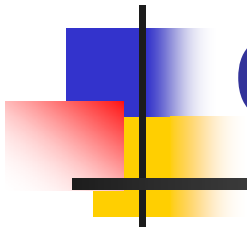
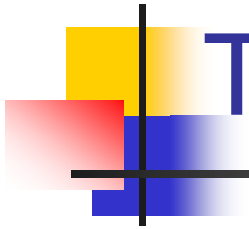


Basic Communication Operations

Alexandre David

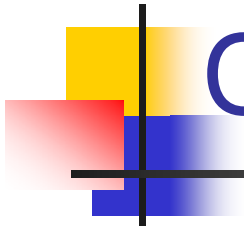
B2-206



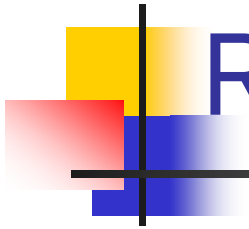
Today

- One-to-all broadcast & all-to-one reduction (4.1).
- All-to-all broadcast and reduction (4.2).
- All-reduce and prefix-sum operations (4.3).

Collective Communication Operations

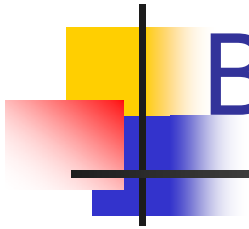


- Represent regular communication patterns.
- Used extensively in most data-parallel algorithms.
- Critical for efficiency.
- Available in most parallel libraries.
- Very useful to “get started” in parallel processing.



Reminder

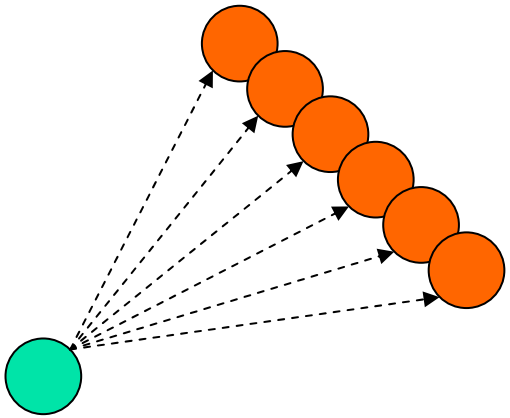
- Result from previous analysis:
  - Data transfer time is roughly the same between *all pairs* of nodes.
  - Homogeneity true on modern hardware (randomized routing, cut-through routing...).
    - $t_s + mt_w$
    - Adjust t_w for congestion: effective t_w.
- Model: bidirectional links, single port.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



