Collective Communications

Arvind Krishnamurthy

Material adapted from slides by Tianqi Chen & Zhihao Jia (CMU), Tushar Krishna & Divya Mahajan (GTech)

COMMUNICATION AMONG TASKS

What are common communication patterns?

Point-to-point communication

- Single sender, single receiver
- Relatively easy to implement efficiently

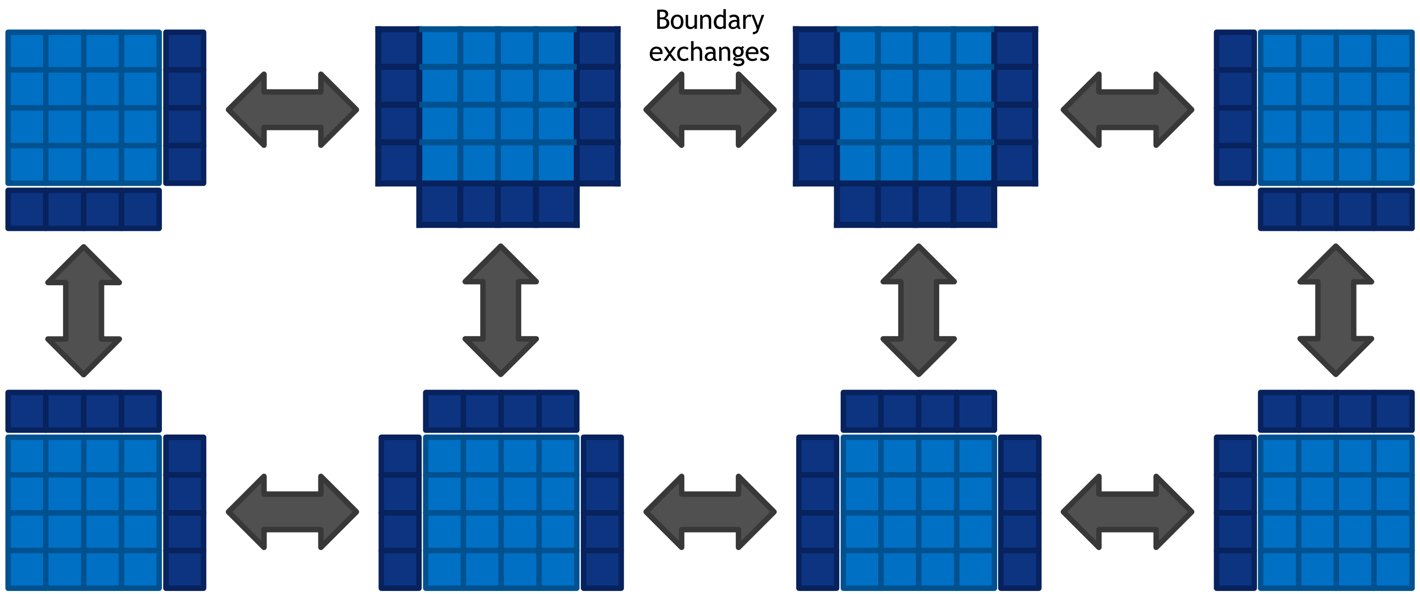
Collective communication

- Multiple senders and/or receivers
- Patterns include broadcast, scatter, gather, reduce, all-to-all, …
- Difficult to implement efficiently

POINT-TO-POINT COMMUNICATION

Single-sender, single-receiver per instance

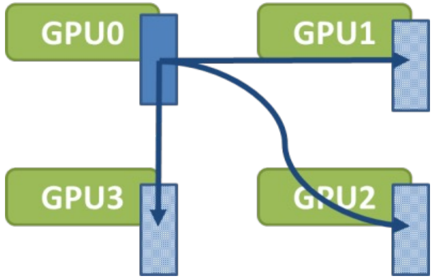
Most common pattern in HPC, where communication is usually to nearest neighbors



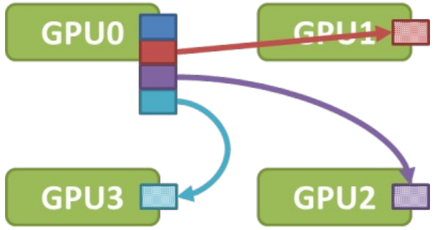
Collective Communication

multiple senders, multiple receivers

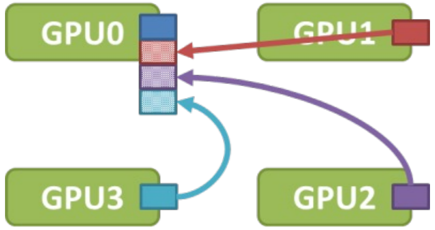
Broadcast



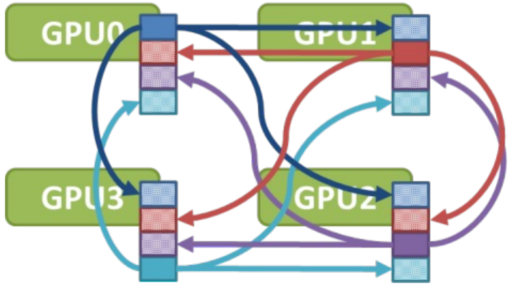
Scatter



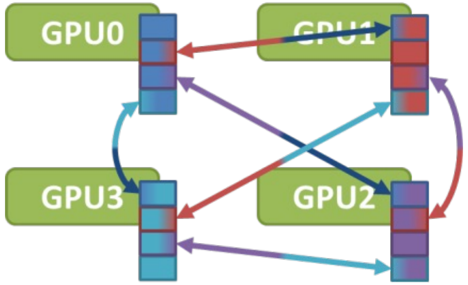
Gather



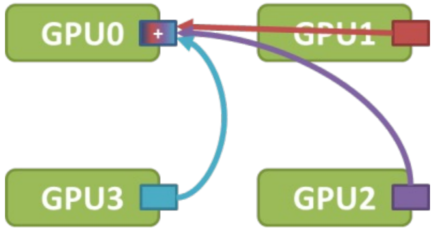
All-Gather



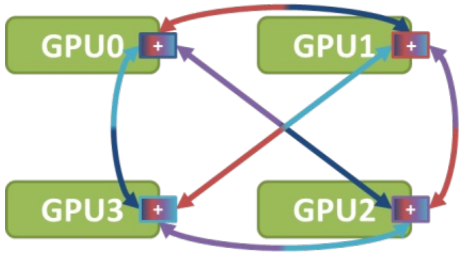
All-to-All



Reduce



All-Reduce



Slides from Nvidia

What Limits the scalability of distributed applications?

Efficiency of parallel computation tasks

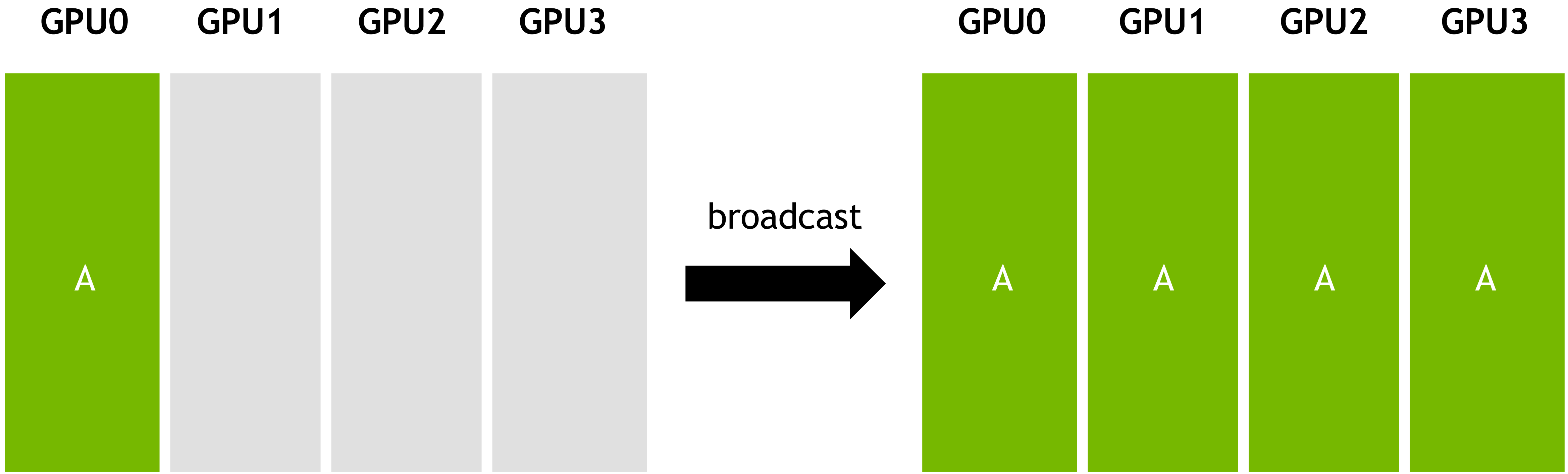
- Amount of exposed parallelism
- Load balance & scheduling overhead

Expense of communications among tasks

- Amount of communication
- Degree of overlap of communication and computation

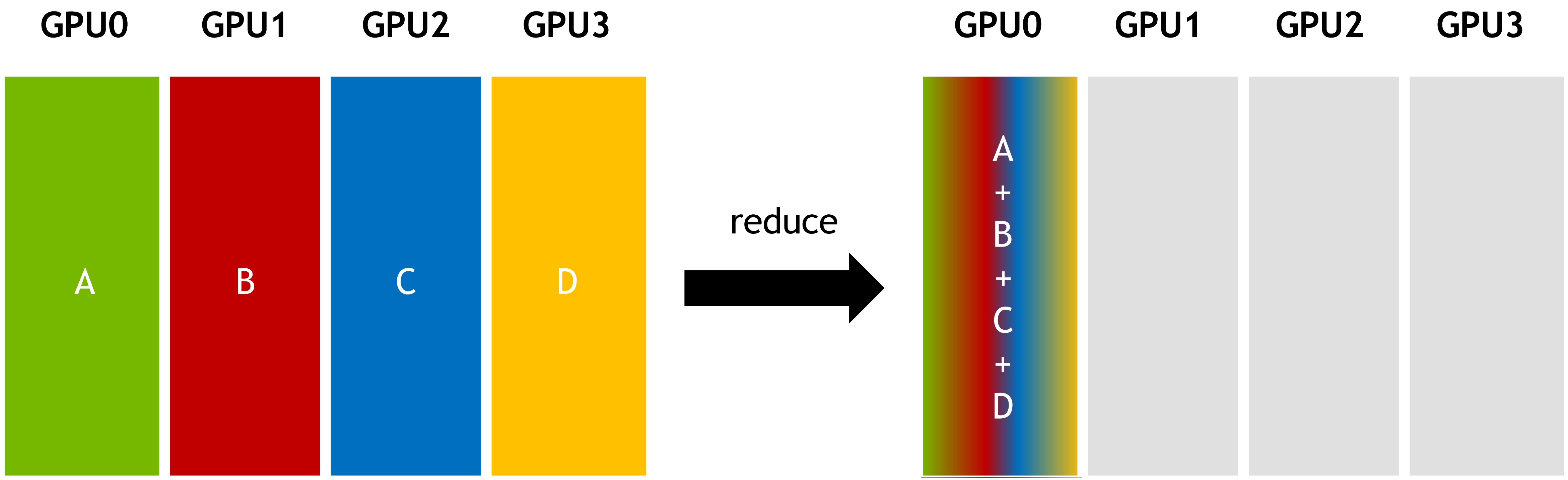
BROADCAST

One sender, multiple receivers



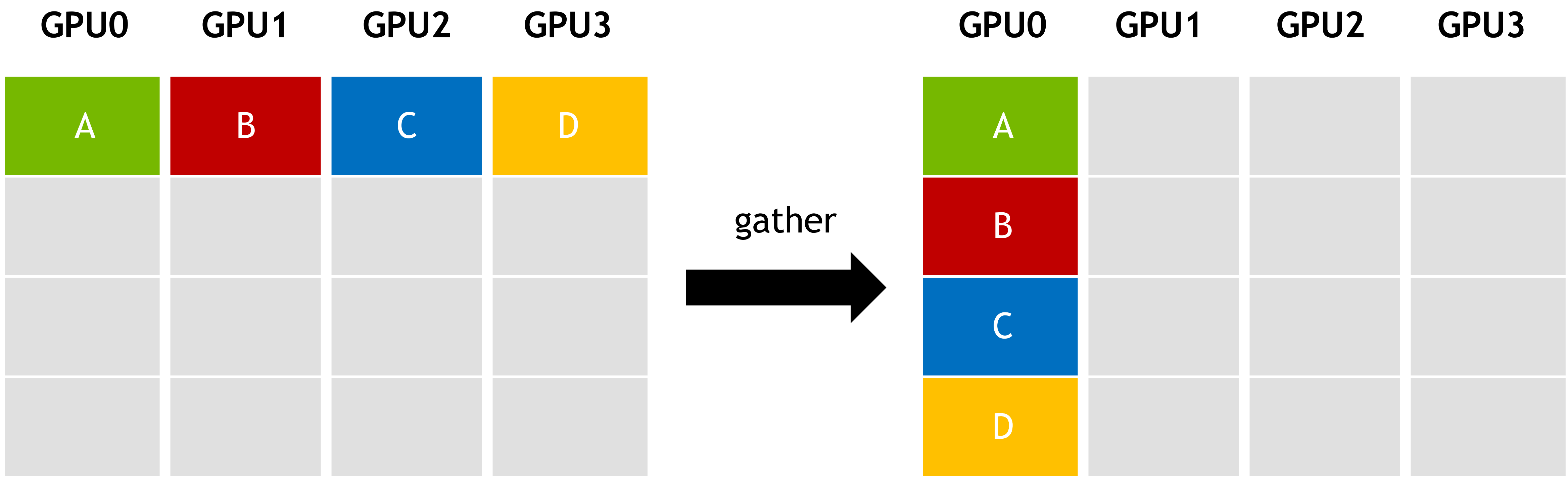
REDUCE

Combine data from all senders; deliver the result to one receiver



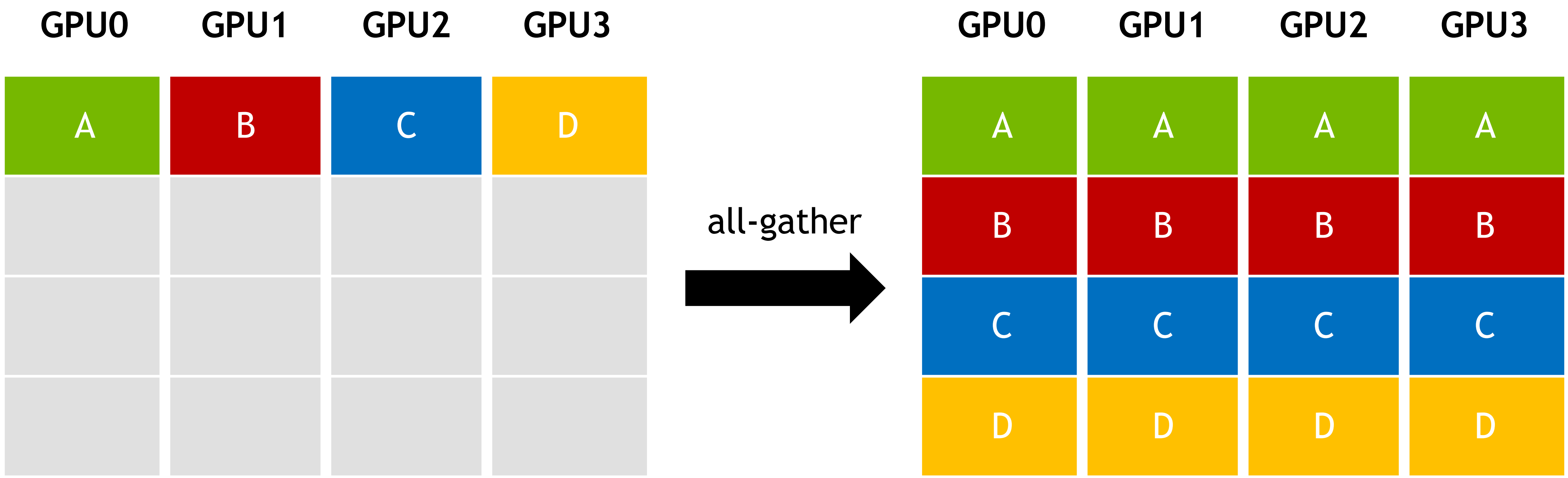
GATHER

Multiple senders, one receiver



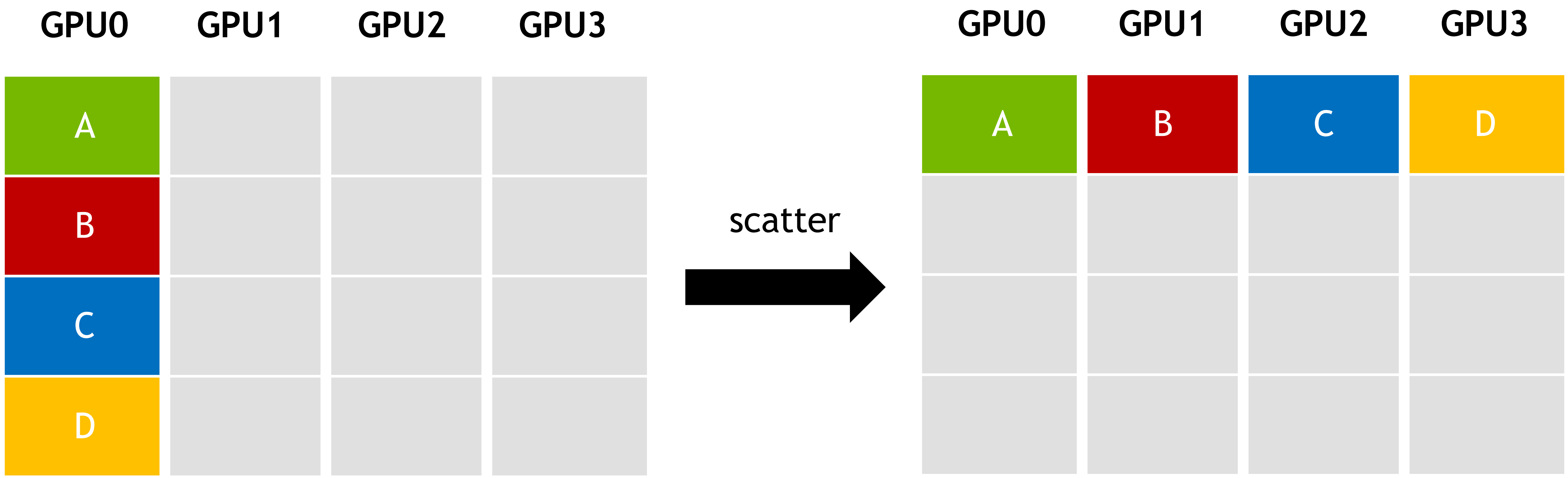
ALL-GATHER

Gather messages from all; deliver gathered data to all participants



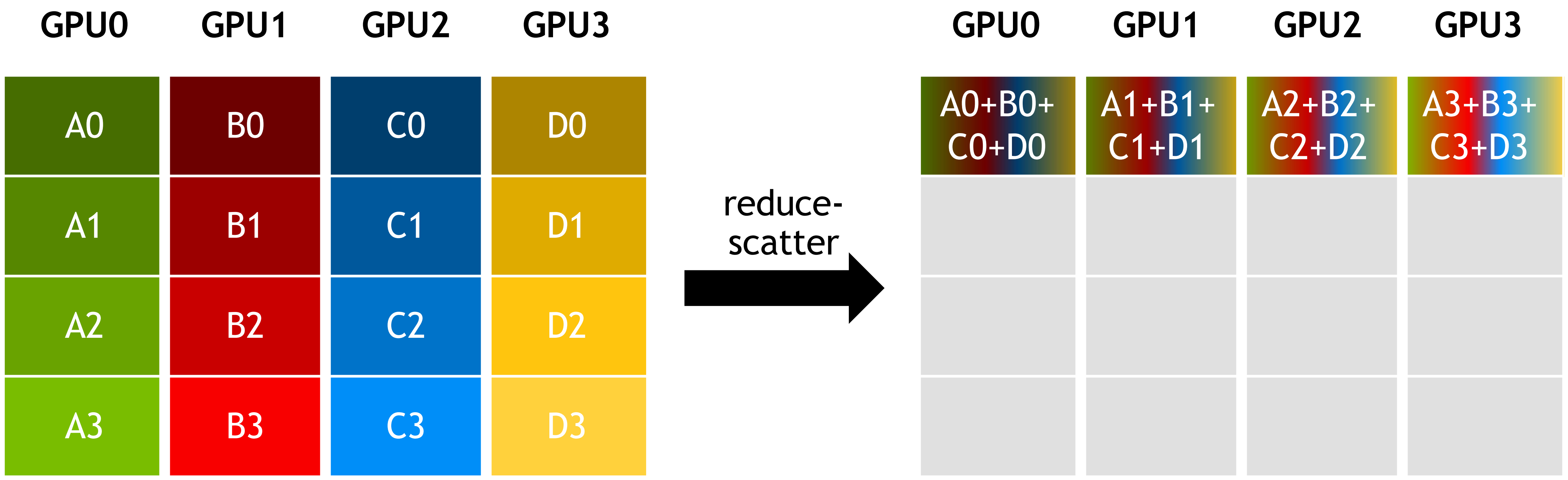
SCATTER

One sender; data is distributed among multiple receivers



REDUCE-SCATTER

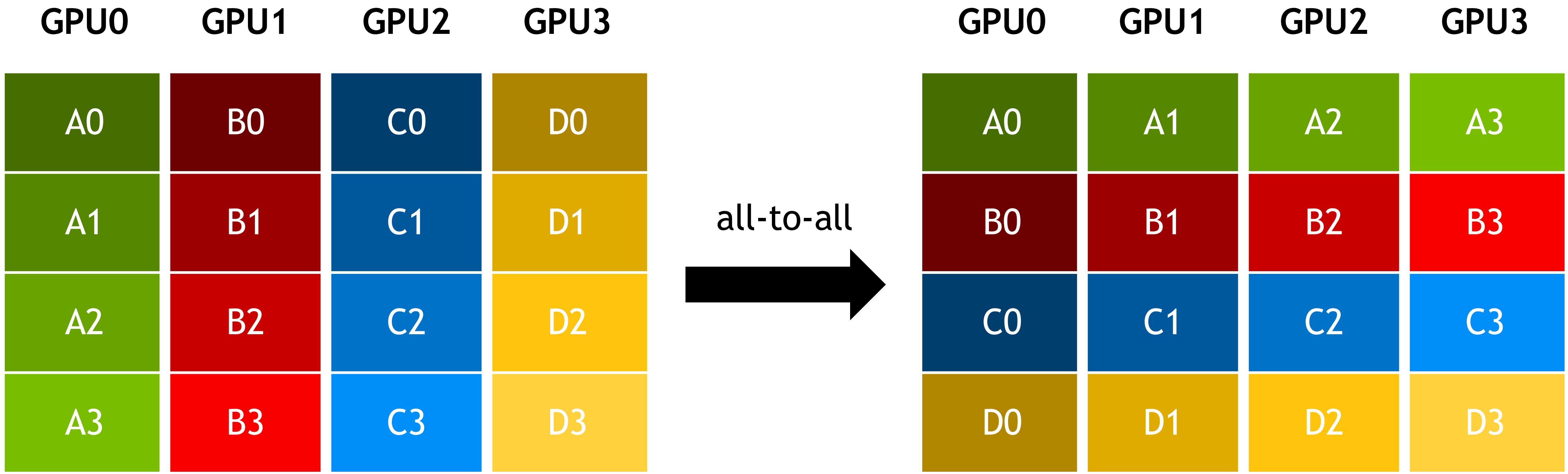
Combine data from all senders; distribute result across participants



