COEN6731 Distributed Software Systems

Week 2: Coordination, Agreement, and Paxos

Gengrui (Edward) Zhang, PhD
Web: gengruizhang.com

Today's outline

The consensus problem

Network assumptions

Failure assumptions

Paxos

The consensus problem

Let’s go to the beach!

Let’s go get some food!

Let’s go see a movie!

The consensus problem

Let’s go to the beach!

Let’s go get some food!

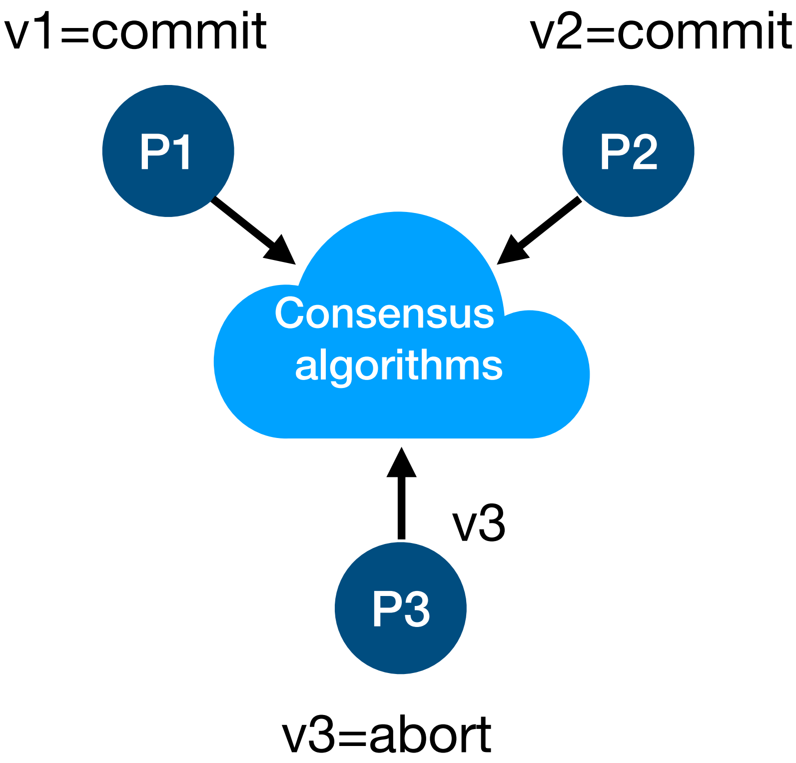
Let’s go see a movie!

What’s important in reaching agreement?

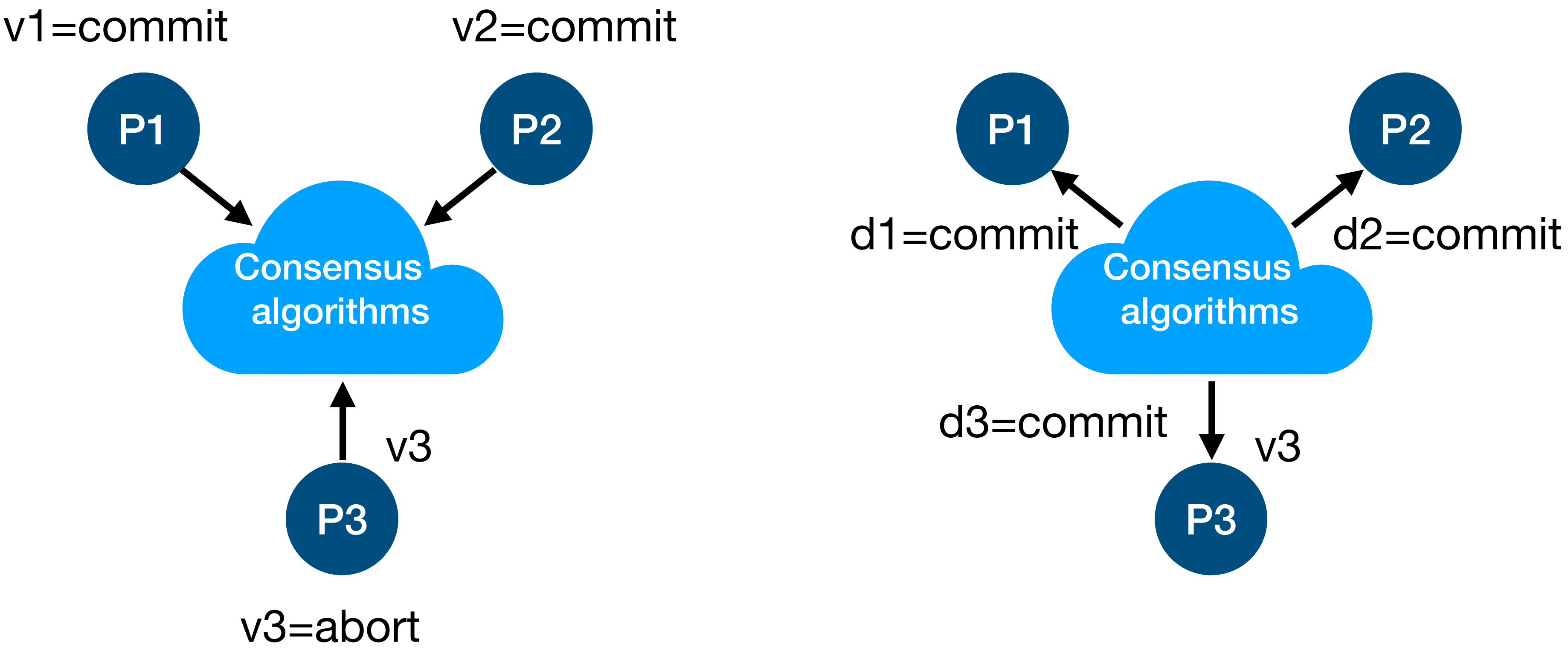
1. Agree on the activities
2. Agree on the order of activities

The happened-before relation of activities

Consensus in distributed systems



Consensus in distributed systems



Formally, the consensus problem

- To reach consensus, every process p_i begins in the **undecided** state and **proposes** a single value v_i, drawn from a set D $(i = 1,2,\ldots,N)$.
- Processes communicate with one another, exchanging values.
- Each process then sets the value of a **decision variable**, d_i.
- After that, each process enters the **decided** state, where d_i $(i = 1,2,\ldots,N)$ do not change

Formally, the consensus problem

- To reach consensus, every process p_i begins in the **undecided** state and **proposes** a single value v_i, drawn from a set D $(i = 1,2,\ldots,N)$.
- Processes communicate with one another, exchanging values.
- Each process then sets the value of a **decision variable**, d_i.
- After that, each process enters the **decided** state, where d_i $(i = 1,2,\ldots,N)$ do not change