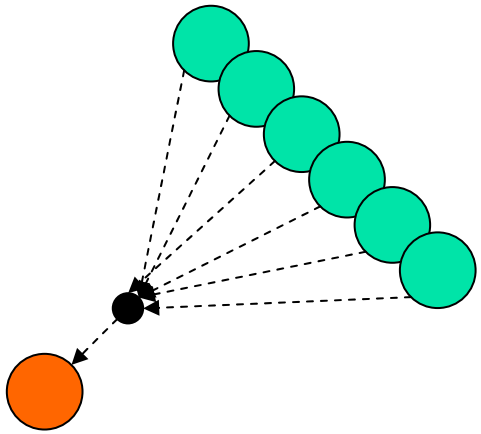
Broadcast/Reduction

- 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

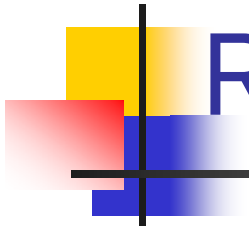


Broadcast



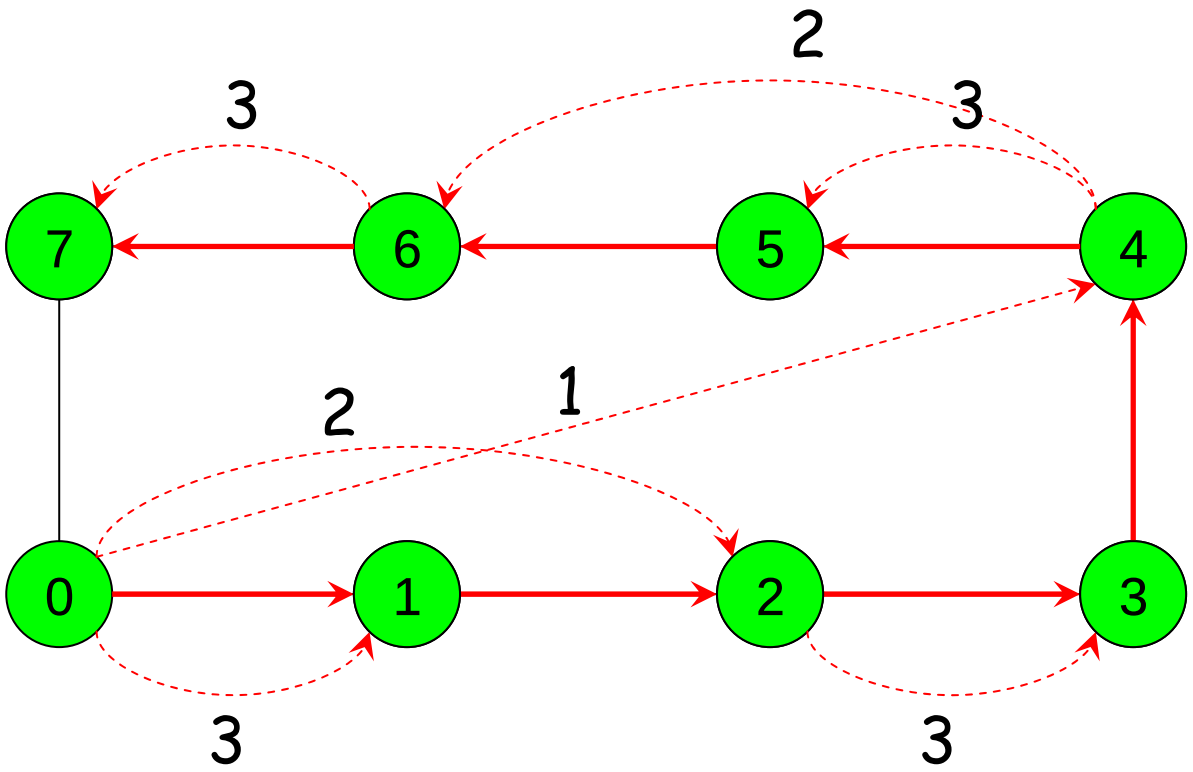
Reduce

One-to-All Broadcast – Ring/Linear Array

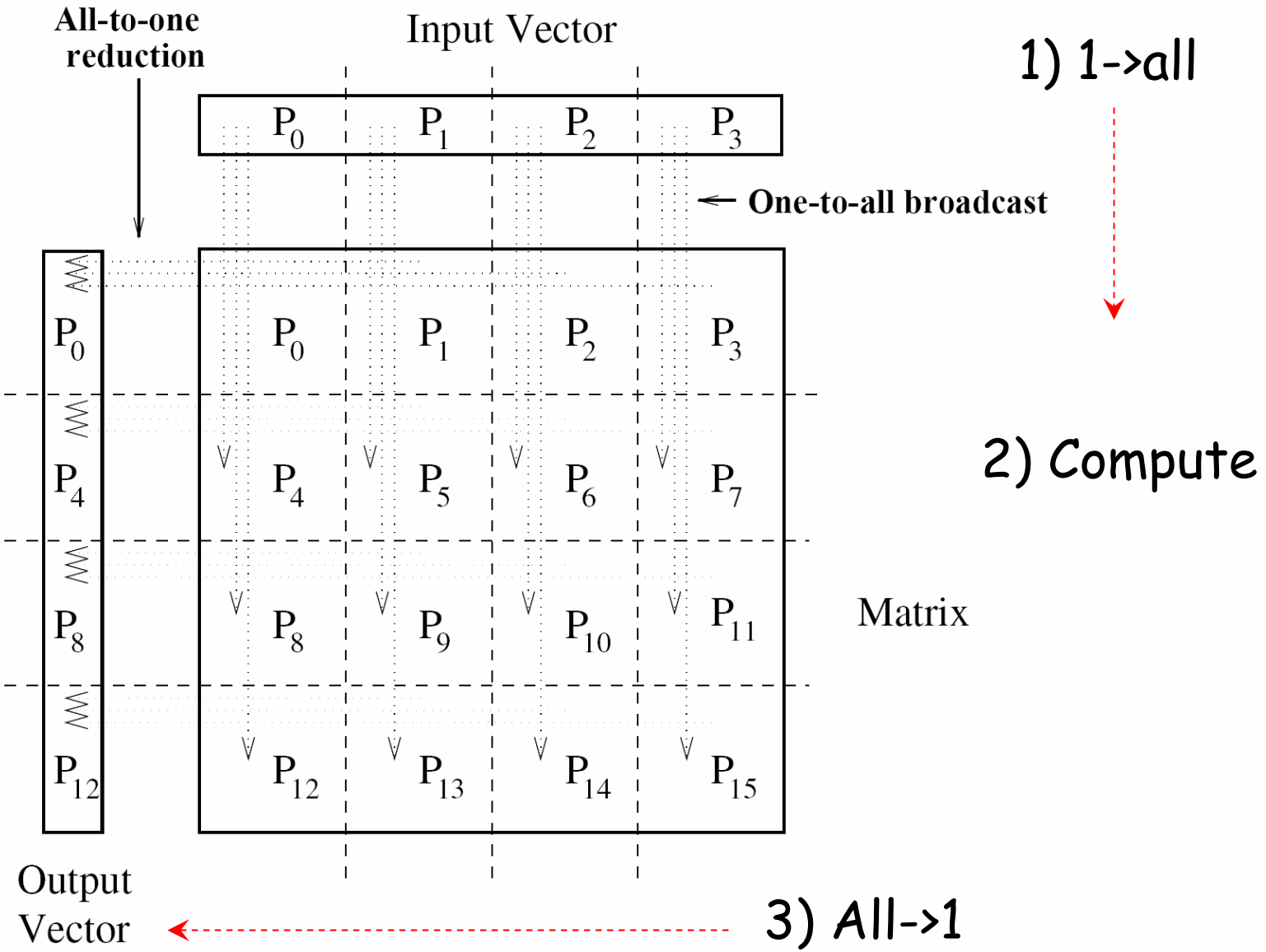


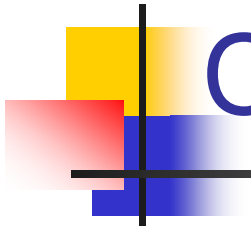
- Naïve approach: send sequentially.
  - Bottleneck.
  - Poor utilization of the network.
- Recursive doubling:
  - Broadcast in $\log p$ steps (instead of p).
  - Divide-and-conquer type of algorithm.
  - Reduction is similar.

