CS168, Fall 2024

Beyond Client-Server (part 2)
Collective Communication

Sylvia Ratnasamy

Recall

- Last time: apps based on group vs. client-server interactions
  - Multicast as a communication primitive
  - Implemented in the network (L3) or as an overlay (L7)
  - Our context: the Internet at large

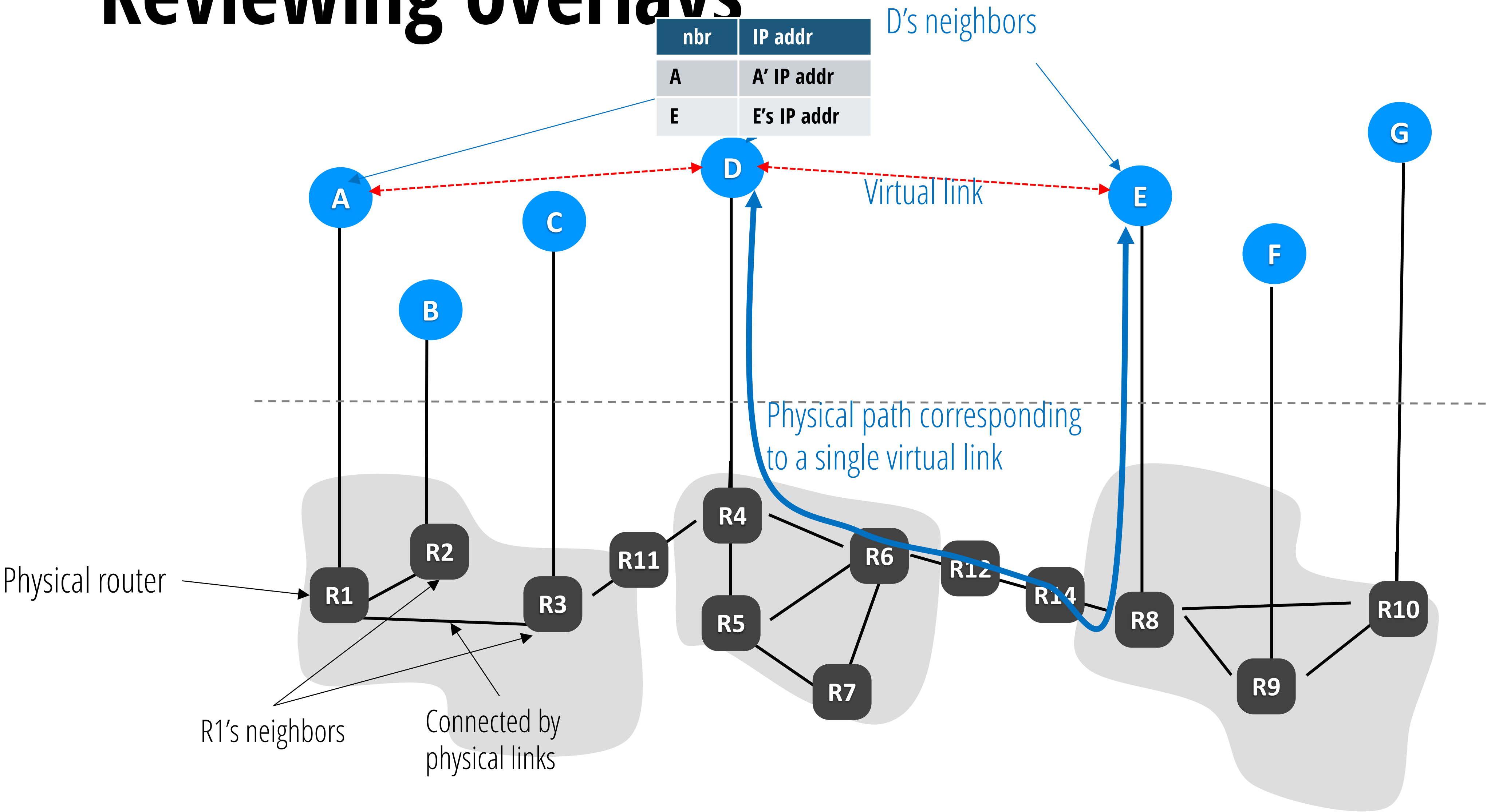
- Today: distributed training as our app
  - Also based on groups of hosts that communicate
  - But very different context: a single datacenter, different goals, *etc.*
  - Leading to new communication primitives: collectives

Outline

- Quick review: Overlays
- Context on distributed training: app and infrastructure
- Communication collectives: definitions and the function they provide
- How collectives are implemented
  - Focus on one collective: AllReduce
  - Implementation at overlay and underlay level

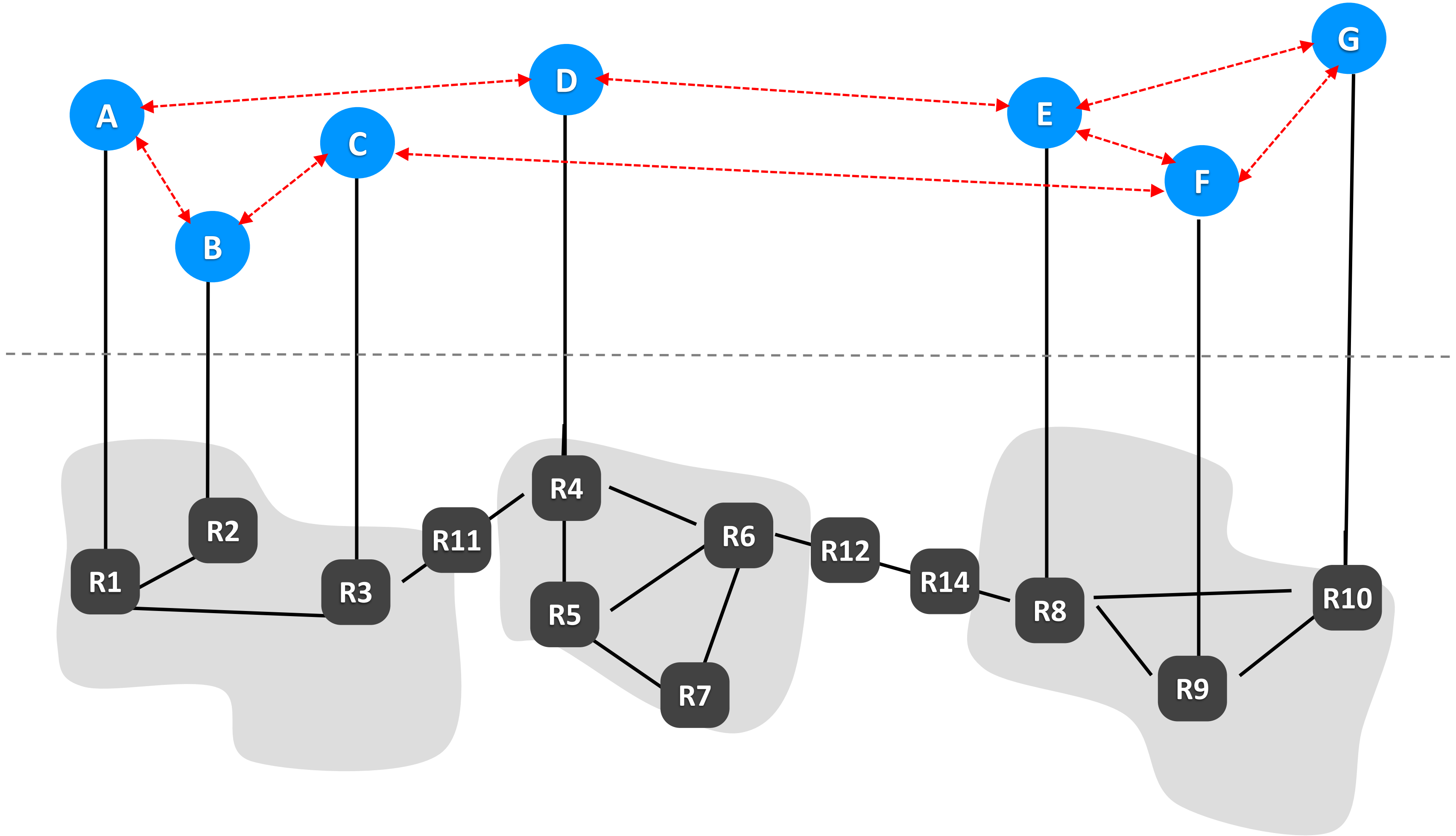
Reviewing overlays

nbr	IP addr
A	A' IP addr
E	E's IP addr



A virtual network (the "overlay") on top of an underlying physical network (the "underlay")

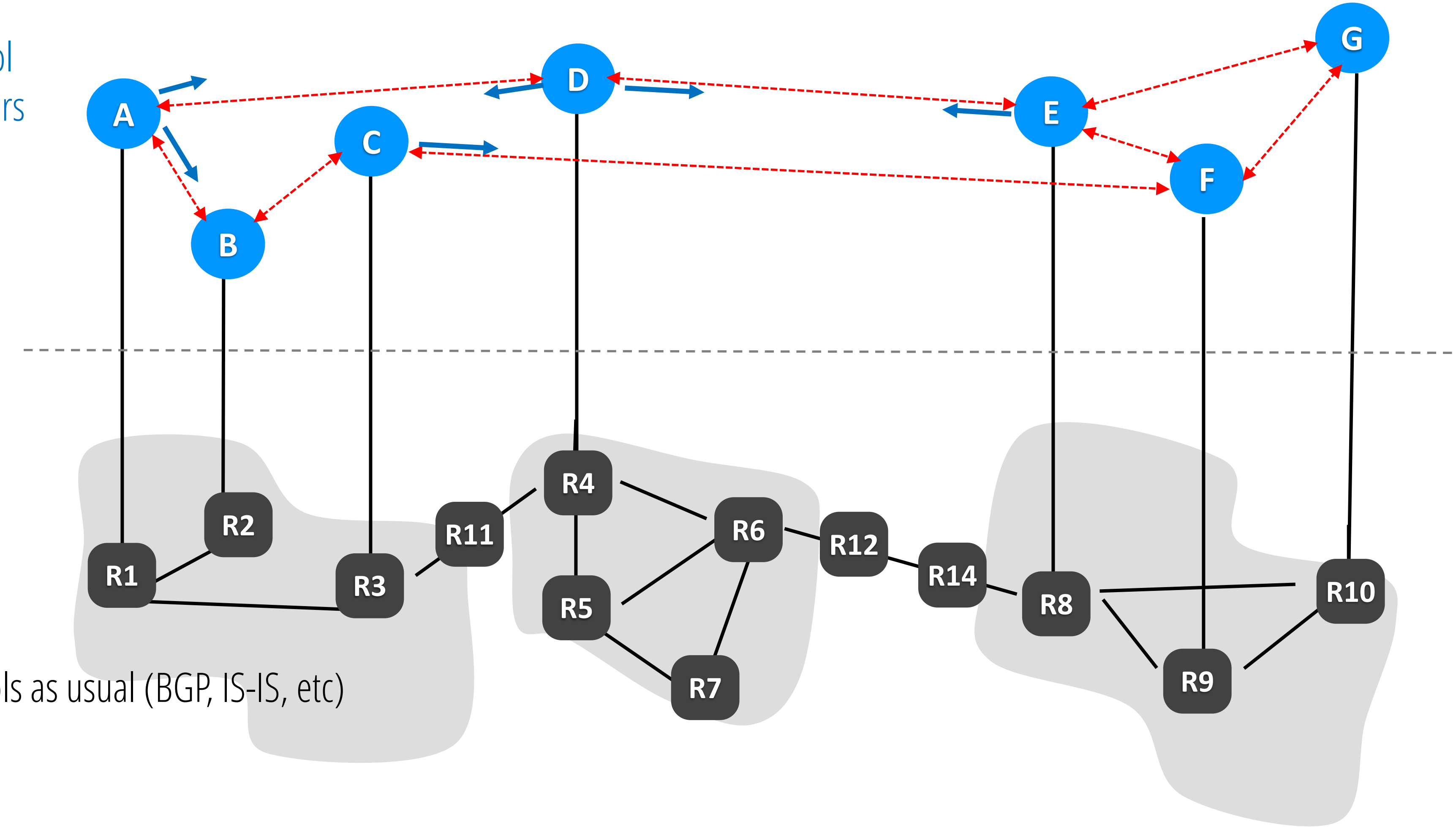
Reviewing overlays



A virtual network (the "overlay") on top of an underlying physical network (the "underlay")