Slides from Nvidia

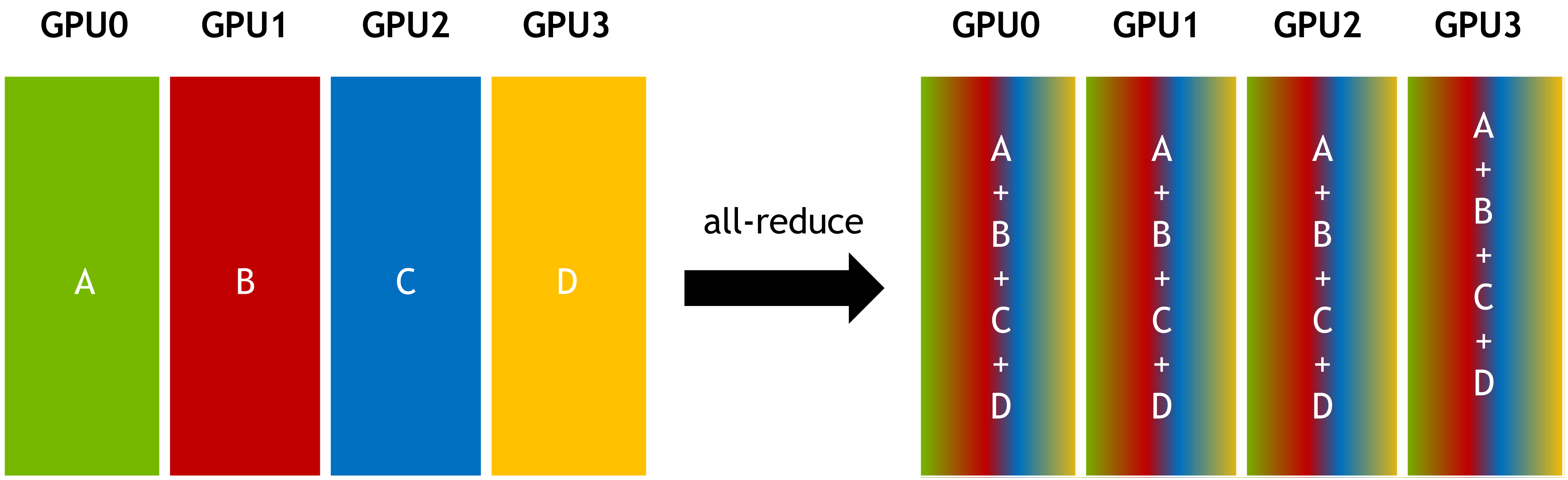
ALL-TO-ALL

Scatter/Gather distinct messages from each participant to every other



ALL-REDUCE

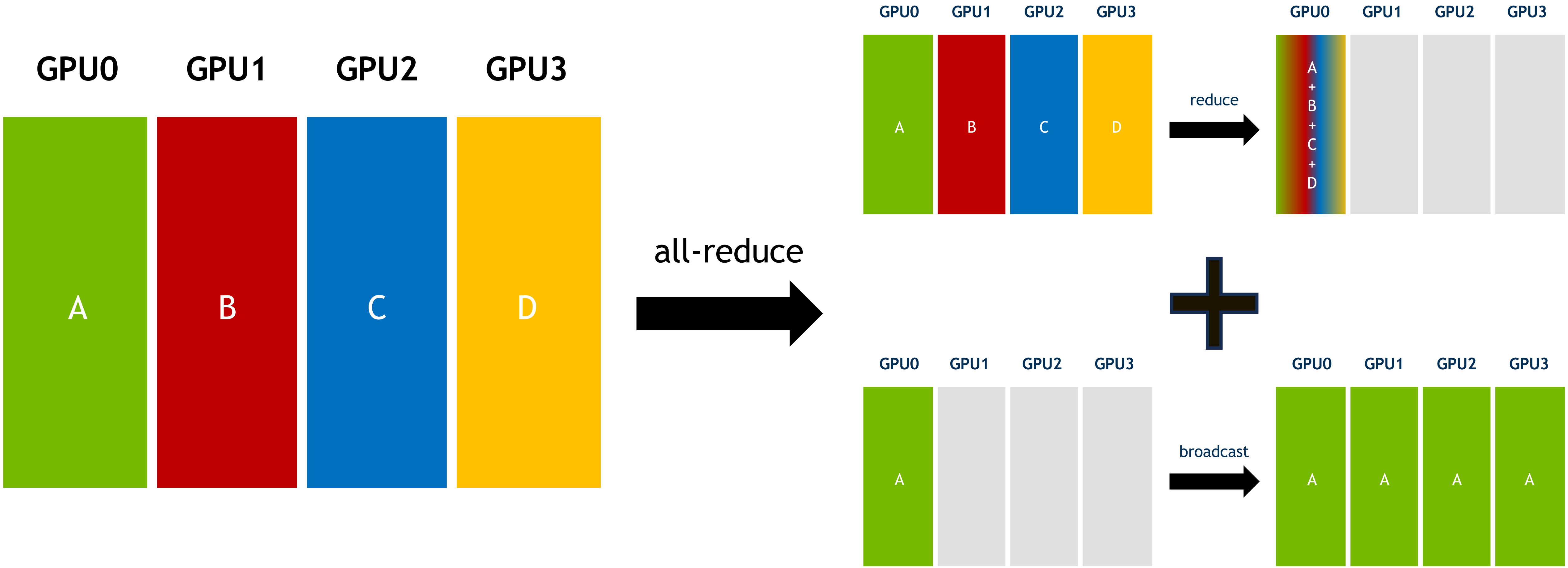
Combine data from all senders; deliver the result to all participants



Slides from Nvidia

ALL-REDUCE

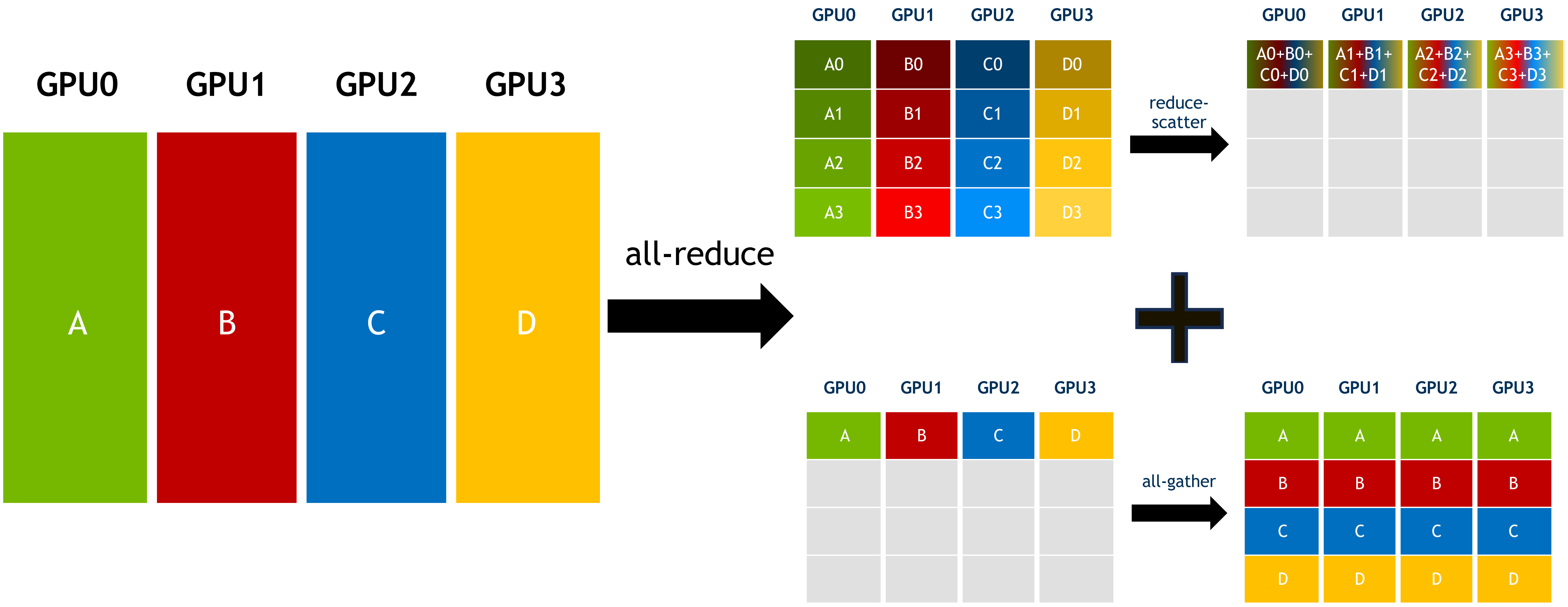
Combine data from all senders; deliver the result to all participants



Slides from Nvidia

ALL-REDUCE

Combine data from all senders; deliver the result to all participants



Slides from Nvidia

THE CHALLENGE OF COLLECTIVES

Collectives are often avoided because they are expensive. Why?

Having multiple senders and/or receivers compounds communication inefficiencies

- For small transfers, latencies dominate; more participants increase latency
- For large transfers, bandwidth is key; bottlenecks are easily exposed
- May require topology-aware implementation for high performance
- Collectives are often blocking/non-overlapped

THE CHALLENGE OF COLLECTIVES

Many implementations seen in the wild are suboptimal

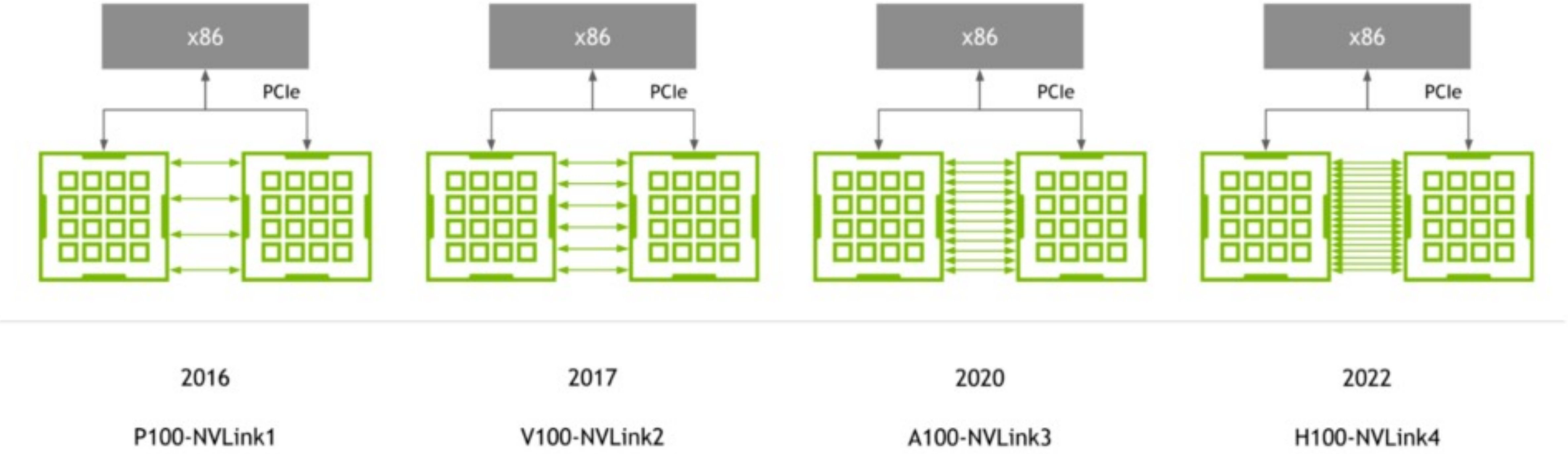
Depends on various factors:

1. Underlying network topology – hierarchical, fat-free, etc.
2. Implementation – ring vs tree-based collectives (logical topology on the network)
3. Data scheduling – compute and communication overlap

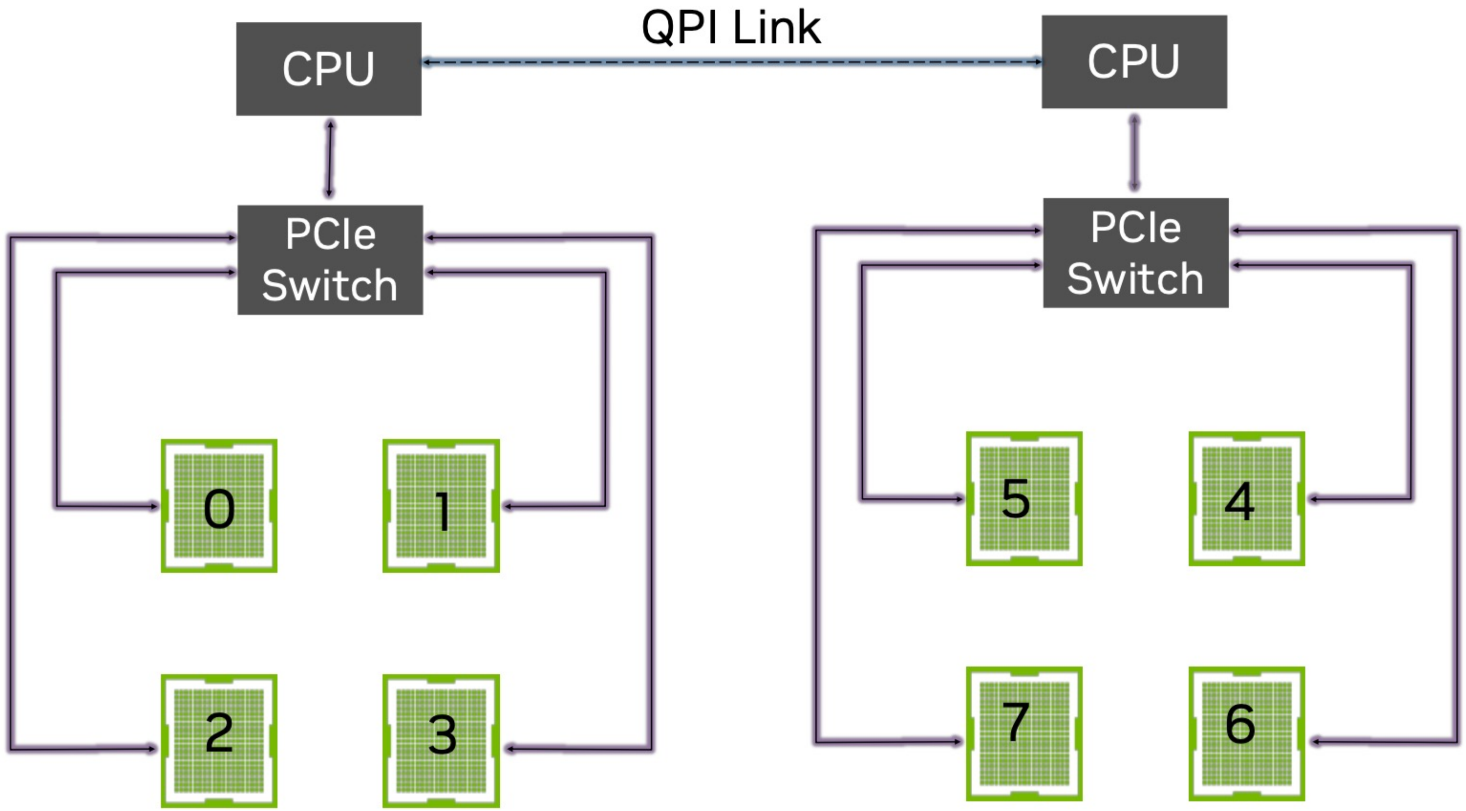
HARDWARE PLATFORMS

1. Multi-GPU boxes interconnected by a datacenter network (E.g., NVIDIA, AMD)
2. Custom interconnects as in TPUv4

NVLink

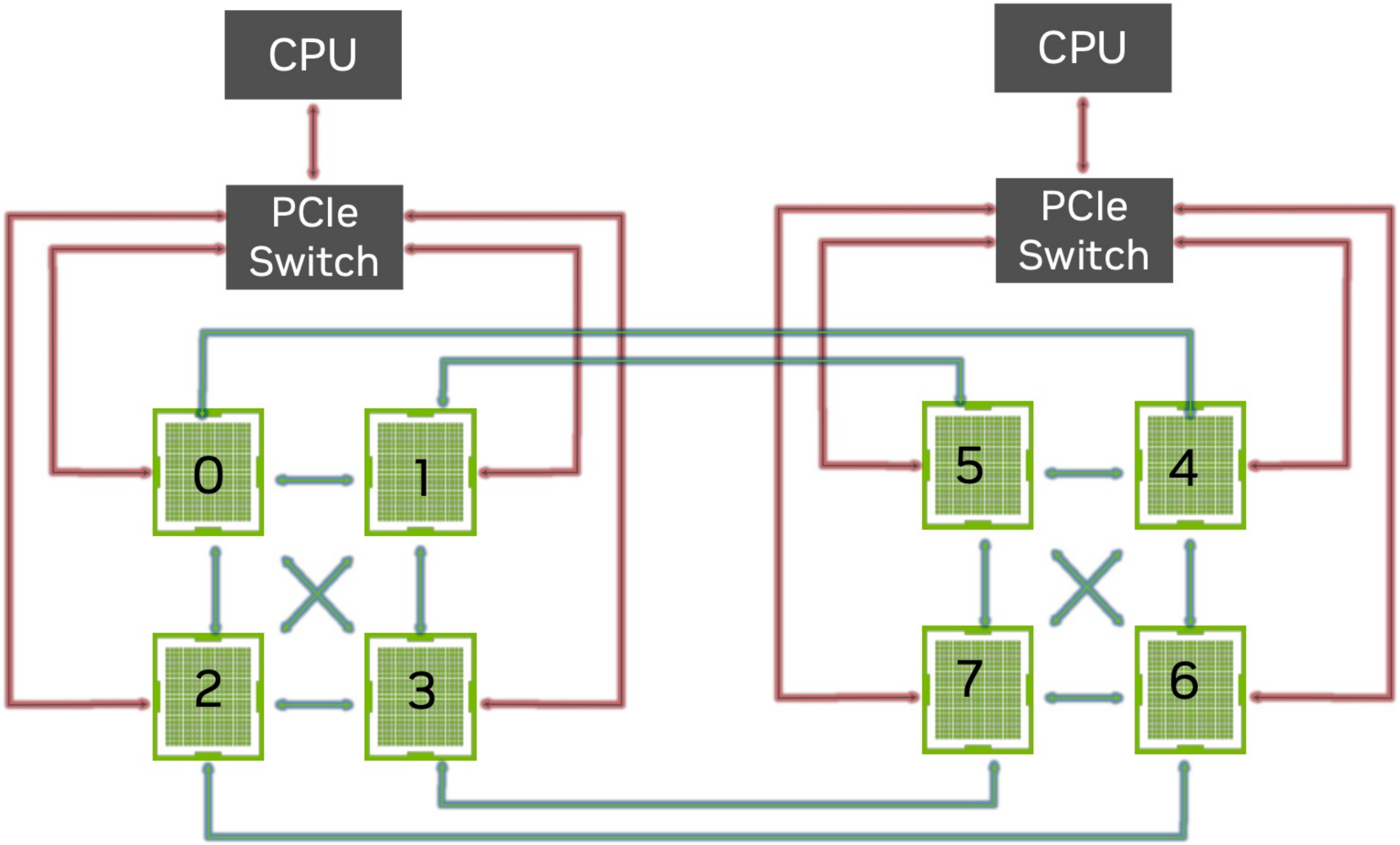

x86
PCIe
x86
PCIe
x86
PCIe
x86
PCIe
2016
P100-NVLink1
2017
V100-NVLink2
2020
A100-NVLink3
2022
H100-NVLink4

Data Parallelism – PCIe based



Data loading and gradient averaging share communication resources → congestion

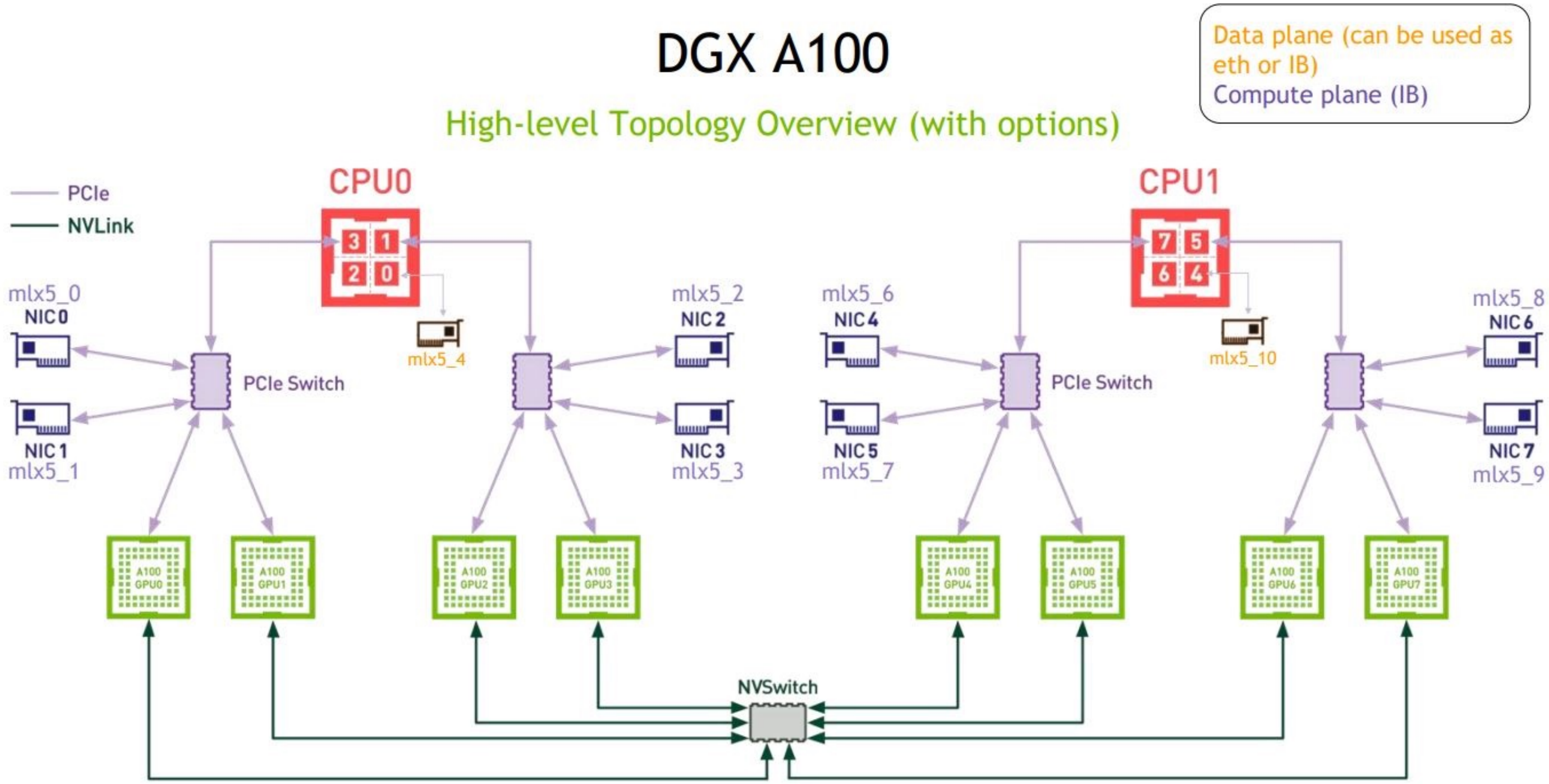
Data Parallelism – NVLink



Datal loading on PCIe, gradient averaging on NVLINK → no congestion

Nvidia DGX Machine – A100

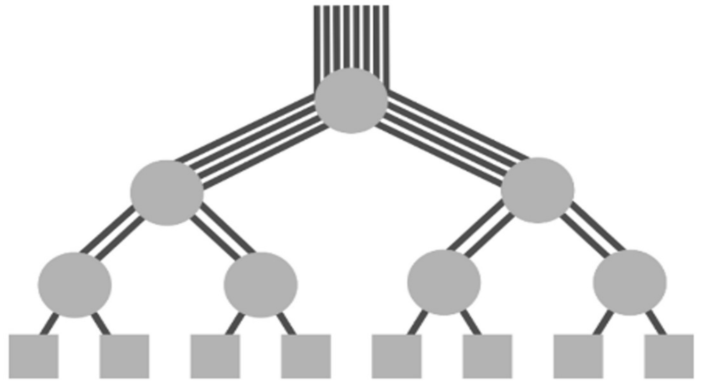
Number of GPUs	8
NVLink Bandwidth	300 GBps per GPU
NIC PCIe Bandwidth	12.5 GBps per GPU



