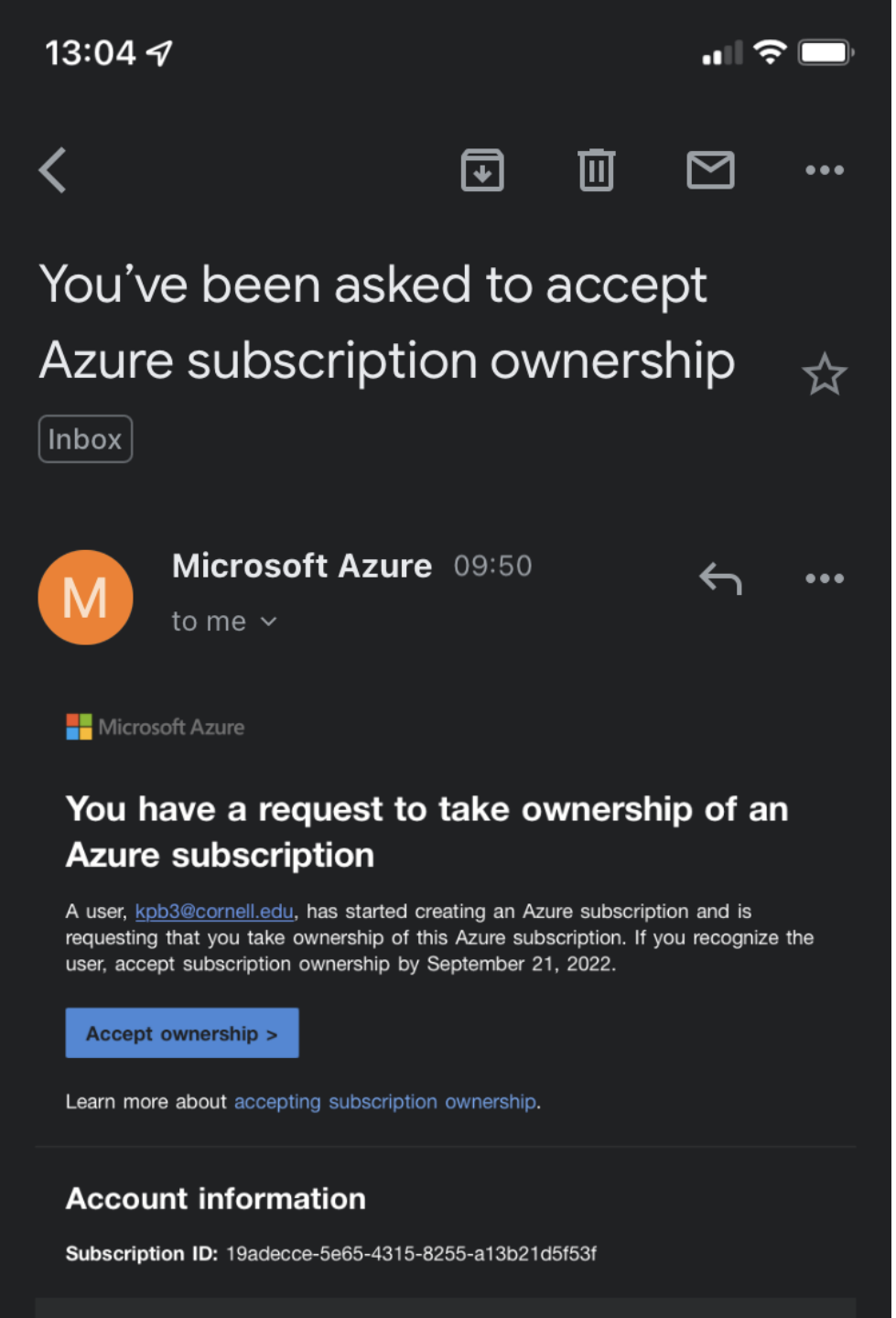
Paxos Made …

Yifan Wang



Logistics

- OHs are all set.
- First batch of Azure credits are sent out.

Logistics

Project groups.

Due date is this Friday.

In case you are still in a group of 1, please contact staff.

Project ideas.

If you plan to use something involving raspberry pi, we have a unique opportunity for you.

Logistics

Project groups.

Due date is this Friday.

In case you are still in a group of 1, please contact staff.

Project ideas.

If you plan to use something involving raspberry pi, we have a unique opportunity for you.

Logistics

- Prelims: 2 Prelims, Oct. 5, Nov. 16, 7:30 – 9:30 PM

https://edstem.org/us/courses/26354/discussion/1739988

- Project idea from Siemens:

If you plan to use something involving raspberry pi, I have a unique opportunity for you.

- Project Team Formation:

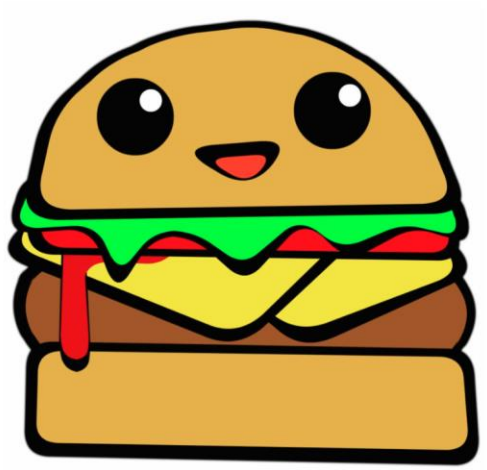
If you need help with finding team members, contact us.

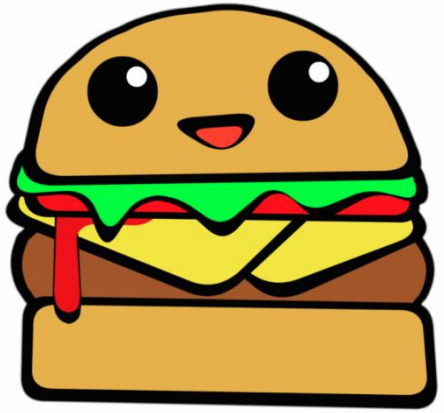
Paxos made …

- At the very beginning, 1989
- Then, in 1998, Leslie wrote, The part-time parliament
- Then, in 2001, Leslie wrote, Paxos made simple (11 pages)
- Then, RVR wrote, Paxos Made Moderately Complex (36 pages) https://www.cs.cornell.edu/courses/cs7412/2011sp/paxos.pdf

Distributed Consensus

Burgers or Pizza?





or

?

- Band director went home, people easily distracted.
- Yelling fails. Use person-to-person communication
- No fun if group splits up, everyone needs to end up with the same decision
- Hungry: must come to a decision

Consensus

- Safety:
  - Validity
    - Only a single value may be chosen
    - Only a value that has been proposed may be chosen
  - Integrity: A server never learns that a value has been chosen unless it really has
- Lively:
  - Termination: Some proposed value is eventually chosen
  - Agreement: If a value is chosen, servers eventually learn about it

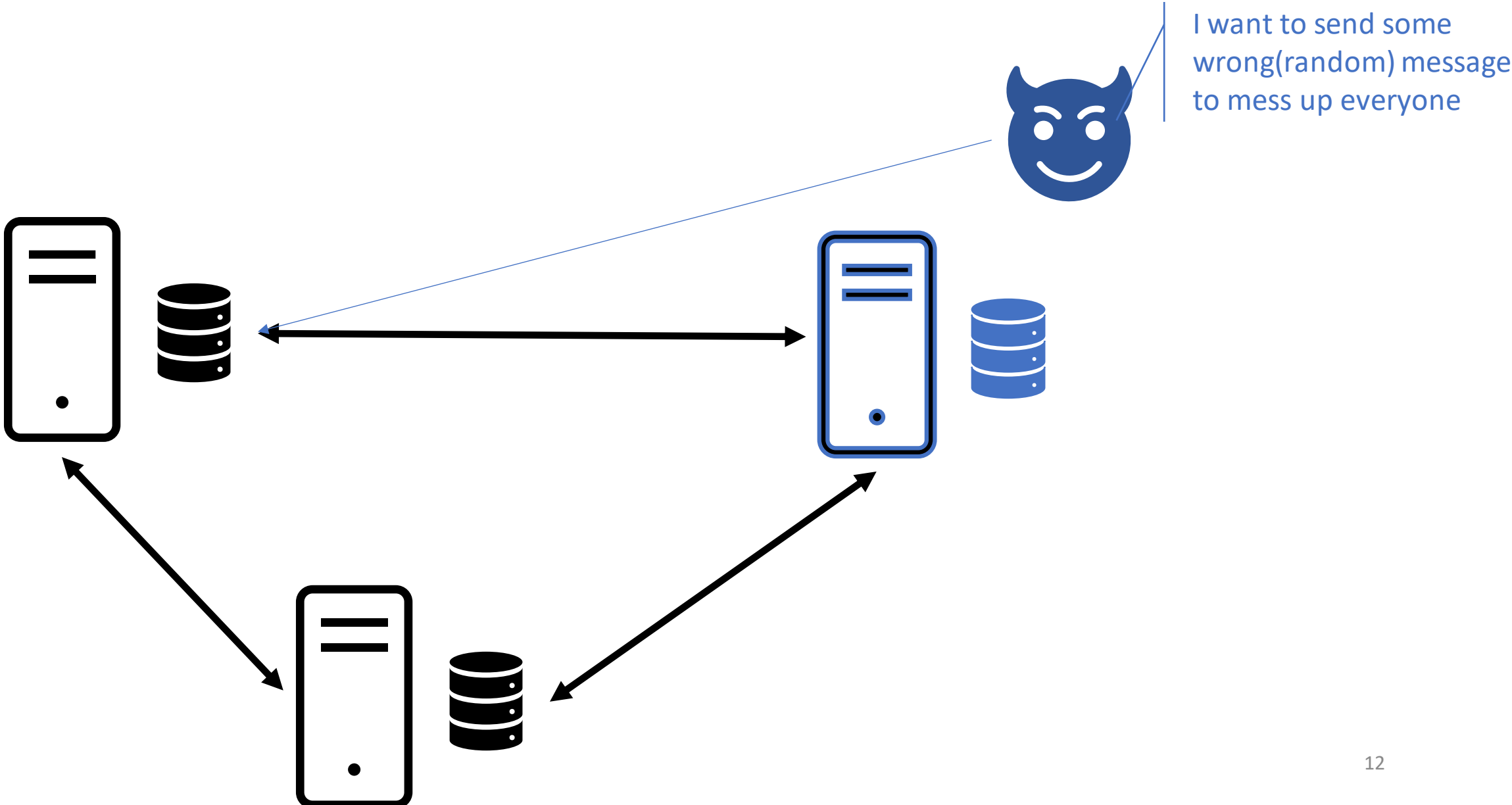
Implementations of State Machine Replication