Reviewing overlays

Run a routing protocol between virtual routers

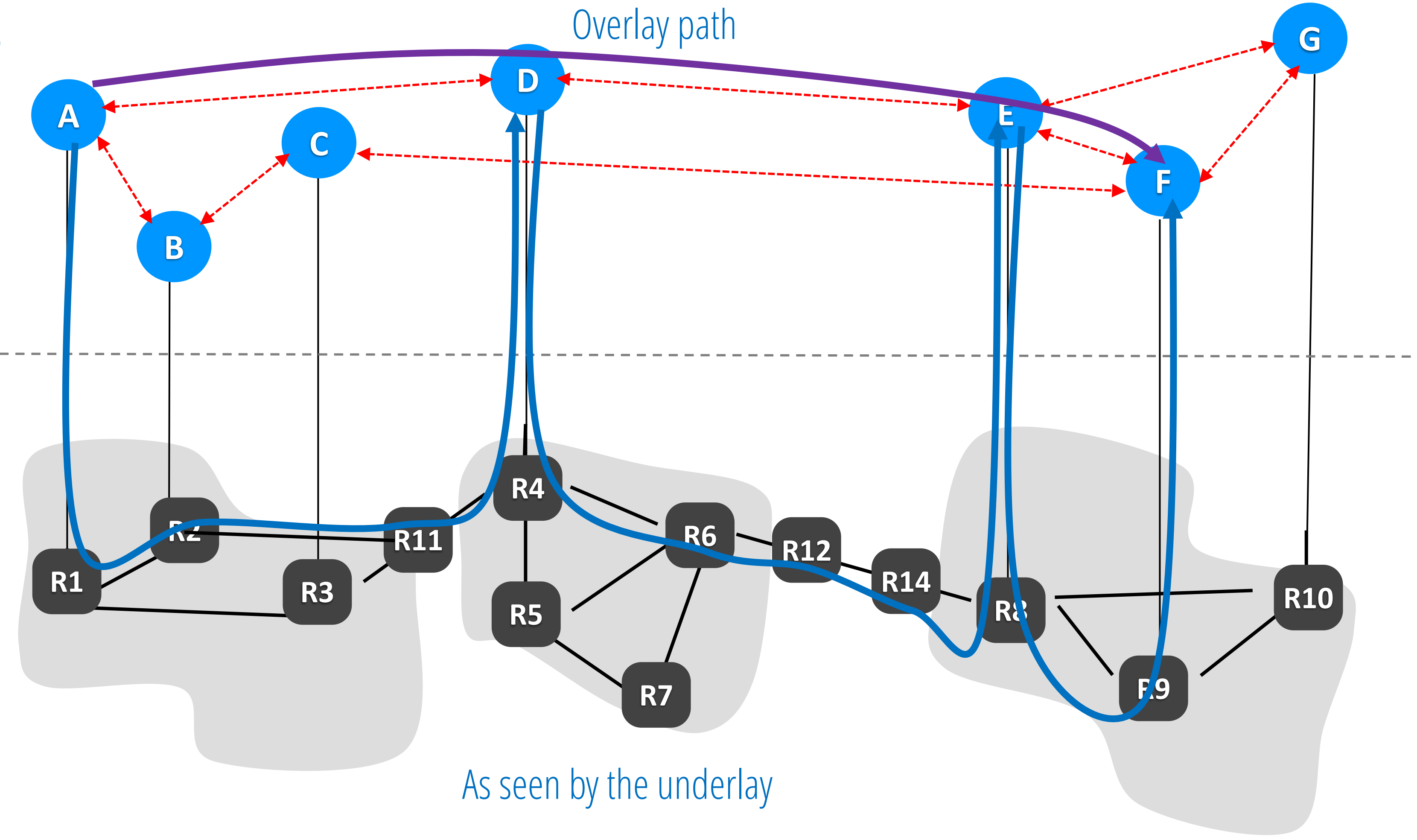


Run routing protocols as usual (BGP, IS-IS, etc)

Reviewing overlays

Overlay routing table

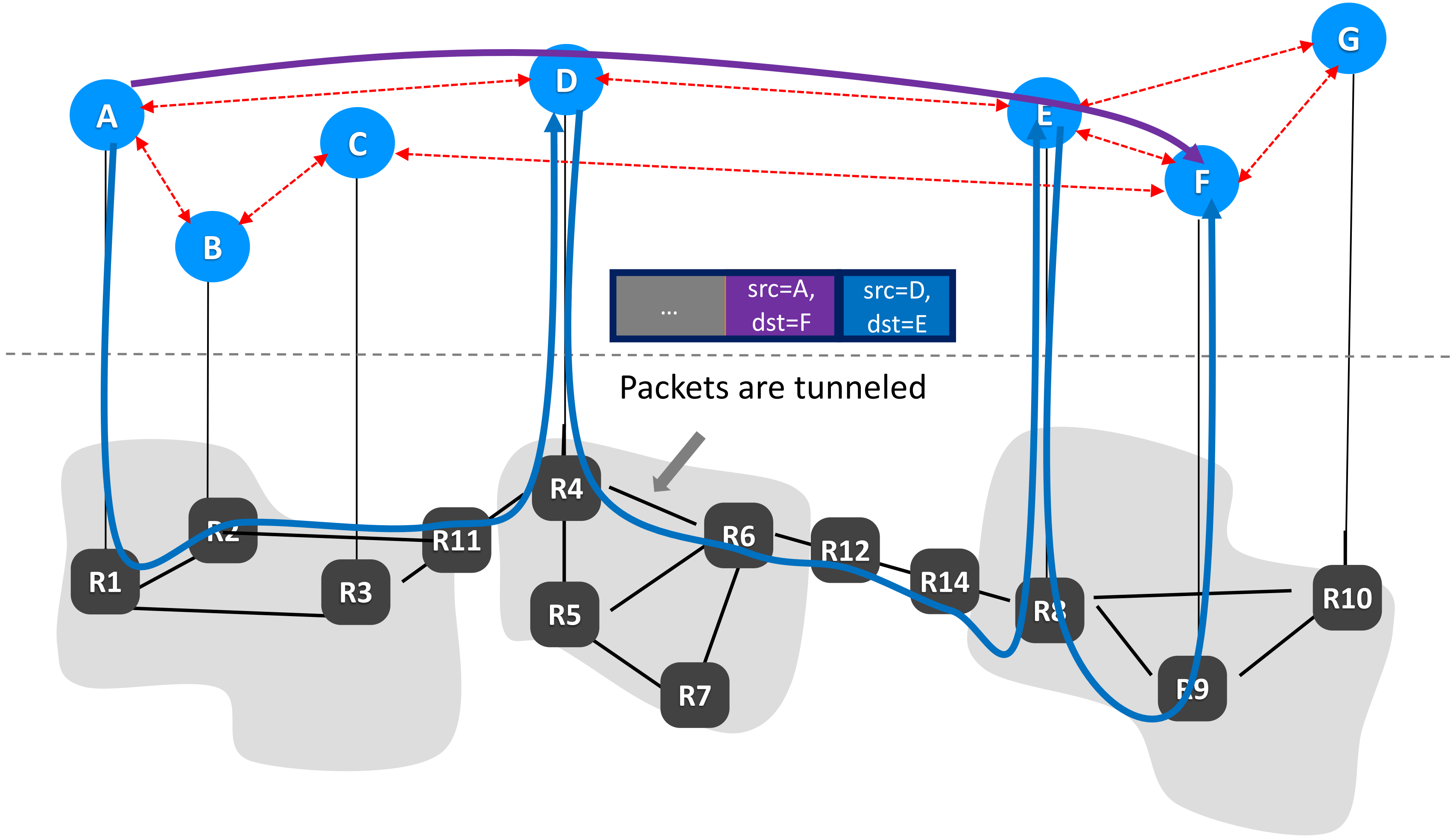
dst	Next-hop
G	D
C	B
F	D
…	…



Underlay routing tables

dst	Next-hop
prefix1	R2
prefiix2	R3
prefix3	R2
…	…

Reviewing overlays



Outline

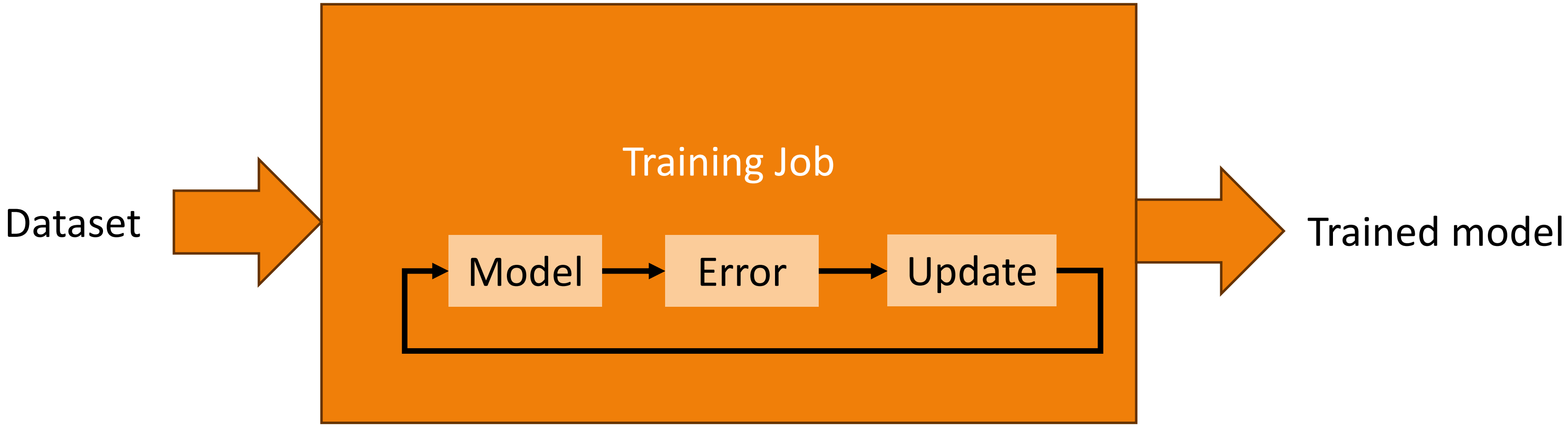
- Quick review: Overlays

- Context on distributed training: app and infrastructure

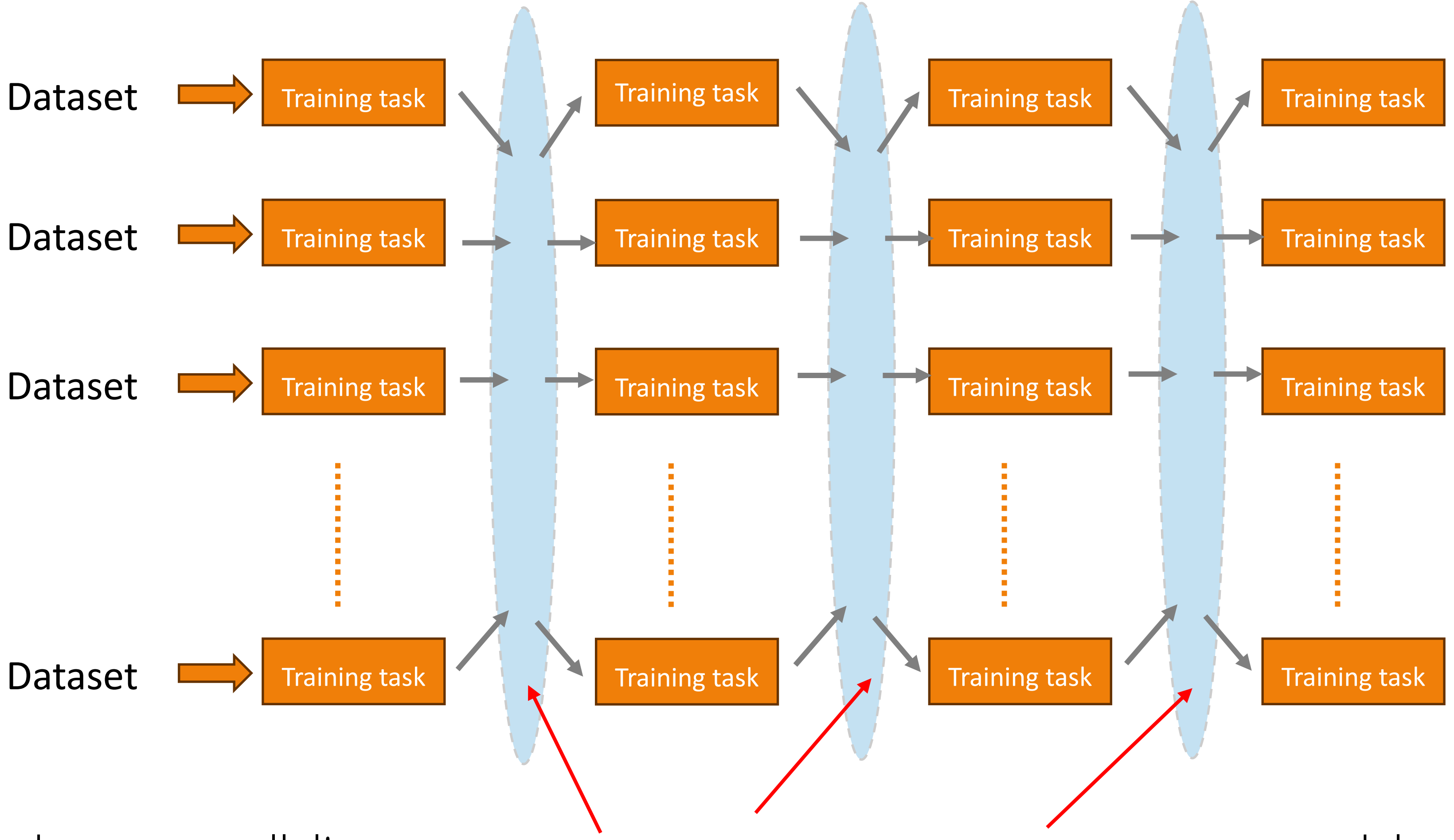
- Communication collectives: definitions and the function they provide

