Chap11: Distributed & Parallel Computing for Deep Learning

National Tsing Hua University

2023 Fall Semester

Outline

- Brief Introduction of Deep Learning
  - Computing Demand for Training
  - GPU Solutions
- Distributed Deep Learning Computations
  - Parallel strategies
  - Optimization strategies
  - Memory Optimization
  - TensorFlow & Horovod
- Trend & Future of Deep Learning Computing
  - ML Systems & MLOps
  - AI Cloud Services & 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
  - Memory Optimization
  - TensorFlow & Horovod
- Trend & Future of Deep Learning Computing
  - ML Systems & MLOps
  - AI Cloud Services & AI Chips
  - Federated Learning
  - Remarks

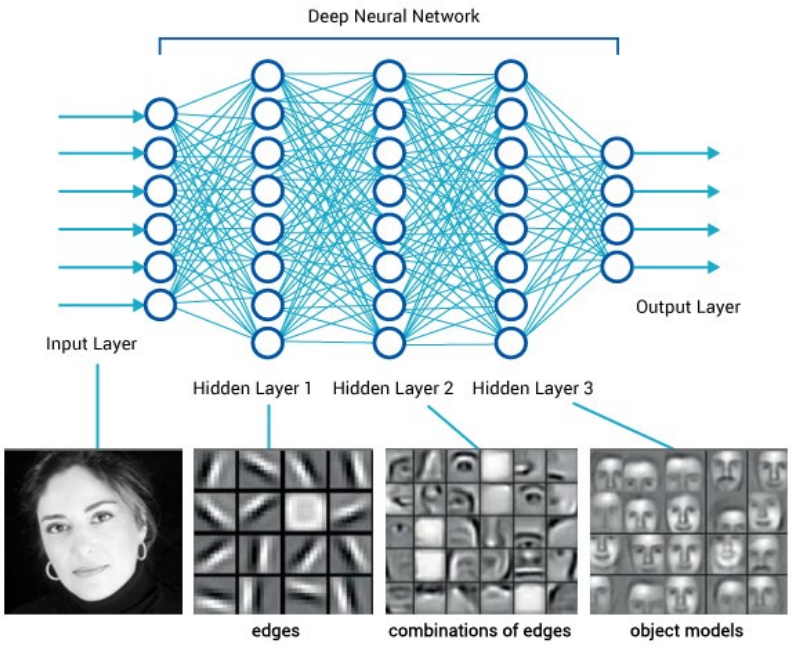
What is Deep Learning?

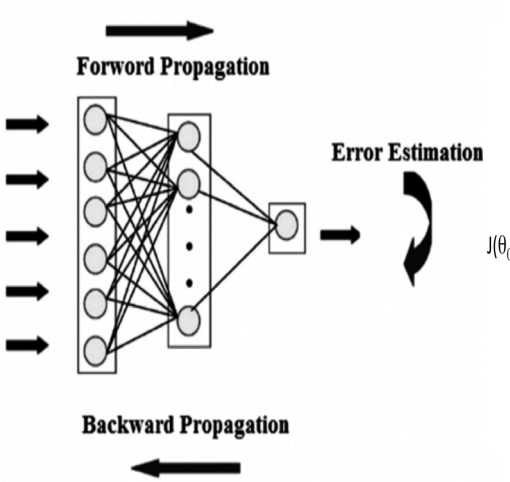
- **AI**: it emphasizes the creation of intelligent machines that work and react like humans
- **Machine Learning**: it provides systems the ability to automatically learn and improve from experience without being explicitly programmed
- **Deep Learning**: a subfield of machine learning concerned with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

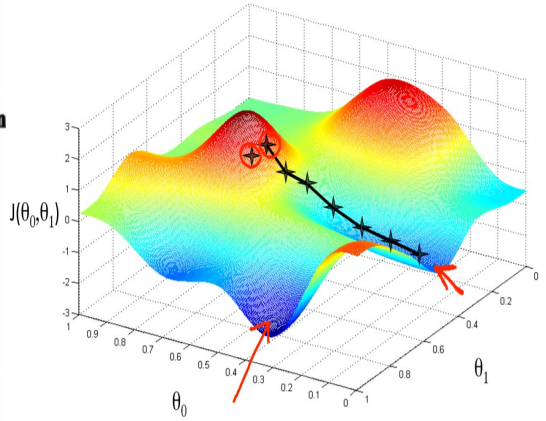
Since an early flush of optimism in the 1950s, smaller subsets of artificial intelligence – first machine learning, then deep learning, a subset of machine learning – have created ever larger disruptions.

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

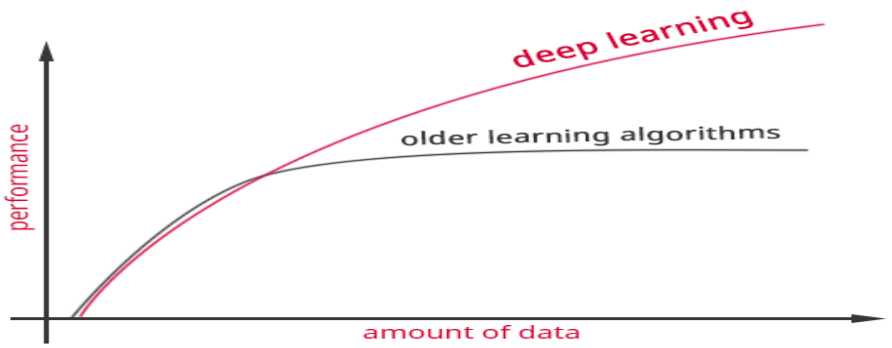


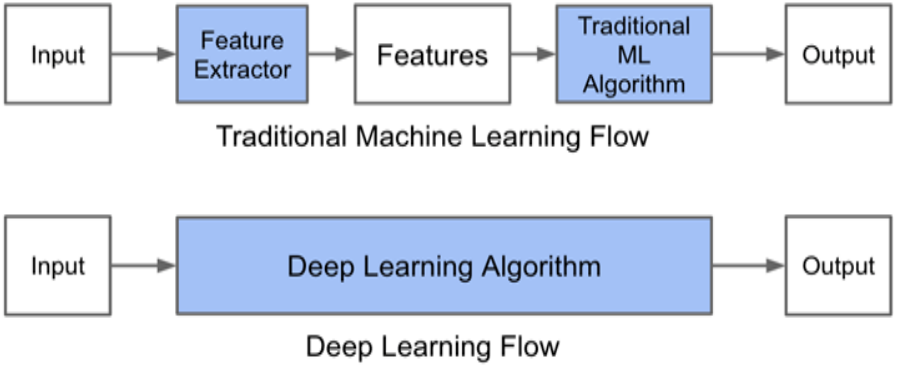




Strengths of DL

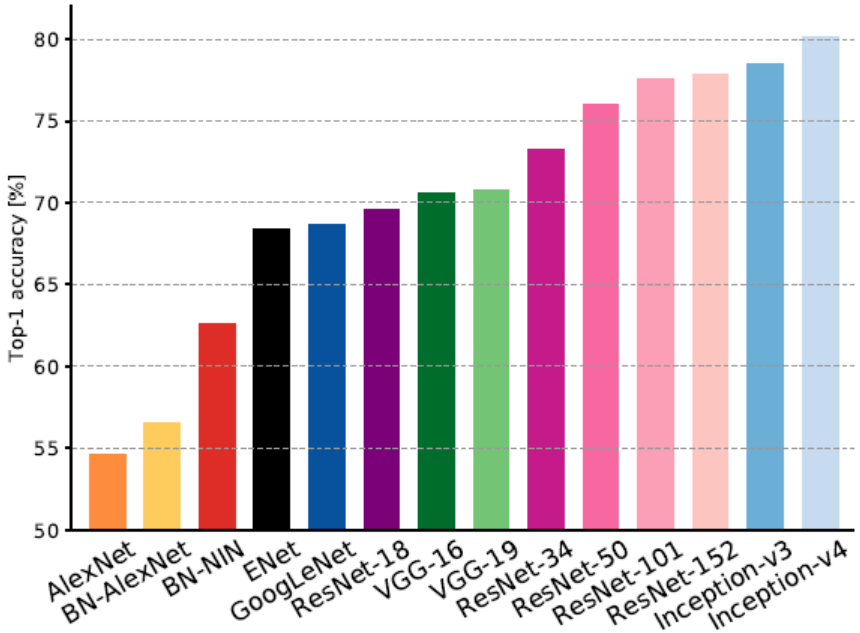
- Adapt to a wide variety of data
  - Adapted to new problems relatively easily
- Require less statistical training
  - Automatically fine-tune the learning procedure
- Learn with simple algorithms
  - But with ability to produce complex model
- Scale to large data sets
  - More data leads to more accurate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

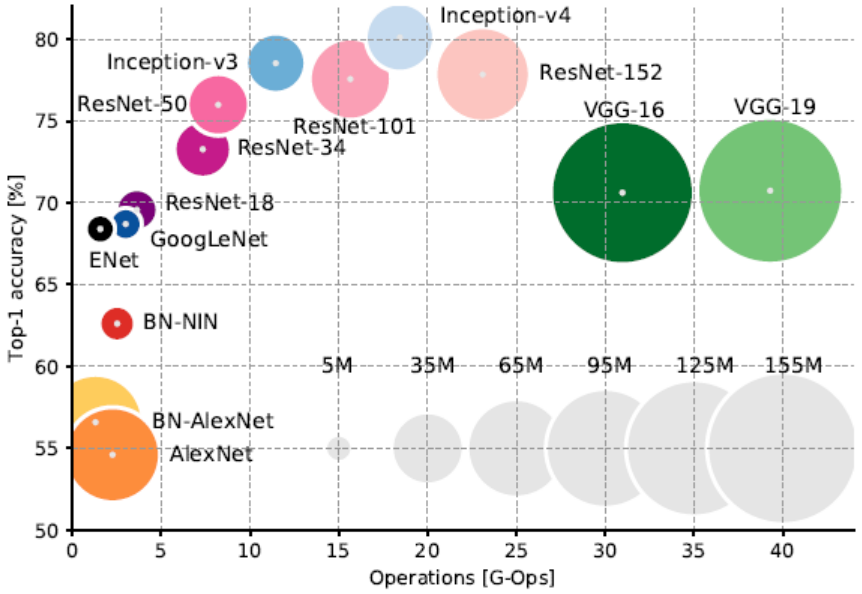




Growing Demand for Computing

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

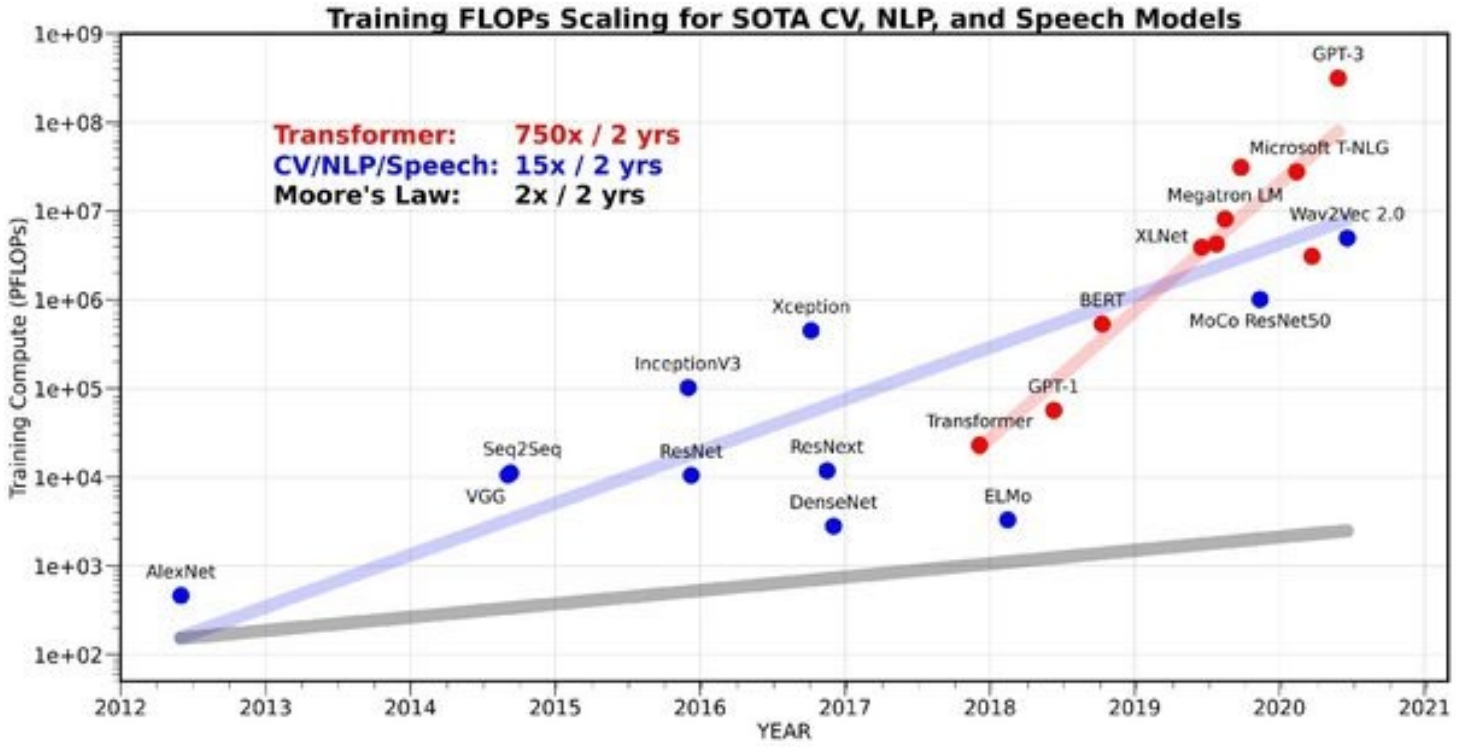




Source: Alfredo Canziani, "AN ANALYSIS OF DEEP NEURAL NETWORK MODELS FOR PRACTICAL APPLICATIONS".

Growing Demand for Computing

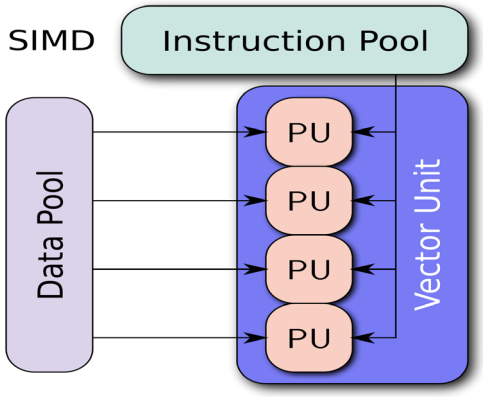
- 3.5 month doubling time since. (18 month double time for Moore's Law)
- 30K growth in 6 years

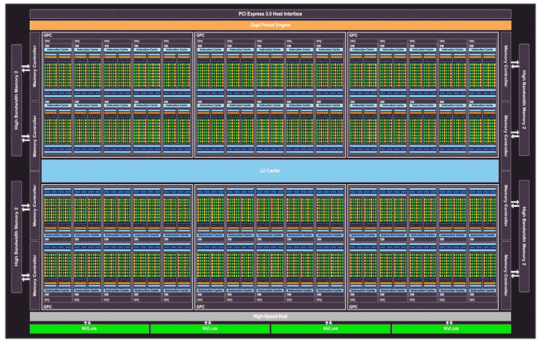


Source: openAI [https://openai.com/blog/ai-and-compute/]