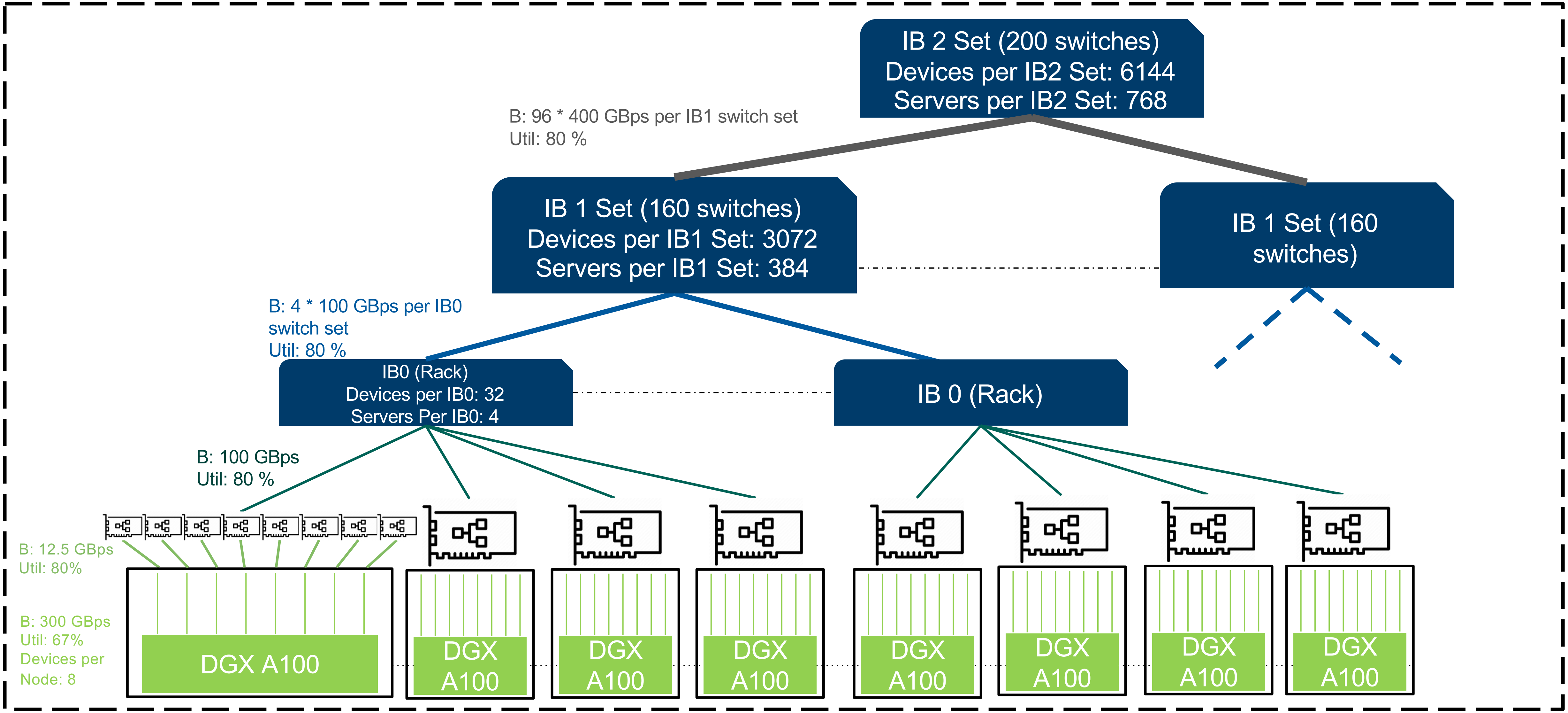
Topology - Fat Trees

- For any switch, the number of links going *down* to its siblings is equal to the number of links going *up* to its parent in the upper level.

- If oversubscribed, the number of uplinks is less than the number of downlinks
- **Non-blocking**: the number of uplinks and downlinks are in proportion of 1:1
    - blocking factor = # downlink/ # uplink



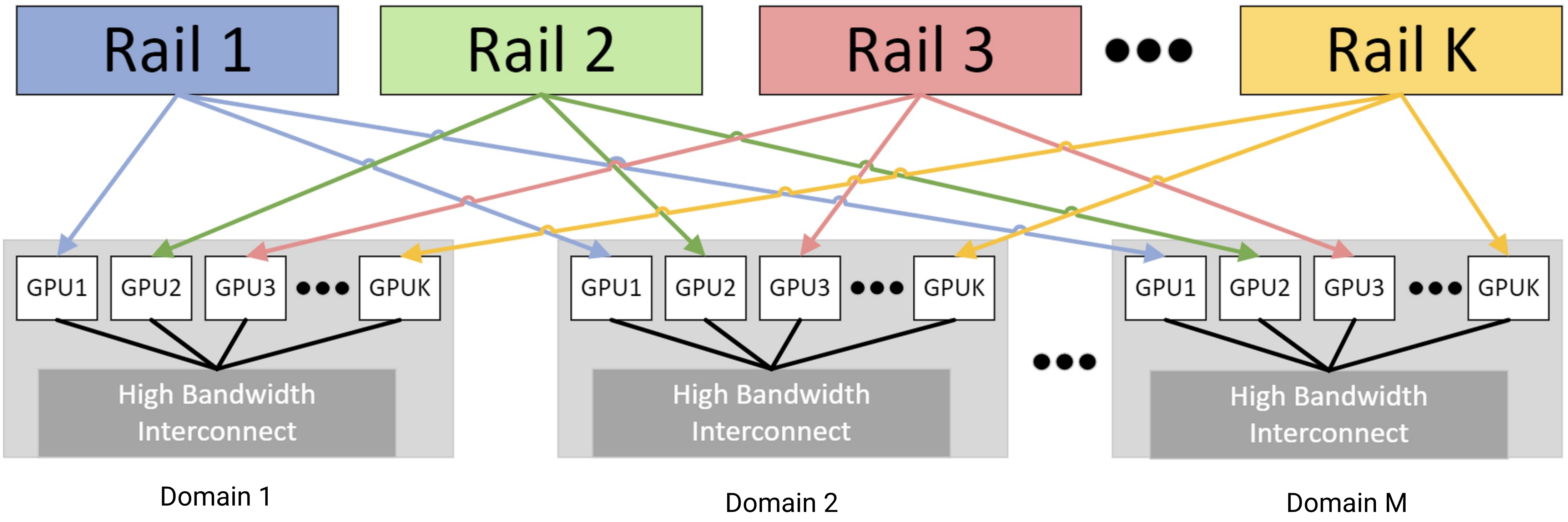
4-Tier Topology-Fat Tree



Rail only (Meta)

Across each High Bandwidth Domain, only GPUs of the same index (same rail) are connect via rail interconnection

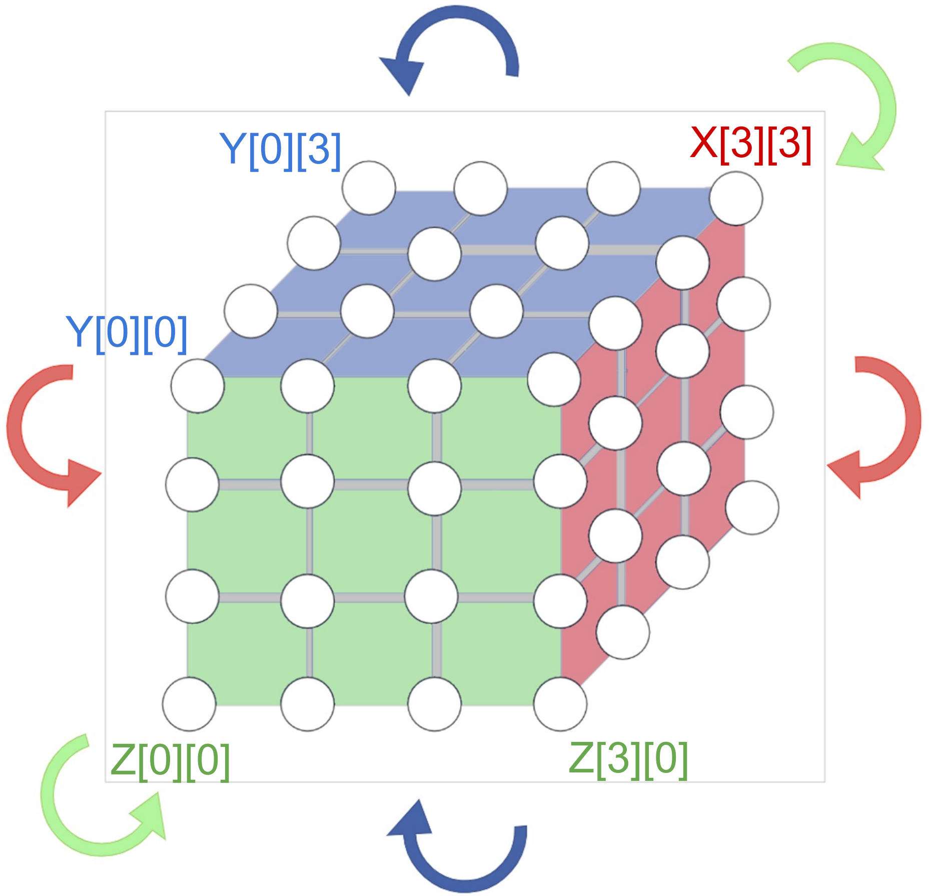
- Only way for Inter-Domain communication of GPU in different rails, are by hopping through GPUs



Optimized Network Architectures for Large Language Model Training with Billions of Parameters

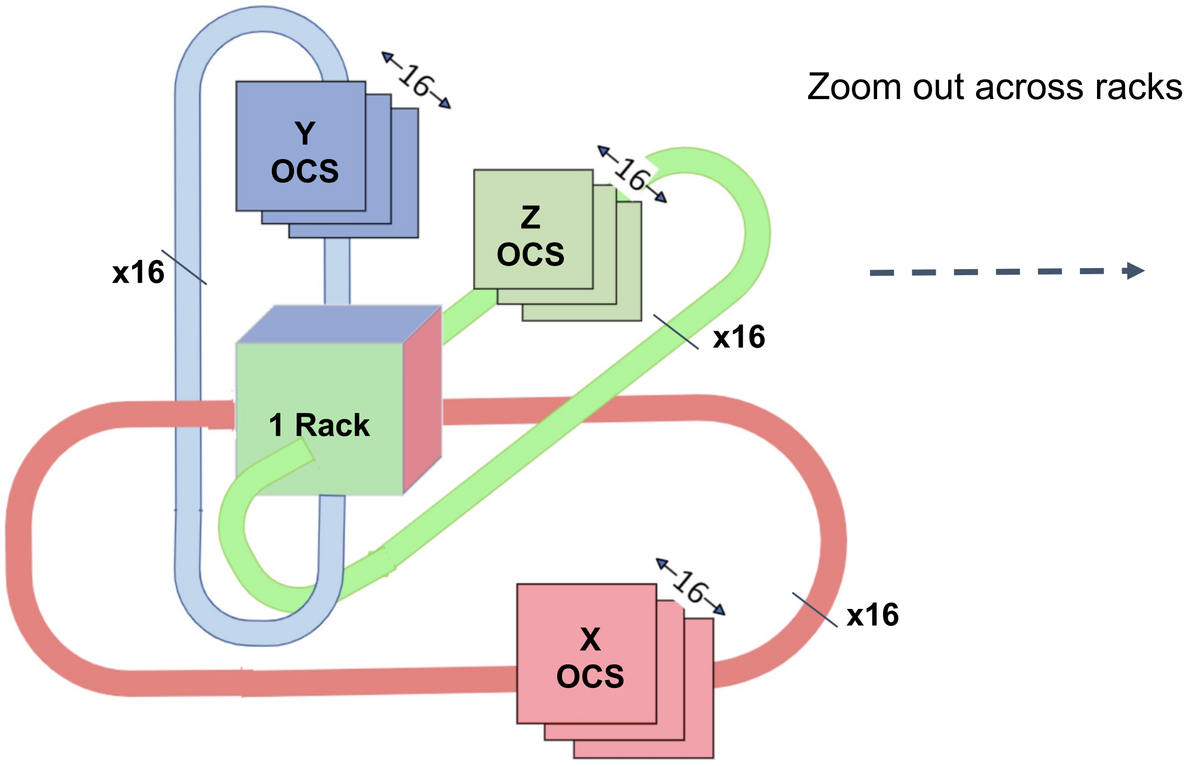
Google TPUv4 Racks

Topology	3D torus
TPUs per Rack	64 (4x4x4)
Intra- rack connection	Inter-Chip Interconnection links 50 GB/s
Inter- rack connection	OCS links connected to surface nodes
Total TPUs per Pod	4096

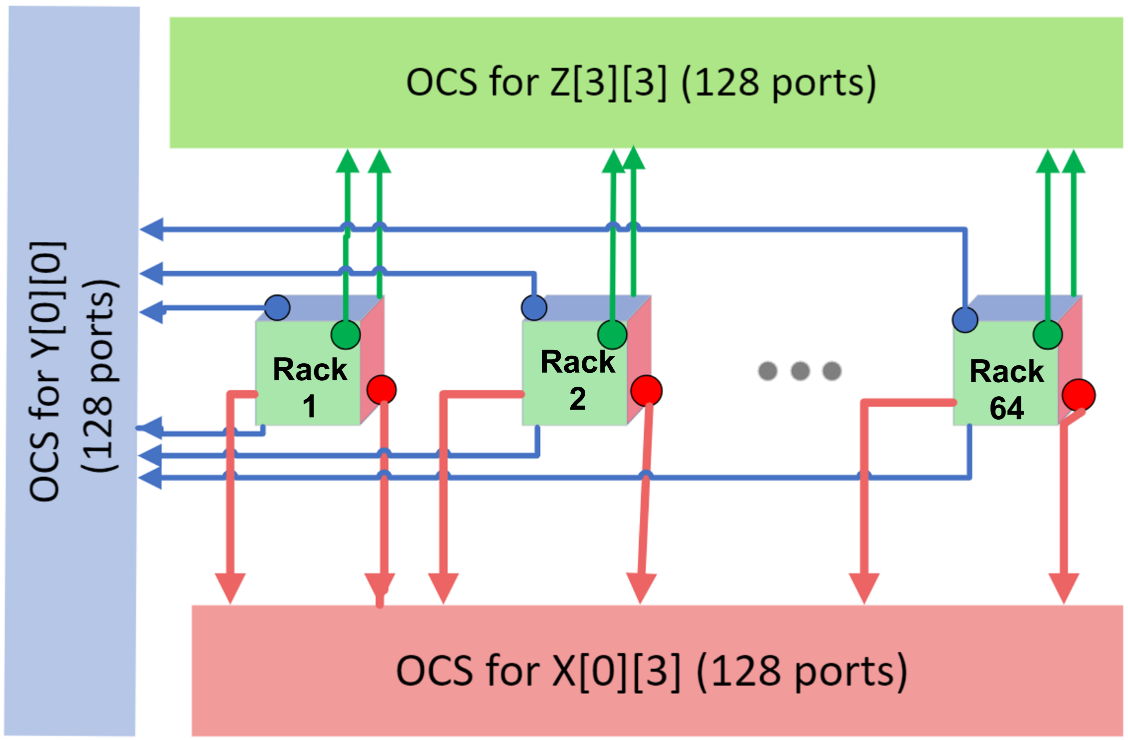


One TPUv4 rack

TPUv4 connections across racks



OCS connection for 1 rack.



Three out of the 48 OCS connections across 64 Racks

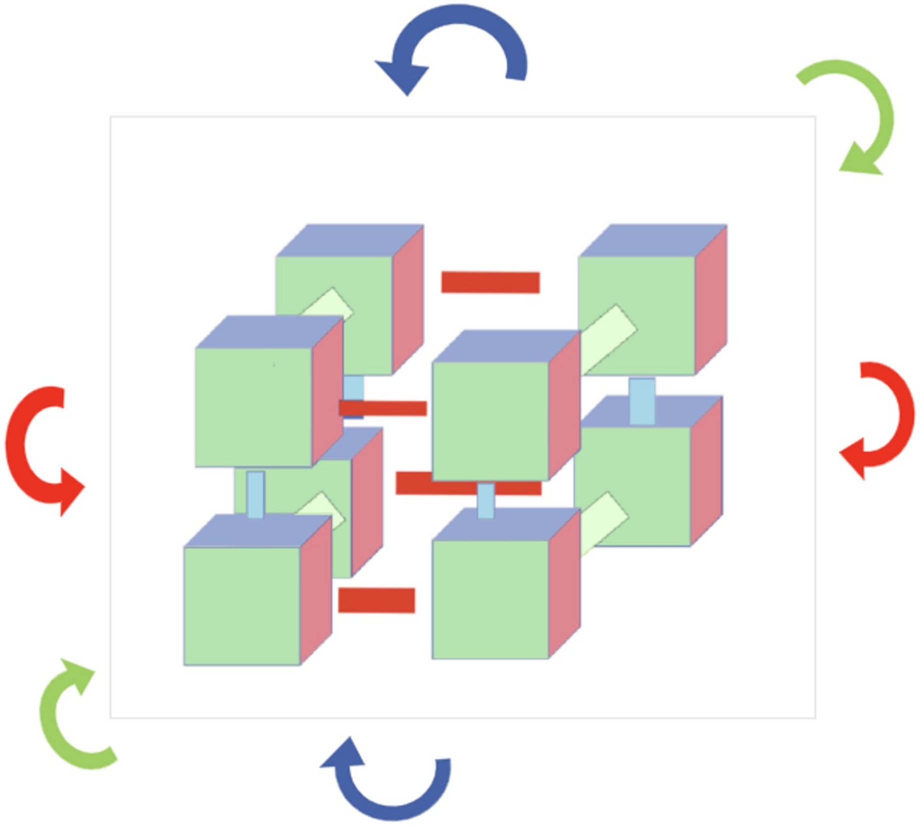
- 16 OCSes per dimension (**connected to nodes in both opposite faces**)
- **48** OCSes connected to each rack (cube)

- Each OCS is connected to all 64 racks in the pod.

