Performance, Scalability Messaging

Ashish Sharma

Final Project

1. Team Name
   Team Members (1 – 3)
   Feb 29
2. Project Proposal
   March 14
3. Project Update
   Apr 6
4. Final Presentation
   Apr 25-27

Quiz: Recap

Q1. Specs of the fastest machine

Q2. Hierarchy vs. Locality

- 3120000 cores; 54.9 PFLops
- Memory hierarchy refers to the hierarchy in speed and storage that is observed in accessible memory. Ranges from the fastest and smallest (L1 cache), to the slowest and largest (disk)
- Memory Locality refers to the proximity in space (contigious memory blocks) or time (access the same block across multiple cycles)

Q3. Spatial vs. Temporal Locality

Q4. R_{PEAK} of quad core, 2.25Ghz, 6ops/cycle

- Reduce Latency
  → Temporal Locality
- Increase Bandwidth
  → Spatial Locality

- $2.25 * 10^9 * 4 * 6 = 54$GFlops

Q5. 32KB L1 cache with a 1ns latency
4GBs via a bus with 100ns latency
bus is 1 word wide
Multiply 32x32

FLOPS = Computation / (T_{comp} + T_{comp})

64K Ops

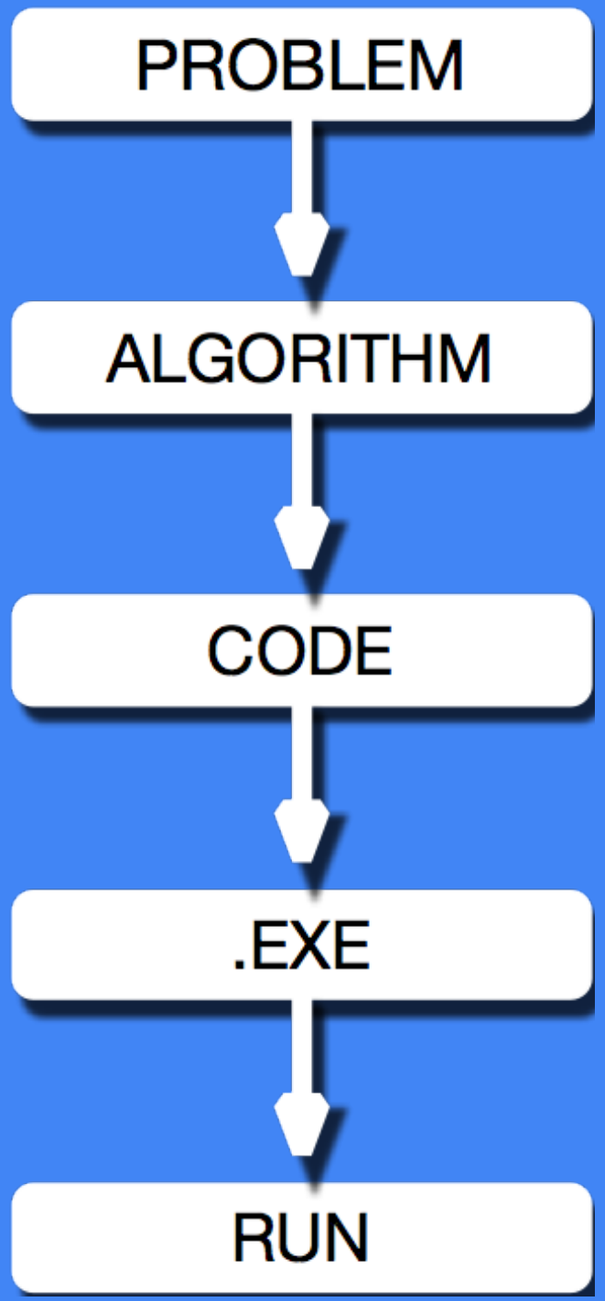
T_{net} → 100ns*1024 = ~100us
A, B → 100us each

T_{comp} → 64Kops@6 ops/cycle
~11K cycles
→ 11K * 2.25 * 10^9 = ~25us

FLOPS: 64K/(100+100+25)us = ~280MFlops

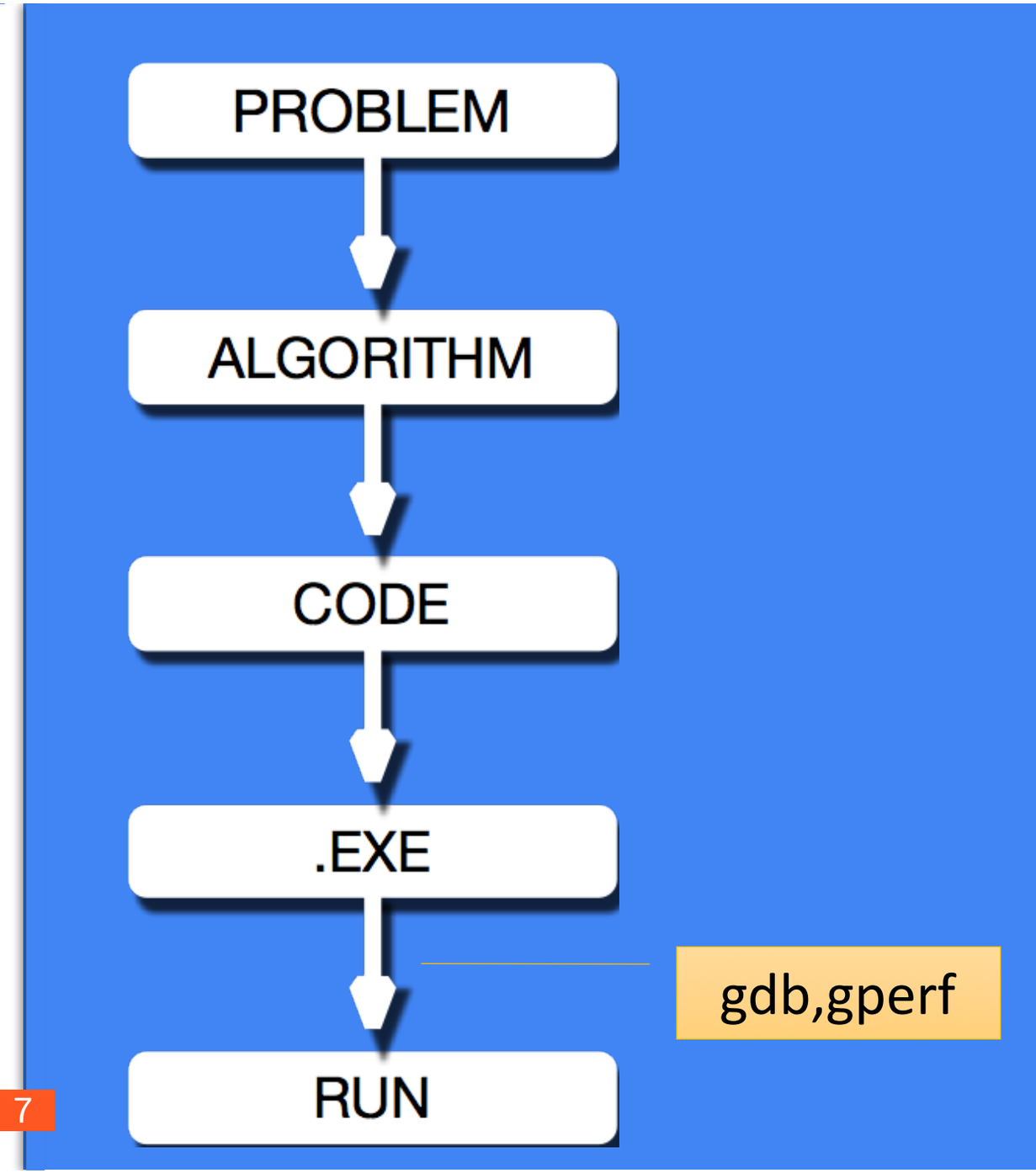
Goal?

Learn about Code
Performance and Scalability



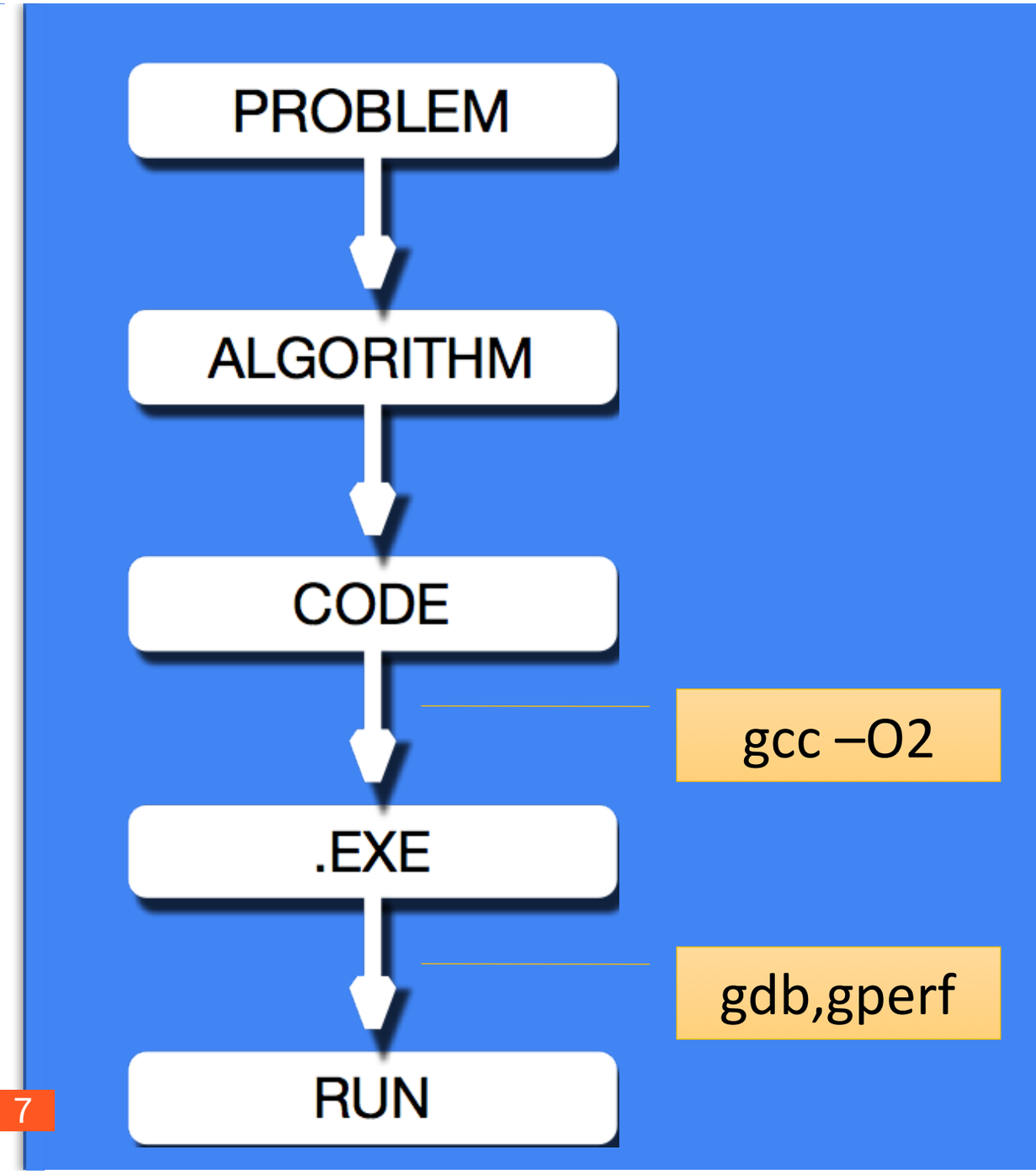
Goal?

Learn about Code
Performance and Scalability



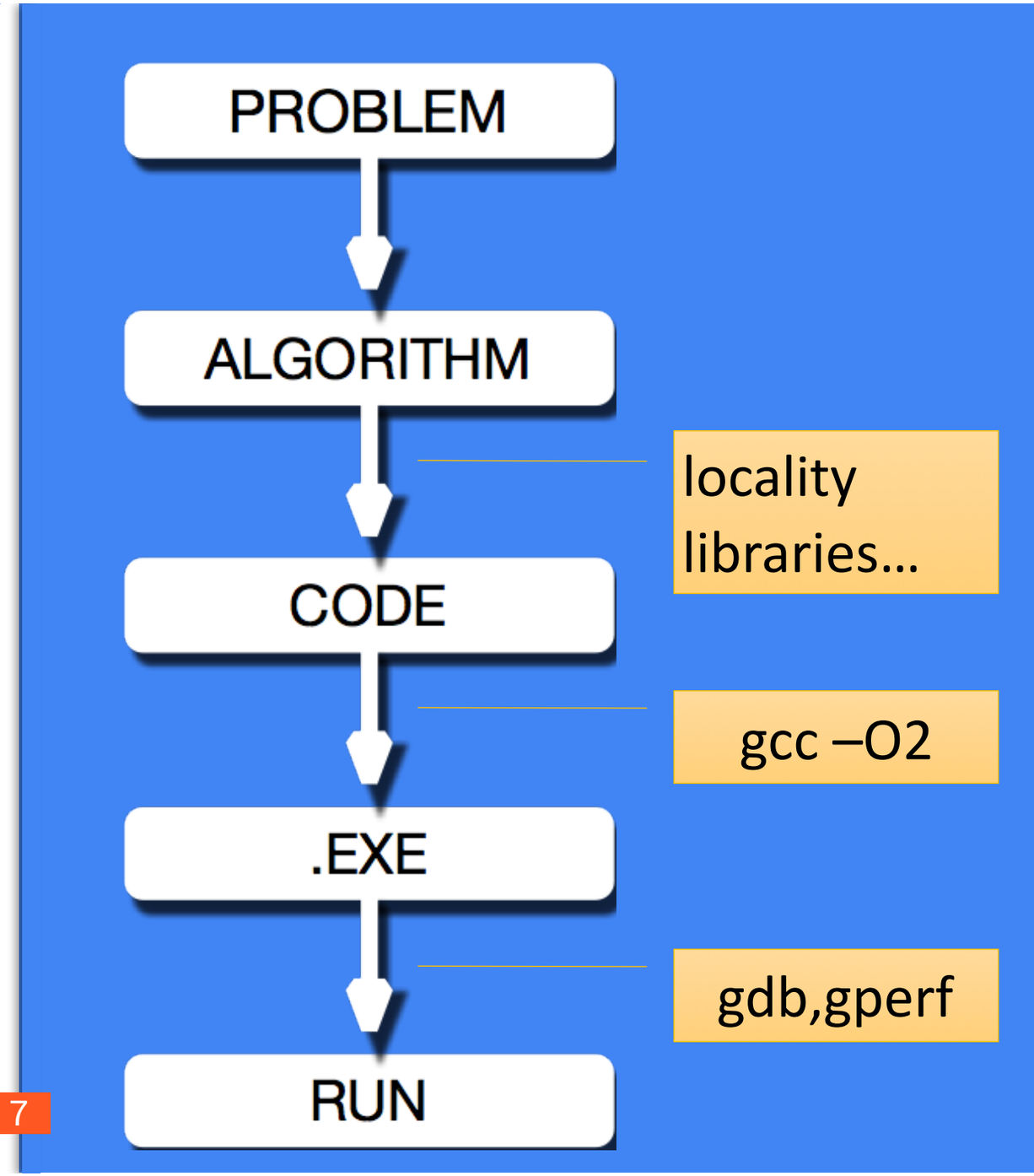
Goal?

Learn about Code
Performance and Scalability



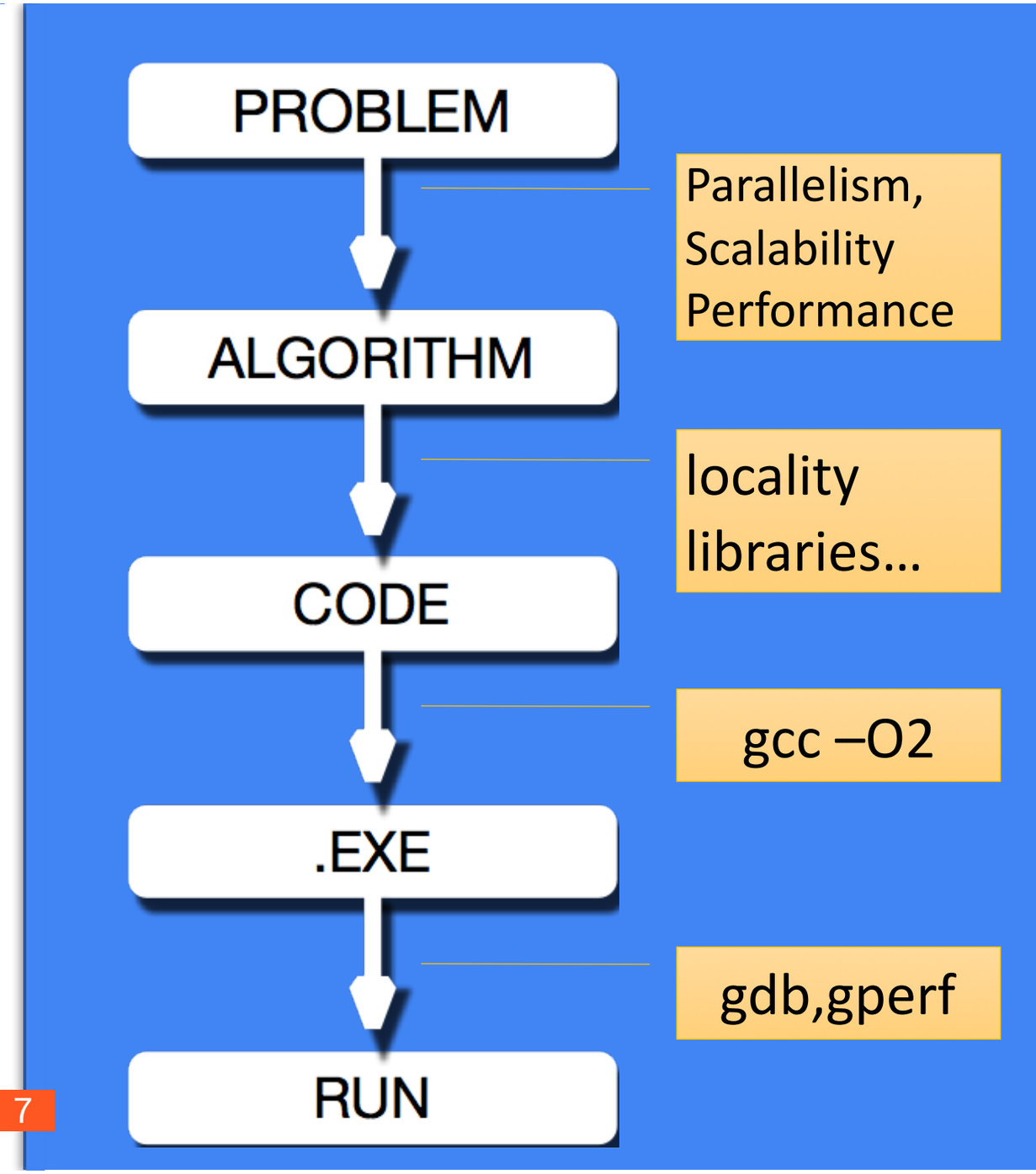
Goal?

