare metal vs. Docker

NTT Confidential



ImageNet
6*tesla k80, 4 hosts (24GPUs)

No overhead by containerization

2. AI and DL: Summary

- **Evaluated data-parallel deep learning performance in our environment.**
  - Infiniband EDR/Docker/ChainerMN
- **Minimal changes to original standalone training script**
- **Easily scalable to multi nodes/GPUs.**
- **Almost no overhead observed by containerlization in our experiments.**

Design and implementation of our environment will be presented in the following slides.

3. Deploy: Deployments Section Outlines

- **AI, GPU Environments Design**
  - Requirements
  - Network Architecture
  - GPU Monitoring
- **Storage Environments Design**
  - Requirements
  - Design
- **Summary**

3. Deploy: NW and GPU Design Requirements

- **Too many inter-GPU communications (intra- and inter-node) in ChainerMN training**

  ⇒ **High bandwidth and low latency network architecture**

- **High GPU temperature slowdown or shutdown GPU clock**

  ⇒ **Monitoring GPU temperature and clock**

3. Deploy: Network architecture overview

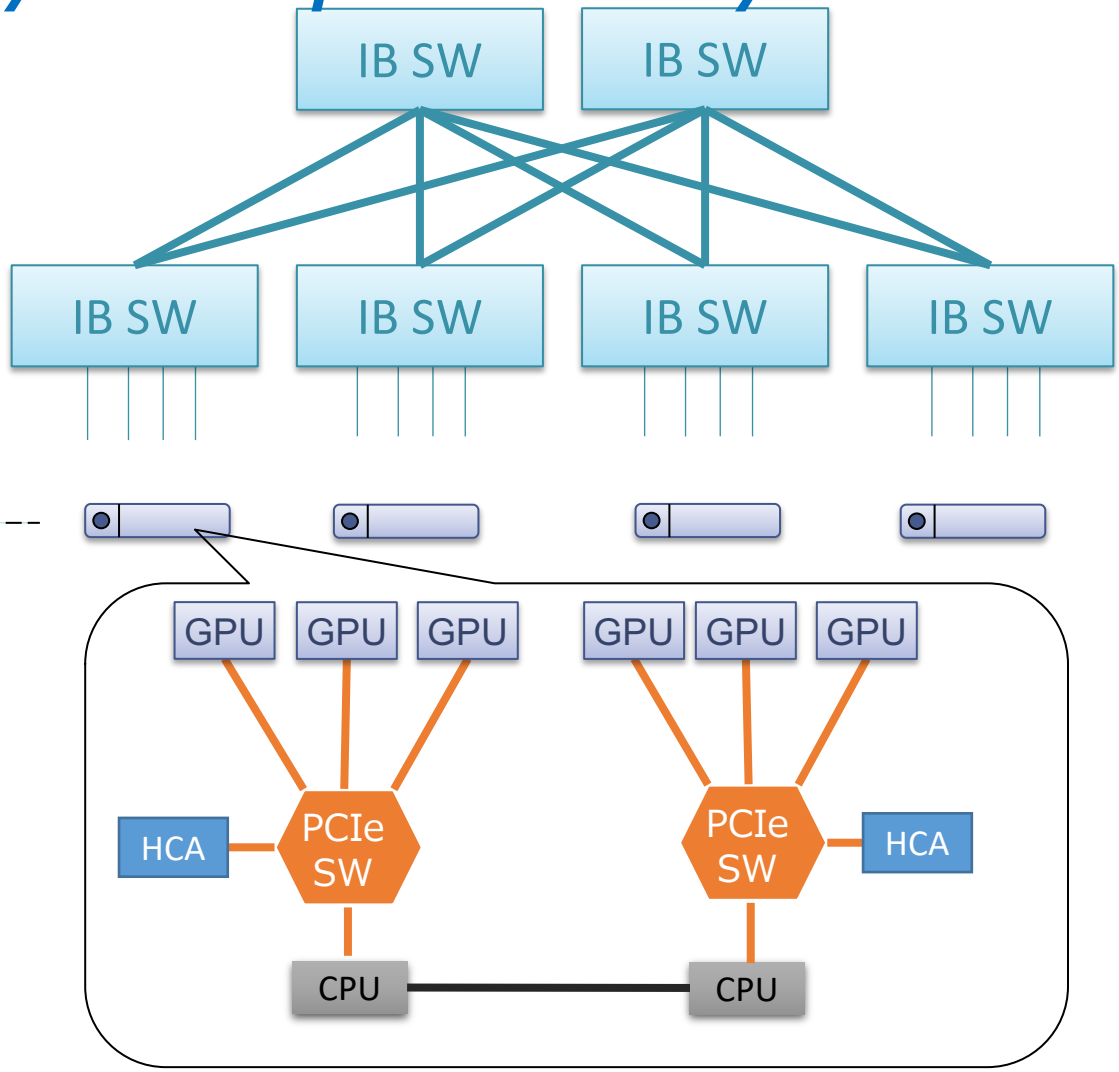
Network topology techniques on 2 layers

Non-blocking InfiniBand (IB) network

outside server

inside server

PCIe topology aware device placement



3. Deploy: Non-blocking InfiniBand Network

- **InfiniBand**
  - Mellanox 36-ports EDR 100Gbps Switch (SB7780)

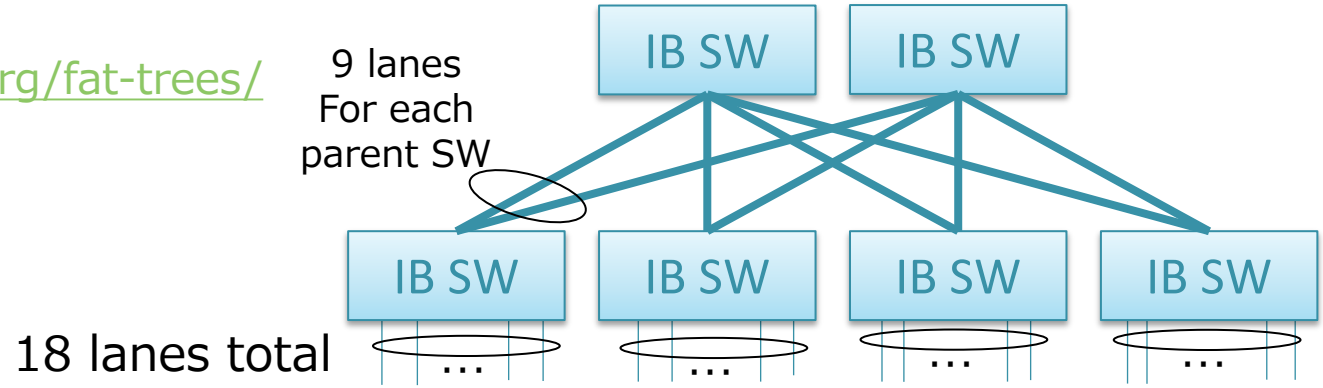
  -> Low latency(100ns~), high bandwidth(100Gbps)

