UC DAVIS
COLLEGE OF ENGINEERING

Asynchronous Byzantine Agreement Protocols
Gabriel Bracha

Presenters : Devang, Prasannadatta, Ajinkya, Arnav

What is Consensus

Consensus is the problem of reaching agreement on a single value among a set of distributed agents.

- Enabling agreement among processes, even with some faulty ones.
- Utilizing quorums to guarantee the persistence of decisions even when system failure occurs.
- Consensus mechanisms, such as leader-based and leader-less, employ varying approaches to achieve agreement.

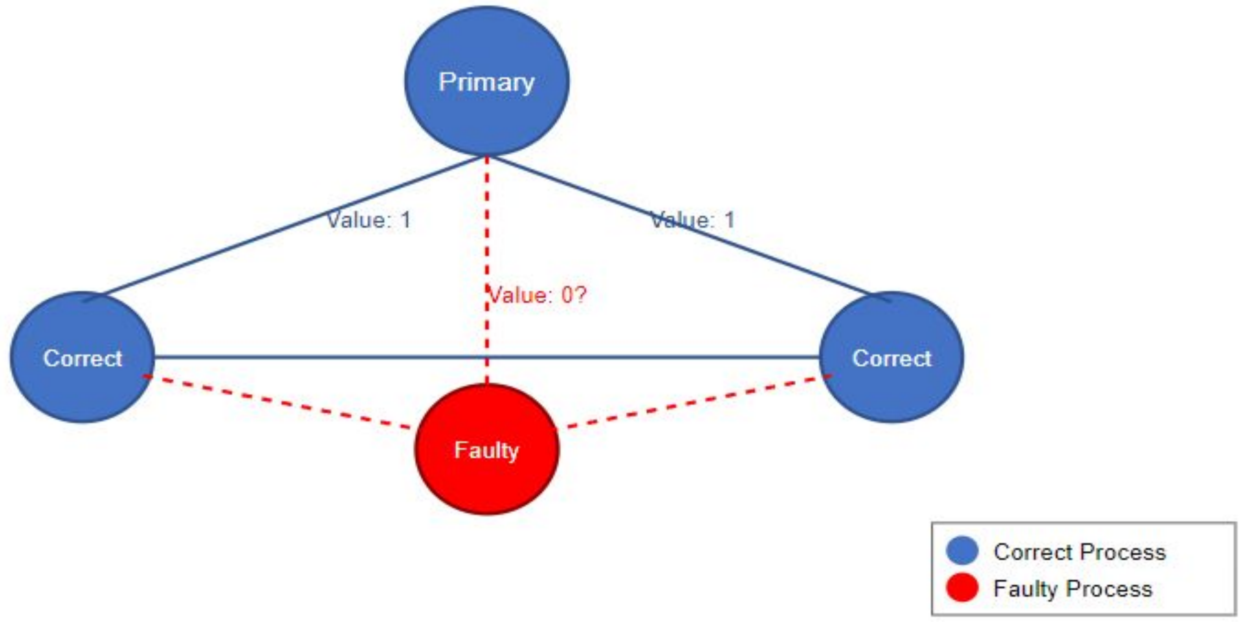
Consensus Properties

3 Main Properties for Consensus - Agreement, Validity, Termination

1. Agreement - All correct processes in the system eventually decide on the same value.
2. Validity - If all correct processes start with the same value, then all correct processes decide on that value.
3. Termination(Liveness) - This property guarantees every non-faulty process eventually reaches a decision.

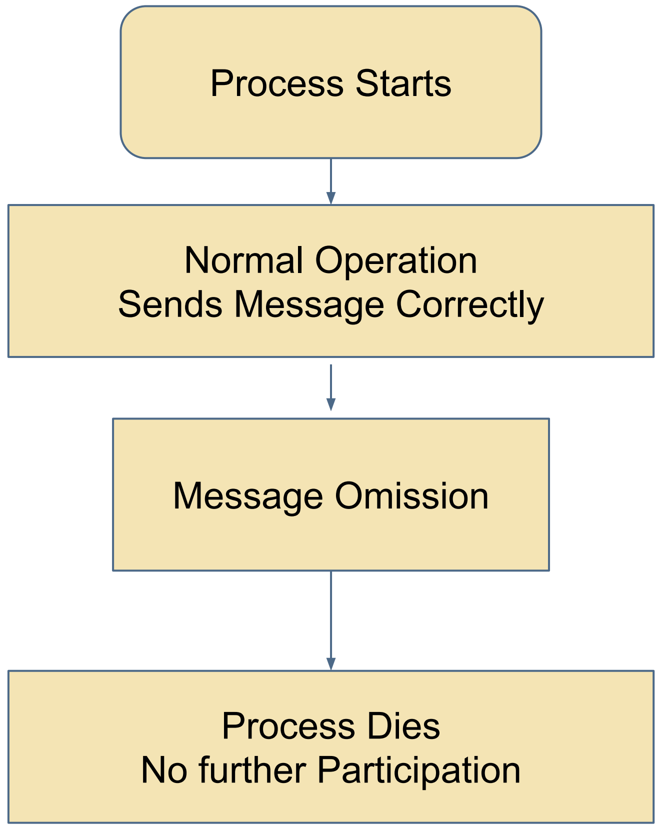
Distributed Consensus with Byzantine Faults

- All correct processes have to agree on transmitter's broadcast value.
- t-resilient protocols maintain agreement and validity despite up to t faulty processes.



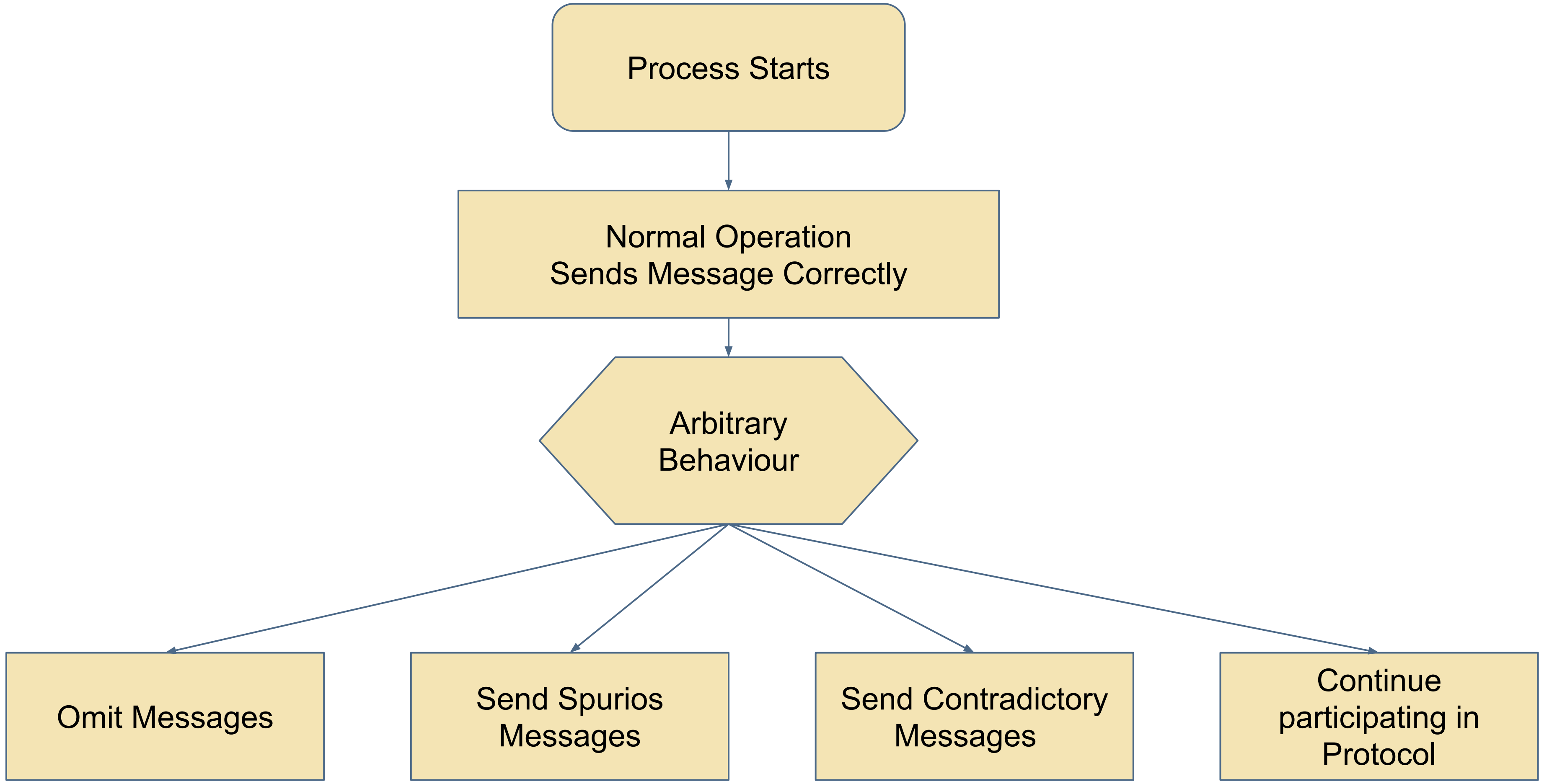
Fail Stop Process

Process omits messages, then ceases all communication. Easily detectable



Byzantine Processes

These processes can deviate from protocol in arbitrary ways. Difficult to detect



Synchronous vs Asynchronous vs Partial Synchrony Systems

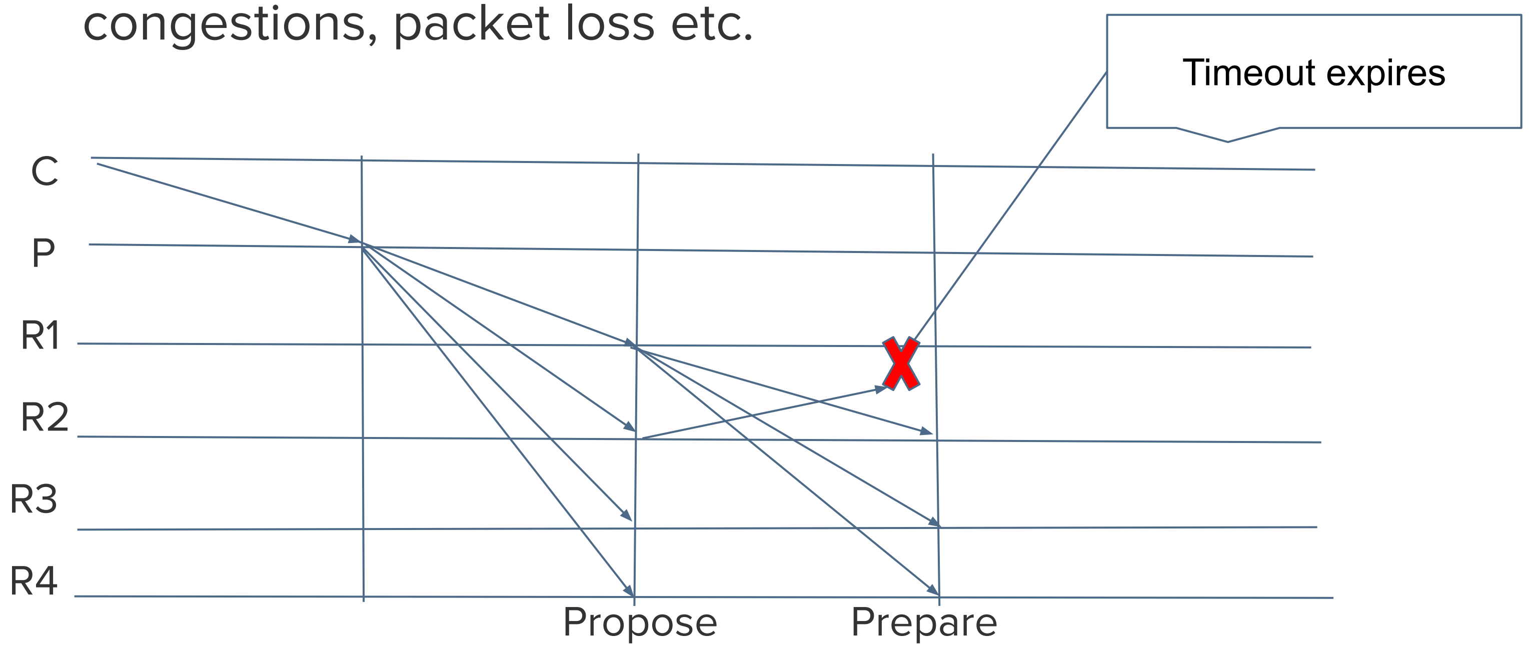
- Synchronous Systems
  - Fixed, known time bounds for message delivery and process execution.
  - Processes operate in predictable, synchronized steps or rounds.

