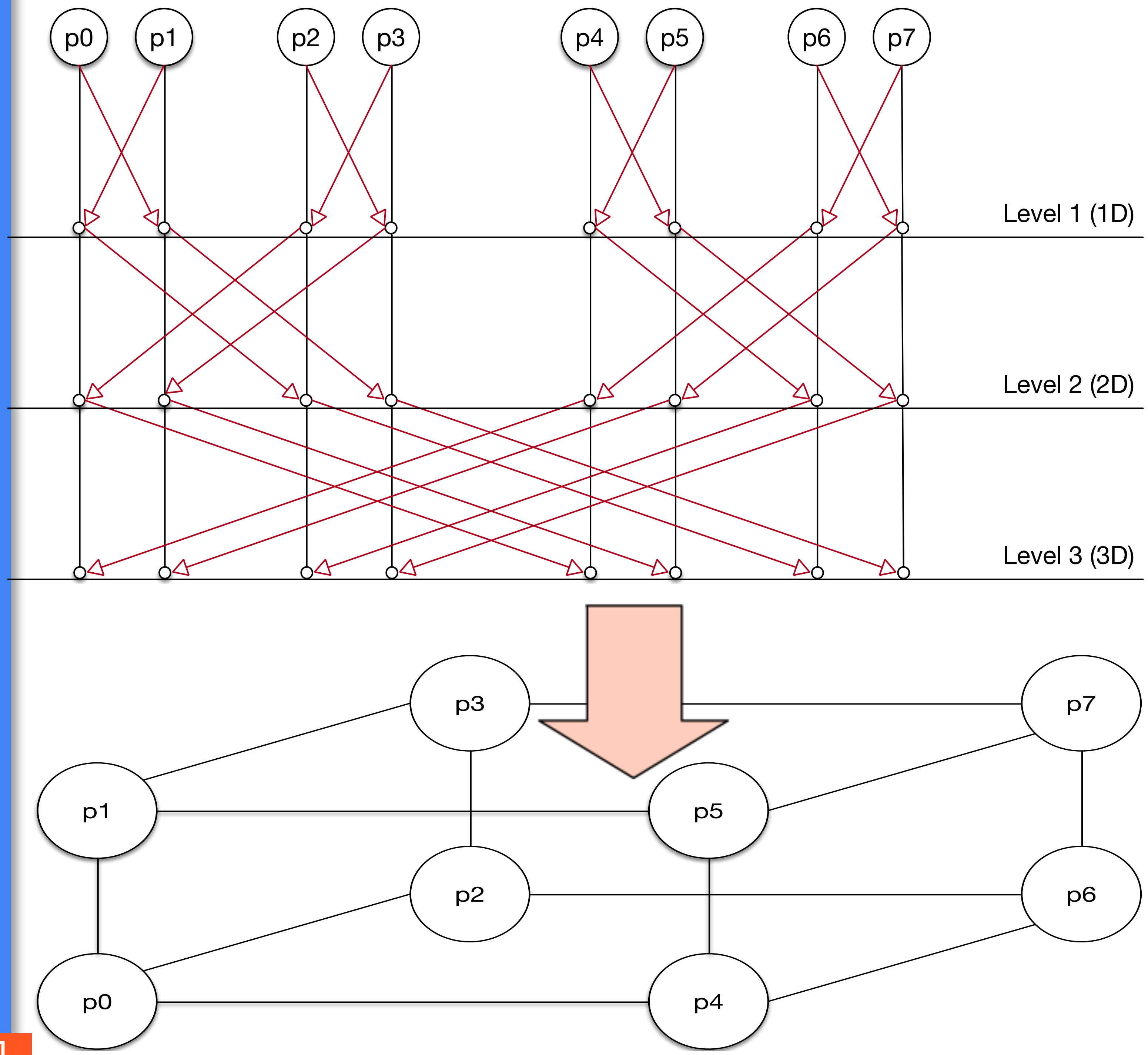
Hypercubes (or Butterfly)

More than networking topology, it is a data layout/ partitioning strategy that let's you navigate your space in a predictive (and thus programmatic) fashion



Navigating in this space

Say you want to move from:

P0 → P1 (in blue)

