



**Chapter 6**  
**Consensus**  
**Distributed Systems**  
**SS 2015**  
**Fabian Kuhn**

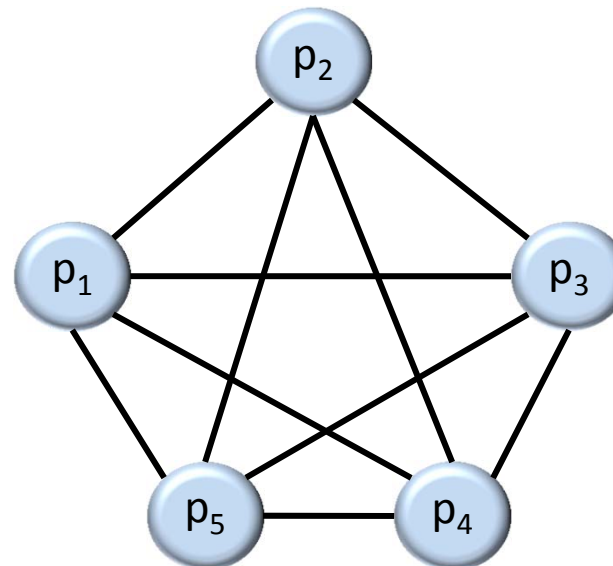
# Overview

---

- Introduction
- Consensus #1: Shared Memory
- Consensus #2: Wait-free Shared Memory
- Consensus #3: Read-Modify-Write Shared Memory
- Consensus #4: Synchronous Systems
- Consensus #5: Byzantine Failures
- Consensus #6: A Simple Algorithm for Byzantine Agreement
- Consensus #7: The Queen Algorithm
- Consensus #8: The King Algorithm
- Consensus #9: Byzantine Agreement Using Authentication
- Consensus #10: A Randomized Algorithm
- Shared Coin
  
- Slides by R. Wattenhofer (ETHZ)

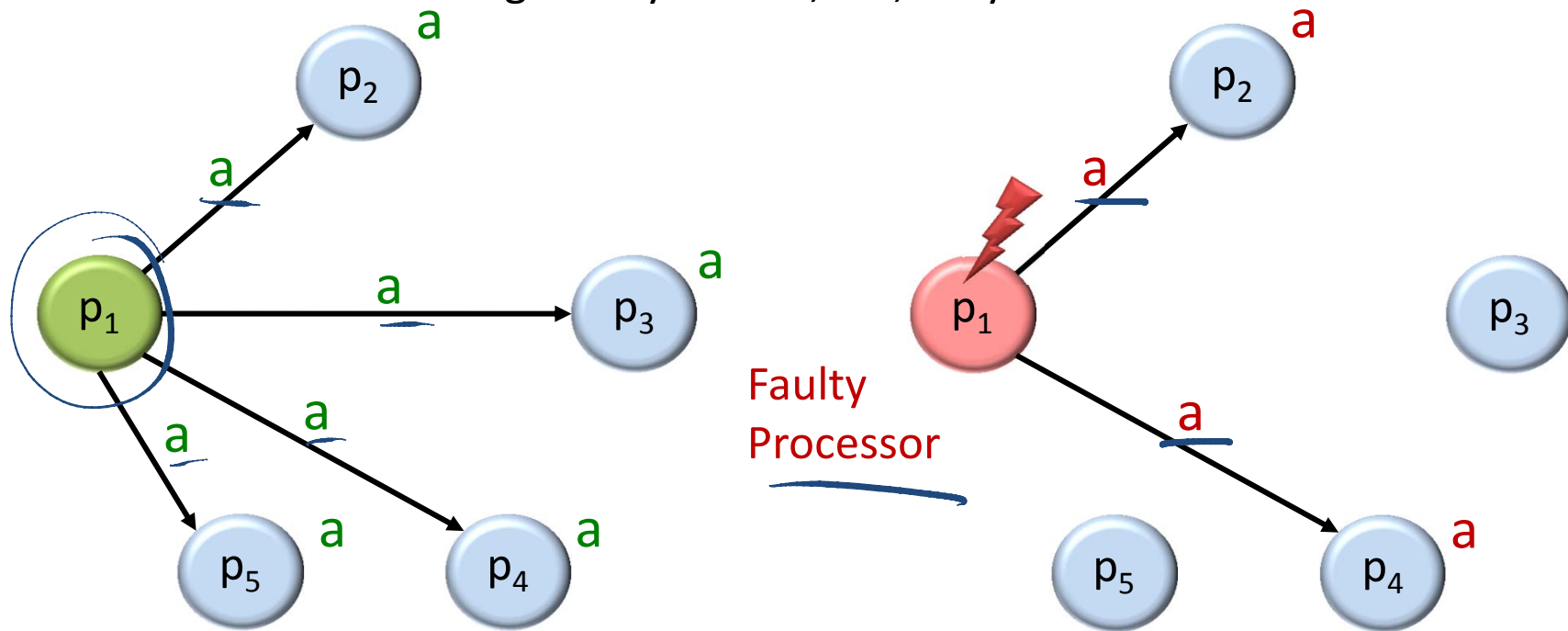
# Consensus #4: Synchronous Systems

- One can sometimes tell if a processor had crashed
  - Timeouts
  - Broken TCP connections
- Can one solve consensus at least in synchronous systems?
- Model
  - All communication occurs in synchronous rounds
  - Complete communication graph

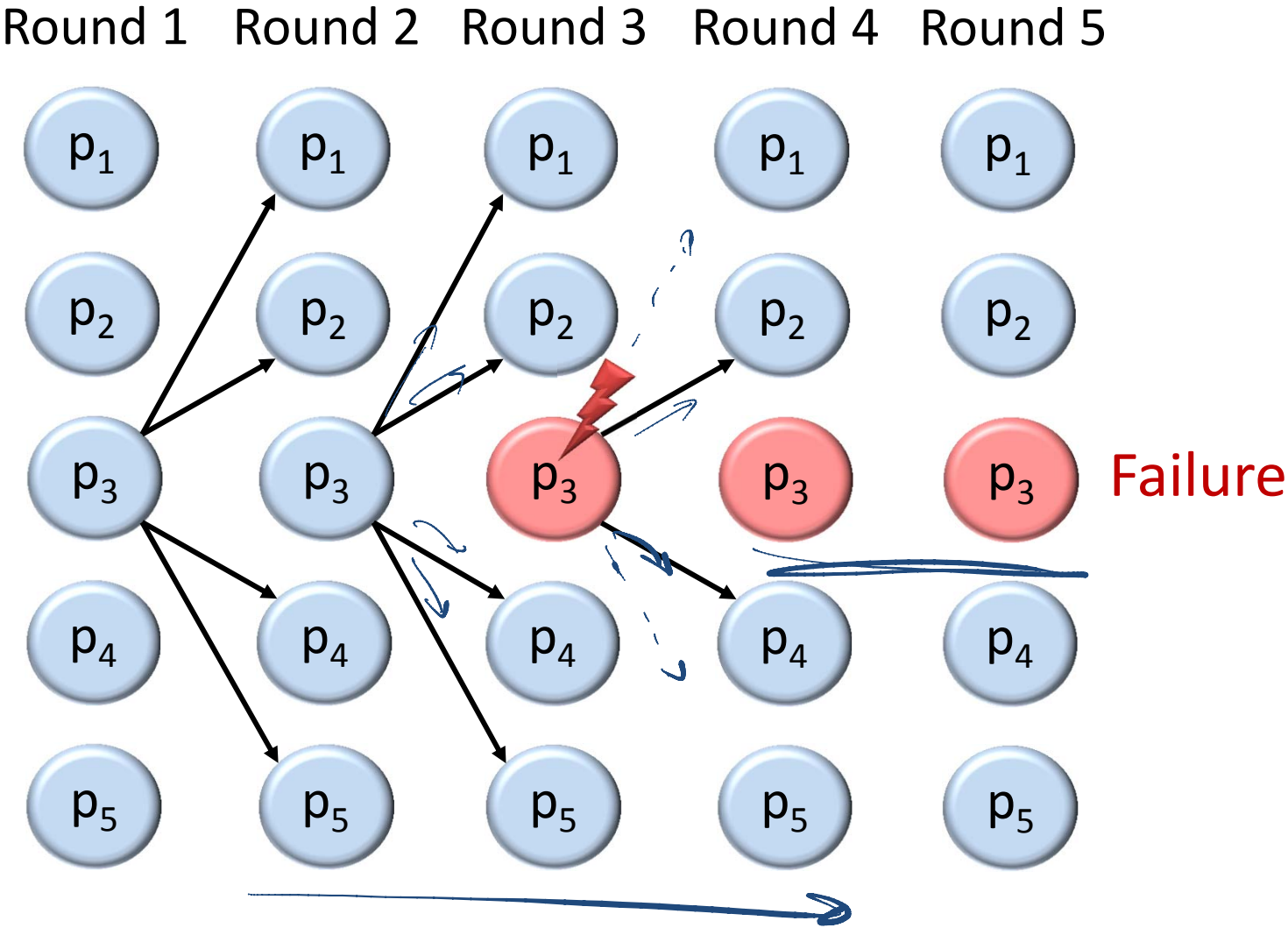


# Crash Failures

- Broadcast: Send a message to all nodes in one round
  - At the end of the round everybody receives the message **a**
  - Every process can broadcast a value in each round
- Crash Failures: A broadcast can fail if a process crashes
  - Some of the messages may be lost, i.e., they are never received



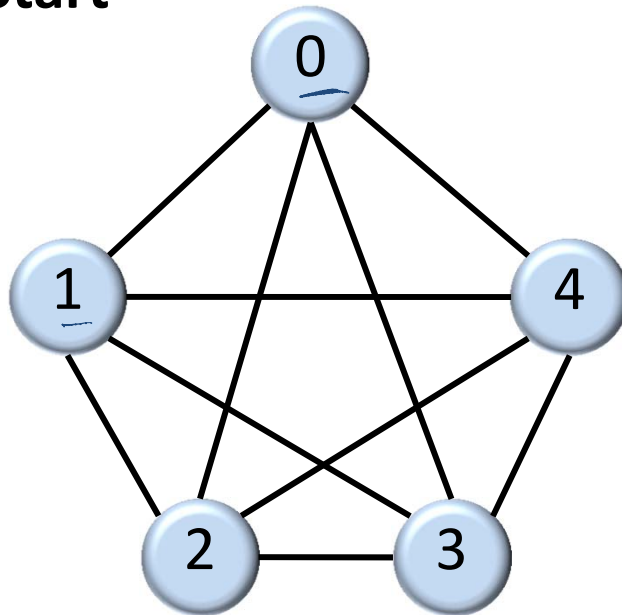
# Process disappears after failure



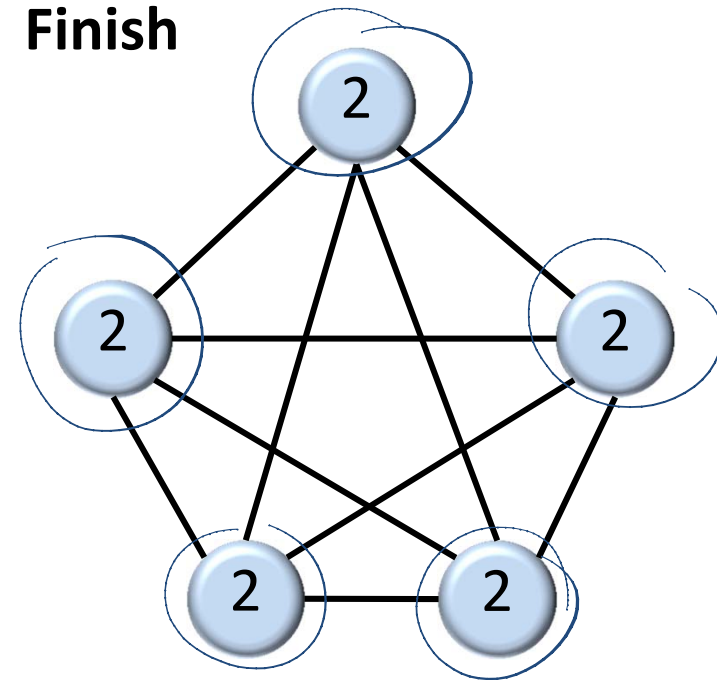
# Consensus Repetition

- Input: everybody has an initial value
- Agreement: everybody must decide on the same value

Start



Finish

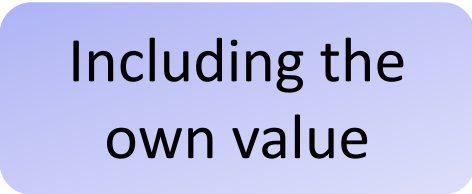


- Validity condition: If everybody starts with the same value, everybody must decide on that value

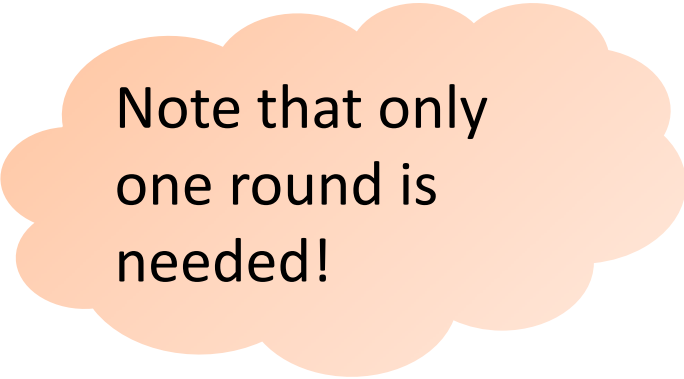
# A Simple Consensus Algorithm

## Each process:

1. Broadcast own value
2. Decide on the minimum of all received values



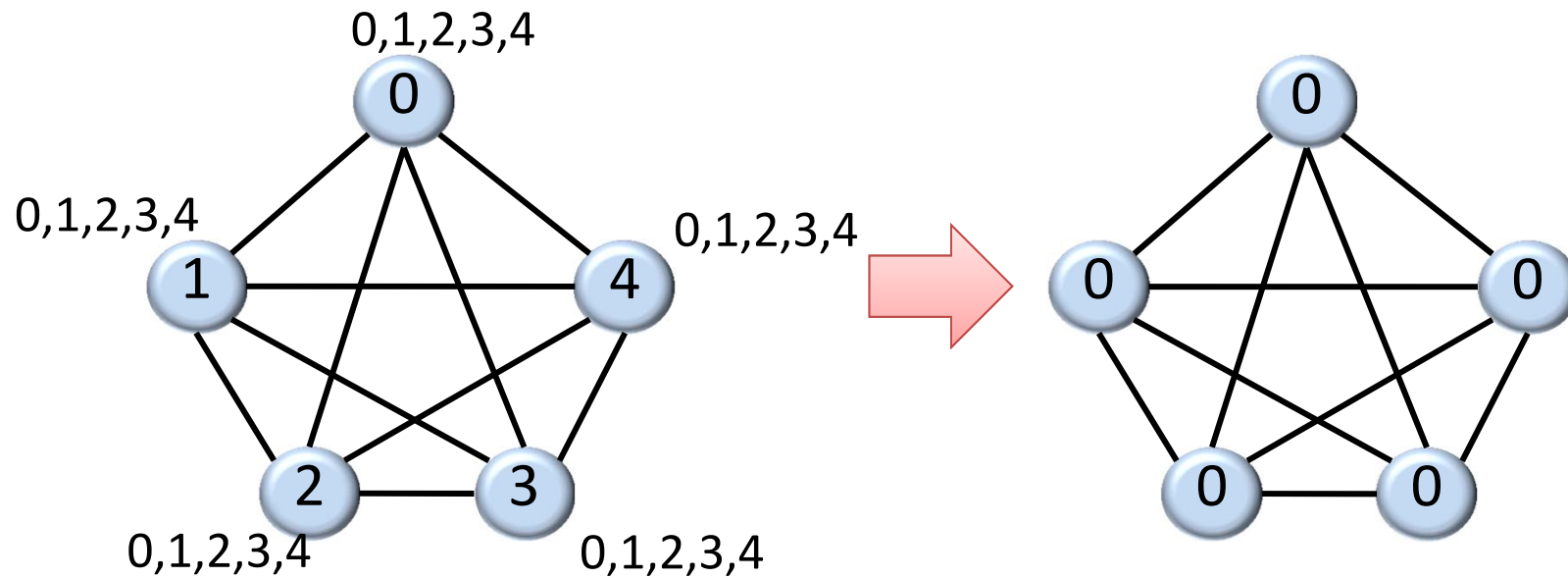
Including the  
own value



Note that only  
one round is  
needed!

# Execution Without Failures

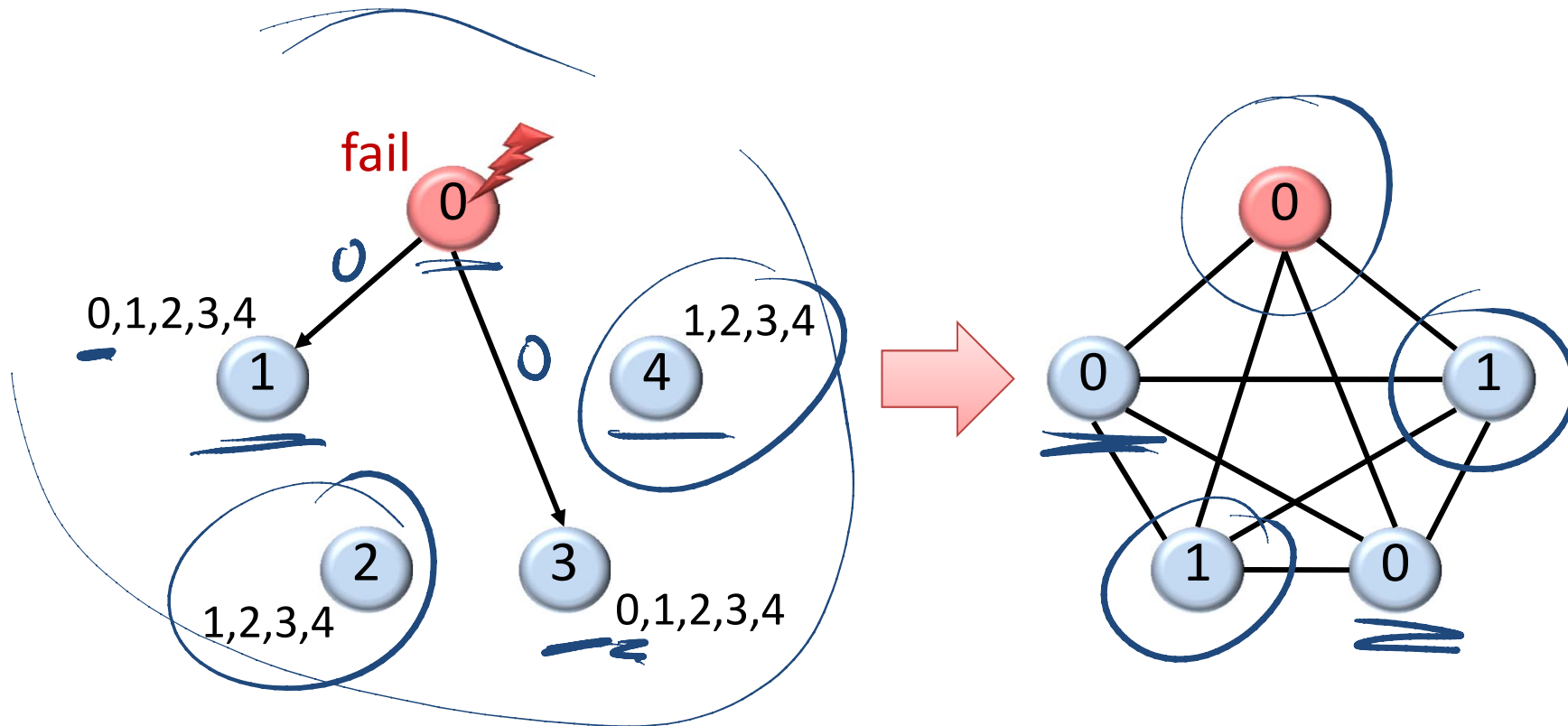
- Broadcast values and decide on minimum → Consensus!
- Validity condition is satisfied: If everybody starts with the same initial value, everybody sticks to that value (minimum)





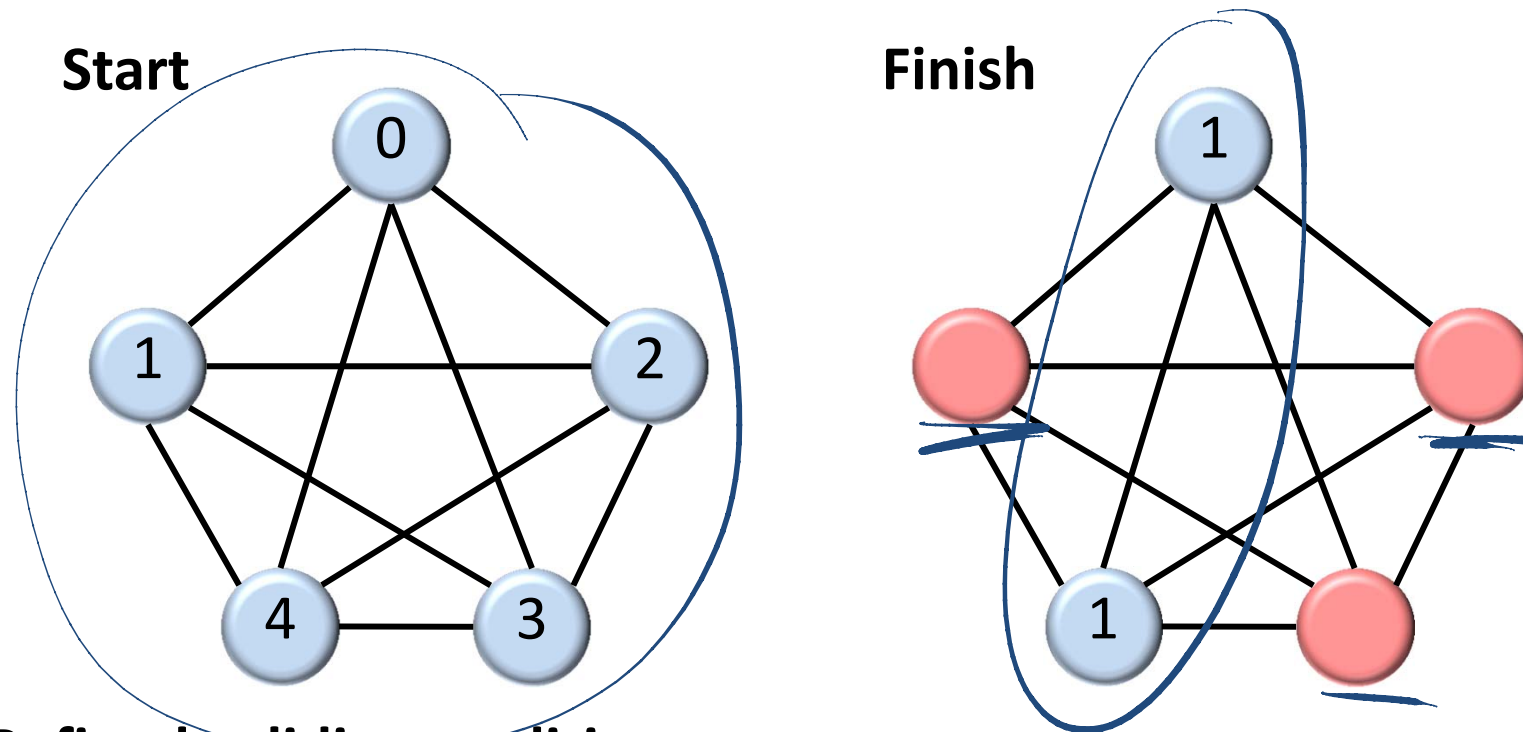
# Execution With Failures

- The failed processor doesn't broadcast its value to all processors
- Decide on minimum  $\rightarrow$  No consensus!



# $f$ -Resilient Consensus Algorithm

- If an algorithm solves consensus for  $f$  failed processes, we say it is an  $f$ -resilient consensus algorithm
- Example: The input and output of a 3-resilient consensus alg.



