Consensus

CMPS 4760/6760: Distributed Systems

Acknowledgement: slides adapted from Indranil Gupta's lecture notes:
https://courses.engr.illinois.edu/cs425/fa2019/index.html

Overview

- Distributed Mutual Exclusion (15.2)
- Leader Election (15.3)
- Group communication (6.2,15.4,18.2)
- Consensus (15.5, 21.5.2)

Consensus

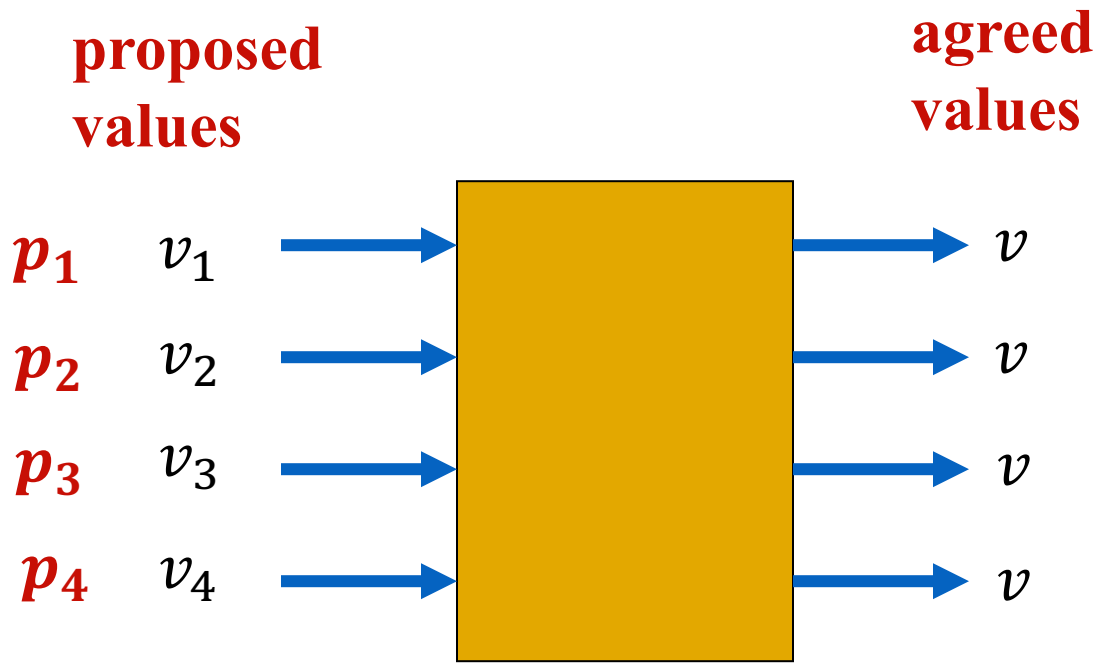
- Problem definition (15.5.1)
- Consensus in synchronous systems
  - Consensus under crash failures (15.5.2)
  - Byzantine generals problem (15.5.3)
- Consensus in asynchronous systems
  - FLP Impossibility Result (15.5.4)
- Paxos (21.5.2)

Consensus Problem

- Problem: a collection of processes need to agree on a value after one or more of them has proposed what that value should be
- Reaching agreement is a fundamental requirement in distributed computing
  - Leader election / Mutual Exclusion
  - Commit or Abort in distributed transactions
  - Reaching agreement about which process has failed
  - Air traffic control system: all aircrafts must have the same view

Consensus

- Each process P_i beings in undecided and proposes value $v_i \in D$.
- Processes exchange values with each other via message passing
- P_i enters the decided state by setting the value of a decision variable d_i (write-once).



Requirements

- Termination: Eventually each correct process sets its decision variable
- Agreement: For any two processes P_i and P_j, if they are correct and have entered the *decided* state, then $d_i = d_j$
- Integrity: If the correct processes all proposed the same value v, then for any correct process P_i in the *decided* state, $d_i = v$

Assumptions

- *N* processes, message passing only
- Communication is reliable
- Processes can fail: crash or byzantine
- Up to some number f of N processes are faulty
- Messages are not signed (‘oral’ messages)

When processes cannot fail

- A simple solution to solve consensus:
  - Each P_i reliably multicasts its proposed value to the group
  - Each P_i waits until it has collected all N values and then sets $d_i = majority(v_1, v_2, \ldots v_N)$
    - If no majority exists, $majority(v_1, v_2, \ldots v_N) = \perp$
    - Other functions can also be applied, e.g., min or max for values that are ordered

When processes can fail

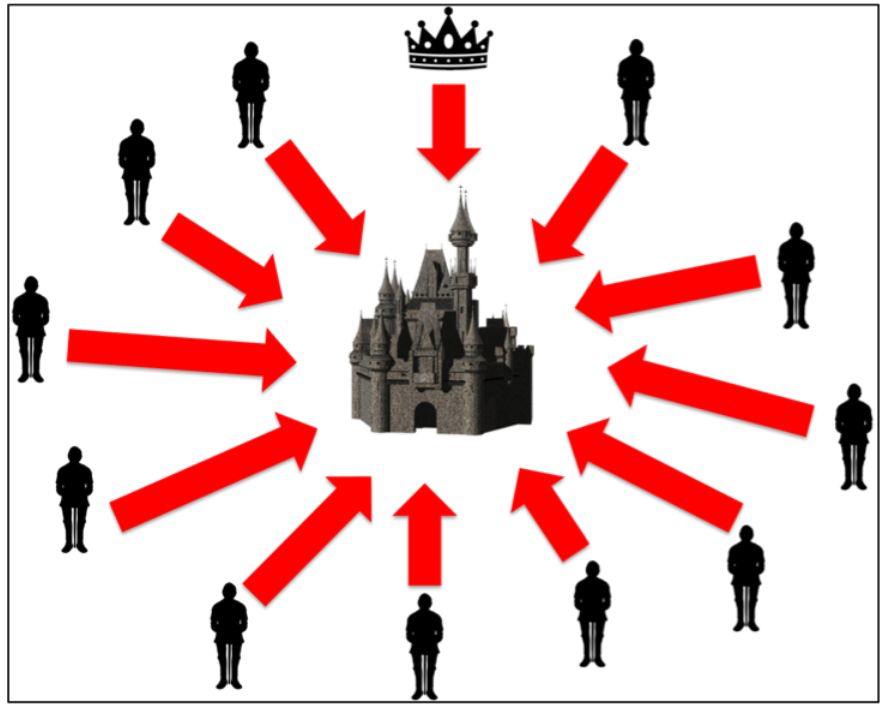
- Can we always achieve consensus if processes can crash?
  - Yes if the system is synchronous
  - No if the system is asynchronous even with a single process failure (FLP impossibility result)
    - Whatever protocol/algorithm you suggest, there is always a worst-case possible execution (with failures and message delays) that prevents the system from reaching consensus
    - Subsequently, safe or probabilistic solutions have become quite popular to consensus or related problems.
- What if process can fail in arbitrary (Byzantine) ways?
  - No if the system is asynchronous
  - Yes if the system is synchronous and $N > 3f$

Consensus and RTO-Multicast

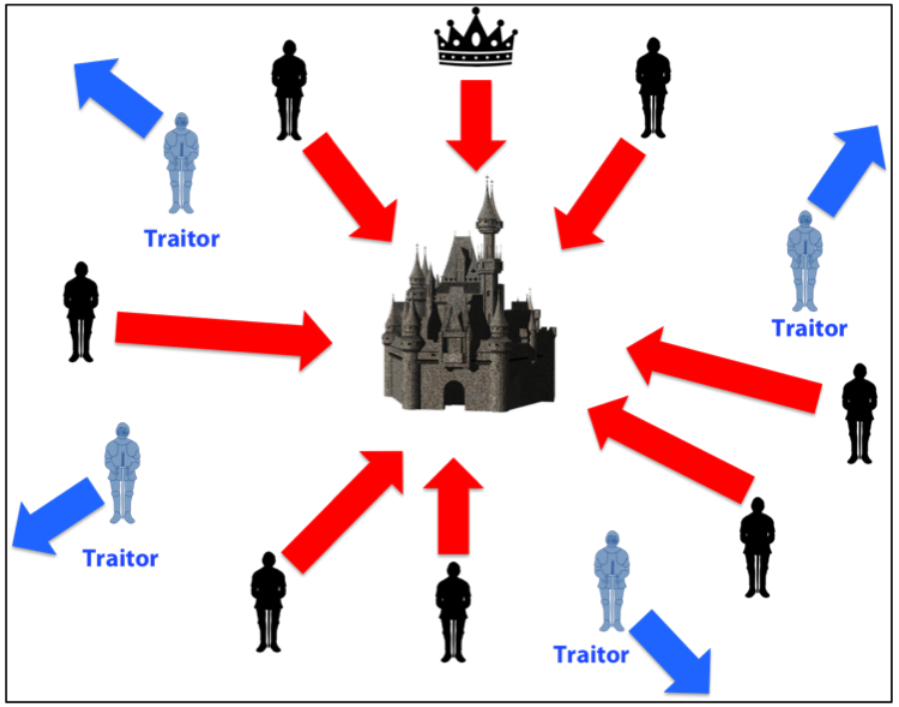
- Implementing consensus using RTO-multicast
  - Each P_i multicasts its proposed value to the group using RTO-multicast
  - Each P_i sets $d_i =$ the first value it delivers.