TPUv4 connections across racks

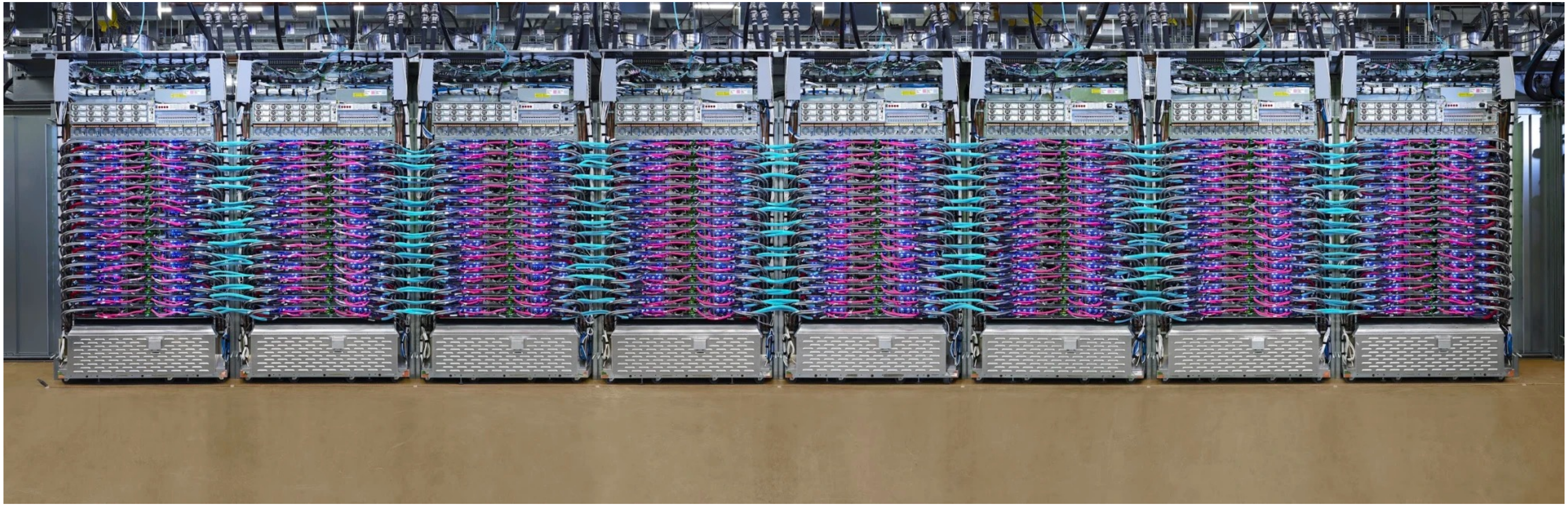
With OCS, we can create any arbitrary topology with 4i x 4j x 4k nodes

- 8x8x8 is 8 racks in assembled as a 3D Torus
- **With 4096 Nodes, we will have a 16x16x16 Topology**



8x8x8 Topology with OCS connections

Often deploy these devices in a cluster or a pod



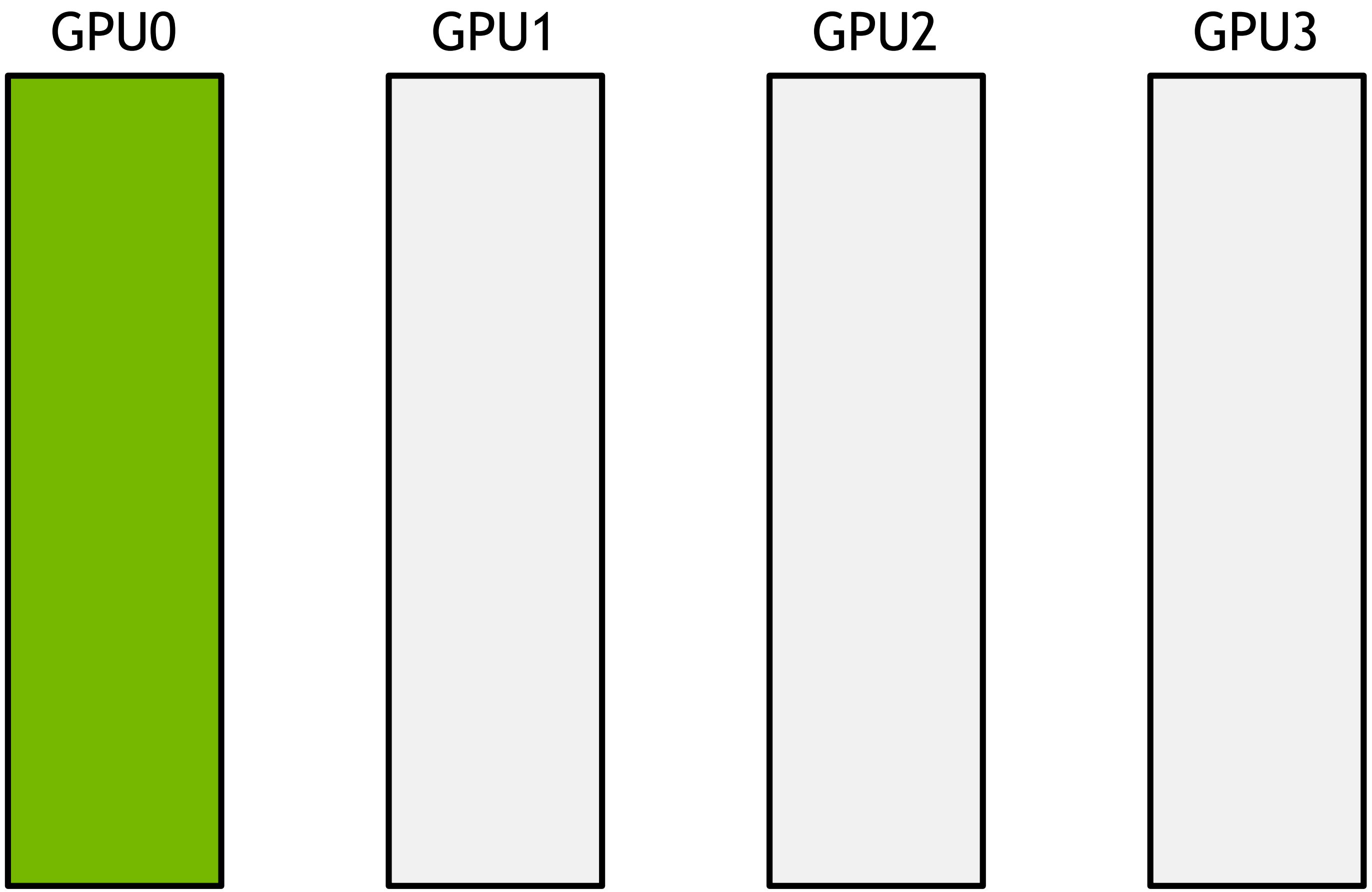
Eight of 64 racks for one 4096-chip supercomputer

Figure Courtesy: Google Cloud

RING-BASED COLLECTIVES: A PRIMER

BROADCAST

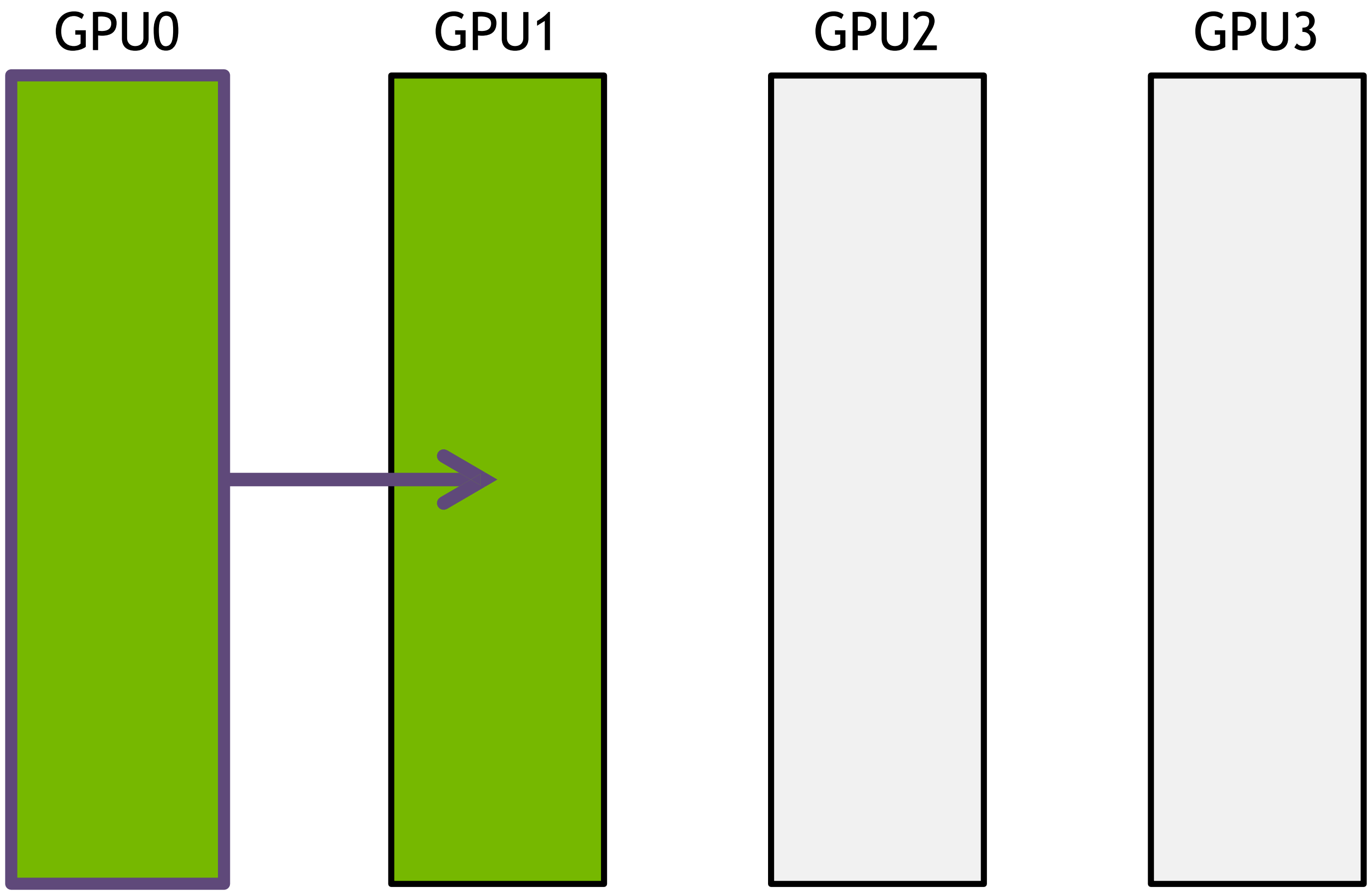
with unidirectional ring



Slides from Nvidia

BROADCAST

with unidirectional ring



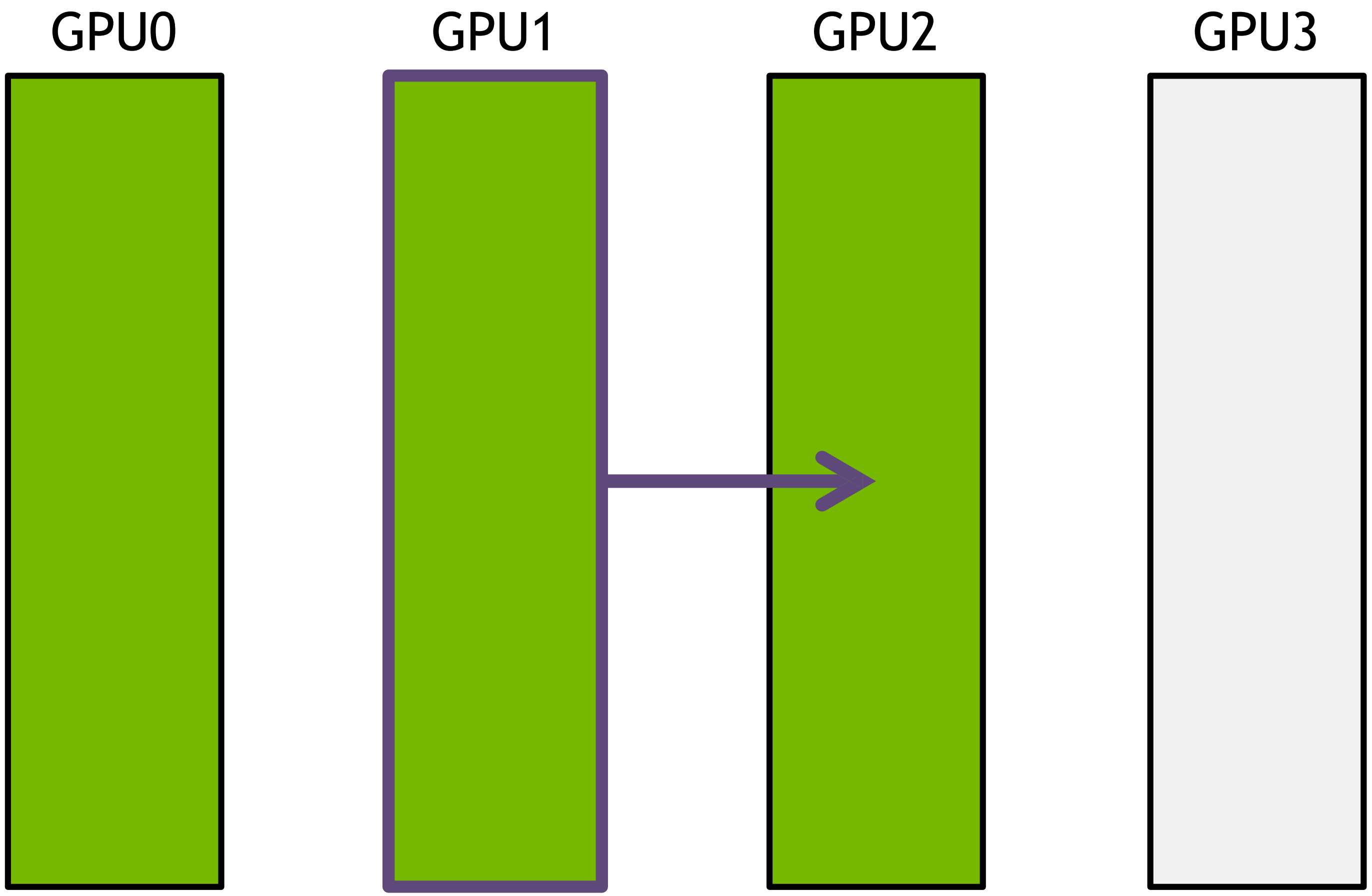
Step 1: $\Delta t = N/B$

N: bytes to broadcast

B: bandwidth of each link

BROADCAST

with unidirectional ring



Step 1: $\Delta t = N/B$

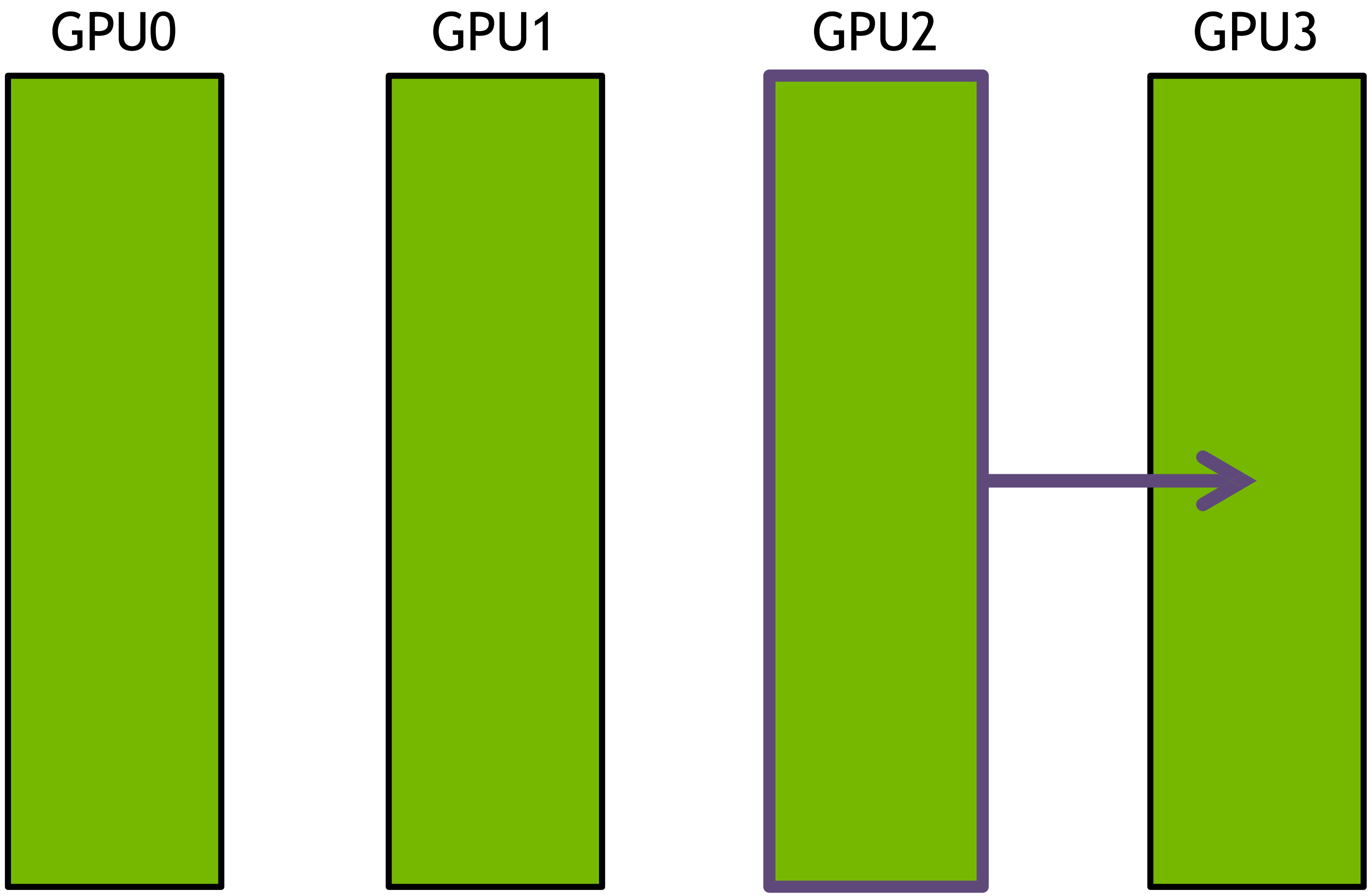
Step 2: $\Delta t = N/B$

N: bytes to broadcast

B: bandwidth of each link

BROADCAST

with unidirectional ring



Step 1: $\Delta t = N/B$

Step 2: $\Delta t = N/B$

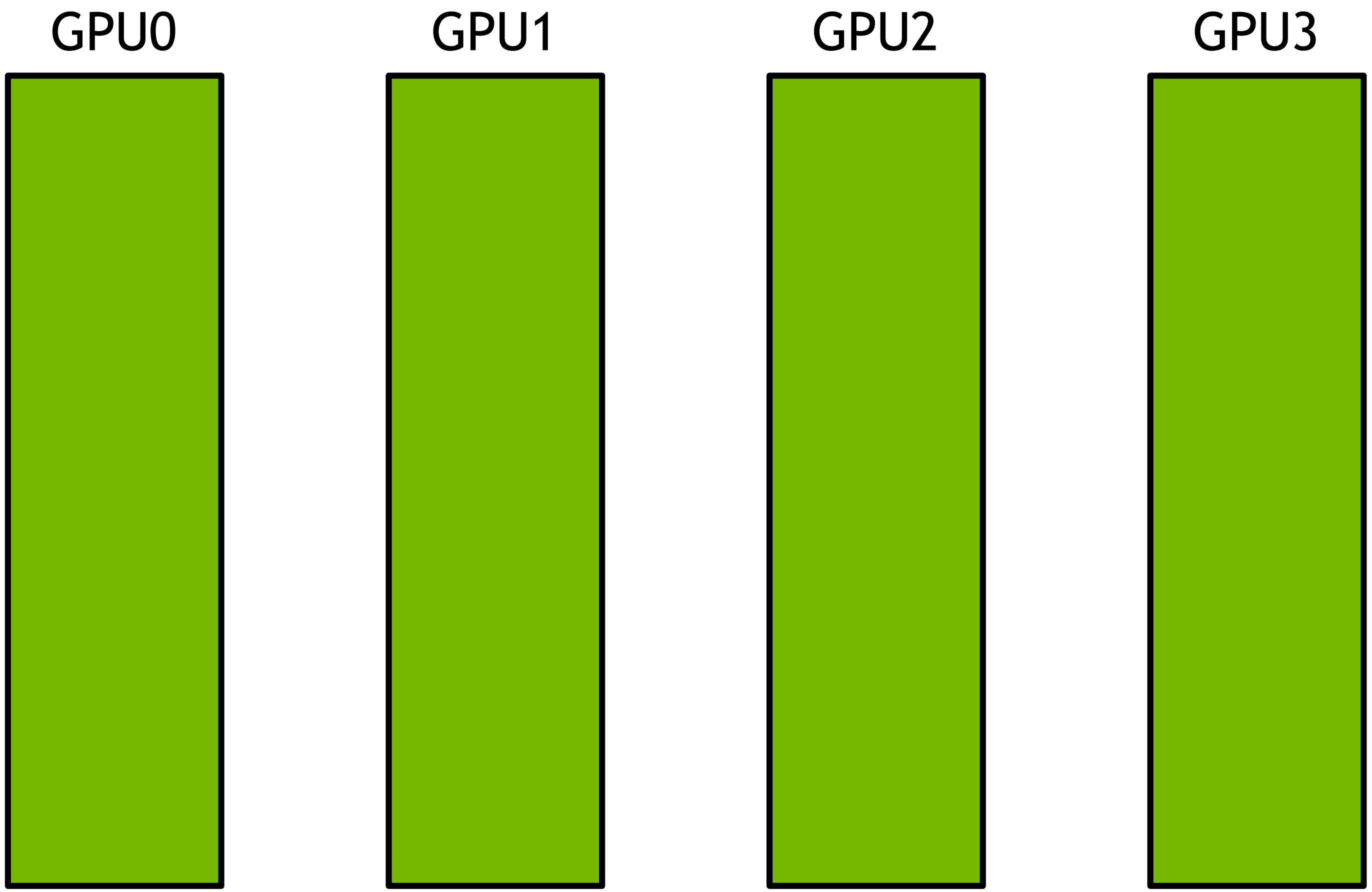
Step 3: $\Delta t = N/B$

N: bytes to broadcast

B: bandwidth of each link

BROADCAST

with unidirectional ring



Step 1: $\Delta t = N/B$

Step 2: $\Delta t = N/B$

Step 3: $\Delta t = N/B$

Total time: $(k-1)N/B$

N: bytes to broadcast

B: bandwidth of each link

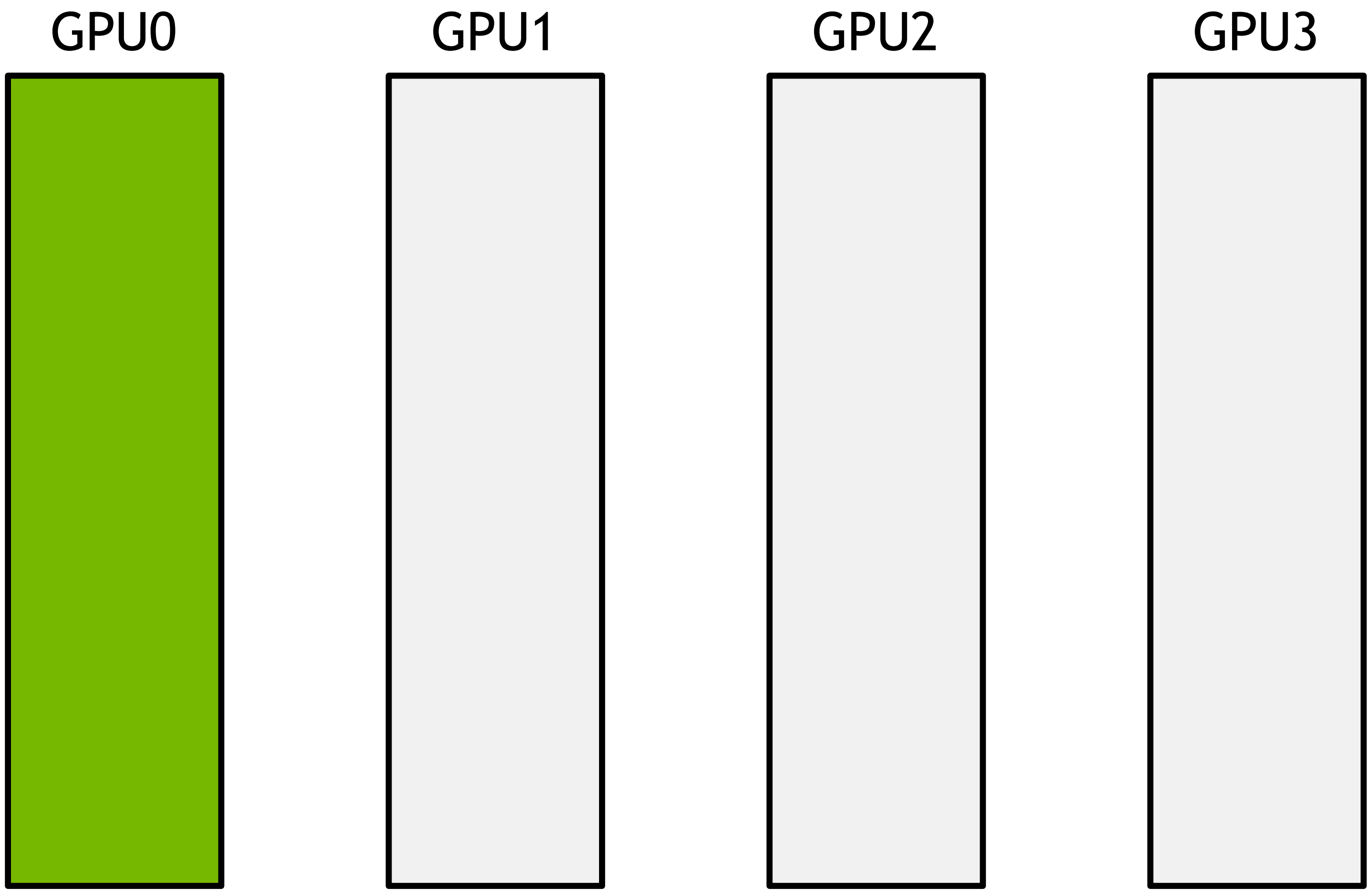
k: number of GPUs

Outline

- Collective Communication
  - Collective Patterns – mathematical operators
  - Collective Algorithm – the way it is logically executed in the network
  - Collective Scheduling - the schedule of the data transfer and compute