- **Refined validity condition:**  
All processes decide on a value that is available initially

# An $f$ -Resilient Consensus Algorithm



**Each process:**

**Round 1:**

Broadcast own value

**Round 2 to round  $f + 1$ :**

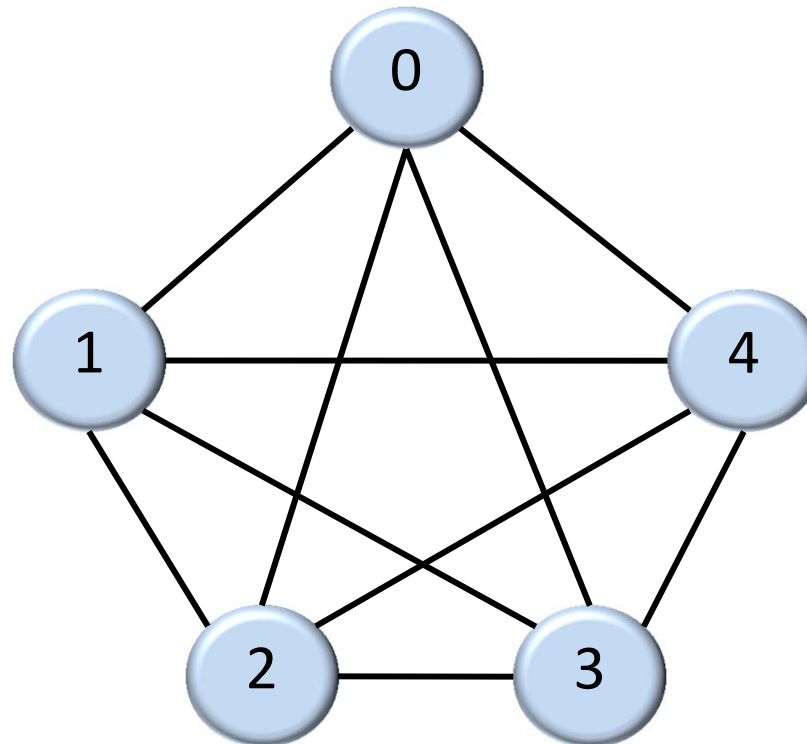
Broadcast the minimum of the received values  
unless it has been sent before

**End of round  $f + 1$ :**

Decide on the minimum value received

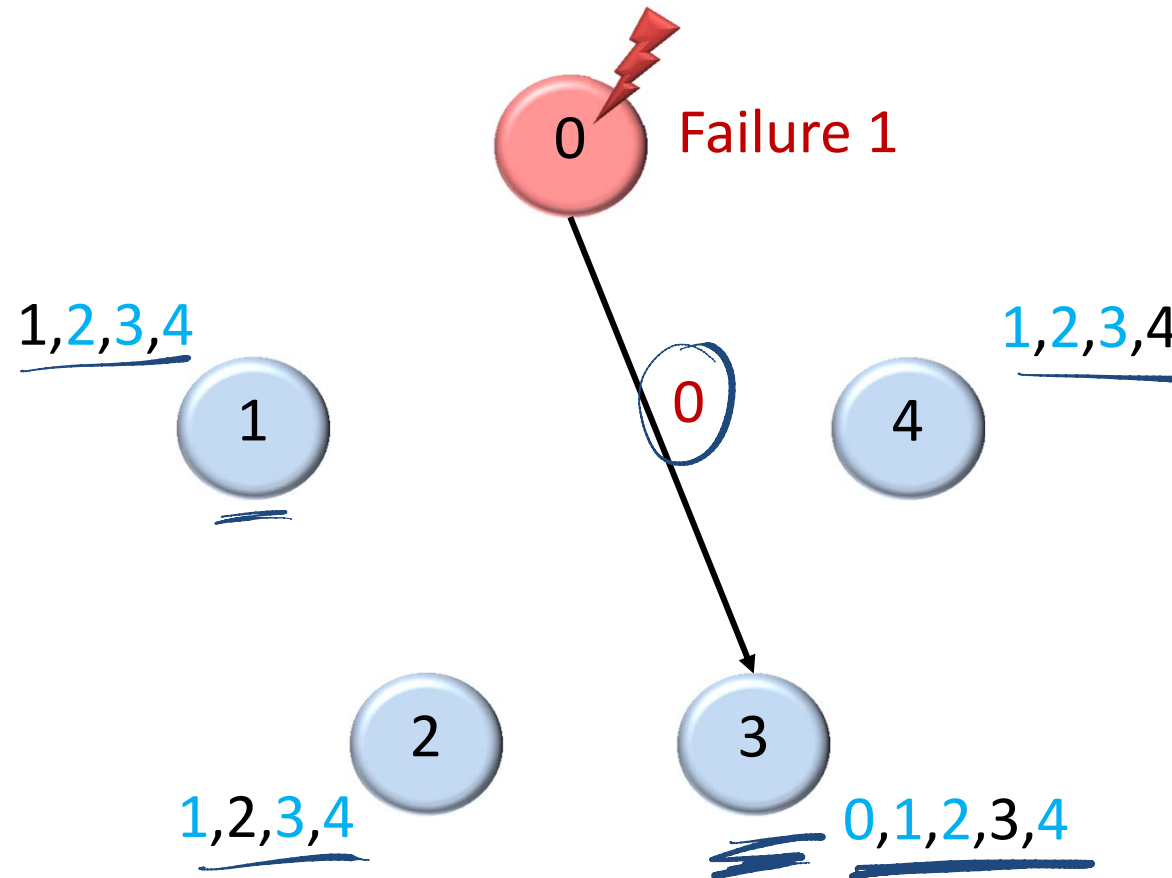
# An $f$ -Resilient Consensus Algorithm

- Example:  $f = 2$  failures,  $f + 1 = 3$  rounds needed



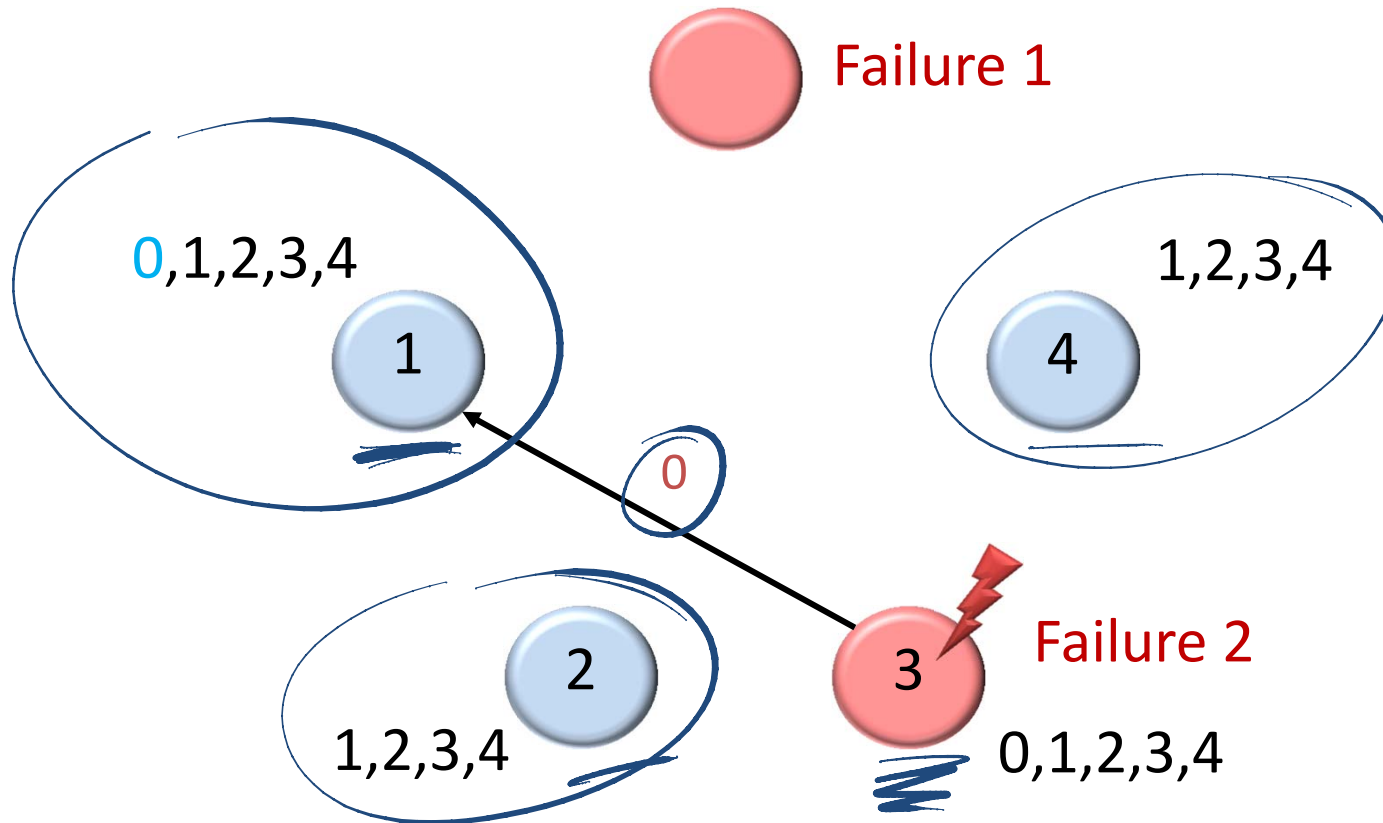
# An $f$ -Resilient Consensus Algorithm

- Round 1: Broadcast all values to everybody



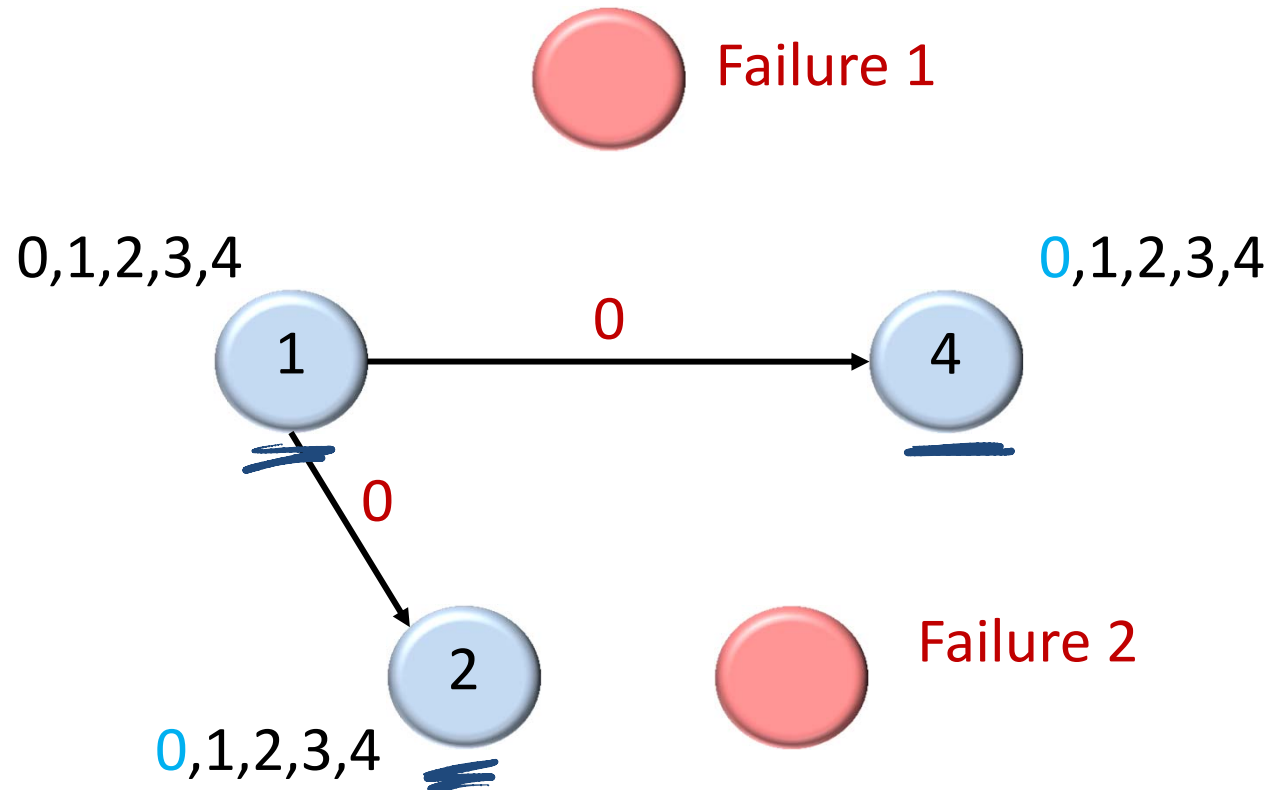
# An $f$ -Resilient Consensus Algorithm

- Round 2: Broadcast all new values to everybody



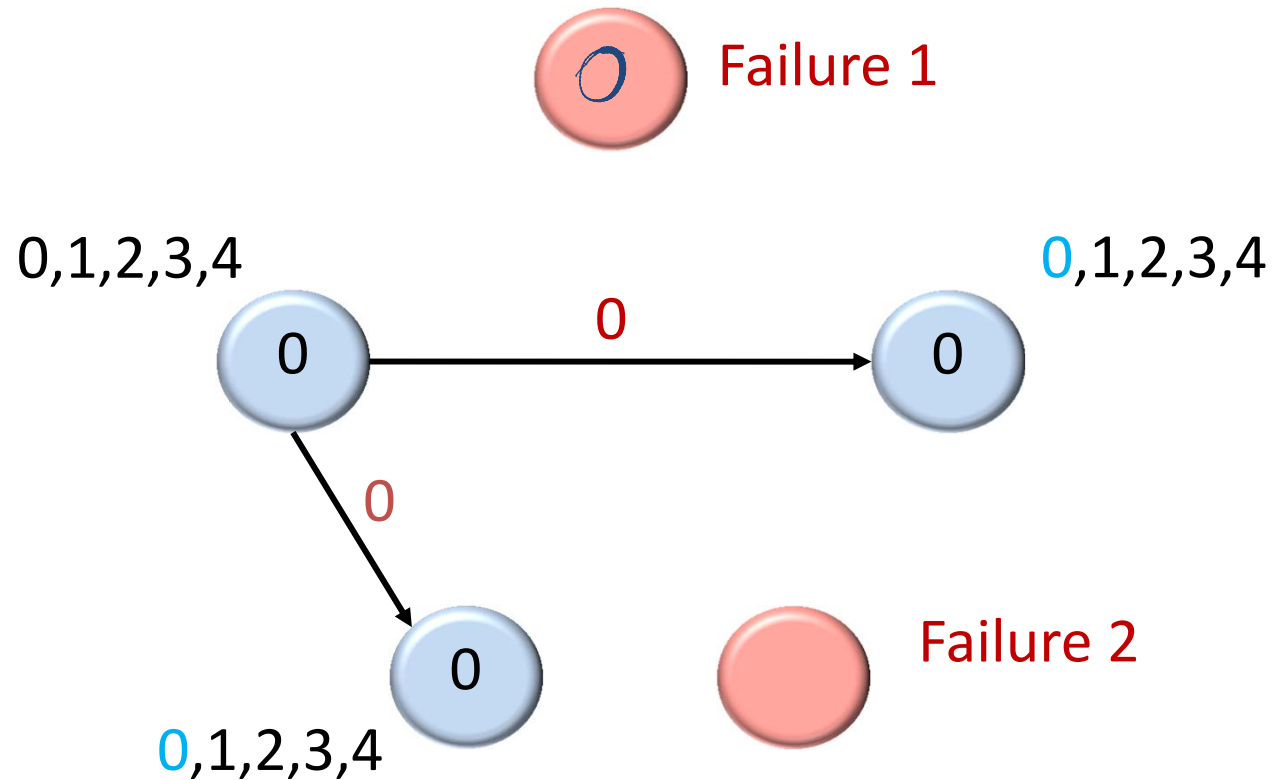
# An $f$ -Resilient Consensus Algorithm

- Round 3: Broadcast all new values to everybody



# An $f$ -Resilient Consensus Algorithm

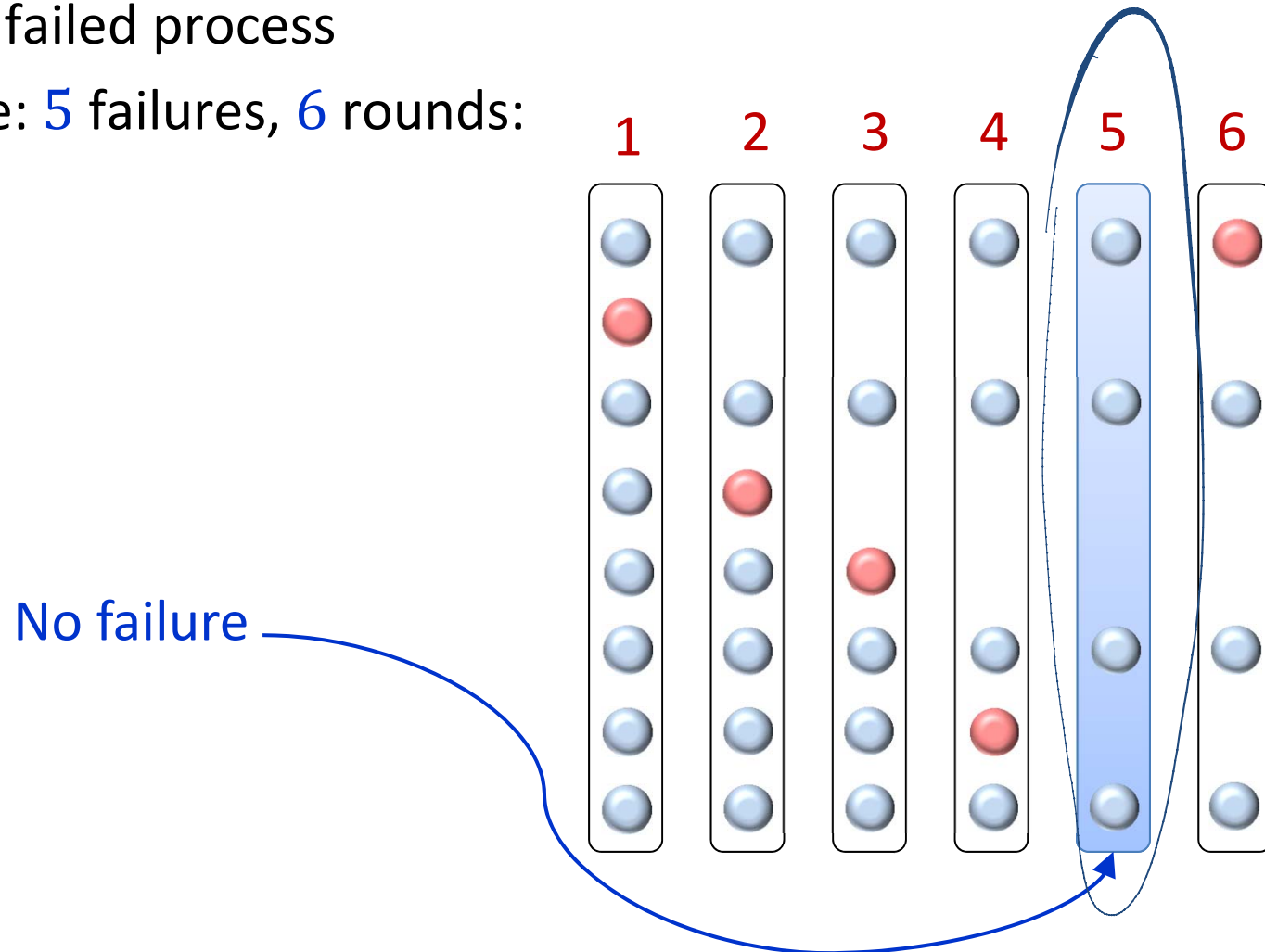
- Decide on minimum  $\rightarrow$  Consensus!





# Analysis

- If there are  $f$  failures and  $f + 1$  rounds, then there is a round with no failed process
- Example: 5 failures, 6 rounds:



# Analysis

---



- At the end of the round with no failure
  - Every (non faulty) process knows about all the values of all the other participating processes
  - This knowledge doesn't change until the end of the algorithm
- Therefore, everybody will decide on the same value
- However, as we don't know the exact position of this round, we have to let the algorithm execute for  $f + 1$  rounds
- **Validity:** When all processes start with the same input value, then consensus is that value

# Theorem

$$n \geq f + 2$$

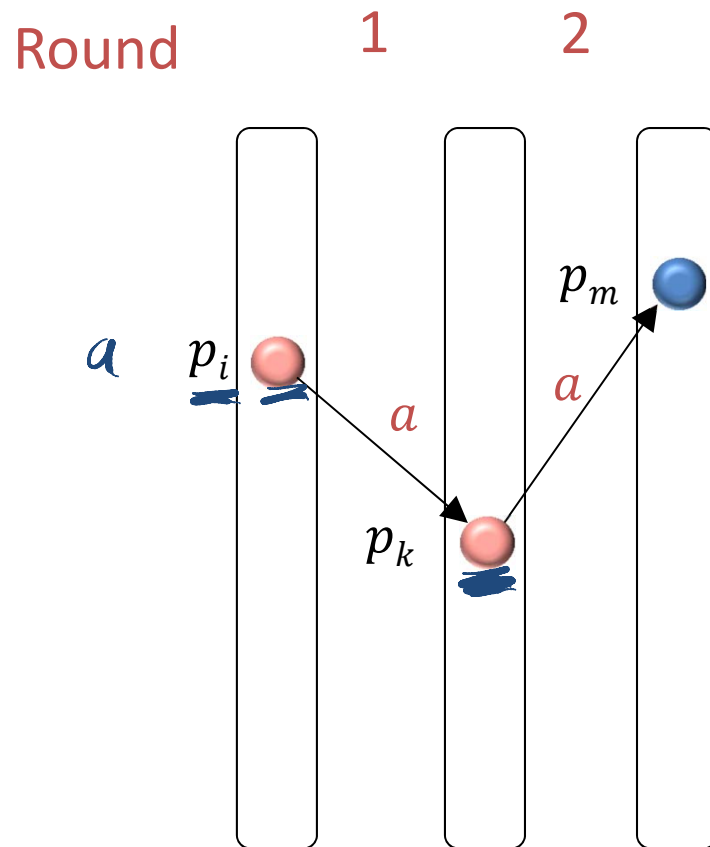
## Theorem

If at most  $f \leq n - 2$  of  $n$  nodes of a synchronous message passing system can crash, at least  $f + 1$  rounds are needed to solve consensus.