- Implementing RTO-multicast using consensus [Chandra and Toueg 1996]

Byzantine Generals



Coordinated Attack Leading to Victory



Uncoordinated Attack Leading to Defeat

Byzantine Generals

- Three or more generals are to agree to attack or to retreat
- One, the commander, issues the order
- The others, lieutenants to the commander, decide whether to attack or retreat
- Both the commander and the generals can be treacherous

Byzantine Generals

- Termination: Eventually each correct process sets its decision variable
- Agreement: For any two processes P_i and P_j, if they are correct and have entered the *decided* state, then $d_i = d_j$
- Integrity: If the commander is correct, then all correct processes decide on the value that the commander proposed

Byzantine Generals and Consensus

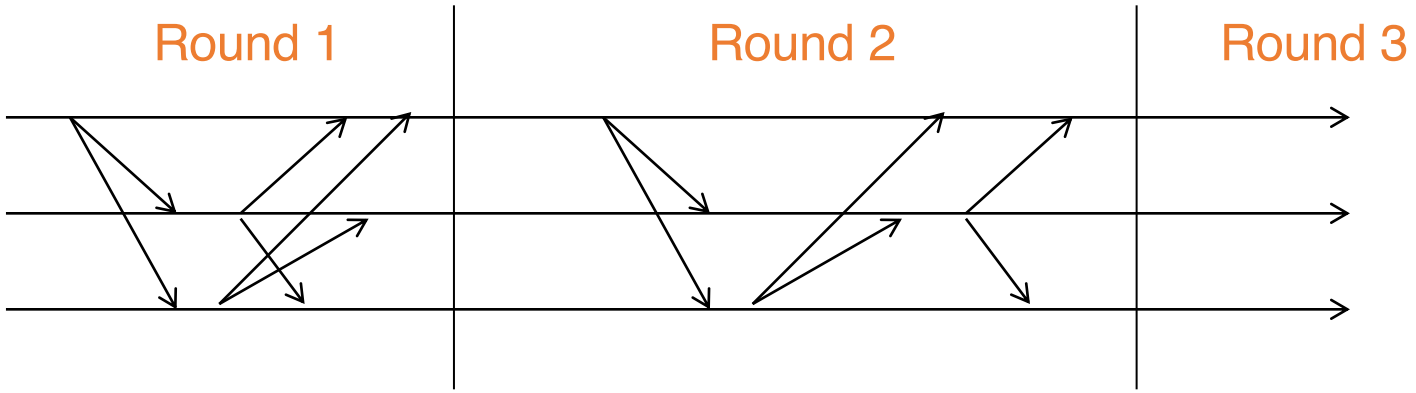
- BG from C: We can construct a solution to BG from C as follows:
  - The commander P_j sends its proposed value to itself and each of the lieutenants (P_j may be faulty)
  - All generals run C with the values $v_1, v_2 \ldots, v_N$ that they receive
- C from BG: homework

Consensus

- Problem definition (15.5.1)
- Consensus in synchronous systems
  - Consensus under crash failures (15.5.2)
  - Byzantine generals problem (15.5.3)
- Consensus in asynchronous systems
  - FLP Impossibility Result (15.5.4)
- Paxos (21.5.2)

Consensus in a Synchronous System with Crash Failures

- At most f processes crash (f is known)
- All processes are synchronized and operate in "rounds" of time
- The algorithm proceeds in $f + 1$ rounds (with timeout), using reliable communication to all members
- $Values_i^r$: the set of proposed values known to P_i at the beginning of round r



Consensus in a Synchronous System with Crash Failures

At most f processes crash (f is known)

The algorithm proceeds in $f + 1$ rounds (with timeout), using reliable communication to all members

$Values_i^r$: the set of proposed values known to P_i at the beginning of round r

Initially $Values_i^0 = \{\}$; $Values_i^1 = \{v_i\}$

for round $r = 1$ to $f + 1$ do

multicast ($Values_i^r - Values_i^{r-1}$) // iterate through processes, send each a message

$Values_i^{r+1} = Values_i^r$

for each V_j received

$Values_i^{r+1} = Values_i^{r+1} \cup V_j$

end

end

d_i = minimum$(Values_i^{f+2})$

Consensus in a Synchronous System with Crash Failures

- Message complexity: $O((f+1)N^2)$
- The simple algorithms guarantees
  - Termination: each correct process terminates in $f+1$ rounds
  - Integrity: set V contains only the proposed values
  - Agreement: Let V_i denote the set of values of P_i after the round $f+1$

    Claim: If any value v is in the set V_i for some correct process P_i, then it is also in the set of V_j of any other correct process P_j

Consensus in a Synchronous System with Crash Failures

Claim: If any value v is in the set V_i for some correct process P_i, then it is also in the set of V_j of any other correct process P_j

Proof by contradiction:

- Assume that after $f + 1$ rounds, P_i possesses a value v that P_j does not possess.

  →P_i must have received v in the very last round

  →Else, P_i would have sent v to P_j in that last round

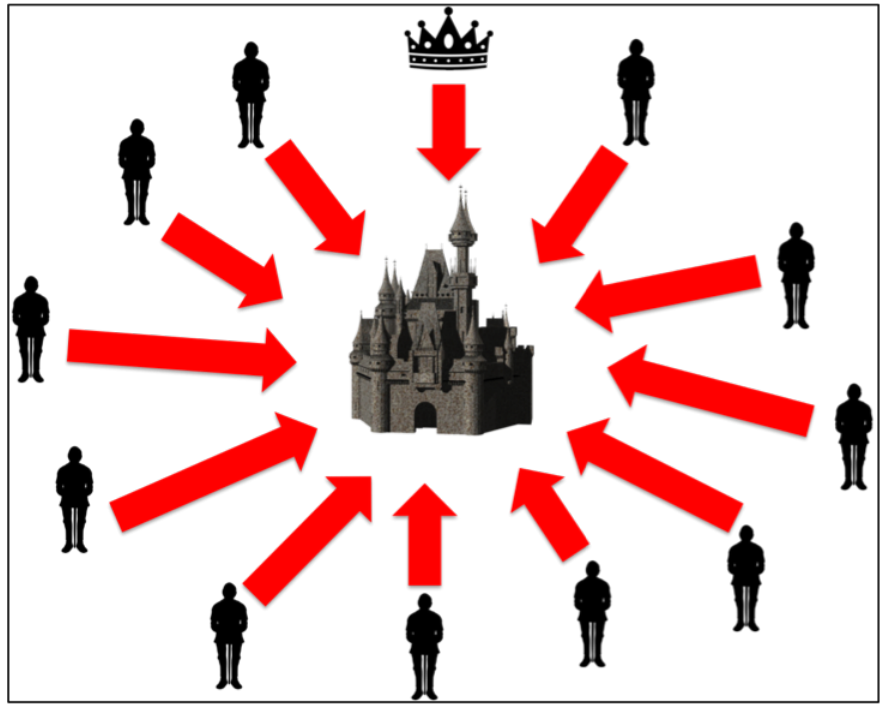
  → So, in the last round: a third process, P_k, must have sent v to P_i, but then crashed before sending v to P_j.

  → Similarly, a fourth process sending v in the last-but-one round must have crashed; otherwise, both P_k and P_j should have received v.

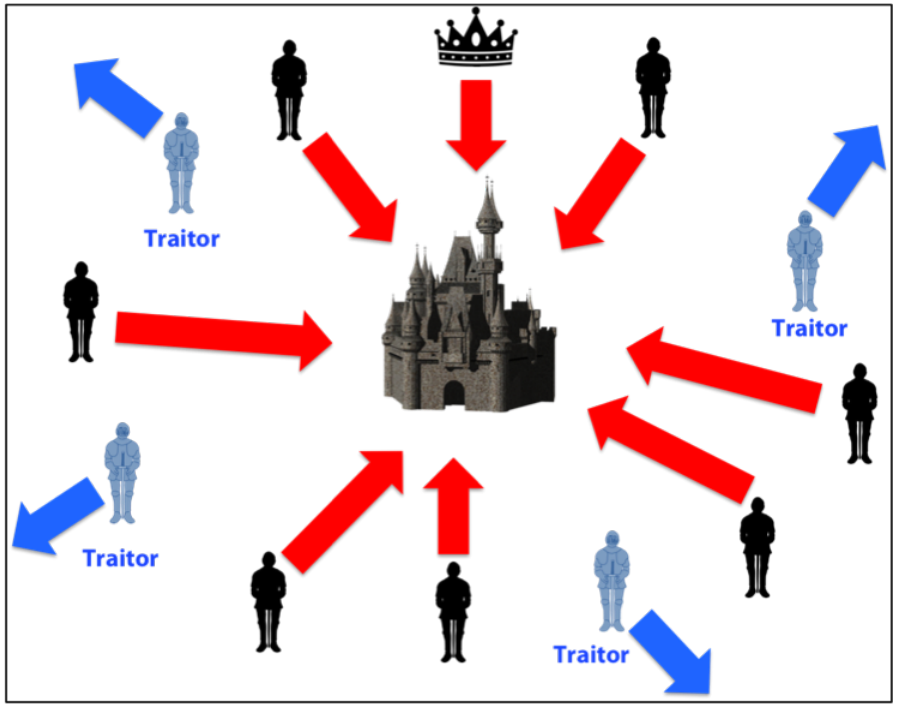
  → Proceeding in this way, we infer at least one (unique) crash in each of the preceding rounds.

