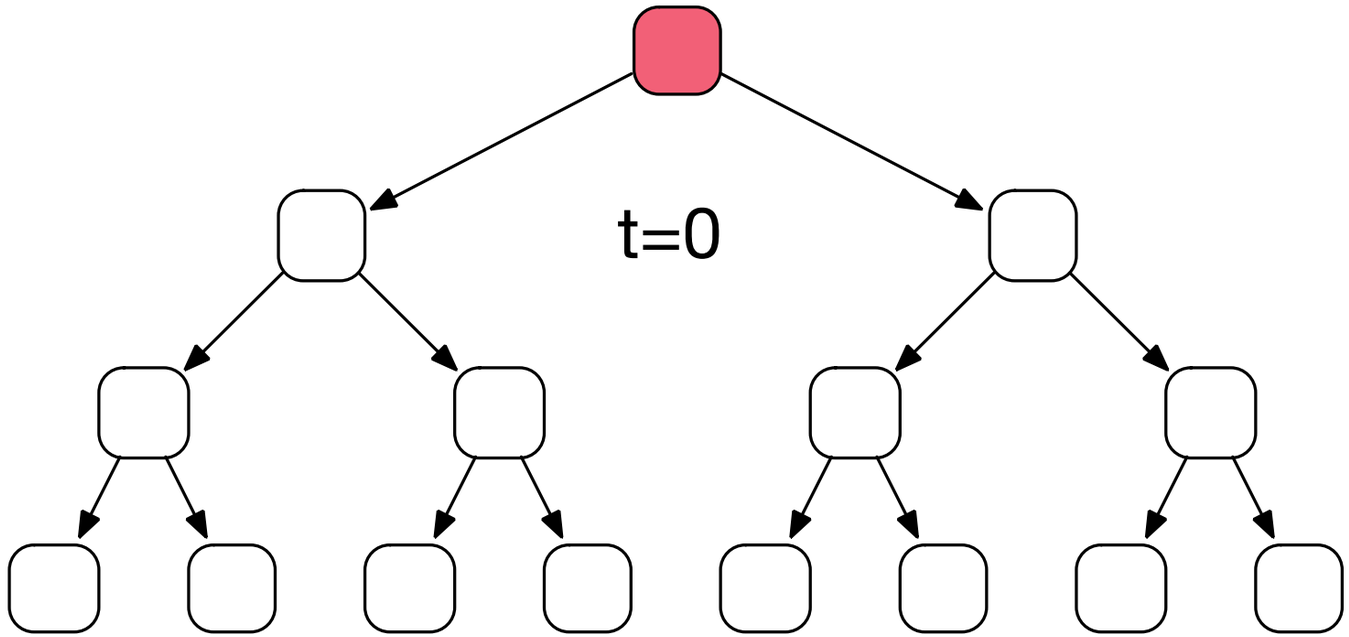
Housekeeping

1. Send Jim your info so that he may create an account for you on our cluster
   - Details on Slack
   - Deadline: 11:59pm
   - If you don't send him your info, you WILL NOT get an account on the cluster
2. Wed class is hands-on. Bring laptops
3. Select the paper for your class presentation
   - Details on Slack
   - Deadline: 02/03, 11:59pm
   - Schedule will be posted on Thurs (02/04)
   - First presentation: 02/10

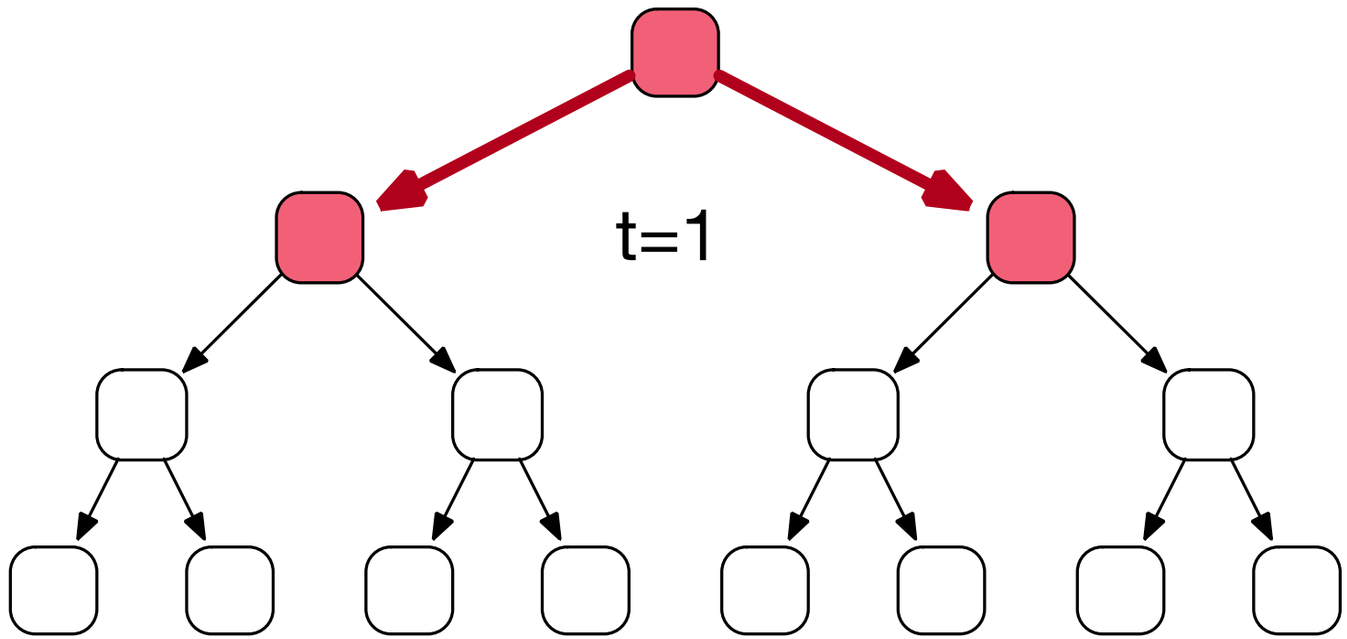
Recap
(Communication Strategies)

Recursive Doubling for One-To-All (Broadcast)



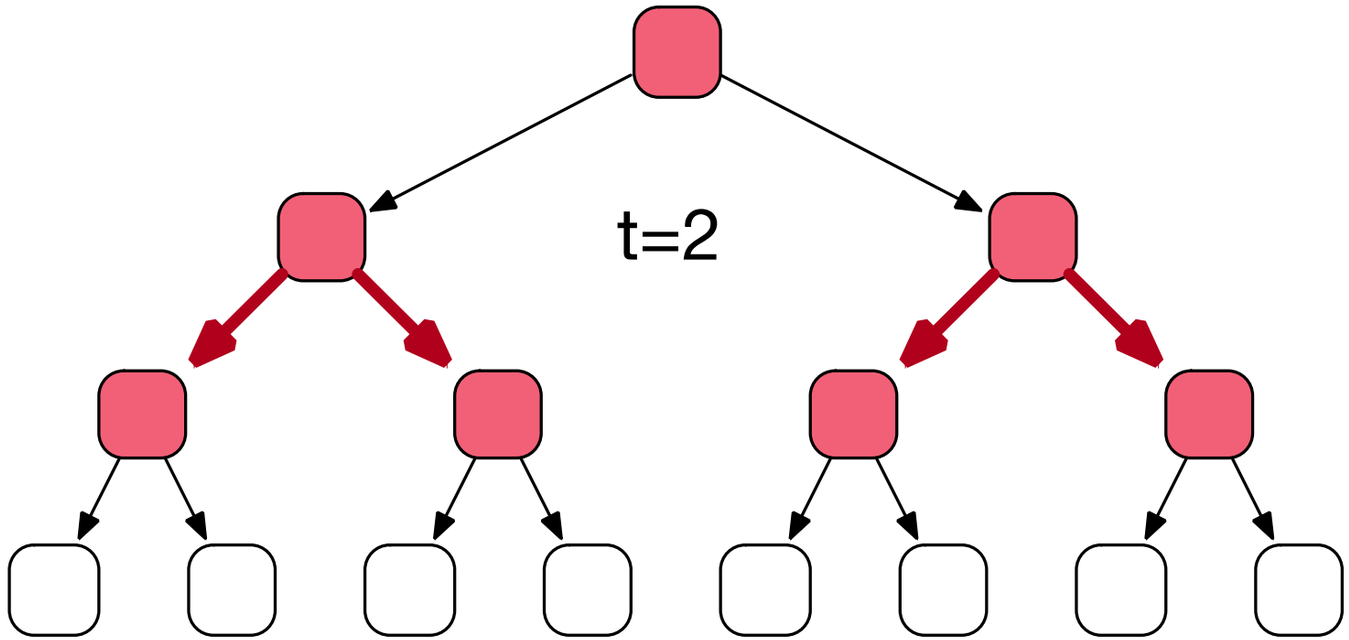
Recap
(Communication Strategies)

Recursive Doubling for One-To-All (Broadcast)



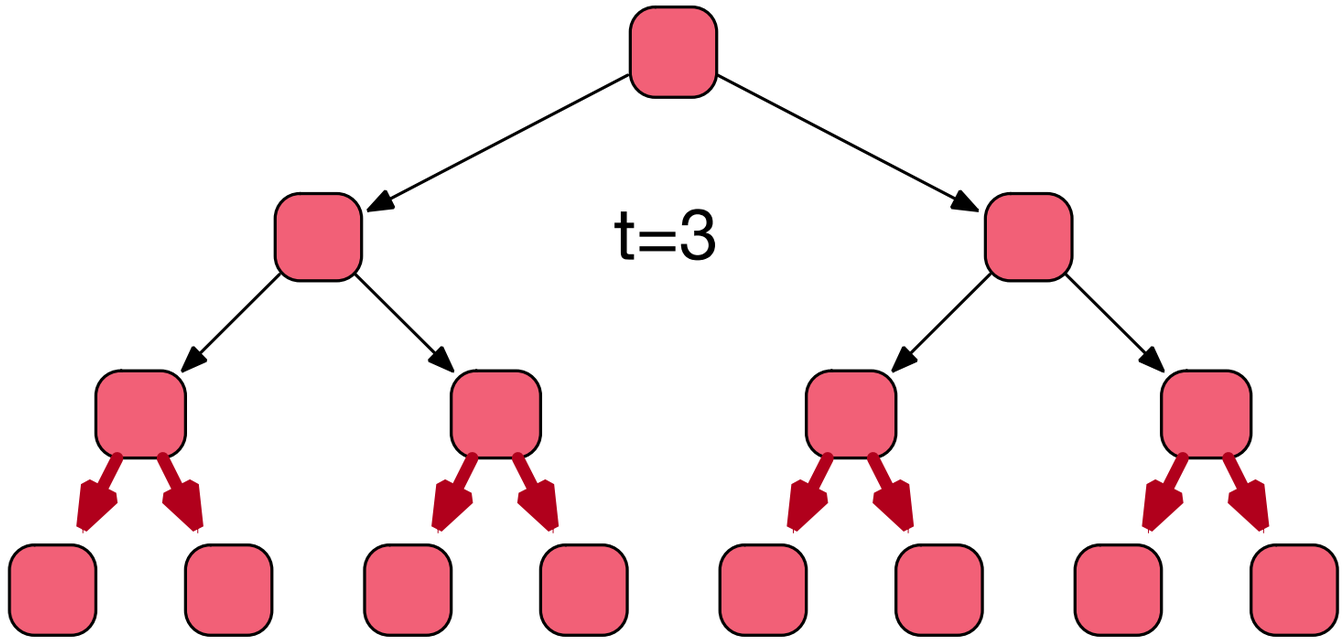
Recap
(Communication Strategies)

Recursive Doubling for One-To-All (Broadcast)

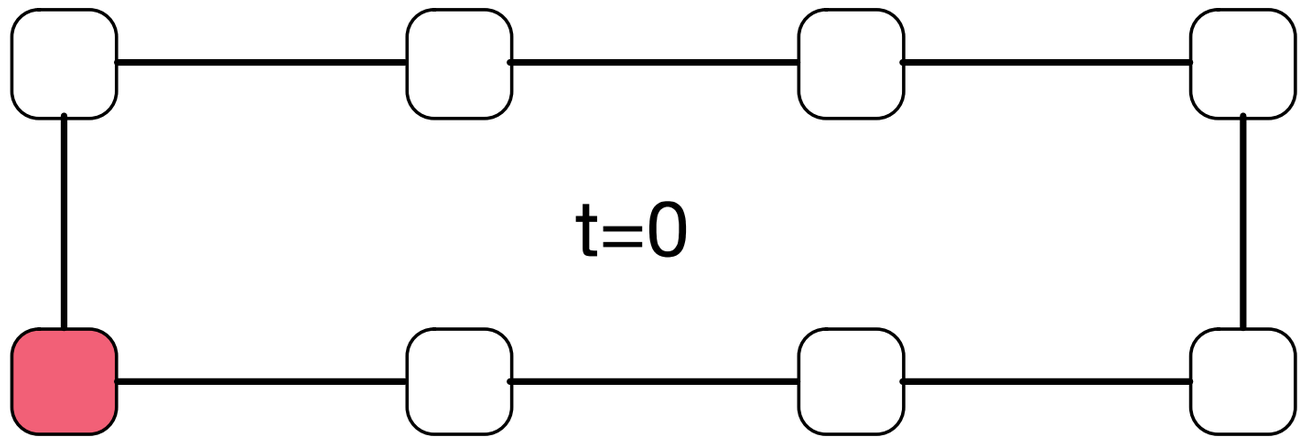


Recap
(Communication Strategies)

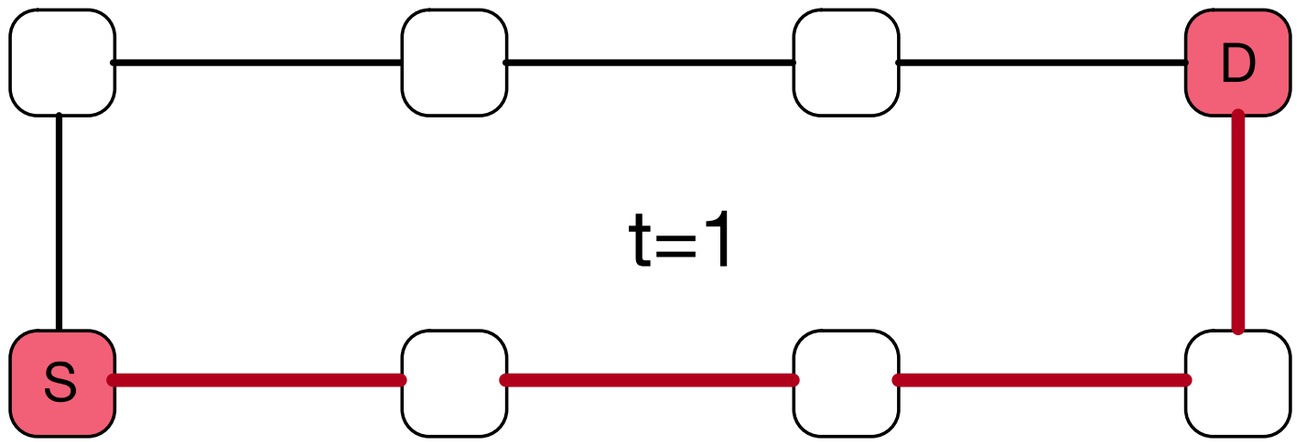
Recursive Doubling for One-To-All (Broadcast)



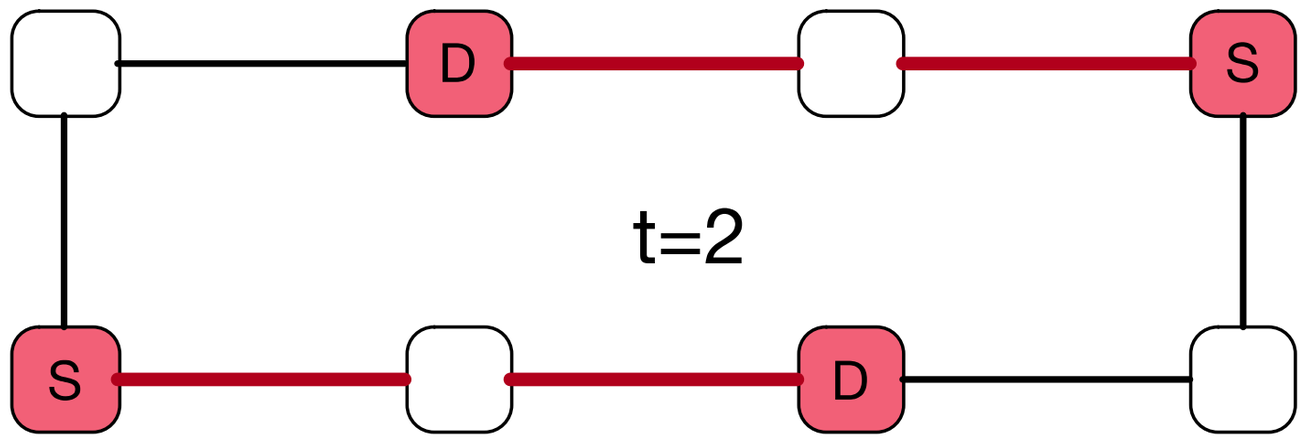
Recap
(Communication Strategies)
in a Ring



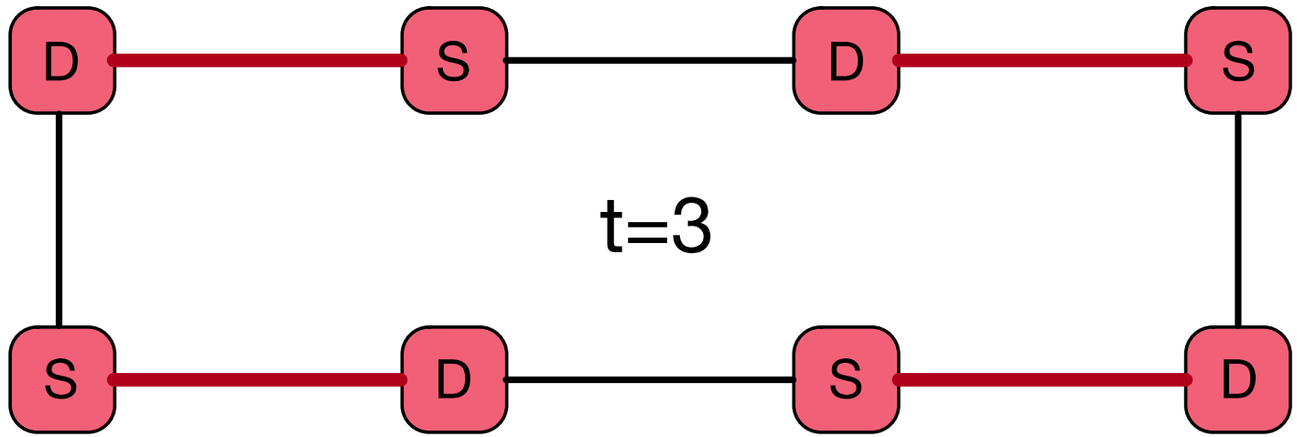
Recap
(Communication Strategies)
in a Ring



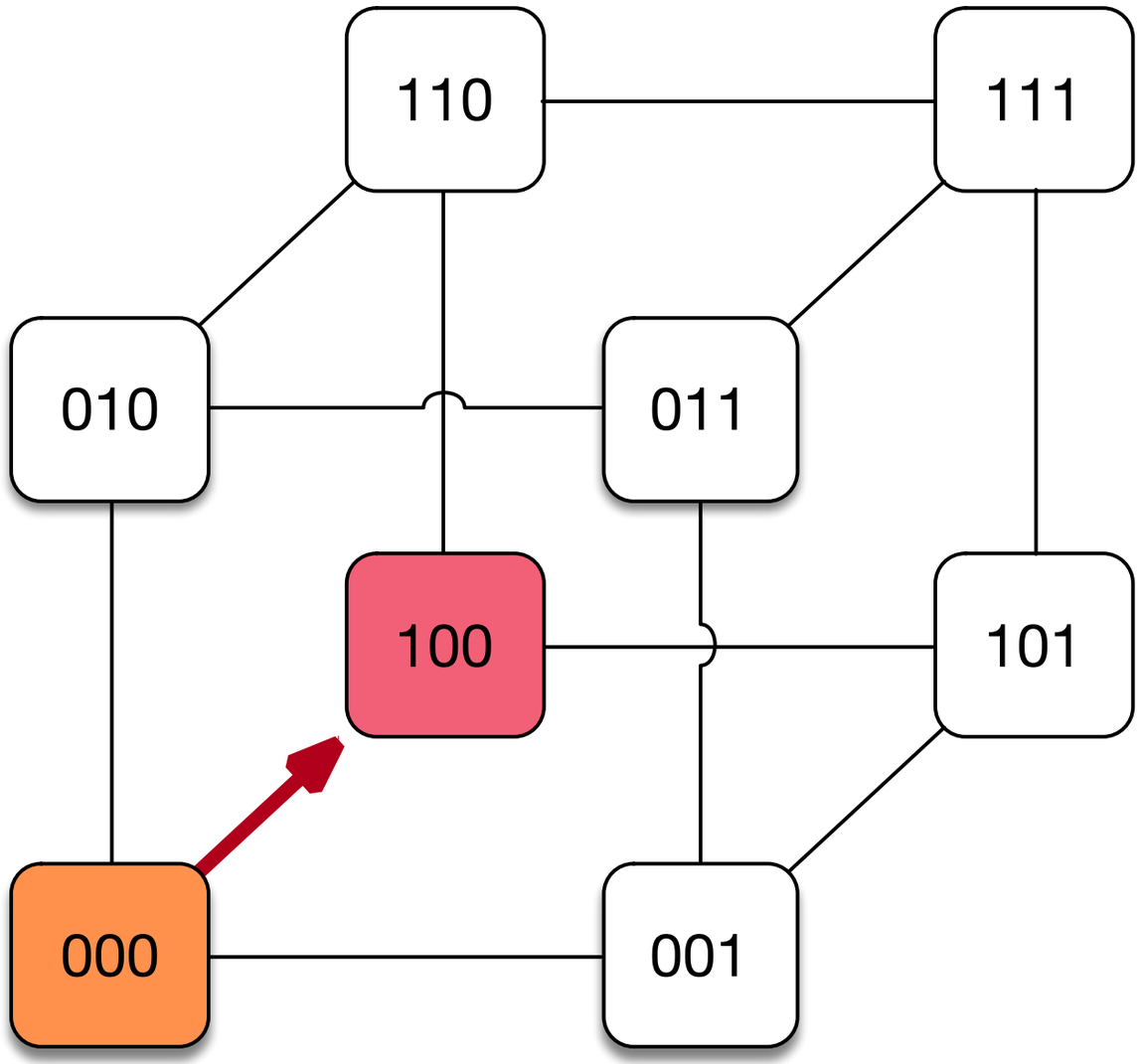
Recap
(Communication Strategies)
in a Ring