Recursive Doubling



Example: Matrix*Vector

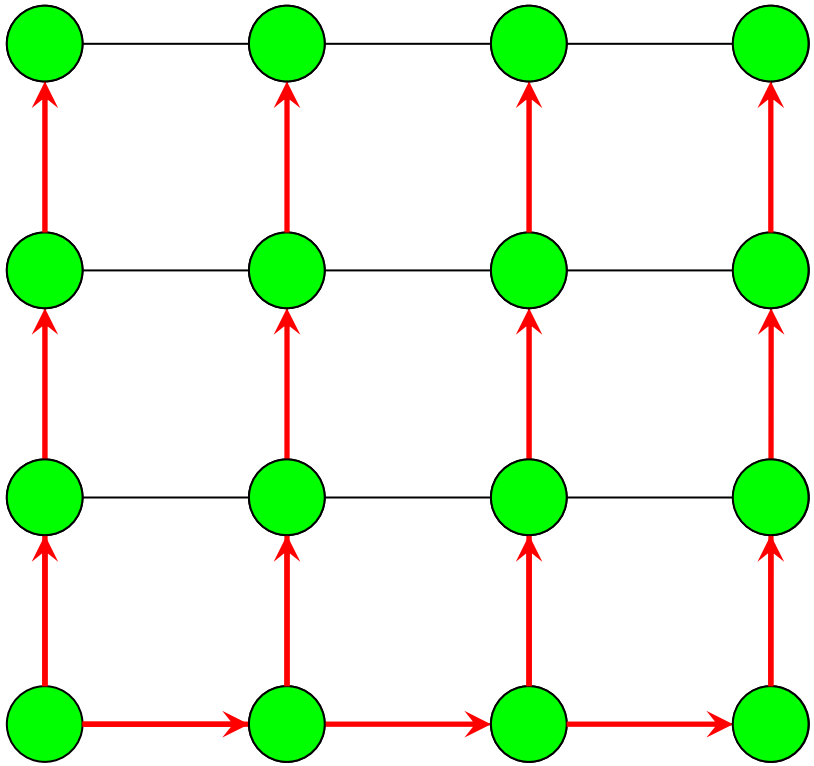




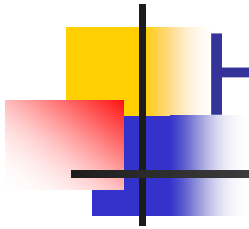
One-to-All Broadcast – Mesh

- 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

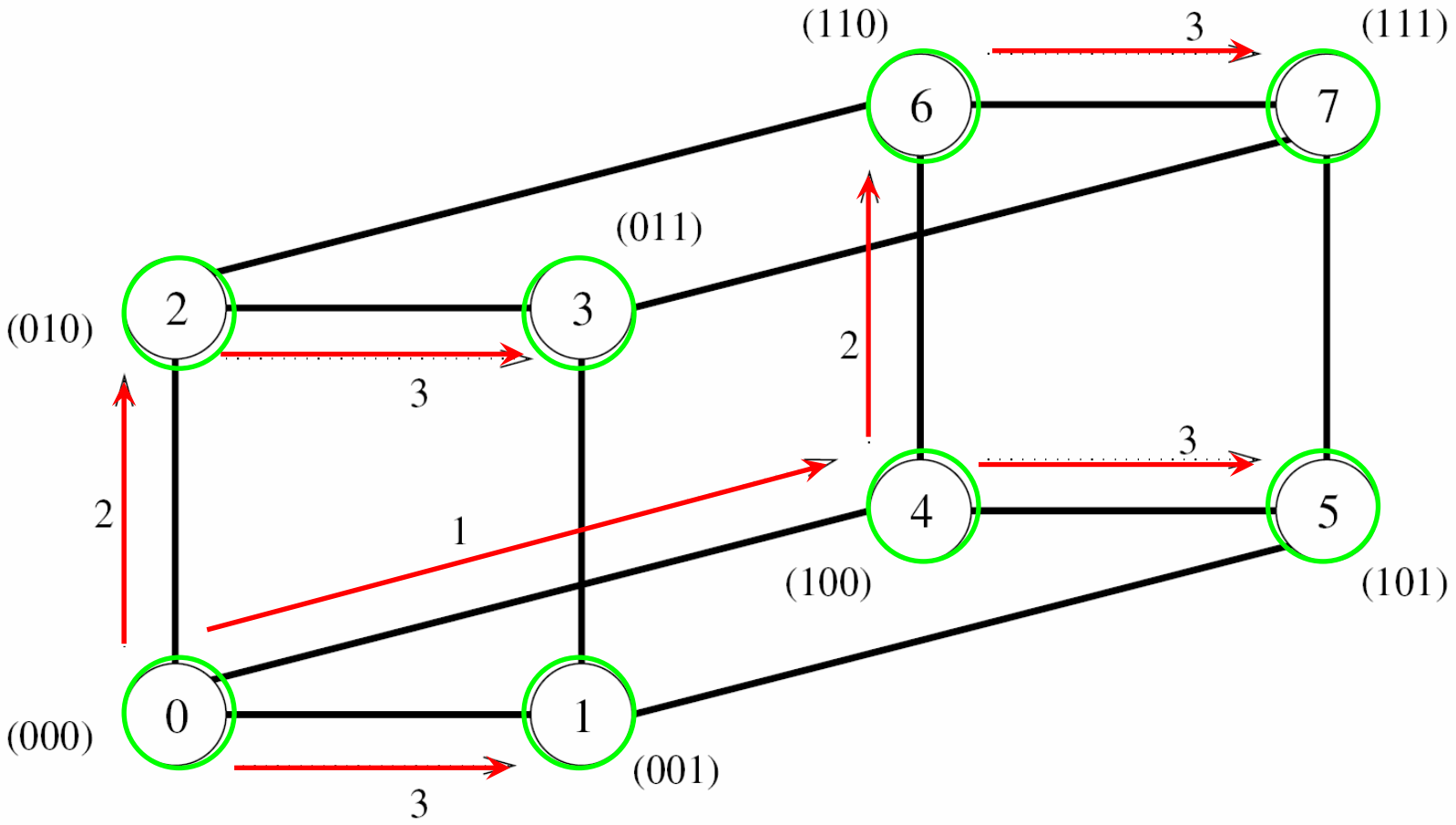


One-to-All Broadcast – Hypercube

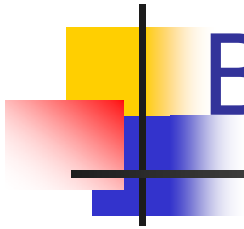


- Hypercube with 2^d nodes = d-dimensional mesh with 2 nodes in each direction.
- Similar algorithm in d steps.
- Also in $\log p$ steps.
- Reduction follows the same pattern.

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

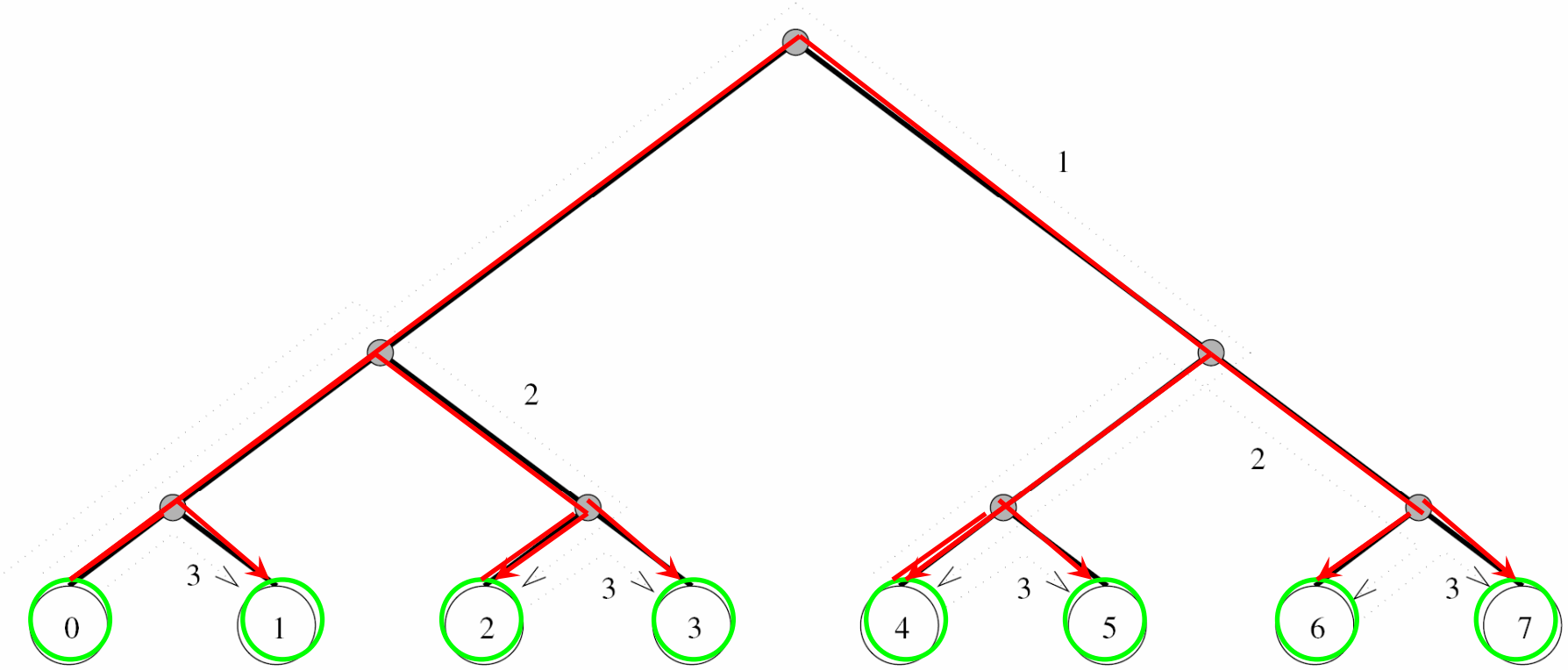
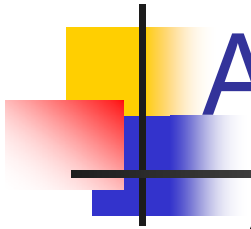


Figure 4.7 One-to-all broadcast on an eight-node tree.



Algorithms

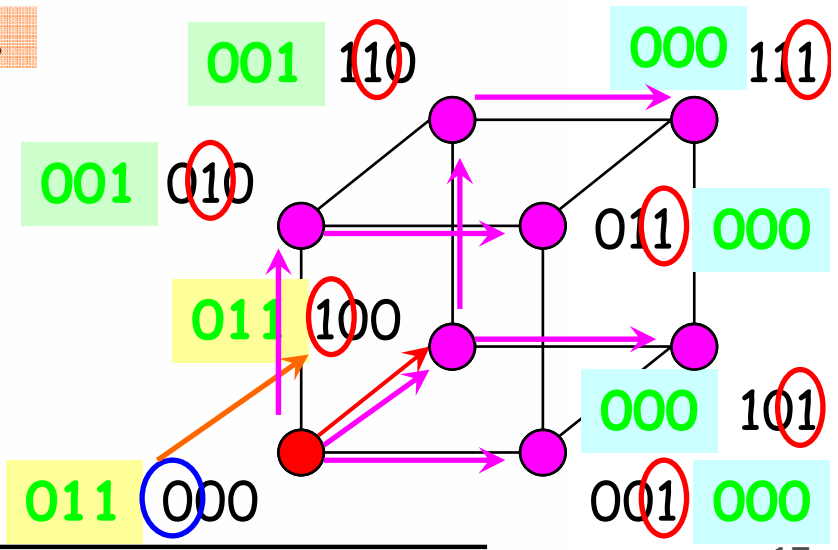
- So far we saw pictures.
- Not enough to implement.
- Precise description
  - to implement.
  - to analyze.
- Description for hypercube.
- Execute the following procedure on all the nodes.

