Fault-Tolerant Routing in Dual-Cube

Yamin Li

Department of Computer Science
Faculty of Computer and Information Sciences
Hosei University, Tokyo 184-8584 Japan

http://cis.k.hosei.ac.jp/~yamin/

Tutorial Outline

- Motivation
- Dual-cube interconnection network
- Collective communications
- Disjoint paths
- Fault-free cycle embedding
- Fault-tolerant routing
- References

Section I

Motivation

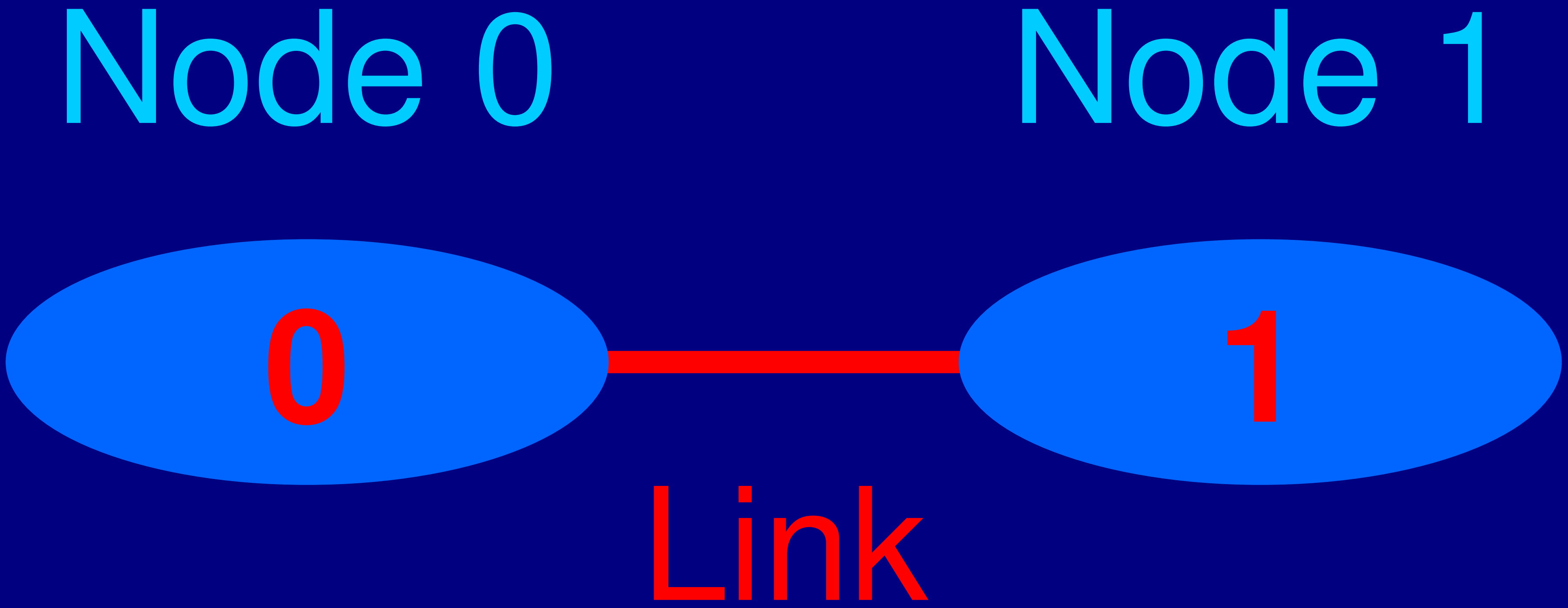
WWW — What We Want

- Dual-cube: a new interconnection network
    - Low node degree (number of links per node)
    - Shorter diameter (distance between two nodes)
    - Symmetric (with recursive structure)
    - Easy to route (similar to hypercube)
- Algorithms for basic communication operations
- Linear array or ring embedding
- Algorithms for fault-tolerant routing
    - Local-information based
    - Run at linear time

Hypercubes

- The binary hypercube has been widely used as the interconnection network in parallel systems:
  - Intel iPSC, nCUBE, Connection Machine CM-2, SGI Origin 2000/3000.
- A hypercube network of dimension *n*, or *n*-cube, contains up to 2^n nodes and has *n* edges per node.
- If unique *n*-bit binary addresses are assigned to the nodes of hypercube, then an edge connects two nodes if and only if their binary addresses differ in a single bit.

1-Cube



Node 0

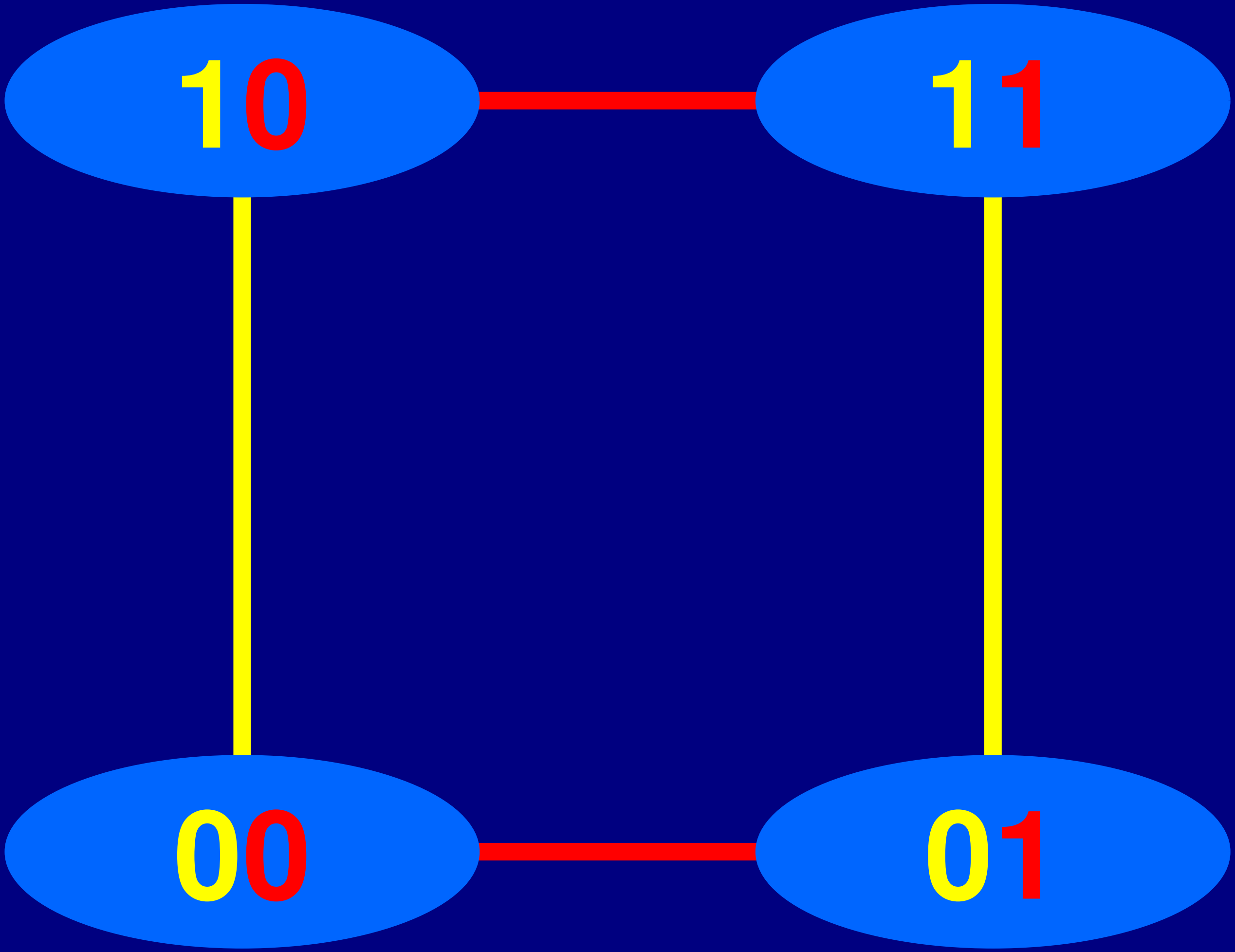
Node 1

0

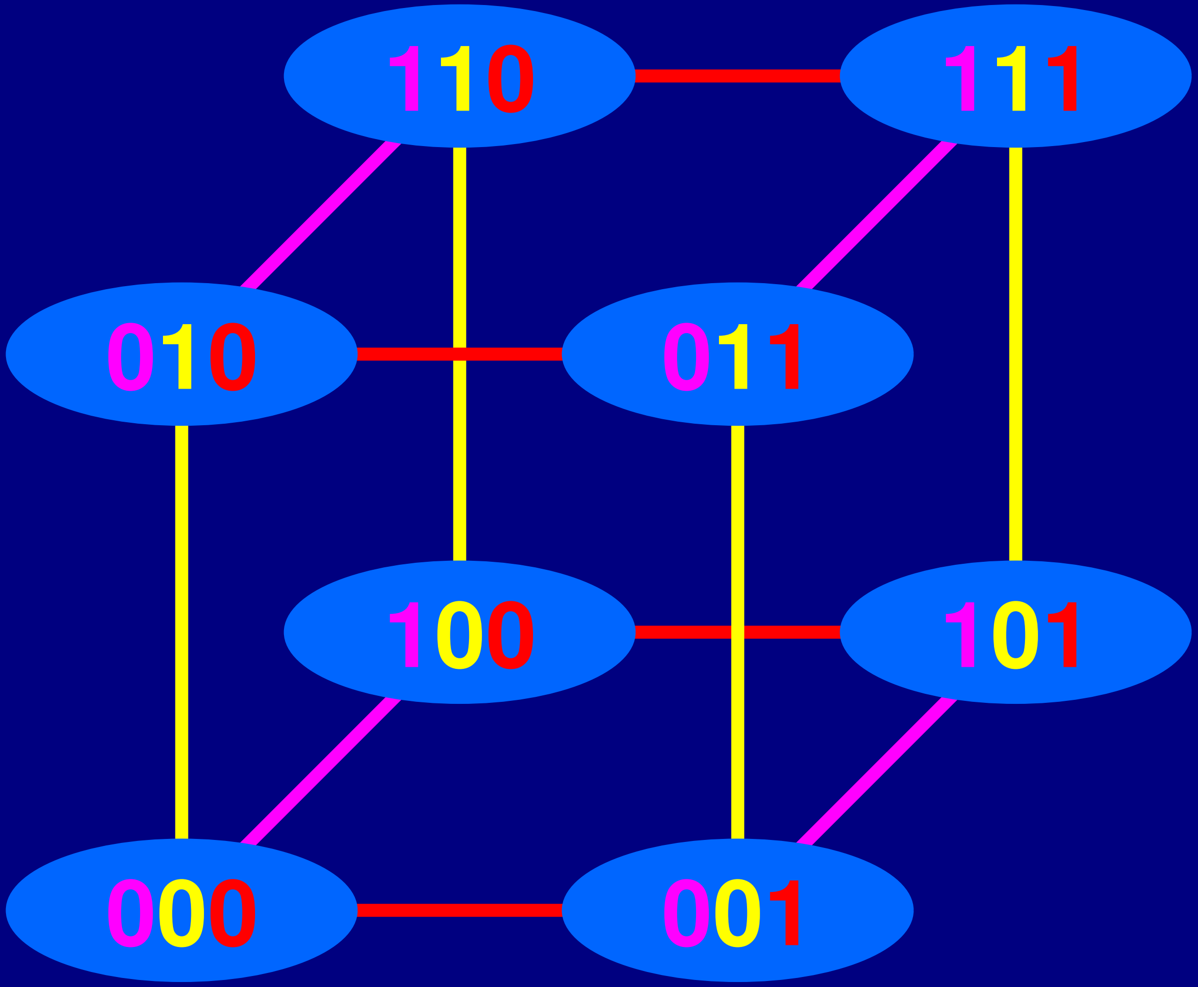
1

Link

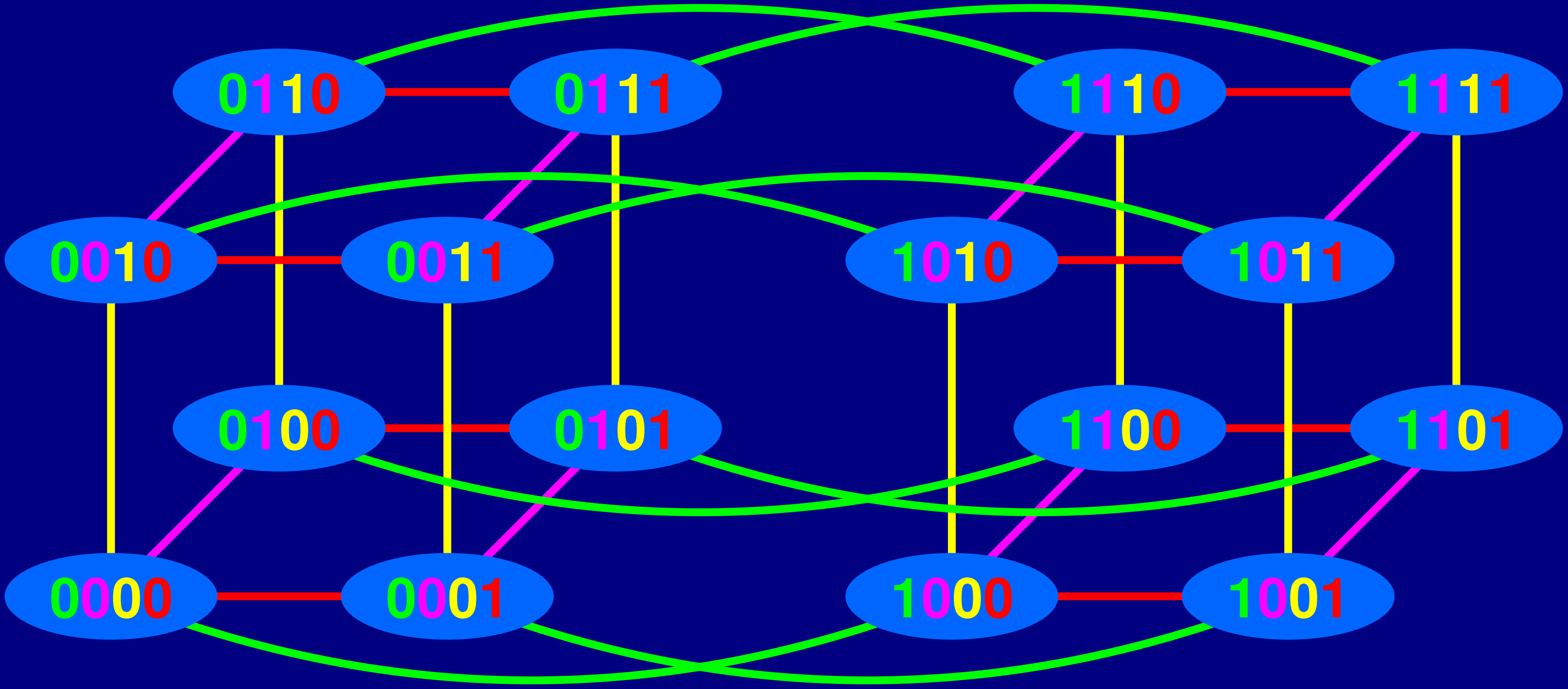
2-Cube



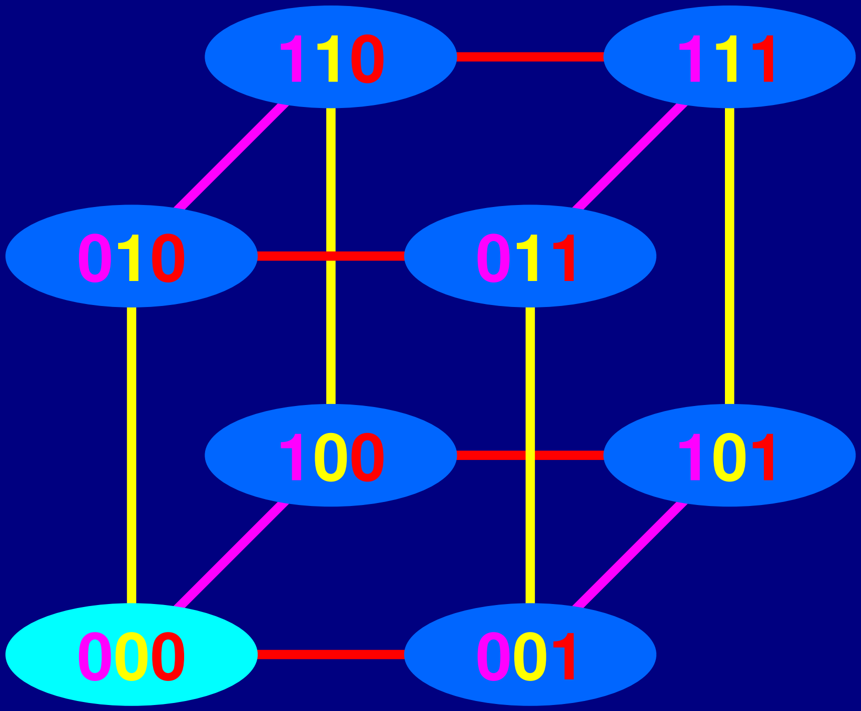
3-Cube



4-Cube


0110
0111
1110
1111
0010
0011
1010
1011
0100
0101
1100
1101
0000
0001
1000
1001

Average Distance of n-Cube



Source node: 000

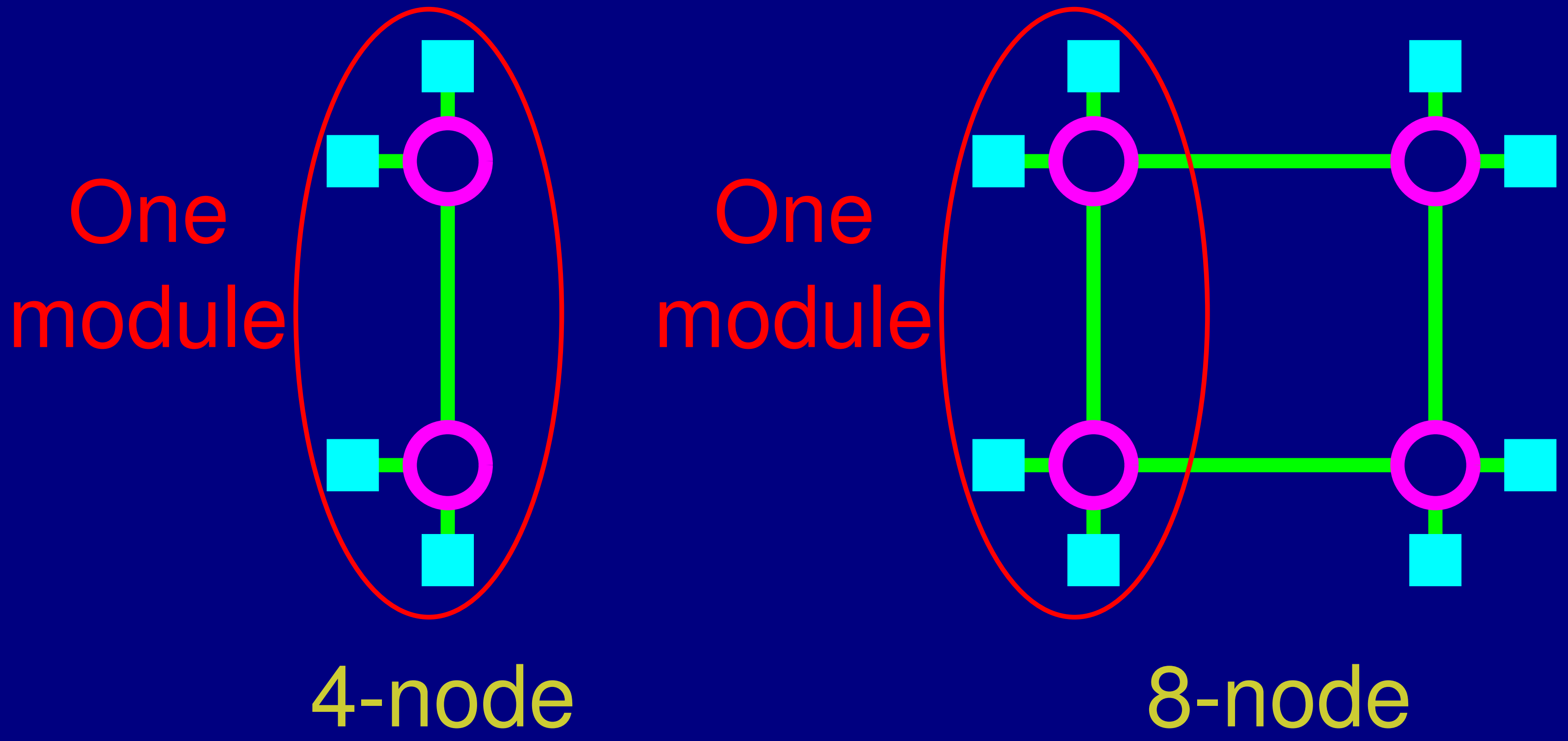
Distance	Number of nodes			
0	1	000		
1	3	001	010	100
2	3	011	101	110
3	1	111		

$$D = \left(\sum_{i=0}^{n} \binom{n}{i} \times i\right)/2^n = (n \times 2^{n-1})/2^n = \frac{n}{2}$$

Topological Properties of n-Cube

- Degree: n
- Diameter (maximum distance): n
- Average distance: $n/2$
- Bisection width: $2^n/2 = 2^{n-1}$
- Number of Links: $2^n n$
- Cost (Degree × Diameter): n^2

SGI Origin2000 — 1D/2D (8/16 CPUs)



One module

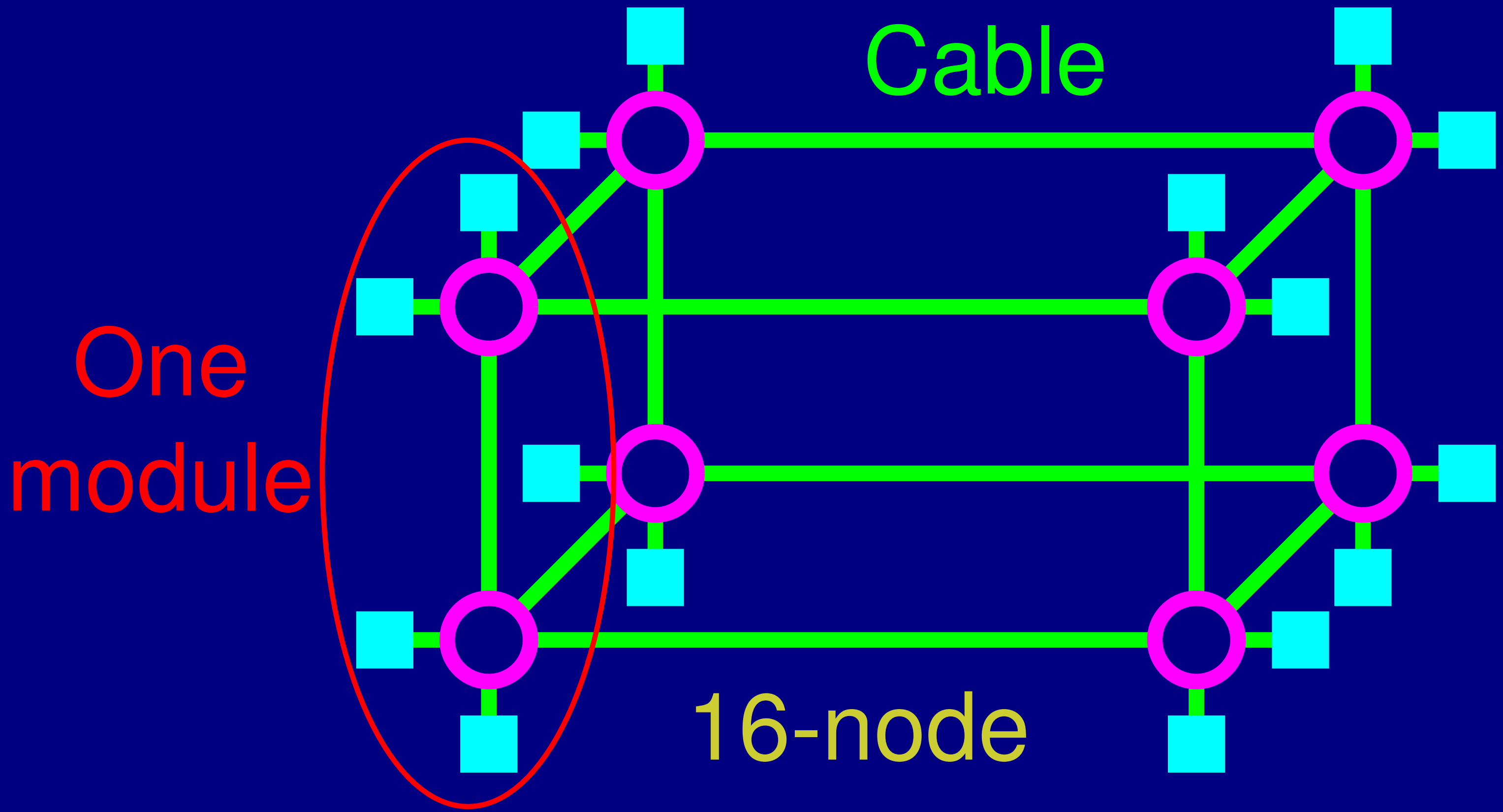
One module

4-node

8-node

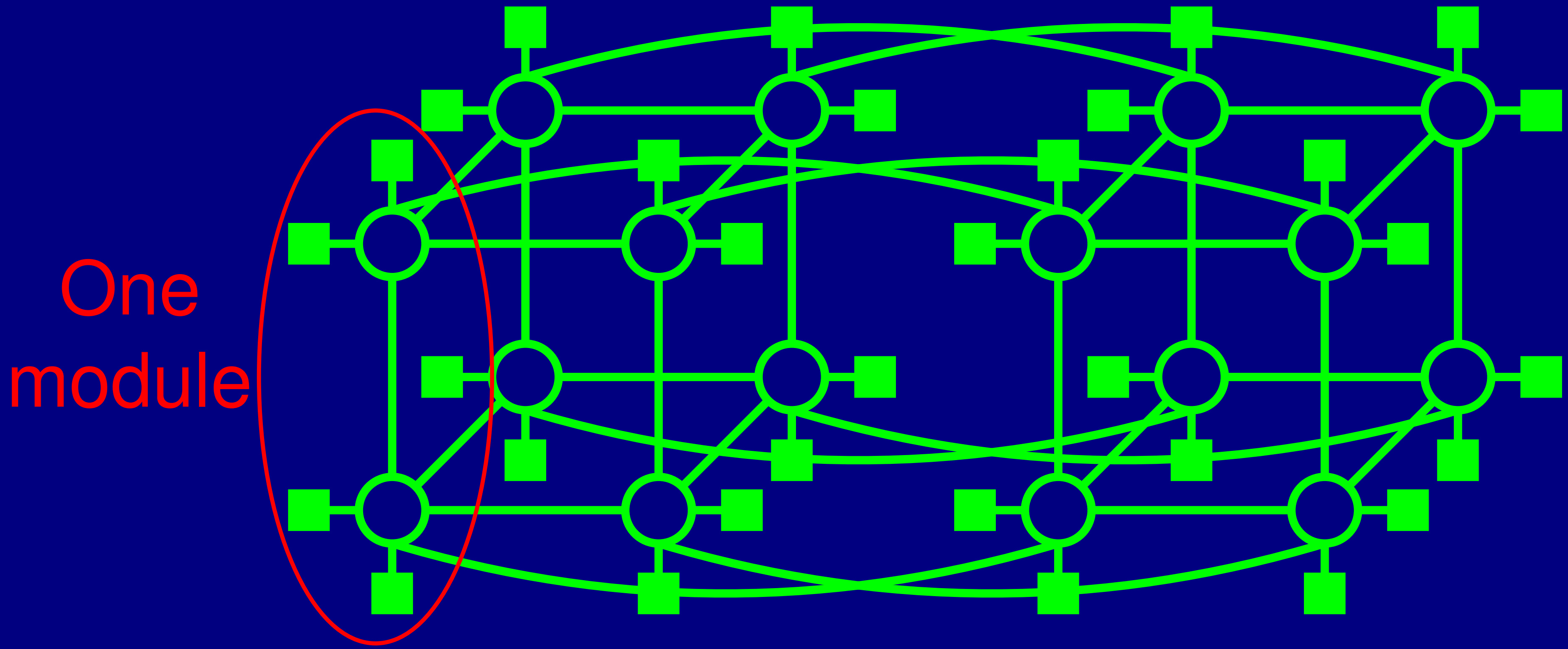
○ Router ■ Node board (2 CPUs)

SGI Origin2000 — 3D (32 CPUs)



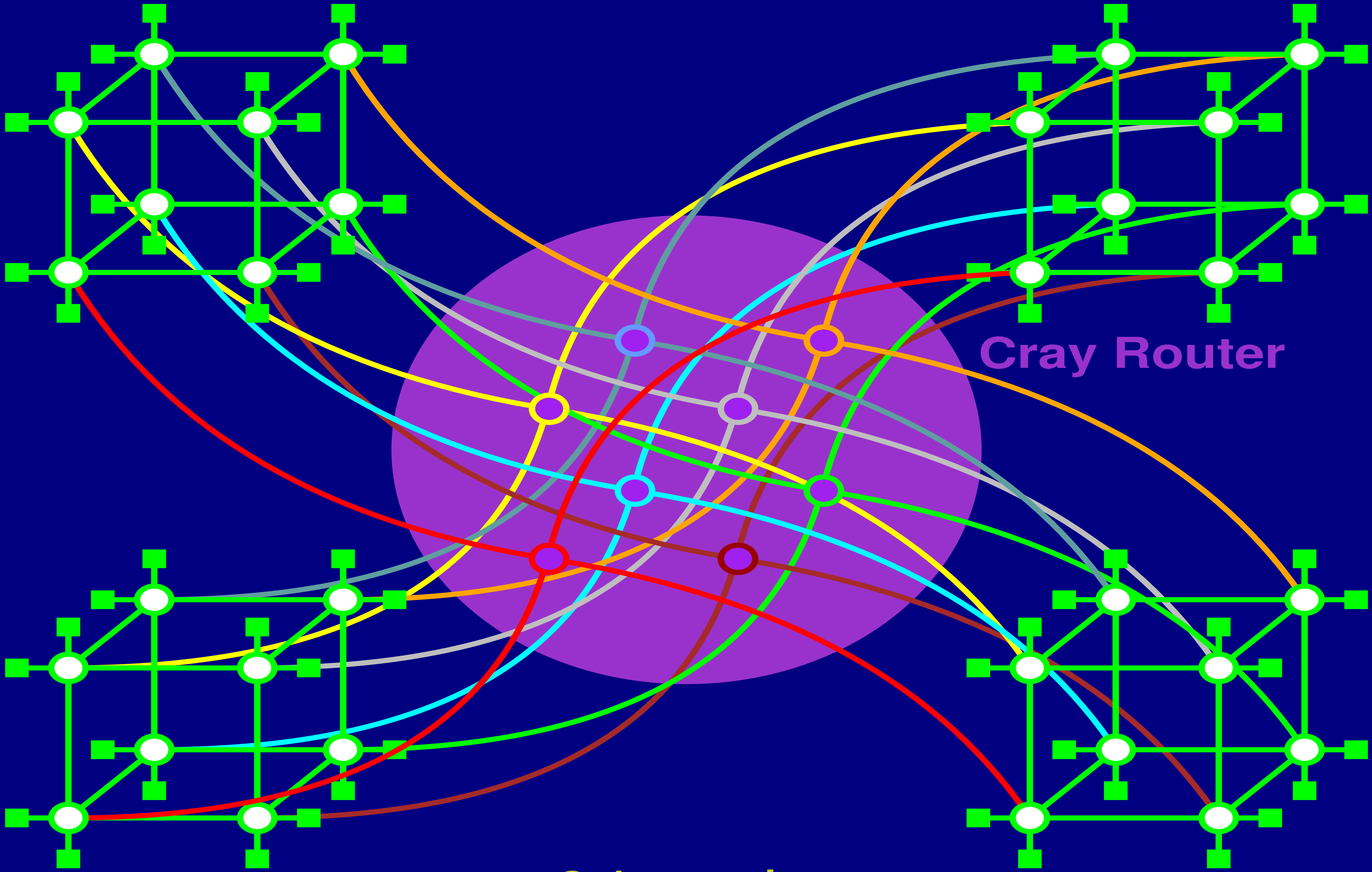
○ Router ■ Node board (2 CPUs)

SGI Origin2000 — 4D (64 CPUs)



32-node

SGI Origin2000 — 5D (128 CPUs)

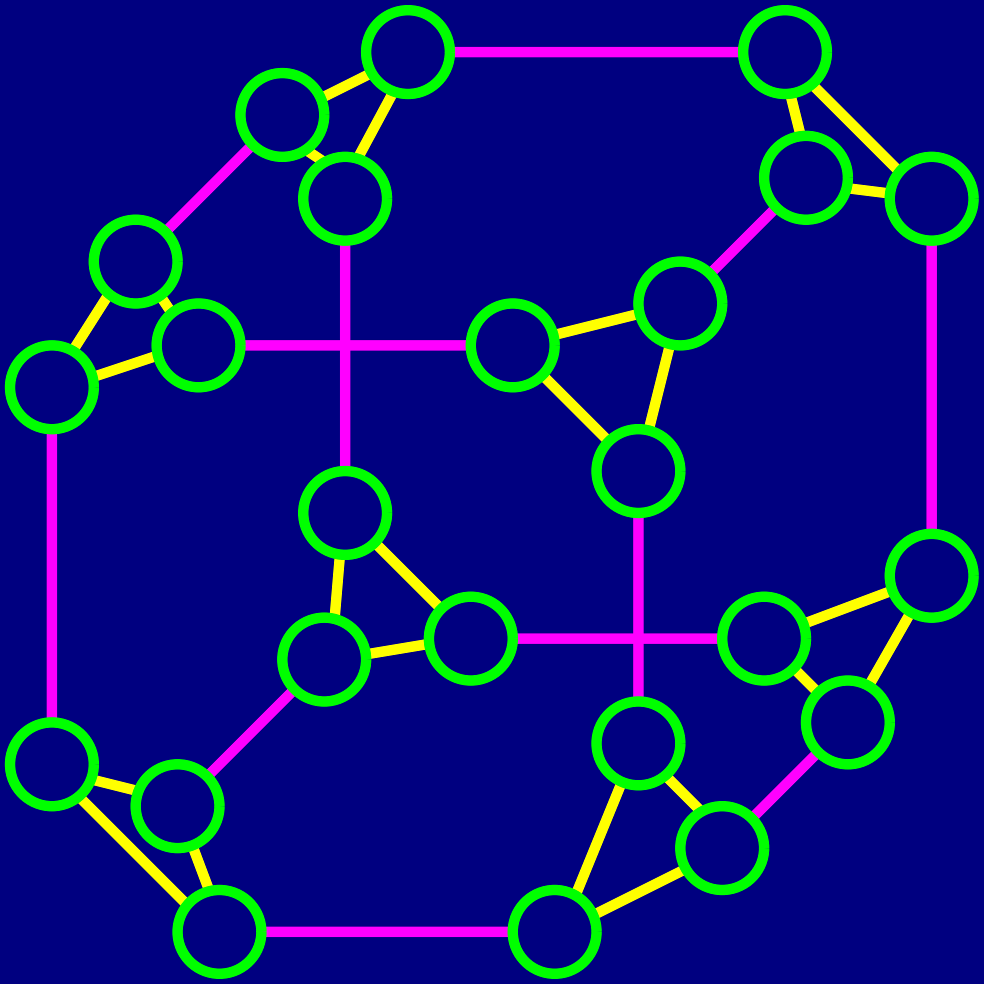


64-node

The Major Drawback of Hypercube

- The number of communication links for each node increases with the increase in the total number of nodes in the system.
    - $n = log_2 N$
        - n: The number of links per node
        - N: The number of nodes in system
    - In order to connect more nodes with the fixed number of links, SGI Origin 2000 uses a special router to link multiple hypercubes.
        - Cray Router
            - Does not connect to CPU boards
- Low degree alternatives to hypercube are needed.

Cube-Connected Cycles (CCC)



Fixed degree (3)

Module: cycle

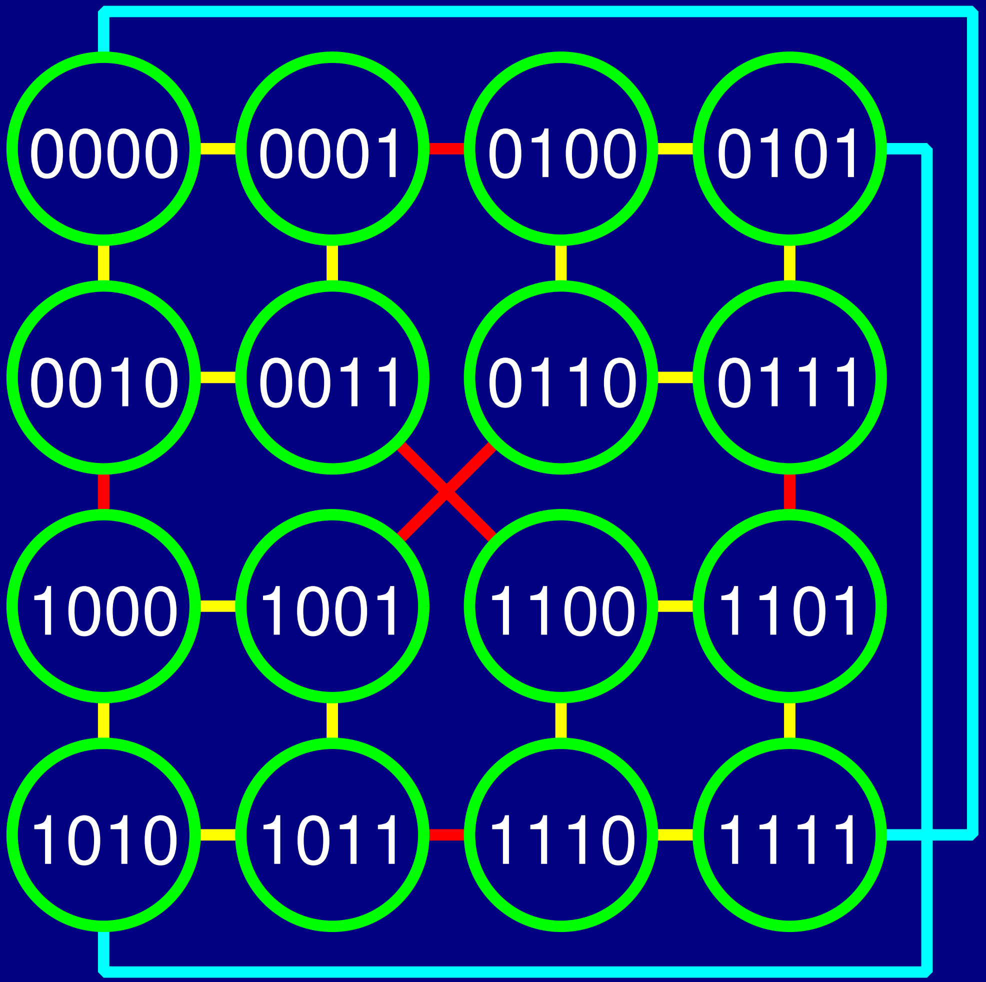
Long diameter

Hierarchical Cubic Network (HCN)

- The node set of the HCN(n) is $\{(X, Y)\}$:
  - X and Y are binary sequences of length n.
- Each node (X, Y) is adjacent to

1. $(X, Y^{(k)})$ for all $1 \leq k \leq n$,
   - where $Y^{(k)}$differs from Y at the kth bit position,
2. (Y, X) if $X \neq Y$, and
3. $(\overline{X}, \overline{Y})$ if $X = Y$,
   - where $\overline{X}$ and $\overline{Y}$ are the bitwise complements of X and Y, respectively.