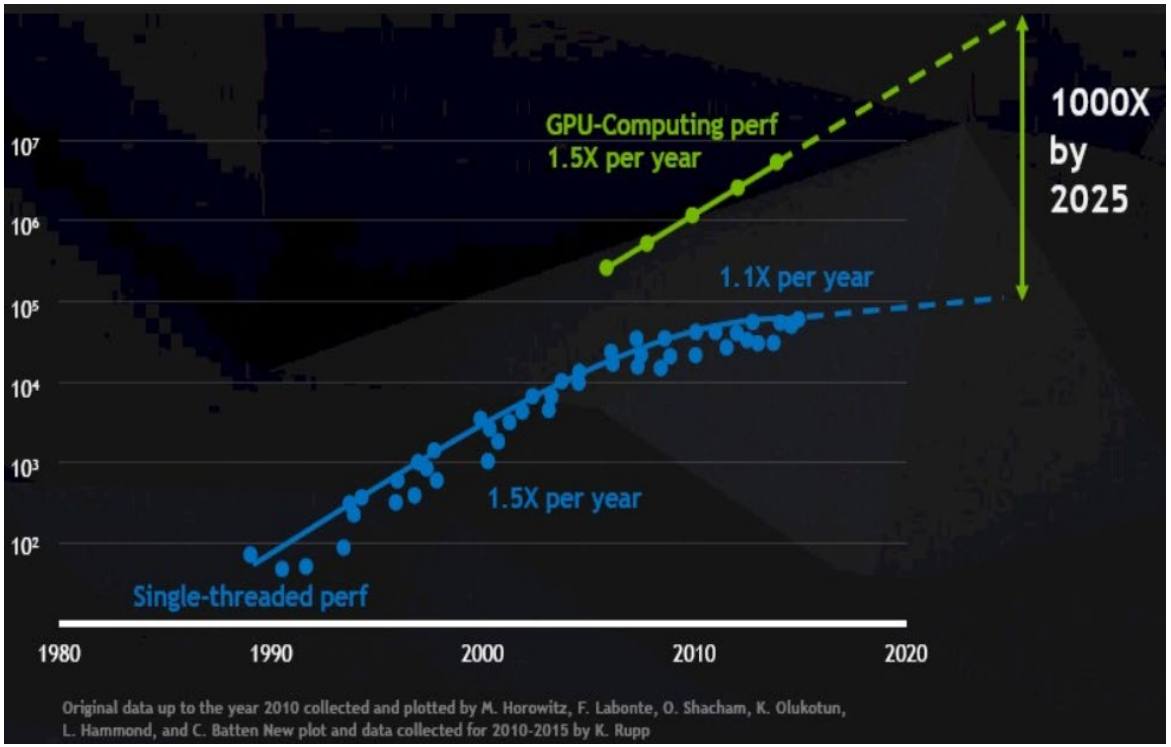
Many-Core Processor: GPU

- Accelerator based on SIMD processor architecture





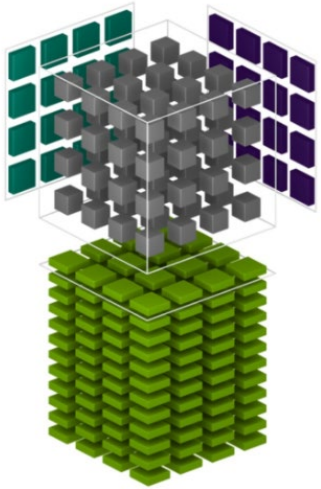
5,120 cores in a GPU



Images from Nvidia

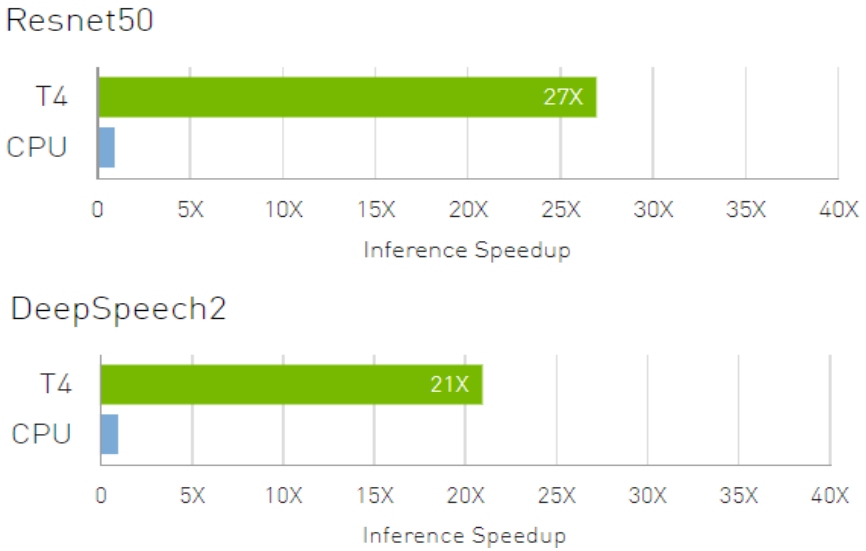
TensorCore

- 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

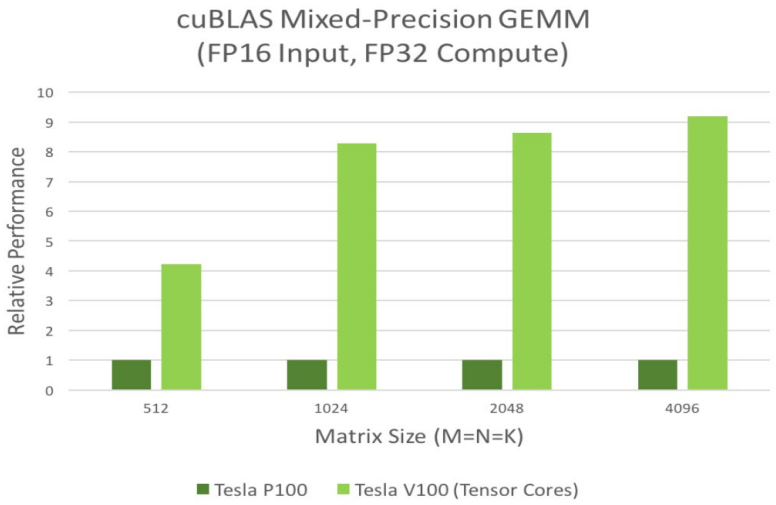


TensorCore

- 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

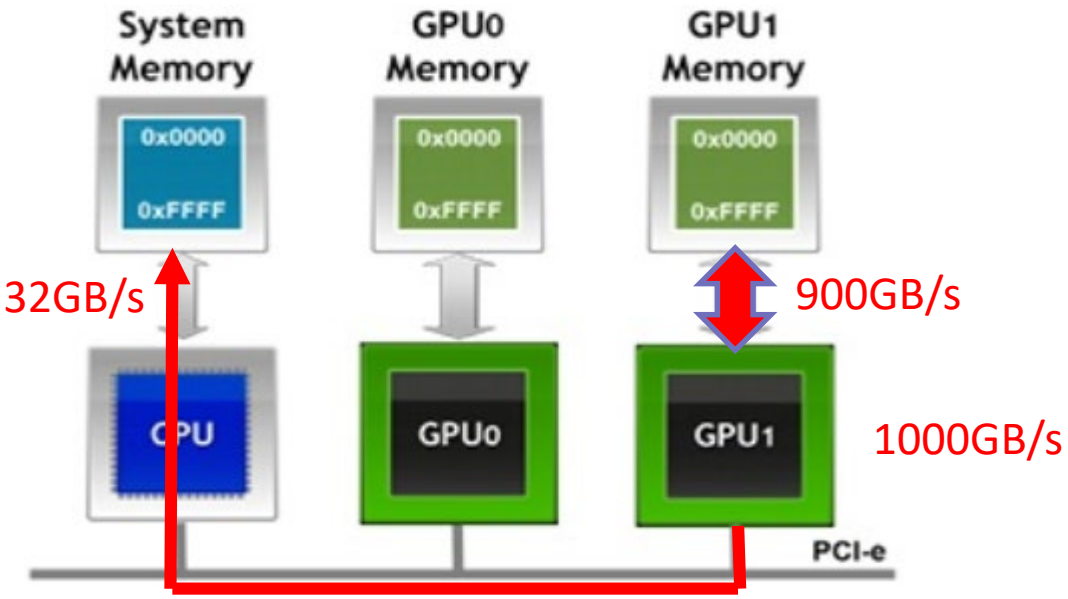


Chip-to-chip GPU-to-CPU speedups | NVIDIA Tesla T4 GPU vs Xeon Gold 6140 CPU



Input matrices are half precision, computation is single precision.

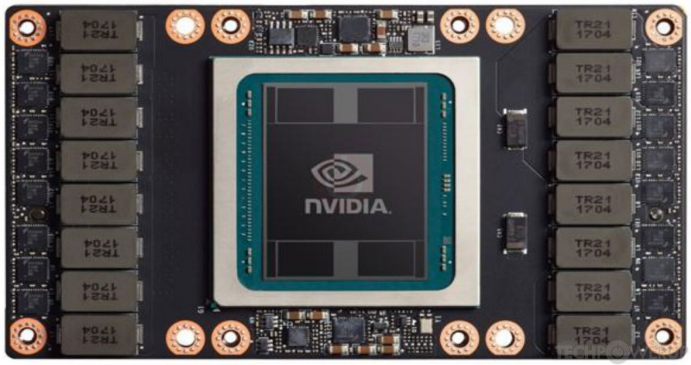
GPU: Memory Access Bottleneck



- 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

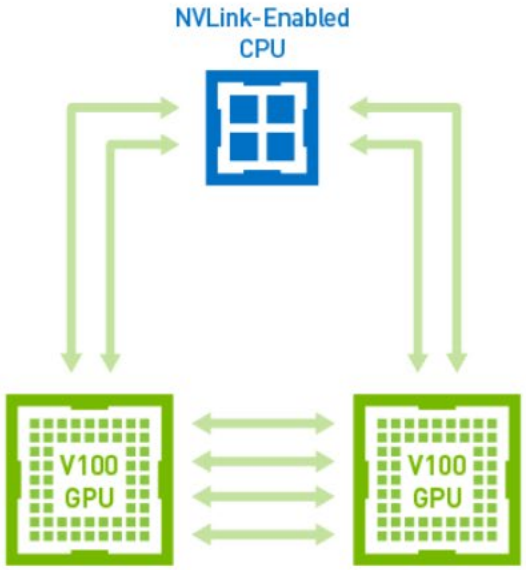
NV-Link Fabric

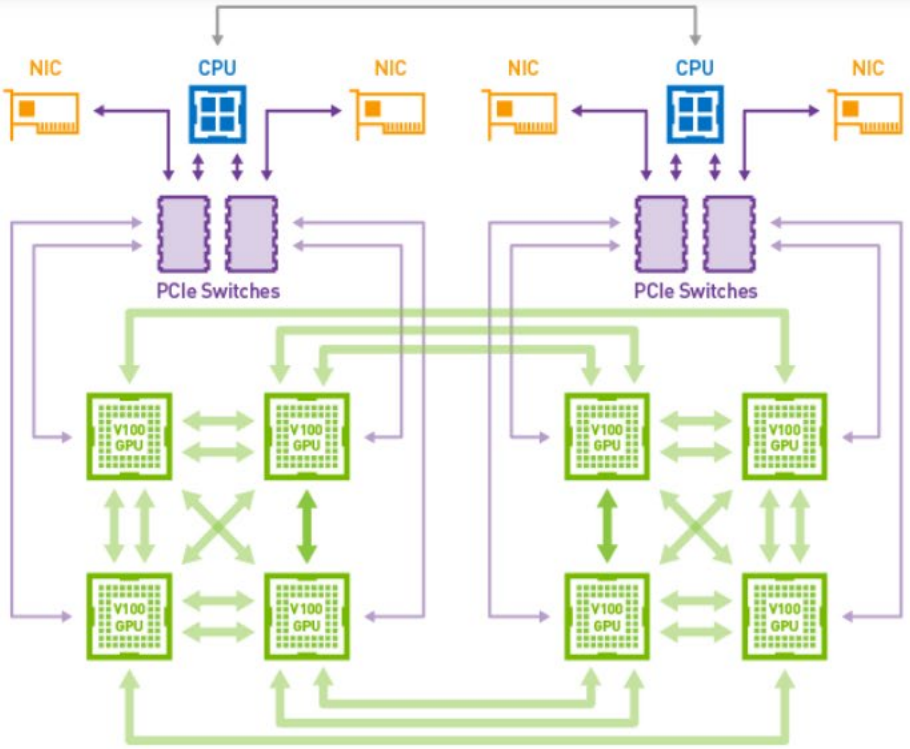
- A high-bandwidth, energy-efficient **interconnect** that enables ultra-fast communication between the CPU and GPU, and between GPUs
- Achieve 300GB/s data sharing rates
  - **5 to 12 times faster** than the traditional **PCIe Gen3** interconnect:
- Must use the SXM GPU module



NV-Link Fabric

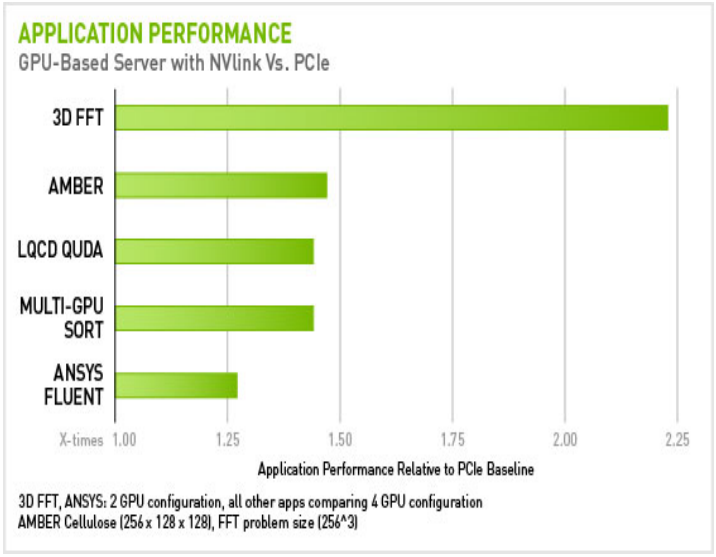
- Only NVLink-Enabled CPU can use NVLink to transfer data from host mem. to device mem.

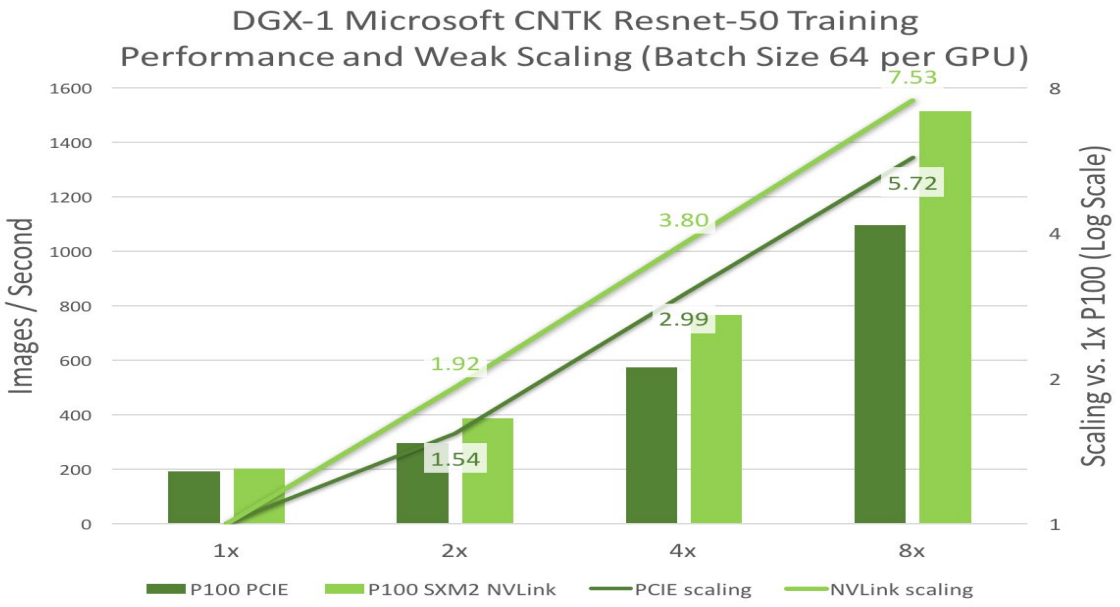




NV-Link

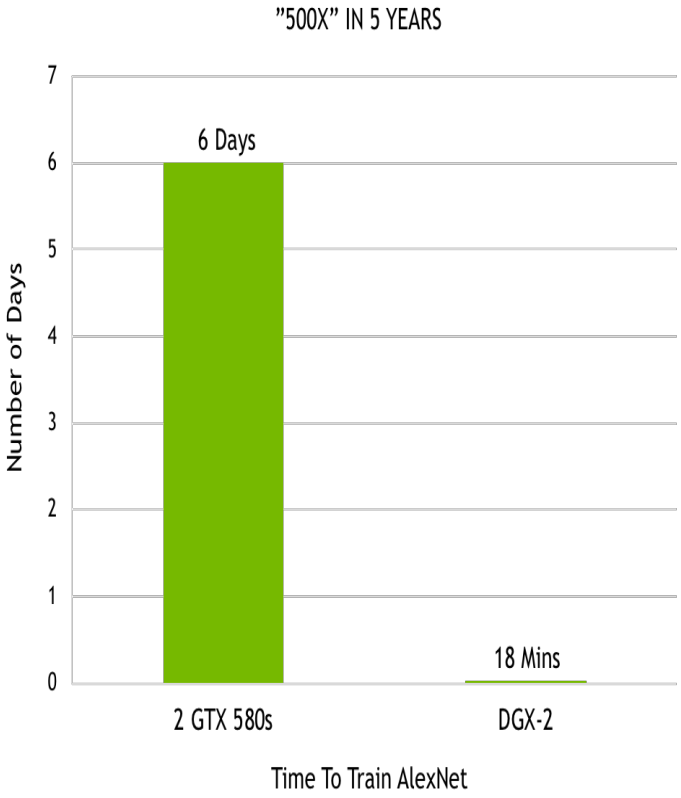
- Higher network performance deliver higher overall application performance, and better scalability (less communication overhead)