--- two widely used protocols

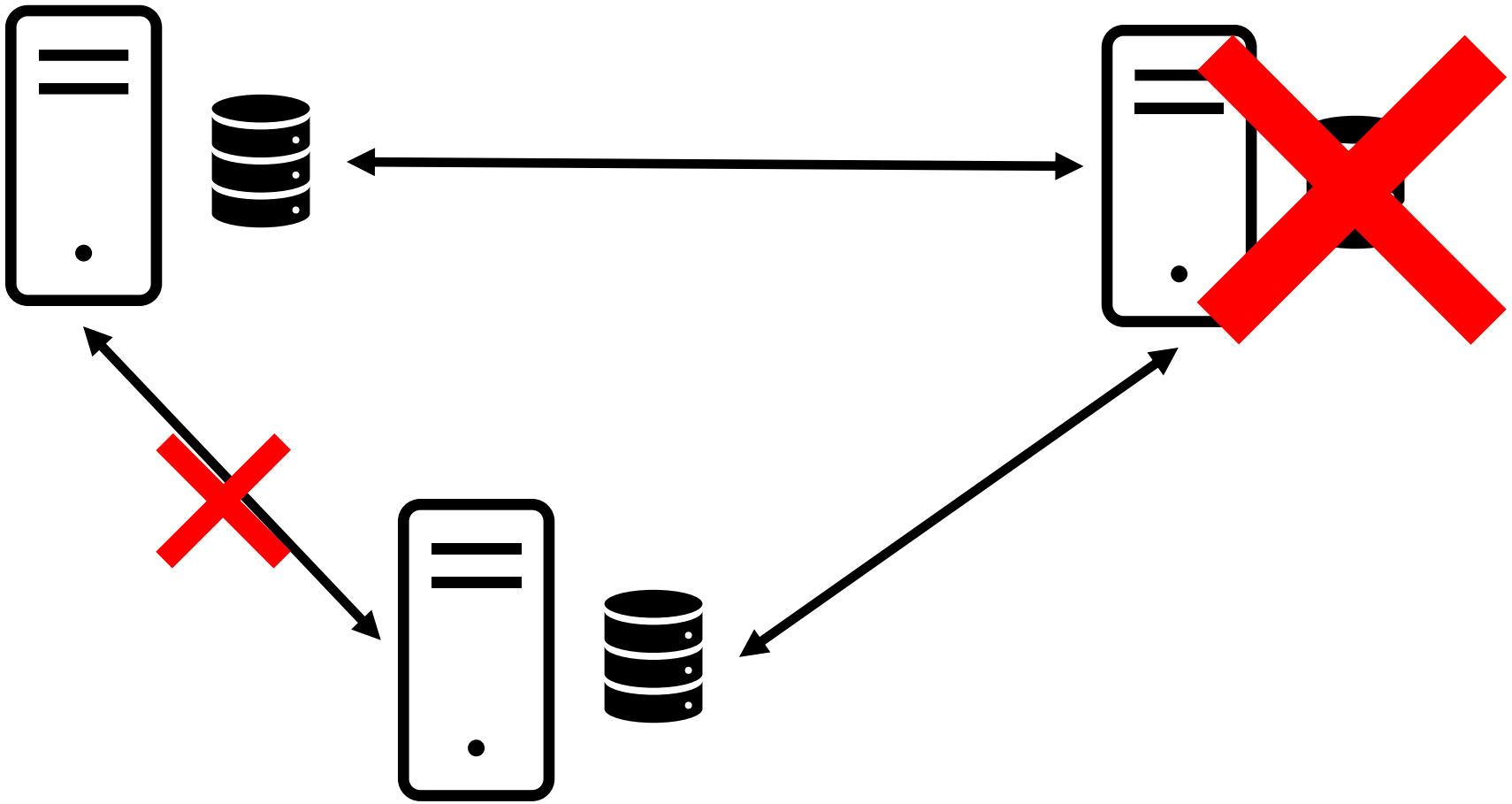
- Paxos
- Chain replication

Paxos and Consensus Proof

Failure Models --- Byzantine Failure



Failure Models --- Paxos



Basic Paxos

--- Roles

- Proposers
  - Active: put forth particular values to be chosen
  - Handle client request
- Acceptors:
  - Passive: respond to messages from proposers
  - Responses represent votes that from consensus
  - Store chosen value, state of the decision process
- Listeners:
  - Want to know which value was chosen (could also be implemented as part of acceptors)

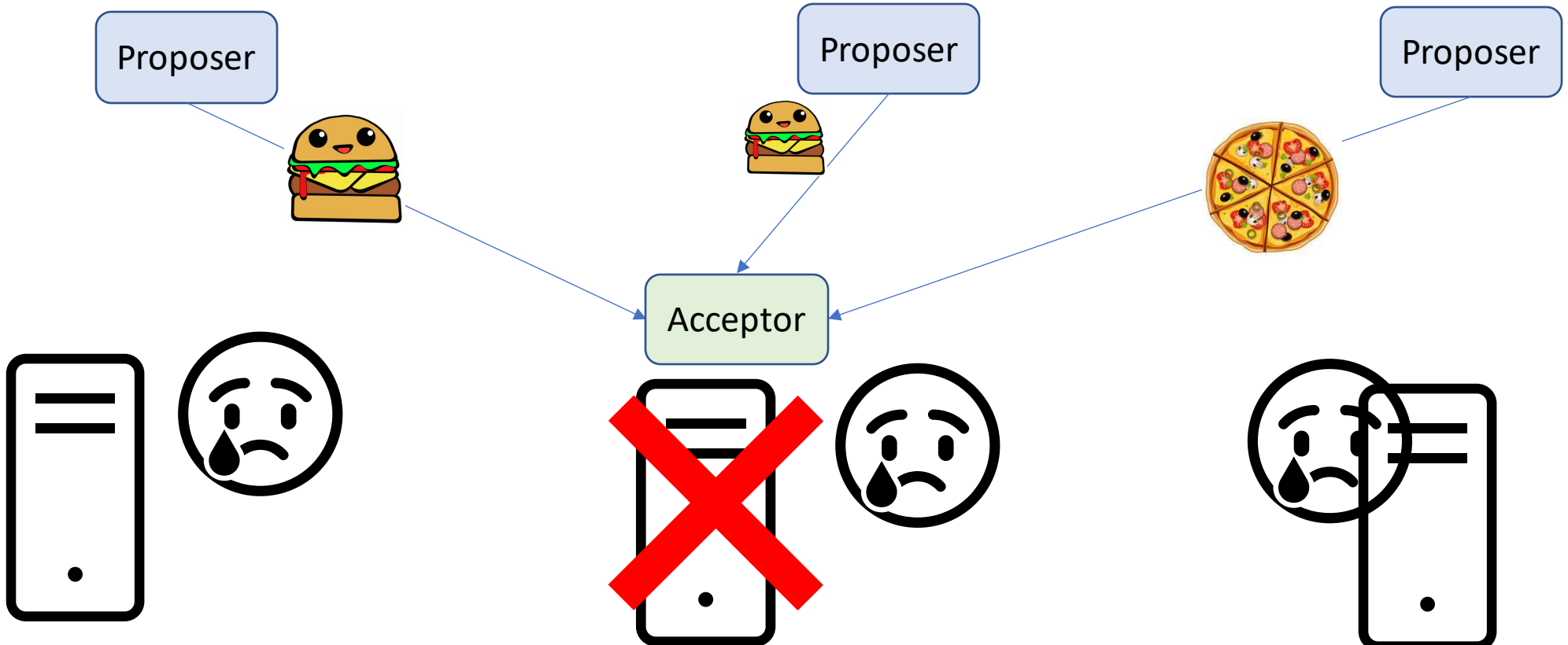
Consensus

- Safety:
  - Validity
    - Only a single value may be chosen
    - Only a value that has been proposed may be chosen
  - Integrity: A server never learns that a value has been chosen unless it really has
- Lively:
  - Termination: Some proposed value is eventually chosen
  - Agreement: If a value is chosen, servers eventually learn about it

Basic Paxos

Choosing **ONE** value

- Each server could be proposer, acceptor, listener at the same time
- Single acceptor doesn't work, but need to have a quorum of acceptors

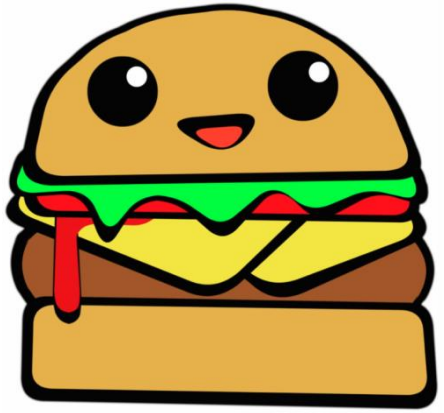


Basic Paxos

Choosing **ONE** value

- Each server could be proposer, acceptor, listener at the same time
- Single acceptor doesn't work, but need to have a quorum of acceptors
  - An acceptor may accept the proposed value.
  - The value is chosen when a large enough set of acceptors have accepted it.
  - How large is enough?

    To ensure at most one value is chosen, we let a large enough set consist of any majority of the agents.



or

?

Majority wins!

Two steps in Paxos:

1. What's happening
   - Need to ask majority
2. Let's go for burgers/pizza!
   - Majority must agree

Basic Paxos

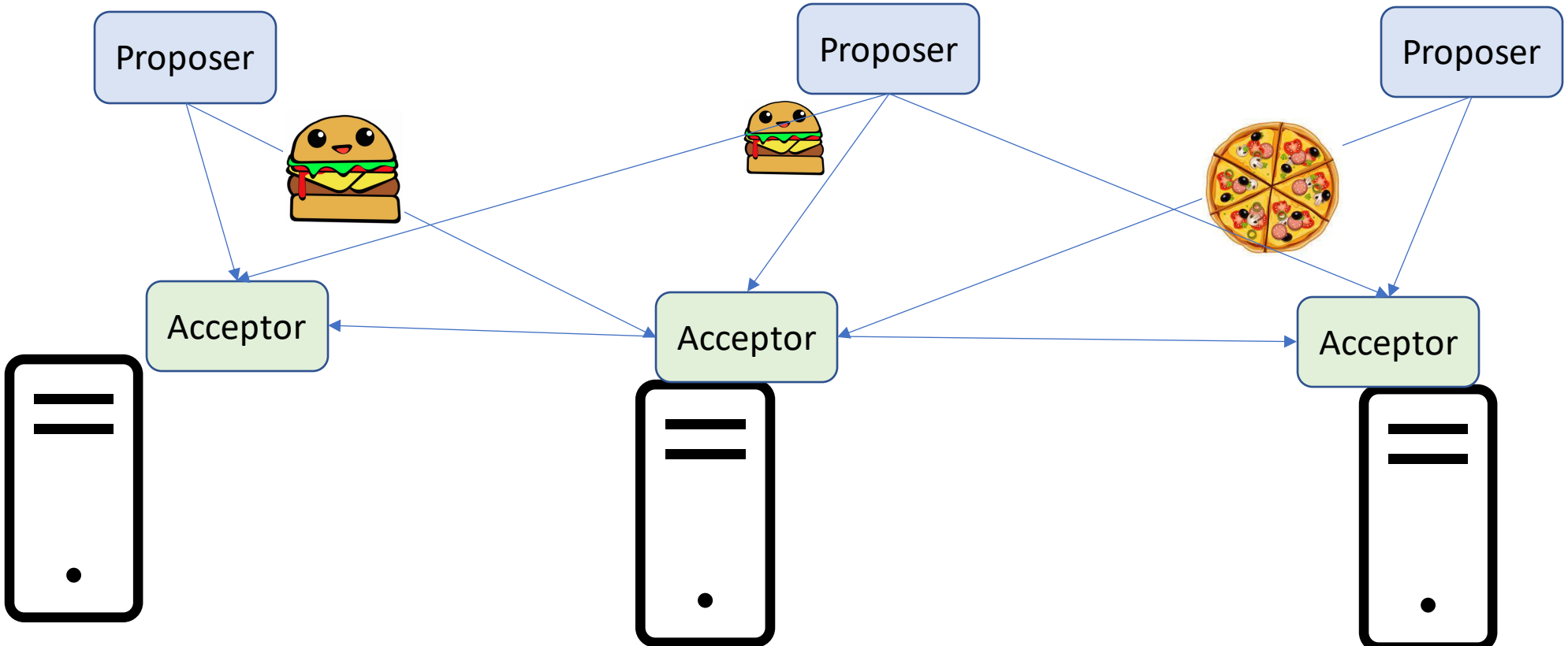
Choosing **ONE** value

- Each server could be proposer, acceptor, listener at the same time
- Single acceptor doesn't work, but need to have a quorum of acceptors
  - Needs multiple acceptors
  - Value v is chosen if it is accepted by a majority of acceptors
  - If one acceptor crashes, chosen value still available

Basic Paxos

Choosing **ONE** value

- Each server could be proposer, acceptor, listener at the same time
- Single acceptor doesn't work, but need to have a quorum of acceptors



Basic Paxos

Choosing **ONE** value

- Acceptors' Invariant

  (We want a value to be chosen even if only one value is proposed by a single proposer.)

  - P1. An acceptor must accept the first proposal that it receives.

Basic Paxos

Choosing **ONE** value

- Acceptors' Invariant

  (We want a value to be chosen even if only one value is proposed by a single proposer.)

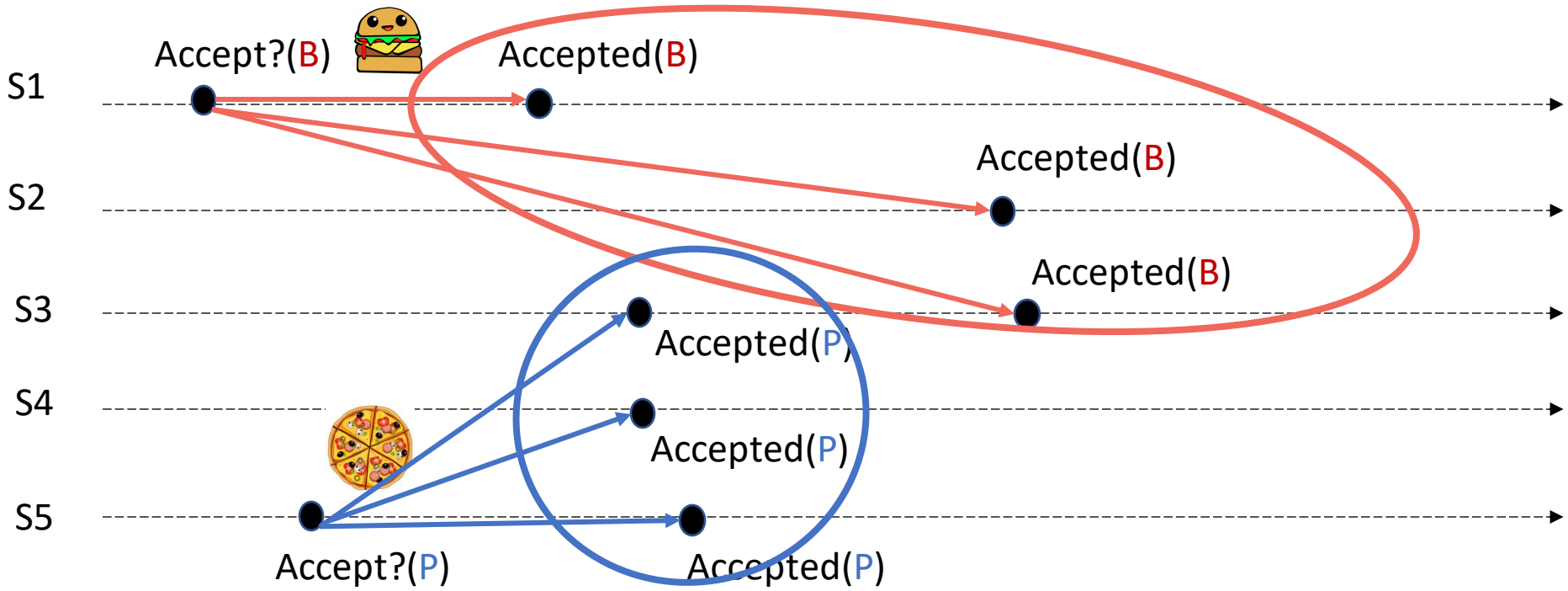
  - P1. An acceptor must accept the first proposal that it receives.

