Chap12: Distributed & Parallel Computing for Deep Learning

National Tsing Hua University

2020 Fall Semester

Outline

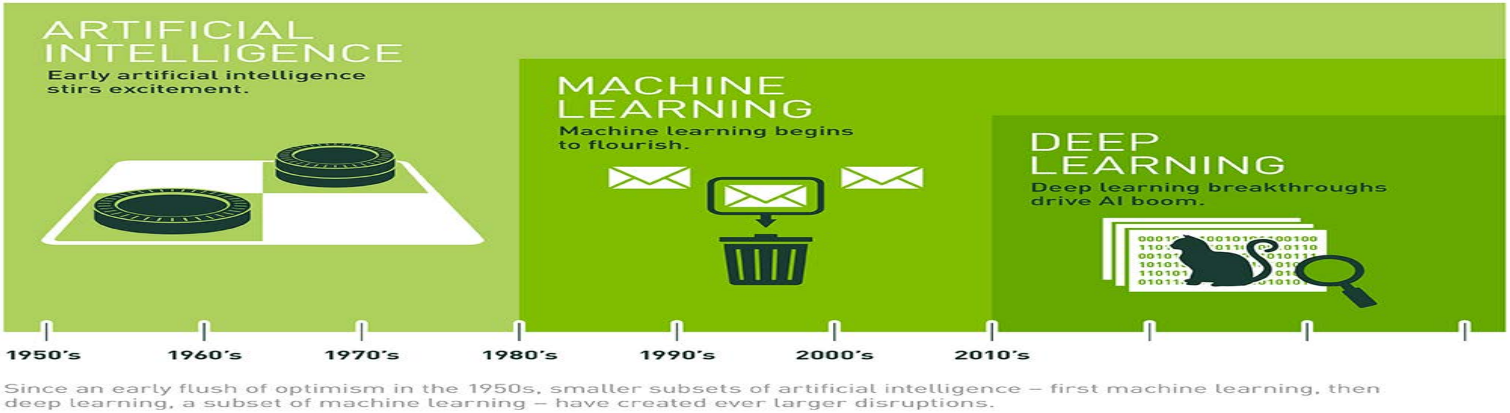
- Brief Introduction of Deep Learning
  - Computing Demand for Training
  - GPU Solutions
- Distributed Deep Learning Computations
  - Parallel strategies
  - Optimization strategies
- Distributed Deep Learning Frameworks
  - TensorFlow & Horovod
- Trend & Future of Deep Learning Computing
  - ML Systems & AutoML
  - Edge computing, CS-1 machine & AI Chips
  - Federated Learning
  - Remarks

Outline

- Brief Introduction of Deep Learning
  - Computing Demand for Training
  - GPU Solutions
- Distributed Deep Learning Computations
  - Parallel strategies
  - Optimization strategies
- Distributed Deep Learning Frameworks
  - TensorFlow & Horovod
- Trend & Future of Deep Learning Computing
  - ML Systems & AutoML
  - Edge computing, CS-1 machine & AI Chips
  - Federated Learning
  - Remarks

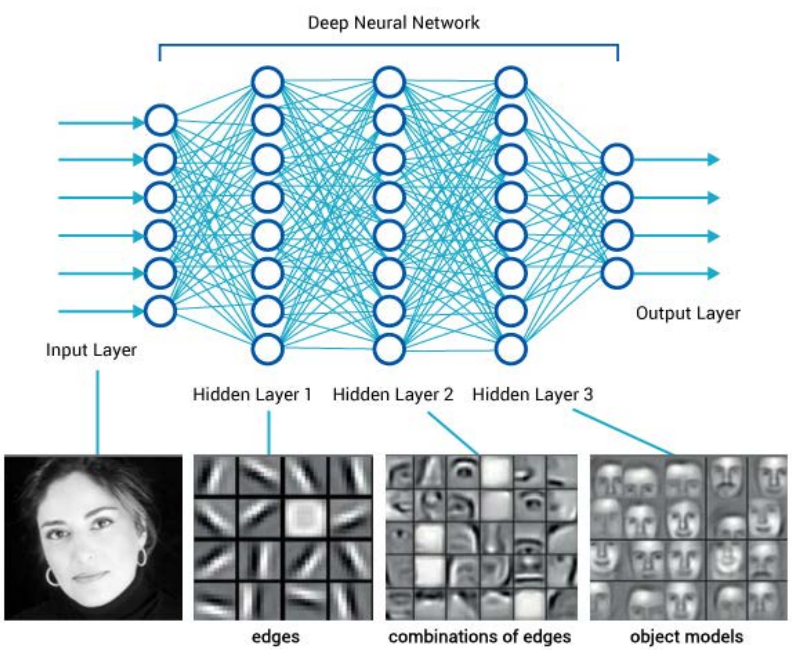
What is Deep Learning?

- **AI**: it emphasizes the creation of intelligent machines that work and react like humans
- **Machine Learning**: it provides systems the ability to automatically learn and improve from experience without being explicitly programmed
- **Deep Learning**: a subfield of machine learning concerned with algorithms inspired by the structure and function of the brain called artificial neural networks



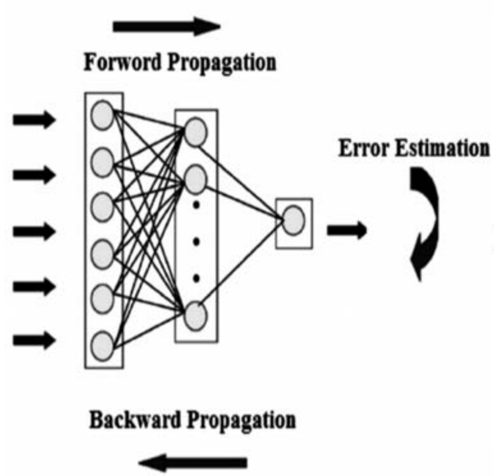
What is Deep Learning?

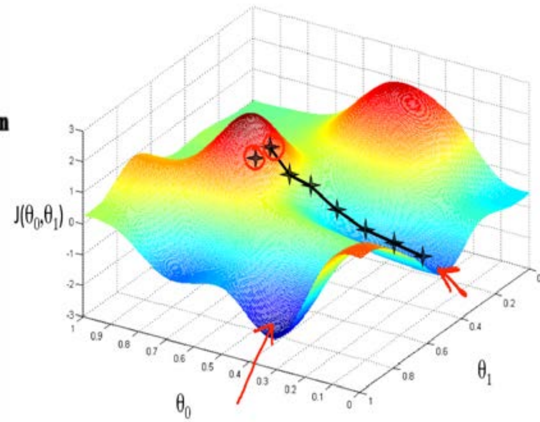
- Based on universal approximation theorem
  - A model constructed with a **greedy layer-by-layer method**, such as the artificial neural network
  - Model must be trained iteratively by large set of training data using the gradient decent algorithm



f()= dog

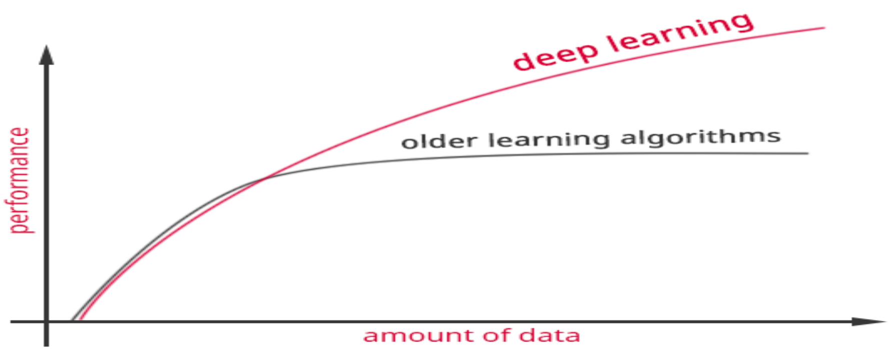
f()= cat

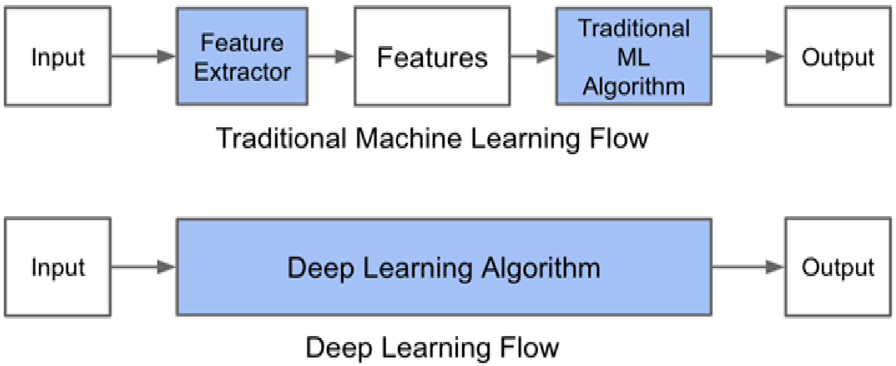




Strengths of DL

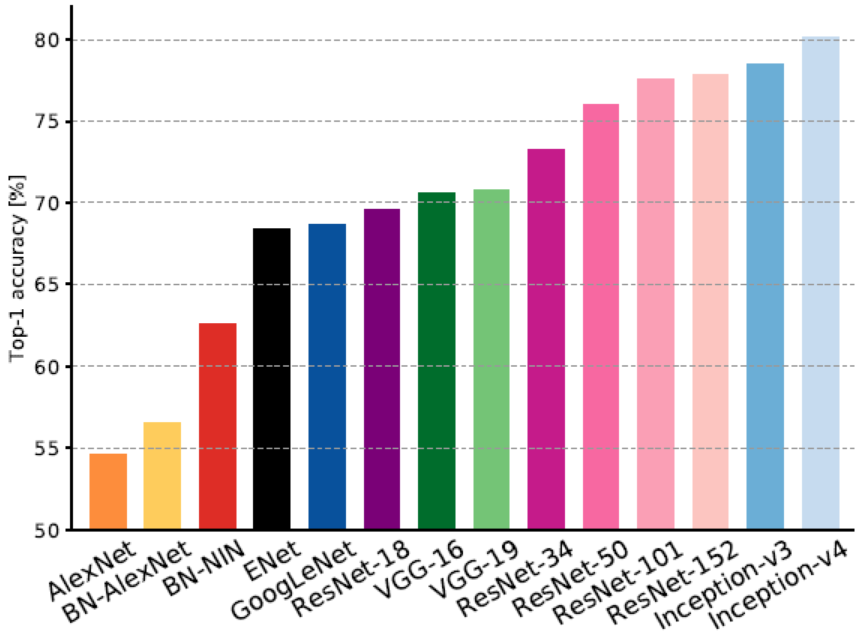
- Adapt to a wide variety of data
  - Adapted to new problems relatively easily
- Require less statistical training
  - Automatically fine-tune the learning procedure
- Learn with simple algorithms
  - But with ability to produce complex model
- Scale to large data sets
  - More data leads to more accurate results

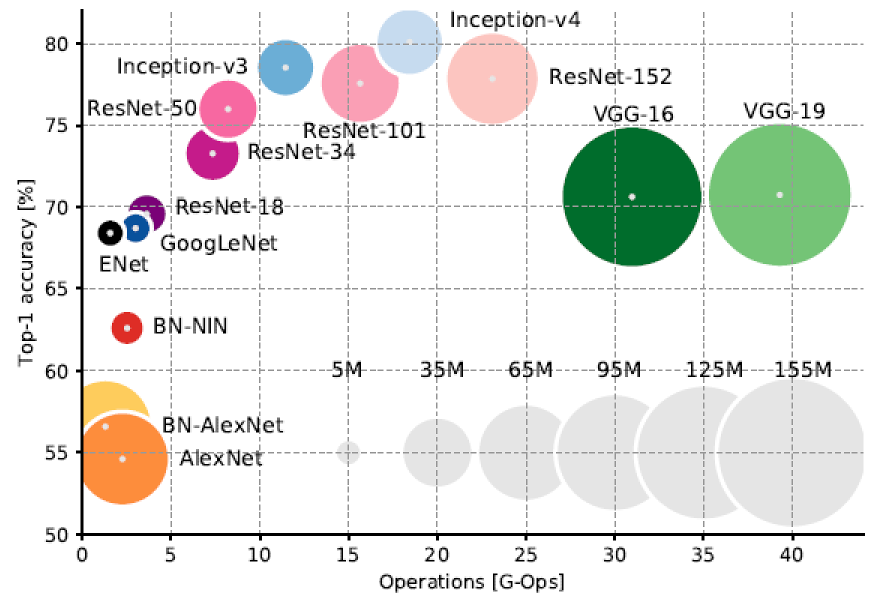




Growing Demand for Computing

- Larger training dataset
- Larger model
- More train iterations
- More tuning parameters

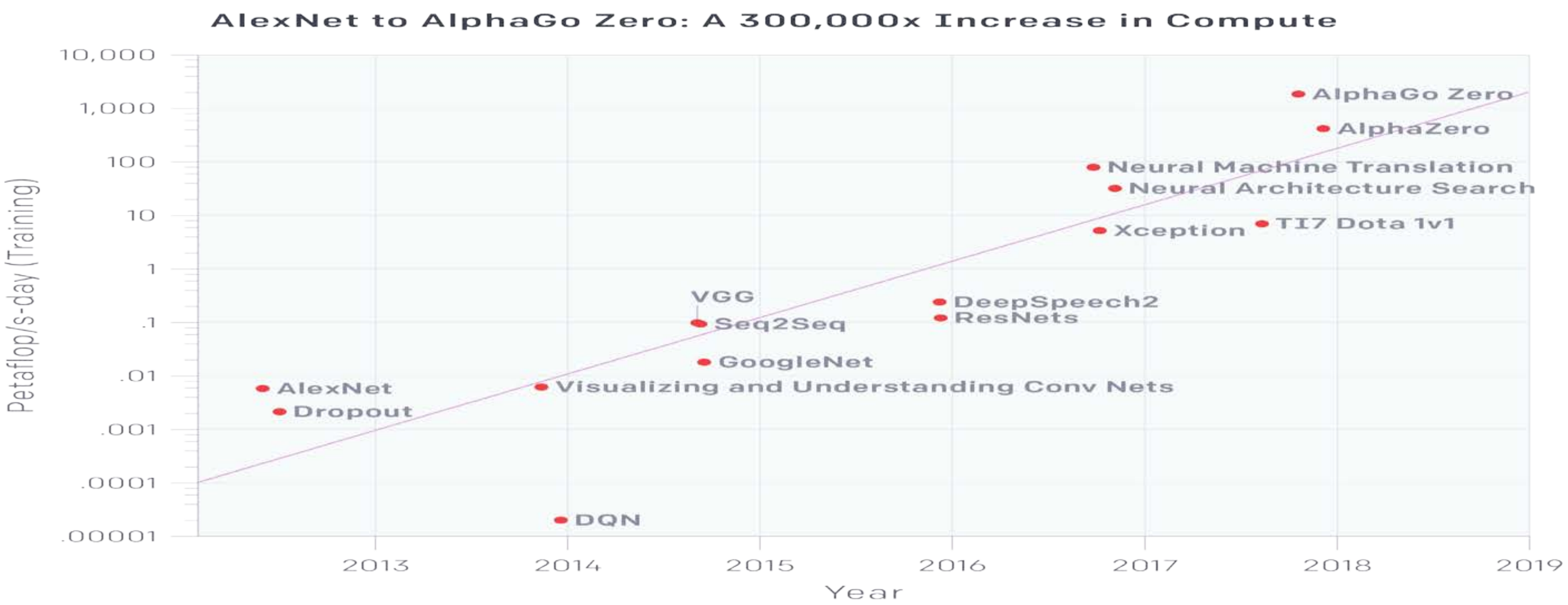




Source: Alfredo Canziani, “AN ANALYSIS OF DEEP NEURAL NETWORK MODELS FOR PRACTICAL APPLICATIONS”.

Growing Demand for Computing

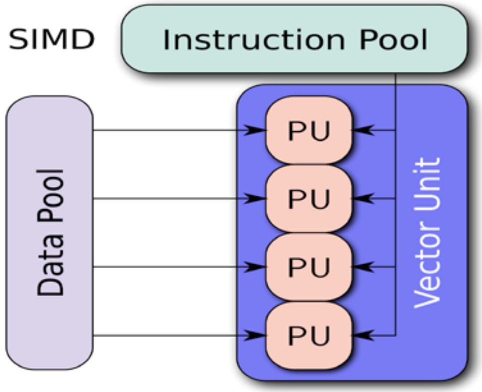
- 3.5 month doubling time since. (18 month double time for Moore's Law)
- 30K growth in 6 years



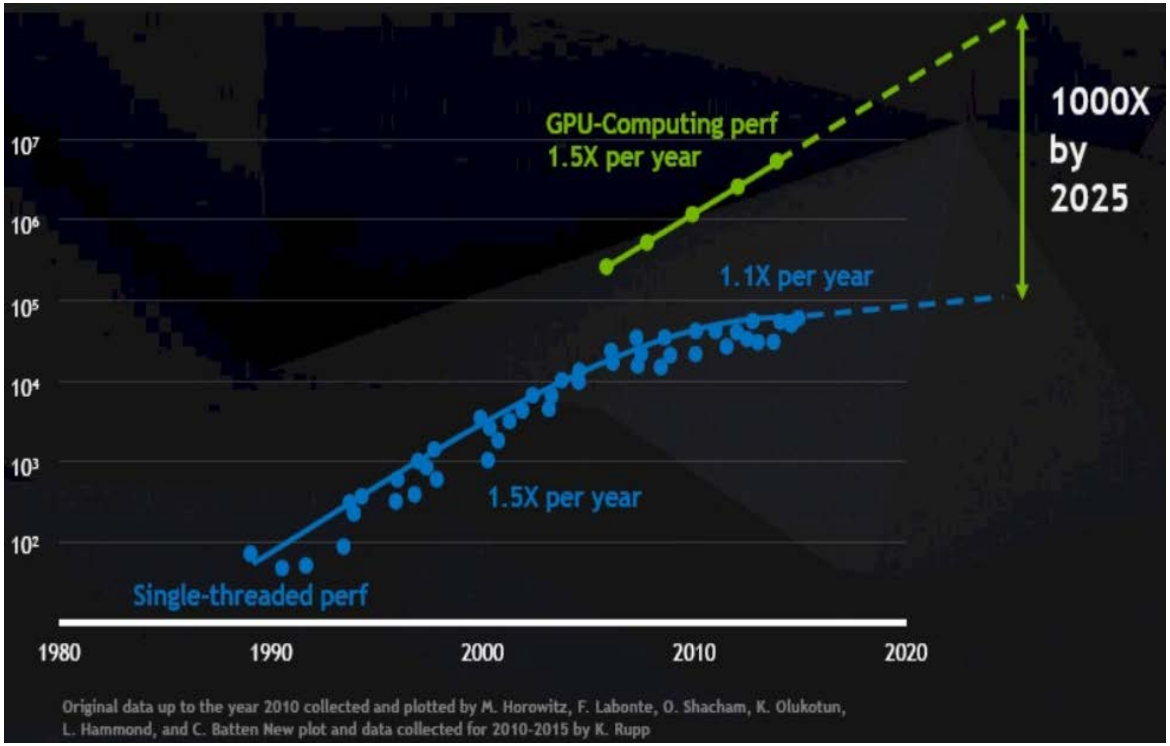
Source: openAI [https://openai.com/blog/ai-and-compute/]

