https://synergy.ece.gatech.edu

ASTRA-sim Tutorial
@HotI 2024
Aug 23, 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, School of ECE
Georgia Institute of Technology
tushar@ece.gatech.edu

Will Won
Ph.D. Student, School of CS
Georgia Institute of Technology
william.won@gatech.edu

Contributors

Georgia Tech
Jinsun Yoo
Joongun Park
Changhai Man
Divya Kiran Kadiyala

Meta
Saeed Rashidi

AMD
Brad Beckmann
Furkan Eris
Kishore Punniyamurthy

NVIDIA
Srinivas Sridharan
Taekyung Heo

Intel
Sudarshan Srinivasan

+ many more industry/academia collaborators
+ growing!

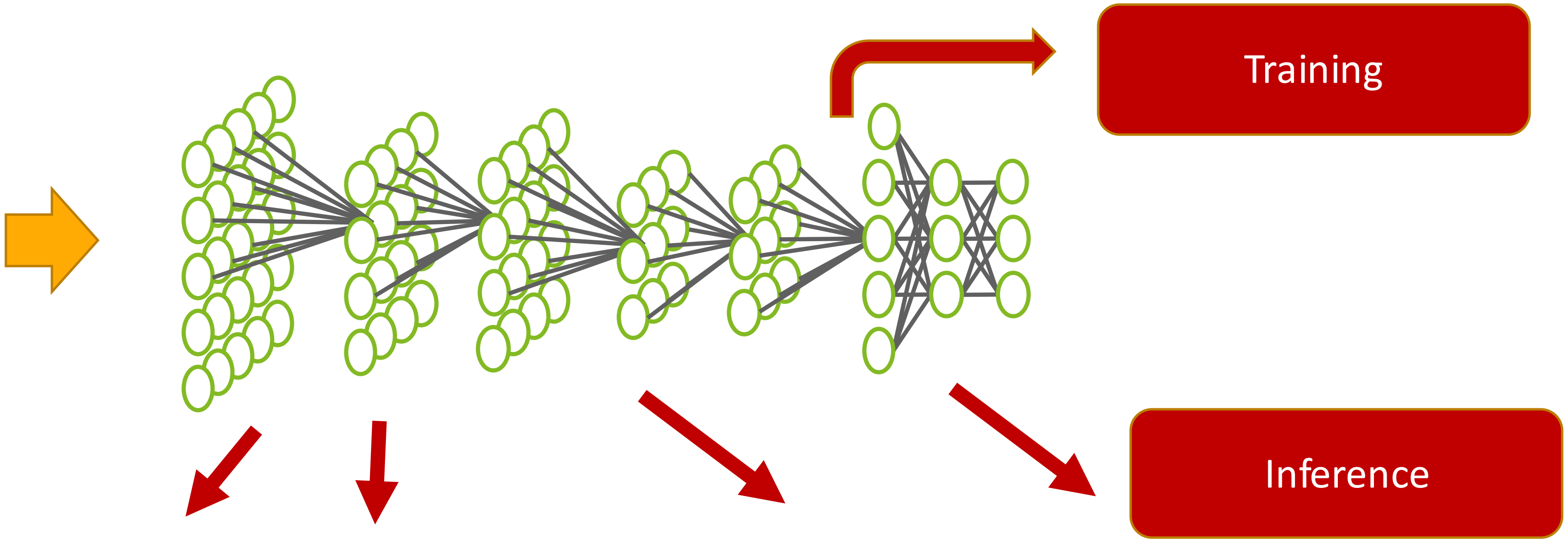
ASTRA-sim Tutorial - Agenda

Time (PDT)	Topic	Presenter
3:00 – 3:30 pm	**Introduction to Distributed ML**	Tushar Krishna
3:30 – 3:45 pm	**Overview of Chakra and ASTRA-sim**	Tushar Krishna
3:45 – 4:35 pm	**Deeper Dive into Chakra and ASTRA-sim**	Will Won
	Workload, System, and Network Layers	
4:35 – 4:45 pm	**Demo**	Will Won
4:45 – 5:00 pm	**Closing Remarks**	Tushar Krishna

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

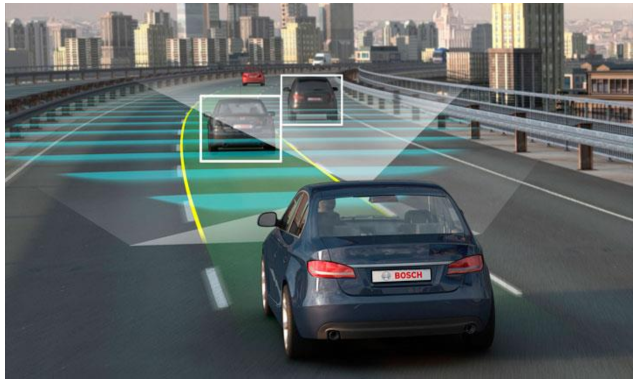
Attention: Tutorial is being recorded

The engine driving the AI Revolution



Training

Inference



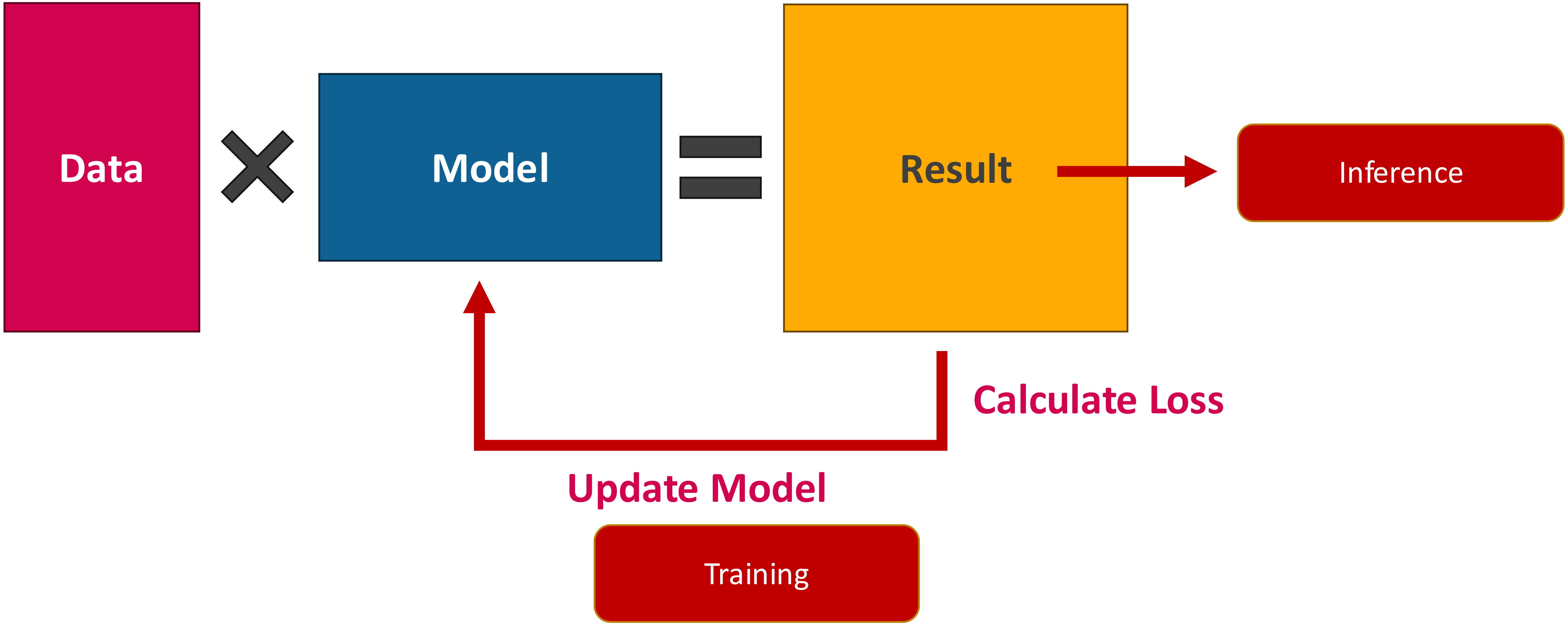
Object Detection

Speech Recognition

Language Understanding

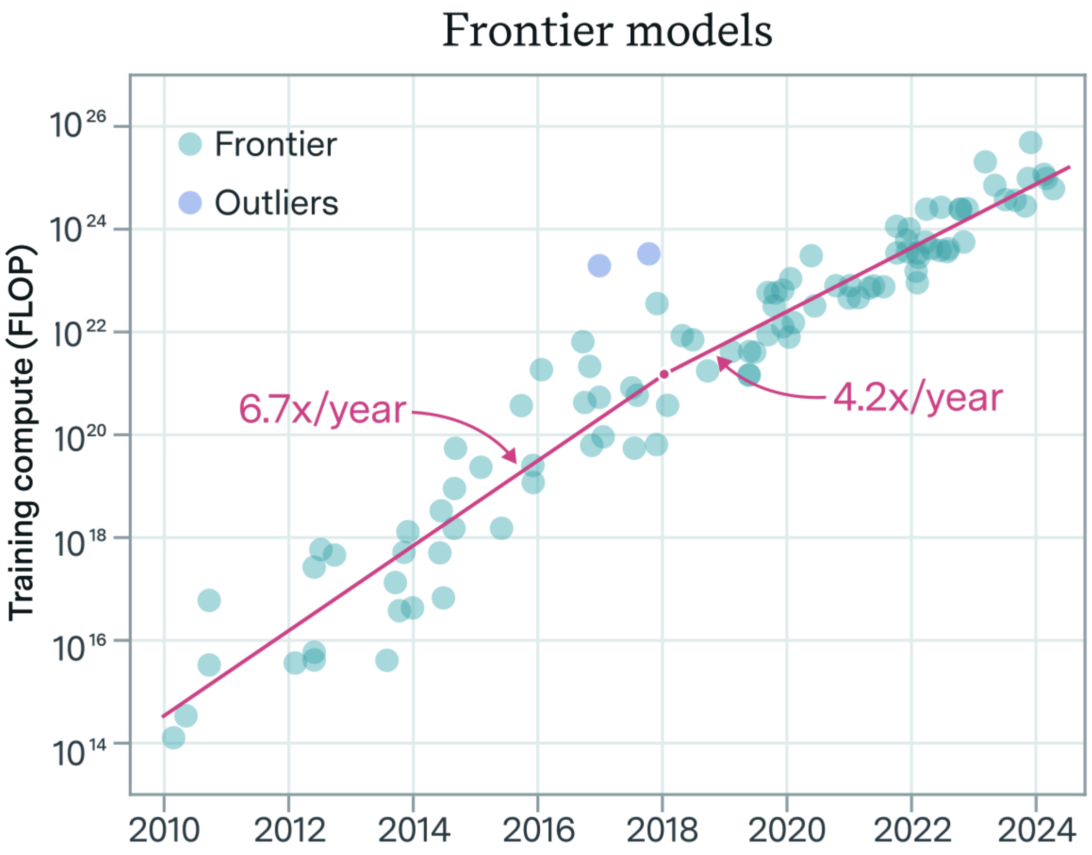
Recommender Systems

Core of ML Execution



Trend 1: Large ML Models

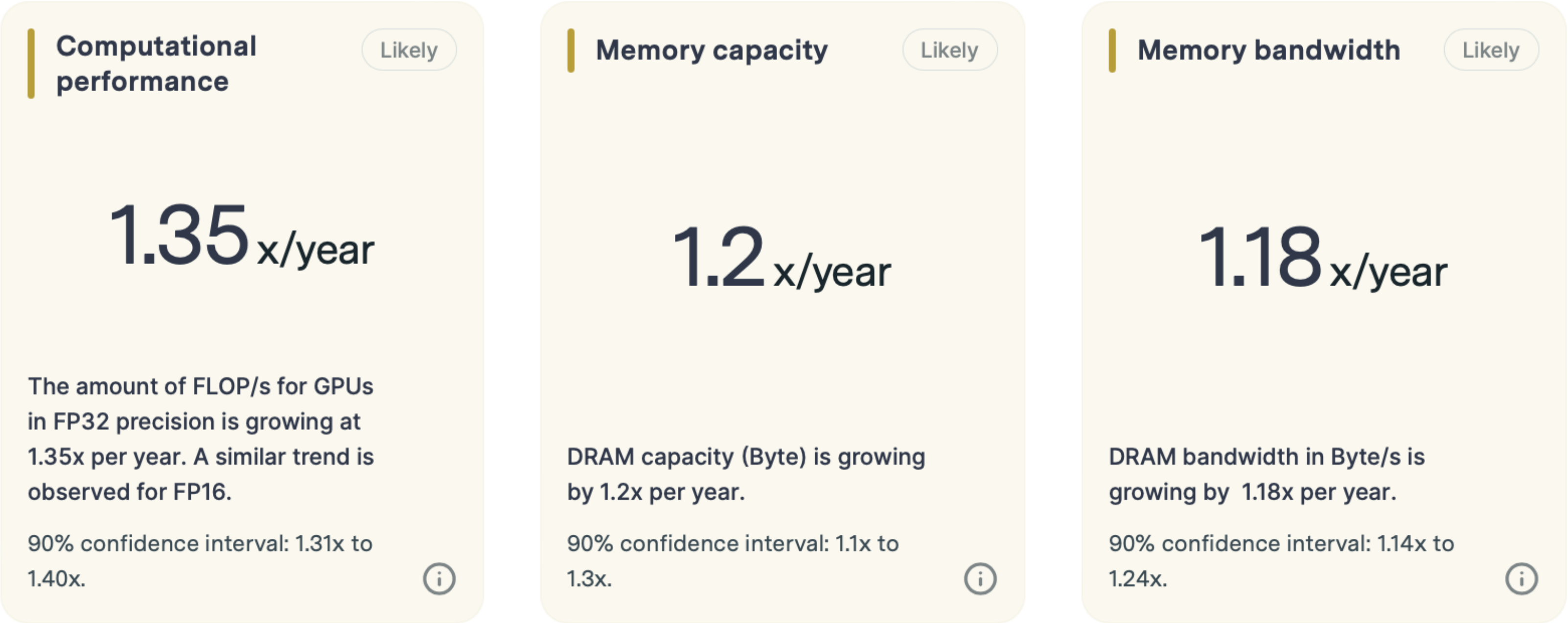
- Machine Learning (ML) models are scaling at an unprecedented rate



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

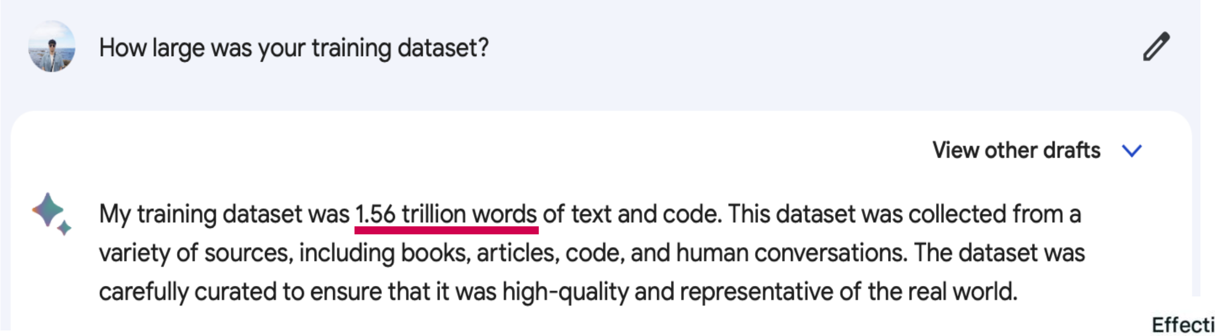
DRAM bandwidth in Byte/s is growing by 1.18x per year.

90% confidence interval: 1.14x to 1.24x.

https://epochai.org/trends

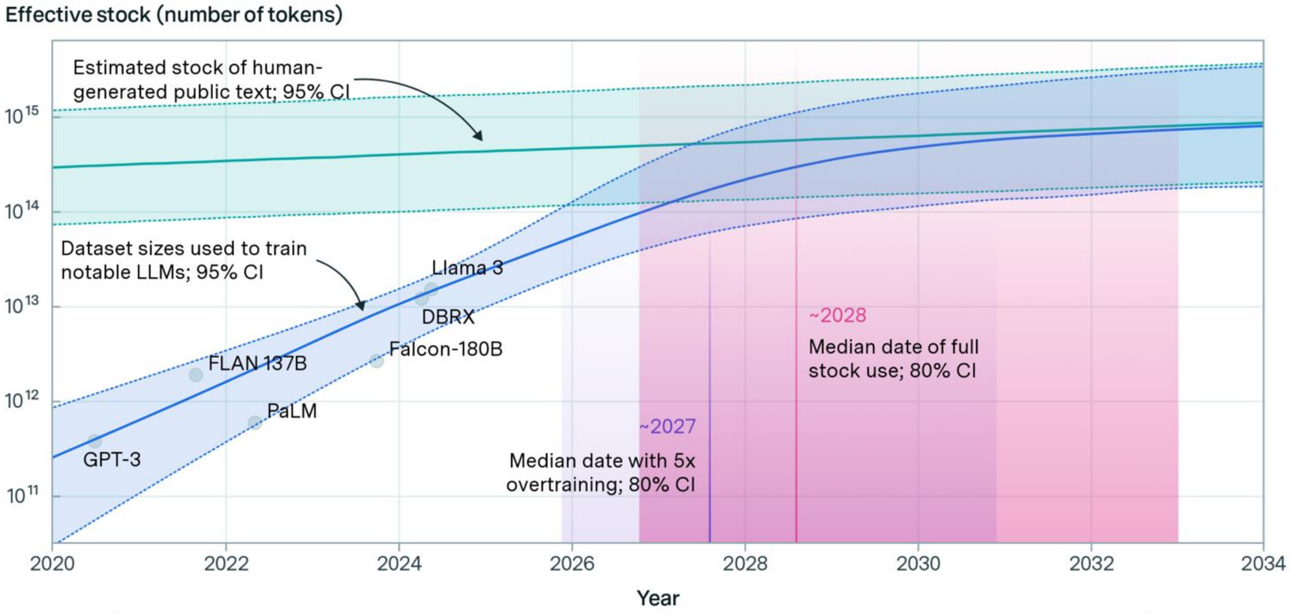
Trend 3: Training Dataset

- Huge training dataset



How large was your training dataset?

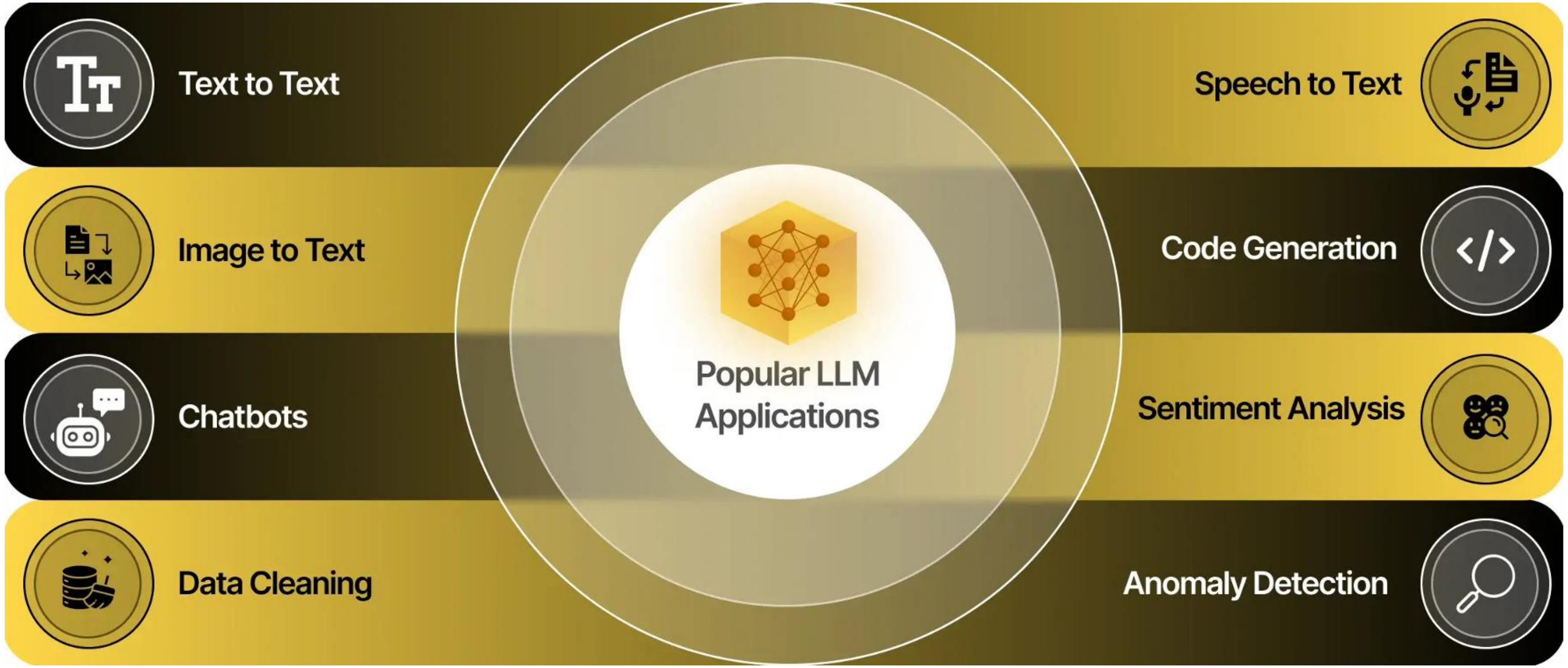
My training dataset was 1.56 trillion words of text and code. This dataset was collected from a variety of sources, including books, articles, code, and human conversations. The dataset was carefully curated to ensure that it was high-quality and representative of the real world.



https://epochai.org/trends

Trend 4: Serving Models

- Various use cases of ML inferences



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

System Implications

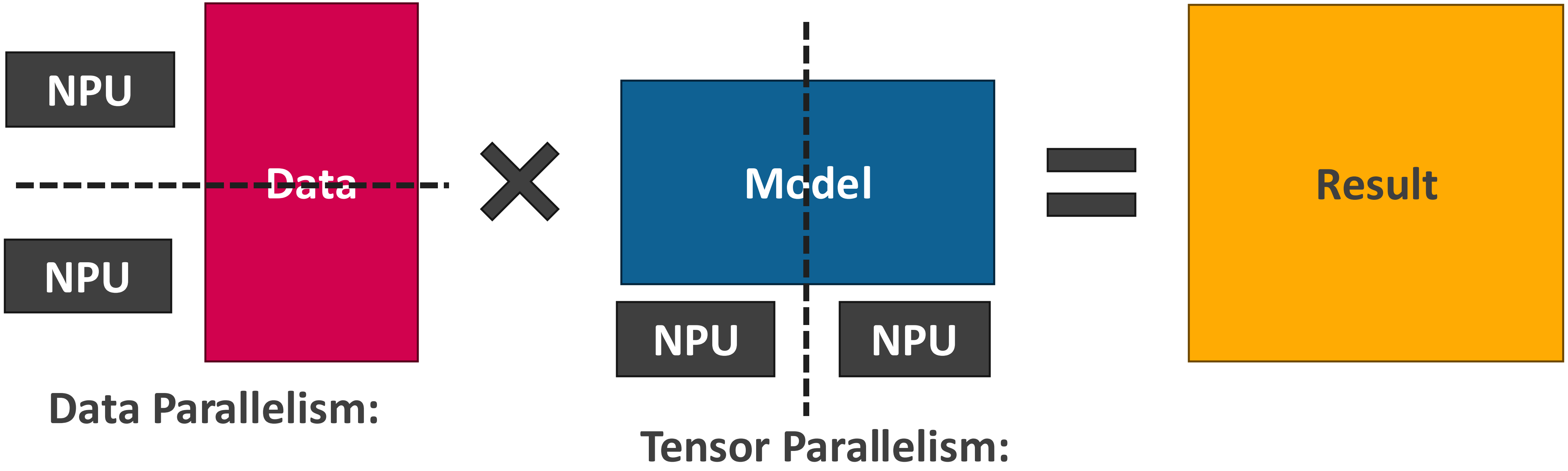
- **Compute**
  - Zeta-scale floating-point operations
  - 355 GPU-years to train GPT-3

- **Memory**
  - 10s of TB required
  - Multiple Neural Processing Units (NPUs) are required to simply *fit* LLM weights