Learn about Code
Performance and Scalability



Goal?

Learn about Code
Performance and Scalability



Step 1: Choosing an algorithm for PROBLEM

Algorithm → Biggest impact on efficiency and scalability

Step 1: Choosing an algorithm for PROBLEM

Algorithm → Biggest impact on efficiency and scalability

- It may not be intuitively obvious which algorithm is best.

Step 1: Choosing an algorithm for PROBLEM

Algorithm → Biggest impact on efficiency and scalability

- It may not be intuitively obvious which algorithm is best.
  - A method that is highly efficient on one or a few processors may not be optimal for a large-scale parallel system.

Step 1: Choosing an algorithm for PROBLEM

Algorithm → Biggest impact on efficiency and scalability

- It may not be intuitively obvious which algorithm is best.
  - A method that is highly efficient on one or a few processors may not be optimal for a large-scale parallel system.
  - REASON: Code = Σ(sub-algorithms, sub-tasks)

Step 1: Choosing an algorithm for PROBLEM

Algorithm → Biggest impact on efficiency and scalability

- It may not be intuitively obvious which algorithm is best.
  - A method that is highly efficient on one or a few processors may not be optimal for a large-scale parallel system.
  - REASON: Code = Σ(sub-algorithms, sub-tasks)
  - And these sub-algorithms/approaches may not have the best scalability for your problem, at the scales you want to work in.

Step 1: Choosing an algorithm for PROBLEM

Algorithm → Biggest impact on efficiency and scalability

- It may not be intuitively obvious which algorithm is best.
  - A method that is highly efficient on one or a few processors may not be optimal for a large-scale parallel system.
  - REASON: Code = Σ(sub-algorithms, sub-tasks)
  - And these sub-algorithms/approaches may not have the best scalability for your problem, at the scales you want to work in.

SOLUTION: Start Early. Model Performance

So what is a scalable and efficient algorithm

So what is a scalable and efficient algorithm

1. Depends on the characteristics of your platform

So what is a scalable and efficient algorithm

1. Depends on the characteristics of your platform
2. Can decompose work into (sort-of) independent sub-tasks

So what is a scalable and efficient algorithm

1. Depends on the characteristics of your platform
2. Can decompose work into (sort-of) independent sub-tasks
3. Can compute >> Communicate. Why??
   Memory Hierarchy

So what is a scalable and efficient algorithm

1. Depends on the characteristics of your platform
2. Can decompose work into (sort-of) independent sub-tasks
3. Can compute >> Communicate. Why??
   Memory Hierarchy
4. Can minimize communication between sub-tasks

So what is a scalable and efficient algorithm

1. Depends on the characteristics of your platform
2. Can decompose work into (sort-of) independent sub-tasks
3. Can compute >> Communicate. Why??
   Memory Hierarchy
4. Can minimize communication between sub-tasks
5. Minimal global communication. Why??

So what is a scalable and efficient algorithm

1. Depends on the characteristics of your platform
2. Can decompose work into (sort-of) independent sub-tasks
3. Can compute >> Communicate. Why??
   Memory Hierarchy
4. Can minimize communication between sub-tasks
5. Minimal global communication. Why??
6. Initiate as many tasks as possible

SCALABILITY

SCALABILITY

- Common metric for measuring scalability is SPEEDUP

SCALABILITY

- Common metric for measuring scalability is SPEEDUP
- How much faster is your serial code vs. your parallel code

SCALABILITY

- Common metric for measuring scalability is SPEEDUP
- How much faster is your serial code vs. your parallel code
  - Ts → Time to complete serial version of the problem

SCALABILITY

- Common metric for measuring scalability is SPEEDUP
- How much faster is your serial code vs. your parallel code
  - Ts → Time to complete serial version of the problem
  - Tp → Time to complete parallel version of the problem

SCALABILITY

- Common metric for measuring scalability is SPEEDUP
- How much faster is your serial code vs. your parallel code
  - Ts → Time to complete serial version of the problem
  - Tp → Time to complete parallel version of the problem
  - Speedup = Ts / Tp

SCALABILITY

- Common metric for measuring scalability is SPEEDUP
- How much faster is your serial code vs. your parallel code
  - Ts → Time to complete serial version of the problem
  - Tp → Time to complete parallel version of the problem
  - Speedup = Ts / Tp

SCALABILITY

- Common metric for measuring scalability is SPEEDUP
- How much faster is your serial code vs. your parallel code
  - Ts → Time to complete serial version of the problem
  - Tp → Time to complete parallel version of the problem
  - Speedup = Ts / Tp

Say 10 processors, finish in 1/10th the time. Speedup = 10 → LINEAR

SCALABILITY

- Common metric for measuring scalability is SPEEDUP
- How much faster is your serial code vs. your parallel code
  - Ts → Time to complete serial version of the problem
  - Tp → Time to complete parallel version of the problem
  - Speedup = Ts / Tp

Say 10 processors, finish in 1/10th the time. Speedup = 10 → LINEAR

SUPERLINEAR → Tp > 1/10

Why Parallelize?

Why Parallelize?

- One reason → DO IT FASTER

Why Parallelize?

- One reason → DO IT FASTER
- Remember, Scalability is a measure of Speedup

Why Parallelize?

- One reason → DO IT FASTER
- Remember, Scalability is a measure of Speedup
- One way to define scalability, using speedup is as follows:

Why Parallelize?

- One reason → DO IT FASTER
- Remember, Scalability is a measure of Speedup
- One way to define scalability, using speedup is as follows:

  Fix the problem size and Increase number of processors

Why Parallelize?

- One reason → DO IT FASTER
- Remember, Scalability is a measure of Speedup
- One way to define scalability, using speedup is as follows:

Fix the problem size and Increase number of processors

STRONG SCALING

Why Parallelize?

- One reason → DO IT FASTER
- Remember, Scalability is a measure of Speedup
- One way to define scalability, using speedup is as follows:

Fix the problem size and Increase number of processors

STRONG SCALING

Problem size is 123456789.

Why Parallelize?

- One reason → DO IT FASTER
- Remember, Scalability is a measure of Speedup
- One way to define scalability, using speedup is as follows:

Fix the problem size and Increase number of processors

STRONG SCALING

Problem size is 123456789.

Takes 100s(1p) →

Why Parallelize?

- One reason → DO IT FASTER
- Remember, Scalability is a measure of Speedup
- One way to define scalability, using speedup is as follows:

Fix the problem size and Increase number of processors

STRONG SCALING

Problem size is 123456789.

Takes 100s(1p) →

10s(10p) →

Why Parallelize?

- One reason → DO IT FASTER
- Remember, Scalability is a measure of Speedup
- One way to define scalability, using speedup is as follows:

Fix the problem size and Increase number of processors

STRONG SCALING

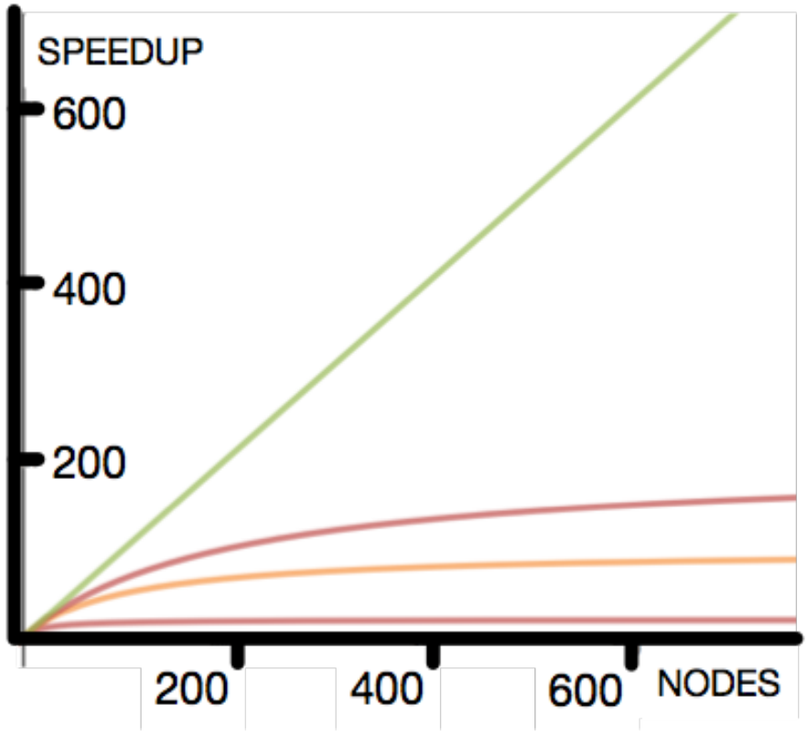
Problem size is 123456789.

Takes 100s(1p) →

10s(10p) →

1s(100p)

Amdahl's Law



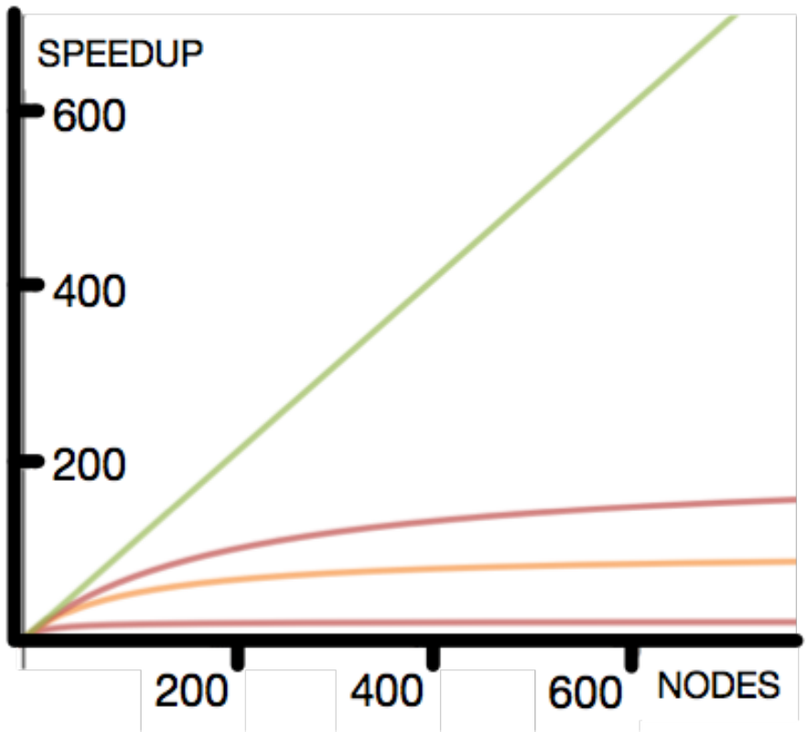
https://goo.gl/cpOaoc

Amdahl's Law

$$S = (T_S / T_P)$$

$$S = \frac{Ts}{(1-P)*Ts+P*{}^{Ts}/_{N}}$$

$$S = \frac{1}{(1-P) + P/N}$$



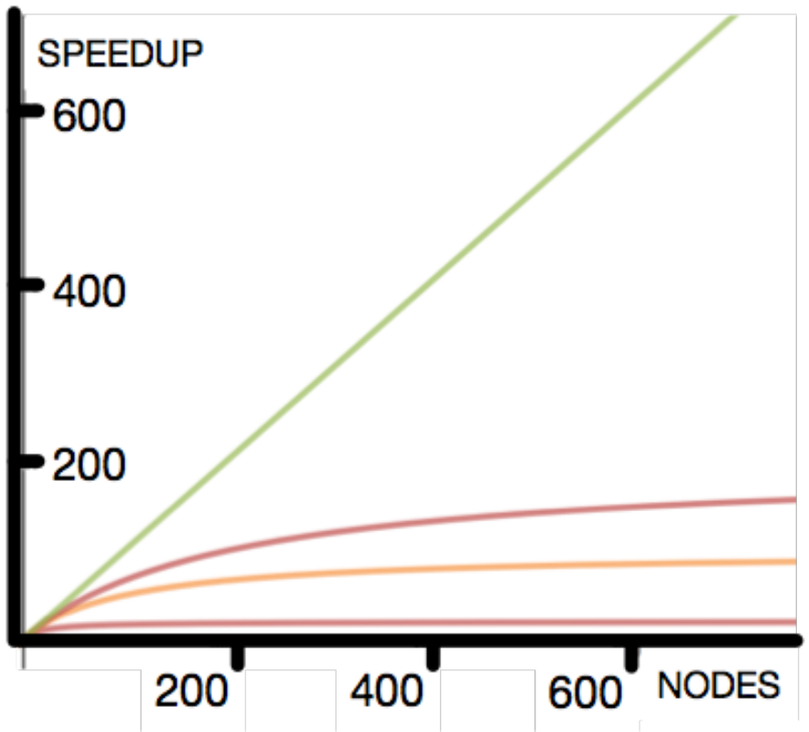
https://goo.gl/cpOaoc

Amdahl's Law

$$S = (T_S / T_P)$$

$$S = \frac{Ts}{(1-P)*Ts+P*{}^{Ts}/_{N}}$$

$$S = \frac{1}{(1-P) + P/N}$$



Adding more cores does not always work.

Unless your code is 100% efficient at using multiple cores, you will receive less and less of a benefit by adding more cores

https://goo.gl/cpOaoc

Faster is hard. So let's do LARGER

- Scalability does not have to mean faster. It can mean → process larger

- Aka Gustafson's Law → *"an observation that Amdahl's assumptions don't match the way people use parallel processors. People scale their problems to match the power available, in contrast to Amdahl's assumption that the problem is always the same no matter how capable the computer"*

- Called **Weak Scaling**

$$S = (1 - P) + P * N$$

Suggested Readings

"Estimating CPU Performance using Amdahl's Law"
https://goo.gl/Dg0Mc9

"Amdahl's Law, Gustafson's Trend, and the Performance Limits of Parallel Applications"
https://goo.gl/mBcvR9

https://goo.gl/4P28Xj

(5mins)

What does it mean to communicate?

