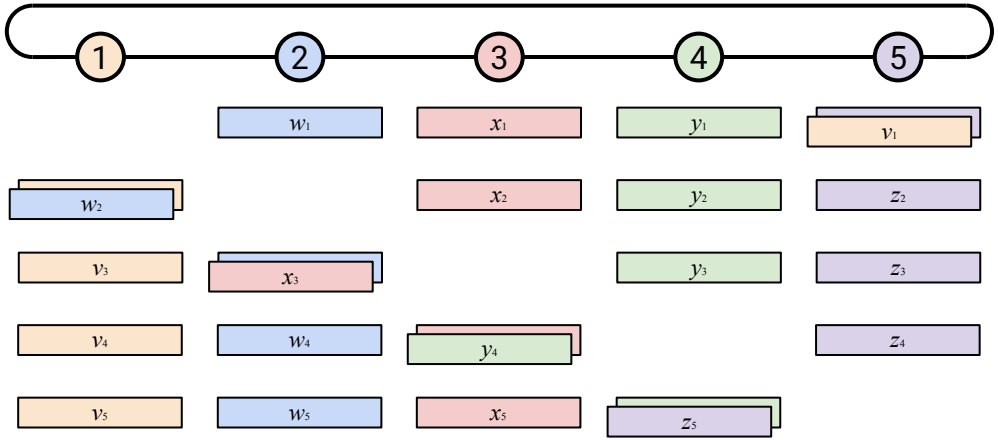


Lecture 23 (Beyond Client-Server 2)

Collective Operations

CS 168, Spring 2026 @ UC Berkeley

Slides credit: Sylvia Ratnasamy, Rob Shakir, Peyrin Kao

Motivation: Distributed AI Training

Lecture 23, CS 168, Spring 2026

Motivation: AI Training

Modern AI systems require training models on huge amounts of data.

For this class, we'll use a very simplified model of AI training.

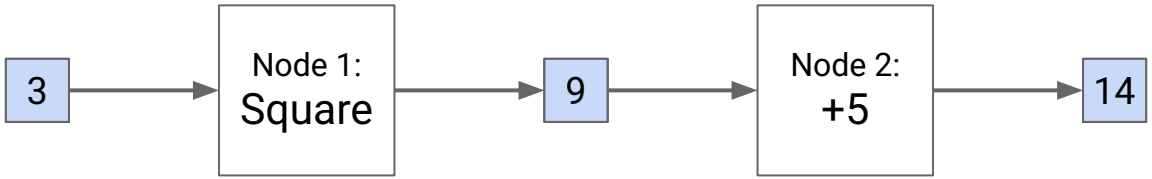
Distributed Training

Modern AI training jobs are far too large to be run on a single machine.

- The job must be **distributed** across multiple machines.

Many options for how to distribute the work. Examples:

- Each node works on a subset of the training data.
- Each node works on a subset of the model.
- Each node works on a subset of the task (pipelining).
- For this class, we don't care which option is used.

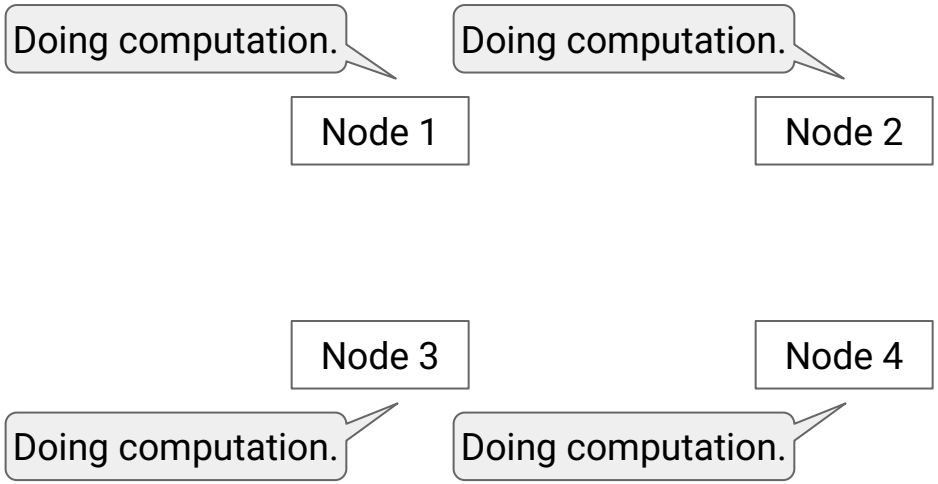


Example of pipelining. The task is to compute f(x) = x^2 + 5.
We split the task into two stages.

Nodes Exchanging State

Regardless of how you distribute the work, nodes must periodically exchange state.

- Example: Each node has a piece of the solution, and they must exchange state to reconstruct the full solution.
- Example: Node 2 needs data from Node 1 to continue its computation.

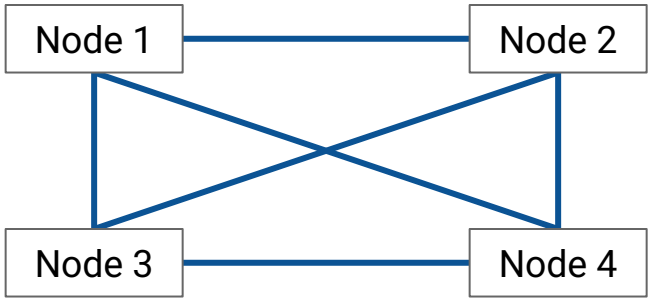


1. **Split job into sub-tasks.**
   Each node runs a sub-task.
2. After every node finishes their sub-task, the nodes exchange a large amount of state.
3. Proceed to the next task, and repeat Steps 1–2 for the next task.

Nodes Exchanging State

Regardless of how you distribute the work, nodes must periodically exchange state.

- Example: Each node has a piece of the solution, and they must exchange state to reconstruct the full solution.
- Example: Node 2 needs data from Node 1 to continue its computation.

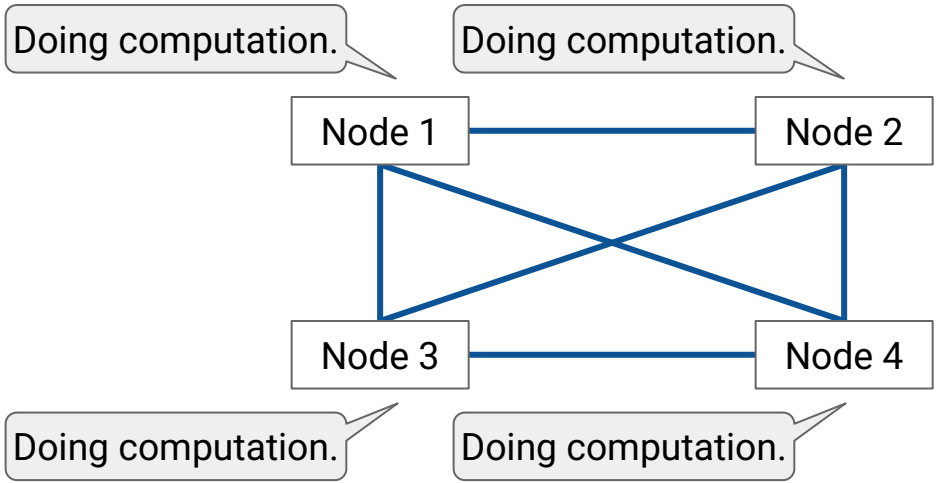


1. Split job into sub-tasks.
   Each node runs a sub-task.
2. **After every node finishes their sub-task, the nodes exchange a large amount of state.**
3. Proceed to the next task, and repeat Steps 1–2 for the next task.

Nodes Exchanging State

Regardless of how you distribute the work, nodes must periodically exchange state.

- Example: Each node has a piece of the solution, and they must exchange state to reconstruct the full solution.
- Example: Node 2 needs data from Node 1 to continue its computation.



1. Split job into sub-tasks.
   Each node runs a sub-task.
2. After every node finishes their sub-task, the nodes exchange a large amount of state.
3. Proceed to the next task, and repeat Steps 1–2 for the next task.

Distributed Training Infrastructure

Lecture 23, CS 168, Spring 2026

Distributed AI Training

- Motivation
- **Infrastructure**

Defining Collectives

- Definition
- Redistribution Operations
- Consolidation Operations
- Duals and Compositions

Implementing Collectives

- Mesh Topology
- Single Root
- Tree
- Ring (Naive)
- Ring (Optimized)
- Overlay/Underlay Topologies

Distributed Training Infrastructure: GPUs, TPUs

What exactly is each "node"?

- GPUs (Graphics Processing Units): Chips that are good at AI operations.
- TPUs (Tensor Processing Units): AI-optimized chips designed by Google.

The nodes are inter-connected in a datacenter-like network.

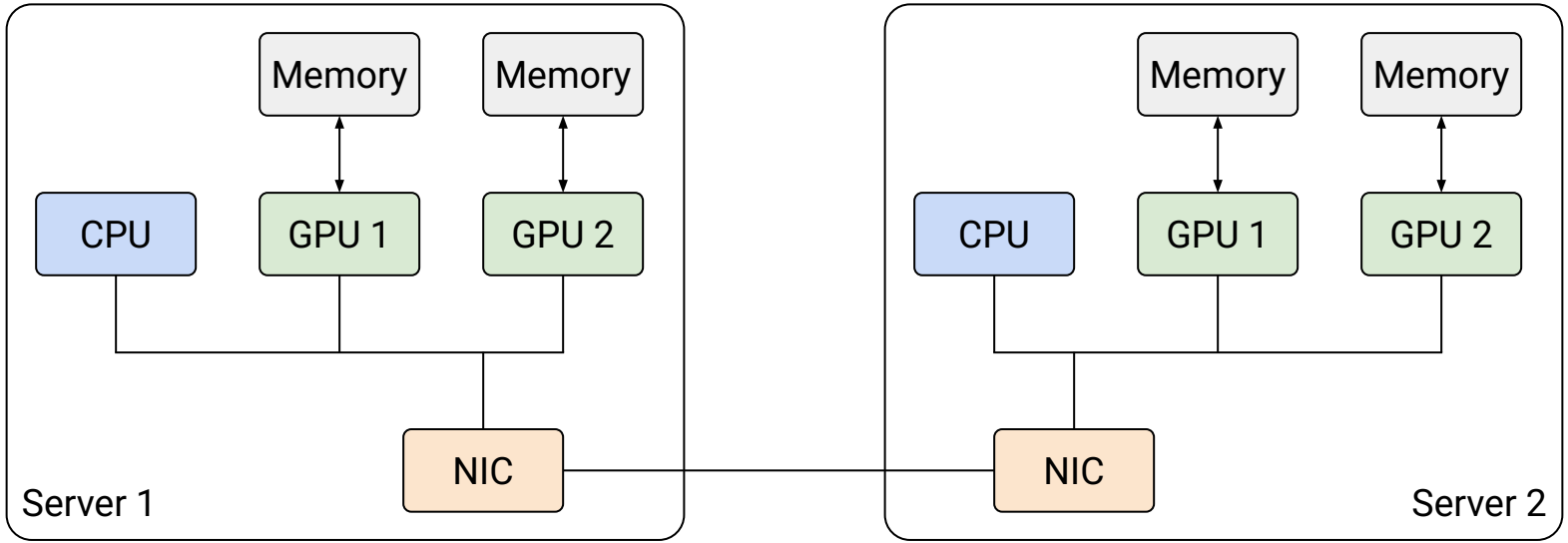
Recall the properties of a datacenter network:

- Nodes are physically close (e.g. same building).
- Nodes are organized in a structured topology (e.g. Clos network).
- Nodes are homogeneous (e.g. all the same model of GPU).
- Links have very high bandwidth.

Distributed Training Infrastructure: Servers

Each server in the datacenter has:

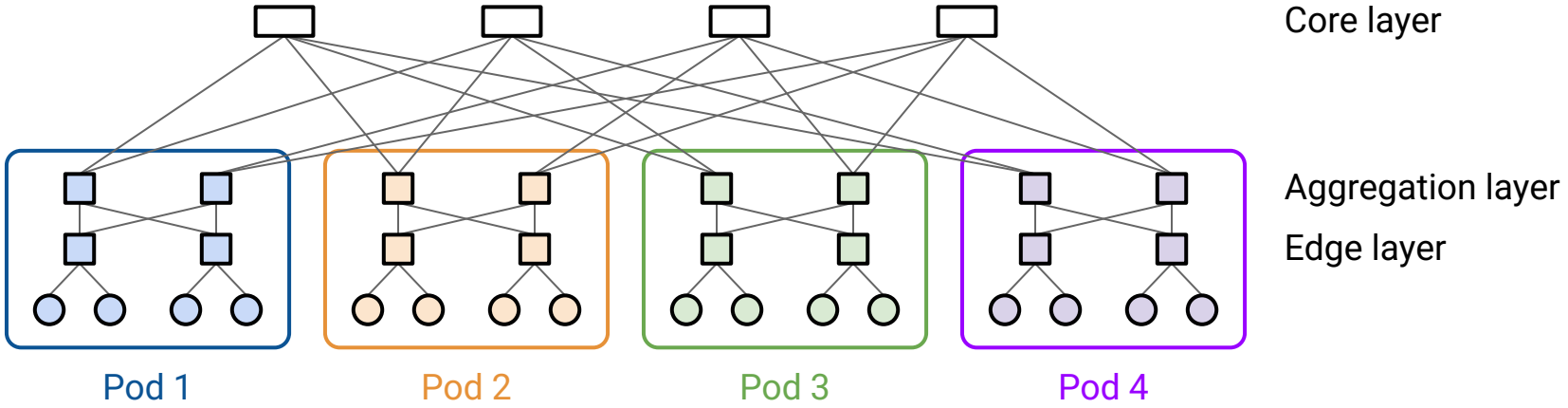
- One or more GPUs/TPUs. Used to do the AI training.
- A relatively weak CPU. Used for miscellaneous operations, not AI training.
- A NIC, shared by all the chips on this server.



Distributed Training Infrastructure: Network Topology

The servers are connected in a structured topology.

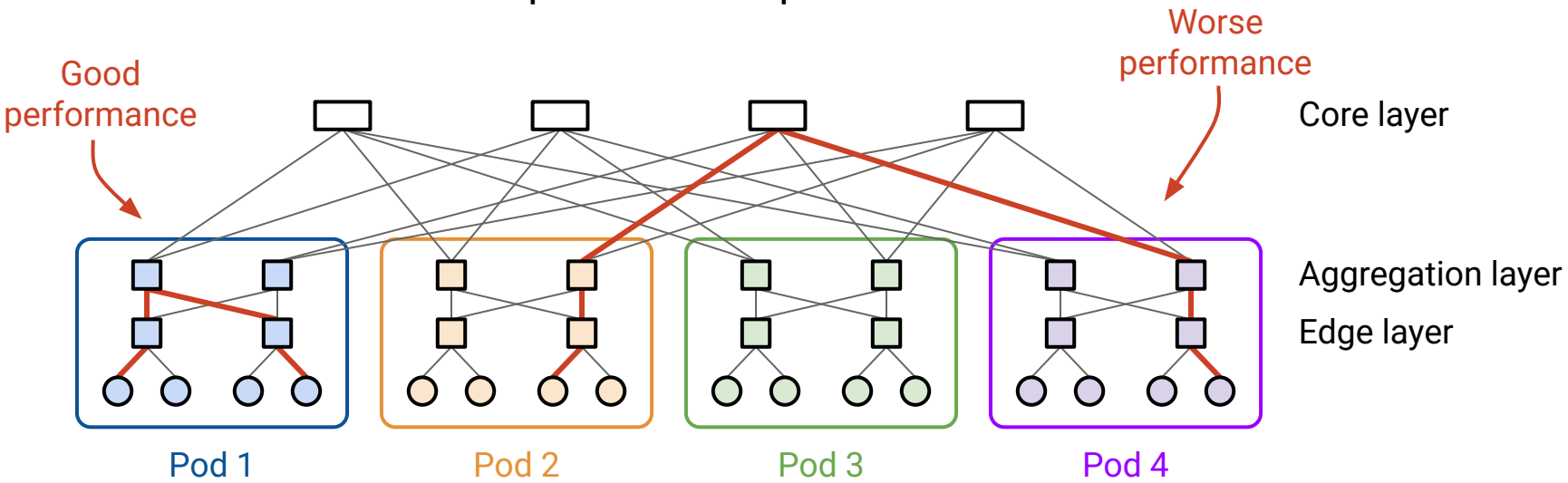
- Many different possible topologies. Example: Fat-tree Clos topology.
- Remember: Each server (○ ○ ○ ○) could have one or more GPUs/TPUs.



Distributed Training Infrastructure: Network Topology

Notice: Some pairs of nodes are more closely-connected than others.

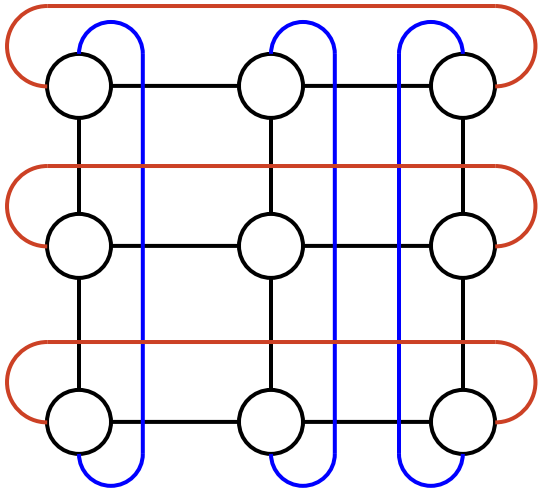
- Two nodes on the same server: Great performance.
  - Don't need to use the network at all.
  - Nodes can communicate with effectively infinite bandwidth and zero latency.
- Two nodes in the same pod: Good performance.
- Two nodes in different pods: Worse performance.



Distributed Training Infrastructure: Network Topology

Alternate topology: **2D torus**: A square, where the edges wrap around.

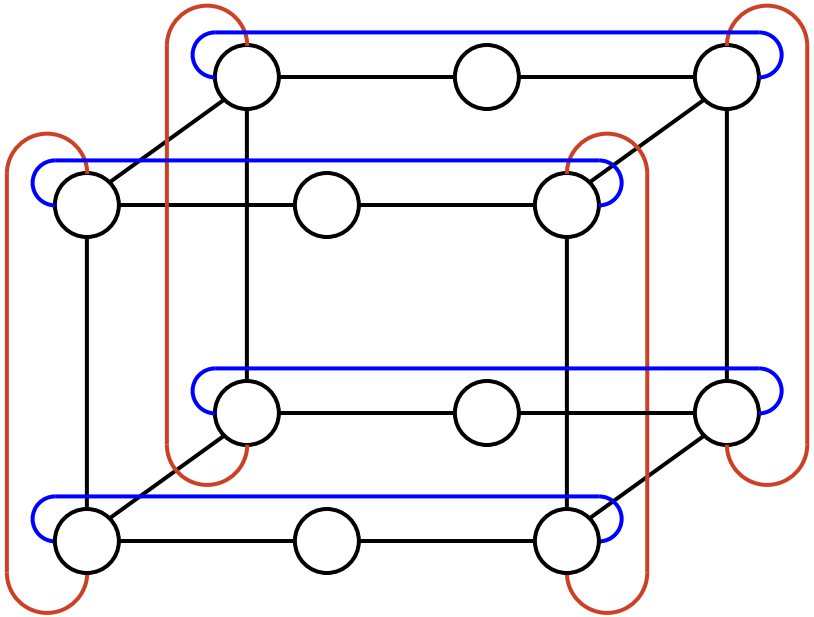
- TPUs can be directly connected to each other with ICI (Inter-Core Interconnect), with no routers needed.
- As before, some pairs of nodes are more closely-connected than others.
- "Wrap around" means: If you're at a bottom node and traverse a link down, you'll end up at the top node. (And similar for the other directions.)



Distributed Training Infrastructure: Network Topology

Alternate topology: **3D torus**: A cube, where the edges wrap around.

- TPUs can be directly connected to each other with ICI (Inter-Core Interconnect), with no routers needed.
- As before, some pairs of nodes are more closely-connected than others.



Collective Communication: Definition

Lecture 23, CS 168, Spring 2026

Collective Communication: Definition

Collective communication: A group of nodes that exchange data in a coordinated manner as part of a group computation.

- The nodes work together to achieve a common goal.
- The nodes have to exchange data during that process.

Brief history:

- Originally developed for supercomputers.
- Now an active area of research again in the context of AI training.
- Modern implementations: NCCL (Nvidia), MSCCL (Microsoft), etc.
- NCCL code is available online if you're curious.

Why is Collective Communication Different?

Highly structured communication.

- General Internet: Hosts communicate whenever they want.
- Collectives: We know about a tightly-scripted set of exchanges that must occur.

Dedicated network infrastructure.

- General Internet: Many applications/tenants/users running at the same time.
- Collectives: The AI training job is the only job running.
  This lets us plan out how much bandwidth will be used ahead of time.

Data is transformed as it's exchanged.

- General Internet: The data sent is identical to the data received.
- Collectives: Nodes perform computation as the data is forwarded across nodes.

Collective Communication: Operations

Collective operations: A set of basic communication patterns.

- Users invoke these basic operations to implement their desired communication.
- Think of these as the "API" for distributed communication.

Our focus:

- **What**: The definitions of these collectives, e.g. inputs and outputs.
- **How**: How these collectives are implemented in the network.
- **Why**: We won't discuss why AI training can be broken down into these operations.

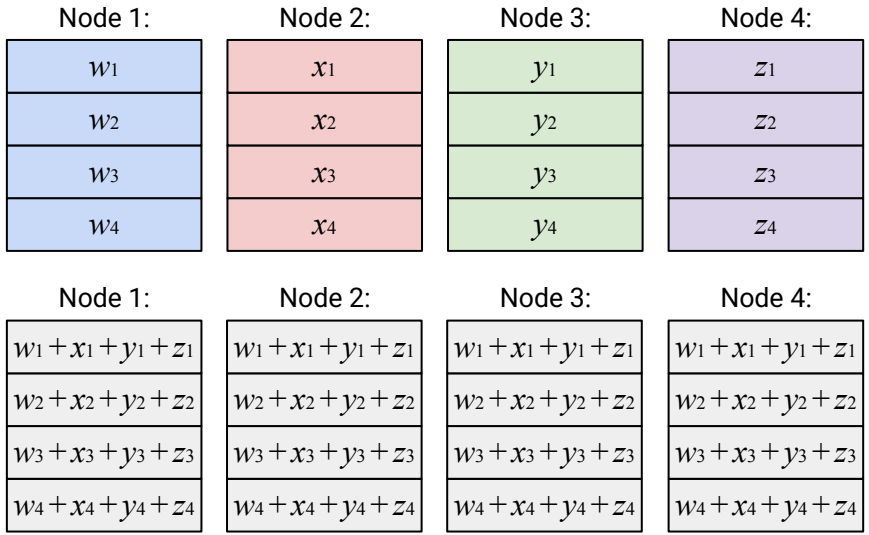
Collective Communication: Setup

There are *p* nodes. Each node specifies a *p*-element vector of input data, and receives a *p*-element vector of output data.

- Each vector element could be an integer, a chunk of training data, etc.
- Each operation specifies what output values should be sent to each node.
- The data could be transformed in the output, e.g. summing vector elements.

Input: Each node specifies a *p*-element vector of input data.

(e.g. Specify the memory address of a read buffer.)



p = 4 in this example.

Output: Each node receives a *p*-element vector of output data.

(e.g. Specify a memory address, and receive output at that write buffer.)

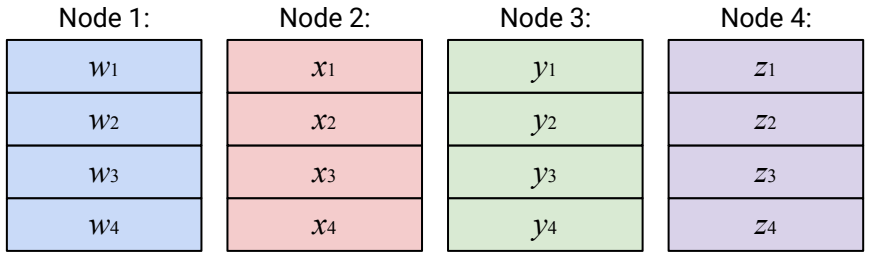
Collective Communication: Setup

Before the operation, a centralized controller does some setup.

- Tell each node its number ("you are Node 1").
- Tell each node the value of *p* ("there are 4 nodes in total").
- Specify the memory addresses of the read buffer and write buffer.

Input: Each node specifies a *p*-element vector of input data.

(e.g. Specify the memory address of a read buffer.)

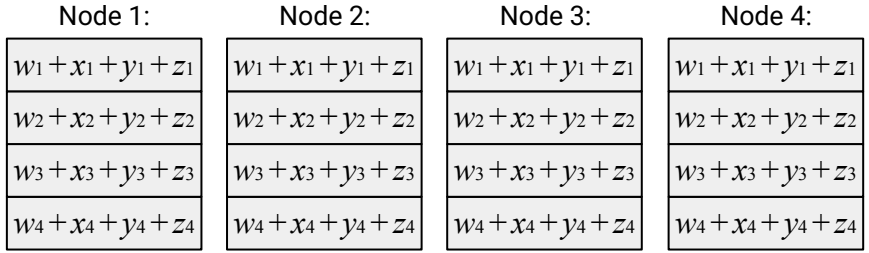


p = 4 in this example.

AllReduce

Output: Each node receives a *p*-element vector of output data.

(e.g. Specify a memory address, and receive output at that write buffer.)



Collective Communication: Setup

During the operation, nodes exchange data to produce the desired output.

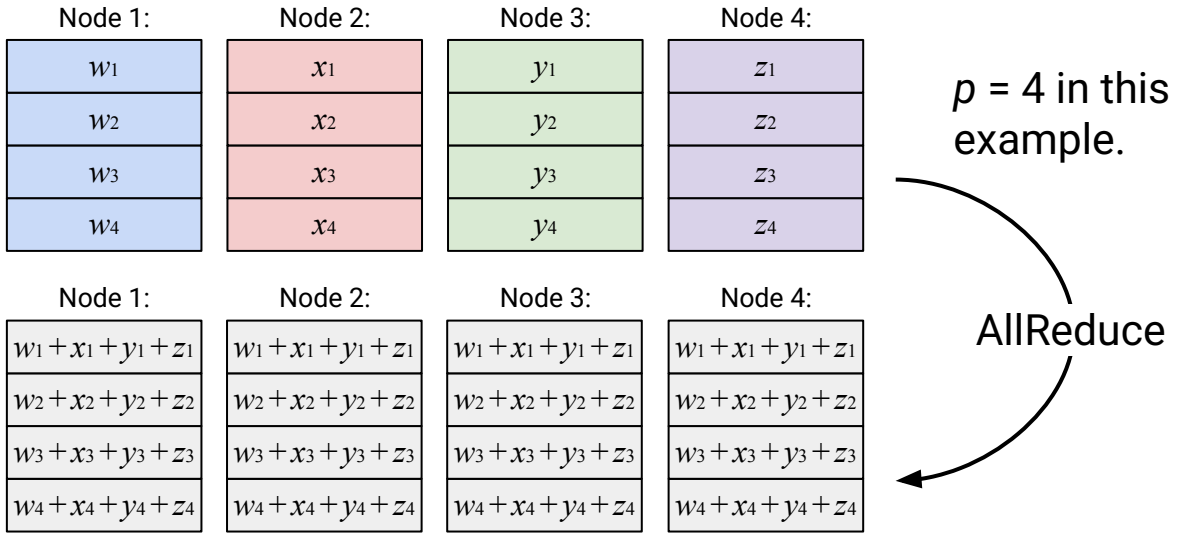
- Each node calls the operation, in parallel, at the same time.
- The operation is blocking: Must wait for all nodes to finish before we can proceed to the next task.

Input: Each node specifies a *p*-element vector of input data.

(e.g. Specify the memory address of a read buffer.)

Output: Each node receives a *p*-element vector of output data.

(e.g. Specify a memory address, and receive output at that write buffer.)



Collective Communication Definition: Recap

To recap, collective operations are:

- **Orchestrated**: A centralized controller plans out the job ahead of time.
- **Synchronized**: Every node starts the operation at the same time.
- **Blocking**: Must wait for all nodes to finish before proceeding.

We'll now define the 7 collective operations.

- First, we'll define the inputs and outputs (the *what*).
- Then, we'll look at how the operation is implemented in the network (the *how*).

The operations can roughly split into two categories:

- **Redistribution** operations move data around without transforming it.
- **Consolidation** operations aggregate many pieces of data into a single output.

Redistribution Operations

Lecture 23, CS 168, Spring 2026

Distributed AI Training

- Motivation
- Infrastructure

Defining Collectives

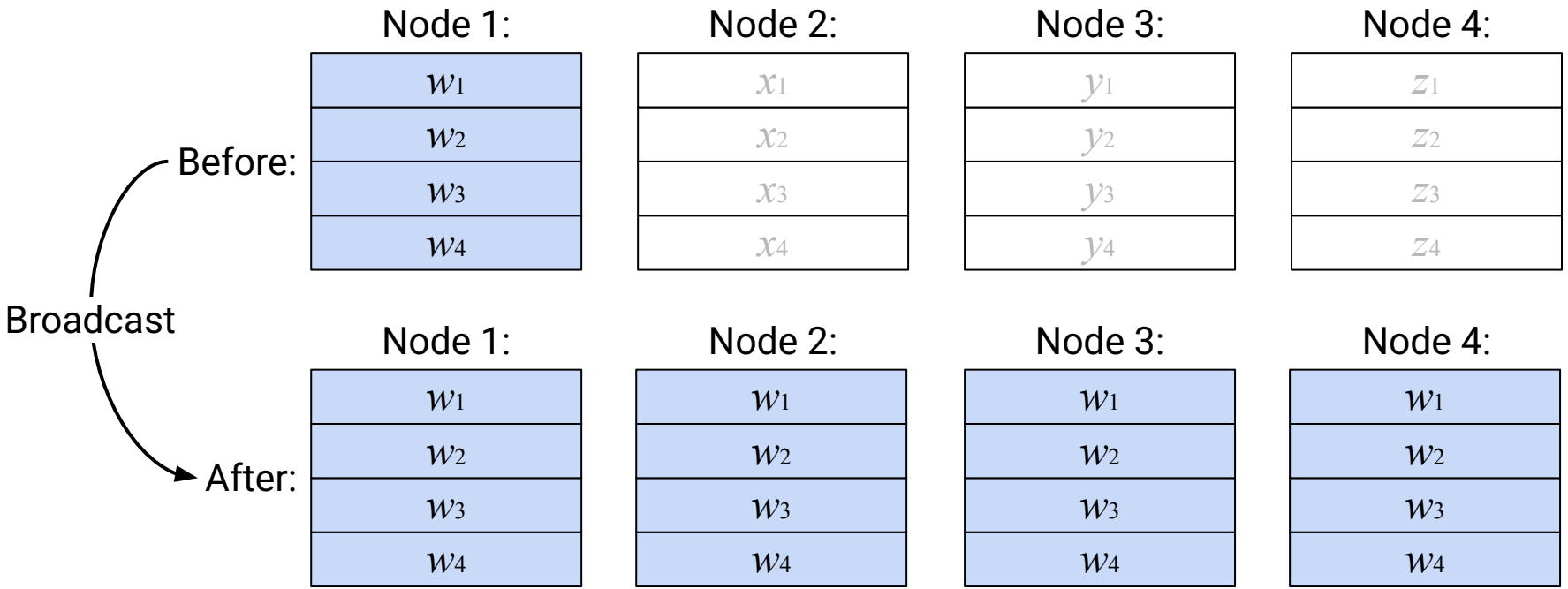
- Definition
- **Redistribution Operations**
- Consolidation Operations
- Duals and Compositions

Implementing Collectives

- Mesh Topology
- Single Root
- Tree
- Ring (Naive)
- Ring (Optimized)
- Overlay/Underlay Topologies

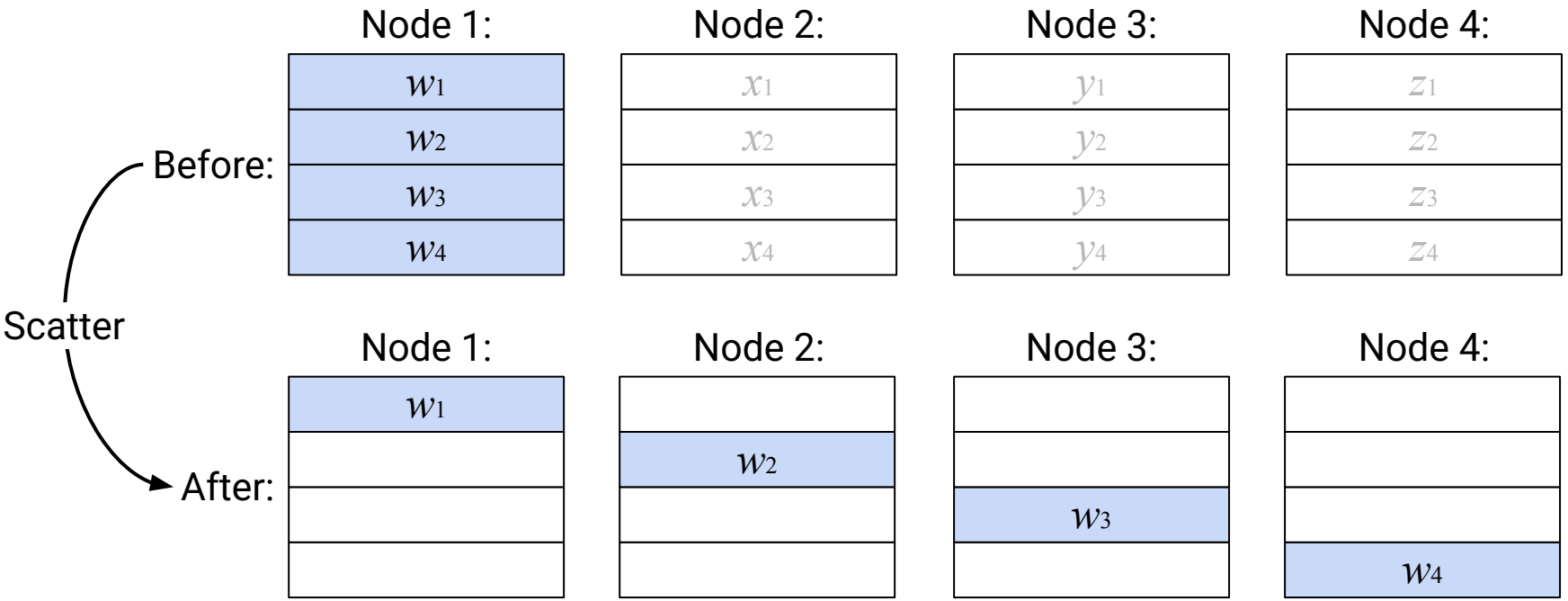
Broadcast

The target node sends its vector to everybody.