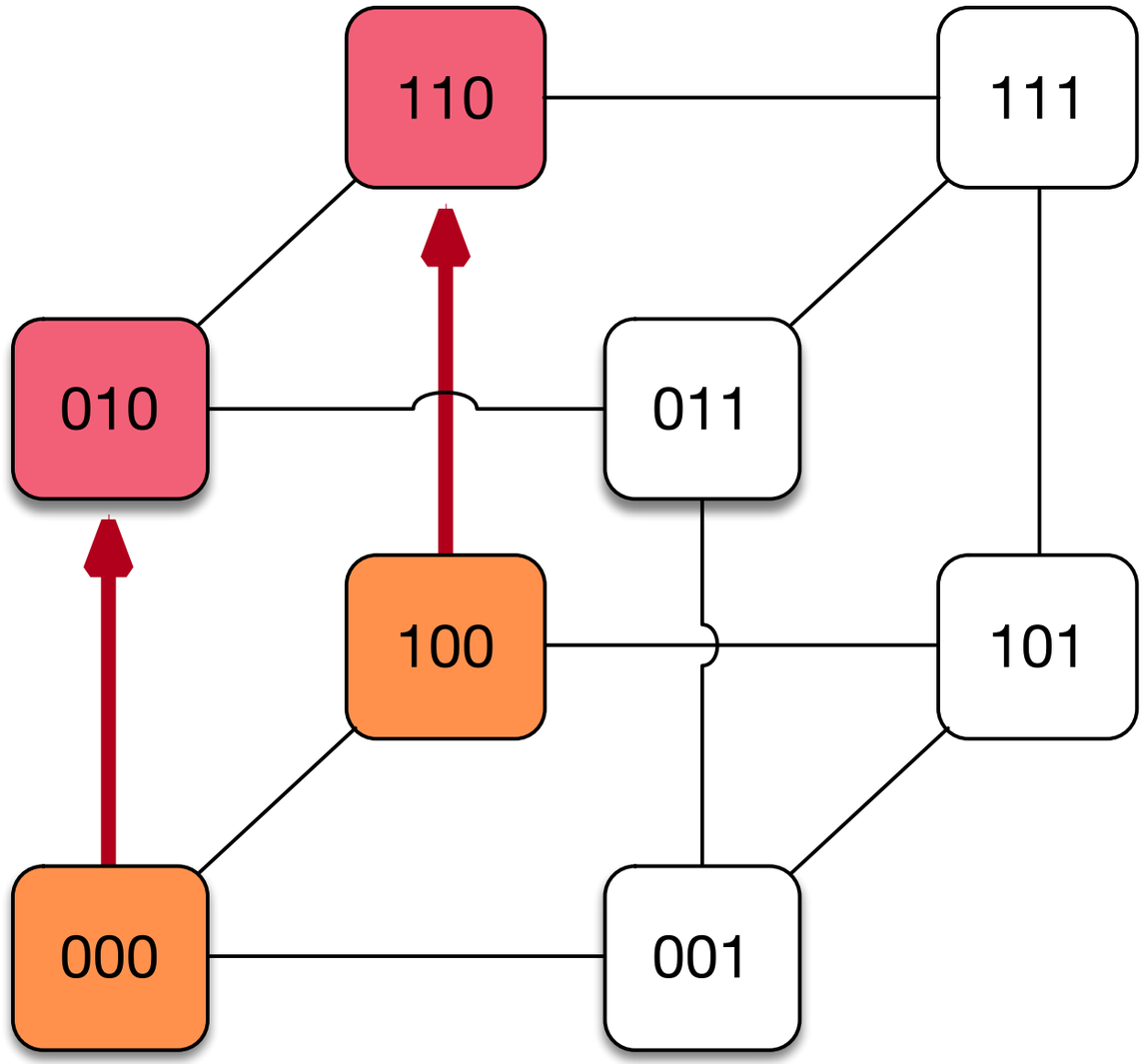
Recap
(Communication Strategies)
in a Ring



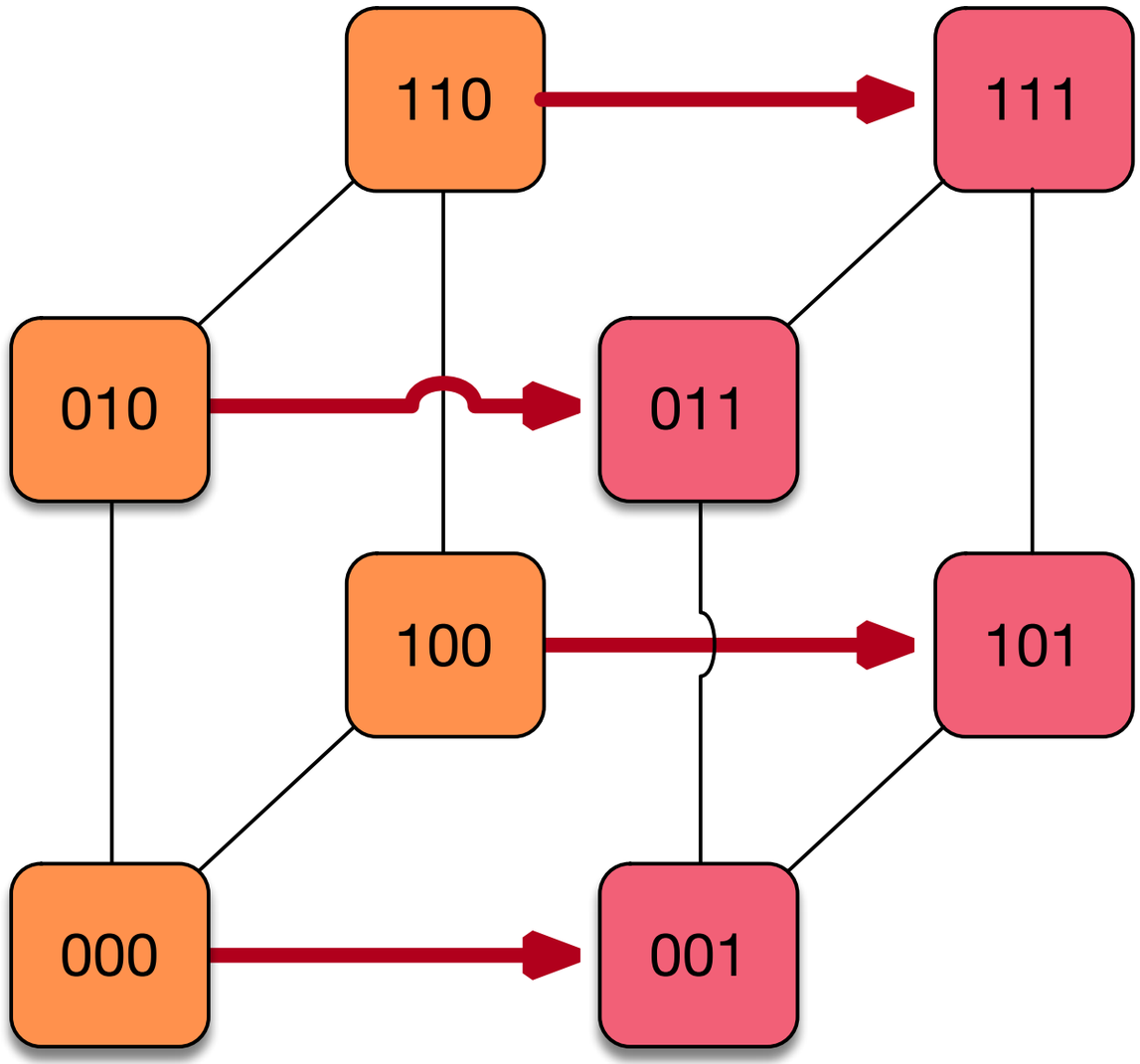
Recap
(Communication Strategies)
in a Hypercube



Recap
(Communication Strategies)
in a Hypercube



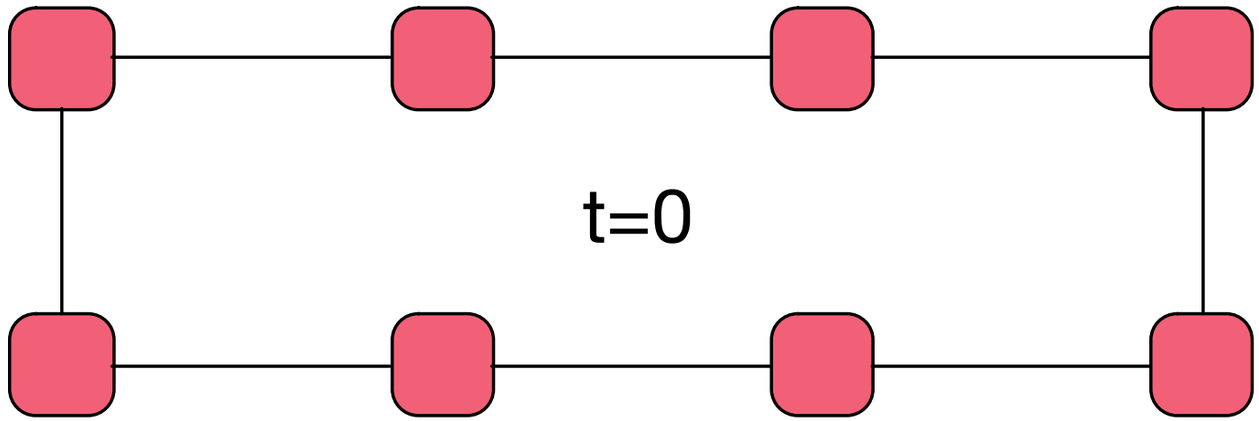
Recap
(Communication Strategies)
in a Hypercube



Recap
(Communication Strategies)

Reduction

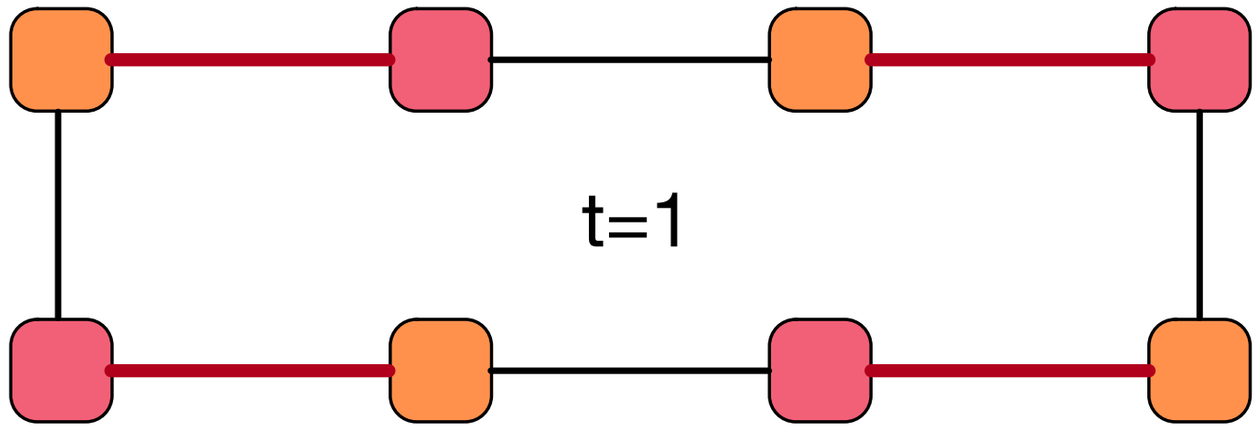
Question….



Recap
(Communication Strategies)

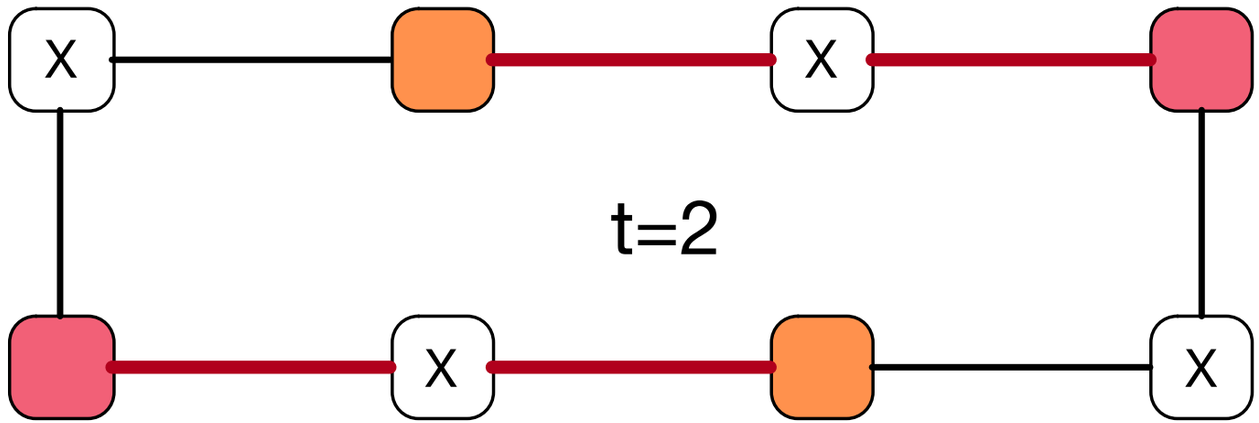
Reduction

Question….



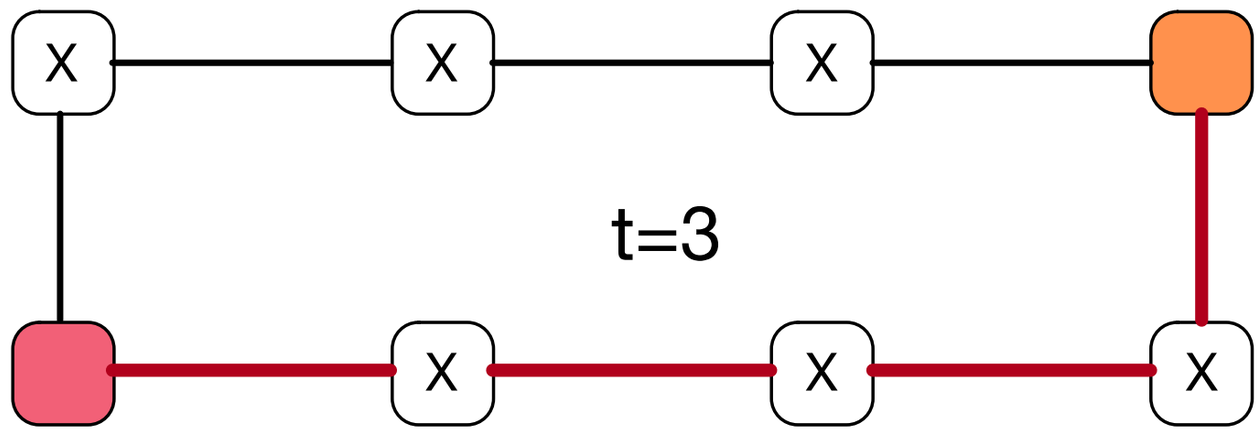
Recap
(Communication Strategies)
Reduction

Question….



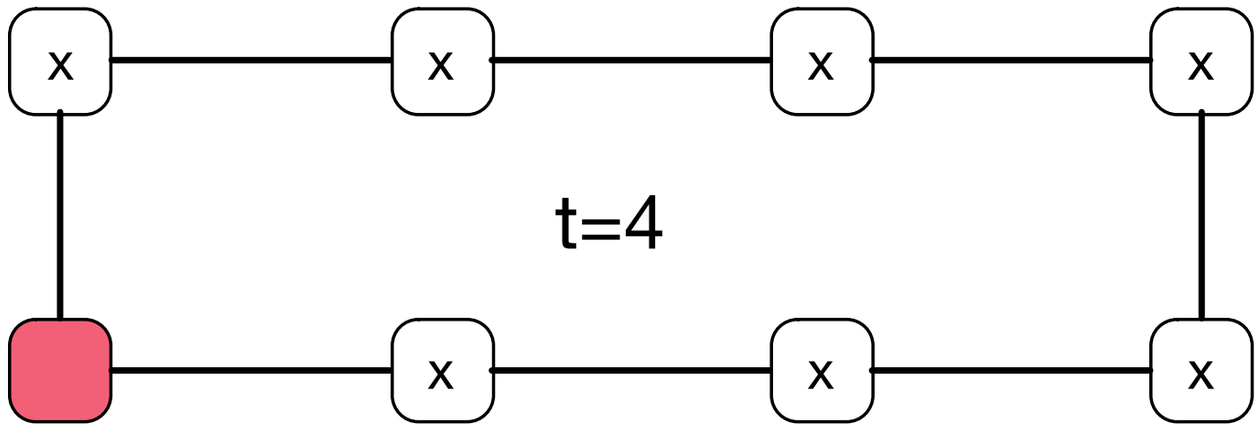
Recap
(Communication Strategies)
Reduction

Question….



Recap
(Communication Strategies)
Reduction

Question….



Cost Revisited → Simplistic Definition

- Cost of sending a single message = $T_{setup} + T_{transmission}$

→ $T_s + T_w$ x Size of data

→ $T_s + T_w x\ m$

Total Cost in a hypercube = $(T_s + T_w x\ m)$ x number of messages

log p messages

Cost → $(T_s + T_w x\ m) x \log p$

Formalized definition of cost

- Cost for sending **s** bytes of data → $T_{ops}^{\alpha,\beta}(s,1) = \alpha + s.\beta$
  - α → Latency
  - β → Bandwidth
  - Each processor can either send/receive at any given time

- E.g.: Ring Network → P processors
- What is the cost if all messages are sent in sequence
  - Volume = P.s
  - Cost = P.$(\alpha + s.\beta)$
- What is the cost if messages are sent simultaneously
  - Cost = $\alpha + s.\beta$

Cost in a Ring

- Ring Network → P processors
- Cost if all messages are sent in sequence
  - Volume = P.s
  - Cost → $T_{seq}^{\alpha,\beta}(s,P) = P.(\alpha + s.\beta)$
- Cost if messages are sent simultaneously
  - P = 1
  - Cost → $T_{sim}^{\alpha,\beta}(s,1) = \alpha + s.\beta$

Strategies for Collective Communication

✓ Broadcast

- All to All (Broadcast and Reduction)
- Scatter-Gather

All to All vs. Broadcast

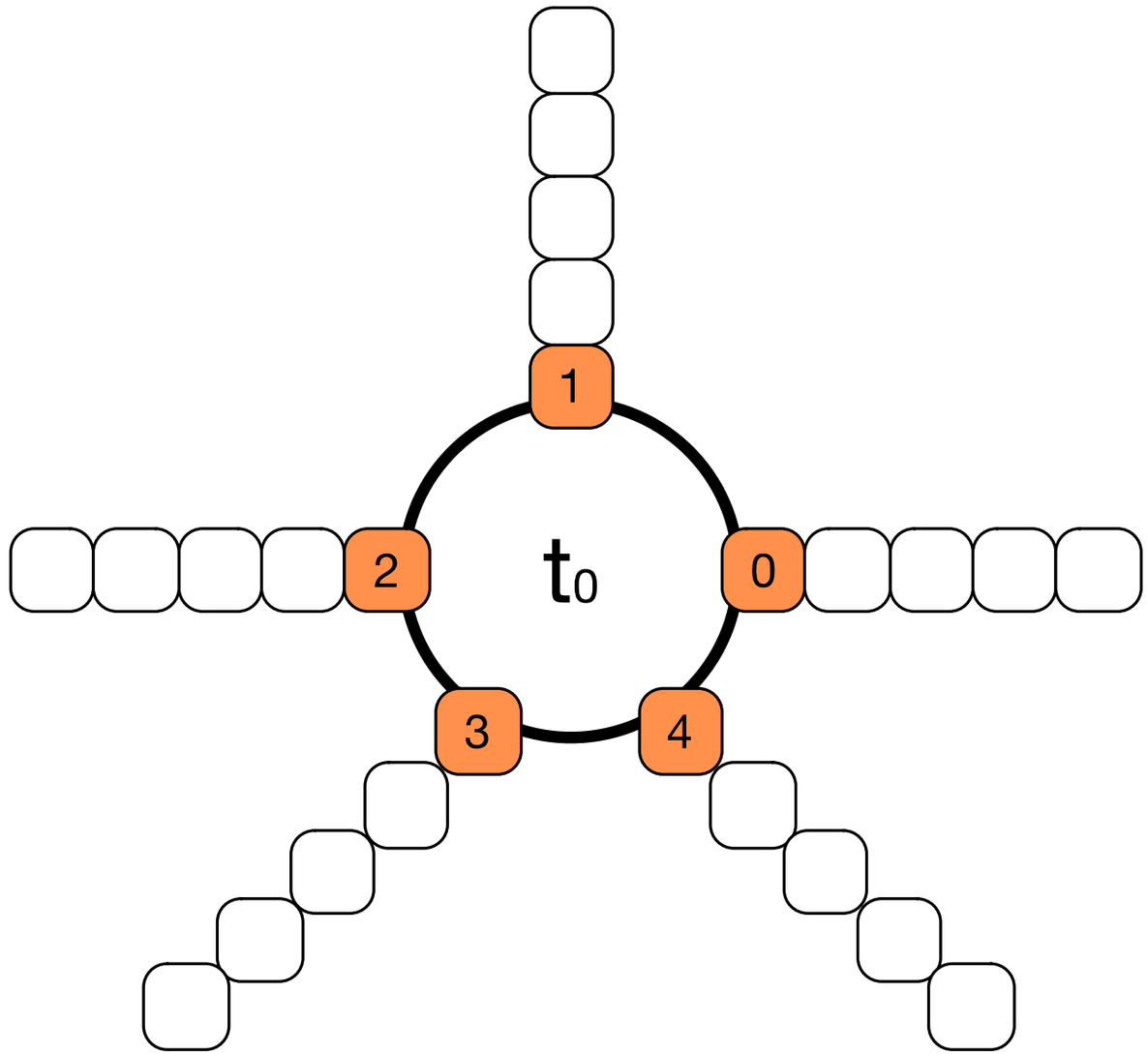
- Generalization of broadcast in which each processor is the source as well as destination.
- A process sends the same m-word message to every other process, but different processes may broadcast different messages.