- Not a shouting match (~~All to All at the Same Time~~)
- More like some nodes to some other nodes
  - Node → Processing Core
  - Wire → Communication Channel
- Cost of sending a single message = $T_{setup} + T_{transmission}$

→ $T_s + T_w$ x Size of data

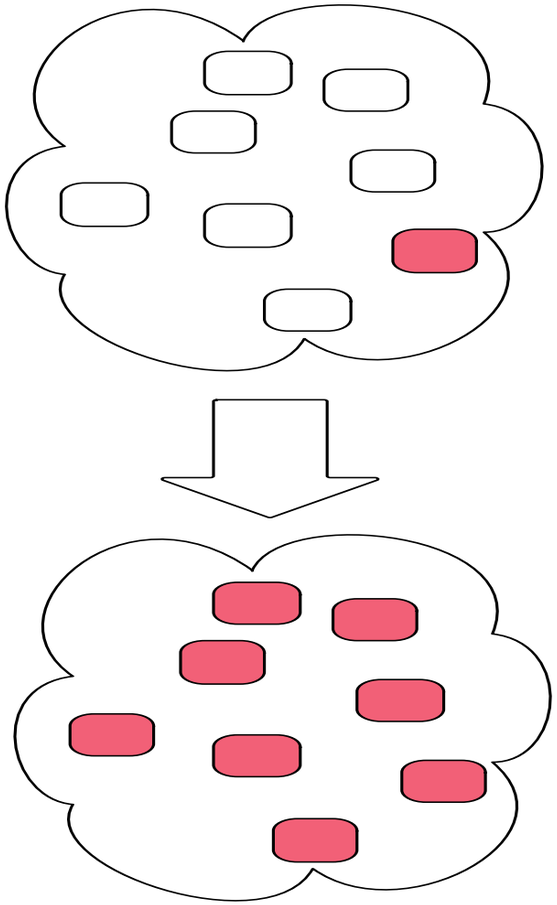
→ $T_s + T_w$ x m

Total Cost = ($T_s + T_w$ x m) x number of messages

Strategies to communicate

Objective: One node will send to All other nodes

Commonly called One-to-All Broadcast

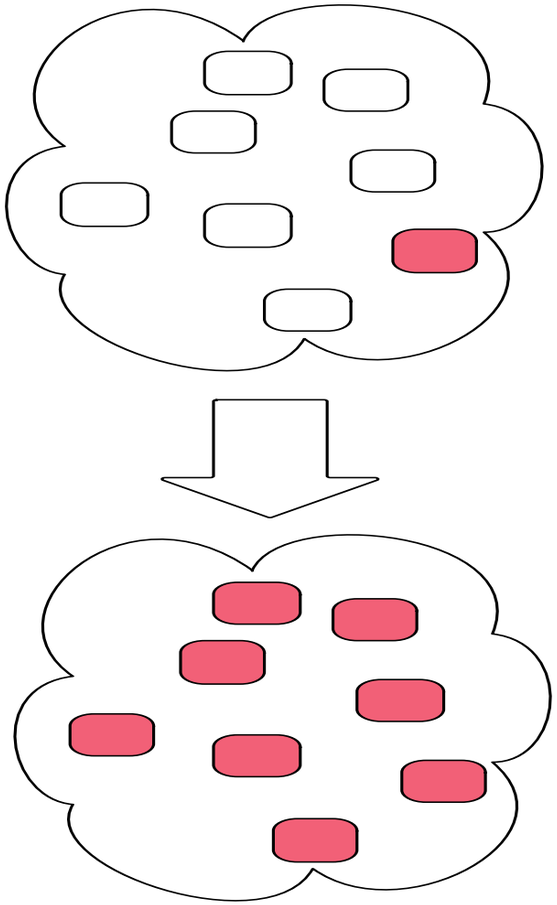


Strategies to communicate

Objective: One node will send to All other nodes

Commonly called One-to-All Broadcast

Simple Strategy: One node sends to each of the other (p-1) nodes



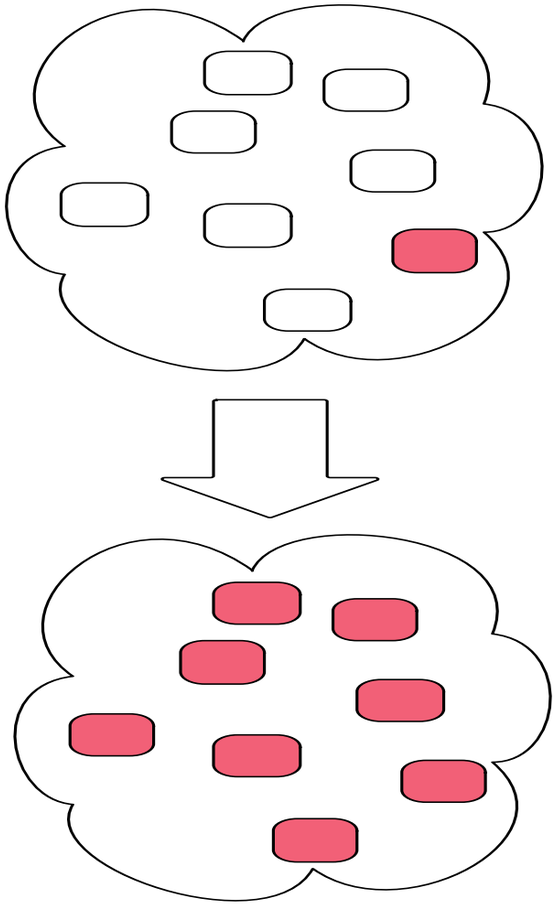
Strategies to communicate

Objective: One node will send to All other nodes

Commonly called One-to-All Broadcast

Simple Strategy: One node sends to each of the other (p-1) nodes

Why is this a bad idea?



Strategies to communicate

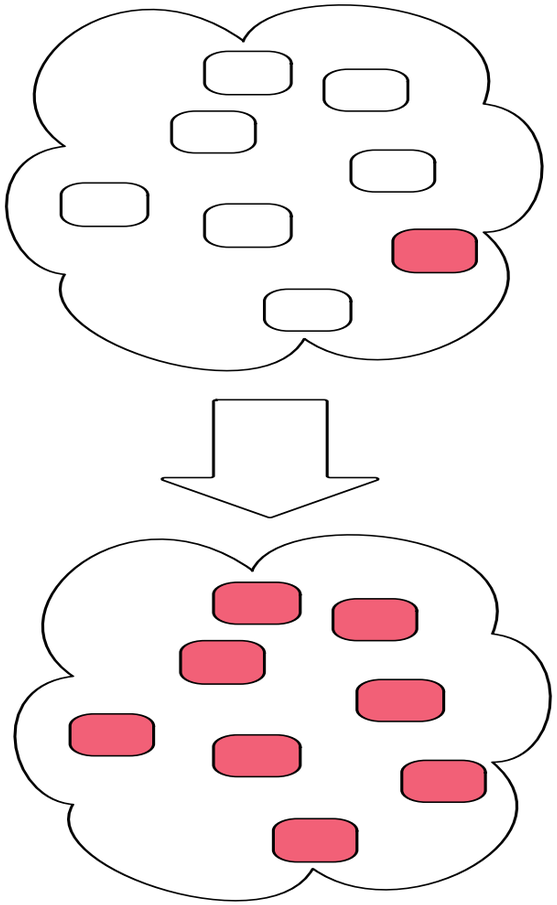
Objective: One node will send to All other nodes

Commonly called One-to-All Broadcast

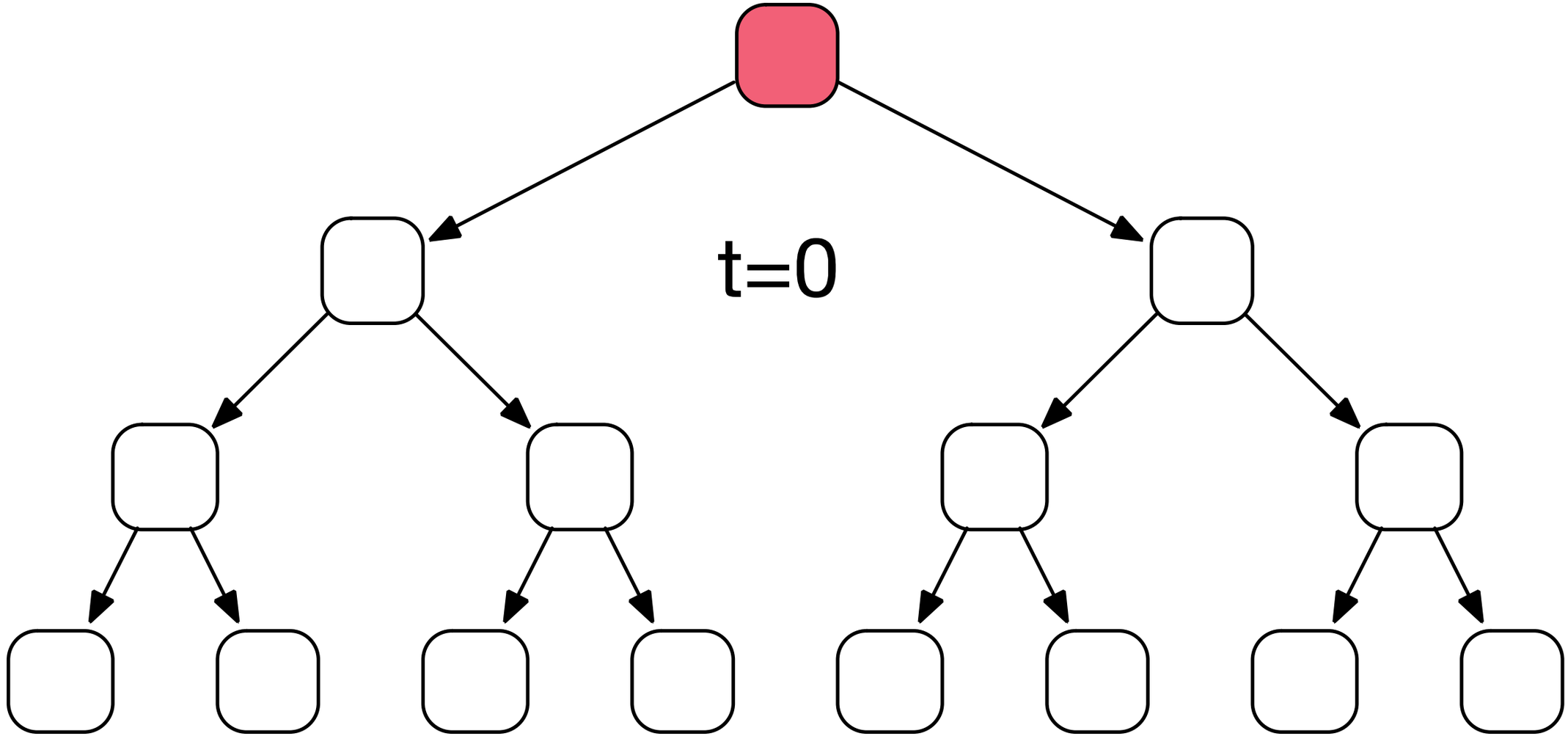
Simple Strategy: One node sends to each of the other (p-1) nodes

Why is this a bad idea?

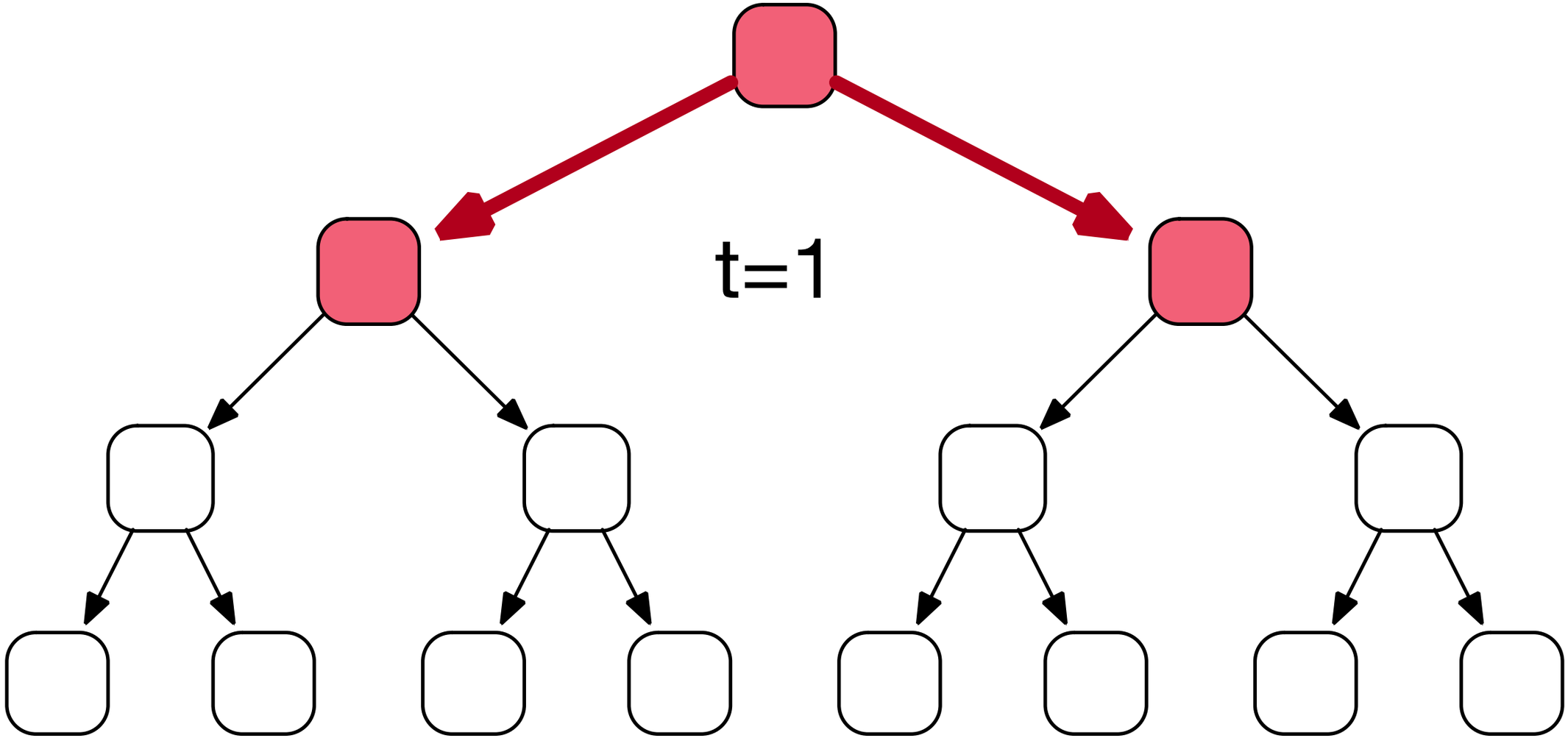
- You are using a fraction of the network
  i.e., you are only using one node at a time



