



Georgia Tech School of Electrical and Computer Engineering  
College of Engineering



<http://synergy.ece.gatech.edu>



ISCA Tutorial

June 19th, 2022

# Enabling HW/SW Co-Design of Distributed Deep Learning Training Platforms

*ASTRA-sim Tutorial*



## Tushar Krishna

Associate Professor, School of ECE  
Georgia Institute of Technology

[tushar@ece.gatech.edu](mailto:tushar@ece.gatech.edu)

# Welcome



**Tushar Krishna**

Associate Professor, School of ECE  
Georgia Institute of Technology

[tushar@ece.gatech.edu](mailto:tushar@ece.gatech.edu)



**Saeed Rashidi**

PhD Student, School of ECE  
Georgia Institute of Technology  
[saeed.rashidi@ece.gatech.edu](mailto:saeed.rashidi@ece.gatech.edu)

## Presenters



**Will Won**

Ph.D. Student, School of CS  
Georgia Institute of Technology  
[william.won@gatech.edu](mailto:william.won@gatech.edu)



**Taekyung Heo**

Postdoctoral Fellow, School of ECE  
Georgia Institute of Technology  
[tkheo@casys.kaist.ac.kr](mailto:tkheo@casys.kaist.ac.kr)



**Srinivas Sridharan**

Research Scientist, Meta

## Collaborators



**Sudarshan Srinivasan**

Research Scientist, Intel

# Agenda

Time (EDT)	Topic	Presenter
8:30 – 9:30	<b>Introduction to Distributed Deep Learning Training Platforms</b>	Tushar Krishna
9:30 – 10:30	<b>ASTRA-sim</b>	Saeed Rashidi
10:30 – 11:00	<b>Coffee Break</b>	
11:00 – 11:50	<b>Demo and Exercises</b>	William Won and Taekyung Heo
11:50 – 12:00	<b>Extensions and Future Development</b>	Taekyung Heo

## Tutorial Website

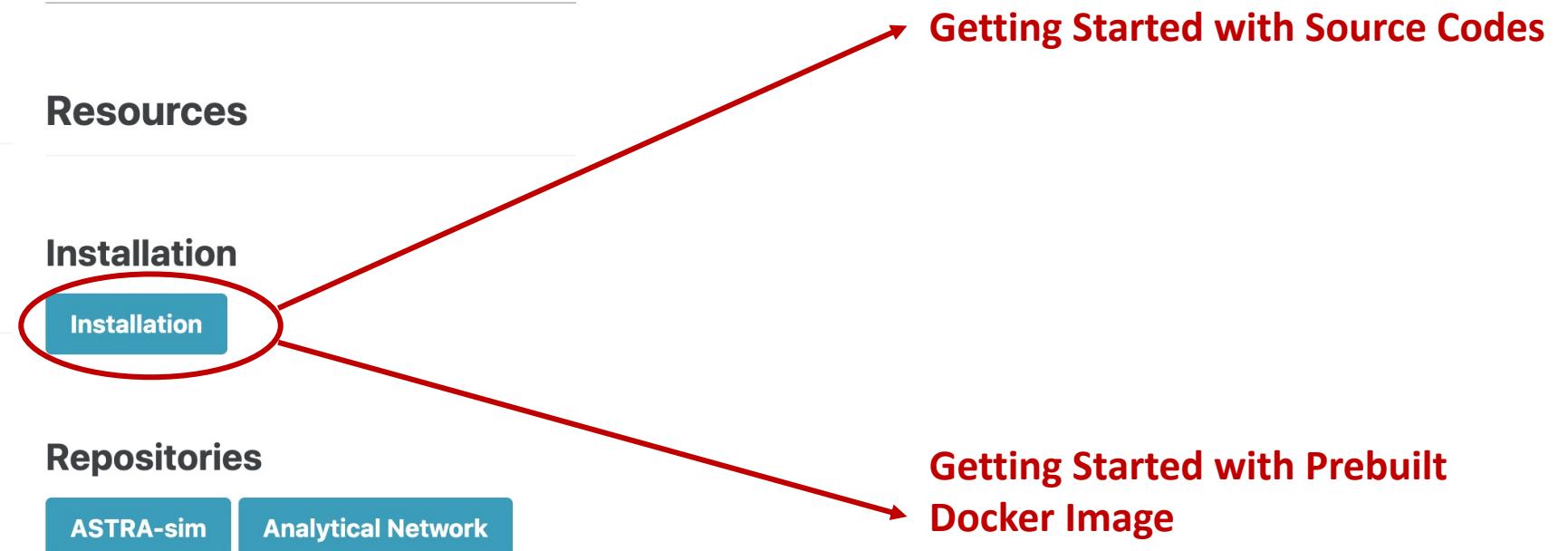
*includes agenda, slides, ASTRA-sim installation instructions (via source + docker image)*

<https://astra-sim.github.io/tutorials/isca-2022>

**Attention:** Tutorial is being recorded

# ASTRA-sim Installation

- Please go ahead and install ASTRA-sim!
- Instructions here:



# Agenda

Time (EDT)	Topic	Presenter
8:30 – 9:30	<b>Introduction to Distributed Deep Learning Training Platforms</b>	Tushar Krishna
9:30 – 10:30	<b>ASTRA-sim</b>	Saeed Rashidi
10:30 – 11:00	<b>Coffee Break</b>	
11:00 – 11:50	<b>Demo and Exercises</b>	William Won and Taekyung Heo
11:50 – 12:00	<b>Extensions and Future Development</b>	Taekyung Heo

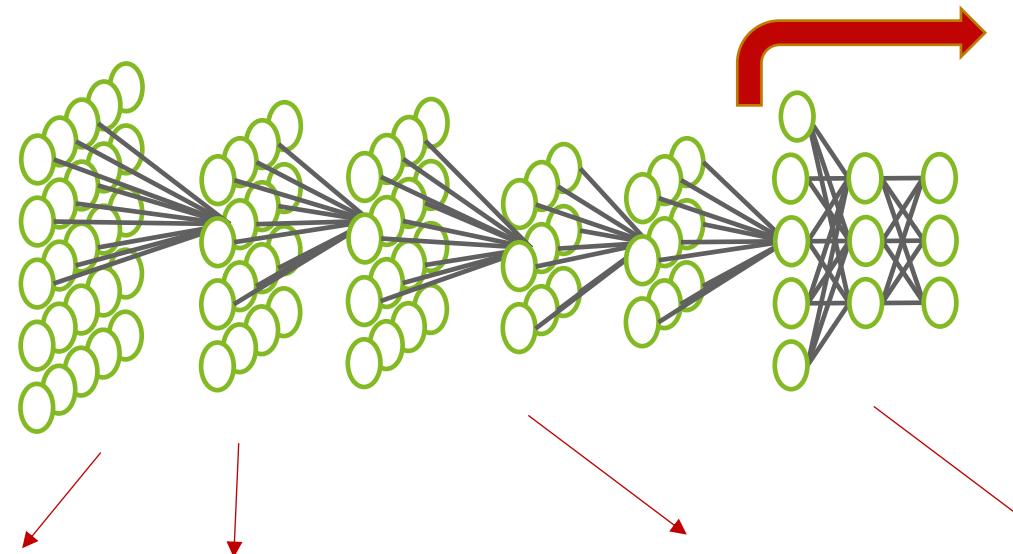
## Tutorial Website

*includes agenda, slides, ASTRA-sim installation instructions (via source + docker image)*

<https://astra-sim.github.io/tutorials/isca-2022>

**Attention:** Tutorial is being recorded

# The engine driving the AI Revolution



Training

*Training a deep neural network (DNN) involves feeding it a training dataset to update its weights to model the underlying function representing the dataset*



Object Detection



Speech Recognition



Language  
Understanding



Recommender Systems

# “Training” in the context of ML

- Machine Learning algorithms learn to make decisions or predictions based on data.
- We can categorize current ML algorithms based on the following three characteristics
  - **Feedback from data**
    - Supervised learning
    - Unsupervised learning
    - Semi-supervised learning
    - Reinforcement learning
  - **Purpose / Task**
    - Anomaly Detection
    - Classification
    - Clustering
    - Dimensionality Reduction
    - Representation Learning
    - Regression
  - **Method (for hyperparameter optimization)**
    - SGD
    - EA
    - Rule-based
    - Topic Models
    - ..

*We focus on Supervised Learning with SGD  
--> most popular for DNNs*

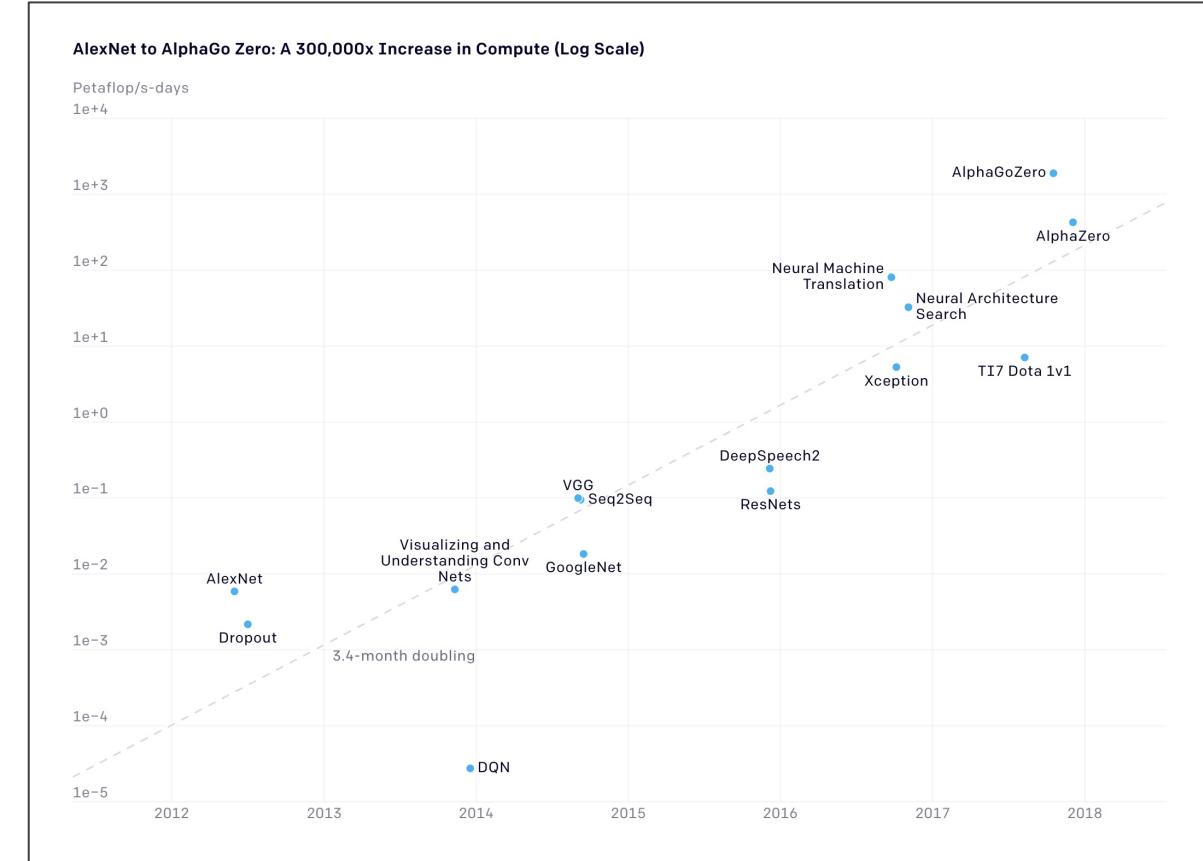
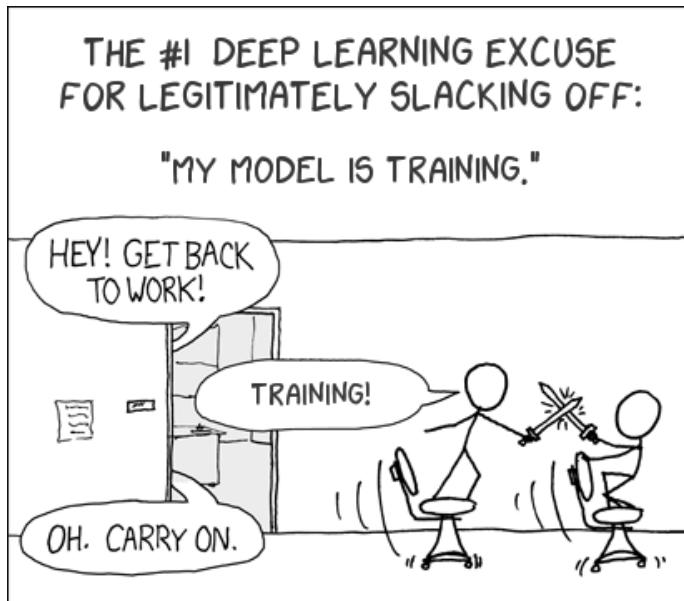
*Source: A Survey on Distributed Machine Learning*  
<https://dl.acm.org/doi/abs/10.1145/3377454>

# DL Training: The Phases

- Each training algorithm consists of 3 computation phases:
  - 1. Forward pass (inference):
    - The process of finding output activations using inputs and weights.
  - 2. Weight gradient computation:
    - The process of finding the gradient of weights (with respect to the loss function) using output gradients and inputs.
  - 3. Input gradient computation:
    - The process of finding the gradient of inputs (with respect to the loss function) using output gradients and weights.
- Operations 2 & 3 together are called backpropagation.
- The **training loop** dictates the order in which we issue the basic operations and (possibly) their related communication tasks.

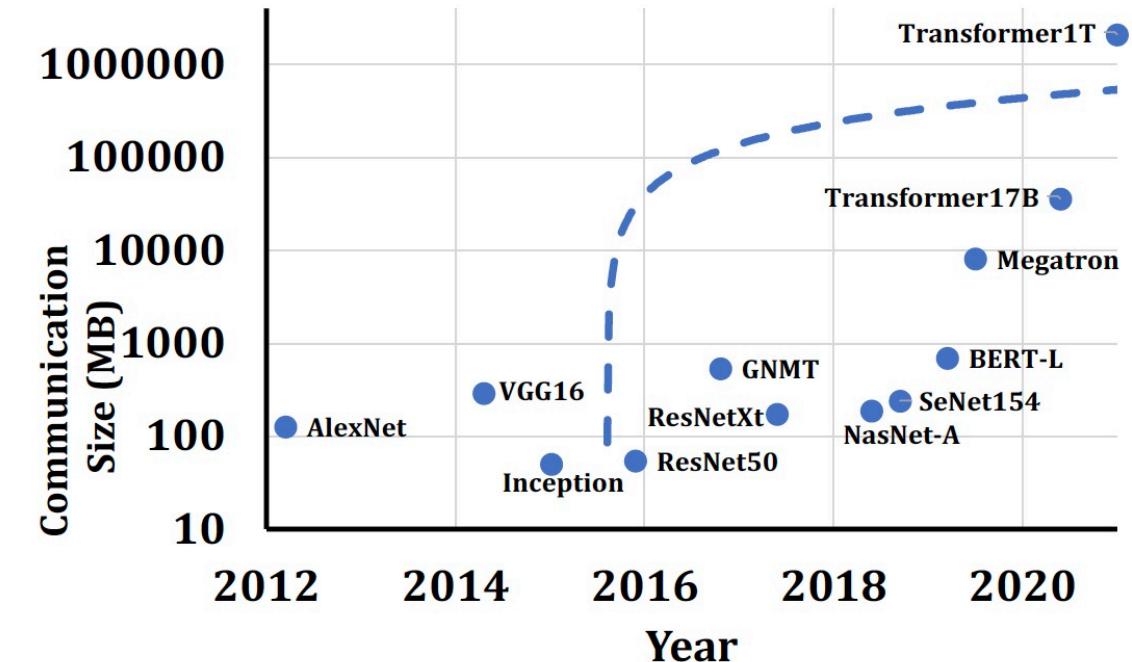
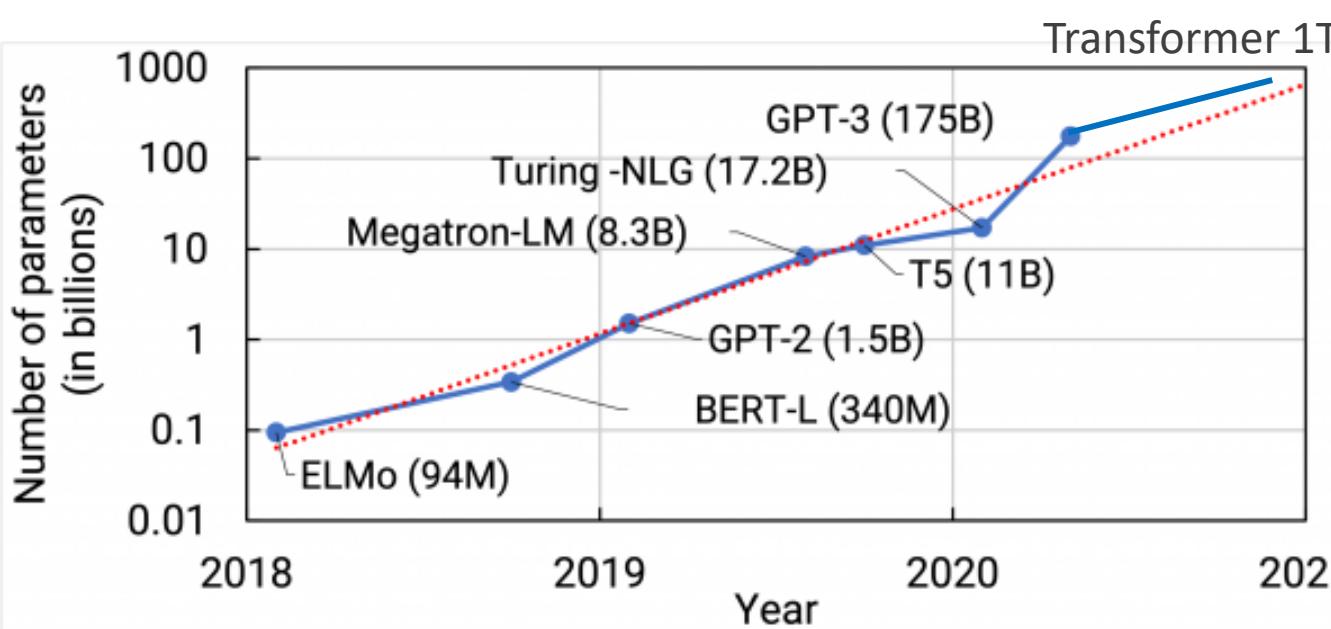
# Deep Learning Training Challenge

- **Training time is increasing**
  - DNNs are becoming larger
    - Turing NLG: 17.2 B Parameters
    - Megatron LM: 8.3B Parameters
  - Training samples are becoming larger
  - Moore's Law has ended



Source: <https://openai.com/blog/ai-and-compute/>

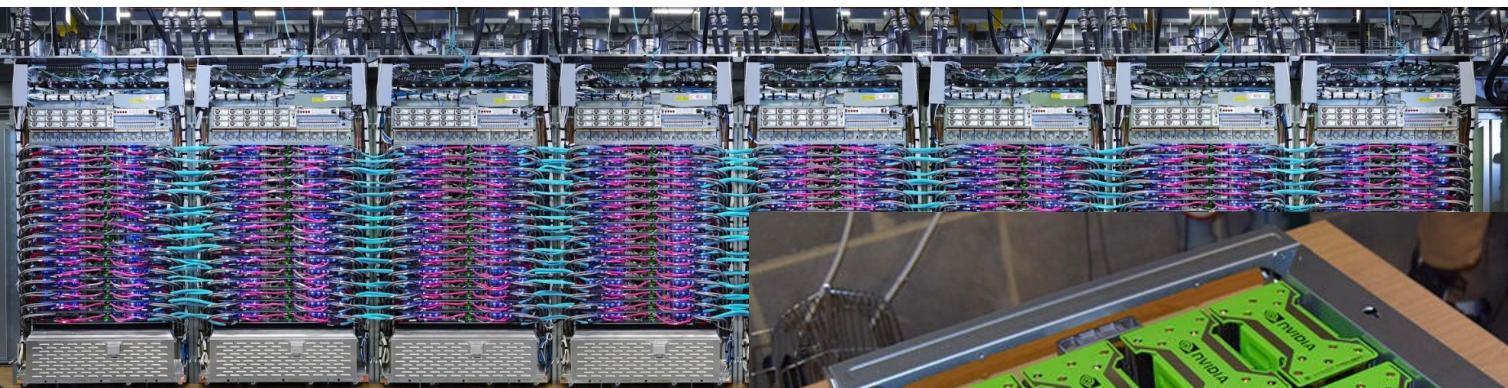
# Key Challenge: Large Models → Large Comms



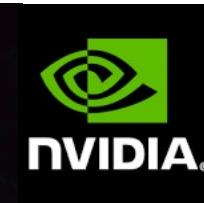
## Challenges:

- Multiple NPUs are required to fit large-scale models
- e.g., 16 NPUs for GPT-3 (175B params)  
128 NPUs for Transformer-1T (1T params) (using ZeRO stage 2)

# Enter: DL Training Platforms

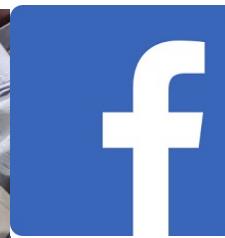


Google TPU v3



DGX 2

- ✓ Build customized chips for training
- ✓ Scale the training across more compute nodes



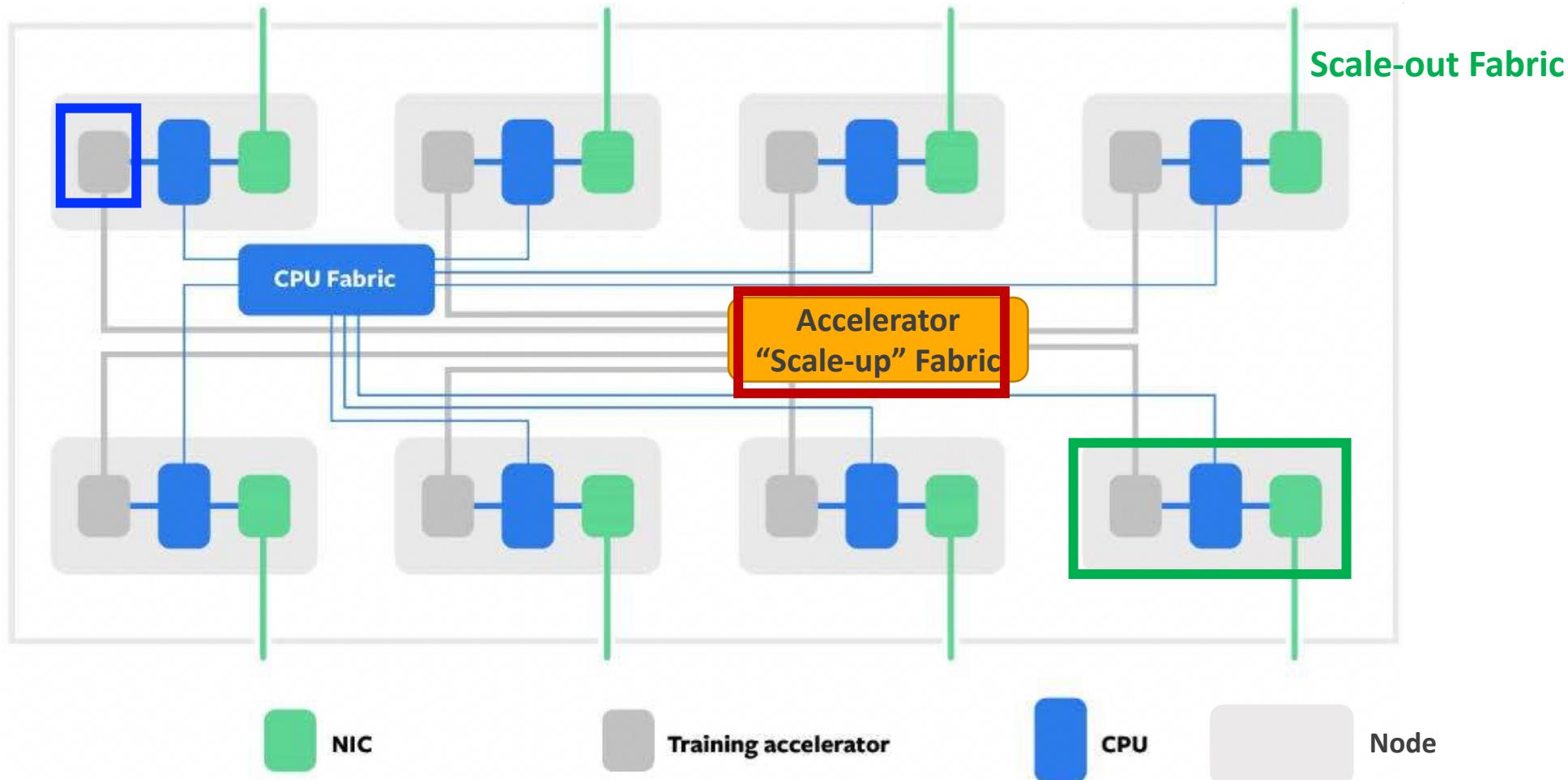
Zion

# And many more ...

---

- Cerebras CS2
- Tesla Dojo
- NVIDIA DGX + Mellanox SHARP switches
- Intel Habana
- IBM Blueconnect
- ...

