Paxos Made Simple

Leslie Lamport, 1 Nov 2001

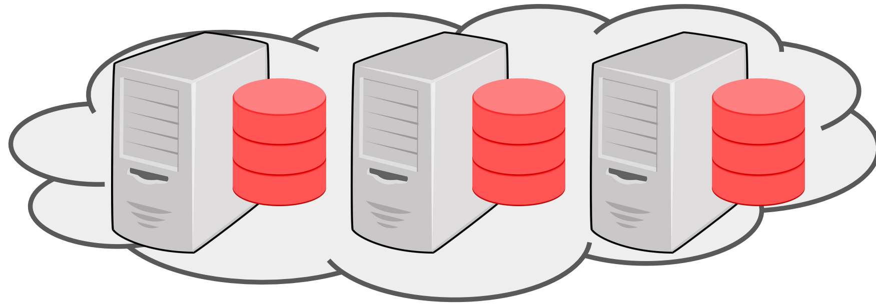
Idea

➔ Fault tolerance on distributed systems

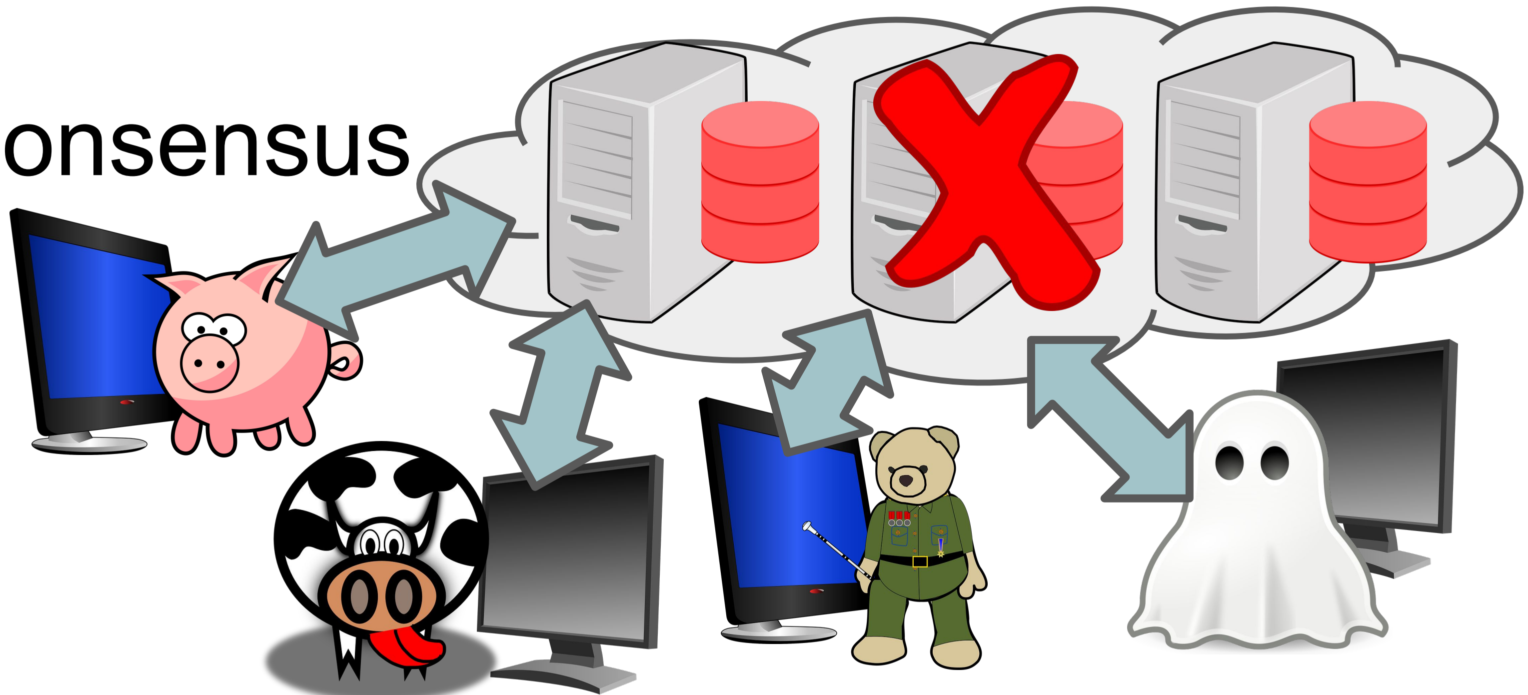
Why even have distributed systems?

How?

- We want a distributed database to come to a consensus.
  - Meaning there is a *consistent state* across multiple servers when dealing with <u>multiple clients</u> that interact with these servers **independently** and **concurrently**



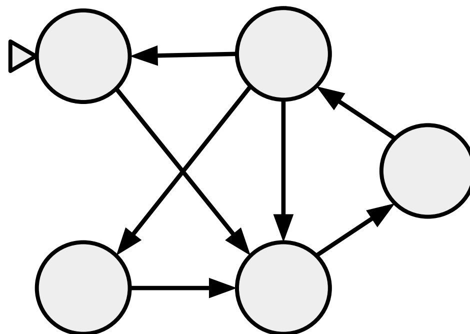
Consensus



- ***If*** computers always agree
  - All computers are equivalent
  - Failure is no problem!
  - Living the dream
    - At the cost of some complexity...

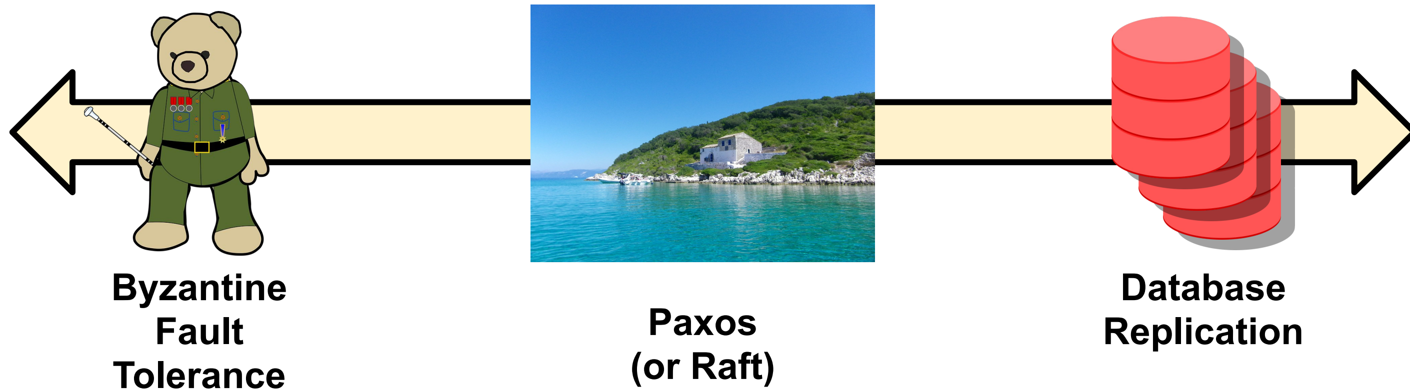
How to use consensus

- Store each clients commands as a sequence (or a log!)
- If those commands were to be executed by each node sequentially, the node would reach the desired state