3. Deploy: Non-blocking InfiniBand Network

- **Fat-Tree topology**
  - Bandwidth assurance in bust traffic case
  - the number of downlinks equals to the number of uplinks (Non-blocking network)

Full Link bandwidth for every node pair

http://clusterdesign.org/fat-trees/

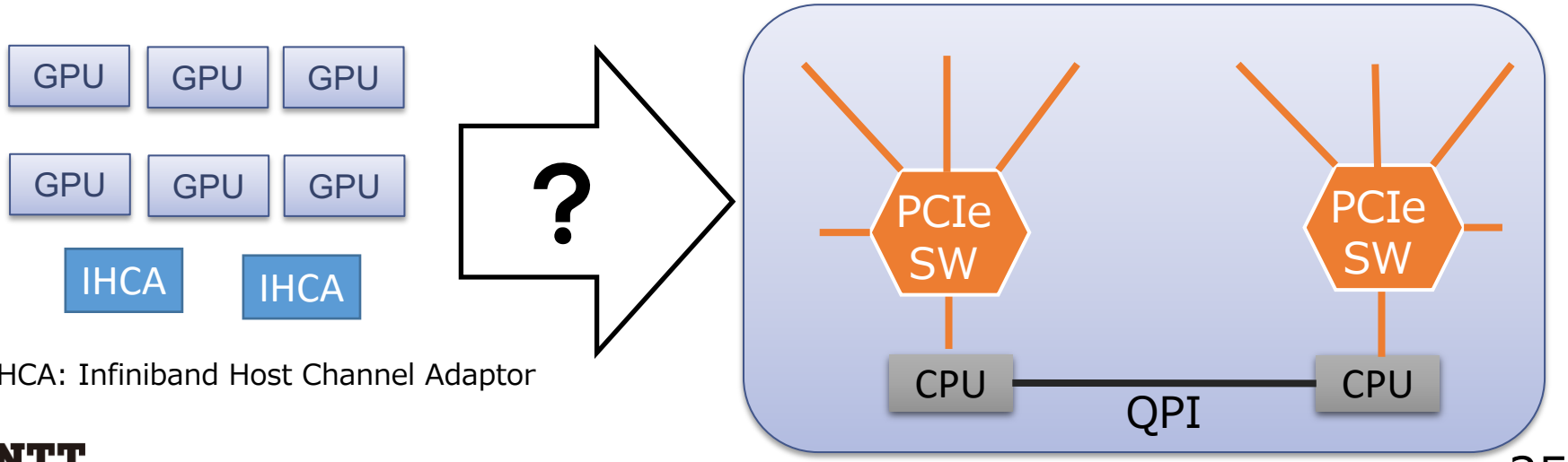


3. Deploy: Architectural Issue inside Server

- **Preferred Server Type**
  - High GPU density: Many PCIe slots per server
- **Server Specification**
  - Dual-Socket Intel Xeon CPU E5-2697 v3
  - Generation 3, 16 lanes (15.75 GB/s) PCIe x 4 slots / CPU
  - QPI (19.2 GB/s) connection between CPUs
- **PCIe topology inside Server**

  → How Many GPU per PCIe?

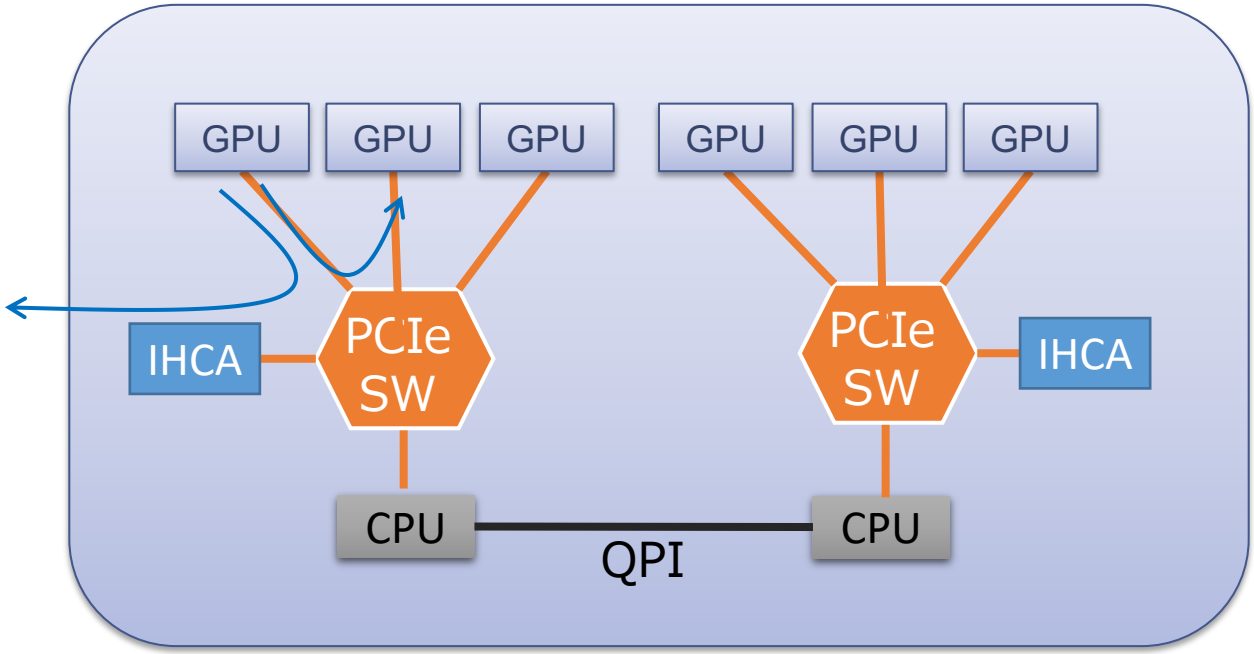
  → How Many Infiniband NIC per PCIe?