- How collectives are implemented
  - Focus on one collective: AllReduce
  - Implementation at overlay and underlay level

Training: 10,000 ft. view (for CS168)



Distributed Training: 10,000 ft. view (for CS168)



Many options for how we paralleliz ... ıta, model, pipeline, hybrid, *etc.*)

Collective Communication

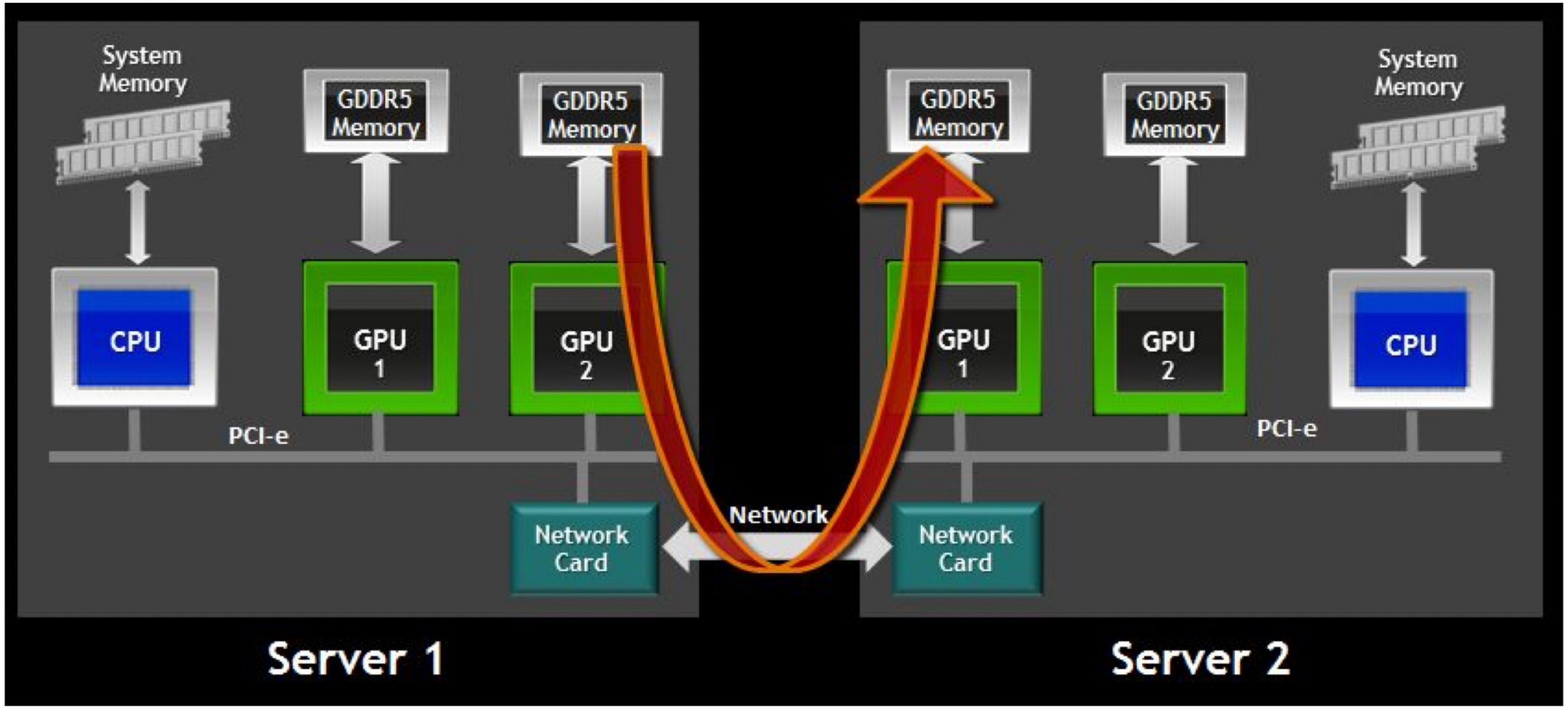
- A group of nodes that exchange data in a coordinated manner as part of a group computation
- Nature of this exchange depends on the details of the training and its parallelization
- Distilled into a foundational set of communication patterns referred to as "collectives"
- This lecture: **what** are these collectives and h**ow are they implemented** in the network
  - We won't get into **why** training/parallelization leads to these particular patterns

Distributed Training Infrastructure

- Nodes are specialized Graphics Processing Units (GPUs) or Tensor Processing Units (TPUs)
  - Typical scale: few 100s up to tens of 1000s

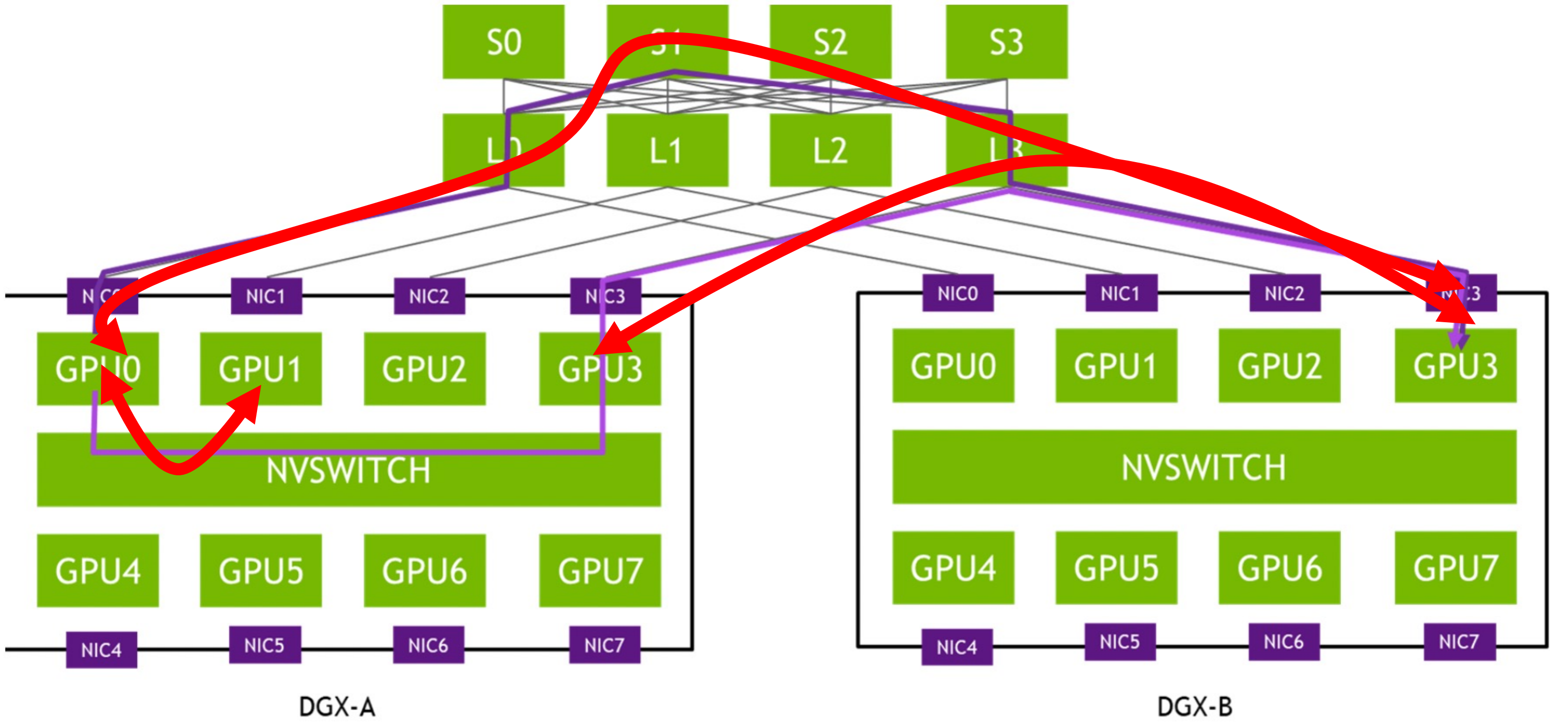
- Typically interconnected by a **dedicated datacenter-like** network
  - Dedicated → not shared with other jobs/apps
  - Datacenter-like →
    - Physically close
    - Regular topology (Clos, Torus)
    - Homogeneous
    - High bandwidth

Distributed Training Infrastructure



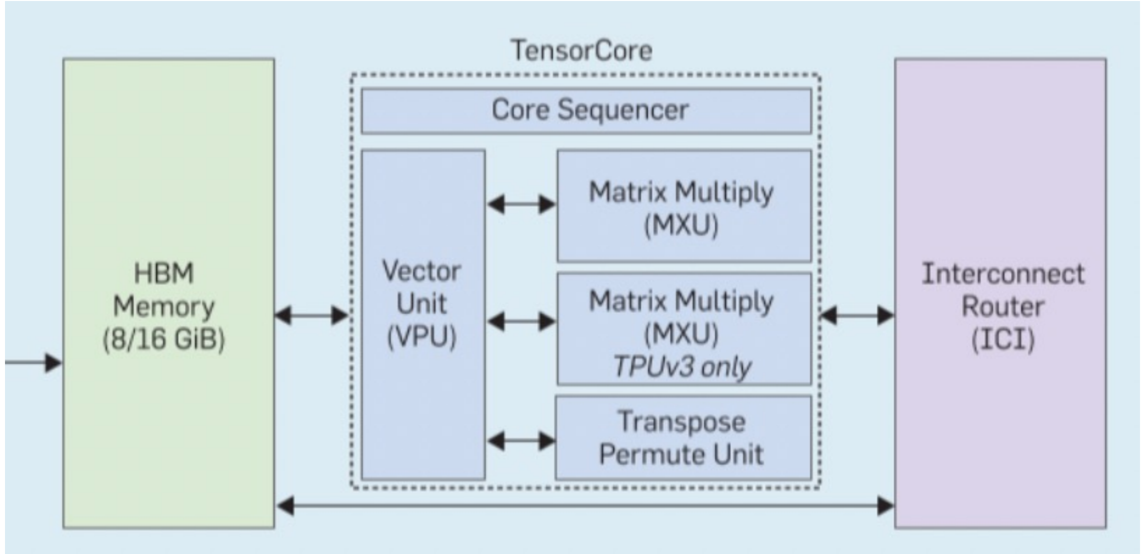
Source: Nvidia

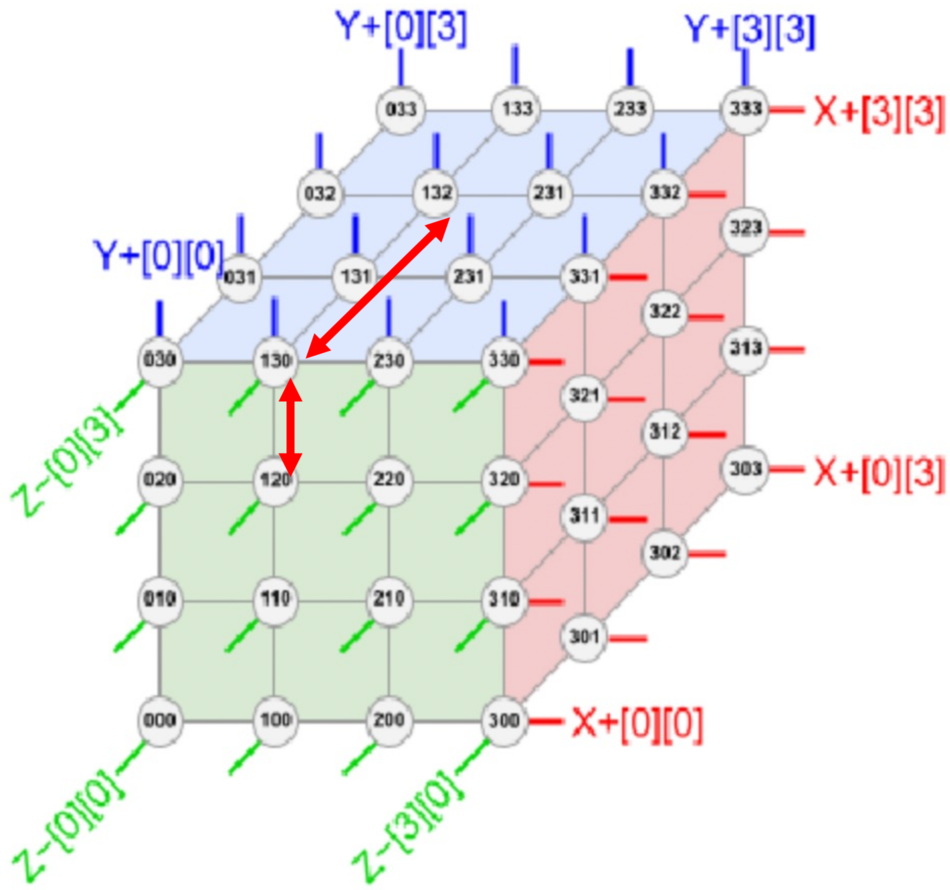
Distributed Training Infrastructure: GPUs