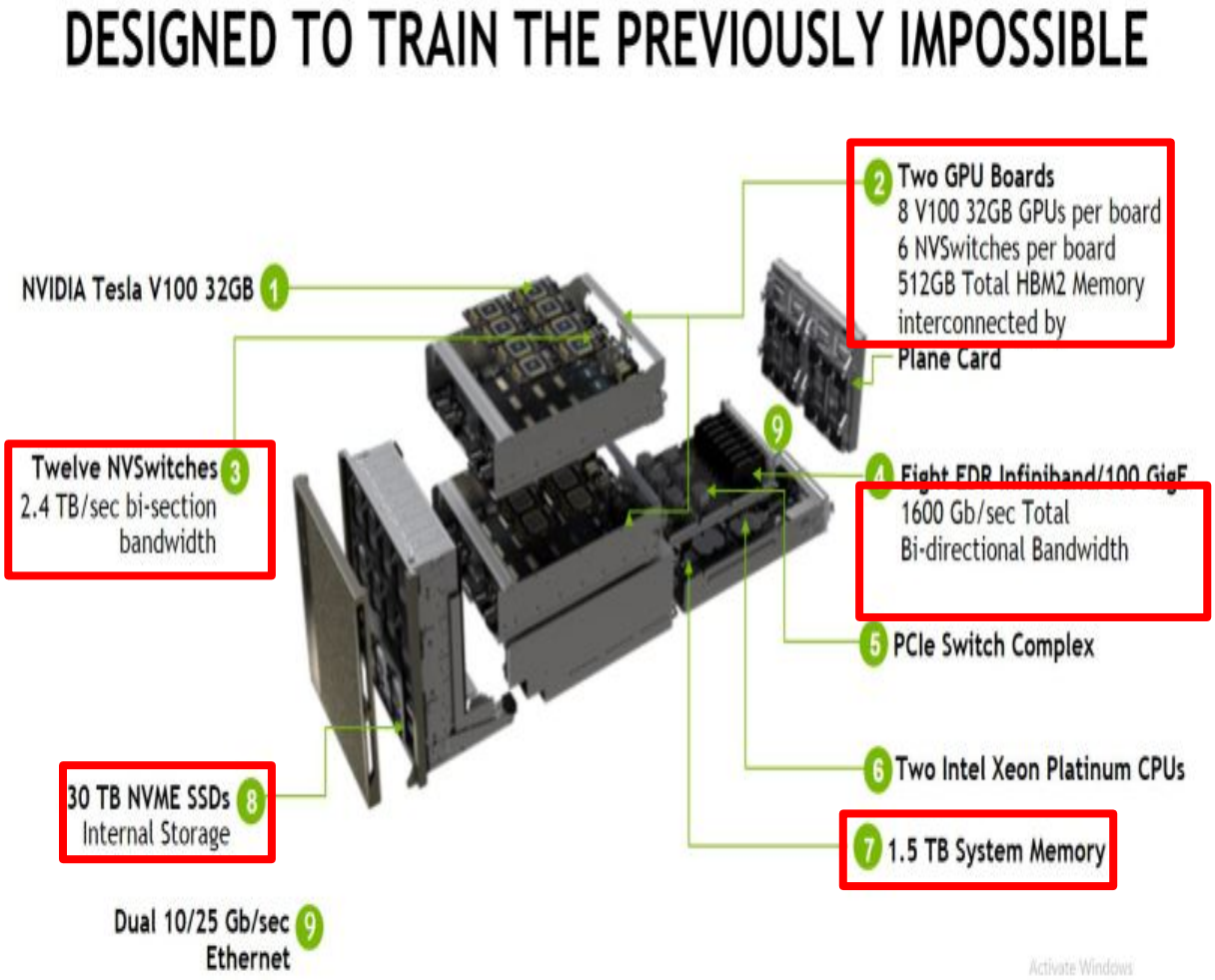




Images from Nvidia

DGX-2 GPU Server

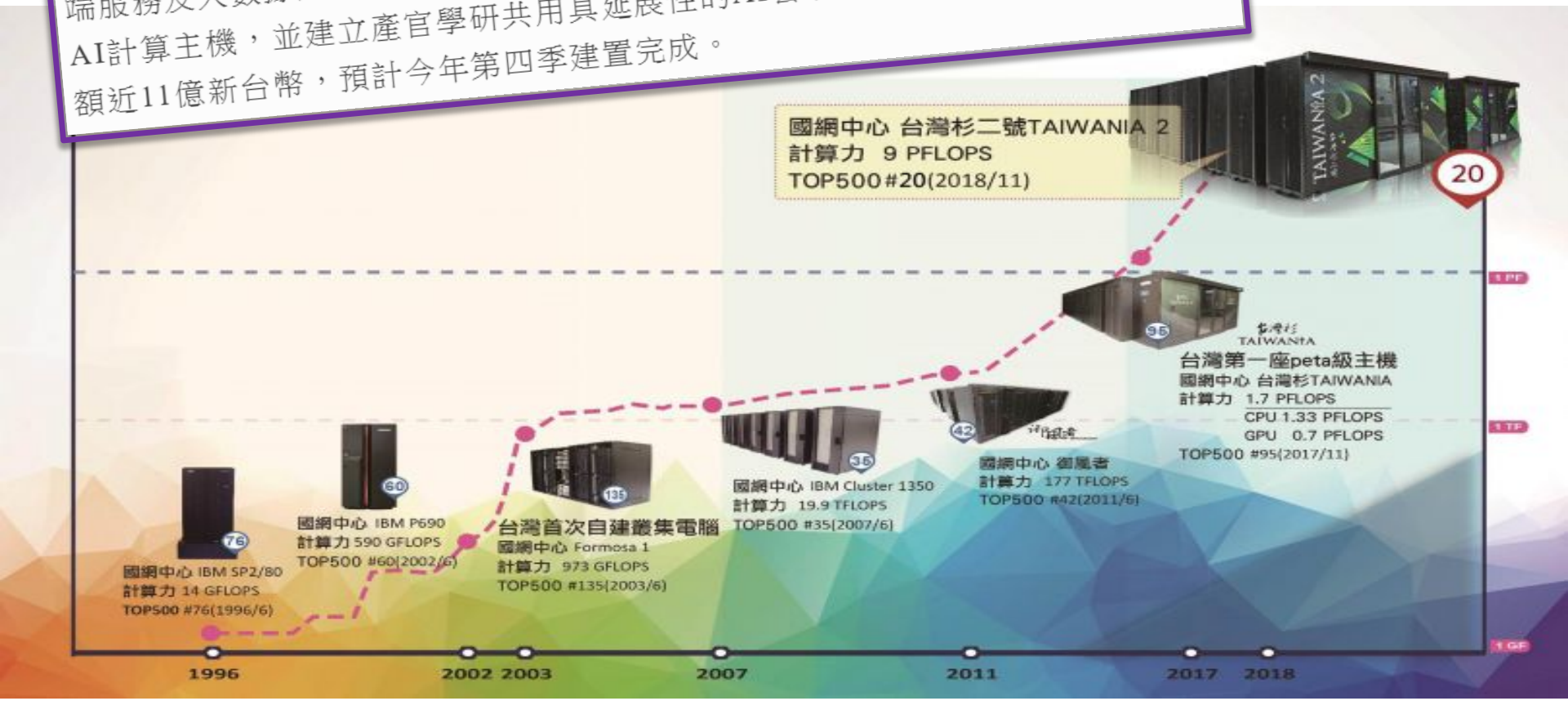




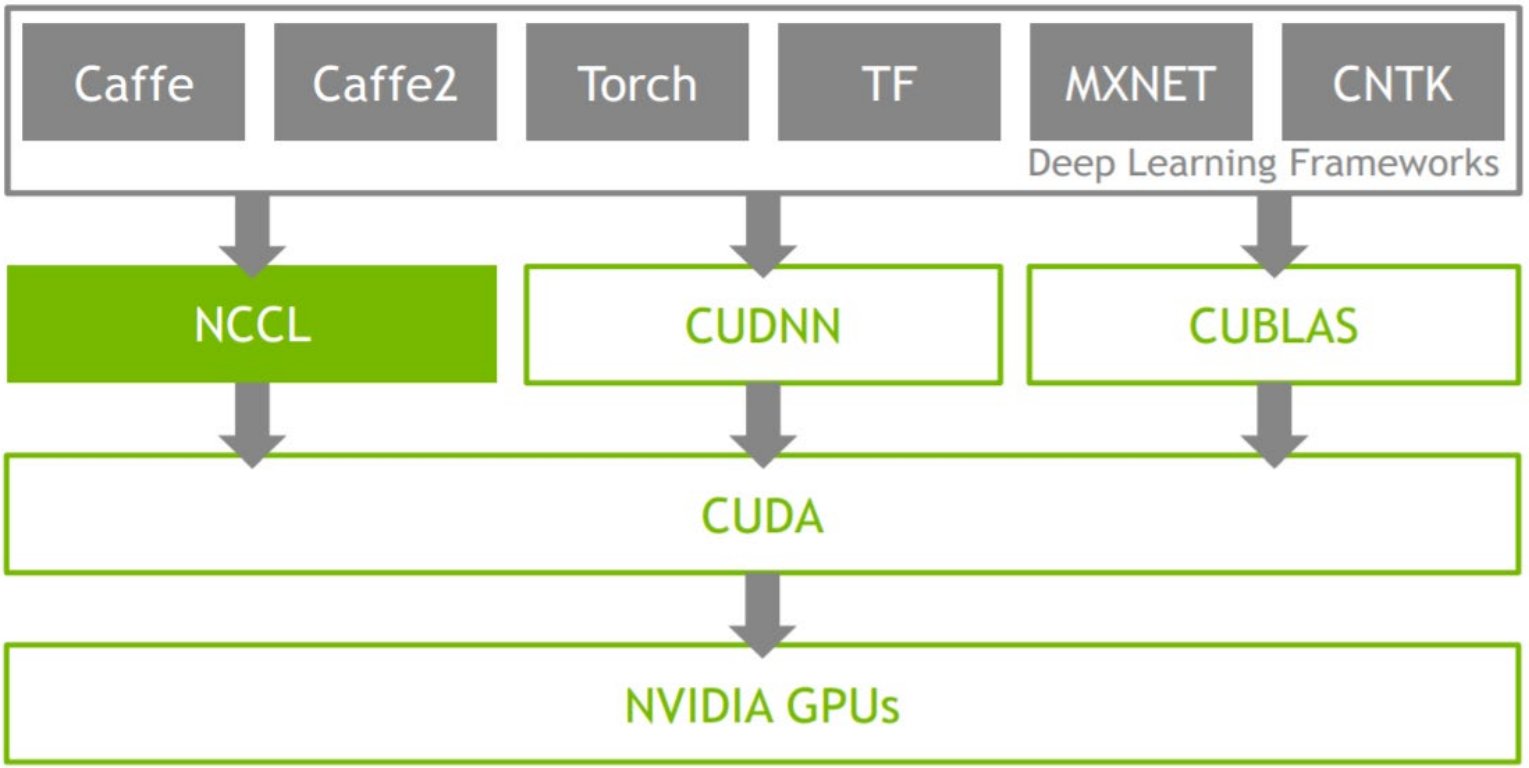
Images from Nvidia

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。

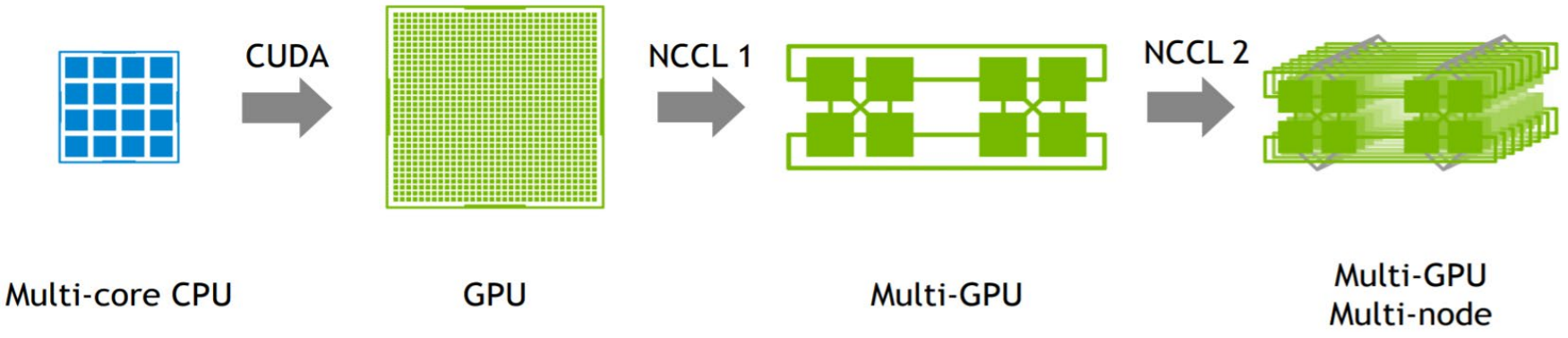


CUDA Libraries for Deep Learning



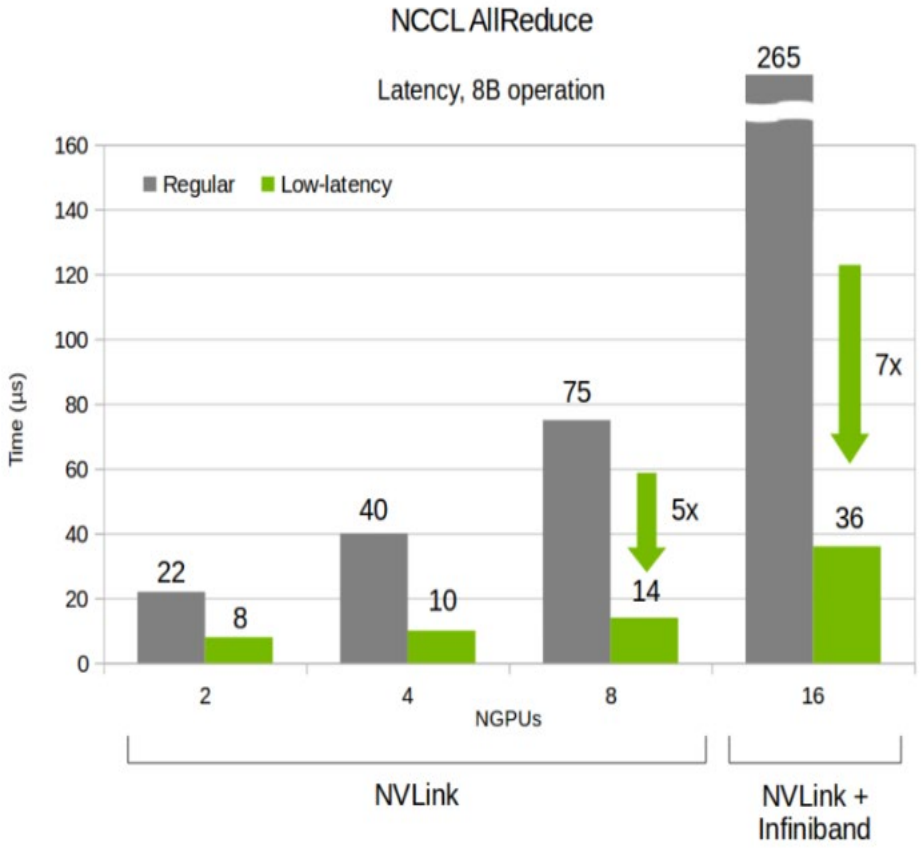
NCCL

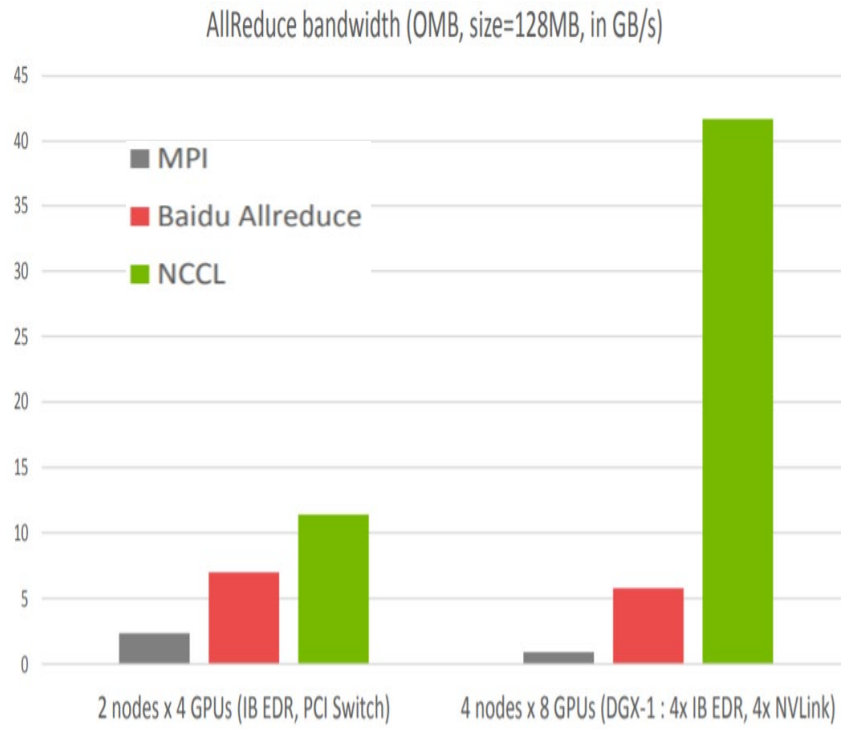
- NVIDIA Collective Communications Library
  - Optimized implementation of inter-GPU communication operations, such as **allreduce**
  - Deep learning frameworks can rely on NCCL's highly optimized, MPI compatible and topology aware routines, to take full advantage of all available GPUs within and across 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

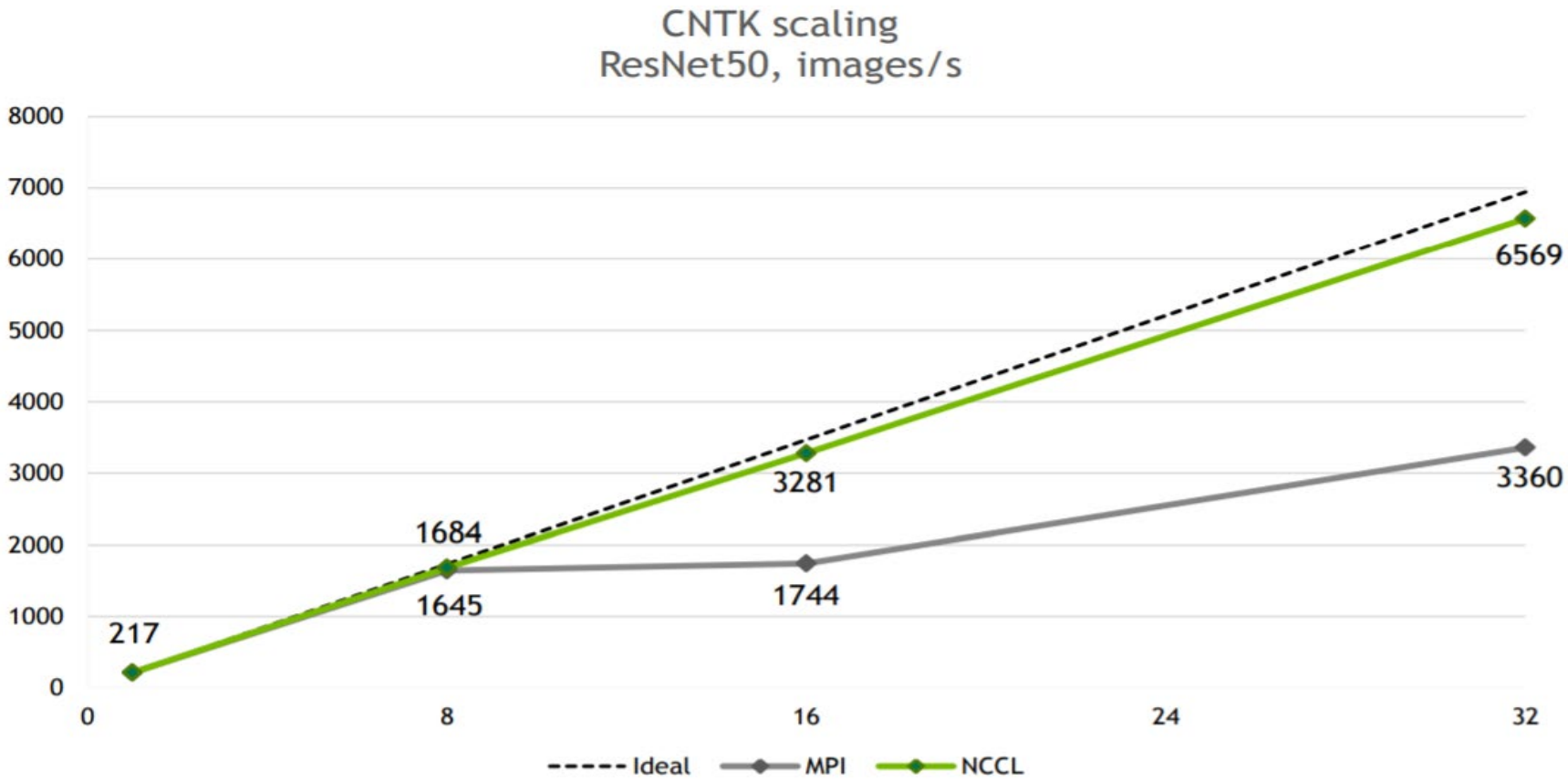
- Performance improvement on BW and Latency





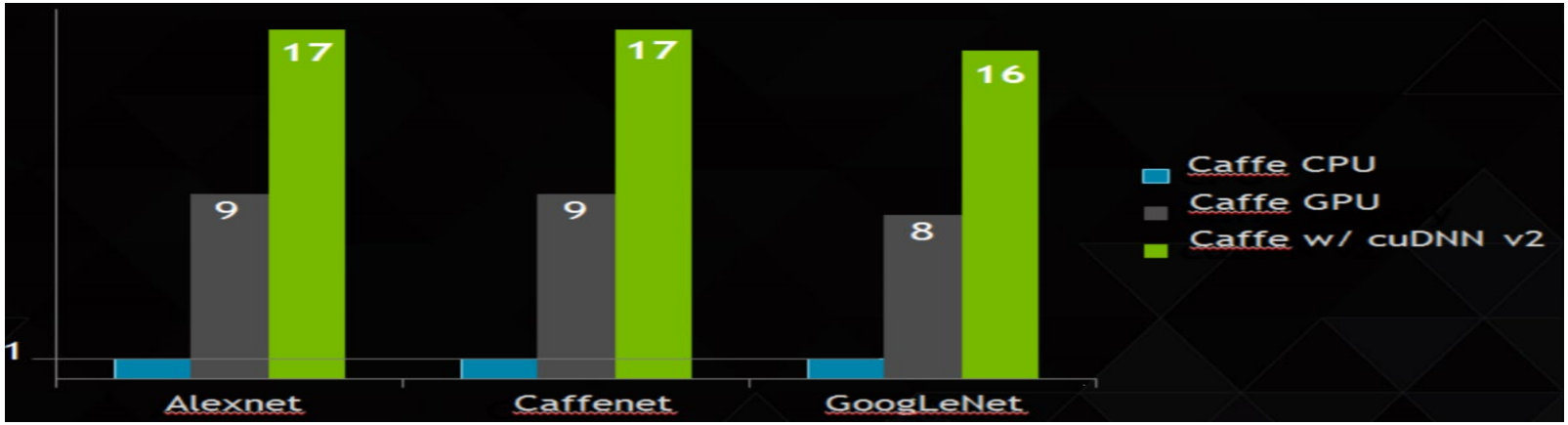
NCCL

- 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



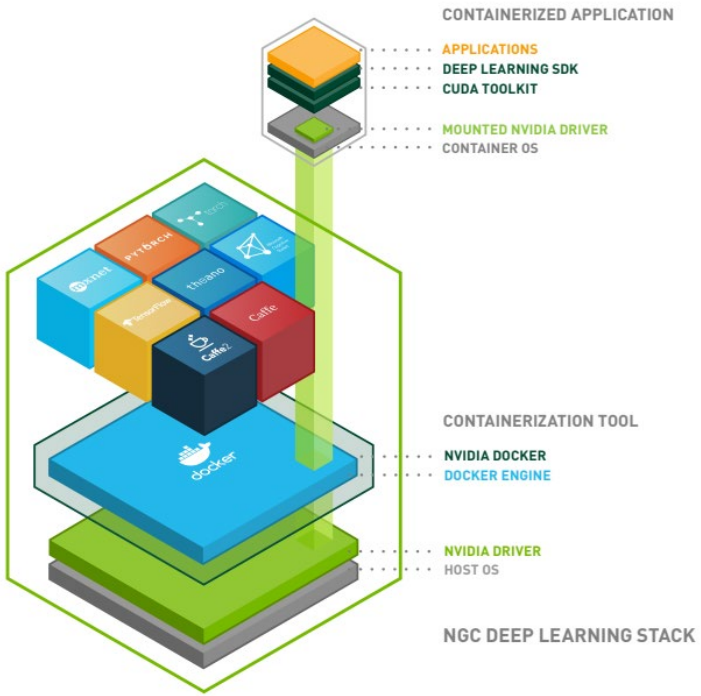
cuBLAS

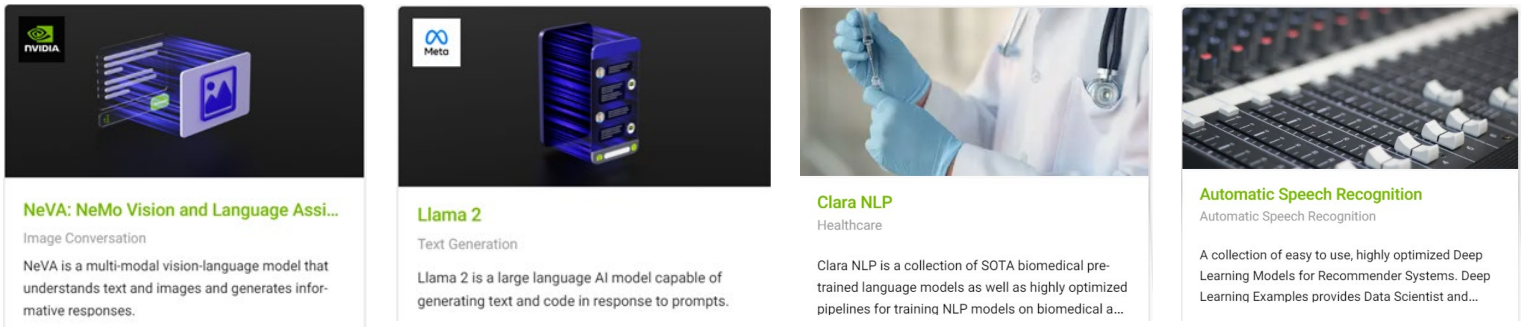
- BLAS: Basic Linear Algebra Subprograms
  - Defines a set of common functions for scalars, vectors, and matrices
    - E.g., cublaslasmax(): finds the smallest(first) index in a vector that is a maximum for that vector
  - Good for anything that uses heavy linear algebra computations
    - E.g., graphics,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