### Proof idea:

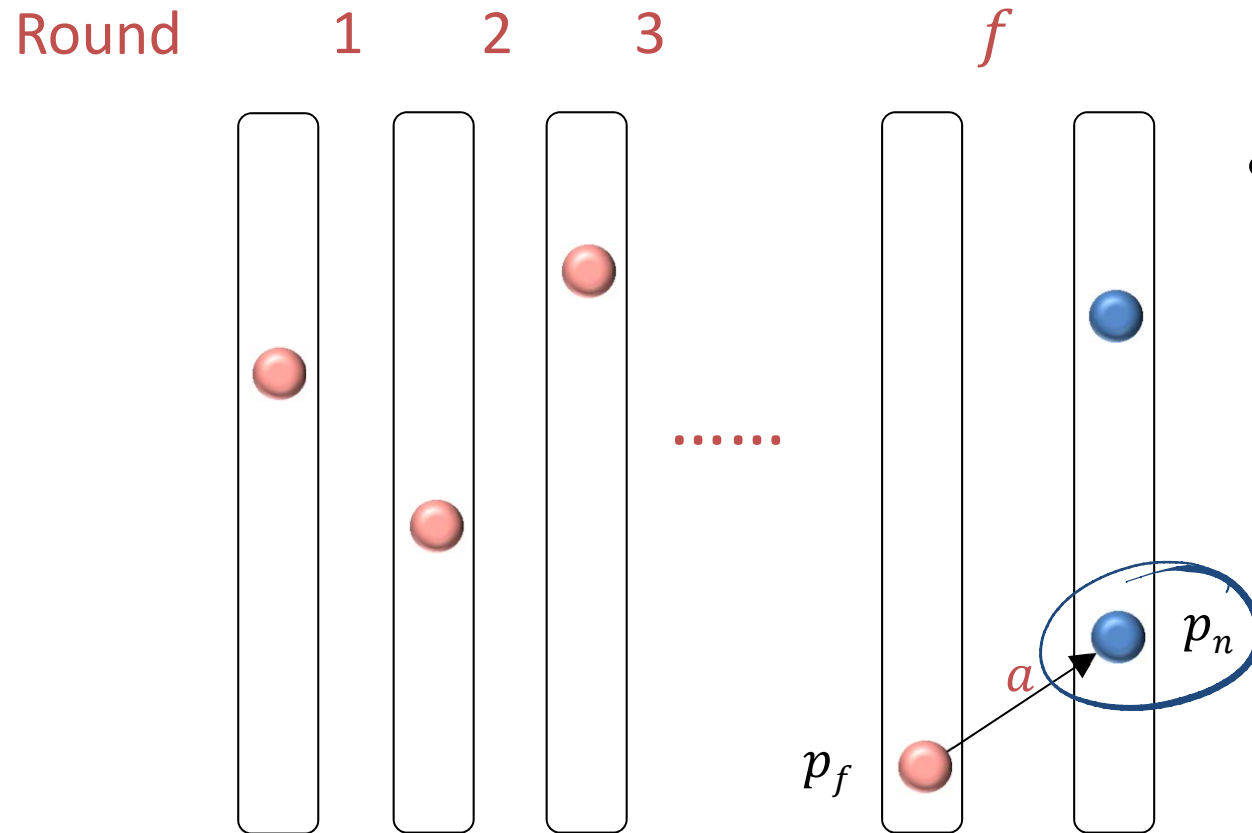
- Show that  $f$  rounds are not enough if  $n \geq f + 2$
- Before proving the theorem, we consider a  
“worst-case scenario”: In each round one of the processes fails

# Lower Bound on Rounds: Intuition



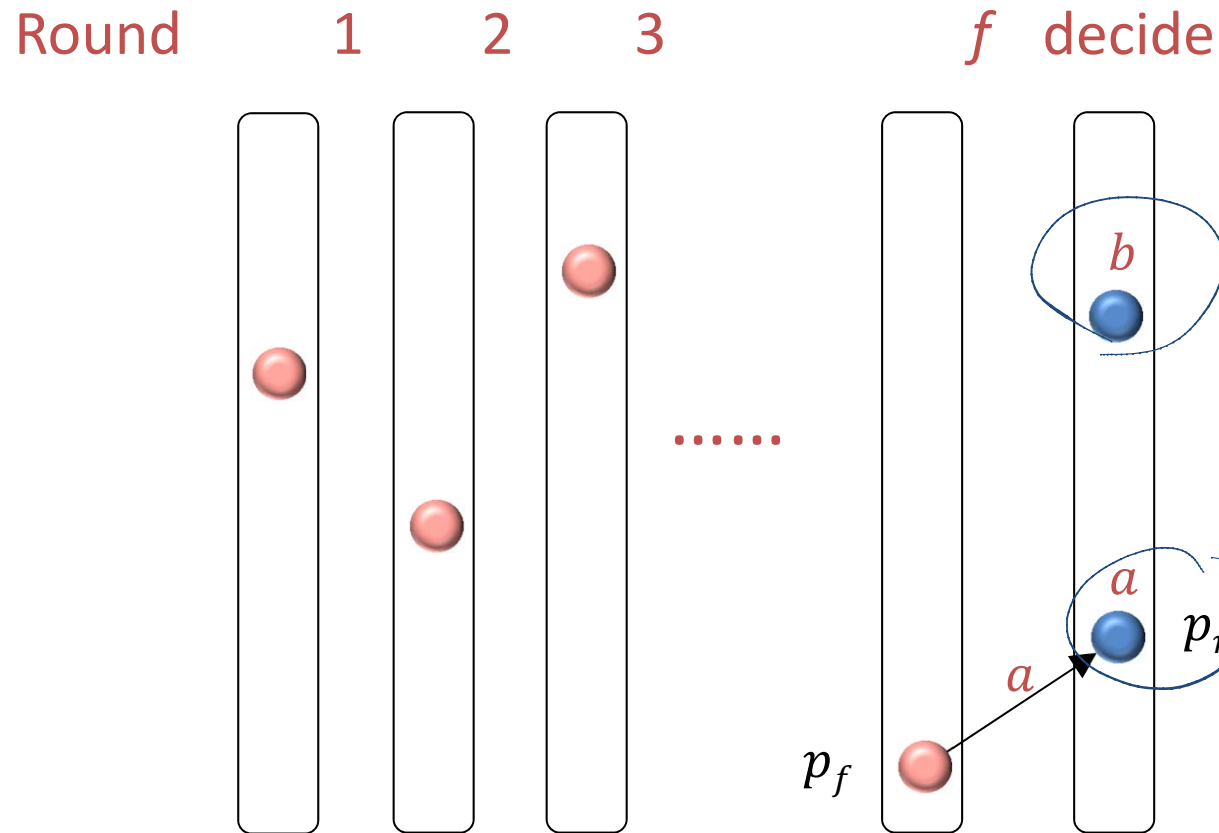
- Before process  $p_i$  fails, it sends its value  $a$  only to one process  $p_k$
- Before process  $p_k$  fails, it sends its value  $a$  to only one process  $p_m$

# Lower Bound on Rounds: Intuition



- At the end of round  $f$  only one process  $p_n$  knows about value  $a$

# Lower Bound on Rounds: Intuition



- Process  $p_n$  may decide on  $a$  and all other processes may decide on another value  $b$
- $f$  rounds are not enough  
 $\Rightarrow$  at least  $f + 1$  rounds are needed

# Lower Bound on Rounds: Proof

## Recall (from Chapters 1 & 2):

- For the impossibility proof of the two generals problem, we used an indistinguishability proof
- Execution  $E$  is indistinguishable from execution  $E'$  for some node  $v$  if  $v$  sees the same things in both executions.
  - same inputs and messages (schedule)
- If  $E$  is indistinguishable from  $E'$  for  $v$ , then  $v$  does the same thing in both executions.
  - We denoted this by  $E|v = E'|v$   $E|v$

## Similarity:

- Call  $E_i$  and  $E_j$  **similar** if  $E_i|v = E_j|v$  for some node  $v$

$$E_i \sim_v E_j \Leftrightarrow E_i|v = E_j|v$$

# Lower Bound on Rounds: Proof

## Similarity Chain:

$$E_2 \uparrow v_2 = E_3 \uparrow v_2$$

- Consider a sequence of executions  $E_1, E_2, E_3, \dots, E_T$  such that

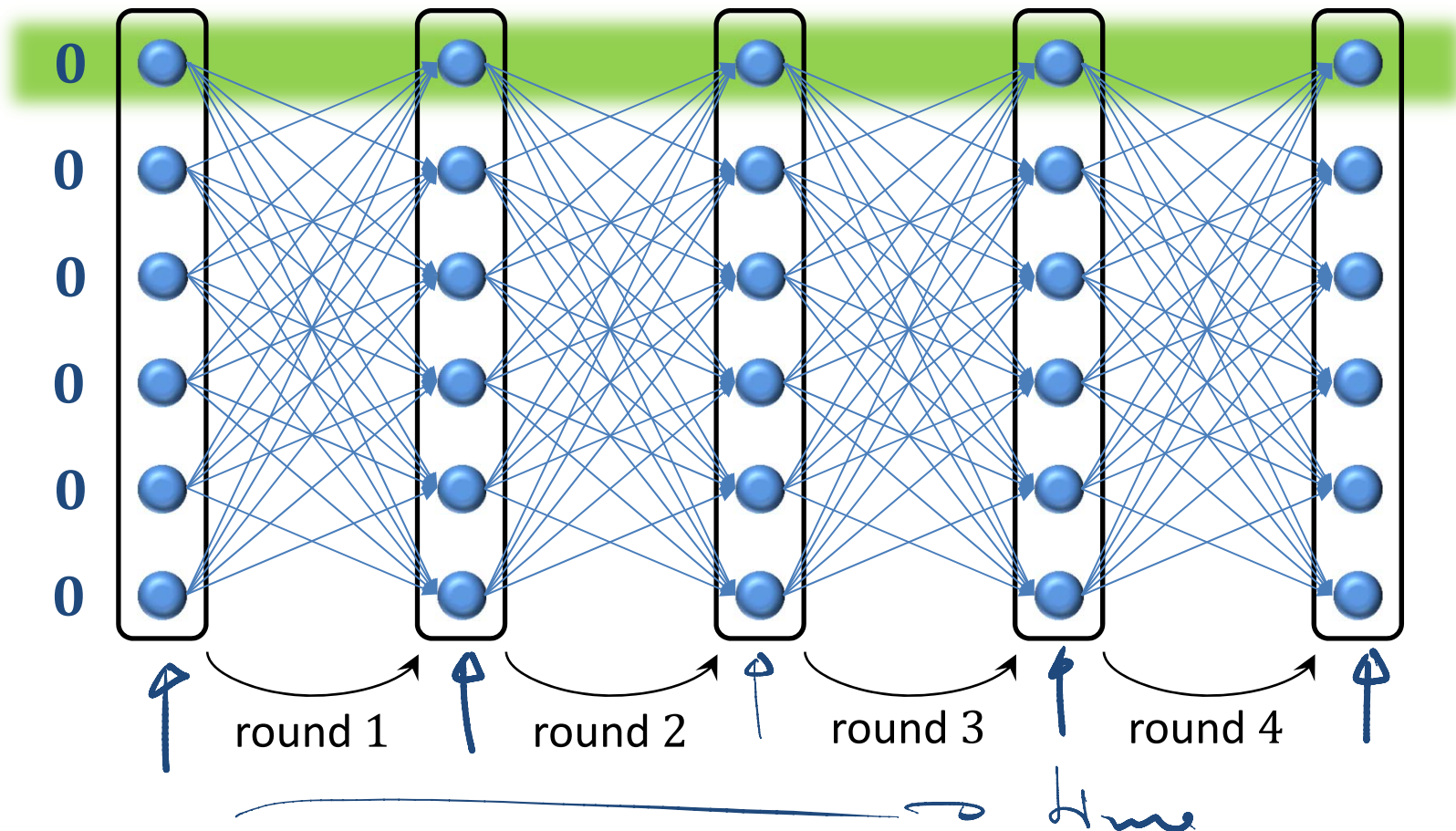
$$\forall i \geq 1 : E_i \sim_{v_i} E_{i+1}$$

- any two consecutive executions  $E_i$  and  $E_{i+1}$  are indistinguishable for some node  $v_i$  (we assume that  $v_i$  does not crash in  $E_i$  and  $E_{i+1}$ )
- Indistinguishability:**  
 $\forall i \geq 1 : \underline{v_i}$  decides on the same value in  $E_i$  and  $E_{i+1}$
- Agreement:**  
 $\forall i \geq 1 : \text{All nodes decide on the same value in } E_i \text{ and } E_{i+1}$
- Hence, **all executions  $E_1, \dots, E_T$  have the same decision value!**
- Goal:**  
 $E_1$ : no crashes, all inputs are 0;  $E_T$ : no crashes, all inputs are 1



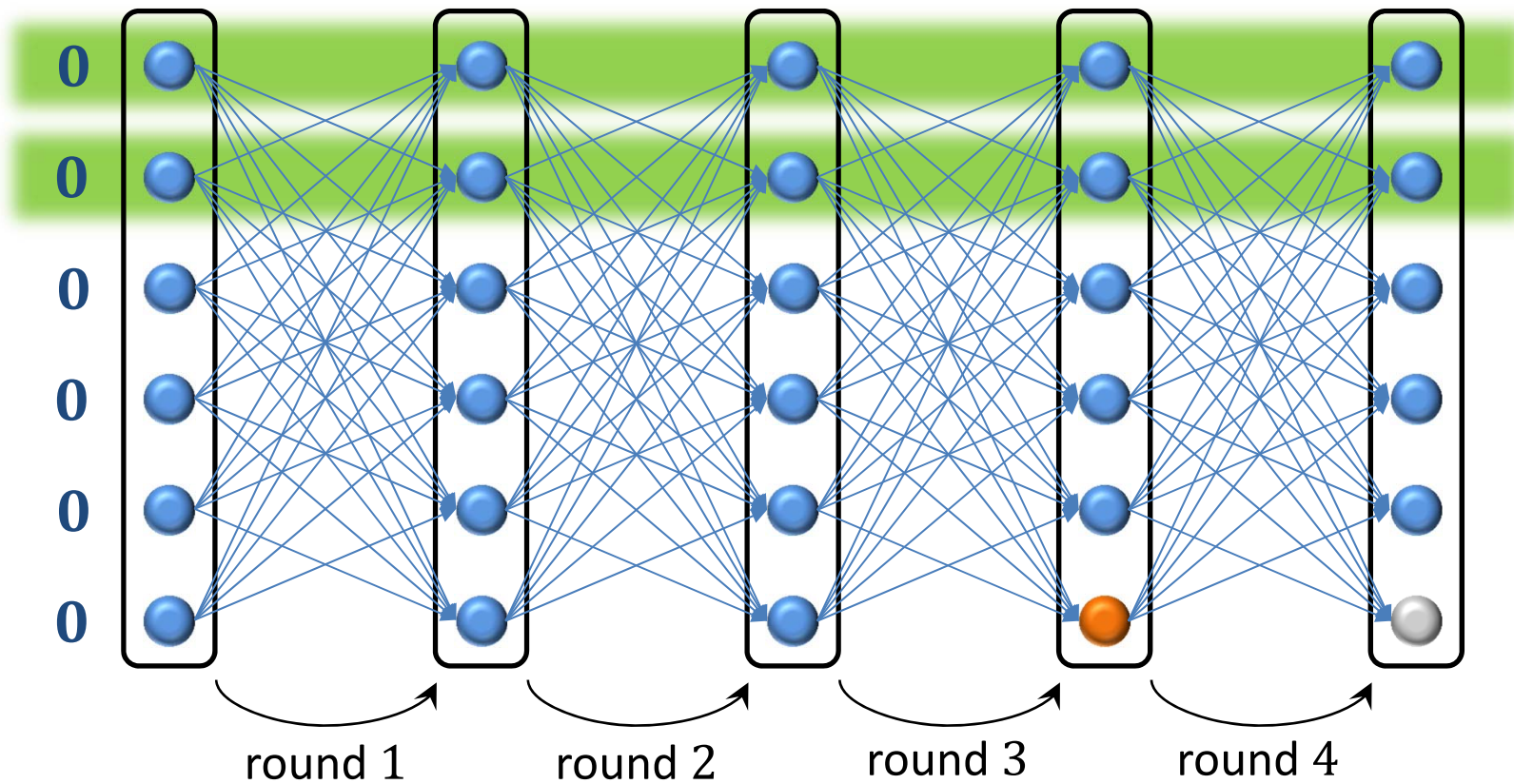
# Lower Bound on Rounds: Proof

Example:  $f = 4$ ,  $n = 6$     Need to show: 4 rounds are not enough



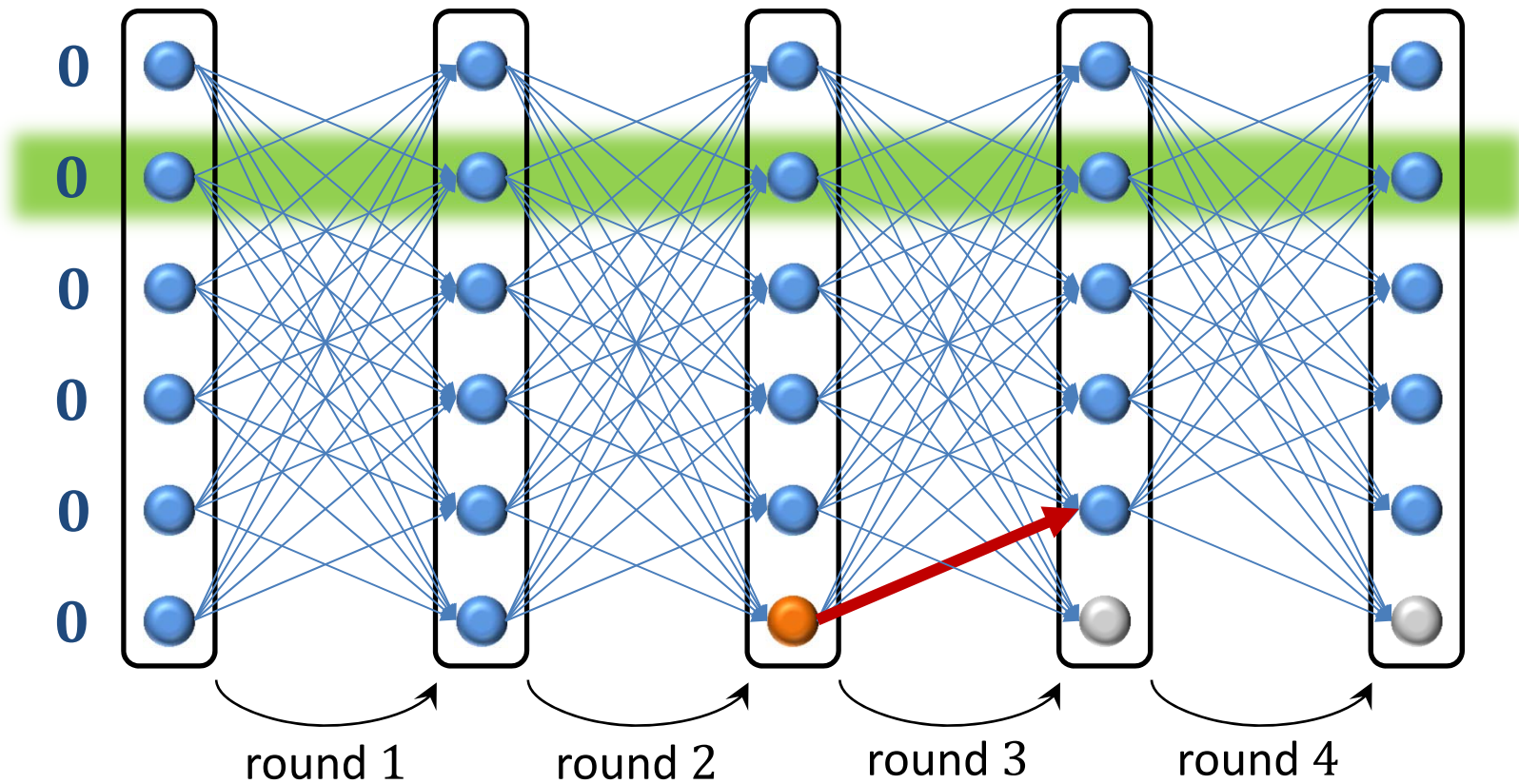
# Lower Bound on Rounds: Proof

Example:  $f = 4, n = 6$     Need to show: **4 rounds are not enough**



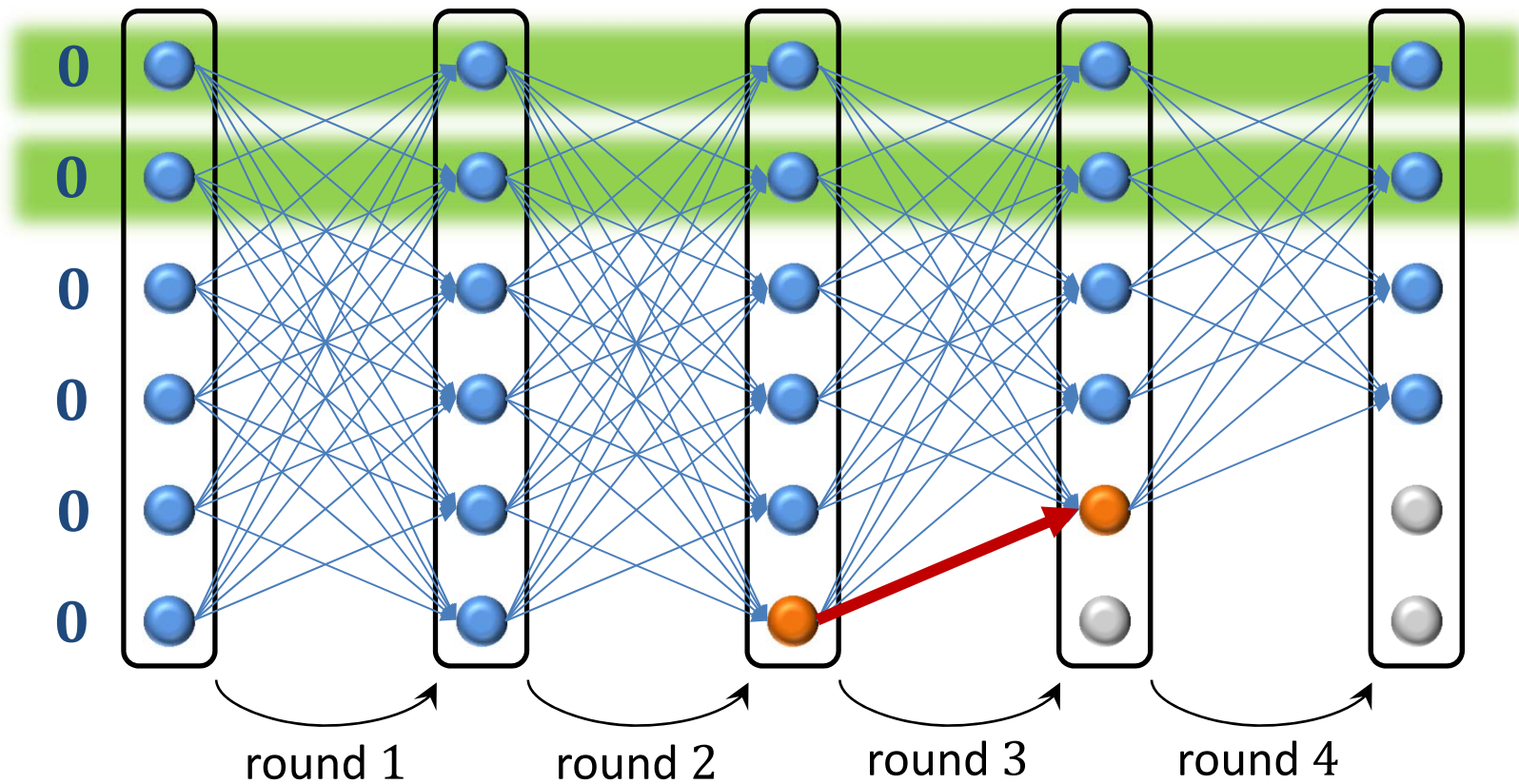
# Lower Bound on Rounds: Proof

Example:  $f = 4, n = 6$     Need to show: **4 rounds are not enough**



# Lower Bound on Rounds: Proof

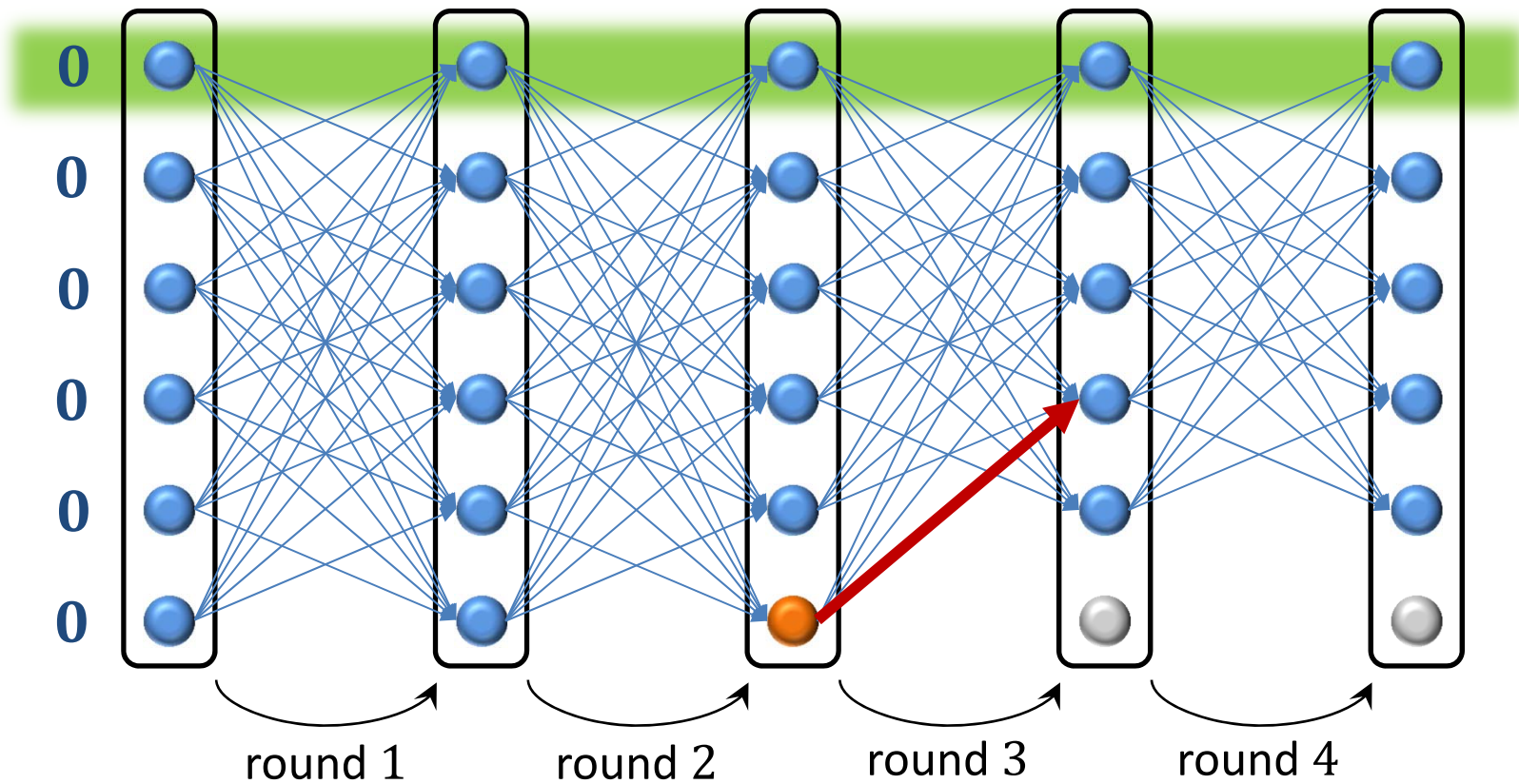
Example:  $f = 4, n = 6$     Need to show: **4 rounds are not enough**