In short, all correct processes commit the **same value** in the **same order**

Today's outline

The consensus problem

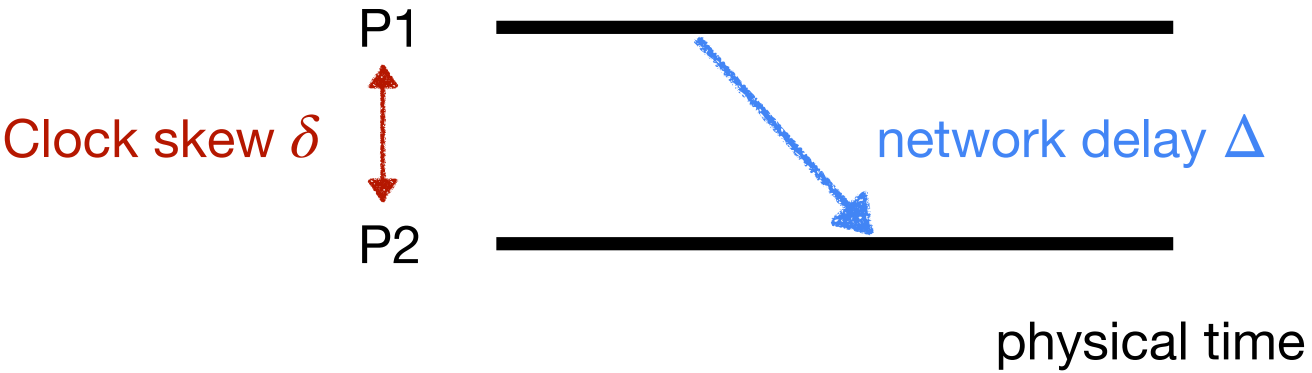
Network assumptions

Failure assumptions

Paxos

System model: network synchrony

- Synchronous
- Asynchronous
- Partially synchronous

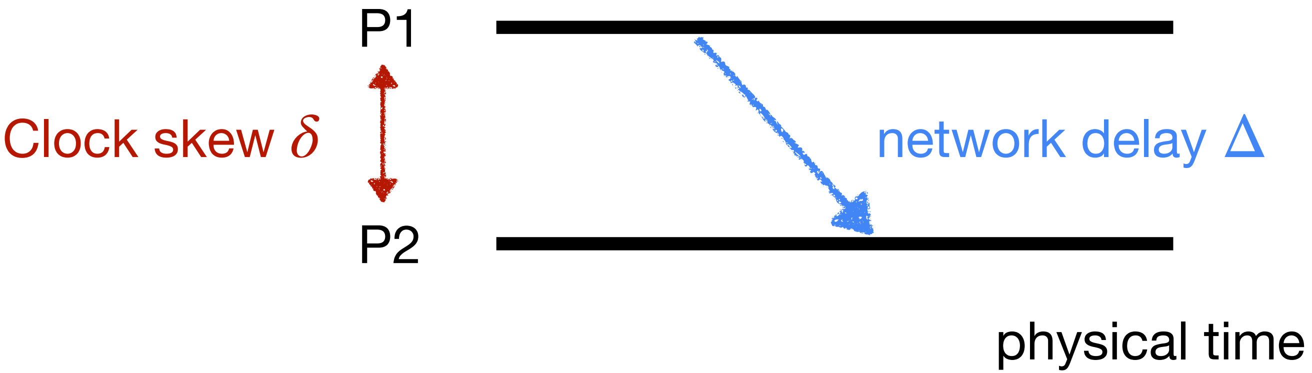


System model: network synchrony

- Synchronous
- Asynchronous
- Partially synchronous

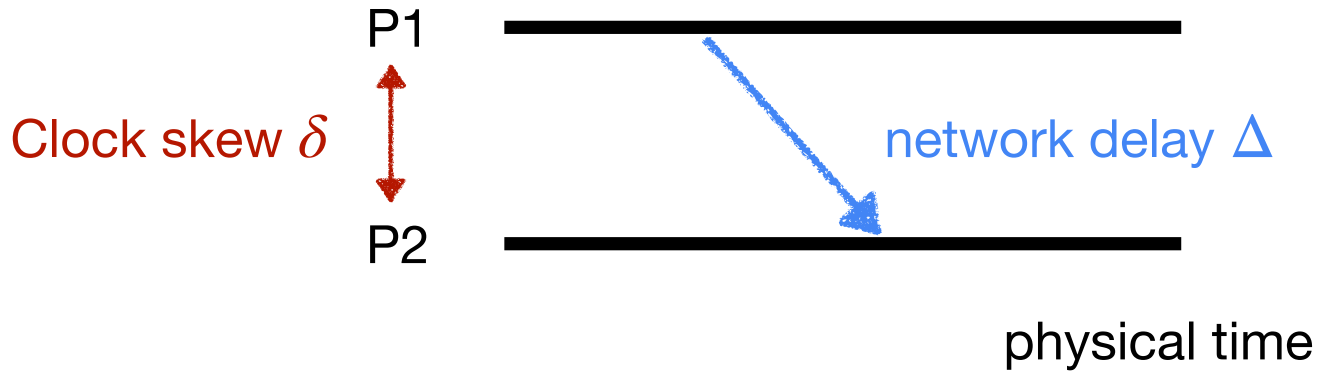
Synchronous:

Both δ and Δ have a fixed upper bound



System model: network synchrony

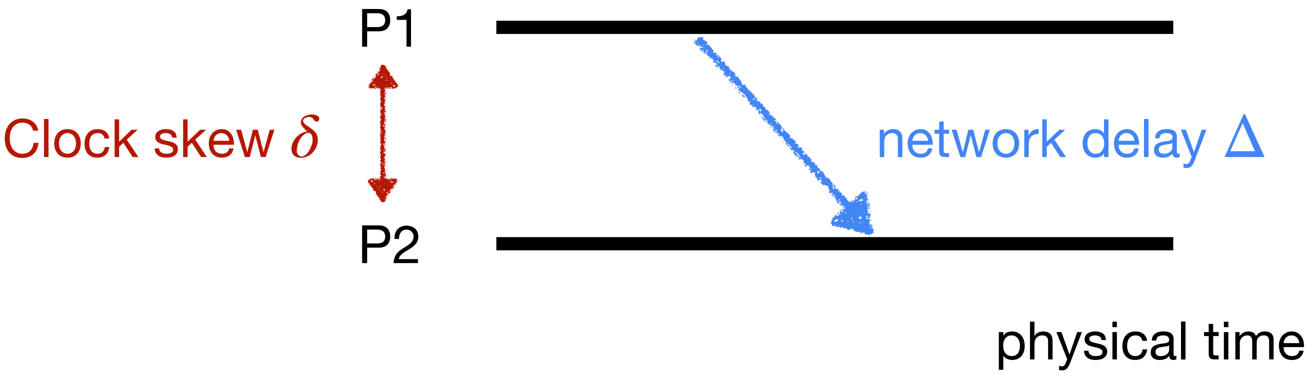
- Synchronous
- Asynchronous
- Partially synchronous



