Georgia Tech School of Electrical and Computer Engineering
College of Engineering

http://synergy.ece.gatech.edu

MLSys Tutorial

August 31st, 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

Welcome

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

Saeed Rashidi
PhD Student, School of ECE
Georgia Institute of Technology
saeed.rashidi@ece.gatech.edu

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

Presenters

Taekyung Heo
Postdoctoral Fellow, School of ECE
Georgia Institute of Technology
taekyung@gatech.edu

Srinivas Sridharan
Research Scientist
Meta
ssrinivas@fb.com

Collaborators and Contributors

Sudarshan Srinivasan
Research Scientist, Intel

Tarannum Khan
Graduate Student, CS
UT Austin

Aditya Akella
Professor, CS
UT Austin

+ growing!

Agenda

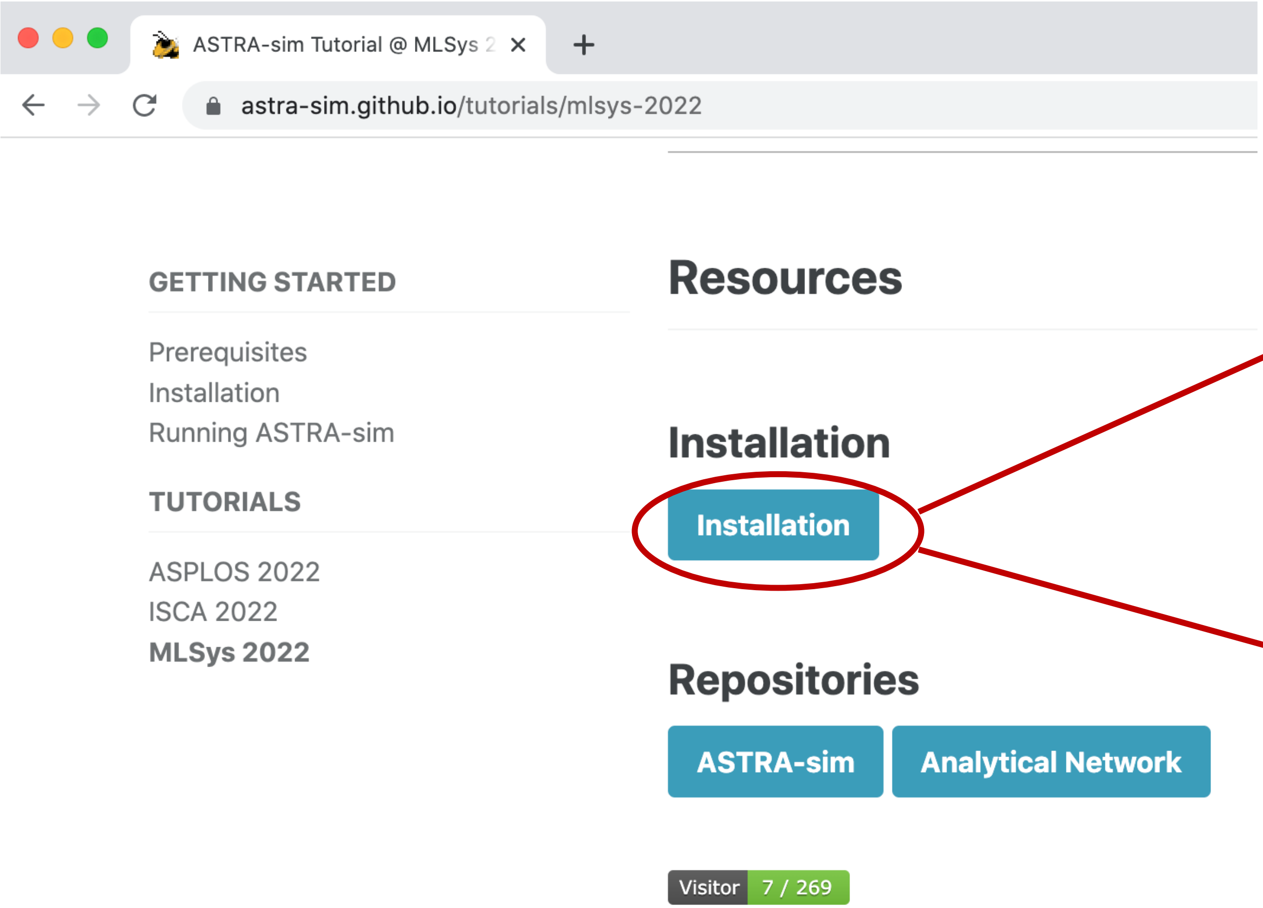
Time (PDT)	Topic	Presenter
1:00 – 2:00	**Introduction to Distributed DL Training**	Tushar Krishna
2:00 – 2:20	**Challenges on Distributed Training Systems**	Srinivas Sridharan
2:20 – 3:30	**Introduction to ASTRA-sim simulator**	Saeed Rashidi
3:30 – 4:00	**Coffee Break**	
4:00 – 4:50	**Hands-on Exercises on Using ASTRA-sim**	William Won and Taekyung Heo
4:50 – 5:00	**Closing Remarks and Future Developments**	Taekyung Heo

Tutorial Website
includes agenda, slides, ASTRA-sim installation instructions (via source + docker image)
https://astra-sim.github.io/tutorials/mlsys-2022

Attention: Tutorial is being recorded

ASTRA-sim Installation

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



Getting Started with Source Codes

Getting Started with Prebuilt Docker Image

Agenda

Time (PDT)	Topic	Presenter
1:00 – 2:00	**Introduction to Distributed DL Training**	Tushar Krishna
2:00 – 2:20	**Challenges on Distributed Training Systems**	Srinivas Sridharan
2:20 – 3:30	**Introduction to ASTRA-sim simulator**	Saeed Rashidi
3:30 – 4:00	**Coffee Break**	
4:00 – 4:50	**Hands-on Exercises on Using ASTRA-sim**	William Won and Taekyung Heo
4:50 – 5:00	**Closing Remarks and Future Developments**	Taekyung Heo

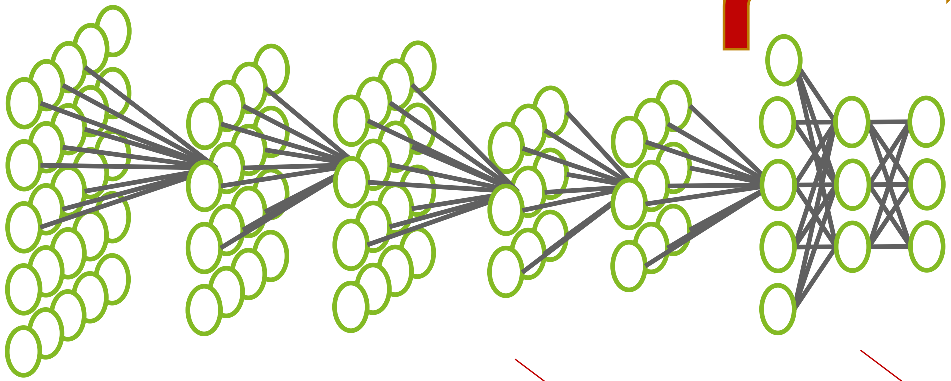
Tutorial Website

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

https://astra-sim.github.io/tutorials/mlsys-2022

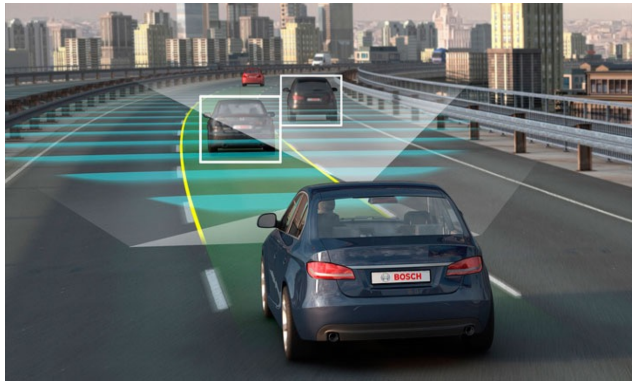
Attention: Tutorial is being recorded

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
  - **Purpose / Task**
    - Anomaly Detection
    - Classification
    - Clustering
    - Dimensionality Reduction
    - Representation Learning
    - Regression
  - **Feedback from data**
    - Supervised learning
    - Unsupervised learning
    - Semi-supervised learning
    - Reinforcement learning
  - **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

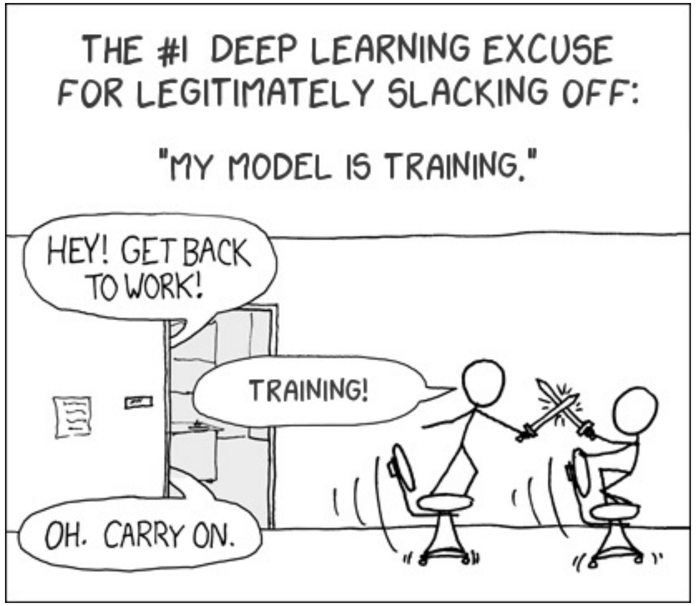
DL Training: The Phases

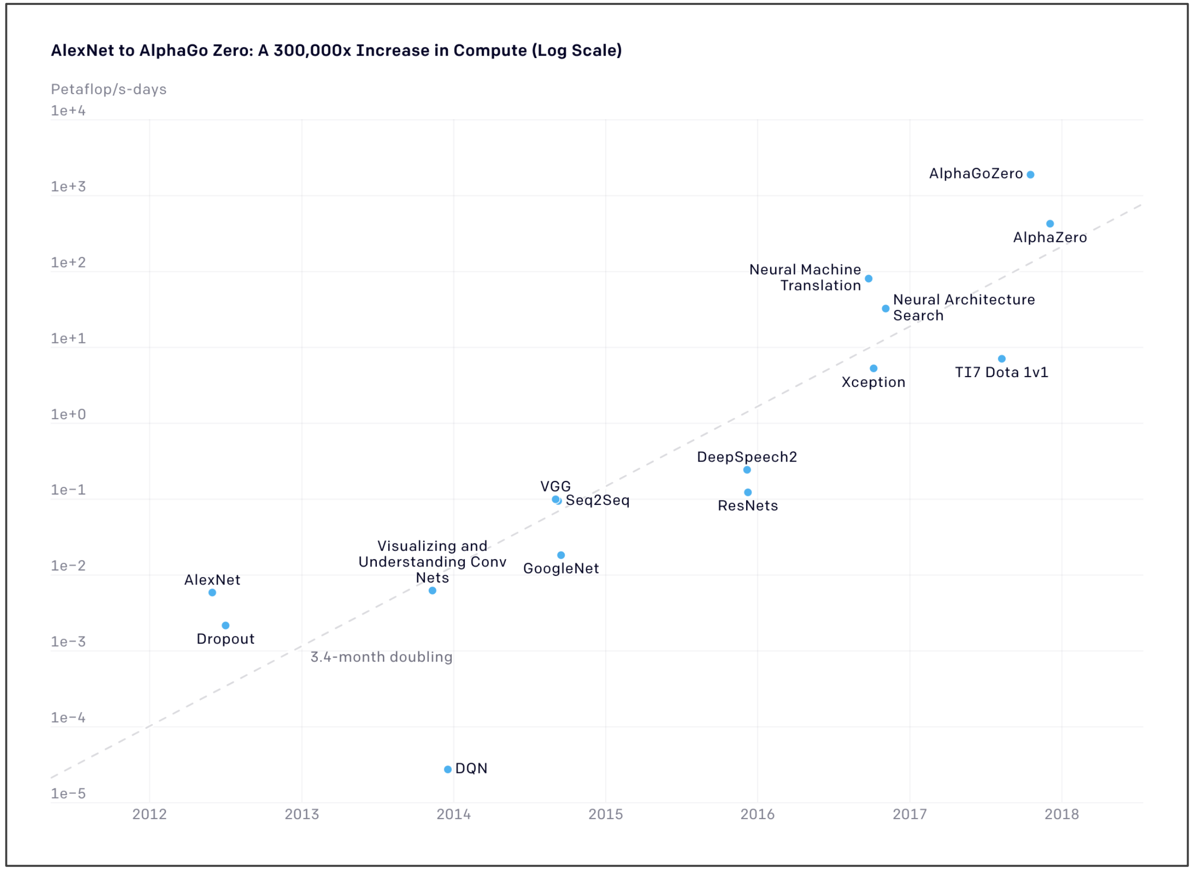
- Each training algorithm consists of 3 computation phases:
  - 1. Forward pass (inference):
    - The process of finding output activations using inputs and weights.
  - 2. Weight gradient computation:
    - The process of finding the gradient of weights (with respect to the loss function) using output 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

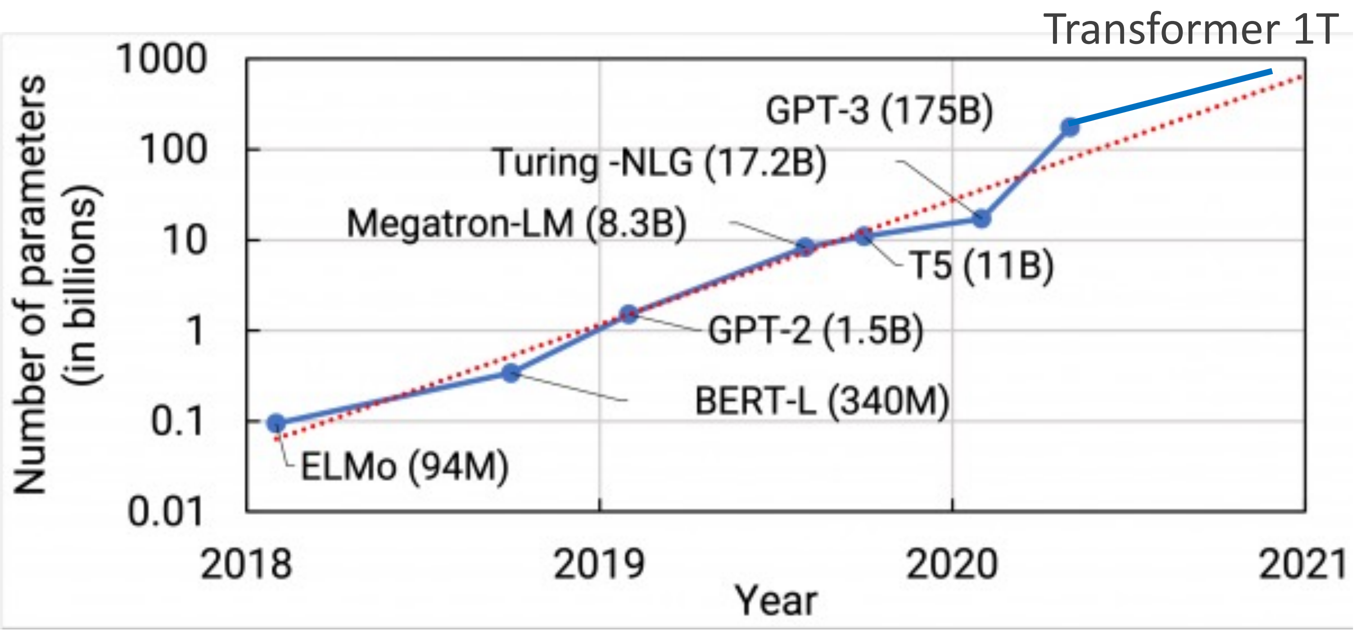
- **Training time is increasing**
  - DNNs are becoming larger
    - Turing NLG: 17.2 B Parameters
    - Megatron LM: 8.3B Parameters
  - Training samples are becoming larger
  - Moore's Law has ended

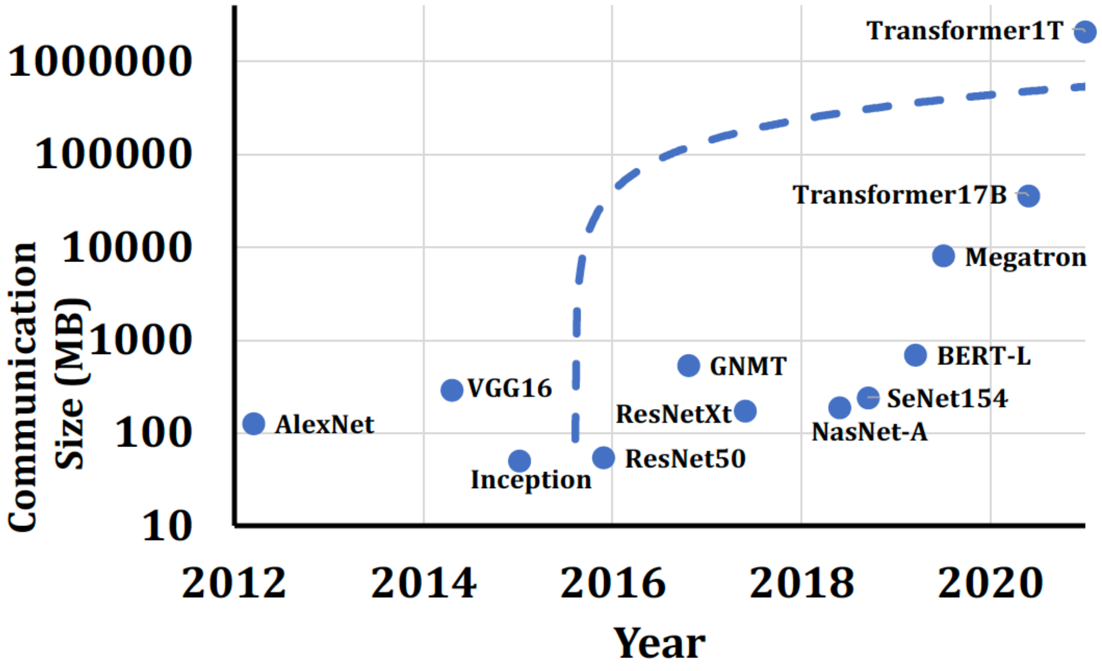




Source: https://openai.com/blog/ai-and-compute/

Key Challenge: Large Models → Large Comms



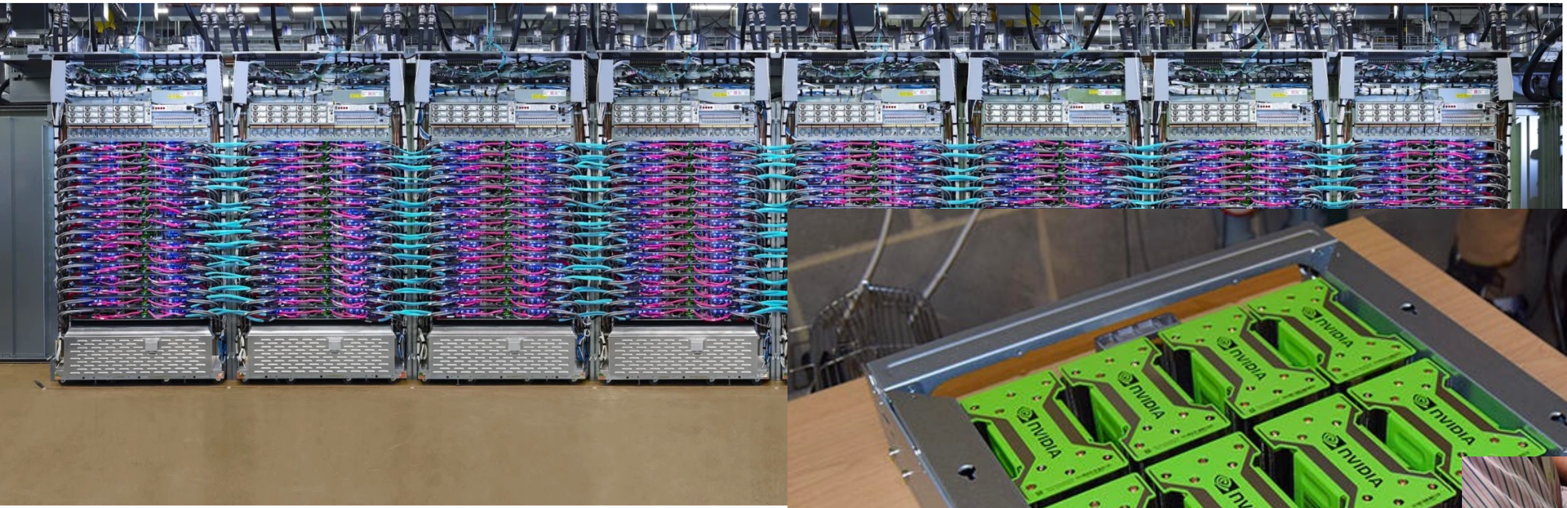


Challenges:

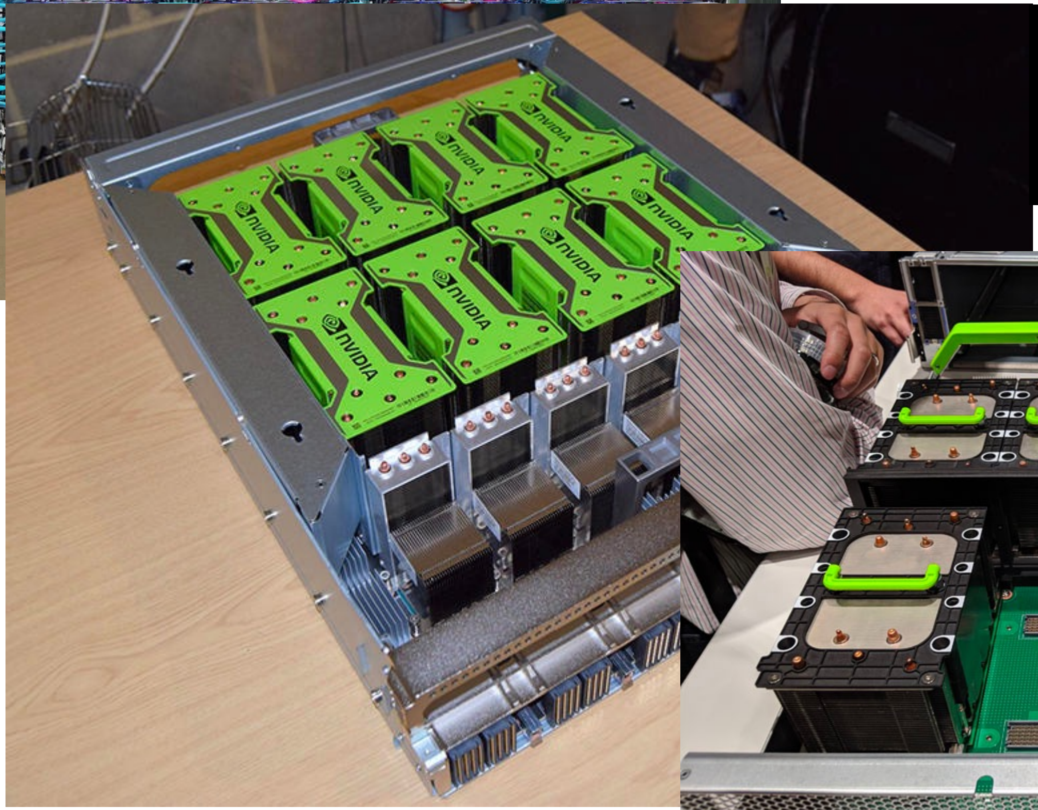
- Multiple NPUs are required to fit large-scale models
- e.g., 16 NPUs for GPT-3 (175B params)

  128 NPUs for Transformer-1T (1T params) (using ZeRO stage 2)

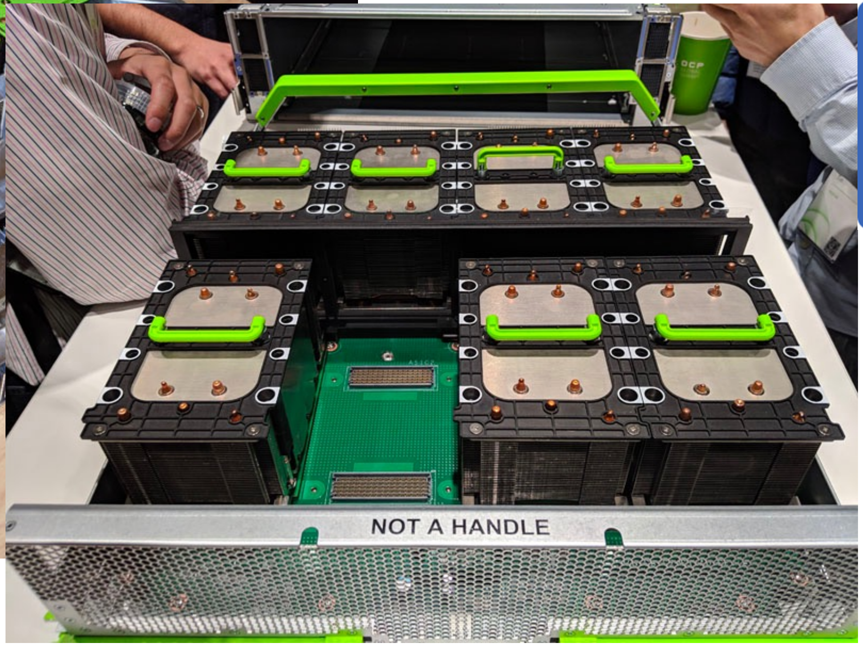
Enter: DL Training Platforms



TPU v3



DGX 2

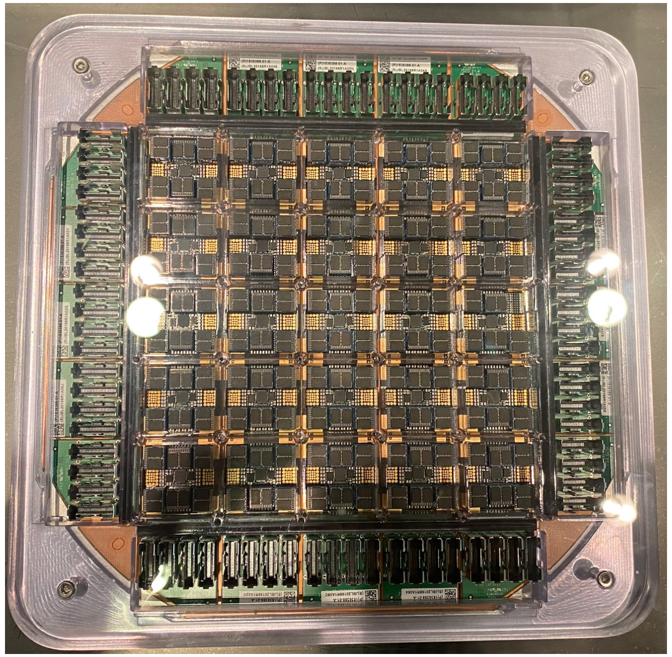


Zion

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

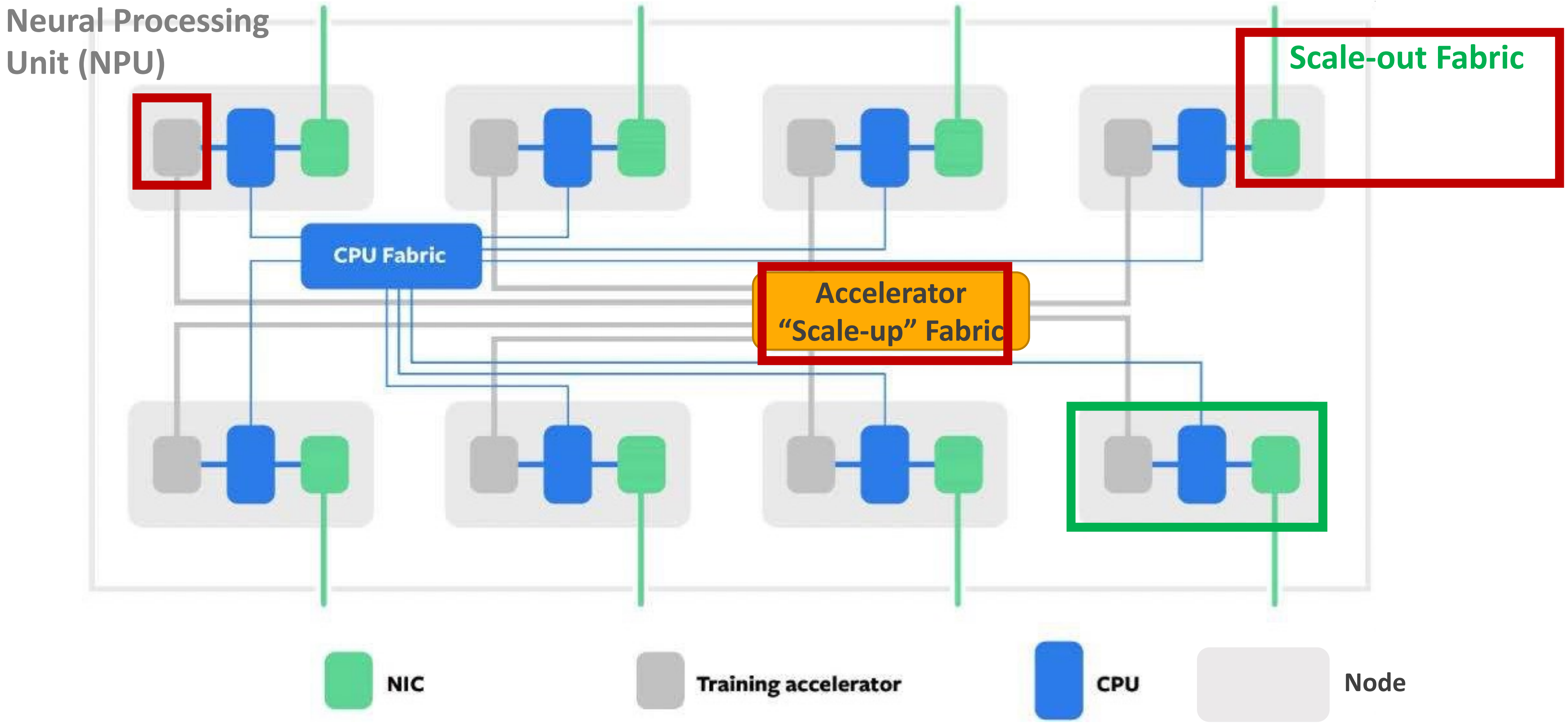
And many more …

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



Tesla Dojo

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"***

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

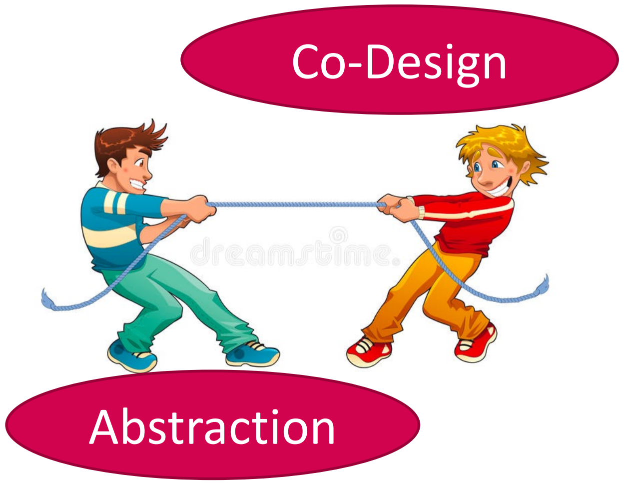


Figure Courtesy: Srinivas Sridharan (Facebook)