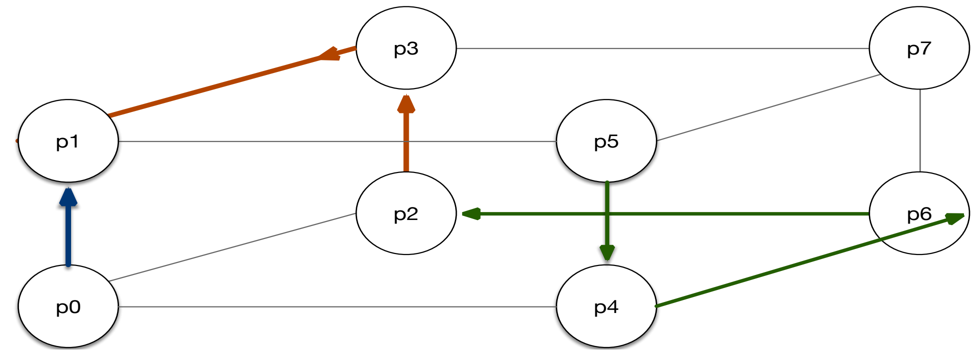
000 → 001

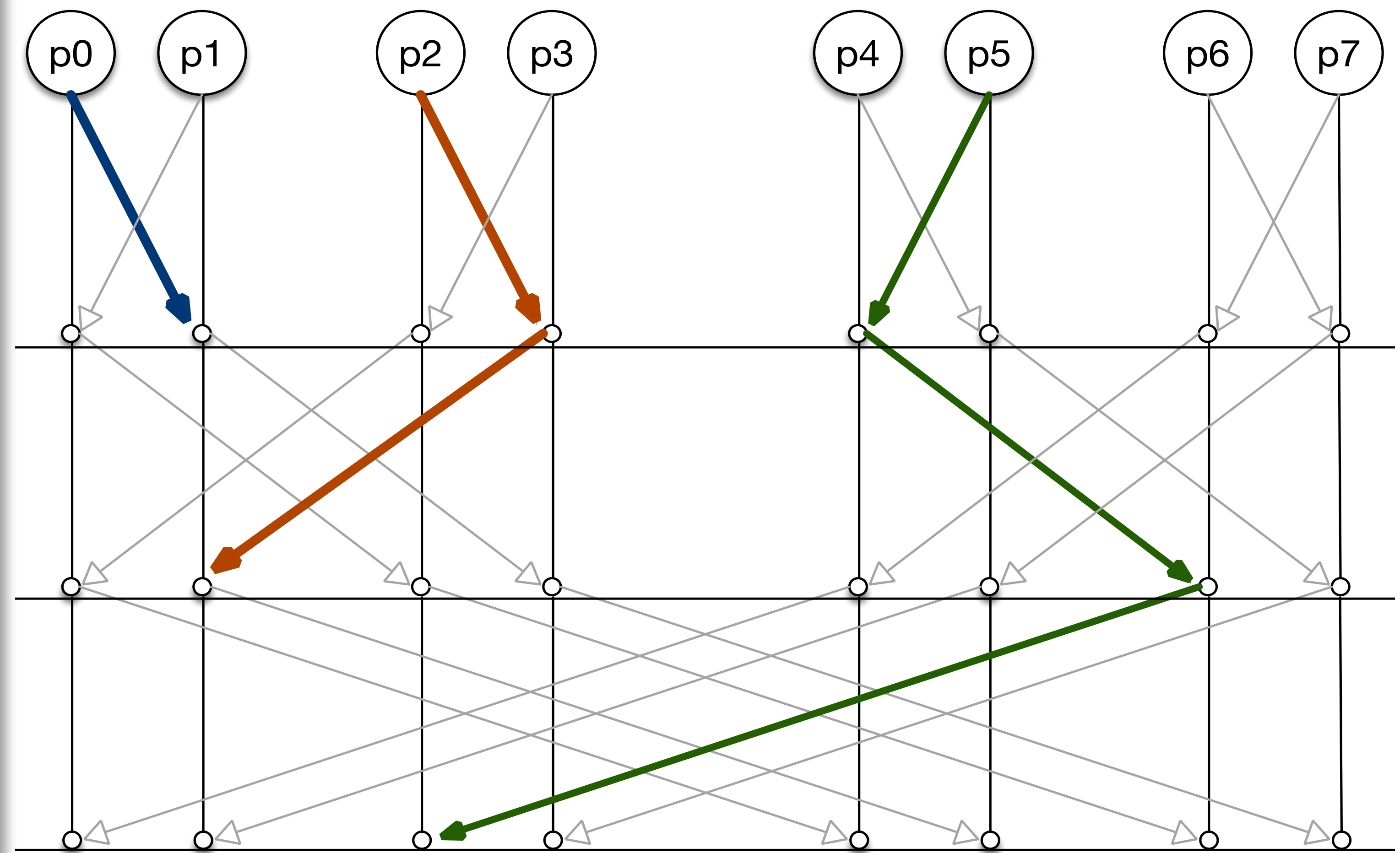
P2 → P3 (in orange)

010 → 011 → 001

P5 → P2 (in green)

