(How to Implement) Basic Communication Operations

Alexandre David
1.2.05
adavid@cs.aau.dk

Overview

- One-to-all broadcast & all-to-one reduction
- All-to-all broadcast and reduction
- All-reduce and prefix-sum operations
- Scatter and Gather
- All-to-All Personalized Communication
- Circular Shift
- Improving the Speed of Some Communication Operations

Collective Communication Operations

- Represent regular communication patterns.
- Used extensively in most data-parallel algorithms.
- Critical for efficiency.
- Available in most parallel libraries.
- Very useful to “get started” in parallel processing.
- Basic model: t_s+mt_w time for exchanging a *m*-word message with cut-through routing.

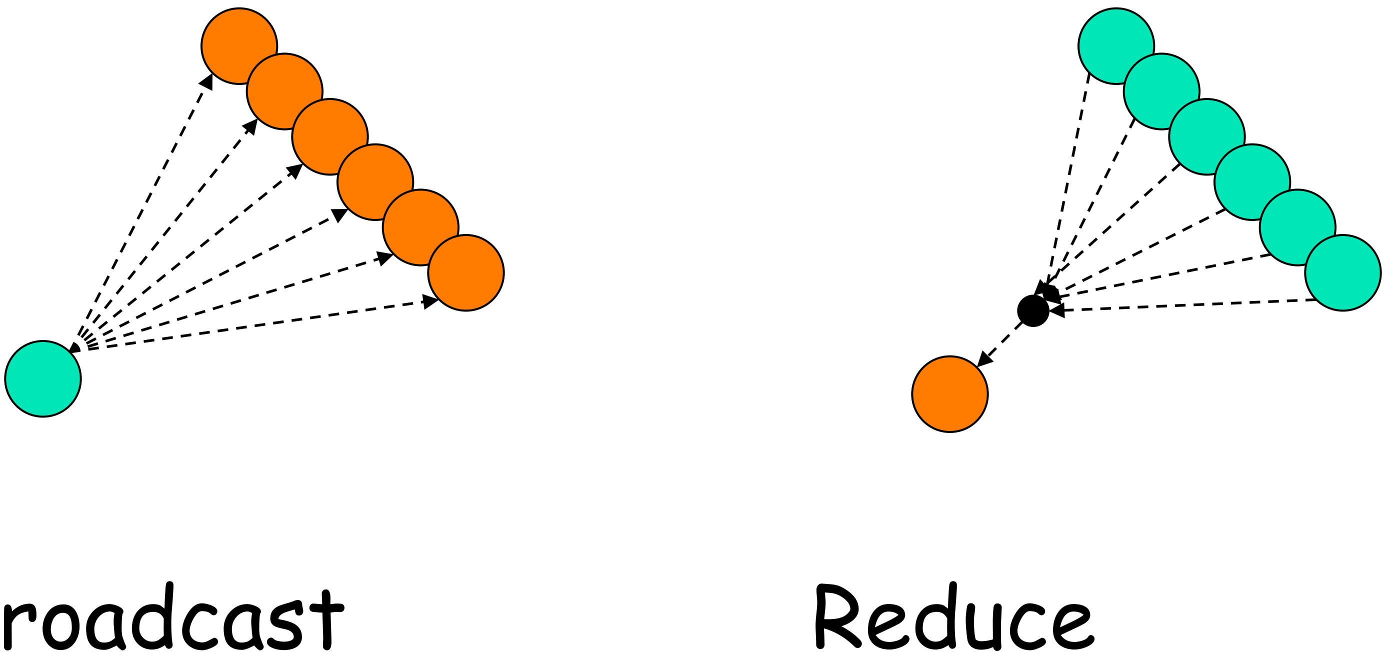
Interesting:

- To know:
  - Data transfer time is roughly the same between *all pairs* of nodes.
  - Homogeneity true on modern hardware (randomized routing, cut-through routing…).
    - t_s+mt_w
    - Adjust t_w for congestion: effective t_w.
- Model: bidirectional links, single port.
- Communication with point-to-point primitives.

Broadcast/Reduction

- One-to-all broadcast:
  - Single process sends identical data to all (or subset of) processes.
- All-to-one reduction:
  - Dual operation.
  - *P* processes have *m* words to send to one destination.
  - Parts of the message need to be *combined*.

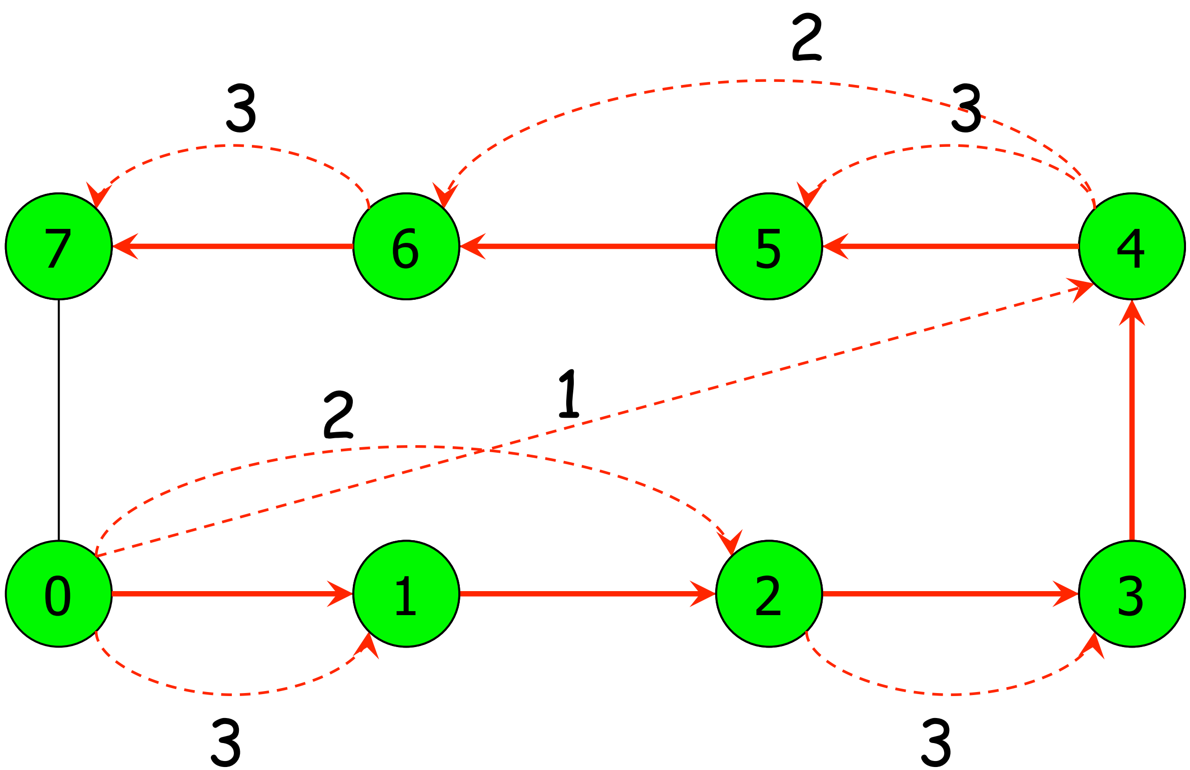
Broadcast/Reduction



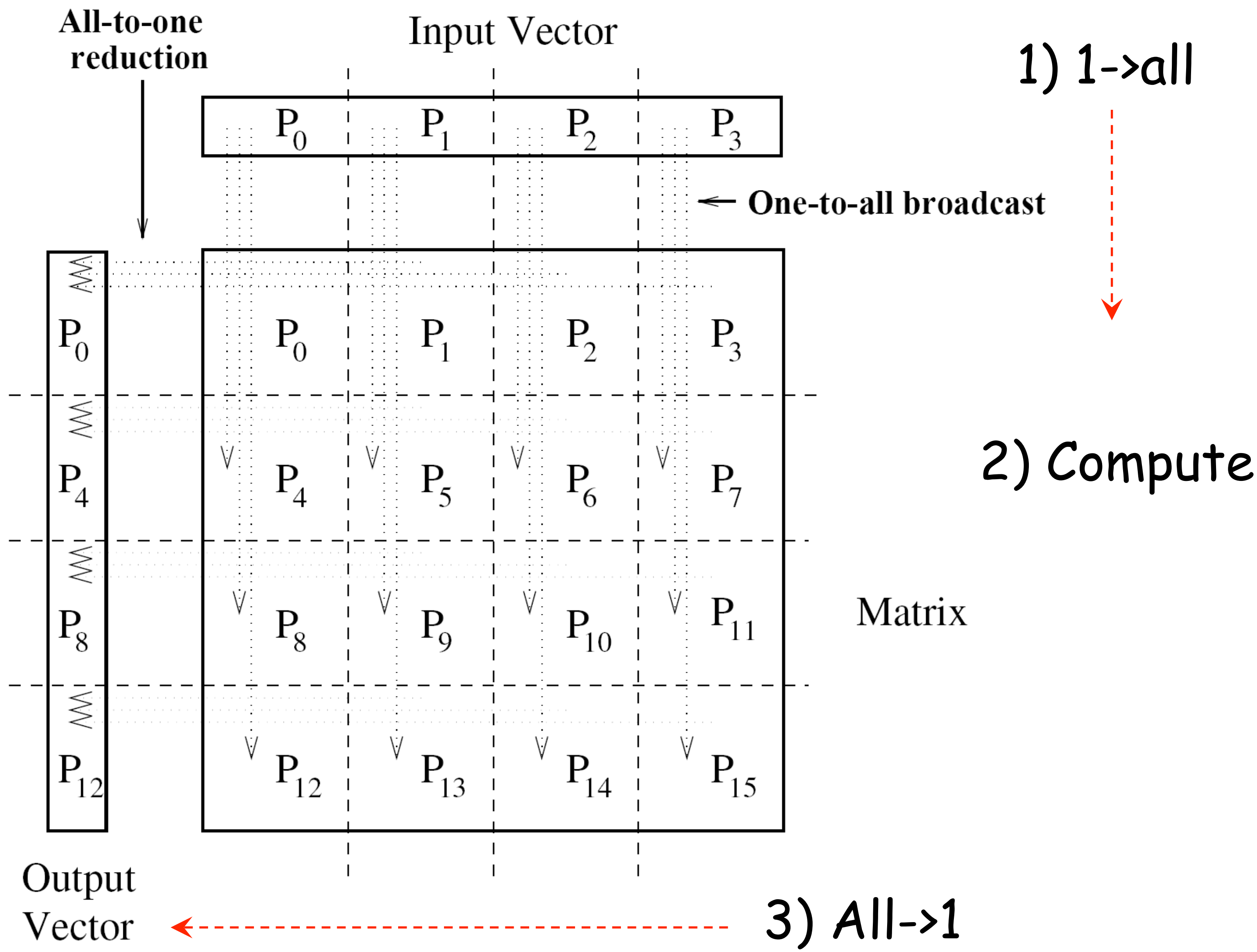
One-to-All Broadcast – Ring/Linear Array

- Naïve approach: send sequentially.
  - Bottleneck.
  - Poor utilization of the network.
- Recursive doubling:
  - Broadcast in $\log p$ steps (instead of p).
  - Divide-and-conquer type of algorithm.
  - Reduction is similar.

Recursive Doubling



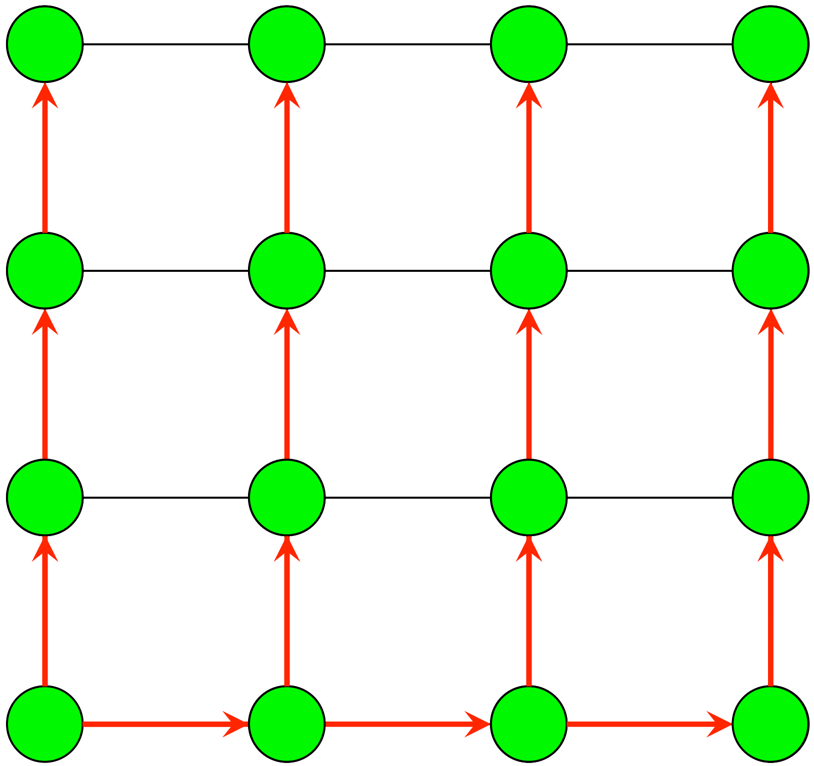
Example: Matrix*Vector



One-to-All Broadcast – Mesh

- Extensions of the linear array algorithm.
  - Rows & columns = arrays.
  - Broadcast on a row, broadcast on columns.
  - Similar for reductions.
  - Generalize for higher dimensions (cubes…).

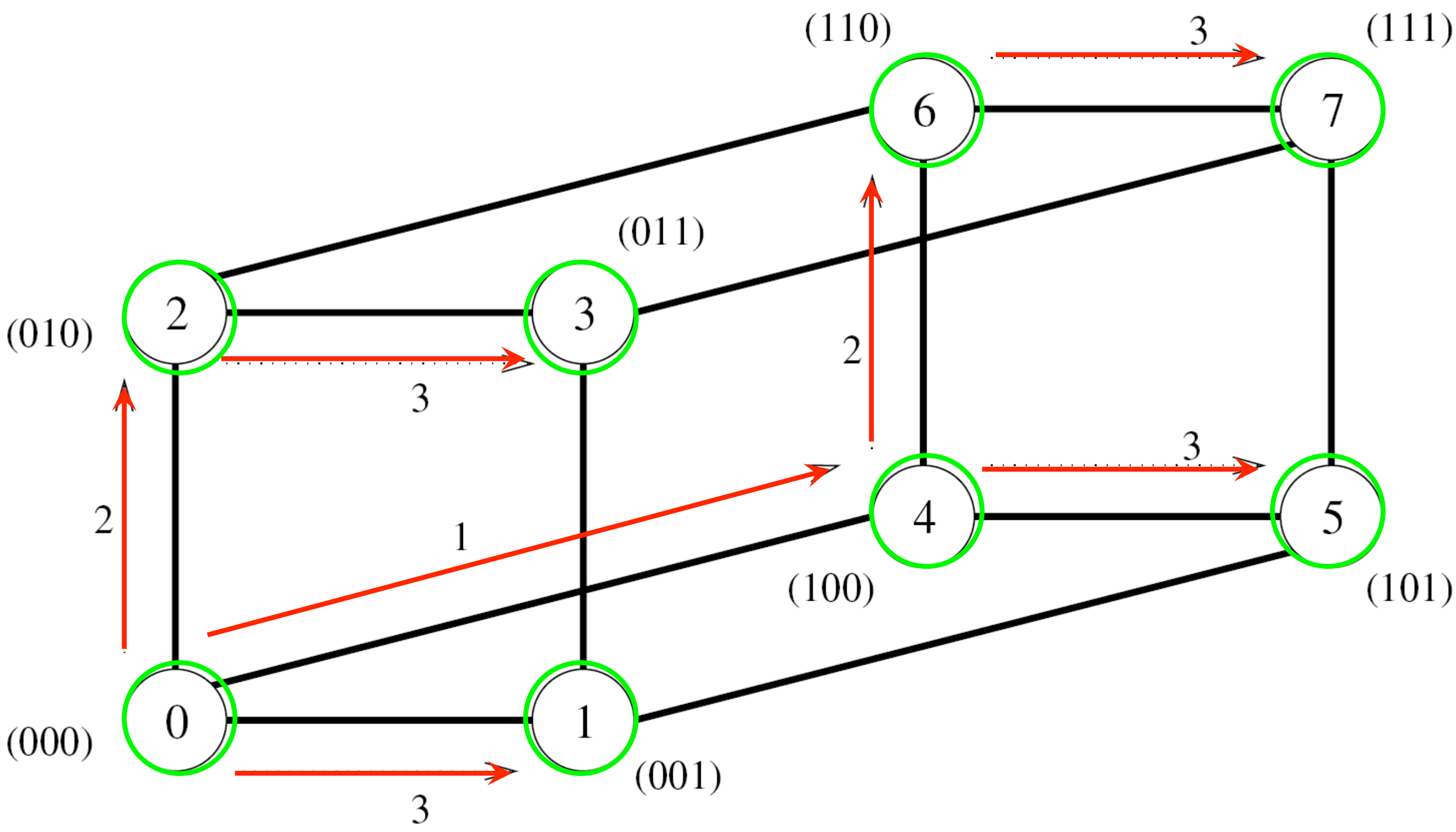
Broadcast on a Mesh



One-to-All Broadcast – Hypercube

- Hypercube with 2^d nodes = d-dimensional mesh with 2 nodes in each direction.
- Similar algorithm in d steps.
- Also in $\log p$ steps.
- Reduction follows the same pattern.