# Components of a DL Training Platform



Modified version of source figure from : “Zion: Facebook Next- Generation Large Memory Training Platform”, Misha Smelyanskiy, Hot Chips 31”

# Understanding DL Training design-space

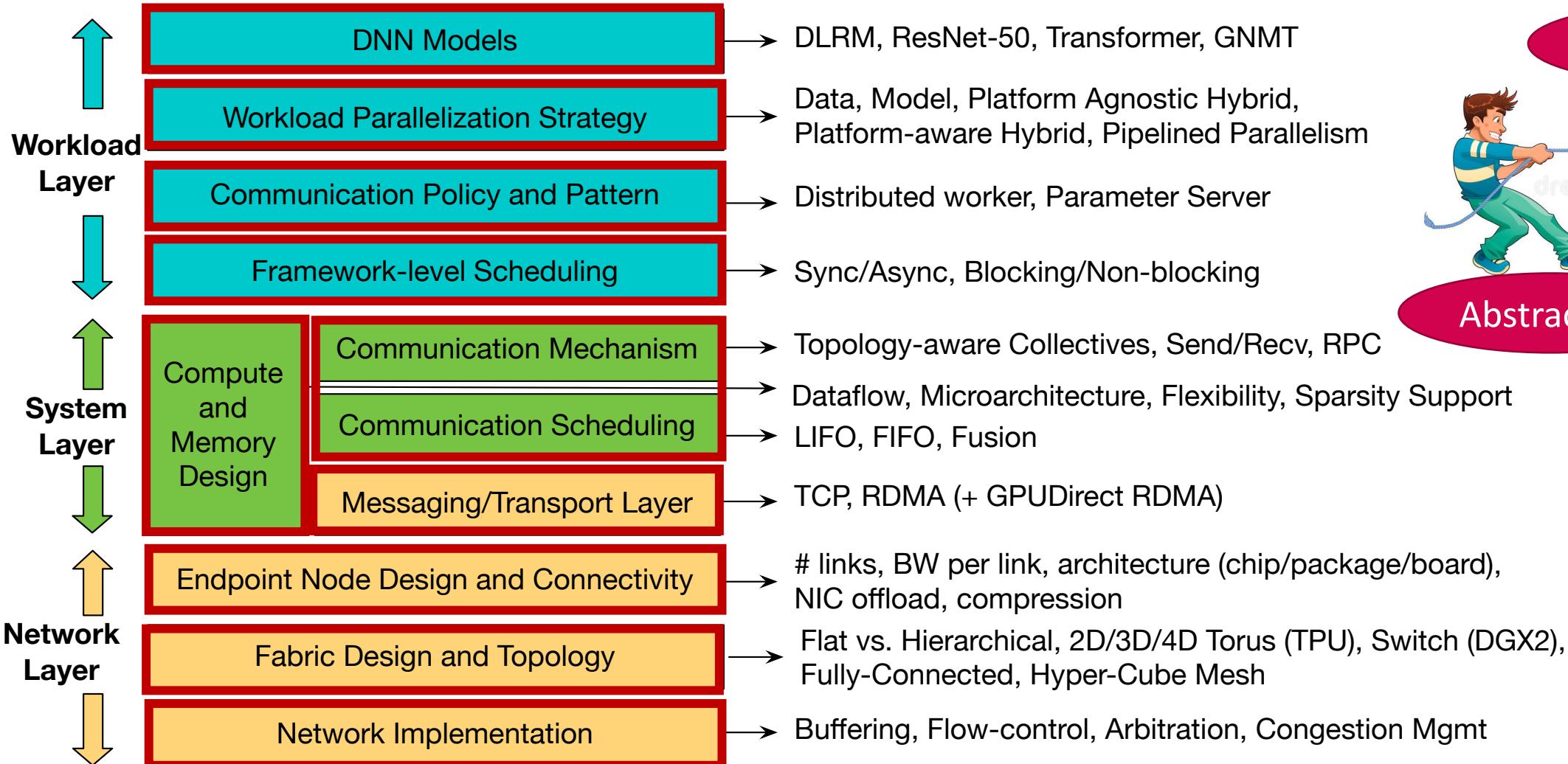


Figure Courtesy: Srinivas Sridharan (Facebook)

# Distributed Training Stack

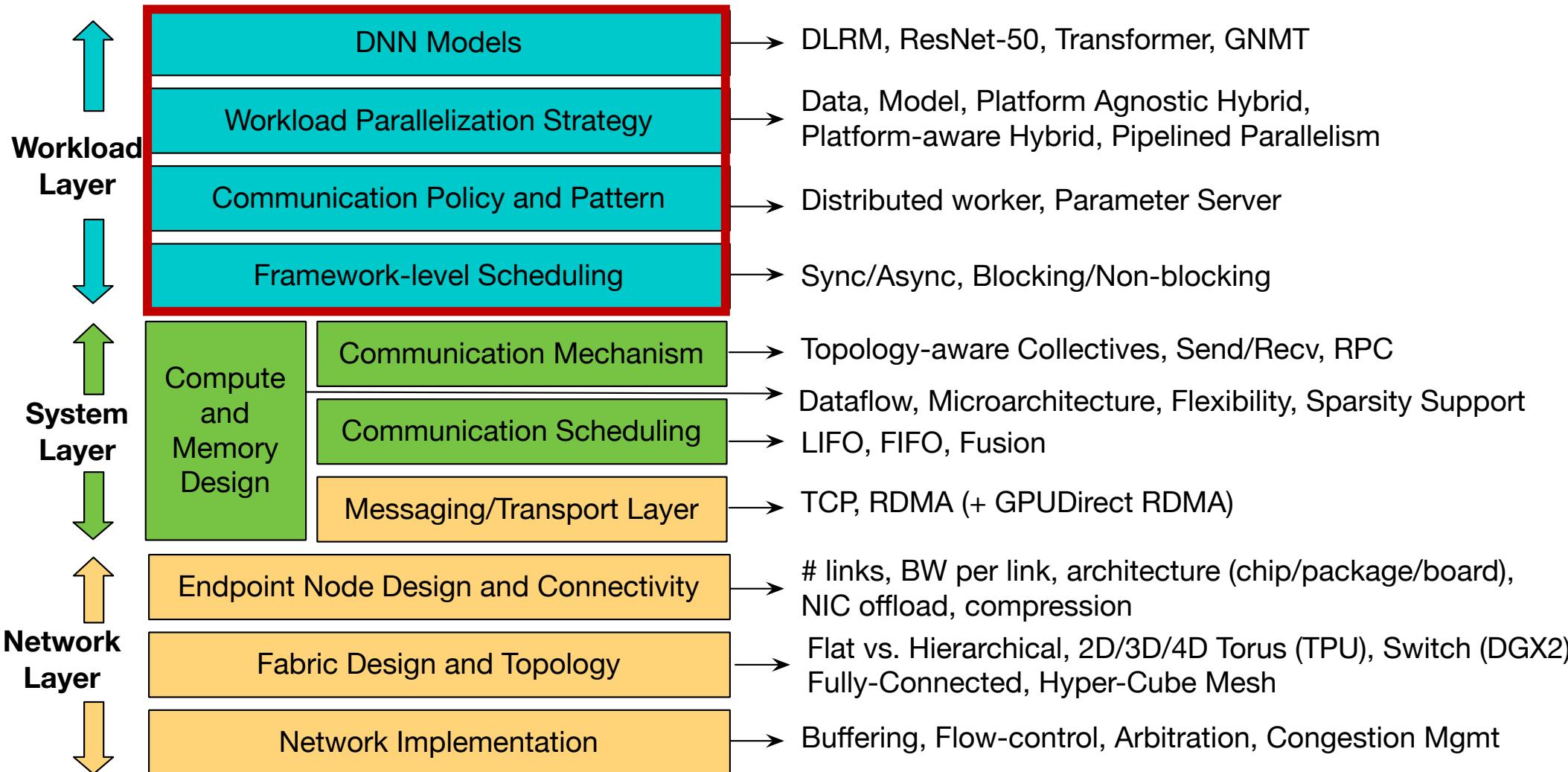
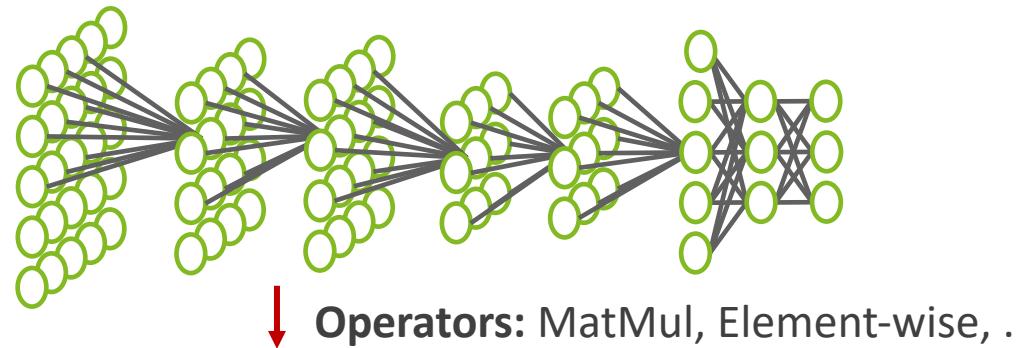
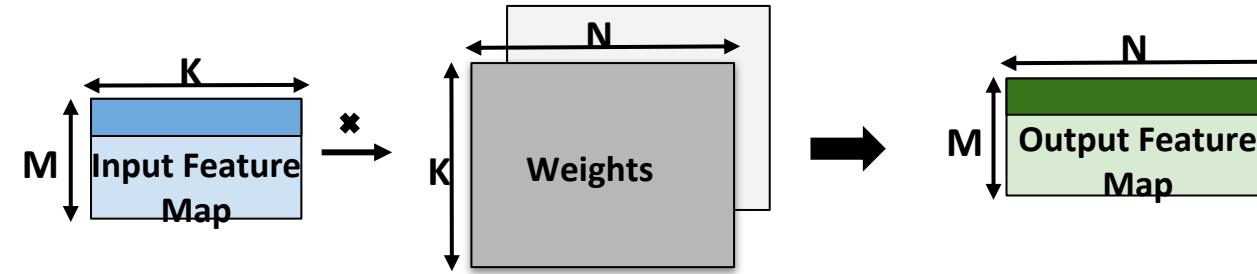


Figure Courtesy: Srinivas Sridharan (Facebook)

# DNN Model



## Forward Pass (Inference and Training)



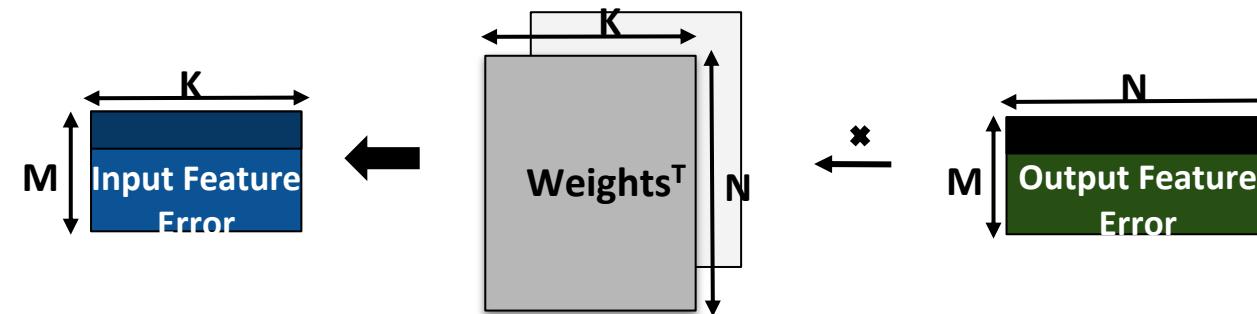
GEMM MNK  
Dimension Representation

M dim: batch size

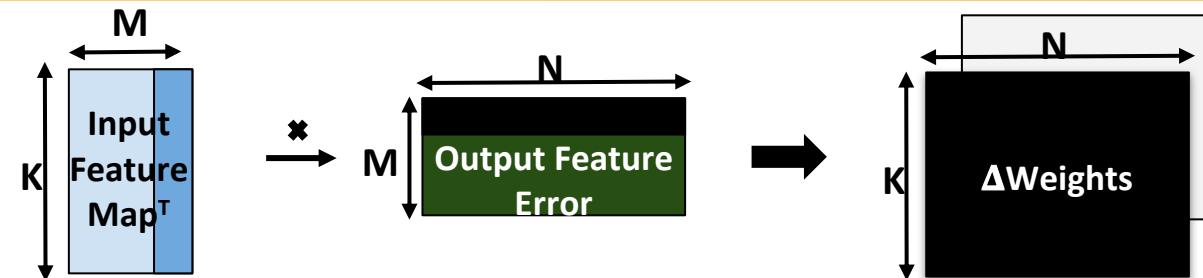
N dim: number of channels in the next layer

K dim:  $[H * W * C]$

## Backward Pass (Training)



## Gradient Computation (Training)

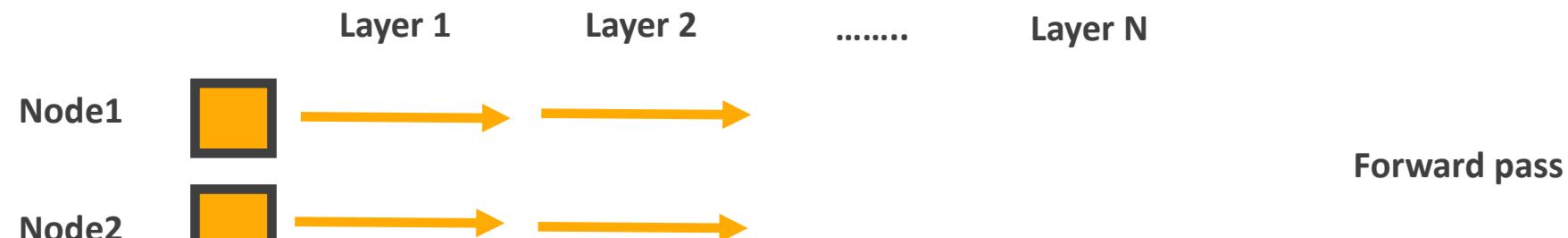


# Parallelization Strategies

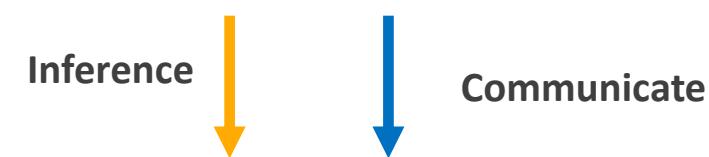
- The way compute tasks are distributed across different compute nodes. Multiple ways to split the tasks:
  - Split the minibatch (**Data-Parallel**)
  - Split the model (**Model-Parallel**)
  - Split the DNN layers: (**Pipeline-Parallel**)
  - ....
- This also defines the communication pattern across different nodes.

# Parallelism: Data-Parallel Training

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

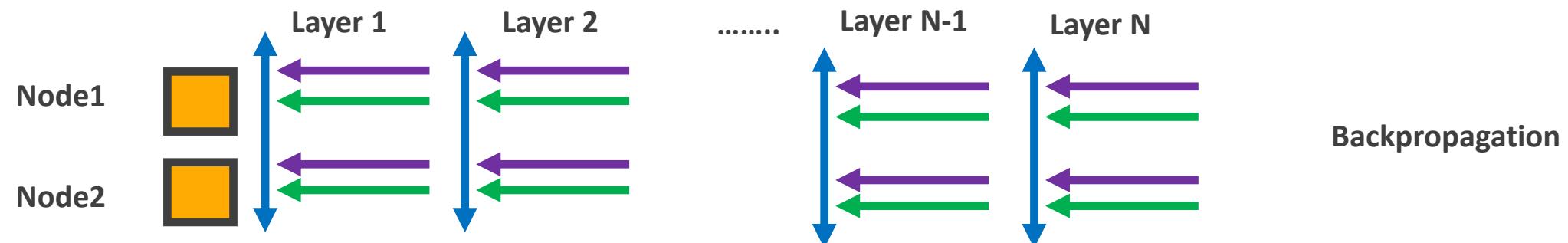


Flow-per-layer: 1. Compute output -> 2. go to the next layer

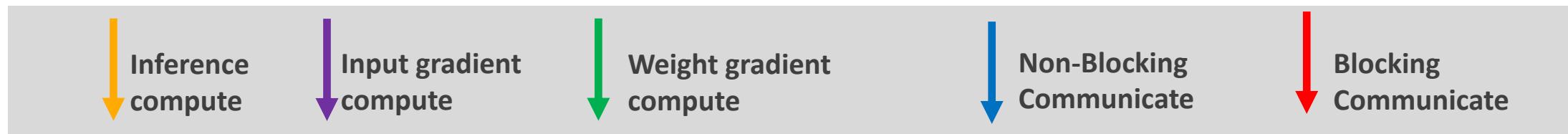


# Parallelism: Data-Parallel Training

- Distribute Data across multiple nodes and replicate model (network) along all nodes.
- **Communicate weight gradients** during the backpropagation pass.
  - Blocking wait during forward pass for collective of previous backpropagation for that layer.

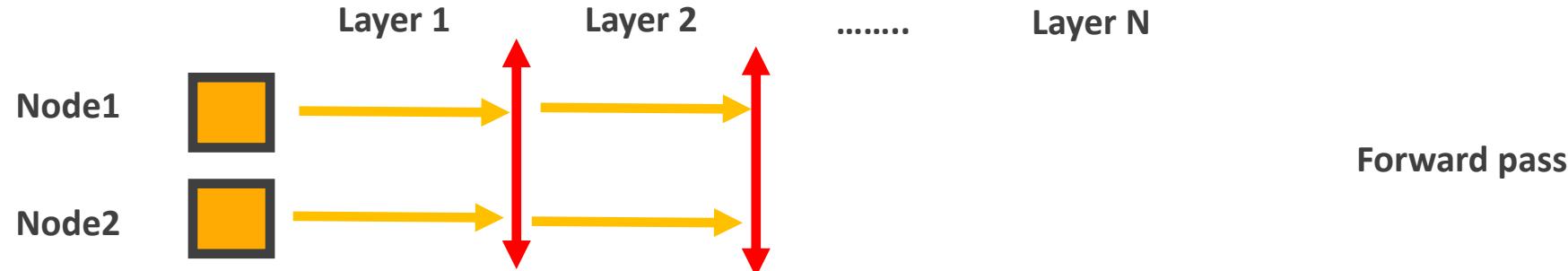


Flow-per-layer: 1. Compute weight gradient-> 2.issue weight gradient comm -> 3.compute input gradient -> 4. go to previous layer

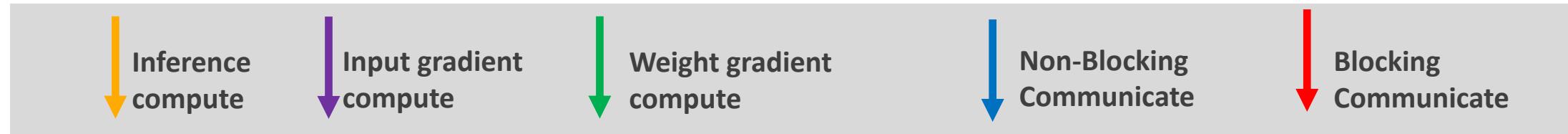


# Parallelism: Model-Parallel Training

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

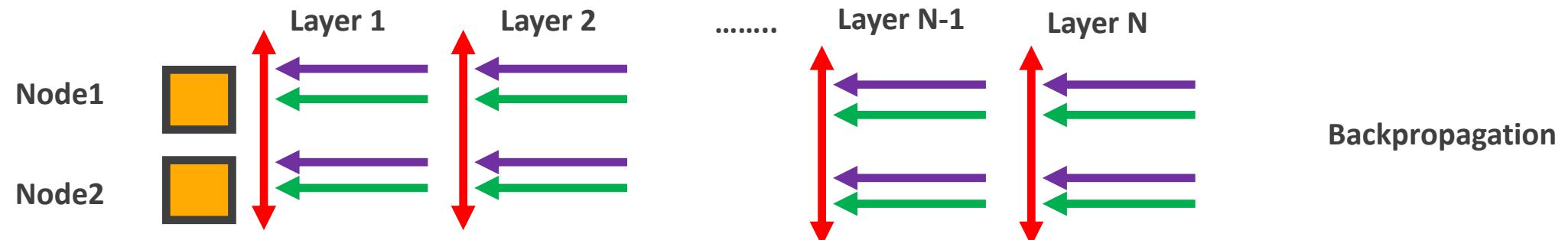


Flow-per-layer: 1. Compute output -> 2. issue output gradient comm -> 3. wait for gradient to be finished -> 4. go to the next layer

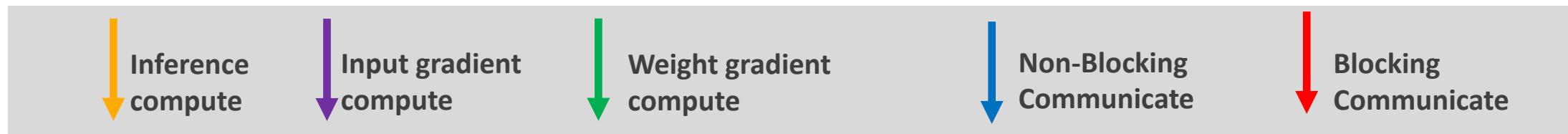


# Parallelism: Model-Parallel Training

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

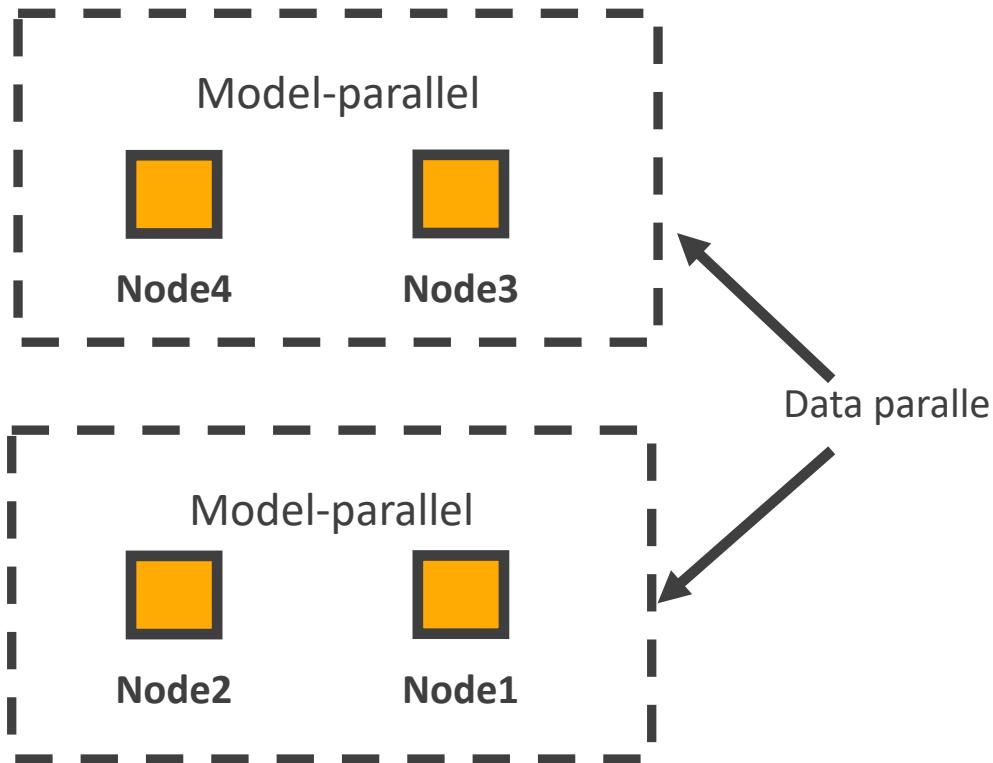


Flow-per-layer: 1. Compute input gradient-> 2.issue input gradient comm -> 3.compute weight gradient -> 4. wait for input gradient -> 5. go to previous layer



# Parallelism: Hybrid Parallel

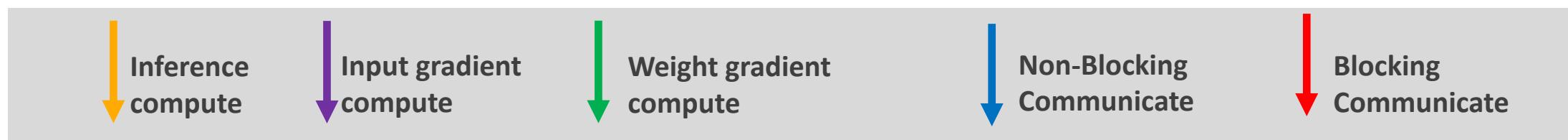
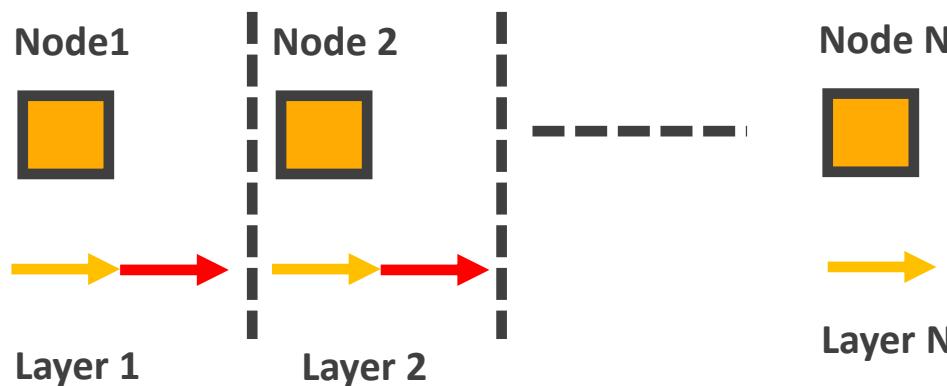
- Partition nodes into groups. Parallelism within a group is model-parallel, across the groups is data-parallel, or vice versa.



Parallelism	Activations during the forward pass	Weight gradients	Input gradients
Data		✓	
Model	✓		✓
Hybrid	partially	partially	partially

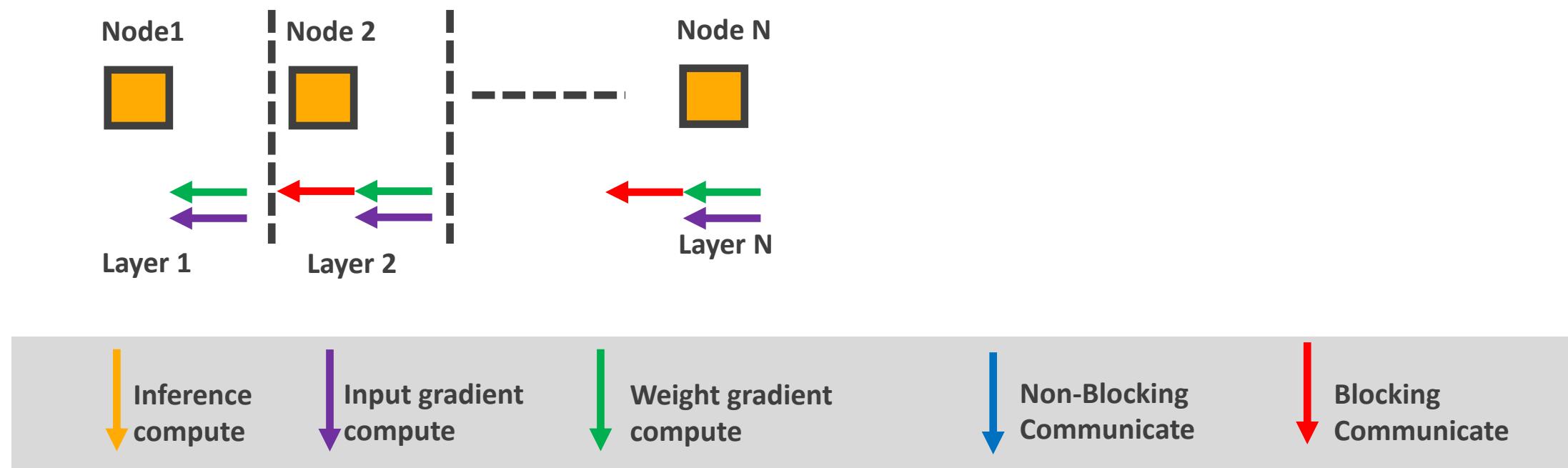
# Parallelism: Pipelined Parallel

- Distribute DNN layers across all nodes.
- Decompose minibatch into microbatches and propagate them to the pipeline in-order.
- **Communicate outputs** during the forward pass.



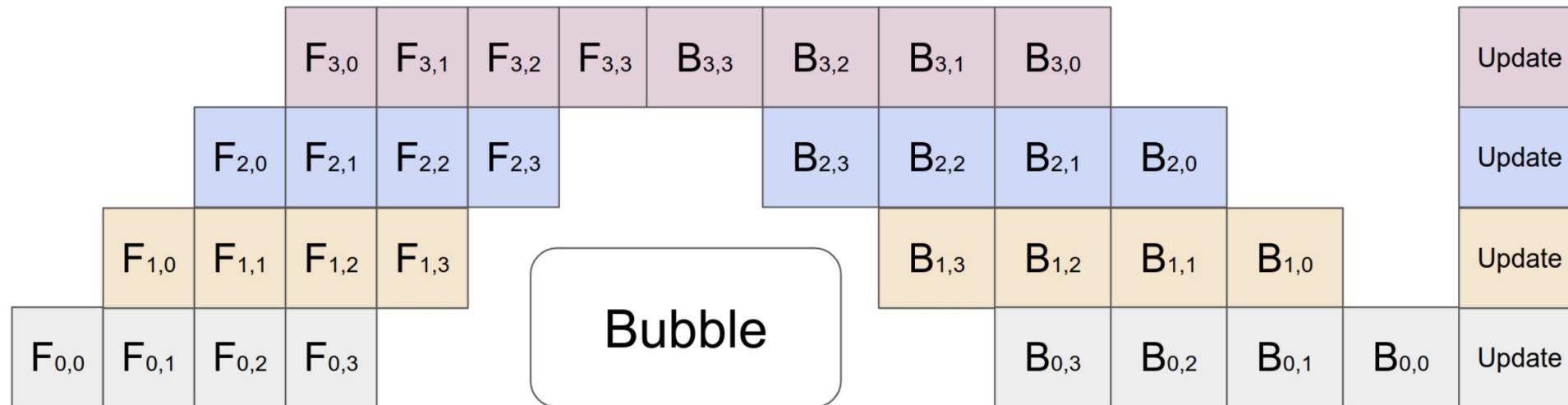
# Parallelism: Pipelined Parallel

- Distribute DNN layers across all nodes.
- Decompose minibatch into microbatches and propagate them to the pipeline in-order.
- **Communicate input gradients** during the backpropagation.



# Parallelism: Pipelined Parallel

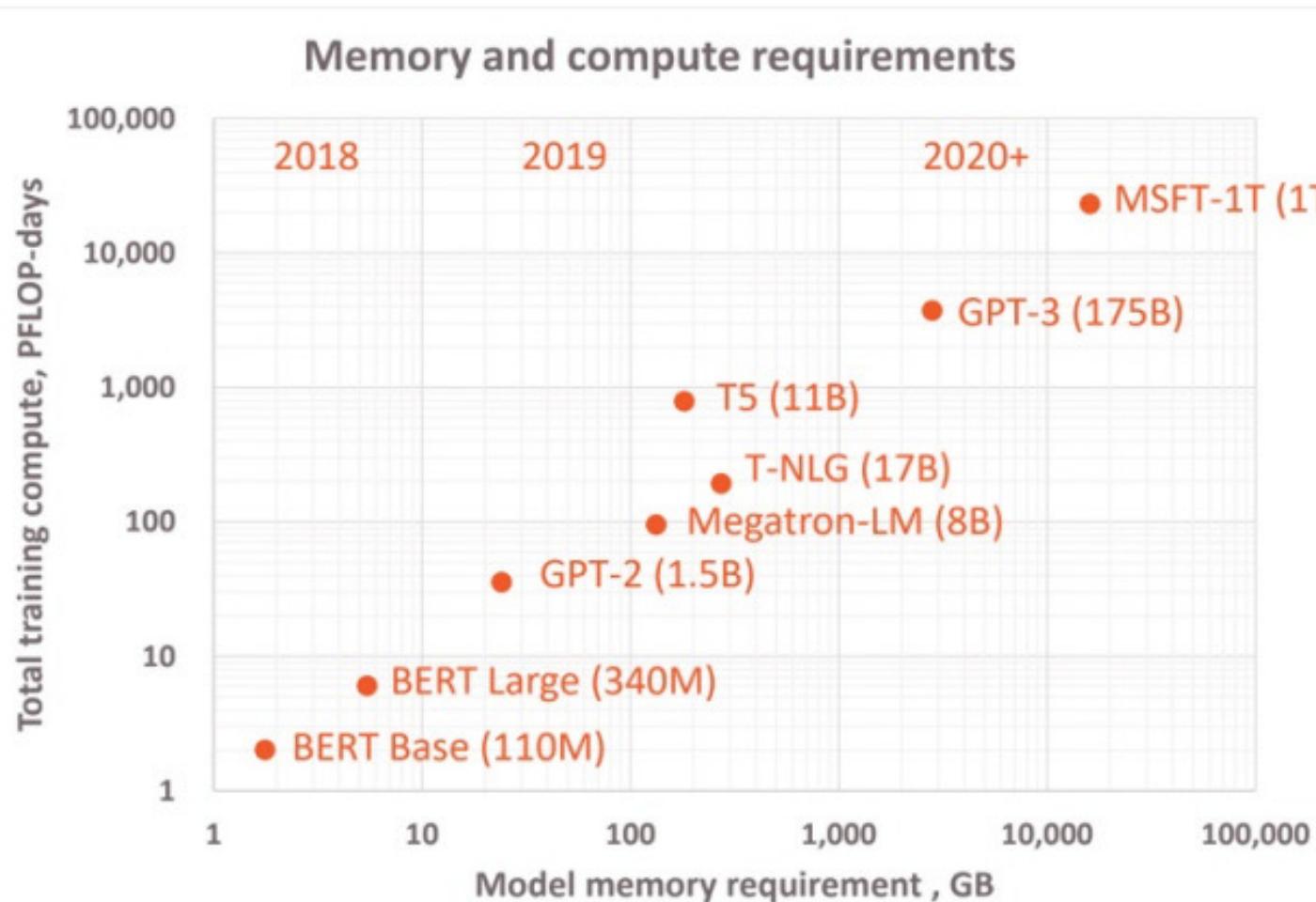
- How a minibatch is broken into micro-batches and pipeline is filled.