Distributed Training Stack

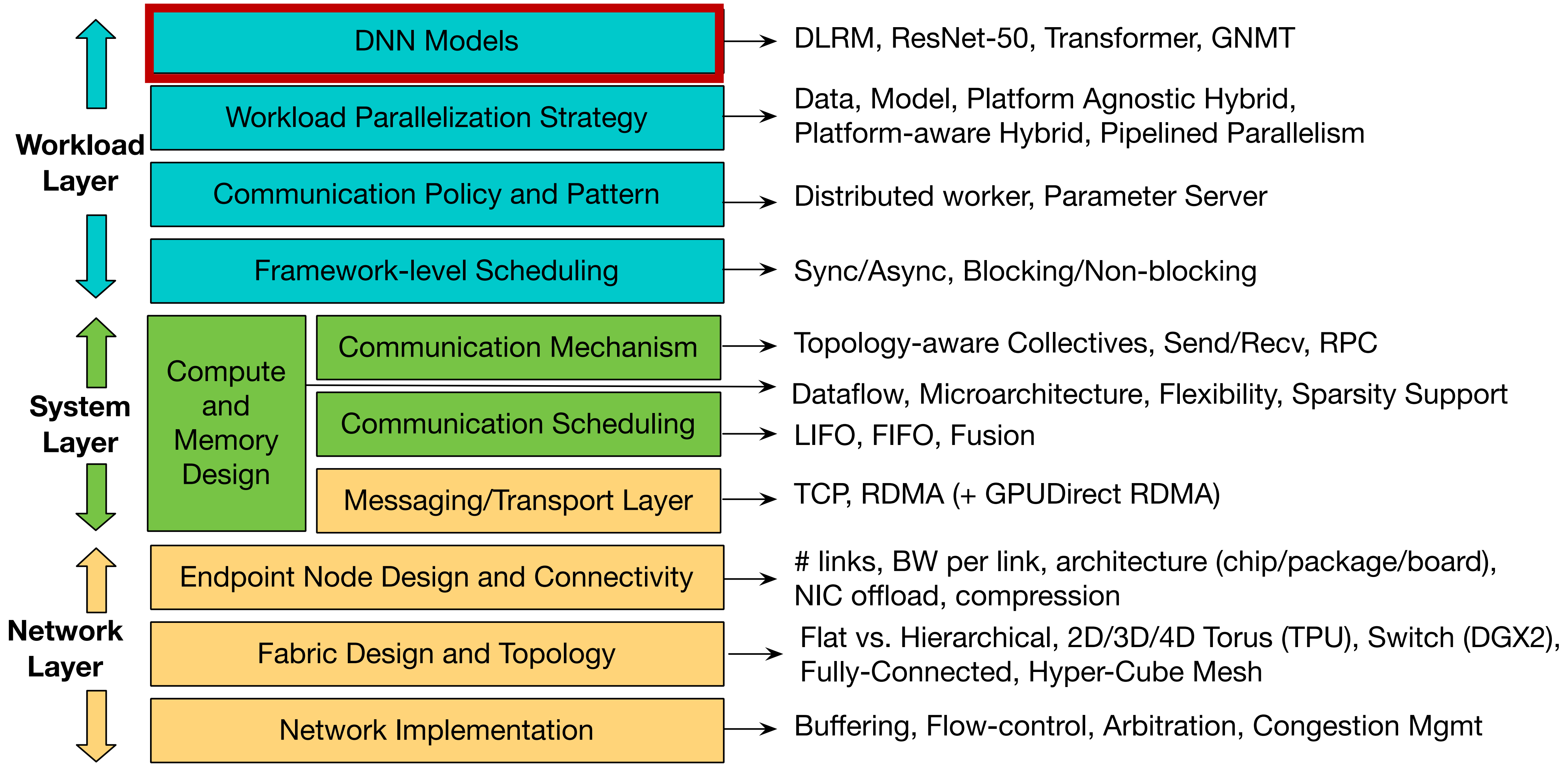
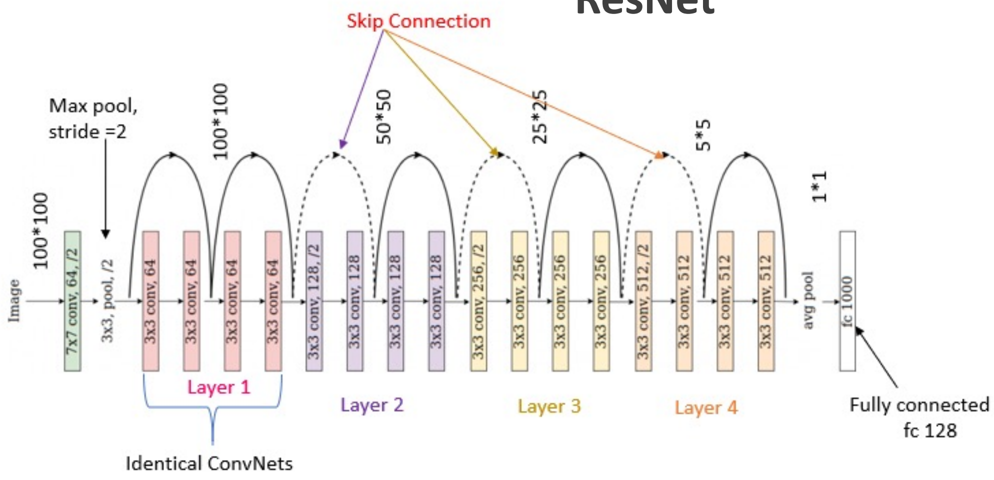


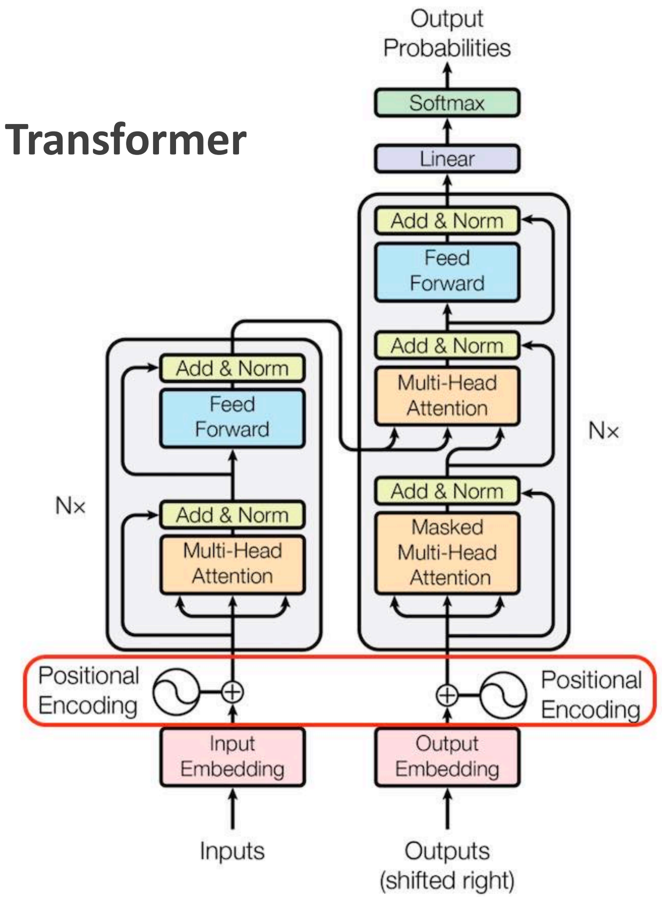
Figure Courtesy: Srinivas Sridharan (Facebook)

DNN Models

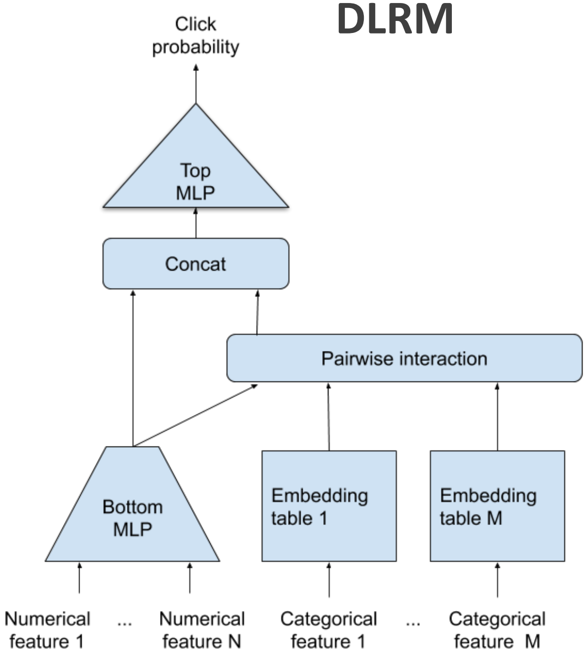
ResNet



Transformer



DLRM



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

Distributed Training Stack

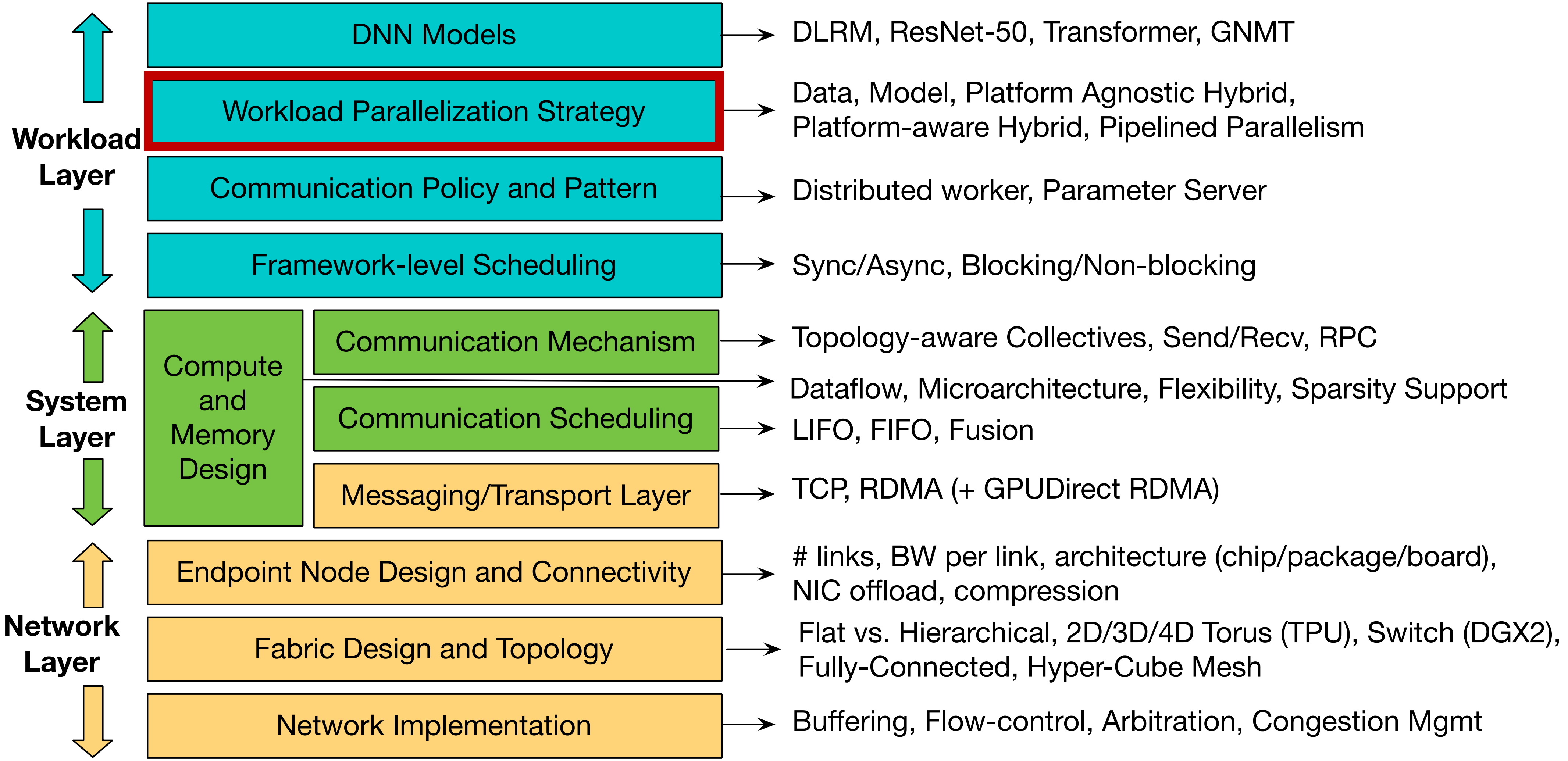


Figure Courtesy: Srinivas Sridharan (Facebook)

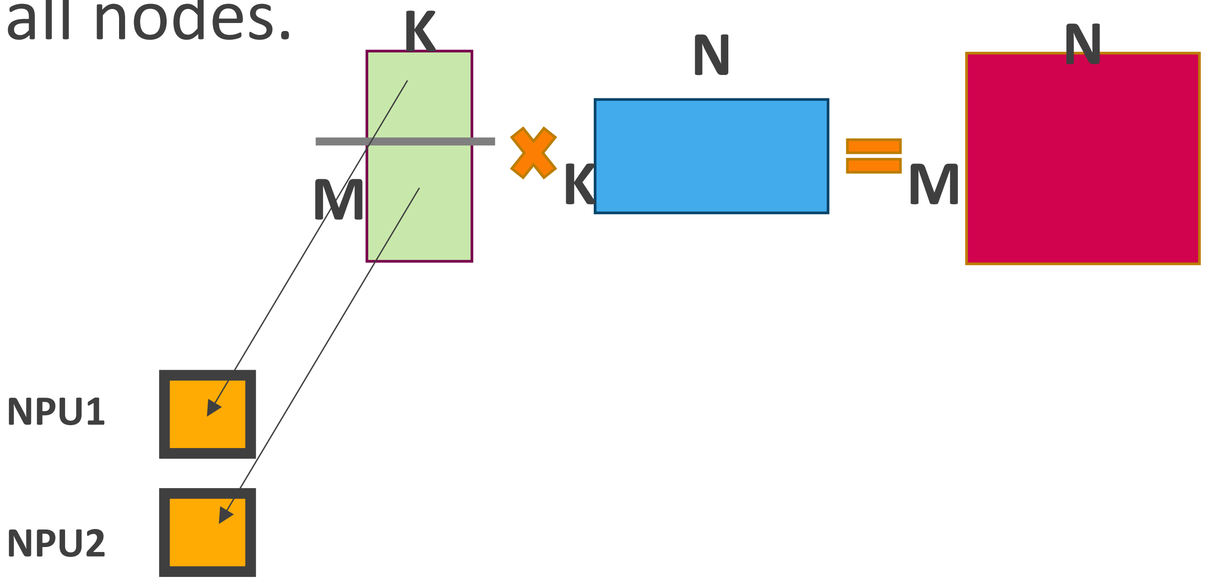
Parallelization Strategies

- 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.

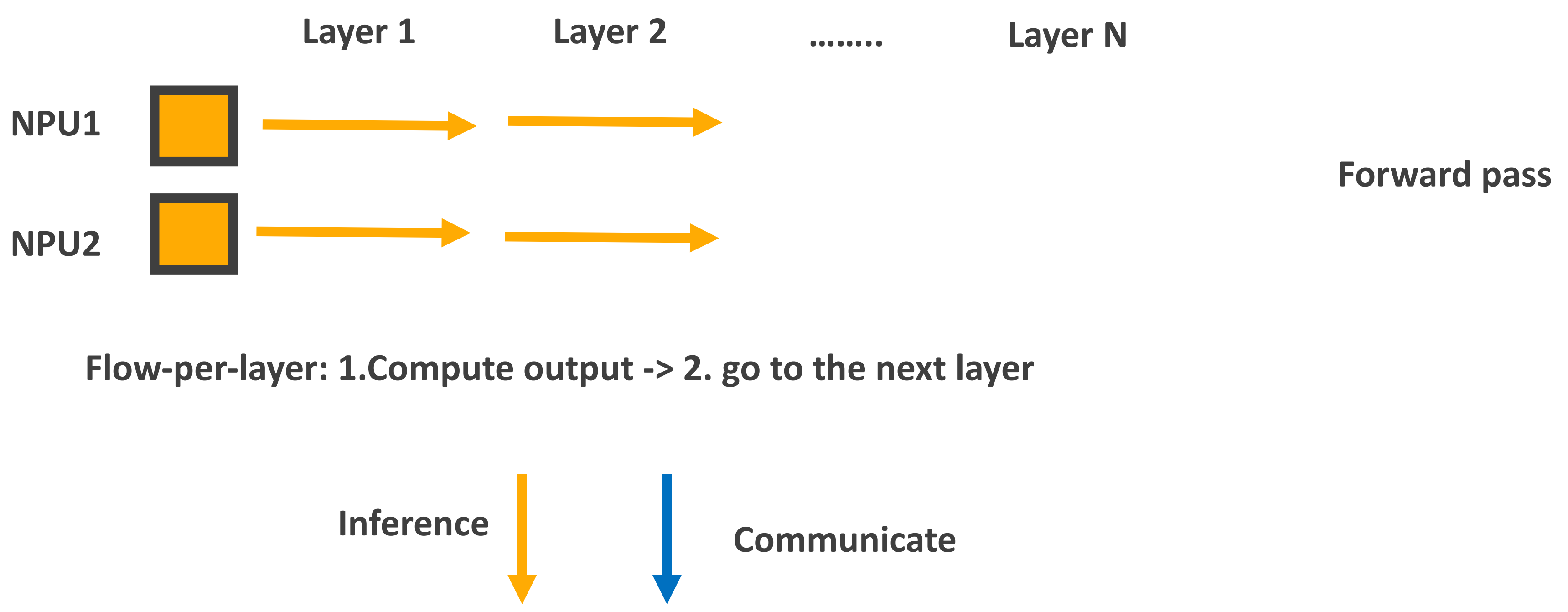
Parallelism: Data-Parallel Training

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



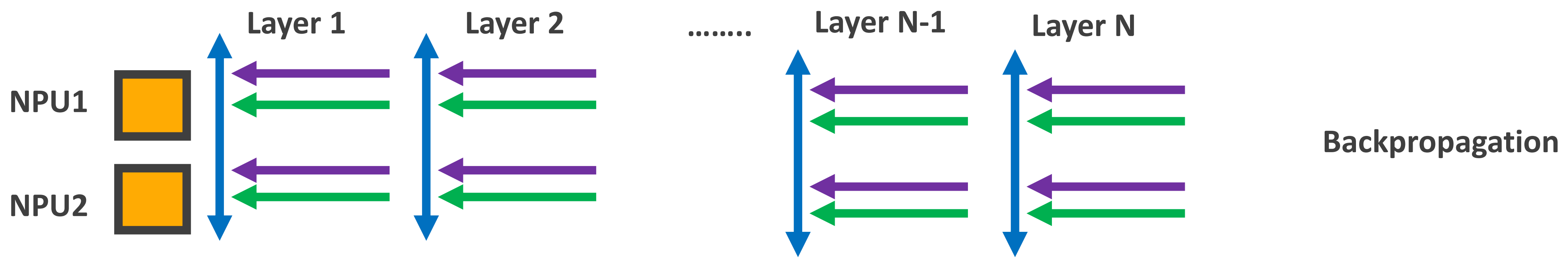
Parallelism: Data-Parallel Training

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



Parallelism: Data-Parallel Training

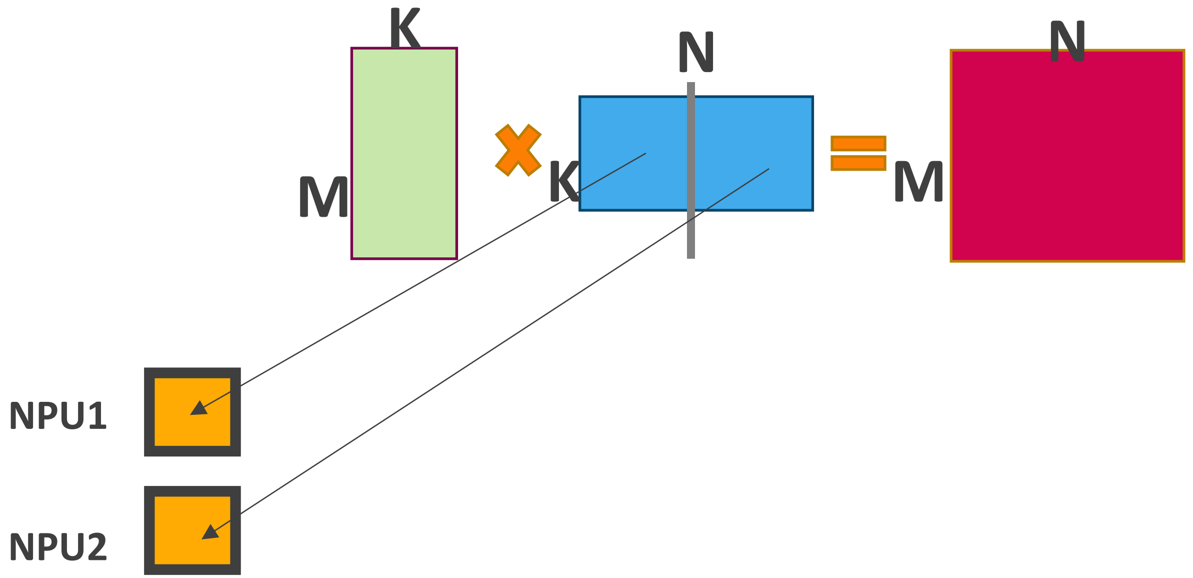
- Distribute Data across multiple nodes and replicate model (network) along all nodes.
- **Communicate weight gradients** during the backpropagation pass.
  - Blocking wait at end of backpropogation for collective before forward pass



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

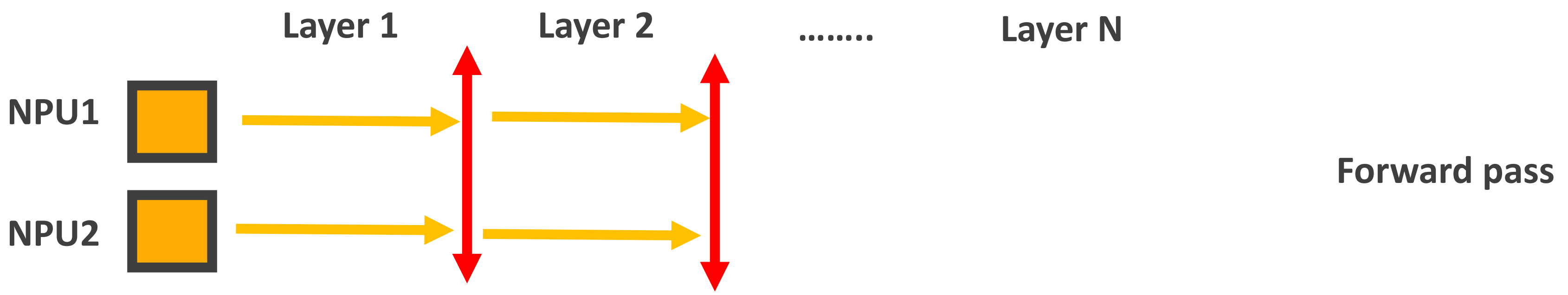
Parallelism: Model-Parallel Training

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

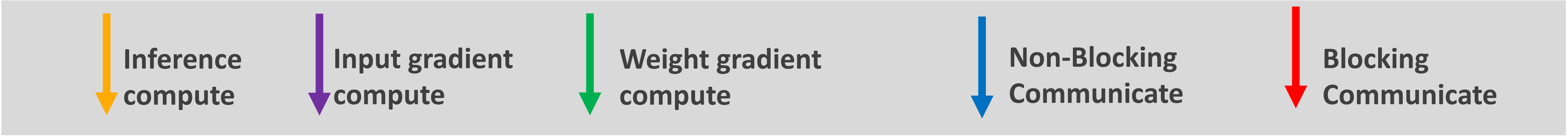


Parallelism: Model-Parallel Training

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

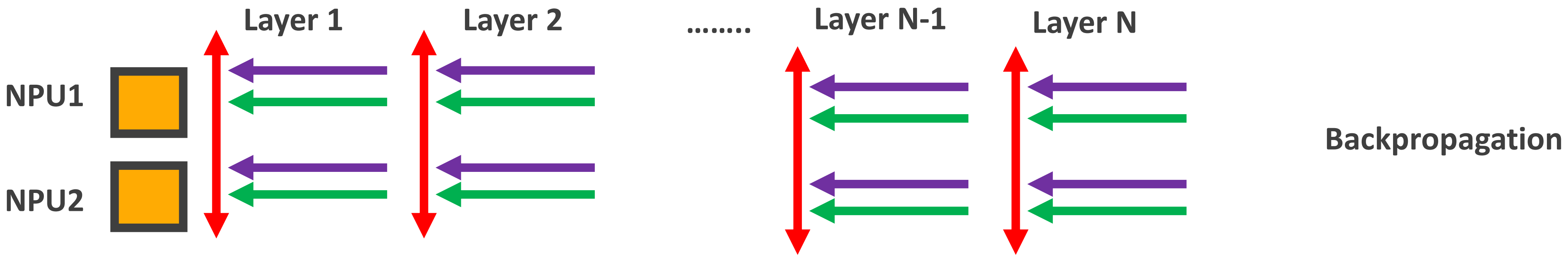


Flow-per-layer: 1.Compute output -> 2. issue output gradient comm -> 3.wait for gradient to be finished -> 4. go to the next layer



Parallelism: Model-Parallel Training

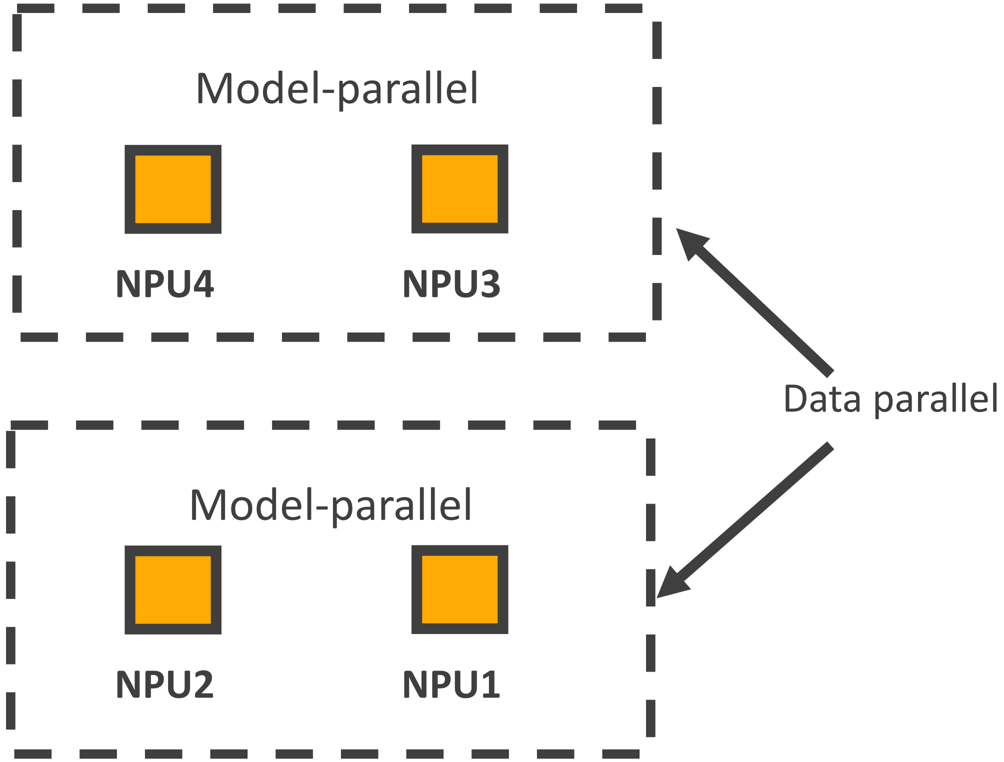
- Distribute Model across all nodes and replicate data along all nodes
- **Communicate input gradients** during the backpropagation pass.



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

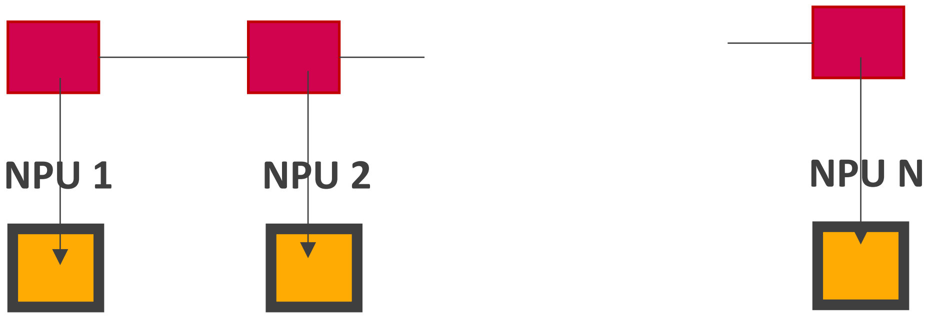
- Partition nodes into groups. Parallelism within a group is model-parallel, across the groups is data-parallel, or vice versa.



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

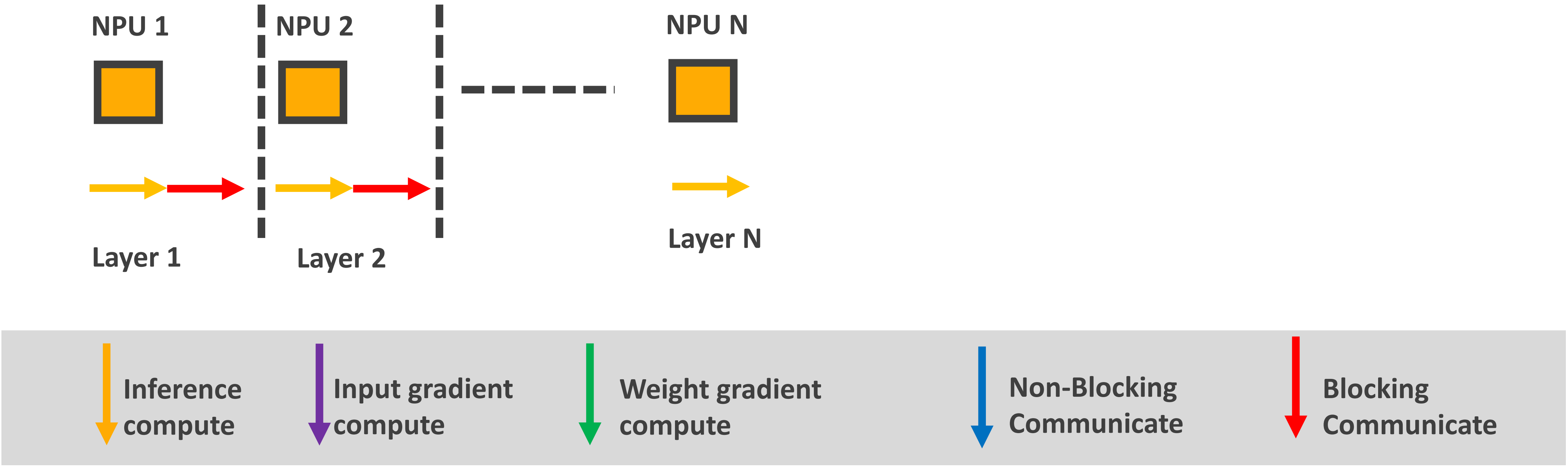
Parallelism: Pipelined Parallel

- Distribute DNN layers across all nodes.