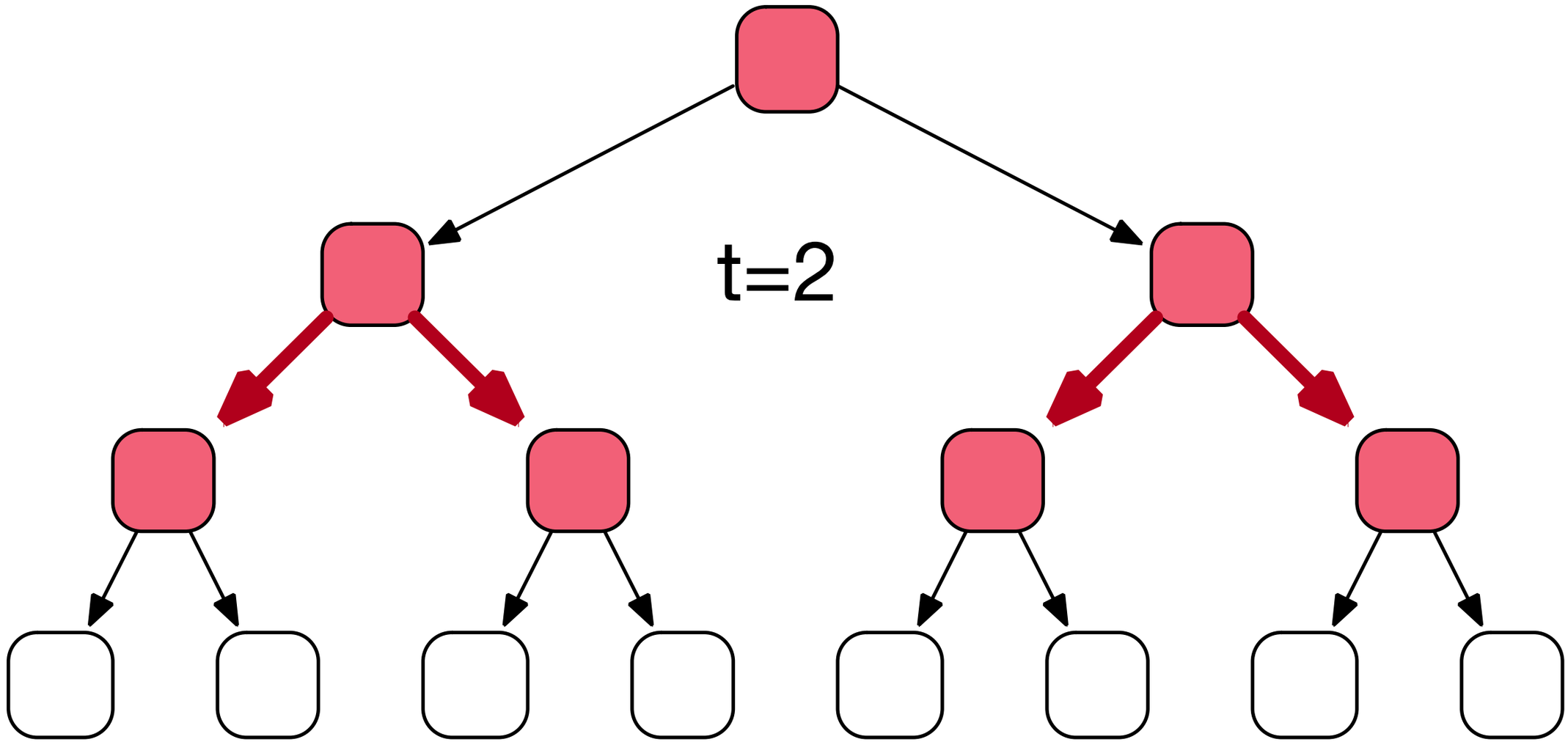
Better Strategy → Recursive Doubling



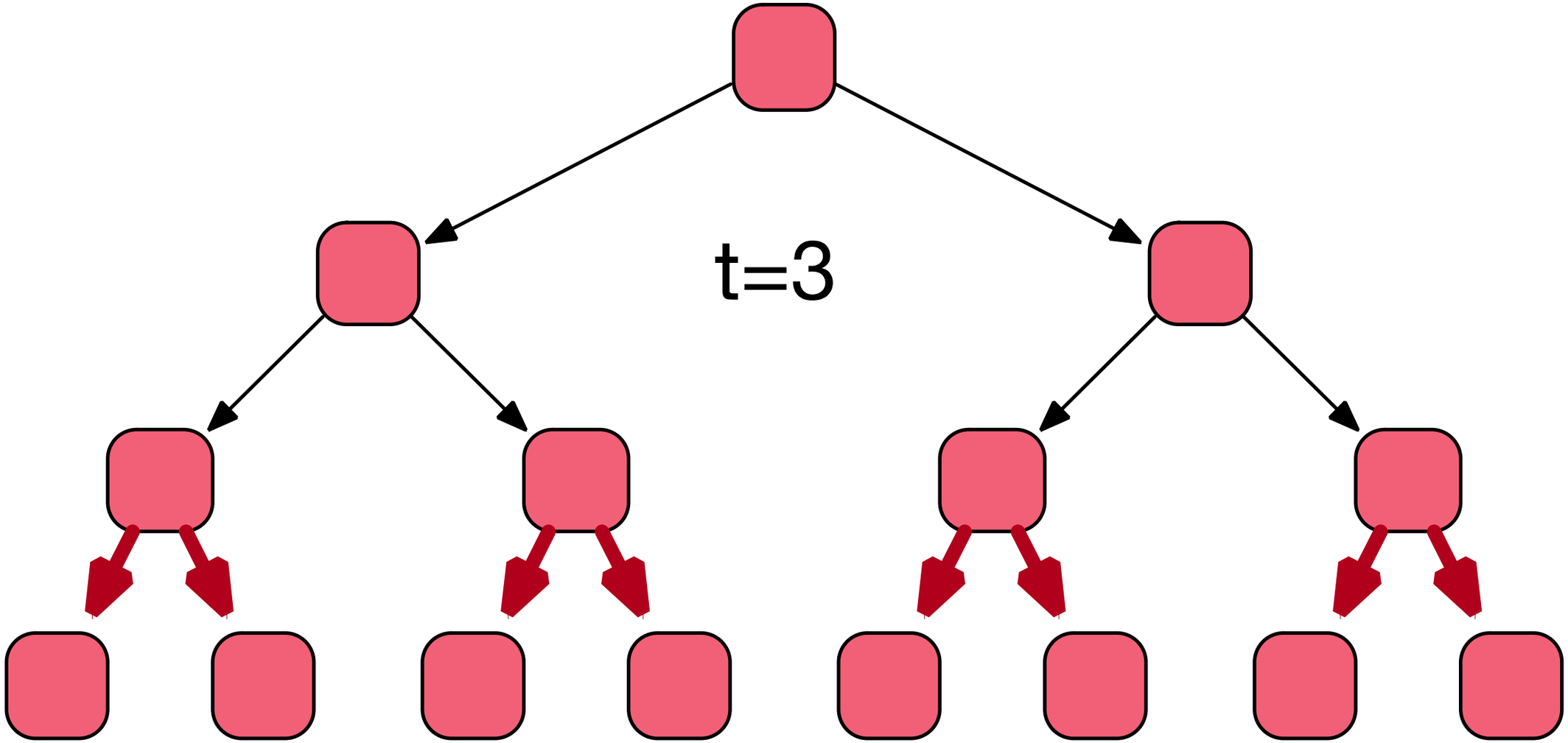
Better Strategy → Recursive Doubling



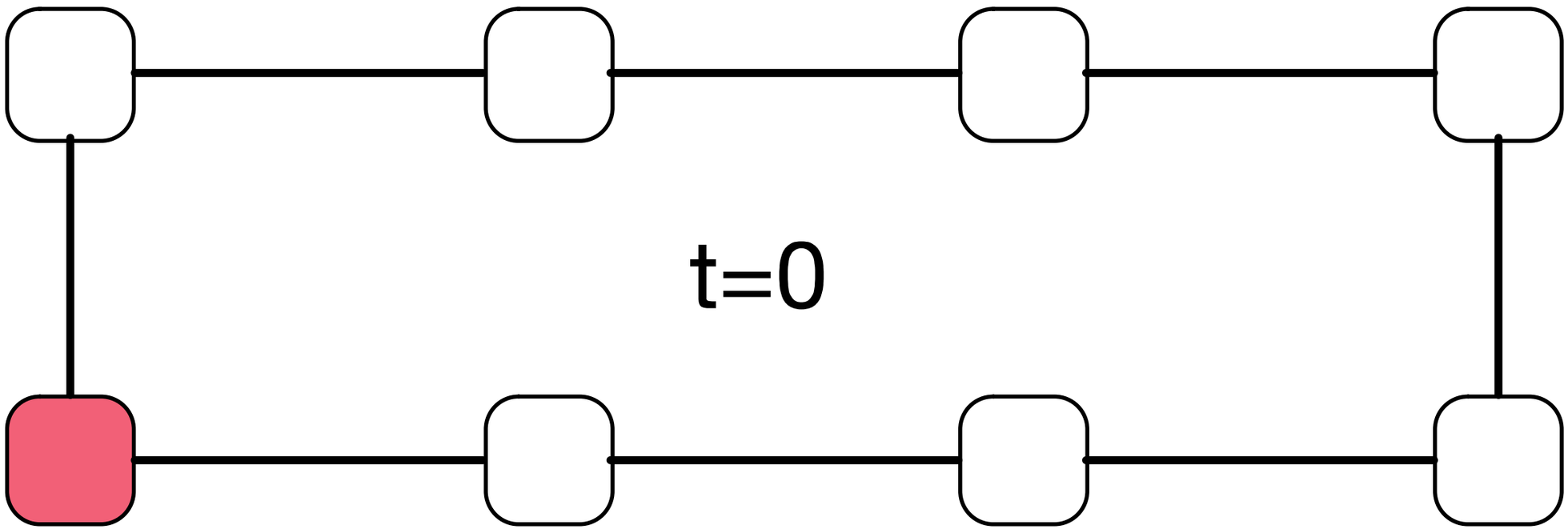
Better Strategy → Recursive Doubling



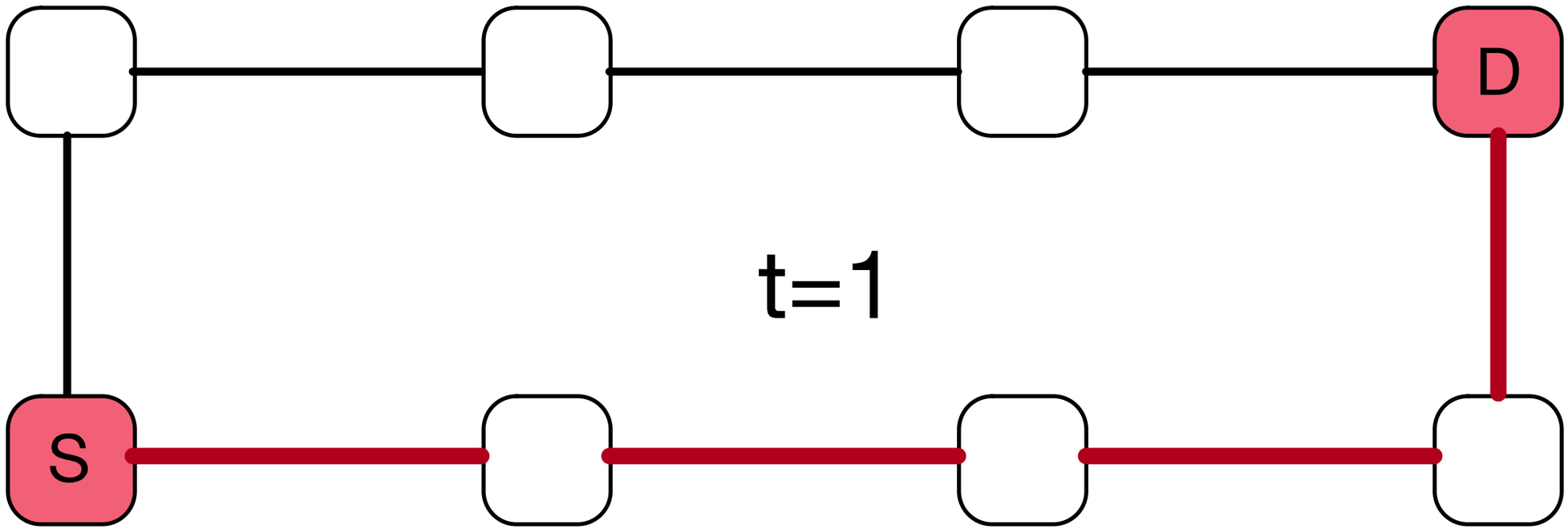
Better Strategy → Recursive Doubling



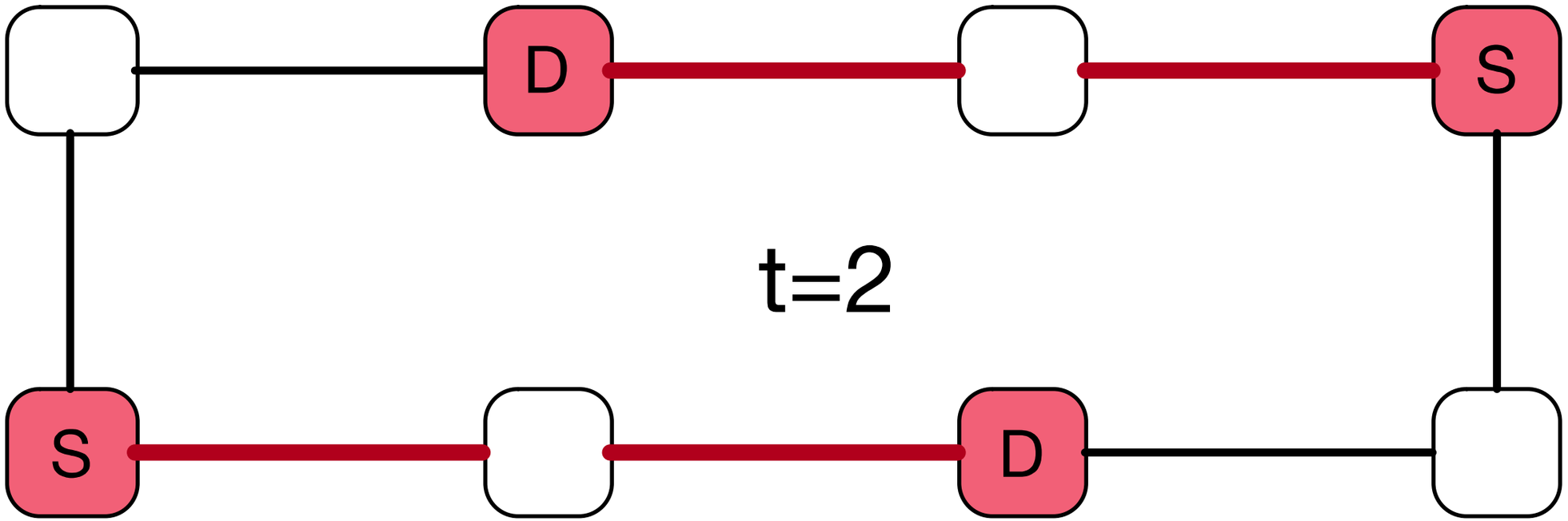
Better Strategy → Recursive Doubling



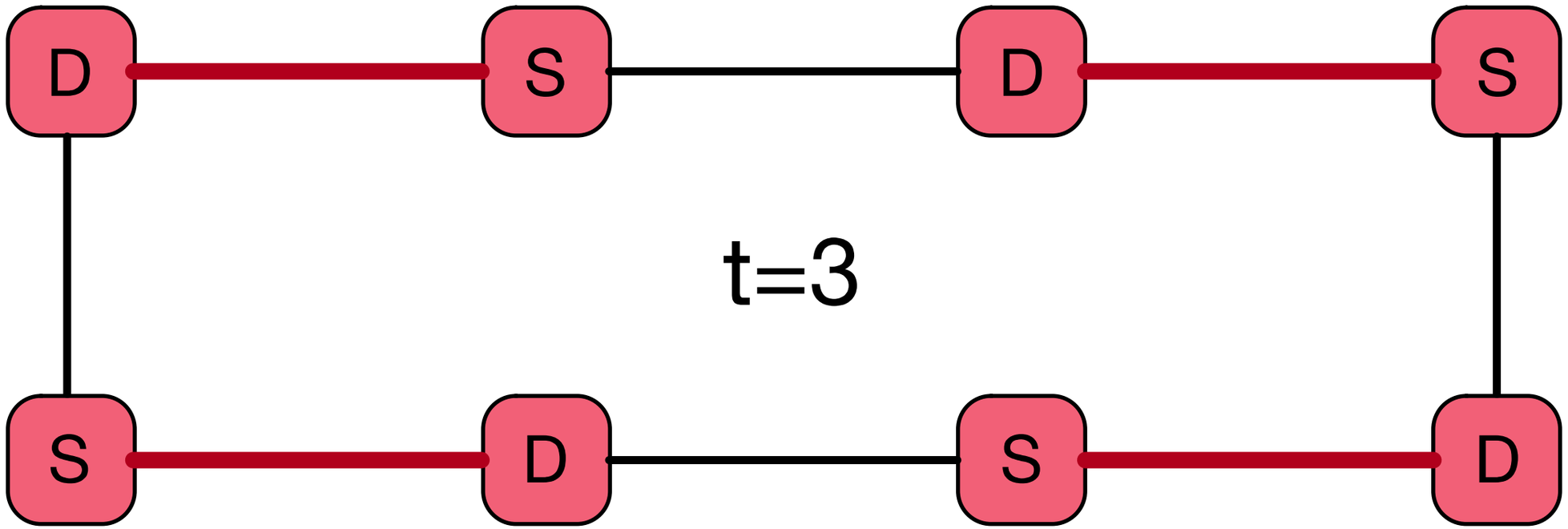
RING



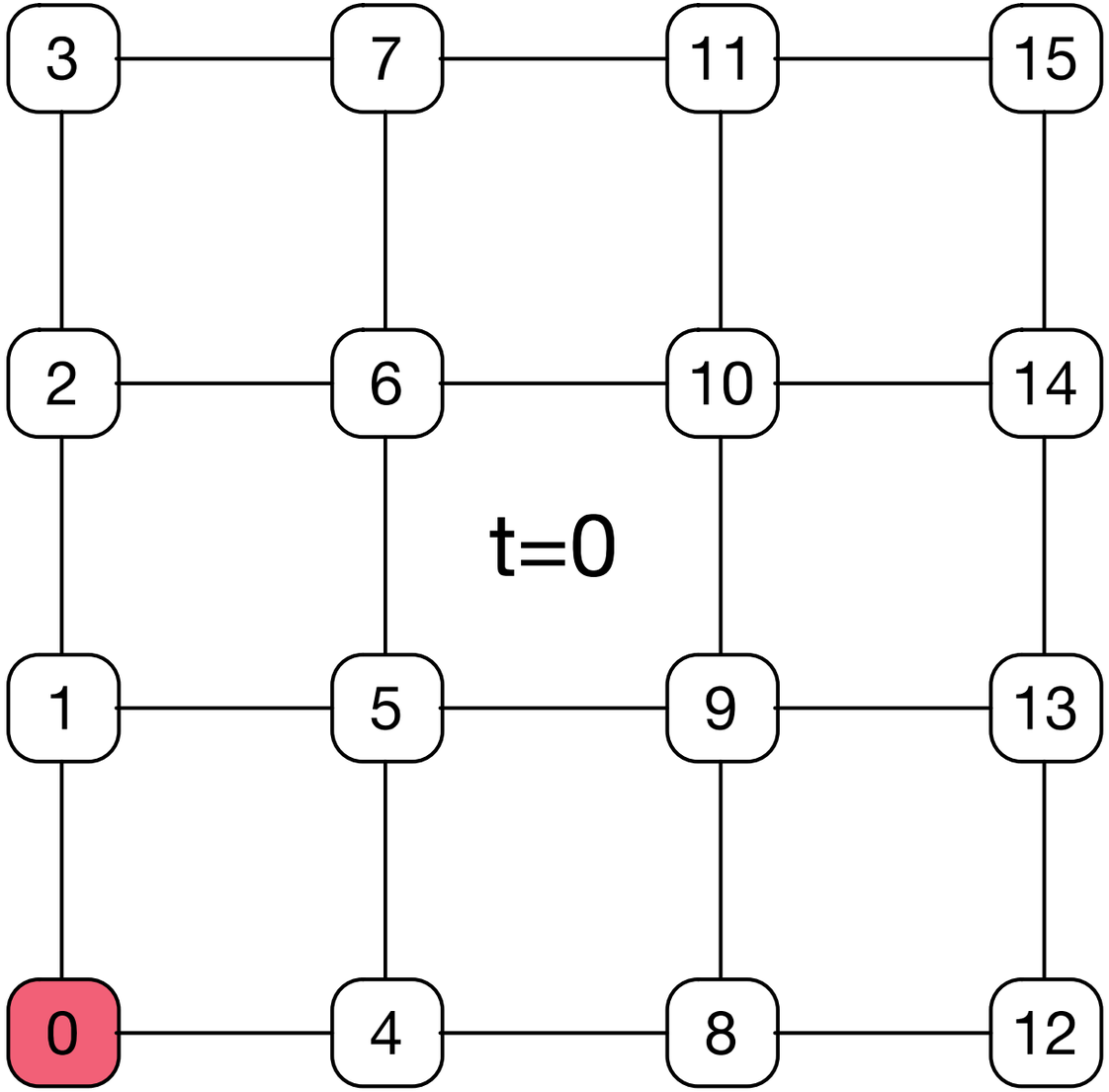
RING



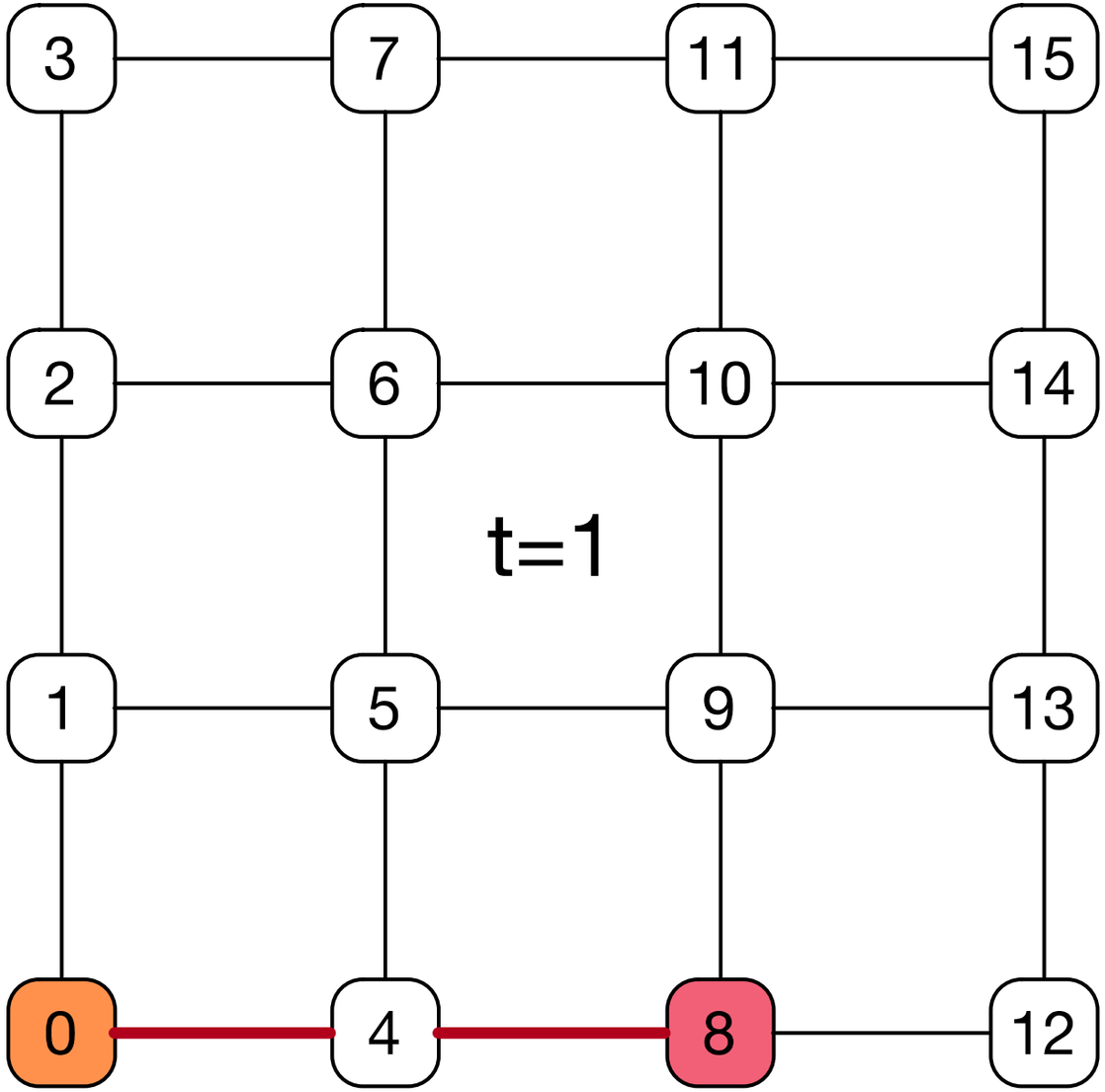
RING



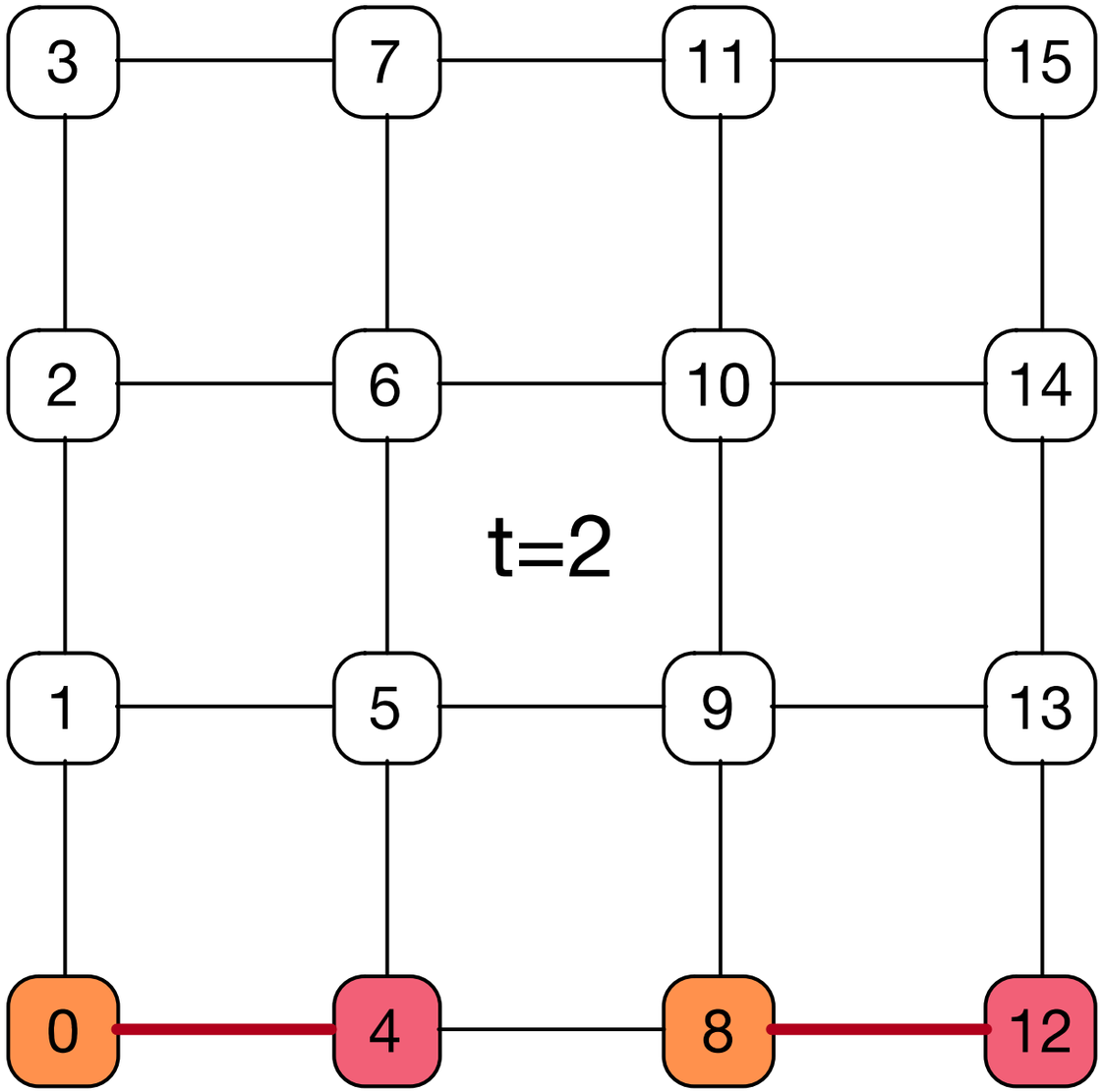
RING



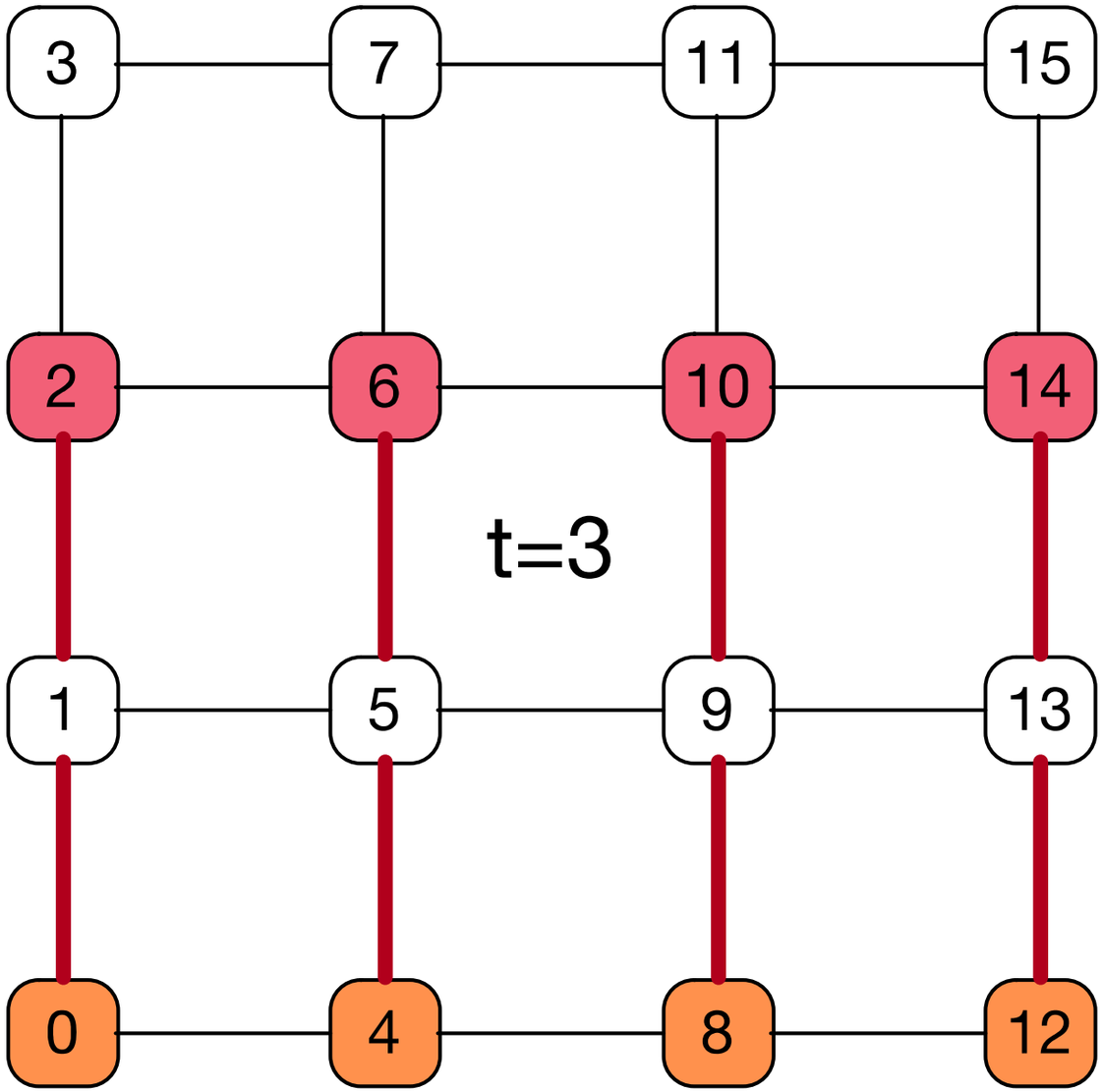
MESH



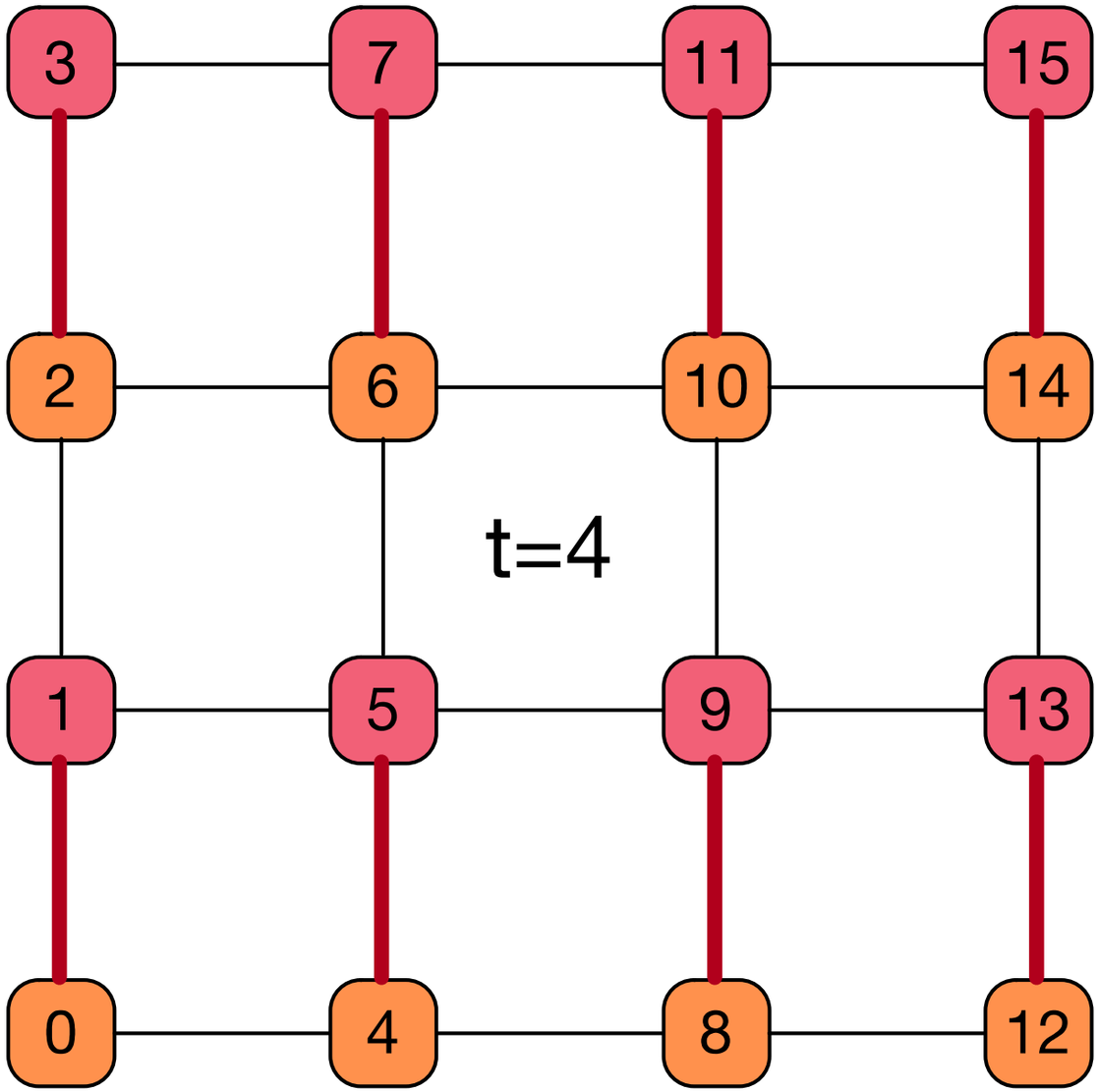
MESH



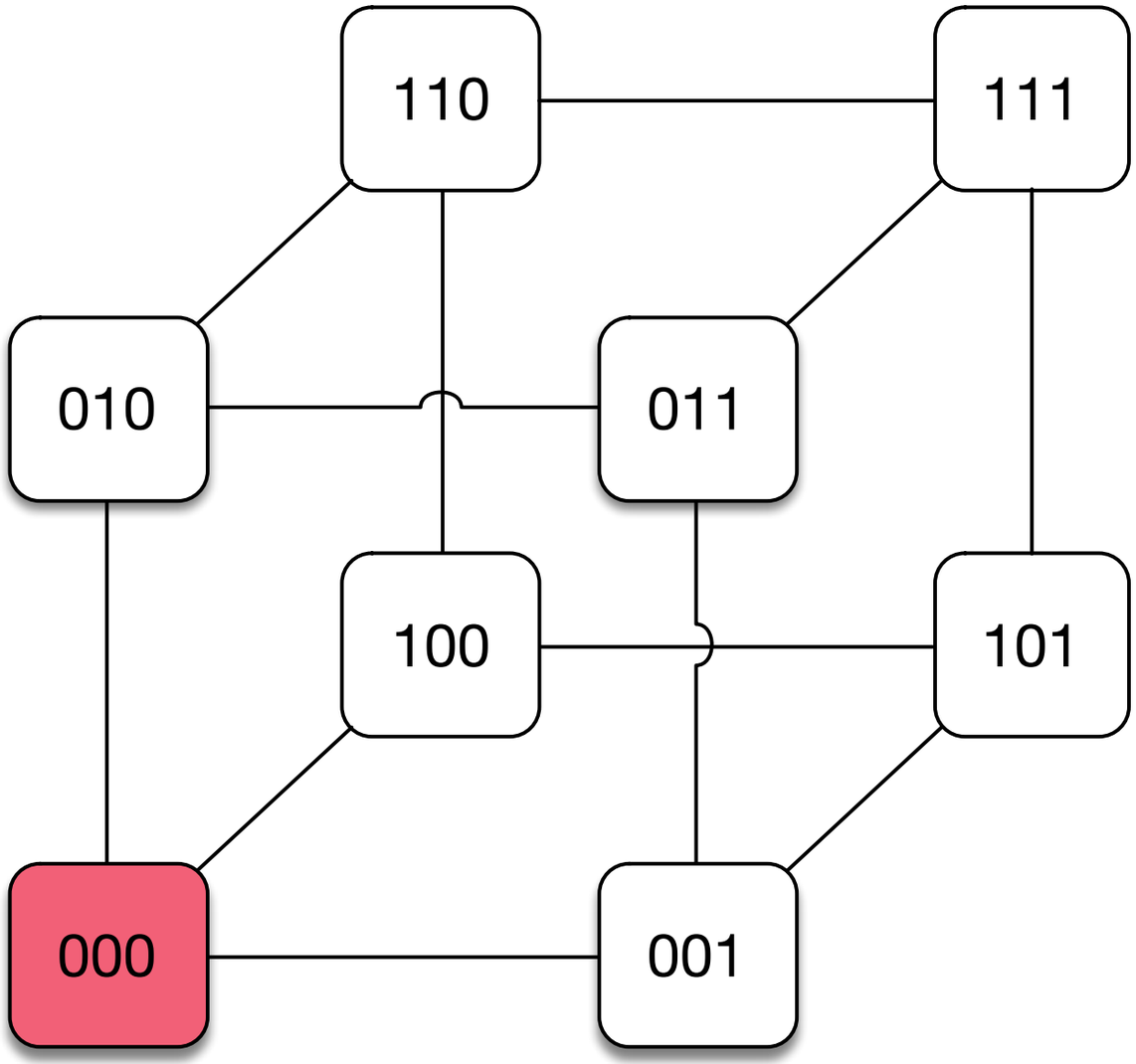
MESH



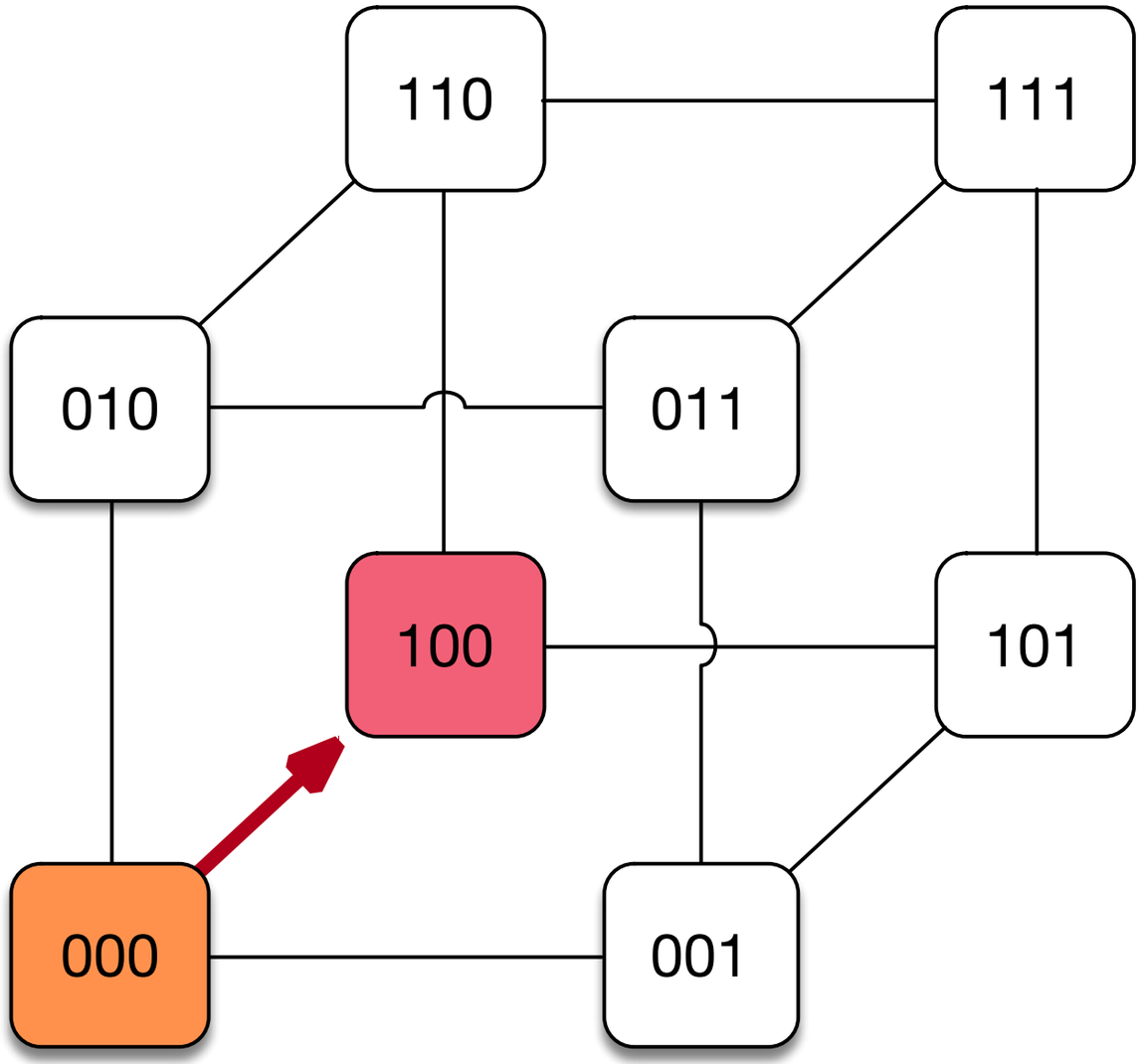
MESH



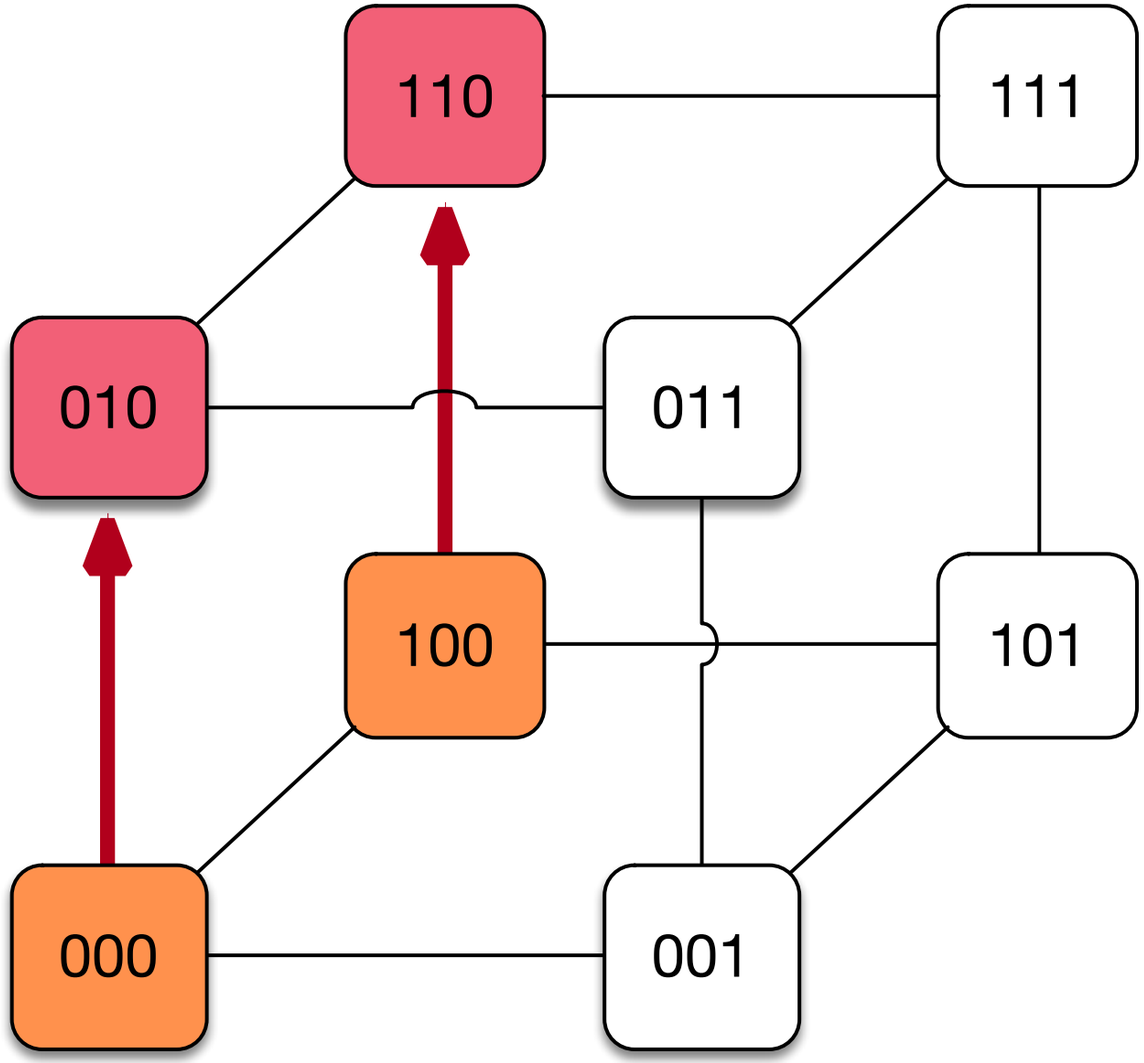
MESH



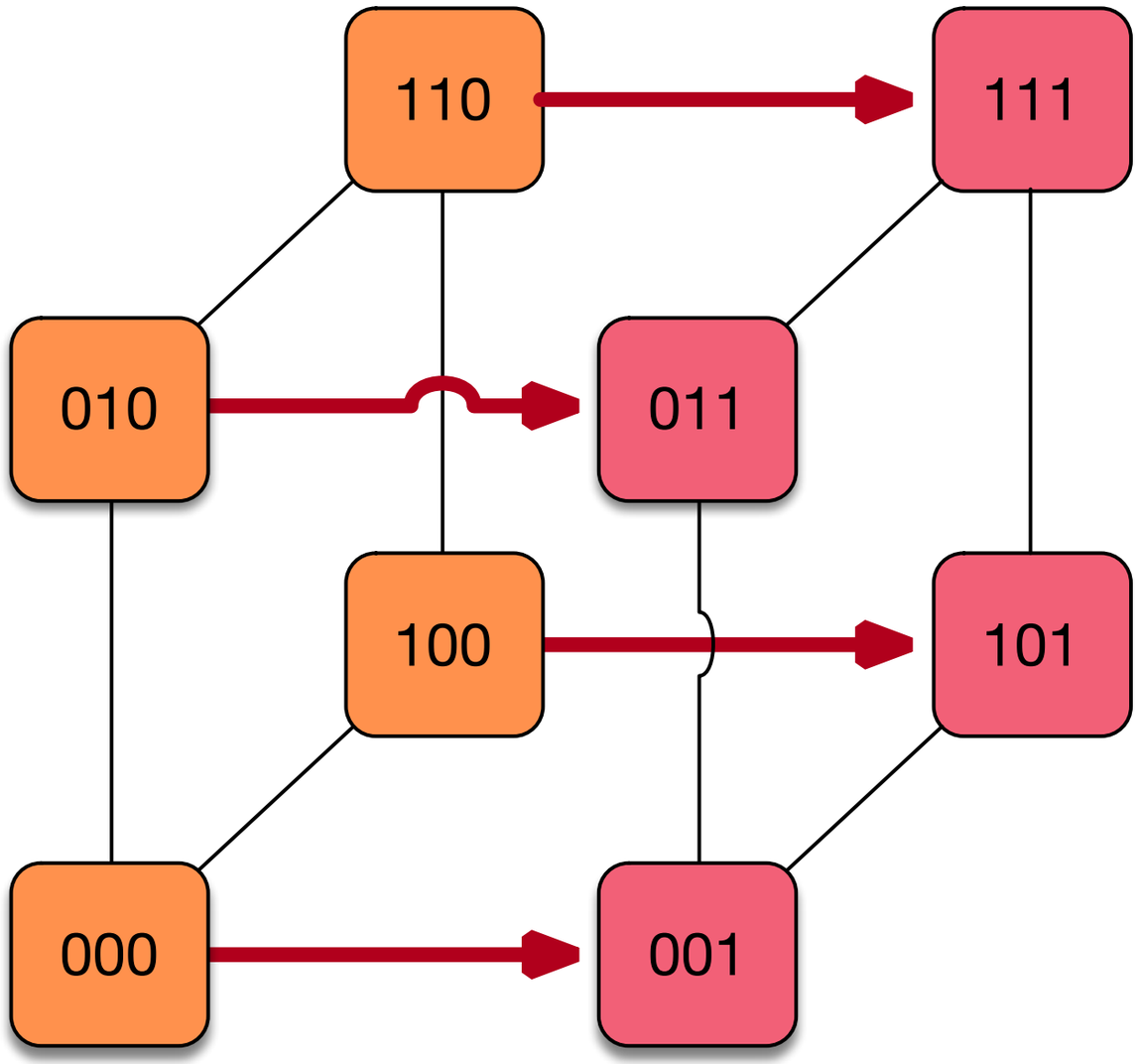
HYPERCUBE



HYPERCUBE



HYPERCUBE



HYPERCUBE

Cost Analysis for (1-n) in a Hypercube

- Cost of sending a single message = $T_{setup} + T_{transmission}$