All-All RING

- Simplest approach: perform p one-to-all broadcasts. This is not the most efficient way.
- Each node first sends to one of its neighbors the data it needs to broadcast.
- In subsequent steps, it forwards the data received from one of its neighbors to its other neighbor.
- Terminates in *p-1* steps.

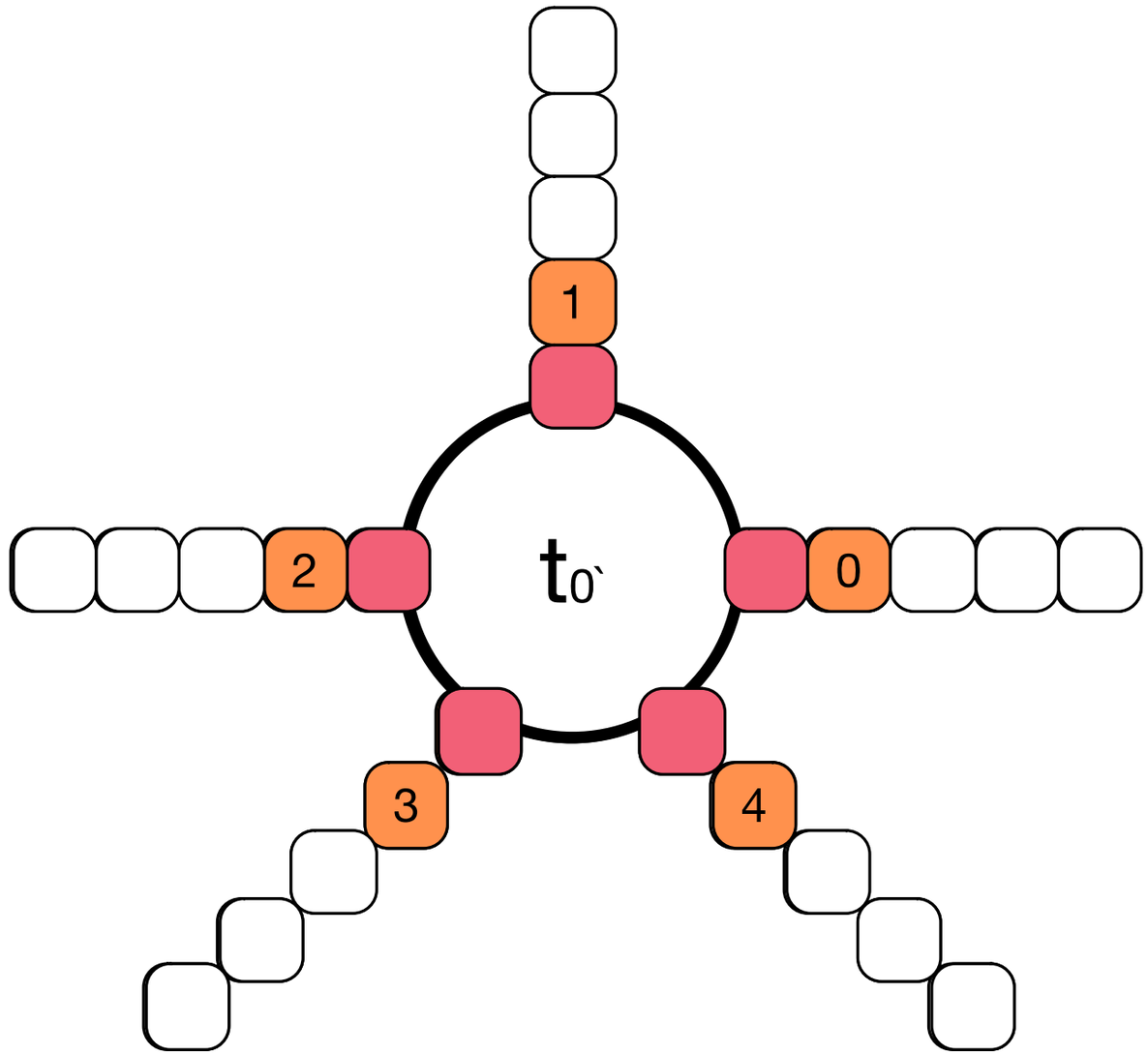
All-All RING

- Simplest approach: perform p one-to-all broadcasts. This is not the most efficient way.
- Each node first sends to one of its neighbors the data it needs to broadcast.
- In subsequent steps, it forwards the data received from one of its neighbors to its other neighbor.
- Terminates in *p-1* steps.



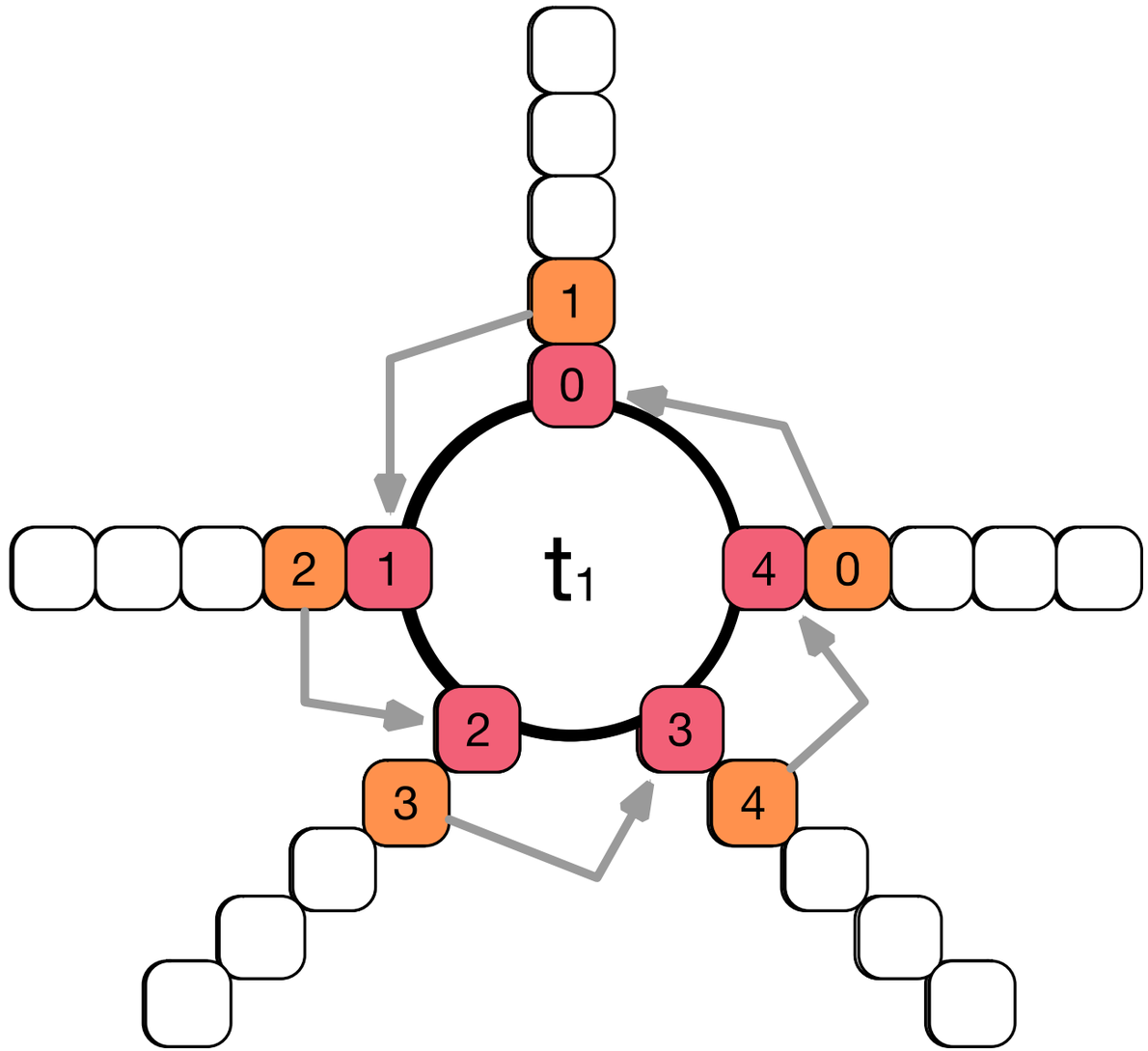
All-All RING

- Simplest approach: perform p one-to-all broadcasts. This is not the most efficient way.
- Each node first sends to one of its neighbors the data it needs to broadcast.
- In subsequent steps, it forwards the data received from one of its neighbors to its other neighbor.
- Terminates in *p-1* steps.



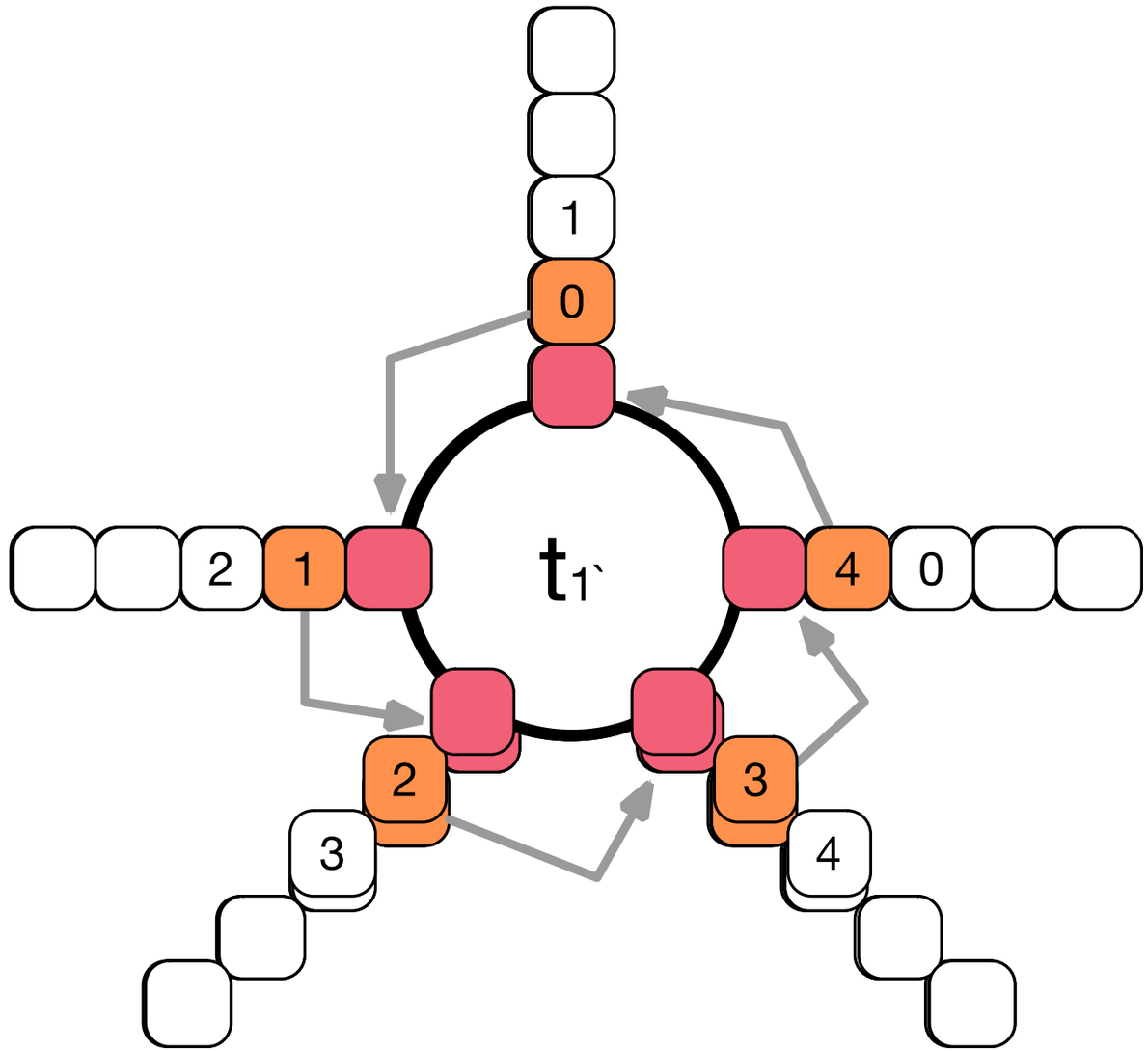
All-All RING

- Simplest approach: perform p one-to-all broadcasts. This is not the most efficient way.
- Each node first sends to one of its neighbors the data it needs to broadcast.
- In subsequent steps, it forwards the data received from one of its neighbors to its other neighbor.
- Terminates in *p-1* steps.



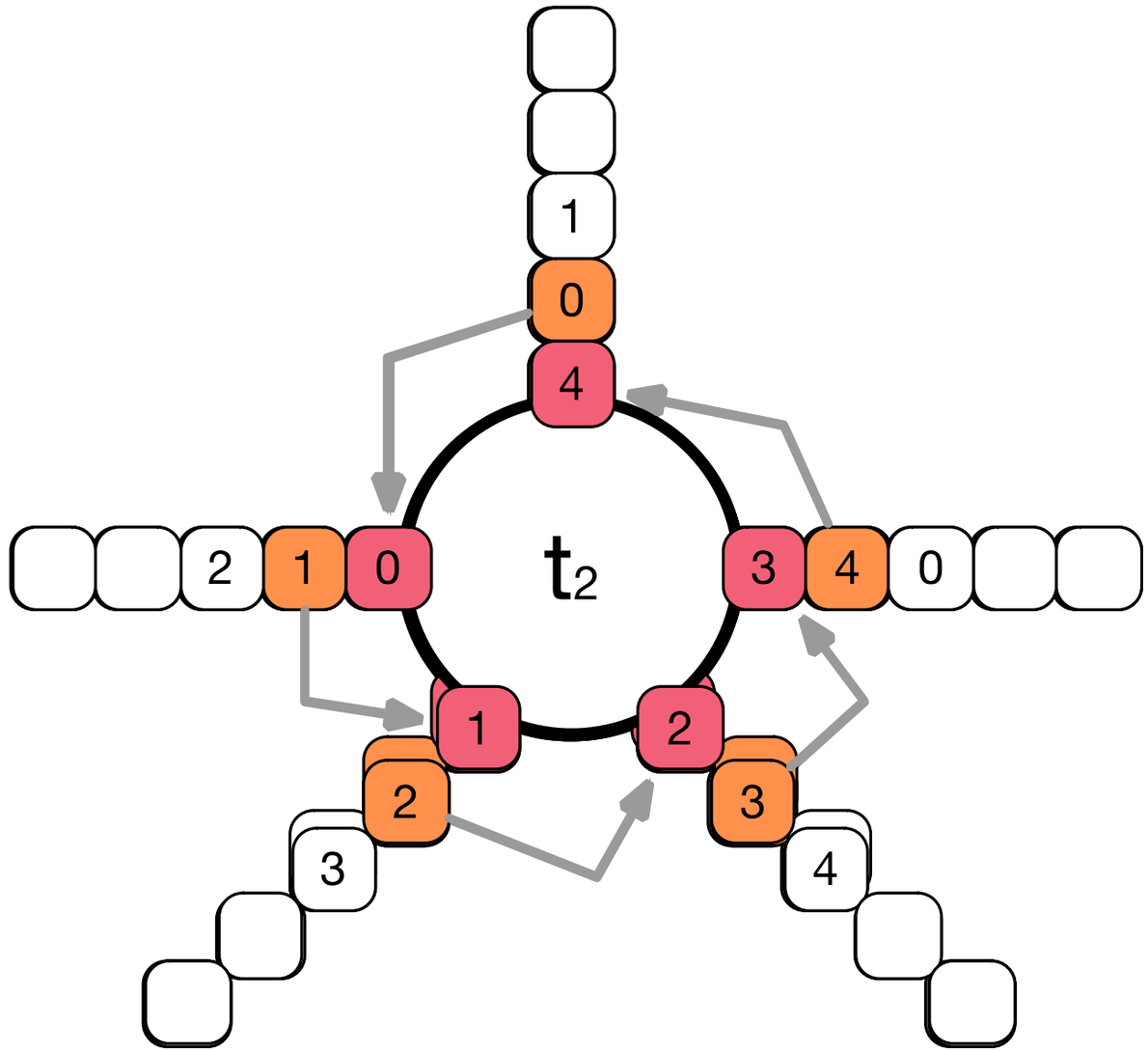
All-All RING

- Simplest approach: perform p one-to-all broadcasts. This is not the most efficient way.
- Each node first sends to one of its neighbors the data it needs to broadcast.
- In subsequent steps, it forwards the data received from one of its neighbors to its other neighbor.
- Terminates in *p-1* steps.



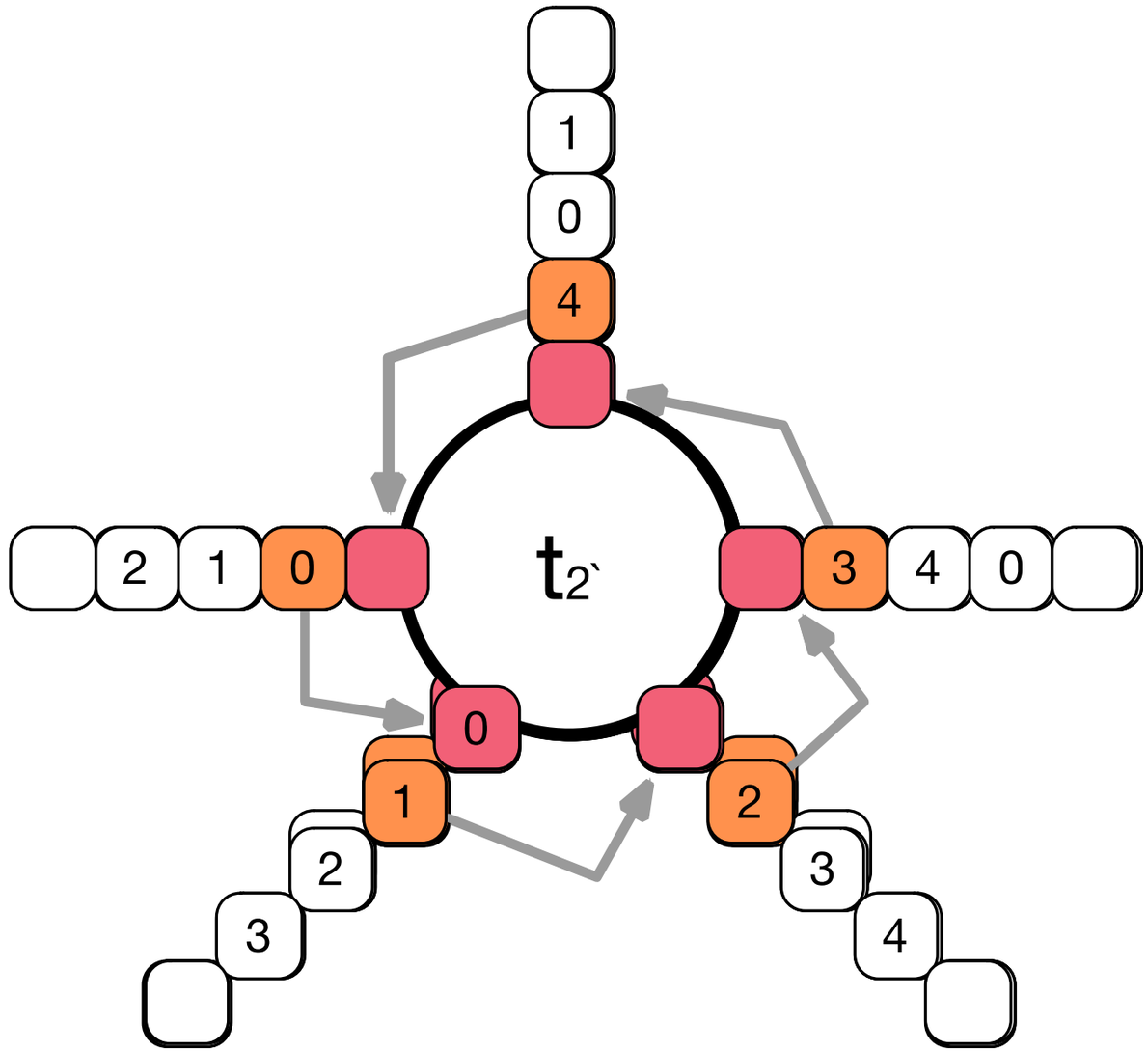
All-All RING

- Simplest approach: perform p one-to-all broadcasts. This is not the most efficient way.
- Each node first sends to one of its neighbors the data it needs to broadcast.
- In subsequent steps, it forwards the data received from one of its neighbors to its other neighbor.
- Terminates in *p-1* steps.



