CSE 6220/CX 4220
Introduction to HPC

Lecture 13: Embeddings

Helen Xu
hxu615@gatech.edu

Georgia Tech College of Computing
School of Computational
Science and Engineering

Slide credits to Prof. Srinivas Aluru

Recall: Hypercubic Permutations

- A hypercubic permutation is one in which the **processors that communicate differ by only one bit** in their ranks.

- In a “relaxed” hypercubic permutation, the bits that are flipped do not have to be in the same index across ranks.

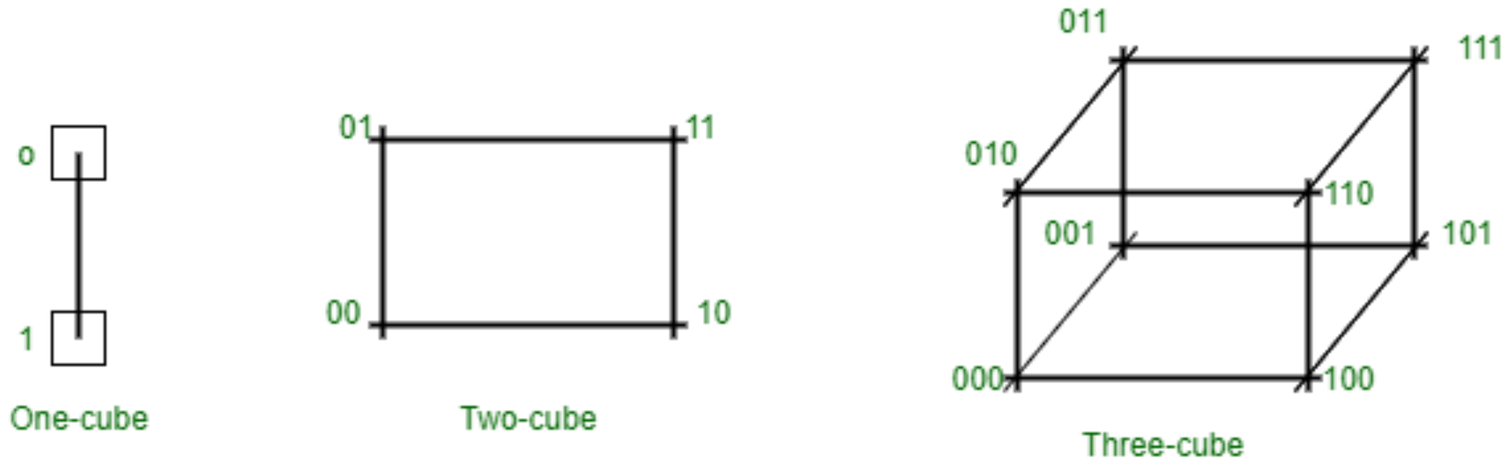
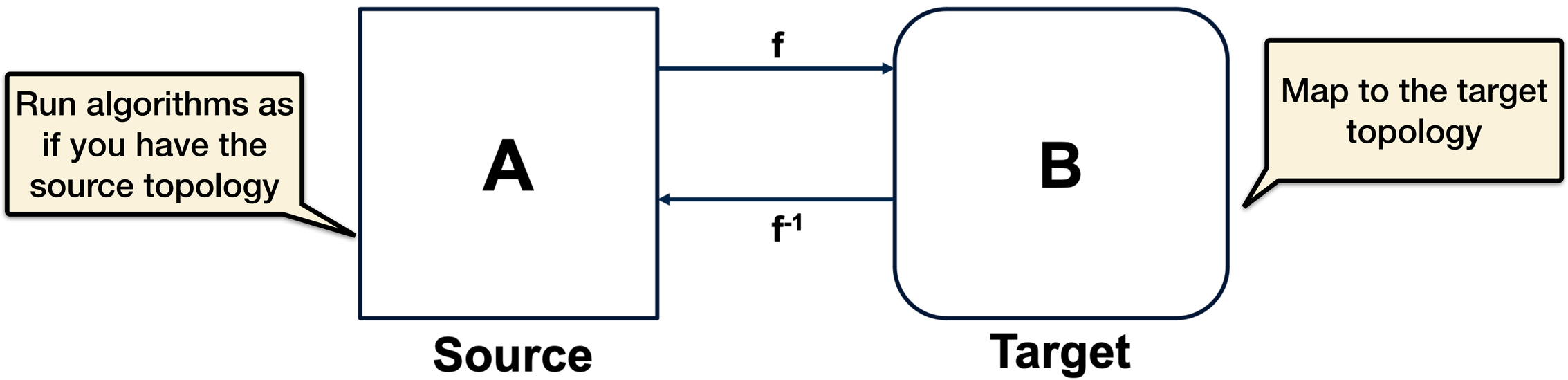


Figure - Hypercube Structures For n = 1,2,3

https://www.geeksforgeeks.org/hypercube-interconnection/

Embeddings



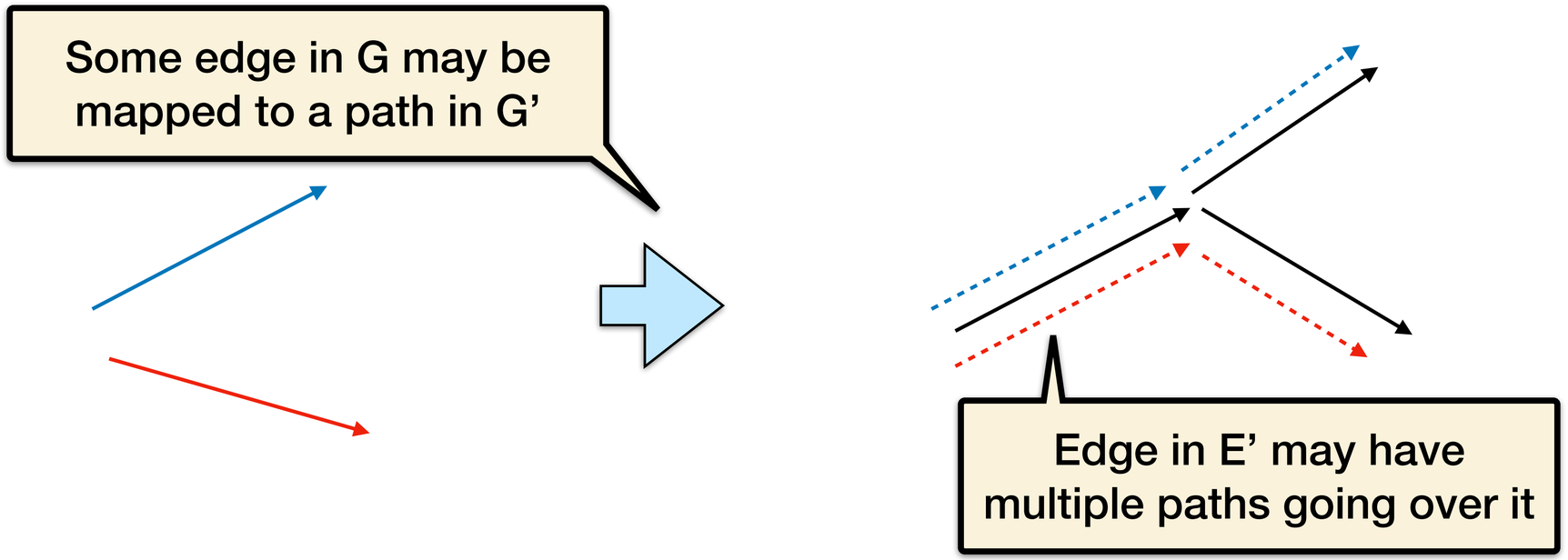
- Algorithm designed for “Source” network. The system you have is “Target” network.
- Source and Target can be modeled as graphs.
- We are interested in the case where “Target” network can support (relaxed) hypercubic permutations.

Embedding: Performance Metrics

Embedding a source graph *G(V,E)* into a target graph *G'(V',E')*:

- ***Congestion:*** Maximum number of edges from E that are mapped onto a single edge in *E'.*
- ***Dilation:*** Maximum number of edges in *E'* that any edge in *E* is mapped to.