Docker Container: NGC

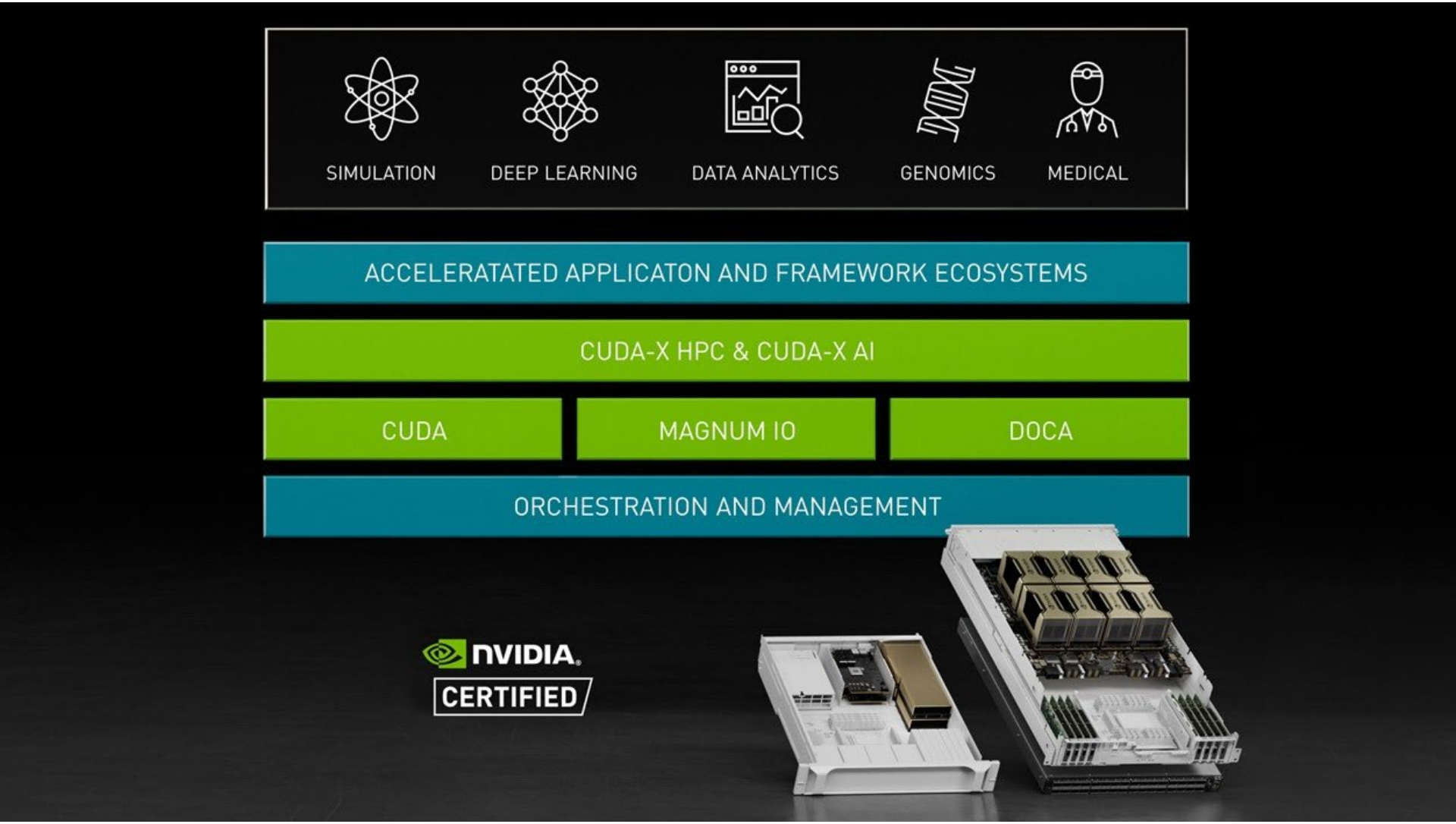
- The hub for GPU-optimized software for deep learning, machine learning, and HPC that provides containers, models, model scripts, and industry solutions
- Allow data scientists, developers and researchers can focus on building solutions and gathering insights faster.





Nvidia SDKs

Listen to what they said

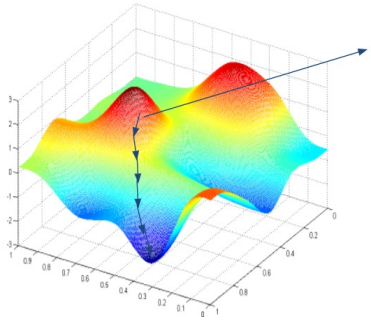


Outline

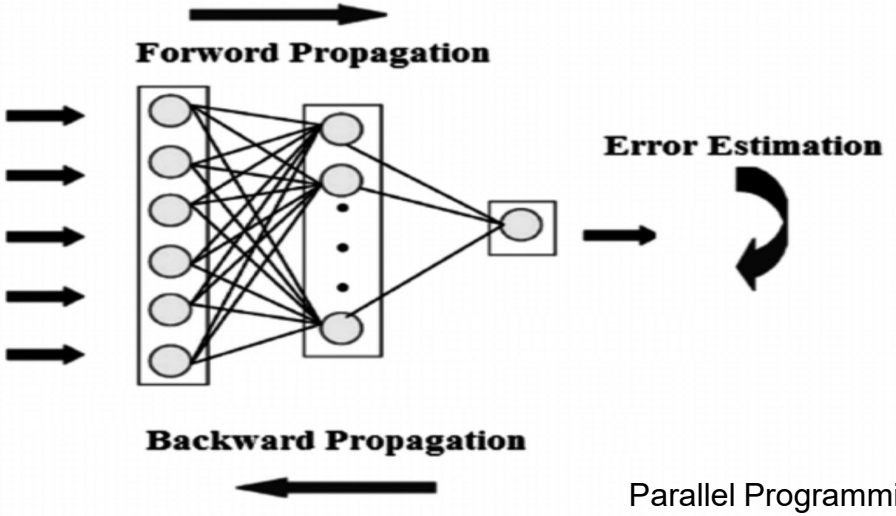
- Brief Introduction of Deep Learning
  - Computing Demand for Training
  - GPU Solutions
- Distributed Deep Learning Computations
  - Parallel strategies
  - Optimization strategies
  - Memory Optimization
  - TensorFlow & Horovod
- Trend & Future of Deep Learning Computing
  - ML Systems & MLOps
  - AI Cloud Services & AI Chips
  - Federated Learning
  - Remarks

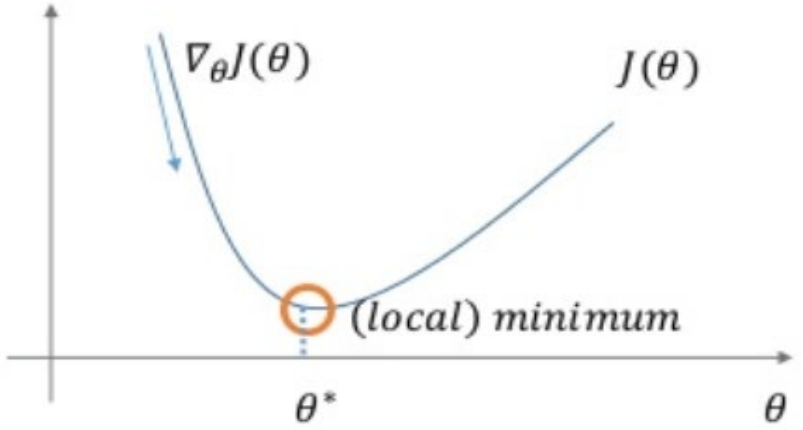
Gradient Descent Algorithm

- 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



$$\theta = \theta - \alpha * \nabla_\theta J(\theta)$$

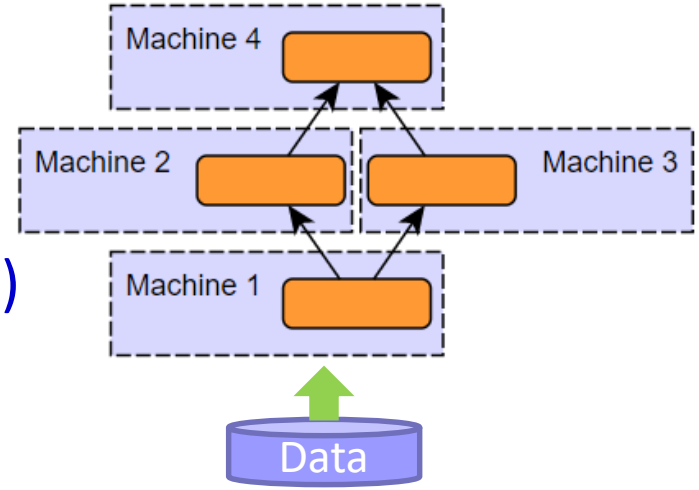




How to Utilize Multiple Machines?

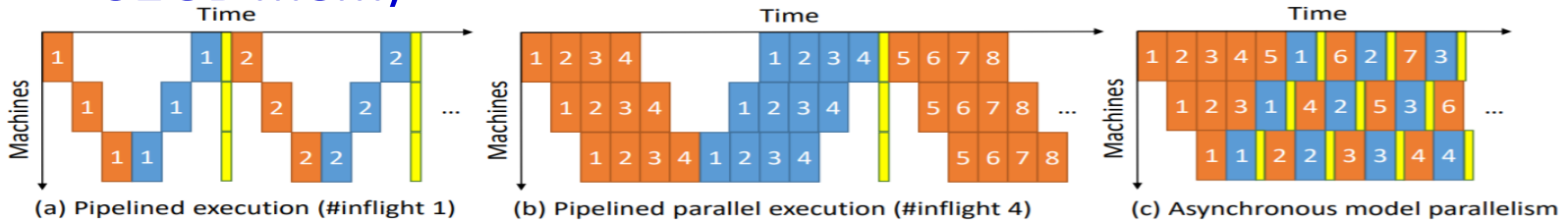
- We could utilize resources by…
  - Running multiple training jobs for different models
  - Running multiple training jobs with the same model, but different hyper-parameters
  - Running a single model training job across multiple machines ➔ **distributed training**
    - Fully utilize the resources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

Model Parallelism



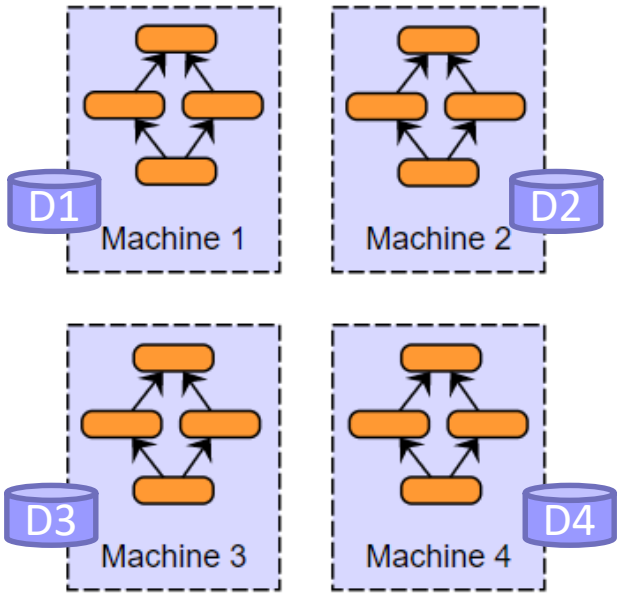
- Parallelization
  - Model is split across machines(GPUs)
  - The whole dataset is replicated
- Weakness
  - Harder to achieve good scalability

  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)



Data Parallelism

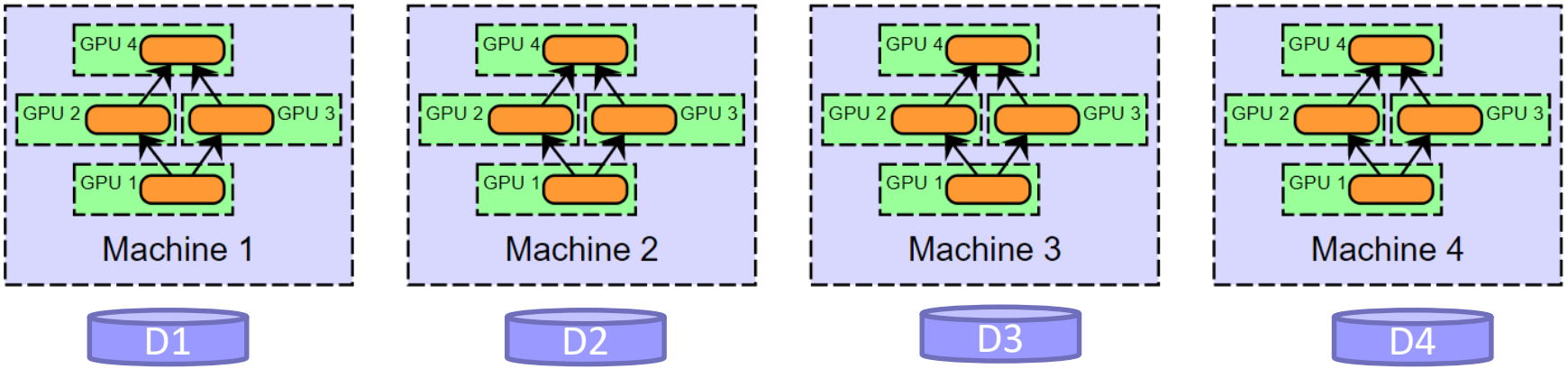
- Parallelization
  - Each machine (GPU) independently evaluate the whole model on a part of the dataset to compute gradient
  - Weight is updated by the average of gradients from all nodes
- Strength
  - Easier to achieve linear scale
  - Preferred approach for distributed systems
- Weakness
  - The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

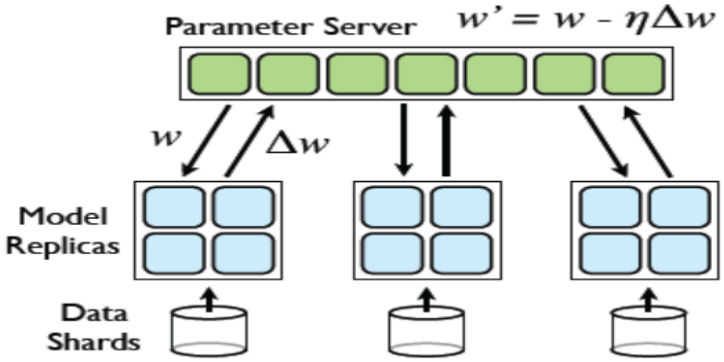
Data + Model Parallelism

- 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

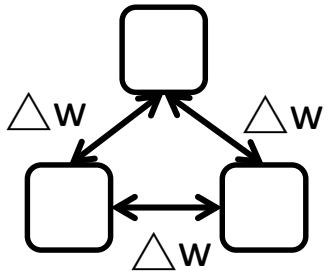


Parameter Server vs. Allreduce

- 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 {

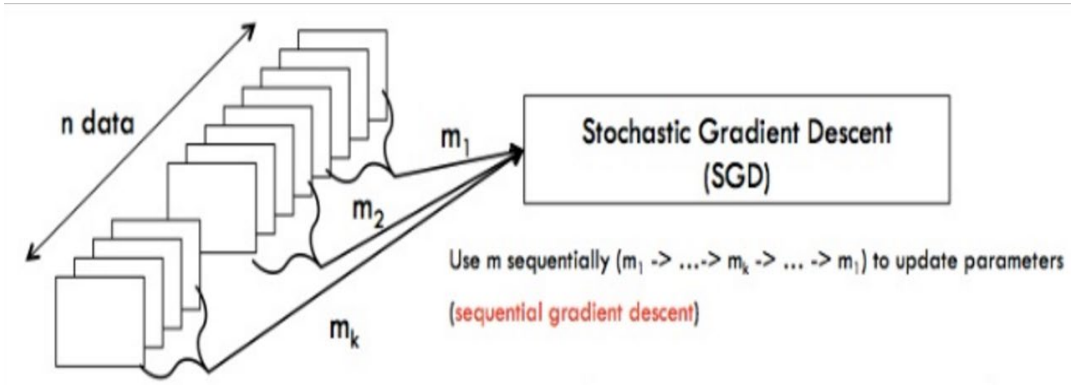
  for $i = 1, 11, 21, 31, \ldots, 991$ {

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

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

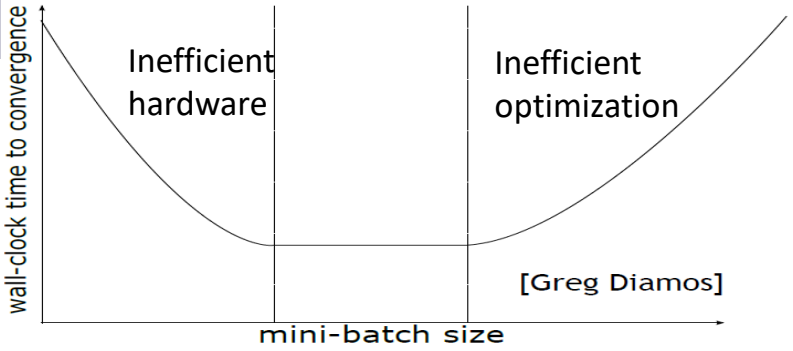
  }

}



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

1. Mini Batch SGD



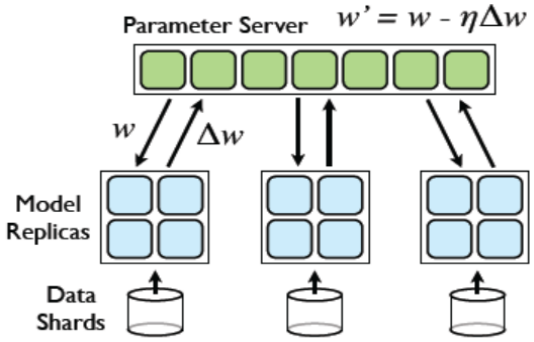
- Advantages:
  - Vectorization: make data parallelism arbitrarily efficient by increasing the batch size (In particular for GPU)
  - Lower communication cost: fewer number of iterations comparing to SGD
  - Smoother update: the variance of the update is reduced

