ASYNCHRONOUS BYZANTINE AGREEMENT PROTOCOLS

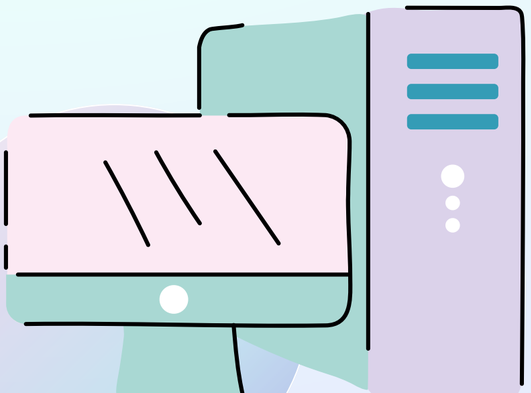
– GABRIEL BRACHA

ECS 265: Distributed Database Systems

GROUP MEMBERS:

1. Aayusha Hadke (924110050)
2. Celine John Philip (924165636)
3. David Chu (916579612)
4. Ishika Bhaumik (924101409)

CONTENTS



INTRODUCTION

Problem: Achieving Byzantine Agreement in asynchronous distributed systems with unreliable communication.

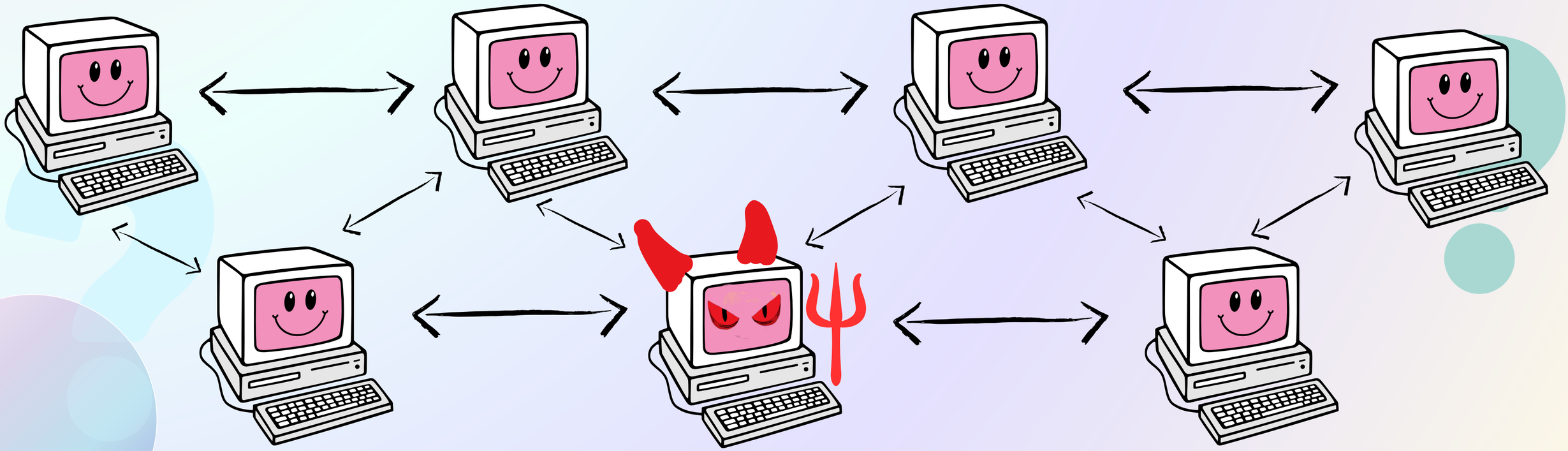
Byzantine faults: Faulty or malicious processes that can lie, omit, or send inconsistent messages.

Challenge: No timing guarantees, deterministic consensus impossible under full asynchrony.

Bracha's solution: A randomized protocol ensuring agreement, validity, and termination when less than n/3 processes are faulty.

What is the problem statement?

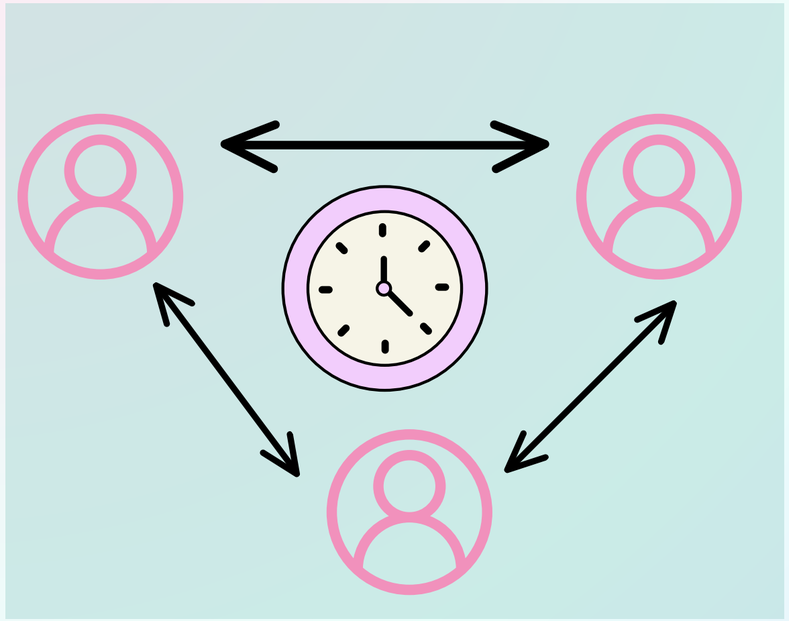
Byzantine Agreement is the problem of reaching reliable consensus among distributed processes, even when some behave maliciously or arbitrarily, by ensuring all honest processes agree on the same value despite the presence of faulty nodes.

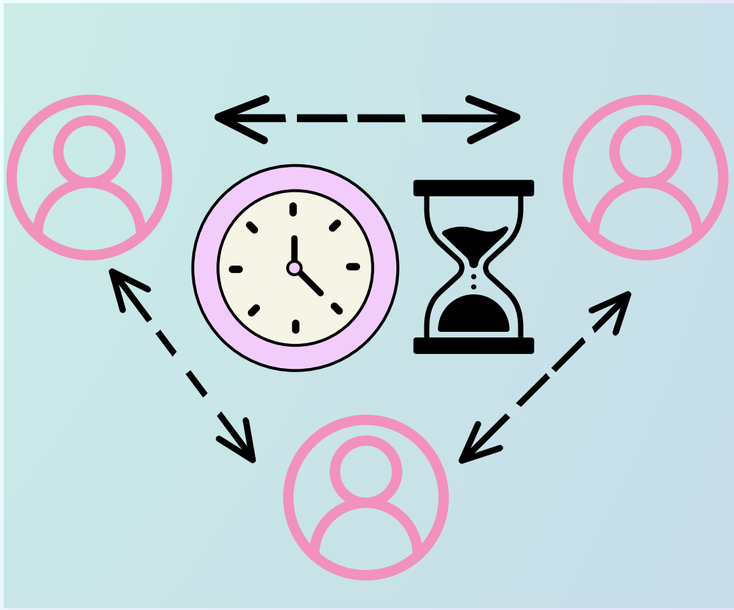


Byzantine Faults and System Models

- Byzantine vs. Fail-Stop Faults: Byzantine nodes behave arbitrarily or maliciously, while fail-stop nodes simply stop responding.
- System Types: Asynchronous: Unbounded, unpredictable delays.
- Randomized Protocols: Randomized = uses probability to ensure eventual consensus
- System Models-
  - The system consists of n processes that communicate by sending messages
  - Communication happens through a reliable message system:
  - No messages are lost, duplicated, or fabricated.
  - Each process can identify the sender of every message it receives.
  - Each process can directly send messages to any other process (fully connected network).
  - Up to t processes may be faulty and can behave arbitrarily (Byzantine faults) where $0 < t < n/3$.