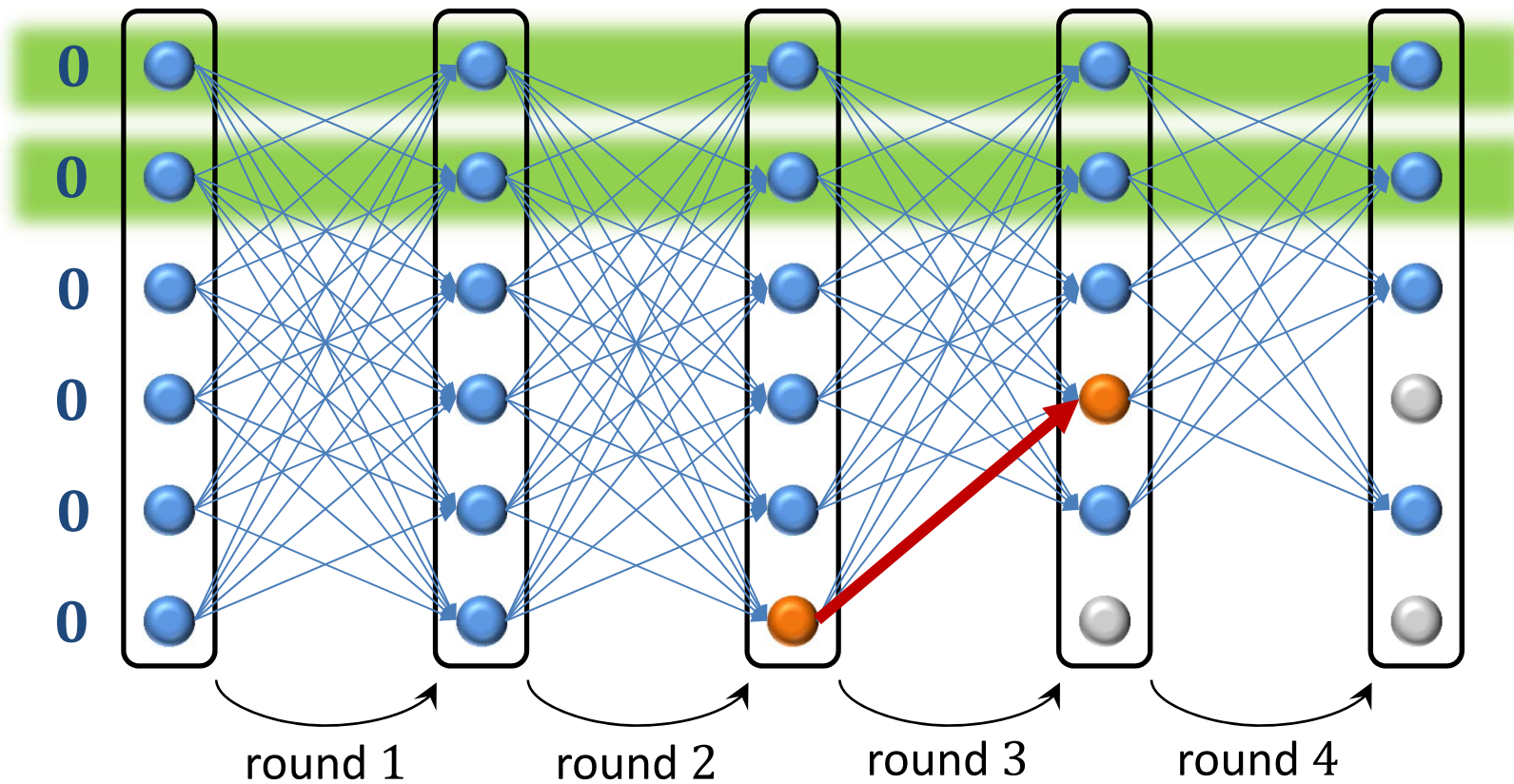
# Lower Bound on Rounds: Proof

Example:  $f = 4, n = 6$     Need to show: **4 rounds are not enough**



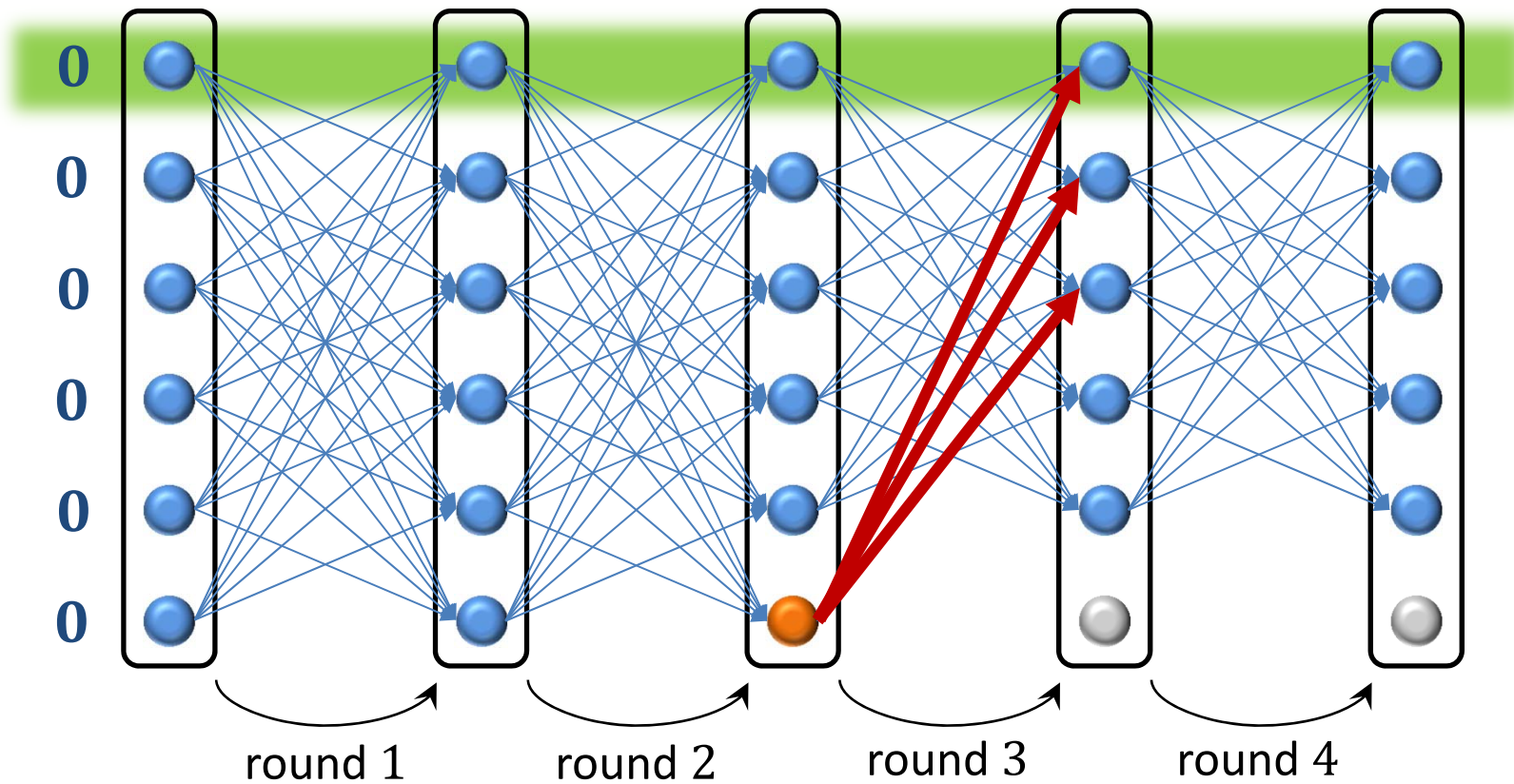
# Lower Bound on Rounds: Proof

Example:  $f = 4, n = 6$     Need to show: **4 rounds are not enough**



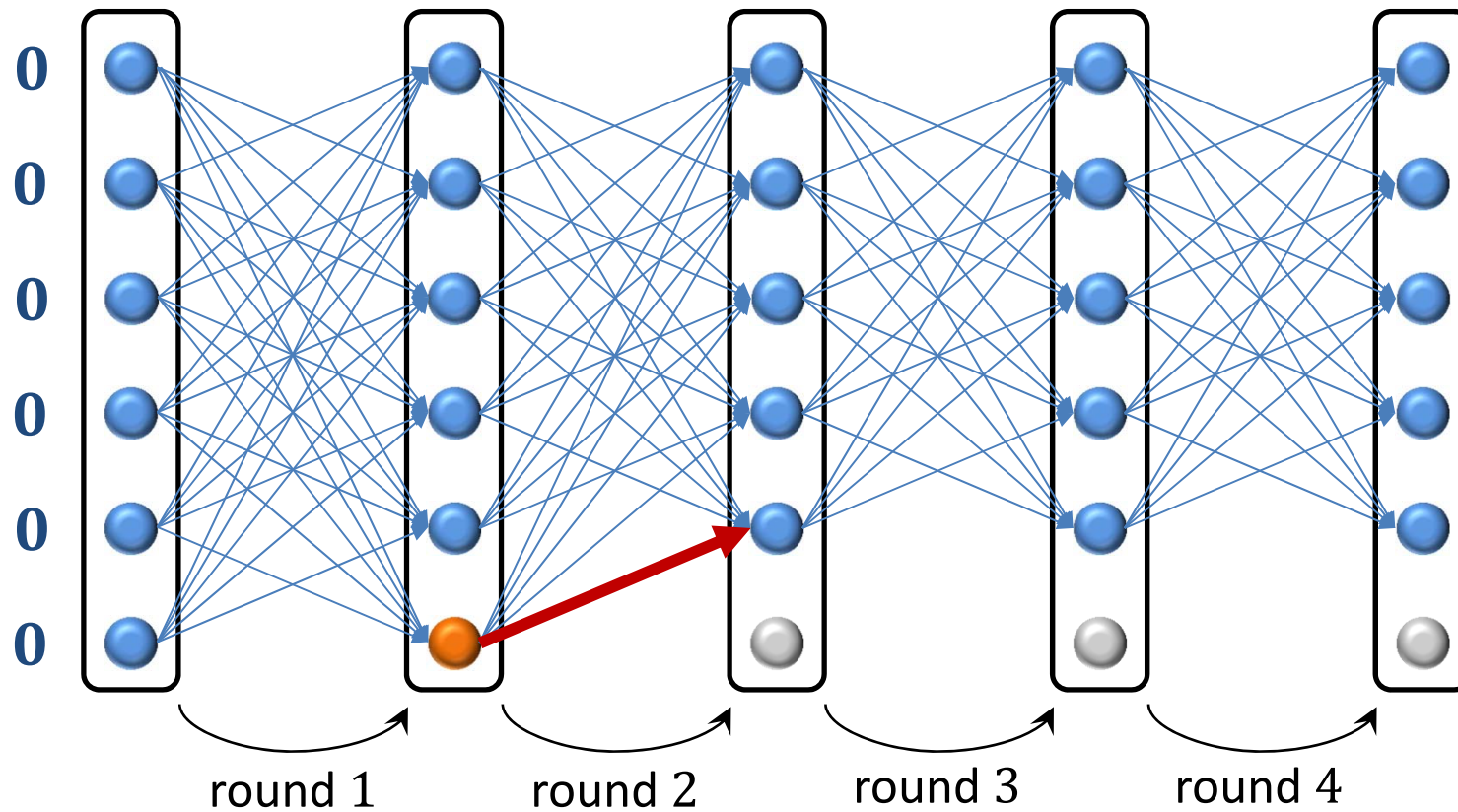
# Lower Bound on Rounds: Proof

Example:  $f = 4, n = 6$     Need to show: **4 rounds are not enough**



# Lower Bound on Rounds: Proof

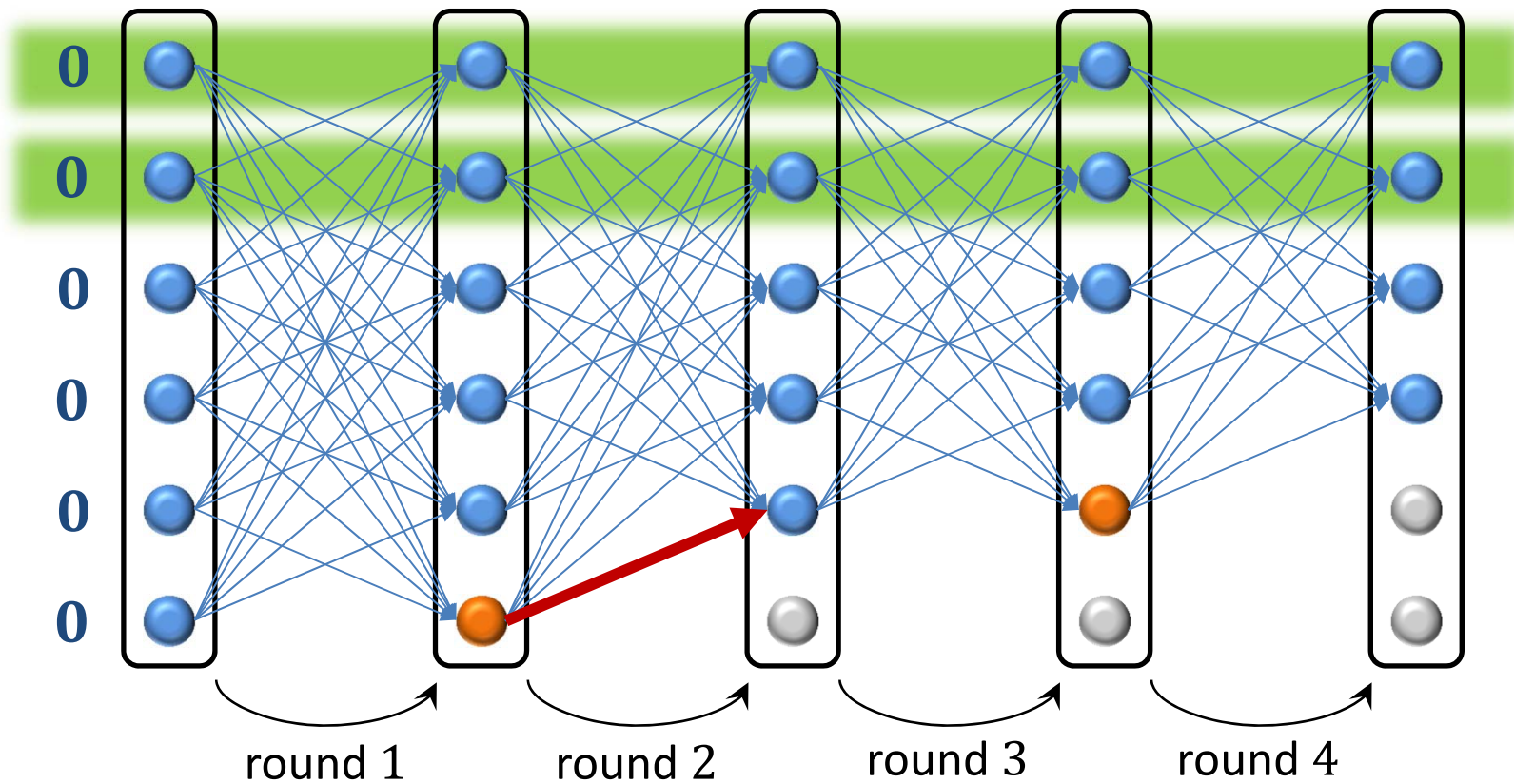
Example:  $f = 4, n = 6$     Need to show: **4 rounds are not enough**





