https://astra-sim.github.io

https://github.com/mlcommons/chakra

ASTRA-sim Tutorial
@MICRO 2024
November 3, 2024

ASTRA-sim and Chakra Tutorial:
Introduction to Distributed ML

Tushar Krishna
Associate Professor
School of ECE, Georgia Institute of Technology
tushar@ece.gatech.edu

Welcome

Presenters

Tushar Krishna
Associate Professor, ECE
Georgia Tech
tushar@ece.gatech.edu

William Won
Ph.D. Candidate, CS
Georgia Tech
william.won@gatech.edu

Joongun Park
Post Doctoral Researcher
Georgia Tech
jpark3234@gatech.edu

Taekyung Heo
Senior HPC Middleware Developer, NVIDIA
theo@nvidia.com

Vinay Ramakrishnaiah
Senior Member of Technical Staff. AMD
vinay.ramakrishnaiah@amd.com

Contributors & Collaborators

Georgia Tech
Jinsun Yoo
Changhai Man
Ziwei Li
Divya Kiran Kadiyala

Meta
Saeed Rashidi
Louis Feng
Sheng Fu
Brian Coutinho
Darshan Sanghani
Adi Gangidi

NVIDIA
Srinivas Sridharan

Intel
Sudarshan Srinivasan

AMD
Ruchi Shah
Brad Beckmann
Furkan Eris
+more

07.31.2023 — San Francisco, CA

Chakra: Advancing Benchmarking and Co-design for Future AI Systems

Announcing Chakra, execution traces and benchmarks working group

+ many more industry/academic researchers & engineers

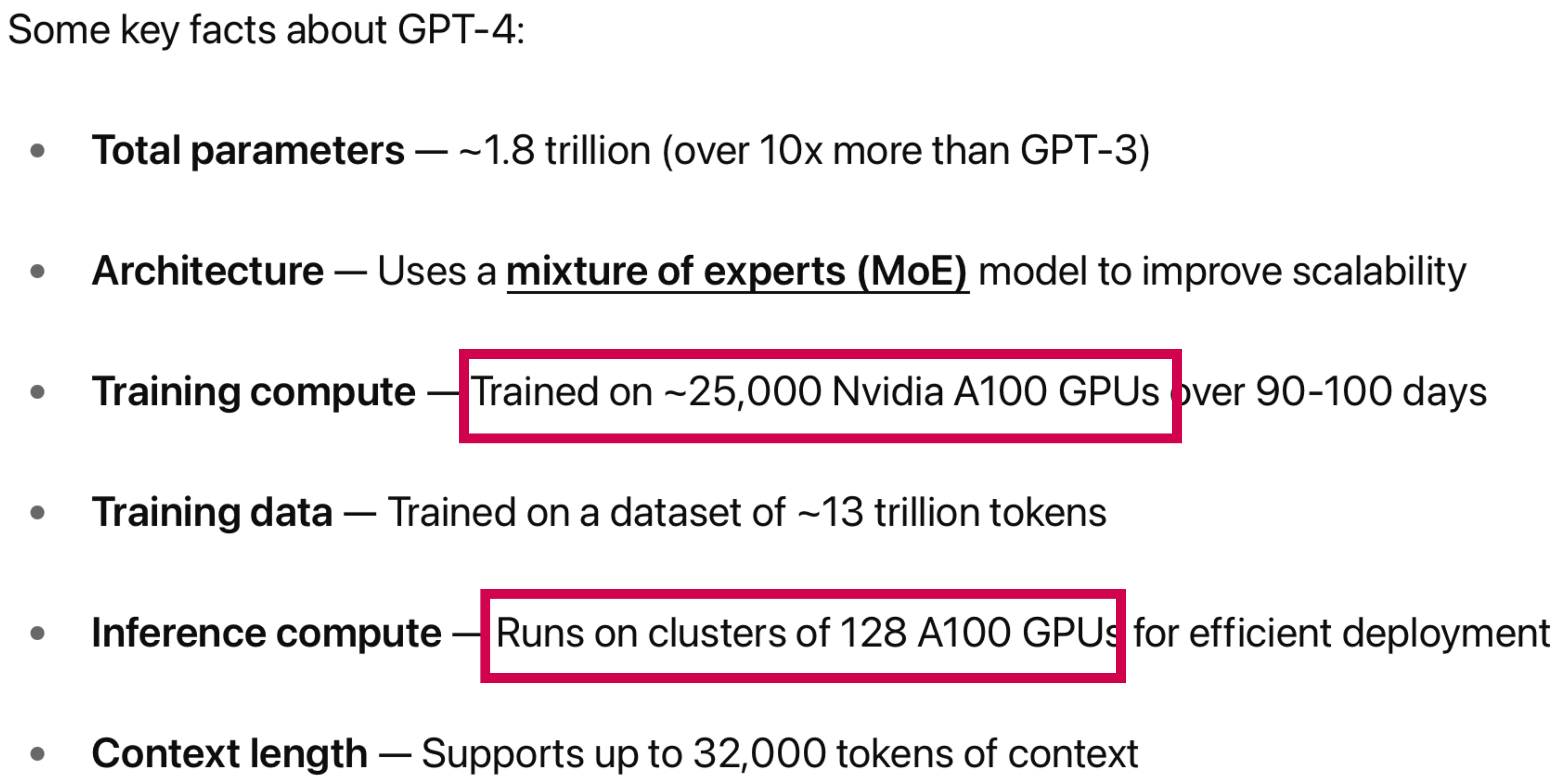
ASTRA-sim Tutorial - Agenda

Time (CST)	Topic	Presenter
1:00 pm	**Overview, Introduction to Distributed ML**	Tushar Krishna (Georgia Tech)
1:40 pm	**Chakra Execution Trace, ASTRA-sim Workload Layer**	Taekyung Heo (NVIDIA)
2:20 pm	**ASTRA-sim System Layer and Network Layer**	William Won (Georgia Tech/AMD)
3:00 pm	**Coffee Break**	
3:30 pm	**Demo: Chakra and ASTRA-sim**	Joongun Park (Georgia Tech)
4:10 pm	**ASTRA-sim New Features**	Vinay Ramakrishnaiah (AMD)
4:40 pm	**ASTRA-sim Wiki and Validation**	William Won (Georgia Tech/AMD)
4:50 pm	**Closing Remarks**	Tushar Krishna (Georgia Tech)

Tutorial Website
includes agenda, slides, ASTRA-sim installation instructions (via source + docker image)
https://astra-sim.github.io/tutorials/micro-2024

Attention: Tutorial is being recorded

AI has become a distributed system problem!



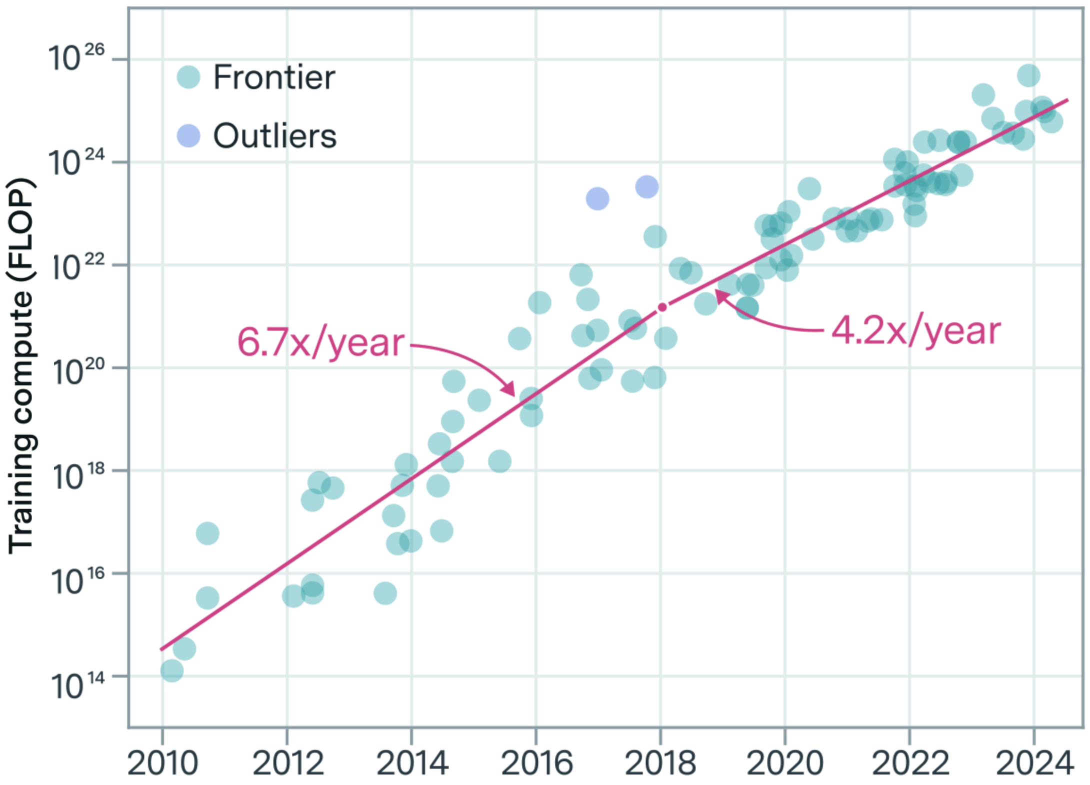
Some key facts about GPT-4:

- **Total parameters** — ~1.8 trillion (over 10x more than GPT-3)
- **Architecture** — Uses a **mixture of experts (MoE)** model to improve scalability
- **Training compute** — Trained on ~25,000 Nvidia A100 GPUs over 90-100 days
- **Training data** — Trained on a dataset of ~13 trillion tokens
- **Inference compute** — Runs on clusters of 128 A100 GPUs for efficient deployment
- **Context length** — Supports up to 32,000 tokens of context

Trend 1: Large ML Models

- ML models are scaling at an unprecedented rate

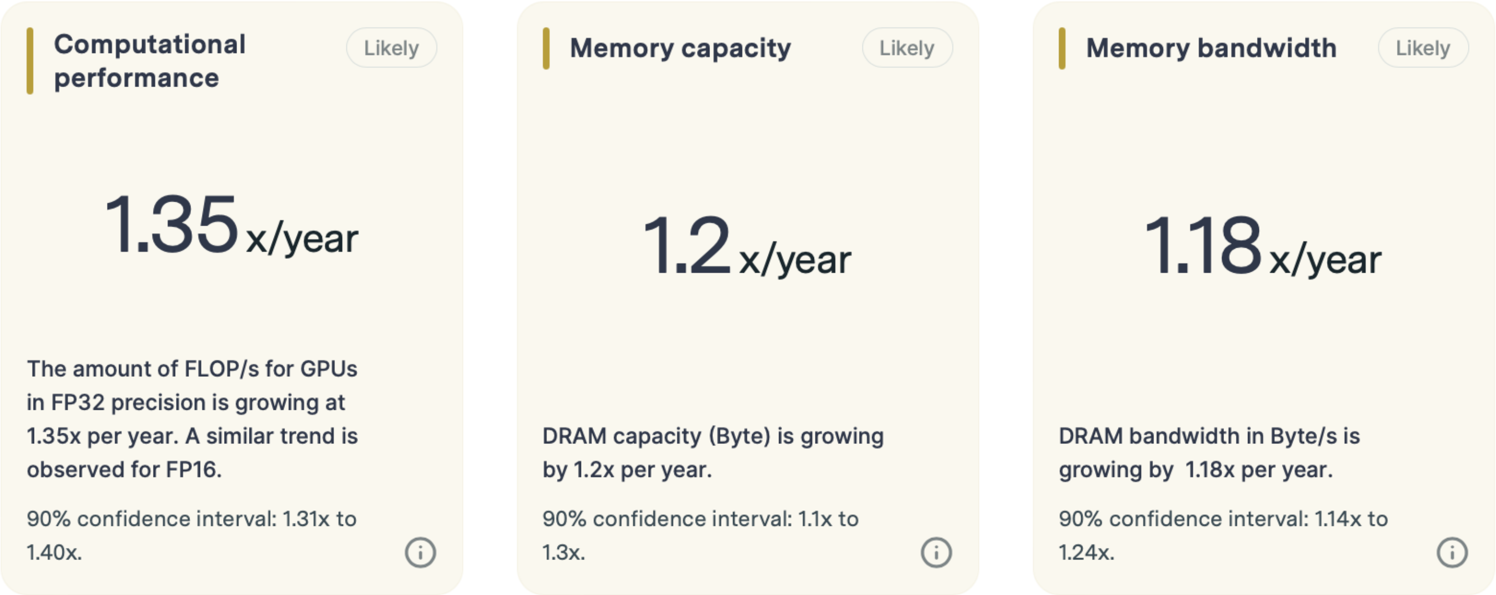
Frontier models



https://epochai.org/trends

Trend 2: Moore's Law

- Cannot simply rely on device scaling



Computational performance Likely

1.35 x/year

The amount of FLOP/s for GPUs in FP32 precision is growing at 1.35x per year. A similar trend is observed for FP16.

90% confidence interval: 1.31x to 1.40x.

Memory capacity Likely

1.2 x/year

DRAM capacity (Byte) is growing by 1.2x per year.

90% confidence interval: 1.1x to 1.3x.

Memory bandwidth Likely

1.18 x/year

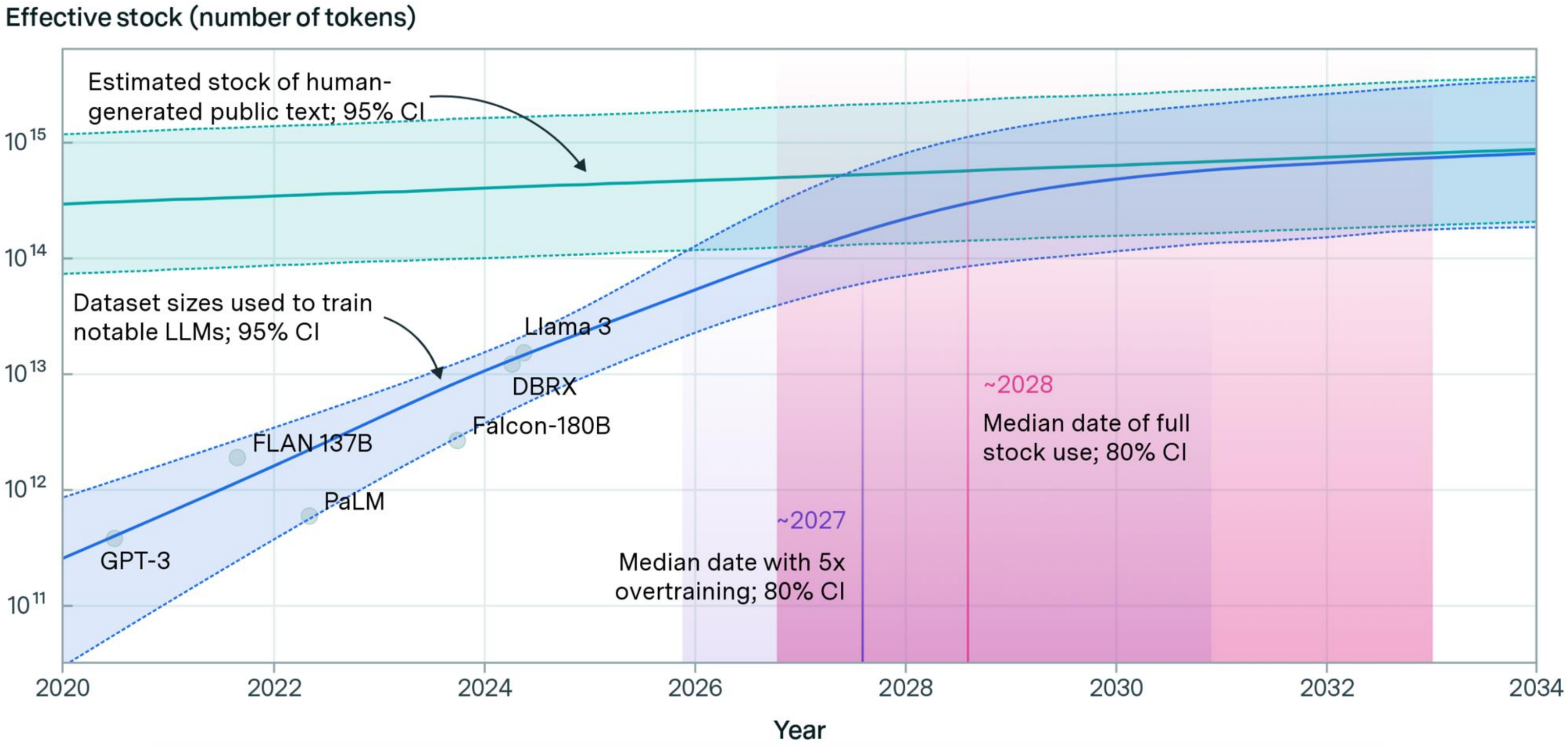
DRAM bandwidth in Byte/s is growing by 1.18x per year.

90% confidence interval: 1.14x to 1.24x.

https://epochai.org/trends

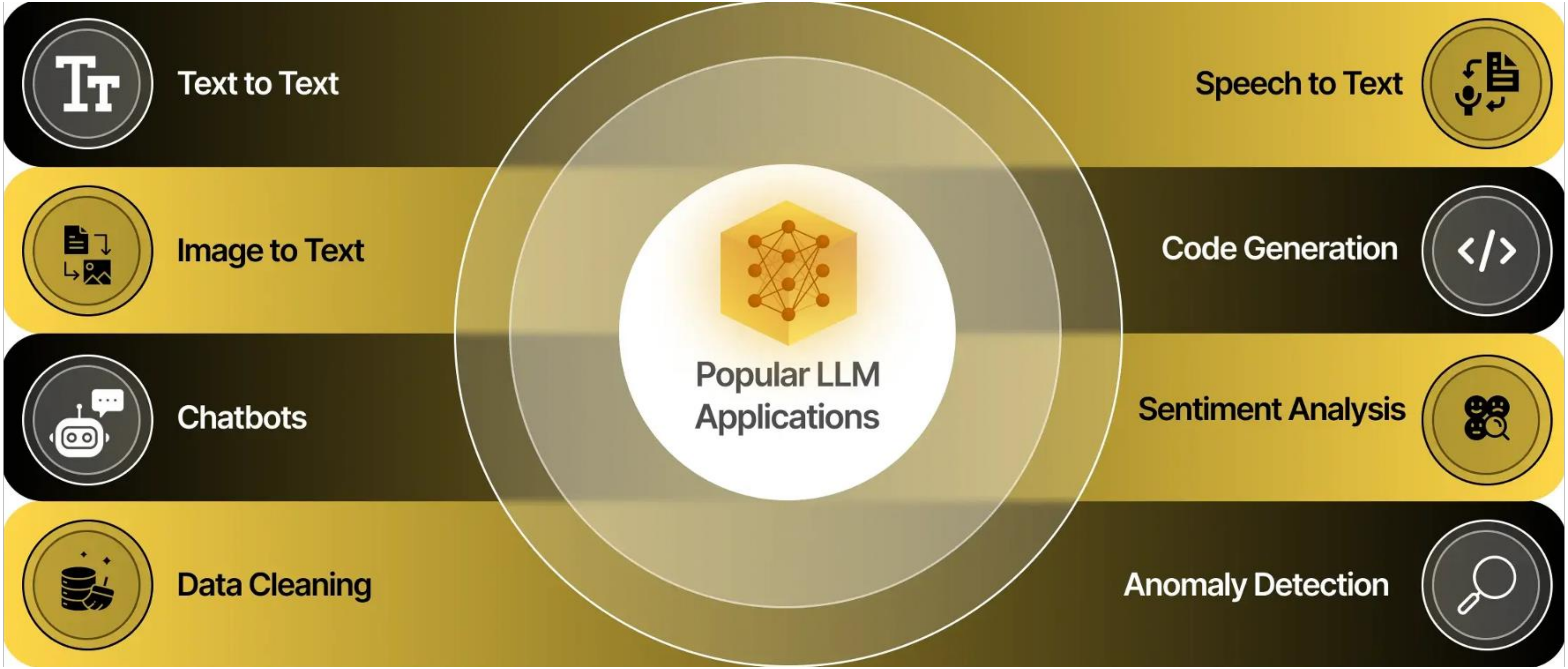
Trend 3: Training Dataset

- Huge training dataset



https://epochai.org/trends

Trend 4: Diverse Serving Use Cases



Source: https://markovate.com/blog/applications-and-use-cases-of-llm/

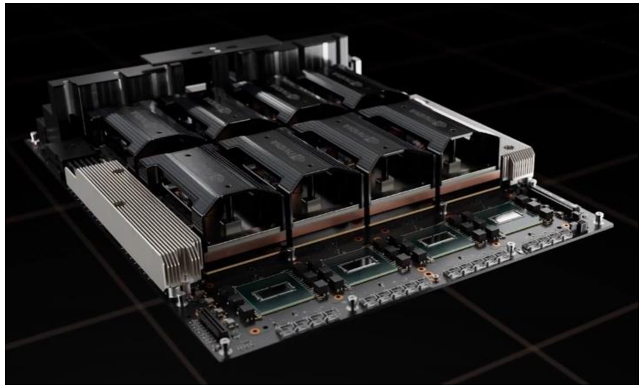
System Implications

- Multiple devices are required to accommodate large-scale ML

- **Compute**
  - In total, **21 YFLOP** for training (GPT-4)
  - Single NVIDIA H100 (2 PFLOPS) → **333 years** to train

- **Memory**
  - **1.8 trillion** parameters (GPT-4)
  - Assuming 2B/param, **3.6 TB** just to store the model
  - H100 HBM (80 GB) → **45 GPUs** just to *fit* the model itself

HPC Platforms for Distributed ML (*aka* AI Supercomputers)