101 → 100 → 110 → 010



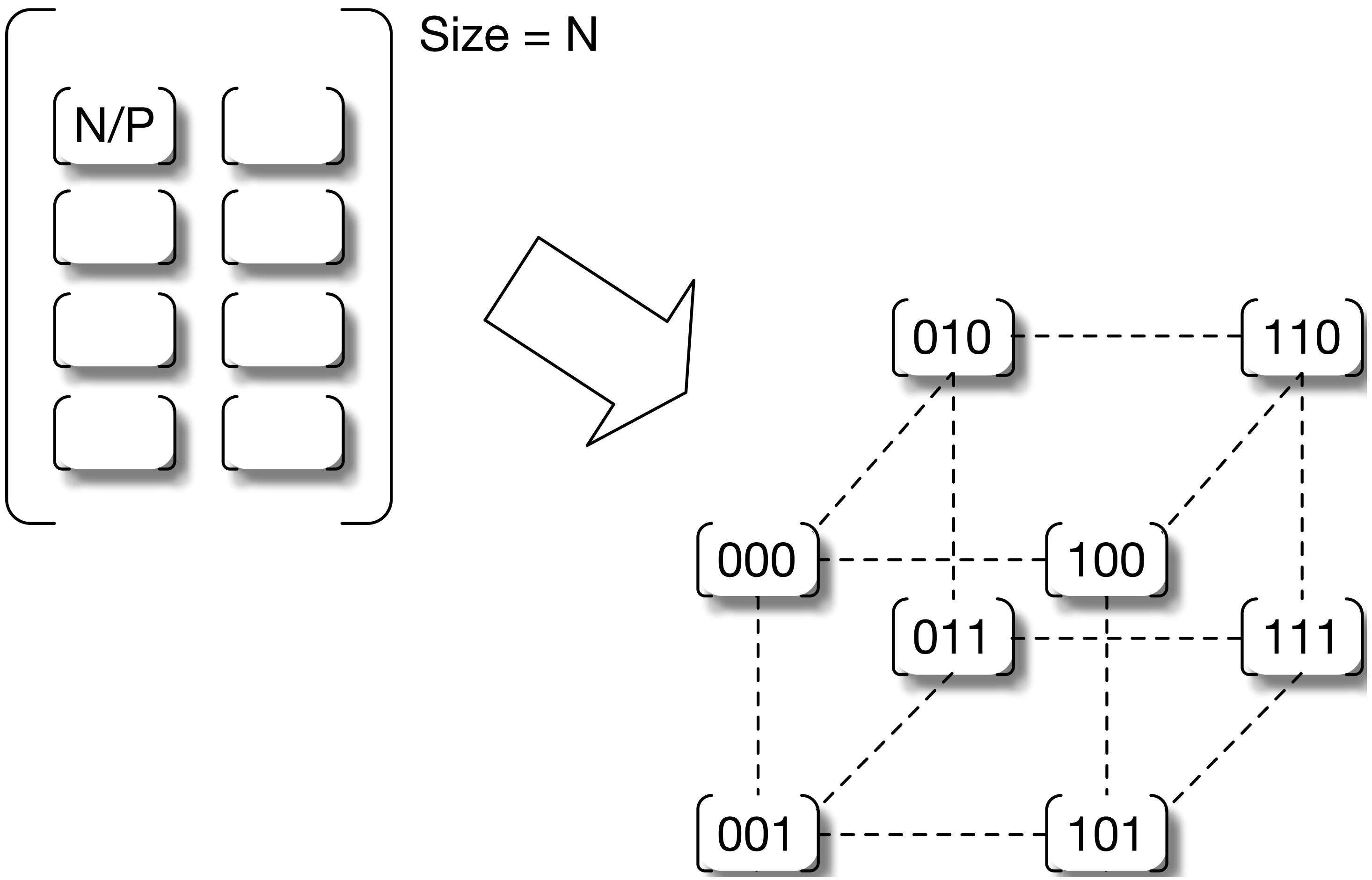


0. Setup

- N elements
- $P = 2^d$ nodes

Step 1 → Distribute data amongst all nodes

Each node → N/P elements

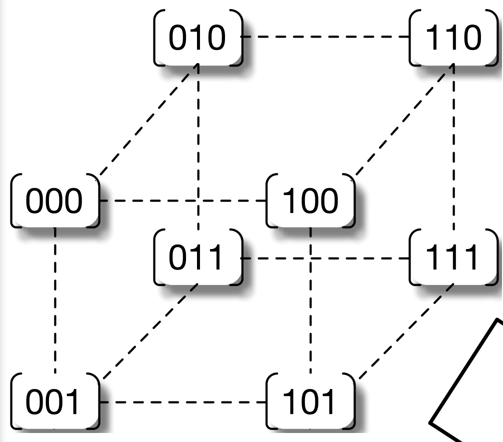


