# Parallel and Distributed Computing
# CS3006 (BCS-6C/6D)
# Lecture 26

Instructor: Dr. Syed Mohammad Irteza

Assistant Professor, Department of Computer Science, FAST

04 May, 2023

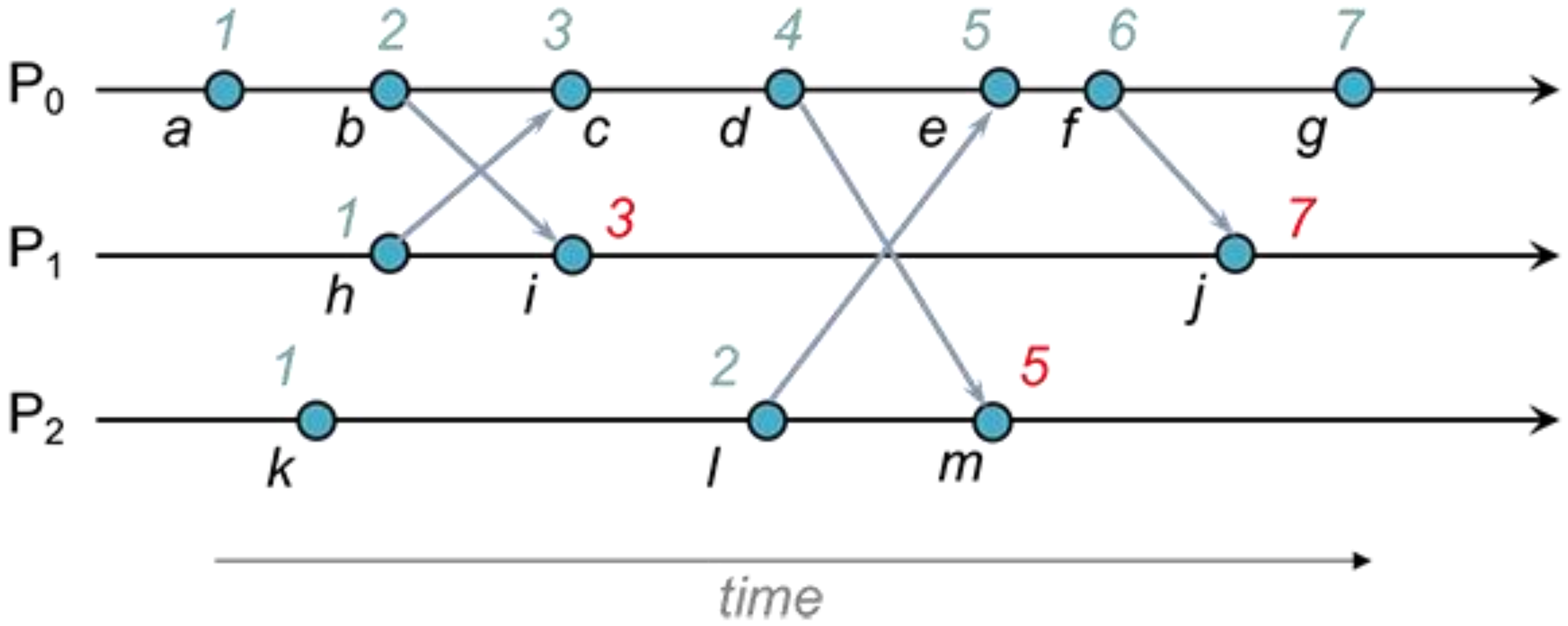
# Previous Lecture

- Web Services
  - UDDI
  - WSDL
  - SOAP
  - XML
- Logical Clocks
  - No global clock
  - “Happens after” relationship

# How to create a web service

- Following link provides a useful example to create and consume a web service in Visual Studio!
  - https://www.c-sharpcorner.com/UploadFile/4d9083/create-simple-web-service-in-visual-studio-2008-2010-2012/

# Another Example: Logical Clocks



# Concurrent events

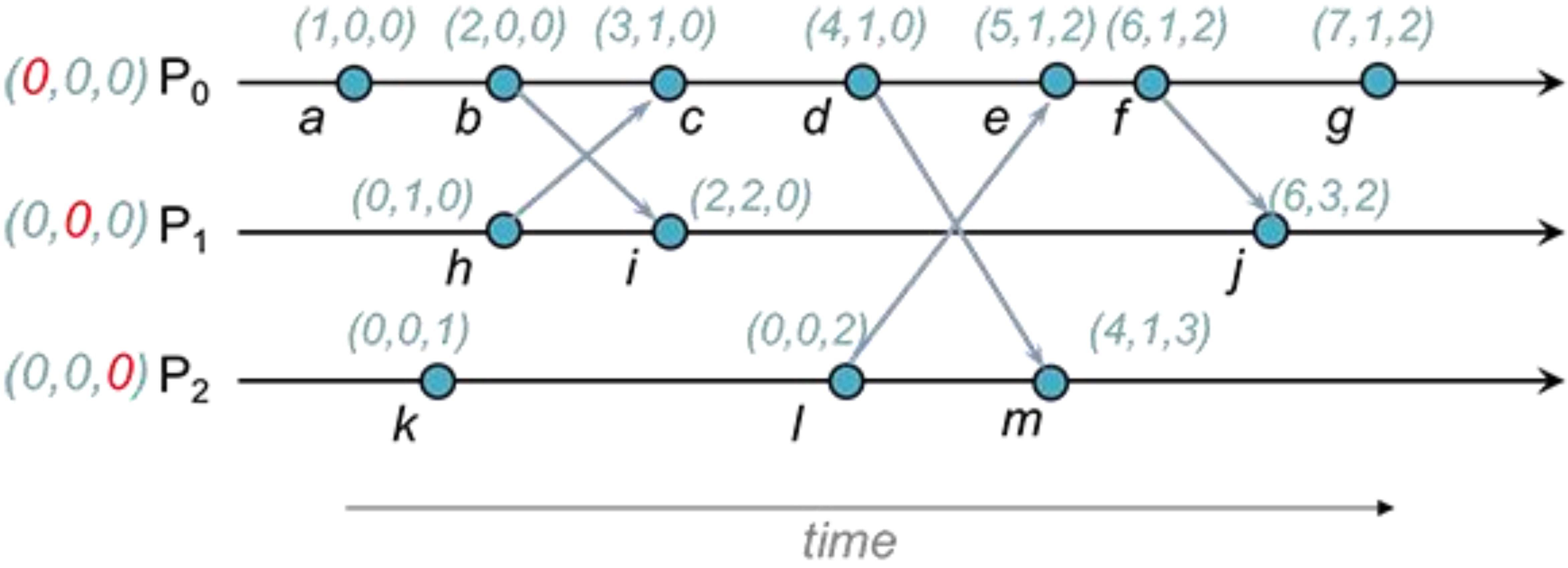
- If a, b are concurrent, LT(a) and LT(b) may have arbitrary values!
- Thus, logical time lets us determine that *a* potentially happened before *b*, but not that *a* definitely did so!
- Example: processes p and q never communicate. Both will have events 1, 2, ... but even if LT(e)<LT(e') e may not have happened before e'

# Vector Clocks

- Extend logical timestamps into a list of counters, one per process in the system
- The rules for updating vector clocks are as follows:
  - Before affixing a vector timestamp to an event, a process increments the element of its local vector that corresponds to its position in the vector.
    - For example, process 0 increments element 0 of its vector, process 1 increments element 1 of its vector, and so on.

  - When a process receives a message, it also first increments its element of the vector (i.e., it applies the previous rule). It then sets the vector of the *received* event to a set of values that contains the higher of two values when doing and element-by-element comparison of the original event’s vector and the vector received in the message.

# Vector Clock Example



# Vector timestamps accurately represent happens-before relation

- Define VT(e)<VT(e’) if,
  - for all i, VT(e)[i]≤VT(e’)[i], and
  - for some j, VT(e)[j]<VT(e’)[j]

- Example: if VT(e)=[2,1,1,0] and VT(e’)=[2,3,1,0] then VT(e)<VT(e’)

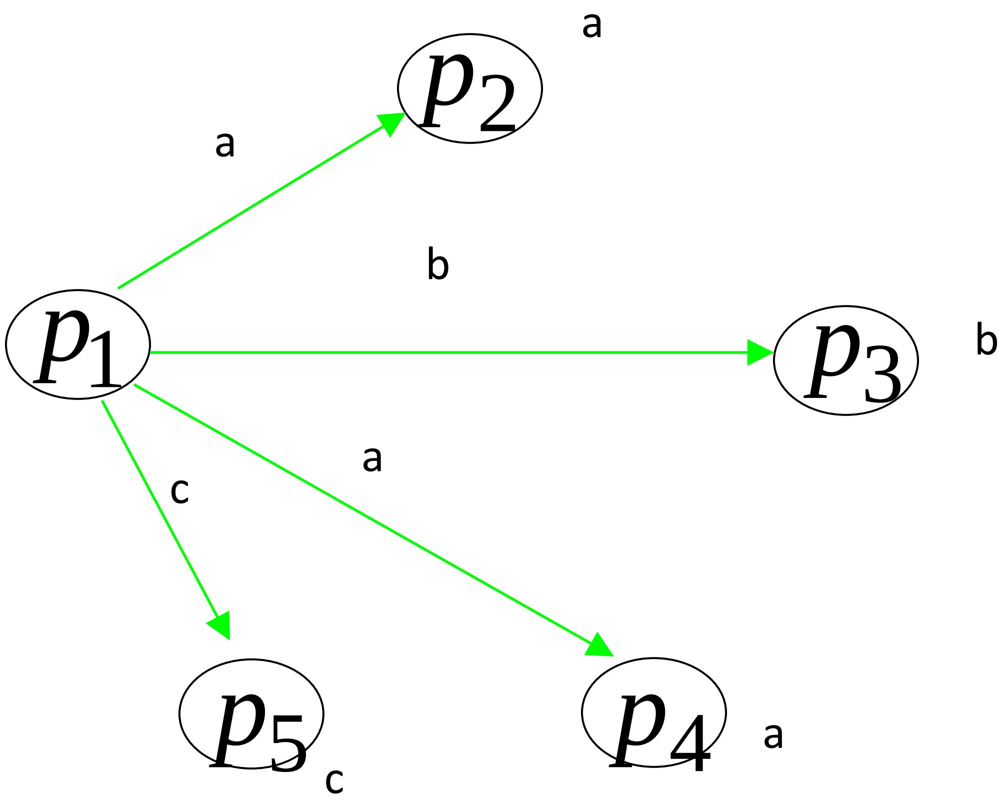
- Notice that not all VT’s are “comparable” under this rule: consider [4,0,0,0] and [0,0,0,4]