System Model Types





Synchronous: Timed and predictable, everyone delivers on time!

Partially Synchronous: Unpredictable at first, but after some time, everything becomes reliable!

Asynchronous: No guarantees, messages could take forever, leaving everyone guessing!

RELIABLE BROADCAST PROTOCOLS

- Purpose: Ensures all honest nodes receive the same message from a sender, despite faulty or malicious nodes.

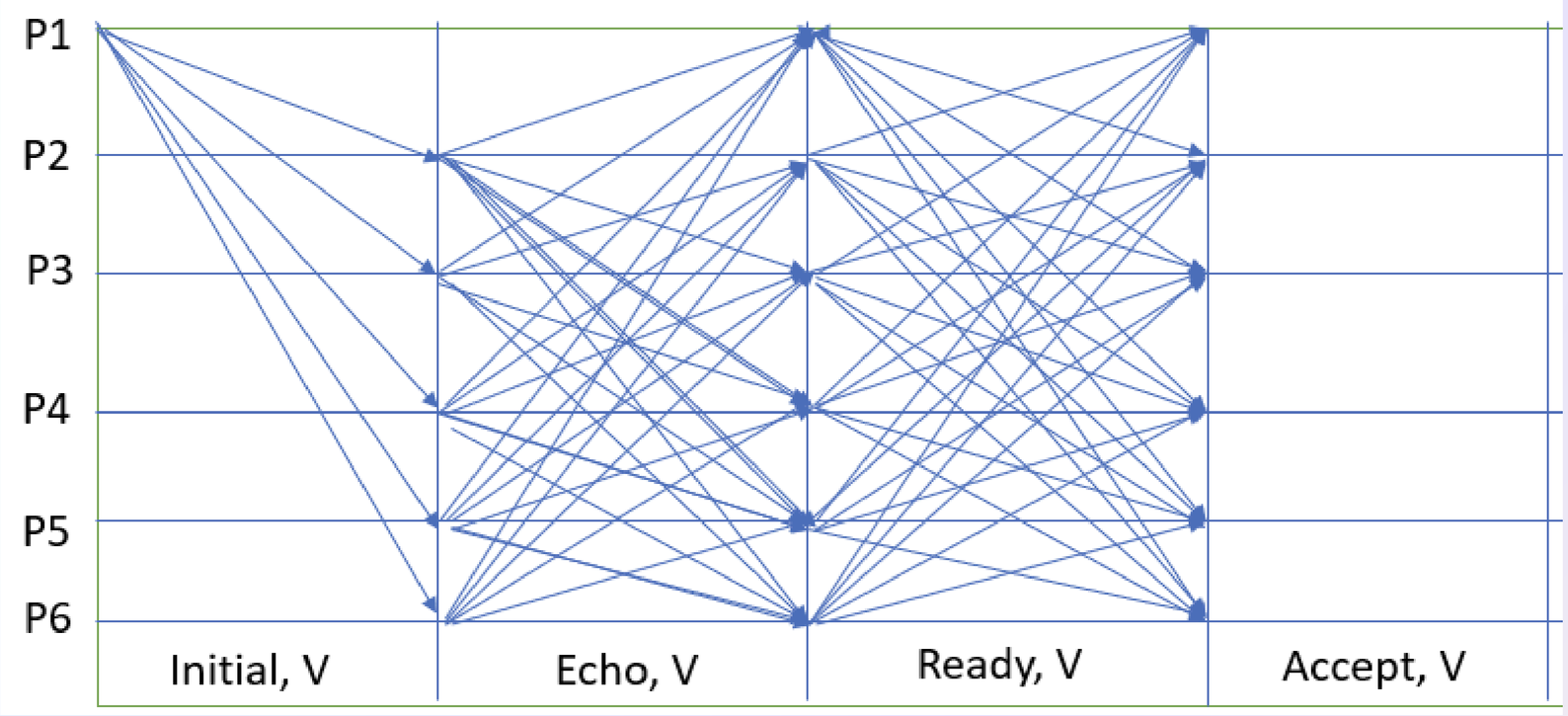
- Key Properties:
  - Validity – honest sender → all deliver message
  - Agreement – if one honest node delivers, all do
  - Integrity – message delivered once only
  - Totality – delivery guaranteed across honest nodes

- Importance: Provides a consistent communication layer, essential for consensus and coordination.

RELIABLE BROADCAST PROTOCOLS CONT.

How It Works

- Broadcast: The sender p transmits an (initial, v) message to all nodes.
- Echo Phase: Each node rebroadcasts an (echo, v) declaring “I heard that p wants to broadcast v!”
- Ready Phase: Nodes send (ready, v) messages. “I’m ready to accept v as the final value.”
- Acceptance: Once a node receives 2t + 1 “ready” messages for the same value, it accepts that value.



RELIABLE BROADCAST vs PBFT

Aspect	Bracha's Reliable Broadcast	PBFT – Practical Byzantine Fault Tolerance
Purpose	Guarantee all honest nodes deliver the **same message** from a sender	Achieve **full Byzantine consensus** and order client operations
Type of Protocol	Communication primitive (lower layer)	Complete consensus protocol (upper layer)
What It Solves	Consistent message delivery despite Byzantine behavior	Agreement + ordering + execution of client requests
Phases	Initial → Echo → Ready → Accept	Propose→ Prepare → Commit → Inform
Leader Needed?	No	Yes
Fault Tolerance	Requires **n ≥ 3t+ 1**; ensures all honest nodes deliver same value	Requires **n ≥ 3t + 1**; ensures all honest nodes commit same order of ops

For n = 4 and t=1

Protocol Action	Threshold Formula	Calculated Threshold	Meaning
Send READY(v)	(n+t)/2	(4+1)/2= **3 ECHOs**	Node must receive **3 ECHO(v)** messages
ACCEPT(v)	2t+1	2(1)+1=**3 READYs**	Node must receive **3 READY(v)** messages

RELIABLE BROADCAST PROTOCOLS

Lemma 1 → No Conflicting Ready Messages

Two correct processes cannot send different "ready" values because each must receive more than (n + t) / 2 matching echo messages.

Conclusion: If two correct processes send ready messages, the value must be the same.

Lemma 2 → No Conflicting Accepts

Two correct processes cannot accept different values because acceptance requires at least 2t + 1 matching ready messages.

Conclusion: If two correct processes accept a value, it must be the same.

Lemma 3 → Acceptance Propagates

If one correct process accepts a value, at least t + 1 correct ready messages ensure all other correct processes will eventually accept it.

Conclusion: If one correct process accepts a value, all correct processes will accept that value.

Lemma 4 → Correct Sender Ensures Acceptance

If the sender is correct, all correct processes eventually receive consistent initial, echo, and ready messages.

Conclusion: A correct sender causes all correct processes to accept the broadcast value.

For n=4 and t =1

Lemma 1 → No Conflicting Ready Messages

- Assume P1 sends ready,V which means its has received 3 echo,V message
- P2 sends ready,U which means it has received 3 echo,U message
- if that's the case 3+3 makes it 6 nodes which isn't true we only have 4 nodes so atleast 1 honest node here is sending the message U as V, So conflicting ready messages.

Lemma 2 → No Conflicting Accepts

- Assume P1 Accepts U which means it has received 3 ready,U message
- Assume P2 accepts V which means it has received 3 ready,V messages
- Which means either there are 6 processes or 4 honest process and both the cases are not possible. Hence no conflicting accepts

Lemma 3 → Acceptance Propagates

If one correct process accepts a value, at least t + 1 correct ready messages are being send which mean further these 2 correct ready messages will propagate hence ensuring that all nodes accept the correct message.