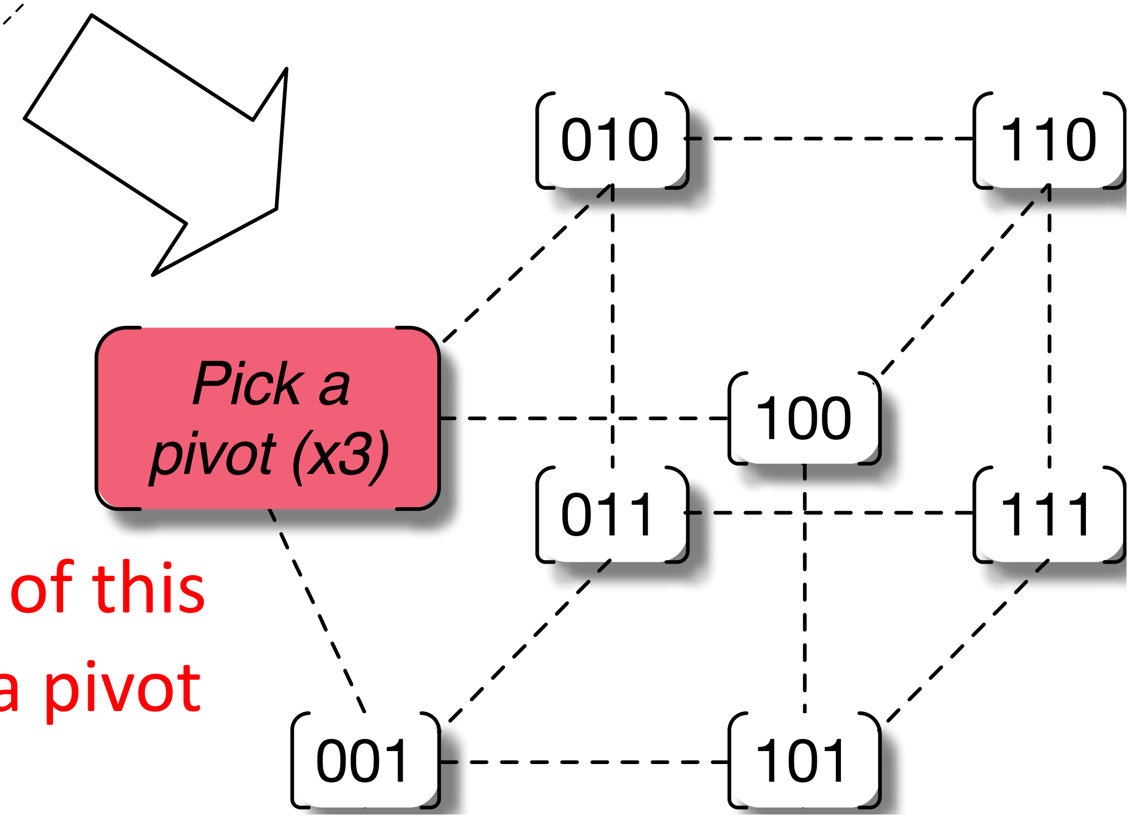
Master-of-the-cube

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





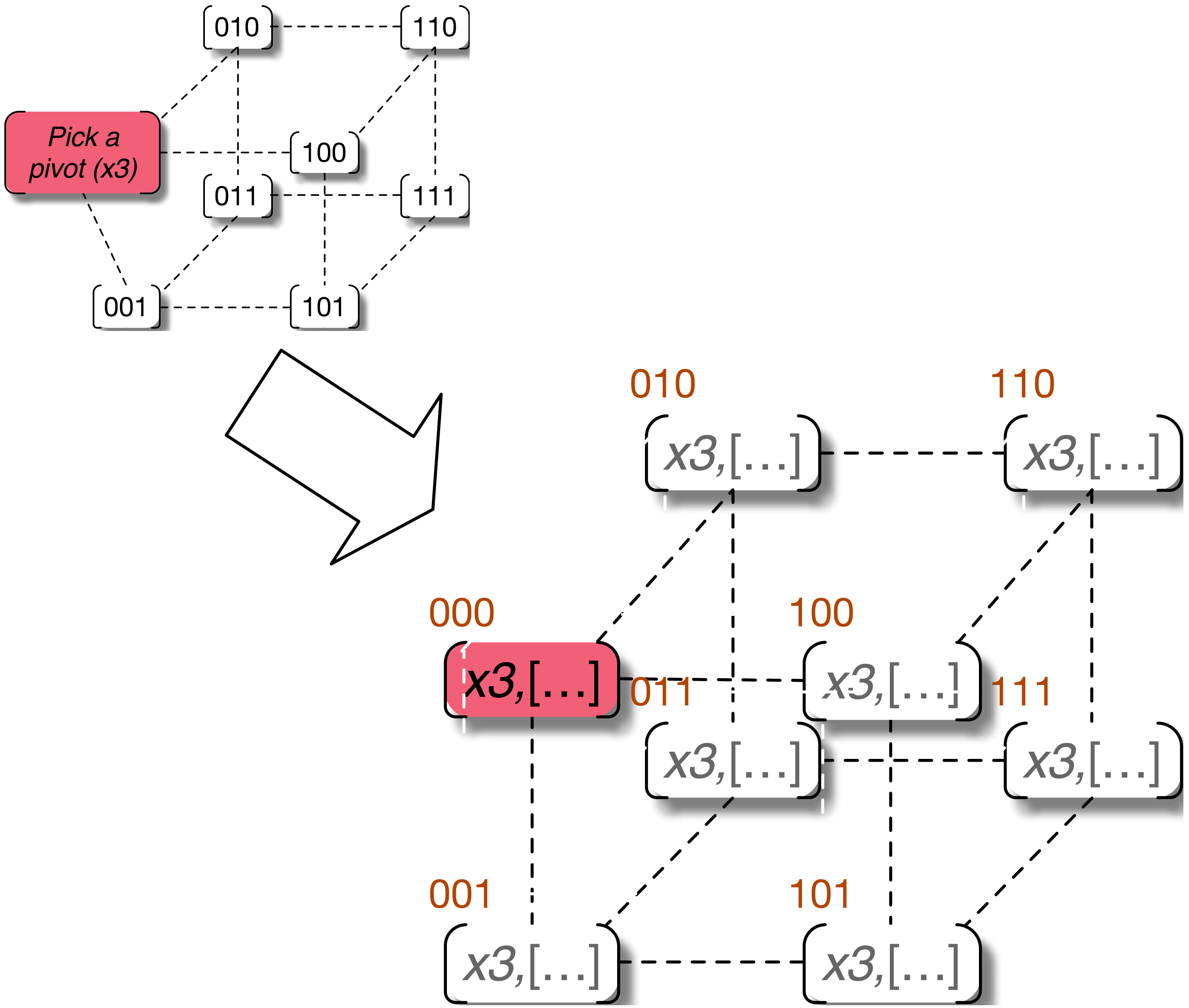
The master of this cube picks a pivot

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.