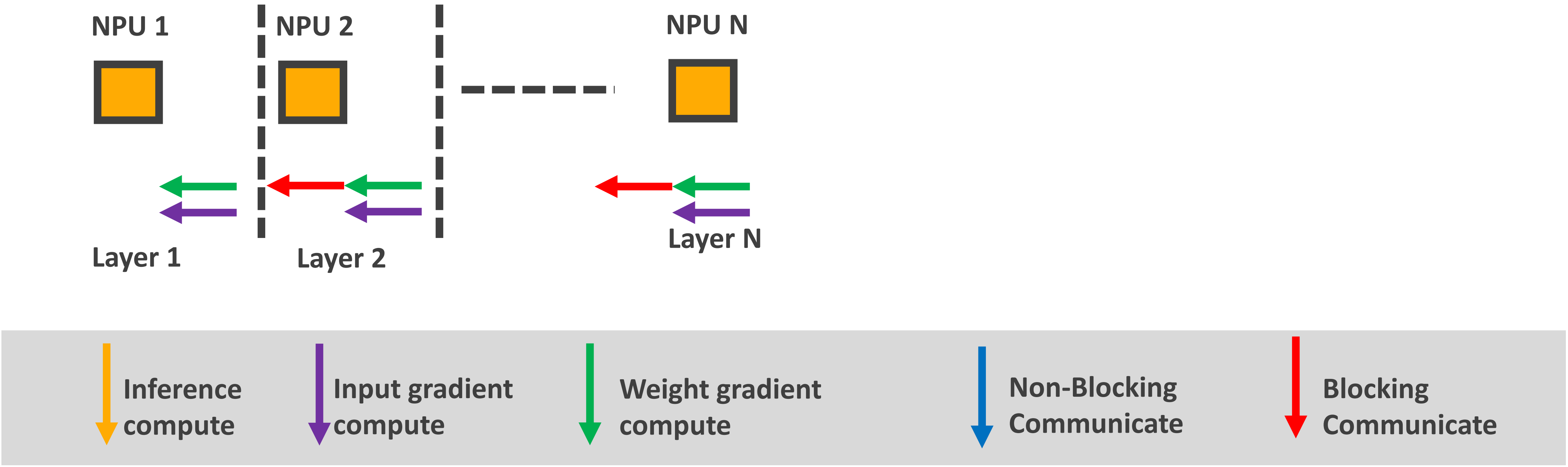
Parallelism: Pipelined Parallel

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



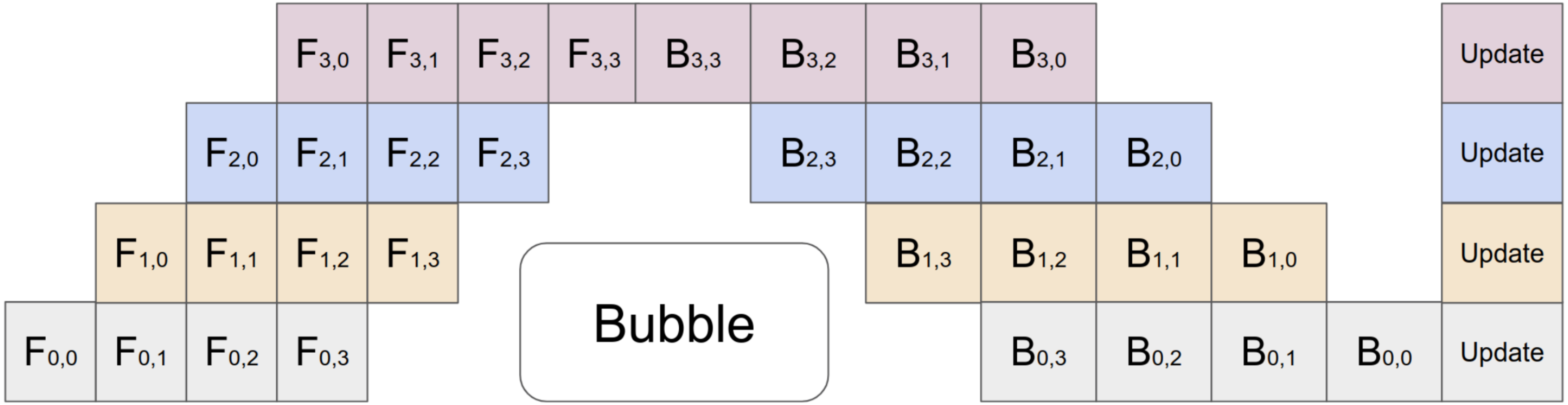
Parallelism: Pipelined Parallel

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



Parallelism: Pipelined Parallel

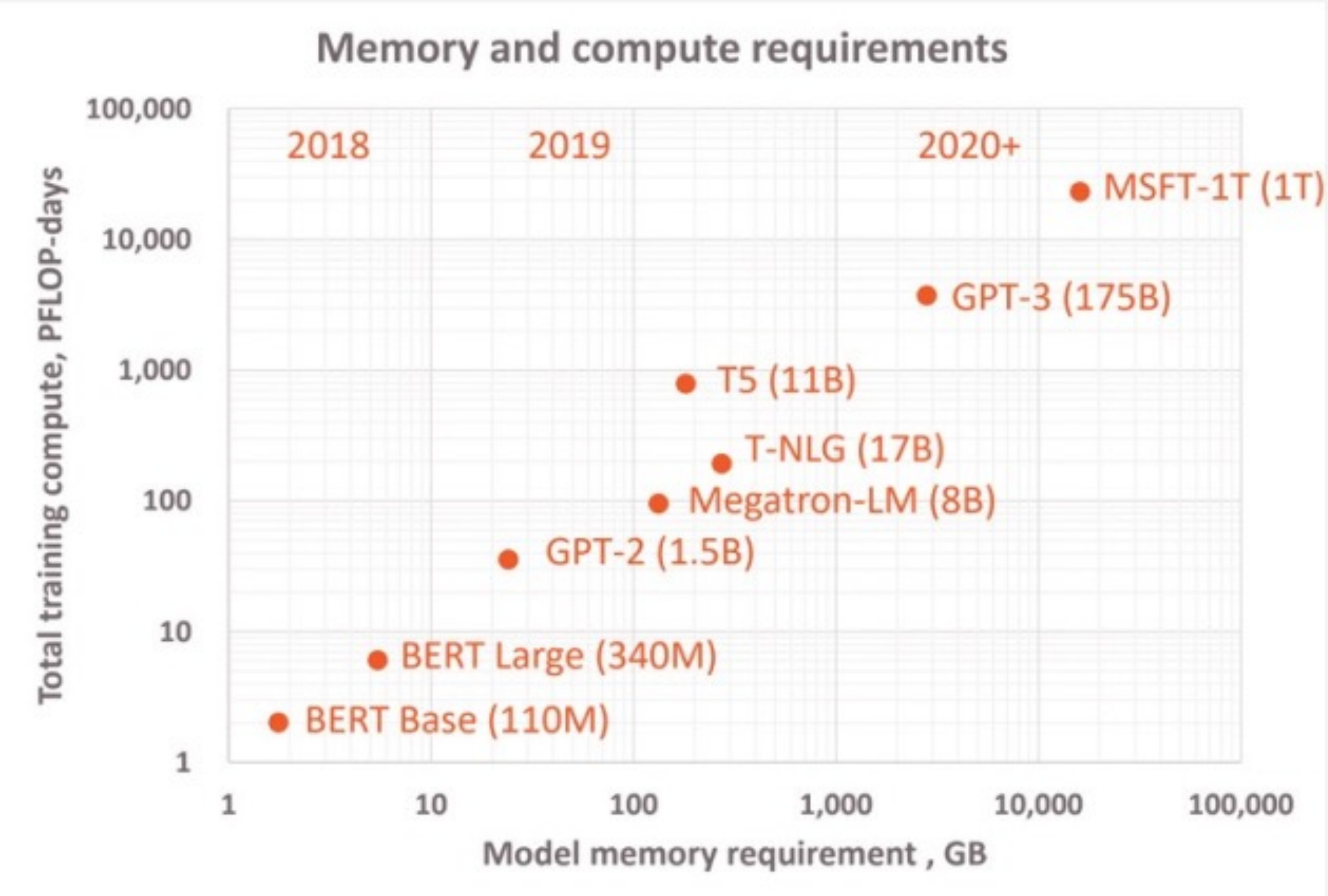
- Decompose minibatch into microbatches and propagate them to the pipeline in-order to enhance utilization
  - Challenge - bubbles



$F_{m,n}$: 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.

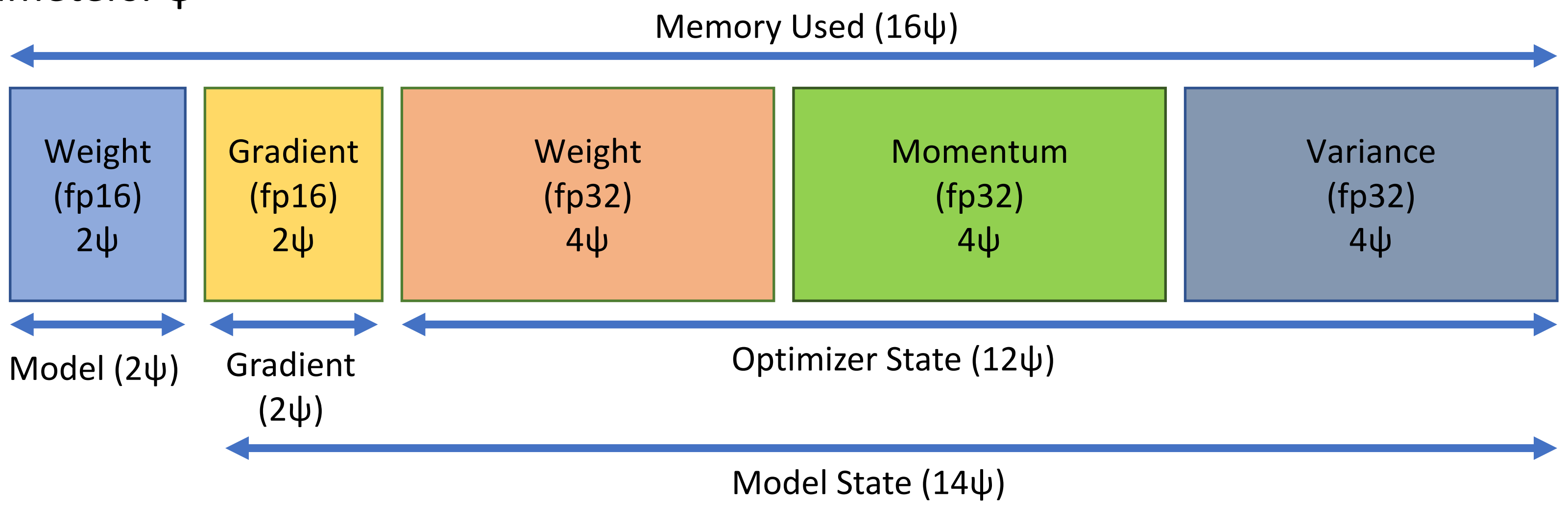
Tomorrow, **multi-Trillion** parameter models and beyond.

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!

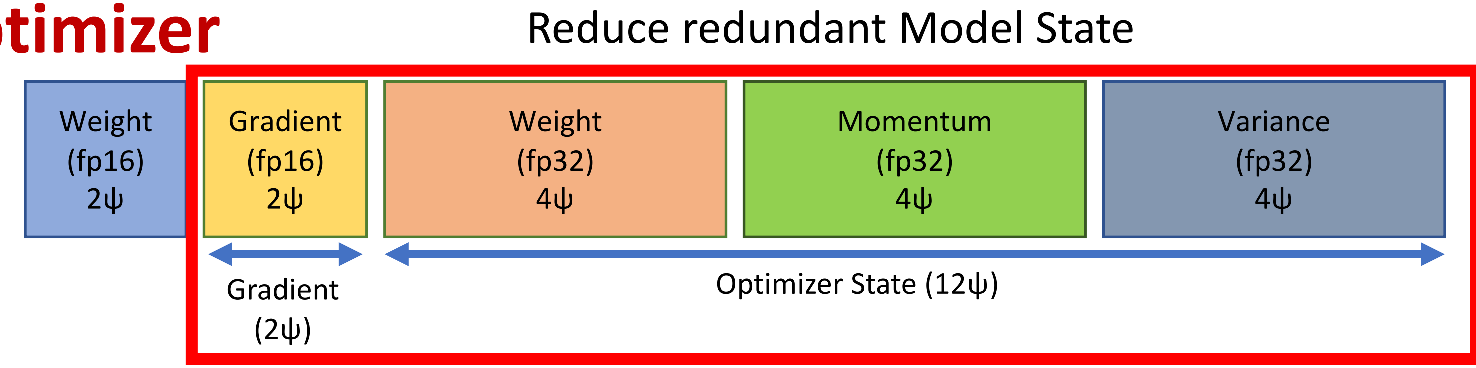
#Parameters: ψ

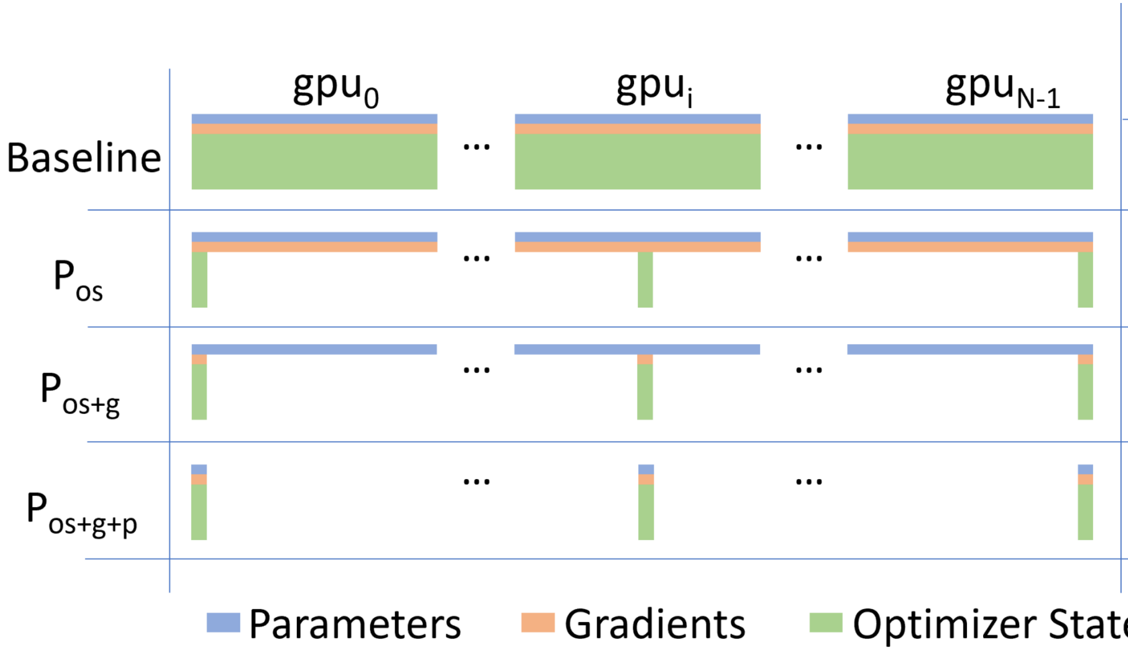


https://www.microsoft.com/en-us/research/blog/zero-deepspeed-new-system-optimizations-enable-training-models-with-over-100-billion-parameters/

Example 1: Microsoft ZeRO

- **ZeRO: Zero Redundancy Optimizer**
  - Partition Optimizer state
  - Partition Gradient state
- Memory vs Communication

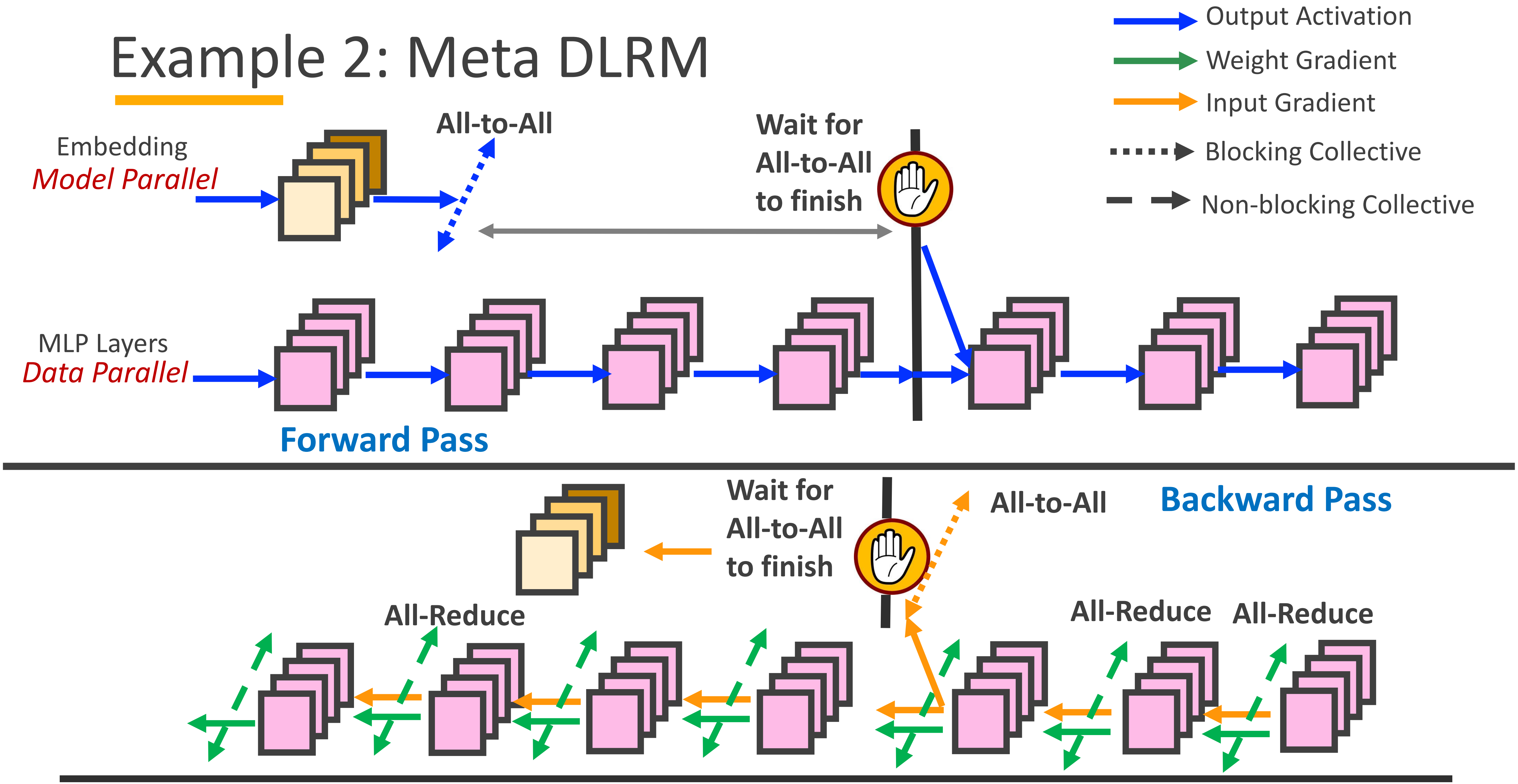




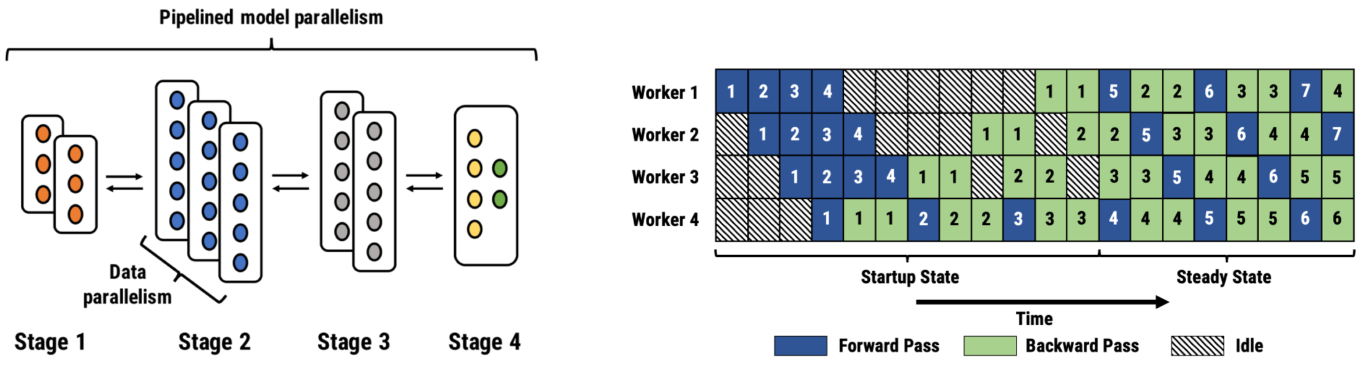
	Memory Consumed	K=12, Ψ=7.5B, N_d=64
Baseline	$(2+2+K)*\Psi$	120GB
P_{os}	$2\Psi+2\Psi+\frac{K*\Psi}{N_d}$	31.4GB
P_{os+g}	$2\Psi+\frac{(2+K)*\Psi}{N_d}$	16.6GB
P_{os+g+p}	$\frac{(2+2+K)*\Psi}{N_d}$	1.9GB

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

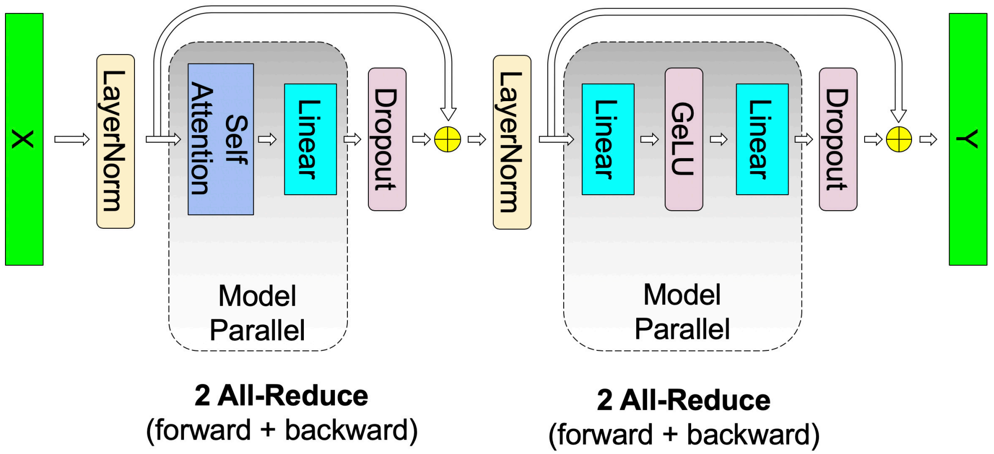
Example 2: Meta DLRM



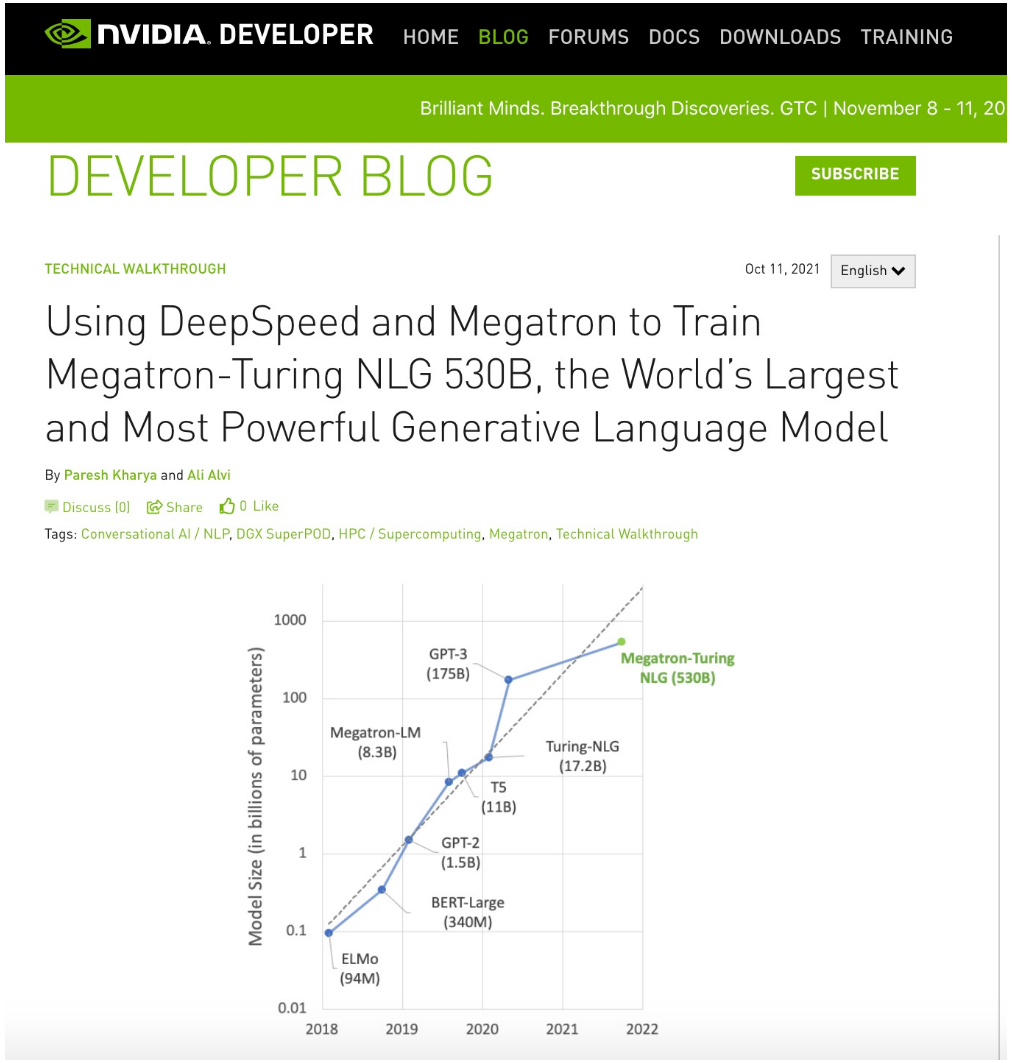
More recent examples



PipeDream (Microsoft)



MegatronLM (NVIDIA)



Distributed Training Stack

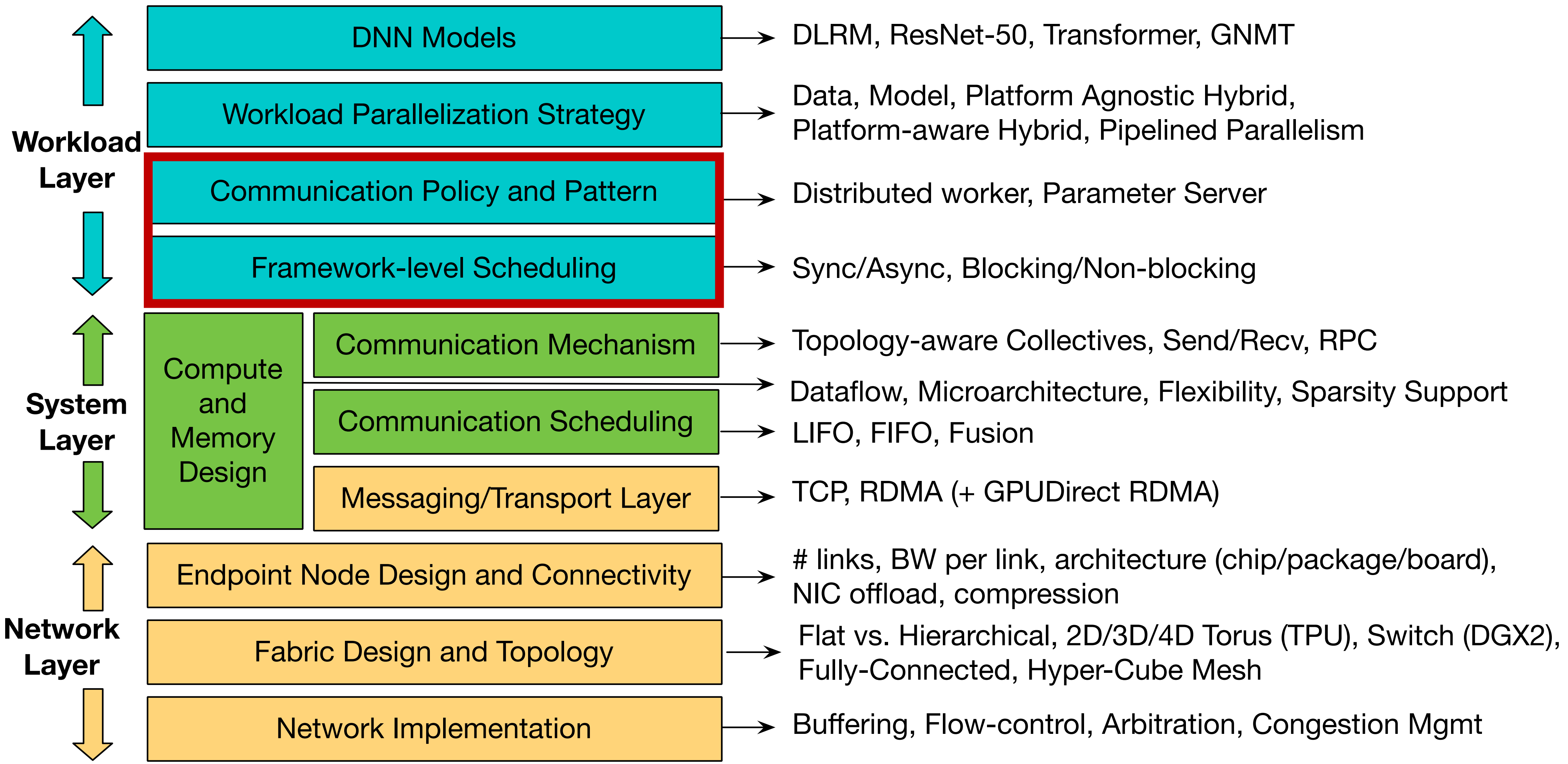


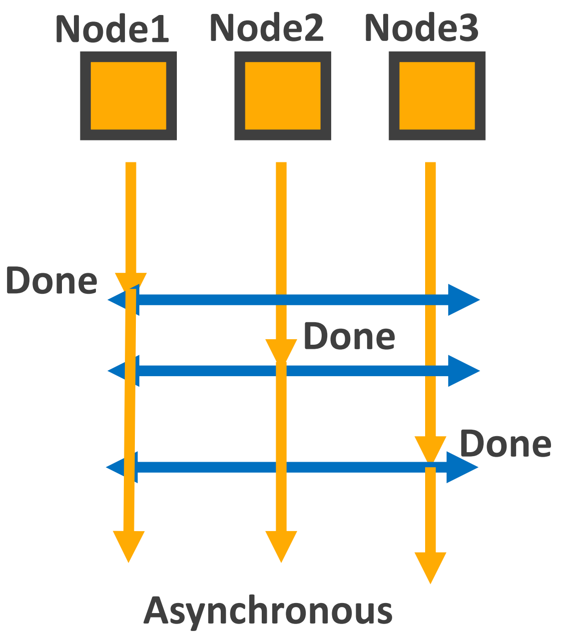
Figure Courtesy: Srinivas Sridharan (Facebook)

Model Parameter Update Mechanisms

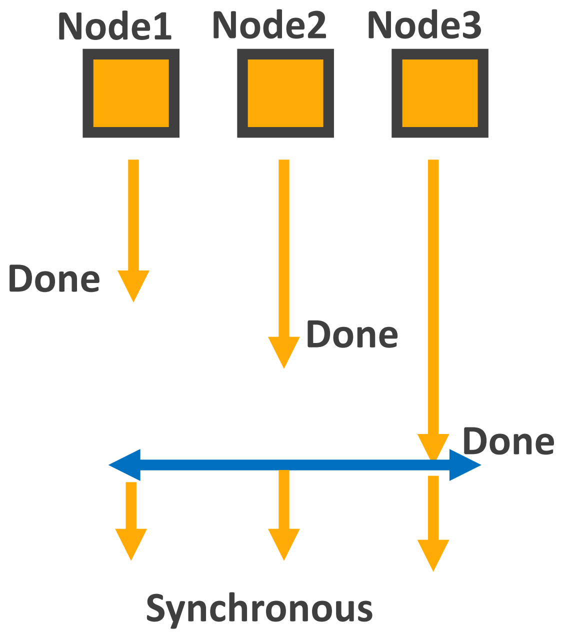
		Synchronization	
		Asynchronous	**Synchronous**
Communication Handling	**Parameter-server**	Centralized or Distributed	Centralized or Decentralized
	Collective-based	N/A	Distributed

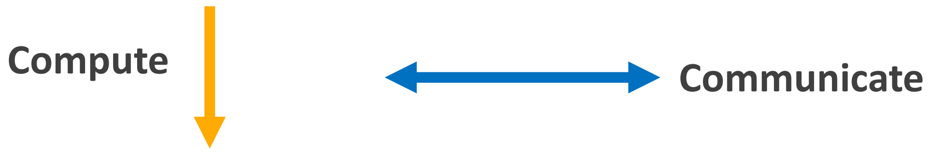
Synchronization: Sync. vs. Async. Training

- Defines when nodes should exchange data
  - Affects convergence time



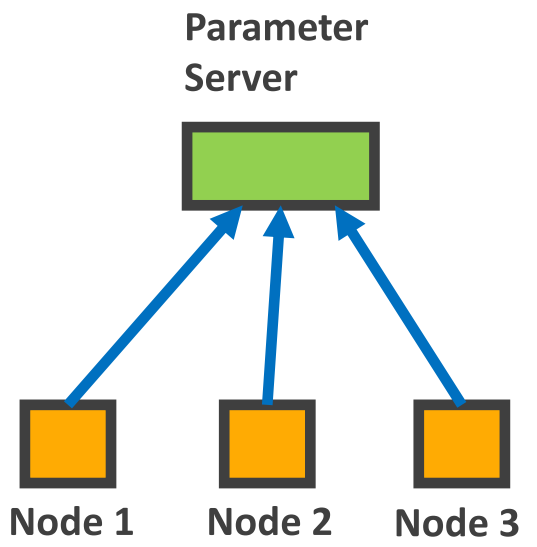
We focus on synchronous training



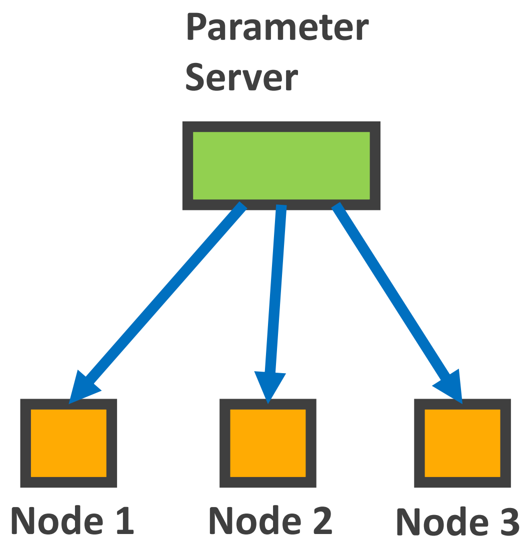


Communication Handling

- **Parameter Server**



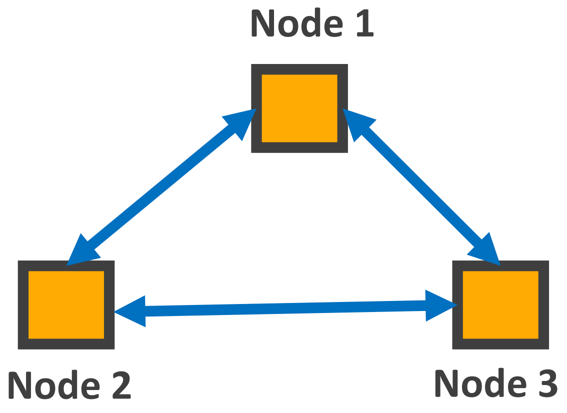
Step 1: Each node sends its model gradients to the parameter server to be reduced with other gradients and update the model

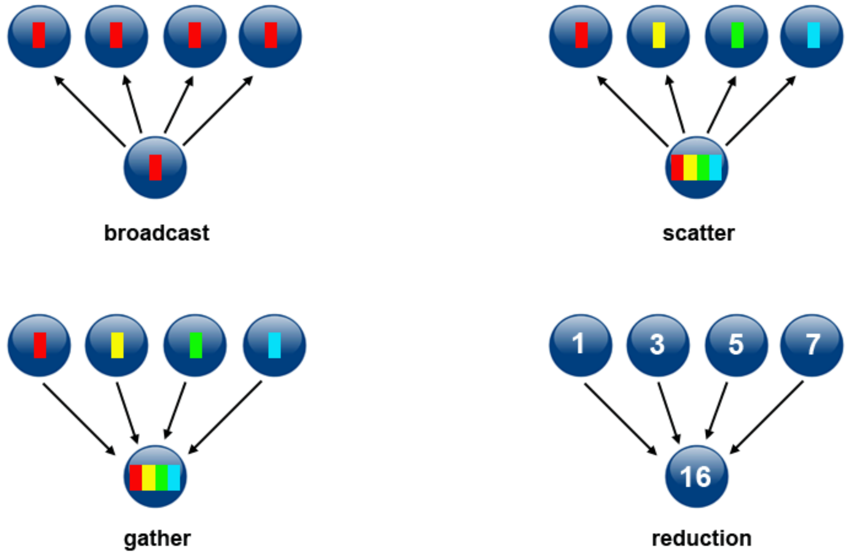


Step 2: The parameter server sends the updated model to the compute nodes to begin the new iteration.