Broadcast Algorithm

```
1.   procedure ONE_TO_ALL_BC(d, my_id, X)
2.   begin
3.       mask := 2^d − 1;                    /* Set all d bits of mask to 1 */
4.       for i := d − 1 downto 0 do          /* Outer loop */
5.           mask := mask XOR 2^i;           /* Set bit i of mask to 0 */
6.           if (my_id AND mask) = 0 then    /* If lower i bits of my_id are 0 */
7.               if (my_id AND 2^i) = 0 then
8.                   msg_destination := my_id XOR 2^i;
9.                   send X to msg_destination;
10.              else
11.                  msg_source := my_id XOR 2^i;
12.                  receive X from msg_source;
13.              endelse;
14.          endif;
15.      endfor;
16.  end ONE_TO_ALL_BC
```

Current dimension

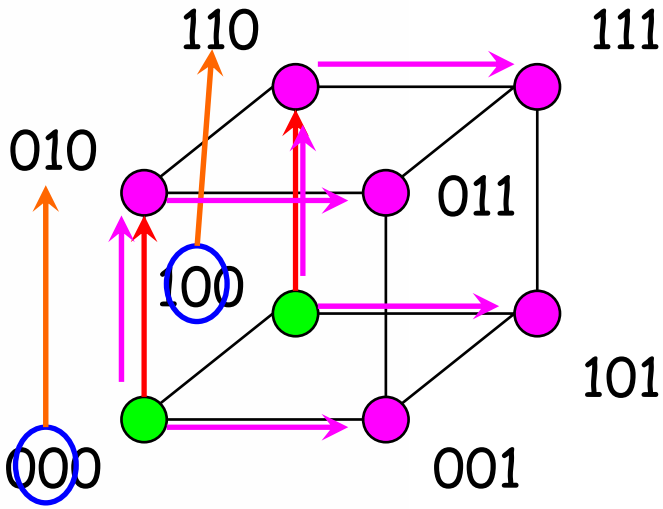
111 001 000 011



Broadcast Algorithm

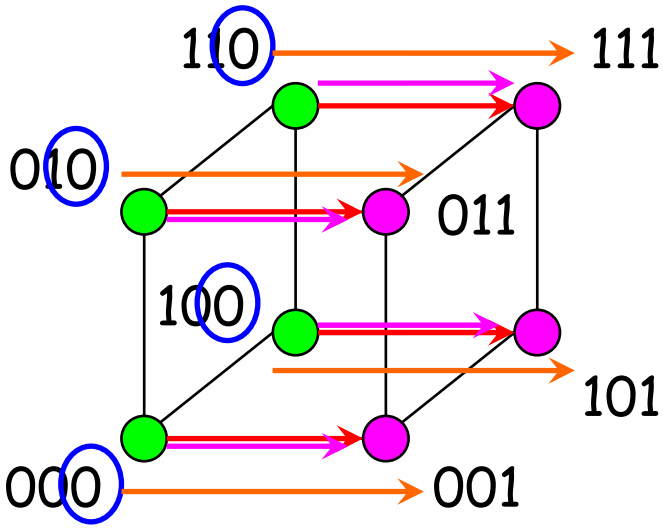
```
1.   procedure ONE_TO_ALL_BC(d, my_id, X)
2.   begin
3.       mask := 2^d − 1;                    /* Set all d bits of mask to 1 */
4.       for i := d − 1 downto 0 do          /* Outer loop */
5.           mask := mask XOR 2^i;           /* Set bit i of mask to 0 */
6.           if (my_id AND mask) = 0 then    /* If lower i bits of my_id are 0 */
7.               if (my_id AND 2^i) = 0 then
8.                   msg_destination := my_id XOR 2^i;
9.                   send X to msg_destination;
10.              else
11.                  msg_source := my_id XOR 2^i;
12.                  receive X from msg_source;
13.              endelse;
14.          endif;
15.      endfor;
16.  end ONE_TO_ALL_BC
```

001 000



Broadcast Algorithm

```
1.   procedure ONE_TO_ALL_BC(d, my_id, X)
2.   begin
3.       mask := 2^d − 1;                          /* Set all d bits of mask to 1 */
4.       for i := d − 1 downto 0 do                /* Outer loop */
5.           mask := mask XOR 2^i;                 /* Set bit i of mask to 0 */
6.           if (my_id AND mask) = 0 then          /* If lower i bits of my_id are 0 */
7.               if (my_id AND 2^i) = 0 then
8.                   msg_destination := my_id XOR 2^i;
9.                   send X to msg_destination;
10.              else
11.                  msg_source := my_id XOR 2^i;
12.                  receive X from msg_source;
13.              endelse;
14.          endif;
15.      endfor;
16.  end ONE_TO_ALL_BC
```



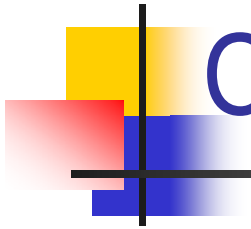
Algorithm For Any Source

```
1.   procedure GENERAL_ONE_TO_ALL_BC(d, my_id, source, X)
2.   begin
3.       my_virtual_id := my_id XOR source;
4.       mask := 2^d − 1;
5.       for i := d − 1 downto 0 do      /* Outer loop */
6.           mask := mask XOR 2^i;    /* Set bit i of mask to 0 */
7.           if (my_virtual_id AND mask) = 0 then
8.               if (my_virtual_id AND 2^i) = 0 then
9.                   virtual_dest := my_virtual_id XOR 2^i;
10.                  send X to (virtual_dest XOR source);
         /* Convert virtual_dest to the label of the physical destination */
11.              else
12.                  virtual_source := my_virtual_id XOR 2^i;
13.                  receive X from (virtual_source XOR source);
         /* Convert virtual_source to the label of the physical source */
14.              endelse;
15.      endfor;
16.  end GENERAL_ONE_TO_ALL_BC
```

Reduce Algorithm

```
1.   procedure ALL_TO_ONE_REDUCE(d, my_id, m, X, sum)
2.   begin
3.       for j := 0 to m − 1 do sum[j] := X[j];
4.       mask := 0;
5.       for i := 0 to d − 1 do
             /* Select nodes whose lower i bits are 0 */
6.           if (my_id AND mask) = 0 then
7.               if (my_id AND 2^i) ≠ 0 then
8.                   msg_destination := my_id XOR 2^i;
9.
10.
11.
12.                  receive X from msg_source;
13.                  for j := 0 to m − 1 do
14.                      sum[j] := sum[j] + X[j];
15.              endelse;
16.          mask := mask XOR 2^i;    /* Set bit i of mask to 1 */
17.      endfor;
18.  end ALL_TO_ONE_REDUCE
```

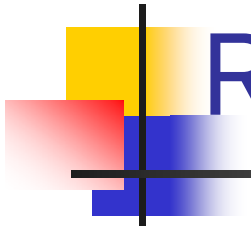
In a nutshell:
reverse the previous one.



Cost Analysis

p processes $\rightarrow \log p$ 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