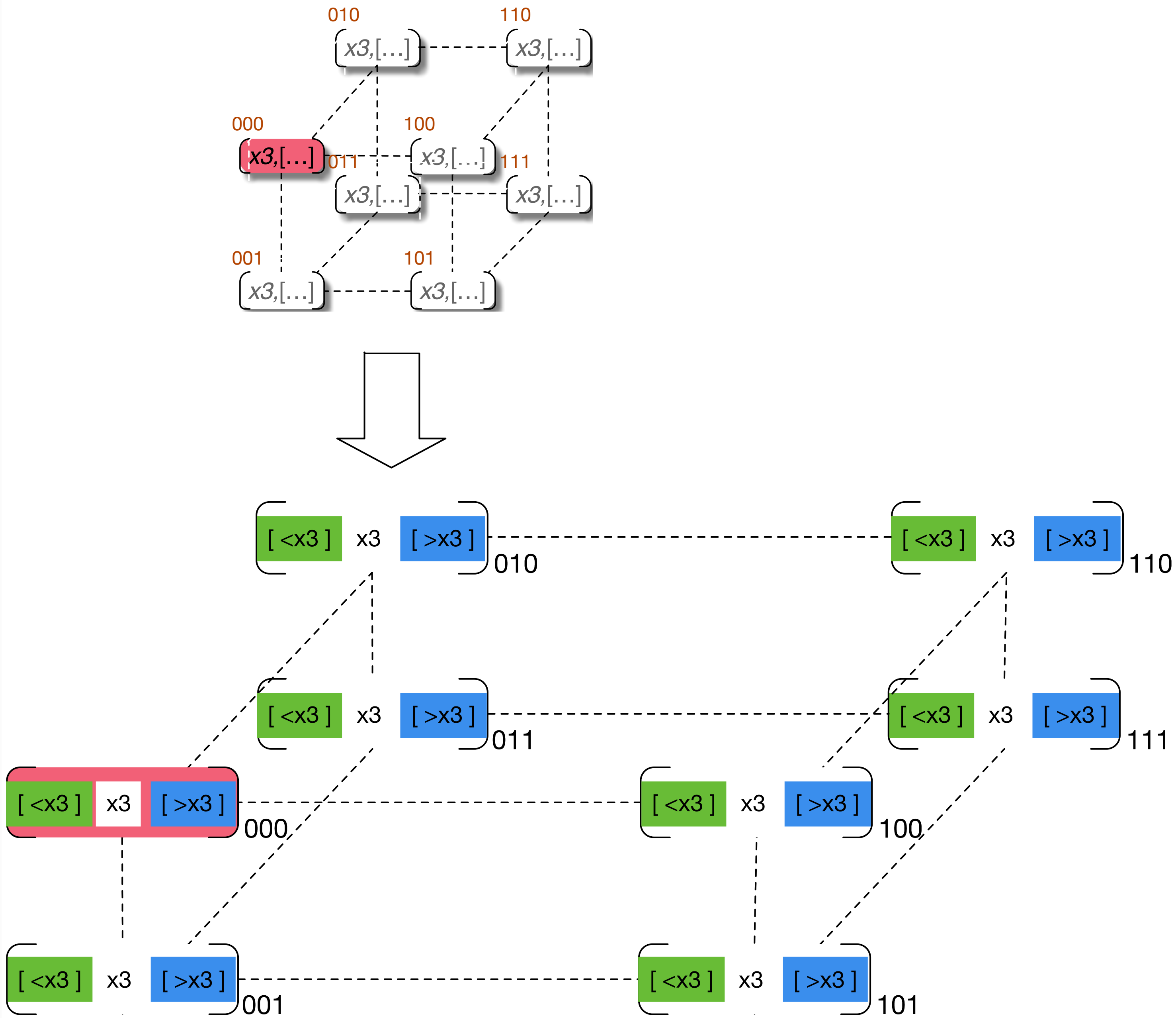
1. Broadcast Pivot

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)