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

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

# Need for more sophisticated schemes ...



1000x **larger models**  
1000x **more compute**  
In just **2 years**

**Today**, GPT-3 with 175 billion params trained on 1024 GPUs for 4 months.

**Tomorrow**, **multi-Trillion** parameter models and beyond.

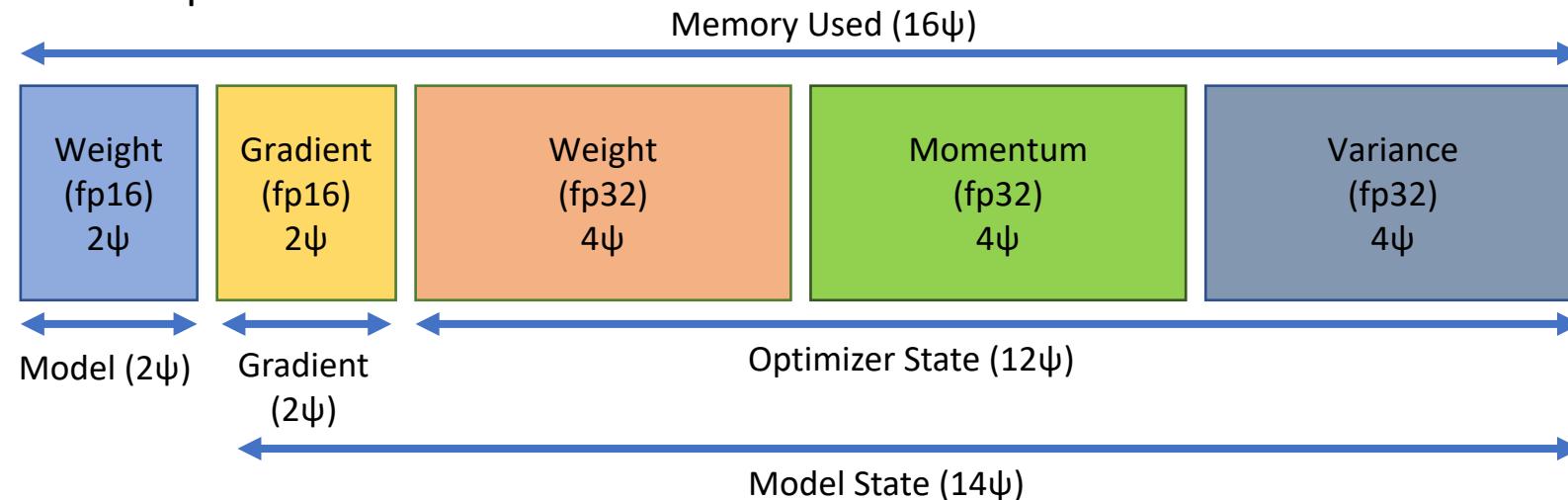
*Source: Cerebras (Hot Chips 2021)*

# Example 1: Microsoft ZeRO

- Motivation

- Data Parallelism (DP): Cannot fit large models
- Model Parallelism (MP): Computations too fine-grained, Large communication overhead, Layer-dependent design
- Large Memory Overhead for Model + Optimizer state
  - **8x overhead over model state!**

#Parameters:  $\psi$



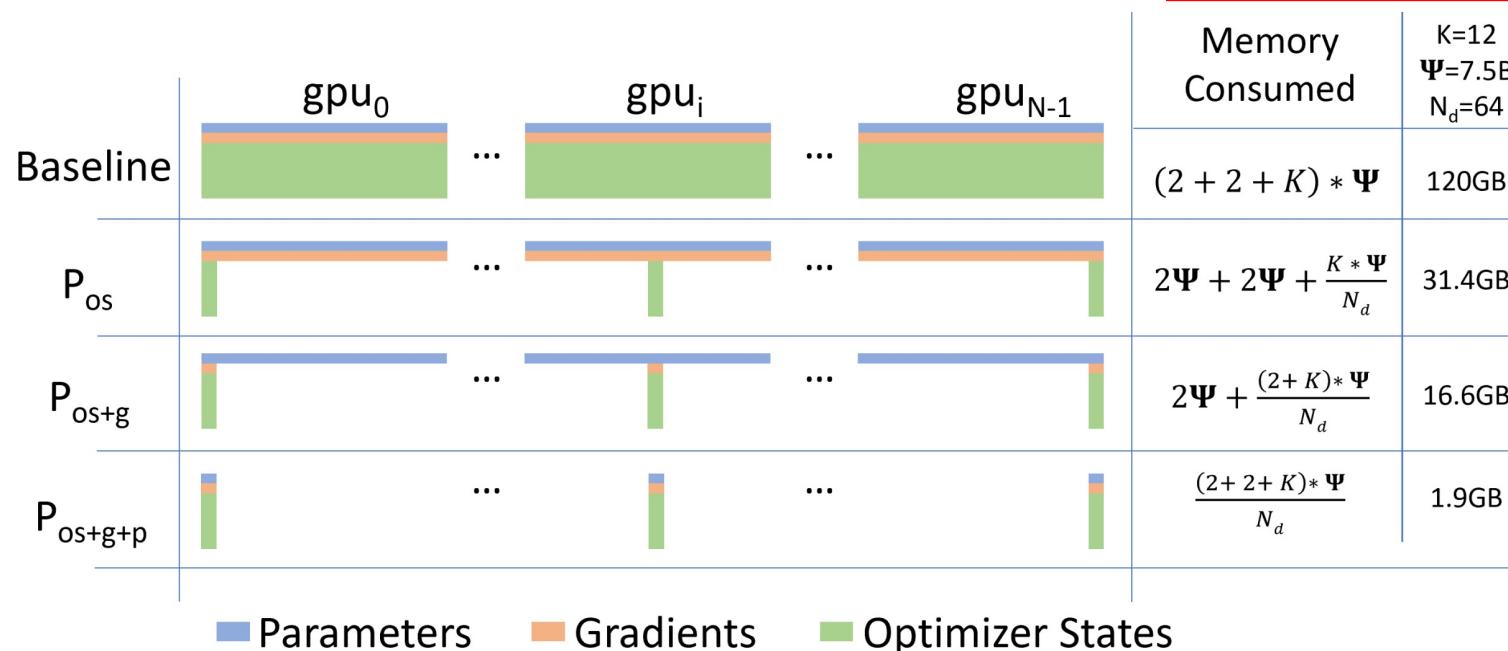
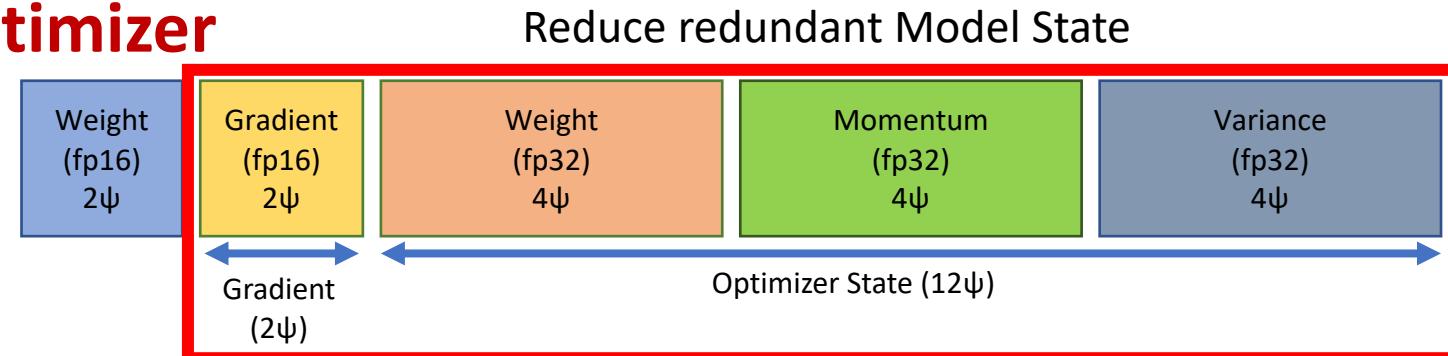
<https://www.microsoft.com/en-us/research/blog/zero-deepspeed-new-system-optimizations-enable-training-models-with-over-100-billion-parameters/>

# Example 1: Microsoft ZeRO

- **ZeRO: Zero Redundancy Optimizer**

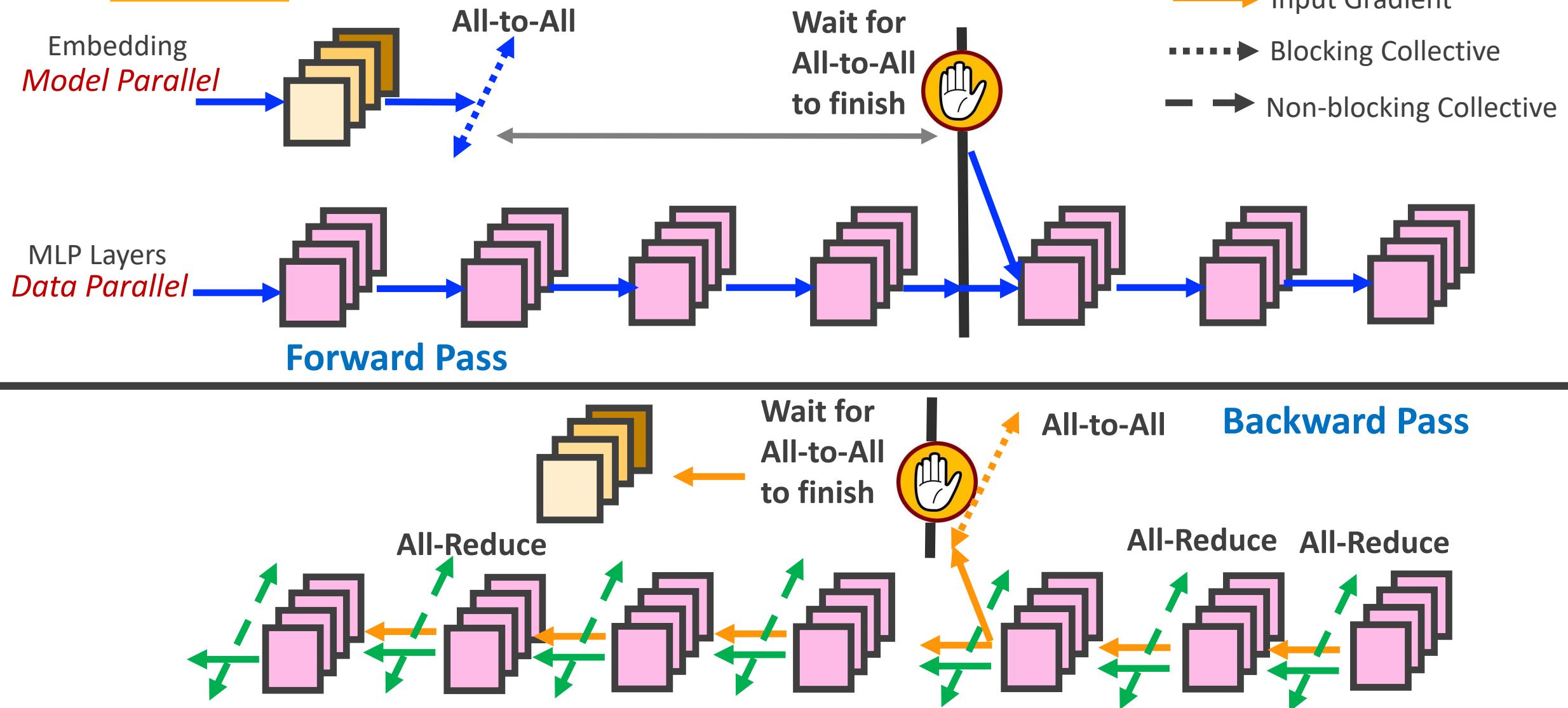
- Partition Optimizer state
- Partition Gradient state

- Memory vs Communication

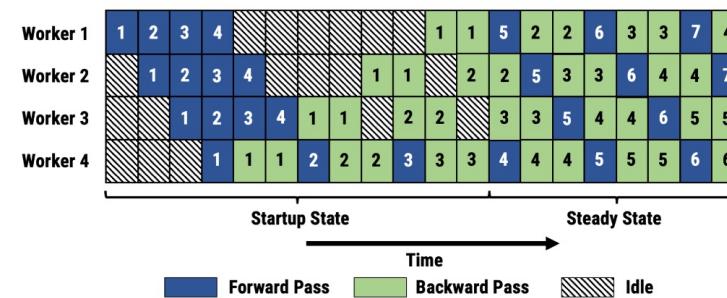
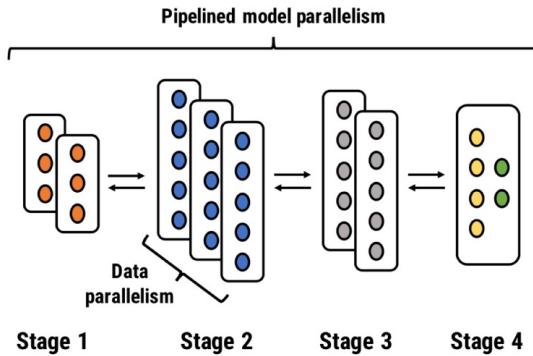


<https://www.microsoft.com/en-us/research/blog/zero-deepspeed-new-system-optimizations-enable-training-models-with-over-100-billion-parameters/>

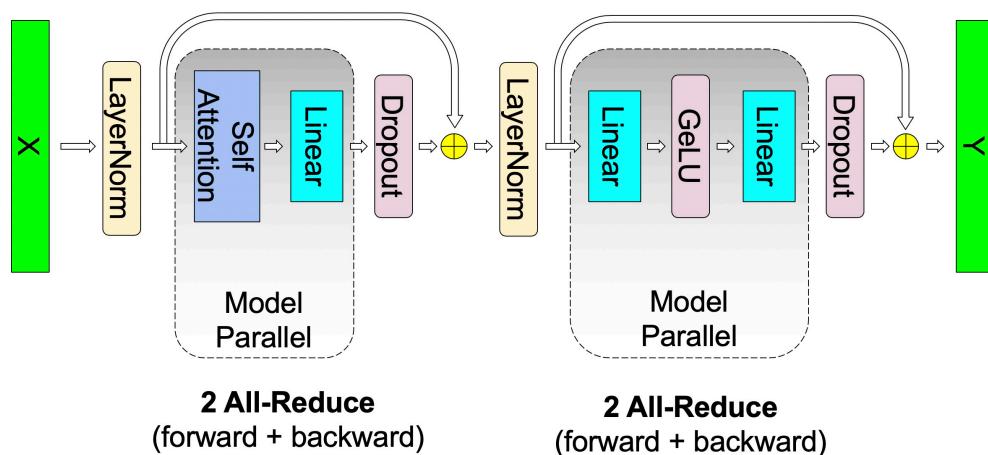
# Example 2: Facebook DLRM



# More recent examples



**PipeDream (Microsoft)**



**MegatronLM (NVIDIA)**

NVIDIA DEVELOPER
[HOME](#)
[BLOG](#)
[FORUMS](#)
[DOCS](#)
[DOWNLOADS](#)
[TRAINING](#)

Brilliant Minds. Breakthrough Discoveries. GTC | November 8 - 11, 2020

[SUBSCRIBE](#)

**DEVELOPER BLOG**

Oct 11, 2021 English ▾

TECHNICAL WALKTHROUGH

By Paresh Kharya and Ali Alvi

[Discuss \[0\]](#)
[Share](#)
[0 Like](#)

Tags: Conversational AI / NLP, DGX SuperPOD, HPC / Supercomputing, Megatron, Technical Walkthrough

Using DeepSpeed and Megatron to Train Megatron-Turing NLG 530B, the World's Largest and Most Powerful Generative Language Model

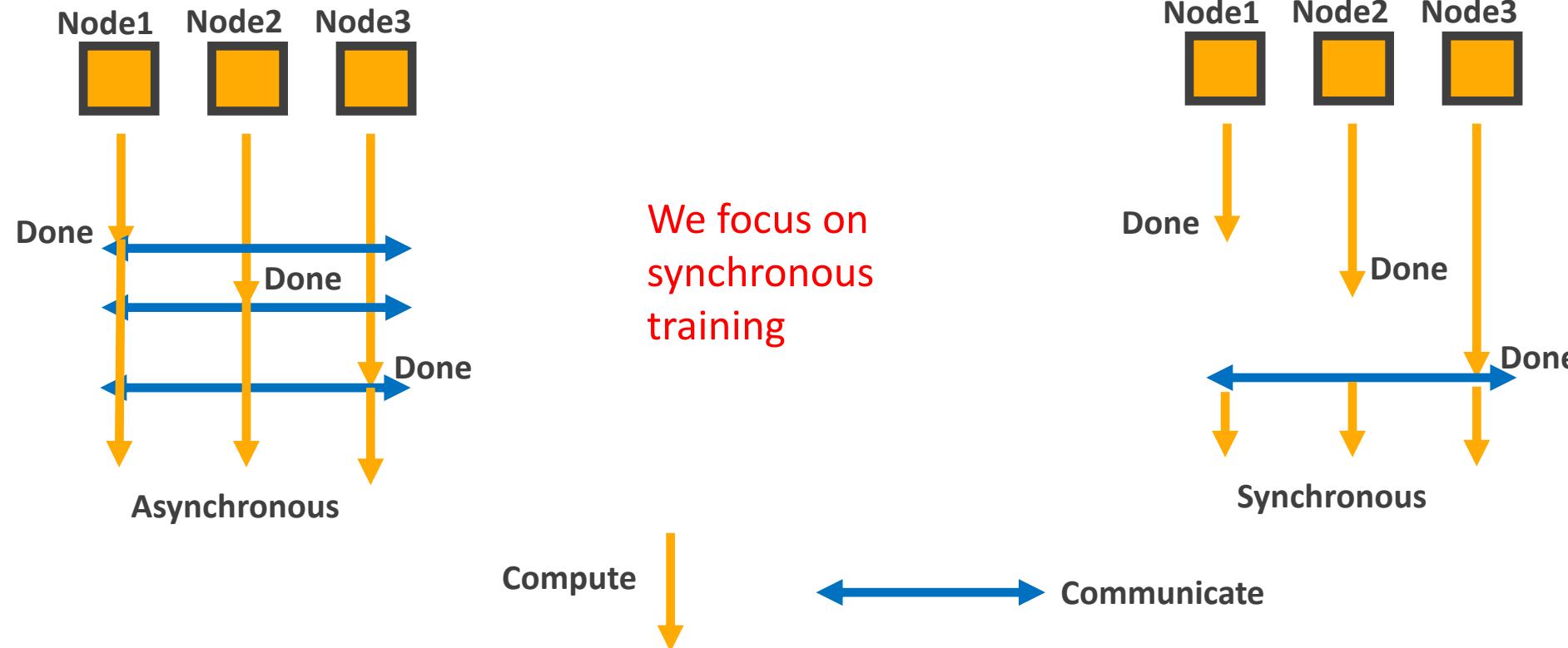
Model	Year	Size (billions of parameters)
ELMo	2018	0.1
BERT-Large	2019	~0.3
GPT-2	2019	1.5B
T5	2020	11B
Megatron-LM (8.3B)	2019	~8.3B
GPT-3 (175B)	2020	175B
Megatron-Turing NLG (530B)	2021	530B
Turing-NLG (17.2B)	2021	17.2B

# Model Parameter Update Mechanisms

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

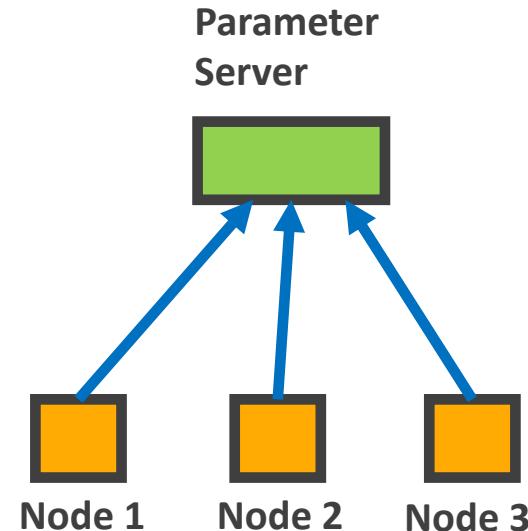
# Synchronization: Sync. vs. Async. Training

- Defines when nodes should exchange data
  - Affects convergence time

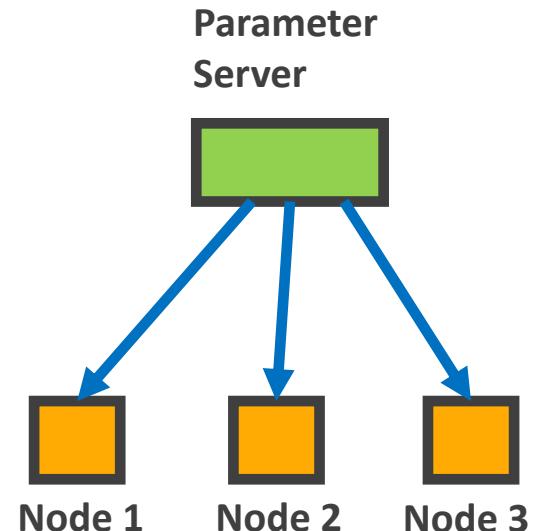


# Communication Handling

- Parameter Server



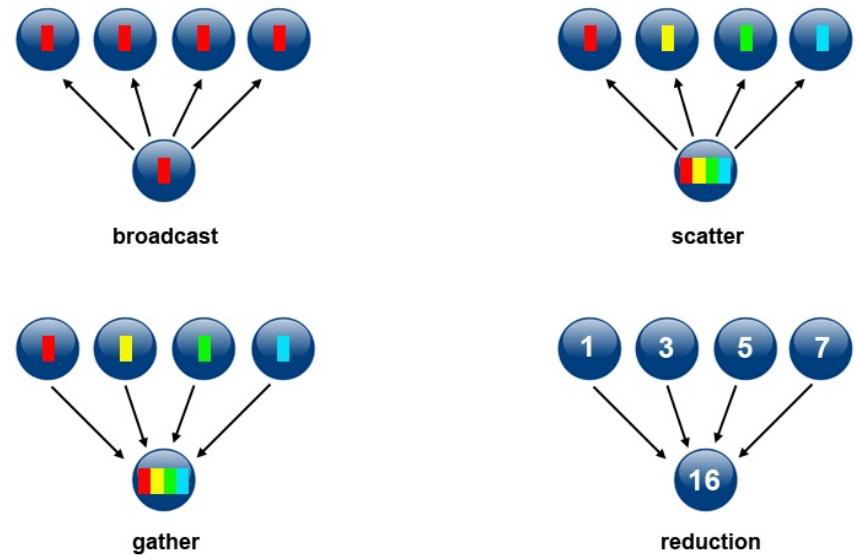
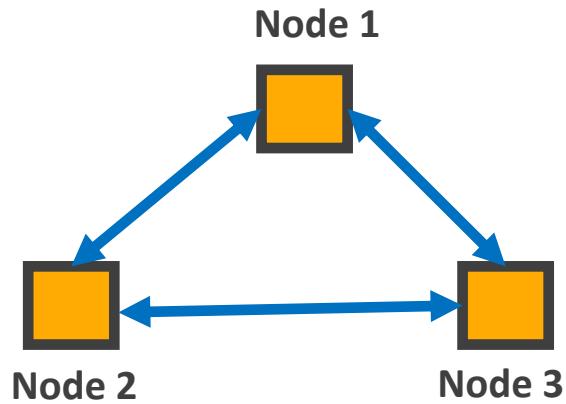
**Step 1:** Each node sends its model gradients to the parameter server to be reduced with other gradients and update the model



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

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

# Different Kinds of Collective Algorithms

- **Reduce-Scatter:**

- Used during input-output exchange due to model-parallelism
- Implementation Algorithms: **Ring-Based, Direct-based, etc.**

- **All-Gather:**

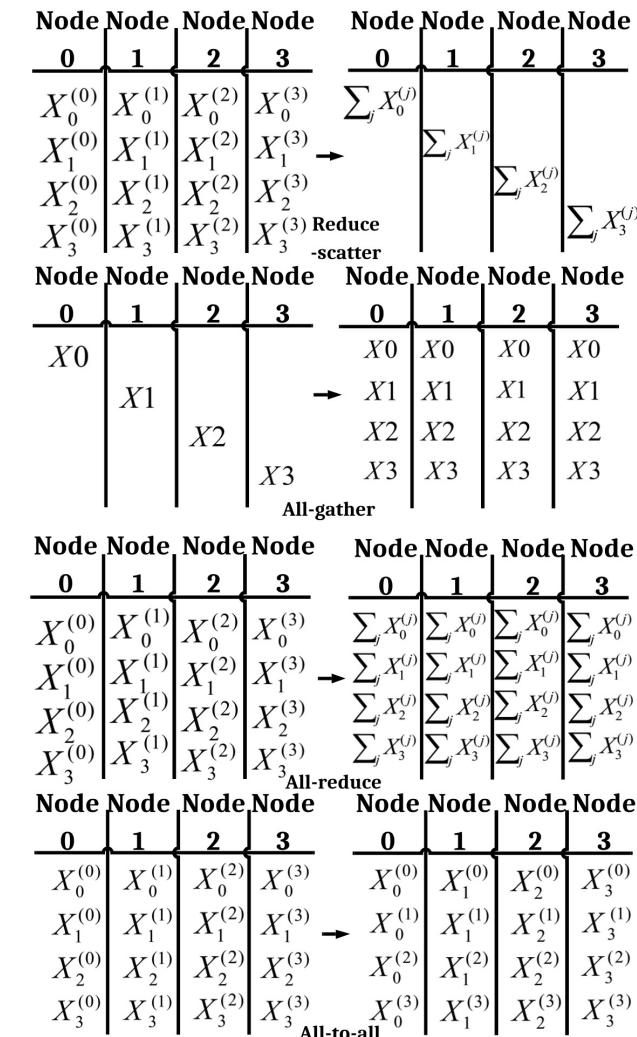
- Used during input-output exchange due to model-parallelism
- Implementation Algorithms: **Ring-Based, Direct-based, etc.**

- **All-Reduce (Reduce-Scatter + All-Gather):**

- Used during input-output exchange due to model-parallelism, or during model-parameter update.
- Implementation Algorithms: **Ring-Based, Direct-based, Tree-based, Halving-doubling, etc..**

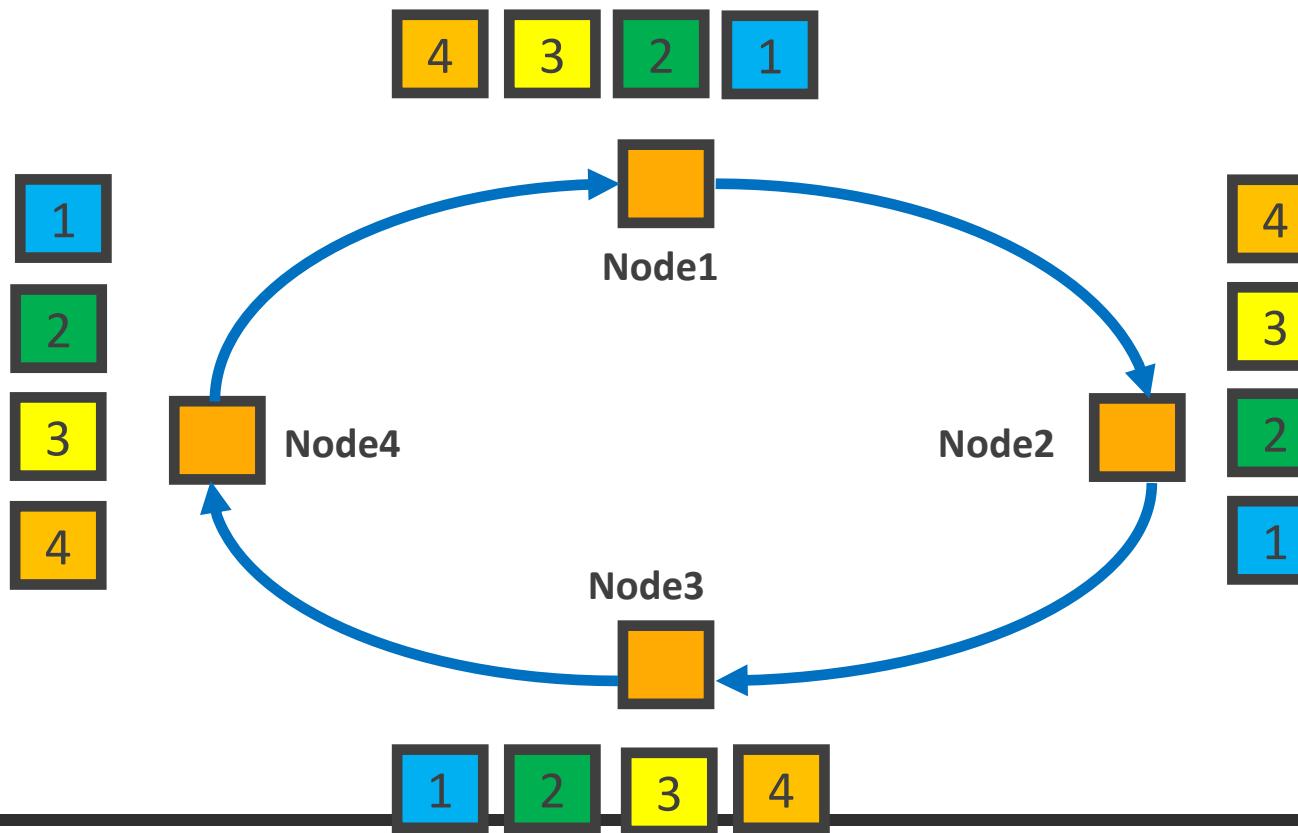
- **All-To-All:**

- Used during input-output exchange due to model-parallelism (e.g., distributed embedding layer on DLRM DNN.).
- Implementation Algorithms: **Direct-based, Ring-Based, etc..**