Lemma 4 → Correct Sender Ensures Acceptance

By Lemma 1 2 and 3 we can say that every honest process as a sender will send consistent echo and ready messages across the system

Correctness Proof

Goal: Show that Protocol 1 satisfies the properties of a reliable broadcast for $t < n/3$.

KEY CONCEPTS

Lemma 1 → No two correct nodes send conflicting ready messages.

Lemma 2 → No two correct nodes can accept different values.

Lemma 3 → If one correct node accepts a value, all eventually do.

Lemma 4 → If sender is correct, all correct nodes accept its value.

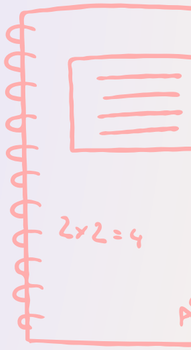
Proof Techniques:

- Use threshold arguments:
- $(n + t)/2$ echo messages and $2t + 1$ ready messages guarantee overlapping honest senders.
- two conflicting sets of size $> (n + t)/2$ must overlap in ≥ 1 correct process.

Conclusion: All correct processes deliver the same message (consistency), and if the sender is correct, everyone accepts it (validity).

Theorem 1: Reliable Broadcast Proven

Statement: For $t < n/3$, Protocol 1 satisfies validity, agreement, integrity, and totality.



PROOF SUMMARY

1. **From Lemma 4 → honest sender ⇒ all deliver.**

2. **From Lemma 3 → if one delivers, all deliver.**

3. **From Lemma 2 → all delivered values identical**

4. **From Lemma 1 → no conflicting ready messages.**

✅ Therefore: Protocol 1 is a reliable broadcast protocol.

Takeaway: A foundation for fault-tolerant communication, guarantees uniform message delivery among all honest nodes.

Correctness Enforcement (Validation Mechanism)

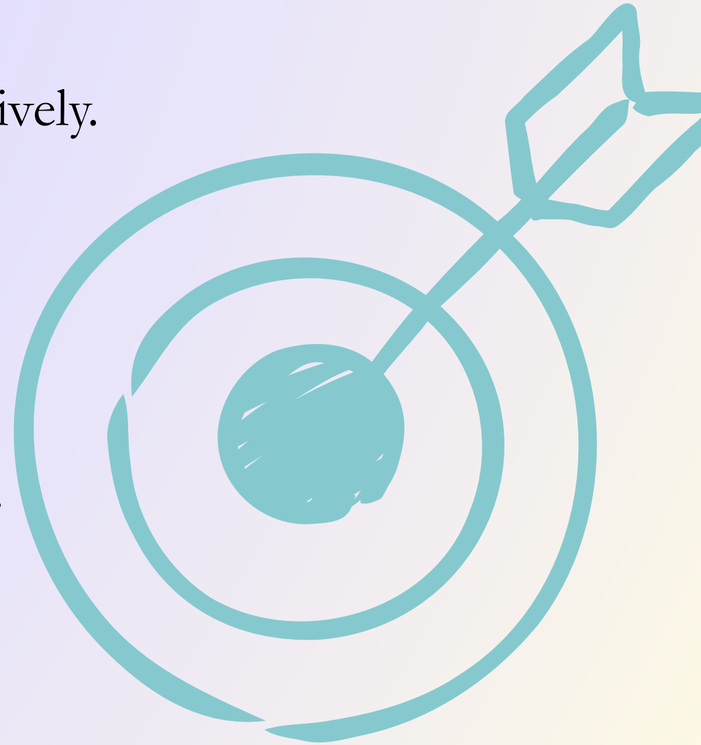
Goal: Force Byzantine processes to behave according to the protocol logic.

Approach: Replace simple send/receive with Broadcast and Validate primitives.

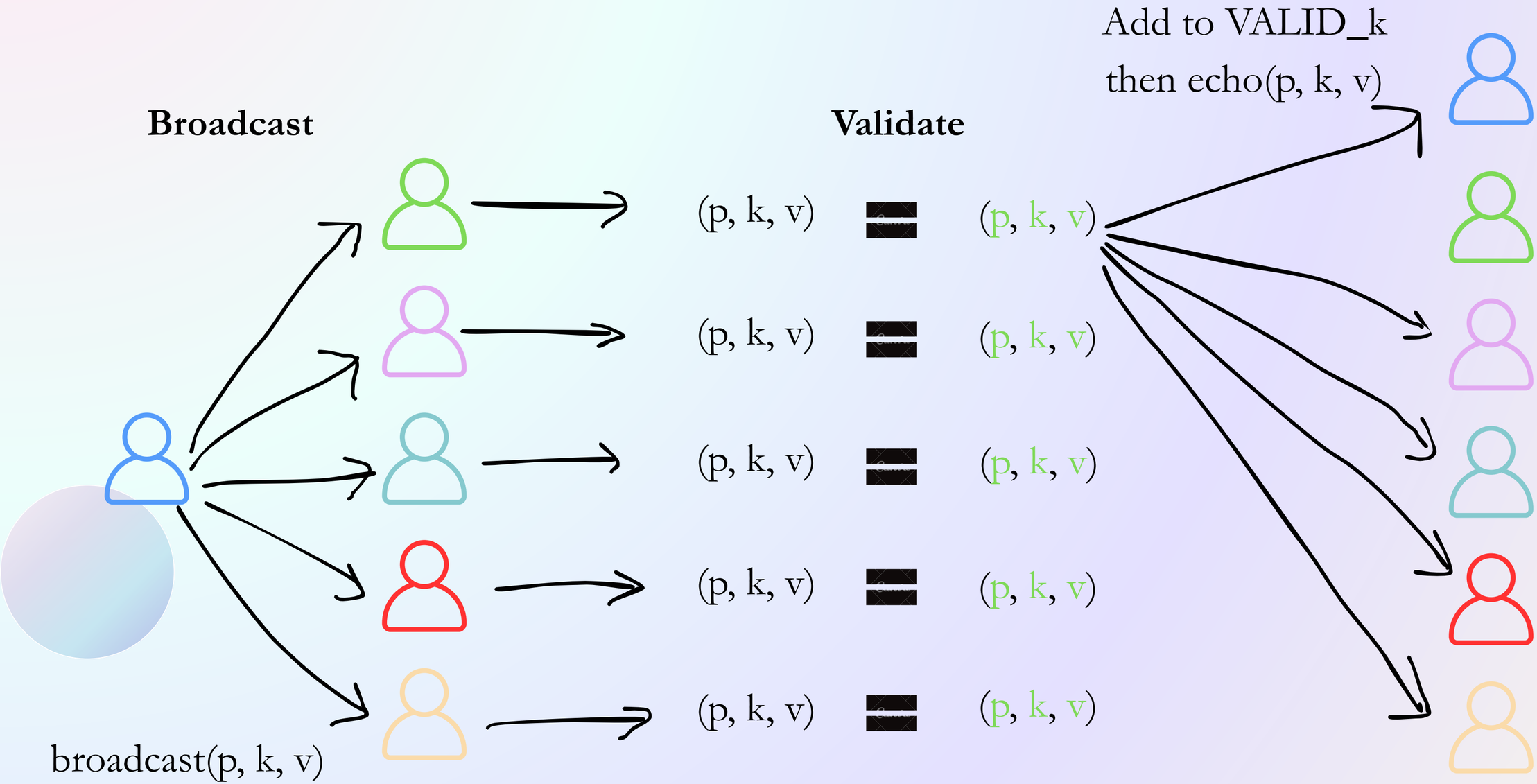
Key Concepts:

- Each k-message tagged as (p, k, v) → identifies sender, round and value respectively.
- Each process keeps VALID_k = set of all accepted, rule-following k-messages.
- Specifically in this paper, v ∈ {0,1}
- A k-message is validated only if it could be produced by a correct process.
- Unvalidated messages ignored (but stored).

Purpose: Restrict Byzantine nodes to only send valid, protocol-consistent messages.