→ $T_s + T_w$ x Size of data

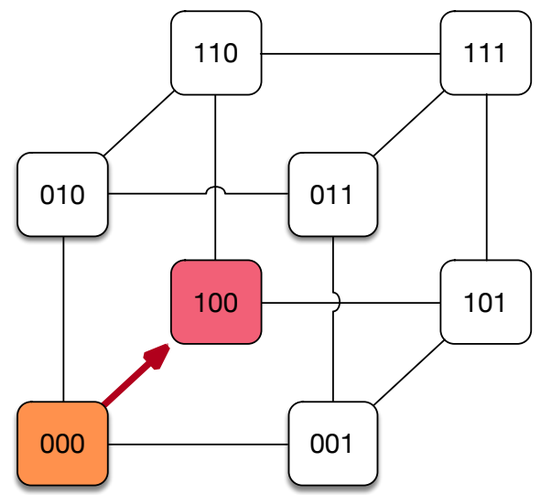
→ $T_s + T_w \times m$

Total Cost = $(T_s + T_w \times m)$ x number of messages

log p messages

Cost → $(T_s + T_w \times m) \times \log p$

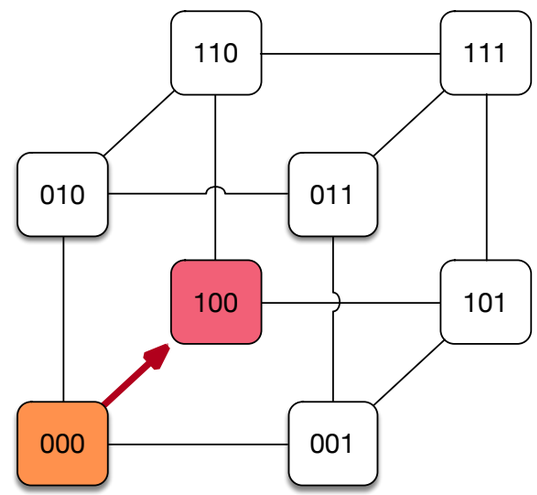
```
procedure ONE_TO_ALL_BC(d, my_id, X)
begin
    mask := 2^d − 1;                          /* Set all d bits of mask to 1 */   (111)
    for i := d − 1 downto 0 do                /* Outer loop */   i = d-1 = 2
        mask := mask XOR 2^i;                 /* Set bit i of mask to 0 */   111 XOR 100 = (011)
(100 & 011) = 000  if (my_id AND mask) = 0 then   /* If lower i bits of my_id are 0 */   (000 & 011) = 000
(100 & 100) = 100      if (my_id AND 2^i) = 0 then                   (000 & 100) = 000
            msg_destination := my_id XOR 2^i;                        (000 XOR 100) = 100
            send X to msg_destination;                               0 → 4
        else                                   (100 XOR 100) = 000
            msg_source := my_id XOR 2^i;
            receive X from msg_source;
        endelse;
    endif;
    endfor;
end ONE_TO_ALL_BC
```



1→All (Case 0 to 4)

my_id = 0

```
procedure ONE_TO_ALL_BC(d, my_id, X)
begin
    mask := 2^d − 1;                          /* Set all d bits of mask to 1 */   (111)
    for i := d − 1 downto 0 do                /* Outer loop */   i = d-1 = 2
        mask := mask XOR 2^i;                 /* Set bit i of mask to 0 */   111 XOR 100 = (011)
(100 & 011) = 000  if (my_id AND mask) = 0 then   /* If lower i bits of my_id are 0 */   (000 & 011) = 000
(100 & 100) = 100      if (my_id AND 2^i) = 0 then   (000 & 100) = 000
                           msg_destination := my_id XOR 2^i;   (000 XOR 100) = 100
                           send X to msg_destination;   0 → 4
                       else   (100 XOR 100) = 000
                           msg_source := my_id XOR 2^i;
                           receive X from msg_source;
                       endelse;
                   endif;
    endfor;
end ONE_TO_ALL_BC
```



1→All (Case 0 to 4)