IHCA: Infiniband Host Channel Adaptor

3. Deploy: Our Design

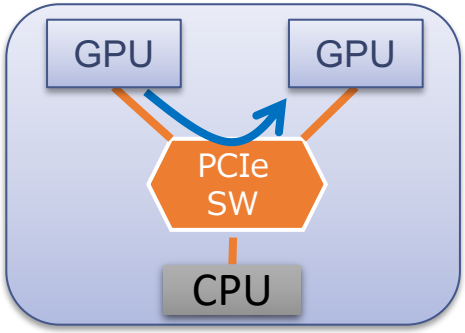
- **3 GPUs and 1 IHCA per PCIe Swicth and CPU**
  - → **No CPU MEM and QPI transfer at GPU communication**



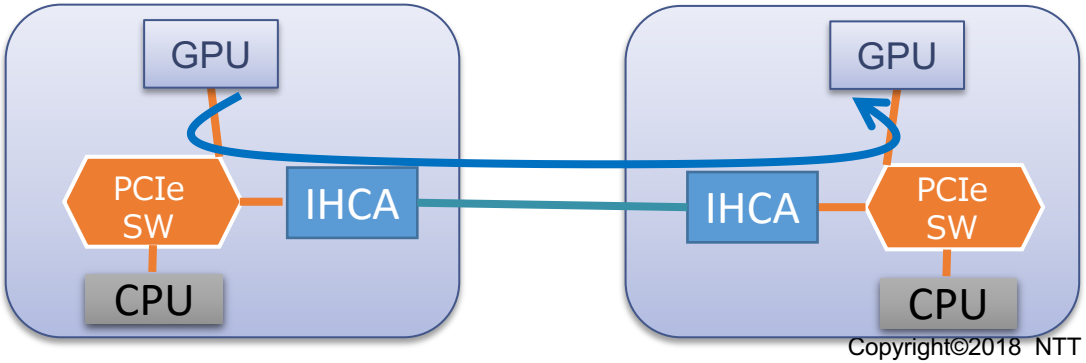
Low latency for inter-GPU communications

3. Deploy: Internal PCIe topology Techniques

- **Direct Memory Access (DMA)**
  - Bypass “GPU and CPU data transfer”
  - GPUs must be connected on the same PCIe root complex

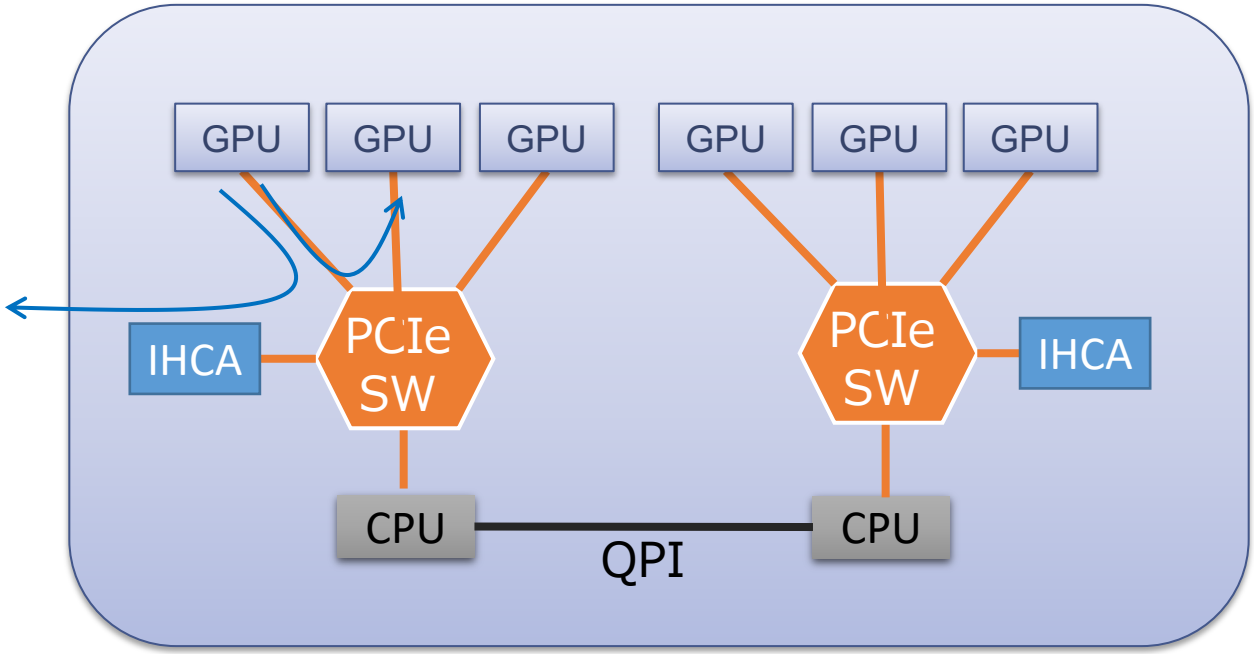


- **GPU Direct RDMA**
  - Bypass “GPU and CPU data transfer” at Multi nodes
  - Mellanox HCA (Host Chanel Adaptor) + NVIDIA GPU required