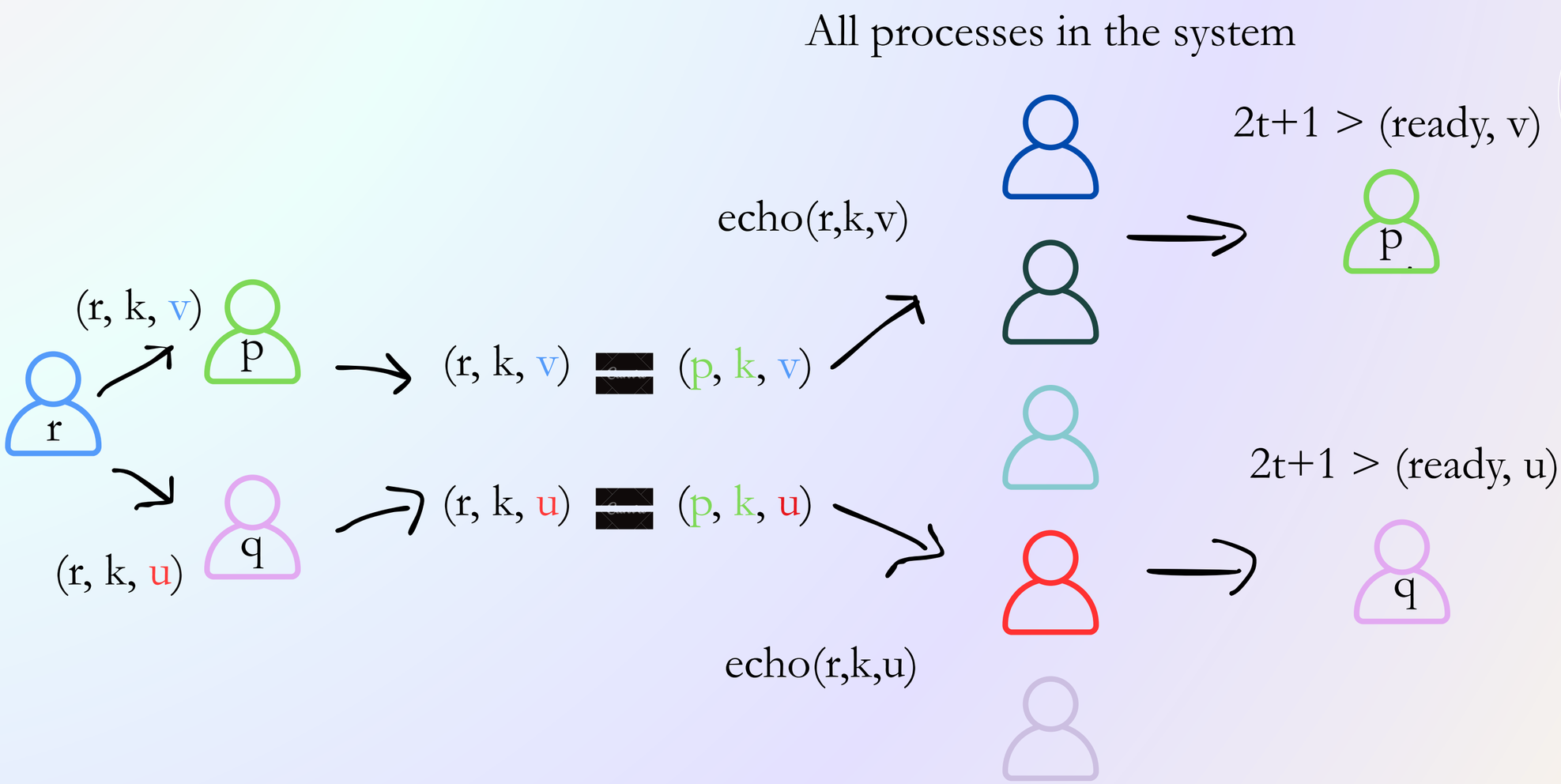


Broadcast and Validation Primitive



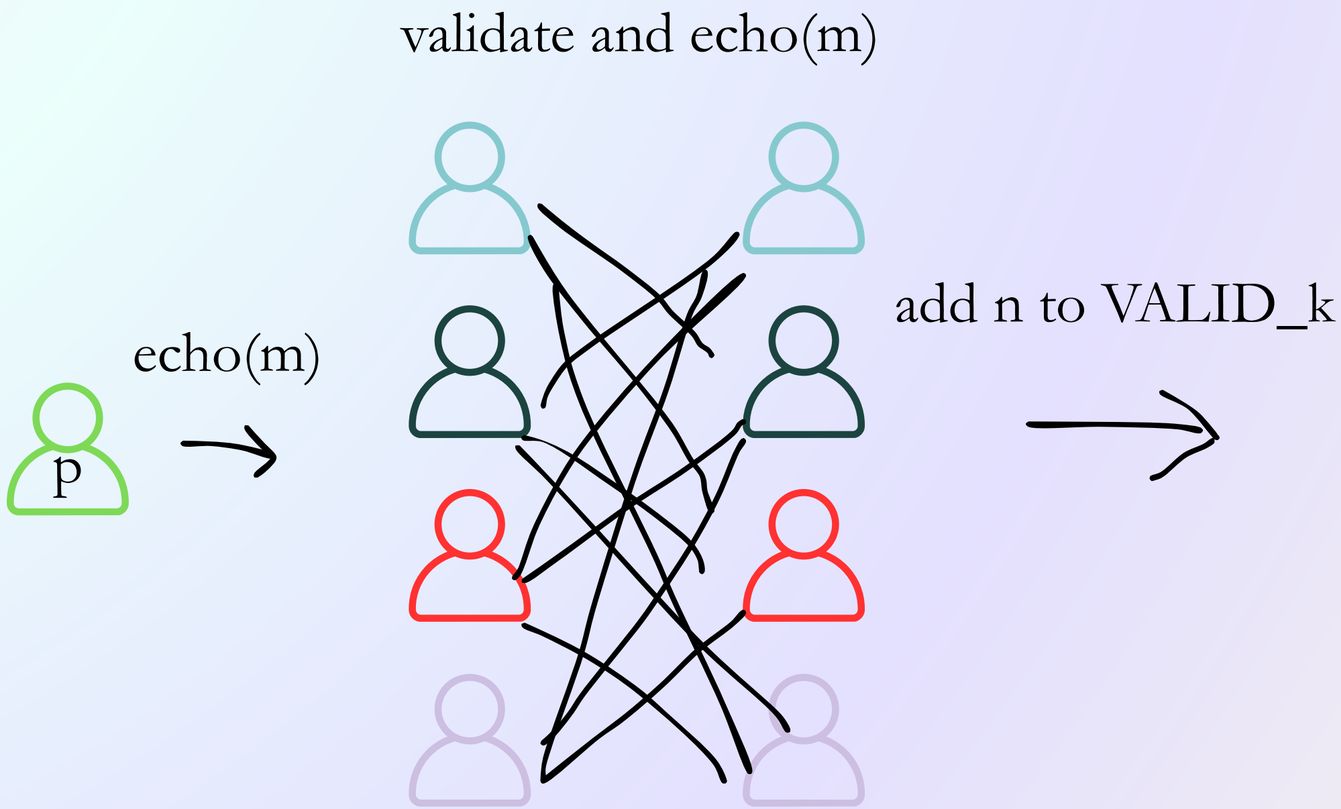
Lemma 5

If two correct processes, p and q, validate (r,k,v) and (r,k,u) messages, respectively, then u=v.