  → This means a total of $f + 1$ crashes, while we have assumed at most f crashes can occur => contradiction.

Byzantine Generals



Coordinated Attack Leading to Victory

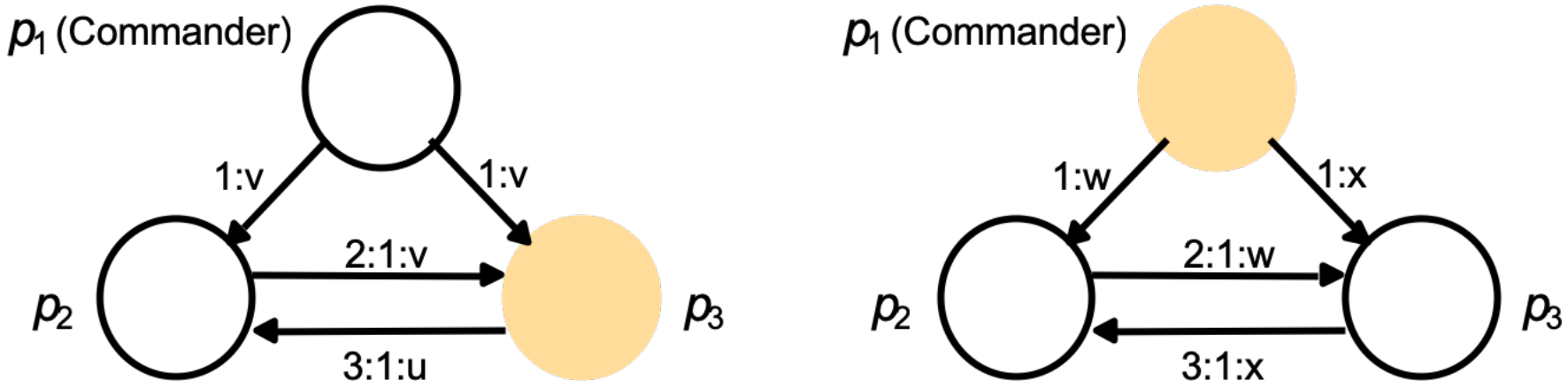


Uncoordinated Attack Leading to Defeat

Impossibility result

- “if the generals can send only oral messages, then no solution will work unless more than $^2/_3$ of the generals are loyal.”
- what are oral messages?
  - every message that is sent is delivered correctly
  - the receiver of a message knows who sent it
  - the absence of a message can be detected

Impossibility with three processes



Faulty processes are shown coloured

Impossibility with $N \leq 3f$: a reduction from the three-process case

Solution with one faulty process

- $N \geq 4, f = 1$

Round 1: the commander sends a value to each of the lieutenants

- The value can be different to different lieutenants if the commander is faulty

Round 2: each of the lieutenants sends the value it received to its peers

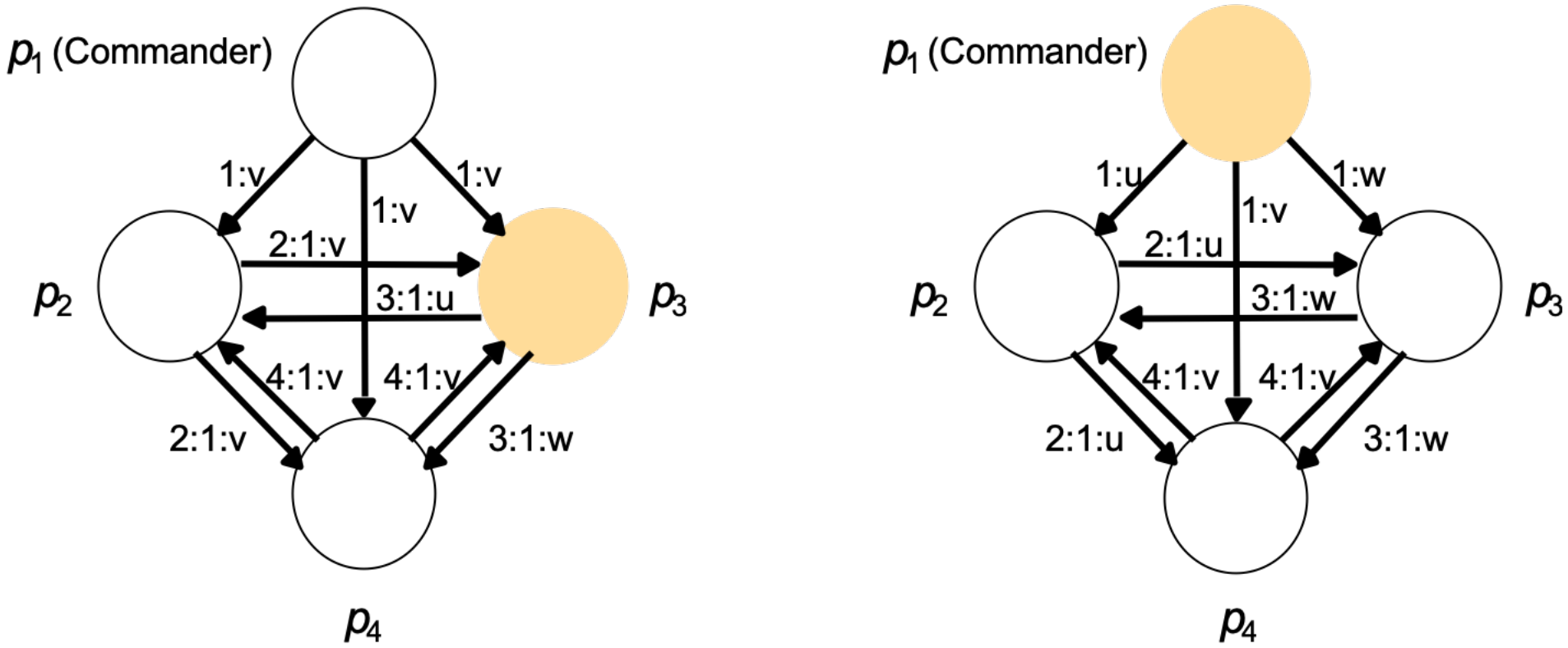
- The value sent can be different from the value received if the lieutenant is faulty

Each lieutenant applies the majority function to the set of values it receives

Solution with one faulty process

- $N \geq 4, f = 1$
- Correctness of the solution
  - If the commander is faulty
    - all the lieutenants are correct and each will have gathered exactly the set of values that the commander sent out.
  - Otherwise, one of the lieutenants is faulty
    - each of its correct peers receives N – 2 copies of the value that the commander sent, plus a value that the faulty lieutenant sent to it.
    - Since $N \geq 4$, $N - 2 \geq 2$, the majority function will ignore any value that a faulty lieutenant sent, and it will produce the value that the commander sent

Four Byzantine Generals



Faulty processes are shown coloured

Solution with f faulty processes

- Lamport *et al.* 1982 gives a general solution for unsigned messages that
  - operates over $f + 1$ rounds - best possible
  - has a message complexity of $O(N^{f+1})$ – can be improved

Consensus with signed messages

- With only oral messages, traitors can lie by telling the wrong command they received
- Signed messages
  - cannot be forged
  - anyone can verify the authenticity
- Consensus can be reached for $N \geq f + 2$ using signed messages
- Dolev and Strong [1983] gives a solution for signed messages that
  - operates over $f + 1$ rounds
  - has a message complexity of $O(N^2)$

Consensus

- Problem definition (15.5.1)
- Consensus in synchronous systems
  - Consensus under crash failures (15.5.2)
  - Byzantine generals problem (15.5.3)
- Consensus in asynchronous systems
  - FLP Impossibility Result (15.5.4)
- Paxos (21.5.2)

FLP Impossibility Result

- One of the most important results in distributed computing
- “Impossibility of Distributed Consensus with One Faulty Process”, Journal of the ACM, Vol. 32, No. 2, 1985.