- Asynchronous Systems
  - No guarantees on timing for message delivery or process execution
  - Messages can be delayed indefinitely, with no bounds on delay or processing time.

- Partially Synchronous Systems
  - Unknown time bounds for delivery and process execution.
  - Practical middle ground between both extremes typically used when analyzing availability (liveness)

Problems with Partial Synchrony Systems

- **Unknown Stabilization Time**: Difficult to determine realistic timing bounds as bounds may change over time.
- **Real-World Variability**: Network conditions can change frequently due to delays, congestions, packet loss etc.



Termination

- **Deterministic Termination**: All correct processes decide by round r, for some a priori known constant r.

- **Probabilistic Termination**: The probability that a correct process is undecided after ‘r’ rounds approaches zero as r approaches infinity.

Deterministic Termination is impossible in asynchronous systems due to the inability to differentiate between a slow process and a failed process.

What does this paper do?

- Demonstrates the ability of the protocol to tolerate up to n/3 faulty processes, improving on the previous n/5 threshold
- Addresses how reliable broadcast and correctness enforcement are implemented
- Makes two key findings about Byzantine Generals protocols in asynchronous systems:

    1) Deterministic consensus is impossible even with a single fault

    2) With weak termination, they are possible if and only if $t < n/3$

Broadcast Primitive

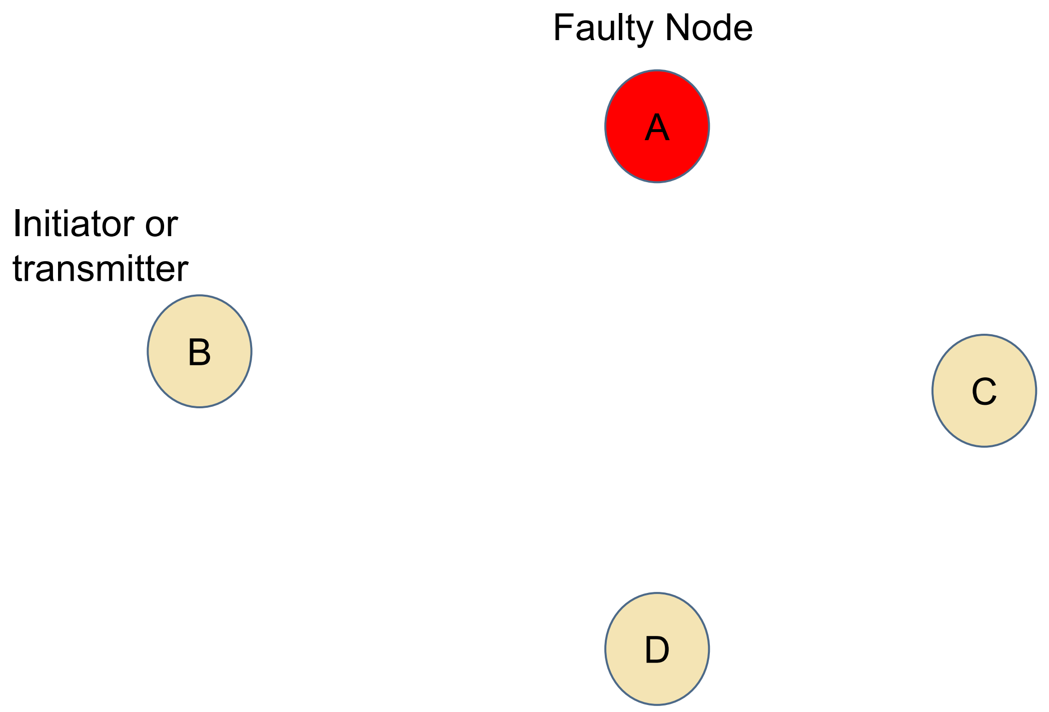
- Used for reliable and consistent communication among nodes in a Byzantine fault-tolerant system.
- Reduces the influence of Byzantine nodes by forcing them to act similarly to Fail-Stop processes, meaning they either send a consistent message to all nodes or send no message at all.
- Consistency is ensured by requiring message validation by multiple nodes before acceptance.

Validation Mechanism

- It works by requiring nodes to reach a threshold of confirmations before accepting a message.
- For example, nodes require confirmation from 2f + 1 other nodes before validation (where f is the maximum number of faulty nodes tolerated).
- Messages that do not receive enough confirmations are discarded or ignored, preventing them from influencing the consensus process.
- Fault tolerance is reinforced by ensuring that nodes only accept messages that have been agreed upon by other correct nodes and thus ignores the faulty nodes.

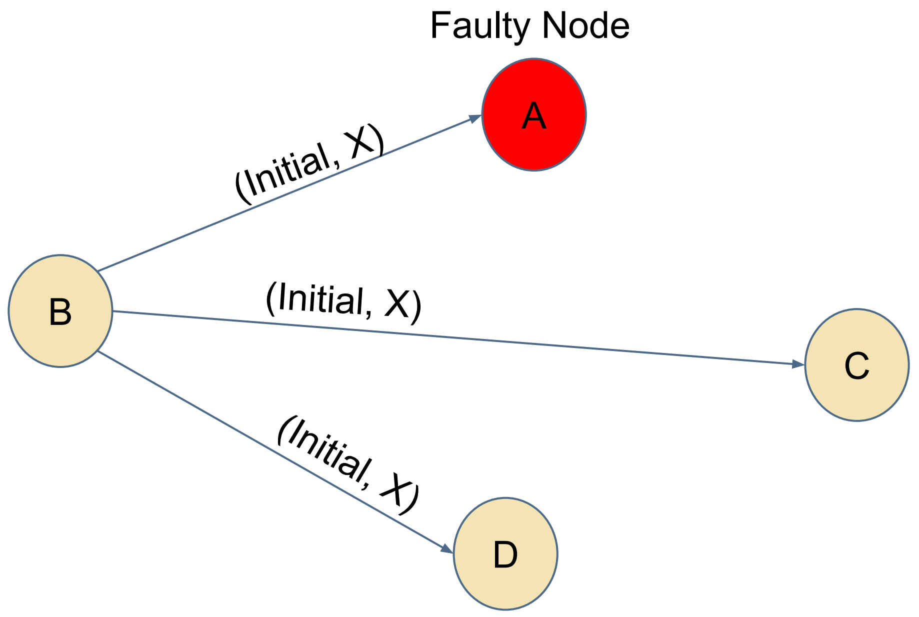
.

Graphical Illustration



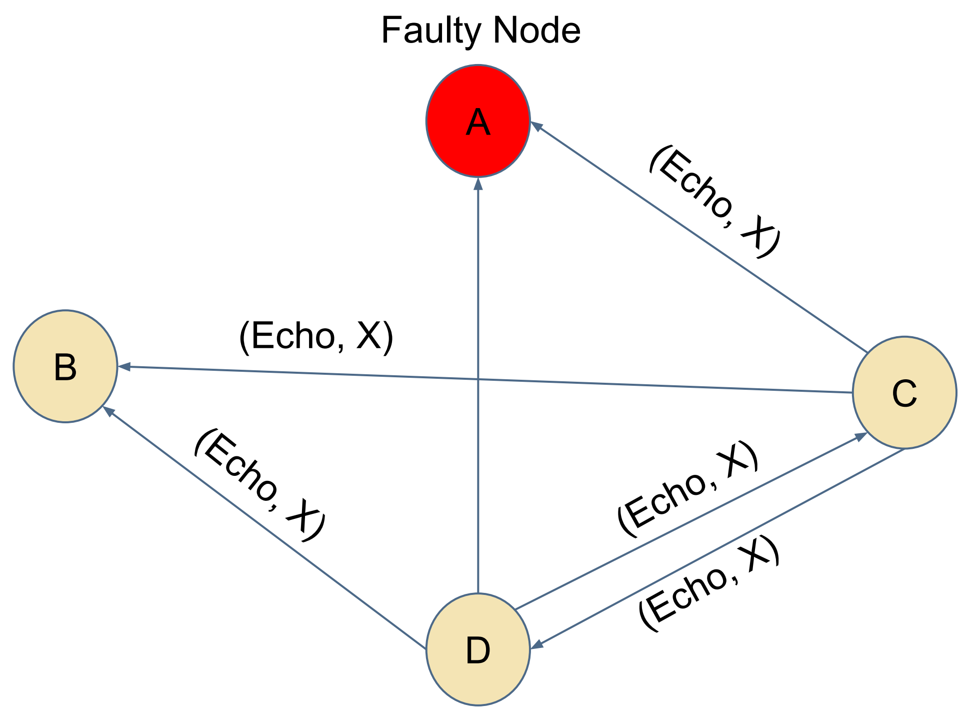
- **Participants**: There are four processes in the system: A, B, C, and D.
- **Faulty Node**: Node A behaves arbitrarily (Byzantine node) and sends incorrect or inconsistent messages.
- **Objective**: The goal is for all the correct processes (B, C, D) to agree on a value despite Node A's faulty behavior.

Initial Phase (Broadcast)



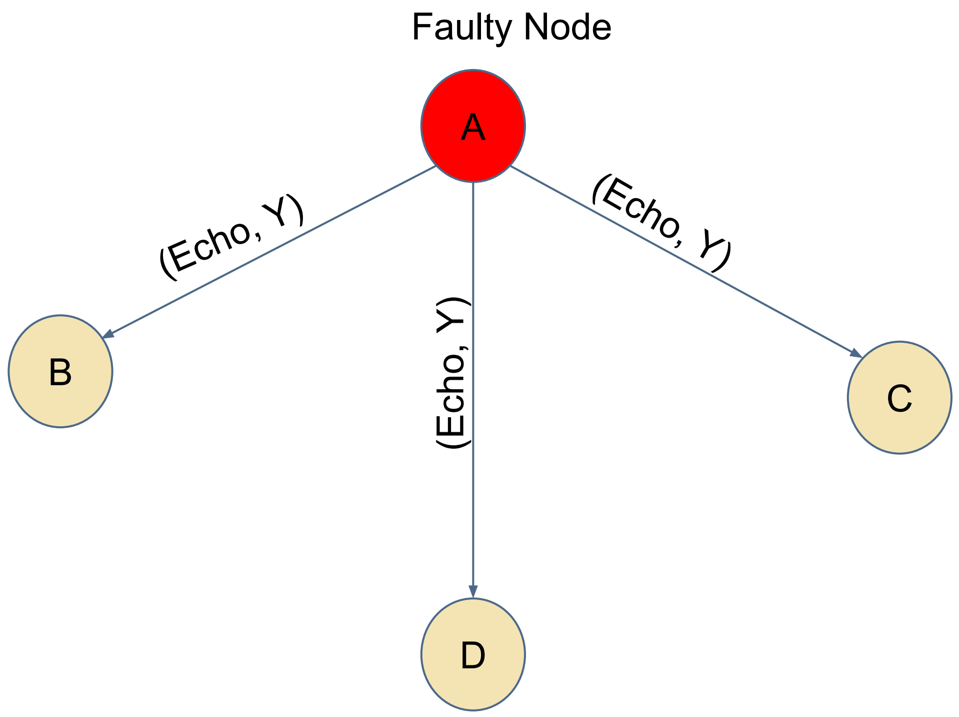
- **Action**: Node B (a correct process) sends an initial message with its proposed value, say **"X"**, to all other processes (A, C, D).
- **Message**: The message is marked as "initial" and contains the value **X**.
- **Purpose**: The goal is for all nodes to become aware of the proposed message and prepare to validate it.
- This phase ensures that all nodes receive the initial message directly from the source.