# Example: Ring Based All-Reduce

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



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

Reduce -scatter

| Node  |
|-------|-------|-------|-------|-------|-------|-------|-------|
| 0     | 1     | 2     | 3     | 0     | 1     | 2     | 3     |
| $X_0$ |
| $X_1$ |
| $X_2$ |
| $X_3$ |

All-gather

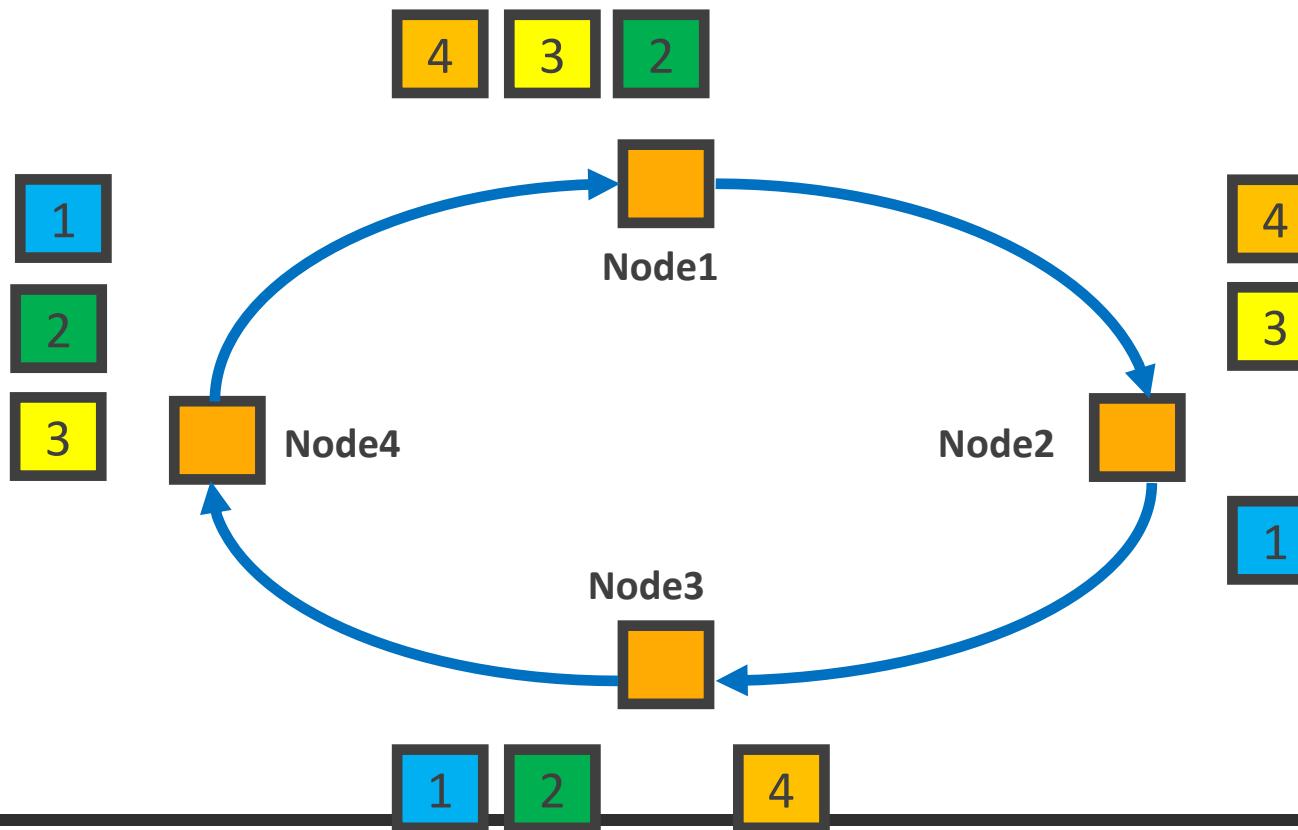
  

Node	Node	Node	Node	Node	Node	Node	Node
0	1	2	3	0	1	2	3
$X_0^{(0)}$	$X_0^{(1)}$	$X_0^{(2)}$	$X_0^{(3)}$	$\sum_j X_0^{(j)}$	$\sum_j X_0^{(j)}$	$\sum_j X_0^{(j)}$	$\sum_j X_0^{(j)}$
$X_1^{(0)}$	$X_1^{(1)}$	$X_1^{(2)}$	$X_1^{(3)}$		$\sum_j X_1^{(j)}$	$\sum_j X_1^{(j)}$	$\sum_j X_1^{(j)}$
$X_2^{(0)}$	$X_2^{(1)}$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

All-reduce

# Example: Ring Based All-Reduce

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



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

Reduce-scatter

| Node  |
|-------|-------|-------|-------|-------|-------|-------|-------|
| 0     | 1     | 2     | 3     | 0     | 1     | 2     | 3     |
| $X_0$ |
| $X_1$ |
| $X_2$ |
| $X_3$ |

All-gather

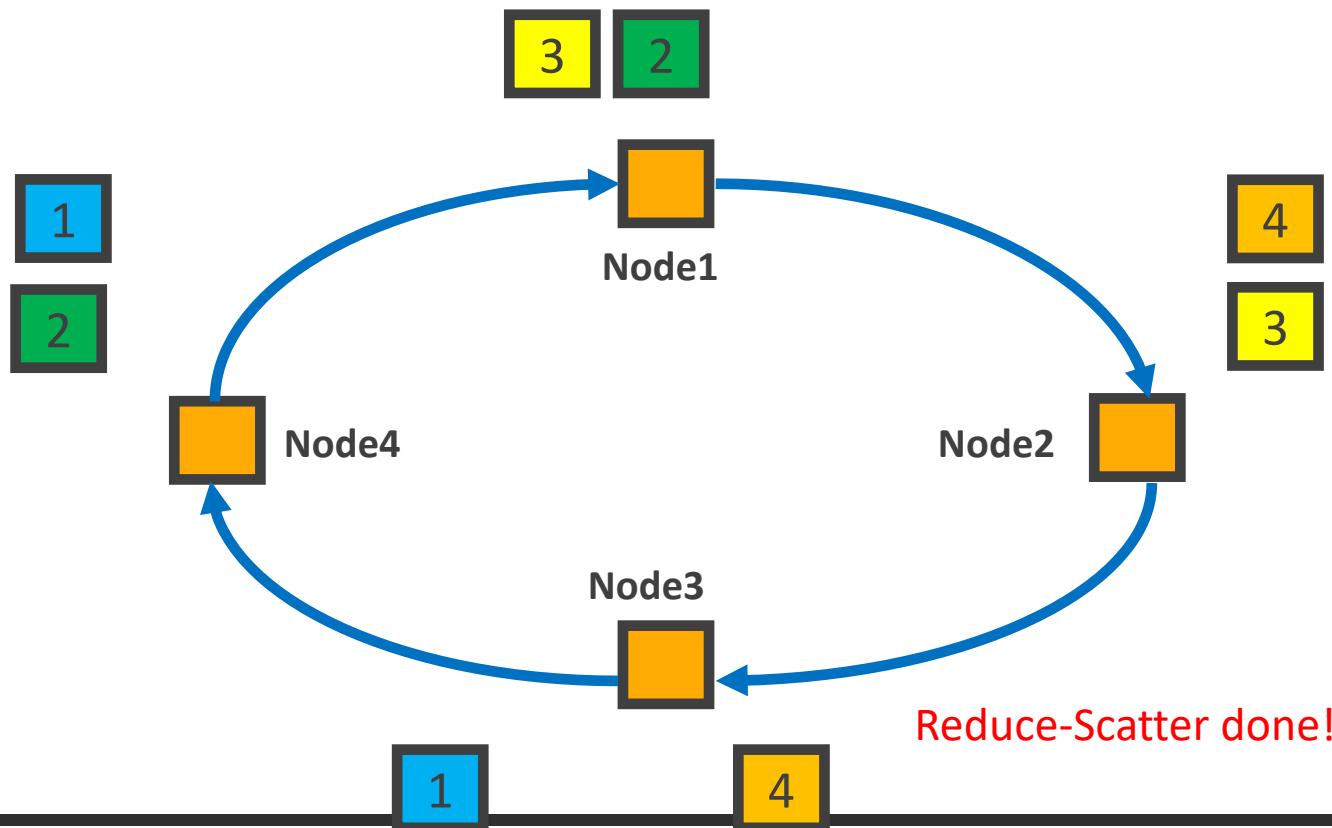
  

Node	Node	Node	Node	Node	Node	Node	Node
0	1	2	3	0	1	2	3
$X_0^{(0)}$	$X_0^{(1)}$	$X_0^{(2)}$	$X_0^{(3)}$	$\sum_j X_0^{(j)}$	$\sum_j X_0^{(j)}$	$\sum_j X_0^{(j)}$	$\sum_j X_0^{(j)}$
$X_1^{(0)}$	$X_1^{(1)}$	$X_1^{(2)}$	$X_1^{(3)}$		$\sum_j X_1^{(j)}$	$\sum_j X_1^{(j)}$	$\sum_j X_1^{(j)}$
$X_2^{(0)}$	$X_2^{(1)}$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

All-reduce

# Example: Ring Based All-Reduce

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



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

Reduce-scatter

| Node  |
|-------|-------|-------|-------|-------|-------|-------|-------|
| 0     | 1     | 2     | 3     | 0     | 1     | 2     | 3     |
| $X_0$ |       |       |       | $X_0$ | $X_0$ | $X_0$ | $X_0$ |
|       | $X_1$ |       |       | $X_1$ | $X_1$ | $X_1$ | $X_1$ |
|       |       | $X_2$ |       | $X_2$ | $X_2$ | $X_2$ | $X_2$ |
|       |       |       | $X_3$ | $X_3$ | $X_3$ | $X_3$ | $X_3$ |

All-gather

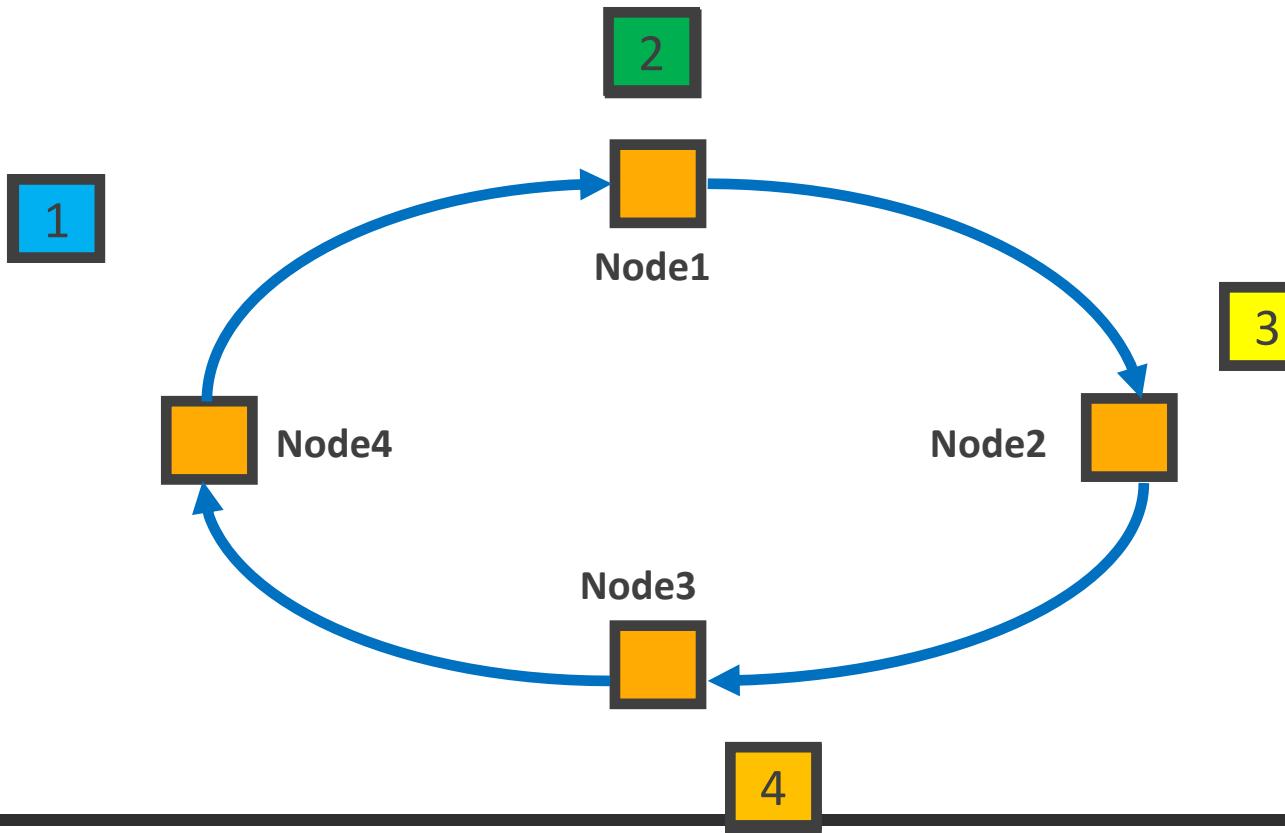
  

Node	Node	Node	Node	Node	Node	Node	Node
0	1	2	3	0	1	2	3
$X_0^{(0)}$	$X_0^{(1)}$	$X_0^{(2)}$	$X_0^{(3)}$	$\sum_j X_0^{(j)}$	$\sum_j X_1^{(j)}$	$\sum_j X_2^{(j)}$	$\sum_j X_3^{(j)}$
$X_1^{(0)}$	$X_1^{(1)}$	$X_1^{(2)}$	$X_1^{(3)}$		$\sum_j X_1^{(j)}$	$\sum_j X_2^{(j)}$	$\sum_j X_3^{(j)}$
$X_2^{(0)}$	$X_2^{(1)}$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

All-reduce

# Example: Ring Based All-Reduce

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



Node	Node	Node	Node	Node	Node	Node	Node	Node	Node	Node	Node
0	1	2	3	0	1	2	3	0	1	2	3
X <sub>0</sub> <sup>(0)</sup>	X <sub>0</sub> <sup>(1)</sup>	X <sub>0</sub> <sup>(2)</sup>	X <sub>0</sub> <sup>(3)</sup>	$\sum_j X_0^{(j)}$							
X <sub>1</sub> <sup>(0)</sup>	X <sub>1</sub> <sup>(1)</sup>	X <sub>1</sub> <sup>(2)</sup>	X <sub>1</sub> <sup>(3)</sup>		$\sum_j X_1^{(j)}$						
X <sub>2</sub> <sup>(0)</sup>	X <sub>2</sub> <sup>(1)</sup>	X <sub>2</sub> <sup>(2)</sup>	X <sub>2</sub> <sup>(3)</sup>			$\sum_j X_2^{(j)}$					
X <sub>3</sub> <sup>(0)</sup>	X <sub>3</sub> <sup>(1)</sup>	X <sub>3</sub> <sup>(2)</sup>	X <sub>3</sub> <sup>(3)</sup>				$\sum_j X_3^{(j)}$				

Reduce-scatter

| Node |
|------|------|------|------|------|------|------|------|------|------|------|------|
| 0    | 1    | 2    | 3    | 0    | 1    | 2    | 3    | 0    | 1    | 2    | 3    |
| X0   |      |      |      | X0   | X0   | X0   | X0   |      |      |      |      |
|      | X1   |      |      |      | X1   | X1   | X1   | X1   |      |      |      |
|      |      | X2   |      |      |      | X2   | X2   | X2   | X2   |      |      |
|      |      |      | X3   |      |      |      | X3   | X3   | X3   | X3   |      |

All-gather

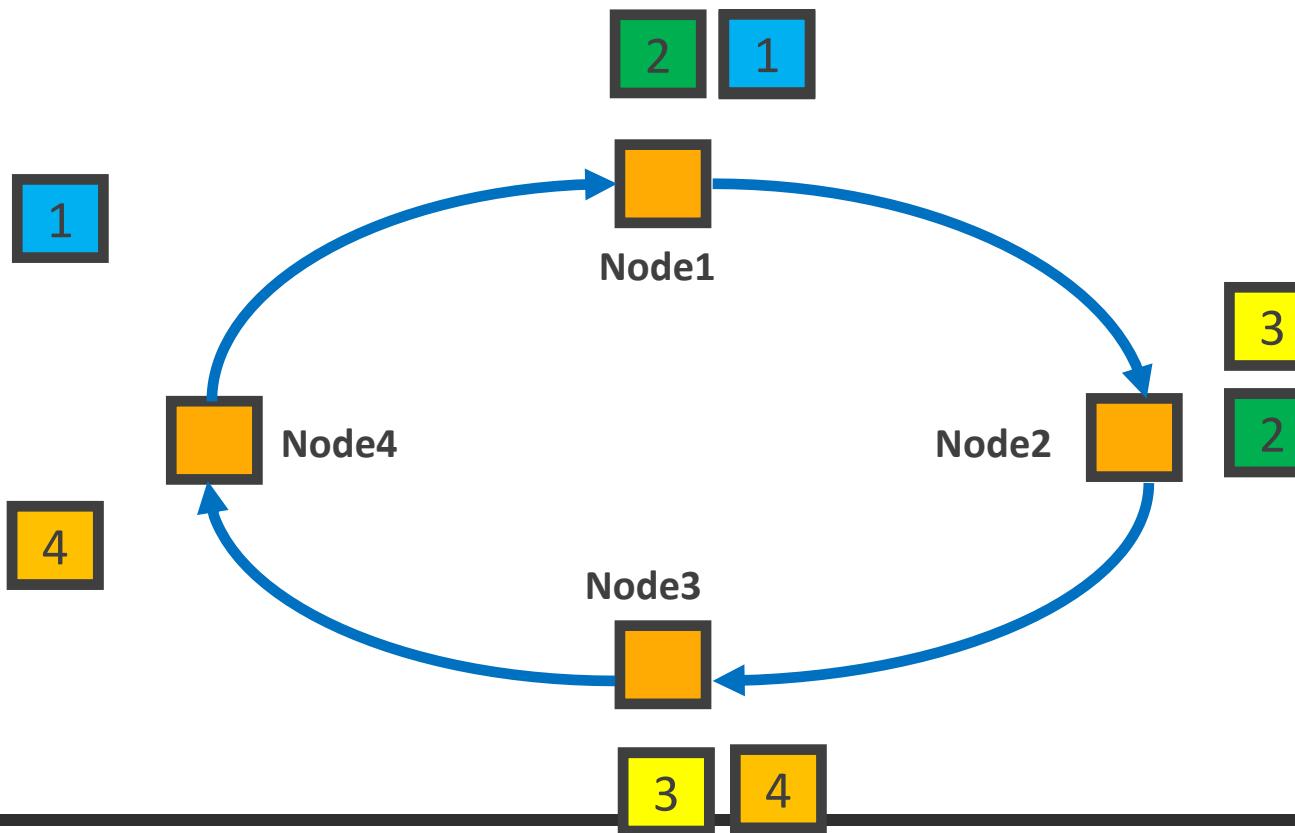
  

Node	Node	Node	Node	Node	Node	Node	Node	Node	Node	Node	Node
0	1	2	3	0	1	2	3	0	1	2	3
X <sub>0</sub> <sup>(0)</sup>	X <sub>0</sub> <sup>(1)</sup>	X <sub>0</sub> <sup>(2)</sup>	X <sub>0</sub> <sup>(3)</sup>	$\sum_j X_0^{(j)}$							
X <sub>1</sub> <sup>(0)</sup>	X <sub>1</sub> <sup>(1)</sup>	X <sub>1</sub> <sup>(2)</sup>	X <sub>1</sub> <sup>(3)</sup>		$\sum_j X_1^{(j)}$						
X <sub>2</sub> <sup>(0)</sup>	X <sub>2</sub> <sup>(1)</sup>	X <sub>2</sub> <sup>(2)</sup>	X <sub>2</sub> <sup>(3)</sup>			$\sum_j X_2^{(j)}$					
X <sub>3</sub> <sup>(0)</sup>	X <sub>3</sub> <sup>(1)</sup>	X <sub>3</sub> <sup>(2)</sup>	X <sub>3</sub> <sup>(3)</sup>				$\sum_j X_3^{(j)}$				

All-reduce

# Example: Ring Based All-Reduce

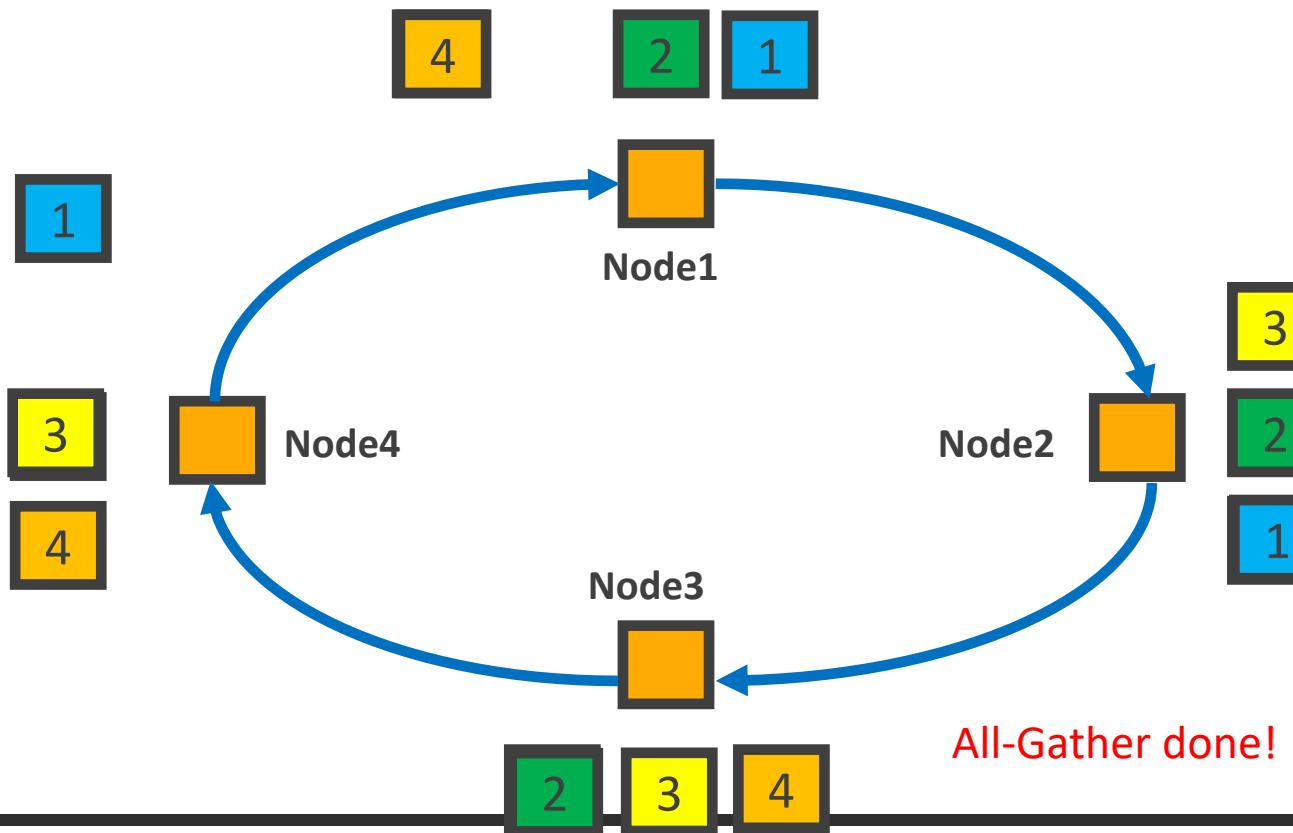
- A ring with N nodes partitions data to N messages
- Collective Communication Flow:



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

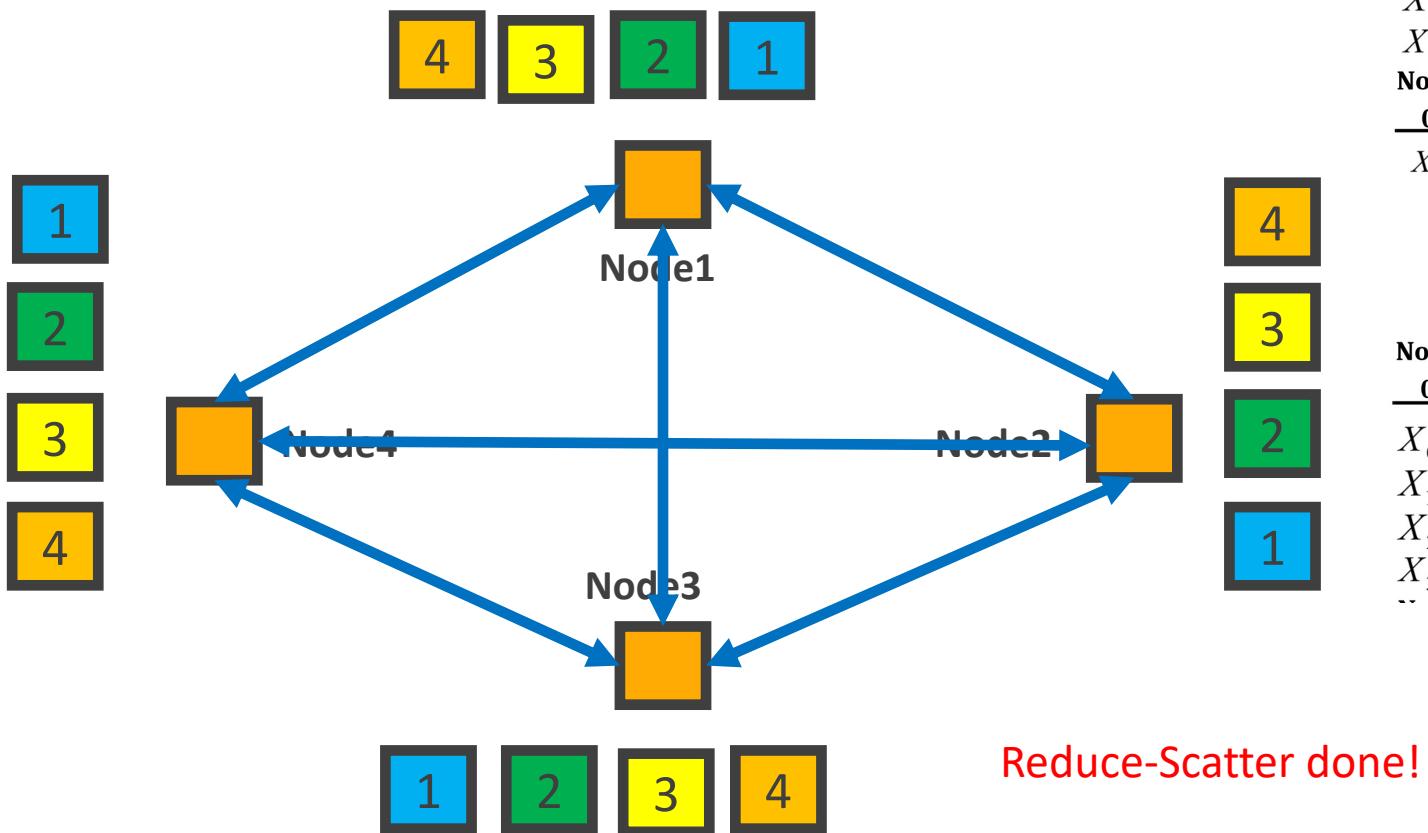
# Example: Ring Based All-Reduce

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



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

# Example: Direct All-Reduce



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

Reduce-scatter

| Node  |
|-------|-------|-------|-------|-------|-------|-------|-------|
| 0     | 1     | 2     | 3     | 0     | 1     | 2     | 3     |
| $X_0$ |
| $X_1$ |
| $X_2$ |
| $X_3$ |