Communication slows down by a factor of congestion x dilation

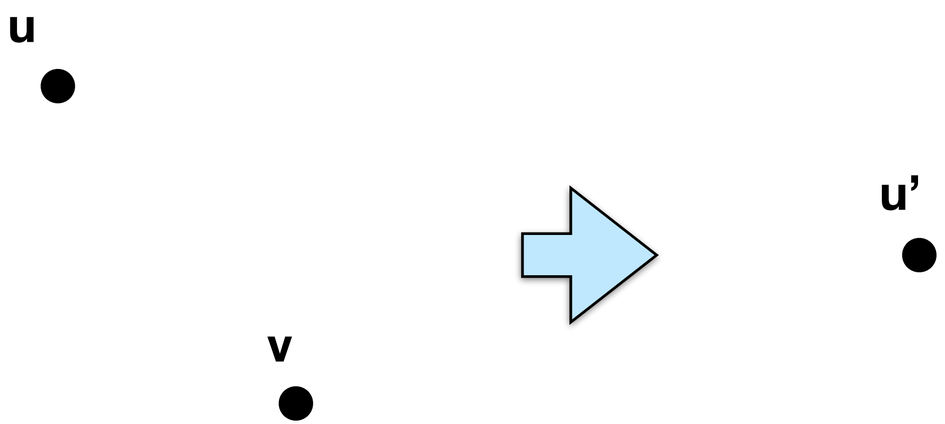


Embedding: Performance Metrics

Embedding a source graph *G(V,E)* into a target graph *G'(V',E')*:

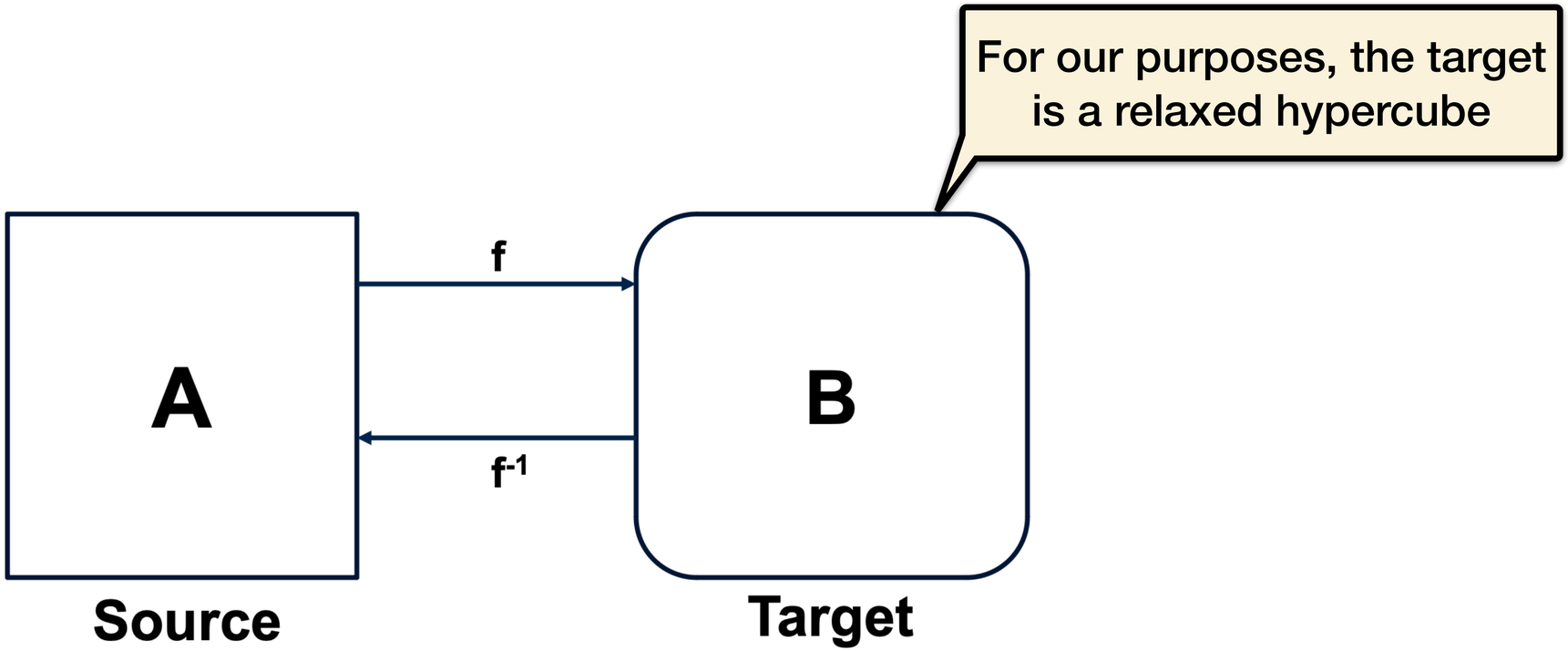
- ***Load:*** Maximum number of nodes in *V* that map to a single node in *V'*.
- ***Expansion:*** The ratio of the number of nodes *V'* to that in *V*.

Computation slows down by a factor of Load.
Expansion > 1 indicates waste of resources.



Mappings

A mapping is a special case of an embedding where **Load = Congestion = Dilation = 1**

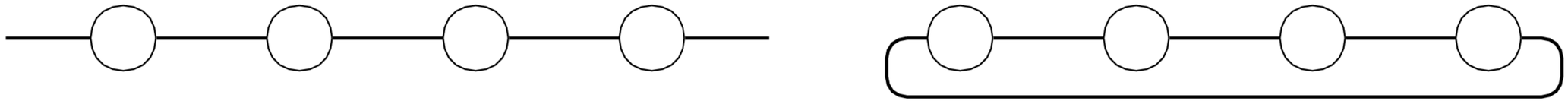


Mapping Properties

When embedding from source to target is a mapping:

- Mapping only needs to specify node mapping. Edge mapping is implied.
- Parallel run-time on the target topology is the same as that on the source topology.
- Efficiency is preserved when execution of the algorithm is shifted from source topology to target topology!
- It is possible there can be a different parallel algorithm with better efficiency directly designed for the target topology.
- This concern is eliminated if the parallel algorithm designed for the source topology is optimally efficient.

Network Topologies: Linear Arrays

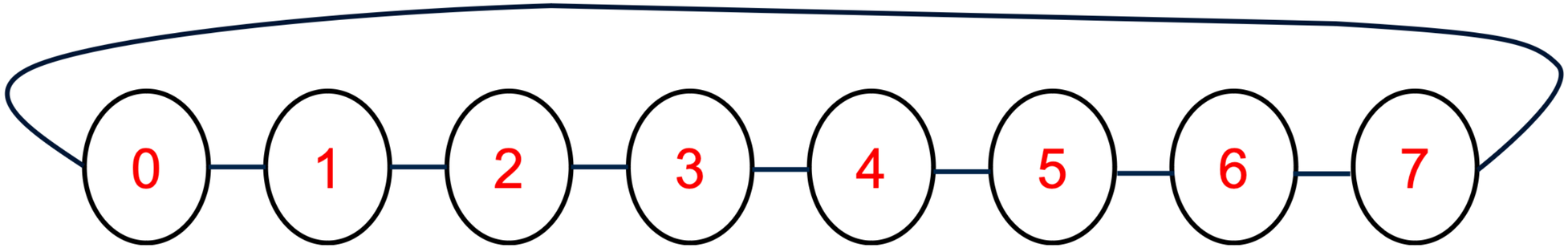


(a) (b)

Linear arrays: (a) with no wraparound links; (b) with wraparound link.

Embedding Ring into a Hypercube

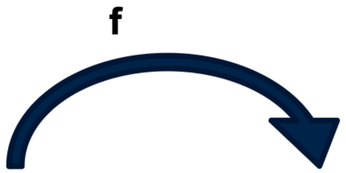
Applies to array, which is just a subset



Why can’t we just directly map the source ranks to the target ranks?

