Detecting failures in distributed systems with the FALCON spy network

Joshua B. Leners, Hao Wu, Wei-Lun Hung, Marcos K. Aguilera, Michael Walfish

Presenter: Neal Mangaokar

Failure Detectors

Failure Detectors

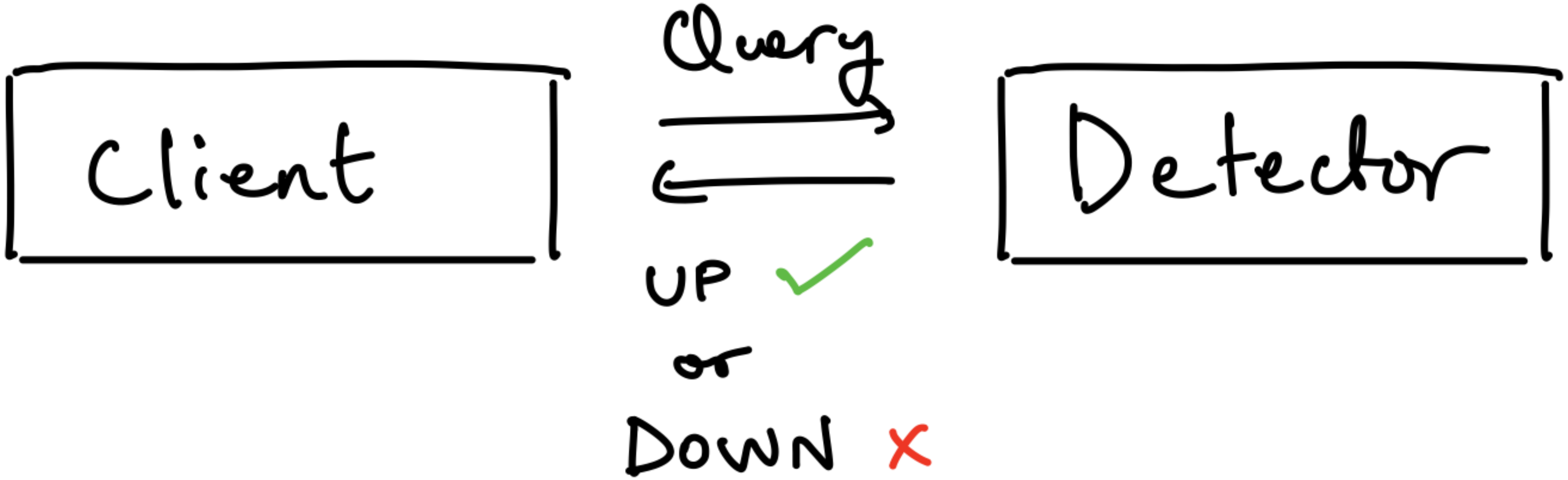
- A service that reports status of a remote process as **UP** or **DOWN:**

Failure Detectors

- A service that reports status of a remote process as **UP** or **DOWN:**

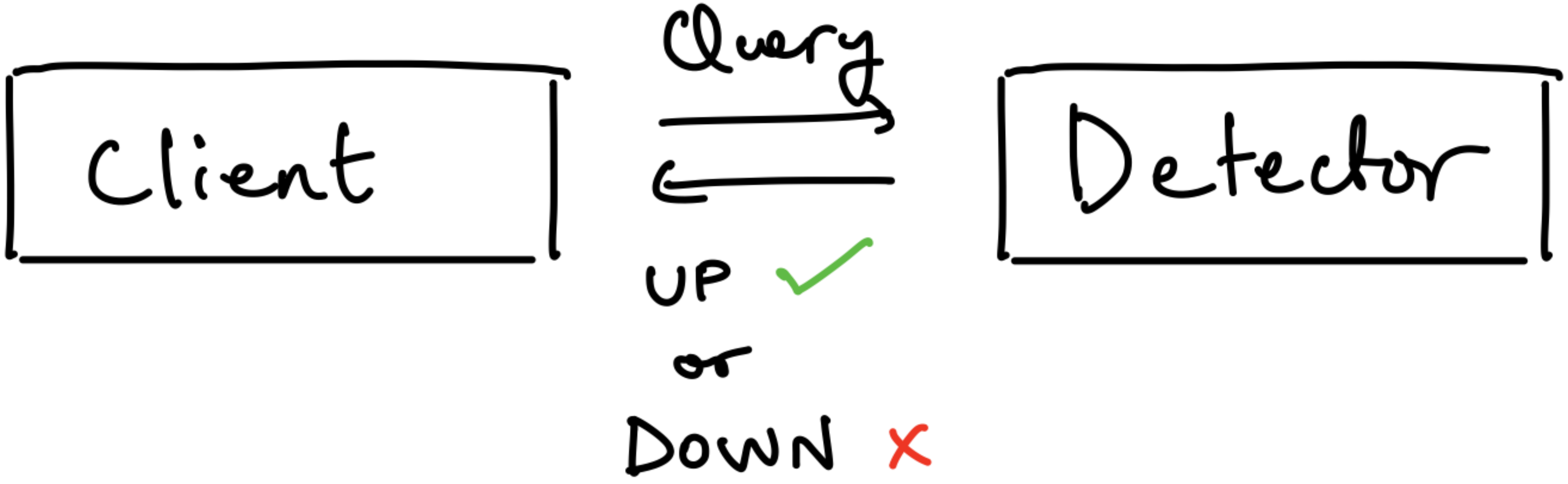
Failure Detectors

- A service that reports status of a remote process as **UP** or **DOWN:**



Failure Detectors

- A service that reports status of a remote process as **UP** or **DOWN:**



- Fundamental primitive in distributed applications

Reliable Failure Detectors (RFDs)