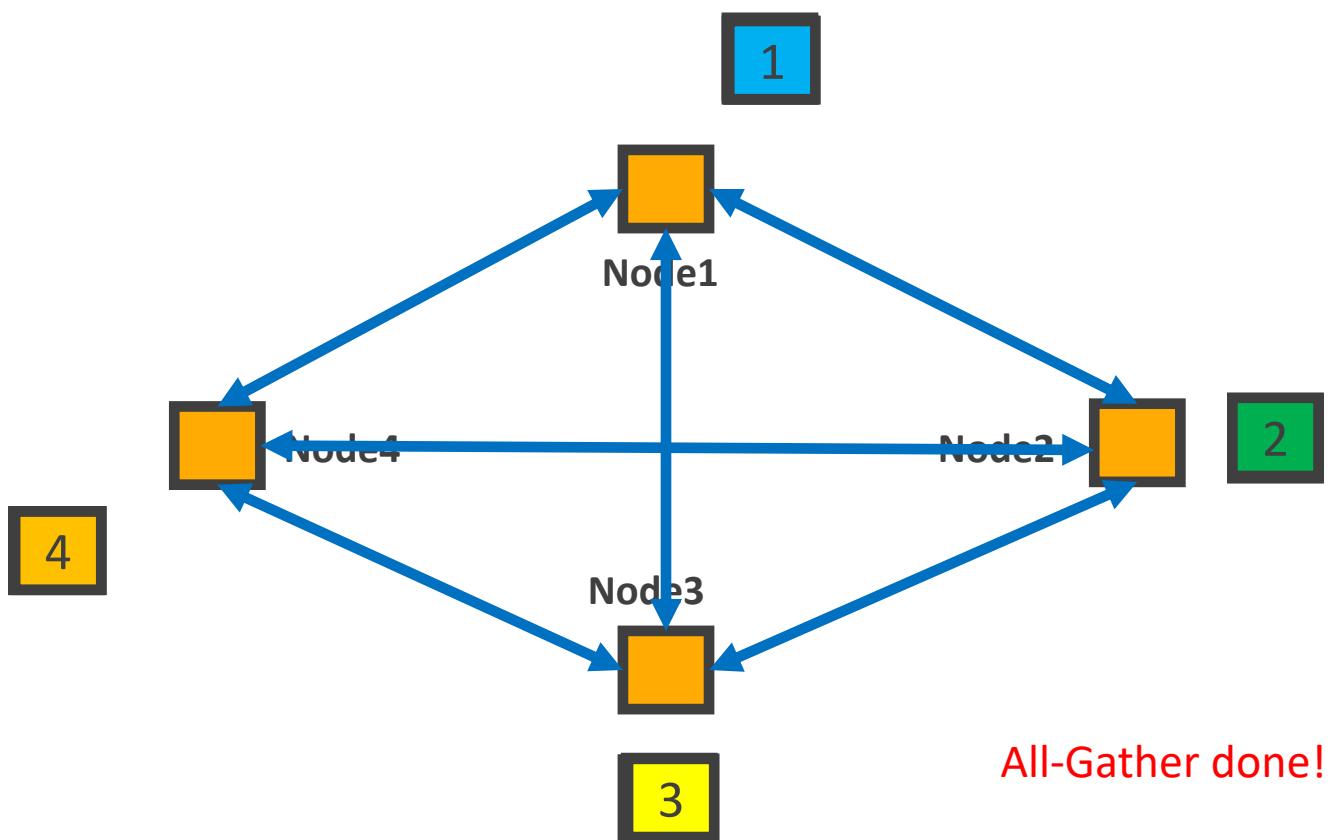
All-gather

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

All-reduce

# Example: Direct All-Reduce



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

Reduce -scatter

| Node  |
|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| 0     | 1     | 2     | 3     | 0     | 1     | 2     | 3     |       |       |
| $X_0$ |       |       |       | $X_0$ | $X_0$ | $X_0$ | $X_0$ |       |       |
|       | $X_1$ |       |       |       | $X_1$ | $X_1$ | $X_1$ | $X_1$ |       |
|       |       | $X_2$ |       |       |       | $X_2$ | $X_2$ | $X_2$ | $X_2$ |
|       |       |       | $X_3$ |       |       |       | $X_3$ | $X_3$ | $X_3$ |

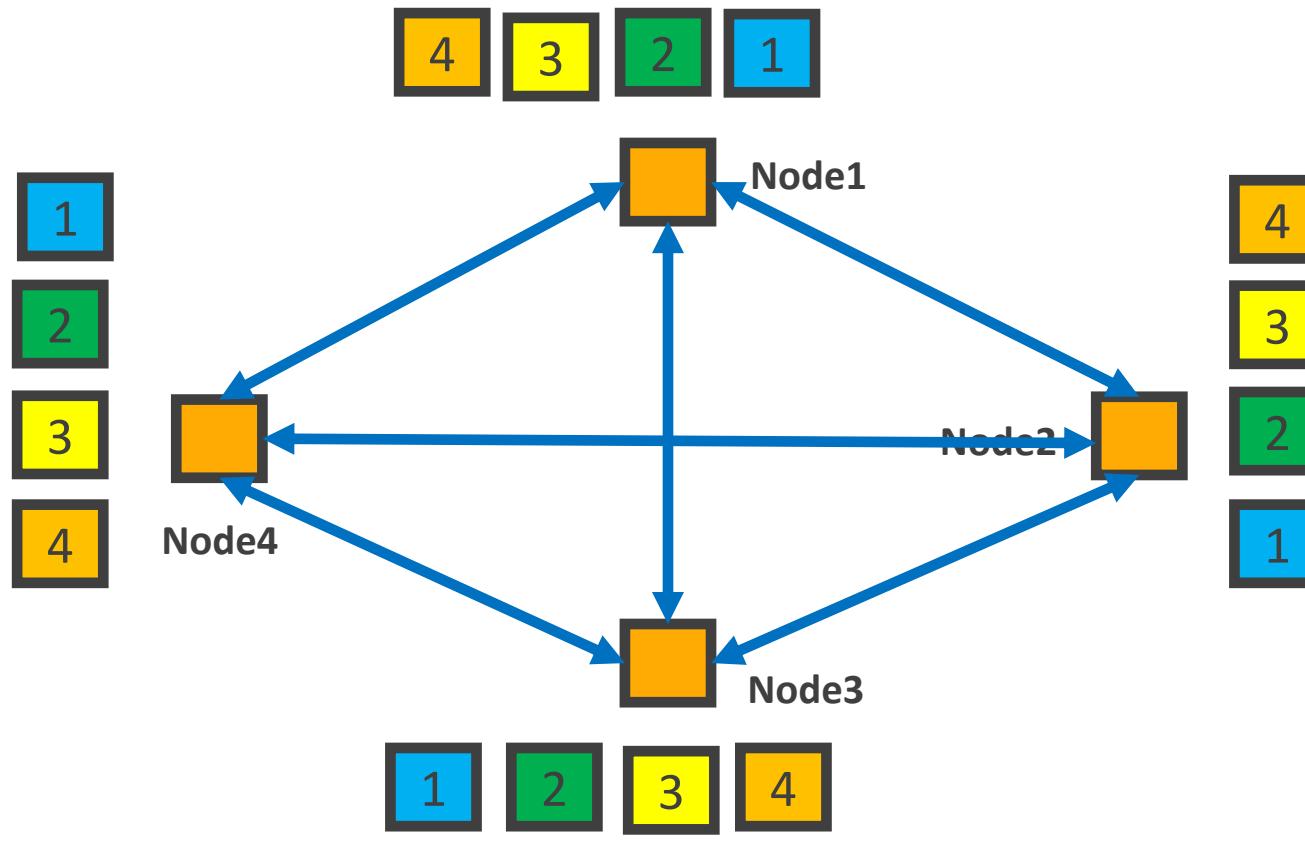
All-gather

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

All-reduce

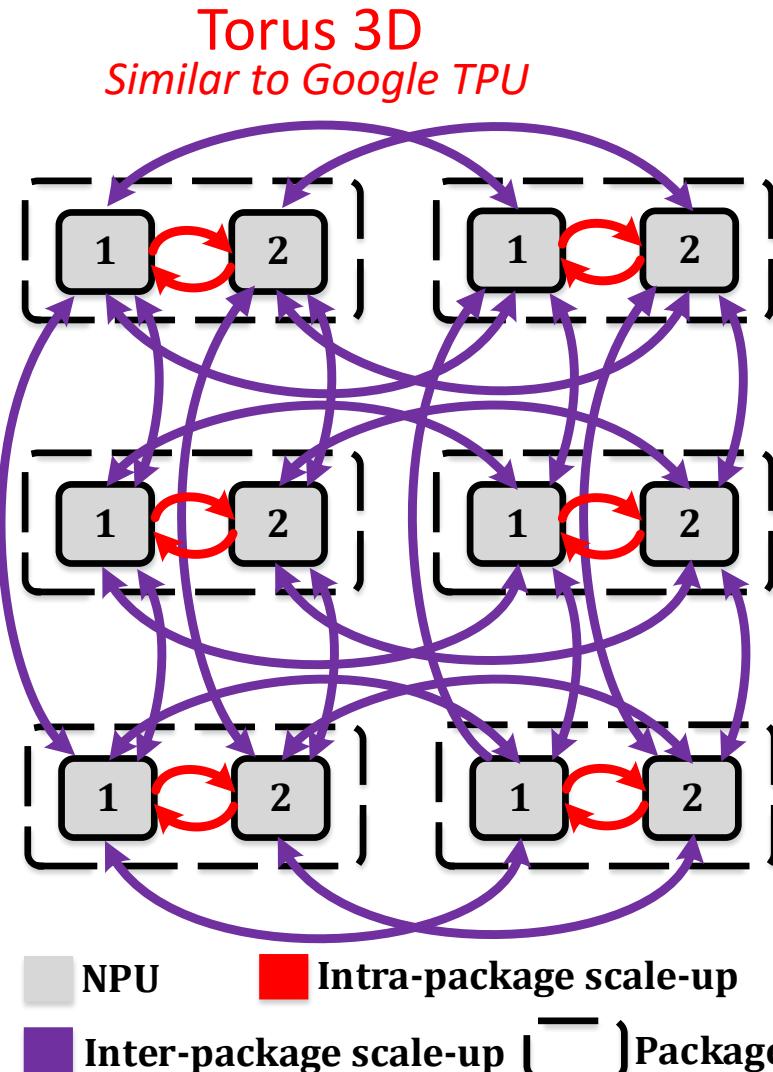
# Example: All-to-All



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

All-to-all

# Collectives on Sophisticated Training Platforms



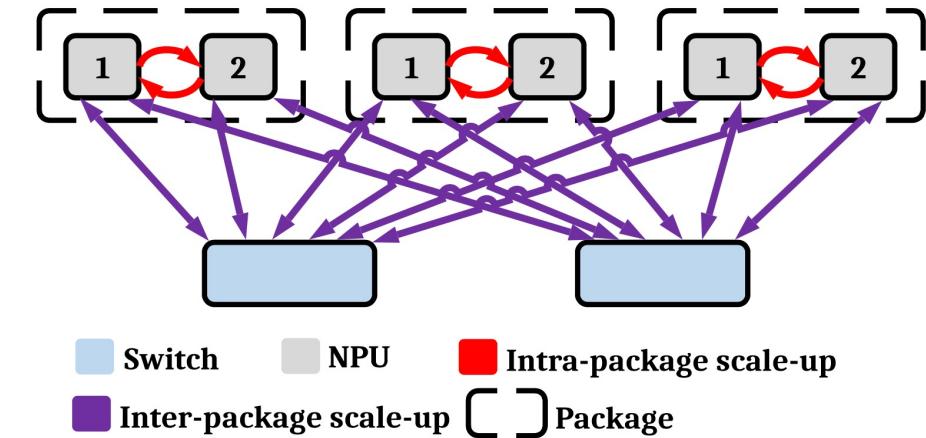
## Hierarchical all-reduce:

- Reduce-scatter **within package**
- All-reduce across rows
- All-reduce across columns
- All-gather **within package**

Heterogeneous Bandwidth

Multi-phase Collectives

**All-To-All**  
*Similar to NVIDIA DGX2*



## Hierarchical all-reduce:

- Reduce-scatter **within package**
- All-reduce across switch
- All-gather **within package**

# 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

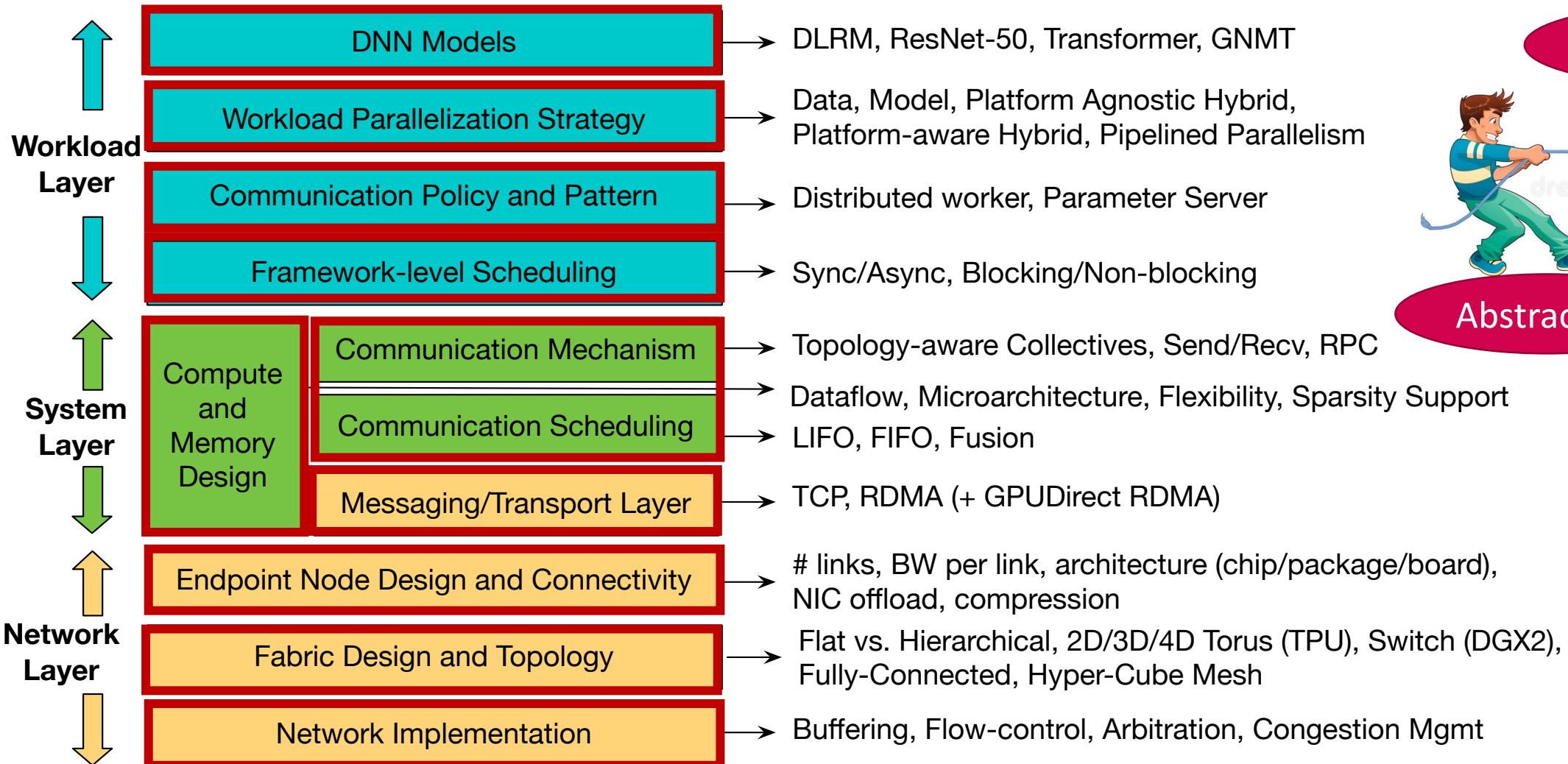
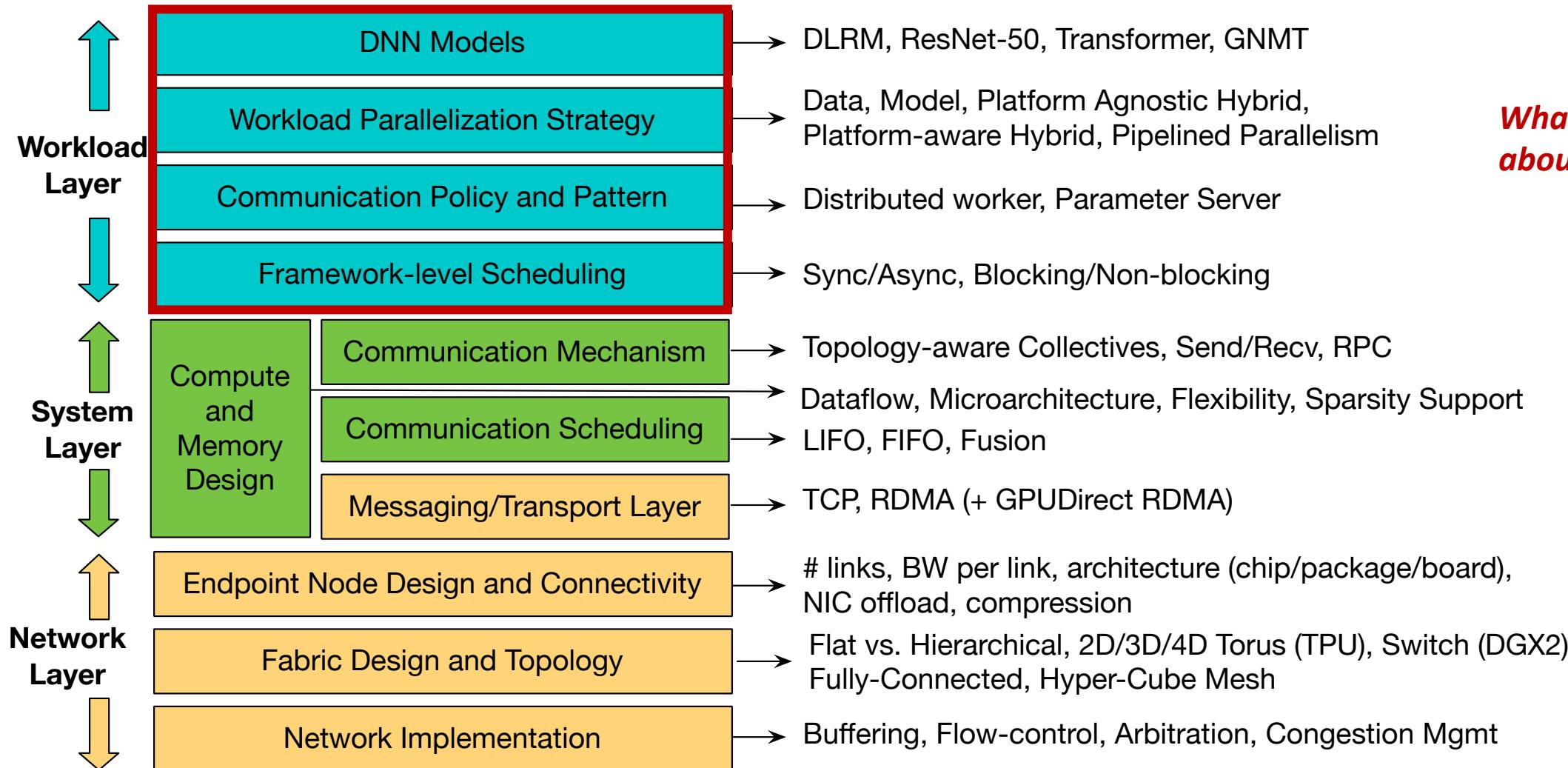


Figure Courtesy: Srinivas Sridharan (Facebook)

# Distributed Training Stack



**What we talked about so-far**

Figure Courtesy: Srinivas Sridharan (Facebook)

# Distributed Training Stack

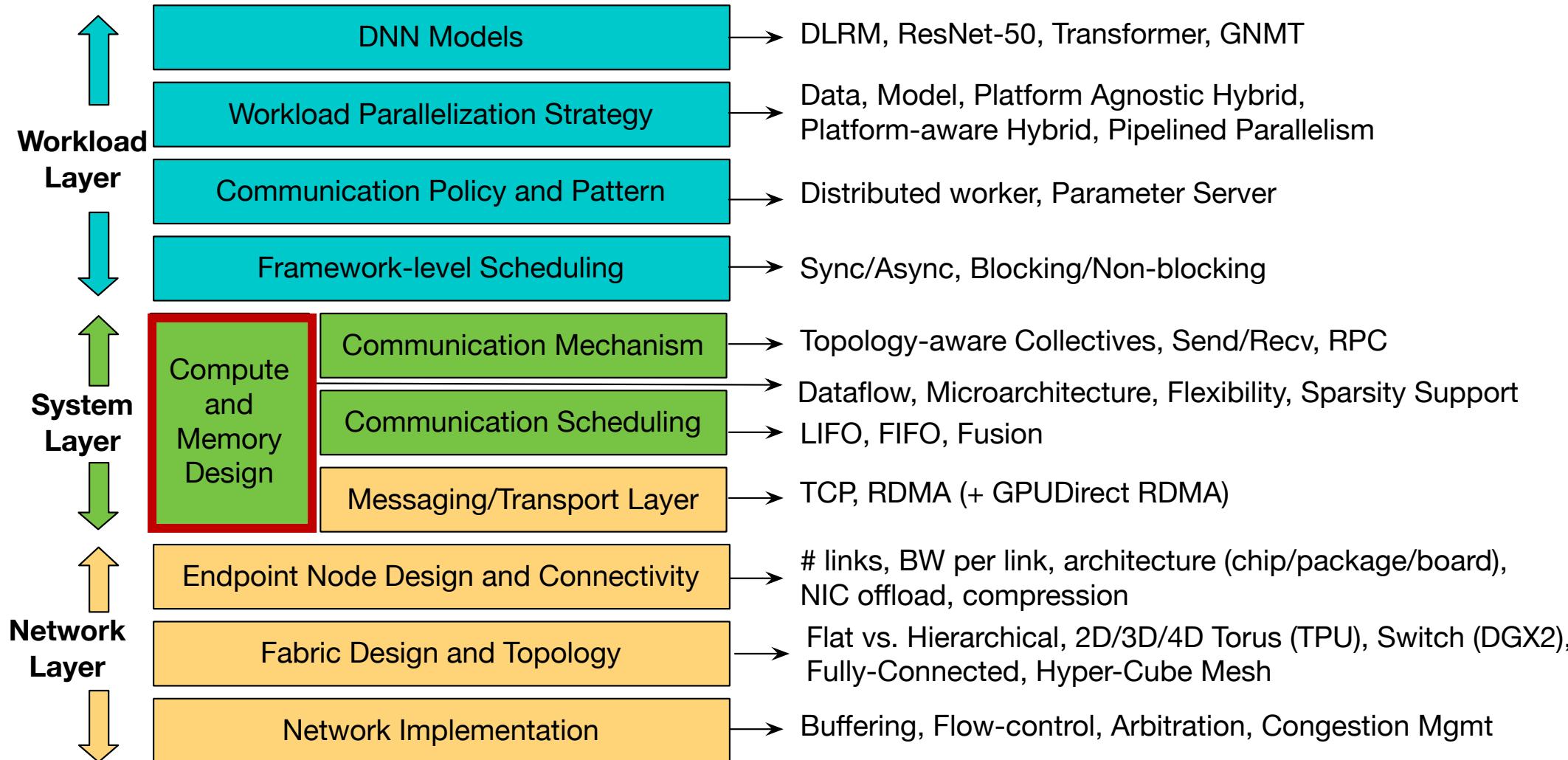
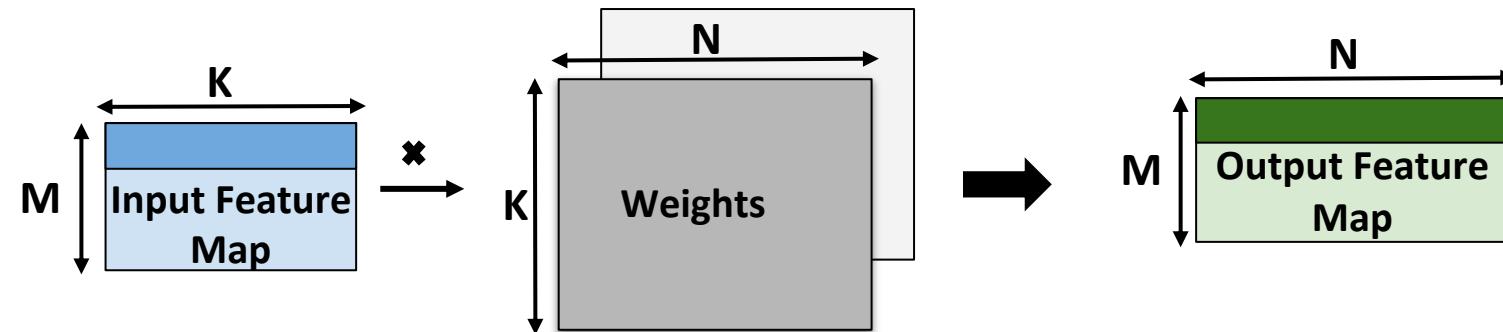


Figure Courtesy: Srinivas Sridharan (Facebook)

# DL Training: The Compute

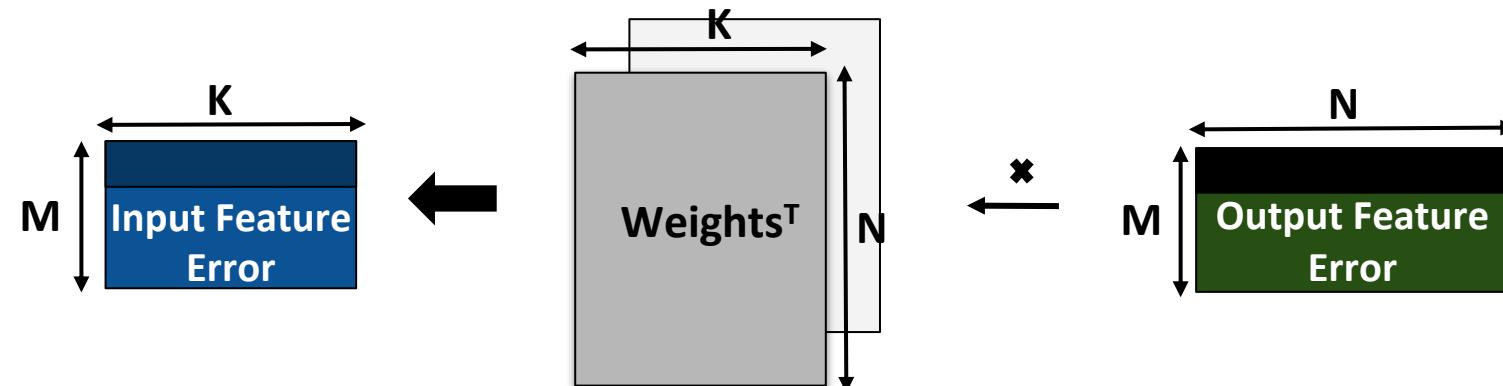
## Forward Pass (Inference and Training)



**GEMM MNK Dimension Representation**

M dim: batch size

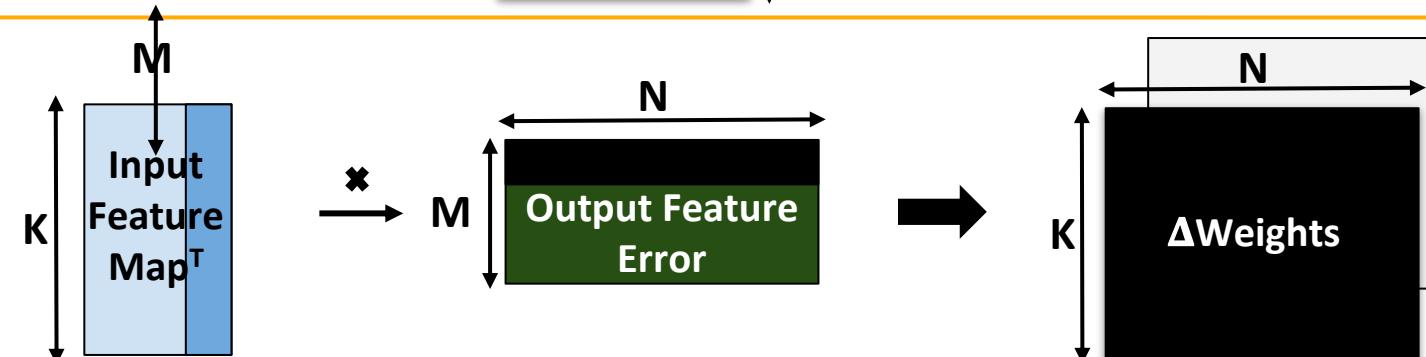
## Backward Pass (Training)



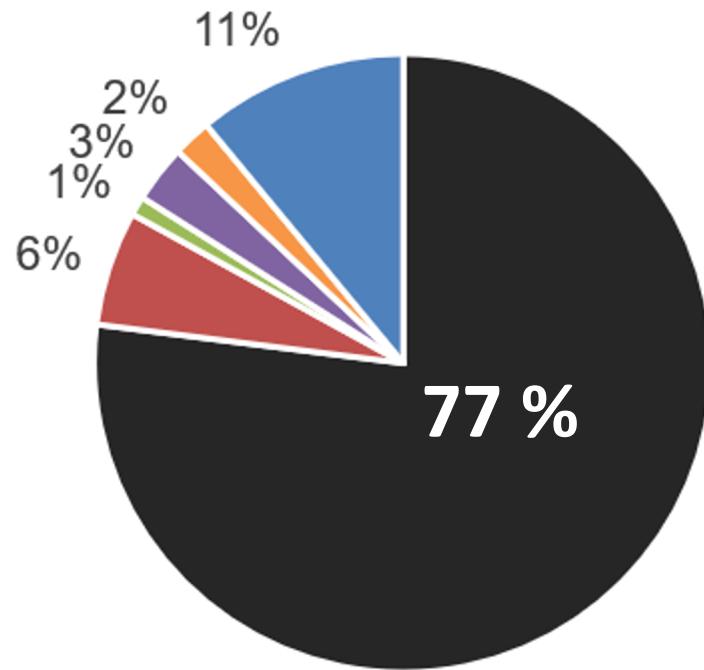
N dim: number of channels in the next layer