What if several values proposed by different proposers at the same time,
every acceptor has accepted a value,
but no single value is accepted by a majority of them?

Basic Paxos

Choosing **ONE** value

- Conflicting choices



Violate Safety property

Basic Paxos

- Each server could be proposer, acceptor, listener at the same time
- Single acceptor doesn't work, but need to have a quorum of acceptors
- Conflicting choices
  - Once chosen a value, the competing one needs to be aborted
  - Must order proposals, reject old ones

Basic Paxos

Proposal Numbers

Round Number	Server Id

- Each proposal has a unique number
  - Higher number take priority over lower number
  - Each time, the proposer chooses a new proposal number higher than anything it has seen/used before
- Proposal Number:
  - Each server stores maxRound locally: the largest round number it has seen so far
  - To generate a new proposal number: Increment maxRound, append with Server ID

Basic Paxos

Choosing **ONE** value

- Acceptors' Invariant

  (We want a value to be chosen even if only one value is proposed by a single proposer.)

  - P1. An acceptor must accept the first proposal that it receives.

  We can allow multiple proposals to be chosen, but we must guarantee that all chosen proposals have the same value. By induction on the proposal number, it suffices to guarantee:

  - P2. If a proposal with value v is chosen, then every higher-numbered proposal that is chosen has value v.

Basic Paxos

Choosing **ONE** value

- Acceptors' Invariant
  - P1. An acceptor must accept the first proposal that it receives.
  - P2. If a proposal with value v is chosen, then every higher-numbered proposal that is chosen has value v.
    - P2a. If a proposal with value v is chosen, then every higher-numbered proposal accepted by any acceptor has value v. (to ensure P2)
    - P2b. If a proposal with value v is chosen, then every higher-numbered proposal issued by any proposer has value v. (to ensure both P1 and P2a)

Basic Paxos

Choosing **ONE** value

- P2b. If a proposal with value v is chosen, then every higher-numbered proposal issued by any proposer has value v.
- Proof and showing how to satisfy P2b?
  - Assume that some proposal with number m and value v is chosen. Show that any proposal issued with number $n > m$ also has value v.
  - Using induction on n: every proposal issued with proposal number in $m \,..\, (n - 1)$ has value v
  - For the proposal numbered m to be chosen, there must be some set C consisting of a majority of acceptors such that every acceptor in C accepted it
  - By induction, every proposal with number in $m \,..(n - 1)$ accepted by any acceptor has value v.

To satisfy P2b, we need P2c invariant

Basic Paxos

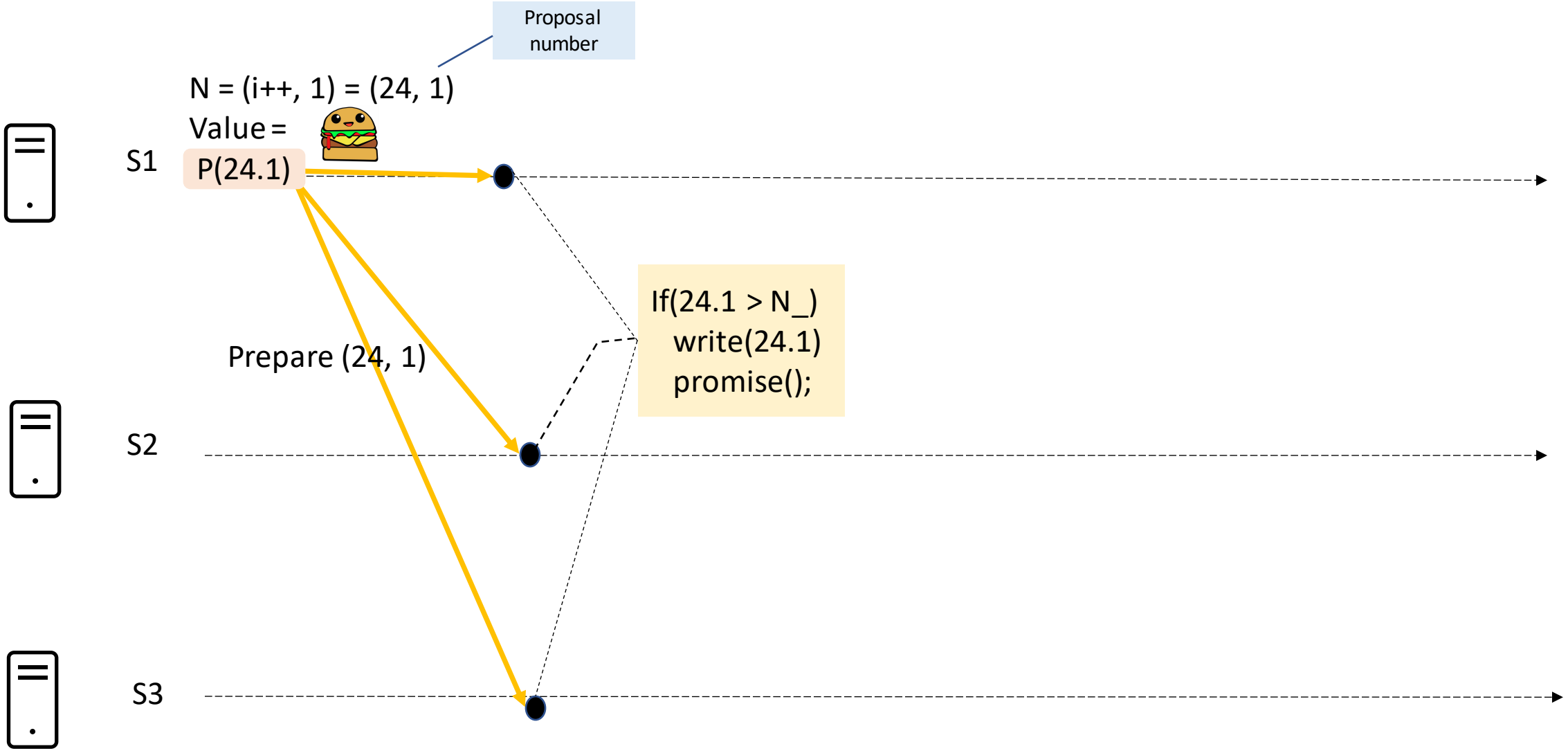
Choosing **ONE** value

- Acceptors' Invariant
  - P1. An acceptor must accept the first proposal that it receives.
  - P2. If a proposal with value v is chosen, then every higher-numbered proposal that is chosen has value v.
    - P2a. If a proposal with value v is chosen, then every higher-numbered proposal accepted by any acceptor has value v. (to ensure P2)
    - P2b. If a proposal with value v is chosen, then every higher-numbered proposal issued by any proposer has value v. (to ensure both P1 and P2a)
    - For any v and n, if a proposal with value v and number n is issued, then there is a set S consisting of a majority of acceptors such that either
      (a) No acceptor in S has accepted any proposal numbered less than n, or
      (b) v is the value of the highest-numbered proposal among all proposals numbered less than n accepted by the acceptors in S. (to ensure P2b)

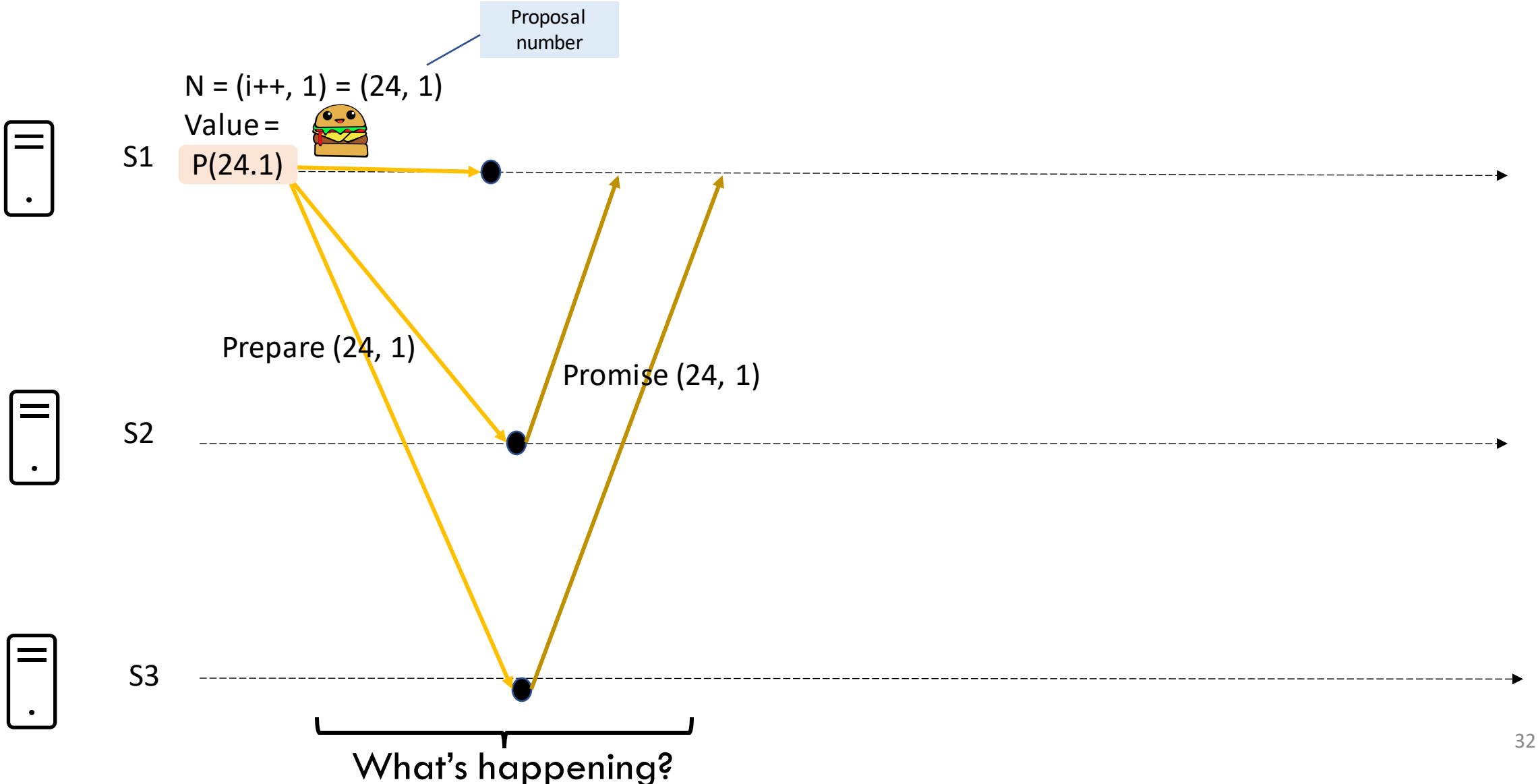
Basic Paxos

	Proposers	Acceptors
Prepare Phase (P)	1) Choose new proposal number n 2) Broadcast Prepare(n) to all servers	3) Respond to Prepare(n) • If n > minProposal then minProposal = n • Return(acceptedProposal, acceptedValue)
	4) When responses received from majority: If any acceptedValues retuned, replace value with acceptedValue for highest acceptedProposal	
Accept Phase (A)	5) Broadcast Accept(n, value) to all servers	6) Respond to Accept(n,value): • If $n \geq minProposal$ then acceptedProposal = minProposal = n acceptedValue = value • Return(minProposal)
	6) When responses received from majority • Any rejections(result > n)? Back to step(1) • Otherwise, value is chosen	

