Unreliable Failure Detectors for Reliable Distributed Systems

A different approach

- Augment the asynchronous model with an unreliable failure detector for crash failures
- Define failure detectors in terms of abstract properties, not specific implementations
- Identify classes of failure detectors that allow to solve Consensus

The Model

General

- asynchronous system
- processes fail by crashing
- a failed process does not recover

Failure Detectors

- outputs set of processes that it currently suspects to have crashed
- the set may be different for different processes

Completeness

Strong Completeness Eventually every process that crashes is permanently suspected by every correct process

Weak Completeness Eventually every process that crashes is permanently suspected by some correct process

Accuracy

Strong Accuracy

No correct process is ever suspected

Weak Accuracy

Some correct process is never suspected

Accuracy

Strong Accuracy

No correct process is ever suspected

Weak Accuracy

Some correct process is never suspected

Eventual Strong Accuracy

There is a time after which no correct process is ever suspected

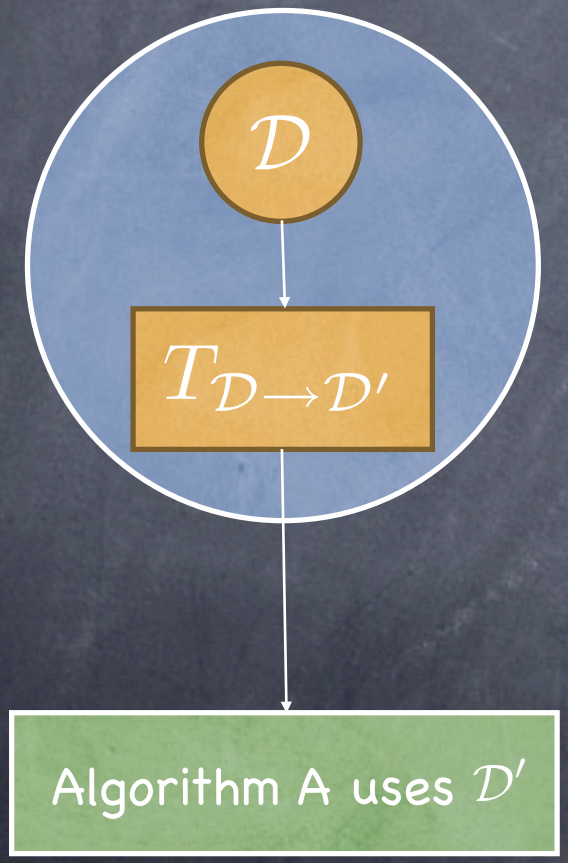
Eventual Weak Accuracy

There is a time after which some correct process is never suspected

Failure detectors

Completeness	Accuracy			
	Strong	Weak	Eventual strong	Eventual weak
Strong	Perfect P	Strong S	$\Diamond P$	$\Diamond S$
Weak	Quasi Q	Weak W	$\Diamond Q$	$\Diamond W$

Reducibility



$T_{\mathcal{D}\to\mathcal{D}'}$ transforms failure detector $\mathcal{D}$ into failure detector $\mathcal{D}'$

If we can transform $\mathcal{D}$ into $\mathcal{D}'$ then we say that $\mathcal{D}$ is stronger than $\mathcal{D}'$ ($\mathcal{D} \geq \mathcal{D}'$) and that $\mathcal{D}'$ is reducible to $\mathcal{D}$

If $\mathcal{D} \geq \mathcal{D}'$ and $\mathcal{D}' \geq \mathcal{D}$ then we say that $\mathcal{D}$ and $\mathcal{D}'$ are equivalent:

$$\mathcal{D} \equiv \mathcal{D}'$$

Simplify, Simplify, Simplify!

H.D. Thoreau

- All weakly complete failure detectors are reducible to strongly complete failure detectors

$$P \geq Q, \quad S \geq W, \quad \Diamond P \geq \Diamond Q, \quad \Diamond S \geq \Diamond W$$

Simplify, Simplify, Simplify!

H.D. Thoreau

- All weakly complete failure detectors are reducible to strongly complete failure detectors

$$P \geq Q, \quad S \geq W, \quad \Diamond P \geq \Diamond Q, \quad \Diamond S \geq \Diamond W$$

- All strongly complete failure detectors are reducible to weakly complete failure detectors (!)

$$Q \geq P, \quad W \geq S, \quad \Diamond Q \geq \Diamond P, \quad \Diamond W \geq \Diamond S$$

Weakly and strongly complete failure detectors are equivalent!