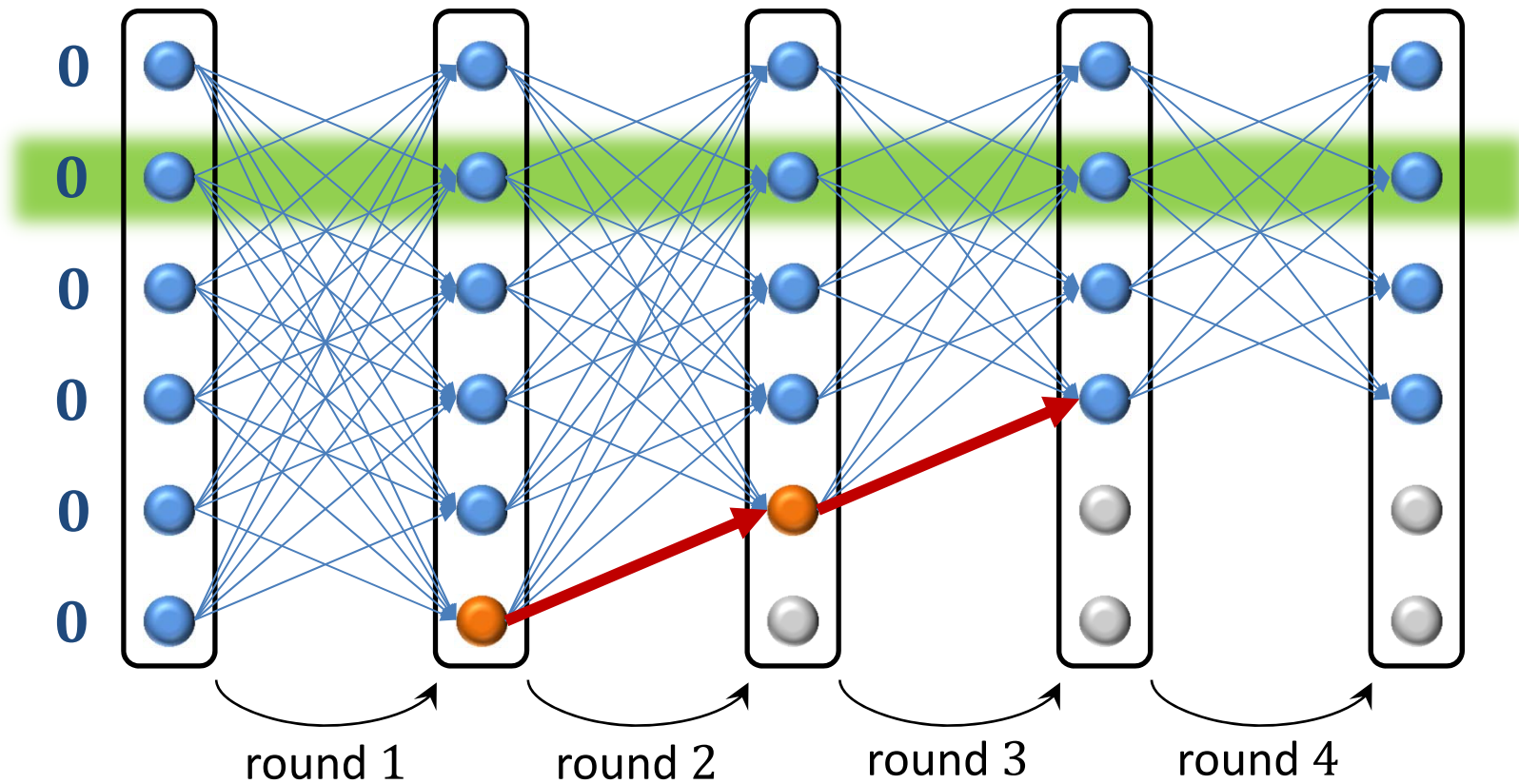
# Lower Bound on Rounds: Proof

Example:  $f = 4, n = 6$     Need to show: **4 rounds are not enough**



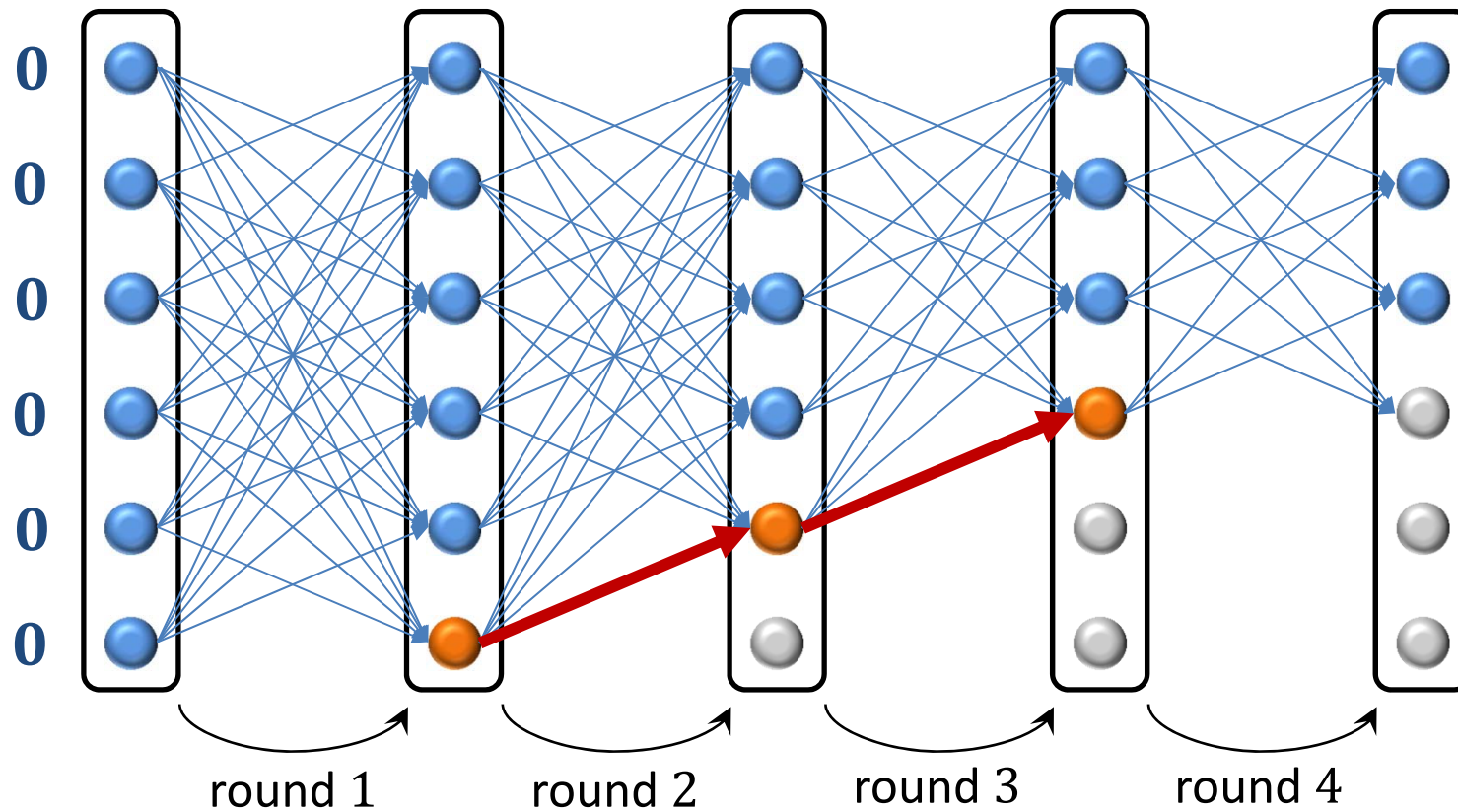
# Lower Bound on Rounds: Proof

Example:  $f = 4, n = 6$     Need to show: **4 rounds are not enough**



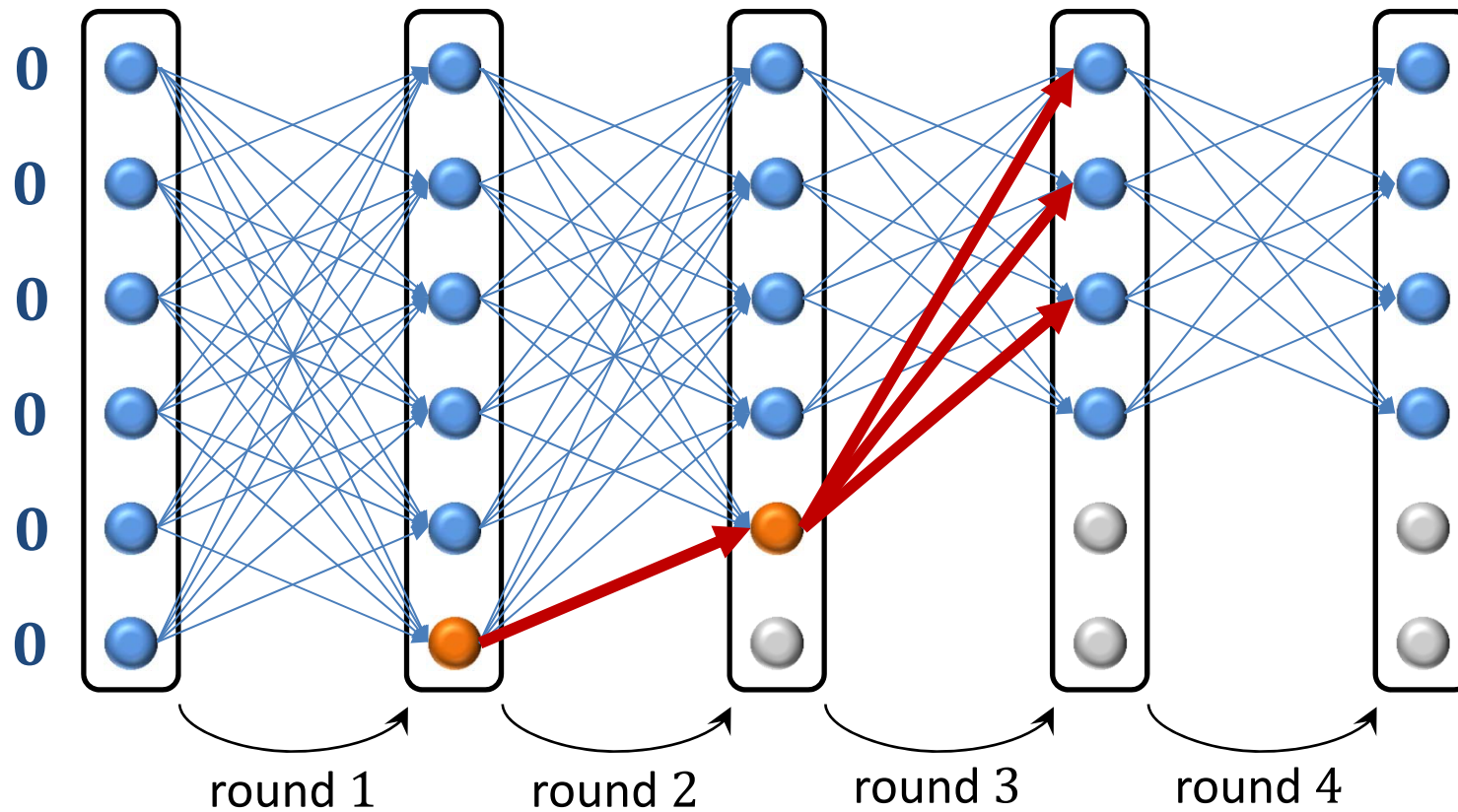
# Lower Bound on Rounds: Proof

Example:  $f = 4, n = 6$     Need to show: **4 rounds are not enough**



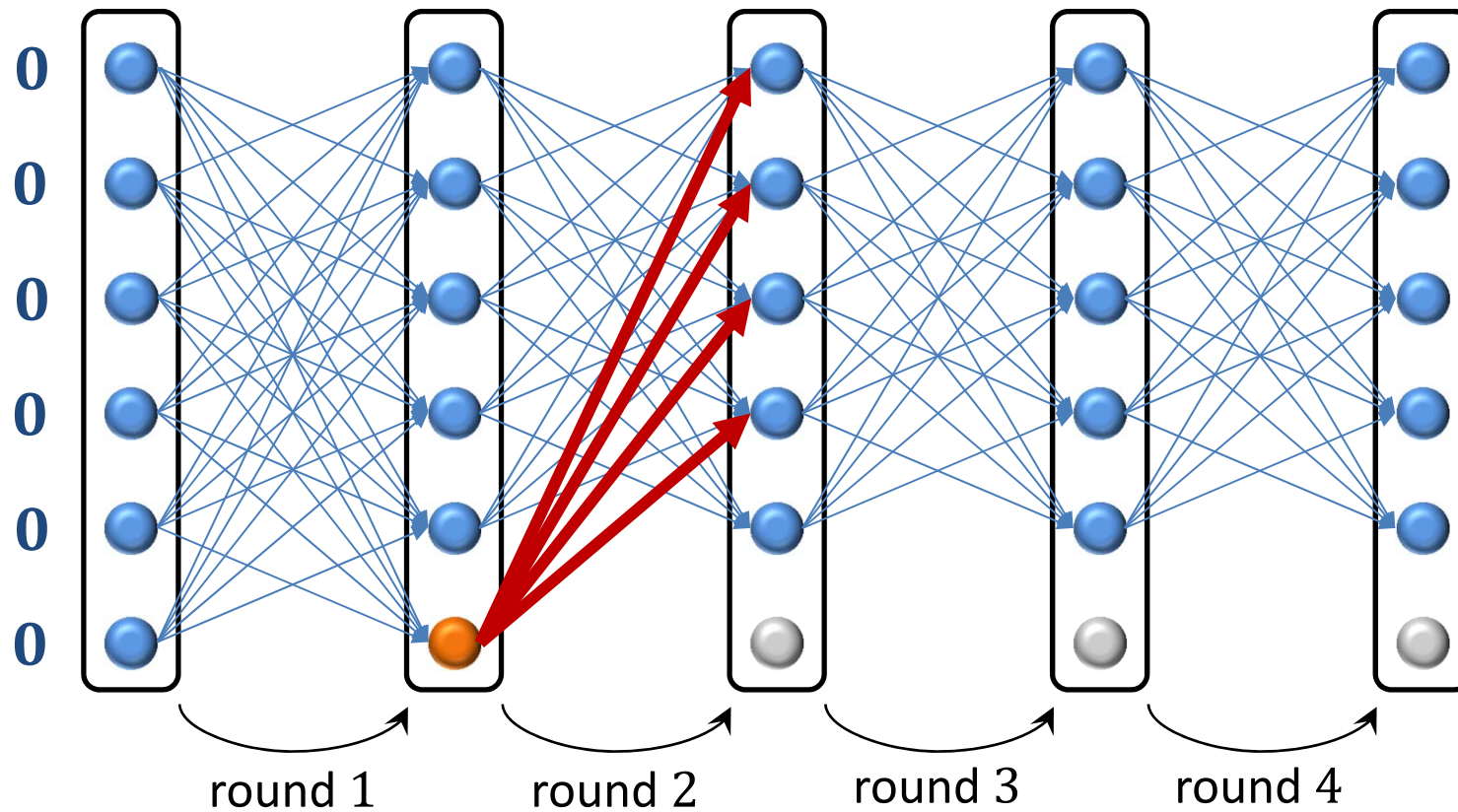
# Lower Bound on Rounds: Proof

Example:  $f = 4, n = 6$     Need to show: **4 rounds are not enough**



# Lower Bound on Rounds: Proof

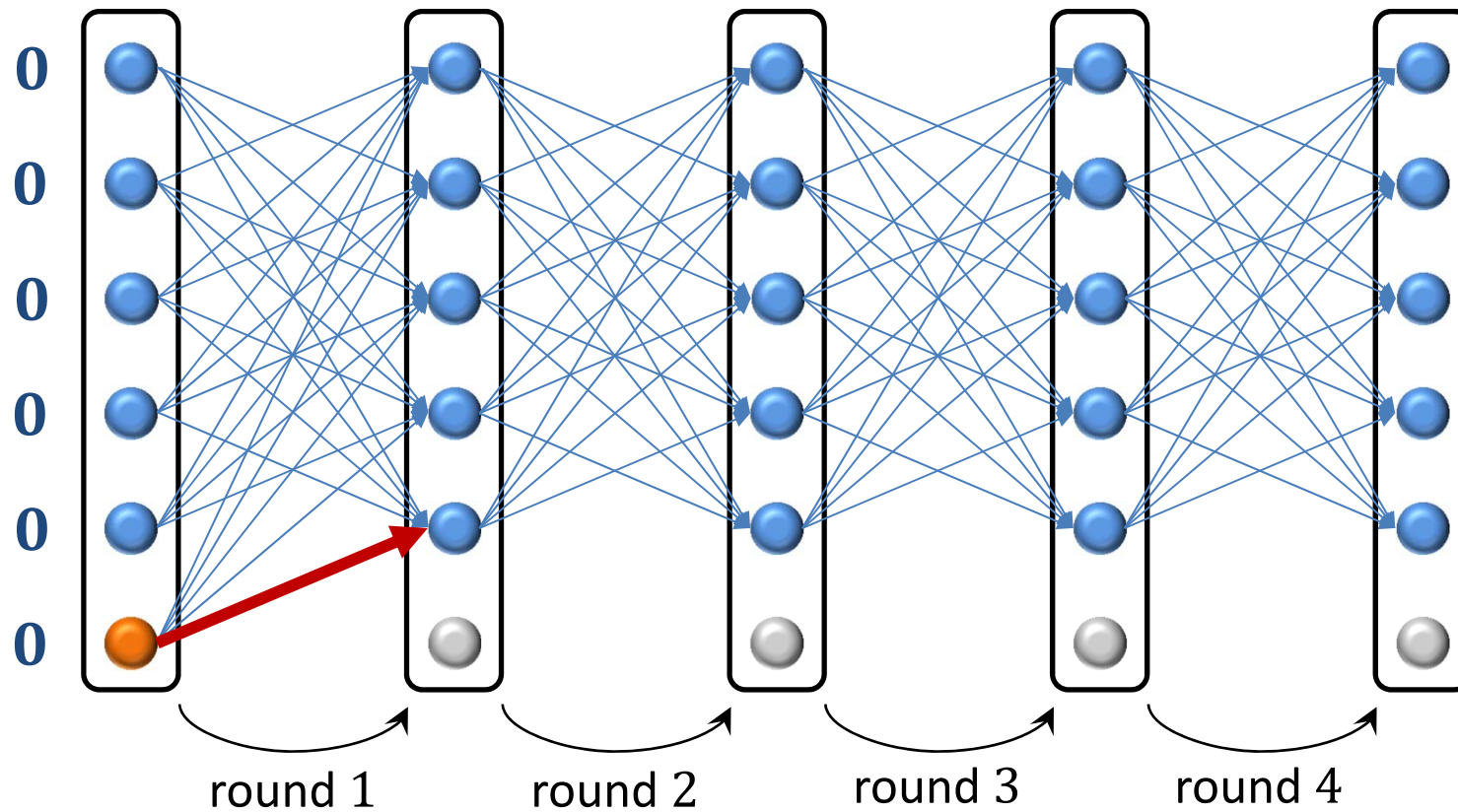
Example:  $f = 4, n = 6$     Need to show: **4 rounds are not enough**





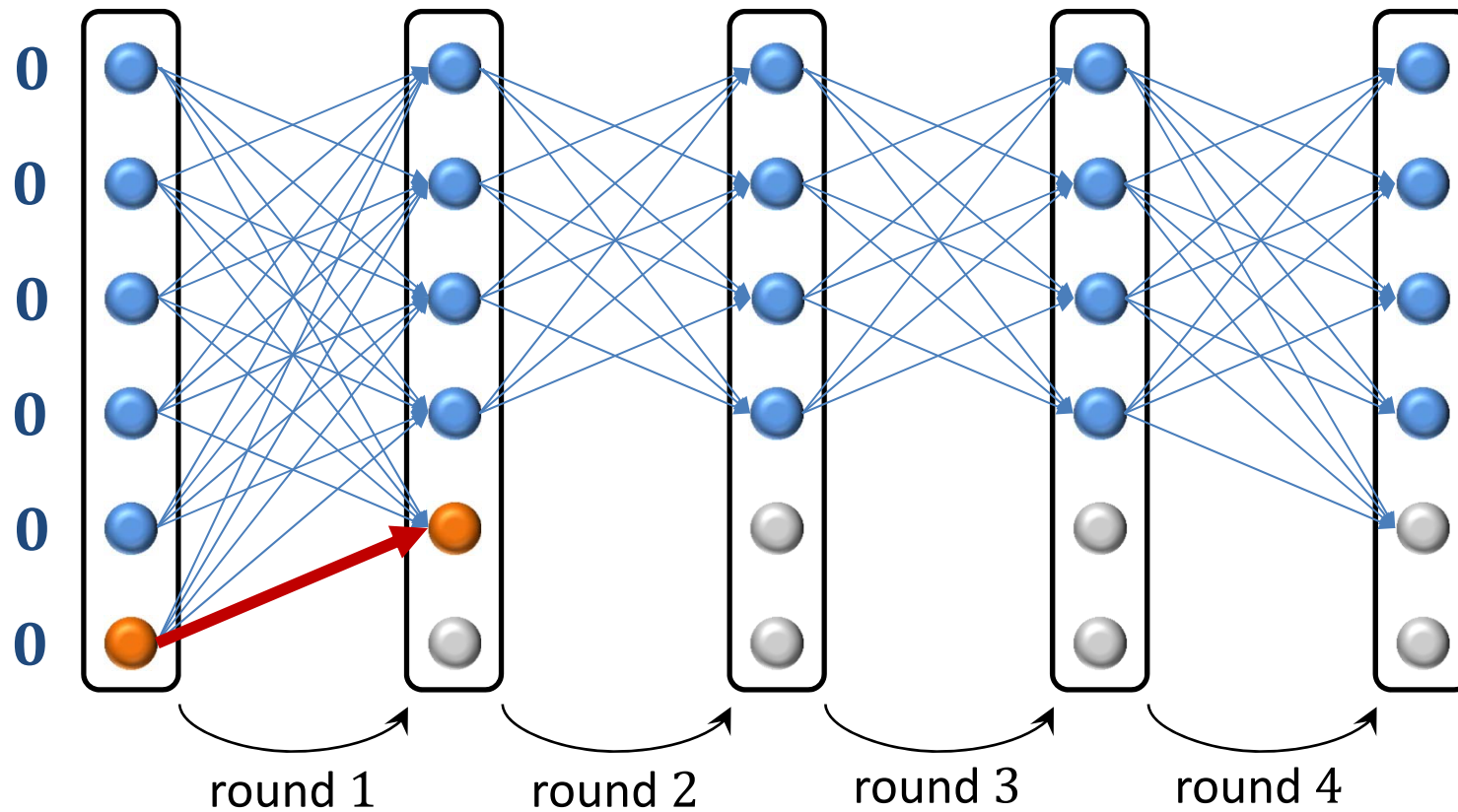
# Lower Bound on Rounds: Proof

Example:  $f = 4, n = 6$     Need to show: **4 rounds are not enough**



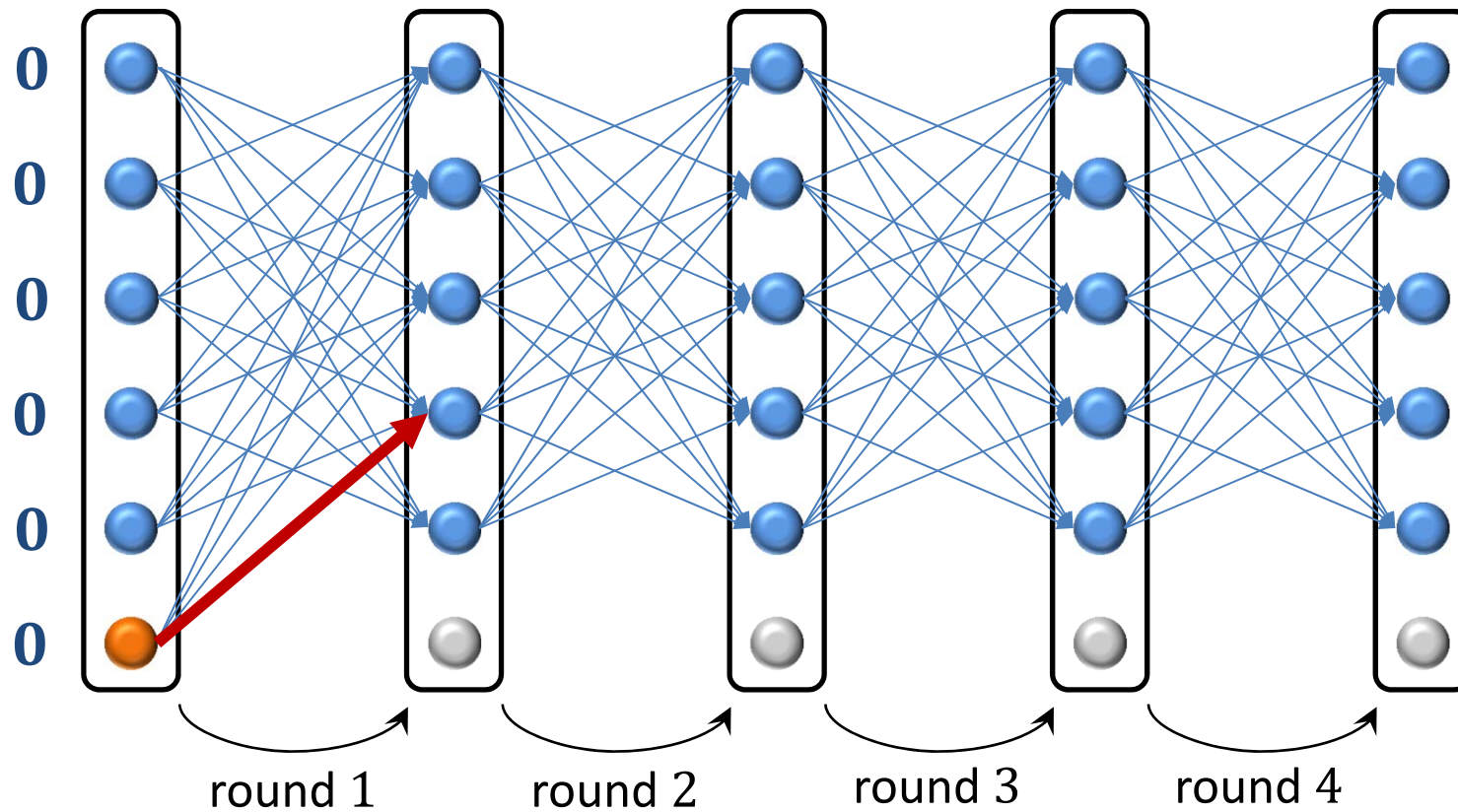
# Lower Bound on Rounds: Proof

Example:  $f = 4, n = 6$     Need to show: **4 rounds are not enough**



# Lower Bound on Rounds: Proof

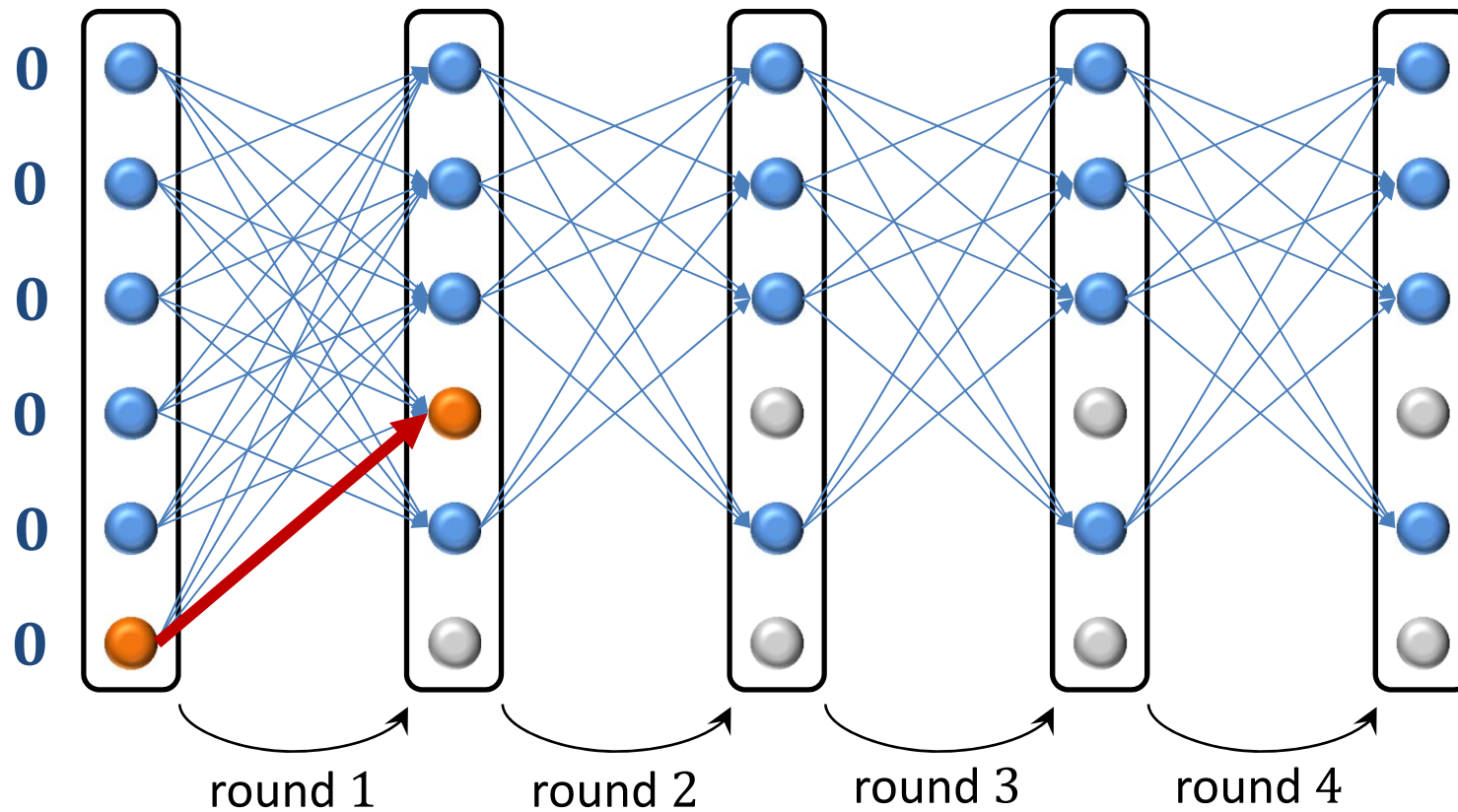
Example:  $f = 4, n = 6$     Need to show: **4 rounds are not enough**