Broadcast on a Hypercube



All-to-One Broadcast
Balanced Binary Tree

- Processing nodes = leaves.
- Hypercube algorithm maps well.
- Similarly good w.r.t. congestion.

Broadcast on a Balanced Binary Tree

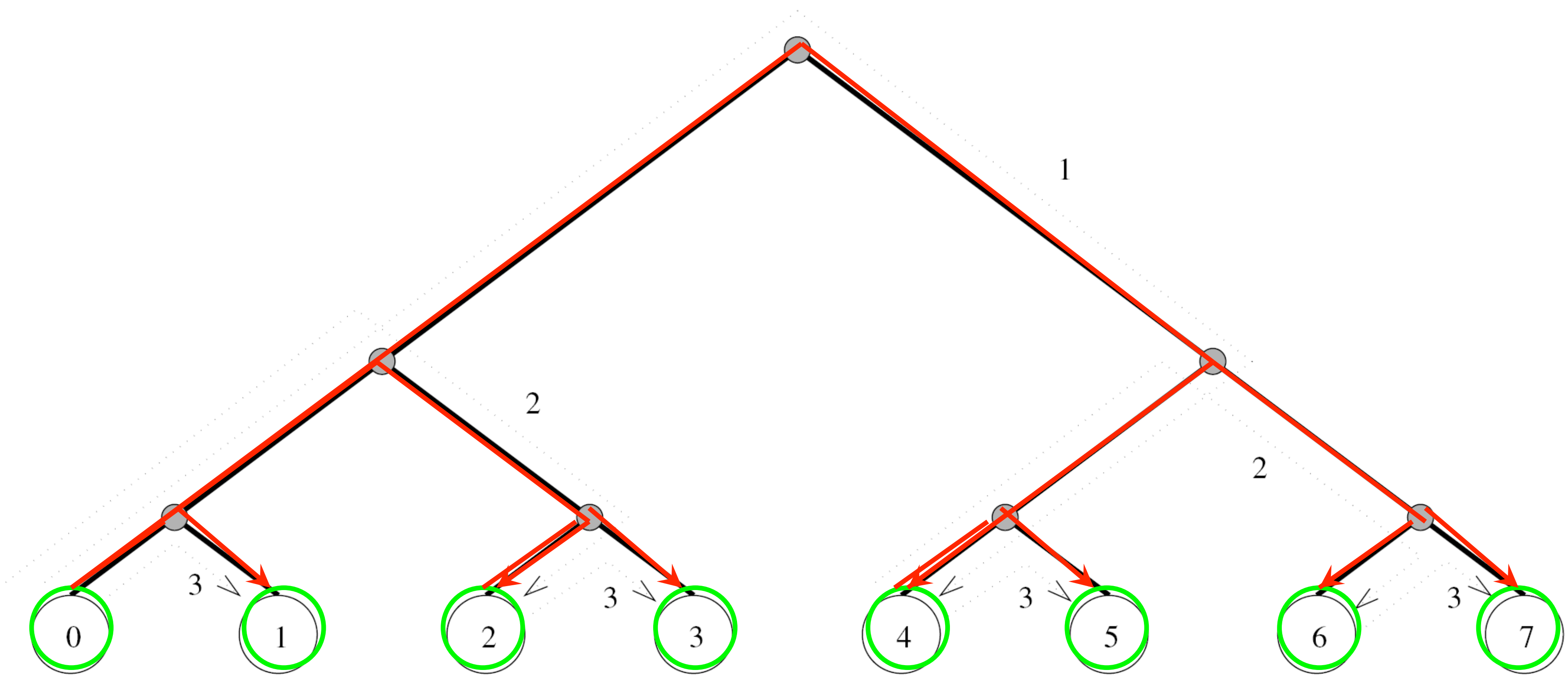


Figure 4.7 One-to-all broadcast on an eight-node tree.

Algorithms

- So far we saw pictures.
- Not enough to implement.
- Precise description
  - to implement.
  - to analyze.
- Description for hypercube.
- Execute the following procedure on all the nodes.

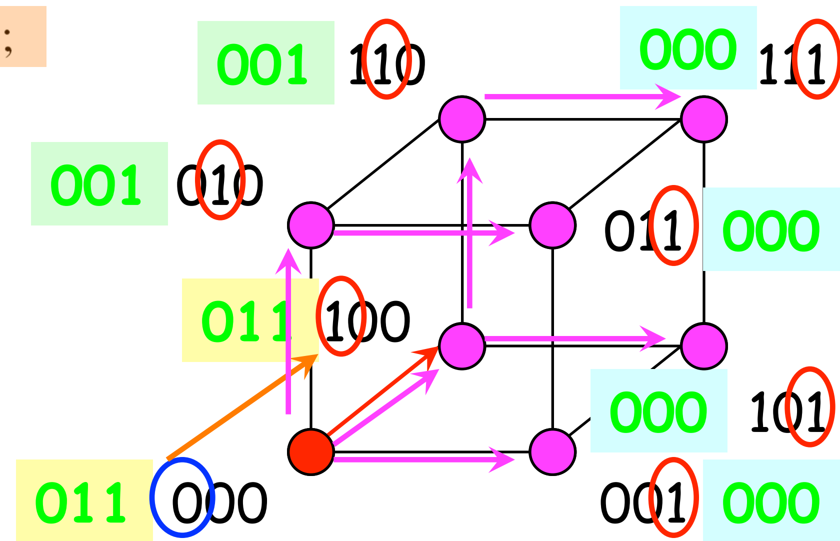
Broadcast Algorithm

```
procedure ONE_TO_ALL_BC(d, my_id, X)
begin
    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_BC
```

Current dimension

111

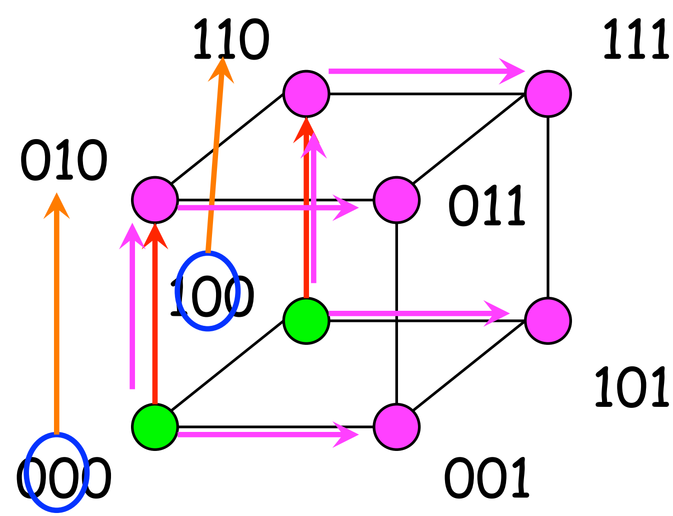
011 001 000



Broadcast Algorithm

```
procedure ONE_TO_ALL_BC(d, my_id, X)
begin
    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_BC
```

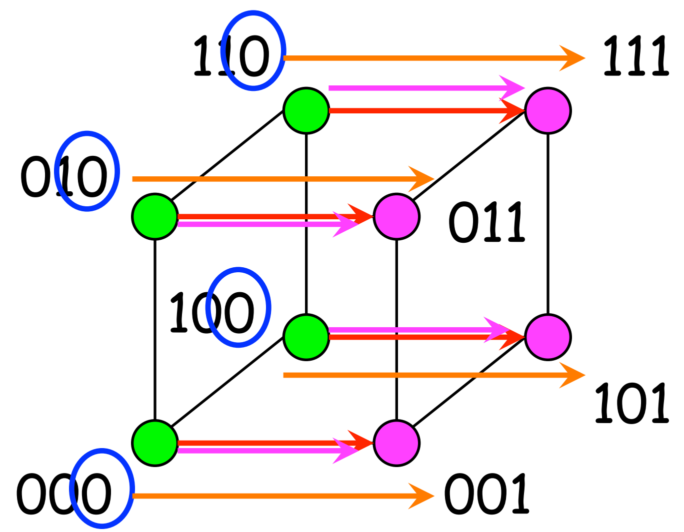
001 000



Broadcast Algorithm

```
procedure ONE_TO_ALL_BC(d, my_id, X)
begin
    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_BC
```

000



Algorithm For Any Source

```
procedure GENERAL_ONE_TO_ALL_BC(d, my_id, source, X)
begin
    my_virtual_id := my_id XOR source;
    mask := 2^d − 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 (my_virtual_id AND 2^i) = 0 then
                virtual_dest := my_virtual_id XOR 2^i;
                send X to (virtual_dest XOR source);
    /* Convert virtual_dest to the label of the physical destination */
            else
                virtual_source := my_virtual_id XOR 2^i;
                receive X from (virtual_source XOR source);
    /* Convert virtual_source to the label of the physical source */
            endelse;
    endfor;
end GENERAL_ONE_TO_ALL_BC
```

Reduce Algorithm

```
procedure ALL_TO_ONE_REDUCE(d, my_id, m, X, sum)
begin
    for j := 0 to m − 1 do sum[j] := X[j];
    mask := 0;
    for i := 0 to d − 1 do
        /* Select nodes whose lower i bits are 0 */
        if (my_id AND mask) = 0 then
            if (my_id AND 2^i) ≠ 0 then
                msg_destination := my_id XOR 2^i;
                receive X from msg_source;
                for j := 0 to m − 1 do
                    sum[j] := sum[j] + X[j];
            endelse;
        mask := mask XOR 2^i;    /* Set bit i of mask to 1 */
    endfor;
end ALL_TO_ONE_REDUCE
```

In a nutshell:
reverse the previous one.

Cost Analysis

p processes → log$p$ steps (point-to-point transfers in parallel).
Each transfer has a time cost of $t_s+t_w m$.
Total time: $T=(t_s+t_w m)\log p$.

All-to-All Broadcast and Reduction

- Generalization of broadcast:
  - Each processor is a source and destination.
  - Several processes broadcast different messages.
- Used in matrix multiplication (and matrix-vector multiplication).
- Dual: all-to-all reduction.

All-to-All Broadcast and Reduction

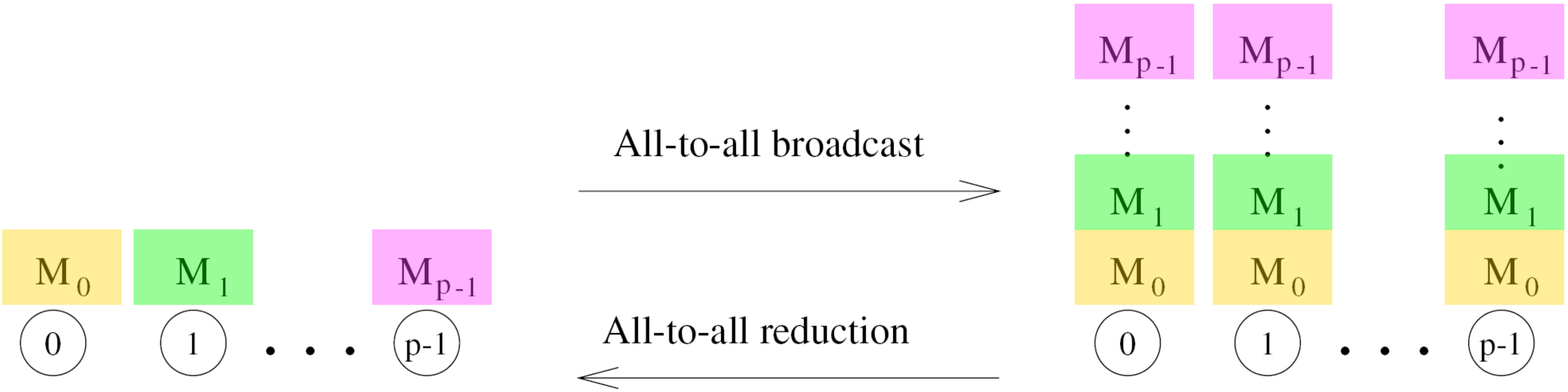
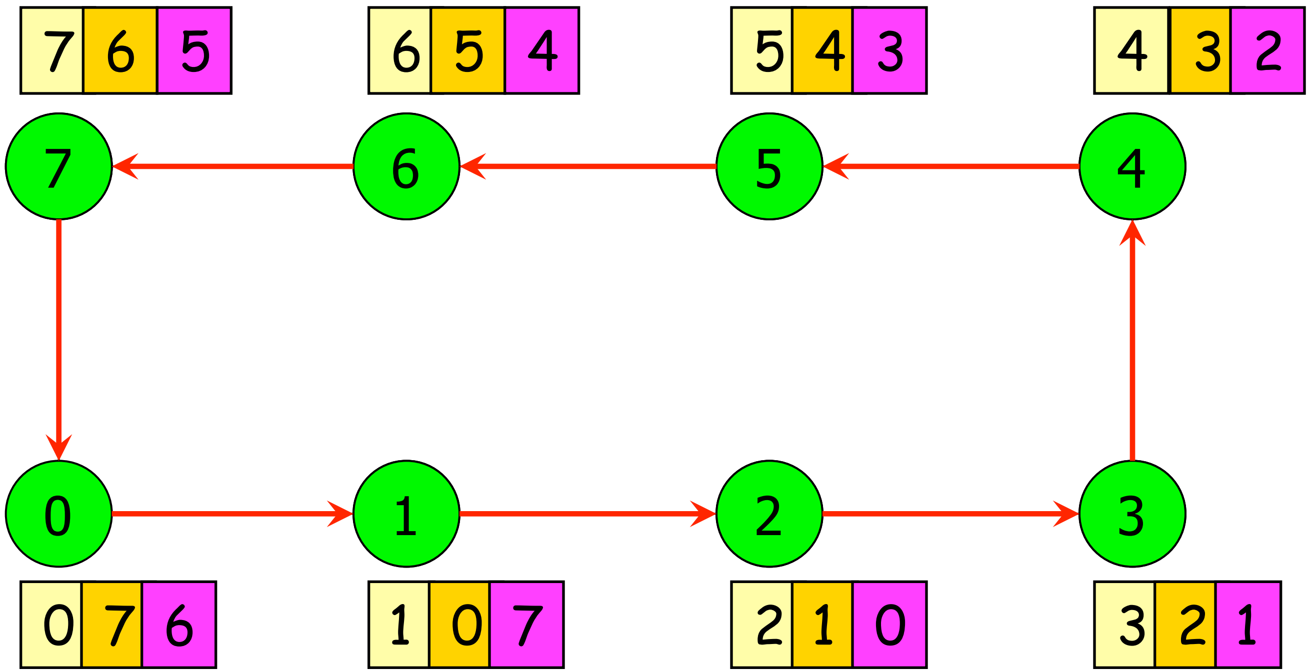


Figure 4.8 All-to-all broadcast and all-to-all reduction.

All-to-All Broadcast – Rings



etc…

All-to-All Broadcast Algorithm

```
procedure ALL_TO_ALL_BC_RING(my_id, my_msg, p, result)
begin
    left := (my_id − 1) mod p;
    right := (my_id + 1) mod p;
    result := my_msg;
    msg := result;
    for i := 1 to p − 1 do
        send msg to right;
        receive msg from left;
        result := result ∪ msg;
    endfor;
end ALL_TO_ALL_BC_RING
```

Ring: mod *p*.

Receive & send – point-to-point.

Initialize the loop.

Forward msg.

Accumulate result.

Algorithm 4.4 All-to-all broadcast on a p-node ring.