Hierarchical Cubic Network (HCN)



$N = 2^{2n-2}$

Shorter diameter

Complex

Hypercube properties are lost

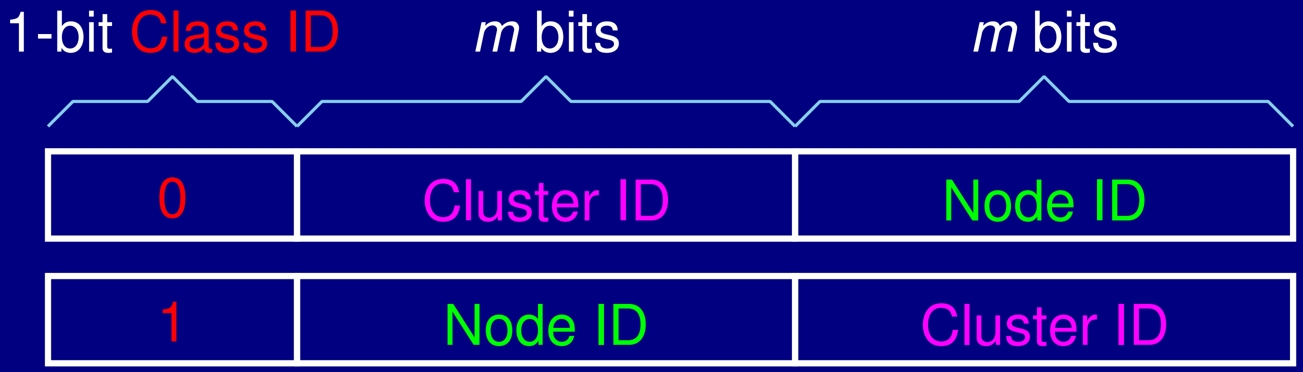
Section II

Dual-Cube

Dual-Cube Interconnection Network

- Can connect $N = 2^{2n-1}$ nodes
- Keeps the main properties of hypercube
- Simple routing algorithm
- Is Hamiltonian
- Performs collective communications efficiently
- Low communication cost for matrix multiplication
- Easy to build disjoint paths
- Maximum length of fault-free cycle embedding
- Efficient fault-tolerant routing

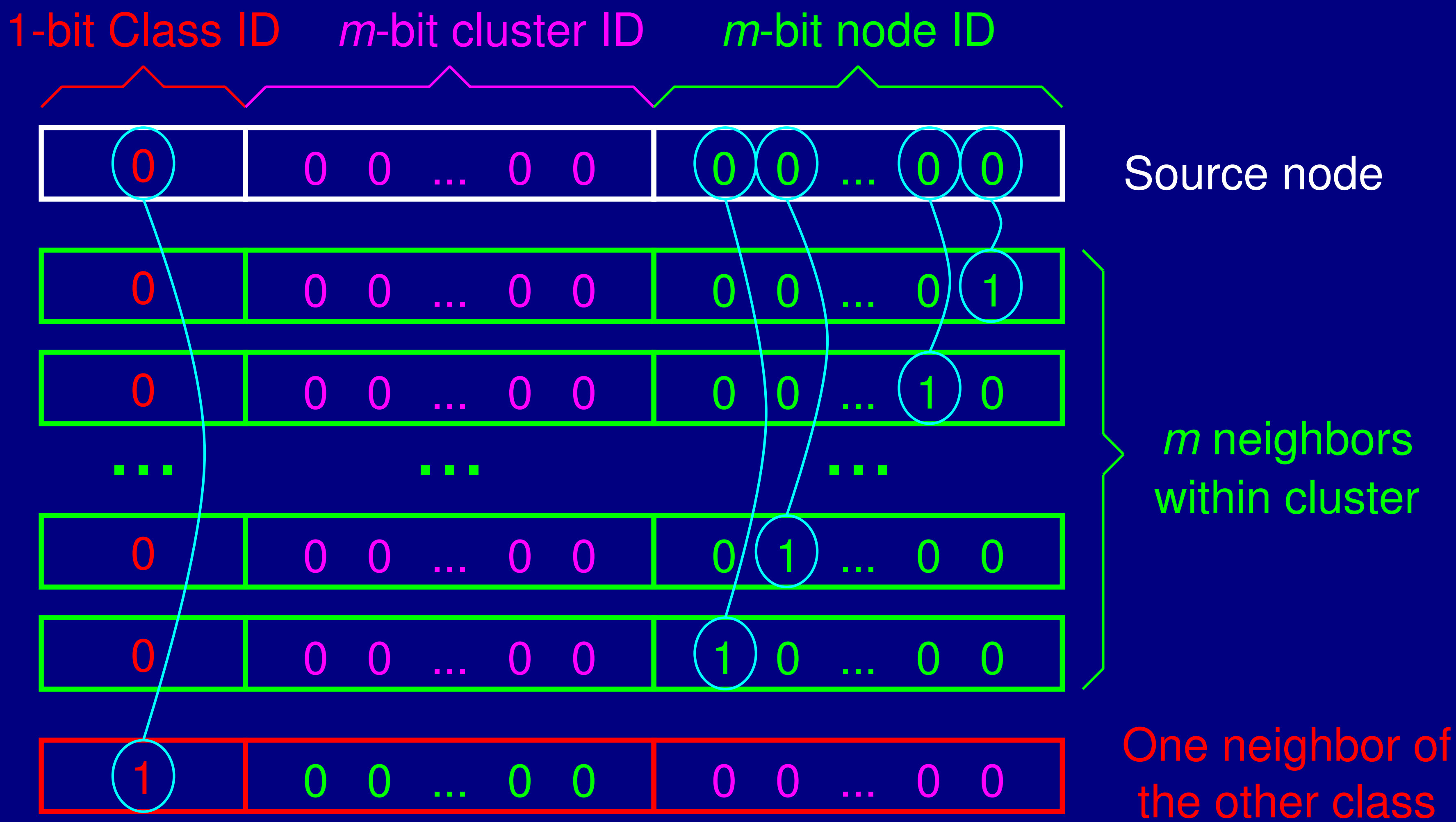
Dual-Cube: DC(m)



1-bit Class ID	m bits	m bits
0	Cluster ID	Node ID
1	Node ID	Cluster ID

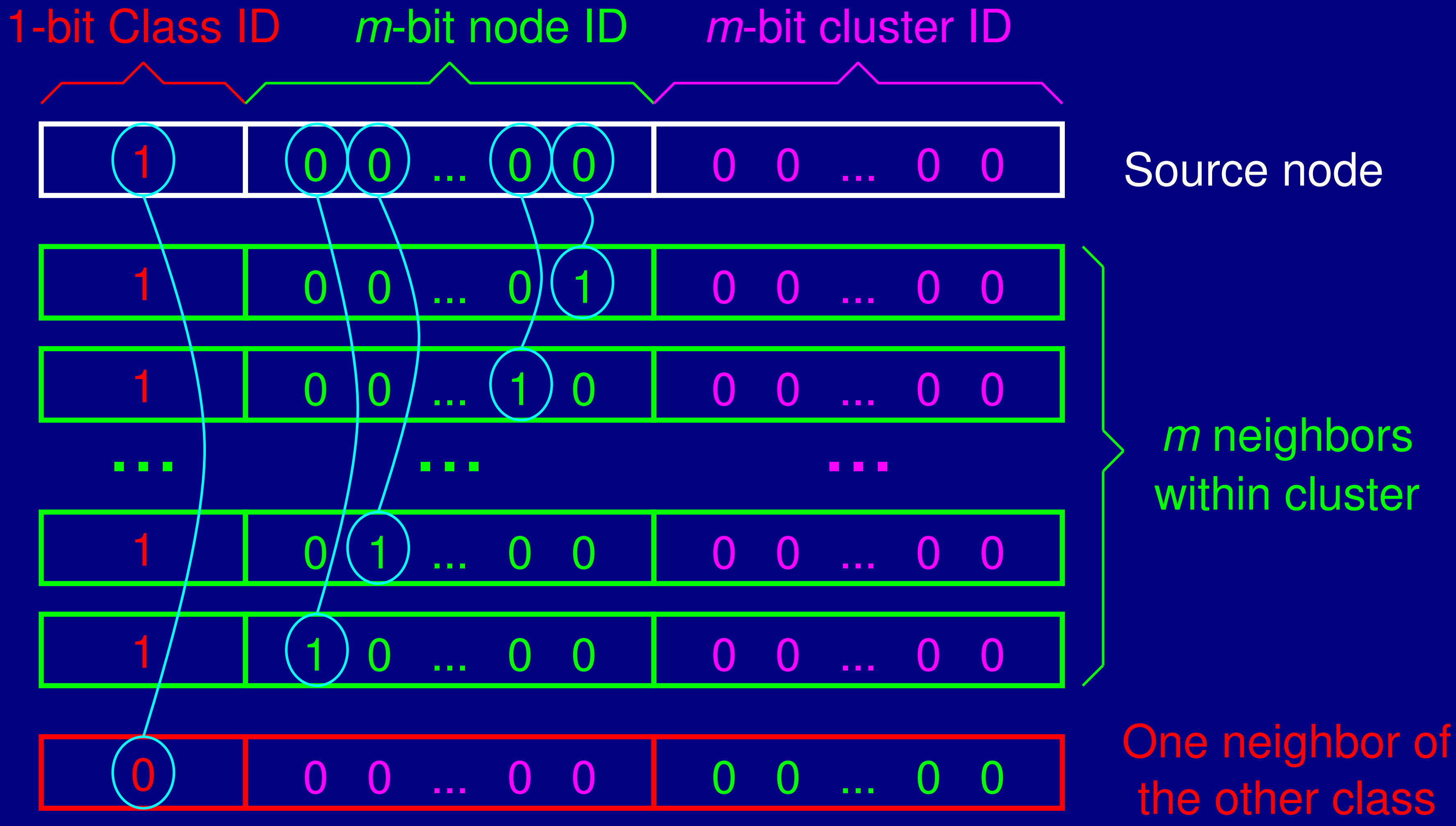
- Each node has $(m + 1)$ links:
  - The m links in node ID builds a *cluster* (m-cube).
  - One link in class ID connects to a node in a cluster of the other class.
  - No links in cluster ID.
- A DC(m) can connect 2^{2m+1} nodes.

Dual-Cube: Neighbors of 000...0000...00



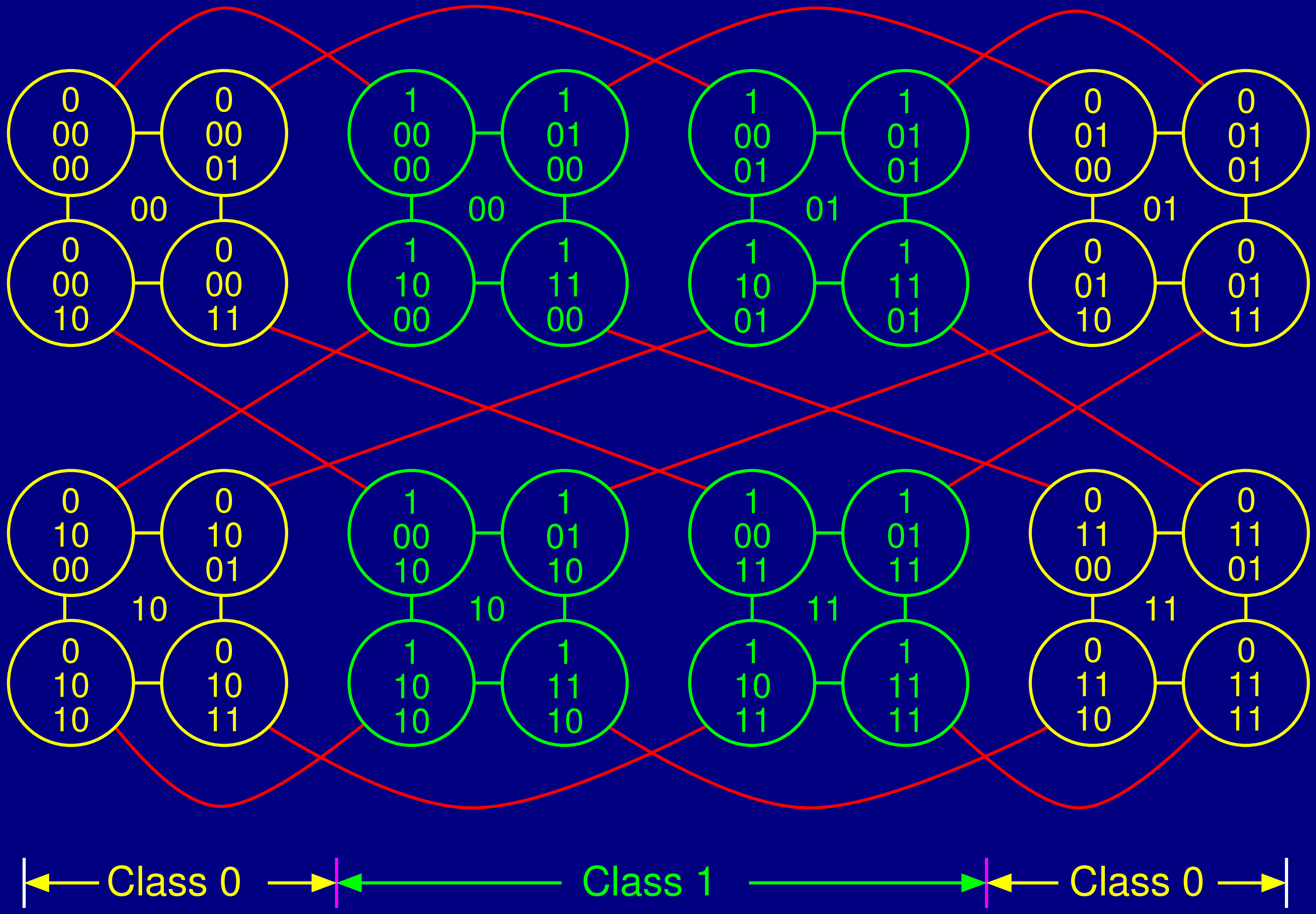
1-bit Class ID	*m*-bit cluster ID	*m*-bit node ID	
0	0 0 ... 0 0	0 0 ... 0 0	Source node
0	0 0 ... 0 0	0 0 ... 0 1	*m* neighbors within cluster
0	0 0 ... 0 0	0 0 ... 1 0	
...	...	...	
0	0 0 ... 0 0	0 1 ... 0 0	
0	0 0 ... 0 0	1 0 ... 0 0	
1	0 0 ... 0 0	0 0 ... 0 0	One neighbor of the other class

Dual-Cube: Neighbors of 100...0000...00

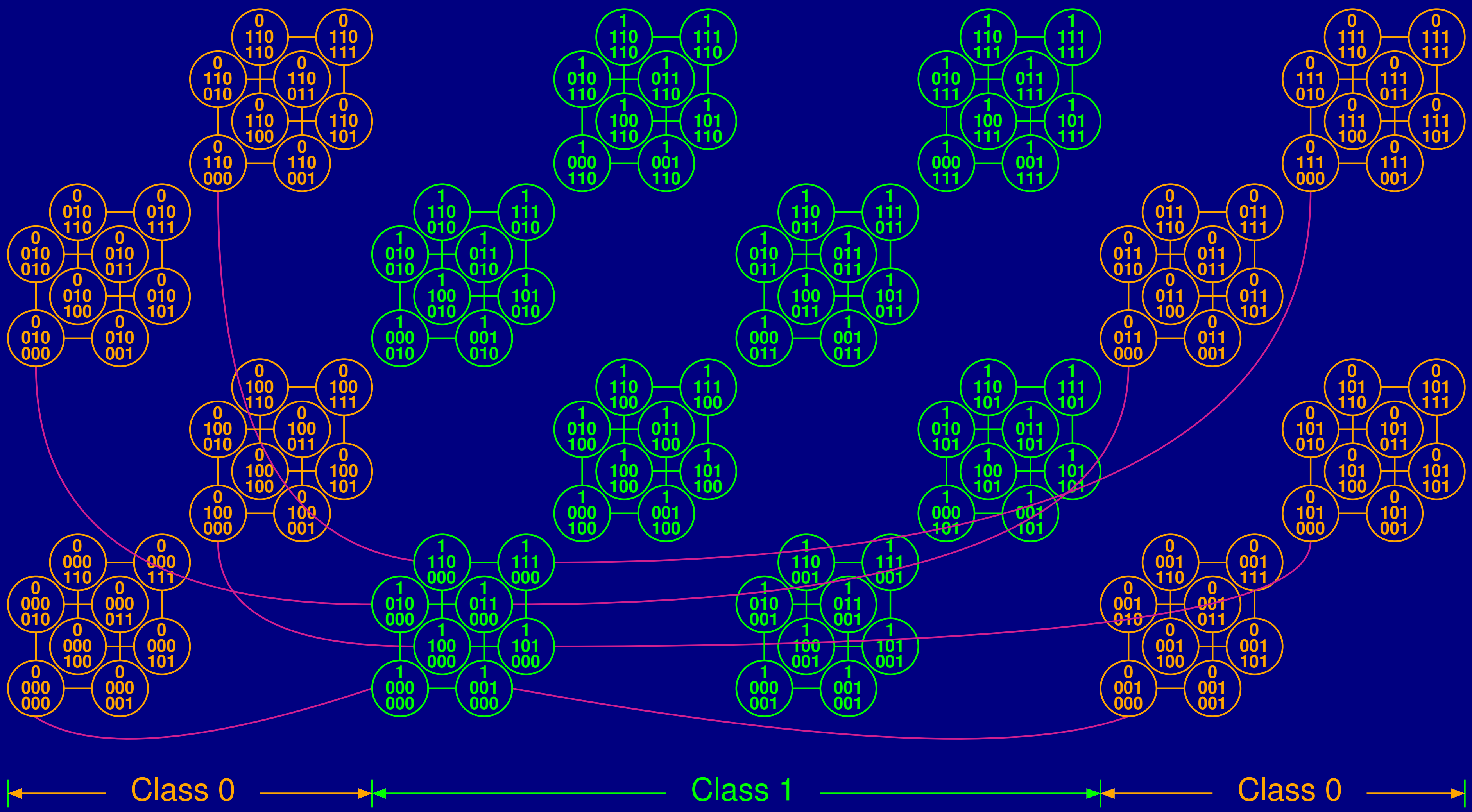


1-bit Class ID	*m*-bit node ID	*m*-bit cluster ID	
1	0 0 ... 0 0	0 0 ... 0 0	Source node
1	0 0 ... 0 1	0 0 ... 0 0	*m* neighbors within cluster
1	0 0 ... 1 0	0 0 ... 0 0	
...	...	...	
1	0 1 ... 0 0	0 0 ... 0 0	
1	1 0 ... 0 0	0 0 ... 0 0	
0	0 0 ... 0 0	0 0 ... 0 0	One neighbor of the other class

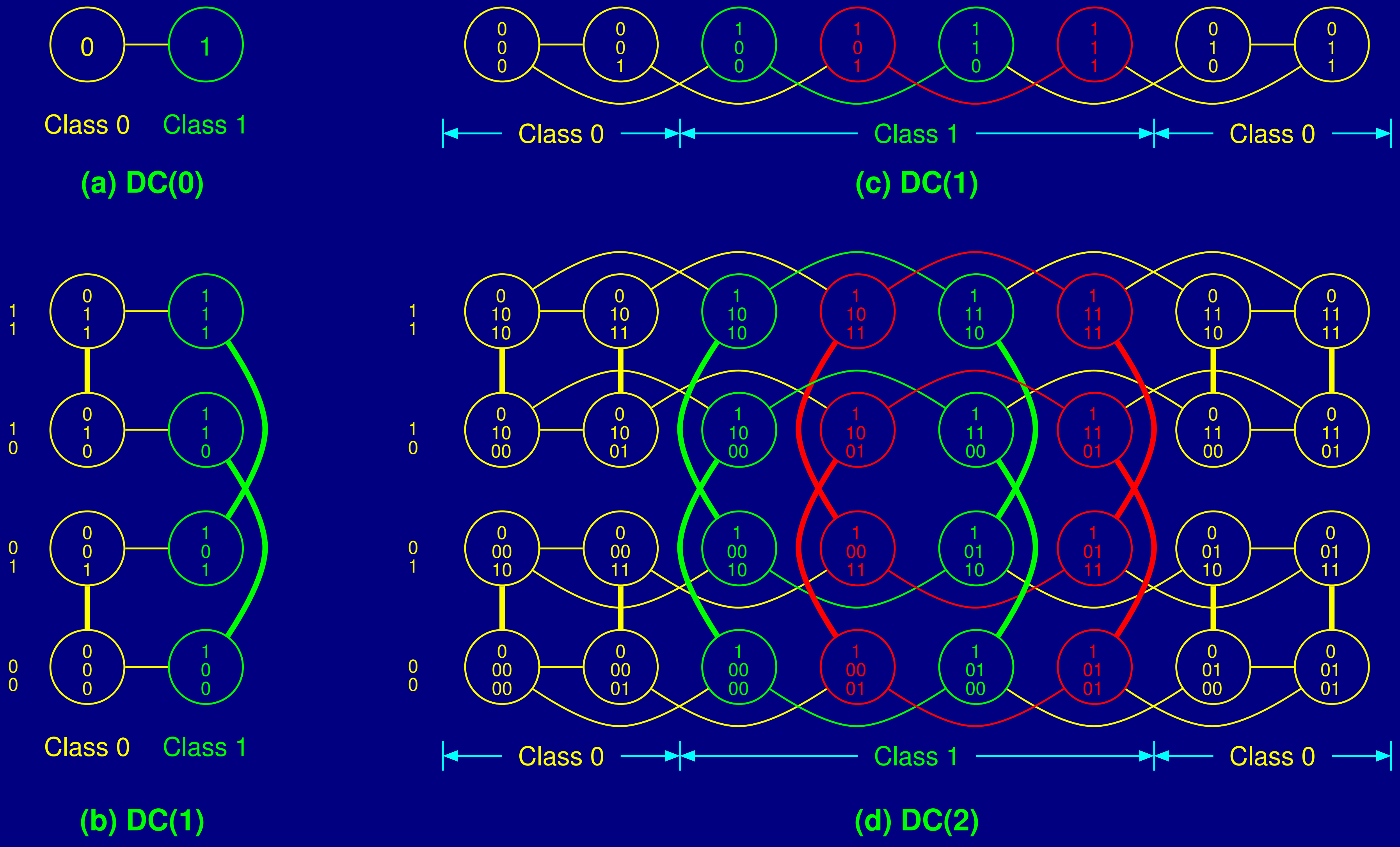
Dual-Cube: DC(2)



Dual-Cube: DC(3)



Dual-Cube: Recursive Construction



(a) DC(0)

(c) DC(1)

(b) DC(1)

(d) DC(2)

DC(m): Routing

$m = 4$ Same cluster	Different classes	Same class
s = 0 0000 0000	s = 0 0000 0000	s = 0 0000 0000
0 0000 0001	0 0000 0001	0 0000 0001
0 0000 0011	0 0000 0011	0 0000 0011
0 0000 0111	0 0000 0111	0 0000 0111
t = 0 0000 1111	0 0000 1111	0 0000 1111
	1 0000 1111	1 0000 1111
	1 0001 1111	1 0001 1111
	1 0011 1111	1 0011 1111
	1 0111 1111	1 0111 1111
	t = 1 1111 1111	1 1111 1111
		t = 0 1111 1111

DC(m): Diameter

$m = 4$

s = 0 0000 0000
0 0000 0001
0 0000 0011
0 0000 0111
0 0000 1111
1 0000 1111
1 0001 1111
1 0011 1111
1 0111 1111
t = 1 1111 1111

Distance = $2m + 1$

s = 0 0000 0000
0 0000 0001
0 0000 0011
0 0000 0111
0 0000 1111
1 0000 1111
1 0001 1111
1 0011 1111
1 0111 1111
1 1111 1111
t = 0 1111 1111

Distance = $2m + 2$

DC(m): Average Distance

- Suppose source node $s = 0$.
- For destination node $t \in$ class 1: Total distance D_1
  $= (m/2 + 1 + m/2) \times 2^m \times 2^m$
  $= (m + 1) \times 2^{2m}$
- For destination node $t \in$ class 0: Total distance D_2
  $= (m/2 + 2 + m/2) \times 2^m \times 2^m - 2 \times 2^m$
  $= (m + 2) \times 2^{2m} - 2 \times 2^m$
  - Where -2×2^m is for t and s in the same cluster
- Average distance
  $= (D_1 + D_2)/2^{2m+1} = (m + 1) + 1/2 - 1/2^m$

DC(m): Properties

- Degree: $m + 1$
- Diameter (maximum distance): $2m + 2$
- Average distance: $(m + 1) + 1/2 - 1/2^m$
- Bisection width: 2^{2m-1}
- Number of Links: $2^{2m+1}(m + 1)$
- Cost (Degree × Diameter): $2(m + 1)^2$

Properties: Dual-Cube vs Hypercube

Same number of nodes: $N = 2^n = 2^{2m+1}$, i.e., $m = (n-1)/2$

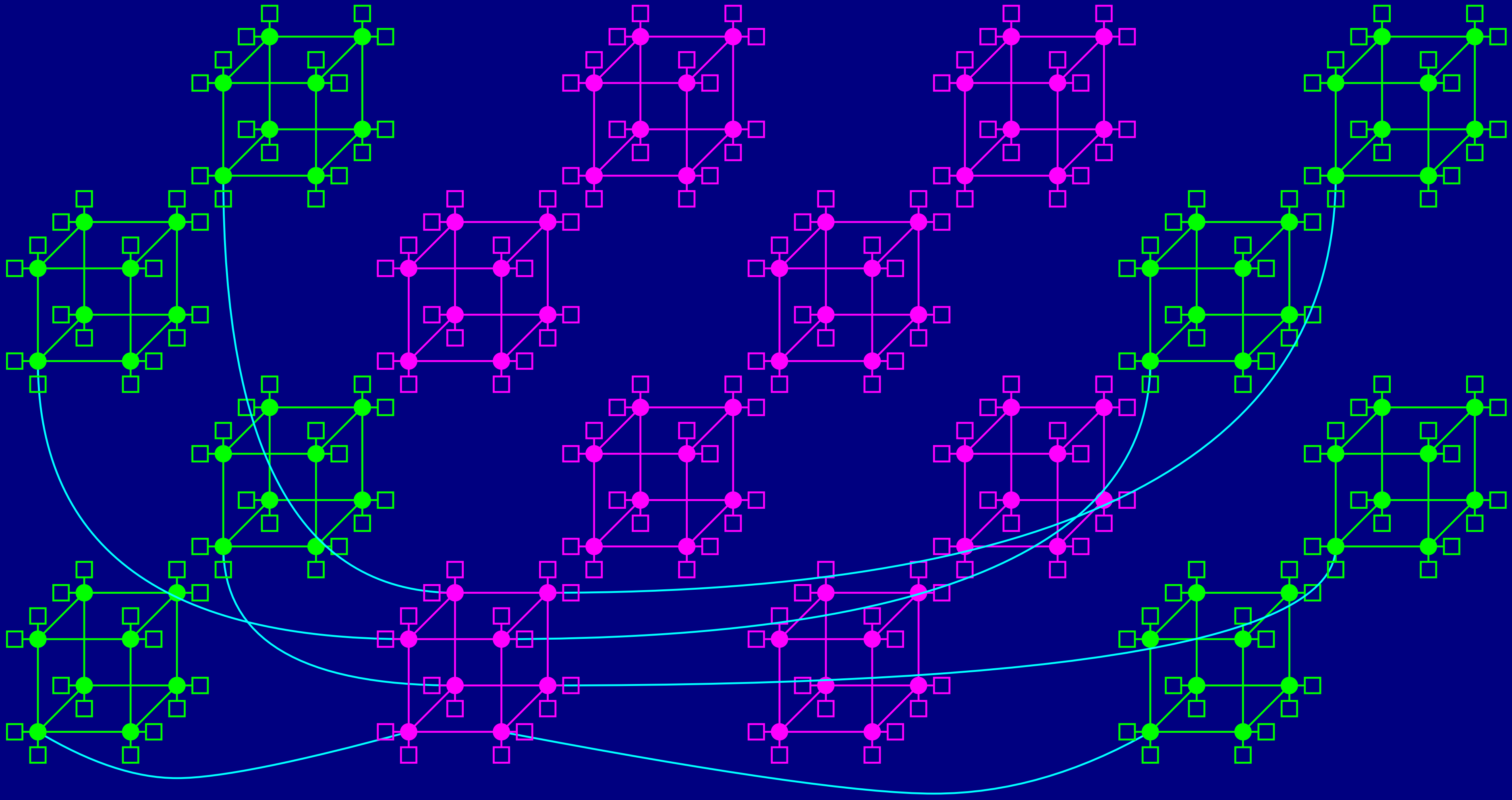
Network	Degree	Diameter	Cost
Hypercube	n	n	n^2
Dual-Cube	$(n+1)/2$	$n+1$	$(n+1)^2/2$

Network	Average distance	Bisection	# of links
Hypercube	$n/2$	$2^n/2$	$2^n n/2$
Dual-Cube	$n/2+1-1/2^{(n-1)/2}$	$2^n/4$	$2^n(n+1)/4$

Apply Dual-Cube to SGI Origin2000

- SGI Origin2000
  - 128 CPUs + Cray Router
- Apply dual-cube to SGI Origin2000
  - No need to use Cray Router
  - Only change the cable connection manner
  - Router: 6 links
    - 2 links for connecting node boards (4 CPUs)
    - 4 links for interconnects
  - $m = 3$, $N = 2^{2m+1} = 128$ routers
  - # CPUs $= 128 \times 4 = 512$

Apply Dual-Cube to SGI Origin2000



Section III

Collective Communications

Models of Communication

- Collective communication is the key issue in parallel computers.
- Based on the number of sending and receiving processors, these communications can be classified into one-to-all and all-to-all.
- The nature of the messages to be sent can be classified as personalized or broadcast.

	broadcasting	personalized
one-to-all	✔	✔
all-to-all	✔	✔

Collective Communication

- Assumptions
  - Communication links are bidirectional:
    - Two directly-connected processors can send messages of size m (in words) to each other simultaneously in time $t_s + t_w m$,
      - where t_s is the message setup time,
      - and t_w is the per-word transfer time.
  - A processor can send a message on only one of its links at a time.
  - Similarly, it can receive a message on only one link at a time.

Store-and-Forward and Cut-Through

- Store-and-forward routing:
  - A message traversing multiple hops is completely received at an intermediate hop before being forwarded to the next hop.
- Cut-through routing:
  - The messages are divided into basic units (flits).
  - The destination address should be fit in a flit.
  - An intermediate hop begins forwarding the message as soon as the hop has read the destination address.
  - All flits are sent on the same path, in sequence.

Store-and-Forward and Cut-Through

- Store-and-forward routing
  - Sending a single message containing m words takes $t_s + t_w m l$ time,
    - where l is the number of links traversed by the message.
  - Upper bound for hypercube: $t_s + t_w m \log p$,
    - where p is the number of nodes in the network.
- Cut-through routing
  - A message can be sent directly from source to a destination l links away in time $t_s + t_w m + t_h l$,
    - where t_h is the per-hop time.

Subsection III.1

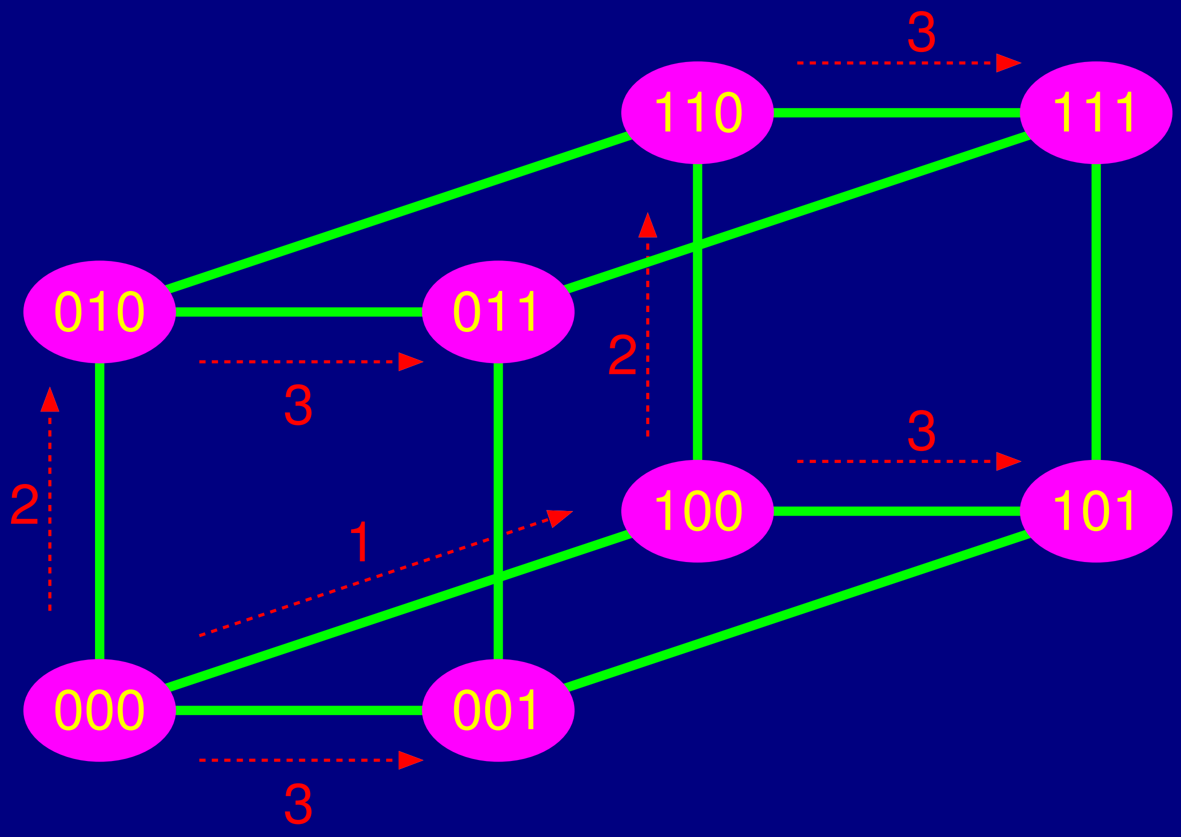
Collective Communication in Hypercube

One-to-All Broadcast

- A single node sends identical data to all other nodes.
- Initially, only the source process has the data of size *m* that needs to be broadcast.
- At the termination of the procedure, there are *p* copies of the initial data — one belonging to each process.
- Use store-and-forwarding routing.
- Show the algorithm for hypercube.
- Node 0 broadcasts a message.

One-to-All Broadcast in Hypercube

Store-and-forwarding routing:



One-to-All Broadcast in Hypercube

- There is a total of log p communication steps.
- Each step takes $t_s + t_w m$ time.
- Therefore, the total time taken by the procedure on a p-node hypercube is

$$T_{one\_to\_all\_b} = (t_s + t_w m) \log p$$

- The pseudocode of the procedure is shown in the next page.
- The procedure is executed at all nodes concurrently.

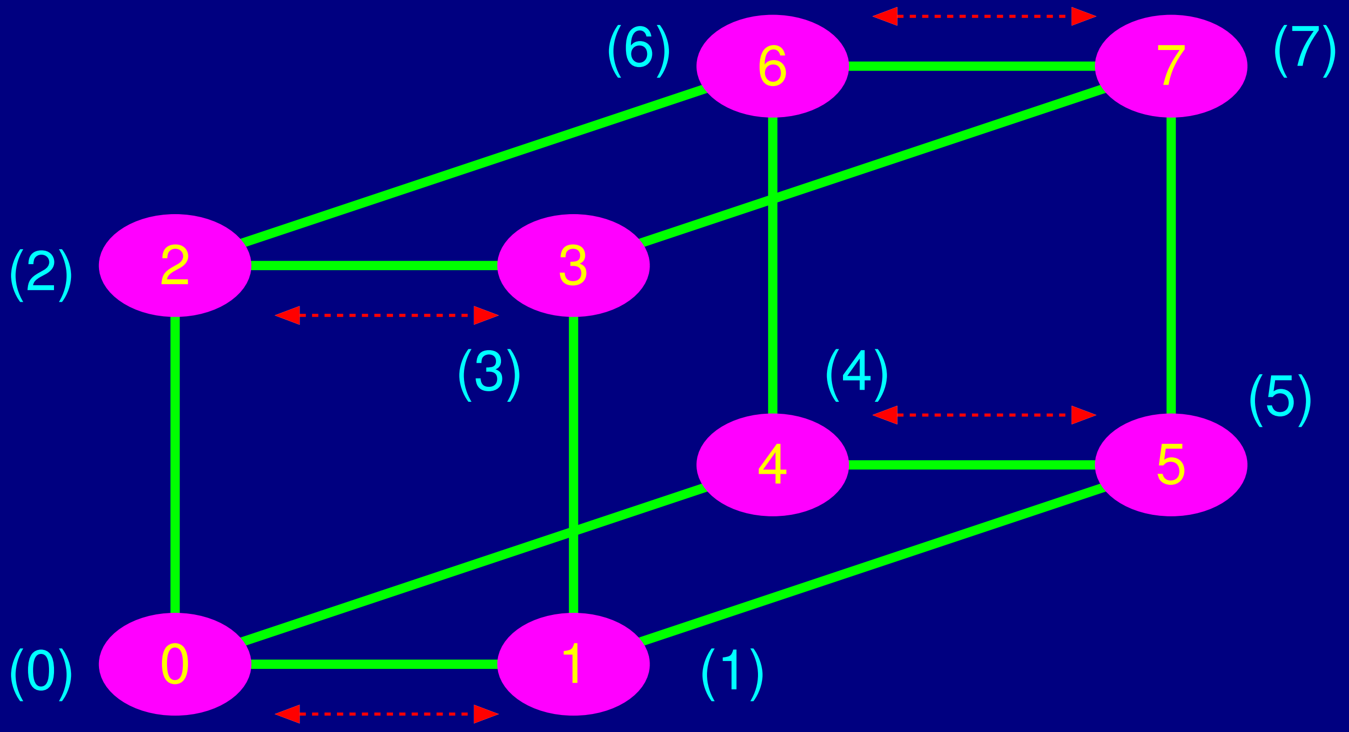
One-to-All Broadcast in Hypercube

```
procedure ONE_TO_ALL_BC_0($d$, $my\_id$, $X$)              /* Source: node 0 */
begin      /* One-to-all broadcast of a message $X$ from node 0 of a $d$-cube */
  $mask := 2^d - 1$;                                /* Set all $d$ bits of mask to 1 */
  for $i := d - 1$ downto 0 do                                     /* Outer loop */
    $mask := mask$ XOR $2^i$;                          /* Set bit $i$ of mask to 0 */
    if ($my\_id$ AND $mask$) $= 0$ then            /* If lower $i$ bits of $my\_id$ are 0 */
      if ($my\_id$ AND $2^i$) $= 0$ then
        $msg\_destination := my\_id$ XOR $2^i$;
        send $X$ to $msg\_destination$;
      else
        $msg\_source := my\_id$ XOR $2^i$;
        receive $X$ from $msg\_source$;
      endelse
    endif
  endfor
end
```

One-to-All Broadcast in Hypercube

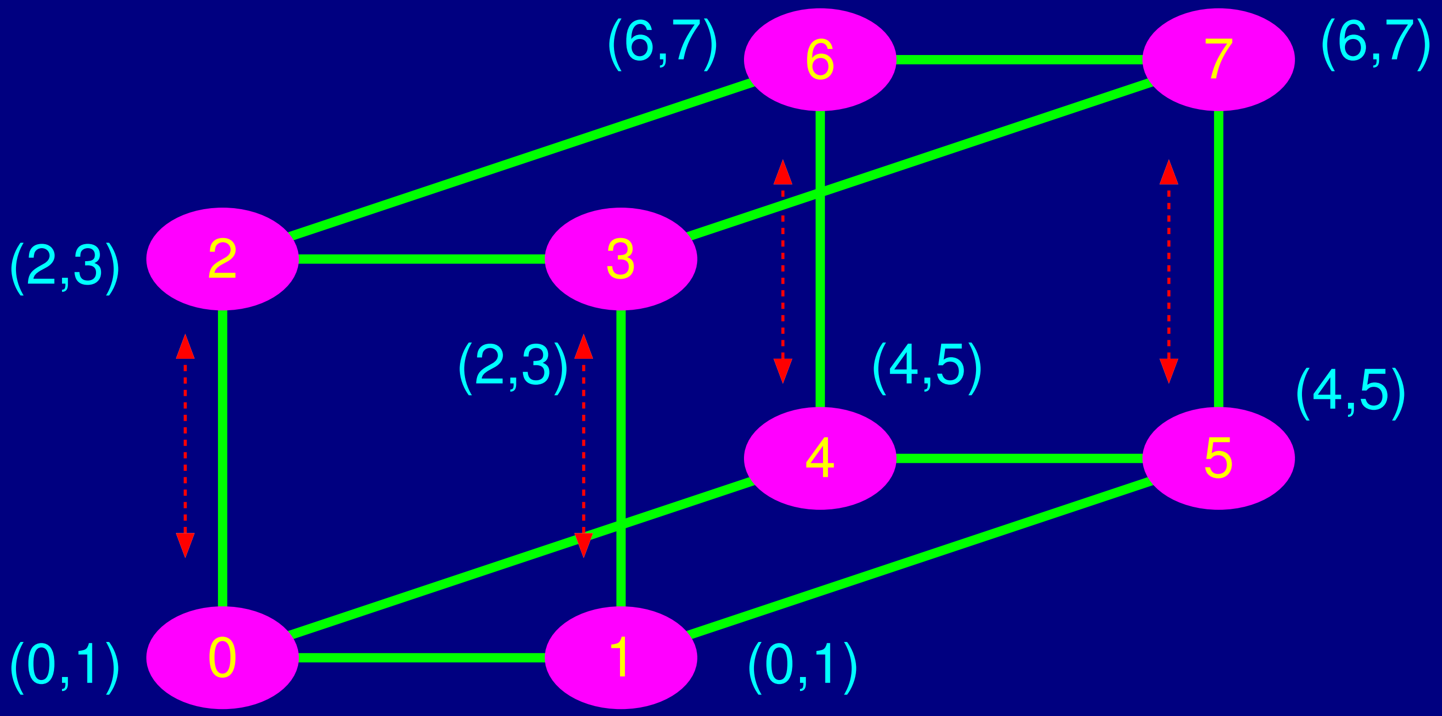
```
procedure ONE_TO_ALL_BC($d$, $my\_id$, $s$, $X$)                    /* Source: node $s$ */
begin $my\_virtual\_id = my\_id$ XOR $s$;
  $mask := 2^d - 1$;                                         /* Set all $d$ bits of mask to 1 */
  for $i := d - 1$ downto 0 do                                          /* Outer loop */
    $mask := mask$ XOR $2^i$;                                  /* Set bit $i$ of mask to 0 */
    if ($my\_virtual\_id$ AND $mask$) $= 0$ then              /* If lower $i$ bits are 0 */
      if ($my\_virtual\_id$ AND $2^i$) $= 0$ then
        $virtual\_destination := my\_virtual\_id$ XOR $2^i$;
        send $X$ to $virtual\_destination$;
      else
        $virtual\_source := my\_virtual\_id$ XOR $2^i$;
        receive $X$ from $virtual\_source$;
      endelse
    endif
  endfor
end
```