- **Communication**
  - TBs of communication traffic

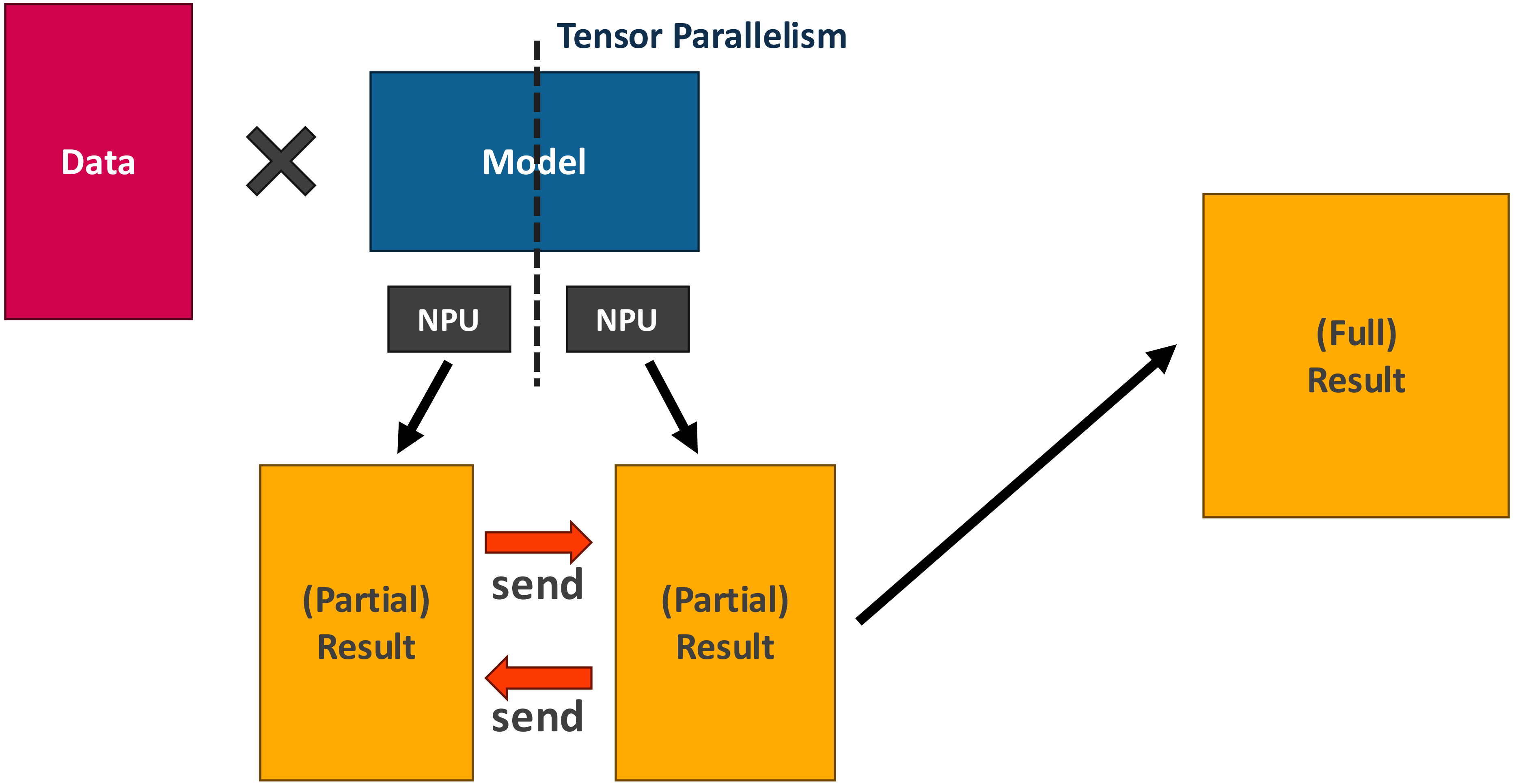
Distributed ML

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

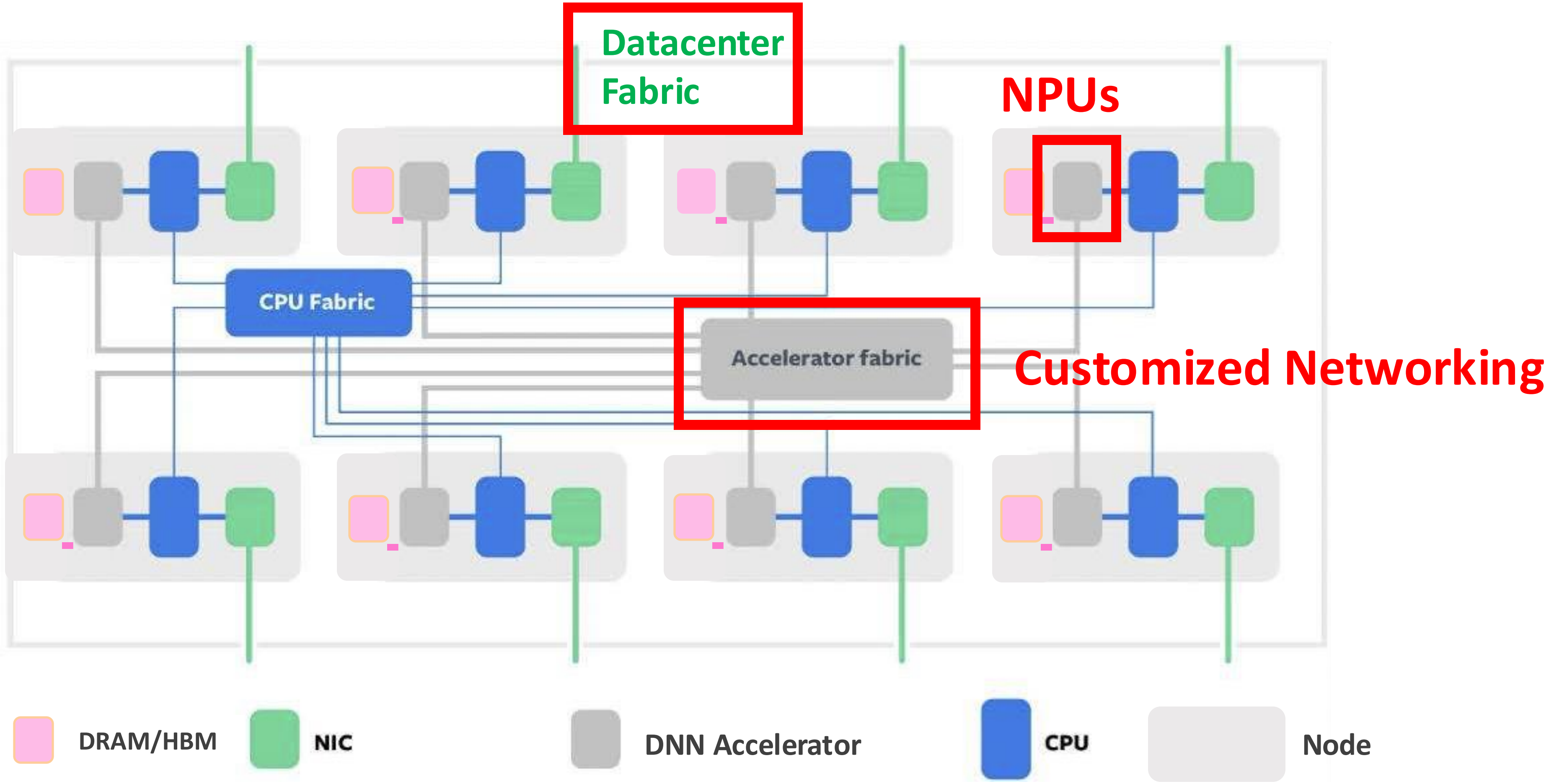


Communication in Distributed ML

- NPUs should communicate to synchronize data



Components of AI Platforms



Datacenter Fabric

NPUs

Customized Networking

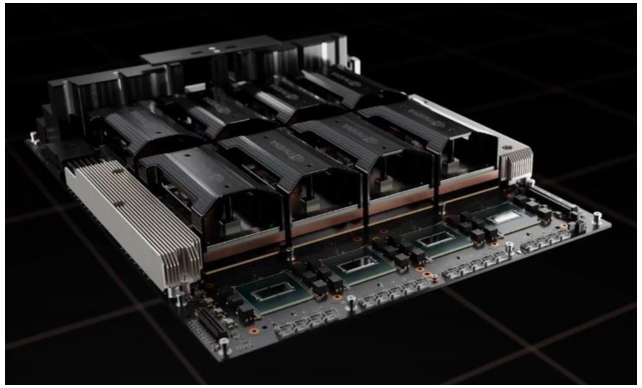
HPC for Distributed ML

- AI Supercomputers

Google Cloud TPUv4

Intel Aurora
Supercomputer

AMD Instinct Platforms

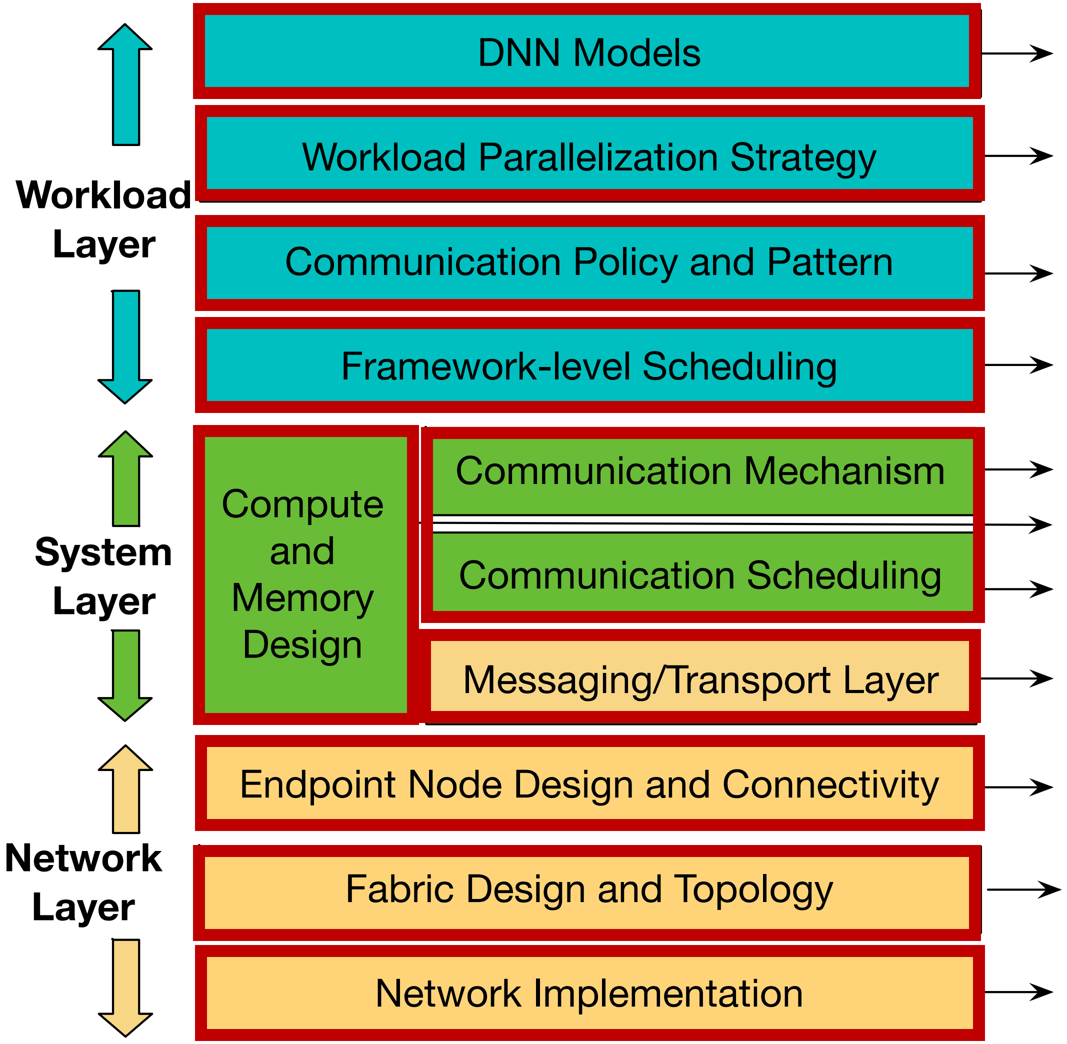


NVIDIA HGX-H100
SuperPod

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 → 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 and Memory Design → Dataflow, Microarchitecture, Flexibility, Sparsity Support
- Communication Scheduling → LIFO, FIFO, Fusion
- Messaging/Transport Layer → TCP, RDMA (+ GPUDirect RDMA)

Network Layer

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

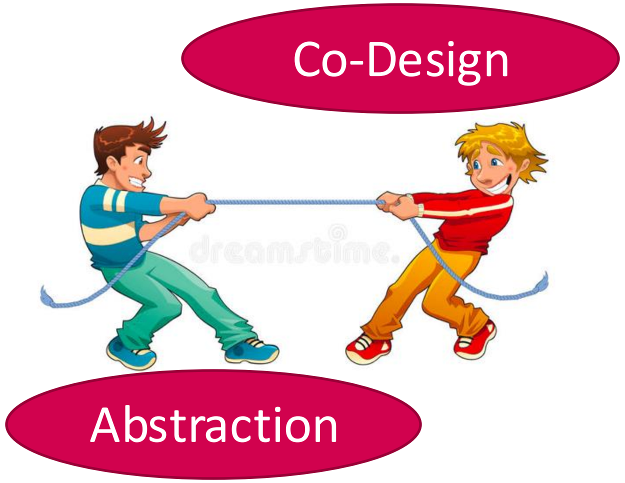
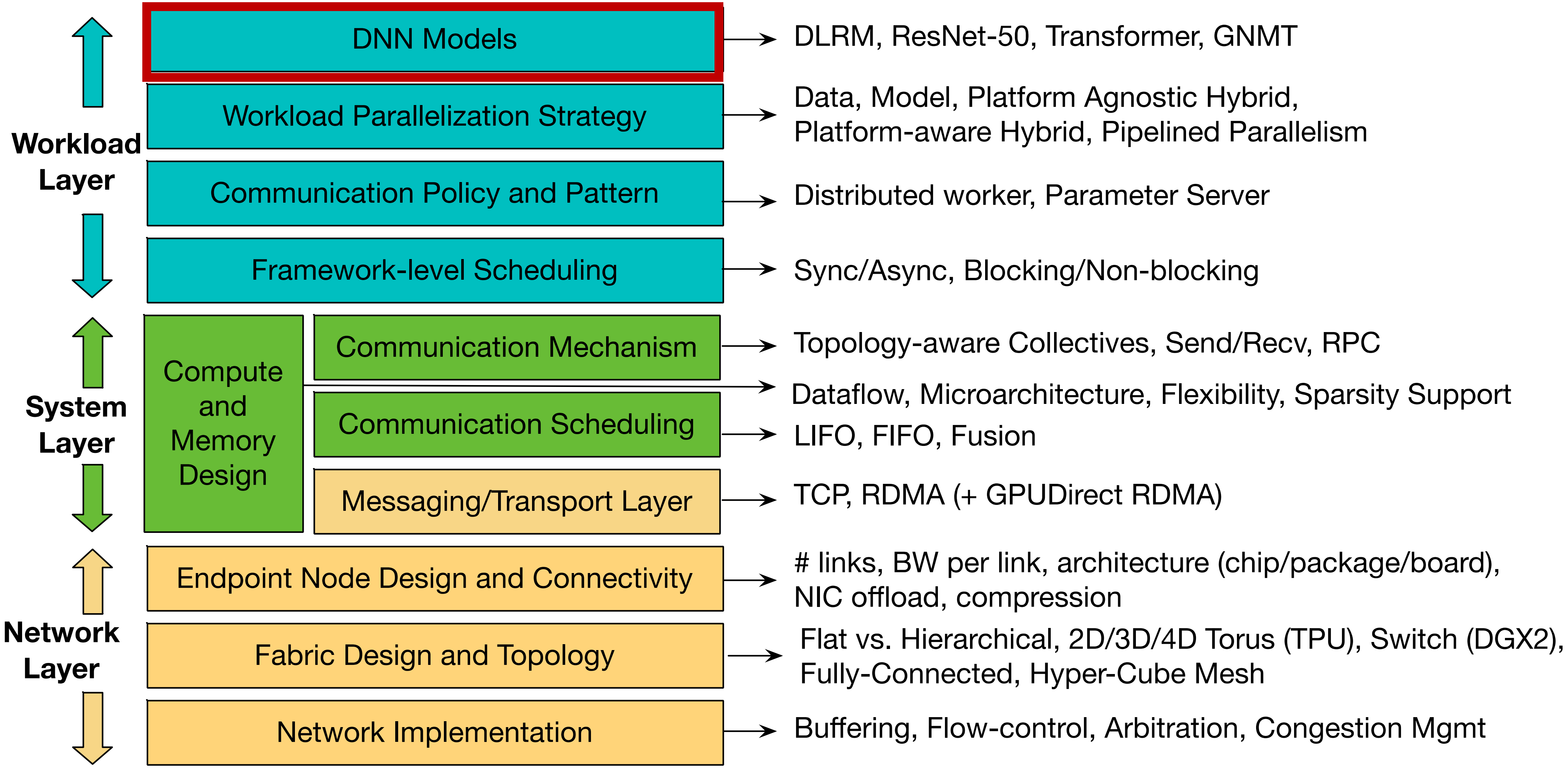


Figure Courtesy: Srinivas Sridharan (NVIDIA)

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 and Memory Design → Dataflow, Microarchitecture, Flexibility, Sparsity Support

Communication Scheduling → LIFO, FIFO, Fusion

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

Network Layer

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

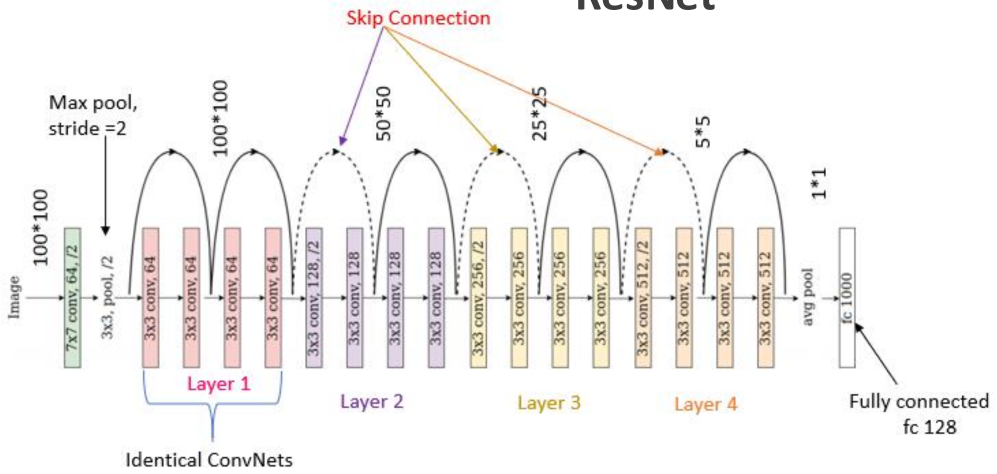
Fabric Design and Topology → Flat vs. Hierarchical, 2D/3D/4D Torus (TPU), Switch (DGX2), Fully-Connected, Hyper-Cube Mesh

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

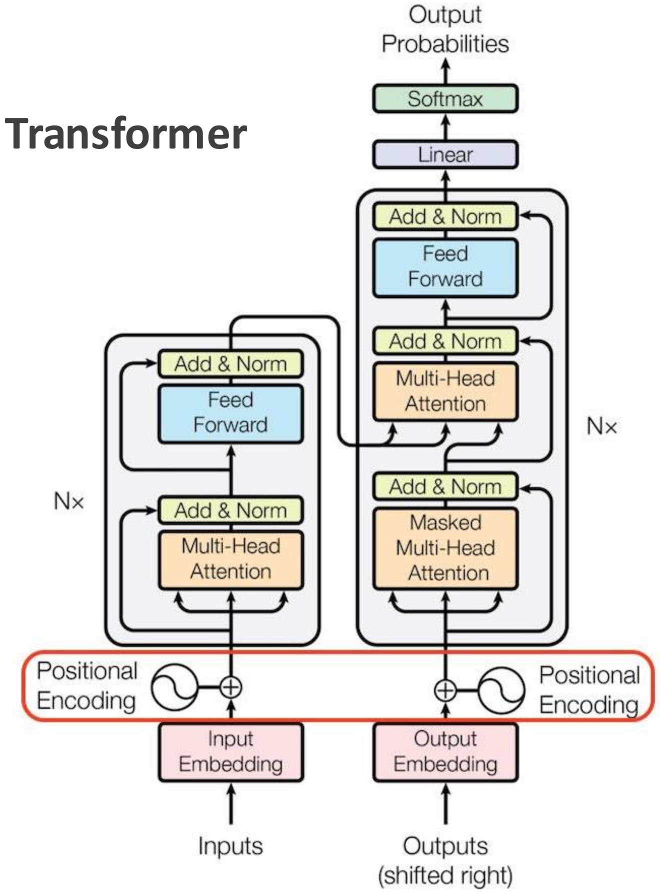
Figure Courtesy: Srinivas Sridharan (NVIDIA)

DNN Models

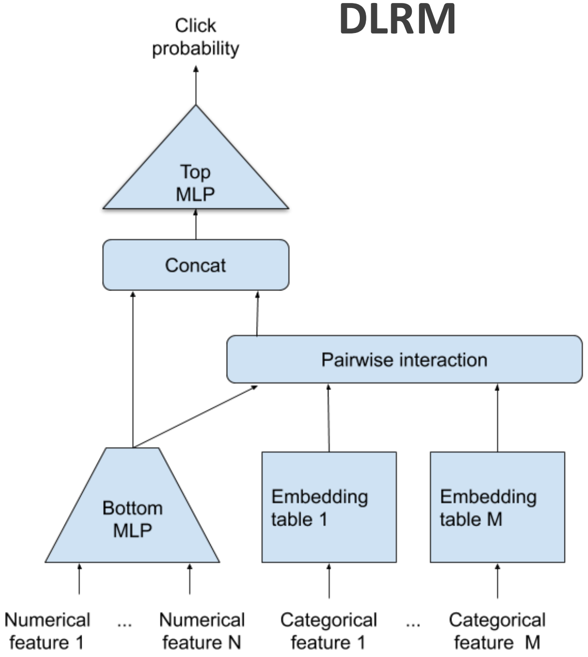
ResNet



Transformer



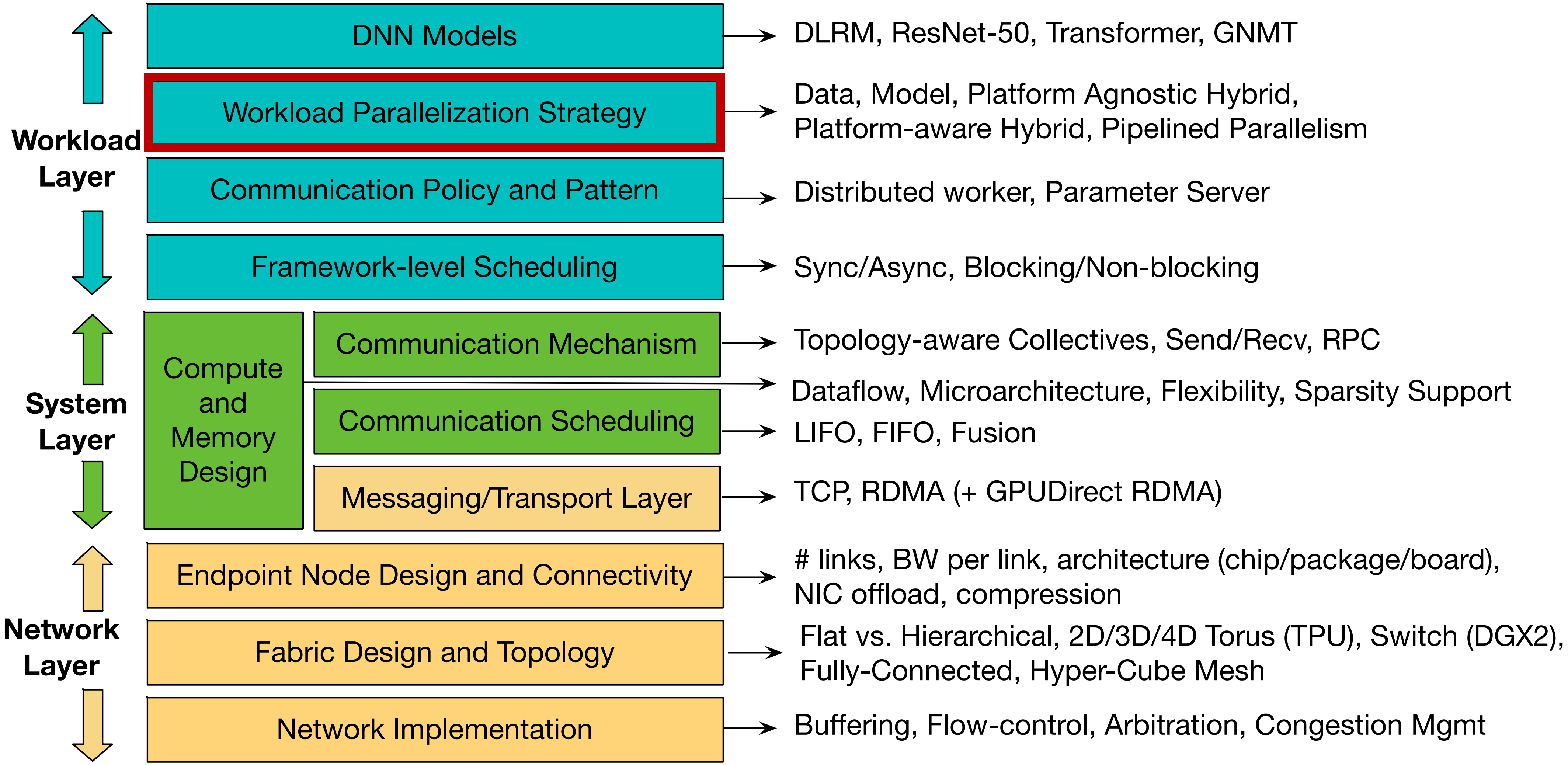
DLRM



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 and Memory Design → Dataflow, Microarchitecture, Flexibility, Sparsity Support
- Communication Scheduling → LIFO, FIFO, Fusion
- Messaging/Transport Layer → TCP, RDMA (+ GPUDirect RDMA)

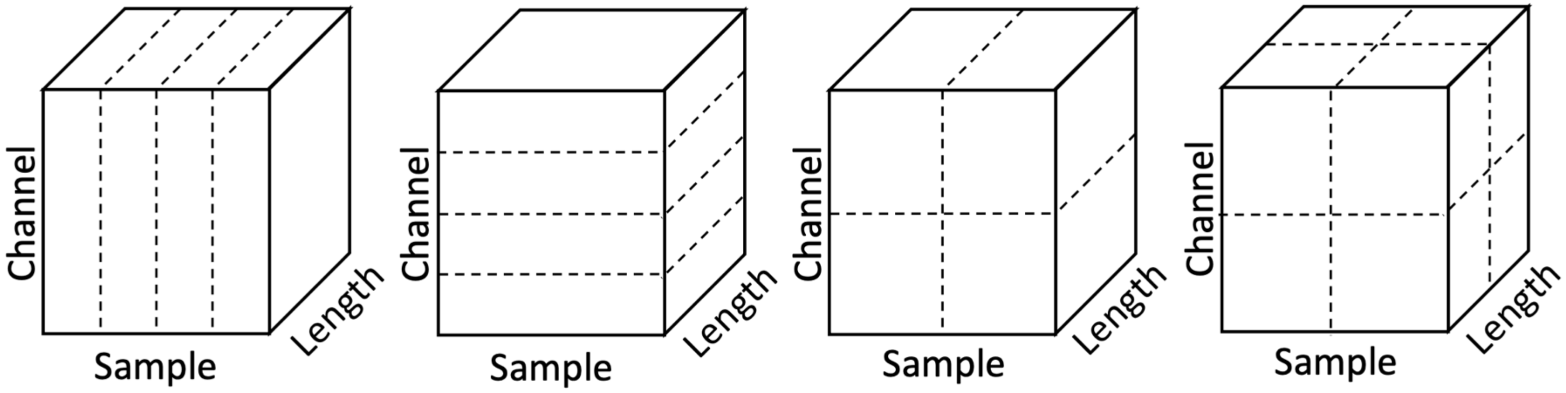
Network Layer

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

- In distributed training, we **distribute model and/or training data**
- **Parallelization strategy** defines how to shard/distribute them
  - Finding an optimal parallelization strategy is active area of research
- Multiple ways to distribute model/data



Data Parallelism (S)