Communication Handling

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





"Collective Communication" (from MPI)

More details later

Exchanging Output Activations or Input Gradients:

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

When are collectives needed?

	Model (i.e. weight) Updates	Input Gradient Exchange	Output Activation Exchange
Param-server	N	Data-parallel: **N** 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**

Distributed Training Stack

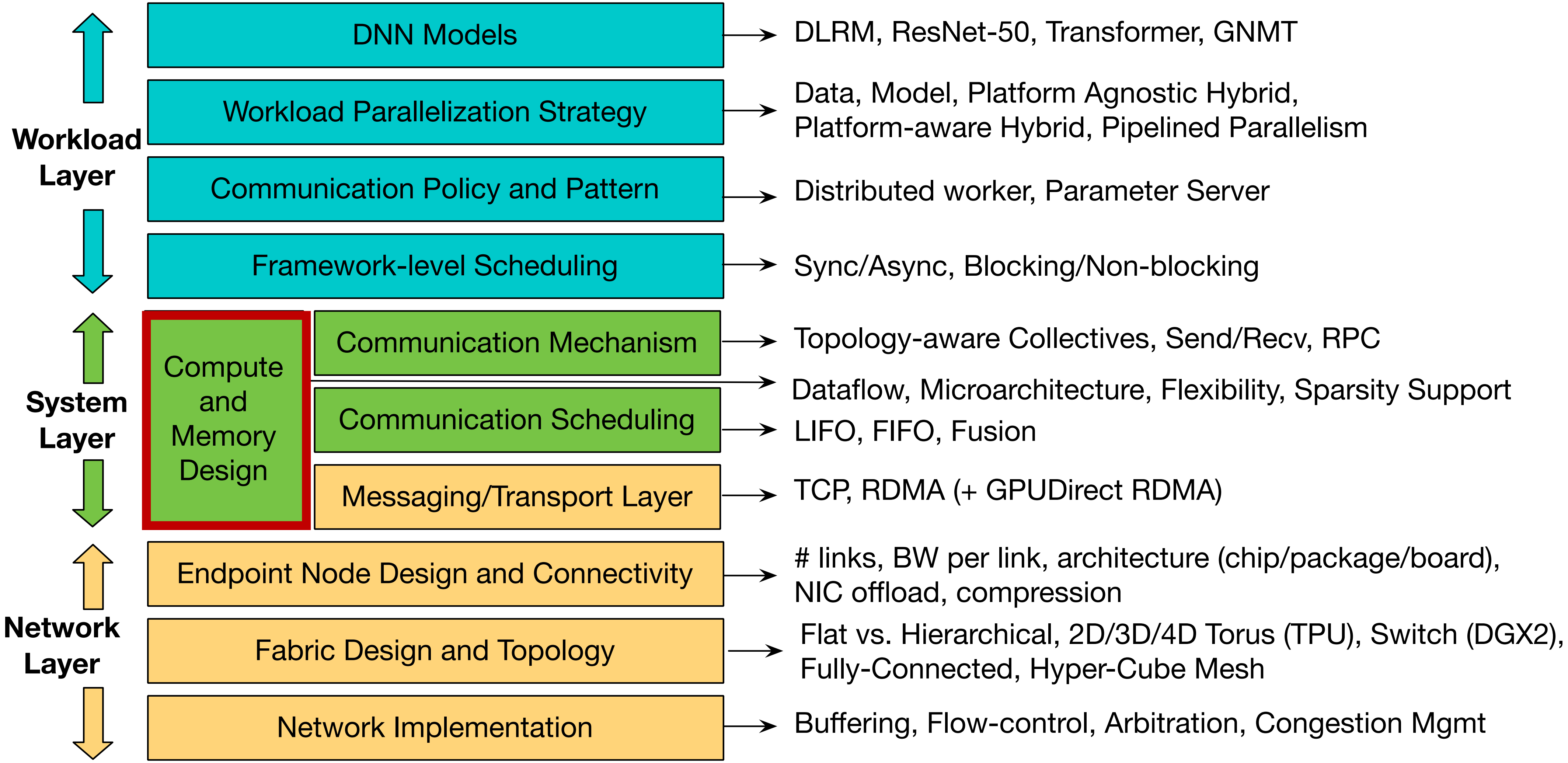
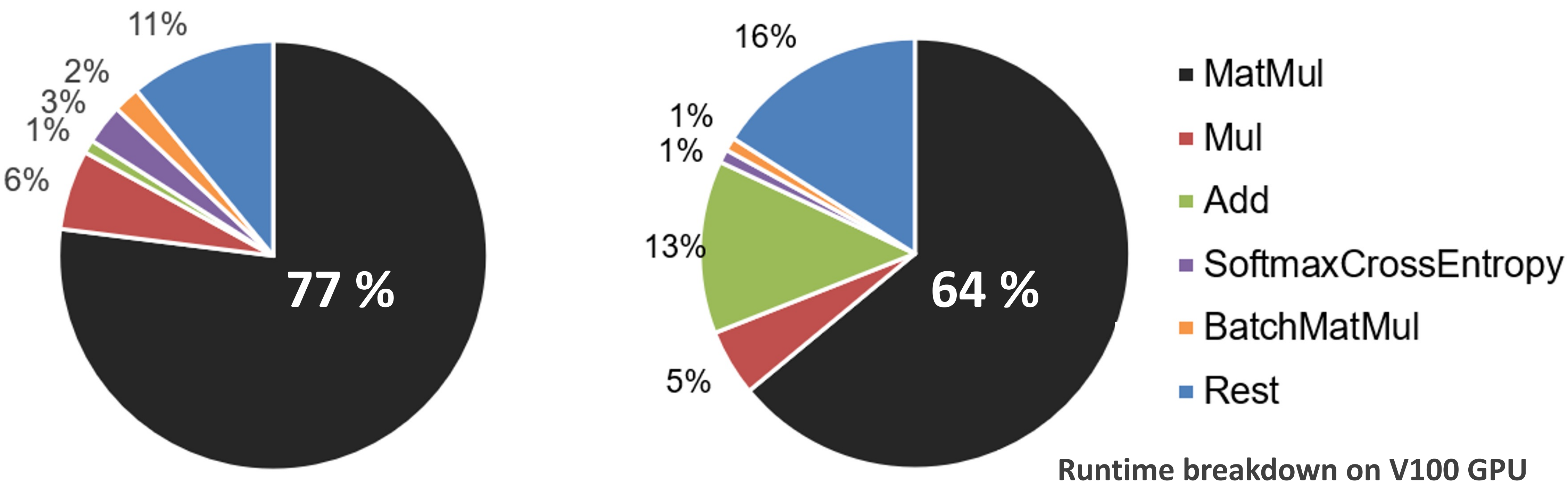


Figure Courtesy: Srinivas Sridharan (Facebook)

Key Compute Kernel during DL Training



Transformer (Language Understanding)

GNMT (Machine Translation)

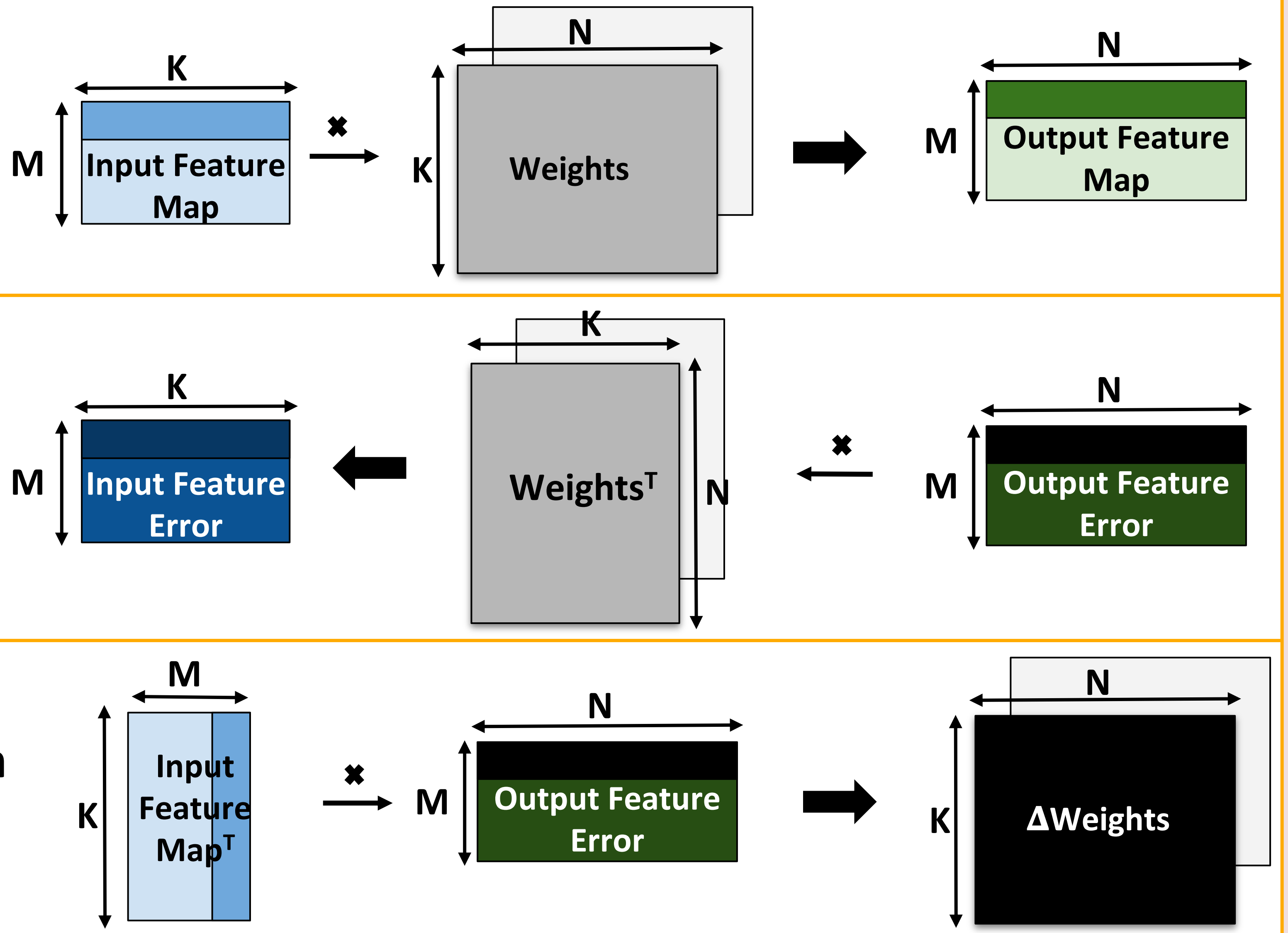
Matrix multiplications (GEMMs) consume around **70%** of the total runtime when training modern deep learning workloads.

GEMMs in Deep Learning

Forward Pass
(Inference and Training)

Backward Pass
(Training)

Gradient Computation
(Training)



GEMM MNK Dimension Representation

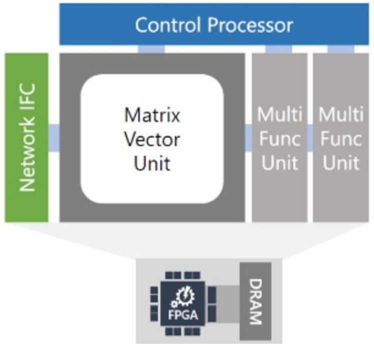
M dim: batch size

N dim: number of channels in the next layer

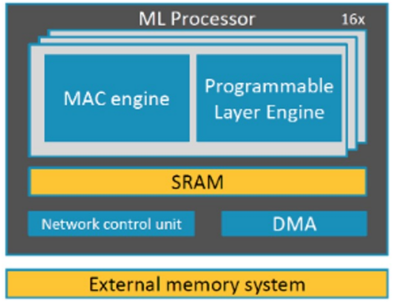
K dim: [H * W * C]

Hardware for Accelerating GEMMs

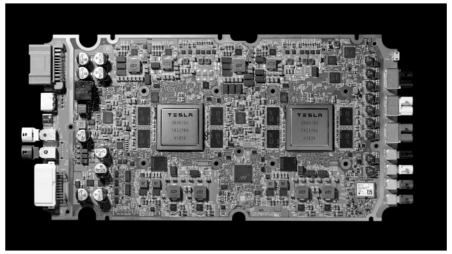
SIMD Architectures



Microsoft Brainwave

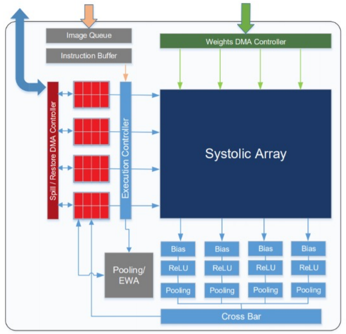


ARM Trillium

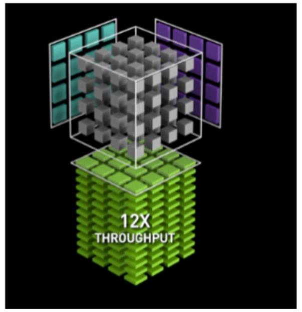


Tesla FSDC

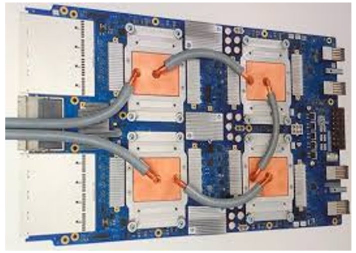
Systolic Architectures



Xilinx xDNN



Nvidia Tensor Cores



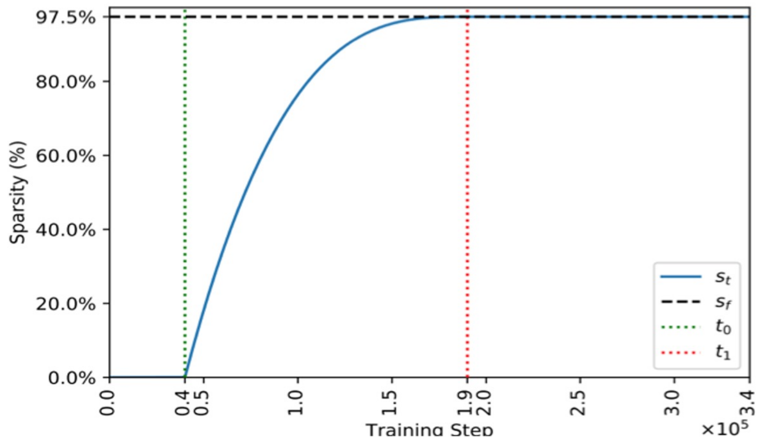
Google TPU

Key Feature:
- Specialized support for GEMMs
- Maximize HW TFLOPS

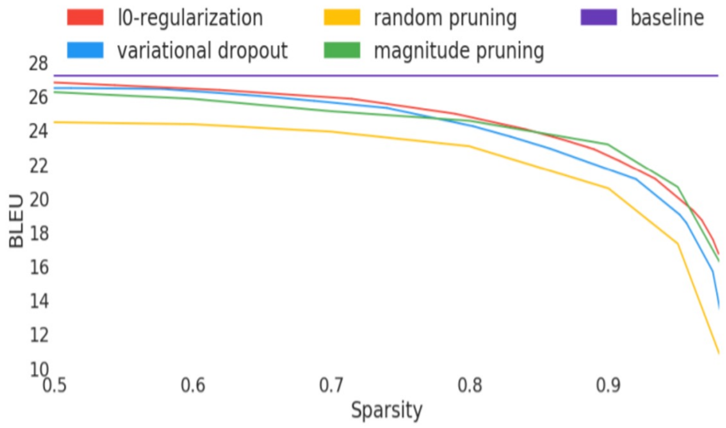
Workload Trends: Irregular & Sparse

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)!



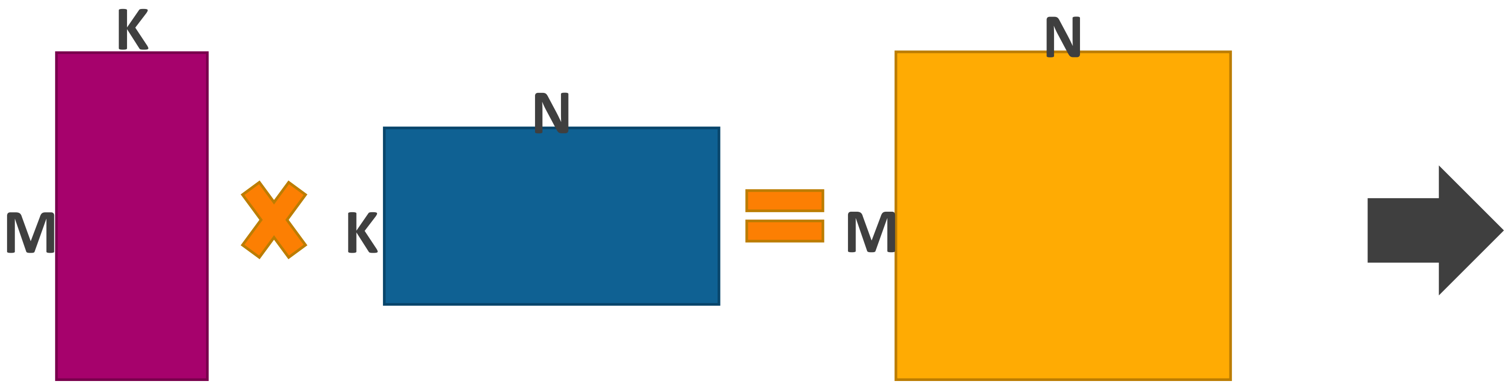
GNMT Pruning - Temporal Sparsity
(https://www.intel.ai/compressing-gnmt-models)

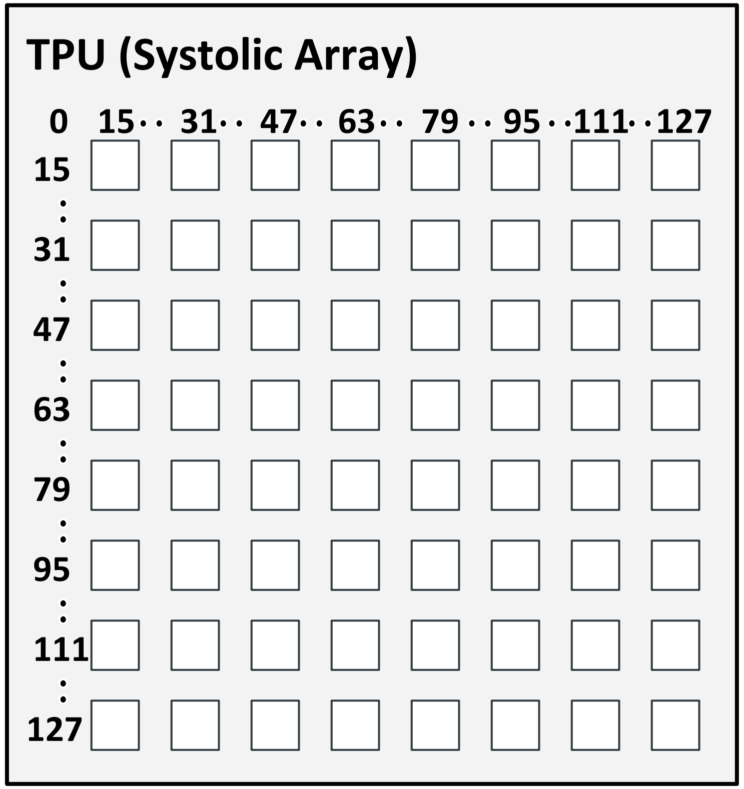


Transformer Sparsity - Impact on BLEU
(The State of Sparsity in Deep Neural Networks, Gale et al., arXiv)

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

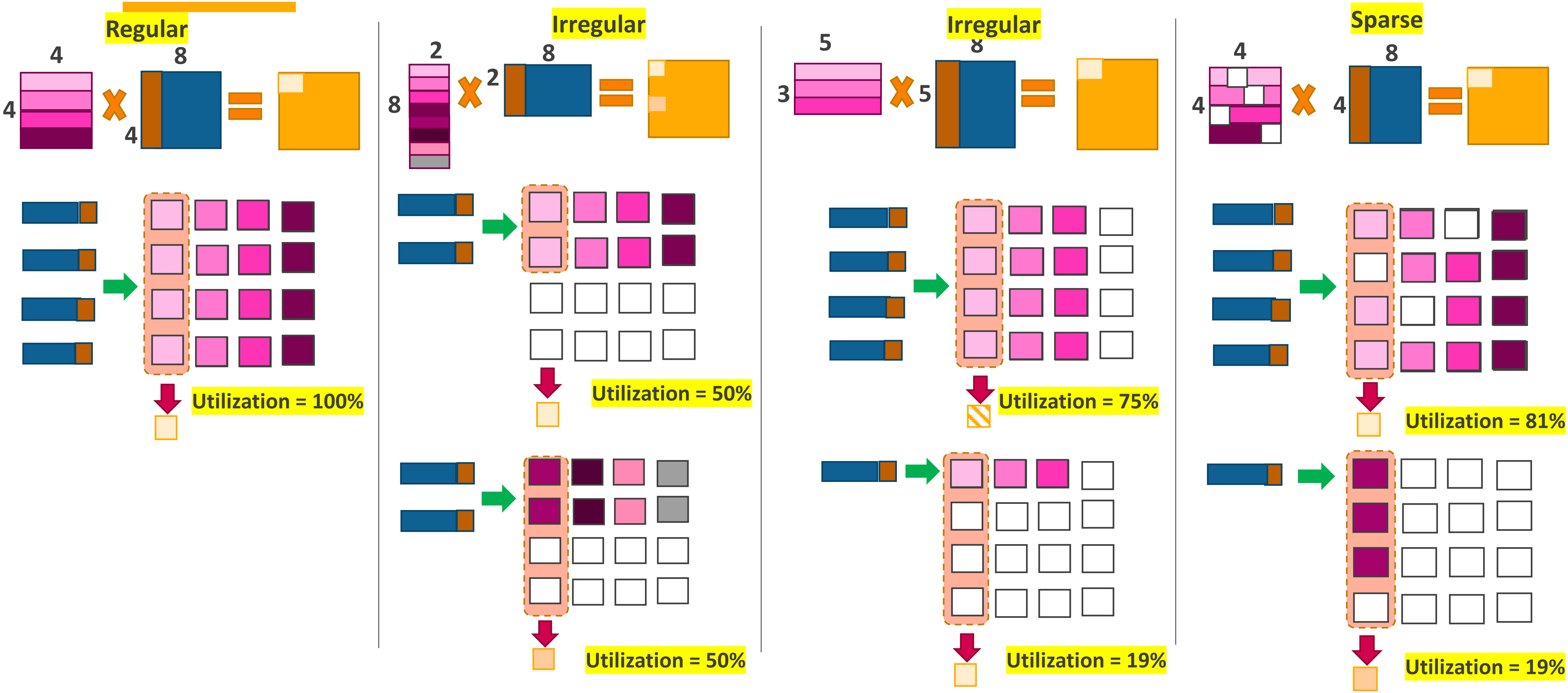
Challenges – Mapping Utilization





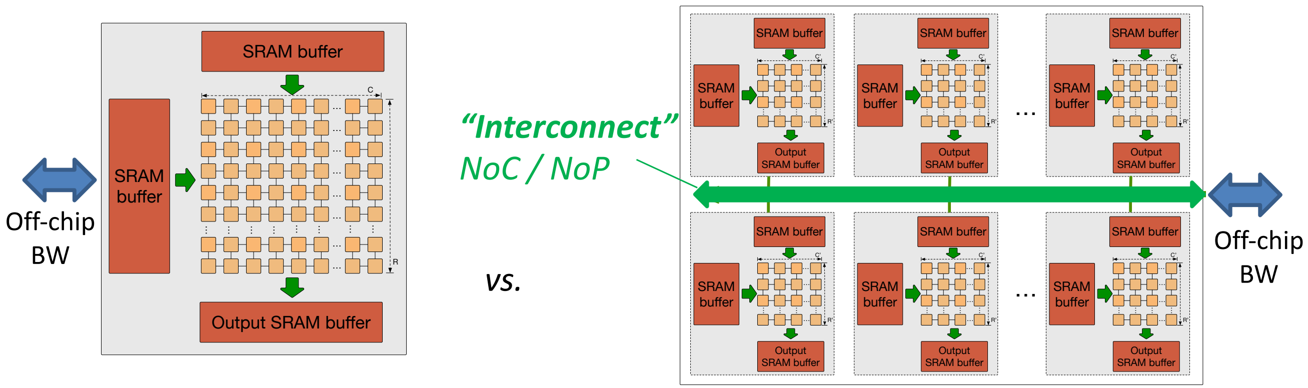
***** Assuming MK matrix is streaming and KN matrix is stationary. (aka weight stationary)***

Challenges – Mapping Utilization

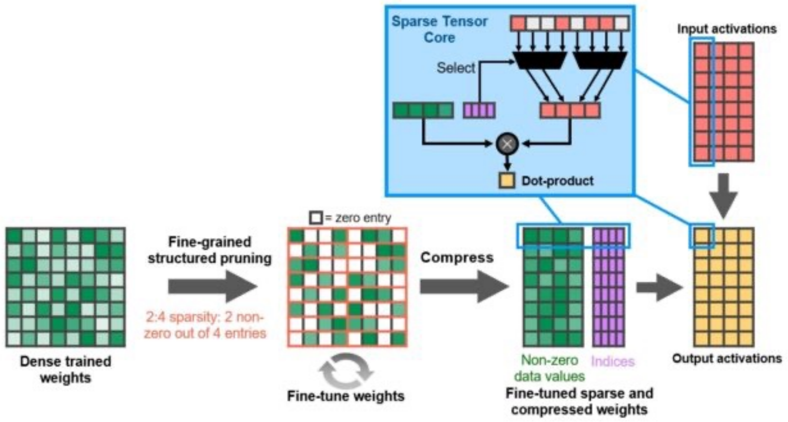


Enhancing Utilization

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

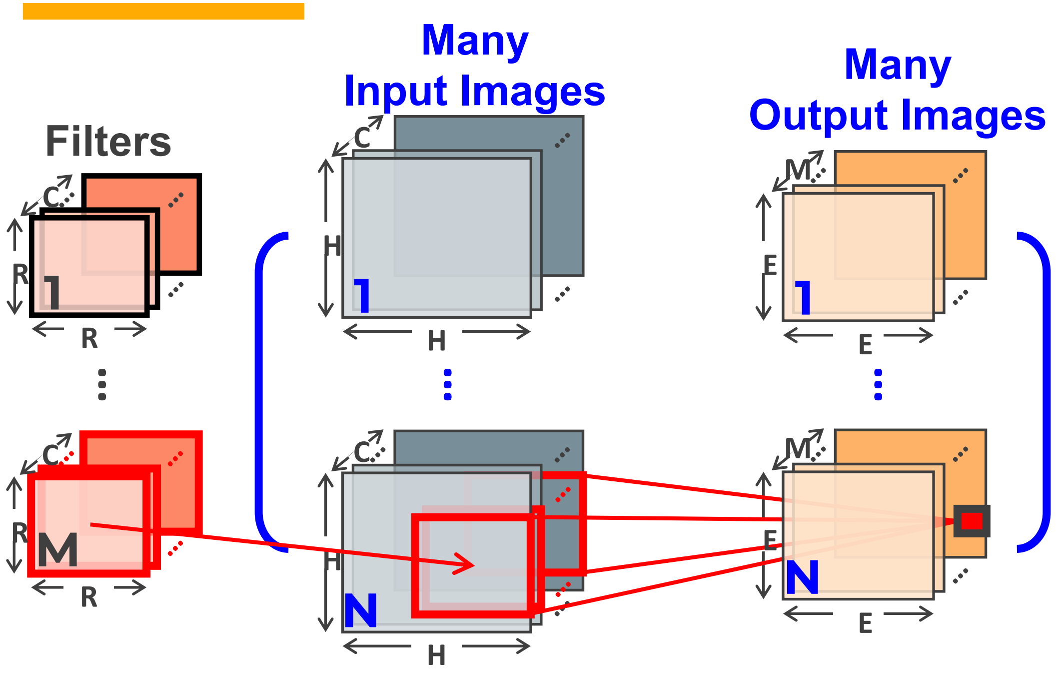


- **Handling Sparse GEMMs**
  - Structured Sparsity Support
    - E.g., NVIDIA A100
  - Unstructured Sparsity Support
    - Active research going on