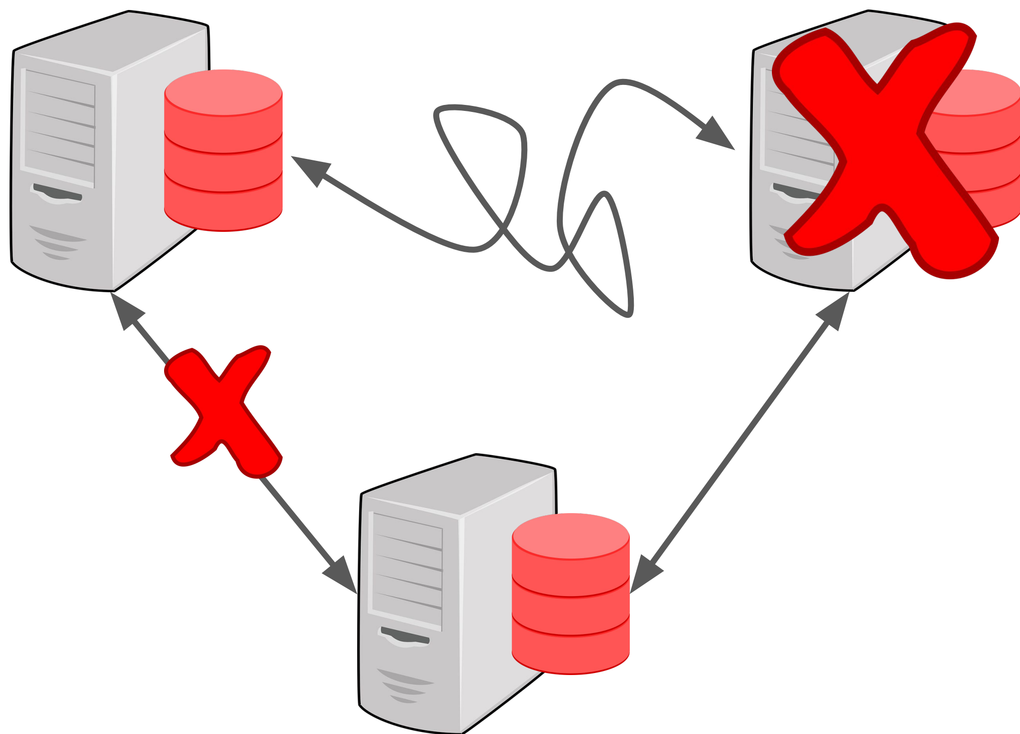
State machine

Alternatives to Paxos



Fault Types:	Byzantine	Fail Stop	
Failover:	Instant		Takes time
Servers:	3m+1	2m+1	m+1
Messages:	Exponential	Linear	

Failure Model



Fail stop,
NOT Byzantine

How does Paxos work?

(Just go to slide 12 for the algorithm)

Three classes of agents

- Proposers
- Acceptors
- Learners

In practice: servers have belong to multiple classes

(this doesn't bother us, the algorithm works regardless)

Paxos ~ Democracy

Paxos works by taking advantage some rules about majorities

"A proposer sends a proposed value to a set 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 large enough?"

"To ensure that only a single value is chosen, we can let a large enough set consist of any majority of the agents. Because any two majorities have at least one acceptor in common, this works if an acceptor can accept at most one value."

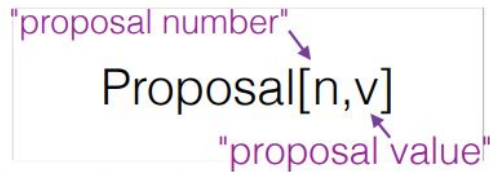
Requirements:

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

P2c: 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.

Proposer	Acceptor	Learner

"Want consensus! How about $v_{default}$?"



PrepareRequest[n]

→

If n >= max rx'd PrepReq,

"New prepreq! Here is my highest Proposal's value."

ResponseToPrepareRequest[Proposal[m,w] or None]

←

If rx'd a majority,
v = max(rx'd Proposals).v or $v_{default}$

AcceptRequest[Proposal[n,v]]

→

"This Proposer gained majority support from As."

Decision[Proposal[n,v]]

→

If rx'd a majority,
value = v

For this algorithm to work we have to strengthen **P1** to:

P1a: An acceptor can accept a proposal numbered n **iff** it has not responded to a *prepare* request having a number greater than n.

In other words: “An acceptor can respond to an *accept* request, accepting the proposal, iff it has not promised not to.”

With this, we are trying to eliminate scenarios where 2 requests come in “reverse order”, or scenarios where something “happened” in between prepare requests.

Example

Burgers or Pizza?

or ?

- Band director went home
- People easily distracted
- No fun if group splits up
- Hungry: must come to a decision fast!
- Yelling fails. Use person-to-person communication.

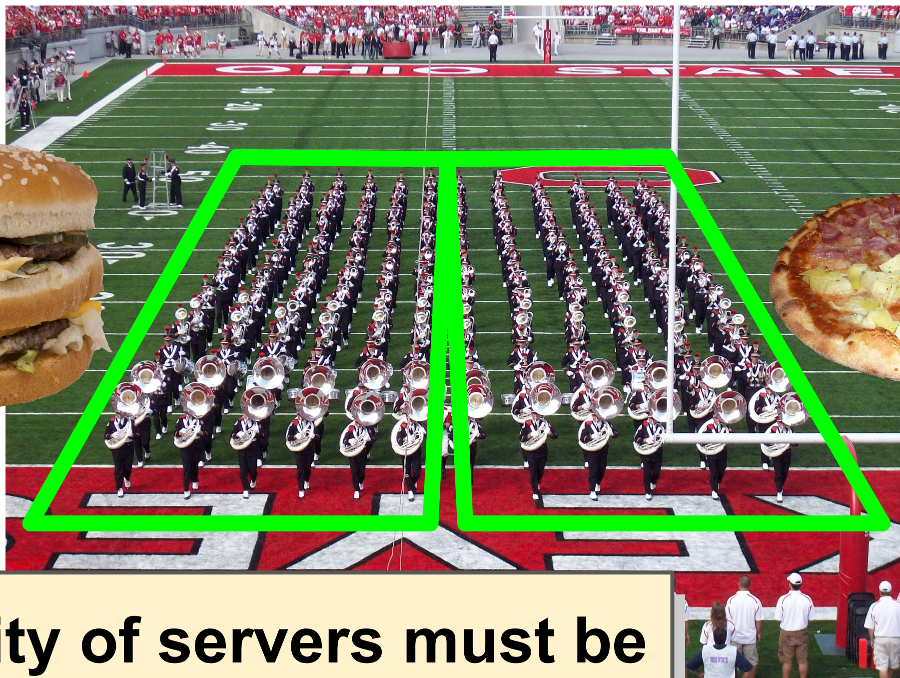
Want to achieve *consensus*

Burgers or Pizza?