Reliable Failure Detectors (RFDs)

- **SAFETY (STRONG ACCURACY):**

RFD reports process is **DOWN** $\implies$ process crashed

Reliable Failure Detectors (RFDs)

- **SAFETY (STRONG ACCURACY):**

RFD reports process is **DOWN** $\Longrightarrow$ process crashed

- **LIVENESS (STRONG COMPLETENESS):**

Process crashes $\Longrightarrow$ RFD eventually reports process is **DOWN**

Bad News

Bad News

- The FLP result: consensus is impossible in asynchronous systems where one process may crash:
    - Can't differentiate between crashed process vs. slow network

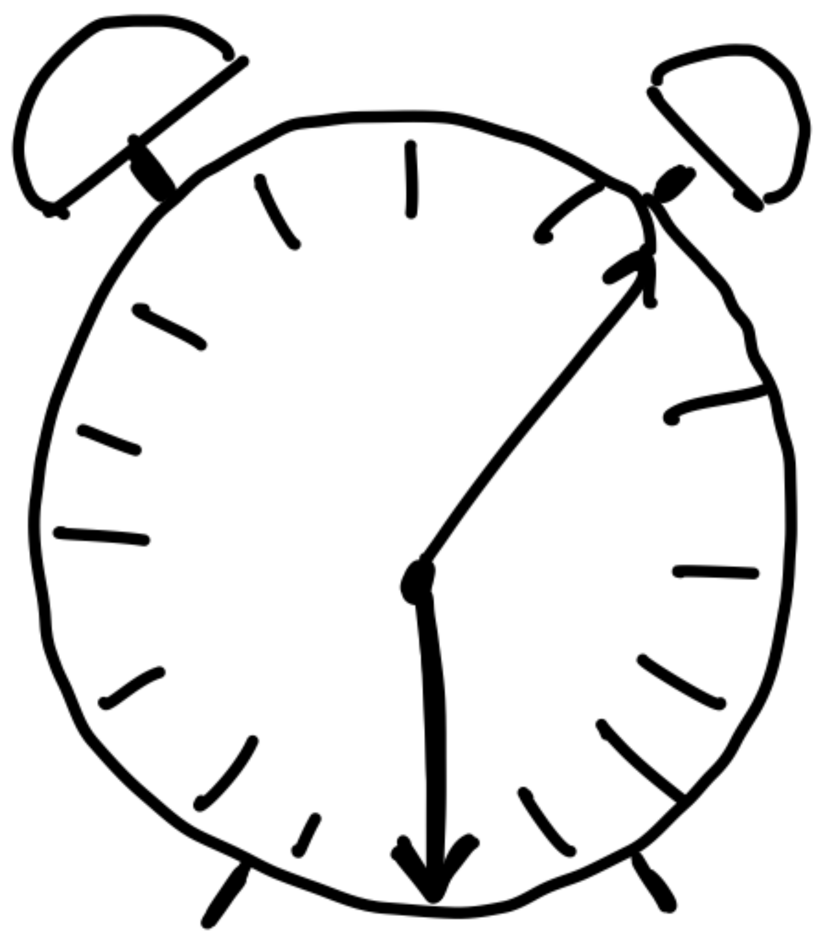
Bad News

- The FLP result: consensus is impossible in asynchronous systems where one process may crash:
    - Can't differentiate between crashed process vs. slow network
- Consensus is reducible to RFD:
    - Consensus is impossible $\implies$ **RFD is impossible**

Solutions

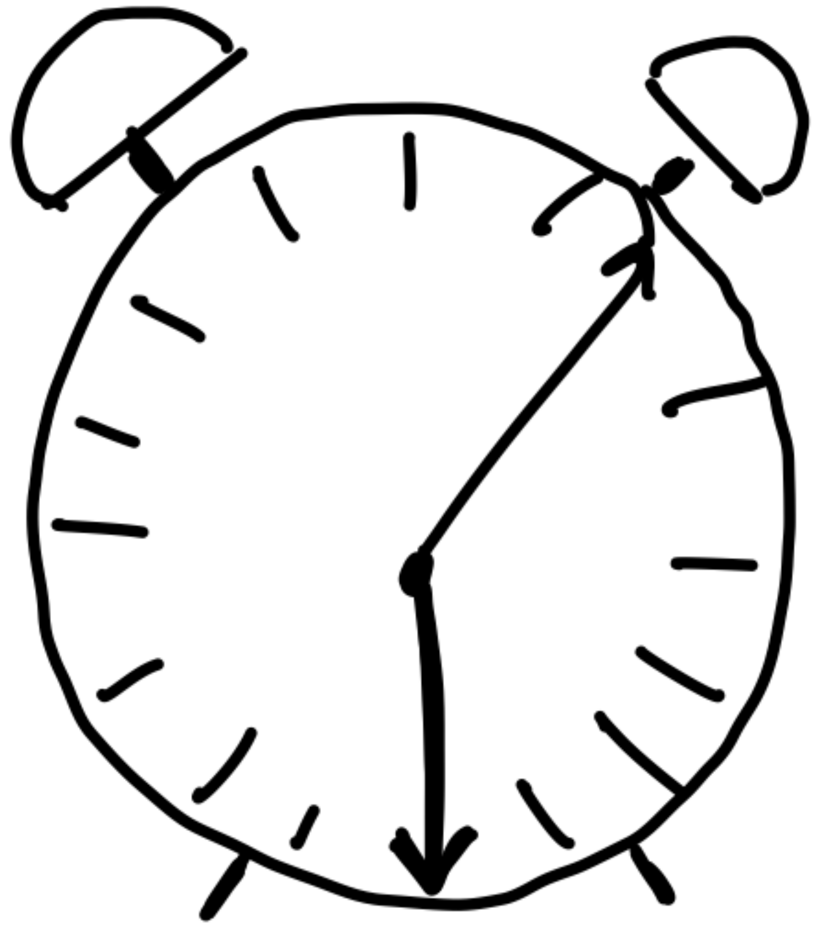
Solutions

- Assume synchrony and use **end-to-end timeouts:**



Solutions

- Assume synchrony and use **end-to-end timeouts:**



- **Problem:** doesn't extend very well to asynchrony:
    - Too short: might violate safety
    - Too long: might be overly slow zzz...