Echo Phase



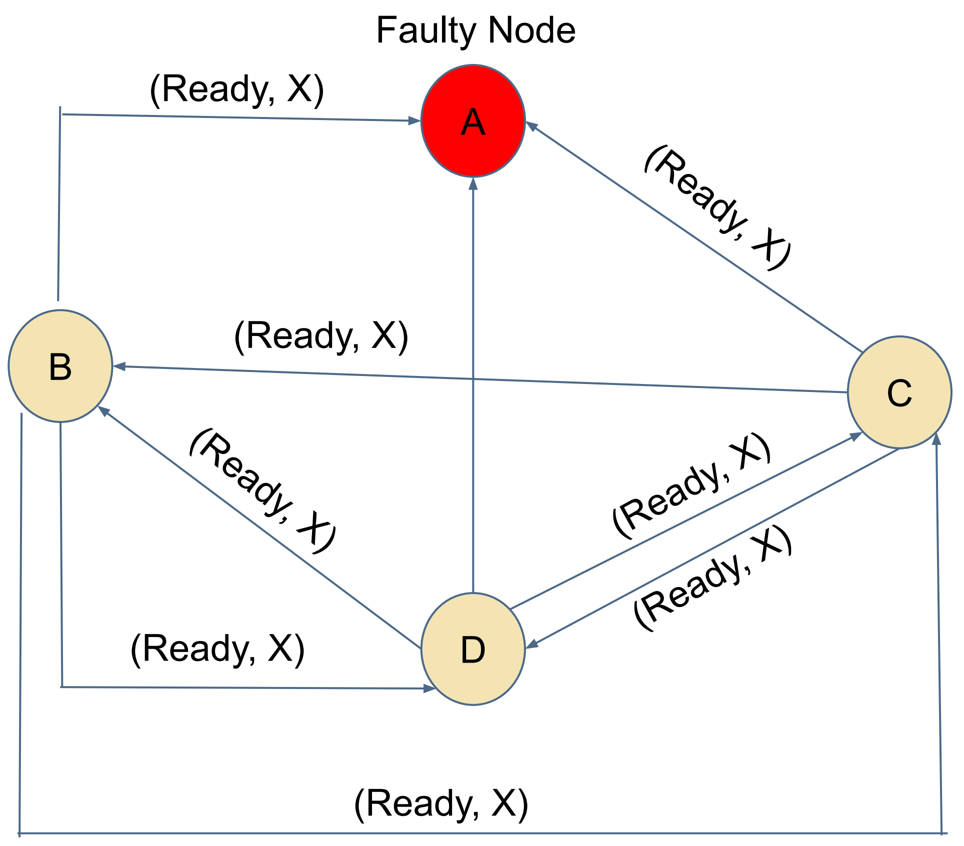
- **Action**: Nodes C and D (correct nodes) receive the initial message from Node B.
- **Message**: Nodes C and D send an "echo" message to each other and to Node B, indicating that they have seen the message and are confirming its validity.
- **Purpose**: The Echo phase acts as the first round of validation, ensuring that each node knows other nodes have received the initial message.
- This phase builds consistency, ensuring that all nodes are aware that a majority received the same initial message.

Echo Phase (Byzantine behavior)



- **Action**: Node A (the faulty node) may try to disrupt the protocol by sending a conflicting or incorrect value (say, **"Y"**) in its echo message.
- **Message**: Node A sends an "echo" message with **"Y"**, contradicting the message that Node B proposed (which was **X**).
- **Purpose**: This is an example of Byzantine behavior. Node A is trying to mislead other nodes by sending inconsistent information.

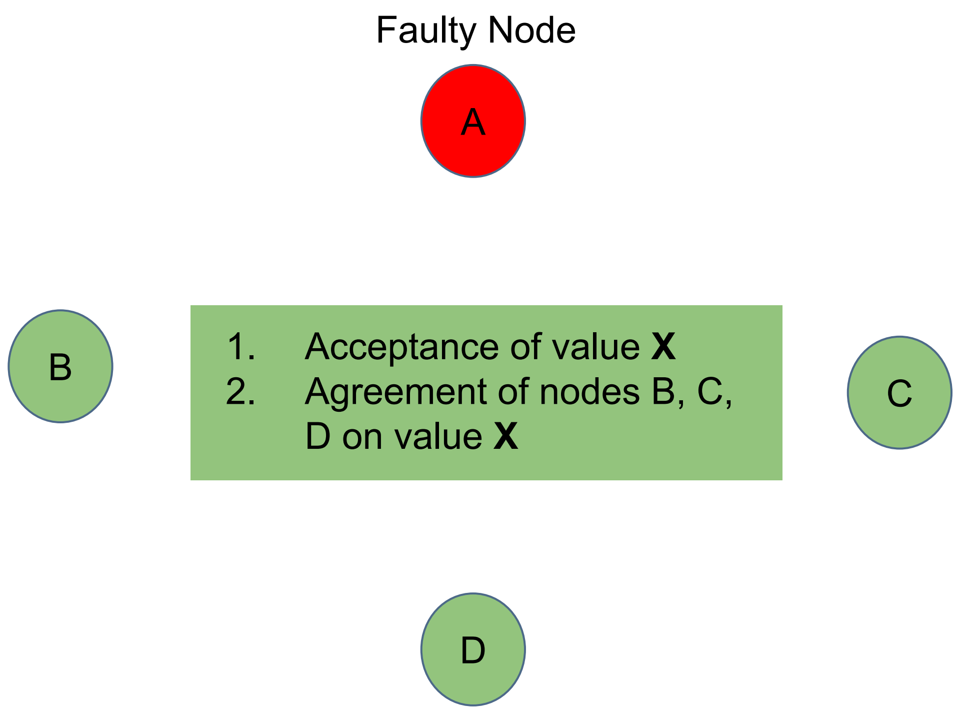
Ready Phase (Validation Begins)



- **Action**: Nodes B, C and D, after receiving enough "echo" + “initial” messages move to the **ready** phase. This is where they validate that the message they received is consistent with others.
- **Message**: Nodes B, C and D send a "ready" message indicating that they are ready to accept the value **X** because they have received enough consistent echoes.
- **Purpose**: The Ready phase finalizes the broadcast by confirming that enough nodes have consistently received the message.

UC DAVIS COLLEGE OF ENGINEERING

Agreement (Accept)



1. Acceptance of value **X**
2. Agreement of nodes B, C, D on value **X**

- **Action**: Nodes B, C and D, after receiving enough "ready" messages (again 2f + 1 messages), will accept the value as **X**.
- **Message**: Nodes B, C and D confirm the agreement on value **X**.
- **Purpose**: This ensures that the decision is final and cannot be overturned, as it's backed by a majority of correct nodes.

How are Byzantine Faults handled?

- Node A is the faulty node and it send **Y** instead of **X** in the echo phase but as nodes do not receive 2f + 1 messages saying that **Y** is the correct value, the faulty message is discarded.
- Only messages. which are validated by the majority of correct nodes, are accepted. This multi-step validation prevents faulty nodes from disrupting the consensus, as only consistent and validated messages proceed to the final agreement.

The Consensus Protocol

1. Asynchronous Consensus Protocol is designed for asynchronous environments, meaning it does not assume any bounds on message delivery times or the relative speeds of processes.

2. The protocol tolerates up to $t < n/3$ Byzantine faults, where n is the total number of processes.

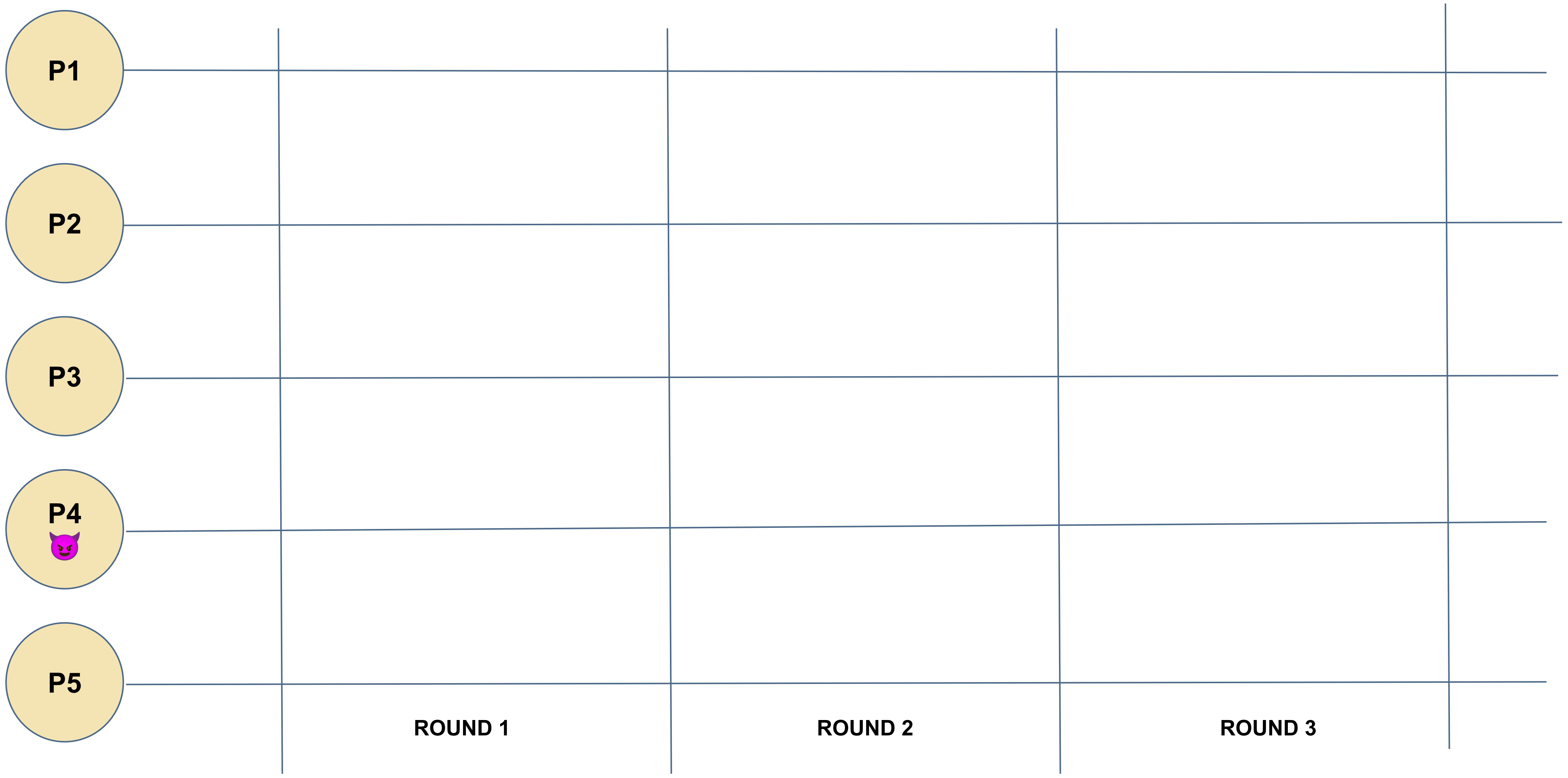
3. The protocol uses randomization (coin toss) to prevent deadlock. If a decision cannot be reached, the protocol employs a coin-toss mechanism to select a random value, ensuring progress.

4. The protocol guarantees that the probability of non-termination approaches zero, ensuring that processes will reach a decision after a finite number of rounds.

