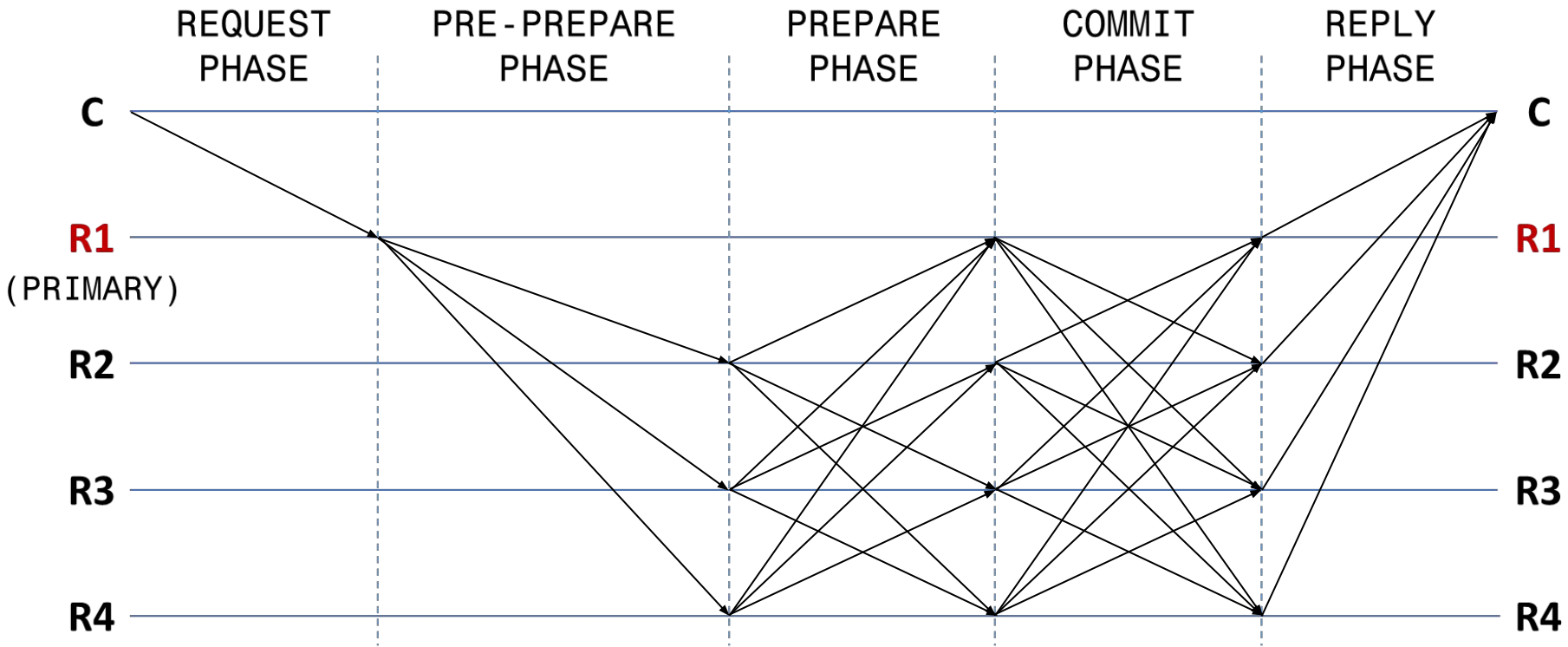
All You Need Is DAG

Tyler Culp Jason Eissayou Simon Draeger

Ananya Pandey Dhairye Gala

November 4, 2024

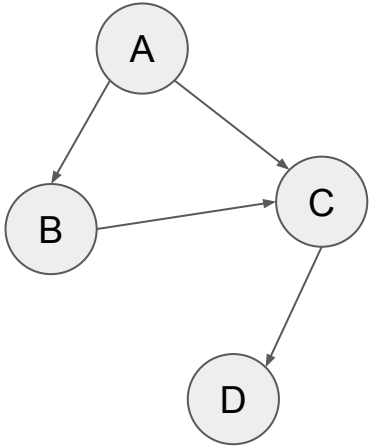
PBFT: Overview



PBFT: Shortcomings		DAG-Rider Improvements
Throughput bottlenecks around the primary	→	Distributes load across replicas
Multiple rounds of communication needed among replicas for ordering	→	Separate layers for ordering and communication => reduced overhead for ordering
View changes require quadratic communication	→	No view changes needed
Messages from non-faulty replicas can be dropped during view changes	→	All messages proposed by non-faulty replicas are guaranteed to be included
Designed for partially synchronous communication	→	Asynchronous communication

Terminology I

- Directed Acyclic Graph (DAG)
  - Graph with directed edges
  - Has no cycles (acyclic)
  - Vertices have an ordering (“topological ordering”)
  - Useful for flow-type relationships

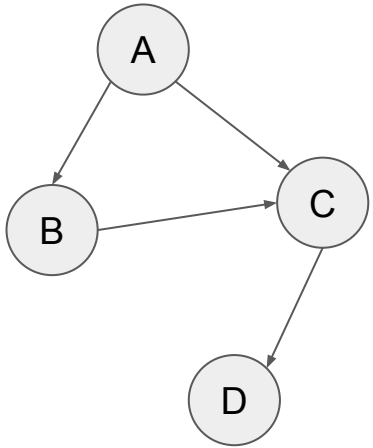


Order:

A → B → C → D

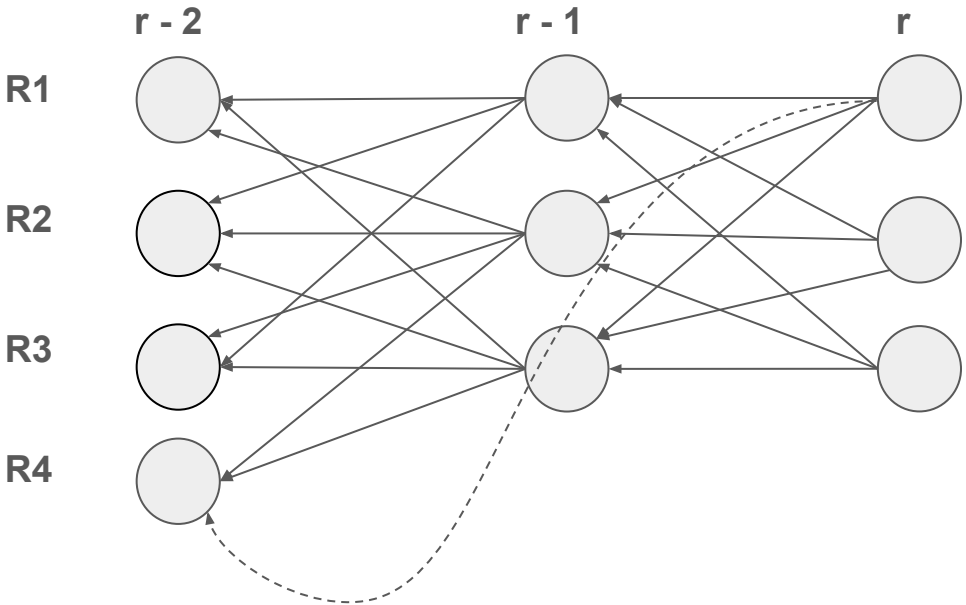
Terminology II

- Vertex: structure containing
  - Block of transactions
  - Set of weak + strong edges
  - Round # + source vertex of proposal



Terminology III

- Strong edge
  - Edge between vertex in round **r** and vertex in round **r - 1**
- Strong path
  - Path between 2 vertices consisting of only strong edges
- Weak edge
  - Edge between vertex in round **r** and vertex in round **r'** s.t. **r' < r - 1**
  - Validity: ensure all vertices have reference pointing to them


r - 2
r - 1
r
R1
R2
R3
R4

Assumptions I

- Reliable Broadcast (Bracha and Toueg, 1985)
  - Messages broadcast by correct replica eventually received by every correct replica
  - Ensures: **eventually** every replica has same DAG
  - DAG-Rider Propose phase: single reliable broadcast
- No timing assumptions (i.e., failure detector)

DAG-Rider: Overview I

- Operates in waves
  - Wave: 4 rounds
- Every correct replica: broadcast (propose) 1 message per round
  - PBFT: only primary proposes
- Proposal
  - Block of transactions
  - Set of vertices (outgoing edges)
  - Round #
- Vertex in DAG ↔ proposal
- These proposals will eventually become vertices in our DAG