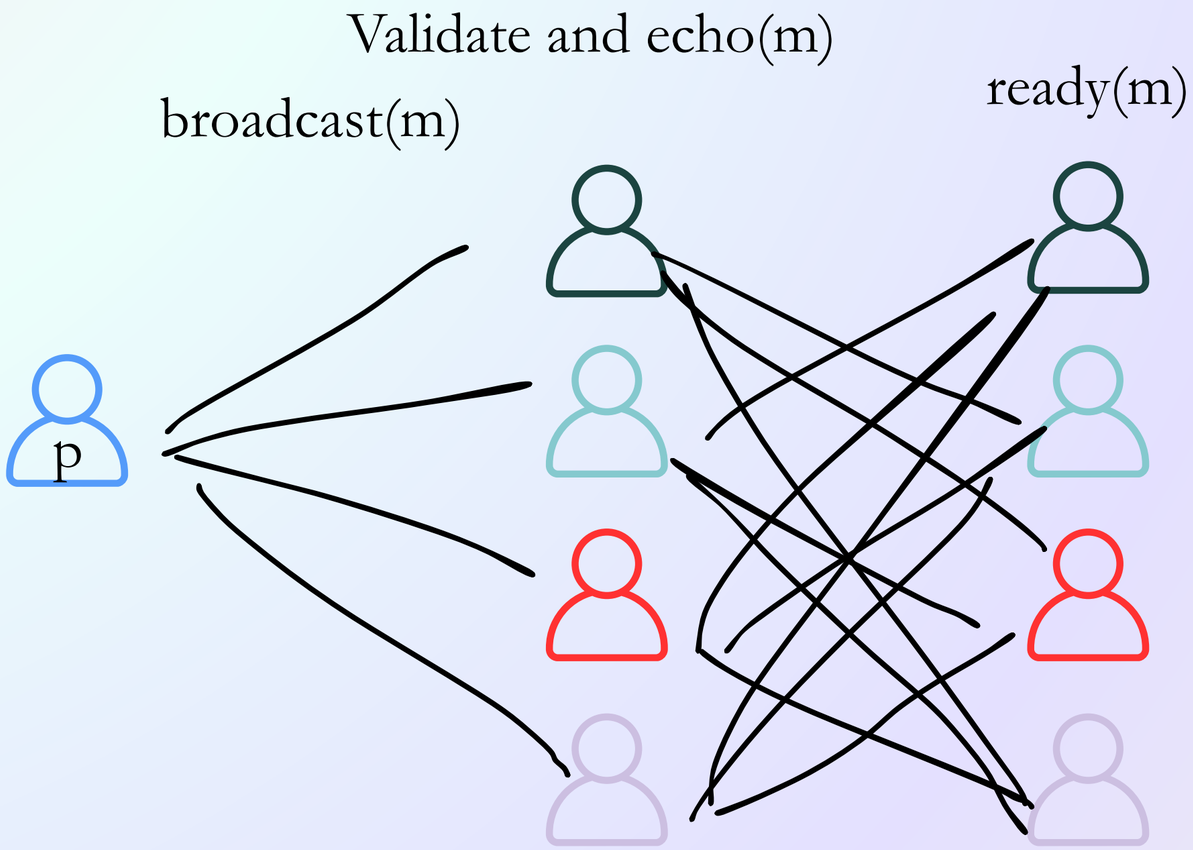
Lemma 6

If a correct process p validates a k-message m, then every other correct process q validates m, i.e., if p and q are correct then VALID_kp = VALID_kq.



Lemma 7

If a correct process p broadcasts a k-message m, then every correct process q eventually validates m.



Validation Correctness

Validation Lemmas (Ensure Correct Behavior Even From Byzantine Nodes)

Lemma 5 → Consistent Validation

If two correct processes validate messages for the same sender and round, they must validate the same value because validation is based on reliable broadcast.

Conclusion: Correct processes cannot validate different values for the same round.

Lemma 6 → Same VALID Sets for All Correct Processes

If one correct process validates a message in round k, every other correct process will eventually validate it as well.

Conclusion: All correct processes eventually have the same VALID set for round k.

Lemma 7 → Correct Broadcast Always Gets Validated

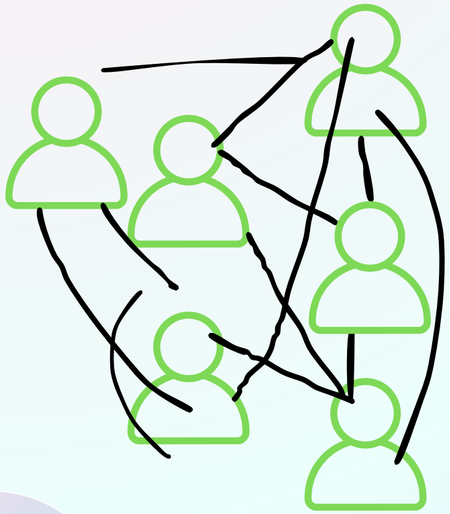
If a correct process broadcasts a message for round k, every correct process will eventually accept and validate that message.

Conclusion: Any message broadcast by a correct process becomes validated by all correct processes.

Takeaway: All honest nodes maintain a consistent set of valid messages in every round.