NVIDIA HGX-H100 SuperPod

Google Cloud TPUv4

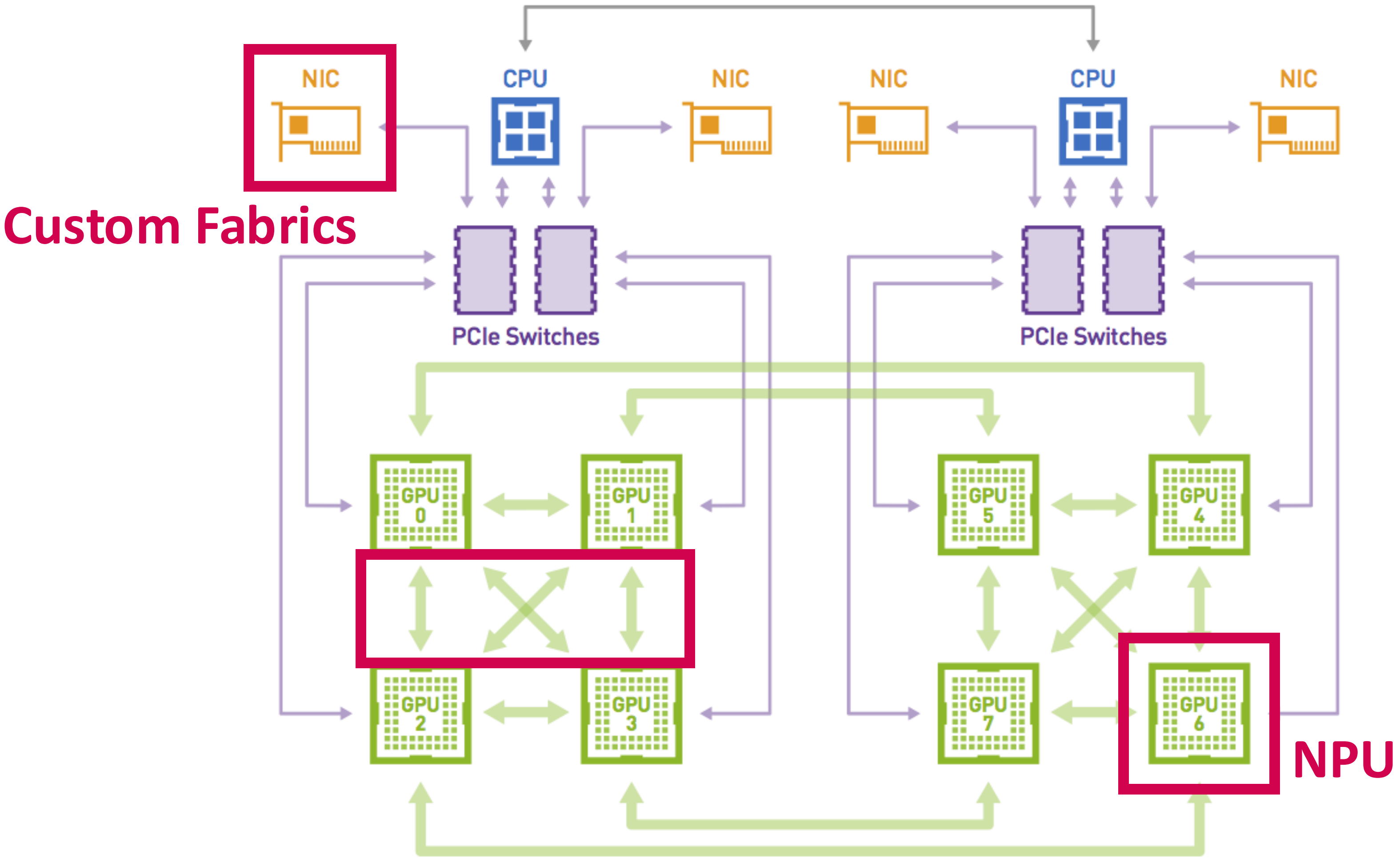
AMD Instinct Platforms

Intel Aurora Supercomputer

And many many more …

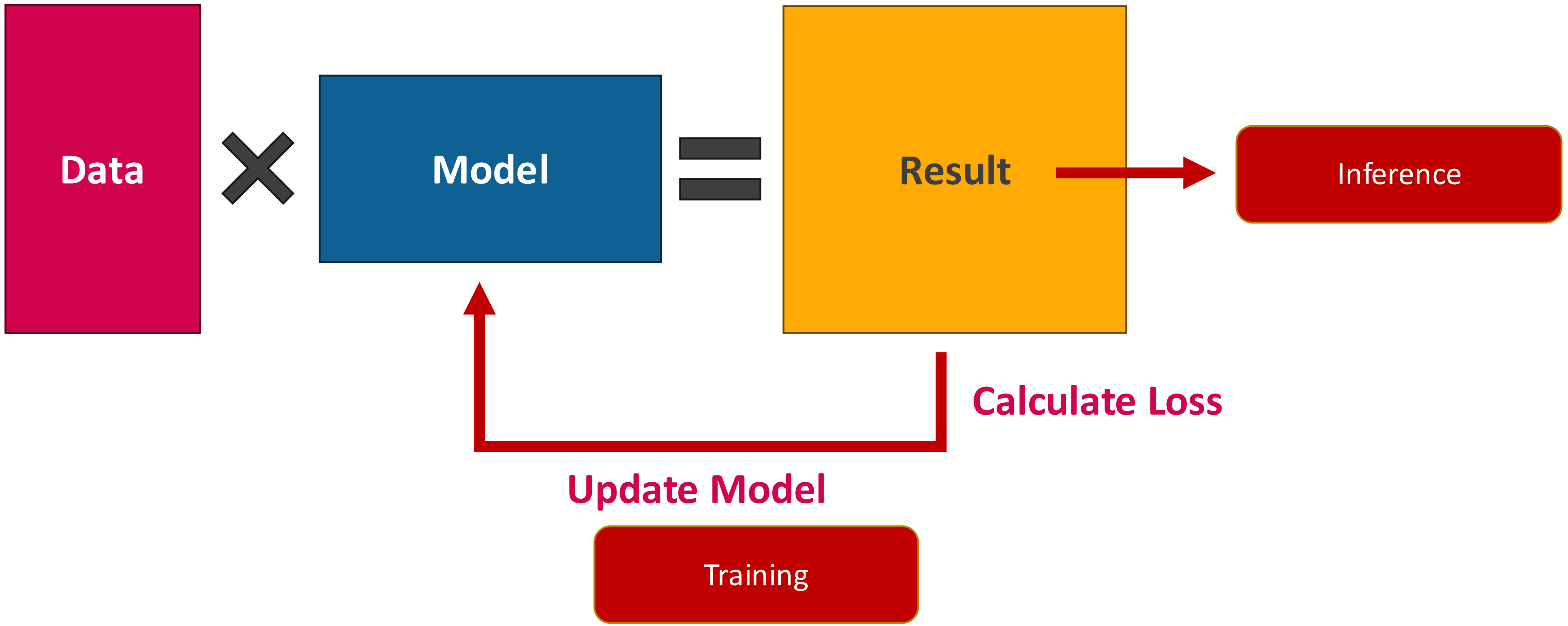
- xAI Collossus
- Cerebras Andromeda
- Tesla Dojo
- IBM BlueConnect
- …

Components of AI Platforms



https://developer.nvidia.com/blog/dgx-1-fastest-deep-learning-syste/

Core of ML Execution



Data × Model = Result

Inference

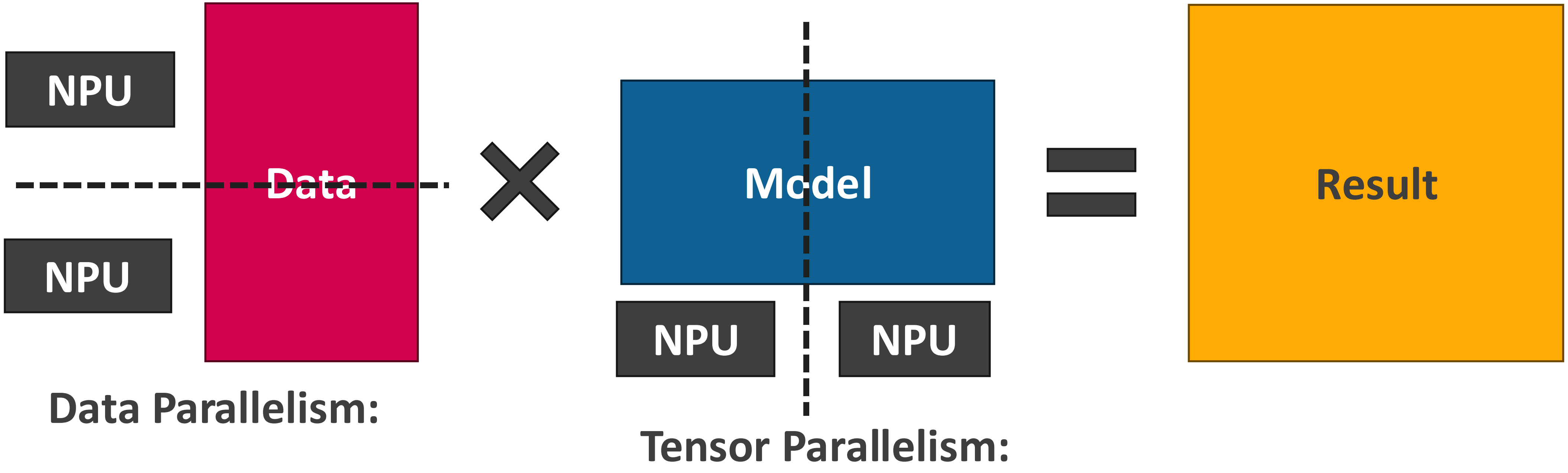
Calculate Loss

Update Model

Training

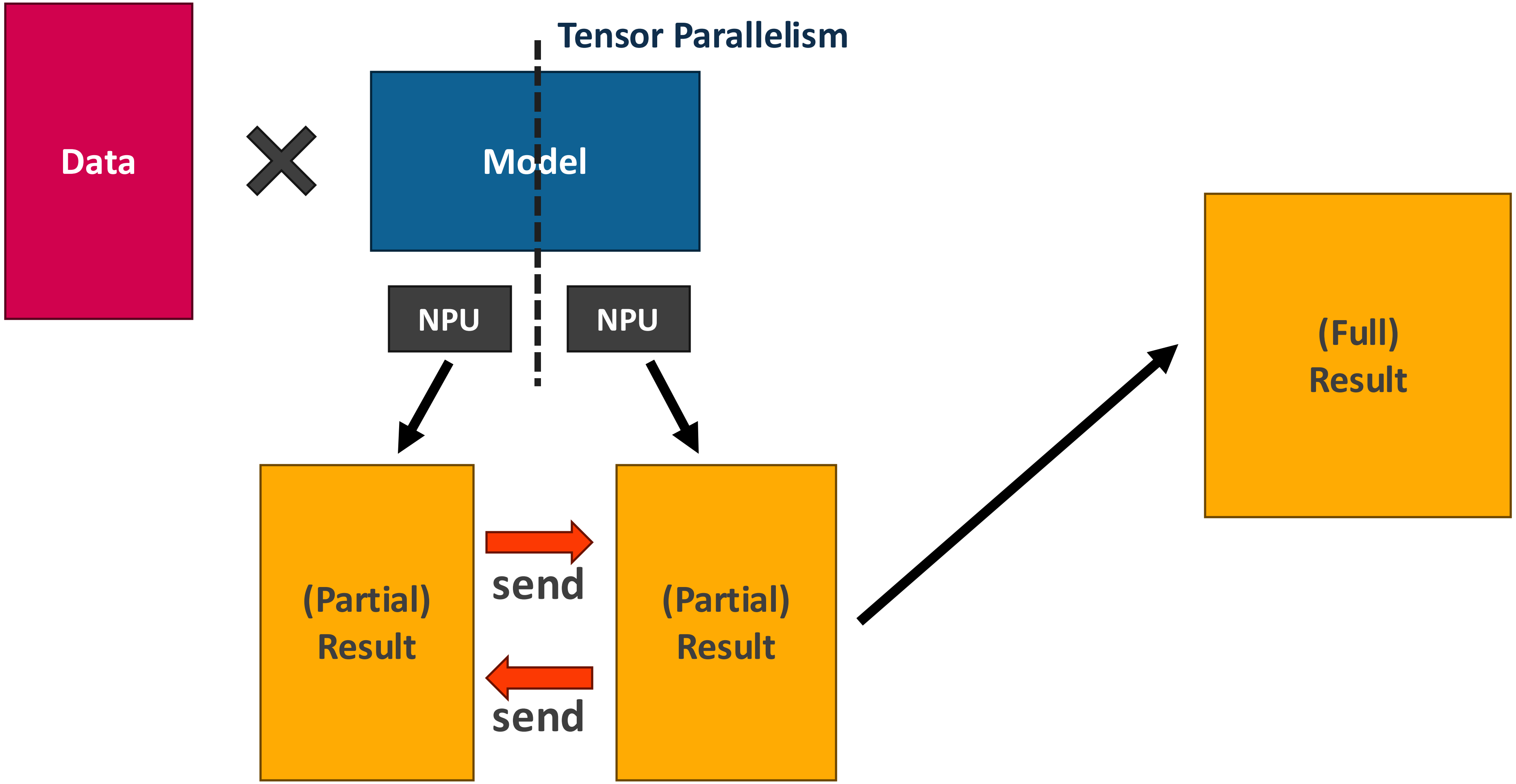
Distributed ML

- Model and/or data should be distributed
  - Across different NPUs (Neural Processing Unit)



Communication in Distributed ML

- NPUs should communicate to synchronize data



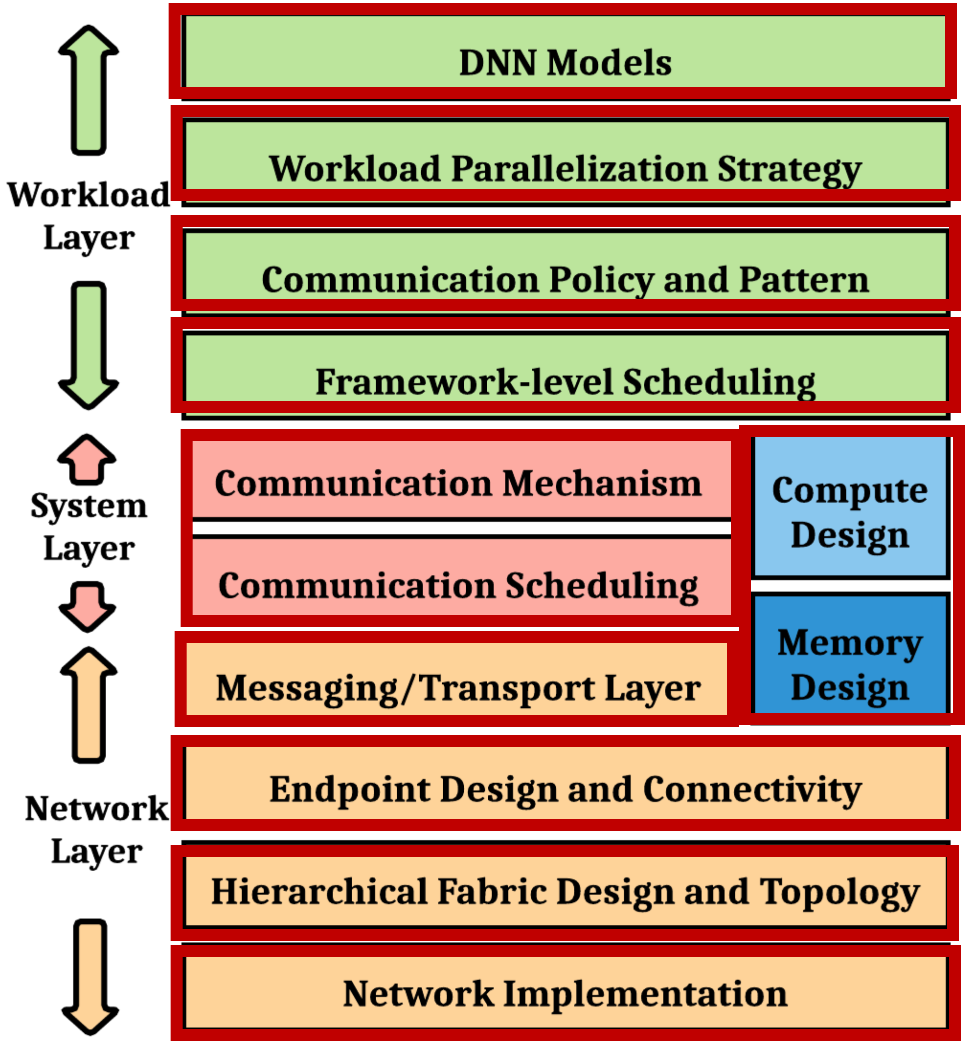
Systems challenges with Distributed Training

- Communication!
  - Inevitable in any distributed algorithm

- What does communication depend on?
  - **synchronization scheme:** synchronous vs. asynchronous.
  - **parallelism approach:** data-parallel, model-parallel, hybrid-parallel., ZeRO ...

- Is it a problem?
  - Depends … can we hide it behind compute?
  - *How do we determine this?*

Understanding DL Training design-space



Workload Layer

DNN Models

Workload Parallelization Strategy

Communication Policy and Pattern

Framework-level Scheduling

System Layer

Communication Mechanism

Communication Scheduling

Compute Design

Memory Design

Messaging/Transport Layer

Network Layer

Endpoint Design and Connectivity

Hierarchical Fabric Design and Topology

Network Implementation

→ DLRM, ResNet-50, Transformer, GNMT

→ Data, Model, Platform Agnostic Hybrid, Platform-aware Hybrid, Pipelined Parallelism

→ Distributed worker, Parameter Server

→ Sync/Async, Blocking/Non-blocking

→ Topology-aware Collectives, Send/Recv, RPC

→ Dataflow, Microarchitecture, Flexibility, Sparsity Support

→ LIFO, FIFO, Fusion

→ TCP, RDMA (+ GPUDirect RDMA)

→ # links, BW per link, architecture (chip/package/board), NIC offload, compression

→ Flat vs. Hierarchical, 2D/3D/4D Torus (TPU), Switch (DGX2), Fully-Connected, Hyper-Cube Mesh

→ Buffering, Flow-control, Arbitration, Congestion Mgmt

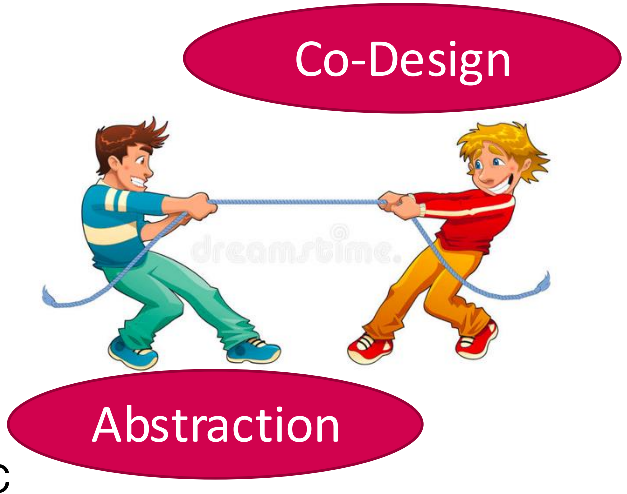
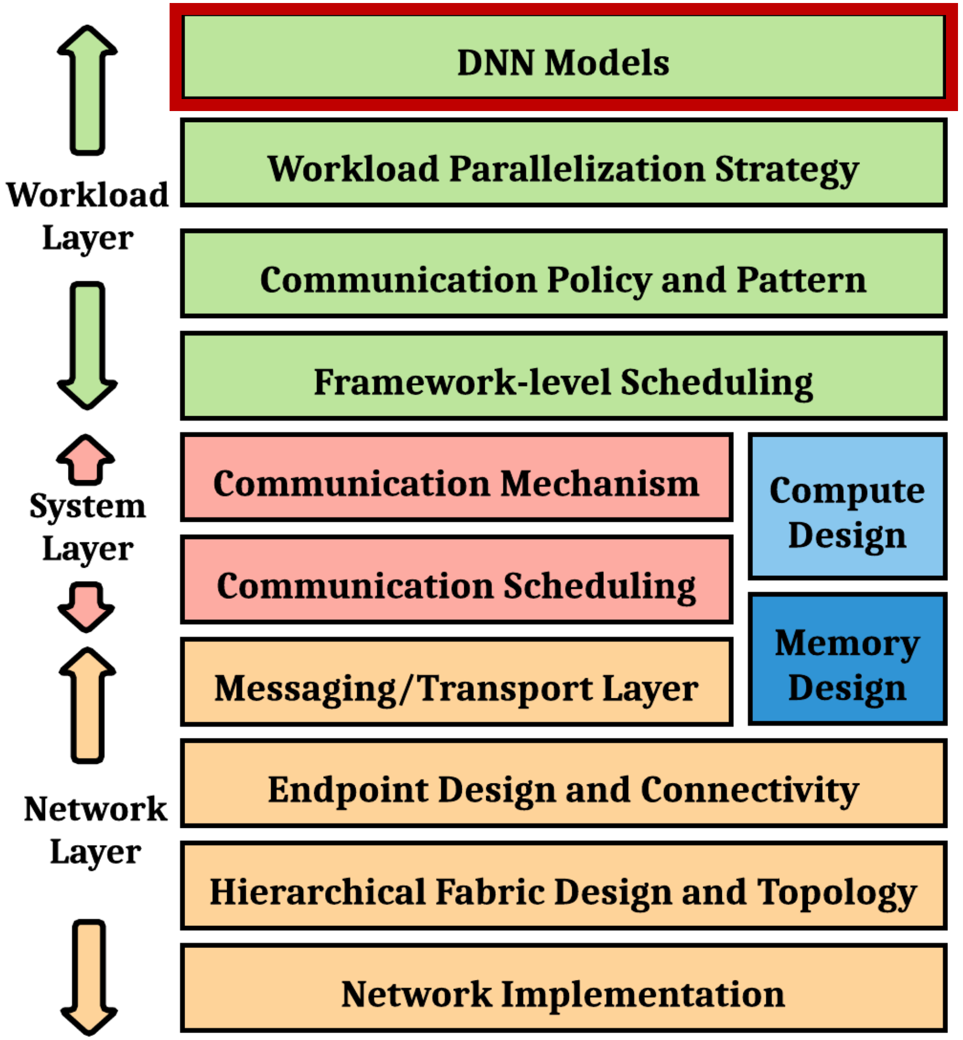