# Failures in Distributed Systems

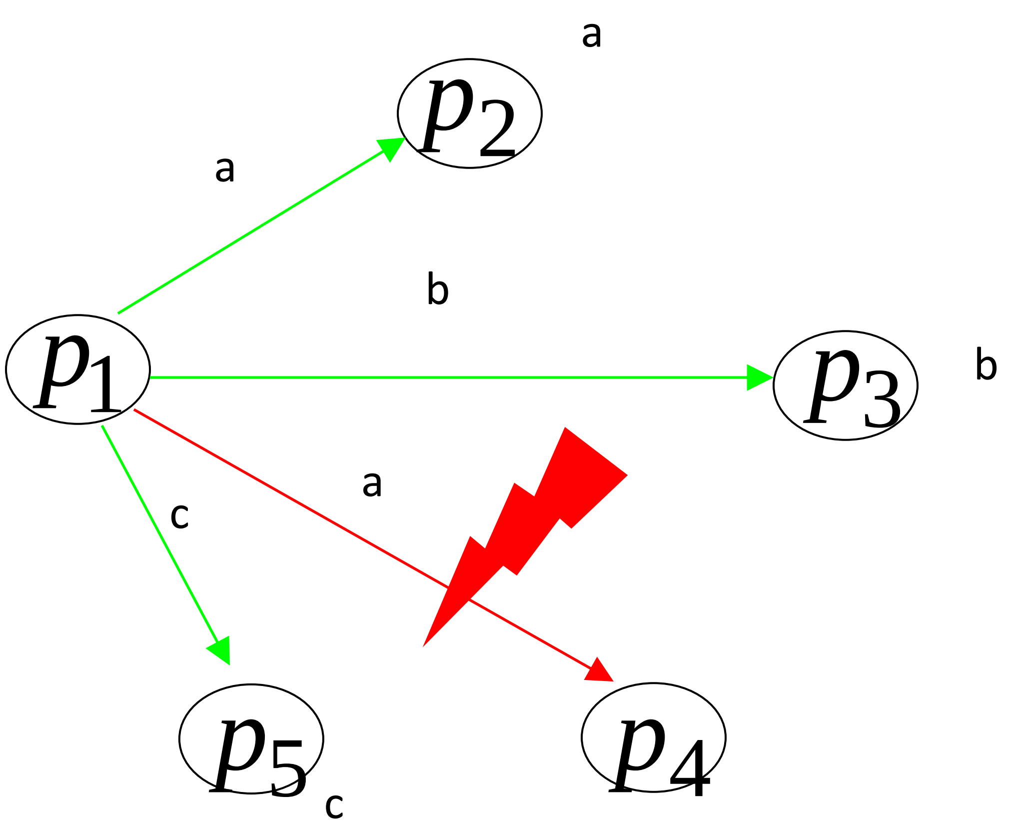
- **Link failure:** A link fails and remains inactive; the network may get disconnected
- **Processor Crash:** At some point, a processor stops taking steps
- **Byzantine processor:** processor changes state arbitrarily and sends messages with arbitrary content