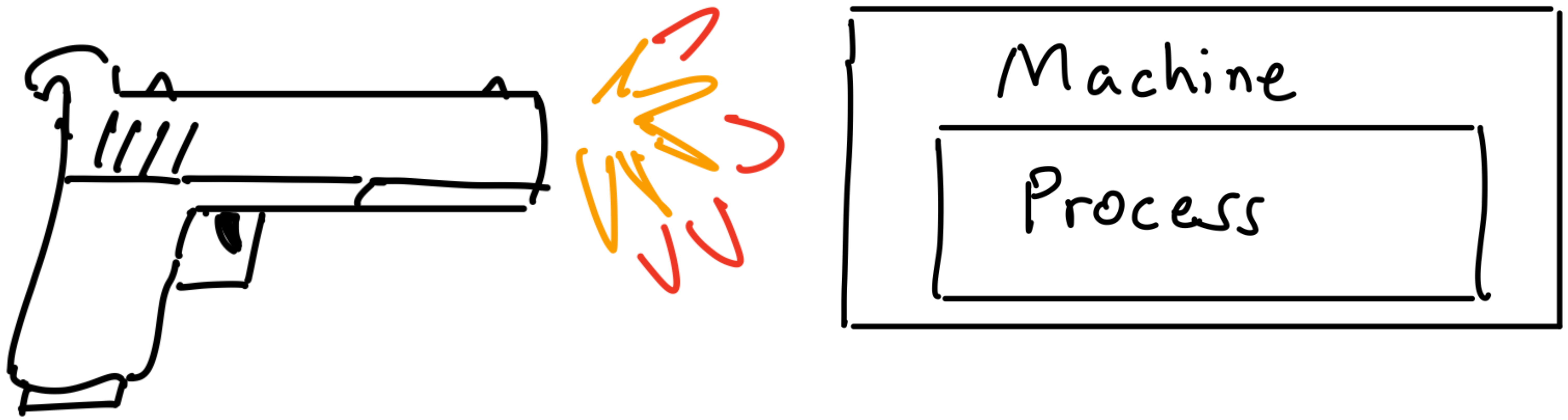
Solutions (cont.)

Solutions (cont.)

- Can I use short timeouts and still be safe?

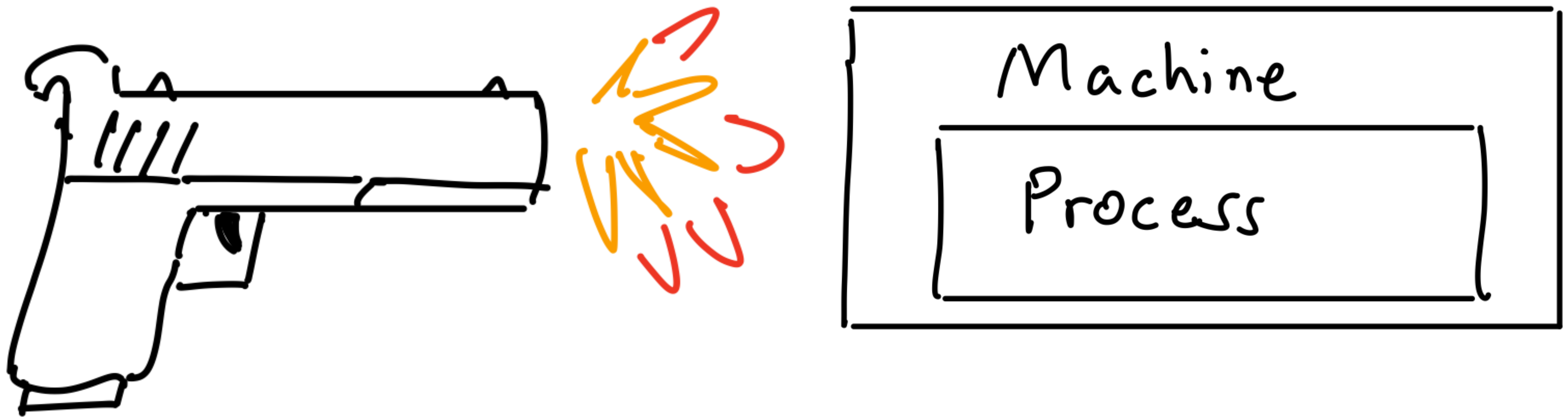
Solutions (cont.)

- Can I use short timeouts and still be safe?
    - **Murder**: after timeout, kill the machine and then report DOWN:



Solutions (cont.)