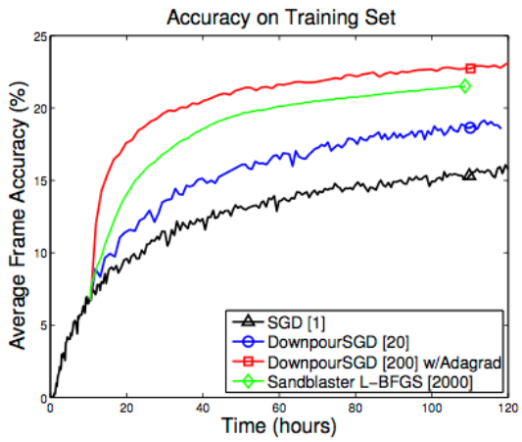
- Risks
  - Very big batch sizes adversely affect the SGD converges rate as well as the quality of the final solution
  - Noise actually can be useful as it may help escape local minima

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

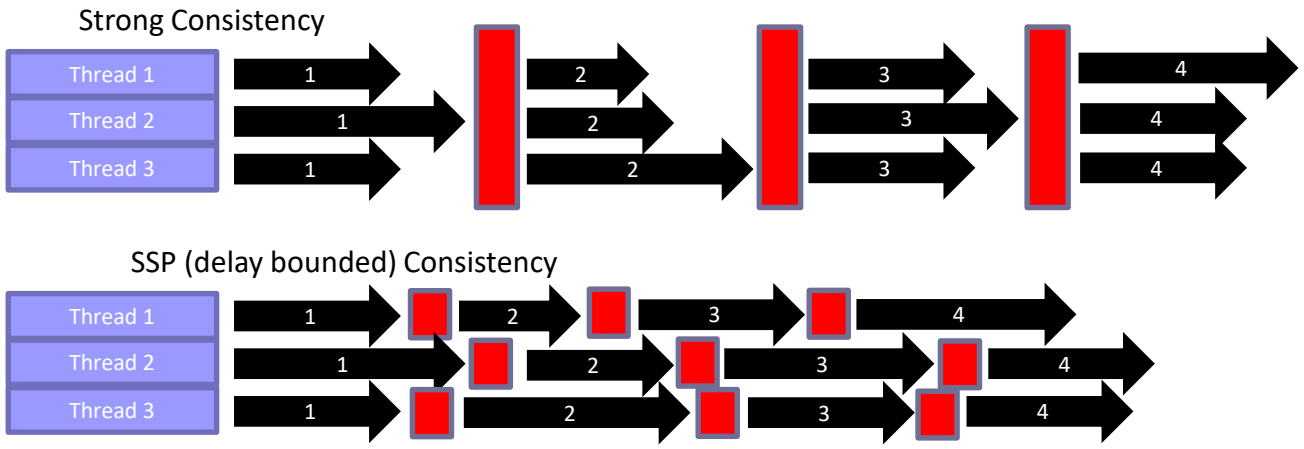




*Google(DistBlief), "*Large Scale Distributed Deep Networks*", In* Neural Information Processing Systems, 2012

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

$$G_{ij\ell}^{\text{quant}}(t) = \mathcal{Q}(G_{ij\ell}(t) + \Delta_{ij\ell}(t-N))$$

$$\Delta_{ij\ell}(t) = G_{ij\ell}(t) - \mathcal{Q}^{-1}(G_{ij\ell}^{\text{quant}}(t))$$

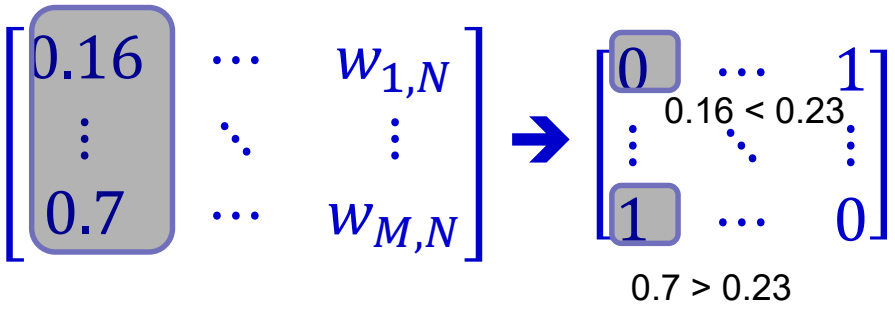
error

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

- 

Effect: communication reduced by 32x

Column avg. ➔ 0.23

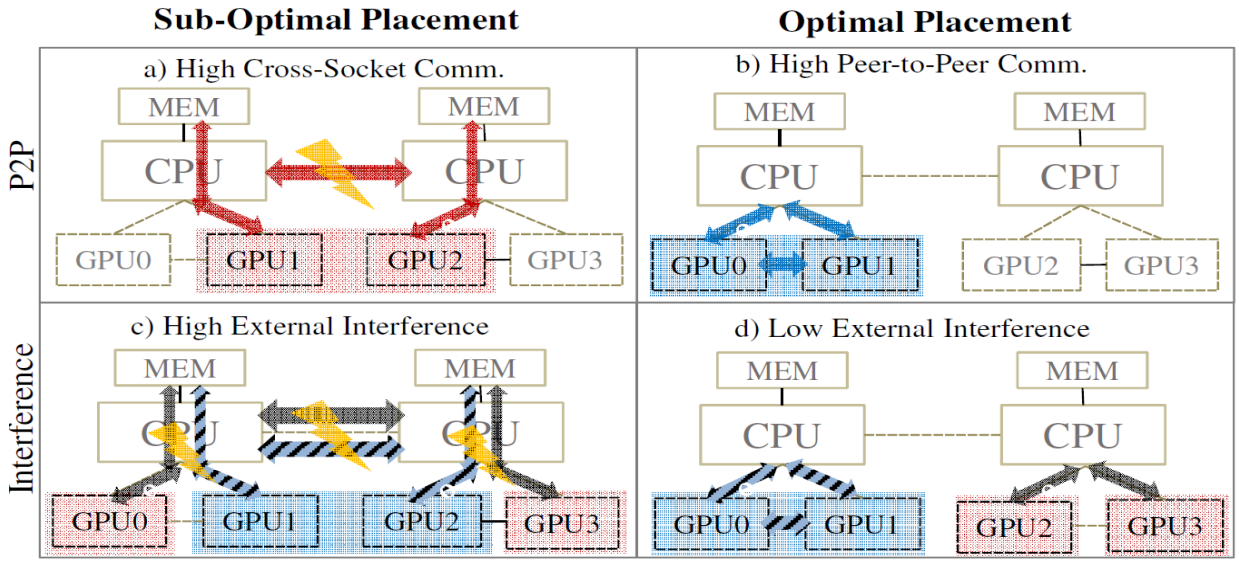
Avg. is sent over network

- 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

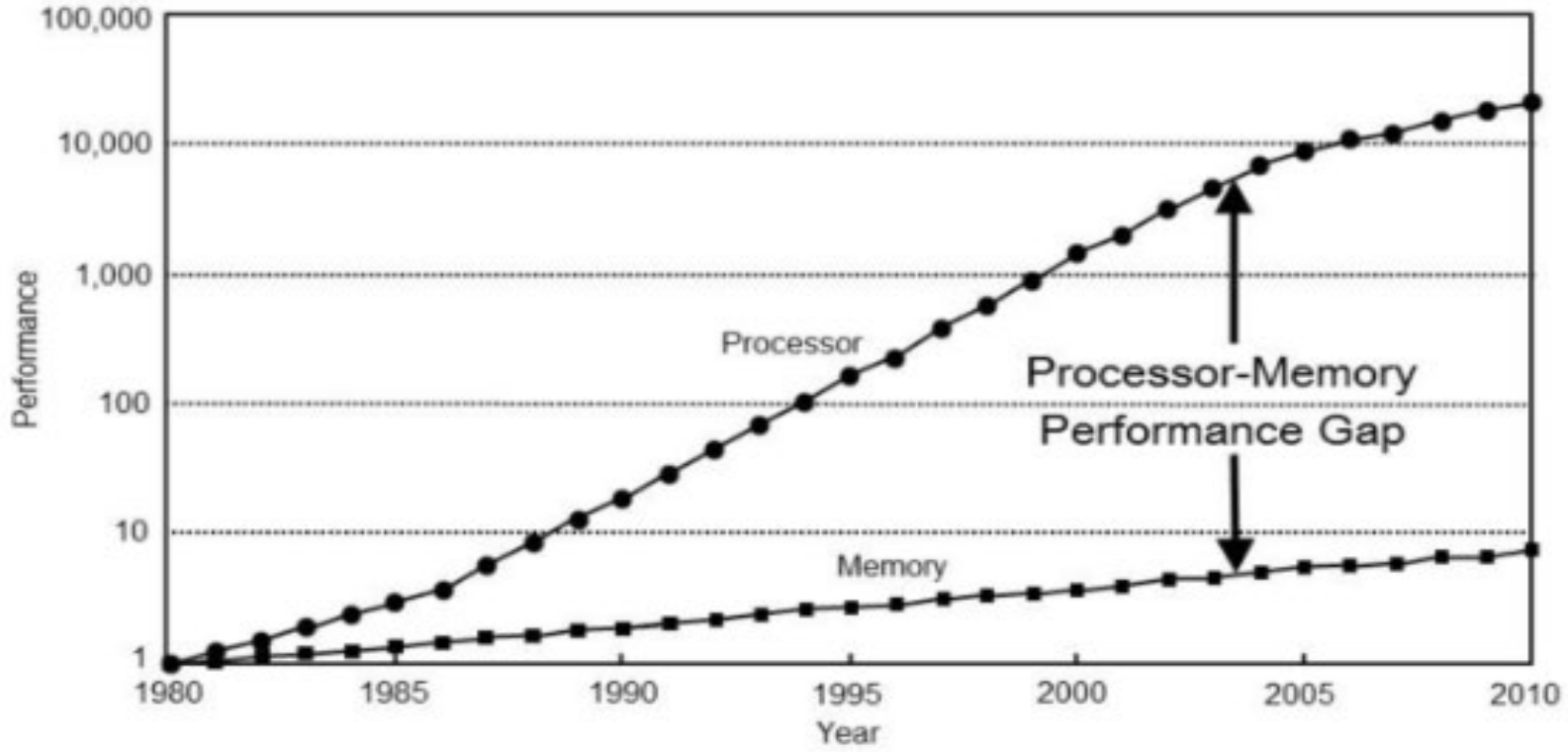
Most systems choose Pack strategy to minimize communication overhead

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

Memory Wall Problem

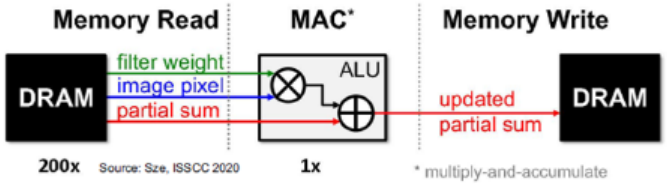


Computer Architecture: A Quantitative Approach by John L. Hennessy, David A. Patterson, Andrea C. Arpaci-Dusseau

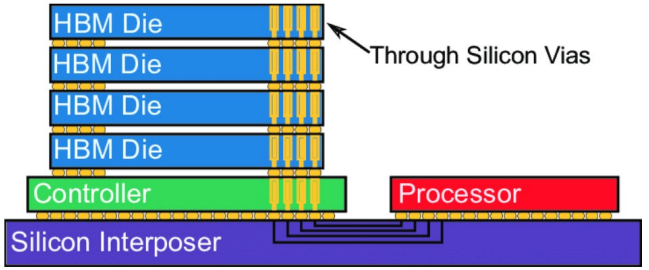
Memory Wall Problem

- Cache
  - Reduce memory access
- High-Bandwidth Memory (HBM)
  - A high-speed computer memory interface

  for 3D-stacked synchronous dynamic random-access memory (SDRAM)
  - Only available on high-performance computing servers due to cost
- Compute-in-memory
  - Memory chip with compute capability
  - for low-power neural network inference



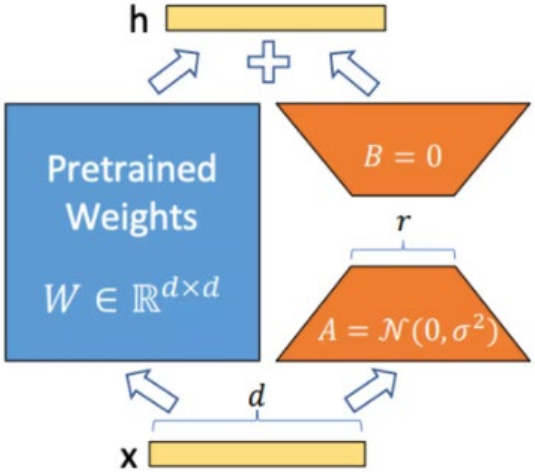
Source: In-Memory Computing for Low-Power Neural Network Inference by Tom Dillinger



Source: Hardware and Software Optimizations for Accelerating Deep Neural Networks: Survey of Current Trends, Challenges, and the Road Ahead

Memory Optimizations: LoRA

- LoRA[1]: A training method that accelerates the training of large models while consuming less memory
  - It adds pairs of rank-decomposition weight matrices (called update matrices) to existing weights, and only trains those newly added weights
  - Advantages:
    - Previous pretrained weights are kept frozen so the model is not as prone to catastrophic forgetting.
    - Rank-decomposition matrices have significantly fewer parameters than the original model

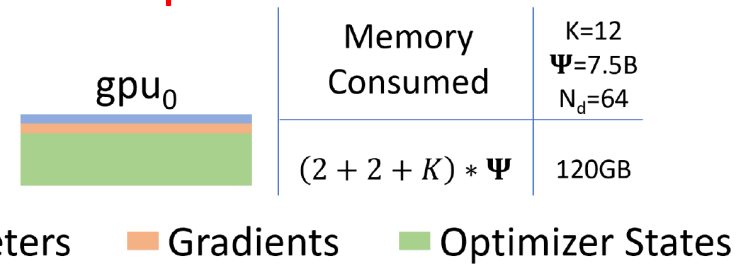


Train GPT-2 (original: 345M parameters) with <11M parameters.

[1]: LoRA: Low-Rank Adaptation of Large Language Models

Memory Optimizations: Zero

- For large models, the majority of the memory is occupied by model states which include the optimizer states (such as momentum and variance in Adam), gradients, and model parameters
- ZeRO-DP aims to efficiently combine the idea of data parallel and model parallel, achieves the computation/communication efficiency of DP while achieving memory efficiency of MP
- ZeRO-DP optimize the memory consumption of model state by partition states on to devices



[1] ZeRO: Memory Optimizations Toward Training Trillion Parameter Models

Memory Optimizations: Zero

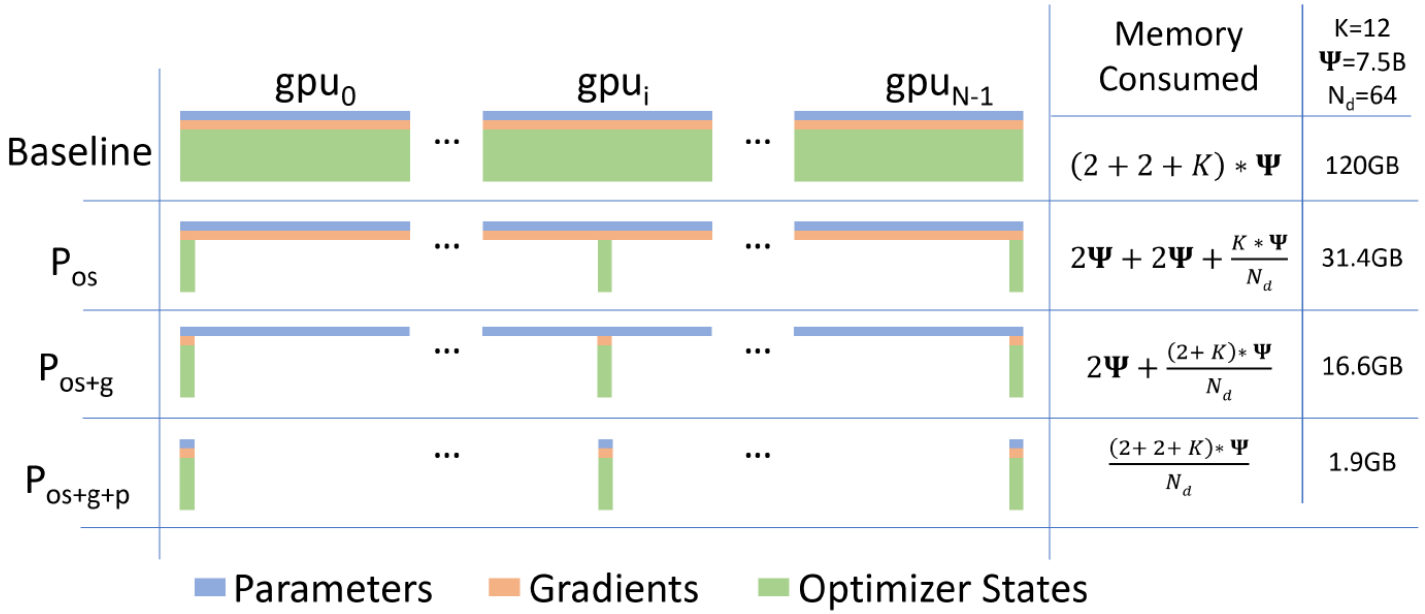
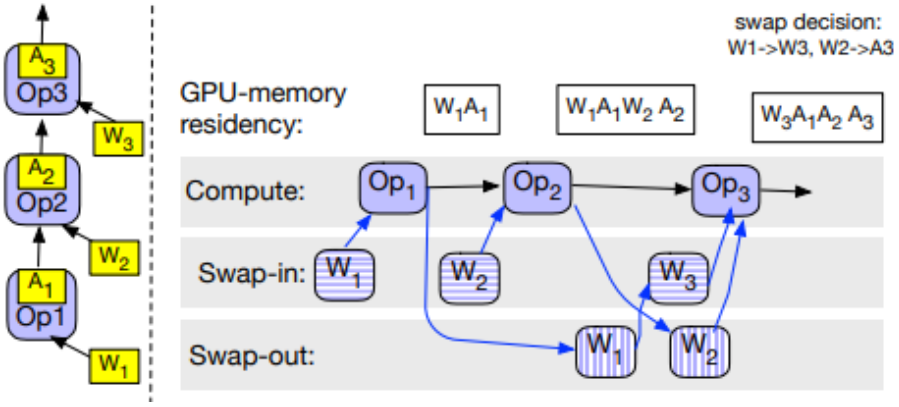


Figure 1: Comparing the per-device memory consumption of model states, with three stages of *ZeRO*-DP optimizations. Ψ denotes model size (number of parameters), K denotes the memory multiplier of optimizer states, and N_d denotes DP degree. In the example, we assume a model size of $\Psi = 7.5B$ and DP of $N_d = 64$ with $K = 12$ based on mixed-precision training with Adam optimizer.