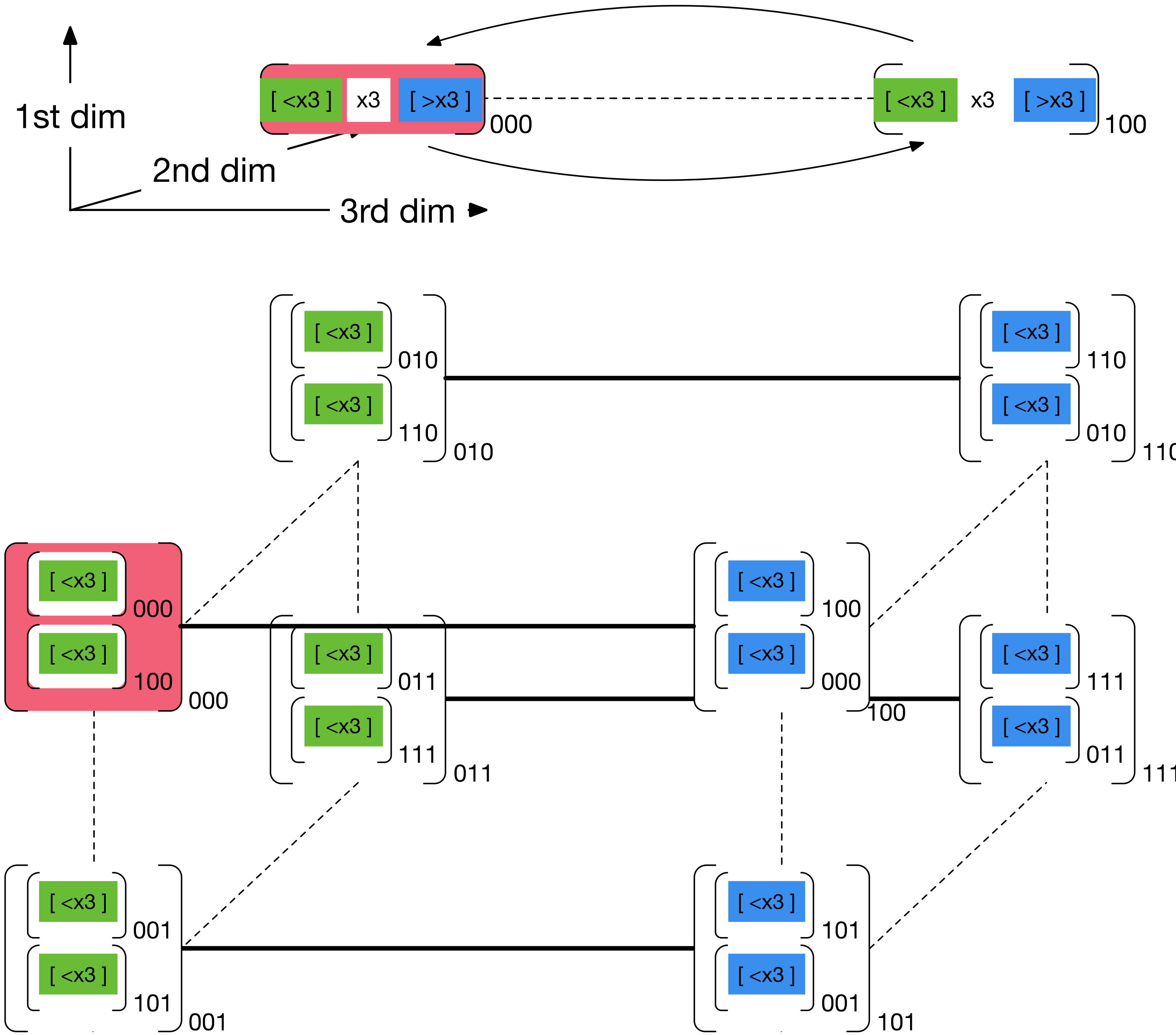


3. On the d^th dimension exchange data

In step 1, d = 3

So we will exchange data on the 3rd dimension. i.e., we will exchange data between 0xx and 1xx
→ y0x and y1x (level 2)
→ yy0 and yy1 (level 3)