Michael J. Fischer

Nancy A Lynch

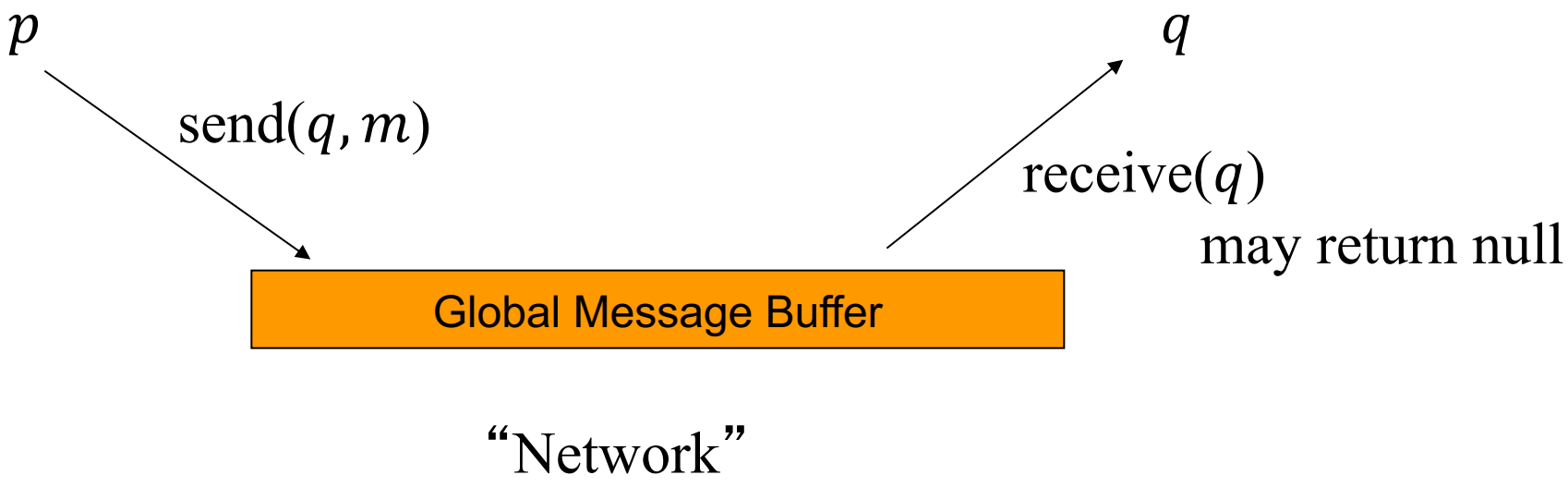
Michael S. Paterson

Models and Assumptions

- Consider an easier problem and a more restrictive system model
- Each process i proposes a binary value v_i in $\{0,1\}$
- Only one process can fail (and we can choose which one)
- A process can fail only by crashing
- Communication is reliable but delay is unbounded
- A weaker termination requirement: some process eventually enters the decision state
- A weaker integrity requirement (non-triviality): both values 0 and 1 should be possible outcomes

Network

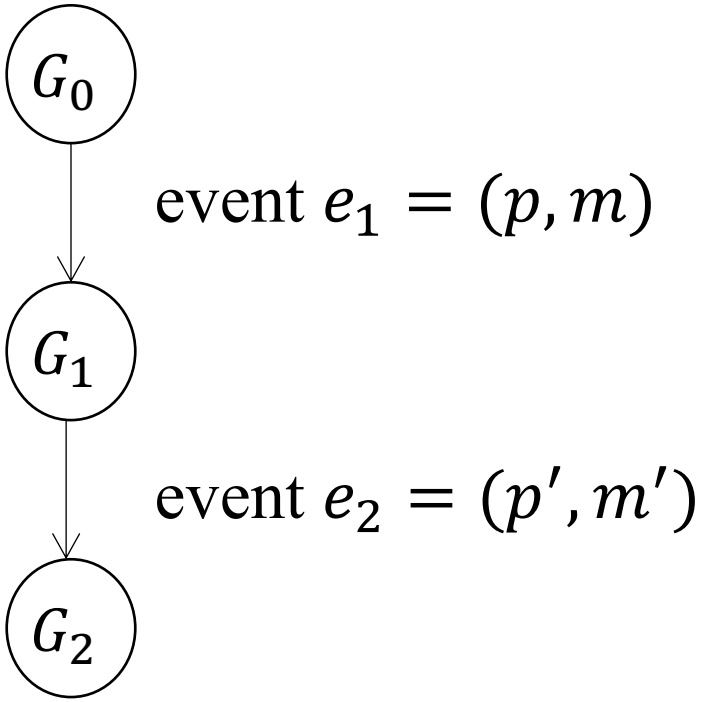
- A global message buffer

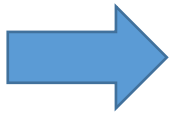


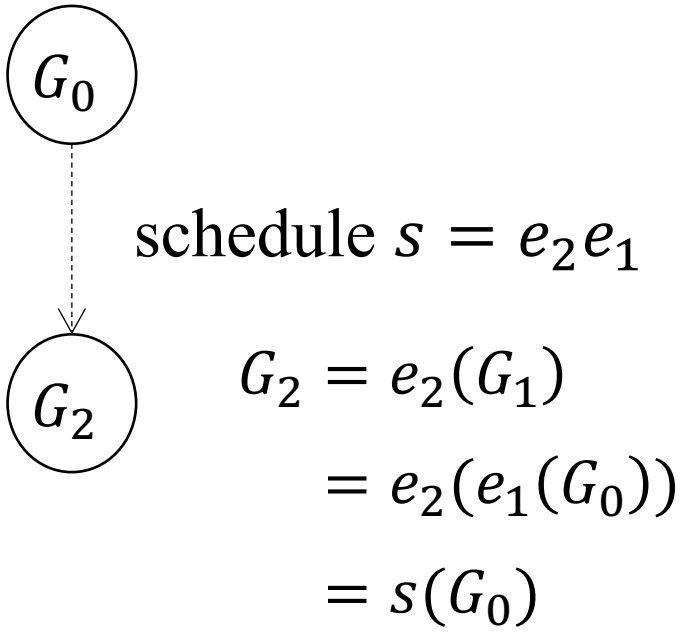
- If receive() is performed an unbounded number of times, then every message is eventually delivered

States

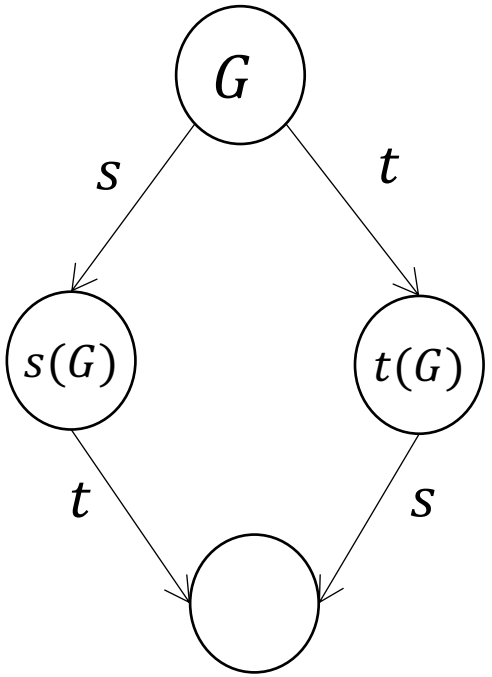
- Global state G: state of all the processes and the state of the global buffer
- $G[i]$: state of process i, initially this includes the proposed value v_i and the decision variable $d_i = \perp$ (undecided)
- An event (p, m) consists of
  - receipt of a message m by a process p
  - processing of m (may change recipient's state)
  - sending out of all necessary messages by p
- Schedule: sequence of events


G_0
event $e_1 = (p, m)$
G_1
event $e_2 = (p', m')$
G_2




G_0
schedule $s = e_2 e_1$
G_2
$G_2 = e_2(G_1)$
$= e_2(e_1(G_0))$
$= s(G_0)$

Commute property of disjoint events

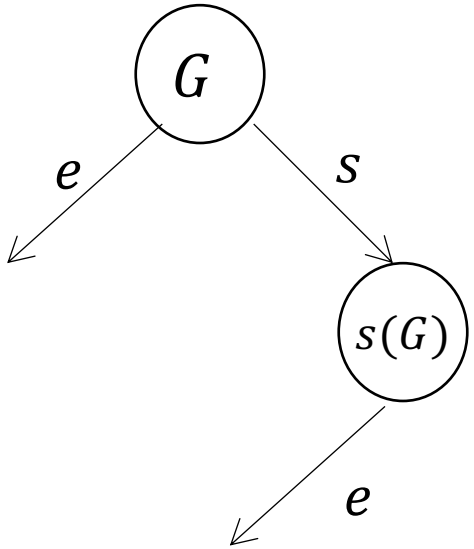


$st(G) = ts(G)$

$st(G) = ts(G)$ if s and t involve disjoint set of receiving processes and are each applicable to G

Asynchrony of events

- Any event may be arbitrarily delayed



If e is enabled at G and $e \notin s$,
it is still enabled at $s(G)$

Main Idea of the Proof

1. There is an initial global state in which the system is indecisive
2. There exists a method to keep the system indecisive

- How to model indecision?