1. What's happening?
2. Let's go for burgers!

Paxos: *almost* this simple

Majority wins!



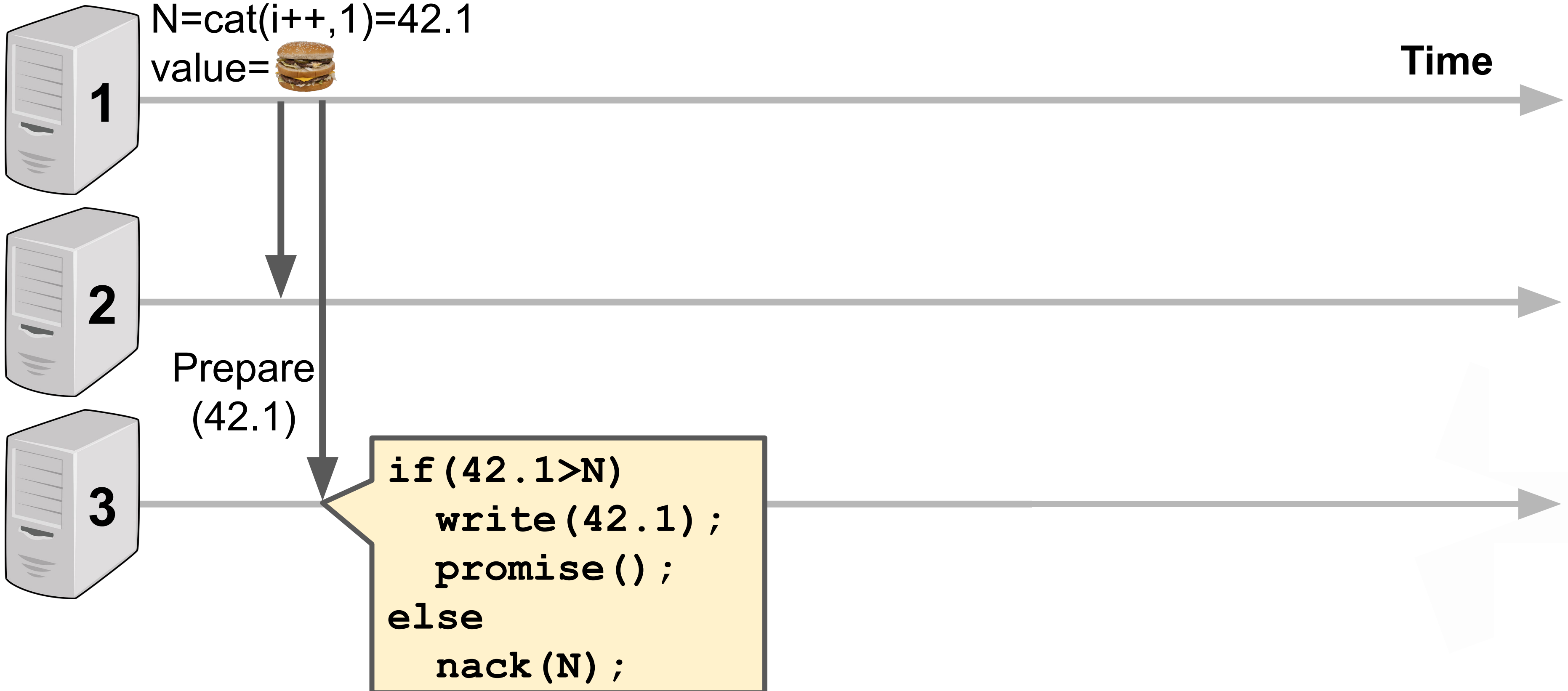
1. What's happening?
   - Need to ask majority
2. Let's go for burgers!
   - Majority must agree

Majority of servers must be up for Paxos to terminate.

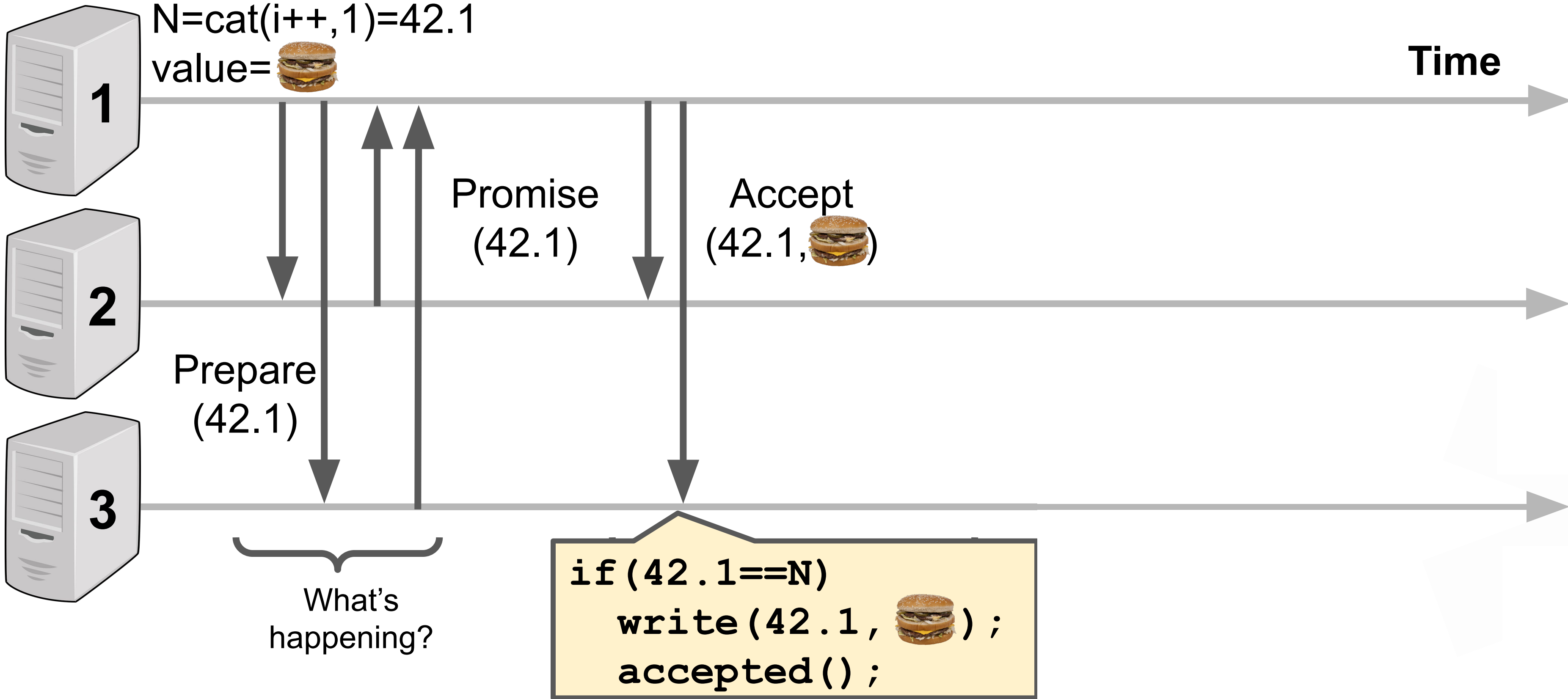
Need 2m+1 servers to tolerate m failures