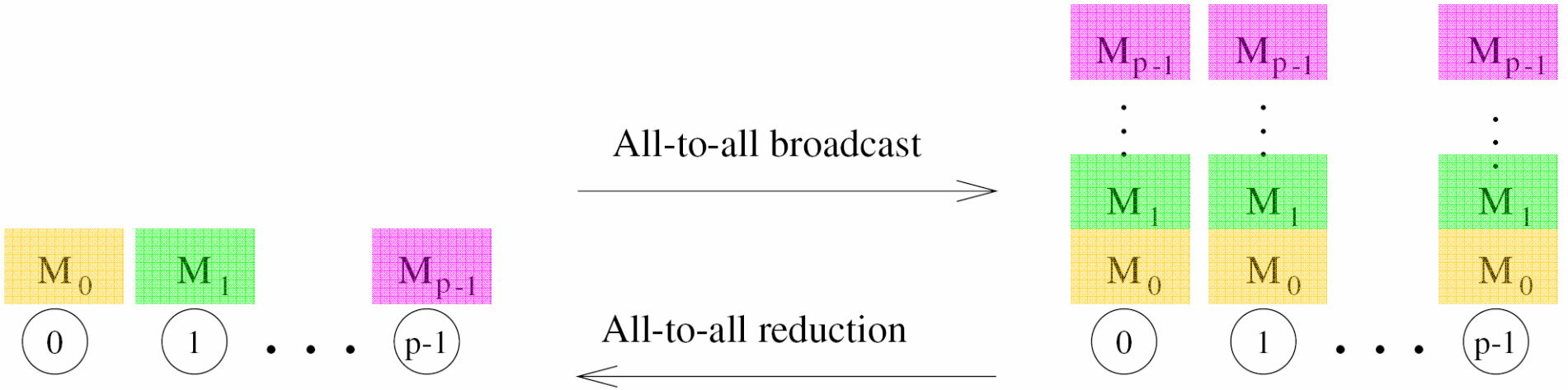
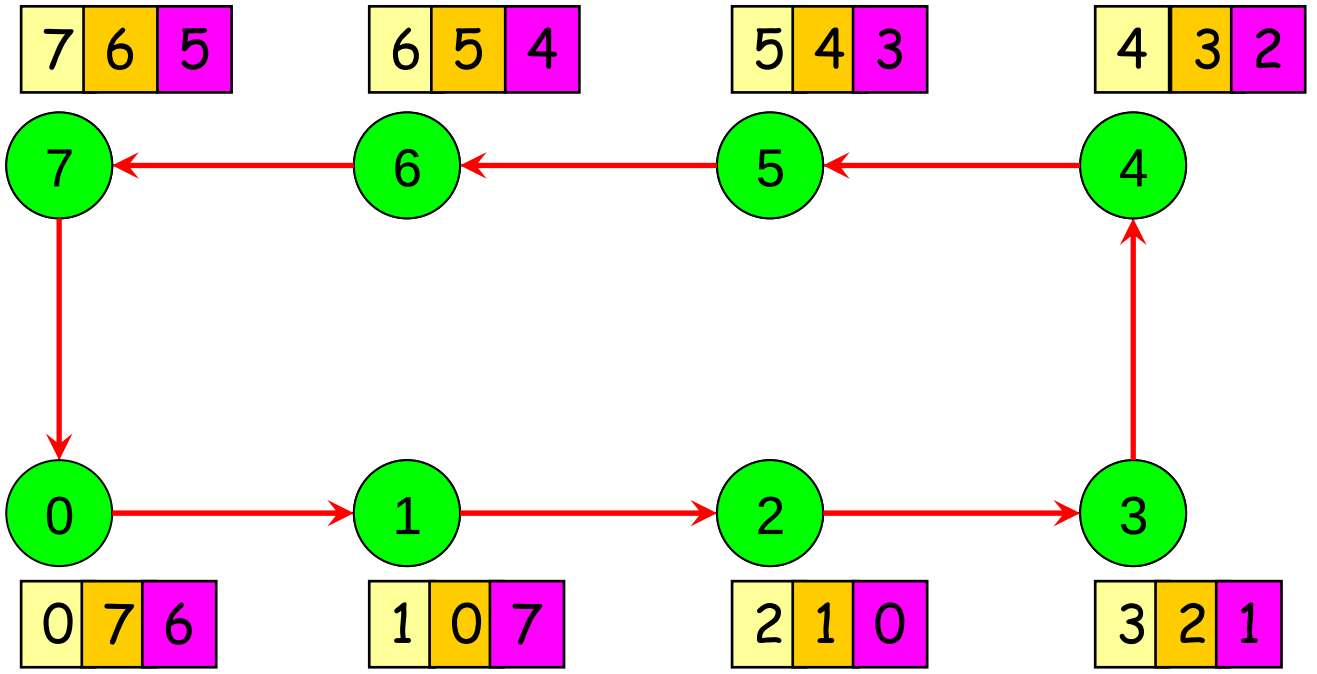


Figure 4.8 All-to-all broadcast and all-to-all reduction.