All-to-All Reduce Algorithm

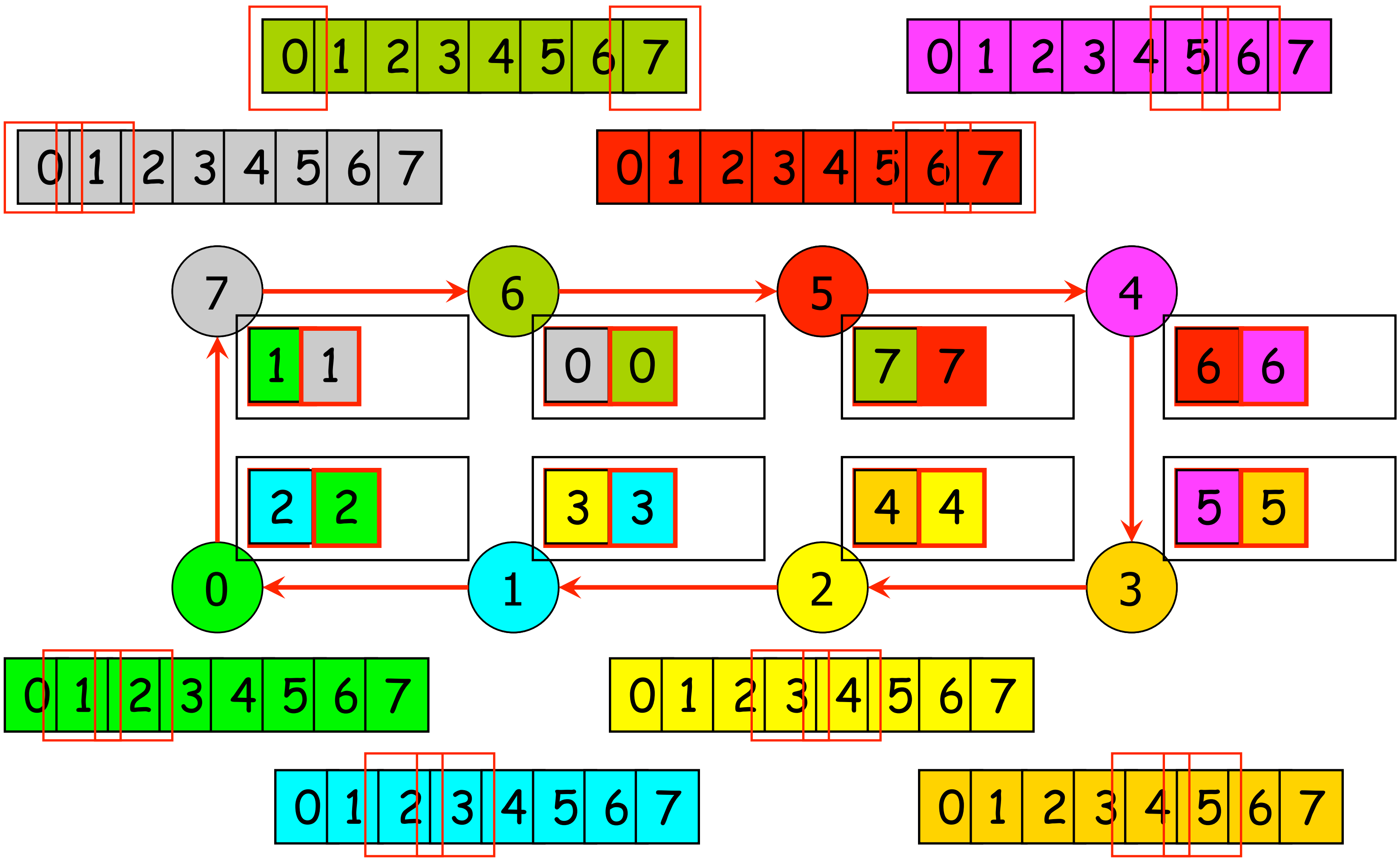
```
procedure ALL_TO_ALL_RED_RING(my_id, my_msg, p, result)
begin
    left := (my_id − 1) mod p;
    right := (my_id + 1) mod p;
    recv := 0;
    for i := 1 to p − 1 do
        j := (my_id + i) mod p;
        temp := msg[j] + recv;
        send temp to left;
        receive recv from right;
    endfor;
    result := msg[my_id] + recv;
end ALL_TO_ALL_RED_RING
```

Accumulate and forward.

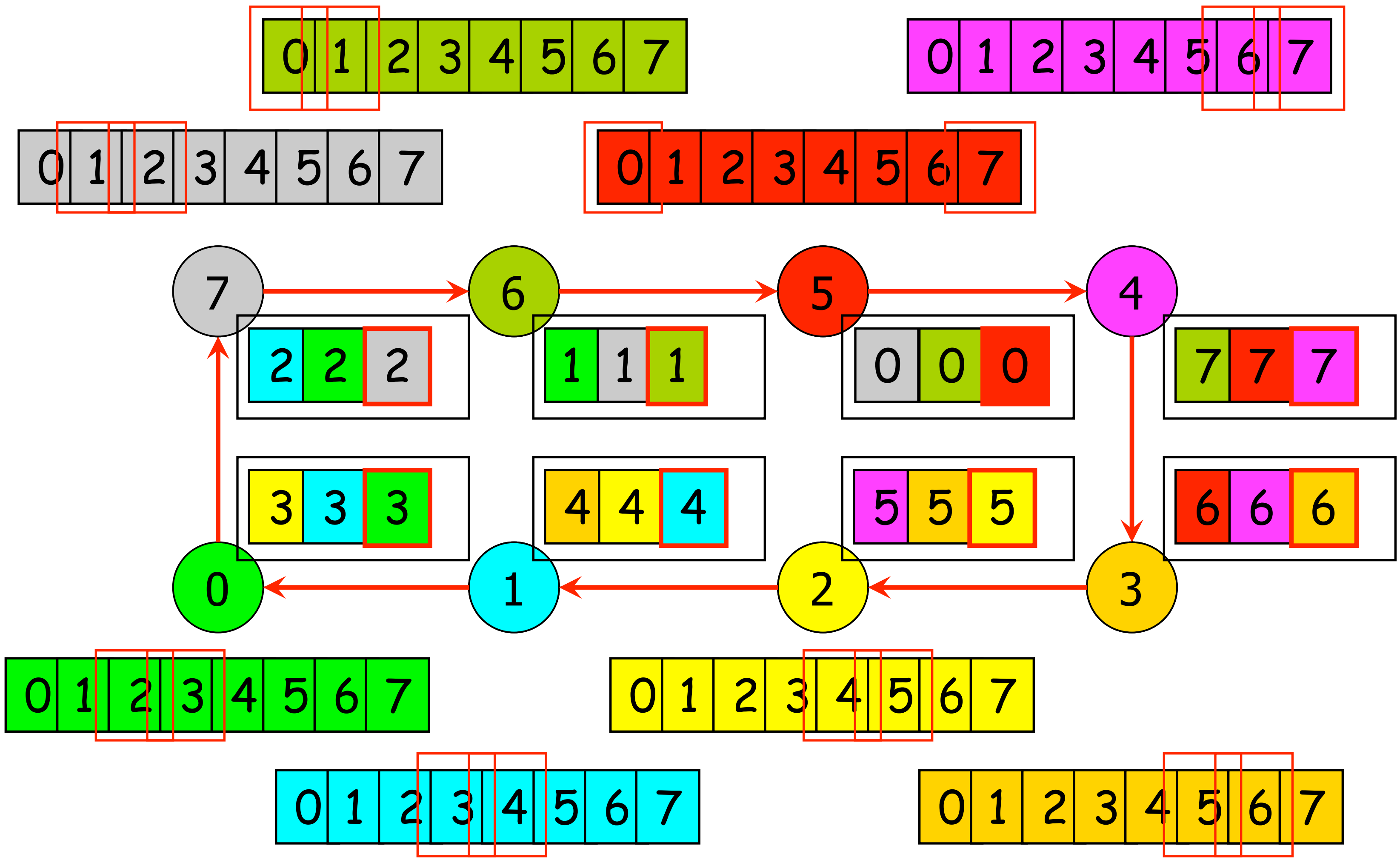
Last message for *my_id*.

Algorithm 4.5 All-to-all reduction on a p-node ring.

All-to-All Reduce – Rings



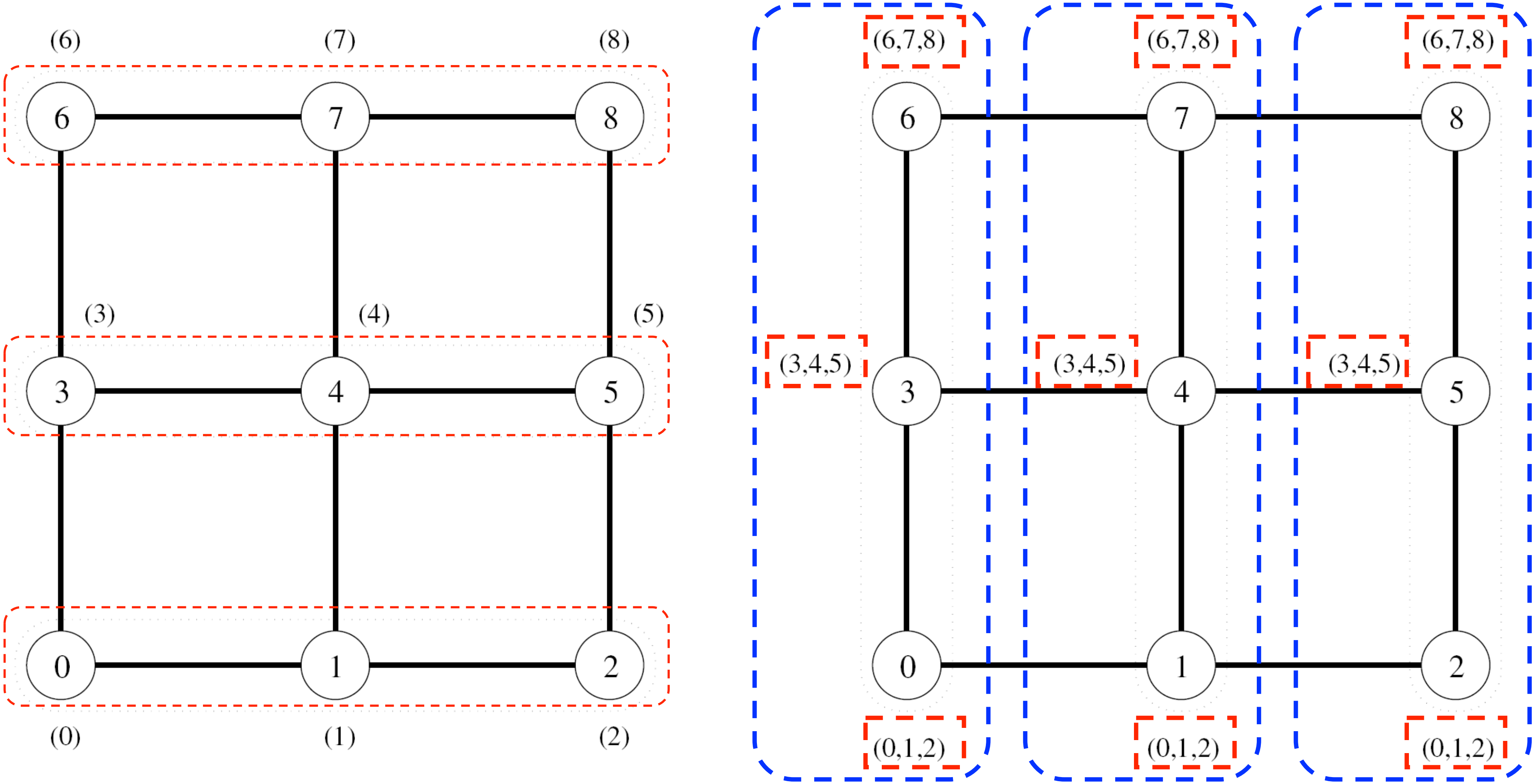
All-to-All Reduce – Rings



All-to-All Broadcast – Meshes

- Two phases:
  - All-to-all on rows – messages size m.
    - Collect sqrt(p) messages.
  - All-to-all on columns – messages size sqrt(p) *m.

All-to-All Broadcast – Meshes



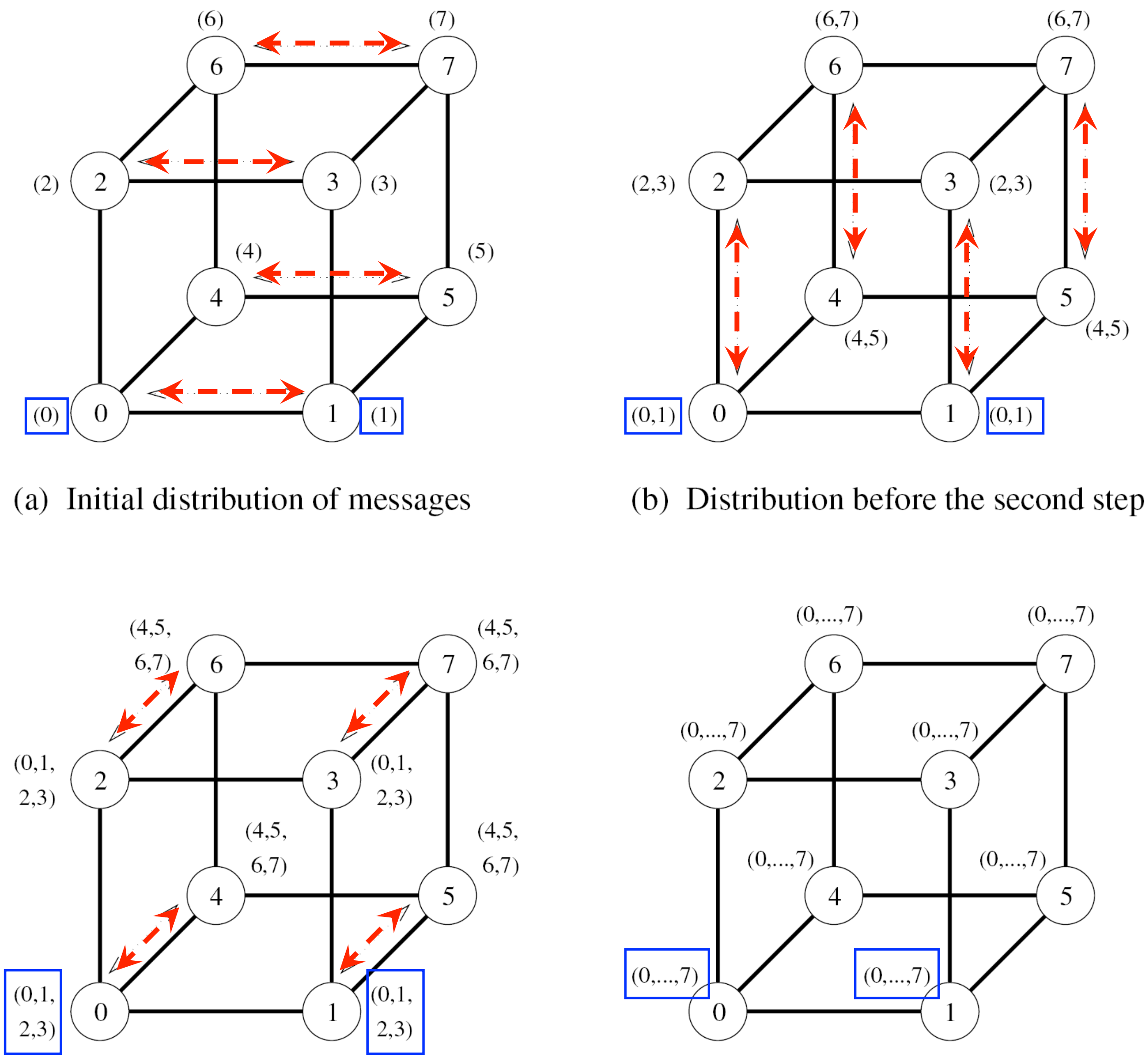
Algorithm

```
procedure ALL_TO_ALL_BC_MESH(my_id, my_msg, p, result)
begin

/* Communication along rows */
    left := my_id − (my_id mod √p) + (my_id − 1)mod√p;
    right := my_id − (my_id mod √p) + (my_id + 1) mod √p;
    result := my_msg;
    msg := result;
    for i := 1 to √p − 1 do
        send msg to right;
        receive msg from left;
        result := result ∪ msg;
    endfor;

/* Communication along columns */
    up := (my_id − √p) mod p;
    down := (my_id + √p) mod p;
    msg := result;
    for i := 1 to √p − 1 do
        send msg to down;
        receive msg from up;
        result := result ∪ msg;
    endfor;
end ALL_TO_ALL_BC_MESH
```

All-to-All Broadcast - Hypercubes

- Generalization of the mesh algorithm to log*p* dimensions.
- Message size doubles at every step.
- Number of steps: log*p*.

All-to-All Broadcast – Hypercubes



(a) Initial distribution of messages

(b) Distribution before the second step

(c) Distribution before the third step

(d) Final distribution of messages

Algorithm

```
procedure ALL_TO_ALL_BC_HCUBE(my_id, my_msg, d, 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 ∪ msg;
    endfor;
end ALL_TO_ALL_BC_HCUBE
```

Loop on the dimensions

Exchange messages

Forward (double size)

Algorithm 4.7 All-to-all broadcast on a d-dimensional hypercube.