All-to-All Broadcast in Hypercube



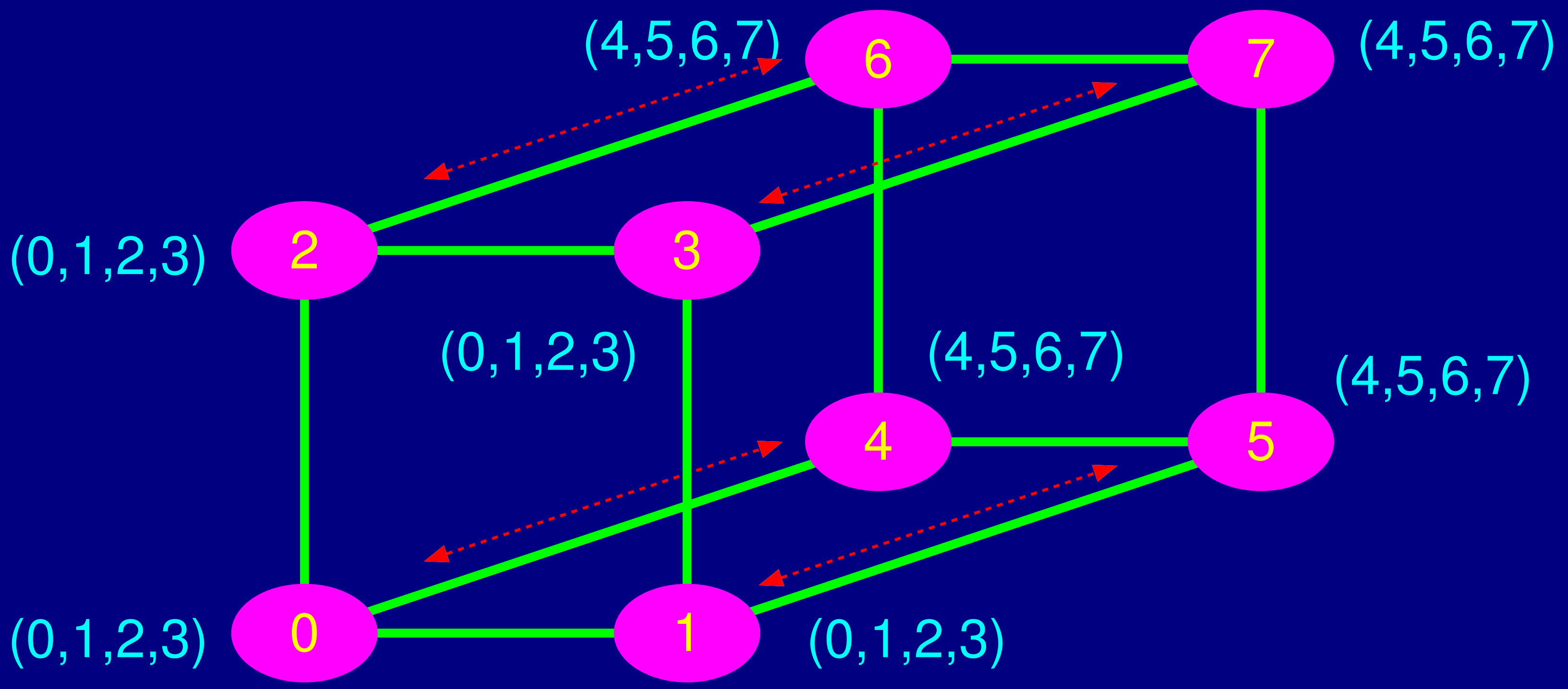
(a) Initial distribution of messages

All-to-All Broadcast in Hypercube



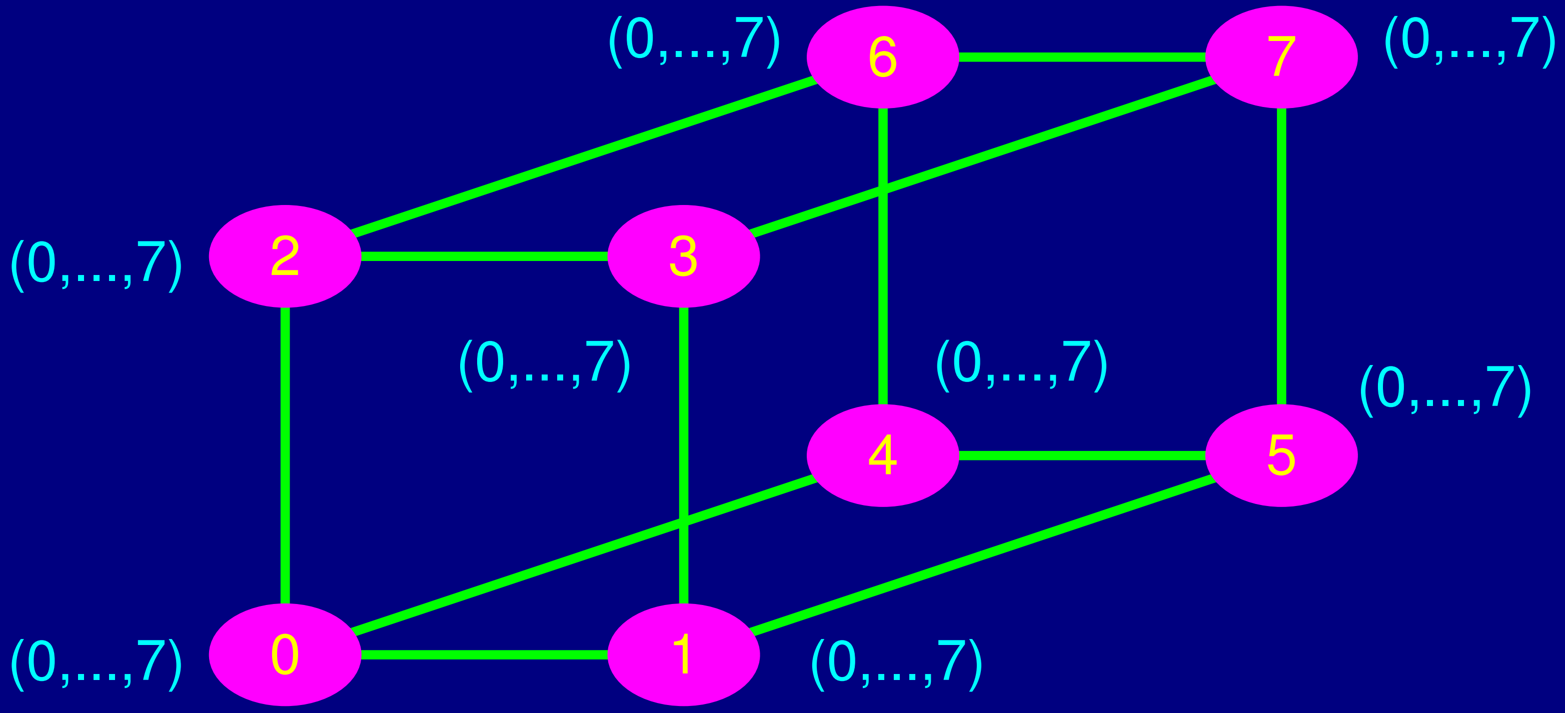
(b) Distribution before the second step

All-to-All Broadcast in Hypercube



(c) Distribution before the third step

All-to-All Broadcast in Hypercube



(d) Final distribution of messages

All-to-All Broadcast in Hypercube

- It takes $d = \log p$ steps ($i = 1, \ldots, d$).
- The size of the messages exchanged in ith step is $2^{i-1}m$.
- The time it takes a pair of nodes to send and receive from each other is $t_s + 2^{i-1}t_w m$.
- Hence, the time it takes to complete the entire procedure is

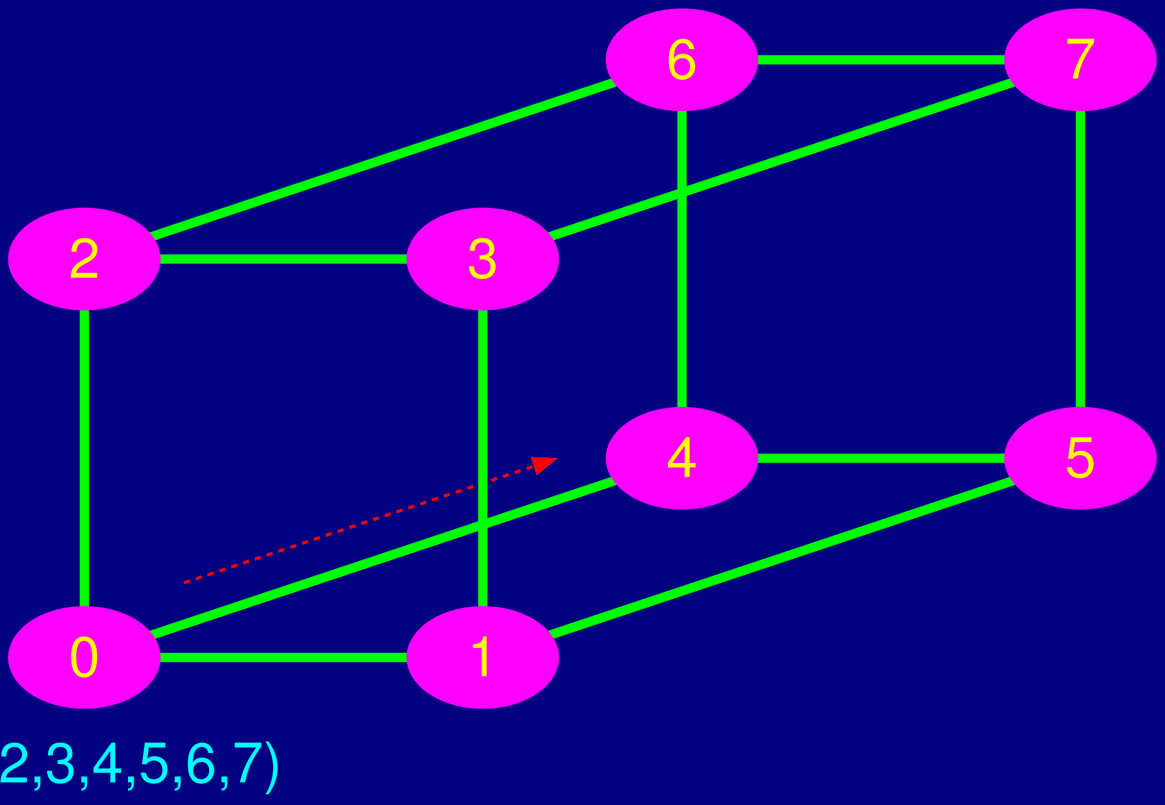
$$T_{all\_to\_all\_b} = \sum_{i=1}^{\log p}(t_s + 2^{i-1}t_w m)$$

$$= t_s \log p + t_w m(p-1)$$

All-to-All Broadcast in Hypercube

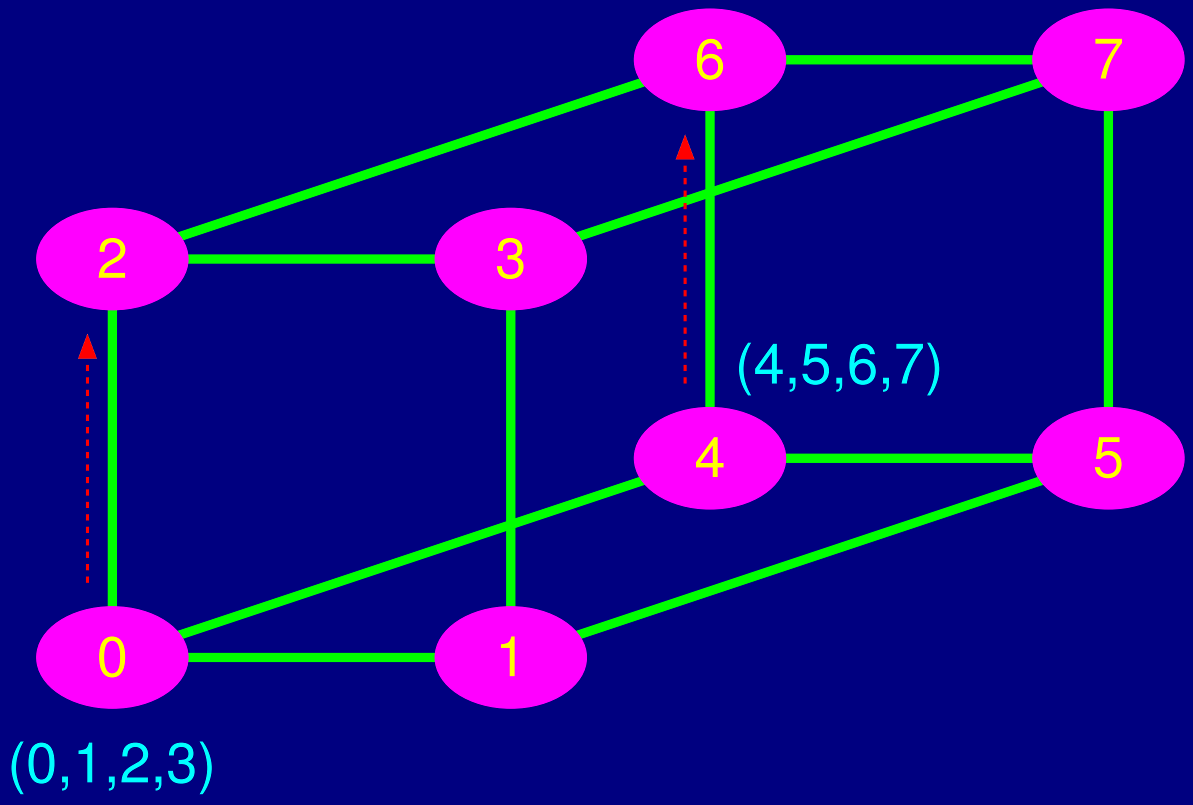
```
procedure ONE_TO_ALL_BC($d$, $my\_id$, $my\_msg$, $result$)
begin
    $result := my\_msg$;
    for $i := 0$ to $d - 1$ do
        $partner := my\_id$ XOR $2^i$;
        send $result$ to $partner$;
        receive $msg$ from $partner$;
        $result := result \cup msg$;
    endfor
end
```

One-to-All Personalized in Hypercube



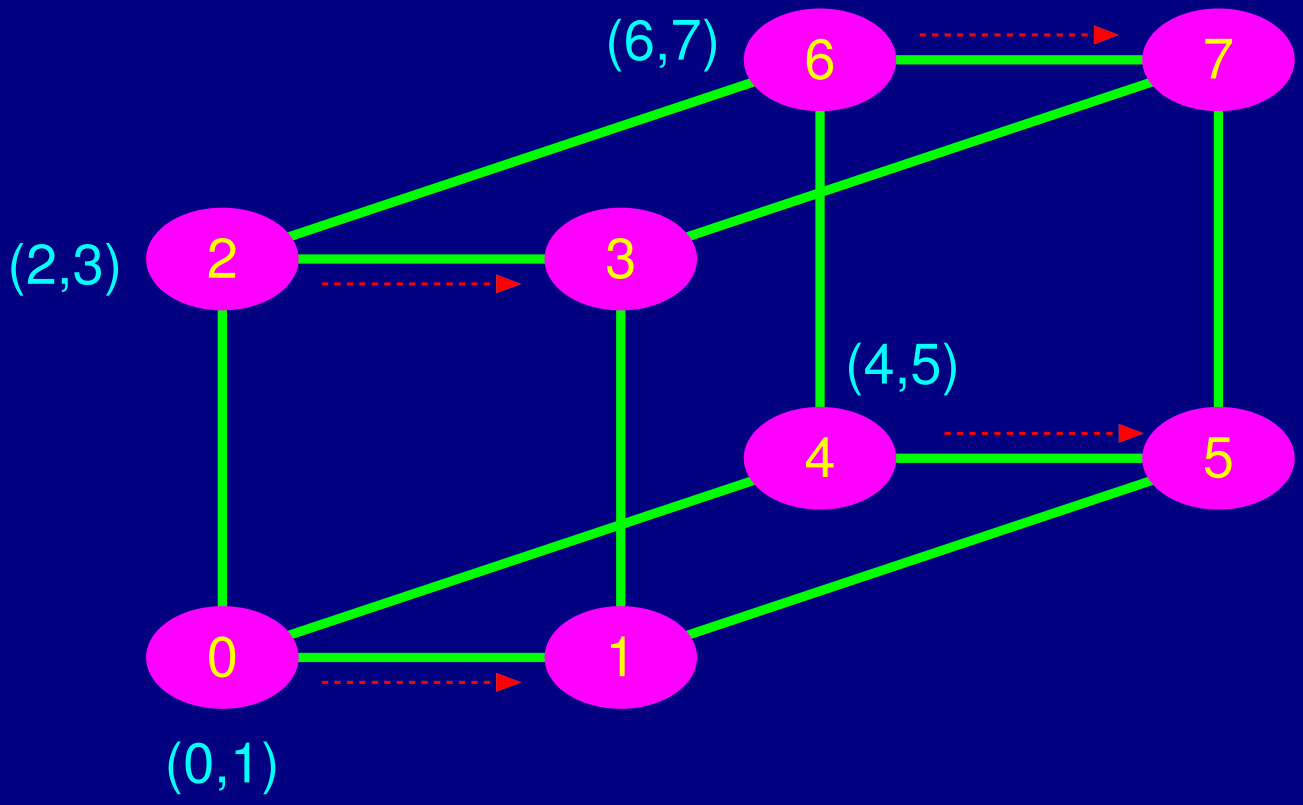
(a) Initial distribution of messages

One-to-All Personalized in Hypercube



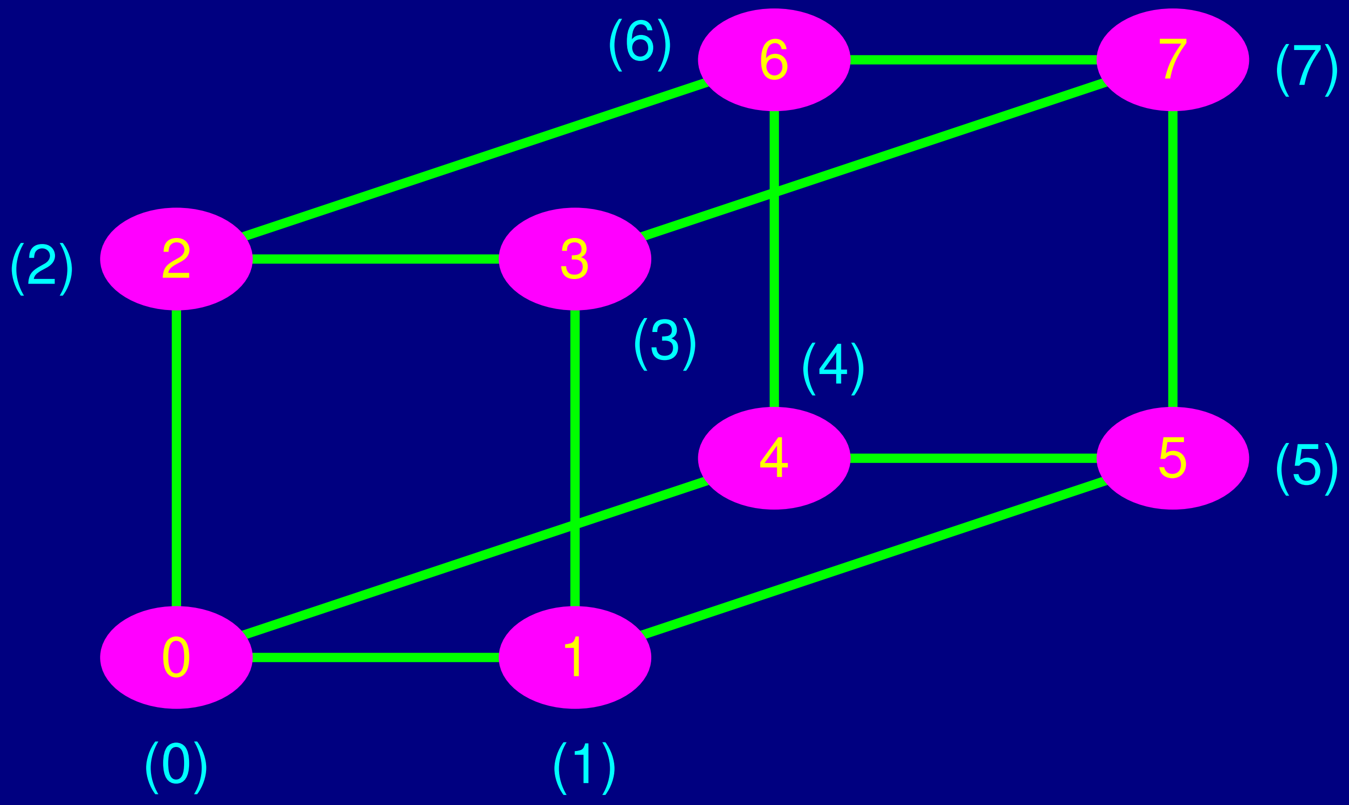
(b) Distribution before the second step

One-to-All Personalized in Hypercube



(c) Distribution before the third step

One-to-All Personalized in Hypercube



(d) Final distribution of messages

One-to-All Personalized in Hypercube

- It takes $d = \log p$ steps ($i = 1, \ldots, d$).
- The size of the messages exchanged in ith step is $2^{d-i}m$.
- The time it takes a pair of nodes to send and receive from each other is $t_s + 2^{d-i}t_w m$.
- Hence, the time it takes to complete the entire procedure is

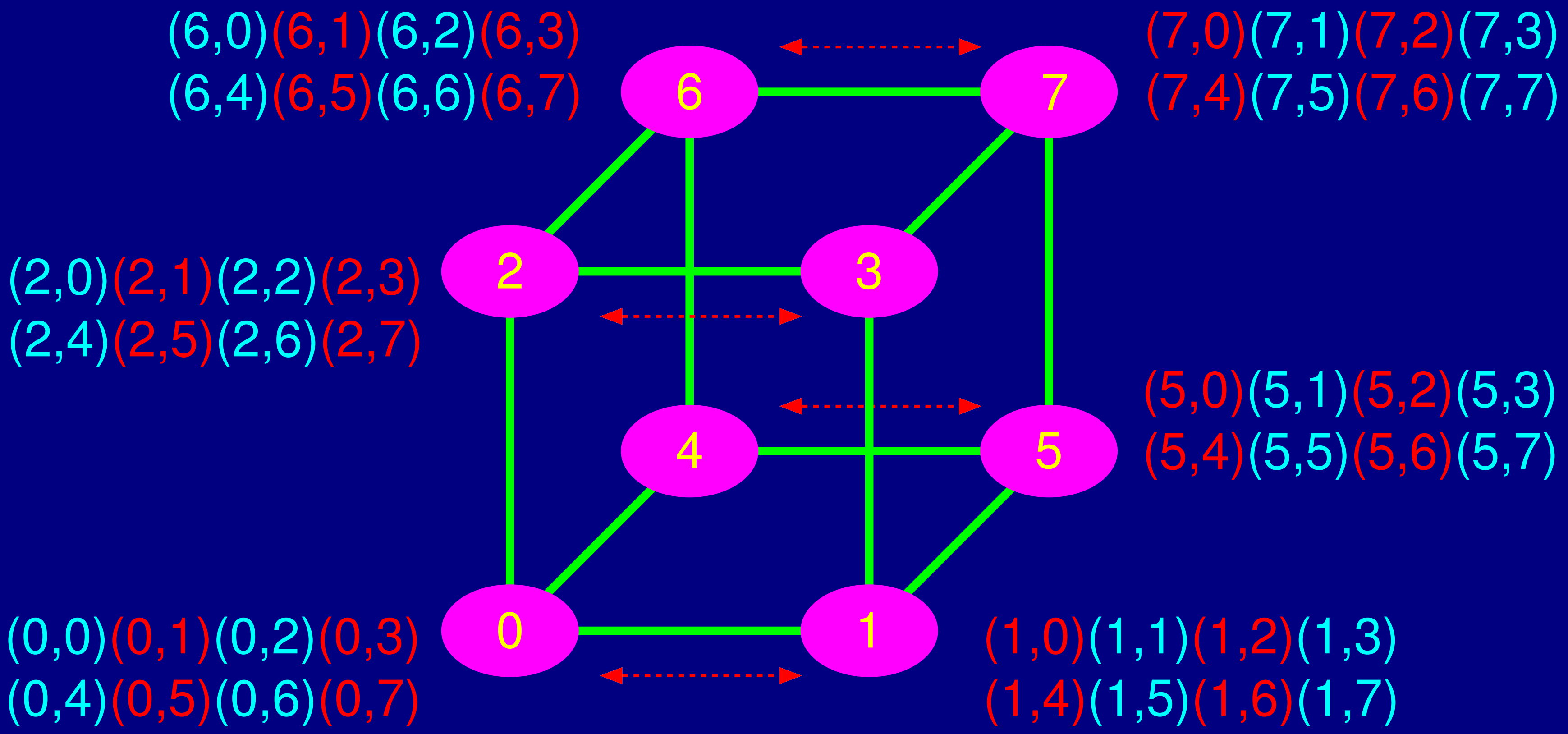
$$T_{one\_to\_all\_pers} = \sum_{i=1}^{\log p}(t_s + 2^{d-i}t_w m)$$

$$= t_s \log p + t_w m(p - 1)$$

All-to-All Personalized in Hypercube

- Each node has a distinct message of size *m* for every other node.
- This is unlike all-to-all broadcast, in which each node sends the same message to all other nodes.
- All-to-all personalized communication is also known as *total exchange*.
- Two versions:
  - SF,
  - CT.

All-to-All Personalized in Hypercube(SF)



(6,0)(6,1)(6,2)(6,3)
(6,4)(6,5)(6,6)(6,7)

(7,0)(7,1)(7,2)(7,3)
(7,4)(7,5)(7,6)(7,7)

(2,0)(2,1)(2,2)(2,3)
(2,4)(2,5)(2,6)(2,7)

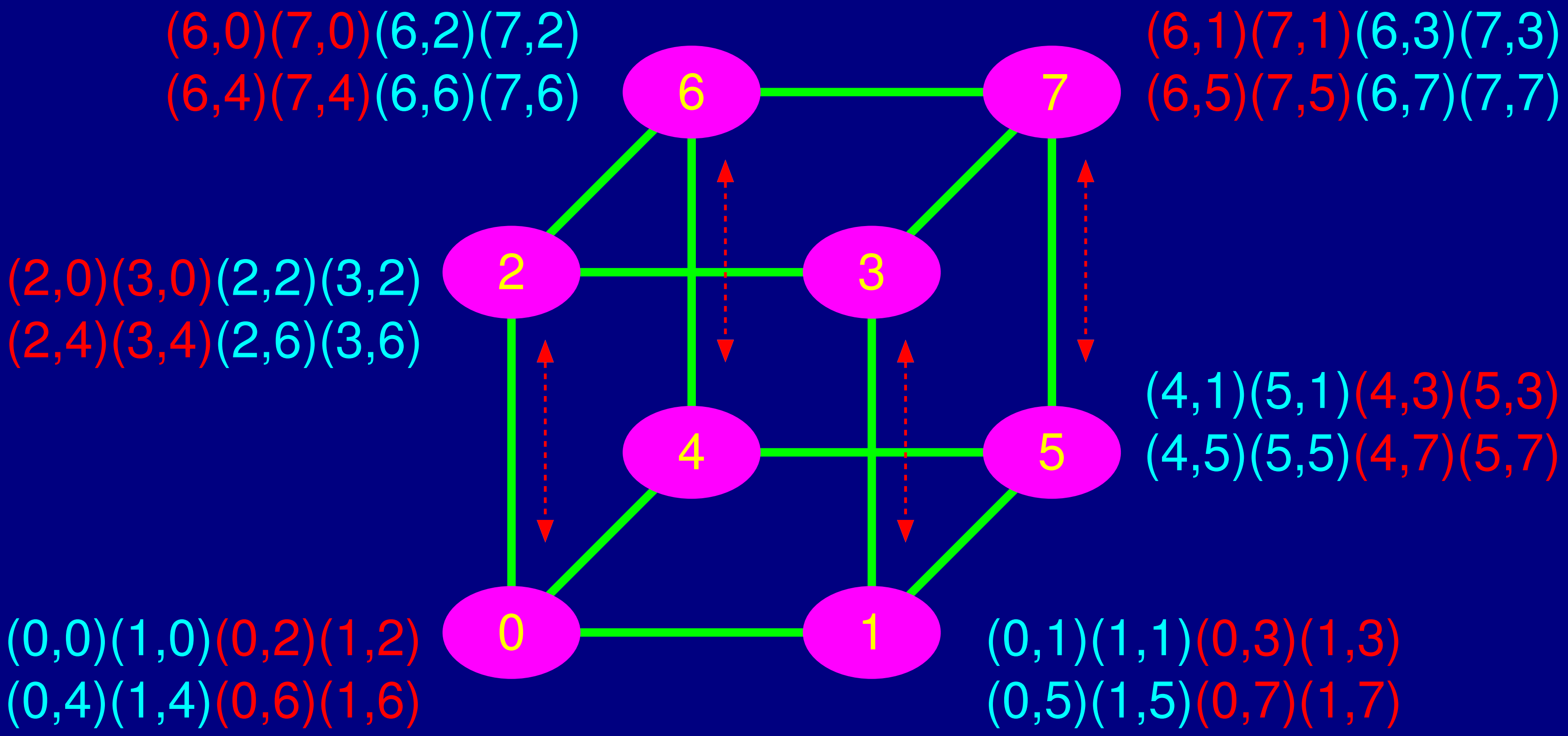
(5,0)(5,1)(5,2)(5,3)
(5,4)(5,5)(5,6)(5,7)

(0,0)(0,1)(0,2)(0,3)
(0,4)(0,5)(0,6)(0,7)

(1,0)(1,1)(1,2)(1,3)
(1,4)(1,5)(1,6)(1,7)

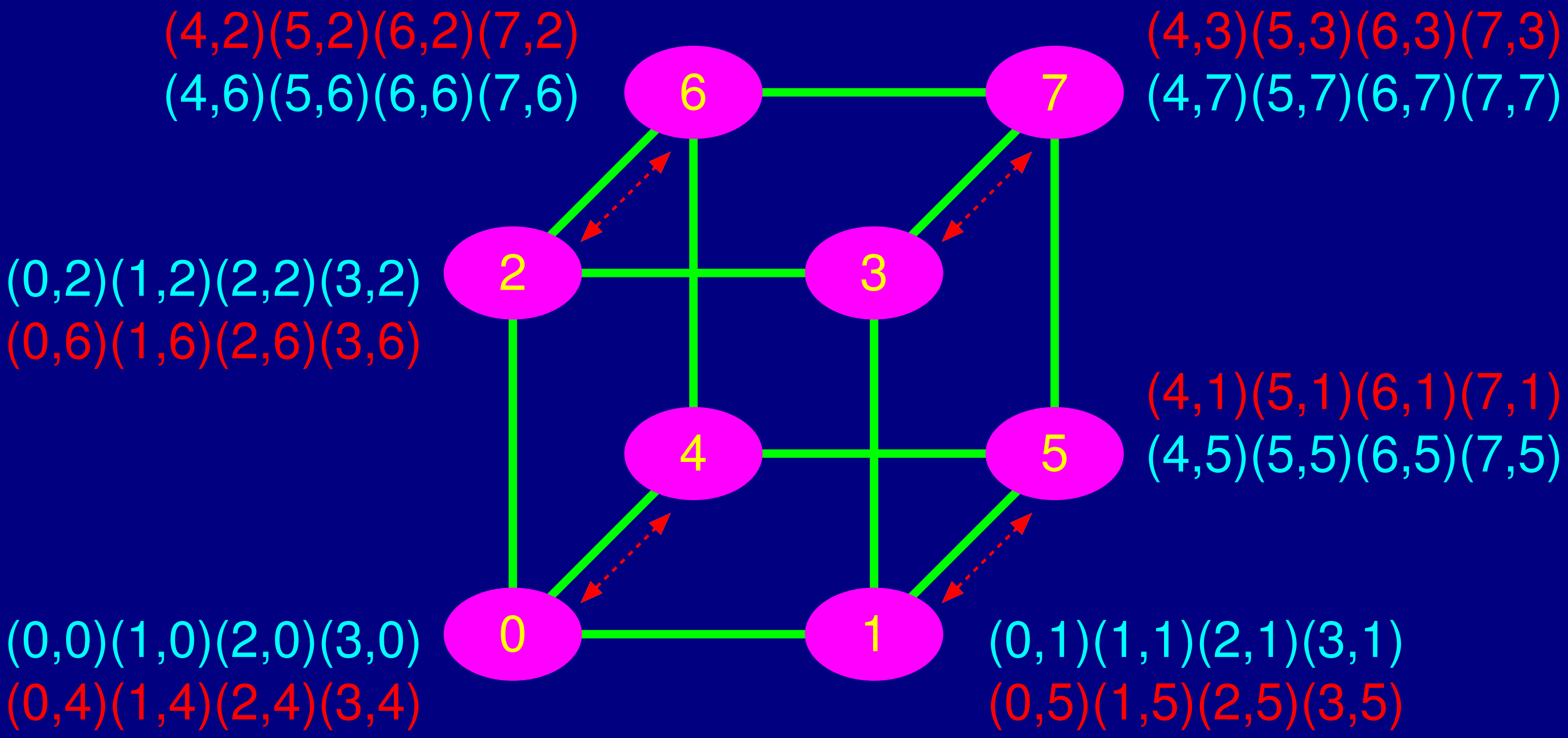
(a) Initial distribution of messages

All-to-All Personalized in Hypercube(SF)



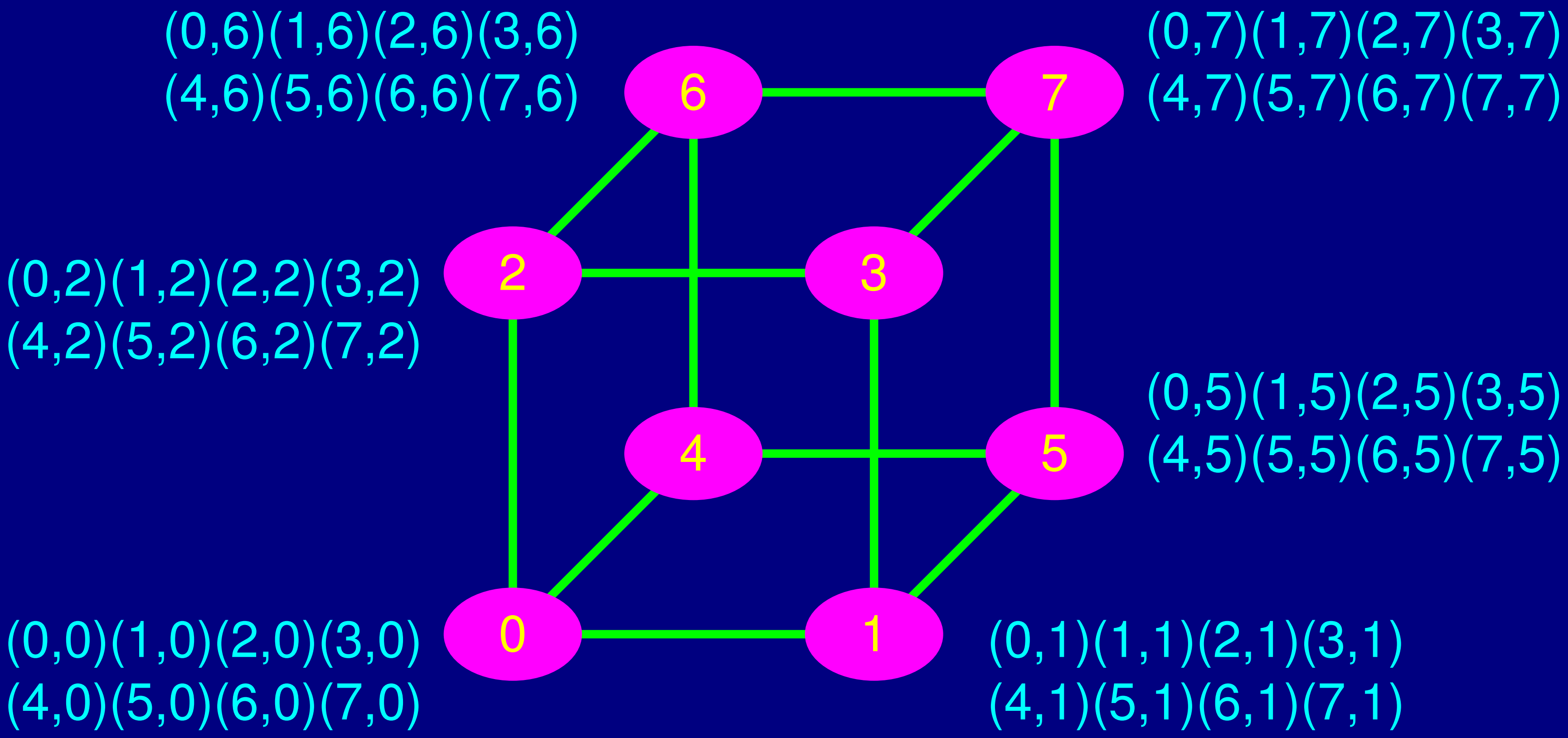
(b) Distribution before the second step

All-to-All Personalized in Hypercube(SF)



(c) Distribution before the third step

All-to-All Personalized in Hypercube(SF)



(0,6)(1,6)(2,6)(3,6)
(4,6)(5,6)(6,6)(7,6)

(0,7)(1,7)(2,7)(3,7)
(4,7)(5,7)(6,7)(7,7)

(0,2)(1,2)(2,2)(3,2)
(4,2)(5,2)(6,2)(7,2)

(0,5)(1,5)(2,5)(3,5)
(4,5)(5,5)(6,5)(7,5)

(0,0)(1,0)(2,0)(3,0)
(4,0)(5,0)(6,0)(7,0)

(0,1)(1,1)(2,1)(3,1)
(4,1)(5,1)(6,1)(7,1)

(d) Final distribution of messages

All-to-All Personalized in Hypercube(SF)

- It takes $d = \log p$ steps.
- The size of the messages exchanged in each step is $mp/2$.
- The time it takes a pair of nodes to send and receive from each other is $t_s + t_w mp/2$.
- Hence, the time it takes to complete the entire procedure is

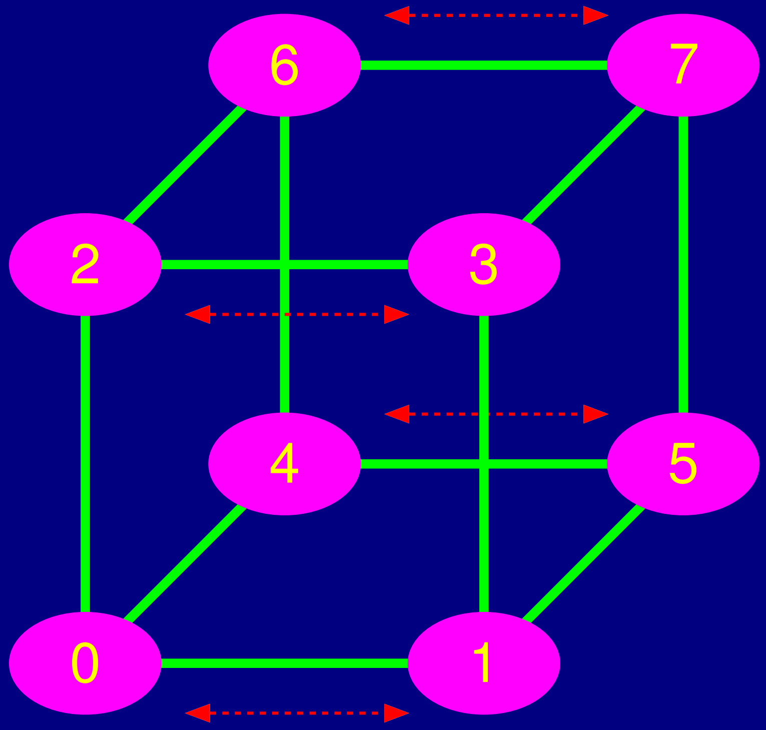
$$T_{all\_to\_all\_pers\_sf} = (t_s + t_w mp/2) \log p$$

- This is not optimal.

All-to-All Personalized in Hypercube(CT)

- Use cut-through routing.
- Each node simply performs $p - 1$ communication steps, exchanging m words of data with a different node in every step.
- In the jth step, node i exchanges data with node (i XOR j).
- In this schedule, all paths in every communication step are congestion-free, and none of the bidirectional links carry more than one message in the same direction.