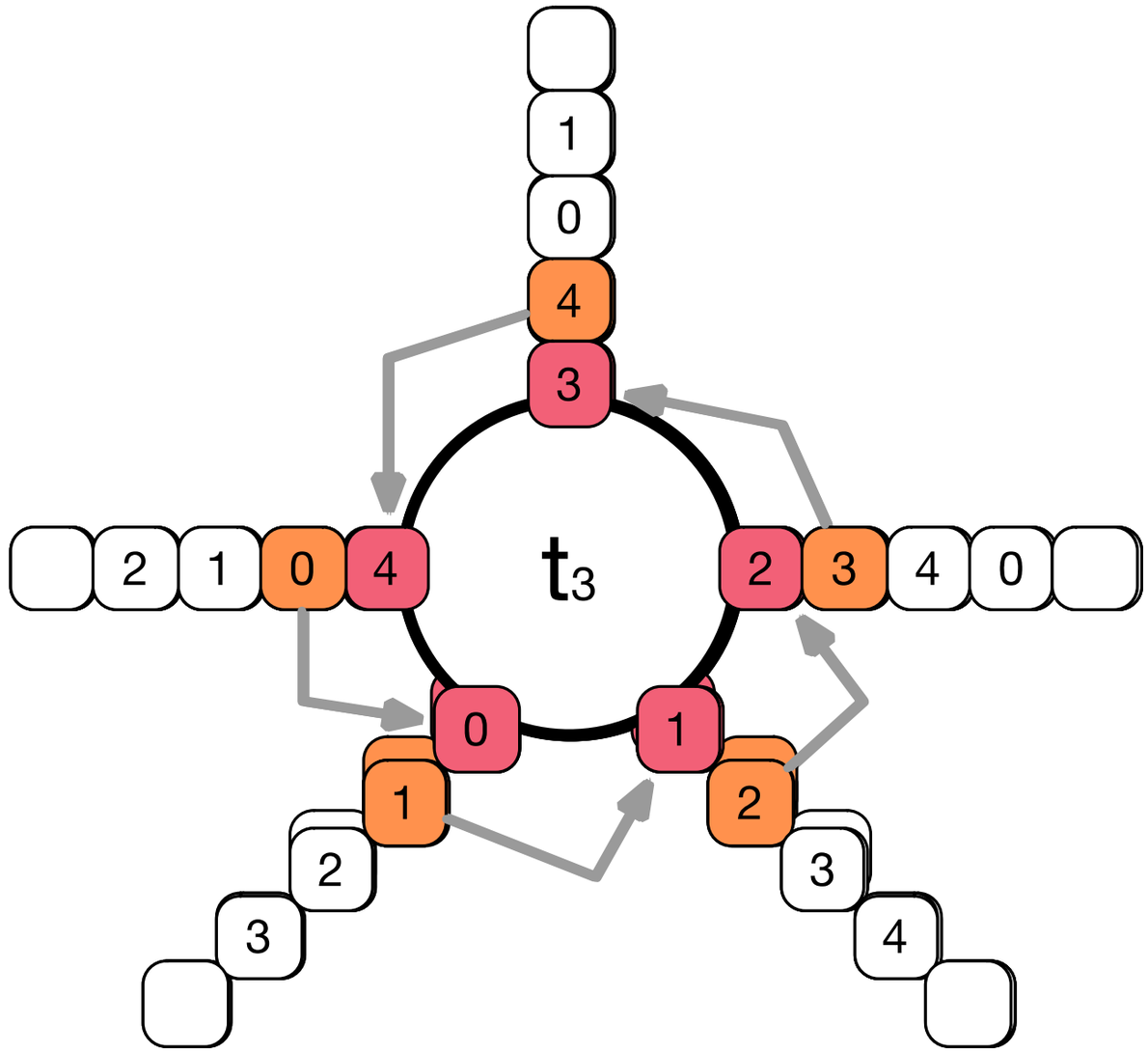
All-All RING

- Simplest approach: perform p one-to-all broadcasts. This is not the most efficient way.
- Each node first sends to one of its neighbors the data it needs to broadcast.
- In subsequent steps, it forwards the data received from one of its neighbors to its other neighbor.
- Terminates in *p-1* steps.



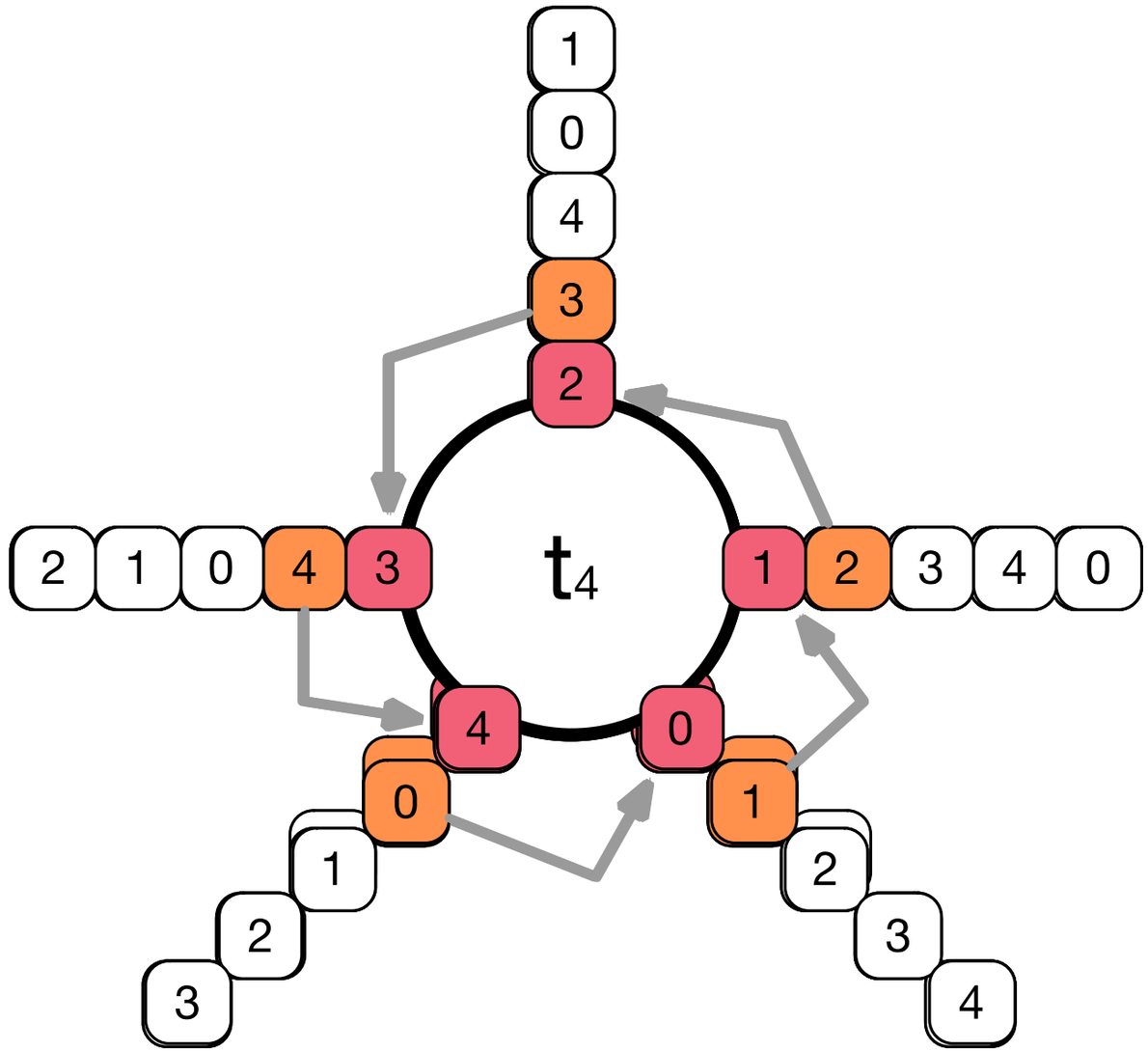
All-All RING

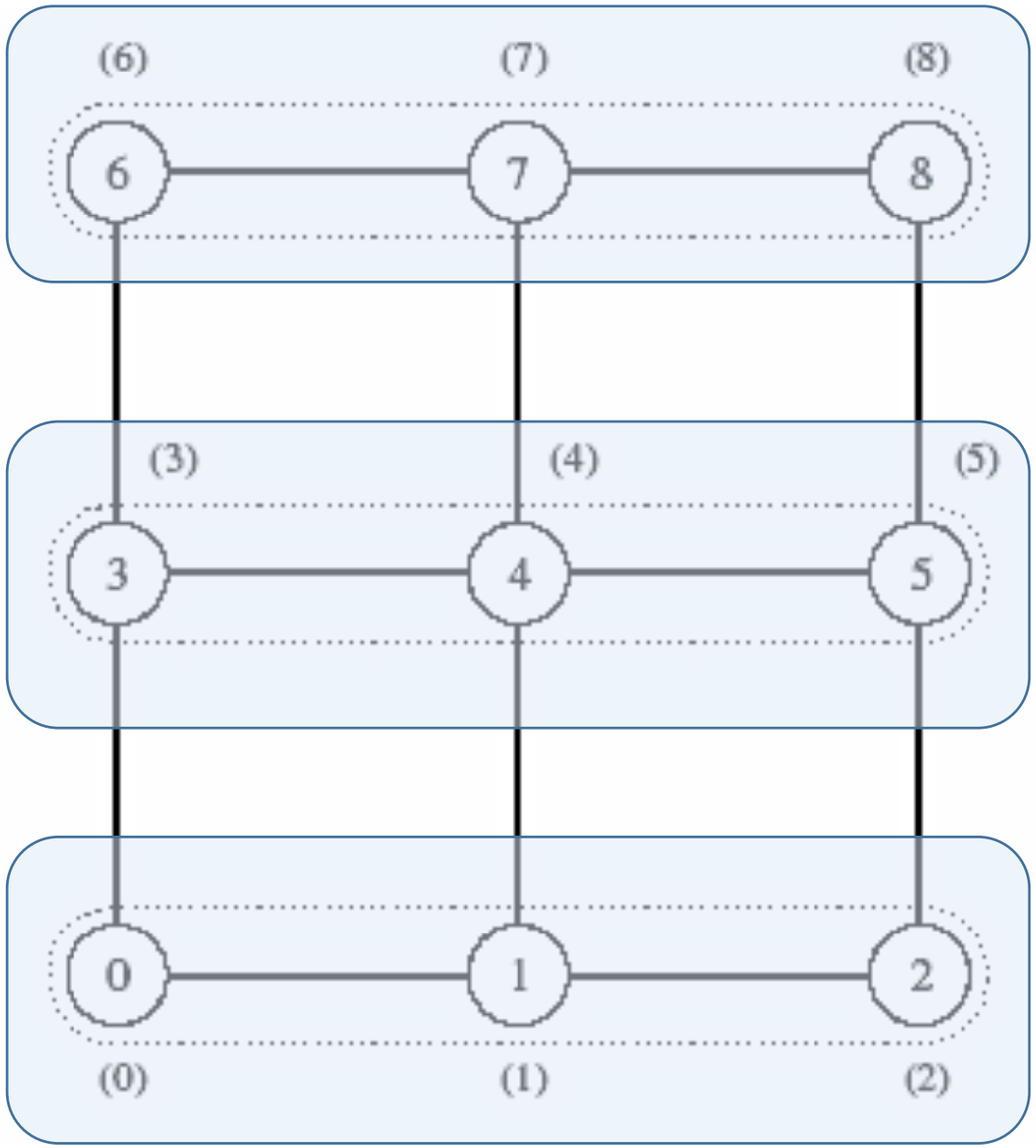
- Simplest approach: perform p one-to-all broadcasts. This is not the most efficient way.
- Each node first sends to one of its neighbors the data it needs to broadcast.
- In subsequent steps, it forwards the data received from one of its neighbors to its other neighbor.
- Terminates in *p-1* steps.

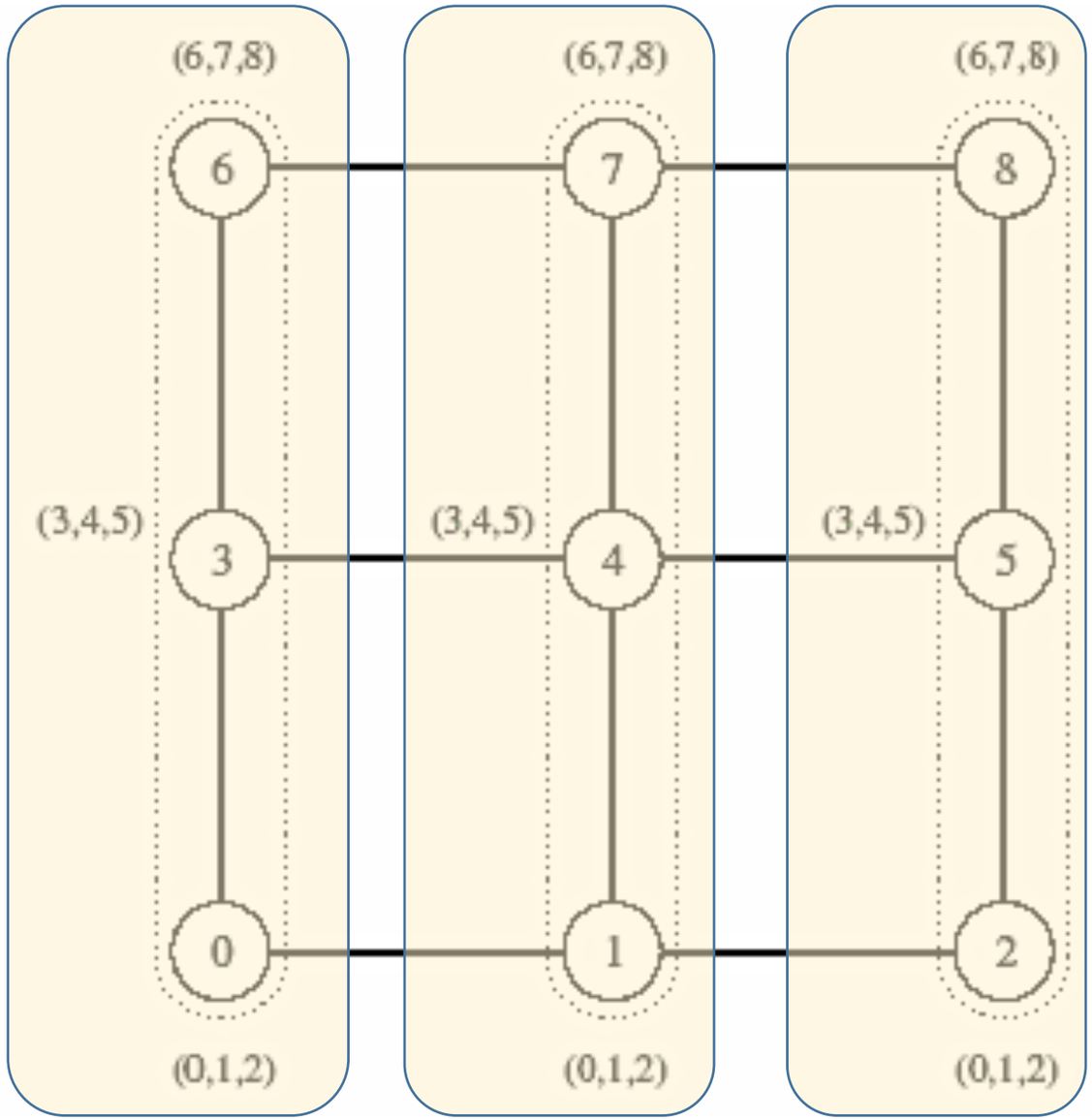


All-All RING

- Simplest approach: perform p one-to-all broadcasts. This is not the most efficient way.
- Each node first sends to one of its neighbors the data it needs to broadcast.
- In subsequent steps, it forwards the data received from one of its neighbors to its other neighbor.
- Terminates in *p-1* steps.