Bivalent states

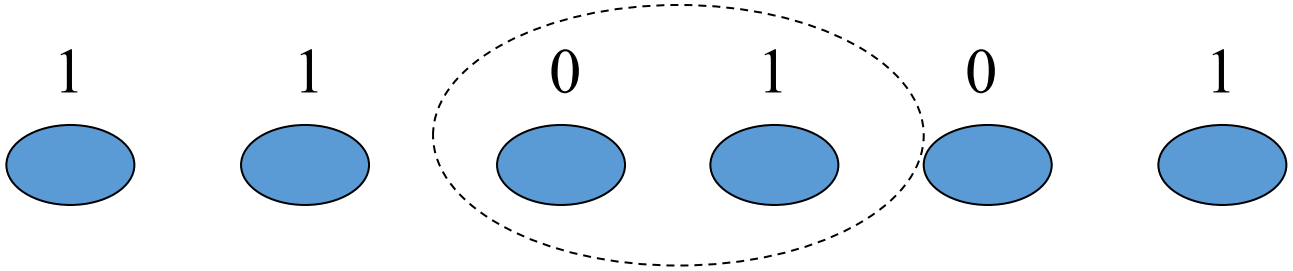
- $G.V$ – set of decision values reachable from global state G
  - If $G.V = \{0\}$, G is 0–valent
  - If $G.V = \{1\}$, G is 1–valent
  - If $|G.V| = 2$, G is bivalent (indecisive)

- Main idea of the proof

1. Every consensus protocol has a bivalent initial global state
2. Starting from a bivalent global state, there is always another bivalent global state that is reachable

Proof of Lemma 1 - Some initial state is bivalent

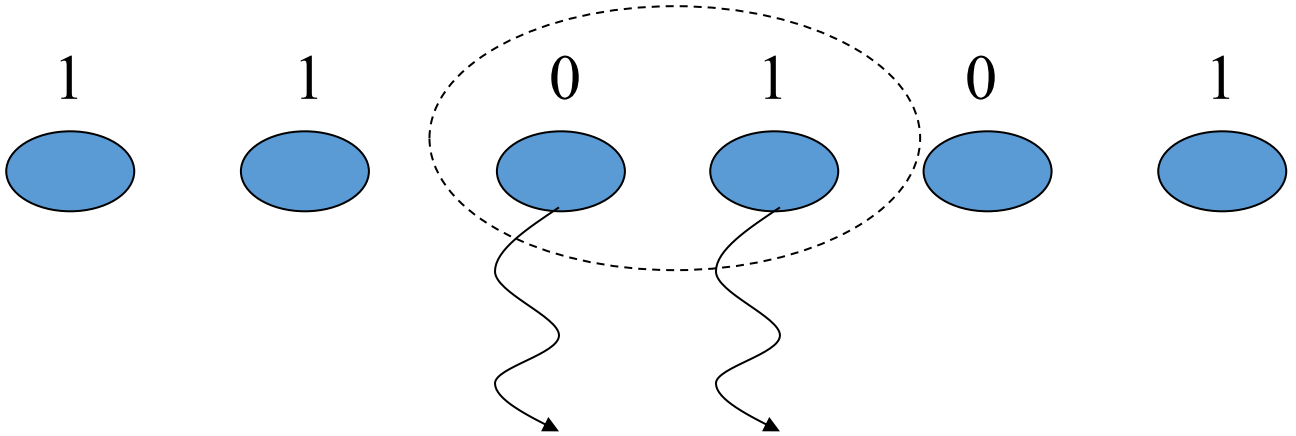
- Suppose all initial global states were either 0-valent or 1-valent
- If there are N processes, there are 2^N possible initial configurations
- The protocol must have both 0-valent or 1-valent states (from integrity)
- Place all initial global states side-by-side (in a lattice), where adjacent states differ in proposed values for exactly one process



- Claim: there has to be some adjacent pair of 1-valent and 0-valent global states

Proof of Lemma 1 (cont.)

- There has to be some adjacent pair of 1-valent and 0-valent global states
- Assume that they differ in the state of p and let p be the process that has crashed (i.e., is silent throughout)
- Both initial states will reach the same decision value for the same sequence of events, a contradiction