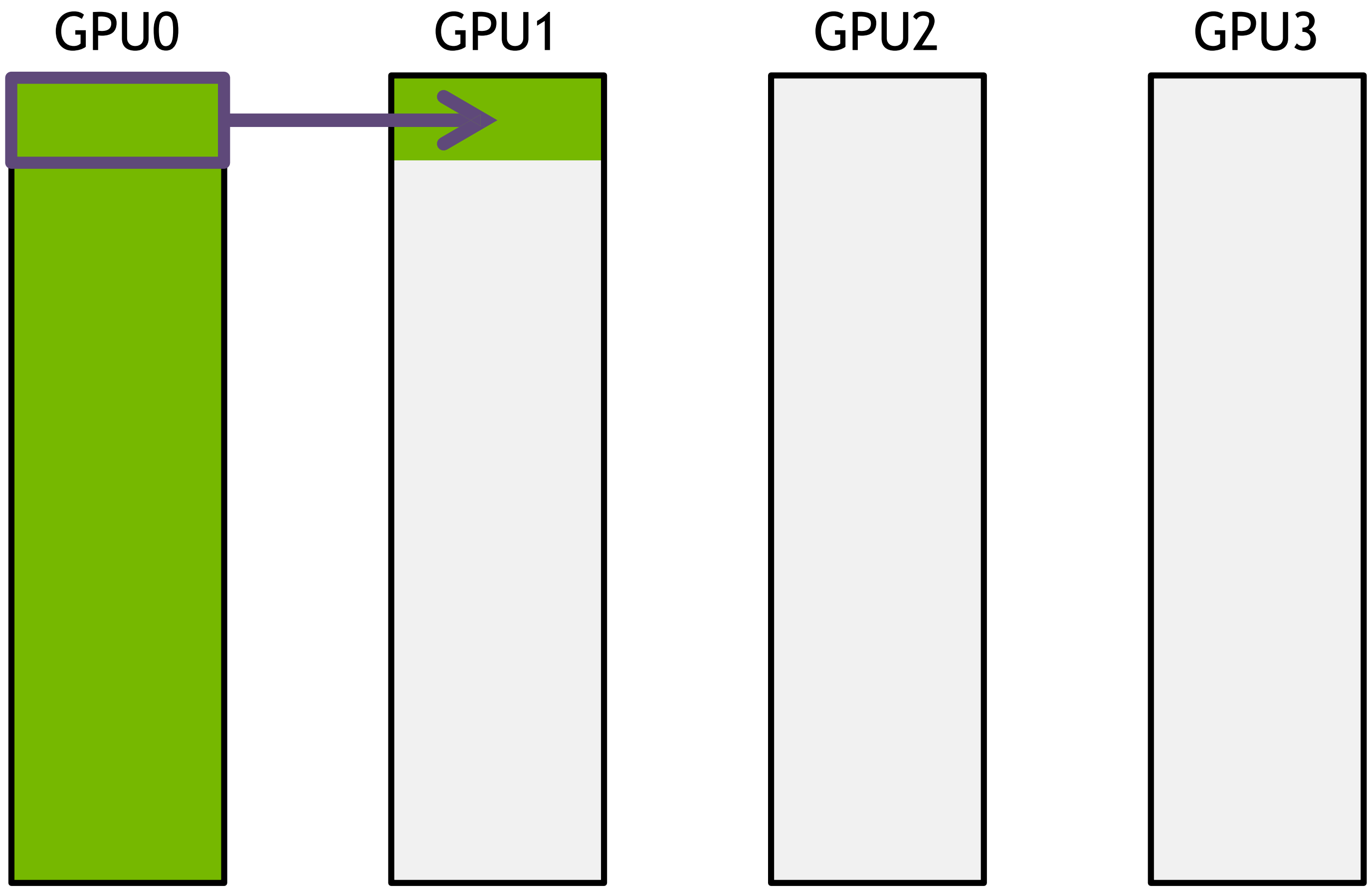
BROADCAST

with unidirectional ring



BROADCAST

with unidirectional ring

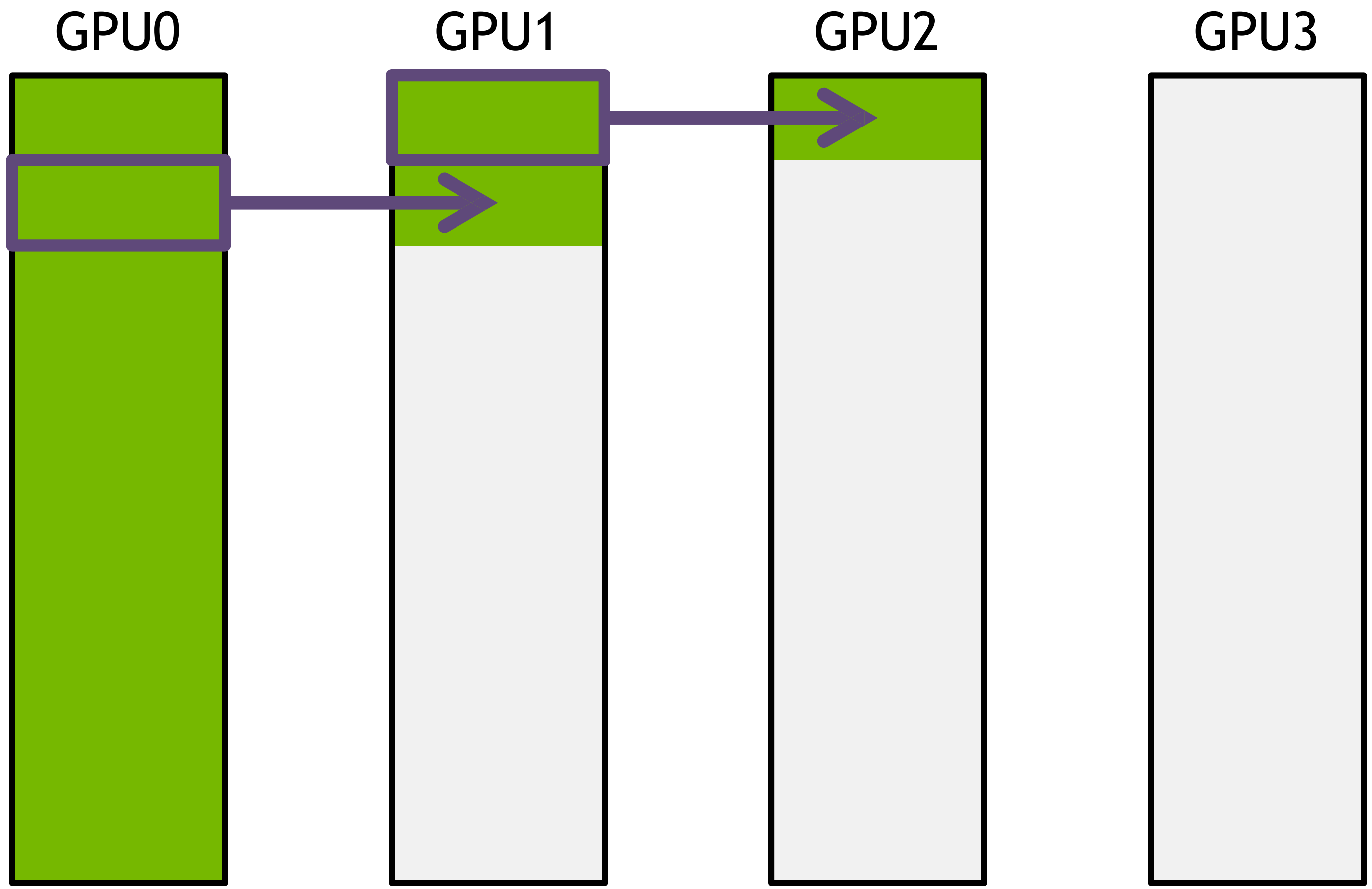


Split data into S messages

Step 1: $\Delta t = N/(SB)$

BROADCAST

with unidirectional ring



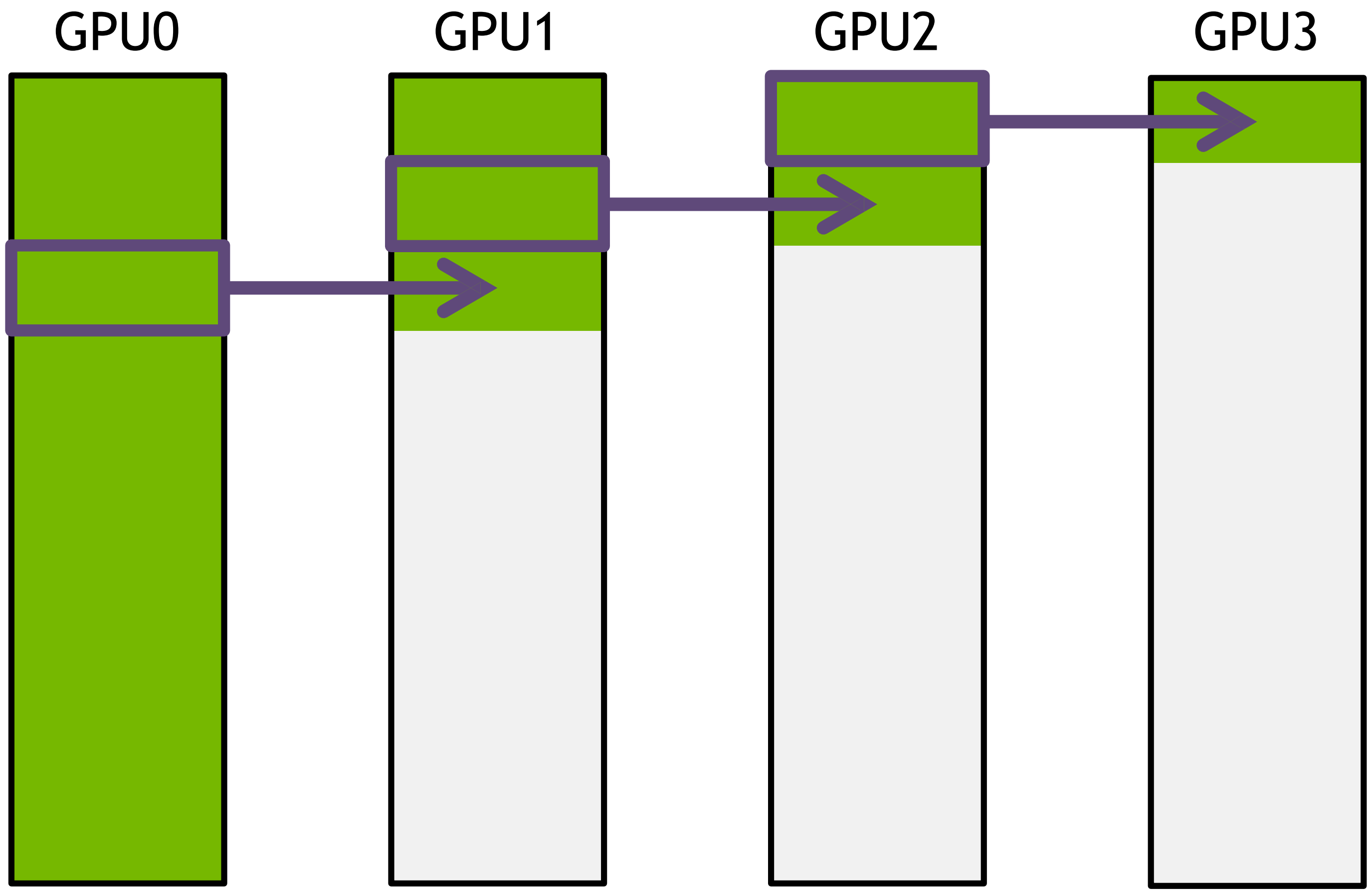
Split data into S messages

Step 1: $\Delta t = N/(SB)$

Step 2: $\Delta t = N/(SB)$

BROADCAST

with unidirectional ring



Split data into S messages

Step 1: $\Delta t = N/(SB)$

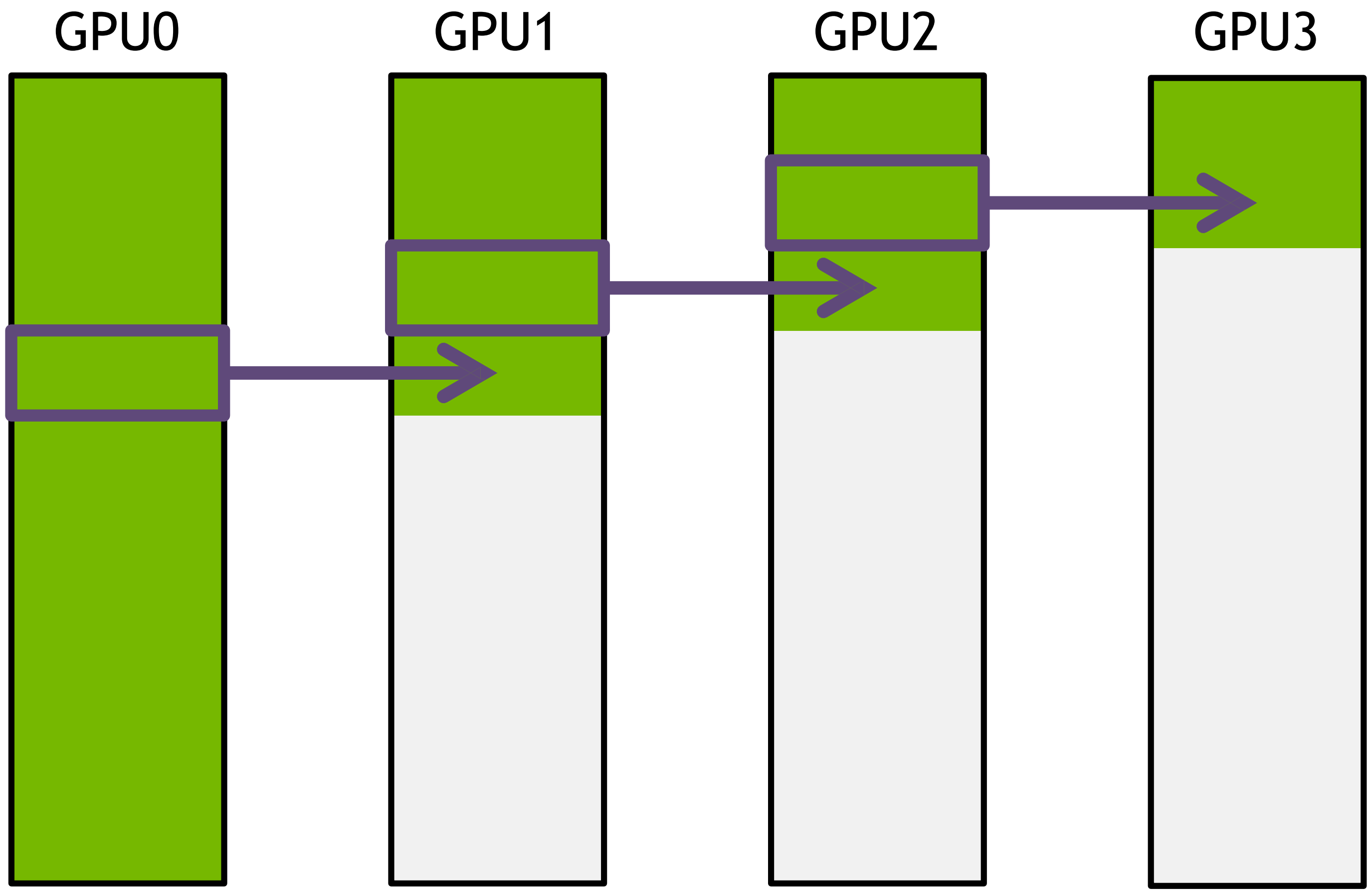
Step 2: $\Delta t = N/(SB)$

Step 3: $\Delta t = N/(SB)$

Slides from Nvidia

BROADCAST

with unidirectional ring



Split data into S messages

Step 1: $\Delta t = N/(SB)$

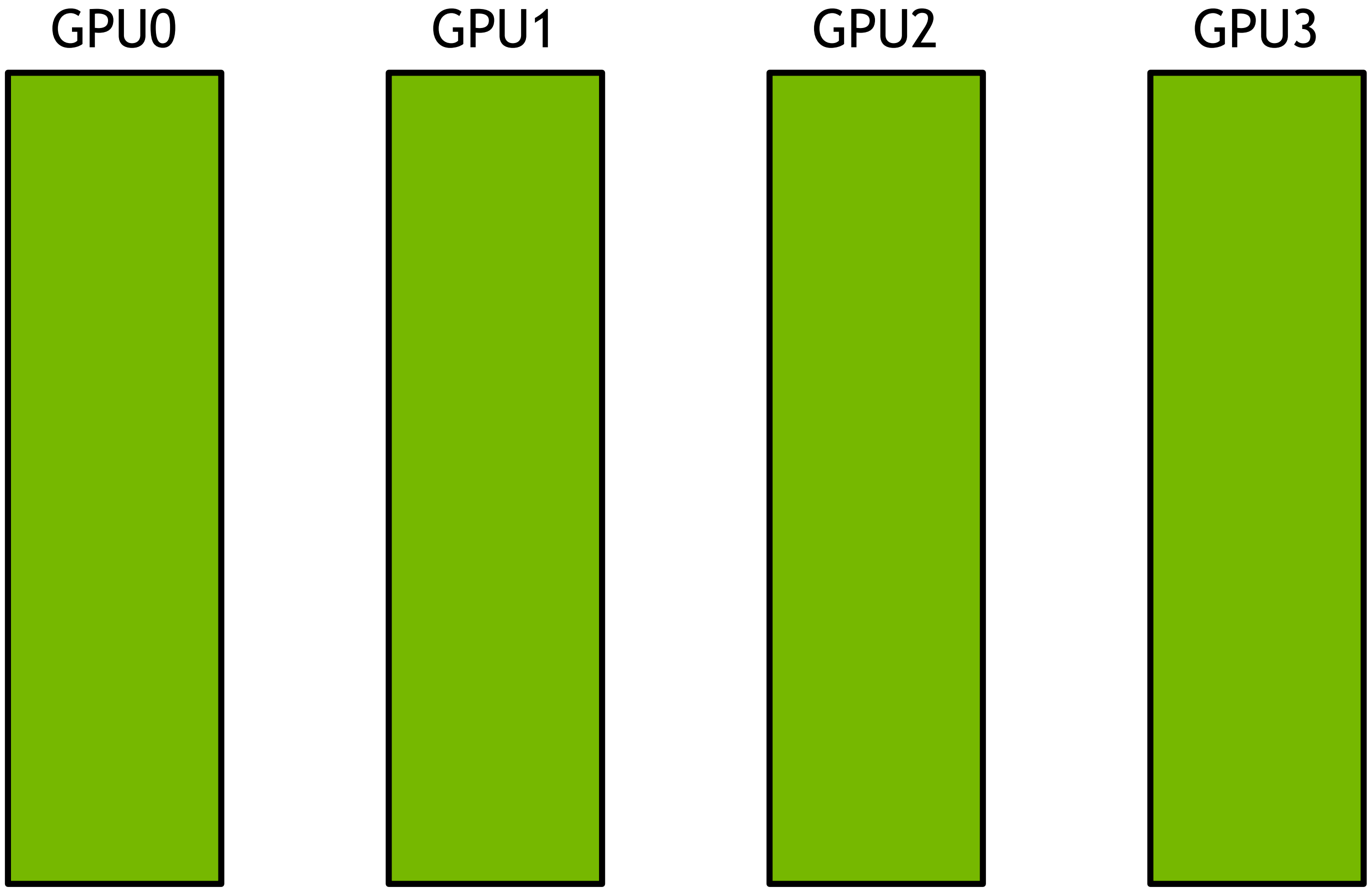
Step 2: $\Delta t = N/(SB)$

Step 3: $\Delta t = N/(SB)$

Step 4: $\Delta t = N/(SB)$

BROADCAST

with unidirectional ring



Split data into S messages

Step 1: $\Delta t = N/(SB)$

Step 2: $\Delta t = N/(SB)$

Step 3: $\Delta t = N/(SB)$

Step 4: $\Delta t = N/(SB)$

...