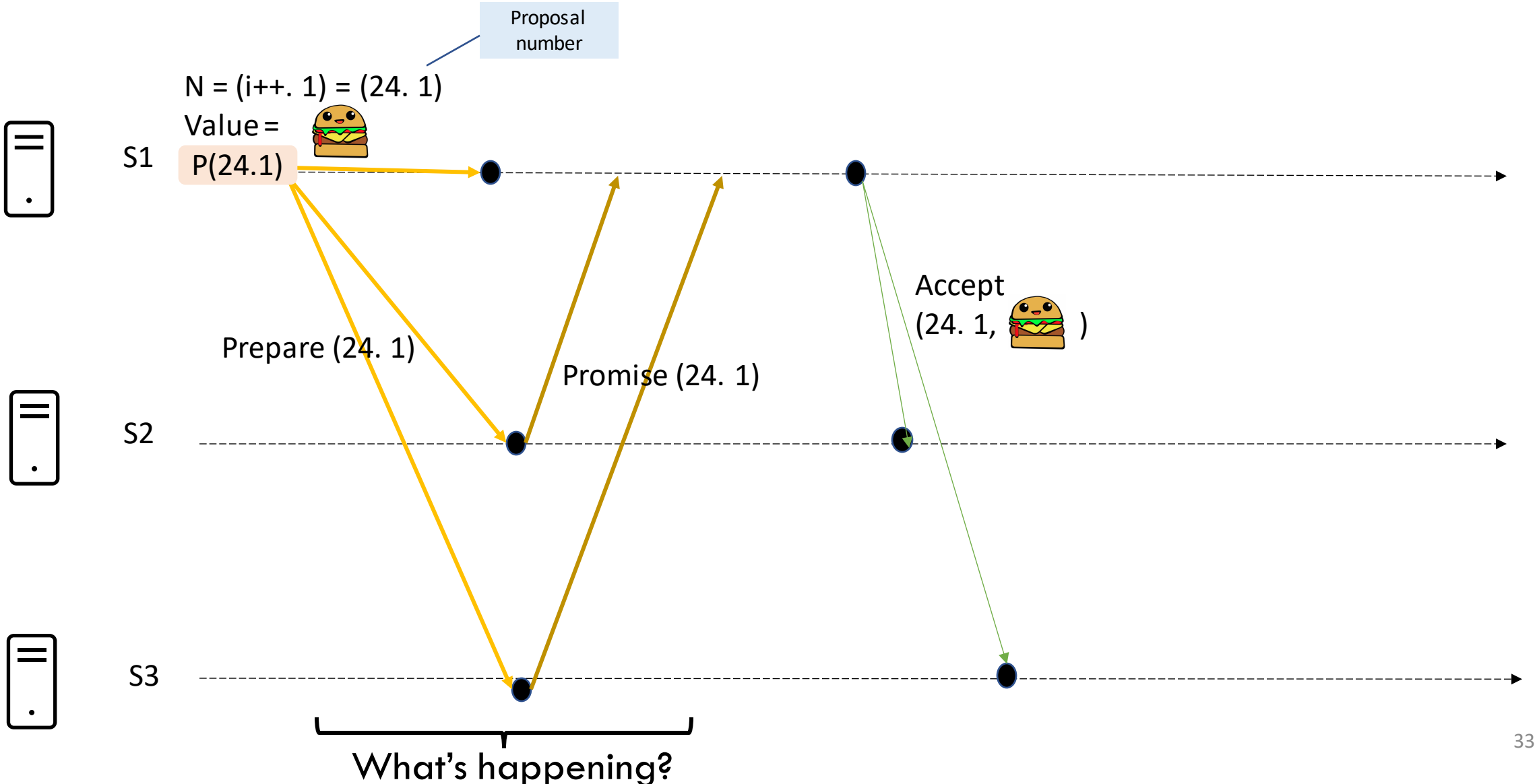
Basic Paxos --- Base case



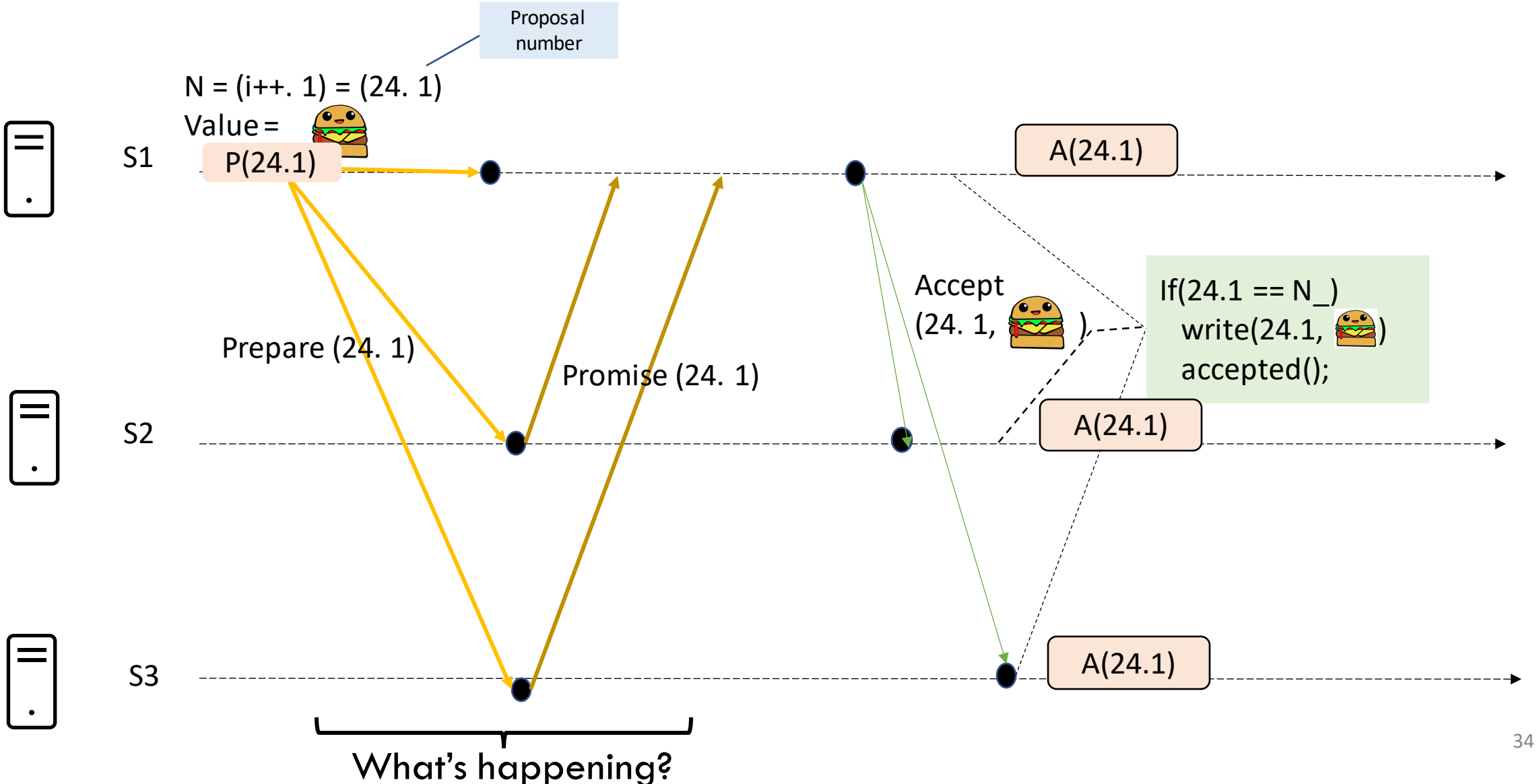
Basic Paxos --- Base case



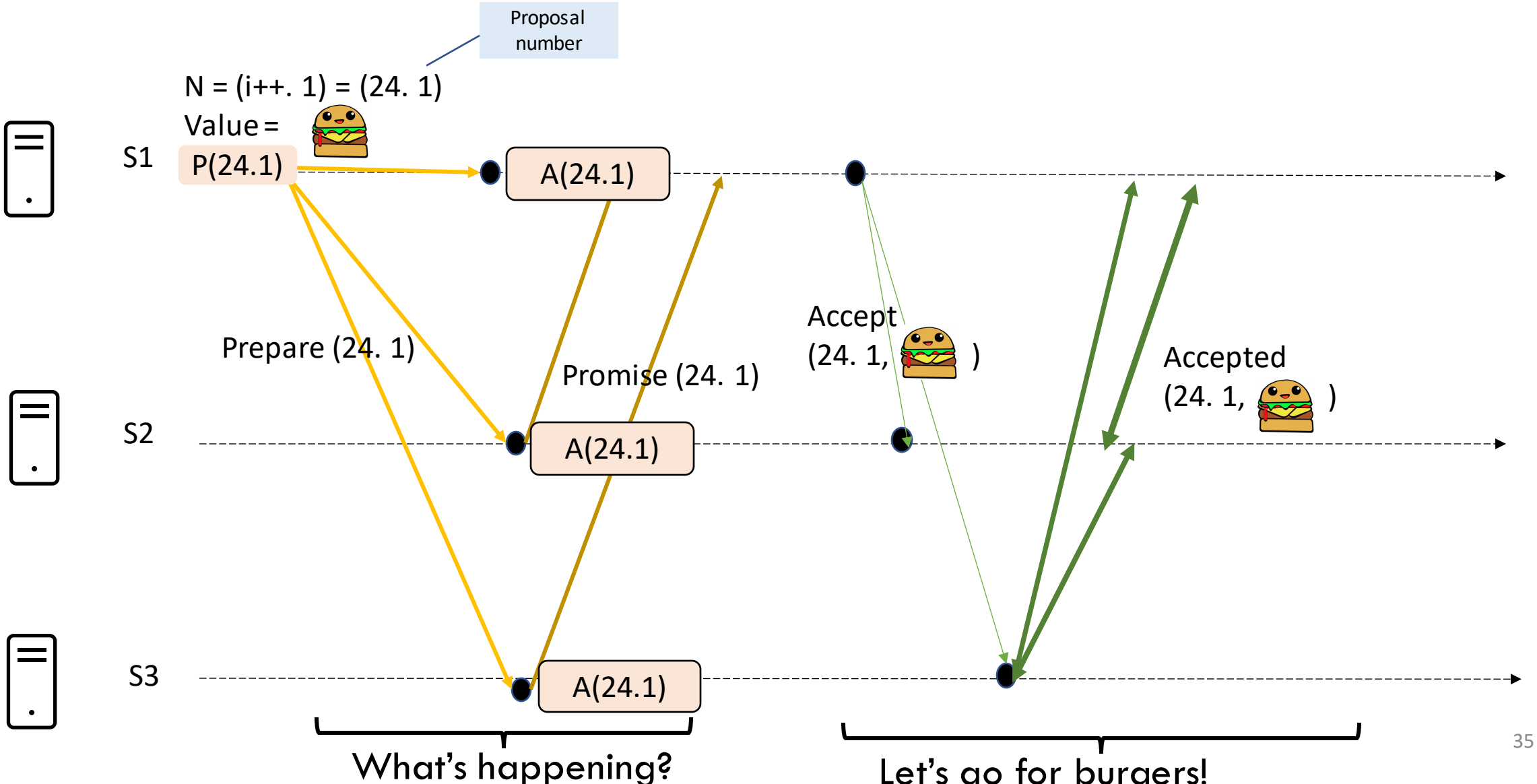
Basic Paxos --- Base case



Basic Paxos --- Base case



Basic Paxos --- Base case



Basic Paxos

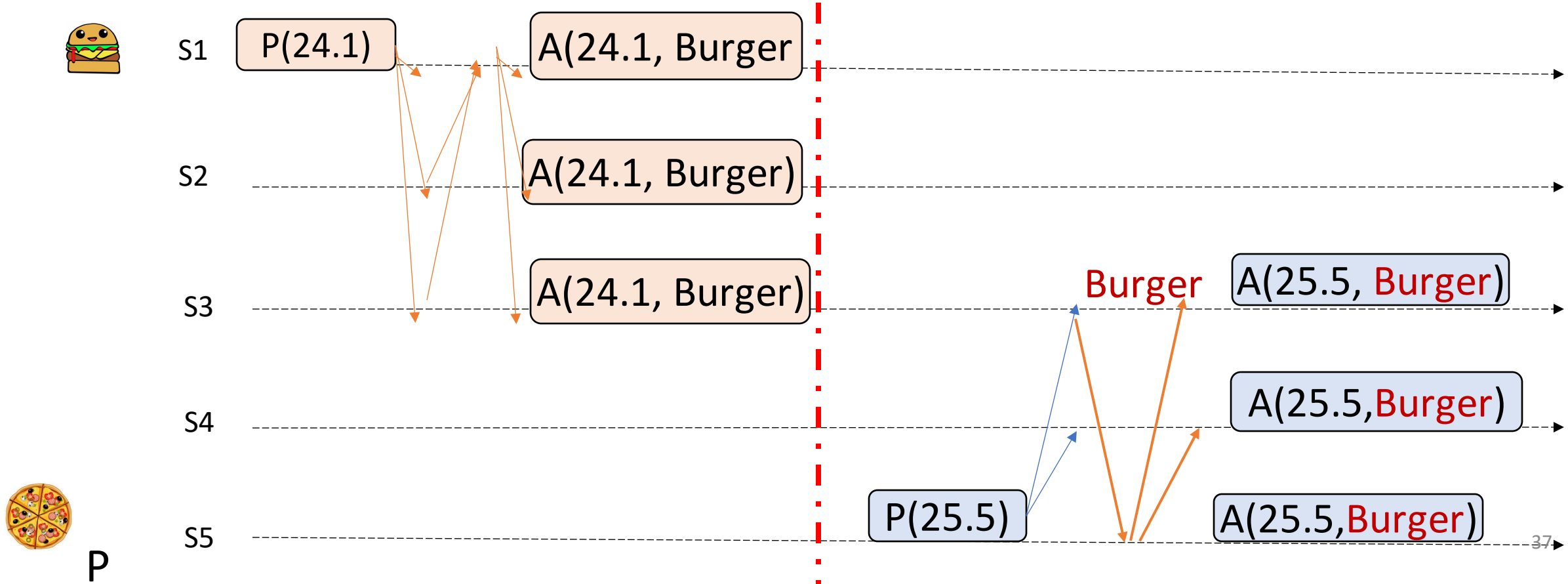
Choosing **ONE** value

- Acceptors' Invariant
  - P1. An acceptor must accept the first proposal that it receives.
  - P2. If a proposal with value v is chosen, then every higher-numbered proposal that is chosen has value v.
    - P2a. If a proposal with value v is chosen, then every higher-numbered proposal accepted by any acceptor has value v. (to ensure P2)
    - P2b. If a proposal with value v is chosen, then every higher-numbered proposal issued by any proposer has value v. (to ensure both P1 and P2a)
    - For any v and n, if a proposal with value v and number n is issued, then there is a set S consisting of a majority of acceptors such that either
      (a) No acceptor in S has accepted any proposal numbered less than n, or
      (b) v is the value of the highest-numbered proposal among all proposals numbered less than n accepted by the acceptors in S. (to ensure P2b)

Basic Paxos --- Competing proposals

Three possibility when later proposal prepares:

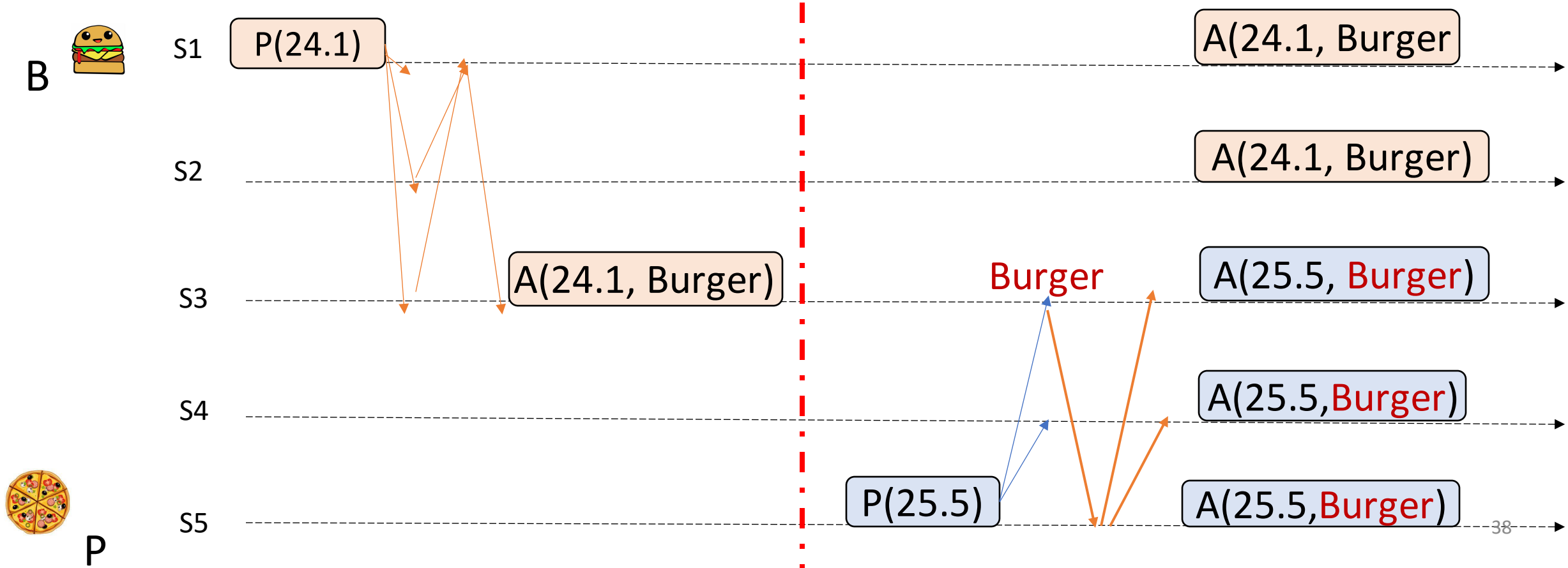
1. Previous value already chosen → new proposal will find it and use it



Basic Paxos --- Competing proposals

Three possibility when later proposal prepares:

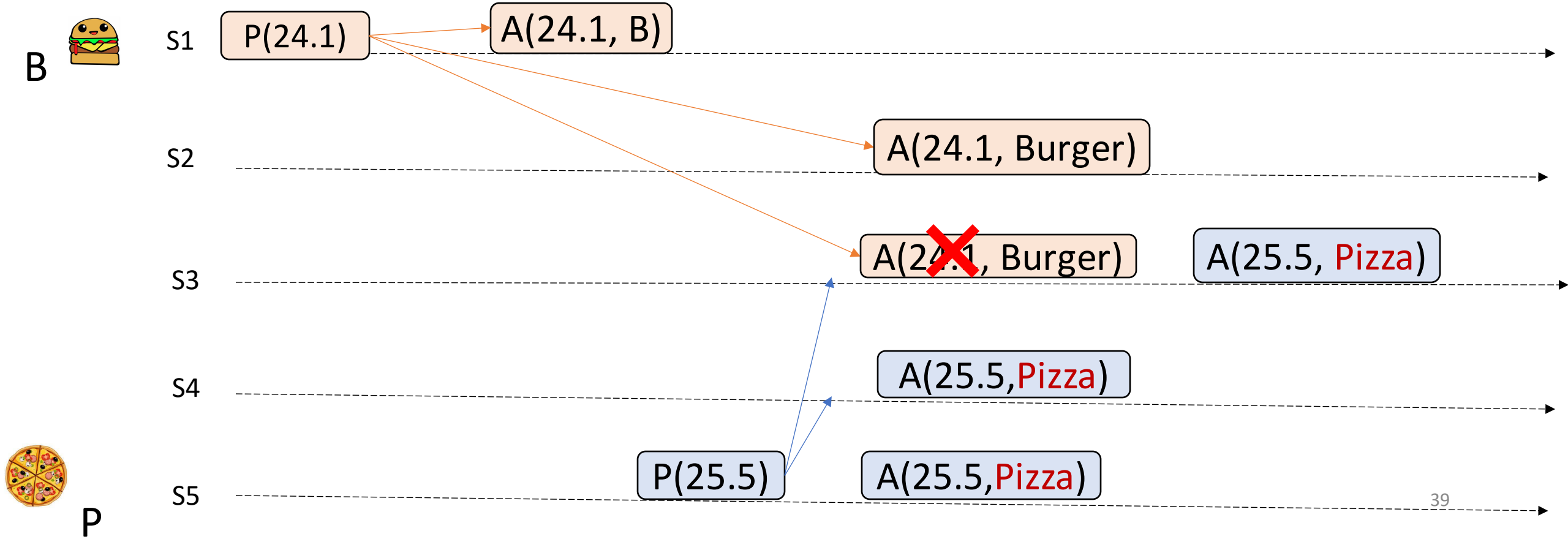
2. Previous value NOT chosen, but new proposal sees it
   - New proposer will use the existing value, and both proposers can succeed



Basic Paxos --- Competing proposals

Three possibility when later proposal prepares:

3. Previous value NOT chosen, but new proposal does NOT see it
   - New proposer uses its own value, and older proposal blocked



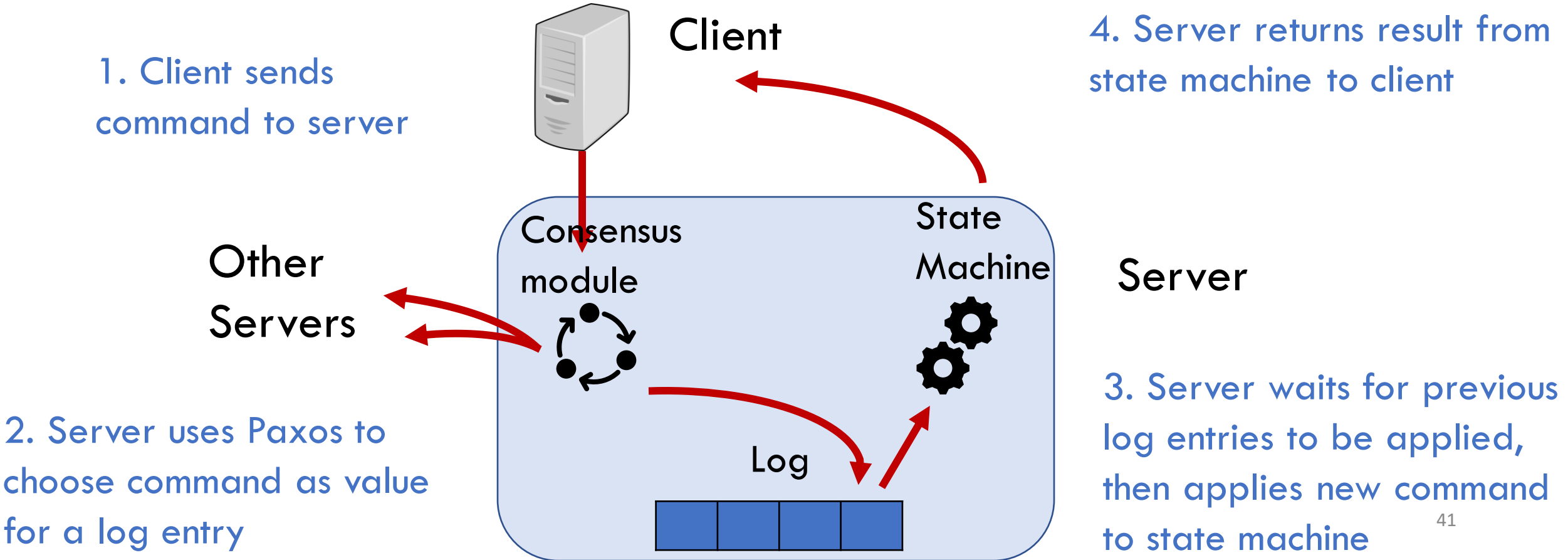
Building practical Paxos applications

- Creating log of agreements
  - Multi-Paxos
- Adding and removing nodes from Paxos
  - Group management
- Applications: Zookeeper, Derecho

Referencing: [7]

Building a practical Paxos application: Multi-Paxos

- Separate instance of Basic Paxos for each entry in the log:
  - Add index argument to Prepare and Accept (select entry in log)

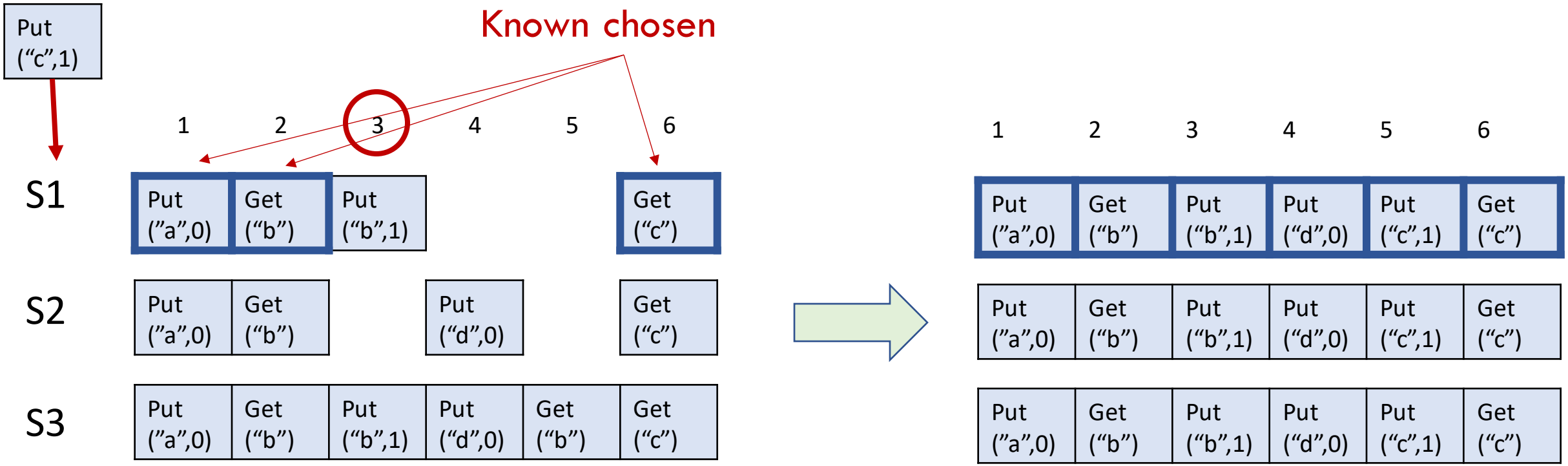


Building a practical Paxos application: Multi-Paxos

- Which log entry to use for a given client request?
- Client protocol

Selecting Log entries

- When request arrives from client:
  1. Find first log entry not known to be chosen