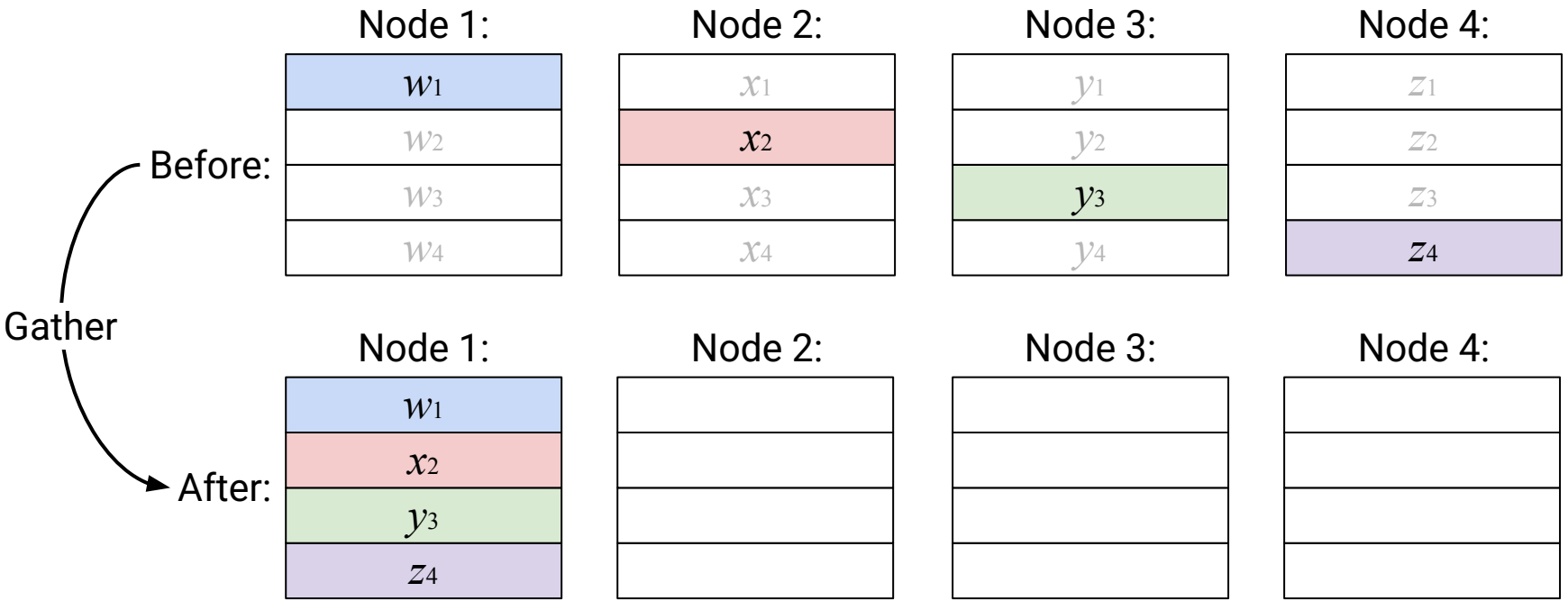
Scatter

The target node sends its *i*th element to the *i*th node.



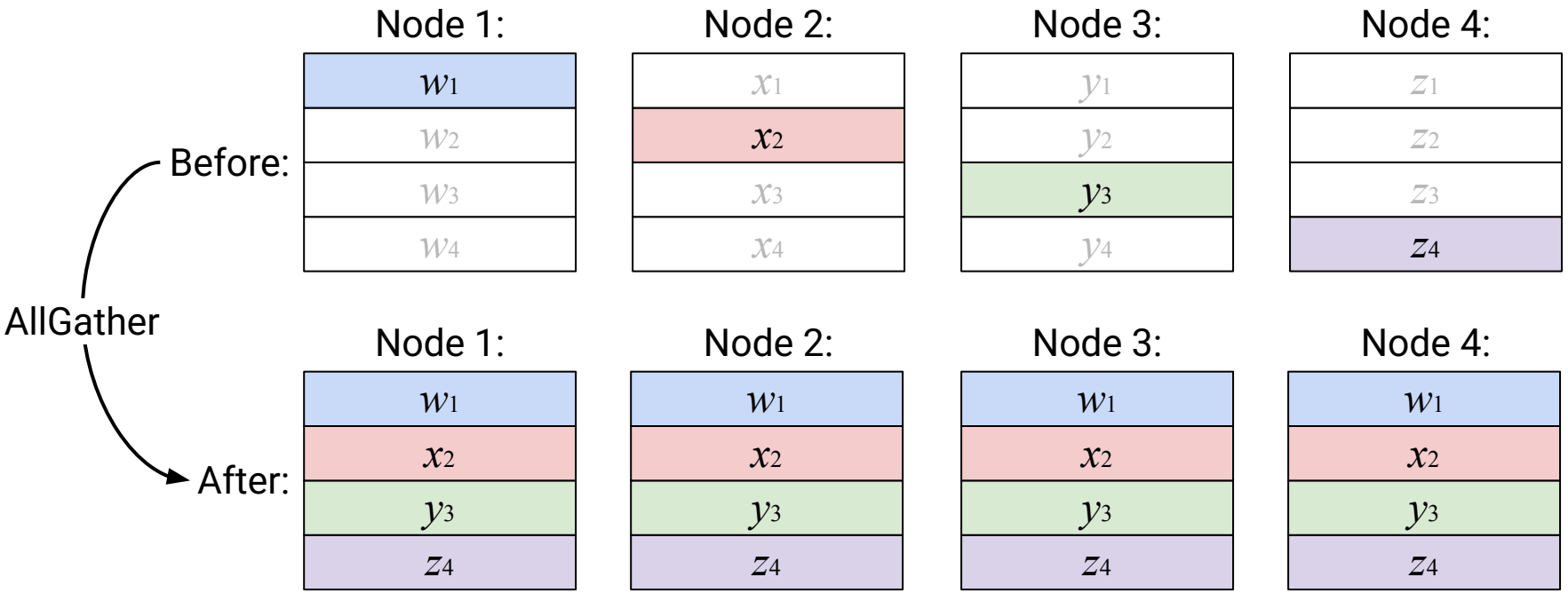
Gather

Node *i* sends its *i*th element to the target node.



AllGather

Node *i* sends its *i*th element to everybody.



Consolidation Operations

Lecture 23, CS 168, Spring 2026

Distributed AI Training

- Motivation
- Infrastructure

Defining Collectives

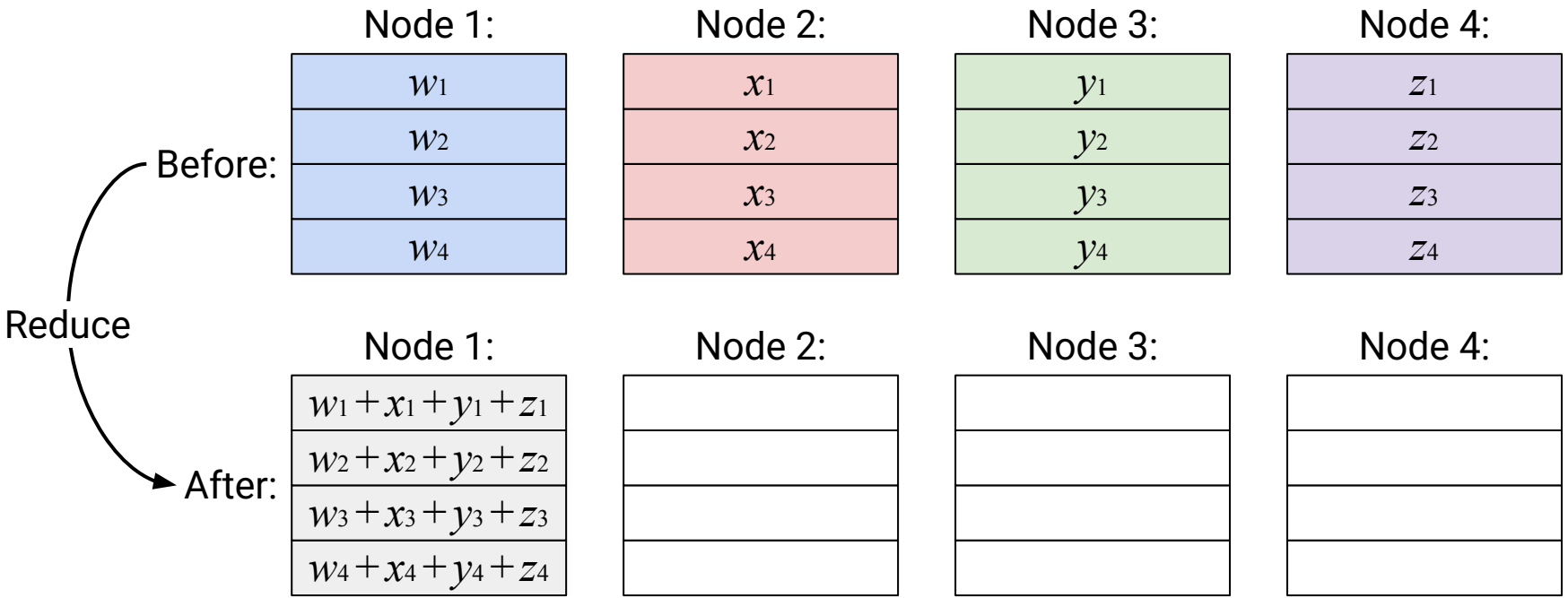
- Definition
- Redistribution Operations
- **Consolidation Operations**
- Duals and Compositions

Implementing Collectives

- Mesh Topology
- Single Root
- Tree
- Ring (Naive)
- Ring (Optimized)
- Overlay/Underlay Topologies

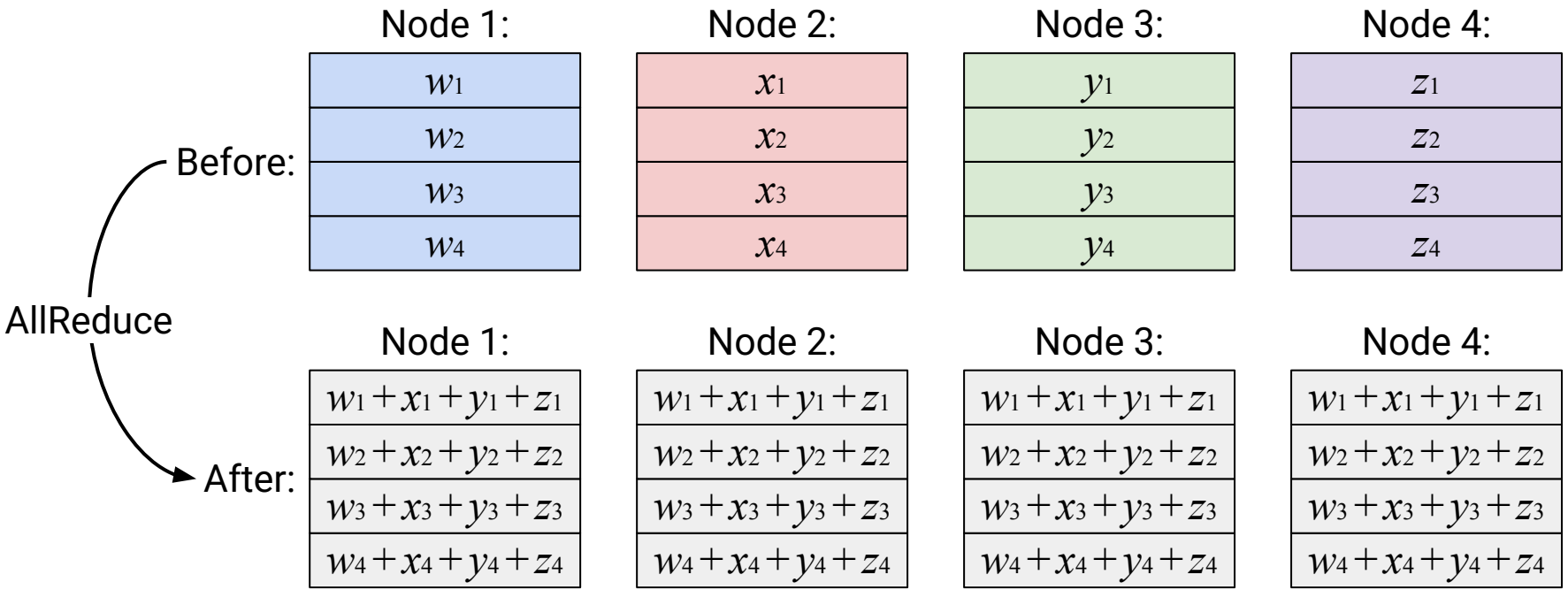
Reduce

Sum up the vectors, then send the resulting sum vector to a target node.



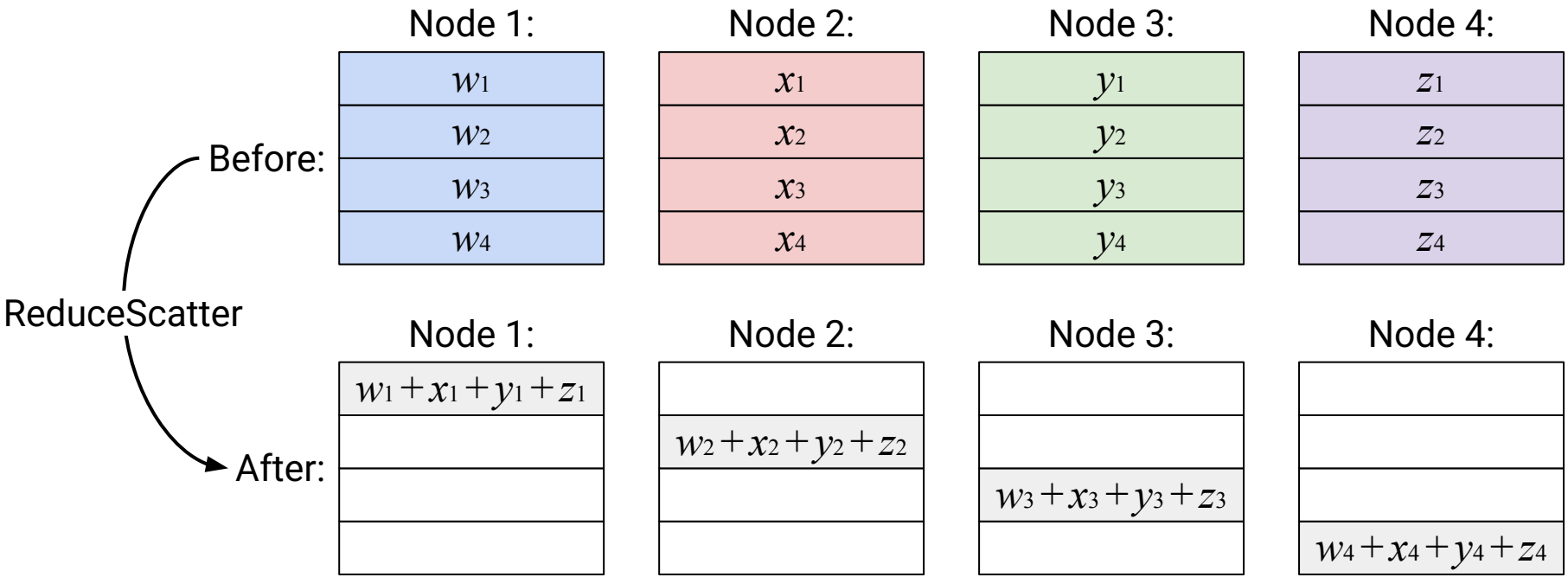
AllReduce

Sum up the vectors, then send the resulting sum vector to all nodes.



ReduceScatter

The *i*th node has the sum of the *i*th elements of each vector.



Duals and Compositions

Lecture 23, CS 168, Spring 2026

Distributed AI Training

- Motivation
- Infrastructure

Defining Collectives

- Definition
- Redistribution Operations
- Consolidation Operations
- **Duals and Compositions**

Implementing Collectives

- Mesh Topology
- Single Root
- Tree
- Ring (Naive)
- Ring (Optimized)
- Overlay/Underlay Topologies

Duals

Some operations are **duals** of each other: One operation is the reverse of the other.

- Analogy: In math, x^2 and $\sqrt{x}$ are duals.

When checking if operations are duals, ignore any reduction operations.

- Just look at which boxes we read from, and which boxes we write to.

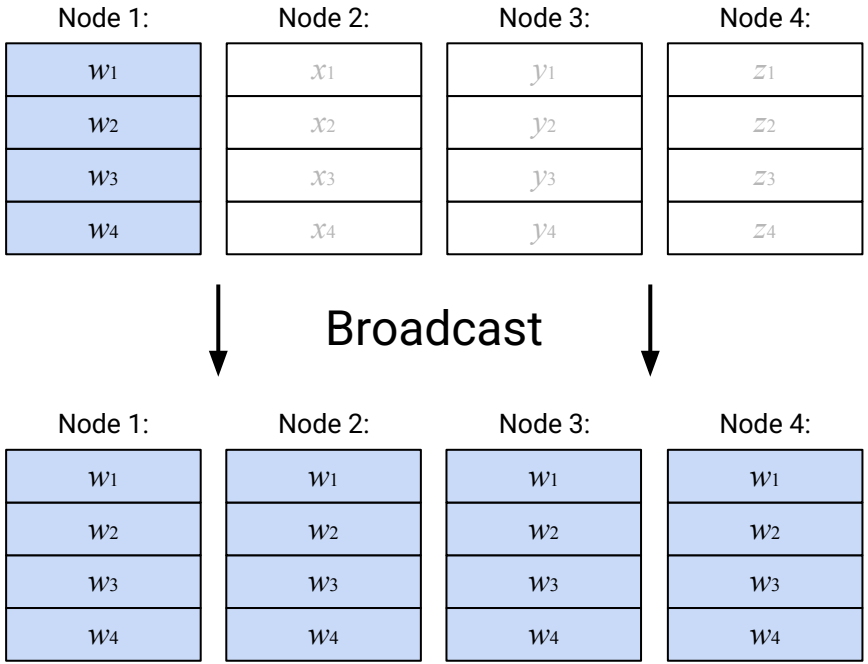
Duals are useful when we think about implementing these collectives.

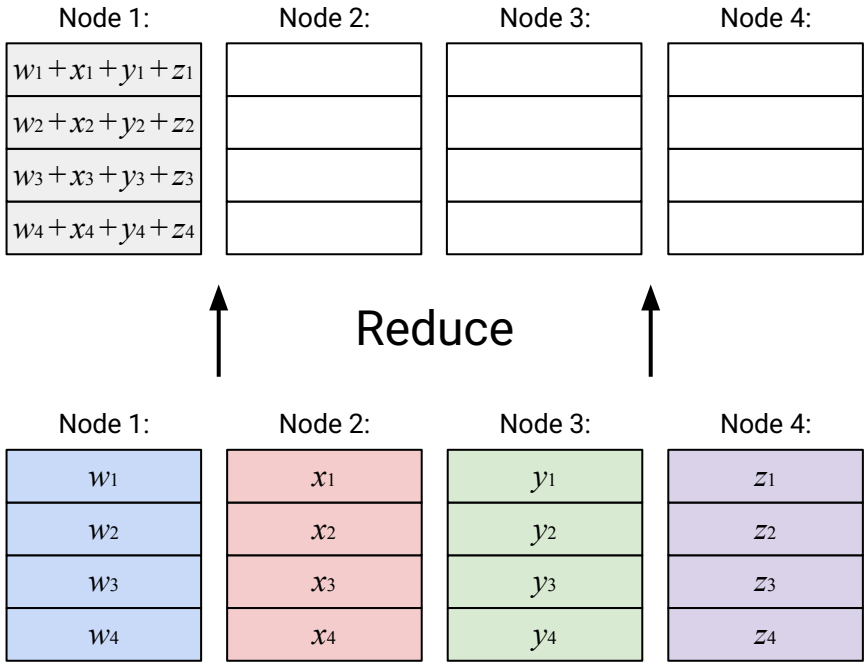
- All else equal, two dual operations use the same bandwidth.
- Total amount of data sent/received is the same in both operations, because we write/read from the same boxes.

Duals

Broadcast and Reduce are duals.

- Broadcast: Read from 4 boxes in root node. Write to all 16 boxes.
- Reduce: Read from 16 boxes. Write to 4 boxes in root node.

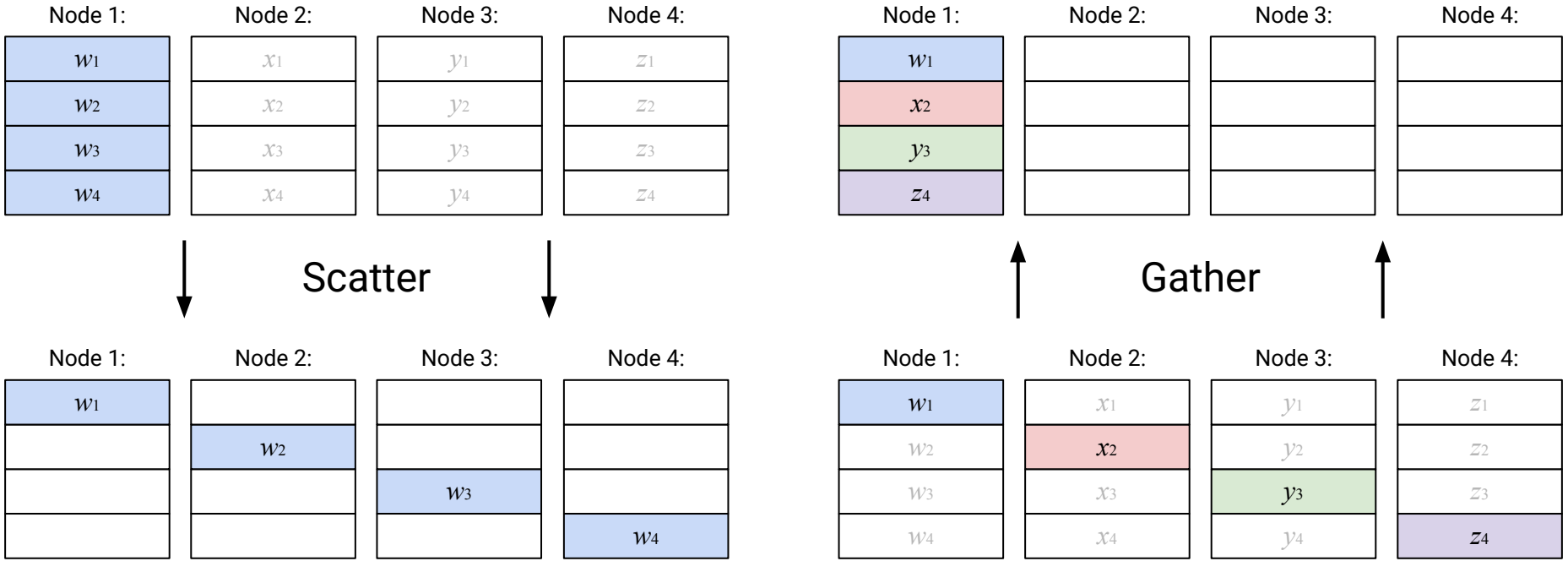




Duals

Scatter and Gather are duals.

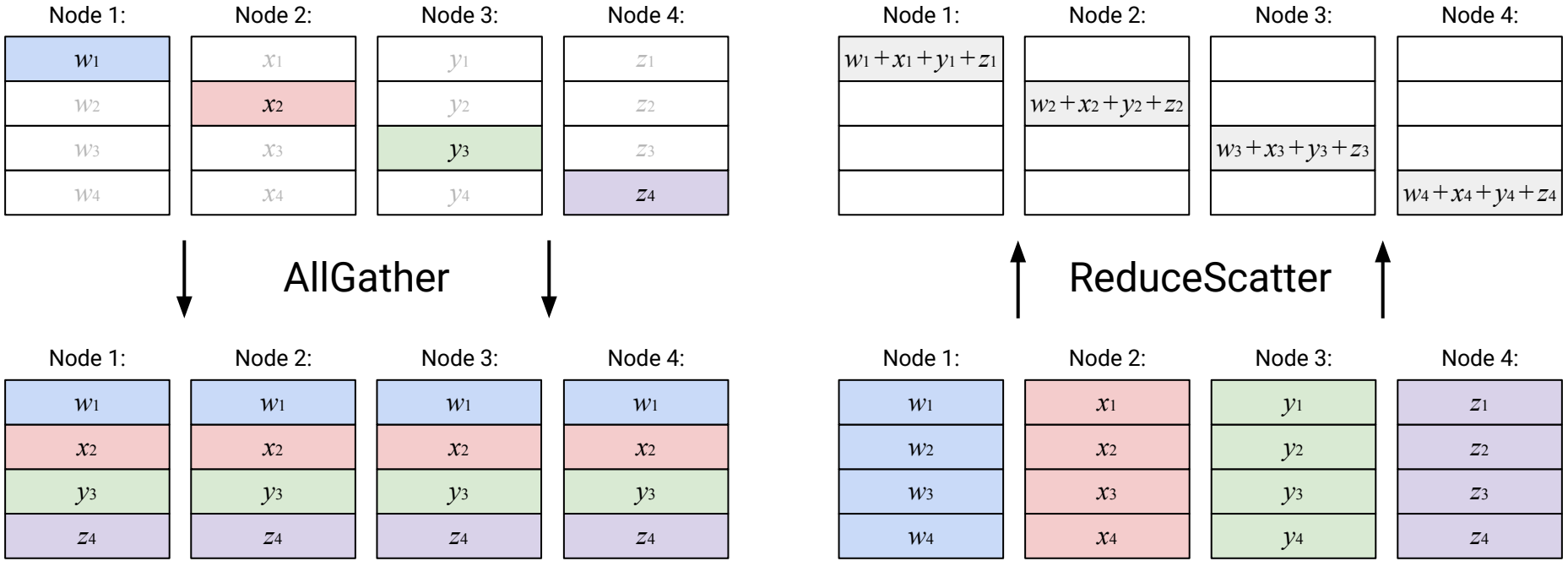
- Scatter: Read from 4 boxes in root node. Write to the *i*th box in *i*th node.
- Gather: Read from the *i*th box in *i*th node. Write to 4 boxes in root node.



Duals

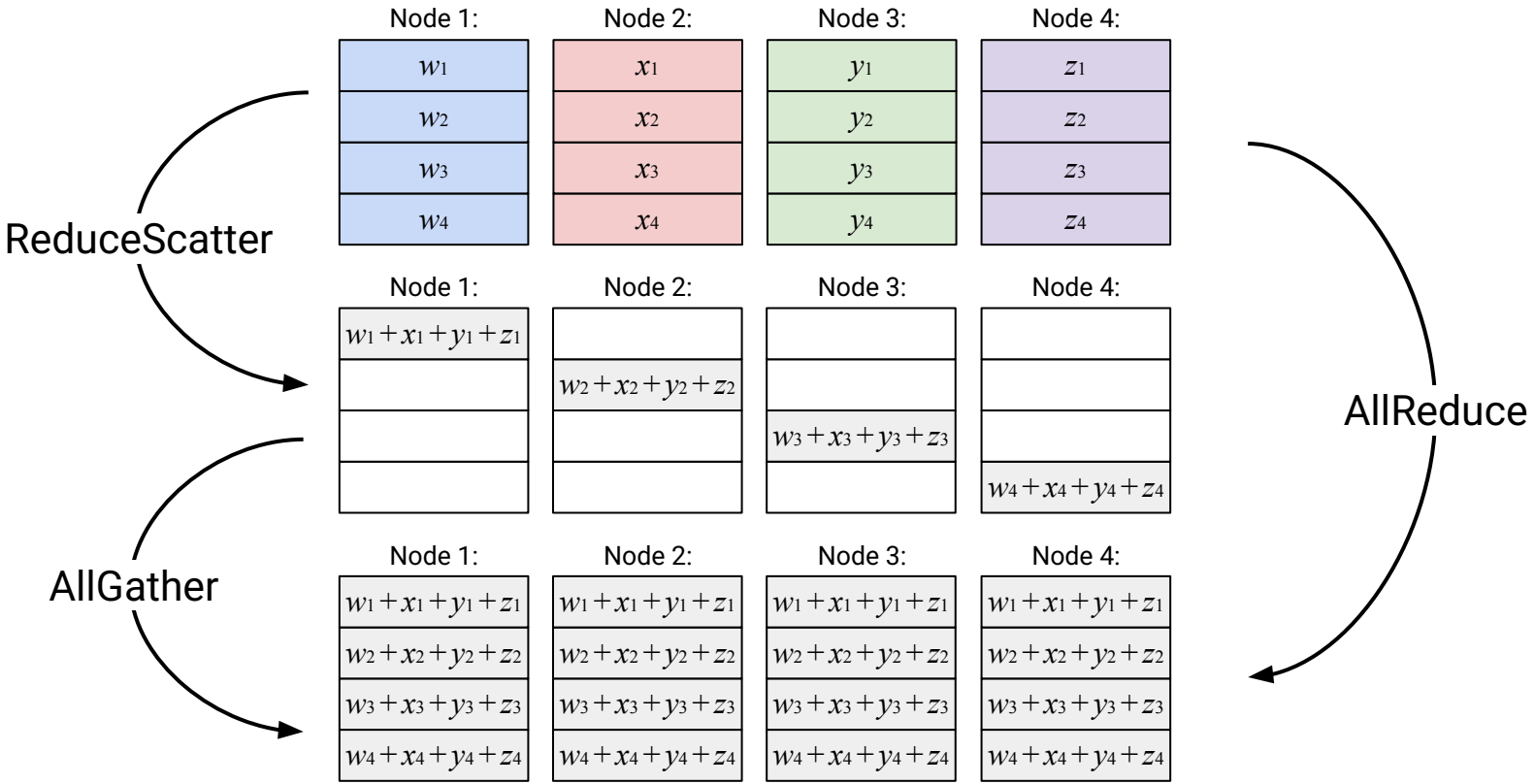
AllGather and ReduceScatter are duals.