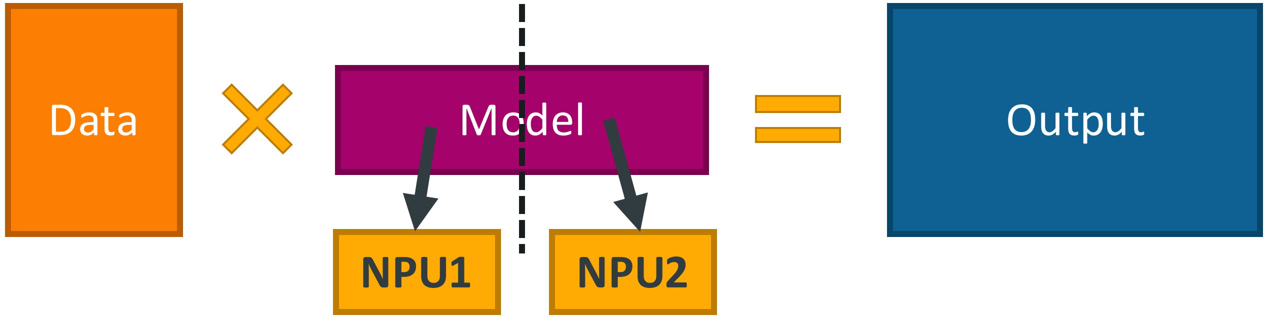
Model Parallelism (P)

Hybrid Parallelism (S, P)

Hybrid Parallelism (S, A, P)

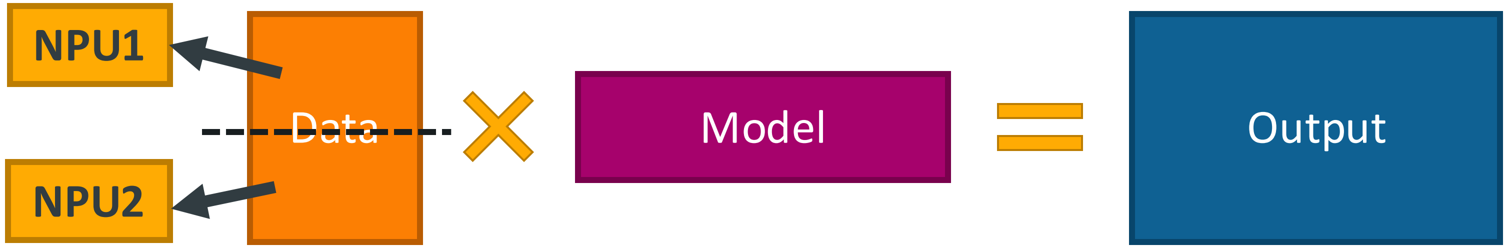
Tensor Parallelism

- Shard and distribute **DNN model** over NPUs
  - In order to **fit large model** on each NPU



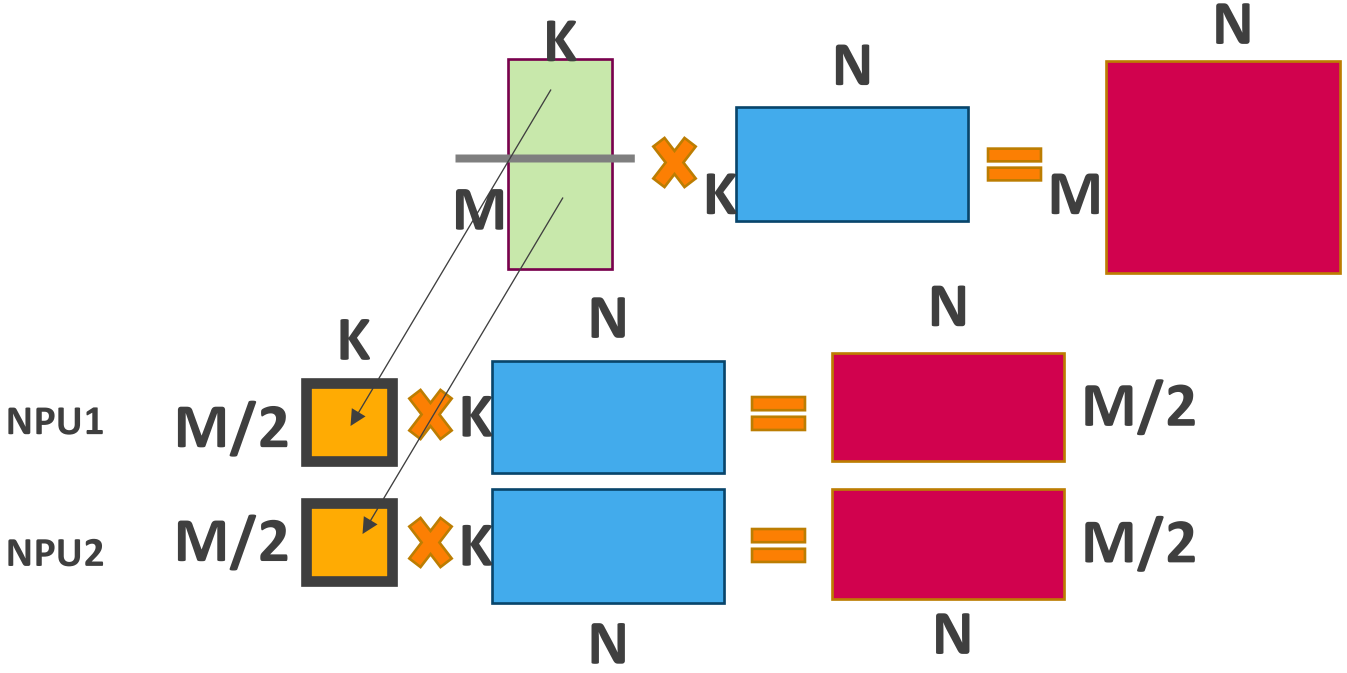
Data Parallelism

- Disperse **training data** over NPUs
  - In order to increase **training throughput**



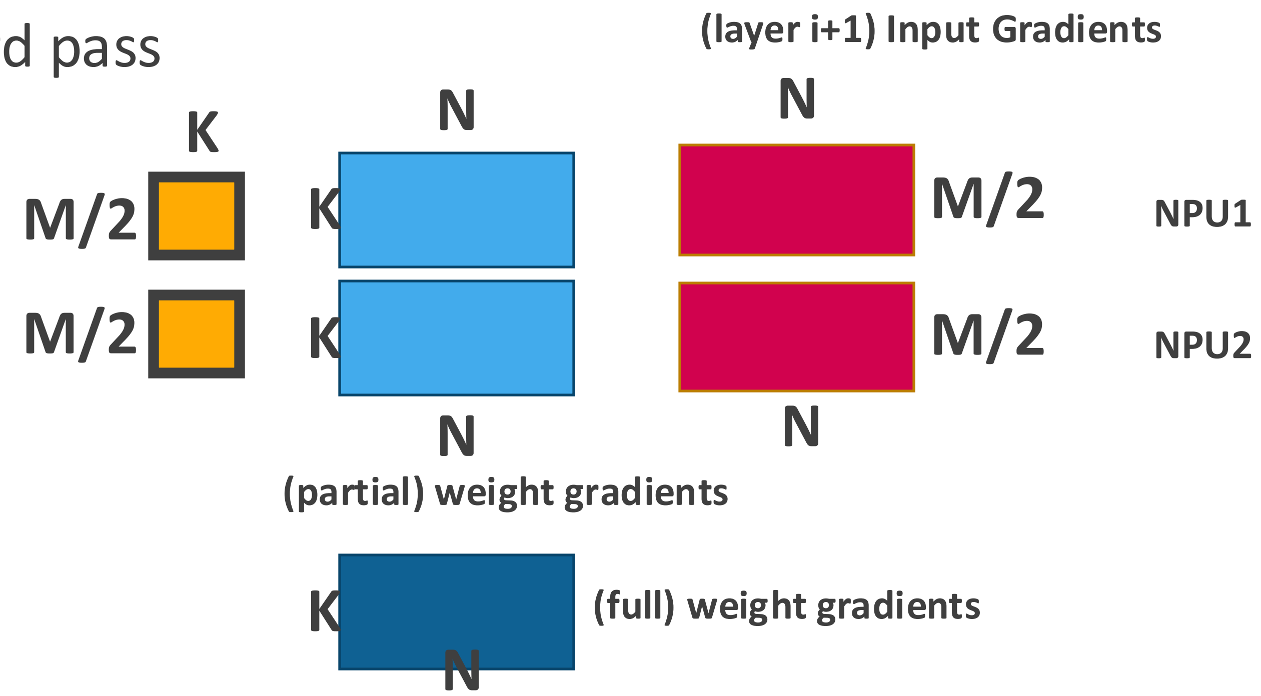
Data Parallel Training (Forward Pass)

- **Distribute training data** across multiple nodes
- **Replicate DNN model** along all nodes.



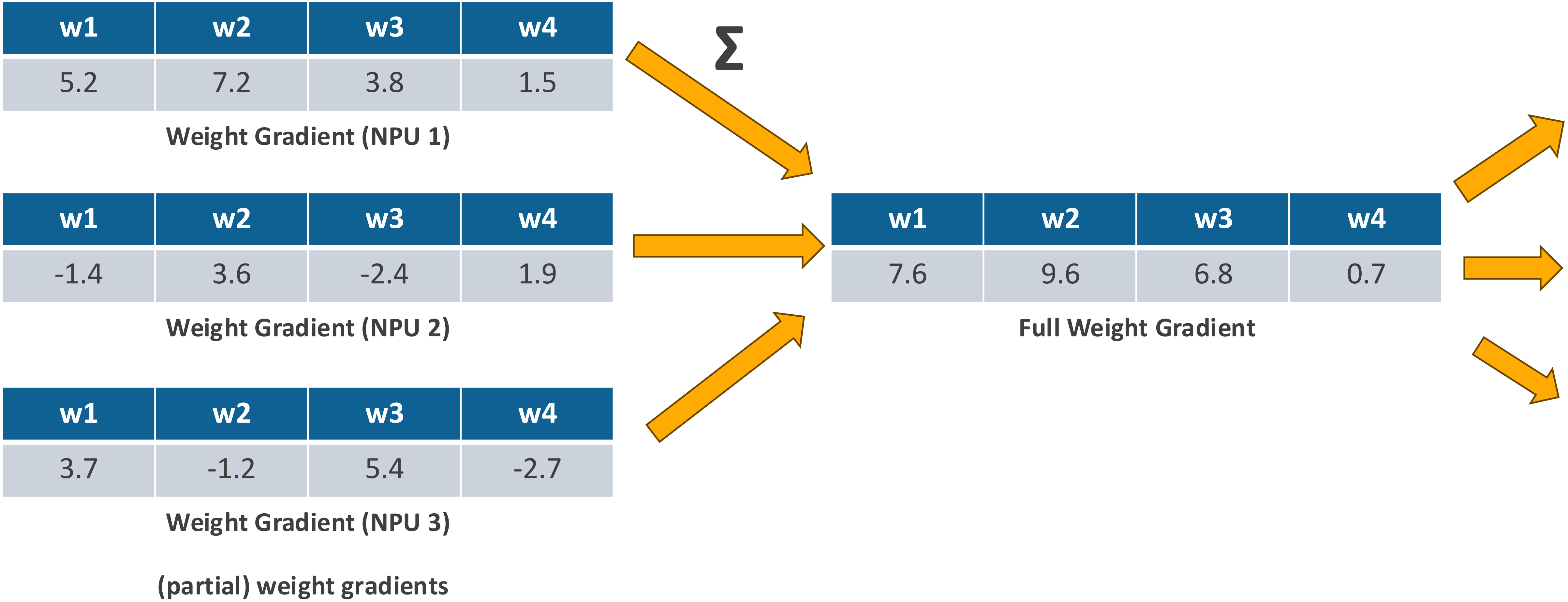
Data Parallel Training (Backward Pass)

- Compute (partial) **Weight Gradients**
- **Synchronize** (partial) weight gradients
  - To compute **(full) weight gradient**
- **Compute** Input Gradients
  - For layer (*i - 1*) backward pass



Weight Gradient Synchronization

- **Sum** partial weight gradients to compute full weight gradients



w1	w2	w3	w4
5.2	7.2	3.8	1.5

Weight Gradient (NPU 1)

w1	w2	w3	w4
-1.4	3.6	-2.4	1.9

Weight Gradient (NPU 2)

w1	w2	w3	w4
3.7	-1.2	5.4	-2.7

Weight Gradient (NPU 3)

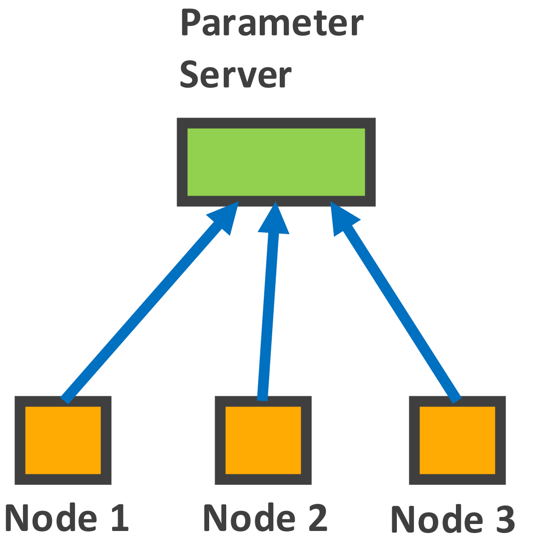
(partial) weight gradients

w1	w2	w3	w4
7.6	9.6	6.8	0.7

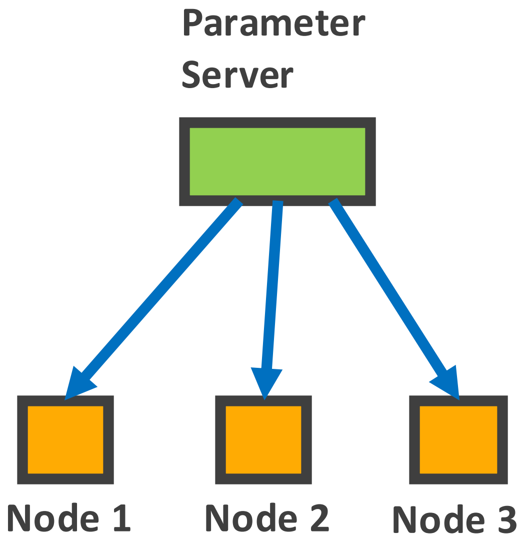
Full Weight Gradient

Communication Handling

- **Parameter Server**



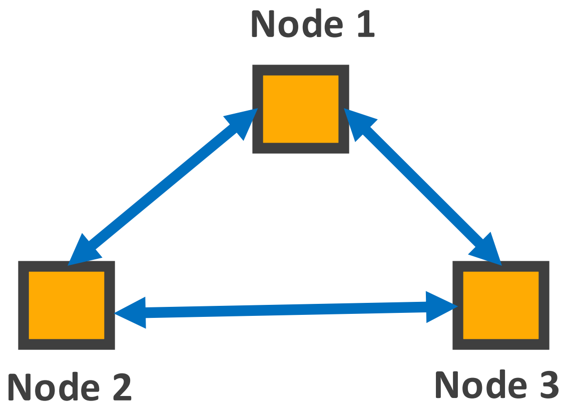
Step 1: Each node sends its model gradients to the parameter server to be reduced with other gradients and update the model

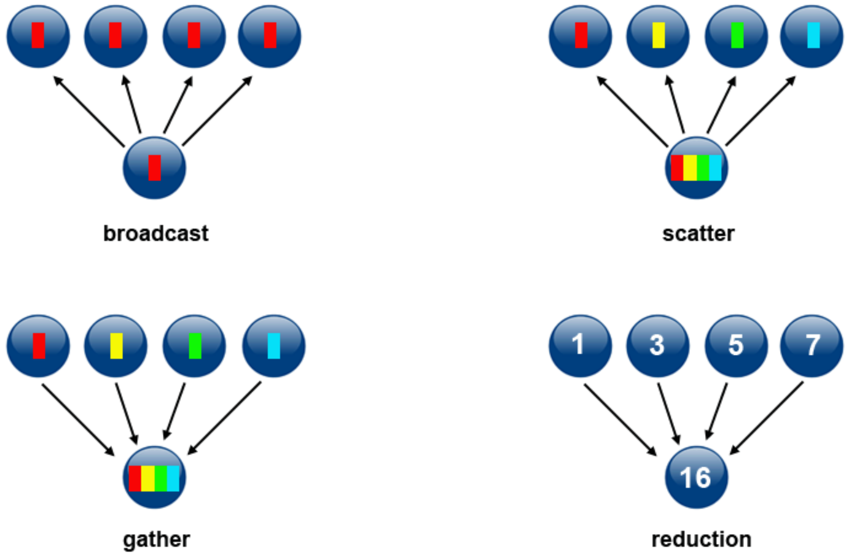


Step 2: The parameter server sends the updated model to the compute nodes to begin the new iteration.

Communication Handling

- **Collective-based:** Compute Nodes directly talk to each other to globally reduce their gradients and update the model through a collective communication pattern (e.g., All Reduce).

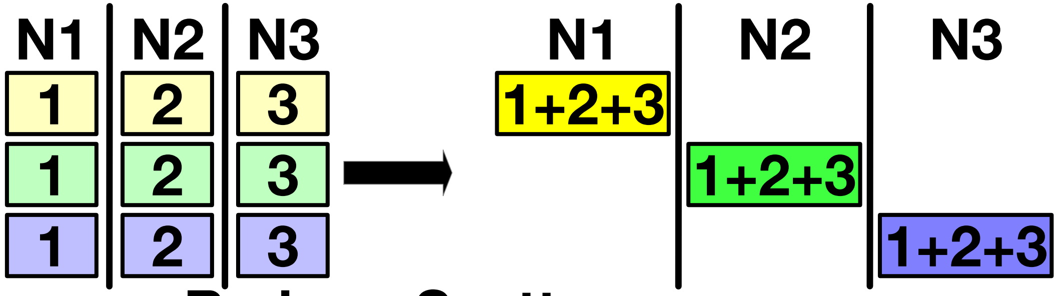




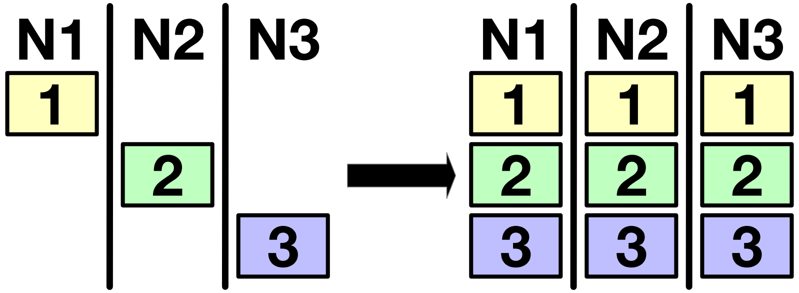
"Collective Communication"
(from MPI)

Collective Communications

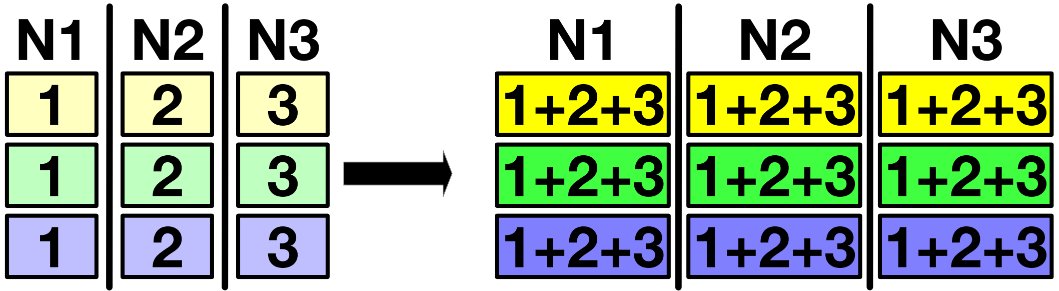
- Distributed ML Communication Pattern → MPI Collectives



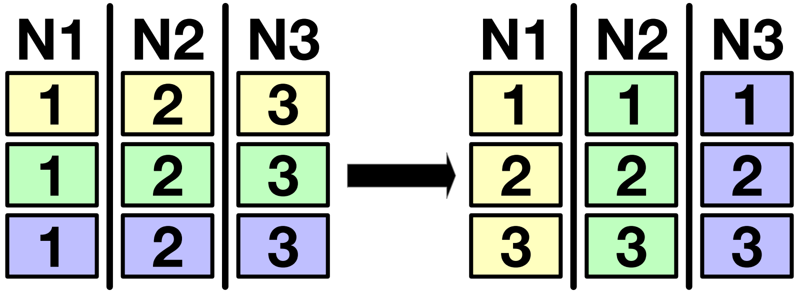
Reduce-Scatter



All-Gather



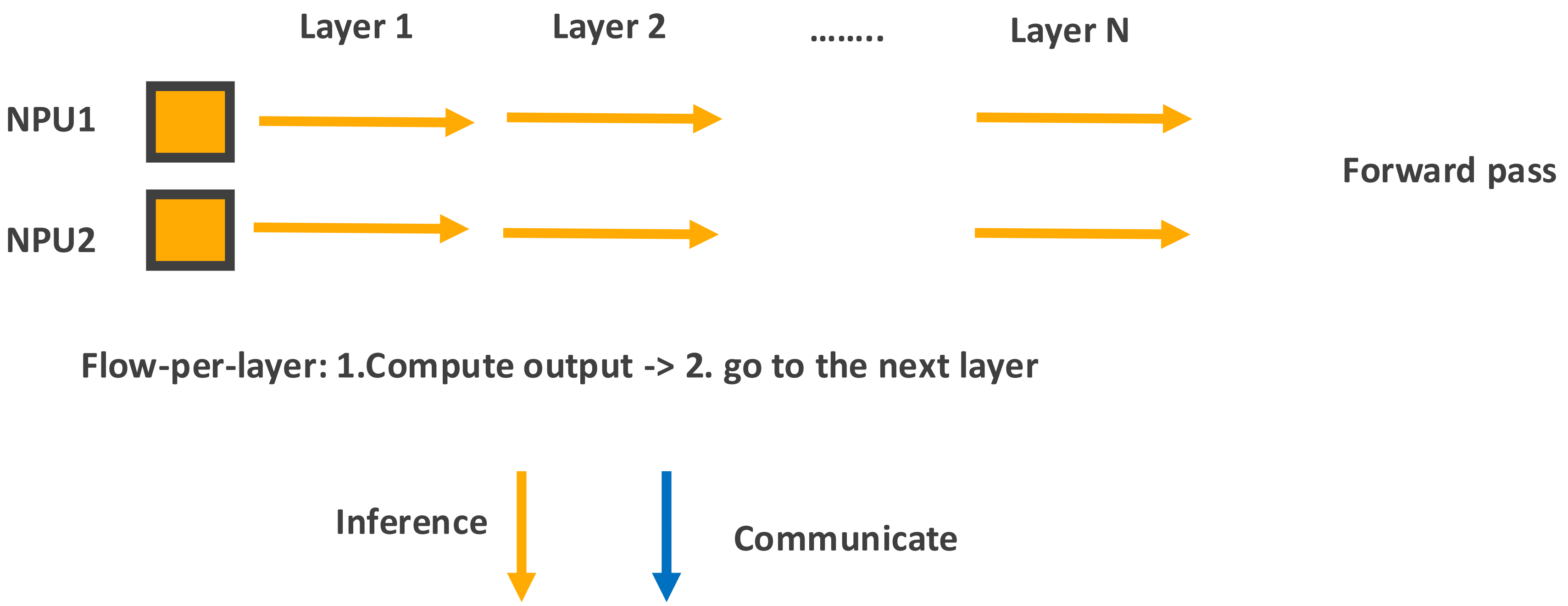
All-Reduce



All-to-All

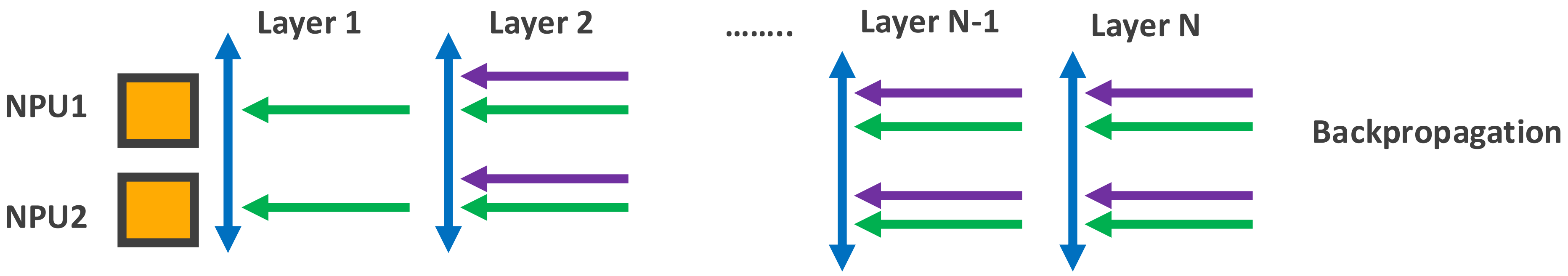
Communication in Data Parallel Training

- **No communication** during the forward pass.