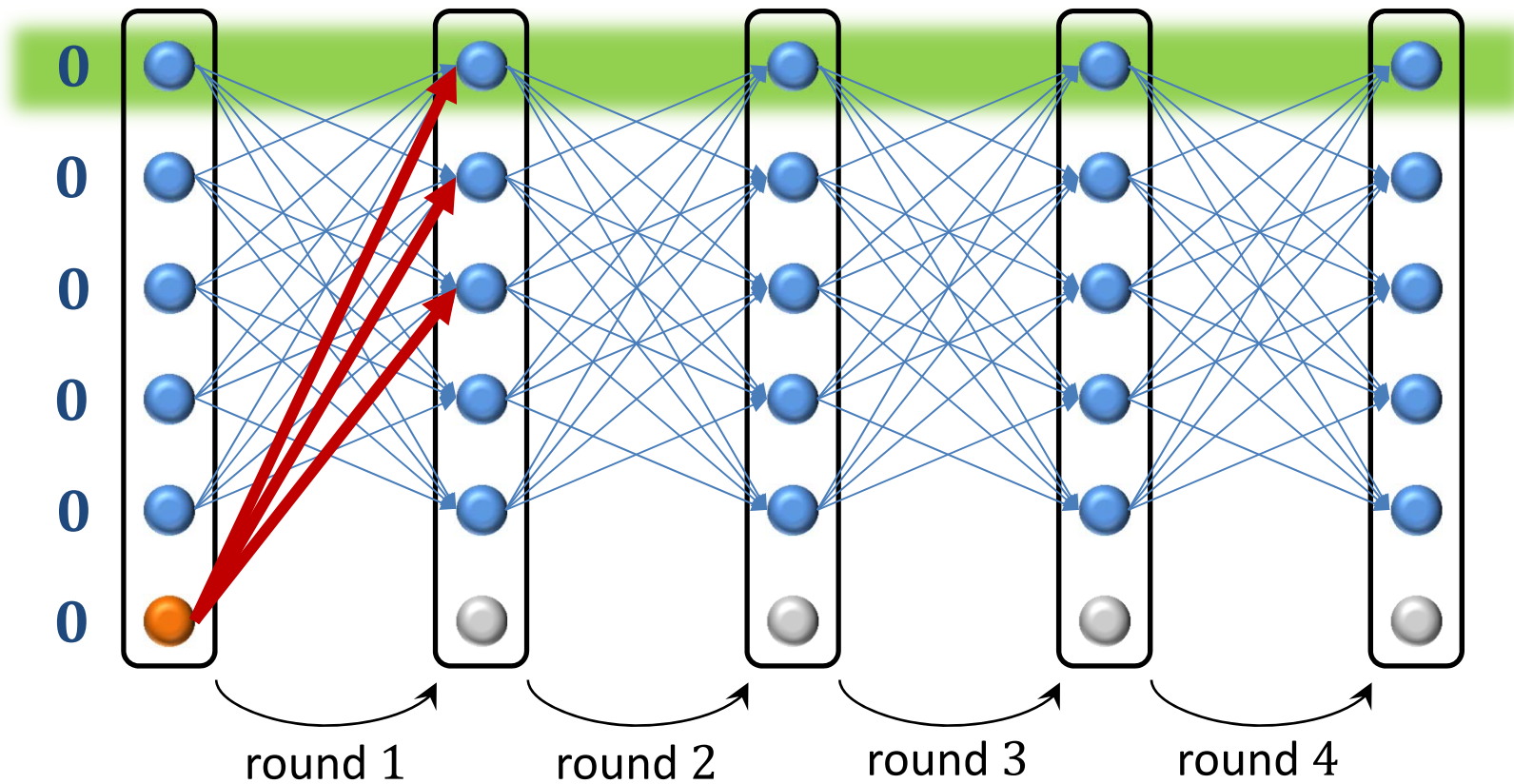
# Lower Bound on Rounds: Proof

Example:  $f = 4, n = 6$     Need to show: **4 rounds are not enough**



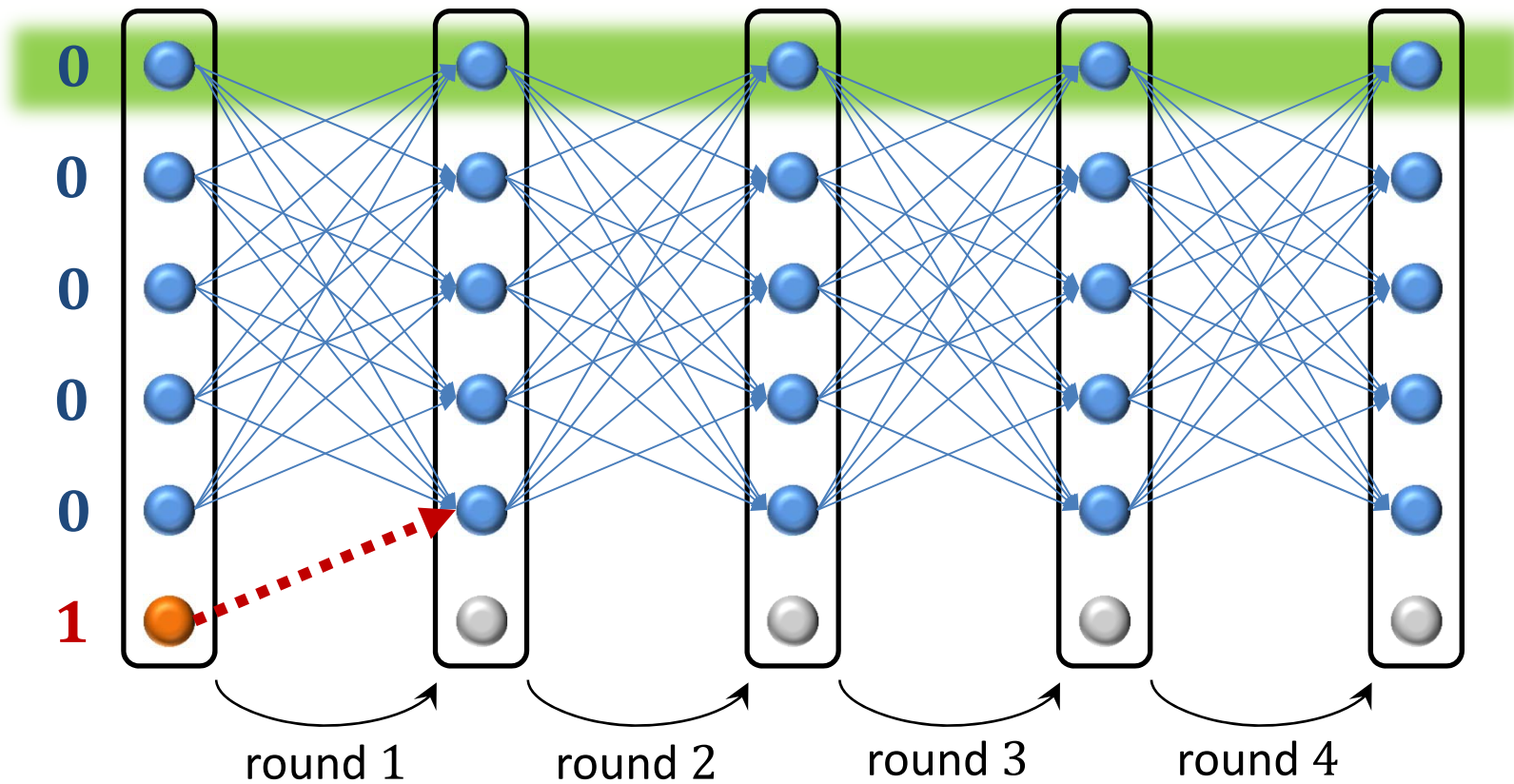
# Lower Bound on Rounds: Proof

Example:  $f = 4, n = 6$     Need to show: **4 rounds are not enough**



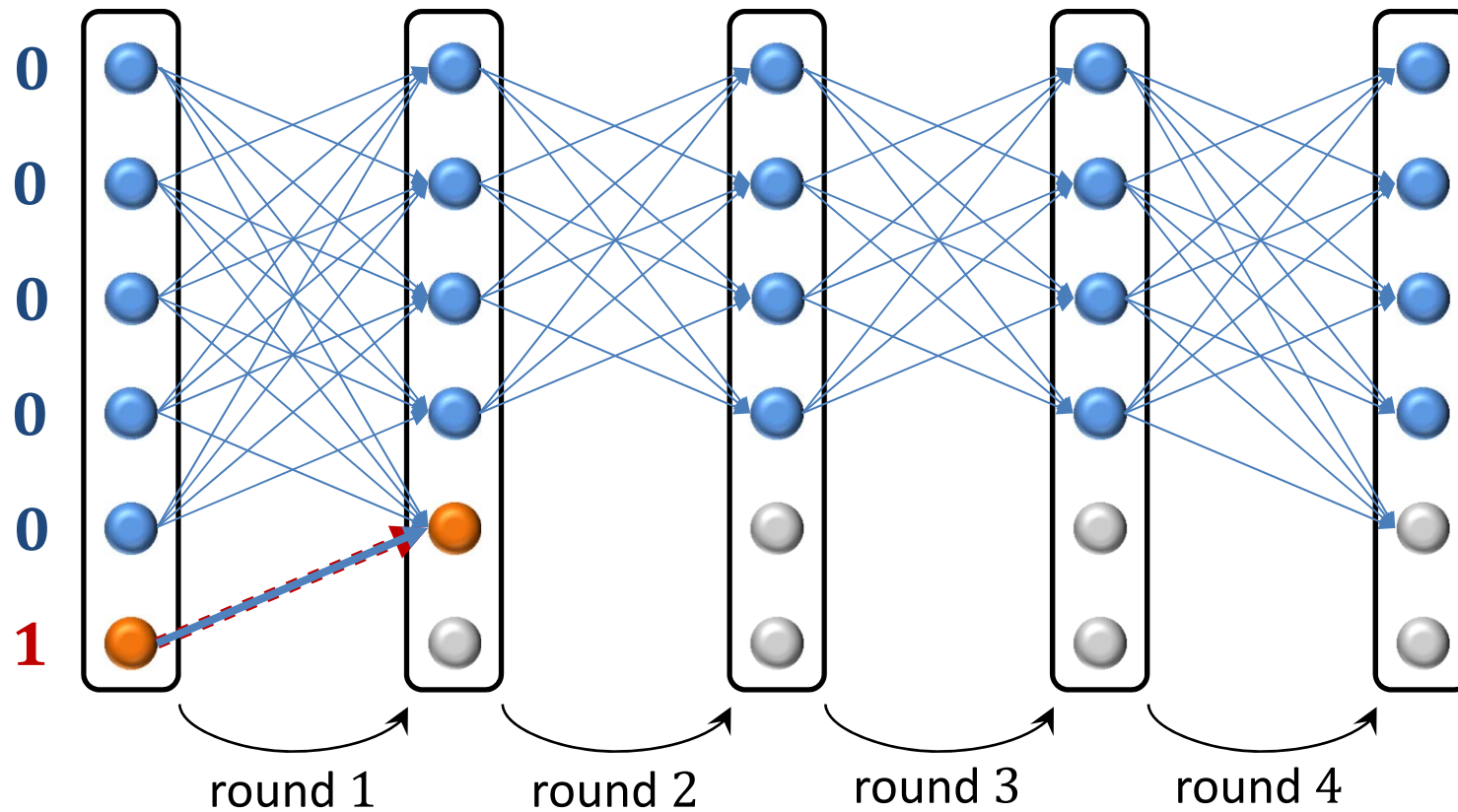
# Lower Bound on Rounds: Proof

Example:  $f = 4, n = 6$     Need to show: **4 rounds are not enough**



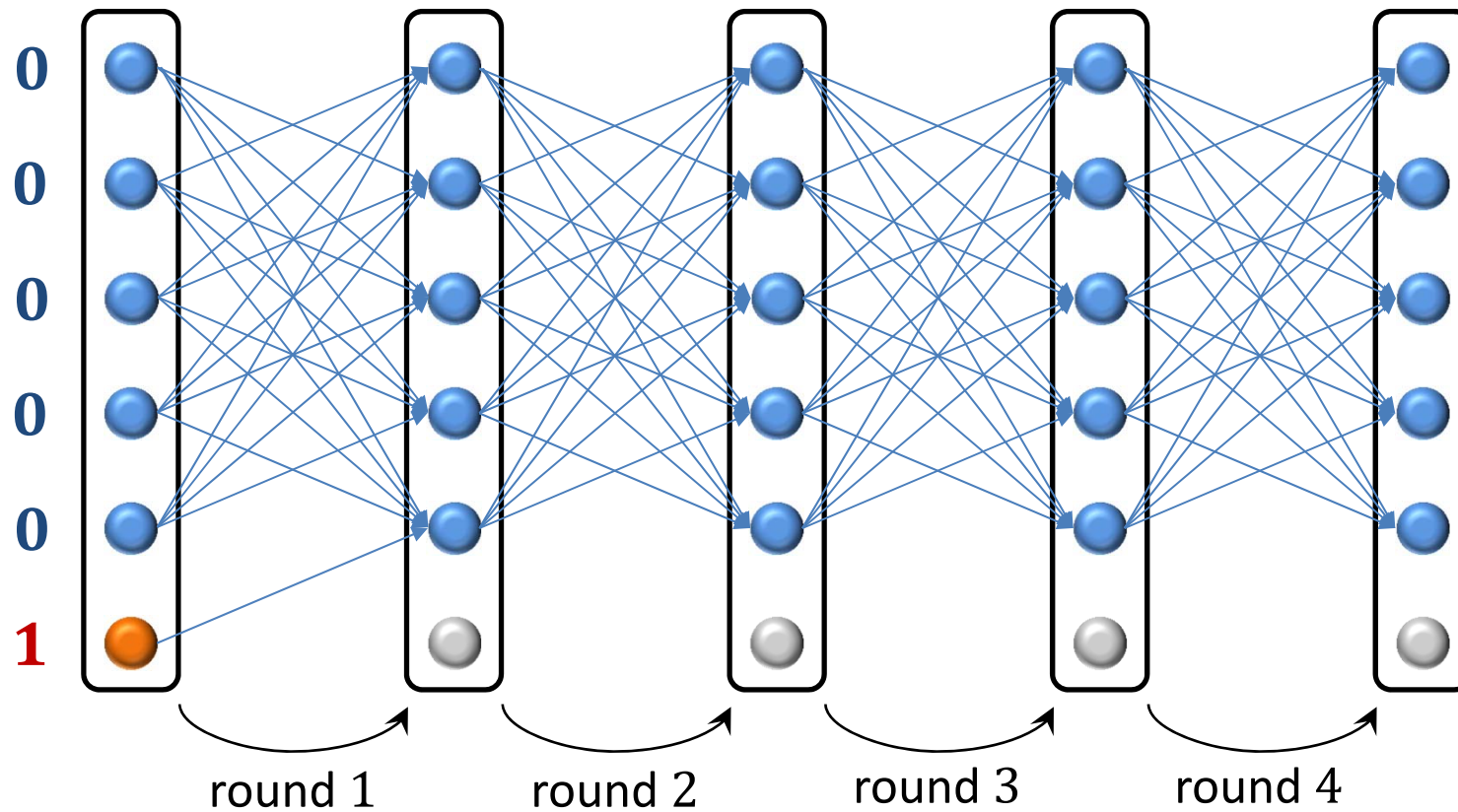
# Lower Bound on Rounds: Proof

Example:  $f = 4, n = 6$     Need to show: **4 rounds are not enough**



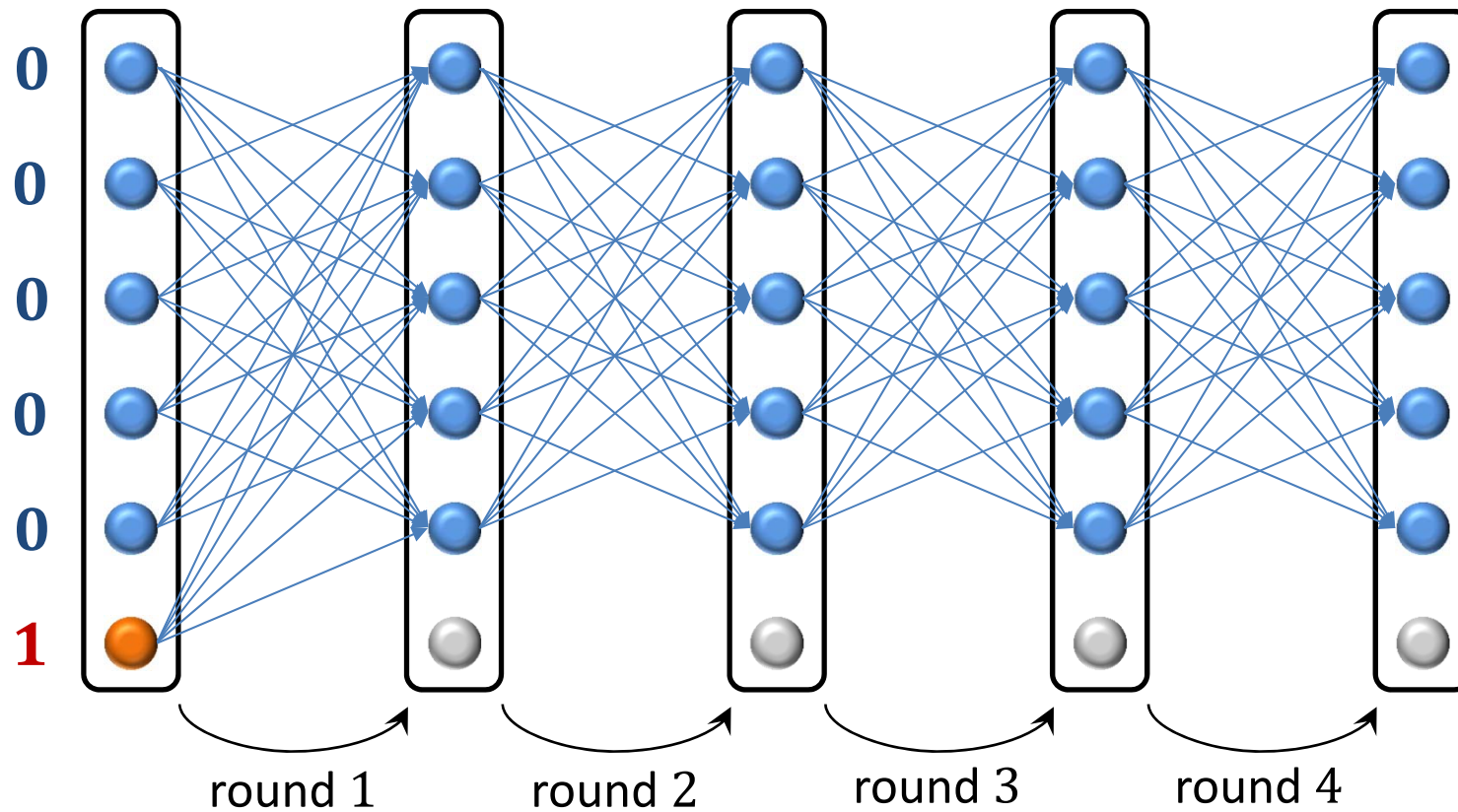
# Lower Bound on Rounds: Proof

Example:  $f = 4, n = 6$     Need to show: **4 rounds are not enough**



# Lower Bound on Rounds: Proof

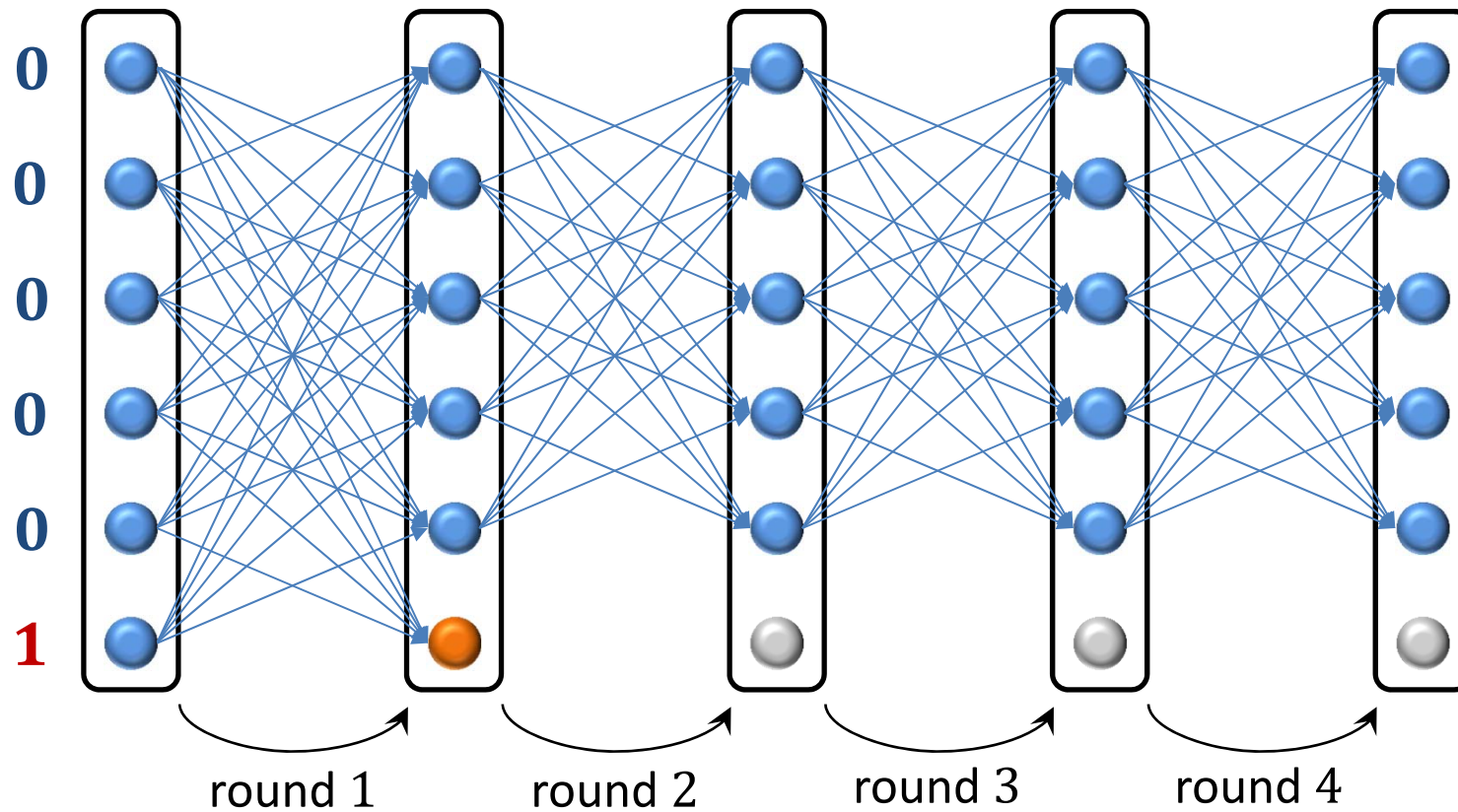
Example:  $f = 4, n = 6$     Need to show: **4 rounds are not enough**





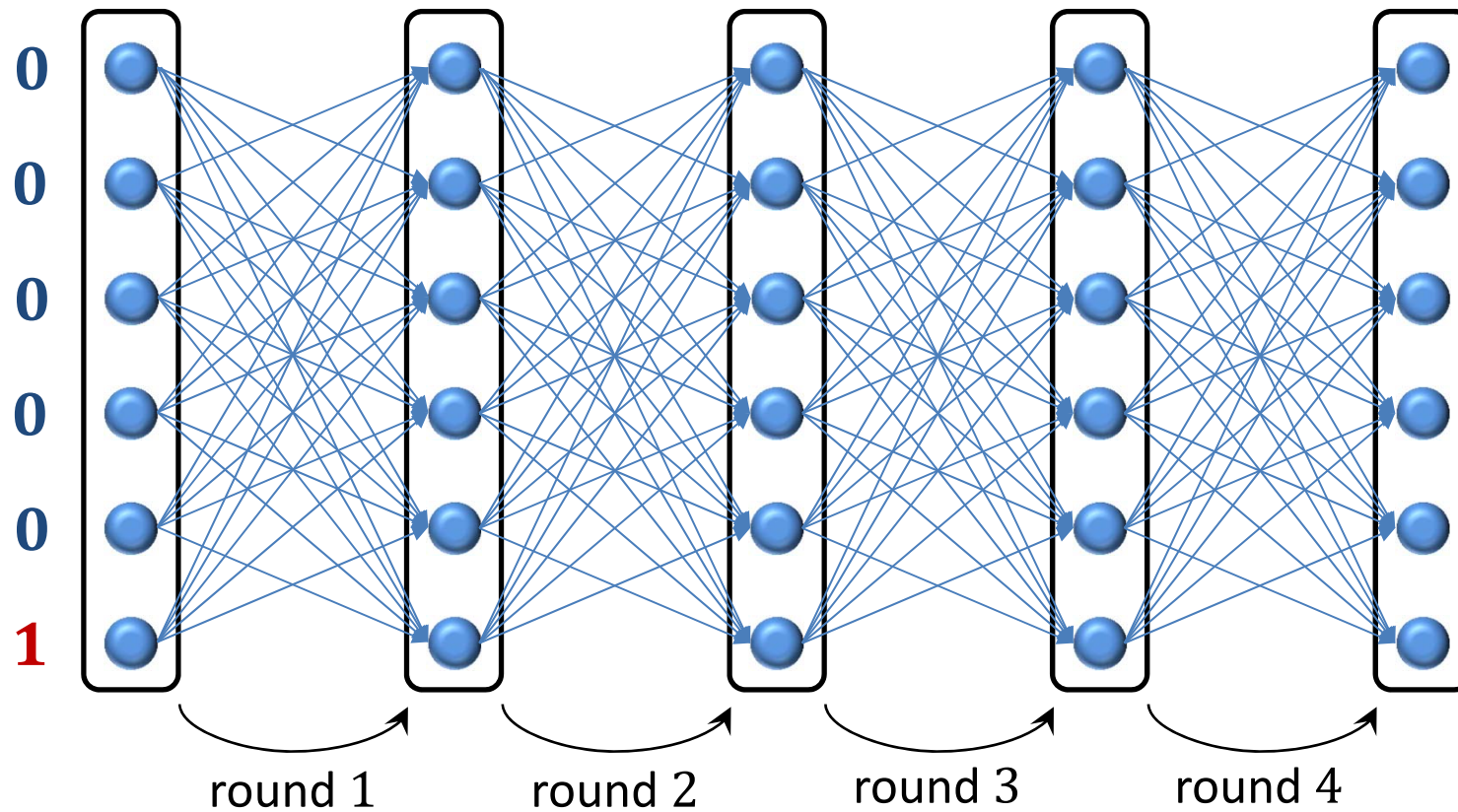
# Lower Bound on Rounds: Proof

Example:  $f = 4, n = 6$     Need to show: **4 rounds are not enough**



# Lower Bound on Rounds: Proof

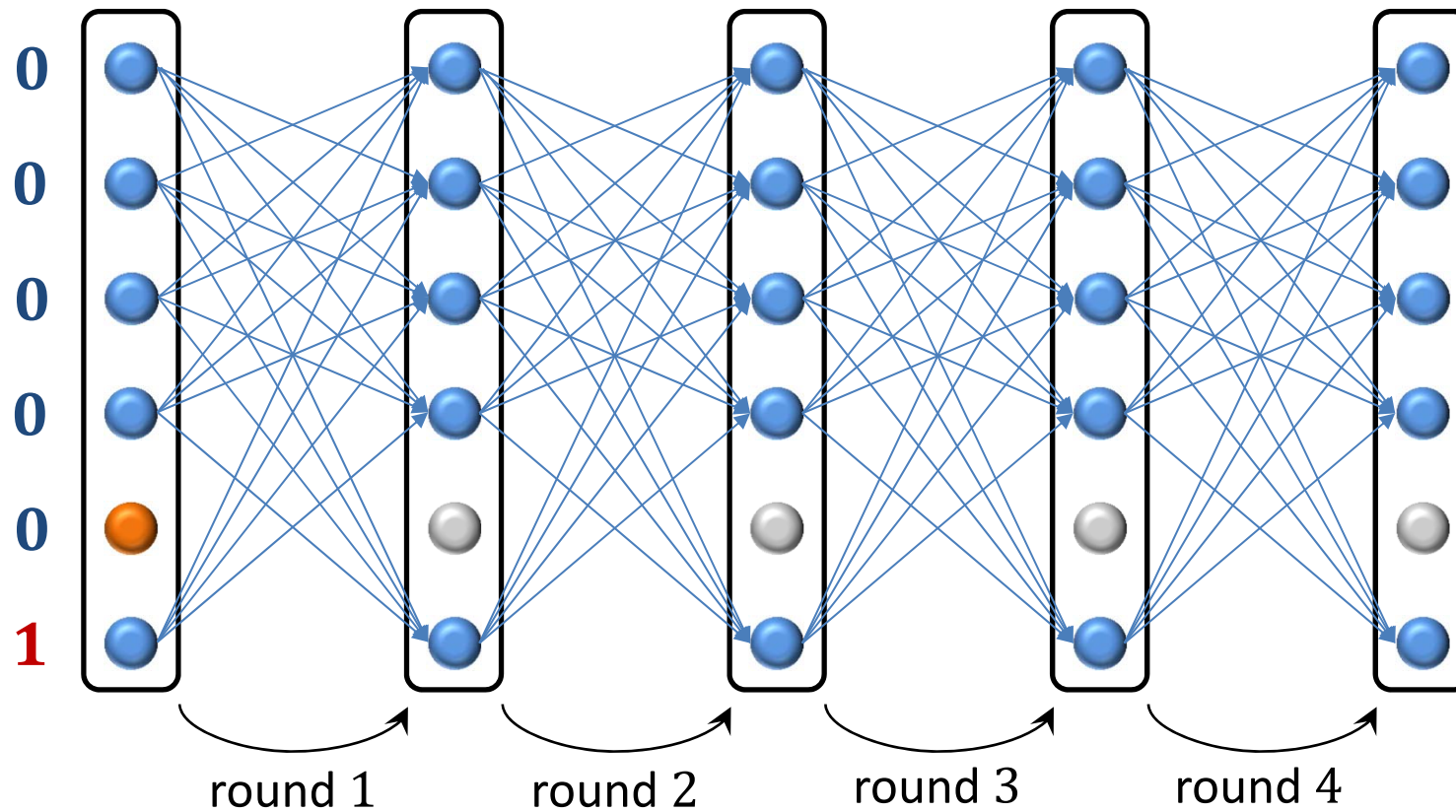
Example:  $f = 4, n = 6$     Need to show: **4 rounds are not enough**