Basic Paxos

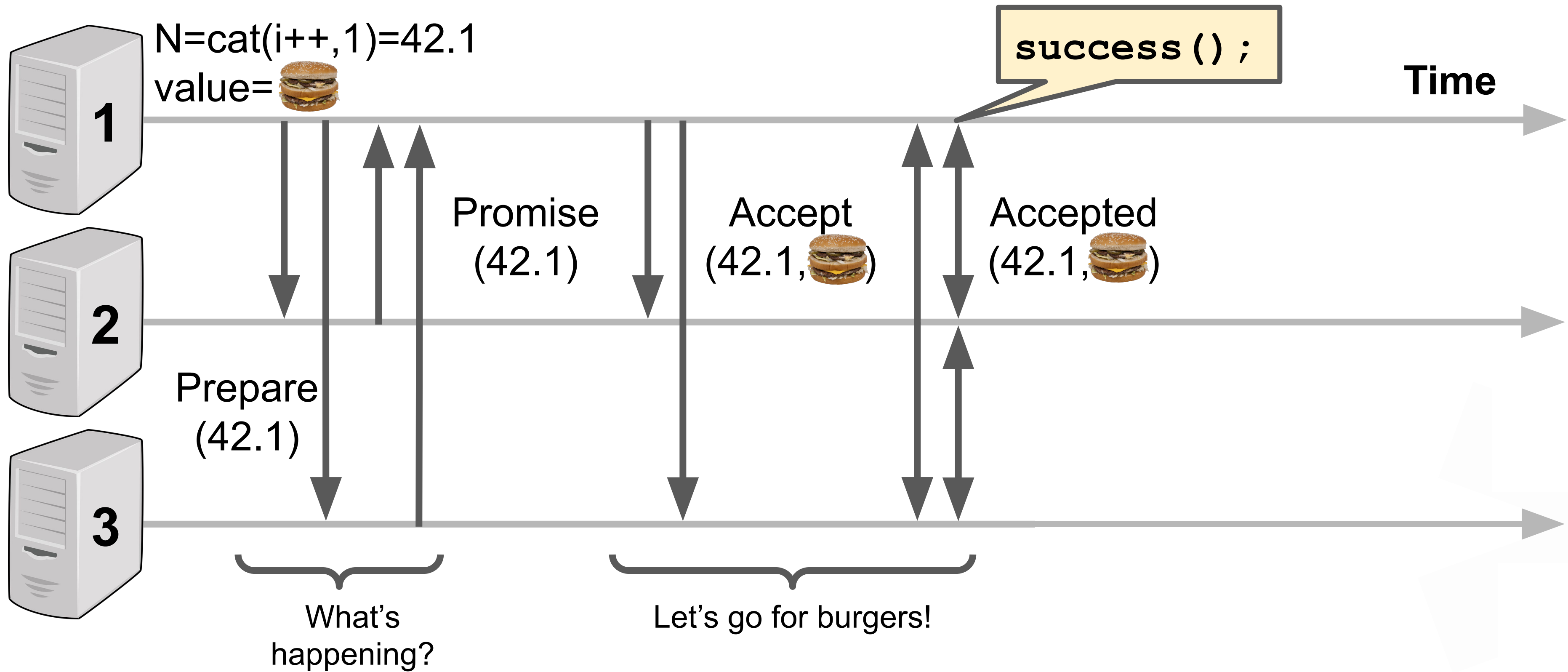
Basic Paxos



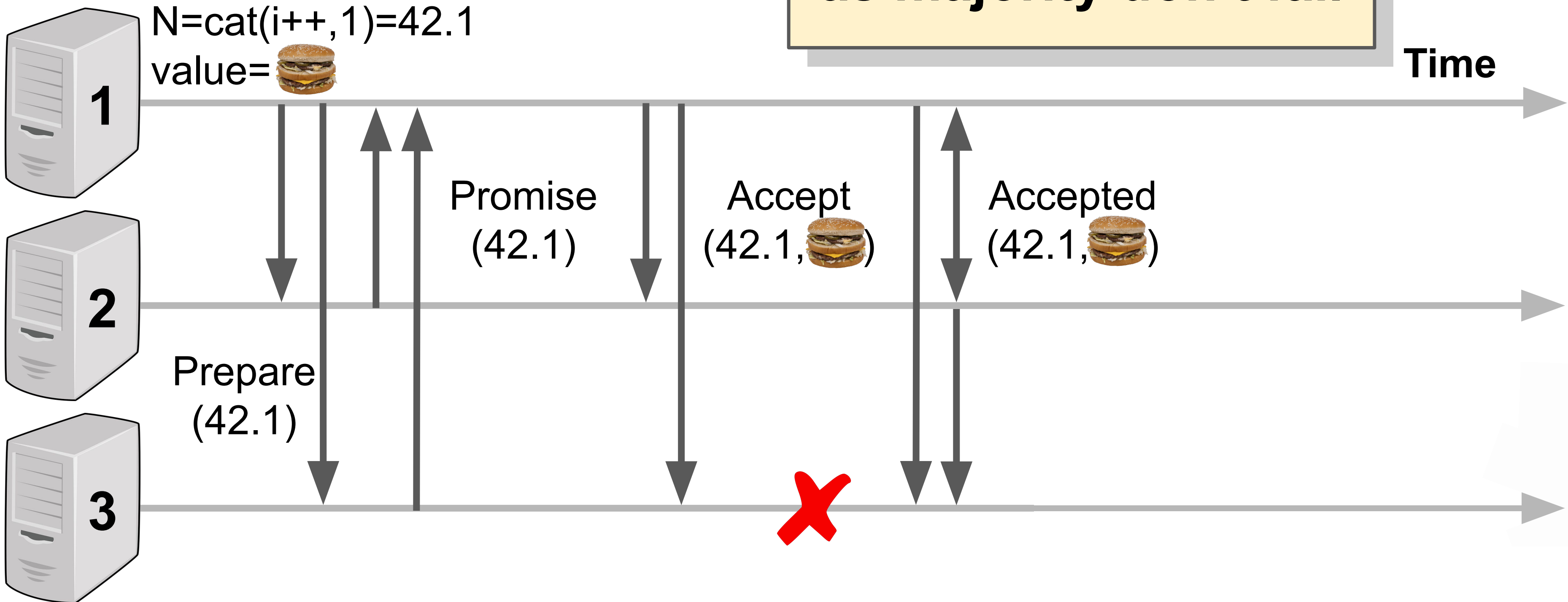
Basic Paxos



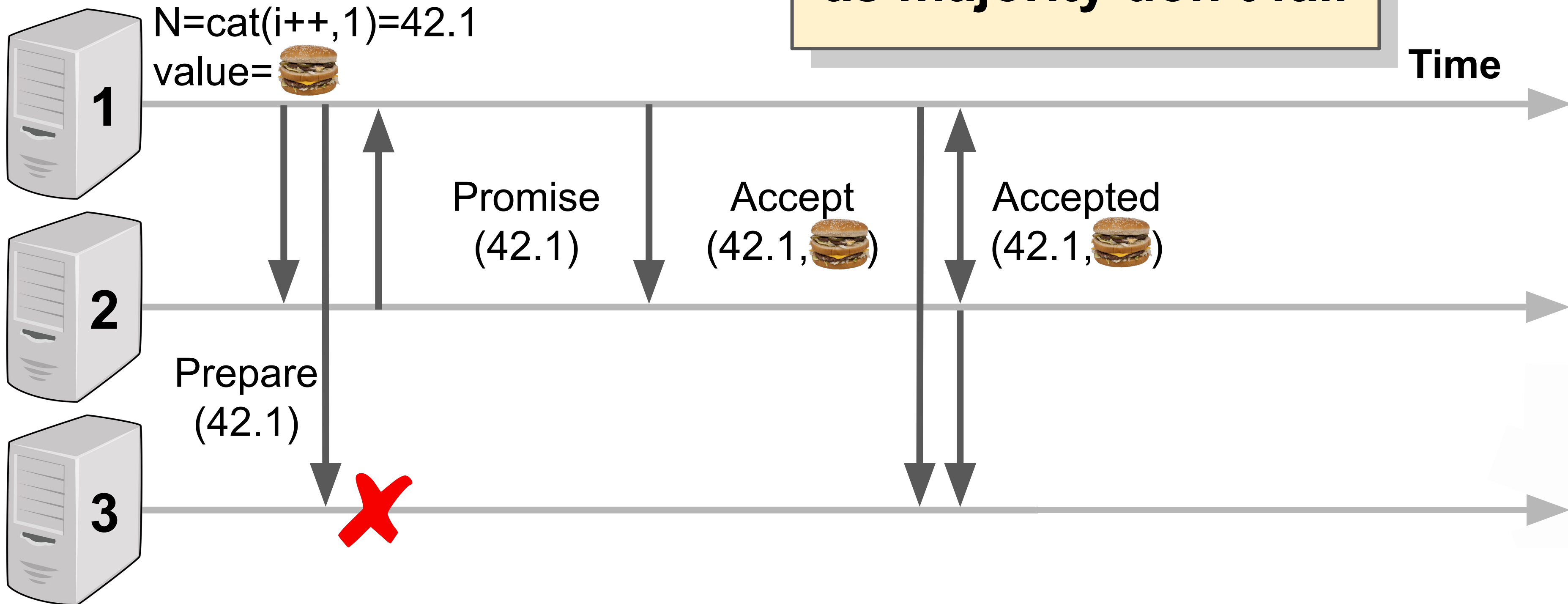