3. Deploy: Our Design

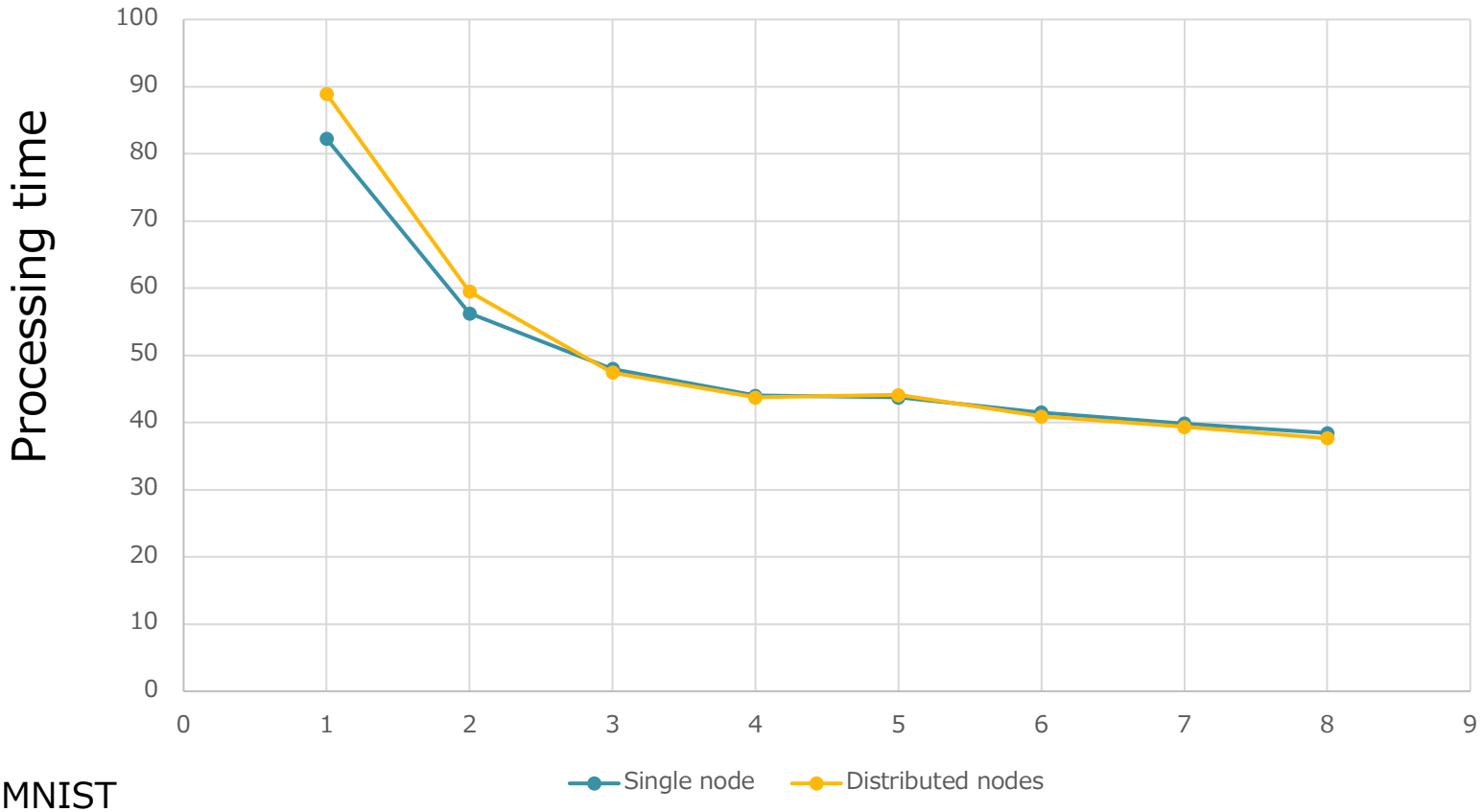
- **3 GPUs and 1 IHCA per PCIe Swicth and CPU**
  - → **No CPU MEM and QPI transfer at GPU communication**



Low latency for inter-GPU communications

REPRINT from page 17

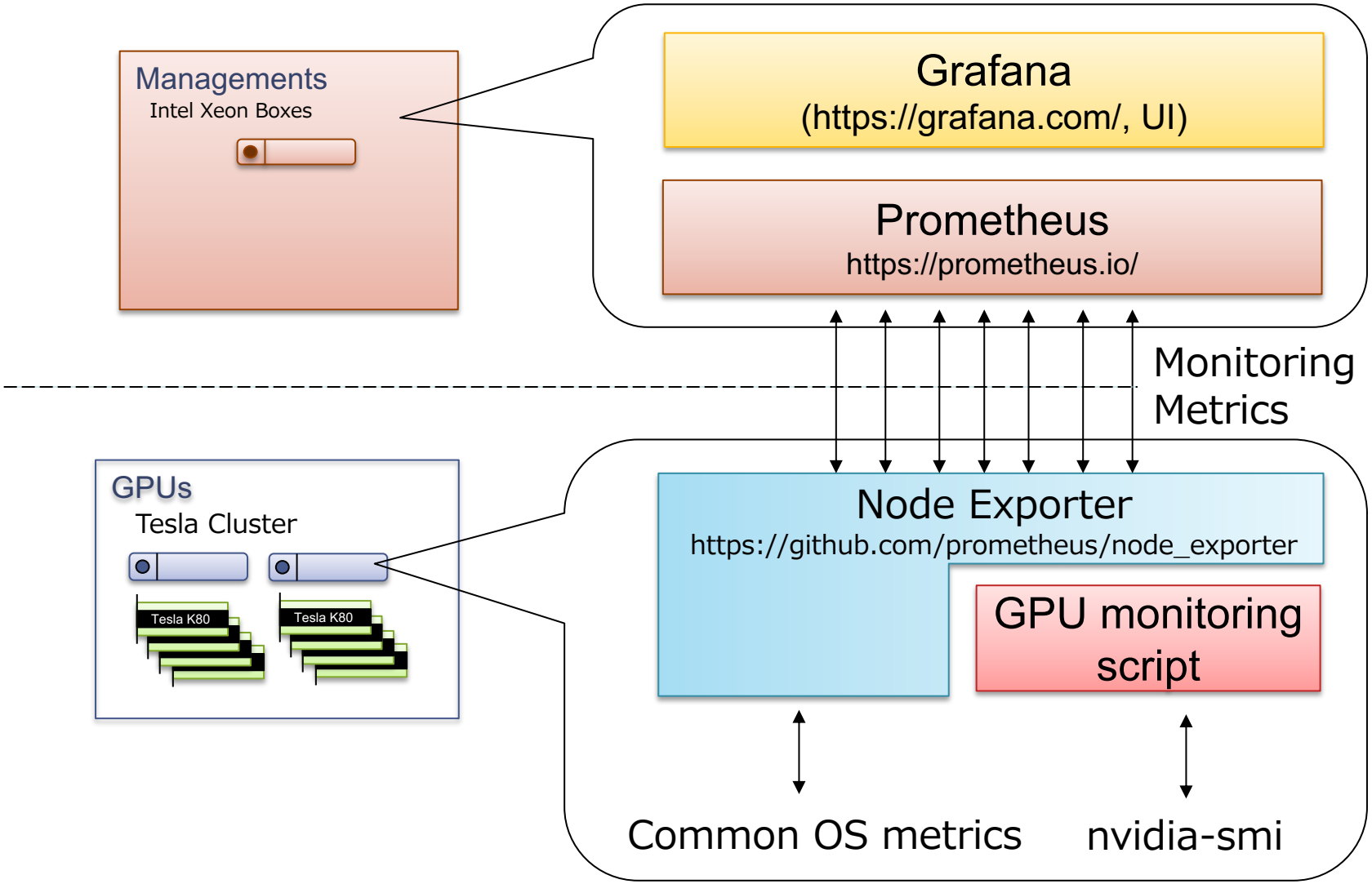
3. Deploy: Evaluation: Single node vs. Multi nodes