All-to-All Broadcast – Rings



etc…

All-to-All Broadcast Algorithm

```
1.   procedure ALL_TO_ALL_BC_RING(my_id, my_msg, p, result)
2.   begin
3.       left := (my_id − 1) mod p;
4.       right := (my_id + 1) mod p;
5.       result := my_msg;
6.       msg := result;
7.       for i := 1 to p − 1 do
8.           send msg to right;
9.           receive msg from left;
10.          result := result ∪ msg;
11.      endfor;
12.  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

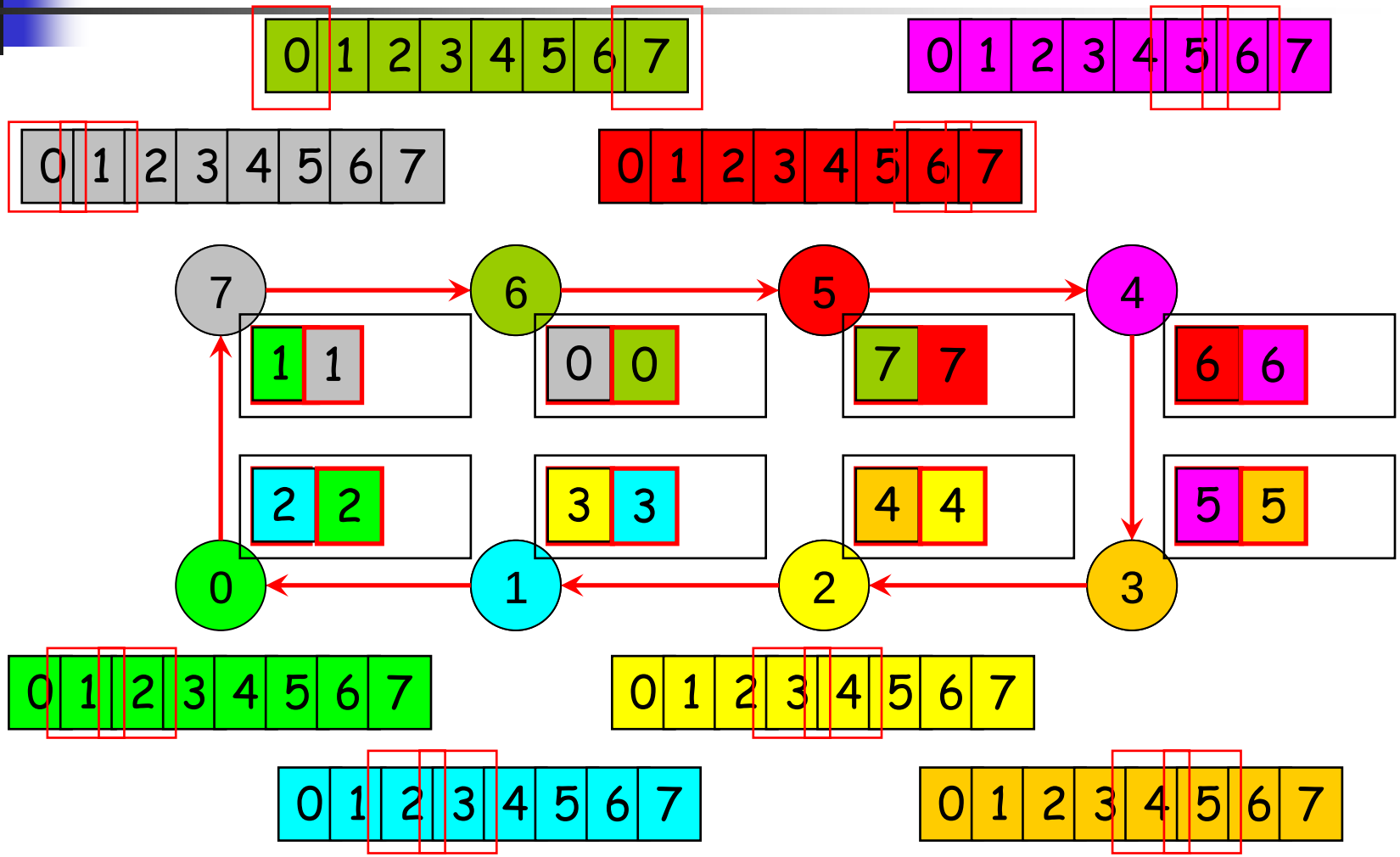
```
1.   procedure ALL_TO_ALL_RED_RING(my_id, my_msg, p, result)
2.   begin
3.       left := (my_id − 1) mod p;
4.       right := (my_id + 1) mod p;
5.       recv := 0;
6.       for i := 1 to p − 1 do
7.           j := (my_id + i) mod p;
8.           temp := msg[j] + recv;
9.           send temp to left;
10.          receive recv from right;
11.      endfor;
12.      result := msg[my_id] + recv;
13.  end ALL_TO_ALL_RED_RING
```

Accumulate and forward.

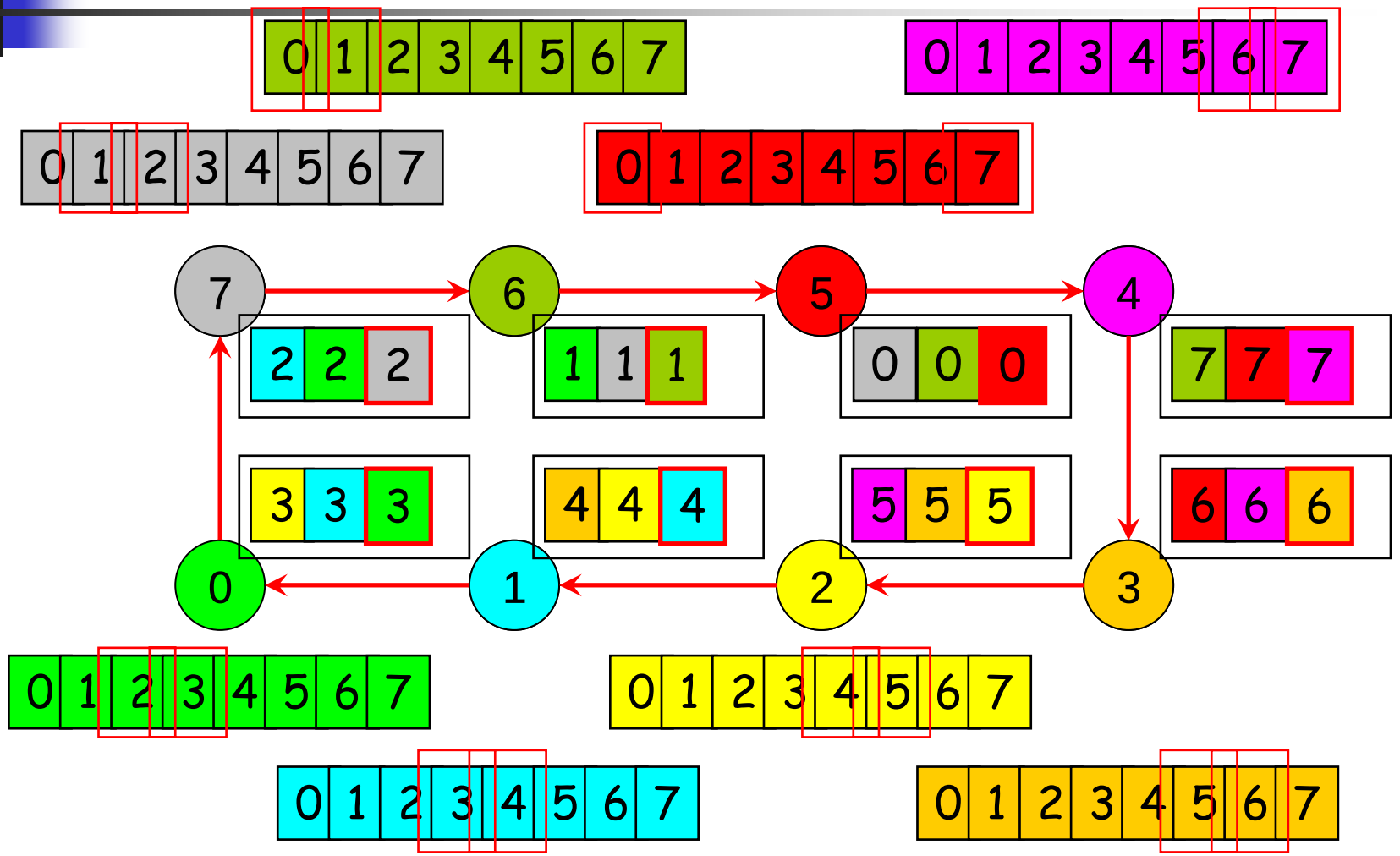
Last message for *my_id*.

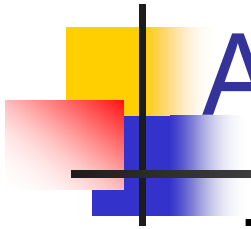
Algorithm 4.5 All-to-all reduction on a p-node ring.

All-to-All Reduce – Rings



All-to-All Reduce – Rings

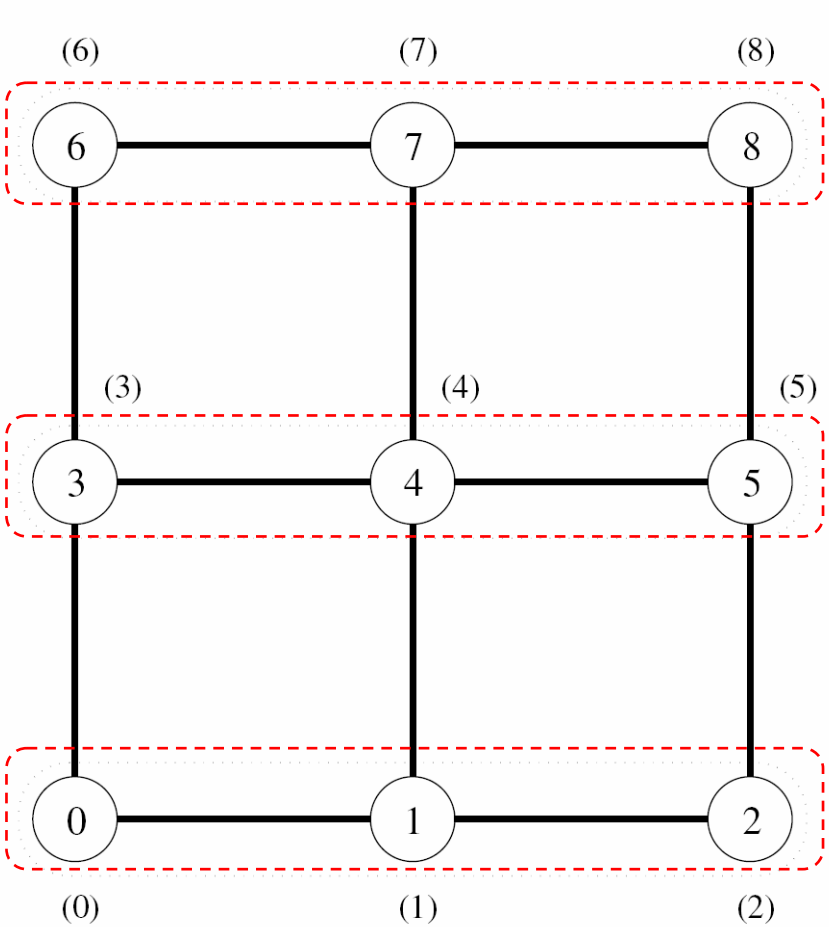


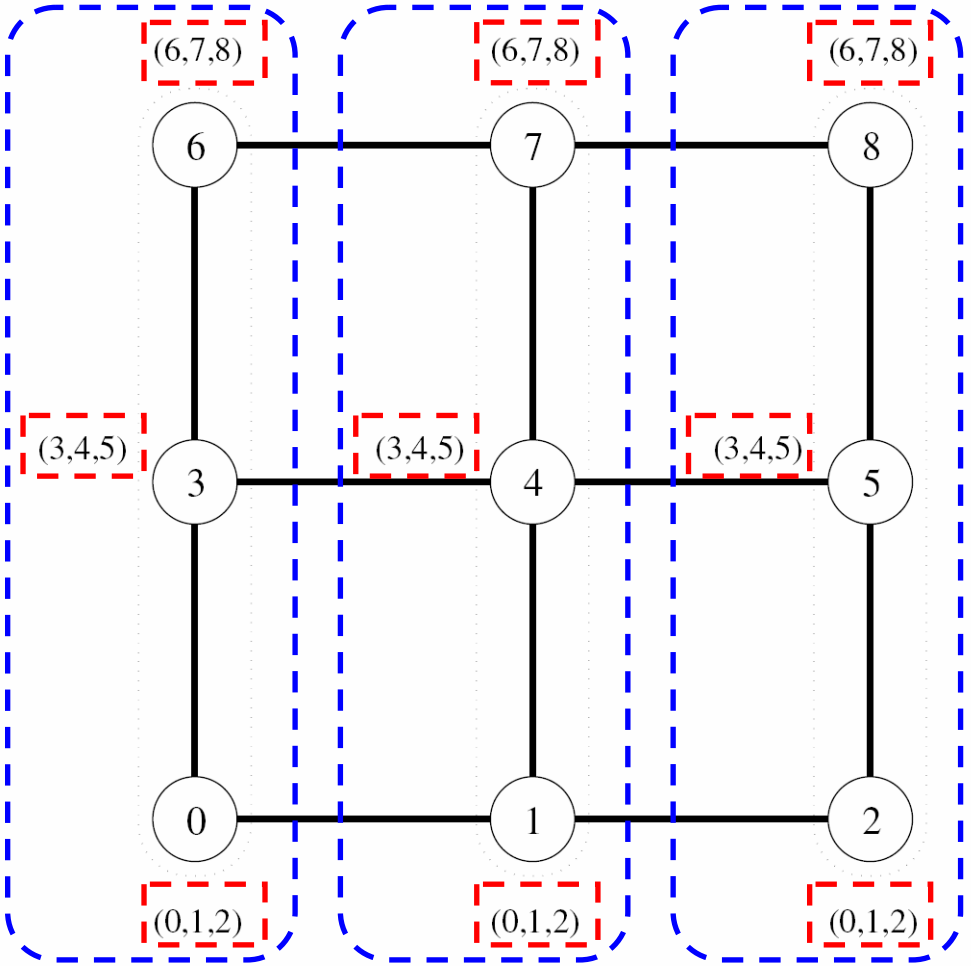


All-to-All Broadcast – Meshes

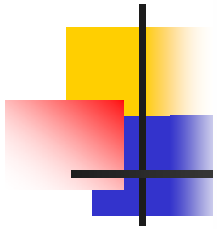
- Two phases:
  - All-to-all on rows – messages size m.
    - Collect sqrt(p) messages.
  - All-to-all on columns – messages size sqrt(p)*m.

All-to-All Broadcast – Meshes



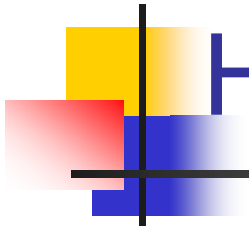


Algorithm



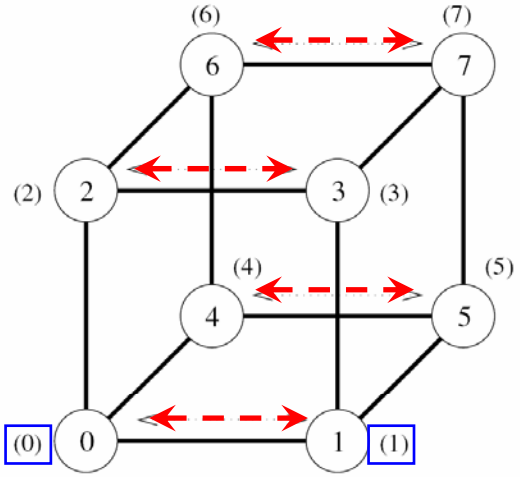
```
1.   procedure ALL_TO_ALL_BC_MESH(my_id, my_msg, p, result)
2.   begin