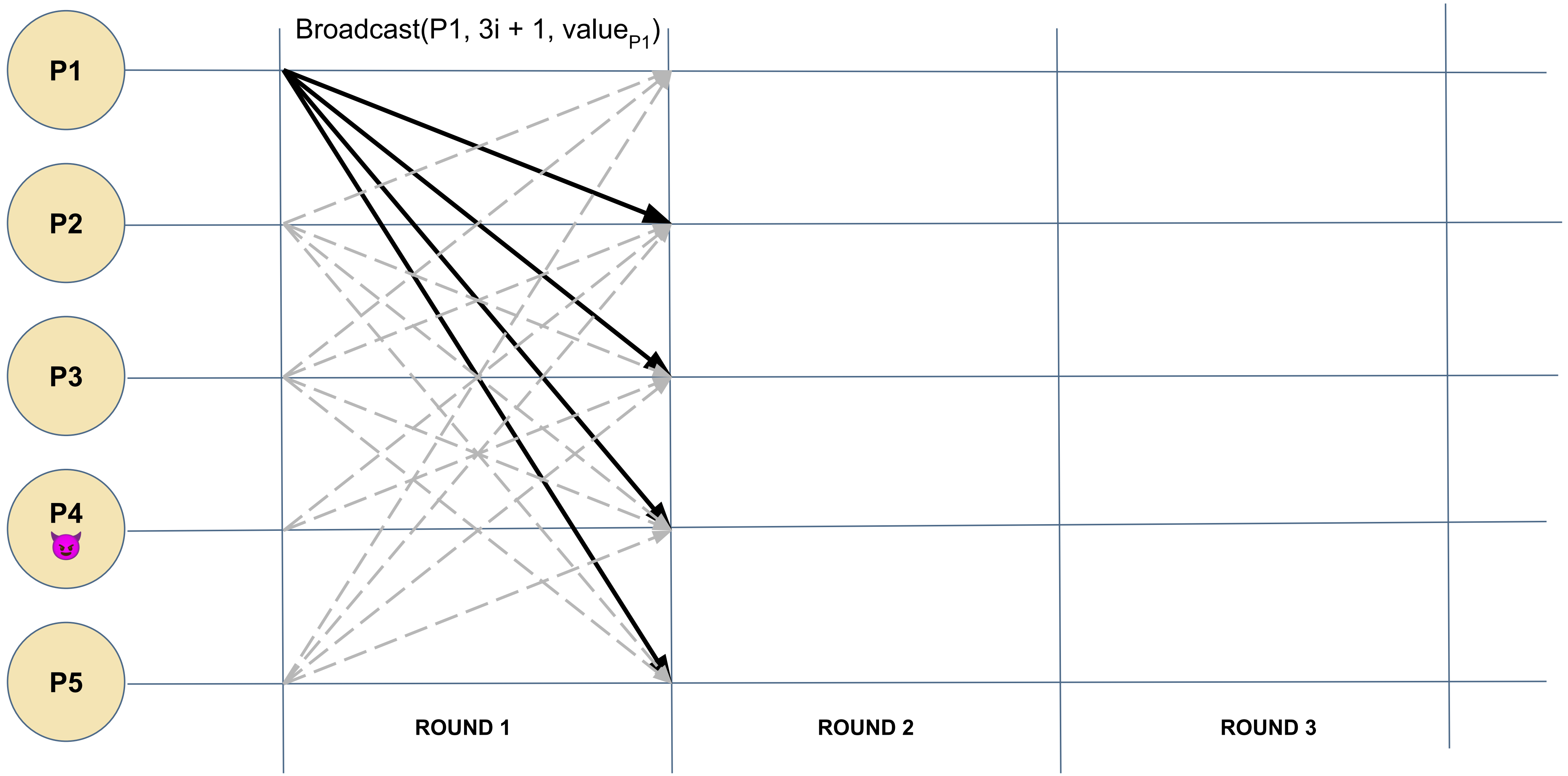
Key Assumptions

- The protocol is conducted in phases and is executed by all processes
- A Process can proceed to phase i+ 1 only after it completed phase i
- Each Phase i consists of rounds 3i + 1, 3i + 2, and 3i + 3, that are instances of the generic Broadcast Primitive.
- The messages contain either a simple value, v, or a tagged value, (d, v), indicating that the process is ready to decide v at that phase.
- For notational convenience, the protocol does not terminate once a decision is made. However, this can be easily accomplished.

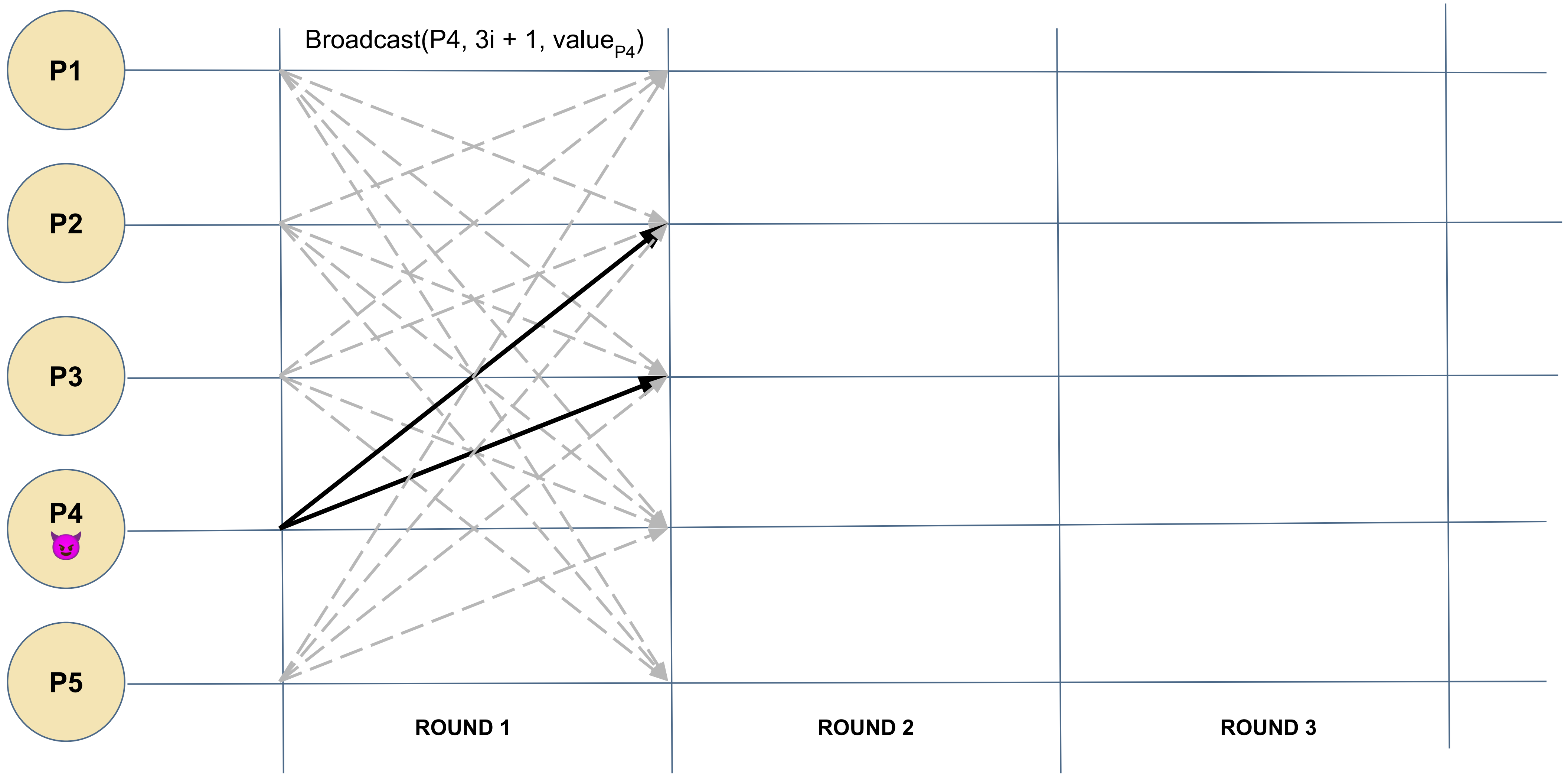
Protocol Introduction



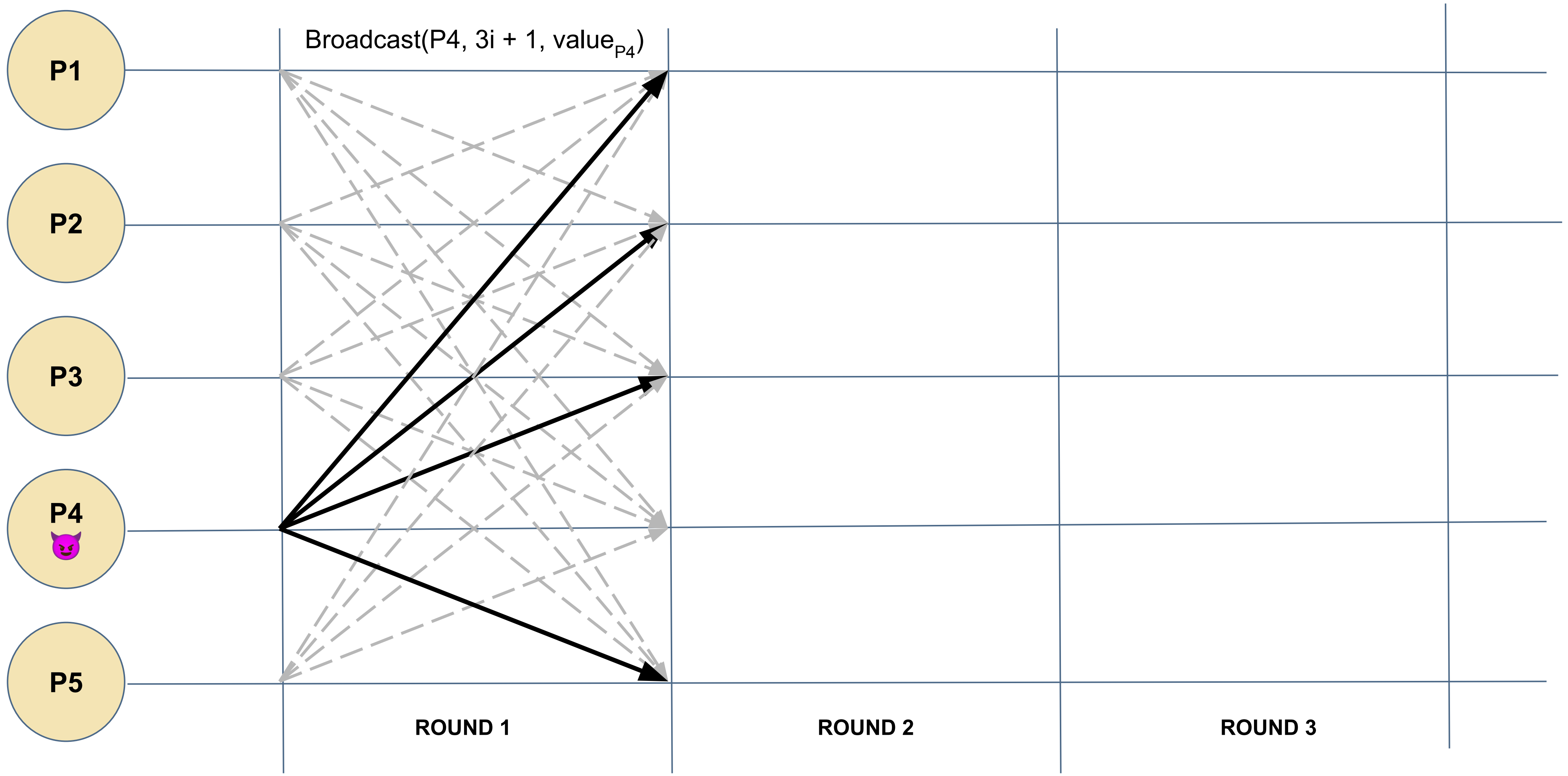
Broadcast in Round 1 by Non Faulty Process