Many-Core Processor: GPU

- Accelerator based on SIMD processor architecture



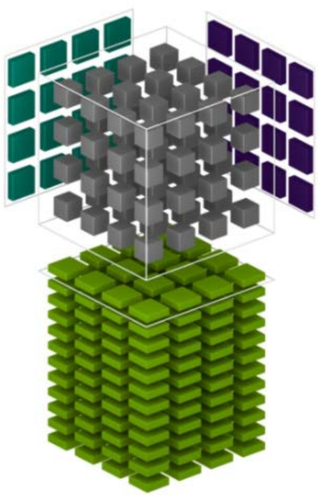
5,120 cores in a GPU

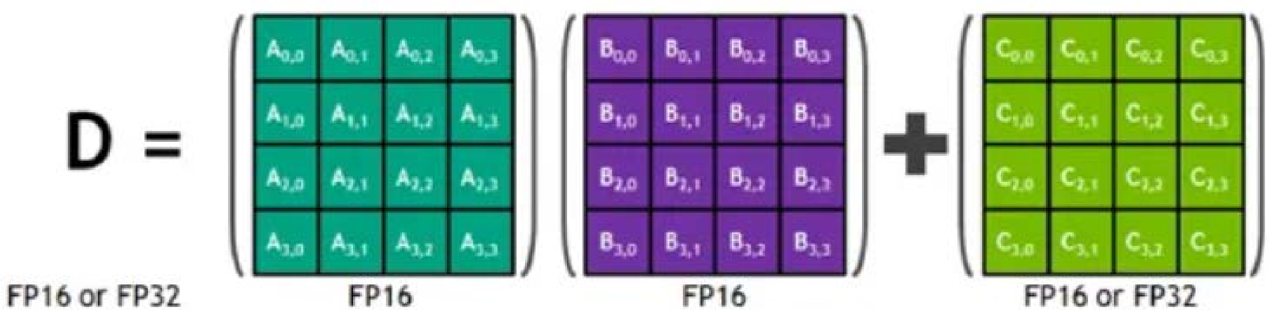


Images from Nvidia

TensorCore

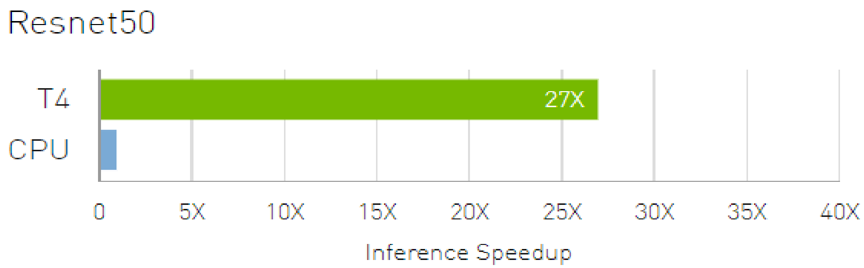
- Supported after Volta architecture
  - 640 TensorCore in Tesla V100 → 120 TFLOPS (16FLOPS on GPU core)
- Accelerate large matrix operations
  - perform mixed-precision matrix multiply and accumulate calculations in a single operation.
  - An common operation in most NN computations

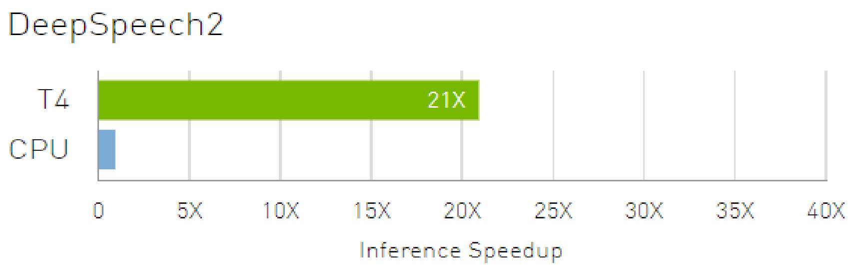




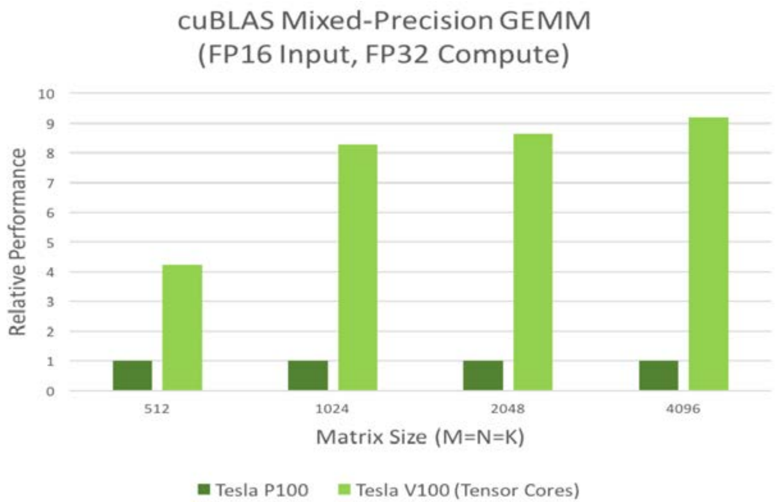
TensorCore

- Enables massive increases in throughput and efficiency
  - T4 has the world's highest inference efficiency, up to 40X higher performance compared to CPUs with just 60% power consumption.
- Currently support in Caffe, MXNet, PyTorch, Theano, TensorFlow
  - But not for CNTK、Chainer、Torch



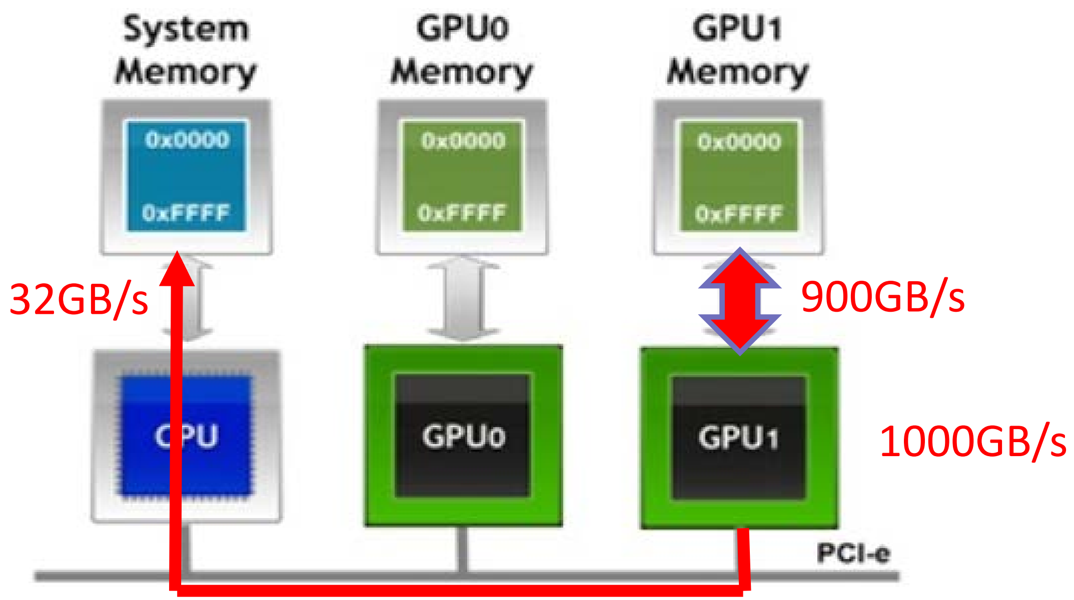


Chip-to-chip GPU-to-CPU speedups | NVIDIA Tesla T4 GPU vs Xeon Gold 6140 CPU



Input matrices are half precision, computation is single precision.

GPU: Memory Access Bottleneck



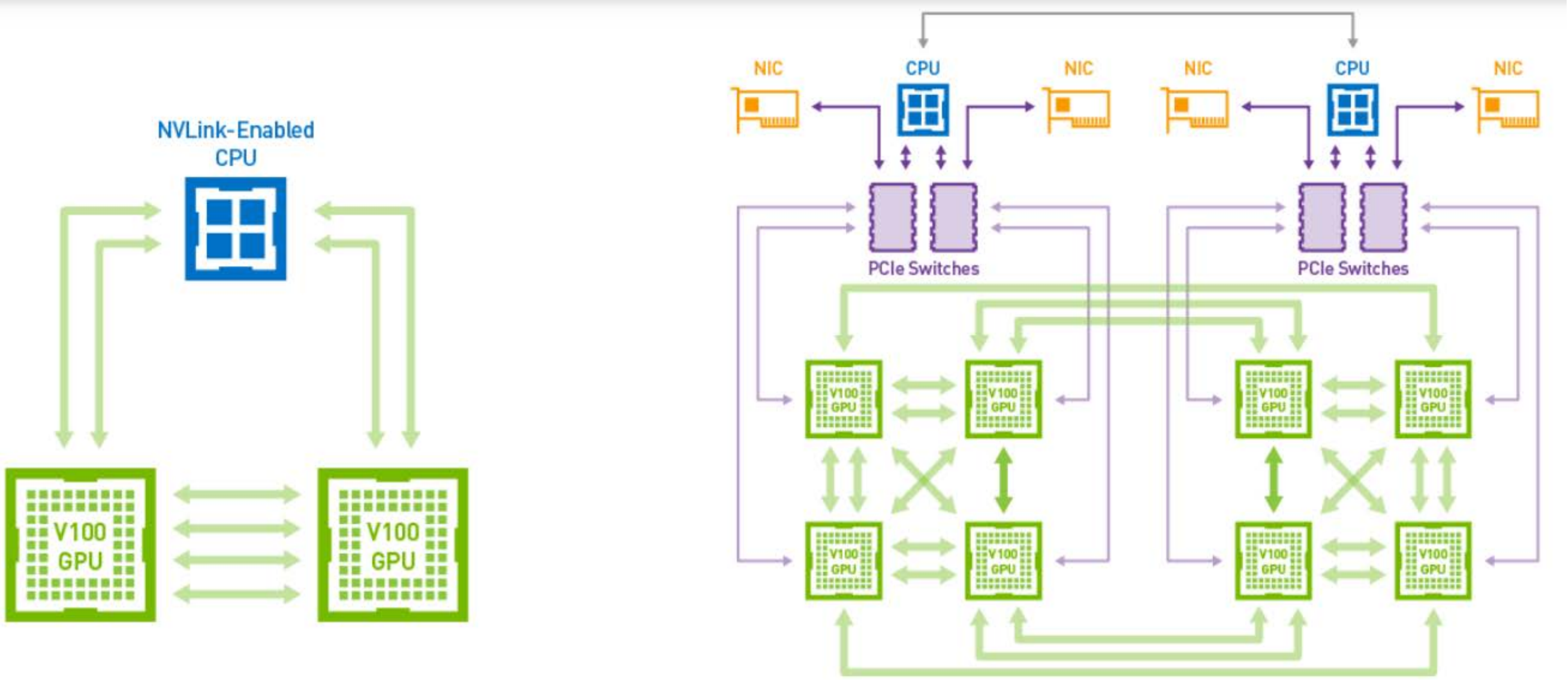
- GPU is capable of processing 1,000GB/s data
- GPU internal memory access can reach 900GB/s
- But **PCI-E Gen4** only provide 32GB/s bandwidth

NV-Link Fabric

- A high-bandwidth, energy-efficient **interconnect** that enables ultra-fast communication between the CPU and GPU, and between GPUs
- Achieve 300GB/s data sharing rates
  - **5 to 12 times faster** than the traditional **PCIe Gen3** interconnect:
- Must use the SXM GPU module

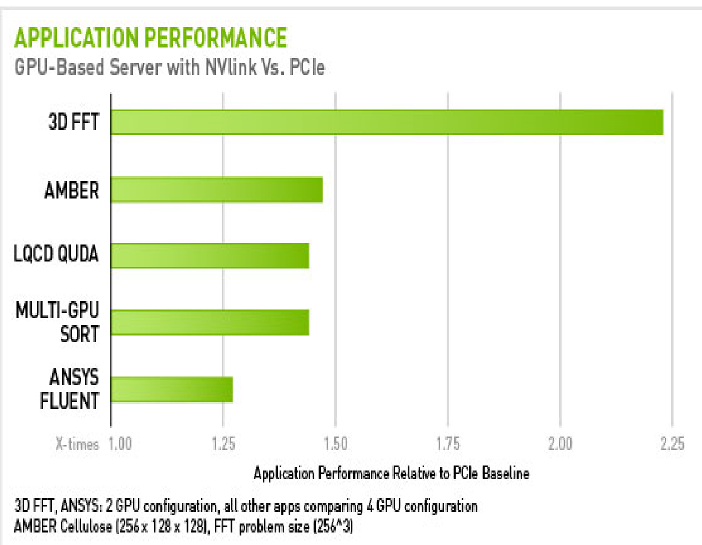
NV-Link Fabric

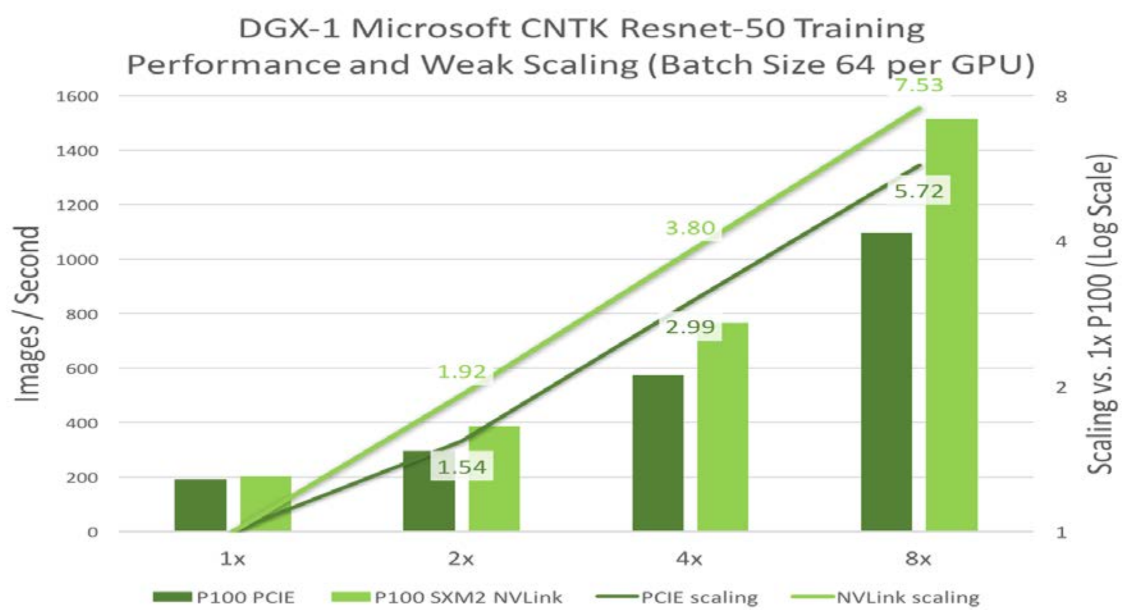
- Only NVLink-Enabled CPU can use NVLink to transfer data from host mem. to device mem.



NV-Link

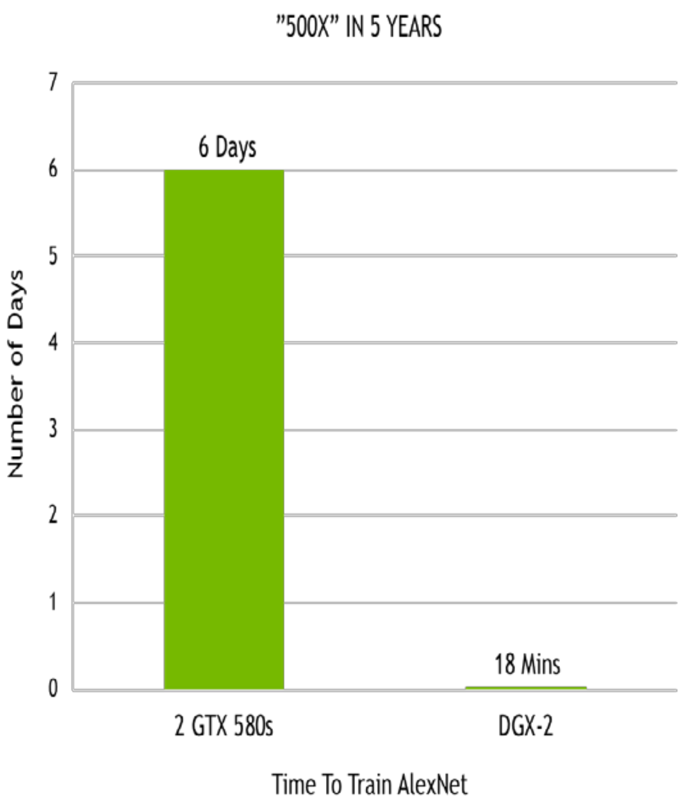
- Higher network performance deliver higher overall application performance, and better scalability (less communication overhead)