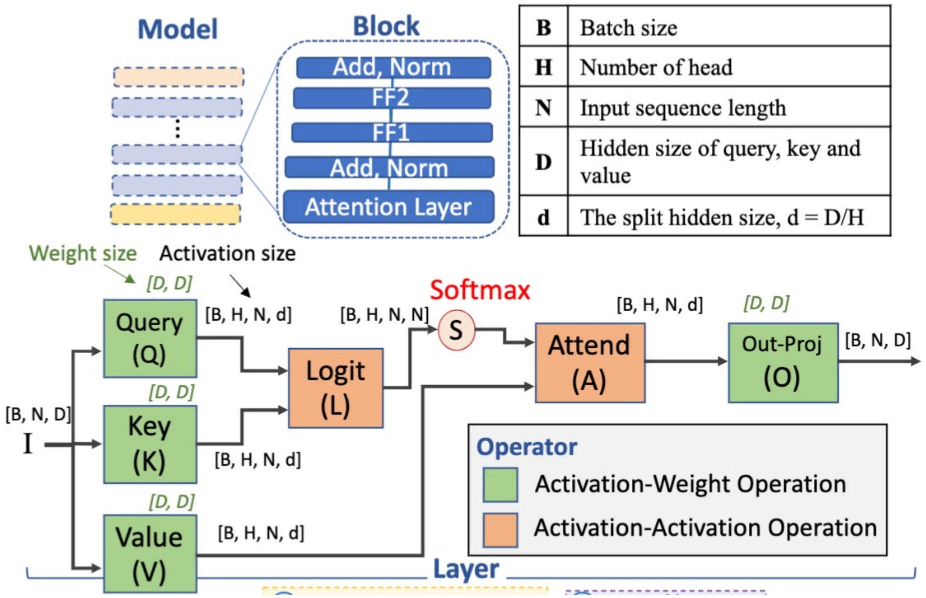


NVIDIA A100 supports 4:2 structured sparsity

Workload Trends: Low Op Intensity

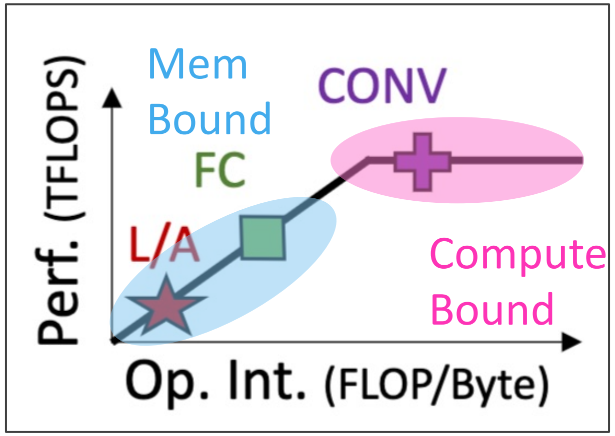


Convolution Layers

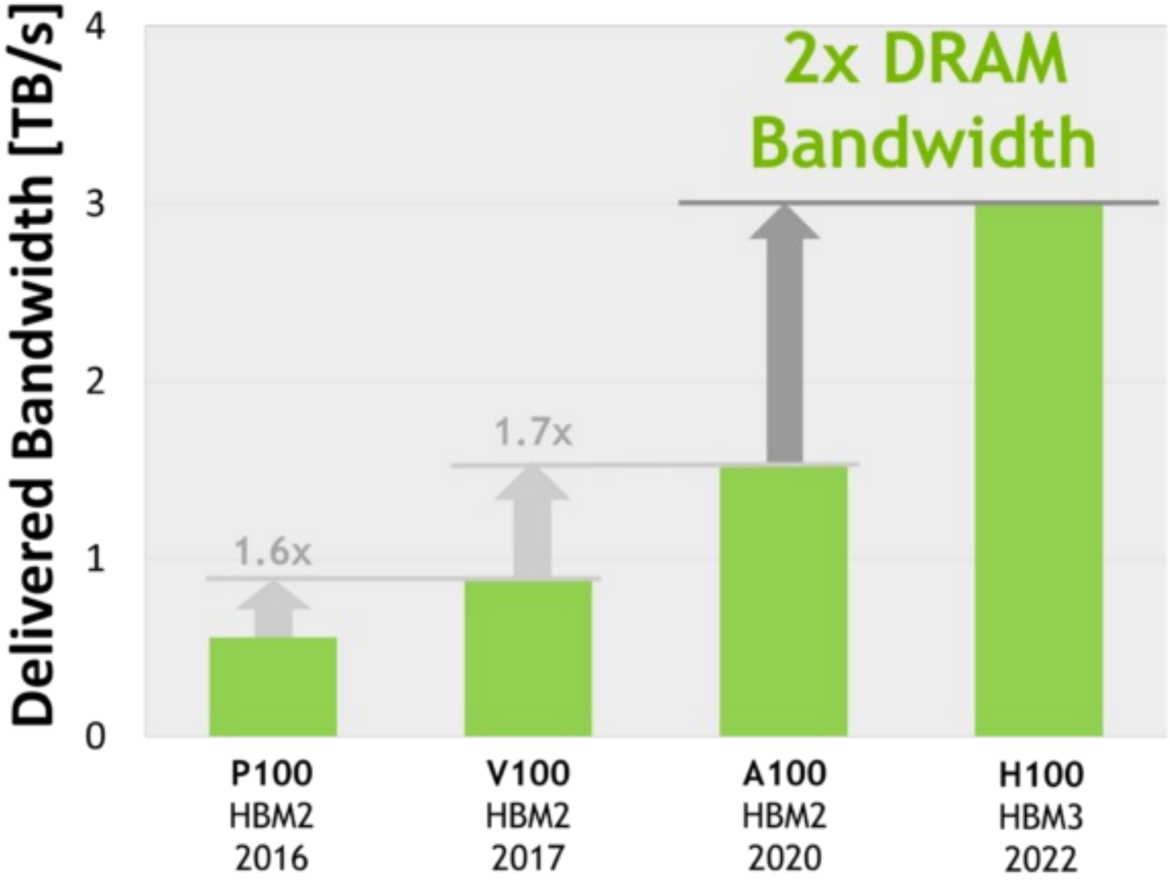


Attention Layer in Transformer Models

Challenges:

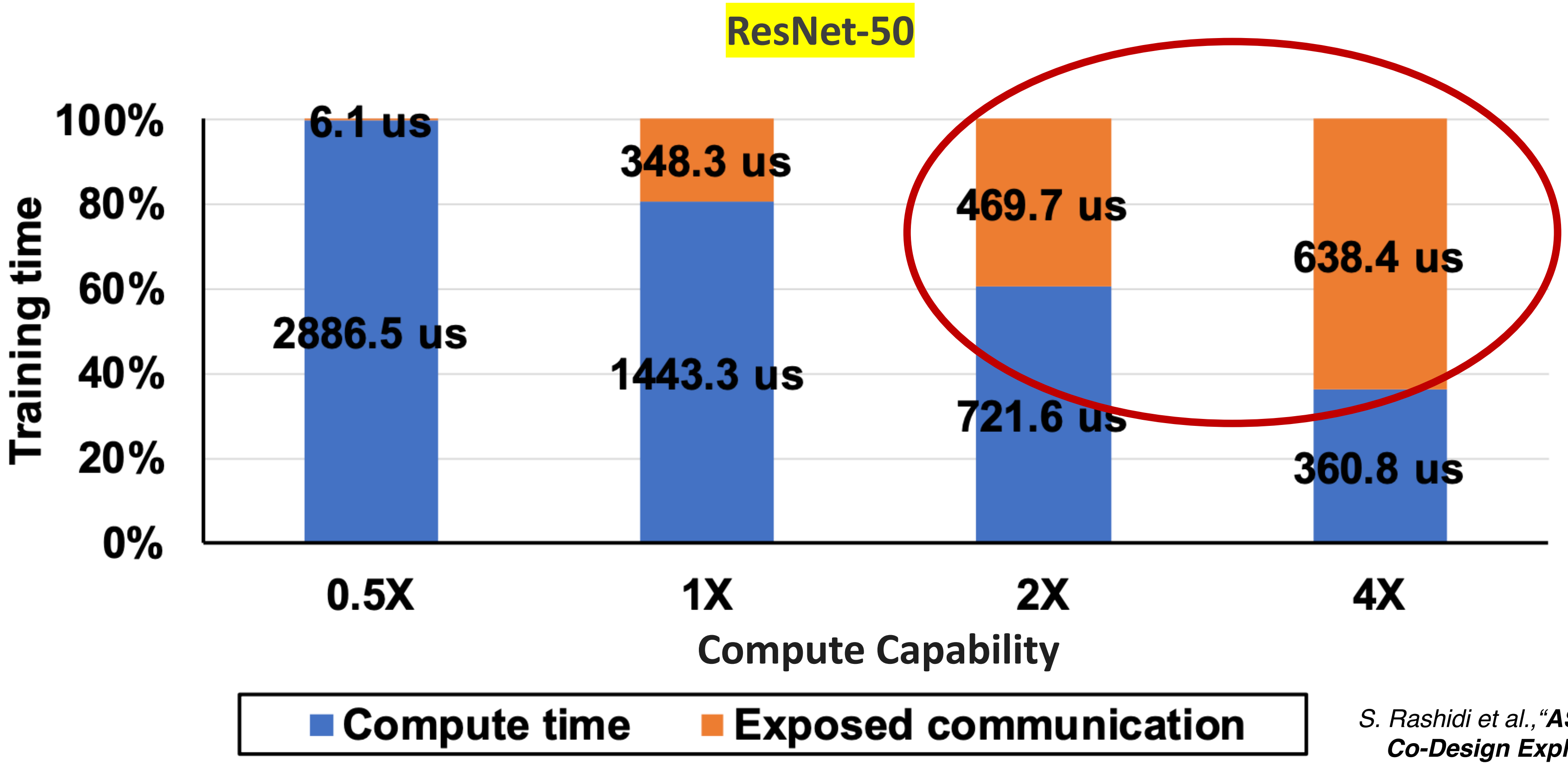


Enhancing Memory Bandwidth



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

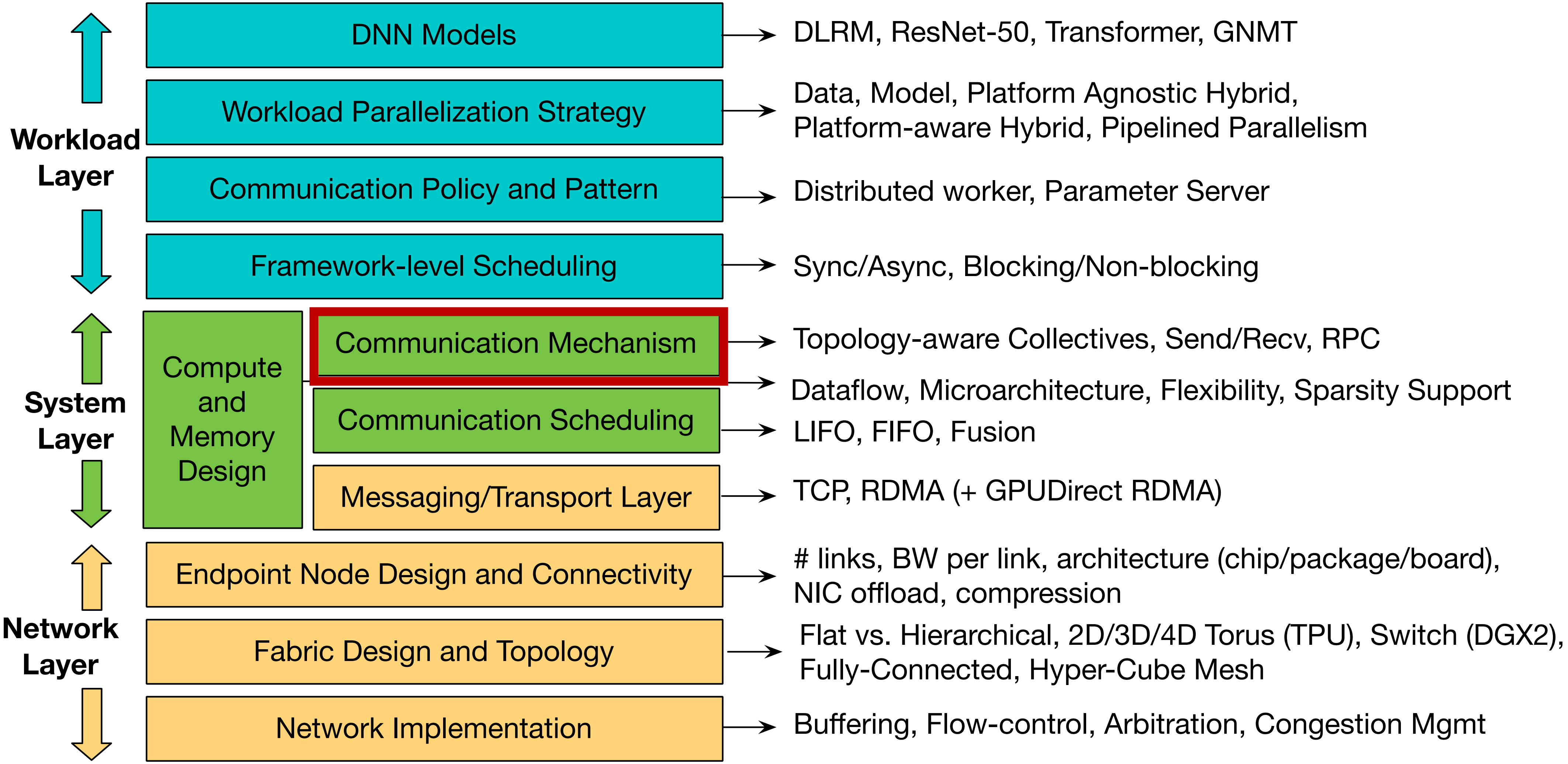
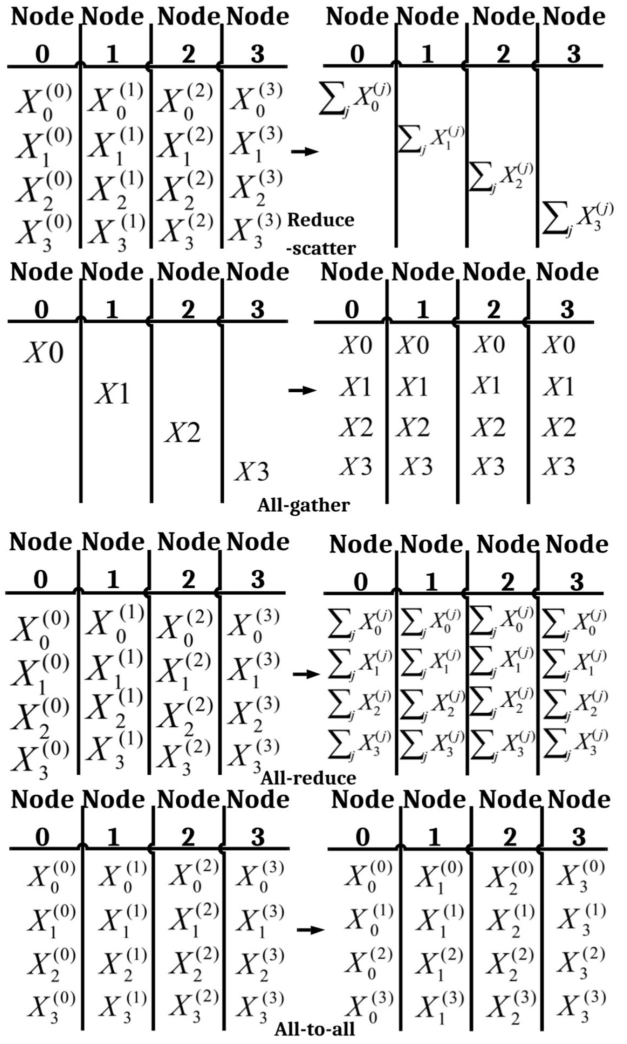


Figure Courtesy: Srinivas Sridharan (Facebook)

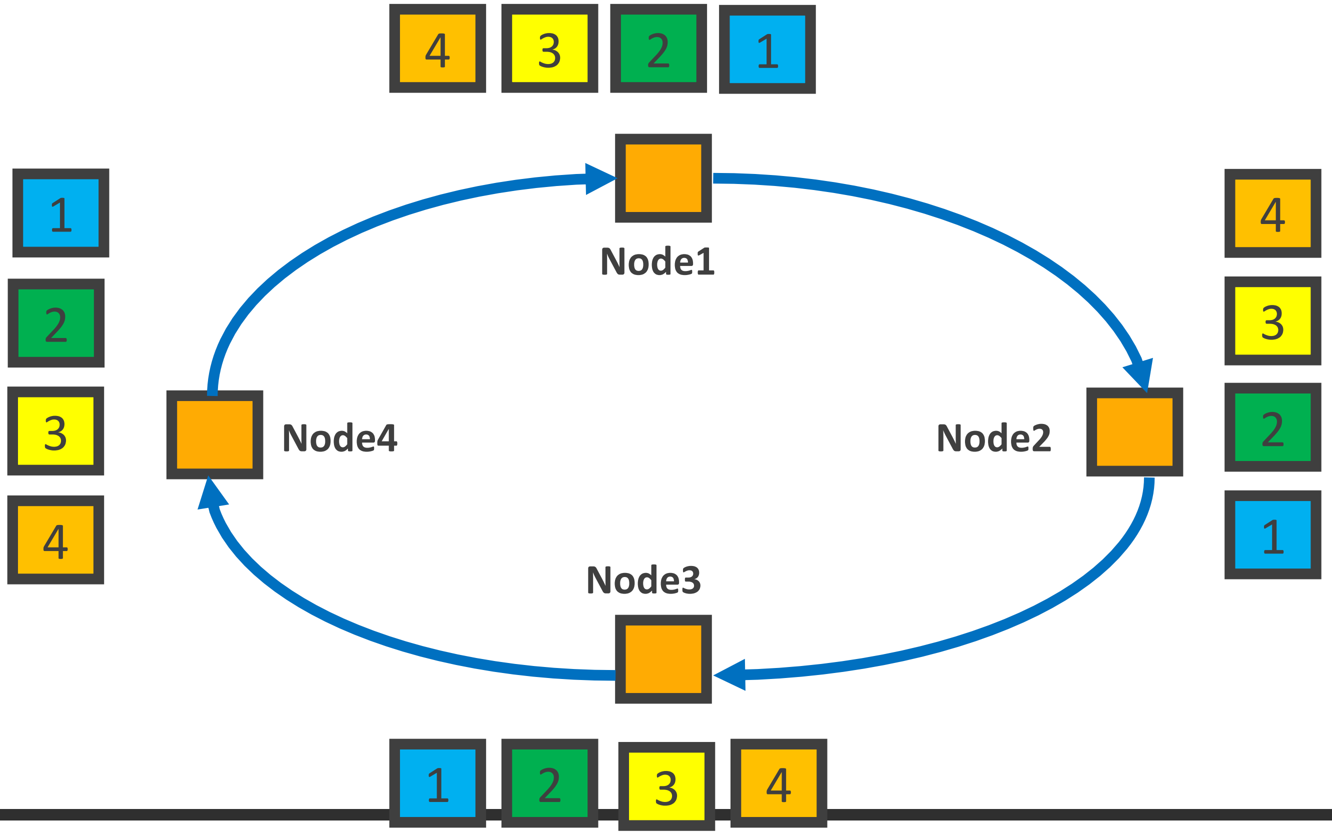
Different Kinds of Collective Algorithms

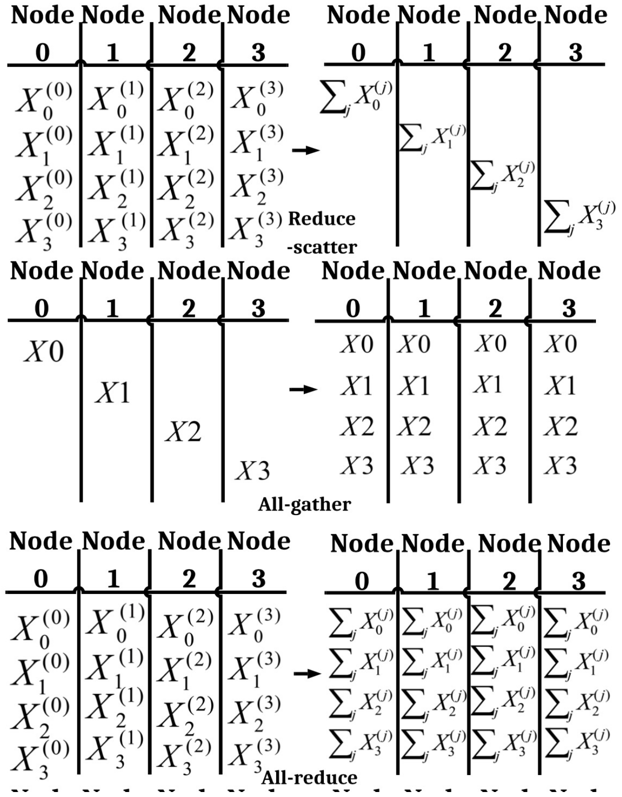
- **Reduce-Scatter:**
  - Used during input-output exchange due to model-parallelism
  - Implementation Algorithms: **Ring-Based, Direct-based, etc.**
- **All-Gather:**
  - Used during input-output exchange due to model-parallelism
  - Implementation Algorithms: **Ring-Based, Direct-based, etc.**
- **All-Reduce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

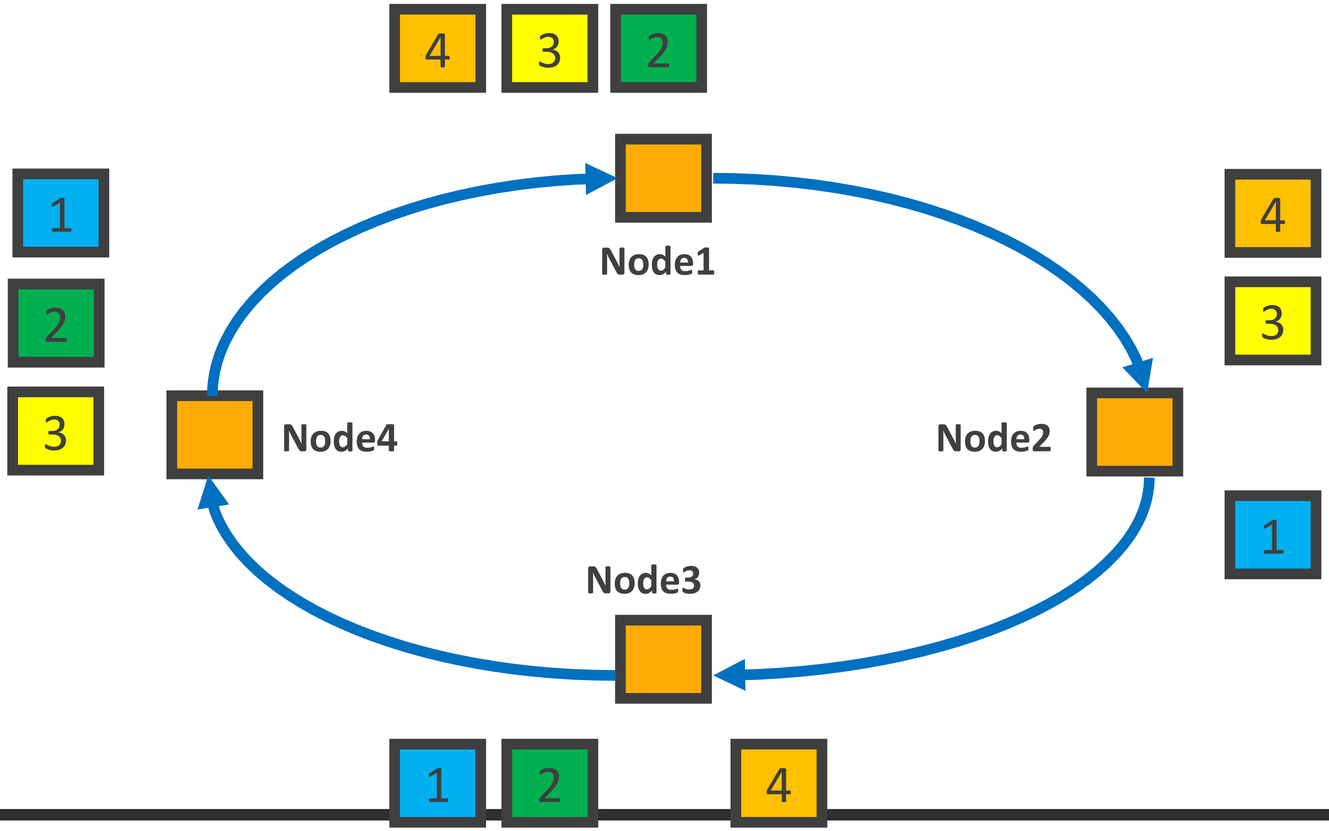
- A ring with N nodes partitions data to N messages
- Collective Communication Flow:

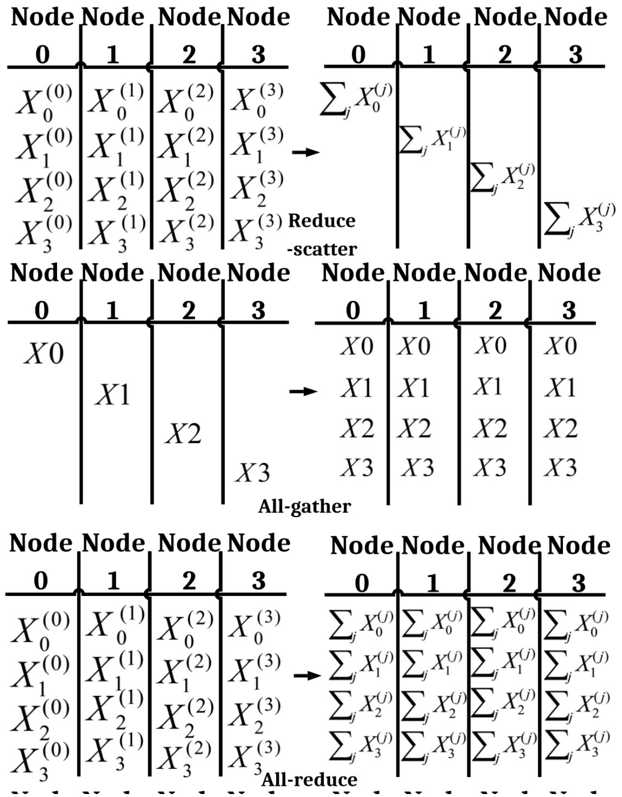




Example: Ring Based All-Reduce

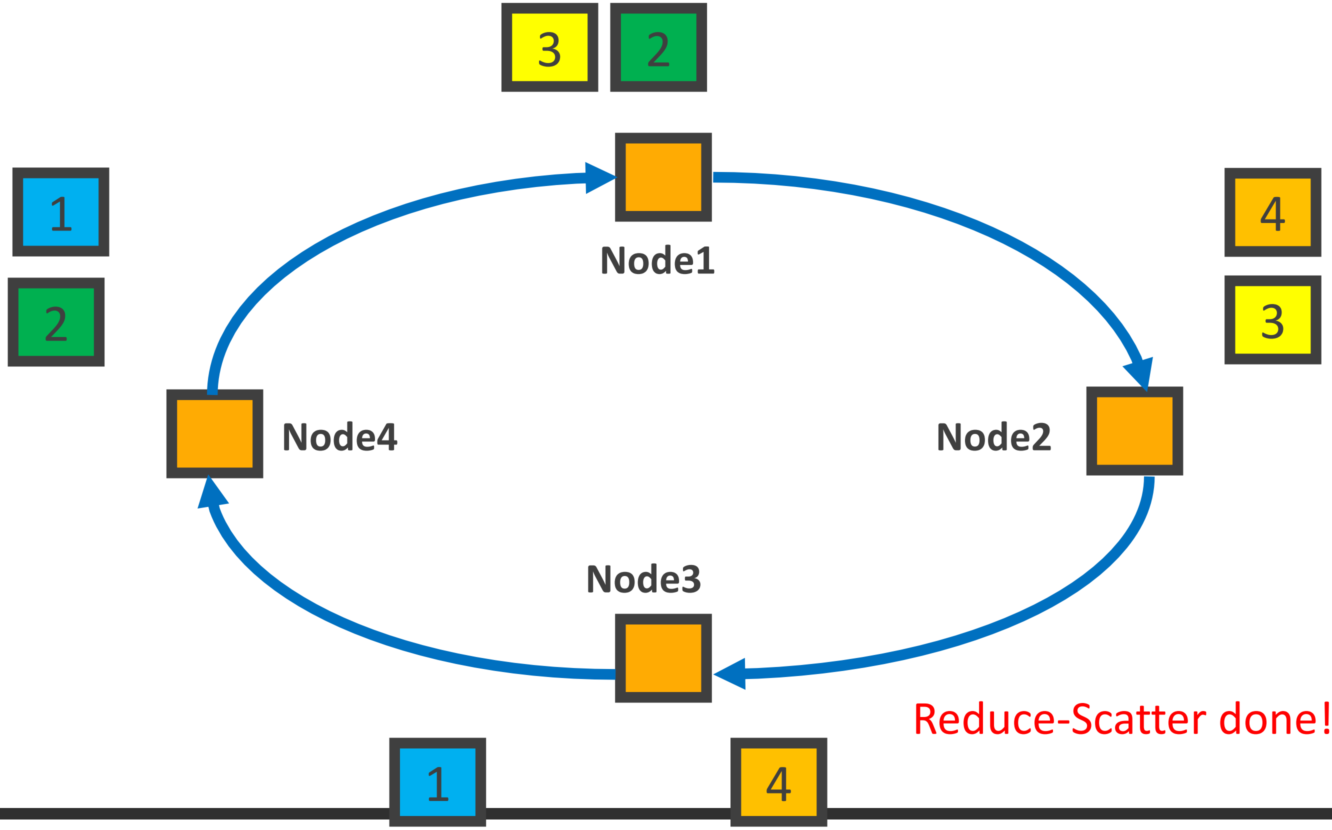
- A ring with N nodes partitions data to N messages
- Collective Communication Flow:



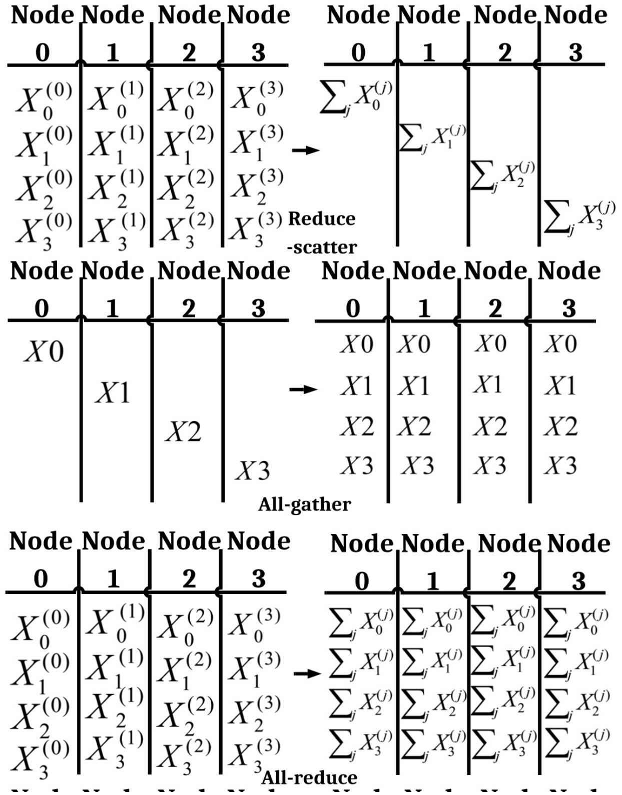


Example: Ring Based All-Reduce

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

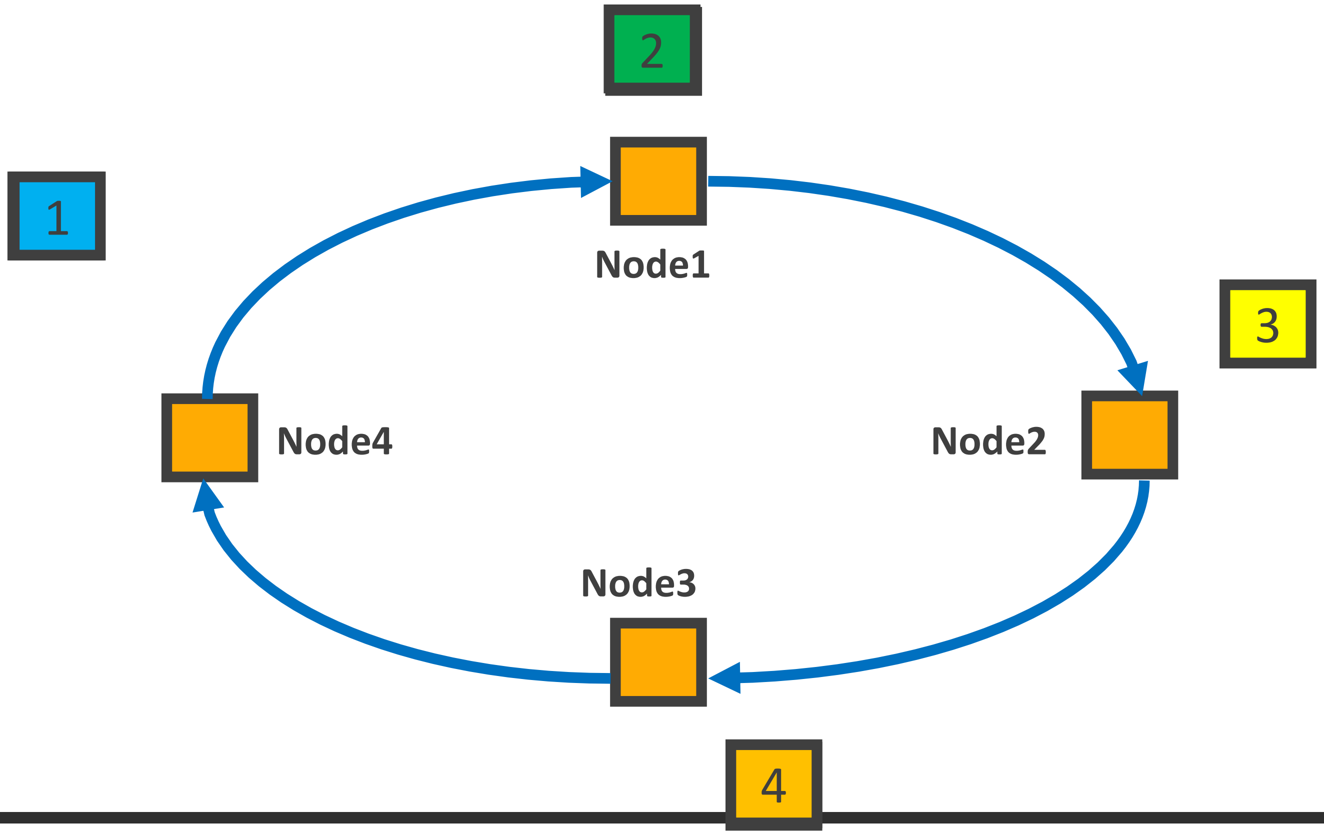


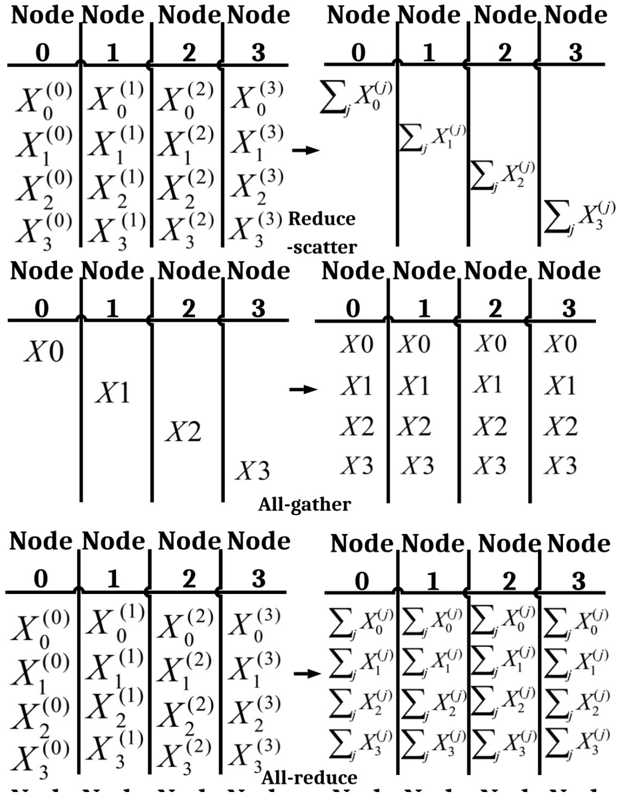
Reduce-Scatter done!



Example: Ring Based All-Reduce

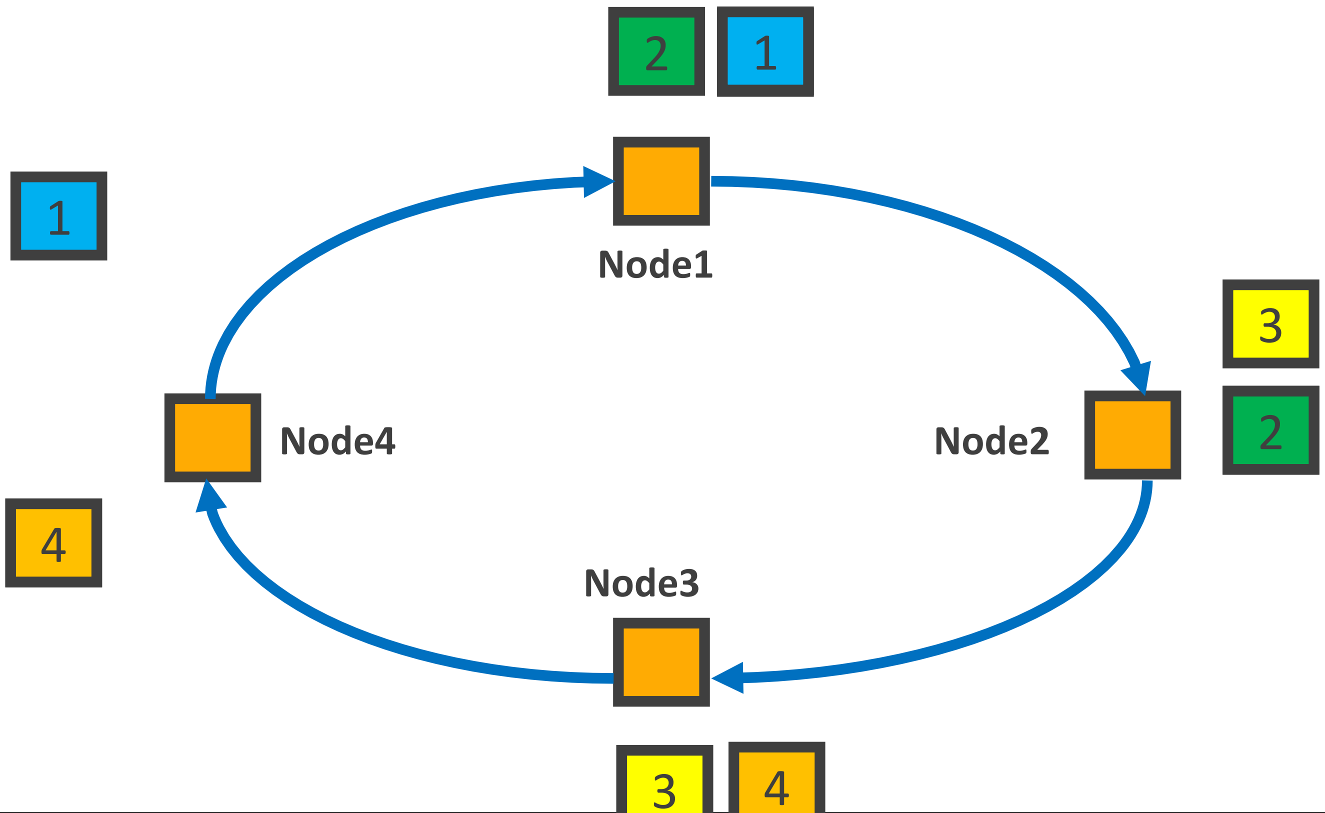
- A ring with N nodes partitions data to N messages
- Collective Communication Flow:

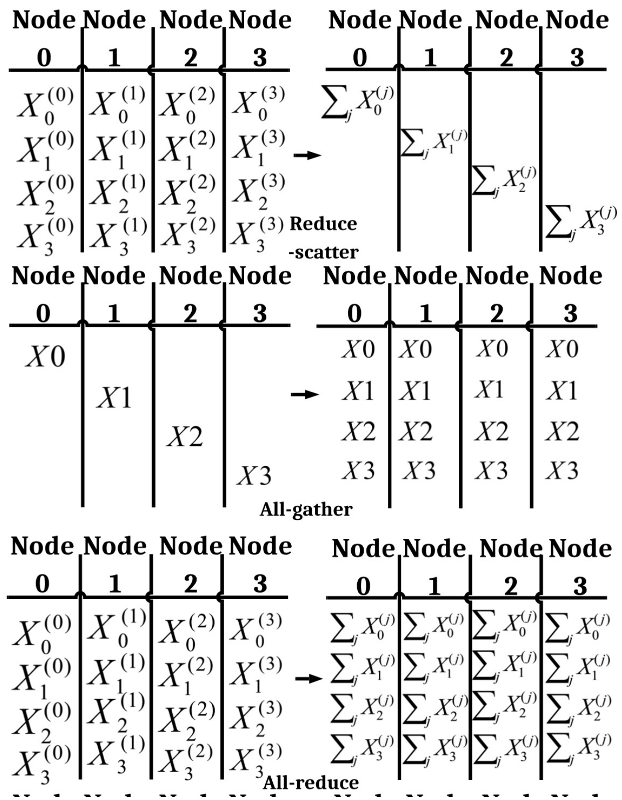




Example: Ring Based All-Reduce

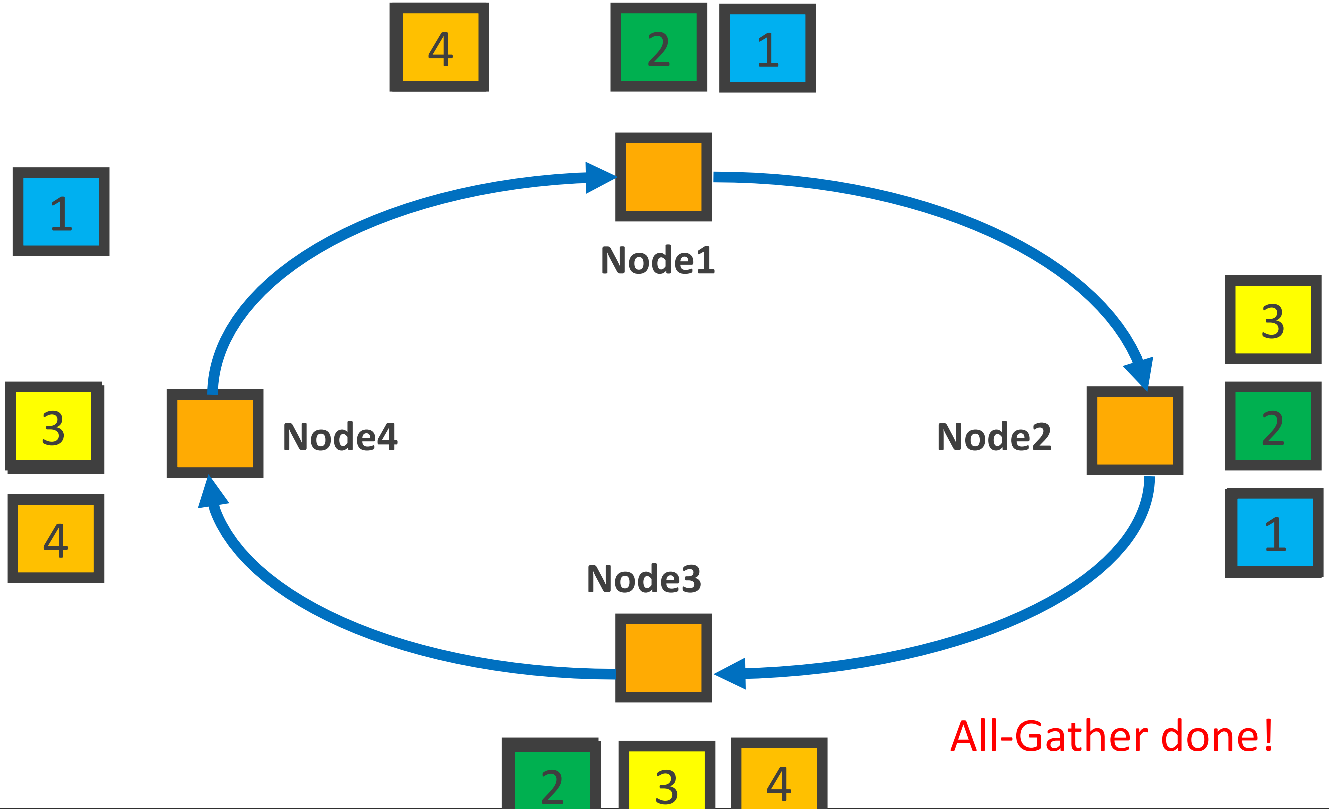
- A ring with N nodes partitions data to N messages
- Collective Communication Flow:

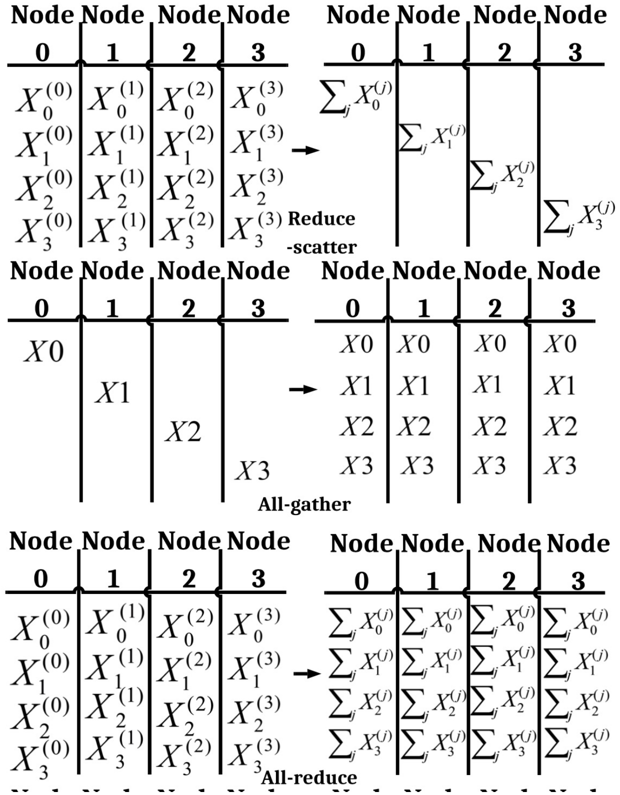




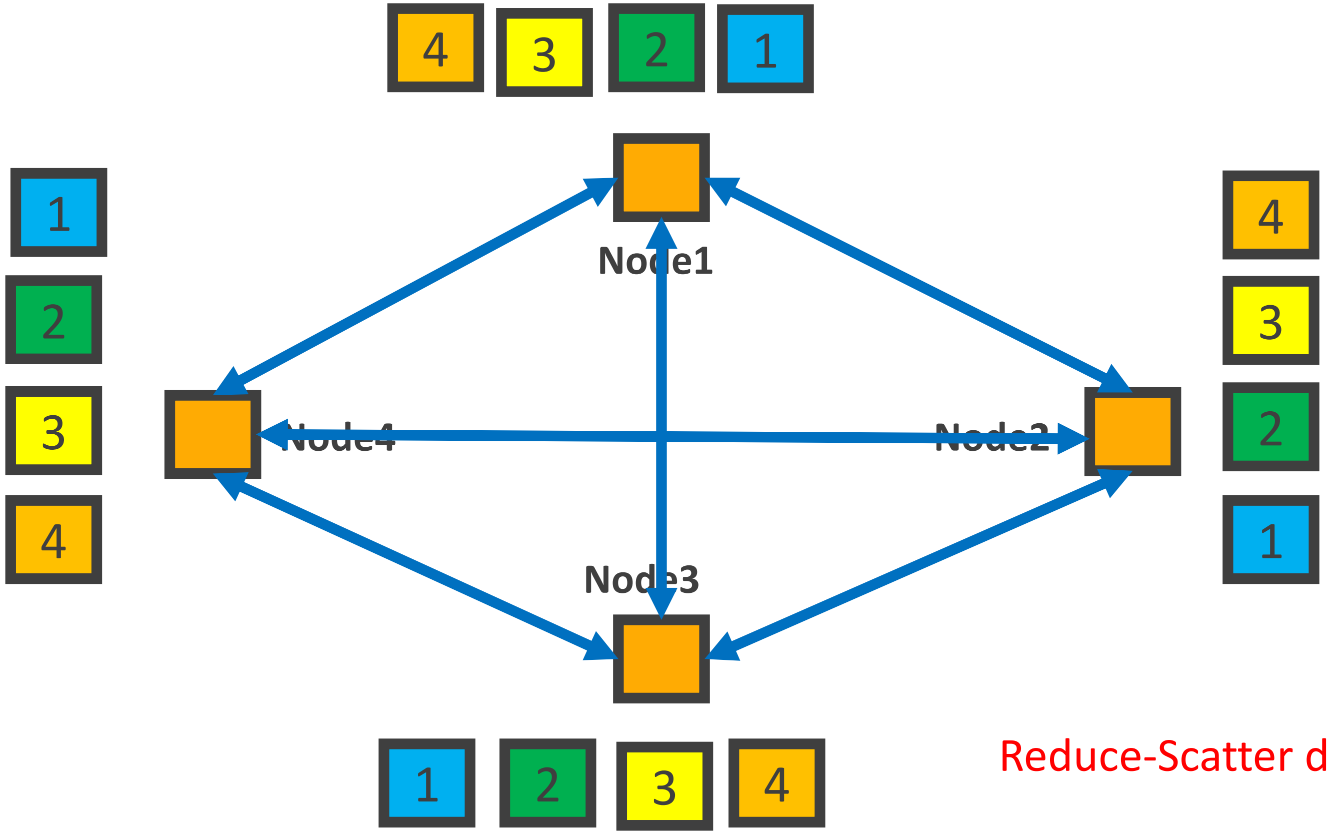
Example: Ring Based All-Reduce

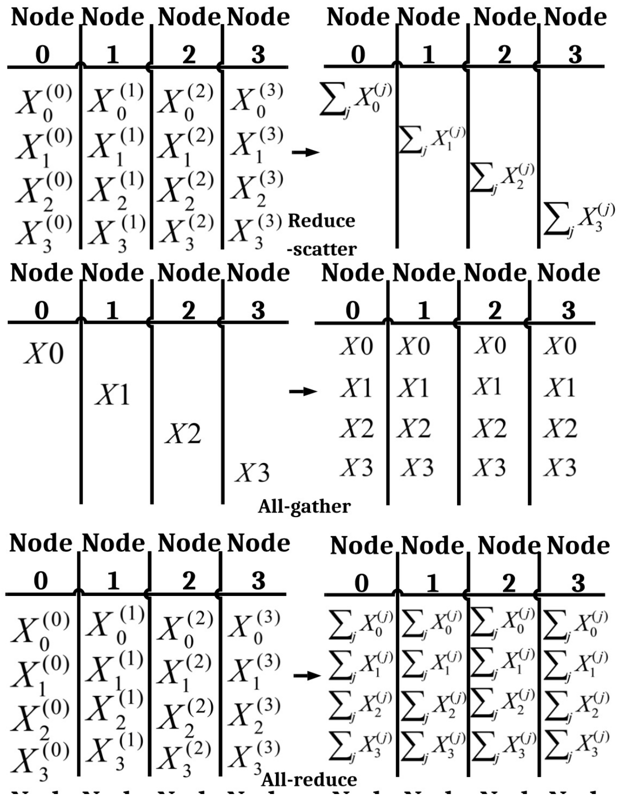
- A ring with N nodes partitions data to N messages
- Collective Communication Flow:





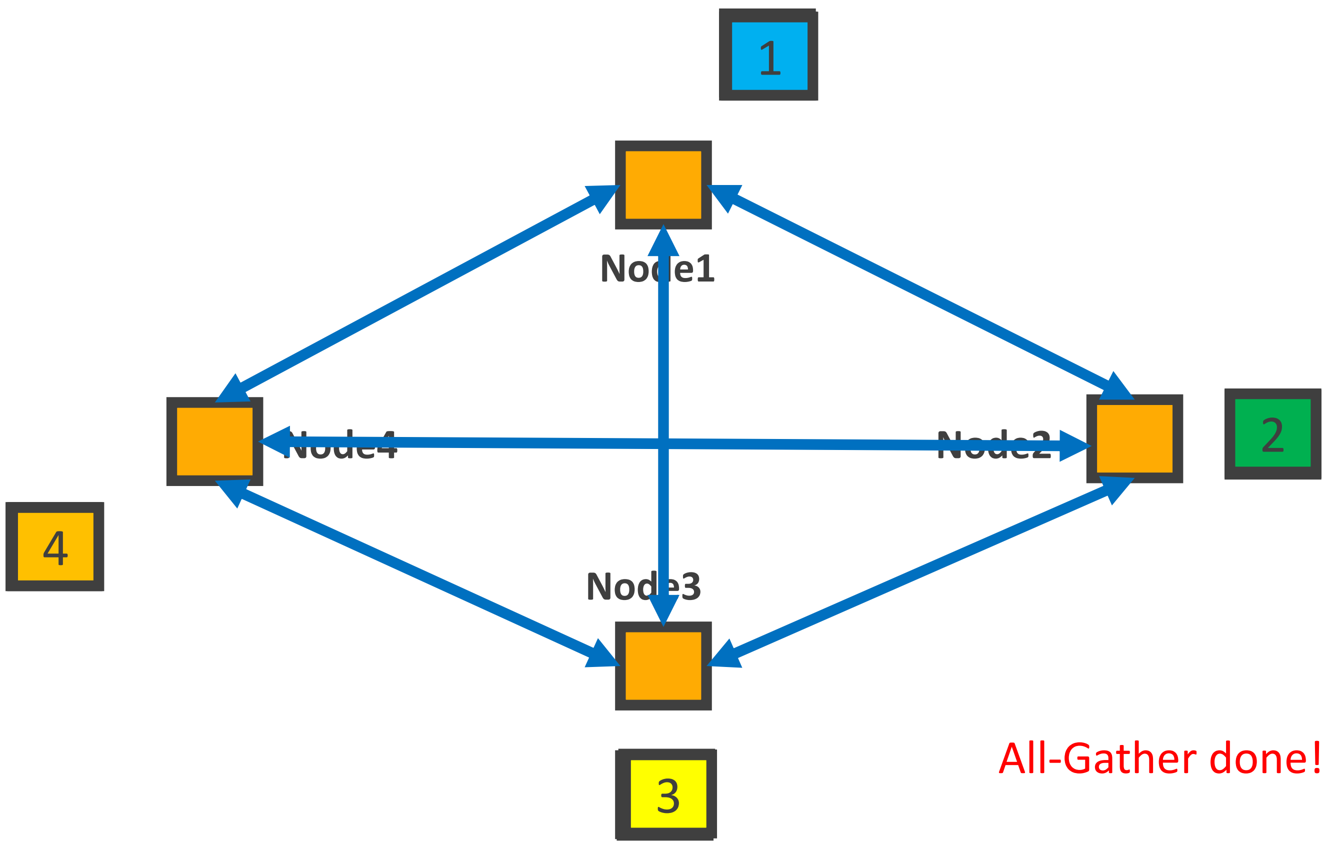
Example: Direct All-Reduce

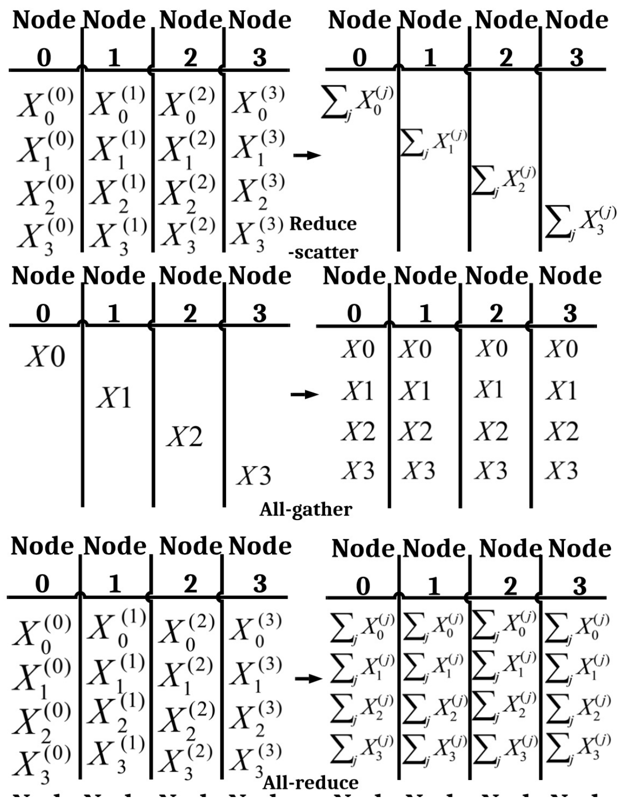




Reduce-Scatter done!

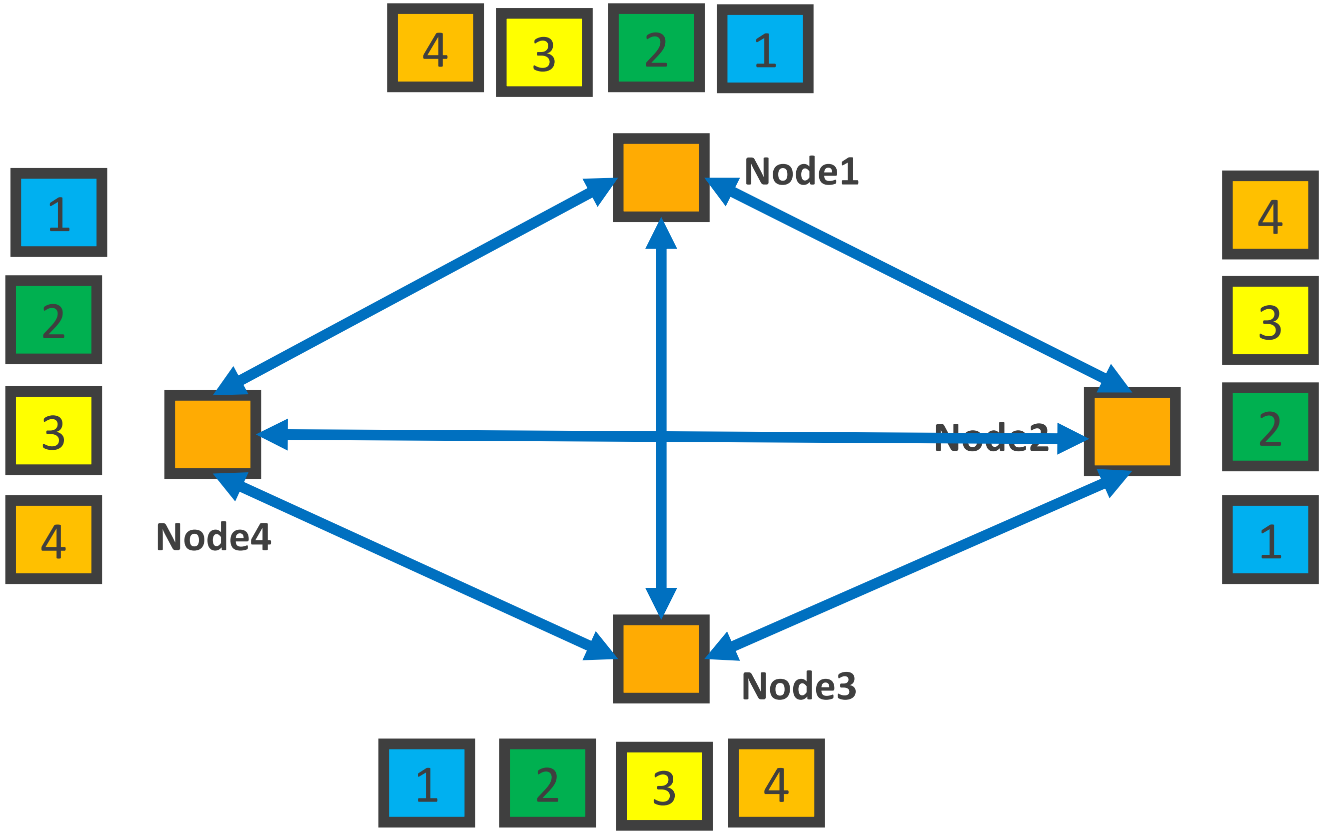
Example: Direct All-Reduce

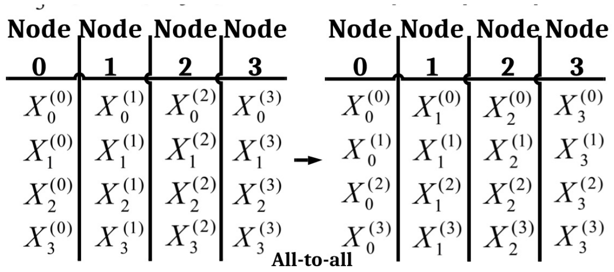




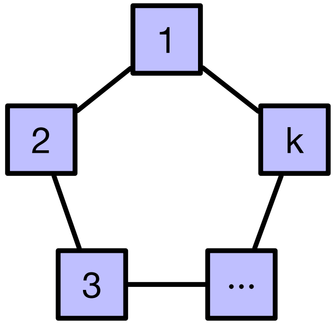
All-Gather done!