Figure Courtesy: Srinivas Sridharan (NVIDIA)

Distributed Training Stack



Workload Layer

DNN Models → DLRM, ResNet-50, Transformer, GNMT

Workload Parallelization Strategy → Data, Model, Platform Agnostic Hybrid, Platform-aware Hybrid, Pipelined Parallelism

Communication Policy and Pattern → Distributed worker, Parameter Server

Framework-level Scheduling → Sync/Async, Blocking/Non-blocking

System Layer

Communication Mechanism → Topology-aware Collectives, Send/Recv, RPC

Compute Design → Dataflow, Microarchitecture, Flexibility, Sparsity Support

Communication Scheduling → LIFO, FIFO, Fusion

Memory Design

Messaging/Transport Layer → TCP, RDMA (+ GPUDirect RDMA)

Network Layer

Endpoint Design and Connectivity → # links, BW per link, architecture (chip/package/board), NIC offload, compression

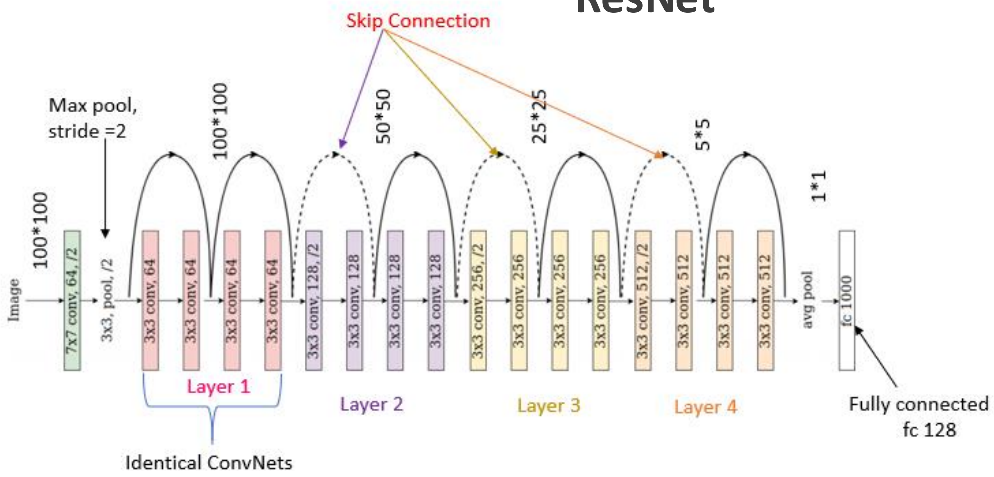
Hierarchical Fabric Design and Topology → Flat vs. Hierarchical, 2D/3D/4D Torus (TPU), Switch (DGX2), Fully-Connected, Hyper-Cube Mesh

Network Implementation → Buffering, Flow-control, Arbitration, Congestion Mgmt

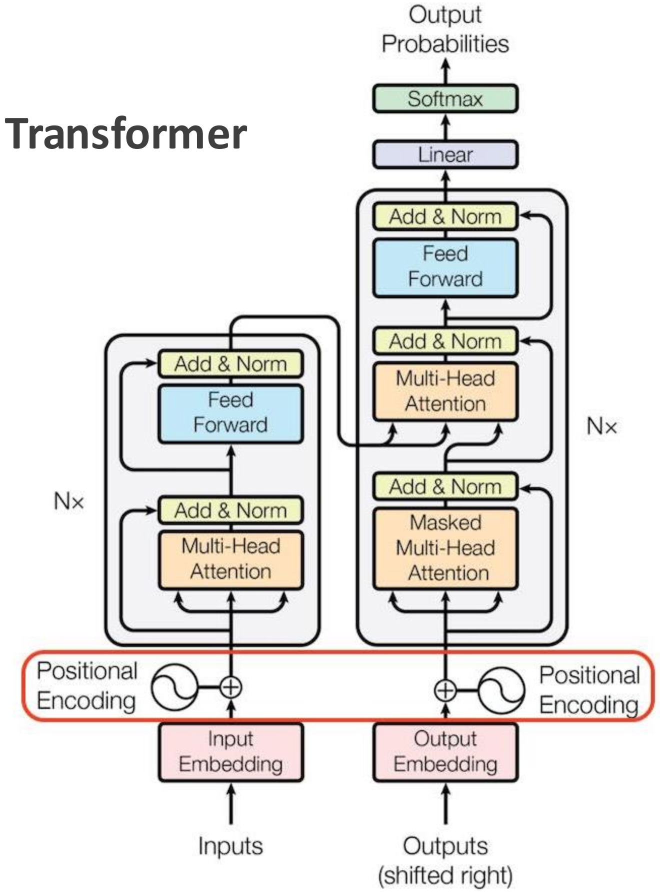
Figure Courtesy: Srinivas Sridharan (NVIDIA)

DNN Models

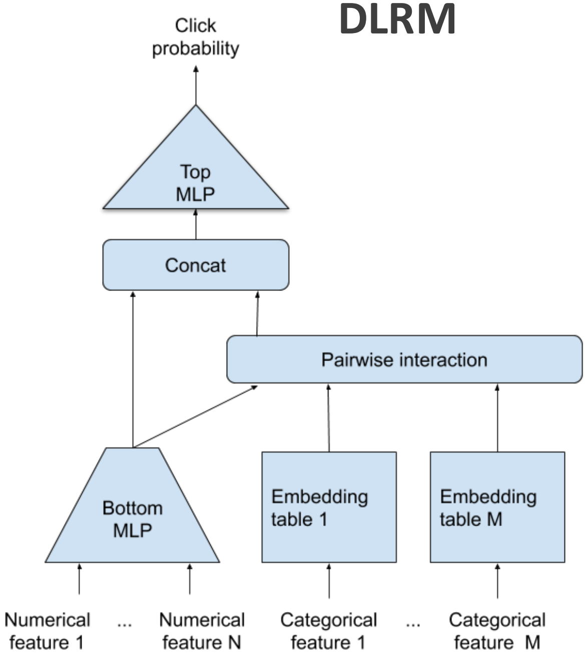
ResNet



Transformer

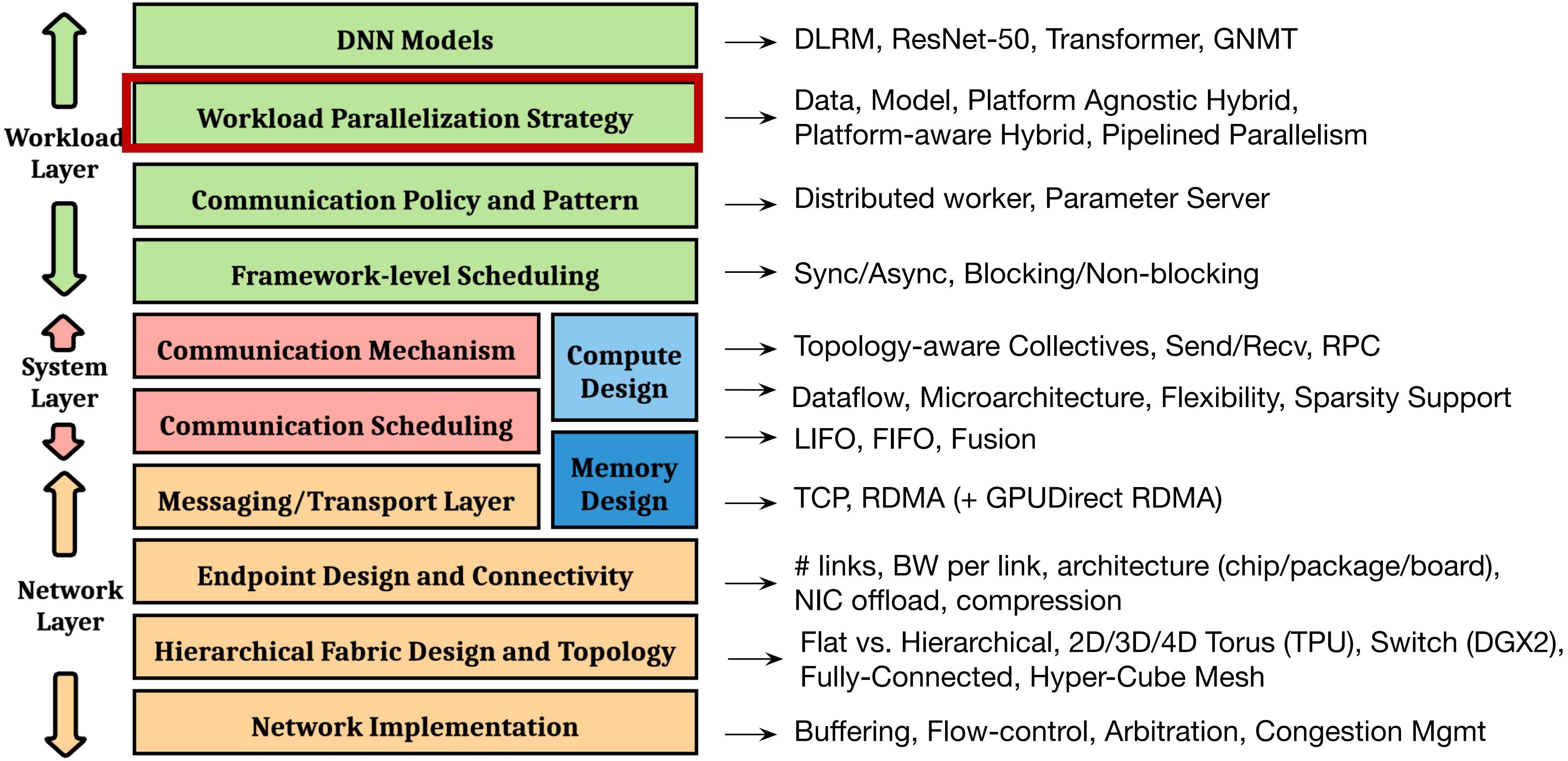


DLRM



Operator Types: CONV2D, Attention, Fully-Connected, …
Parameter sizes: Millions to Trillions

Distributed Training Stack



Workload Layer

- DNN Models → DLRM, ResNet-50, Transformer, GNMT
- Workload Parallelization Strategy → Data, Model, Platform Agnostic Hybrid, Platform-aware Hybrid, Pipelined Parallelism
- Communication Policy and Pattern → Distributed worker, Parameter Server
- Framework-level Scheduling → Sync/Async, Blocking/Non-blocking

System Layer

- Communication Mechanism → Topology-aware Collectives, Send/Recv, RPC
- Compute Design → Dataflow, Microarchitecture, Flexibility, Sparsity Support
- Communication Scheduling → LIFO, FIFO, Fusion
- Memory Design
- Messaging/Transport Layer → TCP, RDMA (+ GPUDirect RDMA)

Network Layer

- Endpoint Design and Connectivity → # links, BW per link, architecture (chip/package/board), NIC offload, compression
- Hierarchical Fabric Design and Topology → Flat vs. Hierarchical, 2D/3D/4D Torus (TPU), Switch (DGX2), Fully-Connected, Hyper-Cube Mesh
- Network Implementation → Buffering, Flow-control, Arbitration, Congestion Mgmt

Figure Courtesy: Srinivas Sridharan (NVIDIA)

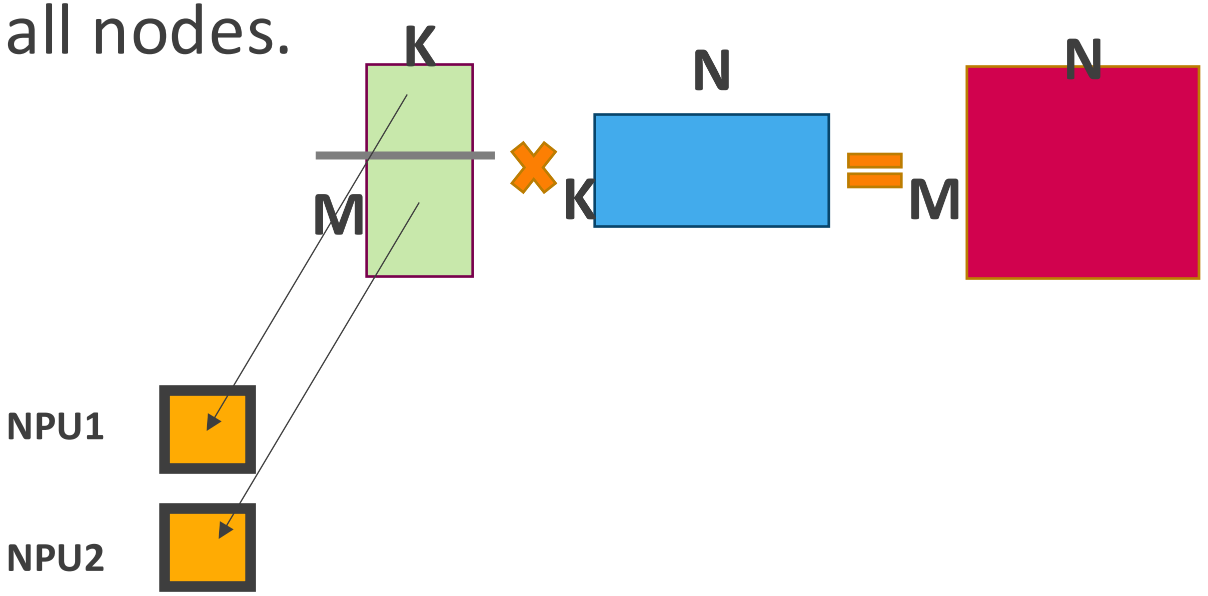
Parallelization Strategies

- The way compute tasks are distributed across different compute nodes. Multiple ways to split the tasks:
  - Split the Minibatch (**Data-Parallel**)
  - Split the Model
    - Across Tensors **(Tensor-Parallel)**
    - Across layers: (**Pipeline-Parallel**)
  - ….

- This also defines the communication pattern across different nodes.

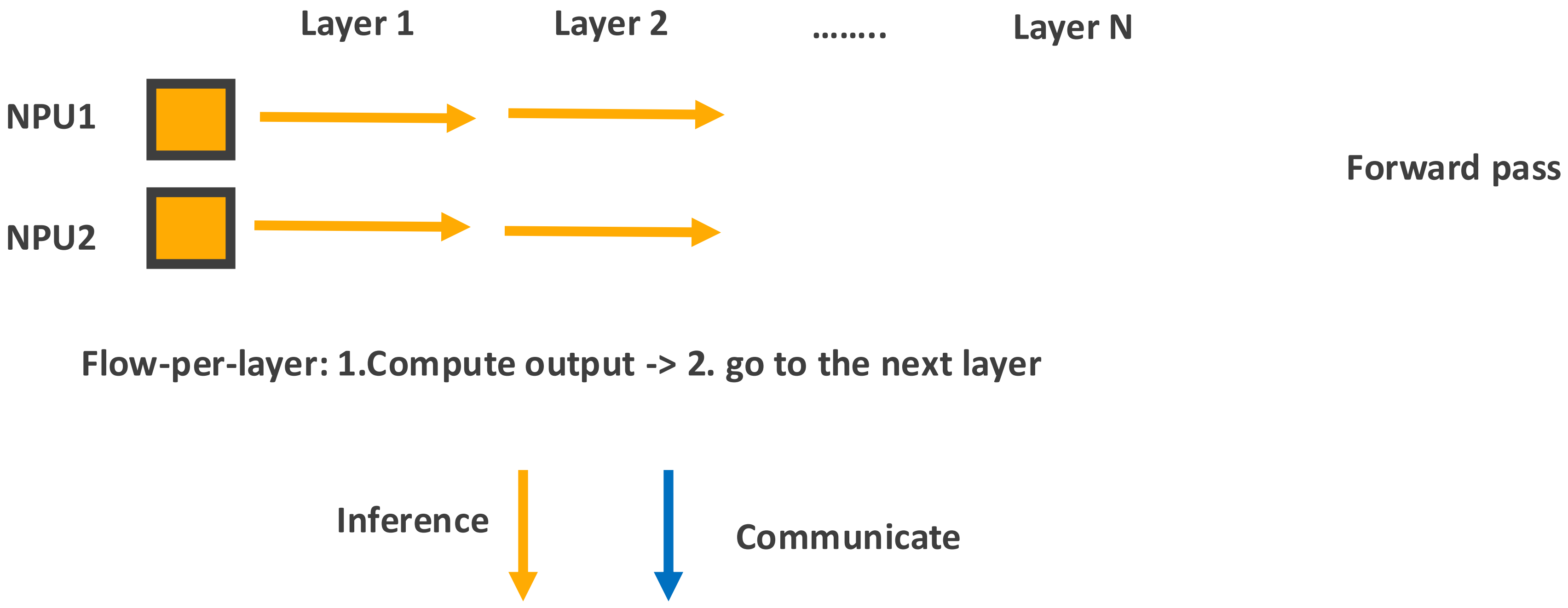
Parallelism: Data-Parallel

- Distribute Data across multiple nodes and replicate model (network) along all nodes.



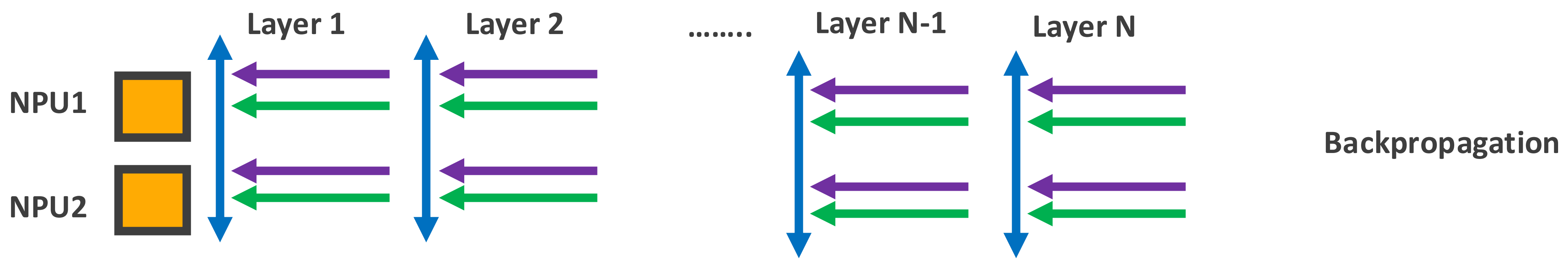
Parallelism: Data-Parallel

- Distribute Data across multiple nodes and replicate model (network) along all nodes.
- **No communication** during the forward pass.



Parallelism: Data-Parallel

- Distribute Data across multiple nodes and replicate model (network) along all nodes.
- **Communicate weight gradients** during the backpropagation pass.
  - *via non-blocking ”All Reduce” collective*
    - Blocking wait at end of backpropogation for collective before forward pass



Flow-per-layer: 1.Compute weight gradient-> 2.issue weight gradient comm -> 3.compute input gradient -> 4. go to previous layer

Inference compute

Input gradient compute

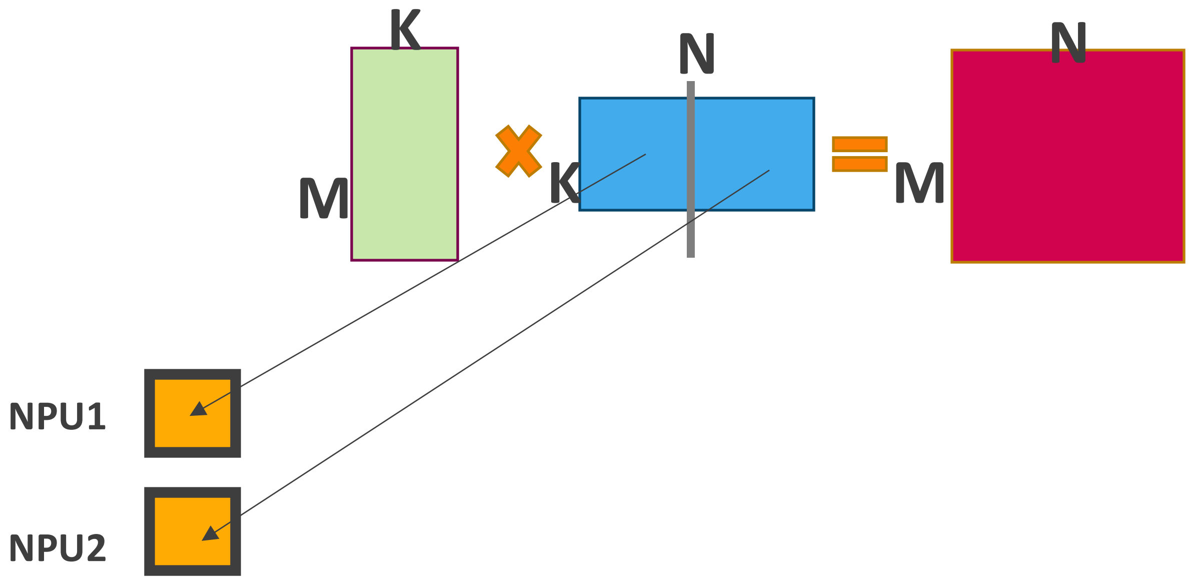
Weight gradient compute

Non-Blocking Communicate

Blocking Communicate

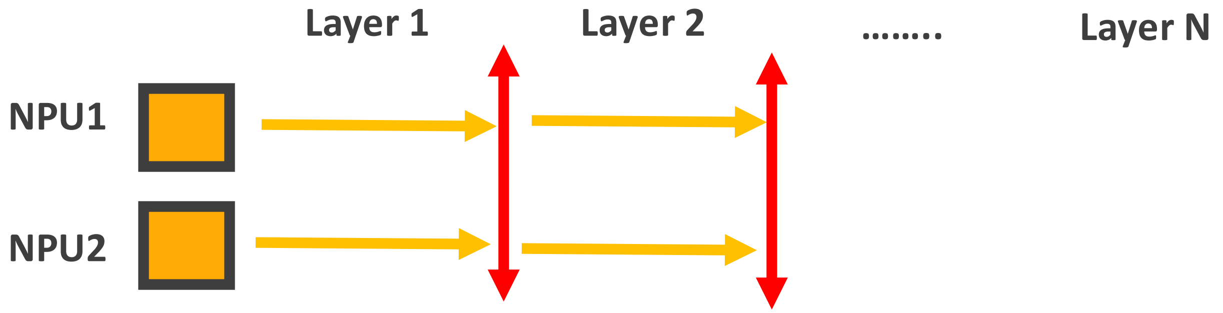
Parallelism: Tensor-Parallel

- Distribute Model across all nodes and replicate data along all nodes.