Source: Nvidia

Distributed Training Infrastructure: TPUs





Outline

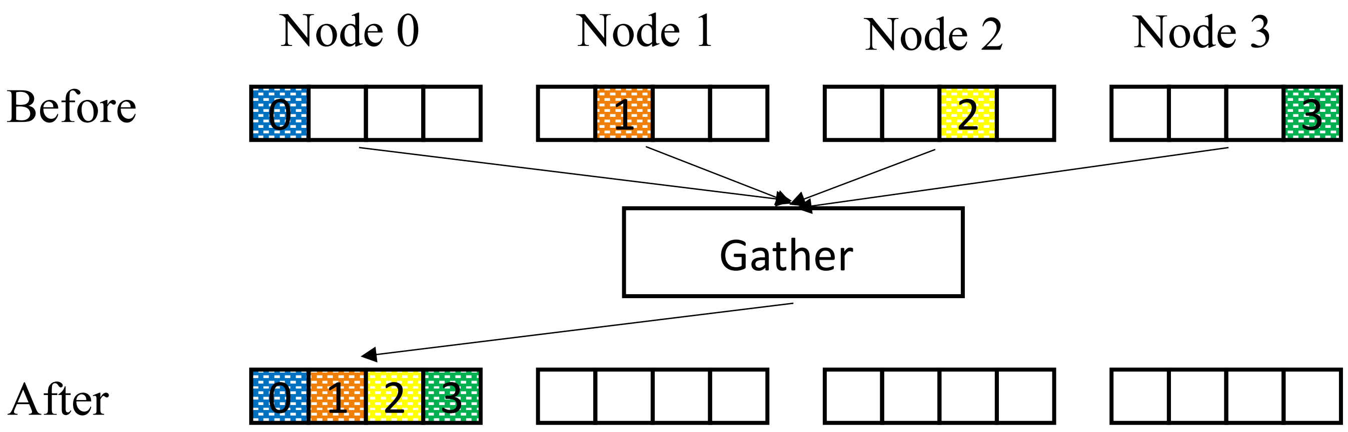
- Quick review: Overlays
- Context on distributed training: app and infrastructure
- Communication collectives: definitions and the function they provide
- How collectives are implemented
  - Focus on one collective: AllReduce
  - Implementation at overlay and underlay level

Collective Communication

- Coordinated exchange of data between a group of nodes (as part of a group computation)
- Originally designed for supercomputers running high-performance computing (HPC) apps
- Now widely used for AI workloads
    - Nvidia Collectives Communication Library (NCCL)
    - MS Azure MSCCL
    - TCCL, TE-CCL, …

A quick example for intuition

(Will return for a deeper look shortly)



Collectives: common characteristics

- **Synchronized**: nodes (GPUs/TPUs) invoke and execute the collective operation in parallel

Machine 1

```
comm = communicator.create()
a = [1, 2, 3]
b = comm.allreduce(a, op=sum)
---------------------------------
assert b == [2, 2, 4]
```

Machine 2

```
comm = communicator.create()
a = [1, 0, 1]
b = comm.allreduce(a, op=sum)
---------------------------------
assert b == [2, 2, 4]
```

Collectives: common characteristics

- **Synchronized**: nodes (GPUs/TPUs) invoke and execute the collective operation in parallel
- **Orchestrated:** centralized job scheduler distributes membership info, member IDs, roles, *etc.*
- **Homogeneous:** all nodes have same resources (compute, BW) and running same code
- **Blocking** (implicit or explicit): collective must complete at all nodes before proceeding
- Data may be **transformed** at intermediate hops in the communication path

Collectives: taxonomy and key operations

- **Data redistribution** (about moving data)
  - Broadcast
  - Scatter
  - Gather
  - AllGather

- **Data consolidation** (about moving *and* aggregating or “reducing” the data)*
  - Reduce
  - Reduce-Scatter
  - AllReduce

*We’ll assume that aggregation = sum

ref: *Chan et al. Concurrency and Practice, 2007*

Notation

- Number of nodes: *p* (=4 in most of our examples)
- Vector of data being exchanged: *x*
- For some operations, *x* is subdivided into subvectors x_i, i=0, 1, .. *p*-1
- Superscript denotes a vector that must be summed with vectors from other nodes:

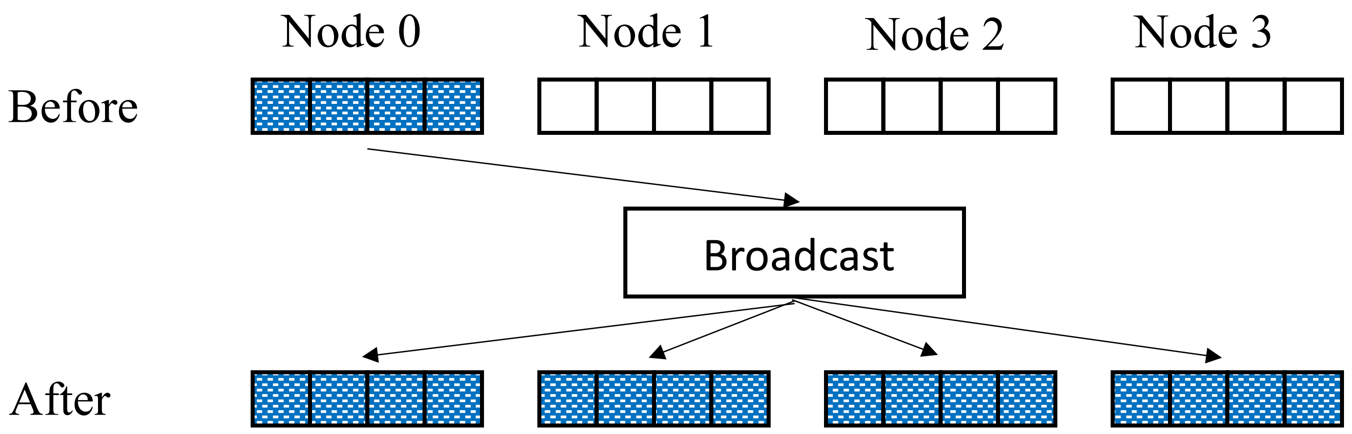
  $\sum_j x_i^{(j)}$ indicates the sum of the subvector x_i from all nodes *j*

ref: *Chan et al. Concurrency and Practice, 2007*

Let’s now look at our set of collectives …

Broadcast (p=4)

Operation	Before				After			
	Node 0	Node 1	Node 2	Node 3	Node 0	Node 1	Node 2	Node 3
Broadcast	x				x	x	x	x

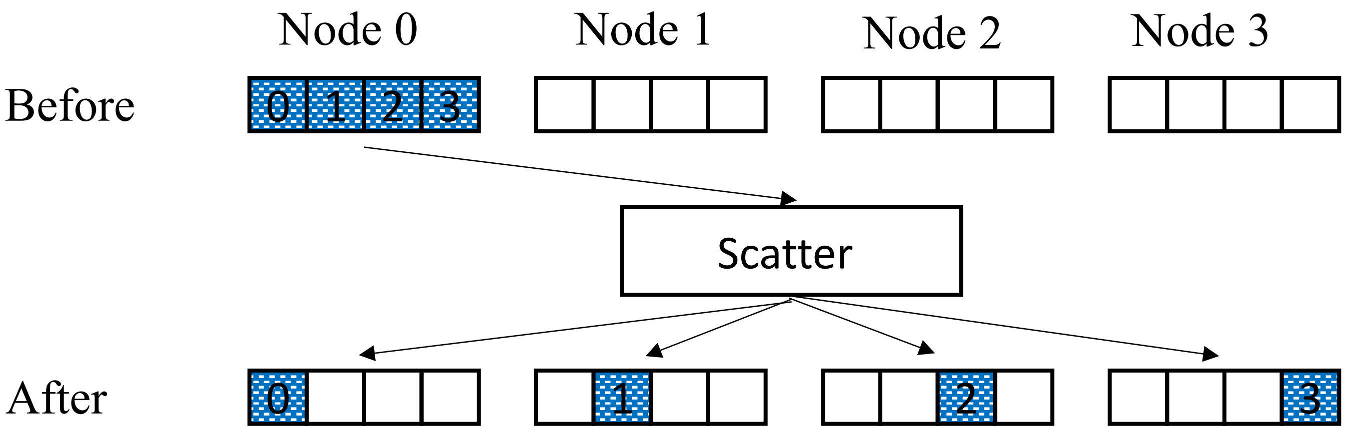


"send to all"

ref: Chan et al. Concurrency and Practice, 2007

Scatter (p=4)

Operation	Before				After			
	Node 0	Node 1	Node 2	Node 3	Node 0	Node 1	Node 2	Node 3
Scatter	x_0				x_0			
	x_1					x_1		
	x_2						x_2	
	x_3							x_3

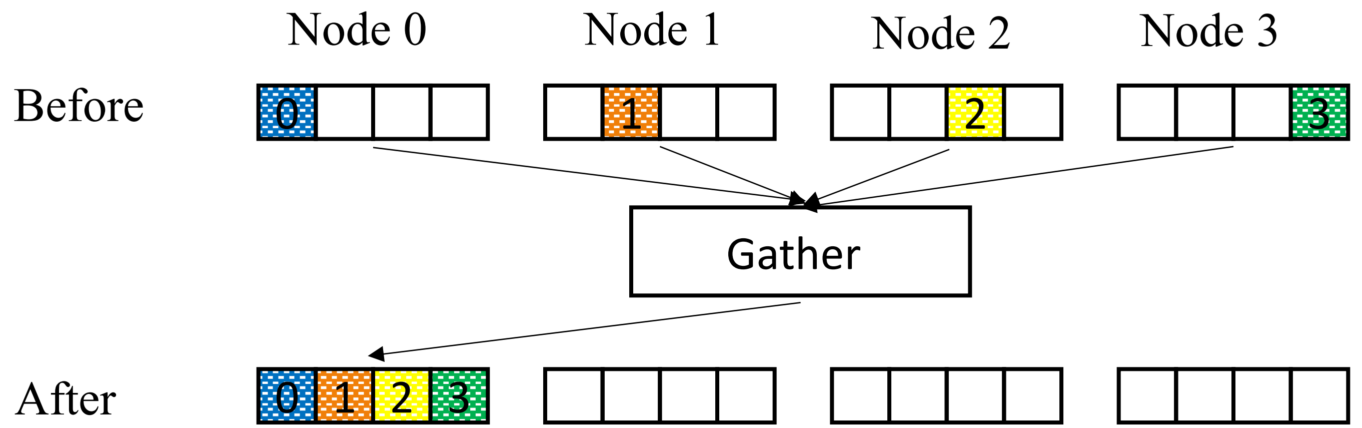


send my i^{th} subvector to node i

ref: Chan et al. Concurrency and Practice, 2007

Gather (p=4)

Operation	Before				After			
	Node 0	Node 1	Node 2	Node 3	Node 0	Node 1	Node 2	Node 3
Gather	x_0	x_1	x_2	x_3	x_0 x_1 x_2 x_3			