All-to-All Reduction – Hypercubes

```
procedure ALL_TO_ALL_RED_HCUBE(my_id, msg, d, result)
begin
    recloc := 0;
    for i := d - 1 to 0 do
        partner := my_id XOR 2^i;
        j := my_id AND 2^i;
        k := (my_id XOR 2^i) AND 2^i;
        senloc := recloc + k;
        recloc := recloc + j;
        send msg[senloc .. senloc + 2^i - 1] to partner;
        receive temp[0 .. 2^i - 1] from partner;
        for j := 0 to 2^i - 1 do
            msg[recloc + j] := msg[recloc + j] + temp[j];
        endfor;
    endfor;
    result := msg[my_id];
end ALL_TO_ALL_RED_HCUBE
```

Similar pattern in reverse order.

Combine results

Algorithm 4.8 All-to-all broadcast on a d-dimensional hypercube. AND and XOR are bitwise logical-and and exclusive-or operations, respectively.

Cost Analysis (Time)

- Ring:
  - $T=(t_s + t_w m)(p-1)$.
- Mesh:
  - $T=(t_s + t_w m)(\sqrt{p}-1)+(t_s + t_w m\sqrt{p})(\sqrt{p}-1)$
    $= 2ts(\sqrt{p} - 1) + t_w m(p-1)$.
- Hypercube:
  - $$T = \sum_{i=1}^{\log p}(t_s + 2^{i-1}t_w m)$$
    $$= t_s \log p + t_w m(p-1).$$

logp steps
message of size $2^{i-1}m$.

Dense to Sparser: Congestion

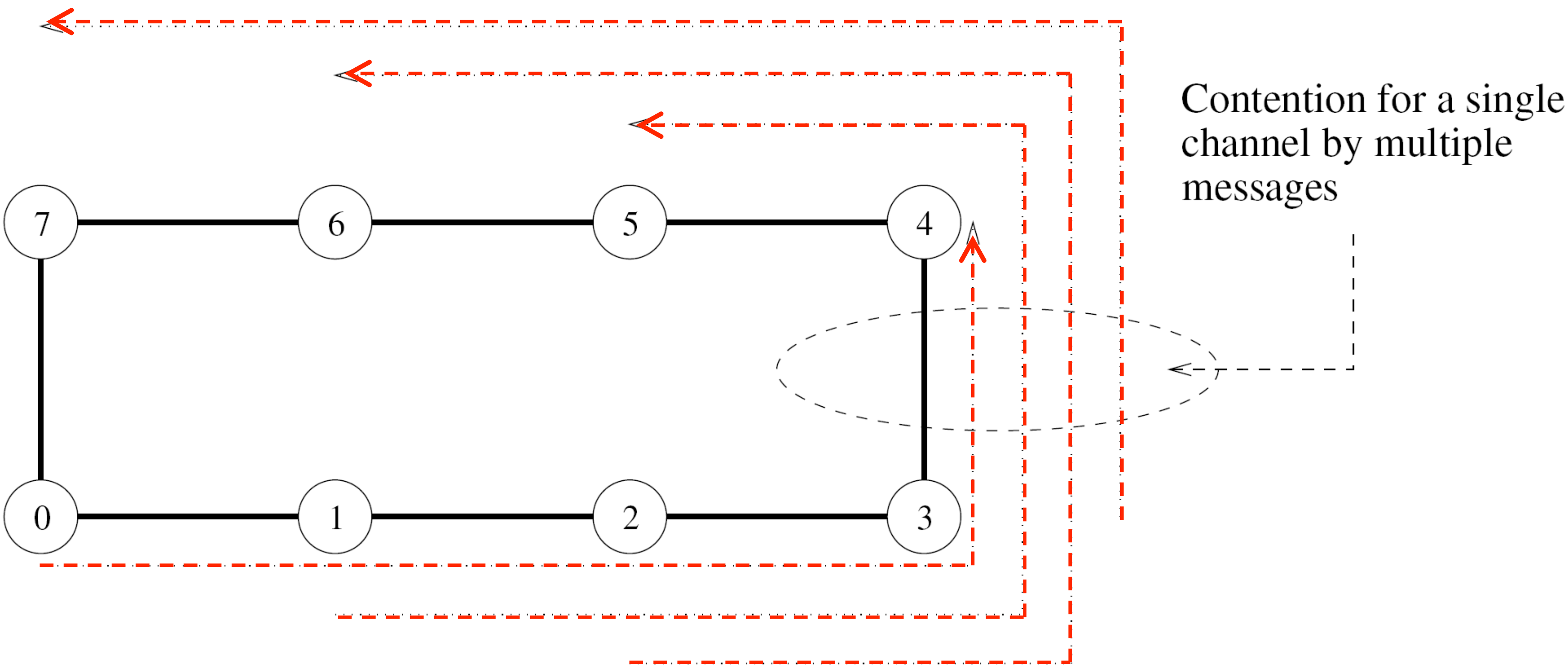
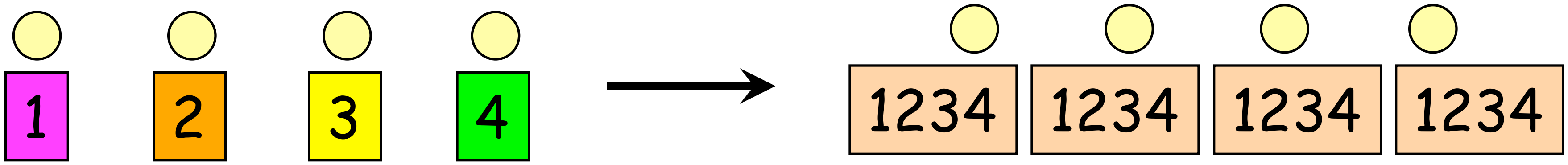


Figure 4.12 Contention for a channel when the communication step of Figure 4.11(c) for the hypercube is mapped onto a ring.

All-Reduce

- Each node starts with a buffer of size *m*.
- The final result is the same combination of all buffers on every node.
- Same as all-to-one reduce + one-to-all broadcast.
- Different from all-to-all reduce.



All-Reduce Algorithm

- Use all-to-all broadcast but
  - Combine messages instead of concatenating them.
  - The size of the messages does not grow.
  - Cost (in log*p* steps): T=$(t_s+t_w m)\log p$.

Prefix-Sum

- Given p numbers $n_0, n_1, \ldots, n_{p-1}$ (one on each node), the problem is to compute the sums $s_k = \sum_{i=0}^{k} n_i$ for all k between 0 and $p-1$.
- Initially, n_k is on the node labeled k, and at the end, the same node holds S_k.

Prefix-Sum Algorithm

```
procedure PREFIX_SUMS_HCUBE(my_id, my_number, d, result)
begin
    result := my_number;
    msg := result;
    for i := 0 to d - 1 do
        partner := my_id XOR 2^i;
        send msg to partner;
        receive number from partner;
        msg := msg + number;
        if (partner < my_id) then result := result + number;
    endfor;
end PREFIX_SUMS_HCUBE
```

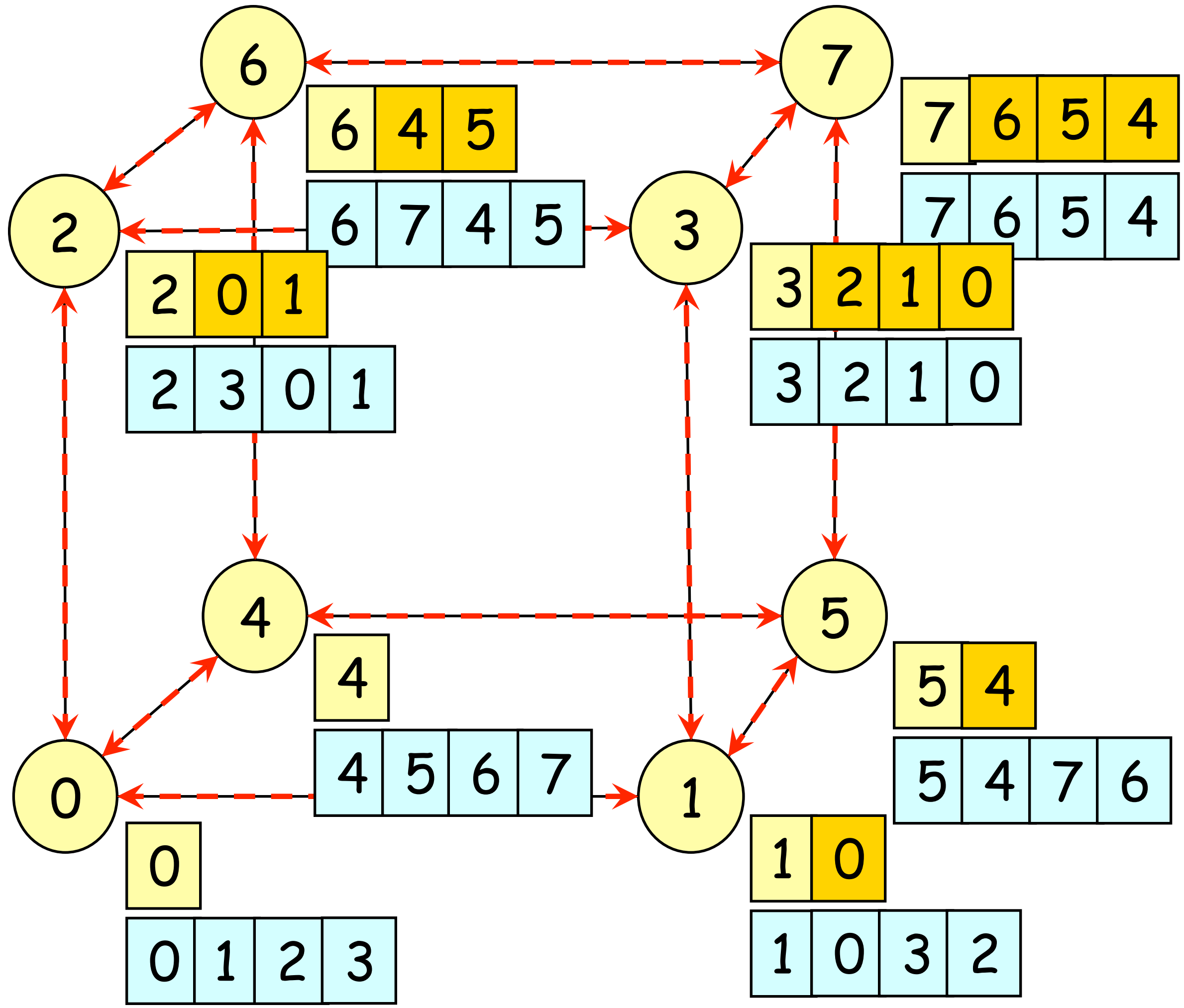
All-reduce

Prefix-sum

Algorithm 4.9 Prefix sums on a d-dimensional hypercube.

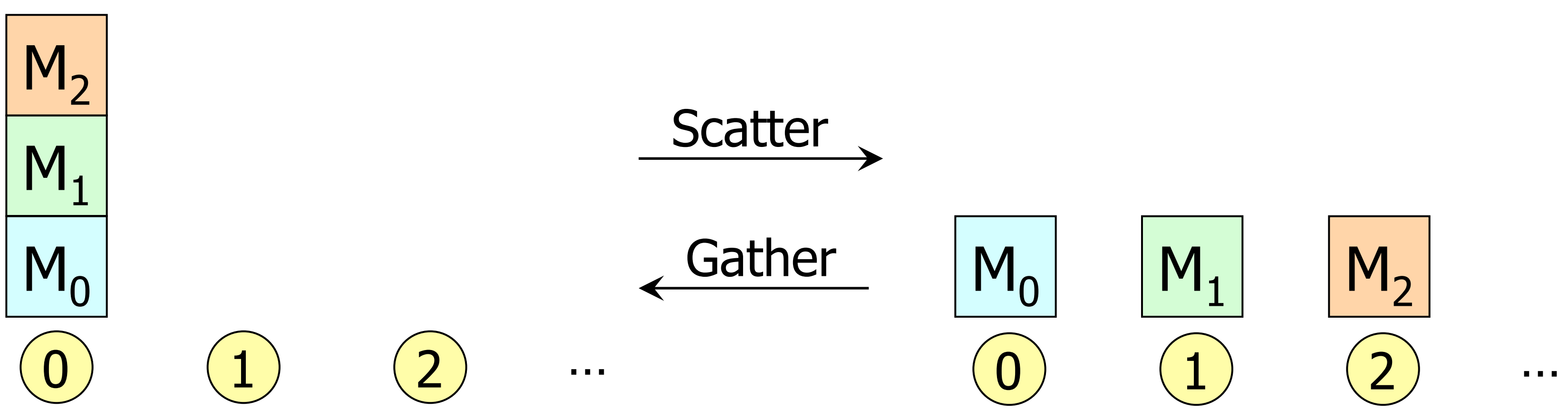
Prefix-Sum

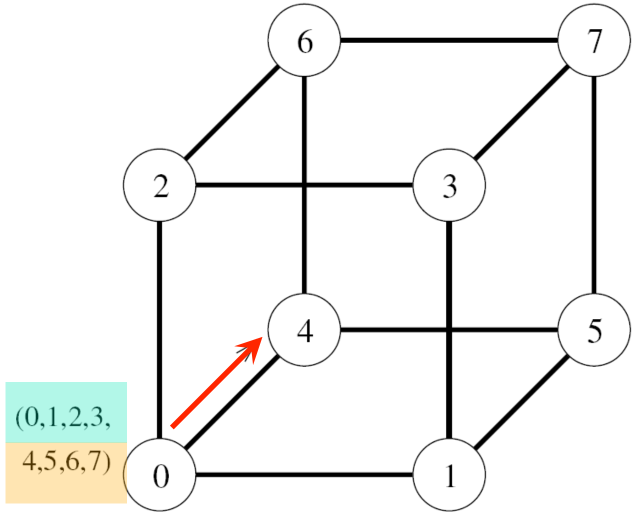
Buffer = all-reduce sum



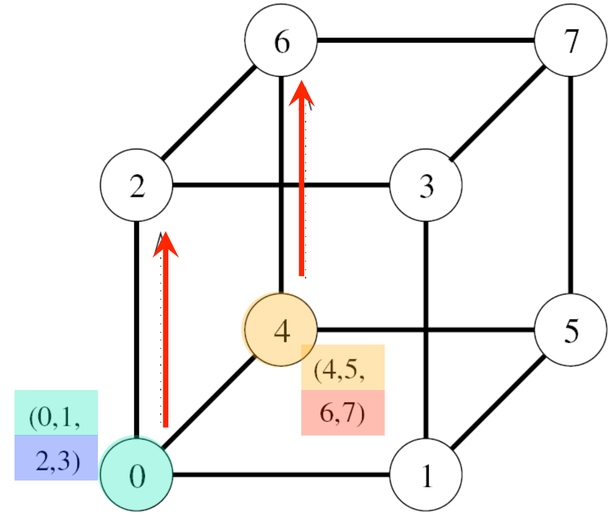
Scatter and Gather

- Scatter: A node sends a unique message to every other node – *unique per node*.
- Gather: Dual operation but the target node does not combine the messages into one.