Consensus Protocol

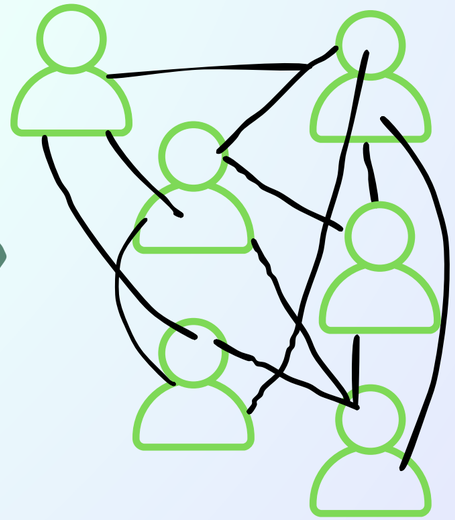
Proposal



Wait until n-t
validated messages
from current round

v = majority value

Vote



Wait until n-t
validated messages
from current round

v = (d,v) if > n/2
msgs have v, else v=v

Decision/Forced Value

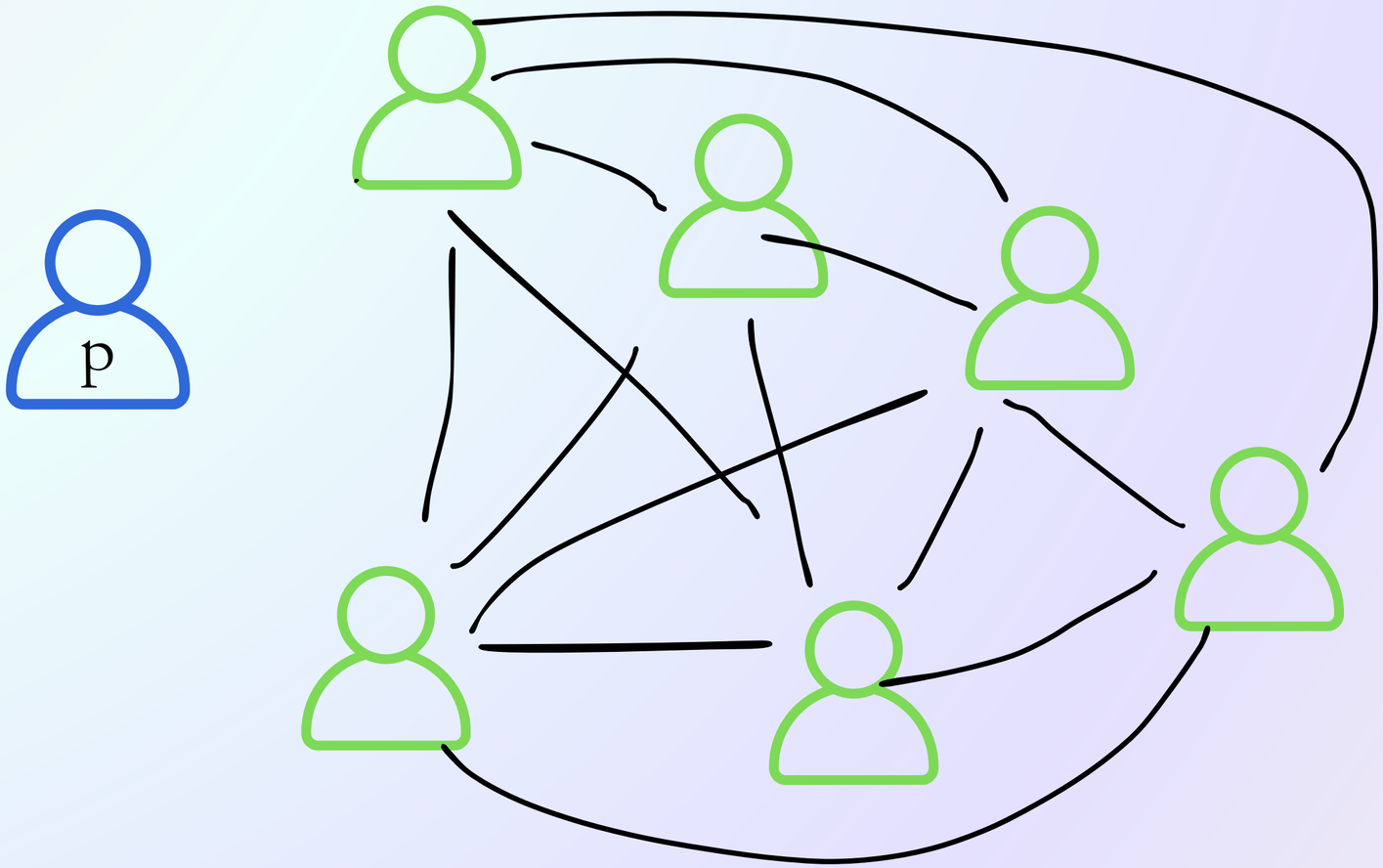
Wait until n-t
validated messages
from current round

If > 2t (d,v) msgs, decision = v,
if > t (d,v) msgs, v = v,
else v = rand(0,1)

Next Phase

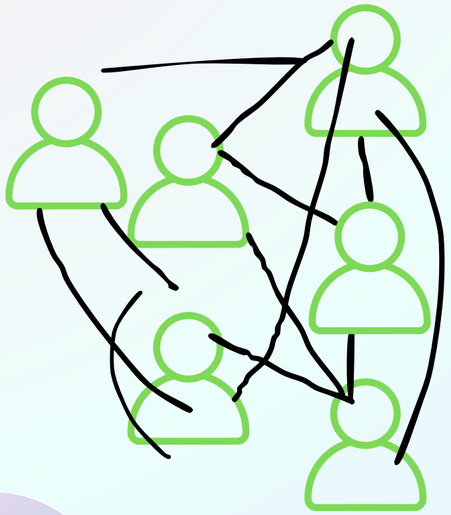
Lemma 8

If a correct process p is at round i, then p will eventually progress to round i+1.



Lemma 9

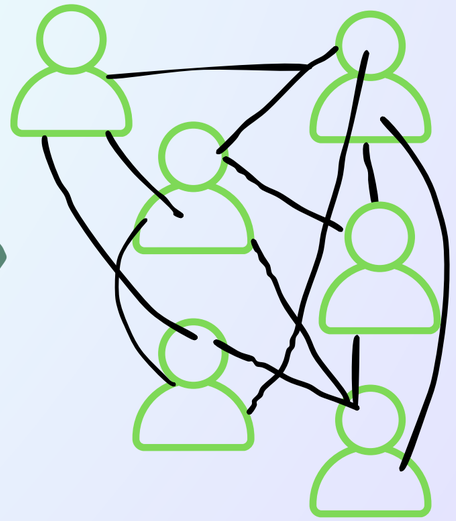
If at the beginning of round 3r+1 all correct process have the same value v, then they all decide v at round 3r+3.



V

Wait until n-t
validated messages
from current round

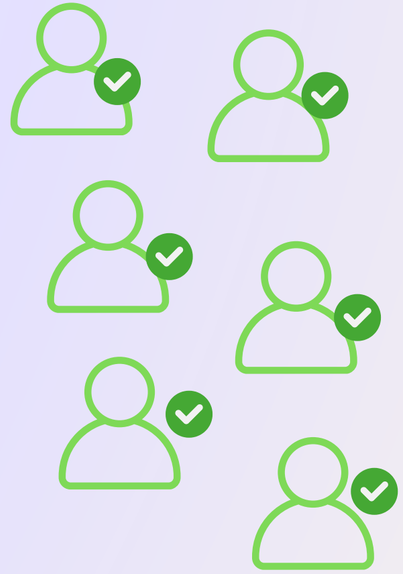
v = majority value



V

Wait until n-t
validated messages
from current round

v = (d,v) if > n/2
msgs have v, else v=v



V

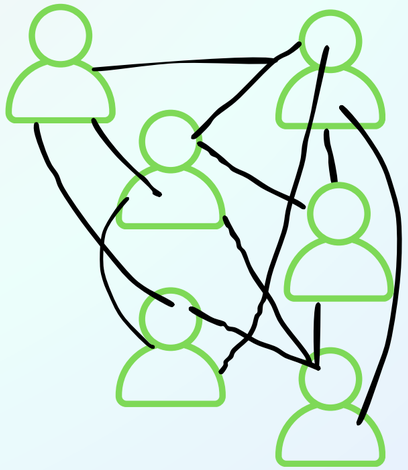
Wait until n-t
validated messages
from current round

If > 2t (d,v) msgs, decision = v,
if > t (d,v) msgs, v = v,
else v = rand(0,1)

Lemma 10

Let p and q be correct processes. If at phase r, p validates a message with value (d,v), and q validates a message with value (d,u) message, then u=v.

Vote



v = (d,v) if > n/2 msgs have v, else v=v

Decision/Forced Value

If > 2t (d,v) msgs, decision = v, if > t (d,v) msgs, v = v, else v = rand(0,1)

Correctness Proof (Consensus Protocol)

Consensus Lemmas (Phase-level Guarantees)

Lemma 8 → Progress (No Deadlock)
If a correct process reaches a round, it will eventually receive enough validated messages to move to the next round.
Conclusion: All correct processes always progress to the next round.

Lemma 9 → Validity of Consensus Value
If all correct processes begin a phase with the same value, that value remains dominant and they all decide it in that phase.
Conclusion: If all correct processes start with the same value, they all decide that value.

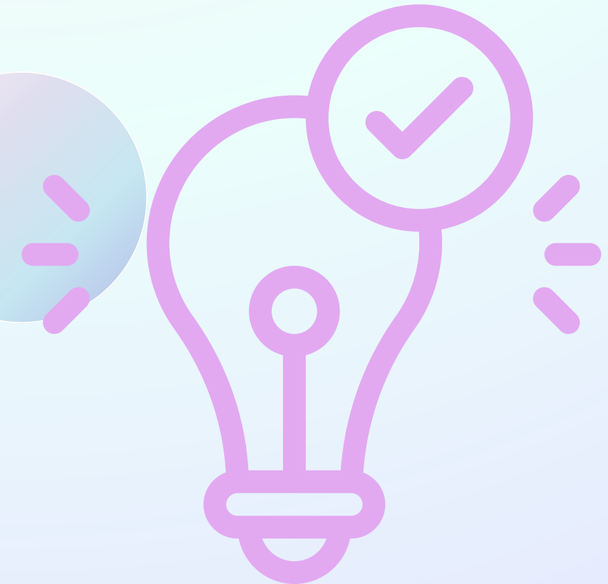
Lemma 10 → No Conflicting Forced Values
If two correct processes validate forced decision values in the same phase, they must have validated overlapping majorities.
Conclusion: Two correct processes cannot validate different forced values.