System model: network synchrony

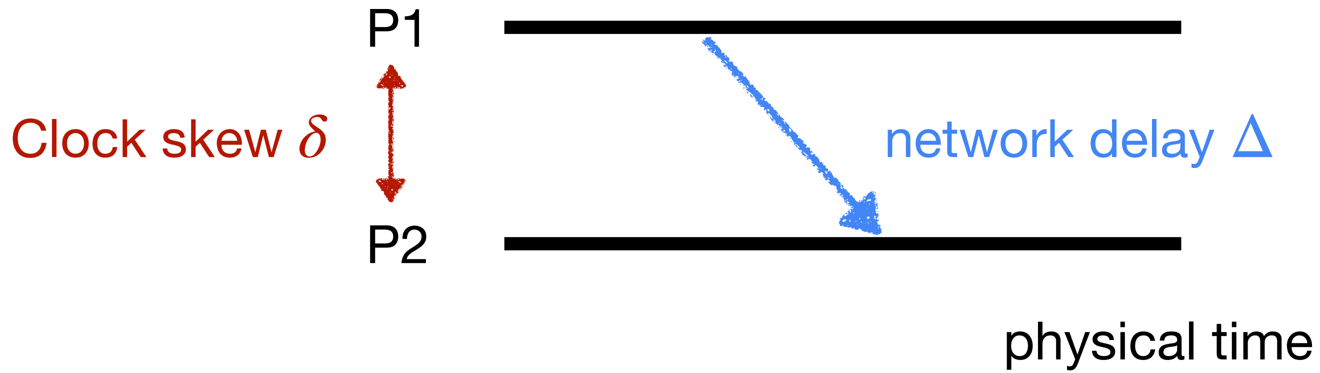
- Synchronous
- Asynchronous
- Partially synchronous

Asynchronous:
No fixed upper bound for message delivery or clock skew (i.e., δ does not exist, or Δ does not exist)



System model: network synchrony

- Synchronous
- Asynchronous
- Partially synchronous

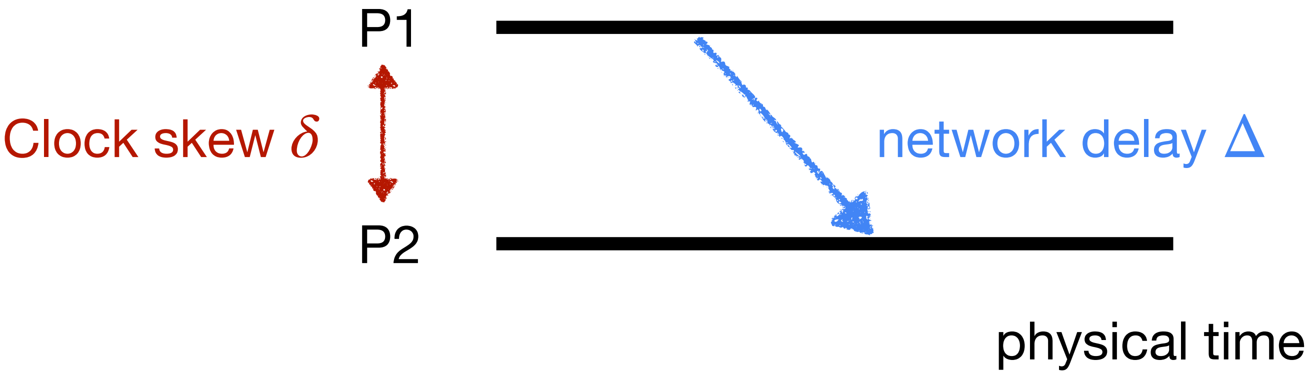


System model: network synchrony

- Synchronous
- Asynchronous
- Partially synchronous

Partially synchronous:
Communication among servers can have a global stabilization time (GST), unknown to processors.

1. δ and Δ both exist but unknown, or
2. δ and Δ are known after GST



chalk talk

Let’s design a simple consensus algorithm

- Assume processes cannot fail
- Synchronous network
- We’d like to have:

Termination: Eventually each correct process sets its decision variable

Agreement: Decision value of all correct processes is the same; if p_i and p_j are correct and ahem entered the decided state, then $d_i = d_j (i, j = 1,2,\ldots,N)$

Integrity/Validity: If the correct processes all proposed the same value, then any correct process in the decided state has chosen that value.

Service properties

Termination: Eventually each correct process sets its decision variable

Agreement: Decision value of all correct processes is the same; if p_i and p_j are correct and ahem entered the decided state, then $d_i = d_j (i, j = 1,2,\ldots,N)$

Integrity/Validity: If the correct processes all proposed the same value, then any correct process in the decided state has chosen that value.

Service properties

Termination: Eventually each correct process sets its decision variable

Agreement: Decision value of all correct processes is the same; if p_i and p_j are correct and ahem entered the decided state, then $d_i = d_j (i, j = 1, 2, \ldots, N)$

Integrity/Validity: If the correct processes all proposed the same value, then any correct process in the decided state has chosen that value.

Something cannot happen

Safety

No two correct nodes decide differently

Service properties

Termination: Eventually each correct process sets its decision variable

Agreement: Decision value of all correct processes is the same; if p_i and p_j are correct and ahem entered the decided state, then $d_i = d_j (i, j = 1,2,\ldots,N)$

Integrity/Validity: If the correct processes all proposed the same value, then any correct process in the decided state has chosen that value.