K dim:  $[H * W * C]$

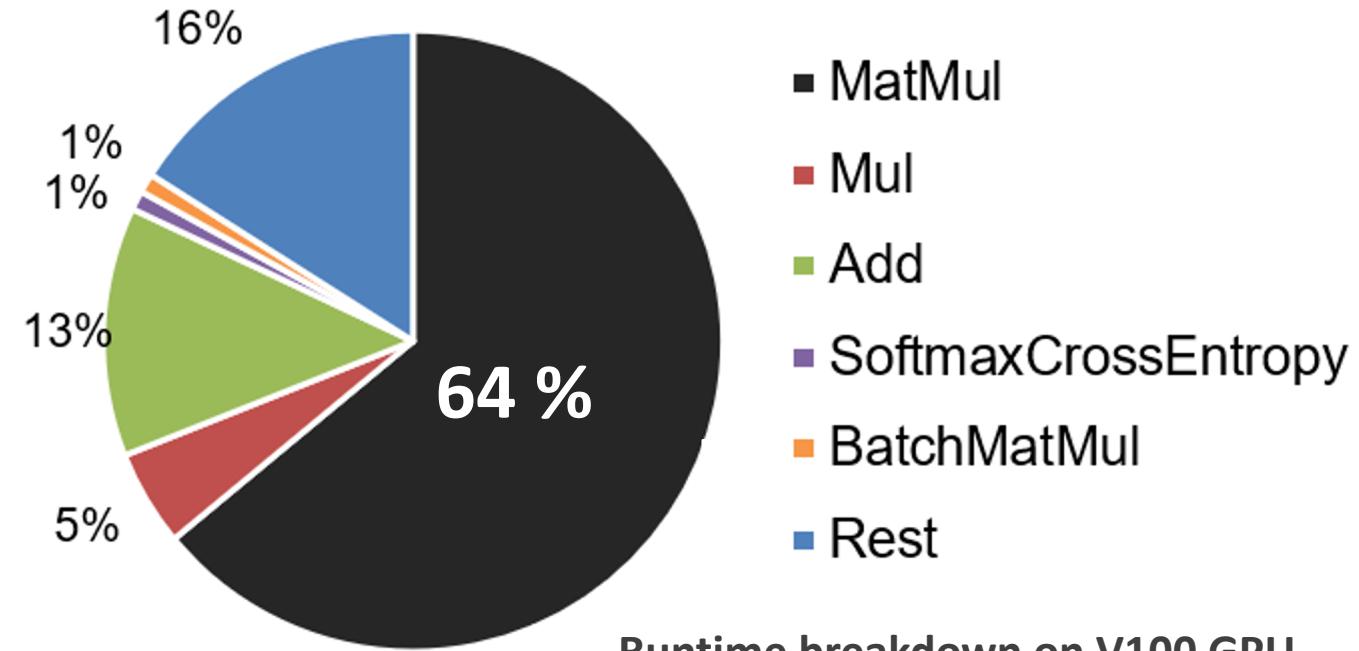
## Gradient Computation (Training)



# Key Compute Kernel during DL Training



**Transformer**  
**(Language Understanding)**



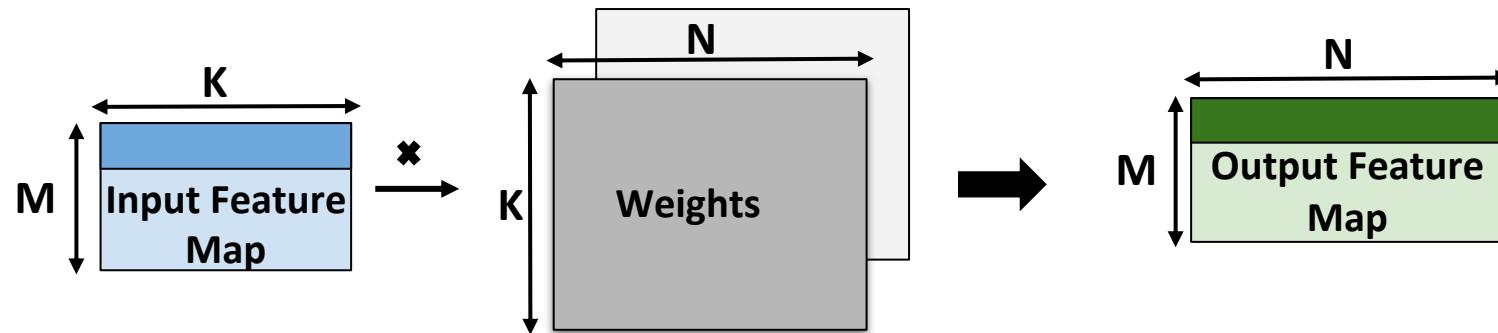
**GNMT**  
**(Machine Translation)**

Runtime breakdown on V100 GPU

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

# GEMMs in Deep Learning

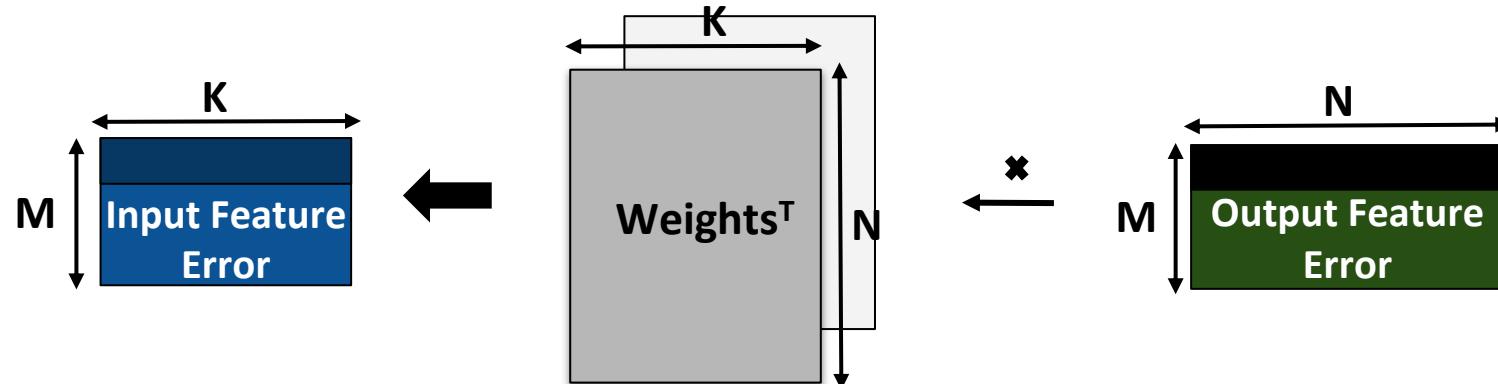
## Forward Pass (Inference and Training)



GEMM MNK  
Dimension  
Representation

M dim: batch size

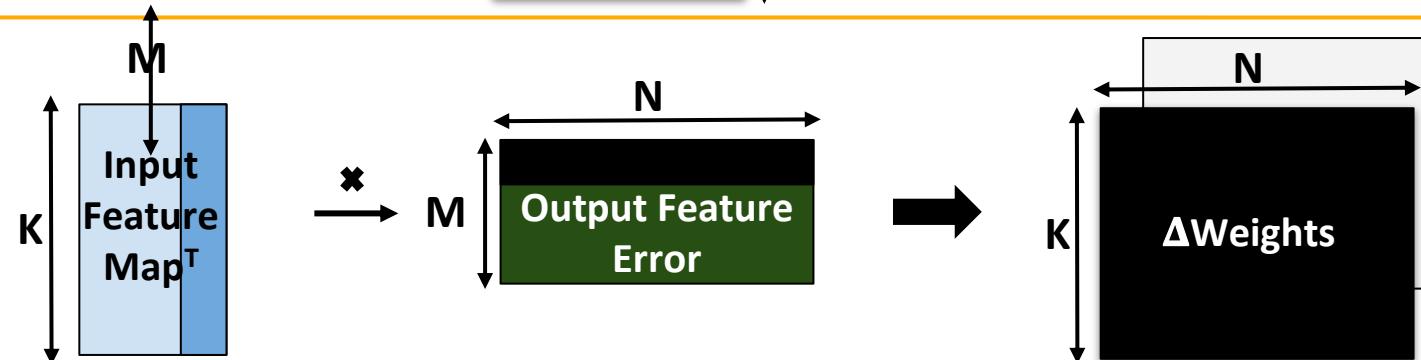
## Backward Pass (Training)



N dim: number of channels in the next layer

K dim:  $[H * W * C]$

## Gradient Computation (Training)

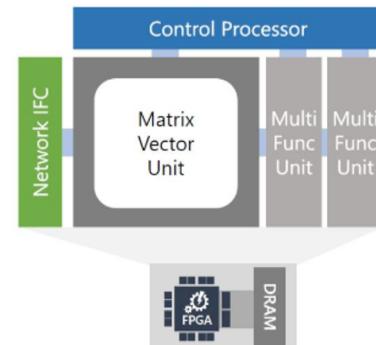


# Hardware for Accelerating GEMMs

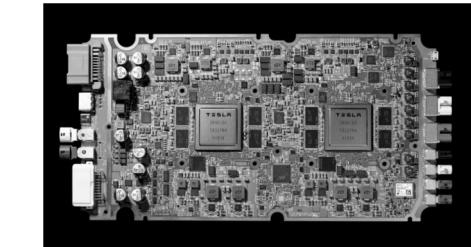
## SIMT Architectures



Nvidia GTX GPUs

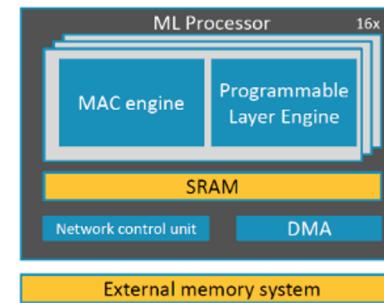


Microsoft Brainwave



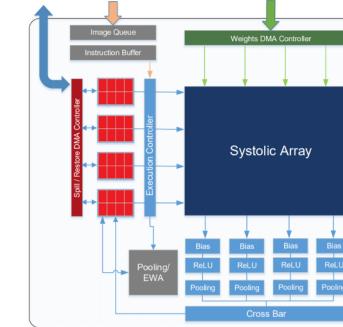
Tesla FSDC

## SIMD Architectures

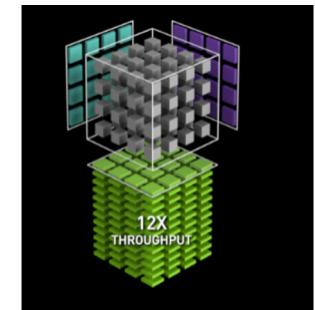


ARM Trillium

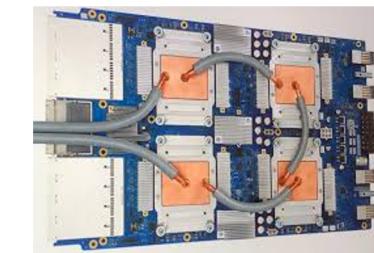
## Systolic Architectures



Xilinx xDNN



Nvidia Tensor Cores



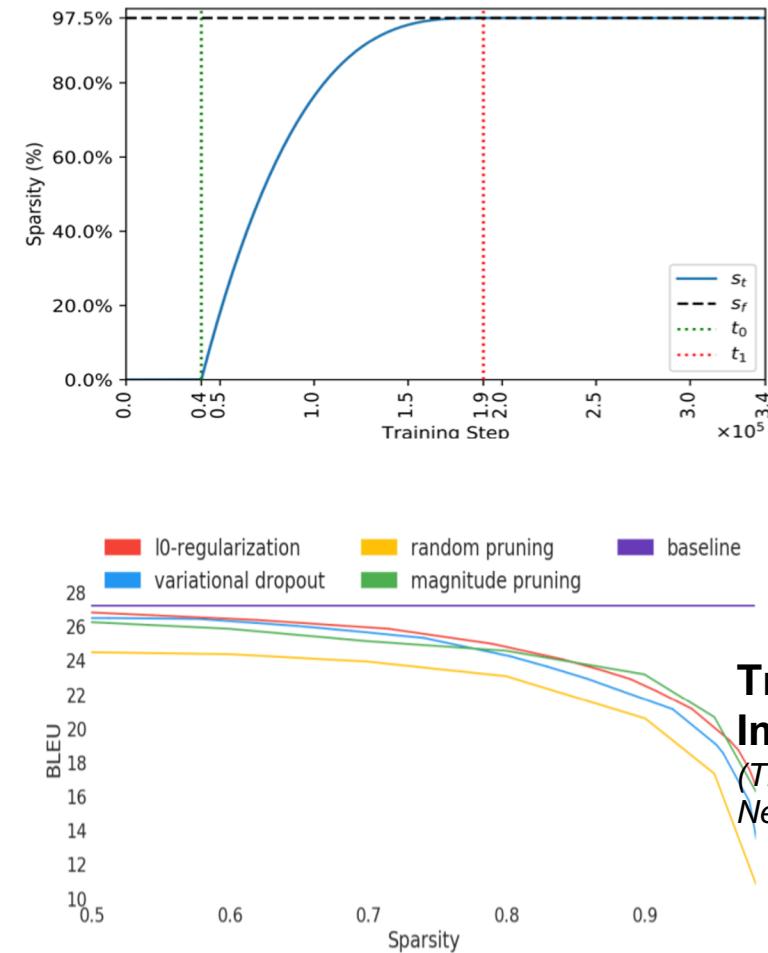
Google TPU

Key Feature: Specialized support for GEMMs

# GEMMs in Modern DL

Workload	Application	Example Dimensions		
		M	N	K
GNMT	Machine Translation	128	2048	4096
		320	3072	4096
		1632	36548	1024
		2048	4096	32
DeepBench	General Workload	1024	16	500000
		35	8457	2560
Transformer	Language Understanding	31999	1024	84
		84	1024	4096
NCF	Collaborative Filtering	2048	1	128
		256	256	2048

GEMMs are irregular (non-square)!

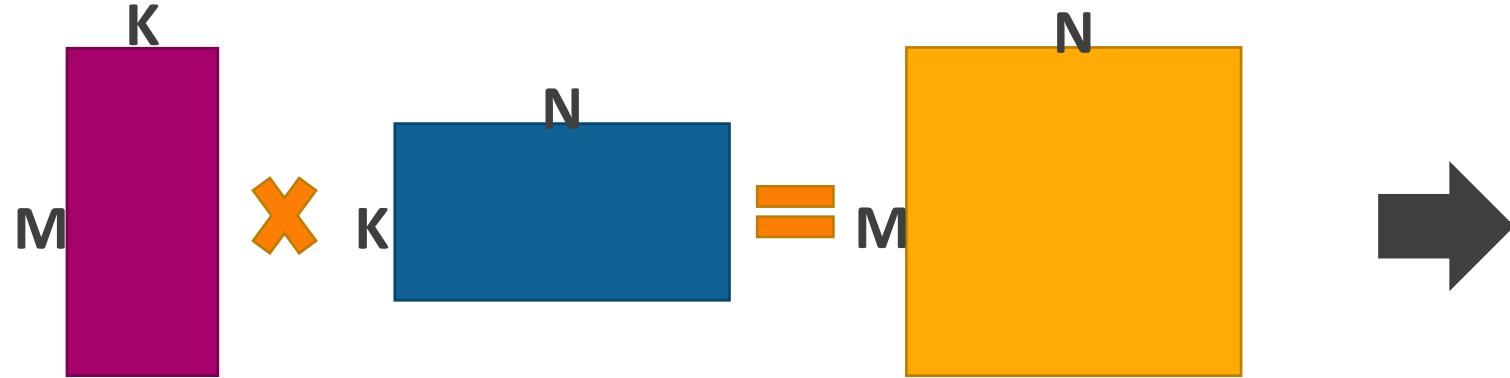


**GNMT Pruning - Temporal Sparsity**  
(<https://www.intel.ai/compressing-gnmt-models>)

**Transformer Sparsity - Impact on BLEU**  
(The State of Sparsity in Deep Neural Networks, Gale et al., arXiv)

GEMMs are Sparse! Weight sparsity ranges from 40% to 90%. Activation sparsity is approximately 30% to 70% from ReLU, dropout, etc.

# Mapping GEMMs on to TPU



**What determines *utilization*?**

Mapping Efficiency

**What determines *stalls*?**

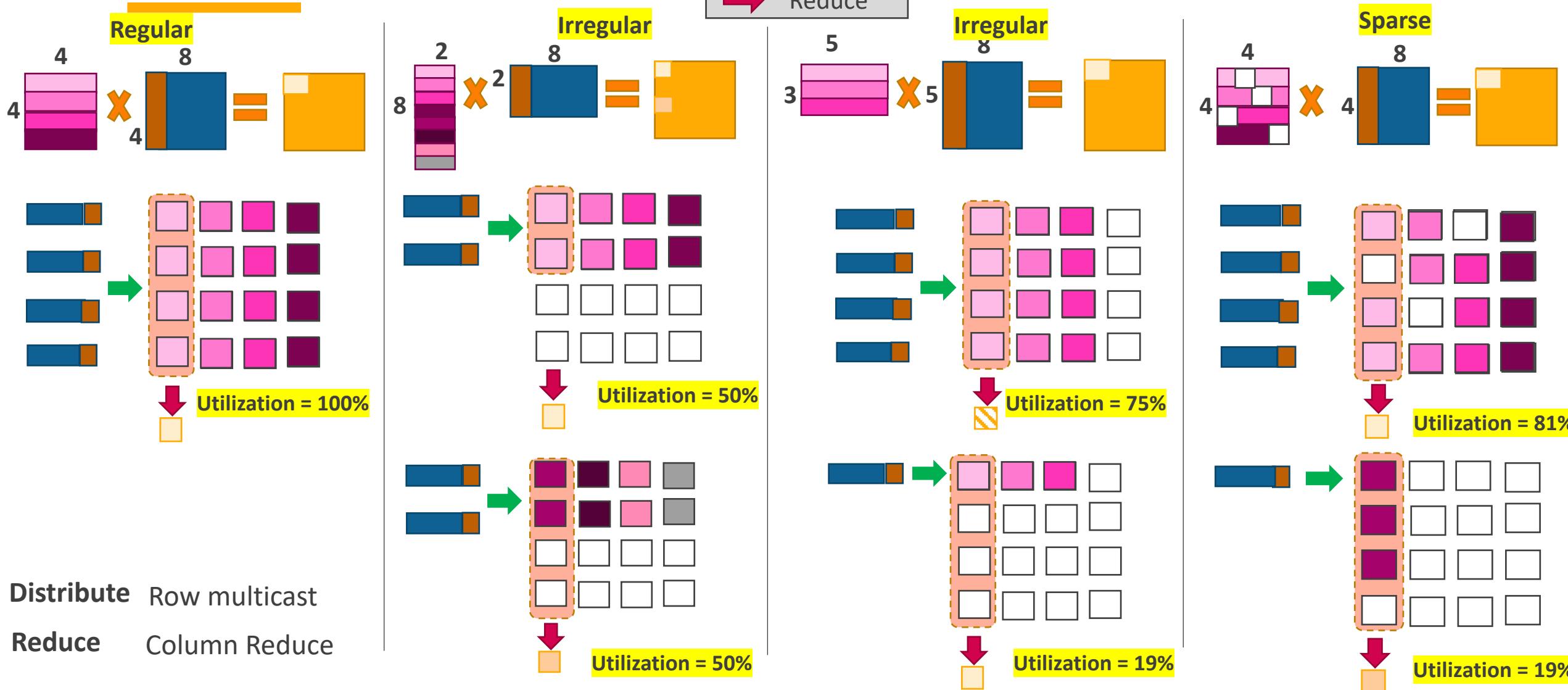
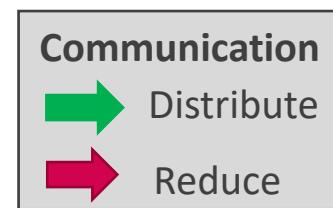
Memory/Interconnect Bandwidth

**TPU (Systolic Array)**

0	15 ..	31 ..	47 ..	63 ..	79 ..	95 ..	111 ..	127
15								
.								
31								
.								
47								
.								
63								
.								
79								
.								
95								
.								
111								
.								
127								

**\*\* Assuming MK matrix is streaming and KN matrix is stationary. (aka weight stationary)**

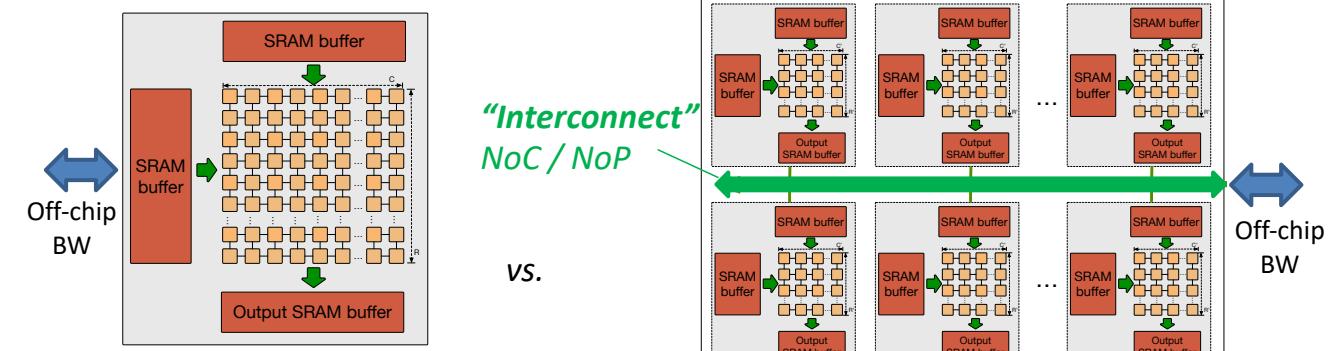
# Mapping Examples



# Mechanisms to increase utilization

- **Handling Irregular GEMMs**

- One large array (e.g., Google TPU) versus several smaller arrays (e.g., NVIDIA Tensor cores)
  - Trade-off: reuse vs utilization



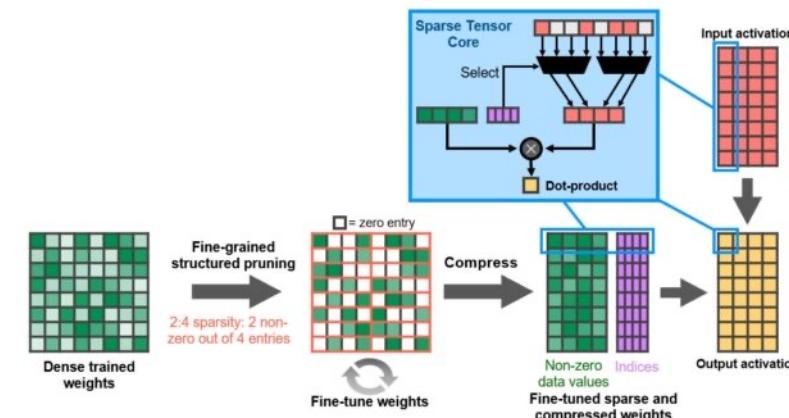
- **Handling Sparse GEMMs**

- Structured Sparsity Support

- E.g., NVIDIA A100

- Unstructured Sparsity Support

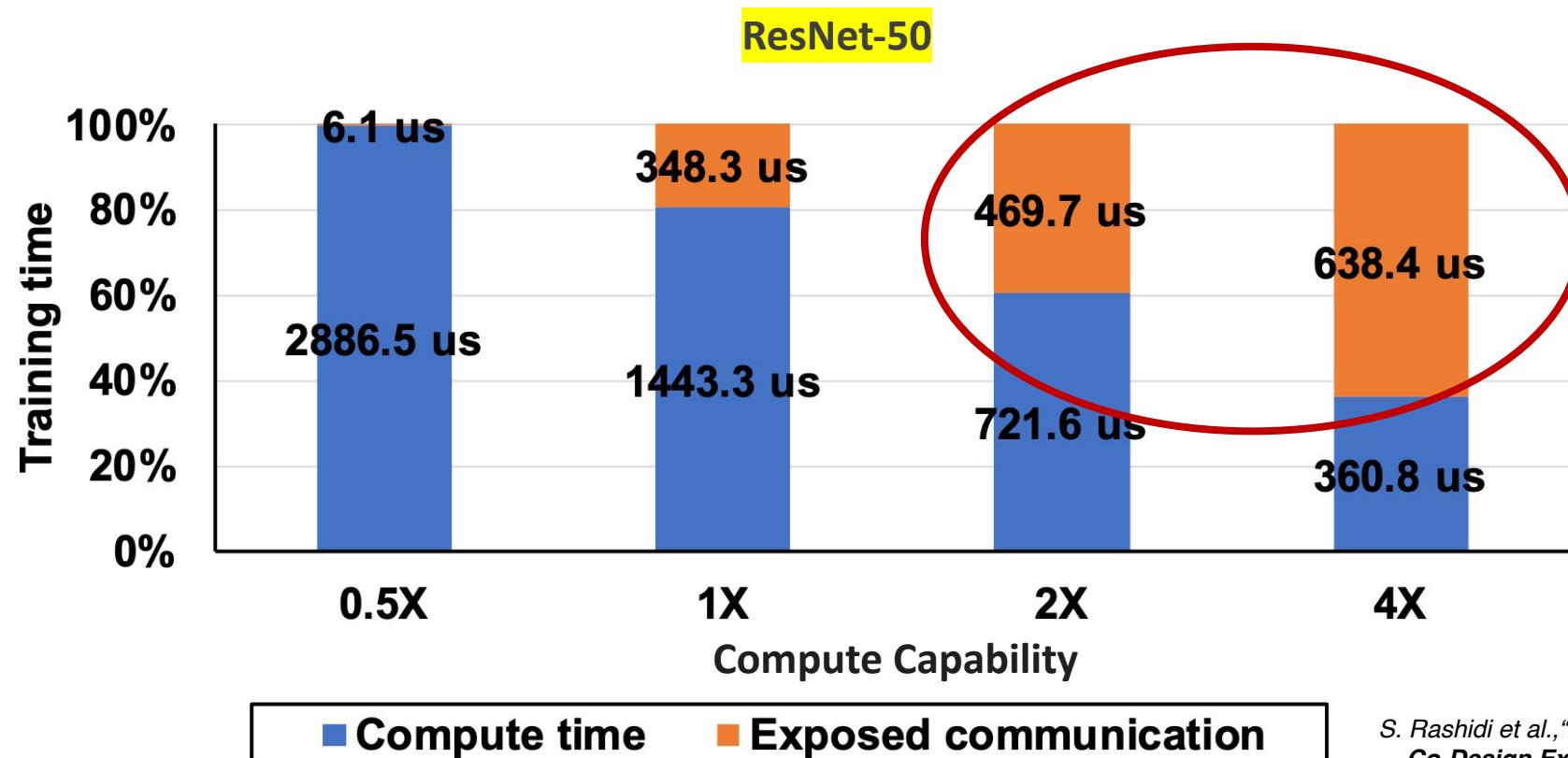
- Active research going on



**NVIDIA A100 supports 4:2 structured sparsity**

# Effect of Enhanced Compute Efficiency on Training

- A Torus 3D with total of 32 nodes (2X4X4) is used.