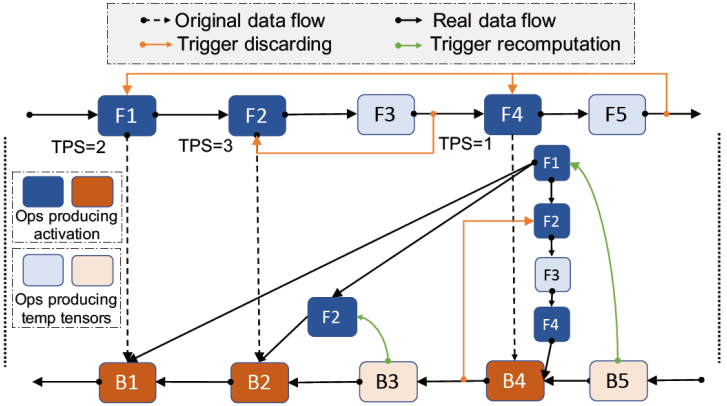
Memory Optimizations

- Swapping
  - Use host memory as the swap space



SwapAdvisor: Pushing Deep Learning Beyond the GPU Memory Limit via Smart Swapping

- Re-computation
  - The values (e.g. activations) computed in forward propagation are used in backward



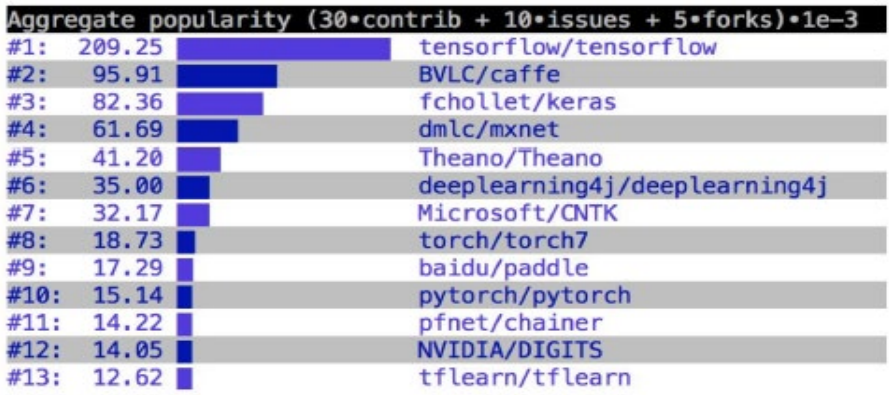
Melon: Breaking the Memory Wall for Resource-Efficient On-Device Machine Learning

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

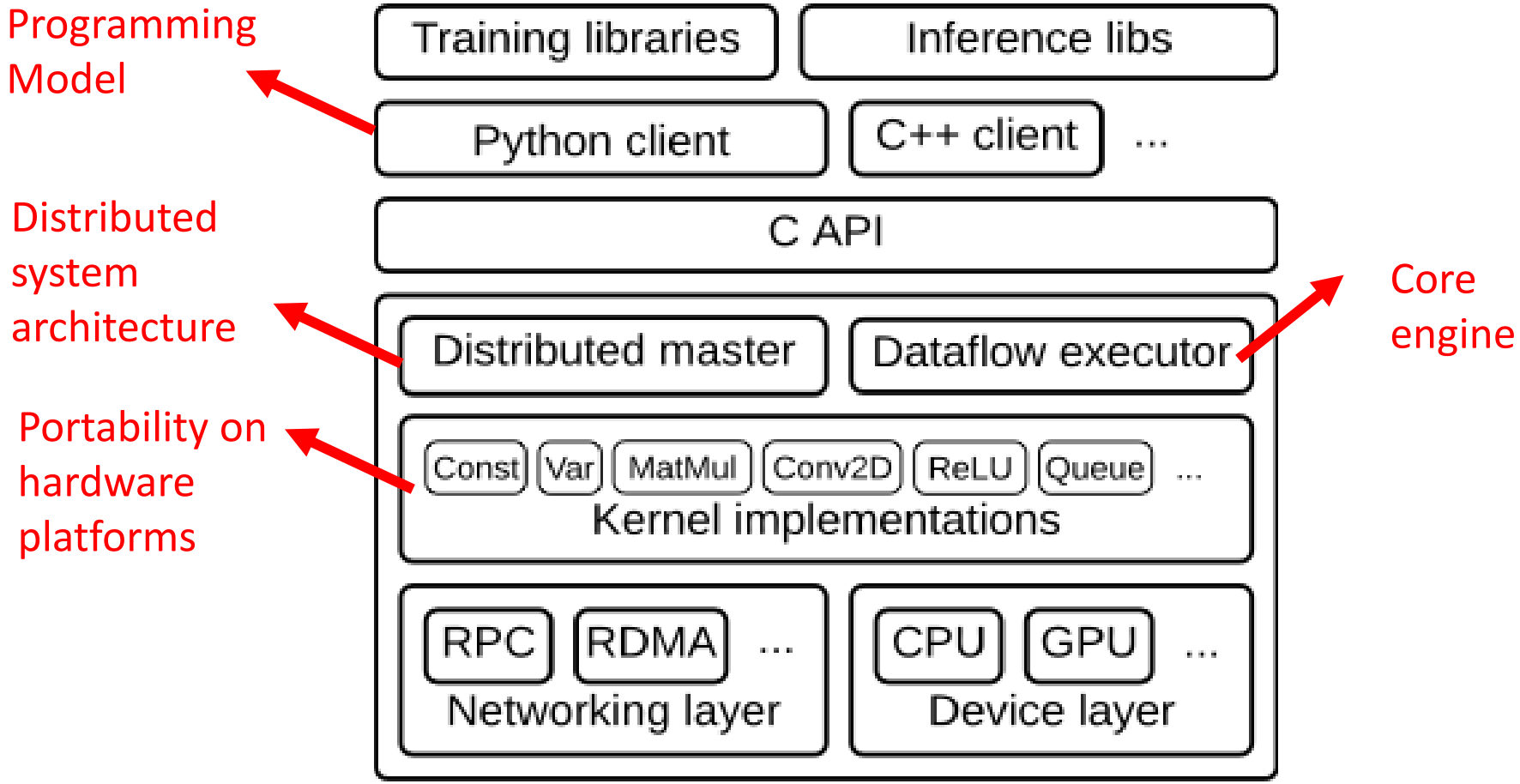


François Chollet, Google Developer

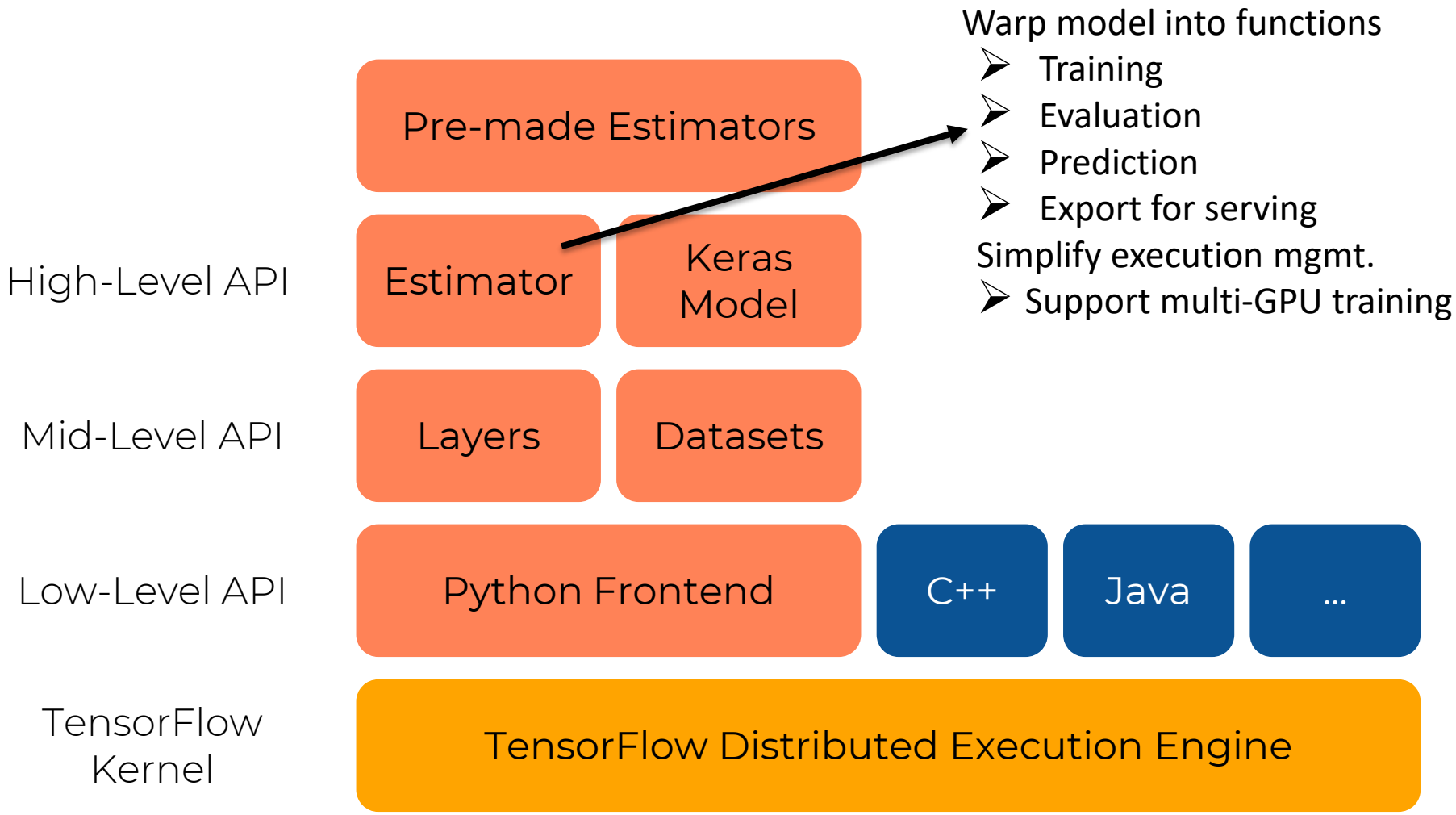
[22] "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems", Preliminary White Paper, November 9, 2015

TensorFlow Runtime

- TensorFlow runtime is a cross-platform library



TensorFlow Architecture



Build-in Strategy for TF Estimator

- These strategies can be called from Keras API as well

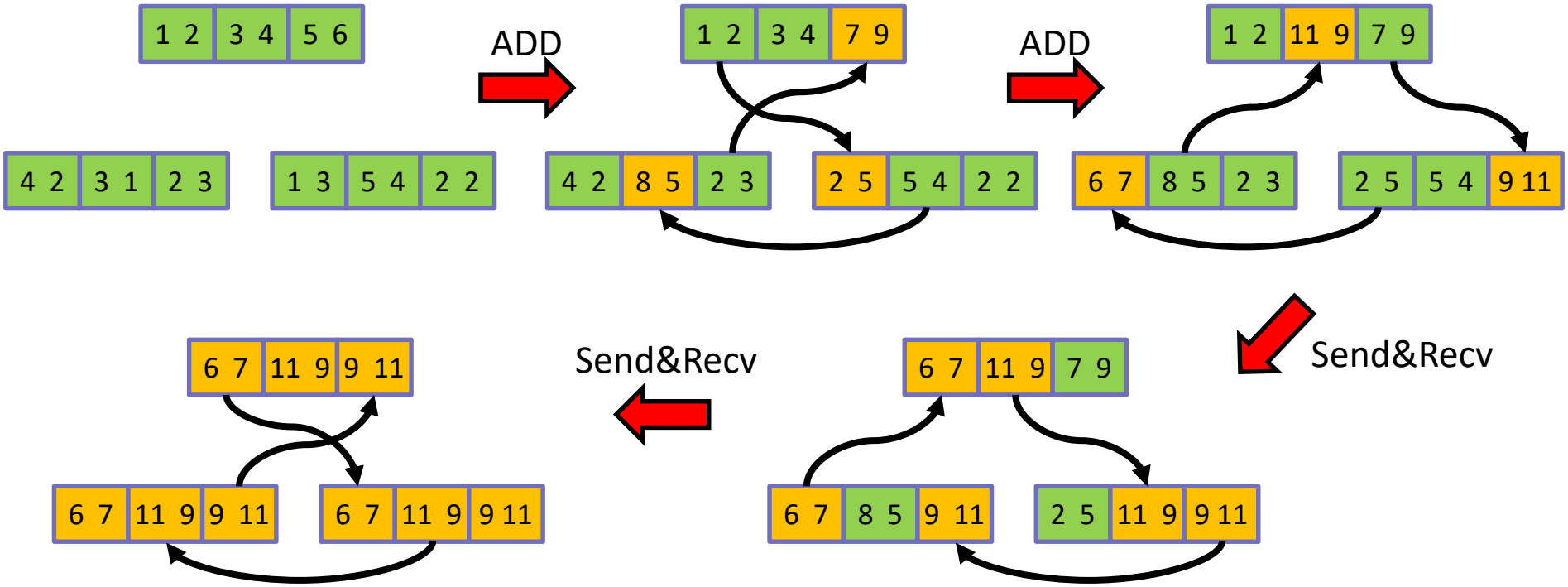
Strategy	Parallelism	Dataset	Comm.	Use Scenario
Mirrored	Single node	Replicated on GPU devices	Allreduce	Data parallelism on a single node
Central Storage	Single node	Keep on host (main memory)	Parameter server	When model is small
Multi Worker	Multi nodes	Replicated on GPU devices	Allreduce	HPC Env.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

Horovod

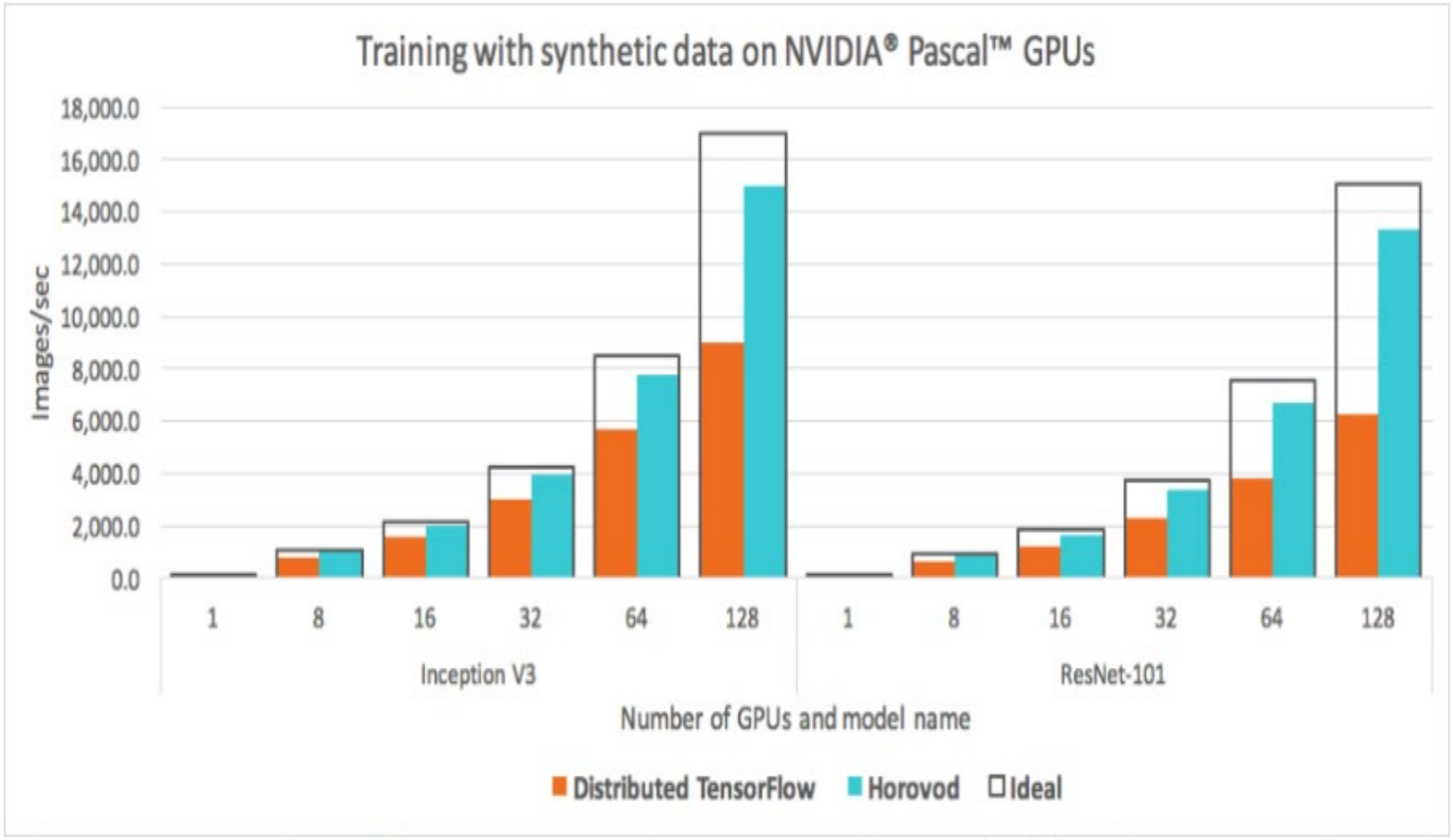
- Distributed training framework for
  - TensorFlow
  - Keras
  - PyTorch
- Separate infrastructure from ML computations
  - Executed like a traditional HPC parallel job
- Use bandwidth-optimal communication protocols
  - Implemented by HPC protocols: **MPI** and NVIDIA Collective Communications Library (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
ResNet-101
Number of GPUs and model name
Distributed TensorFlow
Horovod
Ideal

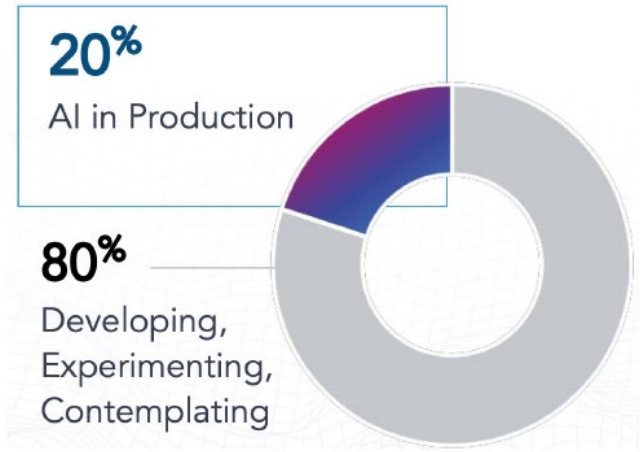
Outline

- Brief Introduction of Deep Learning
  - Computing Demand for Training
  - GPU Solutions
- Distributed Deep Learning Computations
  - Parallel strategies
  - Optimization strategies
  - Memory Optimization
  - TensorFlow & Horovod
- Trend & Future of Deep Learning Computing
  - ML Systems & MLOps
  - AI Cloud Services & AI Chips
  - Federated Learning
  - Remarks

Gaps Between Education & Enterprise

	Education	Enterprise
Objectives	Focus on Model Accuracy and use of Algorithm	Focus on Development/Integration with consideration of both Model Accuracy and Deployment Cost
Approach	Focus on Increasing Model Complexity for Better Accuracy	Keep Model Simple with Satisfied Accuracy
Data Source	Come from a Single or few Files	Come from multiple enterprise systems; need to be integrated, cross referenced, and summarized
Data Size	Typically Small to Medium	Range from Medium to Very Large
Data Characteristic	Typically 80% Clean	Start from raw data with 80% Noise
Tools & Dev. Env.	Limited tools, Standardized Env.	More tools + DevOps + Cloud + 3rd Party Solutions
Workflow	Fixed, Ad hoc	Agile, Dynamic