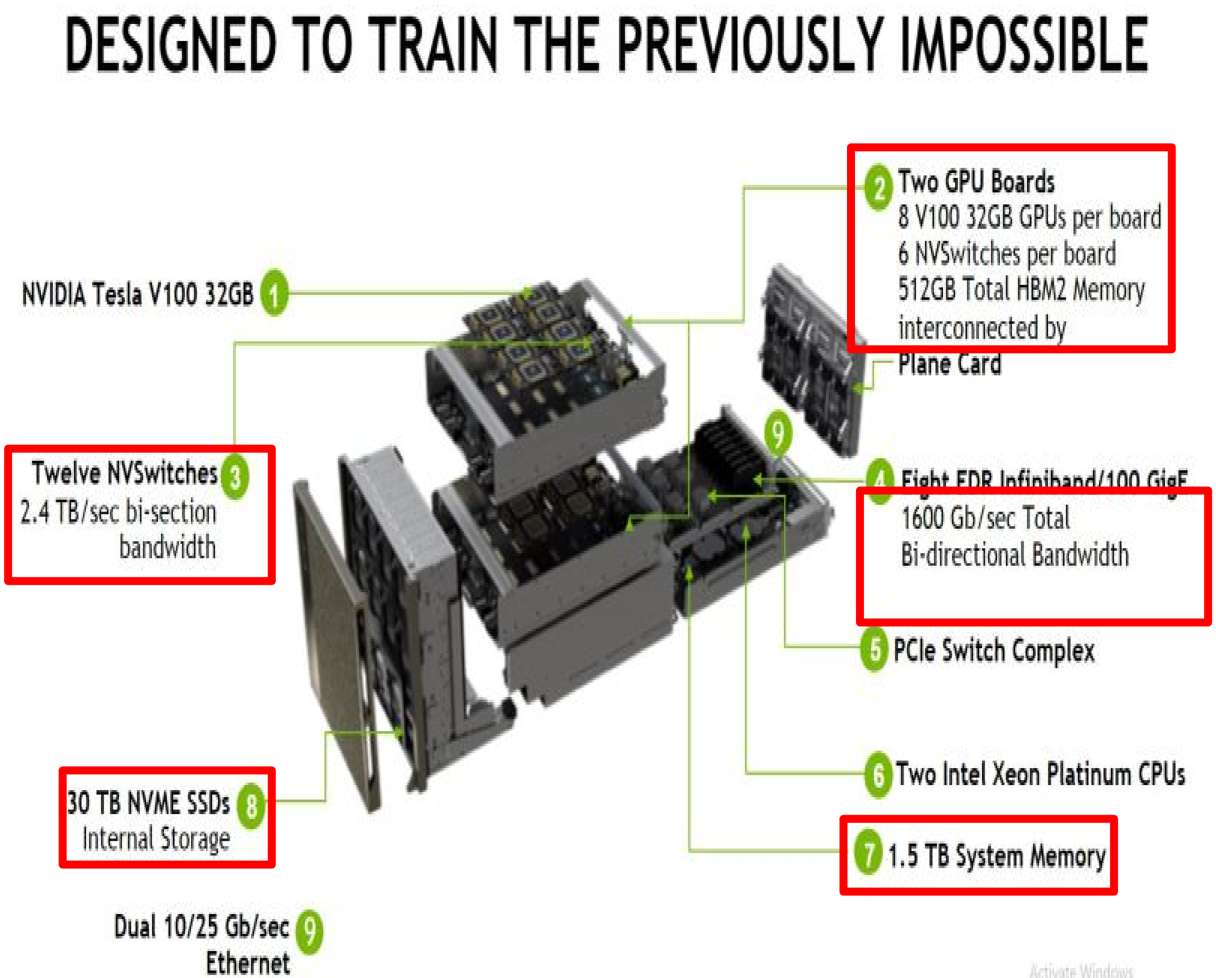




Images from Nvidia

DGX-2 GPU Server

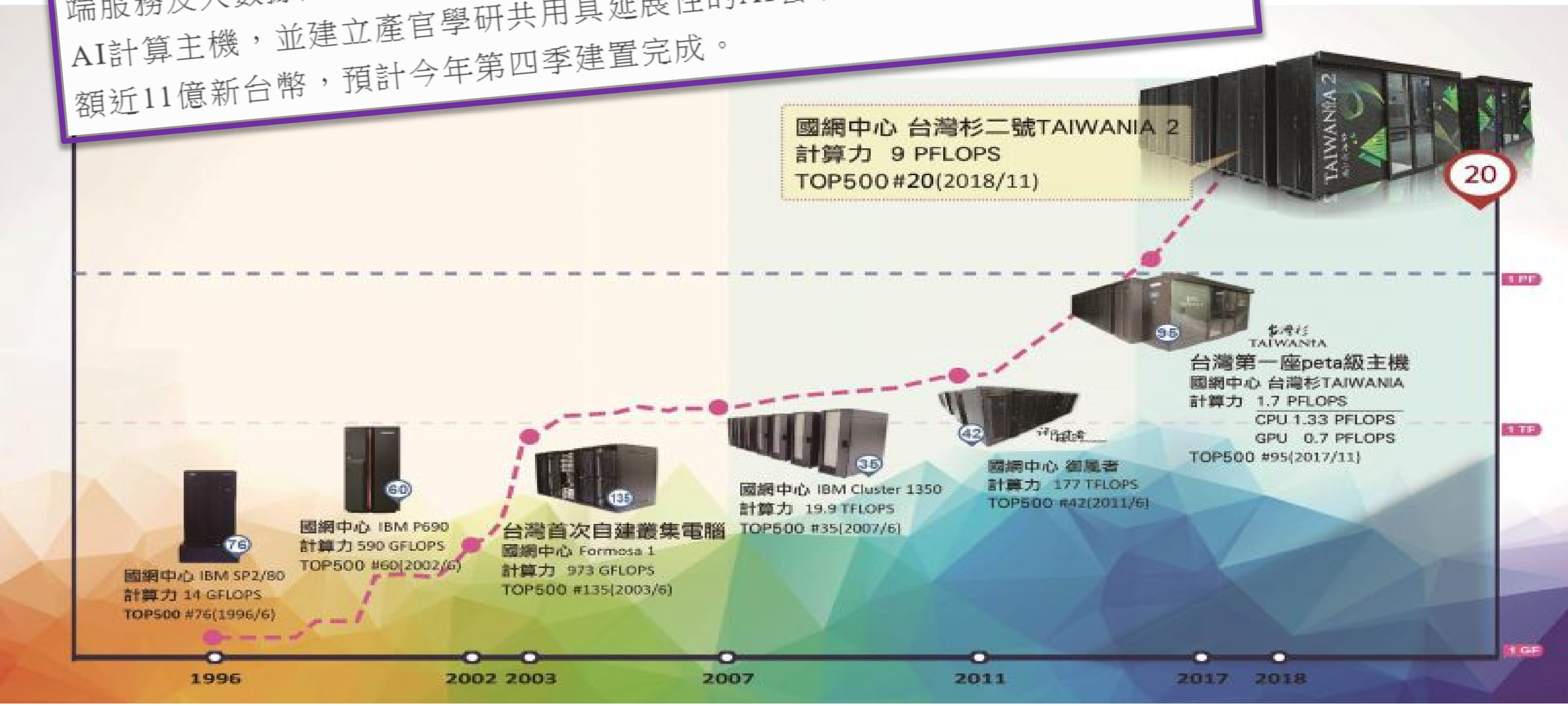




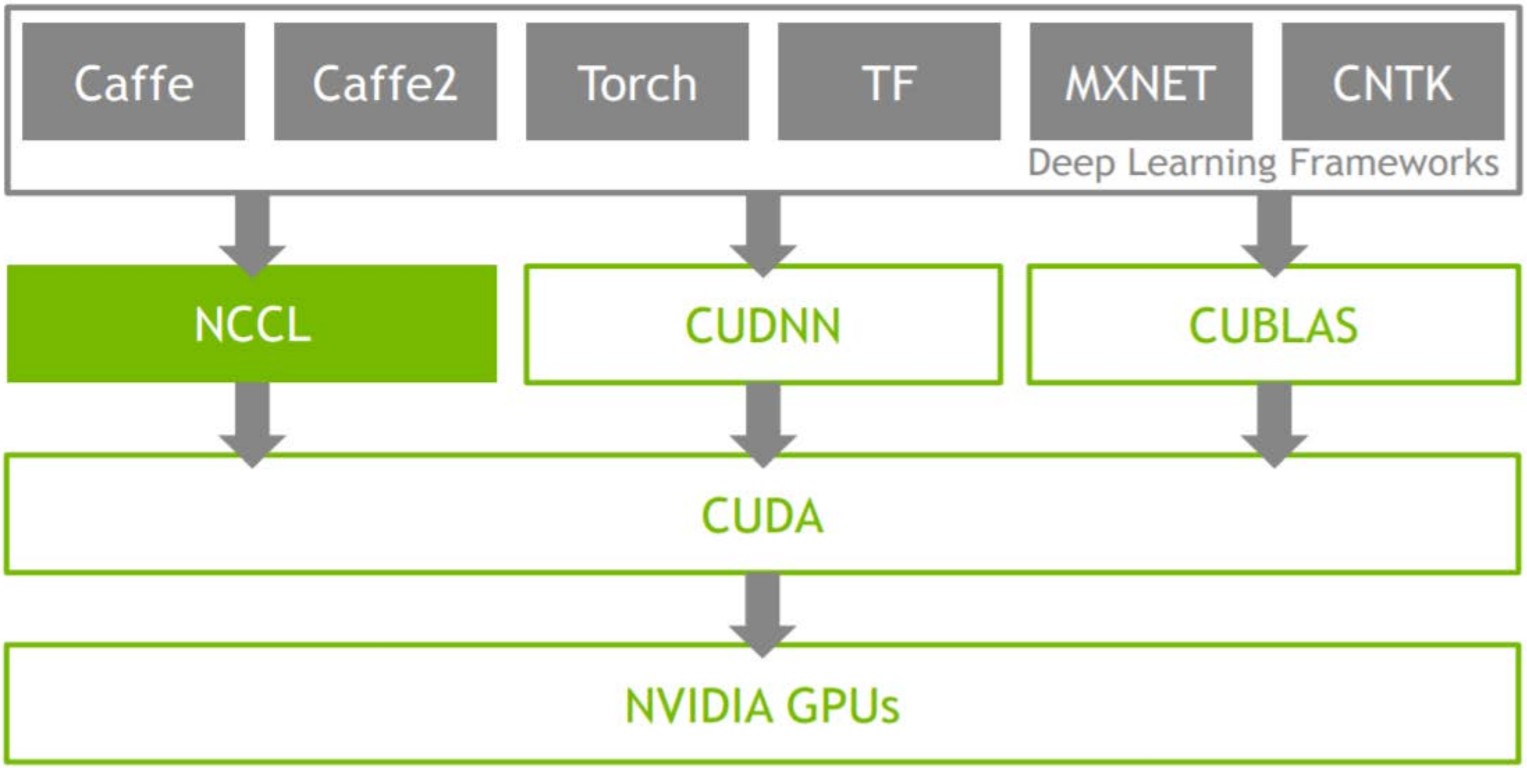
Images from Nvidia

台灣杉二號

【財訊快報／王宜弘報導】搶攻AI商機，台廠大團結！華碩(2357)、廣達(2382)以及台灣大(3045)結盟組成「台灣人工智慧A Team」，成軍後首戰告捷！週一(30日)三方共同宣布取得國家實驗研究院國家高速網路與計算中心「雲端服務及大數據運算設施暨整合式階層儲存系統建置案」，將協助建置新一代的AI計算主機，並建立產官學研共用具延展性的AI雲端大資料計算平台，建置總金額近11億新台幣，預計今年第四季建置完成。

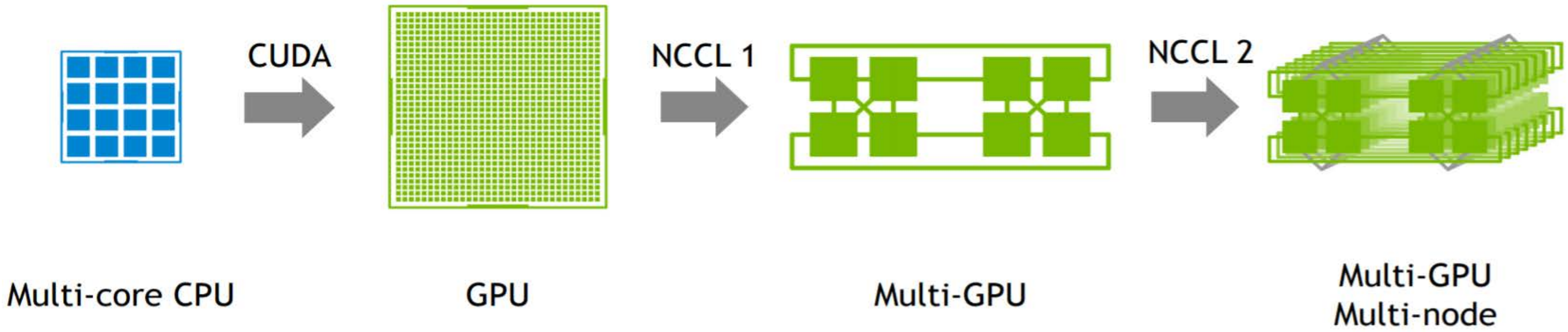


CUDA Libraries for Deep Learning



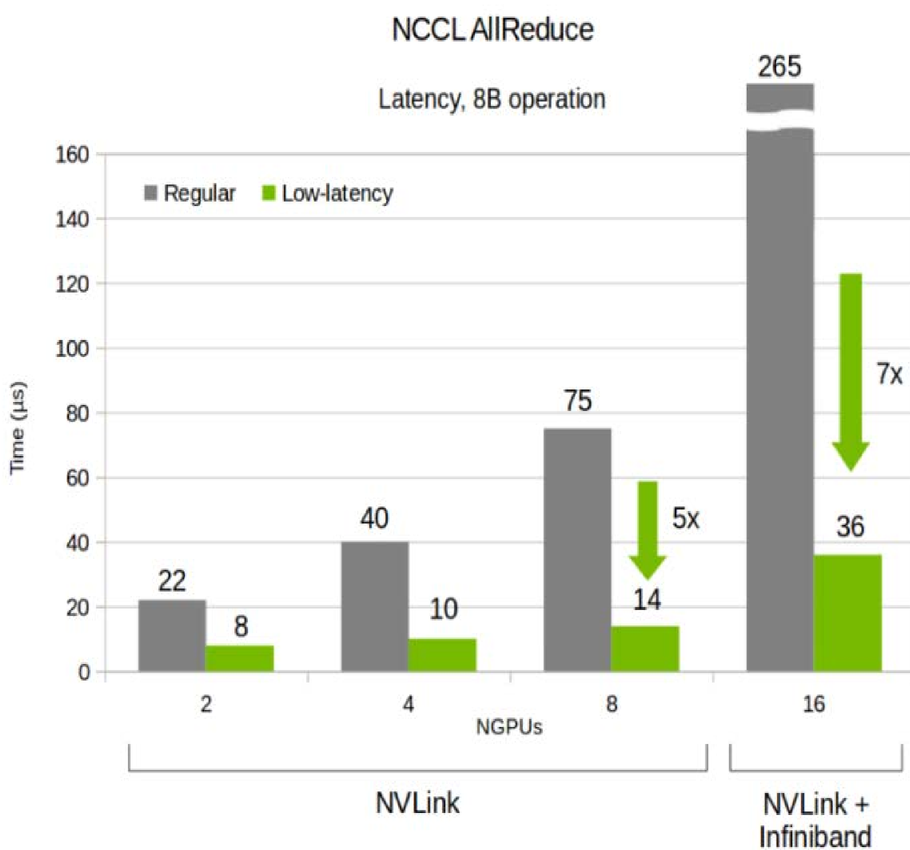
NCCL

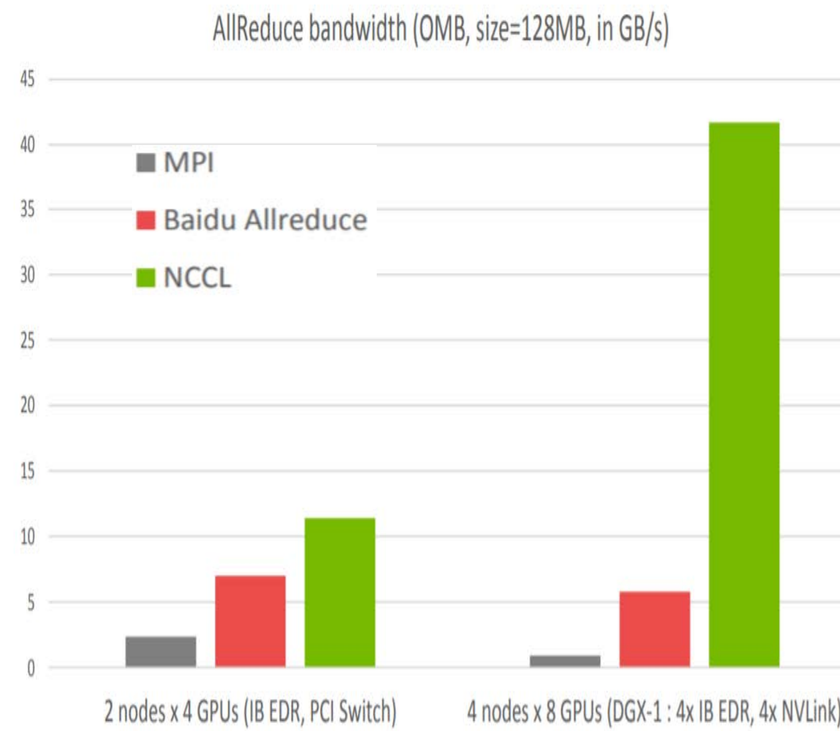
- NVIDIA Collective Communications Library
  - Optimized implementation of inter-GPU communication operations, such as **allreduce**
  - Deep learning frameworks can rely on NCCL's highly optimized, MPI compatible and topology aware routines, to take full advantage of all available GPUs within and across multiple nodes.
  - Optimized for **high bandwidth** and **low latency** over PCIe and NVLink high speed interconnect for intra-node communication and sockets and InfiniBand for inter-node communication



NCCL

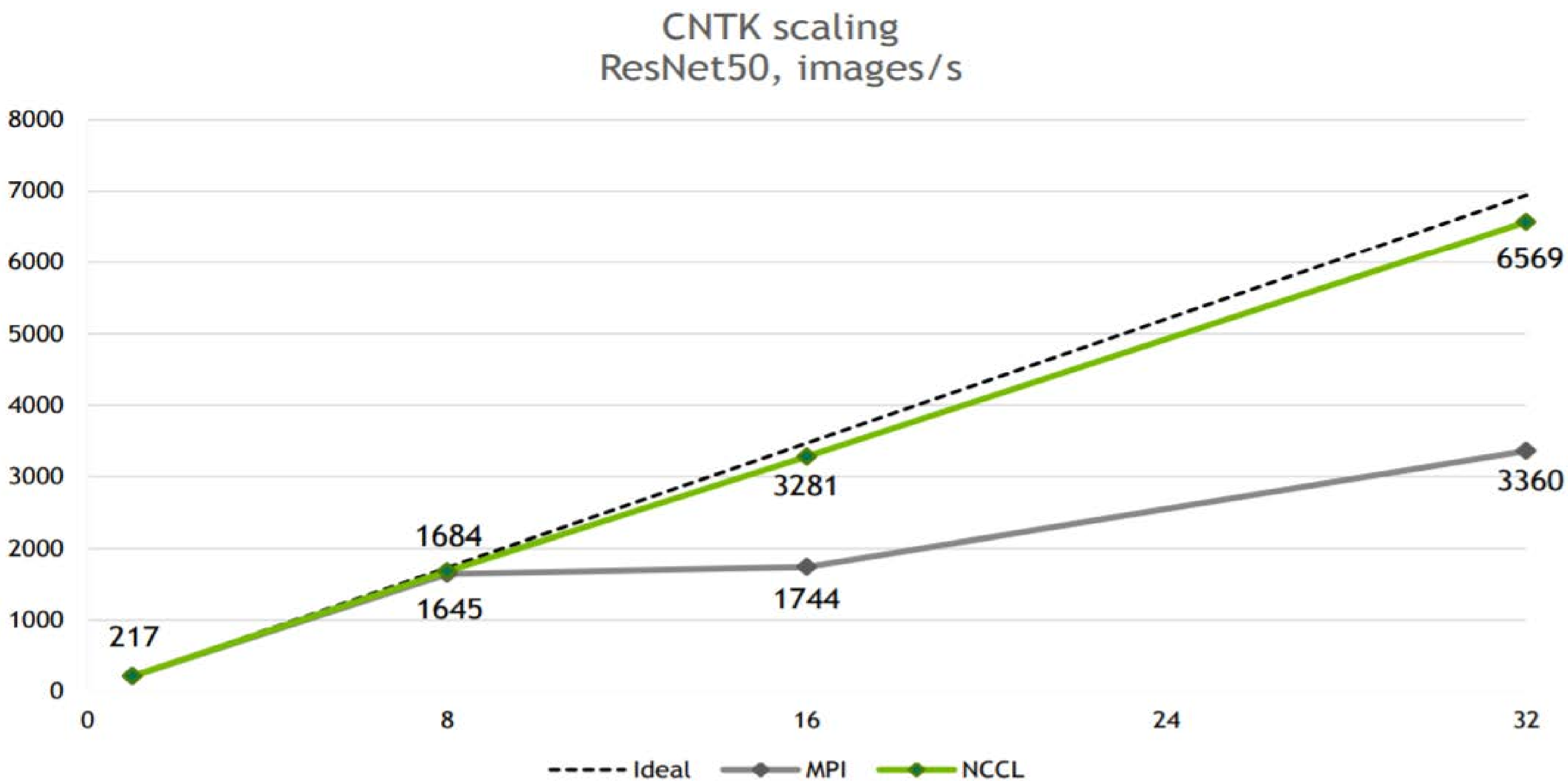
- Performance improvement on BW and Latency





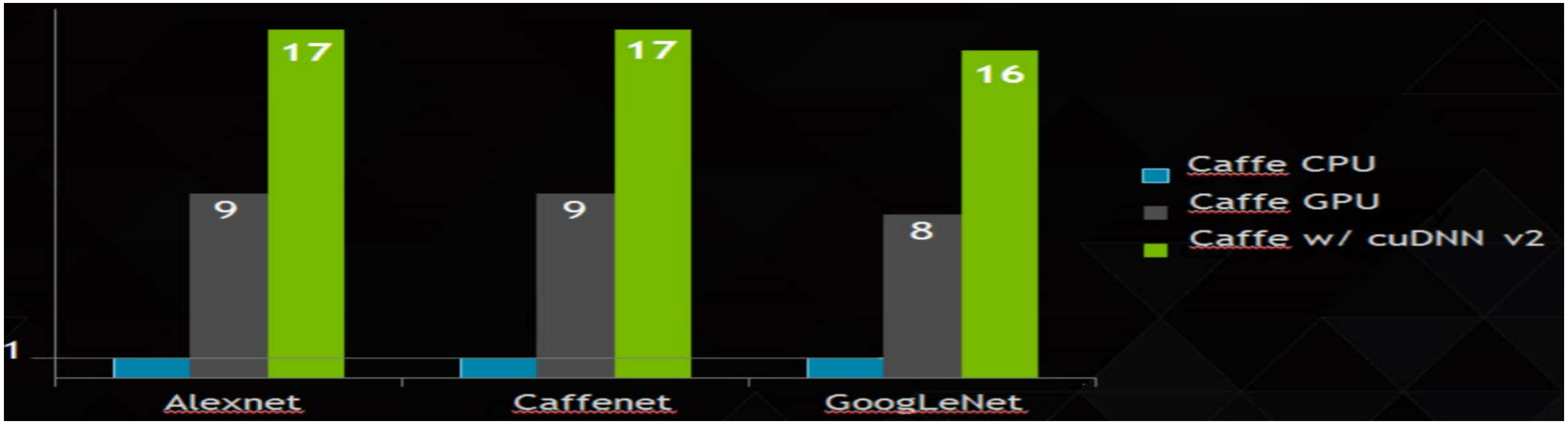
NCCL

- Performance improvement on scalability



cuDNN

- Basic Deep Learning Subroutines:
  - E.g., convolutions, pooling, activation
  - Let user write a DNN application without custom CUDA code
- Flexible Layout
  - Handle any data layout
- Memory – Performance tradeoff
  - Good performance with minimal memory use, great performance with more memory use



cuBLAS

- BLAS: Basic Linear Algebra Subprograms
  - Defines a set of common functions for scalars, vectors, and matrices
    - E.g., cublaslasmax(): finds the smallest(first) index in a vector that is a maximum for that vector
  - Good for anything that uses heavy linear algebra computations
    - E.g., graphics, machine learning, computer vision, physical simulations