- Can I use short timeouts and still be safe?
    - **Murder**: after timeout, kill the machine and then report DOWN:

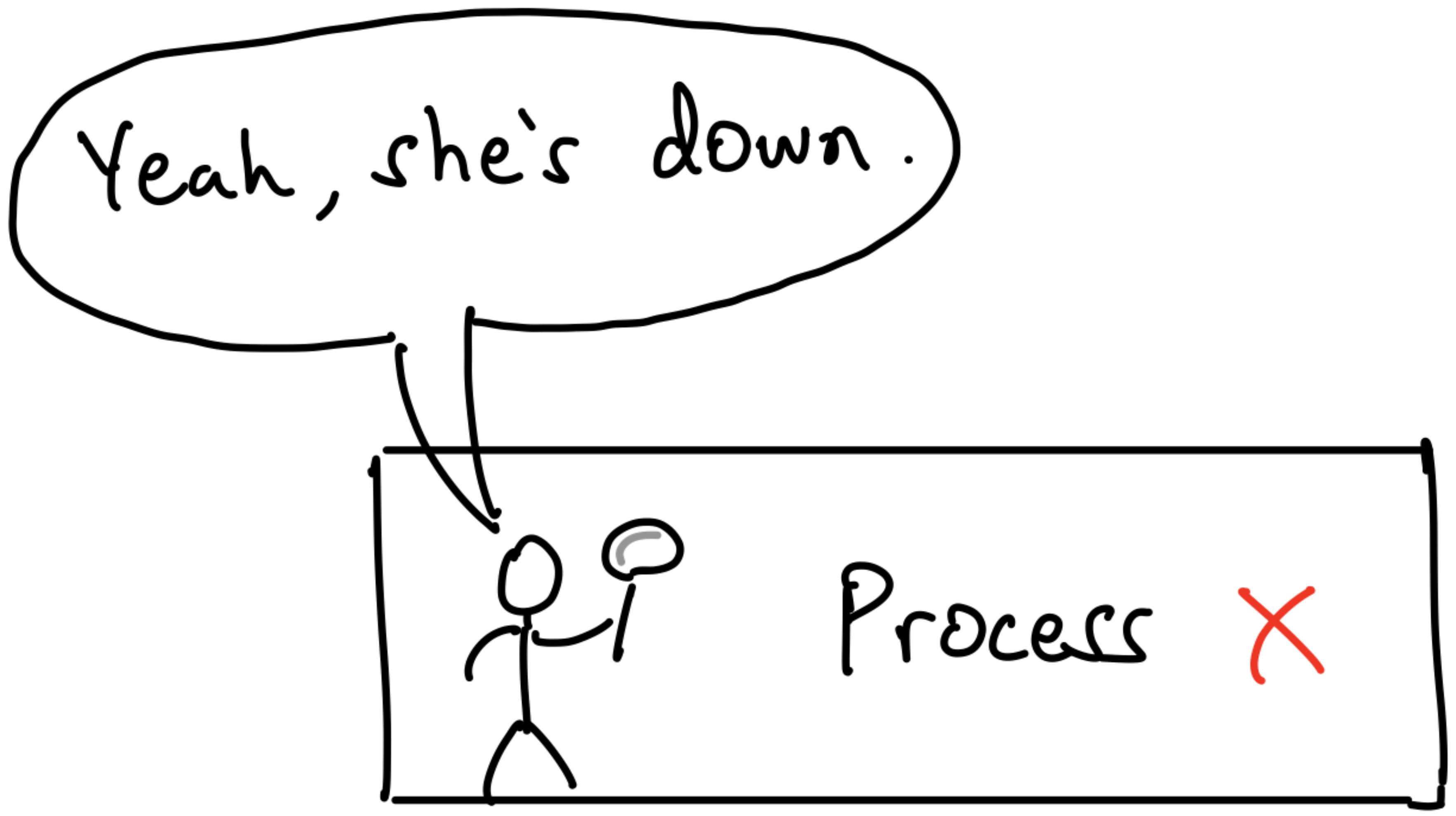


- **Problem:** disruptive (and sociopathic)