- AllGather: Read from the *i*th box in *i*th node. Write to all 16 boxes.
- Reduce: Read from all 16 boxes. Write to the *i*th box in *i*th node.



Compositing Operations

AllReduce = ReduceScatter + AllGather.



AllReduce with Mesh Topology

Lecture 23, CS 168, Spring 2026

Distributed AI Training

- Motivation
- Infrastructure

Defining Collectives

- Definition
- Redistribution Operations
- Consolidation Operations
- Duals and Compositions

Implementing Collectives

- **Mesh Topology**
- Single Root
- Tree
- Ring (Naive)
- Ring (Optimized)
- Overlay/Underlay Topologies

Implementing Collective Operations

So far: We've defined the input and desired output of each operation.

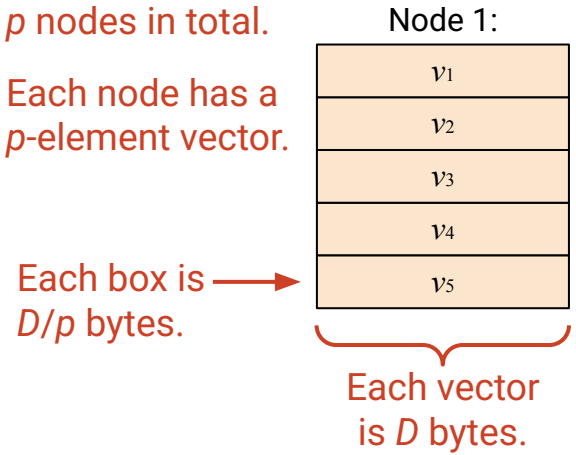
Up next: How do we implement these operations?

- What topology do we use to connect the nodes?
- What data does each node have to send to the other nodes?

Our choices will affect the performance of the operation.

- Total amount of bandwidth used?
- Number of steps for operation to complete?
- Bandwidth each node sends per step?

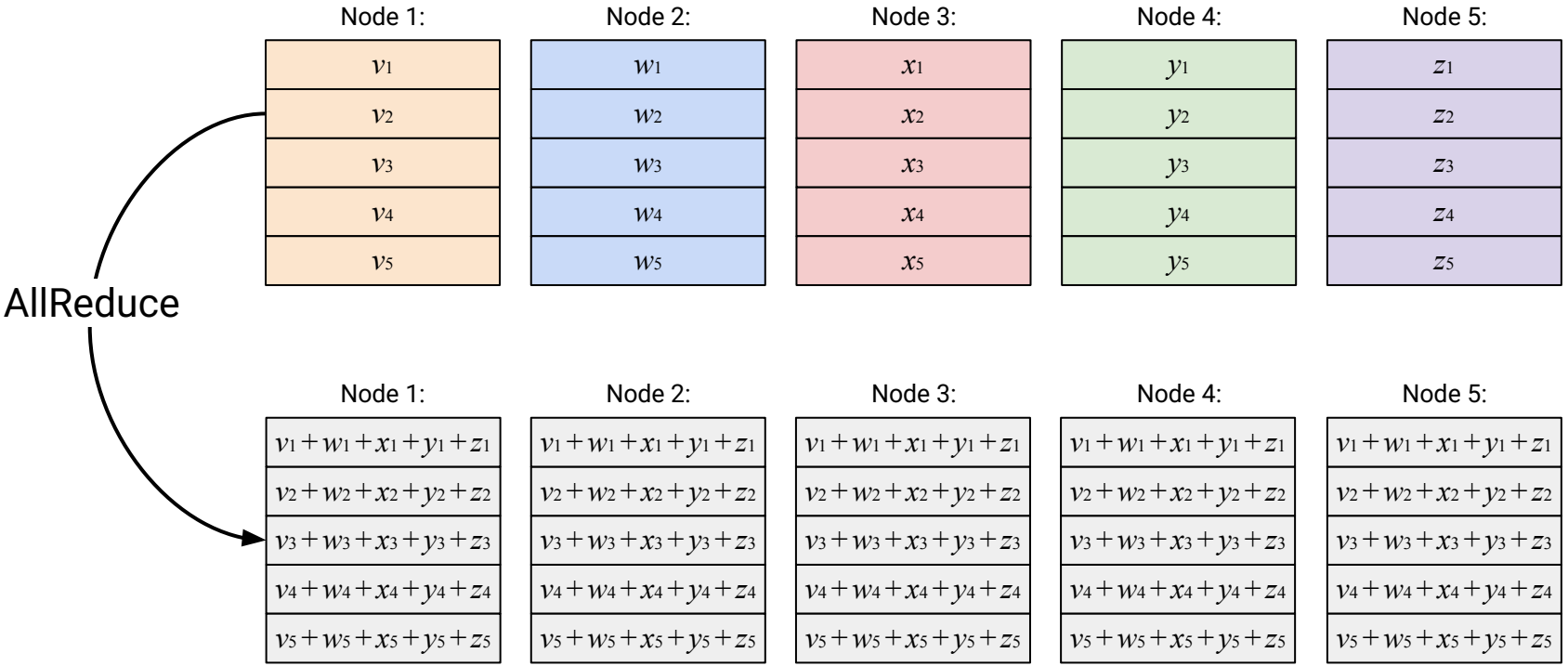
We'll use AllReduce as our running example.



Reminder: AllReduce Operation

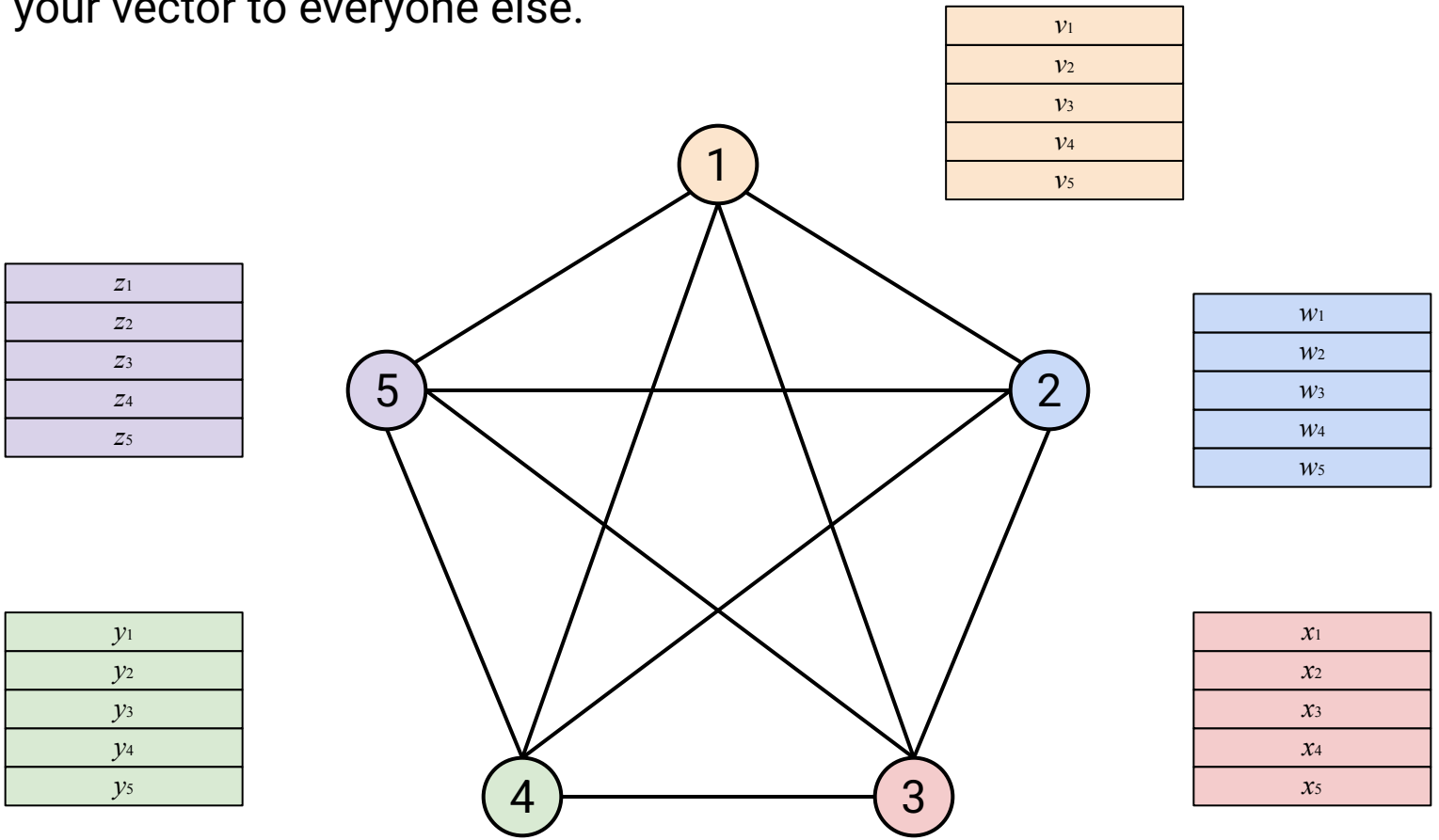
Sum up the vectors, then send the resulting sum vector to all nodes.

For this demo, we'll use $p = 5$ nodes instead of $p = 4$ nodes.



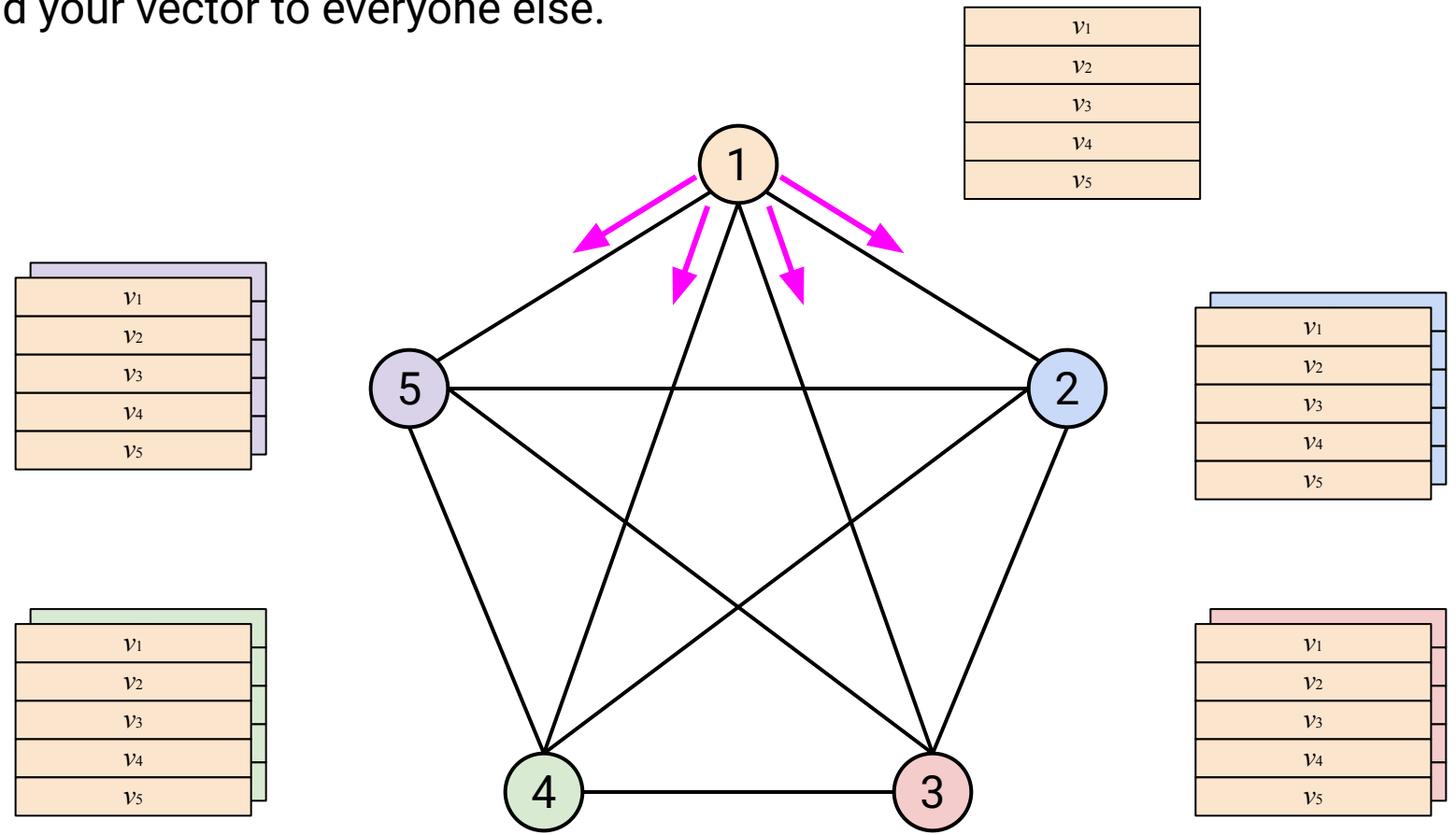
Mesh Topology

1. Send your vector to everyone else.



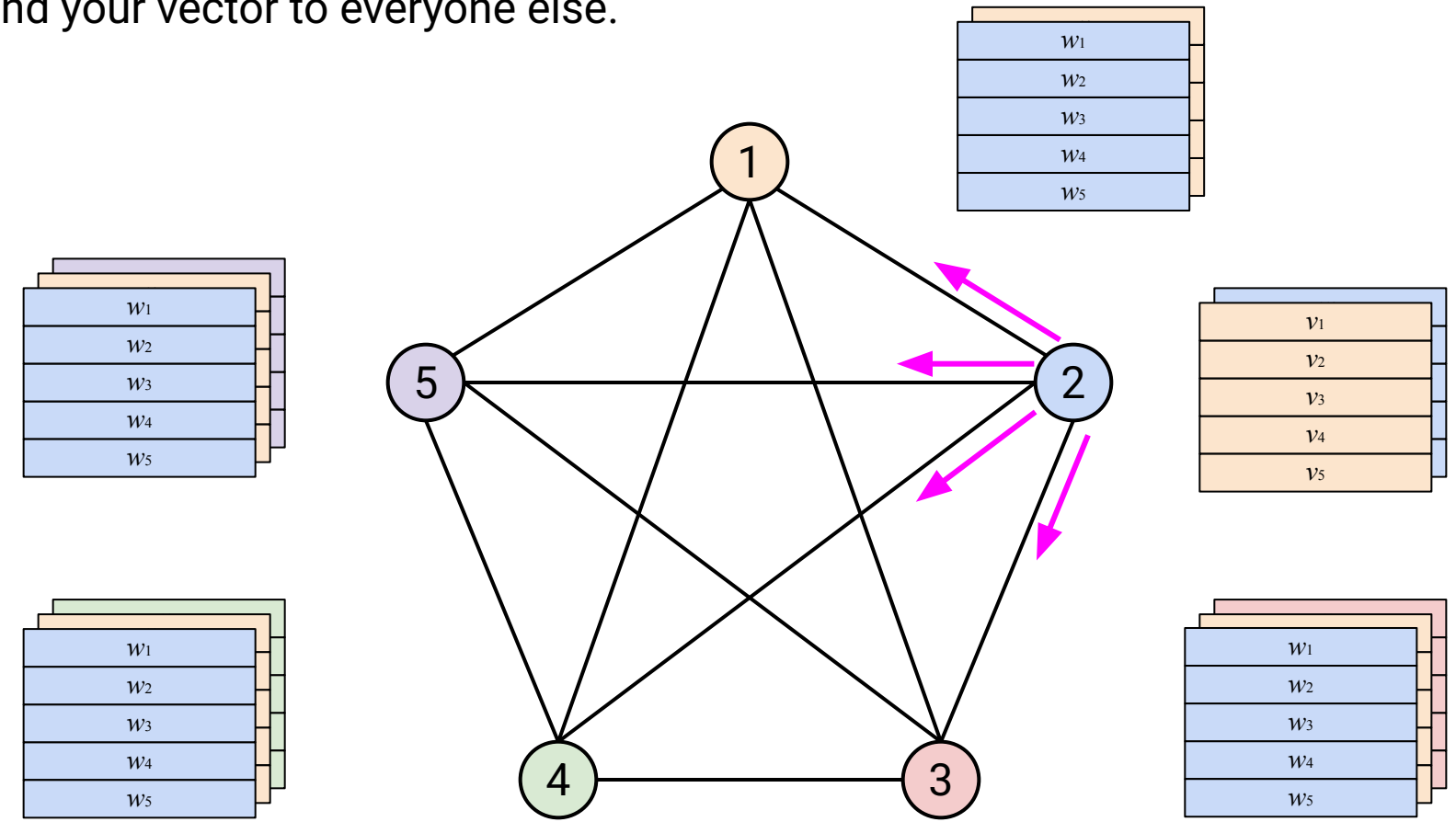
Mesh Topology

1. Send your vector to everyone else.



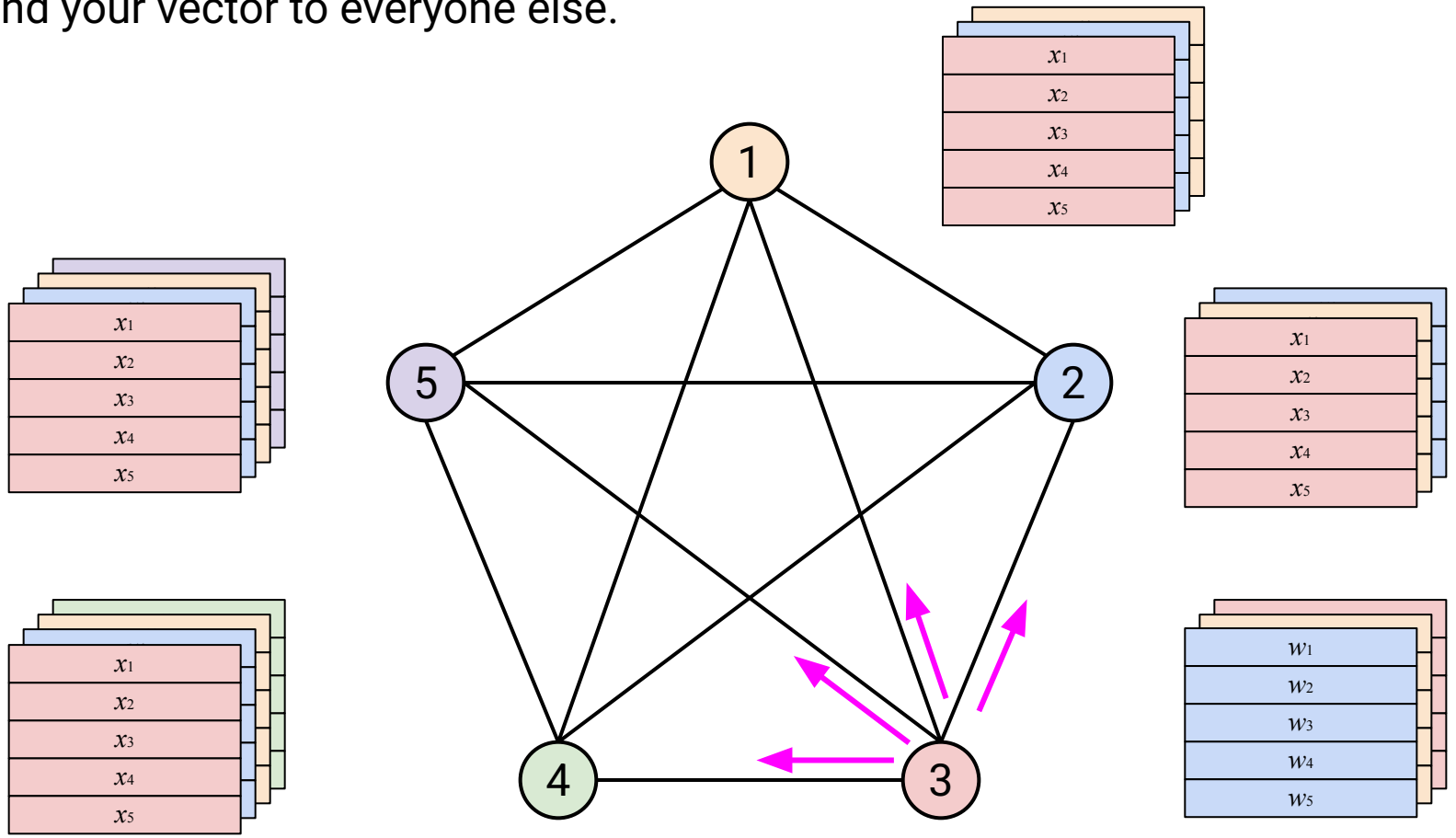
Mesh Topology

1. Send your vector to everyone else.



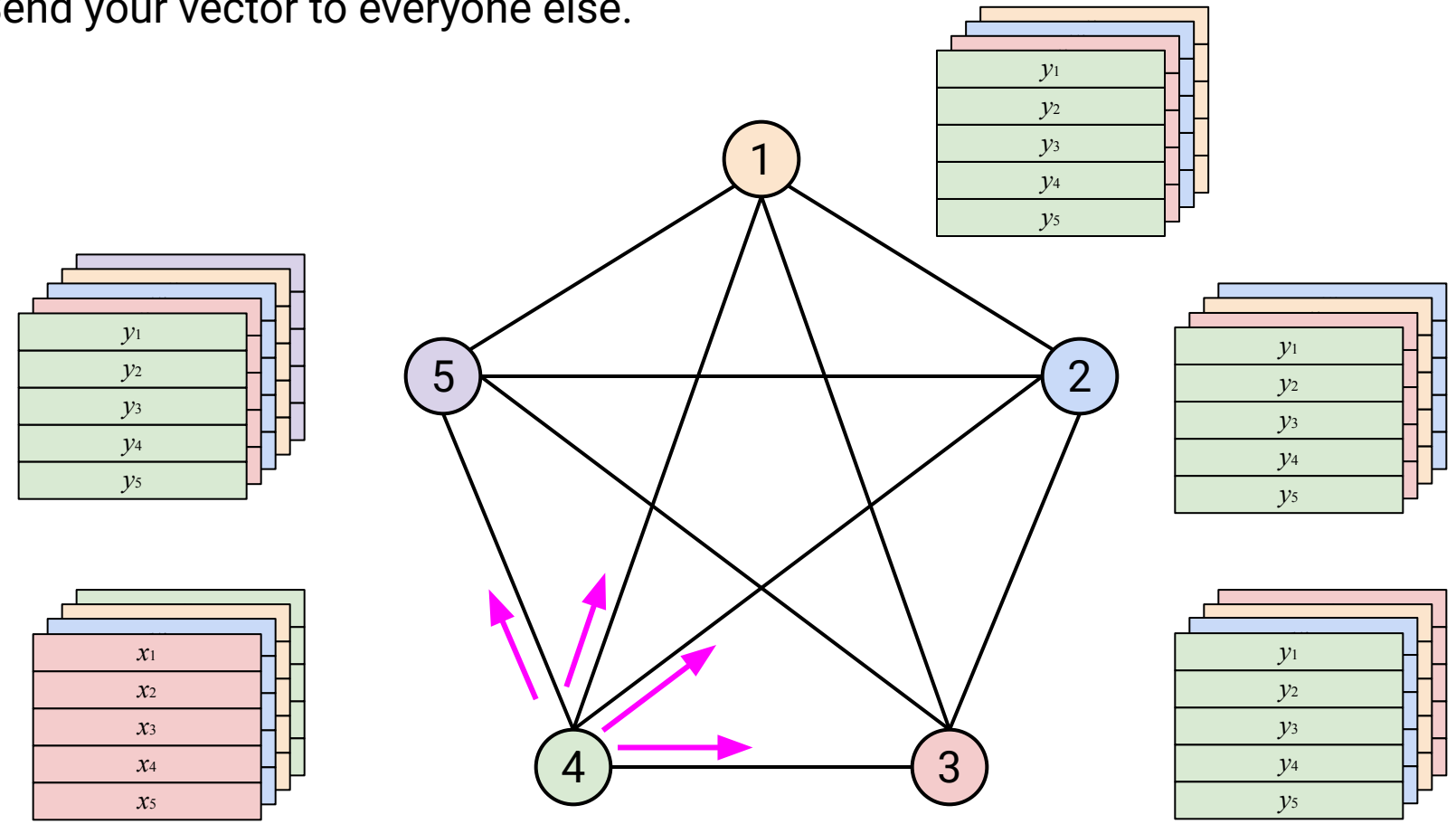
Mesh Topology

1. Send your vector to everyone else.



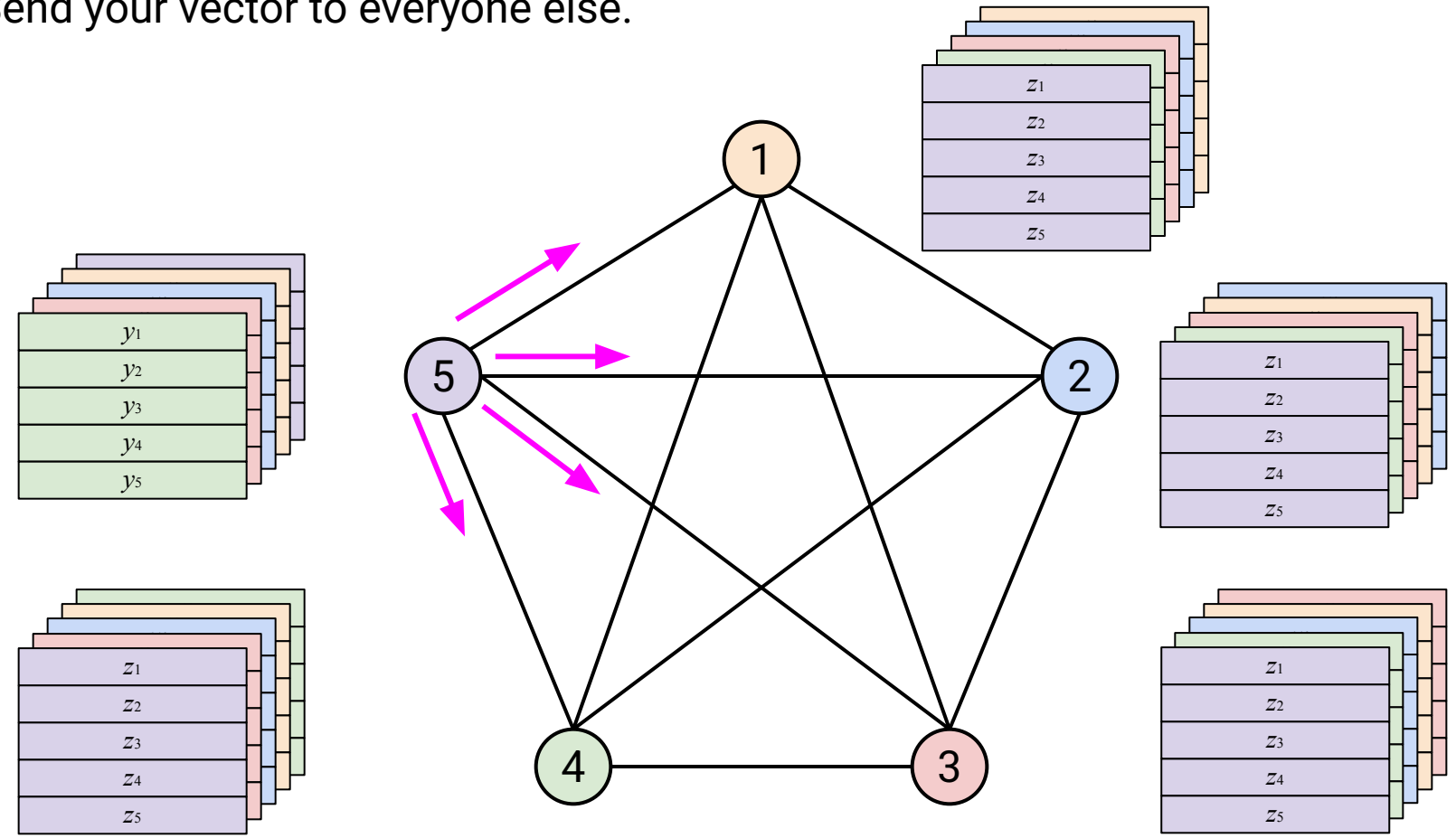
Mesh Topology

1. Send your vector to everyone else.



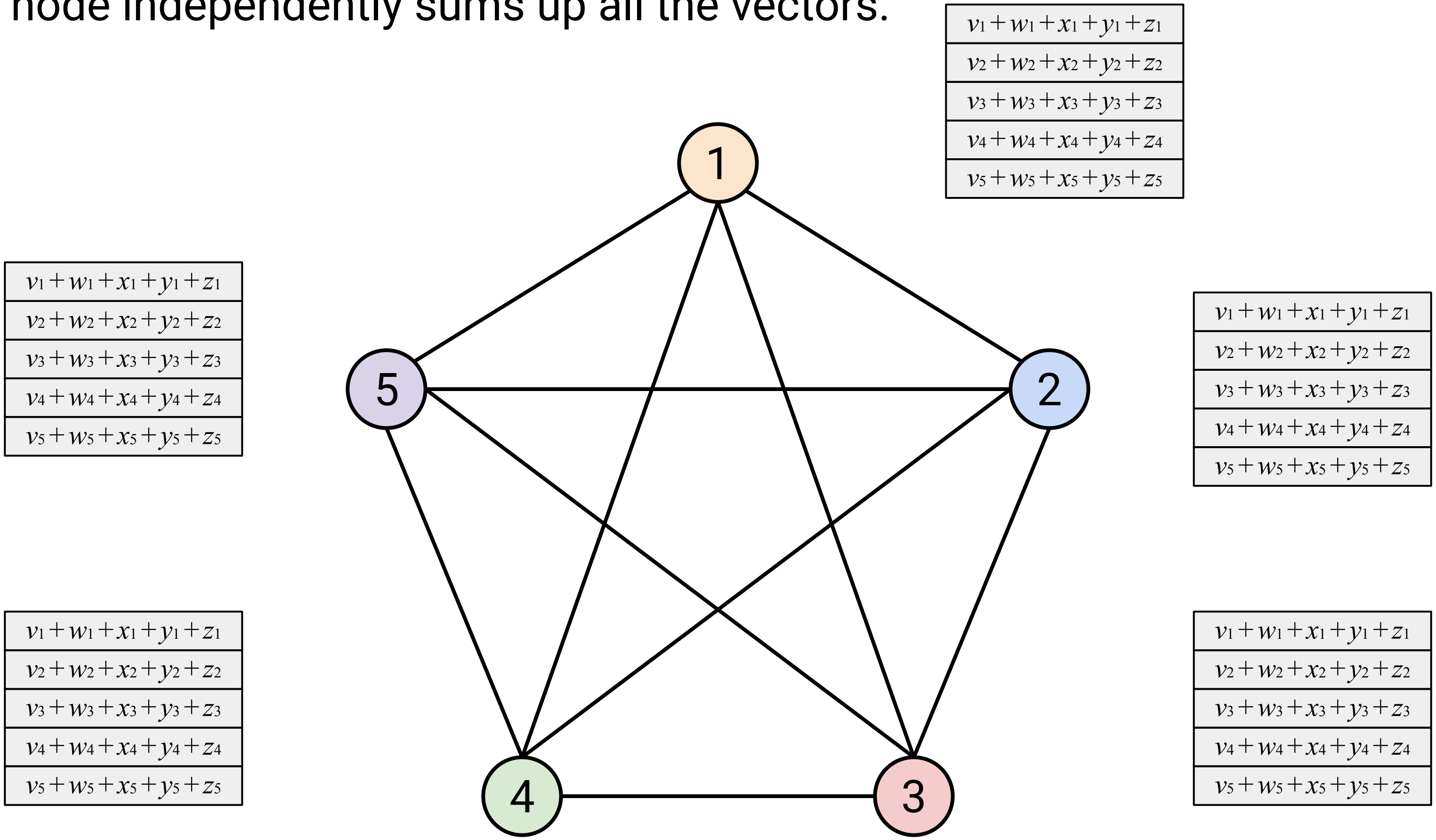
Mesh Topology

1. Send your vector to everyone else.



Mesh Topology

2. Each node independently sums up all the vectors.



Mesh Topology: Efficiency

Total bandwidth: Each node sends its whole vector (D bytes) to all $p-1$ other nodes. There are p nodes in total.

Number of steps: Everyone can simultaneously send their vector to everyone else.

Bandwidth per step: Send D bytes to $p-1$ other nodes.