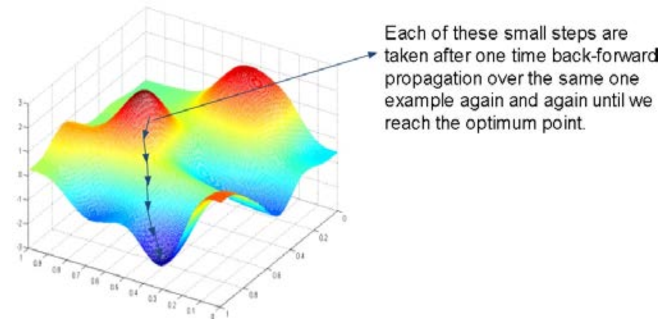
numpy	math	cuBLAS (<T> is one of S, D, C, Z, H)
numpy.multiply(α, χ)	$(\lambda \mathbf{A})_{ij} = \lambda(\mathbf{A})_{ij}$	cublas<T>gemm(α, χ)
numpy.multiply(χ, γ)	$(A \circ B)_{i,j} = (A)_{i,j}(B)_{i,j}.$	cublas<T>gemm(χ, γ)
numpy.multiply(χ, A)	Aχ = C	cublas<T>gemm(χ, A)
numpy.multiply(A, B)	$C \leftarrow \alpha AB + \beta C$	cublas<T>gemm(A, B)

Outline

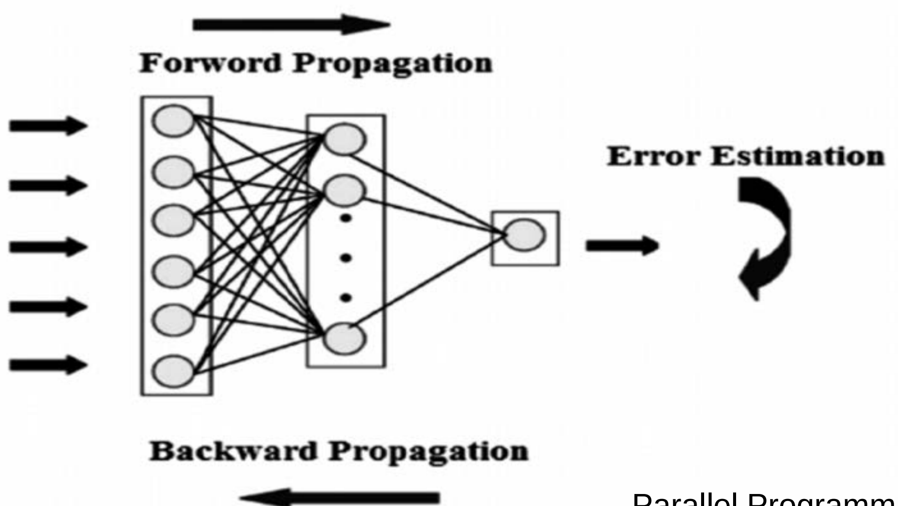
- Brief Introduction of Deep Learning
  - Computing Demand for Training
  - GPU Solutions
- Distributed Deep Learning Computations
  - Parallel strategies
  - Optimization strategies
- Distributed Deep Learning Frameworks
  - TensorFlow & Horovod
- Trend & Future of Deep Learning Computing
  - ML Systems & AutoML
  - Edge computing, CS-1 machine & AI Chips
  - Federated Learning
  - Remarks

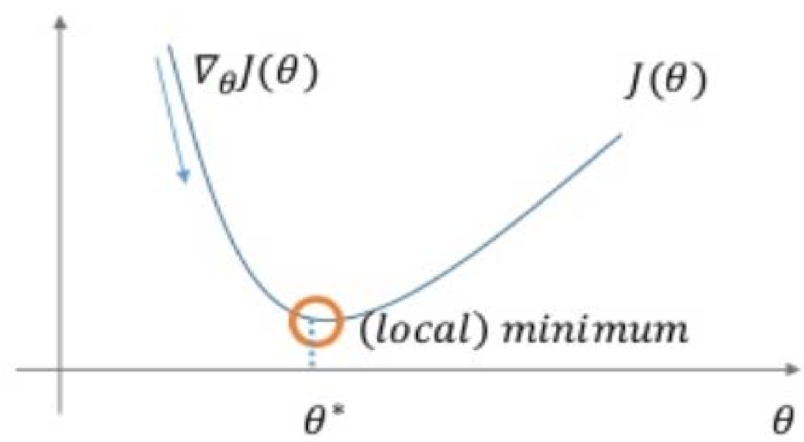
Gradient Descent Algorithm

- Gradient descent is a way to minimize an objective function $J(\theta)$
  - $J(\theta)$: objective function
  - $\theta \in R^d$: model's parameters (weight)
  - $\nabla_\theta J(\theta)$: graident
  - α: learning rate



$$\boxed{\theta = \theta - \alpha * \nabla_\theta J(\theta)}$$



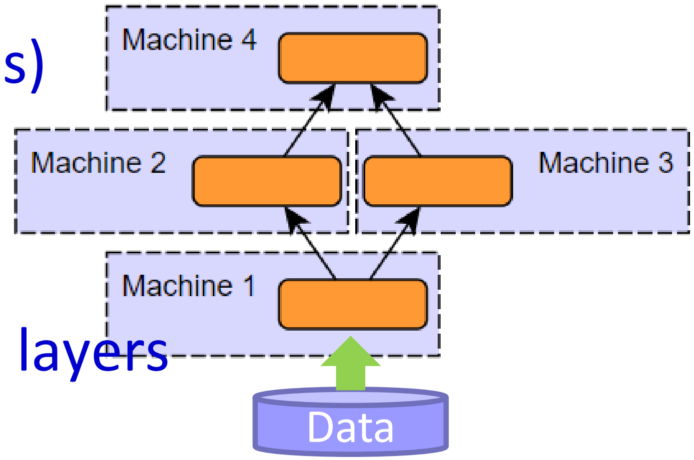


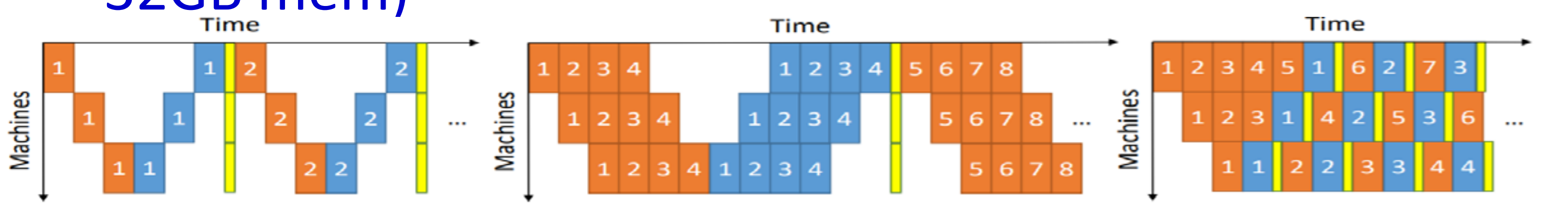
How to Utilize Multiple Machines?

- We could utilize resources by…
  - Running multiple training jobs for different models
  - Running multiple training jobs with the same model, but different hyper-parameters
  - Running a single model training job across multiple machines → **distributed training**
    - Fully utilize the resources of a system not just a single machine

Model Parallelism

- Parallelization
  - Model is split across machines(GPUs)
  - The whole dataset is replicated
- Weakness
  - Harder to achieve good scalability
  due to synchronization point between layers
  (could be addressed by pipeline)
  - Model modification is equired if no shared memory
- Strength
  - More suitable on a single machine with multi-GPUs
  - The only solution when model cannot fit into a GPU (16 or 32GB mem)

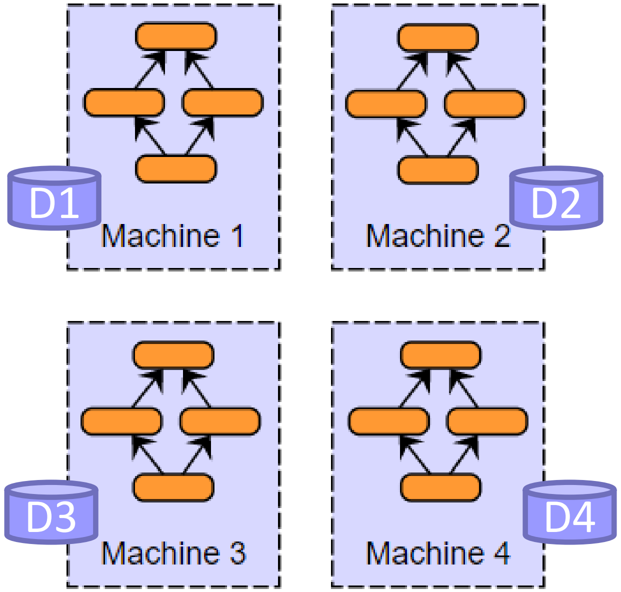




(a) Pipelined execution (#inflight 1) (b) Pipelined parallel execution (#inflight 4) (c) Asynchronous model parallelism

Data Parallelism

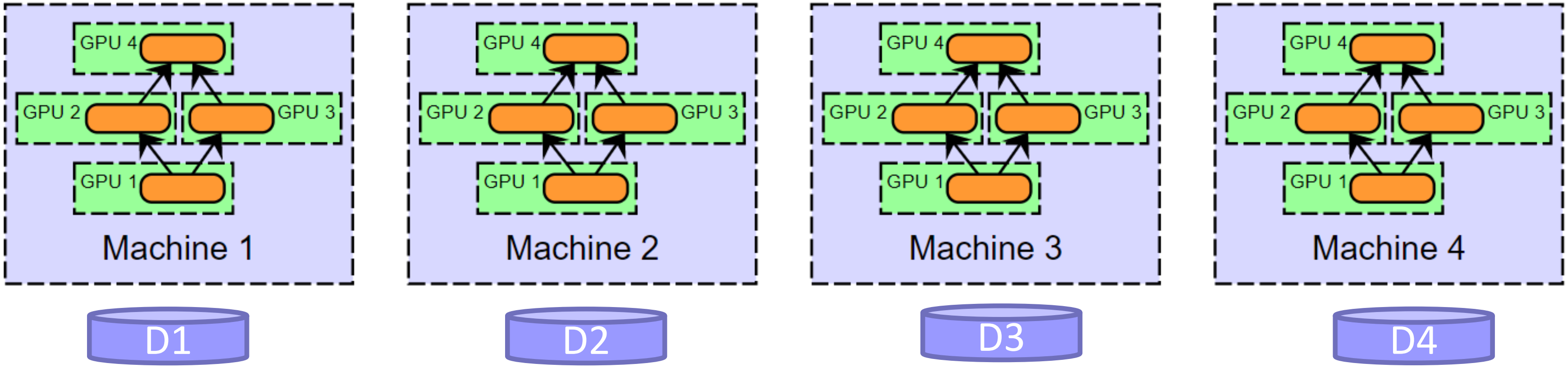
- Parallelization
  - Each machine (GPU) independently evaluate the whole model on a part of the dataset to compute gradient
  - Weight is updated by the average of gradients from all nodes
- Strength
  - Easier to achieve linear scale
  - Preferred approach for distributed systems
- Weakness
  - The whole model must fit into the memory of a node (GPU)



How to minimize the **communication overhead** of distributed stochastic gradient descent(SGD) is critical

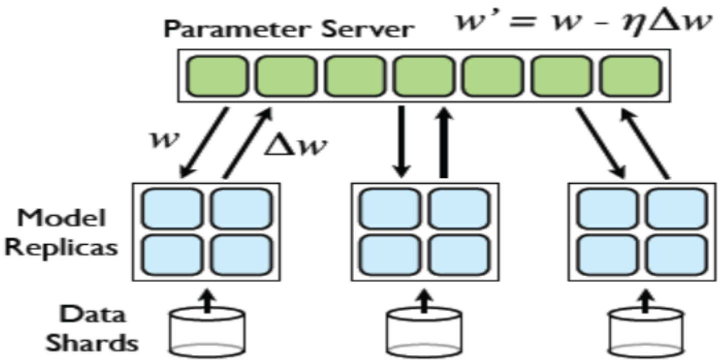
Data + Model Parallelism

- Most commonly used solution in practice
  - Model parallelism is automated done by the compute framework
  - Data parallelism is controlled by programmers
    - Data partition
    - Parameter(weight) swapping

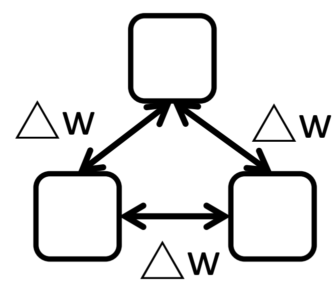


Parameter Server vs. Allreduce

- **Parameter Server (PS):**
  - **De-centralized** across PS servers
  - Worker send gradient & receive weight
  - Support **both synchronized & asynchronized** SGD
  - # PS servers must be tuned
    - Too many ➔ more small messages
    - Too few ➔ network bottleneck



- **Allreduce:**
  - Peer to peer, **fully distributed**
  - Workers send gradient to each other, then compute weight by themselves
  - **Balanced communication load** across links
  - **Need to be synchronized SGD**



Optimization Strategies

- Mini batch
- Asynchronous SGD
- Stale Synchronous SGD
- Quantized SGD
- Task placement
- Principals of Distributed Training

1. Mini Batch SGD

- Algorithms:
  - Batch Gradient Descent: use all m examples in each iteration
  - Stochastic Gradient Descent: use 1 examples in each iteration
  - Mini-batch Gradient Descent: use b examples in each iteration

Say $b = 10, m = 1000$.

Repeat {

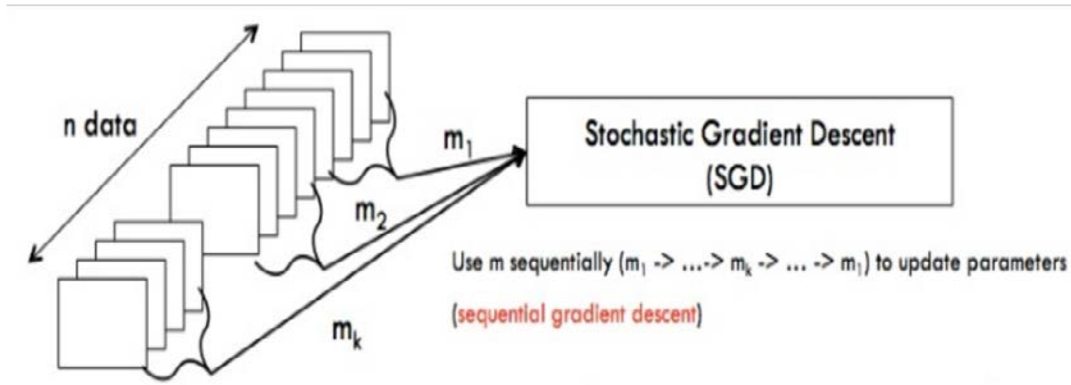
for $i = 1, 11, 21, 31, \ldots, 991$ {

$$\theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} (h_\theta(x^{(k)}) - y^{(k)}) x_j^{(k)}$$

(for every $j = 0, \ldots, n$)

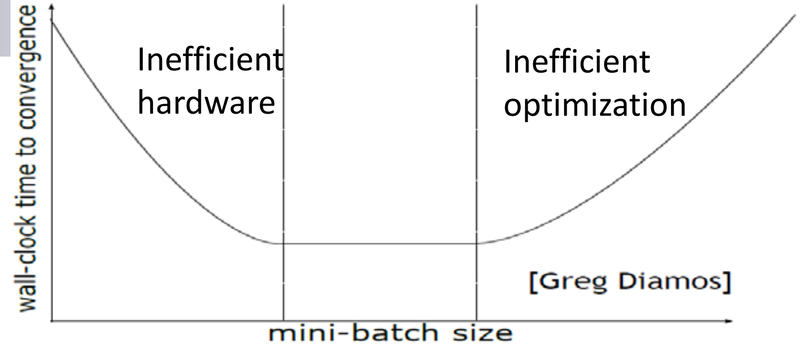
}

}



https://www.coursera.org/learn/machine-learning/lecture/9zJUs/mini-batch-gradient-descent

1. Mini Batch SGD

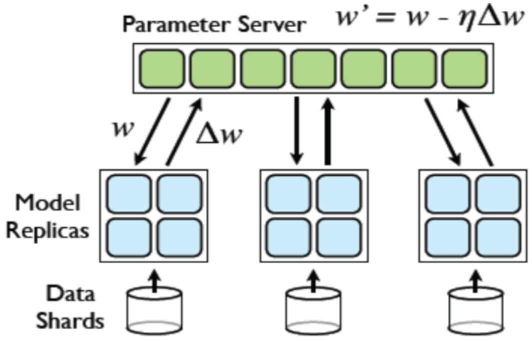


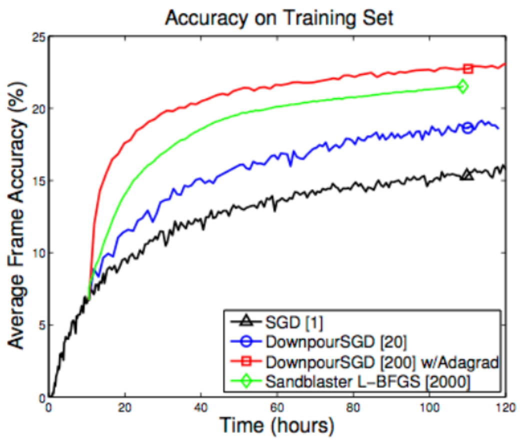
- Advantages:
  - Vectorization: make data parallelism arbitrarily efficient by increasing the batch size (In particular for GPU)
  - Lower communication cost: fewer number of iterations comparing to SGD
  - Smoother update: the variance of the update is reduced
- Risks
  - Very big batch sizes adversely affect the SGD converges rate as well as the quality of the final solution
  - Noise actually can be useful as it may help escape local minima

Mu Li, Tong Zhang, Yuqiang Chen, and Alexander J. Smola. "Efficient mini-batch training for stochastic optimization". In *Proceedings of the 20th ACM SIGKDD, 2014*

2. Asynchronous(Downpour) SGD

- Parameter updates can be handle **asynchronously**.
  - Parameter server shards are updated independently with **inconsistent timestamp**
  - Updates may be **out of order**
  - Training simply stops after N iterations
- In practice, relaxing consistency requirements is remarkably effective, and could achieve even better accuracy

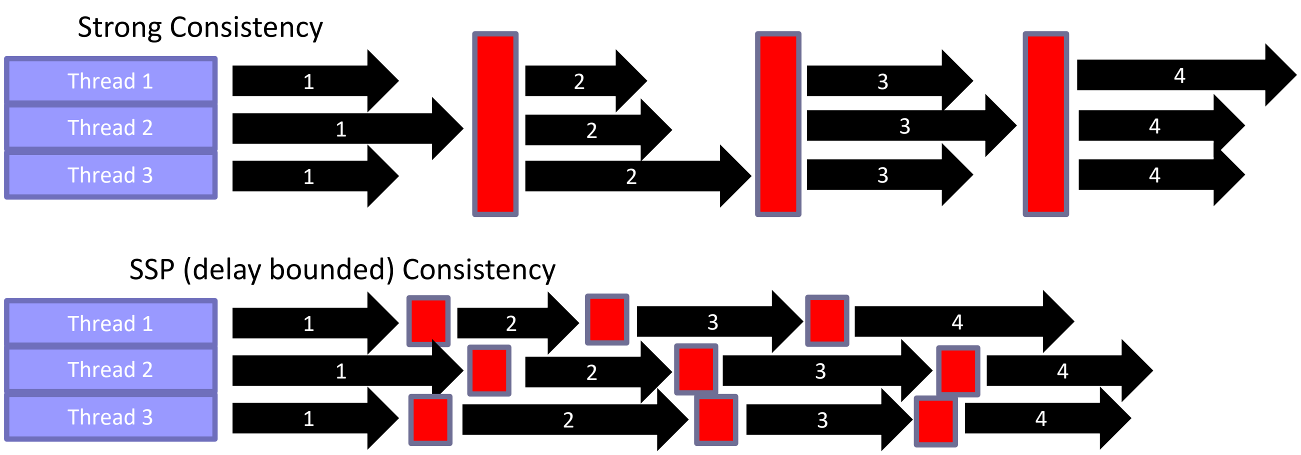




*Google(DistBlief), "*Large Scale Distributed Deep Networks*", In* Neural Information Processing Systems, 2012

3. Stale Synchronous SGD

- SSP consistency model
  - Error-tolerance property of training neural networks
  - Staleness threshold s defines the acceptance range for delays
    - changes no later than s iterations ago are guaranteed to be seen
    - readers may wait for stragglers if it is more than s iterations behind



Hao Zhang, et al., “Poseidon: A System Architecture for Efficient GPU-based Deep Learning on Multiple Machines”, 2015

4. 1-Bit SGD

- Idea: quantize the gradients aggressively—to but one bit per value—if the quantization error is carried forward across mini batches (error feedback)
  - This is a common technique in other areas, such as sigma-delta modulation for DACs (Delta-sigma modulation technique for digital-to-analog conv

$$
\begin{aligned}
G^{\text{quant}}_{ij\ell}(t) &= \mathcal{Q}(G_{ij\ell}(t) + \Delta_{ij\ell}(t-N)) \\
\Delta_{ij\ell}(t) &= G_{ij\ell}(t) - \mathcal{Q}^{-1}(G^{\text{quant}}_{ij\ell}(t))
\end{aligned}
$$

error

gradient parameter

quantizated values

  - As long as error feedback is used, we can quantize all the way to 1 bit at no or nearly no loss of accuracy.