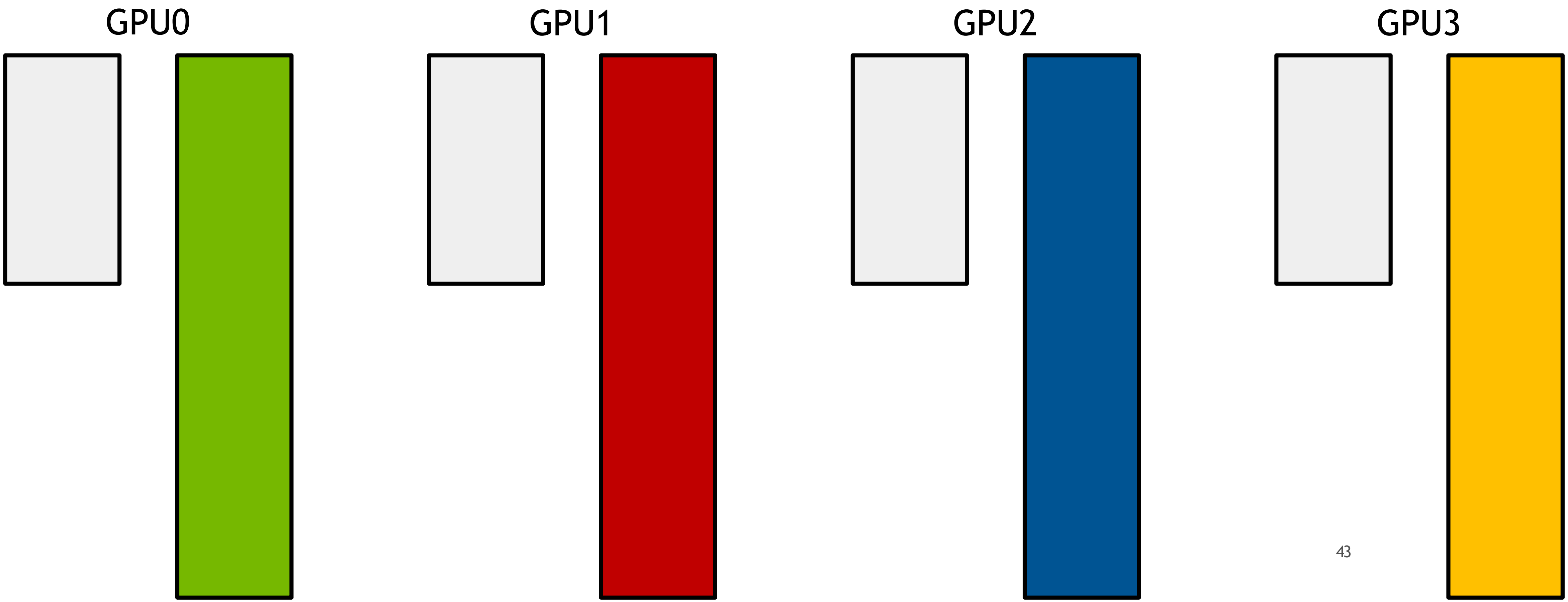
Total time:
$SN/(SB) + (k-2)\,N\,/(SB)$
$= N(S+k-2)/(SB) \rightarrow N/B$

Slides from Nvidia

ALL-REDUCE

with unidirectional ring

Chunk: 1
Step:

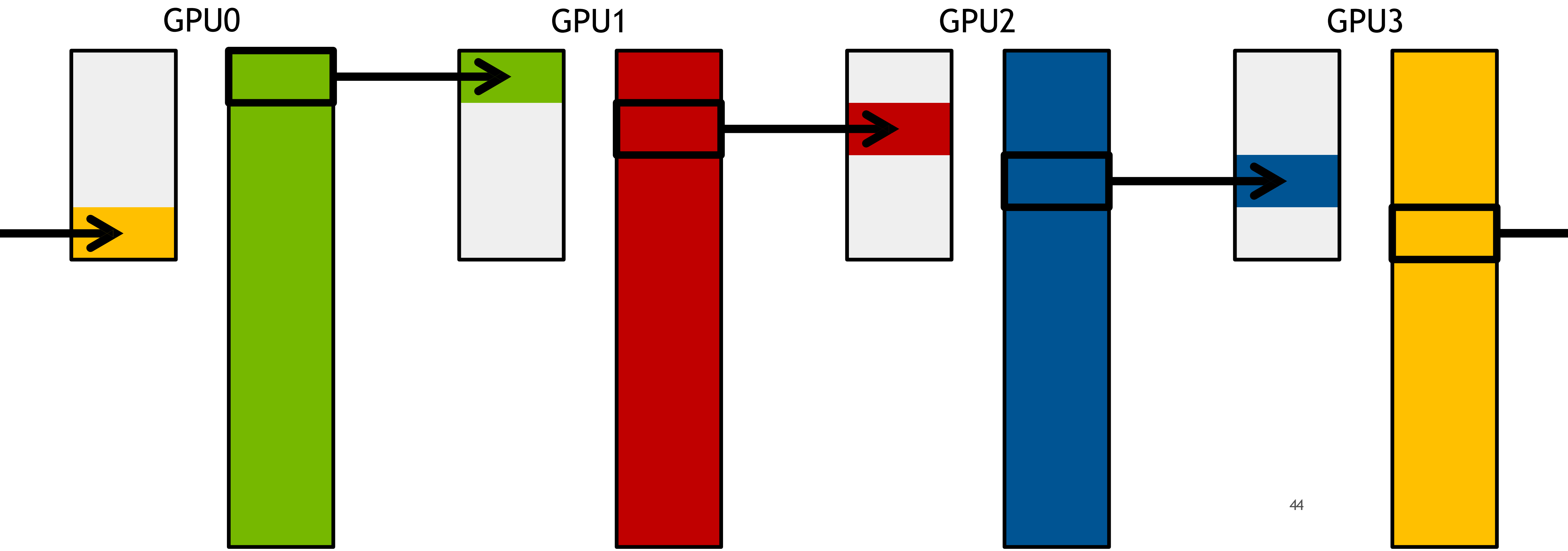


Slides from Nvidia

ALL-REDUCE

with unidirectional ring

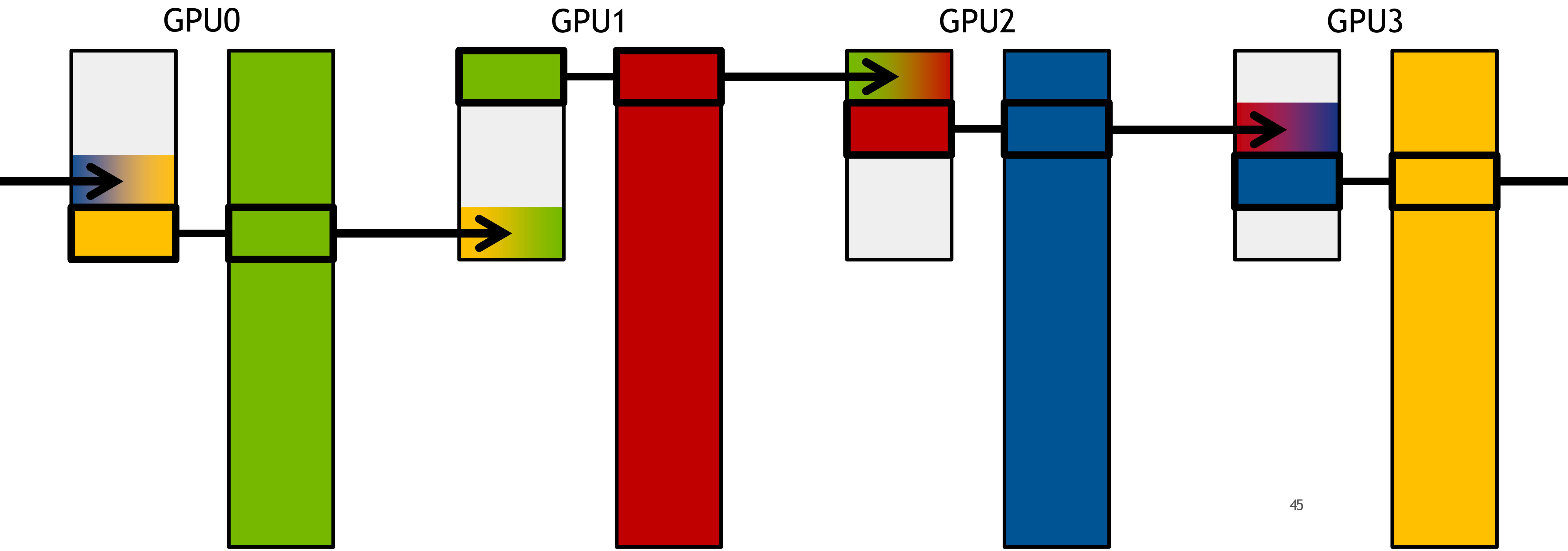
Chunk: 1
Step: 1



Slides from Nvidia

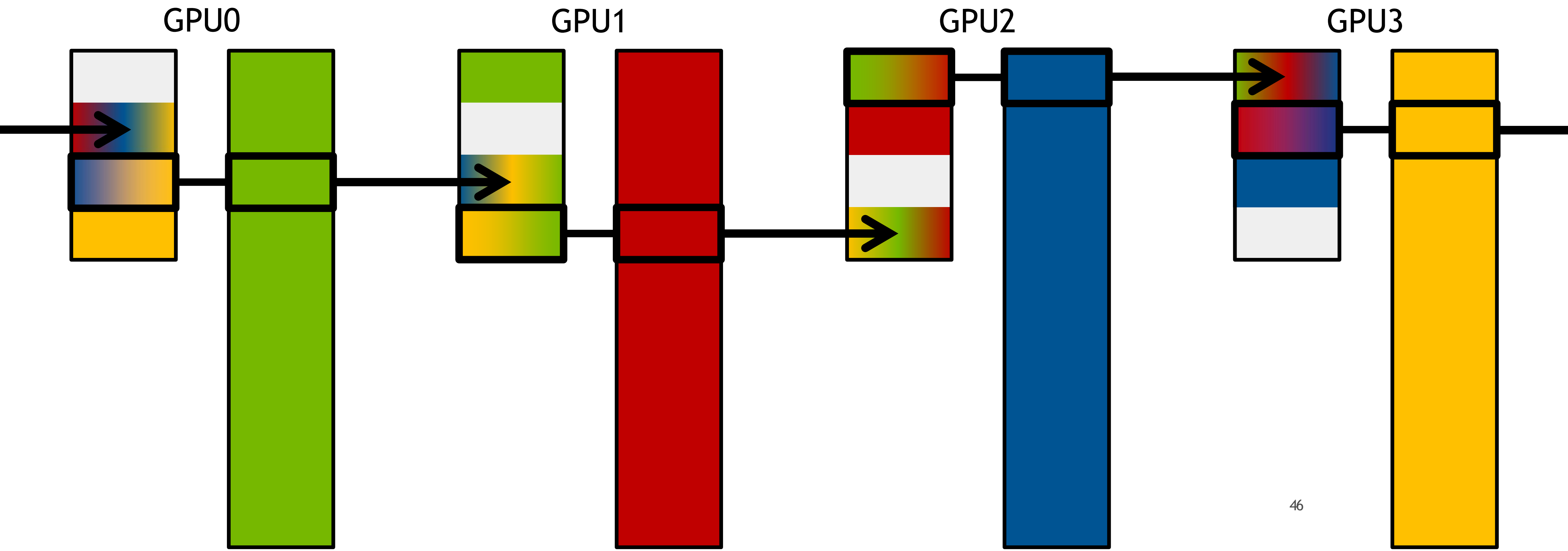
ALL-REDUCE
with unidirectional ring

Chunk: 1
Step: 2



ALL-REDUCE
with unidirectional ring

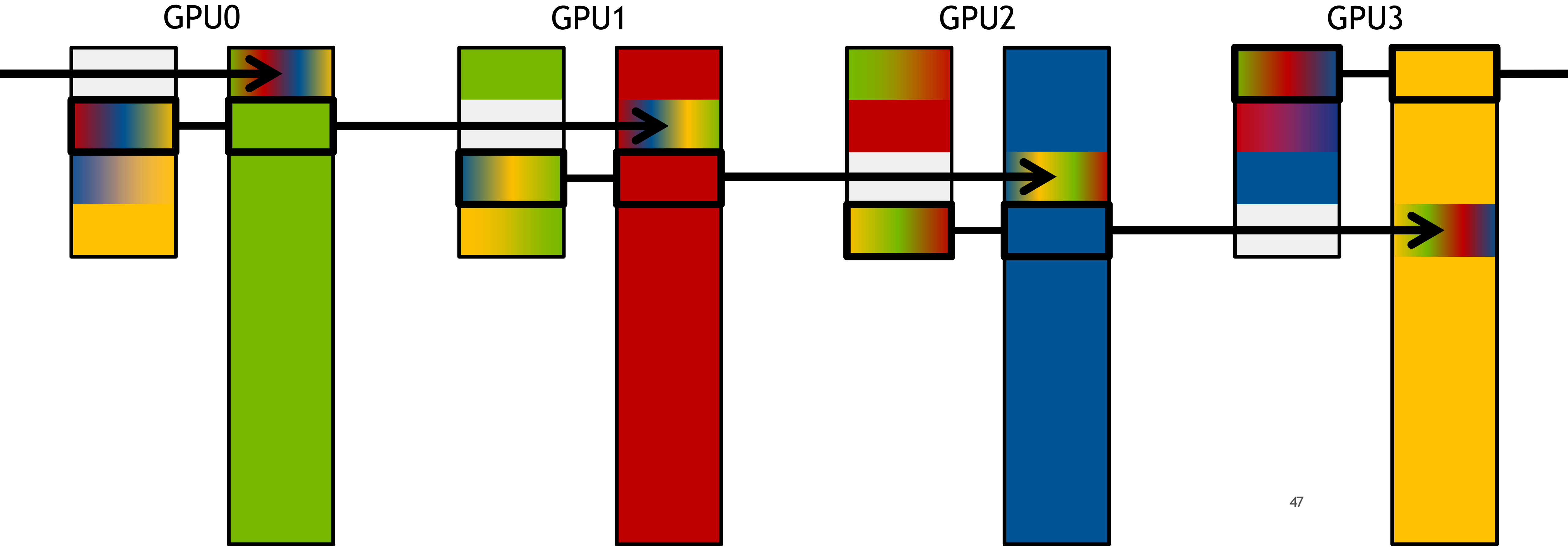
Chunk: 1
Step: 3



Slides from Nvidia

ALL-REDUCE
with unidirectional ring

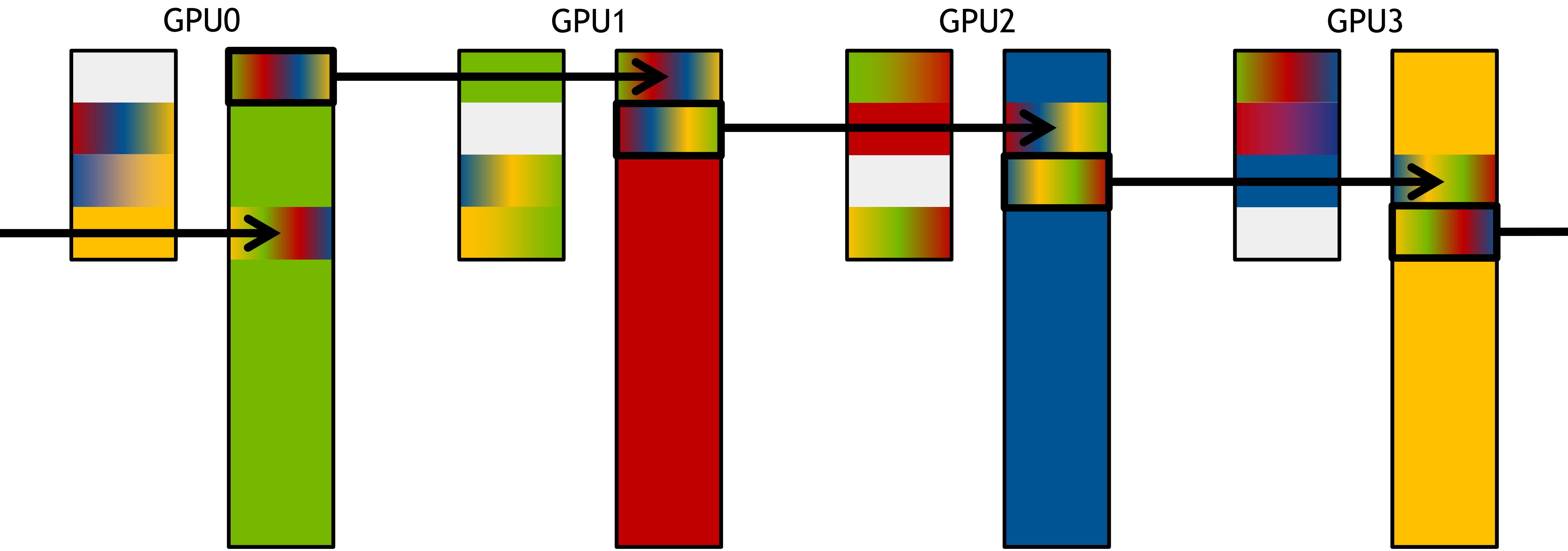
Chunk: 1
Step: 4



ALL-REDUCE

with unidirectional ring

Chunk: 1
Step: 5

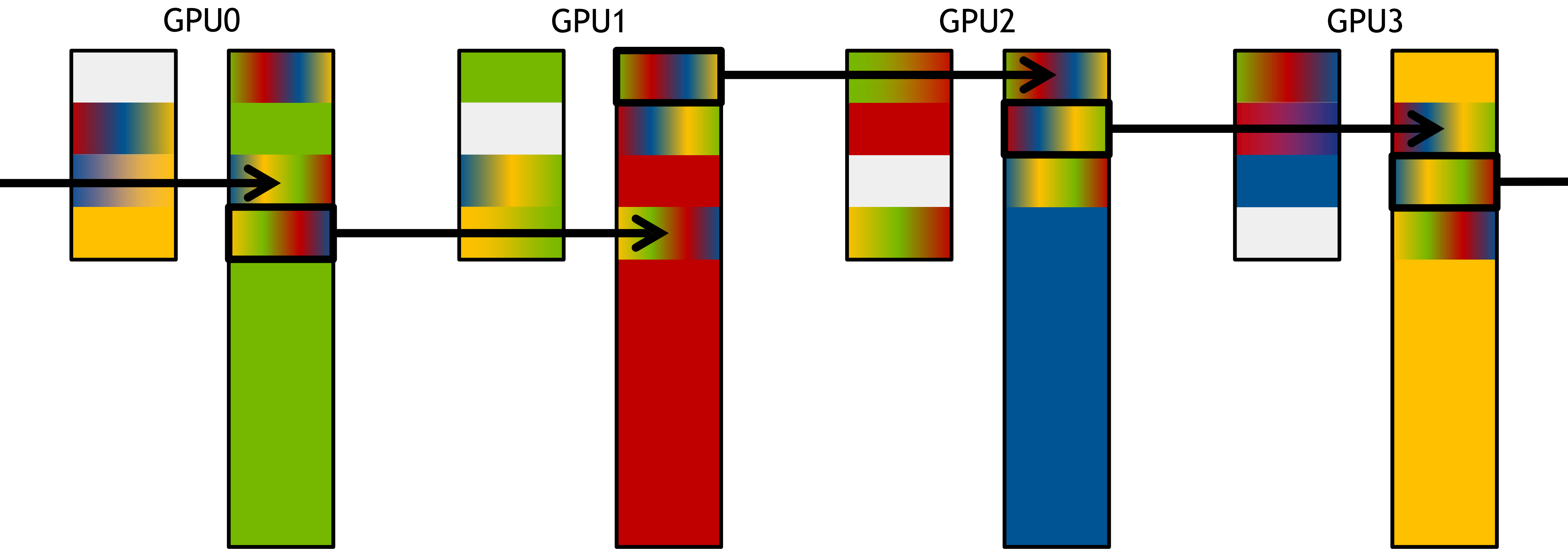


Slides from Nvidia

ALL-REDUCE

with unidirectional ring

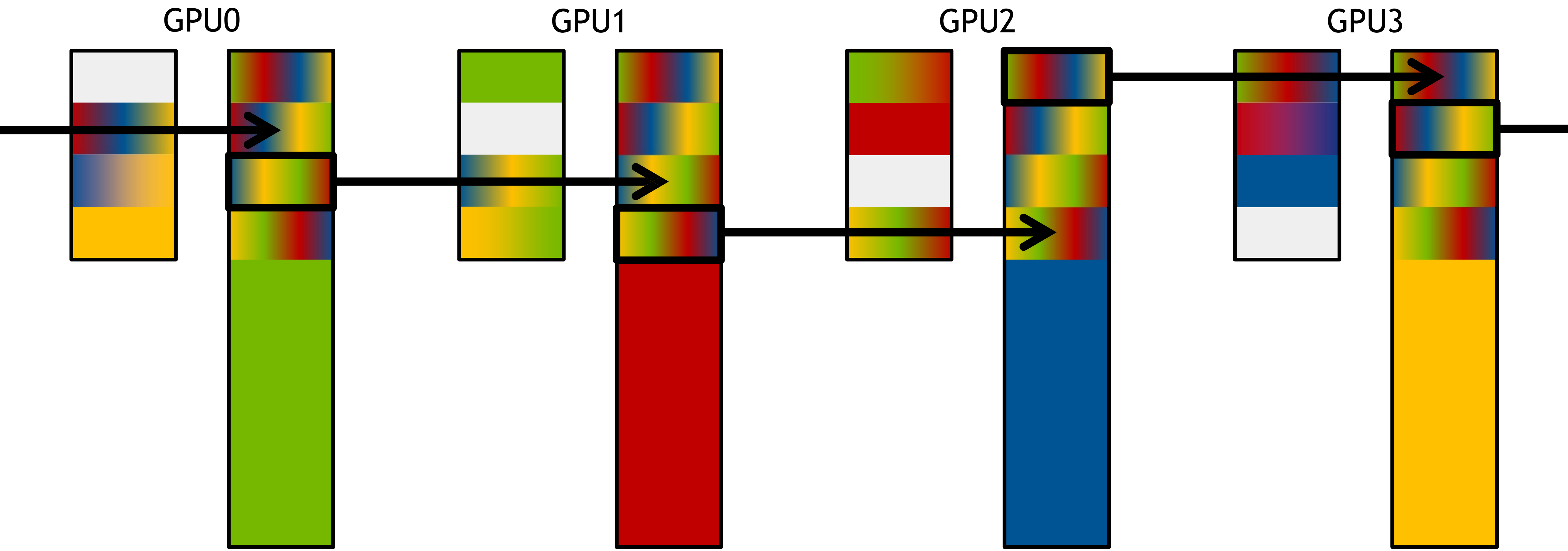
Chunk: 1
Step: 6



ALL-REDUCE

with unidirectional ring

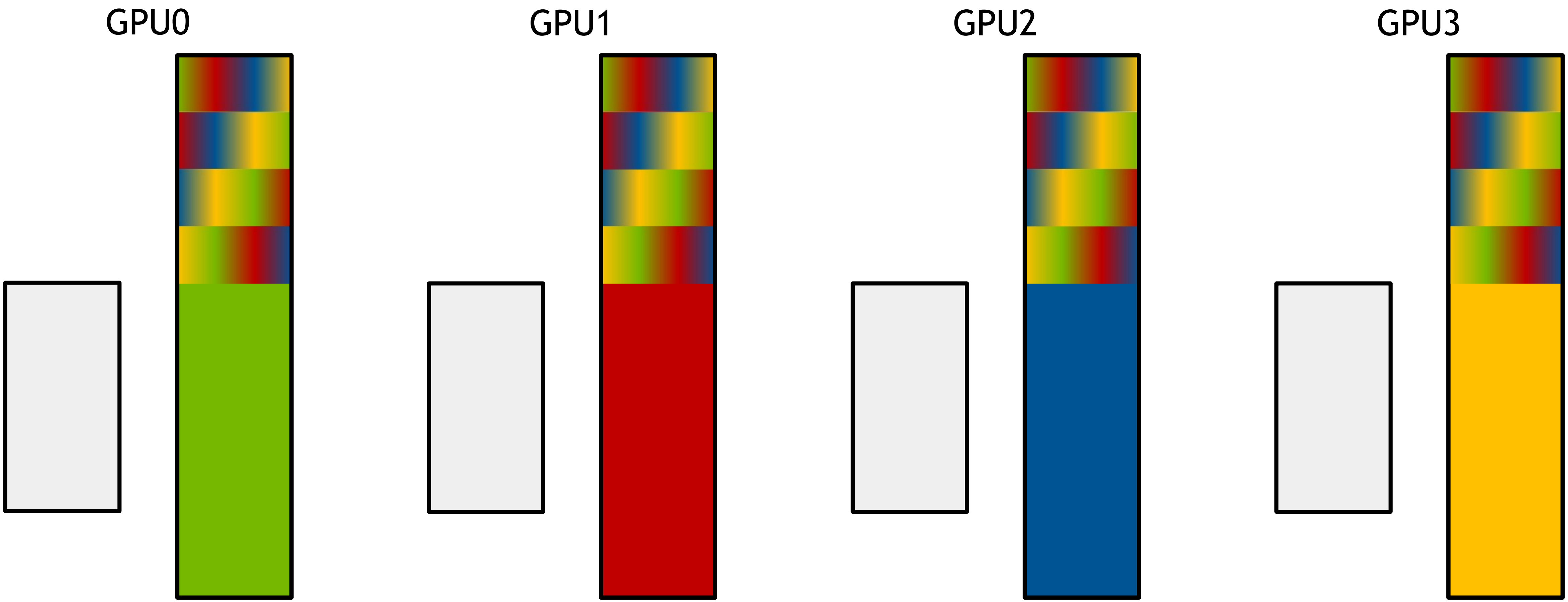
Chunk: 1
Step: 7



ALL-REDUCE

with unidirectional ring

Chunk: 2
Step:

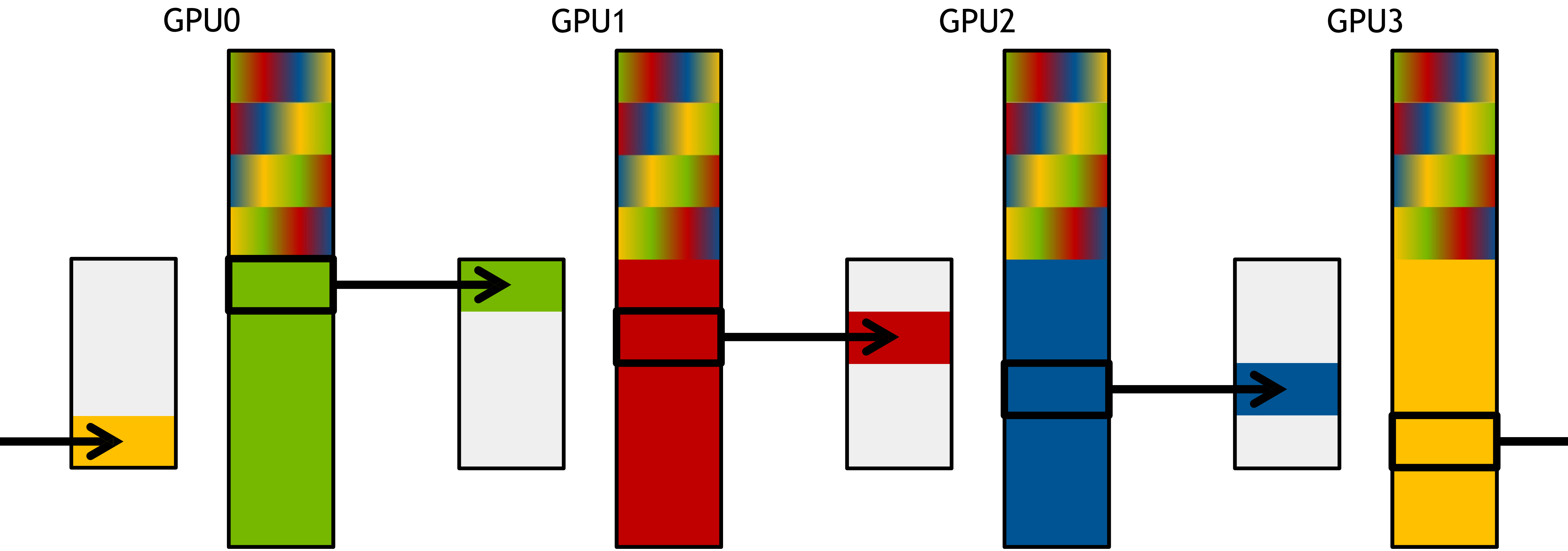


Slides from Nvidia

ALL-REDUCE

with unidirectional ring

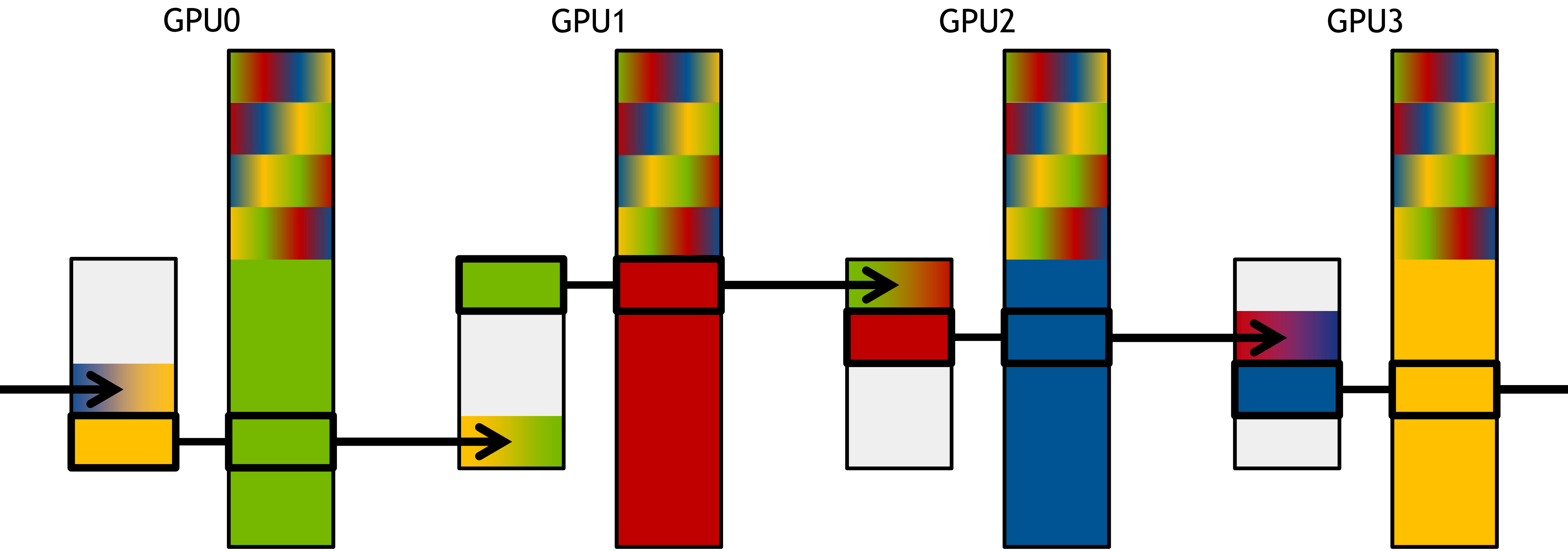
Chunk: 2
Step: 1



ALL-REDUCE

with unidirectional ring

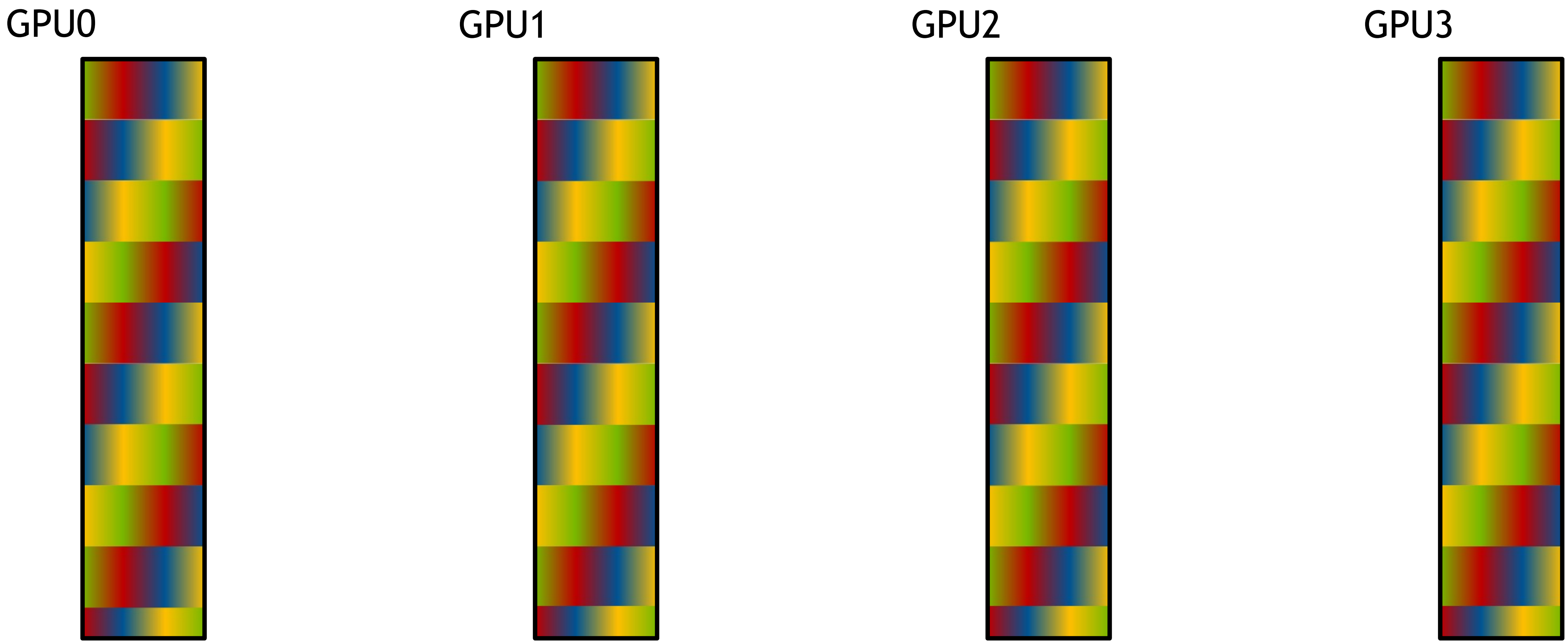
Chunk: 2
Step: 2



ALL-REDUCE

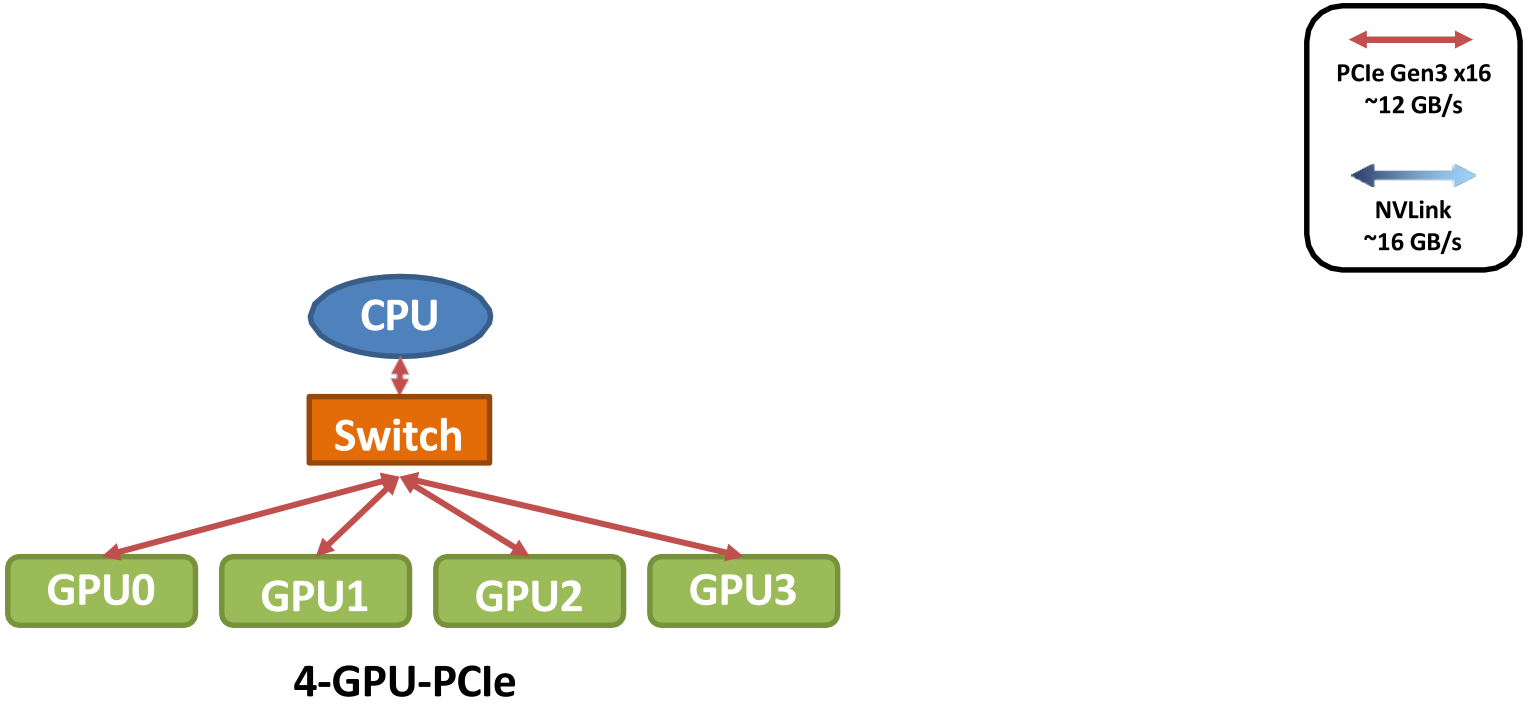
with unidirectional ring

done



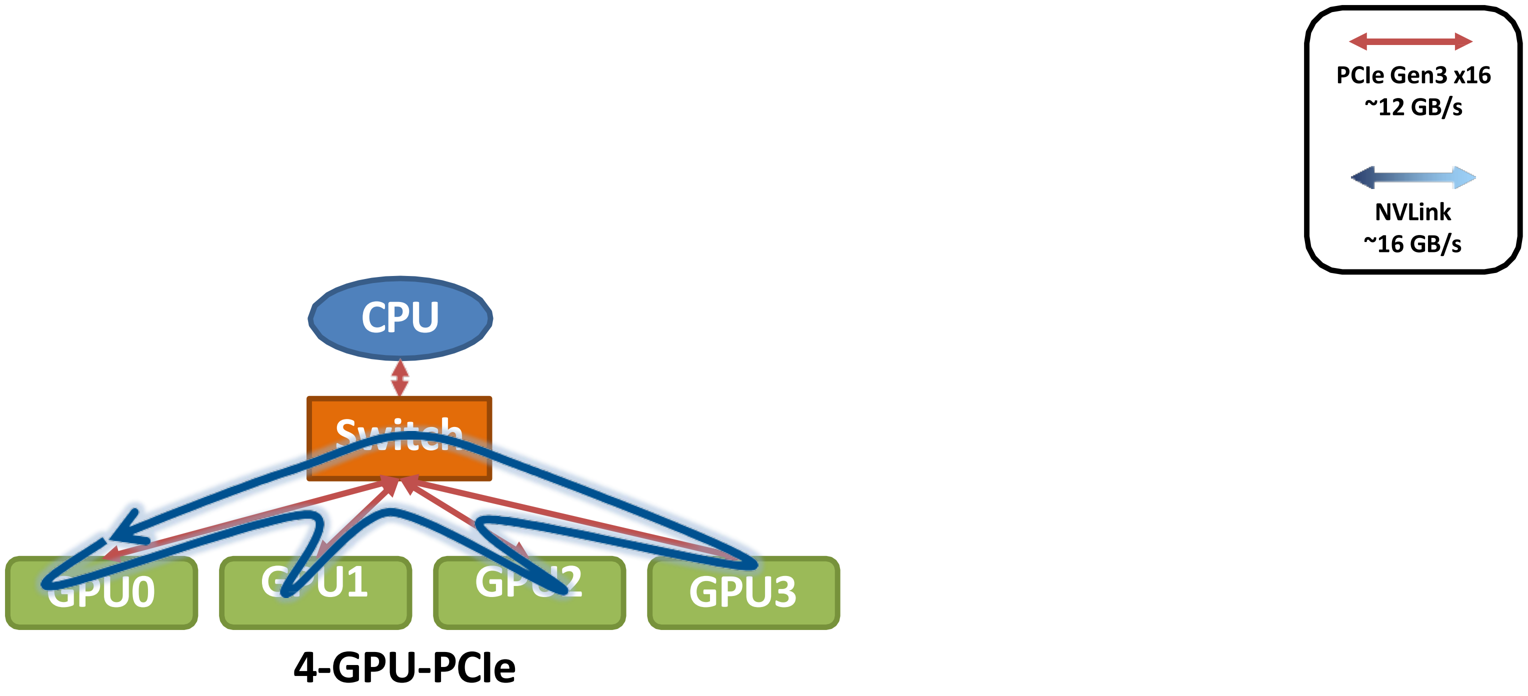
RING-BASED COLLECTIVES

A primer



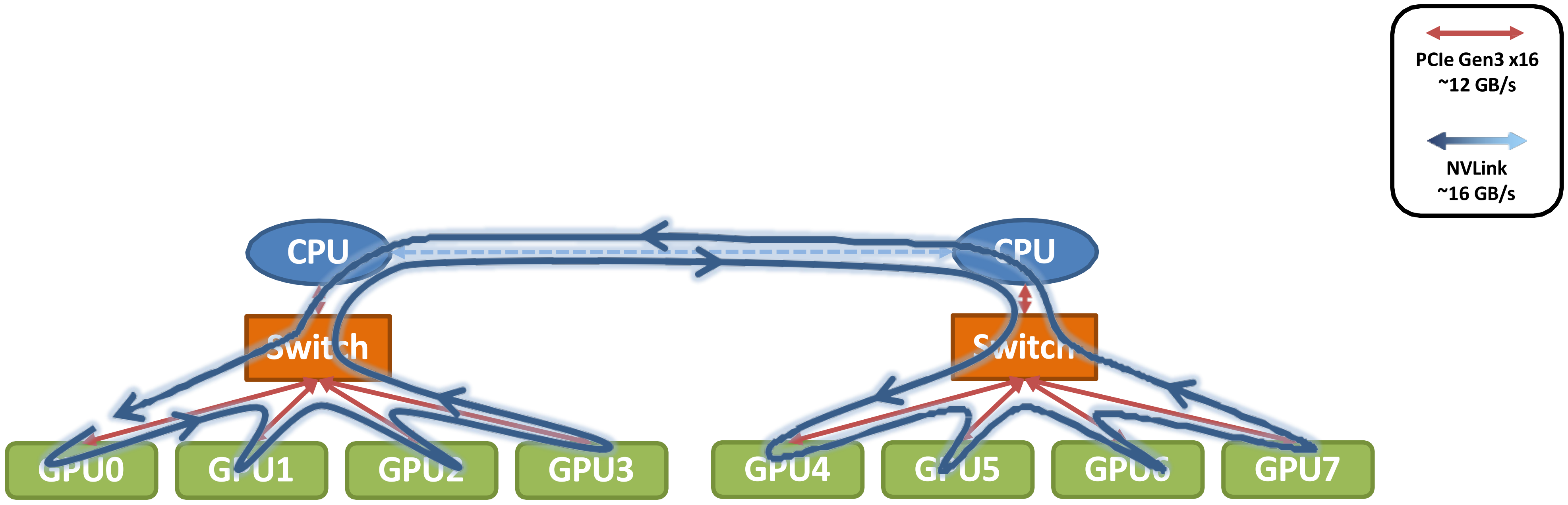
RING-BASED COLLECTIVES

A primer



RING-BASED COLLECTIVES

…apply to lots of possible topologies



THE CHALLENGE OF COLLECTIVES

Many implementations seen in the wild are suboptimal

Scaling requires efficient communication algorithms and careful implementation

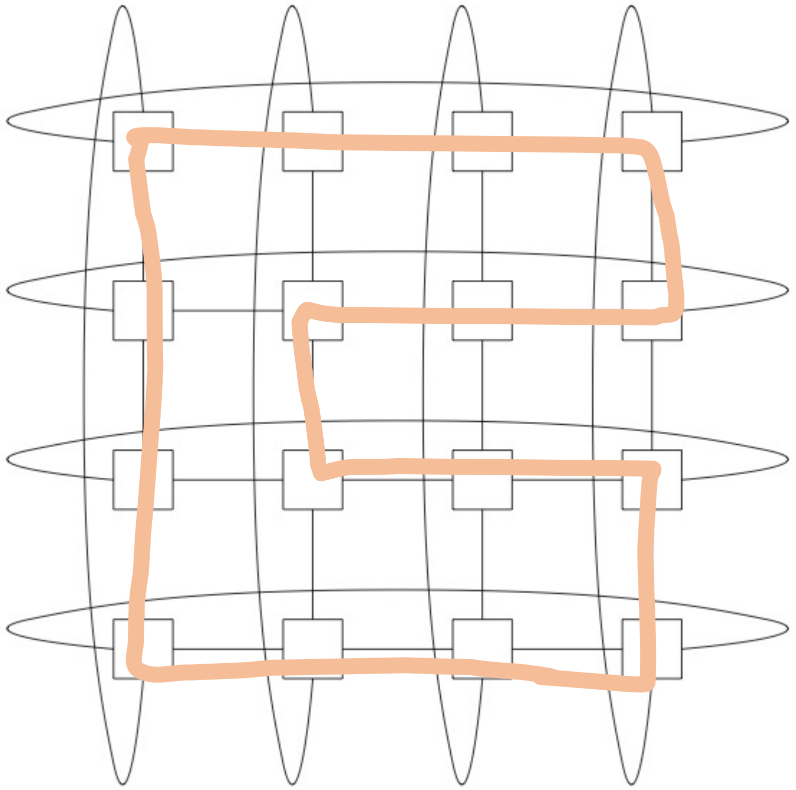
Communication algorithms are implementation and topology-dependent

Topologies can be complex – not every system is a fat tree

Most collectives amenable to bandwidth-optimal implementation on rings, and many topologies can be interpreted as one or more rings [P. Patarasuk and X. Yuan]

Challenge: Collectives on Multi-dimensional networks

Ring Algorithm



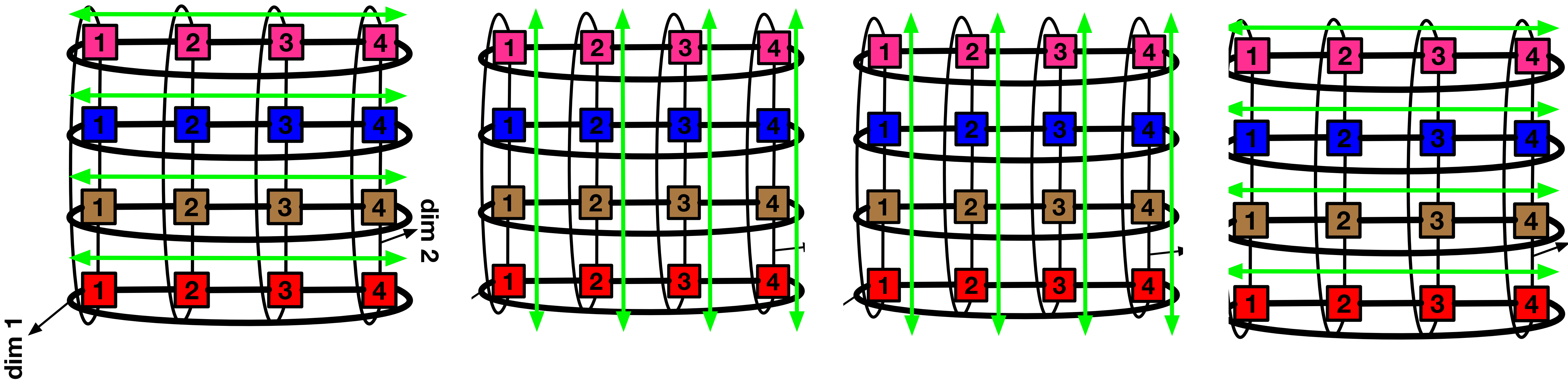
Physical Topology: 2D Torus

Congestion? No

Link Utilization? ~50%

Network Underutilization!!