MNIST
8*tesla K80
2 hosts

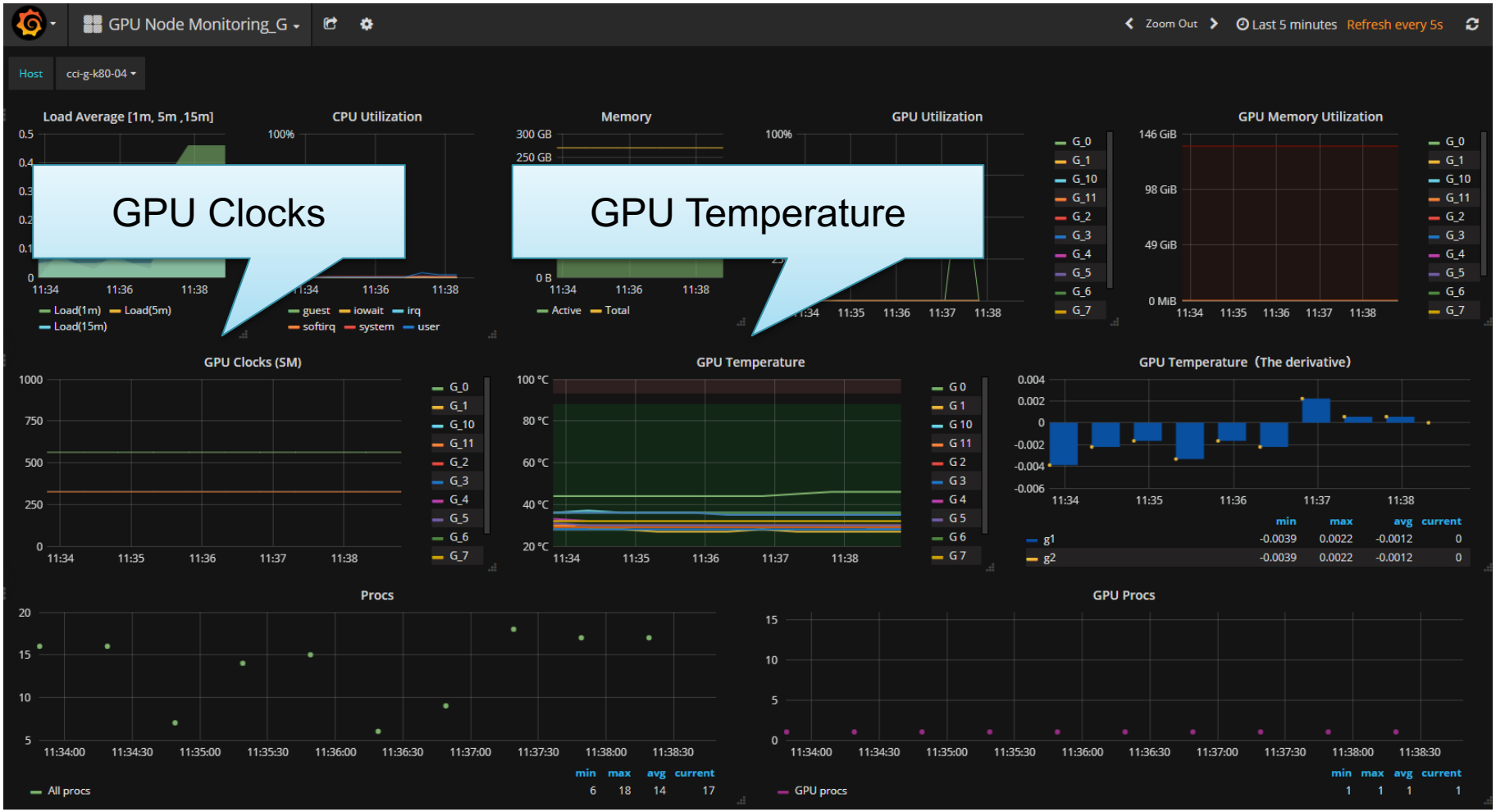
The more GPUs(=jobs), the less overhead in multi nodes