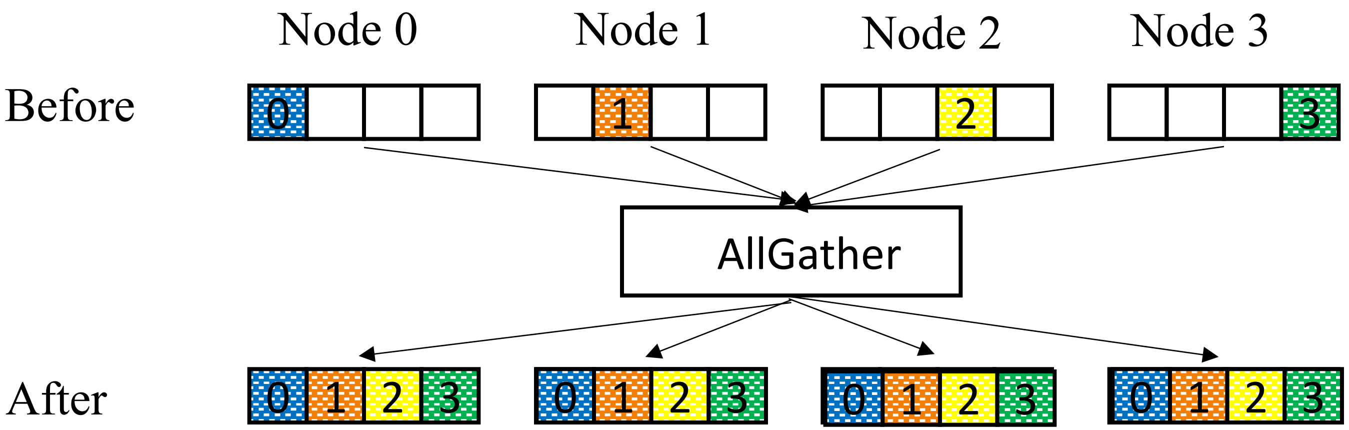


node i sends its
i^{th} subvector to a target node

ref: Chan et al. Concurrency and Practice, 2007

AllGather (p=4)

Operation	Before				After			
	Node 0	Node 1	Node 2	Node 3	Node 0	Node 1	Node 2	Node 3
Allgather	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



Node i sends its i^{th} subvector to all

ref: Chan et al. Concurrency and Practice, 2007

Collectives: taxonomy and key operations

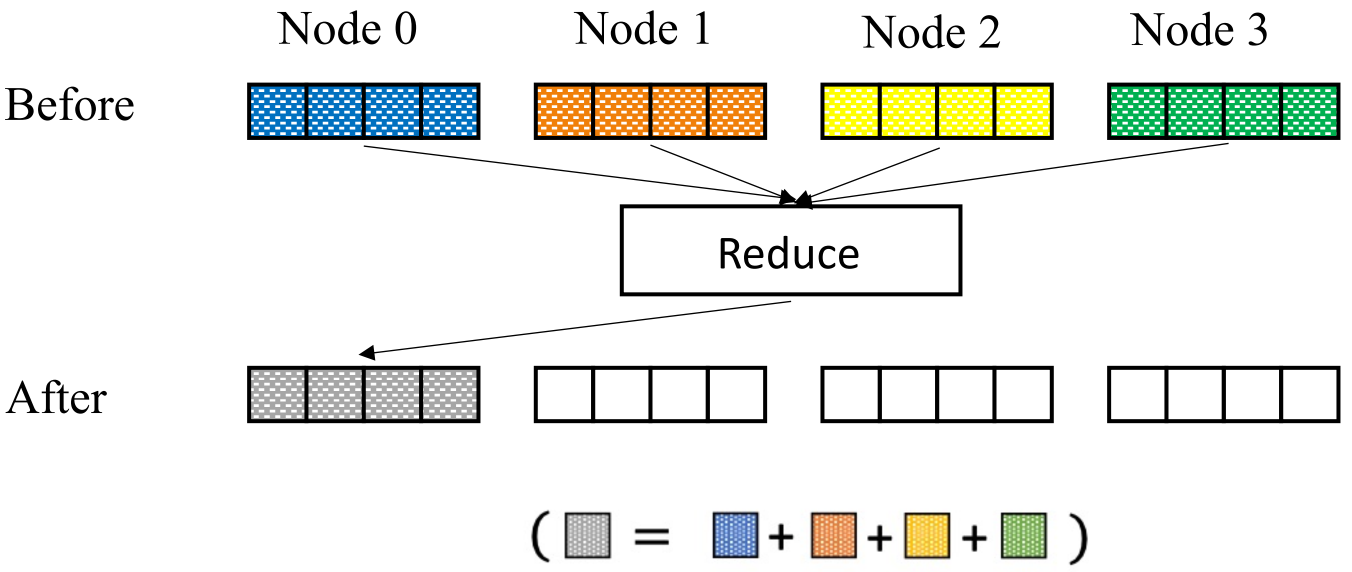
- **Data redistribution** (about moving data)
  - Broadcast
  - Scatter
  - Gather
  - AllGather

- **Data consolidation** (about moving *and* aggregating the data)*
  - Reduce
  - Reduce-Scatter
  - AllReduce

ref: *Chan et al. Concurrency and Practice, 2007*

Reduce (p=4)

Operation	Before				After			
	Node 0	Node 1	Node 2	Node 3	Node 0	Node 1	Node 2	Node 3
Reduce(-to-one)	$x^{(0)}$	$x^{(1)}$	$x^{(2)}$	$x^{(3)}$	$\sum_j x^{(j)}$			

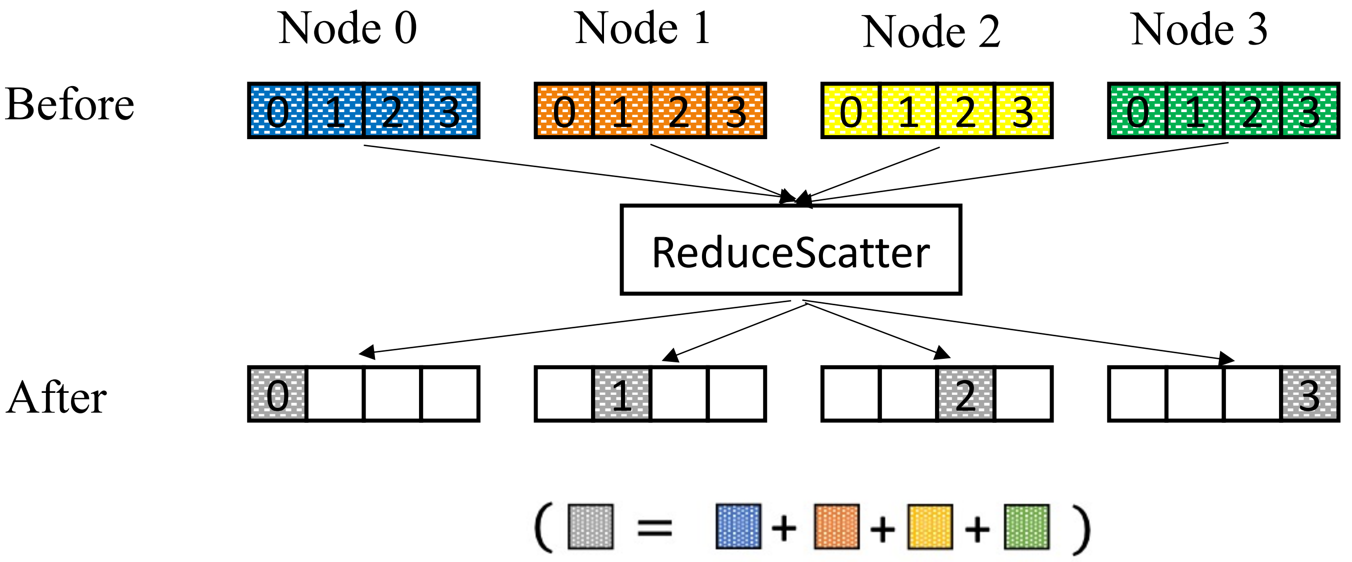


All nodes' vectors are summed at a target node

ref: Chan et al. Concurrency and Practice, 2007

ReduceScatter (p=4)

Operation	Before				After			
	Node 0	Node 1	Node 2	Node 3	Node 0	Node 1	Node 2	Node 3
Reduce-scatter	$x_0^{(0)}$	$x_0^{(1)}$	$x_0^{(2)}$	$x_0^{(3)}$	$\Sigma_j x_0^{(j)}$			
	$x_1^{(0)}$	$x_1^{(1)}$	$x_1^{(2)}$	$x_1^{(3)}$		$\Sigma_j x_1^{(j)}$		
	$x_2^{(0)}$	$x_2^{(1)}$	$x_2^{(2)}$	$x_2^{(3)}$			$\Sigma_j x_2^{(j)}$	
	$x_3^{(0)}$	$x_3^{(1)}$	$x_3^{(2)}$	$x_3^{(3)}$				$\Sigma_j x_3^{(j)}$

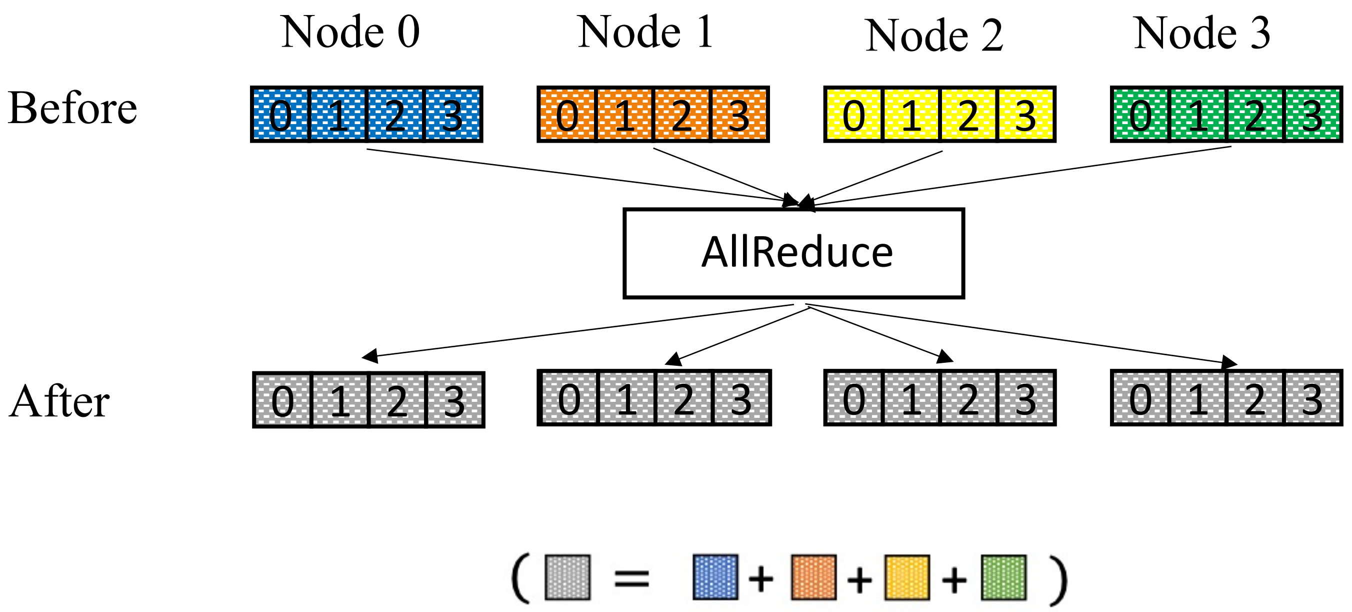


The i^th subvector from all nodes is summed at node i

ref: Chan et al. Concurrency and Practice, 2007

AllReduce (p=4)

Operation	Before				After			
	Node 0	Node 1	Node 2	Node 3	Node 0	Node 1	Node 2	Node 3
Allreduce	$x^{(0)}$	$x^{(1)}$	$x^{(2)}$	$x^{(3)}$	$\Sigma_j x^{(j)}$	$\Sigma_j x^{(j)}$	$\Sigma_j x^{(j)}$	$\Sigma_j x^{(j)}$



All nodes have the sum of all nodes' vectors

ref: Chan et al. Concurrency and Practice, 2007

Some collectives are duals of each other

- Collectives C1 and C2 are duals if *reversing* the communication in C1 yields C2
  - Reversing means A→B becomes B→A
  - We ignore any computation when determining duality