After the exchange, discard the pivot



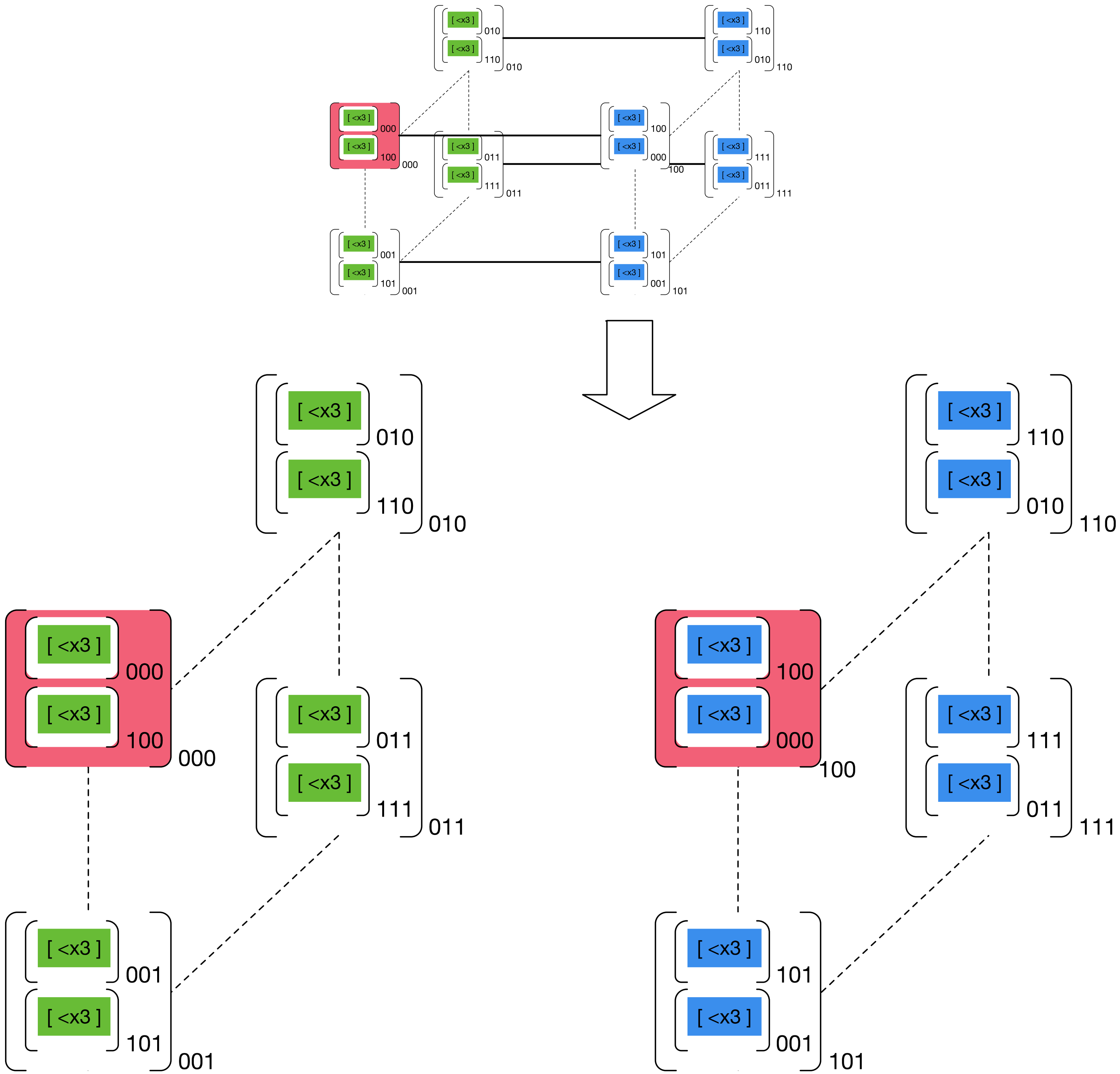
4. Let's split in to 2 d-1 cubes

In step 1, d = 3

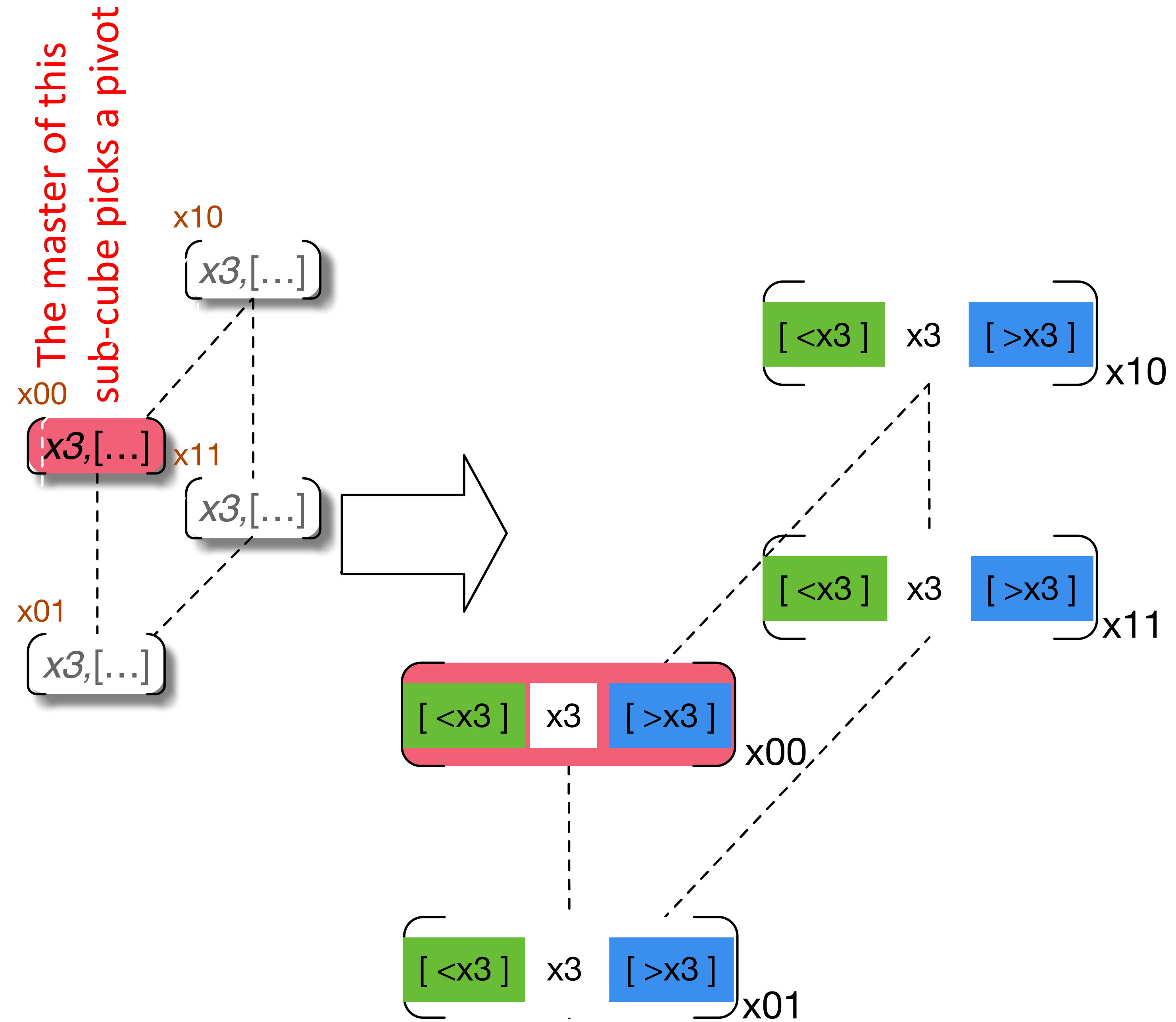
So we will exchange data on the 3rd dimension. i.e., we will exchange data between 0xx and 1xx