Basic Paxos



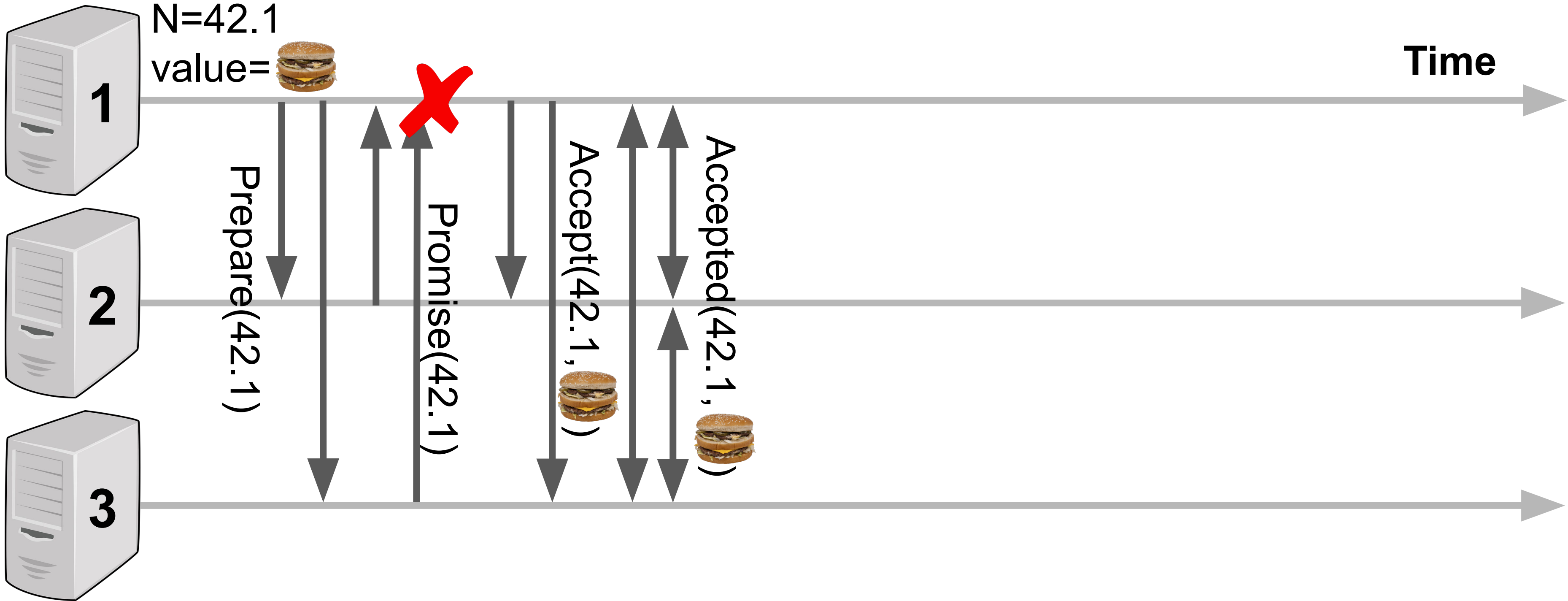
Failures: Acceptor



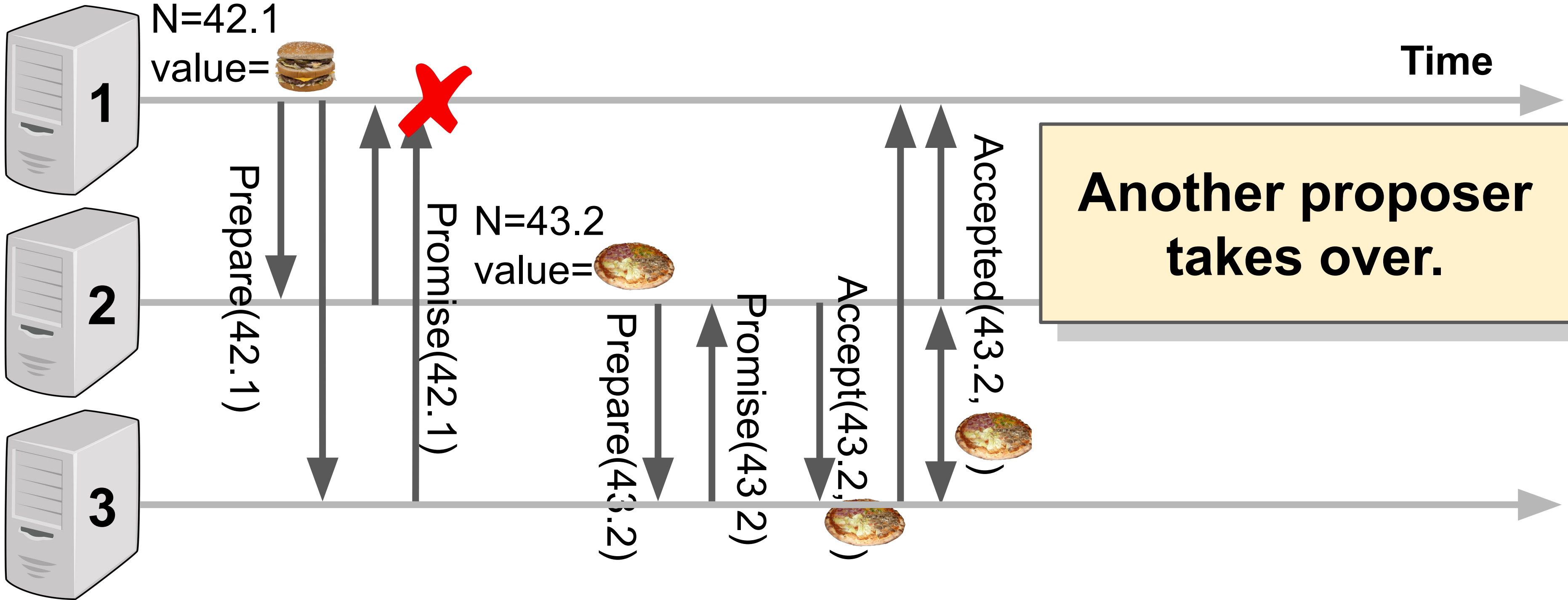
Failures: Acceptor



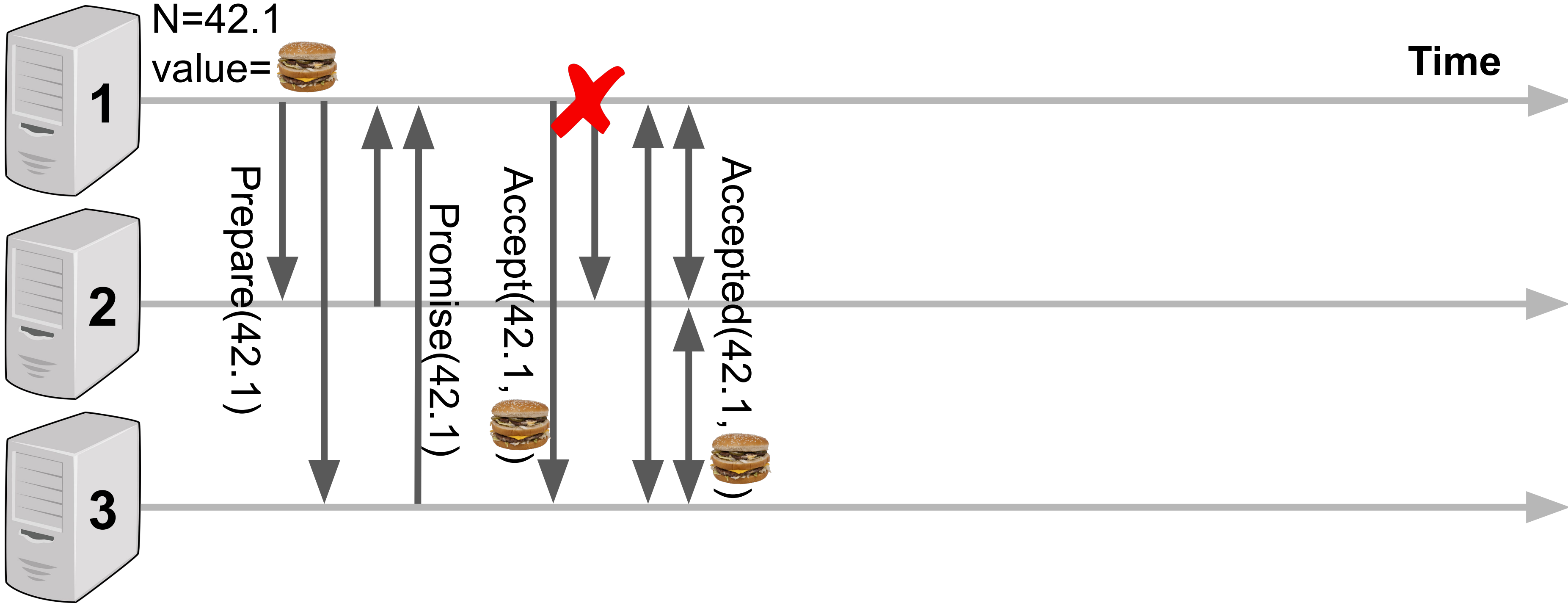