	Total Bandwidth	Number of Steps	Bandwidth Per Step
Mesh	Dp^2	1	Dp

AllReduce with Single Root

Lecture 23, CS 168, Spring 2026

Distributed AI Training

- Motivation
- Infrastructure

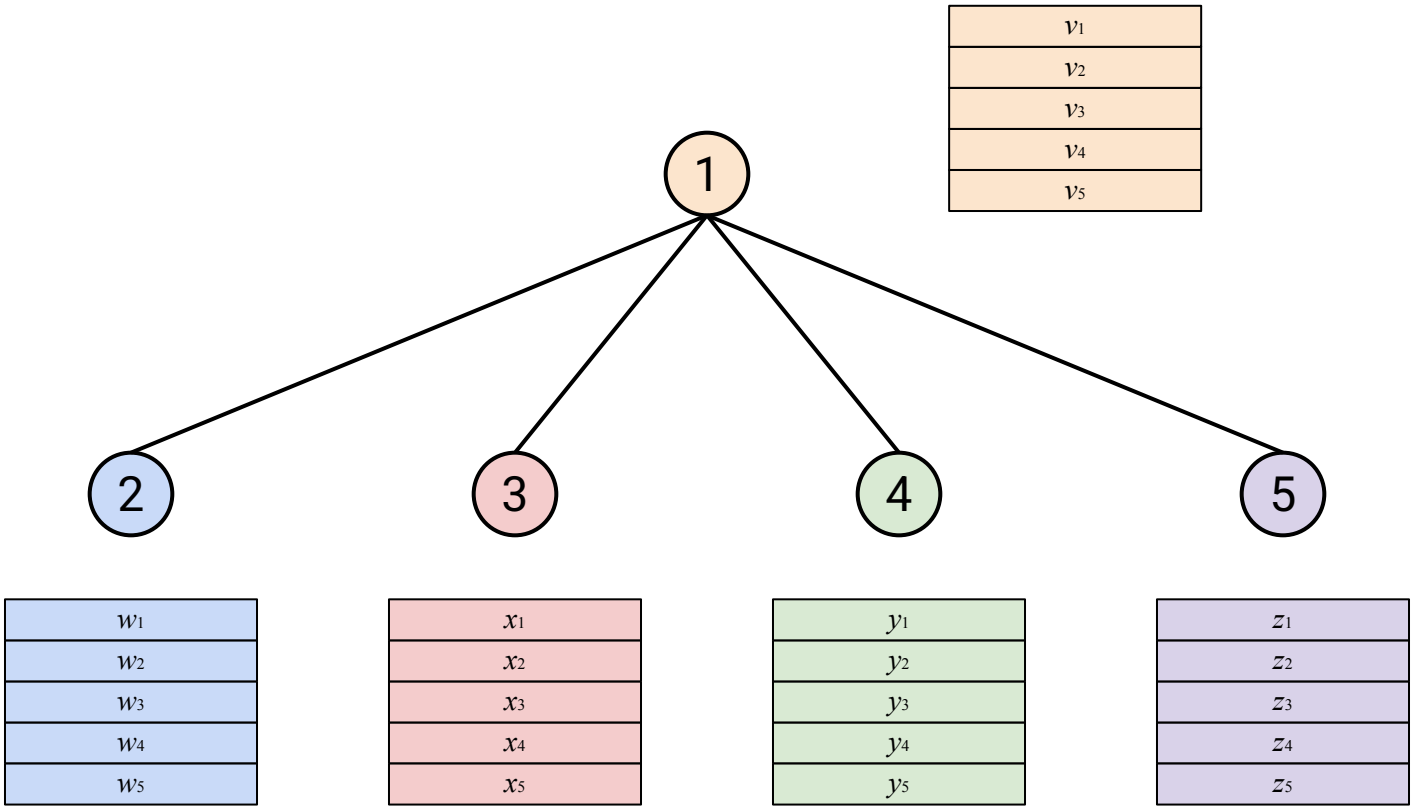
Defining Collectives

- Definition
- Redistribution Operations
- Consolidation Operations
- Duals and Compositions

Implementing Collectives

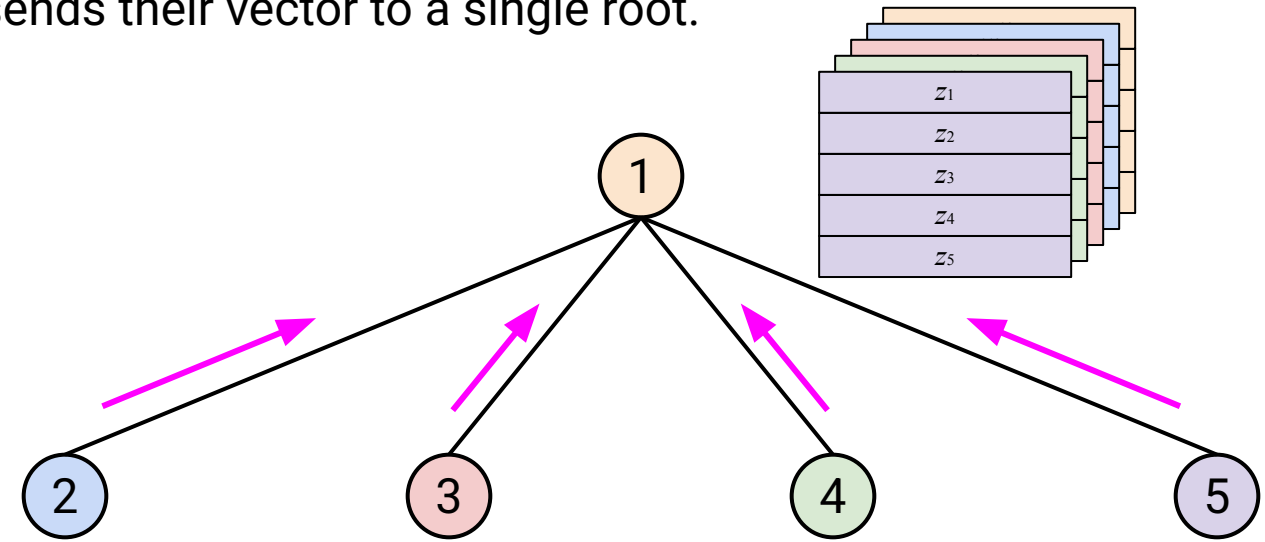
- Mesh Topology
- **Single Root**
- Tree
- Ring (Naive)
- Ring (Optimized)
- Overlay/Underlay Topologies

Mesh Topology



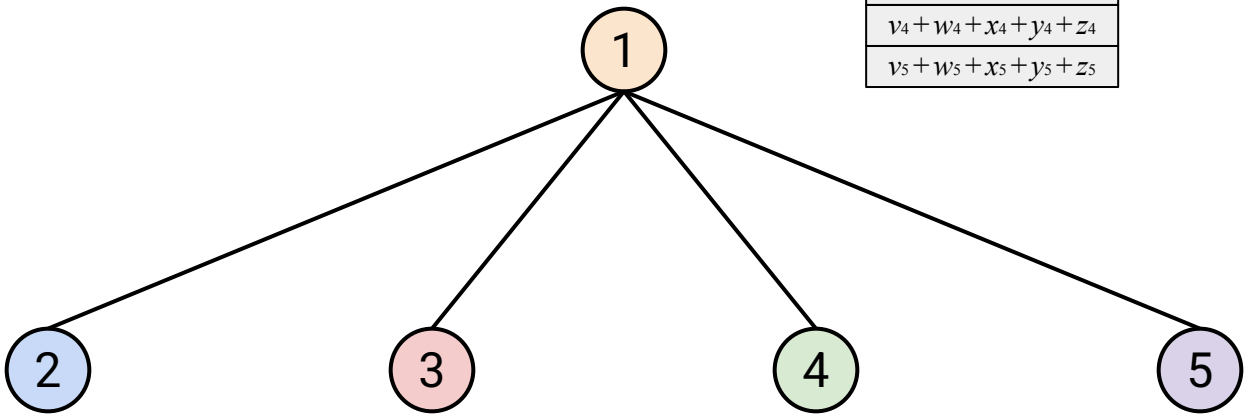
Mesh Topology

1. Everyone sends their vector to a single root.



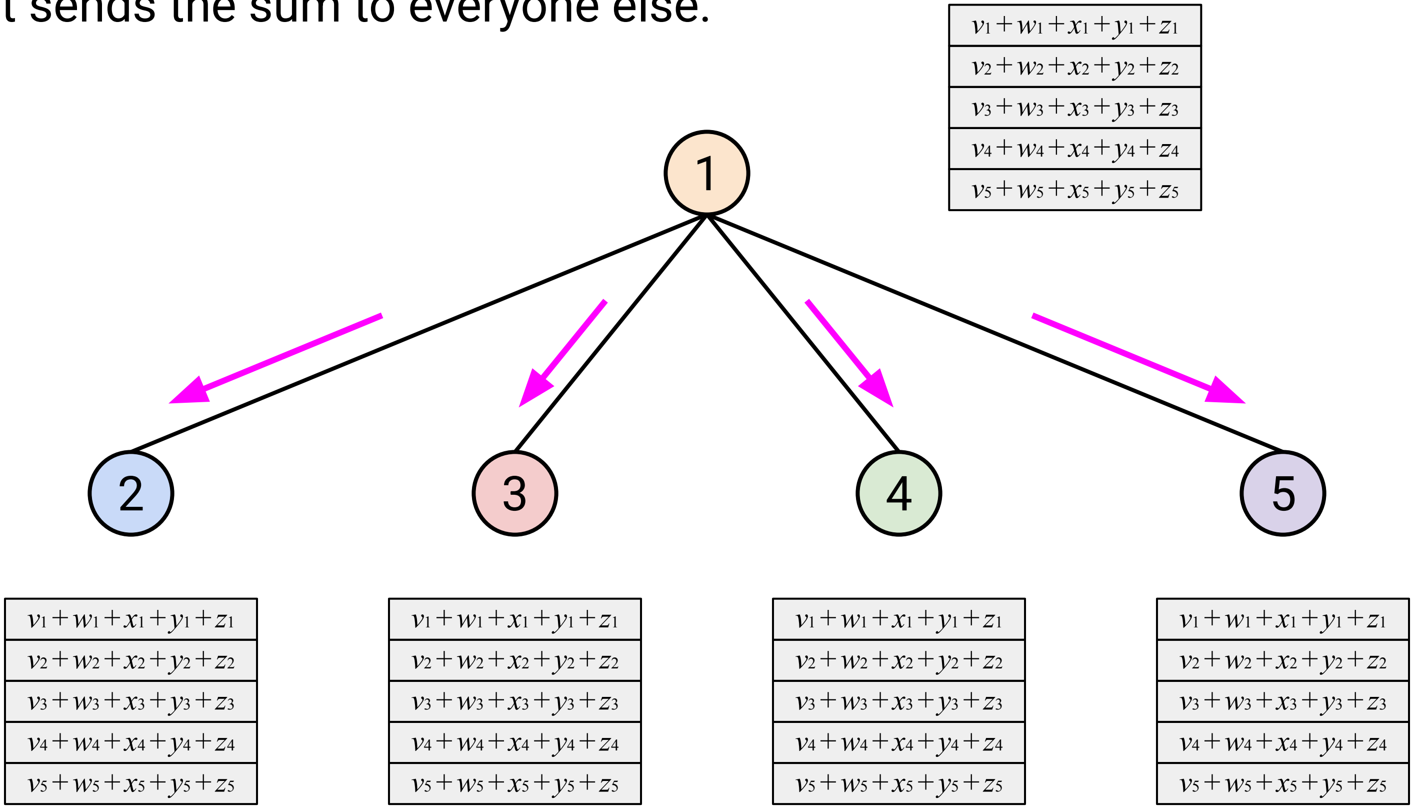
2. The root computes the sum.

$v_1 + w_1 + x_1 + y_1 + z_1$
$v_2 + w_2 + x_2 + y_2 + z_2$
$v_3 + w_3 + x_3 + y_3 + z_3$
$v_4 + w_4 + x_4 + y_4 + z_4$
$v_5 + w_5 + x_5 + y_5 + z_5$



Mesh Topology

3. The root sends the sum to everyone else.



Single Root Topology: Efficiency

Total bandwidth: Each node sends its whole vector (D bytes) to the root. There are p nodes in total.

Number of steps: One step to send to root, and one step to distribute sum from root.

Bandwidth per step: Root has to receive D bytes from the $p-1$ other nodes.

	Total Bandwidth	Number of Steps	Bandwidth Per Step
Mesh	Dp^2	1	Dp
Single Root	Dp	2	Dp

Tree-Based AllReduce

Lecture 23, CS 168, Spring 2026

Distributed AI Training

- Motivation
- Infrastructure

Defining Collectives

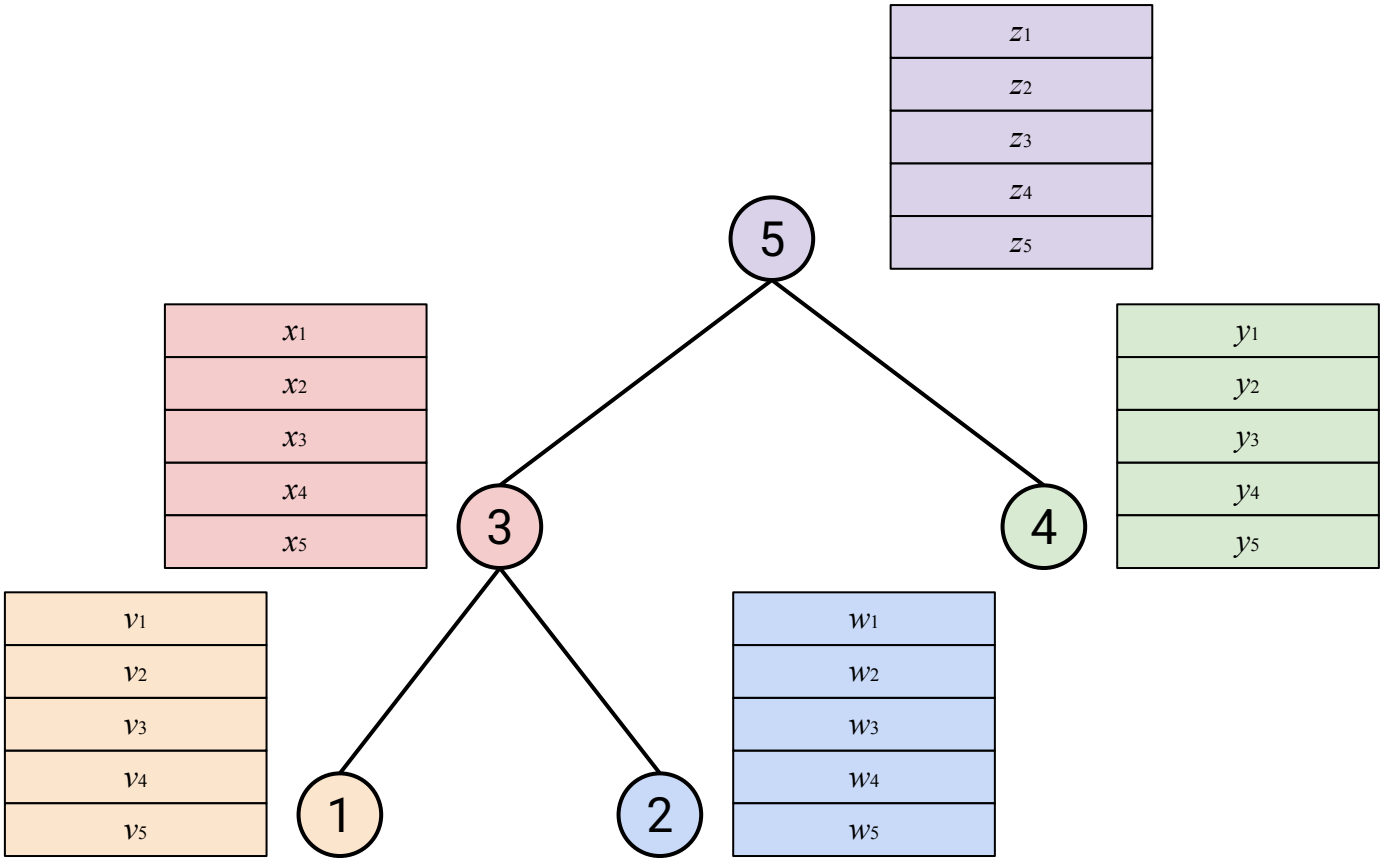
- Definition
- Redistribution Operations
- Consolidation Operations
- Duals and Compositions

Implementing Collectives

- Mesh Topology
- Single Root
- **Tree**
- Ring (Naive)
- Ring (Optimized)
- Overlay/Underlay Topologies

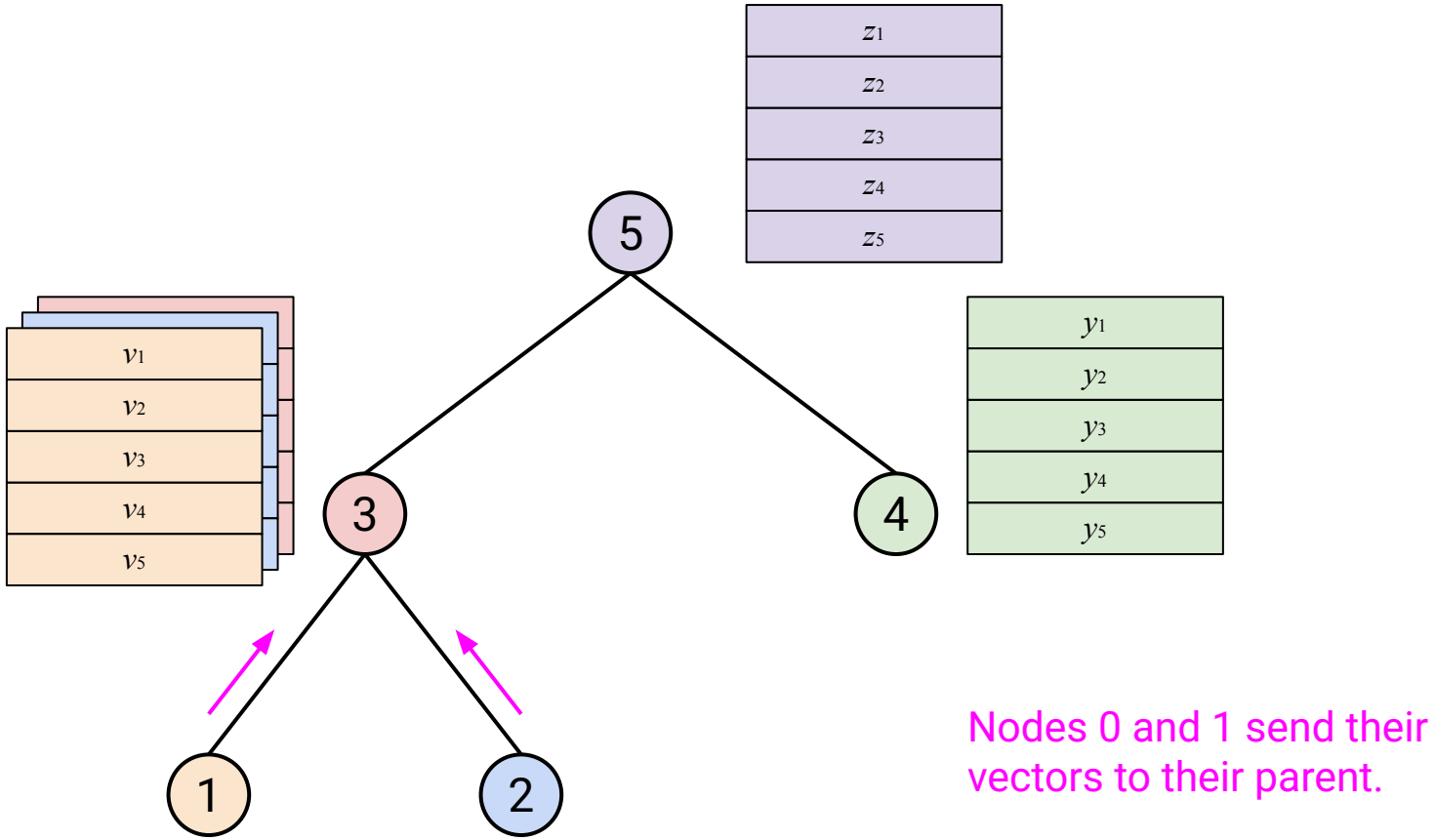
Tree-Based AllReduce

1. Send your vector to your parent. When you receive vector(s), sum them up.



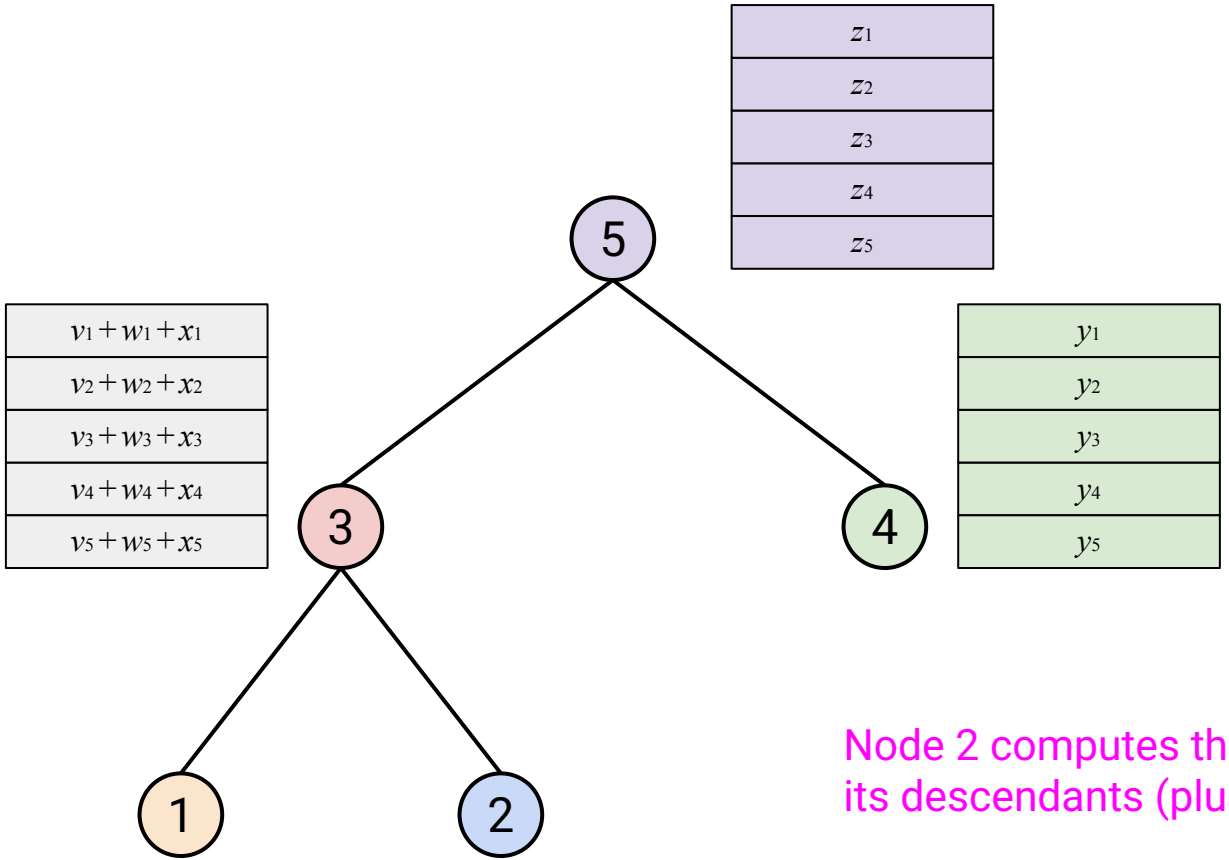
Tree-Based AllReduce

1. Send your vector to your parent. When you receive vector(s), sum them up.



Tree-Based AllReduce

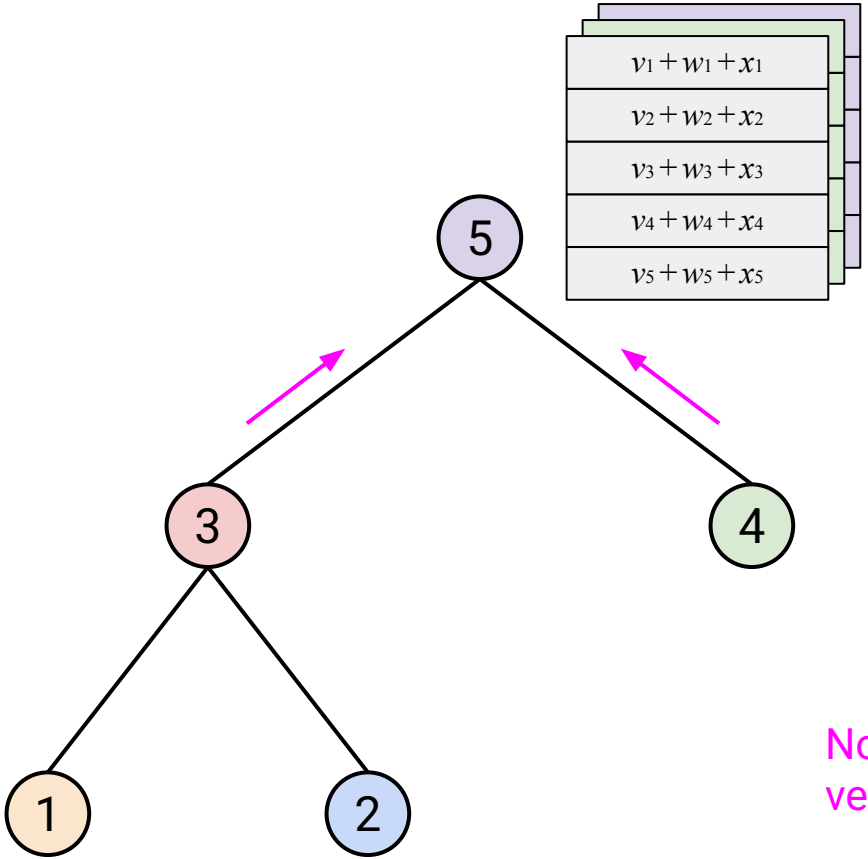
1. Send your vector to your parent. When you receive vector(s), sum them up.



Node 2 computes the sum of all its descendants (plus itself).

Tree-Based AllReduce

1. Send your vector to your parent. When you receive vector(s), sum them up.

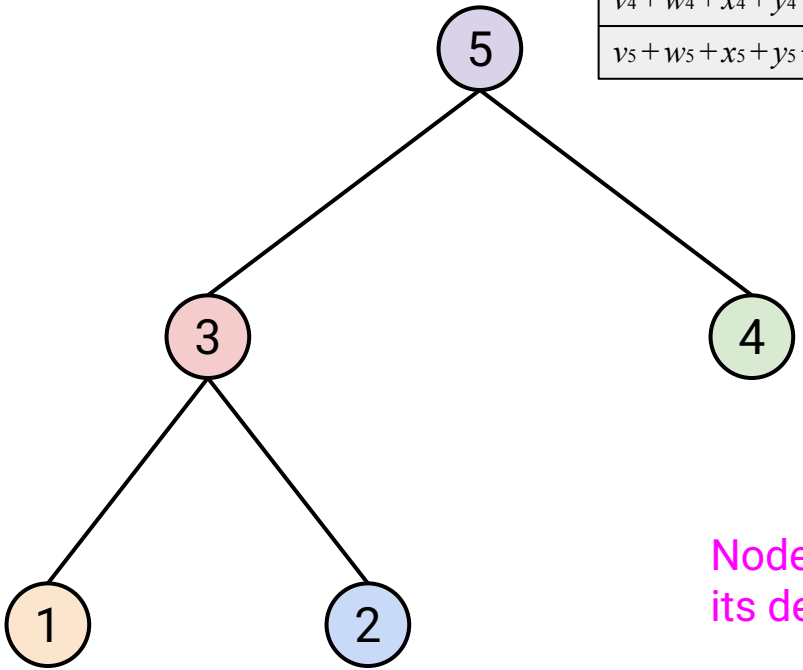


Nodes 2 and 3 send their vectors to their parent.

Tree-Based AllReduce

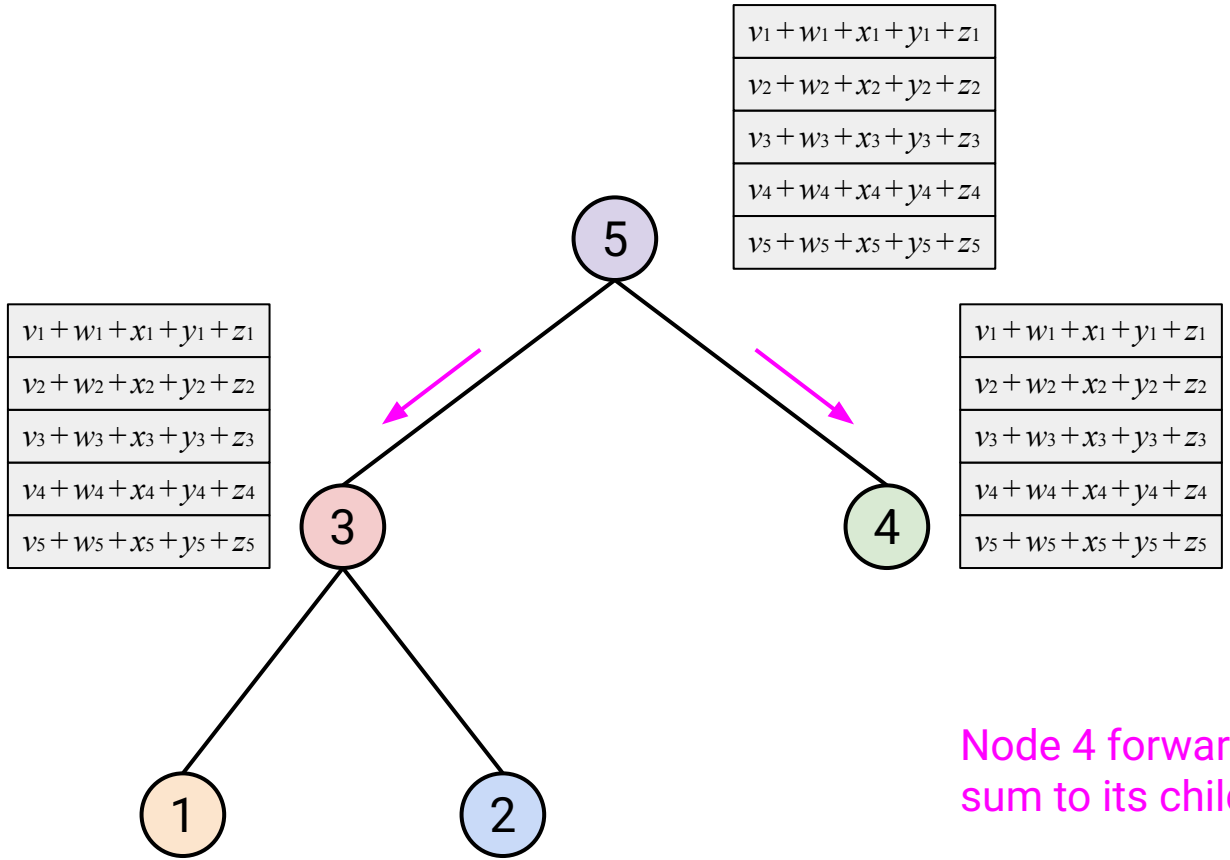
1. Send your vector to your parent. When you receive vector(s), sum them up.