Parallelism: Tensor-Parallel

- Distribute Model across all nodes and replicate data along all nodes.
- **Communicate outputs** during the forward pass.



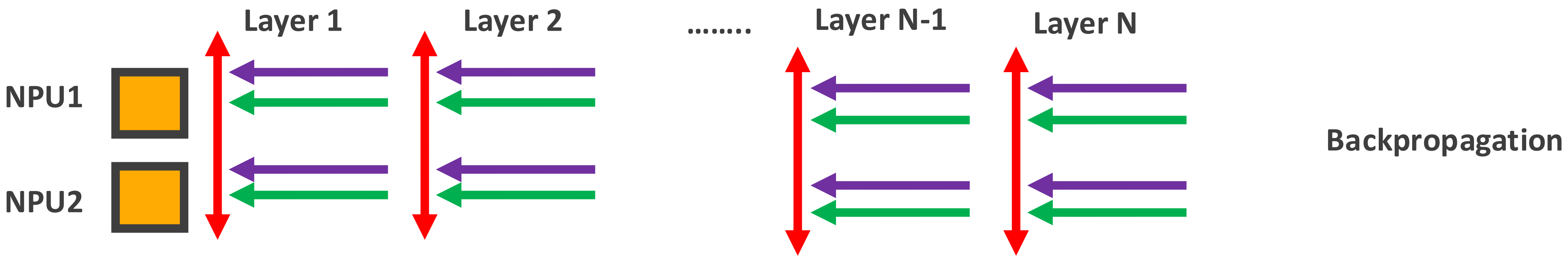
Forward pass

Flow-per-layer: 1.Compute output -> 2. issue output gradient comm -> 3.wait for gradient to be finished -> 4. go to the next layer

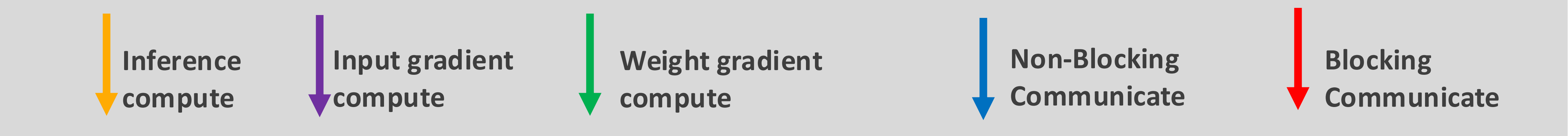
Inference compute | **Input gradient compute** | **Weight gradient compute** | **Non-Blocking Communicate** | **Blocking Communicate**

Parallelism: Tensor-Parallel

- Distribute Model across all nodes and replicate data along all nodes
- **Communicate input gradients** during the backpropagation pass.

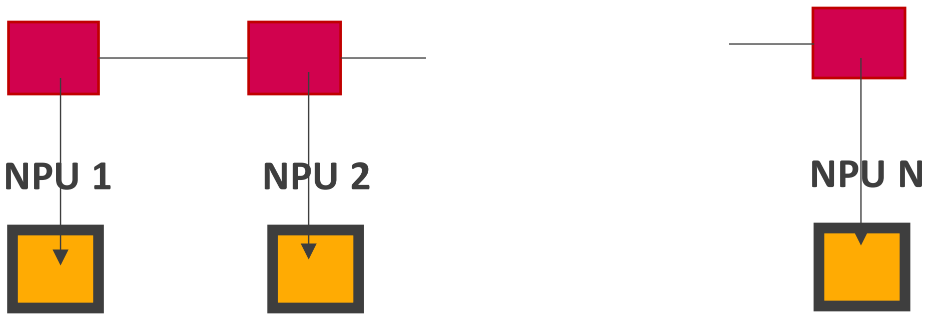


Flow-per-layer: 1.Compute input gradient-> 2.issue input gradient comm -> 3.compute weight gradient -> 4. wait for input gradient -> 5. go to previous layer



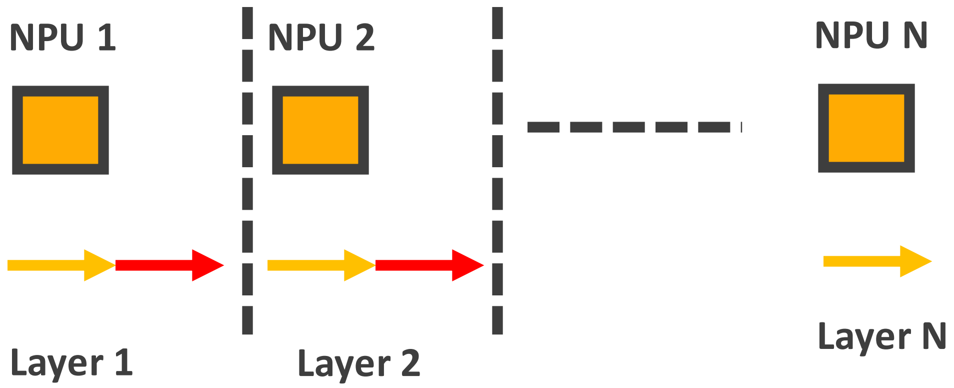
Parallelism: Pipeline-Parallel

- Distribute DNN layers across all nodes.



Parallelism: Pipeline-Parallel

- Distribute DNN layers across all nodes.
- **Communicate outputs** during the forward pass.



Inference compute

Input gradient compute

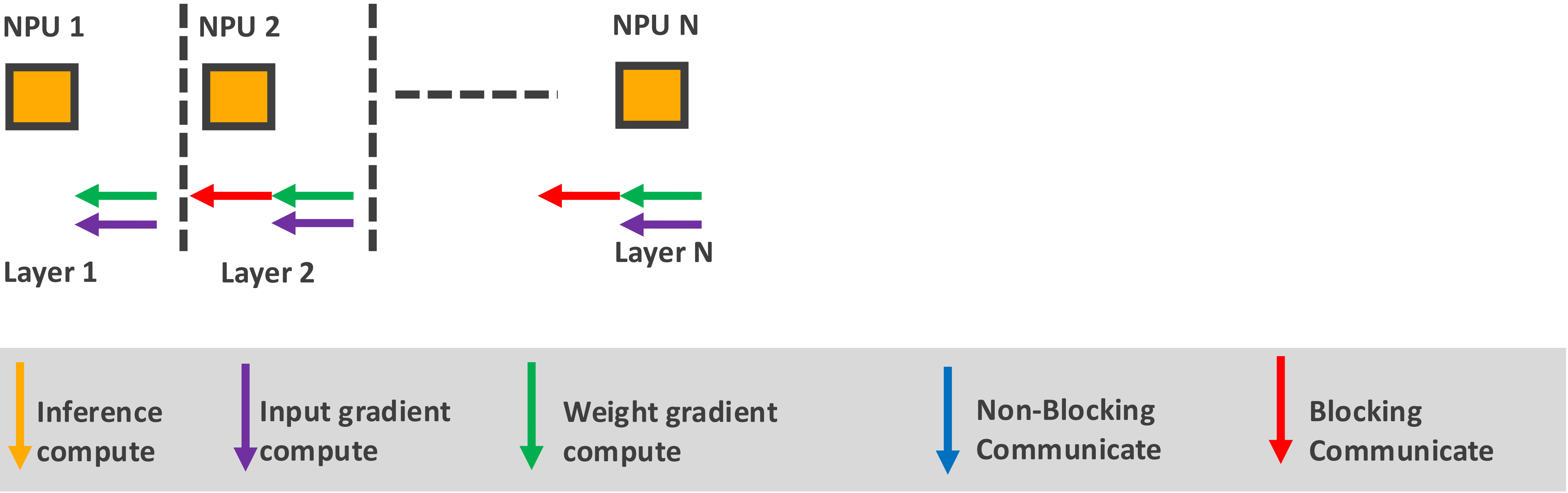
Weight gradient compute

Non-Blocking Communicate

Blocking Communicate

Parallelism: Pipeline-Parallel

- Distribute DNN layers across all nodes.
- **Communicate input gradients** during the backpropagation.



NPU 1

NPU 2

NPU N

Layer 1

Layer 2

Layer N

Inference compute

Input gradient compute

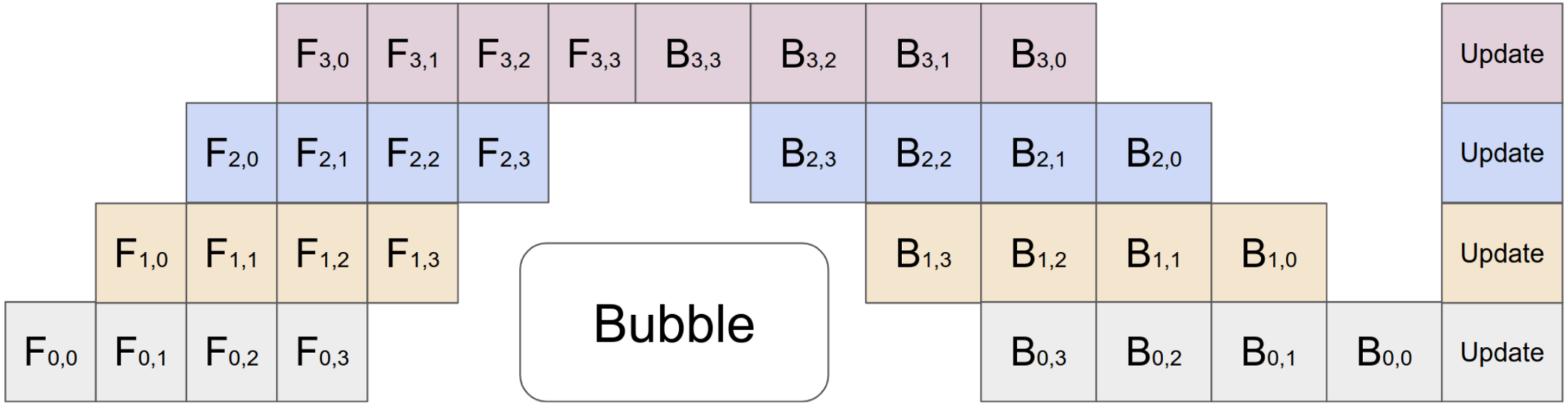
Weight gradient compute

Non-Blocking Communicate

Blocking Communicate

Parallelism: Pipeline-Parallel

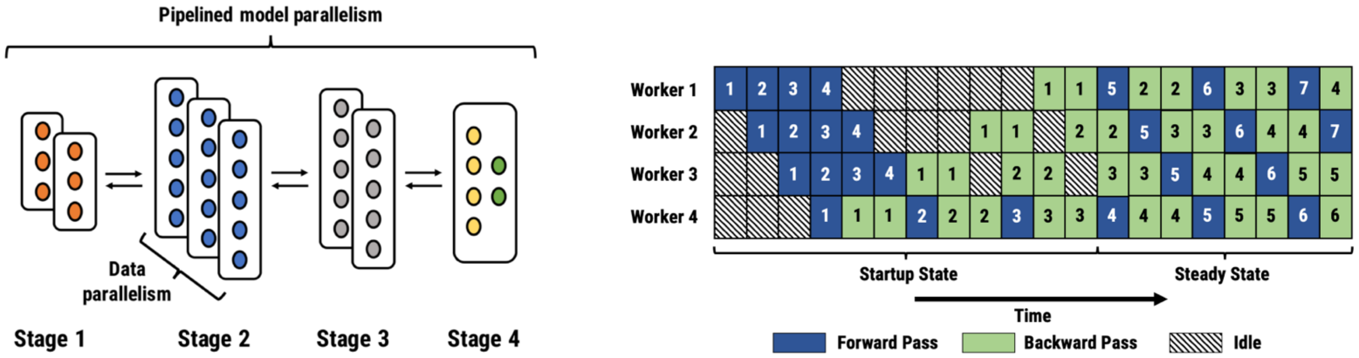
- Decompose minibatch into microbatches and propagate them to the pipeline in-order to enhance utilization
  - Challenge - bubbles


$F_{3,0}$ $F_{3,1}$ $F_{3,2}$ $F_{3,3}$ $B_{3,3}$ $B_{3,2}$ $B_{3,1}$ $B_{3,0}$ Update
$F_{2,0}$ $F_{2,1}$ $F_{2,2}$ $F_{2,3}$ $B_{2,3}$ $B_{2,2}$ $B_{2,1}$ $B_{2,0}$ Update
$F_{1,0}$ $F_{1,1}$ $F_{1,2}$ $F_{1,3}$ $B_{1,3}$ $B_{1,2}$ $B_{1,1}$ $B_{1,0}$ Update
$F_{0,0}$ $F_{0,1}$ $F_{0,2}$ $F_{0,3}$ $B_{0,3}$ $B_{0,2}$ $B_{0,1}$ $B_{0,0}$ Update
Bubble

$F_{m,n}$: forward-pass corresponding to micro-batch #n at device #m.

$B_{m,n}$: back-propagation corresponding to micro-batch #n at device #m.

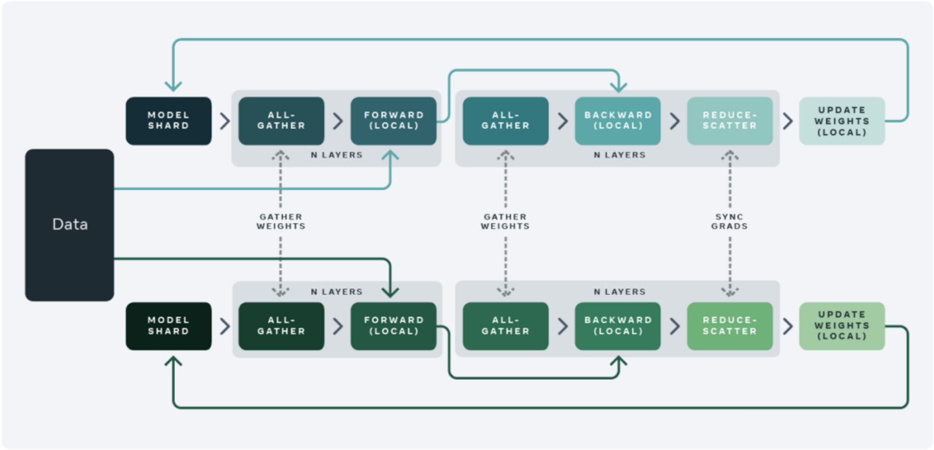
More sophisticated schemes



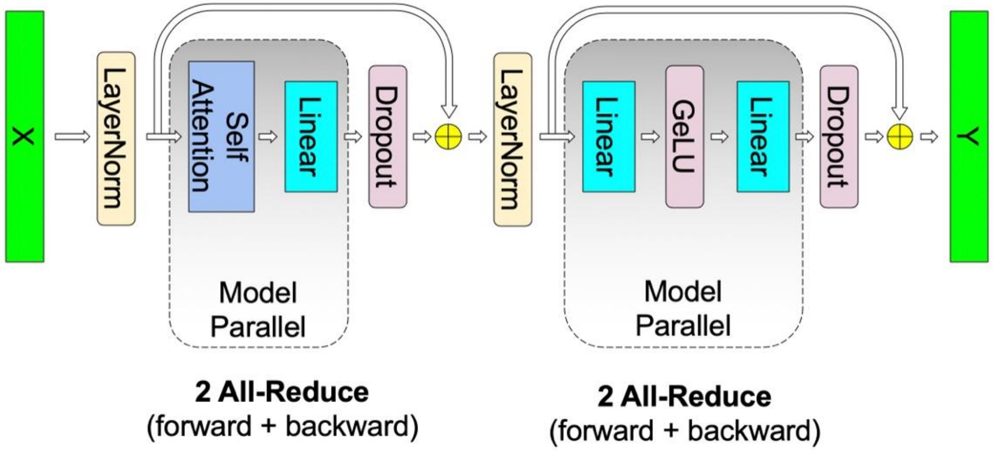
Worker 1	1	2	3	4	▨	▨	▨	▨	▨	▨	1	1	5	2	2	6	3	3	7	4
Worker 2	▨	1	2	3	4	▨	▨	▨	1	1	▨	2	2	5	3	3	6	4	4	7
Worker 3	▨	▨	1	2	3	4	1	1	▨	2	2	▨	3	3	5	4	4	6	5	5
Worker 4	▨	▨	▨	1	1	1	2	2	2	3	3	3	4	4	4	5	5	5	6	6

PipeDream (Microsoft)

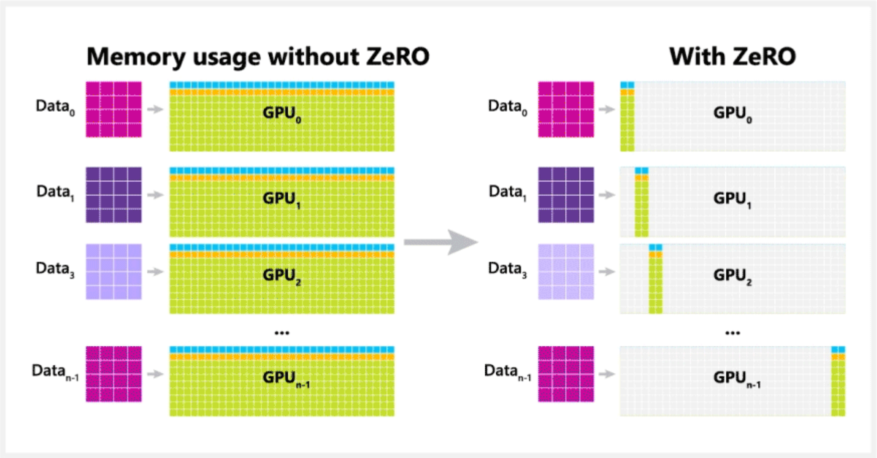
Fully sharded data parallel training



FSDP (Meta)

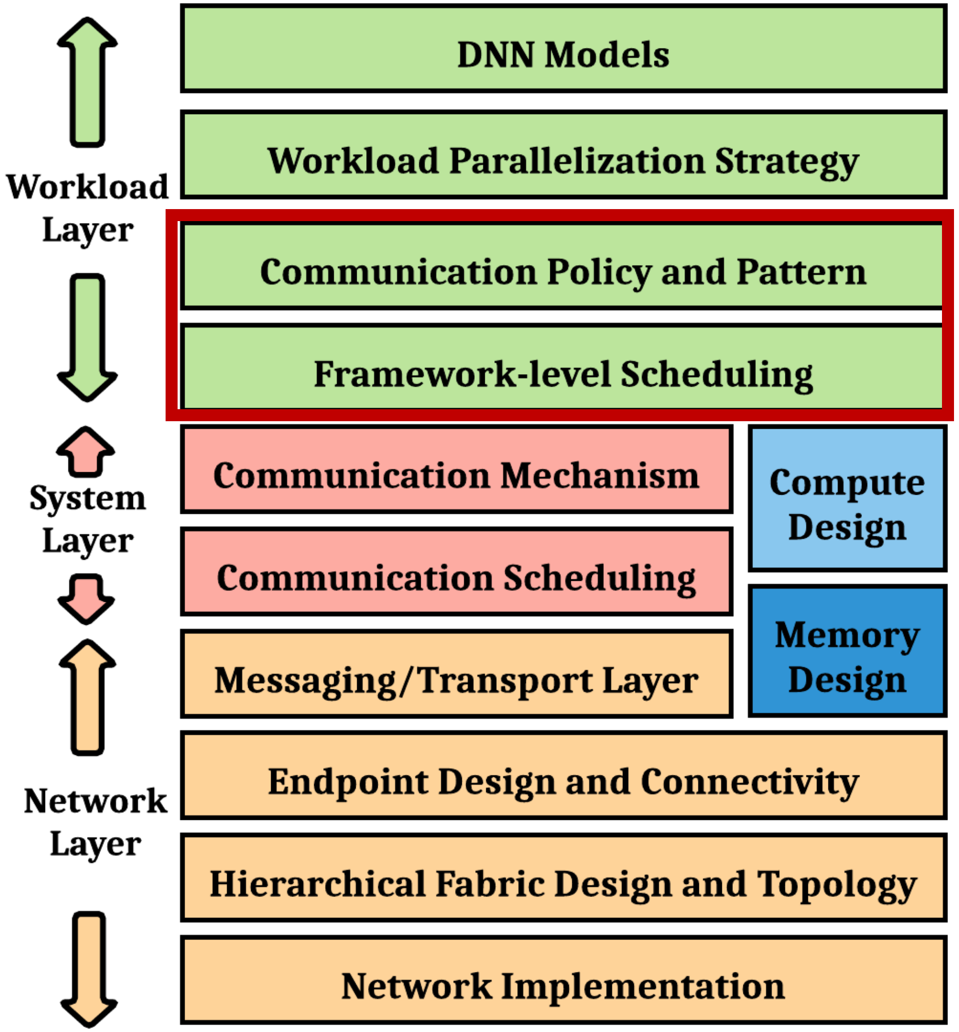


MegatronLM (NVIDIA)



Zero++ (Microsoft)

