The case of the Rotating Coordinator

Solving consensus with $\Diamond W$ (actually, $\Diamond S$)

- Asynchronous rounds
- Each round has a coordinator c
- $c_{id} = (r \, mod \, n) + 1$
- Each process p has an opinion $v_p \in \{0, 1\}$ (with a time of adoption t_p)
- Coordinator collects opinions to form a suggestion
- If they believe c to be correct, processes adopt its suggestion and make it their own opinion
- A suggestion adopted by a majority of processes is "locked"

One round, four phases

Phase 1

Each process, including c, sends its opinion timestamped r to c

One round, four phases

Phase 1

Each process, including c, sends its opinion timestamped r to c

Phase 2

c waits for first $\lceil n/2{+}1 \rceil$ opinions with timestamp r

c selects v, one of the most recently adopted opinions

v becomes c's suggestion for round r

c sends its suggestion to all

One round, four phases

Phase 1

Each process, including c, sends its opinion timestamped r to c

Phase 2

c waits for first $\lceil n/2+1 \rceil$ opinions with timestamp r

c selects v, one of the most recently adopted opinions

v becomes c's suggestion for round r

c sends its suggestion to all

Phase 3

Each p waits for a suggestion, or for failure detector to signal c is faulty

If p receives a suggestion, p adopts it as its new opinion and ACKs to c

Otherwise, p NACKs to c

One round, four phases

Phase 1

Each process, including c, sends its opinion timestamped r to c

Phase 2

c waits for first $\lceil n/2+1 \rceil$ opinions with timestamp r

c selects v, one of the most recently adopted opinions

v becomes c's suggestion for round r

c sends its suggestion to all

Phase 3

Each p waits for a suggestion, or for failure detector to signal c is faulty

If p receives a suggestion, p adopts it as its new opinion and ACKs to c

Otherwise, p NACKs to c

Phase 4

c waits for first $\lceil n/2+1 \rceil$ responses

if all ACKs, then c decides on v and sends DECIDE to all

if p receives DECIDE, then p decides on v

Consensus using $\Diamond S$

v_p := input bit; r := 0; t_p := 0; $state_p$:= undecided

while p undecided do

r := $r+1$

c := $(r \ mod \ n) + 1$

{phase 1: all processes send opinion to current coordinator}

p sends (p, r, v_p, t_p) to c

{phase 2: current coordinator gather a majority of opinions}

c waits for first $\lceil n/2 + 1 \rceil$ opinions (q, r, v_q, t_q)

c selects among them the value v_q with the largest t_q

c sends (c, r, v_q) to all

{phase 3: all processes wait for new suggestions from the current coordinator}

p waits until suggestion (c, r, v) arrives or $c \in \Diamond S_p$

if suggestion is received then {v_p := v; t_p := r; p sends (r, ACK) to c}

else p sends (r, NACK) to c

{phase 4: coordinator waits for majority of replies. If majority adopted the coordinator's suggestion, then coordinator sends request to decide}

c waits for first $\lceil n/2 + 1 \rceil$ (r, ACK) or (r, NACK)

if c receives $\lceil n/2 + 1 \rceil$ ACKs, then c sends (r, DECIDE, v) to all

when p delivers (r, DECIDE, v) then {p decides v; $state_p$:= decided}

$\Diamond S$ Consensus as Paxos

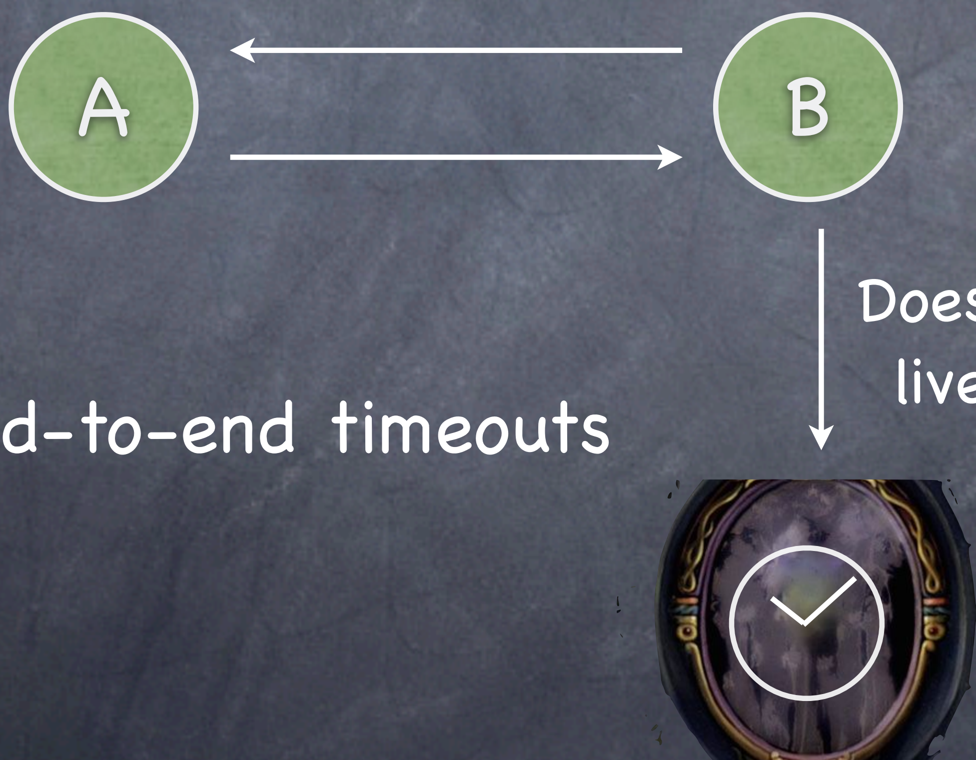
- All processes are acceptors
- In round r, node $(r \bmod n)+1$ serves both as a distinguished proposer and as a distinguished learner
- The round structure guarantees a unique proposal number
- The value that a proposer proposes when no value is chosen is not determined

Wait a second...

- Great to know that $\Diamond W$ can solve consensus...
- ...but that means I can't build $\Diamond W$!

Is there something interesting
I can do with FDs?

Failure detectors in practice



End-to-end timeouts

Failure Detector

Choosing your poison

Loooong timeout	Short timeout	Short kill timeout
Delayed detection	Loss of accuracy	Intrusive

Falcon

Leners et al.

Fast

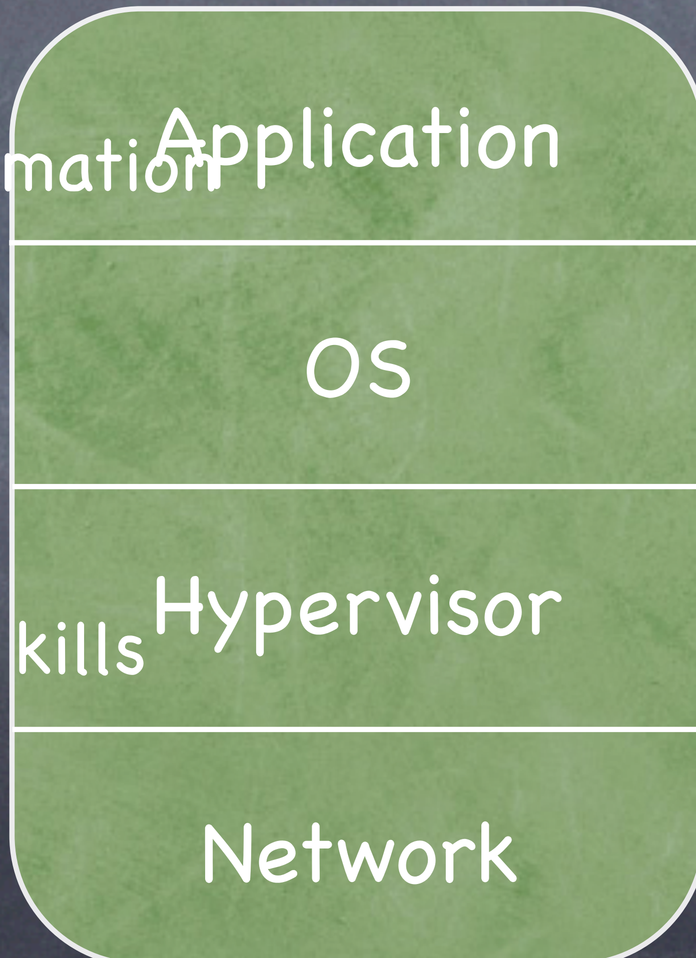
Accurate

Unobtrusive

End-to-end timeouts

There are more things in a process, Horatio,...

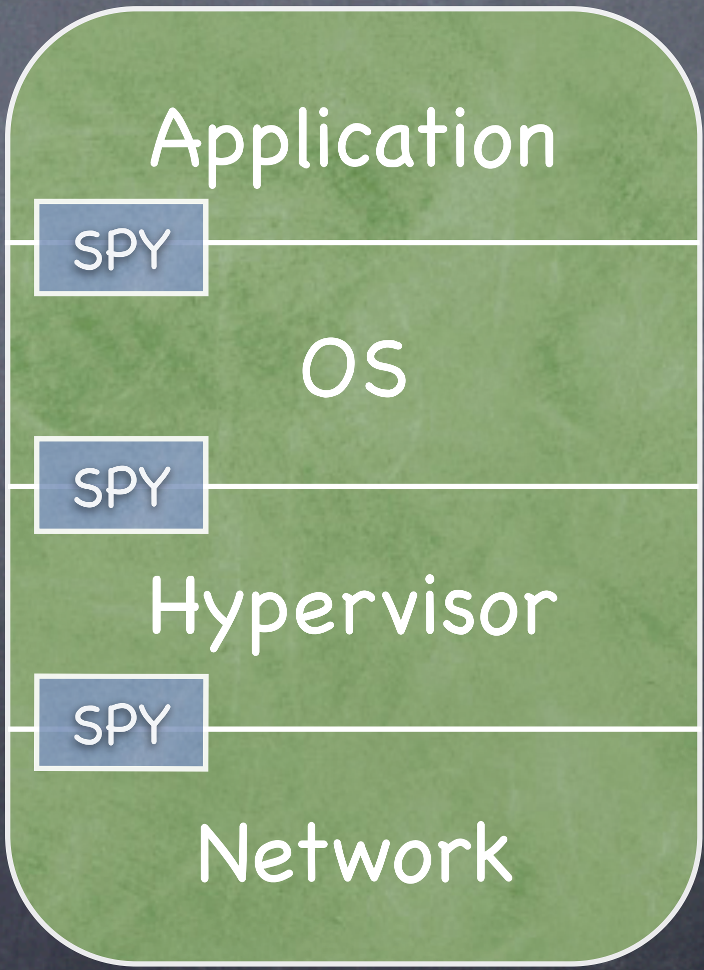
- Gather inside information
- Use local timeouts
- Use lethal force
- Avoid unnecessary kills



Design

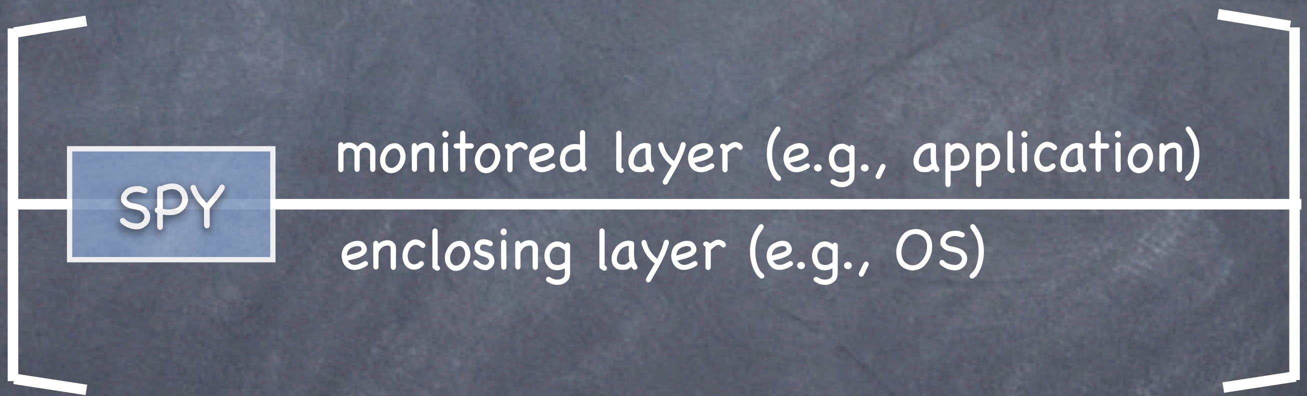
Use spies to get inside the layers

- What do spies do?
- How are spies coordinated?
- What happens in the corner cases?



What do spies do?

- A spy occupies two layers



Mission

- ☐ Gather inside information
- ☐ Kill, if necessary

Application Spy

- Gather inside information
  - Ask the application:
    - "Is the webserver processing HTTP requests?"
  - Ask the OS:
    - "Is the application in the process table?"
- KILL, if necessary

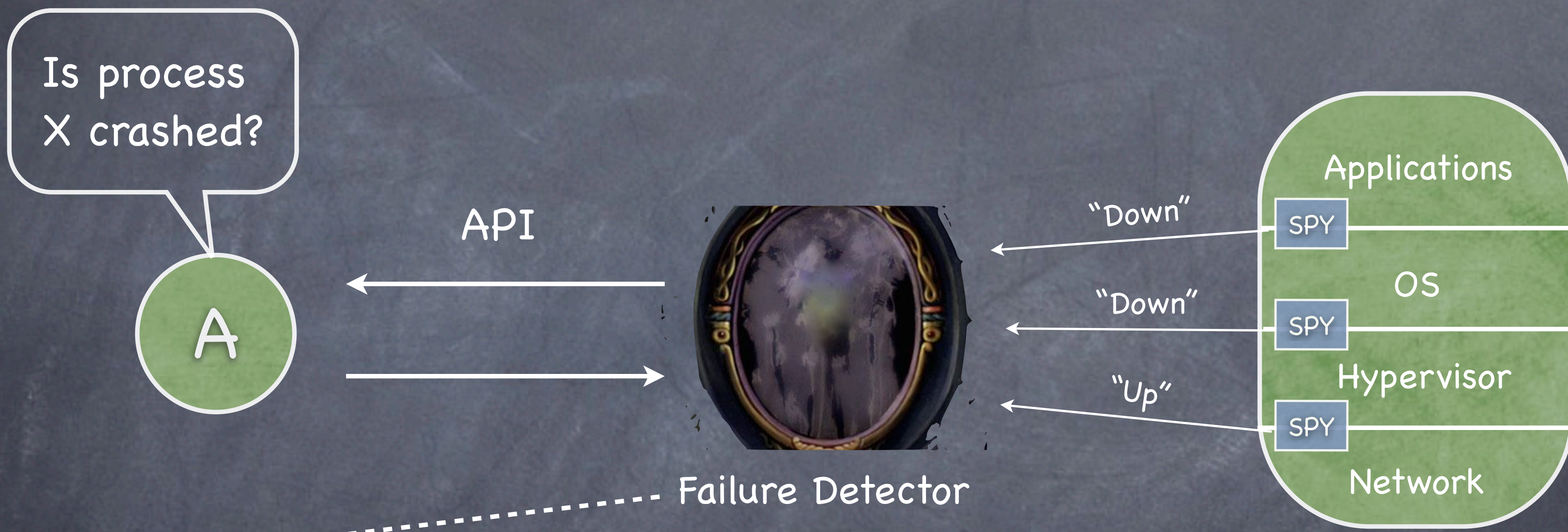
Weapon of choice: OS signals

OS Spy

- Gather inside information
  - Ask the OS:
    - "Are processes being scheduled?"
  - Ask the Hypervisor:
    - "Is the virtual machine active?"
- KILL, if necessary

Weapon of choice: VM termination

How are spies coordinated?



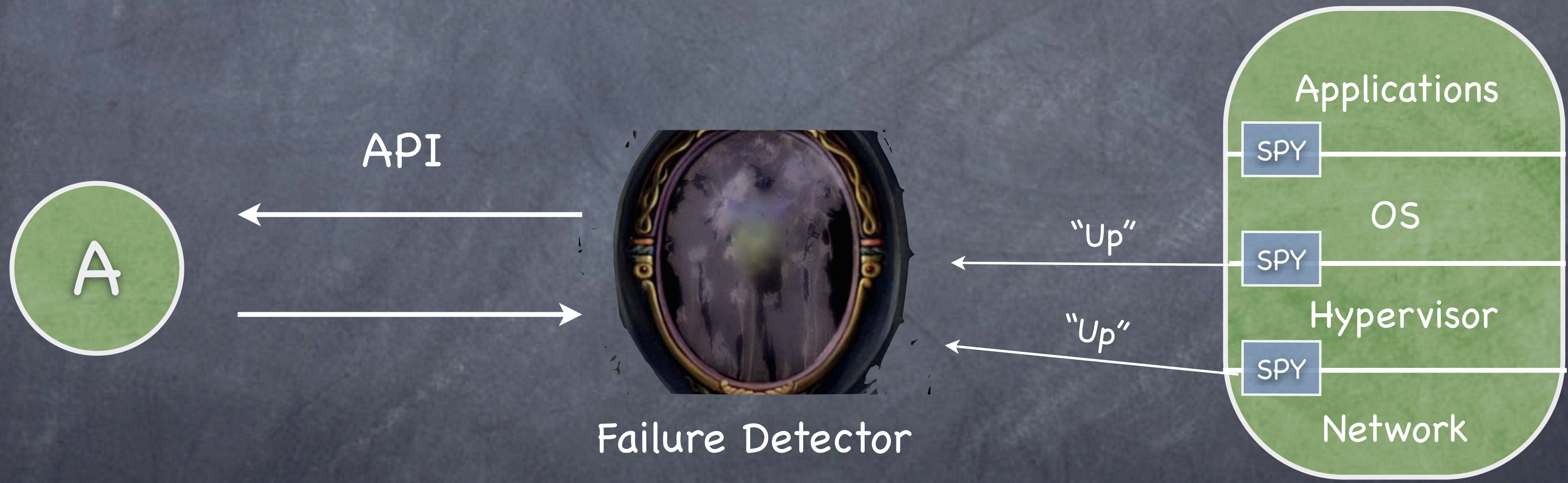
YES	if any spy says "Down"
NO	otherwise

inside information
+ callbacks

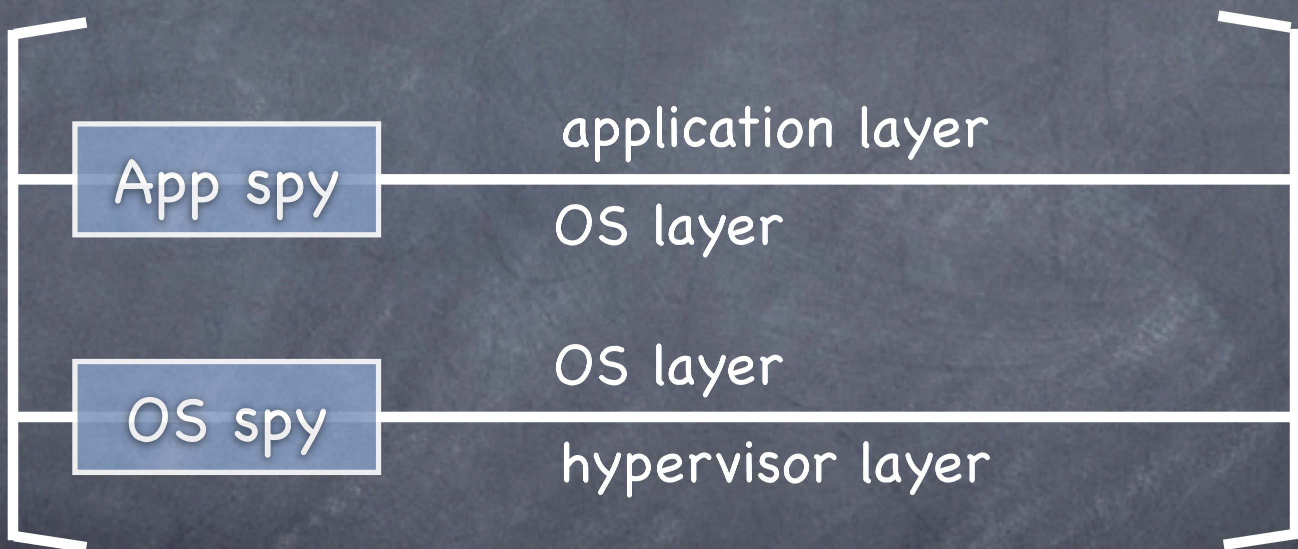
Fast detection

What are the corner cases?

Corner case 1: spy crashes



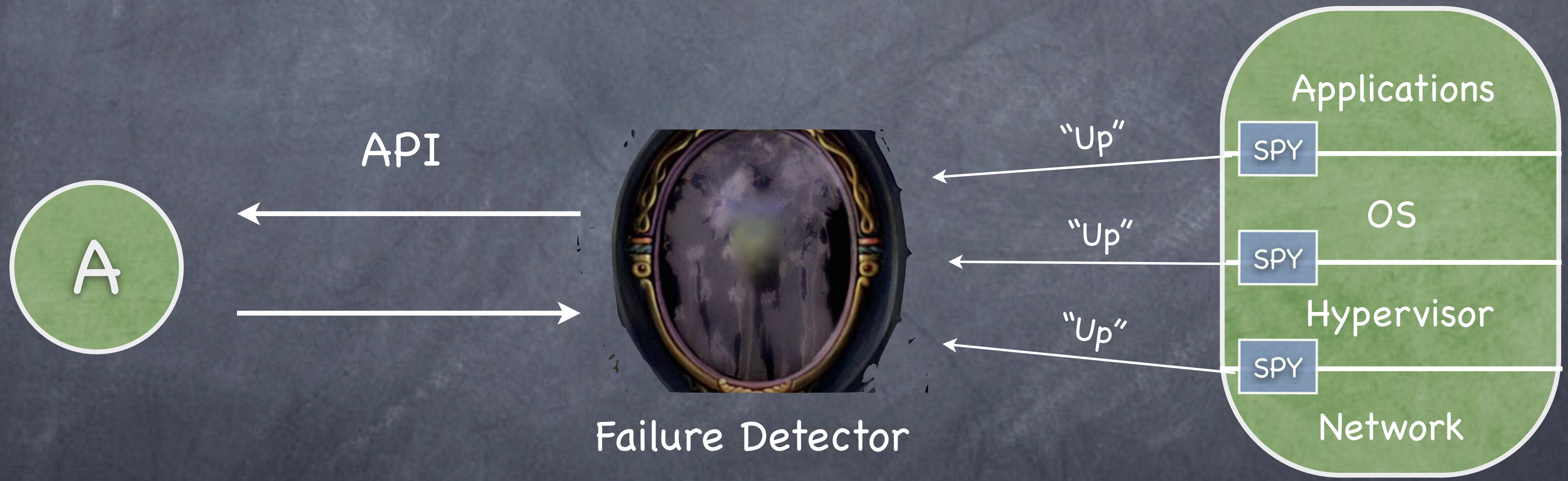
Lower spies monitor higher spies



App spy's enclosing layer is OS spy's monitored layer

What are the corner cases?

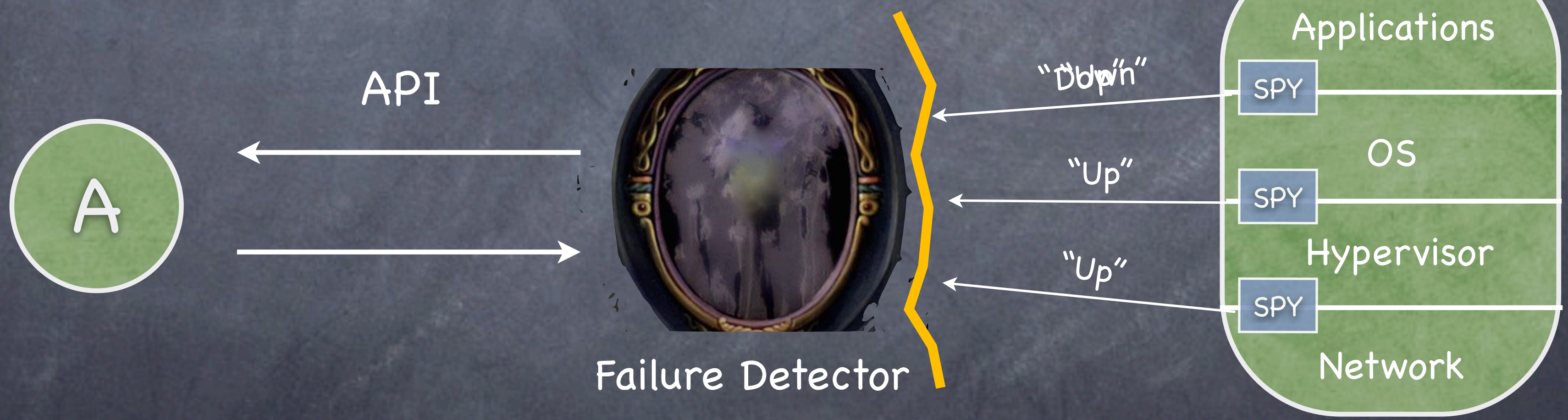
Corner case 2: spy does not detect failures



Fall back to a large end-to-end timeout backed by a remote kill

What are the corner cases?

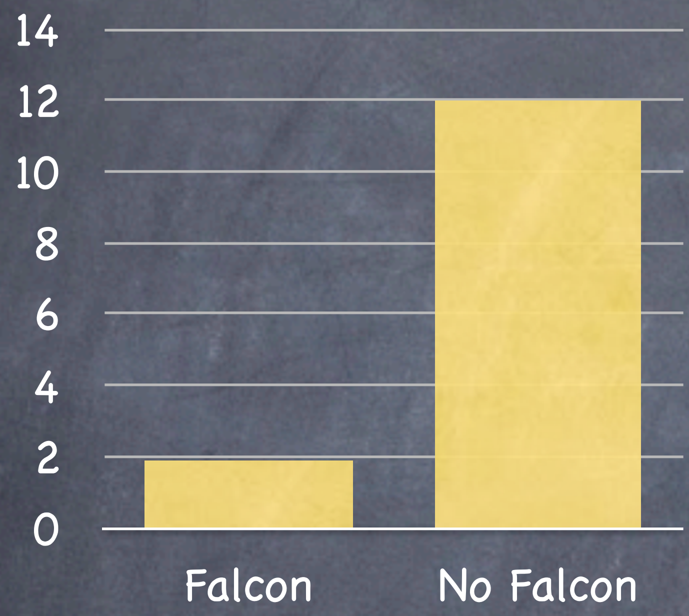
Corner case 3: network partitions



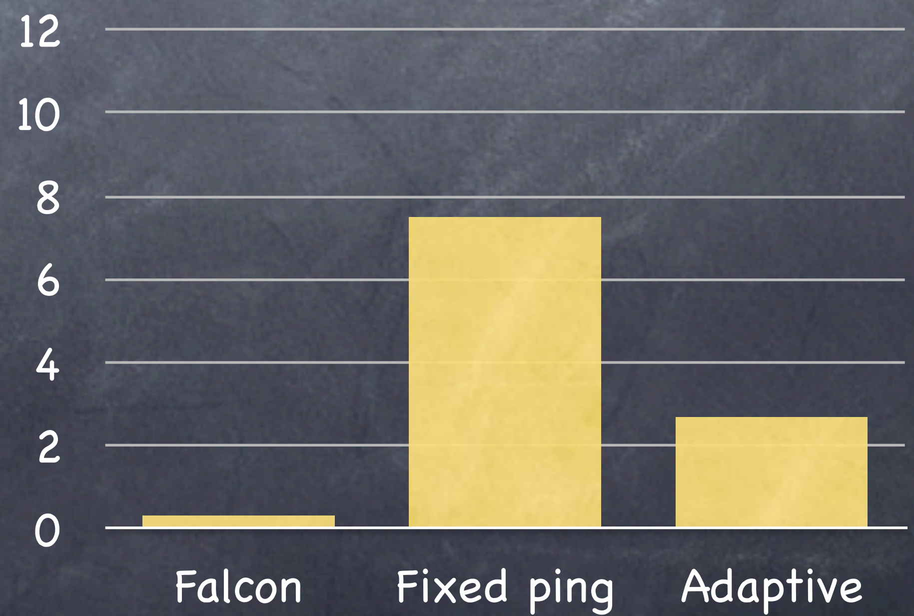
Communication necessary for accuracy

Falcon waits

Falcon: Fast



Median period of unavailability (sec) after ZooKeeper leader has a kernel error
(lower is better)



Median detection time (sec) of a kernel error
(lower is better)

Falcon: Accurate

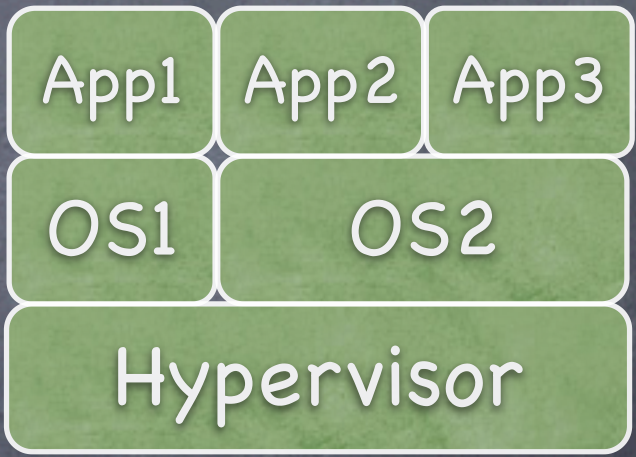
- A spy can confirm that a layer has crashed
- If a layer crashes, all the layers above it are crashed

Simplifies code

With Falcon, replicas initiate leader change only if the primary is dead

Falcon: Unobtrusive

- Kills smallest possible component
- Avoids unwarranted kills
  - e.g., hung system calls
- Uses few system resources
  - heaviest spy ≈ 3% CPU



FLP

What about the asynchronous model?

Theorem

There is no deterministic protocol that solves Consensus in a message-passing asynchronous system in which at most one process may fail by crashing

(Fisher, Lynch, and Paterson. Impossibility of distributed consensus with one faulty process. JACM, Vol. 32, no. 2, April 1985, pp. 374-382)

The Intuition

- In an asynchronous system, a process p cannot tell whether a non-responsive process q has crashed or it is just slow
- If p waits, it might do so forever
- If p decides, it may find out later that q came to a different decision

The Model - 1

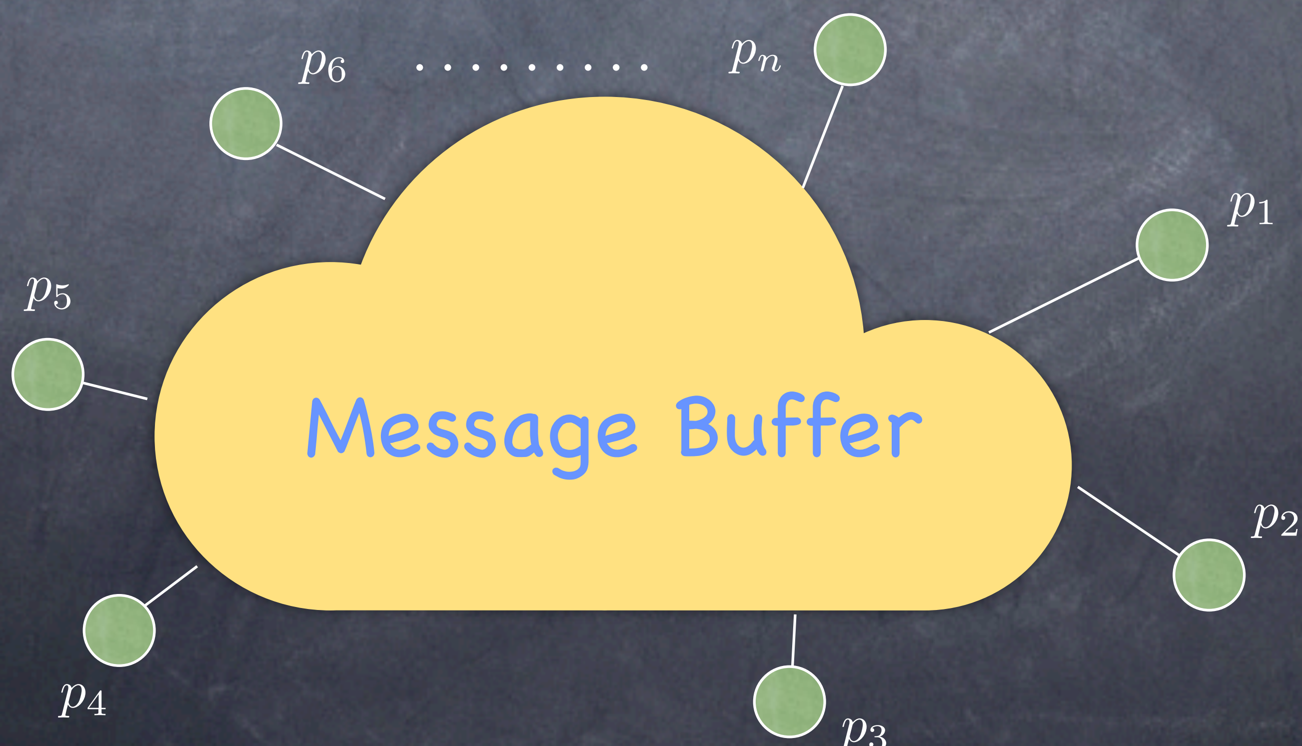
- n processes
- a message buffer

null message

message: (p, data, q) or λ

sender

receiver



The Model - 2

- An algorithm $\mathcal{A}$ is a sequence of steps
- Each step consists of two phases
  - Receive phase - some p removes from buffer $(x, data, p)$ or λ
  - Send phase - p changes its state; adds zero or more messages to buffer
- p can receive λ <u>even if there are messages for</u> p <u>in the buffer</u>

Assumptions

Liveness Assumption:

Every message sent will be eventually received if intended receiver tries infinitely often

One-time Assumption:

p sends m to q at most once

WLOG, process p_i can only propose a single bit b_i

Configurations

- A configuration C of $\mathcal{A}$ is a pair (s, M) where:
  - s is a function that maps each p_i to its local state
  - M is the set of messages in the buffer
- A step $e \equiv (p, m, \mathcal{A})$ is applicable to $C = (s, M)$ if and only if $m \in M \cup \{\lambda\}$. Note: $(p, \lambda, \mathcal{A})$ is always applicable to C
- $C' \equiv e(C)$ is the configuration resulting from applying e to C

Schedules

- A schedule S of $\mathcal{A}$ is a finite or infinite sequence of steps of $\mathcal{A}$
- A schedule S is applicable to a configuration C if and only if either
  - S is the empty schedule $S_{\perp}$ or
  - $S[1]$ is applicable to C ;
    $S[2]$ is applicable to $S[1](C)$; etc.
- If S is finite, $S(C)$ is the unique configuration obtained by applying S to C

Schedules and configurations

- A configuration C' is accessible from a configuration C if there exist a schedule S such that $C' = S(C)$
- C' is a configuration of $S(C)$ if $\exists S'$ prefix of S such that $S'(C) = C'$

Runs

- A run of $\mathcal{A}$ is a pair < I, S > where
  - I is an initial configuration
  - S is an infinite schedule of $\mathcal{A}$ applicable to I
- A run is partial if S is a finite schedule of $\mathcal{A}$
- A run is admissible if every process, except possibly one, takes infinitely many steps in S
- An admissible run is unacceptable if every process, except possibly one, takes infinitely many steps in S without deciding

Structure of the proof

- Show that, for any given consensus algorithm $\mathcal{A}$, there always exists an unacceptable run
- In fact, we will show an unacceptable run in which no process crashes!

Classifying Configurations

0-valent: A configuration C is 0-valent if some process has decided 0 in C, or if all configurations accessible from C are 0-valent

1-valent: A configuration C is 1-valent if some process has decided 1 in C, or if all configurations accessible from C are 1-valent

Bivalent: A configuration C is bivalent if some of the configurations accessible from it are 0-valent while others are 1-valent

Bivalent initial configurations happen

Lemma 1

There exists a bivalent initial configuration

Proof

- Suppose $\mathcal{A}$ solves consensus with 1 crash failure
- Let I_j be the initial configuration in which the first j b_i's are 1
- I_0 is 0-valent; I_n is 1-valent
- By contradiction, suppose no bivalent

Proof

- Suppose $\mathcal{A}$ solves consensus with 1 crash failure
- Let I_j be the initial configuration in which the first j b_i's are 1
- I_0 is 0-valent; I_n is 1-valent
- By contradiction, suppose no bivalent
- Let k be smallest index such that I_k is 1-valent
- Obviously, I_{k-1} is 0-valent
- Suppose p_k crashes before taking any step.
- Since $\mathcal{A}$ solves consensus even with one crash failure, there is a finite schedule S applicable to I_k that has no steps of p_k and such that some process decides in $S(I_k)$
- S is also applicable to I_{k-1}

CONTRADICTION

Commutativity Lemma

Lemma 2

Let S_1 and S_2 be schedules applicable to some configuration C, and suppose that the set of processes taking steps in S_1 is disjoint from the set of processes taking steps in S_2.

Then, $S_1; S_2$ and $S_2; S_1$ are both sequences applicable to C, and they lead to the same configuration.

Procrastination Lemma

Lemma 3

Let C be bivalent, and let e be a step applicable to C.

Then, there is a (possibly empty) schedule S not containing e such that $e(S(C))$ is bivalent

Proof Sketch - 1

- By contradiction, assume there is an e for which no such S exists
- Then, $e(C)$ is monovalent; WLOG assume 0-valent

Proof Sketch - 1

- By contradiction, assume there is an e for which no such S exists
- Then, $e(C)$ is monovalent; WLOG assume 0-valent

Proof Sketch - 1

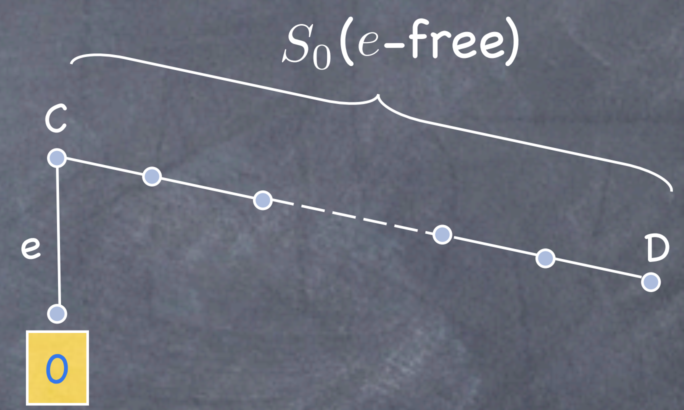
- By contradiction, assume there is an e for which no such S exists
- Then, $e(C)$ is monovalent; WLOG assume 0-valent

Mini Lemma:

There exists an e-free schedule S_0 such that $D = S_0(C)$

Proof Sketch - 1

- By contradiction, assume there is an e for which no such S exists
- Then, $e(C)$ is monovalent; WLOG assume 0-valent

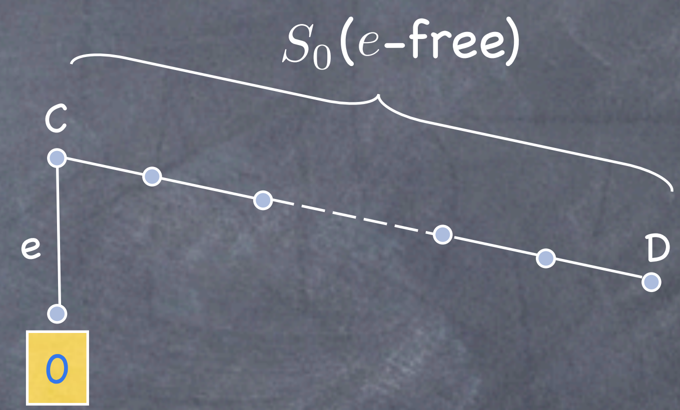


Mini Lemma:

There exists an e-free schedule S_0 such that $D = S_0(C)$

Proof Sketch - 1

- By contradiction, assume there is an e for which no such S exists
- Then, $e(C)$ is monovalent; WLOG assume 0-valent

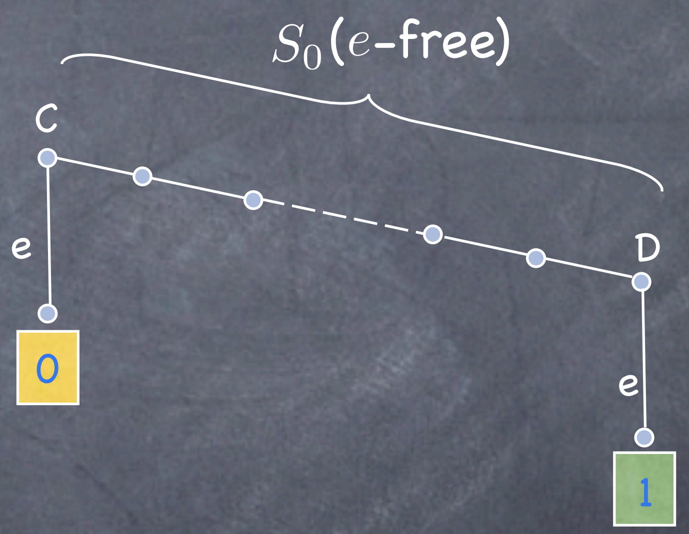


Mini Lemma:

There exists an e-free schedule S_0 such that $D = S_0(C)$ and $e(D)$ is 1-valent

Proof Sketch - 1

- By contradiction, assume there is an e for which no such S exists
- Then, $e(C)$ is monovalent; WLOG assume 0-valent



Mini Lemma:

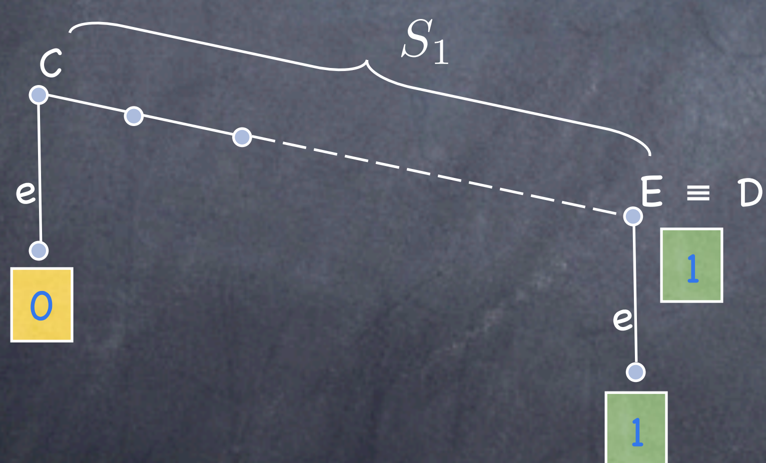
There exists an e-free schedule S_0 such that $D = S_0(C)$ and $e(D)$ is 1-valent

Proof Sketch- 2

Proof of mini Lemma.

Since C is bivalent, there exists a schedule S_1 such that $E = S_1(C)$ is 1-valent

Otherwise, let S_0 be the largest e-free prefix of S_1

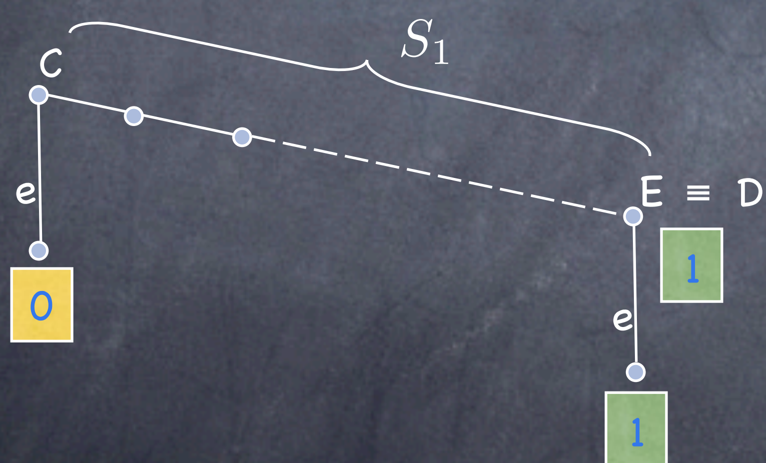


If S_1 is e-free, then D = E

Proof Sketch- 2

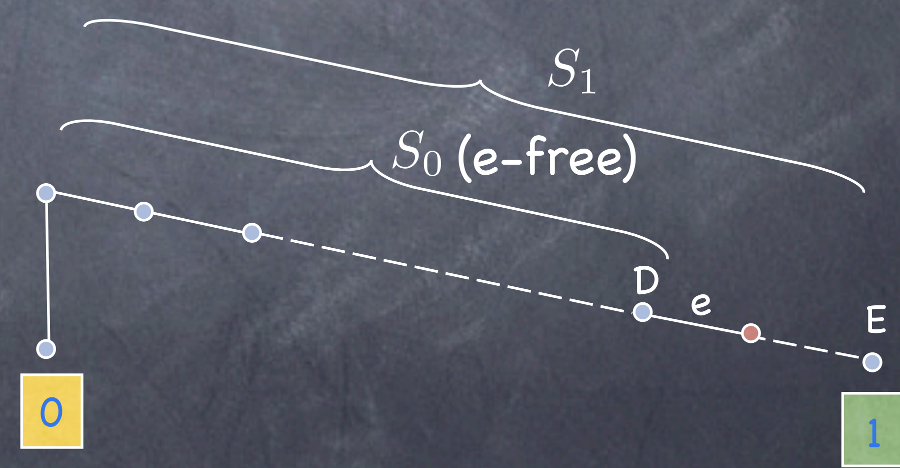
Proof of mini Lemma.

Since C is bivalent, there exists a schedule S_1 such that $E = S_1(C)$ is 1-valent



If S_1 is e-free, then D = E

Otherwise, let S_0 be the largest e-free prefix of S_1

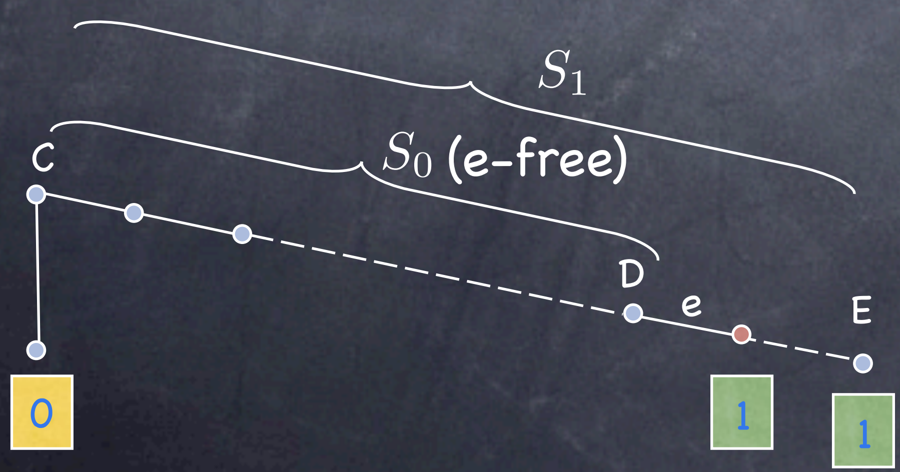


Consider configuration e(D).

Is it 0-valent? Bivalent? 1-valent?

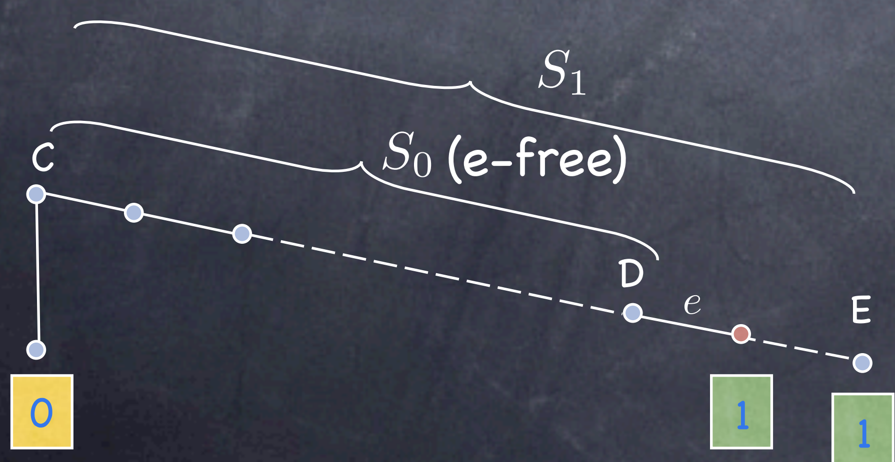
Proof Sketch - 3

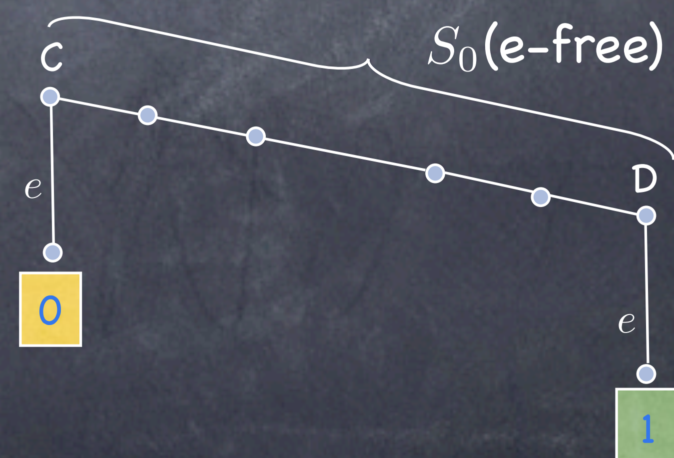
- By assumption, $e(D)$ cannot be bivalent (otherwise we would have proved the Procrastination Lemma with $S = S_0$)
- Since $e(D)$ is monovalent, E is accessible from $e(D)$, and E is 1-valent, then $e(D)$ is 1-valent $\square$



Proof Sketch - 3

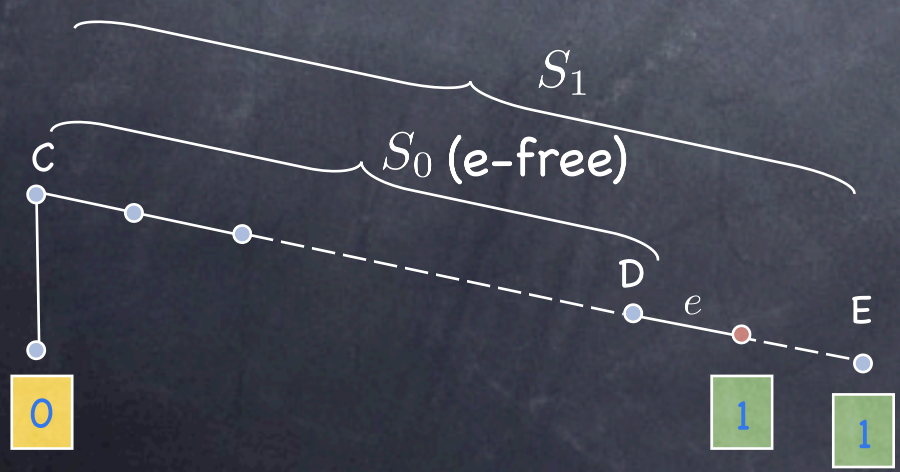
- By assumption, $e(D)$ cannot be bivalent (otherwise we would have proved the Procrastination Lemma with $S = S_0$)
- Since $e(D)$ is monovalent, E is accessible from $e(D)$, and E is 1-valent, then $e(D)$ is 1-valent $\square$





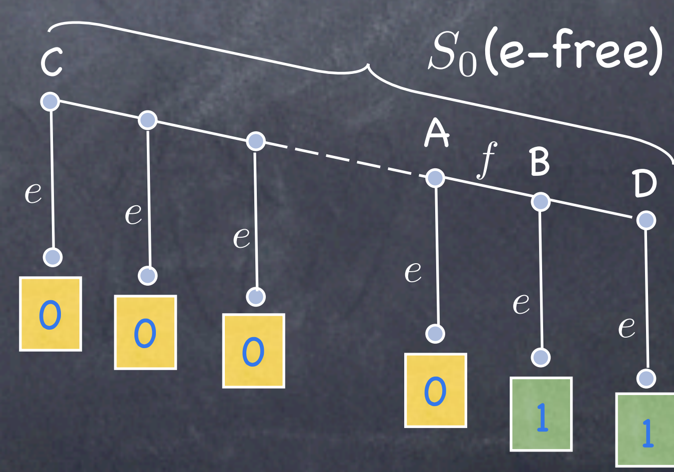
Proof Sketch - 3

- By assumption, $e(D)$ cannot be bivalent (otherwise we would have proved the Procrastination Lemma with $S = S_0$)
- Since $e(D)$ is monovalent, E is accessible from $e(D)$, and E is 1-valent, then $e(D)$ is 1-valent $\square$



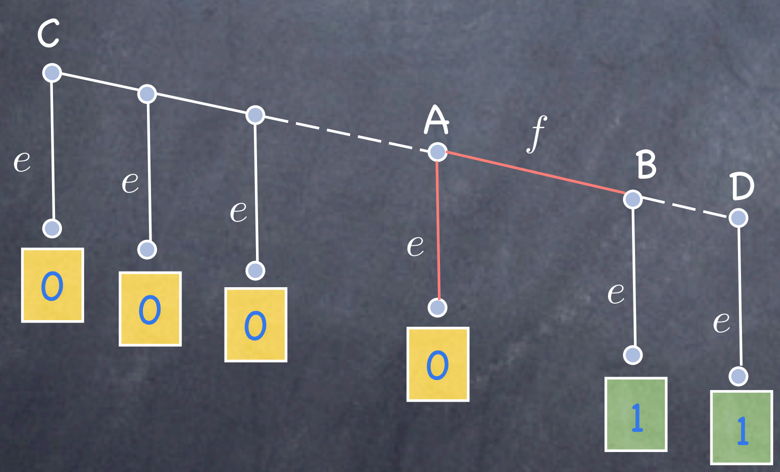
By the mini Lemma, on the "path" from C to D there must be two neighboring configurations A and B and a step f such that

- B = f(A)
- e(A) is 0-valent
- e(B) is 1-valent



Proof Sketch - 4

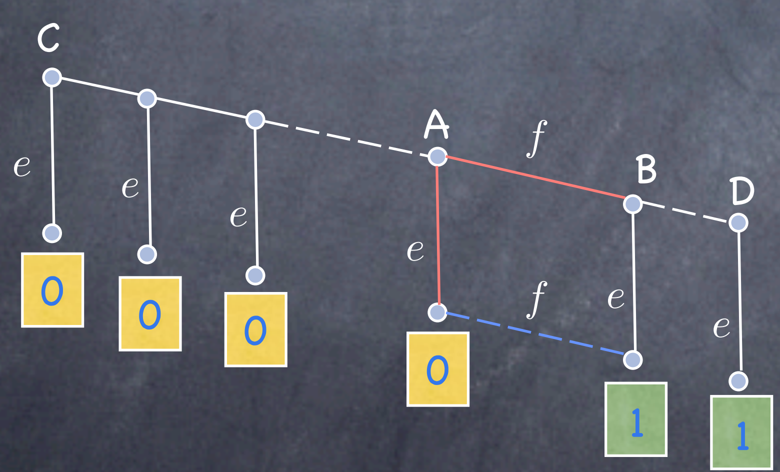
Consider now A and $B = f(A)$



- Claim: The same processes p must take steps e and f

Proof Sketch - 4

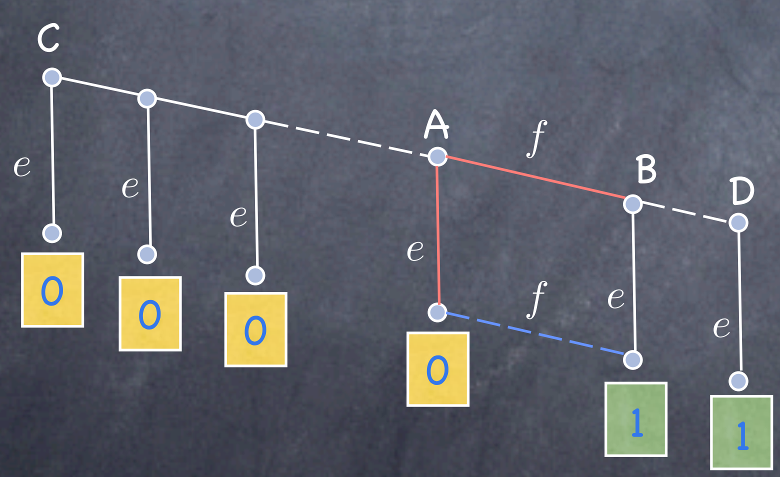
Consider now A and $B = f(A)$



- Claim: The same processes p must take steps e and f
  - Suppose not
  - By Commutativity lemma, $e(B) = e(f(A)) = f(e(A))$

Proof Sketch - 4

Consider now A and $B = f(A)$



- Claim: The same processes p must take steps e and f
  - Suppose not
  - By Commutativity lemma, $e(B) = e(f(A)) = f(e(A))$
  - Impossible since $e(B)$ is 1-valent and $e(A)$ is 0-valent

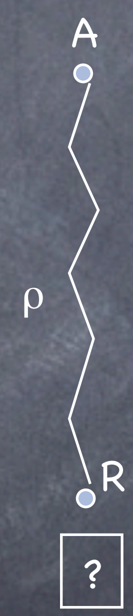
What happens if p fails?

Proof Sketch - 5

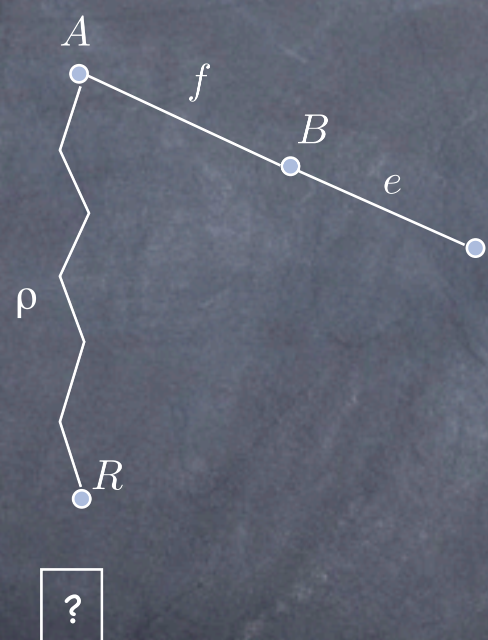
- Since our protocol tolerates a failure, there is a schedule ρ applicable to A such that:
  - ☐ R = ρ(A)
  - ☐ Some process decides in R
  - ☐ p does not take any steps in ρ

Proof Sketch - 5

- Since our protocol tolerates a failure, there is a schedule ρ applicable to A such that:
  - R = ρ(A)
  - Some process decides in R
  - p does not take any steps in ρ
- We show that the decision value in R can be neither 0 nor 1!



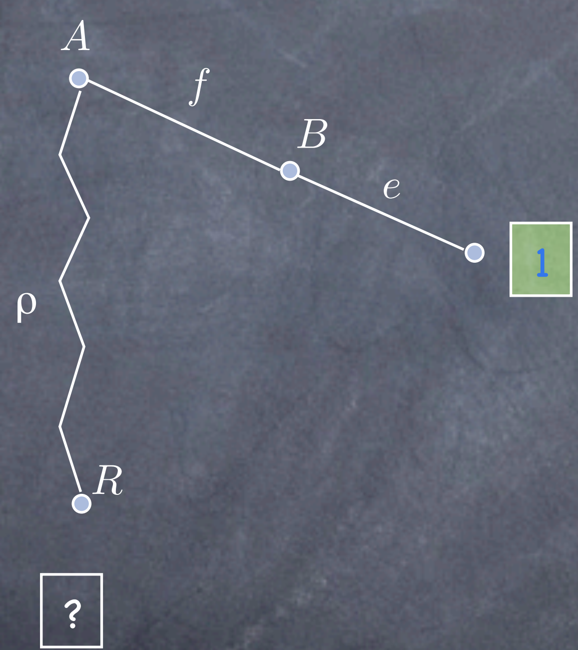
Proof Sketch - 6



Cannot be 0:

- Consider $e(B) = e(f(A))$

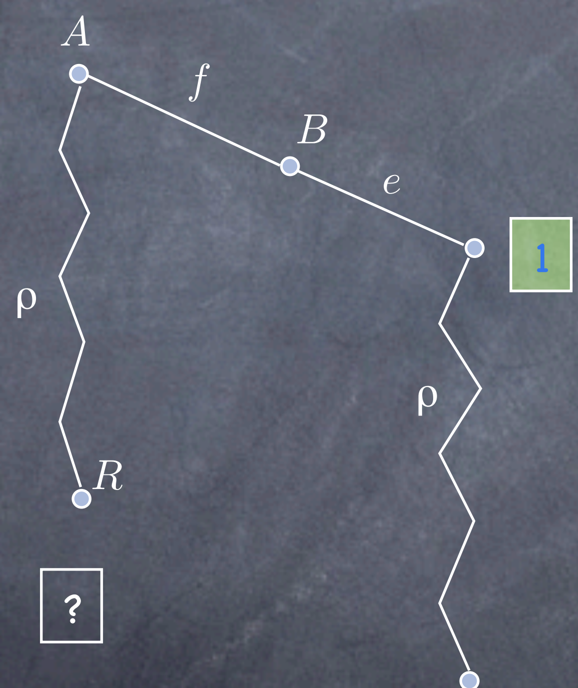
Proof Sketch - 6



Cannot be 0:

- Consider $e(B) = e(f(A))$
- By Mini Lemma, we know it is 1-valent

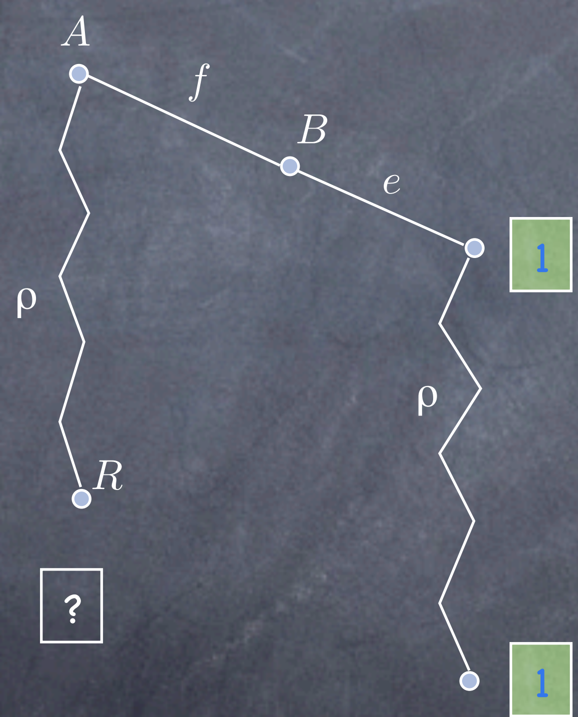
Proof Sketch - 6



Cannot be 0:

- Consider $e(B) = e(f(A))$
- By Mini Lemma, we know it is 1-valent
- Because it contains no steps of p , ρ is applicable to $e(B)$

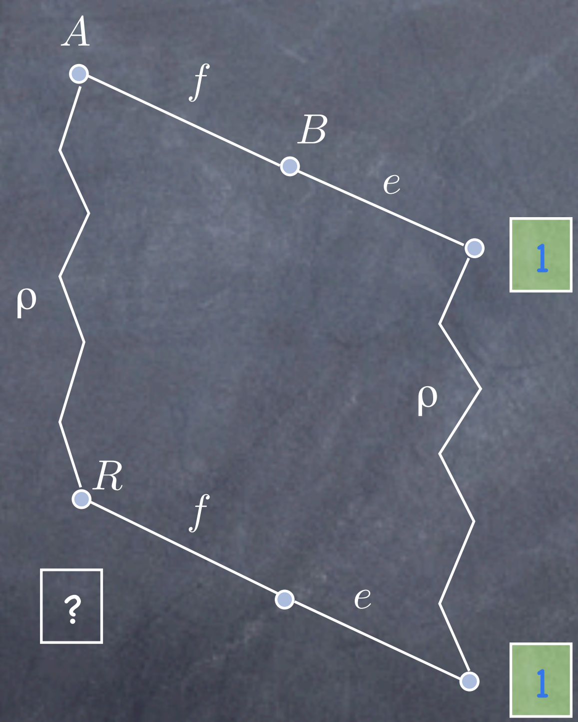
Proof Sketch - 6



Cannot be 0:

- Consider $e(B) = e(f(A))$
- By Mini Lemma, we know it is 1-valent
- Because it contains no steps of p , ρ is applicable to $e(B)$
- The resulting configuration is 1-valent

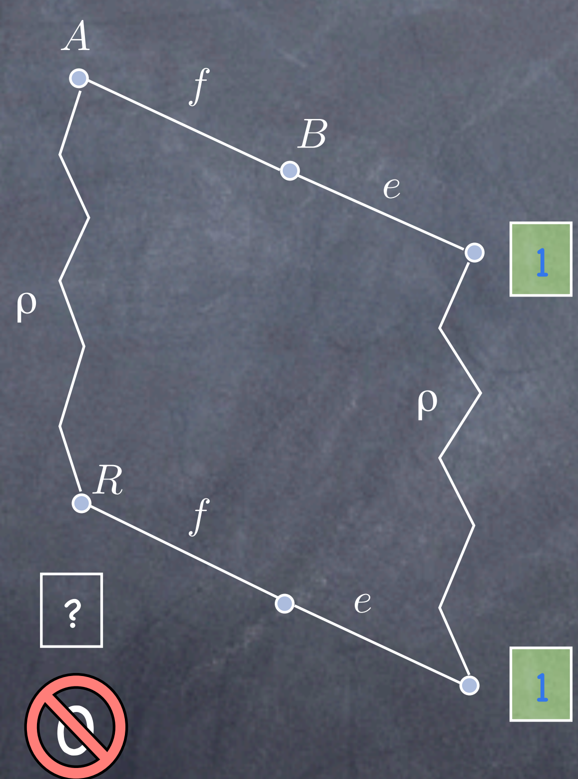
Proof Sketch - 6



Cannot be 0:

- Consider $e(B) = e(f(A))$
- By Mini Lemma, we know it is 1-valent
- Because it contains no steps of p , ρ is applicable to $e(B)$
- The resulting configuration is 1-valent
- By Commutativity Lemma $\rho(e(f(A))) = e(f(\rho(A))) = e(f(R))$

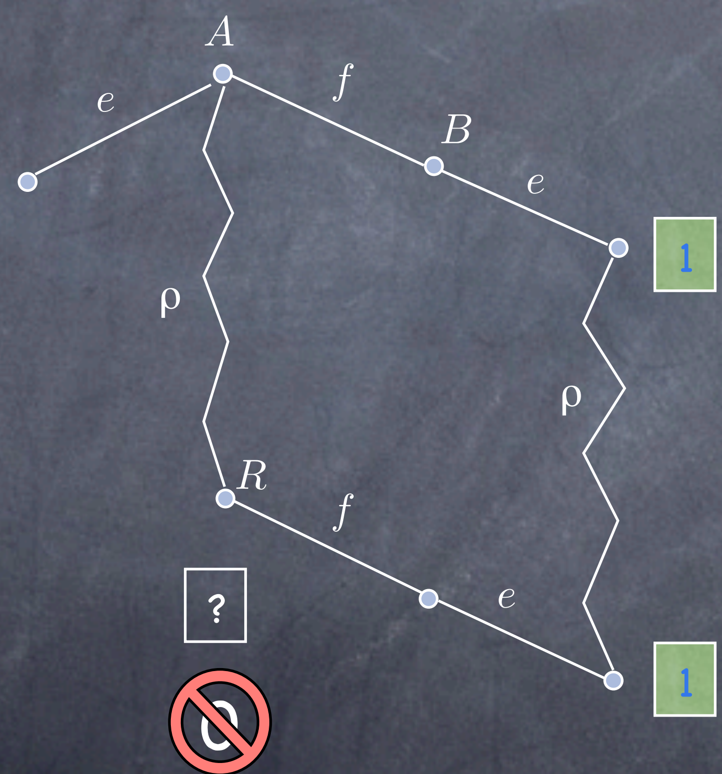
Proof Sketch - 6



Cannot be 0:

- Consider $e(B) = e(f(A))$
- By Mini Lemma, we know it is 1-valent
- Because it contains no steps of p , ρ is applicable to $e(B)$
- The resulting configuration is 1-valent
- By Commutativity Lemma $\rho(e(f(A))) = e(f(\rho(A))) = e(f(R))$
- Since $\rho(e(B))$ is accessible from R , and $\rho(e(B))$ is 1-valent, R cannot be 0-valent

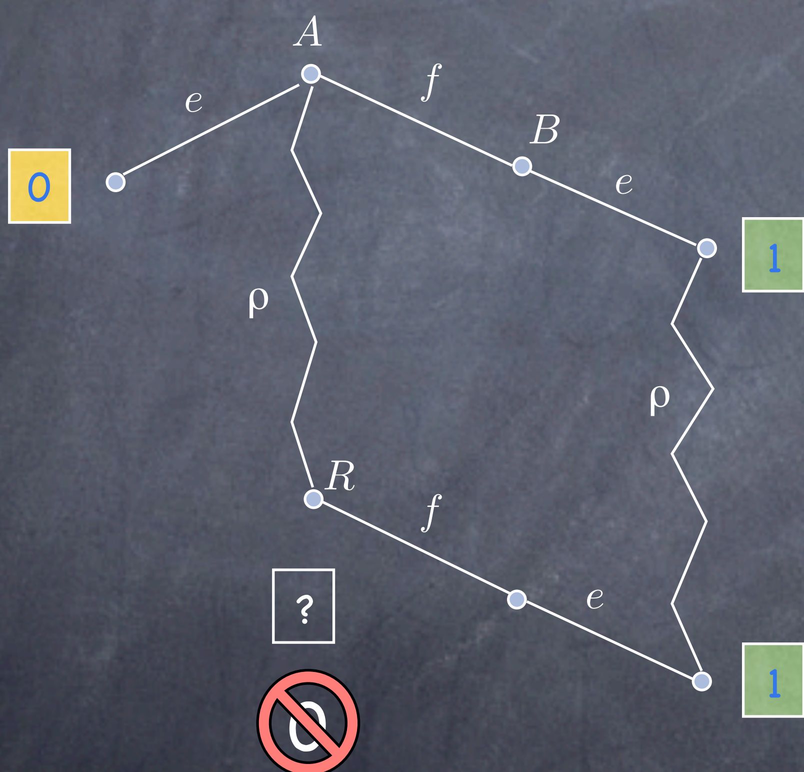
Proof Sketch - 7



Cannot be 1:

- Consider $e(A)$

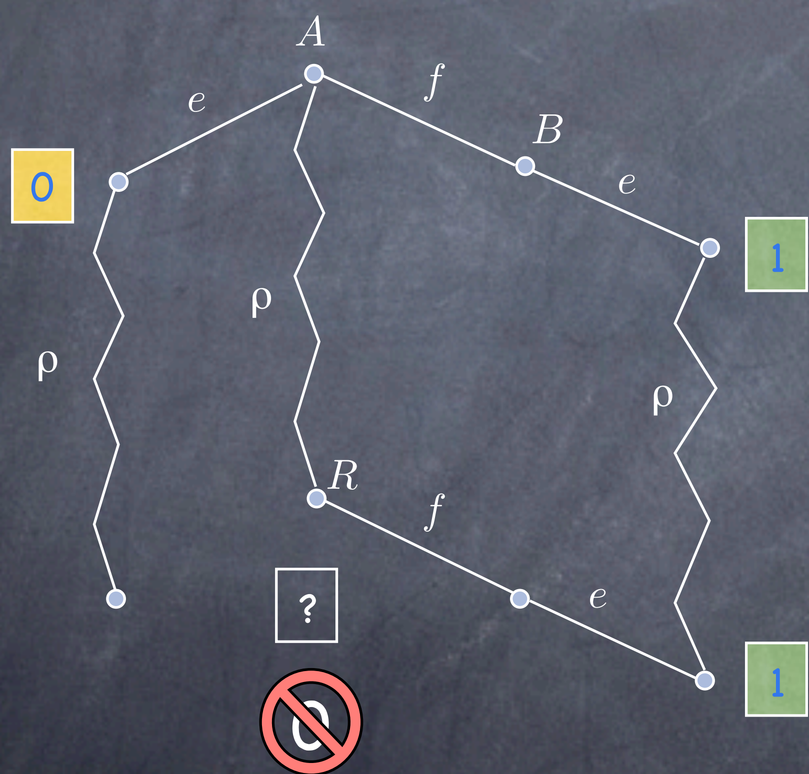
Proof Sketch - 7



Cannot be 1:

- Consider $e(A)$
- By construction, it is 0-valent

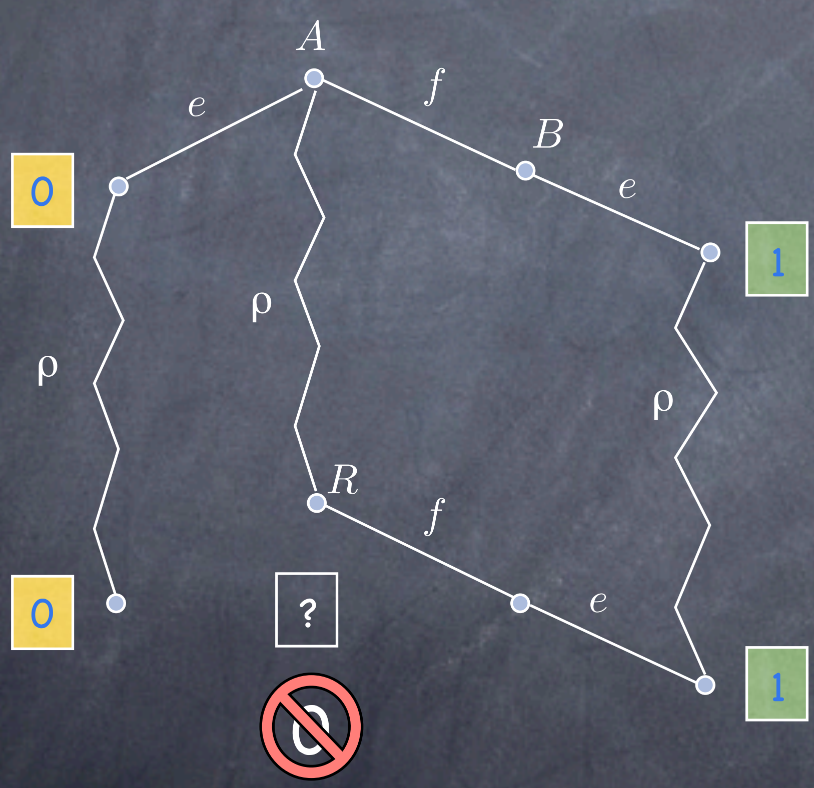
Proof Sketch - 7



Cannot be 1:

- Consider $e(A)$
- By construction, it is 0-valent
- Because it contains no steps of p , ρ is applicable to $e(A)$

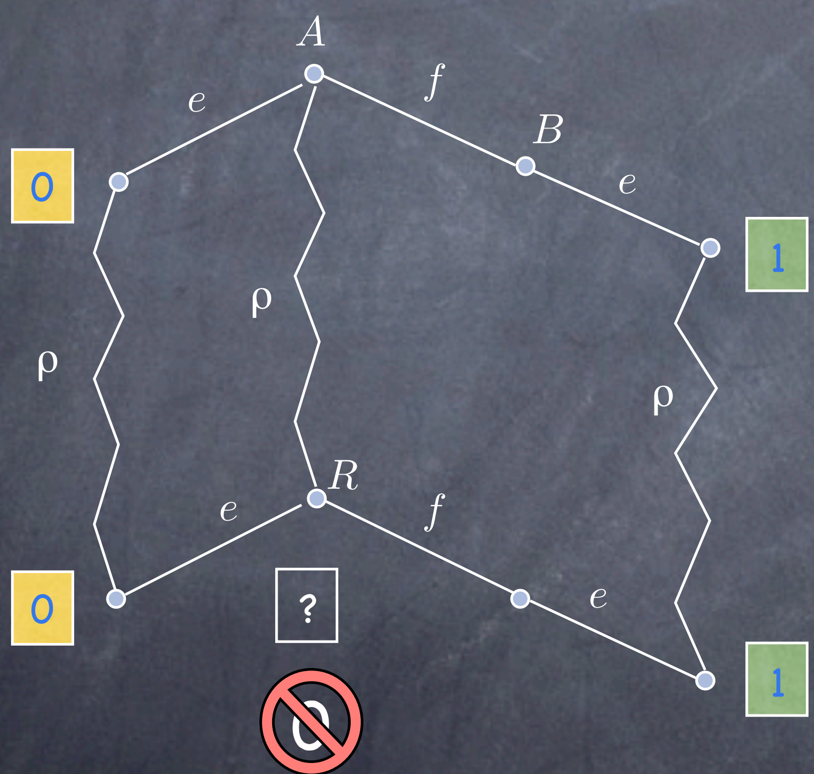
Proof Sketch - 7



Cannot be 1:

- Consider $e(A)$
- By construction, it is 0-valent
- Because it contains no steps of p , ρ is applicable to $e(A)$
- The resulting configuration is 0-valent

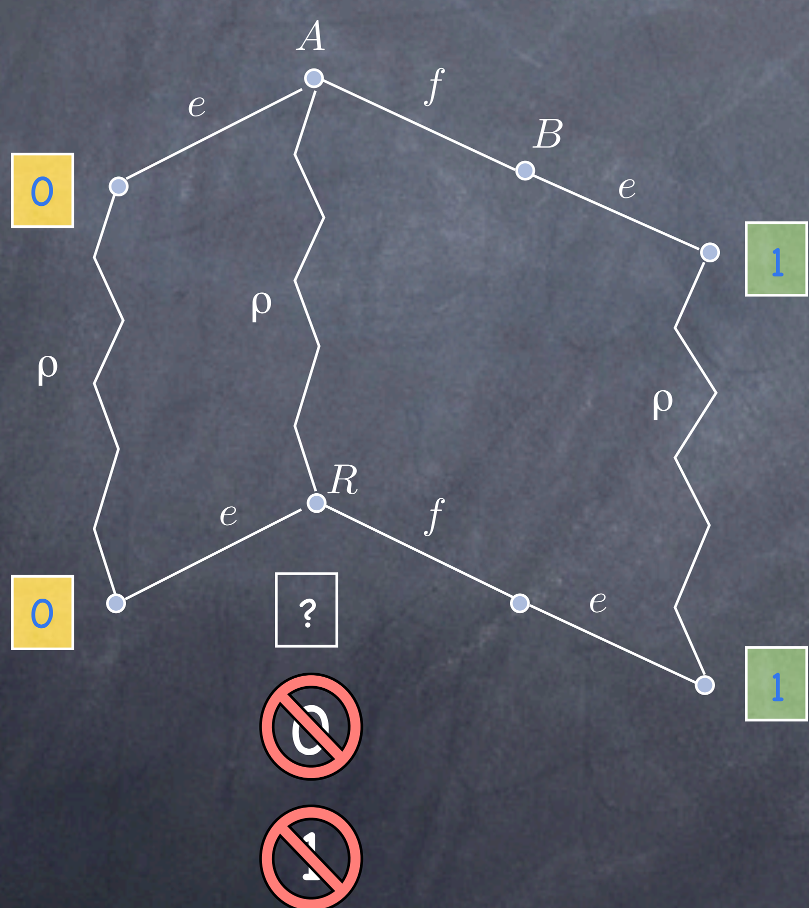
Proof Sketch - 7



Cannot be 1:

- Consider $e(A)$
- By construction, it is 0-valent
- Because it contains no steps of p , ρ is applicable to $e(A)$
- The resulting configuration is 0-valent
- By Commutativity Lemma $\rho(e(A)) = e(\rho(A)) = e(R)$

Proof Sketch - 7



Cannot decide in R: contradiction

Cannot be 1:

- Consider $e(A)$
- By construction, it is 0-valent
- Because it contains no steps of p , ρ is applicable to $e(A)$
- The resulting configuration is 0-valent
- By Commutativity Lemma $\rho(e(A)) = e(\rho(A)) = e(R)$
- Since $\rho(e(A))$ is accessible from R, and $\rho(e(A))$ is 0-valent, R cannot be 1-valent

Proving the FLP Impossibility Result

Theorem

There is no deterministic protocol that solves Consensus in a message-passing asynchronous system in which at most one process may fail by crashing

- By Lemma 1, there exists an initial bivalent configuration I_{biv}
- Consider any ordering $p_{l_1}, \ldots, p_{l_n}$ of $p_1, \ldots, p_n$
- Pick any applicable step $e_1 = (p_{l_1}, m_1)$
- Apply Procrastination lemma to obtain another bivalent configuration

$$C^1_{biv} = e_1(S_1(I_{biv}))$$

- Pick a step $e_2 = (p_{l_2}, m_2)$ applicable to C^1_{biv}
- Apply Procrastination lemma to obtain another bivalent configuration
- Continue as before in a round-robin fashion. How do we choose a step?
- We have built an unacceptable run!

How can one get around FLP?

Weaken the problem

- Weaken termination
  - use randomization to terminate with arbitrarily high probability
  - guarantee termination only during periods of synchrony
- Weaken agreement
  - ε - agreement
    - real-valued inputs and outputs
    - agreement within real-valued small positive tolerance ε
  - k-set agreement
    - Agreement: In any execution, there is a subset W of the set of input values, |W| =k, s.t. all decision values are in W
    - Validity: In any execution, any decision value for any process is the input value of some process

How can one get around FLP?

Constrain input values

- Characterize the set of input values for which agreement is possible

Strengthen the system model

- Introduce failure detectors to distinguish between crashed processes and very slow processes