3. Deploy: Monitoring Software Stack

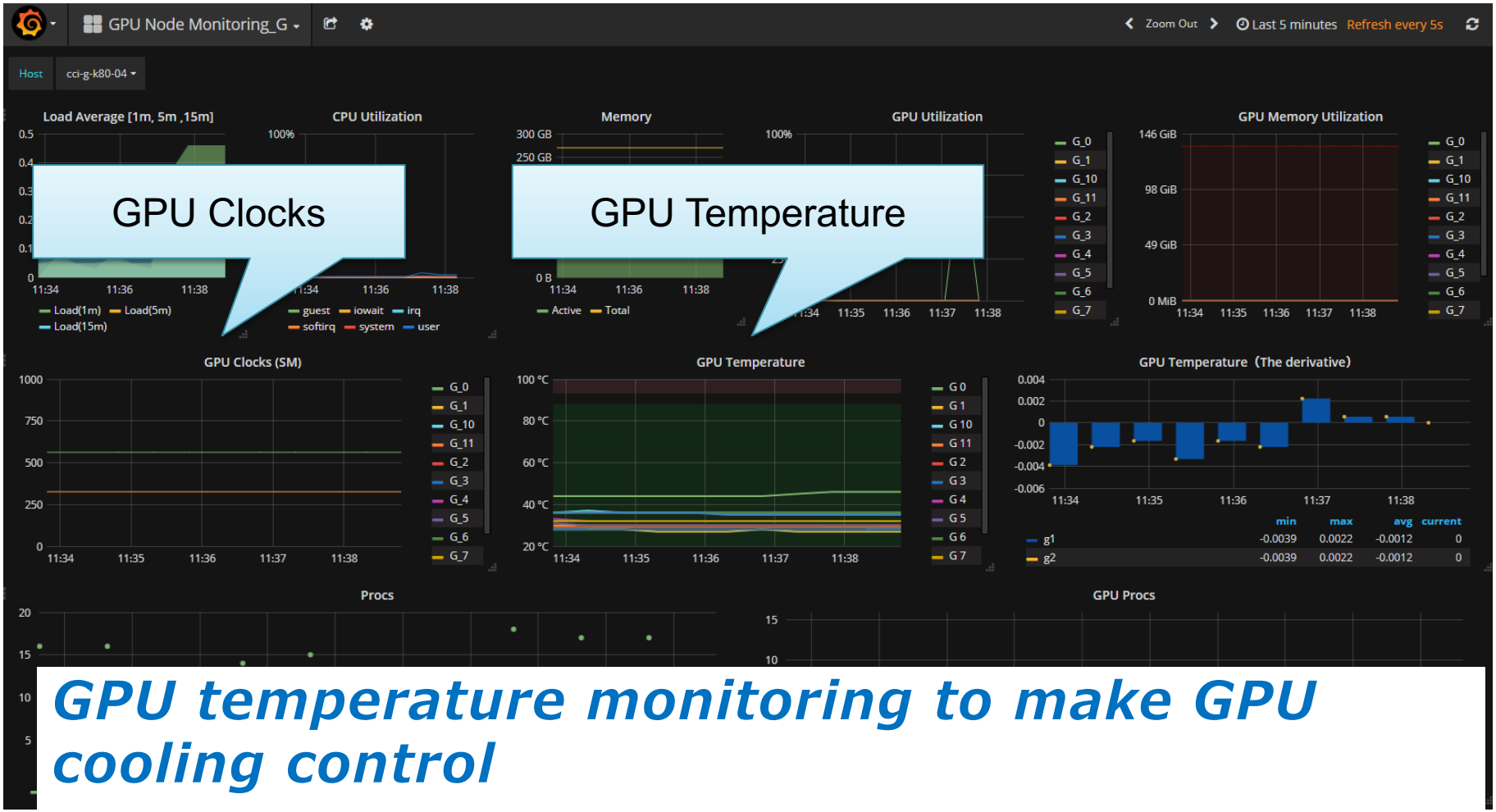


3. Deploy: Monitoring - GPU Server View

3. Deploy: Monitoring - GPU Server View



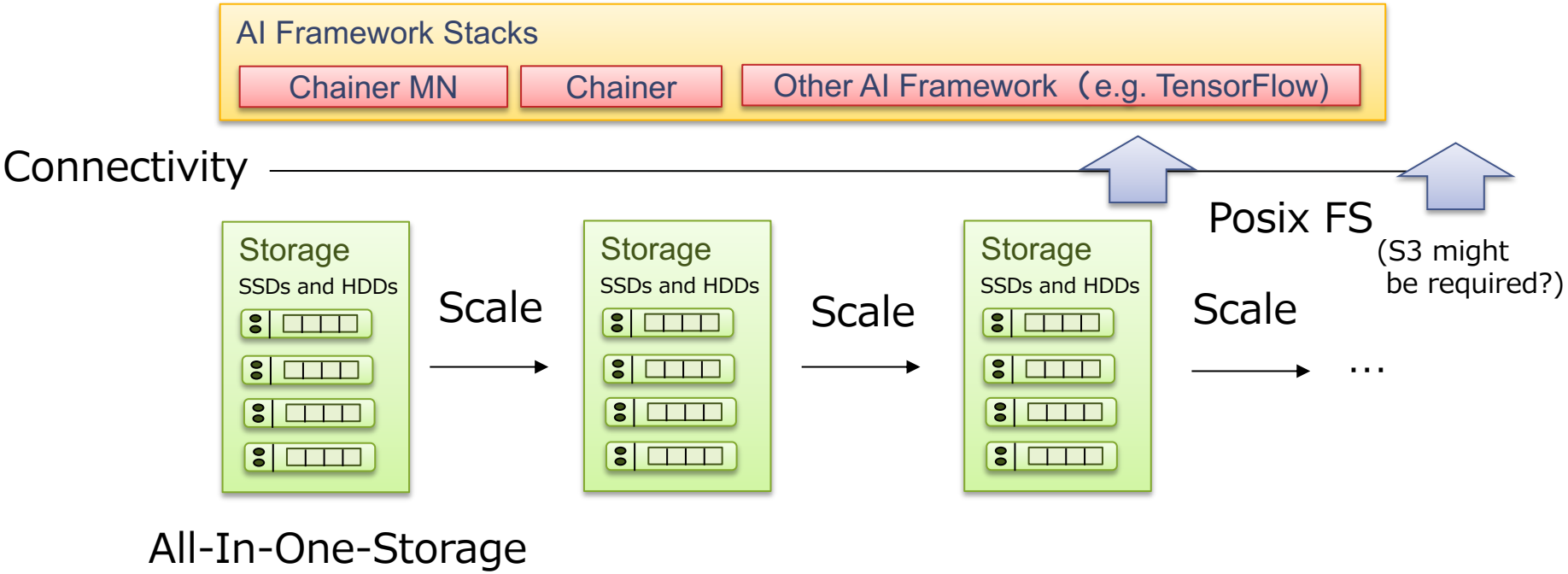
3. Deploy: Monitoring - GPU Server View



GPU temperature monitoring to make GPU cooling control

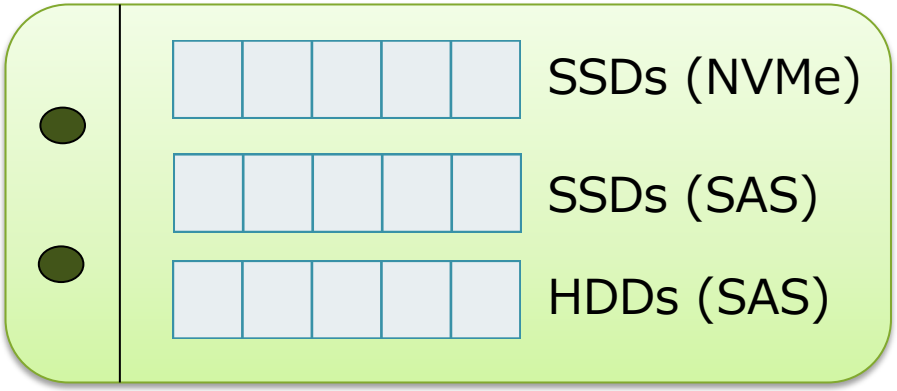
3. Deploy: Storage Design Requirements

- **Small Start – Starts from Scrum**
- **Scalability – Big Data and Parallel Reads**