$v_1 + w_1 + x_1 + y_1 + z_1$
$v_2 + w_2 + x_2 + y_2 + z_2$
$v_3 + w_3 + x_3 + y_3 + z_3$
$v_4 + w_4 + x_4 + y_4 + z_4$
$v_5 + w_5 + x_5 + y_5 + z_5$



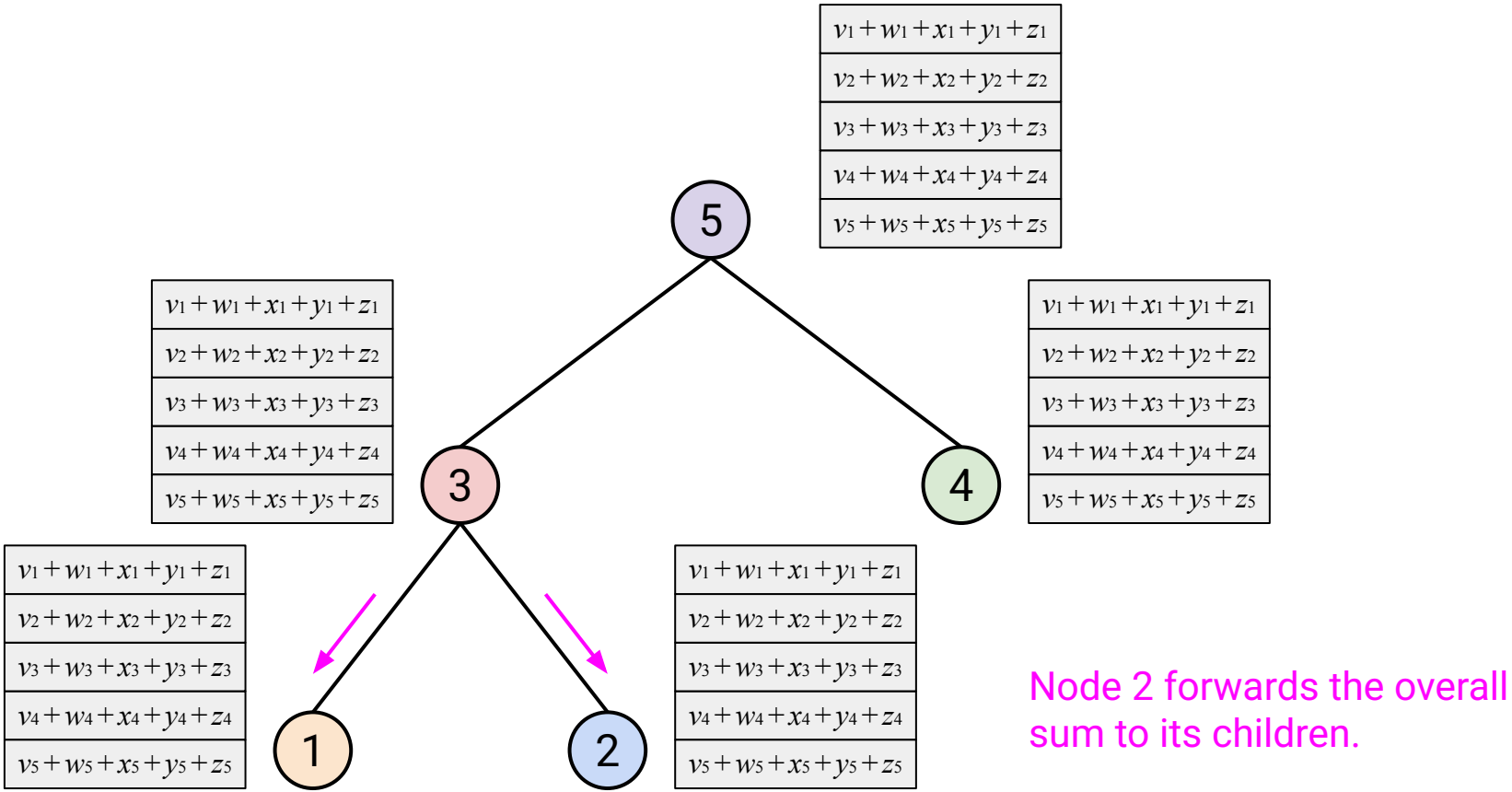
Node 4 computes the sum of all its descendants (plus itself).

2. The overall sum is sent down the tree, starting from the root.



Node 4 forwards the overall sum to its children.

2. The overall sum is sent down the tree, starting from the root.



Tree Topology: Efficiency

Total bandwidth: Each node receives up to 2 vectors (D bytes each) from children. Each node sends 1 vector (D bytes) to parent. p nodes in total.

Number of steps: Tree has p nodes, and height $\log(p)$.

Bandwidth per step: Each node receives up to 2 vectors from children, and sends up to 1 vector to parent. Each vector is D bytes.

	Total Bandwidth	Number of Steps	Bandwidth Per Step
Mesh	Dp^2	1	Dp
Single Root	Dp	2	Dp
Tree	Dp	$\log(p)$	D

Ring-Based AllReduce (Naive)

Lecture 23, CS 168, Spring 2026

Distributed AI Training

- Motivation
- Infrastructure

Defining Collectives

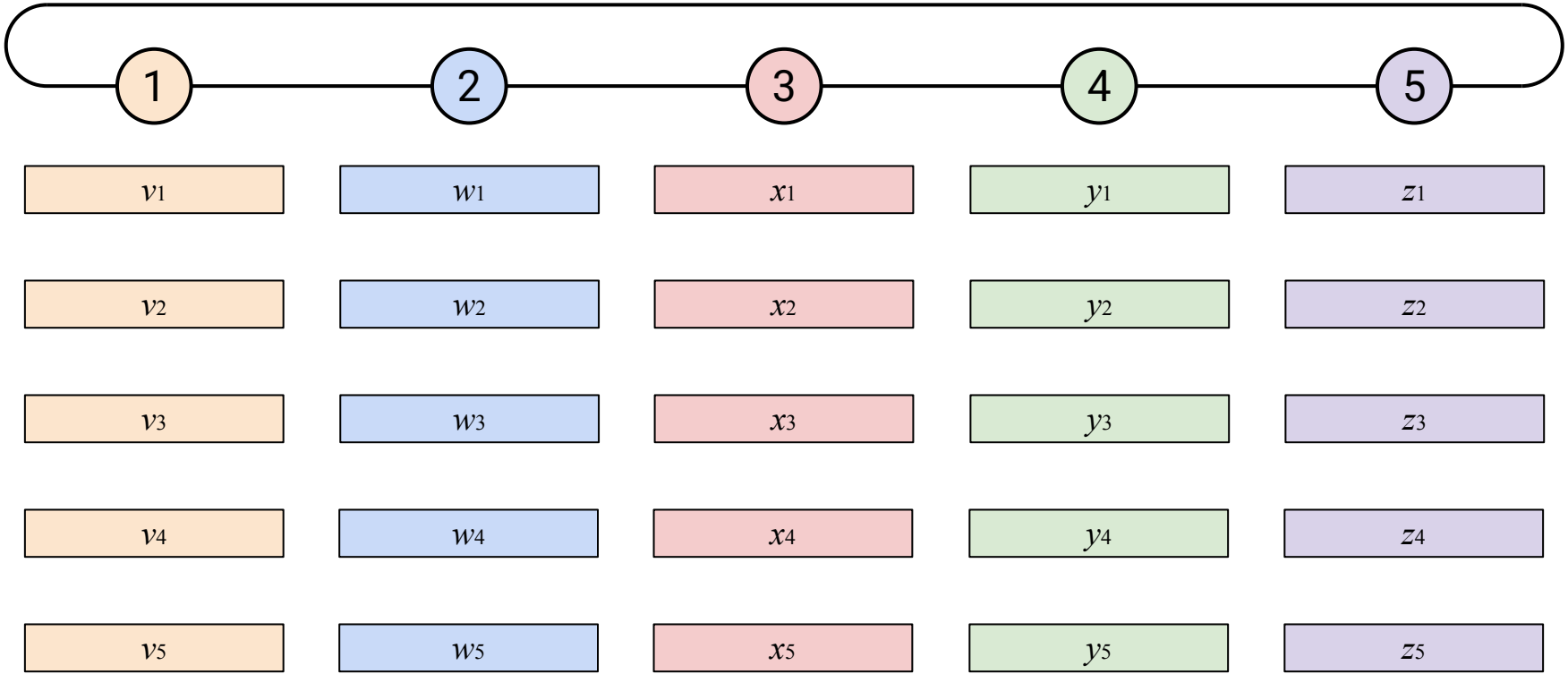
- Definition
- Redistribution Operations
- Consolidation Operations
- Duals and Compositions

Implementing Collectives

- Mesh Topology
- Single Root
- Tree
- **Ring (Naive)**
- Ring (Optimized)
- Overlay/Underlay Topologies

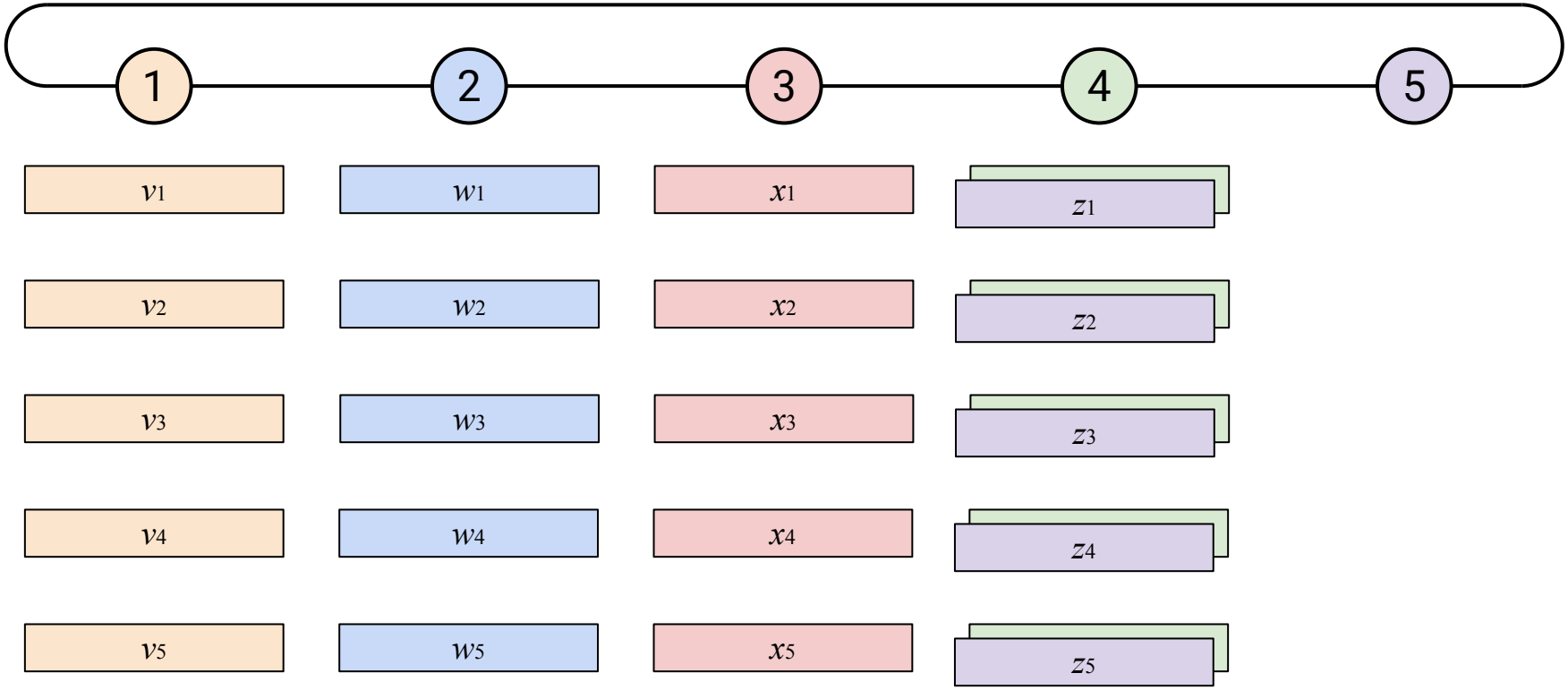
Ring-Based AllReduce (Naive)

1. Send your vector to your left. When you receive a vector, add it to yours.



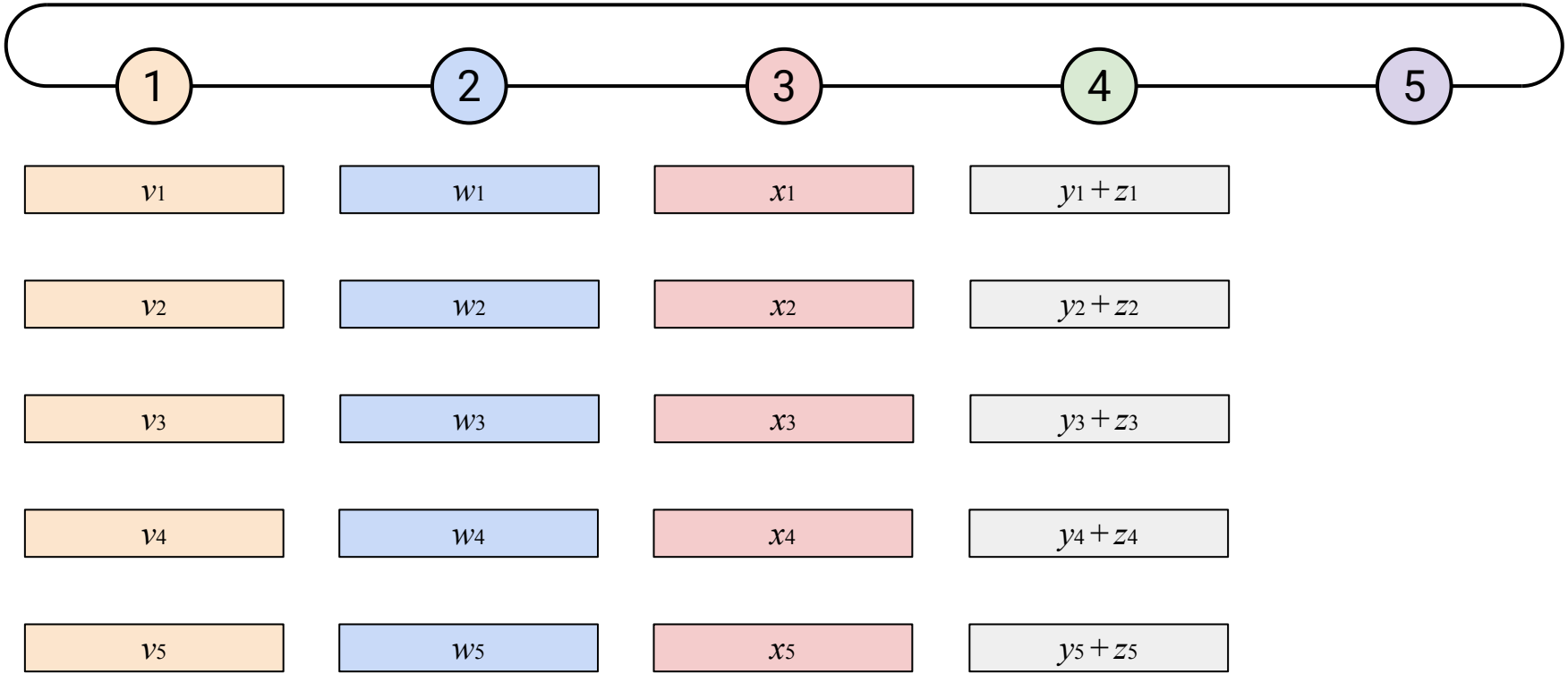
Ring-Based AllReduce (Naive)

1. Send your vector to your left. When you receive a vector, add it to yours.



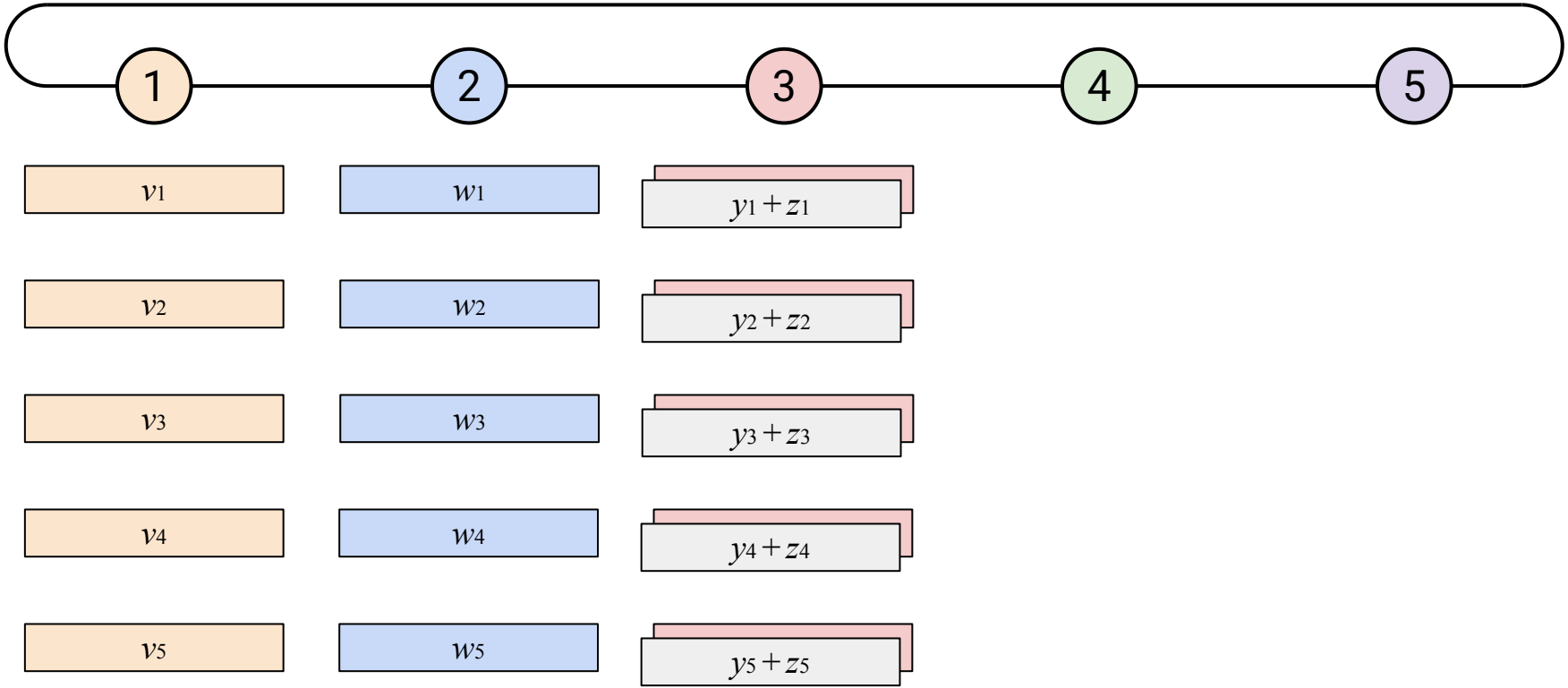
Ring-Based AllReduce (Naive)

1. Send your vector to your left. When you receive a vector, add it to yours.



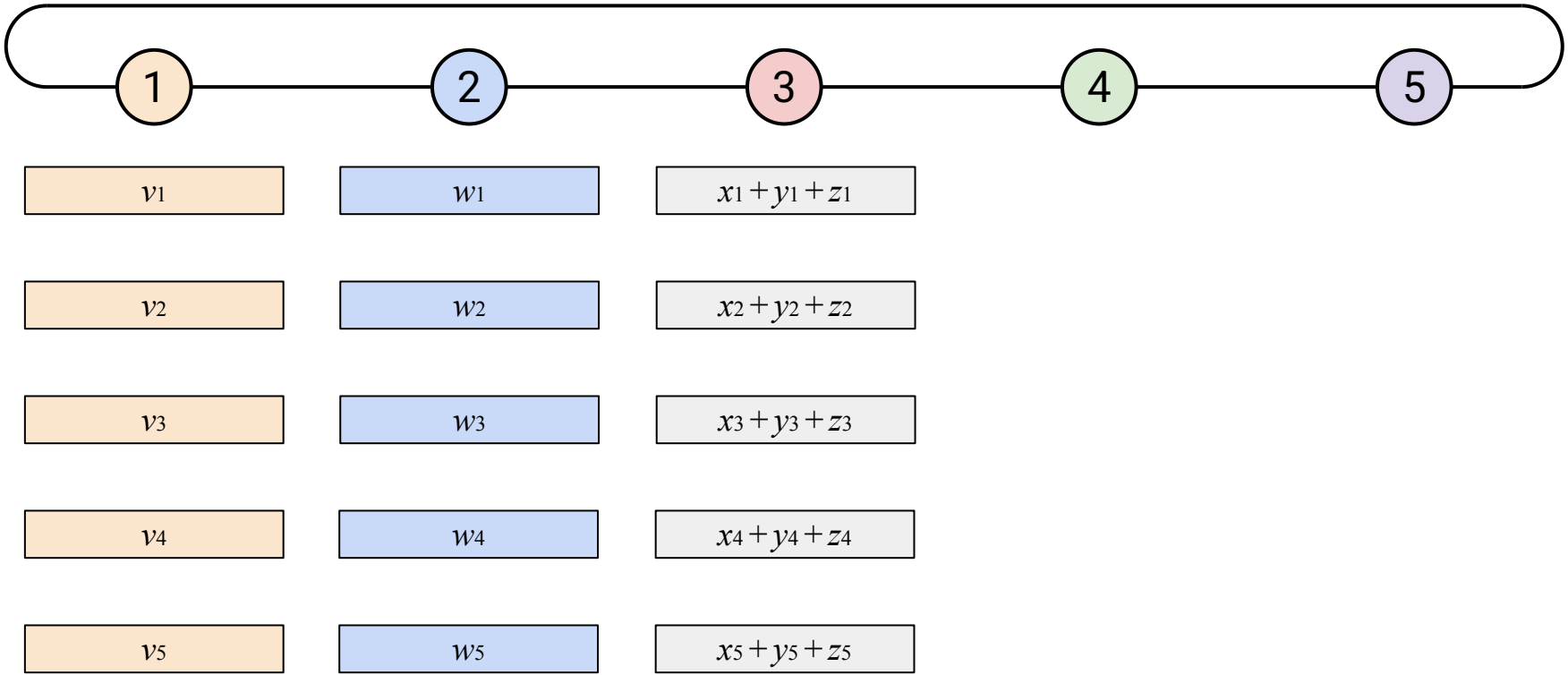
Ring-Based AllReduce (Naive)

1. Send your vector to your left. When you receive a vector, add it to yours.



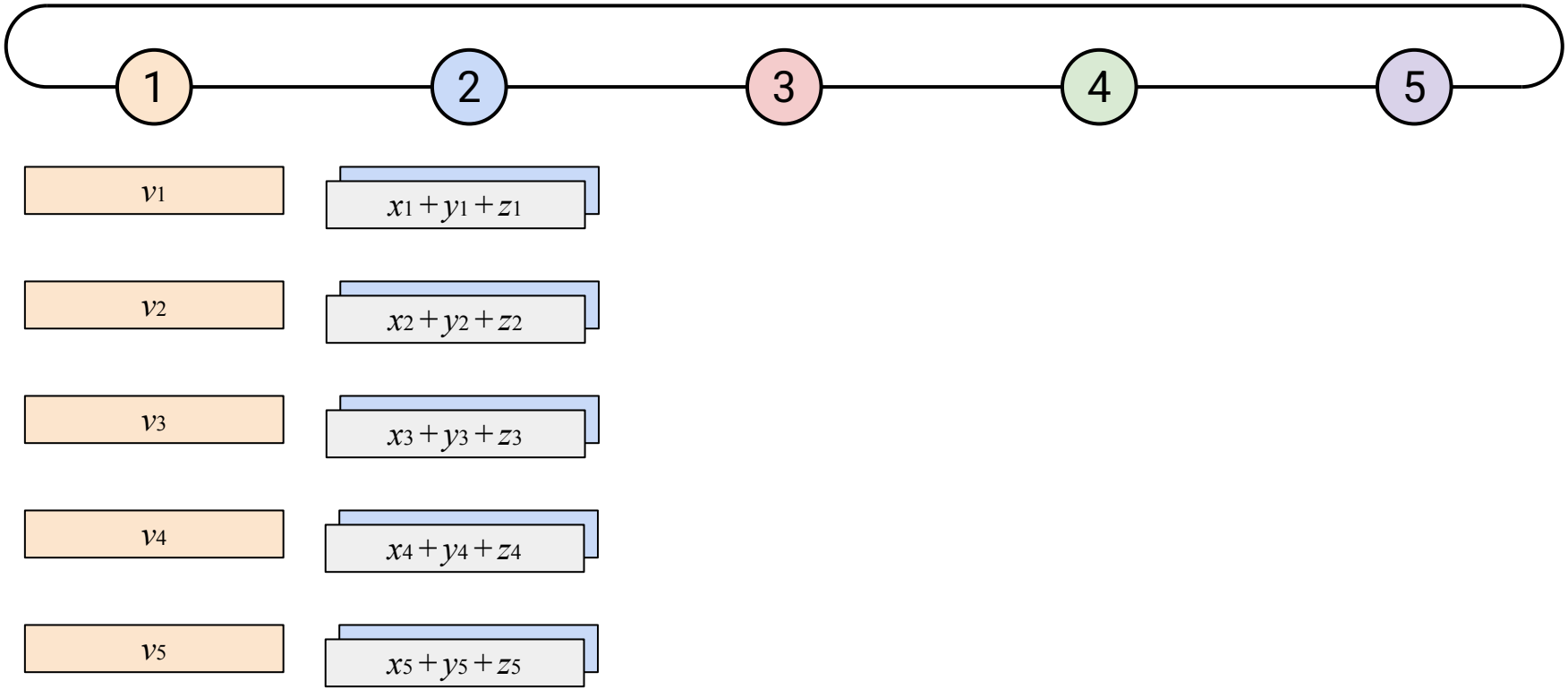
Ring-Based AllReduce (Naive)

1. Send your vector to your left. When you receive a vector, add it to yours.



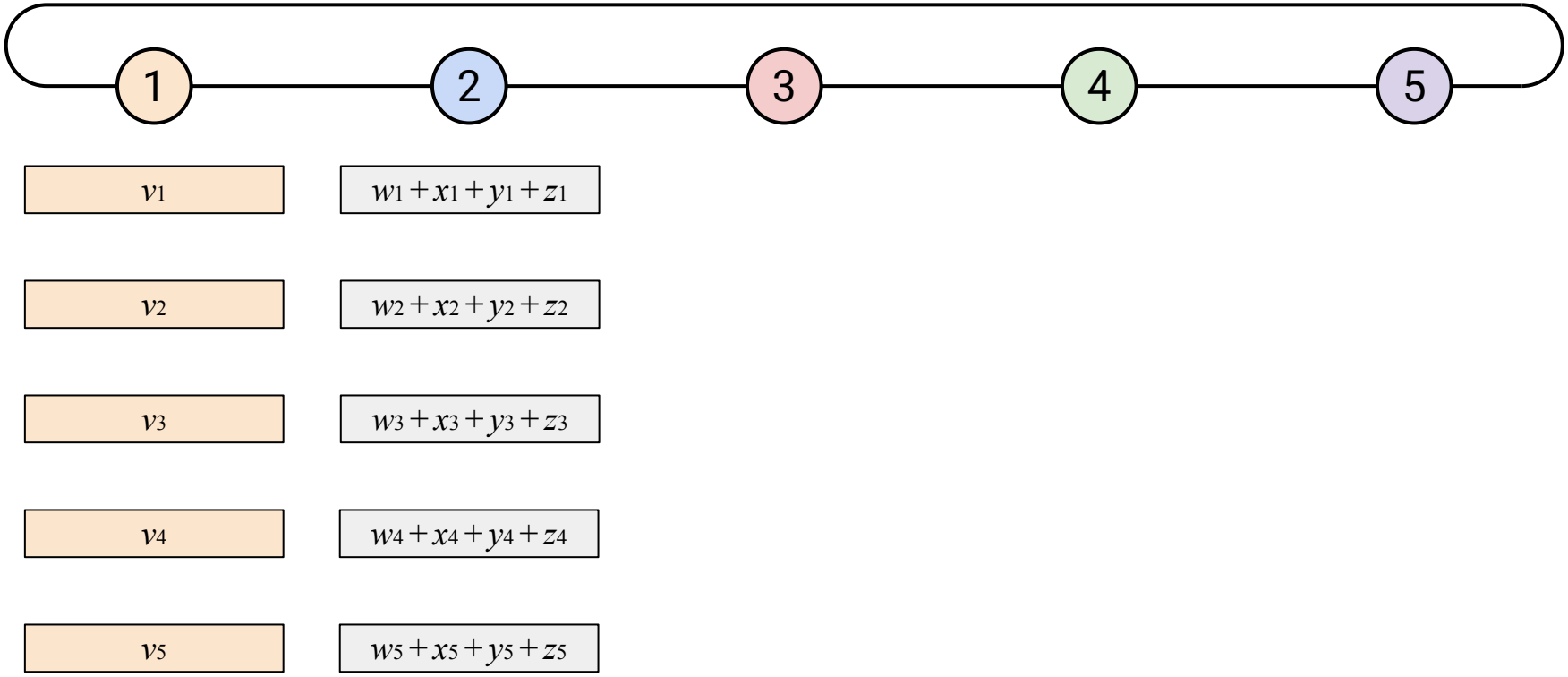
Ring-Based AllReduce (Naive)

1. Send your vector to your left. When you receive a vector, add it to yours.



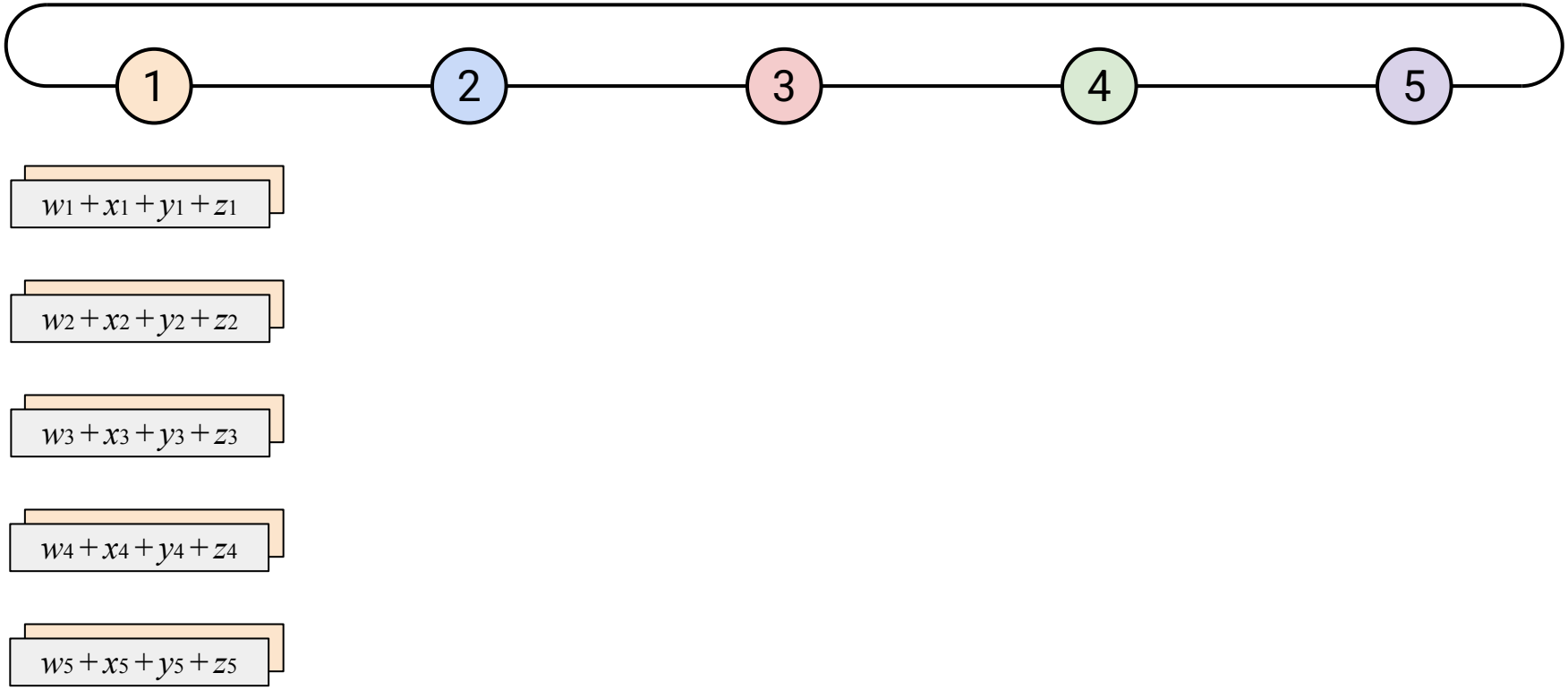
Ring-Based AllReduce (Naive)

1. Send your vector to your left. When you receive a vector, add it to yours.



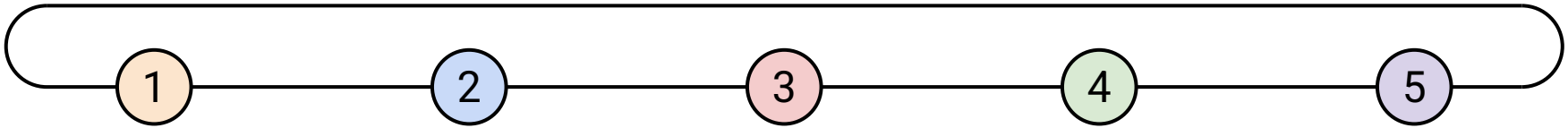
Ring-Based AllReduce (Naive)

1. Send your vector to your left. When you receive a vector, add it to yours.



Ring-Based AllReduce (Naive)

1. Send your vector to your left. When you receive a vector, add it to yours.



$v_1 + w_1 + x_1 + y_1 + z_1$

$v_2 + w_2 + x_2 + y_2 + z_2$

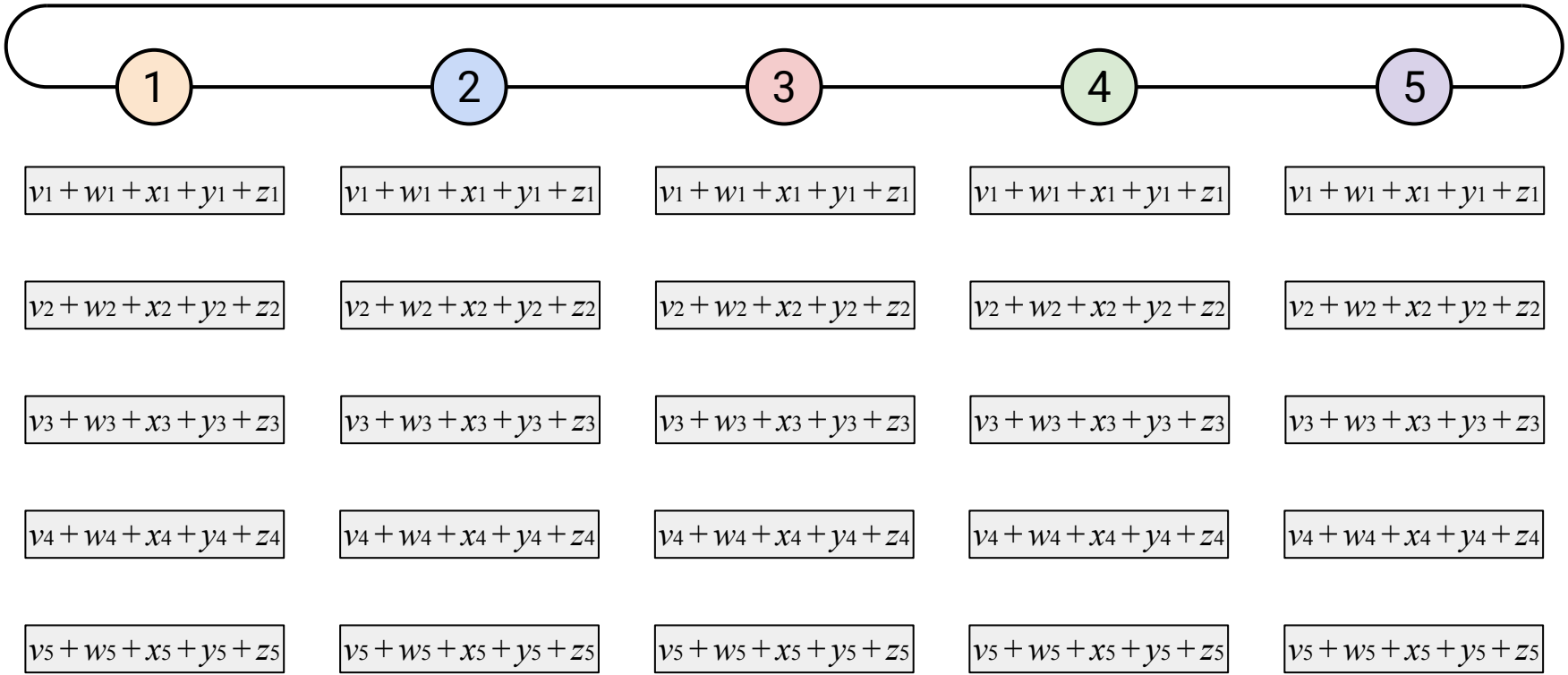
$v_3 + w_3 + x_3 + y_3 + z_3$

$v_4 + w_4 + x_4 + y_4 + z_4$

$v_5 + w_5 + x_5 + y_5 + z_5$

Ring-Based AllReduce (Naive)

2. Send the overall sum to your left, so that everyone gets a copy of it.



Naive Ring Topology: Efficiency

Total bandwidth: Each node receives 1 vector (D bytes) from the right.
Each node sends 1 vector (D bytes) to the left. p nodes in total.