Something cannot happen

Safety

No two correct nodes decide differently

Liveness

Nodes eventually decide

Something must happen

Today's outline

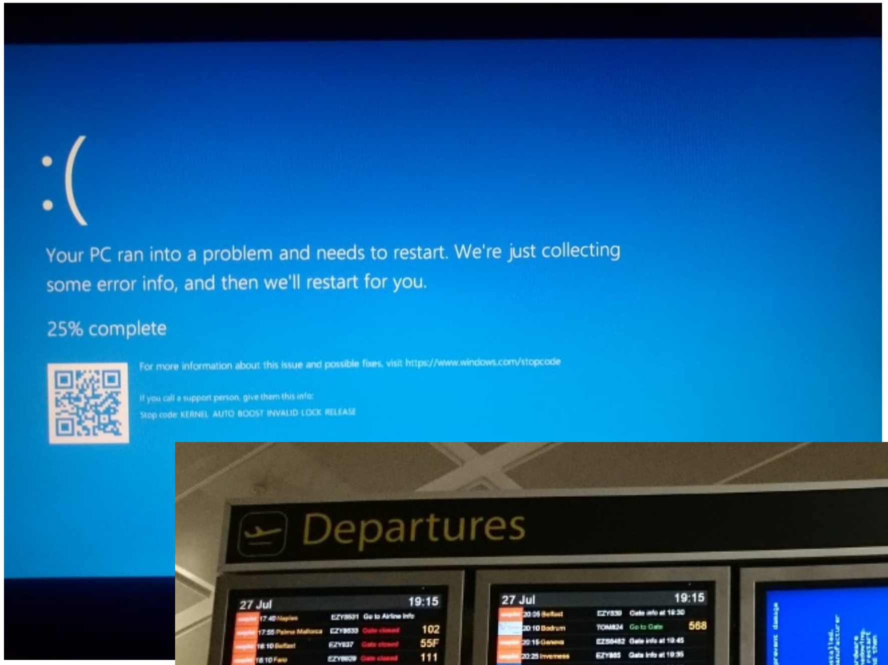
The consensus problem

Network assumptions

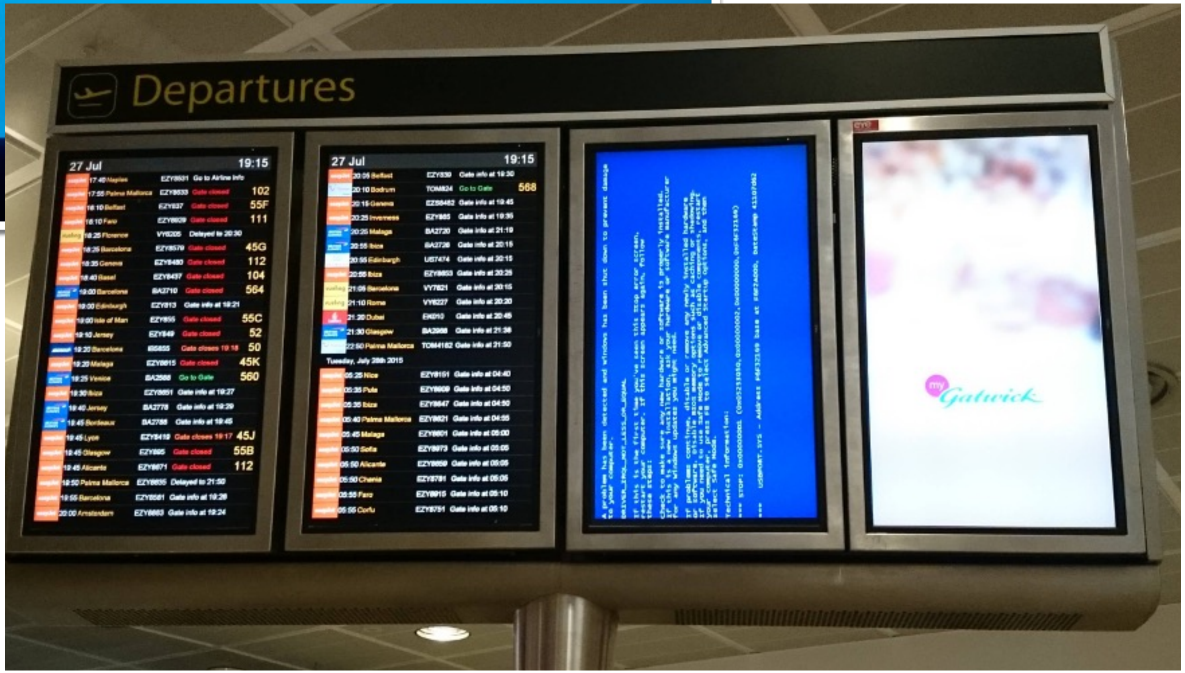
Failure assumptions

Paxos

Faults…



:(
Your PC ran into a problem and needs to restart. We're just collecting some error info, and then we'll restart for you.
25% complete
For more information about this issue and possible fixes, visit https://www.windows.com/stopcode
If you call a support person, give them this info:
Stop code: KERNEL AUTO BOOST INVALID LOCK RELEASE

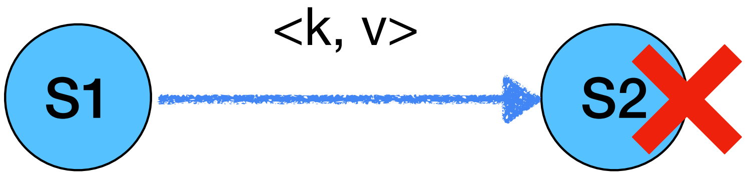


Departures
27 Jul
19:15
27 Jul
19:15

Coca-Cola

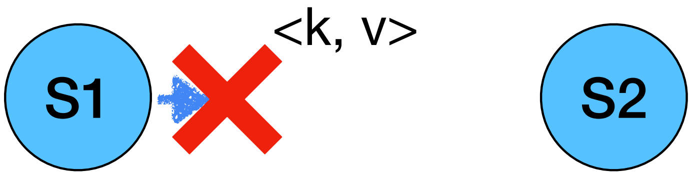
Family of faults

- Crash faults
- Omission faults
  - Send omission
  - Receive omission
- Timing faults



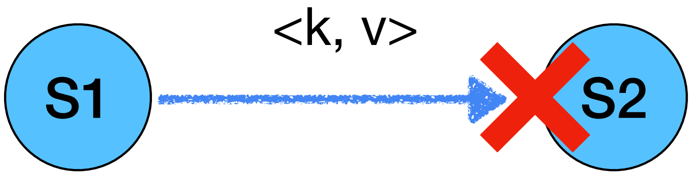
Family of faults

- Crash faults
- Omission faults
  - Send omission
  - Receive omission
- Timing faults



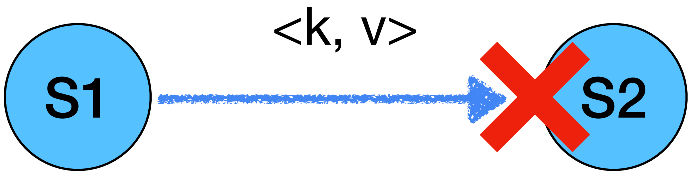
Family of faults

- Crash faults
- Omission faults
  - Send omission
  - Receive omission
- Timing faults



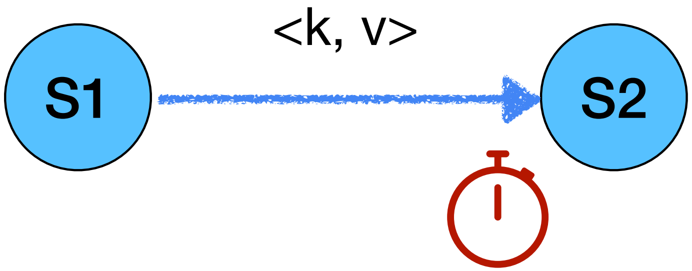
Family of faults

- Crash faults
- Omission faults
  - Send omission
  - Receive omission
- Timing faults



Family of faults

- Crash faults
- Omission faults
  - Send omission
  - Receive omission
- Timing faults

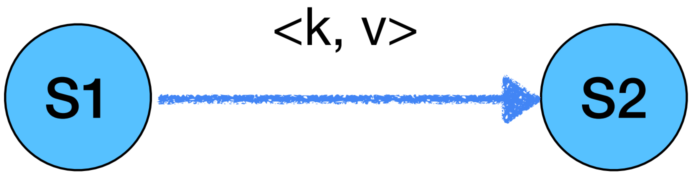



```
if timer.timeout:
   proceed without v
```

Family of faults

- Crash faults
- Omission faults
  - Send omission
  - Receive omission
- Timing faults