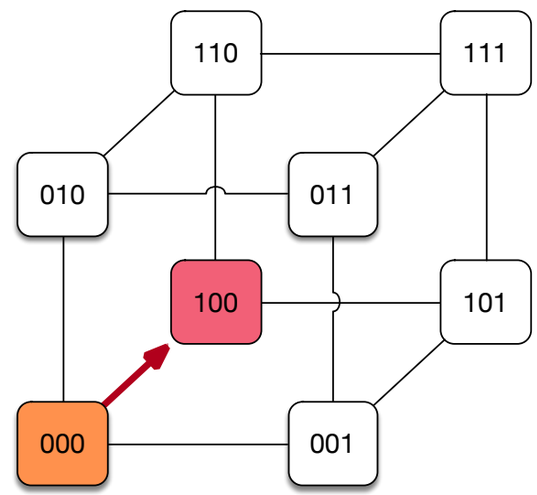
my_id = 0

```
procedure ONE_TO_ALL_BC(d, my_id, X)
begin
    mask := 2^d − 1;                          /* Set all d bits of mask to 1 */   (111)
    for i := d − 1 downto 0 do                /* Outer loop */   i = d-1 = 2
        mask := mask XOR 2^i;                 /* Set bit i of mask to 0 */   111 XOR 100 = (011)
(100 & 011) = 000  if (my_id AND mask) = 0 then   /* If lower i bits of my_id are 0 */   (000 & 011) = 000
(100 & 100) = 100      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
```

(100 XOR 100) = 000

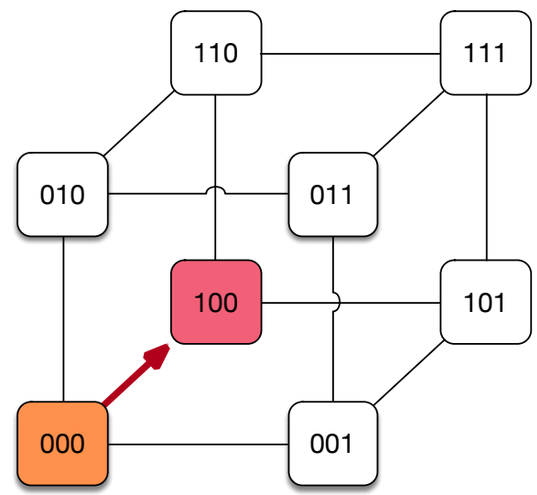
(000 & 100) = 000
(000 XOR 100) = 100
0 → 4

NODE 0



1→All (Case 0 to 4)

```
procedure ONE_TO_ALL_BC(d, my_id, X)
begin
    mask := 2^d − 1;                          /* Set all d bits of mask to 1 */   (111)
    for i := d − 1 downto 0 do                /* Outer loop */   i = d-1 = 2
        mask := mask XOR 2^i;                 /* Set bit i of mask to 0 */   111 XOR 100 = (011)
(100 & 011) = 000  if (my_id AND mask) = 0 then   /* If lower i bits of my_id are 0 */   (000 & 011) = 000
(100 & 100) = 100      if (my_id AND 2^i) = 0 then                   (000 & 100) = 000
            msg_destination := my_id XOR 2^i;                        (000 XOR 100) = 100
            send X to msg_destination;                               0 → 4
        else                                   (100 XOR 100) = 000
            msg_source := my_id XOR 2^i;
            receive X from msg_source;
        endelse;
    endif;
    endfor;
end ONE_TO_ALL_BC
```

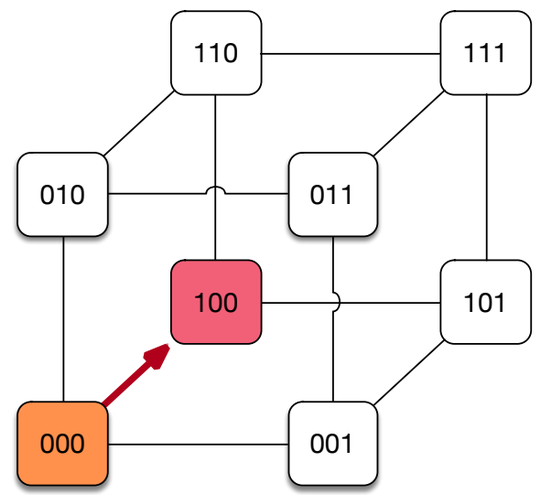


1→All (Case 0 to 4)

```
procedure ONE_TO_ALL_BC(d, my_id, X)
begin
    mask := 2^d − 1;                          /* Set all d bits of mask to 1 */   (111)
    for i := d − 1 downto 0 do                /* Outer loop */   i = d-1 = 2
        mask := mask XOR 2^i;                 /* Set bit i of mask to 0 */   111 XOR 100 = (011)
        if (my_id AND mask) = 0 then          /* If lower i bits of my_id are 0 */   (000 & 011) = 000
            if (my_id AND 2^i) = 0 then                                   (000 & 100) = 000
                msg_destination := my_id XOR 2^i;                         (000 XOR 100) = 100
                send X to msg_destination;                                0 → 4
            else                                                          (100 XOR 100) = 000
                msg_source := my_id XOR 2^i;
                receive X from msg_source;
            endelse;
        endif;
    endfor;
end ONE_TO_ALL_BC
```

(100 & 011) = 000
(100 & 100) = 100

NODE 4



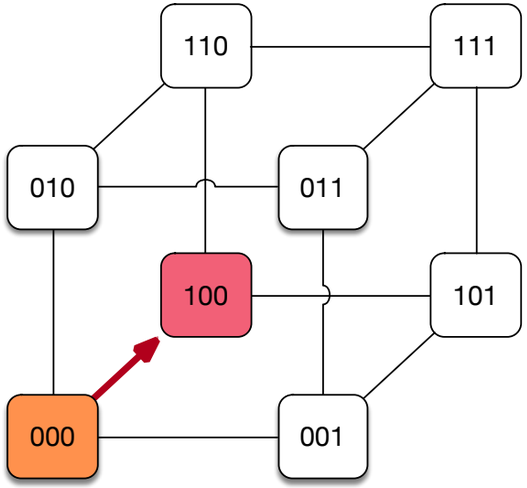
1→All (Case 0 to 4)

my_id = 4

```
procedure ONE_TO_ALL_BC(d, my_id, X)
begin
    mask := 2^d − 1;                    /* Set all d bits of mask to 1 */   (111)
    for i := d − 1 downto 0 do          /* Outer loop */   i = d-1 = 2
        mask := mask XOR 2^i;           /* Set bit i of mask to 0 */   111 XOR 100 = (011)
        if (my_id AND mask) = 0 then    /* If lower i bits of my_id are 0 */   (000 & 011) = 000
            if (my_id AND 2^i) = 0 then                                 (000 & 100) = 000
                msg_destination := my_id XOR 2^i;                       (000 XOR 100) = 100
                send X to msg_destination;                              0 → 4
            else                                                        (100 XOR 100) = 000
                msg_source := my_id XOR 2^i;
                receive X from msg_source;
            endelse;
        endif;
    endfor;
end ONE_TO_ALL_BC
```

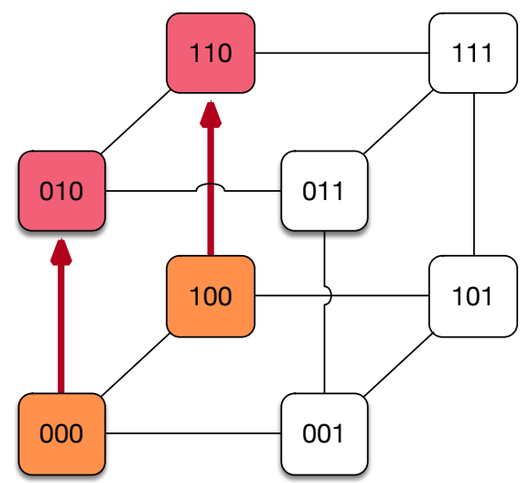
(100 & 011) = 000
(100 & 100) = 100

NODE 4



1→All (Case 0 to 4)

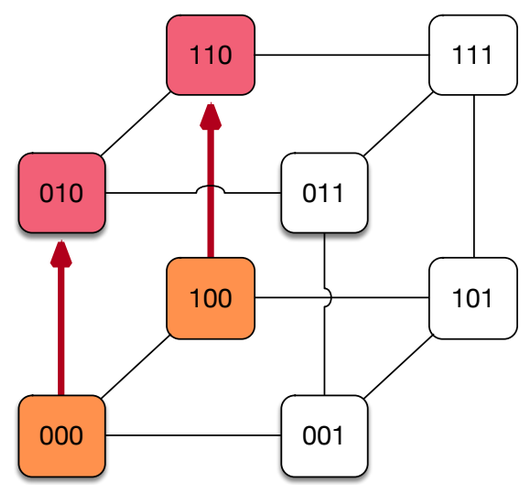
```
procedure ONE_TO_ALL_BC(d, my_id, X)
begin
    mask := 2^d − 1;                          /* Set all d bits of mask to 1 */   (111)
    for i := d − 1 downto 0 do                /* Outer loop */                     i=1
        mask := mask XOR 2^i;                 /* Set bit i of mask to 0 */        111 XOR 010 = (101)
(010 & 101) = 000  if (my_id AND mask) = 0 then  /* If lower i bits of my_id are 0 */   (000 & 101) = 000
(010 & 010) = 010      if (my_id AND 2^i) = 0 then                                (000 & 010) = 000
                           msg_destination := my_id XOR 2^i;                      (000 XOR 010) = 010
                           send X to msg_destination;                             0 → 2
                       else                                                       (010 XOR 010) = 000
                           msg_source := my_id XOR 2^i;
                           receive X from msg_source;
                       endelse;
                   endif;
    endfor;
end ONE_TO_ALL_BC
```



1→All (Case 0 to 2)

my_id = 0

```
procedure ONE_TO_ALL_BC(d, my_id, X)
begin
    mask := 2^d − 1;                          /* Set all d bits of mask to 1 */   (111)
    for i := d − 1 downto 0 do                /* Outer loop */                     i=1
        mask := mask XOR 2^i;                 /* Set bit i of mask to 0 */        111 XOR 010 = (101)
(010 & 101) = 000  if (my_id AND mask) = 0 then  /* If lower i bits of my_id are 0 */   (000 & 101) = 000
(010 & 010) = 010      if (my_id AND 2^i) = 0 then                                (000 & 010) = 000
                           msg_destination := my_id XOR 2^i;                      (000 XOR 010) = 010
                           send X to msg_destination;                             0 → 2
                       else                                                       (010 XOR 010) = 000
                           msg_source := my_id XOR 2^i;
                           receive X from msg_source;
                       endelse;
                   endif;
    endfor;
end ONE_TO_ALL_BC
```



1→All (Case 0 to 2)

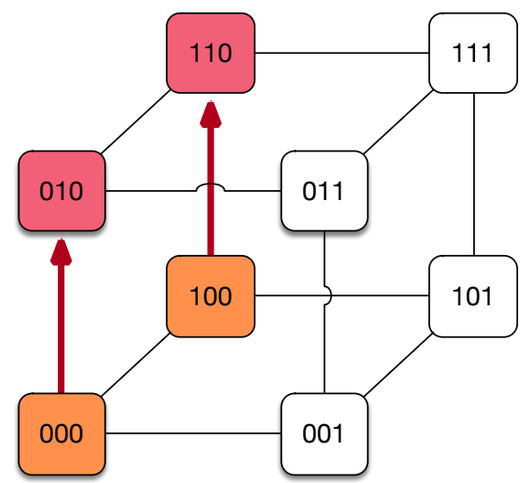
my_id = 0

```
procedure ONE_TO_ALL_BC(d, my_id, X)
begin
    mask := 2^d − 1;                          /* Set all d bits of mask to 1 */   (111)
    for i := d − 1 downto 0 do                /* Outer loop */                     i=1
        mask := mask XOR 2^i;                 /* Set bit i of mask to 0 */        111 XOR 010 = (101)
(010 & 101) = 000  if (my_id AND mask) = 0 then  /* If lower i bits of my_id are 0 */   (000 & 101) = 000
(010 & 010) = 010      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 & 010) = 000
(000 XOR 010) = 010
0 → 2

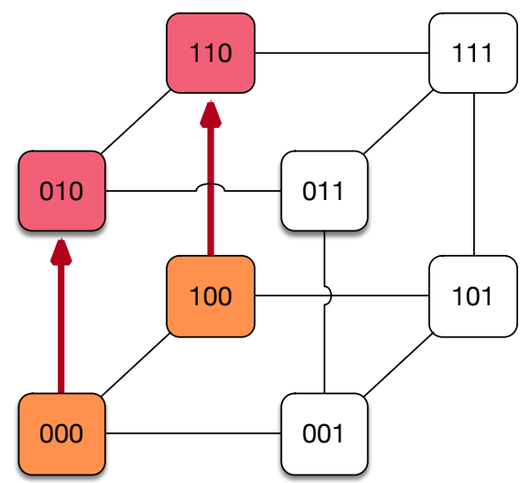
NODE 0

(010 XOR 010) = 000



1→All (Case 0 to 2)

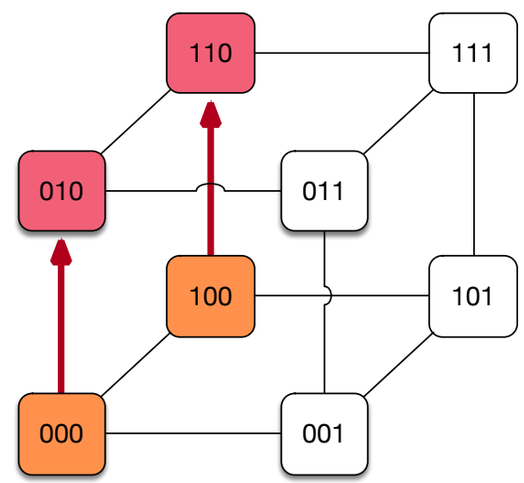
```
procedure ONE_TO_ALL_BC(d, my_id, X)
begin
    mask := 2^d − 1;                          /* Set all d bits of mask to 1 */   (111)
    for i := d − 1 downto 0 do                /* Outer loop */                     i=1
        mask := mask XOR 2^i;                 /* Set bit i of mask to 0 */        111 XOR 010 = (101)
(010 & 101) = 000  if (my_id AND mask) = 0 then  /* If lower i bits of my_id are 0 */   (000 & 101) = 000
(010 & 010) = 010      if (my_id AND 2^i) = 0 then                                (000 & 010) = 000
                           msg_destination := my_id XOR 2^i;                      (000 XOR 010) = 010
                           send X to msg_destination;                             0 → 2
                       else                                                       (010 XOR 010) = 000
                           msg_source := my_id XOR 2^i;
                           receive X from msg_source;
                       endelse;
                   endif;
    endfor;
end ONE_TO_ALL_BC
```



1→All (Case 0 to 2)

my_id = 2

```
procedure ONE_TO_ALL_BC(d, my_id, X)
begin
    mask := 2^d − 1;                          /* Set all d bits of mask to 1 */   (111)
    for i := d − 1 downto 0 do                /* Outer loop */                     i=1
        mask := mask XOR 2^i;                 /* Set bit i of mask to 0 */        111 XOR 010 = (101)
(010 & 101) = 000  if (my_id AND mask) = 0 then  /* If lower i bits of my_id are 0 */   (000 & 101) = 000
(010 & 010) = 010      if (my_id AND 2^i) = 0 then                                (000 & 010) = 000
                           msg_destination := my_id XOR 2^i;                      (000 XOR 010) = 010
                           send X to msg_destination;                             0 → 2
                       else                                                       (010 XOR 010) = 000
                           msg_source := my_id XOR 2^i;
                           receive X from msg_source;
                       endelse;
                   endif;
    endfor;
end ONE_TO_ALL_BC
```



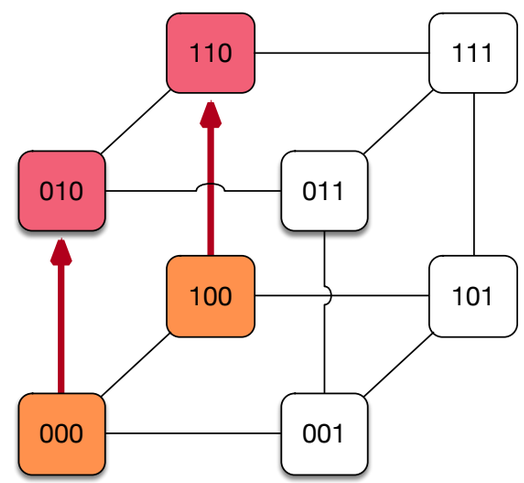
1→All (Case 0 to 2)

my_id = 2

```
procedure ONE_TO_ALL_BC(d, my_id, X)
begin
    mask := 2^d − 1;                          /* Set all d bits of mask to 1 */   (111)
    for i := d − 1 downto 0 do                /* Outer loop */                     i=1
        mask := mask XOR 2^i;                 /* Set bit i of mask to 0 */        111 XOR 010 = (101)
        if (my_id AND mask) = 0 then          /* If lower i bits of my_id are 0 */   (000 & 101) = 000
            if (my_id AND 2^i) = 0 then                                            (000 & 010) = 000
                msg_destination := my_id XOR 2^i;                                  (000 XOR 010) = 010
                send X to msg_destination;                                         0 → 2
            else                                                                   (010 XOR 010) = 000
                msg_source := my_id XOR 2^i;
                receive X from msg_source;
            endelse;
        endif;
    endfor;
end ONE_TO_ALL_BC
```

(010 & 101) = 000
(010 & 010) = 010

NODE 2



1→All (Case 0 to 2)