Hierarchical Collective Algorithms



1. Reduce-Scatter on dim 1 (rows)
2. Reduce-Scatter on dim 2 (cols)
3. All-Gather on dim 2 (cols)
4. All-Gather on dim 1 (rows)

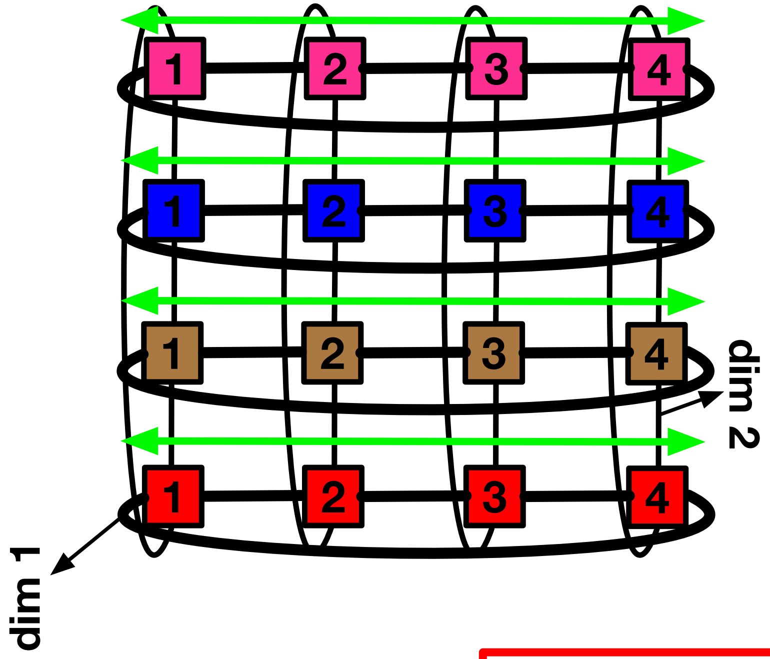
Congestion? No

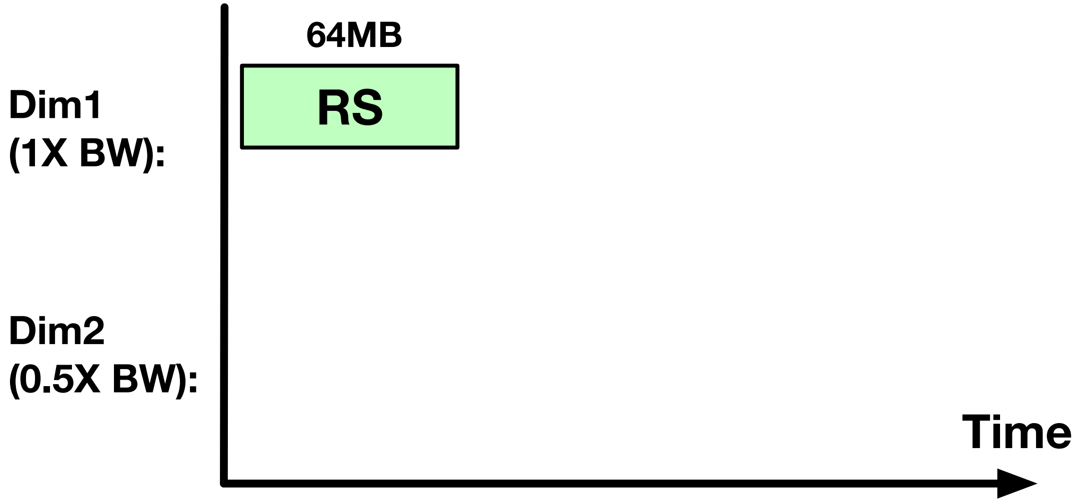
Link Utilization? All links utilized equally over time. But at any instant, only ~50% of the links utilized!

This is where Collective "Scheduling" comes in

Scheduling Collectives (Naïve)

- A 2D torus with: BW(dim1) = 2*BW(dim2)
- Hierarchical All-Reduce on a 64 MB data chunk

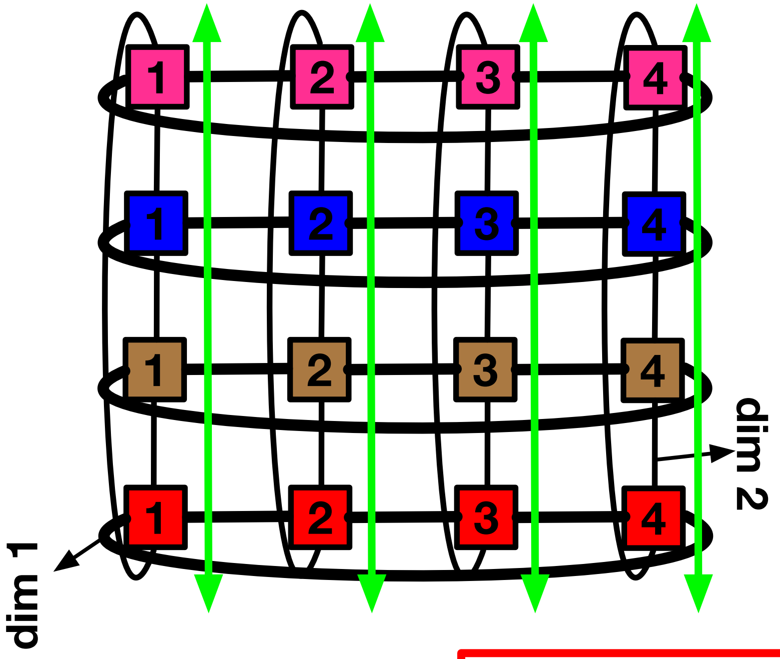


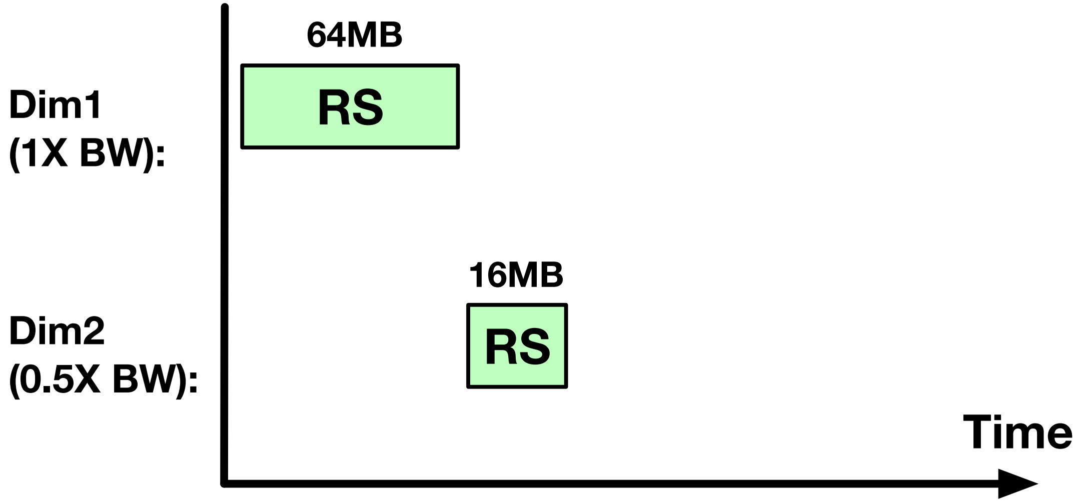


Stage 1) Reduce-Scatter (RS) on dim 1 (rows)

Scheduling Collectives (Naïve)

- A 2D torus with: BW(dim1) = 2*BW(dim2)
- Hierarchical All-Reduce on a 64 MB data chunk

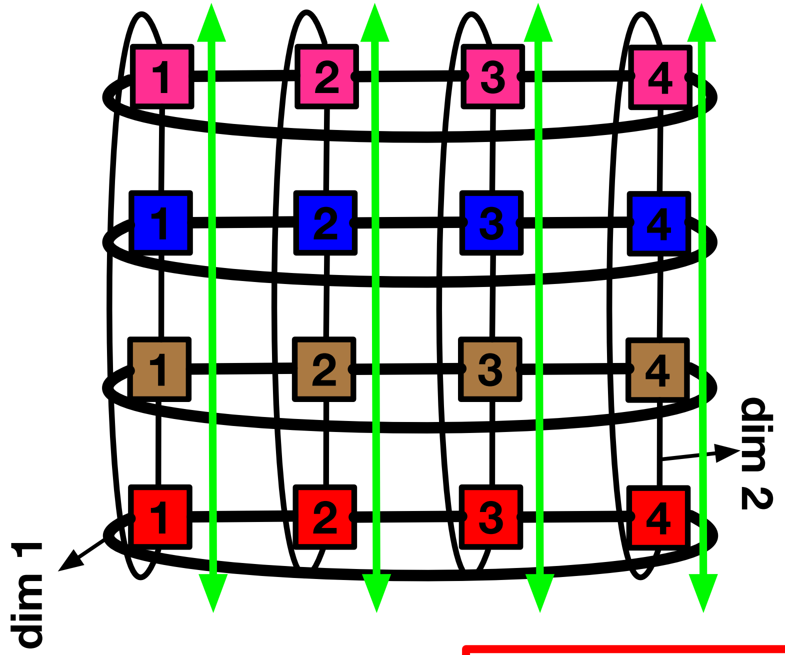


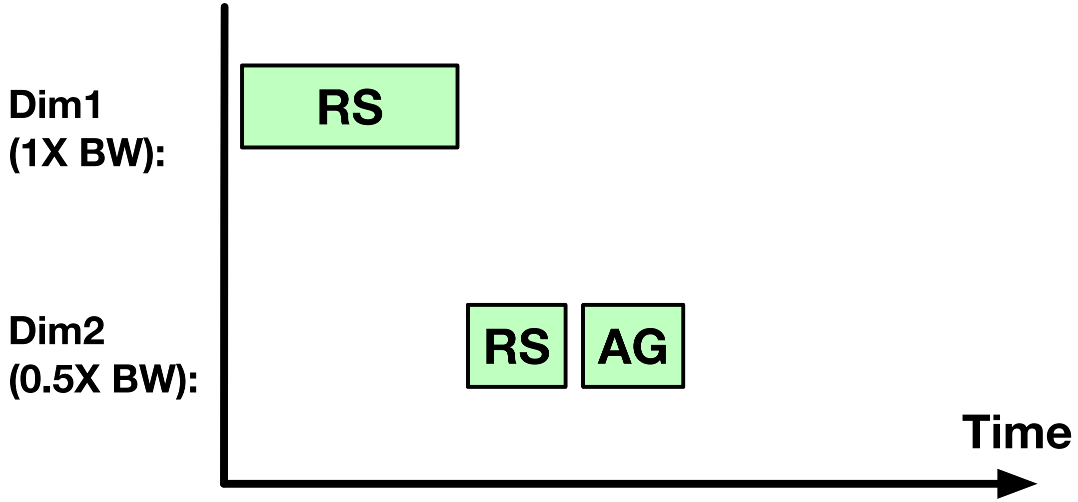


Stage 2) Reduce-Scatter (RS) on dim 2 (cols)

Scheduling Collectives (Naïve)

- A 2D torus with: BW(dim1) = 2*BW(dim2)
- Hierarchical All-Reduce on a 64 MB data chunk

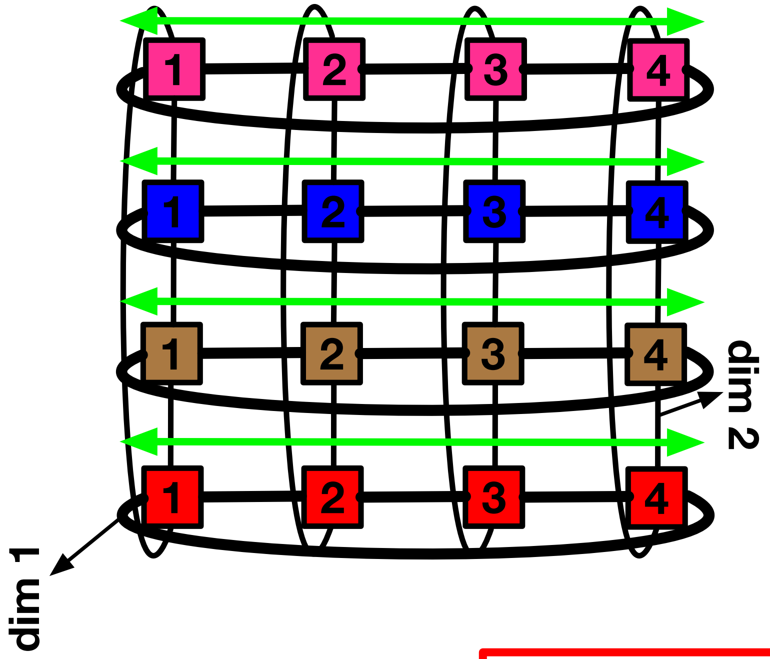


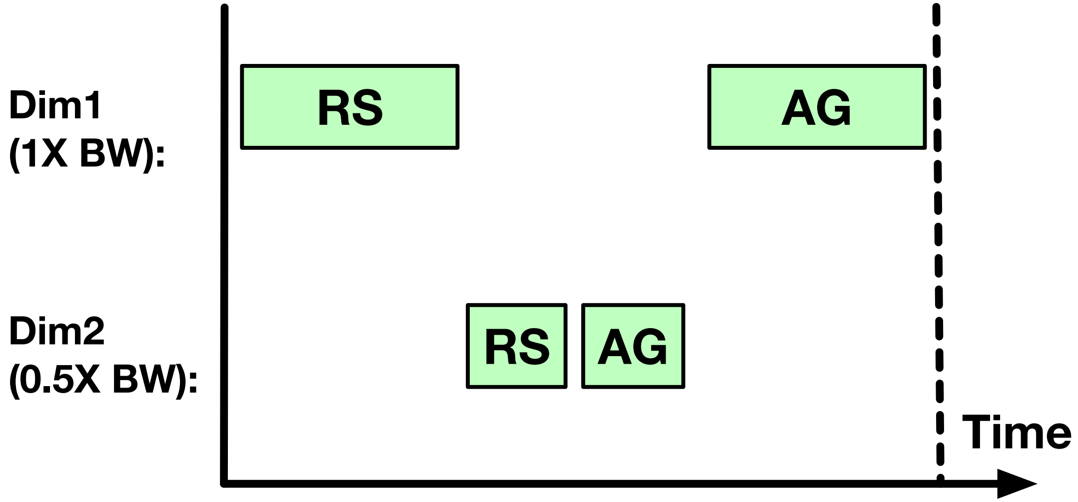


Stage 3) All-Gather (AG) on dim 2 (cols)

Scheduling Collectives (Naïve)

- A 2D torus with: BW(dim1) = 2*BW(dim2)
- Hierarchical All-Reduce on a 64 MB data chunk

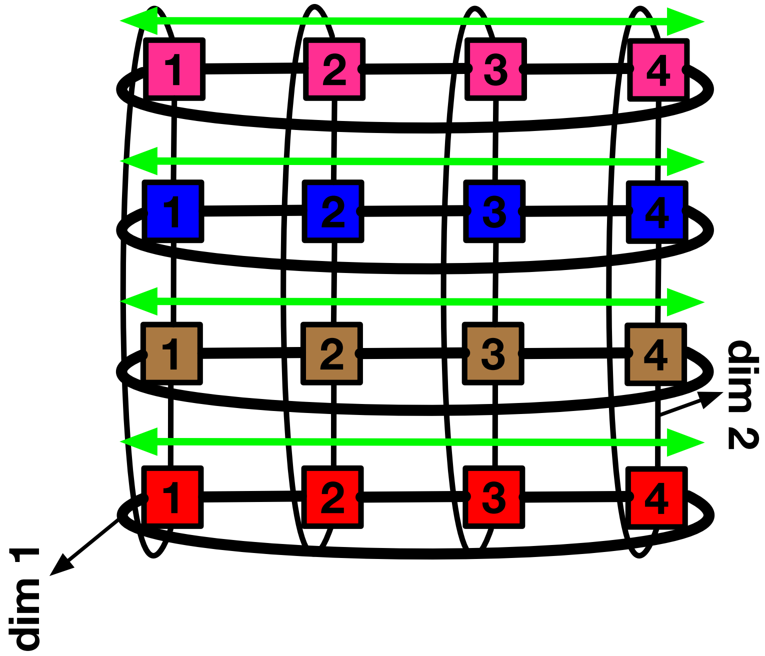


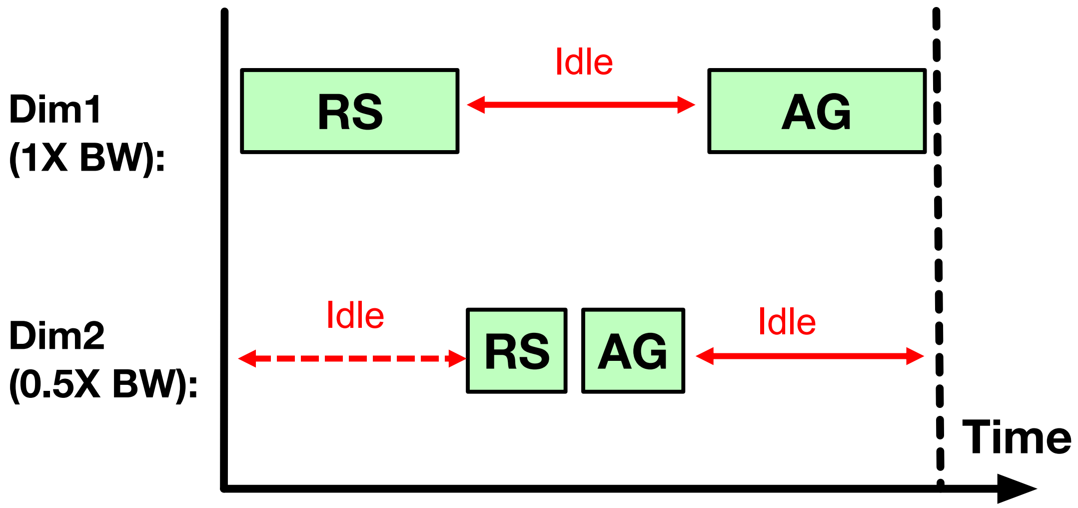


Stage 4) All-Gather (AG) on dim 1 (rows)

Scheduling Collectives (Naïve)

- A 2D torus with: BW(dim1) = 2*BW(dim2)
- Hierarchical All-Reduce on a 64 MB data chunk



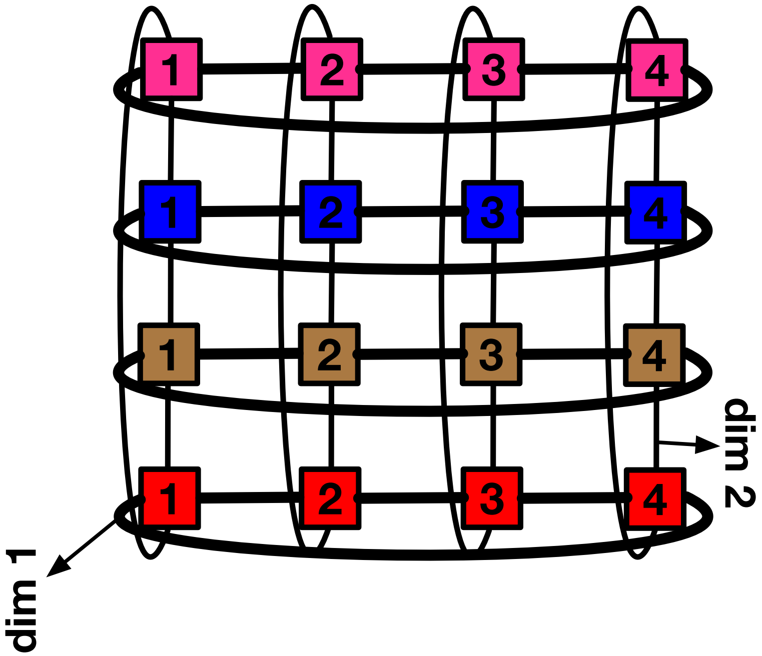


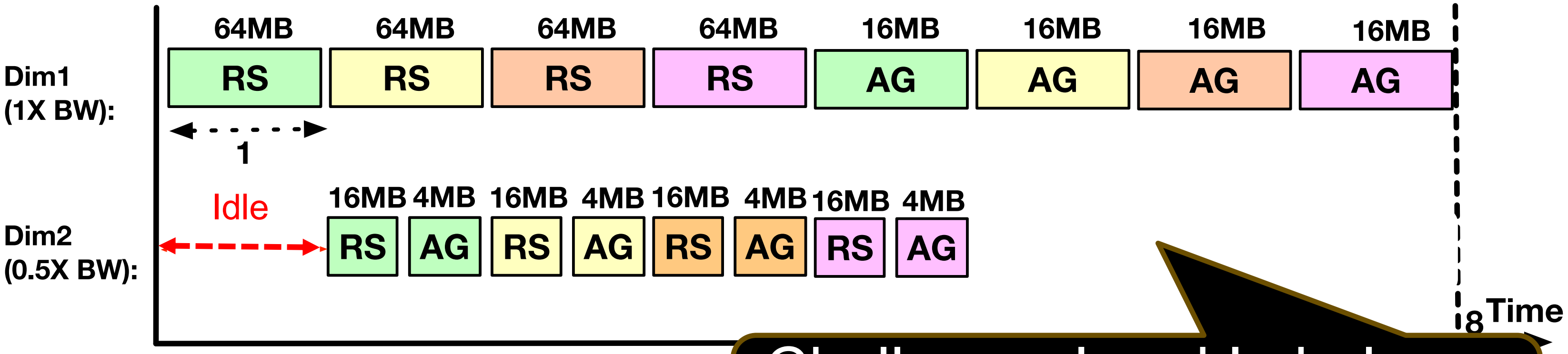
Solution: Pipelined Scheduling of multiple independent chunks

Scheduling Collectives (Baseline)

- A 2D torus with: BW(dim1) = 2*BW(dim2)
- Hierarchical All-Reduce on a 64 MB data chunk

Pipeline All-Reduce across multiple chunks to utilize all dims





Challenge: Load Imbalance across network dimensions

S. Rashidi, W. Won, S. Srinivasan, S. Sridharan, T. Krishna, " ***Themis: A Network Bandwidth-Aware Collective Scheduling Policy for Distributed Training of DL Models".*** ISCA 2022.

Summary of Hierarchical All-Reduce Challenge

- Data is broken into multiple chunks and chunks go to the Reduce-Scatter/All-Gather pipeline stages in order

- **Load Imbalance Challenge**
  - Chunk size changes across stages → Algorithm constraint
  - Static scheduling of chunks → Baseline scheduling
  - Hybrid Bandwidths across dimensions → Technology constraint

S. Rashidi, W. Won, S. Srinivasan, S. Sridharan, T. Krishna, " **Themis: A Network Bandwidth-Aware Collective Scheduling Policy for Distributed Training of DL Models"**. ISCA 2022.