Distributed Training Stack



Workload Layer

System Layer

Network Layer

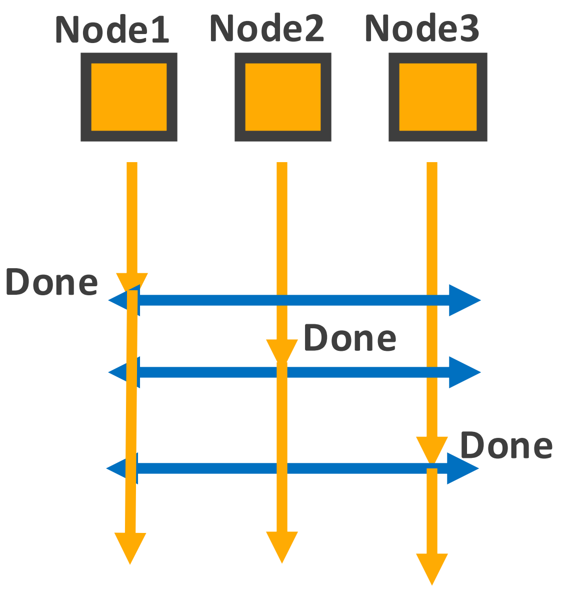
- DNN Models → DLRM, ResNet-50, Transformer, GNMT
- Workload Parallelization Strategy → Data, Model, Platform Agnostic Hybrid, Platform-aware Hybrid, Pipelined Parallelism
- Communication Policy and Pattern → Distributed worker, Parameter Server
- Framework-level Scheduling → Sync/Async, Blocking/Non-blocking
- Communication Mechanism → Topology-aware Collectives, Send/Recv, RPC
- Compute Design → Dataflow, Microarchitecture, Flexibility, Sparsity Support
- Communication Scheduling → LIFO, FIFO, Fusion
- Memory Design
- Messaging/Transport Layer → TCP, RDMA (+ GPUDirect RDMA)
- Endpoint Design and Connectivity → # links, BW per link, architecture (chip/package/board), NIC offload, compression
- Hierarchical Fabric Design and Topology → Flat vs. Hierarchical, 2D/3D/4D Torus (TPU), Switch (DGX2), Fully-Connected, Hyper-Cube Mesh
- Network Implementation → Buffering, Flow-control, Arbitration, Congestion Mgmt

Model Parameter Update Mechanisms

		Synchronization	
		Asynchronous	Synchronous
Communication Handling	Parameter-server	Centralized or Distributed	Centralized or Decentralized
	Collective-based	N/A	Distributed

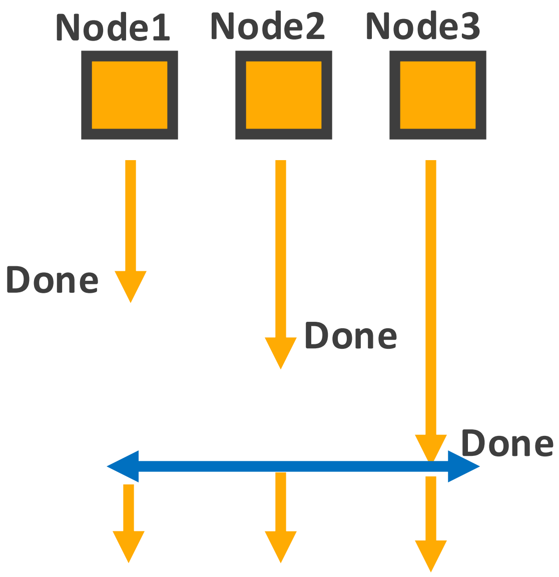
Synchronization: Sync. vs. Async. Training

- Defines when nodes should exchange data
  - Affects convergence time



Asynchronous

Synchronous training most popular



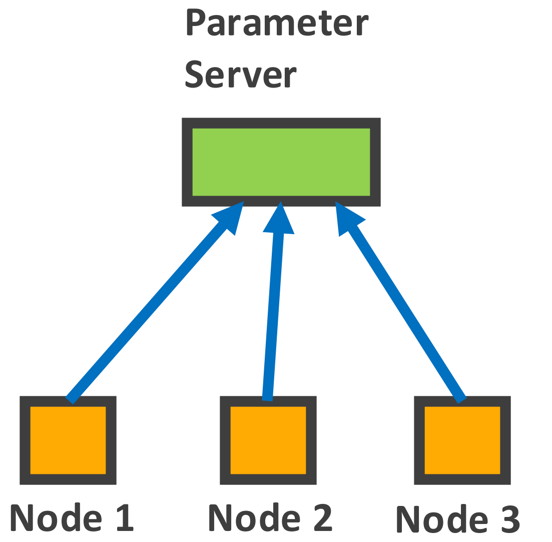
Synchronous

Compute ↓

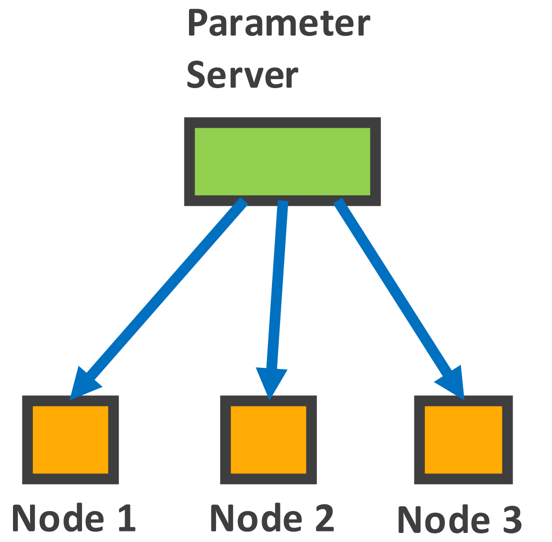
↔ Communicate

Communication Handling

- **Parameter Server**



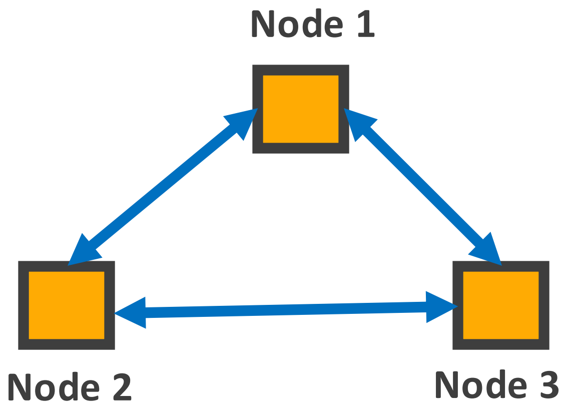
Step 1: Each node sends its model gradients to the parameter server to be reduced with other gradients and update the model

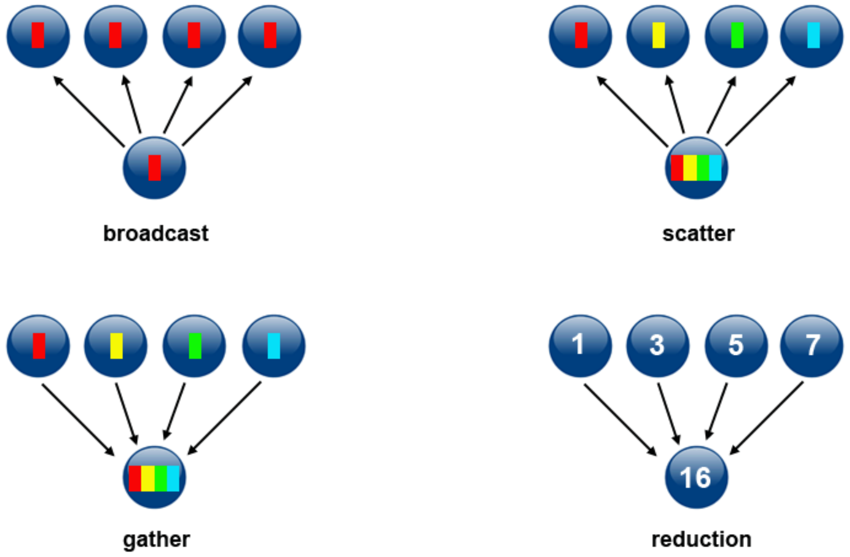


Step 2: The parameter server sends the updated model to the compute nodes to begin the new iteration.

Communication Handling

- **Collective-based:** Compute Nodes directly talk to each other to globally reduce their gradients and update the model through ***All-Reduce*** communication pattern.





"Collective Communication" (from MPI)

More details later

Exchanging Output Activations or Input Gradients:

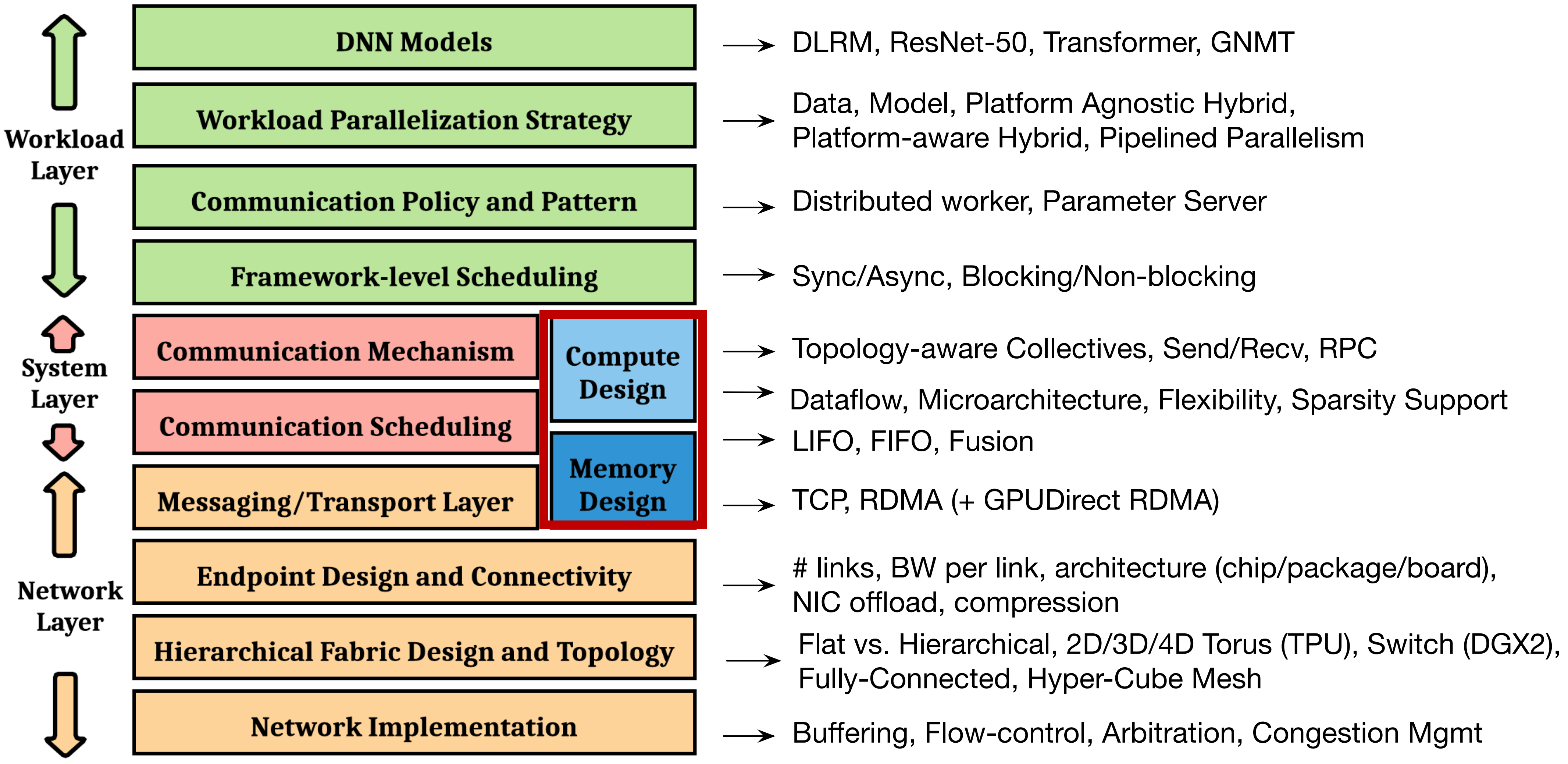
- It may be required depending on the **parallelization strategy** (discussed next)
- Handled either via **collective based patterns** or **direct Node-to-Node sends/recvs** (no parameter server is used).

When are collectives needed?

	Model (i.e. weight) Updates	Input Gradient Exchange	Output Activation Exchange
Param-server	N	Data-parallel: **N** Tensor-parallel: **Usually*** Pipeline-Parallel: **N**	Data-parallel: **N** Tensor-parallel: **Usually*** Pipeline-Parallel: **N**
Collective-based	Y (**All-Reduce**)	Data-parallel: **N** Tensor-parallel: **Usually*** Pipeline-Parallel: **N**	Data-parallel: **N** Tensor-parallel: **Usually*** Pipeline-Parallel: **N**

*** All-reduce, All-gather, Reduce-scatter, All-to-All**

Distributed Training Stack



Workload Layer

System Layer

Network Layer

DNN Models → DLRM, ResNet-50, Transformer, GNMT

Workload Parallelization Strategy → Data, Model, Platform Agnostic Hybrid, Platform-aware Hybrid, Pipelined Parallelism

Communication Policy and Pattern → Distributed worker, Parameter Server

Framework-level Scheduling → Sync/Async, Blocking/Non-blocking

Communication Mechanism → Topology-aware Collectives, Send/Recv, RPC

Compute Design → Dataflow, Microarchitecture, Flexibility, Sparsity Support

Communication Scheduling → LIFO, FIFO, Fusion

Memory Design

Messaging/Transport Layer → TCP, RDMA (+ GPUDirect RDMA)

Endpoint Design and Connectivity → # links, BW per link, architecture (chip/package/board), NIC offload, compression

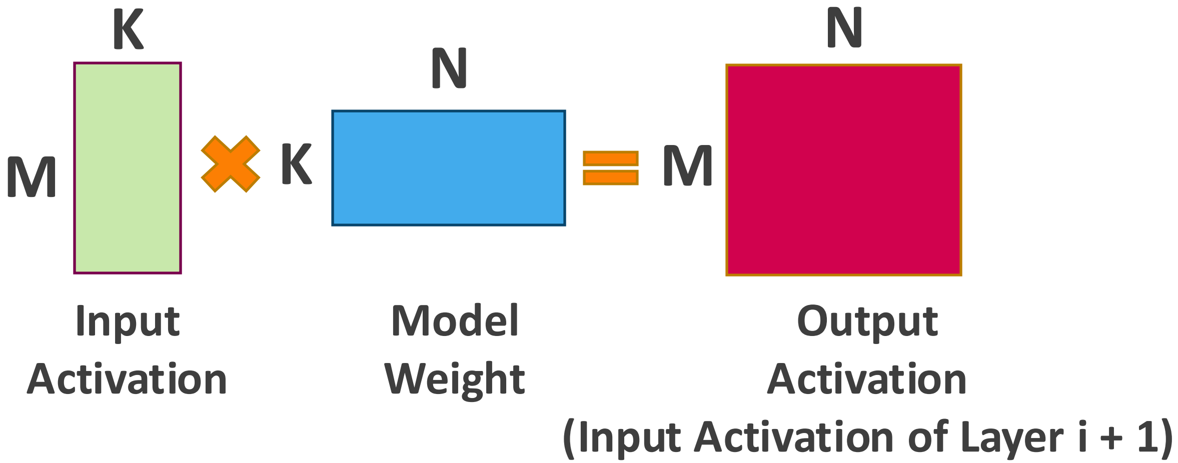
Hierarchical Fabric Design and Topology → Flat vs. Hierarchical, 2D/3D/4D Torus (TPU), Switch (DGX2), Fully-Connected, Hyper-Cube Mesh

Network Implementation → Buffering, Flow-control, Arbitration, Congestion Mgmt

Figure Courtesy: Srinivas Sridharan (NVIDIA)

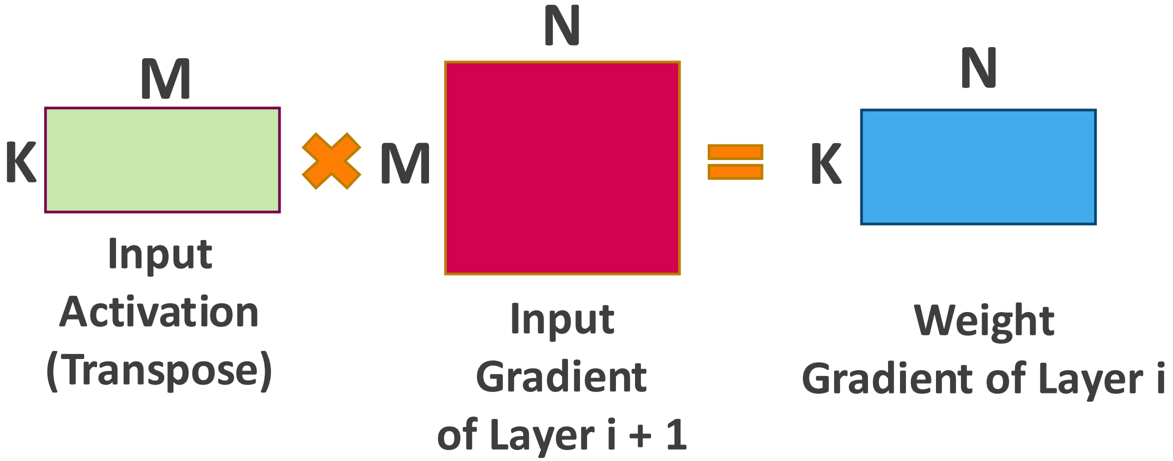
Training: Forward Pass

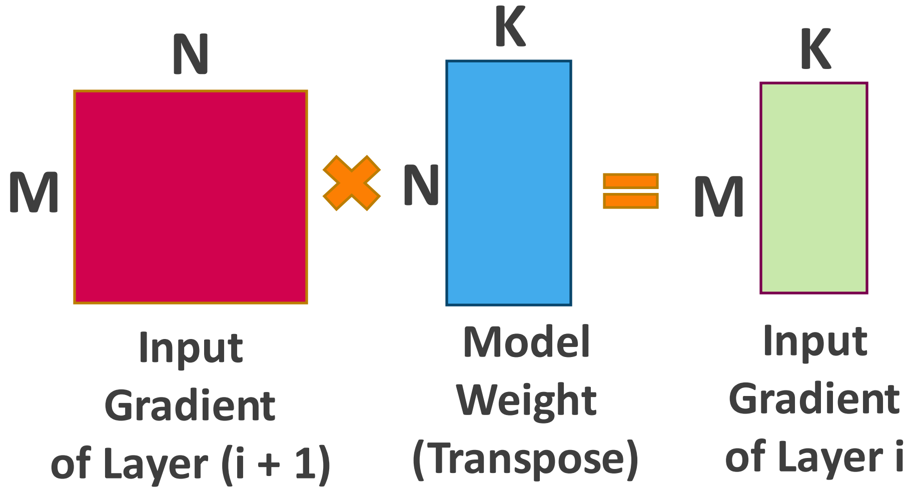
- In forward pass, each DNN layer computes **Output Activation**
  - From **Input Activation** (=output activation from last layer)
  - And **Model Weights**
  - Commonly through **GEMM** (Matrix Multiplication)



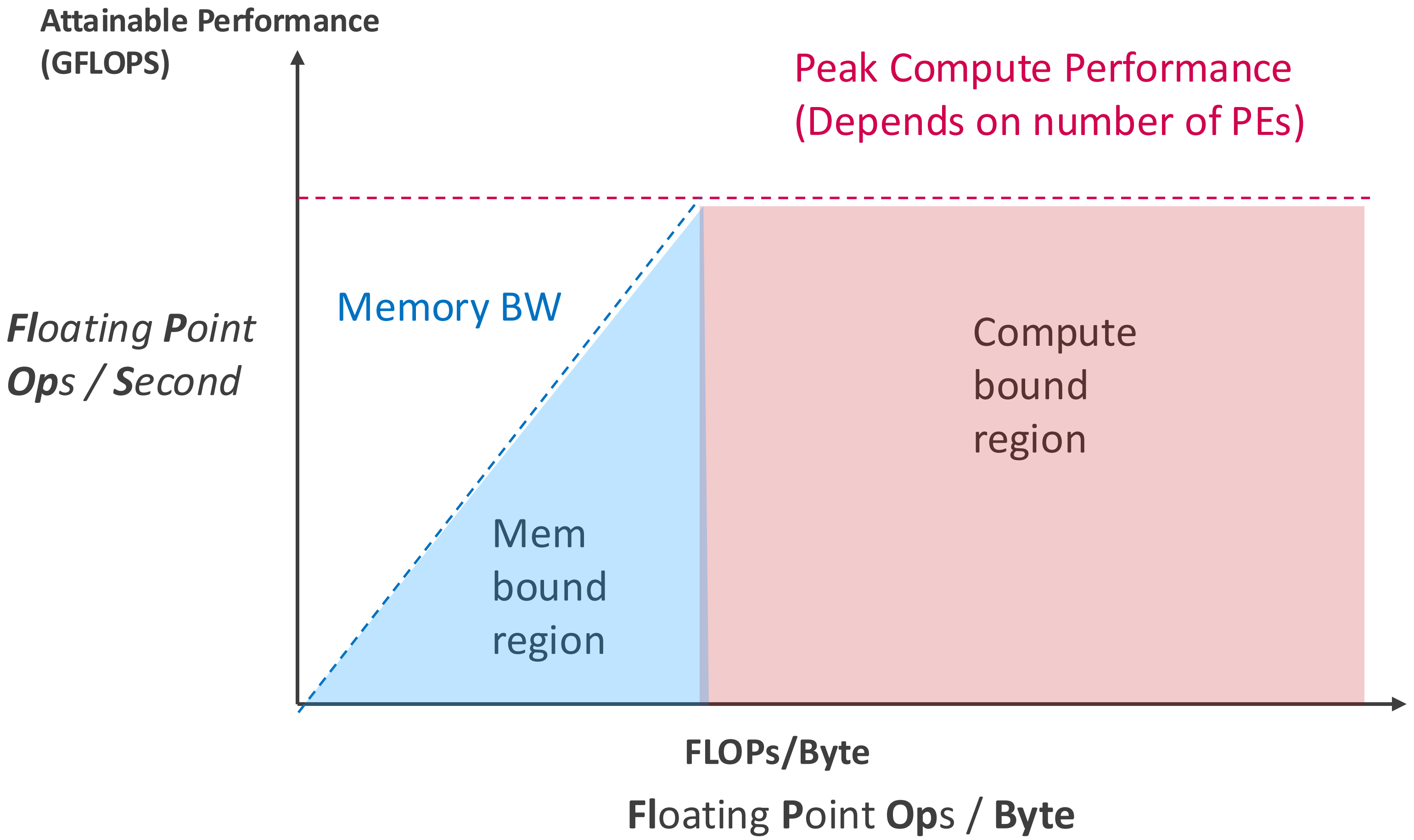
Training: Backward Pass

- In backward pass, each DNN layer computes:
  - **Weight Gradient**: to update model weights
  - **Input Gradient**: required to calculate weight gradient of layer (*i - 1*)
  - Commonly **GEMM** operations