Number of steps: Vector must travel around all p nodes in the ring.

Bandwidth per step: Each node receives 1 vector from the right.
Each node sends 1 vector to the left. Each vector is D bytes.

	Total Bandwidth	Number of Steps	Bandwidth Per Step
Mesh	Dp^2	1	Dp
Single Root	Dp	2	Dp
Tree	Dp	$\log(p)$	D
Ring (Naive)	Dp	p	D

Ring-Based AllReduce (Optimized)

Lecture 23, CS 168, Spring 2026

Distributed AI Training

- Motivation
- Infrastructure

Defining Collectives

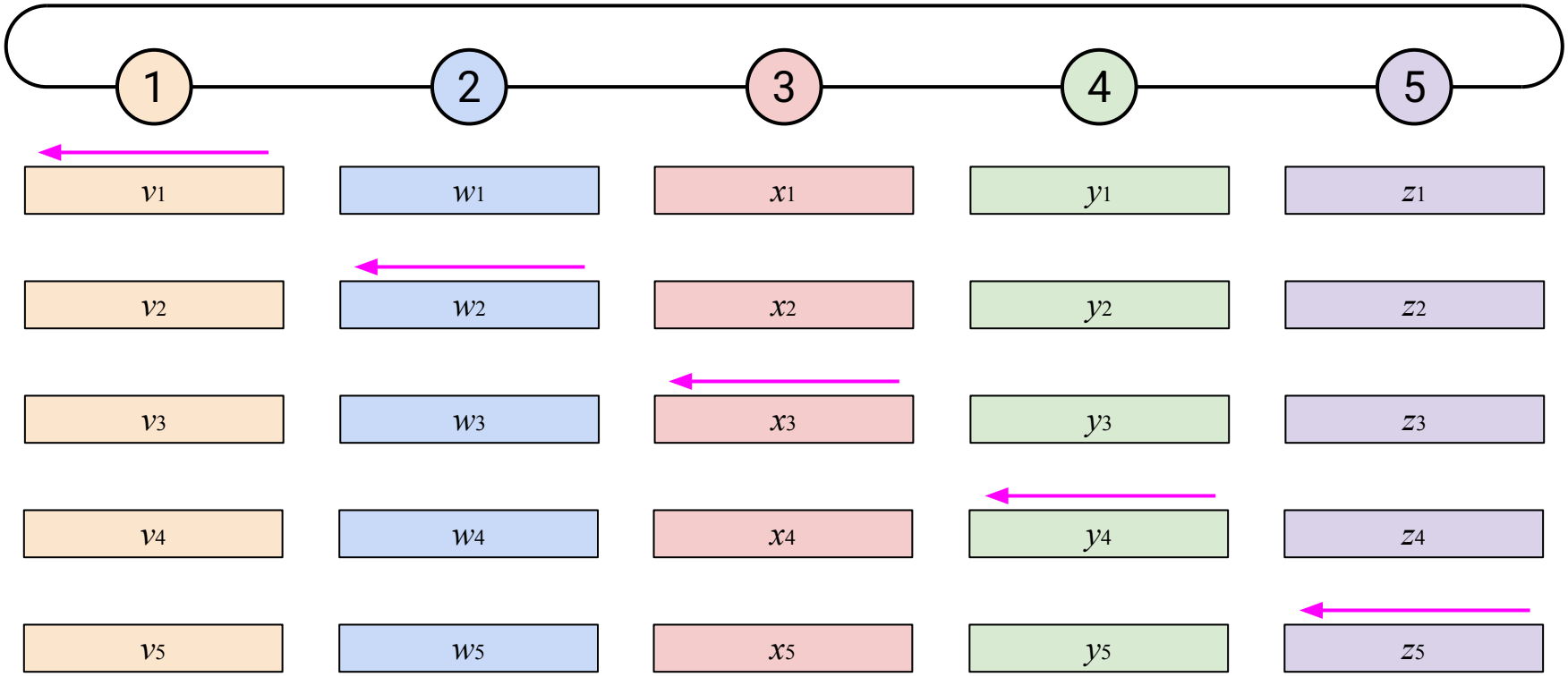
- Definition
- Redistribution Operations
- Consolidation Operations
- Duals and Compositions

Implementing Collectives

- Mesh Topology
- Single Root
- Tree
- Ring (Naive)
- **Ring (Optimized)**
- Overlay/Underlay Topologies

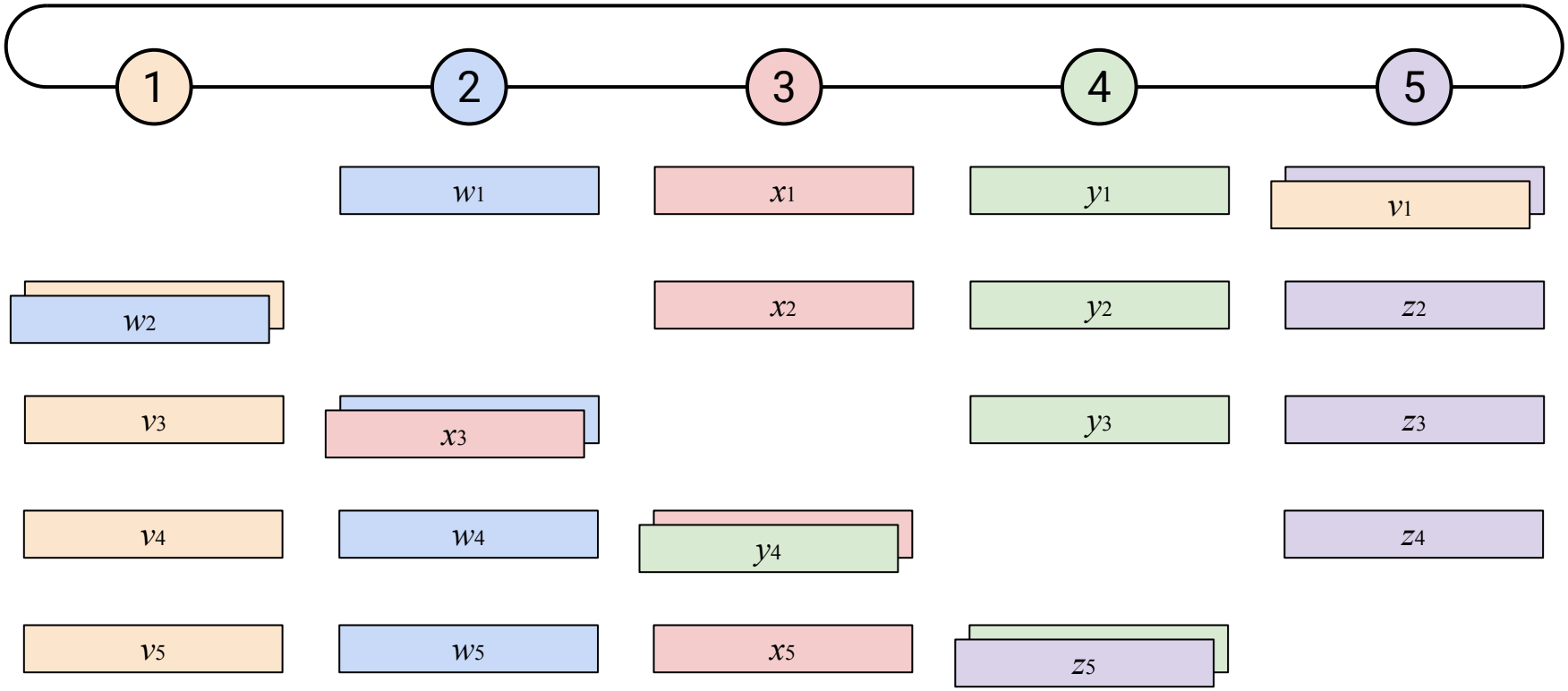
Ring-Based AllReduce (Optimized)

1. Send your vector to your left, one element at a time (i.e. delaying later elements).



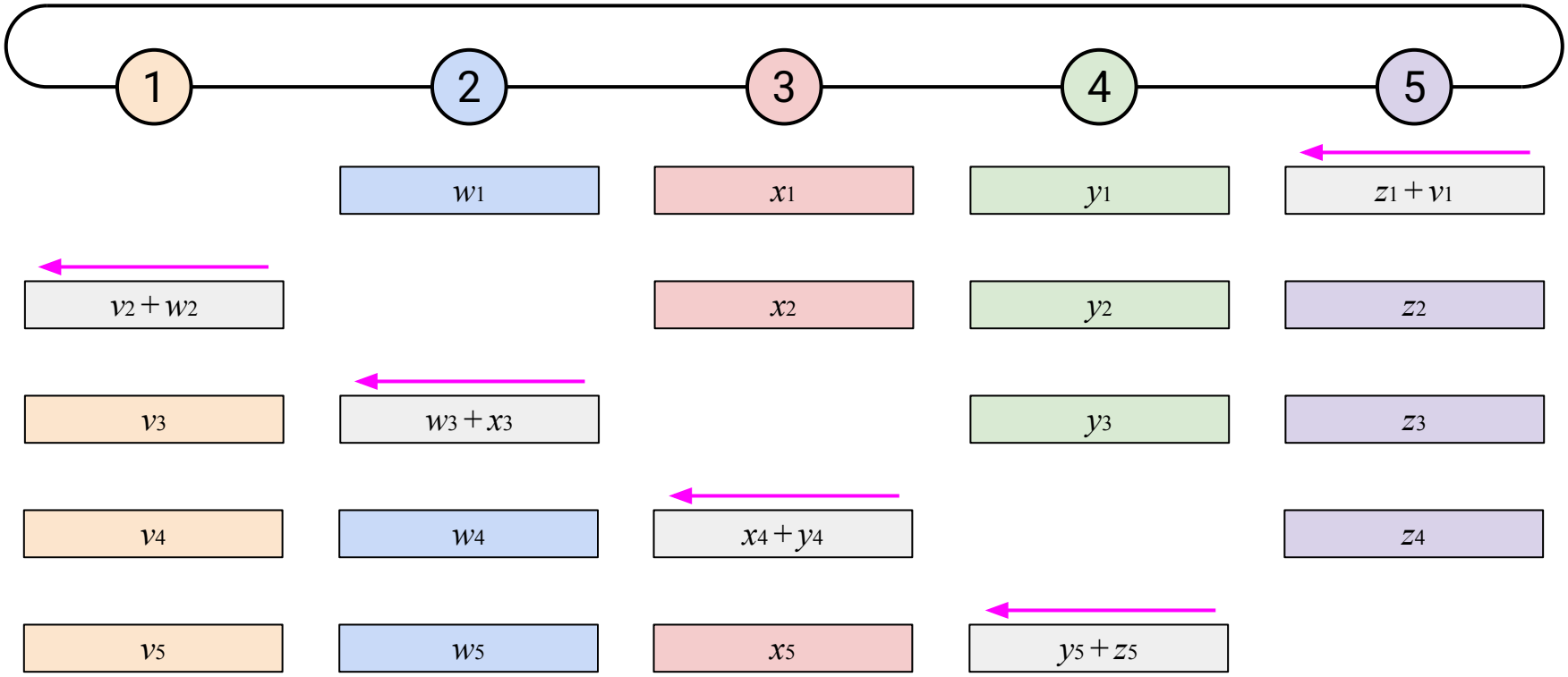
Ring-Based AllReduce (Optimized)

1. Send your vector to your left, one element at a time (i.e. delaying later elements).



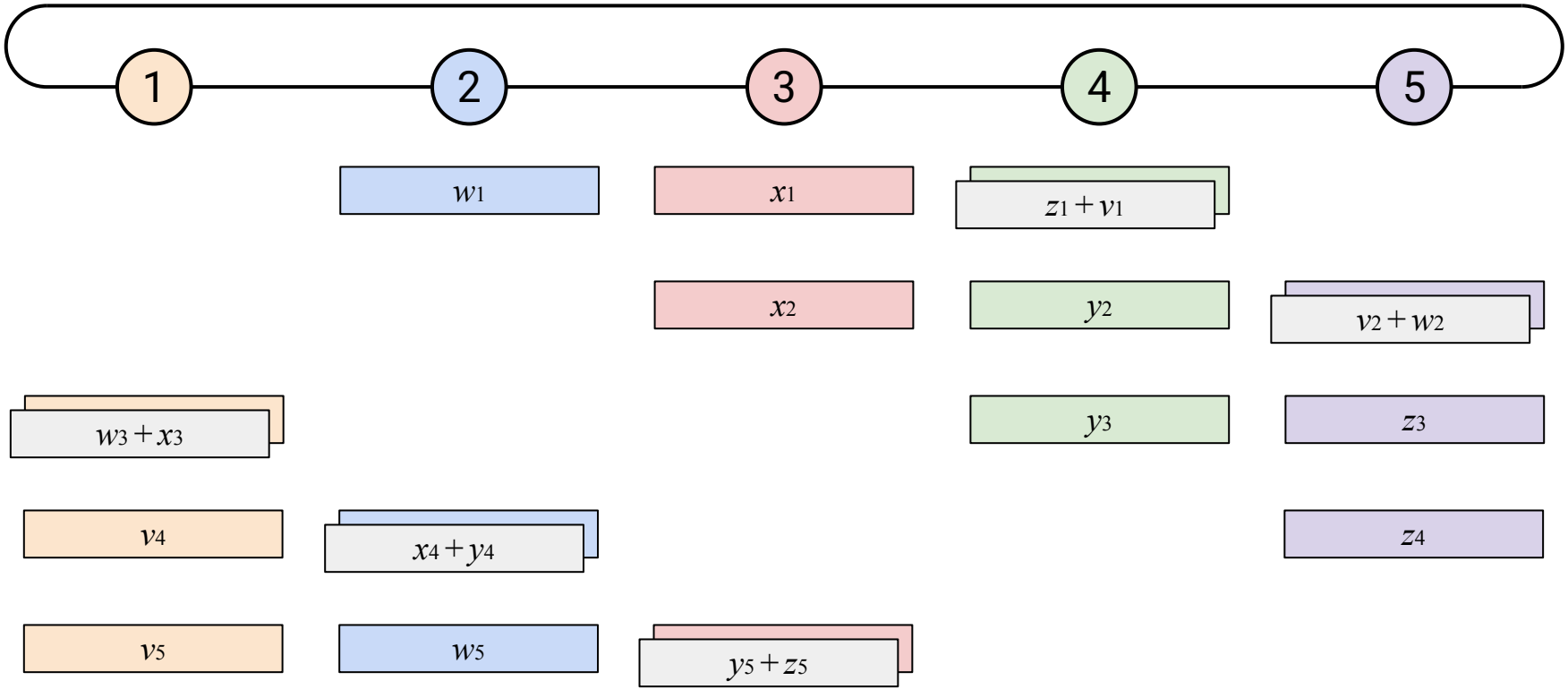
Ring-Based AllReduce (Optimized)

1. Send your vector to your left, one element at a time (i.e. delaying later elements).



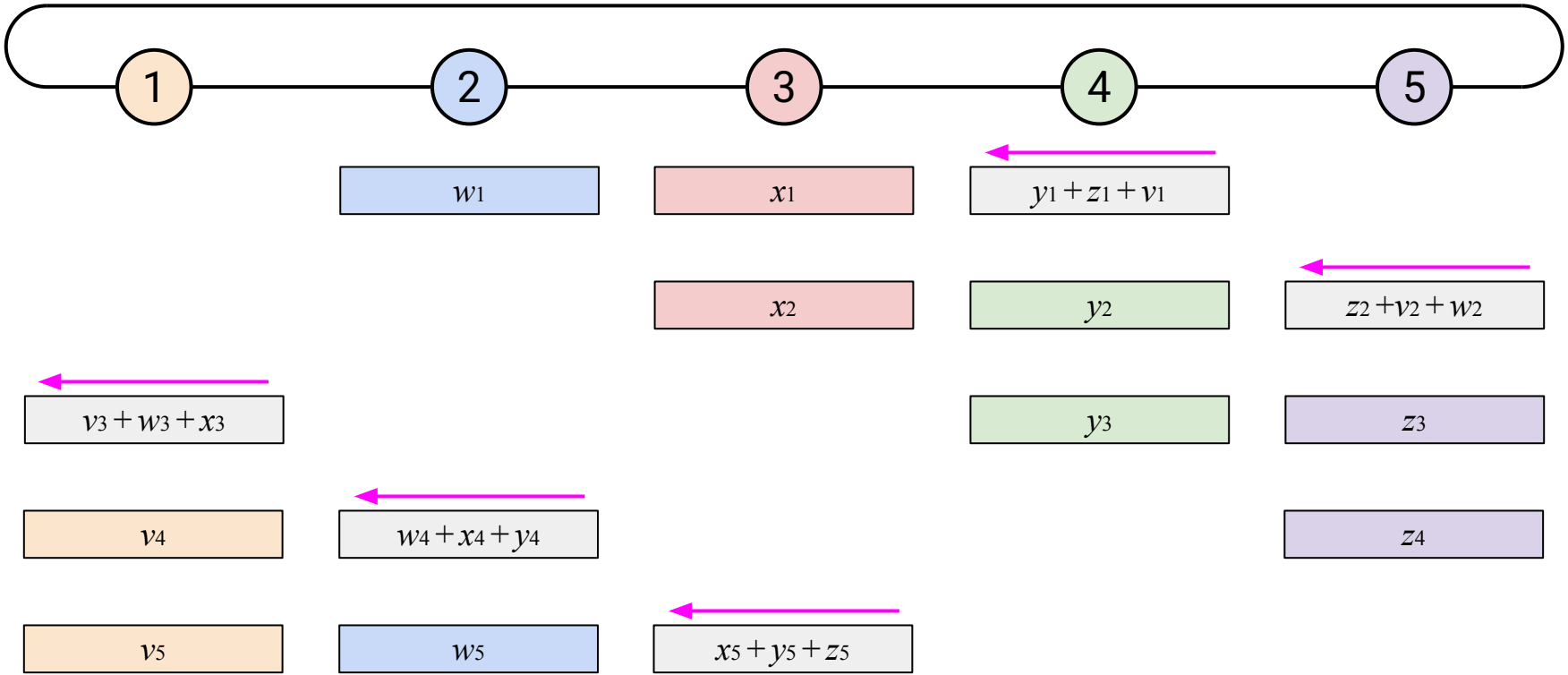
Ring-Based AllReduce (Optimized)

1. Send your vector to your left, one element at a time (i.e. delaying later elements).



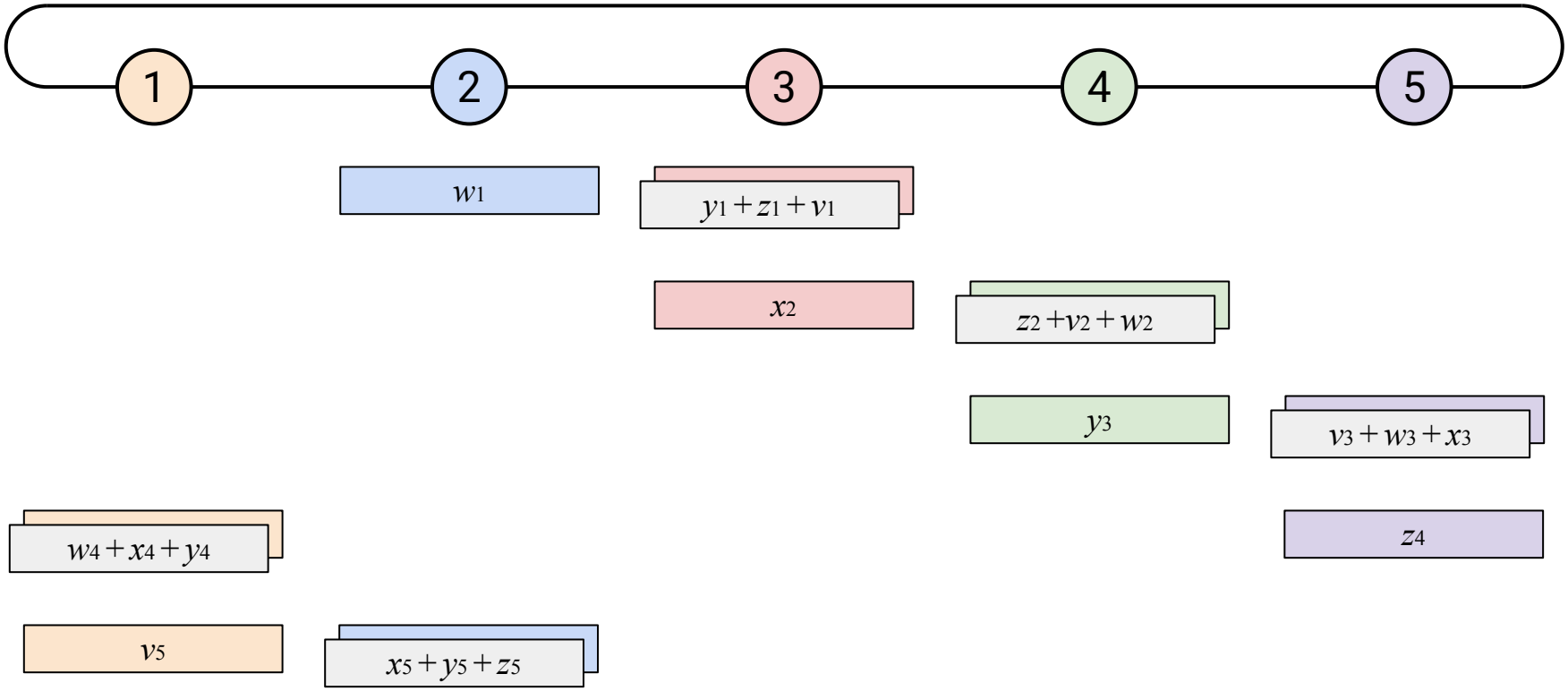
Ring-Based AllReduce (Optimized)

1. Send your vector to your left, one element at a time (i.e. delaying later elements).



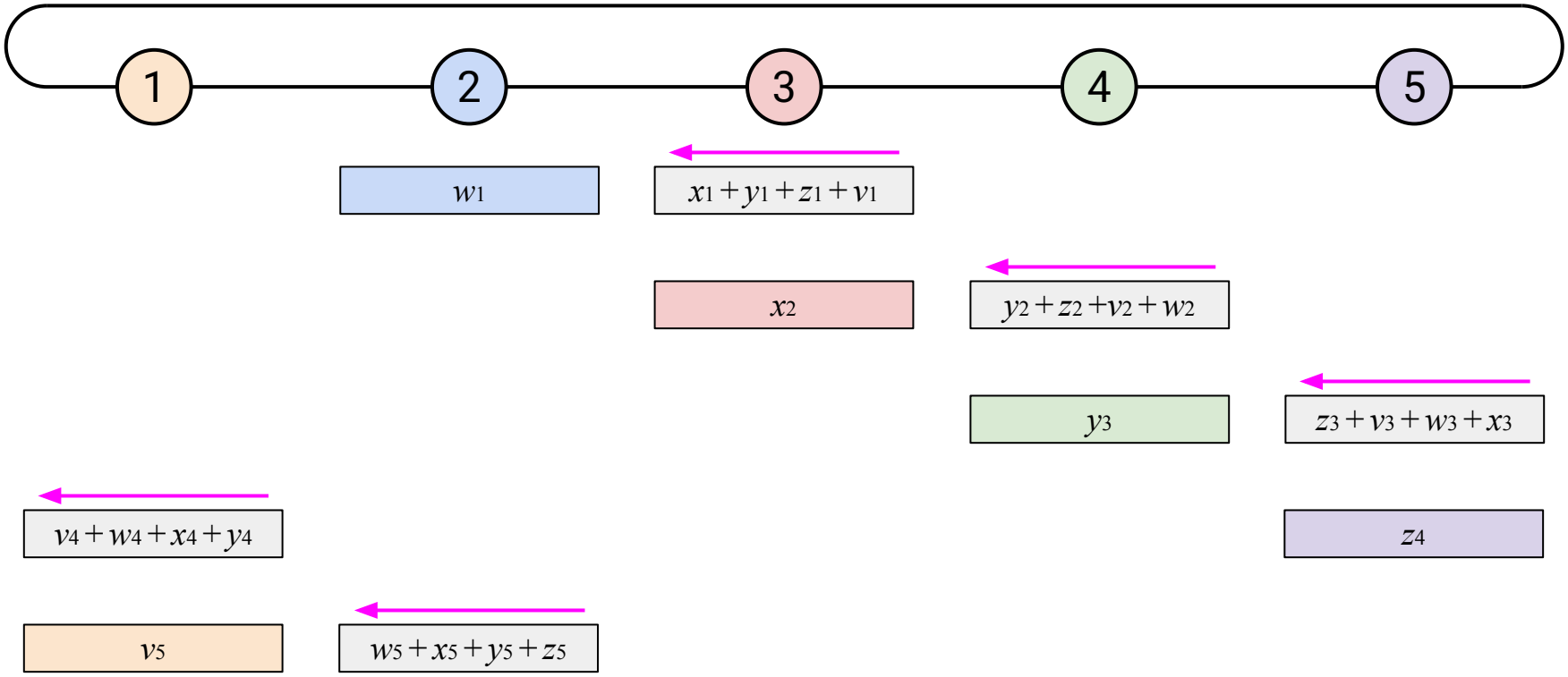
Ring-Based AllReduce (Optimized)

1. Send your vector to your left, one element at a time (i.e. delaying later elements).



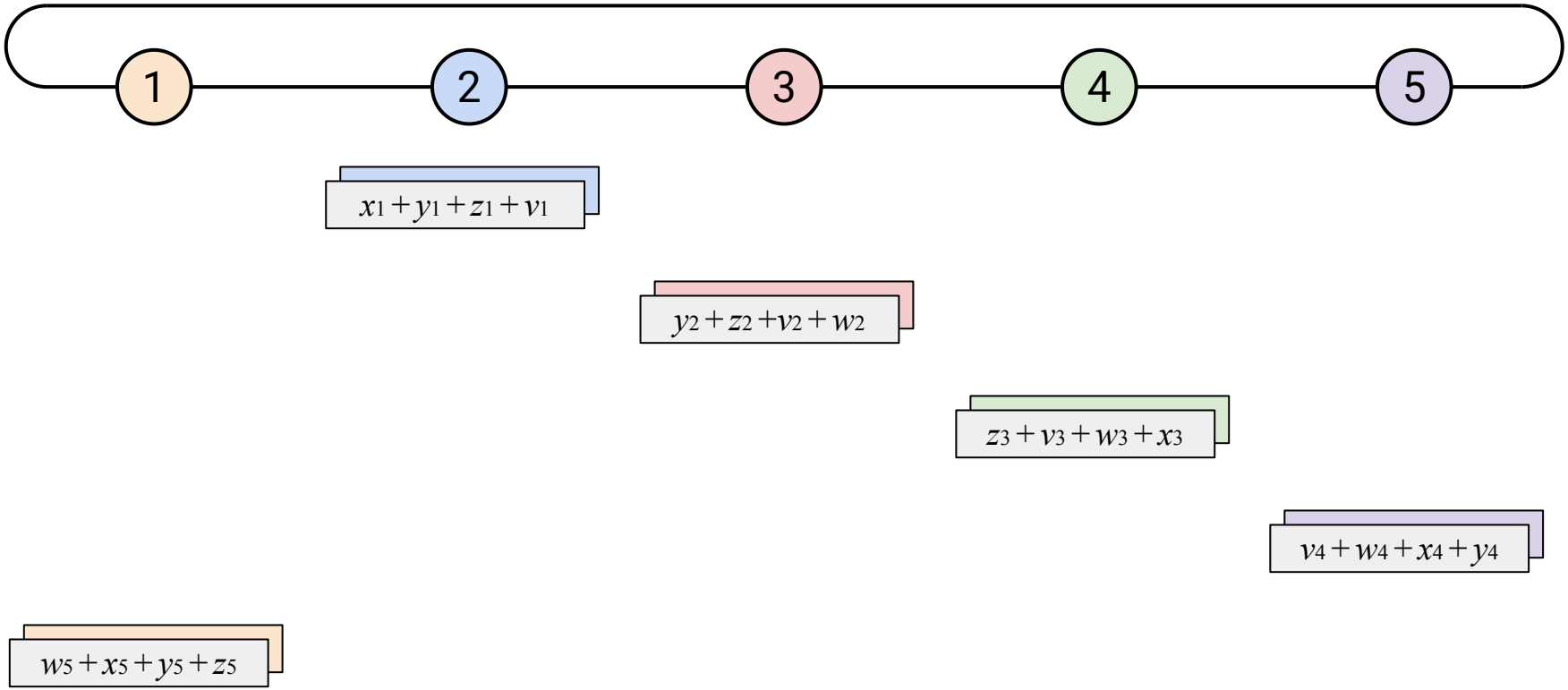
Ring-Based AllReduce (Optimized)