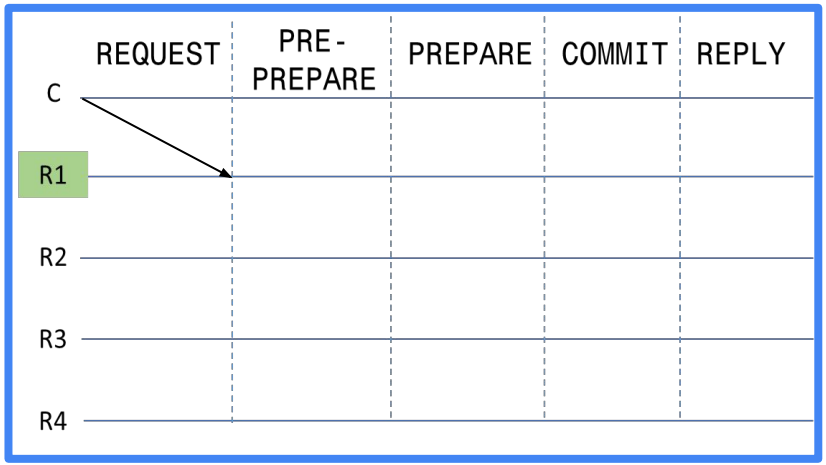
DAG-Rider: Overview II

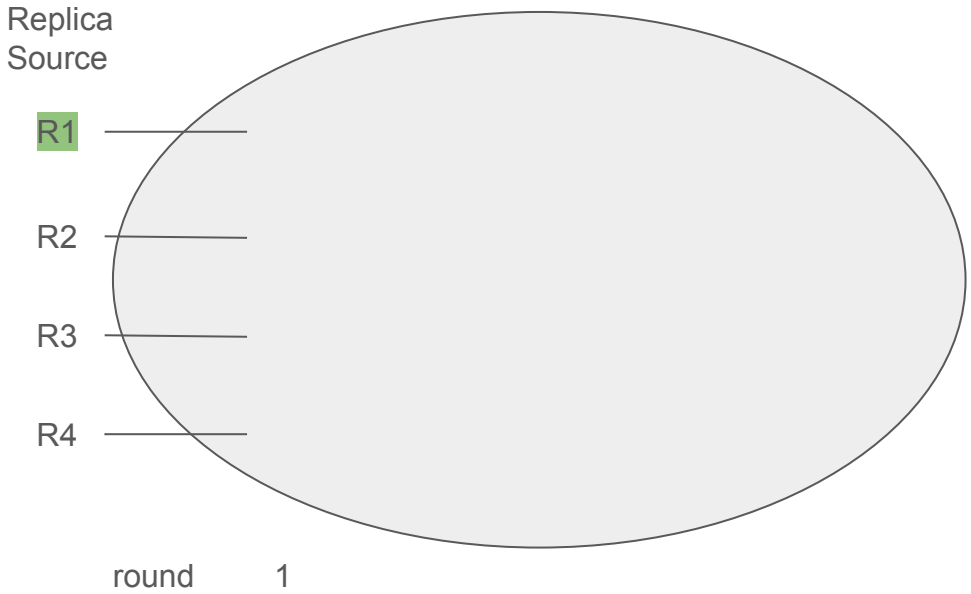
- Replicas incrementally construct a DAG over time
  - Can have unique views of DAG due to network delays
- After 4 rounds
  - Wave ends
  - “Leader” vertex chosen from first round
  - Leader chosen when **f + 1** replicas call *choose_leader(w)* with w = wave #
- Global perfect coin
  - *choose_leader(w)*
  - **f + 1** replicas call function with same input → will return
  - Unpredictability (adversary cannot predict wave leader)

Questions From Before

1. Why is it that we get preservation of nodes in the first round after a node in the second round is added?
2. Why is DAG-Rider able to commit transactions without having everything in a round?
    a. RCC must wait for everything in a round to come in before hand

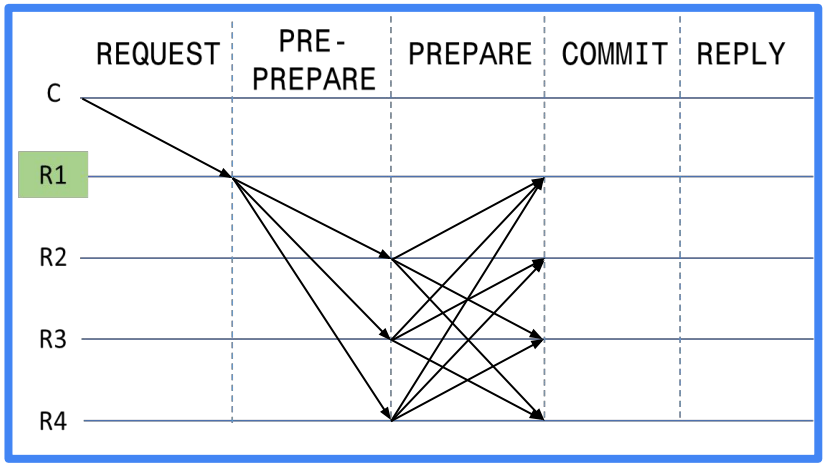
1. Why do we get preservation of nodes in the first round after a node in the second round is added?

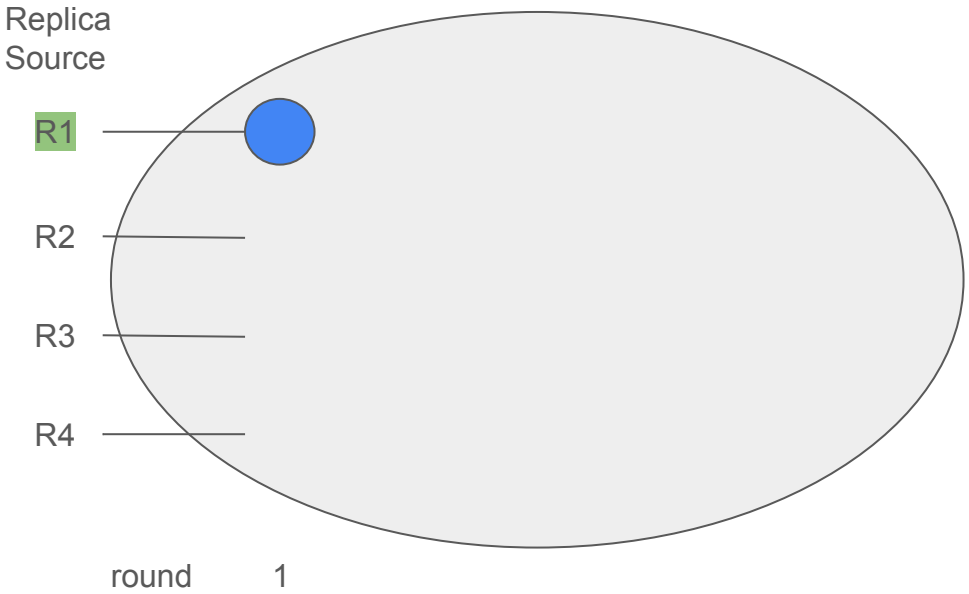




Wave w

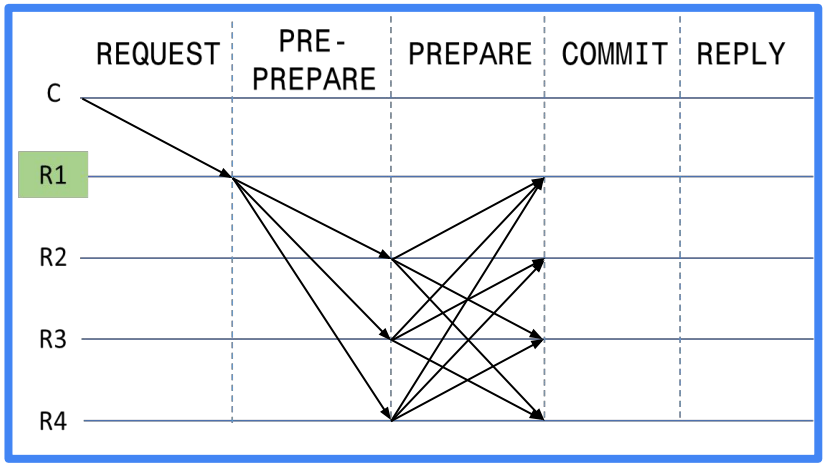
1. Why do we get preservation of nodes in the first round after a node in the second round is added?

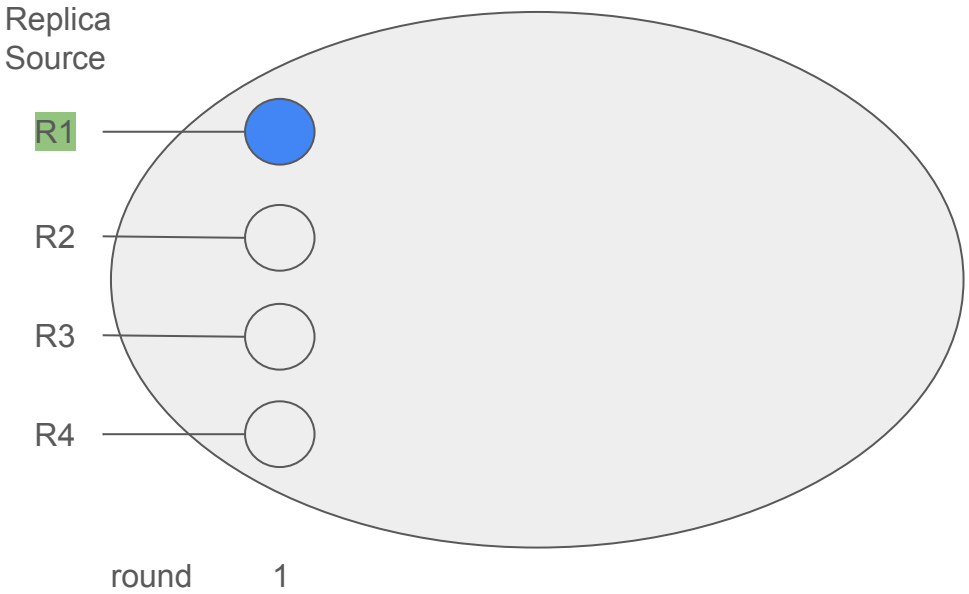




Wave w

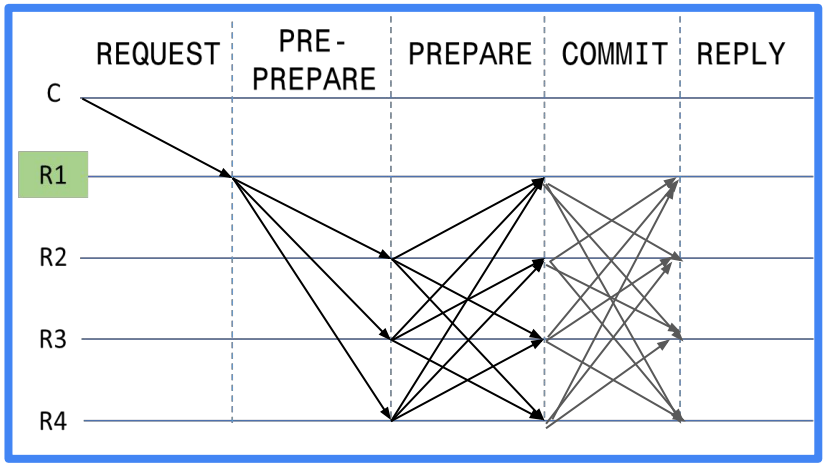
1. Why do we get preservation of nodes in the first round after a node in the second round is added?