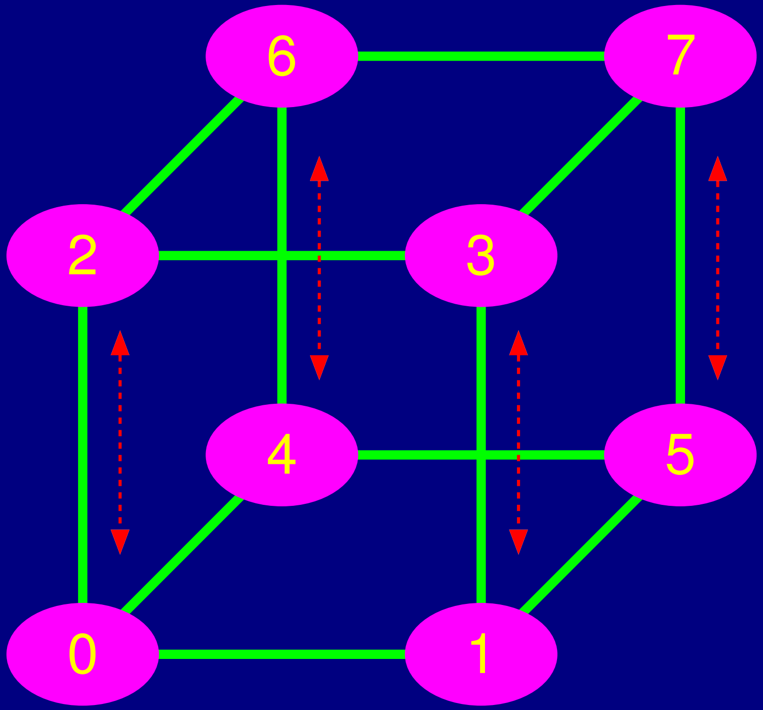
All-to-All Personalized in Hypercube(CT)



0 -> 1
1 -> 0
2 -> 3
3 -> 2
4 -> 5
5 -> 4
6 -> 7
7 -> 6

(1) Step 1 of 7

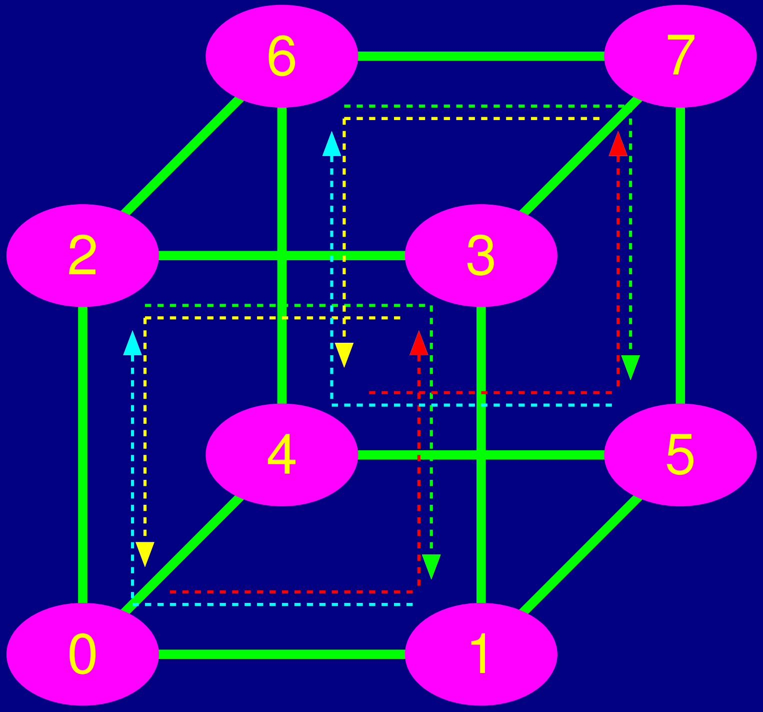
All-to-All Personalized in Hypercube(CT)



```
0 -> 2
1 -> 3
2 -> 0
3 -> 1
4 -> 6
5 -> 7
6 -> 4
7 -> 5
```

(2) Step 2 of 7

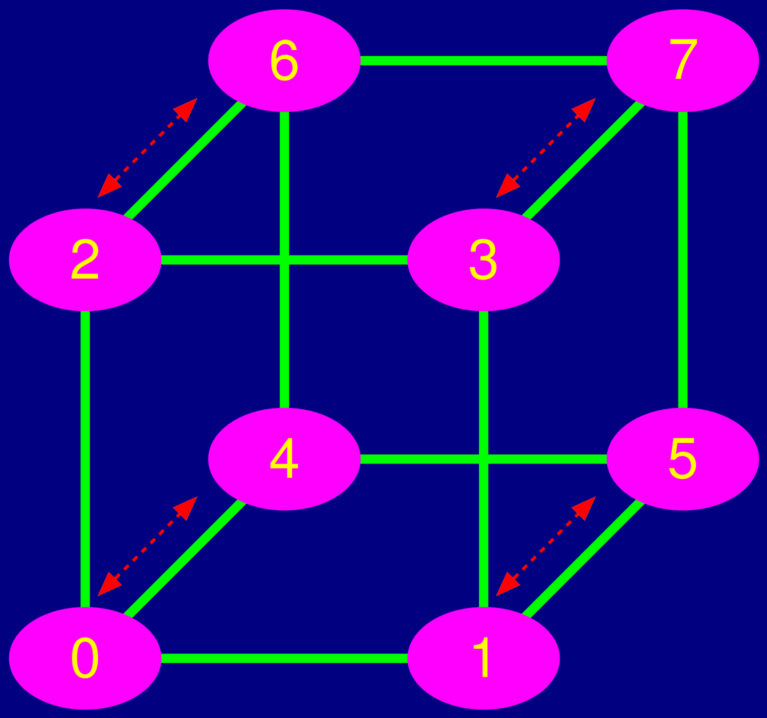
All-to-All Personalized in Hypercube(CT)



0 -> 1 -> 3
1 -> 0 -> 2
2 -> 3 -> 1
3 -> 2 -> 0
4 -> 5 -> 7
5 -> 4 -> 6
6 -> 7 -> 5
7 -> 6 -> 4

(3) Step 3 of 7

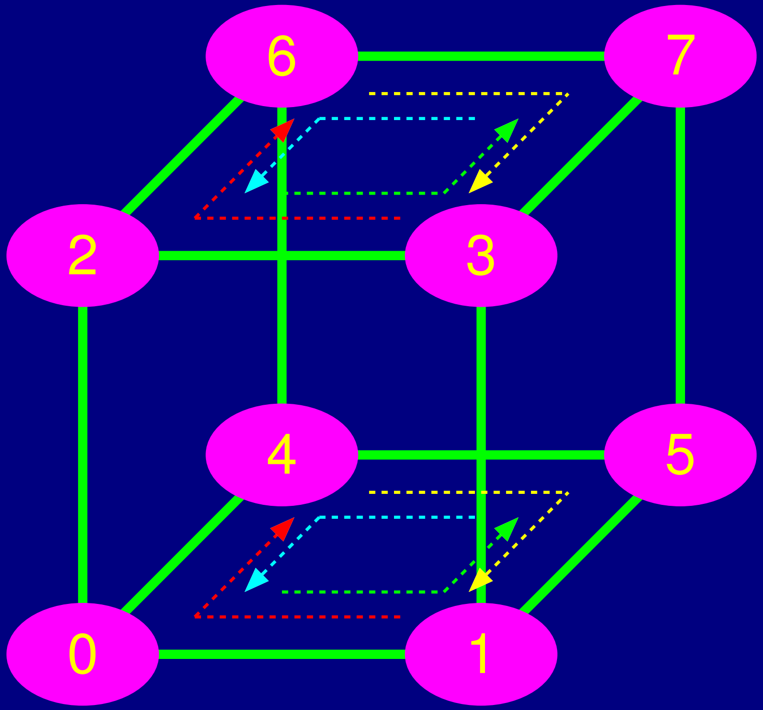
All-to-All Personalized in Hypercube(CT)



```
0 -> 4
1 -> 5
2 -> 6
3 -> 7
4 -> 0
5 -> 1
6 -> 2
7 -> 3
```

(4) Step 4 of 7

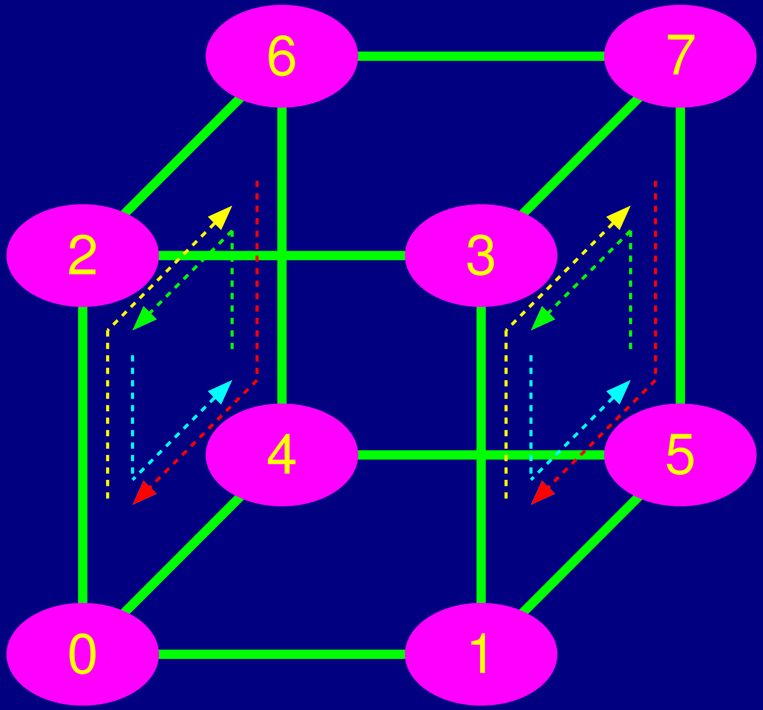
All-to-All Personalized in Hypercube(CT)



```
0 -> 1 -> 5
1 -> 0 -> 4
2 -> 3 -> 7
3 -> 2 -> 6
4 -> 5 -> 1
5 -> 4 -> 0
6 -> 7 -> 3
7 -> 6 -> 2
```

(5) Step 5 of 7

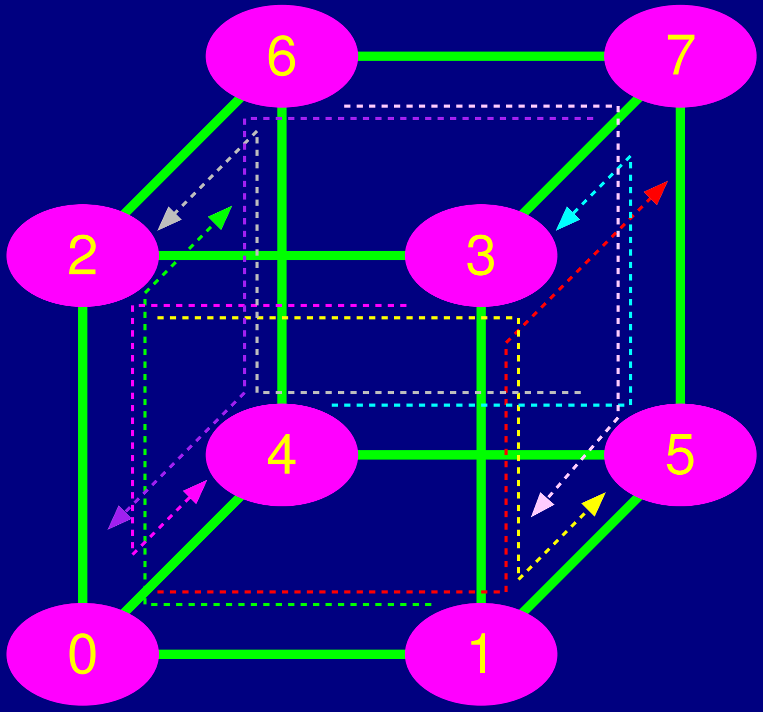
All-to-All Personalized in Hypercube(CT)



0 -> 2 -> 6
1 -> 3 -> 7
2 -> 0 -> 4
3 -> 1 -> 5
4 -> 6 -> 2
5 -> 7 -> 3
6 -> 4 -> 0
7 -> 5 -> 1

(6) Step 6 of 7

All-to-All Personalized in Hypercube(CT)



```
0 -> 1 -> 3 -> 7
1 -> 0 -> 2 -> 6
2 -> 3 -> 1 -> 5
3 -> 2 -> 0 -> 4
4 -> 5 -> 7 -> 3
5 -> 4 -> 6 -> 2
6 -> 7 -> 5 -> 1
7 -> 6 -> 4 -> 0
```

(7) Step 7 of 7

All-to-All Personalized in Hypercube(CT)

- There are $p - 1$ communication steps.
- In each step, node i sends m words to node j.
- It takes $t_s + t_w m + t_h l$, where l is the Hamming distance between i and j.
- For a given i, on a p-node hypercube, the sum of all l for $0 \leq j < p$ is $(p \log p)/2$.
- The total communication time for entire operation is

$$T_{all\_to\_all\_pers\_ct} = (t_s + t_w m)(p - 1) + \frac{1}{2} t_h p \log p.$$

All-to-All Personalized in Hypercube(CT)

```
procedure ALL_TO_ALL_PERSONAL($d$, $my\_id$)
begin
    for $i := 1$ to $2^d - 1$ do
        $partner := my\_id$ XOR $i$;
        send $M_{my\_id,partner}$ to $partner$;
        receive $M_{partner,my\_id}$ from $partner$;
    endfor
end
```

Subsection III.2

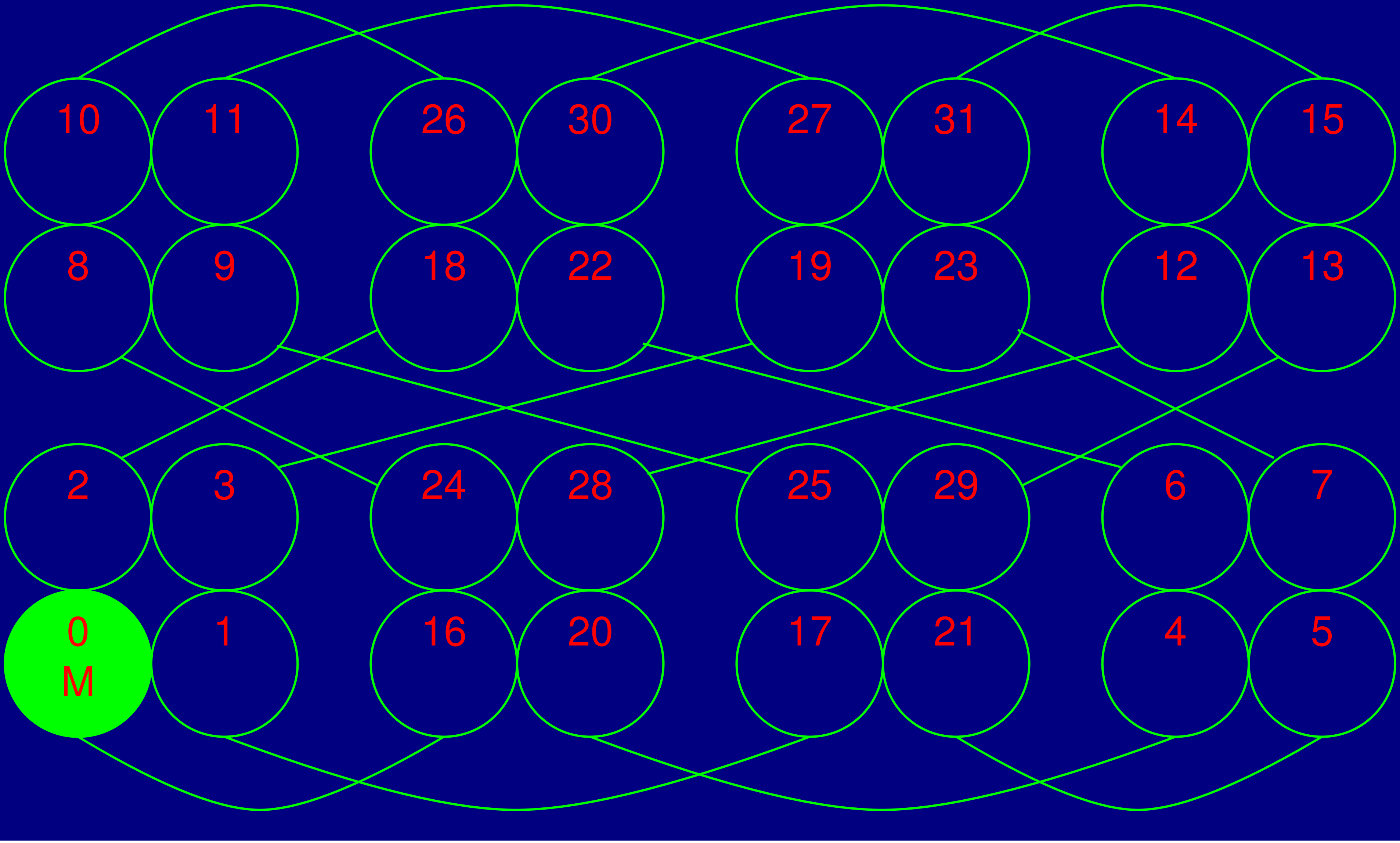
One-to-All Broadcasting in Dual-Cube

A single node *s* sends identical data to all other nodes

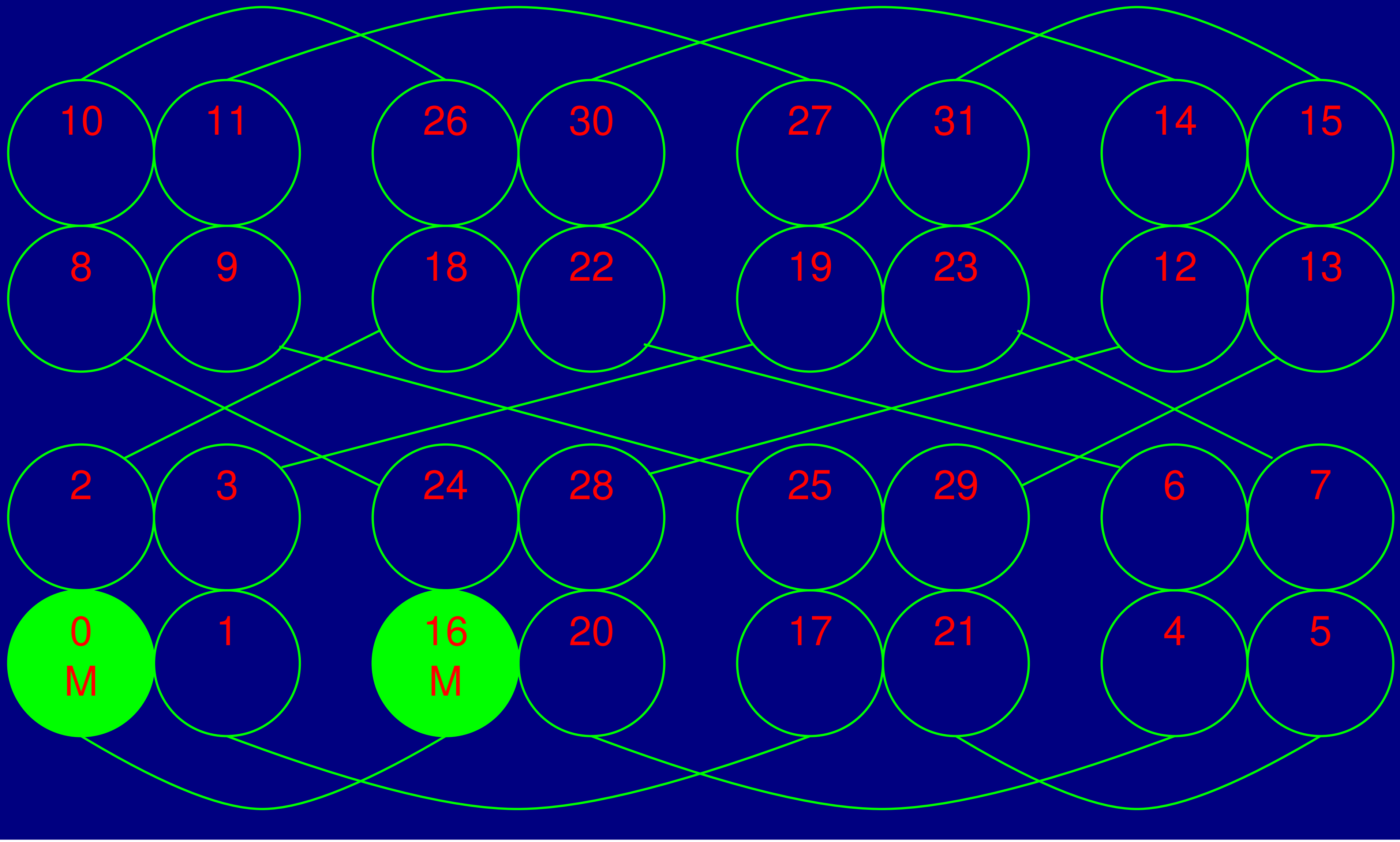
One-to-All Broadcasting Algorithm

1. Send message through cross-edge:
   - The source node s sends the message to s'.
2. Broadcast inside clusters:
   - s and s' broadcast the message simultaneously.
3. Send message through cross-edge:
   - Every node $u \in C_s \setminus \{s\}$ and every node $u' \in C_{s'} \setminus \{s'\}$ send the message to v and v'.
4. Broadcast inside clusters:
   - v and v' broadcast the message simultaneously.

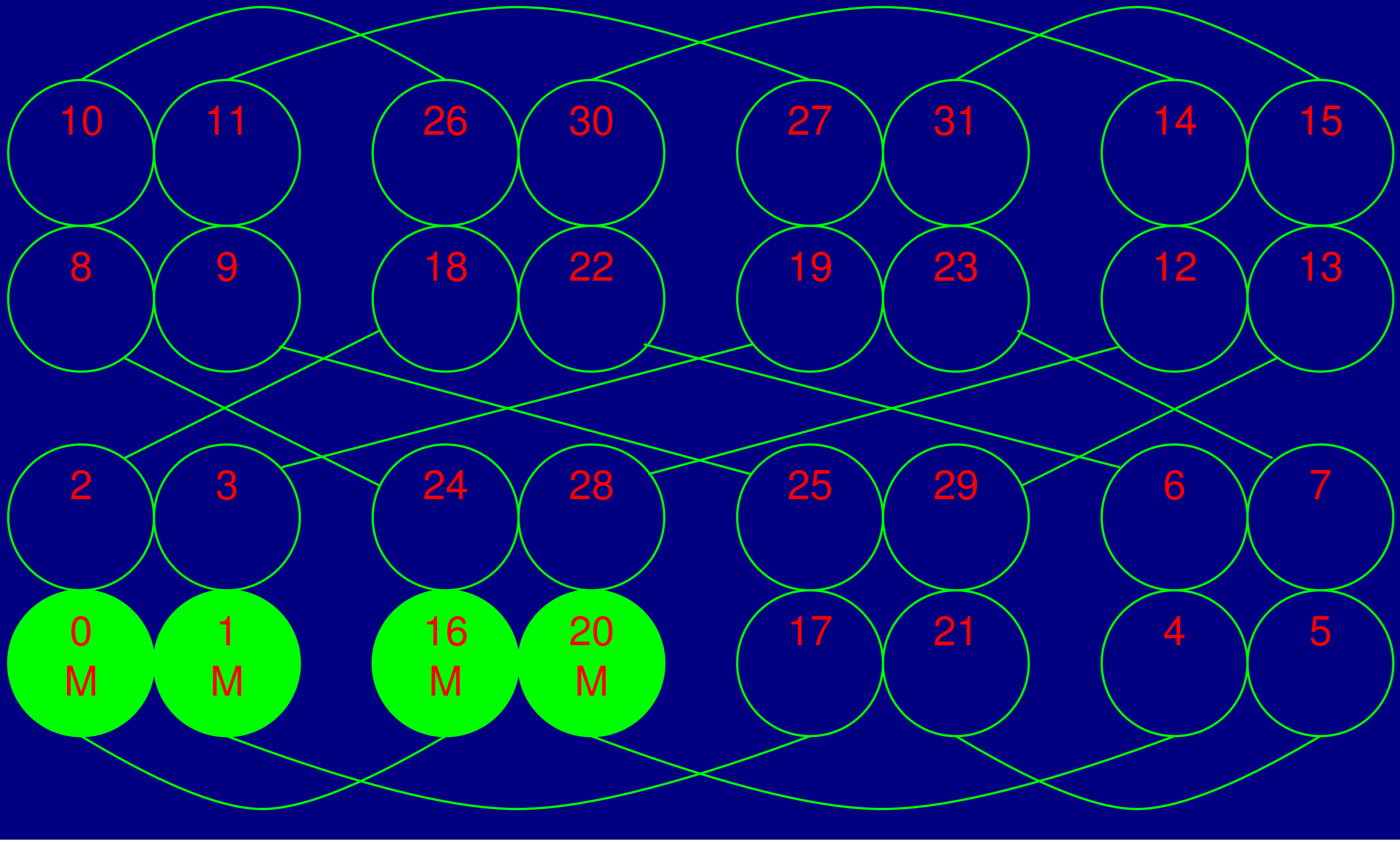
Example of One-to-All Broadcasting



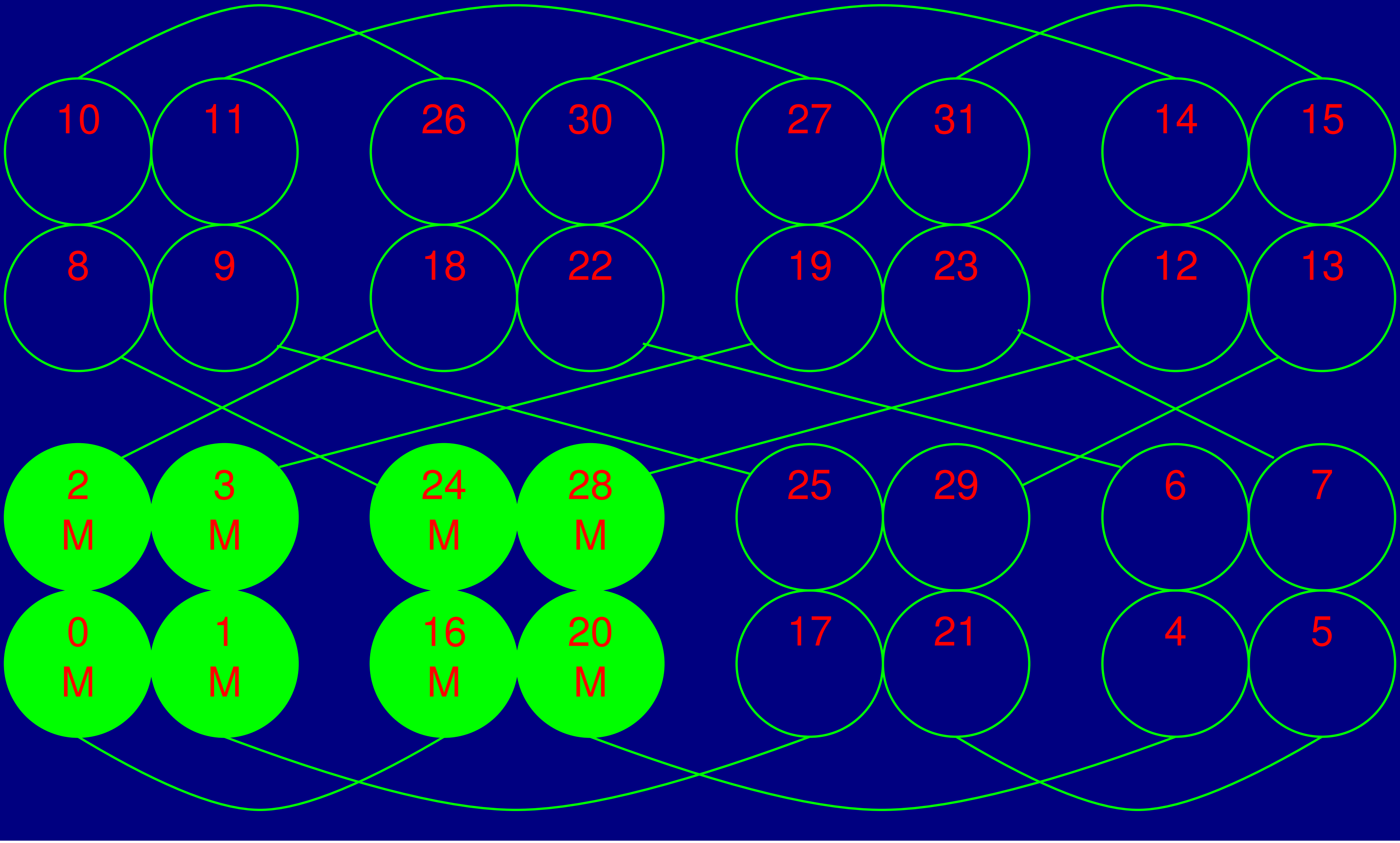
Example (Cross-Edge)



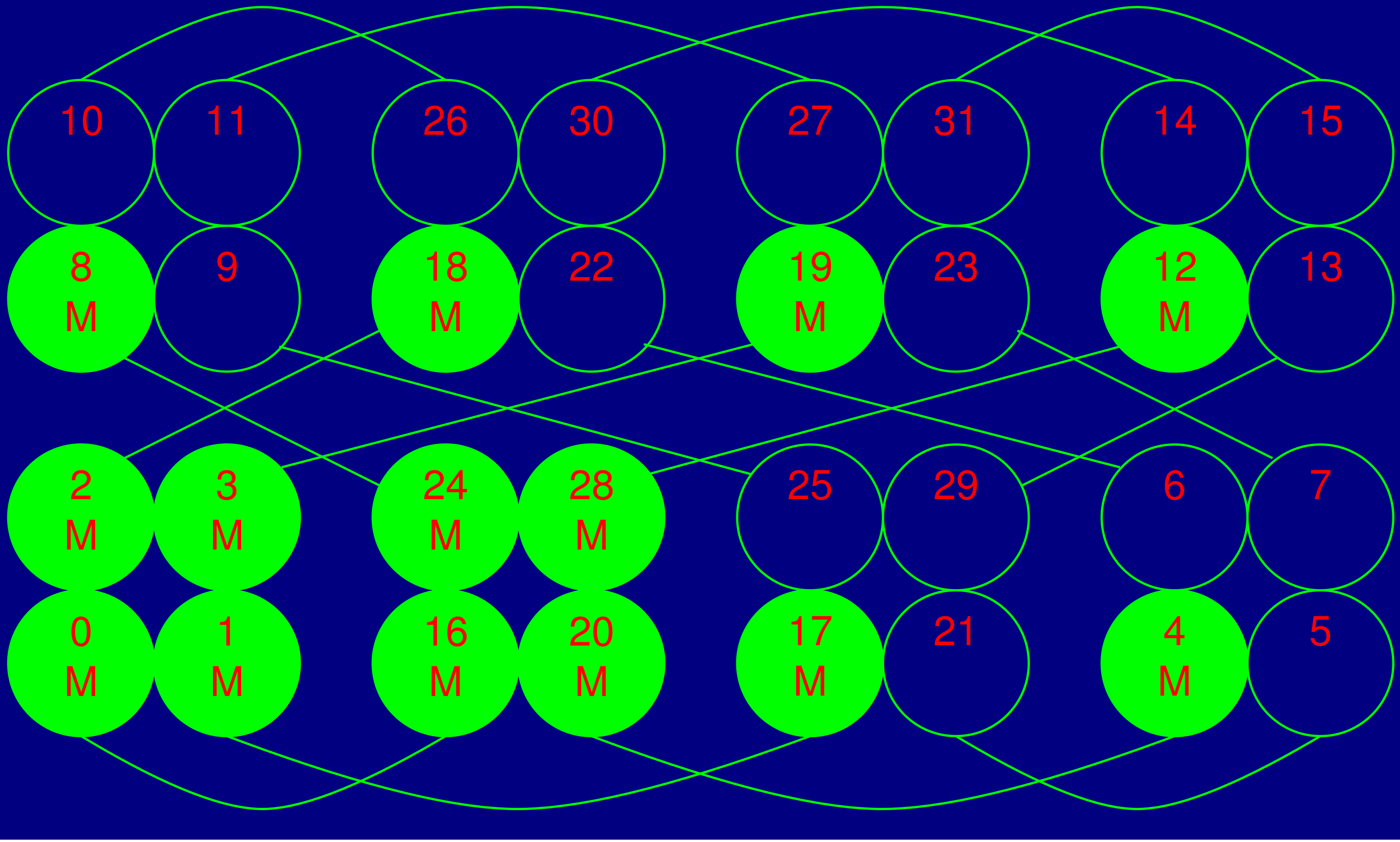
Example (Dimension 0: Horizontal)



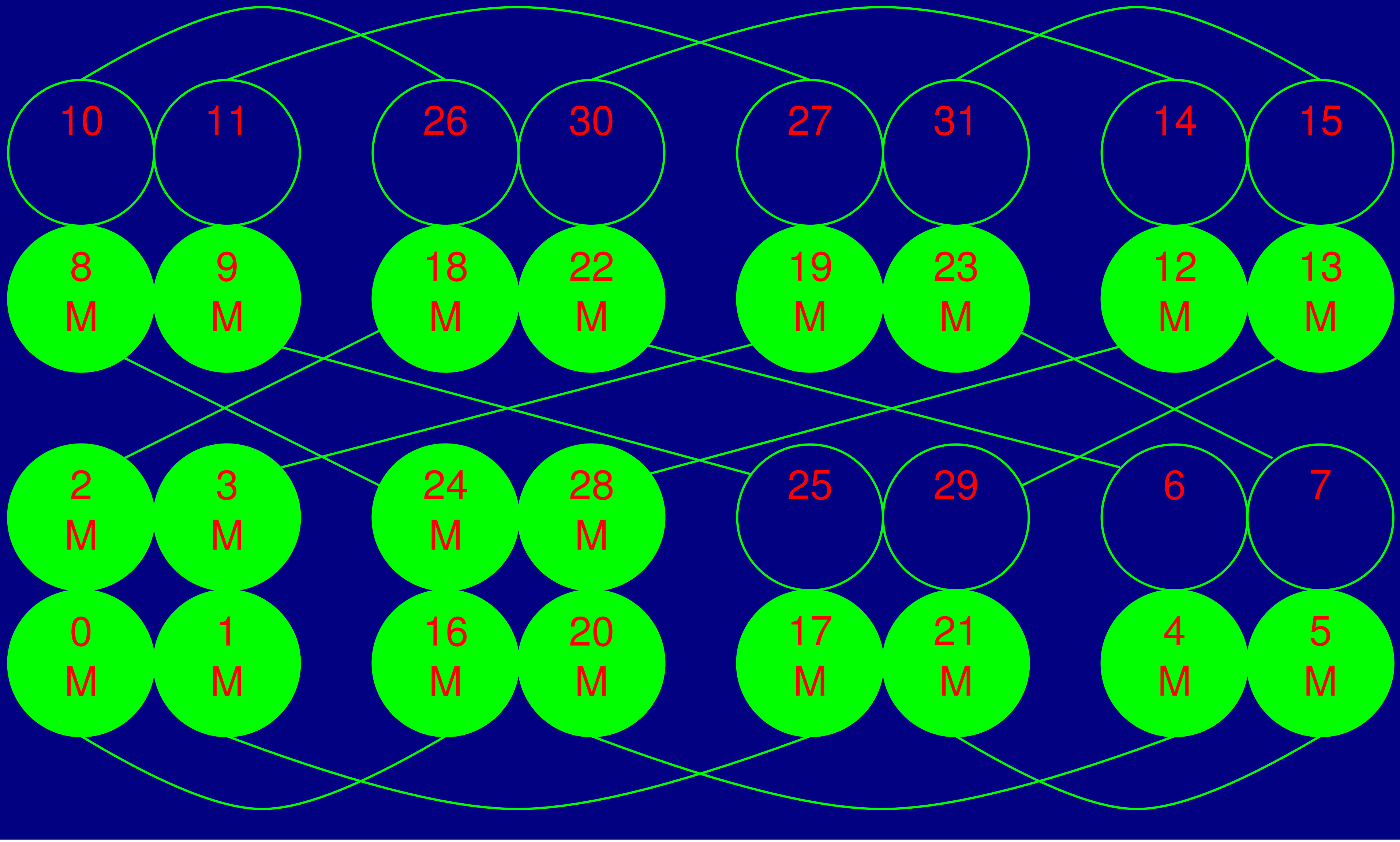
Example (Dimension 1: Vertical)



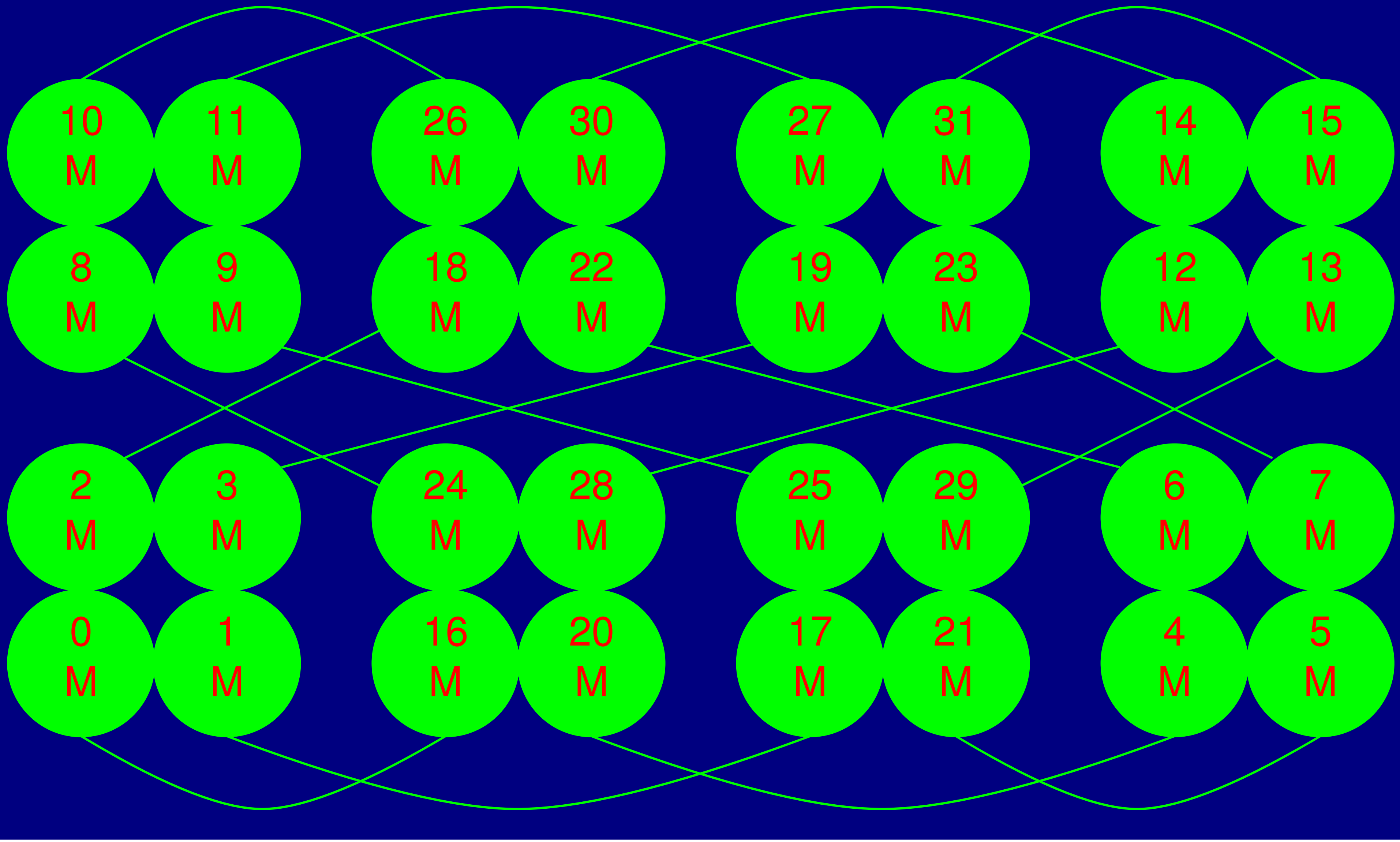
Example (Cross-Edge)



Example (Dimension 0: Horizontal)



Example (Dimension 1: Vertical)



Time of One-to-All Broadcasting

- The steps the broadcasting is completed:
  - $1 + m + 1 + m = 2m + 2$
- Therefore, the total communication time:
  - $$\begin{aligned} T &= (t_s + wt_w + (1 - 1)t_h)(2m + 2) \\ &= (t_s + wt_w)(1 + \log_2 p) \end{aligned}$$
    - $p = 2^{2m+1}$
- Hypercube (n-cube):
  - $T = (t_s + wt_w)n = (t_s + wt_w)\log_2 p$
    - $n = 2m + 1$

Subsection III.3

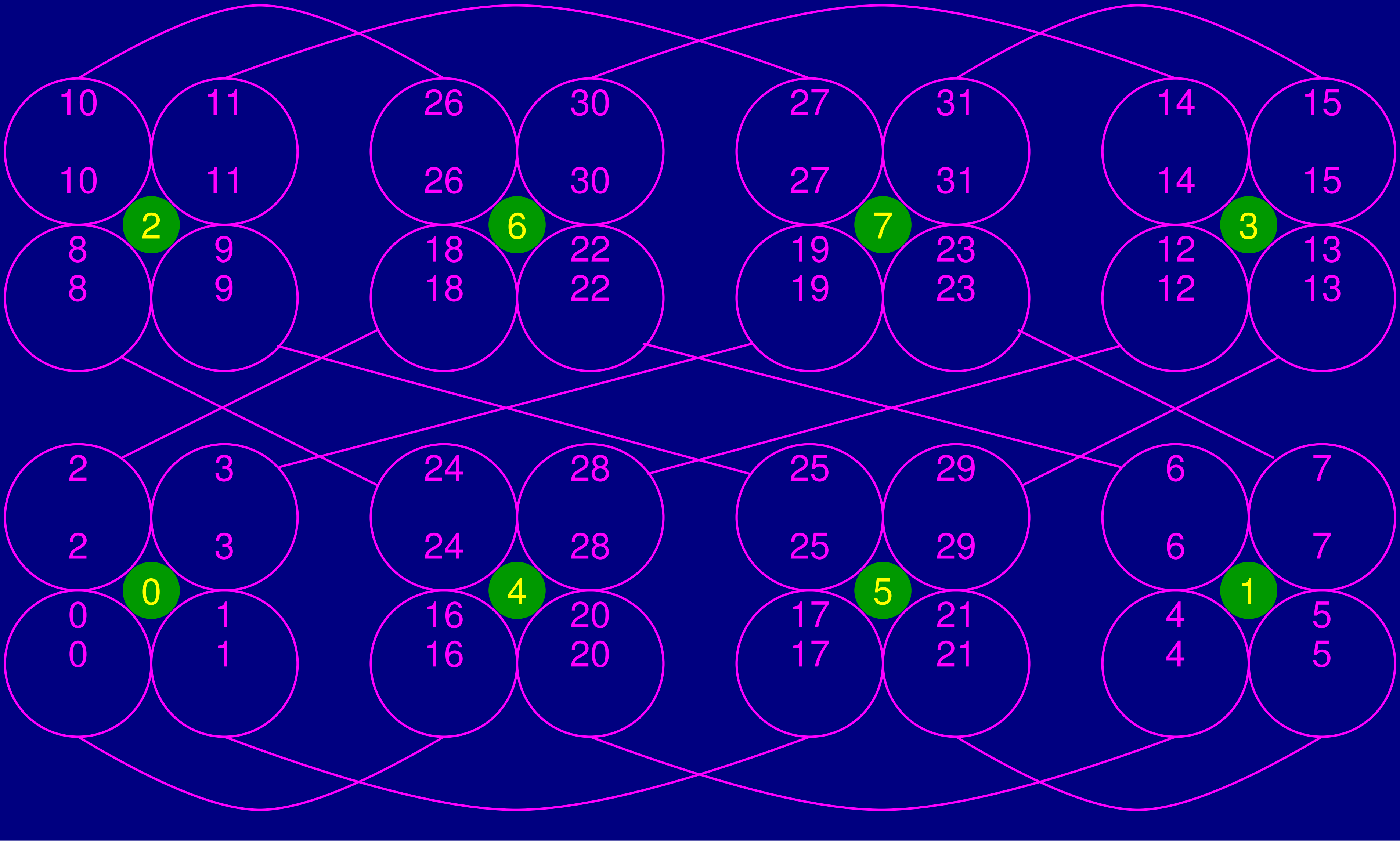
All-to-All Broadcasting in Dual-Cube

In all-to-all broadcast, all nodes initiate a broadcast

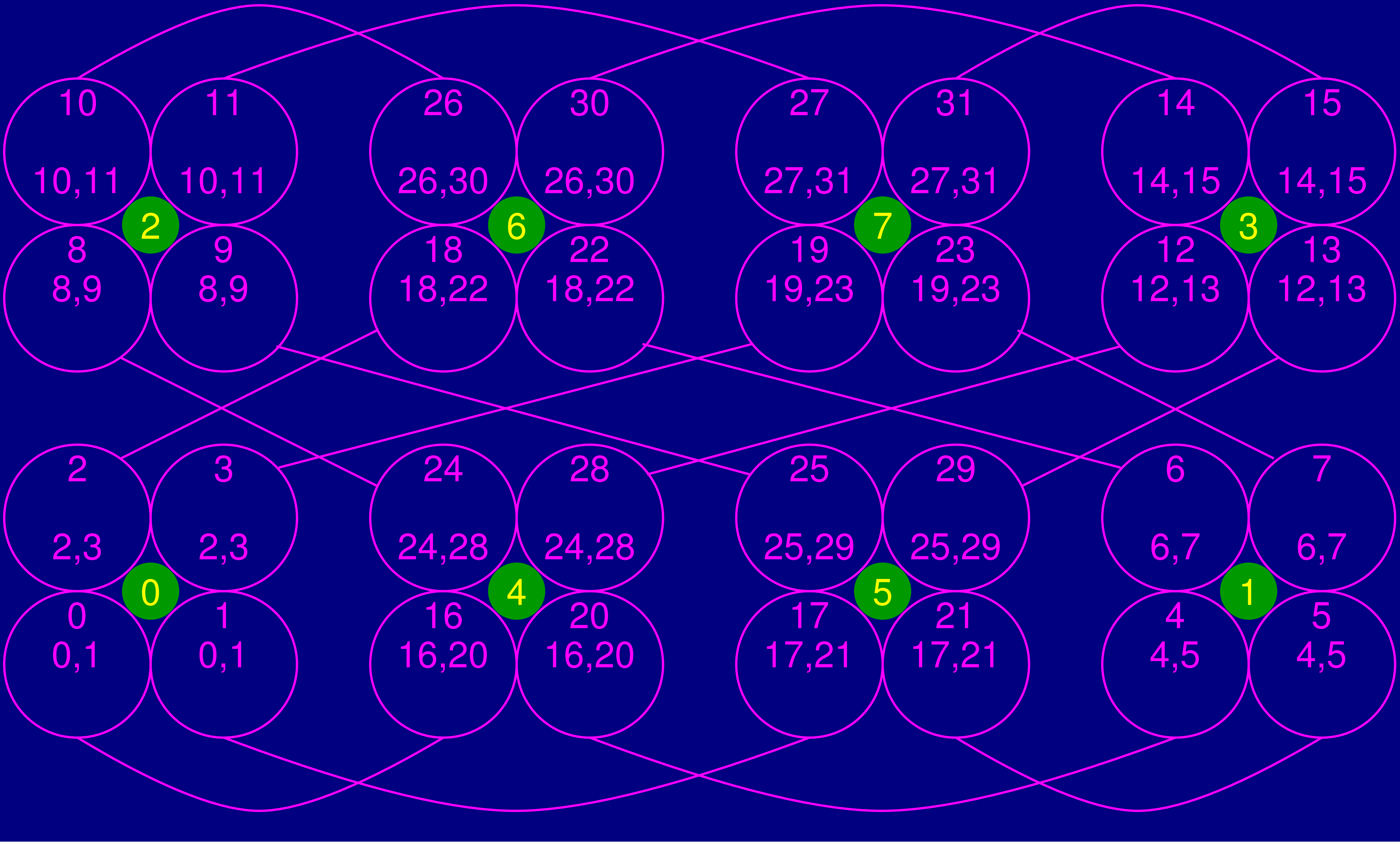
All-to-All Broadcasting Algorithm

1. Broadcasting inside each cluster.
   - $T_1 = \sum_{i=0}^{m-1}(t_s + 2^i w t_w) = m t_s + (2^m - 1) w t_w$
2. Each node sends messages through cross-edge.
   - $T_2 = t_s + 2^m w t_w$
3. The received messages are broadcast inside the cluster.
   - $T_3 = \sum_{i=0}^{m-1}(t_s + 2^{m+i} w t_w) = m t_s + 2^m (2^m - 1) w t_w$
4. Each node sends messages through cross-edge.
   - $T_4 = t_s + (2^{2m} - 2^m) w t_w$