4. 1-Bit SGD

- $$\begin{bmatrix} \boxed{0.16} & \cdots & w_{1,N} \\ \vdots & \ddots & \vdots \\ \boxed{0.7} & \cdots & w_{M,N} \end{bmatrix} \rightarrow \begin{bmatrix} \boxed{0} & \cdots & 1 \\ \vdots & \ddots & \vdots \\ \boxed{1} & \cdots & 0 \end{bmatrix}$$

  0.16 < 0.23

  0.7 > 0.23

  Effect: communication reduced by 32x

  Column avg. → 0.23

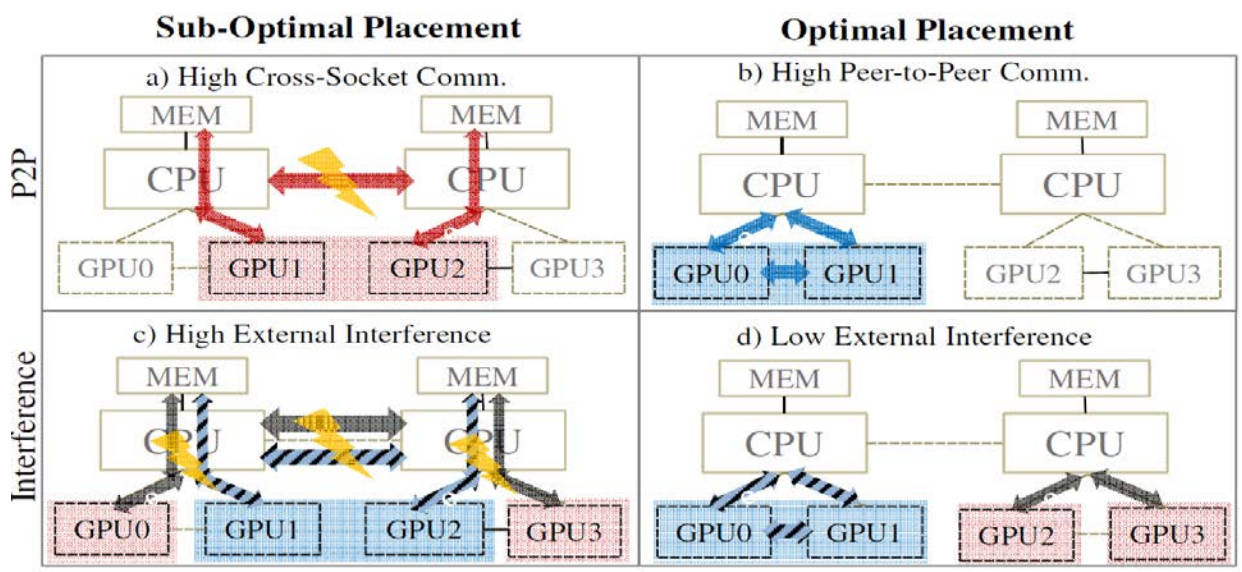
  Avg. is sent over network

- Results:
  - A 160M-parameter model training processes 3300h of data in under 16h on 20 dual-GPU servers—a 10 times speed-up—albeit at a small accuracy loss

Frank Seide, et al., "1-Bit Stochastic Gradient Descent and its Application to Data-Parallel Distributed Training of Speech DNNs", INTERSPEECH 2014

5. Task Placement Problem

- Task placement can significantly affect communication
  - Interference (Bandwidth/Resource contention)
  - Latency delay
  - Resource fragmentation



5. Task Placement Problem

- Pack:
  - Allocate GPUs from the same socket
  - Minimizing the distance between GPUs
  - Prioritize the performance of GPU-to-GPU comm.
- Spread:
  - Allocate GPUs from different sockets
  - Better resource utilization
  - Minimize resource fragmentations

Most systems choose Pack strategy to minimize communication overhead

Marcelo Amaral, et al., “Topology-Aware GPU Scheduling for Learning Workloads in Cloud Environments”, SC 2017

Principals of Distributed Training

- Tuning of batch size & learning rate
  - Use a larger batch size to increase computation / communication ratio
  - Linear scaling rule is a simple technique that scales the learning rate with the batch size linearly
  - Larger batch size could lead to loss in accuracy
- Choose of parallelism
  - Data parallelism across nodes, model parallelism across devices (GPU)
- Choose of communication method:
  - Sync vs. Stale Sync vs. Async; P2P vs. De-centralized
  - PS/Worker ratio
- Resource binding & scheduling
  - Aware of physical network topology

Outline

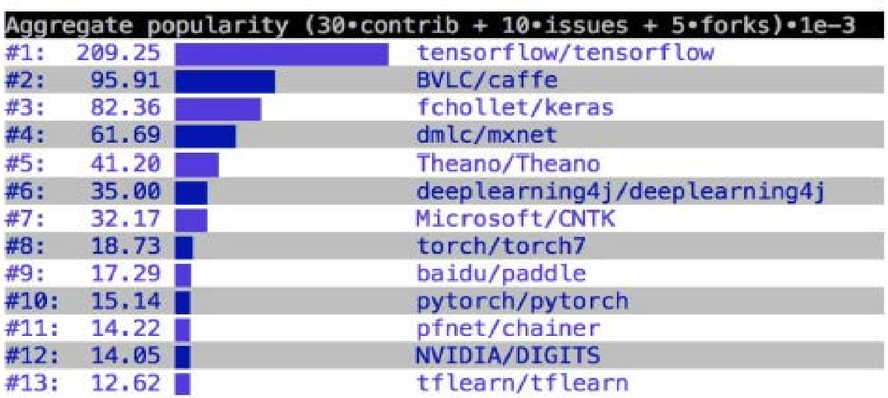
- Brief Introduction of Deep Learning
  - Computing Demand for Training
  - GPU Solutions
- Distributed Deep Learning Computations
  - Parallel strategies
  - Optimization strategies
- Distributed Deep Learning Frameworks
  - TensorFlow & Horovod
- Trend & Future of Deep Learning Computing
  - ML Systems & AutoML
  - Edge computing, CS-1 machine & AI Chips
  - Federated Learning
  - Remarks

Distributed Framework Implementations

Framework	Organization	Model Parallelism	Data Parallelism	GPU	Source
SparkNet	UCB	No	Yes	Yes	https://github.com/amplab/SparkNet
Caffe-MPI	China Inspur	No	Yes	Yes	https://github.com/Caffe-MPI/Caffe-MPI.github.io
MPI-Caffe	VT, U. Indiana	Yes	No	Yes	https://computing.ece.vt.edu/~steflee/mpi-caffe.html
Poseidon (Petuum)	CMU	No	Yes	Yes	https://github.com/petuum/poseidon
COTS HPC	Google	Yes	No	Yes	N.A.
DistBelief	Google	Yes	Yes	No	N.A.
CNTK	Microsoft	Yes	Yes	Yes	https://github.com/Microsoft/CNTK/wiki
Project Adam	Microsoft	Yes	Yes	No	N.A.
Theano	U. Montreal	Yes	Exp (Platoon)	Yes	http://deeplearning.net/software/theano/introduction.html
TensorFlow	Google	Yes	Yes	Yes	https://www.tensorflow.org/
MXNET	CMU, UW, etc.	Yes	Yes	Yes	https://github.com/dmlc/mxnet

TensorFlow

- **Google's 2nd-generation system** for the implementation and deployment of **largescale machine learning models**
- Takes computations described using a **dataflow-like model** and **maps them onto a wide variety of different hardware platforms**
  - ranging from running inference on mobile device platforms to training on GPU clusters
- Simplify the real-world use of ML system.
- One of the most popular frameworks

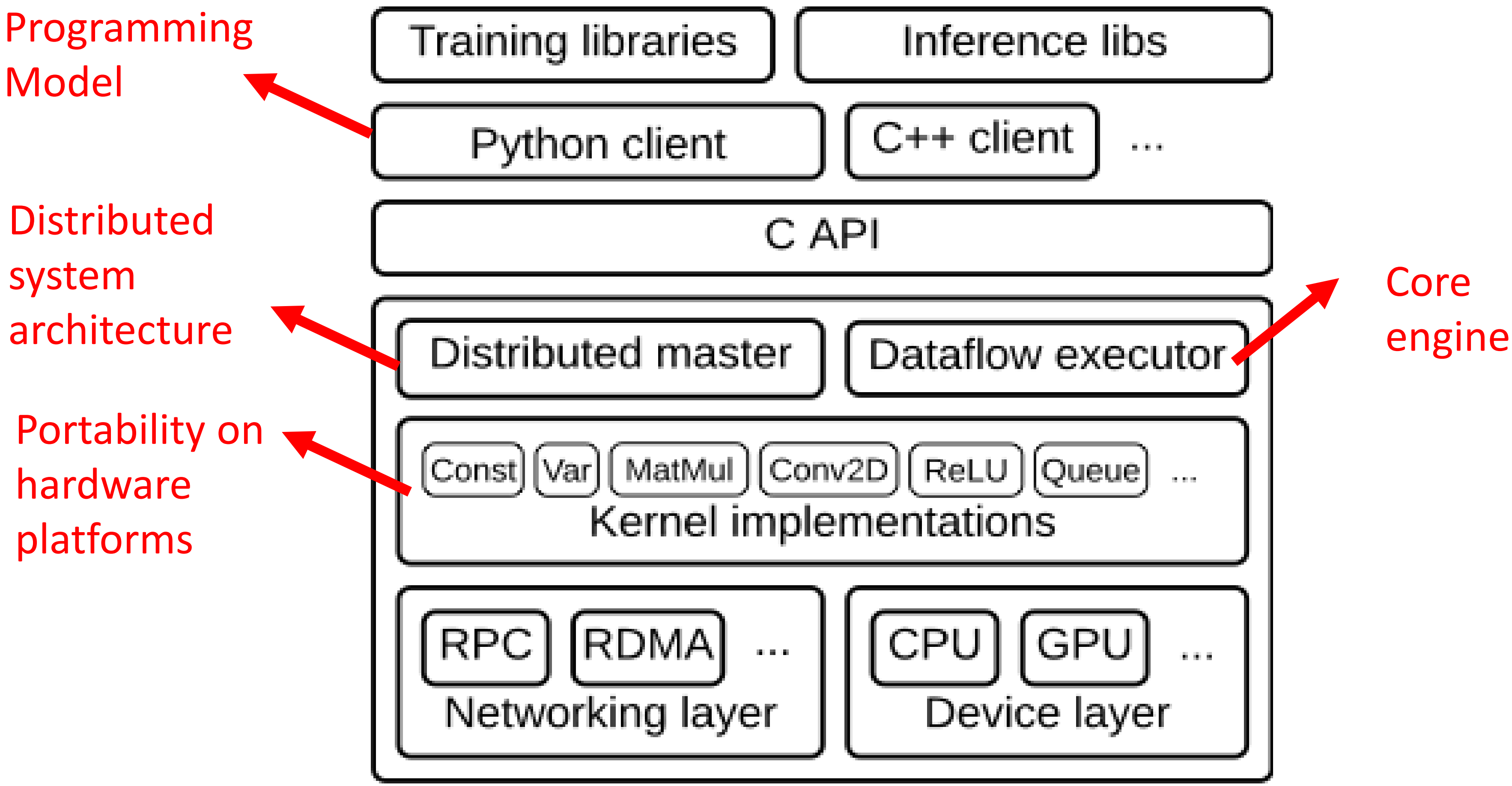


François Chollet, Google Developer

[22] "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems", Preliminary White Paper, November 9, 2015

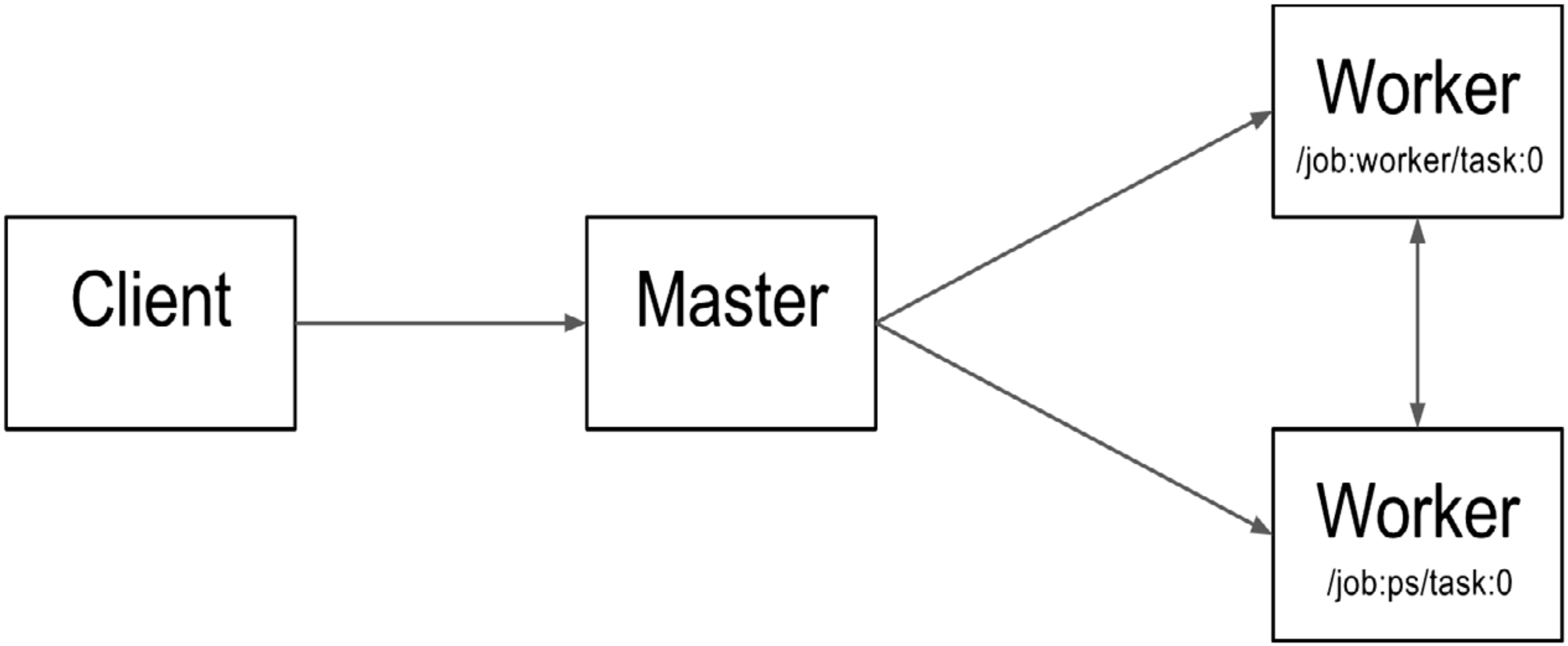
TensorFlow Runtime

- TensorFlow runtime is a cross-platform library



Distributed TensorFlow System Architecture

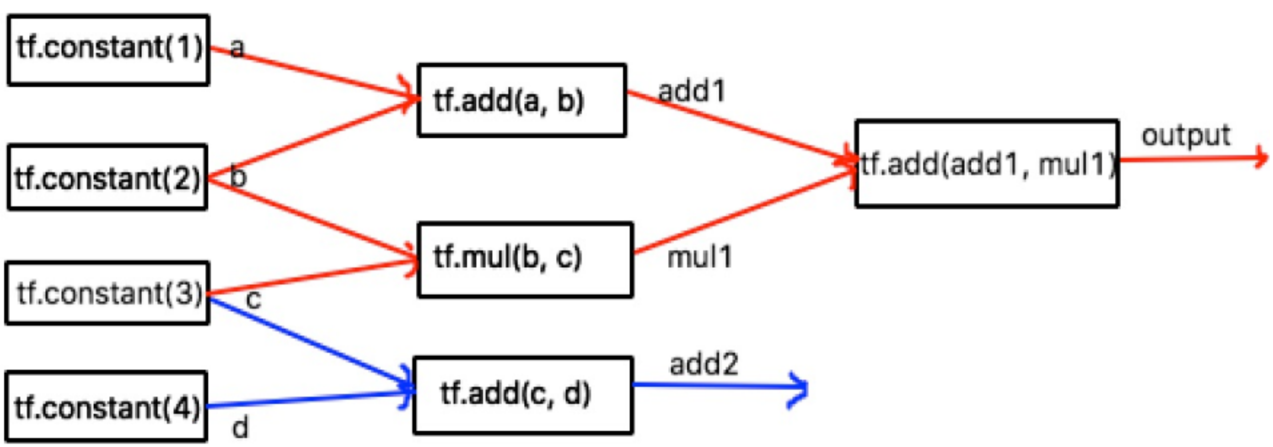
- Distributed Master and Worker Service only exist in distributed TensorFlow.