M
K
Input Activation (Transpose)
N
M
Input Gradient of Layer i + 1
N
K
Weight Gradient of Layer i

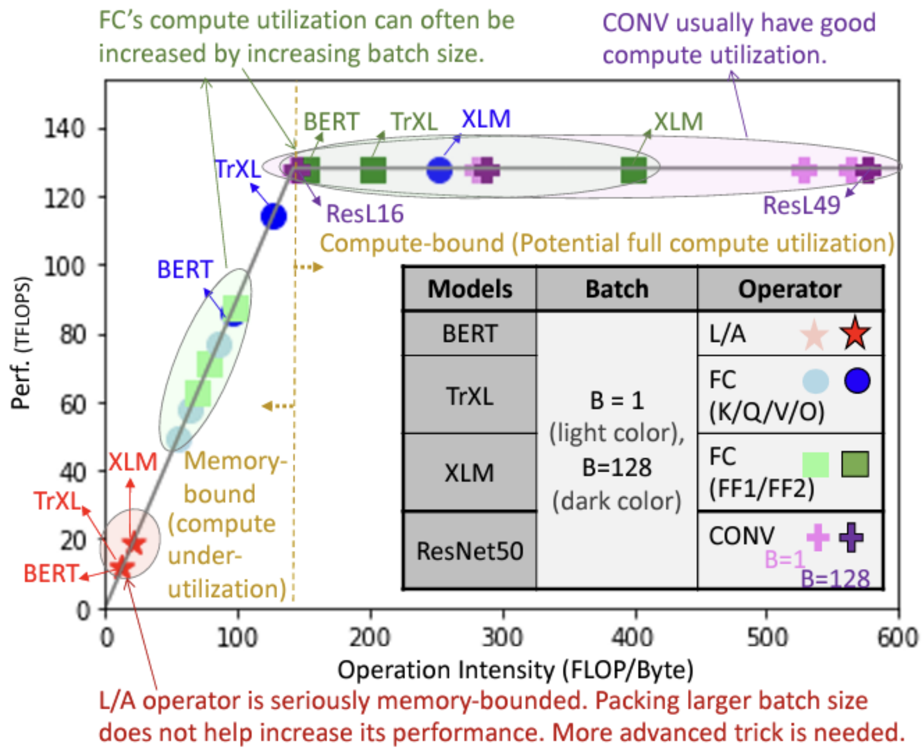


Compute Efficiency Depends on Data Reuse


Attainable Performance (GFLOPS)
Floating Point Ops / Second
Peak Compute Performance (Depends on number of PEs)
Memory BW
Mem bound region
Compute bound region
FLOPs/Byte
Floating Point Ops / Byte

Compute Bound => Throughput bound by number of compute units

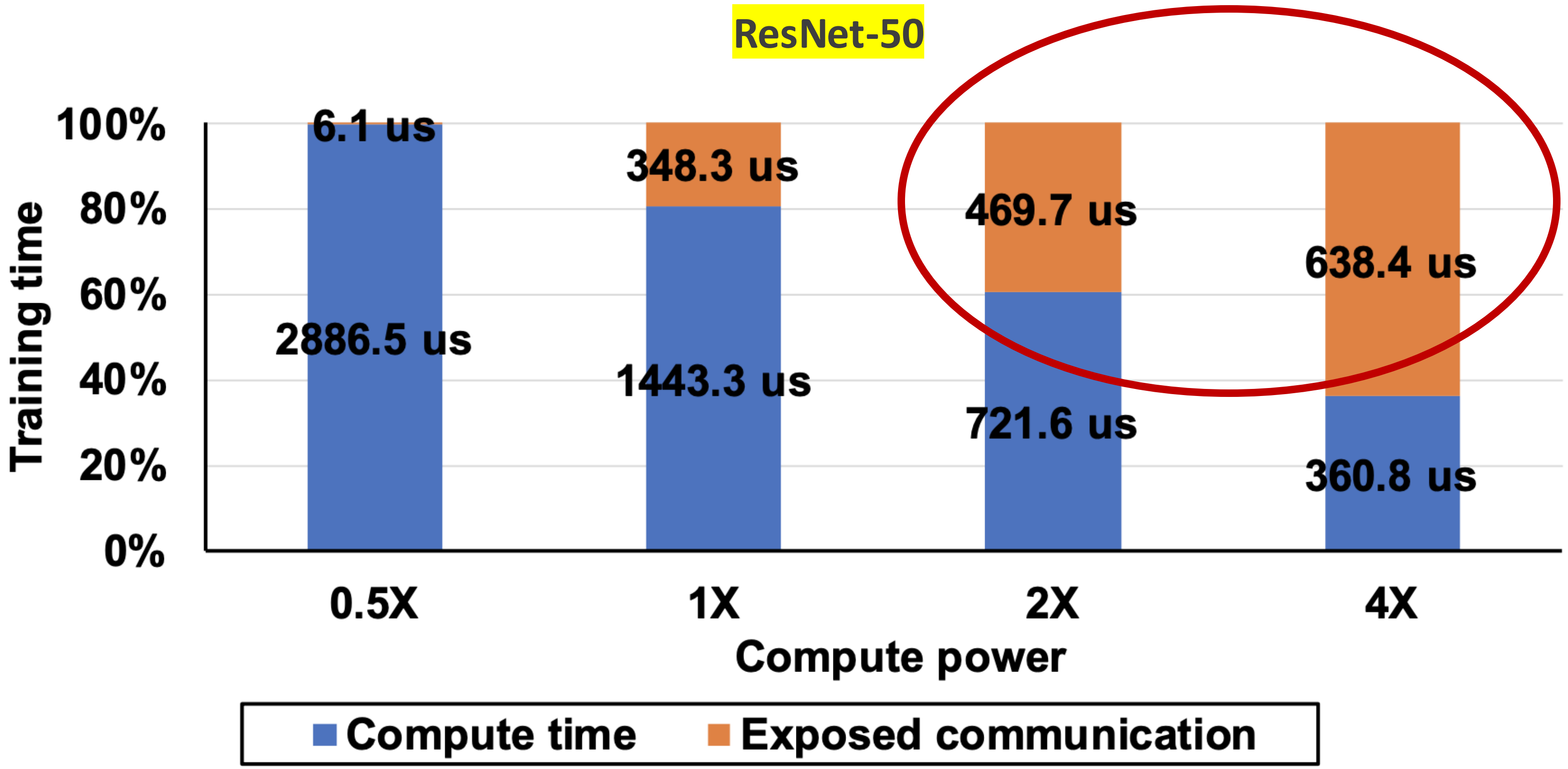
Memory Bound => Throughput bound by Memory BW



Models	Batch	Operator
BERT	B = 1 (light color), B=128 (dark color)	L/A ☆ ★
TrXL		FC (K/Q/V/O) ○ ●
XLM		FC (FF1/FF2) □ ■
ResNet50		CONV ✚ ✚ B=1 B=128

Transformer models are heavily memory bound
(Source: Kao et al, FLAT: An Optimized Dataflow for Mitigating Attention Bottlenecks, ASPLOS 2022)

Effect of Enhanced Compute Efficiency on Communication

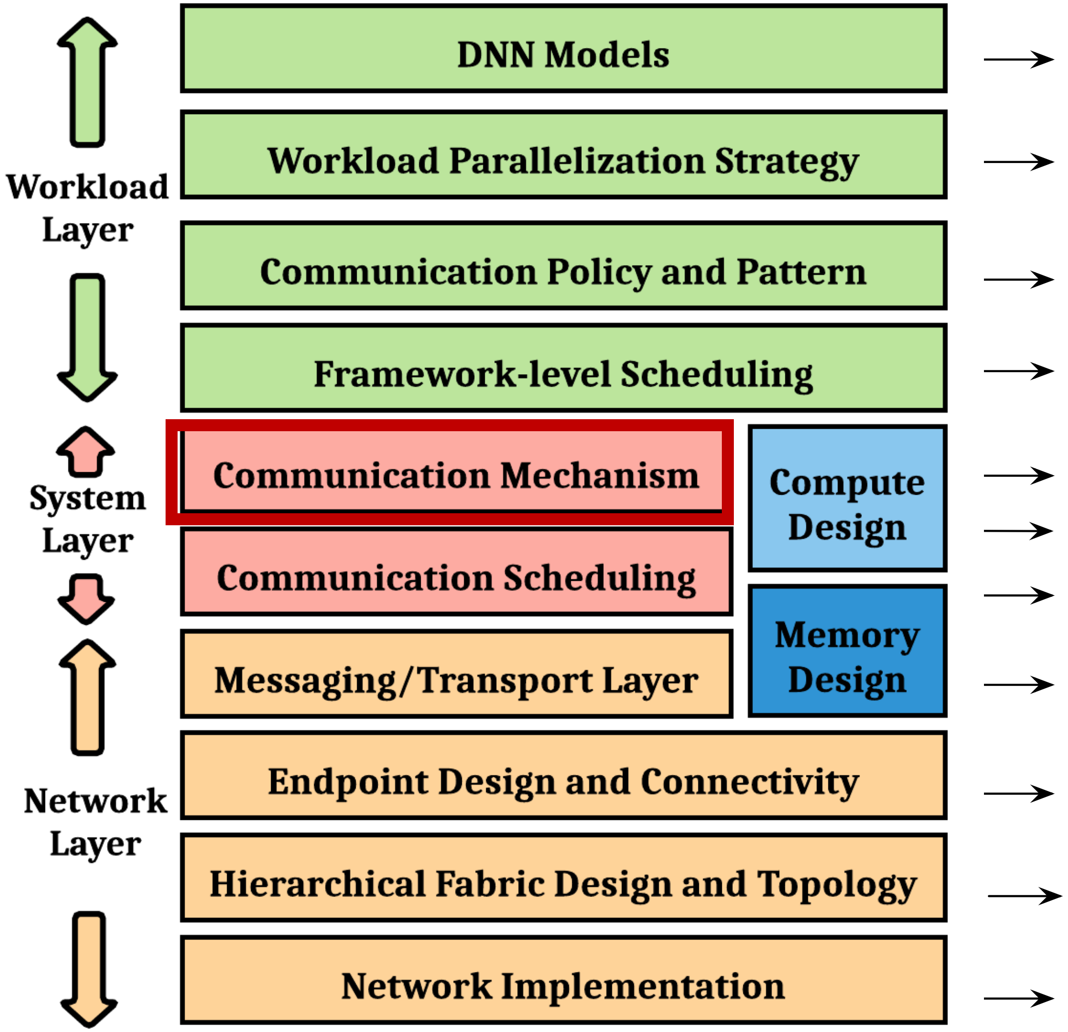


3D torus with total of 32 NPUs (2X4X4)

Compute Capability

*S. Rashidi et al.,"**ASTRA-SIM: Enabling SW/HW Co-Design Exploration for Distributed DL Training Platforms**", ISPASS 2020*

Distributed Training Stack



Workload Layer

System Layer

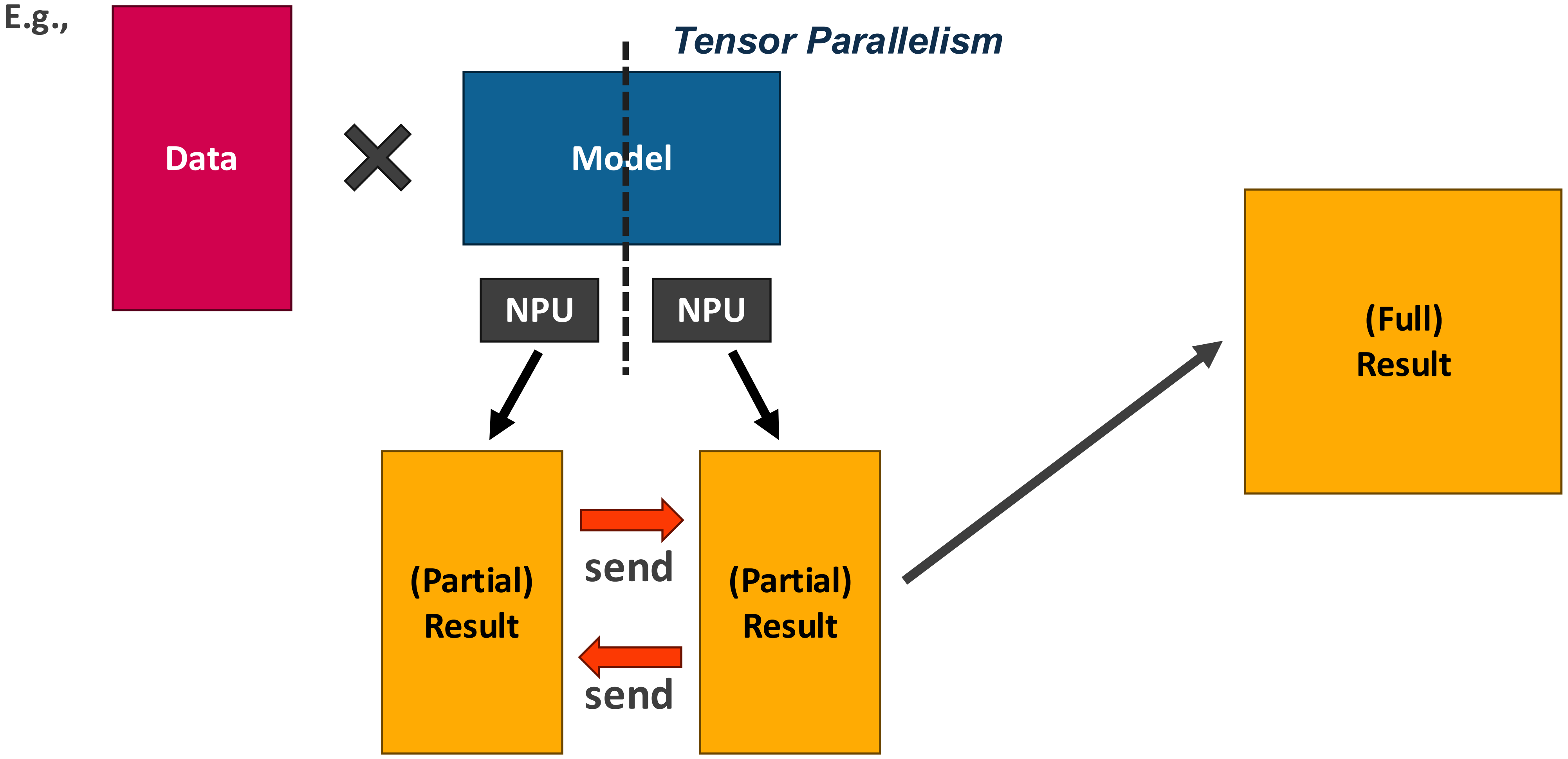
Network Layer

- DNN Models → DLRM, ResNet-50, Transformer, GNMT
- Workload Parallelization Strategy → Data, Model, Platform Agnostic Hybrid, Platform-aware Hybrid, Pipelined Parallelism
- Communication Policy and Pattern → Distributed worker, Parameter Server
- Framework-level Scheduling → Sync/Async, Blocking/Non-blocking
- Communication Mechanism → Topology-aware Collectives, Send/Recv, RPC
- Compute Design → Dataflow, Microarchitecture, Flexibility, Sparsity Support
- Communication Scheduling → LIFO, FIFO, Fusion
- Memory Design
- Messaging/Transport Layer → TCP, RDMA (+ GPUDirect RDMA)
- Endpoint Design and Connectivity → # links, BW per link, architecture (chip/package/board), NIC offload, compression
- Hierarchical Fabric Design and Topology → Flat vs. Hierarchical, 2D/3D/4D Torus (TPU), Switch (DGX2), Fully-Connected, Hyper-Cube Mesh
- Network Implementation → Buffering, Flow-control, Arbitration, Congestion Mgmt

Figure Courtesy: Srinivas Sridharan (NVIDIA)

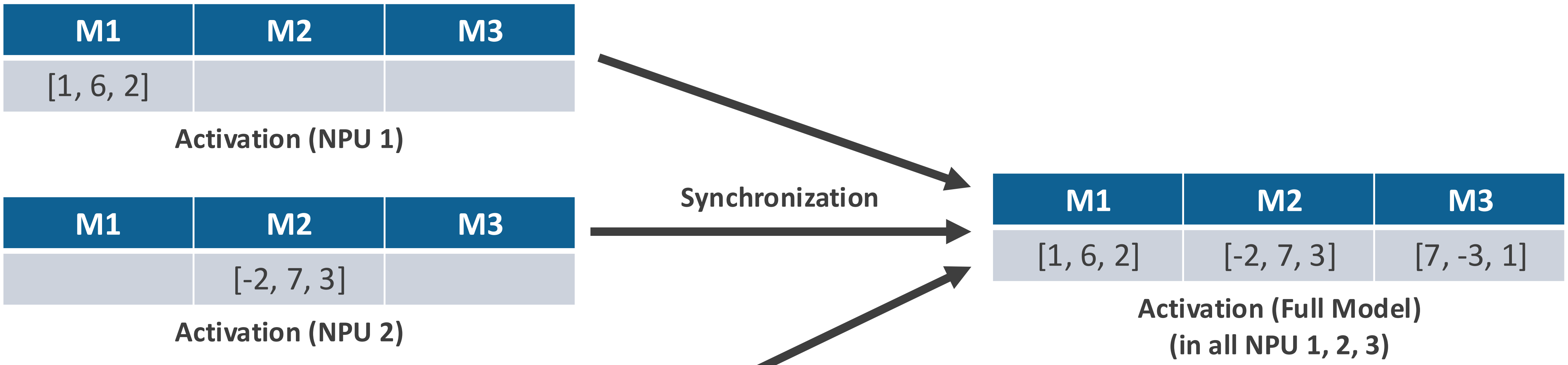
Communication in Distributed ML

- **NPUs should communicate** to synchronize outcomes



Example: Tensor Parallelism

- Each of the NPU produces **part of ML activation results**
  - NPUs then **synchronize** to recover the full activation result



M1	M2	M3
[1, 6, 2]		

Activation (NPU 1)

M1	M2	M3
	[-2, 7, 3]	

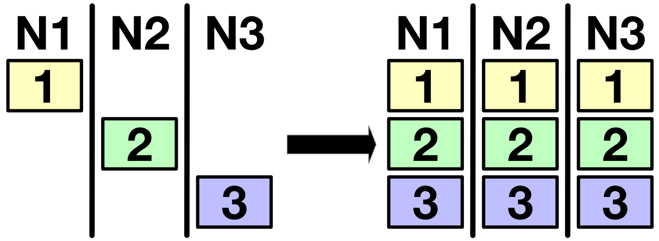
Activation (NPU 2)

M1	M2	M3
		[7, -3, 1]

Activation (NPU 3)

M1	M2	M3
[1, 6, 2]	[-2, 7, 3]	[7, -3, 1]

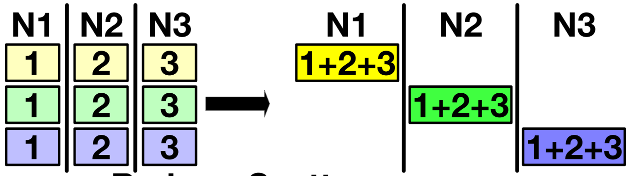
Activation (Full Model)
(in all NPU 1, 2, 3)



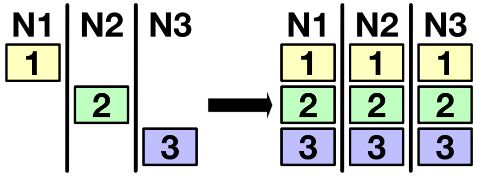
All-Gather

Collective Communication **"Patterns"**

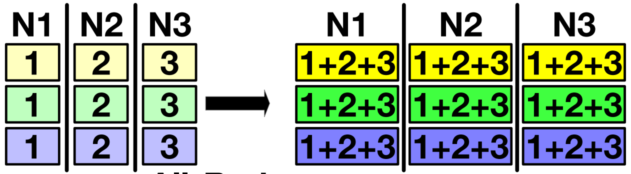
- Used for **communication/ synchronization** in distributed training/inference



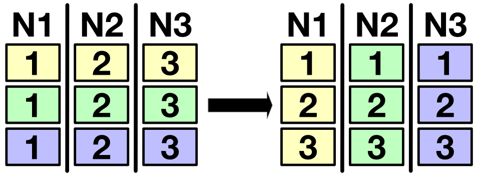
Reduce-Scatter



All-Gather



All-Reduce



All-to-All

- Specific pattern depends on parallelization strategy

Parallelization	Reduce-Scatter	All-Gather	All-Reduce
Data Parallel			✓
Tensor Parallel			✓
Hybrid Parallel	✓	✓	✓
FSDP	✓	✓	
ZeRO	✓	✓	

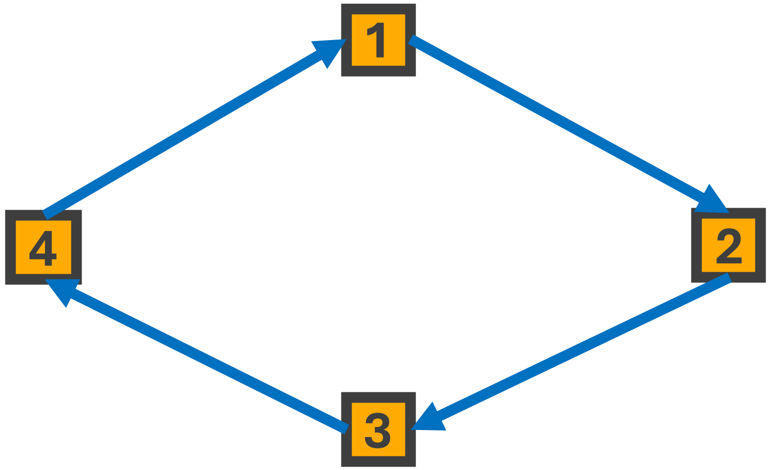
Collective Communication **“Algorithms”**

- Routing algorithm to *implement* collective patterns

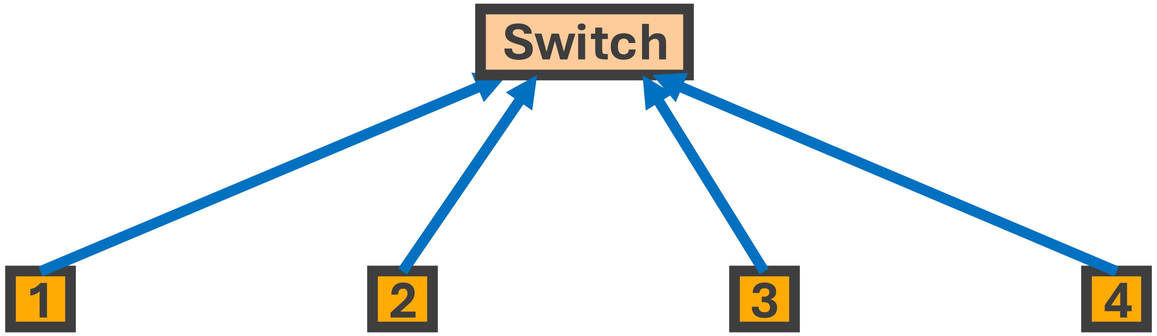
- Collective communication libraries (CCLs, e.g., NCCL, RCCL, oneCCL) use diverse collective algorithms to implement collective communication patterns
  - **Example All-Reduce Algorithms:** Ring, Direct, Halving-Doubling, Rabenseifner, Double Binary Tree, etc.