Communication in Data Parallel Training

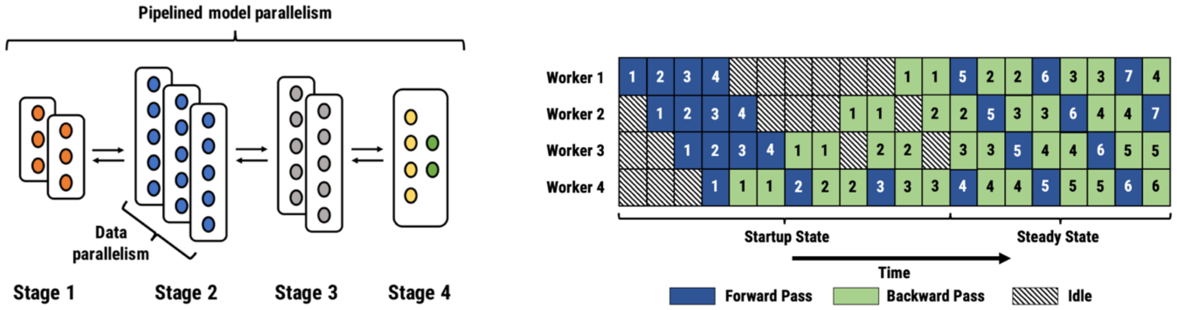
- **Communicate weight gradients** during the backpropagation pass.
  - Via *All Reduce* “Collective”



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**
- ↓ **All Reduce Collective**
- ↓ **Blocking Communicate**

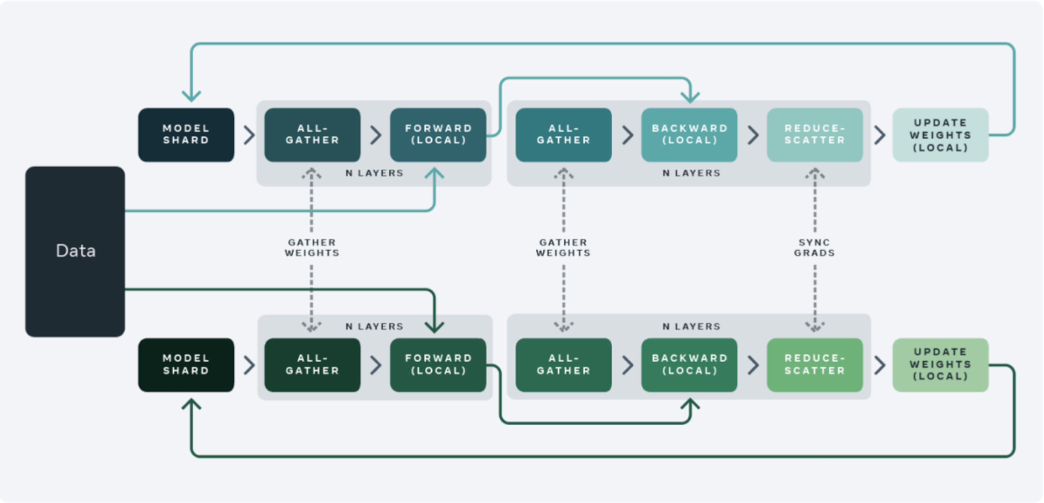
More recent examples



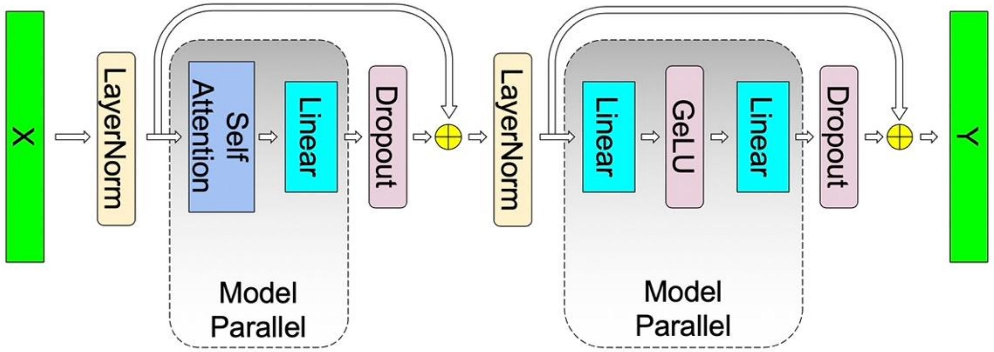
Worker 1	🟦1	🟦2	🟦3	🟦4	▨	▨	▨	▨	▨	▨	🟩1	🟩1	🟦5	🟩2	🟩2	🟦6	🟩3	🟩3	🟦7	🟩4
Worker 2	▨	🟦1	🟦2	🟦3	🟦4	▨	▨	▨	🟩1	🟩1	▨	🟩2	🟩2	🟦5	🟩3	🟩3	🟦6	🟩4	🟩4	🟦7
Worker 3	▨	▨	🟦1	🟦2	🟦3	🟦4	🟩1	🟩1	▨	🟩2	🟩2	▨	🟩3	🟩3	🟦5	🟩4	🟩4	🟦6	🟩5	🟩5
Worker 4	▨	▨	▨	🟦1	🟩1	🟩1	🟦2	🟩2	🟩2	🟦3	🟩3	🟩3	🟦4	🟩4	🟩4	🟦5	🟩5	🟩5	🟦6	🟩6

PipeDream (Microsoft)

Fully sharded data parallel training



FSDP (Meta)

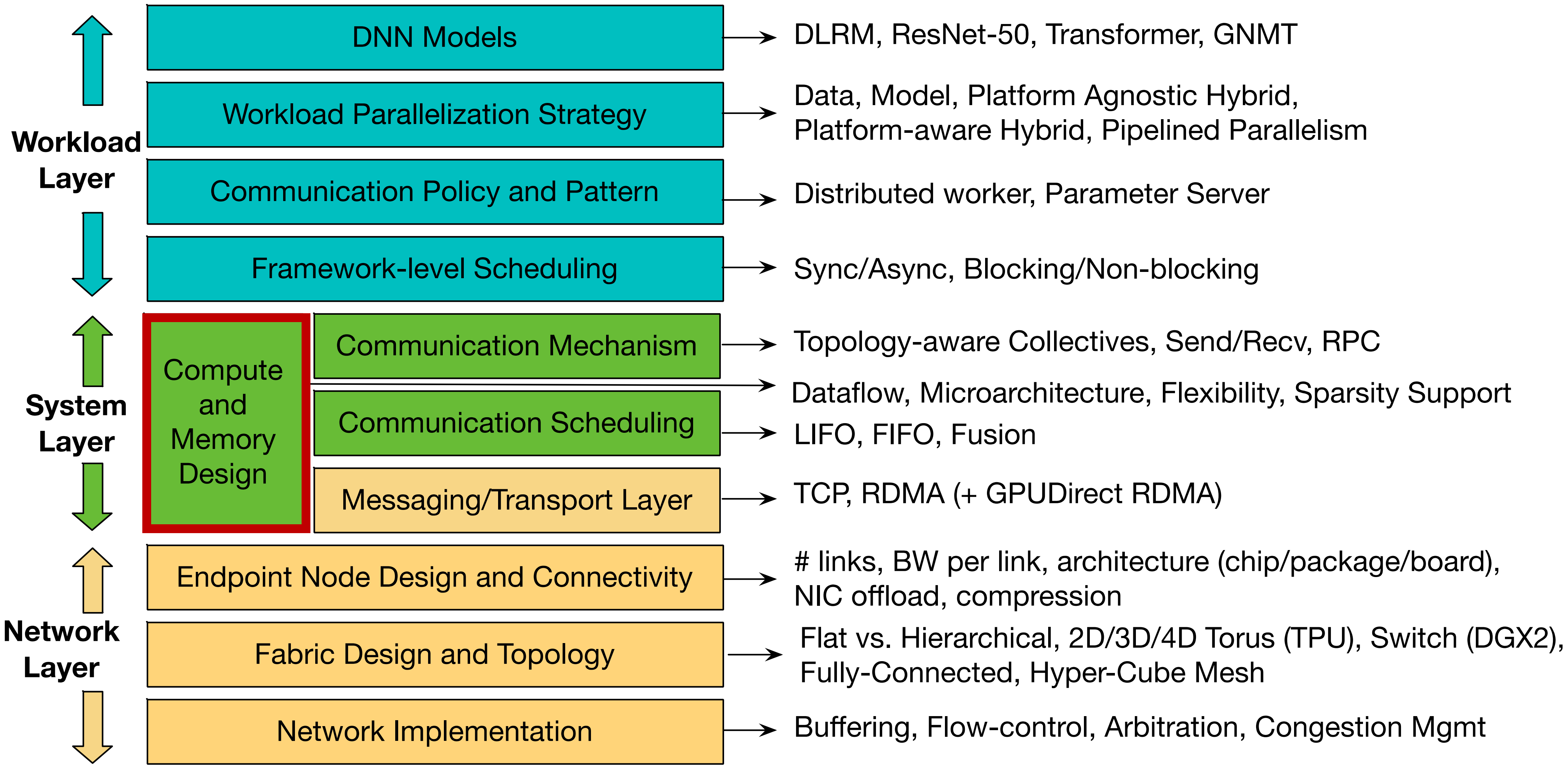


2 All-Reduce
(forward + backward)

2 All-Reduce
(forward + backward)

MegatronLM (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

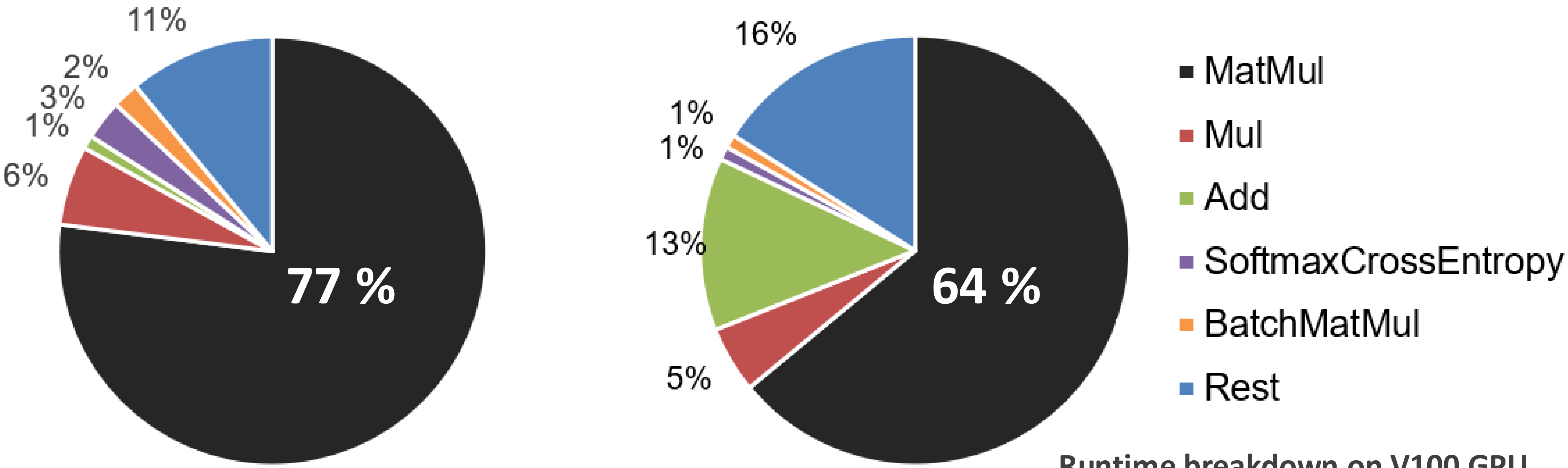
- Communication Mechanism → Topology-aware Collectives, Send/Recv, RPC
- Compute and Memory Design → Dataflow, Microarchitecture, Flexibility, Sparsity Support
- Communication Scheduling → LIFO, FIFO, Fusion
- Messaging/Transport Layer → TCP, RDMA (+ GPUDirect RDMA)

Network Layer

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

Key Compute Kernel during DL Training



Runtime breakdown on V100 GPU

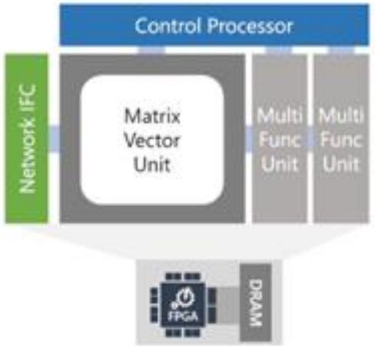
Transformer
(Language Understanding)

GNMT
(Machine Translation)

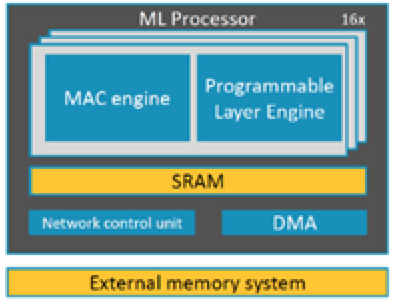
Matrix multiplications (GEMMs) consume around **70%** of the total runtime when training modern deep learning workloads.

Hardware for Accelerating GEMMs

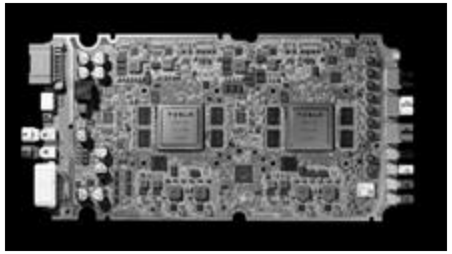
SIMD Architectures



Microsoft Brainwave

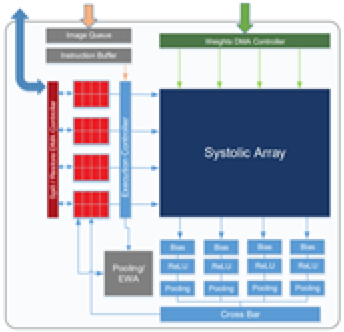


ARM Trillium

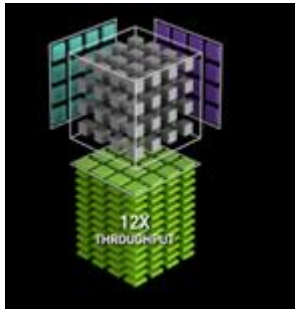


Tesla FSDC

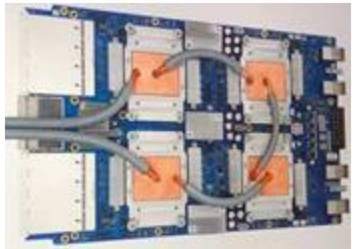
Systolic Architectures



Xilinx xDNN



Nvidia Tensor Cores

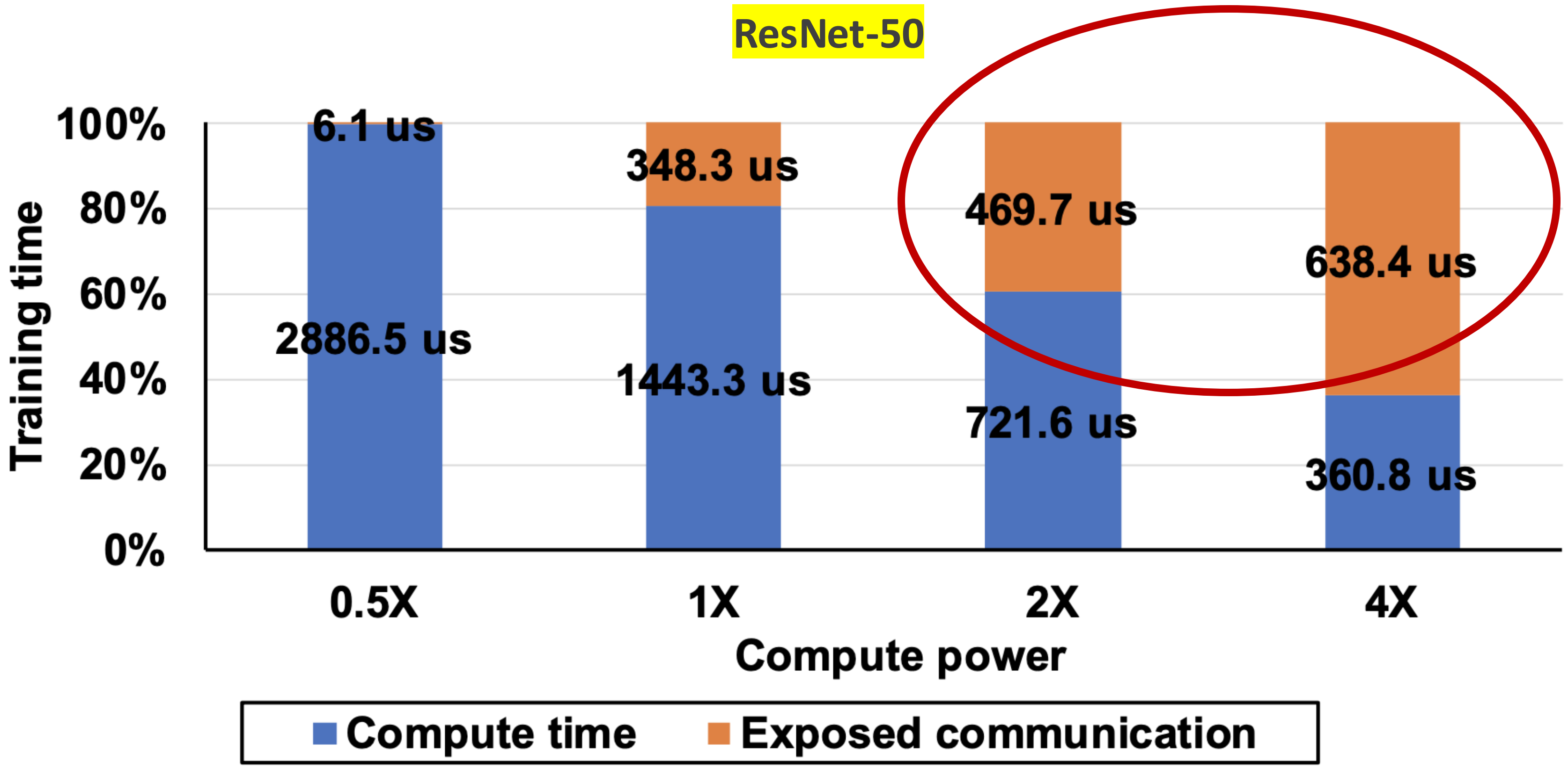


Google TPU

Key Feature:

- Specialized support for GEMMs
- Maximize HW TFLOPS

Effect of Enhanced Compute Efficiency on Training

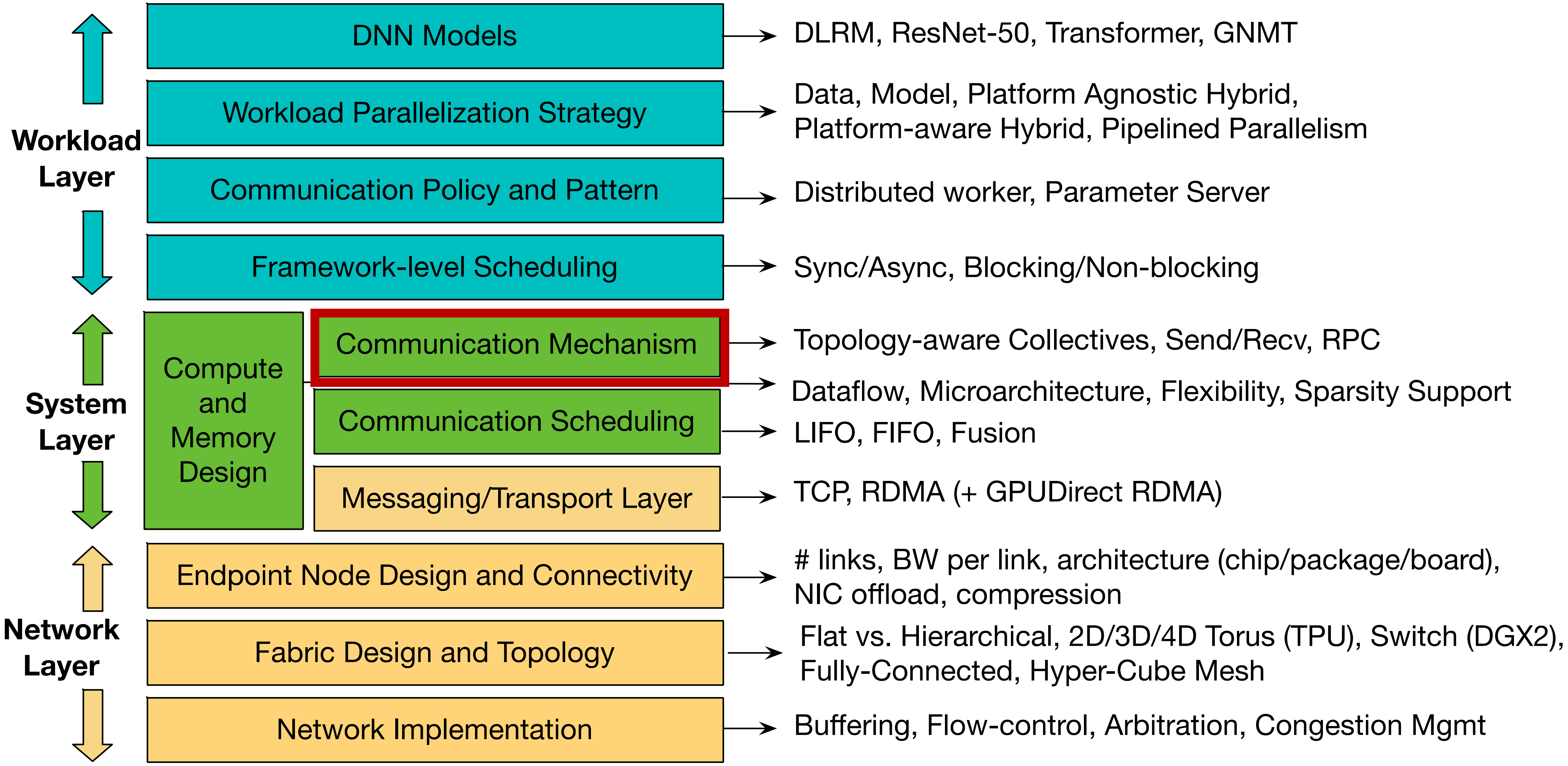


3D torus with total of 32 NPUs (2X4X4)

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

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 and Memory Design → Dataflow, Microarchitecture, Flexibility, Sparsity Support

Communication Scheduling → LIFO, FIFO, Fusion

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

Network Layer

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

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

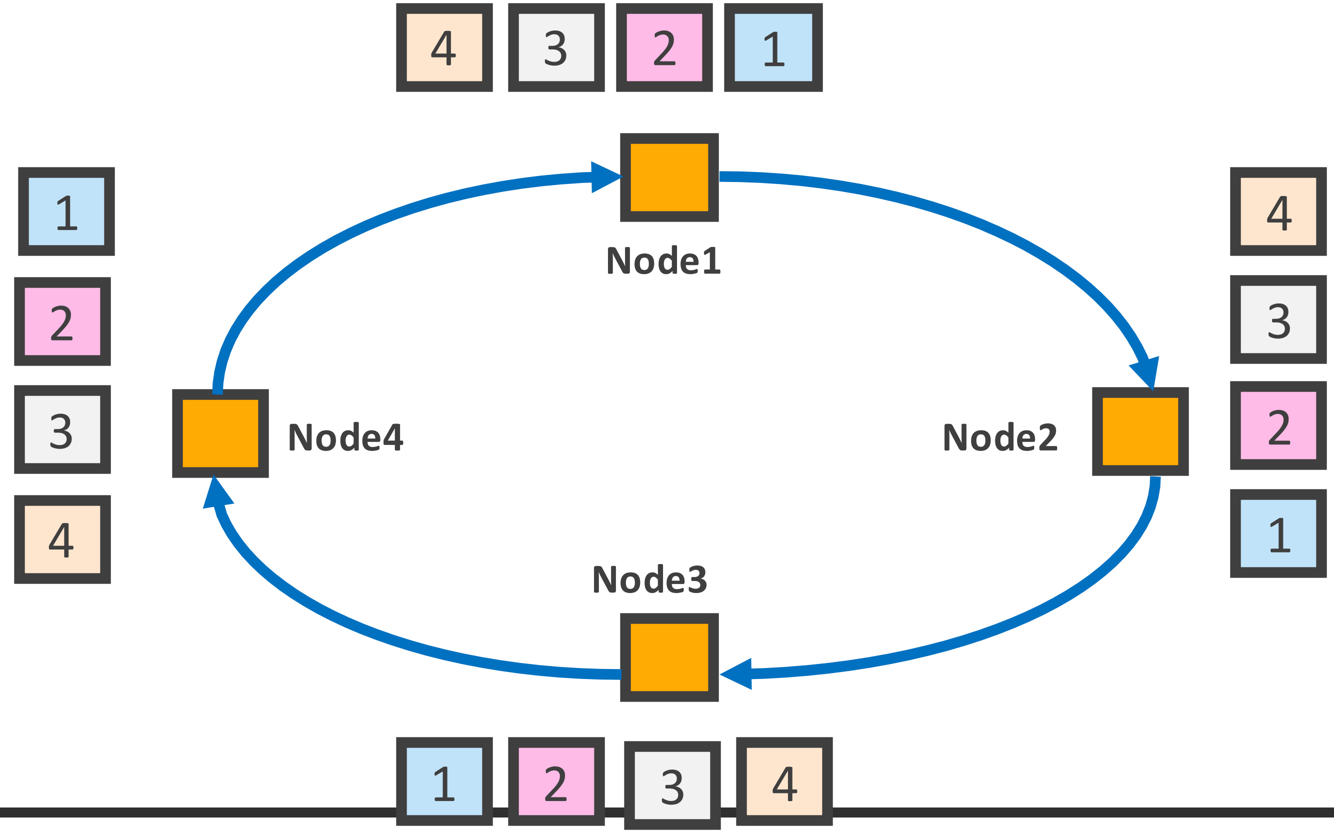
Figure Courtesy: Srinivas Sridharan (NVIDIA)

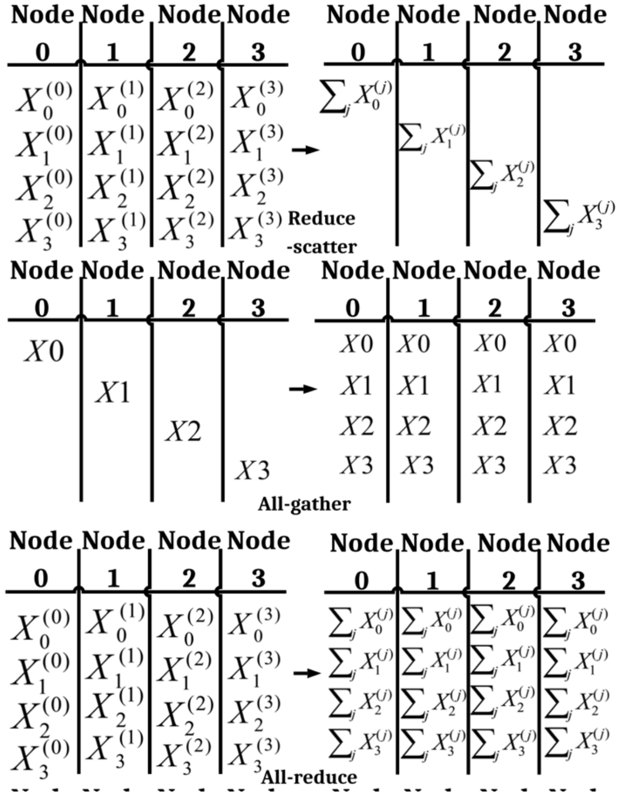
Topology-aware Collective Algorithms

- **Collective algorithm**: implementation of collectives
  - Collective communication libraries (CCLs, e.g., NCCL, RCCL, oneCCL) uses collective algorithms to run collective communications
- Example All-Reduce Algorithms:
  - Ring
  - Direct
  - Halving-Doubling
  - Rabenseifner
  - Double Binary Tree
  - etc.
- Given a network topology, an **efficient mechanism** to run collective communication exists
  - Called **topology-aware collective algorithms**

Example: Ring Based All-Reduce

- A ring with N nodes partitions data to N messages
- Collective Communication Flow:





Reduce-scatter

Node 0	Node 1	Node 2	Node 3
$X_0^{(0)}$	$X_0^{(1)}$	$X_0^{(2)}$	$X_0^{(3)}$
$X_1^{(0)}$	$X_1^{(1)}$	$X_1^{(2)}$	$X_1^{(3)}$
$X_2^{(0)}$	$X_2^{(1)}$	$X_2^{(2)}$	$X_2^{(3)}$
$X_3^{(0)}$	$X_3^{(1)}$	$X_3^{(2)}$	$X_3^{(3)}$

→

Node 0	Node 1	Node 2	Node 3
$\sum_j X_0^{(j)}$			
	$\sum_j X_1^{(j)}$		
		$\sum_j X_2^{(j)}$	
			$\sum_j X_3^{(j)}$

All-gather

Node 0	Node 1	Node 2	Node 3
$X0$			
	$X1$		
		$X2$	
			$X3$

→

Node 0	Node 1	Node 2	Node 3
$X0$	$X0$	$X0$	$X0$
$X1$	$X1$	$X1$	$X1$
$X2$	$X2$	$X2$	$X2$
$X3$	$X3$	$X3$	$X3$

All-reduce

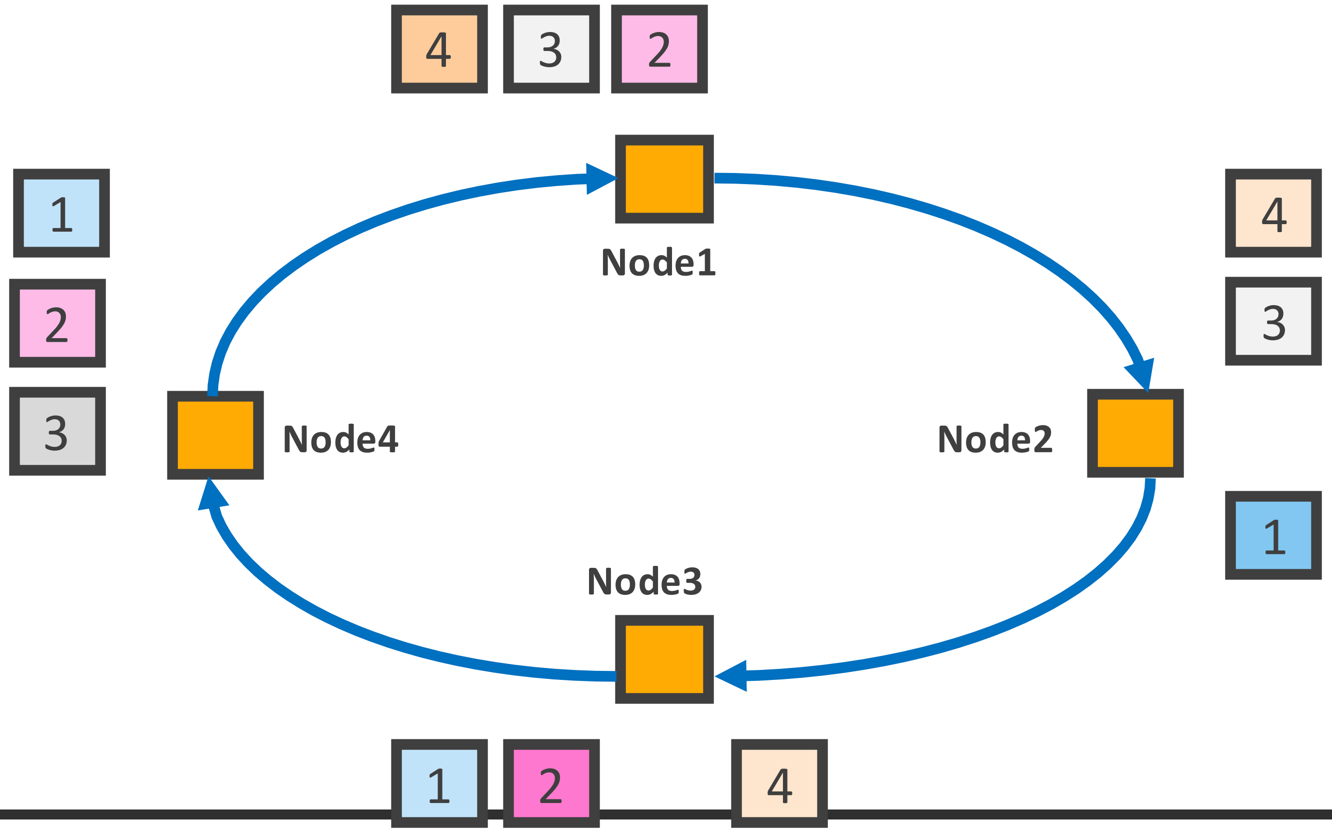
Node 0	Node 1	Node 2	Node 3
$X_0^{(0)}$	$X_0^{(1)}$	$X_0^{(2)}$	$X_0^{(3)}$
$X_1^{(0)}$	$X_1^{(1)}$	$X_1^{(2)}$	$X_1^{(3)}$
$X_2^{(0)}$	$X_2^{(1)}$	$X_2^{(2)}$	$X_2^{(3)}$
$X_3^{(0)}$	$X_3^{(1)}$	$X_3^{(2)}$	$X_3^{(3)}$

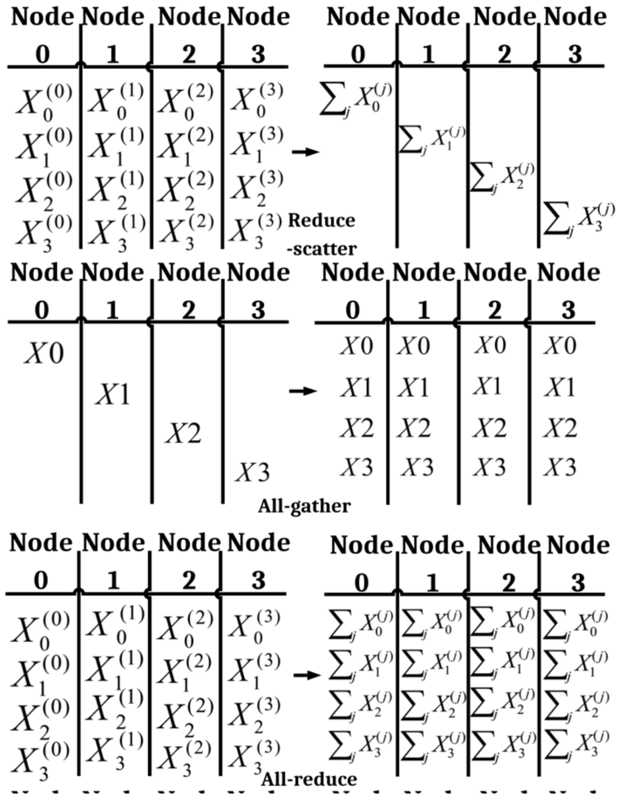
→

Node 0	Node 1	Node 2	Node 3
$\sum_j X_0^{(j)}$	$\sum_j X_0^{(j)}$	$\sum_j X_0^{(j)}$	$\sum_j X_0^{(j)}$
$\sum_j X_1^{(j)}$	$\sum_j X_1^{(j)}$	$\sum_j X_1^{(j)}$	$\sum_j X_1^{(j)}$
$\sum_j X_2^{(j)}$	$\sum_j X_2^{(j)}$	$\sum_j X_2^{(j)}$	$\sum_j X_2^{(j)}$
$\sum_j X_3^{(j)}$	$\sum_j X_3^{(j)}$	$\sum_j X_3^{(j)}$	$\sum_j X_3^{(j)}$

Example: Ring Based All-Reduce

- A ring with N nodes partitions data to N messages
- Collective Communication Flow:





Node 0	Node 1	Node 2	Node 3
$X_0^{(0)}$	$X_0^{(1)}$	$X_0^{(2)}$	$X_0^{(3)}$
$X_1^{(0)}$	$X_1^{(1)}$	$X_1^{(2)}$	$X_1^{(3)}$
$X_2^{(0)}$	$X_2^{(1)}$	$X_2^{(2)}$	$X_2^{(3)}$
$X_3^{(0)}$	$X_3^{(1)}$	$X_3^{(2)}$	$X_3^{(3)}$

→ Reduce-scatter

Node 0	Node 1	Node 2	Node 3
$\sum_j X_0^{(j)}$			
	$\sum_j X_1^{(j)}$		
		$\sum_j X_2^{(j)}$	
			$\sum_j X_3^{(j)}$

Node 0	Node 1	Node 2	Node 3
$X0$			
	$X1$		
		$X2$	
			$X3$

→ All-gather

Node 0	Node 1	Node 2	Node 3
$X0$	$X0$	$X0$	$X0$
$X1$	$X1$	$X1$	$X1$
$X2$	$X2$	$X2$	$X2$
$X3$	$X3$	$X3$	$X3$

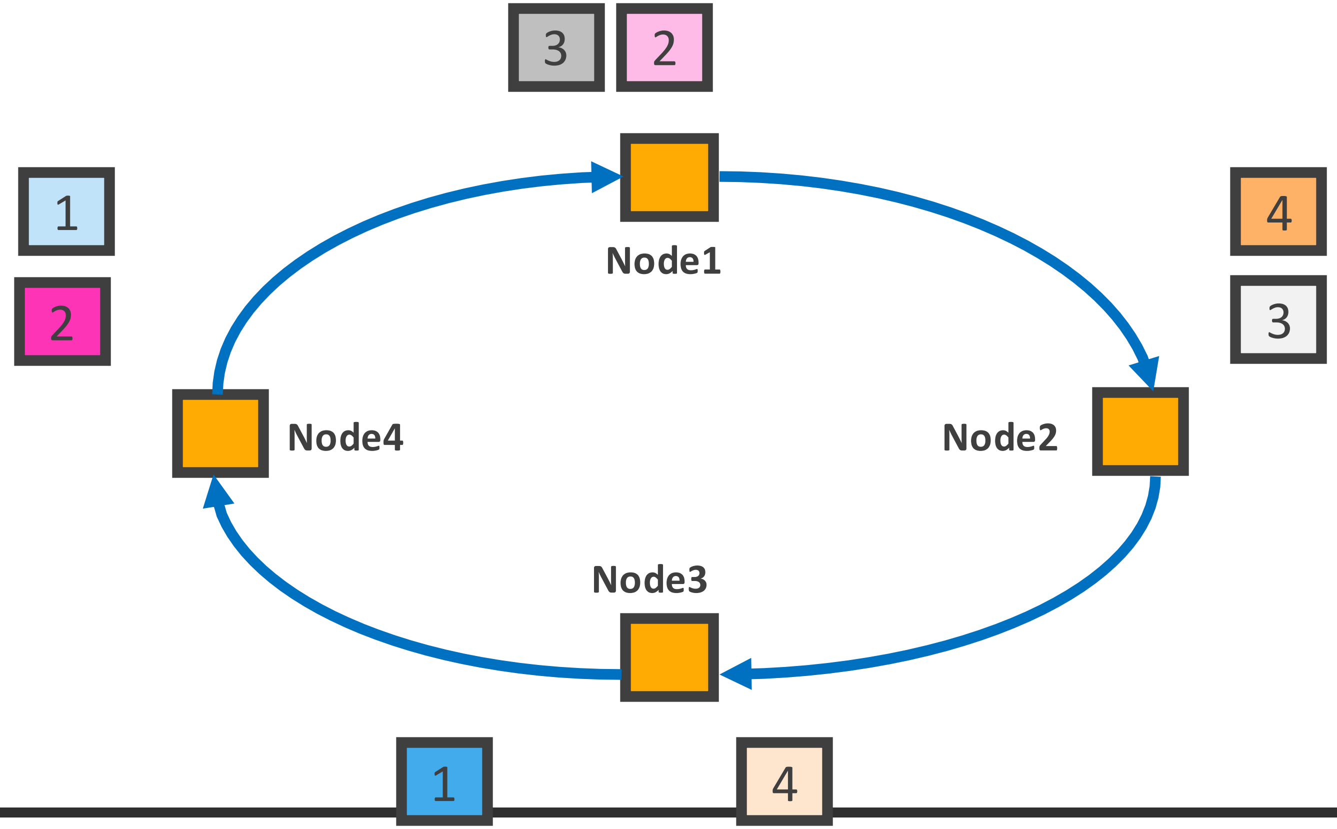
Node 0	Node 1	Node 2	Node 3
$X_0^{(0)}$	$X_0^{(1)}$	$X_0^{(2)}$	$X_0^{(3)}$
$X_1^{(0)}$	$X_1^{(1)}$	$X_1^{(2)}$	$X_1^{(3)}$
$X_2^{(0)}$	$X_2^{(1)}$	$X_2^{(2)}$	$X_2^{(3)}$
$X_3^{(0)}$	$X_3^{(1)}$	$X_3^{(2)}$	$X_3^{(3)}$

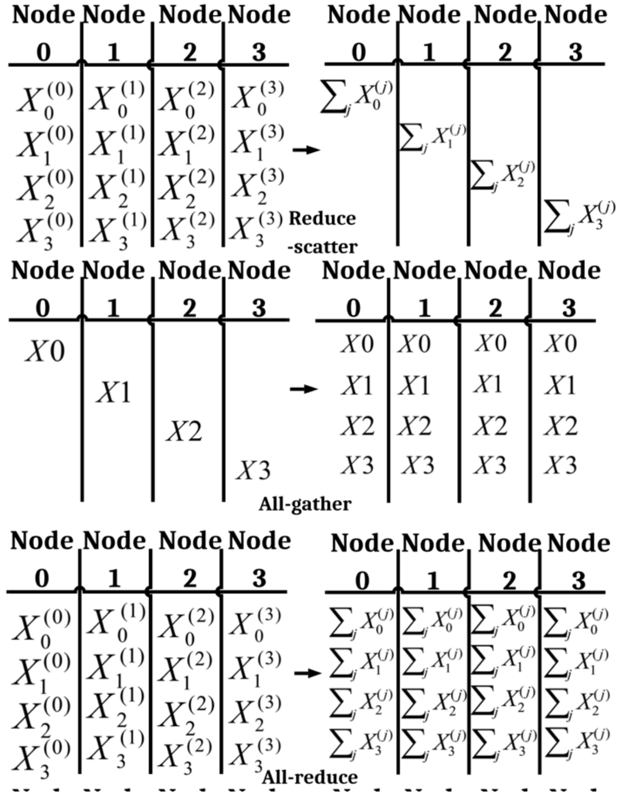
→ All-reduce

Node 0	Node 1	Node 2	Node 3
$\sum_j X_0^{(j)}$	$\sum_j X_0^{(j)}$	$\sum_j X_0^{(j)}$	$\sum_j X_0^{(j)}$
$\sum_j X_1^{(j)}$	$\sum_j X_1^{(j)}$	$\sum_j X_1^{(j)}$	$\sum_j X_1^{(j)}$
$\sum_j X_2^{(j)}$	$\sum_j X_2^{(j)}$	$\sum_j X_2^{(j)}$	$\sum_j X_2^{(j)}$
$\sum_j X_3^{(j)}$	$\sum_j X_3^{(j)}$	$\sum_j X_3^{(j)}$	$\sum_j X_3^{(j)}$

Example: Ring Based All-Reduce

- A ring with N nodes partitions data to N messages
- Collective Communication Flow:





Node 0	Node 1	Node 2	Node 3
$X_0^{(0)}$	$X_0^{(1)}$	$X_0^{(2)}$	$X_0^{(3)}$
$X_1^{(0)}$	$X_1^{(1)}$	$X_1^{(2)}$	$X_1^{(3)}$
$X_2^{(0)}$	$X_2^{(1)}$	$X_2^{(2)}$	$X_2^{(3)}$
$X_3^{(0)}$	$X_3^{(1)}$	$X_3^{(2)}$	$X_3^{(3)}$

→ Reduce-scatter

Node 0	Node 1	Node 2	Node 3
$\sum_j X_0^{(j)}$			
	$\sum_j X_1^{(j)}$		
		$\sum_j X_2^{(j)}$	
			$\sum_j X_3^{(j)}$

Node 0	Node 1	Node 2	Node 3
$X0$			
	$X1$		
		$X2$	
			$X3$

→ All-gather

Node 0	Node 1	Node 2	Node 3
$X0$	$X0$	$X0$	$X0$
$X1$	$X1$	$X1$	$X1$
$X2$	$X2$	$X2$	$X2$
$X3$	$X3$	$X3$	$X3$

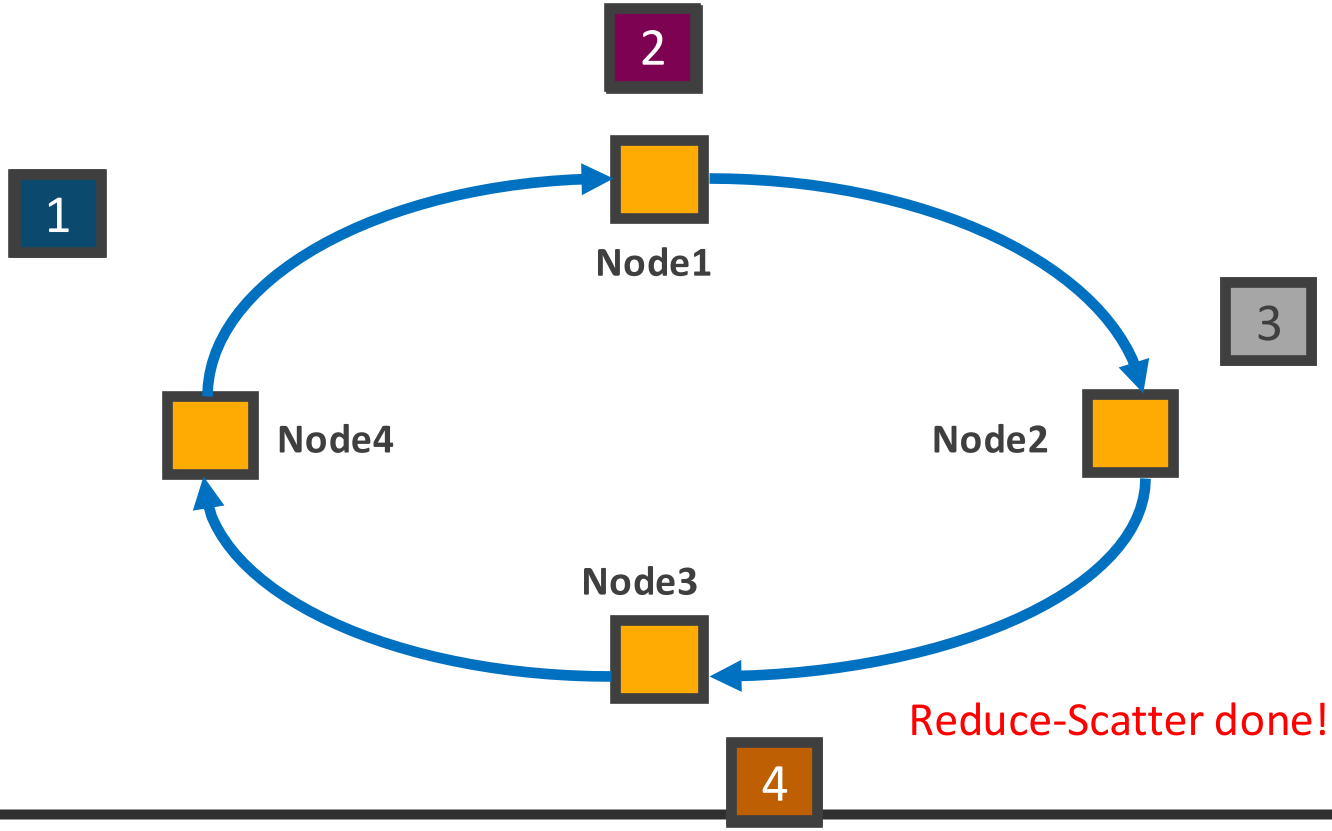
Node 0	Node 1	Node 2	Node 3
$X_0^{(0)}$	$X_0^{(1)}$	$X_0^{(2)}$	$X_0^{(3)}$
$X_1^{(0)}$	$X_1^{(1)}$	$X_1^{(2)}$	$X_1^{(3)}$
$X_2^{(0)}$	$X_2^{(1)}$	$X_2^{(2)}$	$X_2^{(3)}$
$X_3^{(0)}$	$X_3^{(1)}$	$X_3^{(2)}$	$X_3^{(3)}$

→ All-reduce

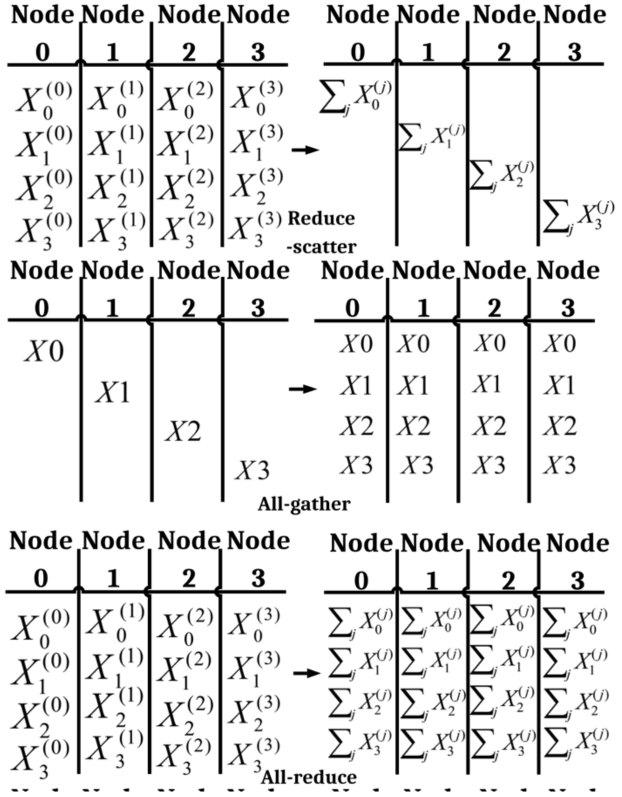
Node 0	Node 1	Node 2	Node 3
$\sum_j X_0^{(j)}$	$\sum_j X_0^{(j)}$	$\sum_j X_0^{(j)}$	$\sum_j X_0^{(j)}$
$\sum_j X_1^{(j)}$	$\sum_j X_1^{(j)}$	$\sum_j X_1^{(j)}$	$\sum_j X_1^{(j)}$
$\sum_j X_2^{(j)}$	$\sum_j X_2^{(j)}$	$\sum_j X_2^{(j)}$	$\sum_j X_2^{(j)}$
$\sum_j X_3^{(j)}$	$\sum_j X_3^{(j)}$	$\sum_j X_3^{(j)}$	$\sum_j X_3^{(j)}$

Example: Ring Based All-Reduce

- A ring with N nodes partitions data to N messages
- Collective Communication Flow:



Reduce-Scatter done!