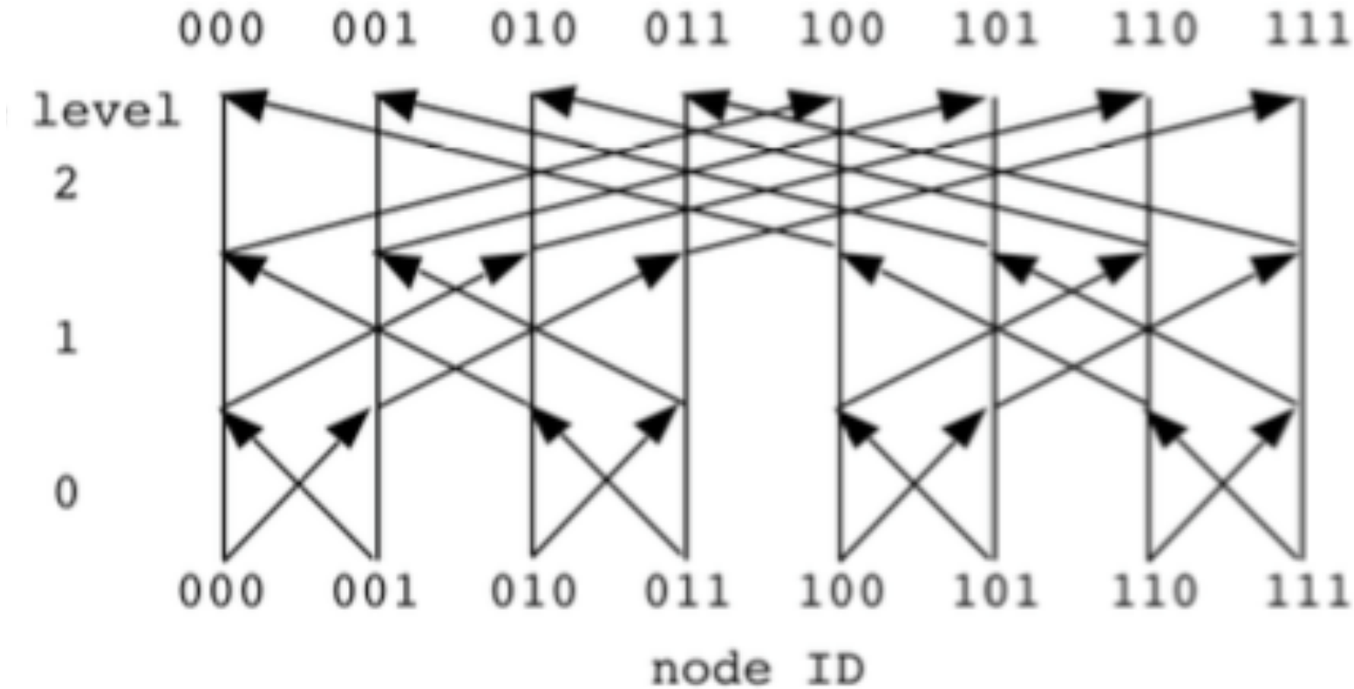




All-to-All in a Mesh

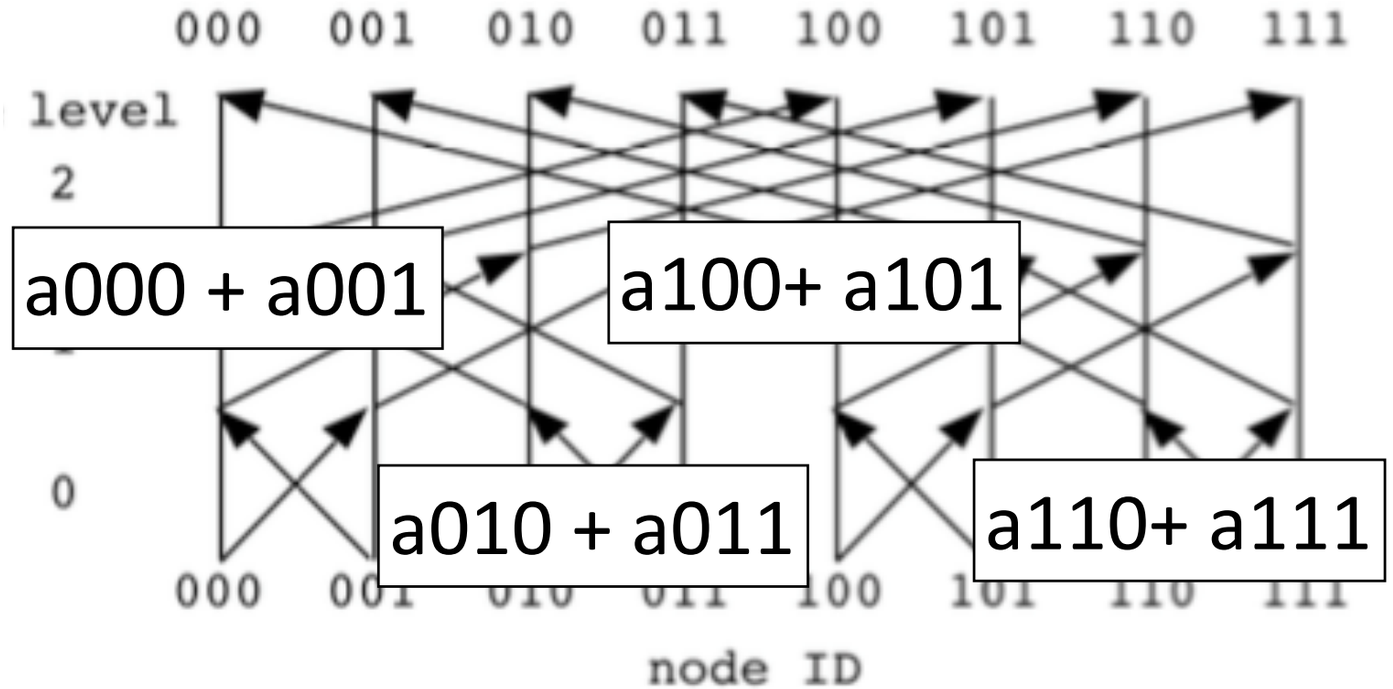
All-Gather in a Hypercube

- Butterfly → a different way to think about hypercubes



All-Gather in a Hypercube

- Butterfly → a different way to think about hypercubes

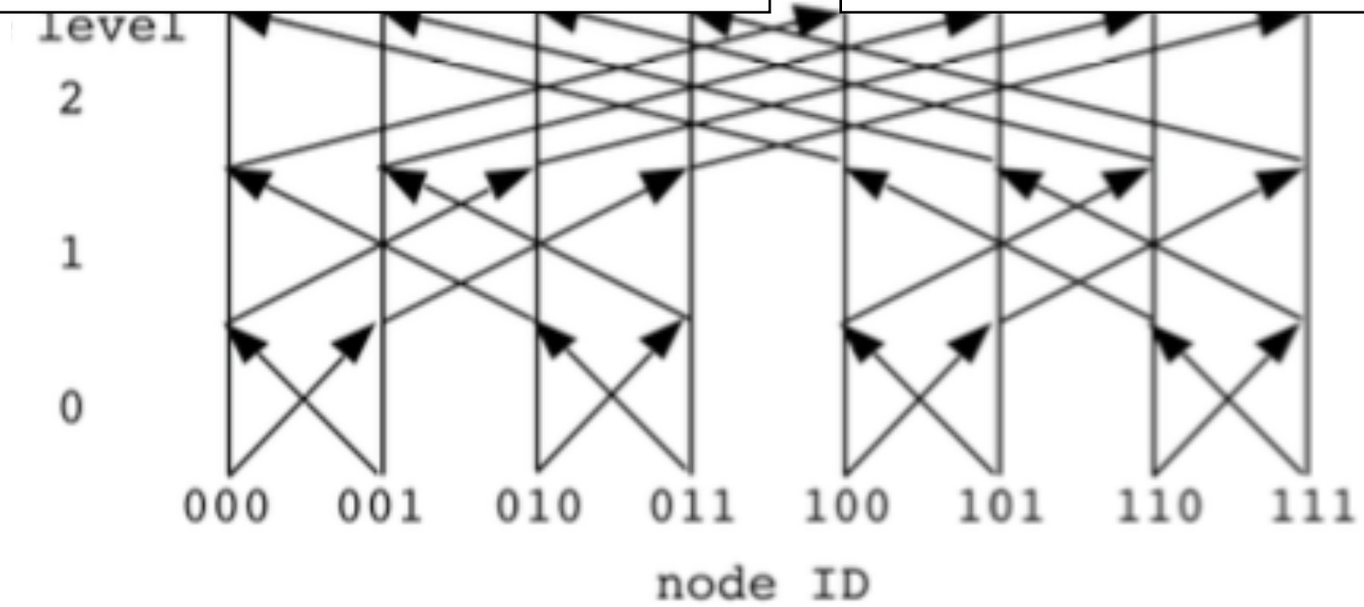


All-Gather in a Hypercube

- Butterfly → a different way to think about hypercubes

a-000 + a-001 + a-010 + a011

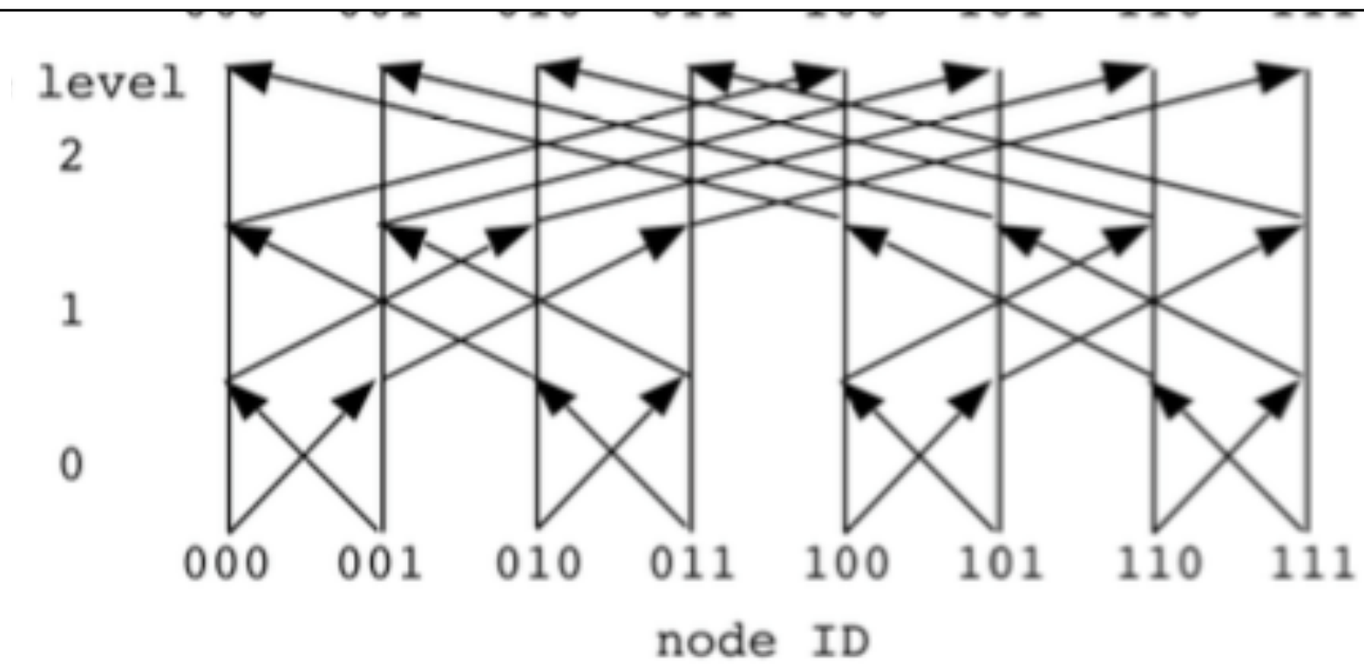
a100 + a101 + a110 + a111



All-Gather in a Hypercube

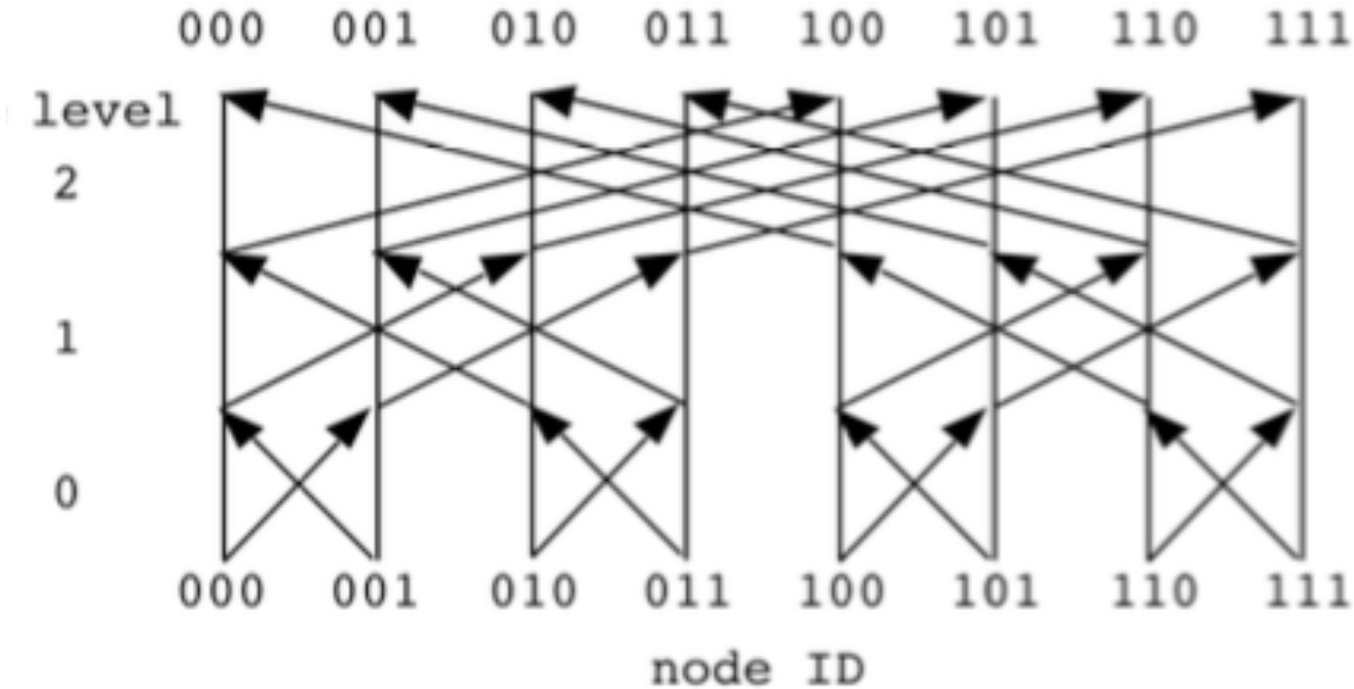
- Butterfly → a different way to think about hypercubes

a-000 + a-001 + a-010 + a-011 + a-100 + a-101 + a-110 + a-111



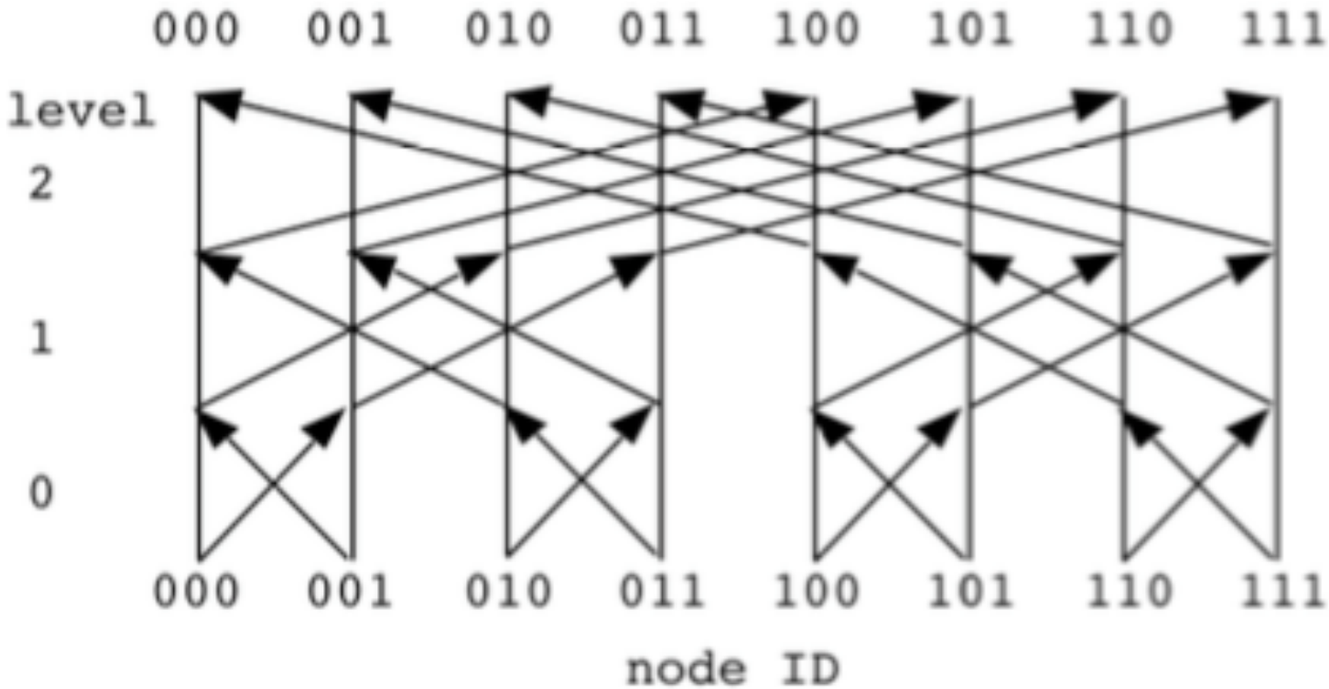
All-Gather in a Hypercube

- Butterfly → a different way to think about hypercubes



All-Gather in a Hypercube

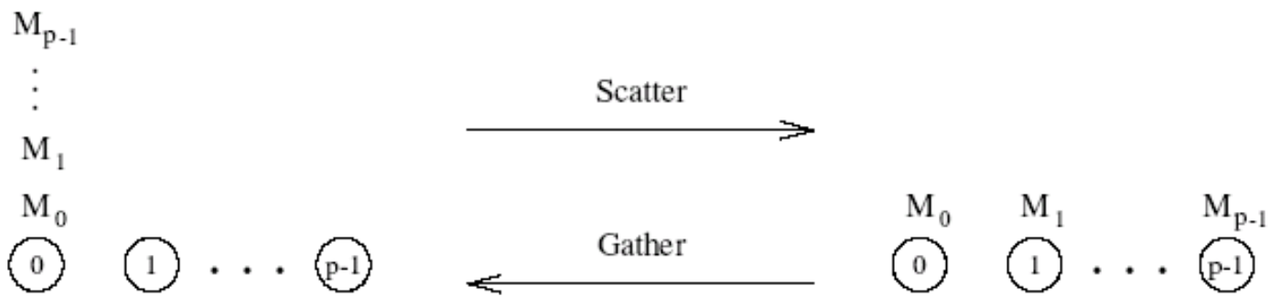
- Butterfly → a different way to think about hypercubes



At each level l, a process exchanges messages with a partner whose node ID differs only at the i^{th} bit position (Hamming Distance).

Scatter and Gather

- *Scatter* → a single node sends a unique message of size *m* to every other node (different nodes, different contents).
- In the *gather* operation, a single node collects a unique message from each node.

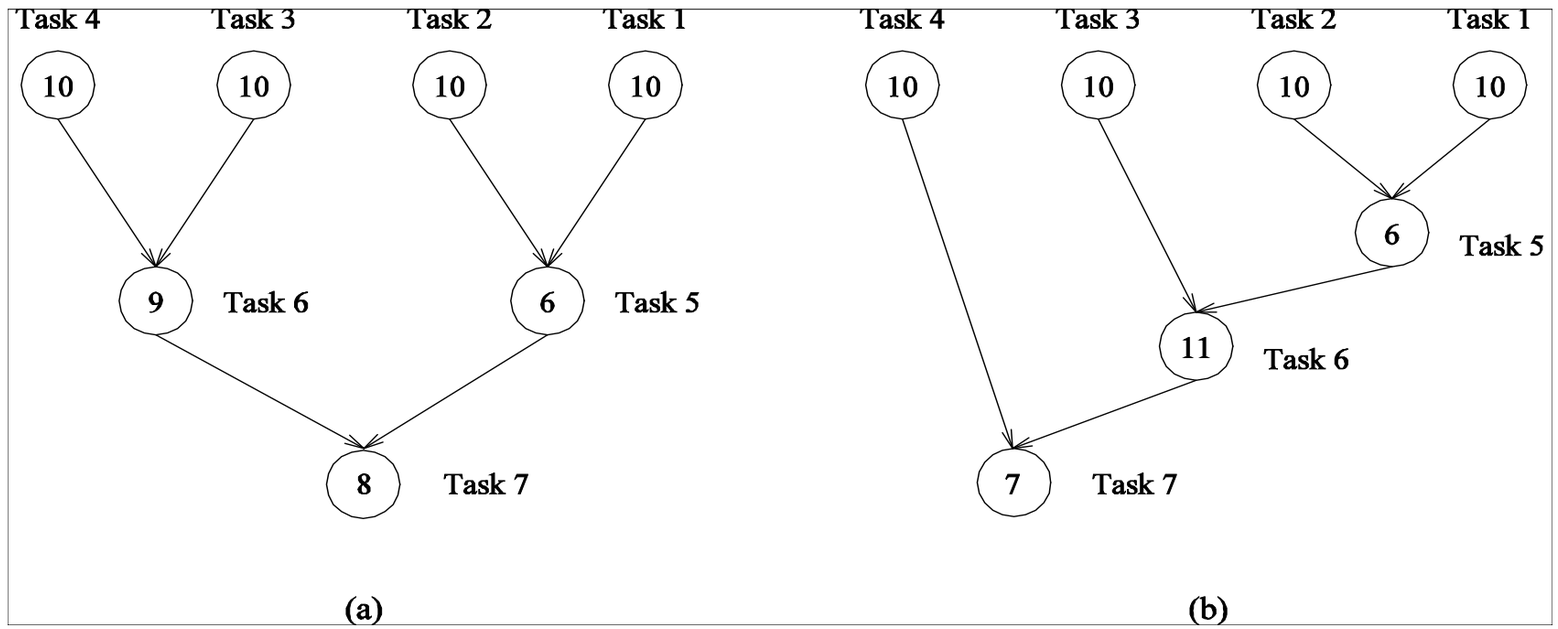


- While the scatter operation is fundamentally different from broadcast, the algorithmic structure is similar, except for differences in message sizes (messages get smaller in scatter and stay constant in broadcast).
- The gather operation is exactly the inverse of the scatter operation and can be executed as such.

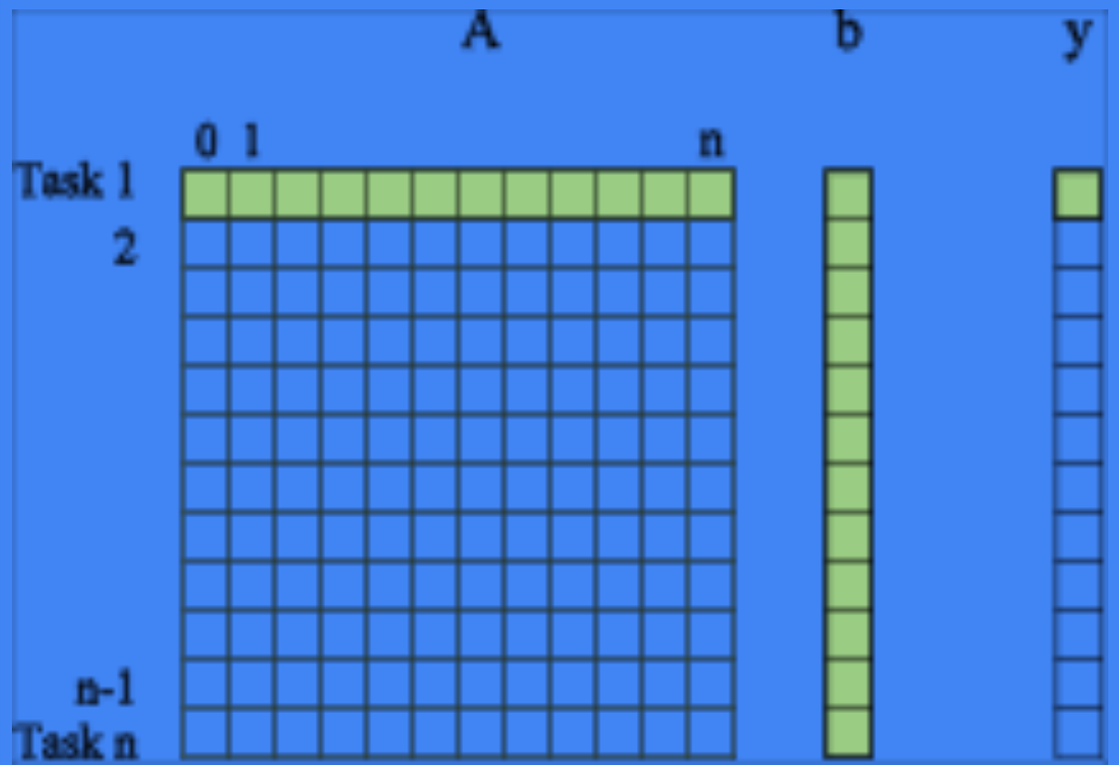
Designing a Parallel Algorithm

Terminology

- Decomposition: Break your problem into tasks that can be executed concurrently
- Task Dependency Graph: directed graph with nodes corresponding to tasks and edges indicating that the result of one task is required for processing the next.