- Given a network topology, an **efficient algorithm** to run collective communication is called a **topology-aware collective algorithm**

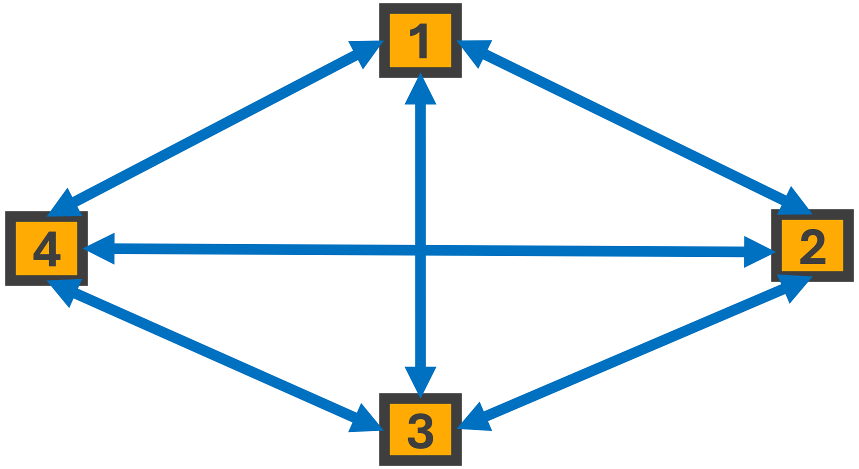
Example



Physical Topology: Ring



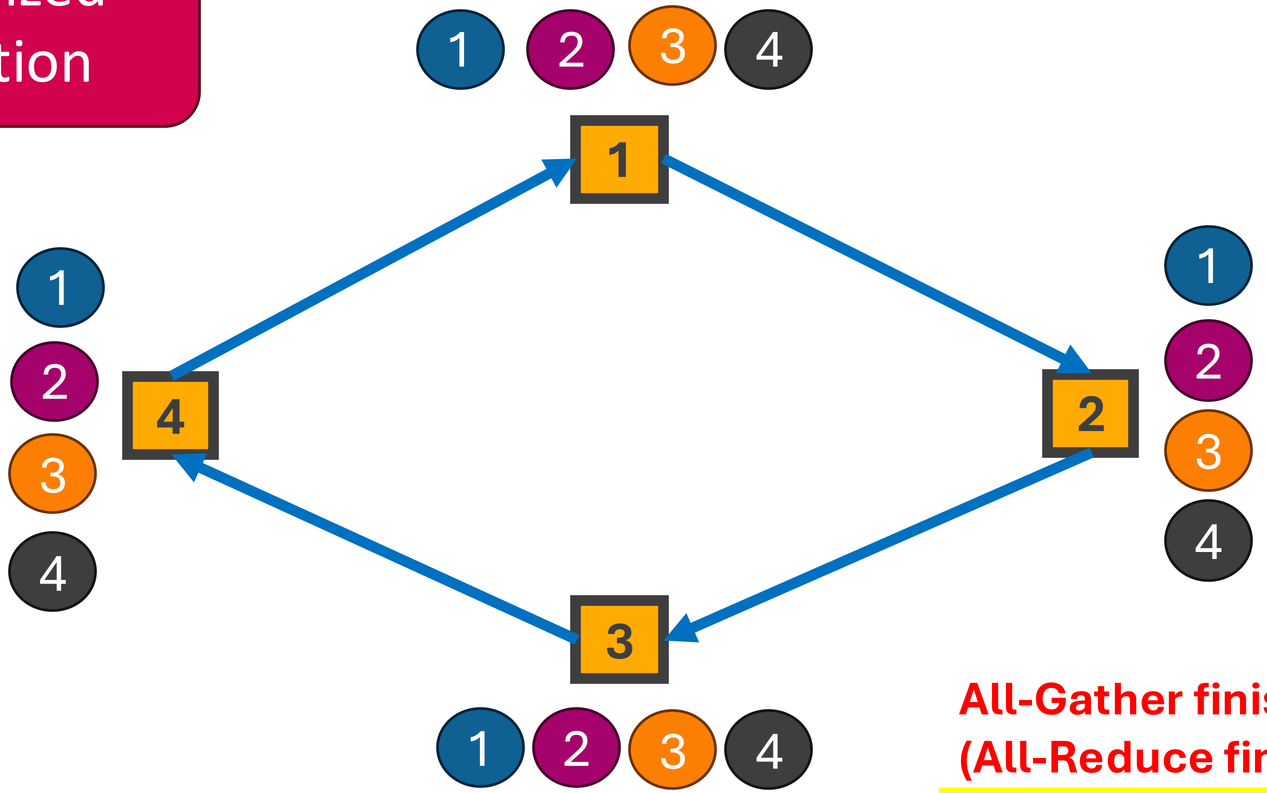
Physical Topology: Switch

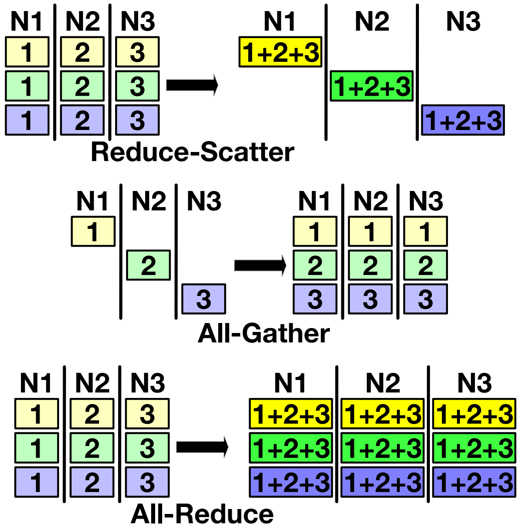


Physical Topology: Fully Connected

Collective Algorithm: Ring All-Reduce

- ✓ All links utilized
- ✓ No congestion





■ NPU

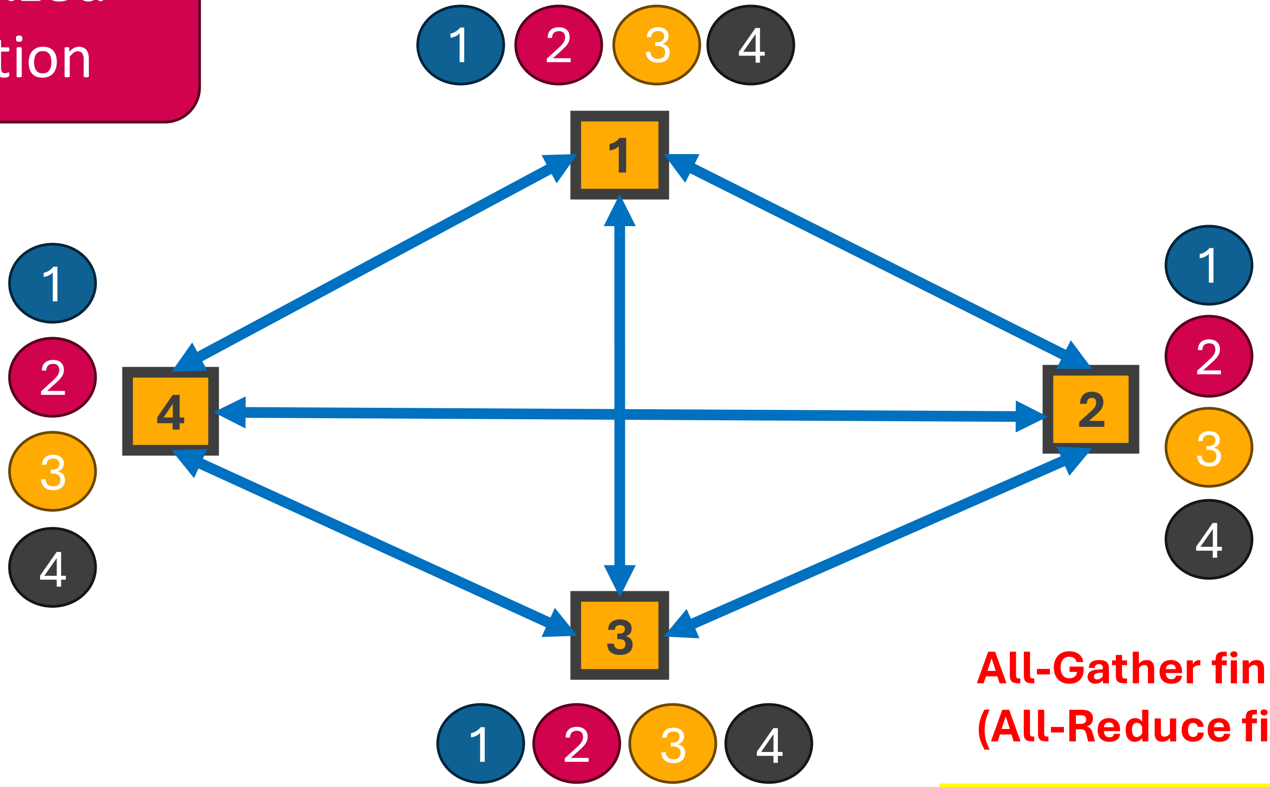
○ Chunk

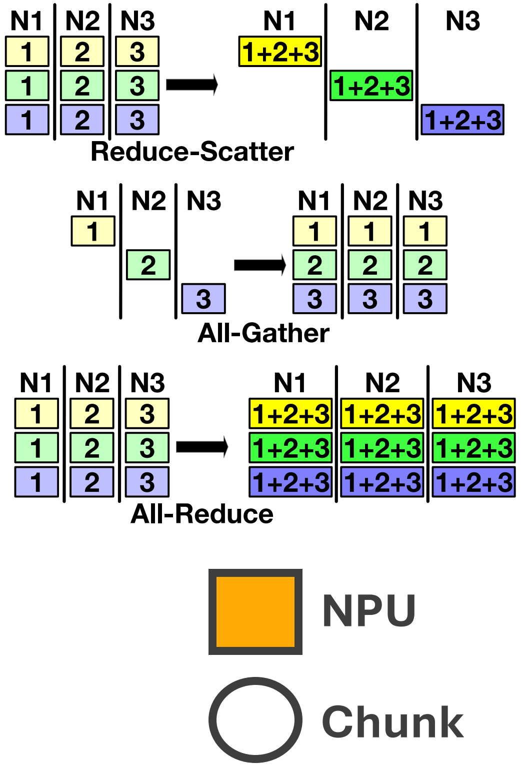
All-Gather finished
(All-Reduce finished)

Physical Topology: Ring

Collective Algorithm: Direct All-Reduce

✓ All links utilized
✓ No congestion



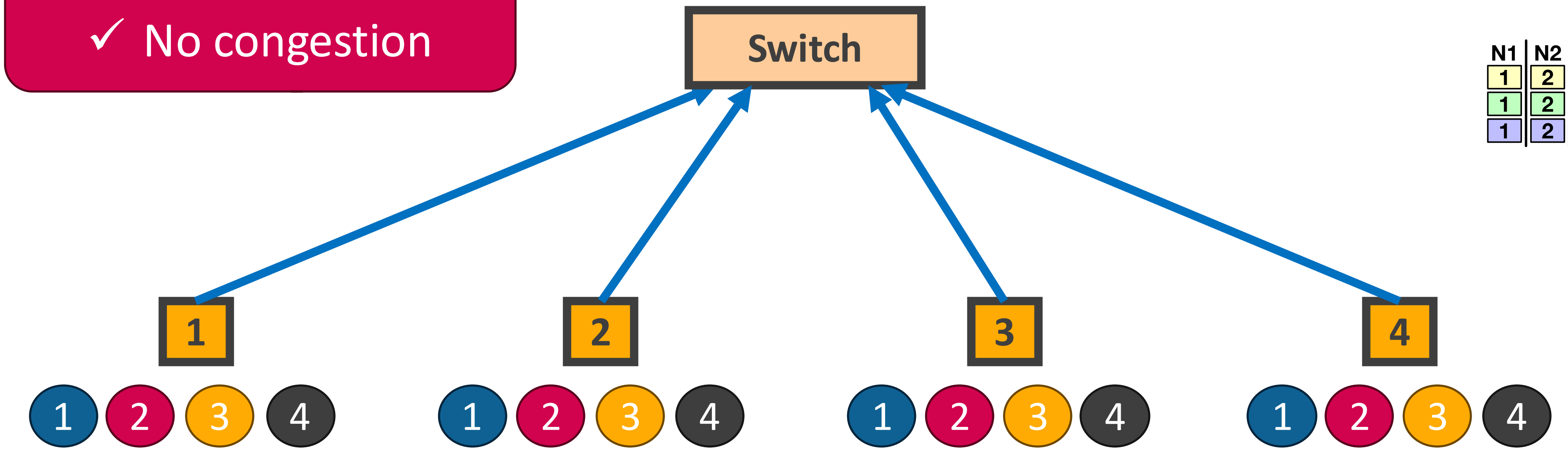


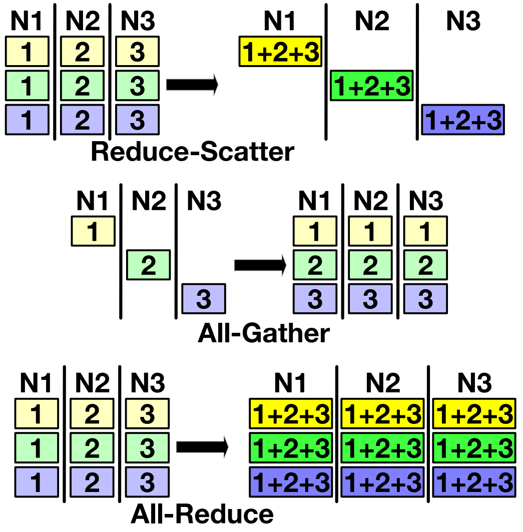
All-Gather finished
(All-Reduce finished)

Physical Topology: Fully-Connected

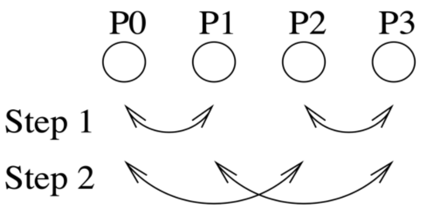
Collective Algorithm: Recursive Halving Doubling All-Reduce

✓ All links utilized
✓ No congestion





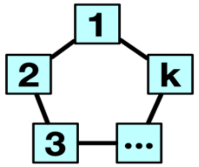
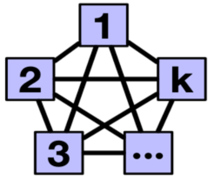
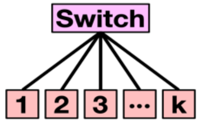
All-Gather finished
(All-Reduce finished)



Physical Topology: Switch

Summary: Basic Collective Algorithms

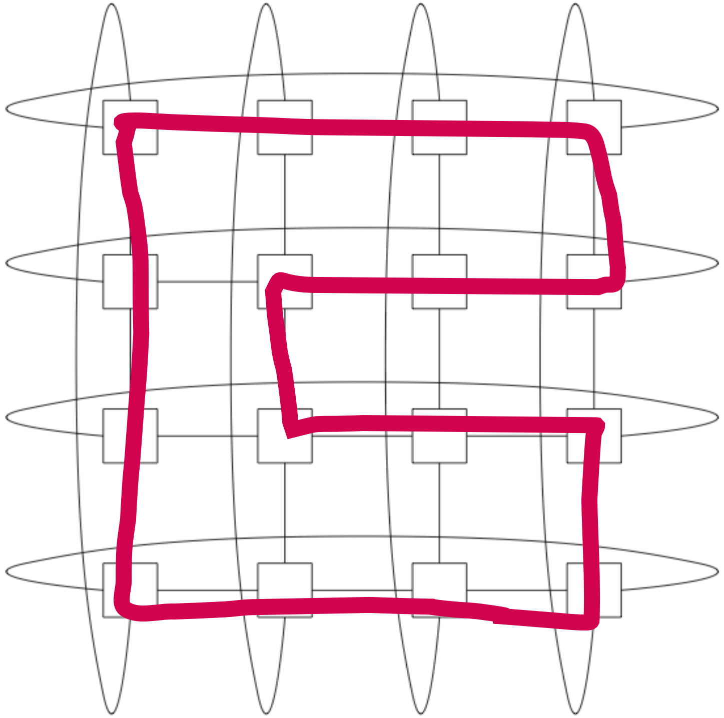
- **No network congestion** while running collective communication

Topology Building Block		Topology-aware Collective Algorithm
	Ring	Ring
	FullyConnected	Direct
	Switch	HalvingDoubling

What about other topologies?

Topology-aware Collective Algorithms

- Optimal collective algorithm heavily depends on network topology
  - Simple collective algorithms will not directly map



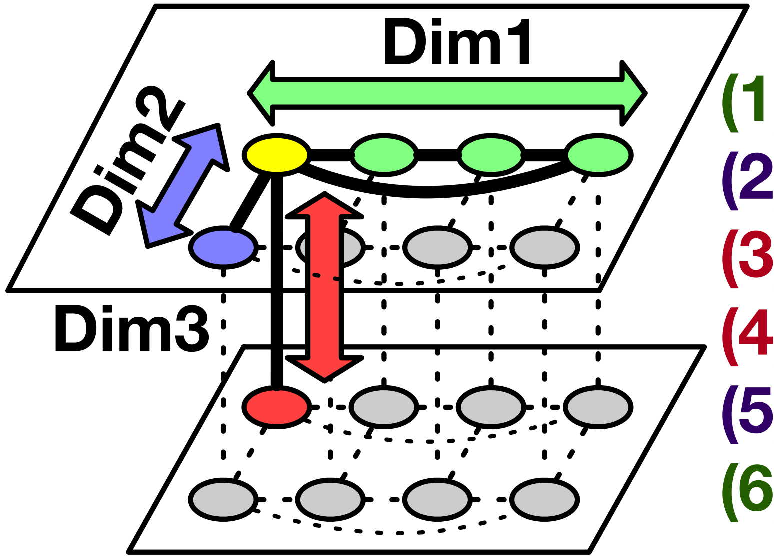
Ring Algorithm

Network Underutilization!!