A Better Solution: FALCON

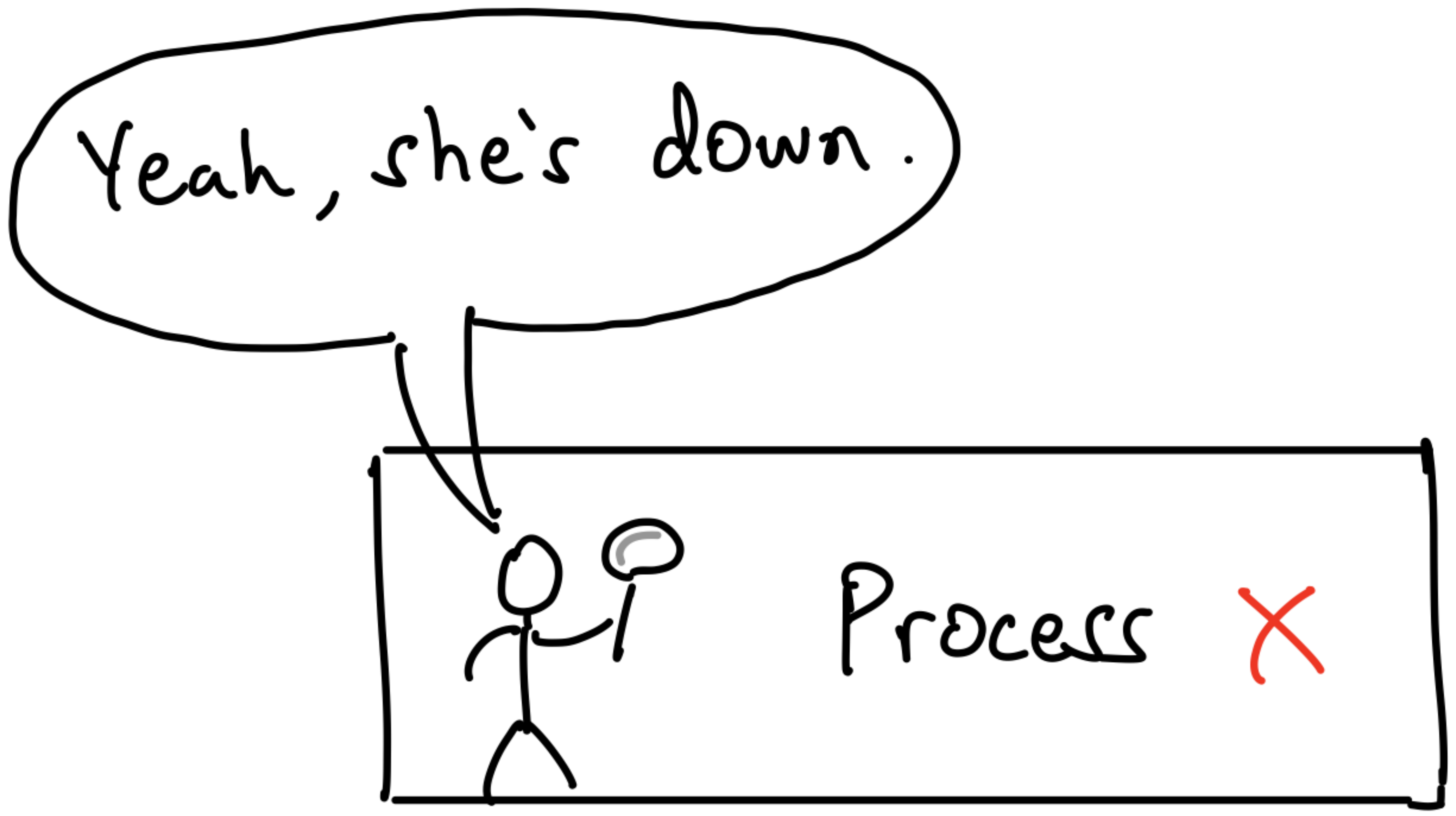
A Better Solution: FALCON

- **Detect** failures with insider information from process:
    - can make us almost always safe and fast



A Better Solution: FALCON

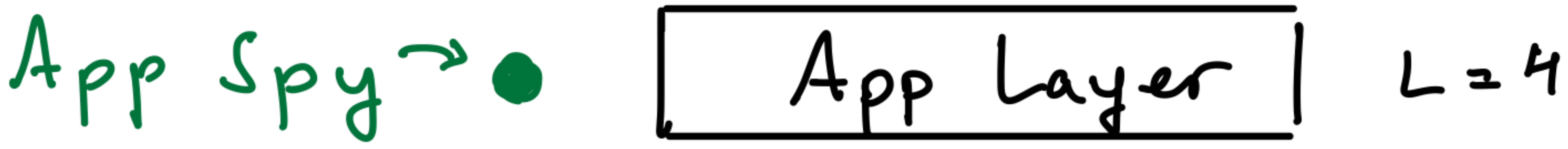
- **Detect** failures with insider information from process:
  - can make us almost always safe and fast
- **Fallback** onto suboptimal solutions as a failsafe:
  - Timeouts + killing ensures liveness and safety when “almost always” fails
- **Don’t worry** about extreme asynchrony:
  - Block!



Collecting Insider Information with Spies

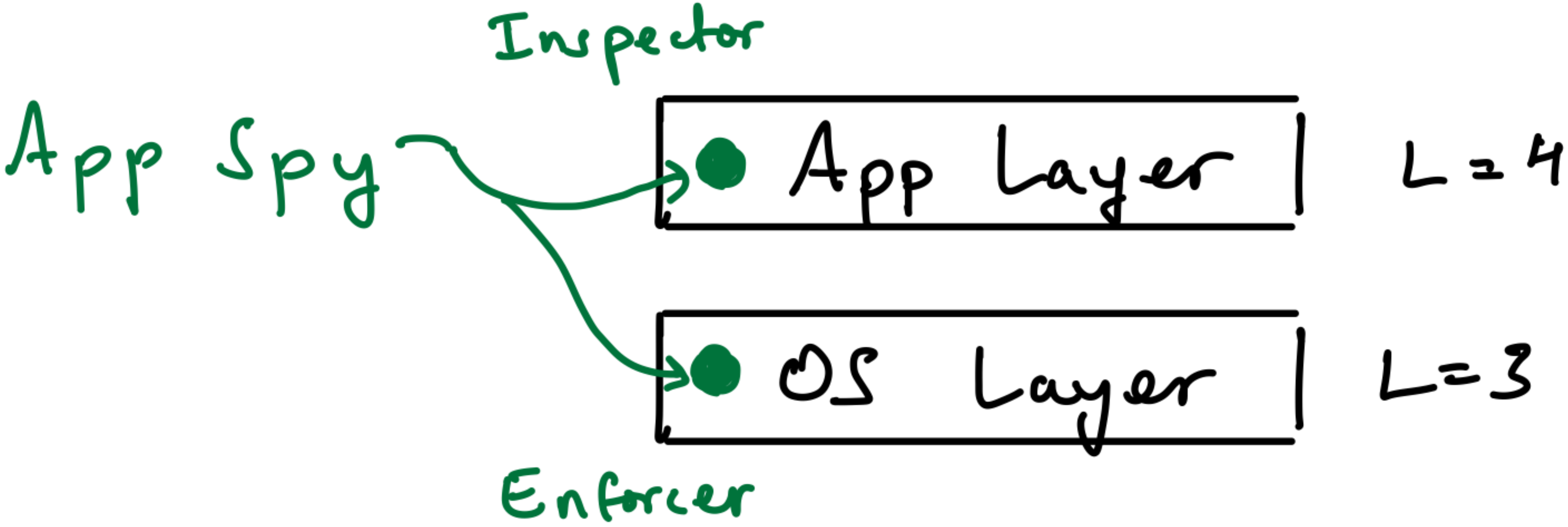
Collecting Insider Information with Spies