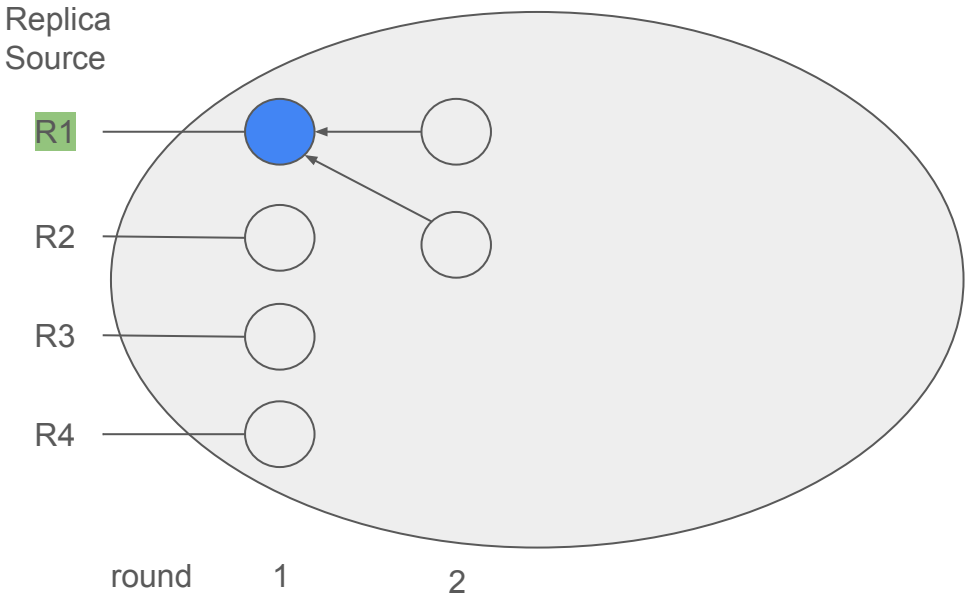




Wave w

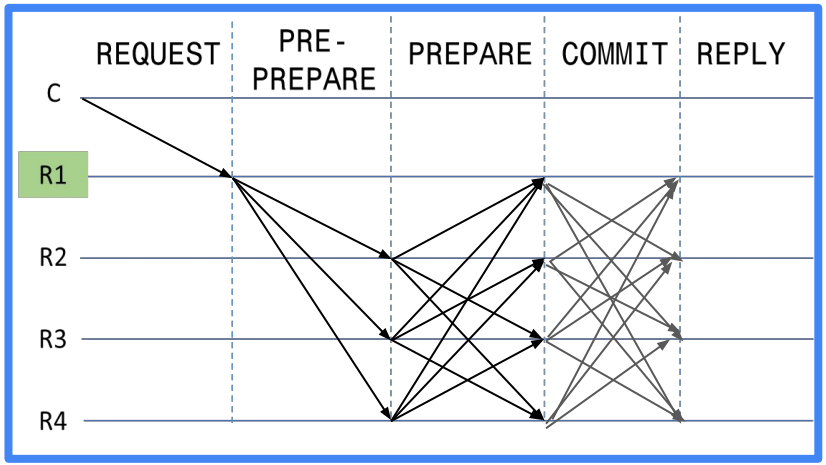
1. Why do we get preservation of nodes in the first round after a node in the second round is added?

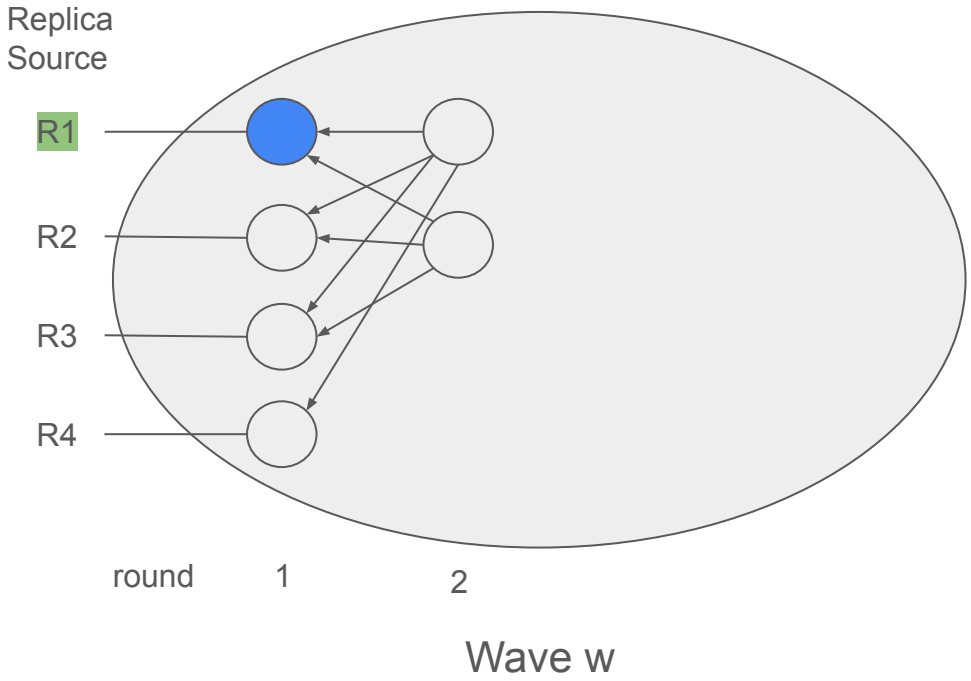




Wave w

1. Why do we get preservation of nodes in the first round after a node in the second round is added?



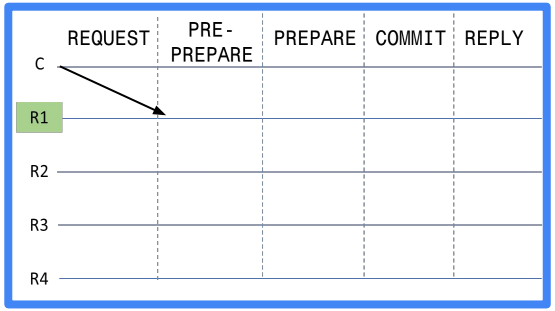


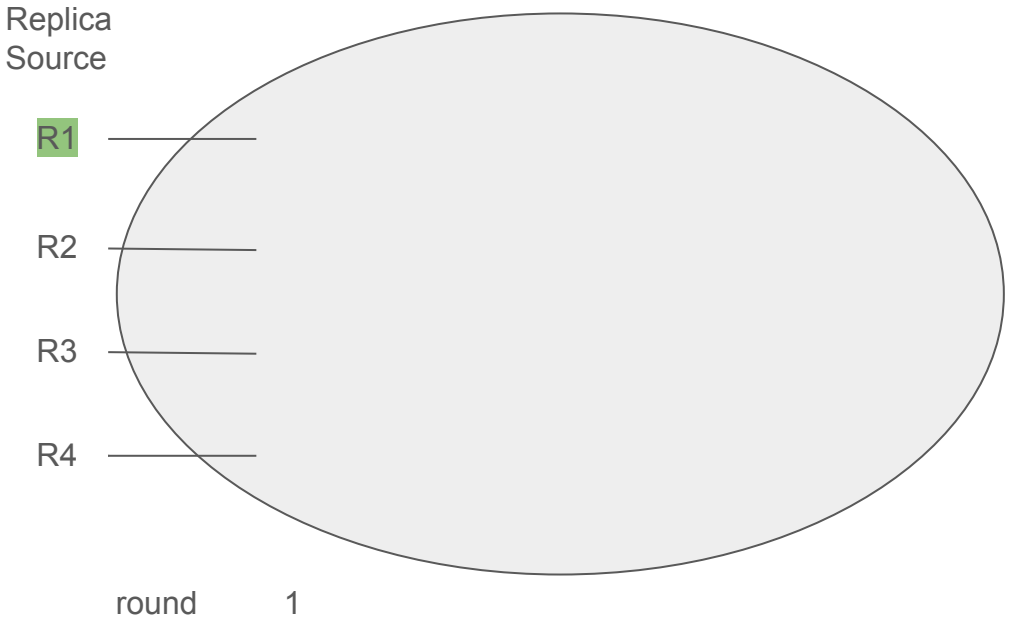
Why is DAG-Rider able to commit transactions without having everything in a round?