Physical Topology: 2D Torus

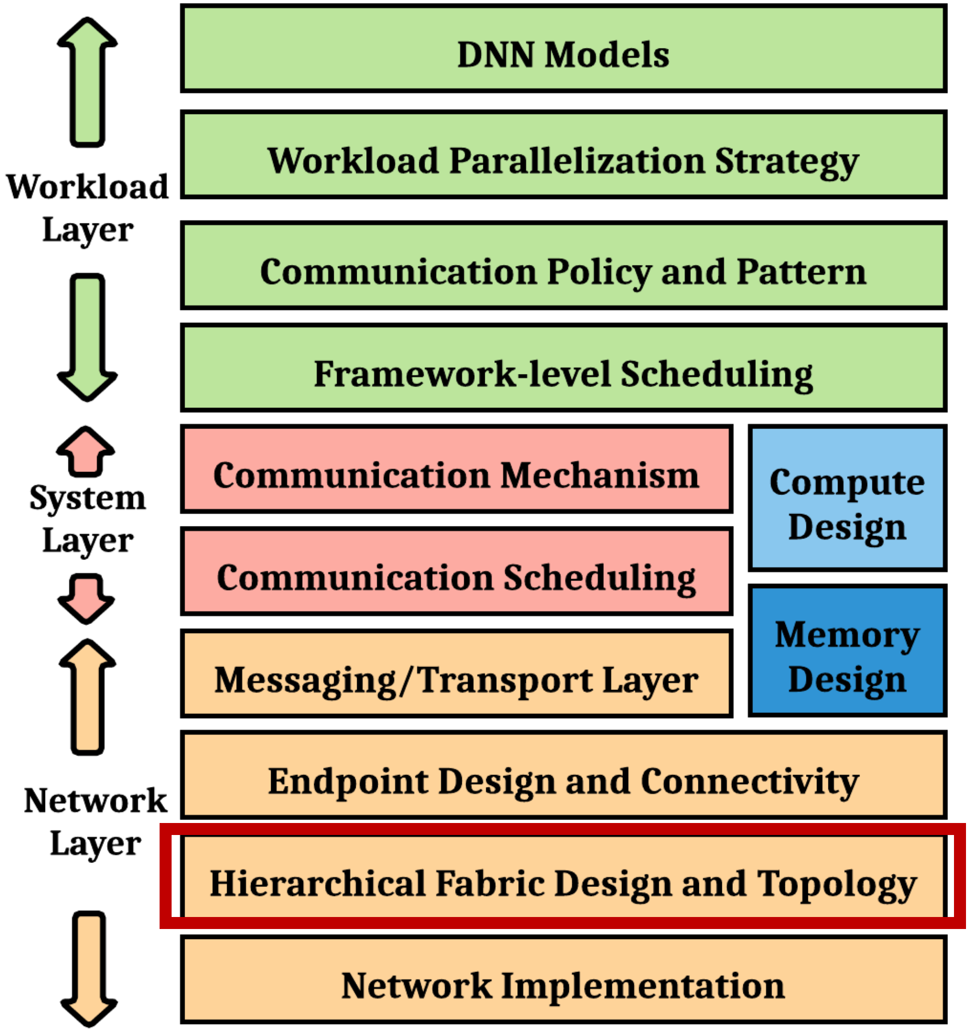
Multi-dimensional Collective Algorithm

- **Phased approach** of Reduce-Scatter and All-Gather



(1) Dim 1: Reduce-Scatter
(2) Dim 2: Reduce-Scatter
(3) Dim 3: Reduce-Scatter
(4) Dim 3: All-Gather
(5) Dim 2: All-Gather
(6) Dim 1: All-Gather

Distributed Training Stack



Workload Layer

System Layer

Network Layer

DNN Models → DLRM, ResNet-50, Transformer, GNMT

Workload Parallelization Strategy → Data, Model, Platform Agnostic Hybrid, Platform-aware Hybrid, Pipelined Parallelism

Communication Policy and Pattern → Distributed worker, Parameter Server

Framework-level Scheduling → Sync/Async, Blocking/Non-blocking

Communication Mechanism → Topology-aware Collectives, Send/Recv, RPC

Compute Design → Dataflow, Microarchitecture, Flexibility, Sparsity Support

Communication Scheduling → LIFO, FIFO, Fusion

Memory Design

Messaging/Transport Layer → TCP, RDMA (+ GPUDirect RDMA)

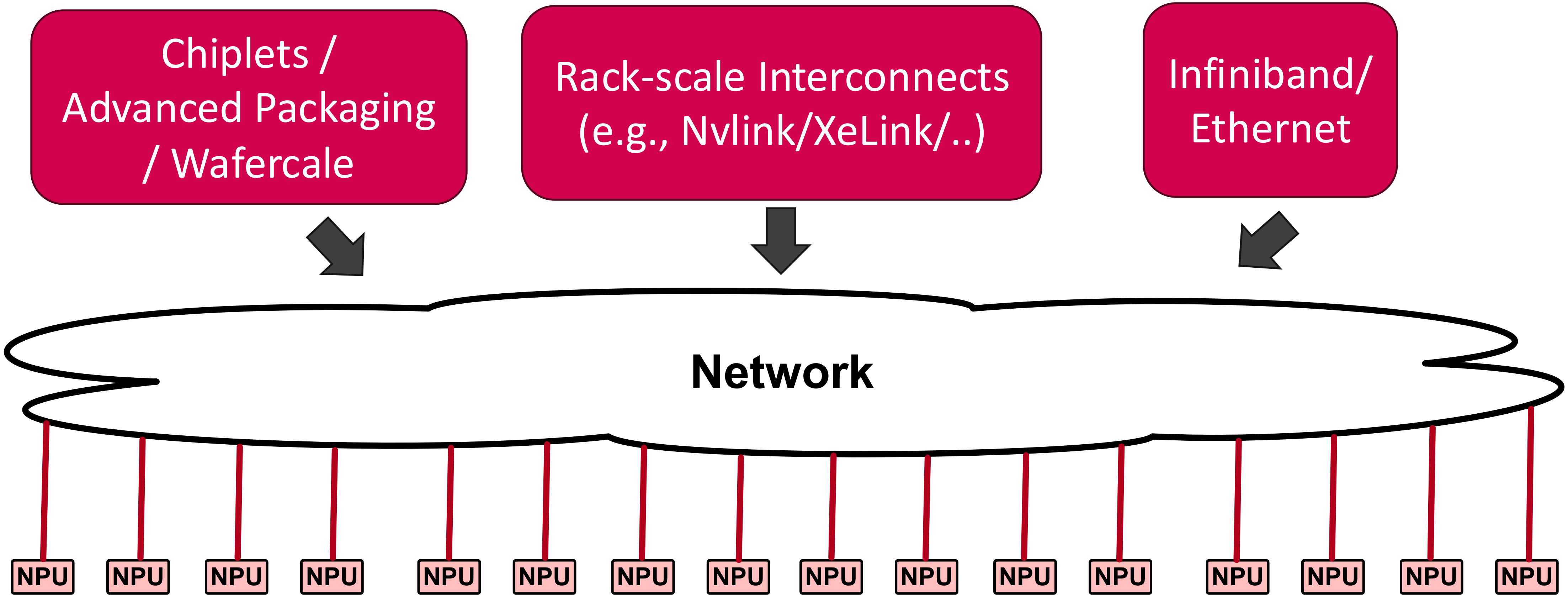
Endpoint Design and Connectivity → # links, BW per link, architecture (chip/package/board), NIC offload, compression

Hierarchical Fabric Design and Topology → Flat vs. Hierarchical, 2D/3D/4D Torus (TPU), Switch (DGX2), Fully-Connected, Hyper-Cube Mesh

Network Implementation → Buffering, Flow-control, Arbitration, Congestion Mgmt

Figure Courtesy: Srinivas Sridharan (Facebook)

Networking Technologies

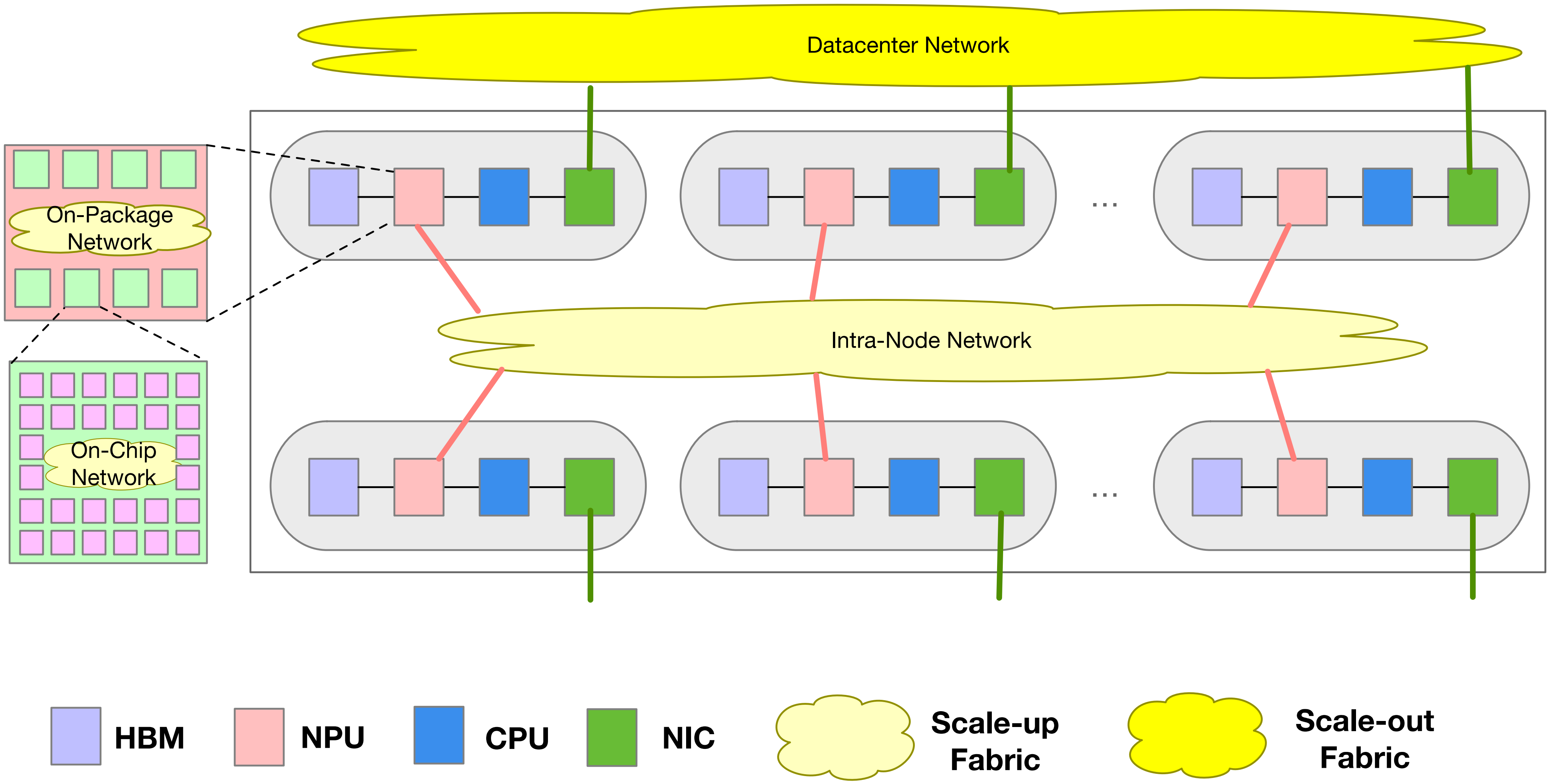


Chiplets / Advanced Packaging / Wafercale

Rack-scale Interconnects (e.g., Nvlink/XeLink/..)

Infiniband/ Ethernet

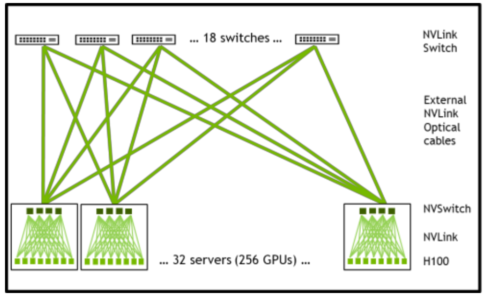
Hierarchical Network Architectures



Examples

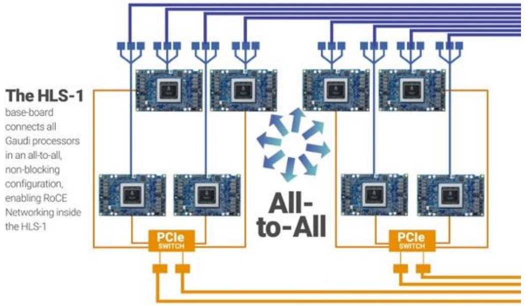
Scale up → scale out

NVIDIA



NVswitch → Infiniband

Intel



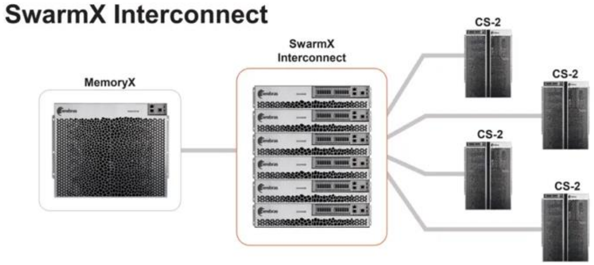
The HLS-1 base-board connects all Gaudi processors in an all-to-all, non-blocking configuration, enabling RoCE Networking inside the HLS-1

Custom NICs → RoCE

Google

3D Electrical Torus → Optical

Cerebras

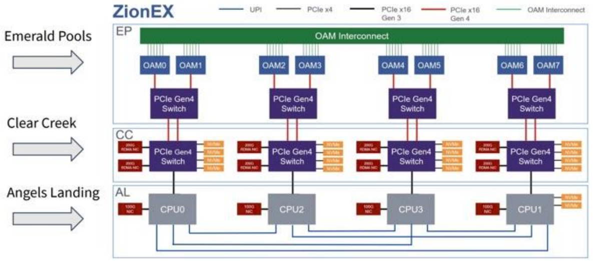


Wafer-scale → SwarmX Tree

AMD

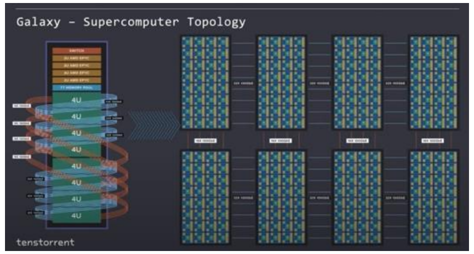
Infiniti → Infiniti

Meta



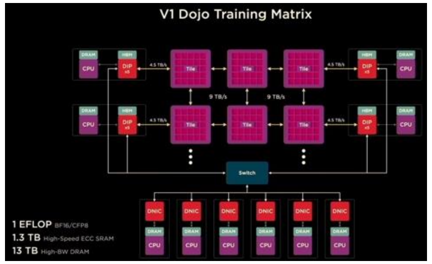
NVlink → RoCE

Tensorrent



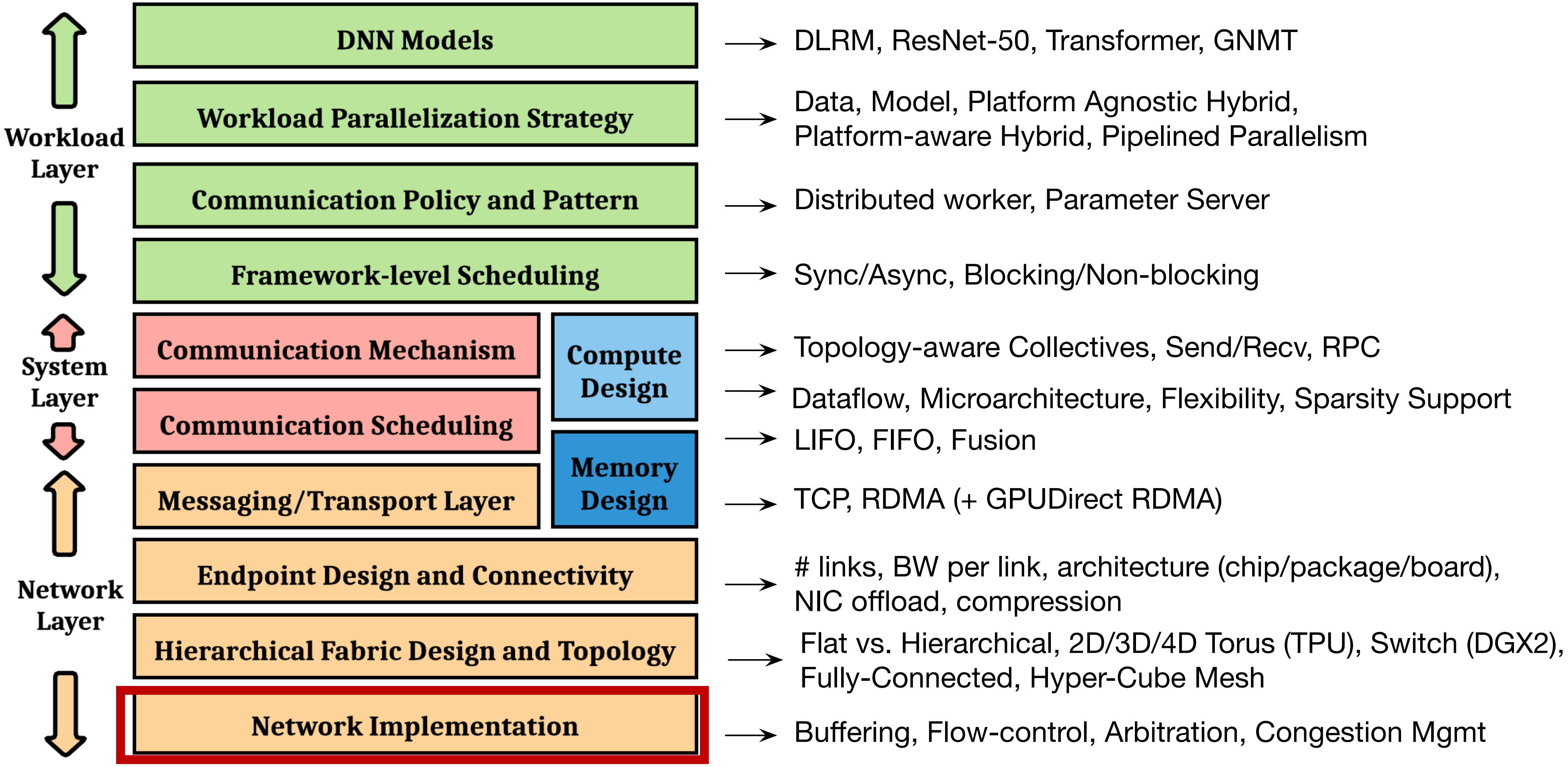
On-package Mesh → off-chip mesh

Tesla



On-package Mesh → Ethernet

Distributed Training Stack



Workload Layer

System Layer

Network Layer

DNN Models → DLRM, ResNet-50, Transformer, GNMT

Workload Parallelization Strategy → Data, Model, Platform Agnostic Hybrid, Platform-aware Hybrid, Pipelined Parallelism

Communication Policy and Pattern → Distributed worker, Parameter Server

Framework-level Scheduling → Sync/Async, Blocking/Non-blocking

Communication Mechanism → Topology-aware Collectives, Send/Recv, RPC

Compute Design → Dataflow, Microarchitecture, Flexibility, Sparsity Support

Communication Scheduling → LIFO, FIFO, Fusion

Memory Design

Messaging/Transport Layer → TCP, RDMA (+ GPUDirect RDMA)

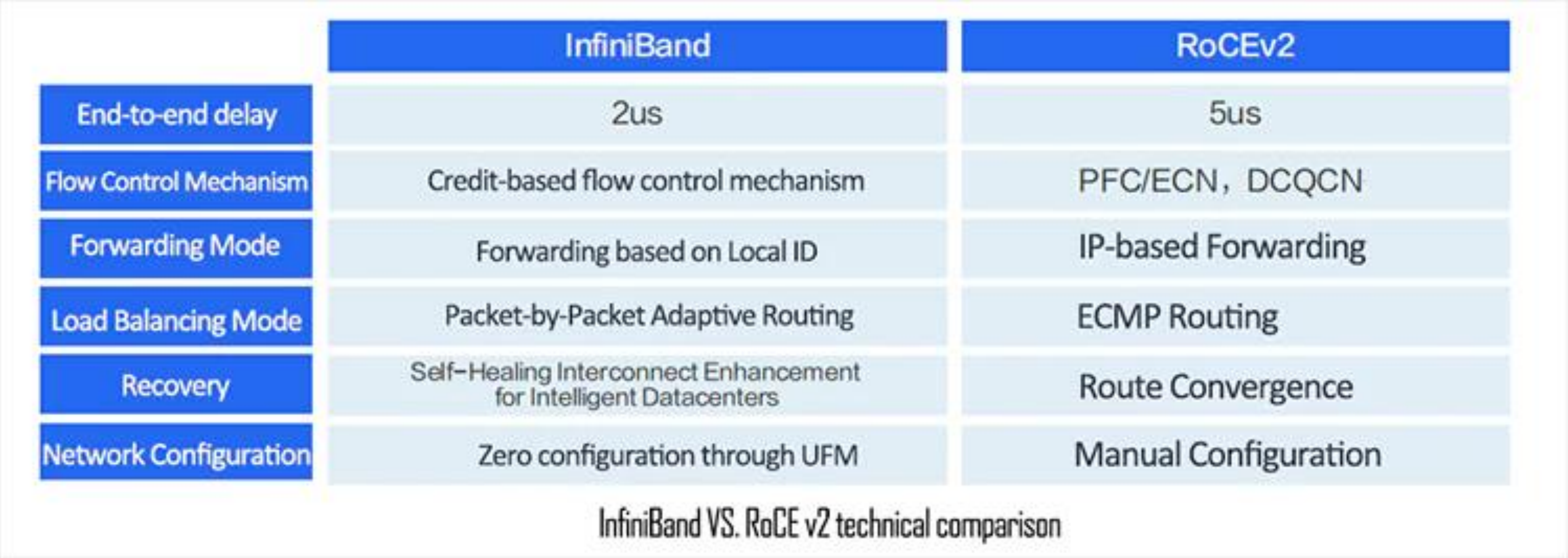
Endpoint Design and Connectivity → # links, BW per link, architecture (chip/package/board), NIC offload, compression

Hierarchical Fabric Design and Topology → Flat vs. Hierarchical, 2D/3D/4D Torus (TPU), Switch (DGX2), Fully-Connected, Hyper-Cube Mesh

Network Implementation → Buffering, Flow-control, Arbitration, Congestion Mgmt

Figure Courtesy: Srinivas Sridharan (NVIDIA)

Example: Infiniband vs RoCE



	InfiniBand	RoCEv2
End-to-end delay	2us	5us
Flow Control Mechanism	Credit-based flow control mechanism	PFC/ECN，DCQCN
Forwarding Mode	Forwarding based on Local ID	IP-based Forwarding
Load Balancing Mode	Packet-by-Packet Adaptive Routing	ECMP Routing
Recovery	Self-Healing Interconnect Enhancement for Intelligent Datacenters	Route Convergence
Network Configuration	Zero configuration through UFM	Manual Configuration

InfiniBand VS. RoCE v2 technical comparison

Summary and Takeaways

- Design of Distributed AI/ML Platforms is an ongoing open-research area
- Many emerging supercomputing systems being designed specifically for this problem!
- Co-design of algorithm and system offers high opportunities for speedup and efficiency