Logs before

Logs after

Referencing: [7]

Selecting Log entries

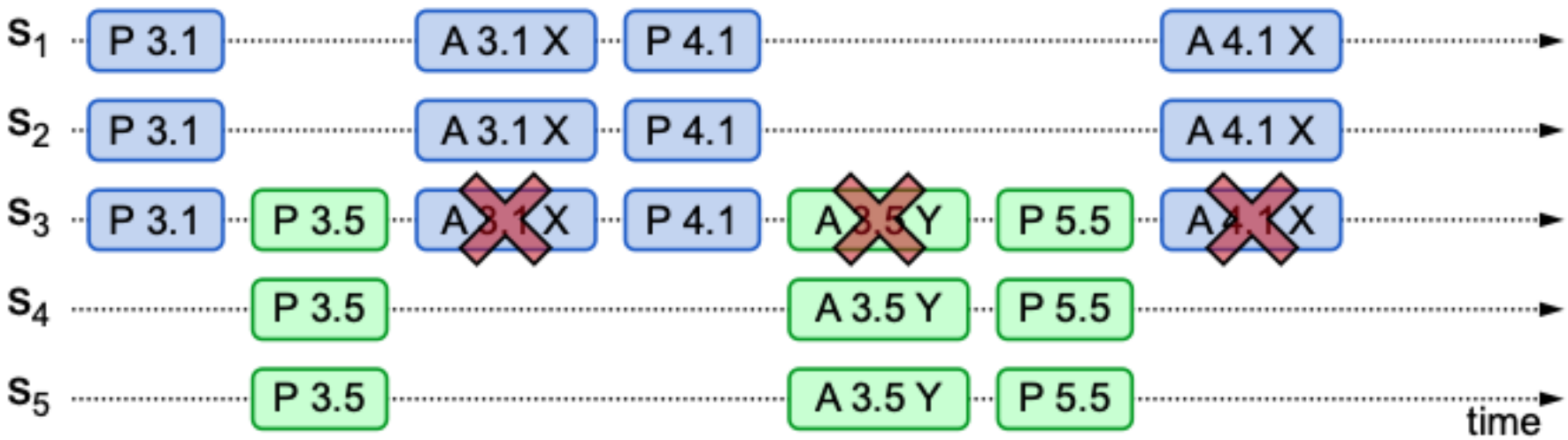
- Servers can handle multiple client requests concurrently
  - Select different log entries for each
- Must apply commands to state machine in log order

Improving Efficiency

- Use Basic Paxos is inefficient:
  - With multiple concurrent proposers, conflicts and restarts are likely (higher load → more conflicts)
  - 2 rounds of RPCs for each value chosen (Prepare, Accept)

Basic Paxos --- What could go wrong?

- Two or more simultaneous proposers
  - Can live lock
  - Avoid with leader election



Graph taken from[7]

Improving Efficiency

- Use Basic Paxos is inefficient:
  - With multiple concurrent proposers, conflicts and restarts are likely (higher load → more conflicts)
  - 2 rounds of RPCs for each value chosen (Prepare, Accept)
- Solutions:
  - Pick a leader:
    - At any given time, only one server acts as Proposer
  - Eliminate most Prepare RPCs
    - Prepare once for the entire log (not once per entry)
    - Most log entries can be chosen in a single round of RPCs

Referencing: [7]

Leader Election

- One simple approach from Lamport:
  - Let the server with highest ID act as leader
  - Each server sends a heartbeat message to every other server every T ms
  - If a server hasn't received heartbeat from server with higher ID in last 2T ms, it acts as leader:
    - Accepts requests from clients
    - Act as proposer and acceptor
  - If server not leader:
    - Rejects client requests (or redirect to leader)
    - Acts only as acceptor

Eliminating Prepares

- Why is Prepare needed?
  - Block old proposals
    - Make proposal numbers refer to the entire log, not just one entry
  - Find out about (possibly) chosen values
    - Return highest proposal accepted for current entry
    - Return noMoreAccceptred: no proposals accepted for any log entry beyond current one
  - If acceptor responds to Prepare with noMoreAccepted, skip future Prepares with that acceptor (until Accept rejected)
  - Once leader receives nomoreAccepted from majority of acceptors, no need for Prepare RPCs
    - Only 1 round of RPCs needed per log entry (Accepts)

Building a practical Paxos application: Client protocol

- Send commands to leader
  - If leader unknown, contact any server
  - If contacted server not leader, it will redirect to leader
- Leader does not respond until command has been chosen for log entry and executed by leader's state machine
- If request times out (e.g., leader crash):
  - Client reissues command to some other server
  - Eventually redirected to new leader
  - Retry request with new leader

Implementations of State Machine Replication

- Paxos
  - leaders and acceptors process, and voting mechanism
- Chain replication

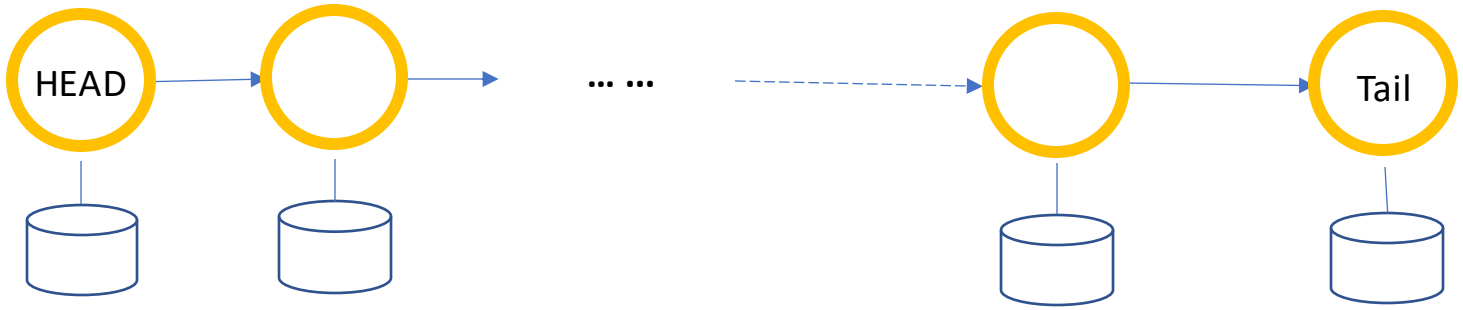
Chain Replication

Chain Replication

- Large-scale storage services
- High throughput and availability
- Strong consistency guarantee : All accesses are seen by all parallel processes (nodes) in the same order (sequentially)
  1. Operations to query or update are executed in sequential order
  2. The effects of an update operation are reflected on all subsequent query operations, and on all servers.

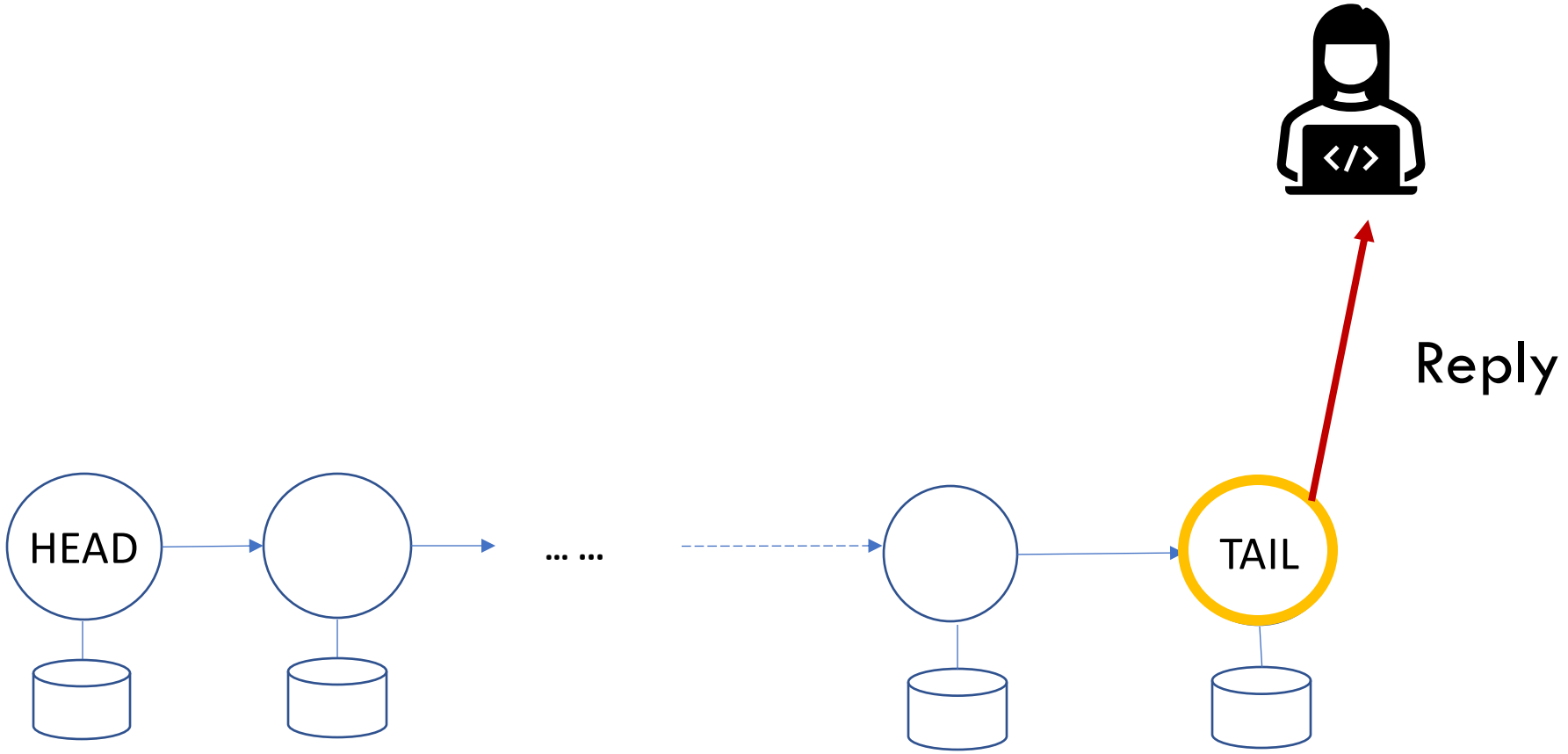
Chain Replication

- The servers replicating the objects are linearly ordered to form a chain.