1. Send your vector to your left, one element at a time (i.e. delaying later elements).



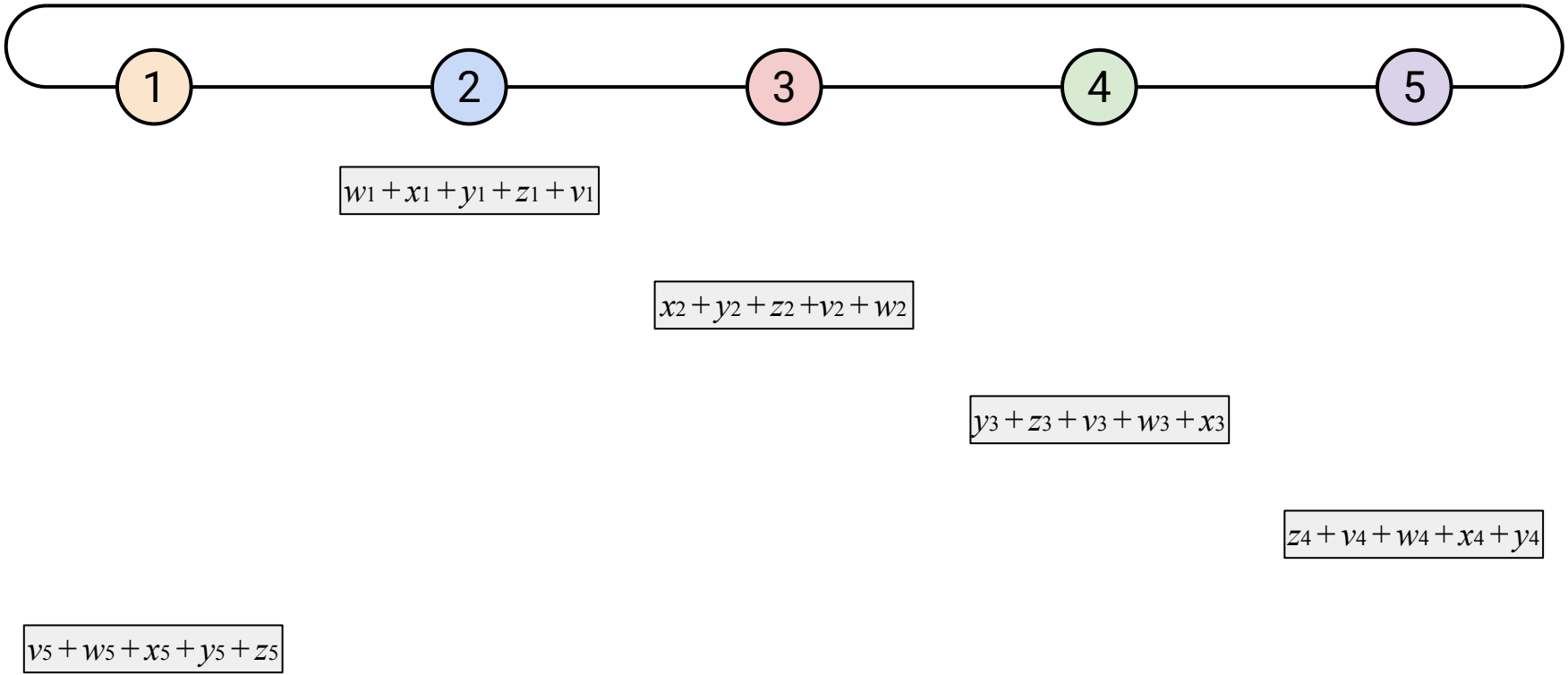
Ring-Based AllReduce (Optimized)

1. Send your vector to your left, one element at a time (i.e. delaying later elements).



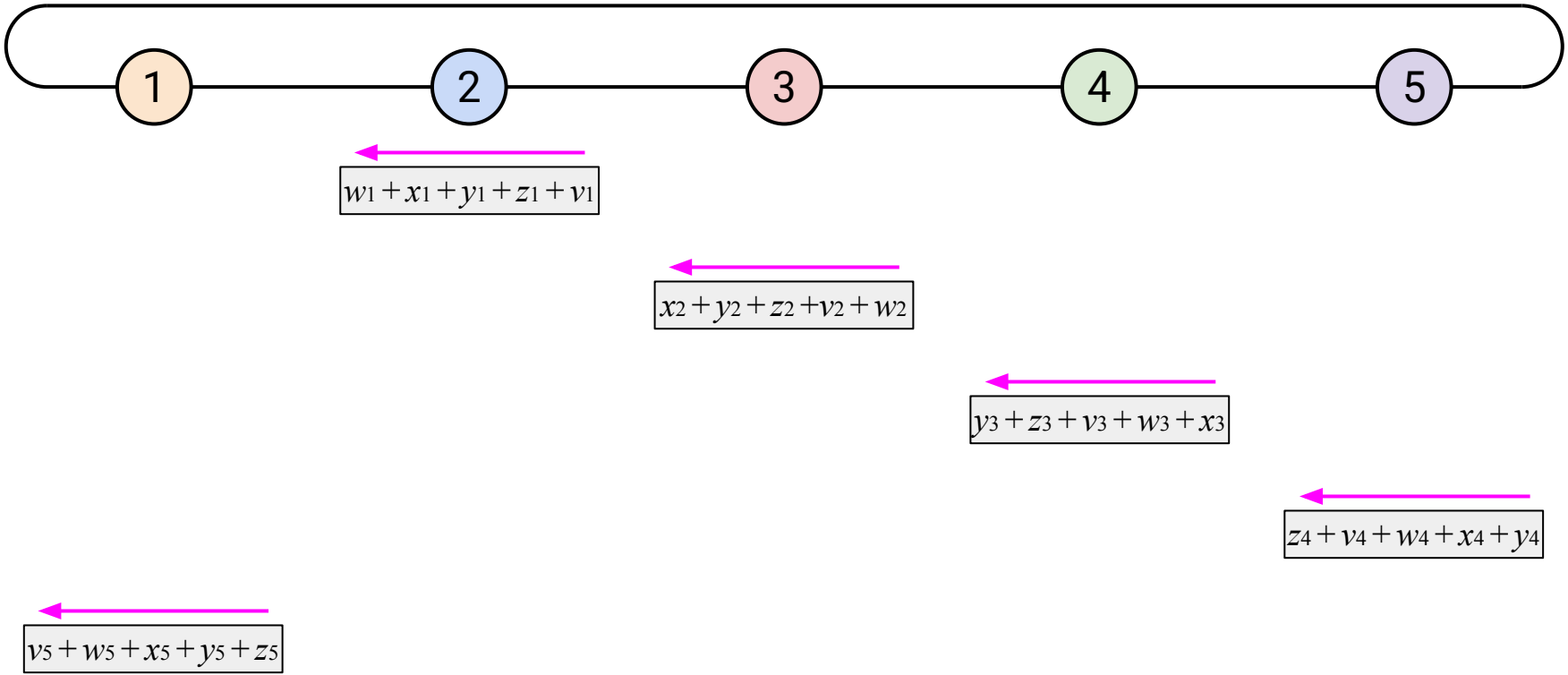
Ring-Based AllReduce (Optimized)

1. Send your vector to your left, one element at a time (i.e. delaying later elements).



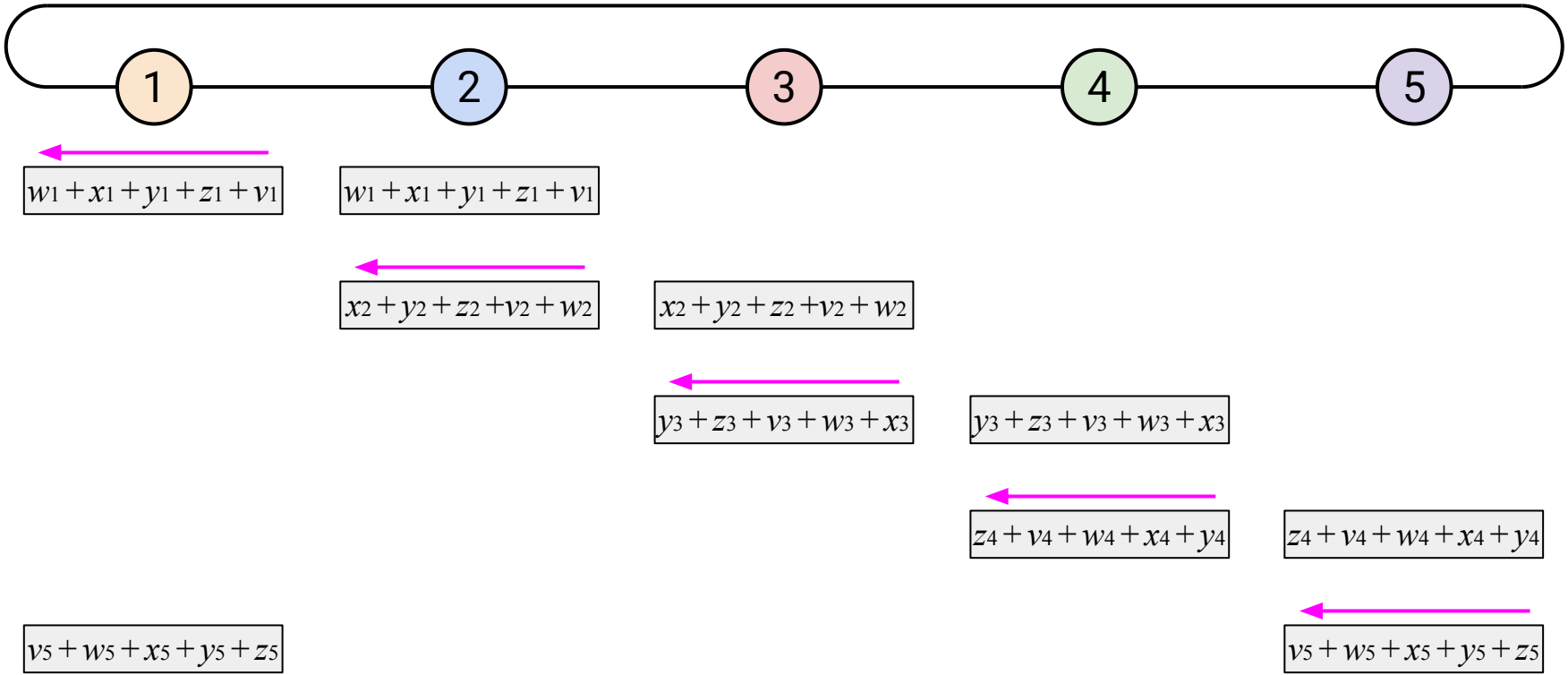
Ring-Based AllReduce (Optimized)

2. Send the overall sum to the left.



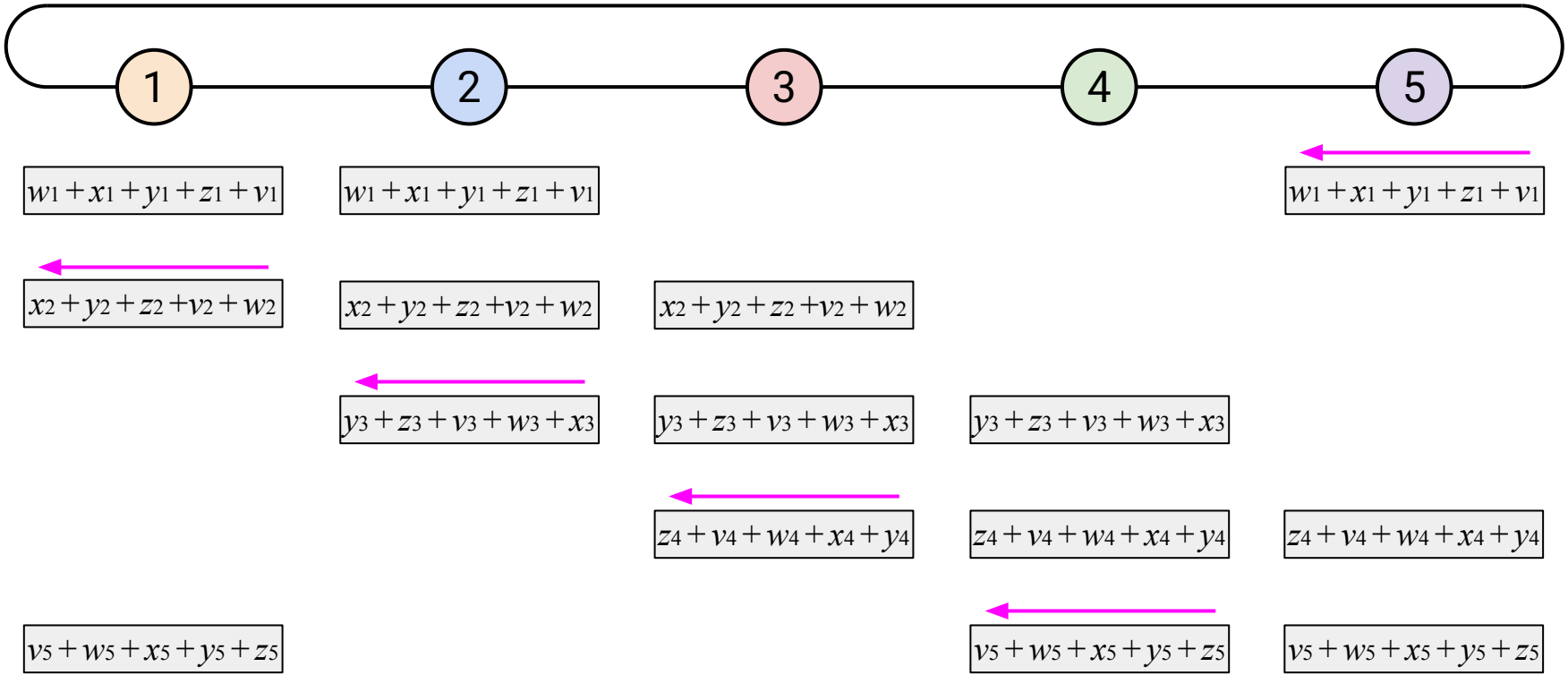
Ring-Based AllReduce (Optimized)

2. Send the overall sum to the left.



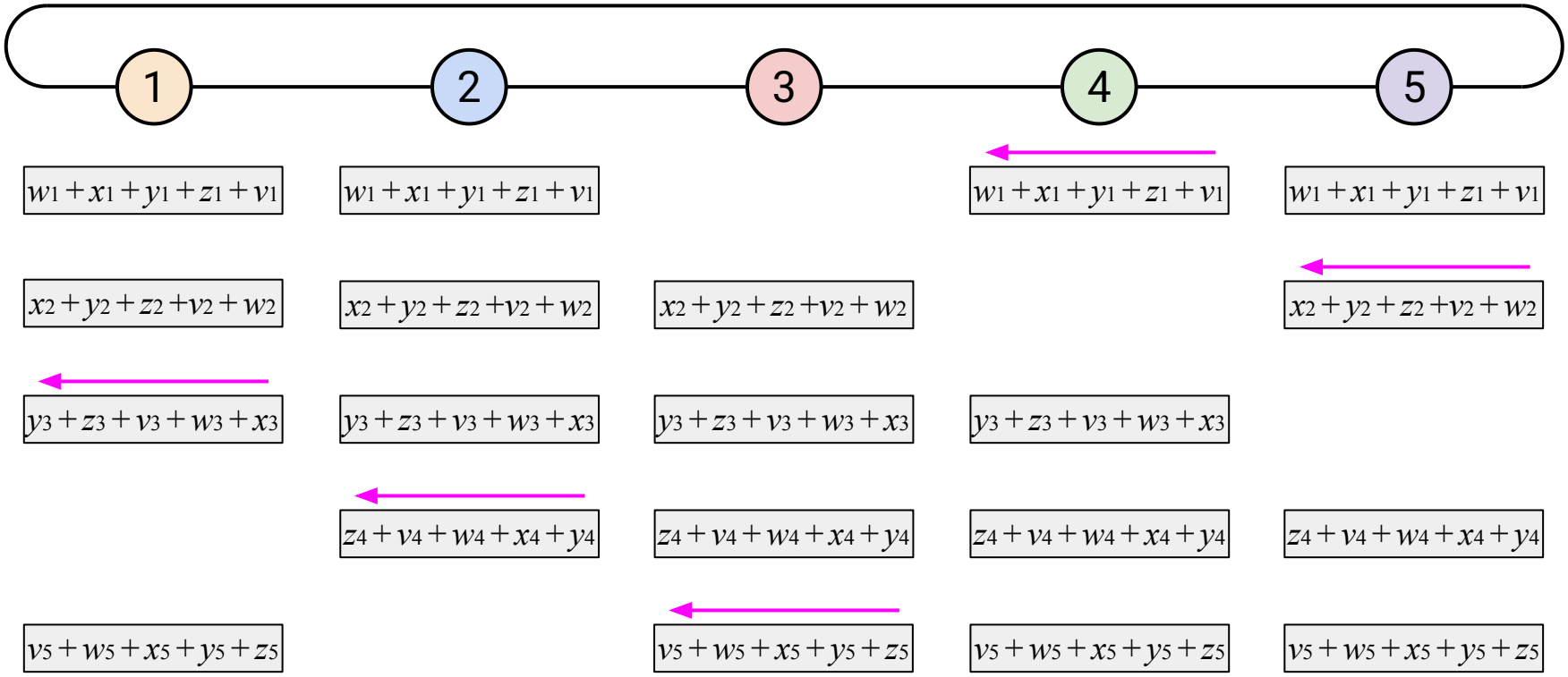
Ring-Based AllReduce (Optimized)

2. Send the overall sum to the left.



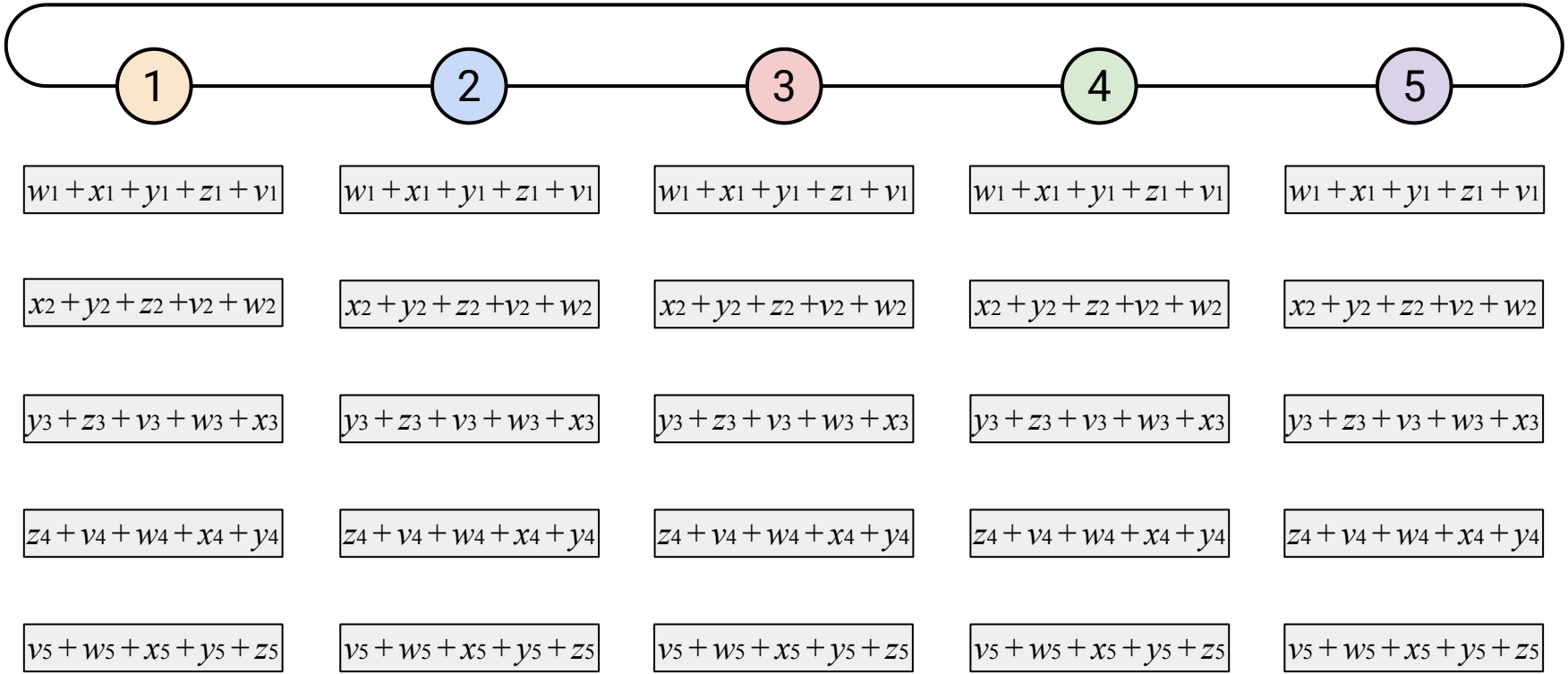
Ring-Based AllReduce (Optimized)

2. Send the overall sum to the left.



Ring-Based AllReduce (Optimized)

2. Send the overall sum to the left.



Naive Ring Topology: Efficiency

Total bandwidth: Each node receives 1 vector (D bytes) from the right.
Each node sends 1 vector (D bytes) to the left. p nodes in total.

Number of steps: Vector must travel around all p nodes in the ring.

Bandwidth per step: Each node receives 1 element (D/p bytes) from the right.
Each node sends 1 element (D/p bytes) to the left.

	Total Bandwidth	Number of Steps	Bandwidth Per Step
Mesh	Dp^2	1	Dp
Single Root	Dp	2	Dp
Tree	Dp	$\log(p)$	D
Ring (Naive)	Dp	p	D
Ring (Optimized)	Dp	p	D/p

Overlay/Underlay Topologies

Lecture 23, CS 168, Spring 2026

Distributed AI Training

- Motivation
- Infrastructure

Defining Collectives

- Definition
- Redistribution Operations
- Consolidation Operations
- Duals and Compositions

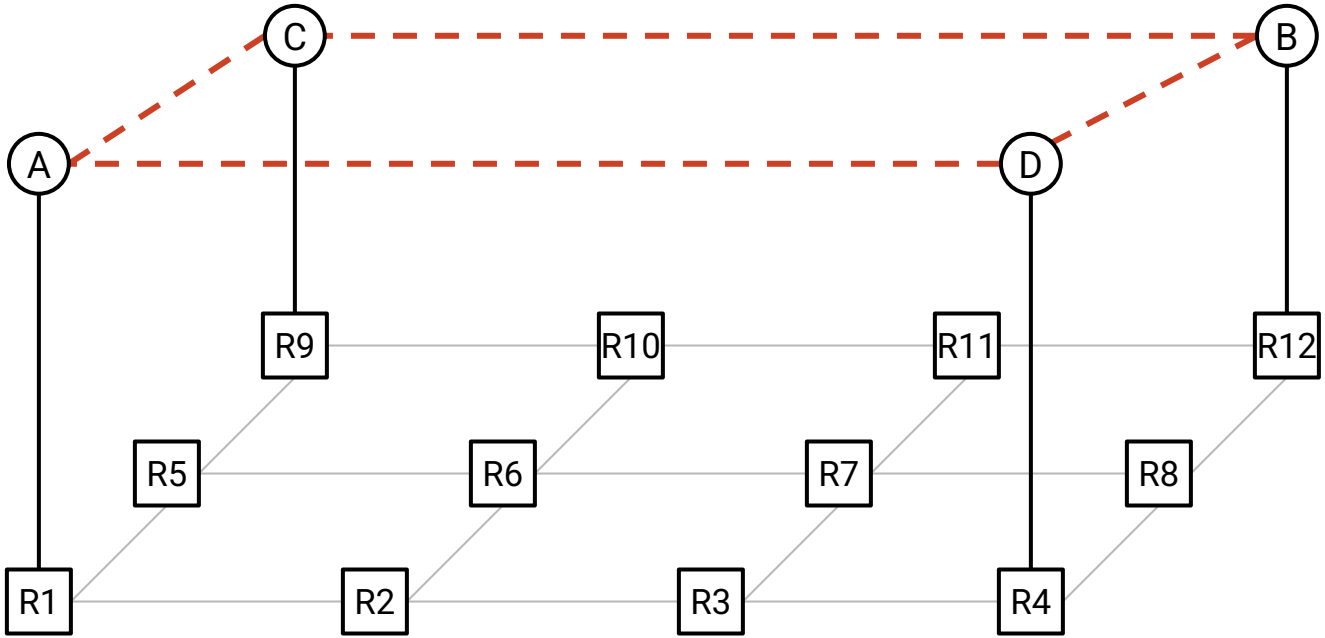
Implementing Collectives

- Mesh Topology
- Single Root
- Tree
- Ring (Naive)
- Ring (Optimized)
- **Overlay/Underlay Topologies**

Overlay and Underlay Topologies

Suppose Hosts A, B, C, D want to run an AllReduce operation.

A, B, C, D are not in a ring topology. But, we can add virtual links to create a ring!



Different Overlay Topologies

Suppose Hosts A, B, C, D want to run an AllReduce operation.

For best performance, in what order should we connect the hosts in a ring?

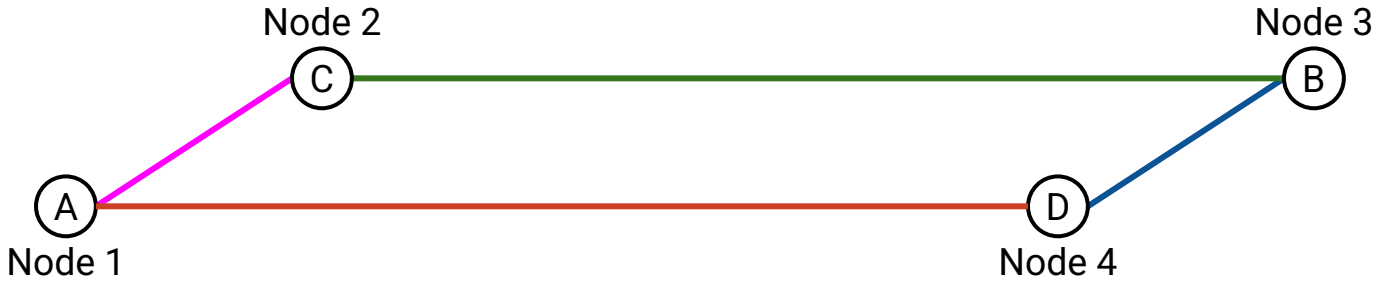
(The way we number the nodes affects the ring we draw.)

Overlay Topology 1:

- Node 1 = A
- Node 2 = C
- Node 3 = D
- Node 4 = B

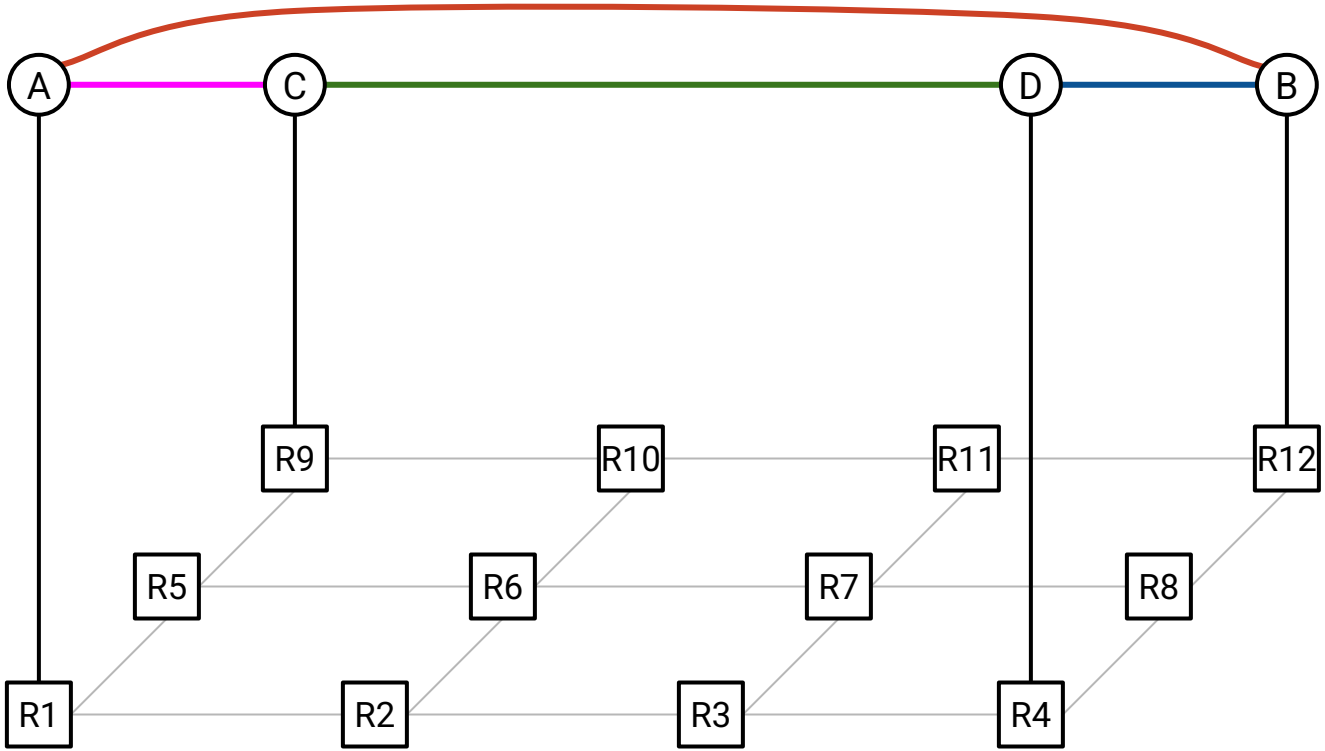
Overlay Topology 2:

- Node 1 = A
- Node 2 = C
- Node 3 = B
- Node 4 = D



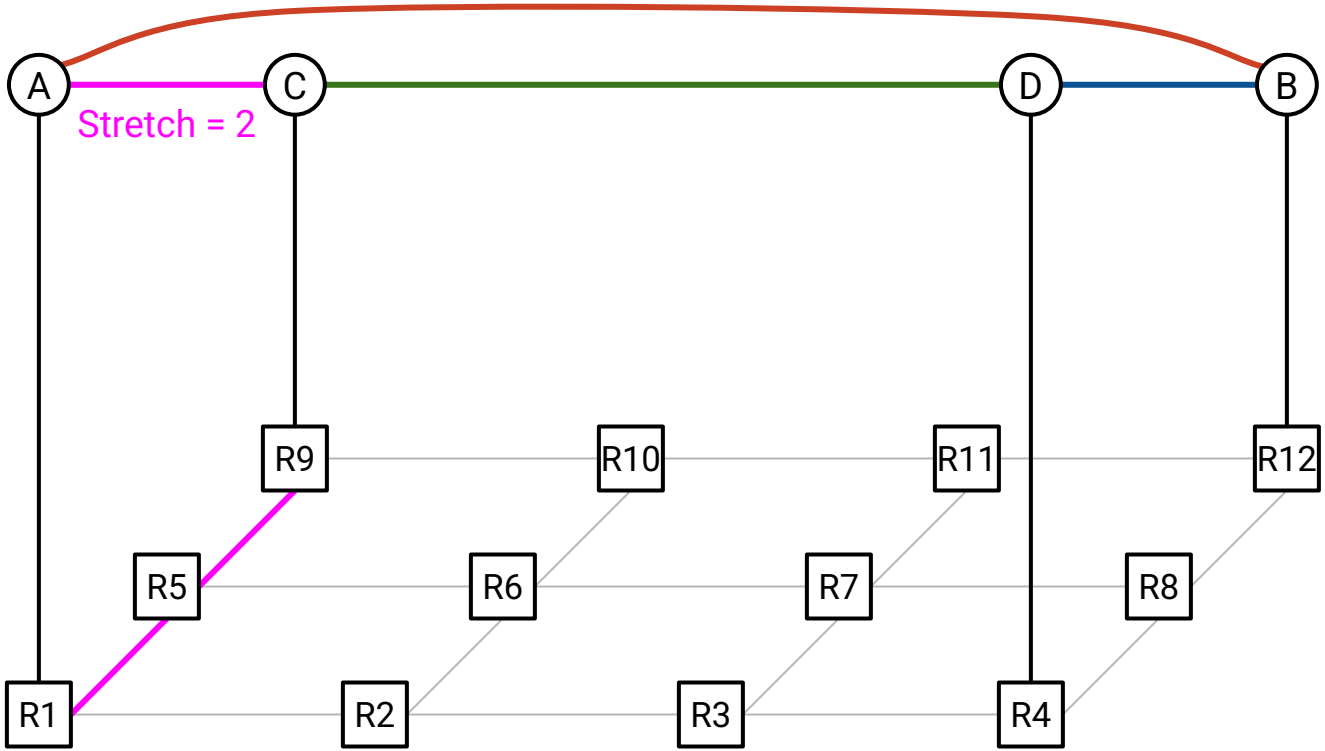
Overlay Ring Topology #1

The average stretch factor depends on how the nodes are numbered.