# Link Failures

Non-faulty
links



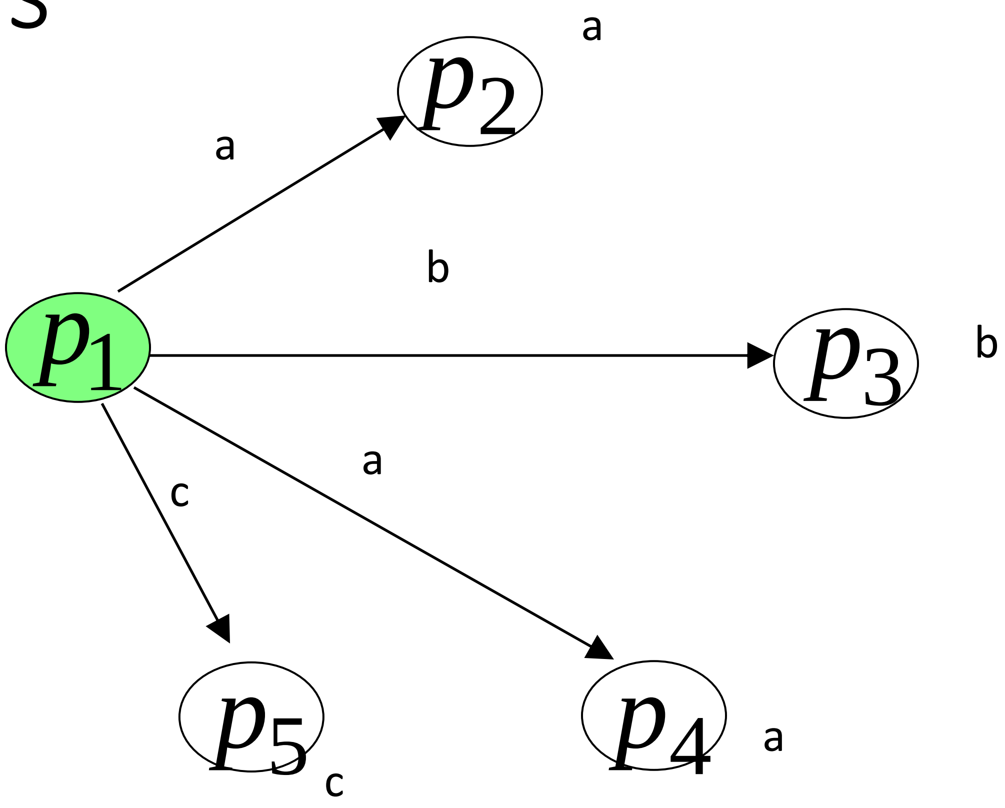
Faulty link



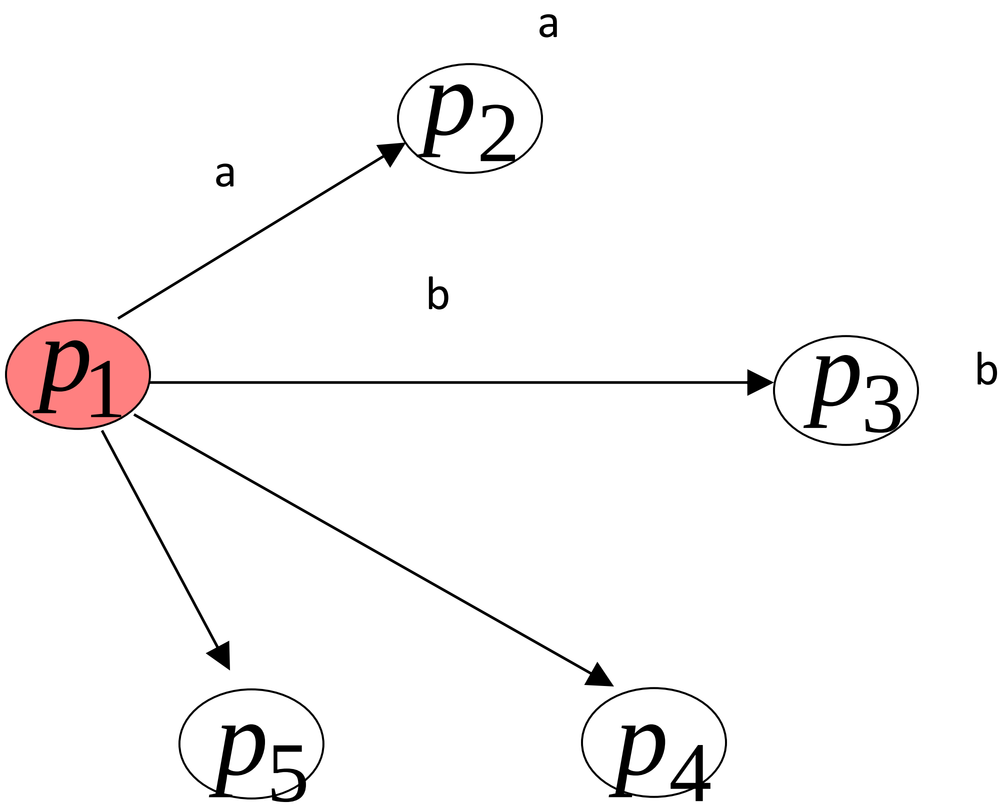

Some of the messages are not delivered

# Crash Failures

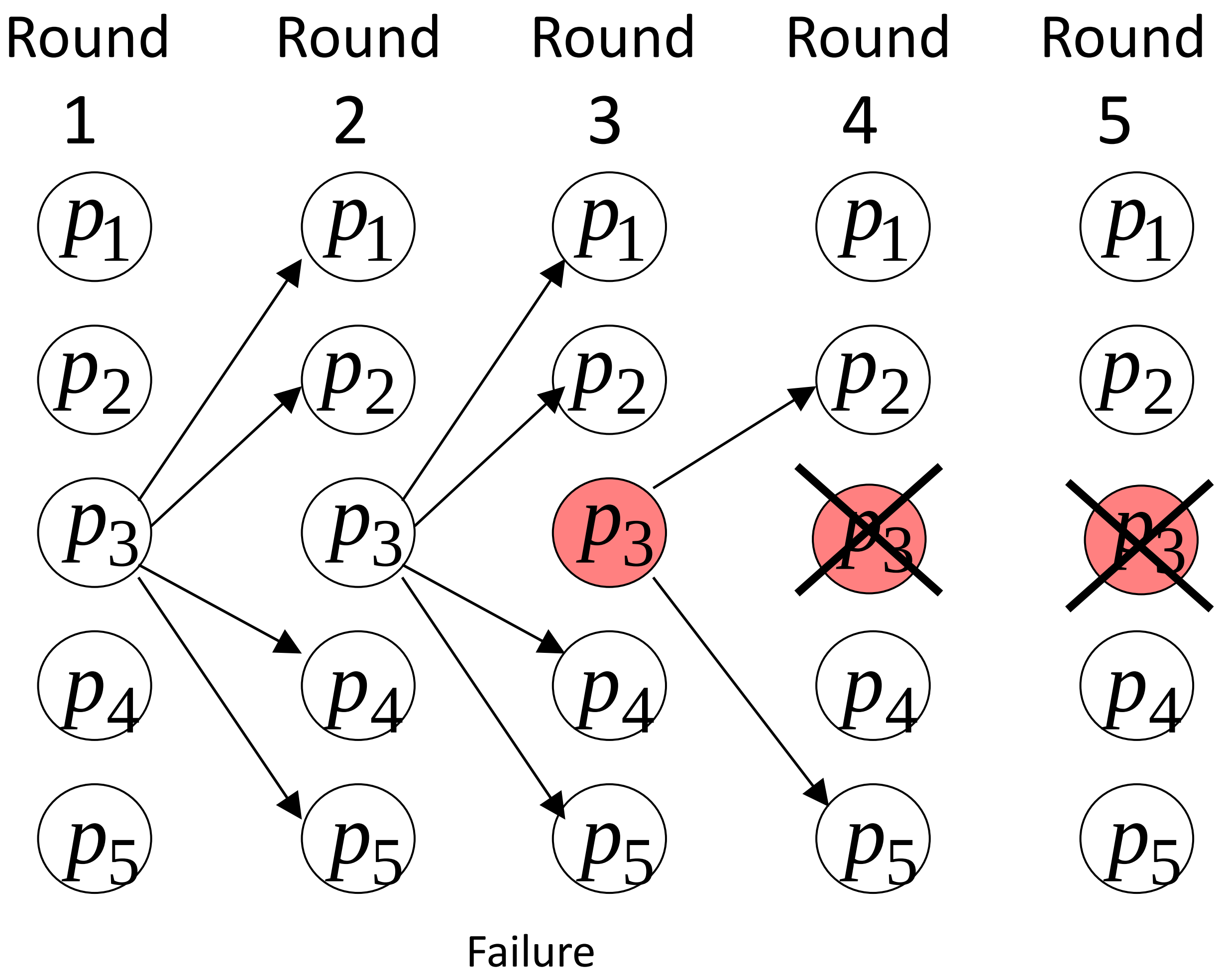
Non-faulty processor



Faulty processor




Some of the messages are not sent



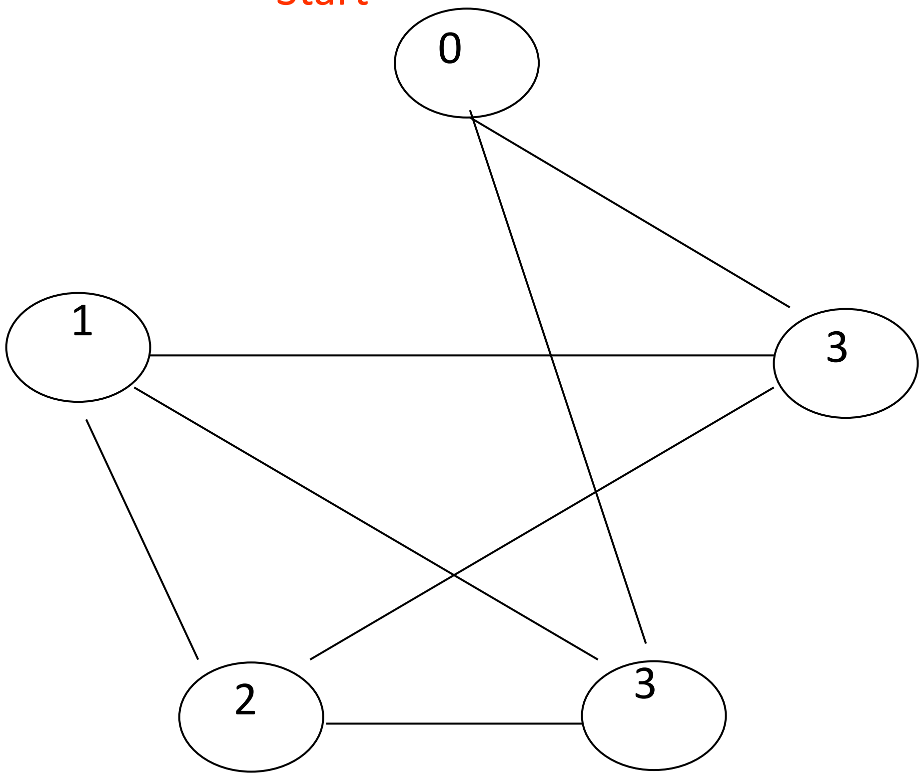

After failure the processor disappears from the network

# Consensus Problem

- Every processor has an input $x \in X$
- *Termination:* Eventually every non-faulty processor must decide on a value $y$.
- *Agreement:* All decisions by non-faulty processors must be the same.
- *Validity:* If all inputs are the same, then the decision of a non-faulty processor must equal the common input (this avoids trivial solutions).

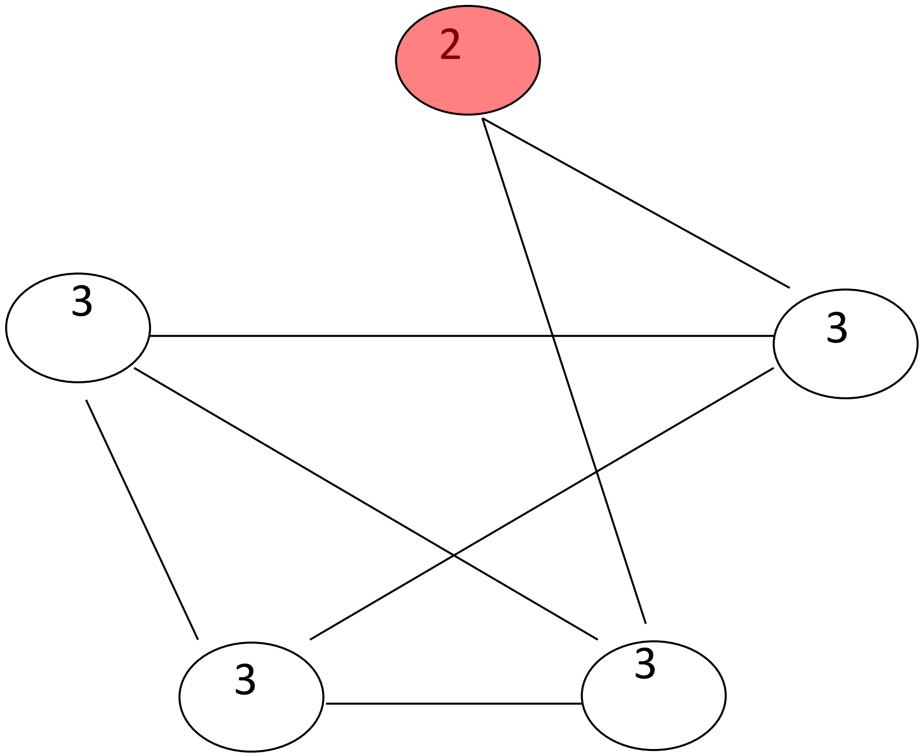
# Agreement

Start




Everybody has an initial value

Finish

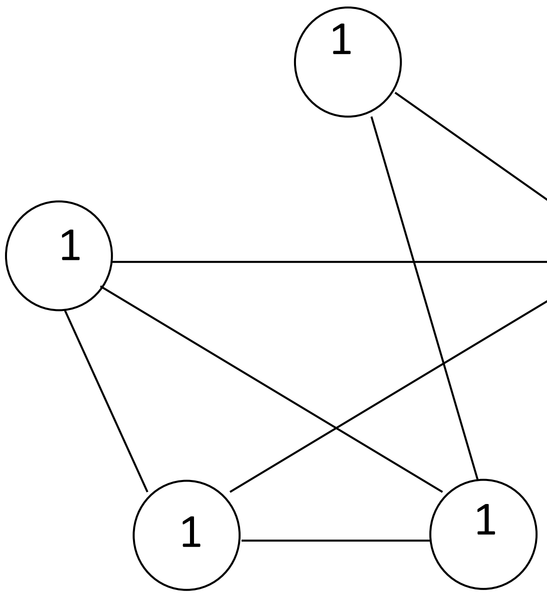



All non-faulty must decide the same value

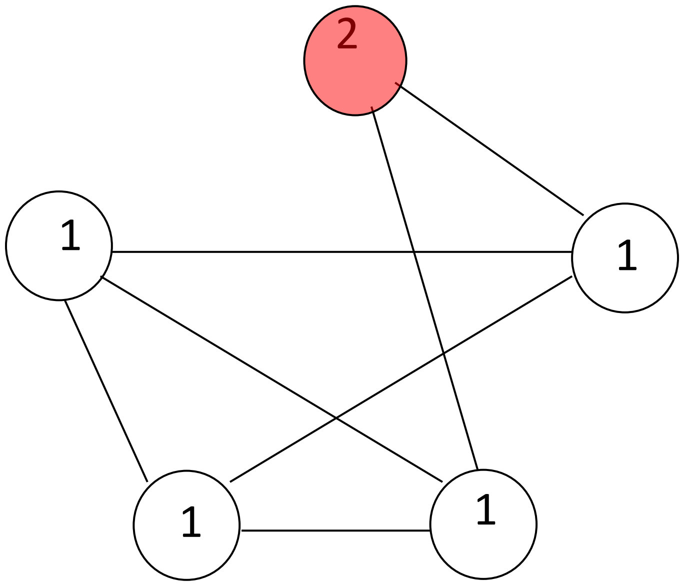
# Validity

If everybody starts with the same value, then non-faulty must decide that value

Start




Finish



# A simple algorithm for fault-free consensus

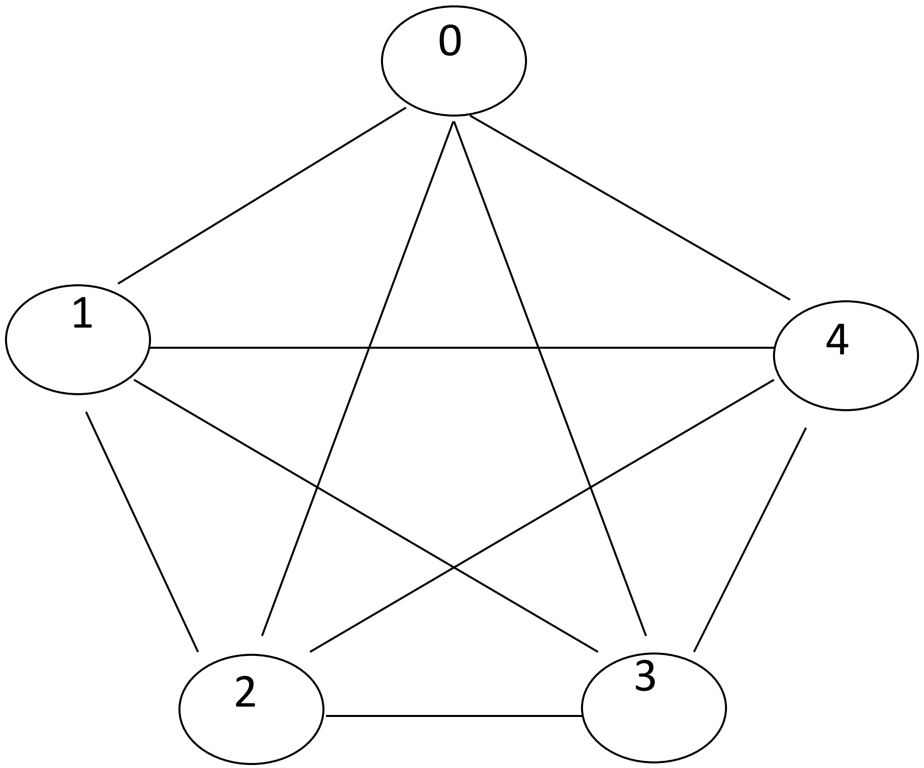
Each processor:

1. Broadcast its input to all processors

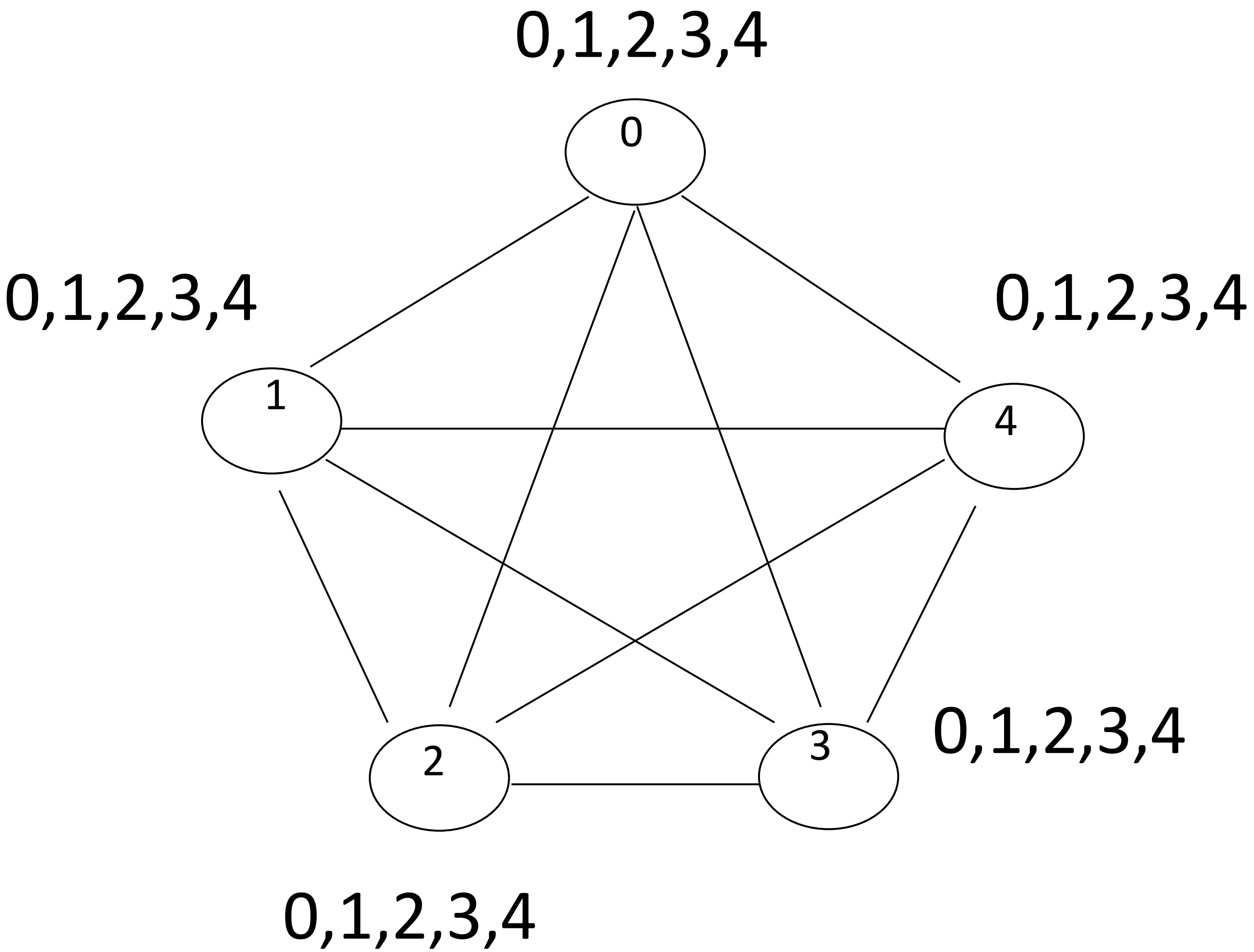
2. Decide on the minimum

(only one round is needed, since the graph is complete)

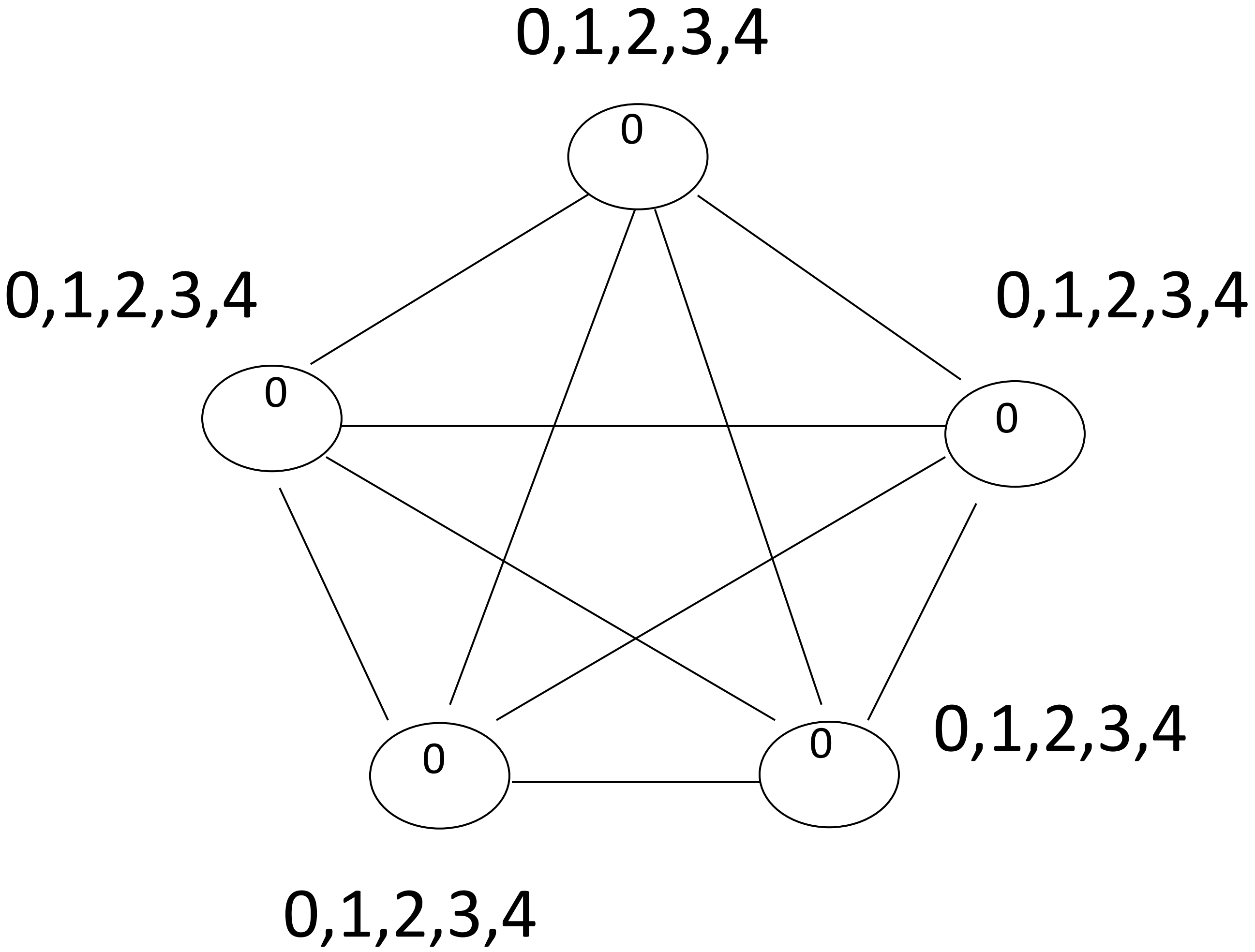
# Start



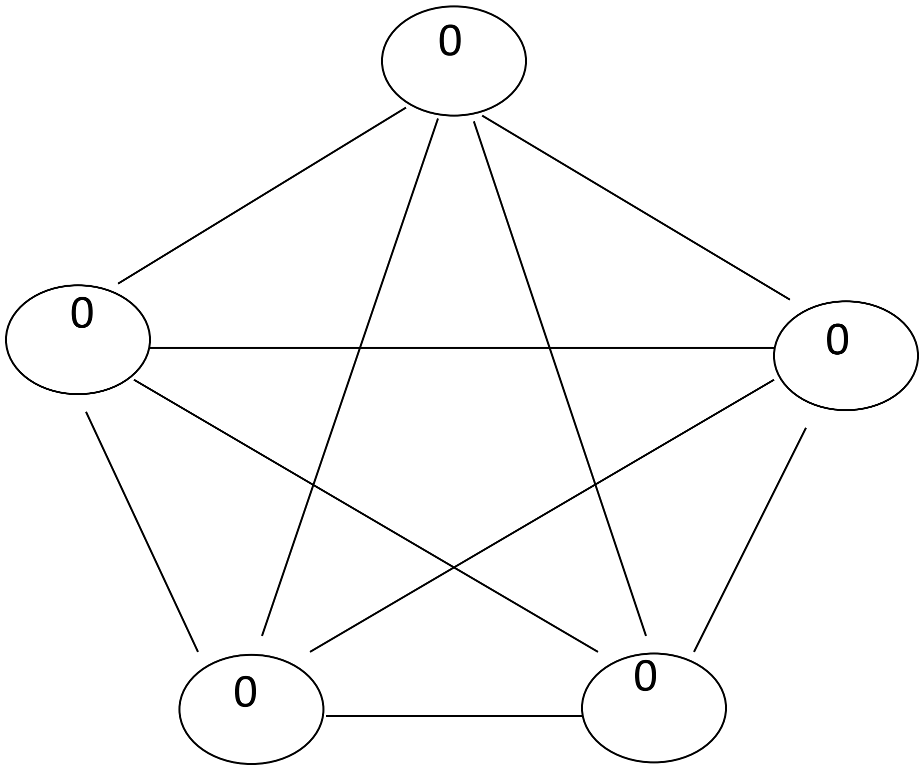
# Broadcast values



# Decide on minimum

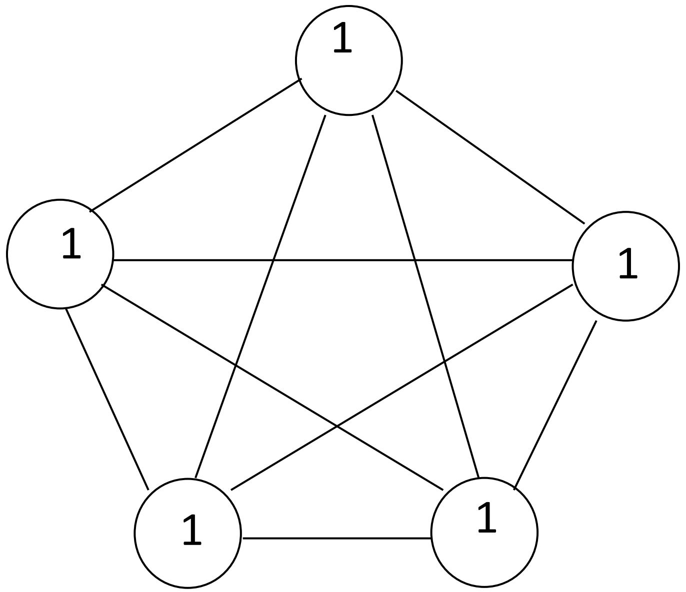


# Finish

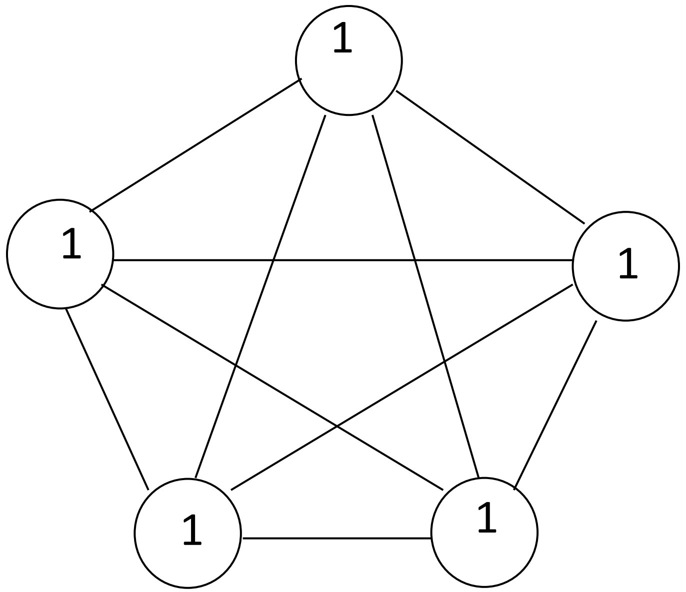


# This algorithm satisfies the validity condition

Start




Finish




If everybody starts with the same initial value, everybody decides on that value (minimum)