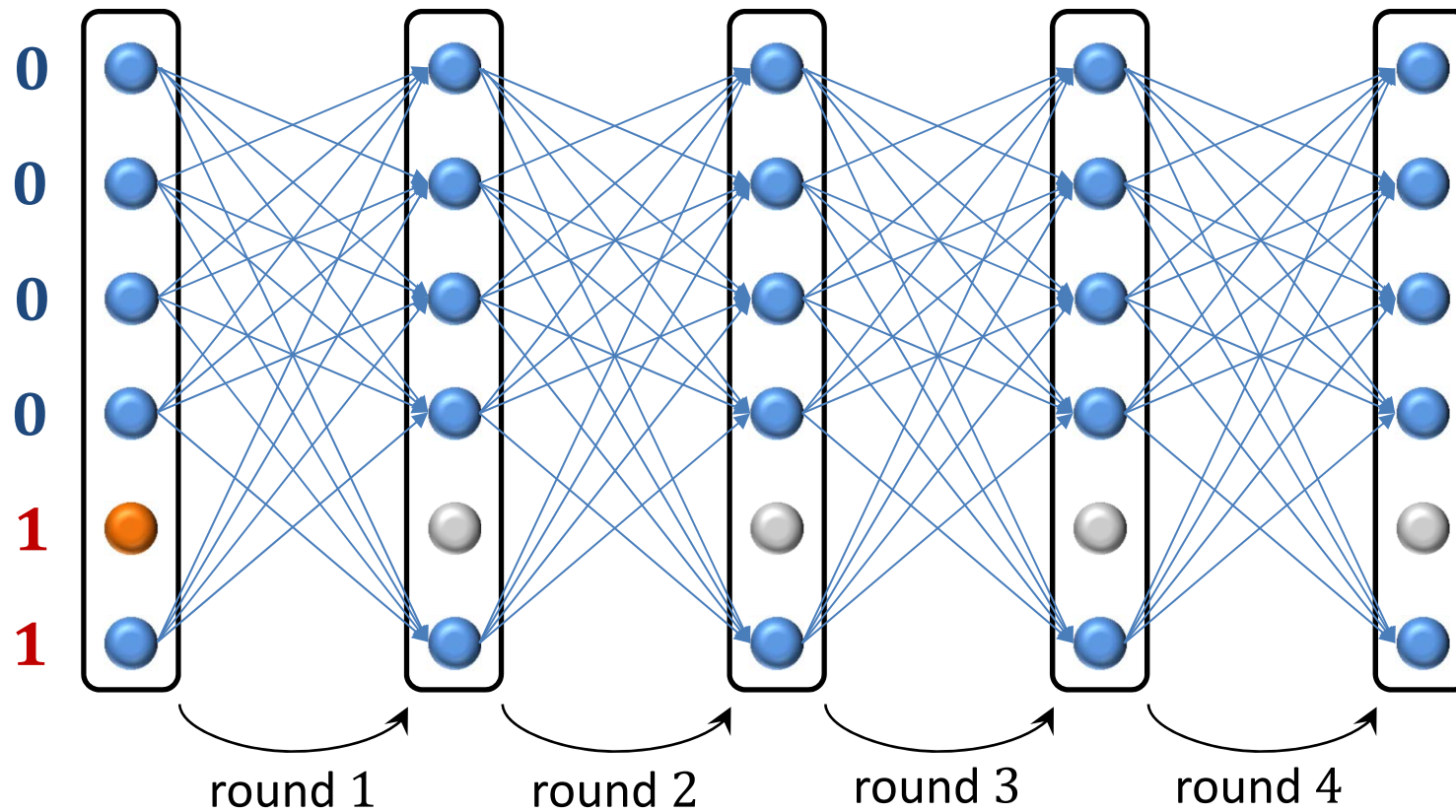
# Lower Bound on Rounds: Proof

Example:  $f = 4, n = 6$     Need to show: **4 rounds are not enough**



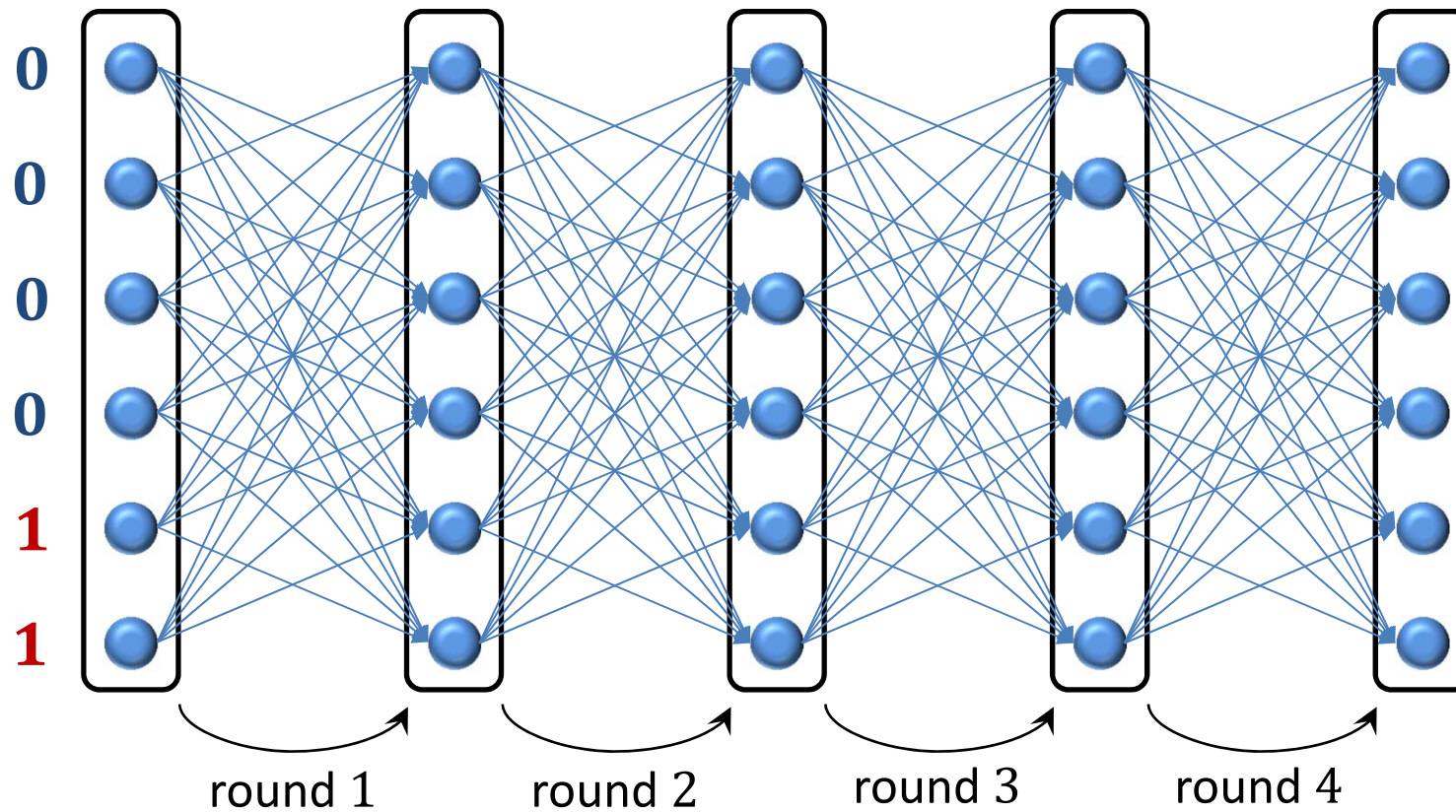
# Lower Bound on Rounds: Proof

Example:  $f = 4, n = 6$     Need to show: **4 rounds are not enough**



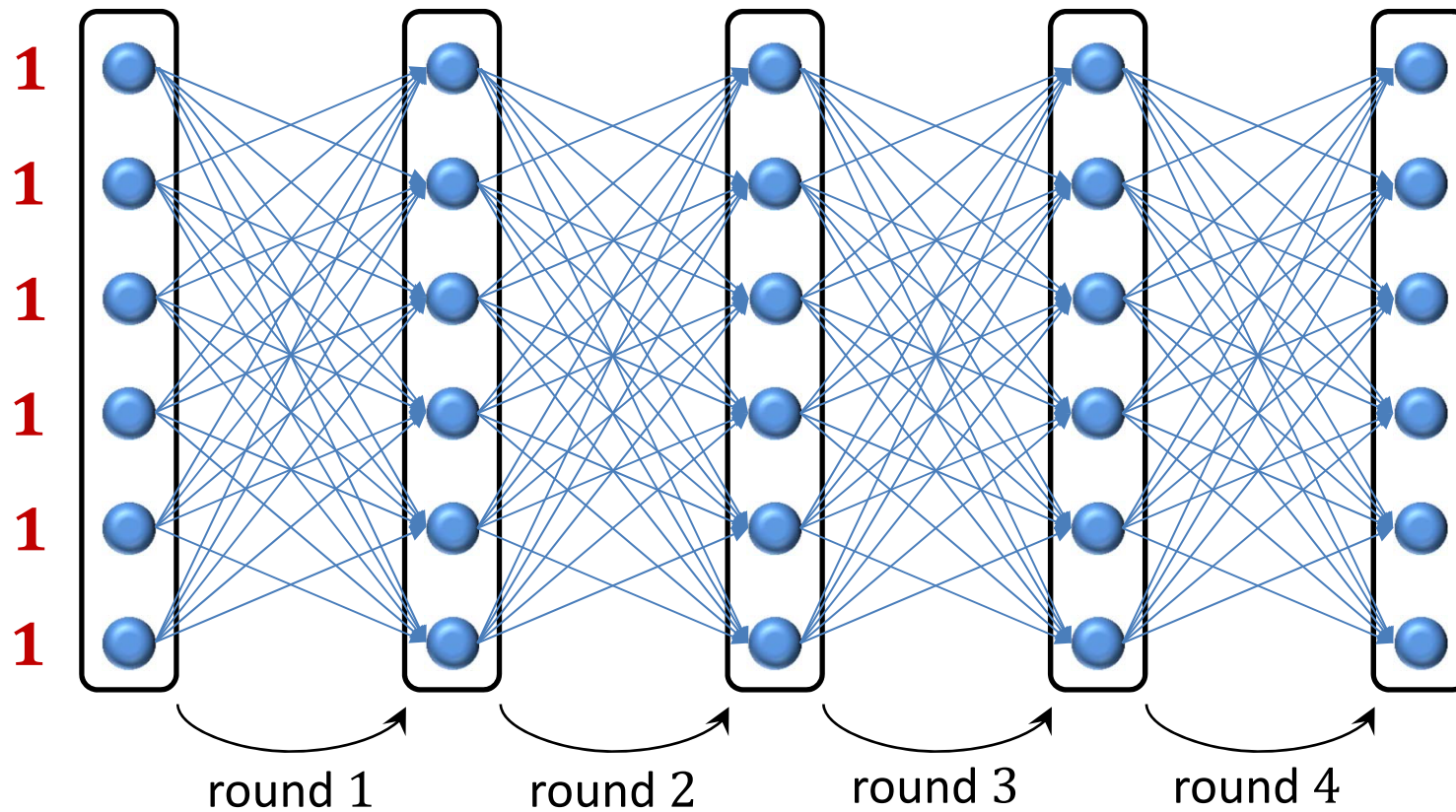
# Lower Bound on Rounds: Proof

Example:  $f = 4, n = 6$     Need to show: **4 rounds are not enough**



# Lower Bound on Rounds: Proof

Example:  $f = 4, n = 6$     Need to show: **4 rounds are not enough**



# Lower Bound on Rounds



## Theorem

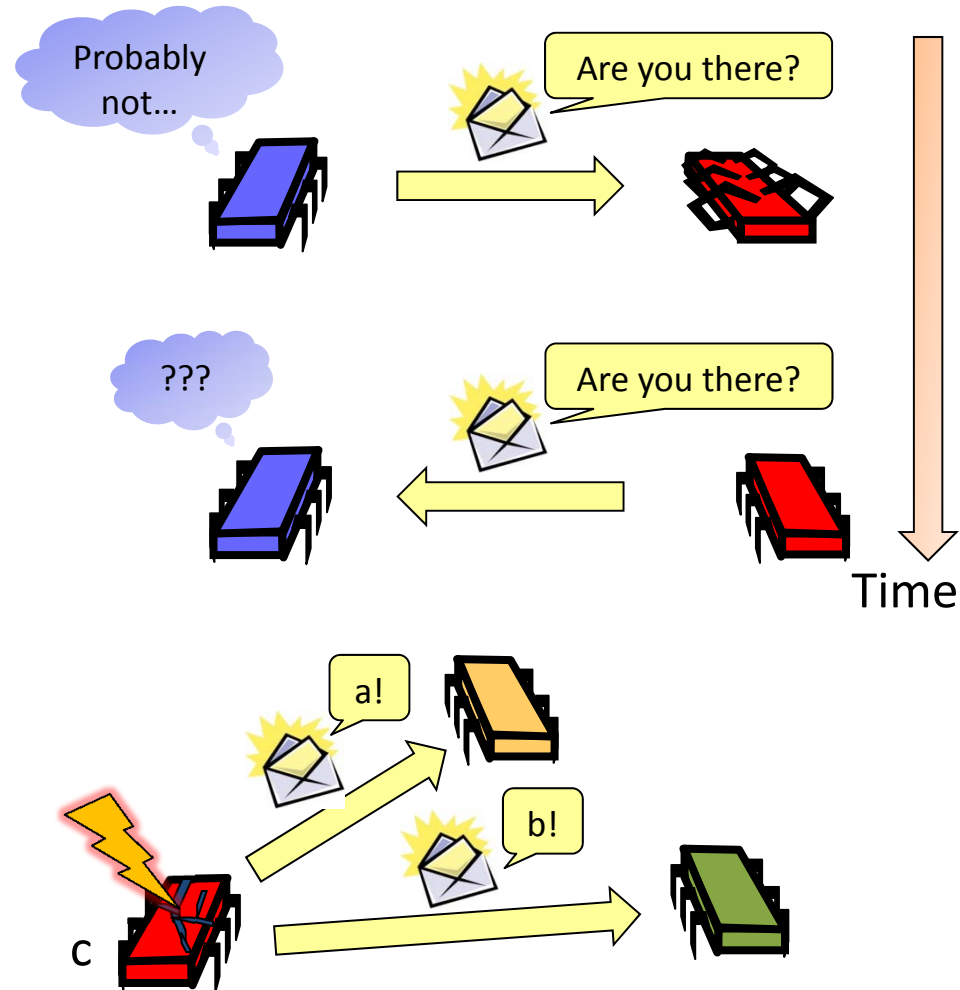
If at most  $f \leq n - 2$  of  $n$  nodes of a synchronous message passing system can crash, at least  $f + 1$  rounds are needed to solve consensus.

## Proof:

- Similarity chain starting with fault-free all-zeroes execution and ending with fault-free all-ones execution
- In all executions, at most one crash per round
- Construction works as long as there are at least 2 non-faulty nodes in each execution ( $n \geq f + 2$ )
- **Validity:** all-zeroes  $\Rightarrow$  decision 0; all-ones  $\Rightarrow$  decision 1  
**Similarity Chain:** same decision in all executions

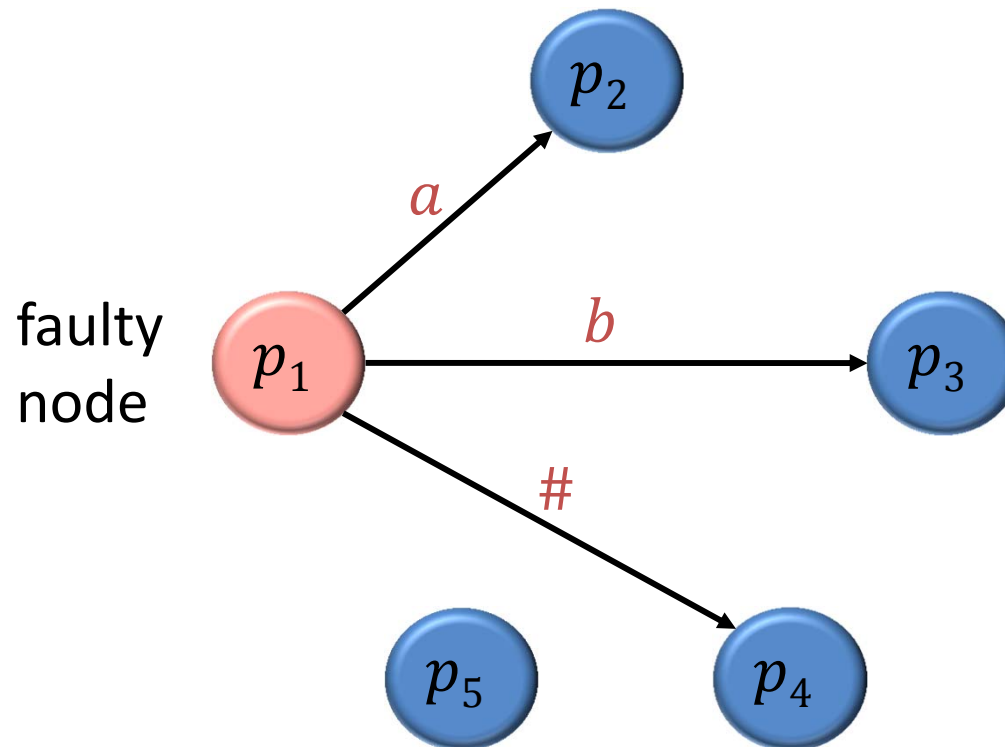
# Arbitrary Behavior

- The assumption that processes crash and stop forever is sometimes too optimistic
- Maybe the processes fail and recover:
- Maybe the processes are damaged:



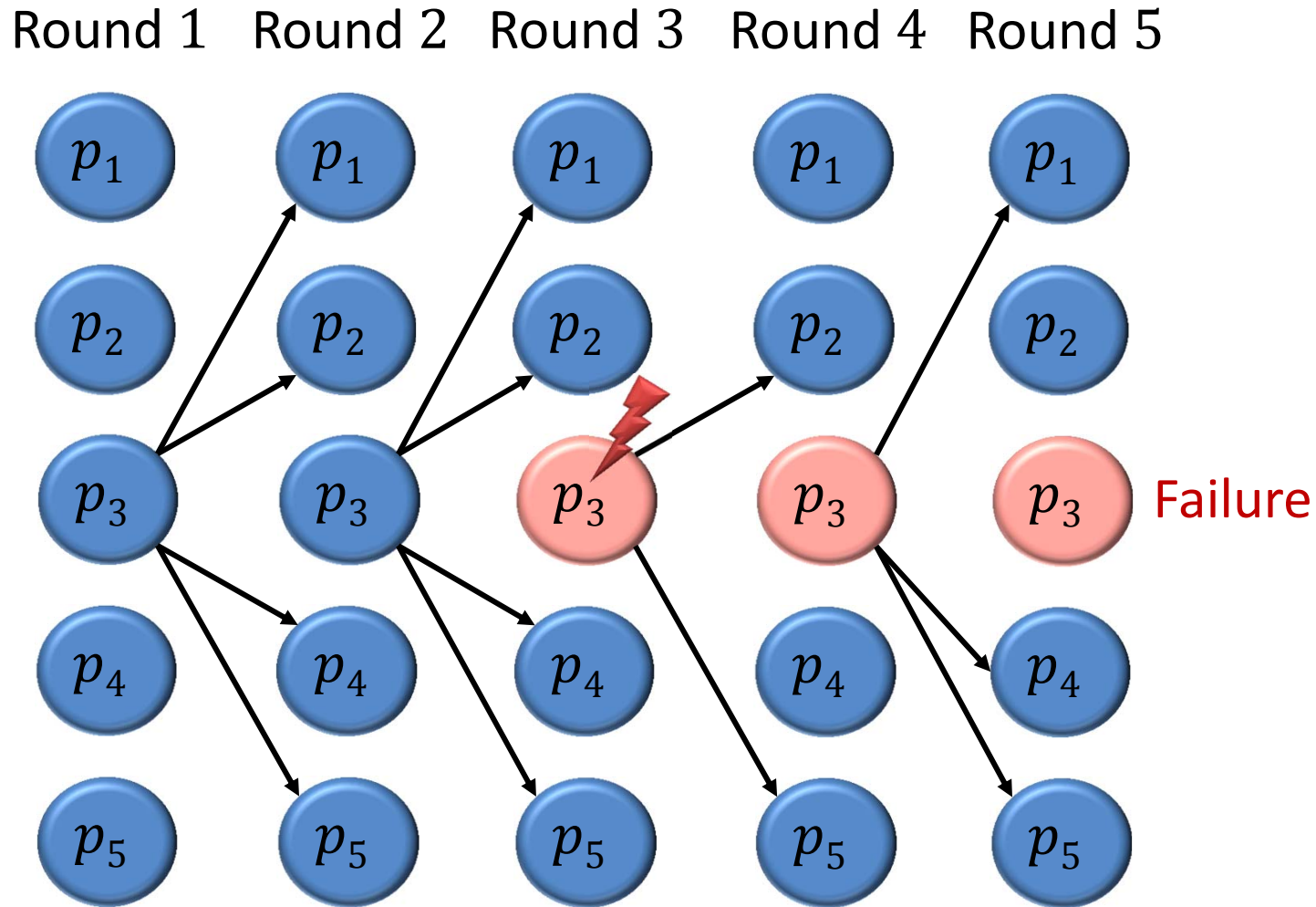
# Consensus #5: Byzantine Failures

- Different processes may receive different values
- A Byzantine process can behave like a crash-failed process





# After Failure, Node Remains in Network





# Consensus with Byzantine Failures

---

- Again: If an algorithm solves consensus for  $f$  failed processes, we say it is an  $f$ -resilient consensus algorithm
- **Validity:** If all non-faulty processes start with the same value, then all non-faulty processes decide on that value
  - Note that in general this validity condition does not guarantee that the final value is an input value of a non-Byzantine process
  - However, if the input is binary, then the validity condition ensures that processes decide on a value that at least one non-Byzantine process had initially
- Obviously, any  $f$ -resilient consensus algorithm requires at least  $f + 1$  rounds (follows from the crash failure lower bound)
- How large can  $f$  be...? Can we reach consensus as long as the majority of processes is correct (non-Byzantine)?