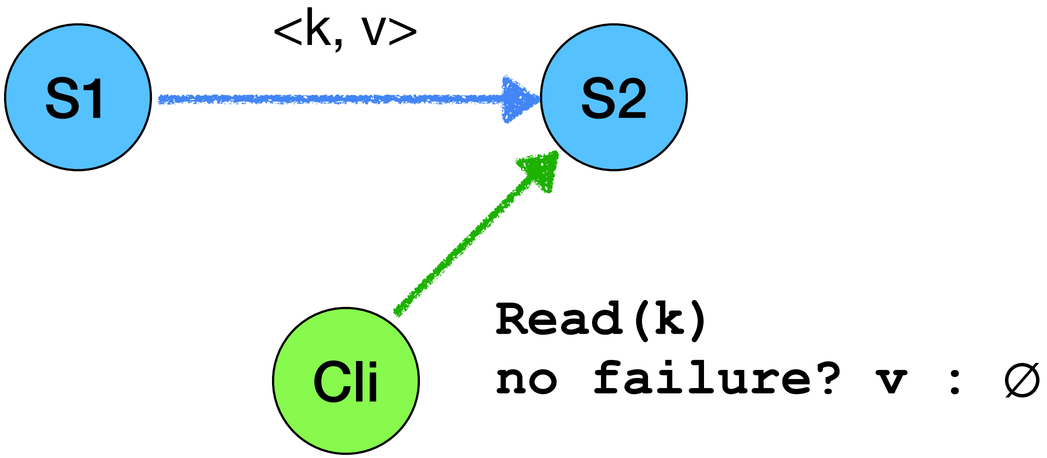
Worst thing that can happen:
S2 does not have the value



Family of faults

- Crash faults
- Omission faults
    - Send omission
    - Receive omission
- Timing faults

Worst thing that can happen:
S2 does not have the value

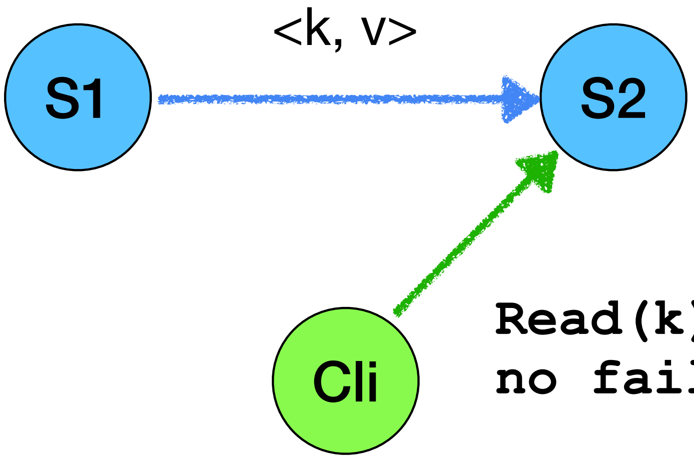


Family of faults

Benign faults

- Crash faults
- Omission faults
  - Send omission
  - Receive omission
- Timing faults

Worst thing that can happen:
S2 does not have the value

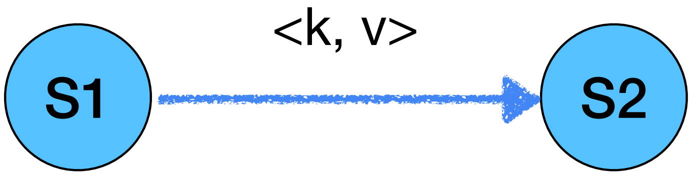



```
Read(k)
no failure? v : ∅
```

Family of faults

Benign faults

- Crash faults
- Omission faults
  - Send omission
  - Receive omission
- Timing faults



Byzantine faults

- Any arbitrary behaviour, e.g.,
  - Stop responding
  - Send erroneous values

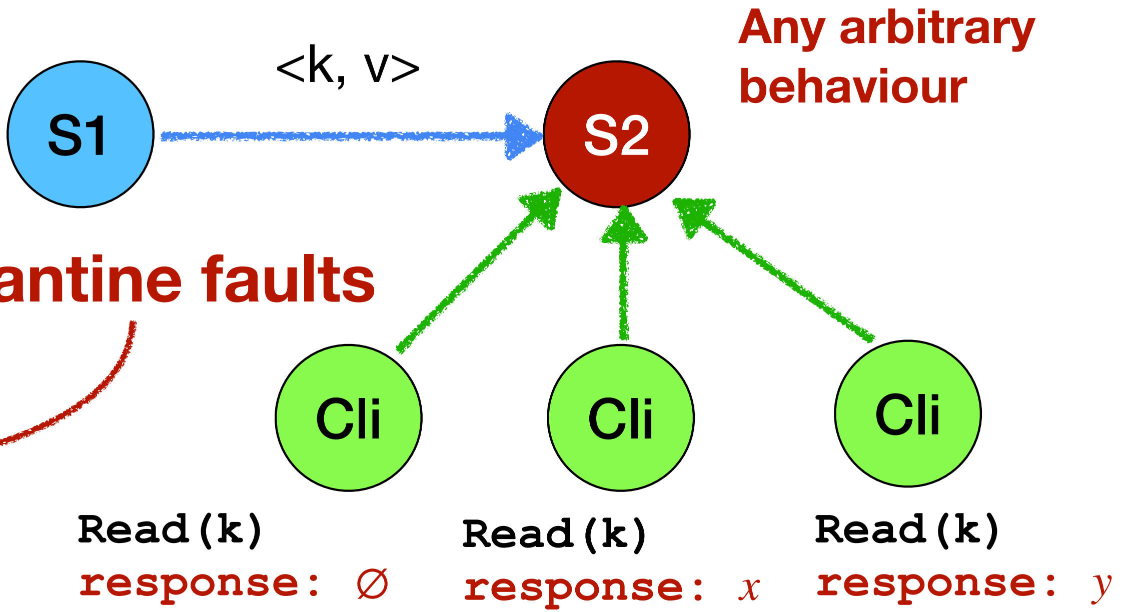
Family of faults

Benign faults

- Crash faults
- Omission faults
  - Send omission
  - Receive omission
- Timing faults