Task Decomposition
Y[N] = A[N][N] . B[N]



Q. What messaging strategy will you use to solve this problem?

A →

B →

Y ←

Consider the execution of the query:

MODEL = ``CIVIC'' AND YEAR = 2001 AND (COLOR = ``GREEN'' OR COLOR = ``WHITE)

on the following database:

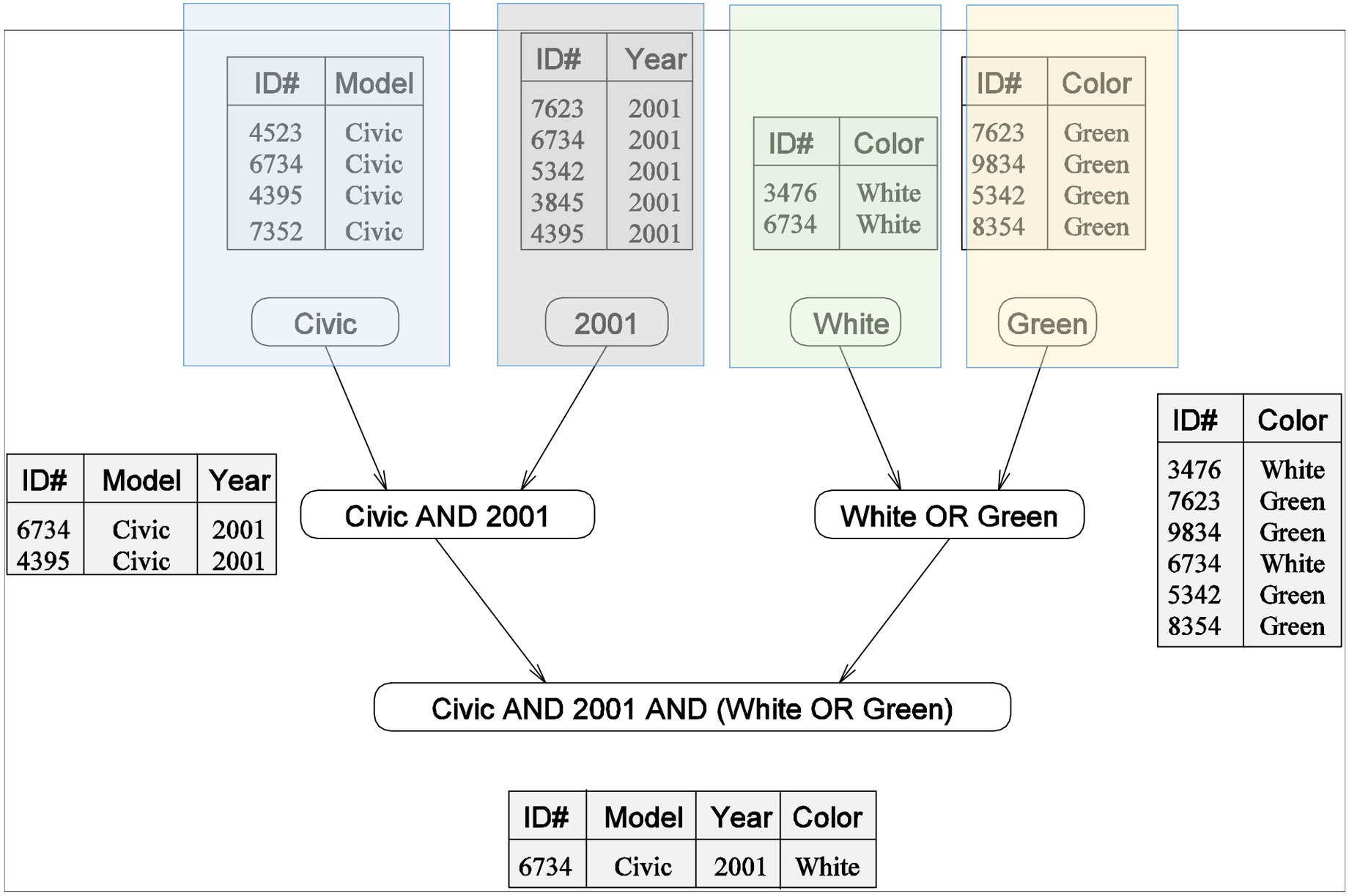
ID#	Model	Year	Color	Dealer	Price
4523	Civic	2002	Blue	MN	$18,000
3476	Corolla	1999	White	IL	$15,000
7623	Camry	2001	Green	NY	$21,000
9834	Prius	2001	Green	CA	$18,000
6734	Civic	2001	White	OR	$17,000
5342	Altima	2001	Green	FL	$19,000
3845	Maxima	2001	Blue	NY	$22,000
8354	Accord	2000	Green	VT	$18,000
4395	Civic	2001	Red	CA	$17,000
7352	Civic	2002	Red	WA	$18,000

Example from Grama et. al.

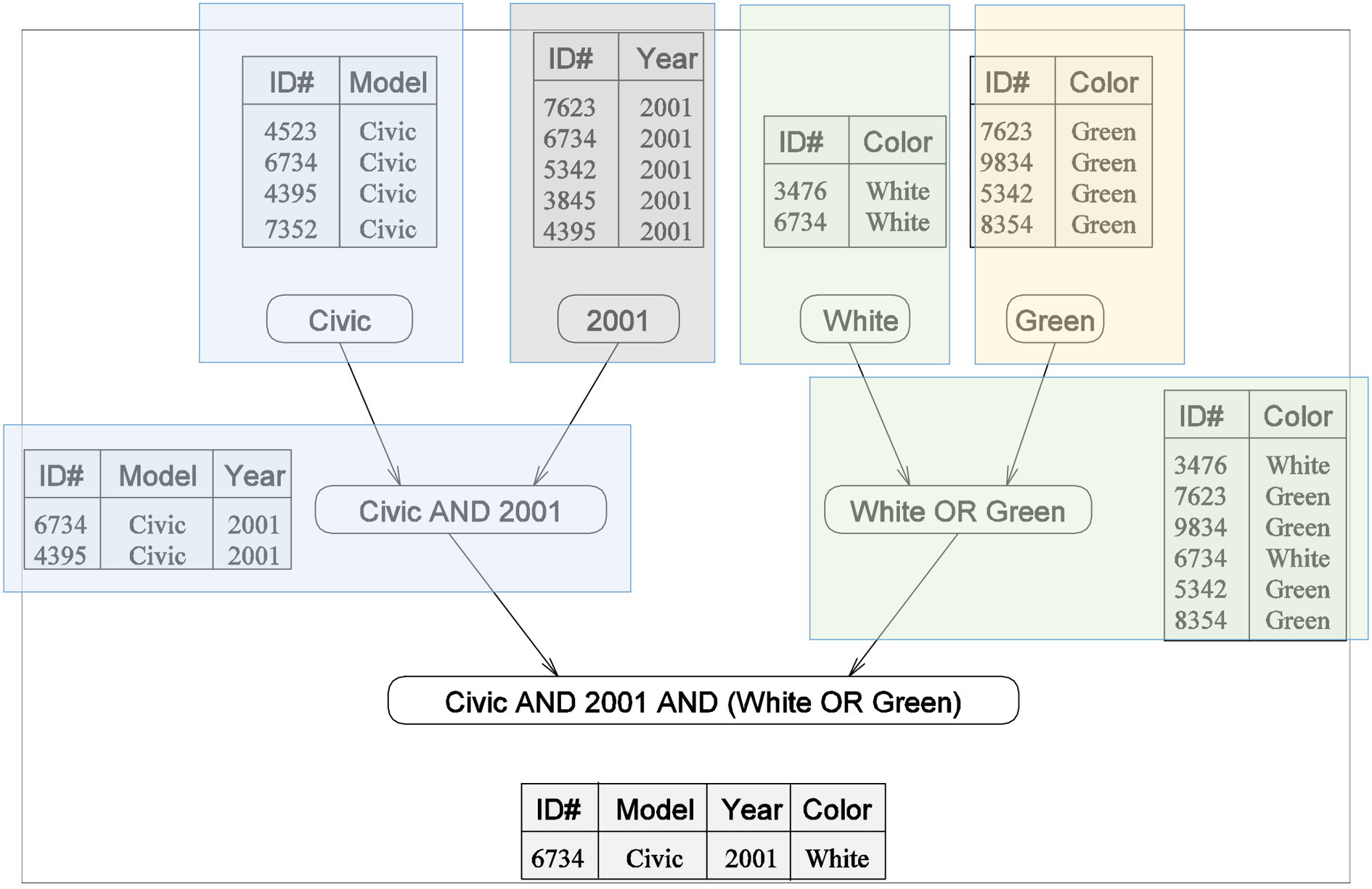
ID#	Model	Year	Color	Dealer	Price
4523	Civic	2002	Blue	MN	$18,000
3476	Corolla	1999	White	IL	$15,000
7623	Camry	2001	Green	NY	$21,000
9834	Prius	2001	Green	CA	$18,000
6734	Civic	2001	White	OR	$17,000
5342	Altima	2001	Green	FL	$19,000
3845	Maxima	2001	Blue	NY	$22,000
8354	Accord	2000	Green	VT	$18,000
4395	Civic	2001	Red	CA	$17,000
7352	Civic	2002	Red	WA	$18,000

Example 2: Database Query

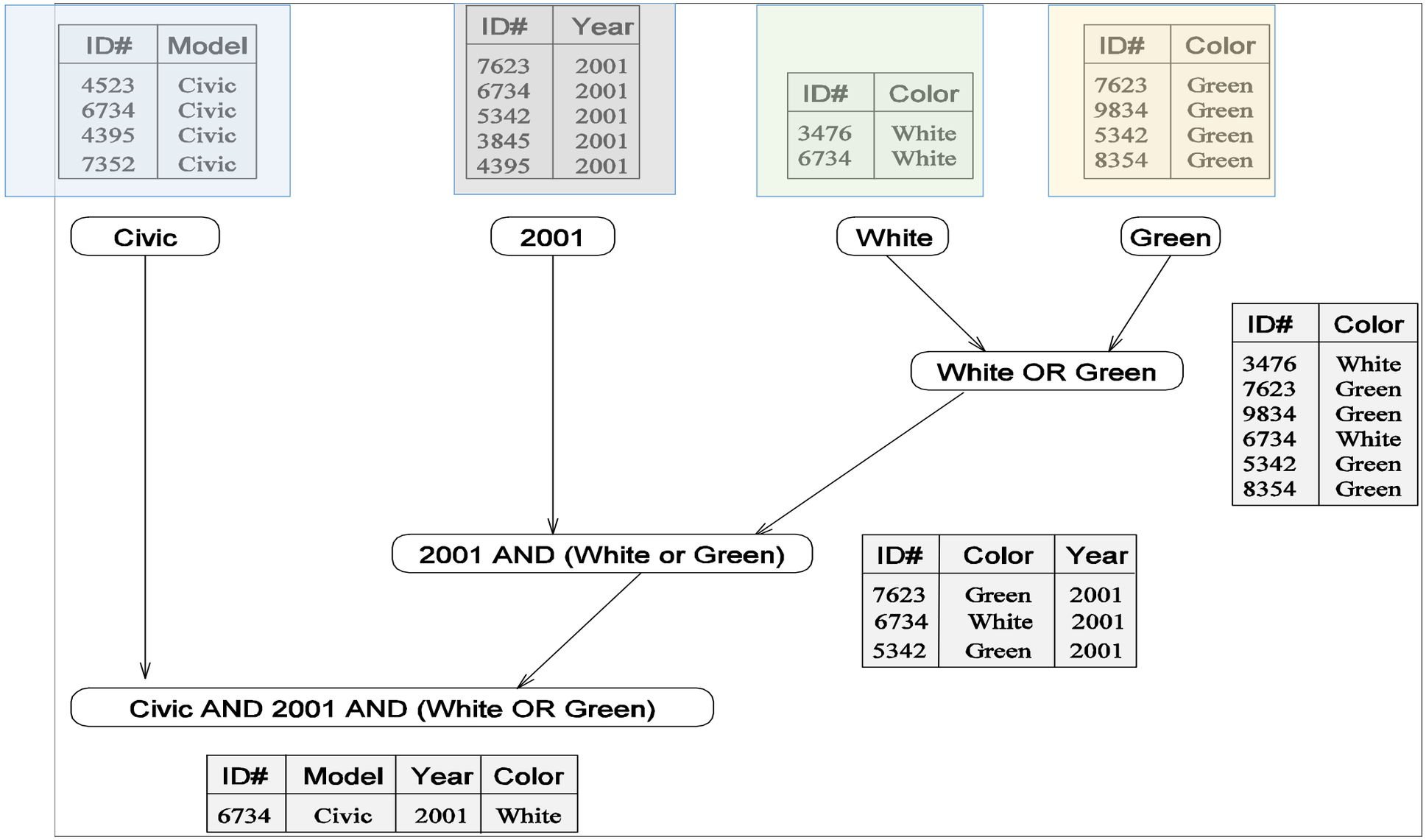
Example from Grama et. al.