Example: All-to-All

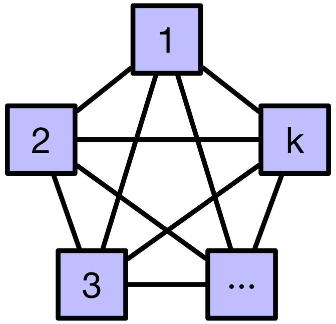




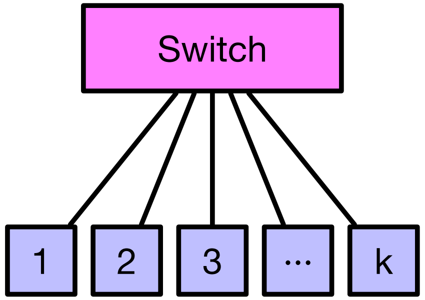
Topology-aware Collectives



(a) Ring(k)

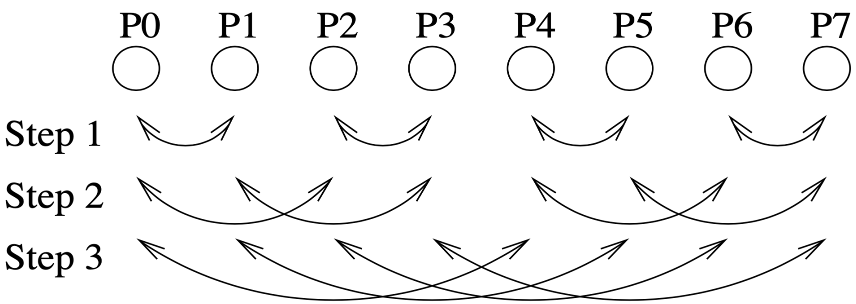


(b) FullyConnected(k)

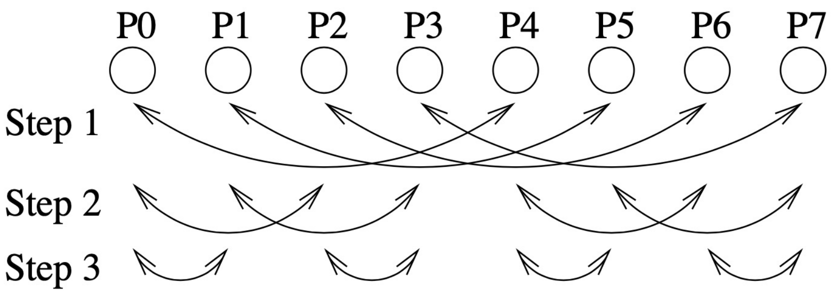


(b) Switch(k)

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



a) Reduce-Scatter phases



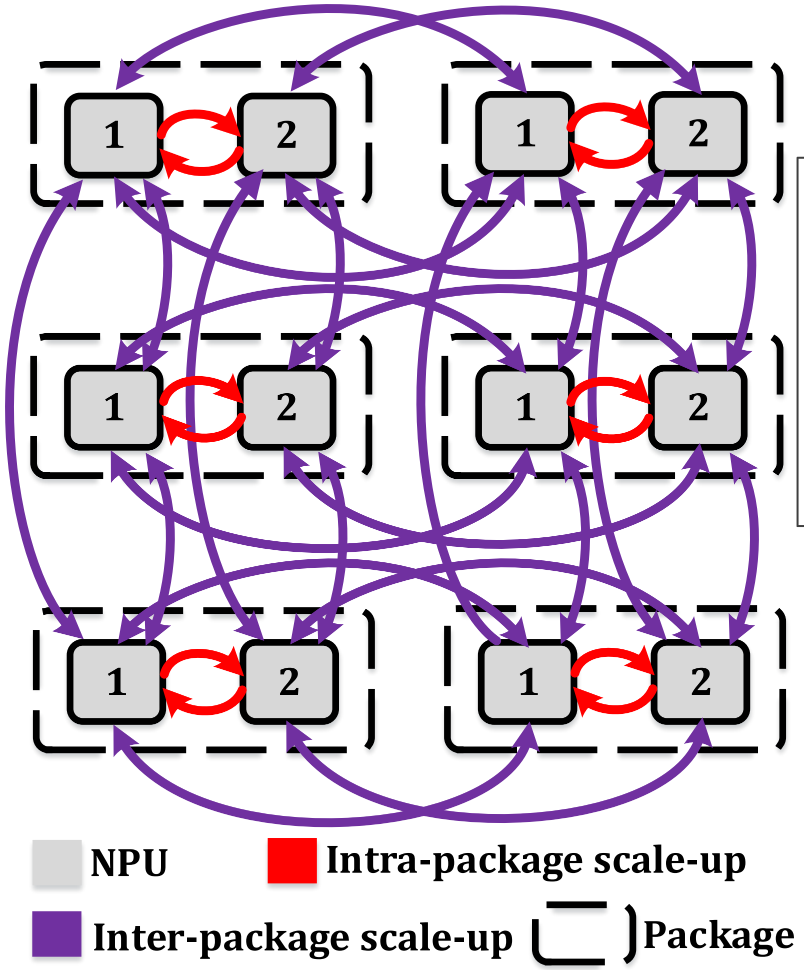
b) All-gather phases

HalvingDoubling All-Reduce

Collectives on Sophisticated Training Platforms

Torus 3D

Similar to Google TPU

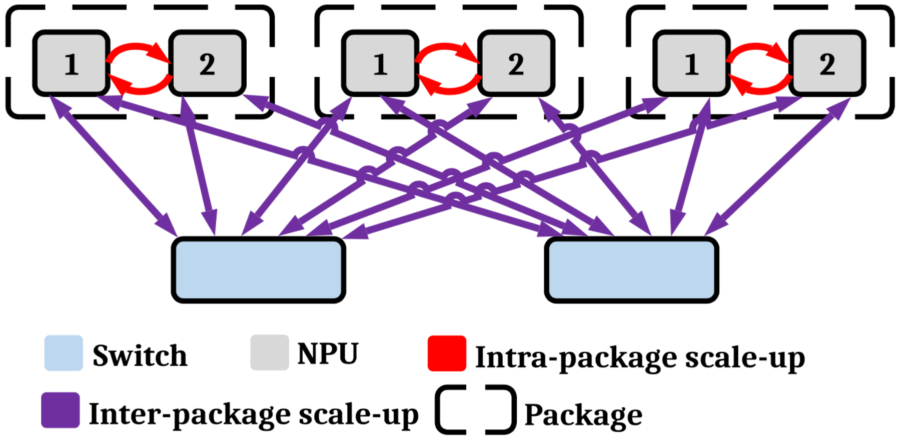


Hierarchical all-reduce:

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

All-To-All

Similar to NVIDIA DGX2



Hierarchical all-reduce:

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

Heterogeneous Bandwidth

Multi-phase Collectives

Distributed Training Stack

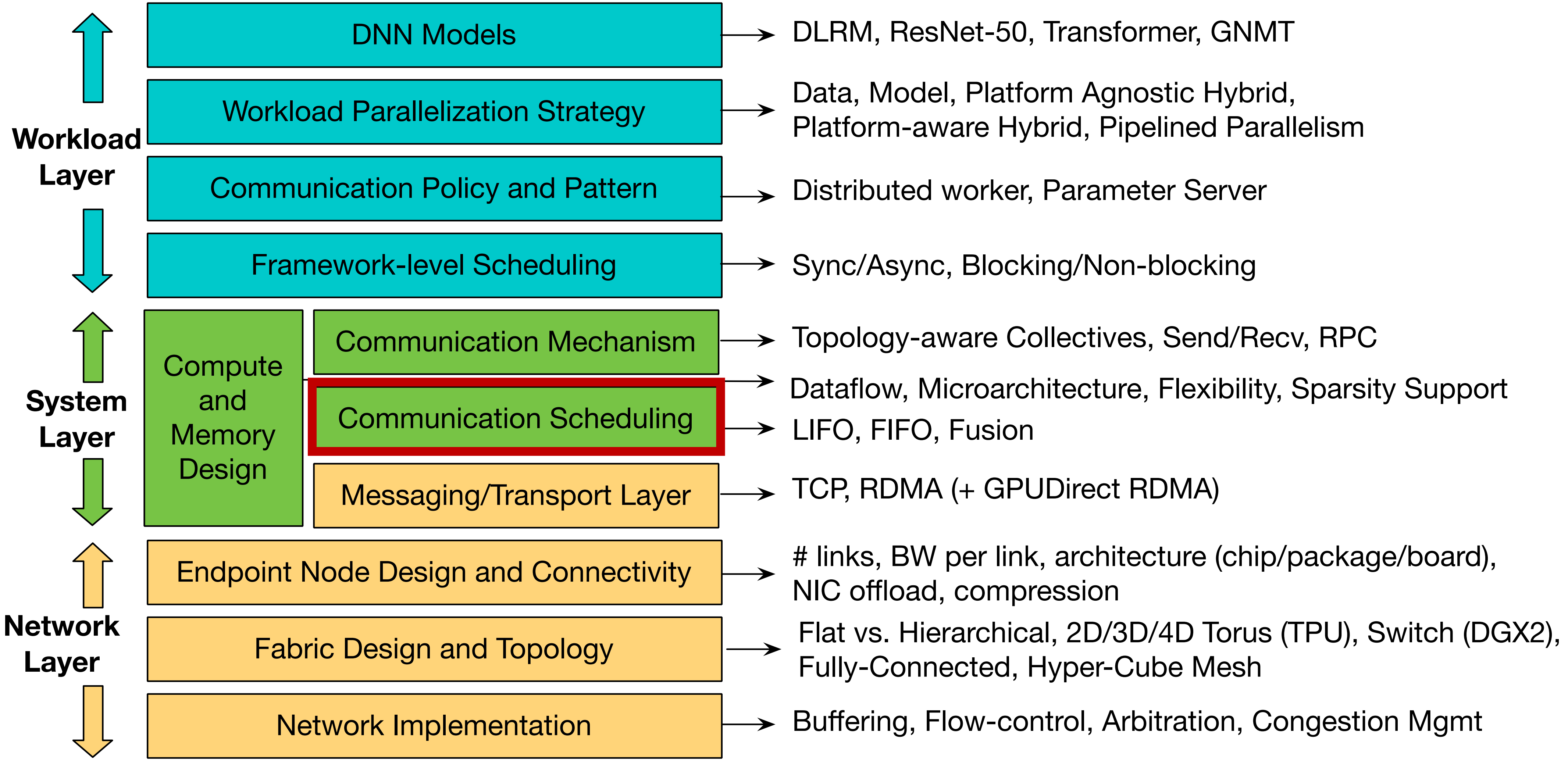
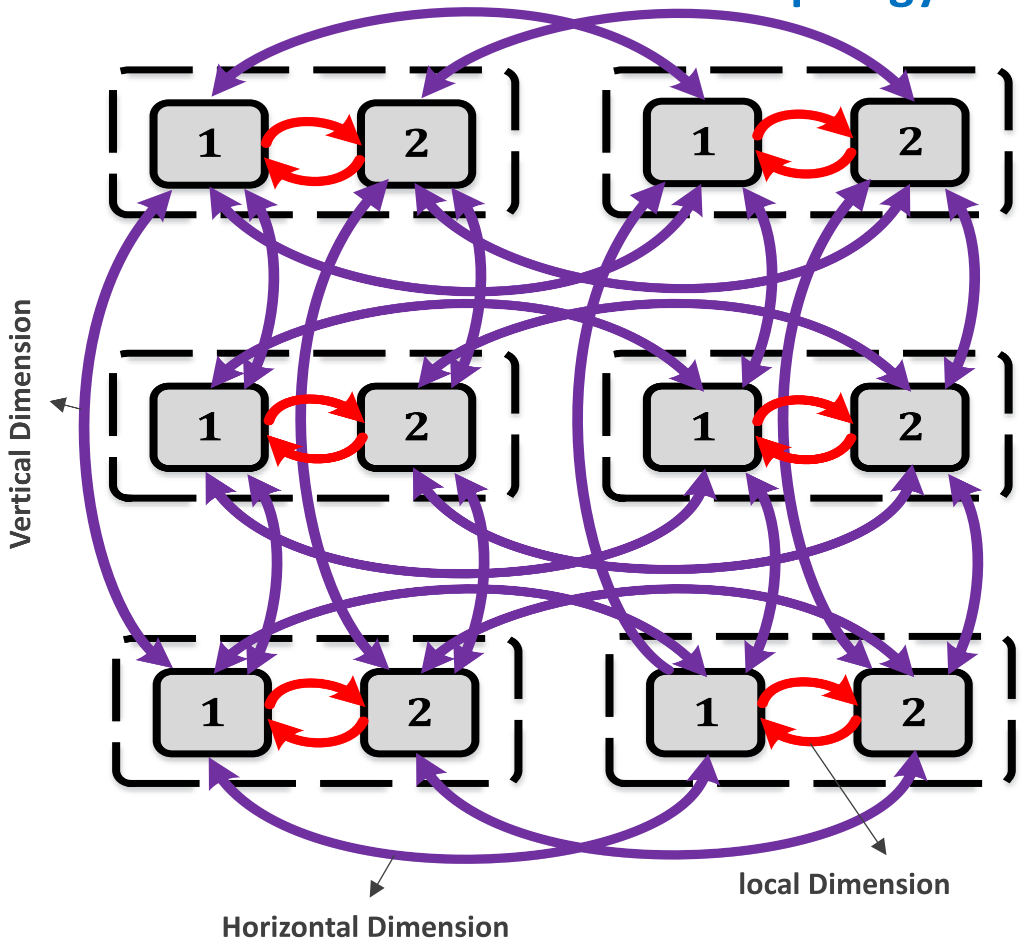


Figure Courtesy: Srinivas Sridharan (Facebook)

Baseline All-Reduce on the Hierarchical Topologies

3D Torus – Hierarchical Topology



Data chunks

start

end

Step:

6 5 4 3 2 1

All-reduce pipeline on the hierarchical topologies:

Suppose the chunk size per NPU is initially 12MB

Chunk size in the initial of the phase:

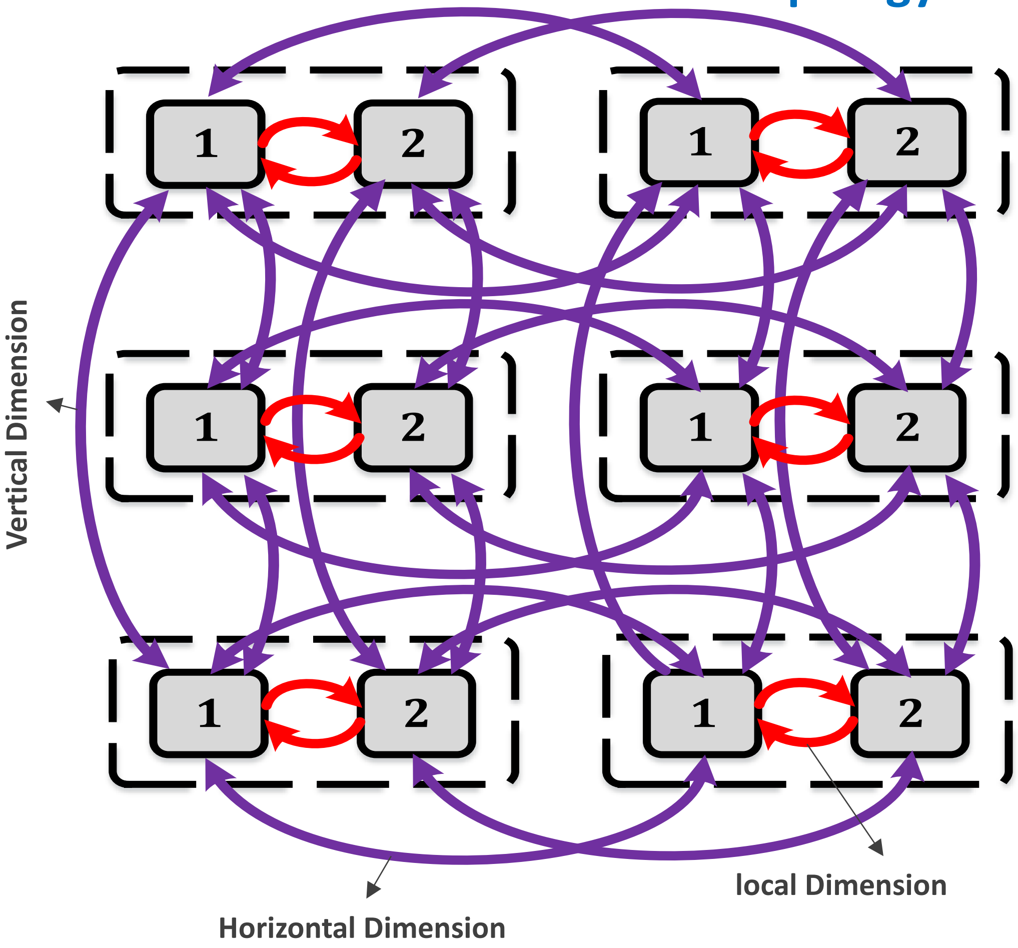
1.	Reduce-scatter on the local dimension	12MB
2.	Reduce-scatter on the horizontal dimension	6MB
3.	Reduce-scatter on the vertical dimension	3MB
4.	All-gather on the vertical dimension	1MB
5.	All-gather on the horizontal dimension	3MB
6.	All-gather on the local dimension	6MB

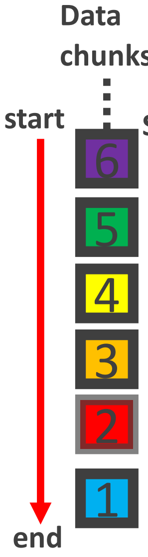
End: 12MB

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



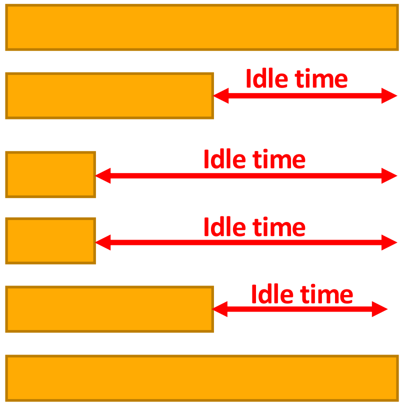


All-reduce pipeline on the hierarchical topologies:

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

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

Distributed Training Stack

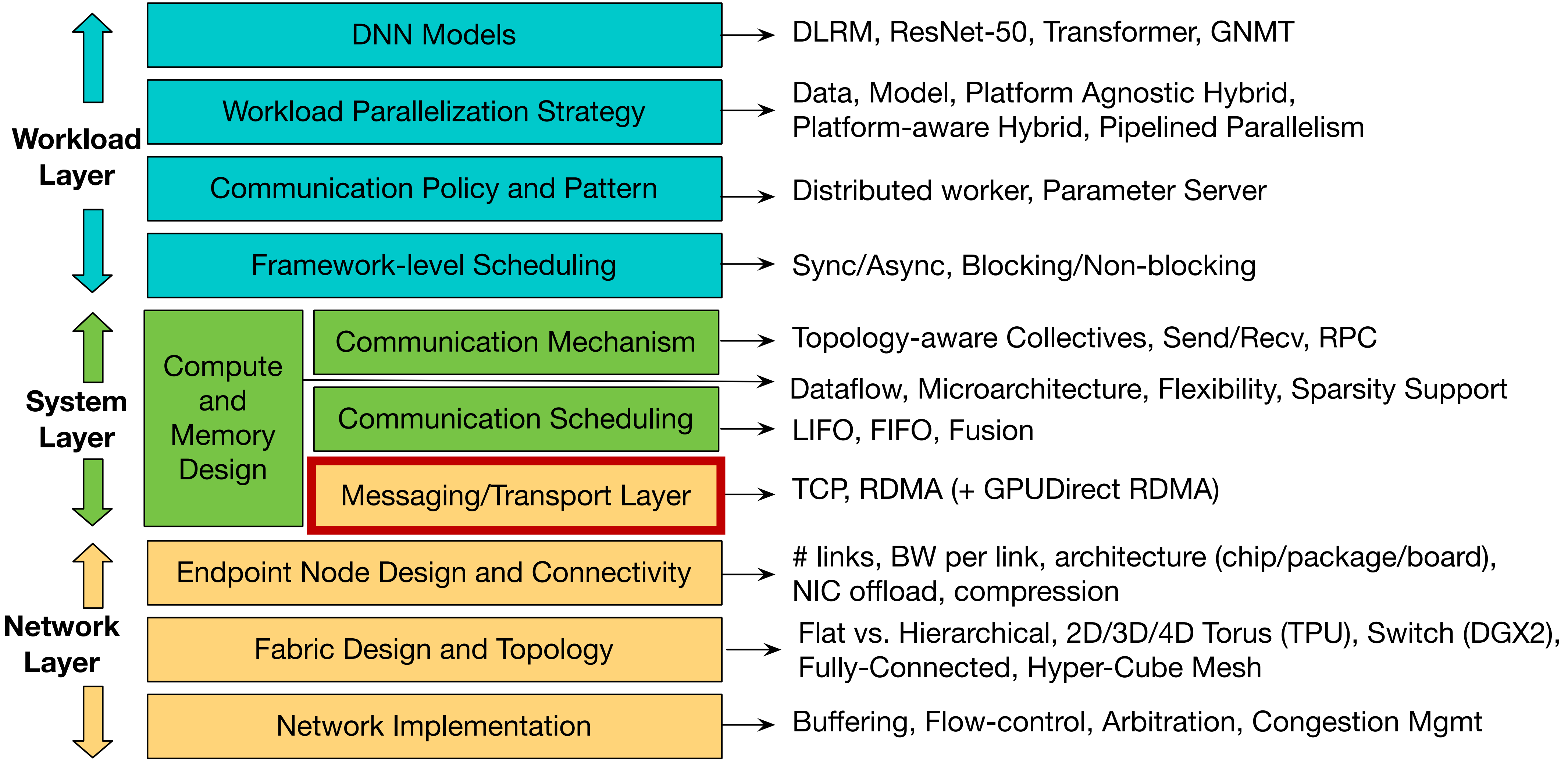
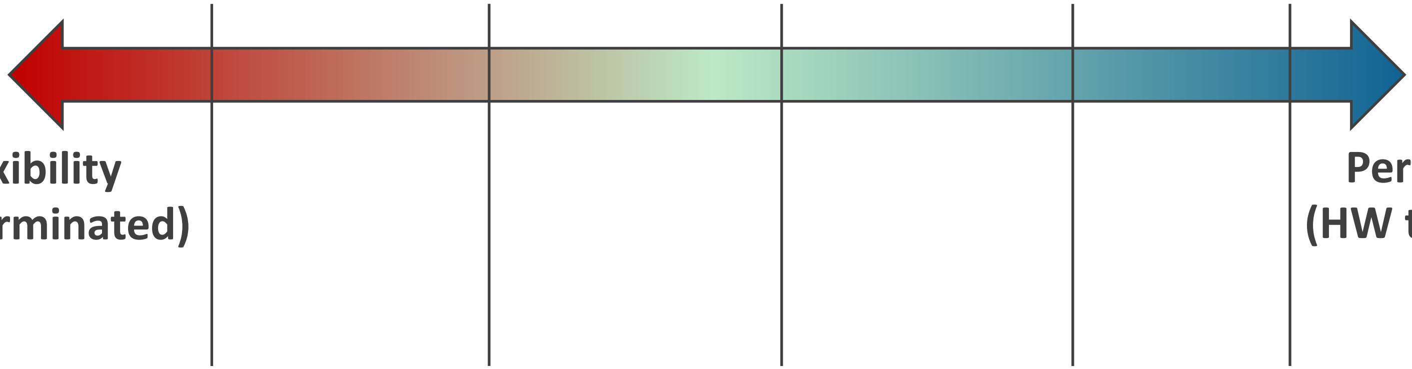


Figure Courtesy: Srinivas Sridharan (Facebook)

Transport Protocols



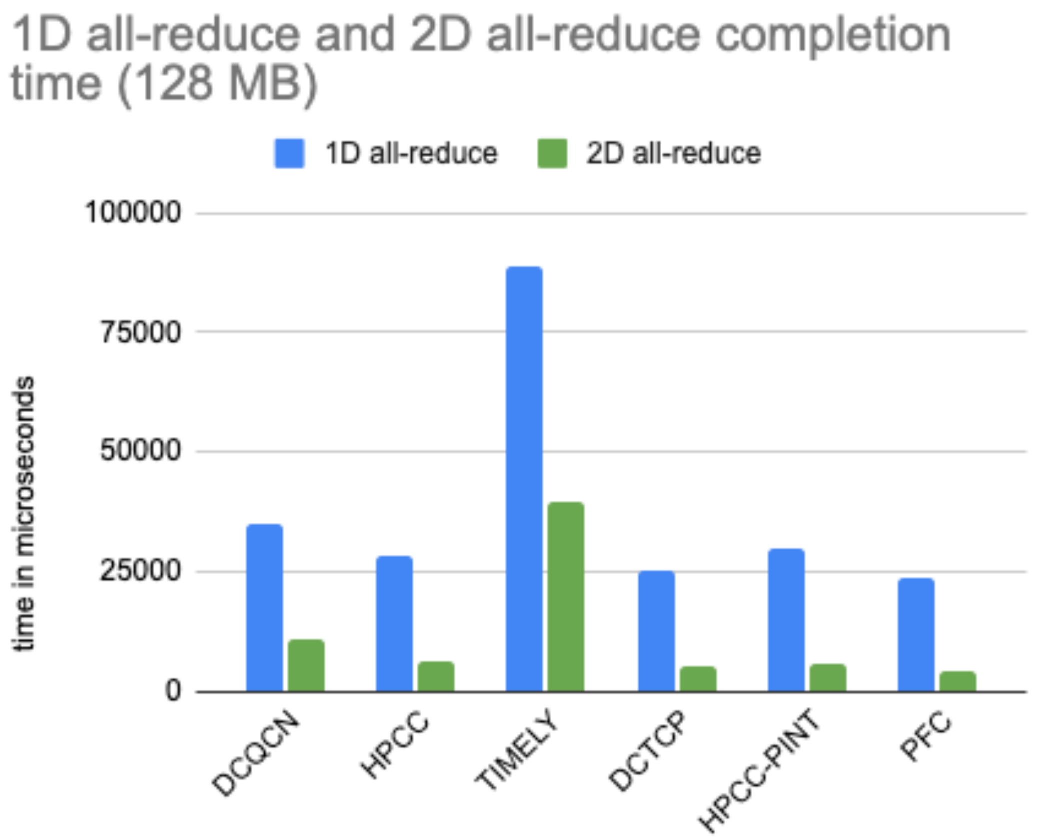
Flexibility (SW terminated) — **Performance (HW terminated)**

	TCP	1RMA*	RDMA	GPUdirect RDMA	GPUdirect RDMA Async
Network Policy - congestion policy	CPU	CPU	NIC	NIC	NIC
Protocol Execution - packetization -..	CPU	NIC	NIC	NIC	NIC
Control Plane	CPU	CPU	CPU	CPU	GPU
Data Plane	Main Memory	Main Memory	Main Memory	GPU Memory	GPU Memory

**A Singhvi et al.,"1rma: Re-envisioning remote memory access for multi-tenant datacenters, SIGCOMM 2020*

Congestion Control

- Enforcement mechanism
  - Window-based vs Rate-based
- What metrics to use?
  - Network telemetry vs RTT



Research Questions:

- Impact on training time
- What is the best policy when having irregular parallelization strategy

*T Khan, S Rashidi, S Sridharan, P Shurpali, A Akella and T Krishna , "**Impact of RoCE Congestion Control Policies on Distributed Training of DNNs**" In Proceedings of the 29th International Symposium on High-Performance Interconnects **(HotI)**, Aug 2022.*

Distributed Training Stack

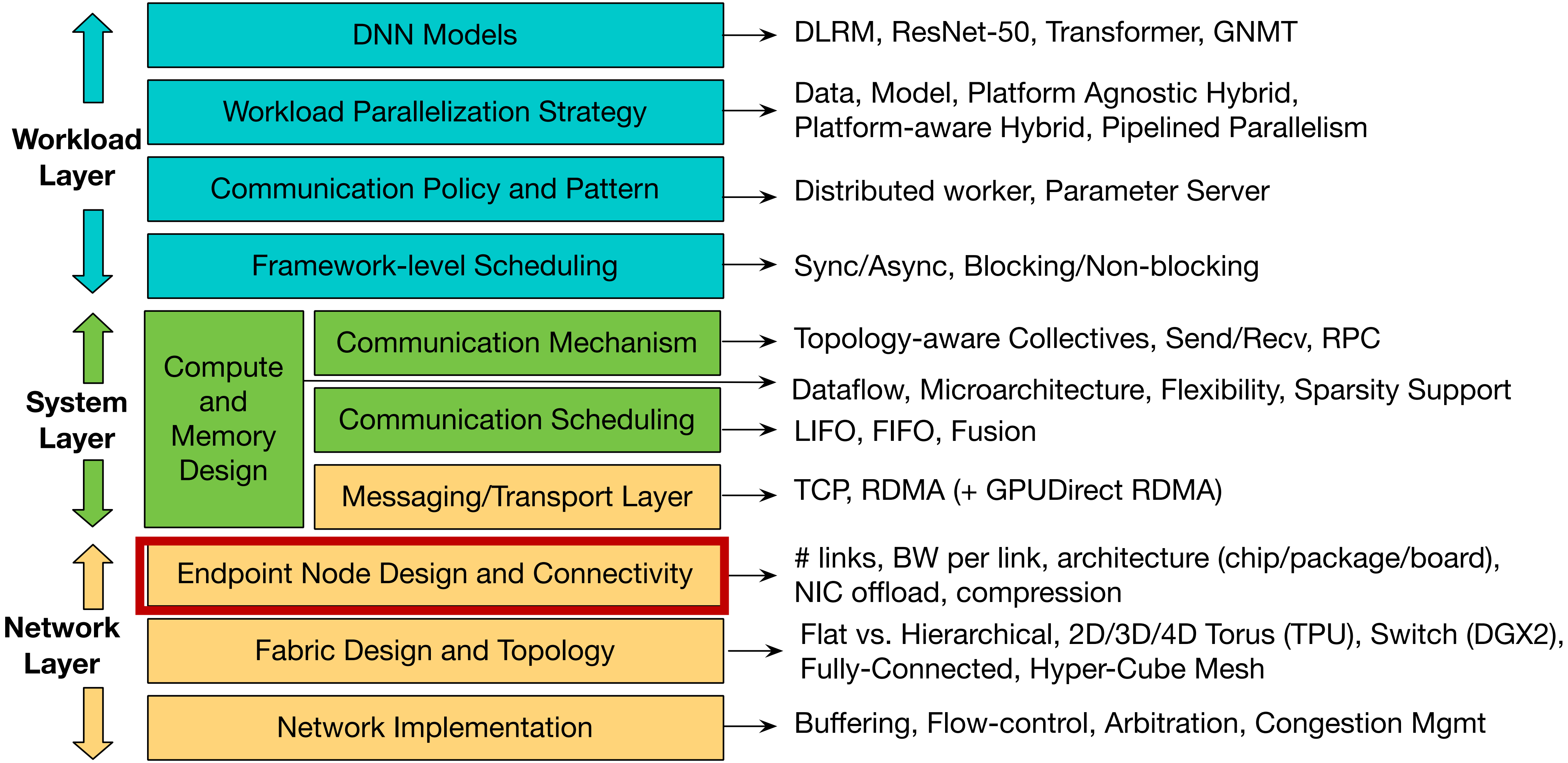
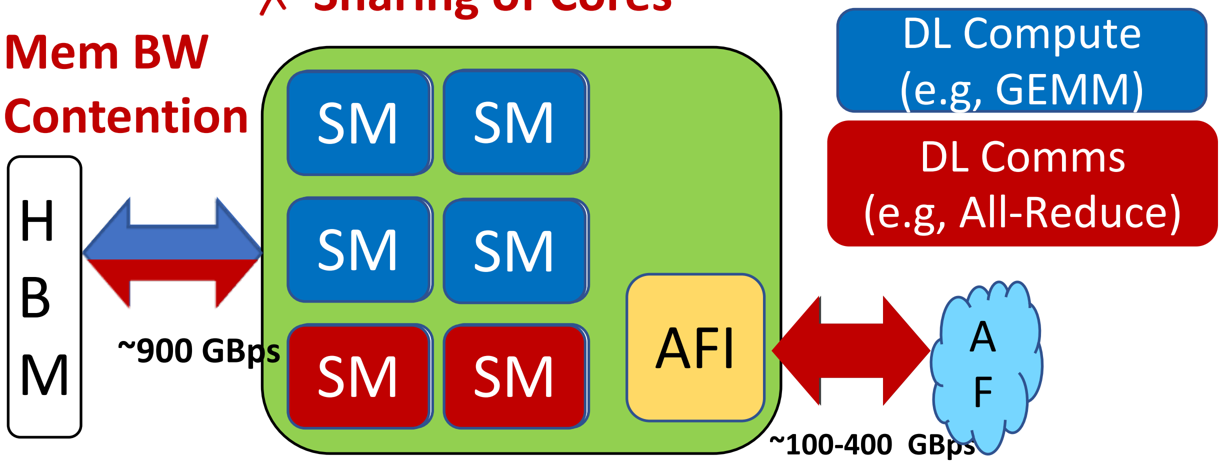


Figure Courtesy: Srinivas Sridharan (Facebook)