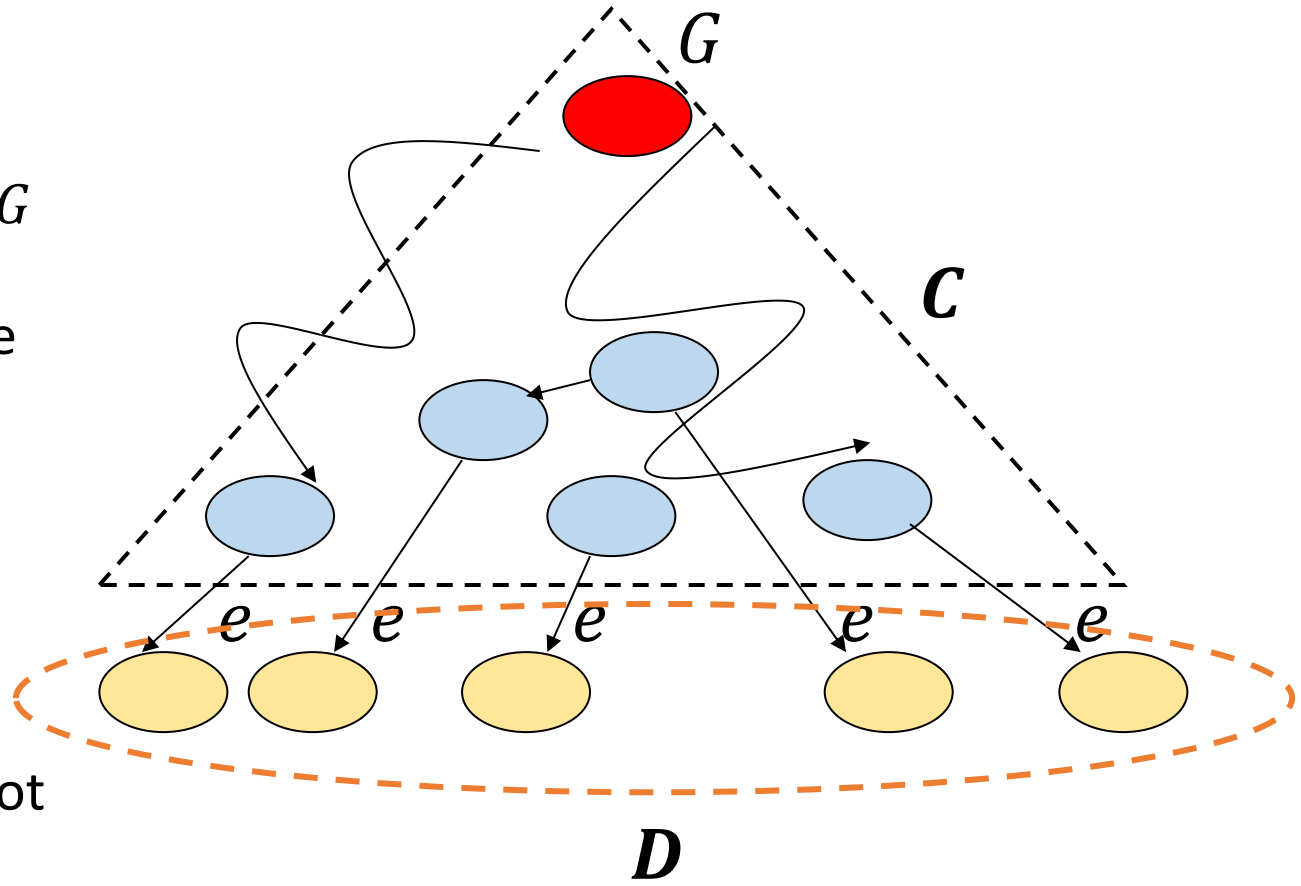
Main Idea of the Proof

1. Every consensus protocol has a bivalent initial global state

2. Starting from a bivalent global state, there is always another bivalent global state that is reachable

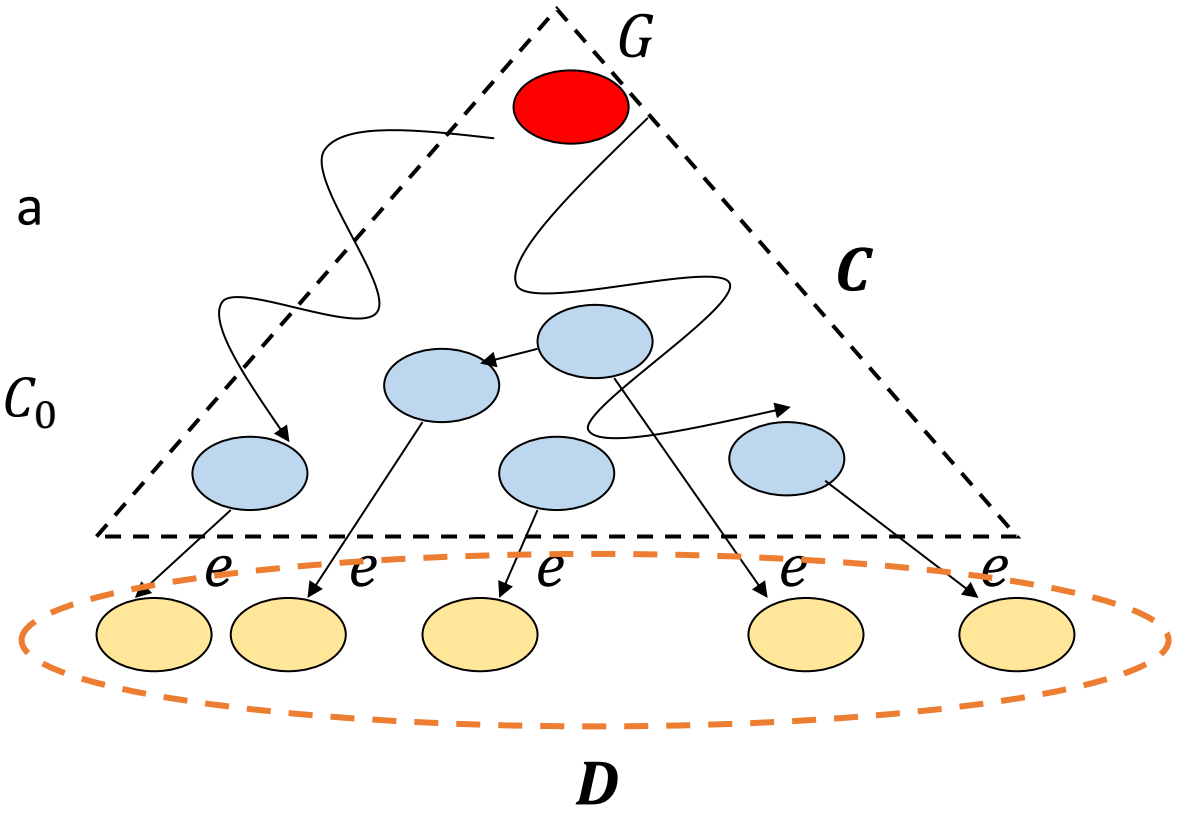
Proof of Lemma 2 - Starting from a bivalent state, there is always another bivalent state that is reachable

- Let G be a bivalent global state of a protocol.
- Let $e = (p, m)$ be an event applicable to G
- Let $\boldsymbol{C}$ be the set of global states reachable from G without applying e
- Let $\boldsymbol{D} = e(\boldsymbol{C})$ (asynchrony of events)
- Claim: $\boldsymbol{D}$ contains a bivalent global state
- Proof by contradiction. Assume $\boldsymbol{D}$ does not contain a bivalent global state



Proof of Lemma 2 (cont.)

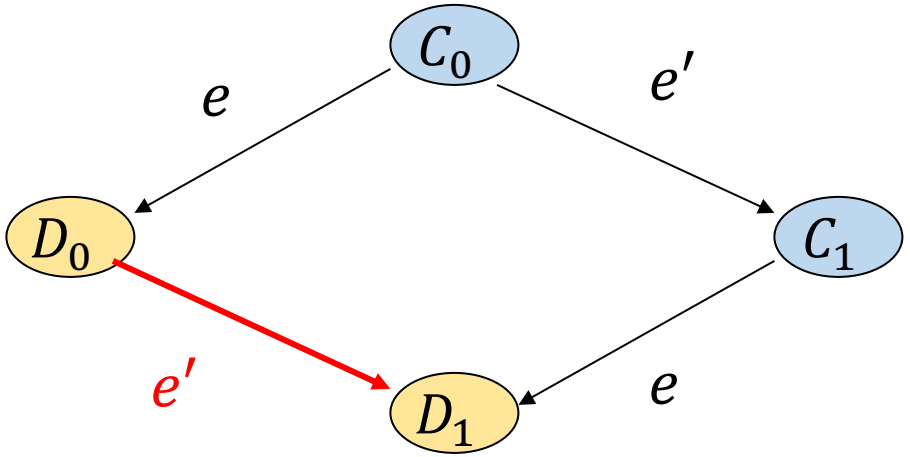
- Proof by contradiction. Assume $\boldsymbol{D}$ does not contain a bivalent global state
- Claim : There are states D_0 and D_1 in $\boldsymbol{D}$, and states C_0 and C_1 in $\boldsymbol{C}$ such that
  - D_0 is 0-valent, D_1 is 1-valent
  - $D_0 = e(C_0), D_1 = e(C_1)$
  - $C_1 = e'(C_0)$ for some event $e' = (p', m')$

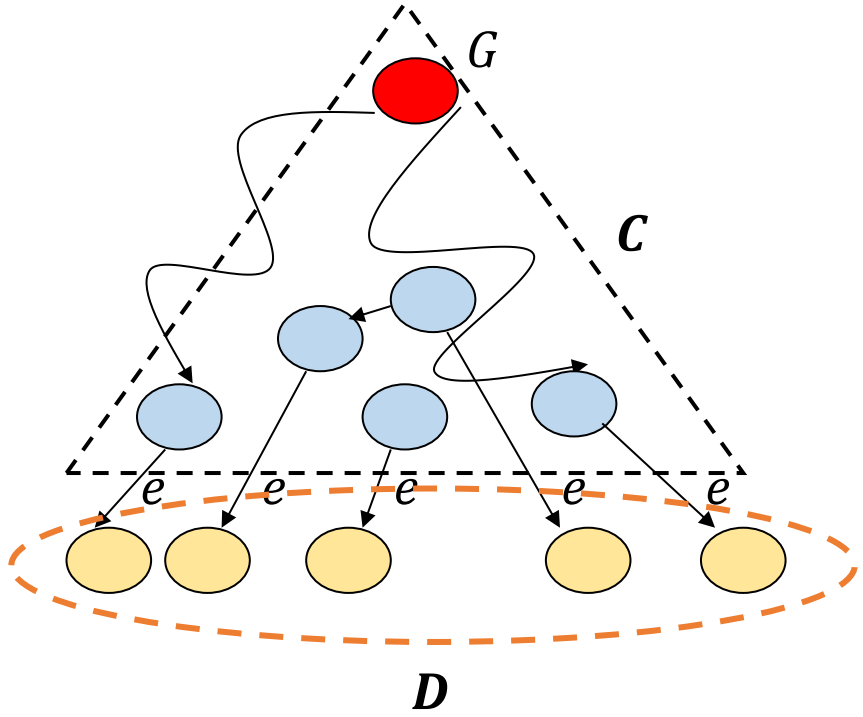


Proof of the claim: (1) $\boldsymbol{D}$ must contain both 0-valent and 1-valent states; (2) consider the shortest sequence t without applying e such that $et(G)$ has different valency from $e(G)$. Let C_0 and C_1 be the last two states reached in sequence t.

Proof of Lemma 2 (cont.)