Distributed TensorFlow: Client

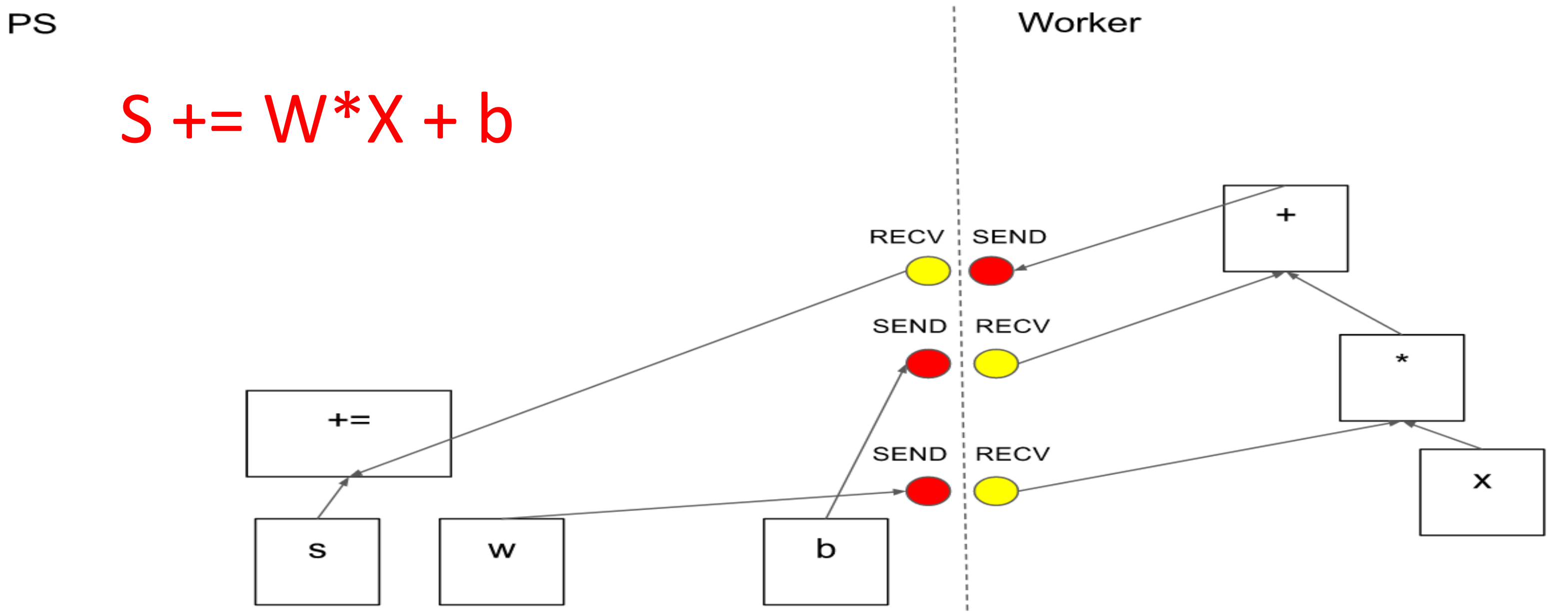
- Users write the client TensorFlow program that **builds the computation graph**
  - Computation is described by a directed graph
  - A tensor is a typed multi-dimensional array (**ephemeral** by default)
  - Variables is a special operation that returns a handle to a **persistent mutable tensor** that survives across executions

```python
# coding=utf-8
import tensorflow as tf

a = tf.constant(1)
b = tf.constant(2)
c = tf.constant(3)
d = tf.constant(4)
add1 = tf.add(a, b)
mul1 = tf.multiply(b, c)
add2 = tf.add(c, d)
output = tf.add(add1, mul1)

with tf.Session() as sess:
    print sess.run(output)
```



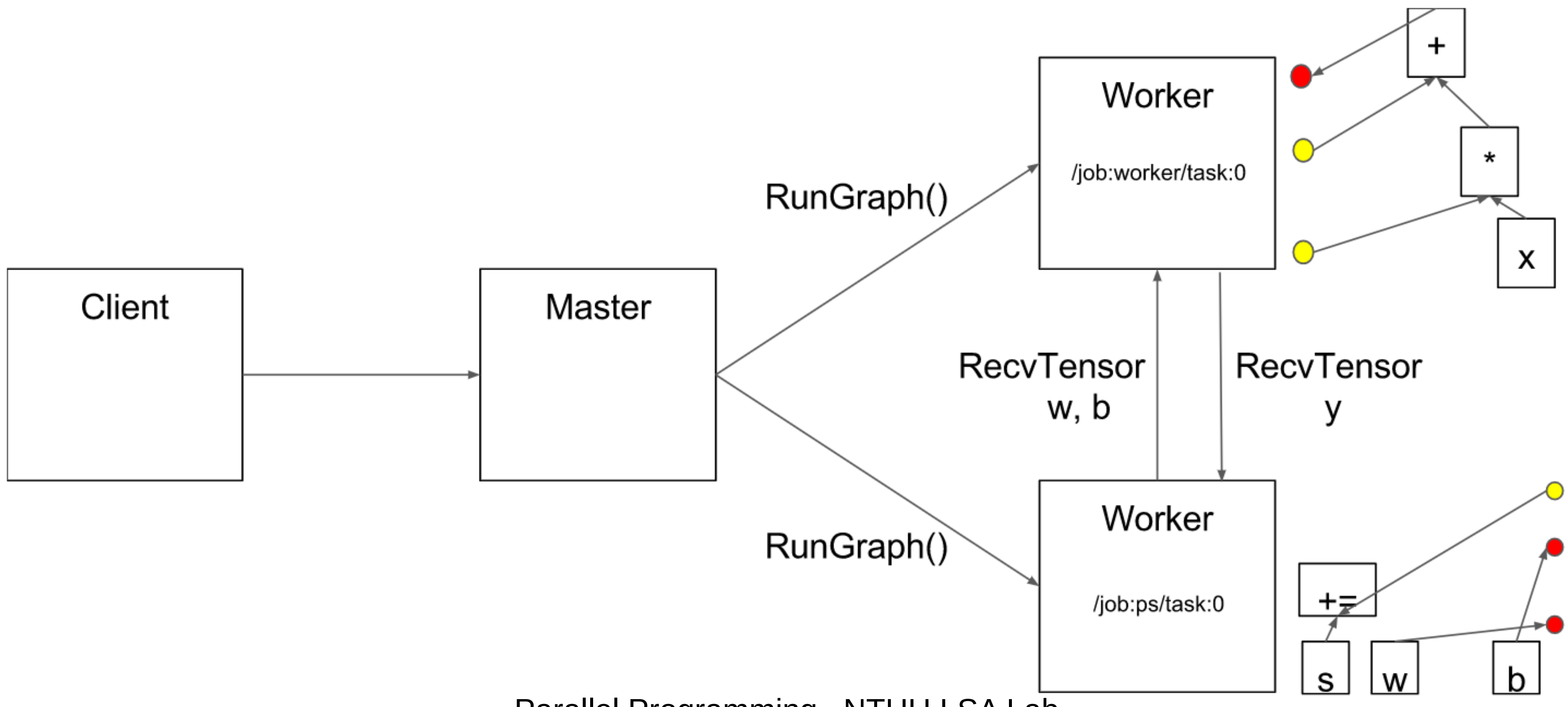
Distributed TensorFlow: Master

- Prunes the graph to obtain the subgraph **required to evaluate the nodes requested by the client**
- Partitions the graph to obtain graph pieces for each participating **device**
  - **Send/Recv OPs** are added by TF to transfer tensors



Distributed TensorFlow: Worker

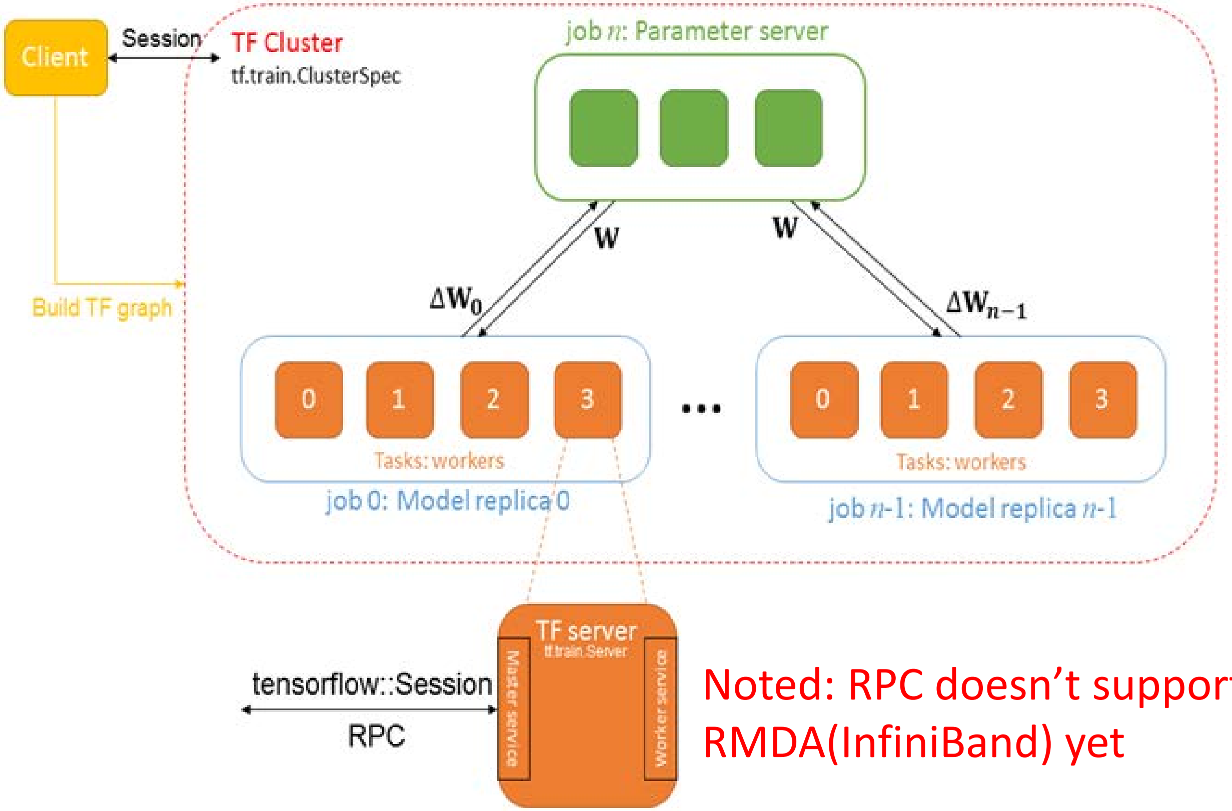
- Handles requests from the master
- Schedules the execution of the kernels for the operations that comprise a local subgraph
- Mediates direct communication between tasks



Distributed TensorFlow: Cluster Spec

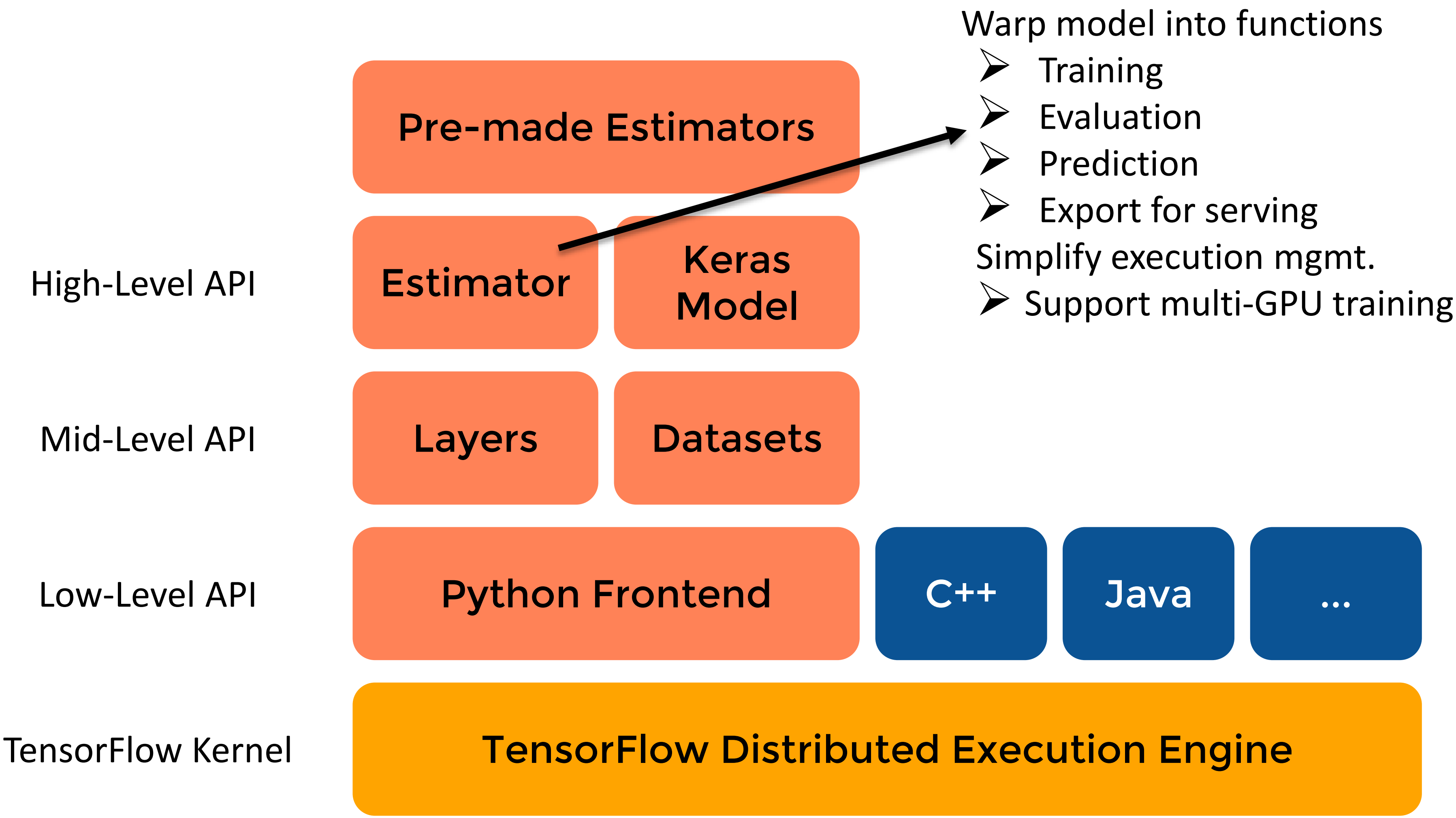
- User must specify the role of master and worker in a cluster spec

```
tf.train.ClusterSpec({
    "worker": [
        "worker0.example.com:2222",
        "worker1.example.com:2222",
        "worker2.example.com:2222"
    ],
    "ps": [
        "ps0.example.com:2222",
        "ps1.example.com:2222"
    ]})
```



Noted: RPC doesn't support RMDA(InfiniBand) yet

TensorFlow Architecture

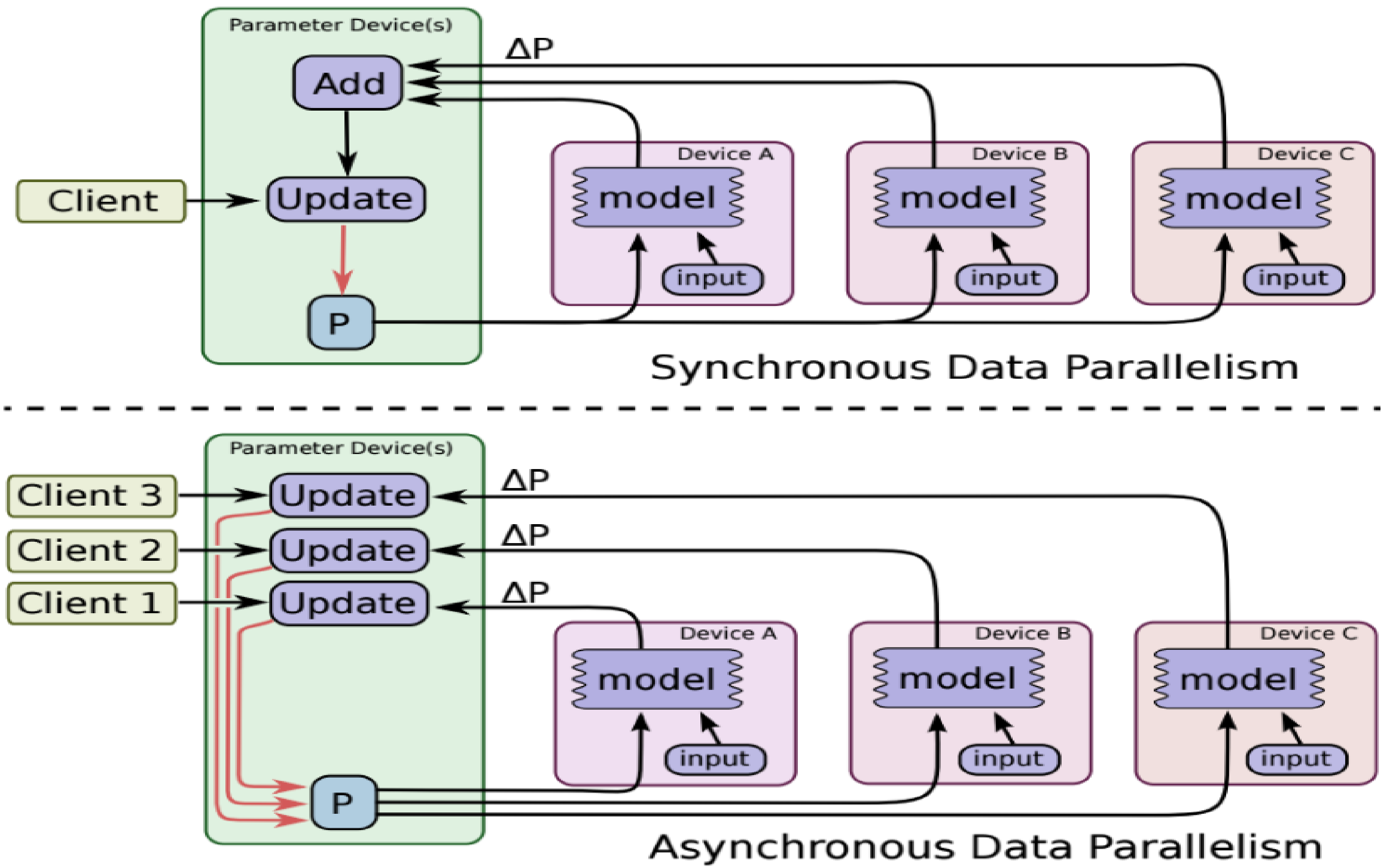


Build-in Strategy for TF Estimator

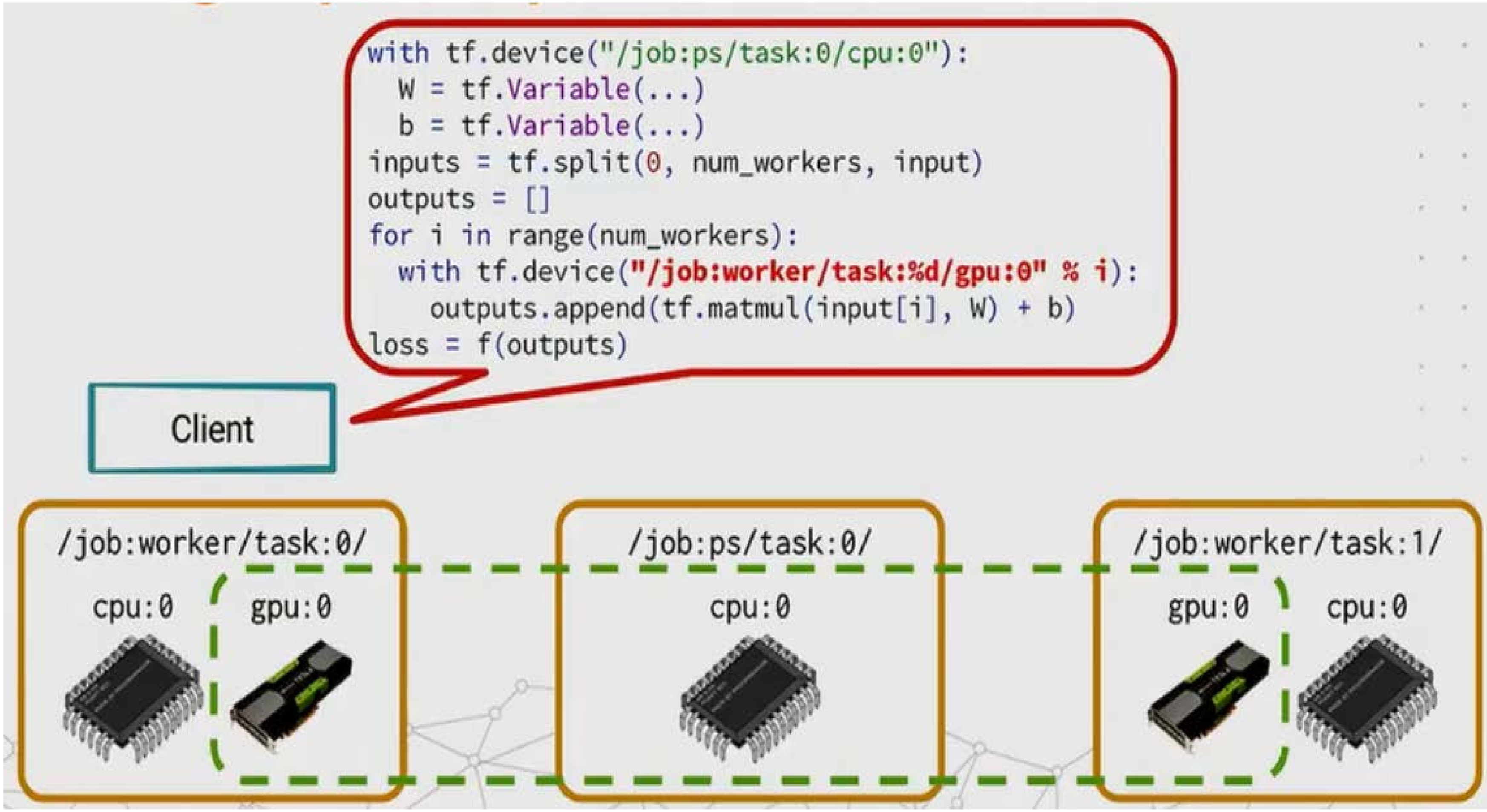
- These strategies can be called from Keras API as well