Growing AI Investments; Few Deployed at Scale



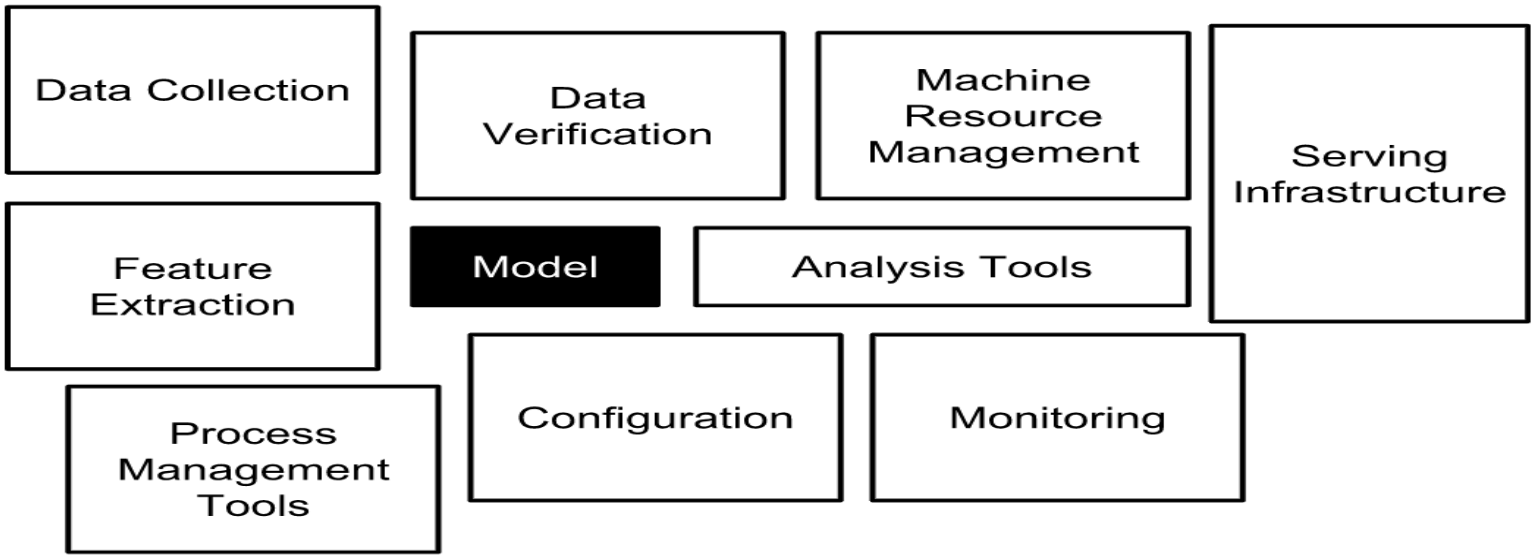
* Survey of 3073 AI-aware C-level Executives by Mckinsey

*Survey of 3073 AI-aware C-level Executives by Mckinsey

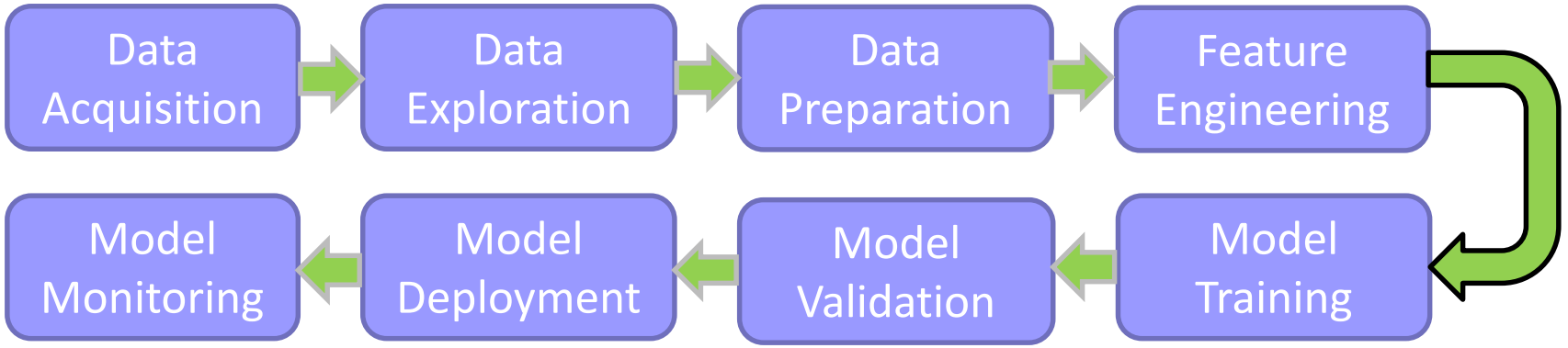
It is easy to get public model and dataset, but hard to deploy and operate the model in real world!!!

ML System

- There is a lot more to AI/ML than just implementing an **algorithm** or a **technique**
  - Less than 5% of the production code is for ML model
- We need **a system** to support, optimize, and automate the whole process



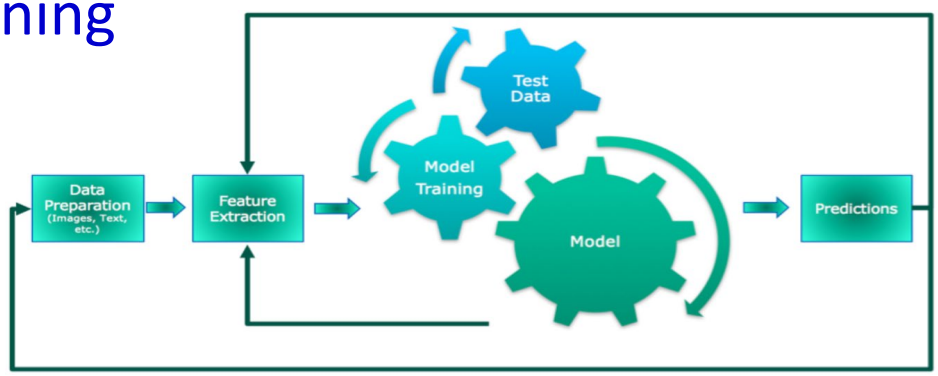
ML Pipeline



- A machine learning pipeline is a way to **codify** and **automate** the **workflow** it takes to produce a machine learning mode
- Machine learning pipelines consist of **multiple sequential steps** that do everything from data extraction and preprocessing to model training and deployment

ML Systems & MLOPs

- ML System: Designing and implementing the Systems that support ML models in real-world
  - Solve range of practical concerns that come with broader adoption
  - Optimize Hardware/Software systems for metrics beyond predictive accuracy
  - Foster a new systems machine learning research community at the intersection of the traditional systems and ML communities
- MLOPs: Continuous delivery and automation pipelines in machine learning



New Community for ML Systems

- Designing and implementing the **Systems** that support ML models in real-world
  - Solve range of practical concerns that come with broader adoption
  - Optimize Hardware/Software systems for metrics beyond predictive accuracy
  - Foster a new systems machine learning research community at the intersection of the traditional systems and ML communities
- Research topics includes…
  - Efficient distributed training/inference
  - AI chip design & optimization
  - Systems to support the full machine learning lifecycle management
  - A lot more to be explored…

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

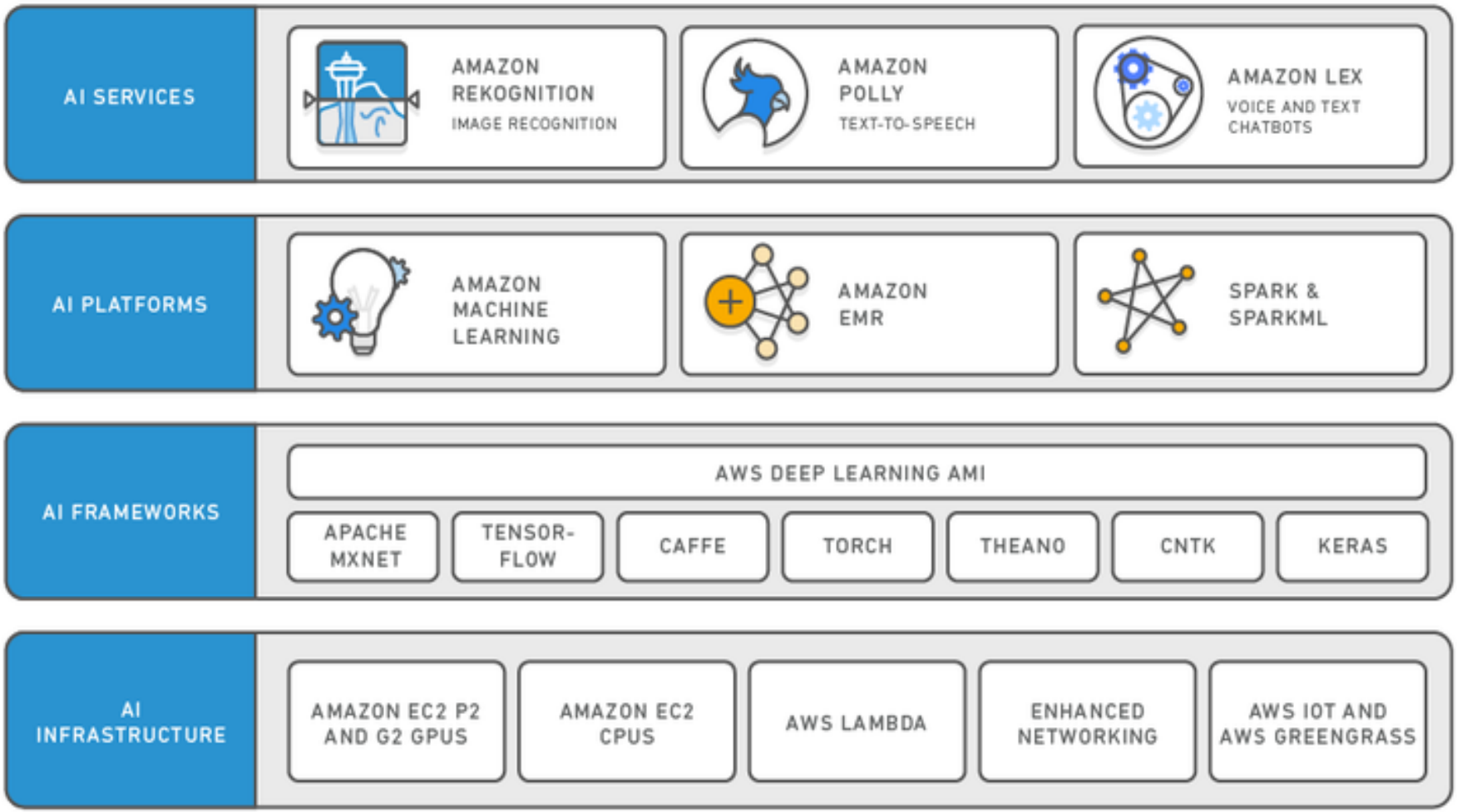
HPC (GPU)

Edge node

Cloud services (AI Chip: ASIC)

Embedded (AI Chip: ASIC, FPGA)

Modern AI Solutions on Public Clouds



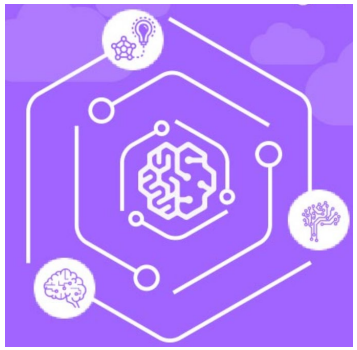
Platform as a Service: AI Platforms

- Platform as a Service:
  - **Support ML Pipeline** from data pre-processing to service lifecycle management
  - **Hide infrastructure** and **management** details from users
    - **Automated management** for computing resources and jobs
    - Deployable on both **public and private clouds**

Azure ML Studio / Machine Learning Service

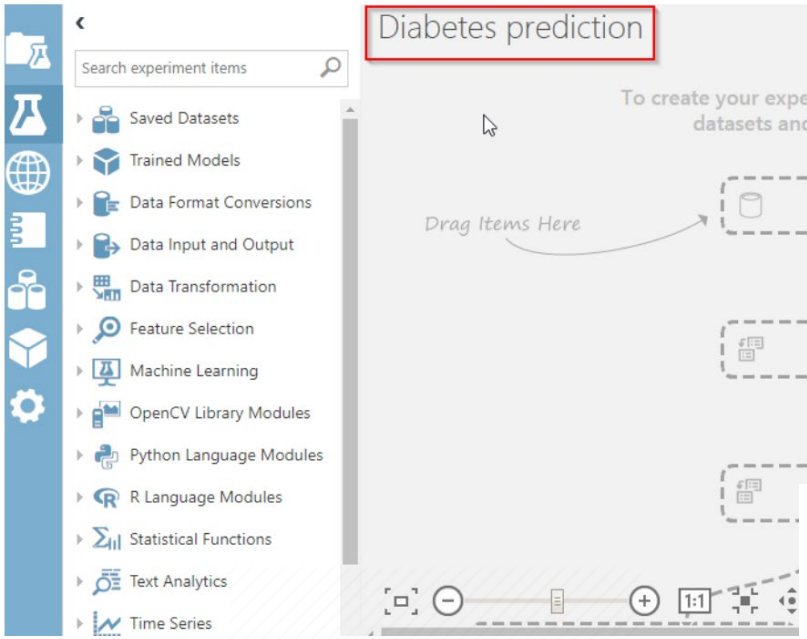
Google AI Platform

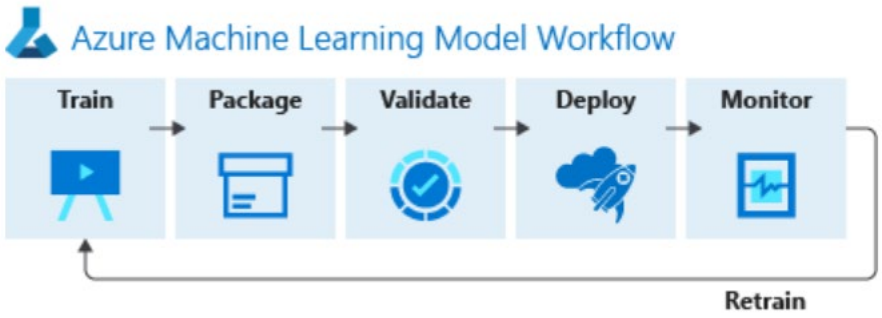
AWS Sagemaker

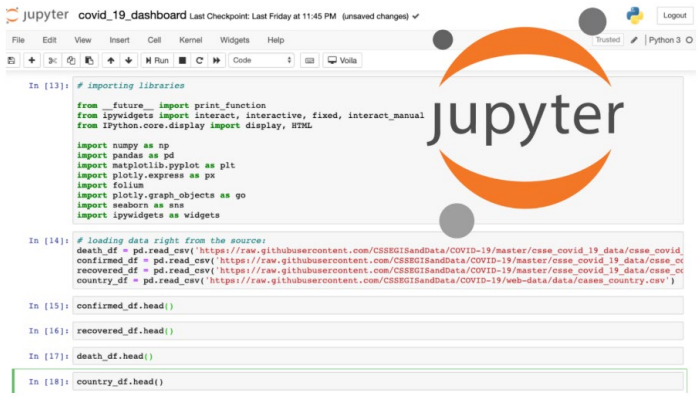


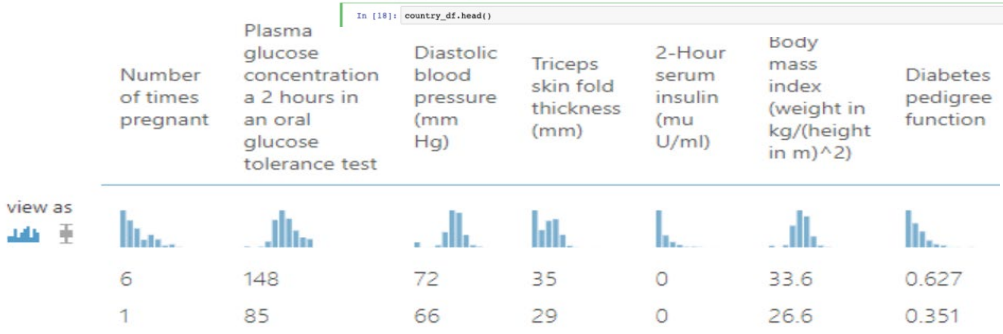
Azure ML Service

Coding







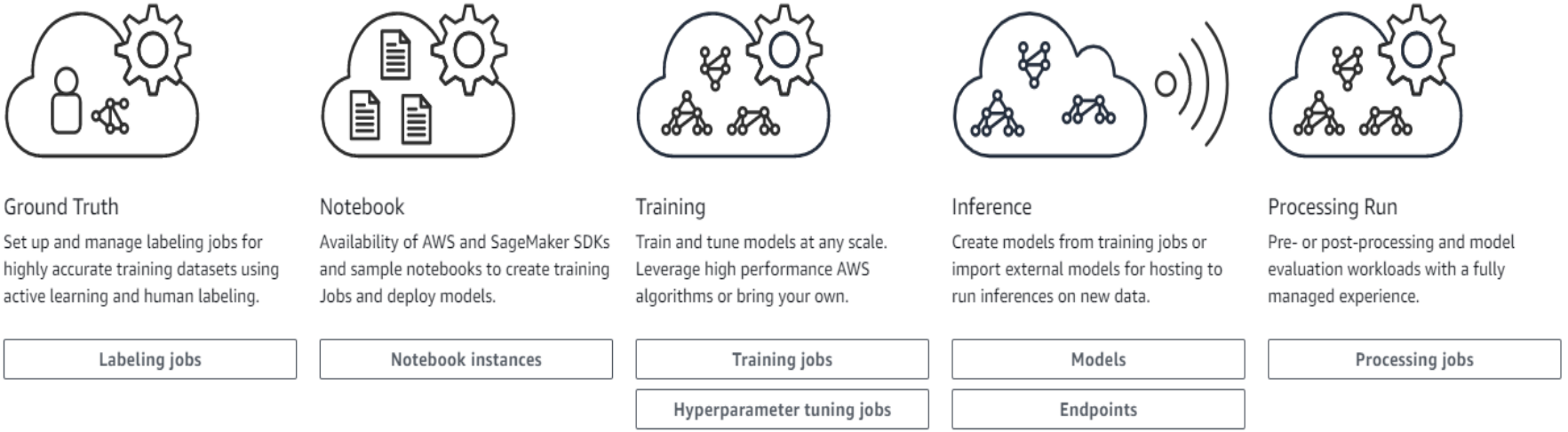


Executing

Monitoring, Logging, Visualization

AWS Sagemaker & Google AI Platform

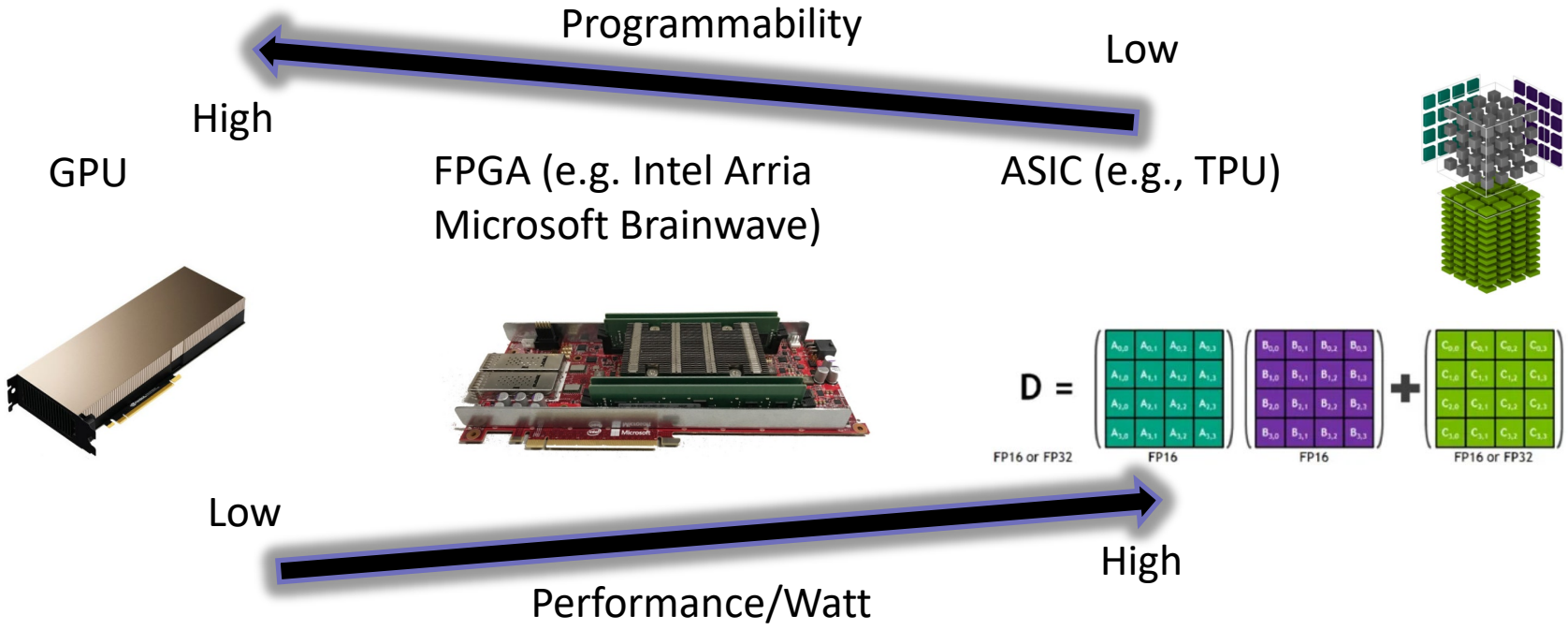
AWS Sagemaker



Google AI Platform

Specialized AI Chip

- Lower cost , Smaller size, Lower power consumption, Higher performance

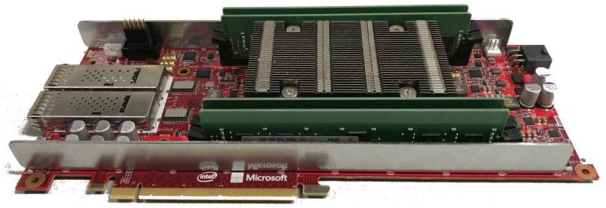


AI Chip for Inference

- 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)