- **Easy Adaption to AI Frameworks**
  - Mount-able in AI computing servers as POSIX FS



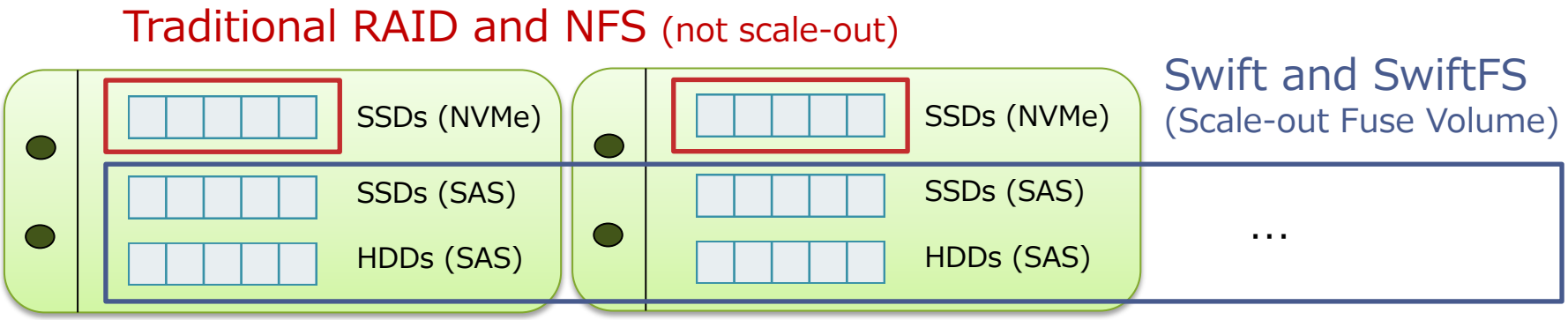
3. Deploy: Another Consideration on Our Constraints

- **Various Hardware Volumes**
  - Hardware reused from another department



3. Deploy: Design - Our Storage Stack Steps

- **Take 2 types approaches**
  1. NFS servers with RAID NVMe volumes for /home
  2. SwiftFS on top of OpenStack Swift for test data sharing



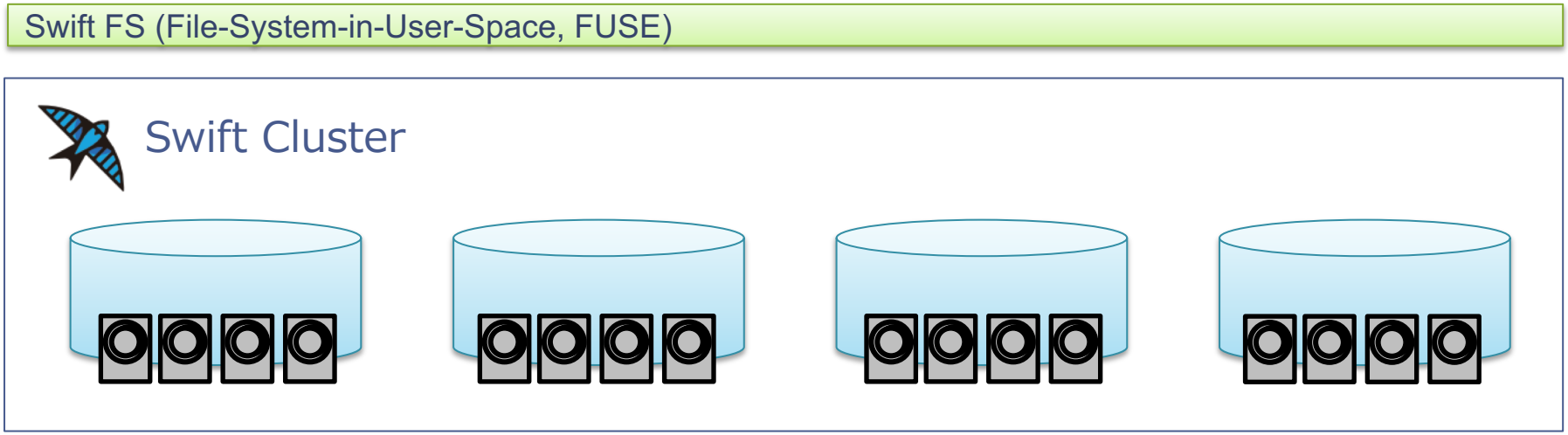
3. Deploy: OpenStack Swift and SwiftFS

OpenStack Swift: Distributed Object Storage

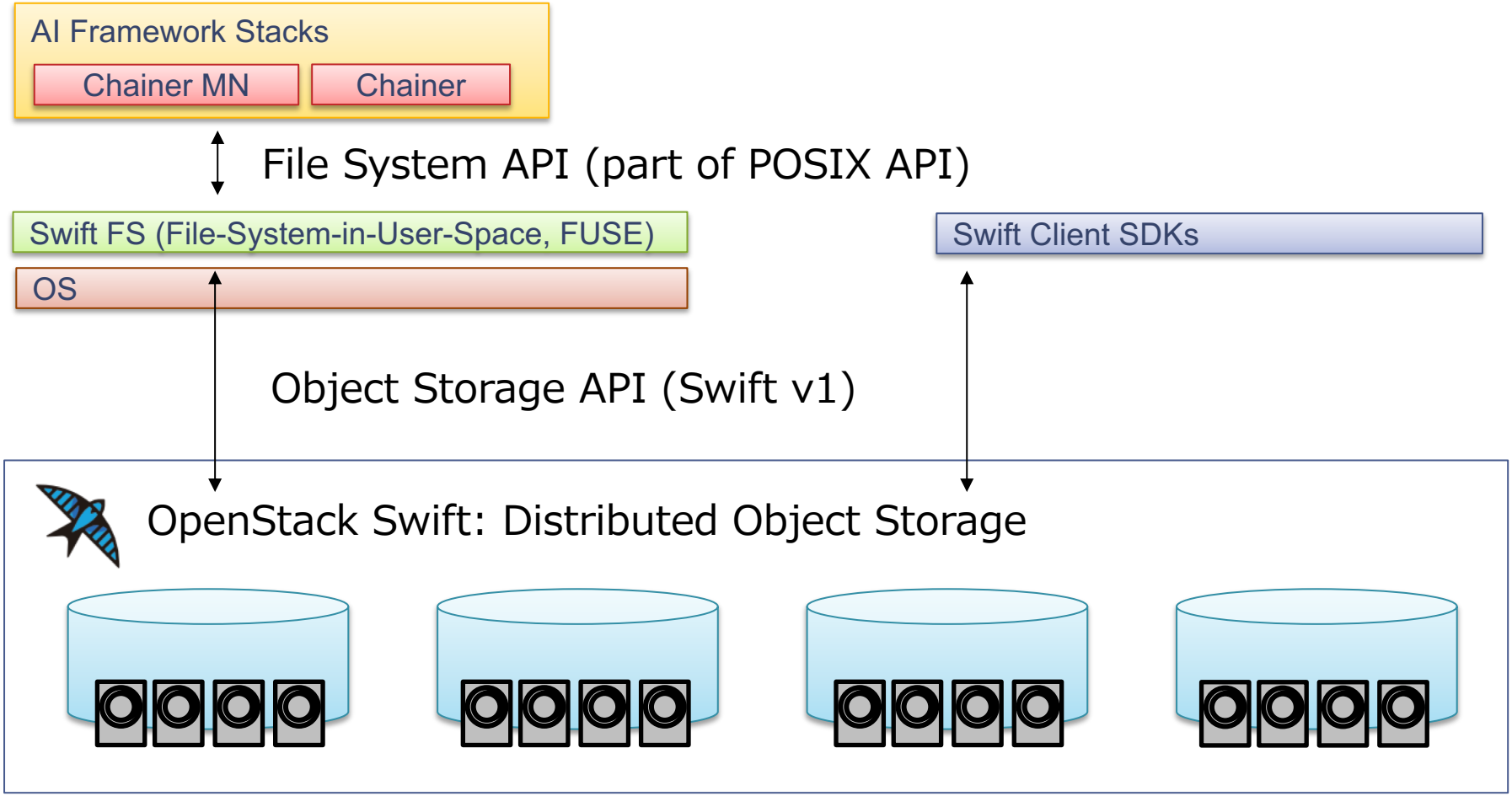
https://docs.openstack.org/swift/latest/

SwiftFS : Fuse-based file system on top of Swift
(not full-POSIX but one-by-one mapping of files and objects)

https://github.com/hironobu-s/swiftfs



3. Deploy: OpenStack Swift and SwiftFS Diagram



3. Deploy: Summary - GPU and NW design

- **Network Topology works well as expected**
  - High bandwidth and low latency
  - No overheads with Parallel GPU computation

- **Prometheus + In-House GPU monitoring script**
  - Monitoring GPU temperature and clock
  - Be useful to control the GPU computation task allocation for cooling plan.

3. Deploy: Summary - Good and Bad at NFS, Swift, SwiftFS

- **Good**
  - Simple configuration for each mature softwares
  - Scale out volumes and reads via Swift and SwiftFS
  - No changes at AI Framework layers

- **Bad**
  - Less durability at NFS than Swift cluster for /home
  - Complexity of cluster management because of SDS mixture

3. Deploy: Summary - Might be another Choice

- **ProxyFS**
  - May be a good solution to make scale-out filesystem only using OpenStack Swift
  - Consideration how to scale the reads on distributed machine learning
  - You can learn at https://goo.gl/6rF68i
- **Lustre, Ceph, GlusterFS, and other SDS solutions**
  - Needs to learn more...
  - Does any SDS solution have bi-modal access between file-system and objects APIs?

Please share/discuss the good solution !!!

4: Lessons Learned

- **You need to choose/survey proper DL framework for your requirement, since *the progress of DL framework is very fast.***
- **There is no off-the-shelf/complete environment publicly available for your own hardware/cluster setup.**
  - OpenMPI Dockerfile without mellanox drivers.
  - DL framework Dockerfiles without distribution libraries.
- ***Scrum/team-work is inevitable*, since DL users cannot solve troubles under DL frameworks.**
- ***We *CAN* build the AI stacks with OSS.***

Questions?

NTT Software Innovation Center

Kota Tsuyuzaki <tsuyuzaki.kota@lab.ntt.co.jp>
Takeharu Eda <eda.takeharu@lab.ntt.co.jp>
Kengo Okitsu <okitsu.kengo@lab.ntt.co.jp>