From Weak Completeness to Strong Completeness

Every process p executes the following:

$output_p$:= 0

cobegin

|| Task 1: repeat forever

{ p queries its local failure detector module $\mathcal{D}_p$ }

$suspects_p := \mathcal{D}_p$

send $(p, suspects_p)$ to all

|| Task 2: when receive$(q, suspects_q)$ from some q

$output_p := (output_p \cup suspects_q) - \{q\}$

coend

The Theorems

Theorem 1 In an asynchronous system with W, consensus can be solved as long as $f \leq n-1$

The Theorems

Theorem 1 In an asynchronous system with W, consensus can be solved as long as $f \leq n-1$

Theorem 2 There is no f-resilient consensus protocol using $\Diamond P$ for $f \geq n/2$

The Theorems

Theorem 1 In an asynchronous system with W, consensus can be solved as long as $f \leq n-1$

Theorem 2 There is no f-resilient consensus protocol using $\Diamond P$ for $f \geq n/2$

Theorem 3 In asynchronous systems in which processes can use $\Diamond W$, consensus can be solved as long as $f < n/2$

The Theorems

Theorem 1 In an asynchronous system with W, consensus can be solved as long as $f \leq n-1$

Theorem 2 There is no f-resilient consensus protocol using $\Diamond P$ for $f \geq n/2$

Theorem 3 In asynchronous systems in which processes can use $\Diamond W$, consensus can be solved as long as $f < n/2$

Theorem 4 A failure detector can solve consensus only if it satisfies weak completeness and eventual weak accuracy–i.e. $\Diamond W$ is the weakest failure detector

Solving consensus using S

- S: Strong Completeness, Weak Accuracy
  - at least some correct process c is never suspected
- Each process p has its own failure detector
- Input values are chosen from the set {0,1}

Notation

We introduce the operators $\oplus, \star, \otimes$

They operate element-wise on vectors whose entries have values from the set $\{0, 1, \perp\}$

$$v \star \perp = v \quad \perp \star v = v$$
$$v \star v = \perp \quad \perp \star \perp = \perp$$

$$v \oplus \perp = v \quad \perp \oplus v = v$$
$$v \oplus v = v \quad \perp \oplus \perp = \perp$$

$$v \otimes \perp = \perp \quad \perp \otimes v = \perp$$
$$v \otimes v = v \quad \perp \otimes \perp = \perp$$

Given two vectors A and B, we write $A \leq B$ if

$A[i] \neq \perp$ implies $B[i] \neq \perp$

Solving Consensus using any $\mathcal{D} \in S$

1: V_p := $(\perp, \ldots, \perp, v_p, \perp, \ldots, \perp)$ {p's estimate of the proposed values}

2: Δ_p:= $(\perp, \ldots, \perp, v_p, \perp, \ldots, \perp)$

3: {phase 1} {asynchronous rounds r_p, $1 \leq r_p \leq n-1$}

4: for r_p := 1 to $n-1$

5: send (r_p, Δ_p, p) to all

6: wait until [$\forall q$: received (r_p, Δ_q, q) or $q \in \mathcal{D}_p$] {query the failure detector}

7: O_p := V_p

8: V_p := $V_p \oplus (\oplus_{q\ received}\ \Delta_q)$

9: Δ_p := $V_p \star O_p$ {value is only echoed the first time it is seen}

10: {phase 2}

11: send (r_p, V_p, p) to all

12: wait until [$\forall q$: received (r_p, V_q, q) or $q \in \mathcal{D}_p$]

13: V_p := $\otimes_{q\ received}\ V_q$ {computes the "intersection", including V_p}

14: {phase 3}

15: decide on leftmost non-$\perp$ coordinate of V_p

A useful Lemma

1: V_p := ($\perp$, …, $\perp$, v_p, $\perp$, …, $\perp$) {p's estimate of the proposed values}
2: Δ_p := ($\perp$, …, $\perp$, v_p, $\perp$, …, $\perp$)
3: {phase 1} {asynchronous rounds r_p, $1 \le r_p \le n-1$}
4: **for** r_p := 1 **to** n-1
5: **send** (r_p, Δ_p ,p) **to** all
6: **wait until** [$\forall q$: received (r_p, Δ_q ,q) **or** $q \in \mathcal{D}_p$]
7: O_p := V_p
8: V_p := $V_p \oplus (\oplus_q$ received $\Delta_q)$
9: Δ_p := $V_p \star O_p$ {value is only echoed first time it is seen}
10: {phase 2}
11: **send** (r_p, V_p ,p) **to** all
12: **wait until** [$\forall q$: received (r_p, V_q ,q) **or** $q \in \mathcal{D}_p$]
13: V_p := $\otimes_q$ received V_q {computes the "intersection", including V_p}
14: {phase 3}
15: decide on leftmost non- $\perp$ coordinate of V_p

Lemma 1 After phase 1 is complete, $V_c \le V_p$ for all processes p that complete phase 1

A useful Lemma

1: V_p := ($\perp$, ..., $\perp$, v_p, $\perp$, ..., $\perp$) {p's estimate of the proposed values}
2: Δ_p := ($\perp$, ..., $\perp$, v_p, $\perp$, ..., $\perp$)
3: {phase 1} {asynchronous rounds r_p, $1 \le r_p \le n-1$}
4: **for** r_p := 1 **to** n-1
5: **send** (r_p, Δ_p ,p) **to** all
6: **wait until** [$\forall$q: received (r_p, Δ_q ,q) **or** q $\in \mathcal{D}_p$]
7: O_p := V_p
8: V_p := $V_p \oplus (\oplus_q$ received $\Delta_q)$
9: Δ_p := $V_p \star O_p$ {value is only echoed first time it is seen}
10: {phase 2}
11: **send** (r_p, V_p ,p) **to** all
12: **wait until** [$\forall$q: received (r_p, V_q ,q) **or** q $\in \mathcal{D}_p$]
13: V_p := $\otimes_q$ received V_q {computes the "intersection", including V_p}
14: {phase 3}
15: decide on leftmost non- $\perp$ coordinate of V_p