Byzantine faults

- Any arbitrary behaviour, e.g.,
  - Stop responding
  - Send erroneous values

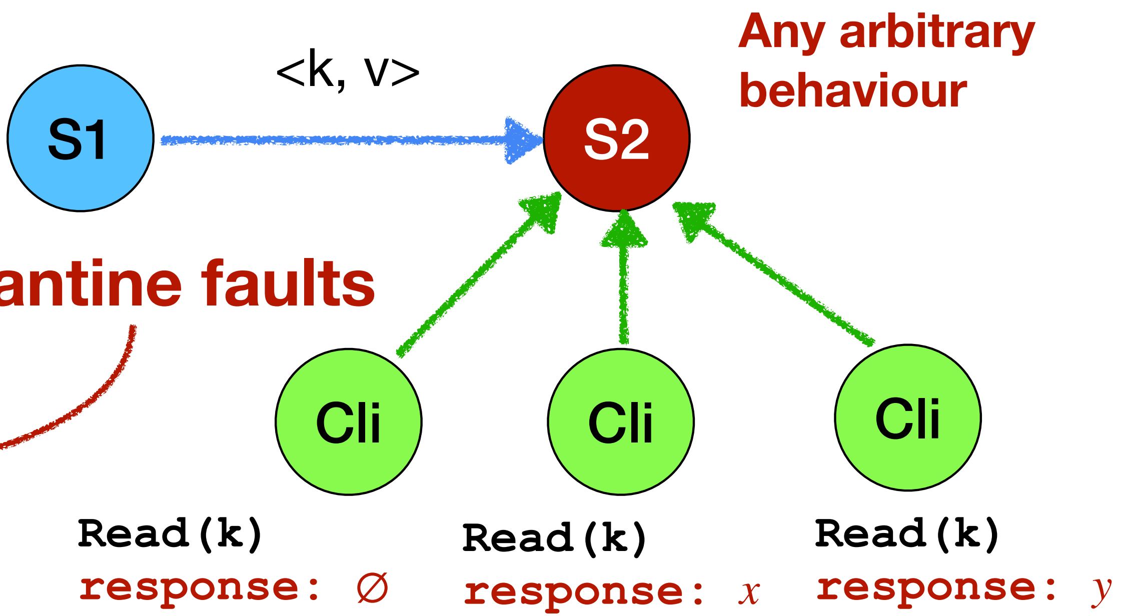


Family of faults

Benign faults

- Crash faults
- Omission faults
  - Send omission
  - Receive omission
- Timing faults

Worst thing that can happen:
Any behaviour that can do the most harm



Byzantine faults

- Any arbitrary behaviour, e.g.,
  - Stop responding
  - Send erroneous values

Family of faults: summary

Fault tolerance

Benign faults

- Crash faults
- Omission faults
    - Send omission
    - Receive omission
- Timing faults

- Crash fault tolerance (CFT) algorithms
    - Paxos, ViewStamped Replication, Raft [ATC'13]
- Applications (everything distributed):
    - File systems: HDFS and GFS
    - Databases: Google Spanner and etcd
    - Coordination: Chubby and Zookeeper

Byzantine faults

- Any arbitrary behaviour, e.g.,
    - Stop responding
    - Send erroneous values

- Byzantine fault tolerance (BFT) algorithms
    - PBFT [OSDI'99], HotStuff [PODC'21], Pompe [OSDI'22]
- Applications (safety critical):
    - Unreliable hardware: Airplanes
    - Blockchains: Facebook Diem, Microsoft CCF

Algorithms we will talk about

- Paxos: <- Today's topic
  - How to choose a value under **benign failures**
- Raft [ATC'14]:
  - How to replicate log under **benign failures**?
- PBFT [OSDI'99]:
  - How to replicate log under **Byzantine (arbitrary) failures**?

Today's outline

The consensus problem

Network assumptions

Failure assumptions

Paxos

Paxos

- Papers:
  - Lamport L. The part-time parliament[J]. ACM Transactions on Computer Systems (TOCS), 1998, 16(2): 133-169.
  - Lamport L. Paxos made simple[J]. ACM Sigact News, 2001, 32(4): 18-25.
- System model
  - Asynchronous
  - CFT: tolerating benign faults (non-Byzantine)

Fundamental #1: Server roles

Proposers (leader)

- receive client requests
- propose received requests
- coordinate consensus process for its proposed requests

Acceptors (follower)

- respond to requests from proposers
- validate states of requests
- store chosen values and state of the process

Fundamental #1: Server roles

Proposers (leader)

- receive client requests
- propose received requests
- coordinate consensus process for its proposed requests

Acceptors (follower)

- respond to requests from proposers
- validate states of requests
- store chosen values and state of the process

Learners (subscriber)

- want to know which value is chosen
- subscribe to acceptors
  - one or a few learners communicate with acceptors
  - propagate the message among learners

Fundamental #1: Server roles

Proposers (leader)

- receive client requests
- propose received requests
- coordinate consensus process for its proposed requests

Acceptors (follower)

- respond to requests from proposers
- validate states of requests
- store chosen values and state of the process

Learners (subscriber)

- want to know which value is chosen
- subscribe to acceptors
  - one or a few learners communicate with acceptors
  - propagate the message among learners

According to the application that uses Paxos,
a server can be a proposer, an acceptor, or both

Fundamental #2: Proposals

- Each proposal has a unique number (proposal number)
  - Higher number take a priority over lower numbers
  - Similar to Lamport clock, proposers can increase a proposal number

Fundamental #3: Phases

Prepare phase
(Phase 1)

Phase 1. (a) A proposer selects a proposal number n and sends a *prepare* request with number n to a majority of acceptors.

(b) If an acceptor receives a *prepare* request with number n greater than that of any *prepare* request to which it has already responded, then it responds to the request with a promise not to accept any more proposals numbered less than n and with the highest-numbered proposal (if any) that it has accepted.

Accept phase
(Phase 2)

Phase 2. (a) If the proposer receives a response to its *prepare* requests (numbered n) from a majority of acceptors, then it sends an *accept* request to each of those acceptors for a proposal numbered n with a value v, where v is the value of the highest-numbered proposal among the responses, or is any value if the responses reported no proposals.

(b) If an acceptor receives an *accept* request for a proposal numbered n, it accepts the proposal unless it has already responded to a *prepare* request having a number greater than n.

Proposers

(1) Choose new proposal number *n*.
(2) Broadcast Prepare(*n*) to all servers.

(4) When responses received from majority, if any *acceptedValue* returned, replace value with *acceptedValue* for highest *acceptedProposal*.

(5) Broadcast Accept(*n*, *value*) to all servers

(7) When responses received from majority:
-> Any rejections (result > n) : go to (1)
-> Otherwise, value is chosen

Acceptors

(3) Respond to Prepare(*n*):
-> If n > *minProposal*, then *minProposal* = *n*
-> Return (*acceptedProposal*, *acceptedValue*)