Purpose: These lemmas ensure the consensus protocol satisfies validity, agreement, and makes progress.

Theorem 2 : Consensus Correctness

Theorem 2 - t-Resilient Consensus ($t < n/3$)

- Validity: If all start with v, all decide v (Lemma 9).
- Agreement: If one correct process decides v, all eventually decide v (Lemmas 6 & 9).
- Termination: Every correct process decides with probability 1 (randomization).
- Conclusion: Protocol 2 is a t-resilient asynchronous consensus protocol for $t < n/3$.



Step 1: Input Consistency
Same inputs → same result

Step 2: Decision Uniformity
One honest decision forces all honest nodes to match

Step 3: Guaranteed Progress
Randomization prevents infinite waiting

Final Outcome
Consensus always reached under $t < n/3$ Byzantine

Termination and Optimality

Termination Proof Summary

Each phase ends either by:

A process forced to value v → others adopt v (Lemma 10).

OR

No forced value → all flip coins and eventually align by chance.

Termination Guarantee:

- Each round has a chance of success.
- Even if agreement doesn't happen right away, the system keeps trying.
- The longer it runs, the smaller the chance of failure becomes.
- Eventually, consensus is guaranteed to happen.

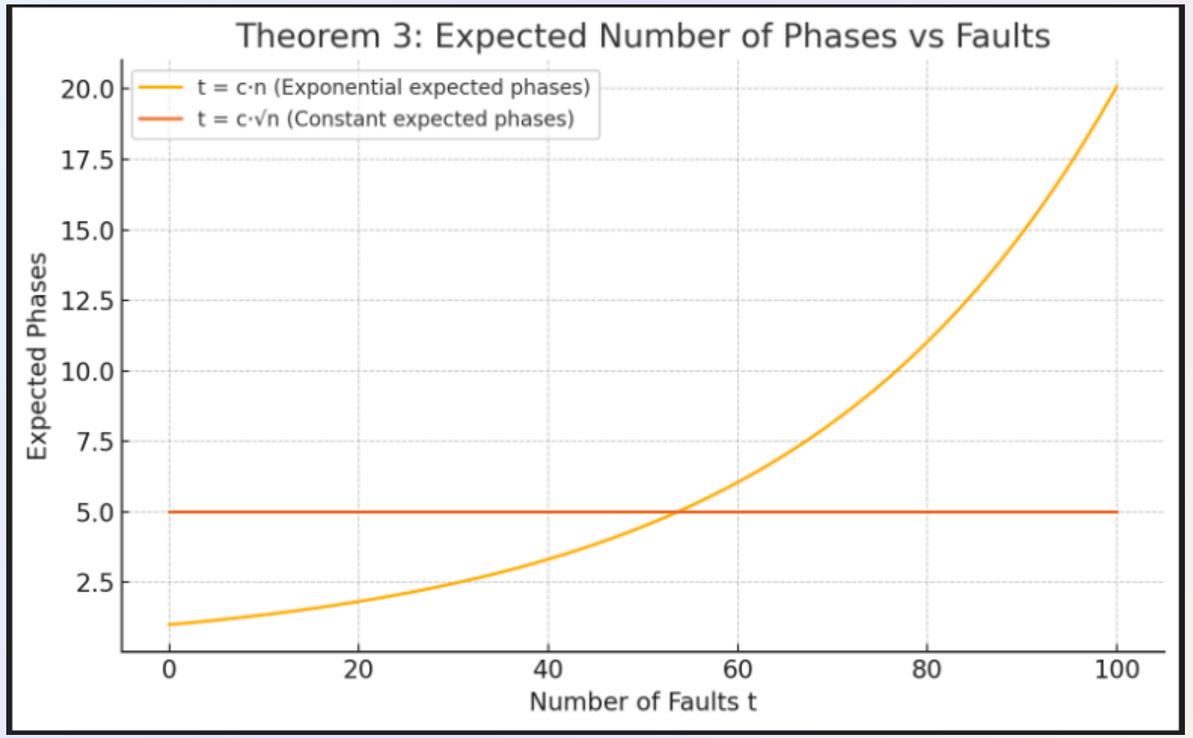
Optimality Result

- Bracha & Toueg (1985): No randomized asynchronous consensus protocol can tolerate $t \geq n/3$.
- Therefore, this protocol achieves the maximum possible fault tolerance.

Theorem 3: Performance Analysis

Theorem 3 - Expected Number of Phases

- (i) If $t = c \cdot n$, expected phases → exponential in n.
- (ii) If $t = c\sqrt{n}$, expected phases → constant (independent of n).



Implications

- Fewer faults → faster consensus.
- Each phase = 3 asynchronous steps (initial, echo, ready).

In synchronous settings, each round ≈ 3 time units → constant expected runtime.

Conclusion: The protocol is efficient for moderate faults and scales reasonably under asynchrony.

Comparison with Rabin's Model

Rabin's Model

- Proposed a global coin toss (shared randomness).
- Requires a trusted dealer + secret sharing for coin distribution.

Bracha's Model

- Uses local, independent coin tosses (no trusted dealer).
- Weaker assumptions → stronger realism.
- Slower expected termination but greater applicability.

Result:
Same fault tolerance ($t < n/3$) with a weaker, fully asynchronous model.

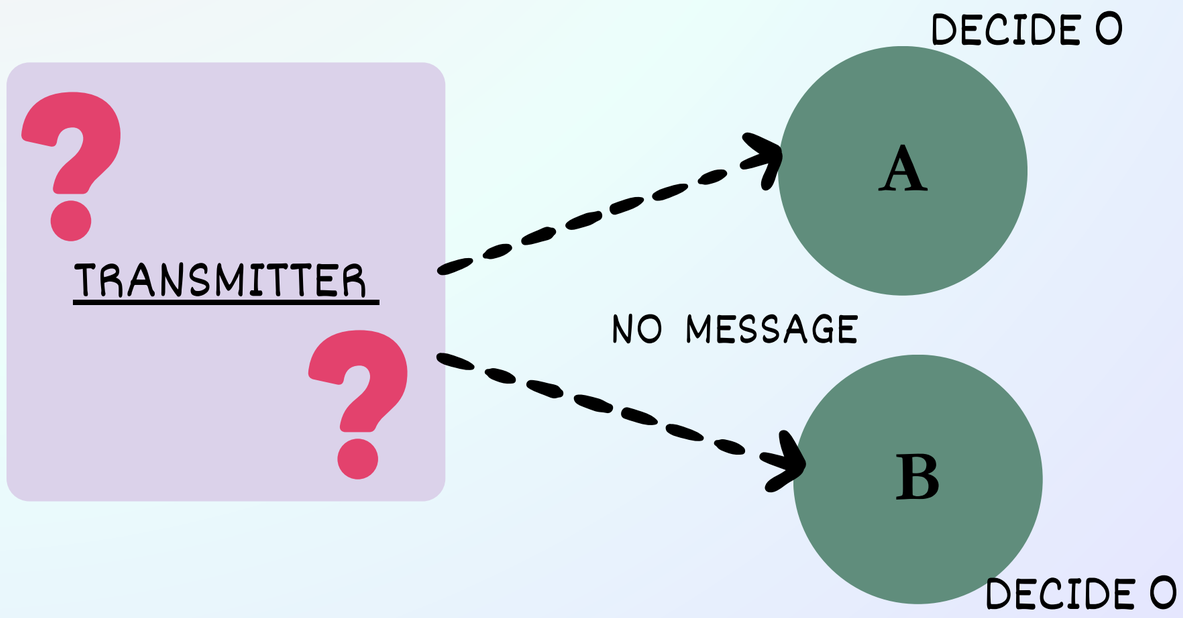
Theorem 4: Asynchronous Byzantine Generals Impossibility

- Byzantine Generals protocols are impossible in asynchronous systems, even with one fault.