Resource Contention at End-point

✗ **Sharing of Cores**

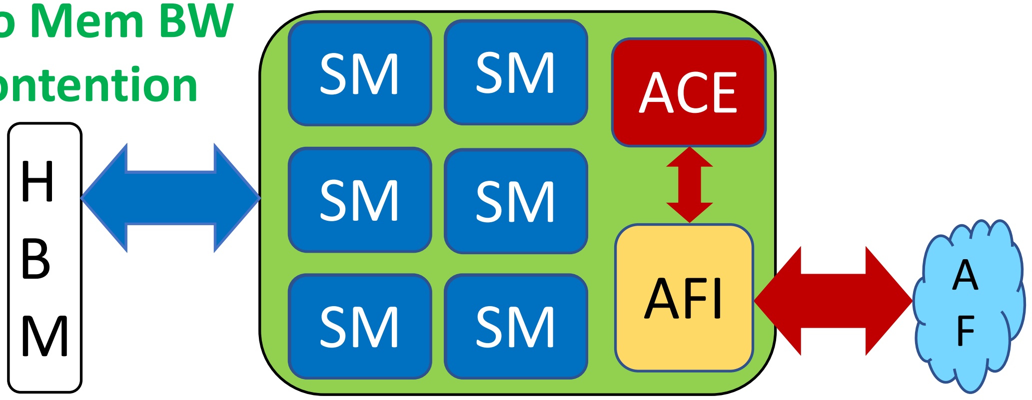
✗ **Mem BW Contention**



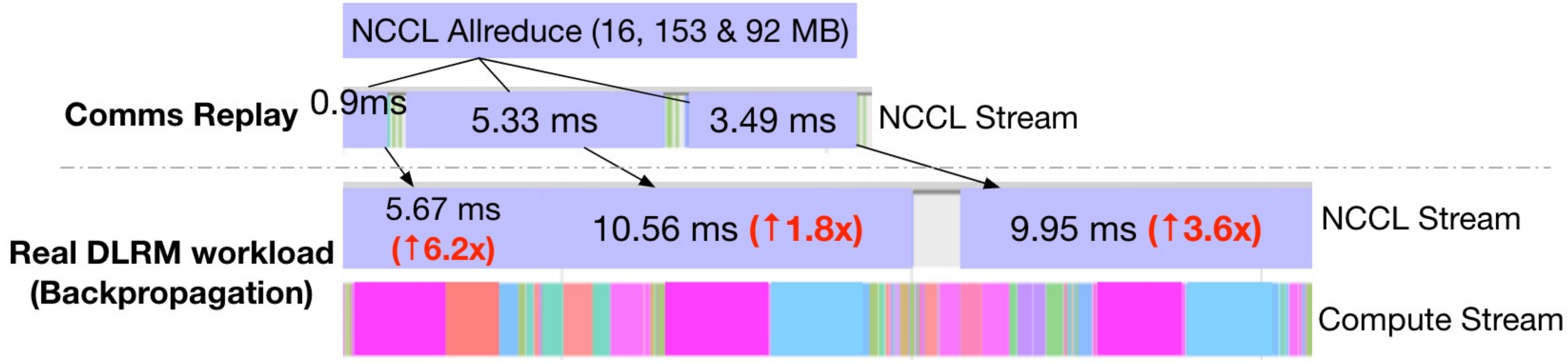
Endpoint in Baseline

✓ **Separate accel. for comms**

✓ **No Mem BW Contention**



Endpoint with Accelerator Collectives Engine (ACE)



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

Alternate approach: offload to switch (e.g., ISCA 2020)

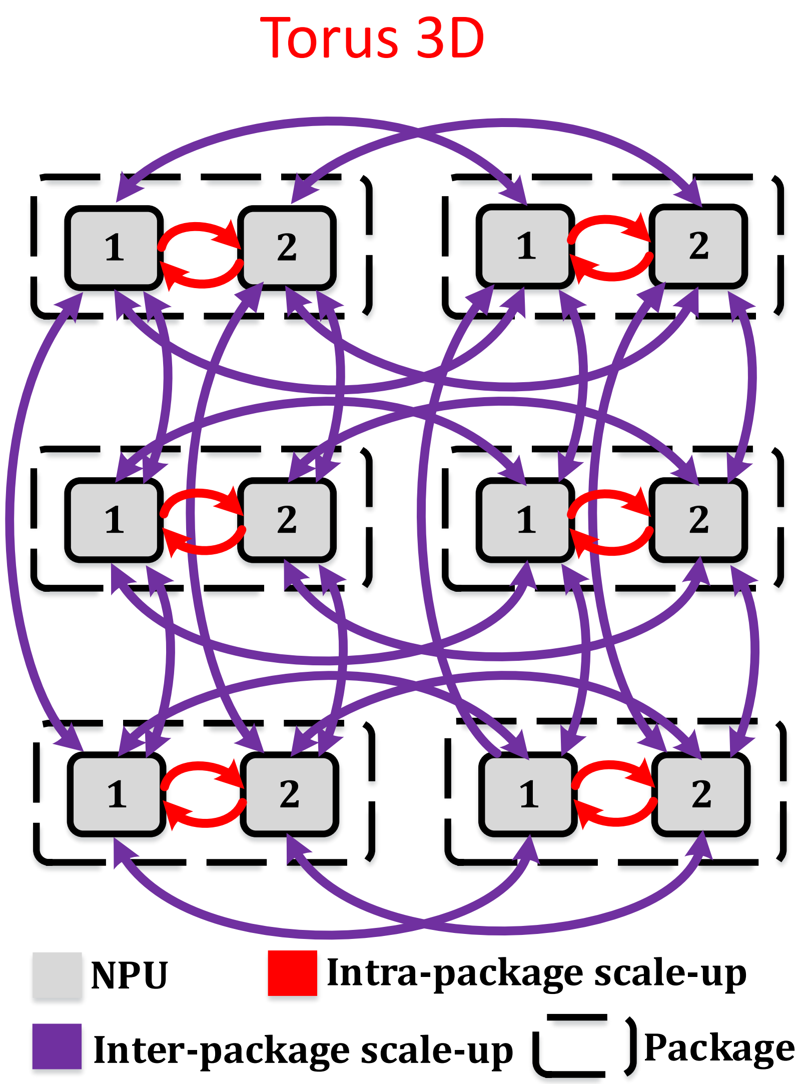
*S. Rashidi et al., "**Enabling Compute-Communication Overlap in Distributed Deep Learning Training Platforms**". ISCA 2021*

Distributed Training Stack

Layer	Component	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 (Facebook)

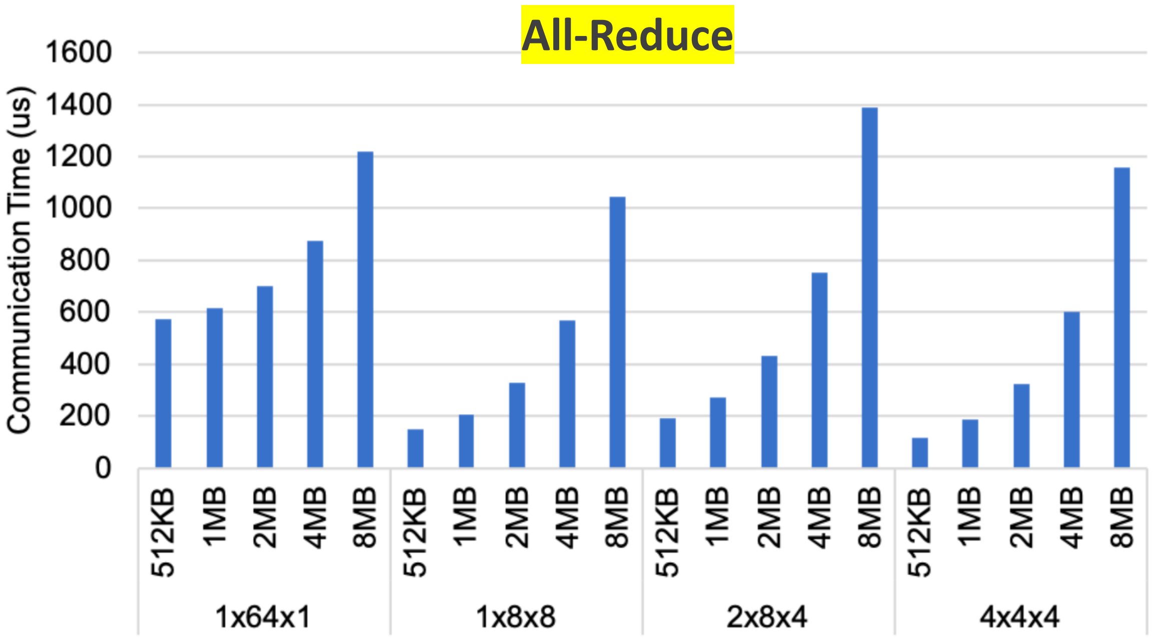
Target System



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

Impact of 1D/2D/3D Torus

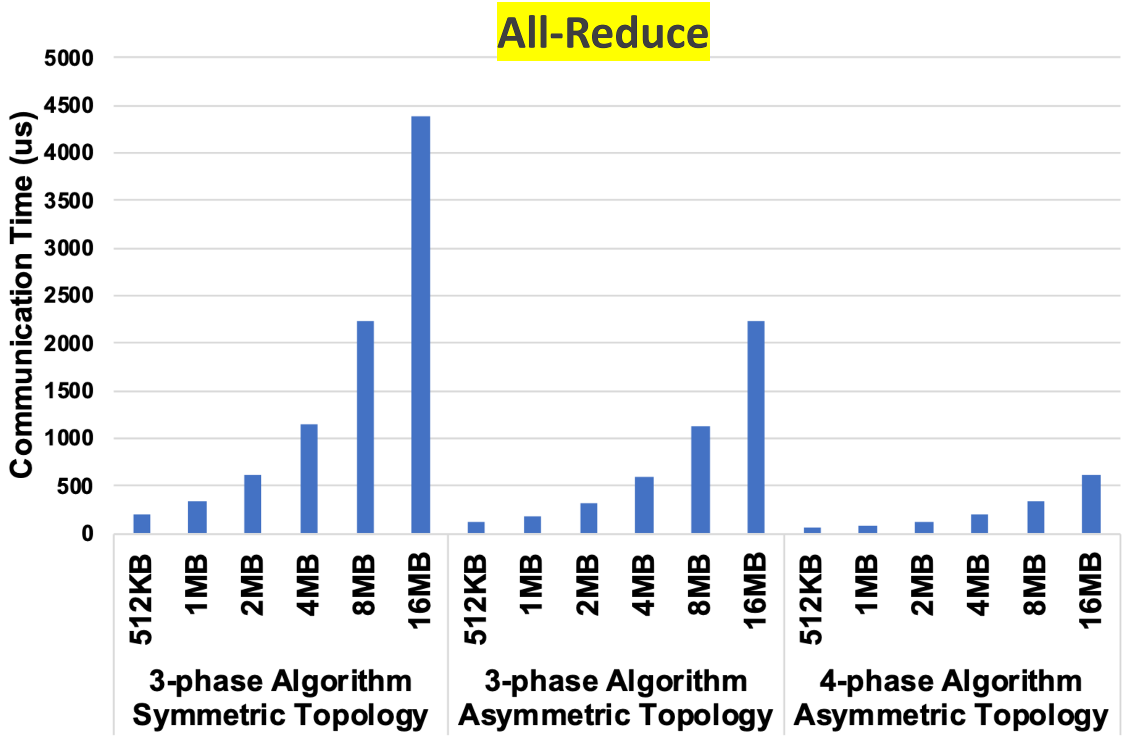
- Adding a dimension decreases the number of steps per collective.
  - For example, going from 1X64X1 to 1X8X8.
- Adding a dimension might increase amount of data each node sends out (depends on the algorithm).
  - For example, going from 1X8X8 to 2X8X4.
- Hence, choosing a topology is a tradeoff between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

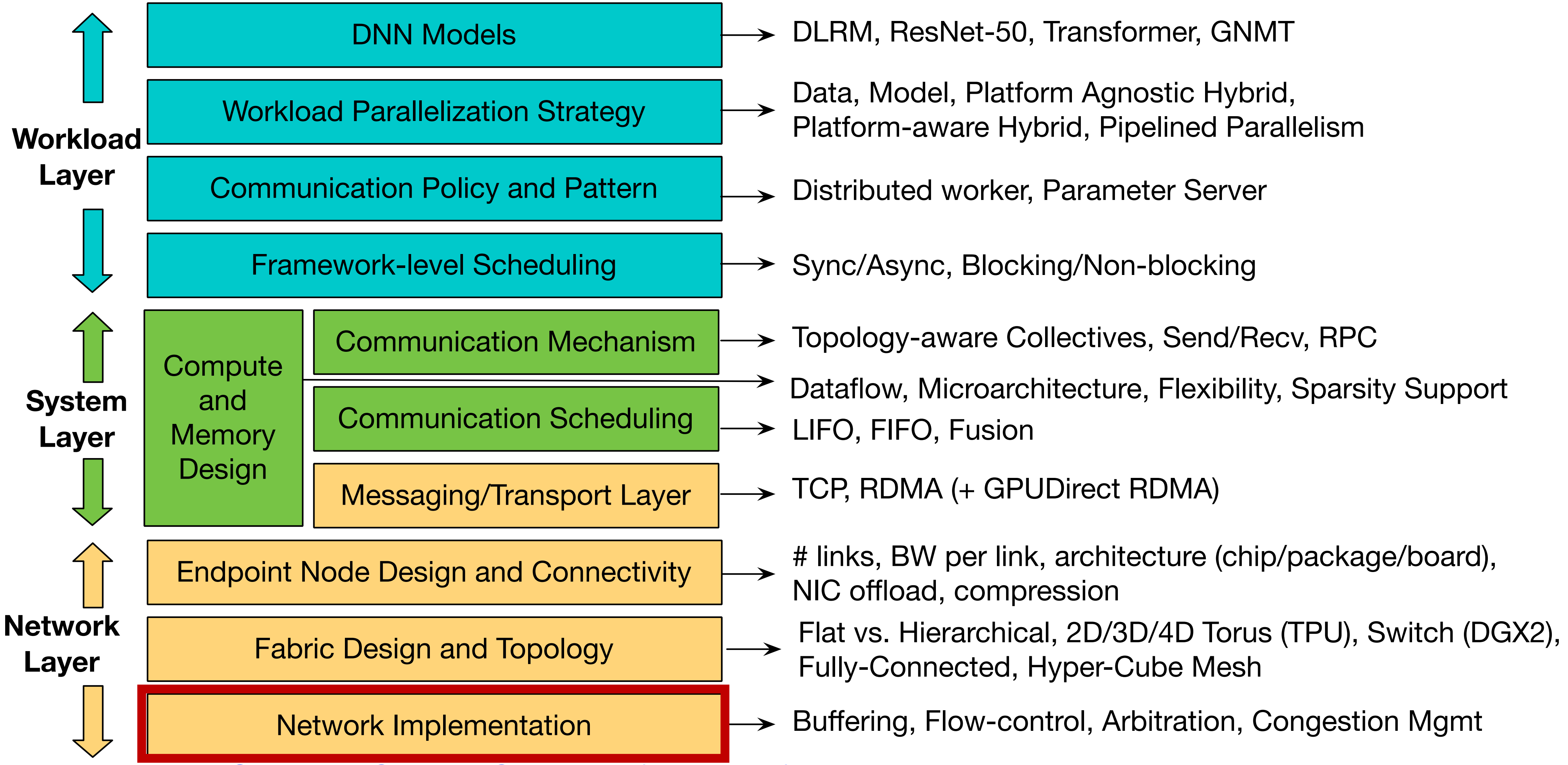
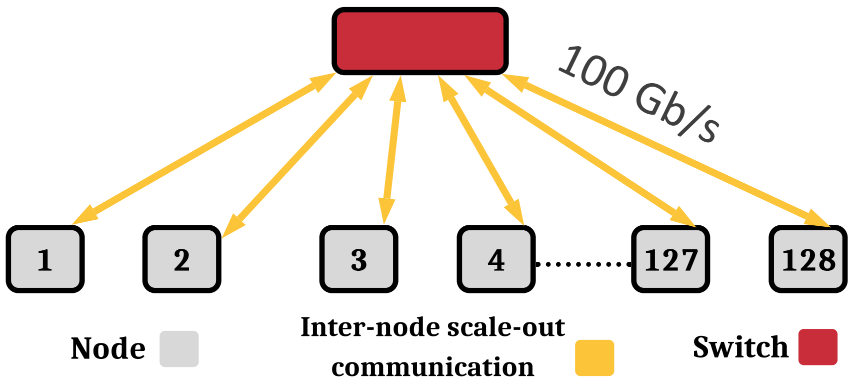
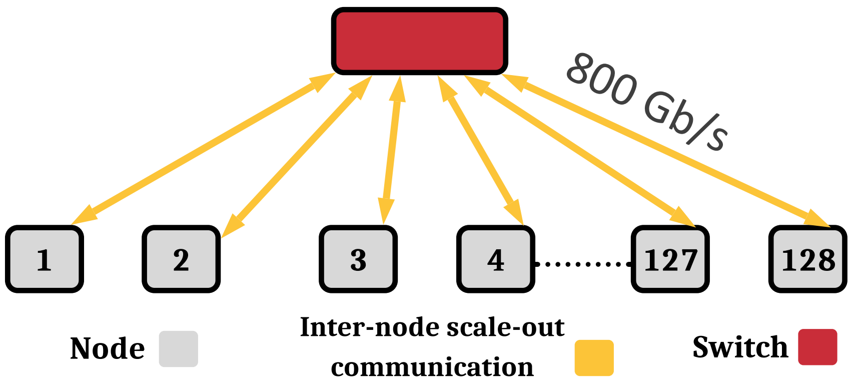


Figure Courtesy: Srinivas Sridharan (Facebook)

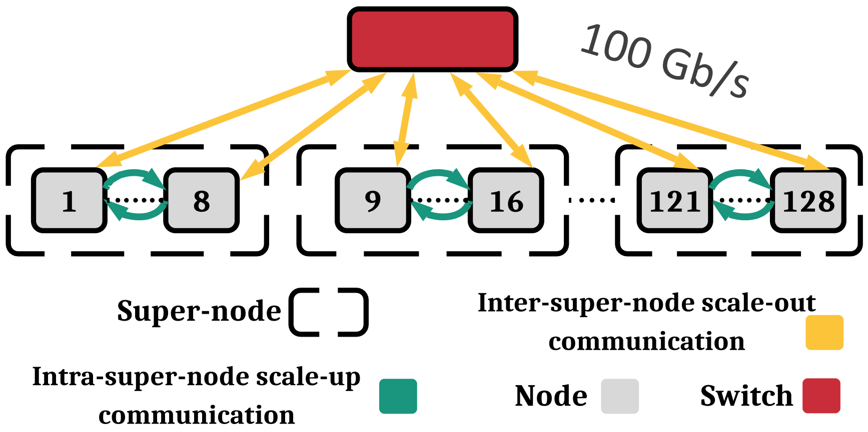
Target Systems



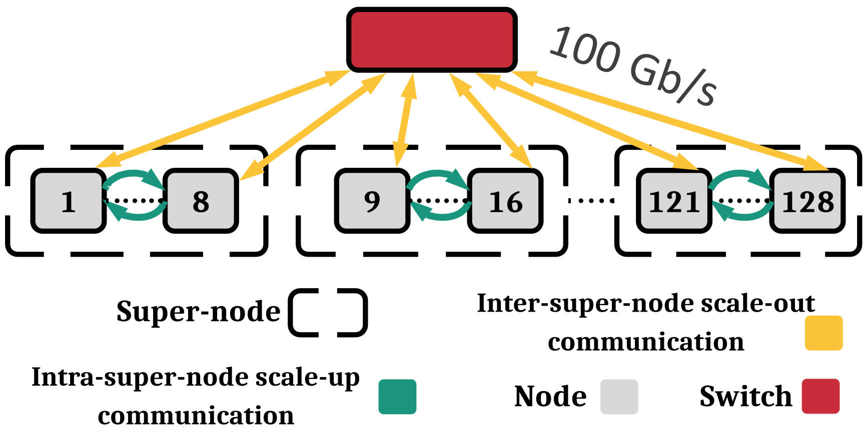
Flat100G
1-phase all-reduce



Flat800G
1-phase all-reduce



Hier
2-phase all-reduce

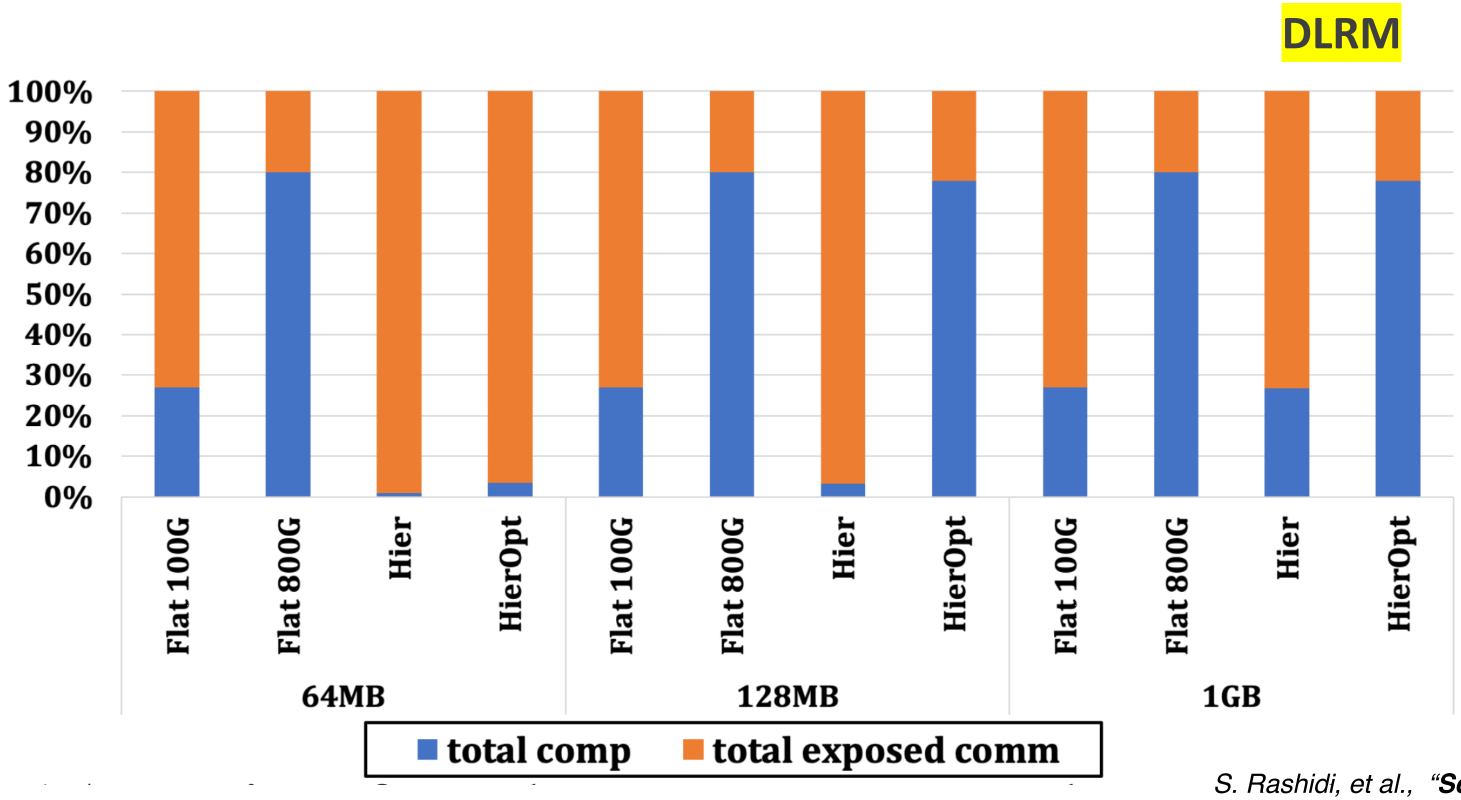


HierOpt
3-phase all-reduce

Effect of Size of Switch Buffer

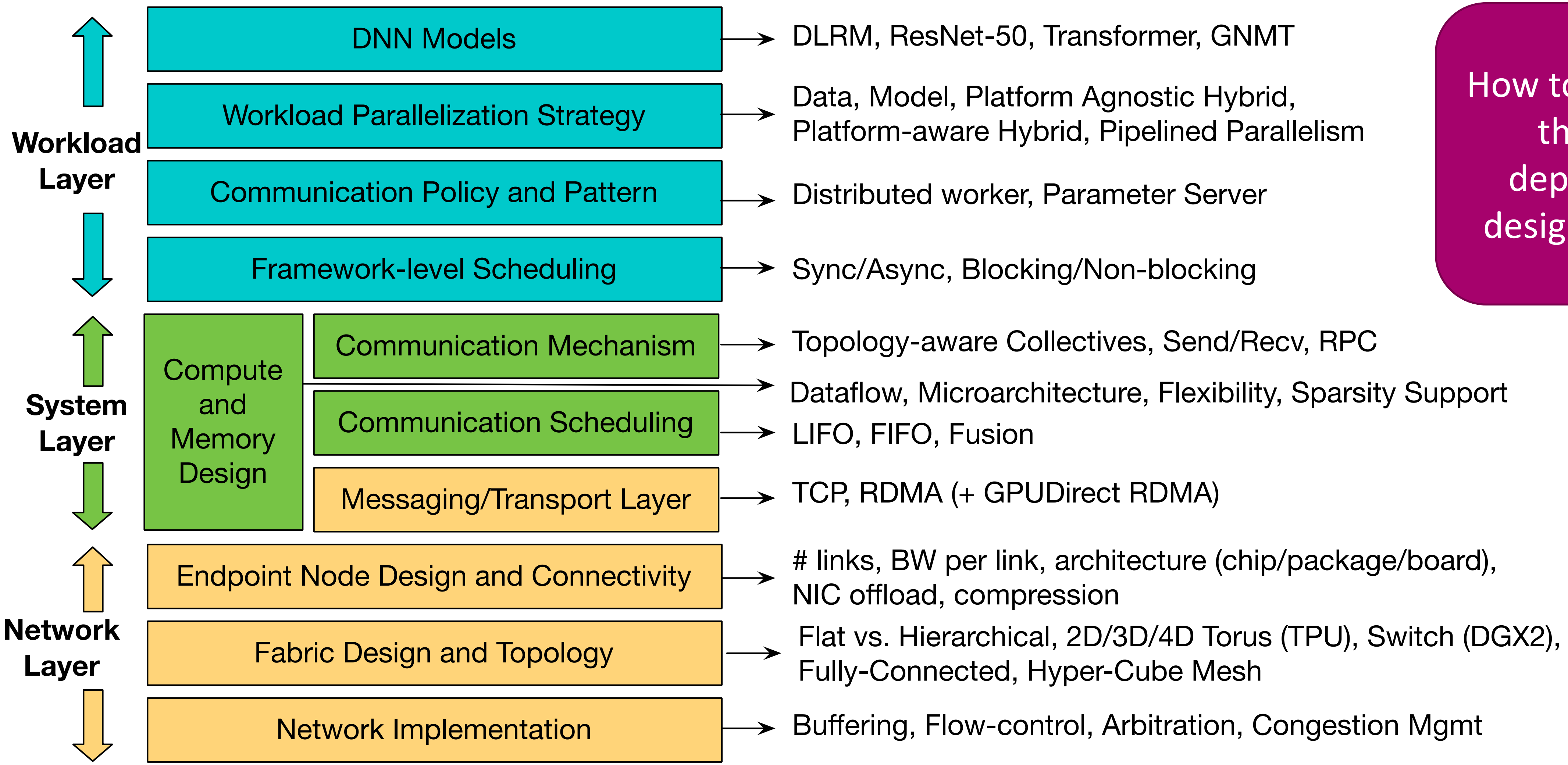
Observations:

- 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)

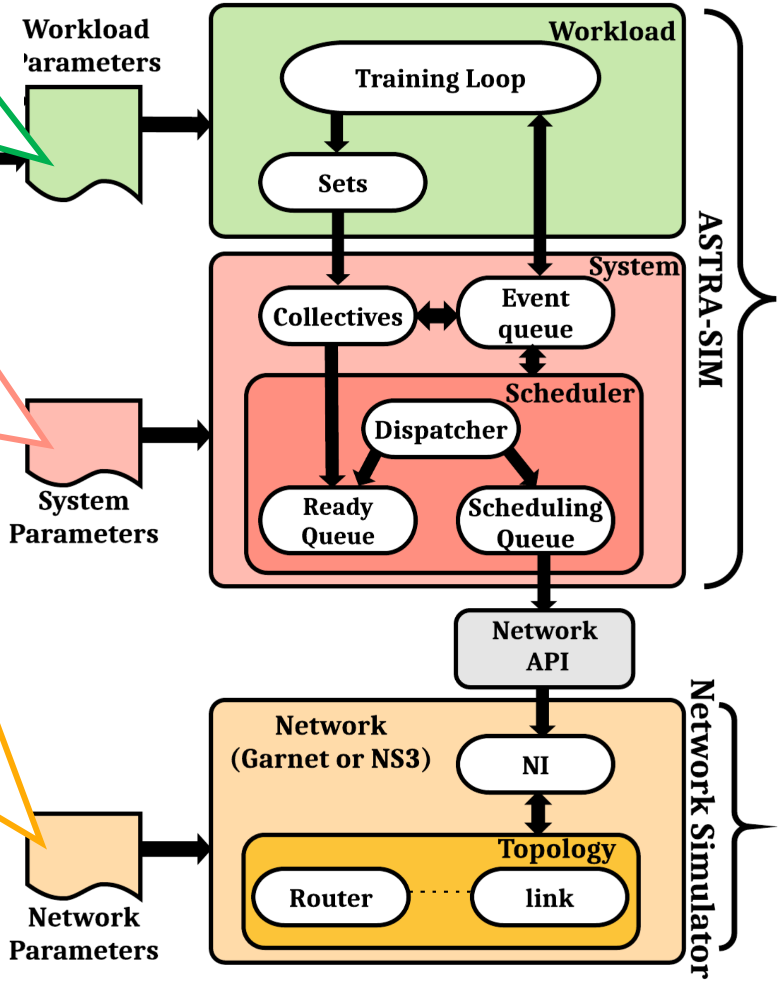
Introducing ASTRA-sim

✓ *Released* ➢ *In progress*

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

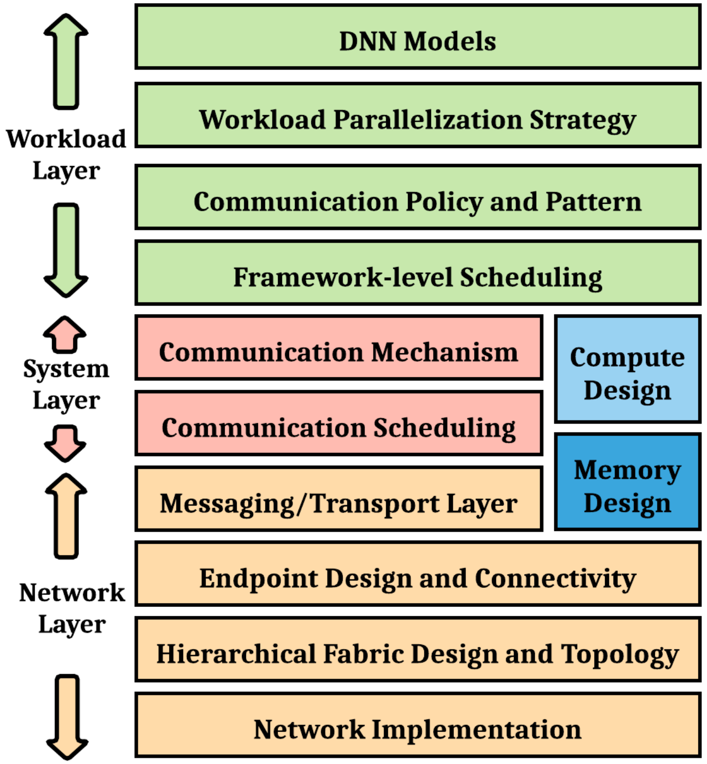
- ✓ Ring based, Tree-based, AlltoAll based, and multi-phase collectives
- ✓ Variety of scheduling policies
- ✓ Compute times fed via offline system measurements or compute simulator

- ✓ 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 (PDT)	Topic	Presenter
1:00 – 2:00	**Introduction to Distributed DL Training**	Tushar Krishna
2:00 – 2:20	**Challenges on Distributed Training Systems**	Srinivas Sridharan
2:20 – 3:30	**Introduction to ASTRA-sim simulator**	Saeed Rashidi
3:30 – 4:00	**Coffee Break**	
4:00 – 4:50	**Hands-on Exercises on Using ASTRA-sim**	William Won and Taekyung Heo
4:50 – 5:00	**Closing Remarks and Future Developments**	Taekyung Heo

Tutorial Website
includes agenda, slides, ASTRA-sim installation instructions (via source + docker image)
https://astra-sim.github.io/tutorials/mlsys-2022

Attention: Tutorial is being recorded