- Client can gather information using a spy:
    - Spy sits on remote machine and monitors "layer" L



Collecting Insider Information with Spies

- Client can gather information using a spy:
  - Spy sits on remote machine and monitors “layer” L

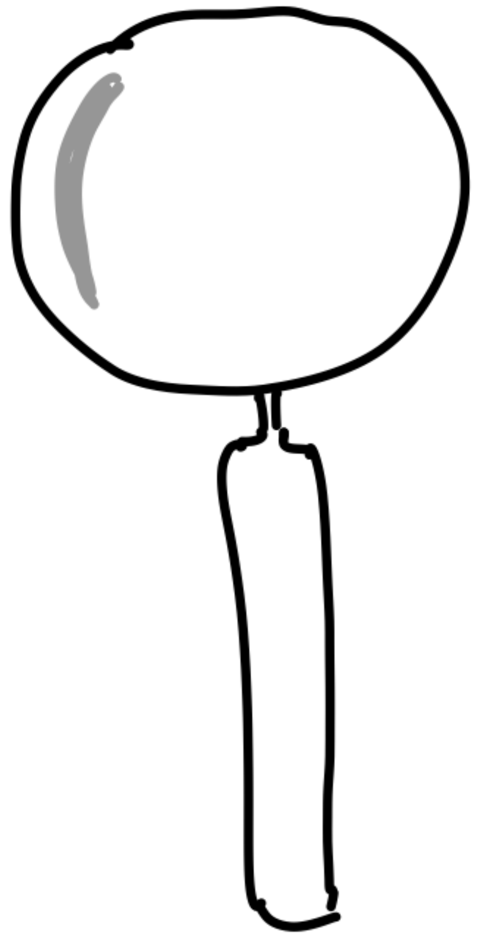


  - Spy monitors L, but occupies L and $L-1$ (inspector and enforcer)

What Information Does a Spy Collect?

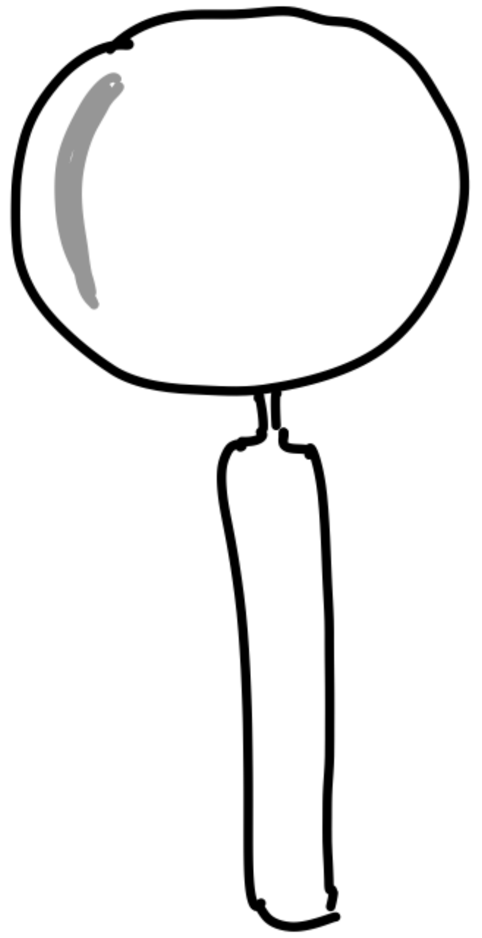
What Information Does a Spy Collect?

- Inspector:
    - Tries to infer operational status (alive, or not)



What Information Does a Spy Collect?

- Inspector:
  - Tries to infer operational status (alive, or not)



- Enforcer:
  - Tells client operational status
  - **LAYER_DOWN** or **LAYER_UP**