Gray Codes

Gray codes are a binary ordering such that two successive values **differ in only one bit**.

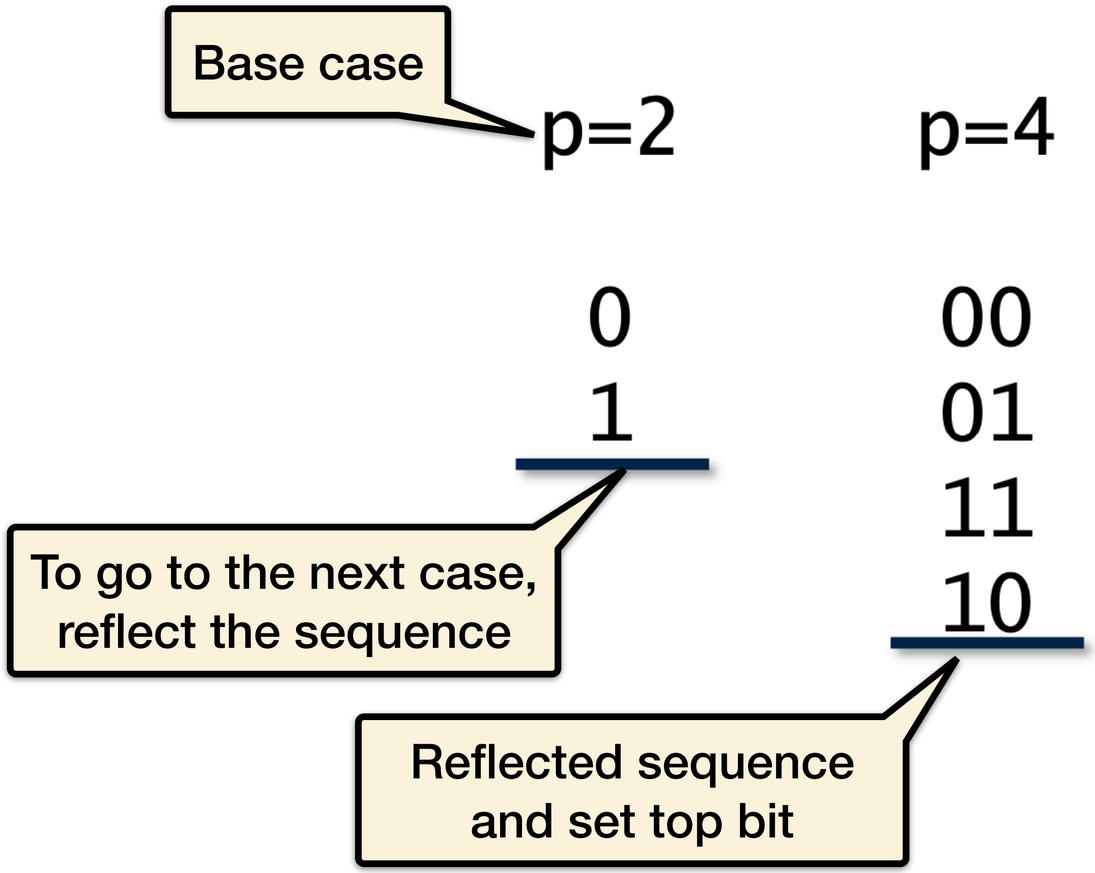


	Binary Code	Gray Code
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

Binary Reflected Gray Code (BRGC)

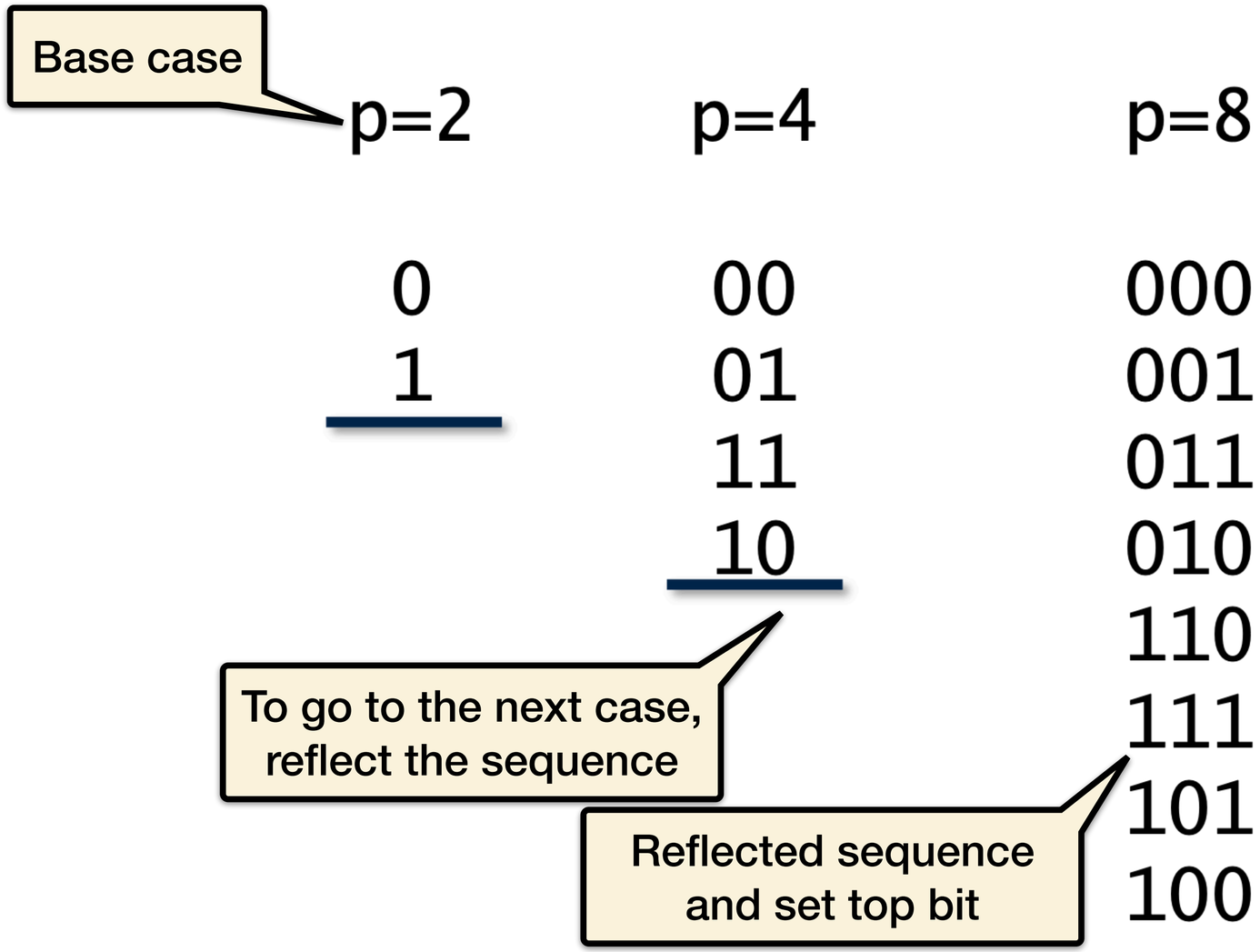
0
1

Binary Reflected Gray Code (BRGC)



If we have a Gray code for b bits, how does this preserve the property of differing by one bit for b+1 bits?

Binary Reflected Gray Code (BRGC)



Embedding a Linear Array into a Hypercube

- A linear array (or a ring) composed of 2^d nodes (labeled 0 through $2^d - 1$) can be embedded into a *d*-dimensional hypercube by mapping node *i* of the linear array onto node *G(i, d)* of the hypercube.

- The function *G(i, x)* is defined as follows:

$$
\begin{aligned}
G(0,1) &= 0 \\
G(1,1) &= 1 \\
G(i,x+1) &= \begin{cases} G(i,x), & i < 2^x \\ 2^x + G(2^{x+1} - 1 - i, x), & i \geq 2^x \end{cases}
\end{aligned}
$$

Embedding a Linear Array into a Hypercube

Adjoining entries (*G(i, d)* and *G(i + 1, d)*) differ from each other at only one bit position

Corresponding processors are mapped to neighbors in a hypercube.