- For a given topology/routing scheme, a collective and its dual see equivalent performance

ref: *Chan et al. Concurrency and Practice, 2007*

Broadcast and Reduce are duals

Operation	Before				After			
	Node 0	Node 1	Node 2	Node 3	Node 0	Node 1	Node 2	Node 3
Broadcast	x				x	x	x	x
Reduce(-to-one)	$x^{(0)}$	$x^{(1)}$	$x^{(2)}$	$x^{(3)}$	$\sum_j x^{(j)}$			

ref: Chan et al. Concurrency and Practice, 2007

Scatter and Gather are duals

Operation	Before				After			
	Node 0	Node 1	Node 2	Node 3	Node 0	Node 1	Node 2	Node 3
Scatter	x_0				x_0			
	x_1					x_1		
	x_2						x_2	
	x_3							x_3
Gather	x_0				x_0			
		x_1			x_1			
			x_2		x_2			
				x_3	x_3			

ref: Chan et al. Concurrency and Practice, 2007

AllGather and ReduceScatter are duals

Operation	Before				After			
	Node 0	Node 1	Node 2	Node 3	Node 0	Node 1	Node 2	Node 3
Allgather	x_0	x_1	x_2	x_3	x_0 x_1 x_2 x_3	x_0 x_1 x_2 x_3	x_0 x_1 x_2 x_3	x_0 x_1 x_2 x_3
Reduce-scatter	$x_0^{(0)}$ $x_1^{(0)}$ $x_2^{(0)}$ $x_3^{(0)}$	$x_0^{(1)}$ $x_1^{(1)}$ $x_2^{(1)}$ $x_3^{(1)}$	$x_0^{(2)}$ $x_1^{(2)}$ $x_2^{(2)}$ $x_3^{(2)}$	$x_0^{(3)}$ $x_1^{(3)}$ $x_2^{(3)}$ $x_3^{(3)}$	$\Sigma_j\, x_0^{(j)}$	$\Sigma_j\, x_1^{(j)}$	$\Sigma_j\, x_2^{(j)}$	$\Sigma_j\, x_3^{(j)}$

ref: Chan et al. Concurrency and Practice, 2007

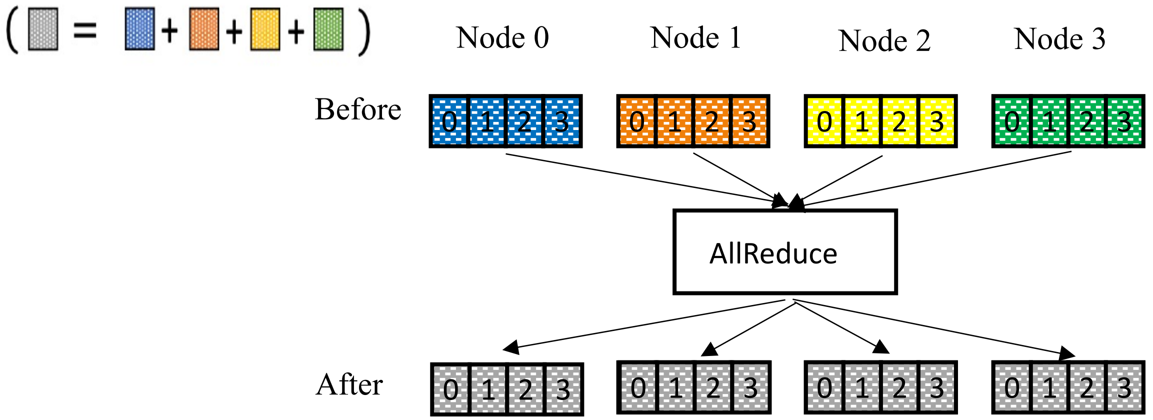
AllReduce does not have a dual

Note: AllReduce = ReduceScatter + AllGather (Convince yourself of this later!)

Outline

- Context on distributed training: app and infrastructure
- Communication collectives: definitions and the function they provide
- How collectives are implemented
    - We'll focus on one collective: AllReduce

Implementing AllReduce



- Let's consider the simplest implementation: a full mesh of node-to-node exchanges
  - Every node sends its vector ***x*** directly to every other node
  - Each node sums all the vectors it receives

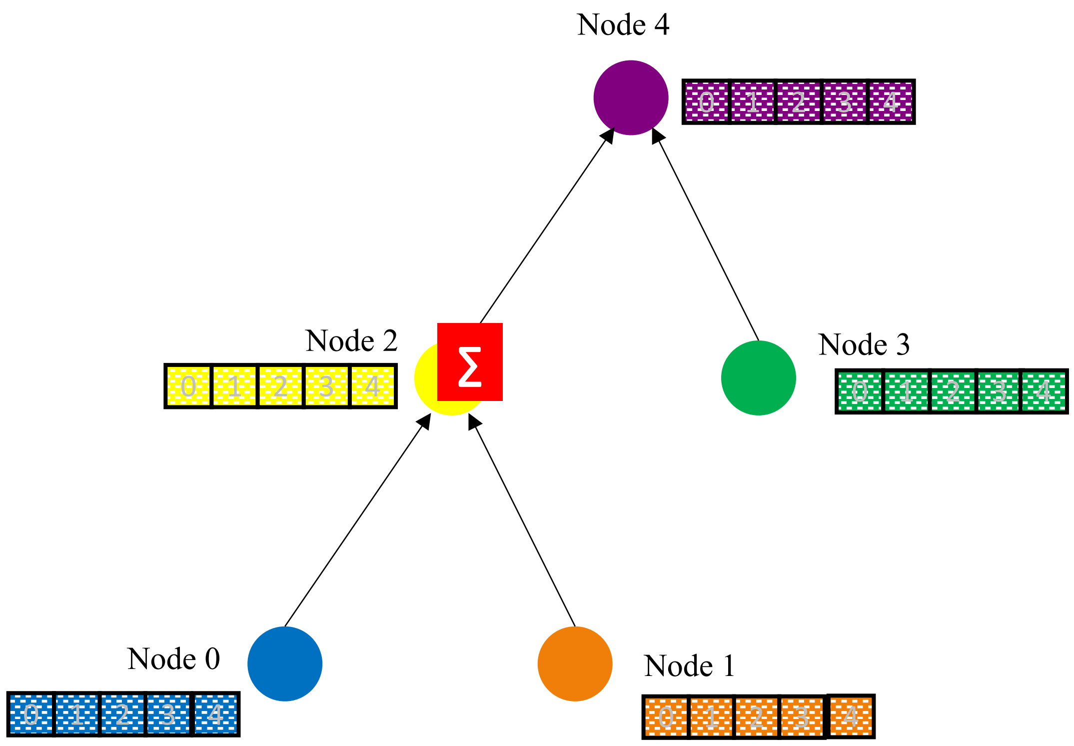
- Assuming ***p*** nodes and that vector ***x*** is of size D bytes:
  - Each node implements *p-1* transmit, receive, and summation operations
  - Each on a vector of size D bytes
  - Total traffic in the network: $O(p^2 \times D)$
  - Consider D ~100s of GB and p ~ 1000s of nodes

- Need a more scalable solution!

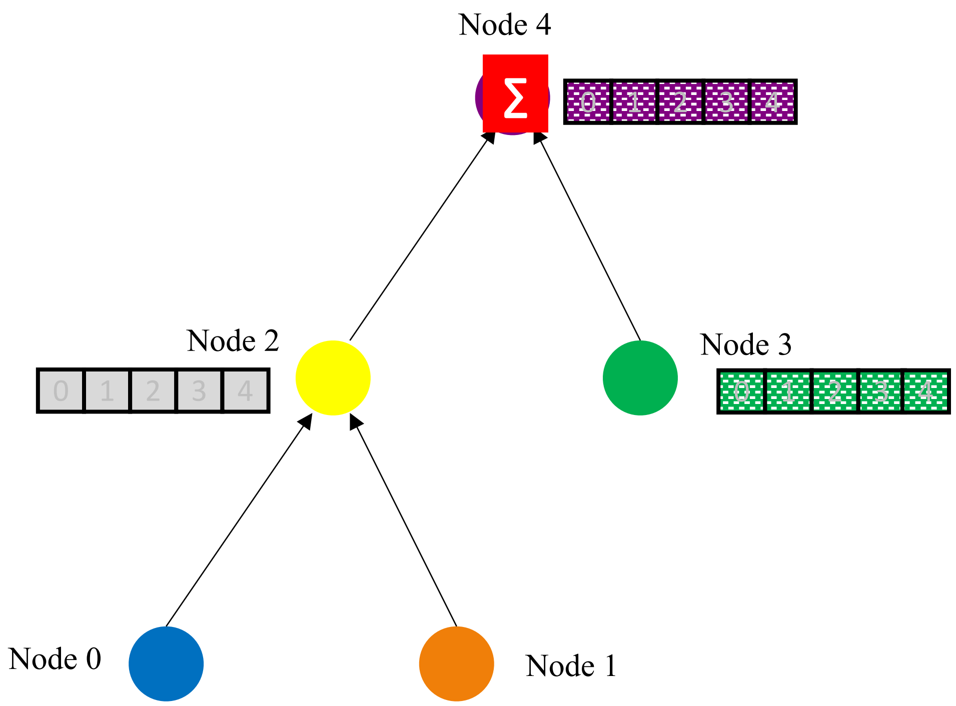
Implementing AllReduce … using overlays

- Idea: construct a virtual topology between the *p* nodes; “reduce” (sum) data as it traverses the overlay
- Details vary depending on the virtual topology selected
- Two typical choices: tree or ring