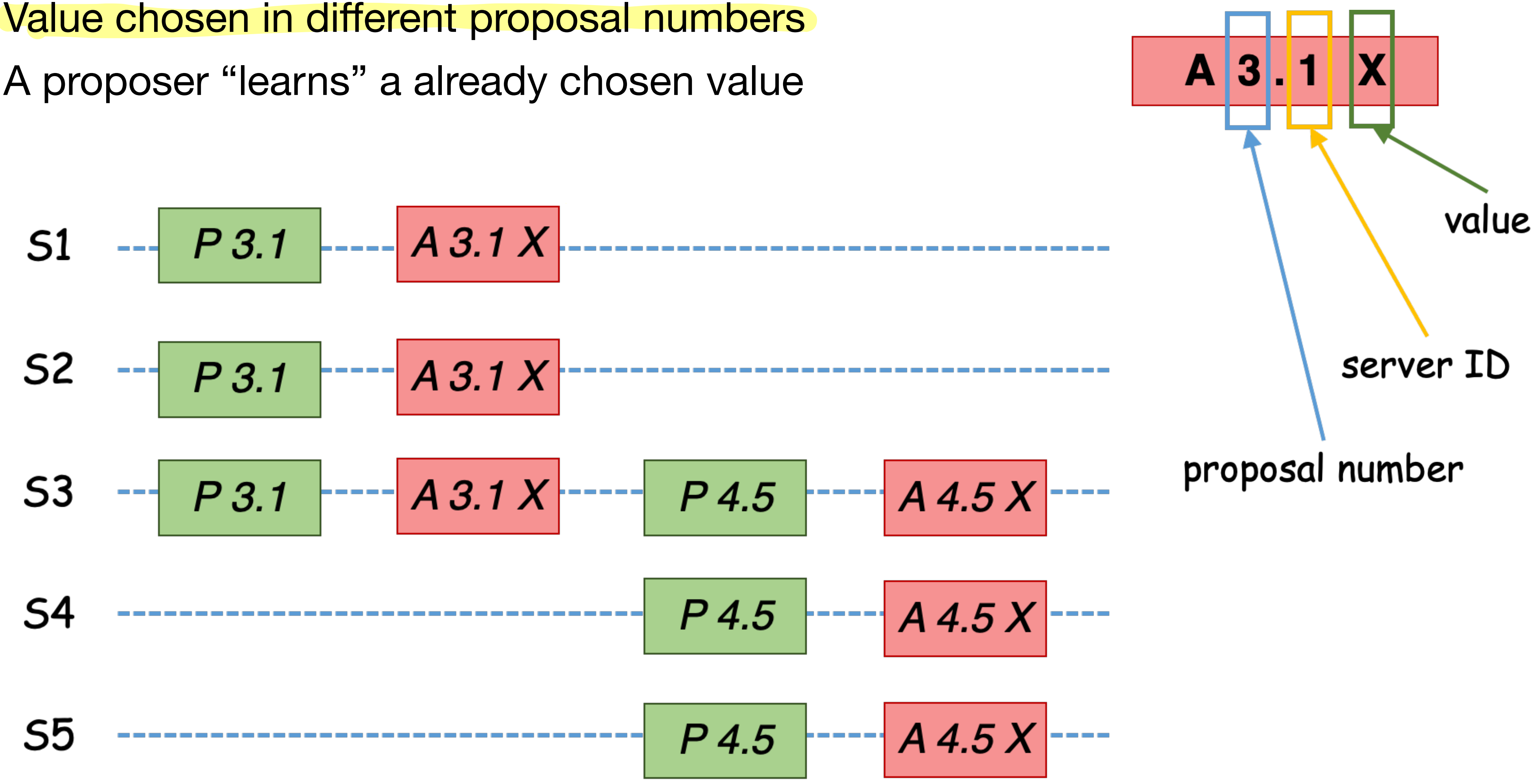
(6) Respond to Accept(*n*, *value*):
-> If *n* >= *minProposal* then
acceptedProposal = *minProposal* = *n*;
acceptedValue = *value*;
-> Return (minProposal)

Acceptors must record *minProposal*, *acceptedProposal*, and *acceptedValue* on stable storage (disk).

From Dr. Diego Ongaro's Paxos lecture

Value chosen in different proposal numbers

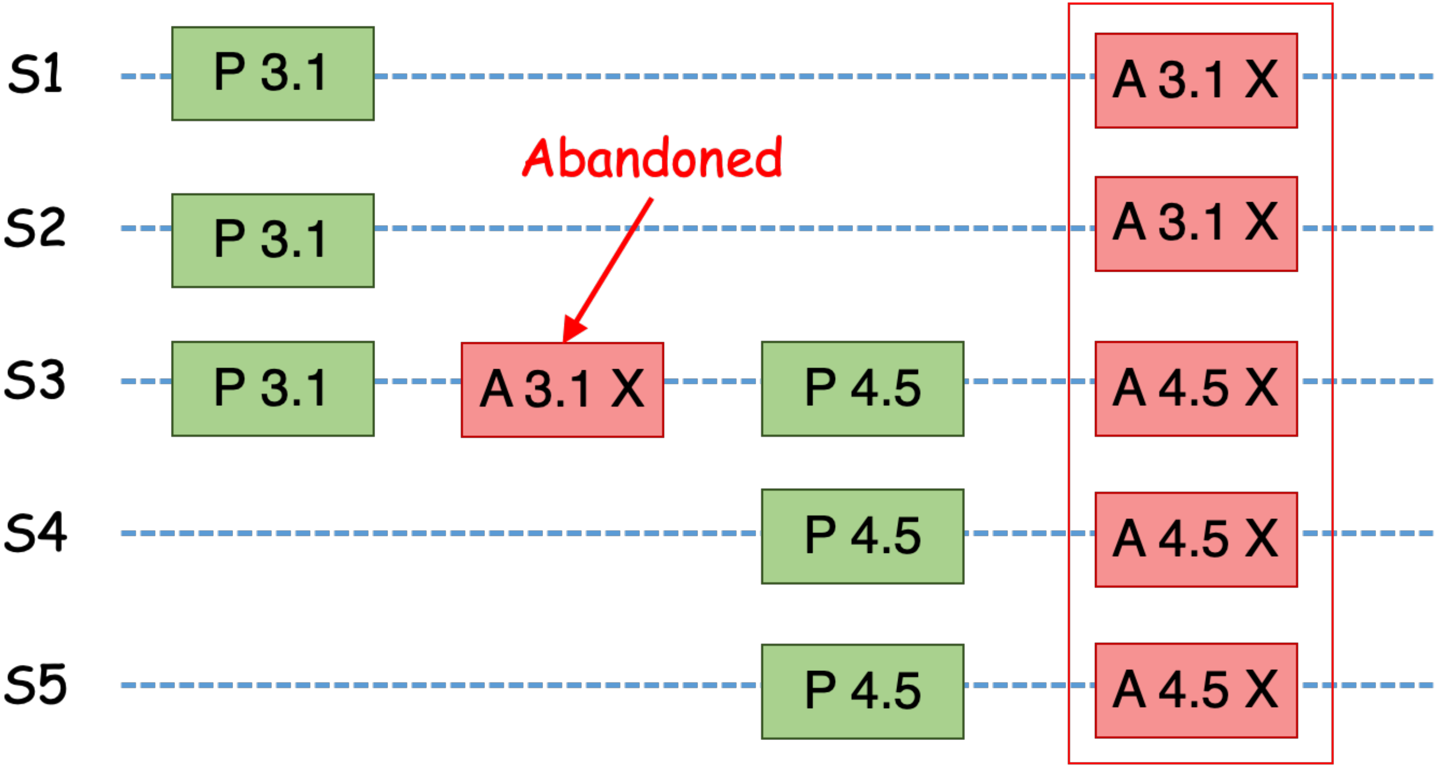
A proposer "learns" a already chosen value