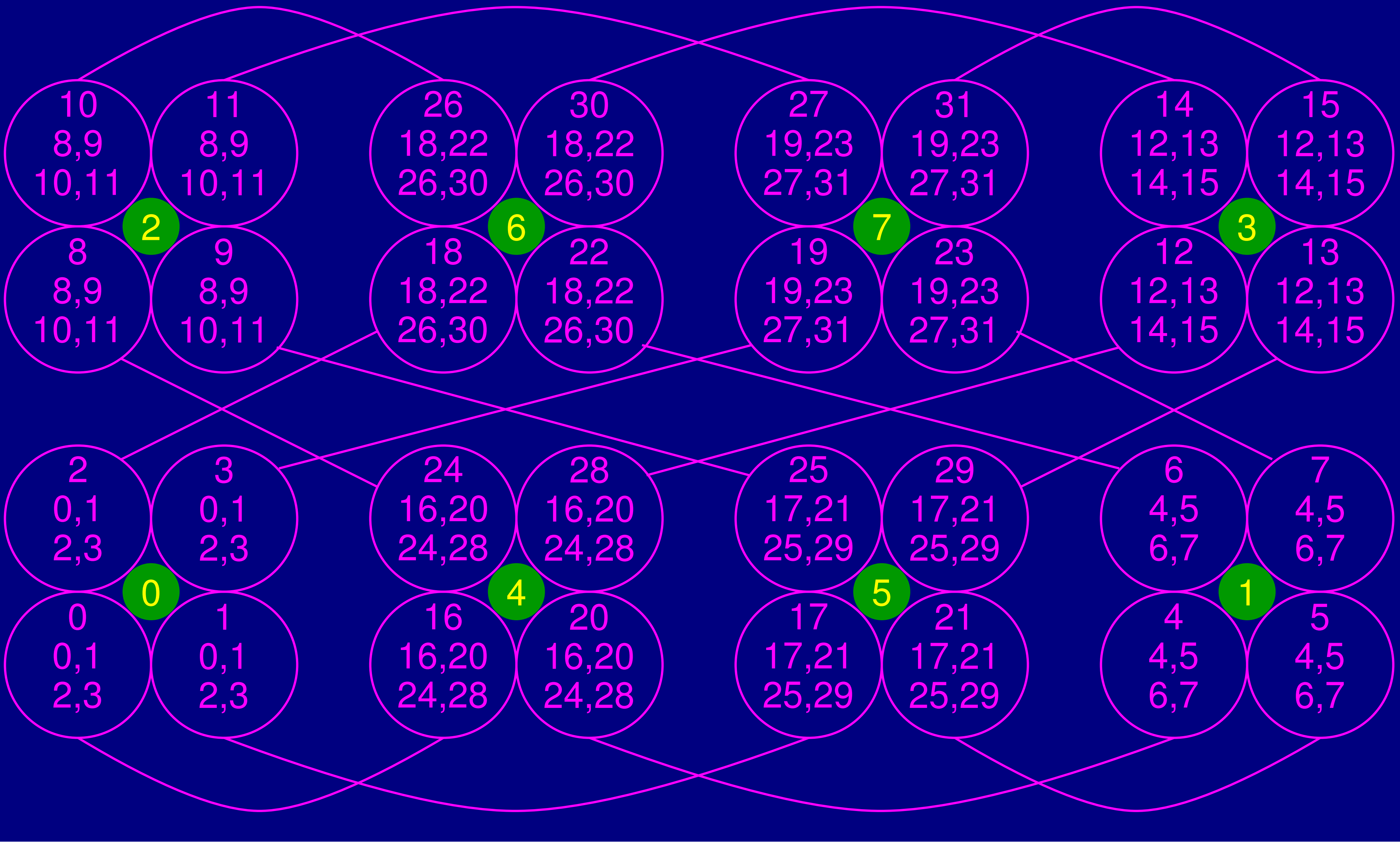
Example of All-to-All Broadcasting



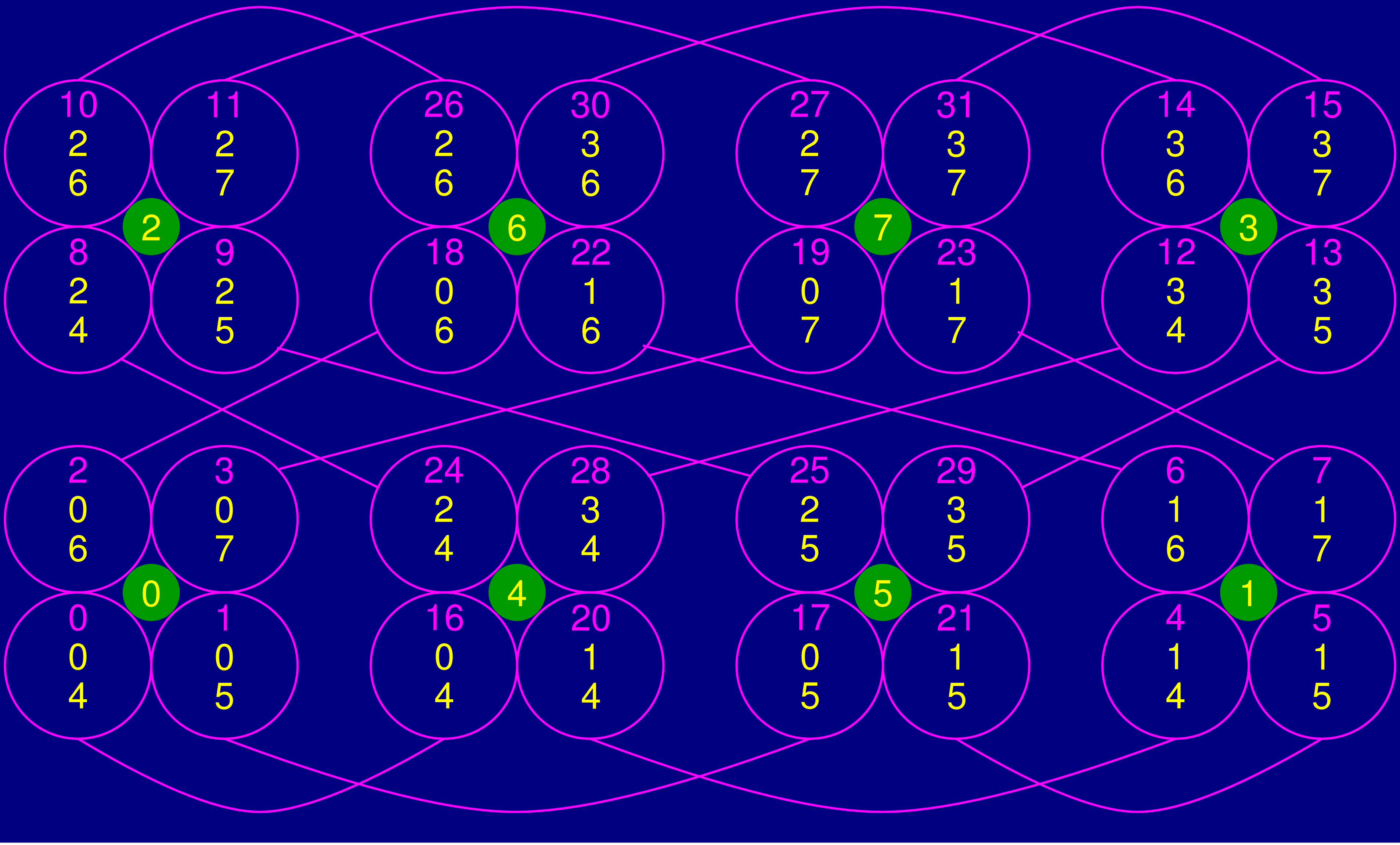
Example (Dimension 0: Horizontal)



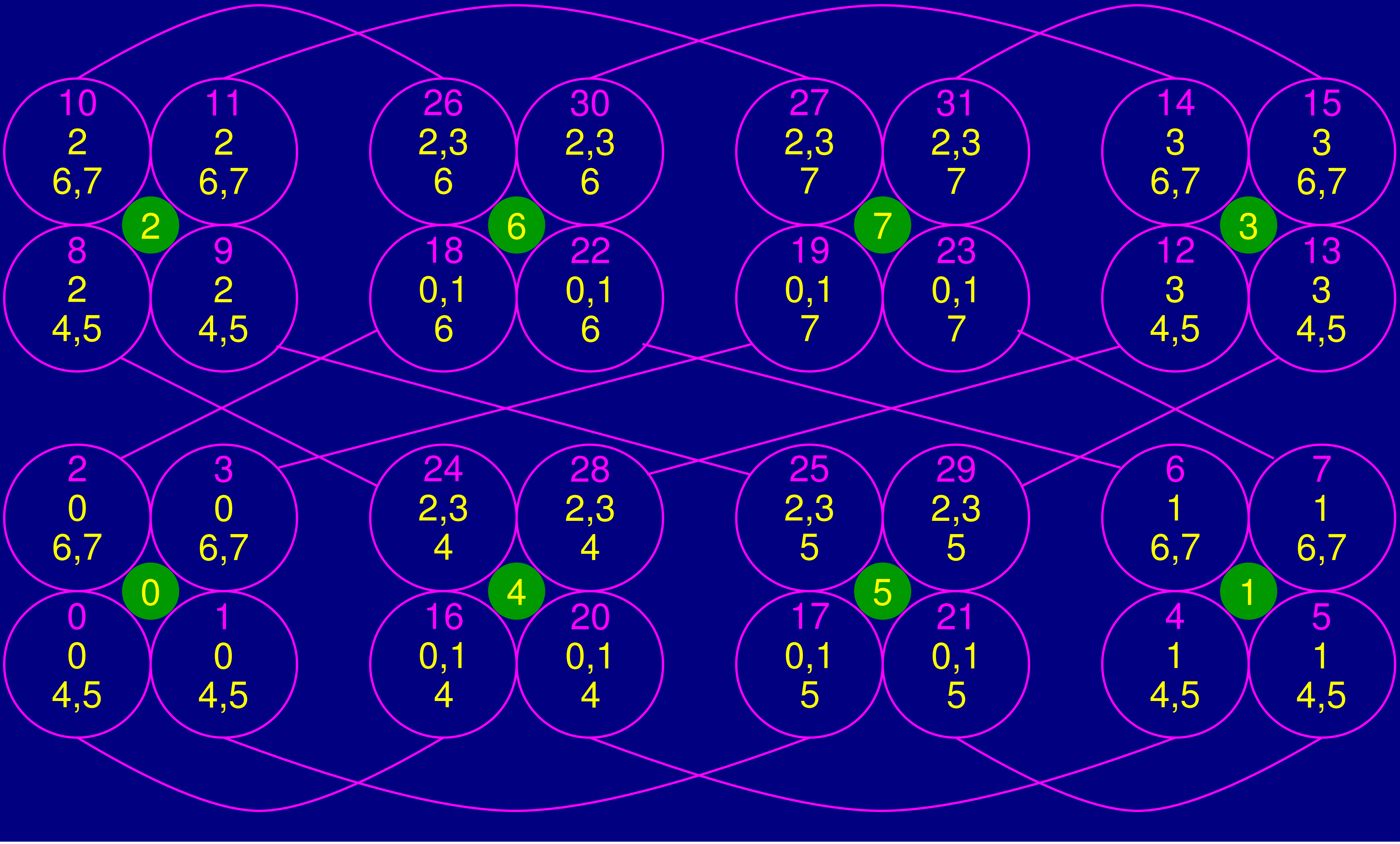
Example (Dimension 1: Vertical)



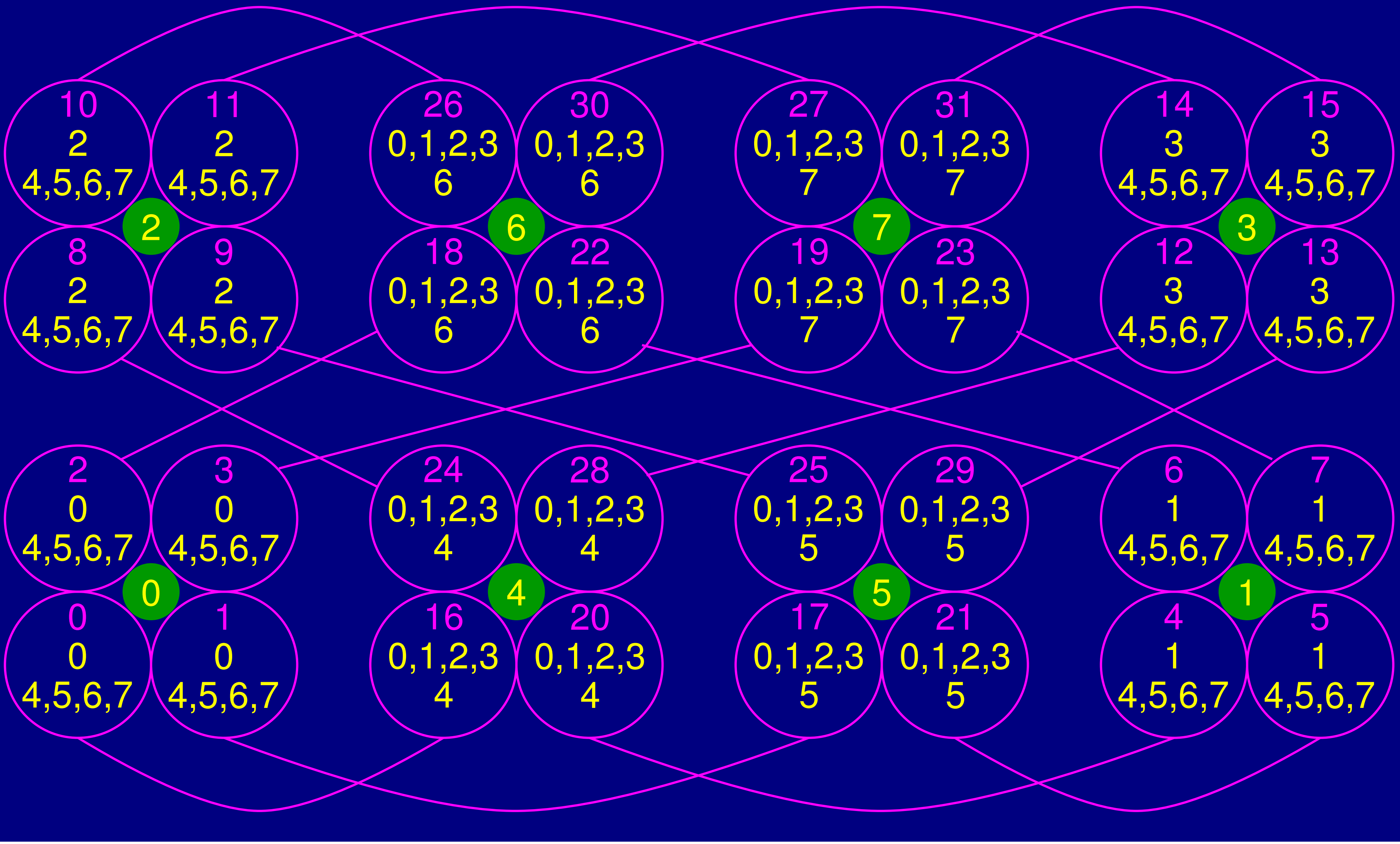
Example (Cross-Edge, Cluster #)



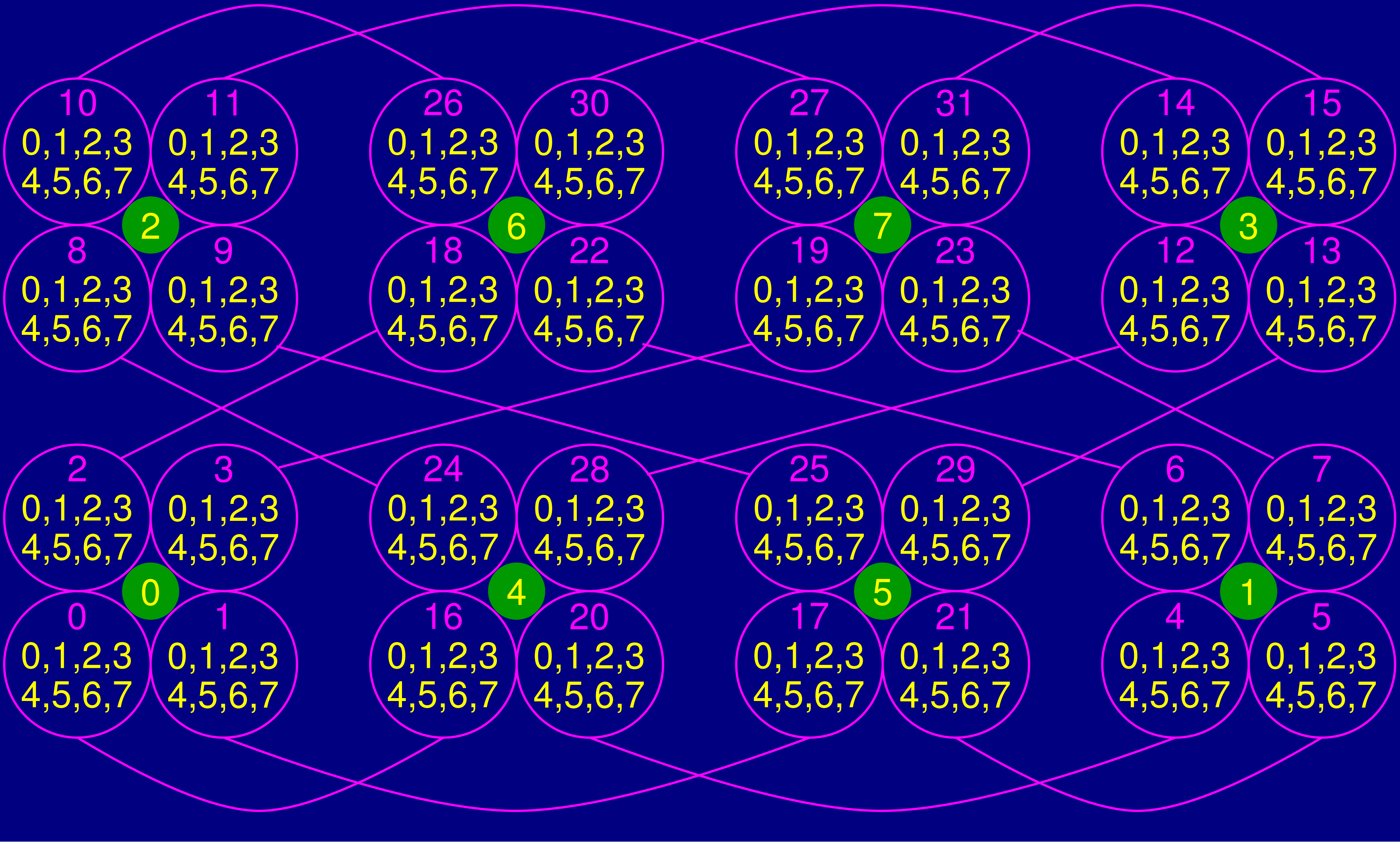
Example (Dimension 0: Horizontal)



Example (Dimension 1: Vertical)



Example (Cross-Edge, Finished)



Time of All-to-All Broadcasting

- $T_1 = mt_s + (2^m - 1)wt_w$
- $T_2 = t_s + 2^m wt_w$
- $T_3 = mt_s + 2^m(2^m - 1)wt_w$
- $T_4 = t_s + (2^{2m} - 2^m)wt_w$
- Total time to complete the all-to-all broadcast $T = T_1 + T_2 + T_3 + T_4 = (1 + \log_2 p)t_s + (p - 1)wt_w$
- Hypercube: $T = (\log_2 p)t_s + (p - 1)wt_w$

Subsection III.4

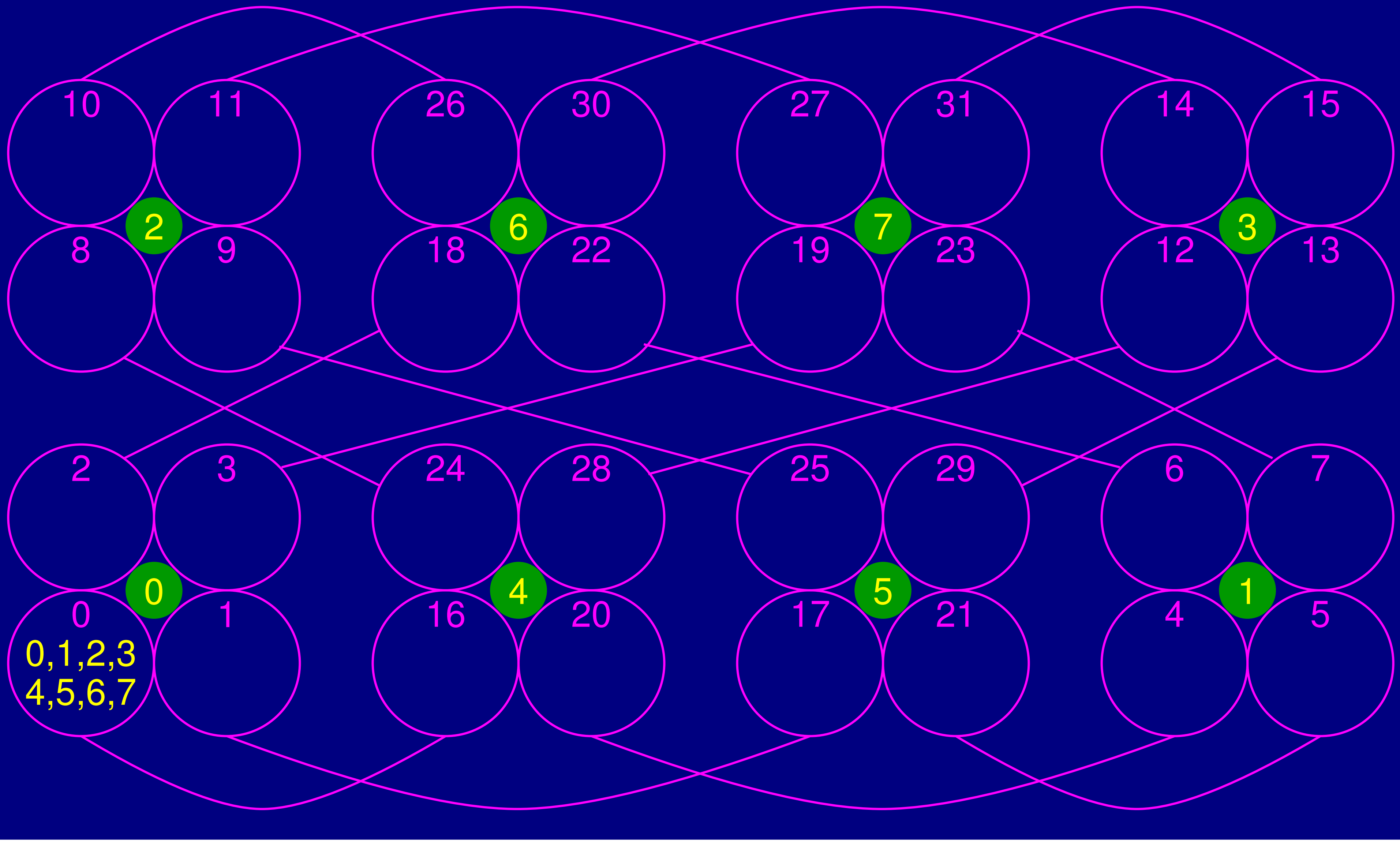
One-to-All Personalized Communication

Node *s* sends a unique message to every other node

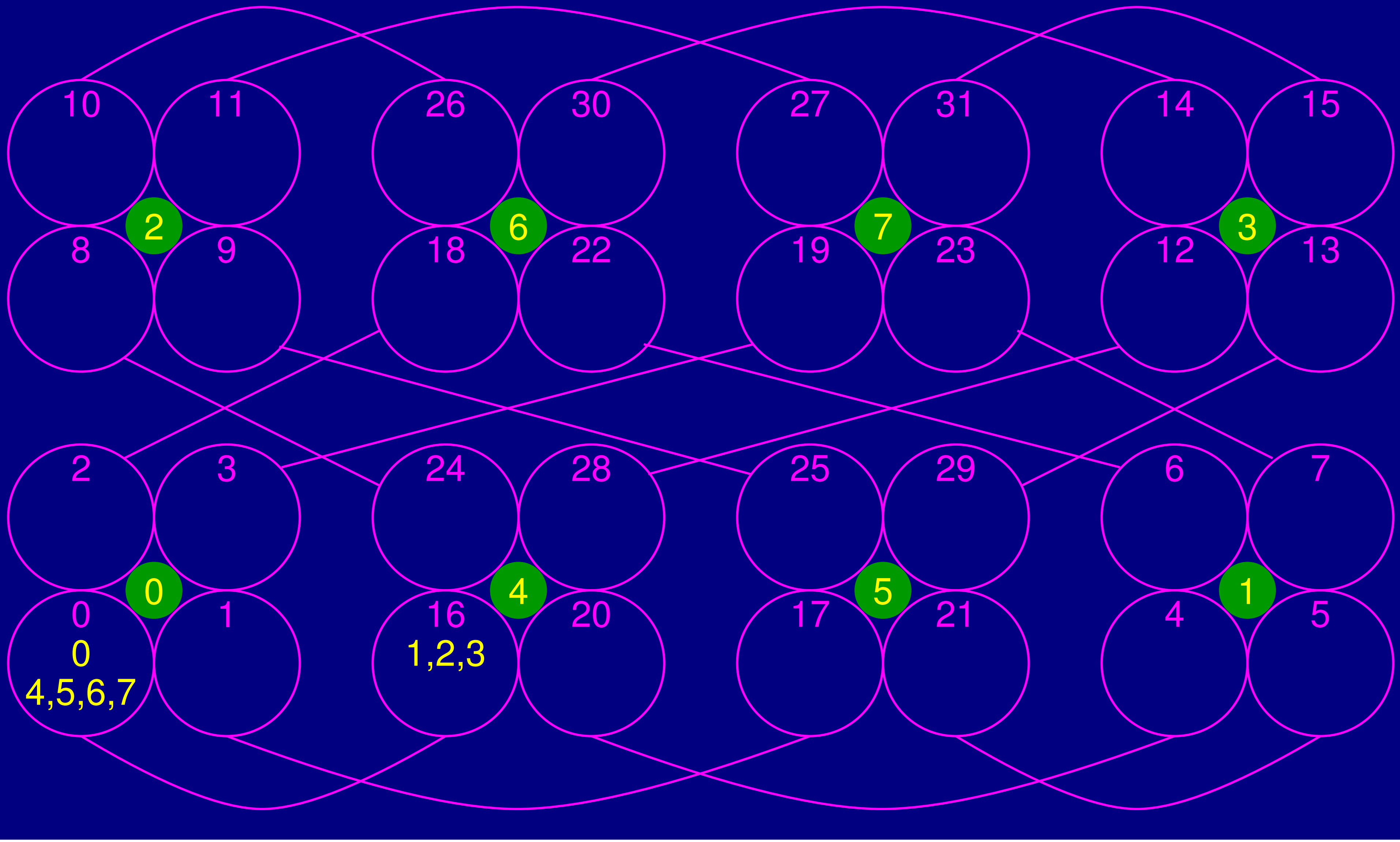
One-to-All Personalized Algorithm

1. Node s sends messages to s' through cross-edge.
   - $T_1 = t_s + (2^{2m} - 2^m) w t_w$
2. Nodes s and s' send messages inside clusters.
   - $T_2 = \sum_{i=0}^{m-1} (t_s + 2^{2m-(i+1)} w t_w) = m t_s + 2^m (2^m - 1) w t_w$
3. Send messages through cross-edge.
   - $T_3 = t_s + 2^m w t_w$
4. Send the messages to all nodes inside clusters.
   - $T_4 = \sum_{i=0}^{m-1} (t_s + 2^{m-(i+1)} w t_w) = m t_s + (2^m - 1) w t_w$

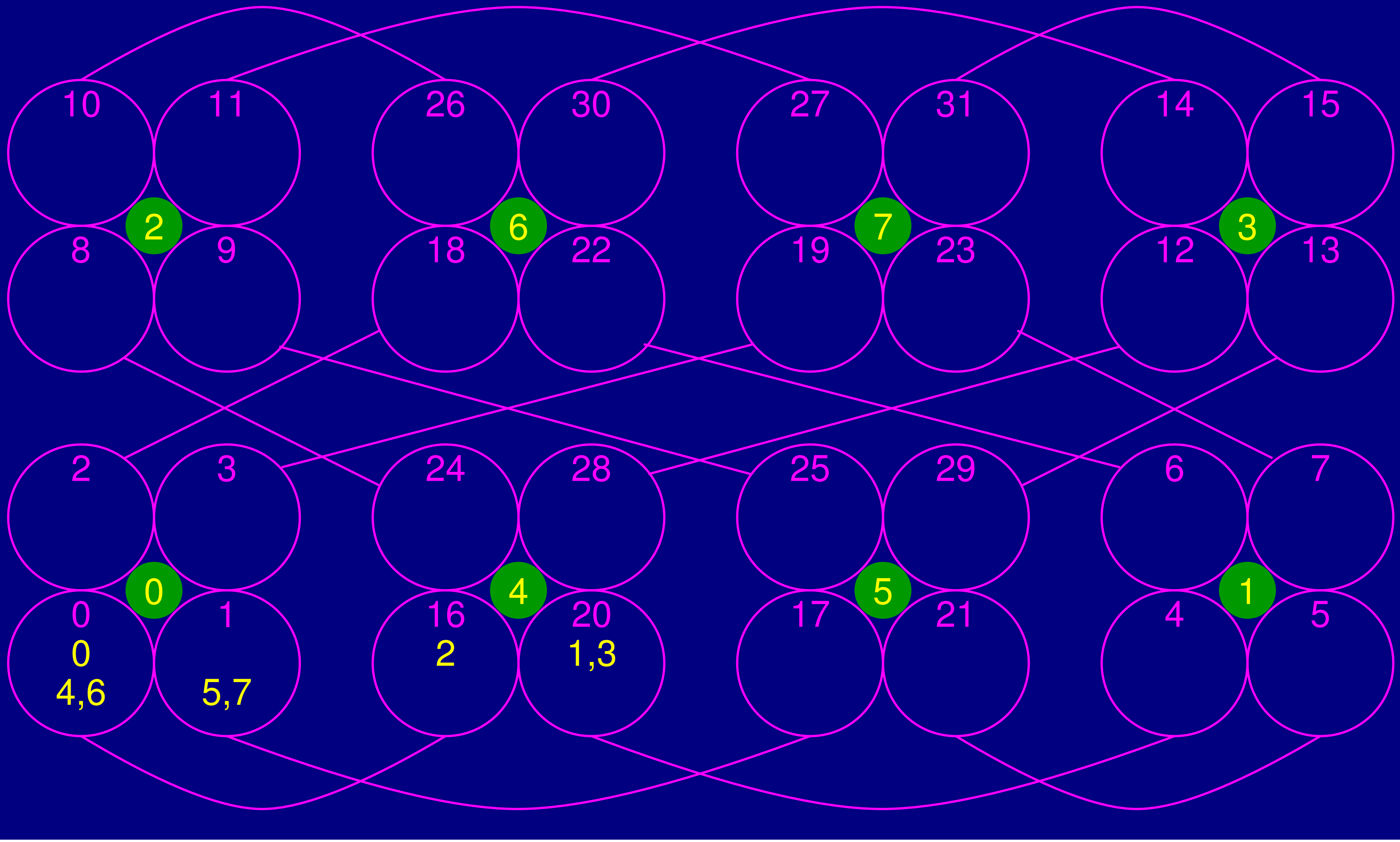
Example of One-to-all Personalized



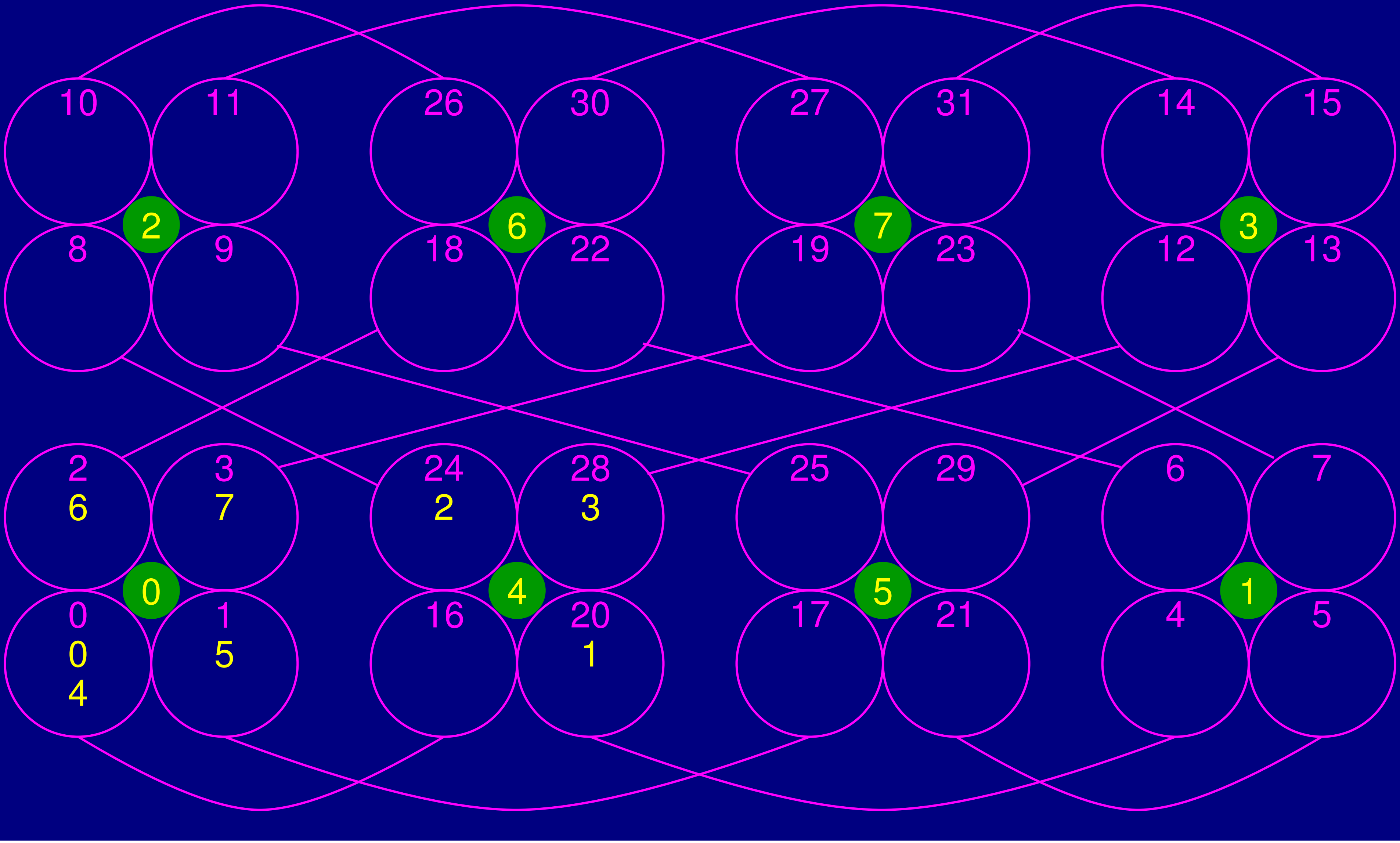
Example (Cross-Edge)



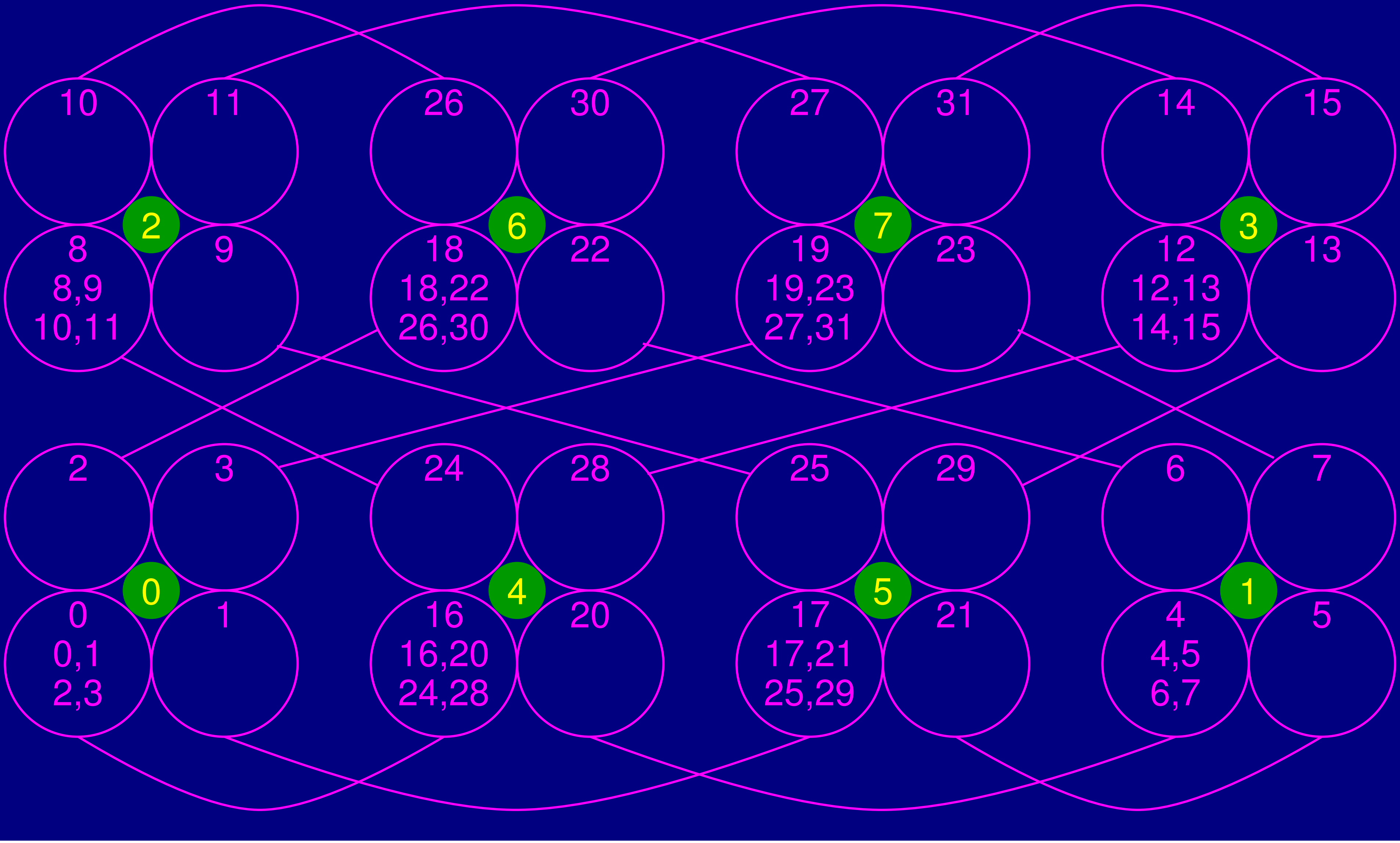
Example (Dimension 0: Horizontal)