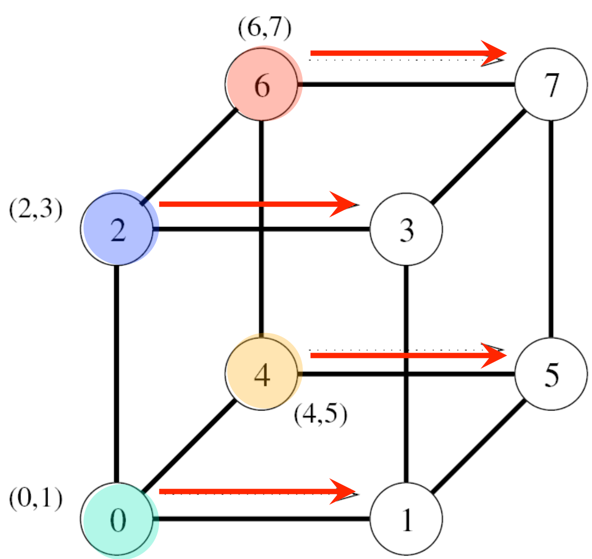




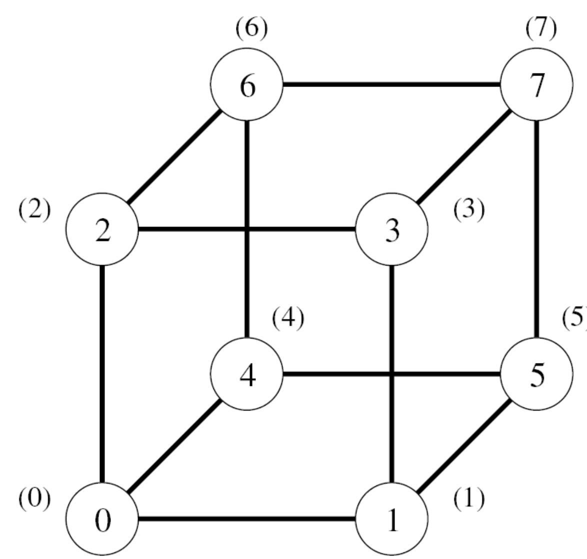
(a) Initial distribution of messages



(b) Distribution before the second step



(c) Distribution before the third step



(d) Final distribution of messages

Cost Analysis

- Number of steps: log*p*.
- Size transferred: *pm/2, pm/4,…,m*.
  - Geometric sum

$$p+\frac{p}{2}+\frac{p}{4}+\ldots+\frac{p}{2^n}=p\frac{1-\frac{1}{2^{n+1}}}{1-\frac{1}{2}}$$

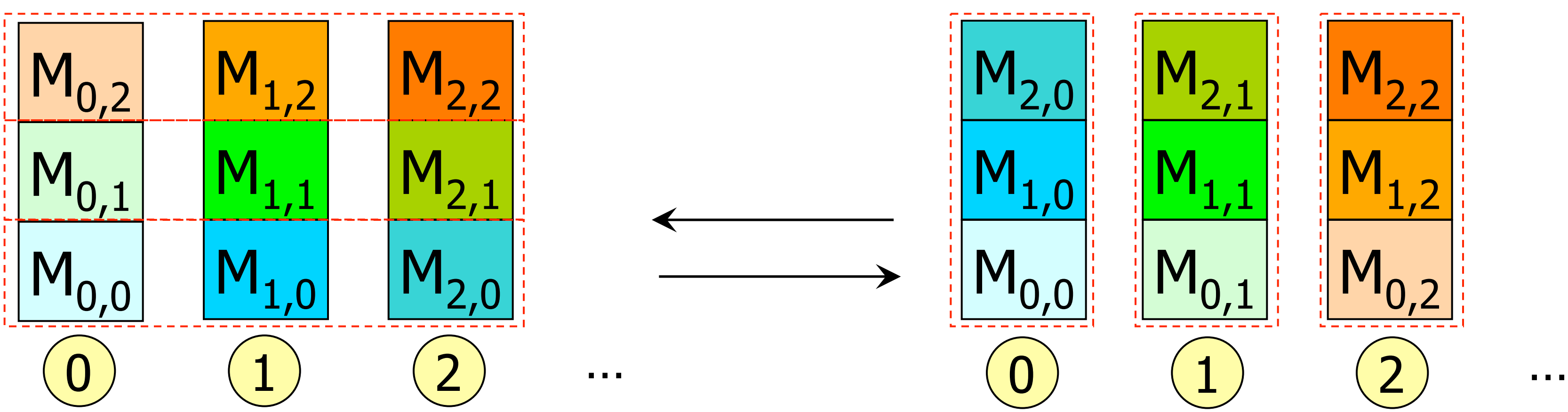
$$\frac{p}{2}+\frac{p}{4}+\ldots+\frac{p}{2^n}=2p(1-\frac{1}{2^{n+1}})-p=2p(1-\frac{1}{2p})-p=p-1$$

$$(2^{n+1}=2^{1+\log p}=2p)$$

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

All-to-All Personalized Communication

- Each node sends a *distinct* message to every other node.



Example: Transpose

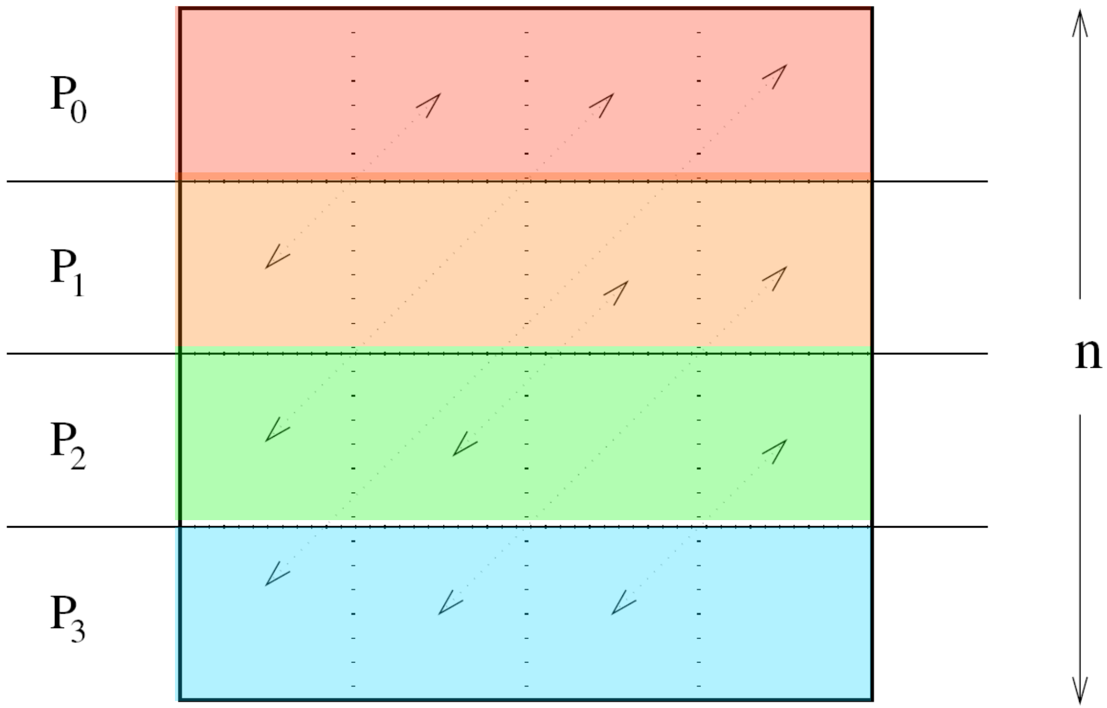
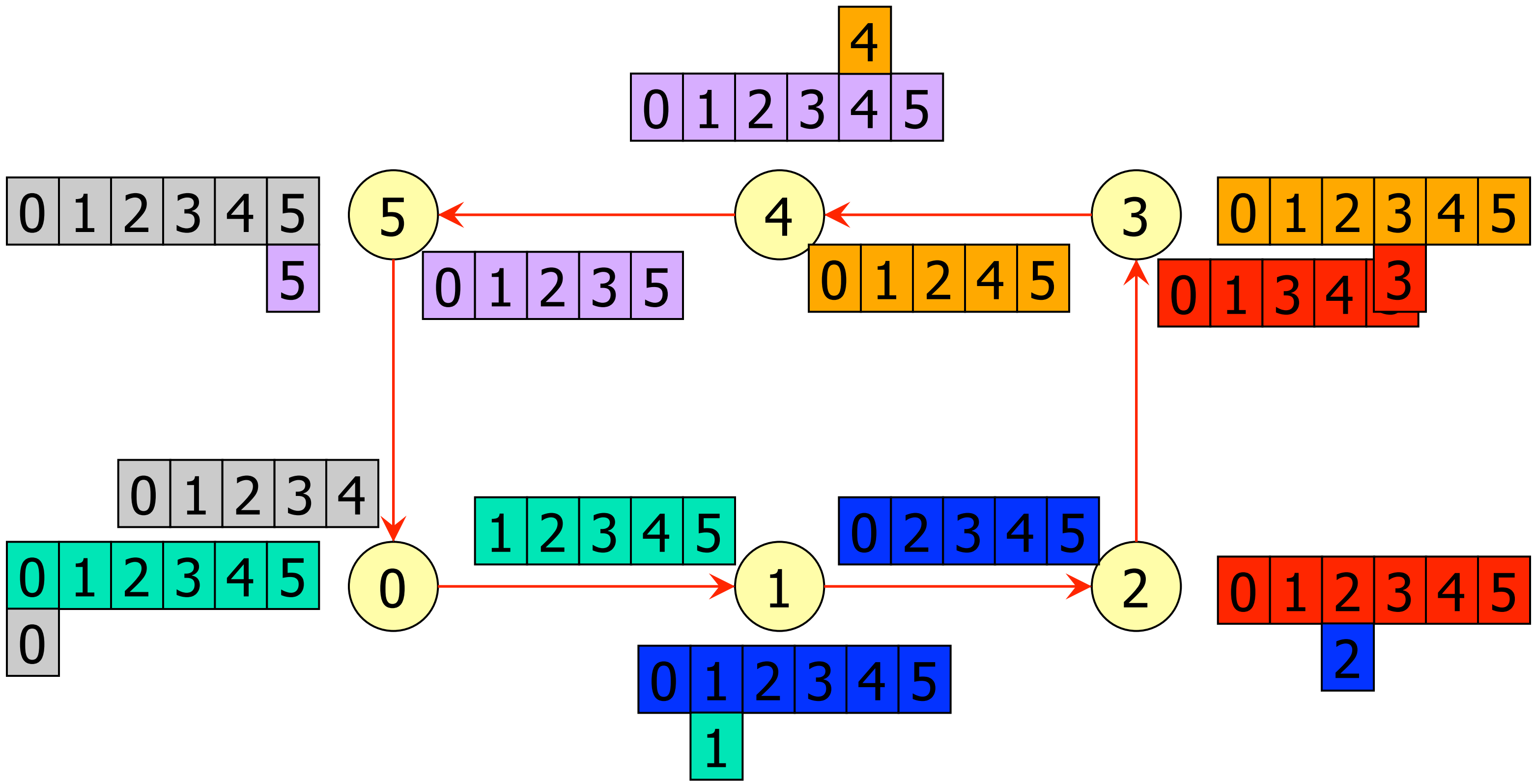
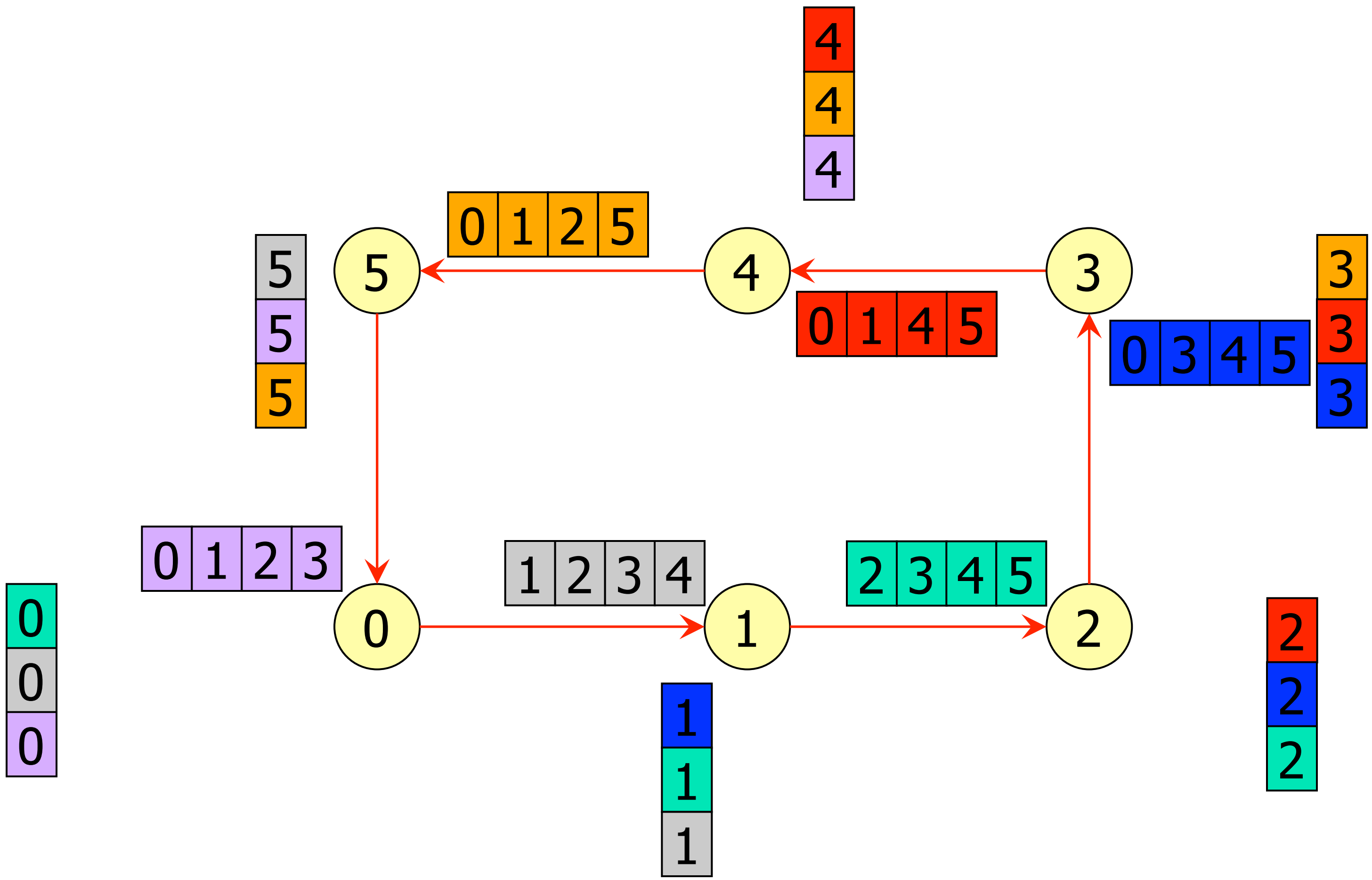


Figure 4.17 All-to-all personalized communication in transposing a 4×4 matrix using four processes.

Total Exchange on a Ring



Total Exchange on a Ring



Cost Analysis

- Number of steps: *p-1*.
- Size transmitted: *m(p-1),m(p-2)…,m*.

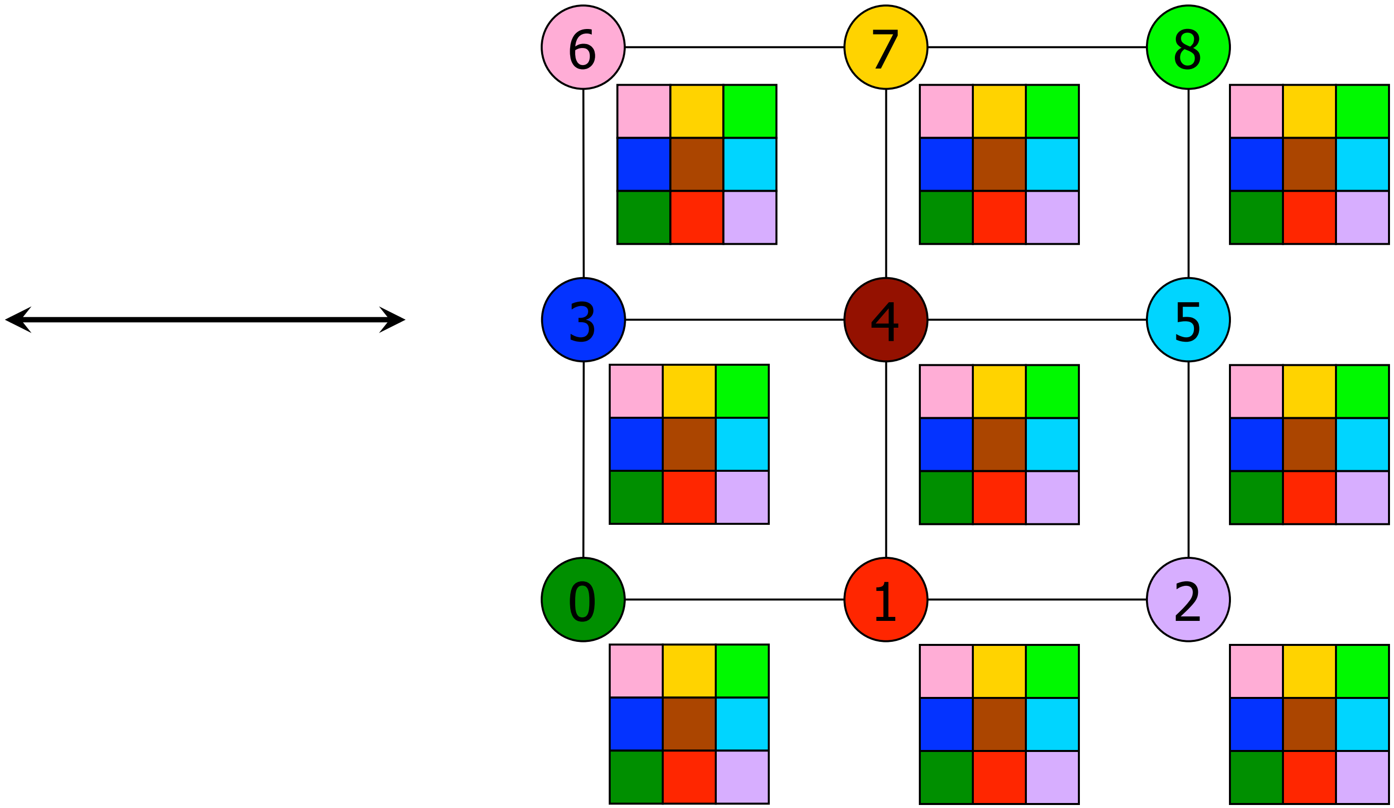
$$T = t_s(p-1) + \sum_{i=1}^{p-1} i t_w m = (t_s + t_w mp/2)(p-1)$$