Strategy	Parallelism	Dataset	Comm.	Use Scenario
Mirrored	Single node	Replicated on GPU devices	Allreduce	Data parallelism on a single node
Central Storage	Single node	Keep on host (main memory)	Parameter server	When model is small
Multi Worker	Multi nodes	Replicated on GPU devices	Allreduce	HPC Env. (similar to Horovod)
Parameter Server	Multi nodes	Replicated on GPU devices	Parameter server	Cloud Env. with heterogeneous computing power and unreliable connection

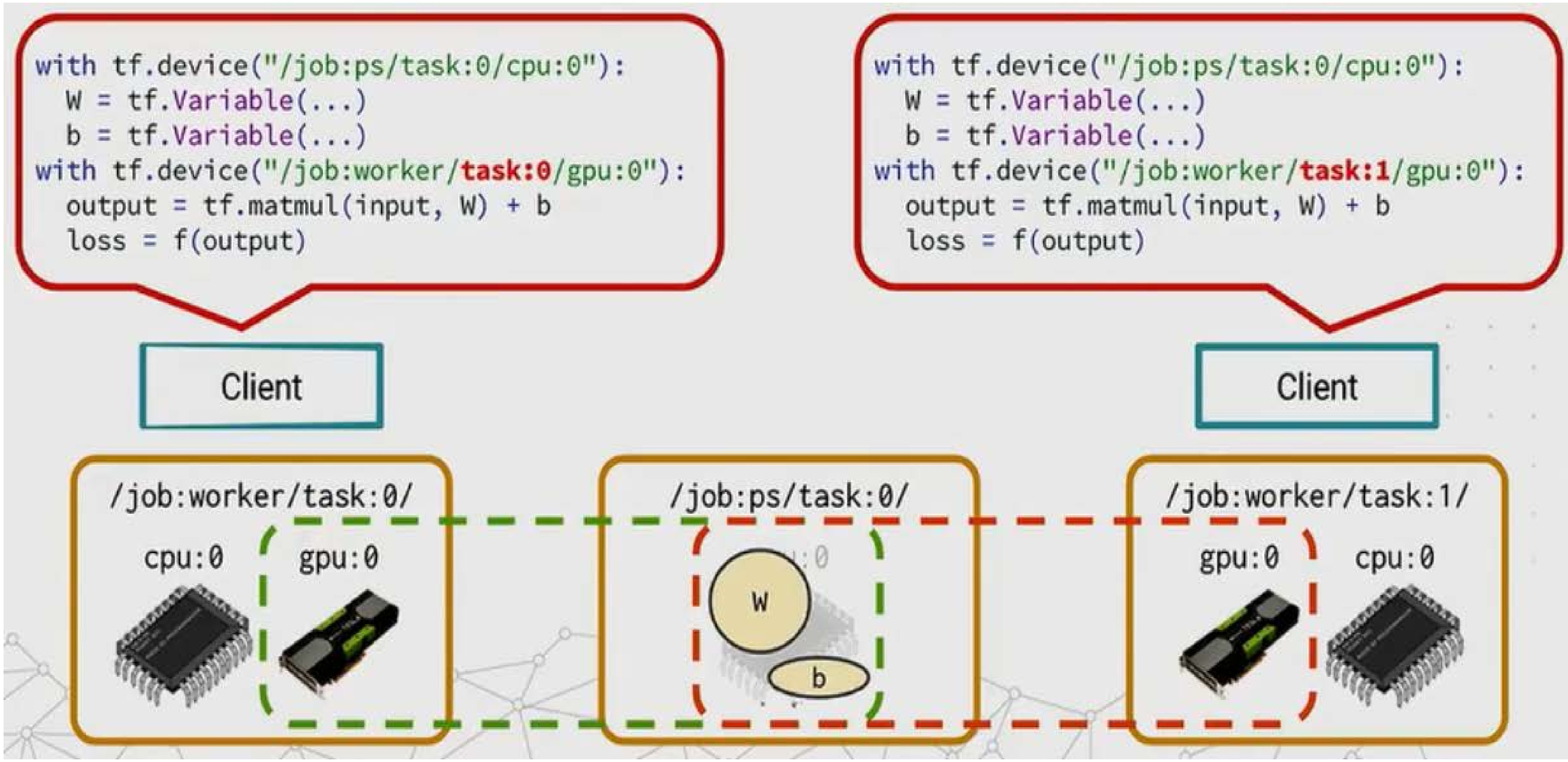
Synchronous vs. Asynchronous Training



In Graph Replication



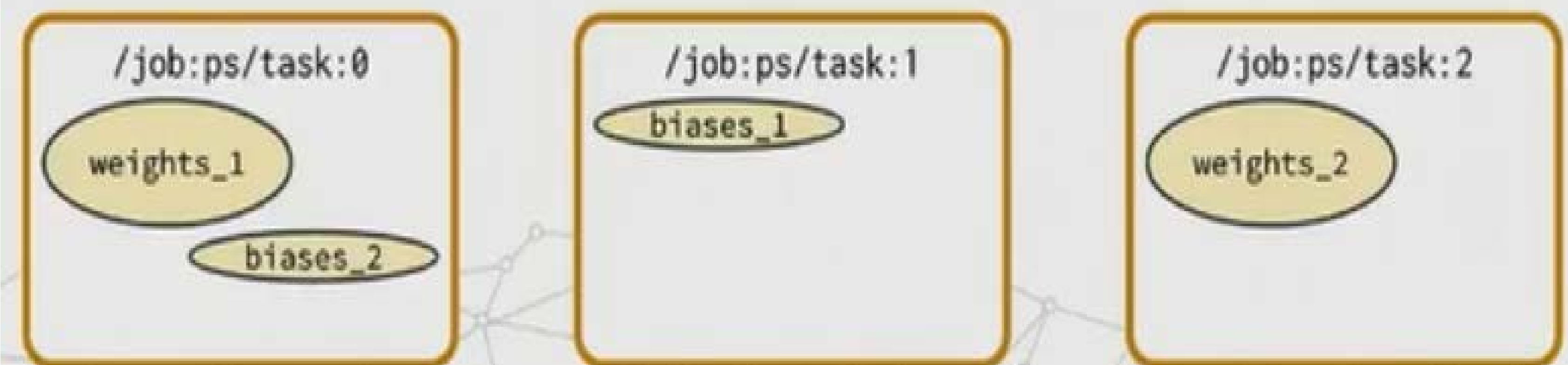
Between Graph Replication



Between graph is more commonly used because its reduce the size of model, and requires no change to the model in user program

Round-Robin Variables on Multiple PS Servers

```
with tf.device(tf.train.replica_device_setter(ps_tasks=3)):

  weights_1 = tf.get_variable("weights_1", [784, 100])
  biases_1 = tf.get_variable("biases_1", [100])
  weights_2 = tf.get_variable("weights_2", [100, 10])
  biases_2 = tf.get_variable("biases_2", [10])
```

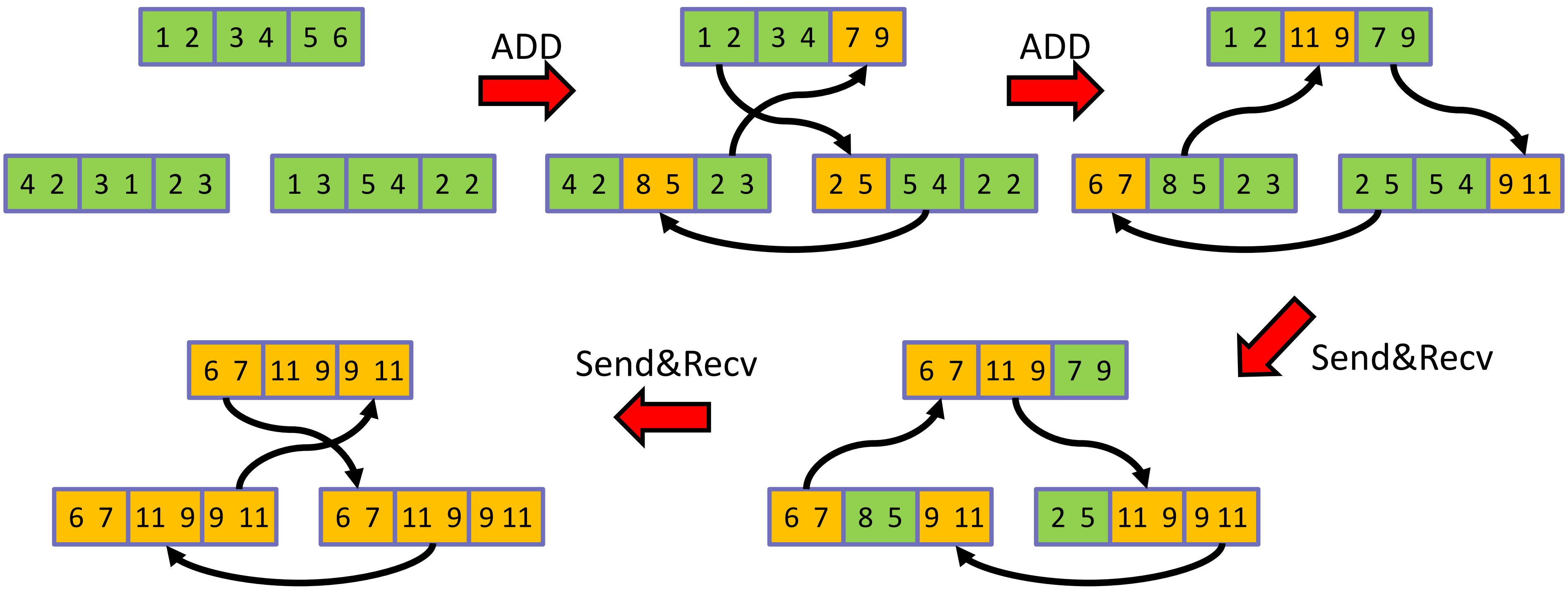


Horovod

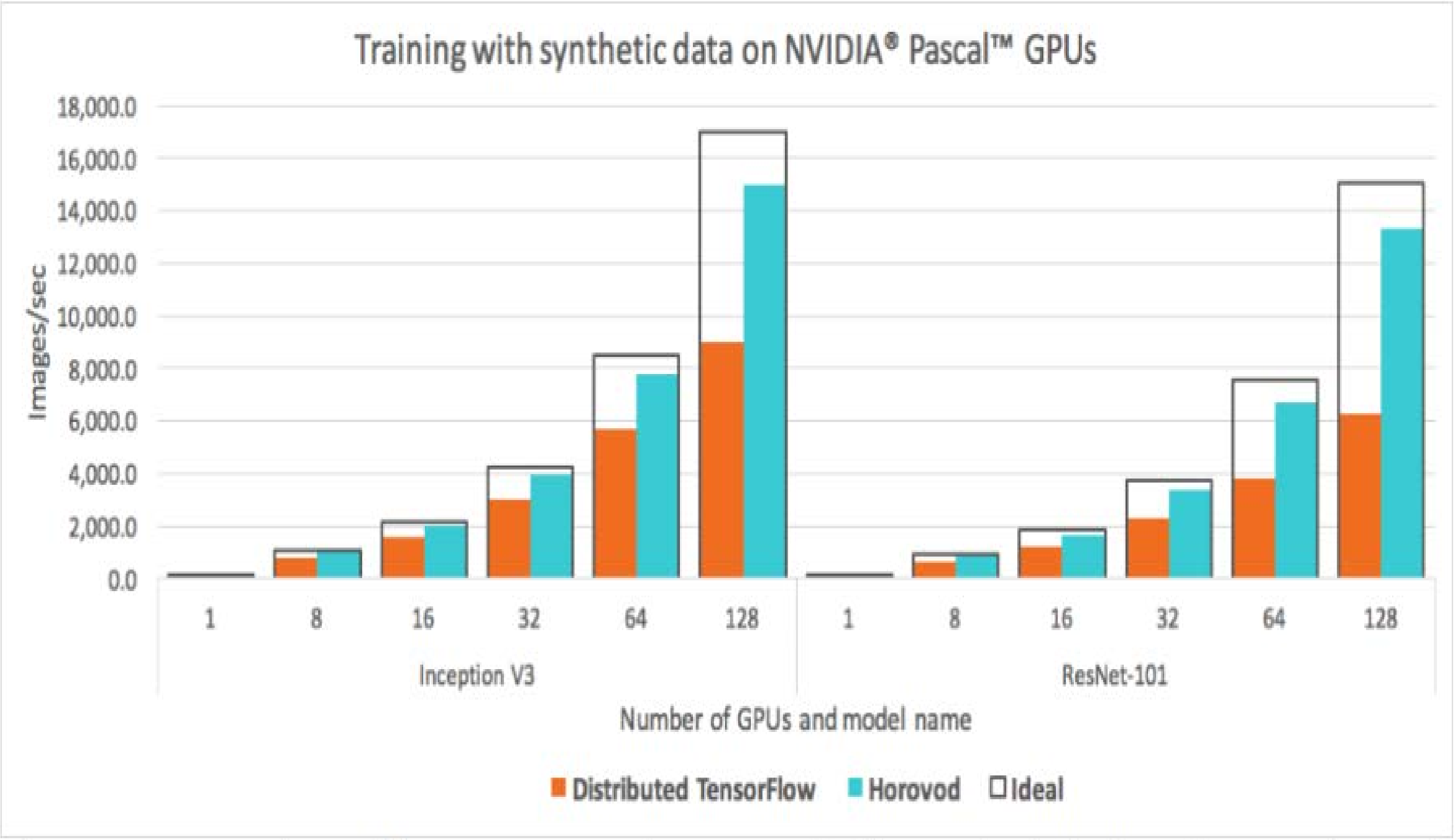
- Distributed training framework for
  - TensorFlow
  - Keras
  - PyTorch
- Separate infrastructure from ML computations
  - Executed like a traditional HPC parallel job
- Use bandwidth-optimal communication protocols
  - Implemented by HPC protocols: **MPI** and NVIDIA Collective Communications Library (NCCL)
  - Utilize RDMA (InfiniBand) if available
- Named after traditional Russian folk dance where participants dance in a circle with linked hands
- Introduction clip from UBER

Horovod: Ring Allreduce

- An allreduce implementation that can full utilize P2P network bandwidth
  - 2*(N-1) iterations: N-1 Adds, N-1 Send&Recv



Horovod



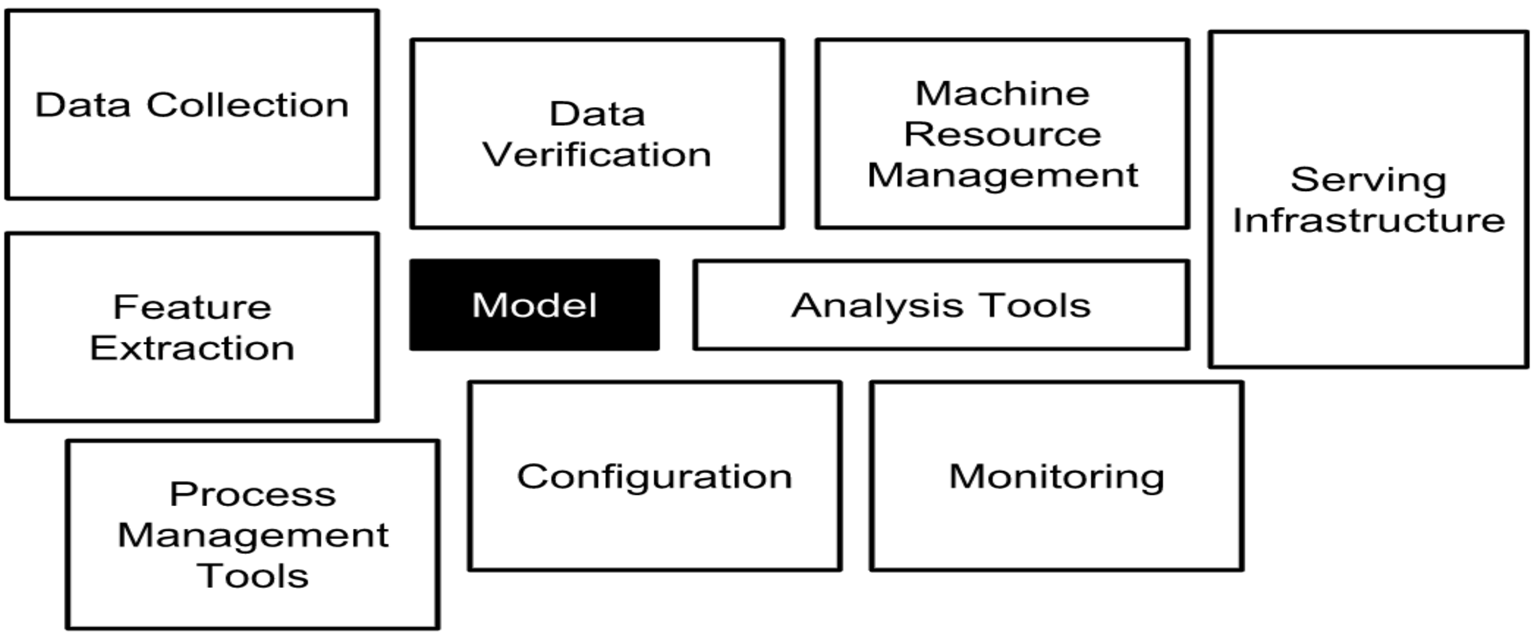
Training with synthetic data on NVIDIA® Pascal™ GPUs
Images/sec
18,000.0
16,000.0
14,000.0
12,000.0
10,000.0
8,000.0
6,000.0
4,000.0
2,000.0
0.0
1
8
16
32
64
128
Inception V3
1
8
16
32
64
128
ResNet-101
Number of GPUs and model name
Distributed TensorFlow
Horovod
Ideal

Outline

- Brief Introduction of Deep Learning
  - Computing Demand for Training
  - GPU Solutions
- Distributed Deep Learning Computations
  - Parallel strategies
  - Optimization strategies
- Distributed Deep Learning Frameworks:
  - TensorFlow & Horovod
- Trend & Future of Deep Learning Computing
  - ML Systems & AutoML
  - Edge computing, CS-1 machine & AI Chips
  - Federated Learning
  - Remarks

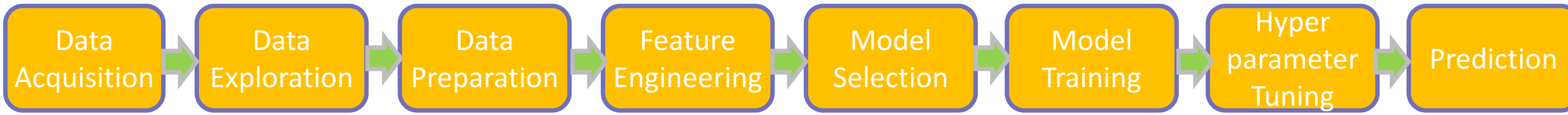
ML System

- There is a lot more to AI/ML than just implementing an **algorithm** or a **technique**
- We need **a system** to support, optimize, and automate the whole process

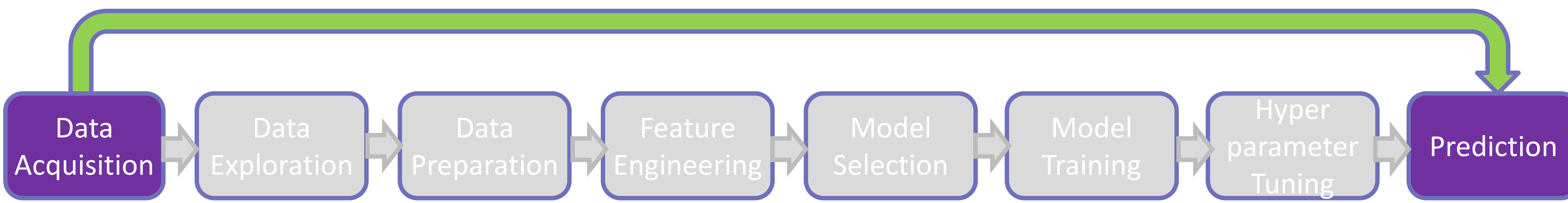


Hyper-parameter tuning & AutoML

- Traditional Machine Learning Workflow



- AutoML Workflow
  - Let users focus on data acquisition & prediction only
  - E.g.: Google Cloud AutoML, Microsoft Custom Vision, Auto-Keras
  - But automation often demands even faster processing speed



End-to-End Deep Learning Lifecycle

	Cloud/HPC/Data center	Edge/Embedded
Training	• High Performance • High Precision • Distributed in Large Scale	• Collaborated Learning • (Federated Learning) • Data Privacy
Inference	• High Throughput • Low Latency • Distributed & Scalable	• Moderate Throughput • Low Latency • Power Efficiency • Low Cost