Example: Application Spy

Example: Application Spy

- Inspector:
  - Is a thread inside the process
  - Looks for signs of life

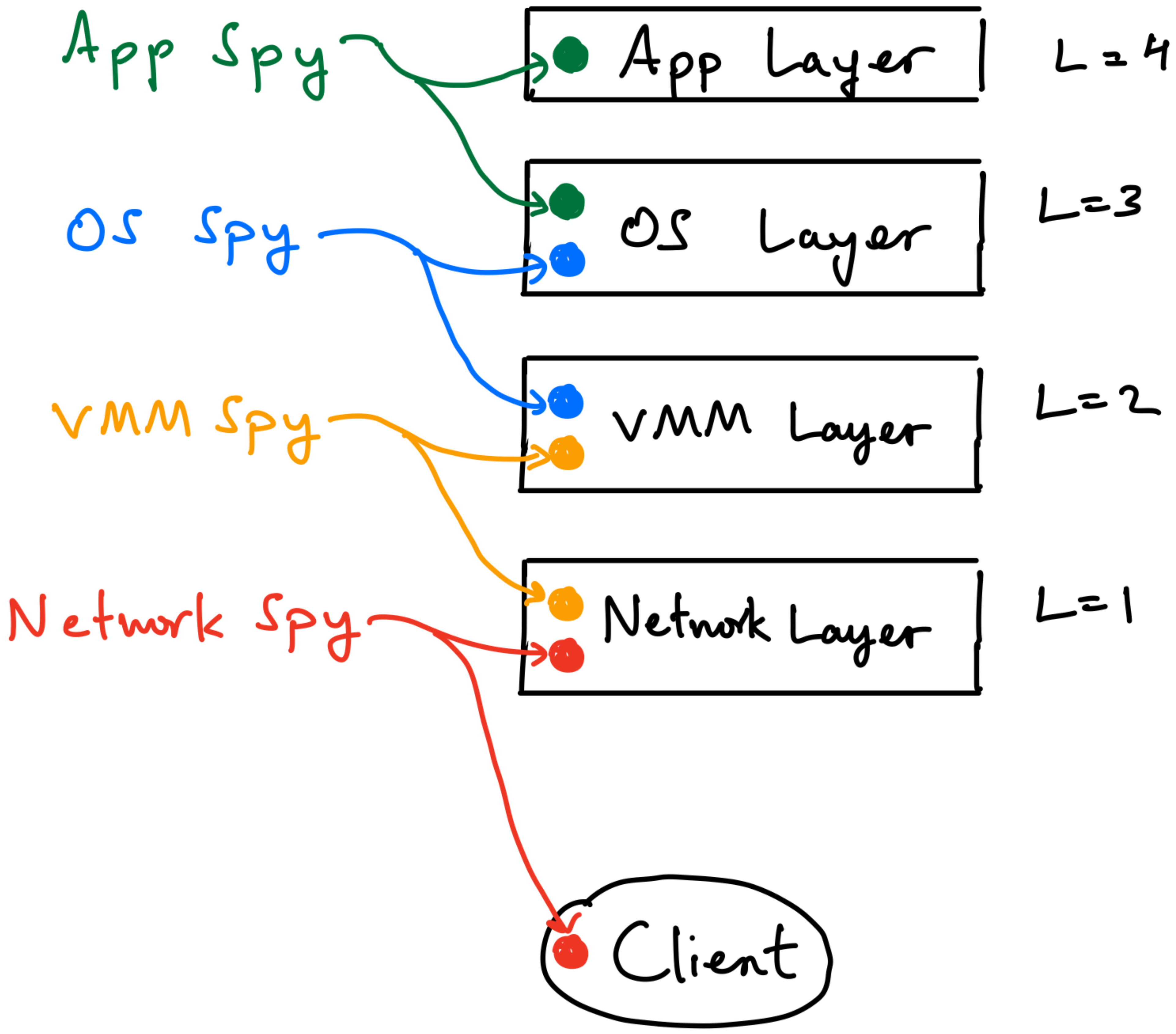
Example: Application Spy

- Inspector:
  - Is a thread inside the process
  - Looks for signs of life
- Enforcer:
  - Is its own process
  - Communicates with inspector via IPC to make quick decisions
  - Before reporting **LAYER_DOWN**, double-checks process table

Collecting More Information: A Spy Network

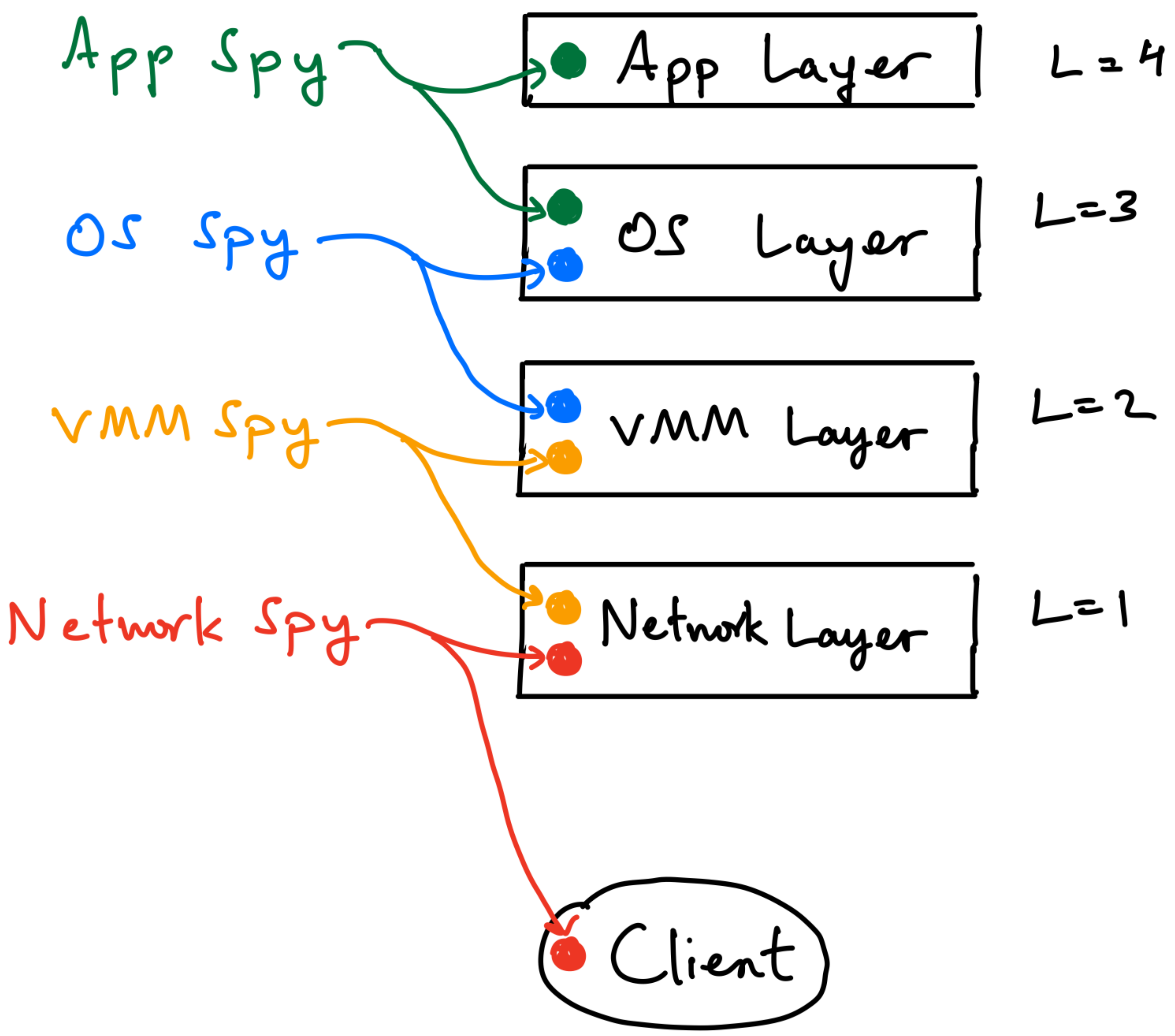
Collecting More Information: A Spy Network