What is the congestion? dilation? expansion? load?

Embedding a Linear Array into a Hypercube

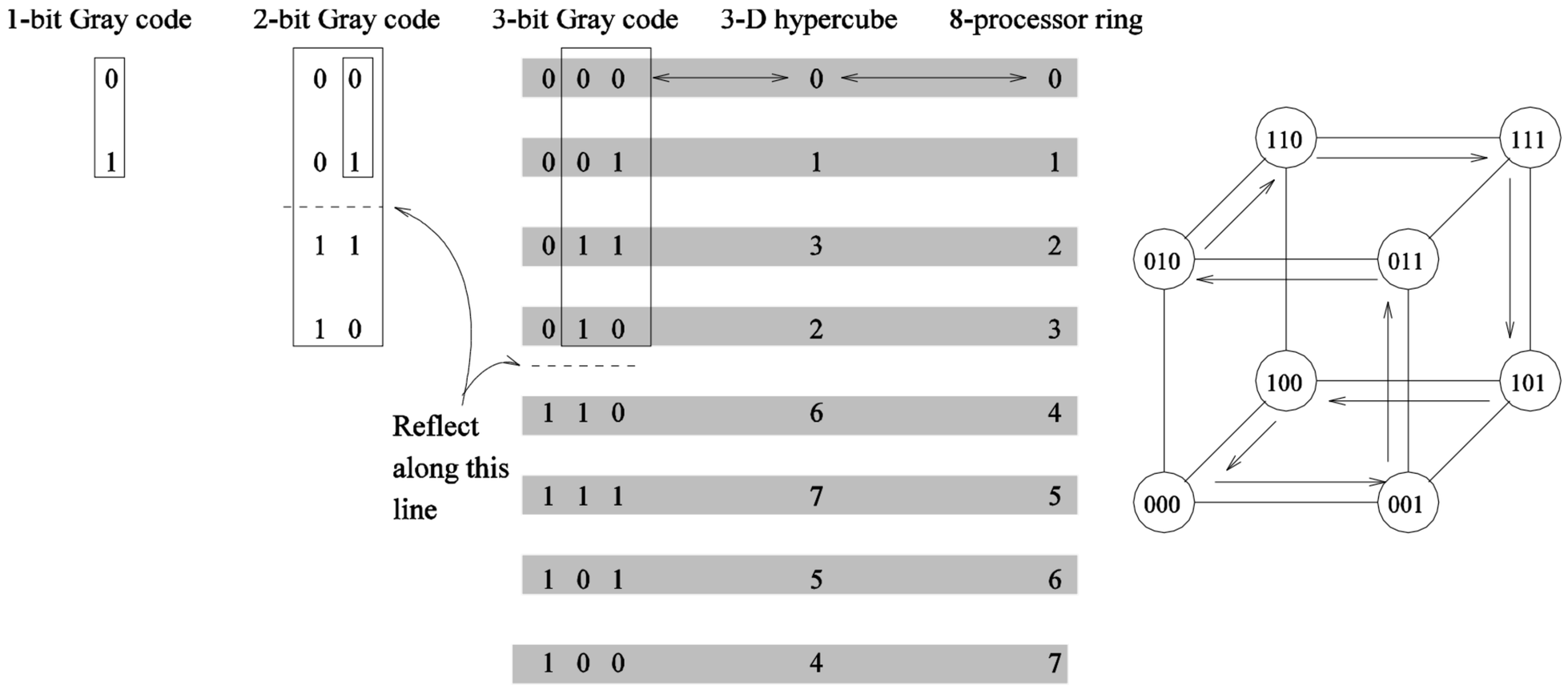
Adjoining entries (*G(i, d)* and *G(i + 1, d)*) differ from each other at only one bit position

Corresponding processors are mapped to neighbors in a hypercube.

What is the congestion? dilation? expansion? load?

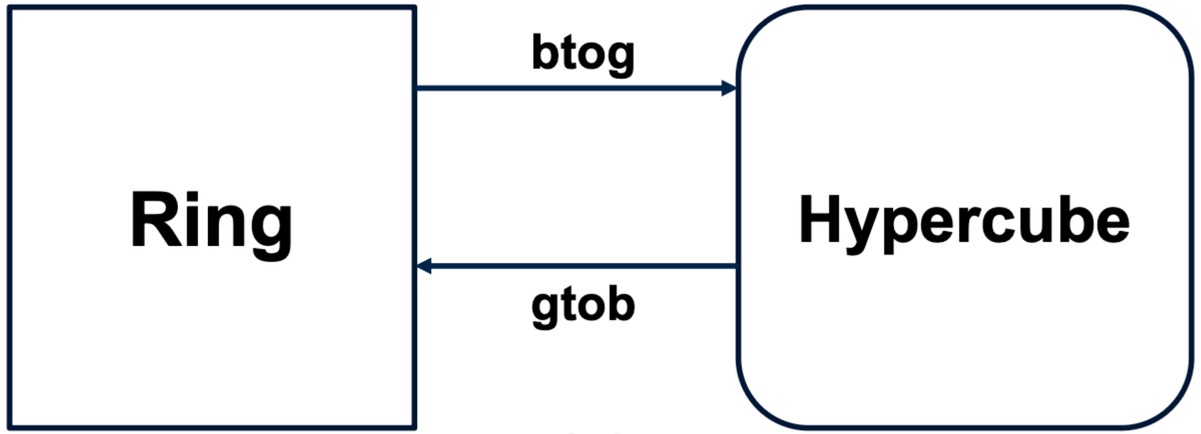
(all are 1 in the mapping)

Example

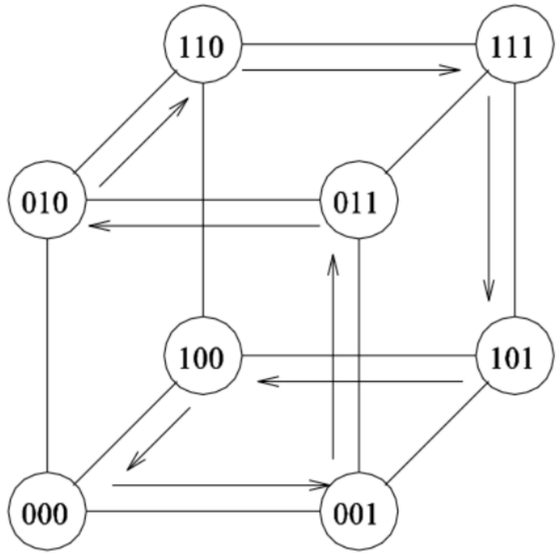


A three-bit reflected Gray code ring, and its embedding into a three-dimensional hypercube.