After the exchange, discard the pivot

Select new masters of the 2 sub-cubes



5. Repeat until you reach 1D cubes



Step 6

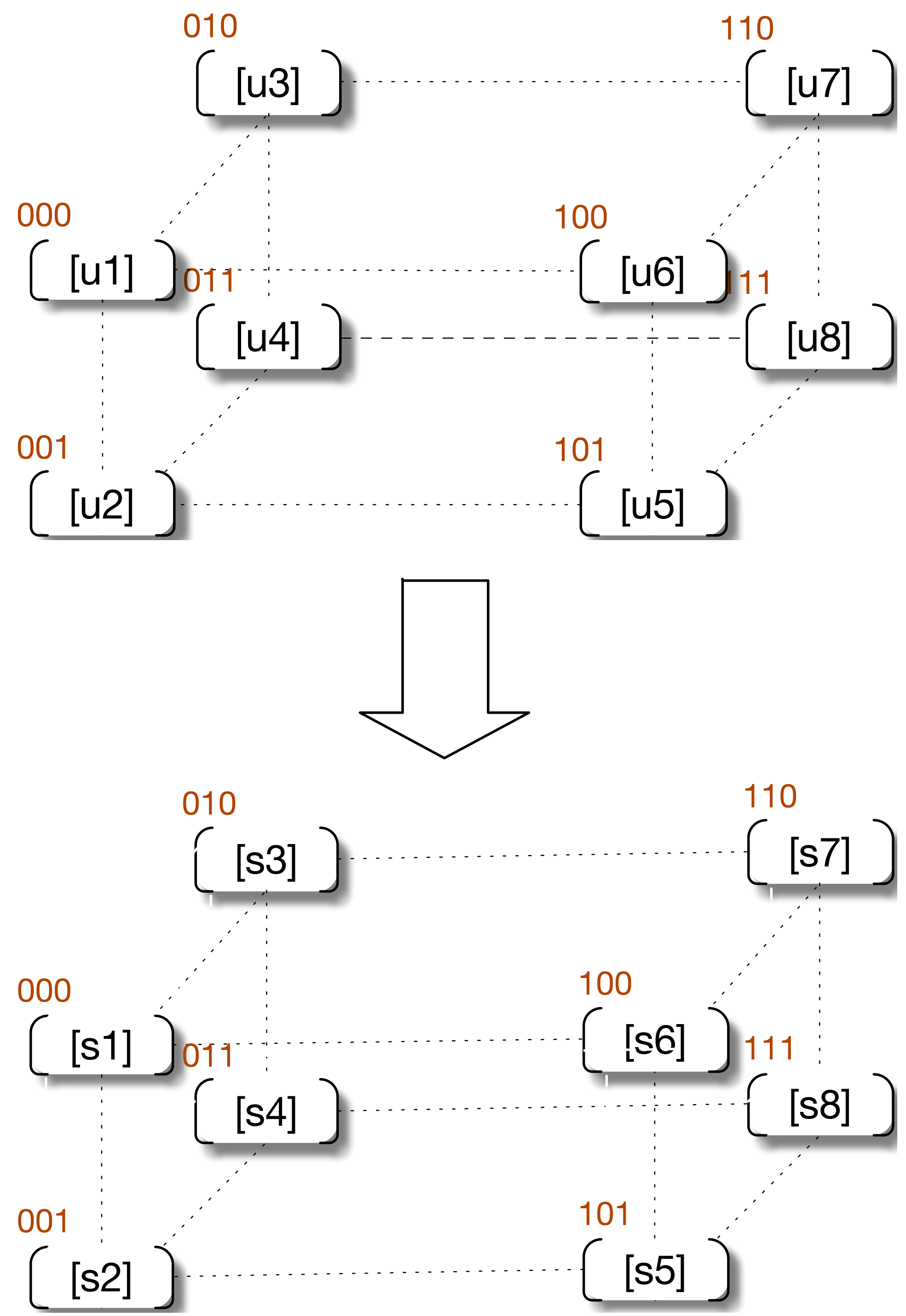
Once you reach 1D sub-cubes, you have your elements chunked into 2^d sorted sets

[u1] < [u2] < [u3] < … < [u8]

Apply quicksort on each of the chunks

[s1] < [s2] < [s3] … < [s8]

S → sorted version of an unsorted chunk U



All to One Reduce

→ 000 will get sorted numbers

Complexity Analysis


```
for L := dimension downto 1  logp
begin
  if master then
    choose a pivot value for the L-dimensional subcube;  n/p
  broadcast the pivot from the master to the subcube members;  [1,logp]
  partition list[0:n_element-1] into two sublists such that  n/p
  list[0:j] ≤ pivot < list[j+1:n_element-1];
  if lower_partner then
    begin
      send the right sublist list[j+1:n_element-1] to partner;
      receive the left sublist of partner;                        n/p
    end
  else if higher_partner then
    begin
      send the left sublist list[0:j] to partner;
      receive the right sublist of partner;          n/p
    end
  n_element := n_element - n_send + n_receive;
end
sequential quicksort to list[0:n_element-1]    (n/p)log(n/p)
```

Slides from Aiichiro Nakano

Complexity Analysis


```
for L := dimension downto 1  logp
begin
  if master then
    choose a pivot value for the L-dimensional subcube;  n/p
  broadcast the pivot from the master to the subcube members;  [1,logp]
  partition list[0:n_element-1] into two sublists such that  n/p
  list[0:j] ≤ pivot < list[j+1:n_element-1];
  if lower_partner then
```

$$T_{\text{average}} = O\left(\frac{n}{p}\log\frac{n}{p}\right) + O\left(\frac{n}{p}\log p\right) + O\left(\log^2 p\right)$$

```
                                         to partner;
                                                        n/p
    end
  else if higher_partner then
    begin
      send the left sublist list[0:j] to partner;
      receive the right sublist of partner;         n/p
    end
  n_element := n_element - n_send + n_receive;
end
sequential quicksort to list[0:n_element-1]    (n/p)log(n/p)
```

Slides from Aiichiro Nakano