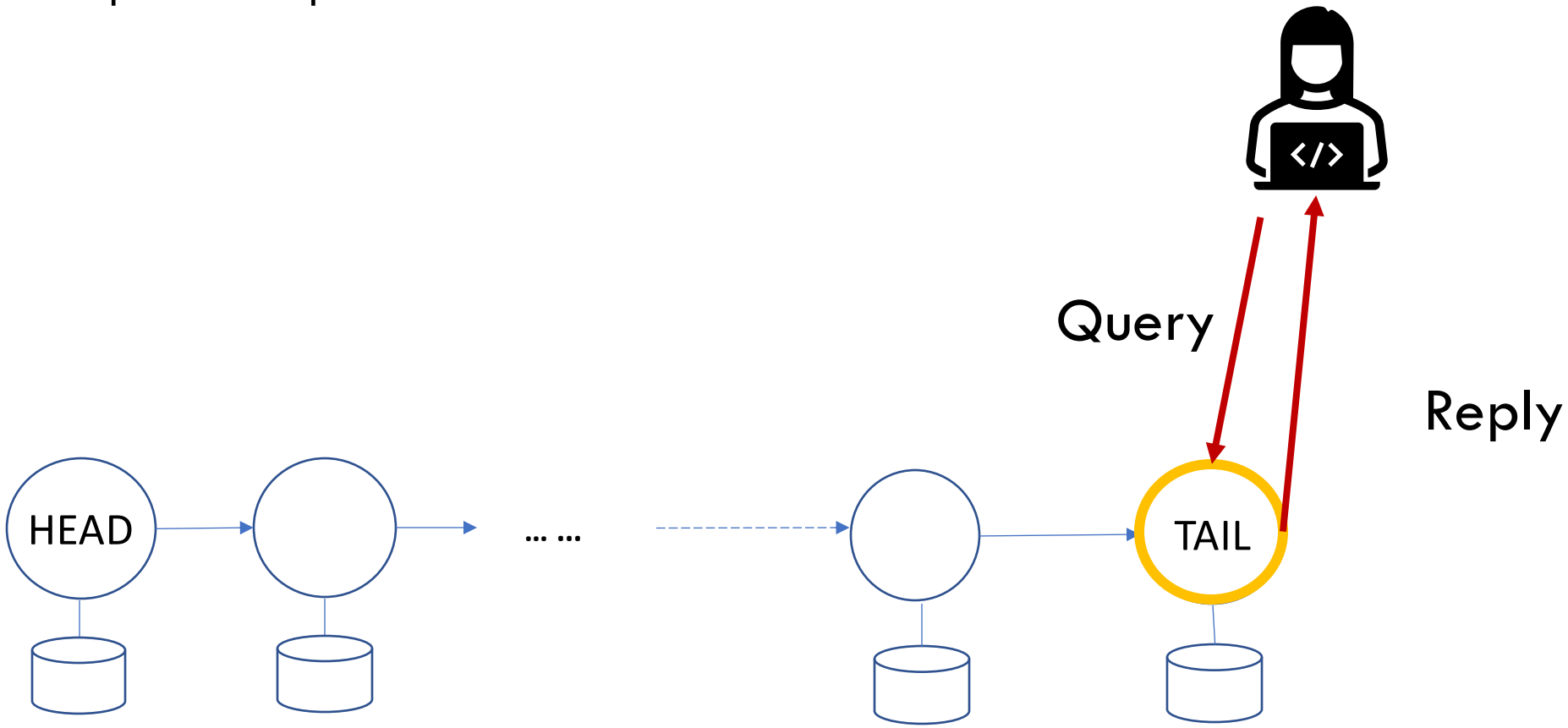
Chain Replication

- Reply Generation.
  - The reply for every request is generated and sent by the tail.



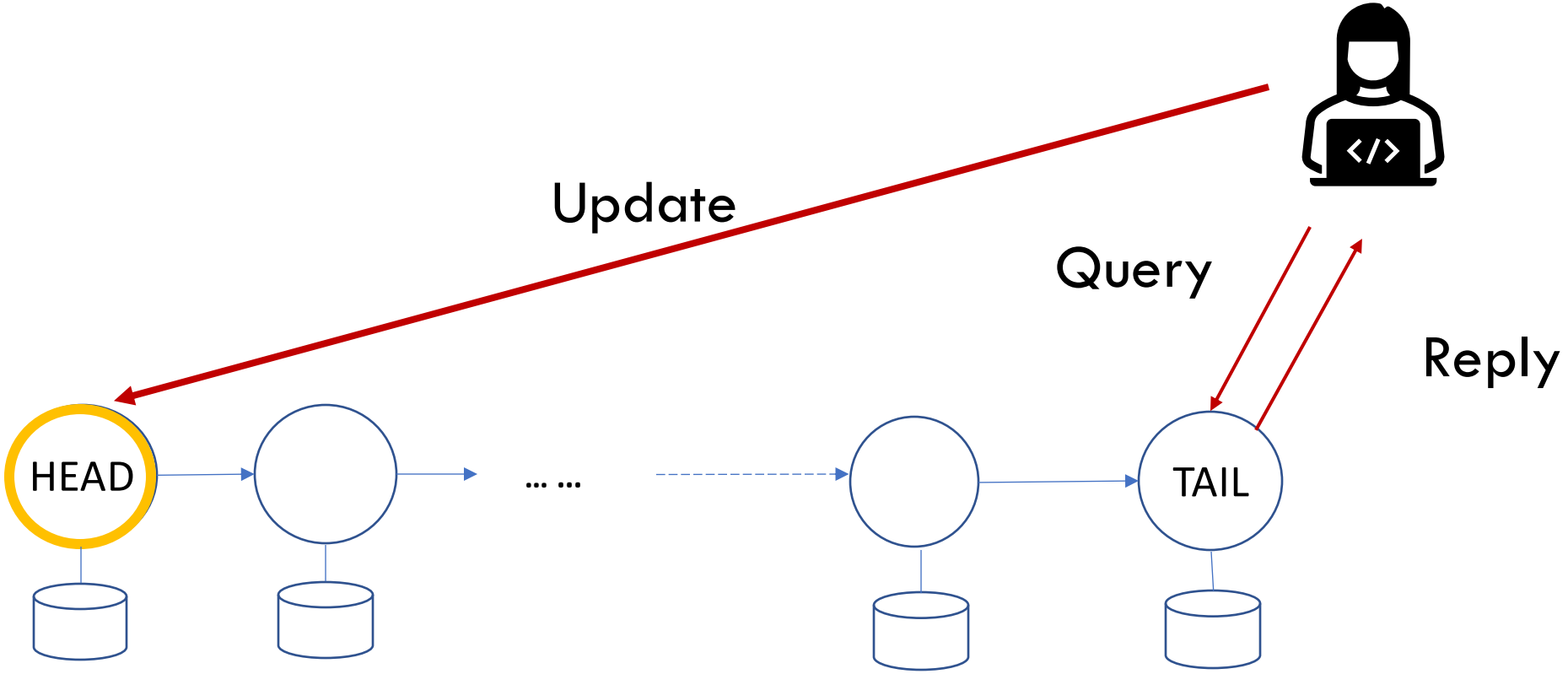
Chain Replication

- Query Processing
  - Each query request(READ) is directed to the tail of the chain and processed there atomically using the replica of objID stored at the tail.



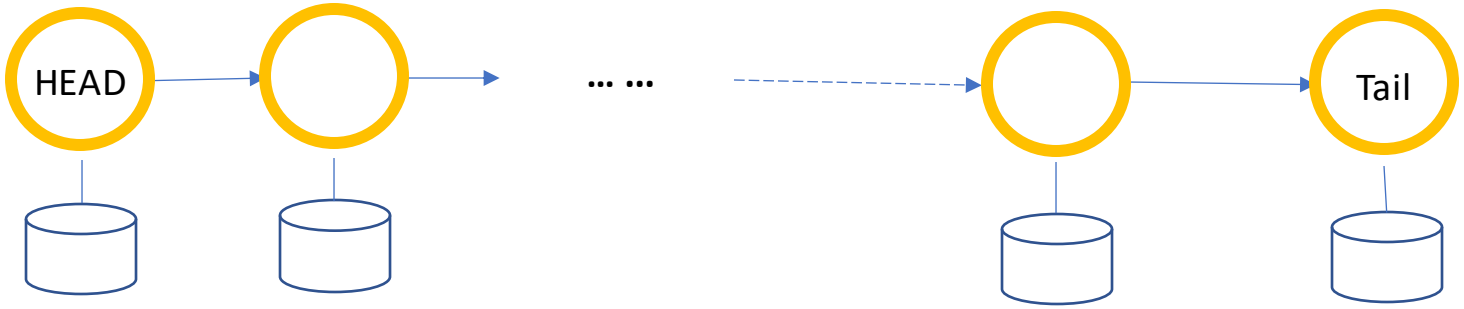
Chain Replication

- Update Processing:
  - Each update request is directed to the head of the chain. The request is processed there atomically using replica of objID at the head, then state changes are forwarded along a reliable FIFO link to the next element of the chain



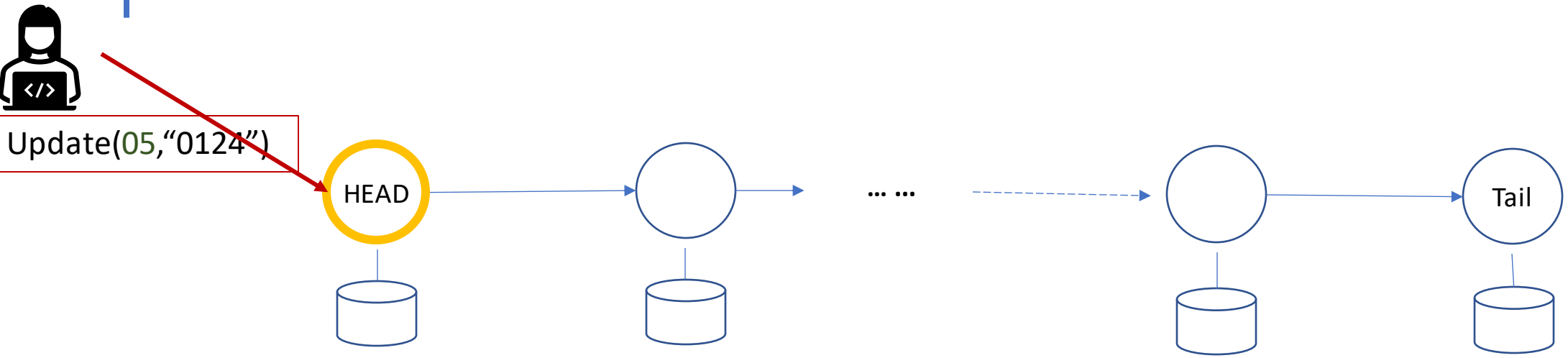
Chain Replication

- Variables stored on each replica servers:
  - Hist: The list of all updates applied to an object objID on a server i
  - Sent: the list of updates forwarded by the server i that haven't been processed by the tail server yet
  - Pending: the list of requests received for an object objID by any server in the chain that haven't been processed by the tail yet



Chain Replication Protocol

T1: Client request r arrives:
$Pending_{objID} := Pending_{objID} \cup \{r\}$

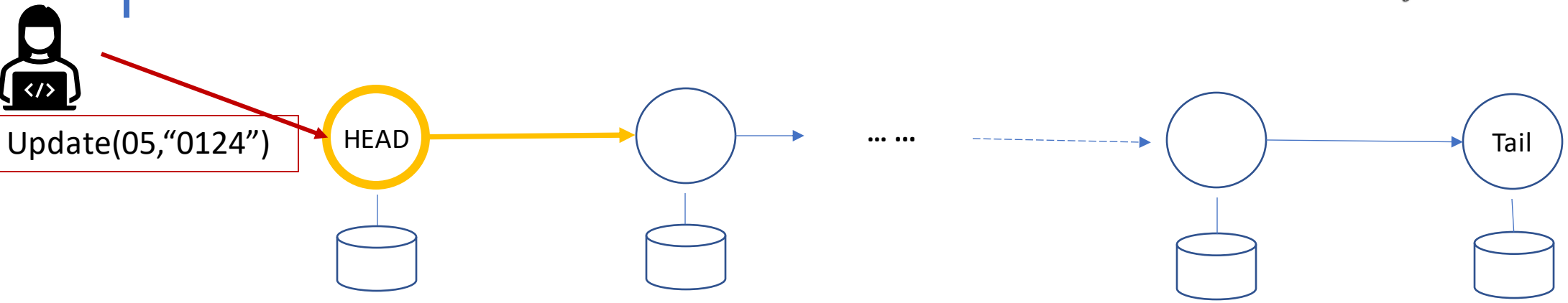


States for ObjID = 05

Pending	**Update(05,“0124”)**		**….**		
Sent					
Hist					

Chain Replication Protocol

T1: Client request r arrives:
$Pending_{objID} := Pending_{objID} \cup \{r\}$

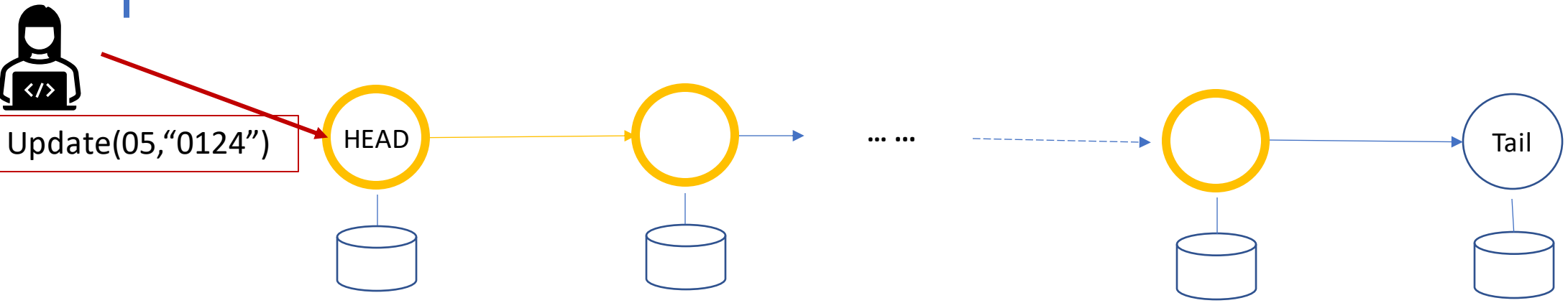


States for ObjID = 05

Pending			**….**		
Sent	**Update(05,“0124”)**				
Hist	Update(05,”0000”) Update(05,“1113”)	Update(05,”0000”) Update(05,“1113”)		Update(05,”0000”) Update(05,“1113”)	Update(05,”0000”) Update(05,“1113”)