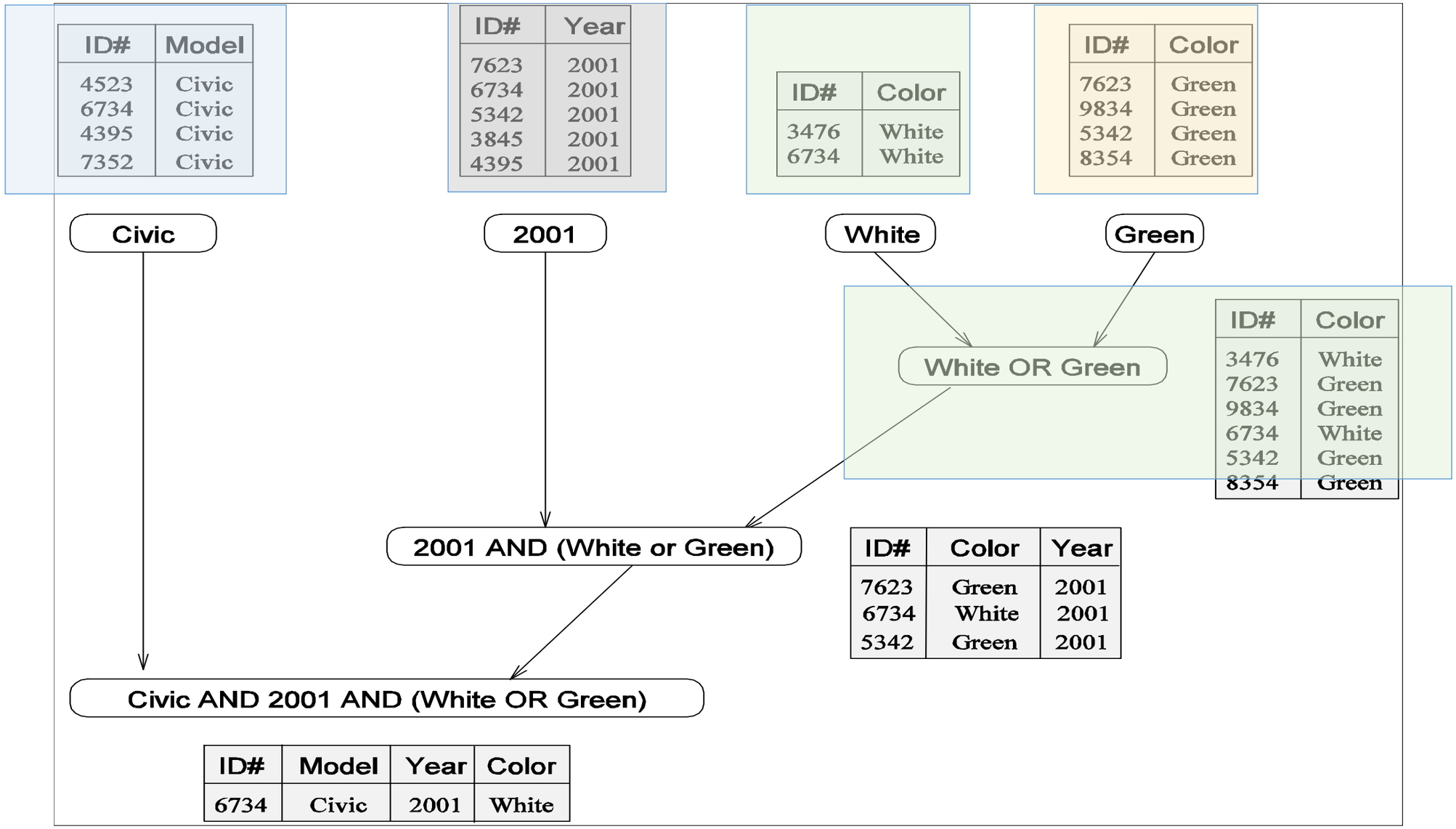
Example 2: Database Query



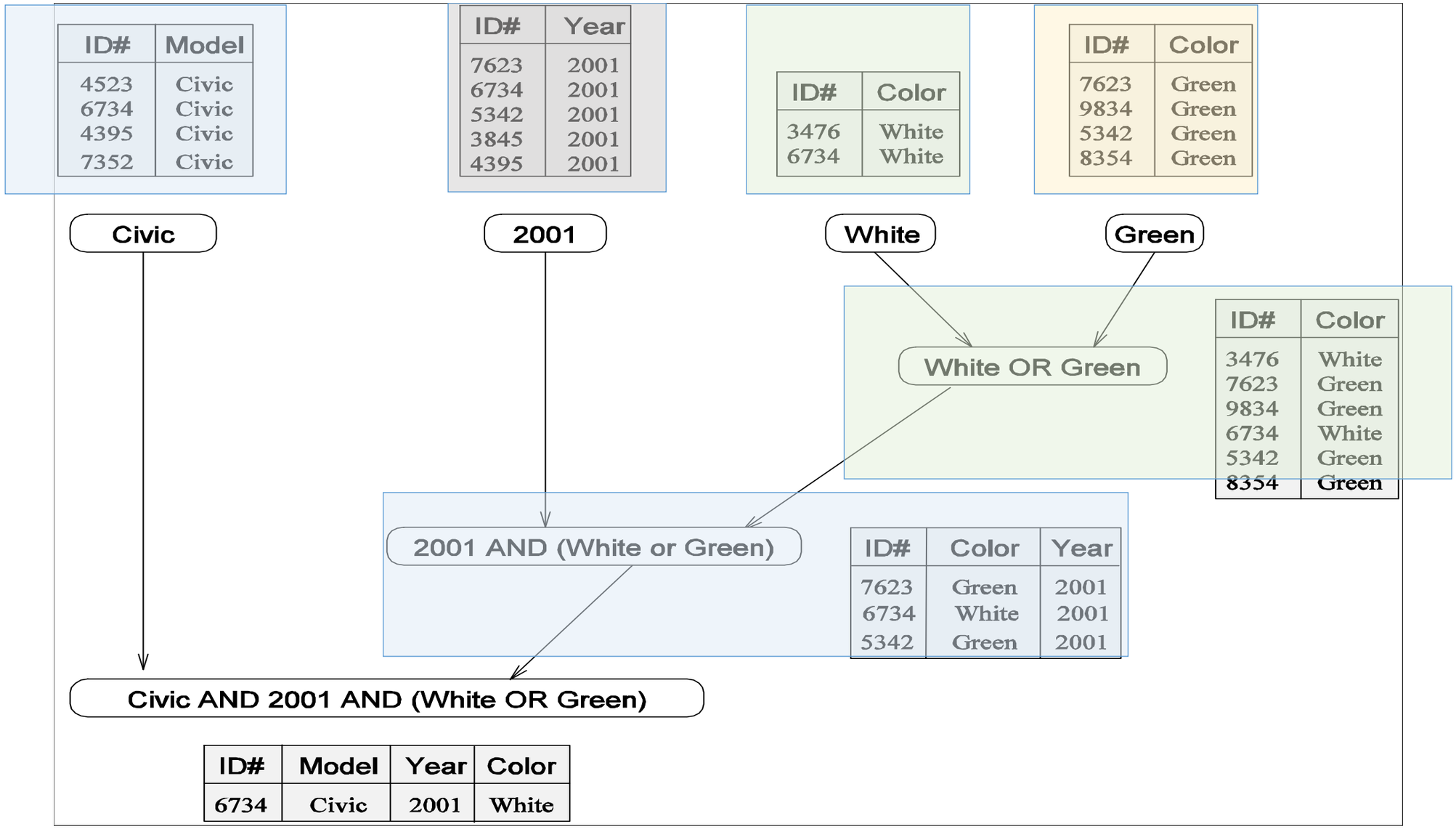
Example 2: Database Query



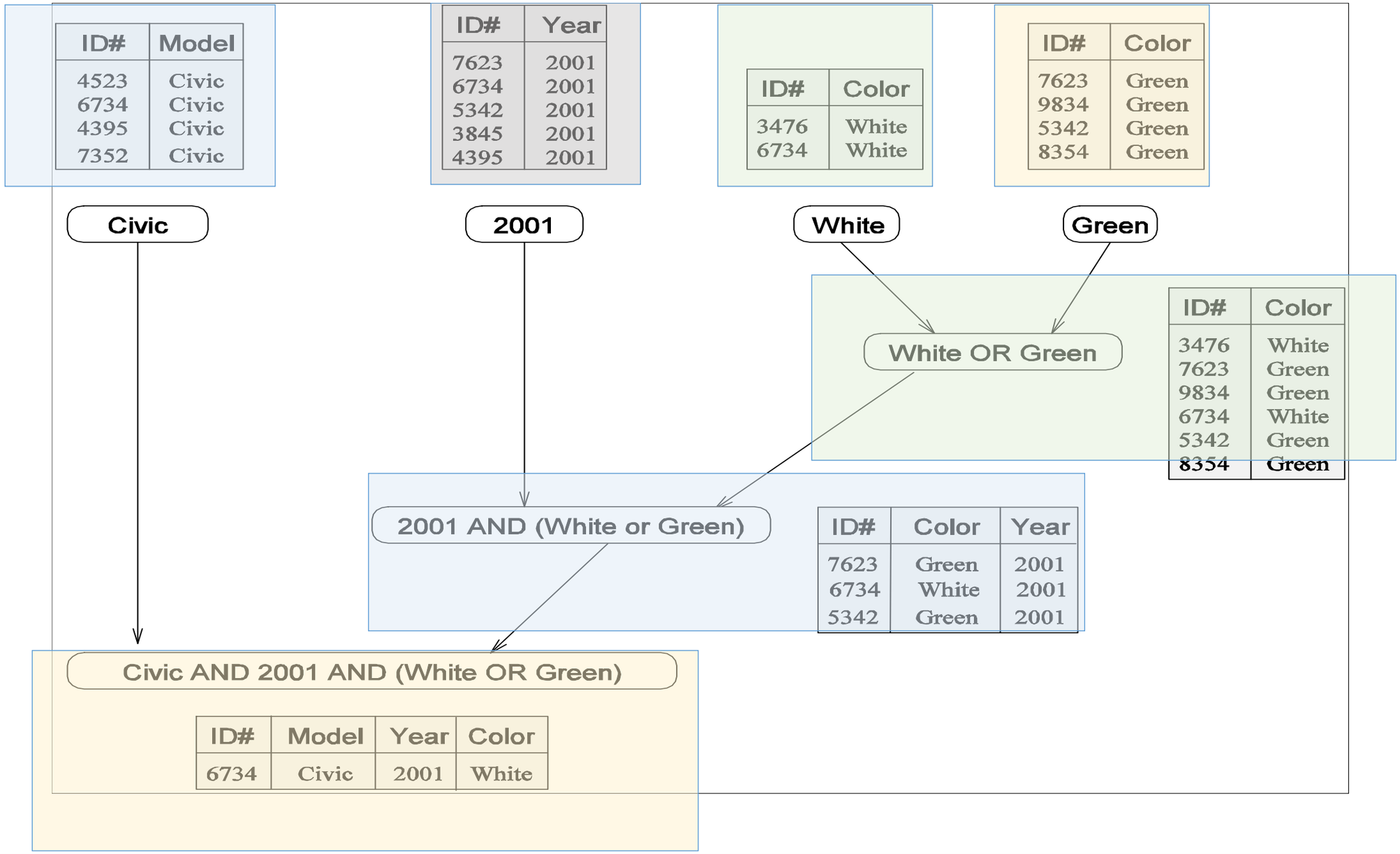
Example 2: Database Query (alternate decomposition strategy)



Example 2: Database Query (alternate decomposition strategy)



Example 2: Database Query (alternate decomposition strategy)

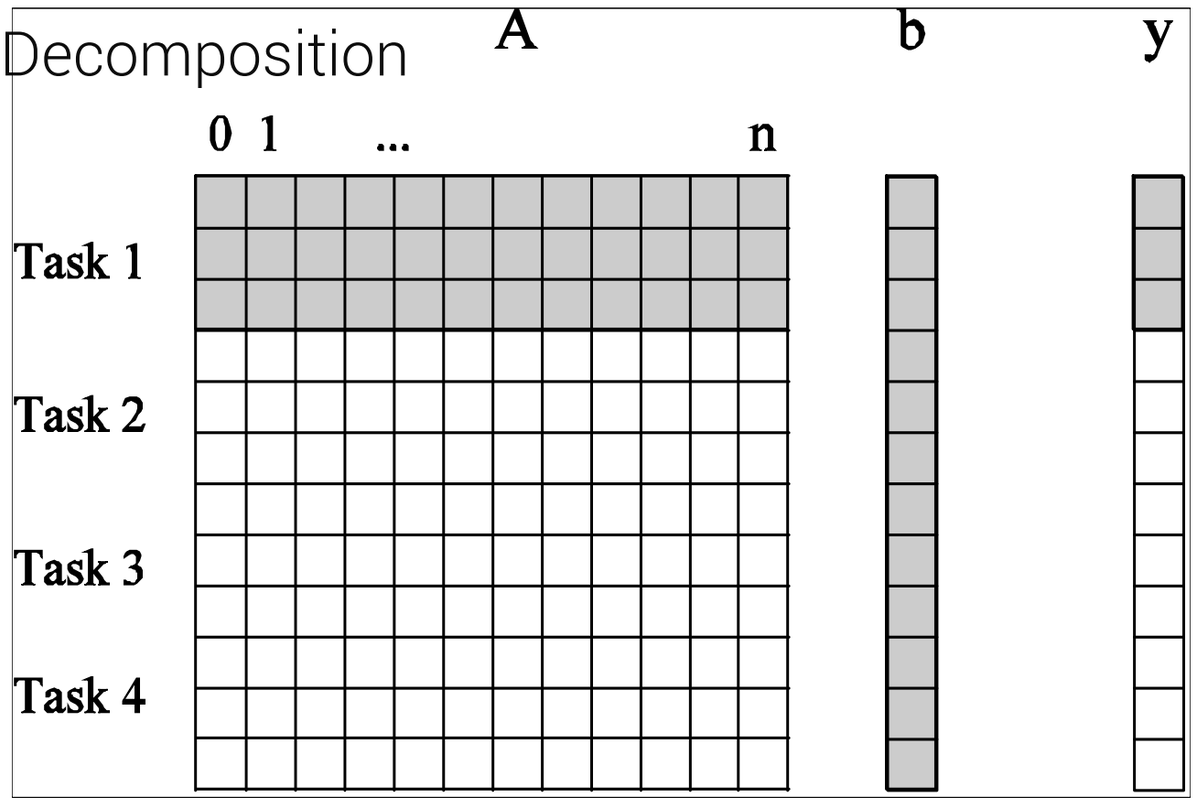


Example 2: Database Query (alternate decomposition strategy)

Granularity

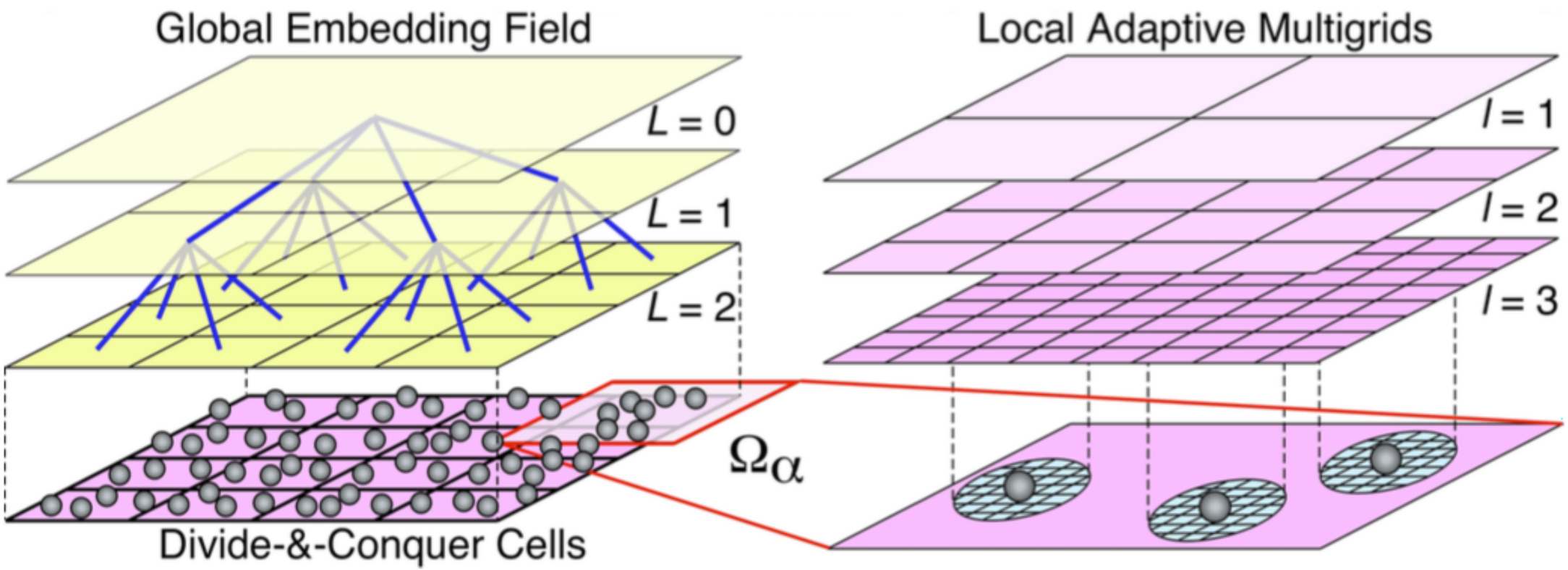
- Num of tasks into which a problem can be decomposed → Granularity

- Fine Grained vs. Coarse Grained Decomposition



Decomposition Techniques

Divide and Conquer



Slides adapted from Nakano, et. al.

D&C illsutrated in QuickSort


```
quicksort(int list[],int left,int right)
{       int j;
        if (left < right) {
                j = partition(list,left,right);
                quicksort(list,left,j-1);
                quicksort(list,j+1,right);
} }
```

- partition: Given list [left : right],
  it first chooses the left-most element as a pivot;
  on return the pivot element is placed at the j-th position, &:
  i) a[left],...,a[j-1] are less than or equal to a[j];
  ii) a[j+1],...,a[right] are greater than or equal to a[j].

Slides adapted from Nakano, et. al.

Quicksort

0	1	2	3	4	5	6	7	8	9
[5	7	2	9	6	8	3	4	1	0]
[3	0	2	1	4]	5	[8	6	9	7]
[1	0	2]	3	[4]	5	[7	6]	8	[9]
[0]	1	[2]	3	[4]	5	[6]	7	[] 8	[9]

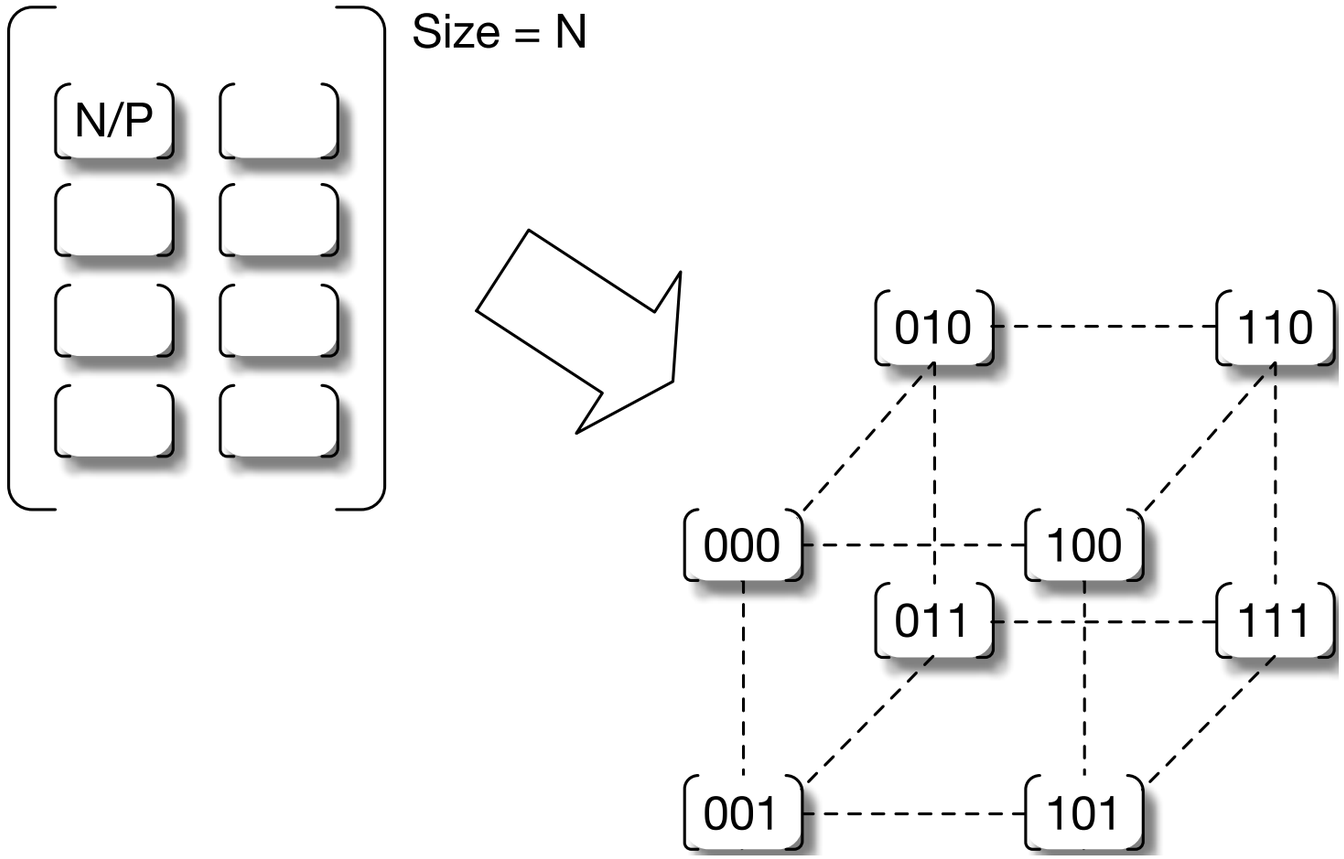
Slides adapted from Nakano, et. al.

0. Setup

- N elements
- P = 2^d nodes

Step 1 → Distribute data amongst all nodes

Each node → N/P elements

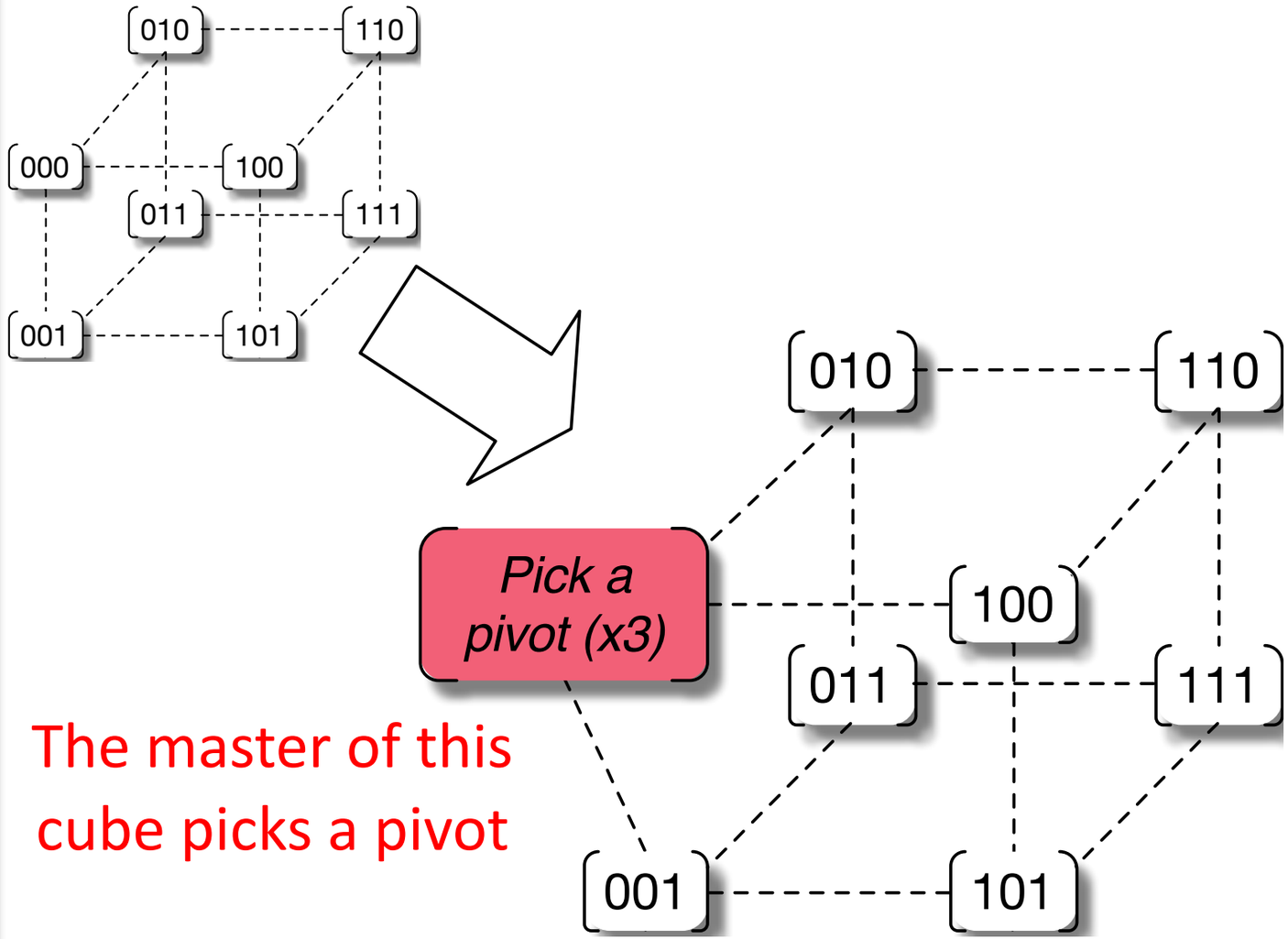


Master-of-the-cube

Master → Responsible for starting the sequence of events that happen in the hypercube.

As you split the hypercube (i.e. process lower dimension hypercubes, each of those dimensions will have their own master)

Dimension	Master of Cube
3D	000
2D	x00 (i.e. 000 & 100 are the masters of their respective subcubes)
1D	xx0



The master of this cube picks a pivot

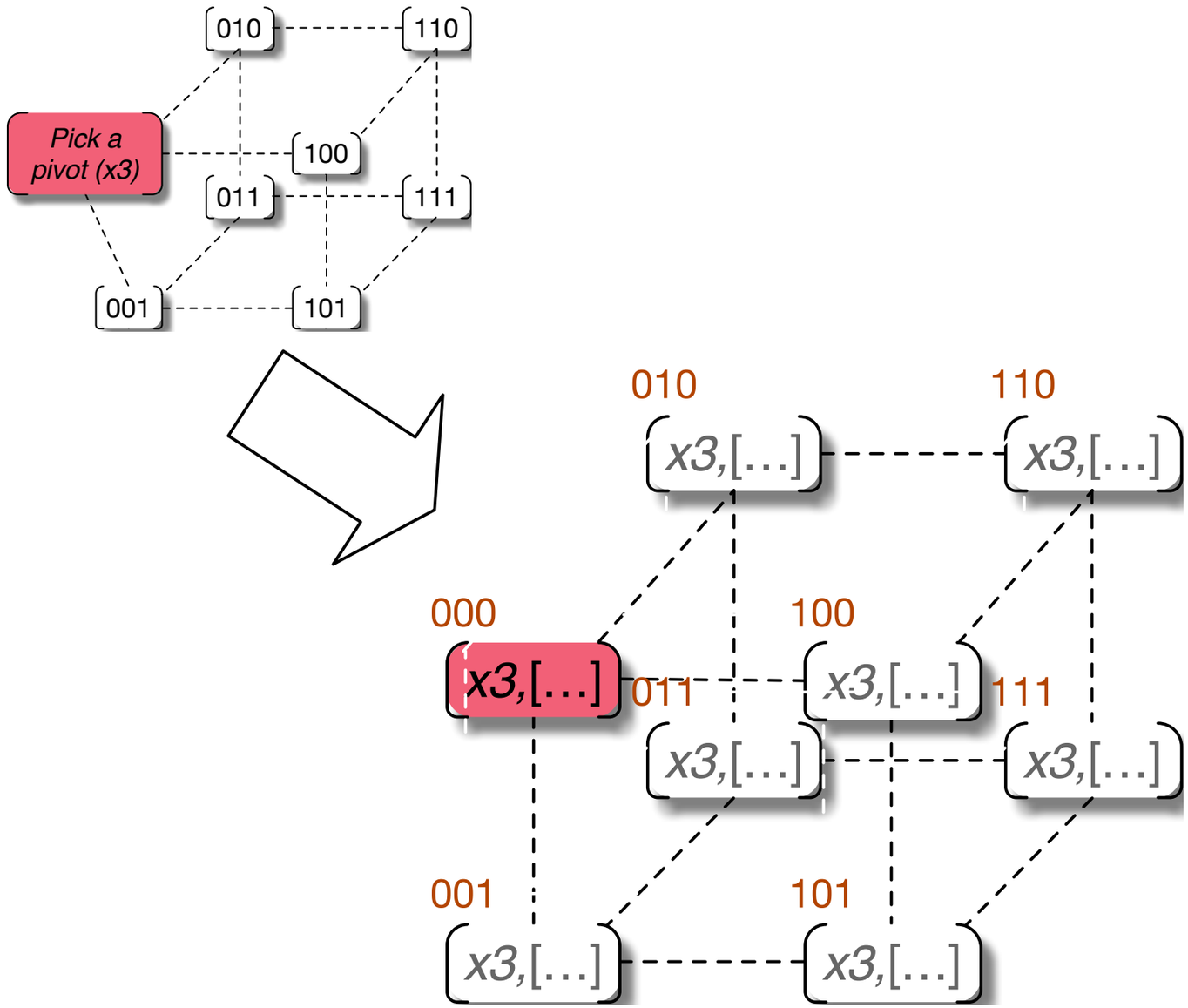
Picking a Pivot

Bad choice of pivot at early stages degrades the performance significantly (no recovery from it). Use the average value elements in the *master-of-the-cube* as a pivot.