Failures: Proposer in Prepare Phase



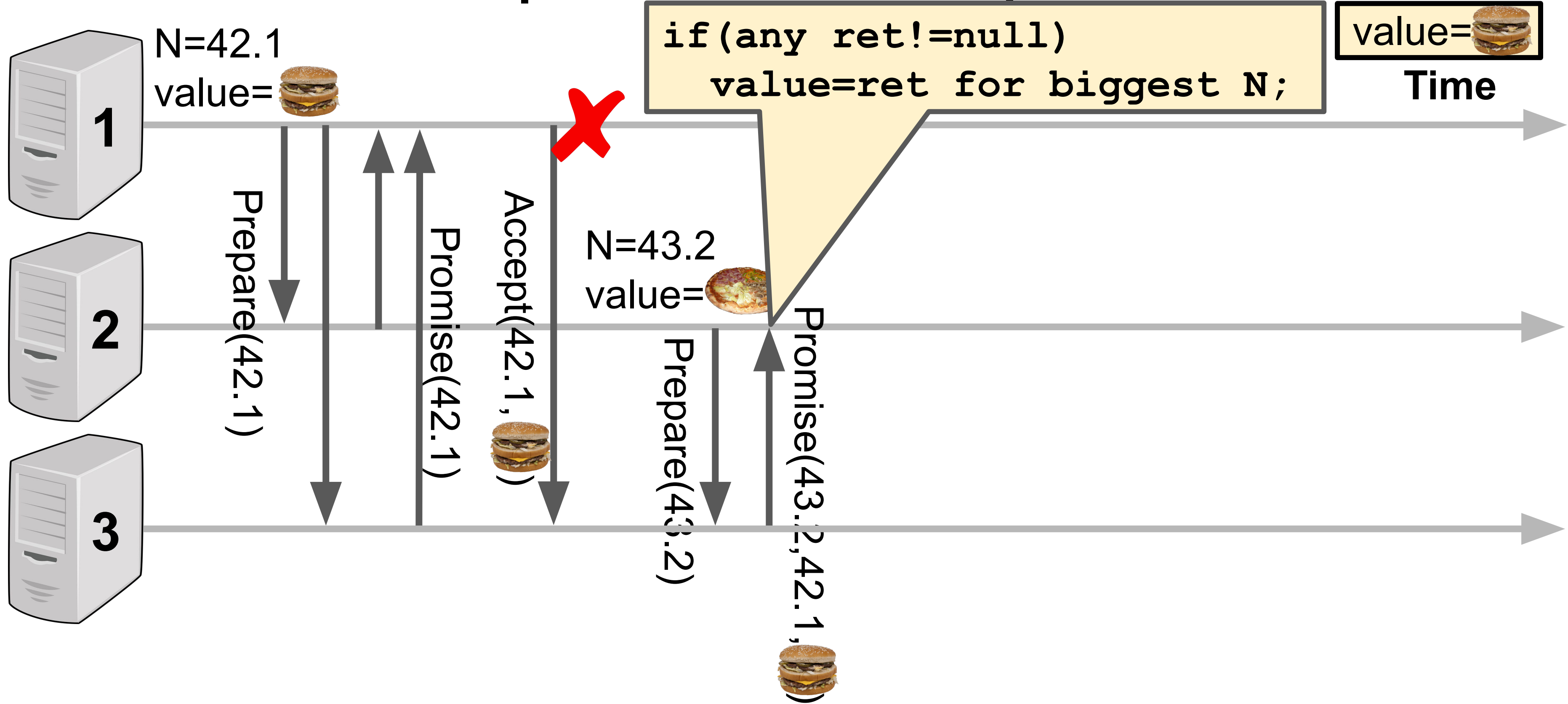
Failures: Proposer in Prepare Phase



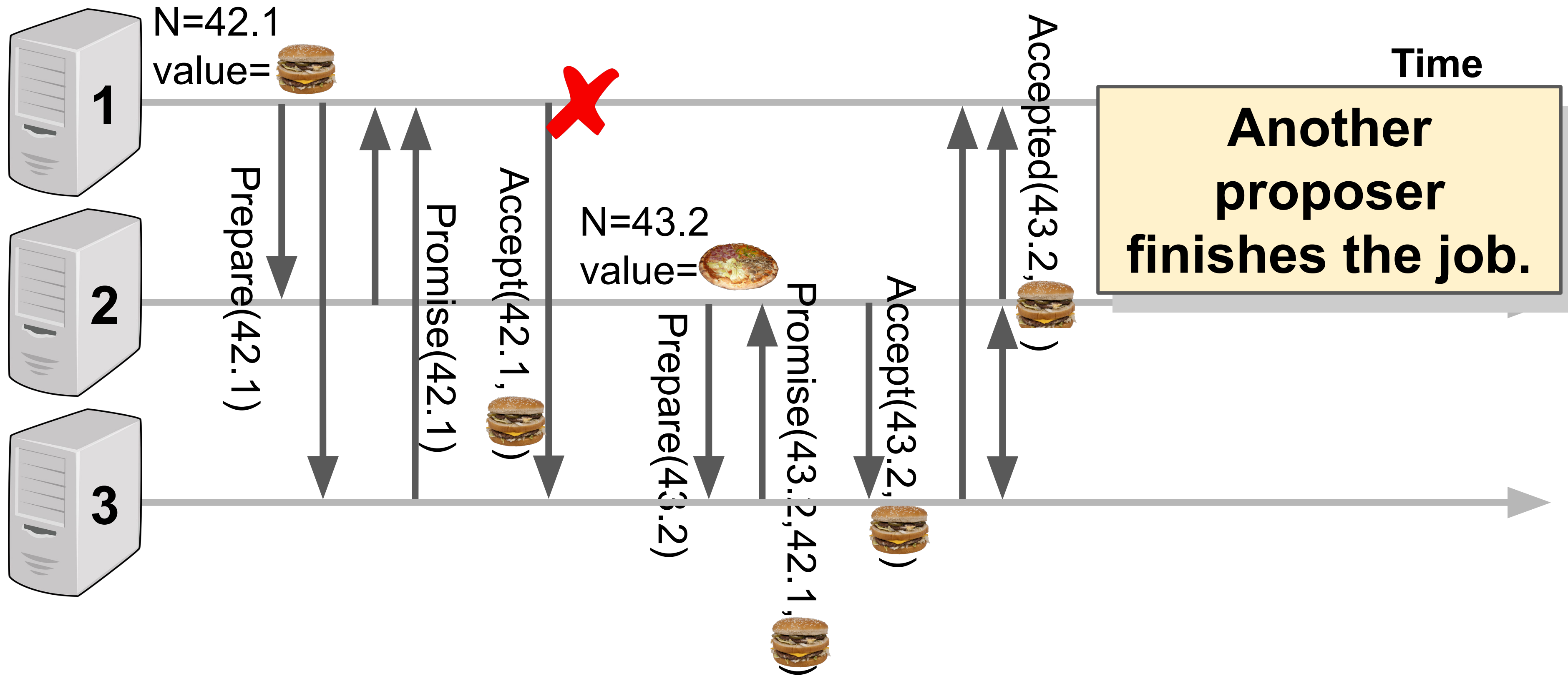
Failures: Proposer in Accept Phase



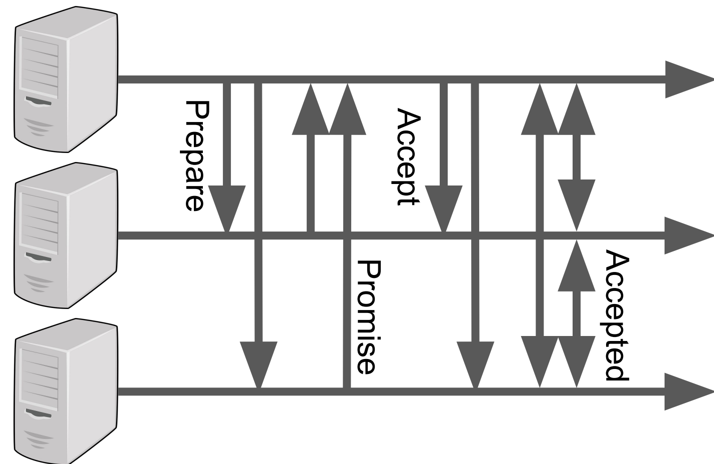
Failures: Proposer in Accept Phase



Failures: Proposer in Accept Phase



What could go wrong?