Embedding Ring to Hypercube



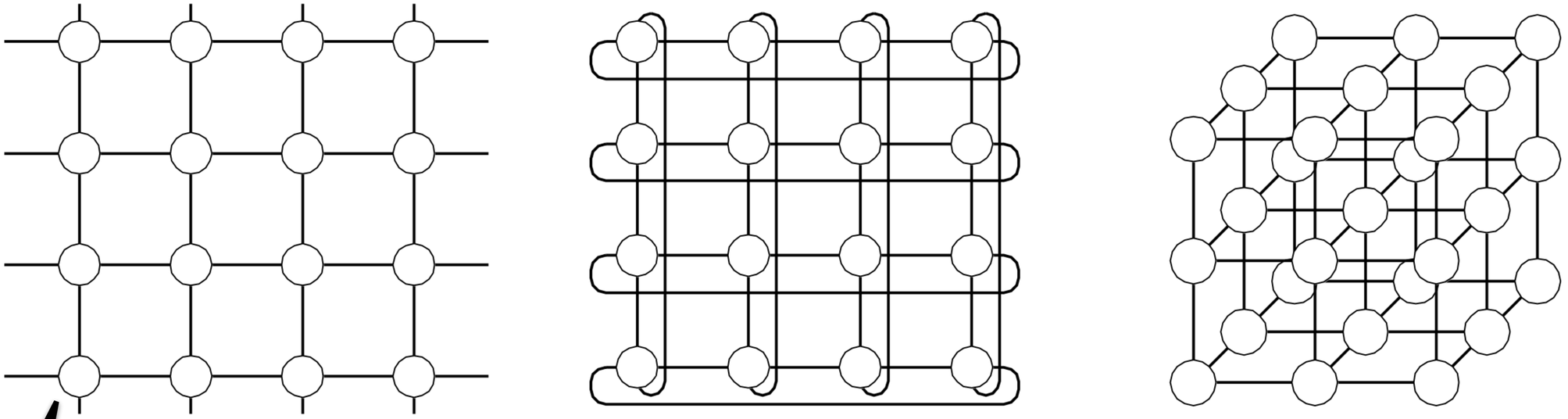
- Assume processor rank is r
- Ring rank = gtob(r)
- Left neighbor = btog[(gtob(r)-1+p) mod p]
- Right neighbor = btog[(gtob(r)+1) mod p]
- Example Processor with rank 3
  - Ring rank = gtob(3) = 2
  - Left neighbor = btog(2-1) = btog(1) = 1
  - Right neighbor = btog(2+1) = btog(3) = 2



Arrow points to right neighbor

Relaxed hypercubic permutation

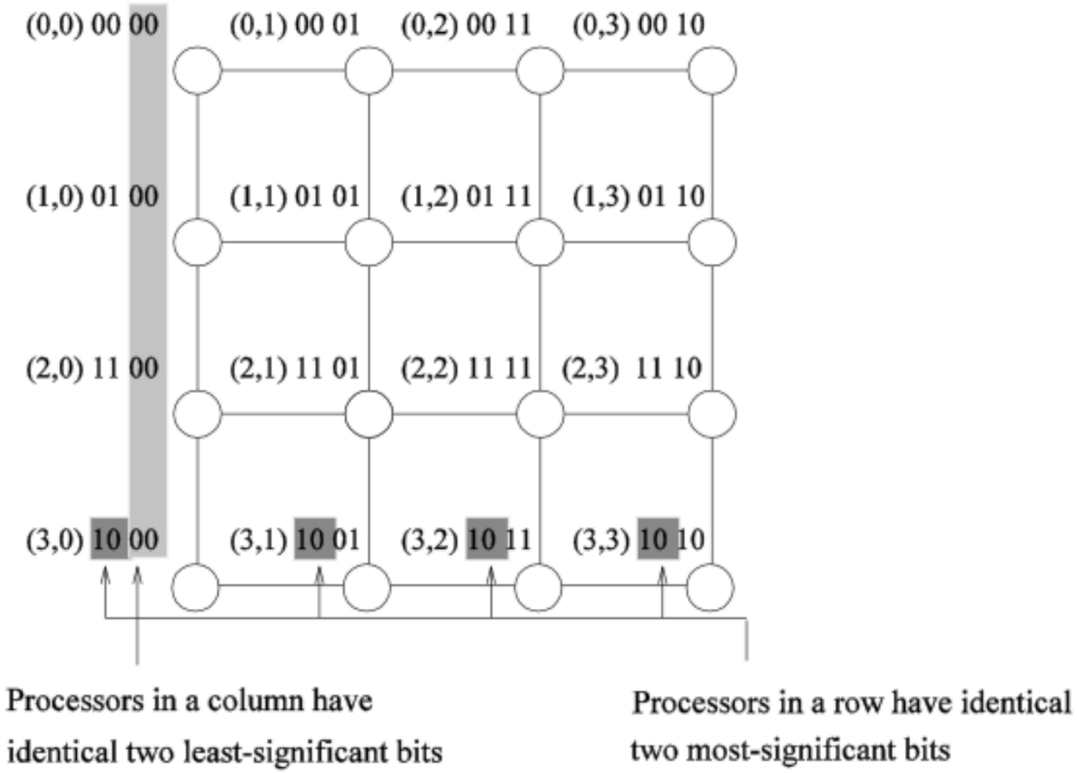
Network Topologies: Two- and Three-Dimensional Meshes



Dimensions need to be a power of 2 length

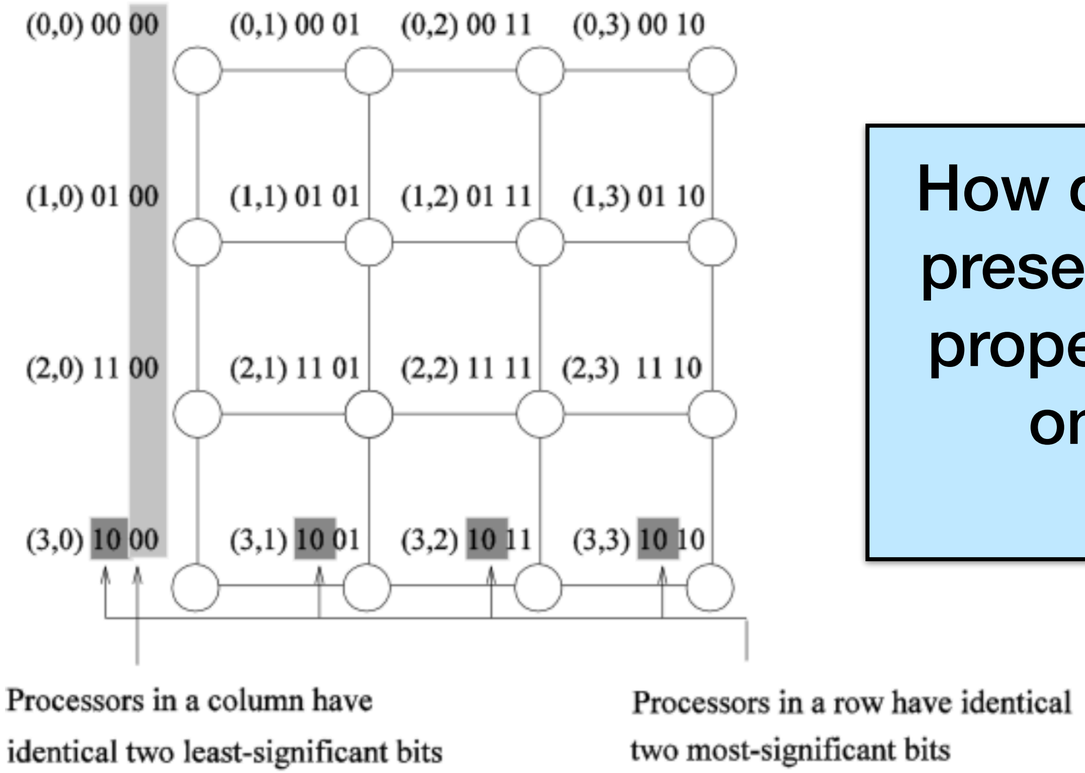
Two and three dimensional meshes: (a) 2-D mesh with no wraparound; (b) 2-D mesh with wraparound link (2-D torus); and (c) a 3-D mesh with no wraparound.

Embedding a Mesh into a Hypercube



- A $2^r \times 2^s$ wraparound mesh can be mapped to a 2^{r+s} node hypercube by mapping node (i, j) of the mesh onto node btog(i) || btog(j) of the hypercube (where || denotes concatenation of the two Gray codes).

Embedding a Mesh into a Hypercube



How does this mapping preserve the Gray code property of differing by one bit between neighbors?

- A $2^r \times 2^s$ wraparound mesh can be mapped to a 2^{r+s} node hypercube by mapping node (i, j) of the mesh onto node btog(i) || btog(j) of the hypercube (where || denotes concatenation of the two Gray codes).