Optimal

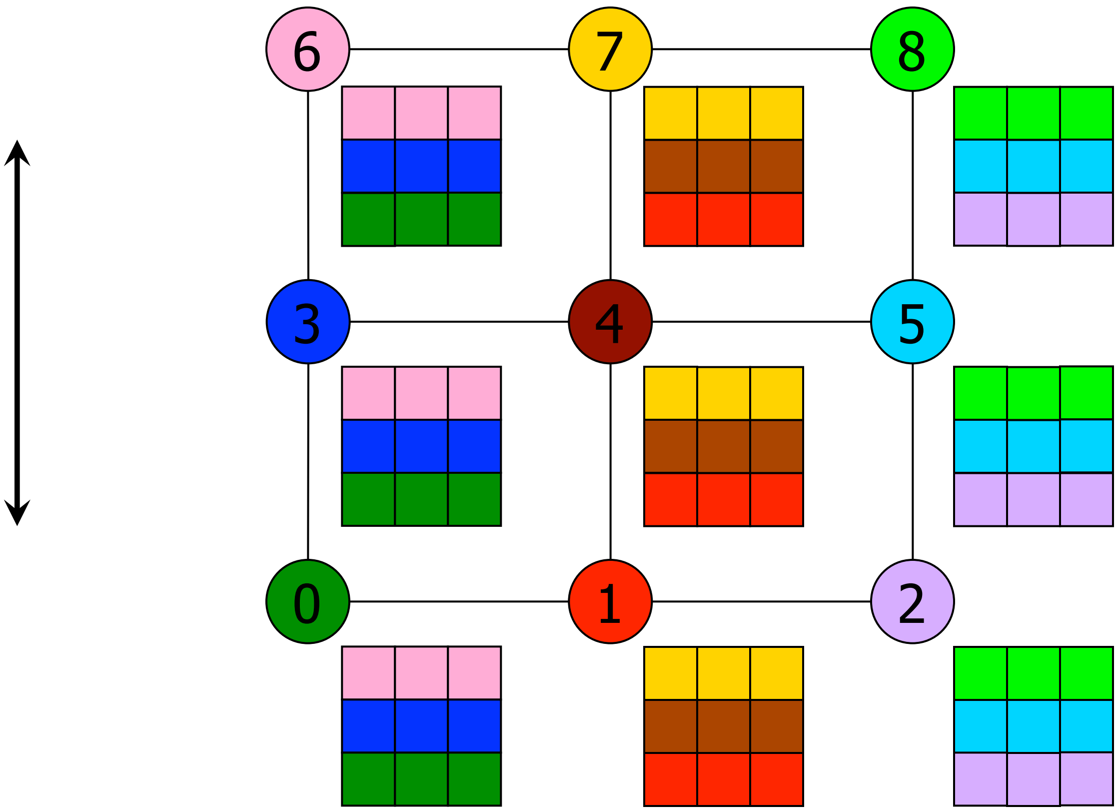
Optimal?

- Check the lowest bound for communication and compare to the one we have.
  - Average distance a packet travels = p/2.
  - There are p nodes that need to transmit m(p-1) words.
  - Total traffic = m(p-1)*p/2*p.
  - Number of link that support the load = p, so communication time ≥ $t_w m(p-1)p/2$.

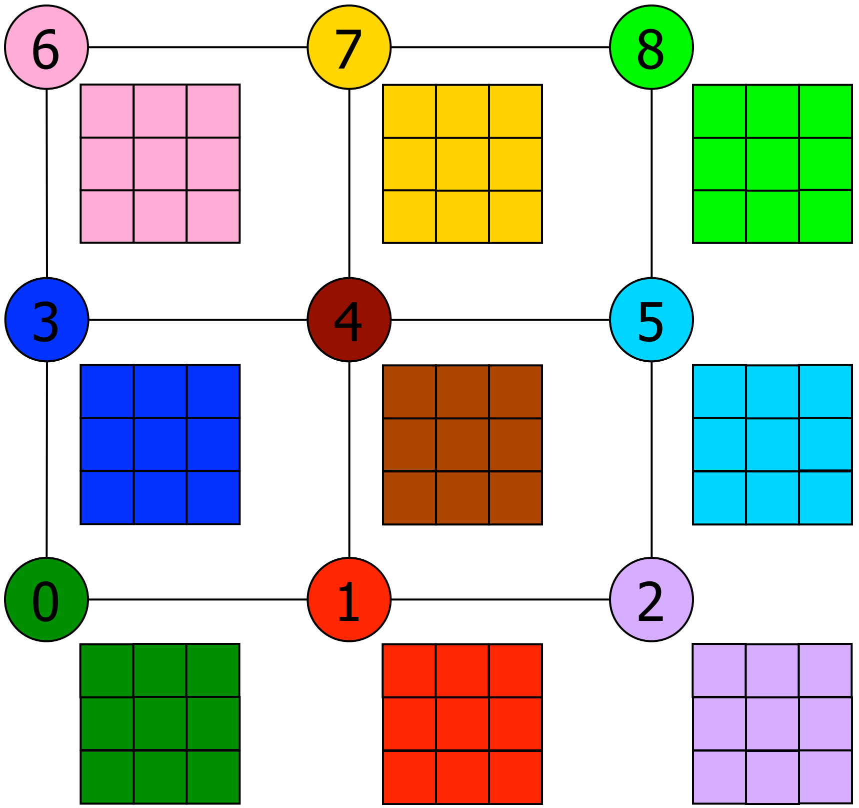
Total Exchange on a Mesh



Total Exchange on a Mesh



Total Exchange on a Mesh



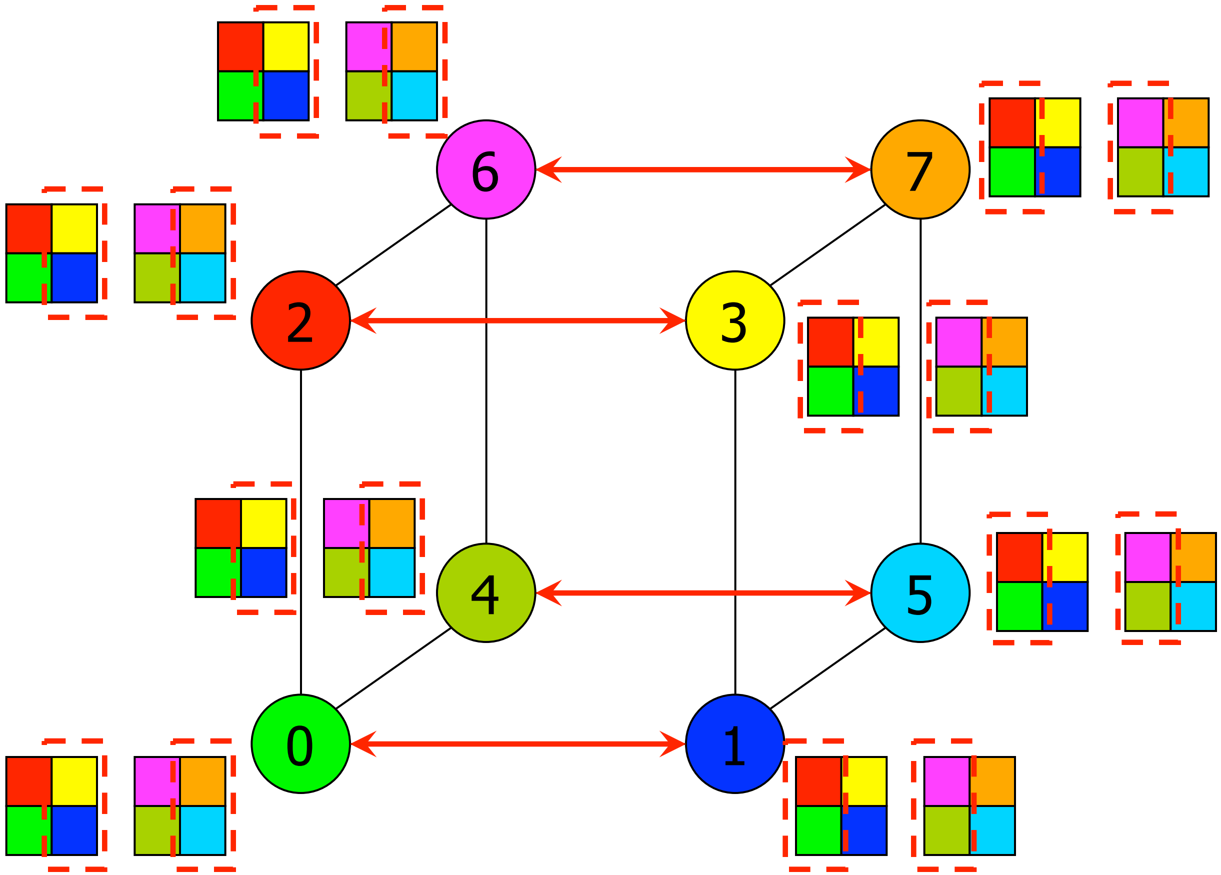
Cost Analysis

- Substitute *p* by $\sqrt{p}$ (number of nodes per dimension).
- Substitute message size *m* by $m\sqrt{p}$.
- Cost is the same for each dimension.
- $T=(2t_s+t_w mp)(\sqrt{p}-1)$

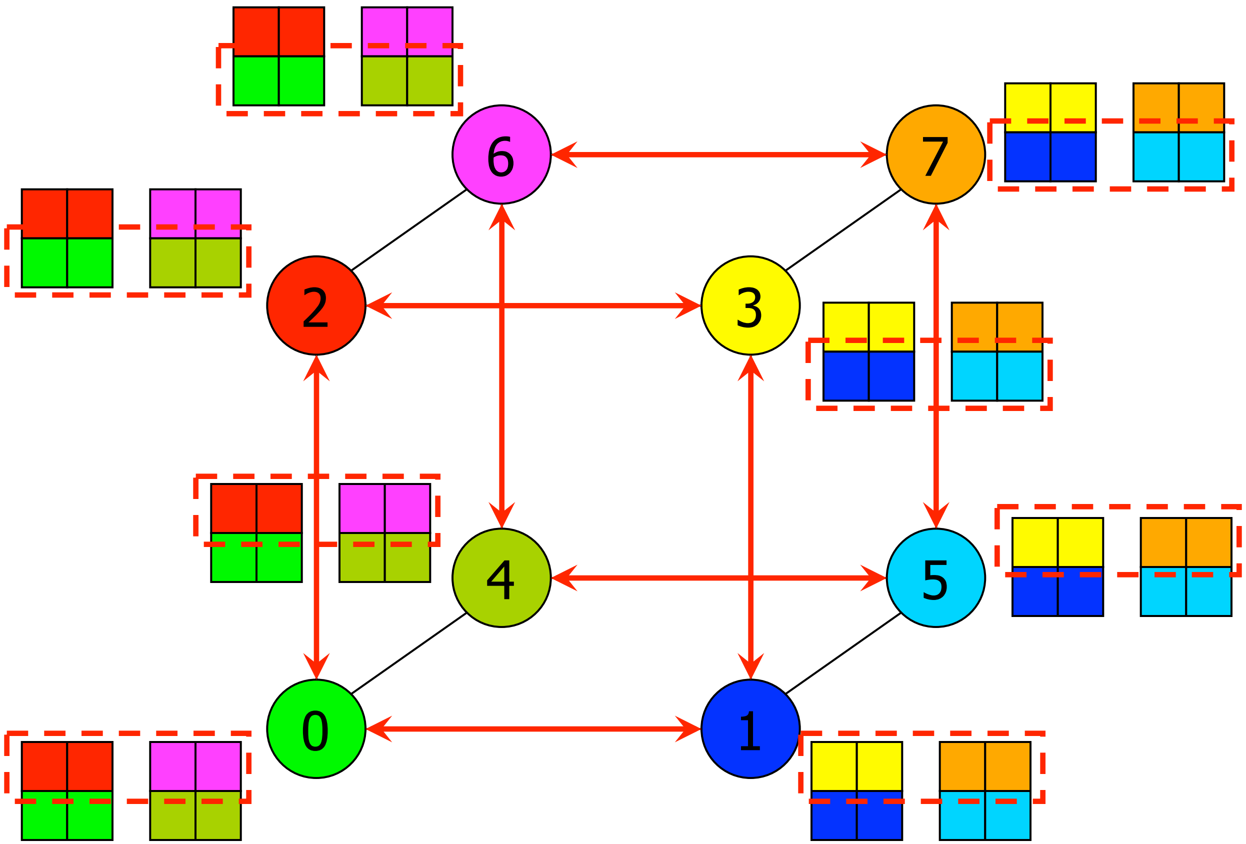
Total Exchange on a Hypercube

- Generalize the mesh algorithm to log*p* steps = number of dimensions, with 2 nodes per dimension.
- Same procedure as all-to-all broadcast.