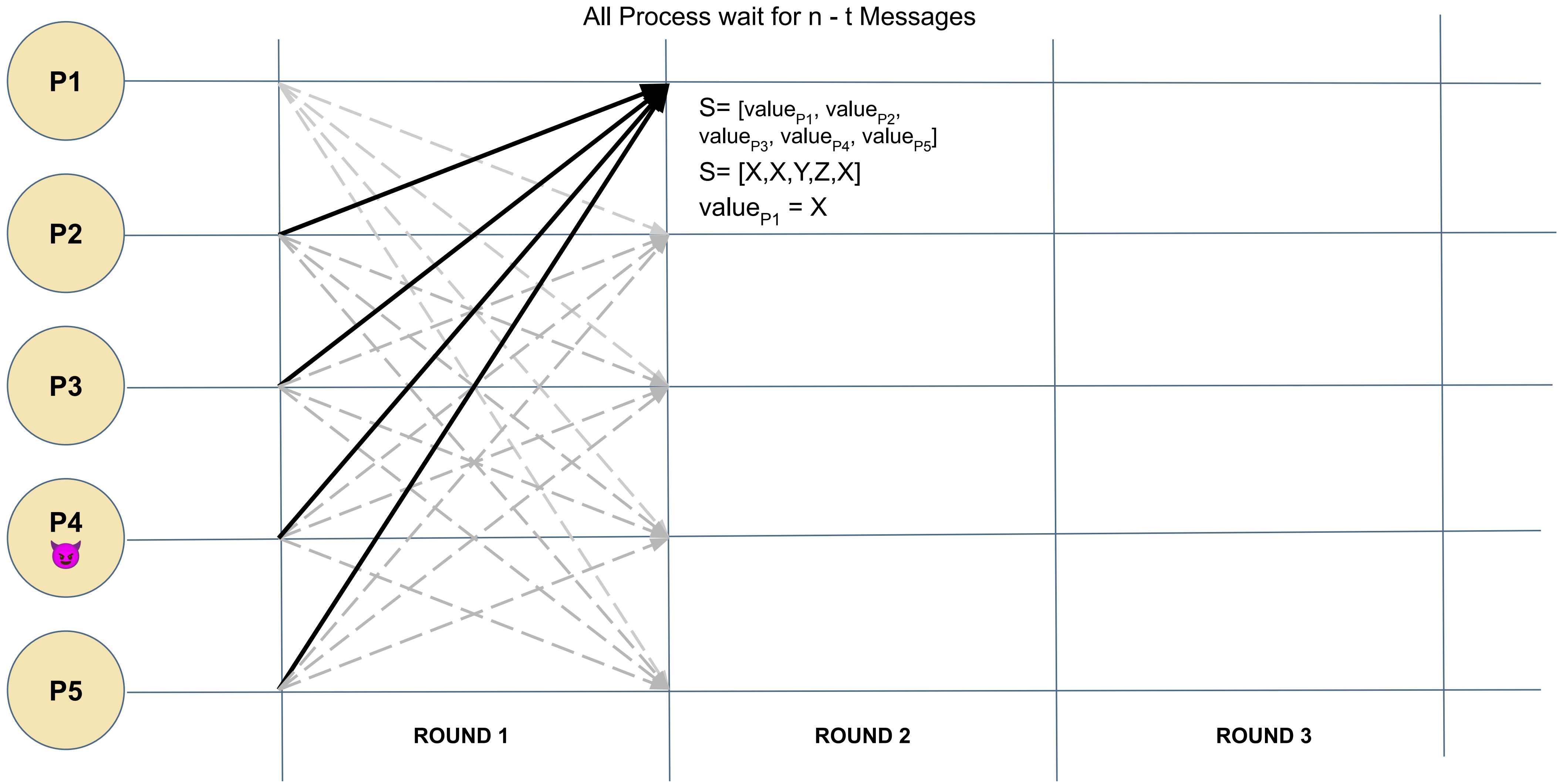
Broadcast in Round 1 by Faulty Process (CASE 1: Broadcast Drop)



Broadcast in Round 1 by Faulty Process (CASE 2: Random Values)



Validating and Updating Messages in Round 1 (CASE 1 : Majority Reached)

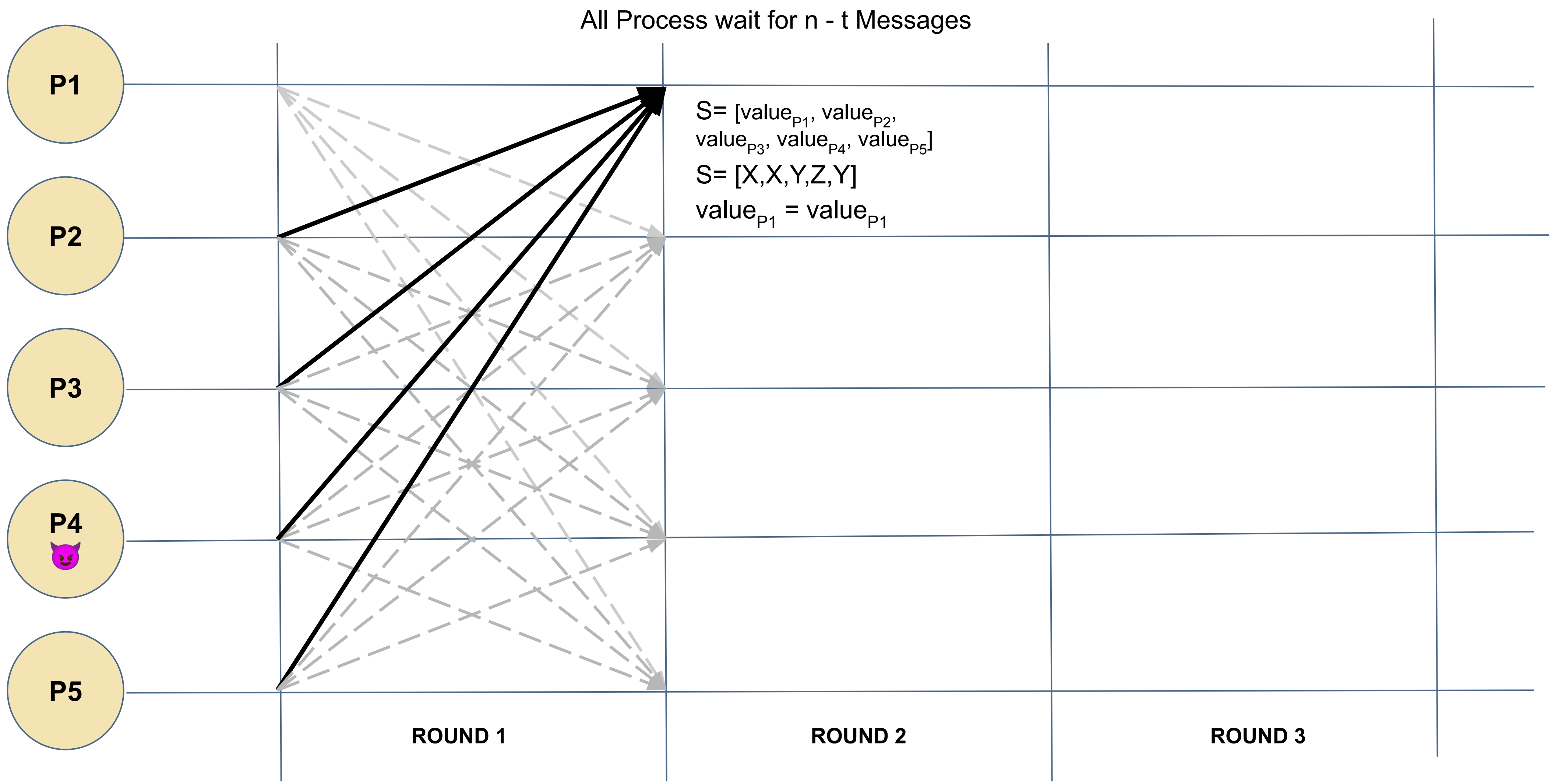


S= [value$_{P1}$, value$_{P2}$, value$_{P3}$, value$_{P4}$, value$_{P5}$]

S= [X,X,Y,Z,X]

value$_{P1}$ = X

Validating and Updating Messages in Round 1 (CASE 2 : Majority not Reached)

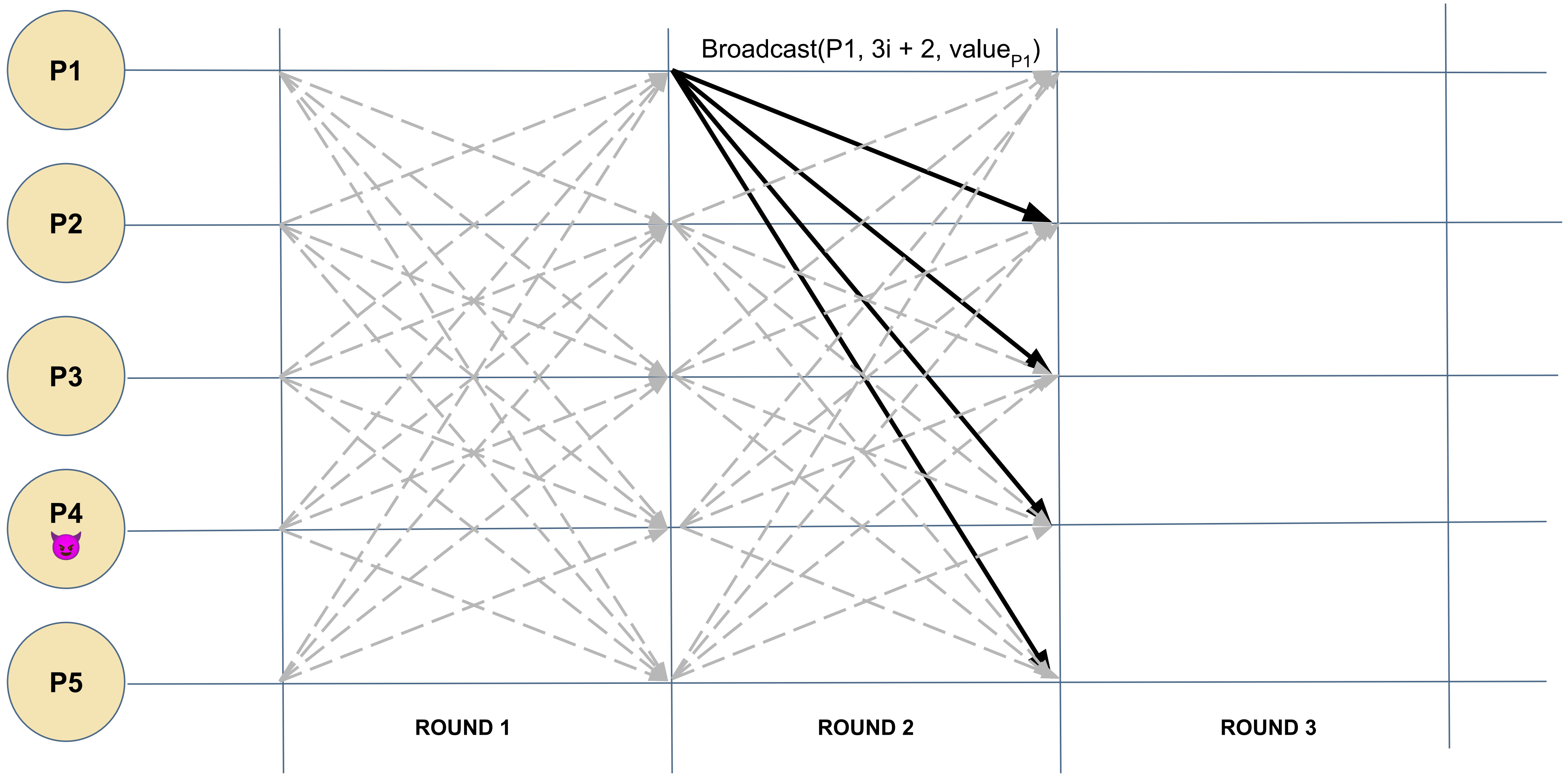


S= [value$_{P1}$, value$_{P2}$, value$_{P3}$, value$_{P4}$, value$_{P5}$]

S= [X,X,Y,Z,Y]