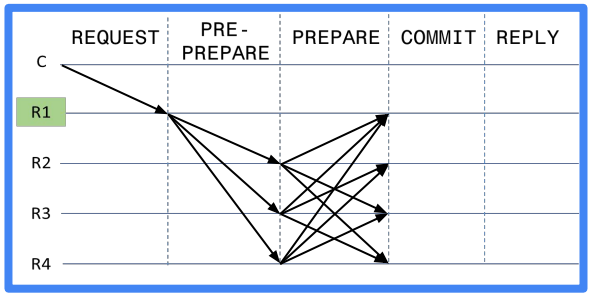
- In DAG Rider for a replica to move from round r to r + 1 it only needs to have seen 2f + 1 broadcasted messages for round r
    - Different from RCC as no longer need to see all 3f + 1 messages before moving on
- We have this looser condition because of our Reliable Broadcast Assumption
    - With Reliable Broadcast no replica can equivocate (or lie) about vertices they propose
    - Because of this we only need f + 1 vertices pointing to an earlier vertex v in the DAG for v to be considered **preserved**

Why is DAG-Rider able to commit transactions without having everything in a round?

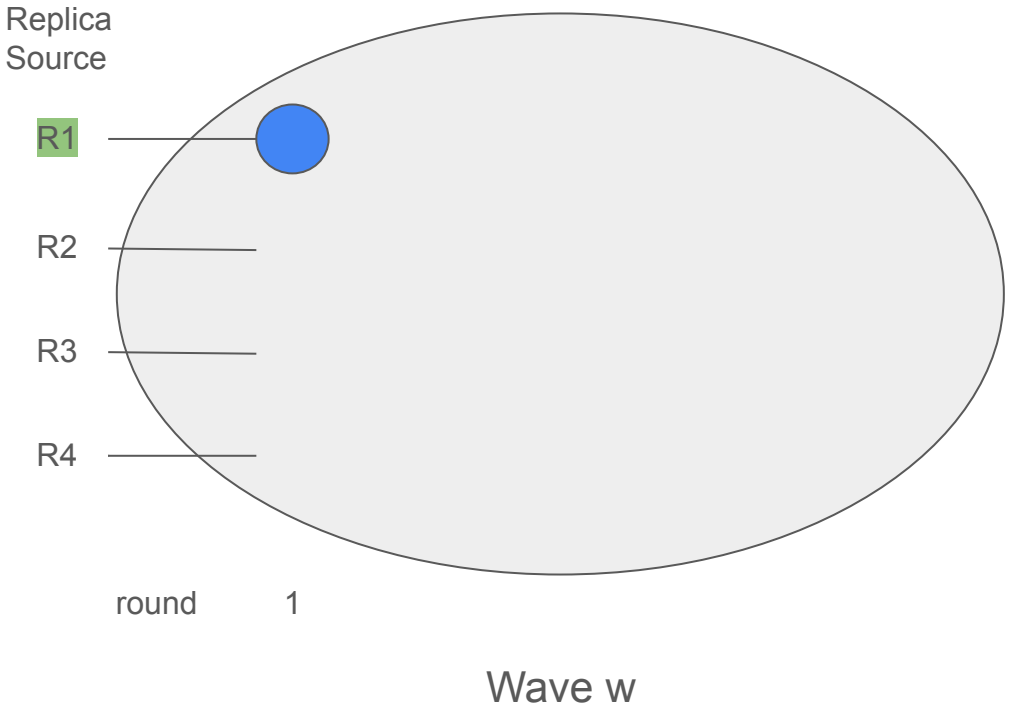
- Only needing f + 1 strong edges for preservation allows for replicas to move on with only 2f + 1 vertices in a round
- Reliable Broadcast also guarantees everyone will eventually catch up to a consistent view, forks are no longer a concern
- DAG-Rider uses notion of wave leaders
    - “Anchor points”
    - They are a single particular node in a wave which all replicas can synchronize on
    - Help guarantee that all replicas will converge on a single view of the DAG

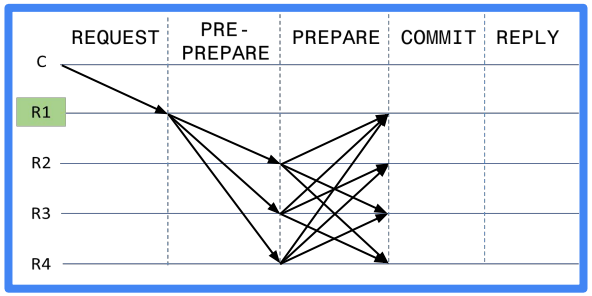

REQUEST
PRE-PREPARE
PREPARE
COMMIT
REPLY
C
R1
R2
R3
R4

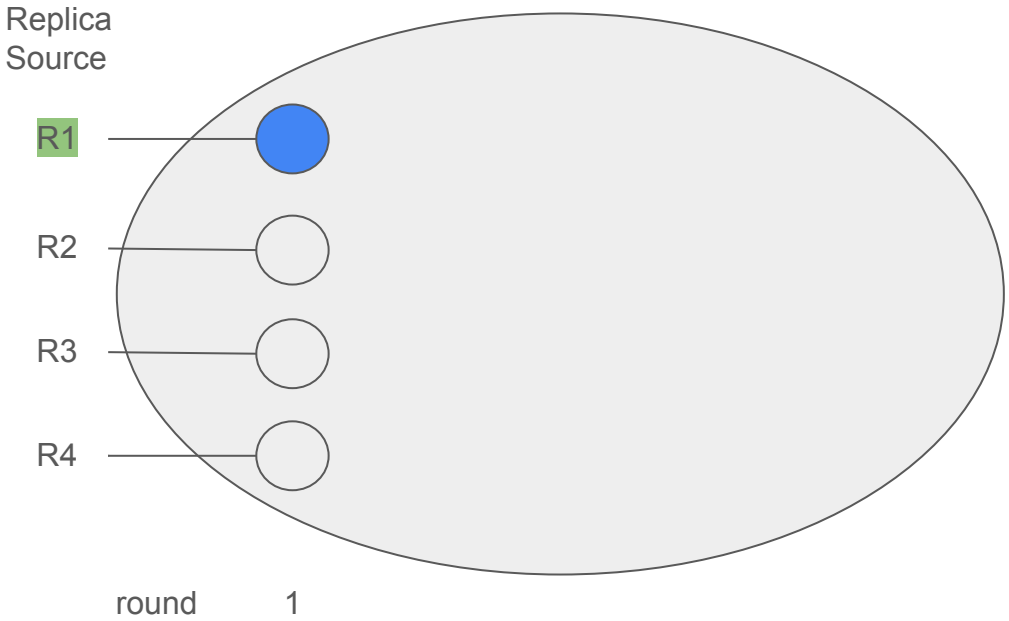

Replica Source
R1
R2
R3
R4
round 1
Wave w



1. This is the DAG from the perspective of R1.

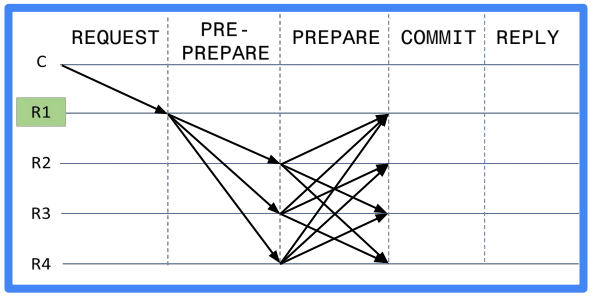


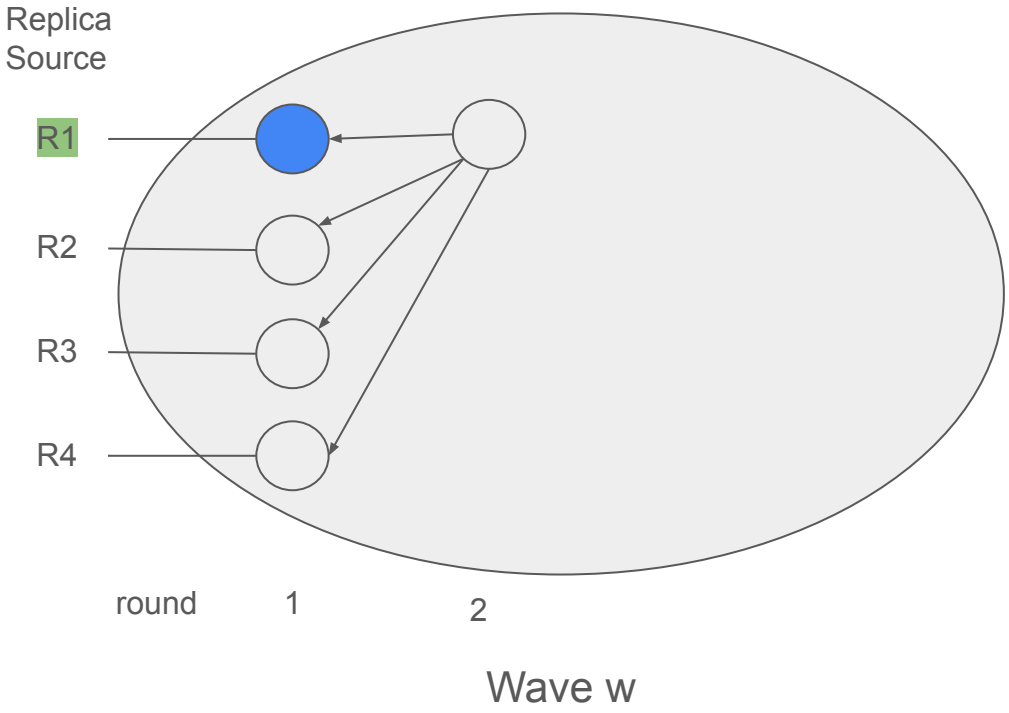




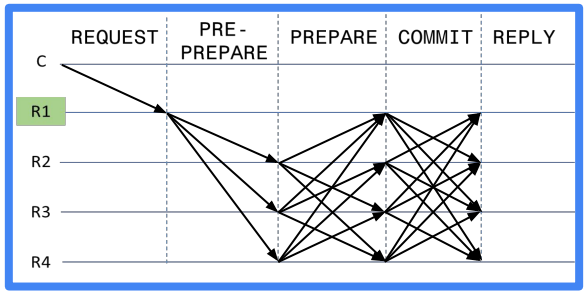
Wave w

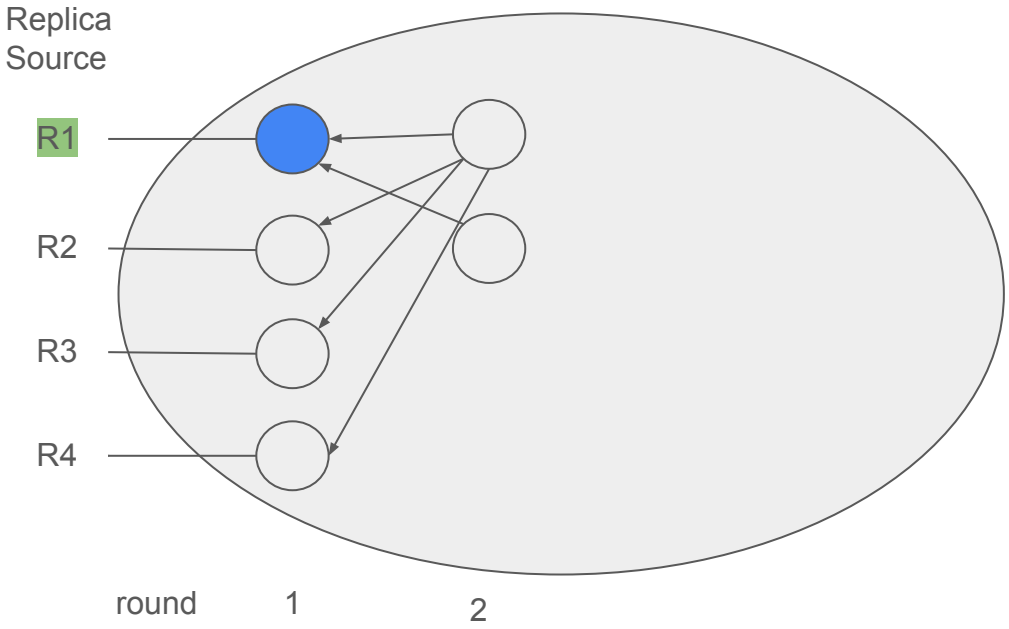
1. This is the DAG from the perspective of R1.
    a. It has seen a broadcasted message from all replicas in round 1





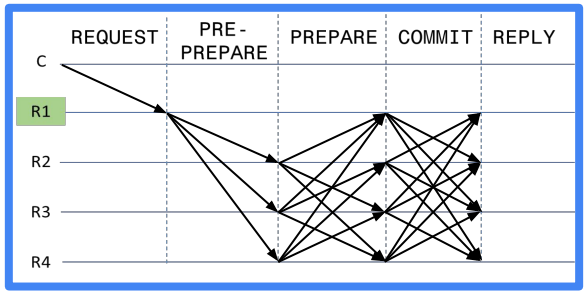
1. This is the DAG from the perspective of R1.
    a. It has seen a broadcasted message from all replicas in round 1
2. Once a replica has 2f + 1 vertices in a round of its local DAG it can move on to the next round
    a. It does this by broadcasting a new message (proposal) to all other replicas

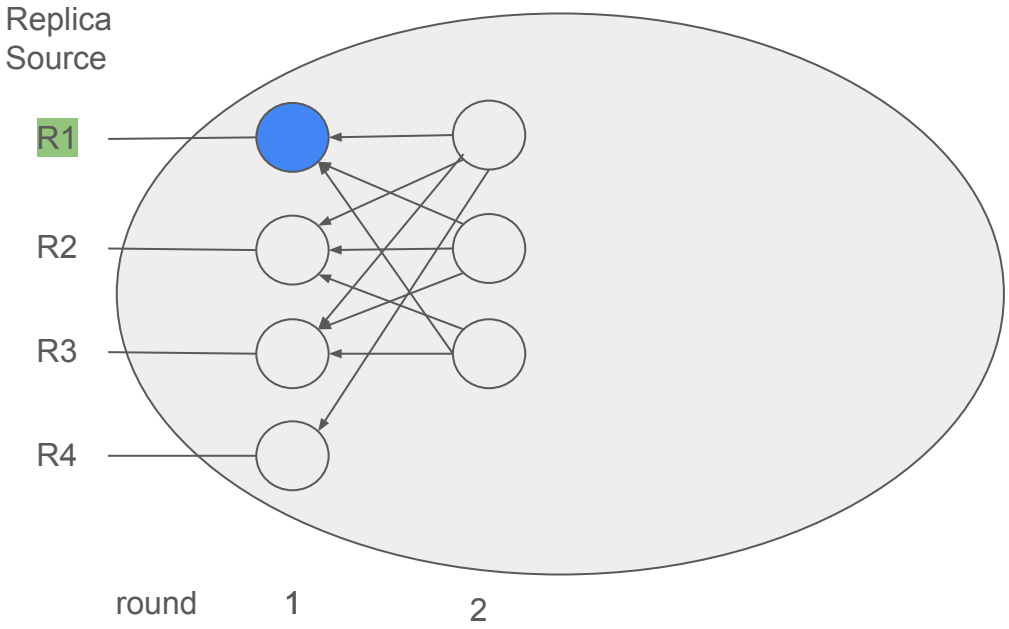




Wave w

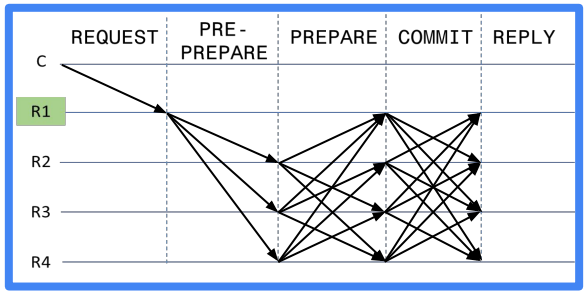
1. This is the DAG from the perspective of R1.
    a. It has seen a broadcasted message from all replicas in round 1
2. Once a replica has 2f + 1 vertices in a round of its local DAG it can move on to the next round
    a. It does this by broadcasting a new message (proposal) to all other replicas
3. The blue vertex is now preserved as f + 1 nodes have edges pointing to it

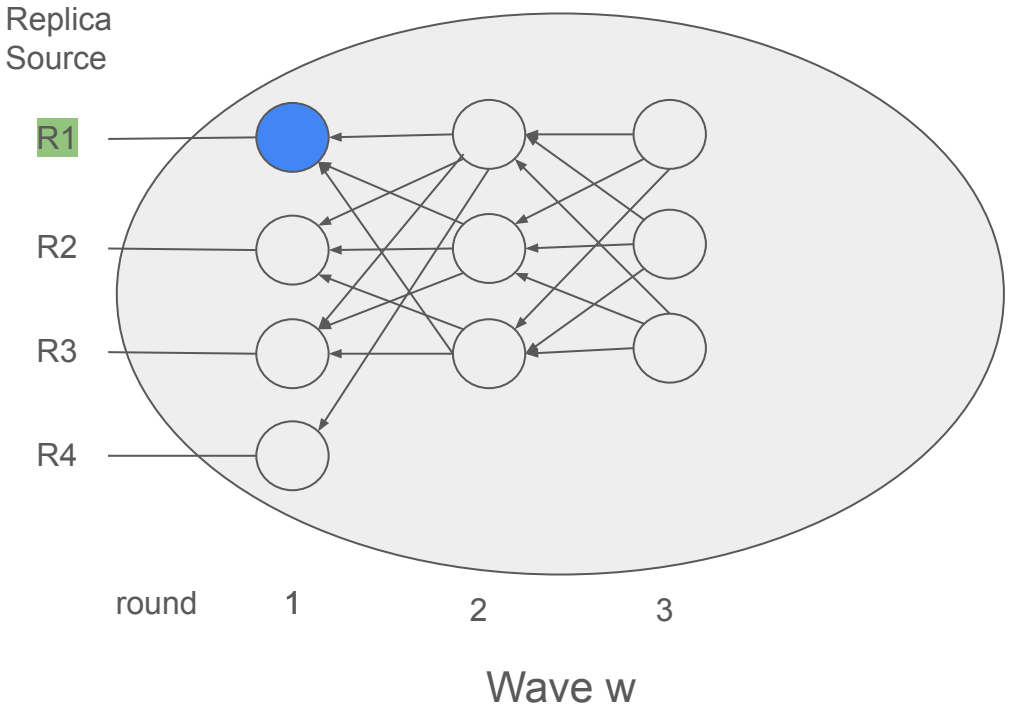




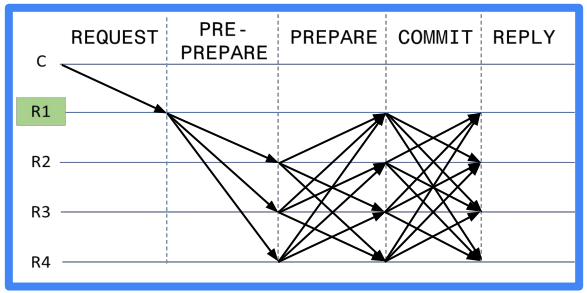
Wave w

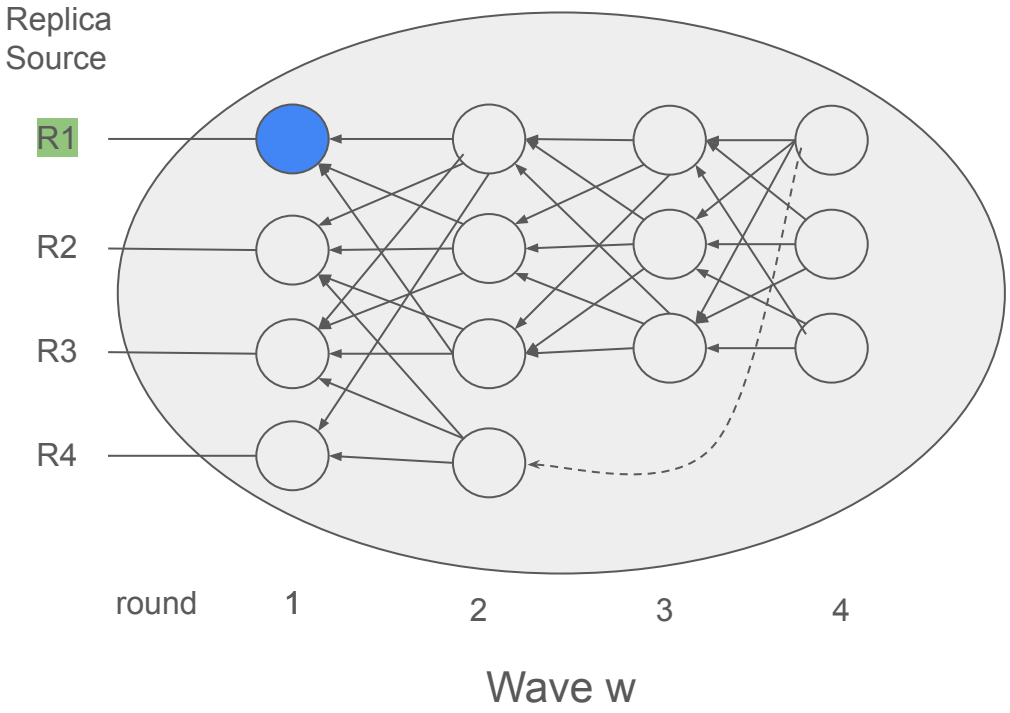
1. This is the DAG from the perspective of R1.
    a. It has seen a broadcasted message from all replicas in round 1
2. Once a replica has 2f + 1 vertices in a round of its local DAG it can move on to the next round
    a. It does this by broadcasting a new message (proposal) to all other replicas
3. The blue vertex is now preserved as f + 1 nodes have edges pointing to it
    a. Replica 1 continues adding seen broadcasts as vertices to its DAG



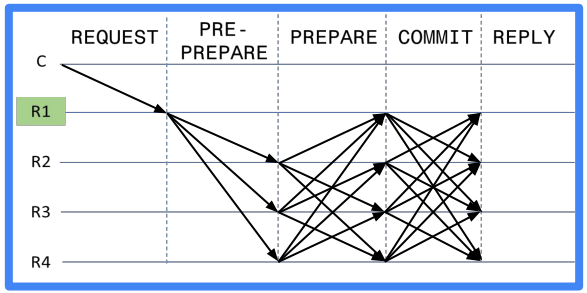


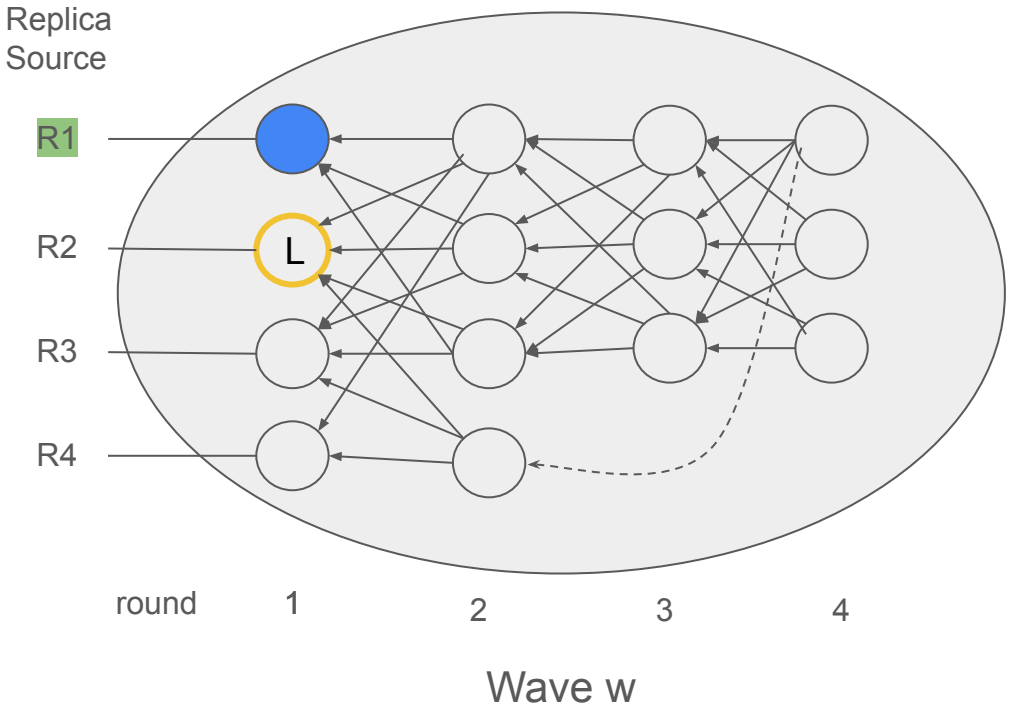
1. This is the DAG from the perspective of R1.
    a. It has seen a broadcasted message from all replicas in round 1
2. Once a replica has 2f + 1 vertices in a round of its local DAG it can move on to the next round
    a. It does this by broadcasting a new message (proposal) to all other replicas
3. The blue vertex is now preserved as f + 1 nodes have edges pointing to it
    a. Replica 1 continues adding seen broadcasts as vertices to its DAG





1. This is the DAG from the perspective of R1.
    a. It has seen a broadcasted message from all replicas in round 1
2. Once a replica has 2f + 1 vertices in a round of its local DAG it can move on to the next round
    a. It does this by broadcasting a new message (proposal) to all other replicas
3. The blue vertex is now preserved as f + 1 nodes have edges pointing to it
    a. Replica 1 continues adding seen broadcasts as vertices to its DAG
4. When a message arrives late the replica still adds it to its DAG, using a “weak” edge to connect it

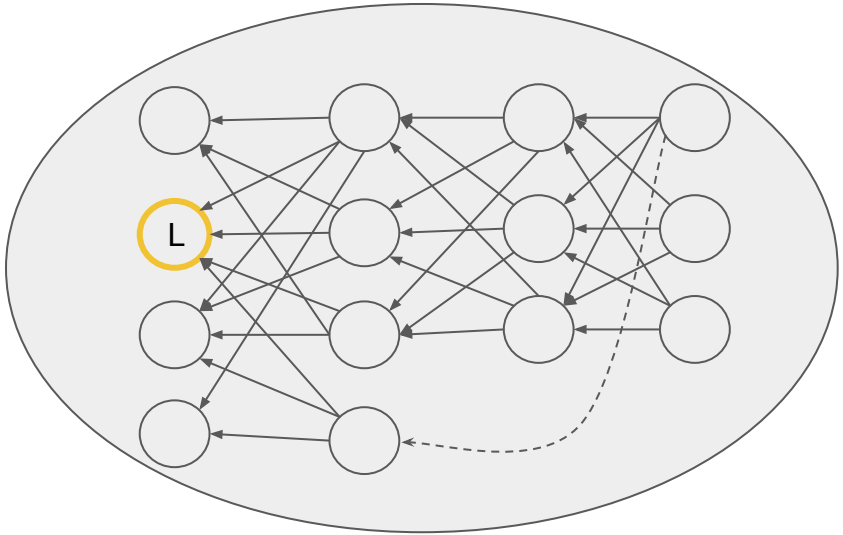




Wave w

1. This is the DAG from the perspective of R1.
    a. It has seen a broadcasted message from all replicas in round 1
2. Once a replica has 2f + 1 vertices in a round of its local DAG it can move on to the next round
    a. It does this by broadcasting a new message (proposal) to all other replicas
3. The blue vertex is now preserved as f + 1 nodes have edges pointing to it
    a. Replica 1 continues adding seen broadcasts as vertices to its DAG
4. When a message arrives late the replica still adds it to its DAG, using a “weak” edge to connect it
5. After 4 rounds a leader vertex is retroactively chosen from the first round

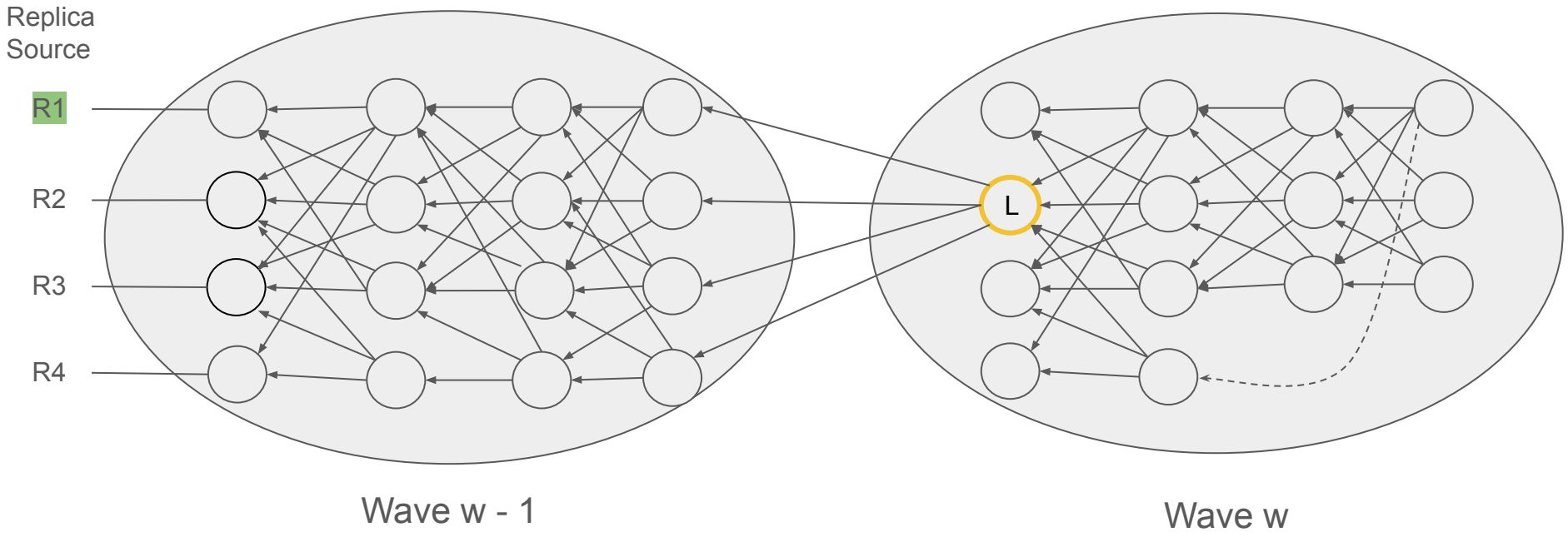
Causal History Commitment



Wave w

- After a wave leader is chosen it can be committed If 2f + 1 vertices in the last round of a wave have a strong path to the wave leader
    - After committing a wave leader, its causal history is deterministically ordered

Causal History Commitment



- This means all nodes in Wave w - 1 in leader L's history are ordered in a deterministic fashion after leader L is committed from wave w
    - After ordering, these nodes are "delivered" essentially marking them as done and saying that they have been successfully ordered

Every node has 5 important pieces of metadata

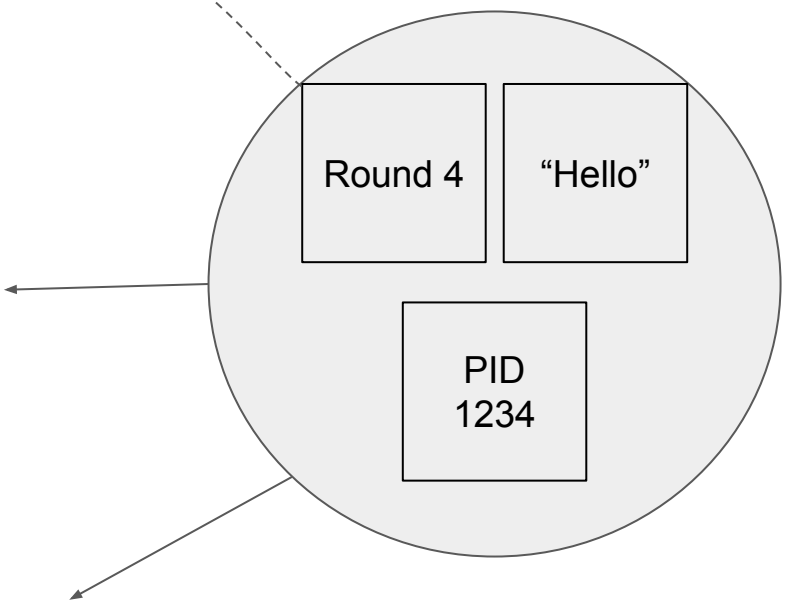
Transaction Block (Data): Holds the actual data or transactions that the process wants to add.

Round Number: Shows when the node was created, helping to keep nodes in the right order in the DAG.

Strong Edges: Connects to at least 2f+1 nodes from the last round, ensuring the DAG structure stays consistent across all replicas.

Source (Process ID): Identifies who created the node, making each one unique and preventing conflicting messages from the same process.

Weak Edges: Links to any older nodes not already connected, making sure late messages are included and no valid data is missed.



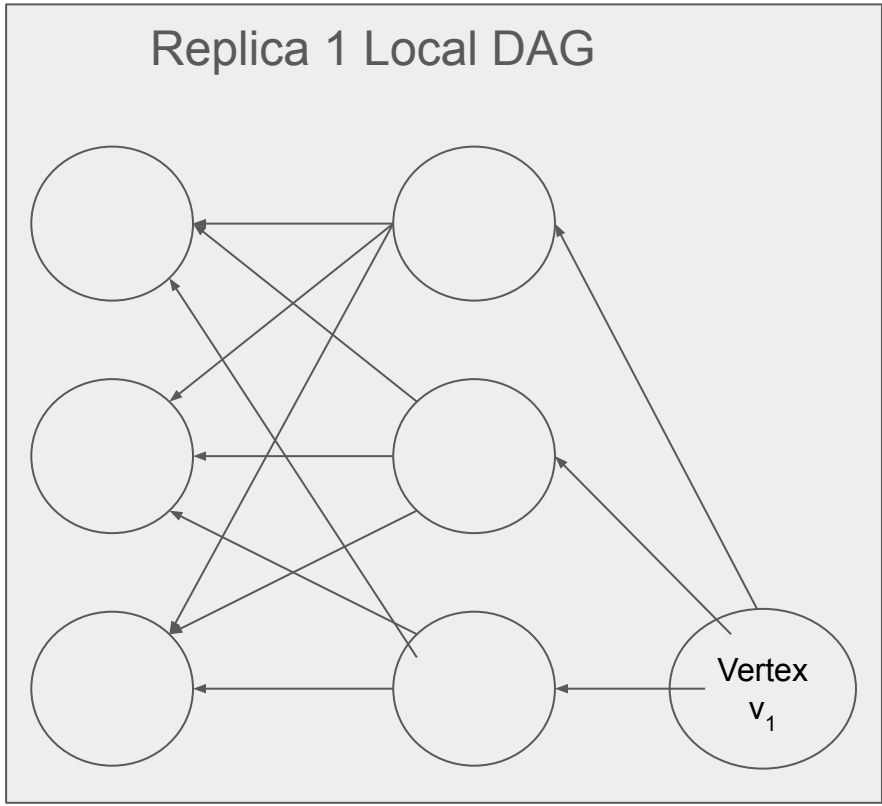
Buffer

A broadcasted vertex must reference at least 2f+1 vertices from the previous round to enter a replica's local buffer.

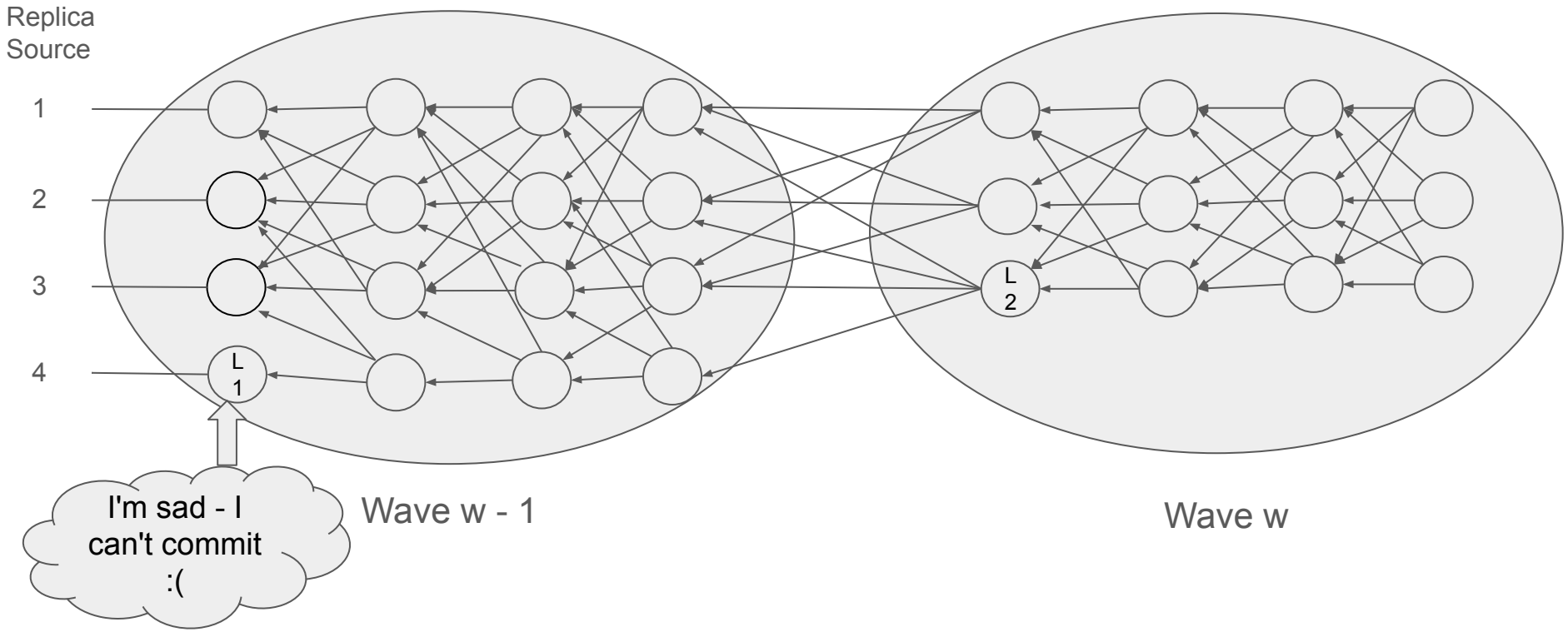
Once in the buffer, the replica repeatedly checks if all referenced vertices from the prior round are present in its local DAG.

When all dependencies are met, the replica removes the vertex from the buffer and adds it to its local DAG.

Replica 1 Buffer

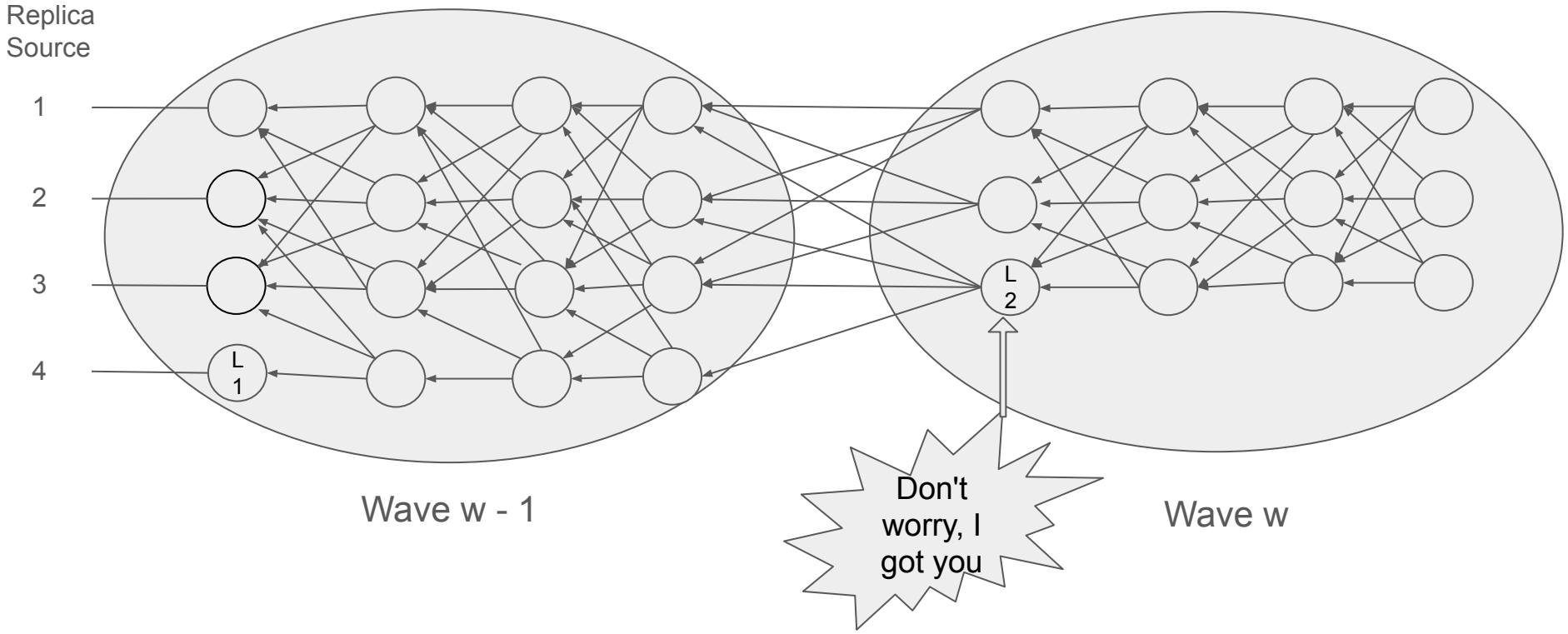


Handling Uncommitted Leaders in DAG Protocol



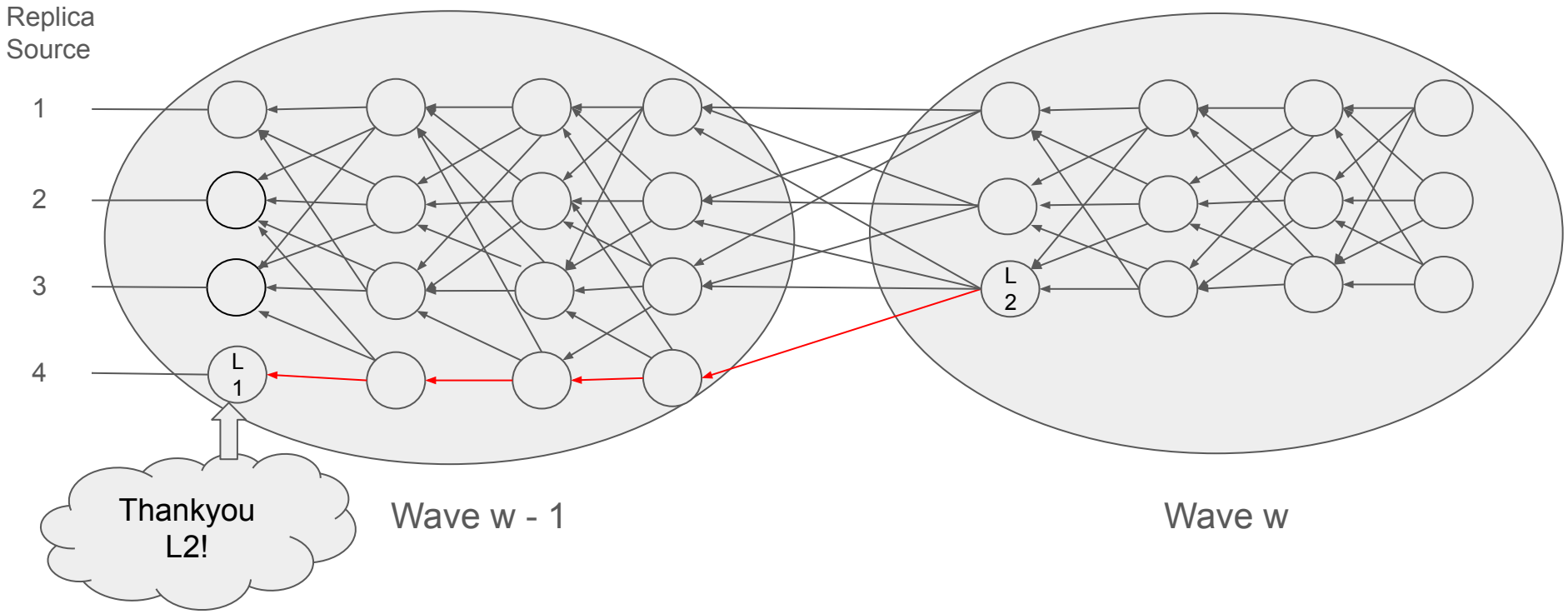
- The commit rule is not met in wave w-1 since there are less than $2f+1$ vertices in round 4 with a strong path to Leader L1.

Handling Uncommitted Leaders in DAG Protocol



- In wave w, L2 meets the commit rule with 2f + 1 connections.

Handling Uncommitted Leaders in DAG Protocol



- Since there's a strong path from L2 to L1, this allows L1 to be committed first.

Why do all replicas order transactions the same in the DAG-Rider protocol?

- Leader-Based Ordering: Global coin elects wave leaders, creating shared reference points for transaction sequence.
- Deterministic Causal History: Once a leader is committed, all replicas deliver transactions from the leader’s causal history.
- Consistent DAG Structures

Concluding DAG-Rider

- DAG-Rider introduces a new type of consensus protocols based on DAGs
- It uses a reliable broadcast abstraction to separate data dissemination and execution of transactions
    - Allows for an asynchronous protocol that can locally order proposals without extra communication
- Able to achieve higher throughput with all replicas proposing transactions at the expense of latency
- Removes the View Change step, simplifying consensus protocols and helping to achieve better throughput