Lemma 1 After phase 1 is complete, $V_c \le V_p$ for all processes p that complete phase 1

Proof We show that

$$V_c[i] = v_i \wedge v_i \neq \perp \Rightarrow \forall p : V_p[i] = v_i$$

Let r be the first round when c sees v_i

- $r \le n-2$
  - c will send to all Δ_c with v_i in round r
  - By weak accuracy, all correct processes receive v_i by next round
- $r = n-1$
  - v_i has been forwarded $n-1$ times: every other process has seen v_i

Two additional cool lemmas

1: V_p := ($\perp$, …, $\perp$, v_p, $\perp$, …, $\perp$) {p's estimate of the proposed values}
2: Δ_p := ($\perp$, …, $\perp$, v_p, $\perp$, …, $\perp$)
3: {Phase 1} {asynchronous rounds r_p, $1 \le r_p \le n-1$}
4: **for** r_p := 1 **to** n-1
5: **send** (r_p, Δ_p ,p) **to** all
6: **wait until** [$\forall q$: received (r_p, Δ_q ,q) **or** $q \in \mathcal{D}_p$]
7: O_p := V_p
8: V_p := $V_p \oplus (\oplus_q$ received $\Delta_q)$
9: Δ_p := $V_p \star O_p$ {value is only echoed first time it is seen}
10: {Phase 2}
11: **send** (r_p, V_p ,p) **to** all
12: **wait until** [$\forall q$: received (r_p, V_q ,q) **or** $q \in \mathcal{D}_p$]
13: V_p := $\otimes_q$ received V_q {computes the "intersection", including V_p}
14: {Phase 3}
15: decide on leftmost non-$\perp$ coordinate of V_p

Lemma 2 After Phase 2 is complete, $V_c = V_p$ for each p that completes phase 2

Proof

- All processes that completed phase 2 have received V_c. Since V_c is the smallest V vector,
$$V_c[i] \neq \perp \Rightarrow V_p[i] \neq \perp \quad \forall p$$
- By the definition of $\otimes$
$$V_c[i] = \perp \Rightarrow V_p[i] = \perp \quad \forall p$$

after phase 2

Lemma 3 $V_c \neq (\perp, \perp, \perp, \ldots, \perp)$

Solving consensus

1: $V_p := (\perp, \ldots, \perp, v_p, \perp, \ldots, \perp)$ {p's estimate of the proposed values}
2: $\Delta_p := (\perp, \ldots, \perp, v_p, \perp, \ldots, \perp)$
3: {phase 1} {asynchronous rounds r_p, $1 \leq r_p \leq n - 1$}
4: **for** $r_p := 1$ **to** n-1
5: **send** (r_p, Δ_p, p) **to** all
6: **wait until** [$\forall q$: received (r_p, Δ_q, q) **or** $q \in \mathcal{D}_p$]
7: $O_p := V_p$
8: $V_p := V_p \oplus (\oplus_q \text{ received } \Delta_q)$
9: $\Delta_p := V_p \star O_p$ {value is only echoed first time it is seen}
10: {phase 2}
11: **send** (r_p, V_p, p) **to** all
12: **wait until** [$\forall q$: received (r_p, V_q, q) **or** $q \in \mathcal{D}_p$]
13: $V_p := \otimes_q \text{ received } V_q$ {computes the "intersection", including v_p}
14: {phase 3}
15: decide on leftmost non- $\perp$ coordinate of V_p

Theorem The protocol to the left satisfies Validity, Agreement, and Termination

Proof

Left as an exercise

A lower bound - I

Theorem Consensus with $\Diamond P$ requires $f < n/2$

A lower bound - I

Theorem Consensus with $\Diamond P$ requires $f < n/2$

Proof

- Suppose n is even, and a protocol exists that solves consensus when $f = n/2$
- Divide the set of processes in two sets of size $n/2$, P_1 and P_2

A lower bound - II

Consider three executions:

$P_1 \leftarrow 0; P_2 \leftarrow 0$

All processes in P_2 crash before they can propose

Detectors work perfectly

A lower bound - II

Consider three executions:

$P_1 \leftarrow 0; P_2 \leftarrow 0$

All processes in P_2 crash before they can propose

Detectors work perfectly

P_1 decides 0

after t_1

A lower bound - II

Consider three executions:

$P_1 \leftarrow 0; P_2 \leftarrow 0$

All processes in P_2 crash before they can propose

Detectors work perfectly

$P_1 \leftarrow 1; P_2 \leftarrow 1$

All processes in P_1 crash before they can propose

Detectors work perfectly

P_1 decides 0

after t_1

A lower bound - II

Consider three executions:

$P_1 \leftarrow 0; P_2 \leftarrow 0$

All processes in P_2 crash before they can propose

Detectors work perfectly

P_1 decides 0

after t_1

$P_1 \leftarrow 1; P_2 \leftarrow 1$

All processes in P_1 crash before they can propose

Detectors work perfectly

P_2 decides 1

after t_2

A lower bound - II

Consider three executions:

$P_1 \leftarrow 0; P_2 \leftarrow 0$

All processes in P_2 crash before they can propose

Detectors work perfectly

P_1 decides 0

after t_1

$P_1 \leftarrow 0; P_2 \leftarrow 1$

No process crashes

Detectors make mistakes: until $max(t_1, t_2)$, P_1 believes P_2 crashed, and vice versa

$P_1 \leftarrow 1; P_2 \leftarrow 1$

All processes in P_1 crash before they can propose

Detectors work perfectly

P_2 decides 1

after t_2

A lower bound - II

Consider three executions:

$P_1 \leftarrow 0; P_2 \leftarrow 0$

All processes in P_2 crash before they can propose

Detectors work perfectly

P_1 decides 0

after t_1

$P_1 \leftarrow 0; P_2 \leftarrow 1$

No process crashes

Detectors make mistakes: until $max(t_1, t_2)$, P_1 believes P_2 crashed, and vice versa

P_1 decides 0
P_2 decides 1

$P_1 \leftarrow 1; P_2 \leftarrow 1$

All processes in P_1 crash before they can propose

Detectors work perfectly

P_2 decides 1

after t_2

The case of the Rotating Coordinator

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

- Asynchronous rounds
- Each round has a coordinator c
- $c_{id} = (r \bmod n) + 1$
- 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

$r := r+1$

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

{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...

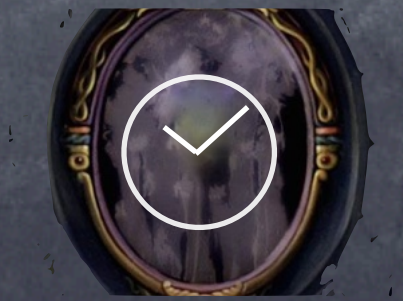
- Great to know that $\Diamond W$ can solve consensus...
- ...but that means I can't build $\Diamond W$!

Is there something interesting
I can do with FDs?

Failure detectors in practice

Does A live?

End-to-end timeouts



Failure Detector

Choosing your poison

Delayed detection

Loss of accuracy

Intrusive

Falcon

Leners et al.

Fast

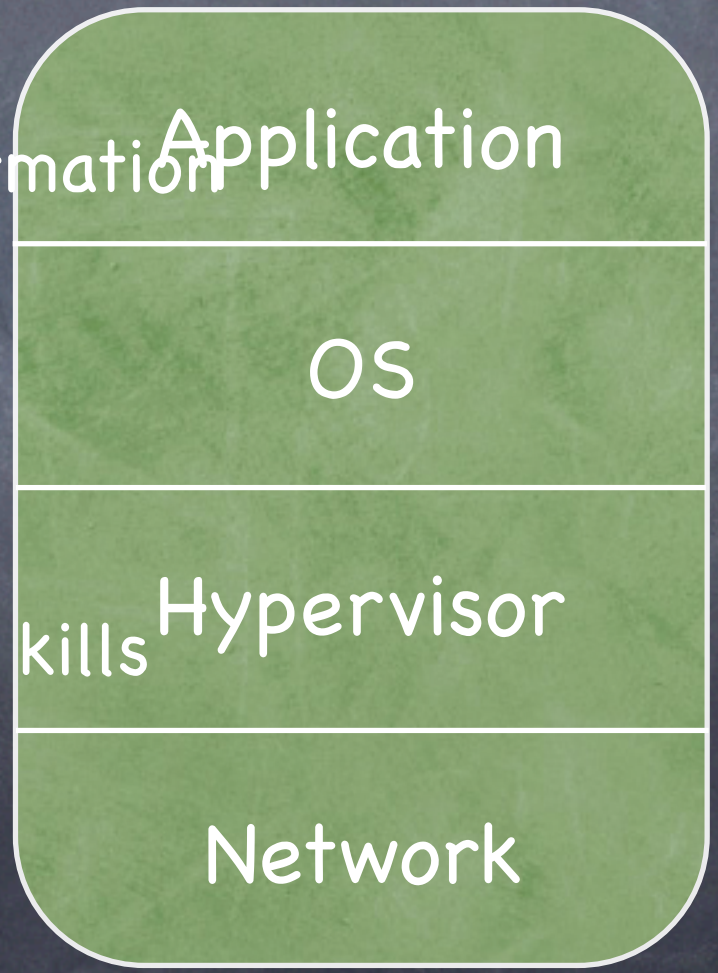
Accurate

Unobtrusive

End-to-end timeouts

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

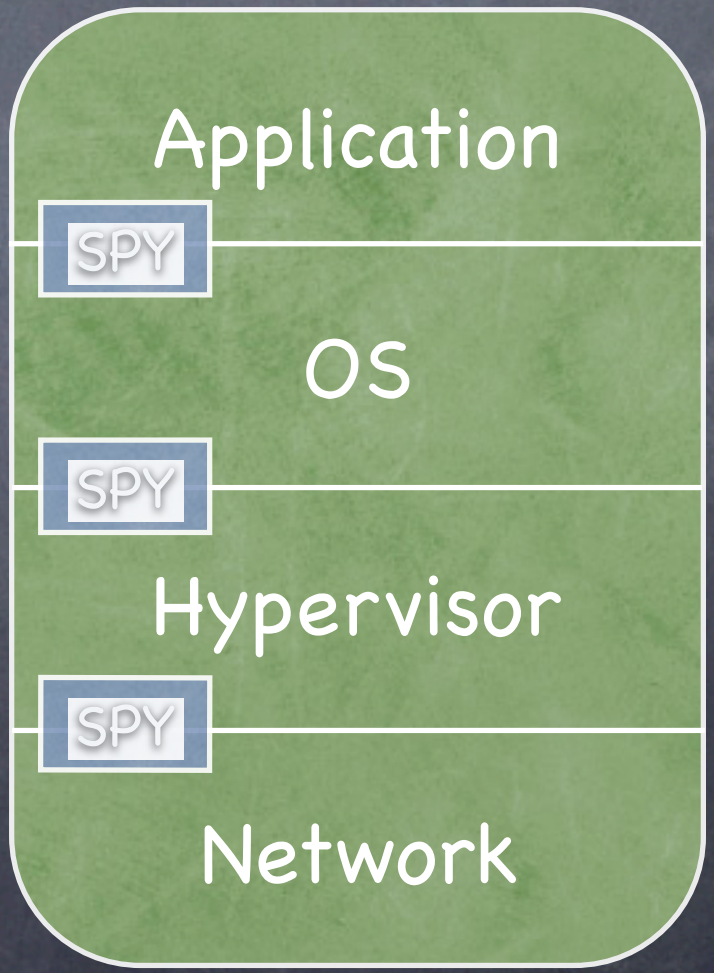
- Gather inside information
- Use local timeouts
- Use lethal force
- Avoid unnecessary kills



Design

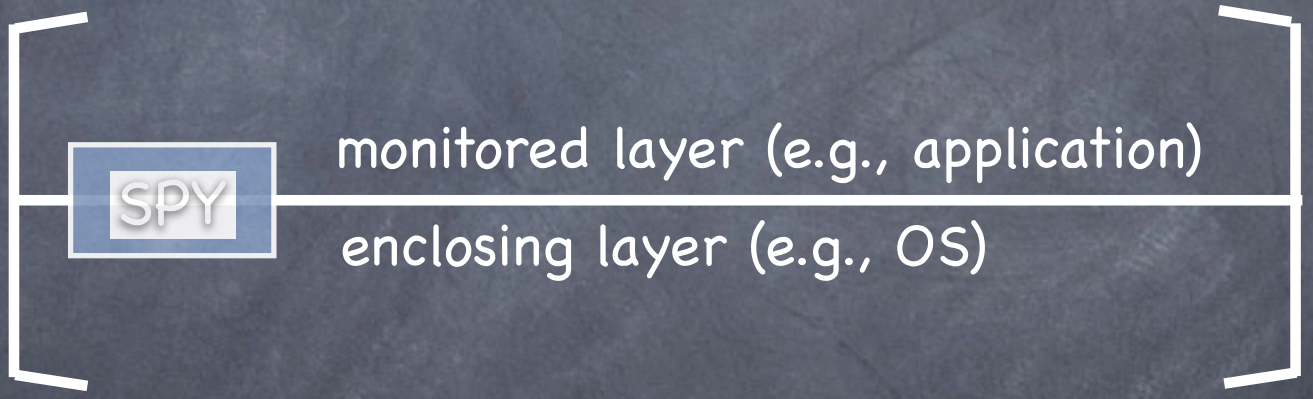
Use spies to get inside the layers

- 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

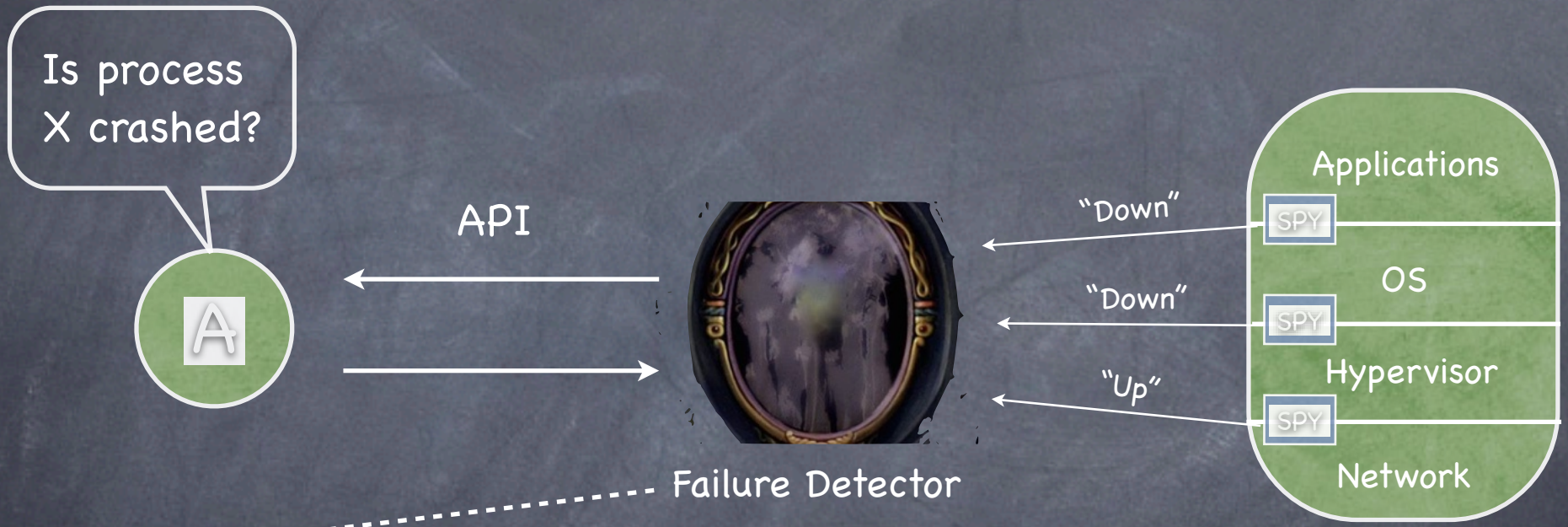
Weapon of choice: OS signals

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

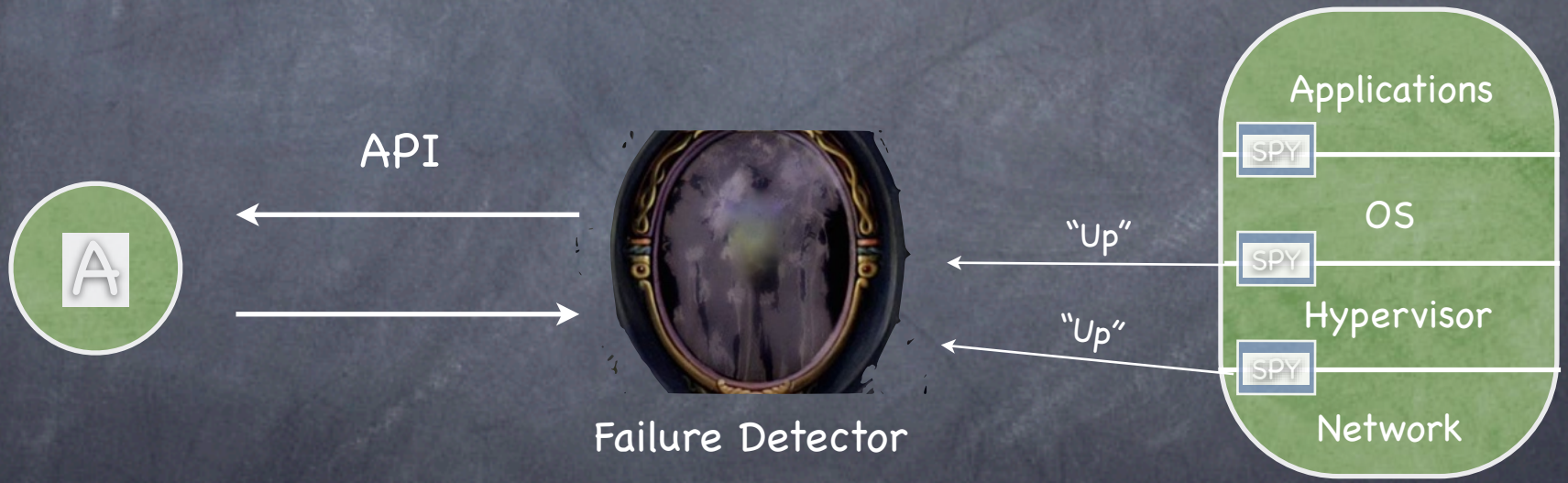
inside information

\+ callbacks

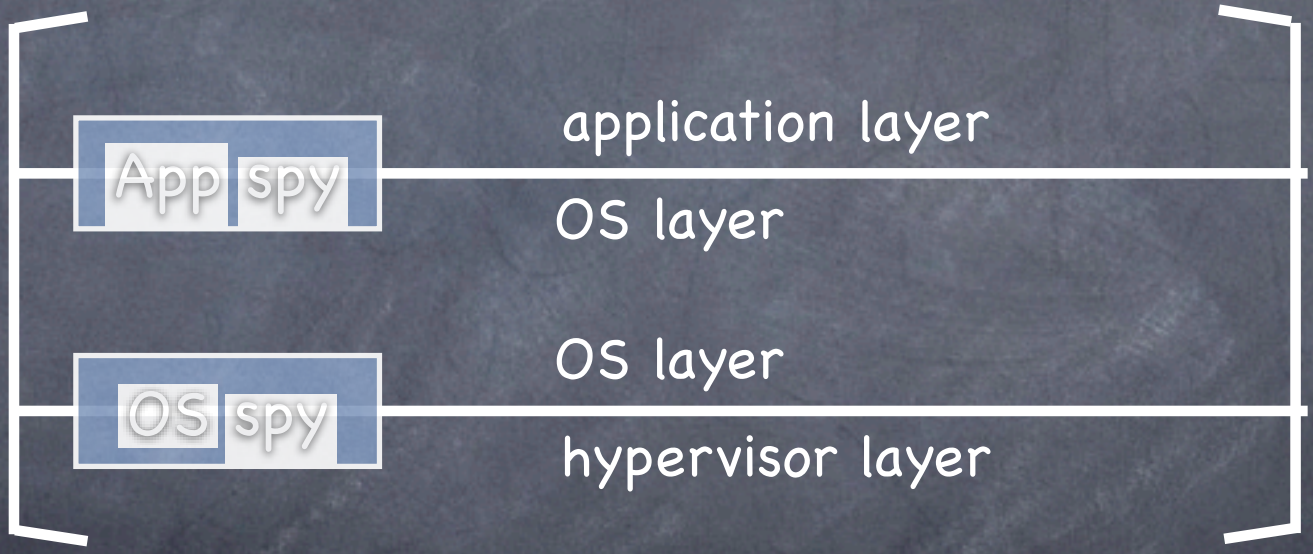
Fast detection

What are the corner cases?

Corner case 1: spy crashes



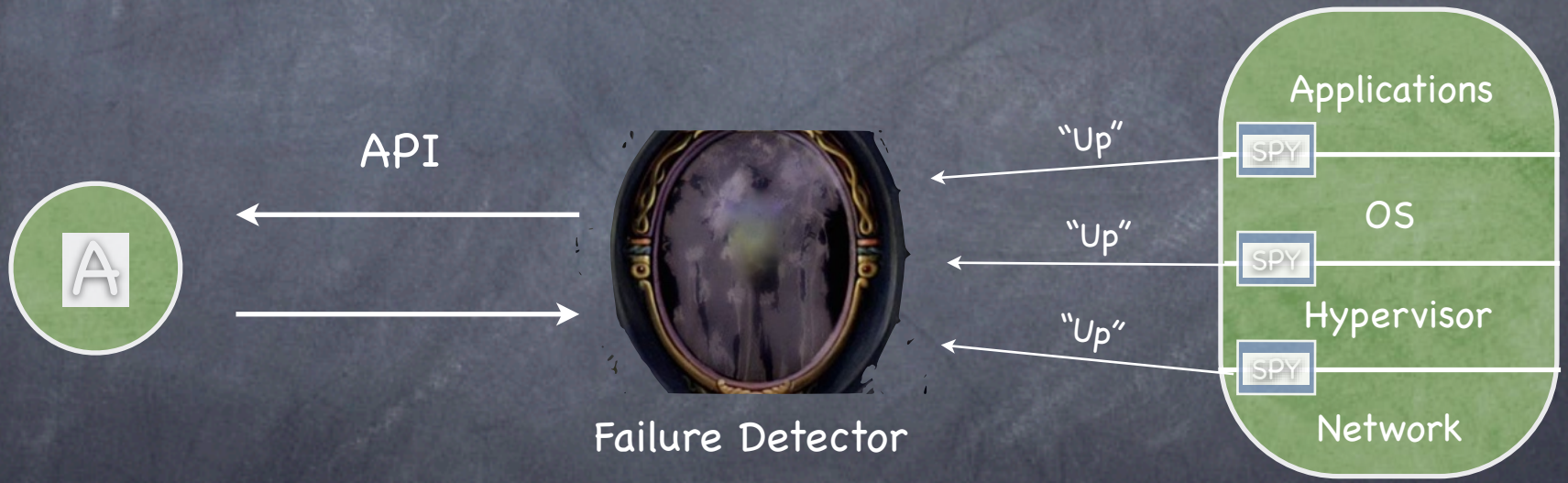
Lower spies monitor higher spies



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

What are the corner cases?