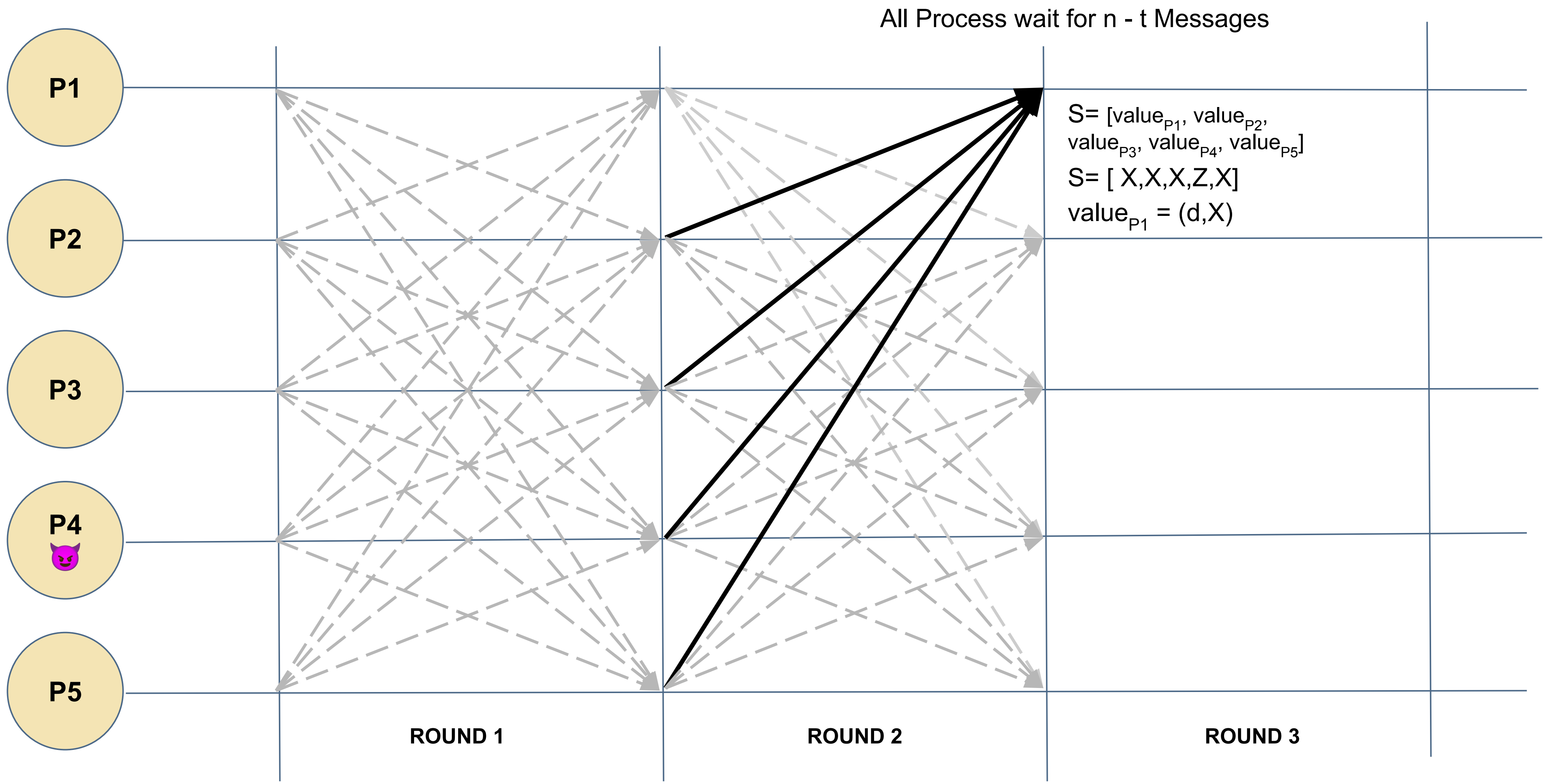
value$_{P1}$ = value$_{P1}$

Broadcast in Round 2



UC DAVIS COLLEGE OF ENGINEERING

Validating and Updating Messages in Round 2 (CASE 1 : More than N/2 validated messages have same value V)

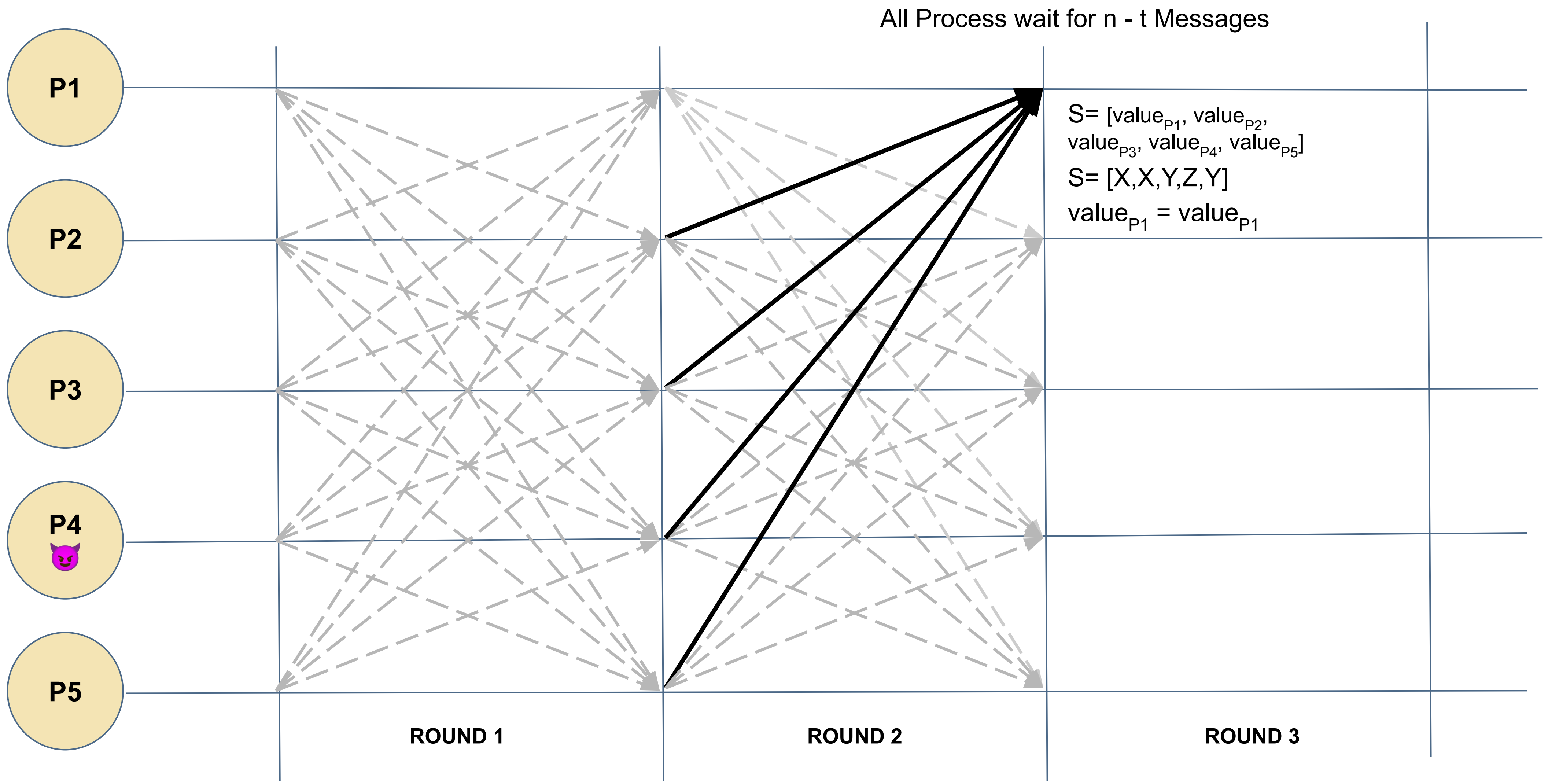


S= [value$_{P1}$, value$_{P2}$, value$_{P3}$, value$_{P4}$, value$_{P5}$]

S= [X,X,X,Z,X]

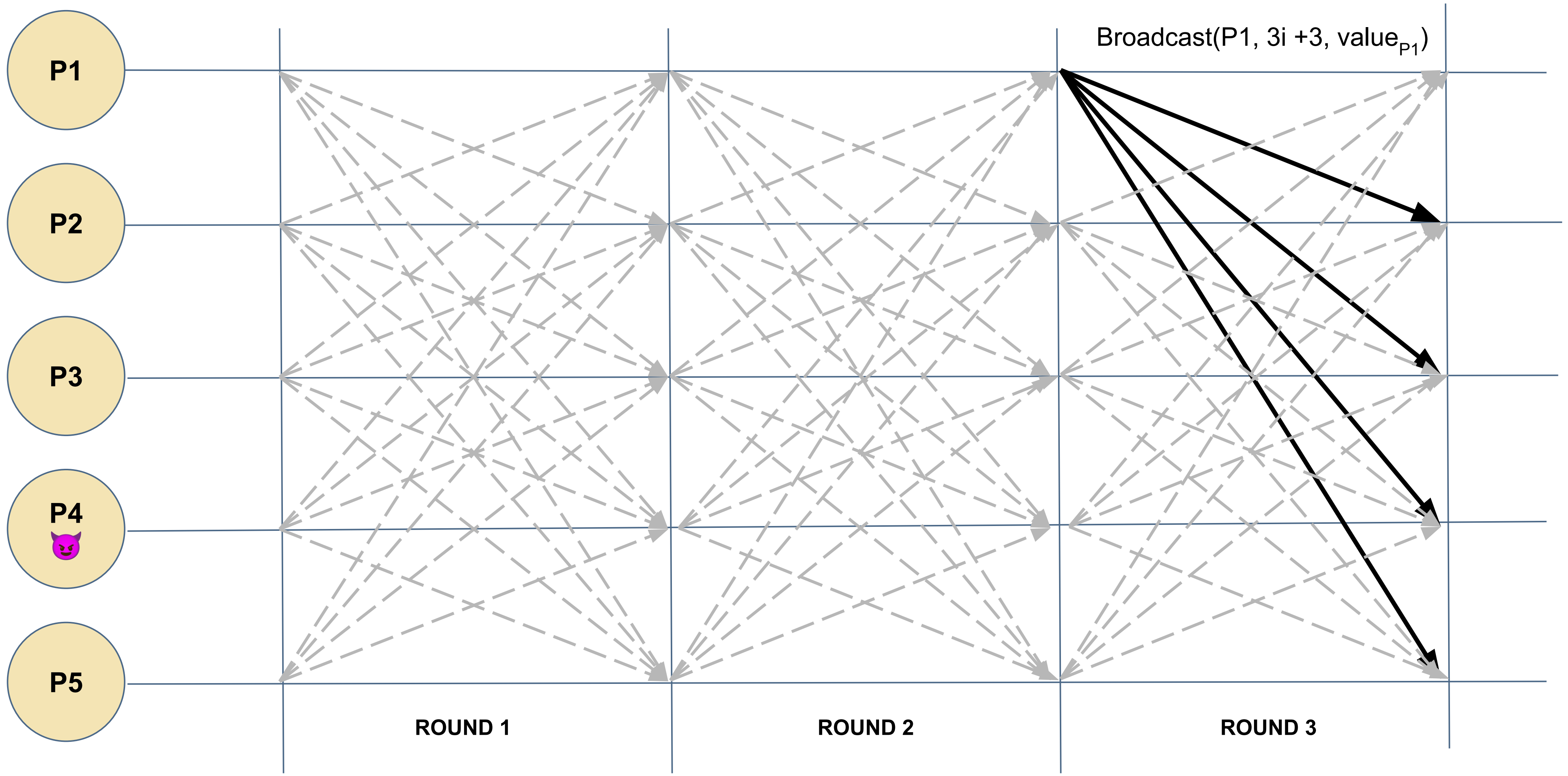
value$_{P1}$ = (d,X)

Validating and Updating Messages in Round 2 (CASE 2 : Less than N/2 Validated message have value V)