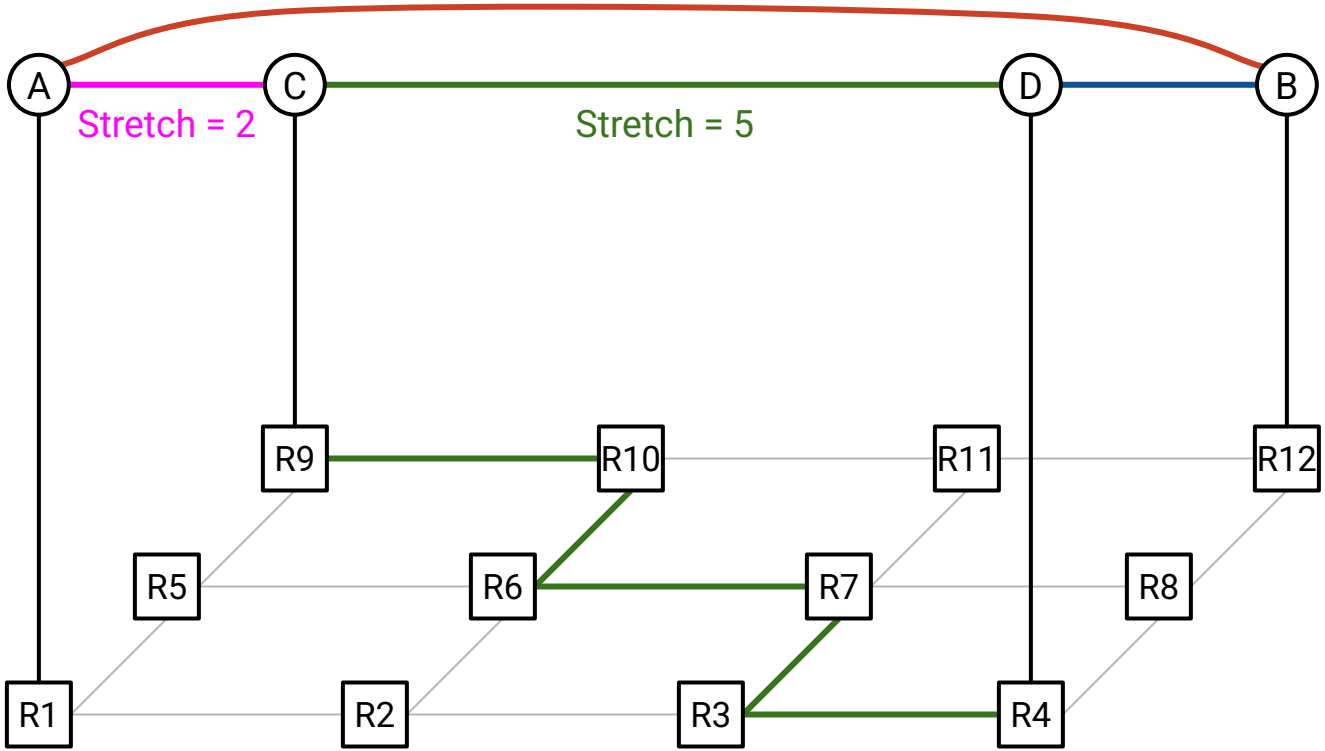
Overlay Ring Topology #1

The average stretch factor depends on how the nodes are numbered.



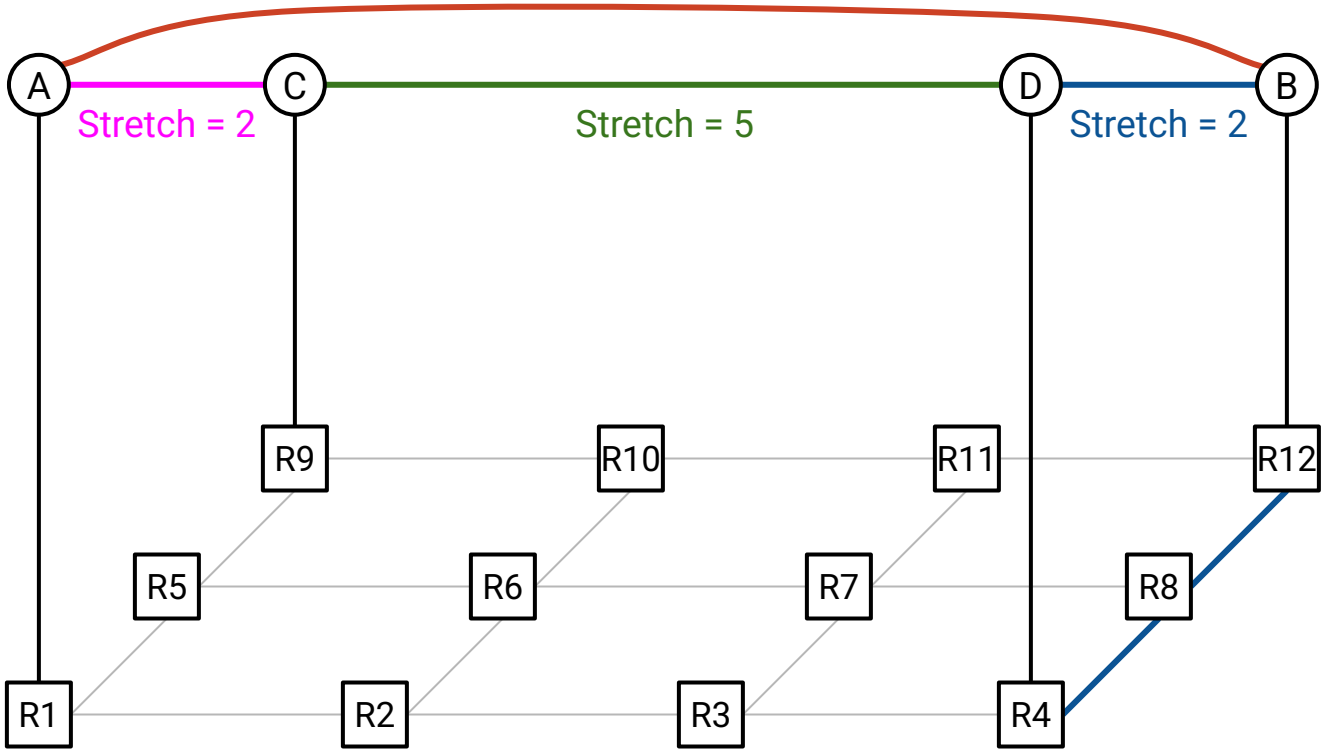
Overlay Ring Topology #1

The average stretch factor depends on how the nodes are numbered.



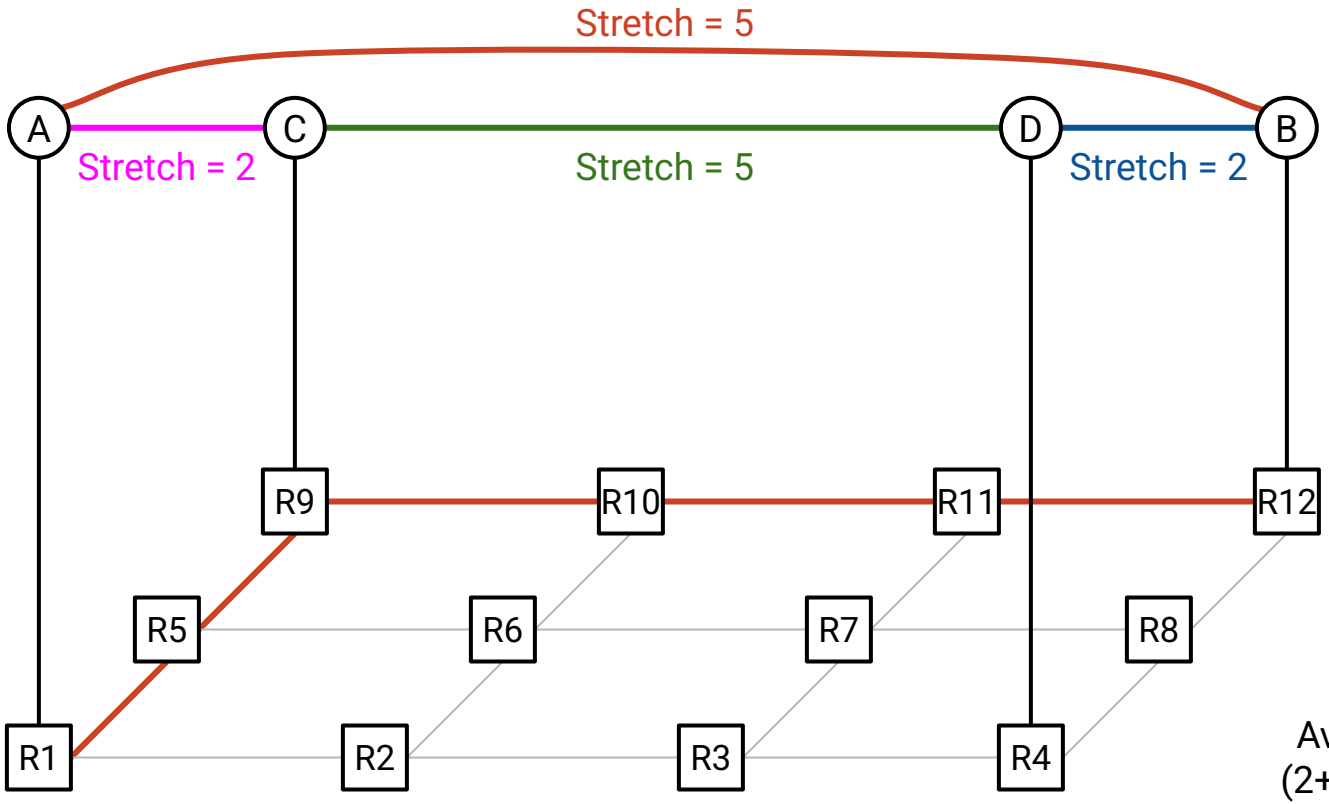
Overlay Ring Topology #1

The average stretch factor depends on how the nodes are numbered.



Overlay Ring Topology #1

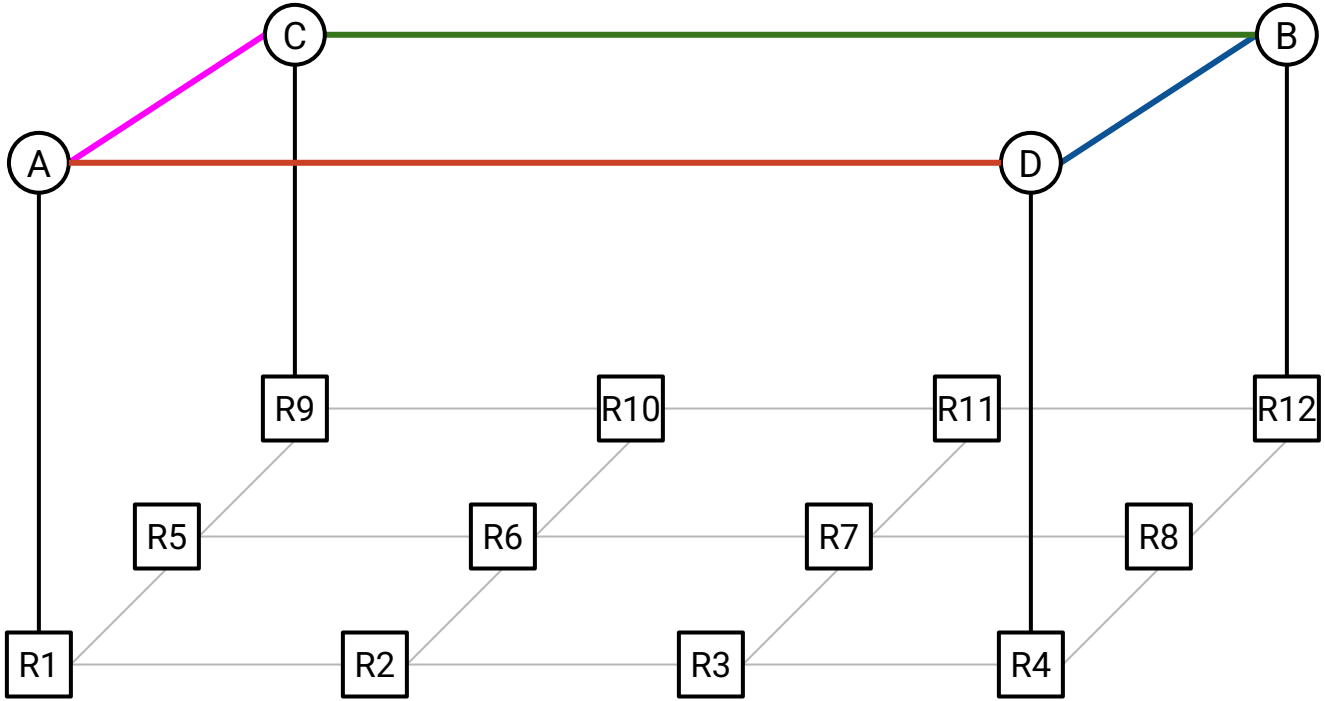
The average stretch factor depends on how the nodes are numbered.



Average stretch:
(2+5+2+5)/4 = 3.5

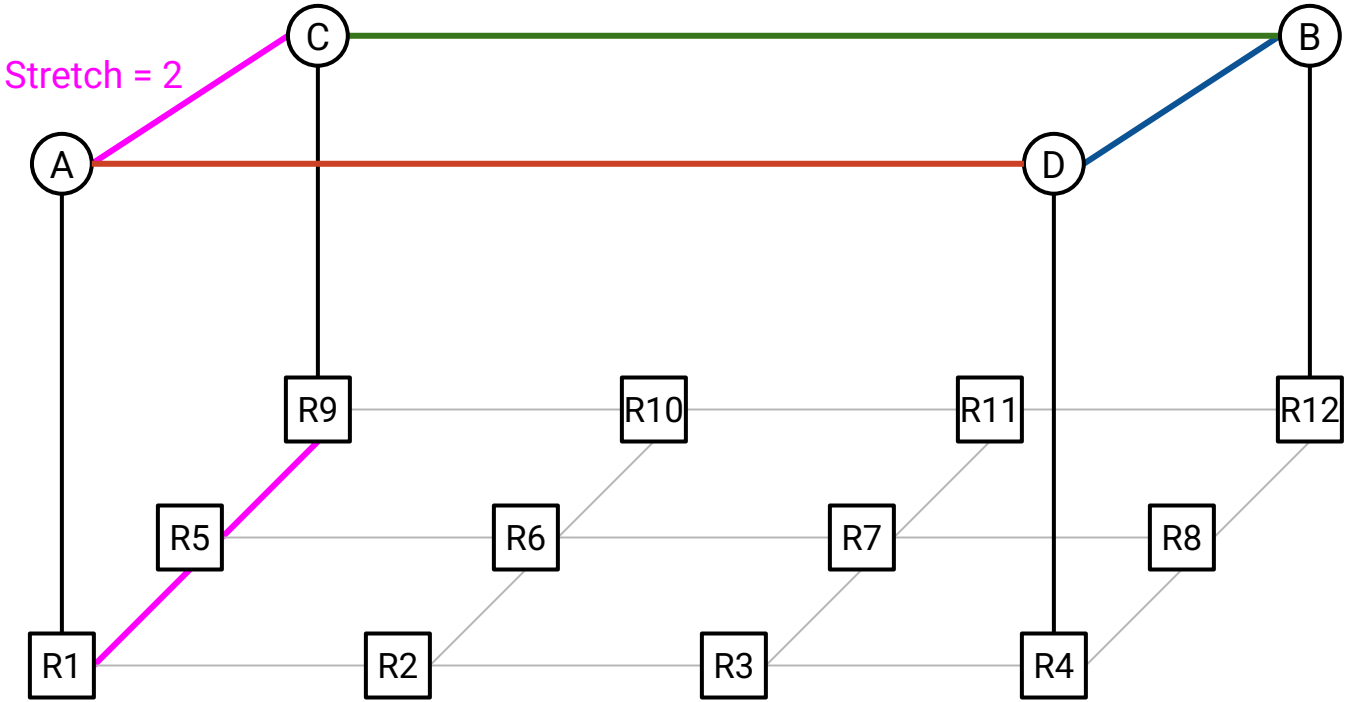
Overlay Ring Topology #2

The average stretch factor depends on how the nodes are numbered.



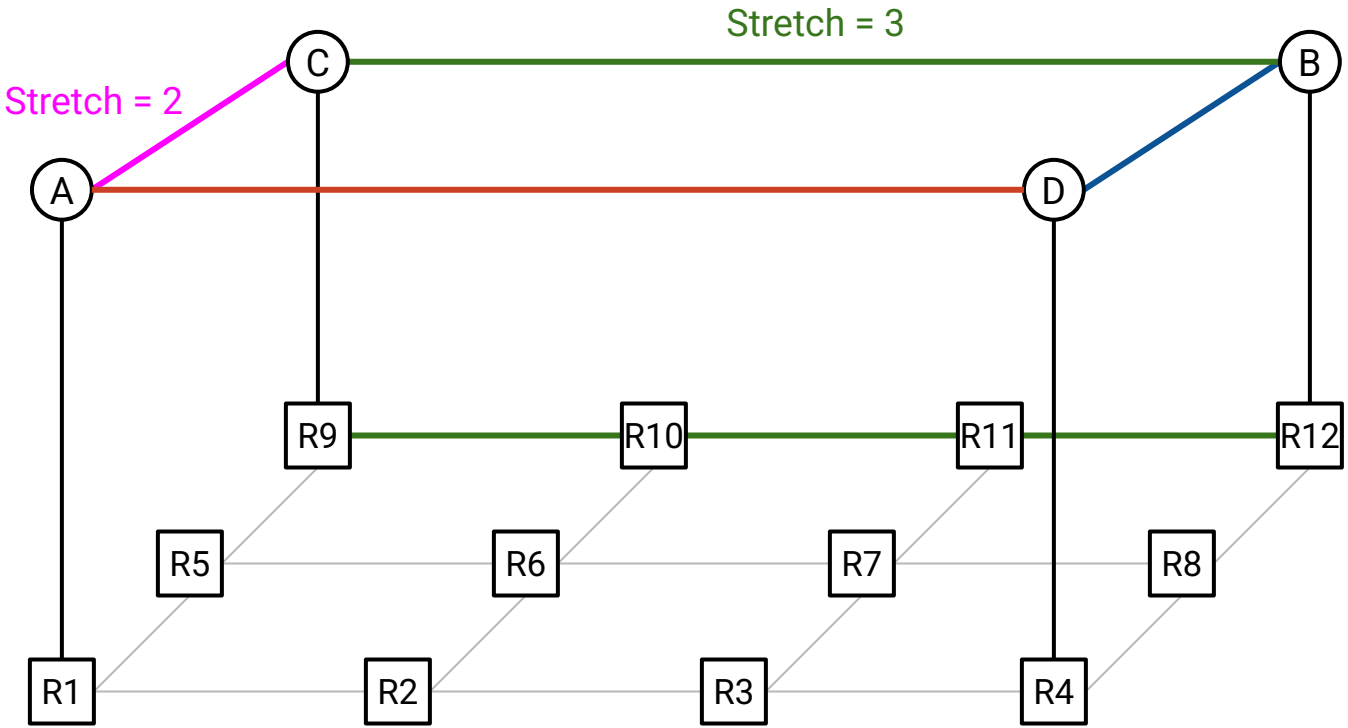
Overlay Ring Topology #2

The average stretch factor depends on how the nodes are numbered.



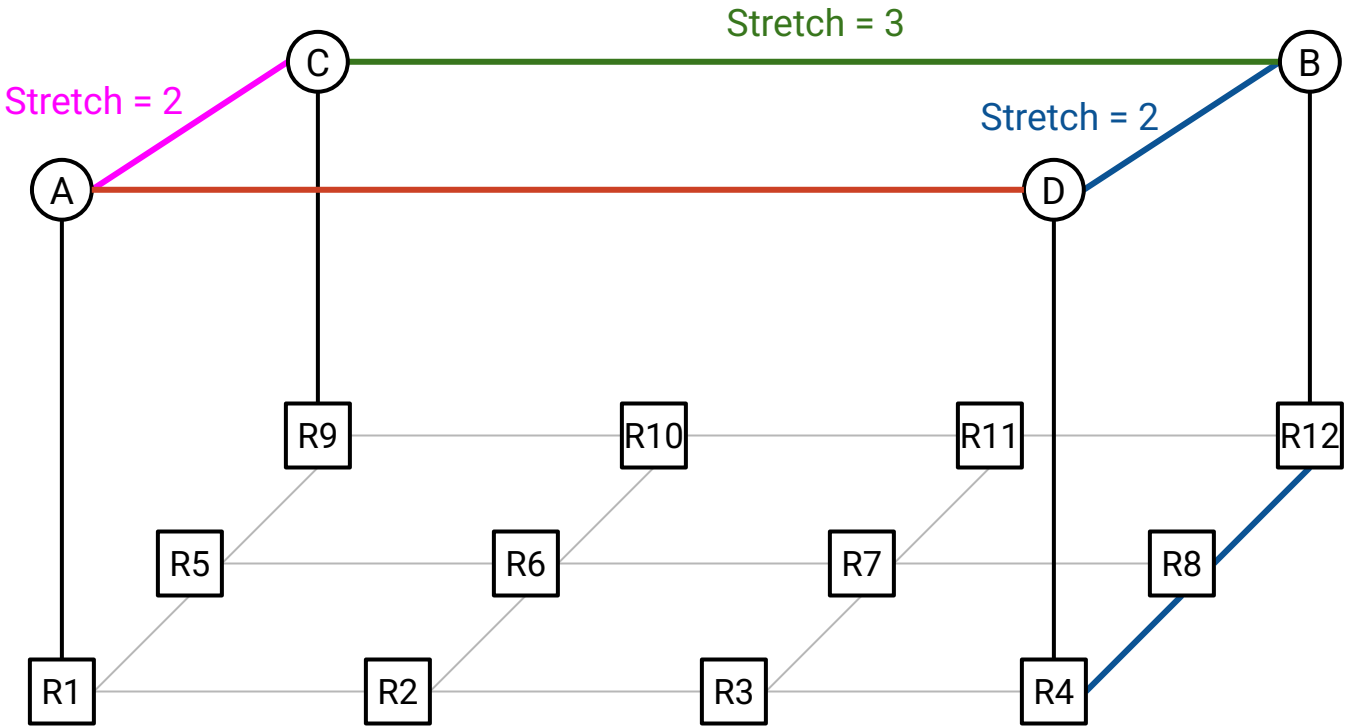
Overlay Ring Topology #2

The average stretch factor depends on how the nodes are numbered.



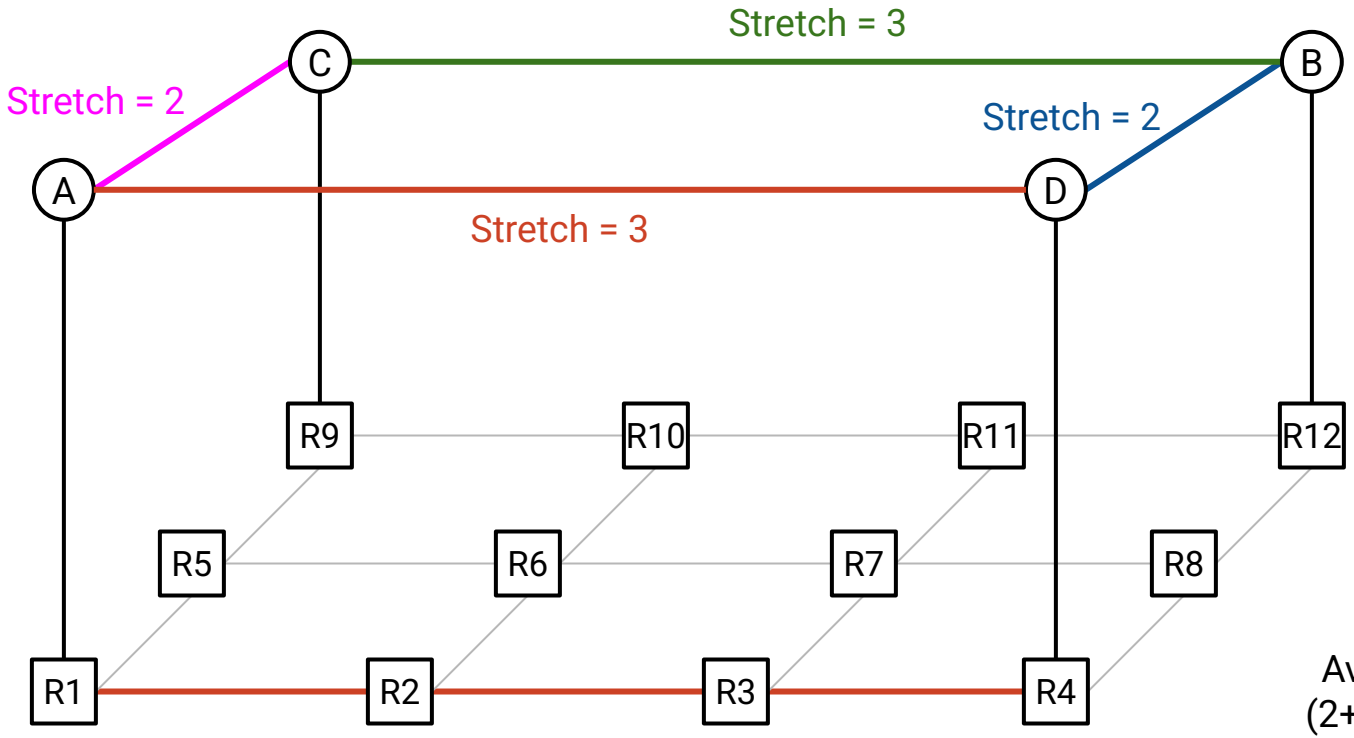
Overlay Ring Topology #2

The average stretch factor depends on how the nodes are numbered.



Overlay Ring Topology #2

The average stretch factor depends on how the nodes are numbered.



Average stretch:
(2+3+2+3)/4 = 2.5

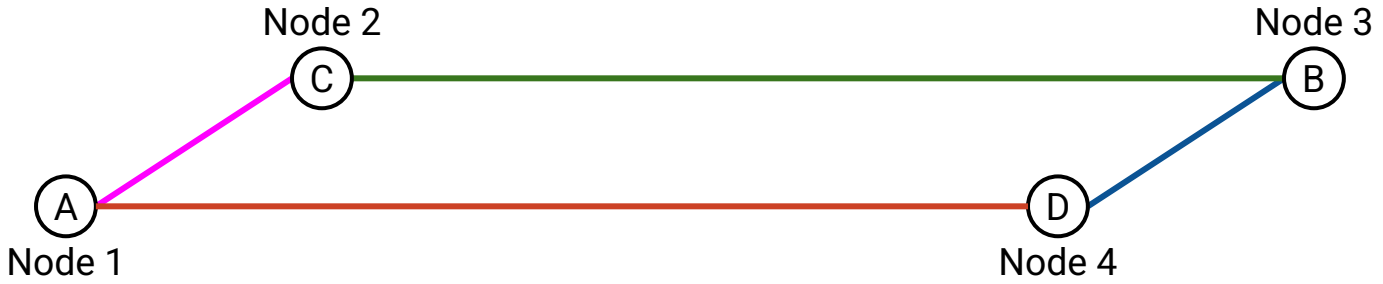
Different Overlay Topologies

Topology 2 achieves better average stretch (i.e. better matches the underlay).

Takeaway: How you draw the overlay network (e.g. the order you connect the hosts) has a big impact on performance.

Overlay Topology 1:
Node 1 = A
Node 2 = C
Node 3 = D
Node 4 = B
Average Stretch: 3.5

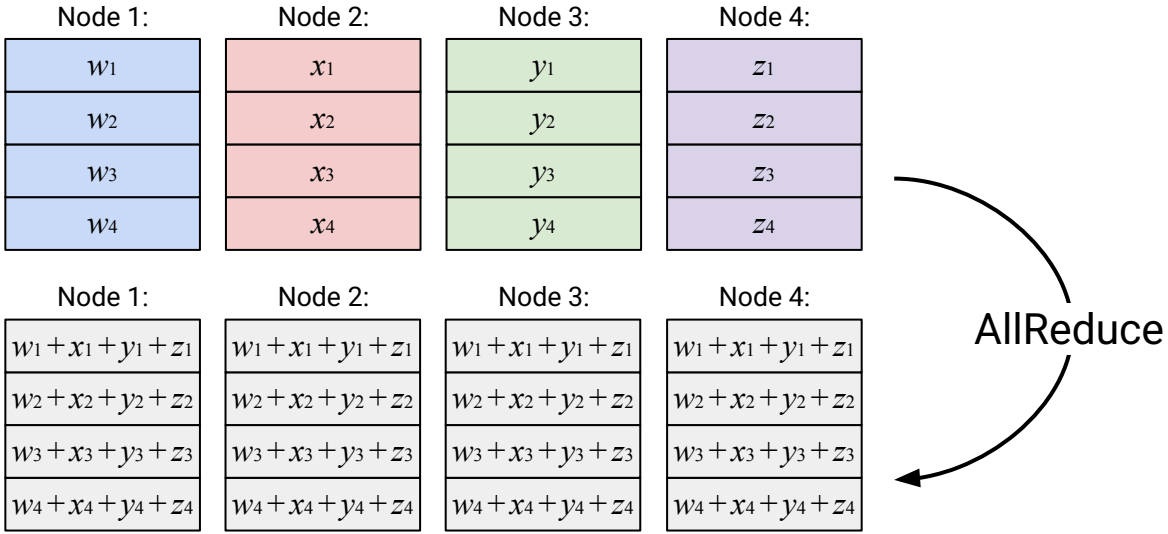
Overlay Topology 2:
Node 1 = A
Node 2 = C
Node 3 = B
Node 4 = D
Average Stretch: 2.5



Summary - Layers of Abstraction

We thought about collectives at 3 different layers of abstraction:

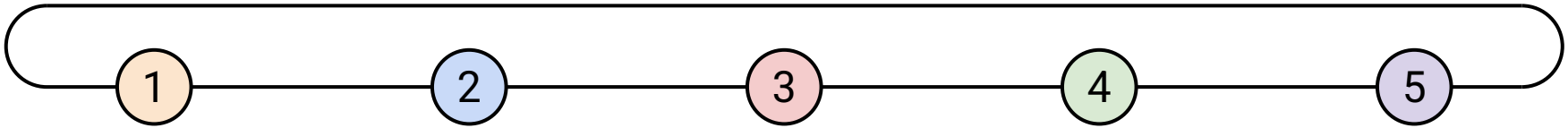
- **Definitions: Users just need to know the input/outputs to use the operations.**



Summary - Layers of Abstraction

We thought about collectives at 3 different layers of abstraction:

- Definitions: Users just need to know the input/outputs to use the operations.
- **Overlay: The data exchanged over virtual links.**



Summary - Layers of Abstraction

We thought about collectives at 3 different layers of abstraction:

- Definitions: Users just need to know the input/outputs to use the operations.
- Overlay: The data exchanged over virtual links.
- **Underlay: How the virtual links correspond to actual physical links.**