Embedding a Mesh into a Hypercube

- $2^r \times 2^s$ mesh; $p = 2^{r+s}$

- rank = bit string of (r+s) bits
- rank = x || y, where
  - x= first r bits,
  - y= last s bits

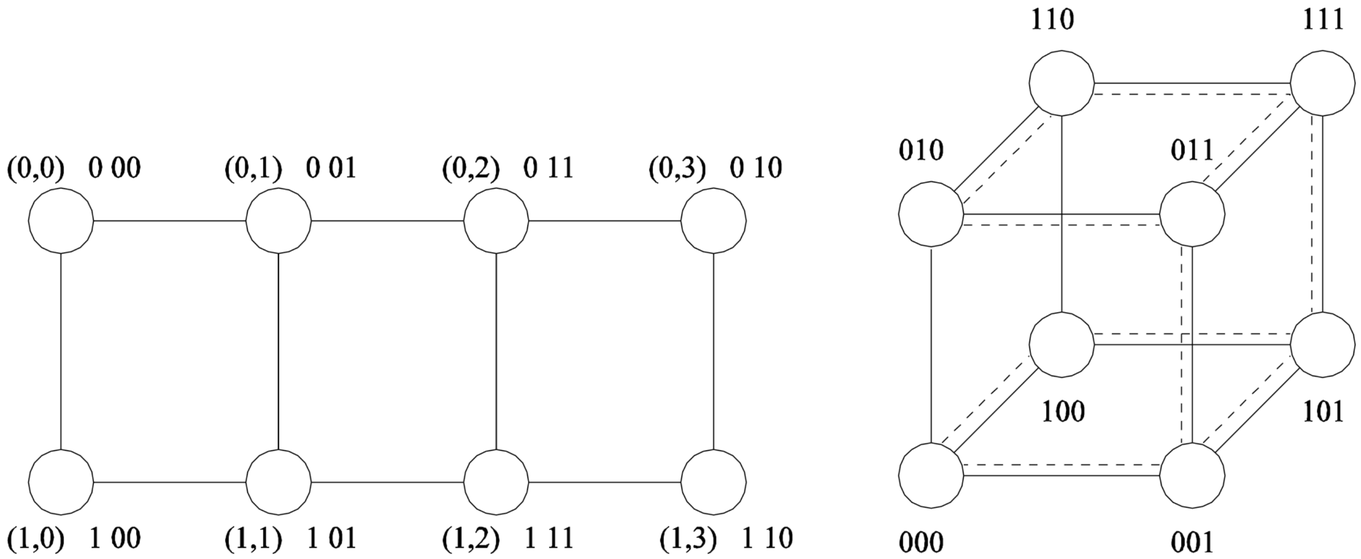
- mesh rank = (gtob(x), gtob(y))
- East = btog[(gtob(x)+1) mod 2^r] || y
- West = btog[(gtob(x)-1+2^r) mod 2^r] || y
- North = x || btog[(gtob(y)-1+2^s) mod 2^s]
- South = x || btog[(gtob(y)+1) mod 2^s]

Example

- 8 x 4 x 16 x 2 x 8 → 8,192 = 2^{13}
- (5, 2, 3, 1, 7) → 111 11 0010 1 100

Binary Code	Gray Code
000	000
001	001
010	011
011	010
100	110
101	111
110	101
111	100

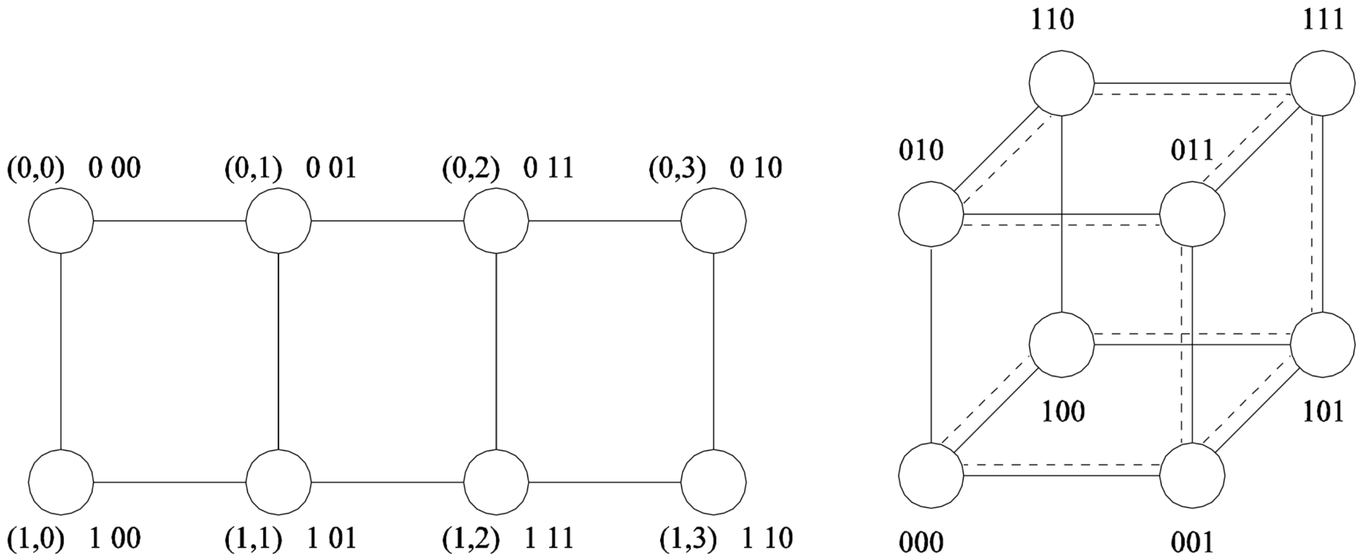
Example



- A 2 × 4 mesh embedded into a three-dimensional hypercube

What is the congestion? dilation? expansion? load?

Example



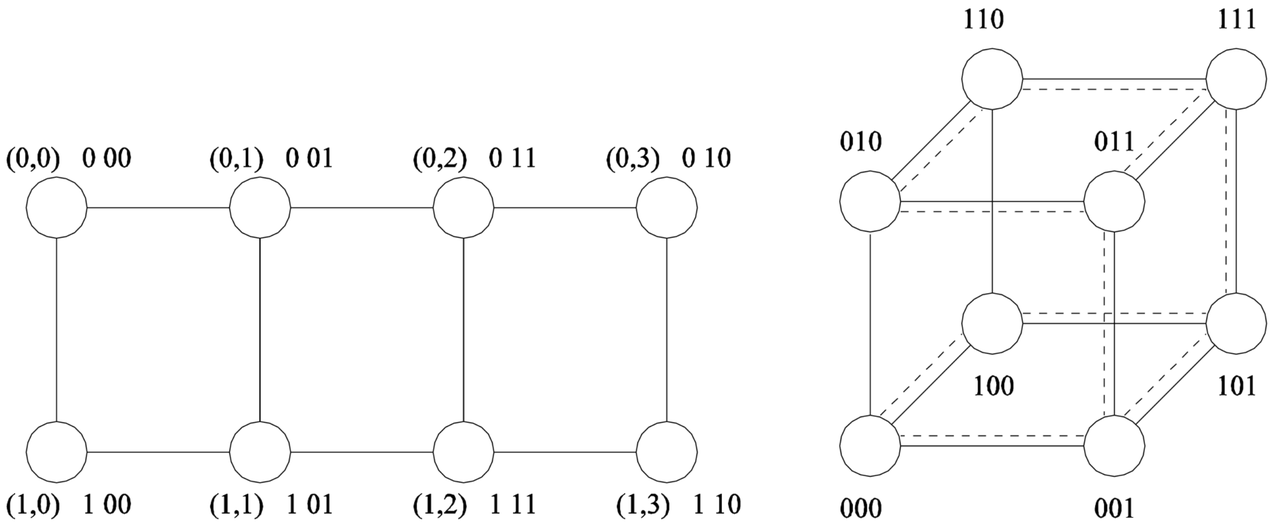
- A 2 × 4 mesh embedded into a three-dimensional hypercube

What is the congestion? dilation? expansion? load?

(all are 1 in the mapping)

Example: Performance Difference Between Mappings

Consider the broadcast operation. How long does it take in a square mesh? What about a hypercube?