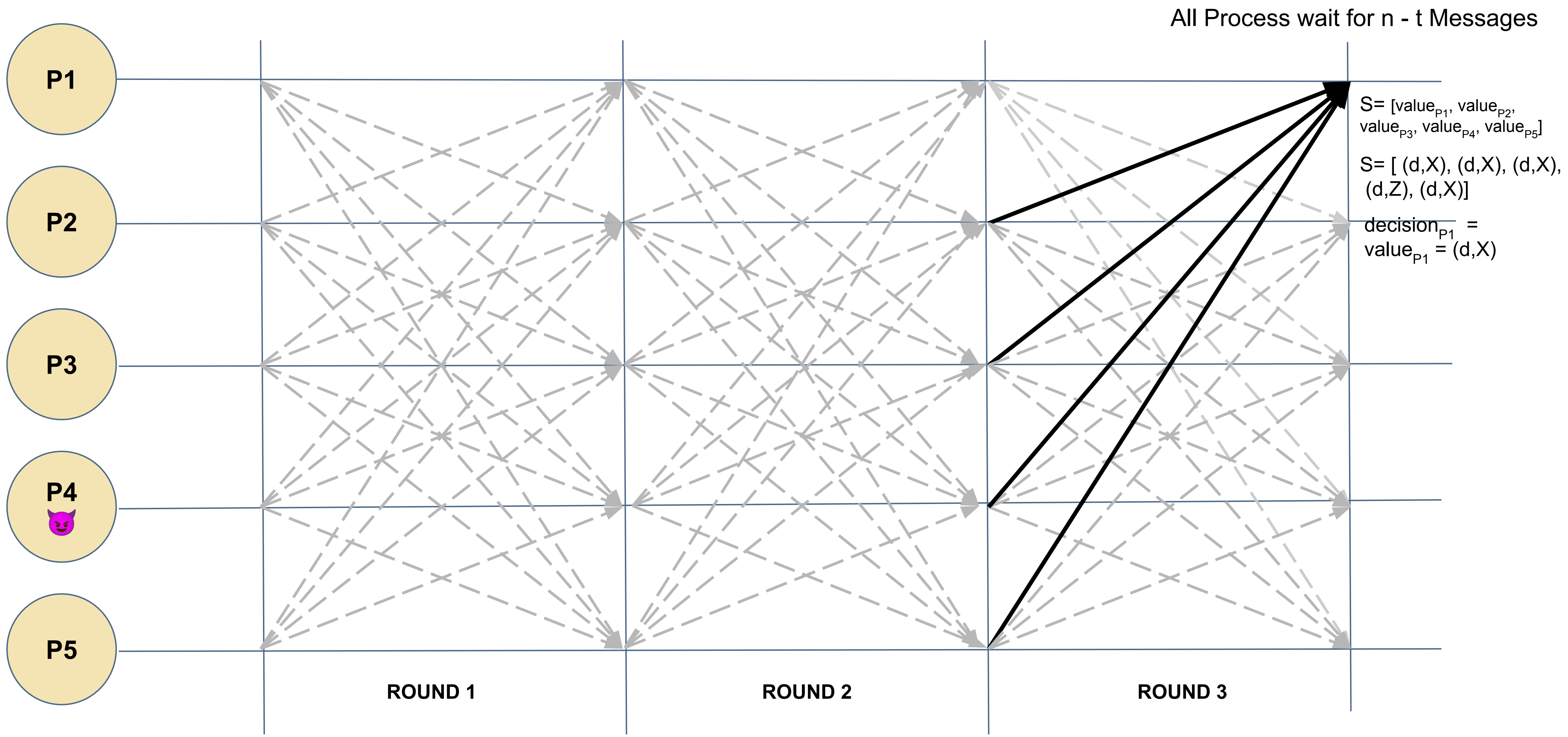

All Process wait for n - t Messages
P1
P2
P3
P4 😈
P5
S= [value$_{P1}$, value$_{P2}$, value$_{P3}$, value$_{P4}$, value$_{P5}$]
S= [X,X,Y,Z,Y]
value$_{P1}$ = value$_{P1}$
ROUND 1
ROUND 2
ROUND 3

UCDAVIS COLLEGE OF ENGINEERING

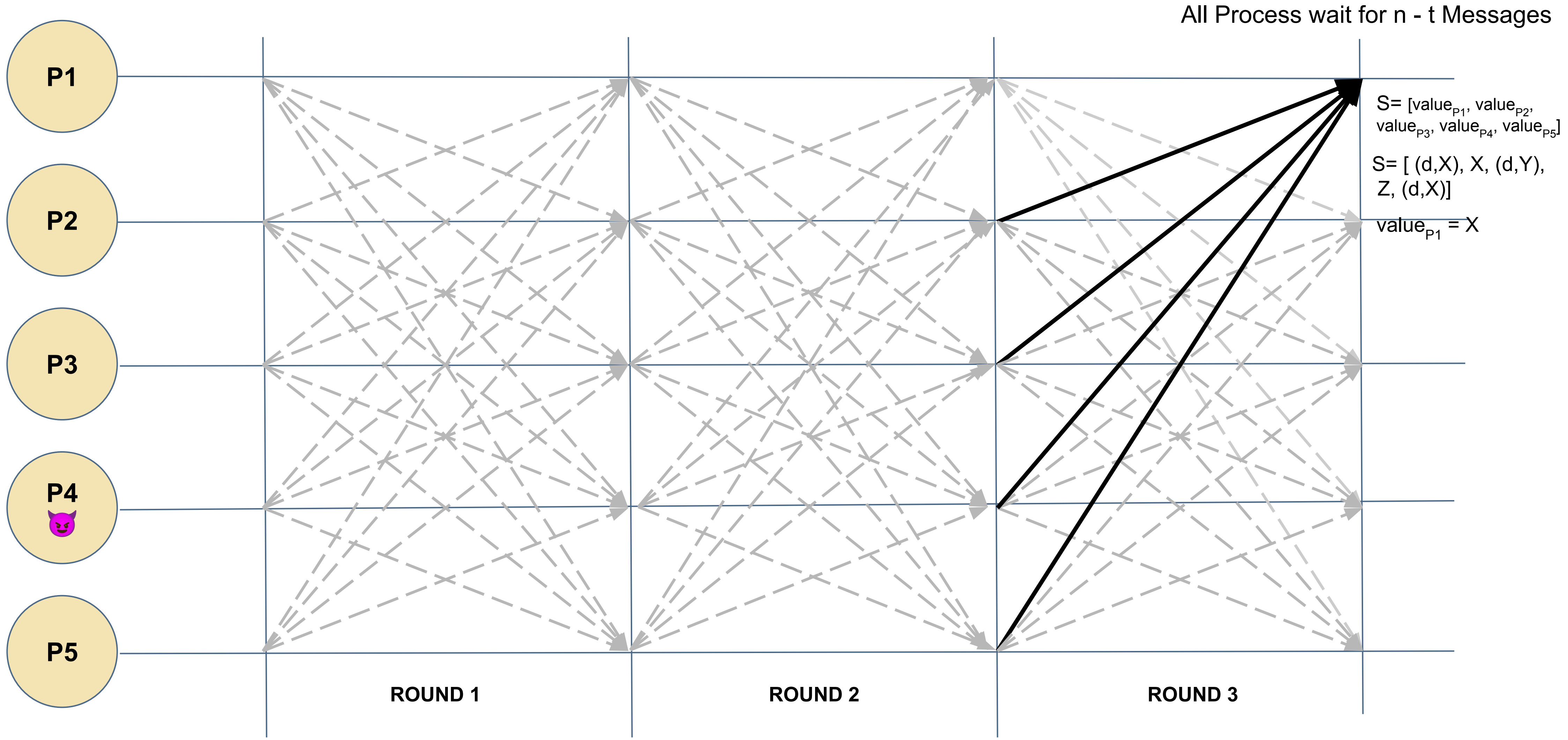
Broadcast in Round 3



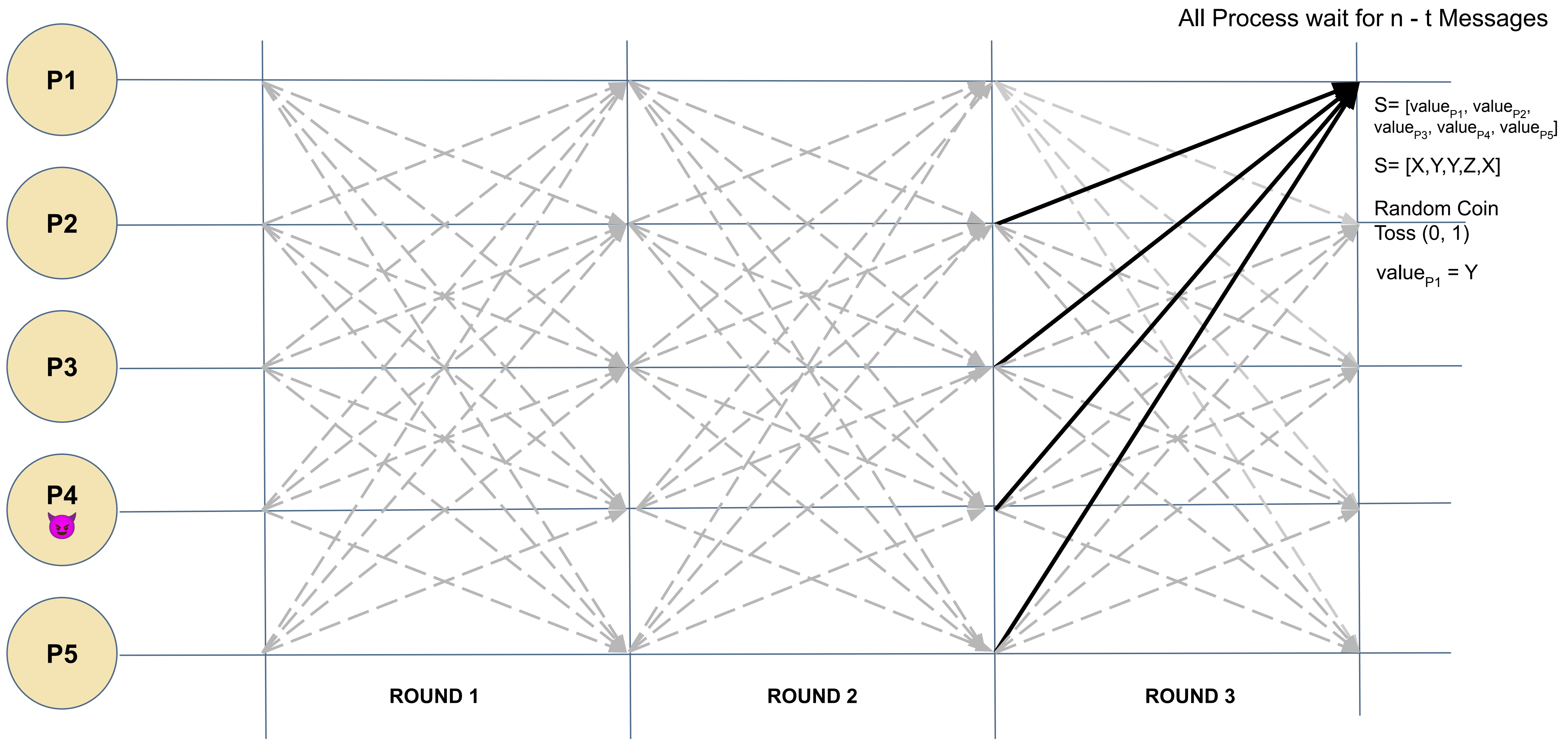
Validating and Updating Messages in Round 3 (CASE 1 : Full Decision (More than 2t messages))


All Process wait for n - t Messages
P1
P2
P3
P4
P5
S= [value$_{P1}$, value$_{P2}$, value$_{P3}$, value$_{P4}$, value$_{P5}$]
S= [(d,X), (d,X), (d,X), (d,Z), (d,X)]
decision$_{P1}$ = value$_{P1}$ = (d,X)
ROUND 1
ROUND 2
ROUND 3

Validating and Updating Messages in Round 3 (CASE 2 :Partial Support (More than t messages))


All Process wait for n - t Messages
P1
P2
P3
P4 😈
P5
S= [value$_{P1}$, value$_{P2}$, value$_{P3}$, value$_{P4}$, value$_{P5}$]
S= [(d,X), X, (d,Y), Z, (d,X)]
value$_{P1}$ = X
ROUND 1
ROUND 2
ROUND 3

Validating and Updating Messages in Round 3 (CASE 3 : Randomization (Less than t messages))



S= [value$_{P1}$, value$_{P2}$, value$_{P3}$, value$_{P4}$, value$_{P5}$]

S= [X,Y,Y,Z,X]

Random Coin Toss (0, 1)

value$_{P1}$ = Y

If Consensus is not Reached
Move to next Phase
$i + 1$

Termination

- The protocol's goal is for all correct processes to agree on a single value by the end of Round 3r + 3 of a phase, which is the final round in each phase.