- Let $C_1 = e'(C_0)$ where $e' = (p', m')$
- Case 1: $p' \neq p$



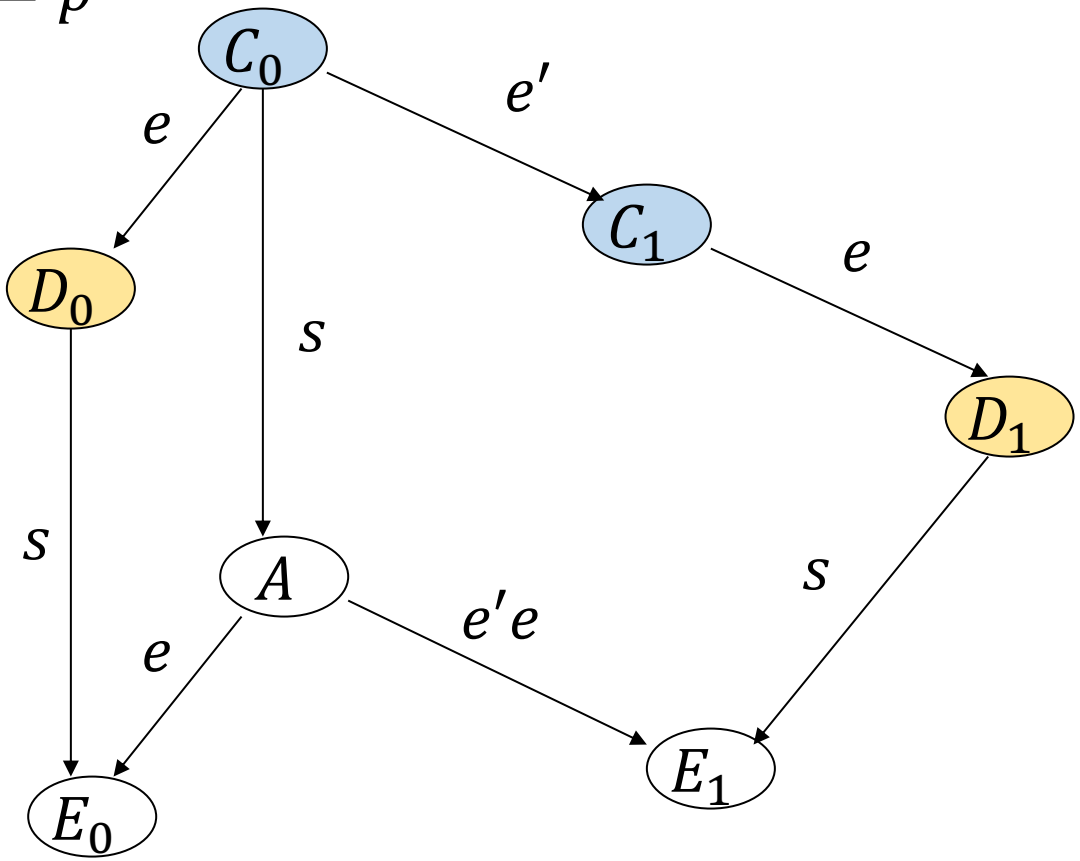


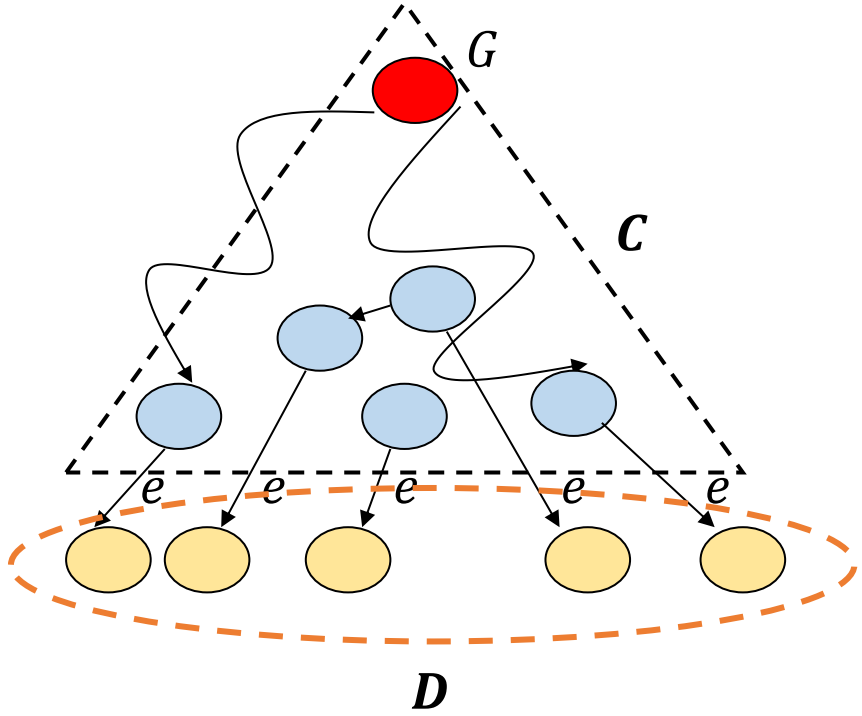
e' is applicable to D_0 by commutativity of disjoint events

=> a contradiction since D_0 is 0-valent and D_1 is 1-valent

Proof of Lemma 2 (cont.)

- Let $C_1 = e'(C_0)$ where $e' = (p', m')$
- Case 2: $p' = p$





s: a finite deciding run from C_0 in which p takes no steps

A is bivalent, contradicting the fact that s is a deciding run

Putting it all together

1. Every consensus protocol has a bivalent initial global state

2. Starting from a bivalent global state, there is always another bivalent global state that is reachable

- Theorem (Impossibility of Consensus): There is always a run of events in an asynchronous distributed system such that the group of processes never reach consensus (i.e., stays bivalent all the time)

Consensus

- Problem definition (15.5.1)
- Consensus in synchronous systems
  - Consensus under crash failures (15.5.2)
  - Byzantine generals problem (15.5.3)
- Consensus in asynchronous systems
  - FLP Impossibility Result (15.5.4)
- Paxos (21.5.2)

Paxos Algorithm

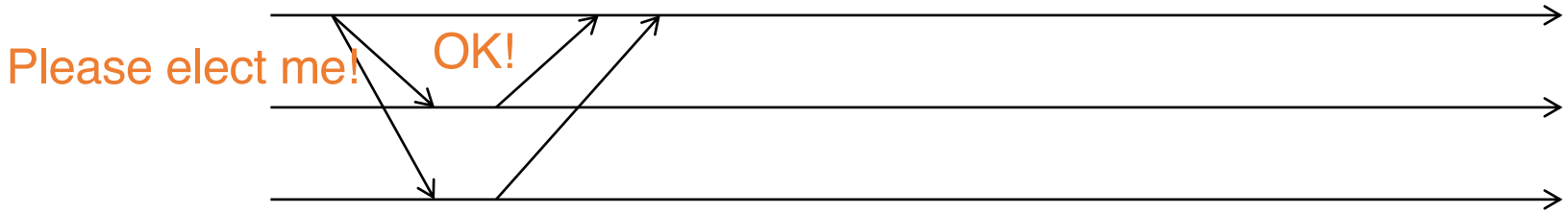
- Invented by Leslie Lamport
- Most popular “consensus-solving” algorithm
- Safety is provided: agreement and integrity
- Liveness is not: no guarantee on termination
  - Provides eventual liveness if majority of of the processes run for long enough with sufficient network stability
- A lot of systems use it
  - Zookeeper (Yahoo!), Google Chubby, and many other companies

Paxos Overview

- Paxos has rounds; each round has a unique ballot id
- Rounds are asynchronous
  - Time synchronization not required
  - If you're in round *j* and hear a message from round *j+1*, abort everything and move over to round *j+1*
  - Use timeouts; may be pessimistic
- Each round itself broken into phases (which are also asynchronous)
  - Phase 1: A leader is elected
  - Phase 2: Leader proposes a value, processes ack
  - Phase 3: Leader multicasts final value

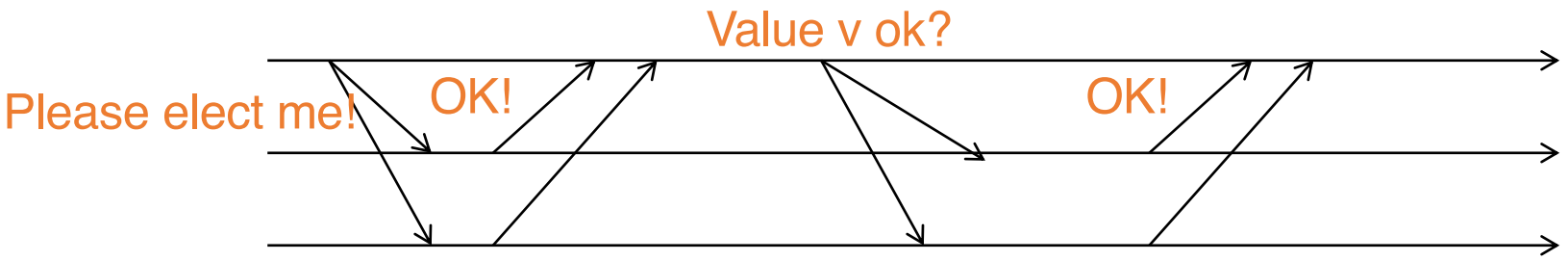
Phase 1 – Election

- Potential leader chooses a unique ballot id, higher than seen anything so far
- Sends to all processes
- Processes wait, respond once to highest ballot id
  - If potential leader sees a higher ballot id, it can't be a leader
  - Paxos tolerant to multiple leaders, but we'll only discuss 1 leader case
  - Processes also log received ballot ID on disk (why?)
- If a process has in a previous round decided on a value v', it includes value v' in its response
- If majority (i.e., quorum) respond OK then you are the leader
  - If no one has majority, start new round
- (If things go right) A round cannot have two leaders (why?)