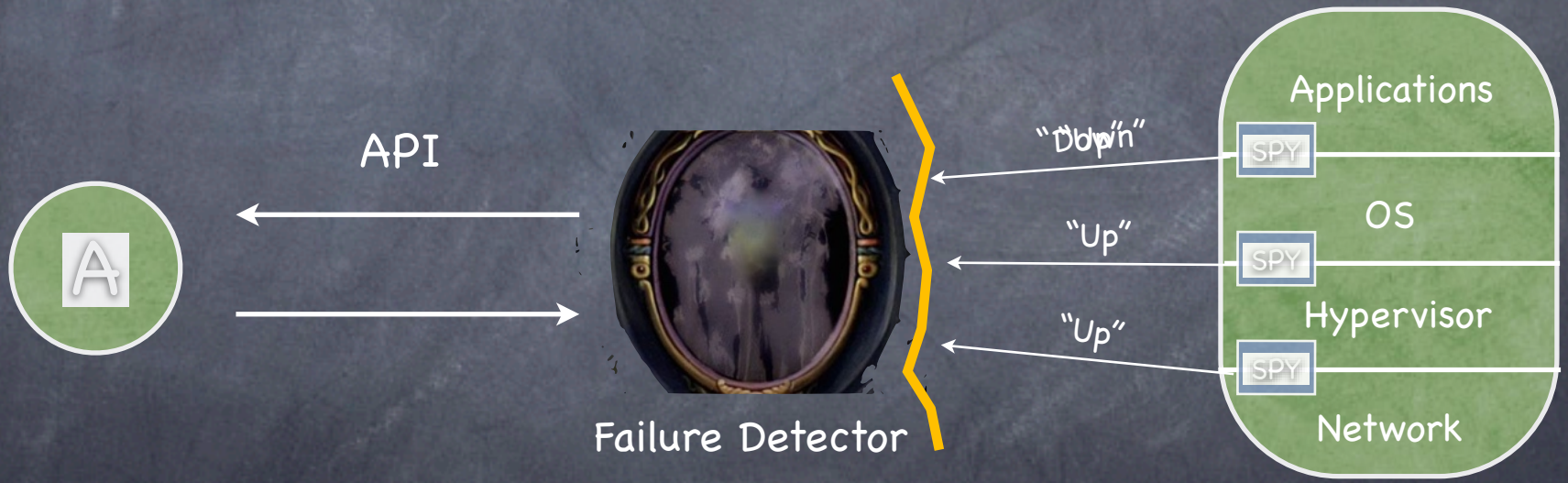
Corner case 2: spy does not detect failures



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

What are the corner cases?

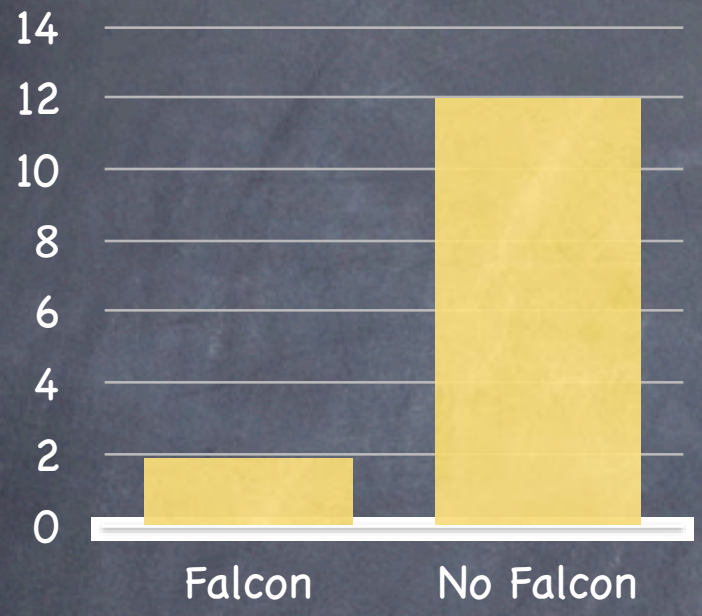
Corner case 3: network partitions



Communication necessary for accuracy

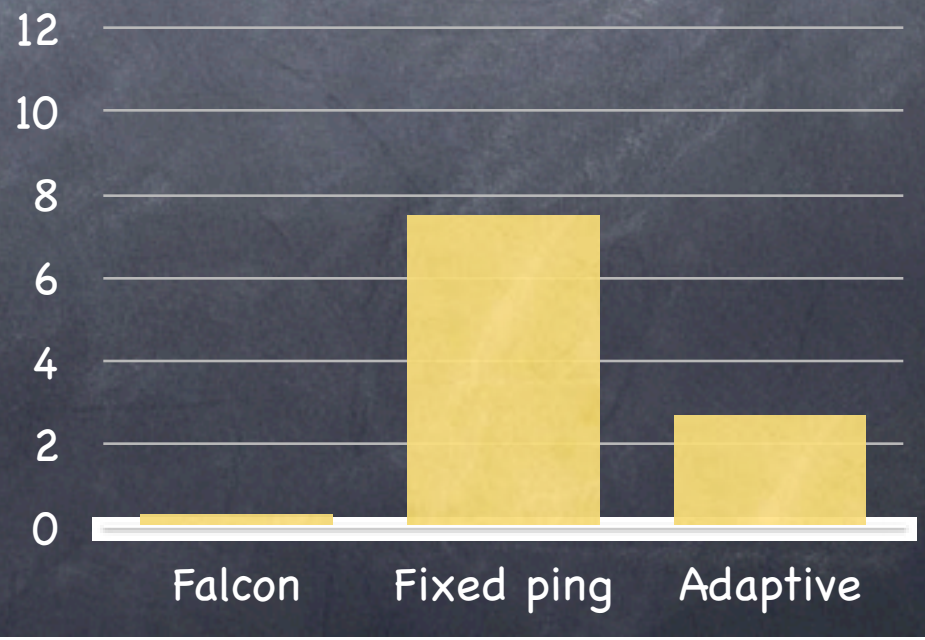
Falcon waits

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