- The two main cases considered are:

  - A correct process has validated enough support to choose a value.

  - No correct process has validated enough messages to pick a specific value, so they all proceed with a coin toss.

CASE 1: A correct process has validated enough support to choose a value.

- Suppose there's a process p that has validated enough messages to decide on a specific value v.

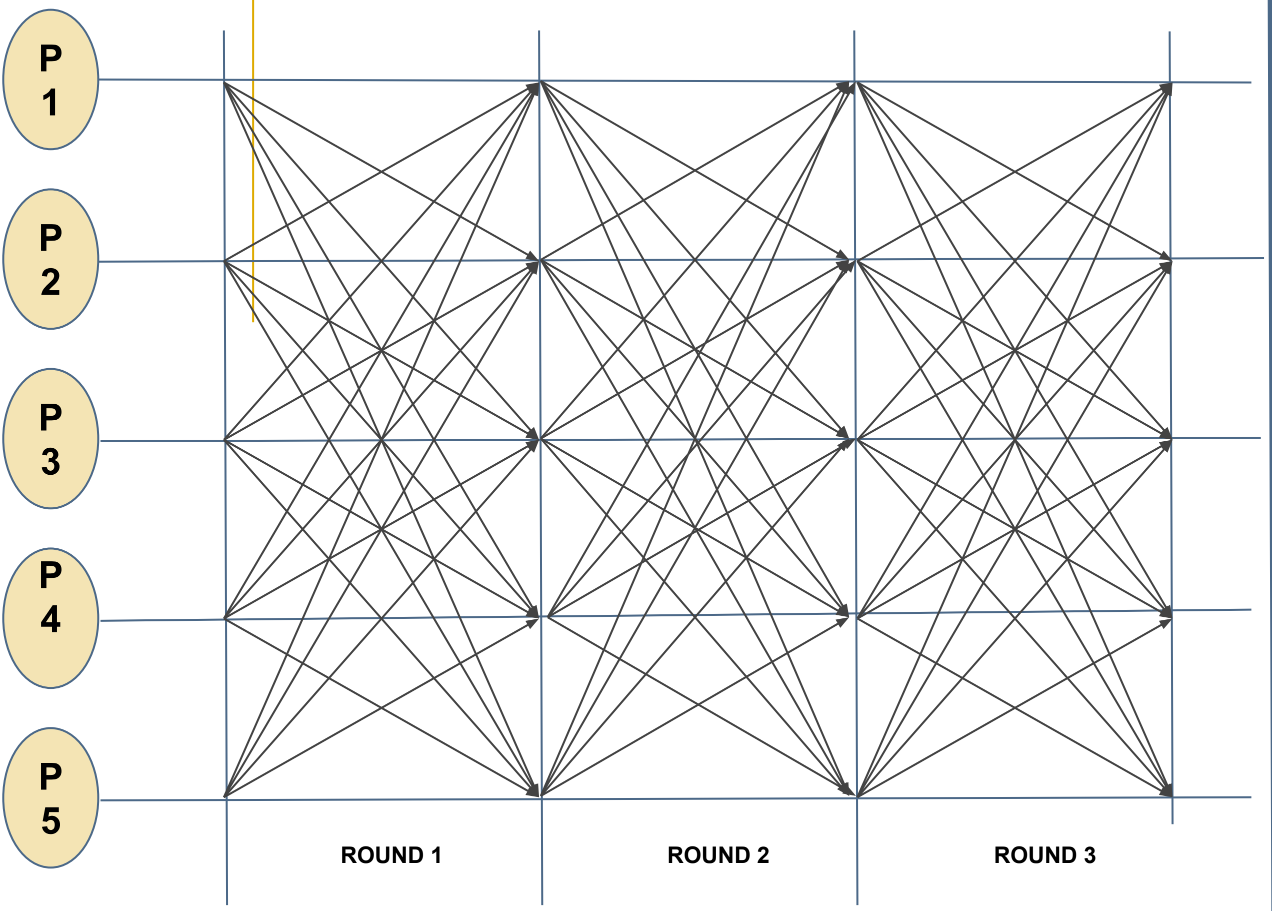
- Other Processes Coin Toss for the Same Value: With a high probability, all other correct processes that need to choose their value through a coin toss in this round will also randomly pick v. This probability is high enough that it's very likely they'll align on the same value.

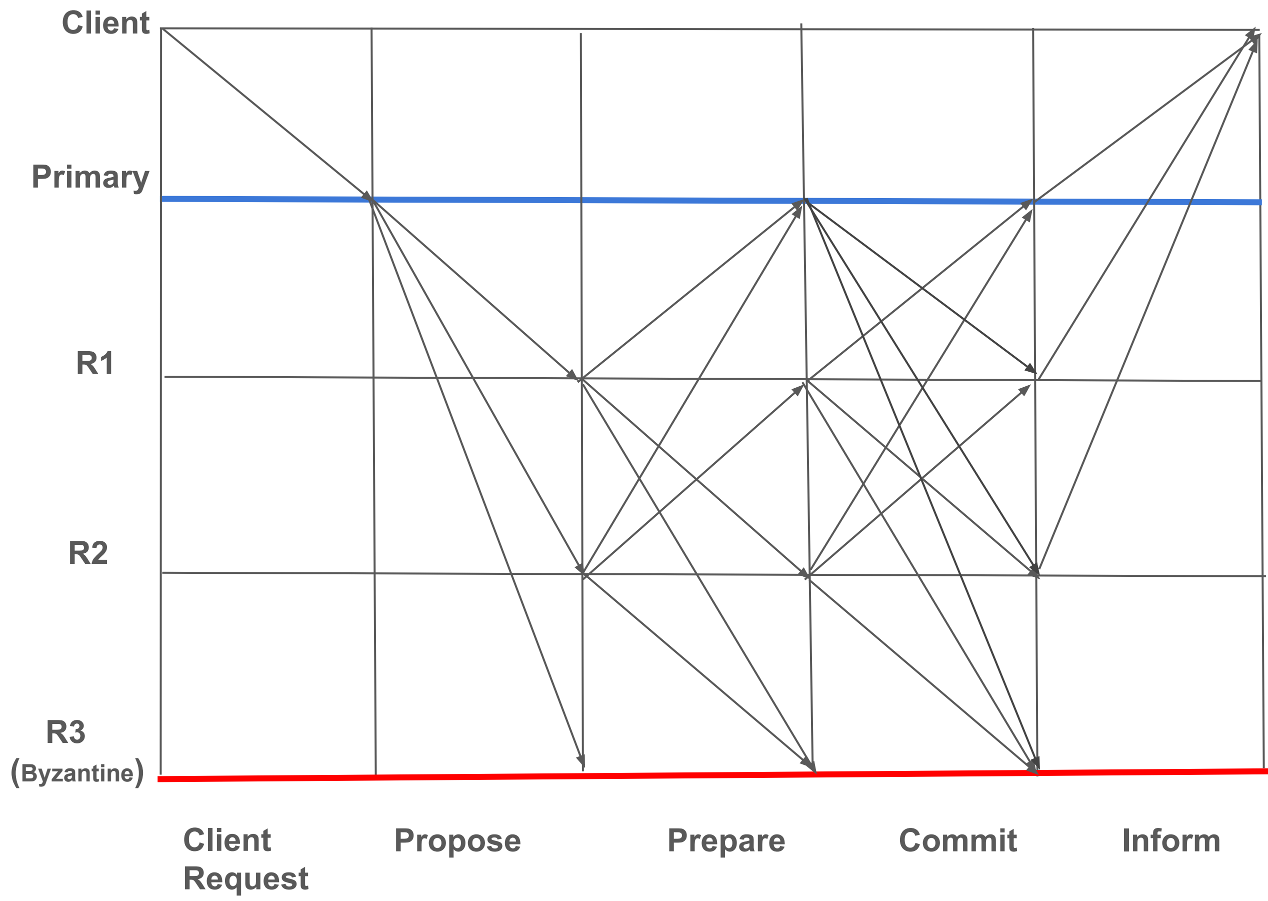
- Outcome: By this round's end, all correct processes will have the same value v, either by validating it or by matching the coin toss result.

CASE 2: No correct process has validated enough messages to pick a specific value, so they all proceed with a coin toss.

- Suppose process p couldn’t validate enough messages to reach a decision, meaning no single value reached a sufficient level of support.
- No Other Process Can Validate a Different Value: If no correct process can validate enough messages for any value, then no process will be forced to pick a specific value.
- All Processes Coin Toss: All correct processes will then choose their values through a coin toss.
- High Probability of Agreement on Coin Toss Result: With a high probability, all correct processes will toss the same result (either 0 or 1), ensuring alignment on a single value.

Comparison of Asynchronous Consensus Protocol with PBFT





Asynchronous Byzantine General Protocol

Synchronous systems: The processes run a consensus protocol after all of them have received values from the transmitter.

Asynchronous systems: The processes do not know whether the transmitter sent no value or whether the sent value has not arrived yet.

- The termination properties for Byzantine Generals protocols are not achievable in an asynchronous system.
- This is because a process does not know at a finite time whether the transmitter sent them a message at all.
- The next slides will demonstrate some examples modifying these termination properties for use in asynchronous protocols.

Let’s say Byzantine General protocols are possible in an asynchronous system.

Case 1: Transmitter is faulty and sends no messages while the other processes are correct. At some finite time T the processes will agree on some arbitrary value i.e. 0, based on the termination property.

Case 2: Same as case 1 but the transmitter is also correct and sends ‘1’ values but they are delayed and don’t arrive at time T. The processes behave the same as case 1 and decide on 0 after waiting until T.

To avoid this scenario, we modify the termination agreement to allow for the case when the broadcast does not terminate.

Termination Agreement (Weak termination):

- According the the modified "weak" termination, if the transmitter is correct then all correct processes will eventually decide (i.e. we allow for the case when the broadcast does not terminate).

- If the transmitter is faulty, either all correct processes eventually decide, or none of them ever decides (the broadcast does not terminate).

The next slide elaborates on this agreement and shows why asynchronous Byzantine Generals protocols with weak termination are only possible when the number of faulty nodes < n/3.

Asynchronous Byzantine General problem with t >= n/3

Let's say the nodes are divided into 3 disjoint categories A, B and C, each of them containing n/3 nodes. Let the transmitter be in A.

Case 1:

Only the transmitter is faulty. It sends '0' messages to processes in A and B and delays messages to be sent to C. A and B now decide on '0' as the n/3 nodes of C are allowed to be faulty.

Case 2:

Only the transmitter is faulty. It sends '1' messages to processes in A and B and delays messages to be sent to C. A and B now decide on '1' in this case.

Next, we will combine these cases to get our contradiction.

Case 3 (combines cases 1 and 2):

- The processes in A are faulty. They send ‘0’ messages to the nodes in B as case 1 and ‘1’ messages to the nodes in C as case 2.
- Let us consider the case where messages between B and C are delayed.
- As in case 1, the nodes in B will decide on ‘0’ ignoring C’s n/3 nodes and the nodes in C will decide on ‘1’ ignoring B’s nodes. This is a contradiction as correct processes are deciding on different values and so the protocol cannot succeed. This will be true whenever t is >= n/3.

We conclude that Byzantine General protocols cannot be possible in any asynchronous system with the number of faulty nodes >= n/3.

Thank You