Node 0	Node 1	Node 2	Node 3
$X_0^{(0)}$	$X_0^{(1)}$	$X_0^{(2)}$	$X_0^{(3)}$
$X_1^{(0)}$	$X_1^{(1)}$	$X_1^{(2)}$	$X_1^{(3)}$
$X_2^{(0)}$	$X_2^{(1)}$	$X_2^{(2)}$	$X_2^{(3)}$
$X_3^{(0)}$	$X_3^{(1)}$	$X_3^{(2)}$	$X_3^{(3)}$

→

Node 0	Node 1	Node 2	Node 3
$\sum_j X_0^{(j)}$			
	$\sum_j X_1^{(j)}$		
		$\sum_j X_2^{(j)}$	
			$\sum_j X_3^{(j)}$

Reduce -scatter

Node 0	Node 1	Node 2	Node 3
$X0$			
	$X1$		
		$X2$	
			$X3$

→

Node 0	Node 1	Node 2	Node 3
$X0$	$X0$	$X0$	$X0$
$X1$	$X1$	$X1$	$X1$
$X2$	$X2$	$X2$	$X2$
$X3$	$X3$	$X3$	$X3$

All-gather

Node 0	Node 1	Node 2	Node 3
$X_0^{(0)}$	$X_0^{(1)}$	$X_0^{(2)}$	$X_0^{(3)}$
$X_1^{(0)}$	$X_1^{(1)}$	$X_1^{(2)}$	$X_1^{(3)}$
$X_2^{(0)}$	$X_2^{(1)}$	$X_2^{(2)}$	$X_2^{(3)}$
$X_3^{(0)}$	$X_3^{(1)}$	$X_3^{(2)}$	$X_3^{(3)}$

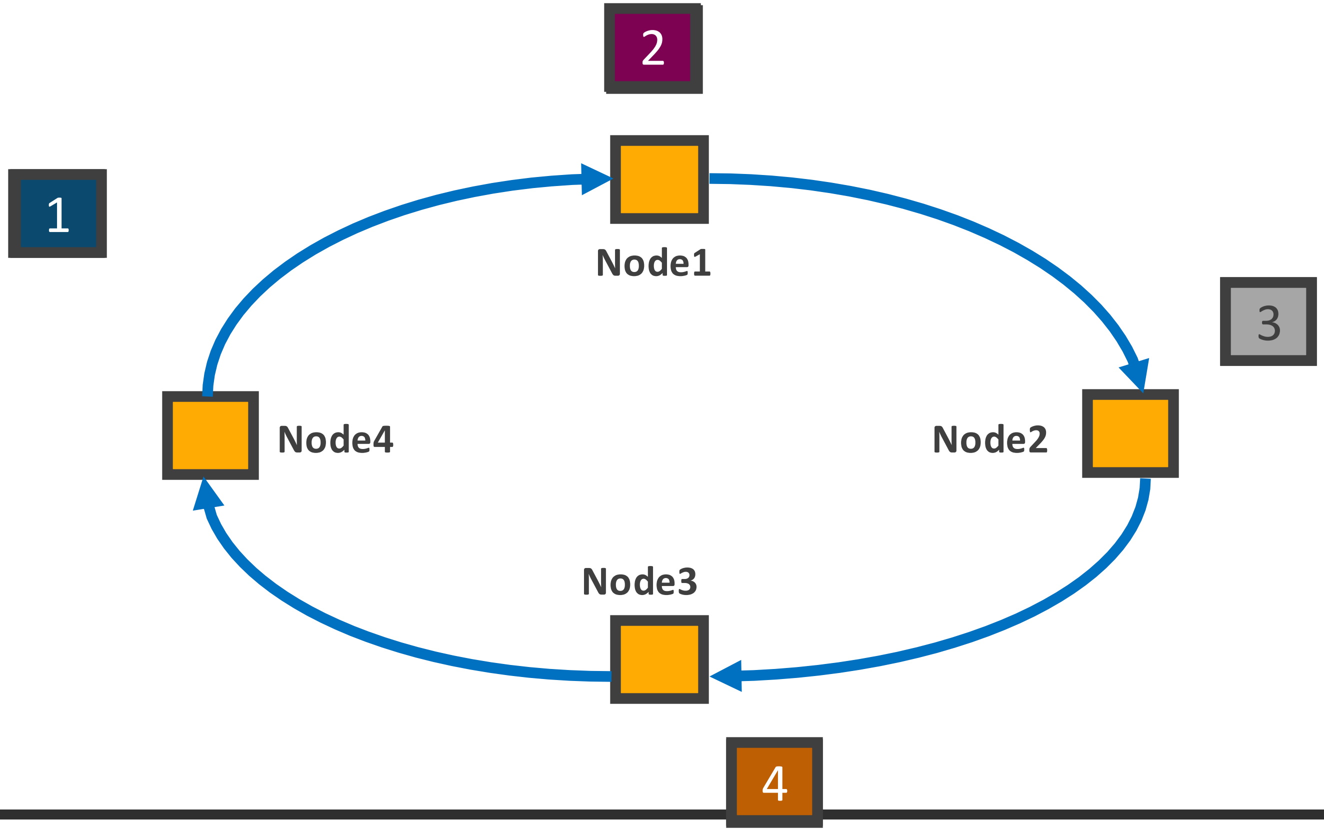
→

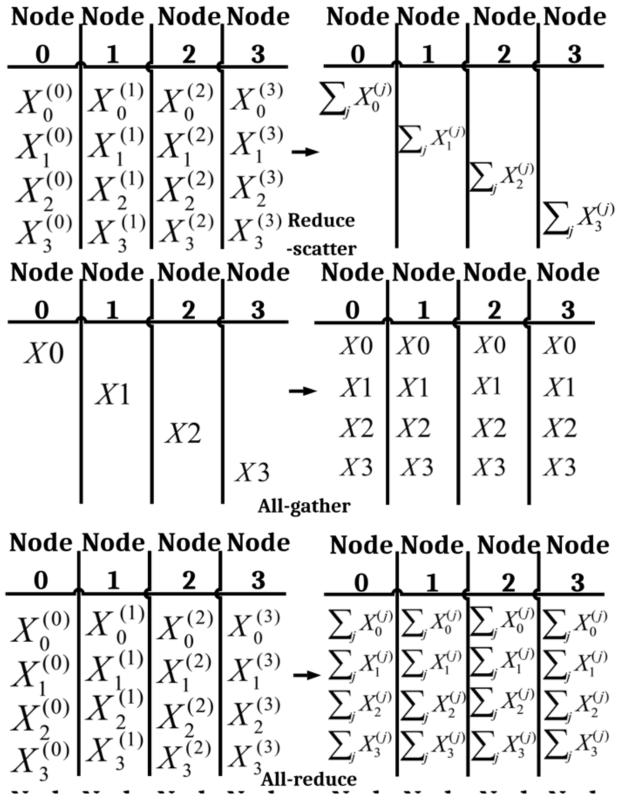
Node 0	Node 1	Node 2	Node 3
$\sum_j X_0^{(j)}$	$\sum_j X_0^{(j)}$	$\sum_j X_0^{(j)}$	$\sum_j X_0^{(j)}$
$\sum_j X_1^{(j)}$	$\sum_j X_1^{(j)}$	$\sum_j X_1^{(j)}$	$\sum_j X_1^{(j)}$
$\sum_j X_2^{(j)}$	$\sum_j X_2^{(j)}$	$\sum_j X_2^{(j)}$	$\sum_j X_2^{(j)}$
$\sum_j X_3^{(j)}$	$\sum_j X_3^{(j)}$	$\sum_j X_3^{(j)}$	$\sum_j X_3^{(j)}$

All-reduce

Example: Ring Based All-Reduce

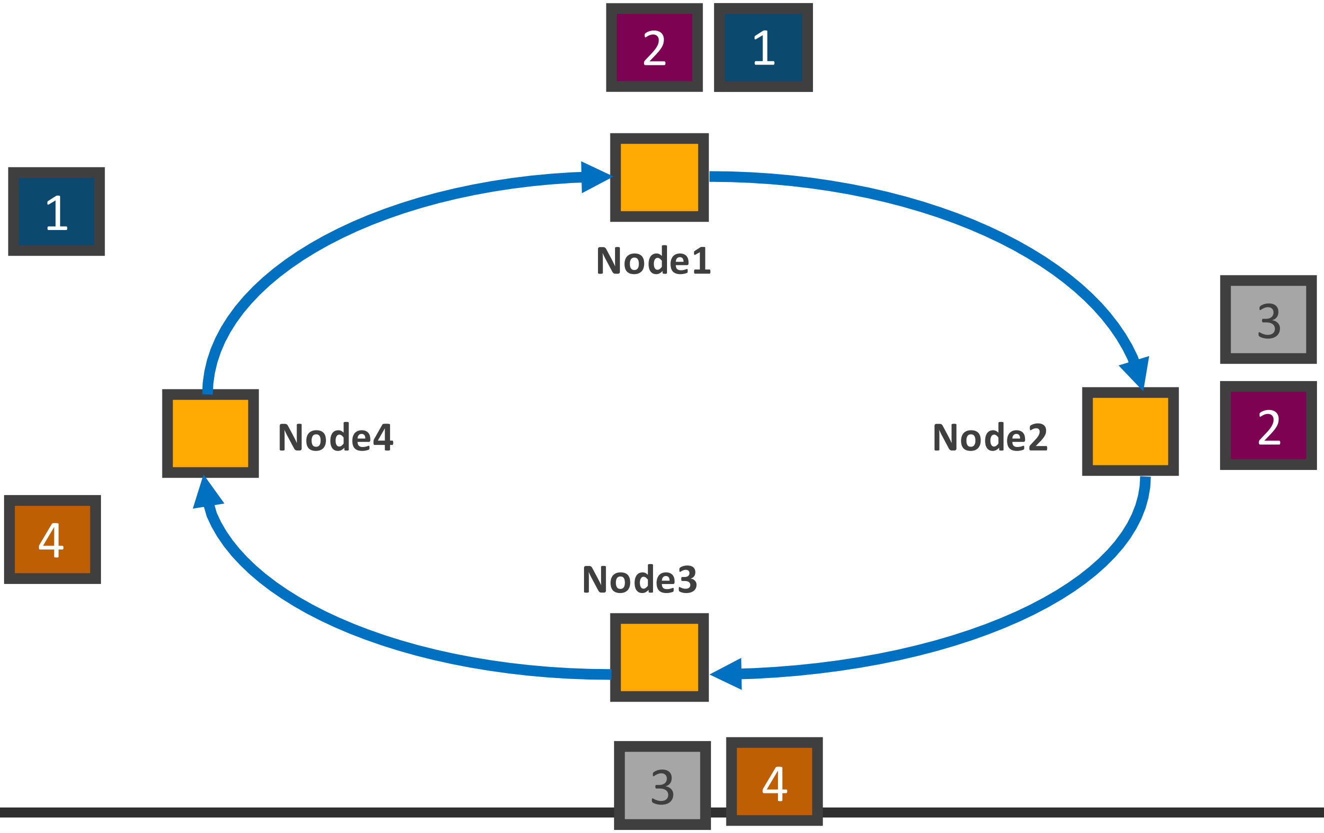
- A ring with N nodes partitions data to N messages
- Collective Communication Flow:

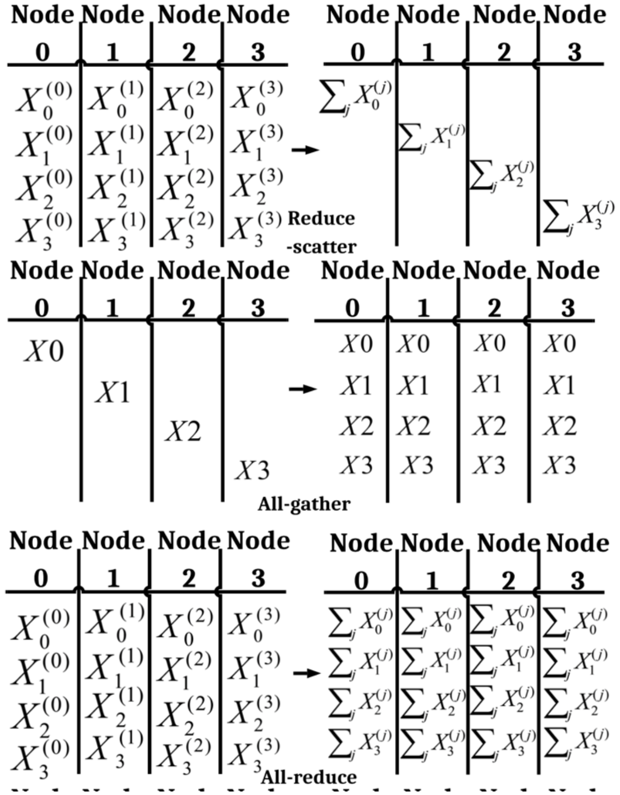




Example: Ring Based All-Reduce

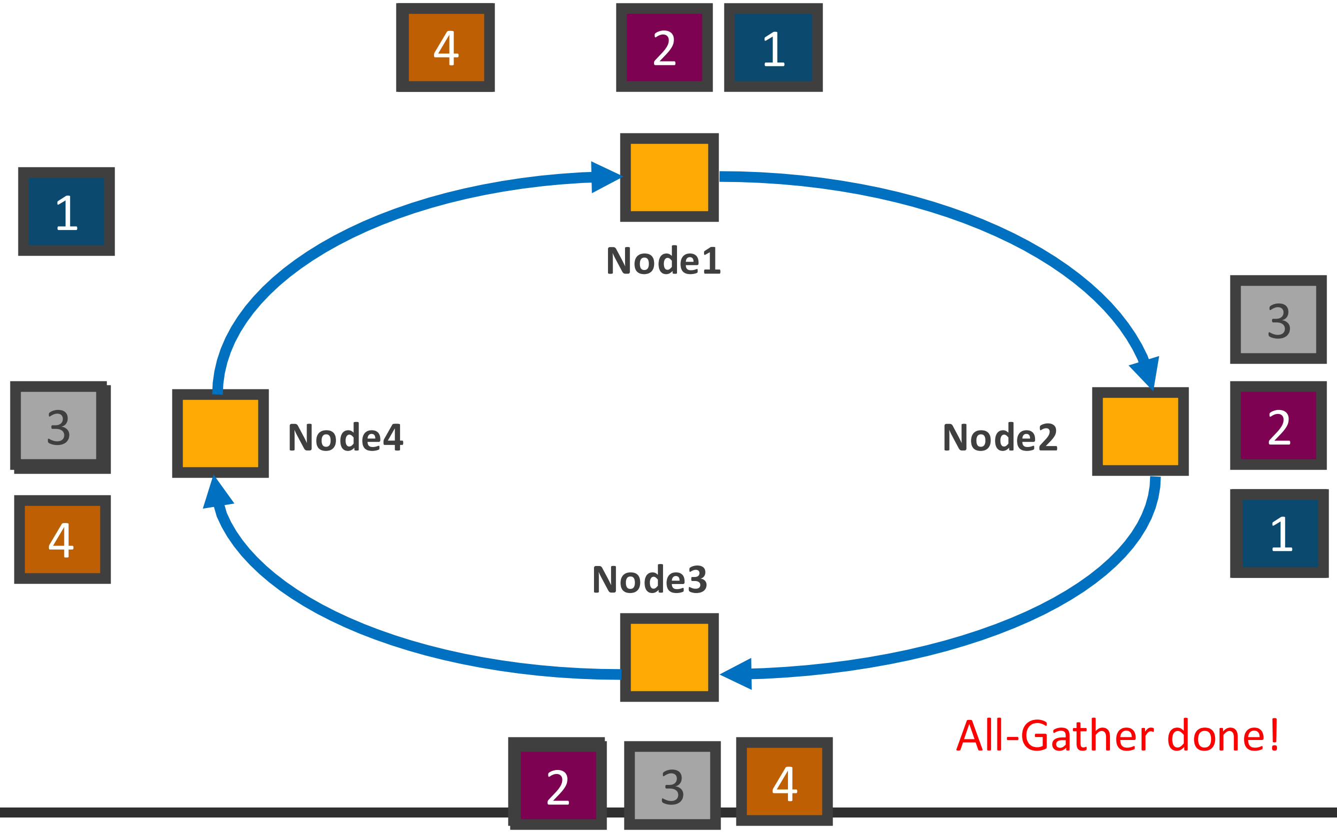
- A ring with N nodes partitions data to N messages
- Collective Communication Flow:



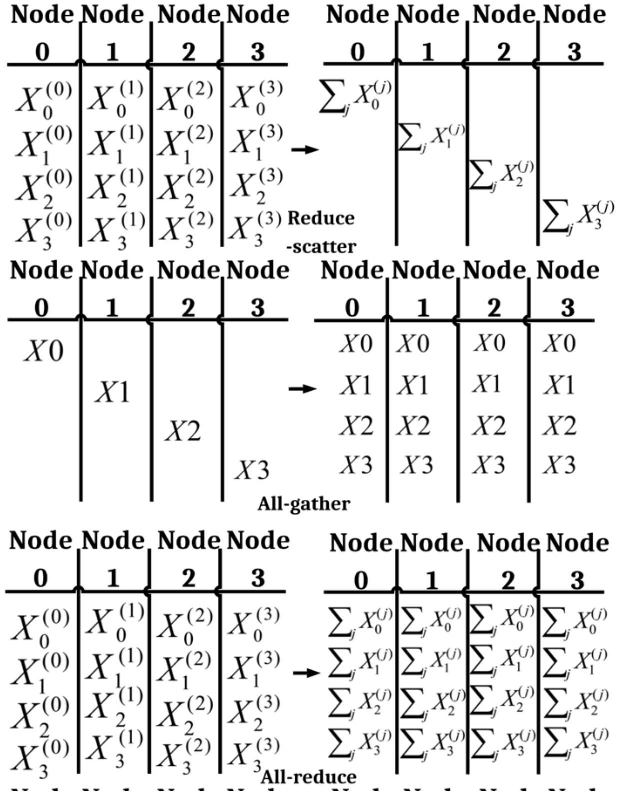


Example: Ring Based All-Reduce

- A ring with N nodes partitions data to N messages
- Collective Communication Flow:



All-Gather done!



Node 0	Node 1	Node 2	Node 3
$X_0^{(0)}$	$X_0^{(1)}$	$X_0^{(2)}$	$X_0^{(3)}$
$X_1^{(0)}$	$X_1^{(1)}$	$X_1^{(2)}$	$X_1^{(3)}$
$X_2^{(0)}$	$X_2^{(1)}$	$X_2^{(2)}$	$X_2^{(3)}$
$X_3^{(0)}$	$X_3^{(1)}$	$X_3^{(2)}$	$X_3^{(3)}$

→ Reduce-scatter

Node 0	Node 1	Node 2	Node 3
$\sum_j X_0^{(j)}$			
	$\sum_j X_1^{(j)}$		
		$\sum_j X_2^{(j)}$	
			$\sum_j X_3^{(j)}$

Node 0	Node 1	Node 2	Node 3
$X0$			
	$X1$		
		$X2$	
			$X3$

→ All-gather

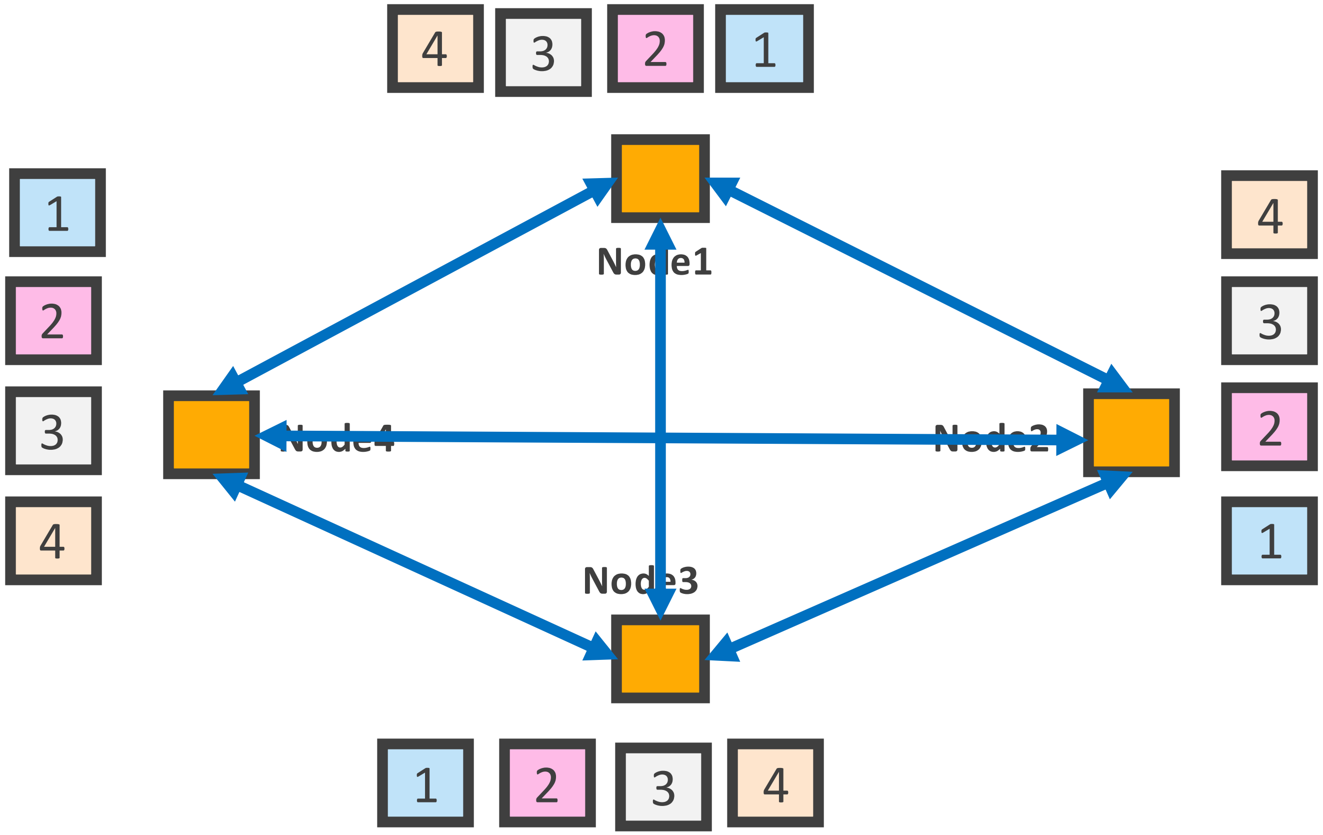
Node 0	Node 1	Node 2	Node 3
$X0$	$X0$	$X0$	$X0$
$X1$	$X1$	$X1$	$X1$
$X2$	$X2$	$X2$	$X2$
$X3$	$X3$	$X3$	$X3$

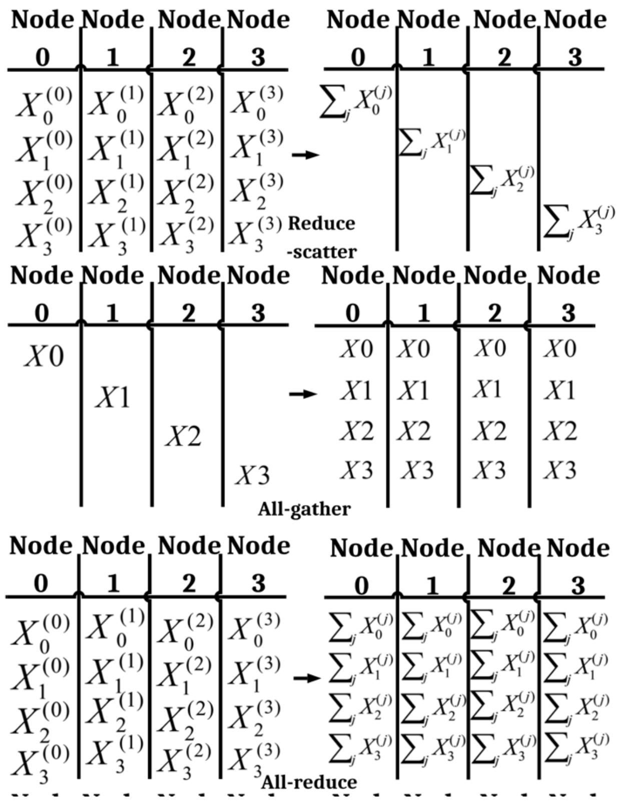
Node 0	Node 1	Node 2	Node 3
$X_0^{(0)}$	$X_0^{(1)}$	$X_0^{(2)}$	$X_0^{(3)}$
$X_1^{(0)}$	$X_1^{(1)}$	$X_1^{(2)}$	$X_1^{(3)}$
$X_2^{(0)}$	$X_2^{(1)}$	$X_2^{(2)}$	$X_2^{(3)}$
$X_3^{(0)}$	$X_3^{(1)}$	$X_3^{(2)}$	$X_3^{(3)}$

→ All-reduce

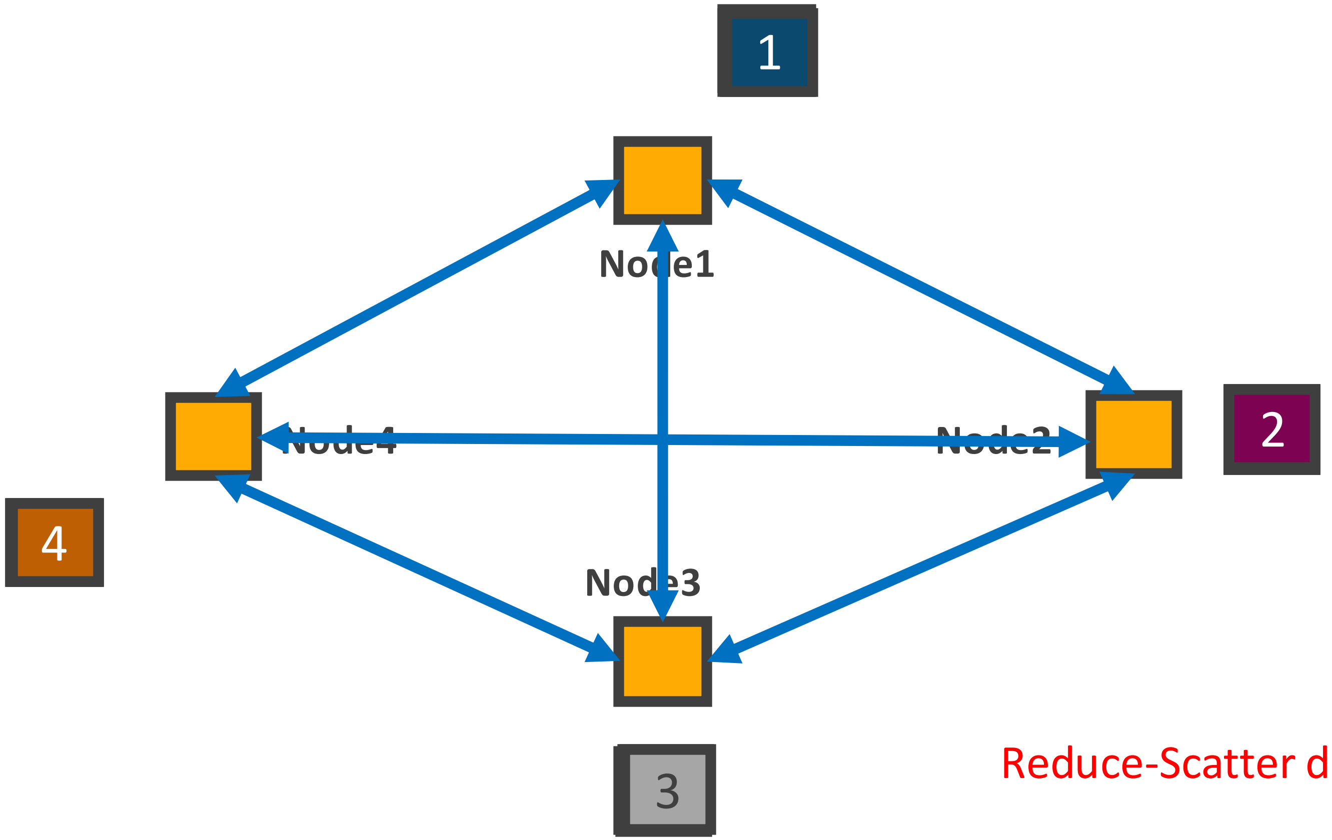
Node 0	Node 1	Node 2	Node 3
$\sum_j X_0^{(j)}$	$\sum_j X_0^{(j)}$	$\sum_j X_0^{(j)}$	$\sum_j X_0^{(j)}$
$\sum_j X_1^{(j)}$	$\sum_j X_1^{(j)}$	$\sum_j X_1^{(j)}$	$\sum_j X_1^{(j)}$
$\sum_j X_2^{(j)}$	$\sum_j X_2^{(j)}$	$\sum_j X_2^{(j)}$	$\sum_j X_2^{(j)}$
$\sum_j X_3^{(j)}$	$\sum_j X_3^{(j)}$	$\sum_j X_3^{(j)}$	$\sum_j X_3^{(j)}$

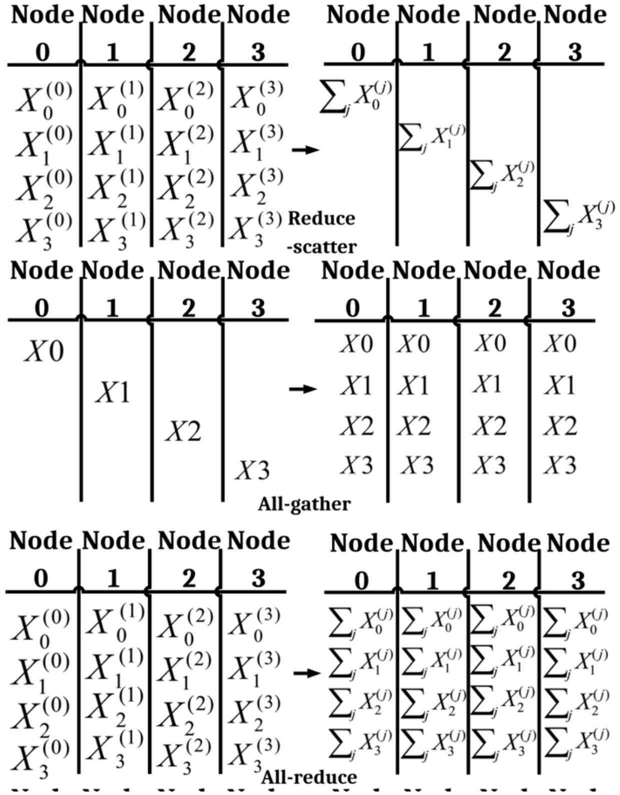
Example: Direct All-Reduce




Node 0
Node 1
Node 2
Node 3
$X_0^{(0)}$ $X_0^{(1)}$ $X_0^{(2)}$ $X_0^{(3)}$
$X_1^{(0)}$ $X_1^{(1)}$ $X_1^{(2)}$ $X_1^{(3)}$
$X_2^{(0)}$ $X_2^{(1)}$ $X_2^{(2)}$ $X_2^{(3)}$
$X_3^{(0)}$ $X_3^{(1)}$ $X_3^{(2)}$ $X_3^{(3)}$
Reduce
-scatter
$\sum_j X_0^{(j)}$
$\sum_j X_1^{(j)}$
$\sum_j X_2^{(j)}$
$\sum_j X_3^{(j)}$
$X0$
$X1$
$X2$
$X3$
All-gather
All-reduce