Chain Replication Protocol

T2: Client request $r \in Pending_{objID}$ ignored:
$Pending_{objID} := Pending_{objID} - \{r\}$

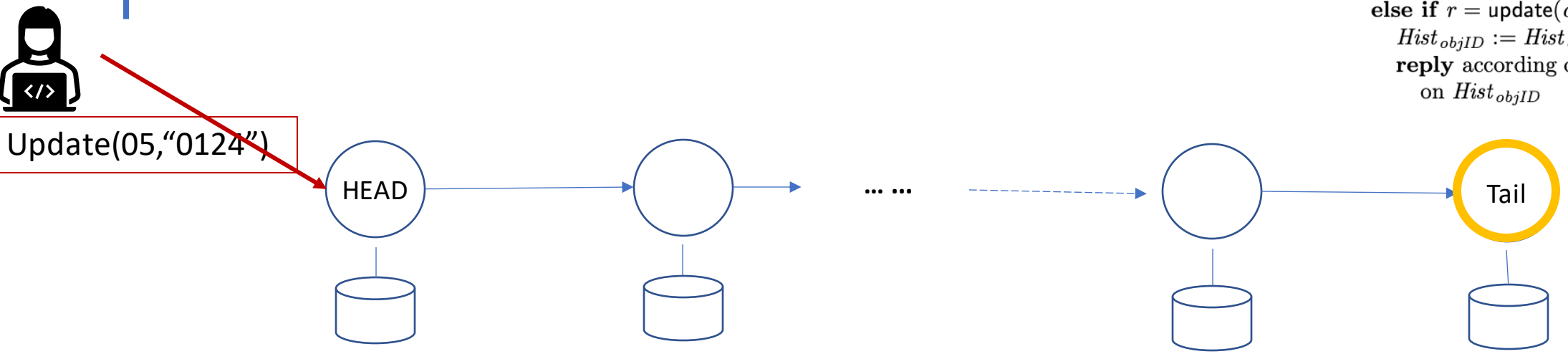


States for ObjID = 05

Pending			….		
Sent	**Update(05,“0124”)**	**Update(05,“0124”)**		**Update(05,“0124”)**	
Hist	Update(05,”0000”) Update(05,“1113”)	Update(05,”0000”) Update(05,“1113”)		Update(05,”0000”) Update(05,“1113”)	Update(05,”0000”) Update(05,“1113”)

Chain Replication Protocol

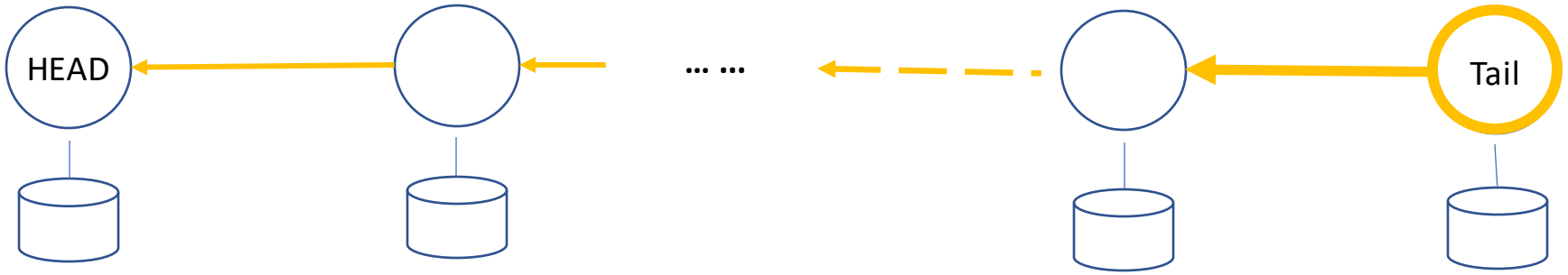
T3: Client request $r \in Pending_{objID}$ processed:
$Pending_{objID} := Pending_{objID} - \{r\}$
if $r = \text{query}(objId, opts)$ **then**
reply according options $opts$ based on $Hist_{objID}$
else if $r = \text{update}(objId, newVal, opts)$ **then**
$Hist_{objID} := Hist_{objID} \cdot r$
reply according options $opts$ based on $Hist_{objID}$



States for ObjID = 05

Pending			….		
Sent	Update(05,“0124”)	Update(05,“0124”)		Update(05,“0124”)	**Update(05,“0124”)**
Hist	Update(05,”0000”) Update(05,“1113”)	Update(05,”0000”) Update(05,“1113”)		Update(05,”0000”) Update(05,“1113”)	Update(05,”0000”) Update(05,“1113”)

Chain Replication Protocol



States for ObjID = 05

Pending			….		
Sent					Update(05,“0124”)
Hist	Update(05,”0000”) Update(05,“1113”) Update(05,“0124”)	Update(05,”0000”) Update(05,“1113”) Update(05,“0124”)		Update(05,”0000”) Update(05,“1113”) Update(05,“0124”)	Update(05,”0000”) Update(05,“1113”) Update(05,“0124”)

Chain Replication Protocol

Update Propagation Invariant: For servers labeled i and j such that i<=j holds (i.e., i is a predecessor of j in the chain), then $Hist^{j}_{objID} \preceq Hist^{i}_{objID}$.

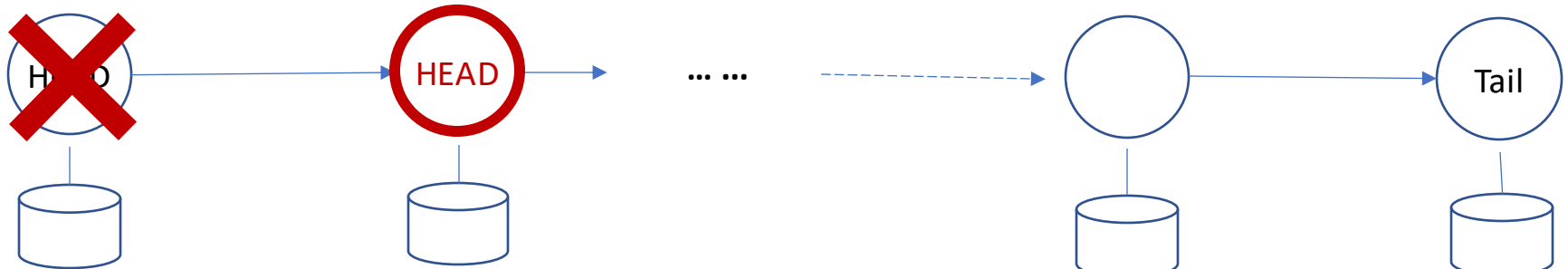
States for ObjID = 05

Pending			….		
Sent					
Hist	Update(05,"0000") Update(05,"1113") Update(05,"0124")	Update(05,"0000") Update(05,"1113") Update(05,"0124")		Update(05,"0000") Update(05,"1113") Update(05,"0124")	Update(05,"0000") Update(05,"1113") Update(05,"0124")

Coping with Server Failures

- Head fail:
  - The master removes H from the chain and makes the successor to H the new head of the chain. (Such a successor must exist if our assumption holds that at most t − 1 servers are faulty.)
  - If the head server was applying an update and failed before forwarding it, the update is removed from pending list. Client will re-issue the update.
  - Consistent with T2

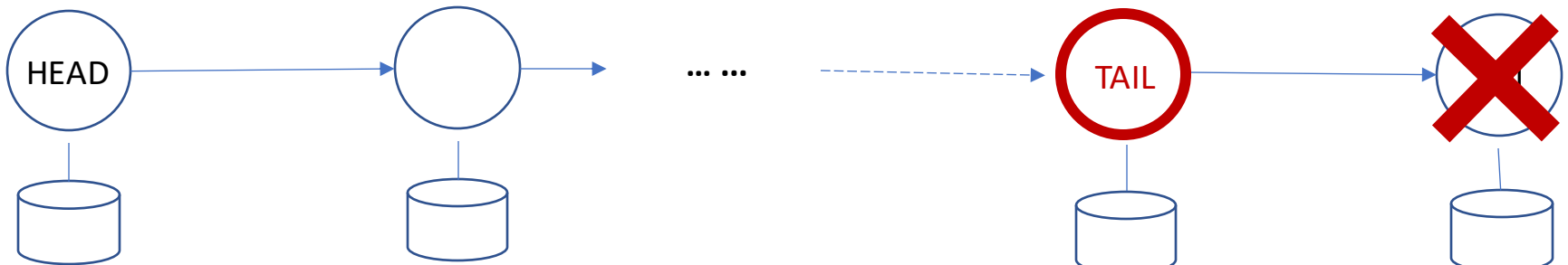
$$\text{T2: Client request } r \in Pending_{objID} \text{ ignored:}$$
$$Pending_{objID} := Pending_{objID} - \{r\}$$



Coping with Server Failures

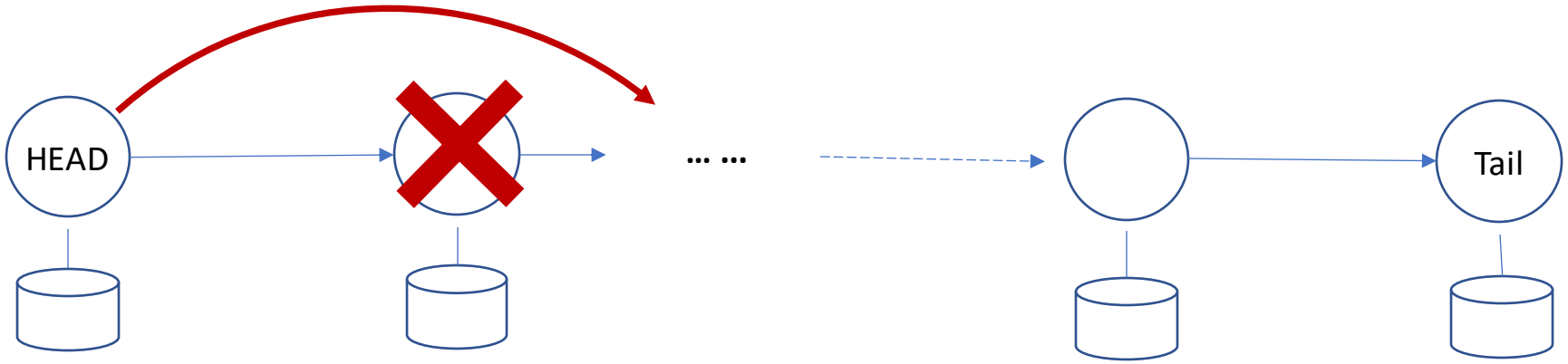
T3: Client request $r \in Pending_{objID}$ processed:
$Pending_{objID} := Pending_{objID} - \{r\}$
if $r = \text{query}(objId, opts)$ **then**
reply according options $opts$ based on $Hist_{objID}$
else if $r = \text{update}(objId, newVal, opts)$ **then**
$Hist_{objID} := Hist_{objID} \cdot r$
reply according options $opts$ based on $Hist_{objID}$

- Tail fail:
  - This case is handled by removing tail T from the chain and making predecessor T− of T the new tail of the chain.
  - If the tail's predecessor was applying an update when the tail failed, then both pending list and hist list are modified
  - Consistent with transition T3
    - Pending list for objID decreases in size because $Hist^{T}_{objID} \preccurlyeq Hist^{T-}_{objID}$



Coping with Server Failures

- Middle Server fail:
  - The master first informs S's successor S+ of the new chain configuration and then informs S's predecessor S−.
  - The sent list of S-1 is transmitted to S+1.
  - **Inprocess Requests Invariant.** If $i \leq j$ then $Hist^{i}_{objID} = Hist^{j}_{objID} \oplus Sent_i$.



References

[1]. Replication Management using the State Machine Approach, Fred B. Schneider,

https://www.cs.utexas.edu/~lorenzo/corsi/cs380d/papers/statemachines.pdf

[2]. ISIS ToolKit, Ken Birman, https://www.cs.cornell.edu/info/projects/isis/ ; Virtual Synchrony, Ken Birman, https://www.cs.cornell.edu/ken/History.pdf

[3]. CS5414 Distributed Computing course, Lorenzo Alvisi, https://www.cs.cornell.edu/courses/cs5414/2020fa/

[4]. Chain Replication for Supporting High Throughput and Availability, Robbert van Renesse, Fred B. Schneider, OSDI 04.

[5]. Sagar Jha's recitation, TA for CS5412 in 2018,2019,2020, SMR note, https://www.cs.cornell.edu/courses/cs5412/2020sp/recitation_slides/replication.pdf

[6]. Paxos explain: https://ongardie.net/static/raft/userstudy/paxos.pdf

[7]. A good course that explain PAXOS in a simplified way referencing : https://www.youtube.com/watch?v=SRsK-ZXTeZ0

[8] Paxos Made Moderately Complex , Robbert Van Renesse and Deniz Altinbuken, Cornell University