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

# Distributed Training Stack

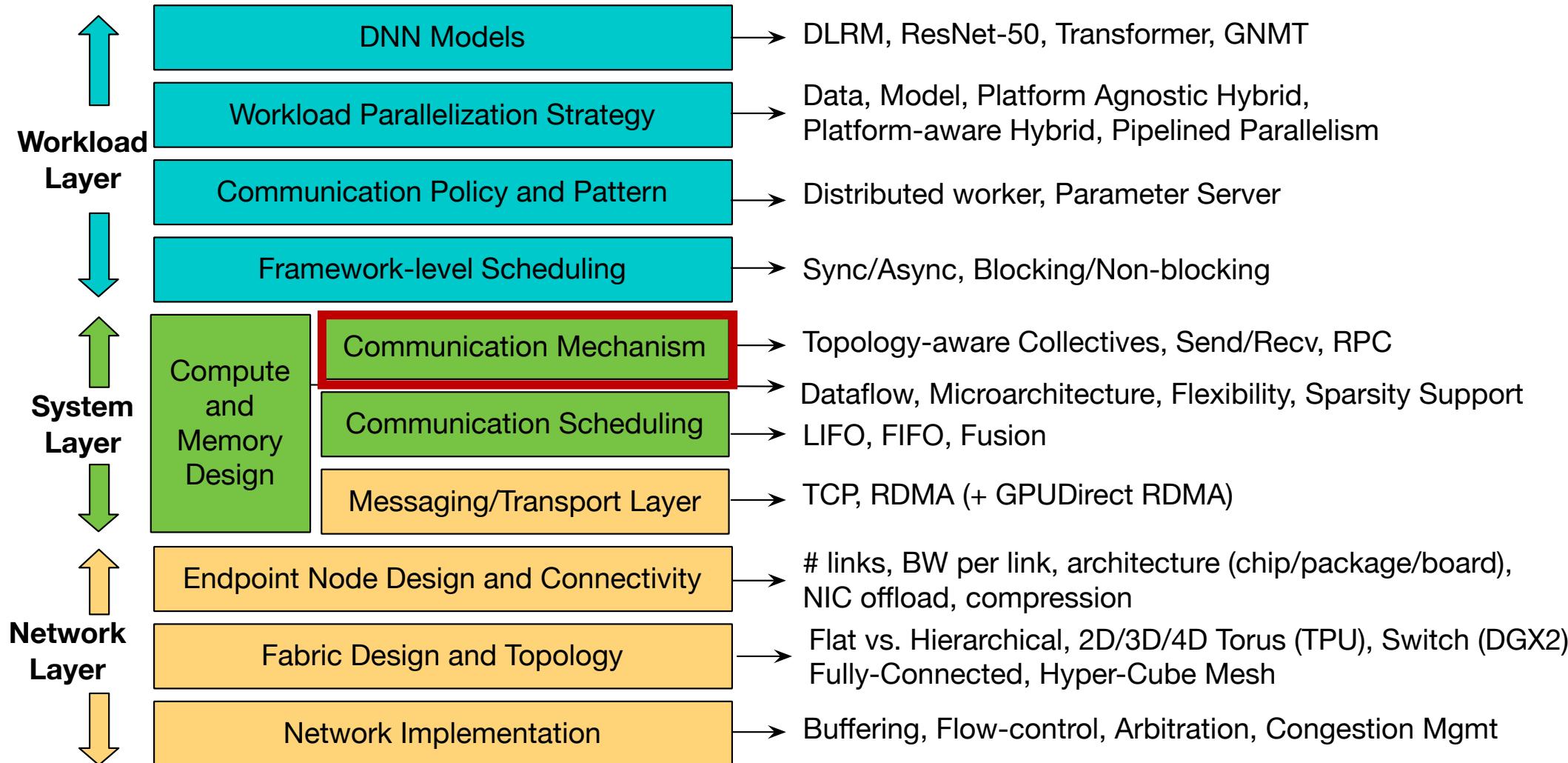
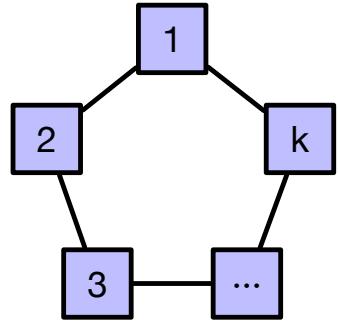
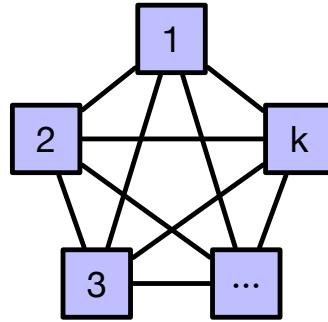
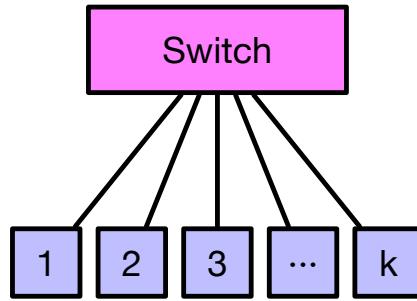
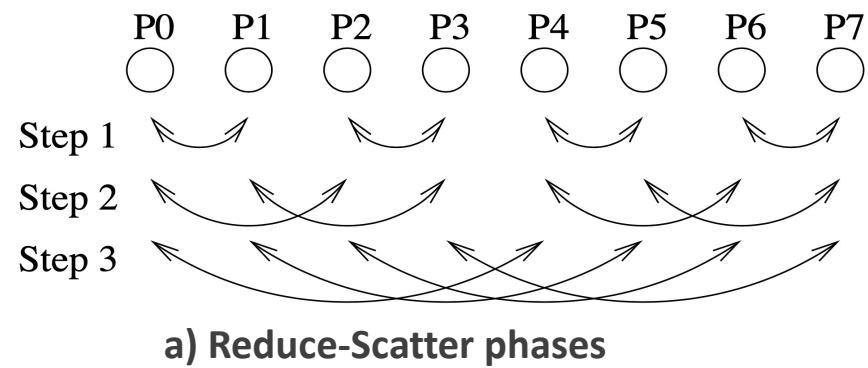


Figure Courtesy: Srinivas Sridharan (Facebook)

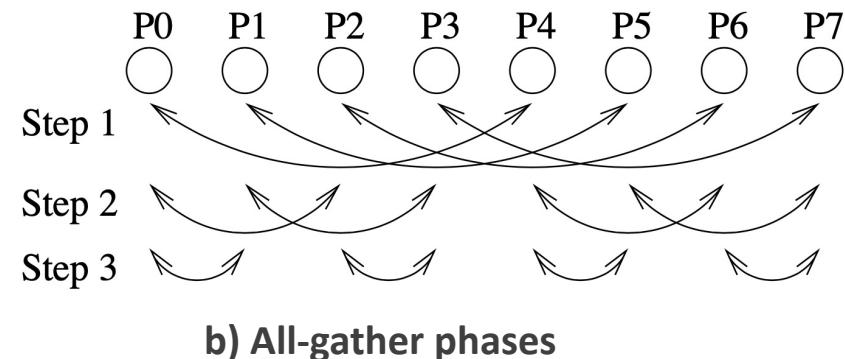
# Topology-aware Collectives

(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

HalvingDoubling All-Reduce

# Distributed Training Stack

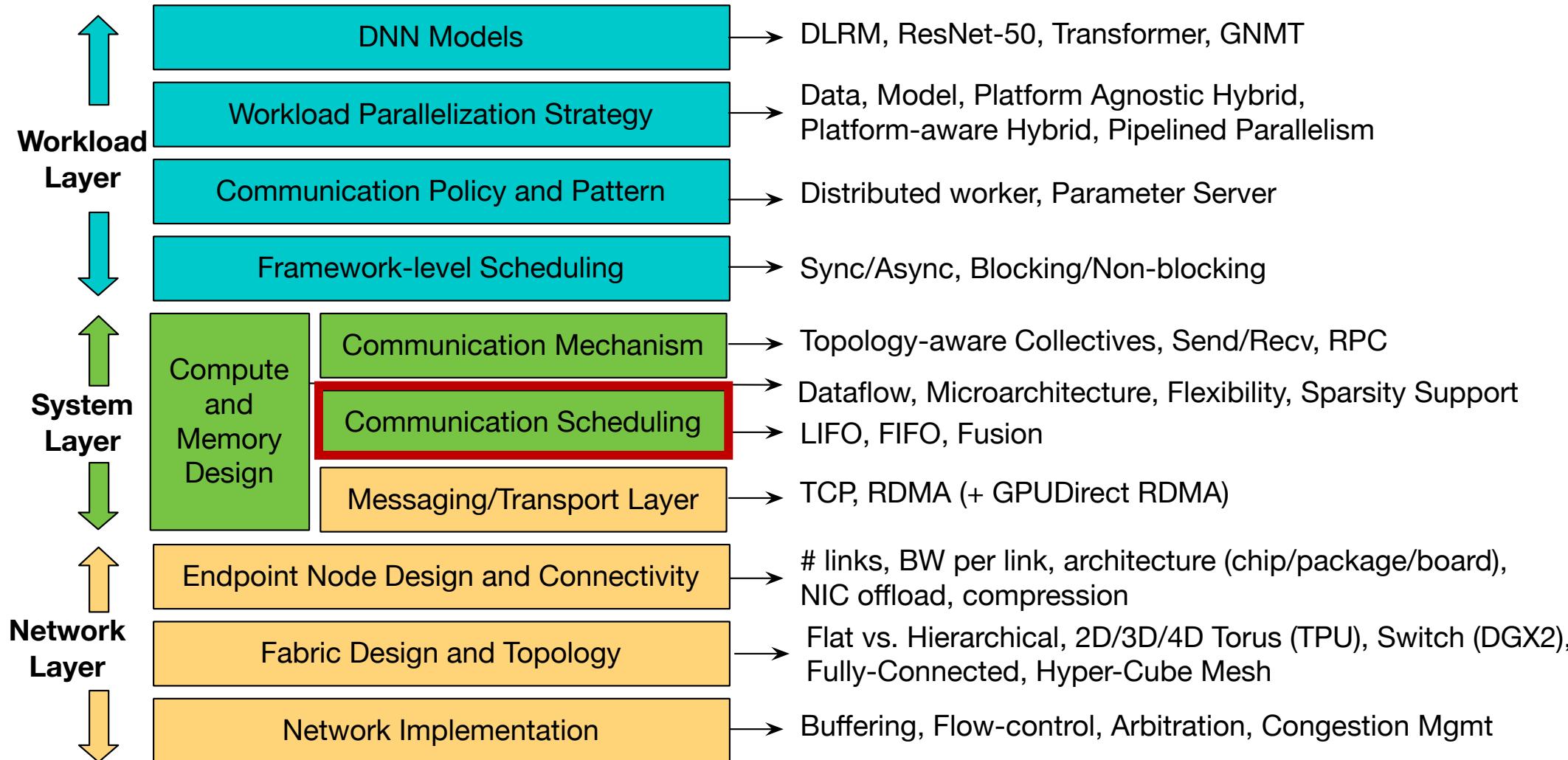
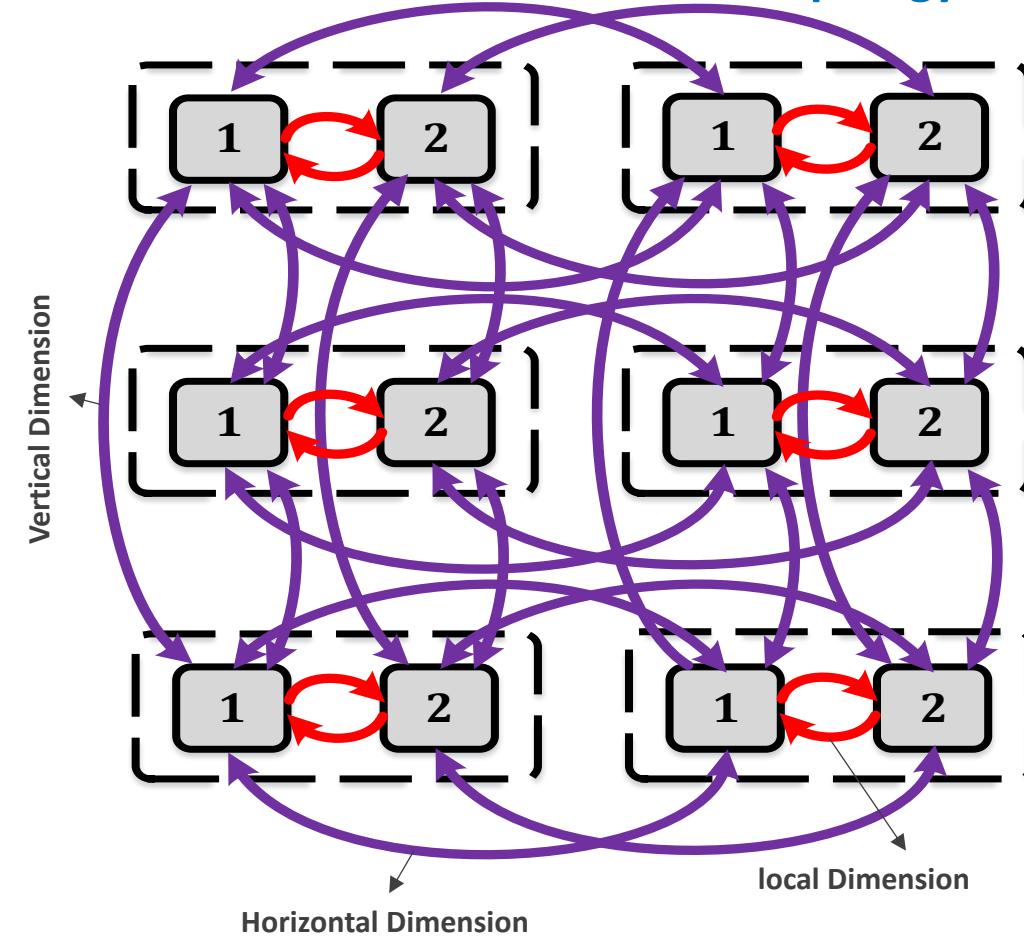


Figure Courtesy: Srinivas Sridharan (Facebook)

# Baseline All-Reduce on the Hierarchical Topologies

## 3D Torus – Hierarchical Topology

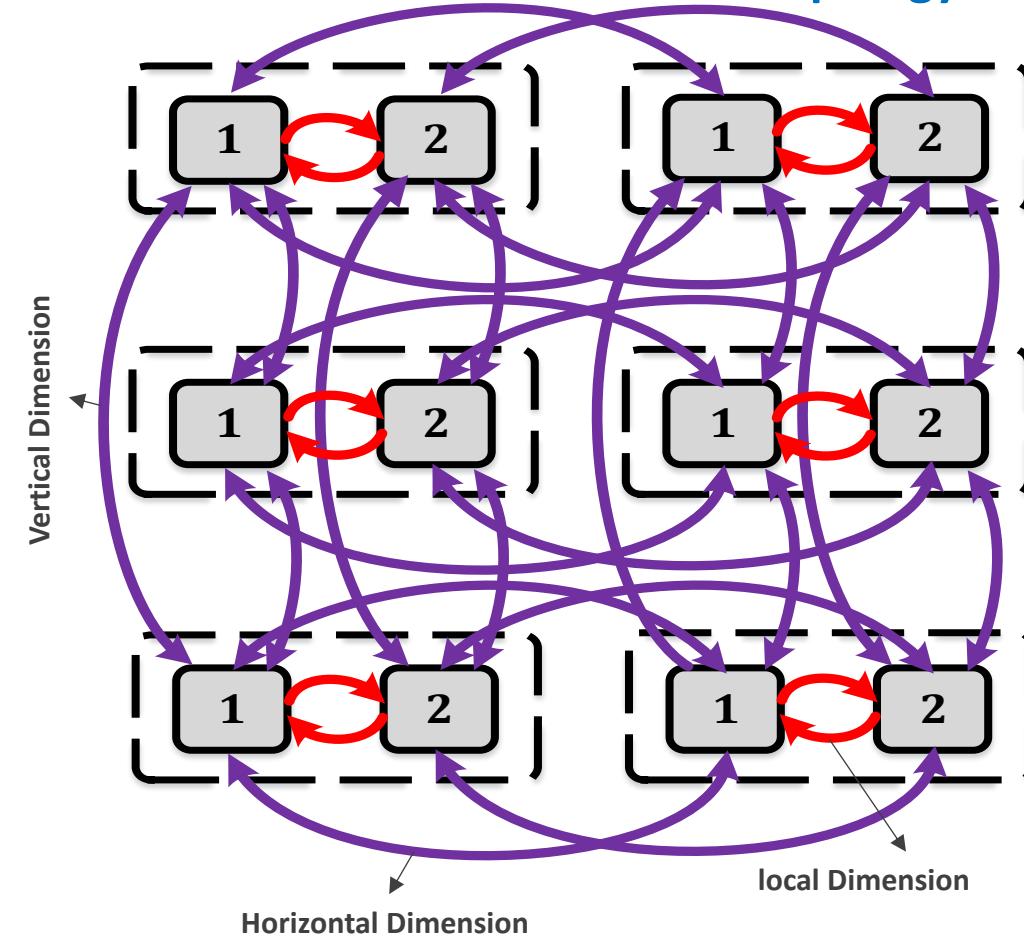


Data chunks	All-reduce pipeline on the hierarchical topologies:	Suppose the chunk size per NPU is initially 12MB
start	Step:	Chunk size in the initial of the phase:
6	1. Reduce-scatter on the <b>local</b> dimension	12MB
5	2. Reduce-scatter on the <b>horizontal</b> dimension	6MB
4	3. Reduce-scatter on the <b>vertical</b> dimension	3MB
3	4. All-gather on the <b>vertical</b> dimension	1MB
2	5. All-gather on the <b>horizontal</b> dimension	3MB
1	6. All-gather on the <b>local</b> dimension	6MB
		End: 12MB
end		

S. Rashidi et al., "Themis: A Network Bandwidth-Aware Collective Scheduling Policy for Distributed Training of DL Models". ISCA 2022.

# Baseline All-Reduce on the Hierarchical Topologies

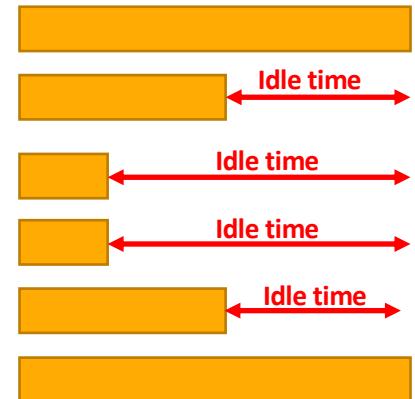
## 3D Torus – Hierarchical Topology



All-reduce pipeline on the hierarchical topologies:

- Step: 1. Reduce-scatter on the **local** dimension
2. Reduce-scatter on the **horizontal** dimension
3. Reduce-scatter on the **vertical** dimension
4. All-gather on the **vertical** dimension
5. All-gather on the **horizontal** dimension
6. All-gather on the **local** dimension

Pipeline Stage latency:



**Problem: Uneven pipeline stage latencies that causes network underutilization**

*For solution to this problem, check out our talk on Themis on Tuesday, June 21, Session 7B, 2:30 – 2:50 PM*

S. Rashidi et al., "Themis: A Network Bandwidth-Aware Collective Scheduling Policy for Distributed Training of DL Models". ISCA 2022.

# Distributed Training Stack

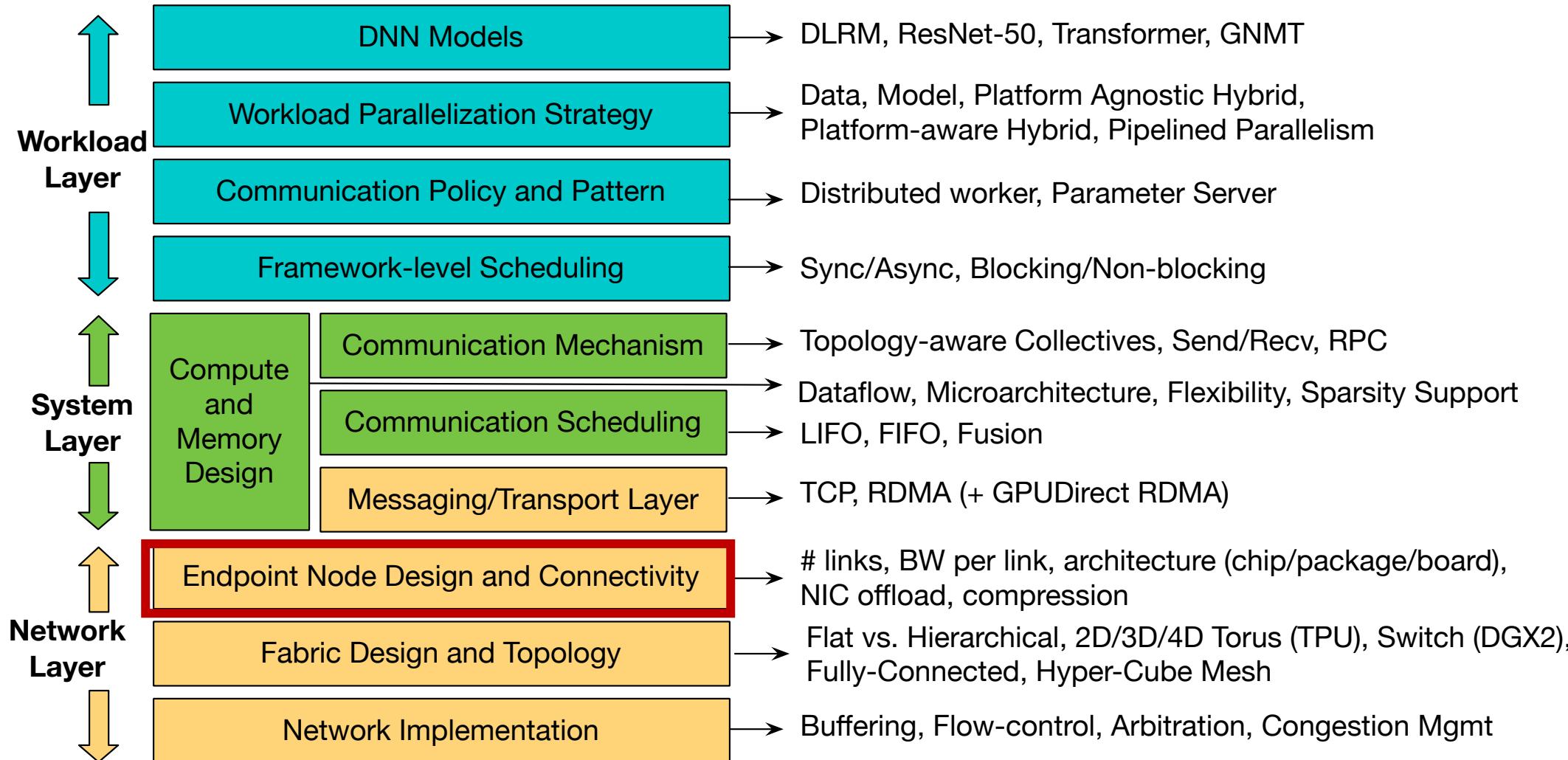
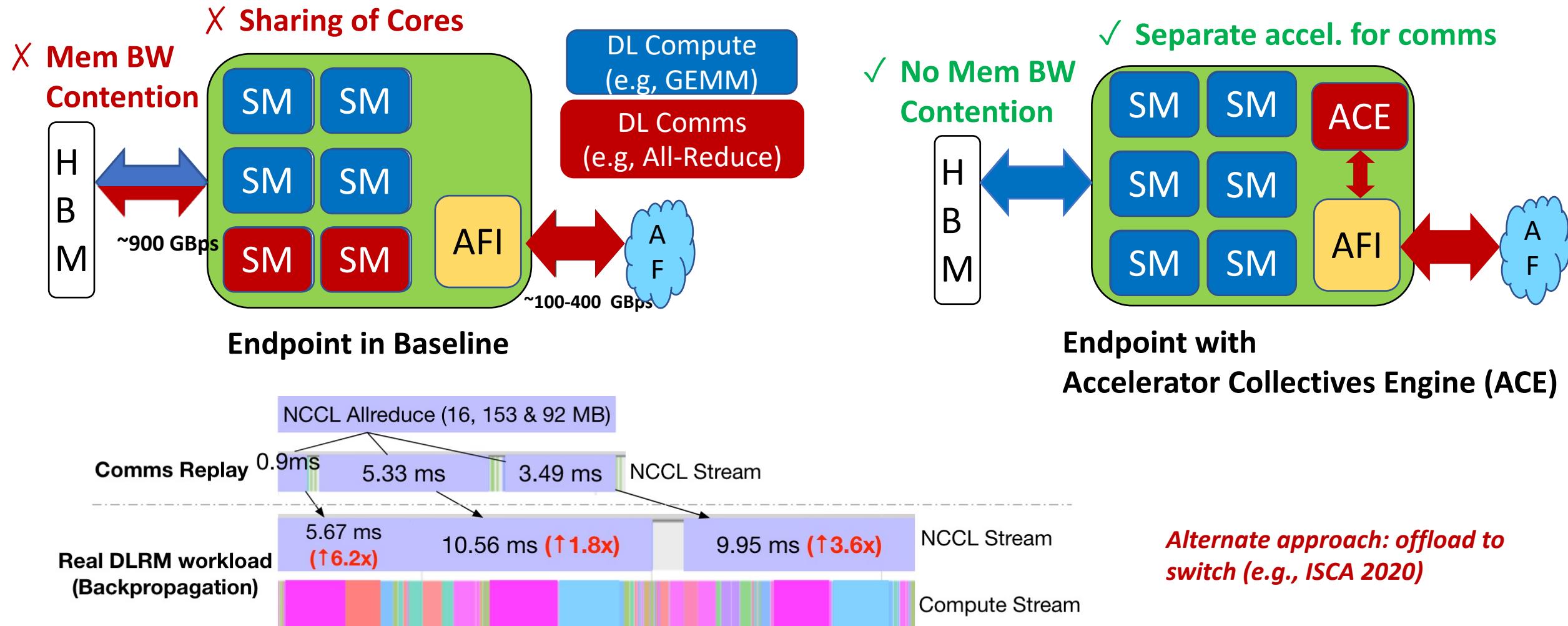


Figure Courtesy: Srinivas Sridharan (Facebook)

# Resource Contention at End-point



(b) Impact of compute-comms overlap on a real-world production-class DLRM workload

S. Rashidi et al., "Enabling Compute-Communication Overlap in Distributed Deep Learning Training Platforms". ISCA 2021

# Distributed Training Stack

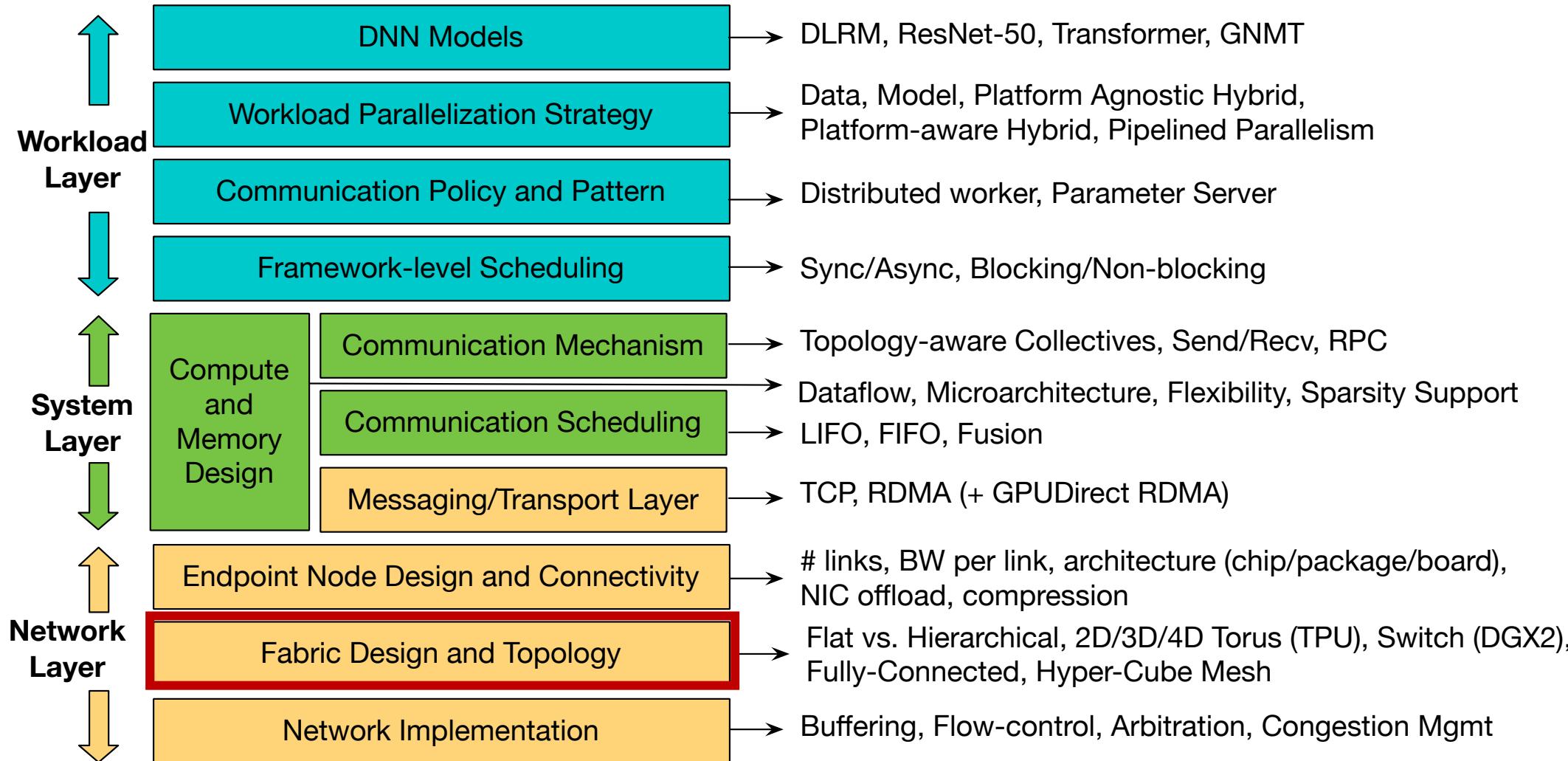
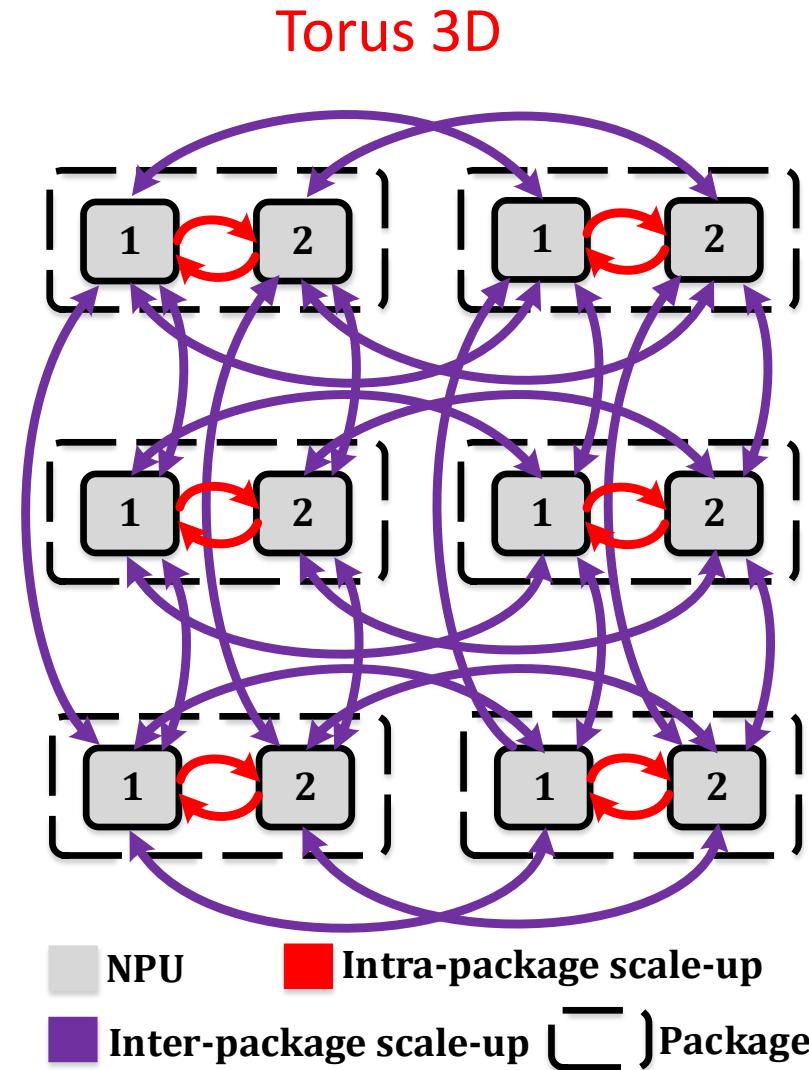