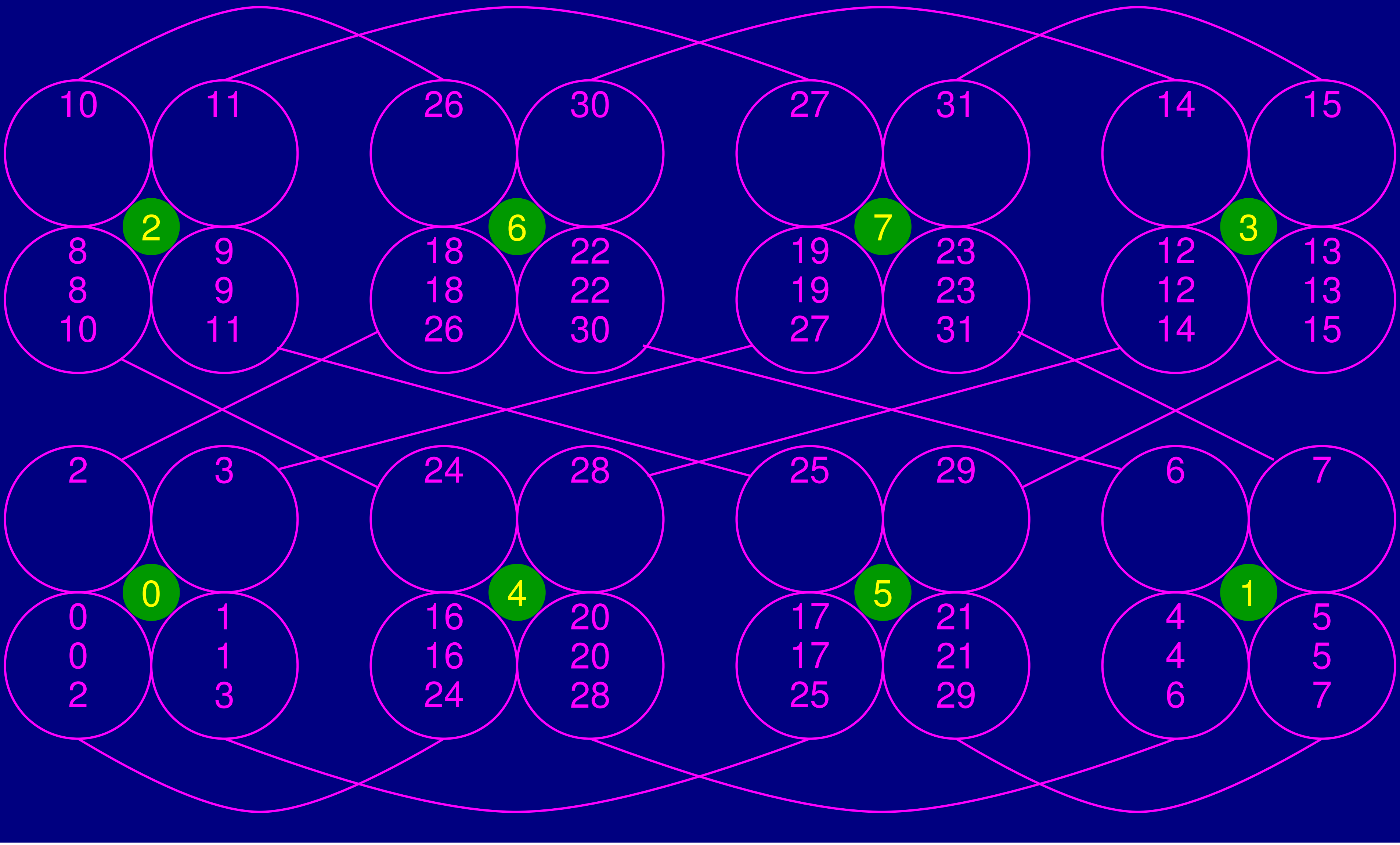
Example (Dimension 1: Vertical)



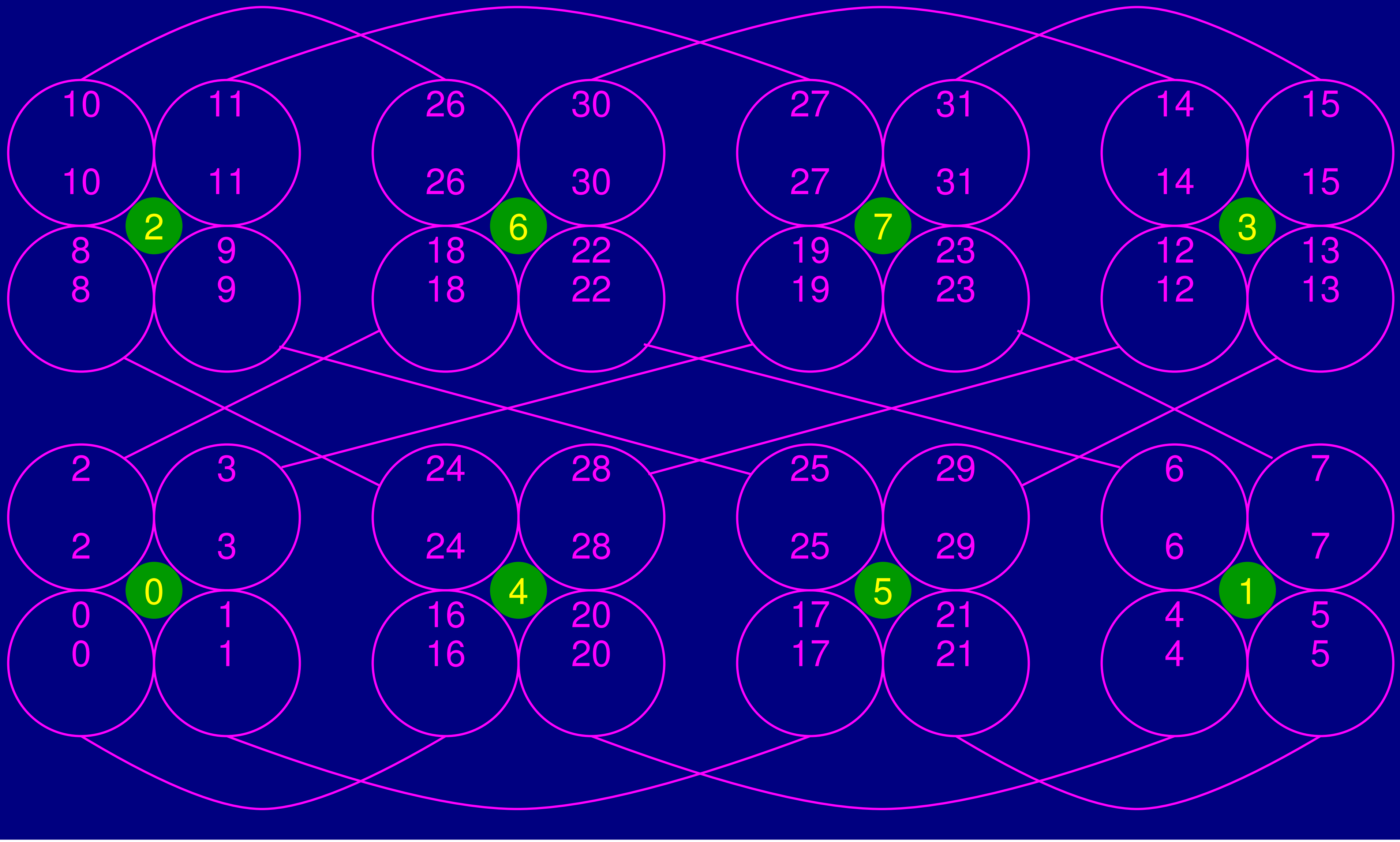
Example (Cross-Edge)



Example (Dimension 0: Horizontal)



Example (Dimension 1: Vertical)



Time of One-to-all Personalized

- $T_1 = t_s + (2^{2m} - 2^m)wt_w$
- $T_2 = mt_s + 2^m(2^m - 1)wt_w$
- $T_3 = t_s + 2^m wt_w$
- $T_4 = mt_s + (2^m - 1)wt_w$
- Total time to complete the all-to-all broadcast $T = T_1 + T_2 + T_3 + T_4 = (1 + \log_2 p)t_s + (p - 1)wt_w$
- Hypercube: $T = (\log_2 p)t_s + (p - 1)wt_w$

Subsection III.5

All-to-All Personalized Communication

Every node sends a unique message to every other node

All-to-All Personalized in Dual-Cube

Using cut-through routing:

1. Send inside cluster
2. Send to other clusters of different class
3. Send to other clusters of same class

Step 1:

- Each node sends $2^m - 1$ messages to the other nodes inside cluster
- Example: node 00000:
  1. $00000 \longrightarrow 00001$
  2. $00000 \longrightarrow 00010$
  3. $00000 \longrightarrow 00001 \longrightarrow 00011$

All-to-All Personalized in Dual-Cube

- There are $2^m - 1$ such nodes inside cluster
- Using cut-through routing:
  - $$T_1 = \sum_{i=1}^{m} (t_s + wt_w + (i-1)t_h) \binom{m}{i}$$
    $$= (2^m - 1)(t_s + wt_w) + gt_h$$
    - $g = \frac{1}{2} m2^m - (2^m - 1)$

All-to-All Personalized in Dual-Cube

Step 2:

- Each node sends 2^{2m} messages to clusters of different classes
- Example: node 00000:
  1. 00000 $\longrightarrow$ 10000
  2. 00000 $\longrightarrow$ 10000 $\longrightarrow$ 10100
  3. 00000 $\longrightarrow$ 10000 $\longrightarrow$ 11000
  4. 00000 $\longrightarrow$ 10000 $\longrightarrow$ 10100 $\longrightarrow$ 11100
  5. 00000 $\longrightarrow$ 00001 $\longrightarrow$ 10001
  6. 00000 $\longrightarrow$ 00001 $\longrightarrow$ 10001 $\longrightarrow$ 10101

All-to-All Personalized in Dual-Cube

7. 00000 → 00001 → 10001 → 11001
8. 00000 → 00001 → 10001 → 10101 → 11101
9. 00000 → 00010 → 10010
10. 00000 → 00010 → 10010 → 10110
11. 00000 → 00010 → 10010 → 11010
12. 00000 → 00010 → 10010 → 10110 → 11110
13. 00000 → 00001 → 00011 → 10011
14. 00000 → 00001 → 00011 → 10011 → 10111
15. 00000 → 00001 → 00011 → 10011 → 11011
16. 00000 → 00001 → 00011 → 10011 → 10111 → 11111

All-to-All Personalized in Dual-Cube

- There are 2^m such clusters with each has 2^m nodes
- Each message travels through a cross-edge
  - The distance of cross-edges contributes $D_{22} = 2^{2m}$
- The distance of routing within 2^m clusters is
  - $D_{23} = 2^m \times \frac{1}{2}m2^m = \frac{1}{2}m2^{2m}$
- Before going through cross-edges, it needs to route in the cluster of the source node. This distance is
  - $D_{21} = 2^m \times \frac{1}{2}m2^m = \frac{1}{2}m2^{2m}$.
- $T_2 = 2^{2m}(t_s + wt_w) + ((D_{21} + D_{22} + D_{23})/2^{2m} - 1)2^{2m}t_h$
  $= 2^{2m}(t_s + wt_w) + m2^{2m}t_h$

All-to-All Personalized in Dual-Cube

Step 3:

- Each node sends 2^{2m} messages to different clusters of same class
  - going out through a cross edge
  - routing within a cluster
  - going out again through a cross edge
  - routing within a cluster
- Example: node 00000:

1. 00000 → 10000 → 10100 → 00100
2. 00000 → 10000 → 10100 → 00100 → 00101

All-to-All Personalized in Dual-Cube

3. 00000 → 10000 → 10100 → 00100 → 00110
4. 00000 → 10000 → 10100 → 00100 → 00101 → 00111

5. 00000 → 10000 → 11000 → 01000
6. 00000 → 10000 → 11000 → 01000 → 01001
7. 00000 → 10000 → 11000 → 01000 → 01010
8. 00000 → 10000 → 11000 → 01000 → 01001 → 01011

9. 00000 → 10000 → 10100 → 11100 → 01100
10. 00000 → 10000 → 10100 → 11100 → 01100 → 01101
11. 00000 → 10000 → 10100 → 11100 → 01100 → 01110
12. 00000 → 10000 → 10100 → 11100 → 01100 → 01101 → 01111

All-to-All Personalized in Dual-Cube

- There are $2^m - 1$ clusters with each has 2^m nodes
- Each message travels through a cross-edge
  - The distance of cross-edges is $D_{31} = (2^m - 1) \times 2^m$
- The distance of routing within $2^m - 1$ clusters is
  - $D_{32} = (2^m - 1) \times \frac{1}{2} m 2^m$
- $D = 2 \times (D_{31} + D_{32})$
- $$\begin{aligned} T_3 &= (2^m - 1) \times 2^m (t_s + w t_w) + (D - (2^m - 1) \times 2^m) t_h \\ &= (2^m - 1) \times 2^m (t_s + w t_w) + (m + 1)(2^m - 1) 2^m t_h \end{aligned}$$

Times: Hypercube vs Dual-Cube

One-to-all broadcast:	
HC	$\log_2 p(t_s + wt_w)$
DC	$(1 + \log_2 p)(t_s + wt_w)$
One-to-all personalized communication:	
HC	$(\log_2 p)t_s + (p - 1)wt_w$
DC	$(1 + \log_2 p)t_s + (p - 1)wt_w$
All-to-all broadcast:	
HC	$(\log_2 p)t_s + (p - 1)wt_w$
DC	$(1 + \log_2 p)t_s + (p - 1)wt_w$
All-to-all personalized communication:	
HC	$(p - 1)(t_s + wt_w) + ((\log_2 p)p/2 - (p - 1))t_h$
DC	$(p - 1)(t_s + wt_w) + ((2 + \log_2 p)p/2 - (p - 1) - \sqrt{2p})t_h$

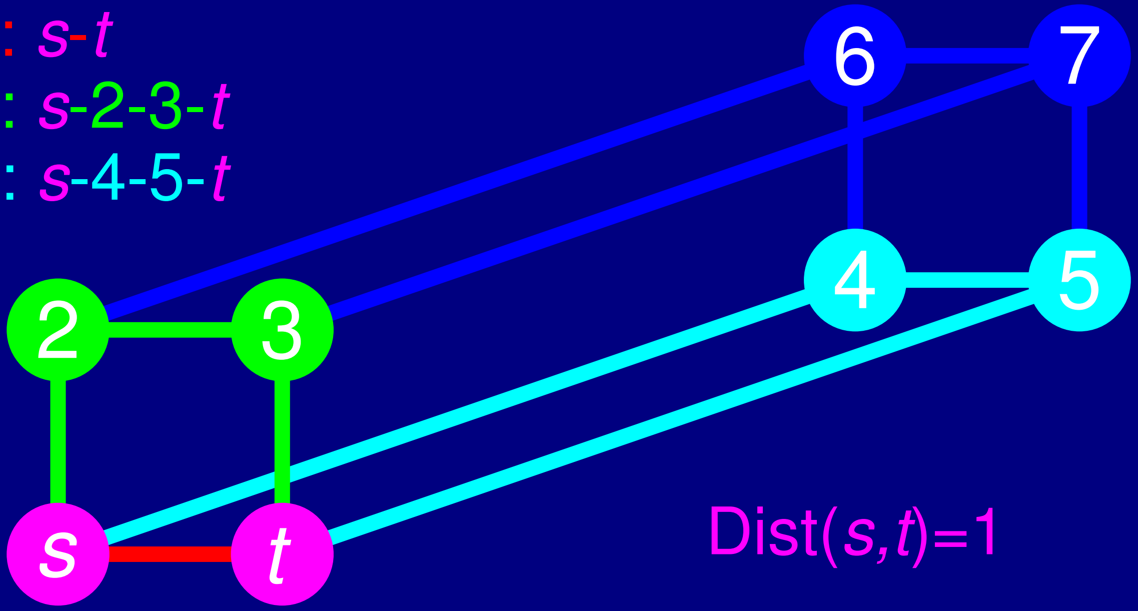
Section IV

Disjoint Paths in Dual-cube

Vertex Disjoint Paths in Hypercube

- Given two nodes *s* and *t*, find multiple paths from *s*→*t*
  - such that no intermediate node is shared
- There are *n* disjoint paths in an *n*-cube:

Path 1: *s*-*t*
Path 2: *s*-2-3-*t*
Path 3: *s*-4-5-*t*

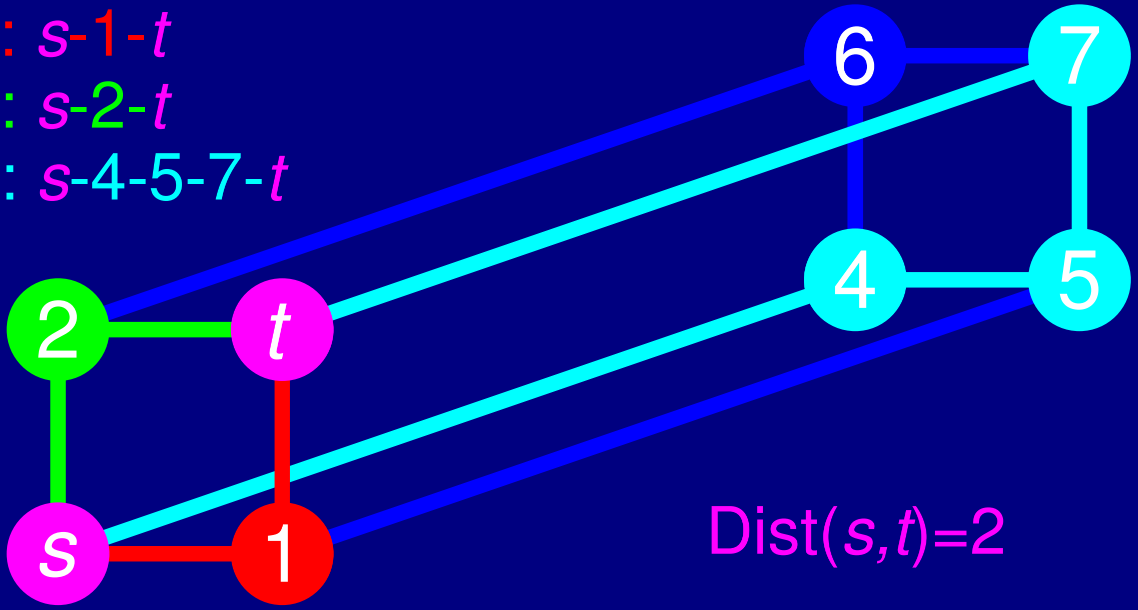


Dist(*s*,*t*)=1

Vertex Disjoint Paths in Hypercube

- Another example of disjoint paths in an 3-cube:

Path 1: *s*-1-*t*
Path 2: *s*-2-*t*
Path 3: *s*-4-5-7-*t*

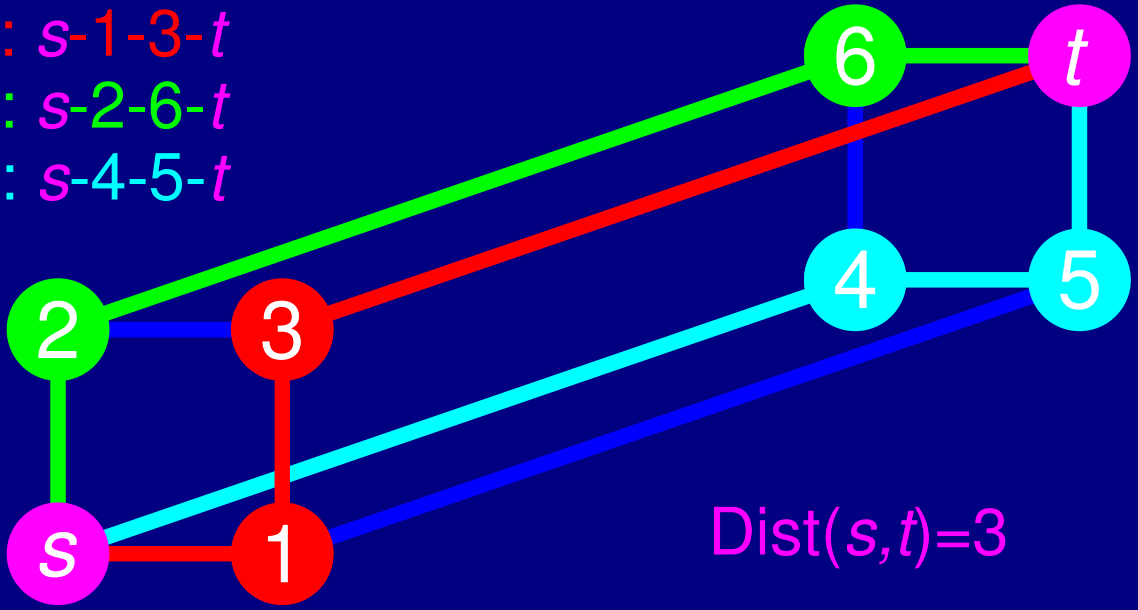


Dist(*s*,*t*)=2

Vertex Disjoint Paths in Hypercube

- Yet another example of disjoint paths in an 3-cube:

Path 1: *s*-1-3-*t*
Path 2: *s*-2-6-*t*
Path 3: *s*-4-5-*t*



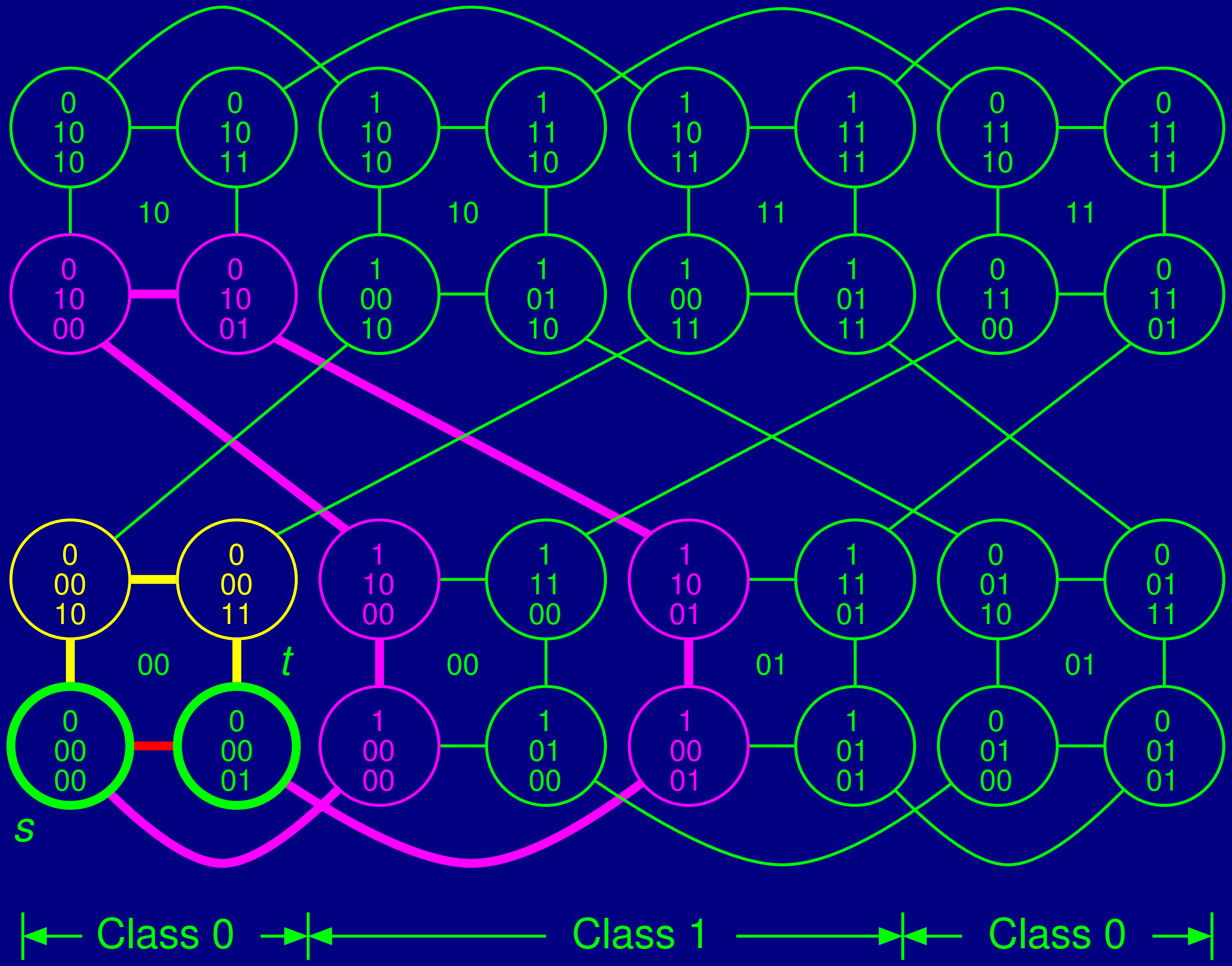
Dist(*s*,*t*)=3

Algorithm of Constructing Disjoint Paths

Case 1: s and t are in same cluster, $C_s = C_t$:

- Construct m disjoint paths inside cluster (m-cube)
- Construct (m+1)th disjoint path:
    - $s \longrightarrow s'$ through across-edge
    - $t \longrightarrow t'$ through across-edge
    - $s' \longrightarrow s'_i$ along with dimension i
    - $t' \longrightarrow t'_i$ along with dimension i
    - $s'_i \longrightarrow s''_i$ through across-edge
    - $t'_i \longrightarrow t''_i$ through across-edge ($C_{s''_i} = C_{t''_i}$)
    - Routing $s''_i \longrightarrow t''_i$ inside cluster

Disjoint Paths: Example (Case 1)

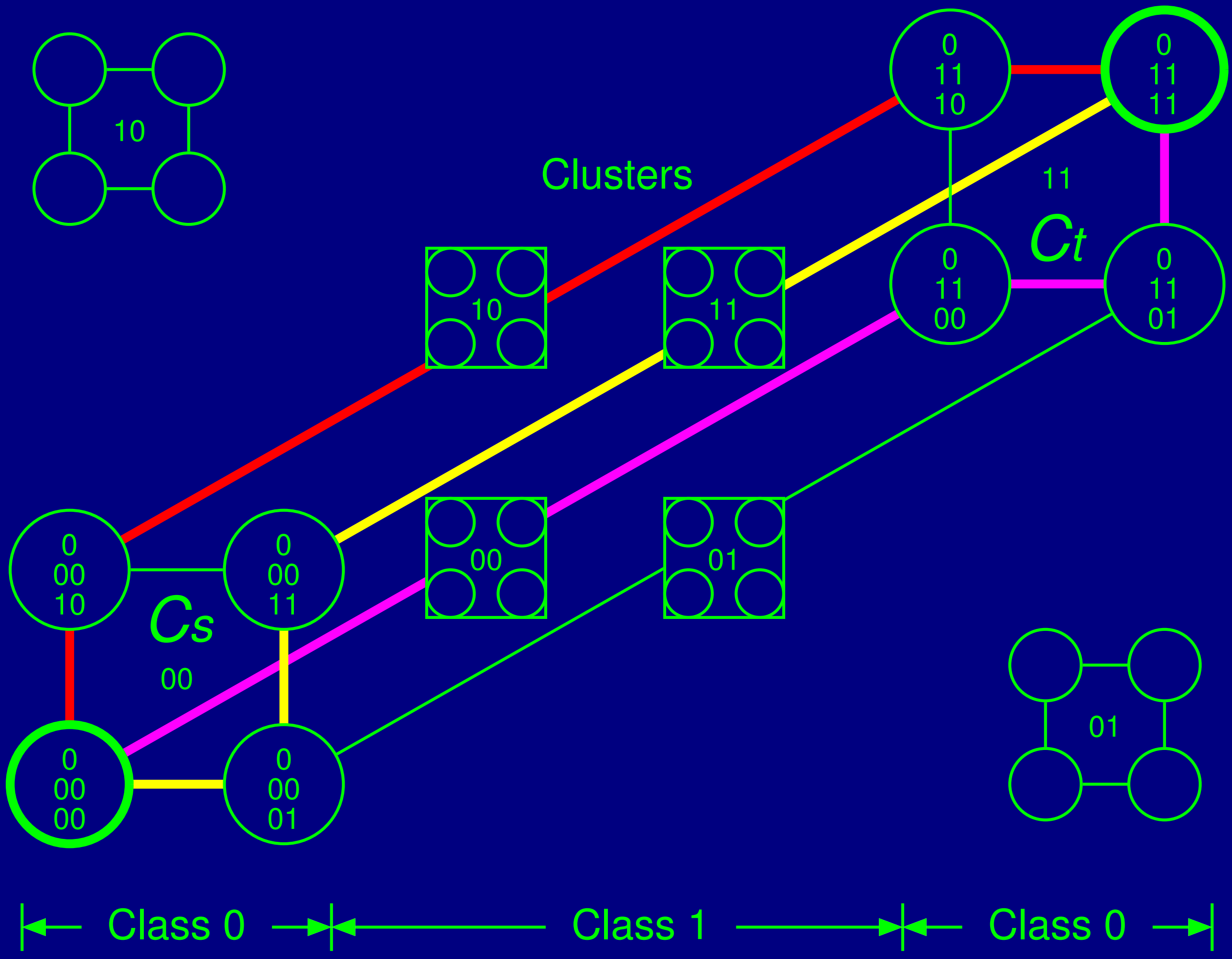


Algorithm of Constructing Disjoint Paths

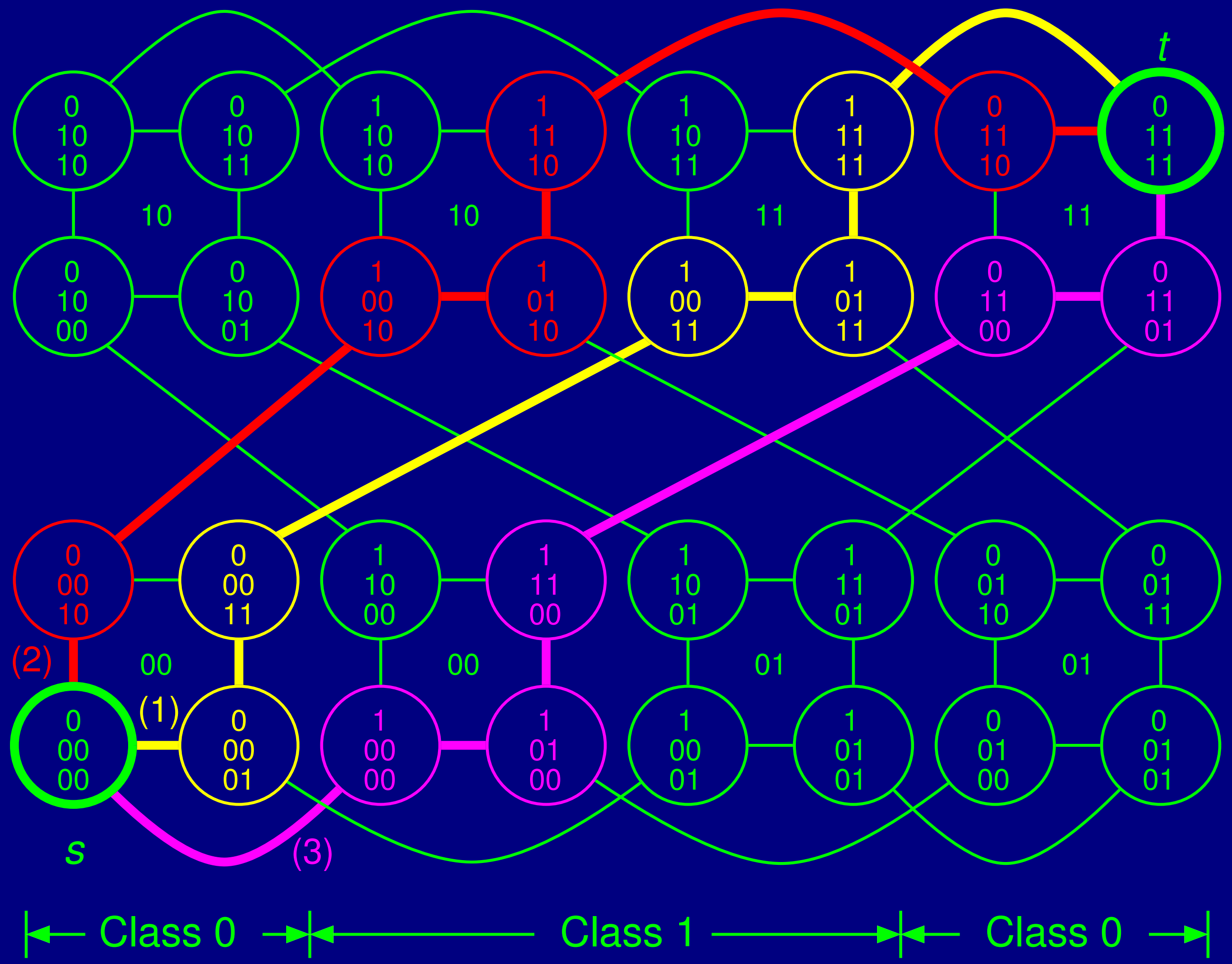
Case 2: s and t are of same class, $C_s \neq C_t$:

- Construct an extended cube:
  - C_s and C_t are 2 m-cubes
  - The class ID of u = class ID of v
  - Connect C_s and C_t to form an $(m+1)$-cube:
    - For $u \in C_s$ and $v \in C_t$ that have same node ID
    - Connect u and v via a cluster C of another class:
      - The cluster ID of C = node ID of u
- Build $m+1$ disjoint paths on the extended $(m+1)$-cube

Extended Cube (Case 2)



Disjoint Paths: Example (Case 2)

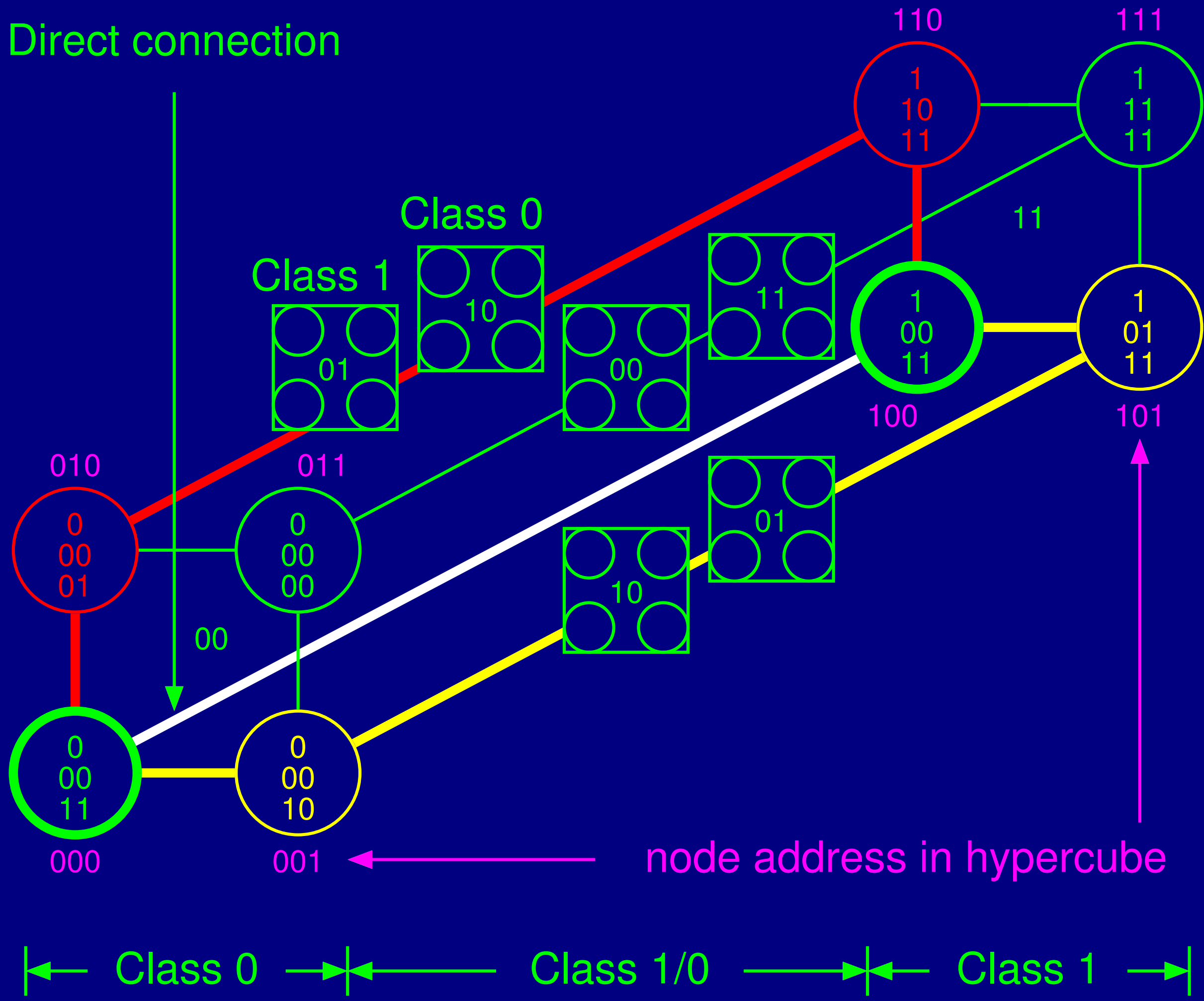


Algorithm of Constructing Disjoint Paths

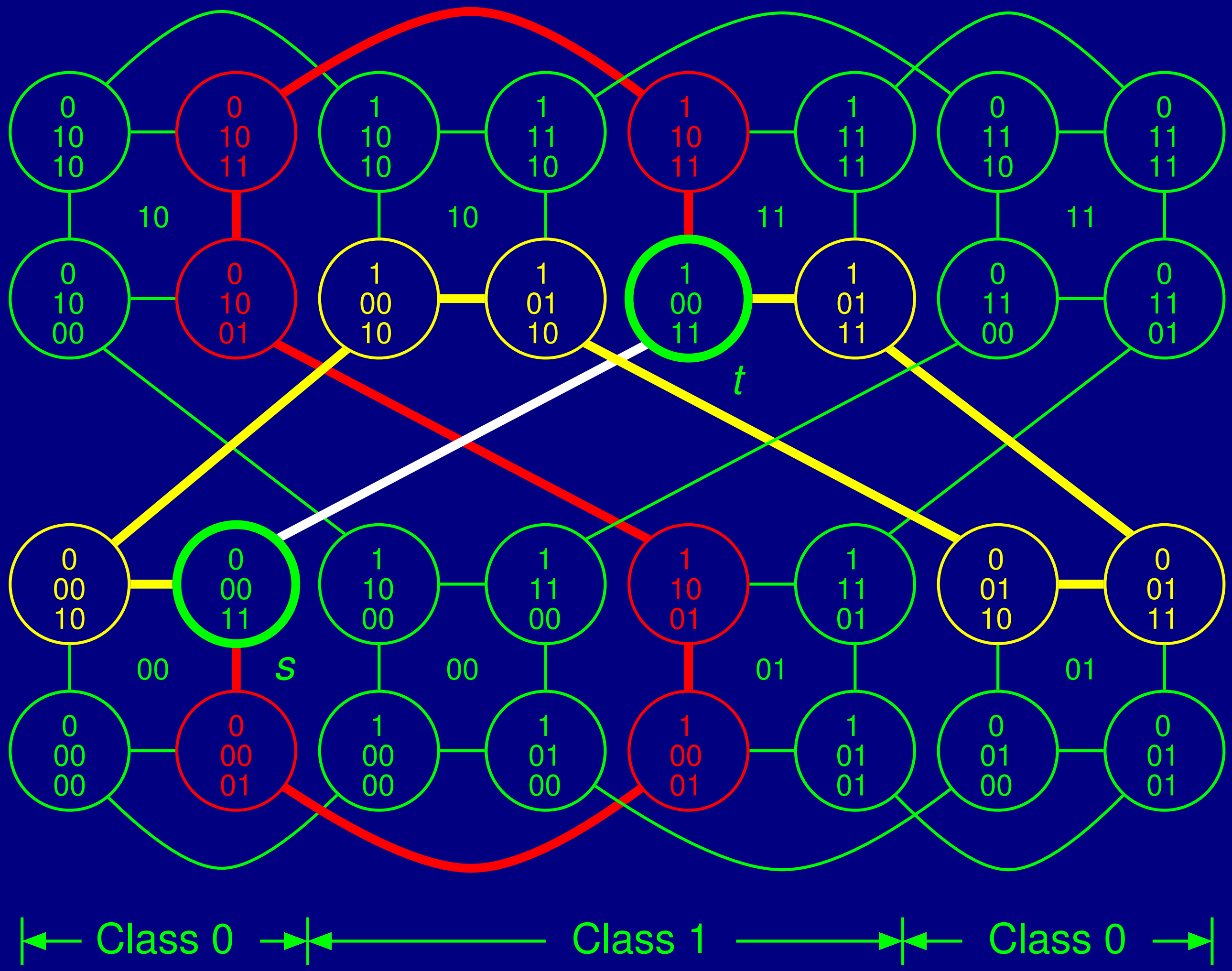
Case 3: s and t are of different classes:

- Construct an extended cube:
  - C_s and C_t are 2 m-cubes
  - The class ID of $u \neq$ class ID of v
  - Connect C_s and C_t to form an $(m{+}1)$-cube:
    - There is a across-edge connects C_s and C_t
    - For other $2^m - 1$ nodes ($u \in C_s$ and $v \in C_t$):
      - $u \longrightarrow u'$ through across-edge
      - $v \longrightarrow v'$ through across-edge
      - There is a across-edge connects $C_{u'}$ and $C_{v'}$
- Build $m+1$ disjoint paths on the extended $(m{+}1)$-cube

Extended Cube (Case 3)



Disjoint Paths: Example (Case 3)



Disjoint Paths in Dual-cube

Theorem: For any two nodes s and t in DC(m), we can find $m + 1$ disjoint paths of length at most $d(s, t) + 6$ in $O(m^2)$ time.

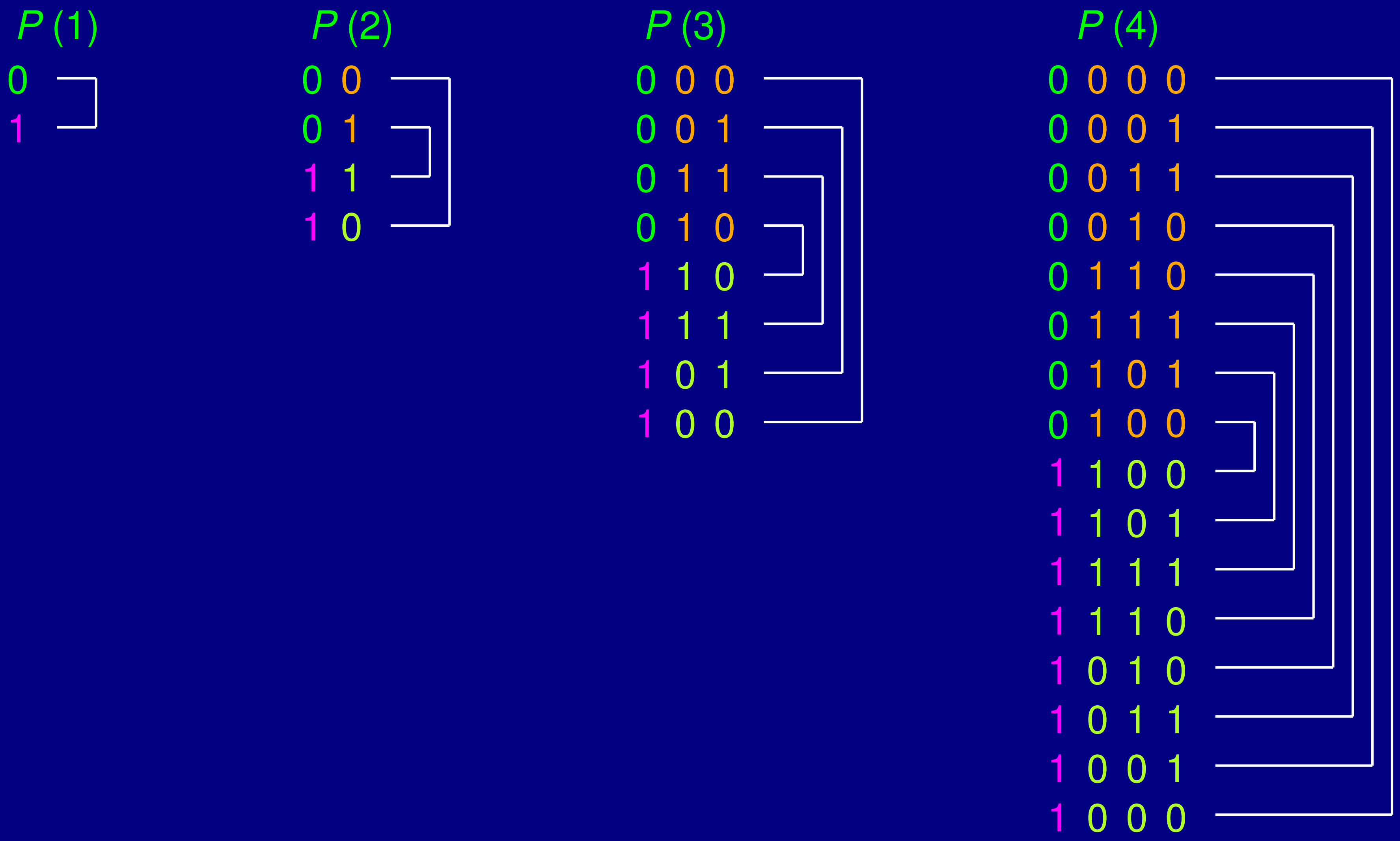
Section V

Hamiltonian Cycle Embedding

Hamiltonian Cycle in Dual-Cube

- A *hamiltonian cycle* of an undirected graph G is a simple cycle that contains every node in G exactly once.
- A graph that contains a hamiltonian cycle is said to be *hamiltonian*. G is *k-link hamiltonian* if it remains hamiltonian after removing any k links.
- It is clear that if graph G is k-connected then G can be at most $(k-2)$-link hamiltonian.
- We show that the $(m+1)$-connected DC(m) is $(m-1)$-link hamiltonian.