From Dr. Diego Ongaro's Paxos lecture

Value chosen in different proposal numbers

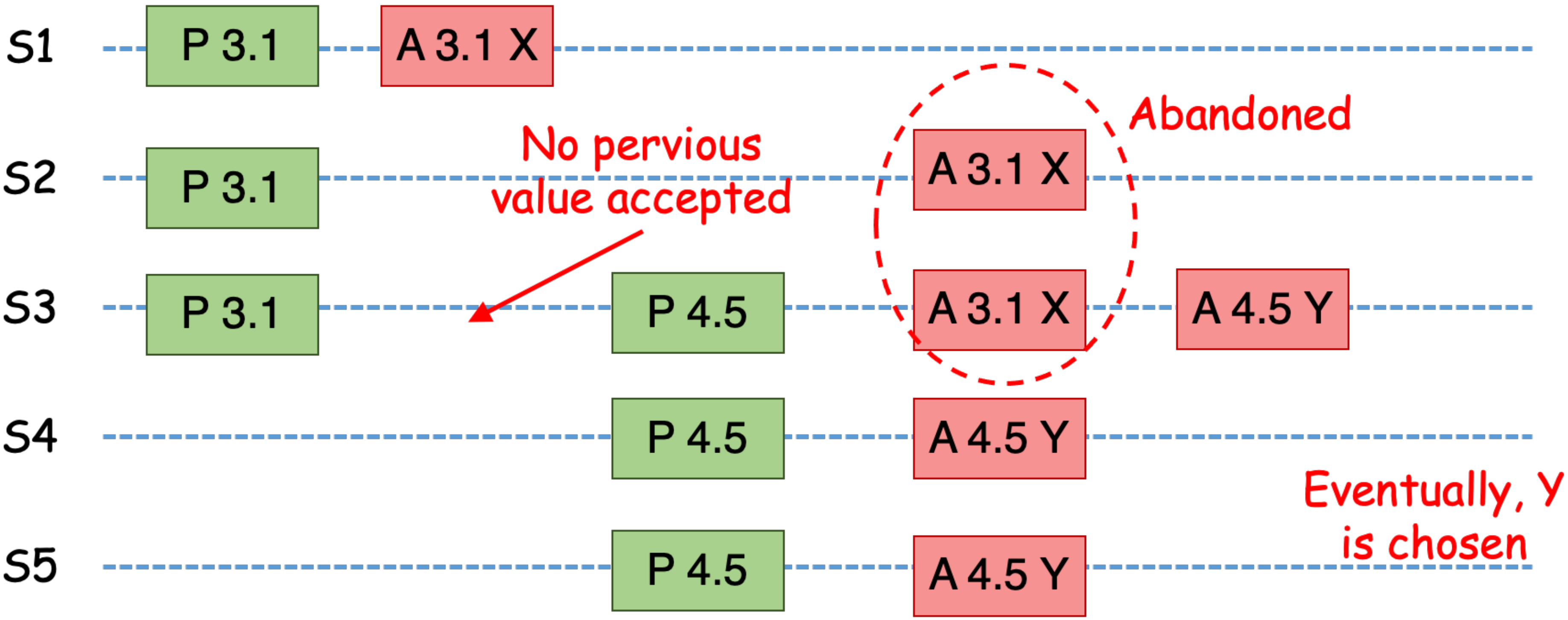
A proposer “learns” a not chosen value



From Dr. Diego Ongaro's Paxos lecture

A new value is chosen in different proposal numbers

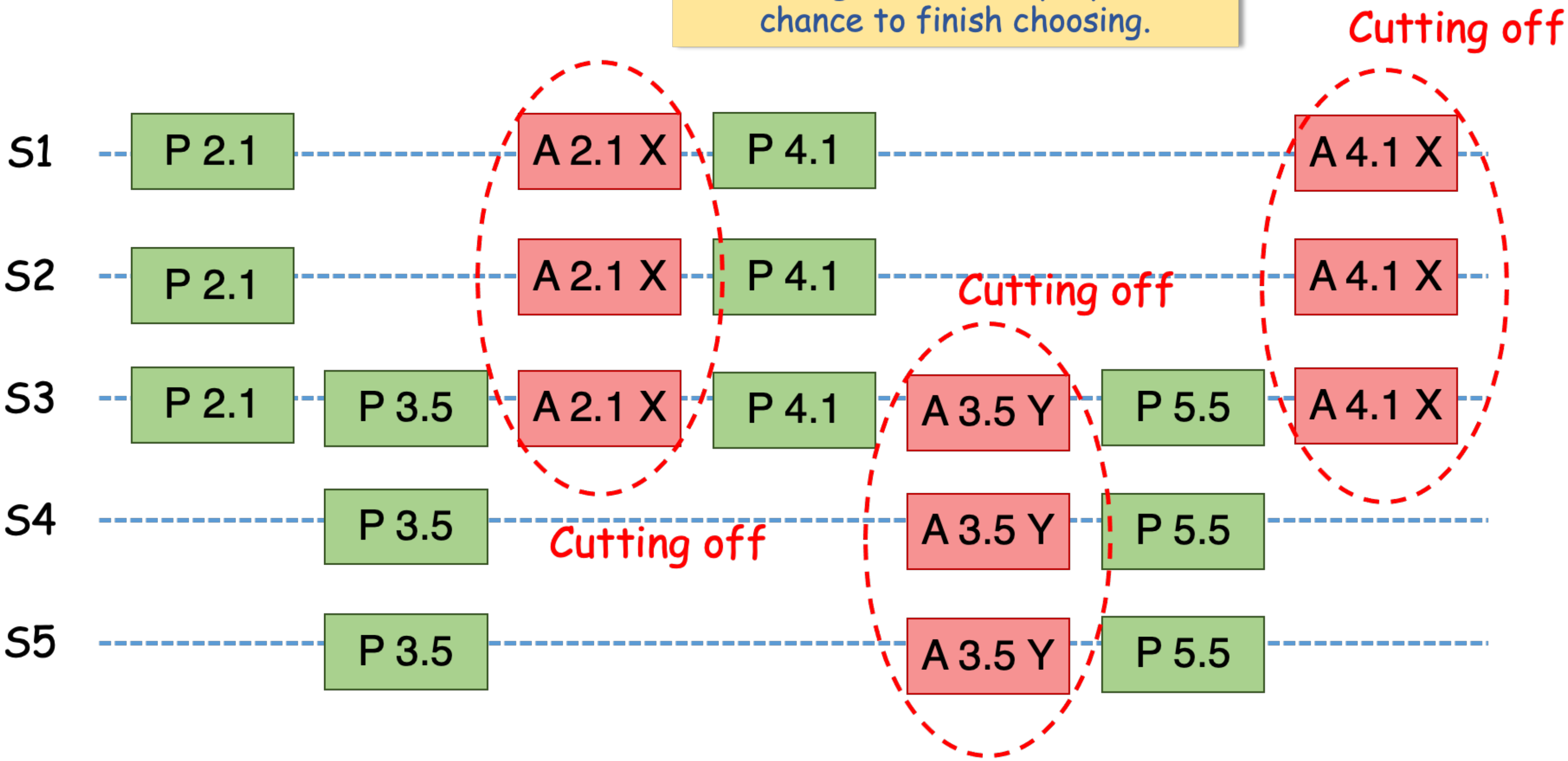
A proposer does not see an unchosen value



From Dr. Diego Ongaro's Paxos lecture

Livelock

Hint:
=> one solution:
Randomized delay before restarting. Give other proposers a chance to finish choosing.



Summary of Paxos

- Anyone can be a proposer/leader
  - Advantages?
  - Disadvantages?
- Only proposer knows which value has been chosen
- If other servers want to know, must execute Paxos with their own proposal
- Competing proposers can cause a livelock

Worksheet