```
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
```

(111)

i=0

111 XOR 001 = (110)

(001 & 110) = 000

(000 & 110) = 000

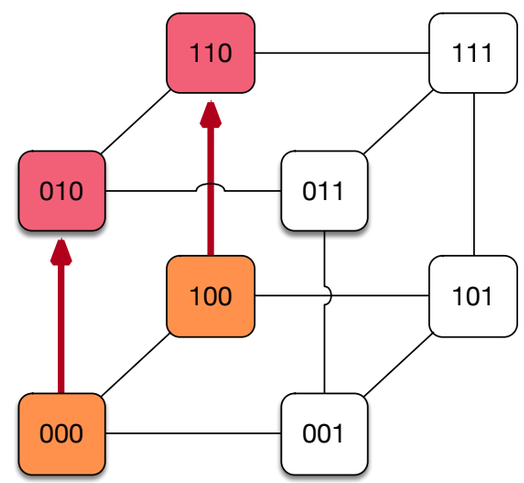
(001 & 001) = 001

(000 & 001) = 000

(000 XOR 001) =001

0 → 1

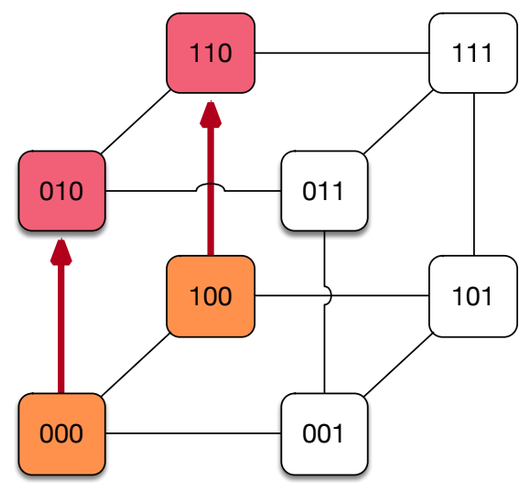
(001 XOR 001) = 000



1→All (Case 0 to 1)

my_id = 0

```
procedure ONE_TO_ALL_BC(d, my_id, X)
begin
    mask := 2^d − 1;                          /* Set all d bits of mask to 1 */   (111)
    for i := d − 1 downto 0 do                /* Outer loop */                     i=0
        mask := mask XOR 2^i;                 /* Set bit i of mask to 0 */         111 XOR 001 = (110)
(001 & 110) = 000  if (my_id AND mask) = 0 then  /* If lower i bits of my_id are 0 */   (000 & 110) = 000
(001 & 001) = 001      if (my_id AND 2^i) = 0 then                                  (000 & 001) = 000
                           msg_destination := my_id XOR 2^i;                        (000 XOR 001) =001
                           send X to msg_destination;                               0 → 1
                       else                                  (001 XOR 001) = 000
                           msg_source := my_id XOR 2^i;
                           receive X from msg_source;
                       endelse;
                   endif;
    endfor;
end ONE_TO_ALL_BC
```



1→All (Case 0 to 1)

my_id = 0

```
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
```

(111)

i=0

111 XOR 001 = (110)

(001 & 110) = 000

(000 & 110) = 000

(001 & 001) = 001

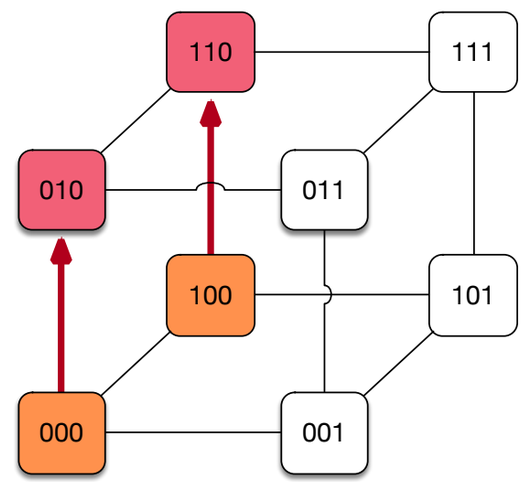
(000 & 001) = 000

(000 XOR 001) =001

0 → 1

NODE 0

(001 XOR 001) = 000



1→All (Case 0 to 1)

```
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
```

(111)

i=0

111 XOR 001 = (110)

(001 & 110) = 000

(000 & 110) = 000

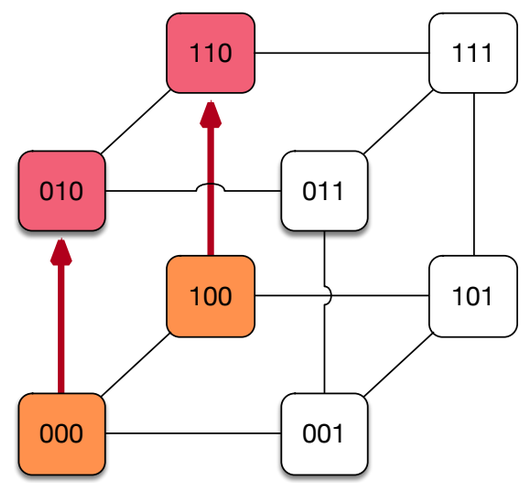
(001 & 001) = 001

(000 & 001) = 000

(000 XOR 001) =001

0 → 1

(001 XOR 001) = 000



1→All (Case 0 to 1)

my_id = 1

```
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
```

(111)

i=0

111 XOR 001 = (110)

(001 & 110) = 000

(000 & 110) = 000

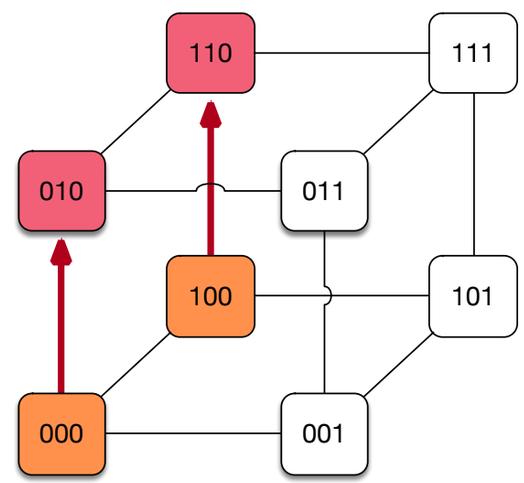
(001 & 001) = 001

(000 & 001) = 000

(000 XOR 001) =001

0 → 1

(001 XOR 001) = 000



1→All (Case 0 to 1)

my_id = 1

```
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
```

(111)

i=0

111 XOR 001 = (110)

(000 & 110) = 000

(000 & 001) = 000

(000 XOR 001) =001

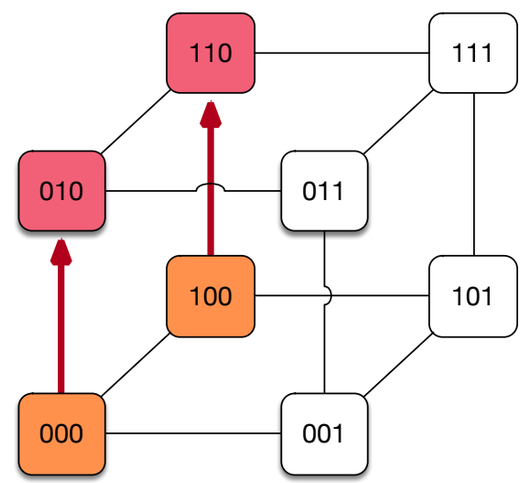
0 → 1

(001 & 110) = 000

(001 & 001) = 001

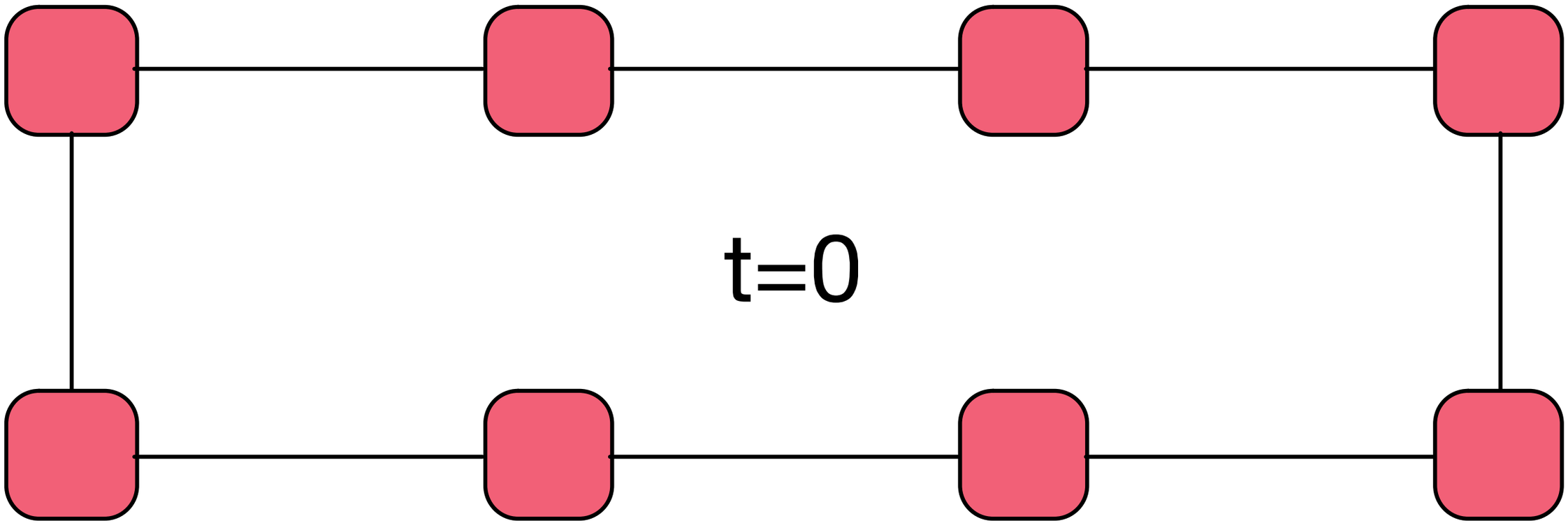
NODE 1

(001 XOR 001) = 000

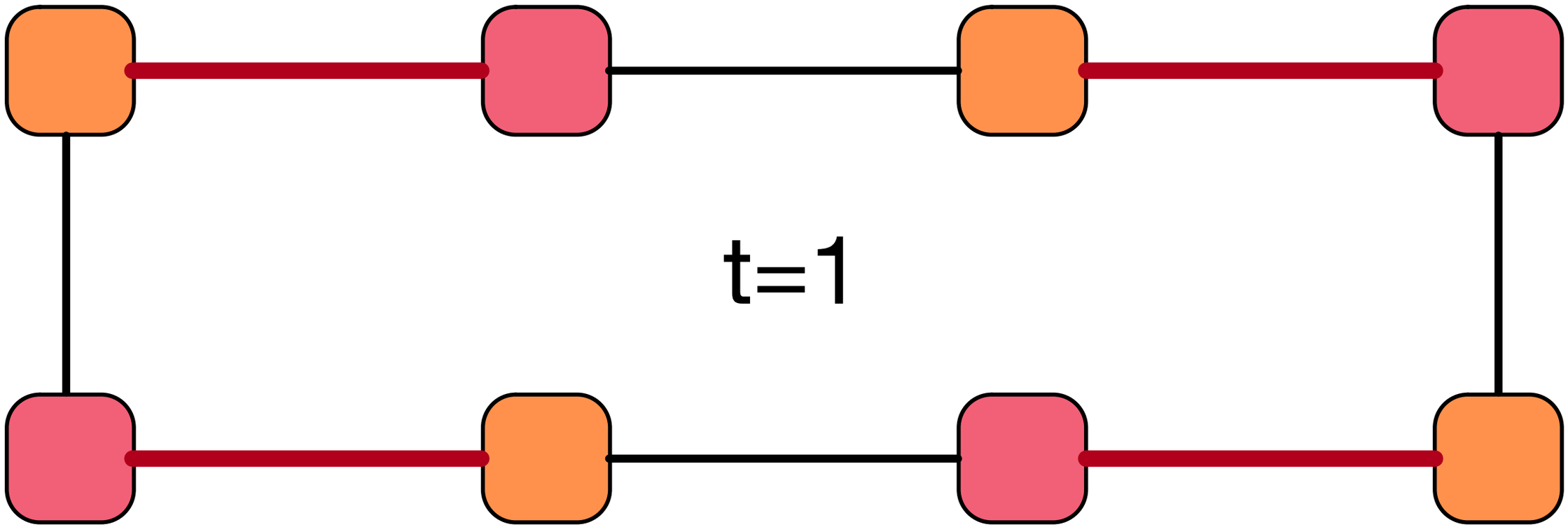


1→All (Case 0 to 1)

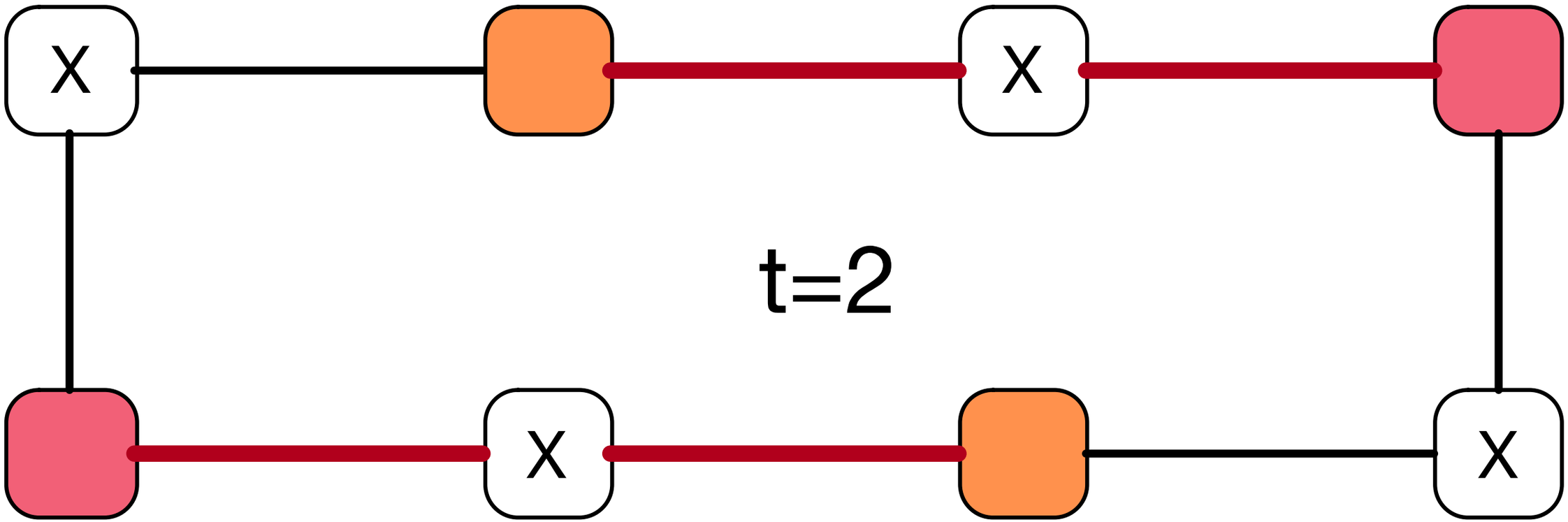
Reduction



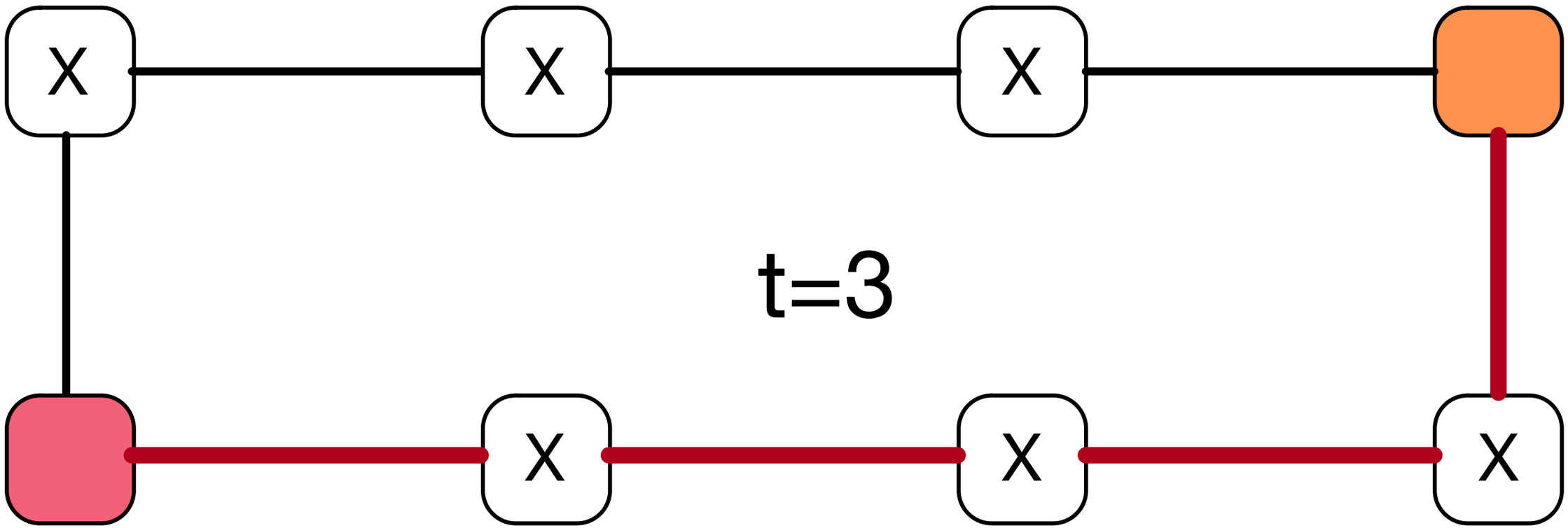
Reduction



Reduction



Reduction



Reduction