1. Broadcast Pivot

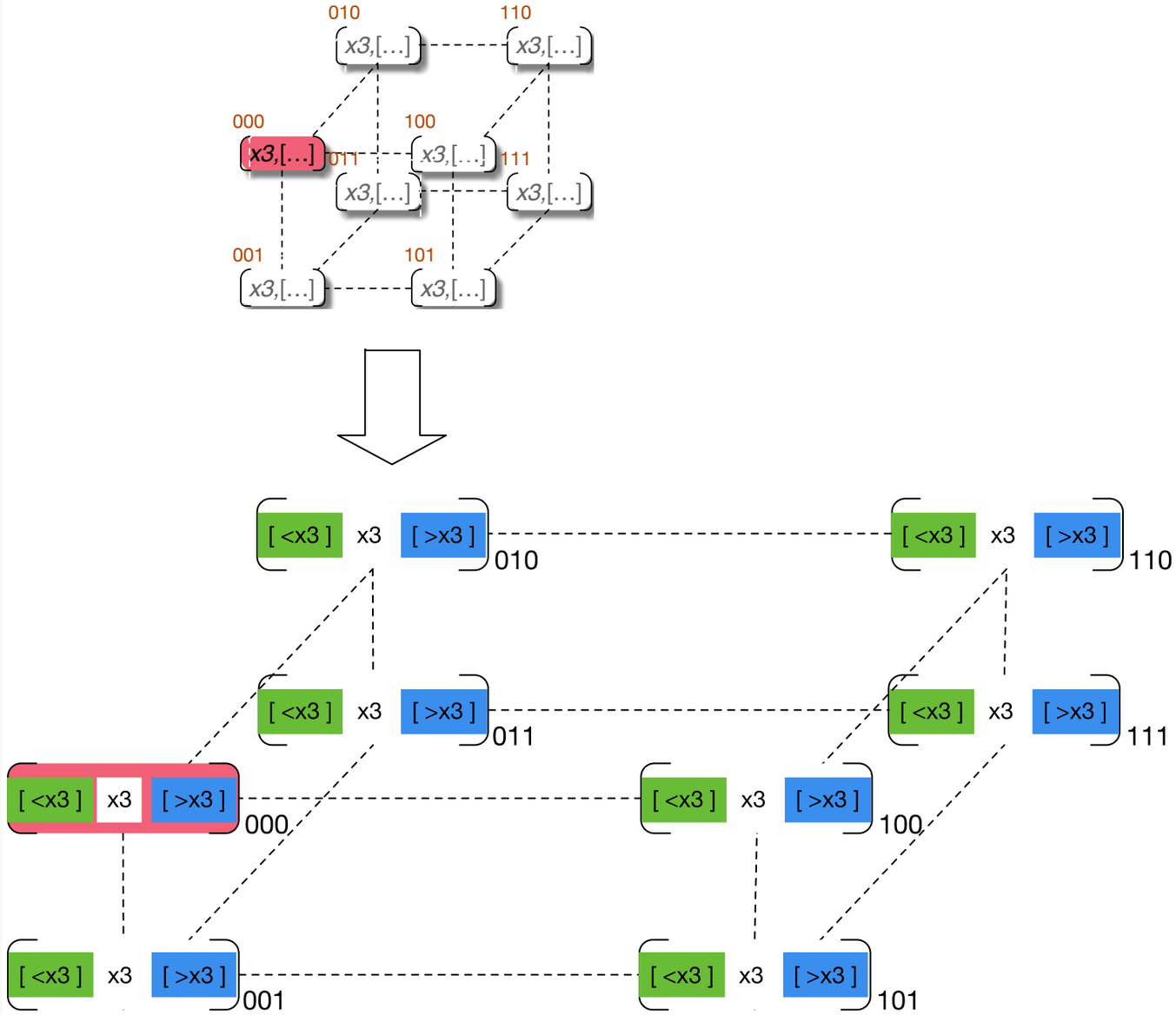
1. Master-of-the-cube broadcasts pivot to all other nodes in the cube

2. Now all nodes have the pivot value



2. In Each node...

1. Split the elements so that they are either greater than or less than the pivot (x3)

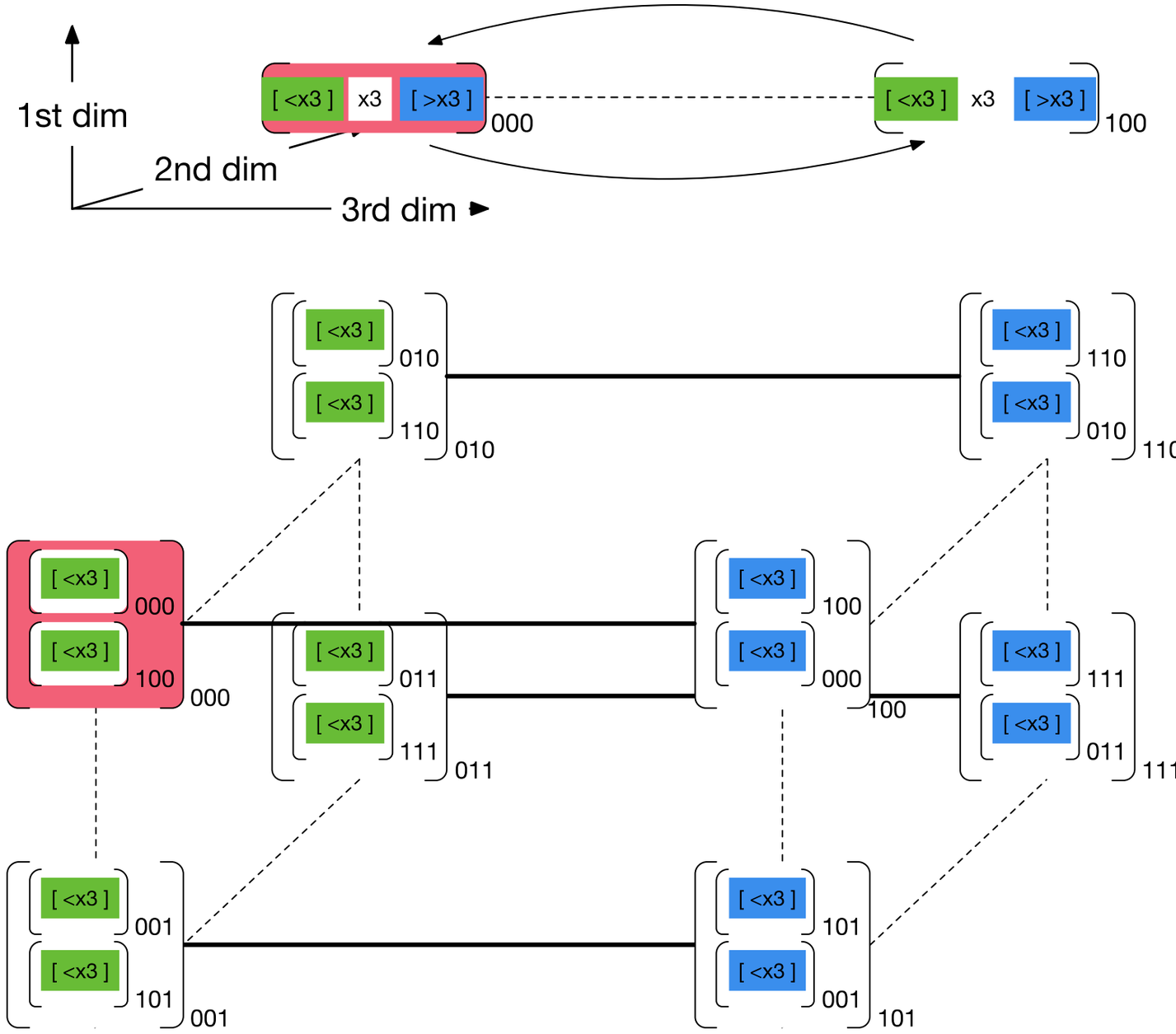


3. On the d^{th} dimension exchange data

In step 1, d = 3

So we will exchange data on the 3rd dimension. i.e., we will exchange data between 0xx and 1xx

After the exchange, discard the pivot



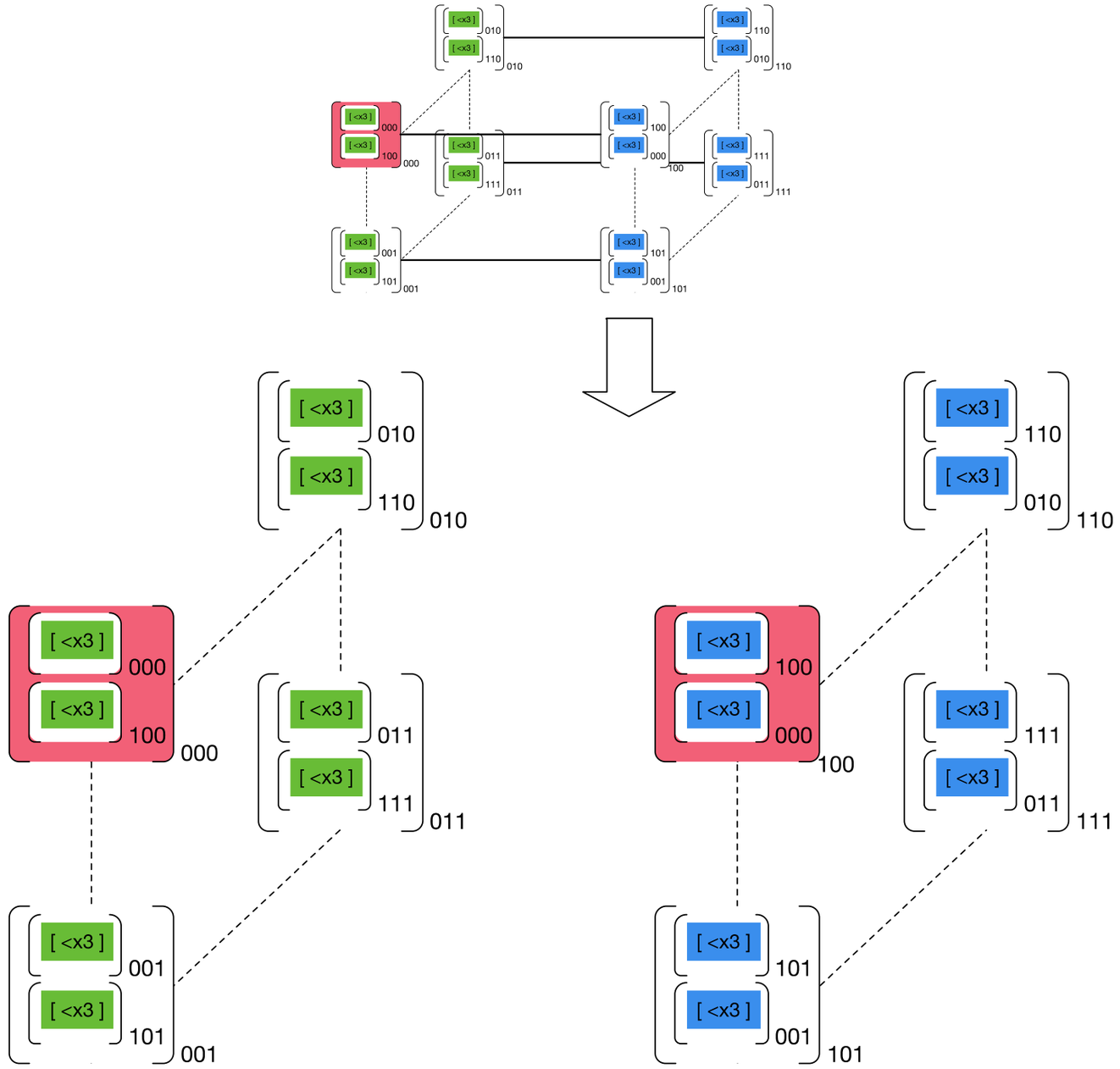
4. Let's split in to 2 d-1 cubes

In step 1, d = 3

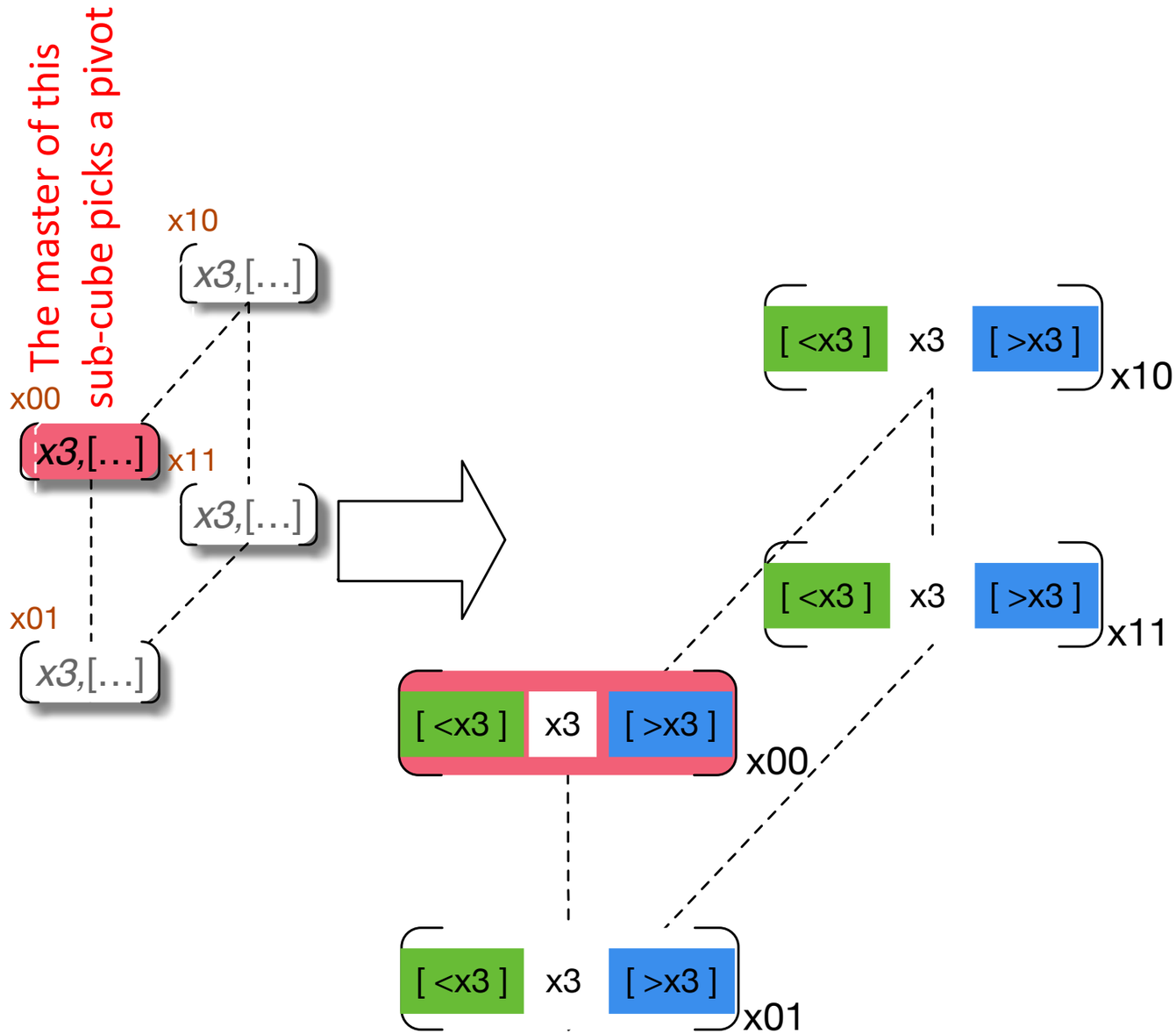
So we will exchange data on the 3rd dimension. i.e., we will exchange data between 0xx and 1xx

After the exchange, discard the pivot

Select new masters of the 2 sub-cubes



5. Repeat until you reach 1D cubes



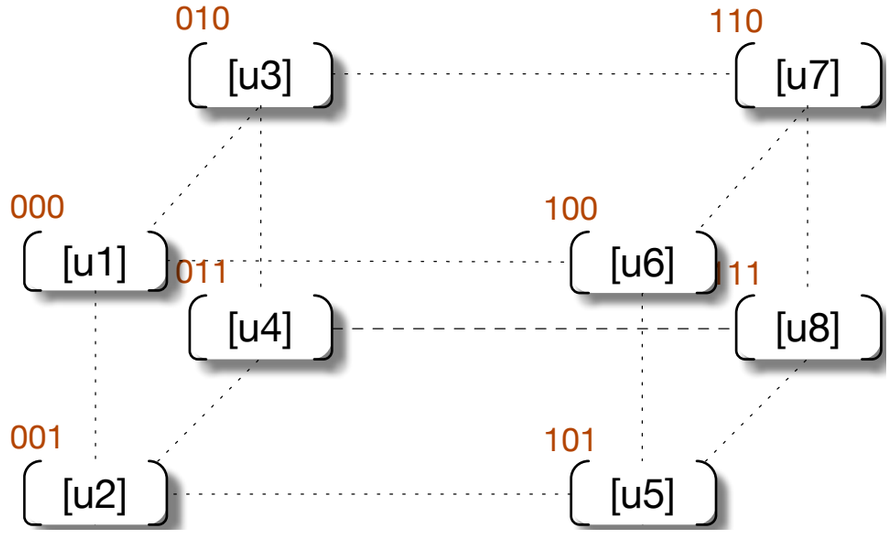
Step 6

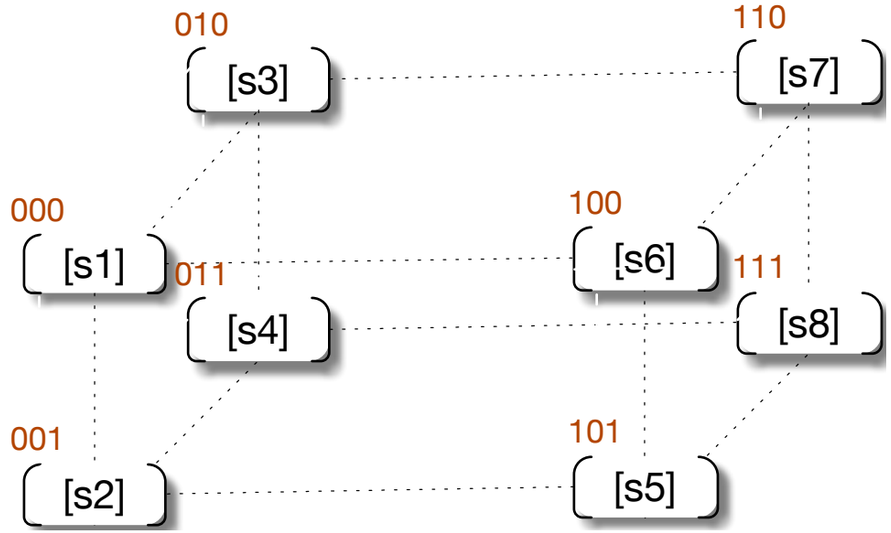
Once you reach 1D sub-cubes, you have your elements chunked into 2^d sorted sets

[u1] < [u2] < [u3] < ... < [u8]

Apply quicksort on each of the chunks

[s1] < [s2] < [s3] ... < [s8]





All to One Reduce

→ 000 will get sorted numbers