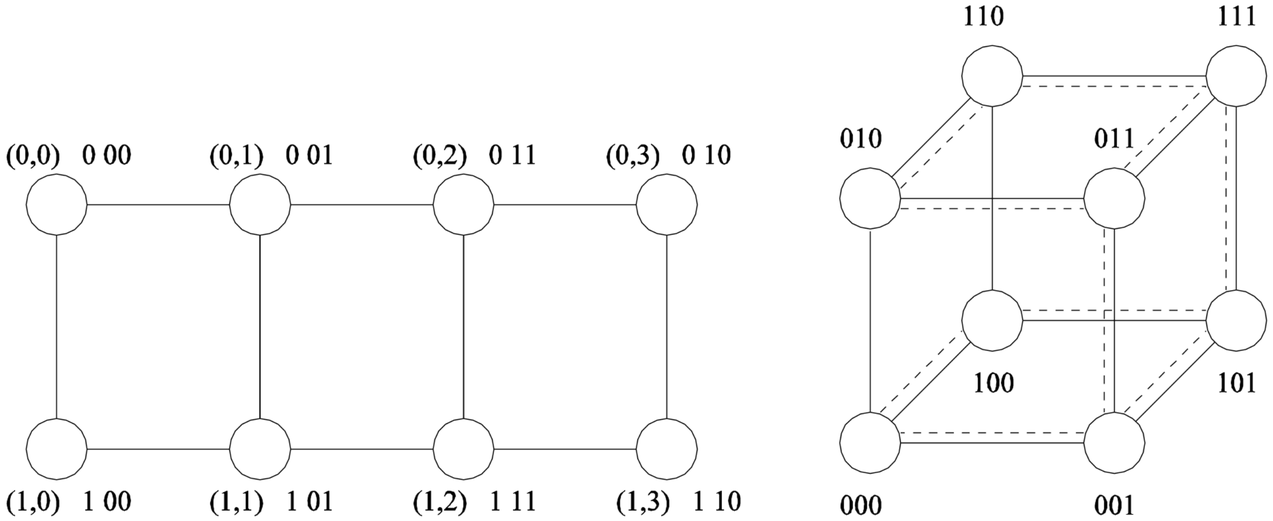


- A 2 × 4 mesh embedded into a three-dimensional hypercube

Example: Performance Difference Between Mappings

Consider the broadcast operation. How long does it take in a square mesh? What about a hypercube?

In mesh = $O(\sqrt{p})$, in hypercube = $O(\lg p)$

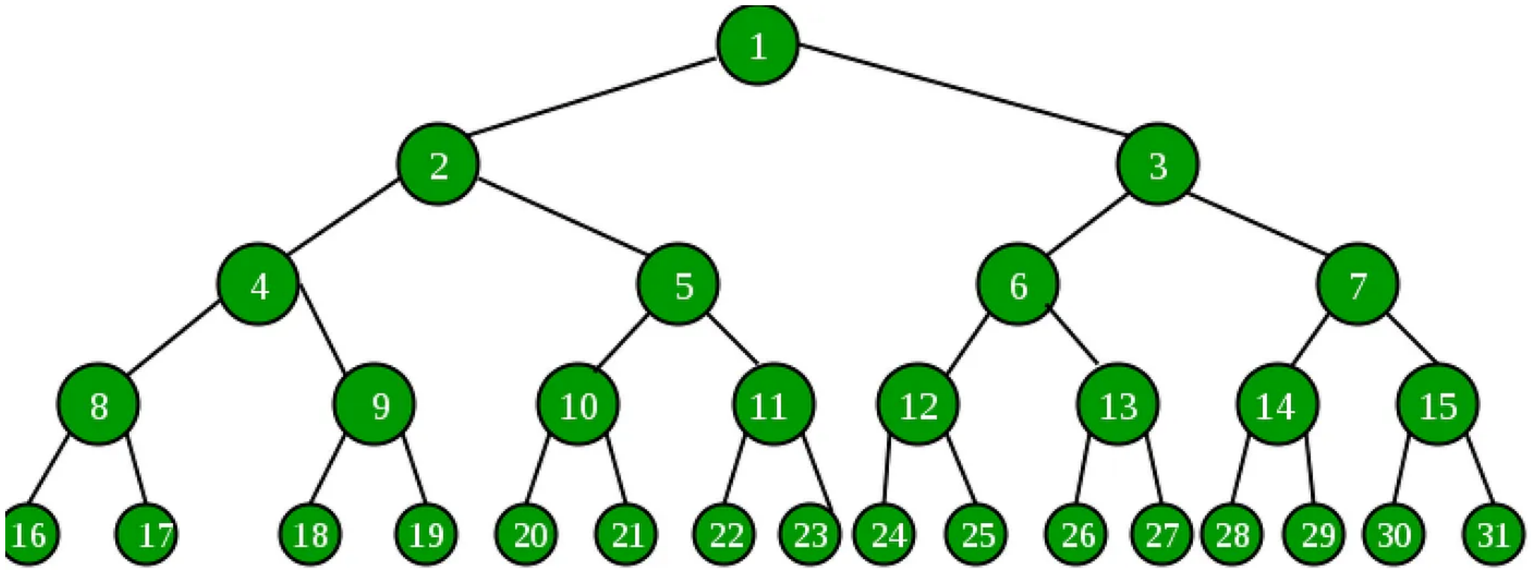


- A 2 × 4 mesh embedded into a three-dimensional hypercube

Binary Tree Topology

- Some applications and algorithms are well-suited to a binary tree topology for the processors.

- For example, it is easy to see the broadcast algorithm on a tree. Or in image processing, images are divided into sections recursively in a tree-like fashion.



https://jerryan.medium.com/binary-tree-cheat-sheet-for-next-interview-7b442a84d70

Can we Embed a Complete Binary Tree into a Hypercube?

Embed p leaf ($2p - 1$ node) binary tree into a $2p$-node hypercube.

- Without loss of generality, assume root is mapped to a node with even number of zeros.
- Then, every alternative level starting from the root is mapped to processors with even number of zeros.
- The entire leaf level is mapped to target processors with same parity (i.e. either even or odd number of zeros).
- Number of such processors of same parity required

$$= p + \frac{p}{4} + \ \ldots\ldots\ldots$$

$$> p$$

➔ Impossible!

Can we Embed a Complete Binary Tree into a Hypercube?

Embed p leaf ($2p - 1$ node) binary tree into a $2p$-node hypercube.

- Without loss of generality, assume root is mapped to a node with even number of zeros.
- Then, every alternative level starting from the root is mapped to processors with even number of zeros.
- The entire leaf level is mapped to target processors with same parity (i.e. either even or odd number of zeros).
- Number of such processors of same parity required

$$= p + \frac{p}{4} + \ \ldots\ldots\ldots$$

$$> p$$

➔ Impossible!

Hinges on property that all processors in a level have the same parity

Modifying Assignment to Perform Mapping

The previous proof hinges on the fact that all processors in the same tree level have the same parity.

How can we fix the assignment to make the mapping go through?

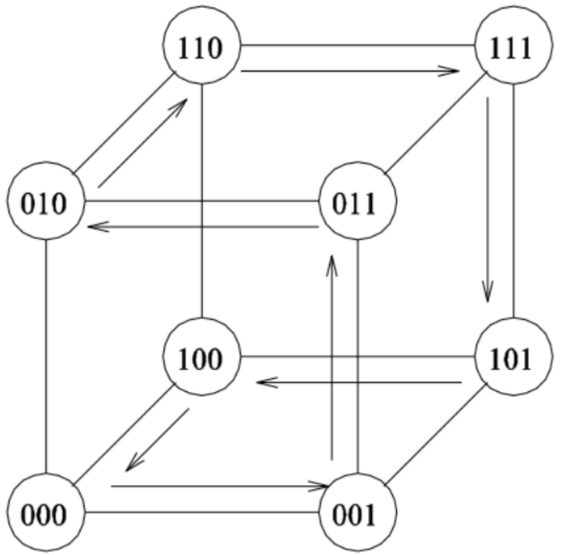
Modifying Assignment to Perform Mapping

The previous proof hinges on the fact that all processors in the same tree level have the same parity.