# Impossibility



## Theorem

There is no  $f$ -resilient Byzantine consensus algorithm for  $n$  nodes for  $f \geq n/3$

## Proof outline

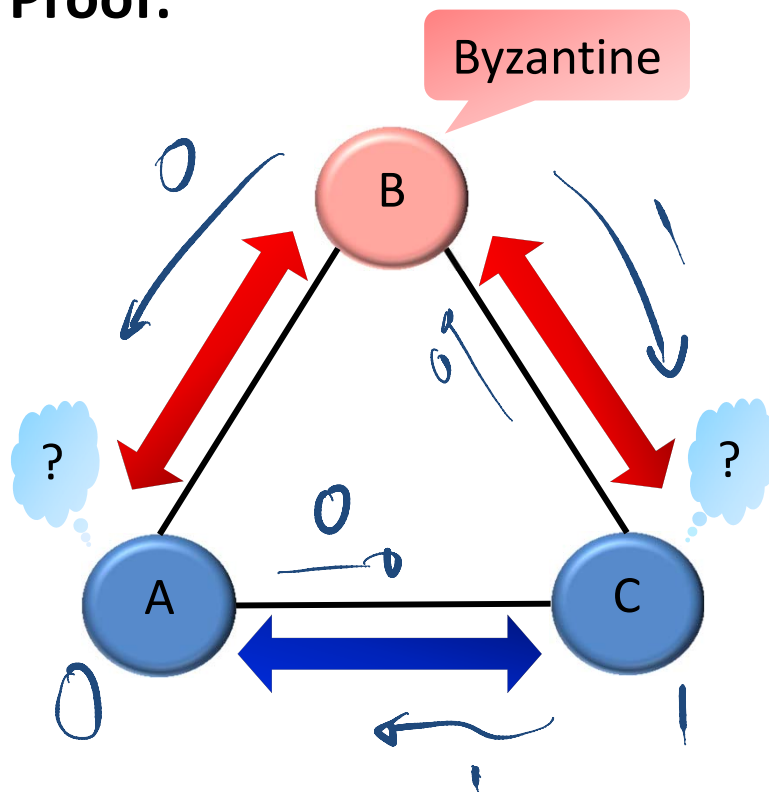
- First, we prove the 3 node case
  - not possible for  $f = 1$
- The general case can then be proved by reduction from the 3 node case
  - Given an algorithm for  $n$  node and  $f$  faults for  $f \geq n/3$ , we can construct a 1-resilient 3-node algorithm

# The 3 Node Case

## Lemma

There is no 1-resilient algorithm for 3 nodes

### Proof:

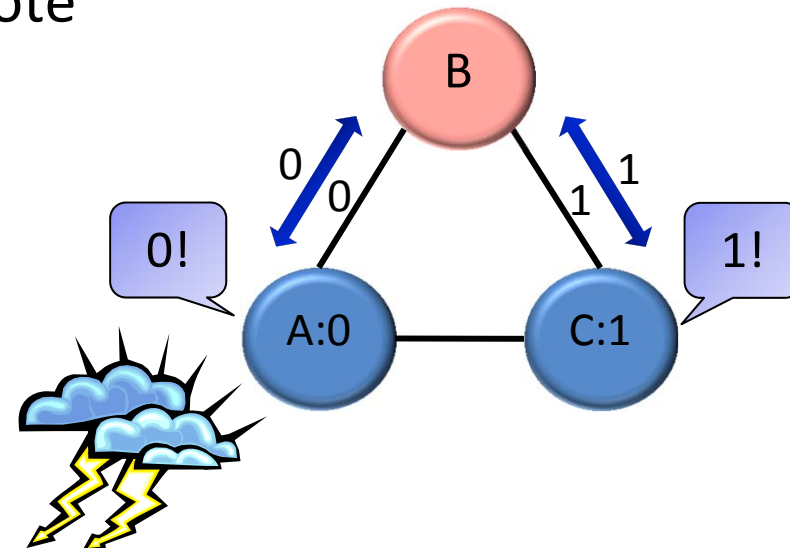
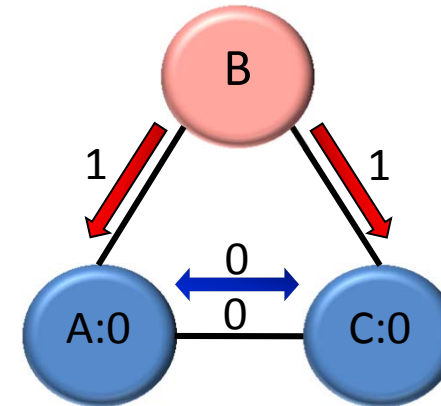


### Intuition:

- Node **A** may also receive information from **C** about **B**'s messages to **C**
- Node **A** may receive conflicting information about **B** from **C** and about **C** from **B** (the same for **C**!)
- It is impossible for **A** and **C** to decide which information to base their decision on!

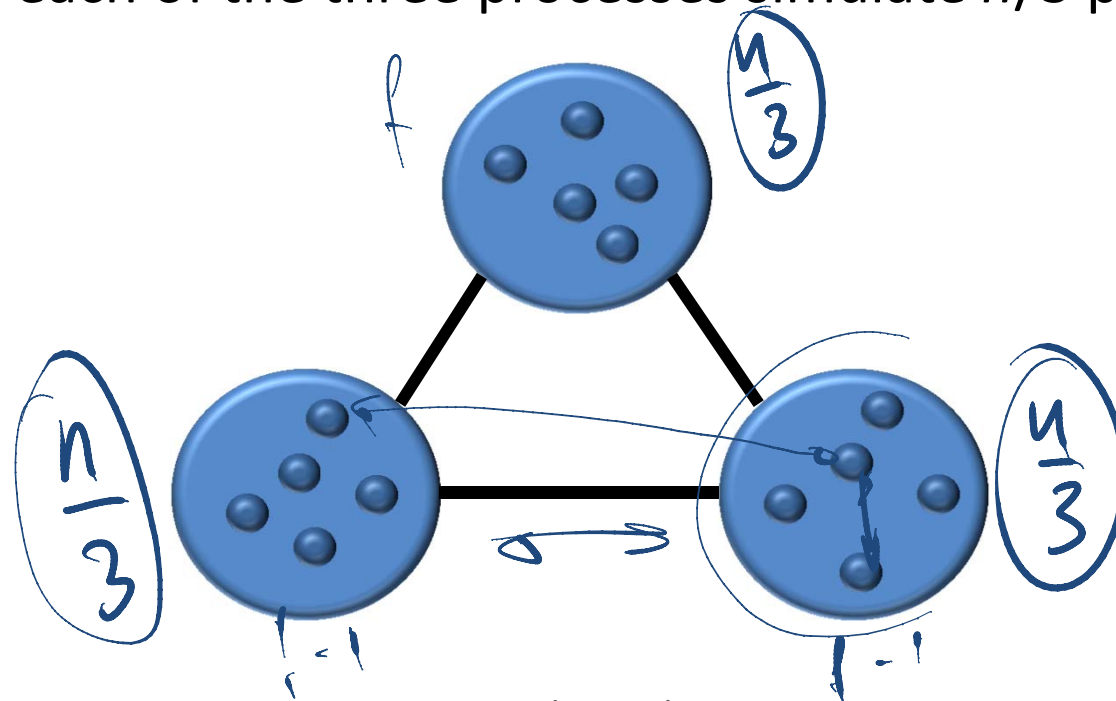
# Proof Sketch

- Assume that both **A** and **C** have input 0. If they decided 1, they could violate the validity condition  $\rightarrow$  **A** and **C** must decide 0 independent of what **B** says
- Similarly, **A** and **C** must decide 1 if their inputs are 1
- We see that the processes must base their decision on the majority vote
- If **A**'s input is 0 and **B** tells **A** that its input is 0  $\rightarrow$  **A** decides 0
- If **C**'s input is 1 and **B** tells **C** that its input is 1  $\rightarrow$  **C** decides 1



# The General Case

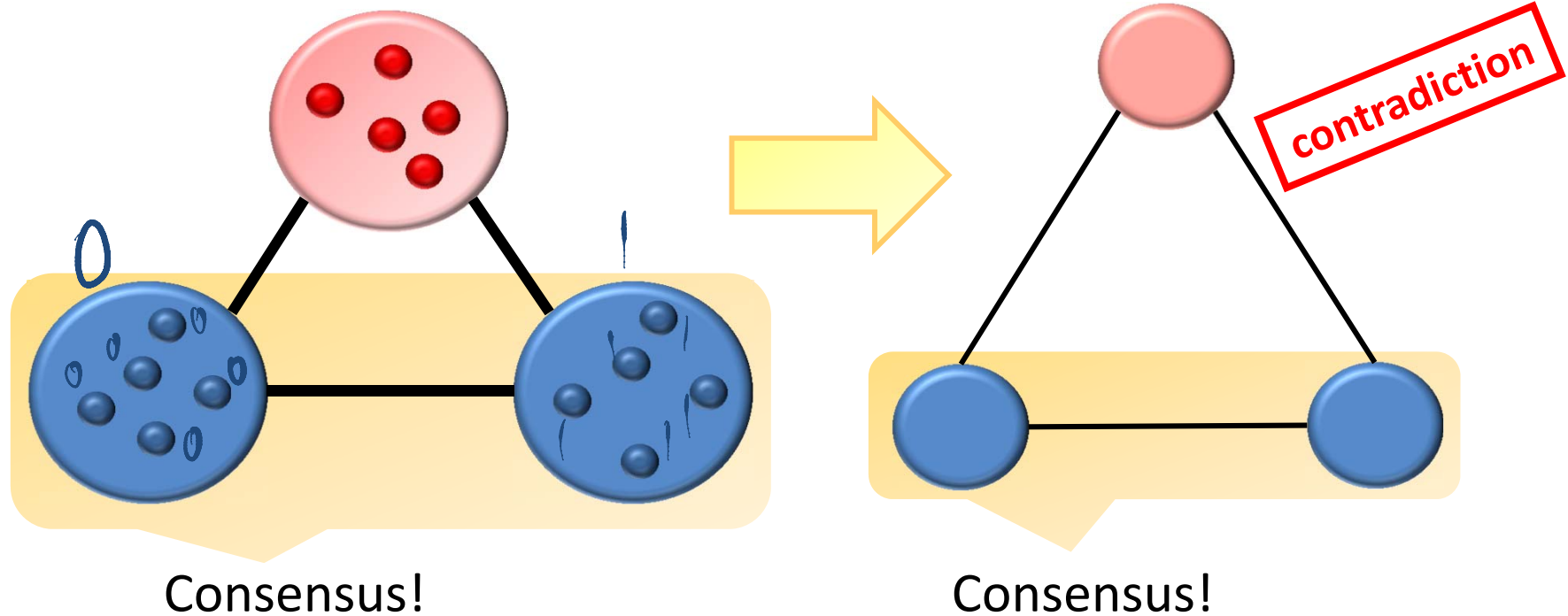
- Assume for contradiction that there is an  $f$ -resilient algorithm  $A$  for  $n$  nodes, where  $f \geq n/3$
- We use this algorithm to solve consensus for 3 nodes where one node is Byzantine!
- For simplicity assume that  $n$  is divisible by 3
- We let each of the three processes simulate  $n/3$  processes



# The General Case

$$f = \frac{n}{3}$$

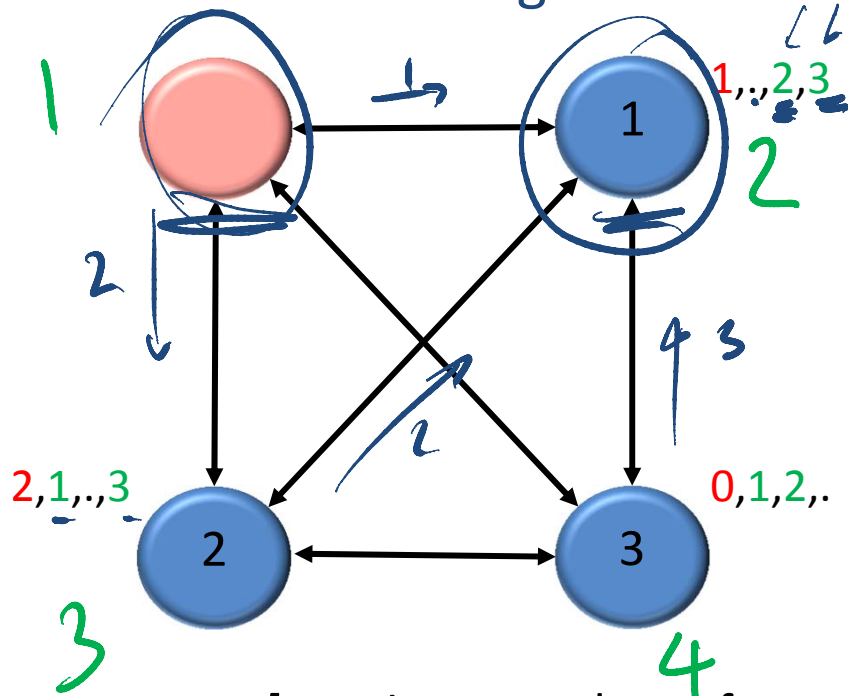
- One of the 3 nodes is Byzantine  $\Rightarrow$  its  $n/3$  simulated nodes may all behave like Byzantine nodes
- Since algorithm A tolerates  $n/3$  Byzantine failures, it can still reach consensus  
 $\Rightarrow$  We solved the consensus problem for three processes!



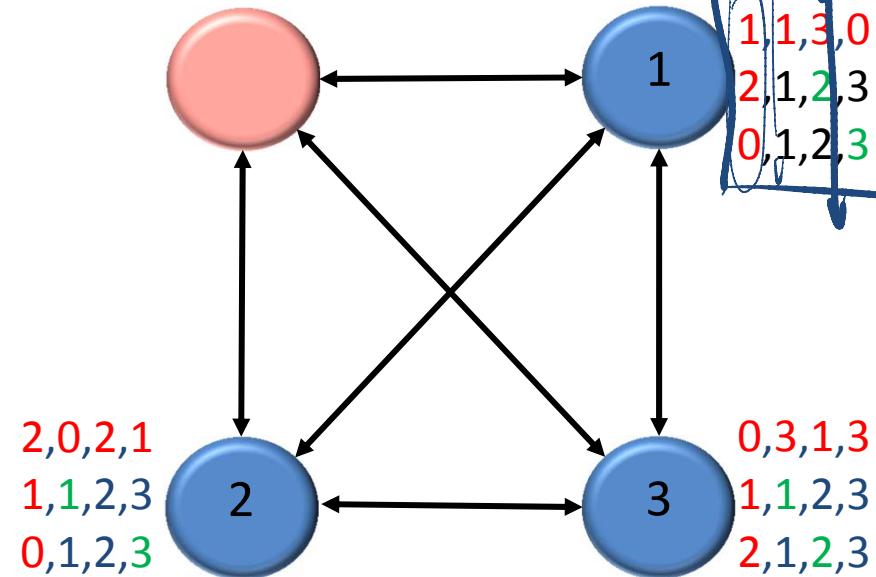
# Cons. #6: Simple Byzantine Agreement Alg.

- Can the nodes reach consensus if  $n > 3f$ ?
- A simpler question: What if  $n = 4$  and  $f = 1$ ?
- The answer is yes. It takes two rounds:

Round 1: Exchange all values



Round 2: Exchange received info

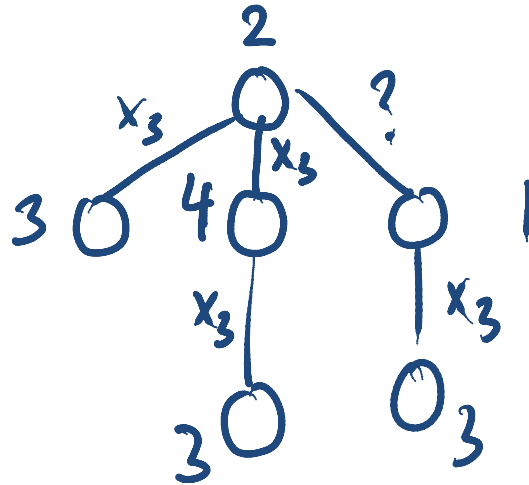


[matrix: one column for each original value, one row for each neighbor]

# Simple Byzantine Agreement Algorithm



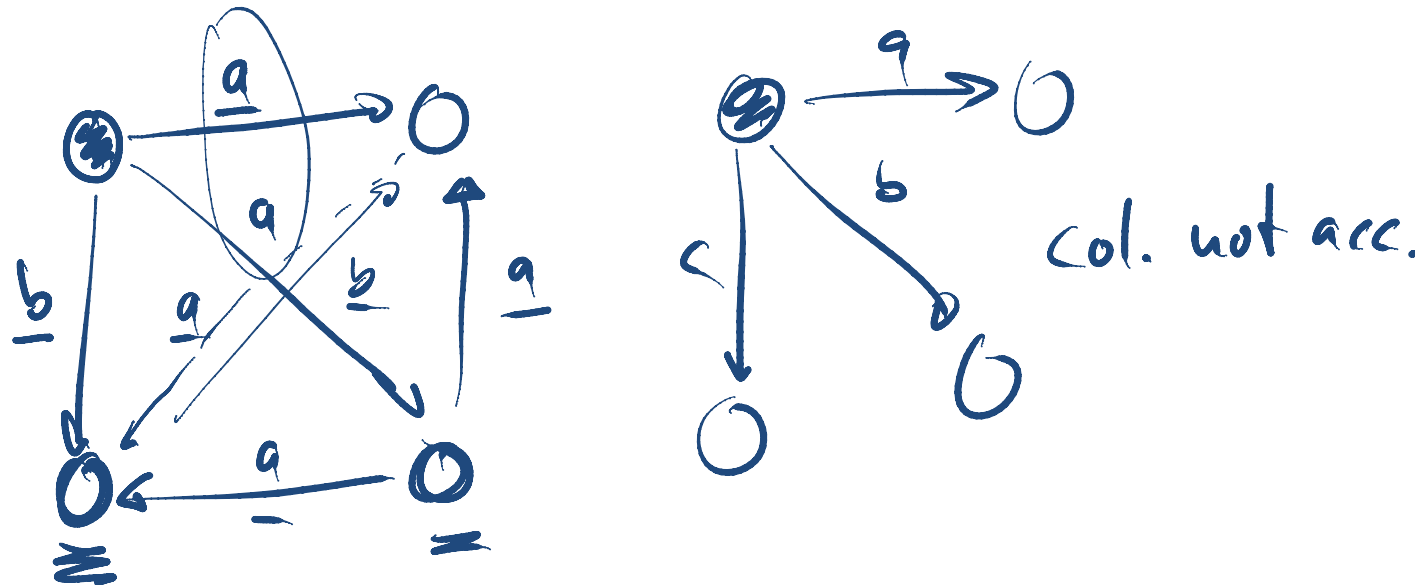
- After round 2, each node has received 12 values, 3 for each of the 4 input values (columns). If at least 2 of the 3 values of a column are equal, this value is accepted, otherwise it is discarded.
  - Values of honest nodes are accepted





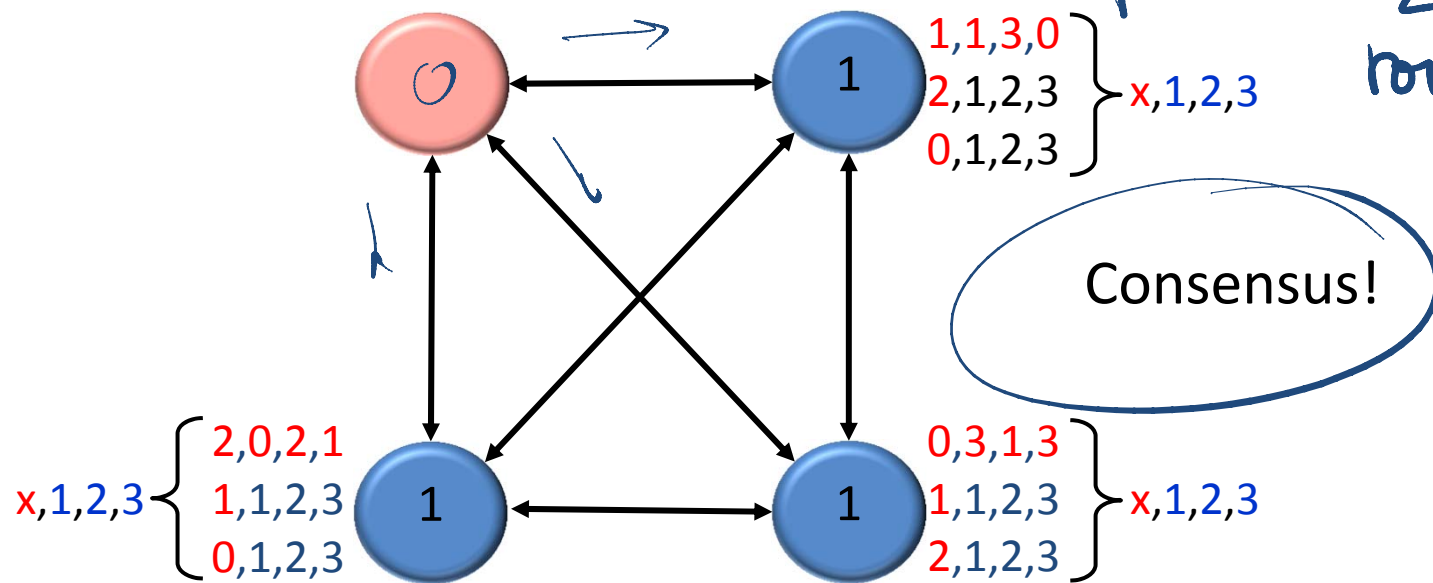
# Simple Byzantine Agreement Algorithm

- After round 2, each node has received 12 values, 3 for each of the 4 input values (columns). If at least 2 of the 3 values of a column are equal, this value is accepted, otherwise it is discarded.
  - Values of honest nodes are accepted
  - The ~~value~~<sup>col.</sup> of the Byzantine node is accepted iff it sends the same value to at least two nodes in the first round.



# Simple Byzantine Agreement Algorithm

- After round 2, each node has received 12 values, 3 for each of the 4 input values (columns). If at least 2 of the 3 values of a column are equal, this value is accepted, otherwise it is discarded.
  - Values of honest nodes are accepted
  - The value of the Byzantine node is accepted iff it sends the same value to at least two nodes in the first round.
- Decide on *most often* ~~minimum~~ accepted value!

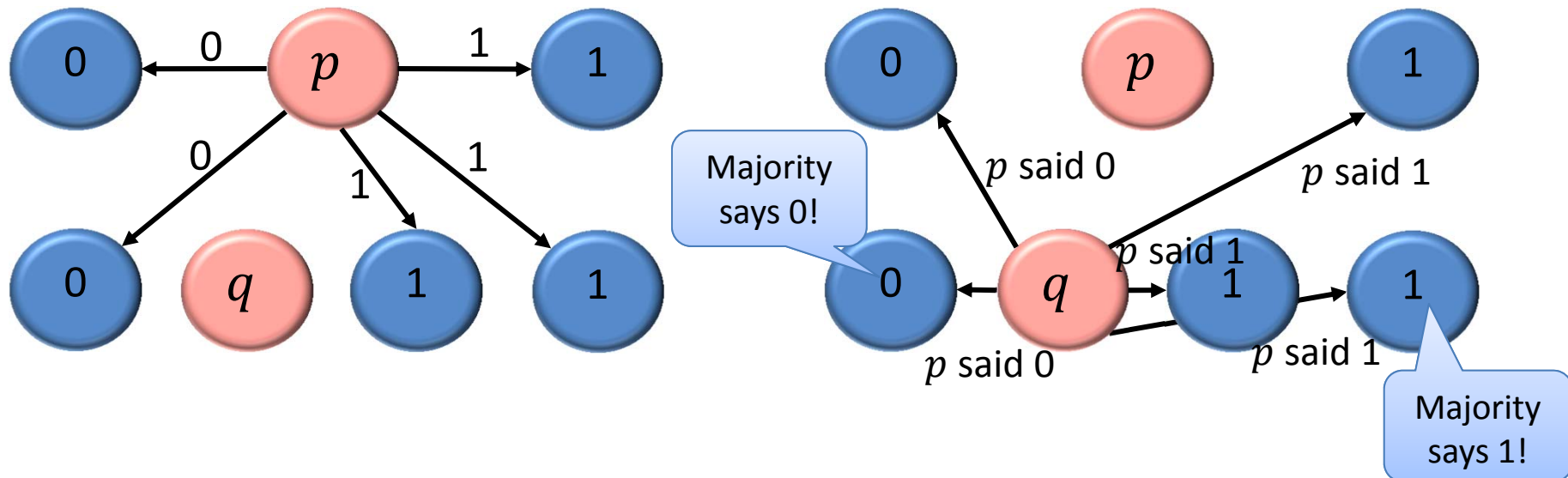


# Simple Byzantine Agreement Algorithm

- Does the algorithm still work in general for any  $f$  and  $n > 3f$ ?
- The answer is no. Try  $f = 2$  and  $n = 7$ :

Round 1: Exchange all values

Round 2: Exchange received info



- The problem is that  $q$  can say different things about what  $p$  sent to  $q$ 
  - What is the solution to this problem?

# Simple Byzantine Agreement Algorithm

- The solution is simple: Again exchange all information!
- This way, the processes can learn that  $q$  gave inconsistent information about  $p$
- Hence,  $q$  can be excluded, and also  $p$  if it also gave inconsistent information (about  $q$ ).
- If  $f = 2$  and  $n > 6$ , consensus can be reached in 3 rounds!
- In fact, the following algorithm solves the problem for any  $f$  and any  $n > 3f$ :

Exchange all information for  $f + 1$  rounds  
Ignore all processes that provided inconsistent information  
Let all processes decide based on the same input

# Simple Byzantine Agreement Algorithm



**The proposed algorithm has several advantages:**

- + It works for **any  $f$**  and  **$n > 3f$** , which is **optimal**
- + It only takes  **$f + 1$  rounds**. This is even **optimal** for crash failures!
- + It **works for any input** and not just binary input

**However, it has some considerable disadvantages:**

- “Ignoring all processes that provided inconsistent information” is **not easy to formalize**
- The **size of the messages increases exponentially!**  
This is a severe problem. It is therefore worth studying whether it is possible to solve the problem with small(er) messages