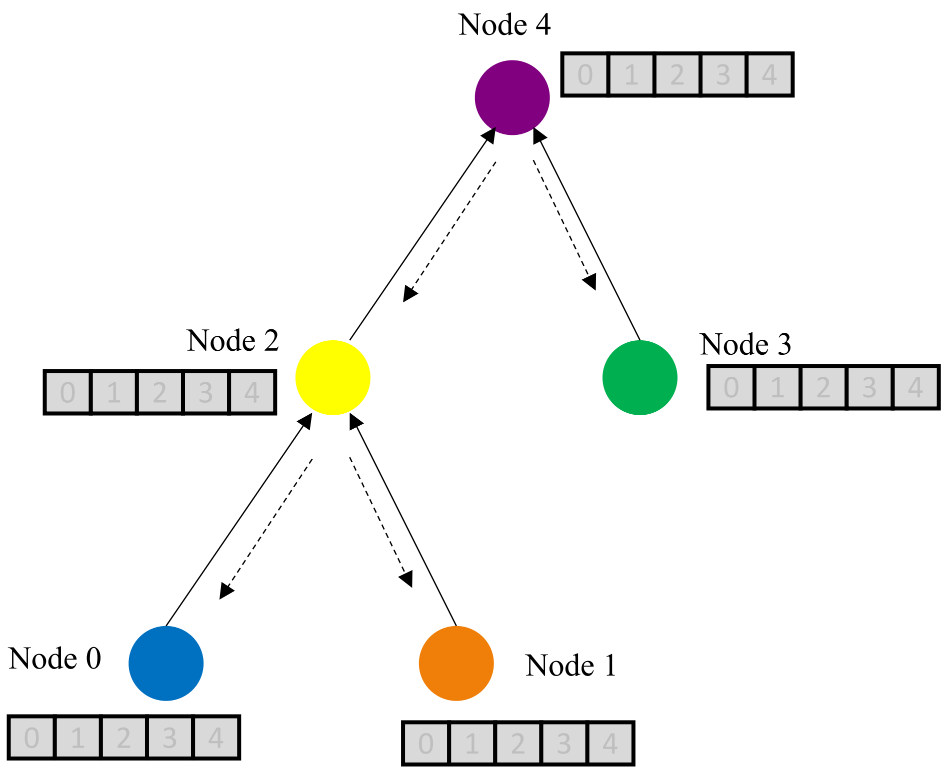
Tree-based AllReduce



Tree-based AllReduce



Tree-based AllReduce

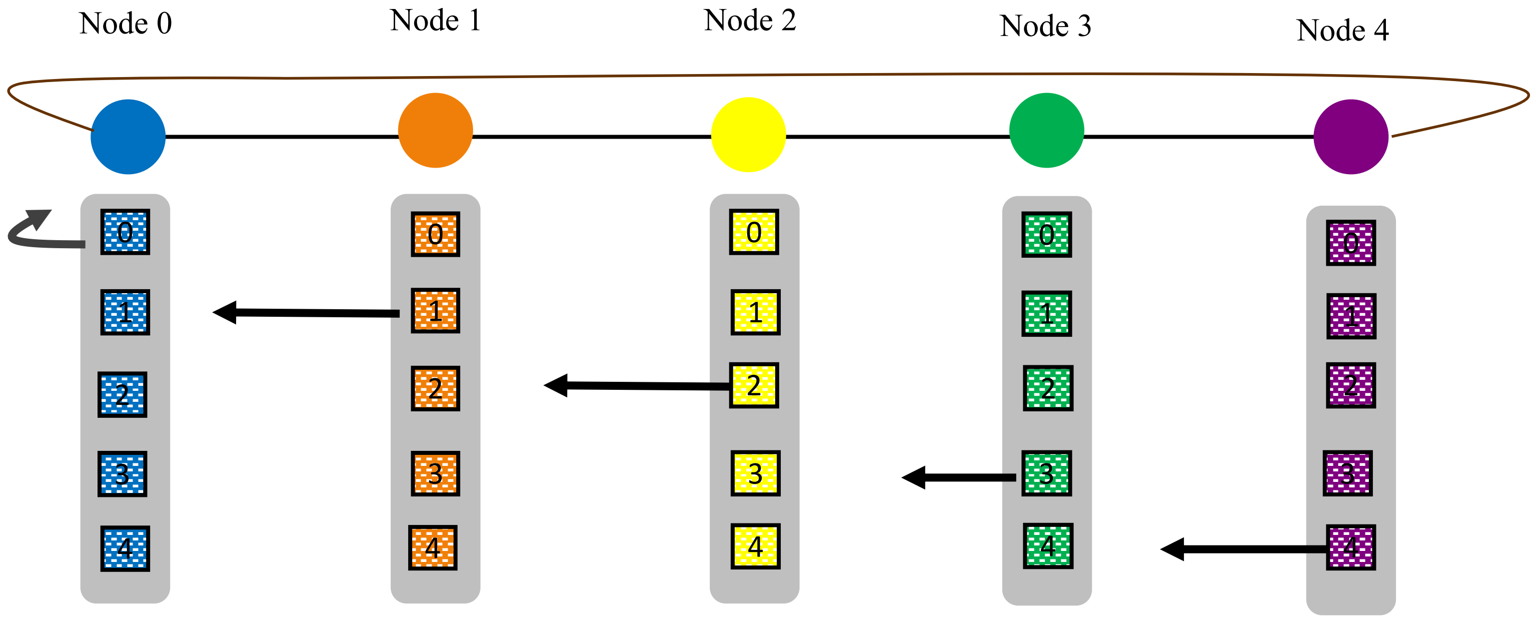


Tree-based AllReduce

- Nodes form a logical tree; aggregate to the root and then broadcast
  - Leaf node: transmits vector to its parent
  - Every intermediate node
    - aggregates (sums) its own vector with that from each child node
    - send the resulting (aggregated) vector to its parent
  - Root broadcasts the final aggregated vector down the tree

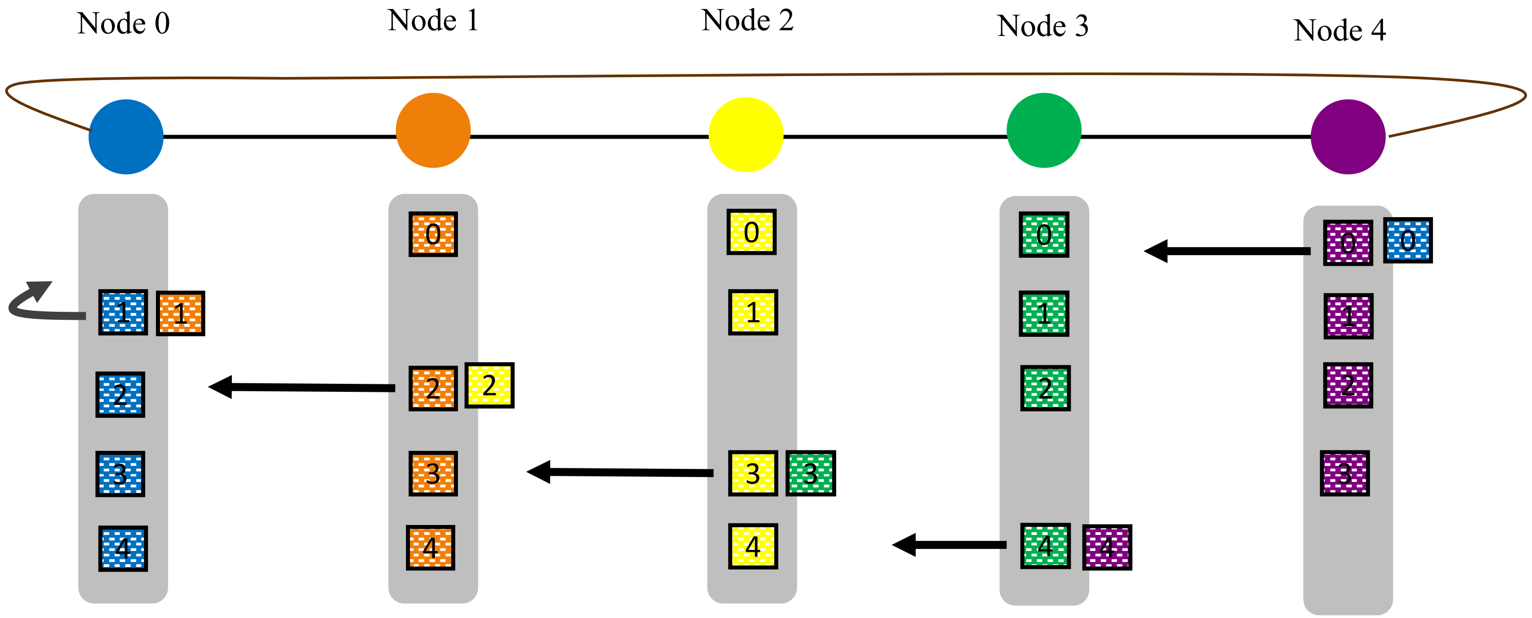
- Assuming a binary tree with ***p*** nodes and vector of D bytes:
  - AllReduce involves O(log P) steps *– to travel up and down the tree*
  - Each node implements 3 transmit, receive, and summation operations , each on a vector of size D bytes
  - Total traffic in the network: O(p x D) across all steps *– much better than with the mesh!*

Ring-based AllReduce



- Initial step: node i sends i^th subvector to its predecessor on the ring

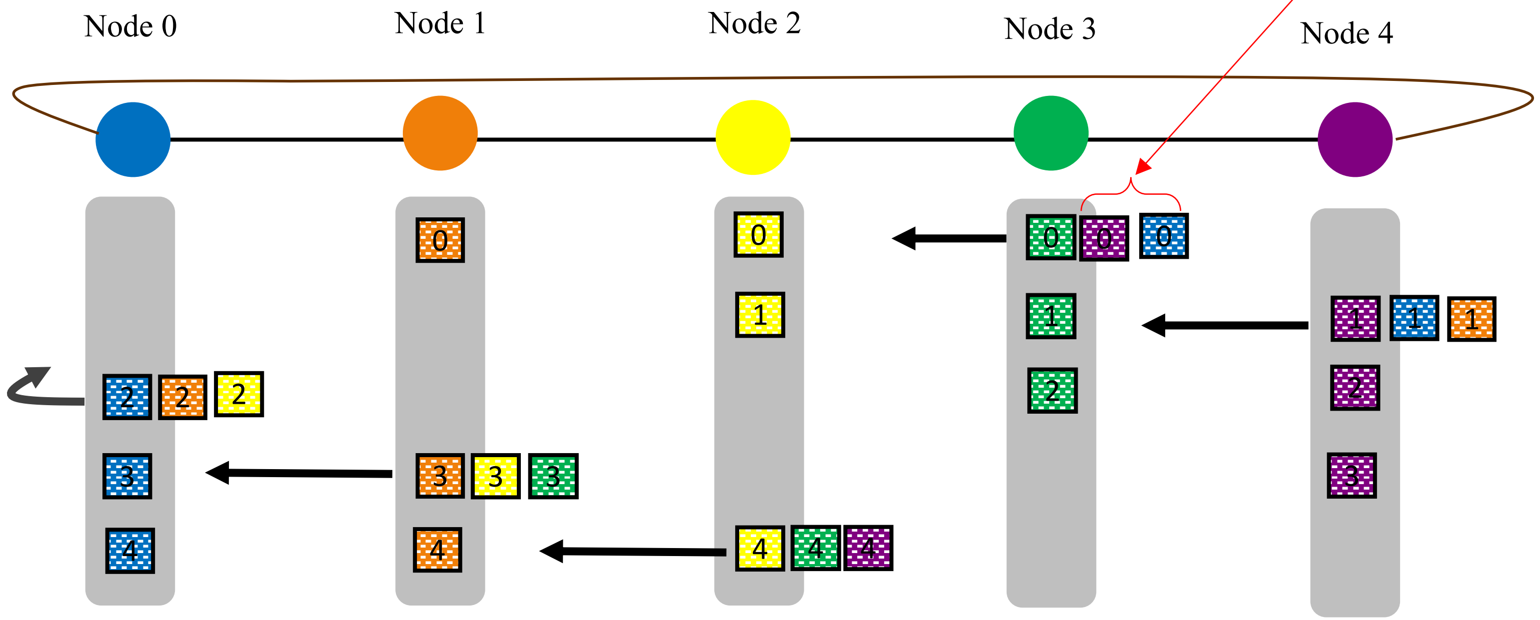
Ring-based AllReduce



- Initial step: node i sends i^th subvector to its predecessor on the ring
- When a node receives the k^th subvector from its successor, it adds this to its own k^th subvector and in the next step, sends this (aggregated) k^th subvector to its predecessor

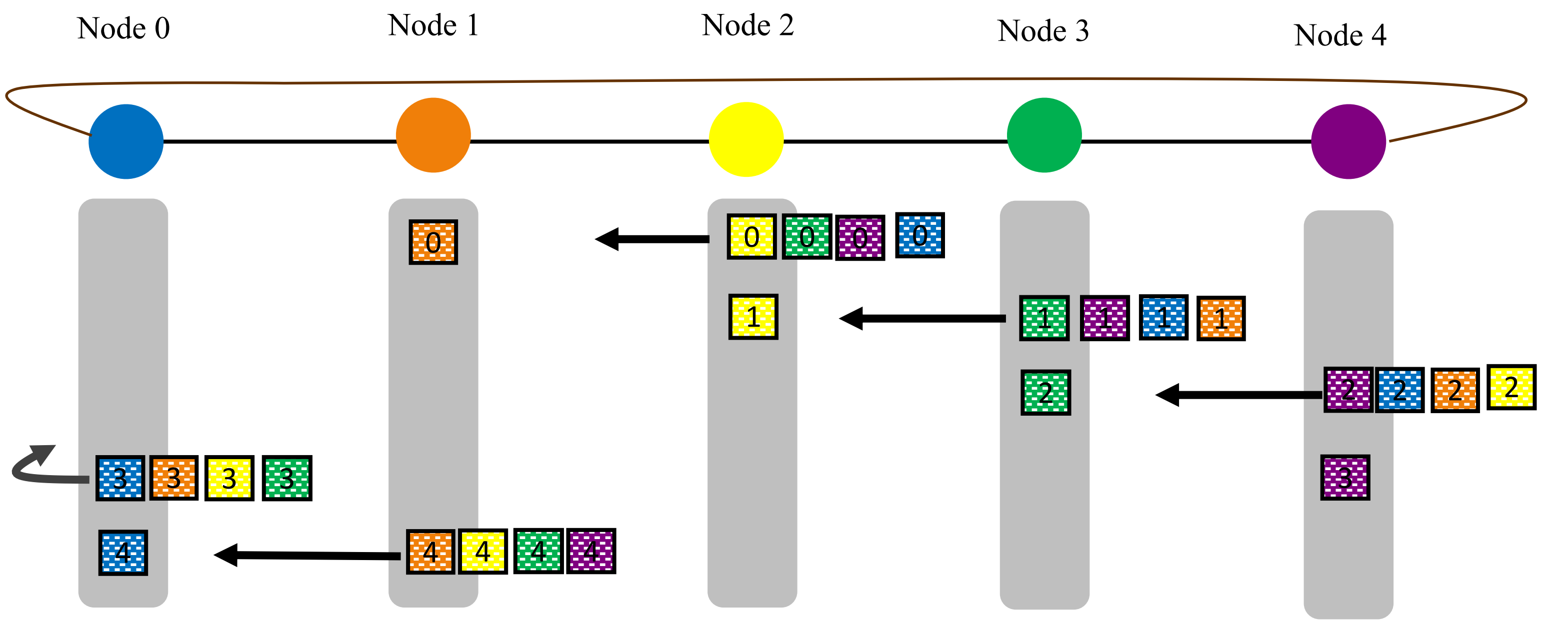
Ring-based AllReduce

In reality: we sent a single subvector equal to the sum of these subvectors (not showing the aggregated subvector for illustration purposes only)