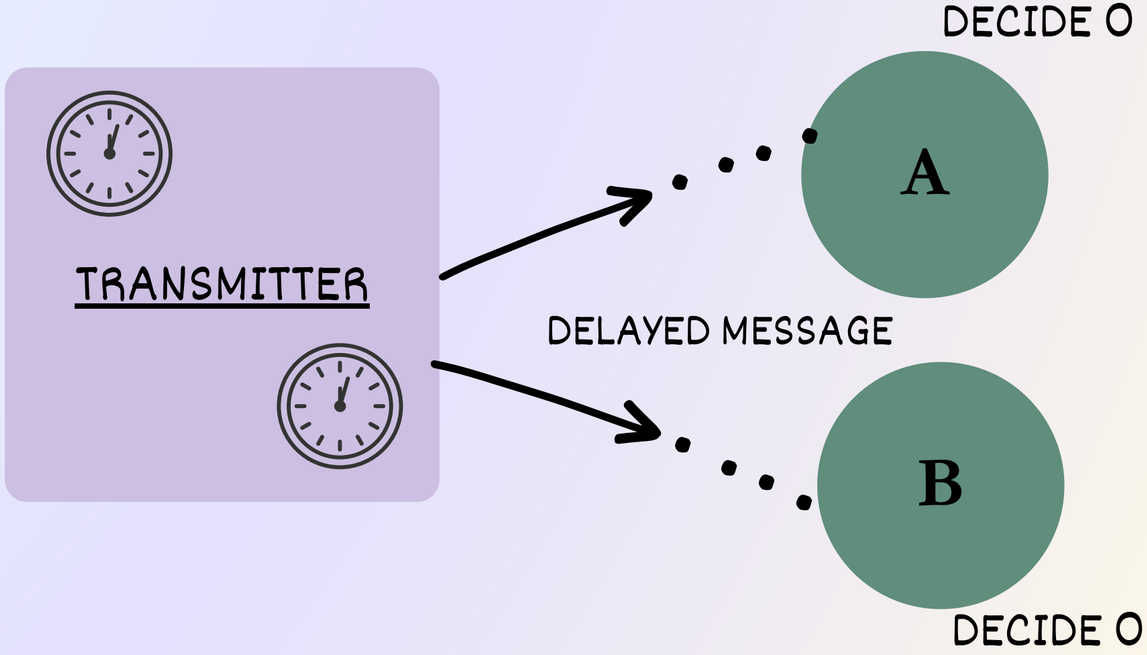
Scenario A: Faulty transmitter sends nothing → others decide 0 at time T_1.

Scenario B: Transmitter is correct but messages delayed until after T_1 → identical state view → decide 0 violating validity.

Asynchrony + Byzantine fault → no protocol can guarantee termination and validity together.



SCENARIO 1: If transmitter is faulty and silent, other nodes must eventually decide based on silence.



SCENARIO 2: If network delays mimic silence, honest messages are ignored, nodes again decide 0, violating validity.

Theorem 5: Weak Termination Possible iff t < n/3

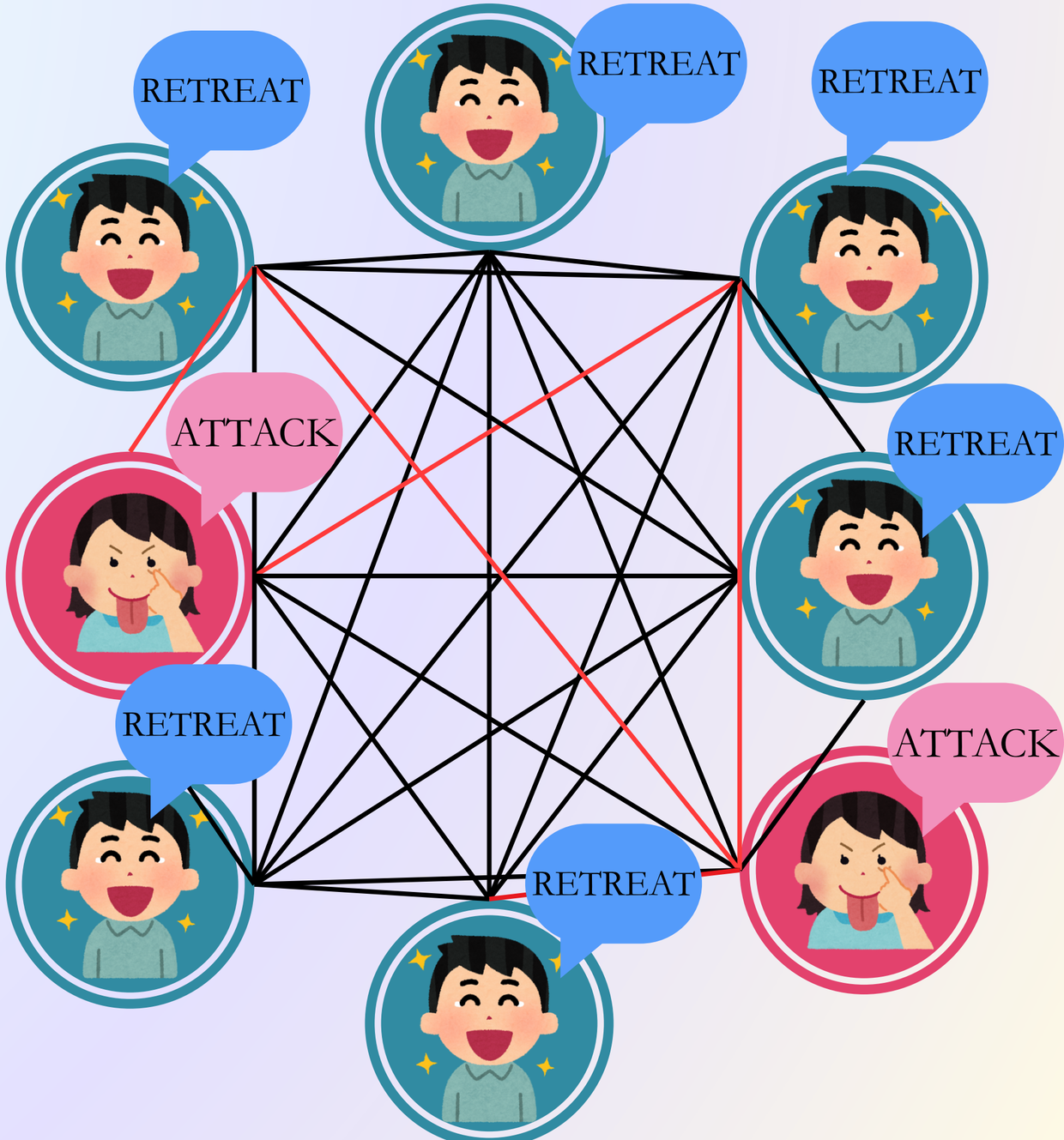
Weak Termination Definition

- If transmitter is correct → all decide.
- If transmitter faulty → either all decide or none decide.

Theorem 5 -Feasibility Bound

- Byzantine Generals protocol with weak termination is possible if $t < n/3$.
- Proof idea:
  - For $t \geq n/3$, construct three groups (A, B, C) of size t.
  - Faulty transmitter A can send 0 to B and 1 to C → different views → contradiction.
  - Hence the limit $t < n/3$ is both necessary and sufficient.

All loyal generals must agree, but Byzantine generals may send conflicting messages.



Conclusion

- Bracha’s randomized protocol proved that probabilistic methods can ensure consensus in asynchronous networks.
- Established the theoretical and practical blueprint for resilient distributed computing.
- Continues to influence fault-tolerant blockchain, cloud, and multi-agent systems today.

Thank You

Please ask if you have any questions