Binary Reflected Gray Code



$P(1)$

0
1

$P(2)$

0 0
0 1
1 1
1 0

$P(3)$

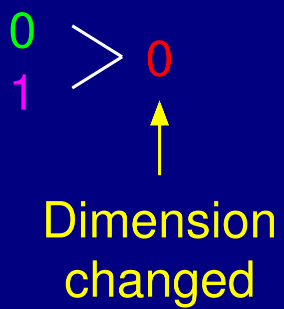
0 0 0
0 0 1
0 1 1
0 1 0
1 1 0
1 1 1
1 0 1
1 0 0

$P(4)$

0 0 0 0
0 0 0 1
0 0 1 1
0 0 1 0
0 1 1 0
0 1 1 1
0 1 0 1
0 1 0 0
1 1 0 0
1 1 0 1
1 1 1 1
1 1 1 0
1 0 1 0
1 0 1 1
1 0 0 1
1 0 0 0

Reflected Dimension List

$D\,(1)$

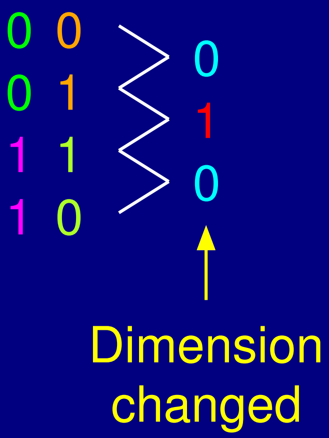


0
1

0

Dimension changed

$D\,(2)$

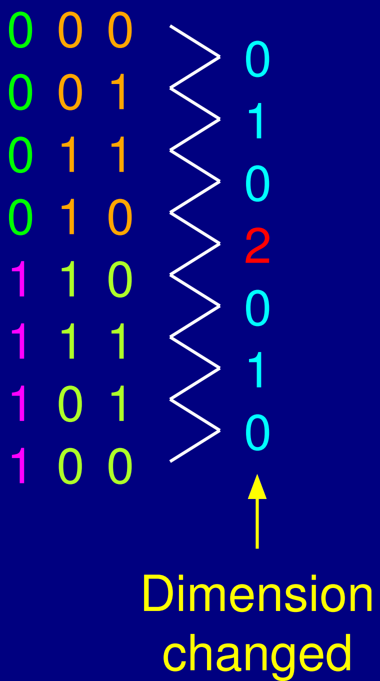


0 0
0 1
1 1
1 0

0
1
0

Dimension changed

$D\,(3)$

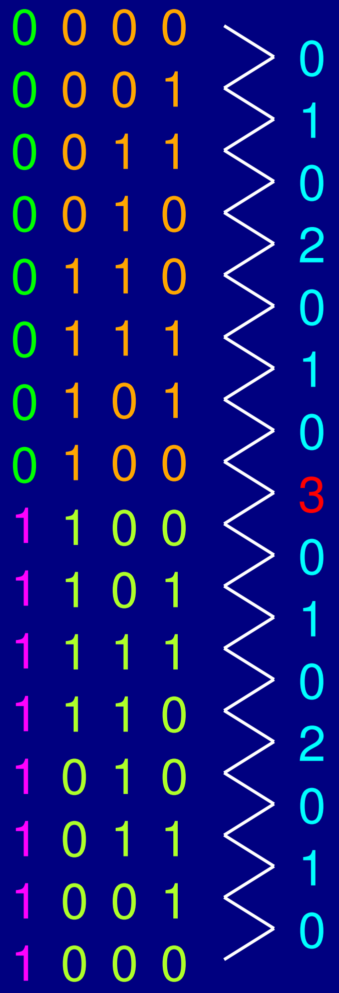


0 0 0
0 0 1
0 1 1
0 1 0
1 1 0
1 1 1
1 0 1
1 0 0

0
1
0
2
0
1
0

Dimension changed

$D\,(4)$



0 0 0 0
0 0 0 1
0 0 1 1
0 0 1 0
0 1 1 0
0 1 1 1
0 1 0 1
0 1 0 0
1 1 0 0
1 1 0 1
1 1 1 1
1 1 1 0
1 0 1 0
1 0 1 1
1 0 0 1
1 0 0 0

0
1
0
2
0
1
0
3
0
1
0
2
0
1
0

$D\,(1) = 0;$
$D\,(n) = (D\,(n\text{-}1),\ n\text{-}1,\ D\,(n\text{-}1));$

Building a Hamiltonian Cycle in Cube

```
Algorithm cubeHC($n$)
begin                                  /* build a hamiltonian cycle $P$ in an $n$-cube */
    $D(n) = \mathrm{DL}(n)$;           /* $D(n)$ is the reflected dimension list */
    $w = 0$;                           /* starting from node 0 */
    $P = w$;                           /* $P$ is the hamiltonian cycle */
    for each dimension number $i$ in $D(n)$ do
        $w = w \oplus 2^i$;            /* find the next node */
        $P = P : w$;                   /* add the node into $P$ */
    endfor
end
Procedure DL($n$)
begin                                  /* build a reflected dimension list for an $n$-cube */
    if ($n == 1$) return (0);
    else return ($\mathrm{DL}(n-1), n-1, \mathrm{DL}(n-1)$);
end
```

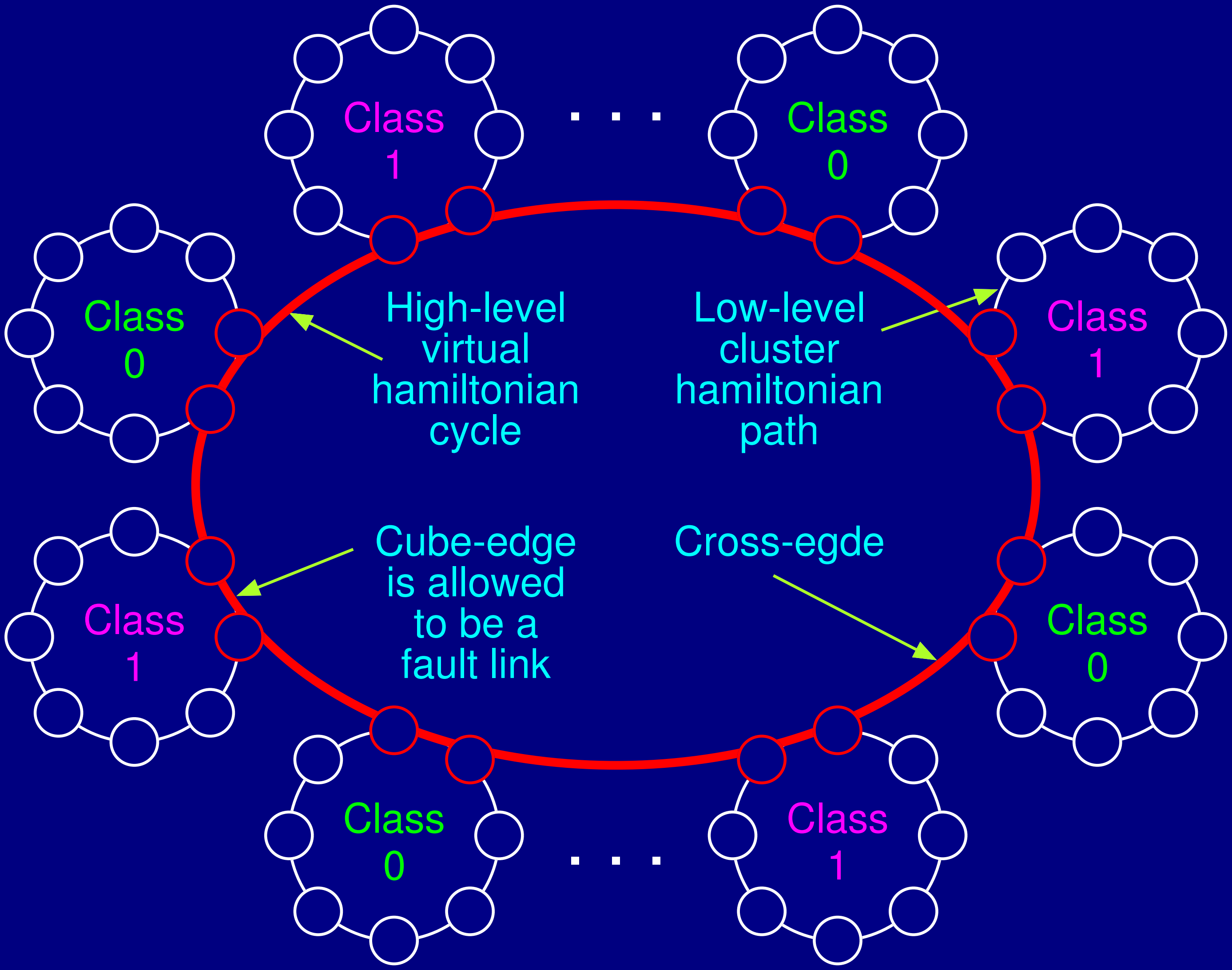
Hamiltonian Cycle in Dual-Cube

- A hamiltonian cycle in a DC(m) can be constructed:
  1. We build a *virtual hamiltonian cycle*, $V(m)$, which connects all the clusters with only two neighboring nodes from each cluster.
  2. In each cluster we replace the edge $e = (u : v)$ with a hamiltonian path $(u \longrightarrow v)$ to connect all the nodes in the cluster to form a hamiltonian cycle in DC(m).
- The virtual hamiltonian cycle in a DC(m) contains equal numbers of cube-edges and cross-edges; the cube-edges and the cross-edges are interleaved.

Hamiltonian Cycle in Dual-Cube

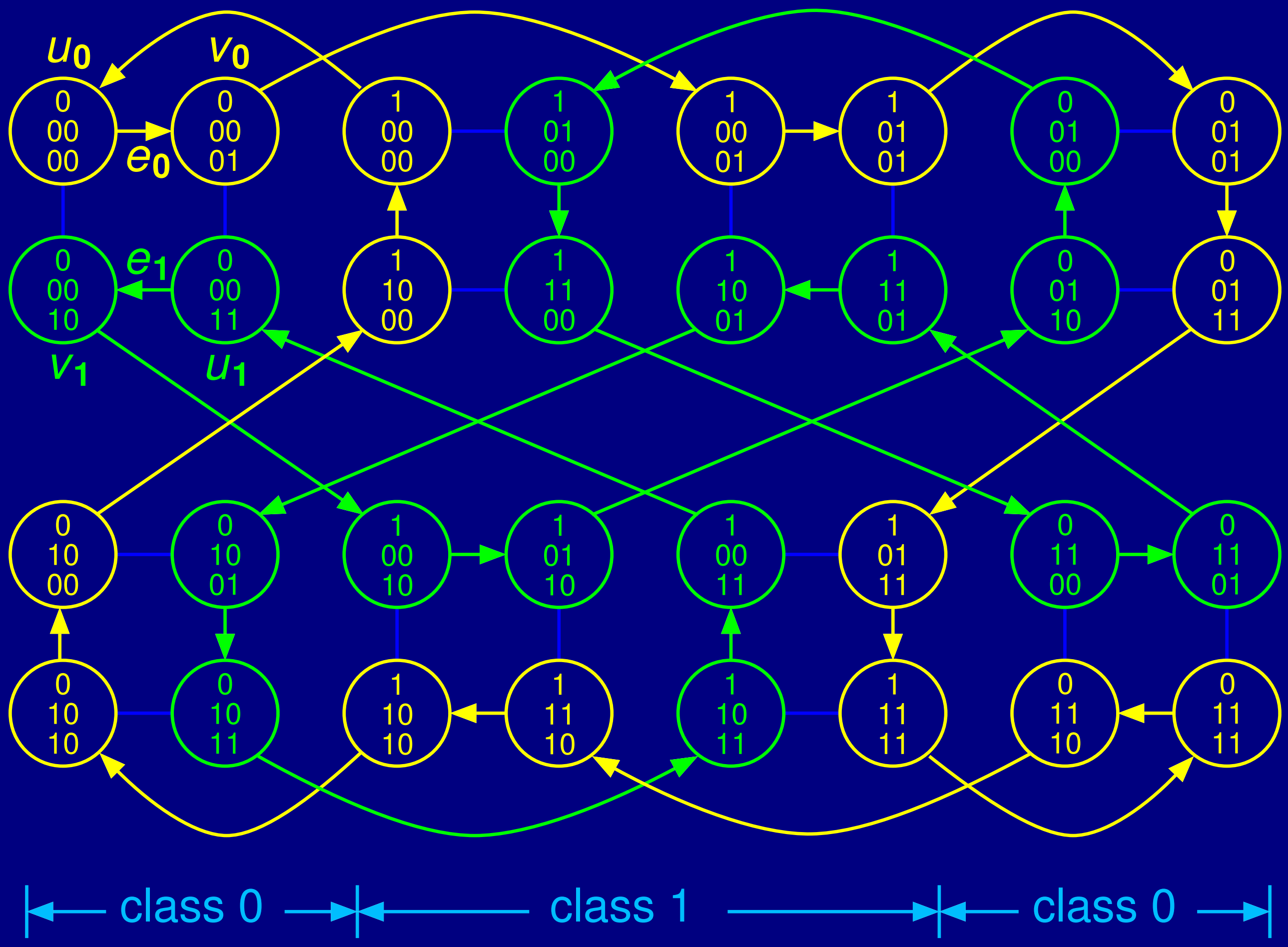
```
Algorithm dualcubeHC($m$)
 begin                                  /* build a hamiltonian cycle $P$ in DC($m$) */
     $DD(m) = \mathrm{DDL}(m)$;
     $EDD(m) = (DD(m), m-1, m-1)$;
     $u = 0$;
     for each dimension number $i$ in $EDD(m)$ do
         if ($u$ is of class 0) $v = u \oplus 2^i$; else $v = u \oplus 2^{m+i}$;
         $P' = \mathrm{cubeHP}(m, u, v)$;  $P = P : P'$;  $u = v \oplus 2^{2m}$;
     endfor
 end
 Procedure DDL($m$)
 begin                             /* build an double-dimension list for a DC($m$) */
     if ($m == 1$) return $(0, 0)$;
     else return $(\mathrm{DDL}(m-1), m-1, m-1, \mathrm{DDL}(m-1))$;
 end
```

Hamiltonian Cycle in Dual-Cube



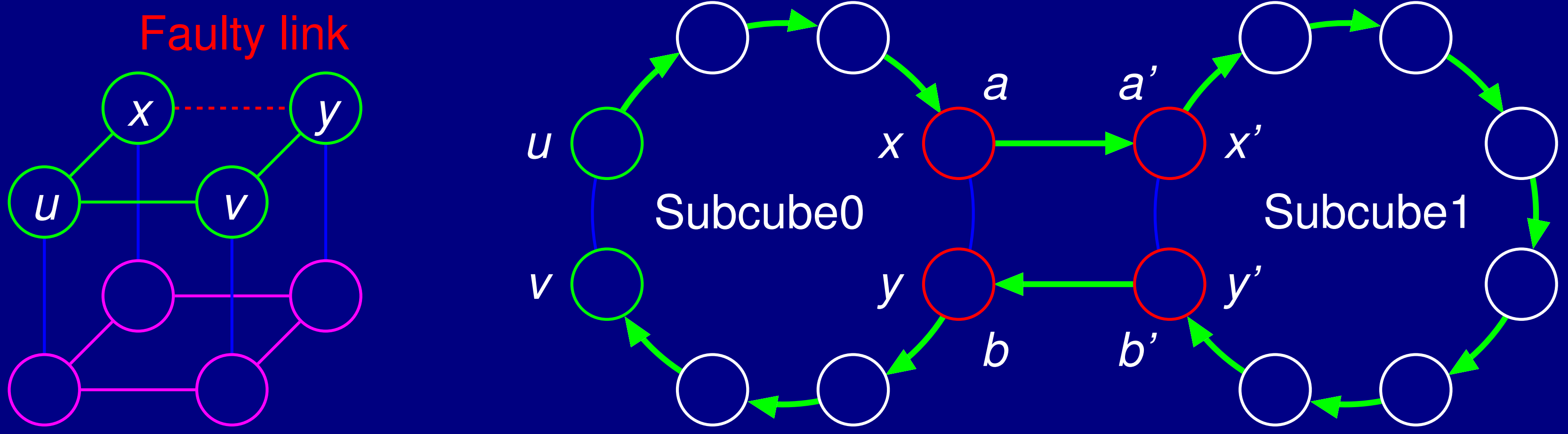
Disjoint Hamiltonian Cycles in Dual-cube

Theorem: *There are 2^{m-1} disjoint virtual hamiltonian cycles in a DC(m).*



Fault-Free Hamiltonian Cycle in DC

- Lemma: *Given any link $e = (u : v)$ in an n-cube with $n - 2$ faulty links, there is a fault-free hamiltonian path $(u \to v)$.*



(a) Subcube dividing (b) Fault-free path

- Theorem: *If a DC(m) contains $m - 1$ faulty links, there is a hamiltonian cycle in the DC(m).*

Proof

- There is a high-level virtual hamiltonian cycle in a DC(m) with $m - 1$ faulty links.
  - Case 1: All the $m - 1$ faulty links appear in a same cluster.
    - Because an m-cube is ($m-2$)-link hamiltonian, there is a fault-free hamiltonian path in that cluster.
    - Let u and v be the first node and the last node of the path, respectively.
    - Then a high-level virtual hamiltonian cycle containing edge ($u : v$) can be built easily because there is no any faulty link outside the cluster.
  - Case 2: Each cluster contains at most $m - 2$ faulty links.
    - The number of disjoint virtual hamiltonian cycles in a cluster is 2^{m-1}, greater than the total number of faulty links which is $m - 1$ for any m.
    - Therefore, there is at least a virtual hamiltonian cycle which does not contain faulty cross-edge.
    - Meanwhile, a fault-free hamiltonian path ($u \longrightarrow v$) in each cluster can be built, where ($u : v$) is a cube-edge in the virtual hamiltonian cycle.
- A fault-free hamiltonian cycle in the DC(m) can be built by replacing each cube-edge ($u : v$) in the virtual hamiltonian cycle with the hamiltonian path ($u \longrightarrow v$) in each cluster.

Section VI

Fault-Tolerant Routing

Fault-Tolerant Routing in Dual-Cube

- Large number of faulty nodes in networks
- Find a fault-free path from source to destination
- Local-information-based:
  - Each node knows only its neighbors’ status
  - No global information of the network is required
- Algorithm runs in linear time
- Builds routing paths of nearly optimal length
- Two algorithms:
  - Adaptive-subcube-based
  - Binomial-Tree-based

Subsection VI.1

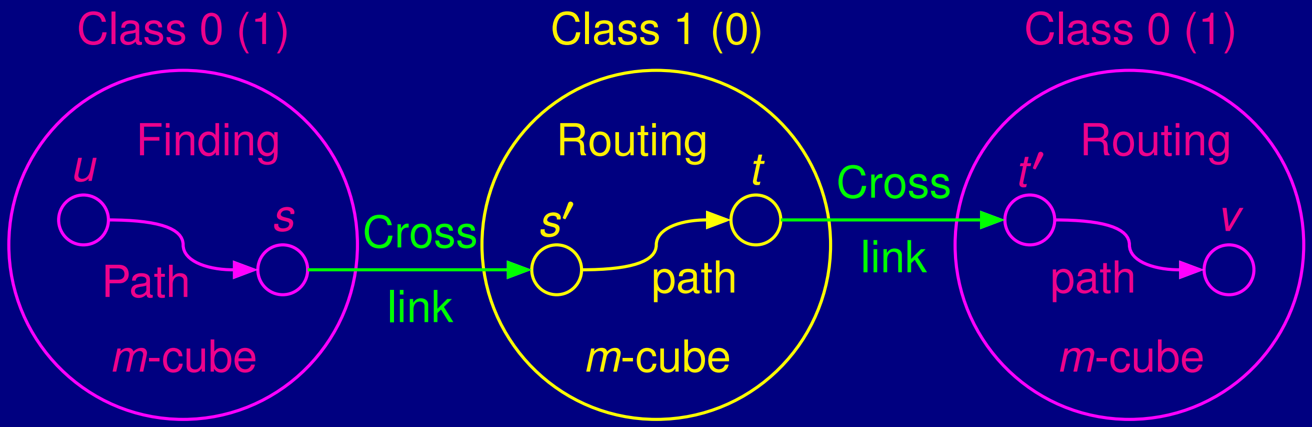
Adaptive-Subcube Fault-Tolerant Routing

Fault Tolerant Routing in Hypercube

- Locally *k*-subcube-connected hypercube
  - In a *k*-subcube, if less than half of the nodes in *k*-subcube are faulty then the nonfaulty nodes of the *k*-subcube make a connected graph.
- Routing *u* to *v* through *k*-subcube
  - Routing first $n - k$ dimensions
    - For each dimension, routing in *k*-subcube with Breadth-First Search
  - Routing the last *k* dimensions
    - For all dimensions, routing to *v* in *k*-subcube with Breadth-First Search
- Adaptive: select a suitable dimension to route

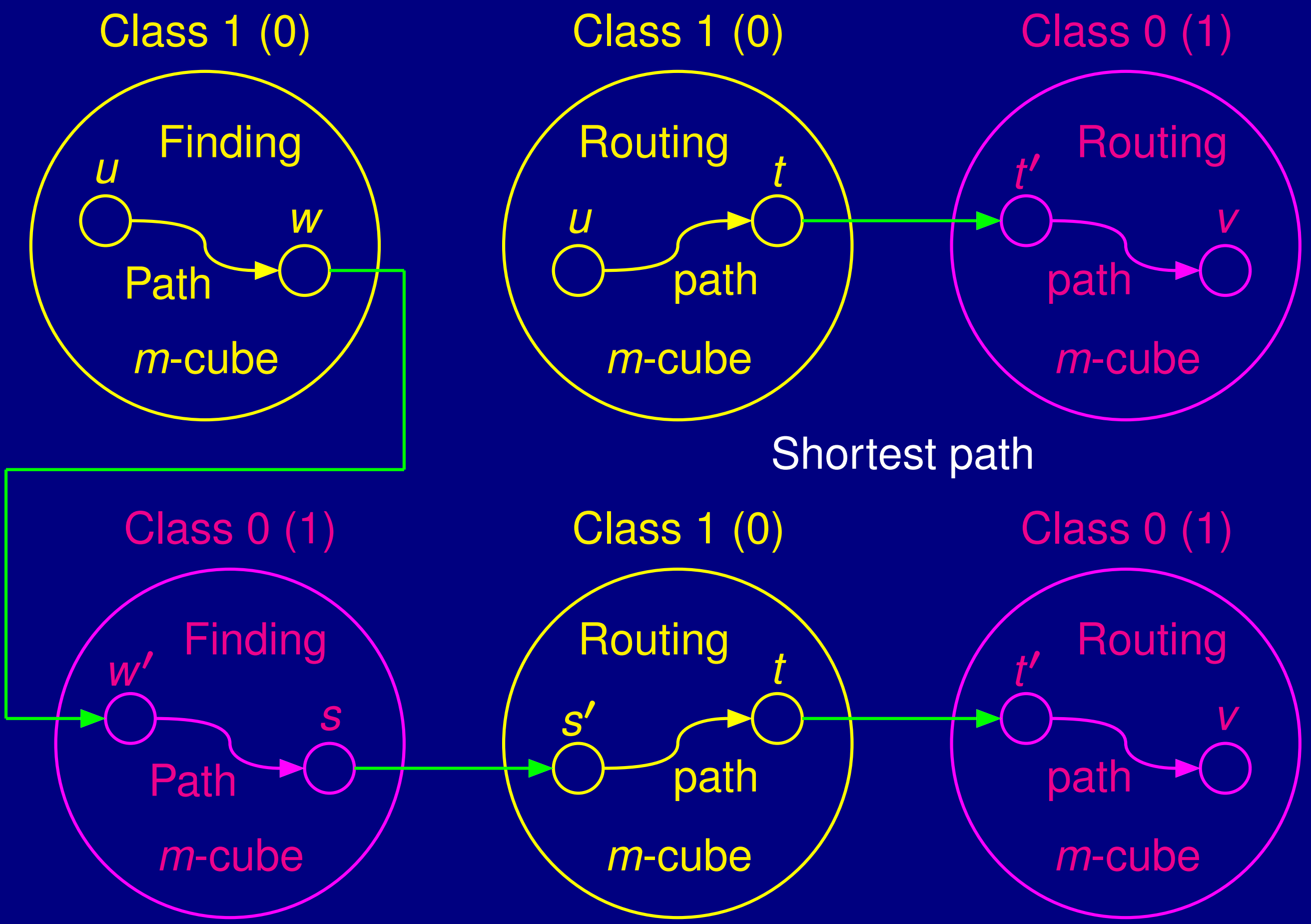
Fault Tolerant Routing in Dual-Cube

- Case 1: Two nodes are in a same cluster
  - This is the simplest case
  - Apply the hypercube routing algorithm directly
  - Our algorithm does not go outside
- Case 2: Two nodes are of same class



Fault Tolerant Routing in Dual-Cube

- Case 3: Two nodes are of different classes



Fault Tolerant Routing in Dual-Cube

		class id	cluster id	node id
u	$=$	(1,	u_cluster_id,	u_node_id)
finding w	$=$	(1,	u_cluster_id,	w_node_id)
w'	$=$	(0,	w_node_id,	u_cluster_id)
finding s	$=$	(0,	w_node_id,	s_node_id)
s'	$=$	(1,	s_node_id,	w_node_id)
routing to t	$=$	(1,	s_node_id,	v_cluster_id)
t'	$=$	(0,	v_cluster_id,	s_node_id)
routing to v	$=$	(0,	v_cluster_id,	v_node_id)

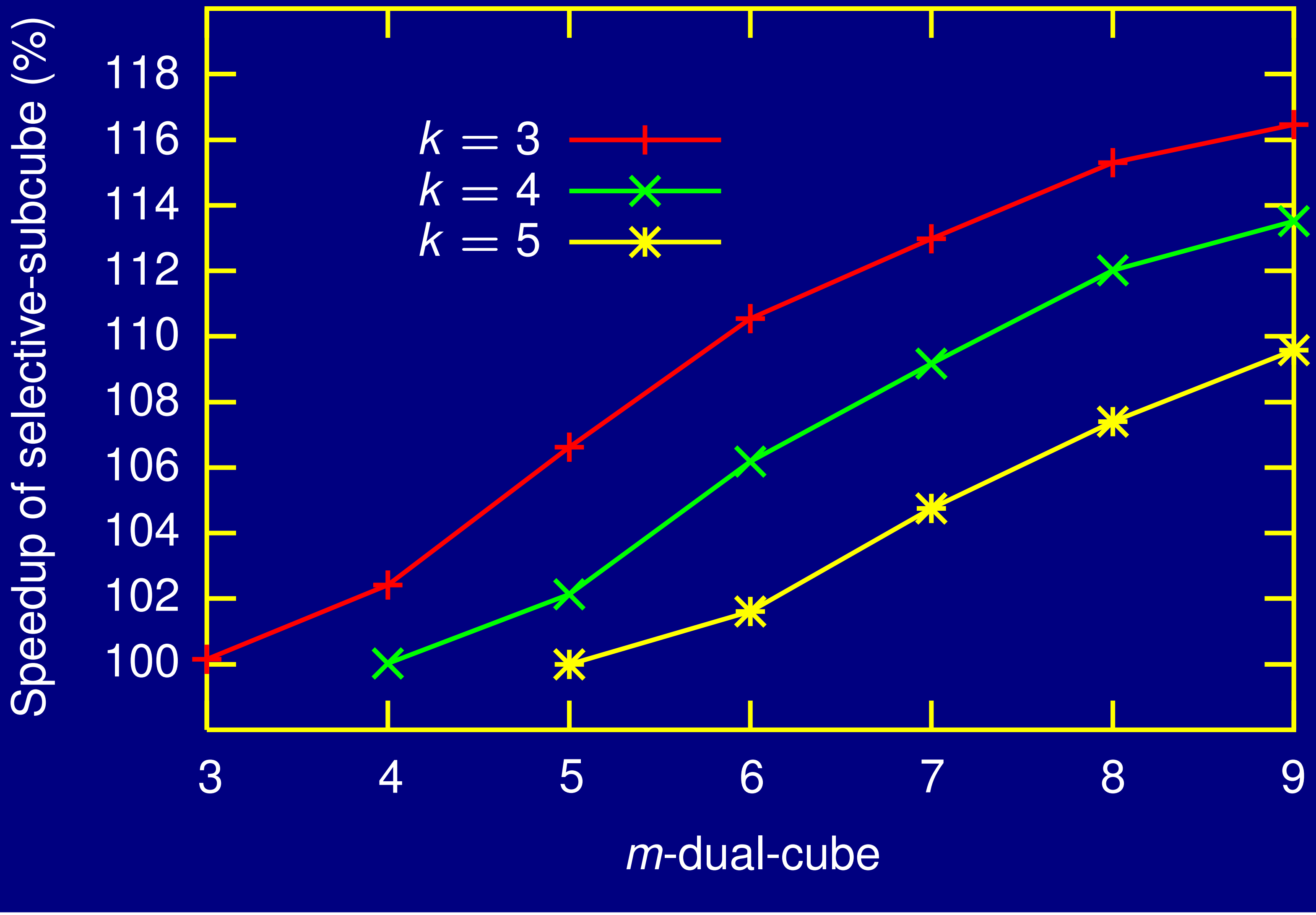
The Time Complexity

- Three cases:
  - u and v are in a same cluster;
    - $t_1 = m2^k$
  - u and v are in different clusters of a same class;
    - $t_2 = 2^k m2^k + m2^k$
  - u and v are of different classes.
    - $t_3 \leq m2^k + 2^k + 2^k m2^k + m2^k$
- Therefore, the time complexity is bounded by
  - $O(\sum_{i=1}^{3} t_i \times p_i) = O(m2^{2k})$ (k is small)

Experimental Results

- Uniform probability distribution of node failures
  - Each node has an equal and independent failure probability p_f
- Seven m-dual-cubes ($m = 3, 4, 5, 6, 7, 8, 9$)
  - $k = 3, 4, 5$, and m
    - Change p_f from 0% to 90%, steped by 10%
      - Test 10,000 times to get the average results
- Two versions are simulated
  - Fixed-subcube
  - Adaptive-subcube

Speedup of Adaptive-Subcube



Performance Parameters

- k: the size of k-subcube
- $p_f(\%)$: the node failure probability
- $p_s(\%)$: the ratio of successful routings
- n_s: the number of successful routings
- n_f: the number of fault routings
- e_m: the maximum number of extra distance
- $e_p(\%)$: the average ratio of the length of the constructed routing path over the length of shortest path of the given two nodes

Routing in 9-Dual-Cubes

k	$p_f(\%)$	$p_s(\%)$	n_s	n_f	e_m	$e_p(\%)$
3	10	99.84	9984	16	12	104.01
3	20	98.50	9850	150	12	109.14
3	30	90.93	9093	907	16	115.04
3	40	70.21	7021	2979	20	121.08
3	50	35.06	3506	6494	18	124.37
3	60	7.97	797	9203	16	120.41
3	70	1.12	112	9888	10	114.09
3	80	0.10	10	9990	2	102.86
3	90	0.02	2	9998	0	100.00

Routing in 9-Dual-Cubes

k	$p_f(\%)$	$p_s(\%)$	n_s	n_f	e_m	$e_p(\%)$
4	10	99.94	9994	6	10	103.98
4	20	99.56	9956	44	12	109.30
4	30	97.20	9720	280	20	115.81
4	40	89.44	8944	1056	20	123.46
4	50	66.90	6690	3310	28	130.83
4	60	25.75	2575	7425	24	135.72
4	70	2.97	297	9703	20	125.83
4	80	0.21	21	9979	8	115.62
4	90	0.02	2	9998	0	100.00

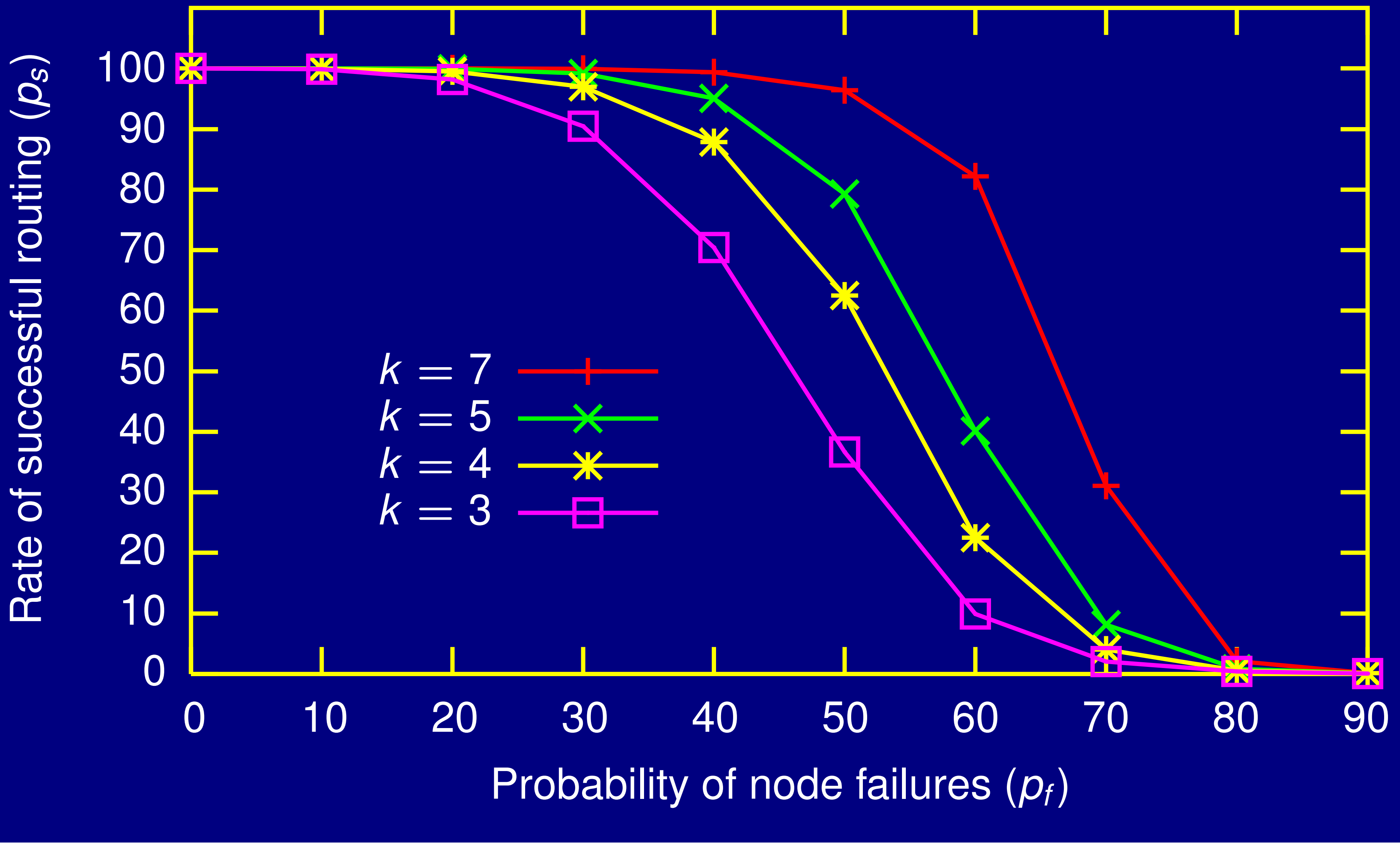
Routing in 9-Dual-Cubes

k	$p_f(\%)$	$p_s(\%)$	n_s	n_f	e_m	$e_p(\%)$
5	10	100.00	10000	0	8	103.97
5	20	99.88	9988	12	12	109.28
5	30	98.97	9897	103	20	116.20
5	40	95.83	9583	417	20	124.52
5	50	84.40	8440	1560	24	132.97
5	60	51.38	5138	4862	40	141.31
5	70	8.63	863	9137	26	141.75
5	80	0.37	37	9963	8	123.17
5	90	0.02	2	9998	0	100.00

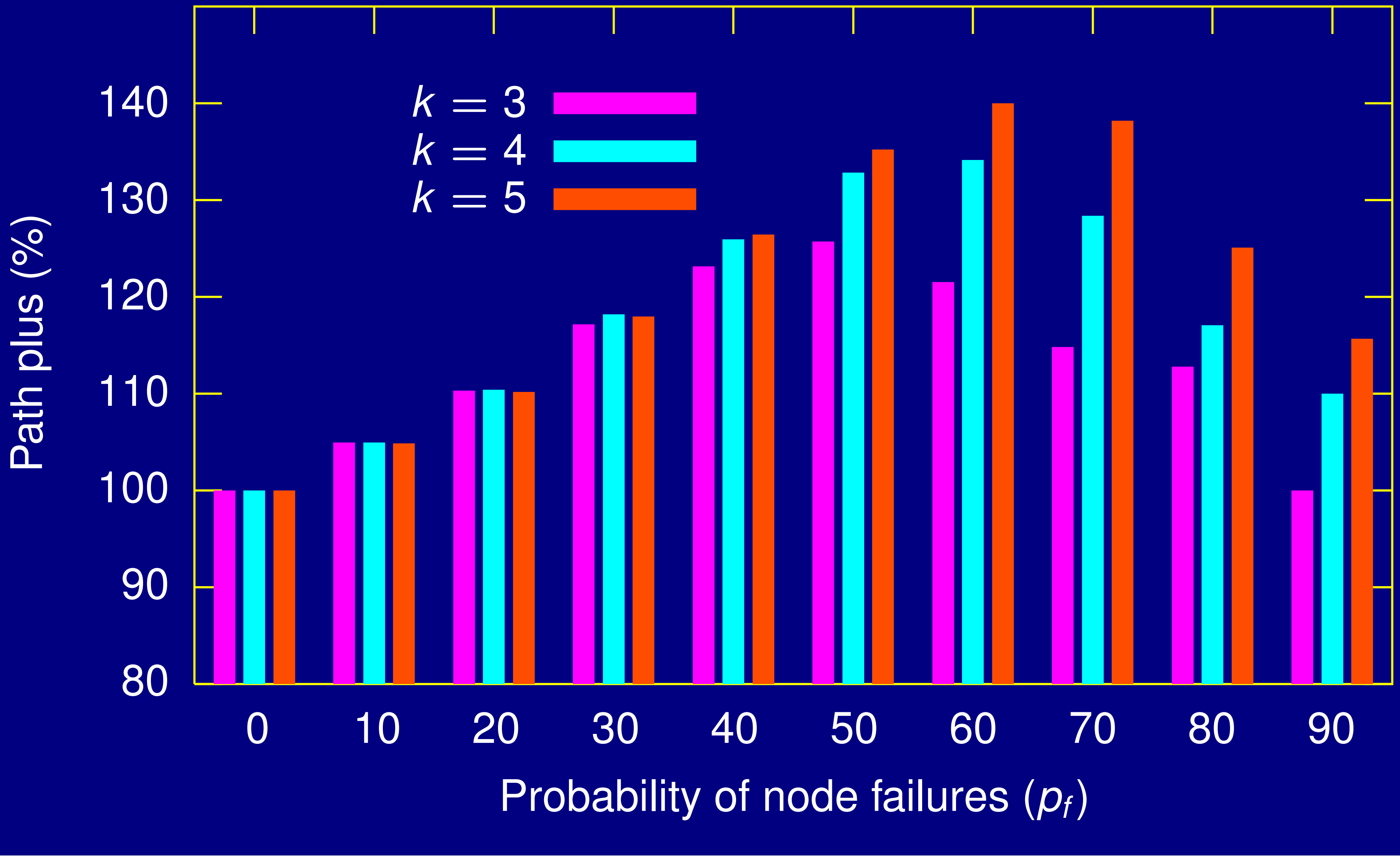
Routing in 9-Dual-Cubes

k	$p_f(\%)$	$p_s(\%)$	n_s	n_f	e_m	$e_p(\%)$
9	10	100.00	10000	0	6	103.52
9	20	100.00	10000	0	6	107.04
9	30	99.99	9999	1	10	110.79
9	40	99.91	9991	9	10	115.57
9	50	99.22	9922	78	14	121.74
9	60	96.27	9627	373	18	131.21
9	70	81.18	8118	1882	28	145.25
9	80	11.57	1157	8843	42	156.00
9	90	0.03	3	9997	8	138.10