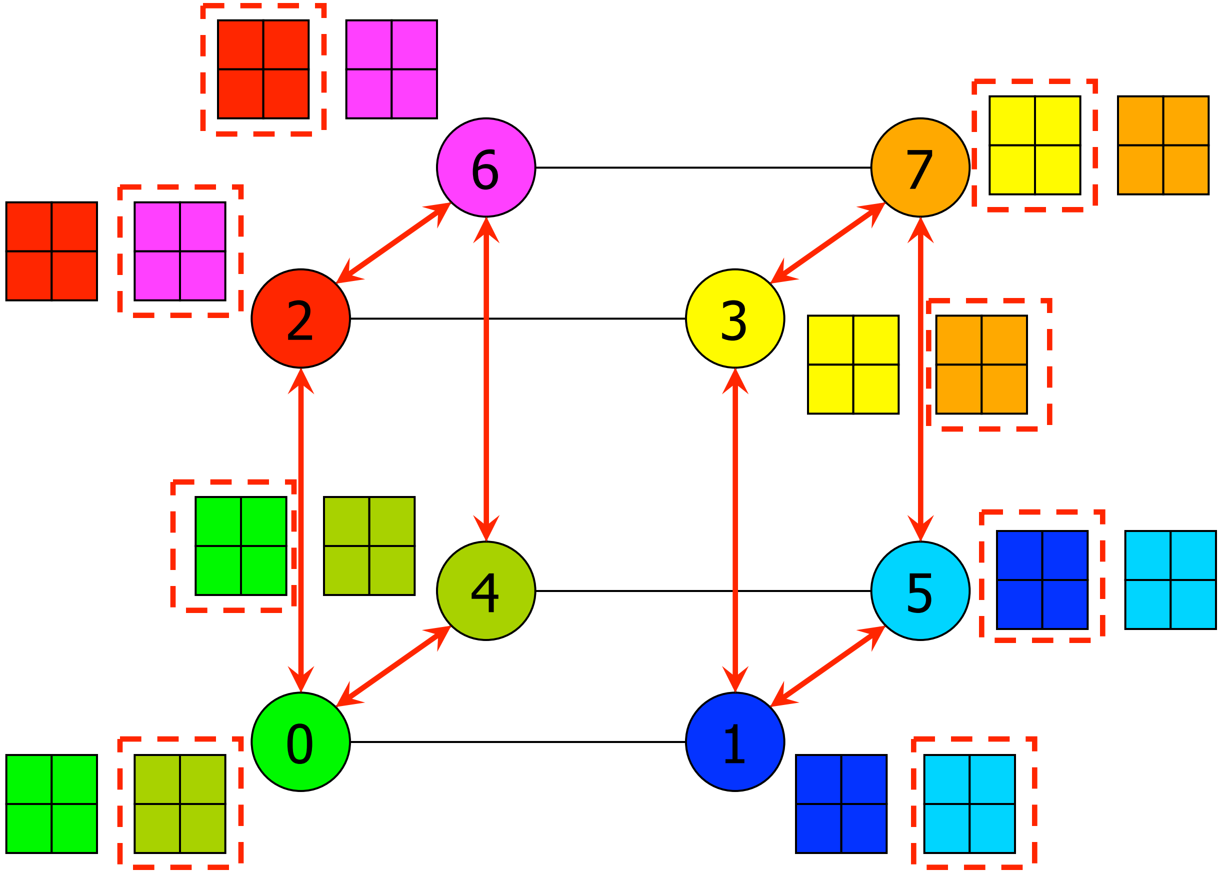
Total Exchange on a Hypercube



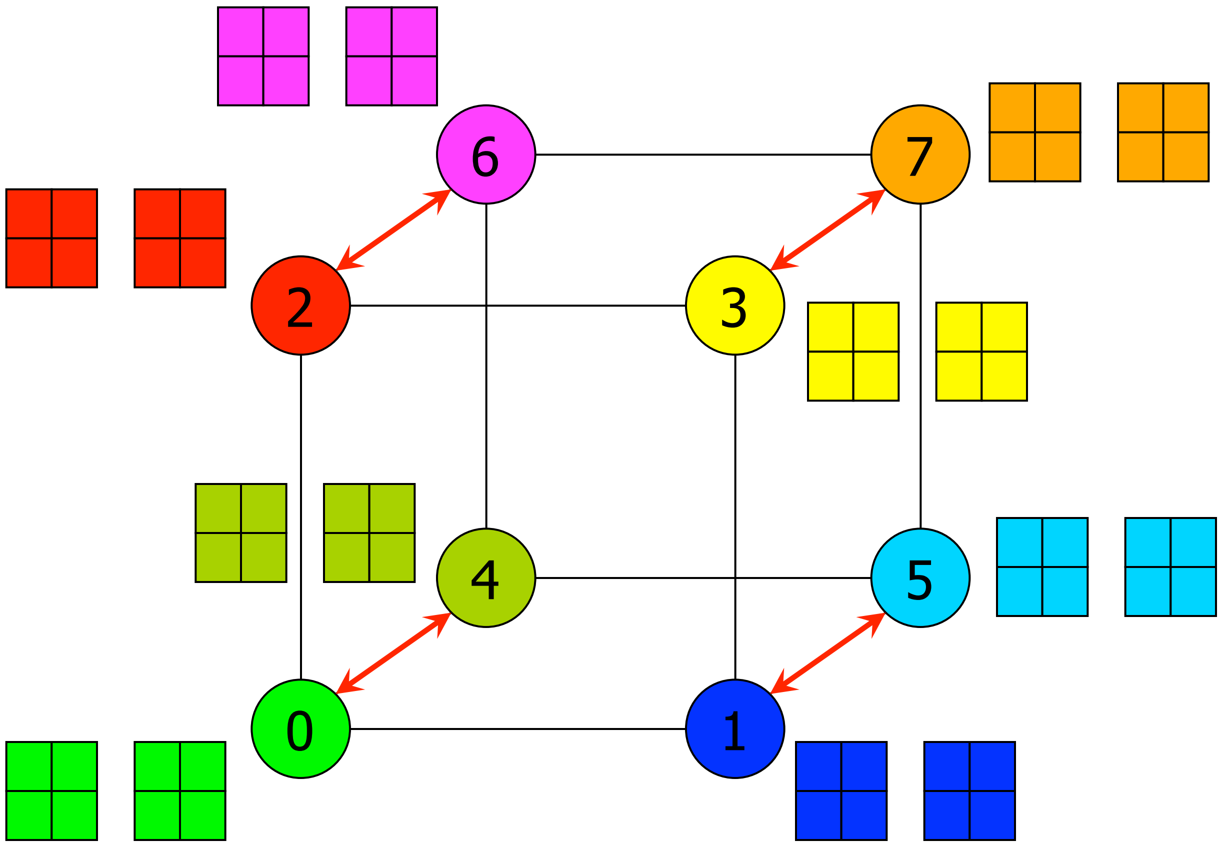
Total Exchange on a Hypercube



Total Exchange on a Hypercube



Total Exchange on a Hypercube



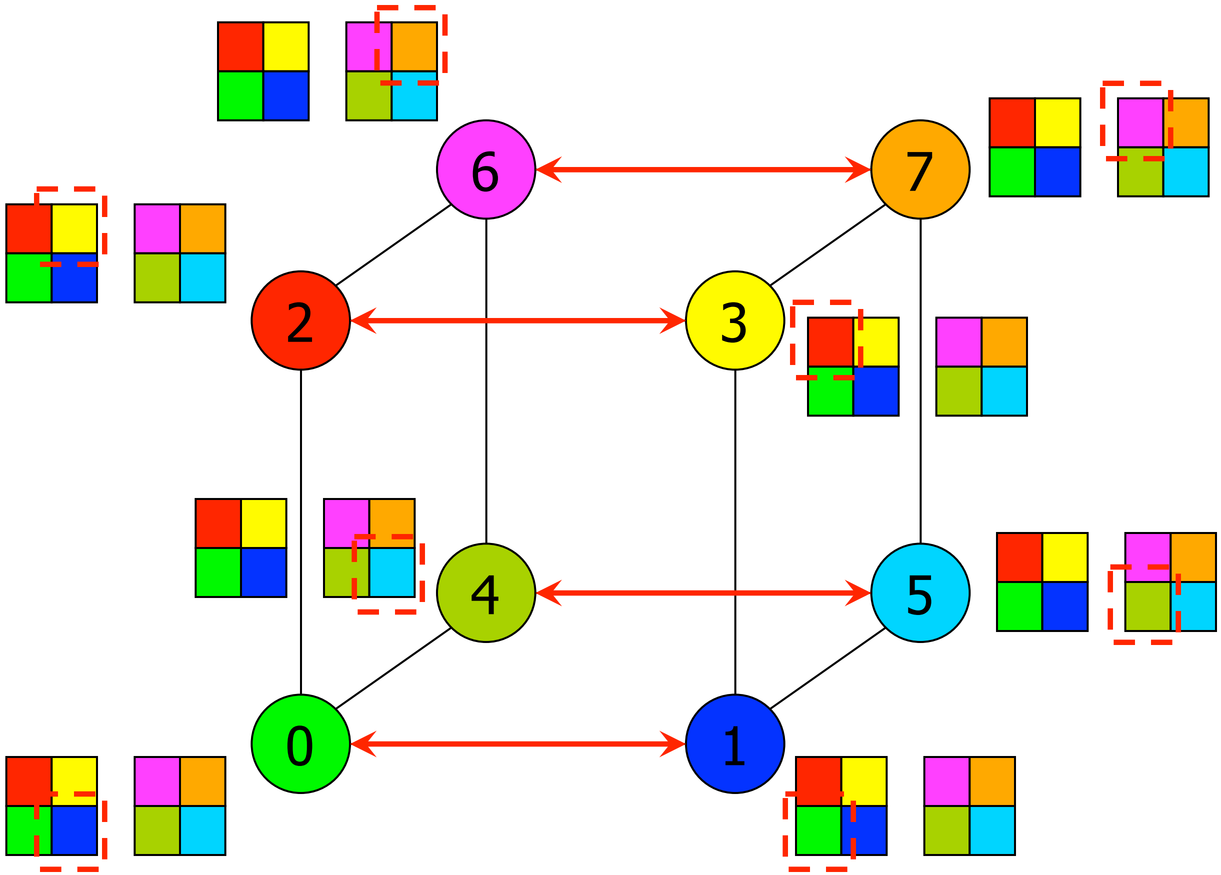
Cost Analysis

- Number of steps: log*p*.
- Size transmitted per step: *pm/2*.
- Cost: $T=(t_s+t_w mp/2)\log p$.
- Optimal? NO
- Each node sends and receives m(p-1) words. Average distance = (log*p*)/2. Total traffic = $p*m(p-1)*\log p/2$.
- Number of links = $p \log p/2$.
- Time lower bound = $t_w m(p-1)$.

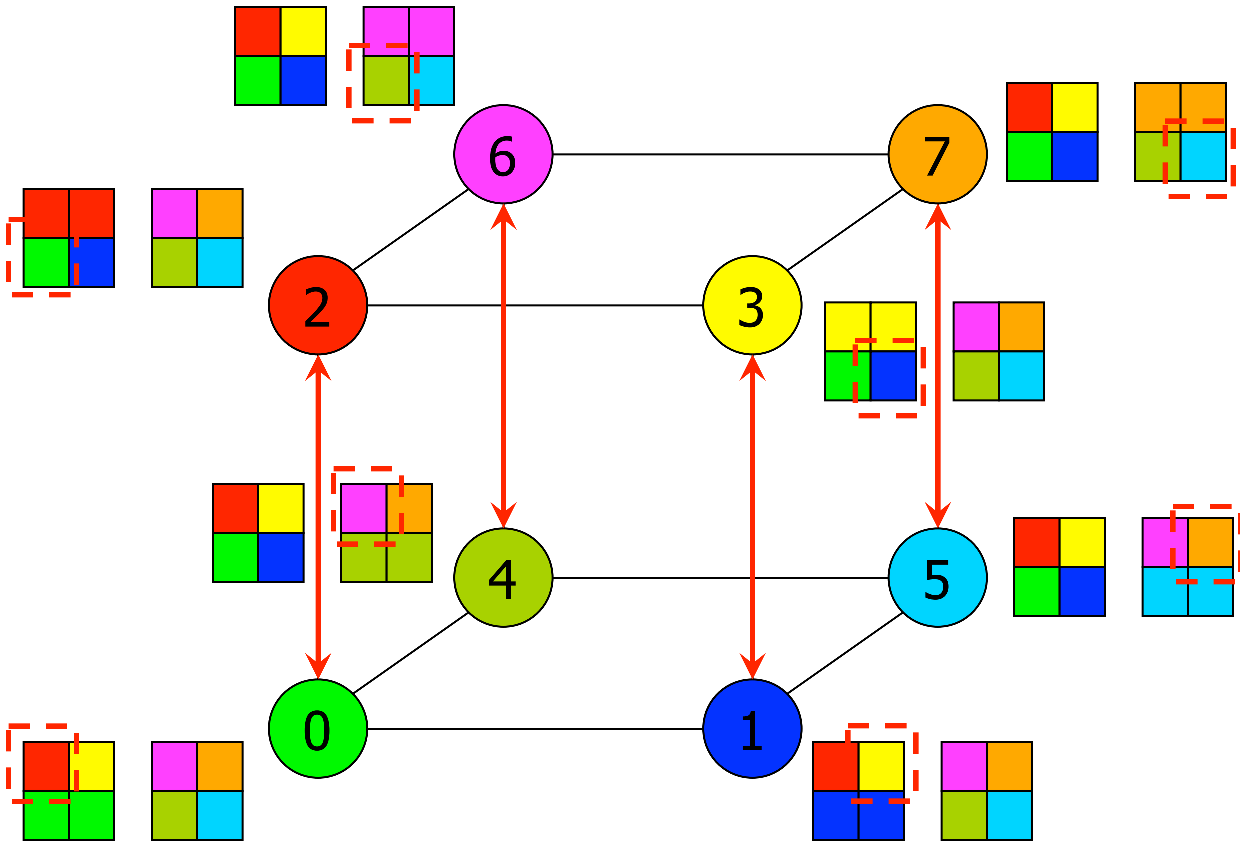
An Optimal Algorithm

- Have every pair of nodes communicate directly with each other – p-1 communication steps – but without congestion.
- At j^{th} step node i communicates with node (i xor j) with E-cube routing.