/* Communication along rows */
3.       left := my_id − (my_id mod √p) + (my_id − 1) mod √p;
4.       right := my_id − (my_id mod √p) + (my_id + 1) mod √p;
5.       result := my_msg;
6.       msg := result;
7.       for i := 1 to √p − 1 do
8.           send msg to right;
9.           receive msg from left;
10.          result := result ∪ msg;
11.      endfor;

/* Communication along columns */
12.      up := (my_id − √p) mod p;
13.      down := (my_id + √p) mod p;
14.      msg := result;
15.      for i := 1 to √p − 1 do
16.          send msg to down;
17.          receive msg from up;
18.          result := result ∪ msg;
19.      endfor;
20.  end ALL_TO_ALL_BC_MESH
```

All-to-All Broadcast - Hypercubes

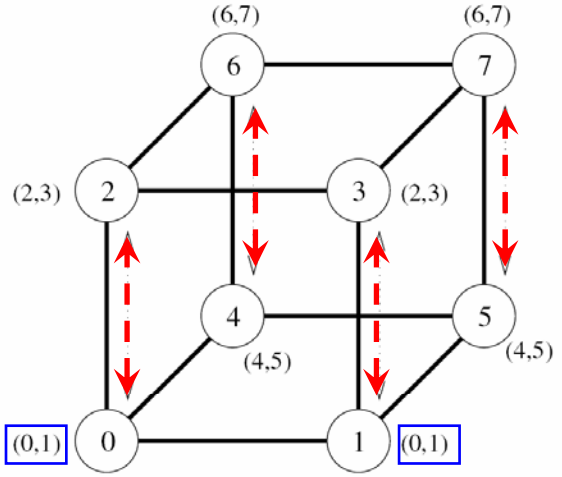


- Generalization of the mesh algorithm to $\log p$ dimensions.
- Message size doubles at every step.
- Number of steps: $\log p$.

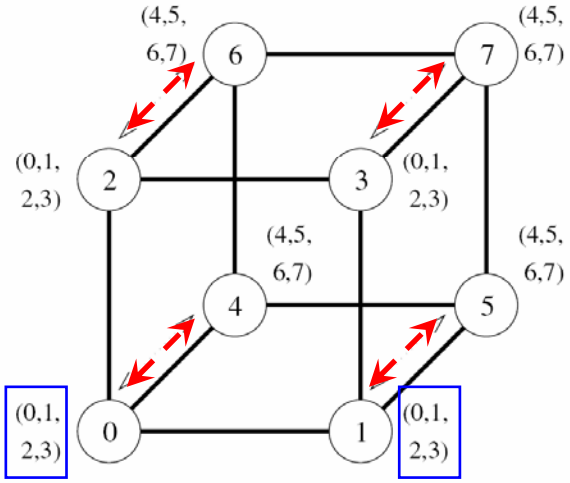
All-to-All Broadcast – Hypercubes



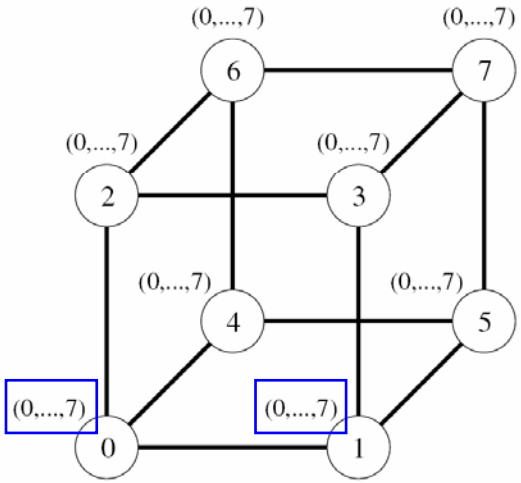
(a) Initial distribution of messages



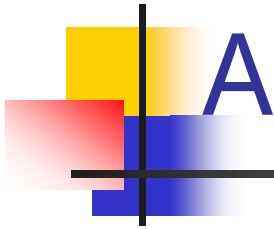
(b) Distribution before the second step



(c) Distribution before the third step



(d) Final distribution of messages



Algorithm