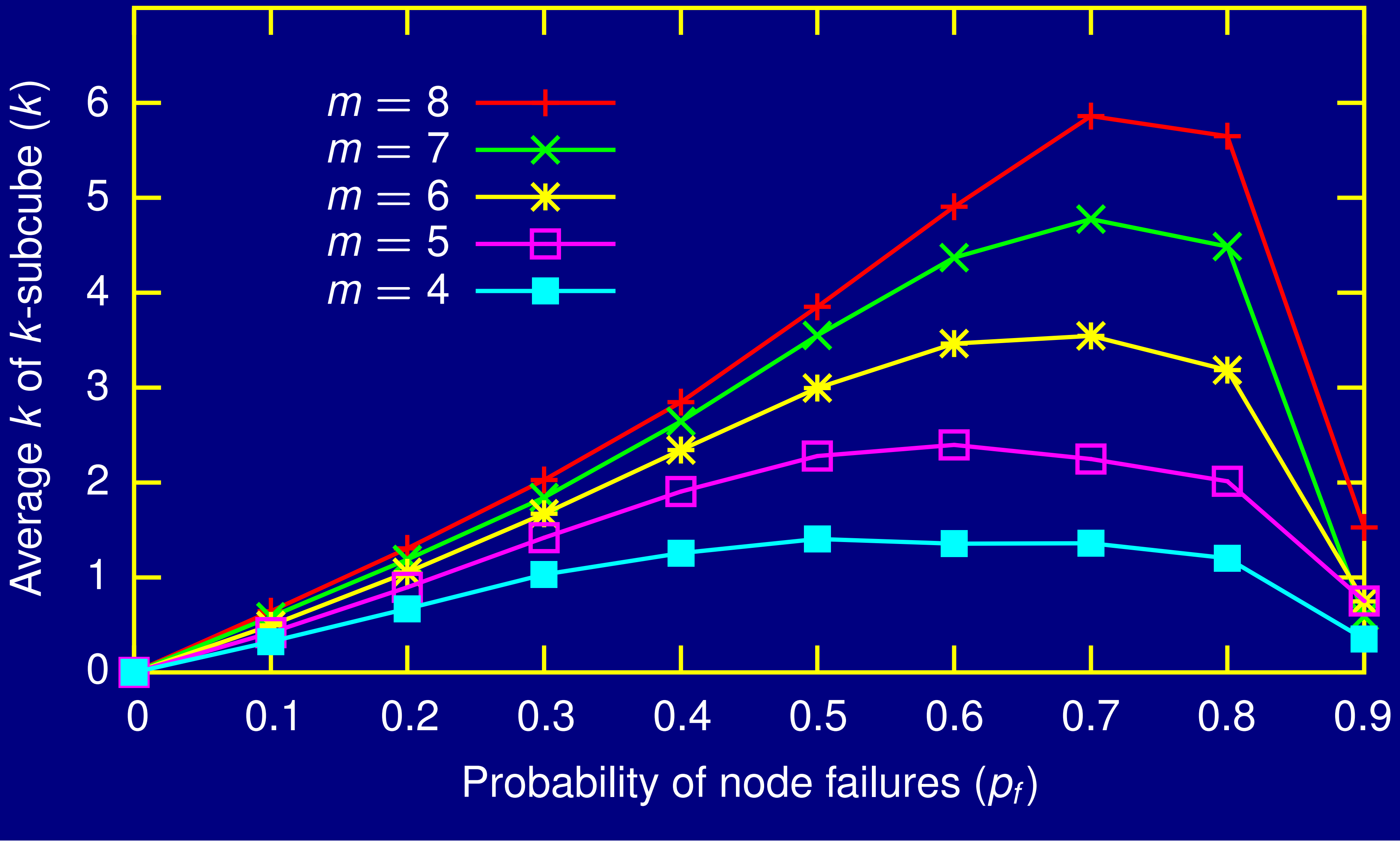
Successful Routing Rate (m = 7)



Ratio of Path Plus (m = 7)



Average k


$m = 8$
$m = 7$
$m = 6$
$m = 5$
$m = 4$
Average k of k-subcube (k)
0
1
2
3
4
5
6
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
Probability of node failures (p_f)

Summary

- We gave a fault-tolerant routing algorithm in dual-cube with a large amount of faulty nodes.
  - Requires only local information about the status of failures
  - Runs at nearly linear time.
  - Simulation results:
    - Dual-cube with 32,768 nodes
    - Contains up to 20 percent faulty nodes
    - Success rate: 99.5 percent

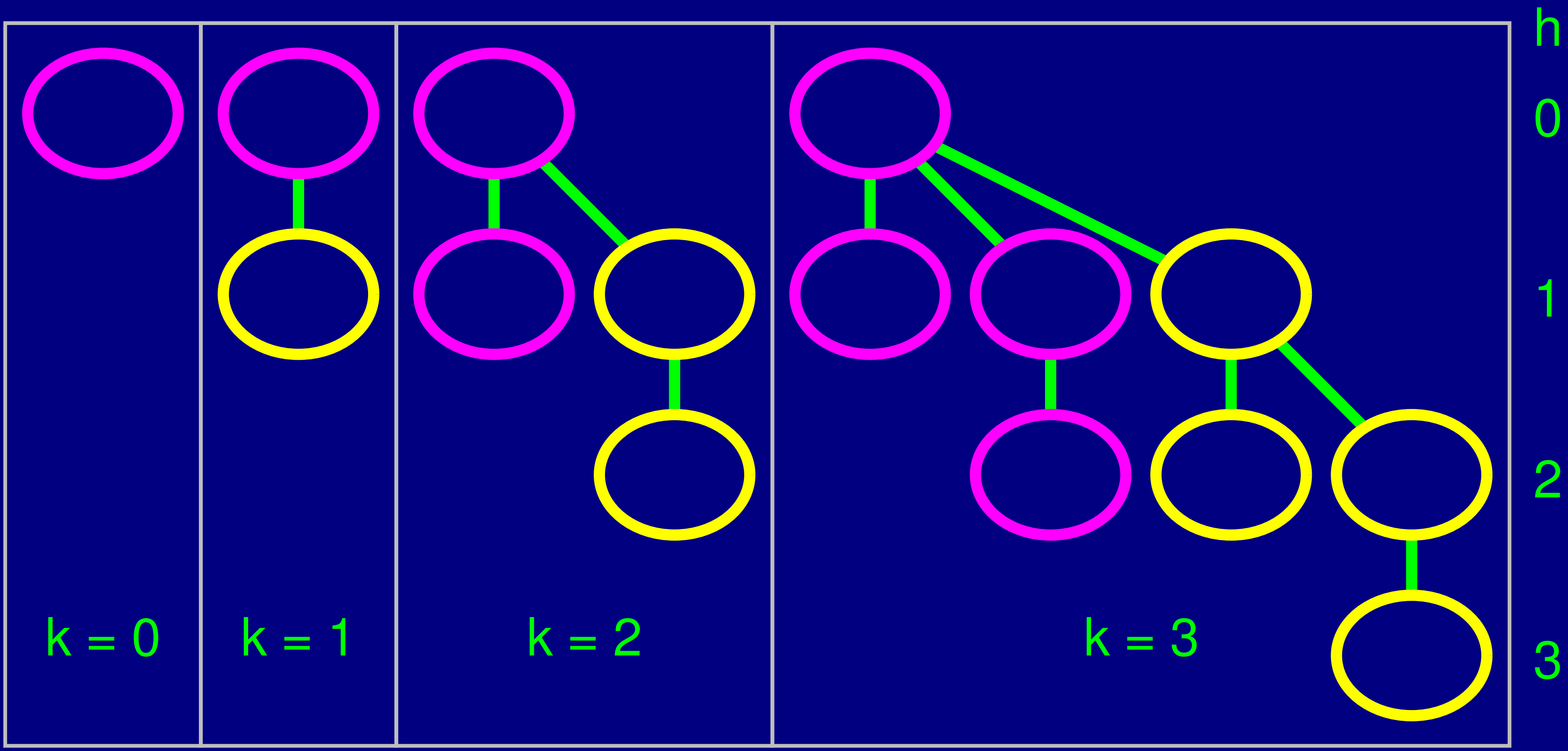
Subsection VI.2

Binomial-Tree Fault-Tolerant Routing

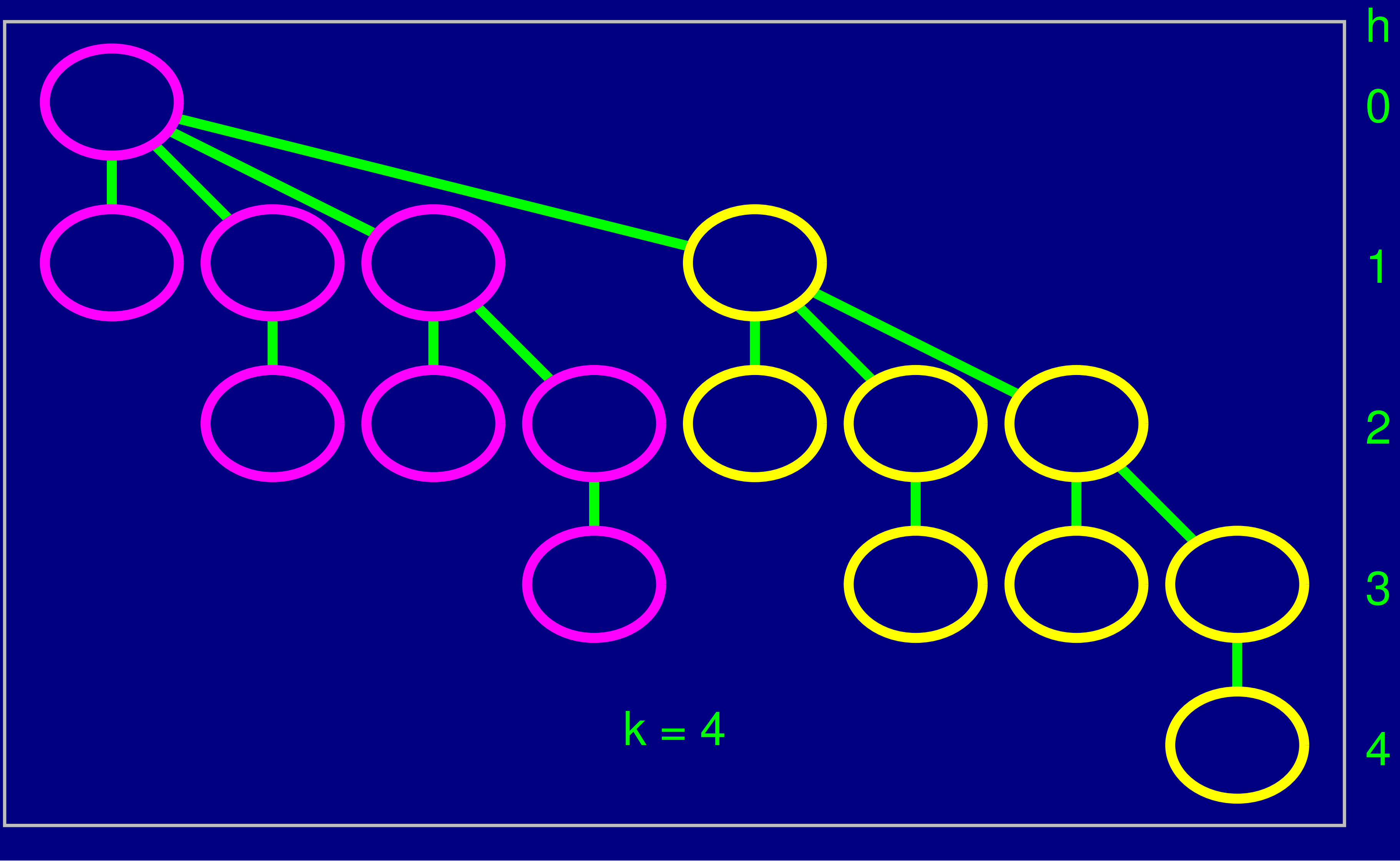
Binomial-Tree Routing

- Propose a fault tolerant routing algorithm
  - For dual-cubes
    - With very large number of faulty nodes
  - By using binomial-tree technique
    - Adaptive
- Performance evaluation of the algorithm
  - Complexity analysis
  - Software simulation
- An example
  - Dual-cube: up to 20 percent faulty nodes
  - Finding path: larger than 98 percent probability

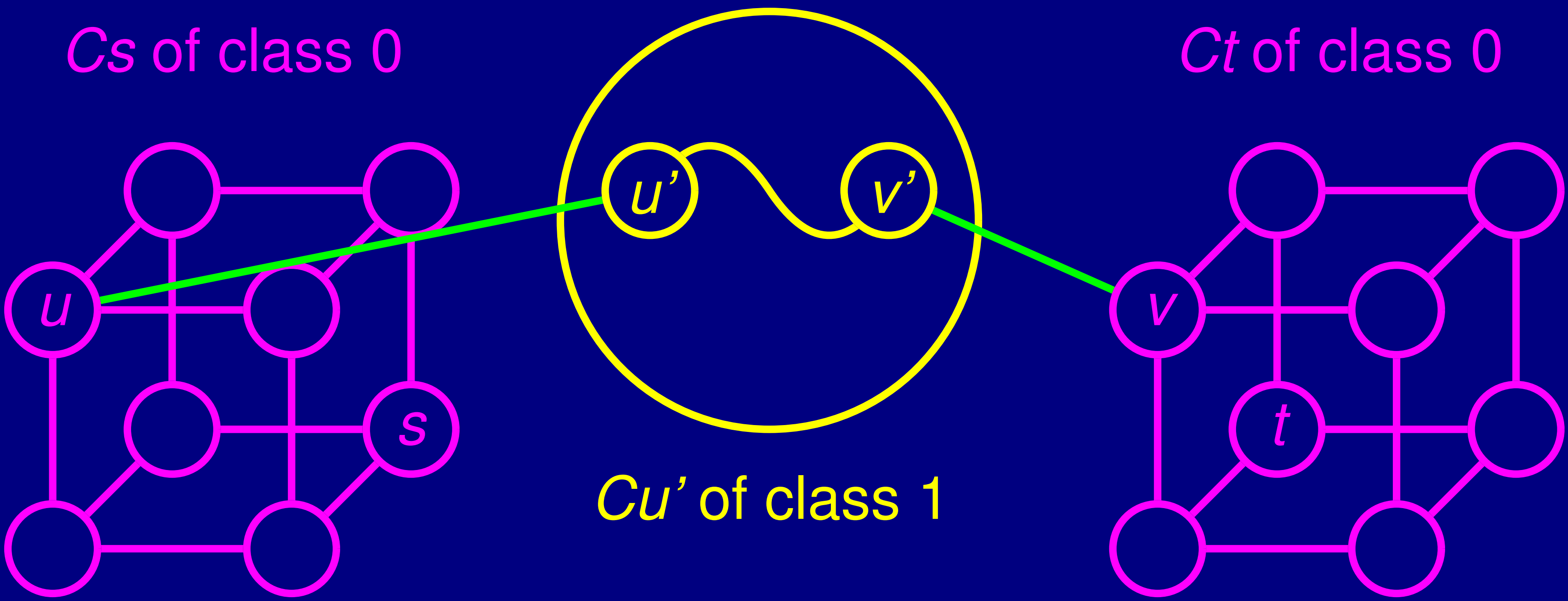
Binomial-Tree (k = 0, 1, 2, 3)



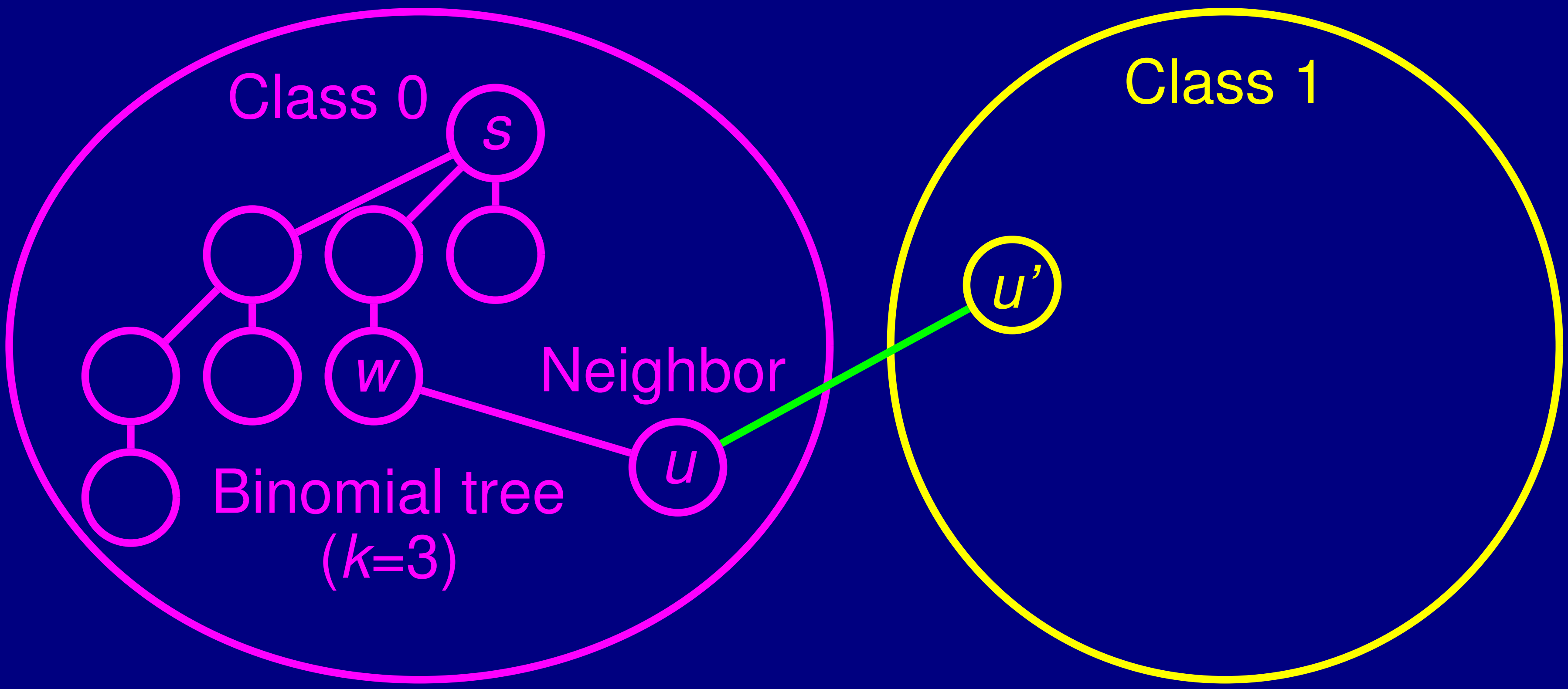
Binomial-Tree (k = 4)



A Virtual (m+1)-Cube

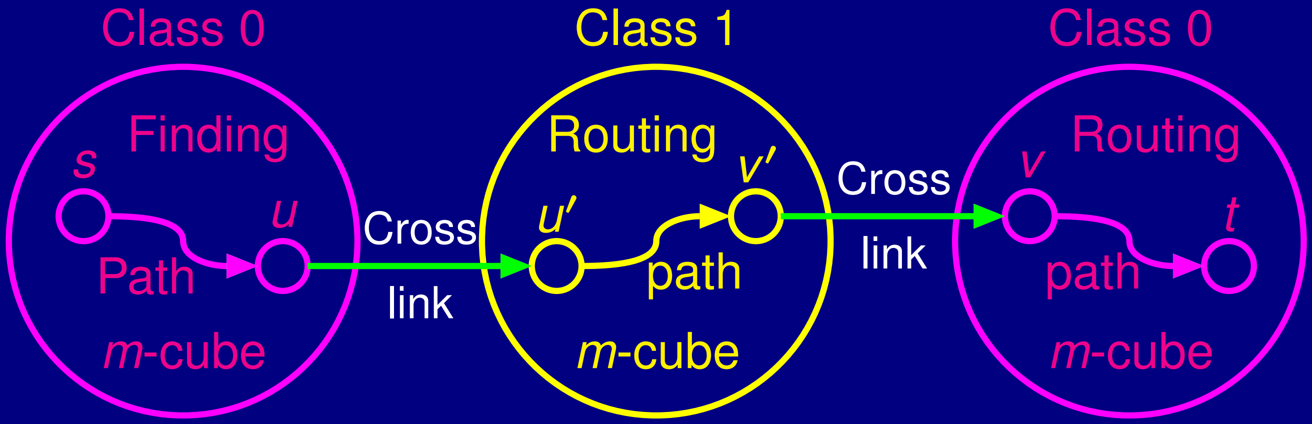


A Path Built with Binomial-Tree



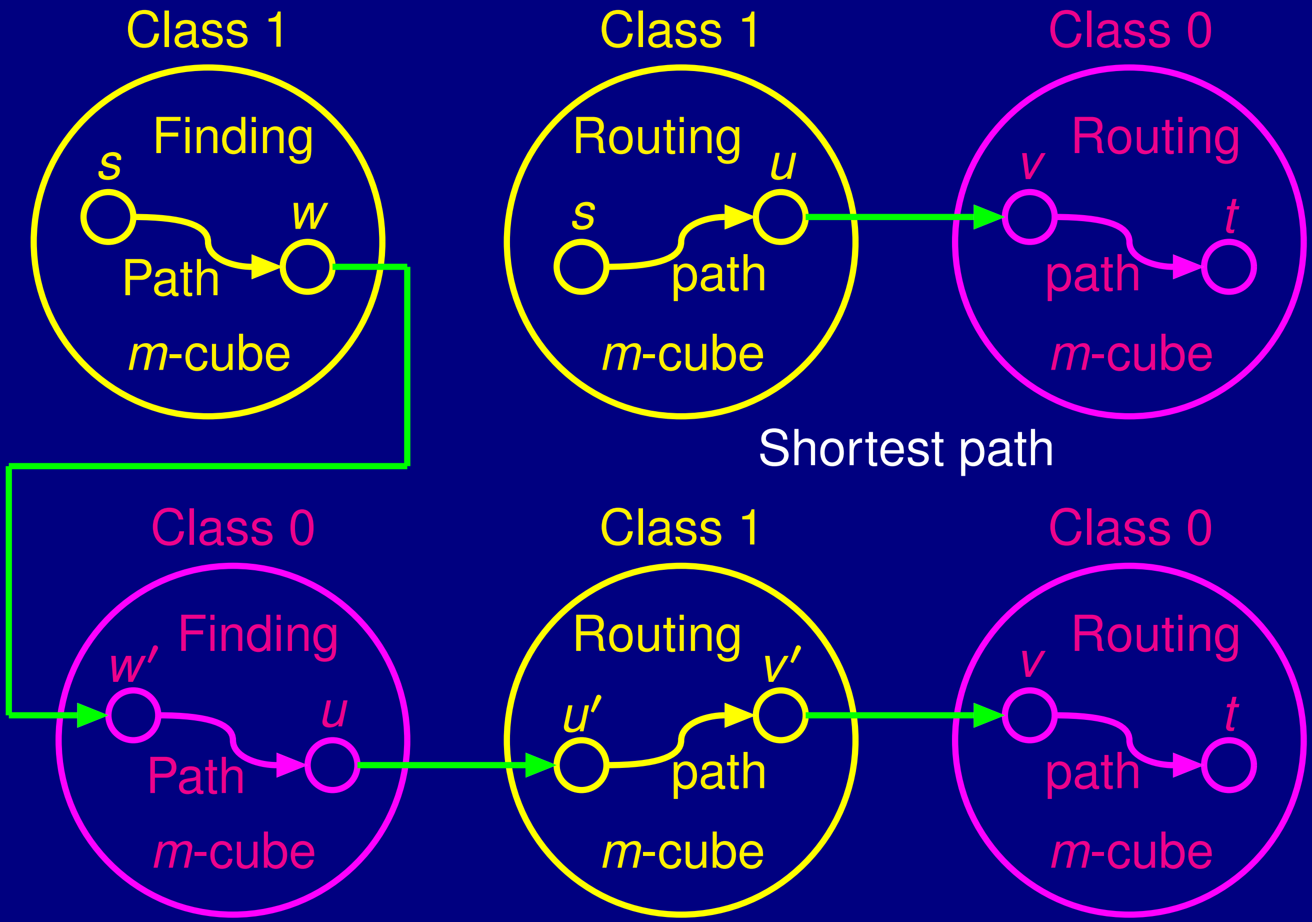
Fault Tolerant Routing in Dual-Cube

- Case 1: Two nodes are in a same cluster
  - This is the simplest case
  - Apply the hypercube routing algorithm directly
  - Our algorithm does not go outside
- Case 2: Two nodes are of same class

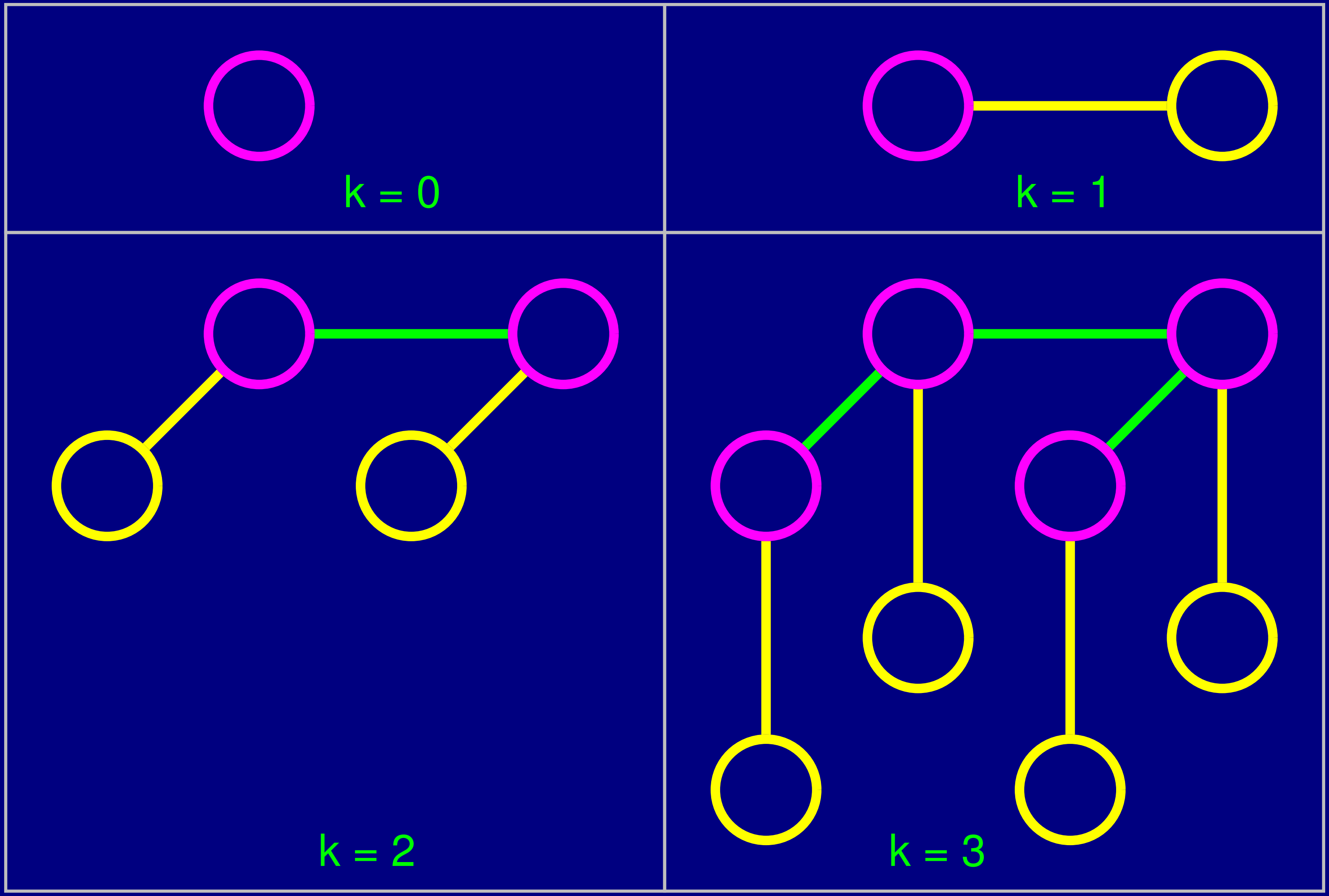


Fault Tolerant Routing in Dual-Cube

- Case 3: Two nodes are of different classes



Building Binomial-Tree



Binomial-Tree Algorithm

- During the building k binomial-tree
  - Check each new node u
    - If $u^{(j)}$ is nonfaulty
      - $w = u^{(j)}$, finish
- Search the k binomial-tree
  - Check each new node's neighbor u
    - If $u^{(j)}$ is nonfaulty
      - $w = u^{(j)}$, finish
- Adaptive
  - The dimensions at which w and t have different values are checked first

Simulations

- Uniform distributions of node failures
- For *m* = 5, 6, 7, 8, and 9 do
  - For *faulty* = 0.1 to 0.5 step by 0.1 do
    - For *i* = 1 to 10,000 step by 1 do
      - Simulation
- Outputs
  - Fault-free path
  - Probability of the successful routing
  - Average path-length / node-distance

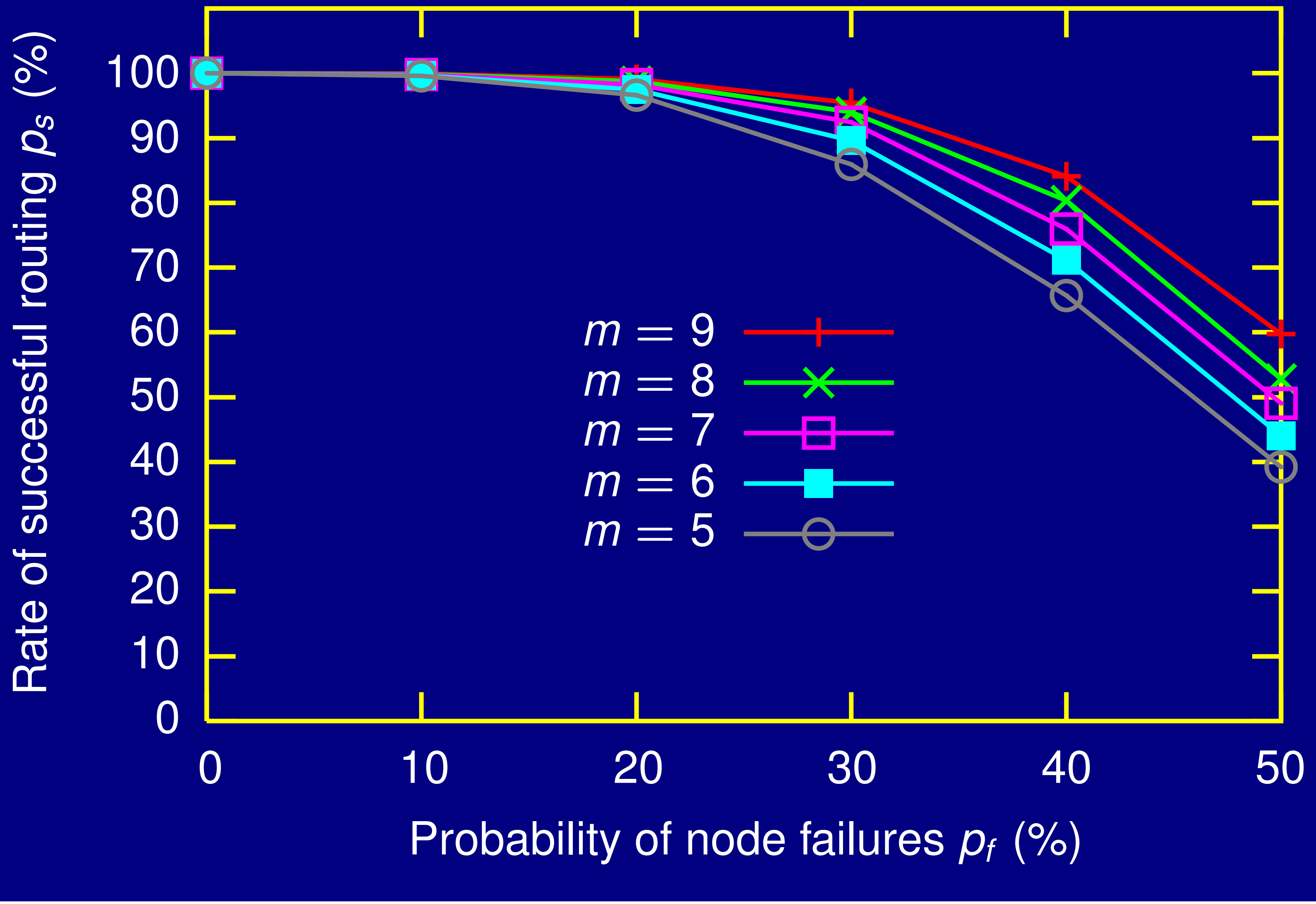
Performance Parameters

- $p_f(\%)$: the node failure probability
- $p_s(\%)$: the ratio of successful routings
- n_s: the number of successful routings
- n_f: the number of fault routings
- e_m: the maximum number of extra distance
- $e_p(\%)$: the average ratio of the length of the constructed routing path over the length of shortest path of the given two nodes

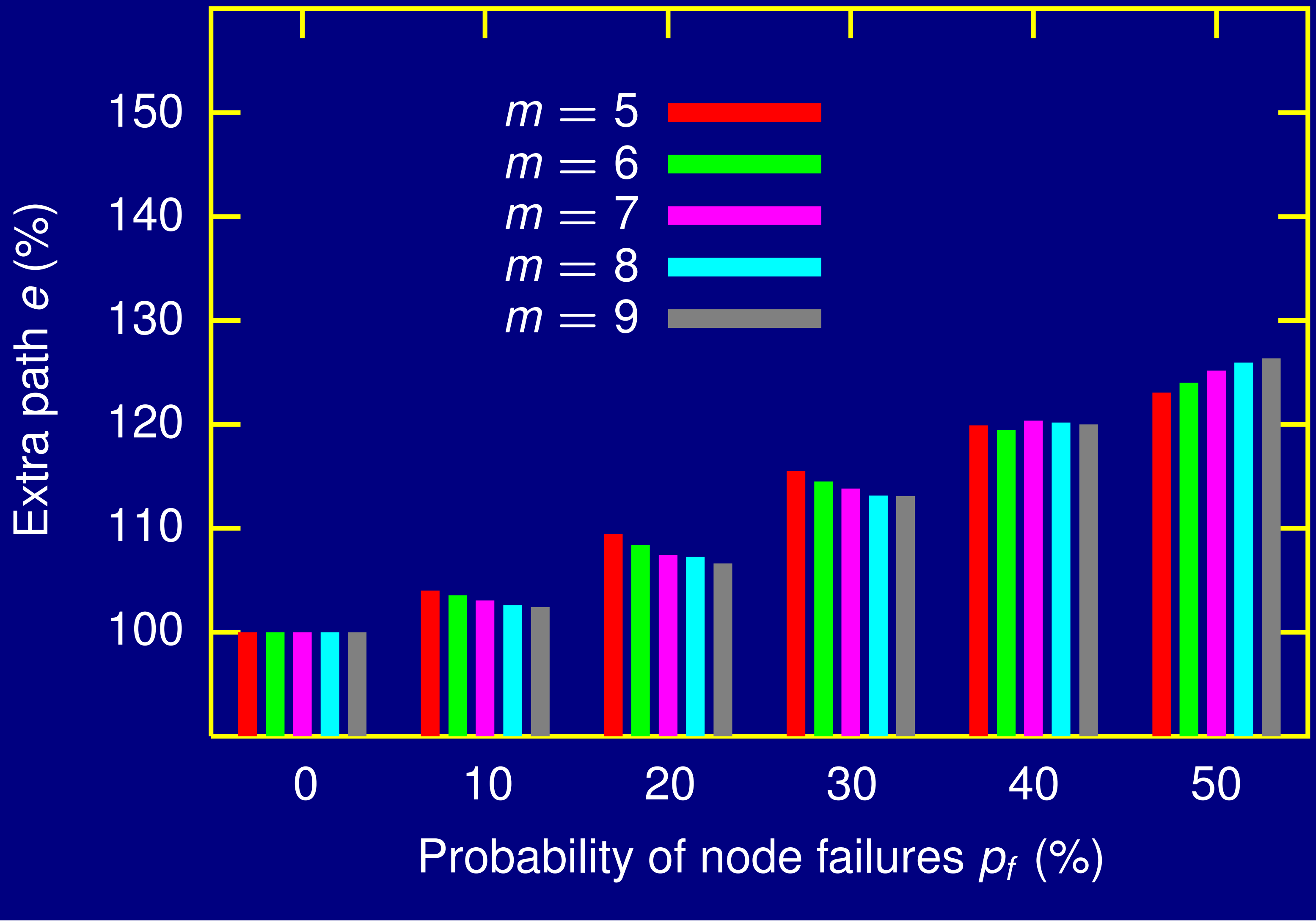
Routing in 7-Dual-Cubes

$p_f(\%)$	$p_s(\%)$	n_s	n_f	e_m	$e_p(\%)$
00	100.00	10000	0	0	100.00
10	99.84	9984	16	10	103.05
20	98.06	9806	194	12	107.39
30	91.40	9140	860	12	113.67
40	73.22	7322	2678	16	119.94
50	46.54	4654	5346	18	124.71

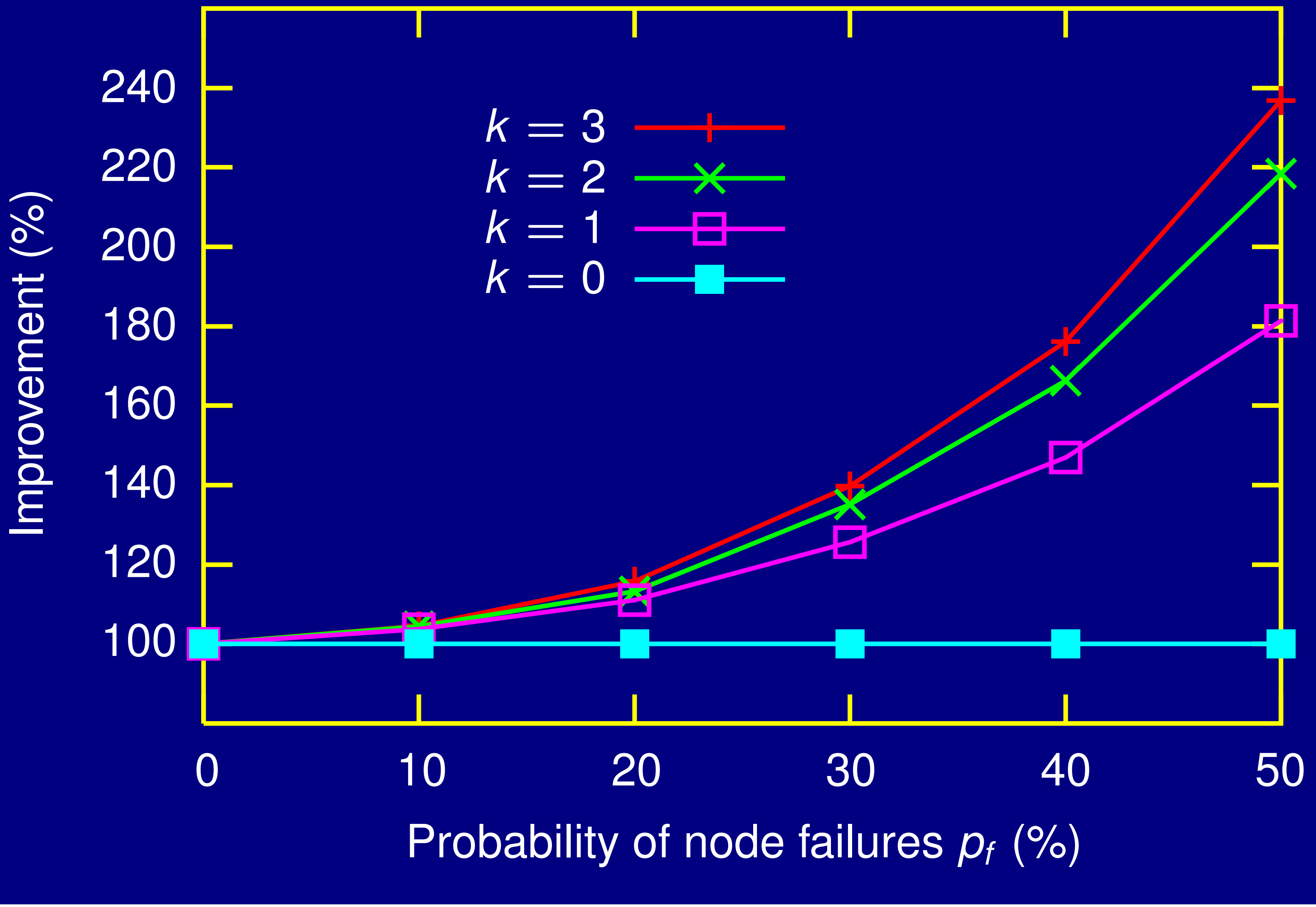
Successful Routing Rate



Ratio of the Path Length to d(s,t)



Effects of k (m = 7)



Summary

- We gave a fault-tolerant routing algorithm in dual-cube with a large amount of faulty nodes.
  - Based on binomial-tree
  - Requires only local information about the status of failures
  - Runs at nearly linear time.
  - Simulation results:
    - Dual-cube with 32,768 nodes
    - Contains up to 20 percent faulty nodes
    - Success rate: 98.07 percent

Conclusions

- Dual-cube: a new interconnection network
  - Low node degree (number of links per node)
  - Shorter diameter (distance between two nodes)
  - Symmetric (with recursive structure)
  - Easy to route (similar to hypercube)
  - Efficient communication operations
  - Linear array or ring embedding
  - Distributed fault-tolerant routing
- Can applyed to SGI Origin2000
  - Links mode nodes without Cray Router

References

1. Yamin Li and Shietung Peng, "Dual-Cubes: A New Interconnection Network for High-performance Computer Clusters", *Proceedings of the 2000 International Computer Symposium, Workshop on Computer Architecture,* December 6-8, 2000, National Chung Cheng University, ChiaYi, Taiwan. pp.51-57.
2. Yamin Li, Shietung Peng, and Wanming Chu, "Hamiltonian Cycle Embedding for Fault Tolerance in Dual-cube", *Proceedings of the IASTED International Conference on Networks, Parallel and Distributed Processing, and Applications (NPDPA 2002),* Tsukuba, Japan, October 2002, pp.1-6.
3. Yamin Li, Shietung Peng, and Wanming Chu, "Efficient Collective Communications in Dual-cube", *The Journal of Supercomputing,* Volume 4, issue 1, 2004, pp.71-90.
4. Yamin Li, Shietung Peng, and Wanming Chu, "Adaptive-Subcube Fault Tolerant Routing in Dual-Cube with Very Large Number of Faulty Nodes", *Proceedings of the ISCA 17th International Conference on Parallel and Distributed Computing Systems,* San Francisco, California USA, September, 2004, pp222-228.
5. Yamin Li, Shietung Peng, and Wanming Chu, "An Efficient Algorithm for Fault Tolerant Routing Based on Adaptive Binomial-Tree Technique in Hypercubes", *Proceedings of the Fifth International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT'04),* Dec. 8-10, 2004, Singapore. pp.196-201.
6. Yamin Li, Shietung Peng, and Wanming Chu, "Fault-Tolerant Cycle Embedding in Dual-Cube with Node Faulty", *International Journal of High Performance Computing and Networking* Vol. 3, No. 1, 2005, pp.45-53.

THX