- One proposer
  - One or more acceptors fails
    - Still works as long as majority are up
  - Proposer fails in prepare phase
    - No-op; another proposer can make progress
  - Proposer fails in accept phase
    - Another proposer overwrites
    - Another proposer finishes the job
- Two or more simultaneous proposers
  - A bit more complex…
  - Can livelock, avoid with exponential backoff or with leader election

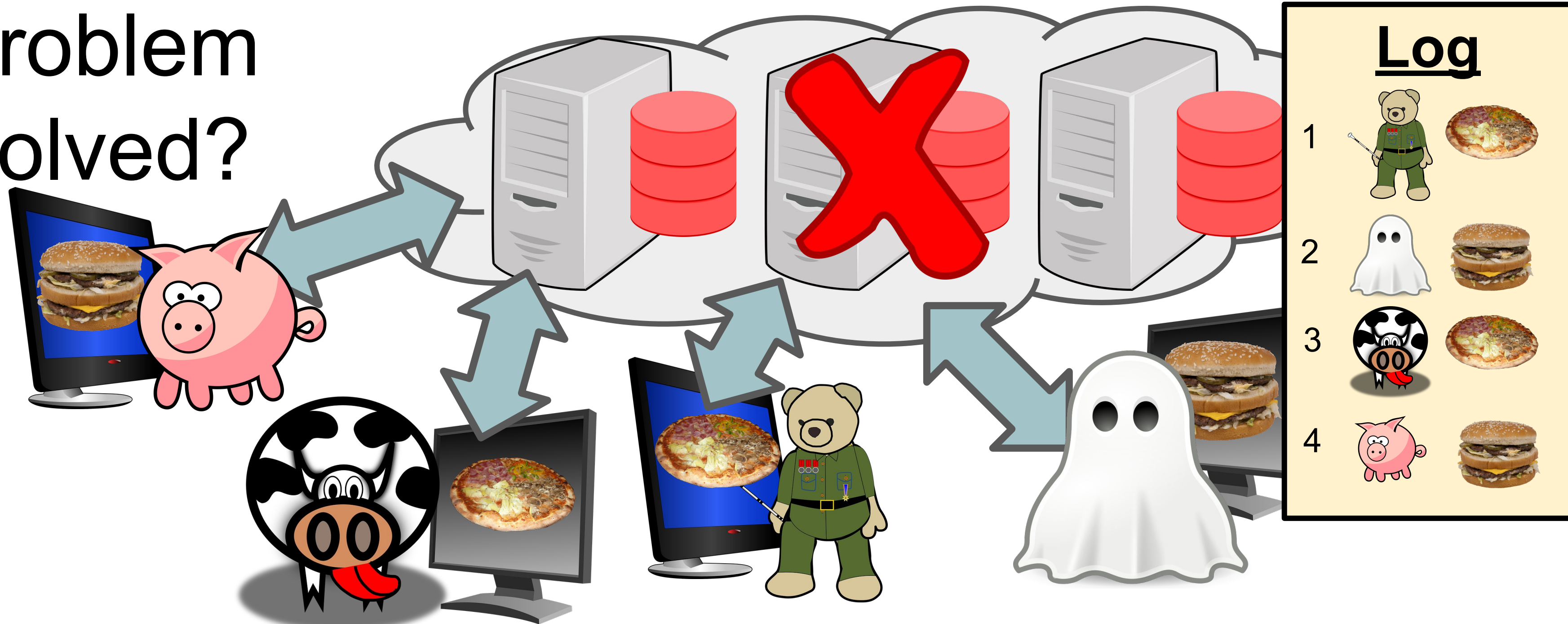
Leader Election

- Can use Paxos itself to elect a leader!
  - ...but what if it livelocks?

- A reliable algorithm for electing a proposer must use either randomness or real time — for example, by using timeouts

“However, safety is ensured regardless of the success or failure of the election.”

Problem Solved?



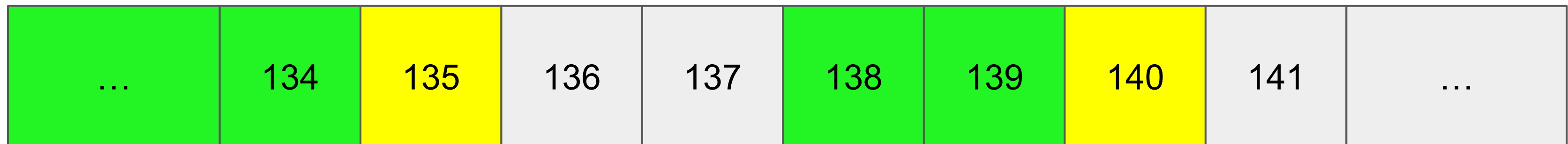
- Paxos lets you make *one choice*
- Multi-Paxos needed for a real system
- Build a *log* of choices

Paxos in the real world

- Creating a log of agreements
- “Propositions” are made to the leader
- Leader decides their order
  - Implementation dependent
  - Leader can issue k rounds of Paxos at a time
- When leader fails, another node takes over.
  - This can be implemented with timers and frequent checks

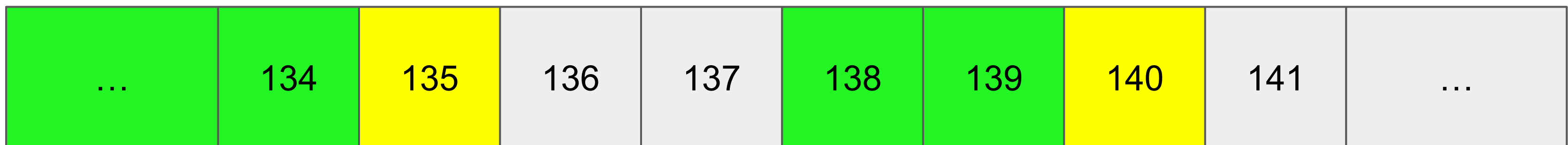
Is there a problem?

- Let’s say a leader is running and a consensus has been reached in runs 1-134, 138-139.
    - This means both phases of the algorithm were successful and logs have been made by the learners
- Let’s also assume that the leader has also sent accept requests for runs 135-137 and 140+, but only 135 and 140 were accepted by a majority of acceptors, but for some reason leader fails and no logs are made.

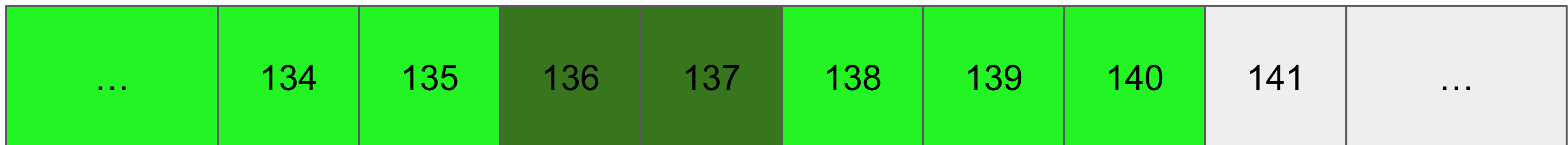


- When new leader is elected, being a learner in all instances of the consensus algorithm, should know most of the values that have already been chosen.
  - Suppose he knows 1-134, 138-139

- He will execute phase 1 of the algorithm for runs 135-137 and 140+.
  - 135 and 140 will return with promised values
  - Others will return empty promises.



- The leader will make sure 135 and 140 runs are completed with their previous promises.
- For operations 136 and 137, there is no way to recover what operations the previous leader wanted to establish, so a special *no-op* operations is send in the 2nd phase of Paxos.
- For operations 141+, the new leader can work normally, as none of them were established by the previous leader.



Conclusions

- Consensus in a fail stop environment can be solved!
- Basic Paxos is not that hard
  - A full featured implementation is notoriously complex
- *Raft* is used way more in practice