Figure Courtesy: Srinivas Sridharan (Facebook)

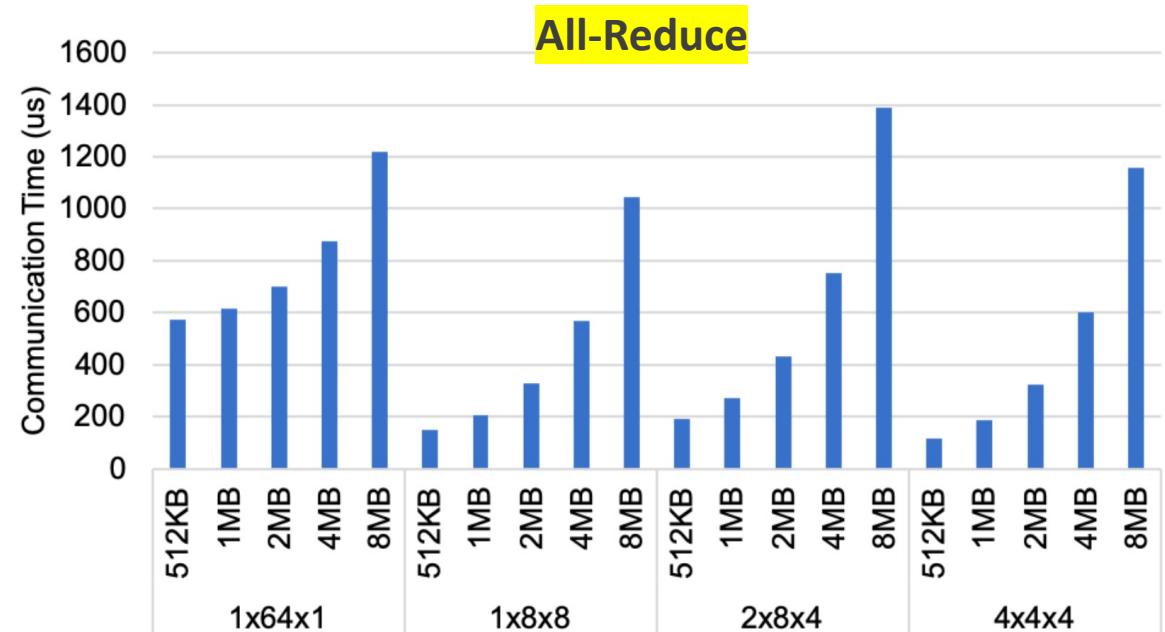
# Target System



X \* Y \* Z dimension  
X= cores within a package  
Y= packages in horizontal dimension  
Z= packages in vertical dimension

# Impact of 1D/2D/3D Torus

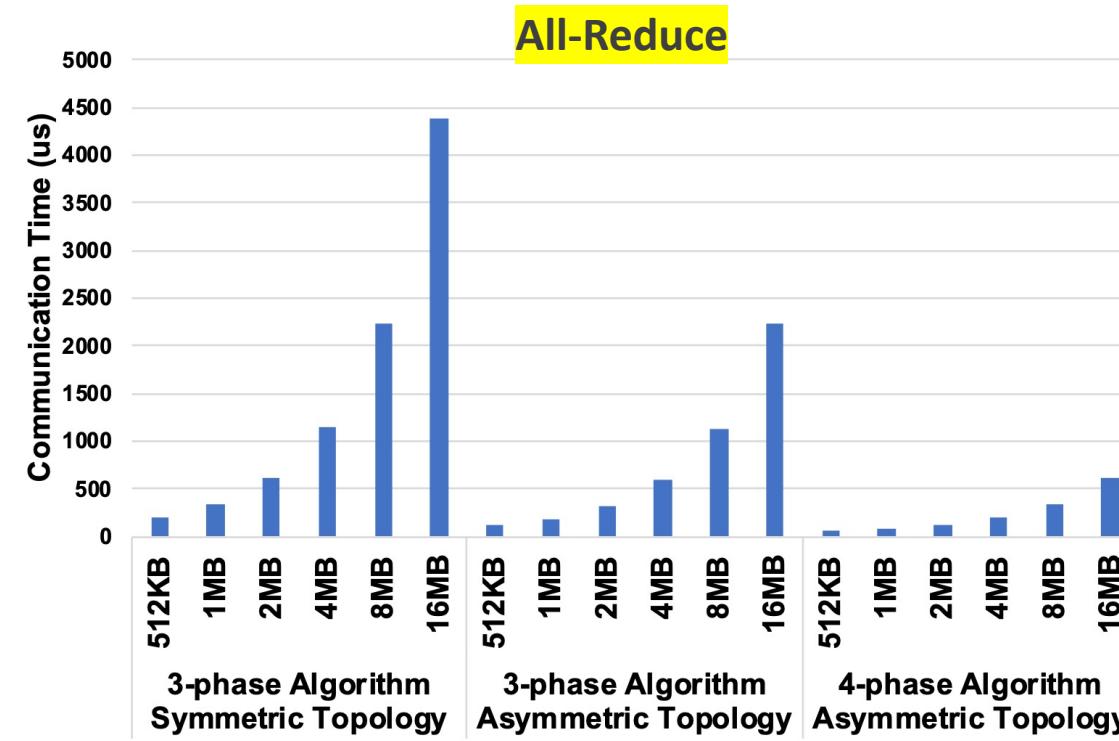
- Adding a dimension decreases the number of steps per collective.
  - For example, going from 1X64X1 to 1X8X8.
- Adding a dimension might increase amount of data each node sends out (depends on the algorithm).
  - For example, going from 1X8X8 to 2X8X4.
- Hence, choosing a topology is a tradeoff between the above 2 effects.



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

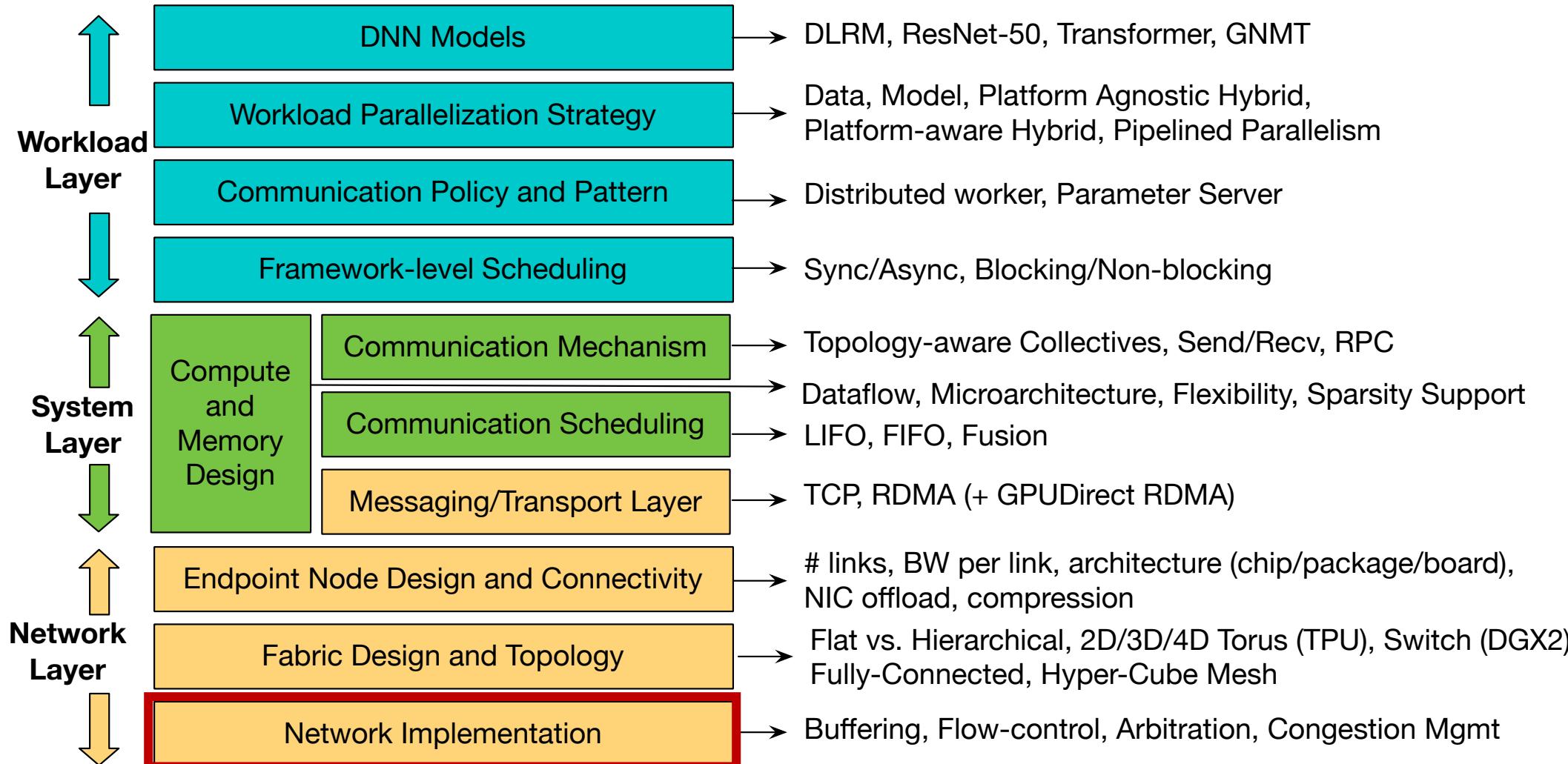
# Impact of Asymmetric Hierarchical Topology

- Having higher intra-package BW improves the performance.
- We can further improve performance by changing the algorithm to leverage this asymmetric BW.



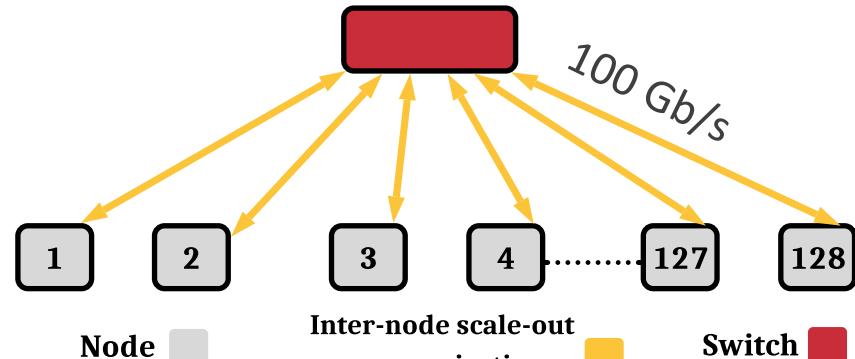
S. Rashidi et al., “**ASTRA-SIM: Enabling SW/HW Co-Design Exploration for Distributed DL Training Platforms**”, ISPASS 2020

# Distributed Training Stack

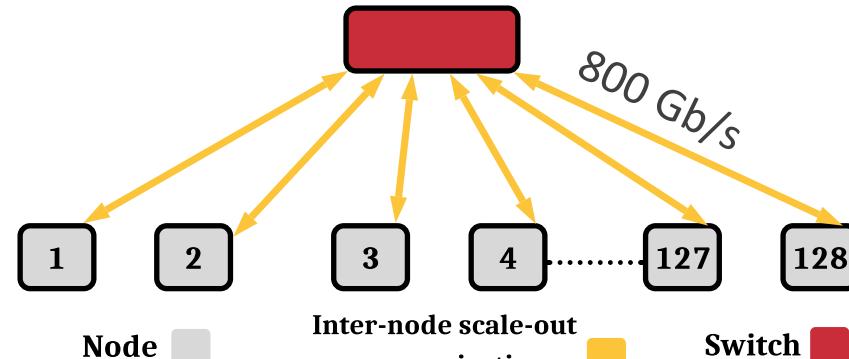


*Figure Courtesy: Srinivas Sridharan (Facebook)*

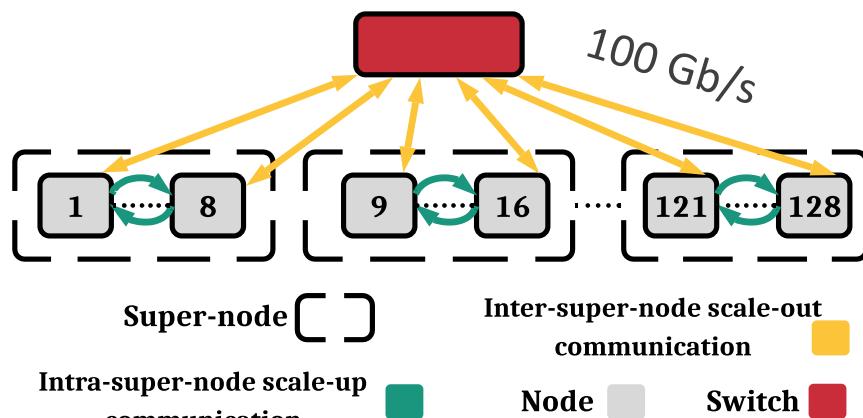
# Target Systems



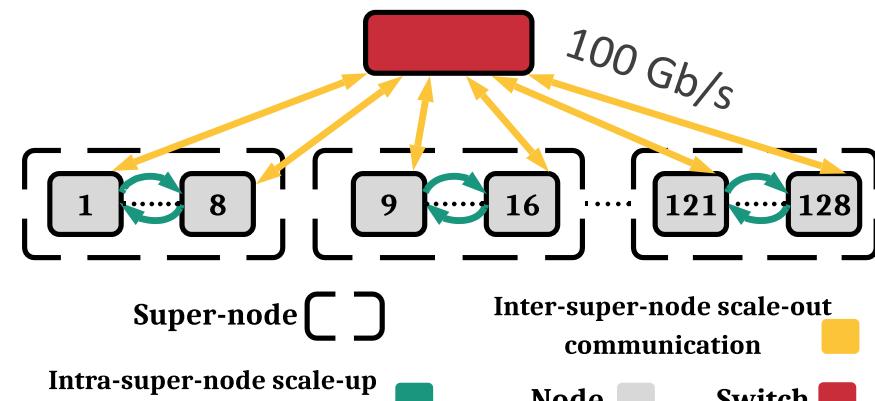
**Flat100G**  
1-phase all-reduce



**Flat800G**  
1-phase all-reduce



**Hier**  
2-phase all-reduce

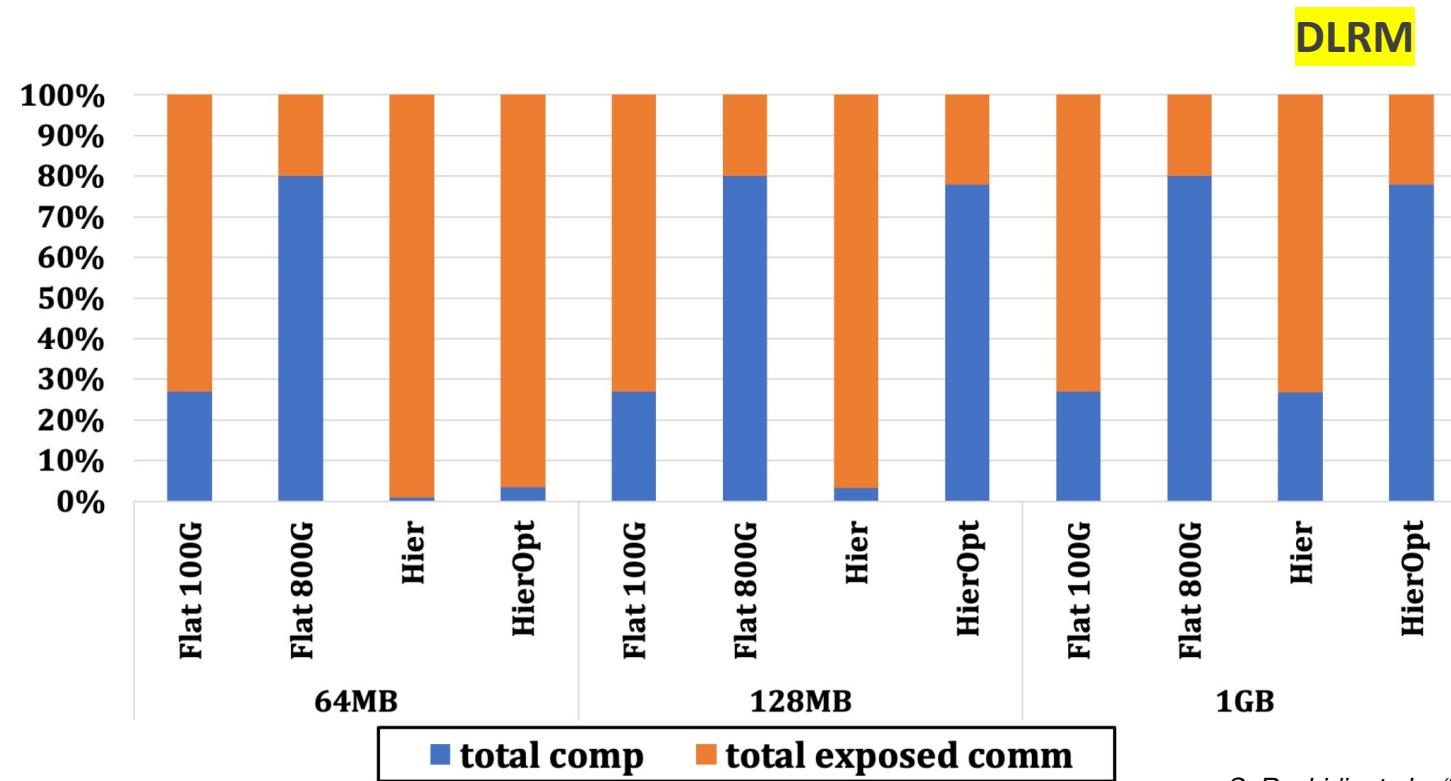


**HierOpt**  
3-phase all-reduce

# Effect of Size of Switch Buffer

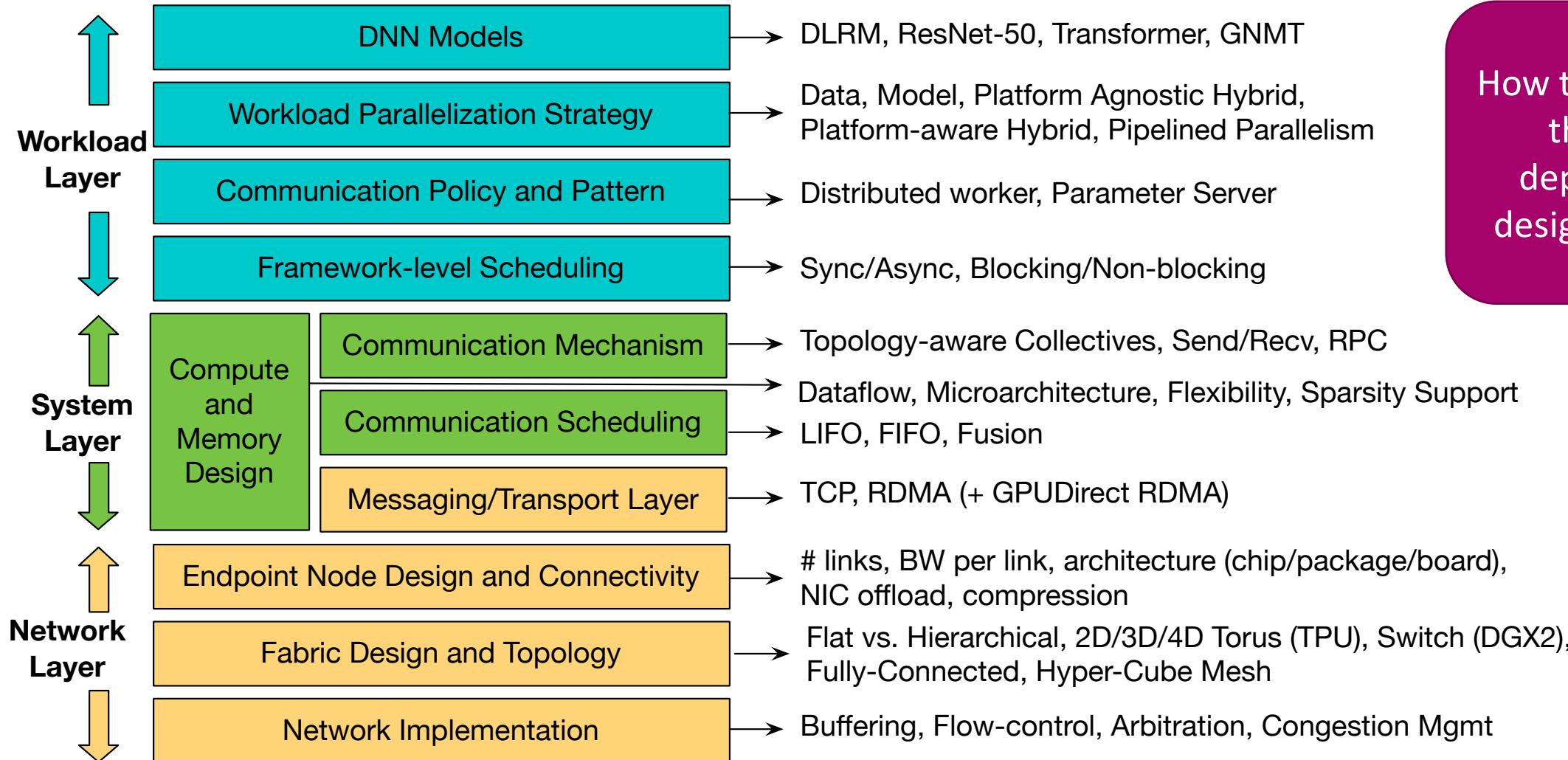
## Observations:

- Flat vs. Hierarch different Sensitivity to global switch size



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

# Distributed Training Stack



How to navigate  
this co-  
dependent  
design-space?

Figure Courtesy: Srinivas Sridharan (Facebook)

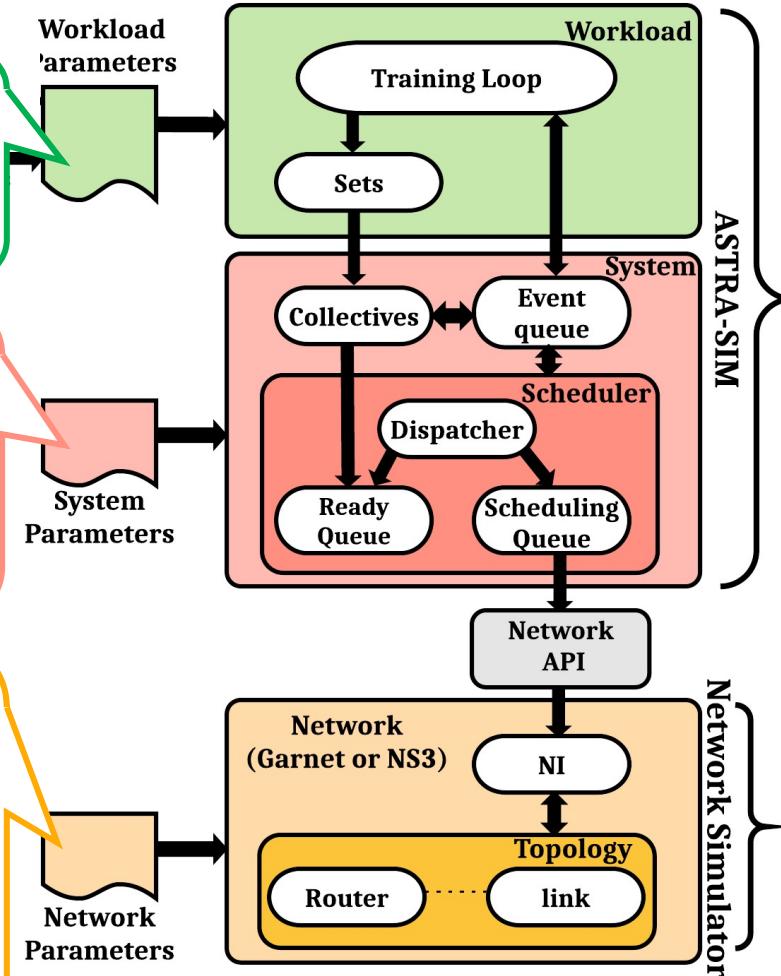
# Introducing ASTRA-sim

✓ Released ➤ In progress

- ✓ Supports Data-Parallel, Model-Parallel, Hybrid-Parallel training loops
- ✓ Extensible to more training loops
  - Graph-based input from PyTorch

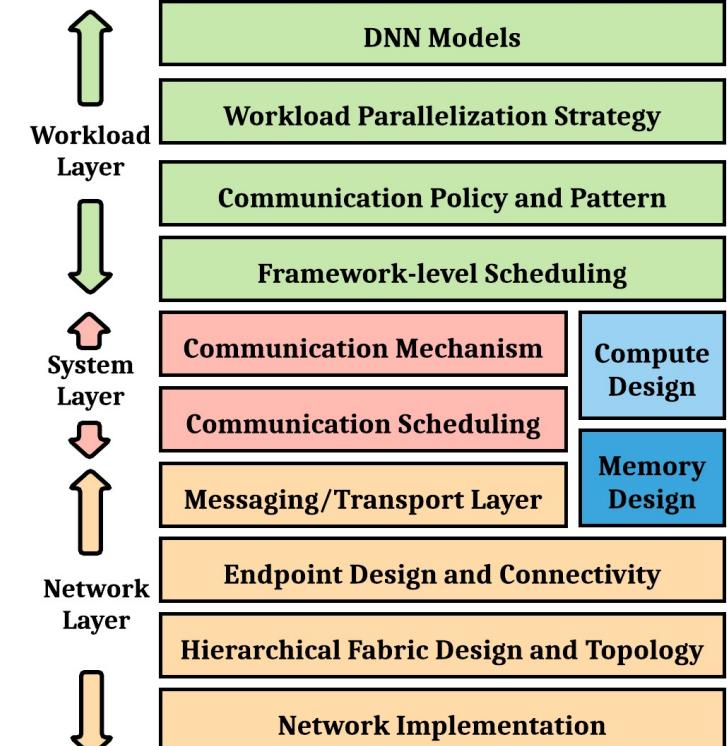
- ✓ Ring based, Tree-based, AlltoAll based, and multi-phase collectives
- ✓ Variety of scheduling policies
- ✓ Compute times fed via offline system measurements or compute simulator

- ✓ Various topologies, flow-control, link bandwidth, congestion control
- ✓ Plug-and-play options
  - ✓ Analytical (roofline)
  - Analytical with congestion
  - ✓ Garnet (credit-based)
  - NS3 (TCP, RDMA)



<http://github.com/astra-sim/astra-sim>

## DL Training Co-Design Stack



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

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

# What Does ASTRA-sim Report?

## ASTRA-sim Reports:

1. End-to-end training time.
2. Total communication time for each communication operation.
3. The amount of **exposed communication** for each communication operation.
4. Total Exposed communication and total computation.
5. More detailed stats such as average message latency per each hierarchical collective phase.

## Network Backend Specific Reports (Depends on the network backend type):

1. Network BW utilization
2. Communication protocol stats, such as packet drops, # of retransmissions, etc.
3. Network switch buffer usage
4. ...

# Summary and Takeaways

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

# Agenda

---

Time (EDT)	Topic	Presenter
8:30 – 9:30	<b>Introduction to Distributed Deep Learning Training Platforms</b>	Tushar Krishna
9:30 – 10:30	<b>ASTRA-sim</b>	Saeed Rashidi
10:30 – 11:00	<b>Coffee Break</b>	
11:00 – 11:50	<b>Demo and Exercises</b>	William Won and Taekyung Heo
11:50 – 12:00	<b>Extensions and Future Development</b>	Tushar Krishna and Saeed Rashidi

## Tutorial Website

*includes agenda, slides, ASTRA-sim installation instructions (via source + docker image)*

<https://astra-sim.github.io/tutorials/isca-2022>

**Attention:** Tutorial is being recorded