One idea - split between **even and odd** in the children.

What is the dilation?

Dilation is 2 - may have to go across 2 links rather than 1



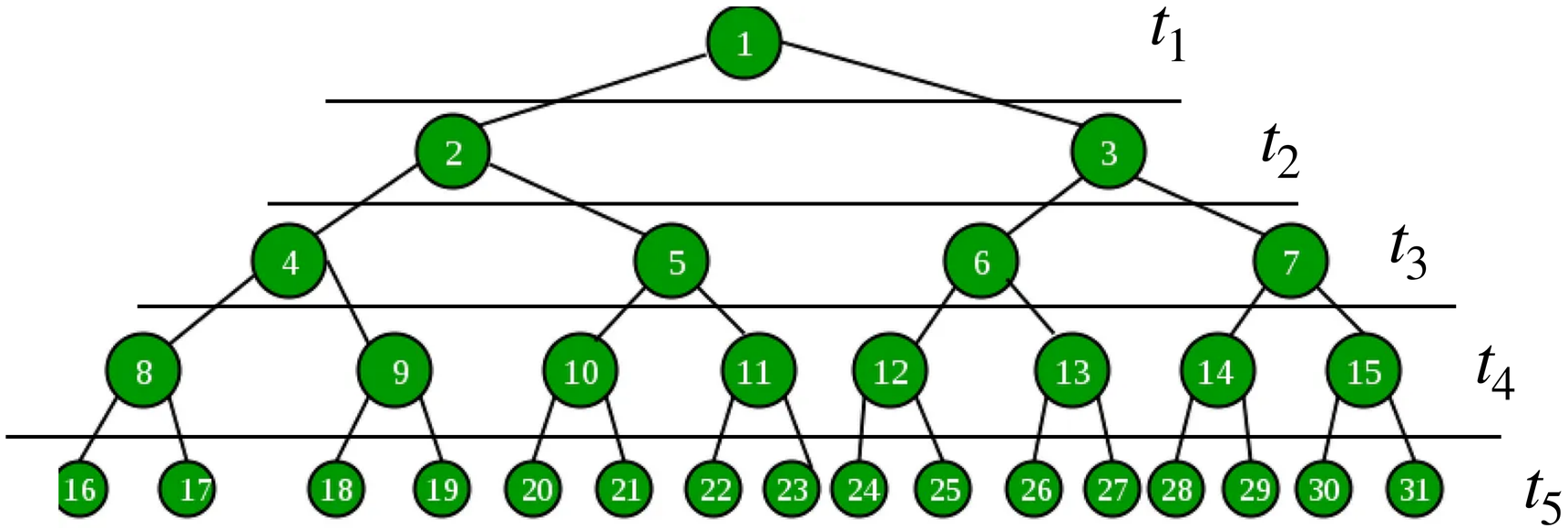
Arrow points to right neighbor

Relaxed hypercubic permutation

Modifying Assignment to Perform Mapping

Most tree algorithms proceed **level by level** - the processors within a level act in parallel, but the levels are done sequentially.

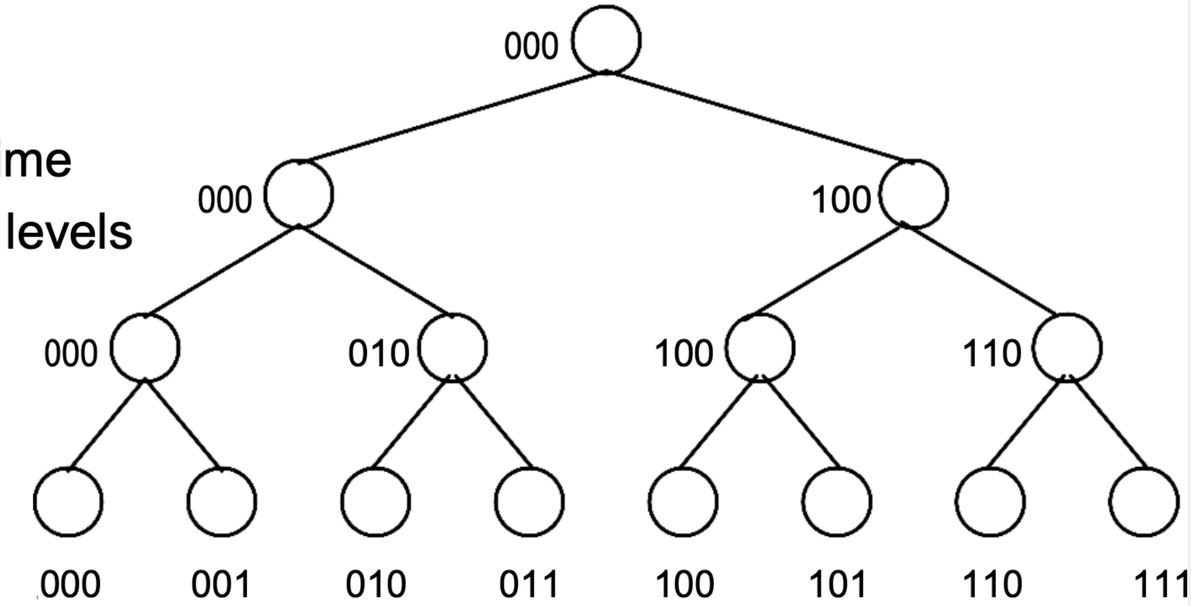
Another idea - **Reuse** some processors between levels



https://jerryan.medium.com/binary-tree-cheat-sheet-for-next-interview-7b442a84d70

Embedding Binary Tree into Hypercube

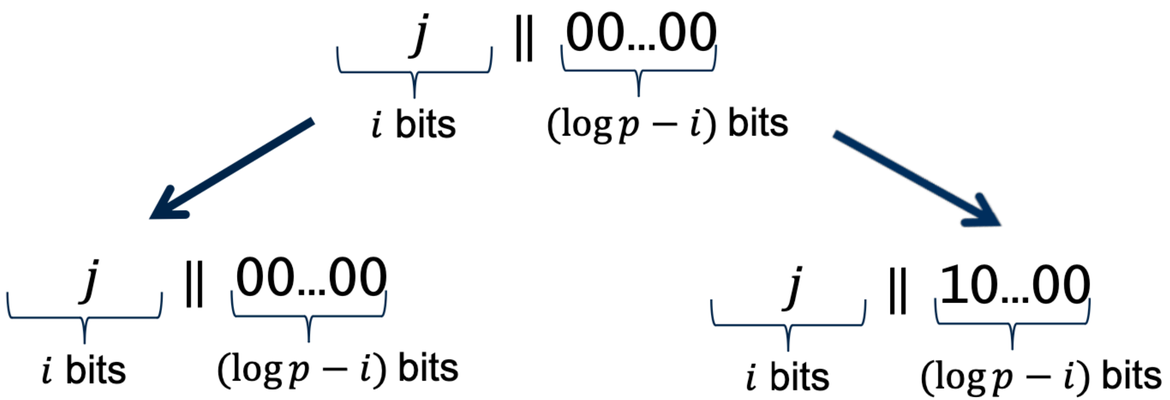
- p-processor binary tree;
  p is a power of 2.
- Comp: one level of tree at a time
- Comm: between consecutive levels
- Tree has $(\log p + 1)$ levels;
  Root at level 0.
- Level i has 2^i nodes,
  numbered $0 \,..\, 2^i - 1$
- Tree node (i, j) mapped to



$$\text{rank} = \underbrace{j}_{i \text{ bits}} \,||\, \underbrace{00\ldots0}_{(\log p - i) \text{ bits}}$$

Embedding Binary Tree into Hypercube

In tree, (i, j) is connected to $(i + 1, 2j)$ and $(i + 1, 2j + 1)$



Processor with rank r participates in levels $(\log p - 1)$ ... $(\log p - j)$
Where j = number of trailing zeroes in r. At level k:

Parent: change $(\log p - k)^{th}$ bit to 0.

Left child: self

Right child: change $(\log p - k - 1)^{th}$ bit to 1.