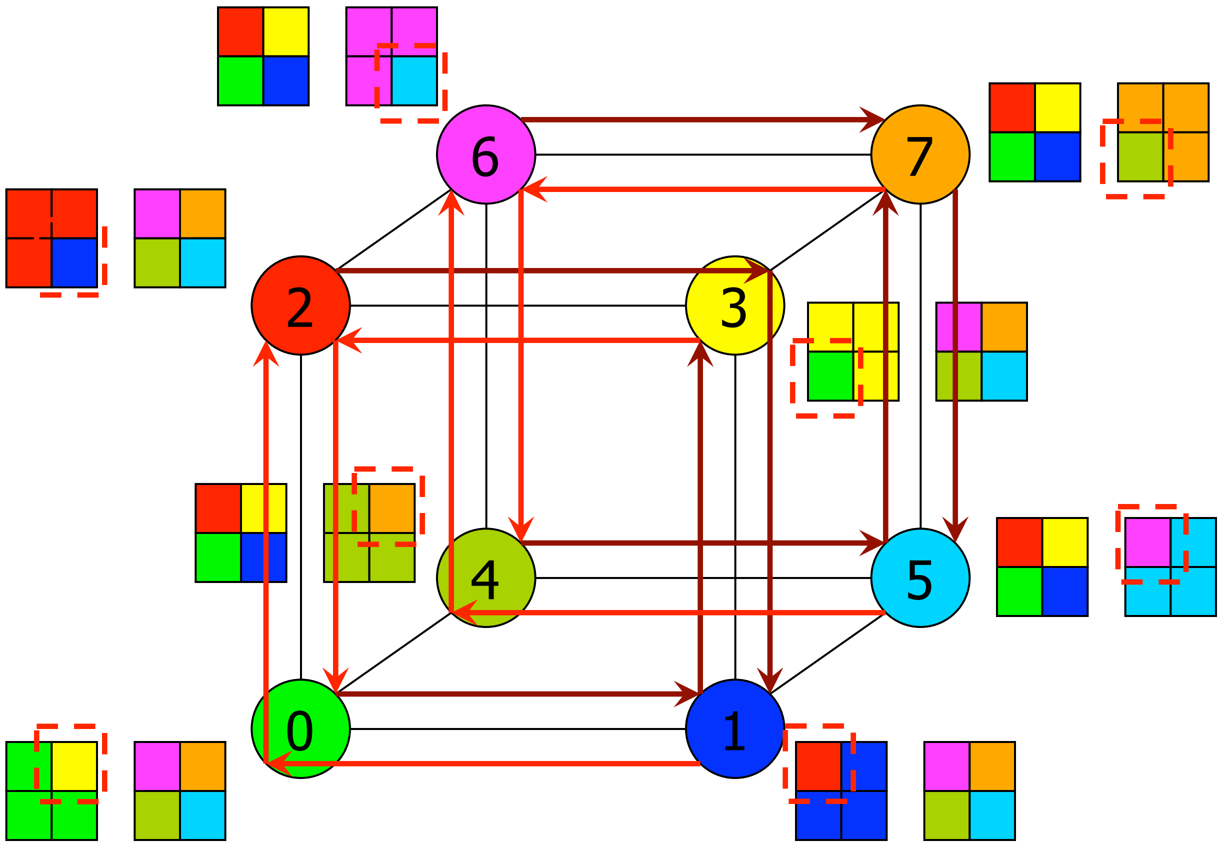
Total Exchange on a Hypercube



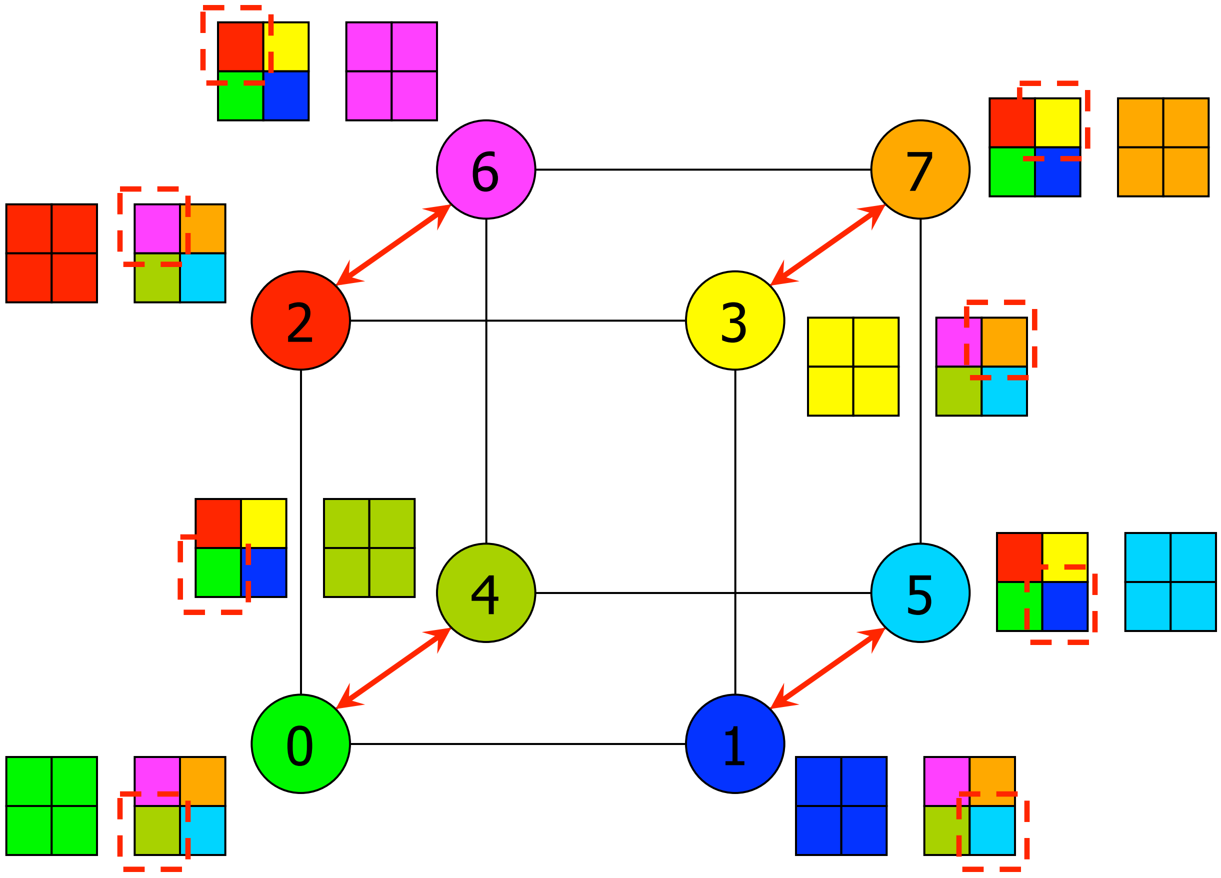
Total Exchange on a Hypercube



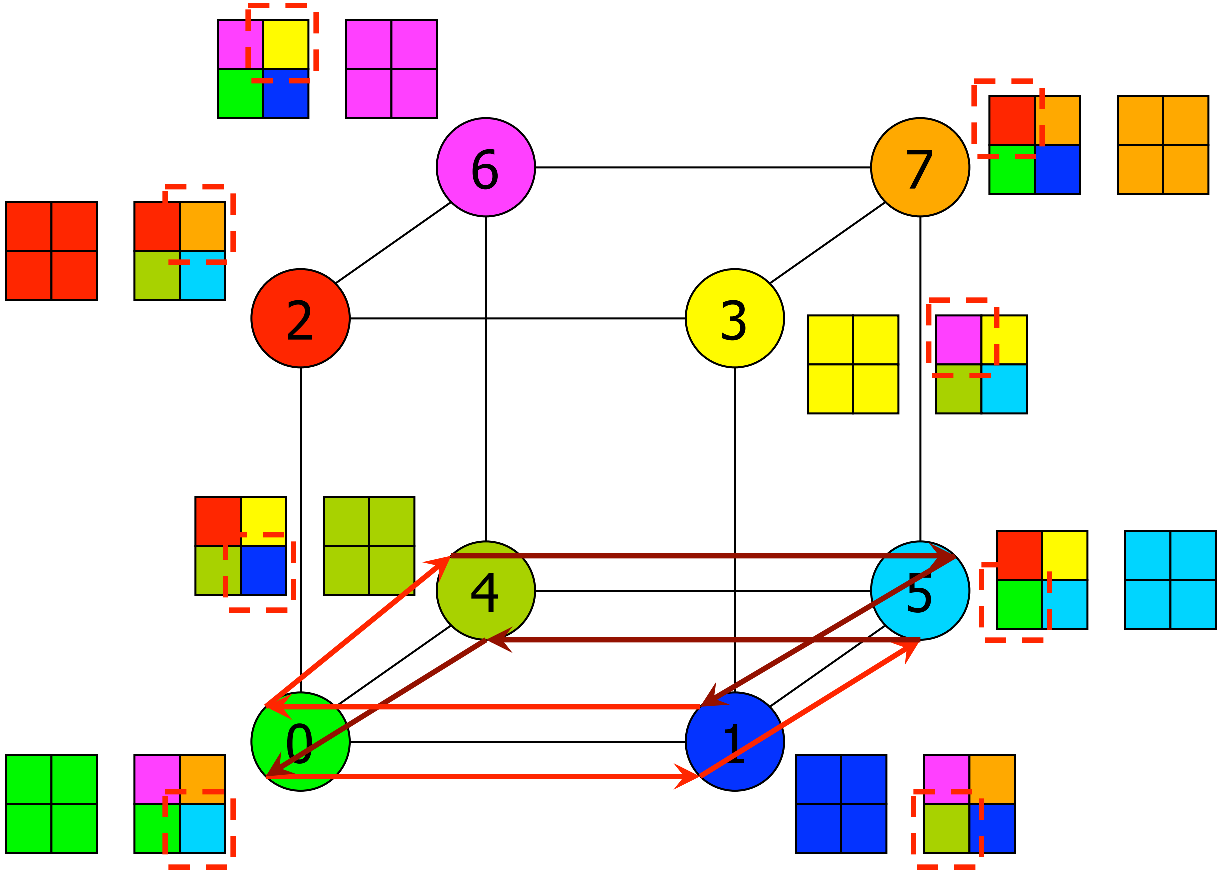
Total Exchange on a Hypercube



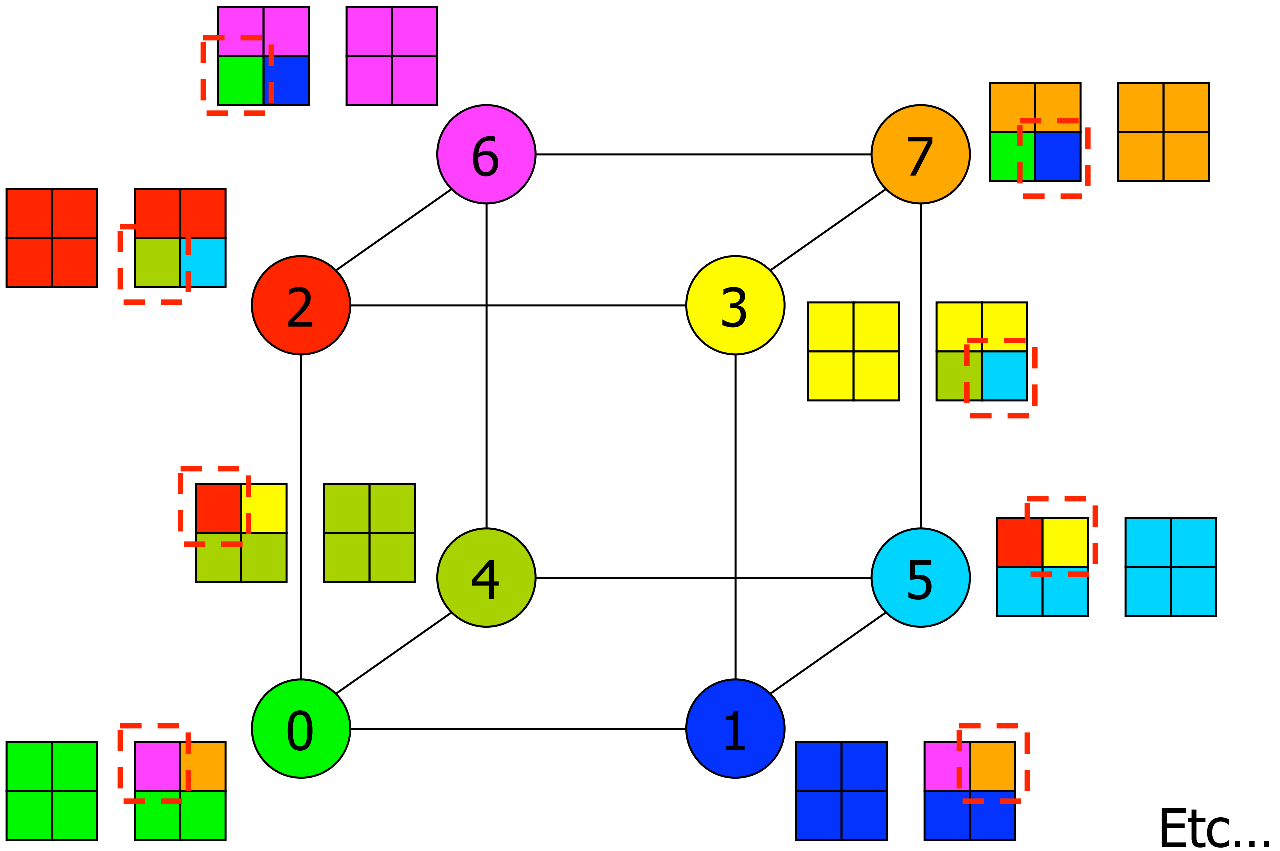
Total Exchange on a Hypercube



Total Exchange on a Hypercube



Total Exchange on a Hypercube



Cost Analysis

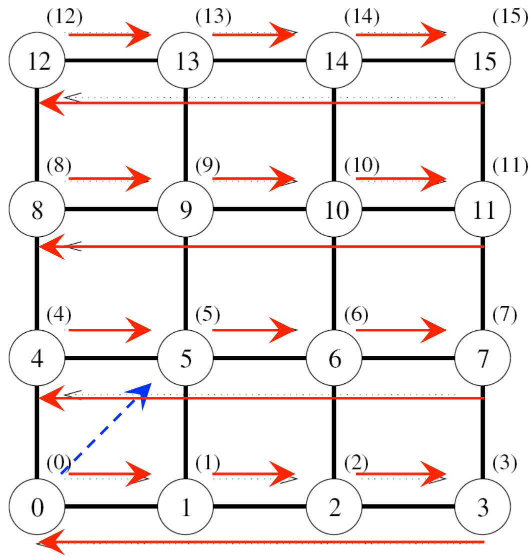
- Remark: Transmit less, only what is needed, but more steps.
- Number of steps: $p-1$.
- Transmission: size m per step.
- Cost: $T=(t_s+t_w m)(p-1)$.
- Compared with $T=(t_s+t_w mp/2)\log p$.
- Previous algorithm better for small messages.

Circular Shift

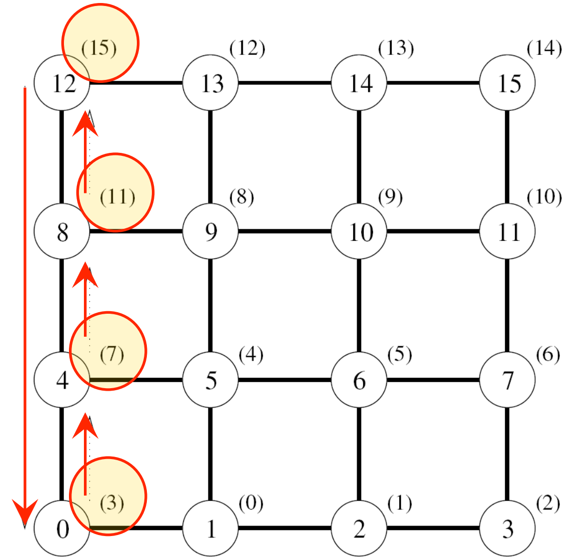
- It's a particular permutation.
- Circular q-shift: Node *i* sends data to node *(i+q) mod p* (in a set of p nodes).
- Useful in some matrix operations and pattern matching.
- Ring: intuitive algorithm in min{q,p-q} neighbor to neighbor communication steps. Why?

Circular 5-shift on a mesh.

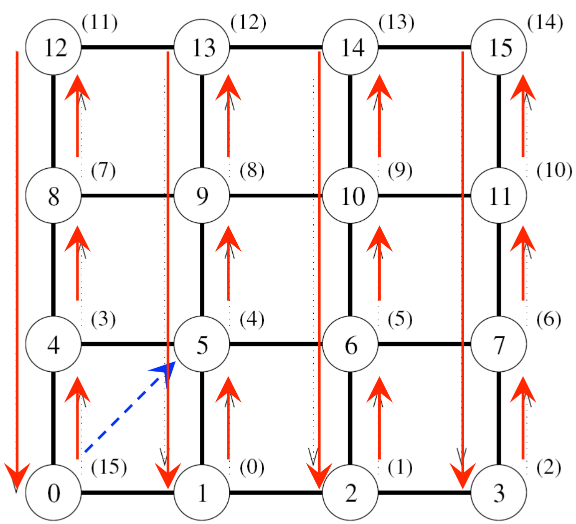
$q \bmod \sqrt{p}$ on rows
compensate
$\lfloor q / \sqrt{p} \rfloor$ on colums



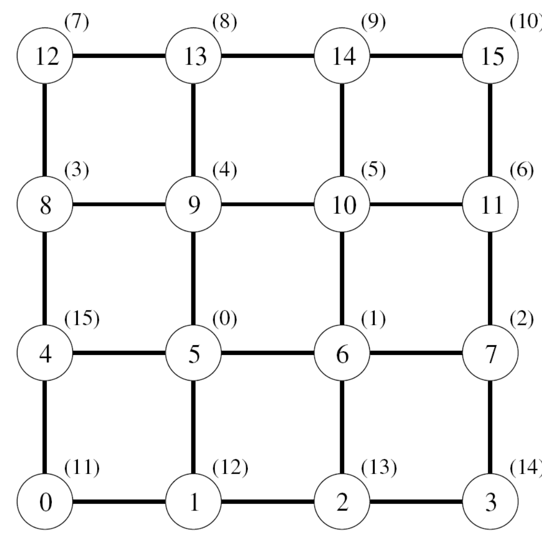
(a) Initial data distribution and the first communication step



(b) Step to compensate for backward row shifts



(c) Column shifts in the third communication step



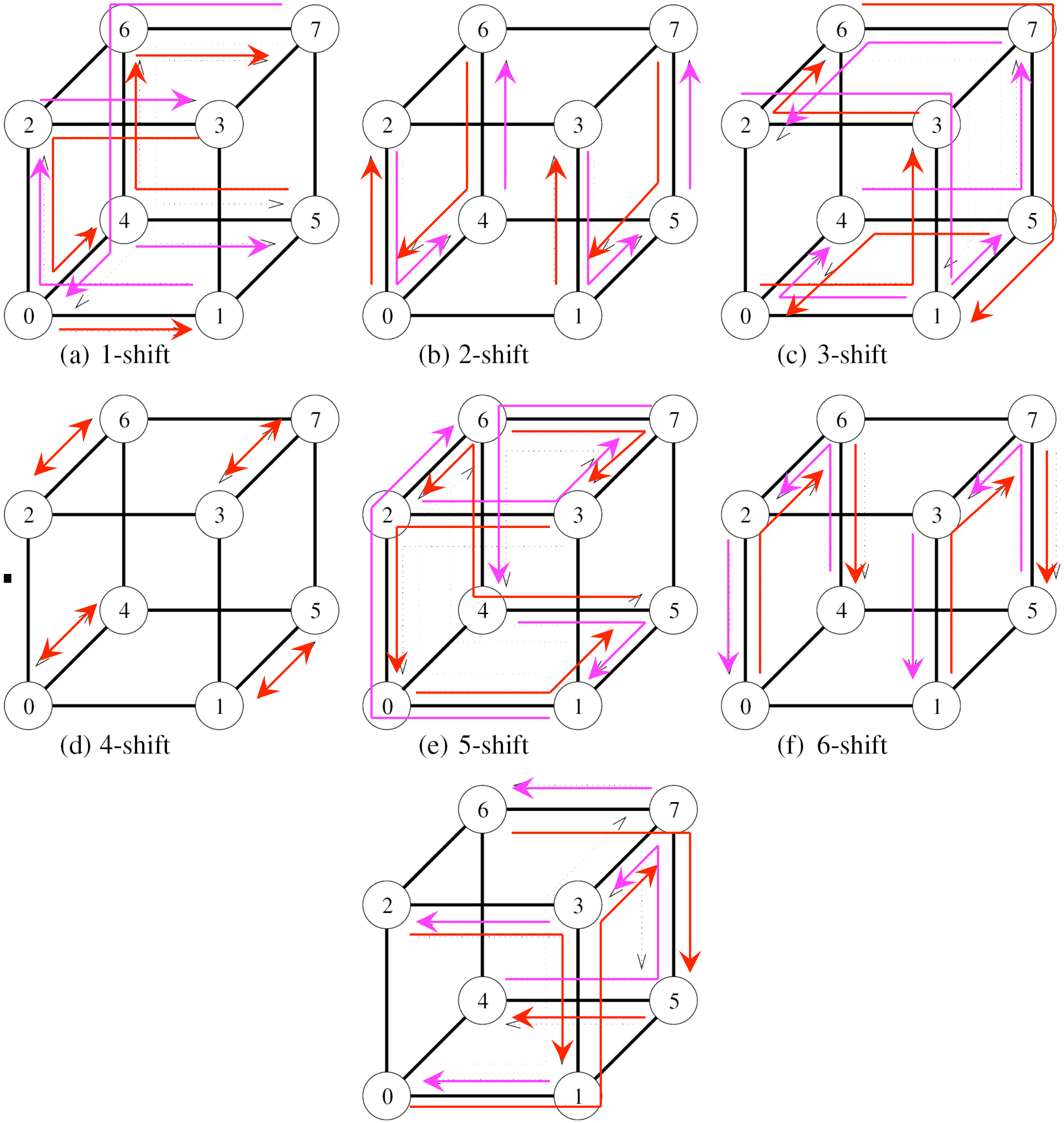
(d) Final distribution of the data

Circular Shift on a Hypercube

- Map a linear array with *2^d* nodes onto a hypercube of dimension *d*.
- Expand q shift as a sum of powers of 2 (e.g. 5-shift = 2^0+2^2).
- Perform the decomposed shifts.
- Use bi-directional links for "forward" (shift itself) and "backward" (rotation part)… log*p* steps.

Or better: Direct E-cube routing.

q-shifts on a 8-node hypercube.



(a) 1-shift (b) 2-shift (c) 3-shift (d) 4-shift (e) 5-shift (f) 6-shift (g) 7-shift

Improving Performance

- So far messages of size *m* were not split.
- If we split them into *p* parts:
  - One-to-all broadcast = scatter + all-to-all broadcast of messages of size *m/p*.
  - All-to-one reduction = all-to-all reduce + gather of messages of size *m/p*.
  - All-reduce = all-to-all reduction + all-to-all broadcast of messages of size *m/p*.