# Consensus with Crash Failures

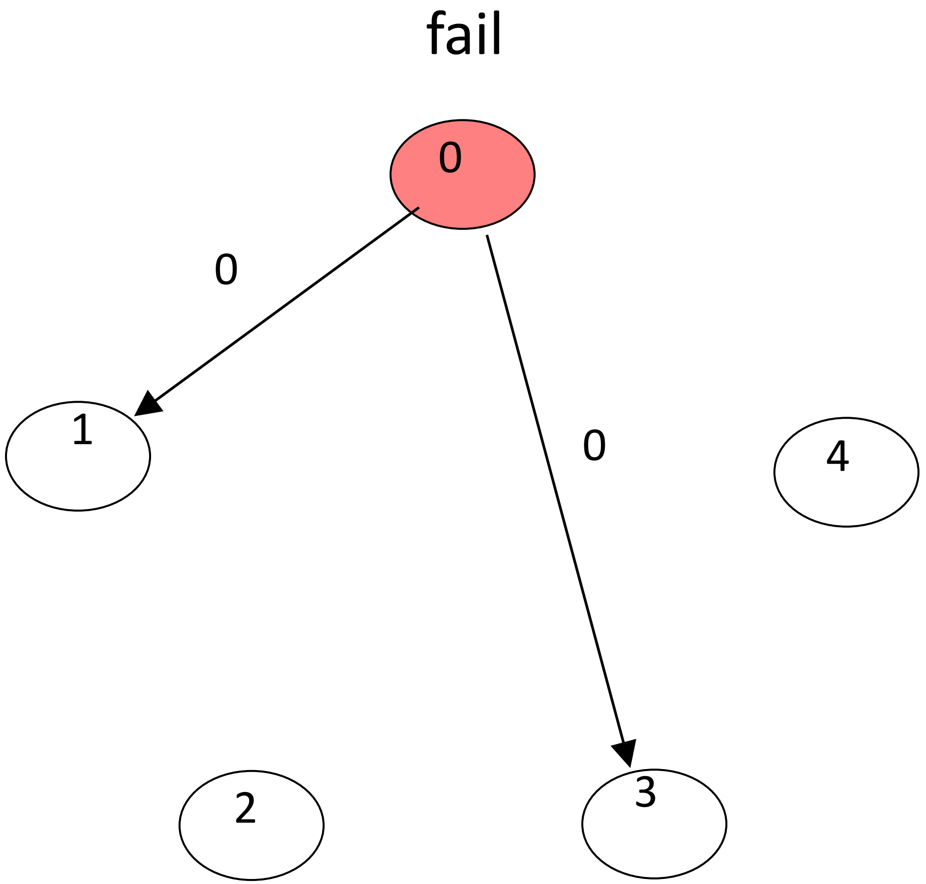
The simple algorithm doesn’t work

Each processor:

1. Broadcast value to all processors

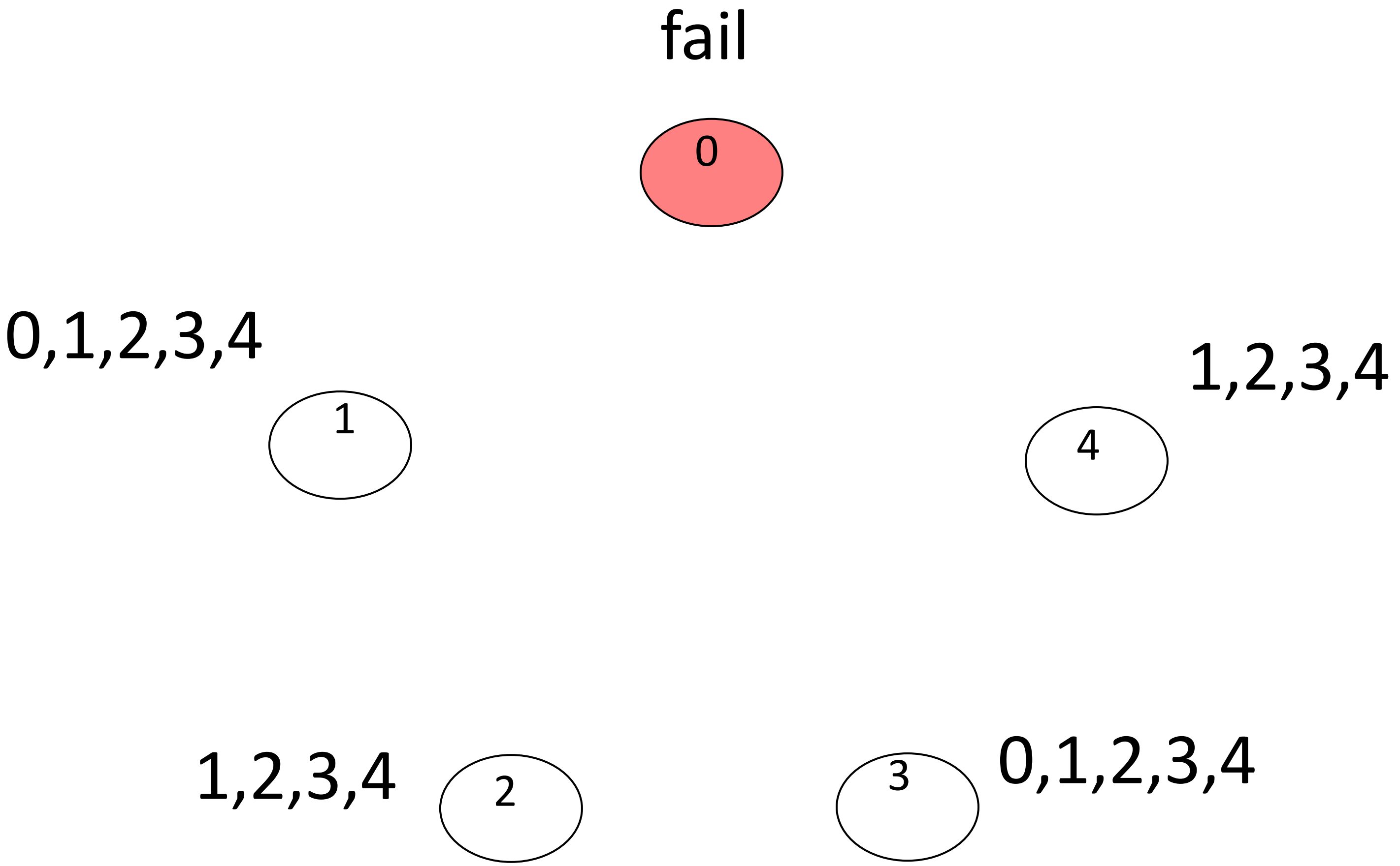
2. Decide on the minimum

# Start

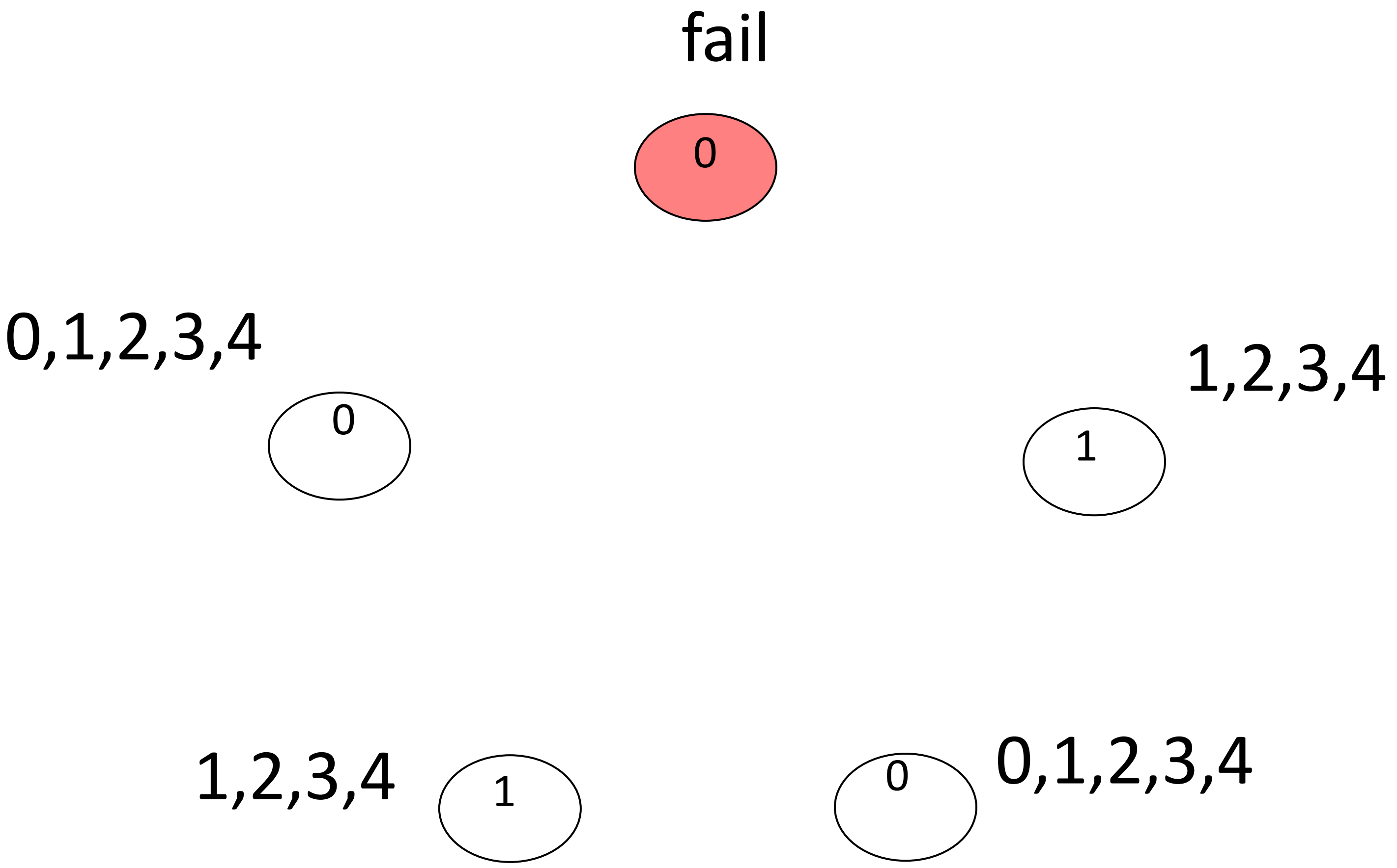



The failed processor doesn’t broadcast its value to all processors

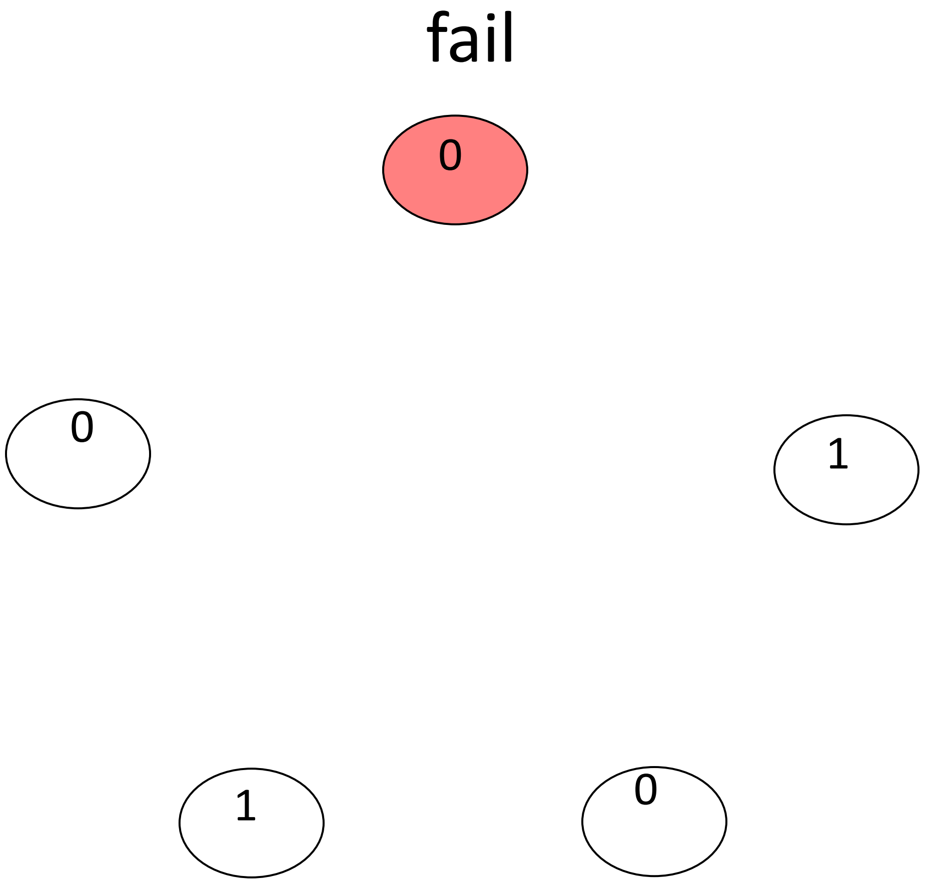
# Broadcasted values



# Decide on minimum



# Finish




No Consensus!!!