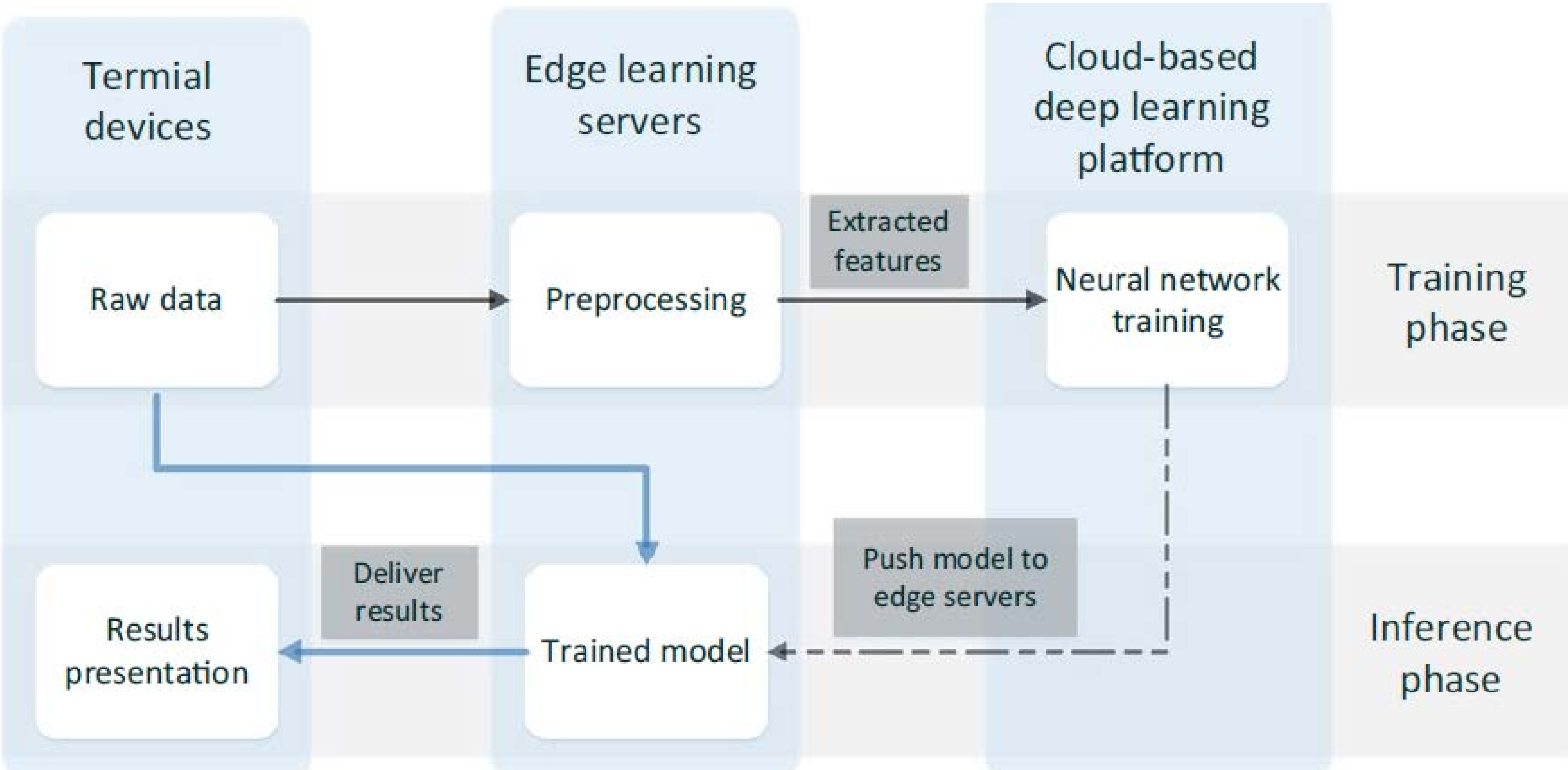
HPC (GPU)

Edge node

Cloud services
(AI Chip: ASIC)

Embedded
(AI Chip: ASIC, FPGA)

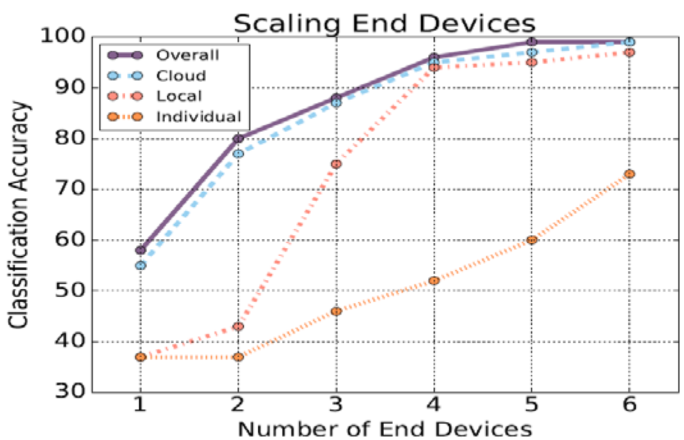
End-to-End Deep Learning Lifecycle

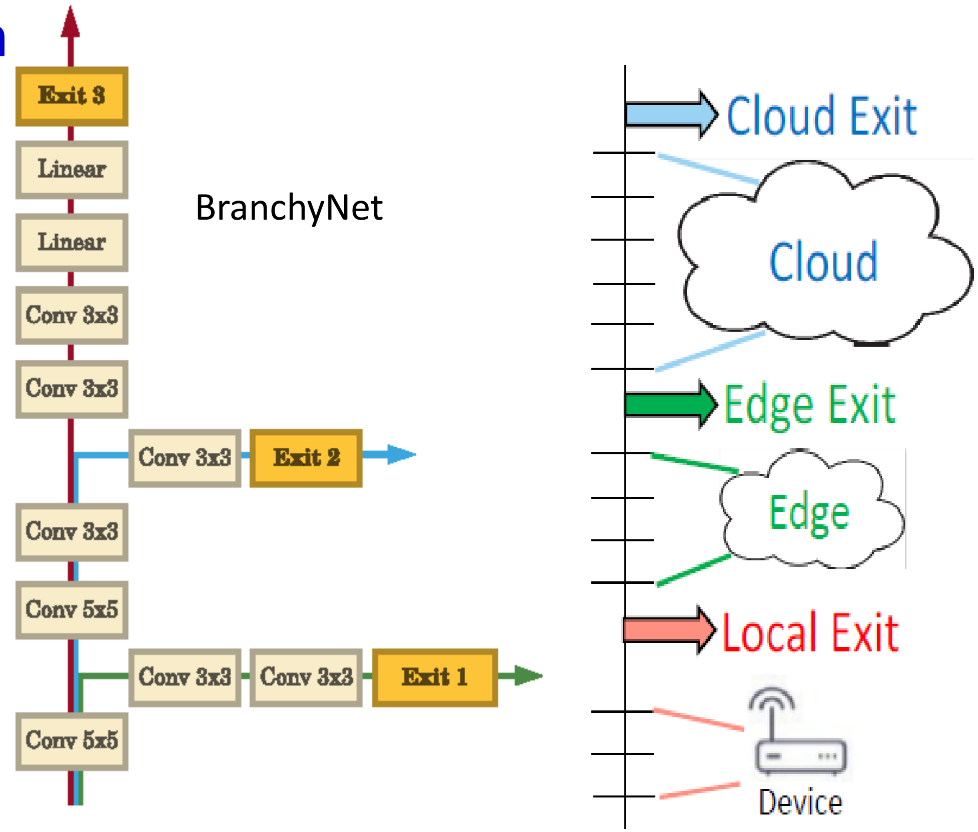


Source: Exploiting the edge power: an edge deep learning framework, CCF Transactions on Networking 2018

Model Parallelism for Inference on Edge

- Strike the balance between accuracy and latency delay
- Improve accuracy by aggregating results from multiple devices

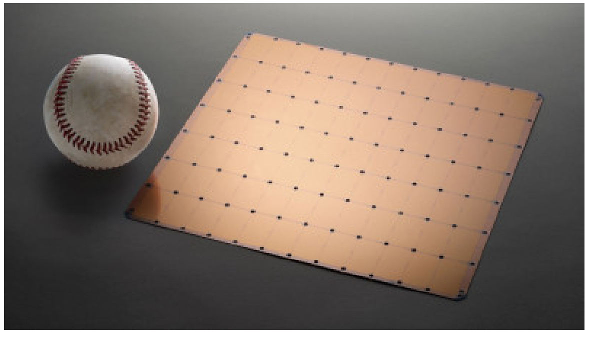




S. Teerapittayanon, B. McDanel and H. T. Kung, "Distributed Deep Neural Networks Over the Cloud, the Edge and End Devices," *2017 ICDCS*, pp. 328-339.

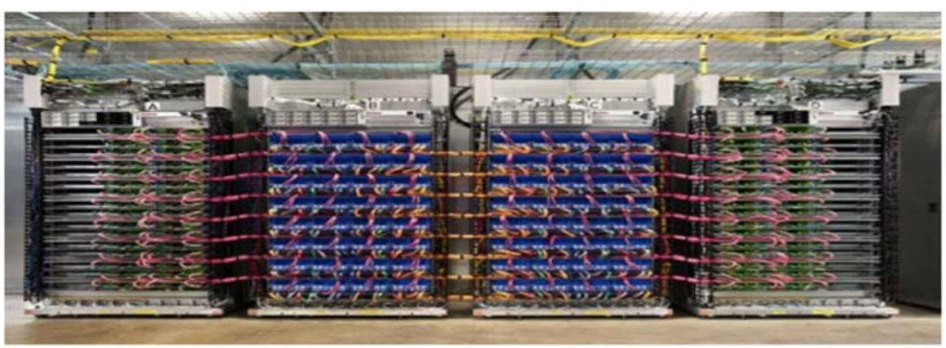
CS-1: World Fastest AI Machine

- Achieve 100- to 1,000-fold improvement over existing AI accelerators
  - Just announced in the Supercomputing Conference last month (Nov. 2019)
  - Going to be deployed in Argonne National Lab
- Made possible by Wafer Scale Engine (WSE)
  - The largest chip ever made at 46,225 square millimeters in area, it is 56.7 times larger than the largest graphics processing unit.
  - 78 times more AI optimized compute cores, 3,000 times more high speed, on-chip memory, 10,000 times more memory bandwidth



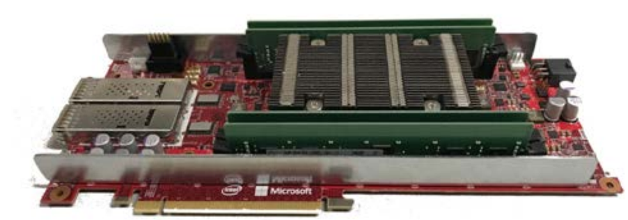
AI Chip for Inference

- Co-design of the network structure and hardware architecture
  - AI Chip: dedicated “Tensor Accelerator”, like TensorCore
- Trade accuracy for energy & cost saving
  - model reduction, low precision computations
- Domain-specific, rather than application-specific
  - A new chip can be used more broadly across multiple applications by reconfiguration



Google Cloud TPU Pod (Hot Chips 2017)

Google’s TPU POD (ASIC)



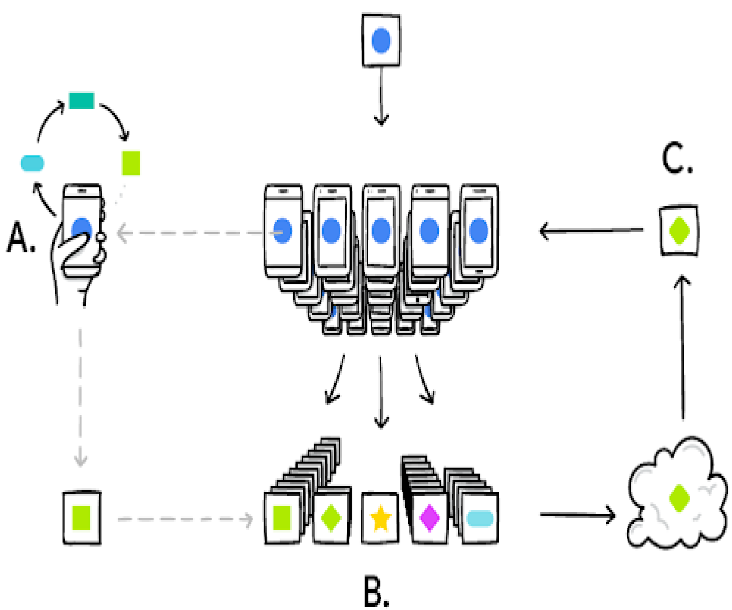
Microsoft's BrainWave (FPGA)

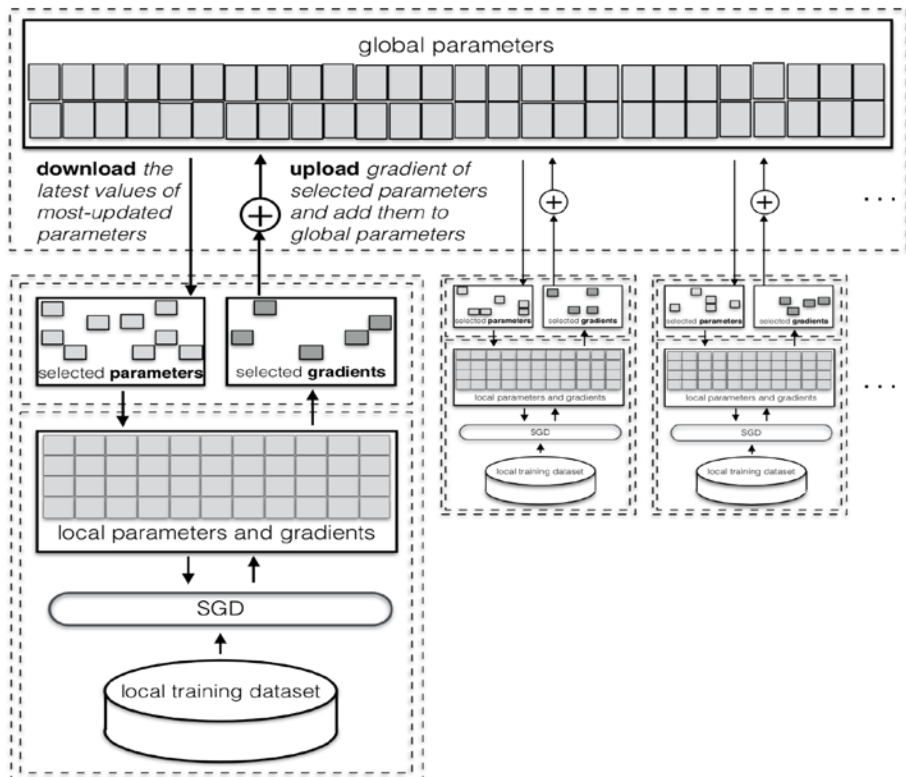
AI Chip for Inference

- "memory wall" problem
  - Increase the capacity of the on-chip memory and brings it closer to the computing units
  - Compute-capable memory referred to as processing-in-memory (PIM)
- Lack of general software toolchain to efficiently translate different machine learning tasks and neural networks into executable binary codes, running on the AI chips
  - Neural network pruning, weight compression and dynamic quantization

Federated Learning

- Training the model with local data
- Preserving data privacy





Remarks

- Future Trend
  - Distributed Deep Learning (even Federated Learning)
  - Extending cloud services to edge or even devices
  - AI Chip Design
  - Development of AutoML & ML Solutions
- Greatest Challenge: **Scalable AI solutions**
  - Reproducible results
  - Generalized strategies
  - Automated process

Reference

[1] https://developer.nvidia.com/gpudirect

[2] http://www.nvidia.com.tw/object/nvlink-tw.html

[3] http://www.nvidia.com/object/deep-learning-system.html

[4] http://timdettmers.com/2014/10/09/deep-learning-data-parallelism/

[5] http://timdettmers.com/2014/11/09/model-parallelism-deep-learning/

[6] https://www.coursera.org/learn/machine-learning/lecture/9zJUs/mini-batch-gradient-descent

[7] Jeffrey Dean and Sanjay Ghemawat, "MapReduce: simplified data processing on large clusters", Commun. ACM 51, 1 (January 2008), 107-113.

[8] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In USENIX NSDI, 2012.

[9] Low, Y., Gonzalez, J., Kyrola, A., Bickson, D., Guestrin, C., and Hellerstein, J., "Distributed GraphLab: A framework for machine learning in the cloud". In VLDB, 2012.

[10] Grzegorz Malewicz, Matthew H. Austern, Aart J.C Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In SIGMOD '10.

Reference

[11] Krizhevsky, Alex. “One weird trick for parallelizing convolutional neural networks.” CoRR abs/1404.5997, 2014 (arXiv:1404.5997).

[12] Mu Li, Tong Zhang, Yuqiang Chen, and Alexander J. Smola. 2014. Efficient mini-batch training for stochastic optimization. In ACM SIGKDD, pages 661-670, 2014.

[13] Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Ranzato, M., Senior, A., Tucker, P., Yang, K., Le, Q., and Ng, A. 2012. Large Scale Distributed Deep Networks. In Advances in Neural Information Processing Systems. NIPS’12.

[14] Hao Zhang, Zhiting Hu, Jinliang Wei, Pengtao Xie, Gunhee Kim, Qirong Ho, Eric Xing, "Poseidon: A System Architecture for Efficient GPU-based Deep Learning on Multiple Machines", In USENIX Annual Technical Conference (ATC 2016)

[15] Frank Seide, Hao Fu, Jasha Droppo, Gang Li, Dong Yu, "1-Bit Stochastic Gradient Descent and its Application to Data-Parallel Distributed Training of Speech DNNs", In Interspeech 2014

[16] P Xie, JK Kim, Y Zhou, Q Ho, A Kumar, Y Yu, E Xing, "Lighter-Communication Distributed Machine Learning via Sufficient Factor Broadcasting", In Conference on Uncertainty in Artificial Intelligence, 2016

Reference

[17] Yongqiang Zou, Xing Jin, Yi Li, Zhimao Guo, Eryu Wang, and Bin Xiao. 2014. Mariana: tencent deep learning platform and its applications. Proc. VLDB Endow. 7, 13 (August 2014), 1772-1777.

[18] Le, Q., Ranzato, M., Monga, R., Devin, M., Chen, K., Corrado, G., Dean, J., and Ng, A, "Building high-level features using large scale unsupervised learning". ICML, 2012.

[19] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. 2009. ImageNet: A Large-Scale Hierarchical Image Database. In CVPR '09.

[20] A. Coates, B. Huval, T. Wang, D.-J. Wu, and A.-Y. Ng, "Deep Learning with COTS HPC Systems," ICML, 2013.

[21] Trishul Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman, "Project Adam: building an efficient and scalable deep learning training system", In Proceedings of OSDI, pages 571-582, 2014.

[22] Mart´ın Abadi, et al., "TensorFlow: Large-scale machine learning on heterogeneous systems", Preliminary White Paper, November 9, 2015.

[23] Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Fredo Durand, and Saman Ama- ´rasinghe. Halide: A language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. ACM SIGPLAN Notices, 48(6):519–530, 2013.

Reference

[24] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In ACM SIGOPS Operating Systems Review, volume 41, pages 59–72. ACM, 2007.

[25] Craig Chambers, Ashish Raniwala, Frances Perry, Stephen Adams, Robert R Henry, Robert Bradshaw, and Nathan Weizenbaum. FlumeJava: easy, efficient data-parallel pipelines. In ACM Sigplan Notices, volume 45, pages 363–375. ACM, 2010.

[26] Derek G. Murray, Malte Schwarzkopf, Christopher Smowton, Steven Smit, Anil Madhavapeddy, and Steven Hand. Ciel: a universal execution engine for distributed data-flow computing. In USENIX NSDI, 2011.

[27] Derek G Murray, Frank McSherry, Rebecca Isaacs, Michael Isard, Paul Barham, and Mart´ın Abadi. Naiad: a timely dataflow system. In ACM Symposium on Operating Systems Principles, pages 439–455. ACM, 2013.

[28] Christopher J Rossbach, Yuan Yu, Jon Currey, JeanPhilippe Martin, and Dennis Fetterly. Dandelion: a compiler and runtime for heterogeneous systems. In ACM Symposium on Operating Systems Principles, pages 49–68. ACM, 2013.