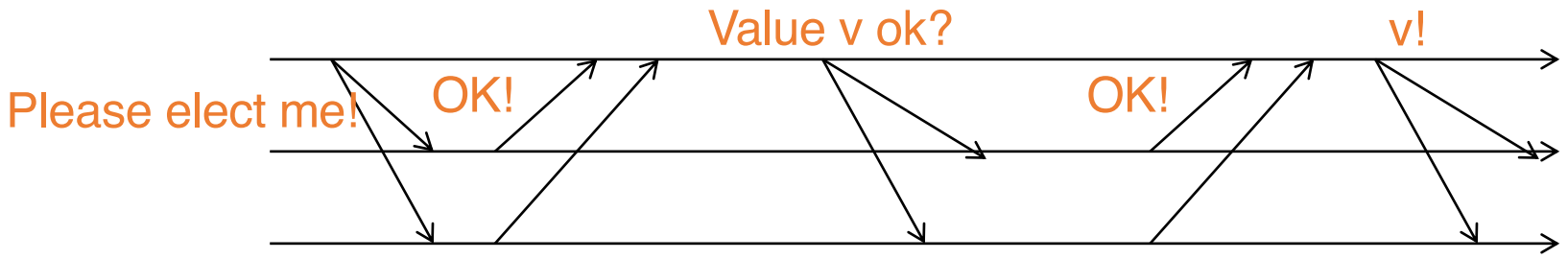
Phase 2 – Proposal

- Leader sends proposed value v to all
  - use v=v’ if some process already decided in a previous round and sent you its decided value v’
- Recipient logs on disk; responds OK



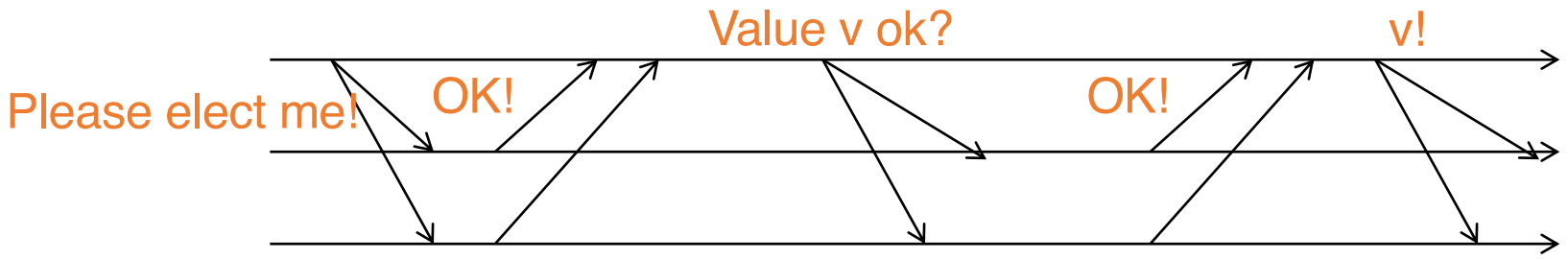
Phase 3 – Decision

- If leader hears a majority of OKs, it lets everyone know of the decision
- Recipients receive decision, log it on disk



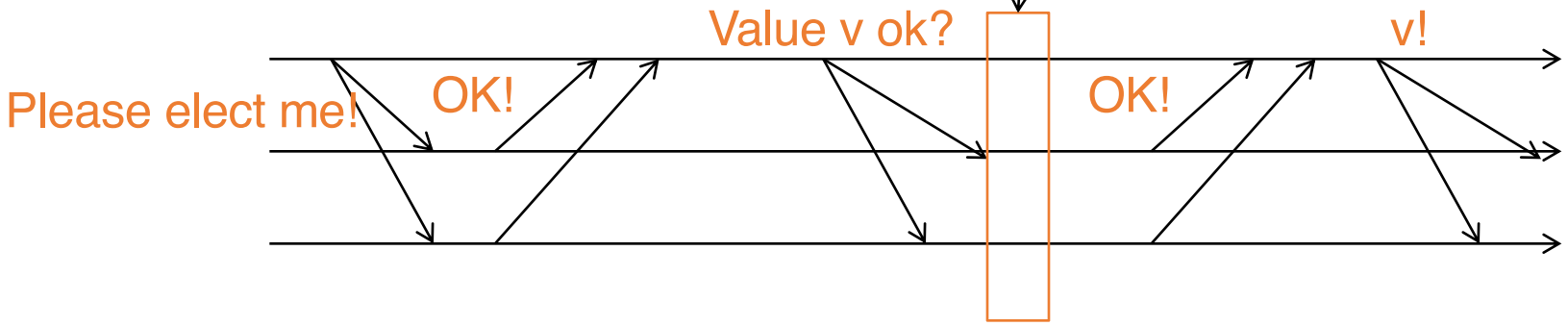
Which is the point of No-Return

- That is, when is consensus reached in the system



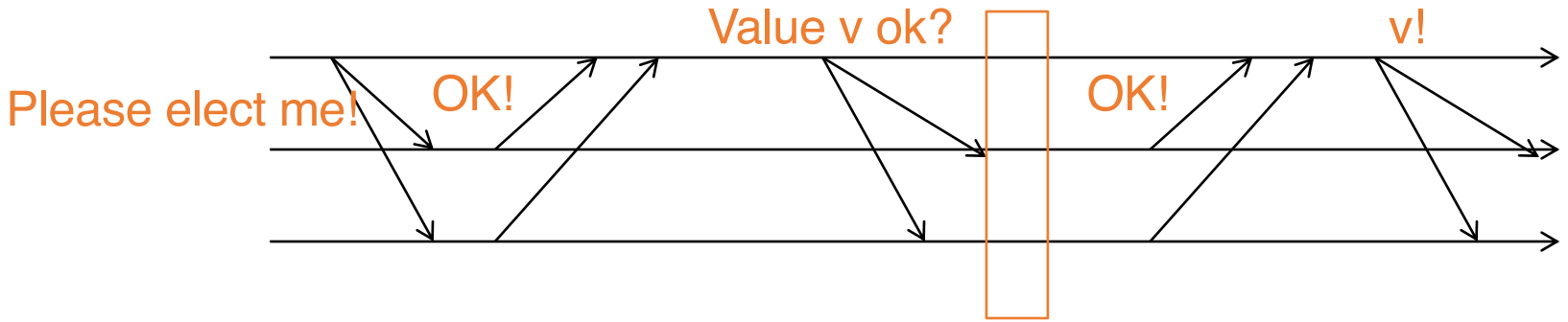
Which is the point of No-Return

- If/when a majority of processes hear proposed value and accept it (i.e., are about to/have respond(ed) with an OK!)
- Processes *may not know it yet*, but a decision has been made for the group
  - Even leader does not know it yet
- What if leader fails after that?
  - Keep having rounds until some round completes



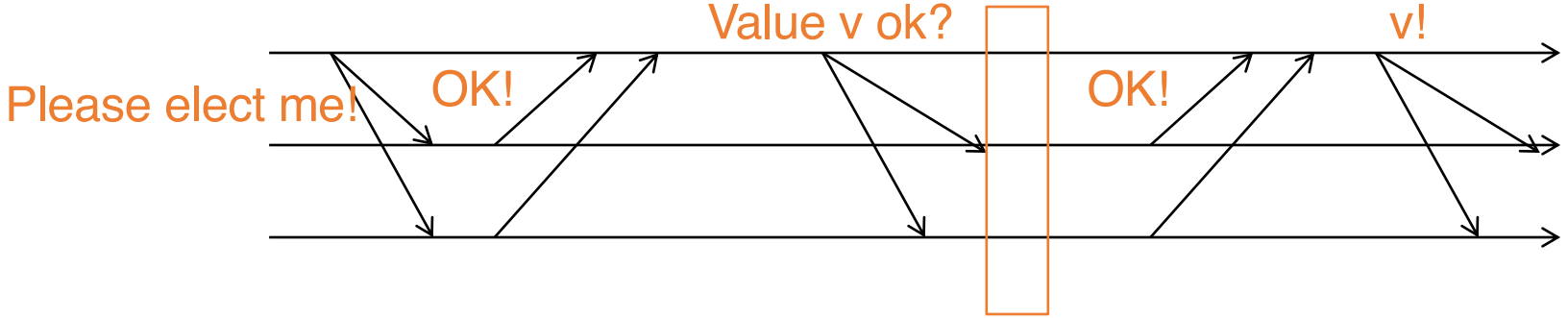
Safety

- If some round has a majority (i.e., quorum) hearing proposed value v' and accepting it (middle of Phase 2), then subsequently at each round either: 1) the round chooses v' as decision or 2) the round fails
- "Proof":
  - Potential leader waits for majority of OKs in Phase 1
  - At least one will contain v' (because two majorities always intersect)
  - It will choose to send out v' in Phase 2
- Success requires a majority, and any two majority sets intersect



What could go Wrong

- Process fails
  - Majority does not include it
  - When process restarts, it uses log to retrieve a past decision (if any) and past-seen ballot ids. Tries to know of past decisions.
- Leader fails
  - Start another round
- Messages dropped
  - If too flaky, just start another round
- Note that anyone can start a round any time
- Protocol may never end – tough luck, buddy!
  - Impossibility result not violated
  - If things go well sometime in the future, consensus reached



Summary

- Consensus is a very important problem
  - Equivalent to many important distributed computing problems that have to do with *reliability*
- Consensus is possible to solve in a synchronous system where message delays and processing delays are bounded
- Consensus is impossible to solve in an asynchronous system where these delays are unbounded
- Paxos protocol: widely used implementation of a safe, eventually-live consensus protocol for asynchronous systems
  - Paxos (or variants) used in Apache Zookeeper, Google's Chubby system, Active Disk Paxos, and many other cloud computing systems