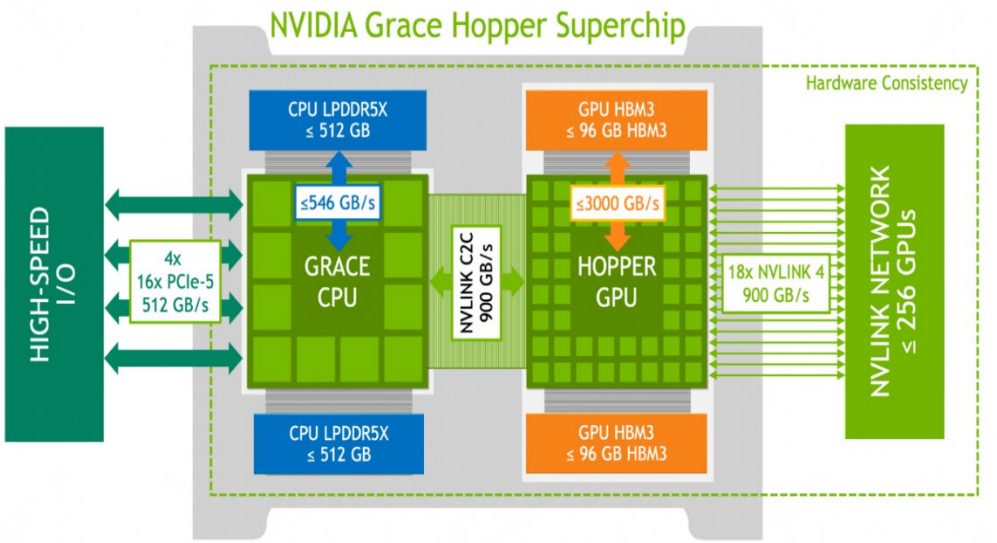
Microsoft's BrainWave (FPGA)

Google Tensor Processing Unit (TPU)

- Specifically for deep learning (tensorflow framework)
- 30–80X higher performance-per-watt than contemporary CPUs and GPUs
  - Only for **reduced precision computation** (e.g. 8-bit precision)
  - Matrix Multiplier Unit: use a to achieve hundreds of thousands of **matric operation** in a single clock cycle
  - Systolic array: The ALUs perform **only multiplications and additions in fixed patterns**
- Reference
  - https://cloud.google.com/blog/big-data/2017/05/an-in-depth-look-at-googles-first-tensor-processing-unit-tpu

NVIDIA Grace Hopper Superchip

- Massive Bandwidth for Compute Efficiency
- Using NVIDIA® NVLink®-C2C to deliver a CPU+GPU coherent memory model for accelerated AI and HPC applications.
- High-speed I/O
- HBM3 memory
- On-chip fabrics
- System-on-chip (SoC)
- ARM-based processors

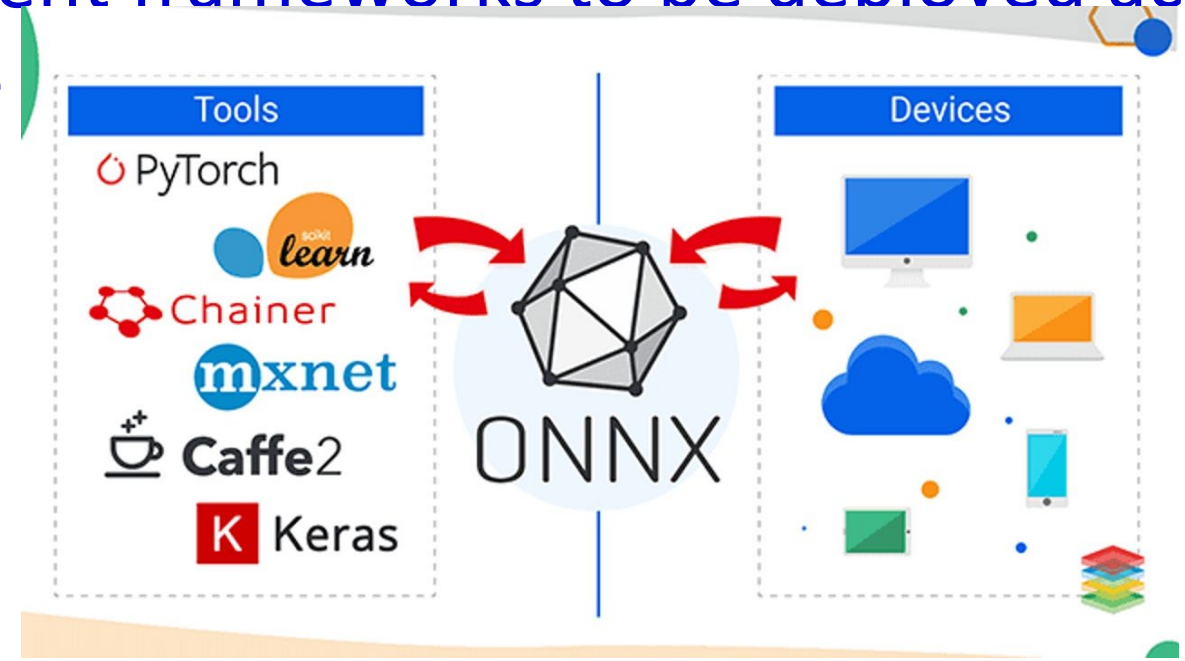


AI Chip for Inference

- "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

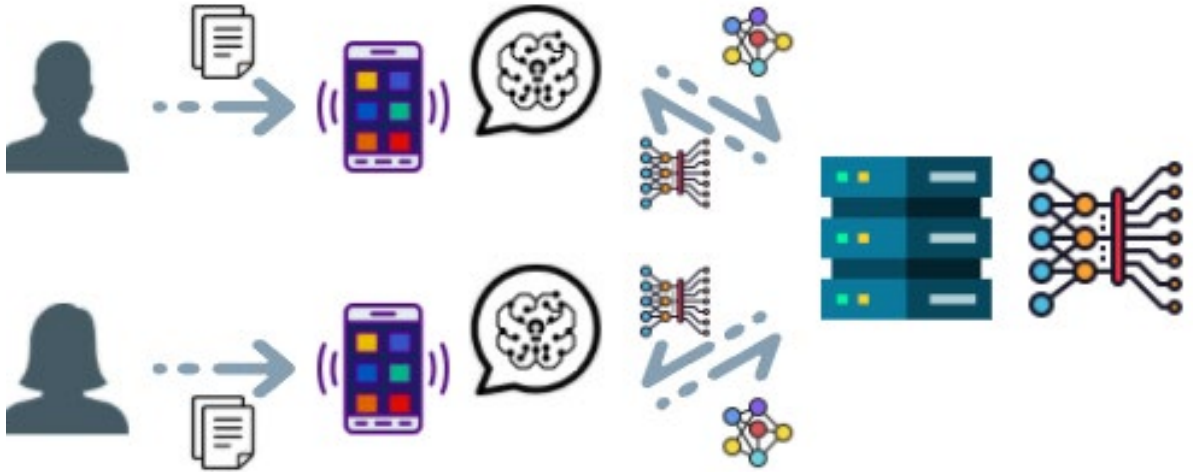
Interoperability

- ONNX is an interoperability layer that enables machine learning models trained using different frameworks to be deployed across a range



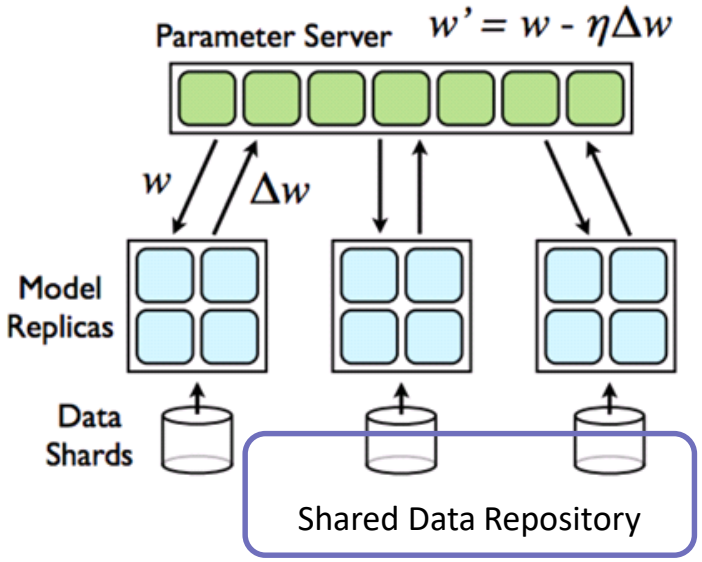
FL: Federated Learning

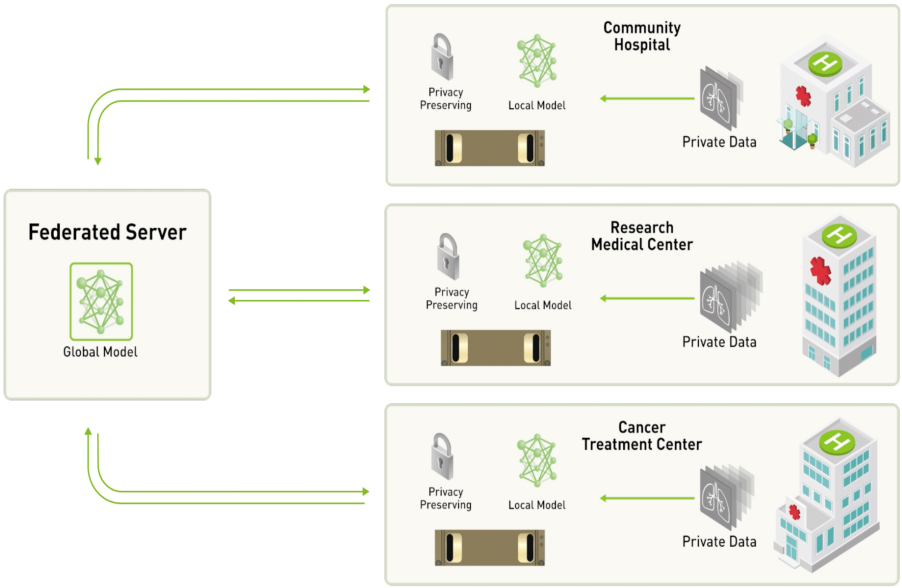
- Definition: Federated Learning is a machine learning setting (model training technique) where multiple entities (clients) collaborate in solving a machine learning problem, under the coordination of a central server or service provider. Each client's raw data is stored locally and not exchanged or transferred; instead, focused updates intended for immediate aggregation are used to achieve the learning objective.



What is FL?

- Moving from decentralized COMPUTATION to decentralized DATA





Distributed Training with Data Parallelism

Federated Learning with Decentralized Data

Learning Procedure of FL

- Initialization: according to the server inputs, a machine learning is chosen to be trained on local nodes and initialized. ➔ What model to be trained?
- Client selection: a fraction of local nodes is selected to start training on local data ➔ Who should involve training?
- Configuration: the central server orders selected nodes to undergo training of the model on their local data in a pre-specified fashion. ➔ How to perform training?
- Reporting: each selected node sends its local model to the server for aggregation. ➔ How to ensure training requirements like fault tolerance, performance, privacy?
- Aggregation: the central server aggregates the received models and sends back the model updates to the nodes. ➔ How to collaborate training?
- Termination: once a pre-defined termination criterion is met the central server aggregates the updates and finalizes the global model. ➔ When to finish training?

Federated Average

- Most well known FL learning method
  - A SGD-based model-averaging method
  - Fully synchronized and parallel algorithm
  - Model is weighted average by the client data size

Algorithm 1 `FederatedAveraging`. The K clients are indexed by k; B is the local minibatch size, E is the number of local epochs, and η is the learning rate.

Server executes:
initialize w_0
for each round $t = 1, 2, \dots$ **do**
  $m \leftarrow \max(C \cdot K, 1)$
  $S_t \leftarrow$ (random set of m clients) → Client selection
  for each client $k \in S_t$ **in parallel do**
    $w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$ → Reporting
  $w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$ → Aggregation weighted by data size

ClientUpdate(k, w): *// Run on client k*
  $\mathcal{B} \leftarrow$ (split $\mathcal{P}_k$ into batches of size B)
  for each local epoch i from 1 to E **do** → Configuration
    for batch $b \in \mathcal{B}$ **do**
      $w \leftarrow w - \eta \nabla \ell(w; b)$
  return w to server

*Hongyi Wang, Mikhail Yurochkin, Yuekai Sun, Dimitris Papailiopoulos, Yasaman Khazaeni, "Federated Learning with Matched Averaging", accepted by ICLR 2020

Why FL is the Future of ML?

- Address the data governance and privacy concern of traditional ML
- Smarter model contributed by more collaborators and datasets
- Enable larger scale training with more efficient communication and limited data transfer
- Less training cost by offloading computations to client devices
- Lower inference latency by keeping updated model at client side
- Driven by the Cloud and Edge computing paradigm and applications

Applications of FL

- Cross-device: Clients are a very large number of mobile or IoT devices.
  - QuickType keyboard, e.g., Gboard
  - Vocal classifier for “Hey Siri”
  - Self-driving cars
  - IoT applications

Cross-device and **Cross-silo** can be different in their **data availability**, **scale**, **client statefulness**, and **performance bottleneck**

- Cross-silo: Clients are different organizations (e.g. medical or financial) or geo-distributed datacenters.
  - Healthcare
  - FinTech
  - Smart manufacturing

Application Domain with **Scalability** and **Privacy** concerns

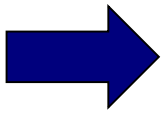
Challenges of FL: Incentive mechanisms

- How to divide earnings among contributing data owners in order to sustain long-term participation
- How to link the incentives with decisions on defending against adversarial data owners to enhance system security

Challenges of FL: Differential privacy

- Example
  - Have a table related to whether you have cancer
    - Have a function which support querying the number of people with cancer
    - Assume that the adversary knows that Jack is in the last row of the table
    - Using the difference between the two outputs, the adversary know that Jack has cancer.

patient	has cancer
Amy	0
Tom	1



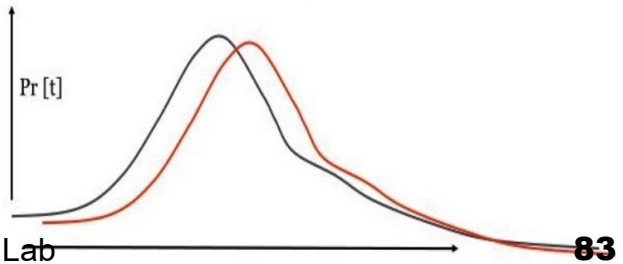
patient	has cancer
Amy	0
Tom	1
Jack	1

Challenges of FL: Differential privacy

- The effect of making an arbitrary single substitution in the database is small enough, the query result cannot be used to infer much about any single individual, and therefore provides privacy.
- An observer seeing its output cannot tell if a particular individual's information was used in the computation.
- ε-differential privacy: ε is the maximum distance between a query on database (D1) and the same query on database (D2).

$$\frac{\Pr[\mathcal{K}(\mathrm{DB} - \mathrm{Me}) = t]}{\Pr[\mathcal{K}(\mathrm{DB} + \mathrm{Me}) = t]} \leq e^{\varepsilon} \approx 1 \pm \varepsilon$$

$$\Pr[\mathcal{A}(D_1) \in S] \leq \exp(\epsilon) \cdot \Pr[\mathcal{A}(D_2) \in S]$$



Challenges of FL: Differential privacy

- Large noise level reduce the quality of the model

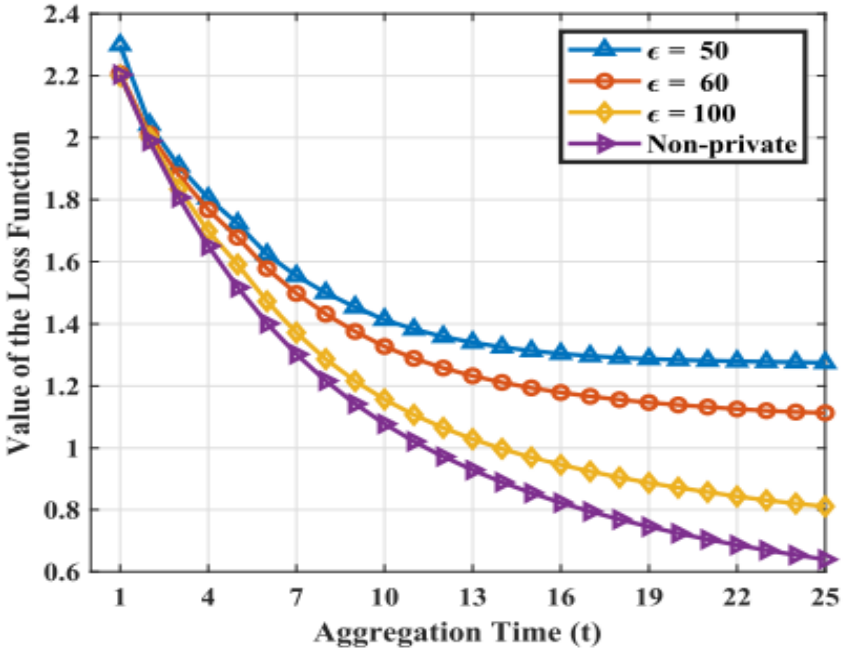


Fig. 2. The comparison of training loss with various protection levels for 50 clients using $\epsilon = 50$, $\epsilon = 60$ and $\epsilon = 100$, respectively.

Challenges of FL: Non-IID data

- Feature distribution skew (Covariate shift)
  - The marginal distributions Pi(x) may vary across clients, even if P(y j x) is shared
  - For example, users who write the same words might still have different stroke width, slant, etc.

- Label distribution skew (Prior probability shift)
  - The marginal distributions Pi(y) may vary across clients, even if P(x j y) is the same.
  - For example, kangaroos are only in Australia

Challenges of FL: Non-IID data

- Same label, different features (Concept shift)
  - The conditional distributions Pi(x j y) may vary across clients even if P(y) is shared.
  - For example, images of homes can vary dramatically around the world
- Same features, different label (Concept shift)
  - The conditional distribution Pi(y j x) may vary across clients, even if P(x) is the same.
  - For example, labels that reflect sentiment have personal and regional variation.
- Quantity skew (Unbalancedness)
  - Clients hold vastly different amounts of data

Challenges of FL: Performance & Efficiency

- Expensive Communication
  - Sparsification and/or quantization of model updates
  - Pipeline the model training phase with model uploading & downloading phase
  - Partial device participation
  - Models are updated locally at devices and only periodically averaged at the server
  - Asynchronous / Stale-synchronous model update
- Others
  - Client availability
  - Fault tolerance

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