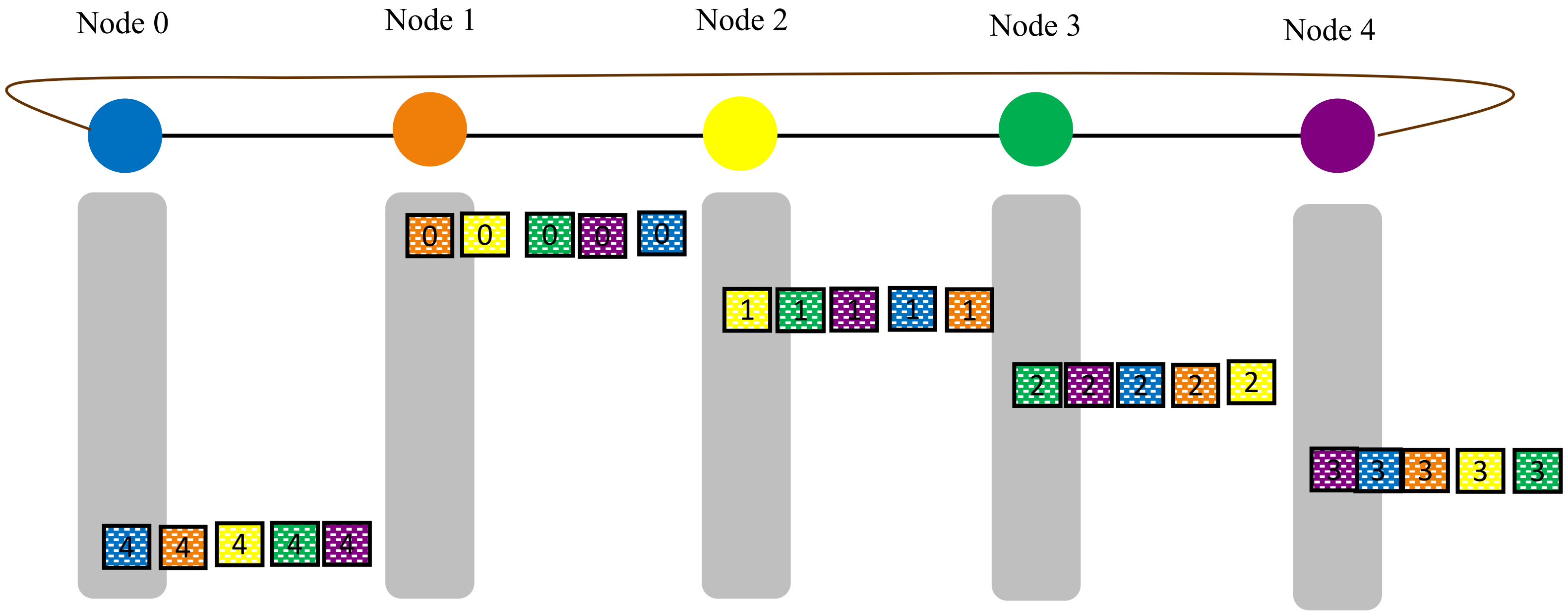
- Initial step: node i sends i^th subvector to its predecessor on the ring
- When a node receives the k^th subvector from its successor, it adds this to its own k^th subvector and in the next step, sends this (aggregated) k^th subvector to its predecessor

Ring-based AllReduce



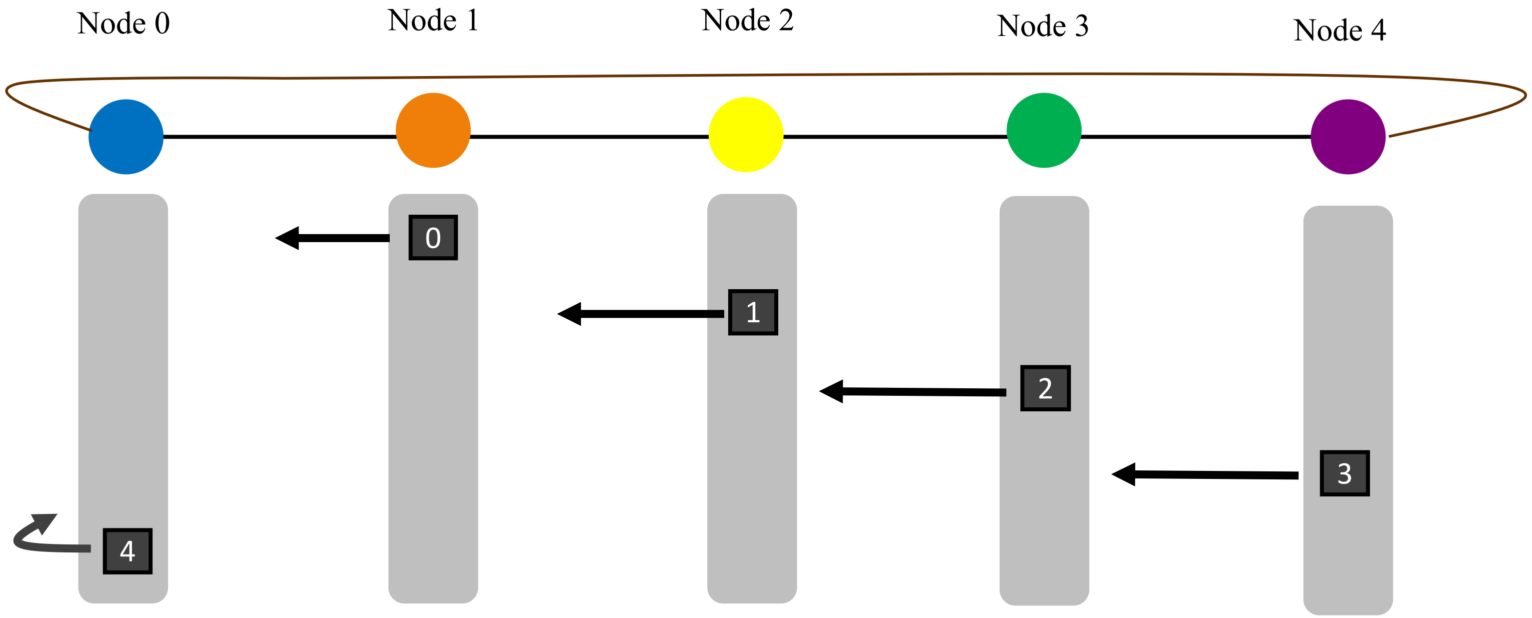
- Initial step: node i sends i^th subvector to its predecessor on the ring
- When a node receives the k^th subvector from its successor, it adds this to its own k^th subvector and in the next step, sends this (aggregated) k^th subvector to its predecessor

Ring-based AllReduce



- Initial step: node i sends i^th subvector to its predecessor on the ring
- When a node receives the k^th subvector from its successor, it adds this to its own k^th subvector and in the next step, sends this (aggregated) k^th subvector to its predecessor
- After p-1 steps, the successor of node i will have the fully aggregated i^th subvector

Ring-based AllReduce



- Initial step: node i sends i^th subvector to its predecessor on the ring
- When a node receives the k^th subvector from its successor, it adds this to its own k^th subvector and in the next step, sends this (aggregated) k^th subvector to its predecessor
- After p-1 steps, the successor of node i will have the fully aggregated i^th subvector
- Repeat (without aggregation); after p-1 steps all nodes have the entire reduced vector!

Ring-based AllReduce

- Process
  - Initial step: node i sends i^th chunk to its predecessor on the logical ring
  - When a node receives the k^th chunk from its successor, it adds this to its own k^th chunk and in the next step, sends this (aggregated) k^th chunk to its predecessor
  - After p-1 steps, the successor of node i will have the fully aggregated i^th chunk
  - Repeat but now without aggregation; after p-1 steps all nodes have the entire reduced vector!

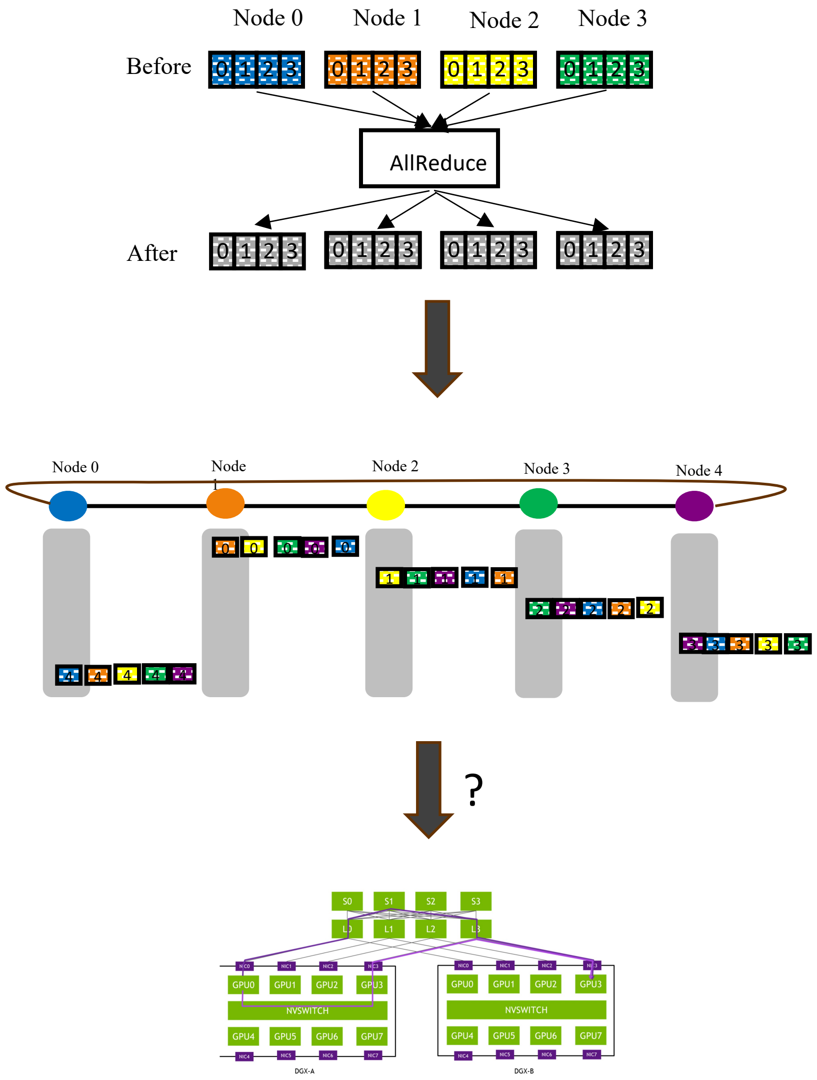
- Assuming a ring with ***p*** nodes and vector of D bytes:
  - Takes ~2*p* steps
  - In each step, a node transmits/receives/sums a subvector of size D/*p* bytes
  - Total traffic in the network across all steps: O(*p* x D)

source: horovod paper

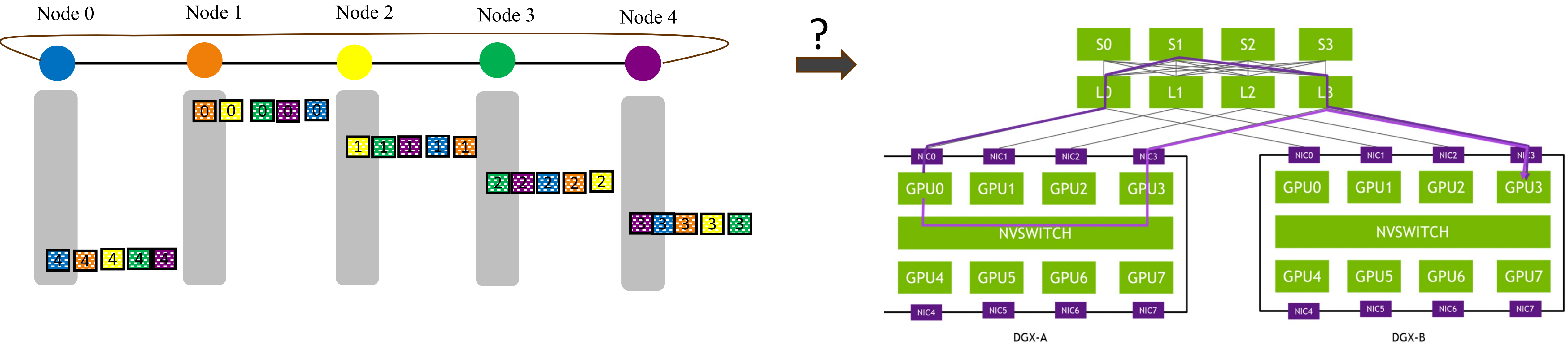
Taking Stock

Three levels at which we can view communication collectives:

- Definition or “service model”
- Overlay viewpoint
- Underlay (physical network) viewpoint



Closing point:



How do we assign overlay nodes to physical GPUs so as to achieve low “stretch” (last lecture)?

Questions?