Example: Direct All-Reduce





Node 0	Node 1	Node 2	Node 3
$X_0^{(0)}$	$X_0^{(1)}$	$X_0^{(2)}$	$X_0^{(3)}$
$X_1^{(0)}$	$X_1^{(1)}$	$X_1^{(2)}$	$X_1^{(3)}$
$X_2^{(0)}$	$X_2^{(1)}$	$X_2^{(2)}$	$X_2^{(3)}$
$X_3^{(0)}$	$X_3^{(1)}$	$X_3^{(2)}$	$X_3^{(3)}$

→ Reduce -scatter

Node 0	Node 1	Node 2	Node 3
$\sum_j X_0^{(j)}$			
	$\sum_j X_1^{(j)}$		
		$\sum_j X_2^{(j)}$	
			$\sum_j X_3^{(j)}$

Node 0	Node 1	Node 2	Node 3
$X0$			
	$X1$		
		$X2$	
			$X3$

→ All-gather

Node 0	Node 1	Node 2	Node 3
$X0$	$X0$	$X0$	$X0$
$X1$	$X1$	$X1$	$X1$
$X2$	$X2$	$X2$	$X2$
$X3$	$X3$	$X3$	$X3$

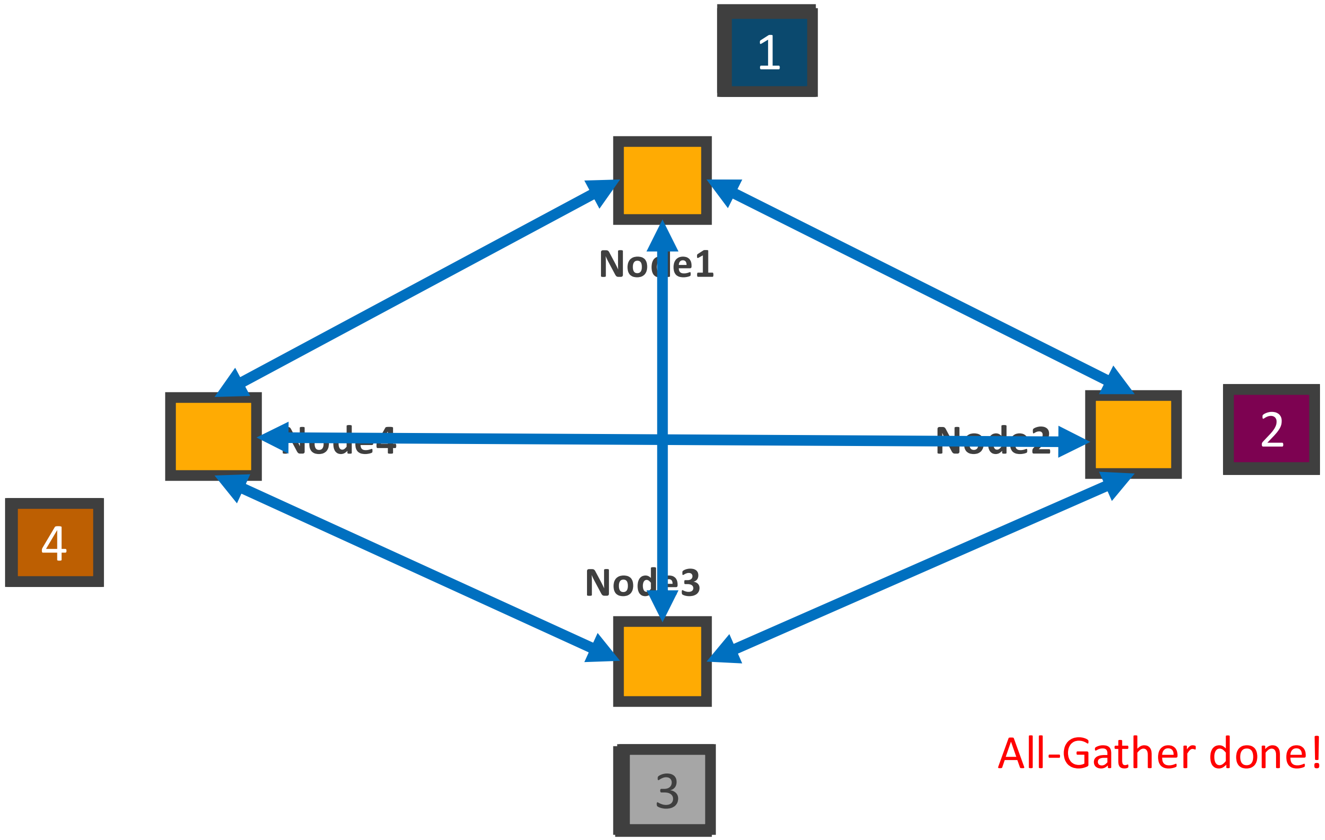
Node 0	Node 1	Node 2	Node 3
$X_0^{(0)}$	$X_0^{(1)}$	$X_0^{(2)}$	$X_0^{(3)}$
$X_1^{(0)}$	$X_1^{(1)}$	$X_1^{(2)}$	$X_1^{(3)}$
$X_2^{(0)}$	$X_2^{(1)}$	$X_2^{(2)}$	$X_2^{(3)}$
$X_3^{(0)}$	$X_3^{(1)}$	$X_3^{(2)}$	$X_3^{(3)}$

→ All-reduce

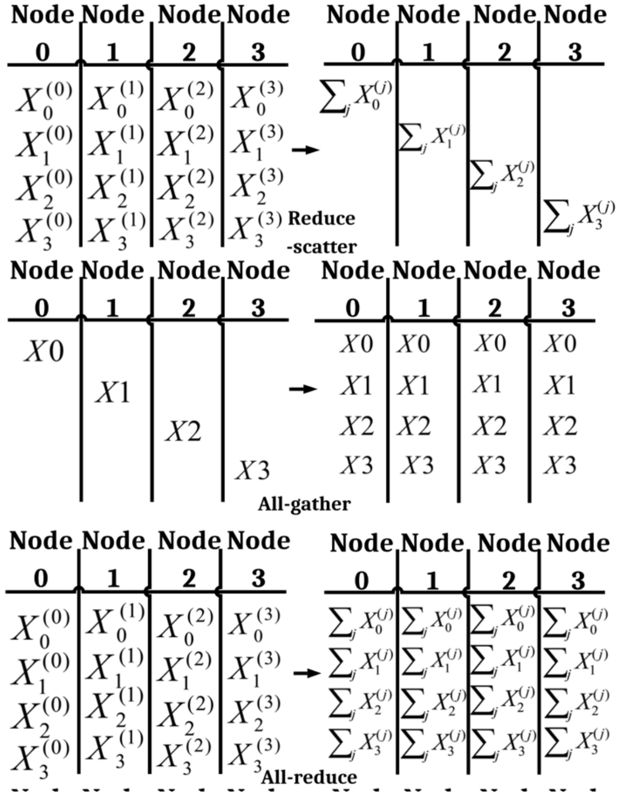
Node 0	Node 1	Node 2	Node 3
$\sum_j X_0^{(j)}$	$\sum_j X_0^{(j)}$	$\sum_j X_0^{(j)}$	$\sum_j X_0^{(j)}$
$\sum_j X_1^{(j)}$	$\sum_j X_1^{(j)}$	$\sum_j X_1^{(j)}$	$\sum_j X_1^{(j)}$
$\sum_j X_2^{(j)}$	$\sum_j X_2^{(j)}$	$\sum_j X_2^{(j)}$	$\sum_j X_2^{(j)}$
$\sum_j X_3^{(j)}$	$\sum_j X_3^{(j)}$	$\sum_j X_3^{(j)}$	$\sum_j X_3^{(j)}$

Reduce-Scatter done!

Example: Direct All-Reduce



All-Gather done!



Node 0	Node 1	Node 2	Node 3
$X_0^{(0)}$	$X_0^{(1)}$	$X_0^{(2)}$	$X_0^{(3)}$
$X_1^{(0)}$	$X_1^{(1)}$	$X_1^{(2)}$	$X_1^{(3)}$
$X_2^{(0)}$	$X_2^{(1)}$	$X_2^{(2)}$	$X_2^{(3)}$
$X_3^{(0)}$	$X_3^{(1)}$	$X_3^{(2)}$	$X_3^{(3)}$

→ Reduce -scatter

Node 0	Node 1	Node 2	Node 3
$\sum_j X_0^{(j)}$			
	$\sum_j X_1^{(j)}$		
		$\sum_j X_2^{(j)}$	
			$\sum_j X_3^{(j)}$

Node 0	Node 1	Node 2	Node 3
$X0$			
	$X1$		
		$X2$	
			$X3$

→ All-gather

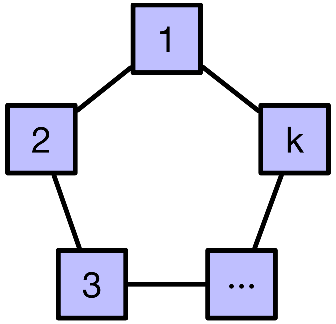
Node 0	Node 1	Node 2	Node 3
$X0$	$X0$	$X0$	$X0$
$X1$	$X1$	$X1$	$X1$
$X2$	$X2$	$X2$	$X2$
$X3$	$X3$	$X3$	$X3$

Node 0	Node 1	Node 2	Node 3
$X_0^{(0)}$	$X_0^{(1)}$	$X_0^{(2)}$	$X_0^{(3)}$
$X_1^{(0)}$	$X_1^{(1)}$	$X_1^{(2)}$	$X_1^{(3)}$
$X_2^{(0)}$	$X_2^{(1)}$	$X_2^{(2)}$	$X_2^{(3)}$
$X_3^{(0)}$	$X_3^{(1)}$	$X_3^{(2)}$	$X_3^{(3)}$

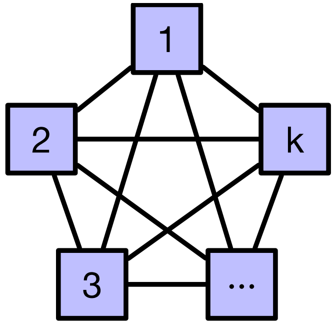
→ All-reduce

Node 0	Node 1	Node 2	Node 3
$\sum_j X_0^{(j)}$	$\sum_j X_0^{(j)}$	$\sum_j X_0^{(j)}$	$\sum_j X_0^{(j)}$
$\sum_j X_1^{(j)}$	$\sum_j X_1^{(j)}$	$\sum_j X_1^{(j)}$	$\sum_j X_1^{(j)}$
$\sum_j X_2^{(j)}$	$\sum_j X_2^{(j)}$	$\sum_j X_2^{(j)}$	$\sum_j X_2^{(j)}$
$\sum_j X_3^{(j)}$	$\sum_j X_3^{(j)}$	$\sum_j X_3^{(j)}$	$\sum_j X_3^{(j)}$

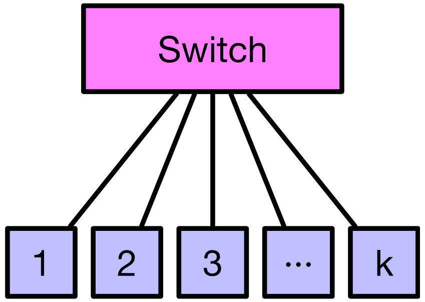
Topology-aware Collectives



(a) Ring(k)

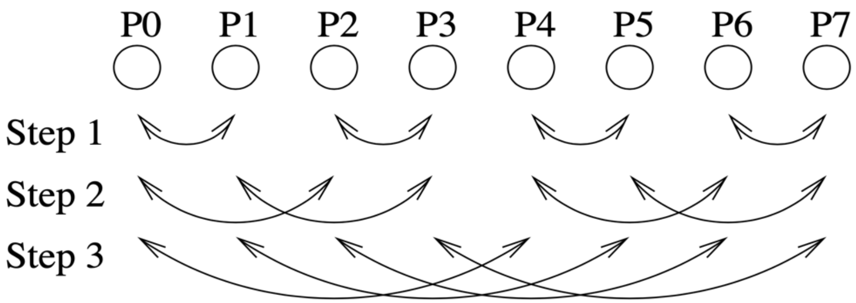


(b) FullyConnected(k)

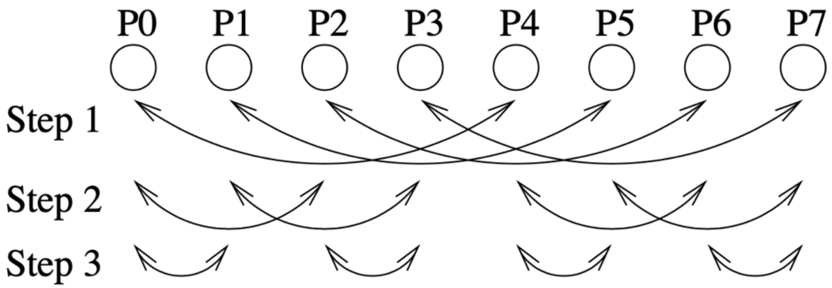


(b) Switch(k)

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



a) Reduce-Scatter phases

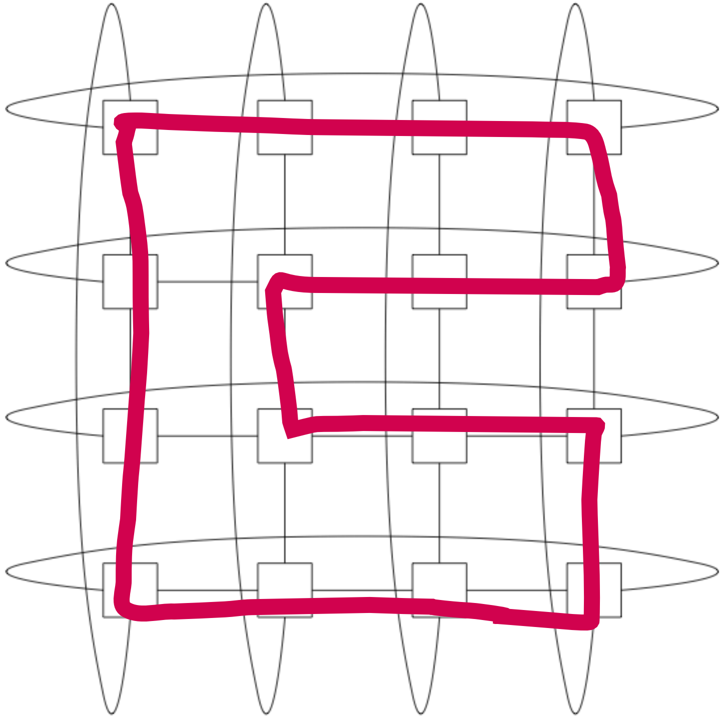


b) All-gather phases

Topology-aware Collective Algorithms

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

Ring Algorithm

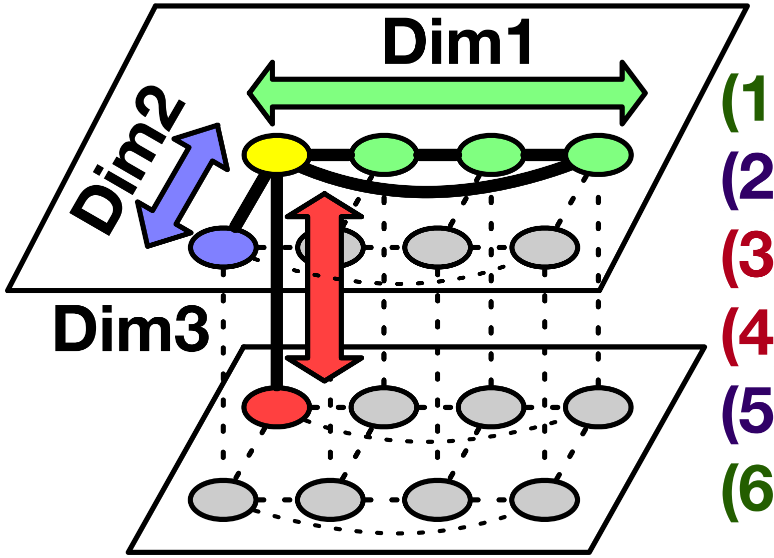


Network Underutilization!!

Physical Topology: 2D Torus

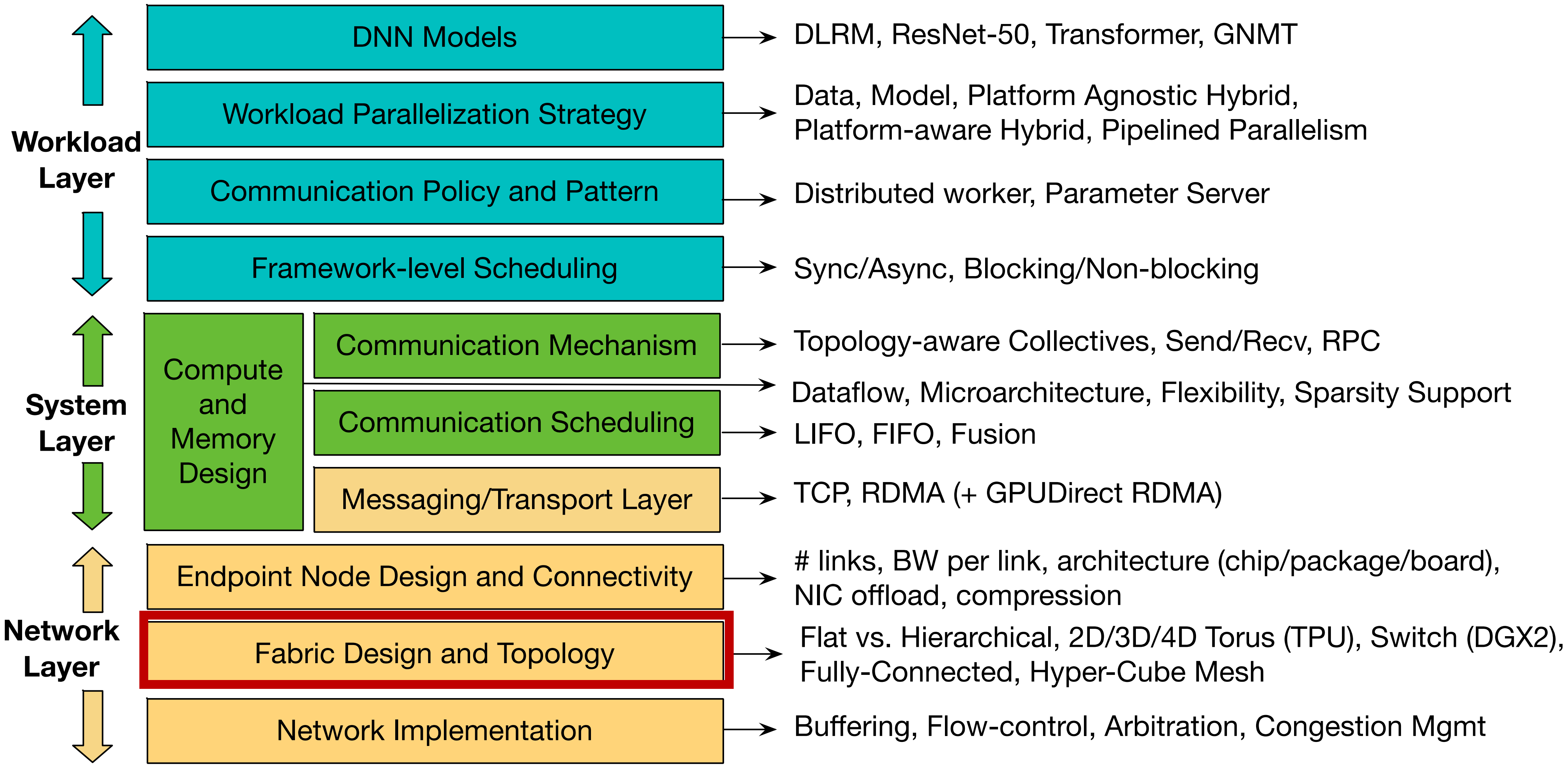
Multi-dimensional Collective Algorithm

- **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

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 and Memory Design → Dataflow, Microarchitecture, Flexibility, Sparsity Support

Communication Scheduling → LIFO, FIFO, Fusion

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

Network Layer

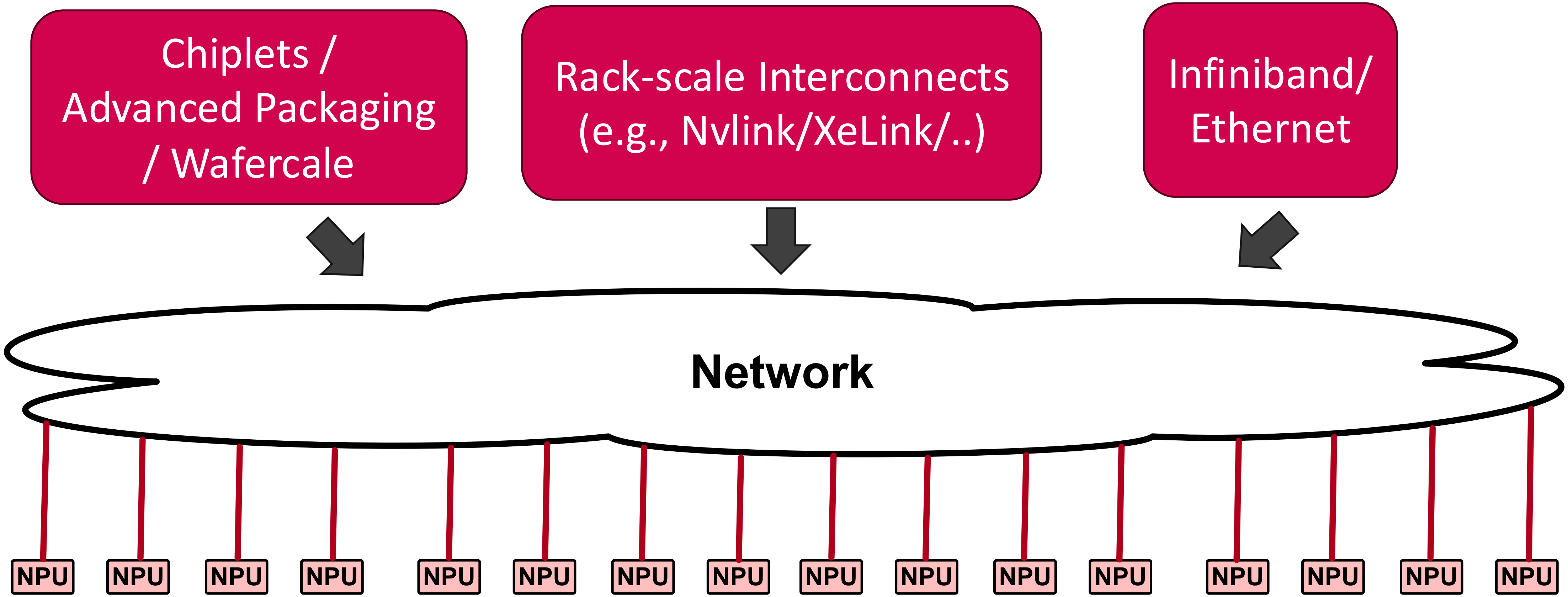
Endpoint Node Design and Connectivity → # links, BW per link, architecture (chip/package/board), NIC offload, compression

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

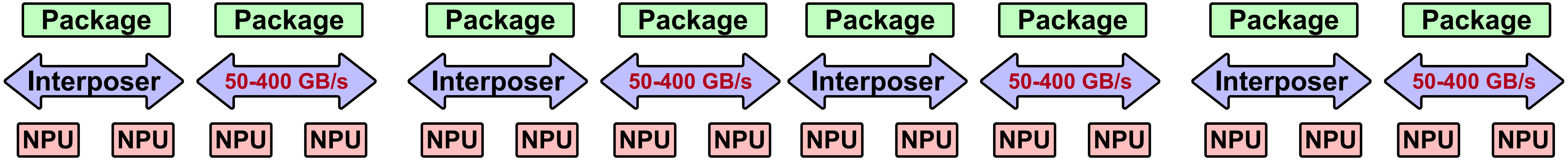


Chiplets / Advanced Packaging / Wafercale

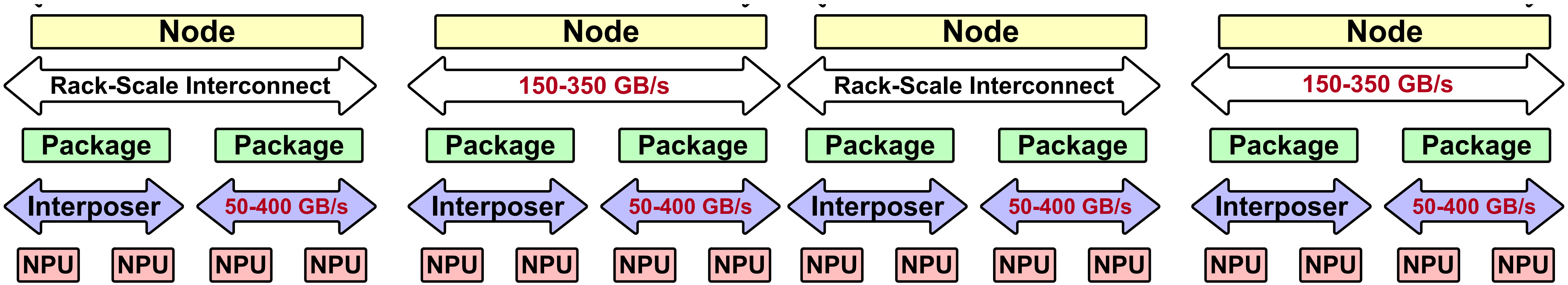
Rack-scale Interconnects (e.g., Nvlink/XeLink/..)