```
1.   procedure ALL_TO_ALL_BC_HCUBE(my_id, my_msg, d, result)
2.   begin
3.       result := my_msg;
4.       for i := 0 to d − 1 do
5.           partner := my_id XOR 2^i;
6.           send result to partner;
7.           receive msg from partner;
8.           result := result ∪ msg;
9.       endfor;
10.  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

```
1.   procedure ALL_TO_ALL_RED_HCUBE(my_id, msg, d, result)
2.   begin
3.       recloc := 0;
4.       for i := d − 1 to 0 do
5.           partner := my_id XOR 2^i;
6.           j := my_id AND 2^i;
7.           k := (my_id XOR 2^i) AND 2^i;
8.           senloc := recloc + k;
9.           recloc := recloc + j;
10.          send msg[senloc .. senloc + 2^i − 1] to partner;
11.          receive temp[0 .. 2^i − 1] from partner;
12.          for j := 0 to 2^i − 1 do
13.              msg[recloc + j] := msg[recloc + j] + temp[j];
14.          endfor;
15.      endfor;
16.      result := msg[my_id];
17.  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:
  - $$T = \sum_{i=1}^{\log p}(t_s + 2^{i-1}t_w m)$$

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

log*p* steps
message of size $2^{i-1}m$.

Dense to Sparser: Congestion

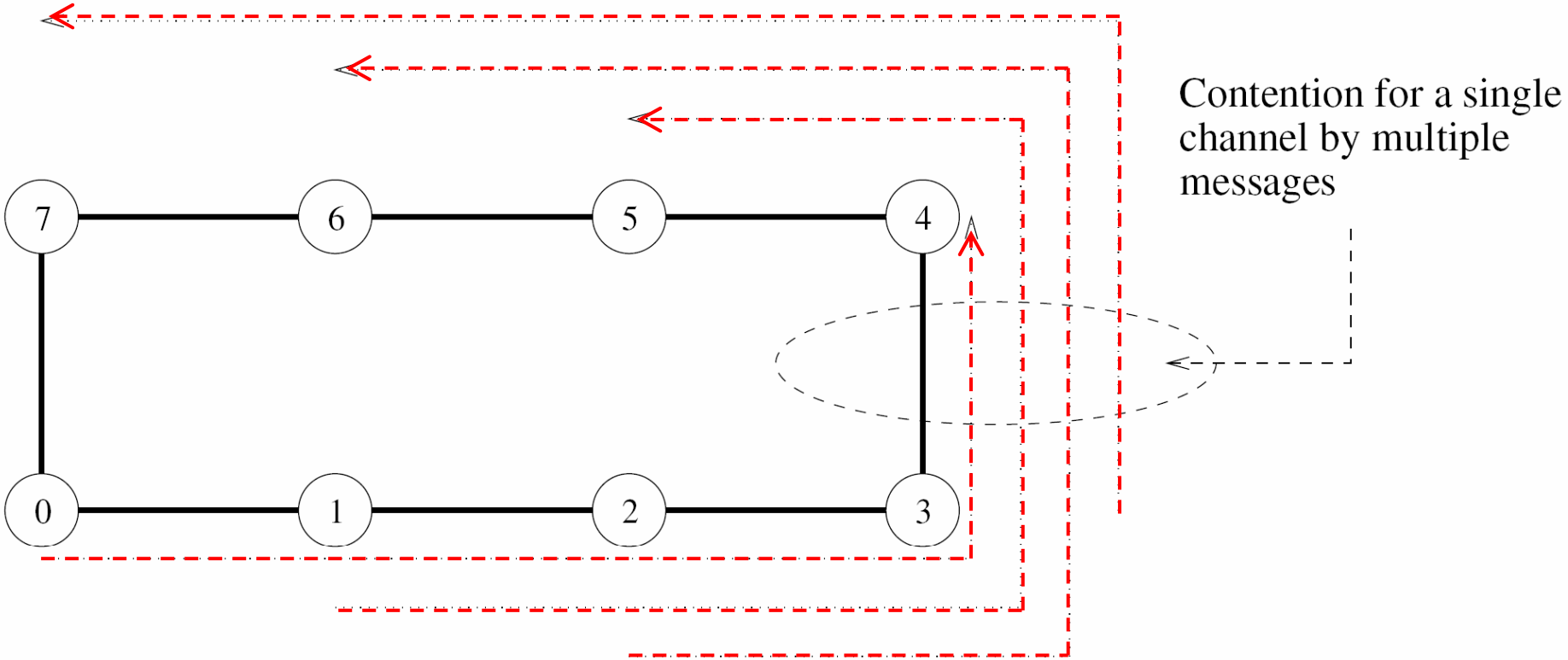
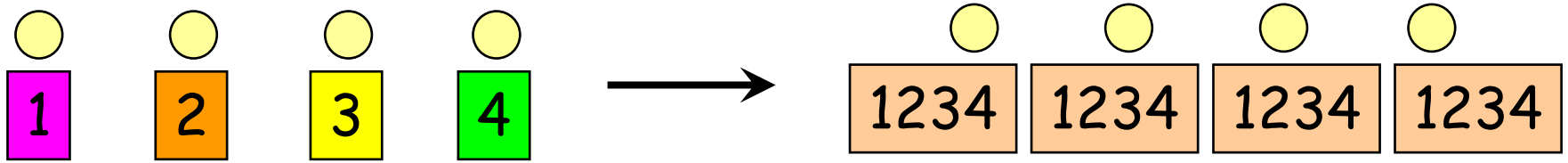
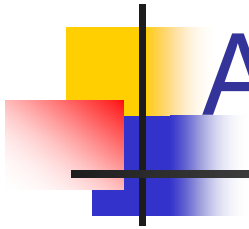


Figure 4.12 Contention for a channel when the communication step of Figure 4.11(c) for the hypercube is mapped onto a ring.

All-Reduce

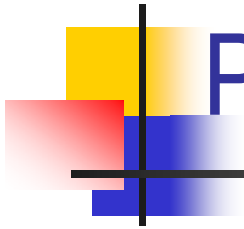
- Each node starts with a buffer of size *m*.
- The final result is the same combination of all buffers on every node.
- Same as all-to-one reduce + one-to-all broadcast.
- Different from all-to-all reduce.





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

```
1.   procedure PREFIX_SUMS_HCUBE(my_id, my_number, d, result)
2.   begin
3.       result := my_number;
4.       msg := result;
5.       for i := 0 to d − 1 do
6.           partner := my_id XOR 2^i;
7.           send msg to partner;
8.           receive number from partner;
9.           msg := msg + number;
10.          if (partner < my_id) then result := result + number;
11.      endfor;
12.  end PREFIX_SUMS_HCUBE
```

All-reduce

Prefix-sum

Algorithm 4.9 Prefix sums on a d-dimensional hypercube.

Prefix-Sum

Buffer = all-reduce sum