If an algorithm solves consensus for
f failed (crashing) processors we say it is:

an f-resilient consensus algorithm

# An f-resilient algorithm

Round 1:

Broadcast my value

Round 2 to round f+1:

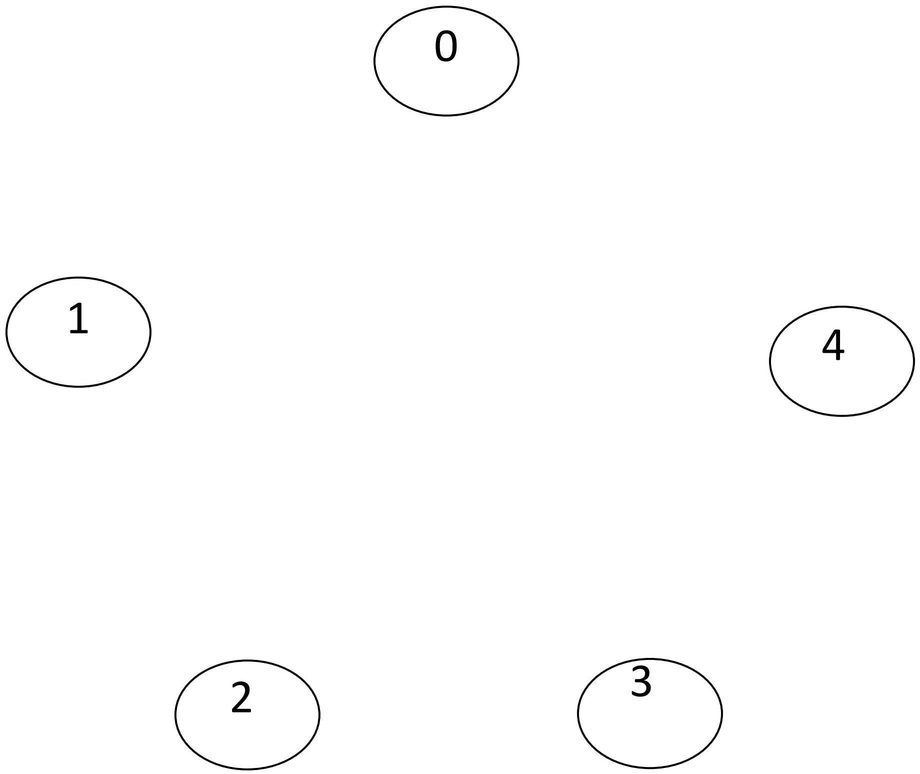
Broadcast any new received values

End of round f+1:

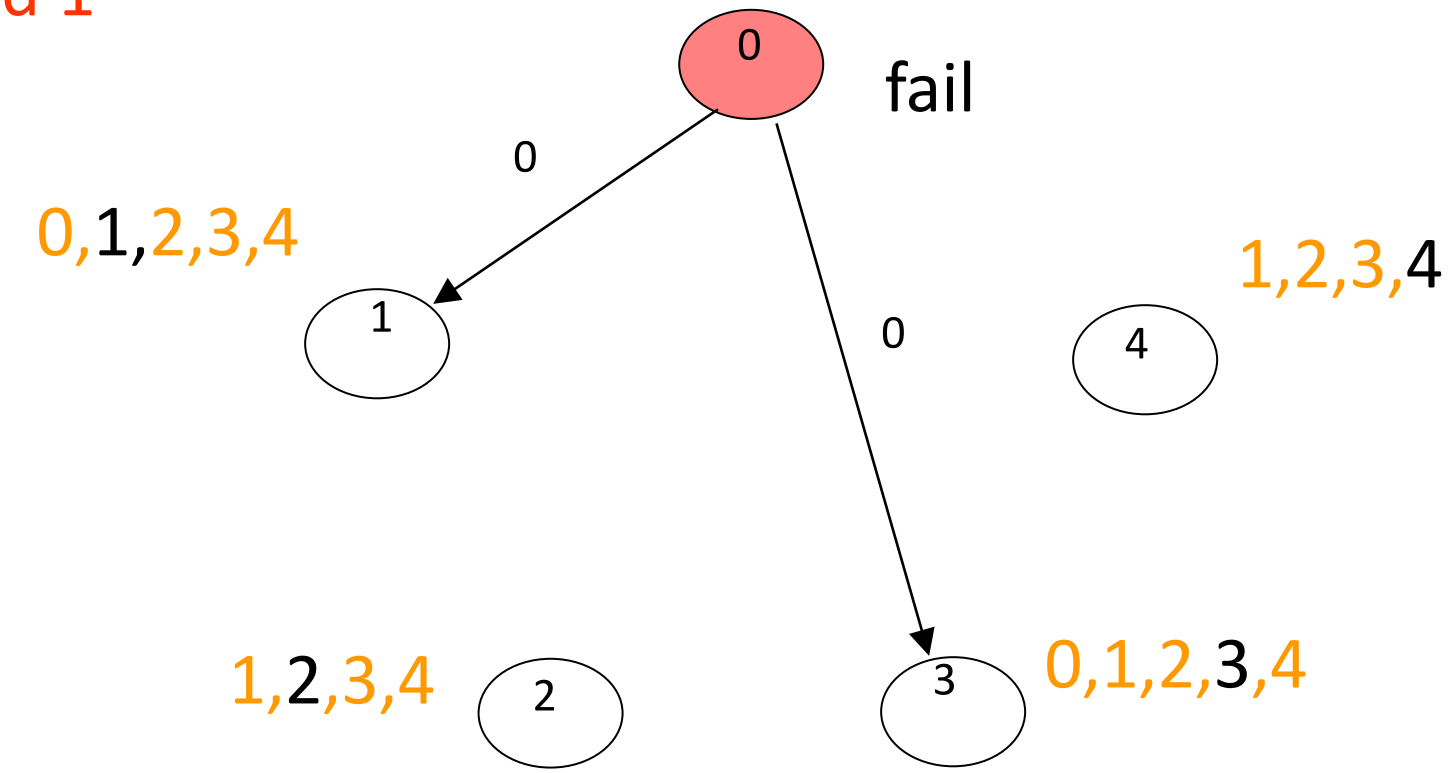
Decide on the minimum value received

# Example: f=1 failures, f+1 = 2 rounds needed

Start

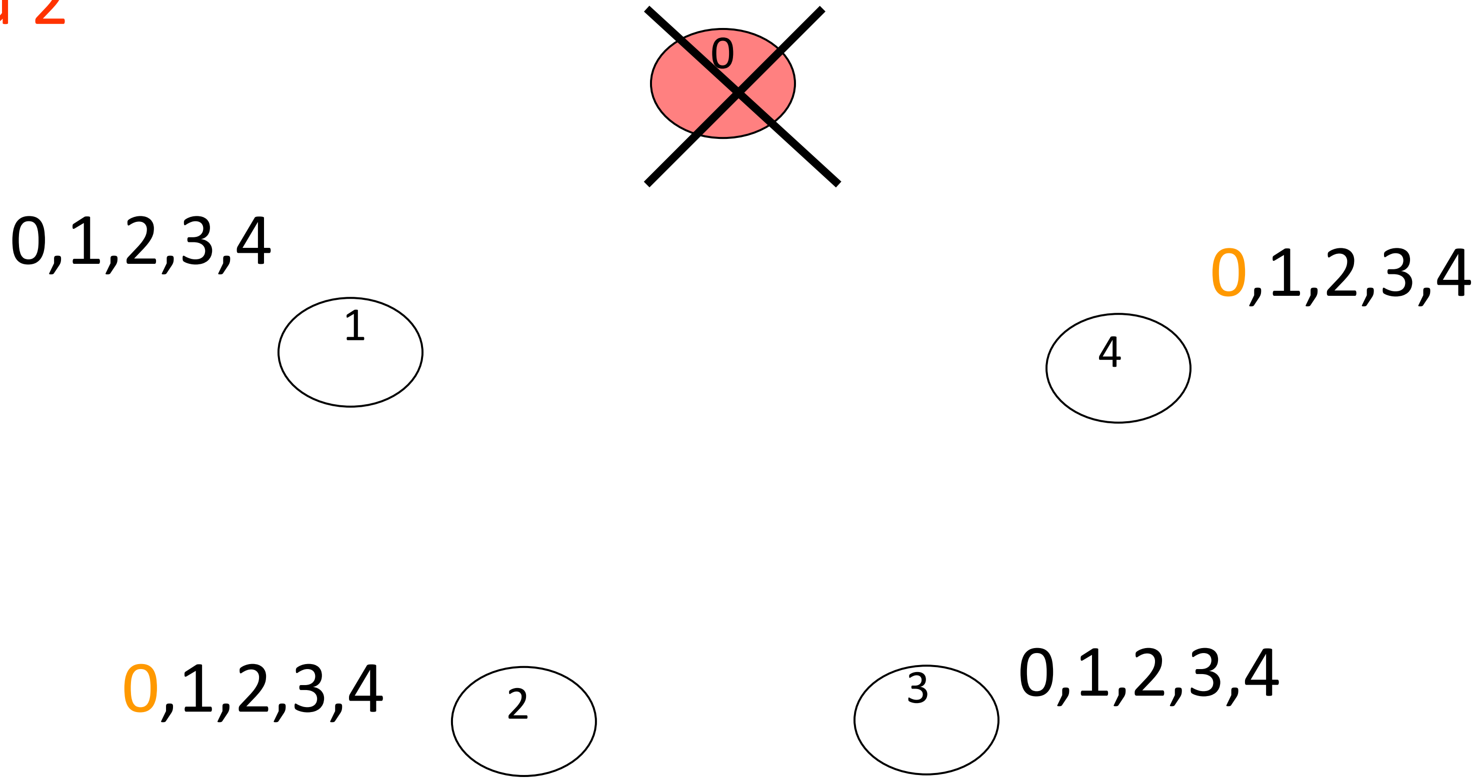


Round 1



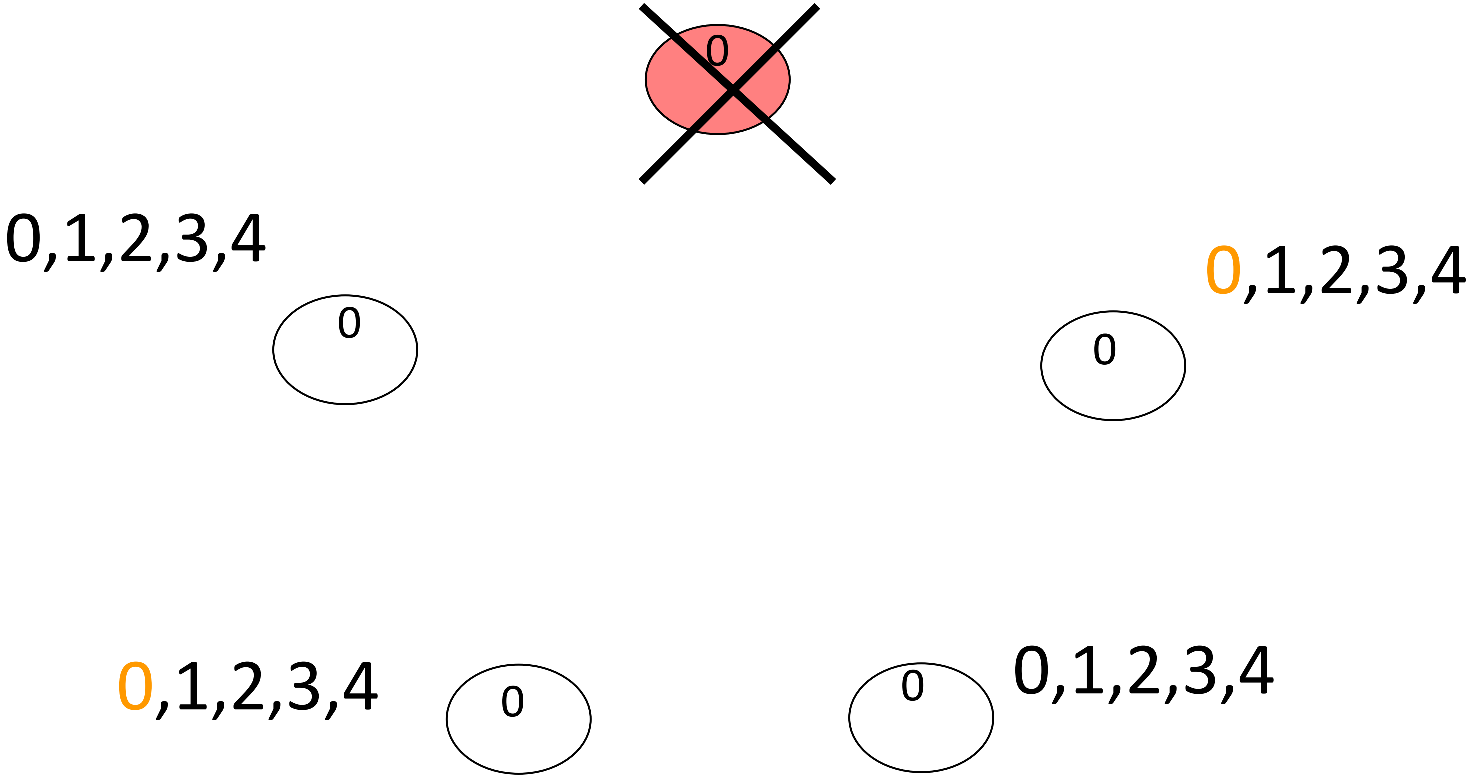

Broadcast all values to everybody

Round 2




Broadcast all new values to everybody

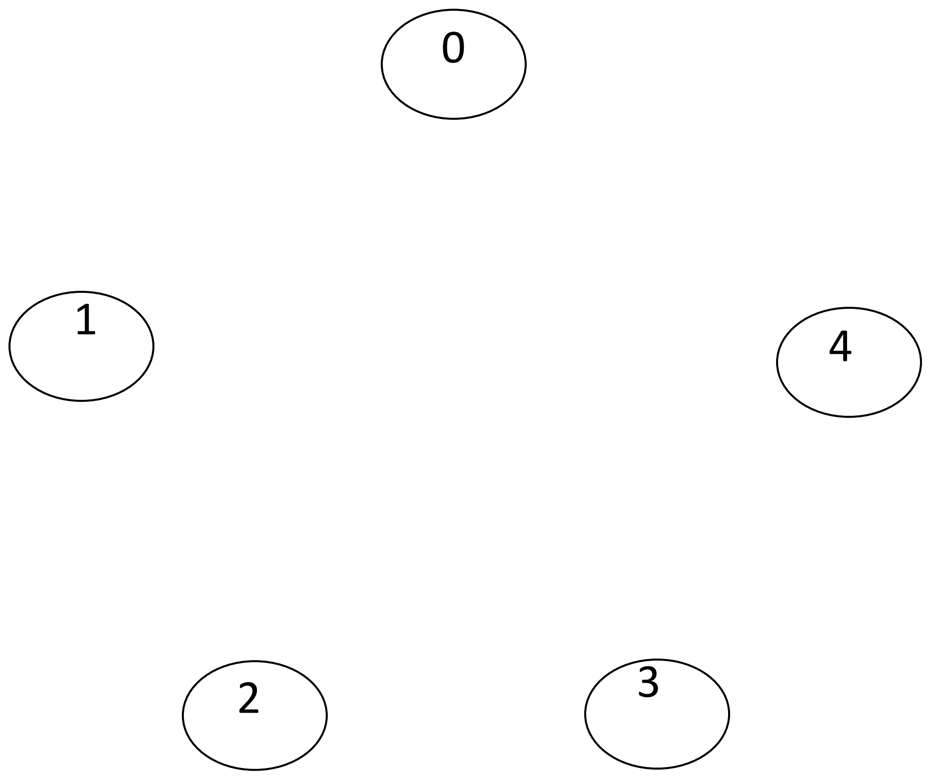
Finish



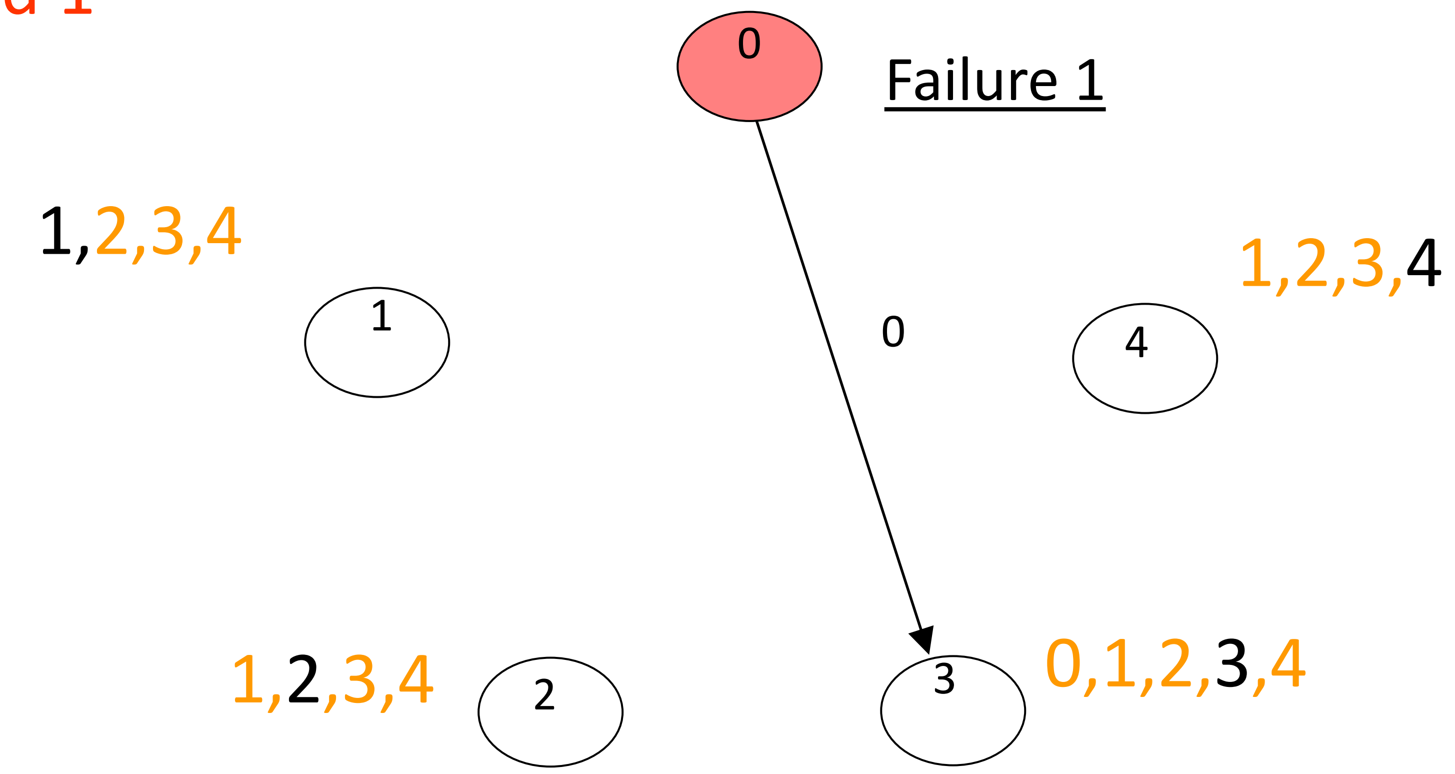

Decide on minimum value

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

Start

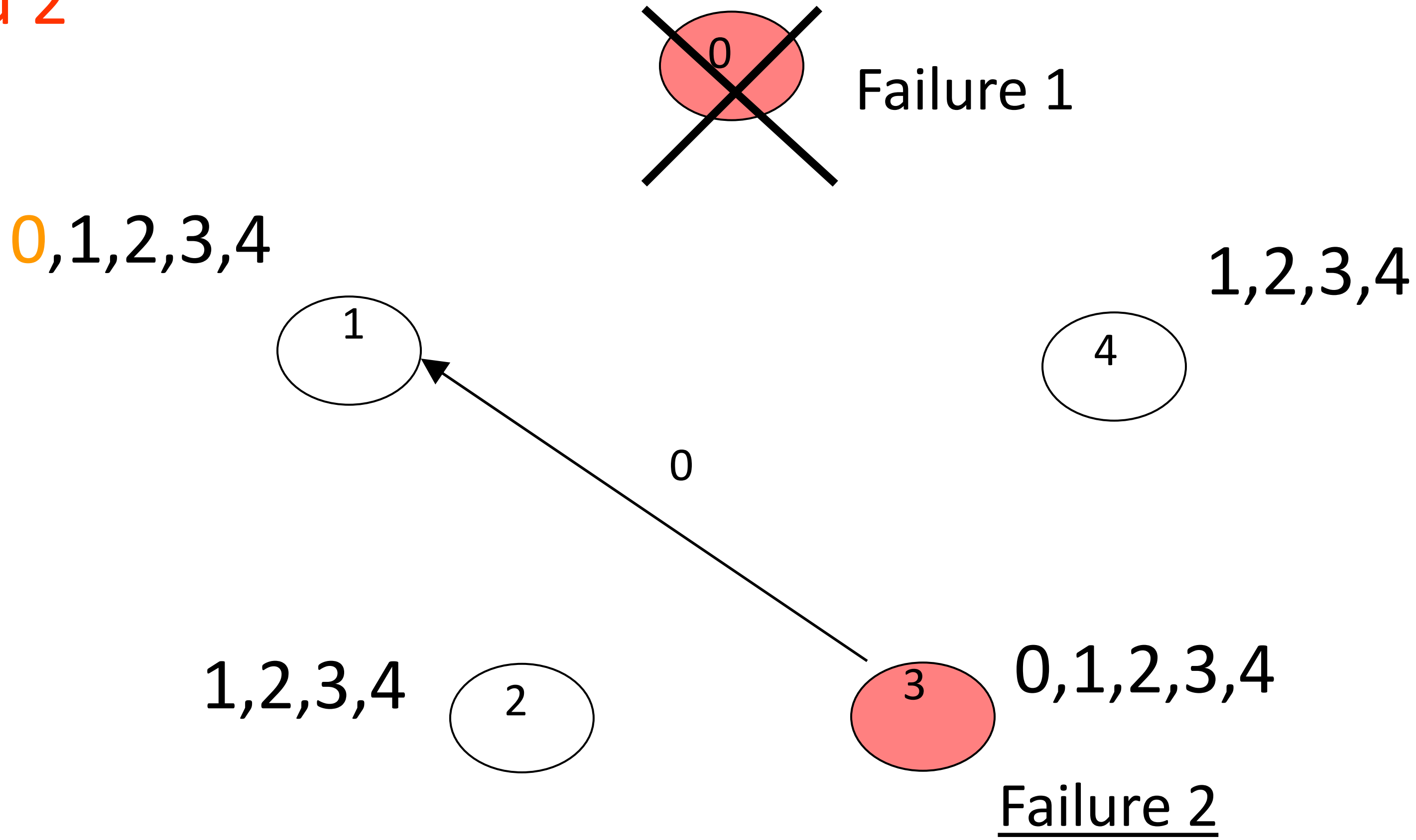


## Round 1



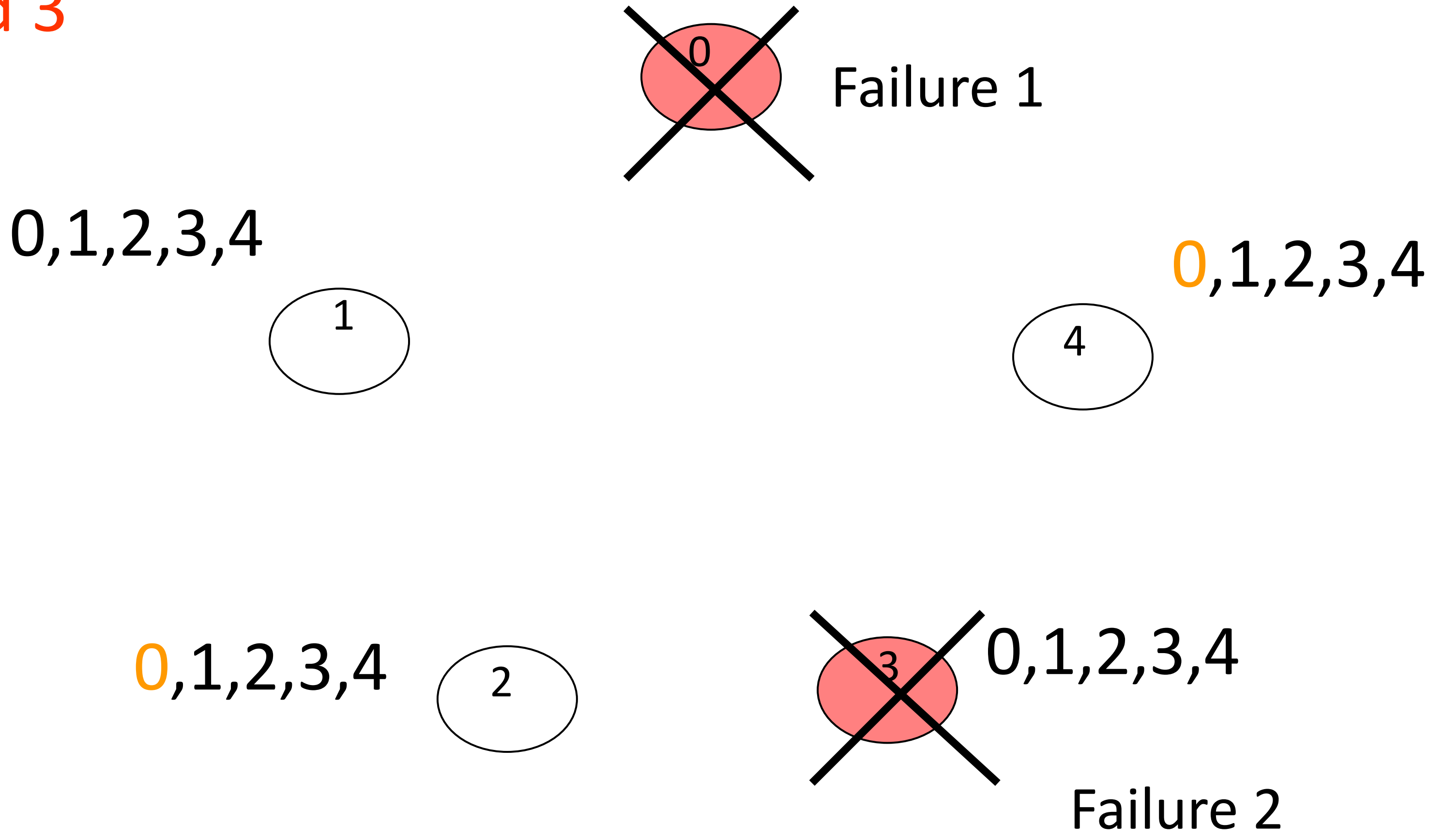

Broadcast all values to everybody

# Round 2



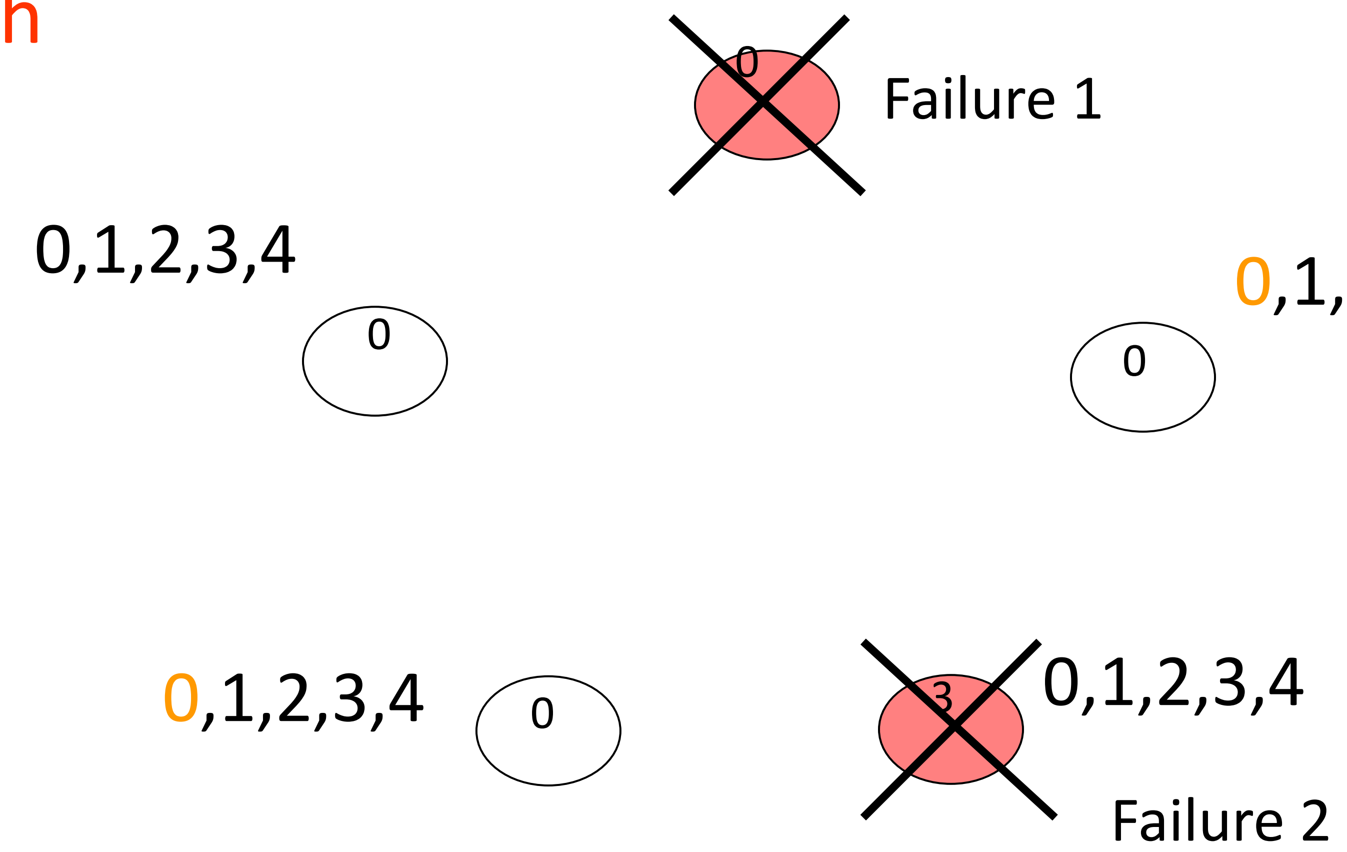

Broadcast new values to everybody

# Round 3




Broadcast new values to everybody

# Finish




Decide on the minimum value

- Lemma: In the algorithm, at the end of the round with no failure, all the processors know the same set of values.

- Proof: For the sake of contradiction, assume the claim is false. Let x be a value which is known only to a subset of (non-faulty) processors. But when a processor knew x for the first time, in the next round it broadcasted it to all. So, the only possibility is that it received it right in this round, otherwise all the others should know x as well. But in this round there are no failures, and so x must be received by all.

# Quiz 05 (8 minutes) – BDS-6A

1. From the RPC call semantics, we have “at-least once” semantics, can you describe a real-world scenario where this would be suitable (3m)

2. In the context of RPC, what is an IDL? Can you provide an example of an IDL? (4m)

3. Why is there a need for logical clocks within distributed systems? (3m)

# Quiz 05 (8 minutes) – BCS-6D

1. From the RPC call semantics, we have “at-most once” semantics, can you describe a real-world scenario where this would be suitable (3m)

2. In the context of RPC, what is the role of a stub, and where is it used? (4m)

3. Why is there a need for logical clocks within distributed systems? (3m)

# Quiz 05 (8 minutes) – BCS-6C

1. From the RPC call semantics, we have “at-least once” semantics, can you describe a real-world scenario where this would be suitable (3m)

2. In the context of RPC, what is an IDL? Can you provide an example of an IDL? (4m)

3. Why is there a need for logical clocks within distributed systems? (3m)

# References

1. Slides of Dr. Haroon Mahmood

Helpful Links: