CS 3351: Paxos Made Simple?

Amy Babay

University of Pittsburgh
School of Computing and Information

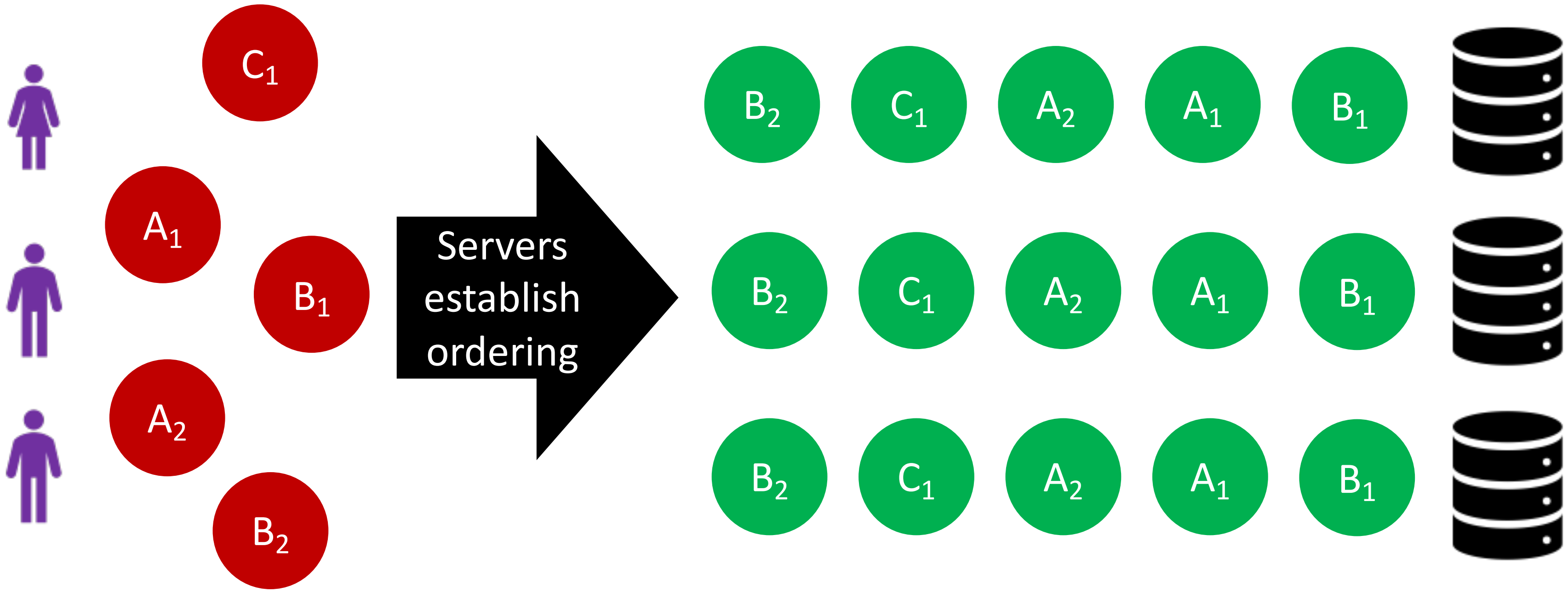
University of Pittsburgh

Background & Motivation

- **State Machine Replication (SMR)**: technique for implementing strongly consistent fault-tolerant services
  - Servers start in the same state
  - Servers apply deterministic updates in the same order
  - => Servers progress through exactly the same sequence of states

Background & Motivation

- **State Machine Replication (SMR)**: technique for implementing strongly consistent fault-tolerant services



Clients generate updates

Servers apply updates in order

Background & Motivation

- As discussed last class, SMR can be implemented as a sequence of **consensus** instances
- **Paxos** is a consensus protocol that is often used to implement SMR
  - Published in technical report in 1989, Journal paper in 1998 ("The Part Time Parliament")
    - Described via analogy to hypothetical Greek parliament
  - Work on clarifying, implementing, optimizing, or replacing Paxos continues in the literature today
  - Foundation for some of the intrusion-tolerant replication protocols we'll see later

System Model

- We have a groups of processes (aka. servers, replicas)
- Processes communicate by sending messages
- Processes can crash *and restart*
  - Processes have access to stable storage to record information
- Messages can take arbitrarily long to be delivered, can be duplicated, and can be lost
- Processes execute the protocol faithfully, and messages are not corrupted (non-Byzantine)

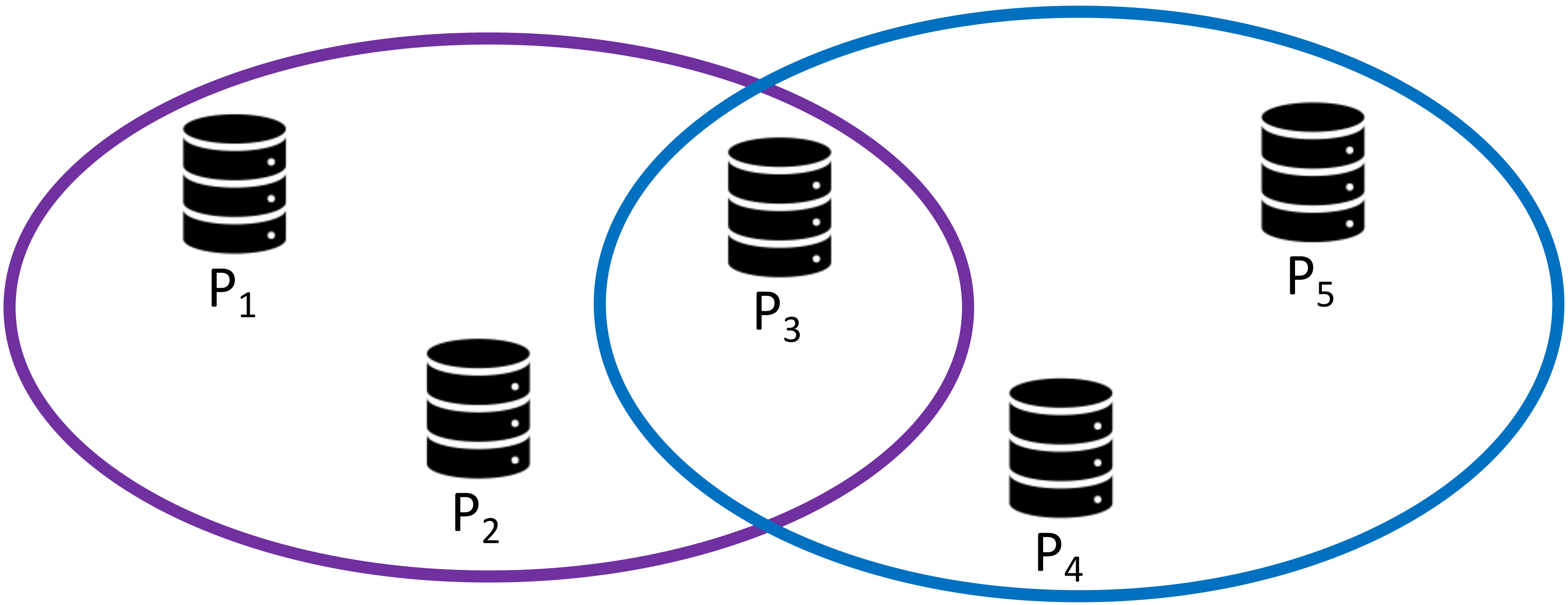
Consensus Requirements (Safety)

- **Validity**: Only a value that has been proposed may be chosen
- **Agreement (1)**: Only a single value is chosen
- **Agreement (2)**: A process never learns that a value has been chosen unless it actually has been

- What about liveness (progress/termination)?

Paxos Consensus Protocol ("Single-Decree Synod")

- Key concept: Any two majorities must intersect in at least one process



Paxos Consensus Protocol (“Single-Decree Synod”)

- Key concept: Any two majorities must intersect in at least one process
- So, to guarantee agreement we:
  1. Only allow a value to be *chosen* if it is accepted by a majority of (acceptor) processes
  2. Require a (proposer) process to communicate with a majority of (acceptor) processes before proposing a value to find out what they’ve previously accepted

Paxos Consensus Protocol

- Subtle points:
  - But what if multiple values have been accepted?
  - Can a process accept a new value after giving its response?
- Solutions:
  - Sequence numbers on proposer's attempts to pass a proposal (often called "view numbers")
  - Promises to not accept anything from lower views after responding

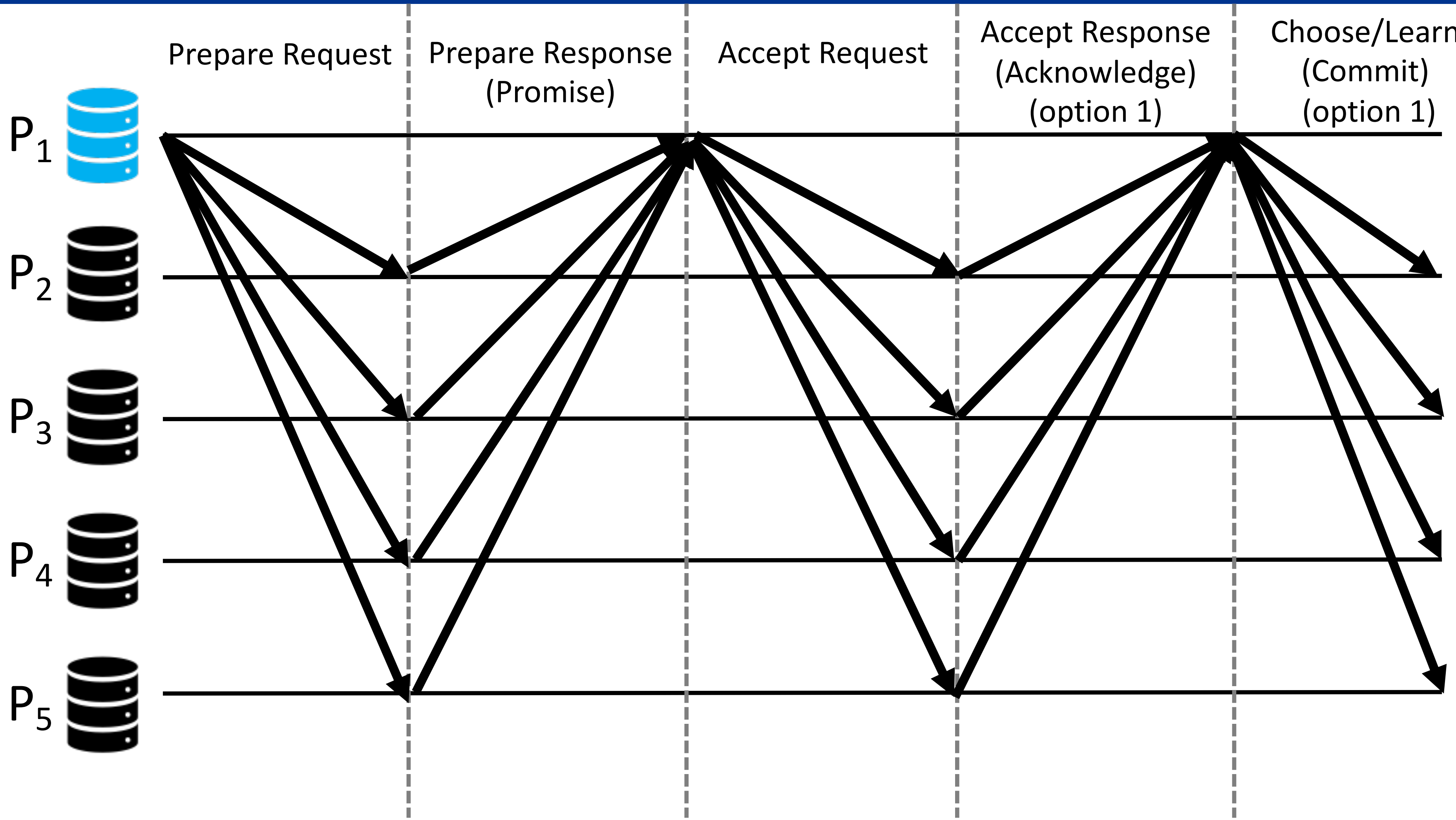
Proposer Actions

- **Send** prepare_request(n)
- **Wait** for majority prepare_response(n,v,m)
  - v : value of last accepted proposal (may be null if none)
  - m: sequence number of last accepted proposal (m < n)
- *If* received some prepare_response(n,v,m) with non-null v, **send** accept_request(n,x) where x is the value associated with highest received m; *else* **send** accept_request(n,x), where x can be any value

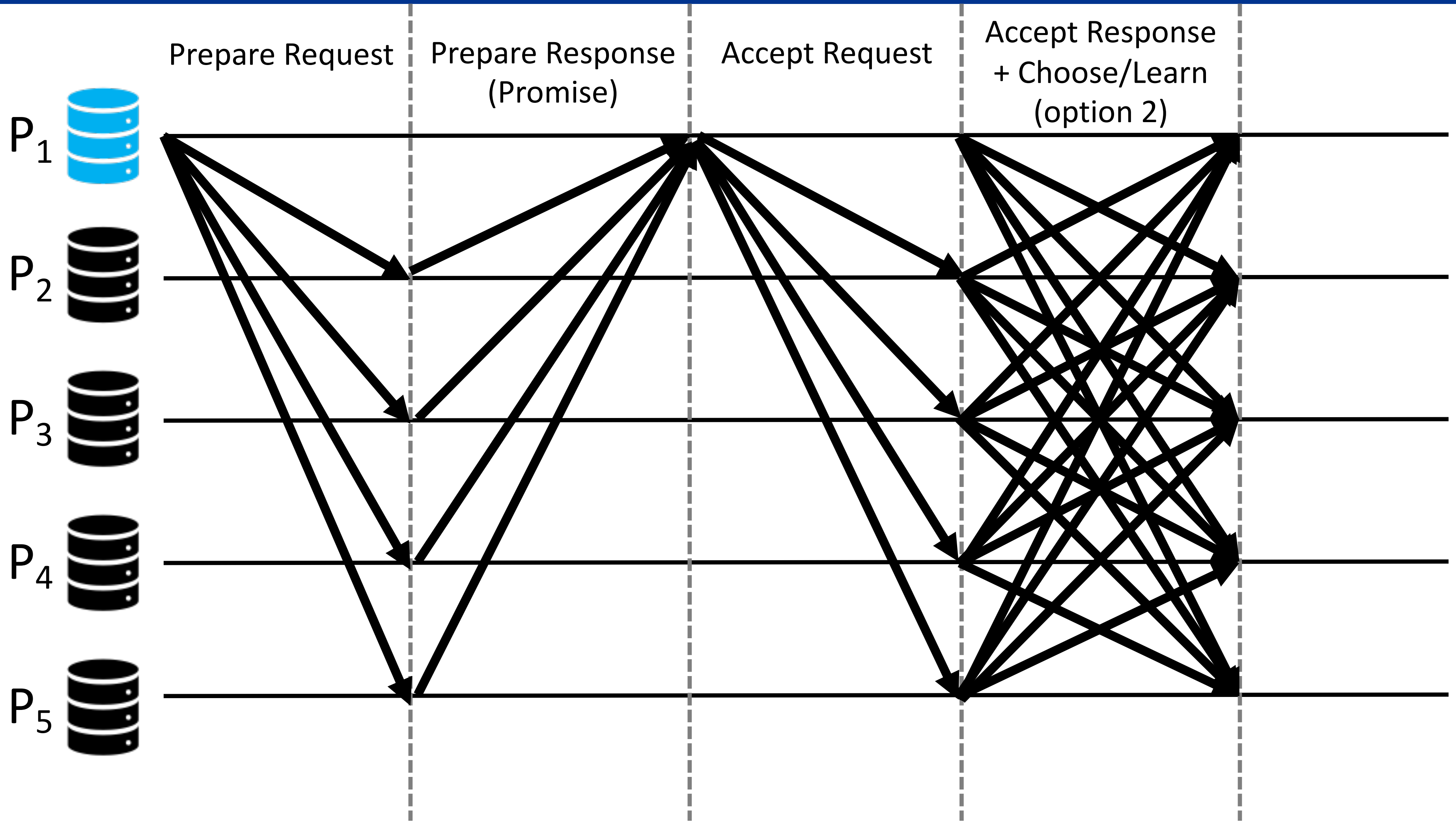
Acceptor Actions

- **Upon receiving** prepare_request(n)
  - *If* previously responded to prepare_request(m) s.t. $m > n$, **do nothing** (or “inform proposer” as performance optimization)
  - *Else if* previously accepted prepare_request(m) s.t. $m < n$, **send** prepare_response(n,v,m)
  - *Else* **send** prepare_response(n,null,0)
- **Upon receiving** accept_request(n,x)
  - *If* previously responded to prepare_request(m) s.t. $m > n$, do nothing (or “inform proposer” as performance optimization)
  - *Else* **accept** value x (and inform learners)

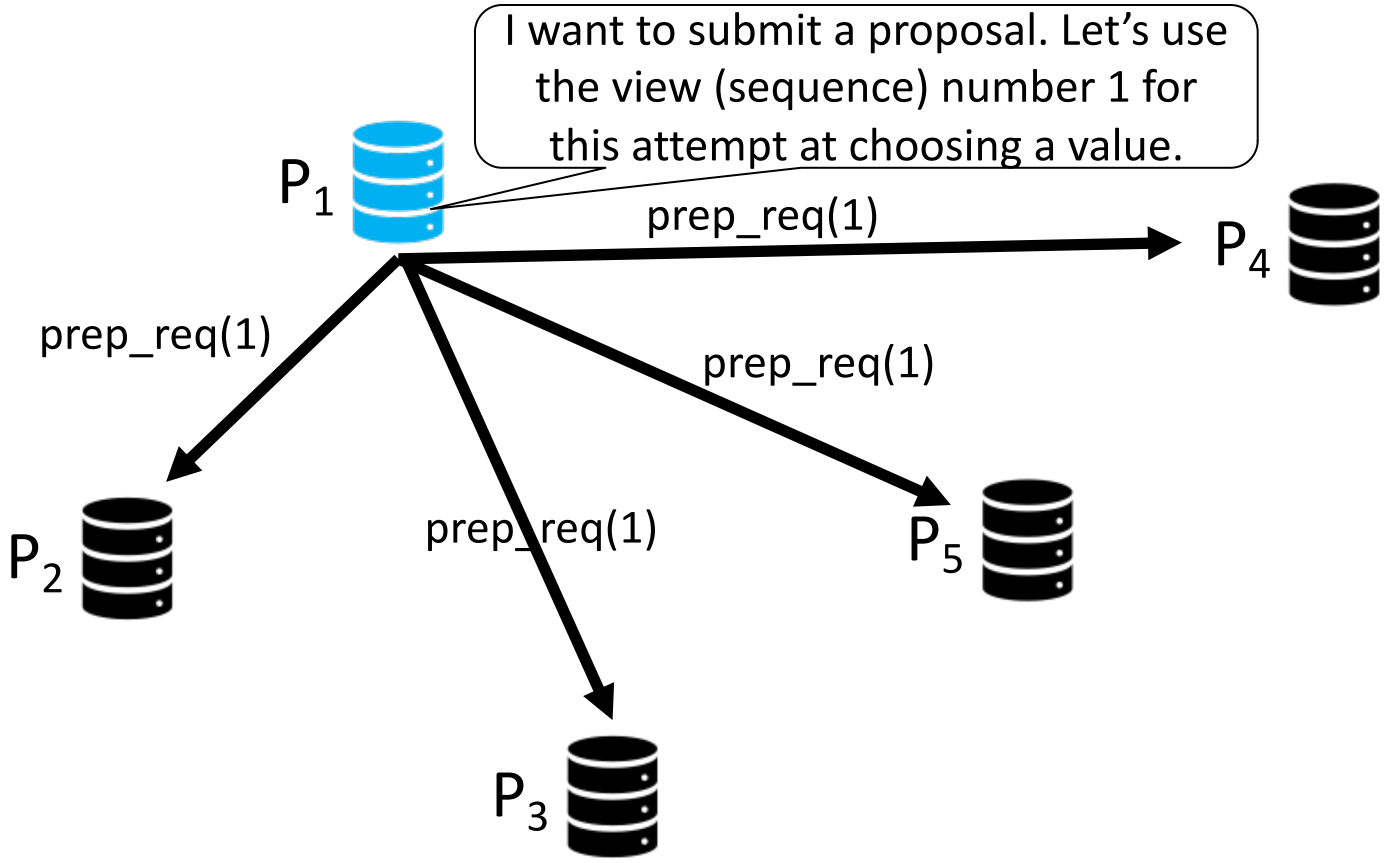
Paxos Consensus Protocol (“Single-Decree Synod”)



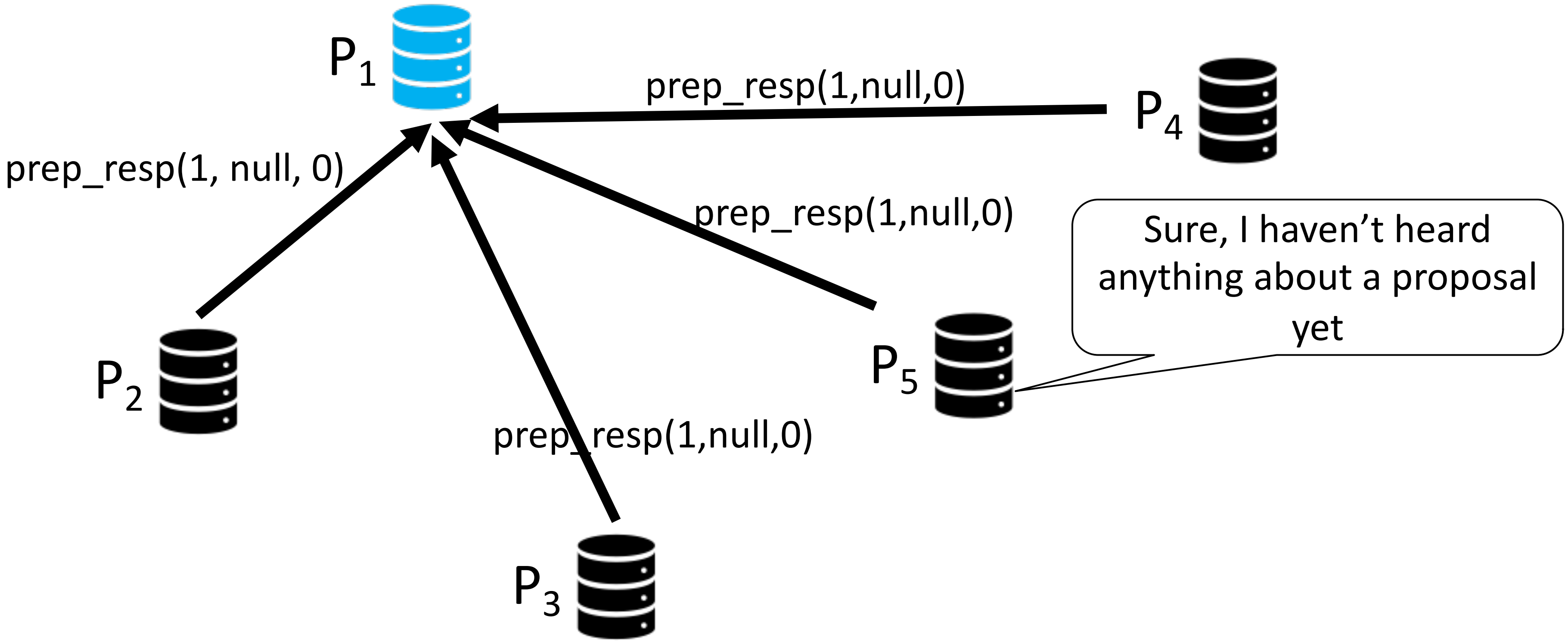
Paxos Consensus Protocol ("Single-Decree Synod")



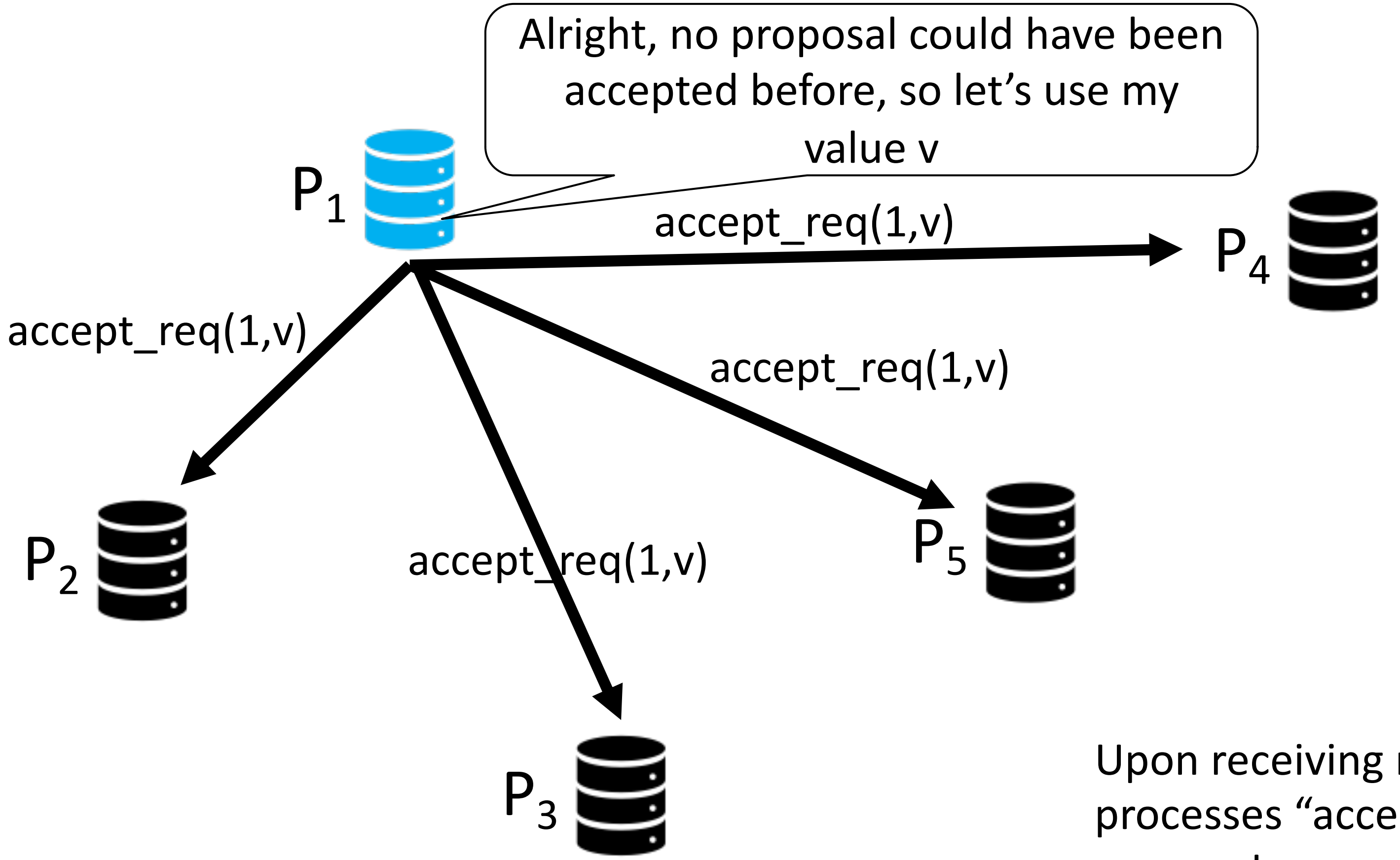
Normal Case: Prepare Request



Normal Case: Prepare Response

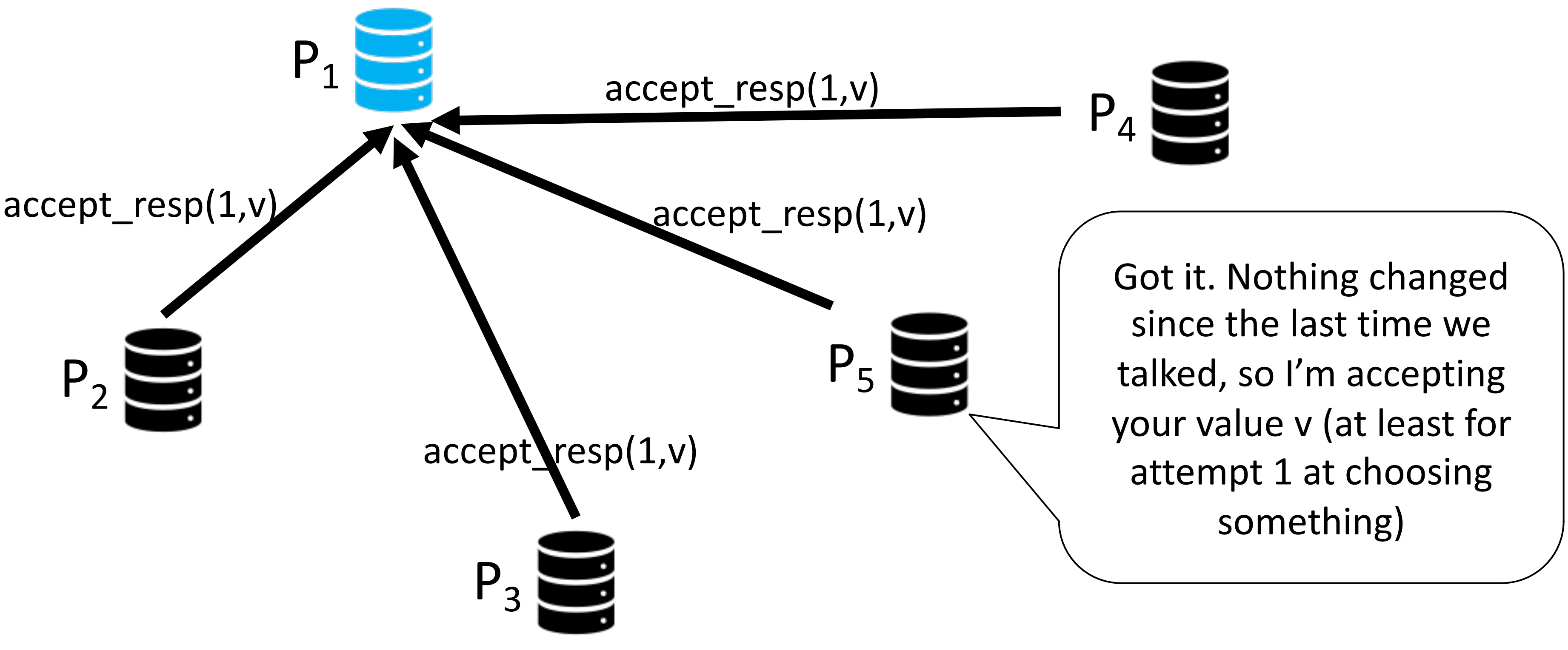


Normal Case: Accept Request

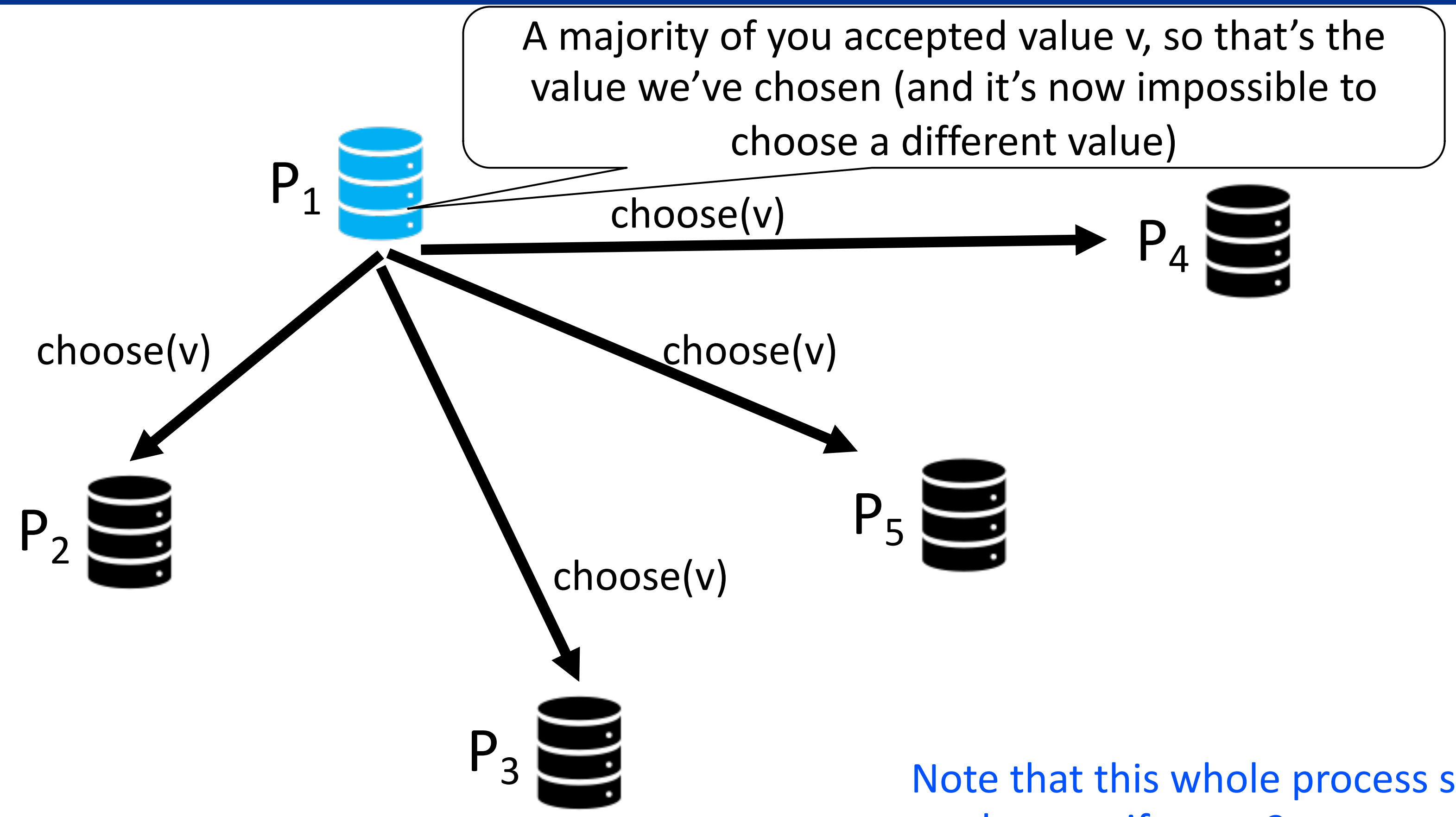


Upon receiving request, all processes "accept" the proposal

Normal Case (Option 1): Inform Distinguished Learner

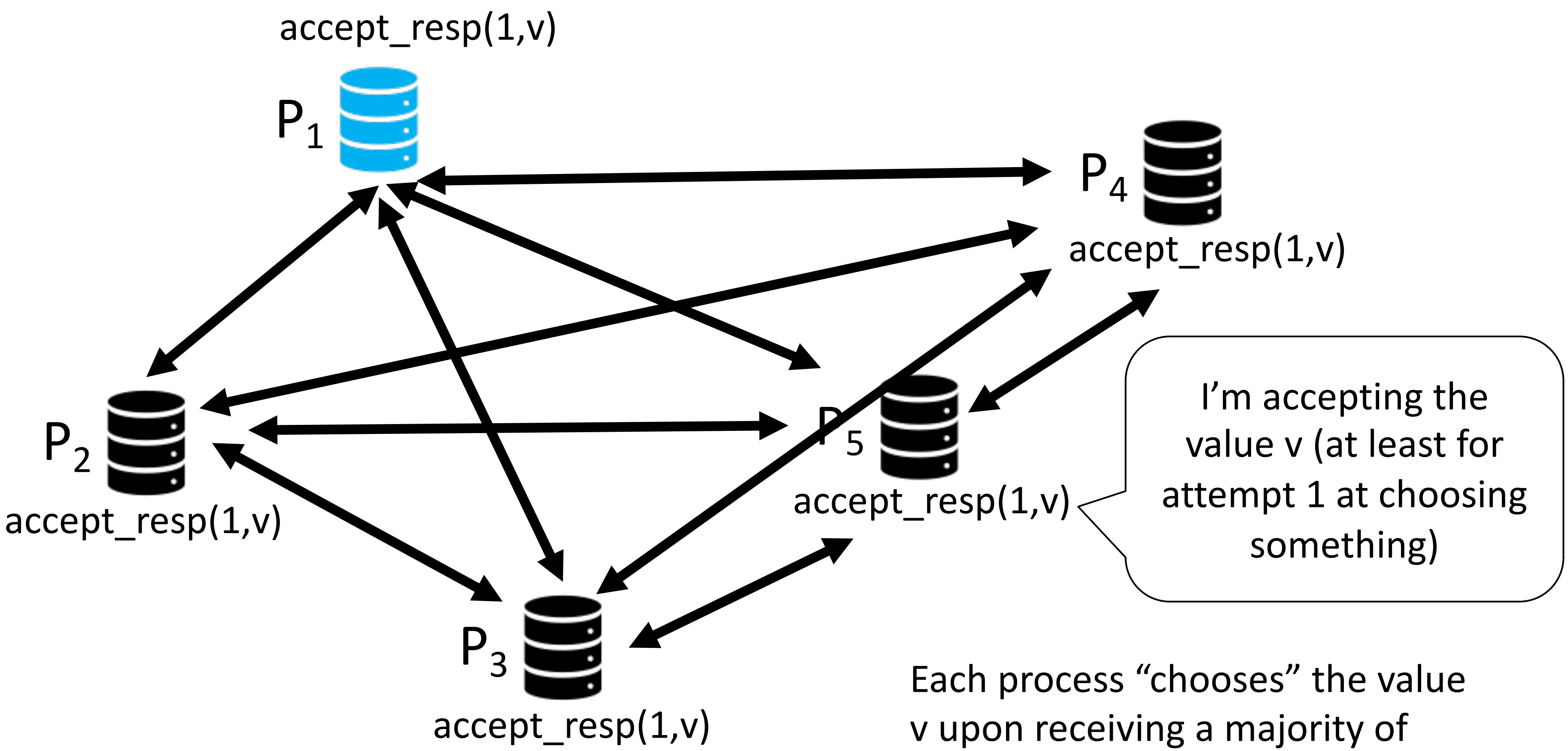


Normal Case (Option 1): Inform Distinguished Learner



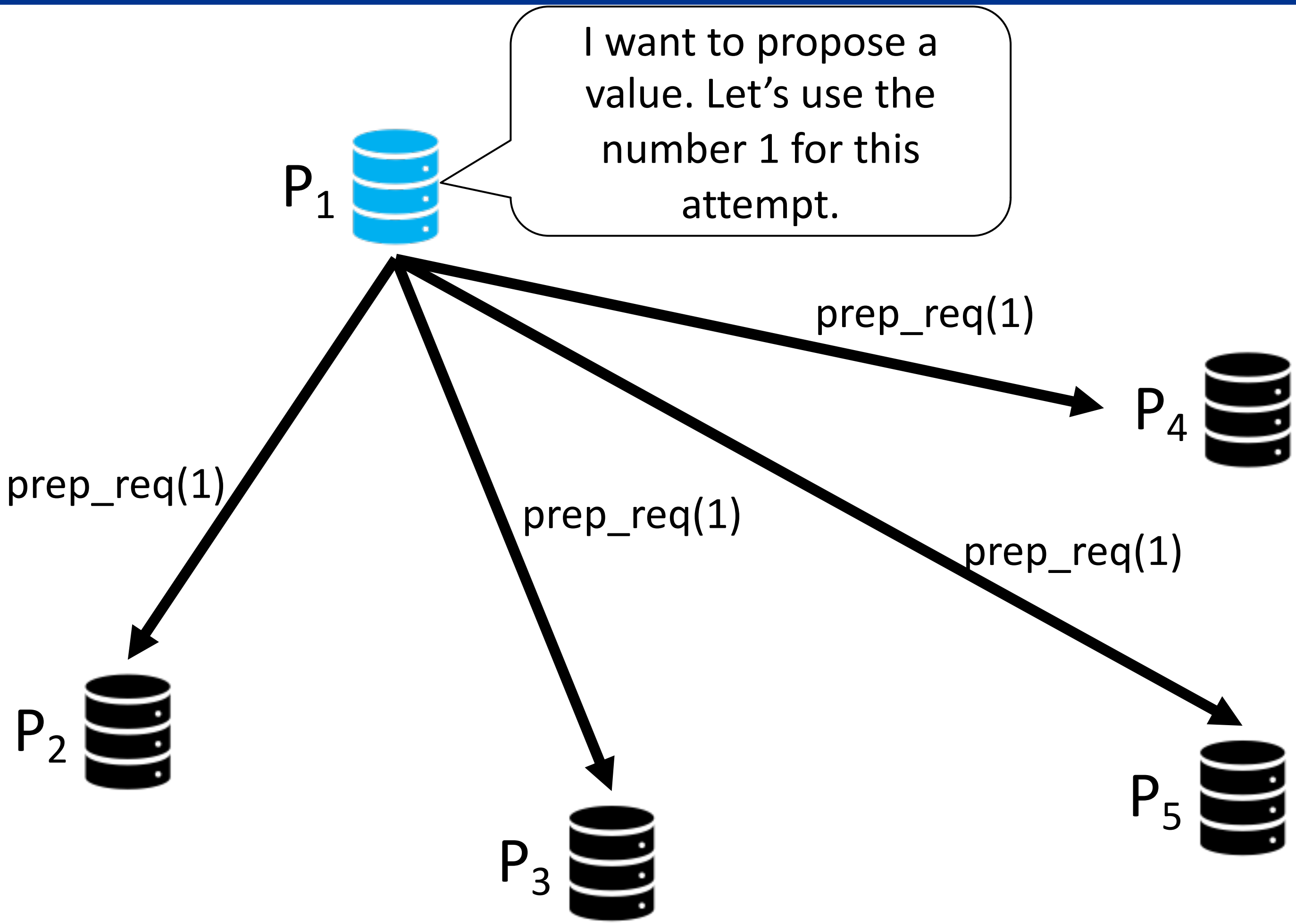
Note that this whole process still works even if up to 2 processes (other than P_1) fail at any point

Normal Case (Option 2): Inform all Learners

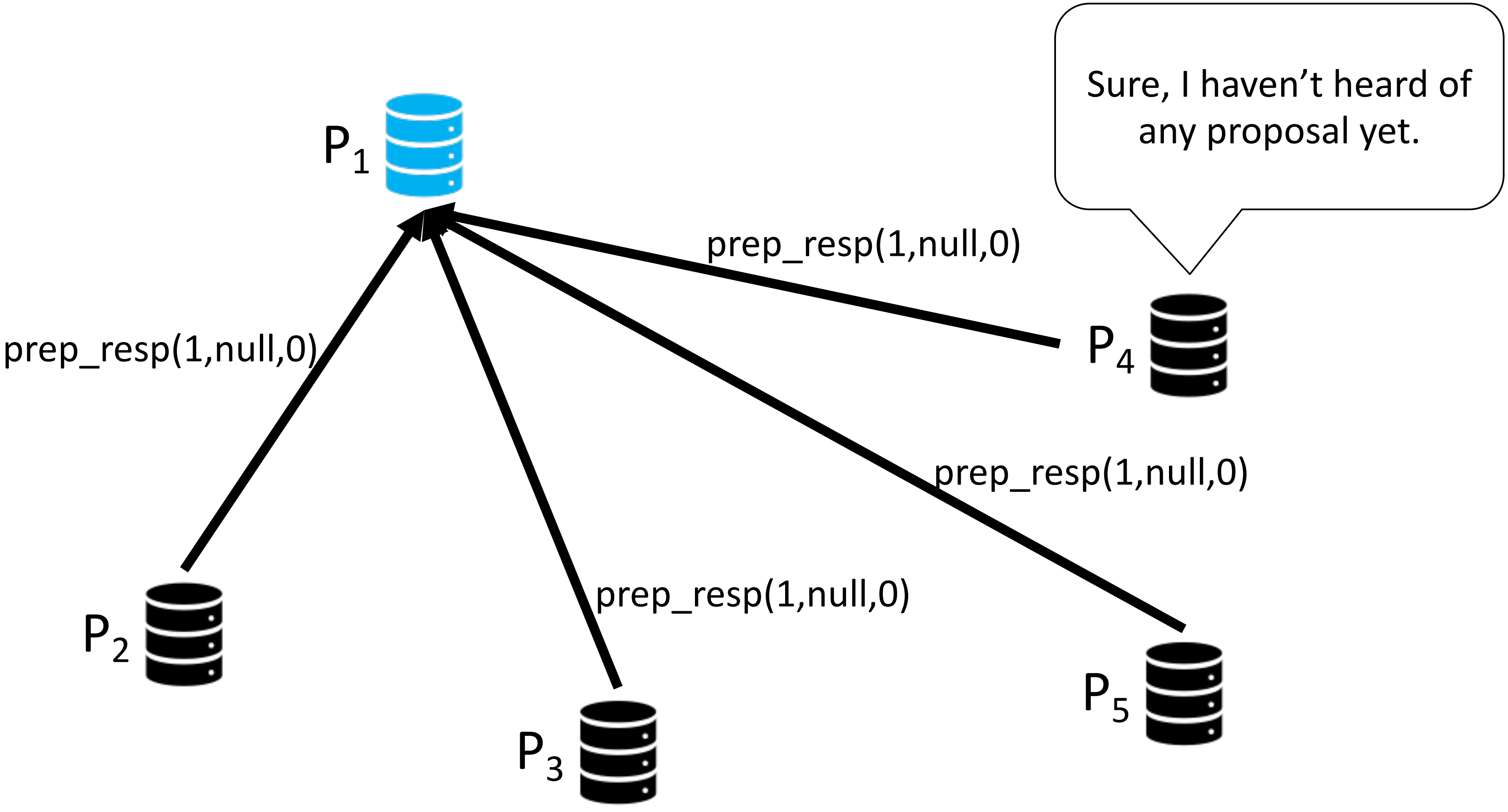


Each process "chooses" the value v upon receiving a majority of accept_resp(1,v)

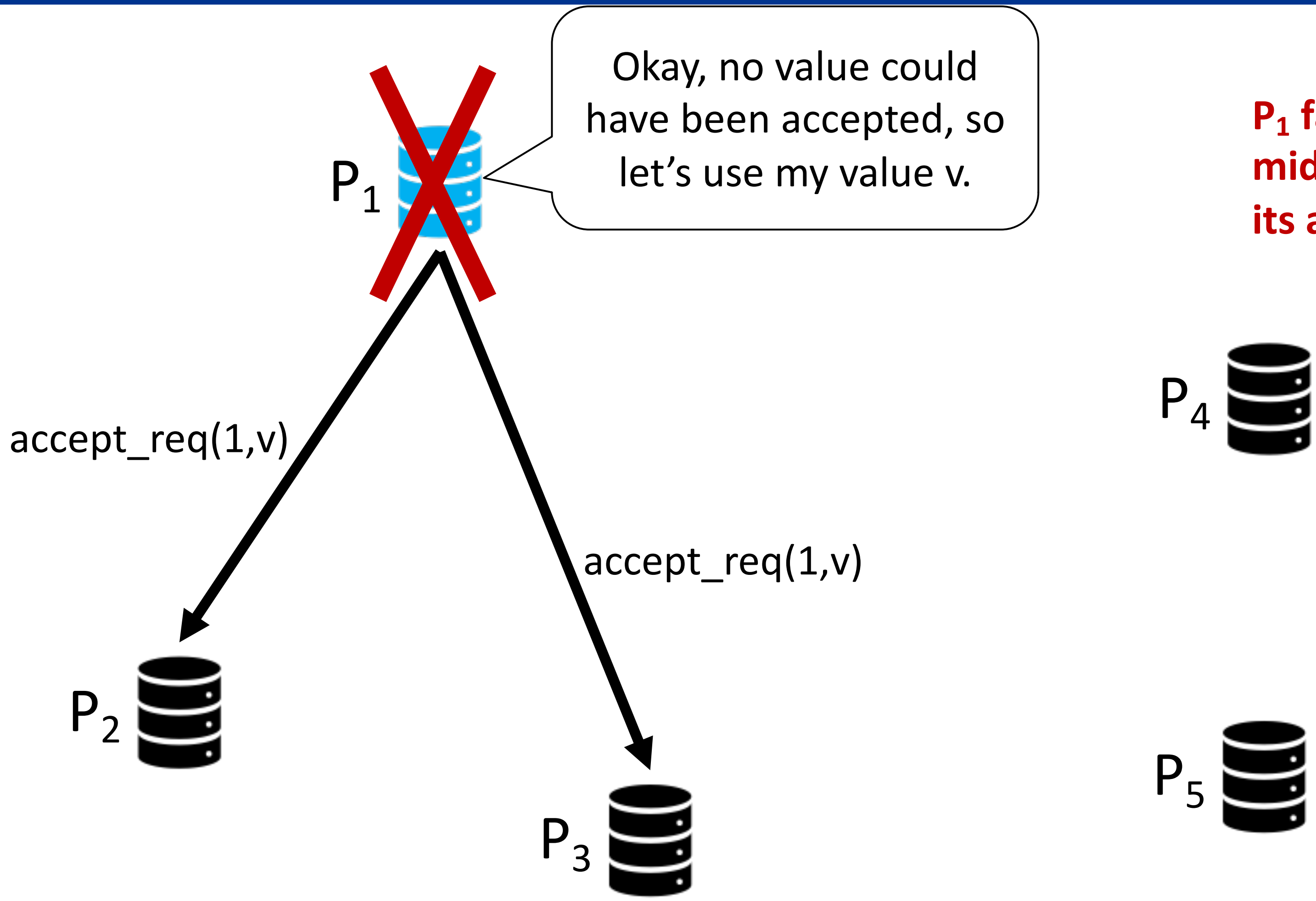
Leader Failure Case: Prepare Request



Leader Failure Case: Prepare Response

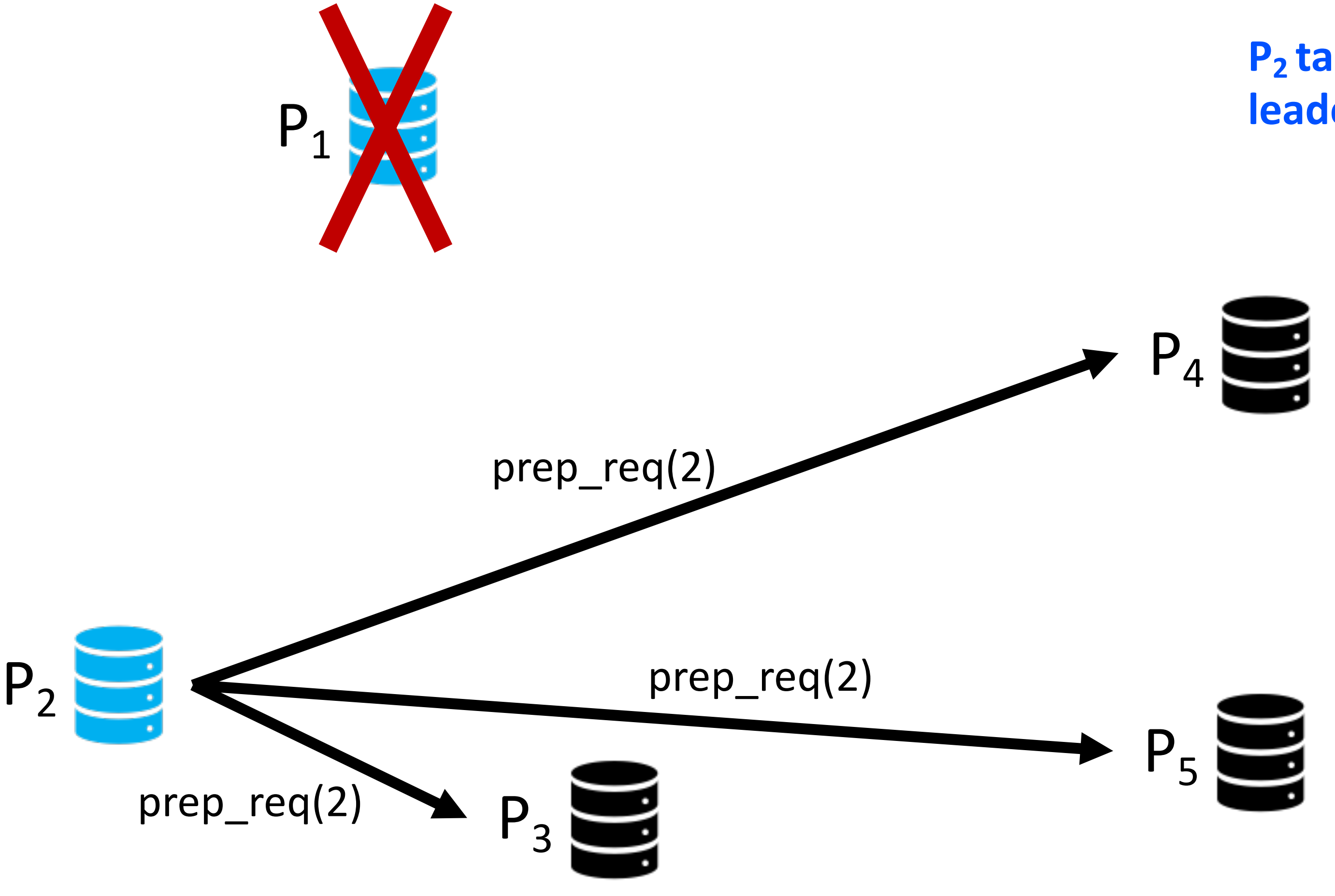


Leader Failure Case: Accept Request



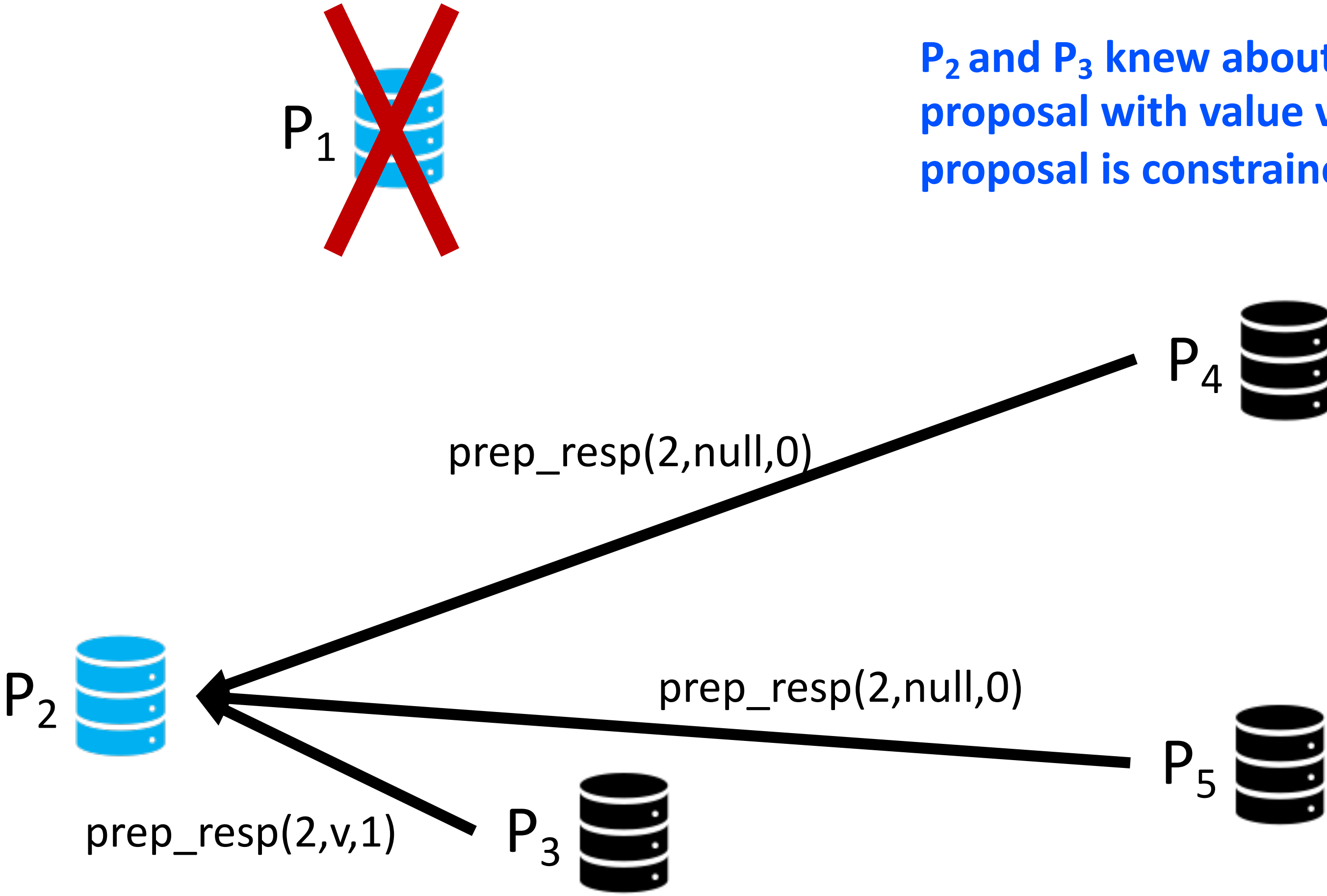
P_1 fails in the middle of sending its accept req!

Leader Failure Case: Prepare Request (Attempt 2)



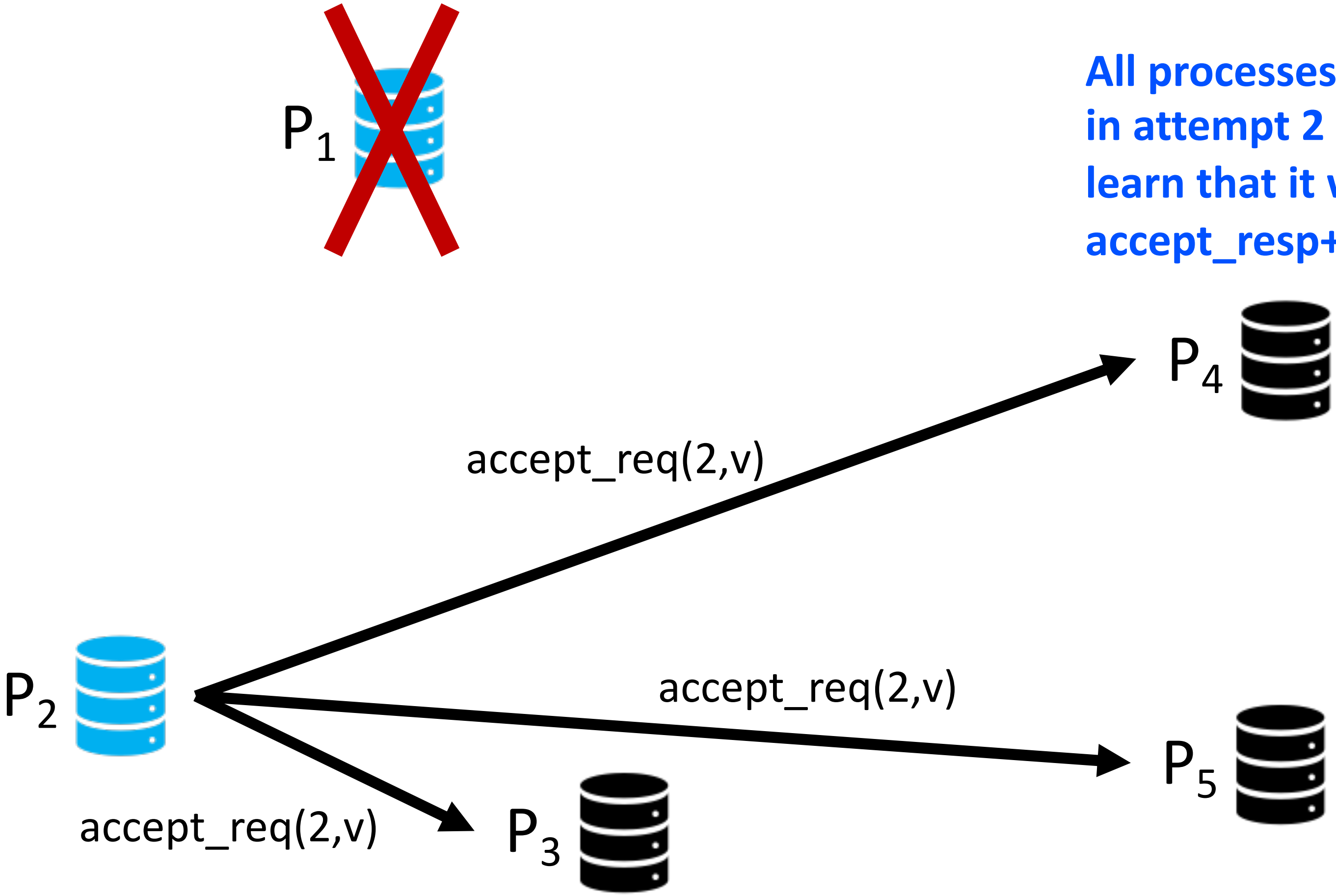
P_2 takes over as leader

Leader Failure Case: Prepare Response (Attempt 2)



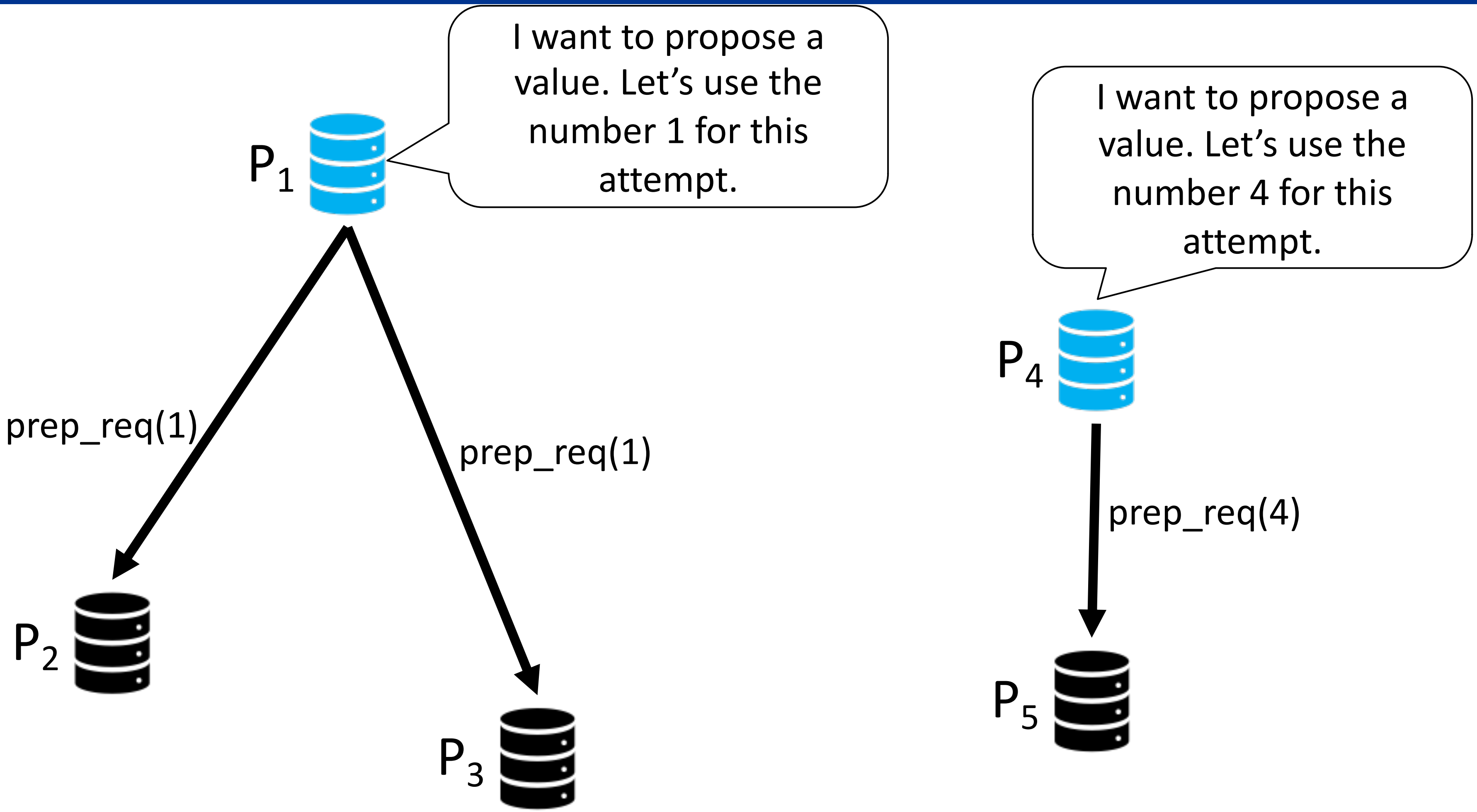
P_2 and P_3 knew about previous proposal with value v, so P_2's proposal is constrained to value v

Leader Failure Case: Accept Request (Attempt 2)

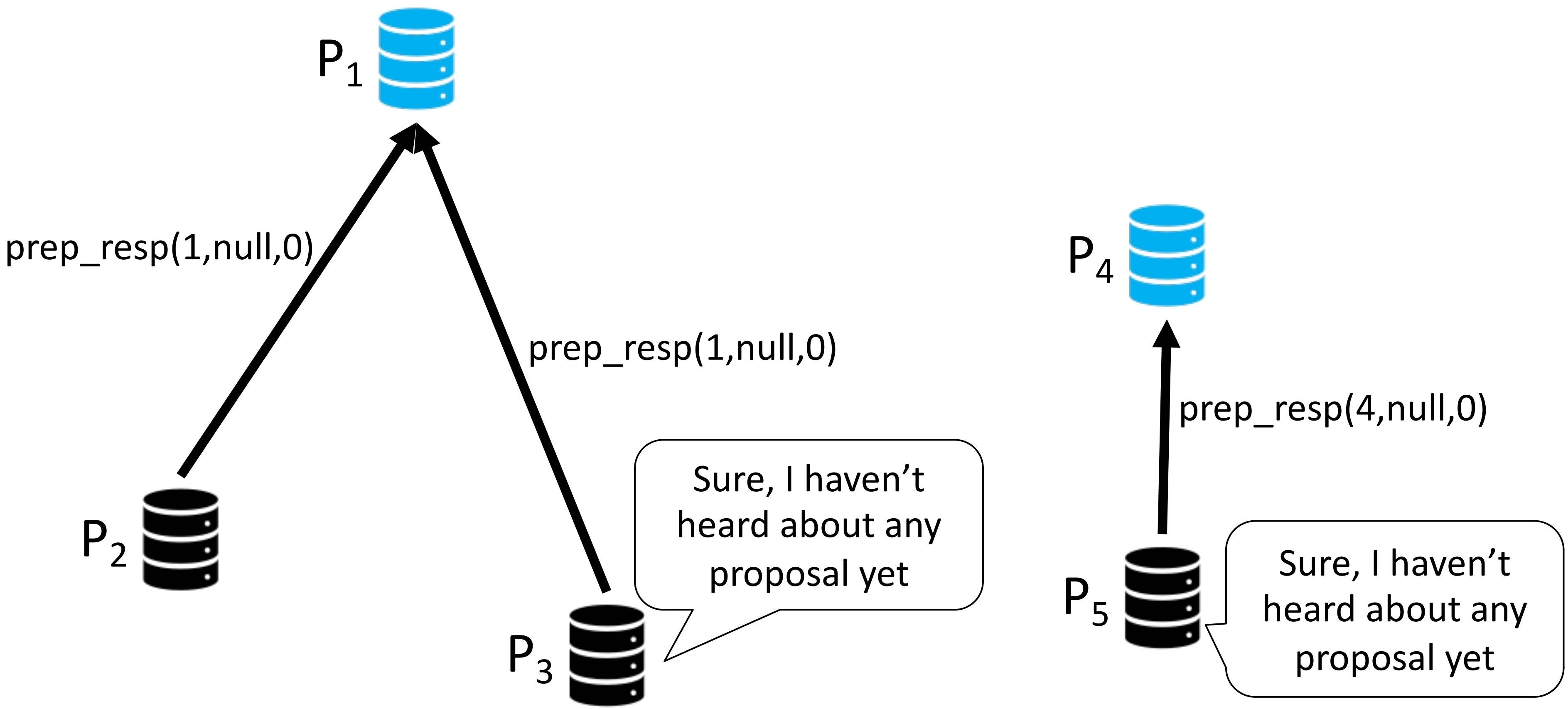


All processes accept value v in attempt 2 (and can then learn that it was chosen via accept_resp+choose)

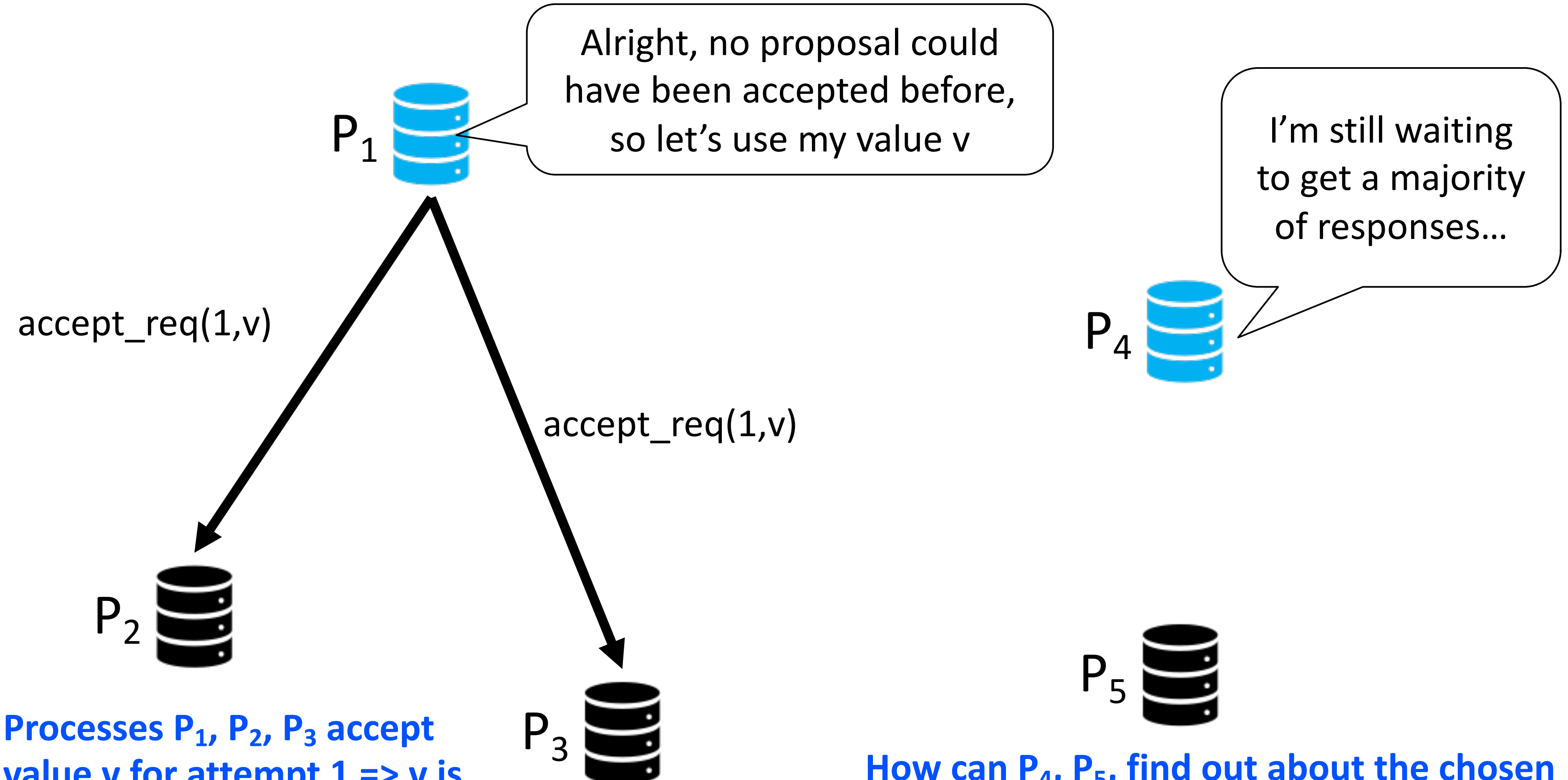
Simple Partition Case: Prepare Request



Simple Partition Case: Prepare Response



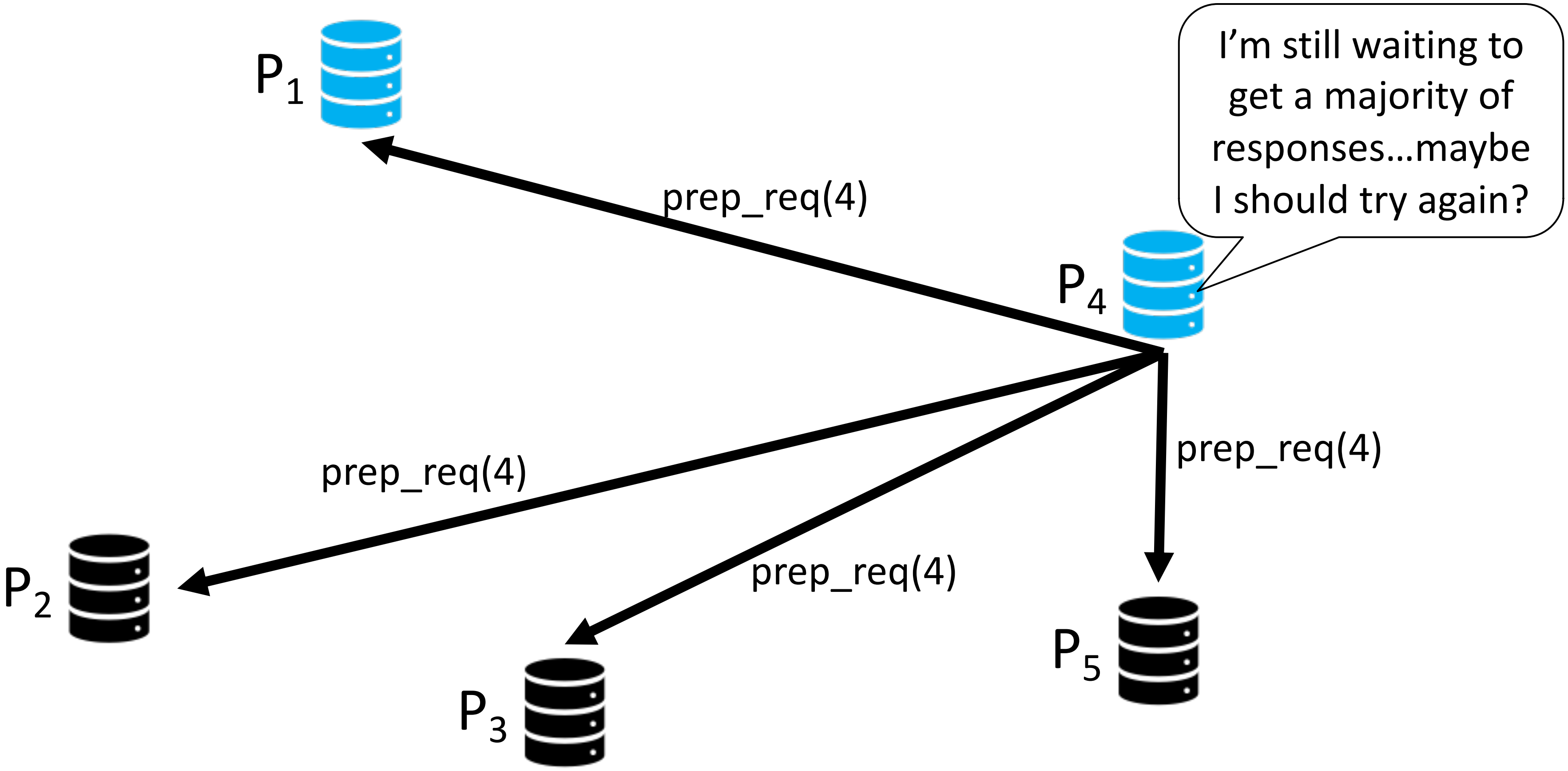
Simple Partition Case: Accept Request



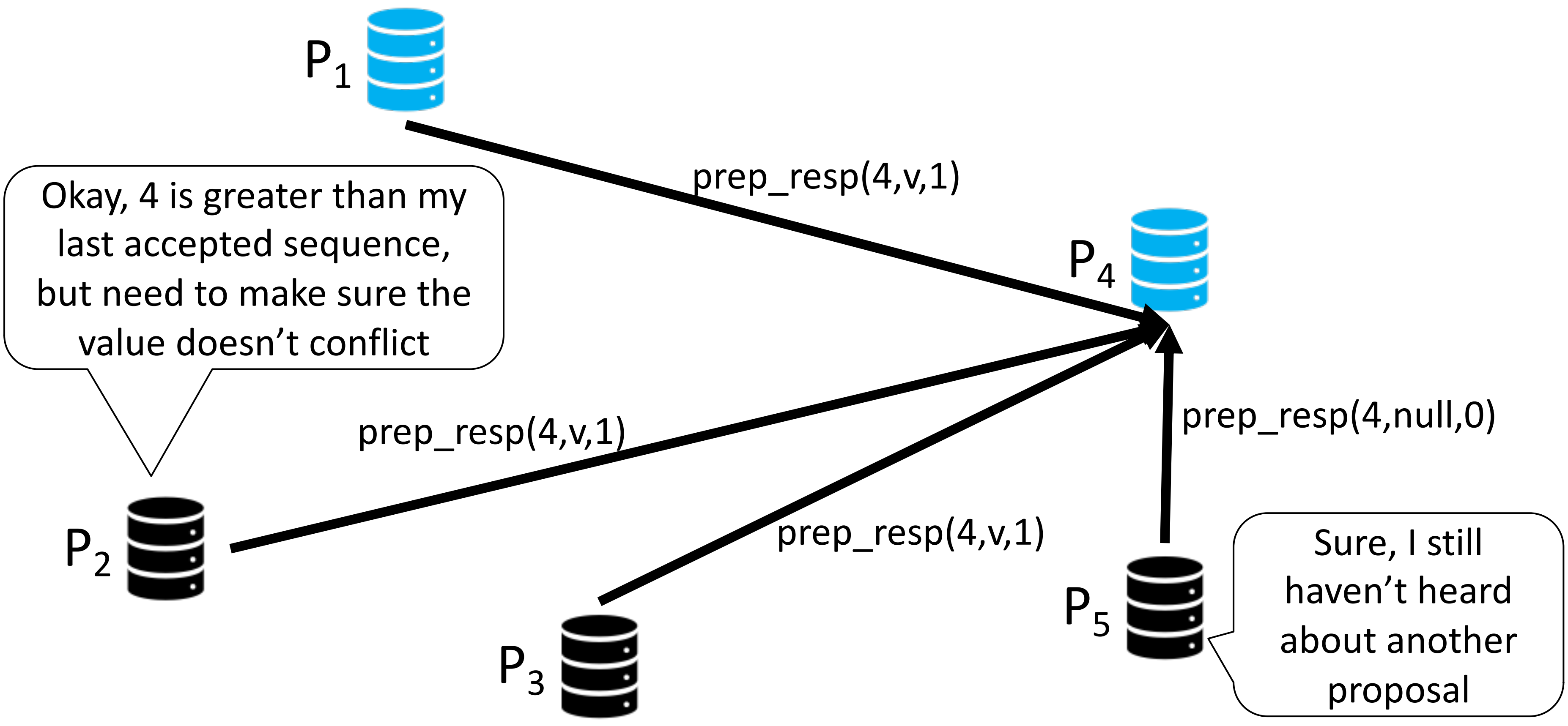
Processes P_1, P_2, P_3 accept value v for attempt 1 => v is chosen

How can P_4, P_5, find out about the chosen value (after the partition heals)?

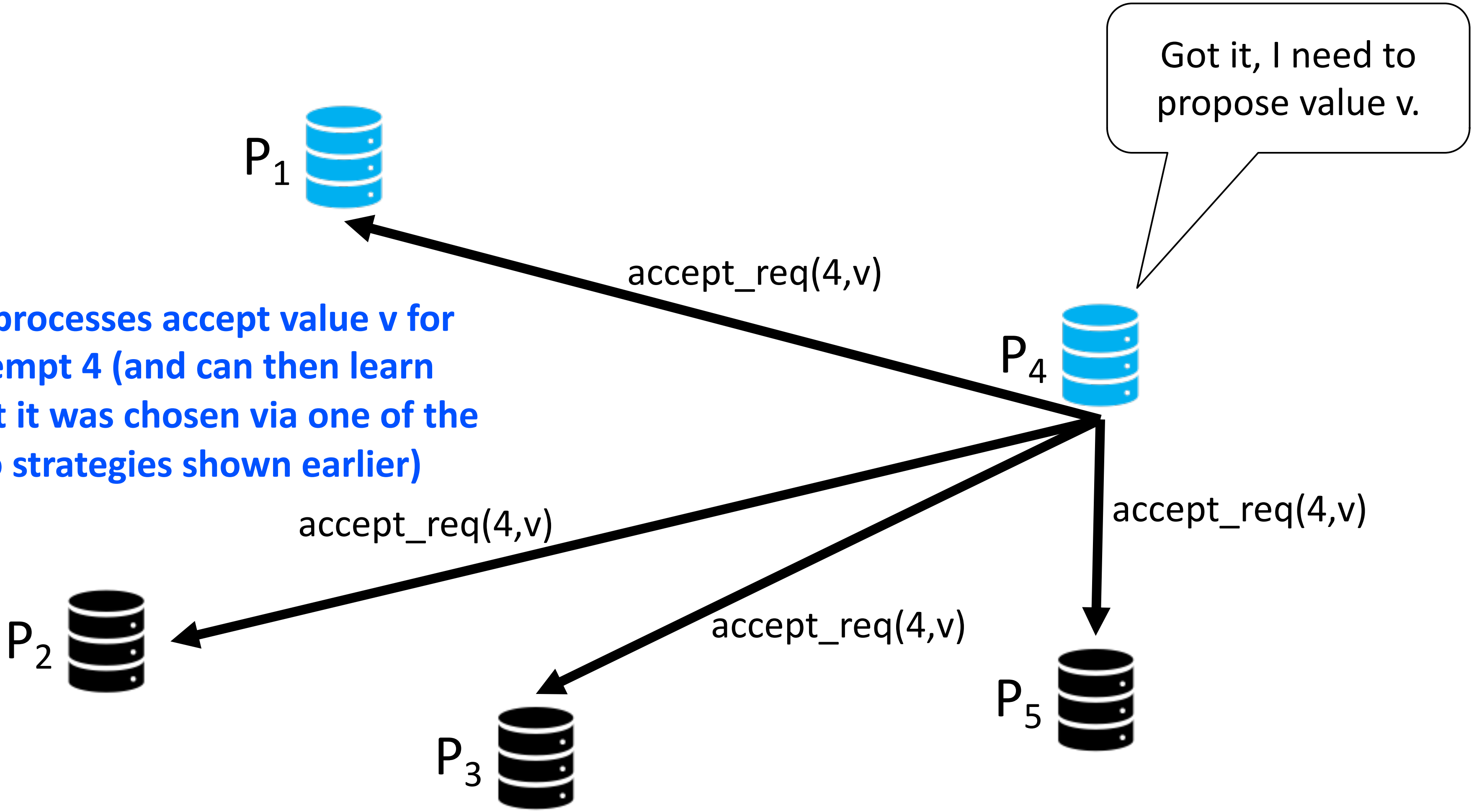
Simple Partition Case: Partition Repair (one option)



Simple Partition Case: Partition Repair (one option)



Simple Partition Case: Partition Repair (one option)



All processes accept value v for attempt 4 (and can then learn that it was chosen via one of the two strategies shown earlier)

From Consensus to SMR (Multi-Paxos)

- Execute a sequence of separate Paxos instances
- Value chosen in i^{th} consensus instance is the i^{th} command executed
- Optimization: stable leader only needs to execute “prepare” phase once

Proposer Actions (Prepare phase)

- **Send** prepare_request(n,i)
  - n : sequence/“view” number
  - i : ordinal of last (consecutive) consensus instance for which the proposer knows a *chosen* value
- **Wait** for majority prepare_response(n,{(j,v_j,m_j), (k,v_k,m_k), …})
  - (j,v_j,m) : (ordinal, value, view) for each ordinal j > i for which the acceptor has previously accepted a value v_j in some view m_j

Proposer Actions (Accept phase)

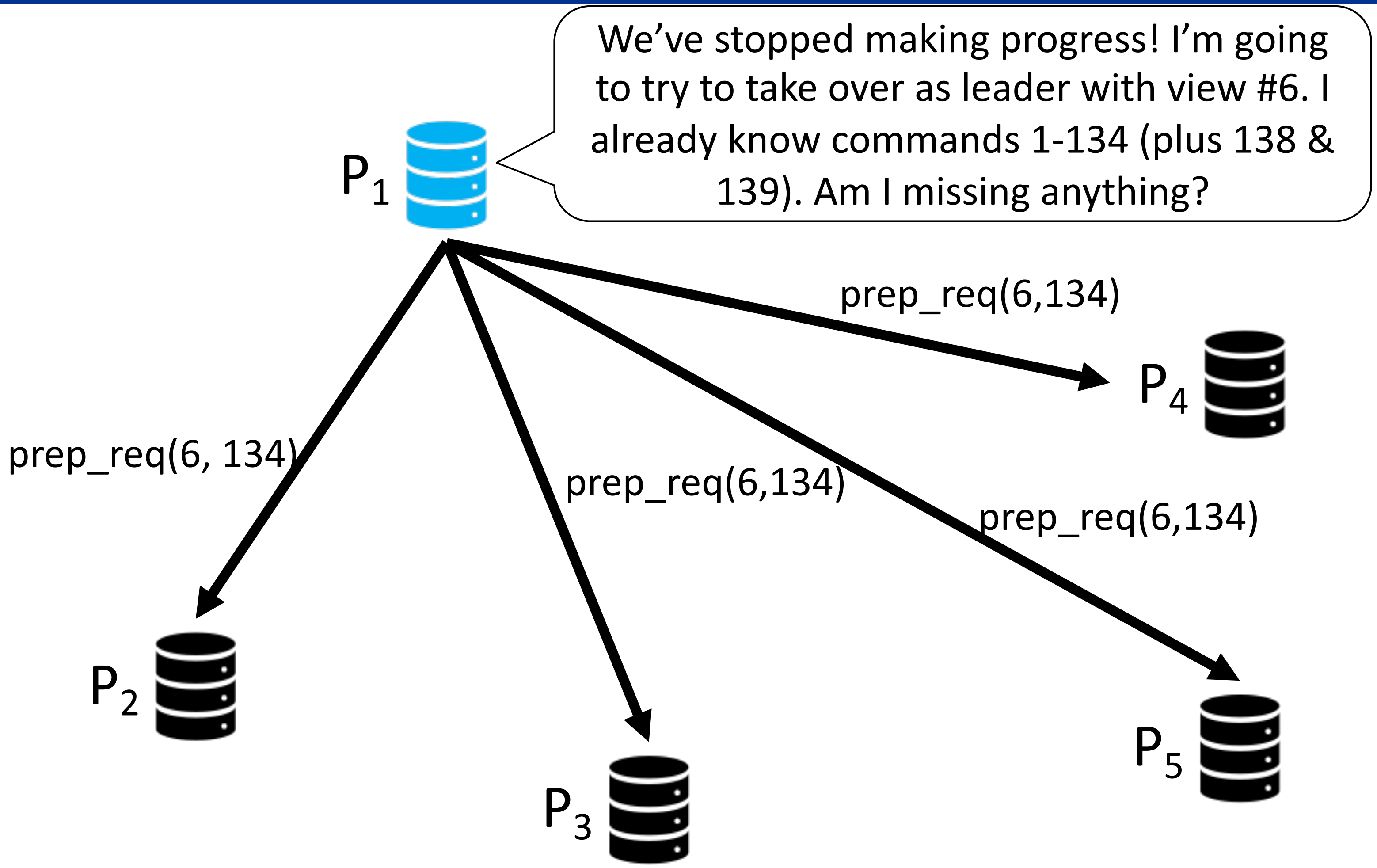
- **Propose all constrained updates:**
  - *For each* ordinal *j* s.t. some v_j was received, **send** accept_request(n,j,v_j) where v_j is the value associated with highest received m_j
- **Fill all holes (via no-ops):**
  - For each ordinal $j < j_{max}$ s.t. no v_j was received, send accept_request(n,j,no-op)
- **Proceed with unconstrained instances**
  - Incoming client requests can be assigned ordinals starting with $j_{max}+1$

Acceptor Actions

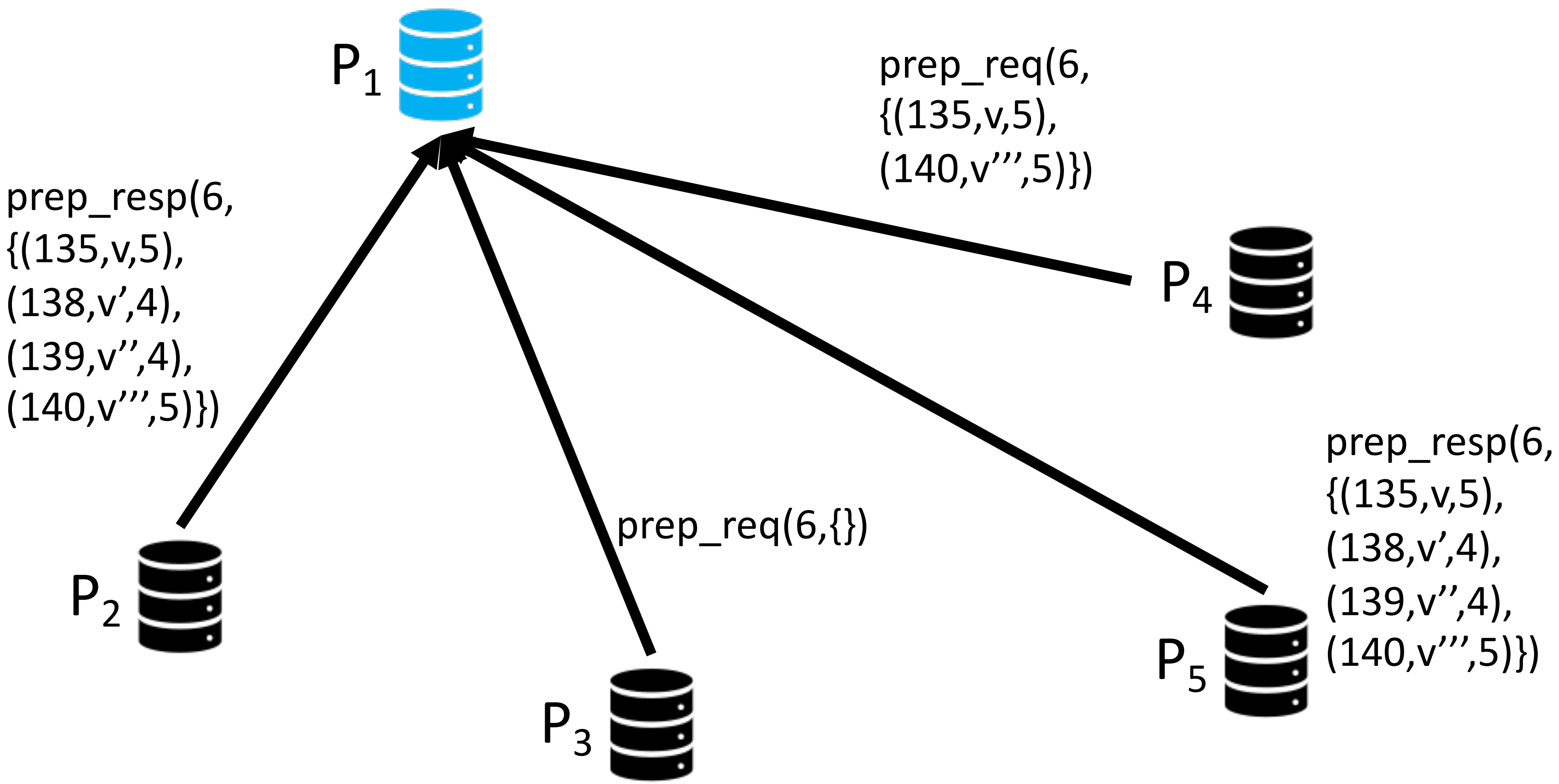
- **Upon receiving** prepare_request(n,i)
  - *If* previously responded to prepare_request(m,*) s.t. m > n, do nothing (or "inform proposer" as performance optimization)
  - *Else* construct constraining update list and send prepare_response(n,{(j,v_j,m_j), (k,v_k,m_k), …})

- **Upon receiving** accept_request(n,j,v_j)
  - *If* previously responded to prepare_request(m,*) s.t. m > n, do nothing (or "inform proposer" as performance optimization)
  - *Else* **accept** value v_j for ordinal j (and inform learners)

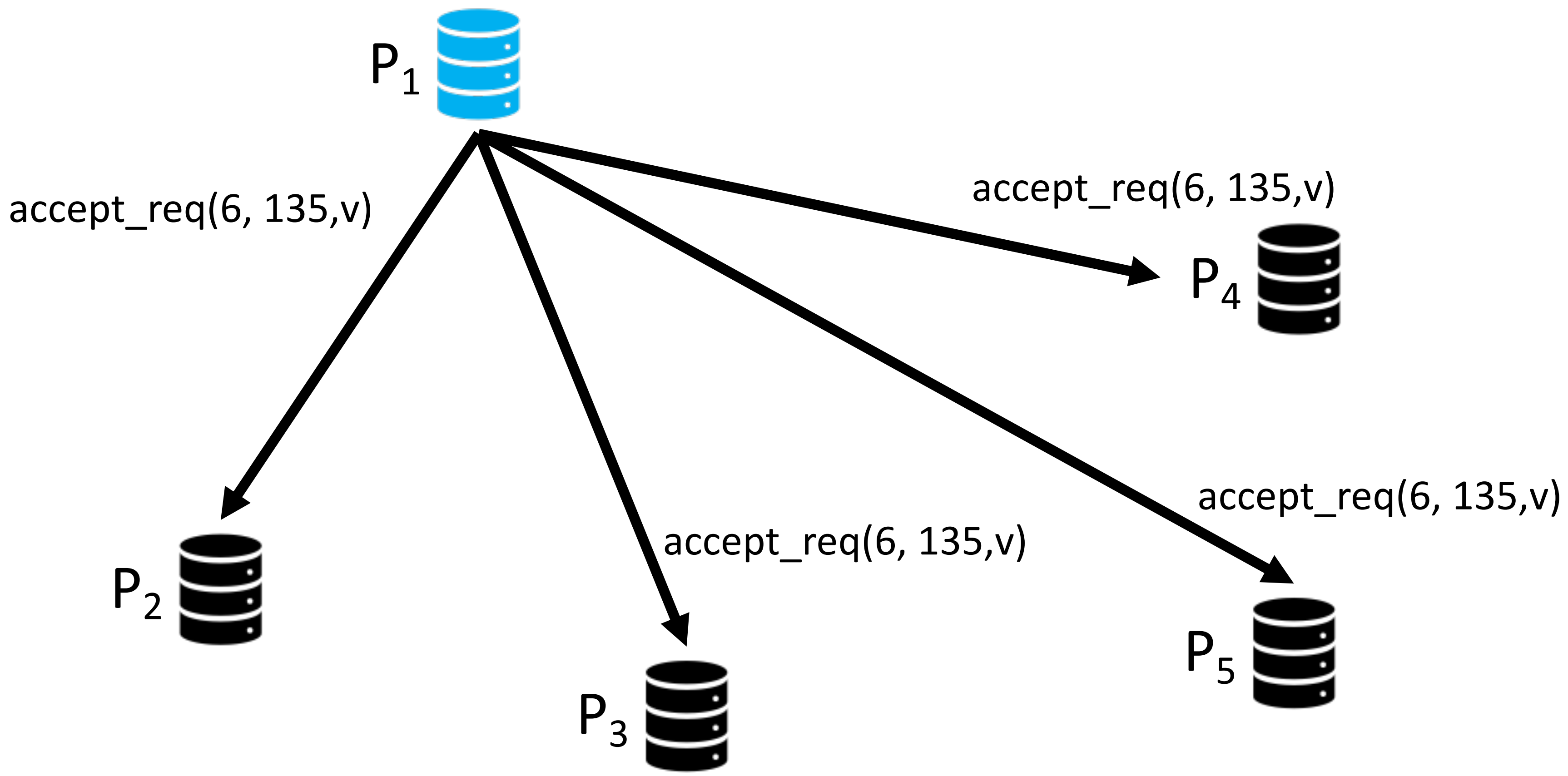
Multi-Paxos Example (from paper)



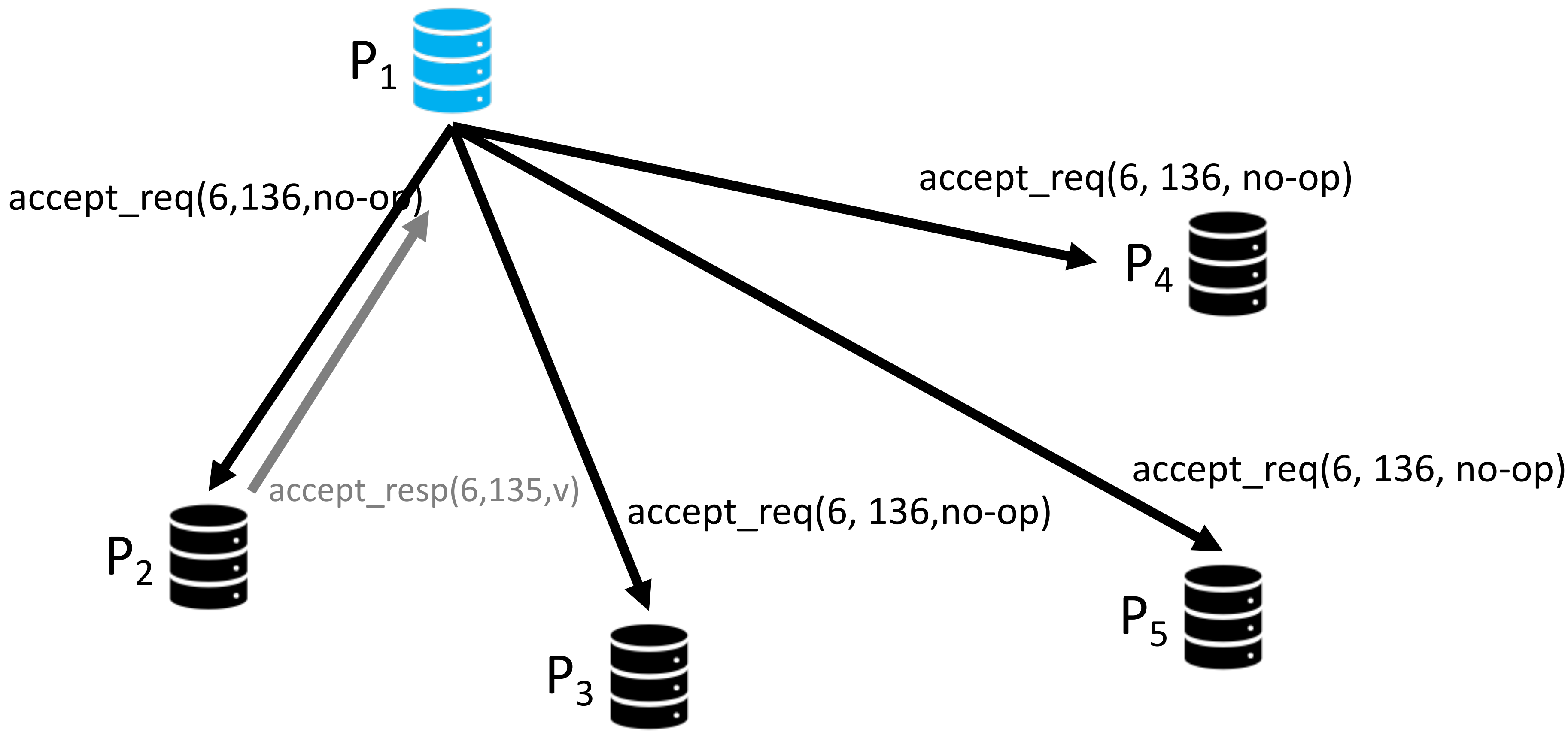
Multi-Paxos Example (from paper)



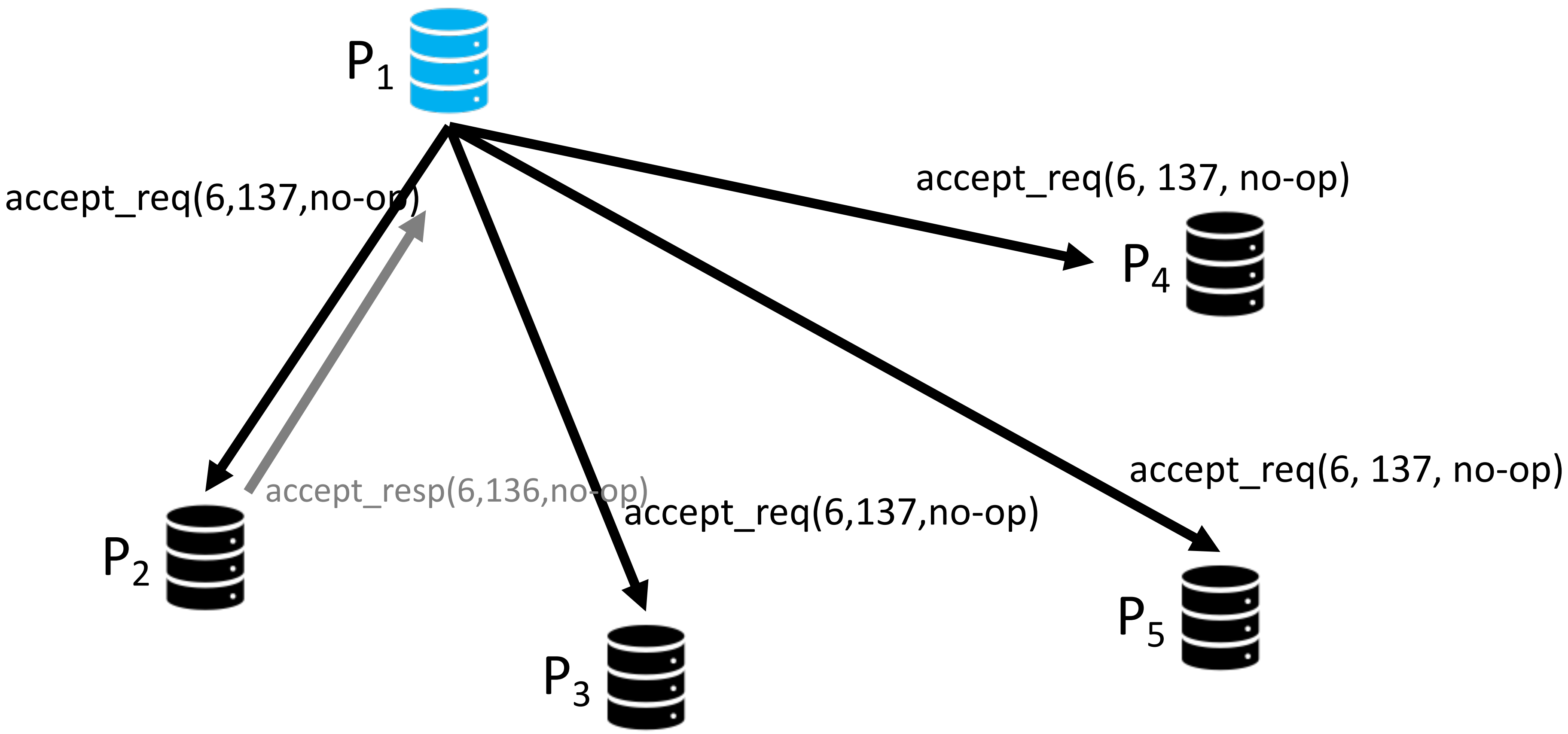
Multi-Paxos Example (from paper)



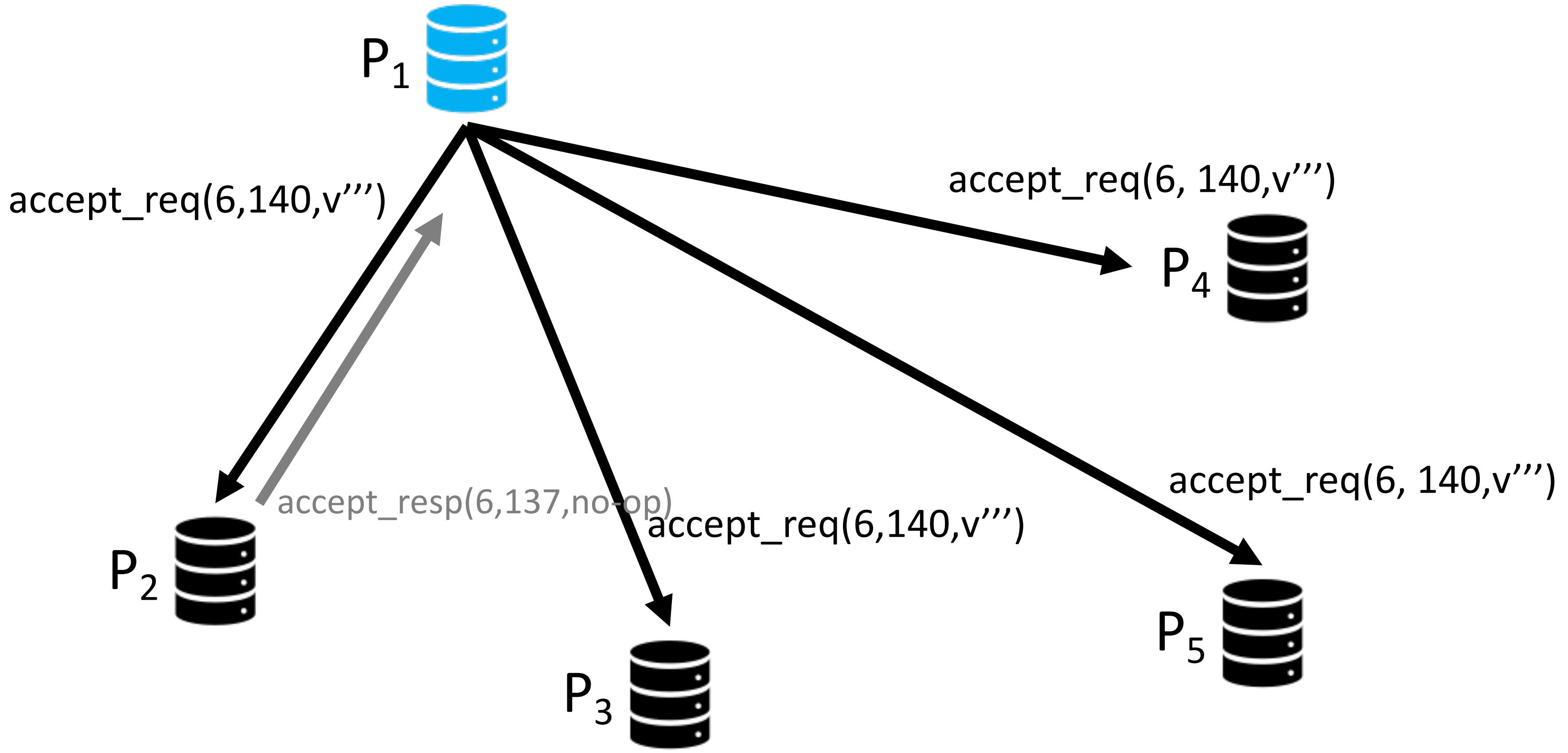
Multi-Paxos Example (from paper)



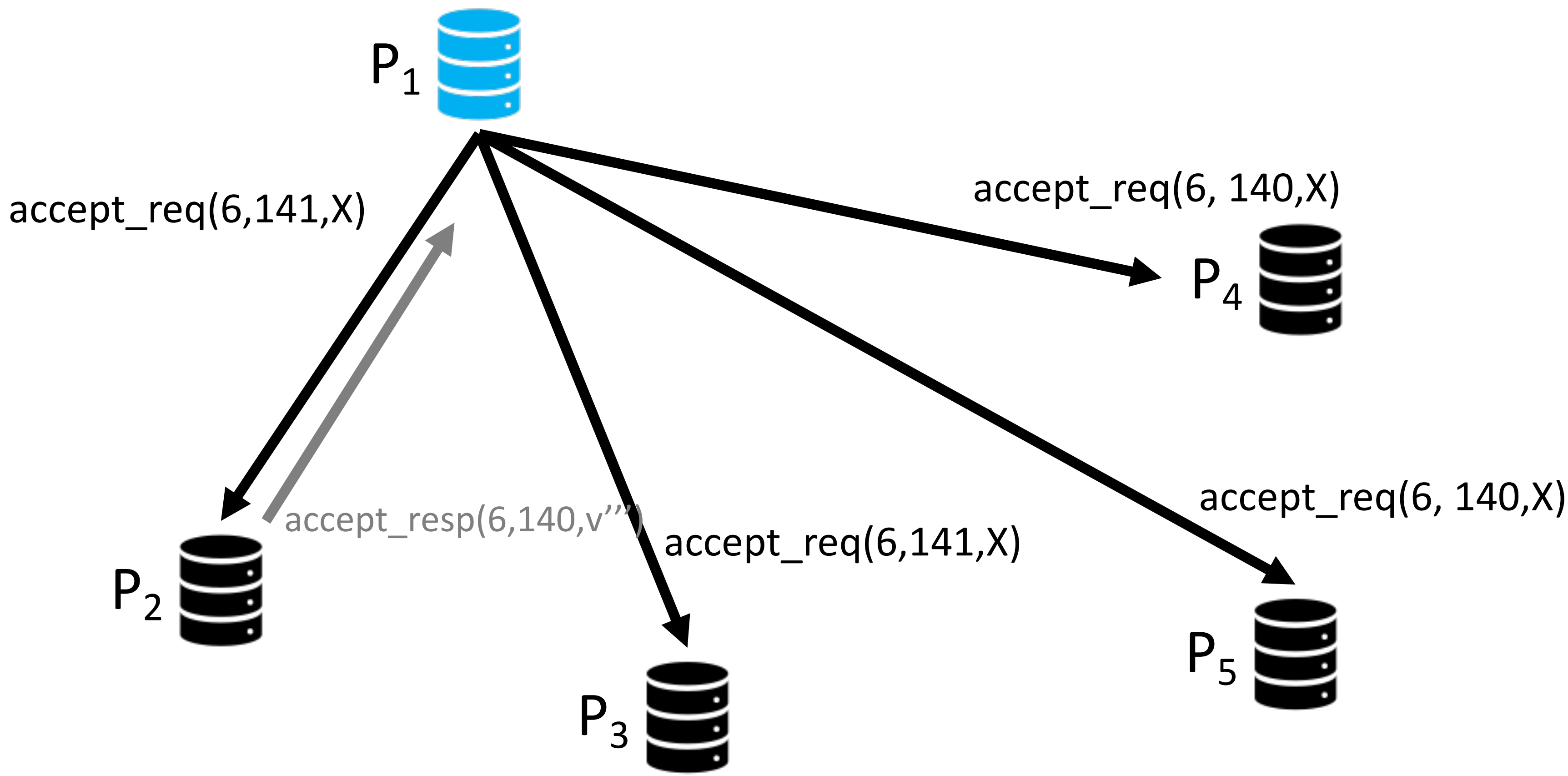
Multi-Paxos Example (from paper)



Multi-Paxos Example (from paper)



Multi-Paxos Example (from paper)



Summary

- Introduced the Paxos consensus algorithm (simplified from original specification)
- Key idea behind preventing inconsistency is simple and intuitive
- Implementation and ensuring liveness (especially ensuring that replicas can actually **execute** ordered events) requires non-trivial extensions
  - Often dismissed as "engineering details"