Infiniband/ Ethernet

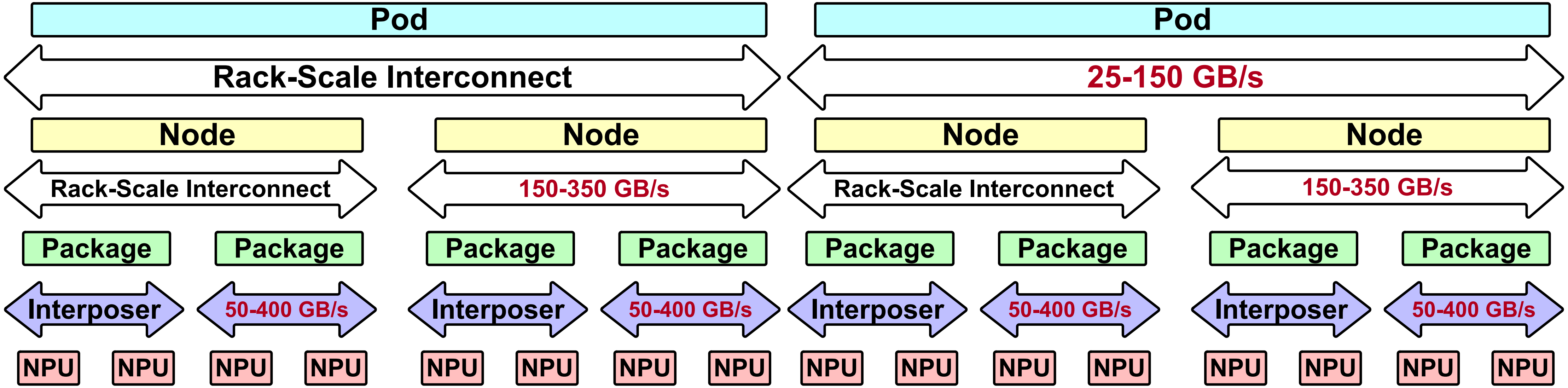
Hierarchical Network Architectures



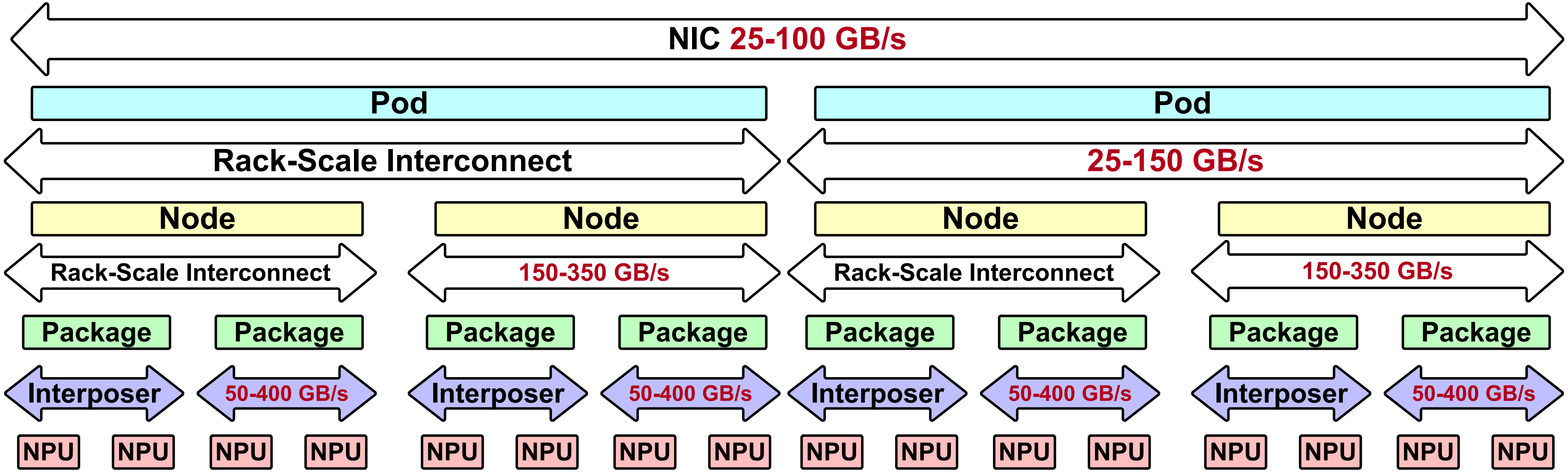
Hierarchical Network Architectures



Hierarchical Network Architectures

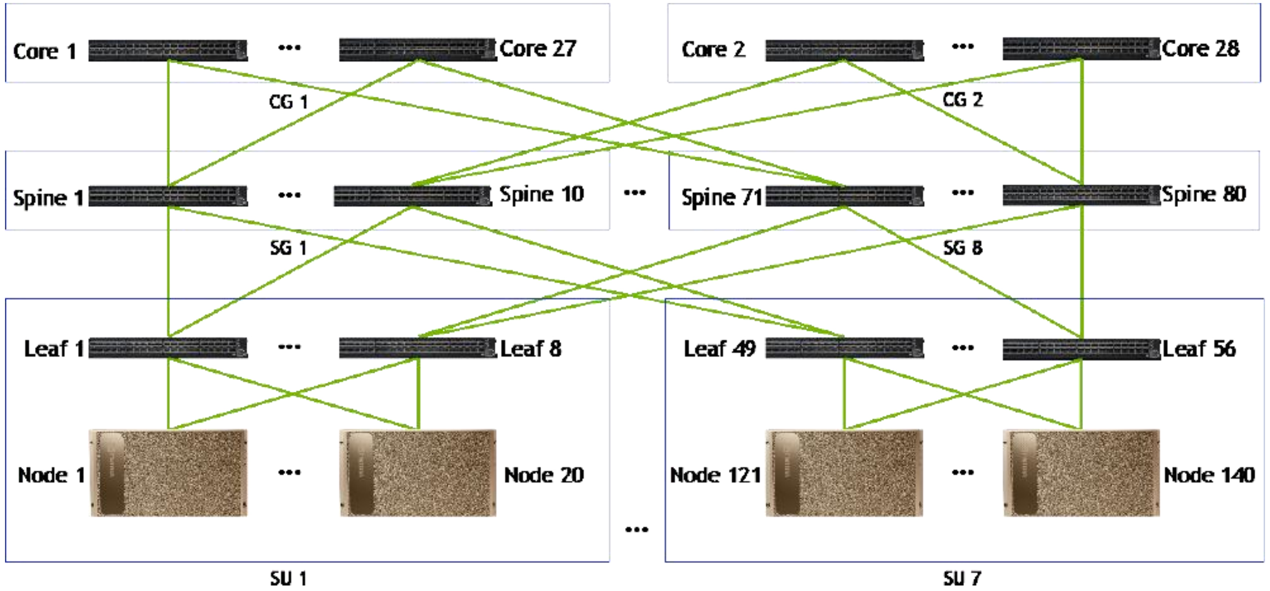


Hierarchical Network Architectures



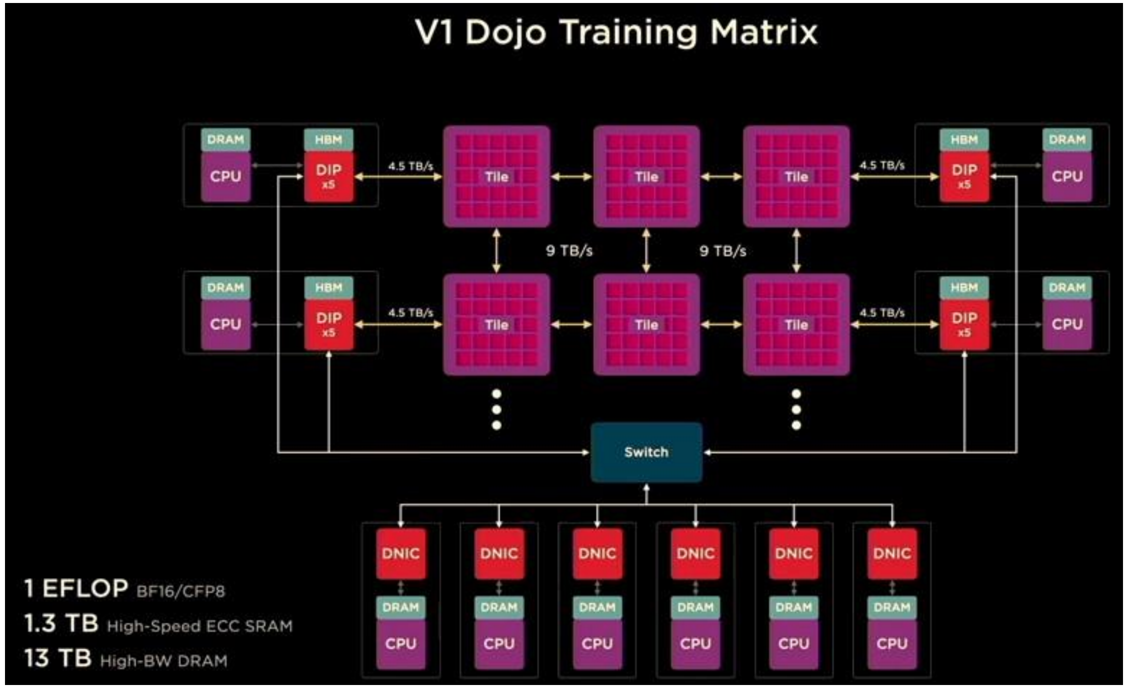
NVIDIA DGX SuperPod

- Multi-level switches

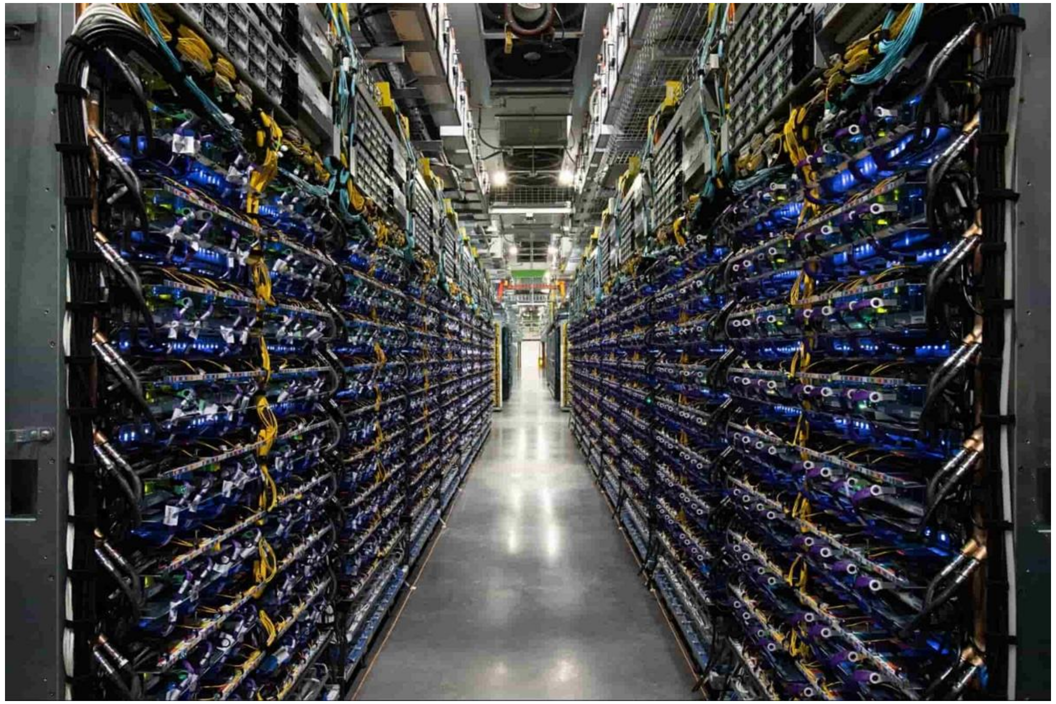


Tesla Dojo ExaPOD

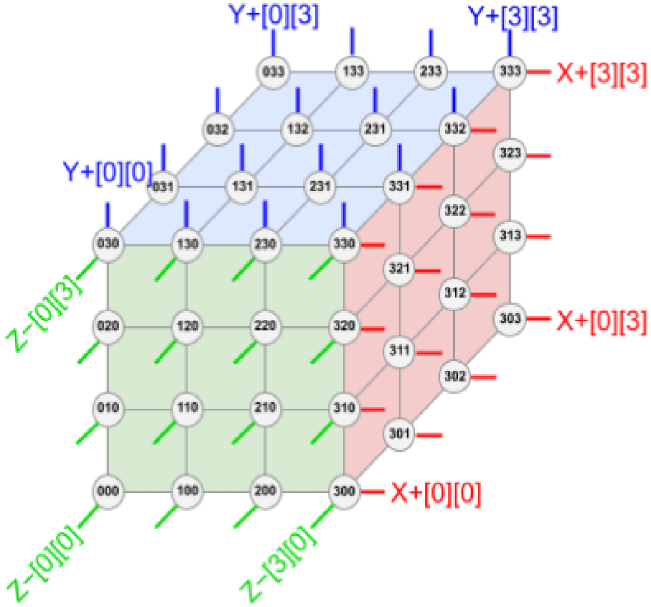
- Scale-out Mesh Network

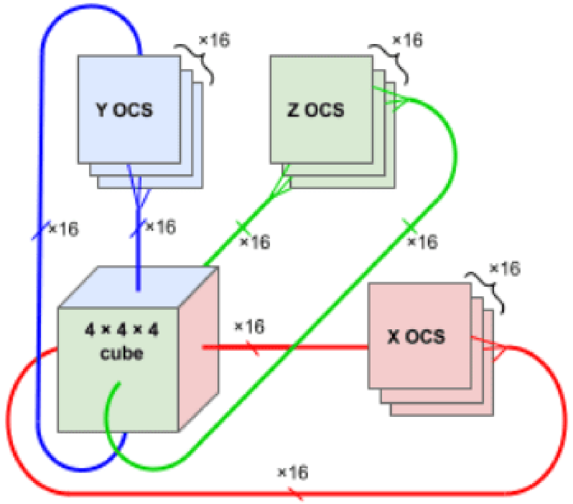


Google Cloud TPU v4

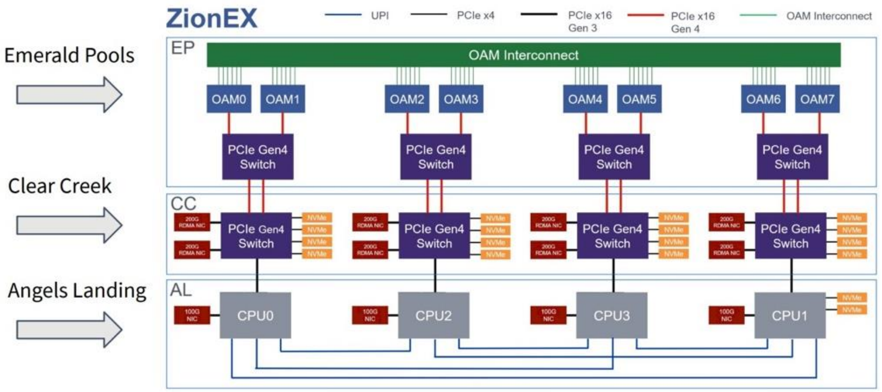


- 3D Torus + Optical Networks

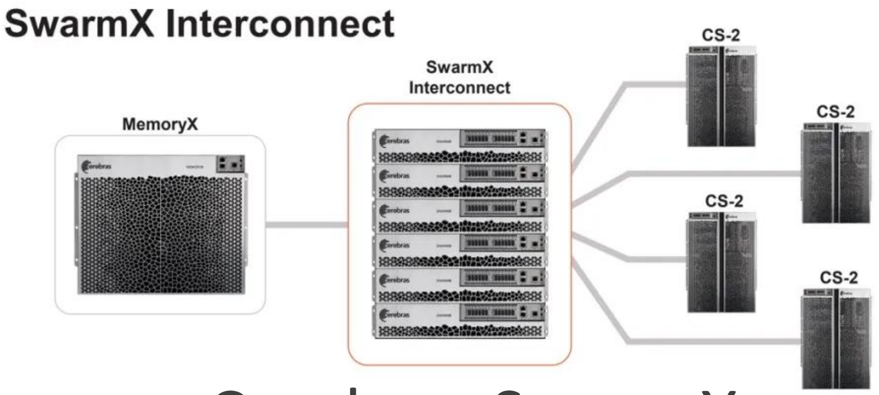




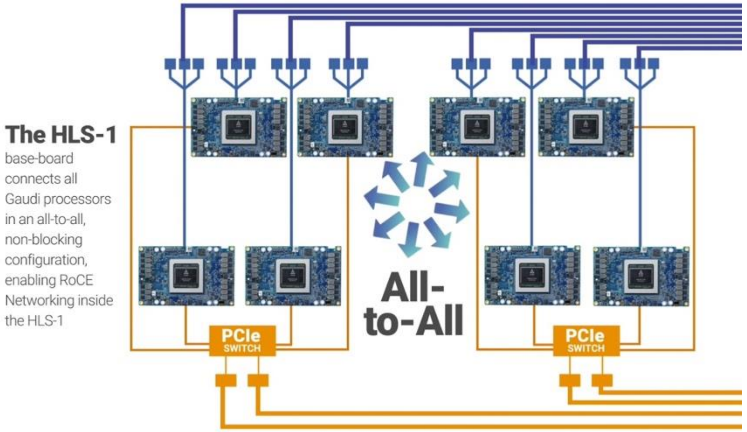
State-of-the-art Training Clusters



Meta ZionEX

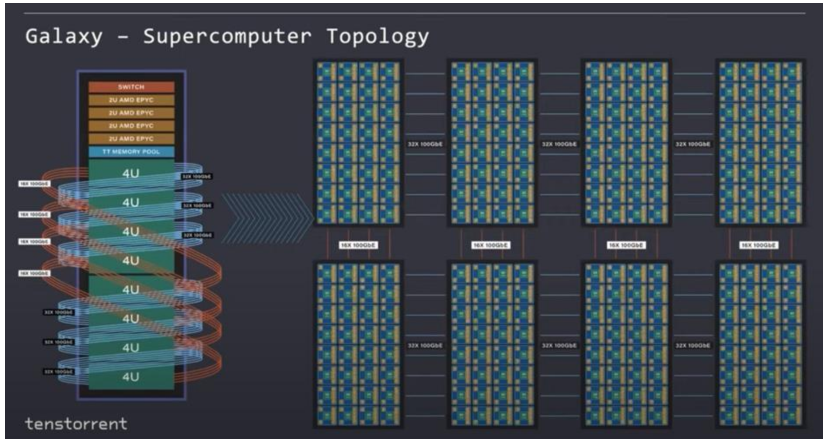


Cerebras SwarmX



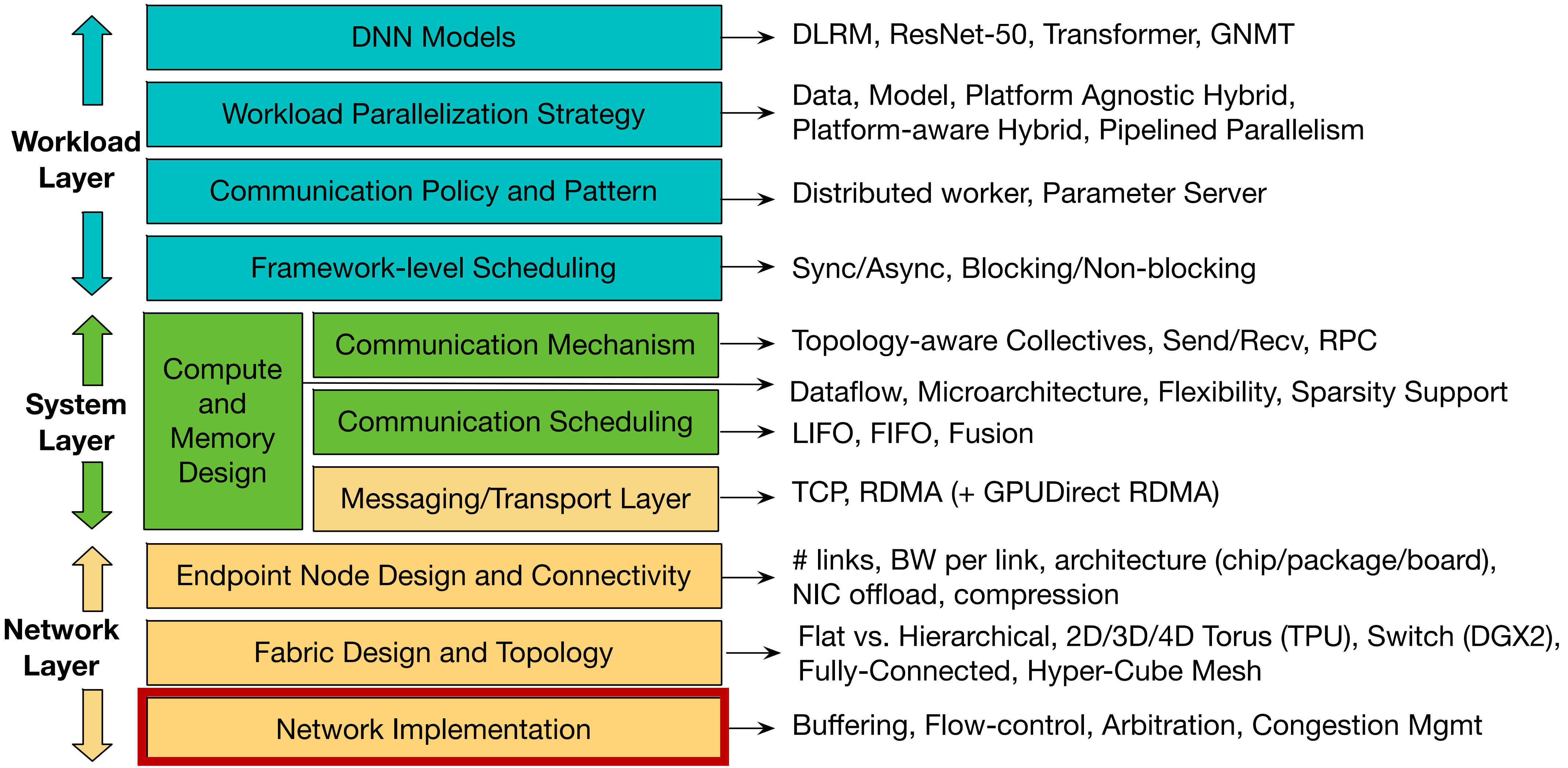
The HLS-1 base-board connects all Gaudi processors in an all-to-all, non-blocking configuration, enabling RoCE Networking inside the HLS-1

Intel Habana HLS-1



Tenstorrent Wormhole

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 and Memory Design → Dataflow, Microarchitecture, Flexibility, Sparsity Support
- Communication Scheduling → LIFO, FIFO, Fusion
- Messaging/Transport Layer → TCP, RDMA (+ GPUDirect RDMA)

Network Layer

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

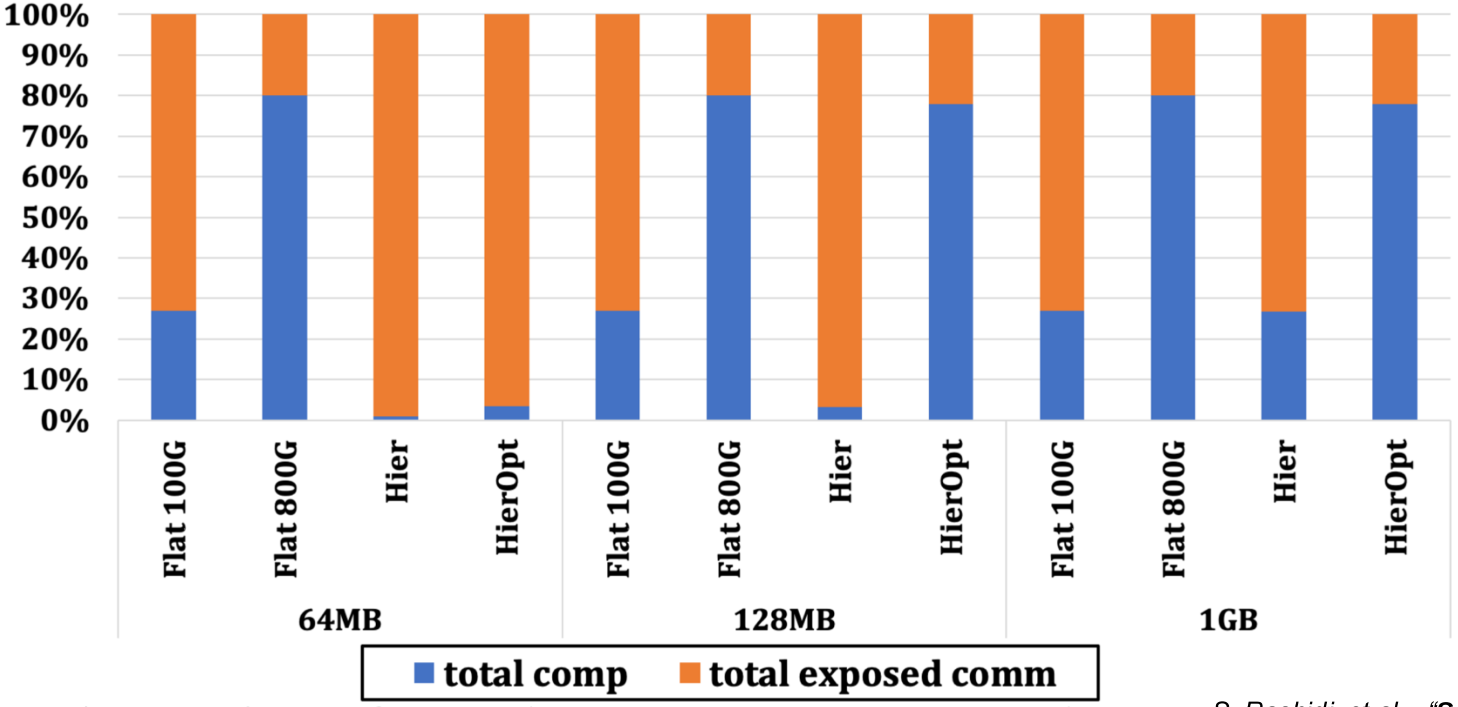
Figure Courtesy: Srinivas Sridharan (NVIDIA)

Effect of Size of Switch Buffer

Observations:

- Flat vs. Hierarch different Sensitivity to global switch size

DLRM



S. Rashidi, et al., *“**Scalable Distributed Training of Recommendation Models: An ASTRA-SIM + NS3 case-study with TCP/IP transport**”, Hot Interconnects 2020*

Summary and Takeaways

- Large Model distributed ML is an ongoing open-research area
- Many emerging supercomputing systems being designed specifically for this problem!
  - NVIDIA HGX + (Mellanox) SHARP switches
  - Cerebras CS2
  - Tesla Dojo
  - Intel Habana
  - IBM Blueconnect
  - …
- Co-design of algorithm and system offers high opportunities for speedup and efficiency