- One spy per layer



Collecting More Information: A Spy Network

- One spy per layer
- Each spy implicitly monitors spy above it
  - App spy enforcer dead ⟺ OS layer dead
  - OS layer dead ⟹ OS spy says **LAYER_DOWN**



Can Spies Mess Up?

Can Spies Mess Up?

- **Problem:** app spy enforcer could die without OS layer dying
- **Solution:** long end-to-end timeout as fallback

Can Spies Mess Up?

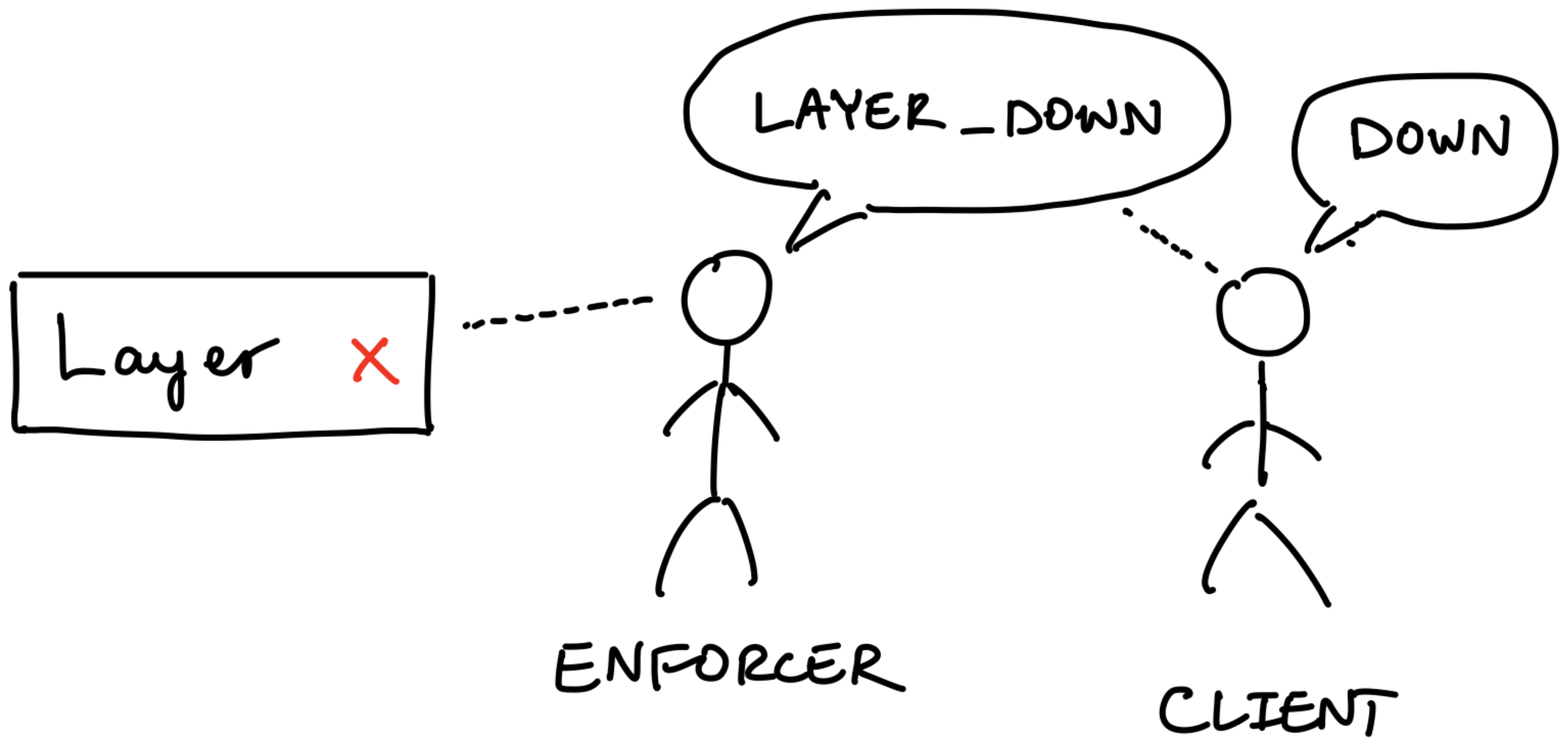
- **Problem:** app spy enforcer could die without OS layer dying
- **Solution:** long end-to-end timeout as fallback

- **Problem:** app spy could miss layer failure (bad "insider" information)
- **Solution:** again, long end-to-end timeout as fallback

The Failure Detection Algorithm

The Failure Detection Algorithm

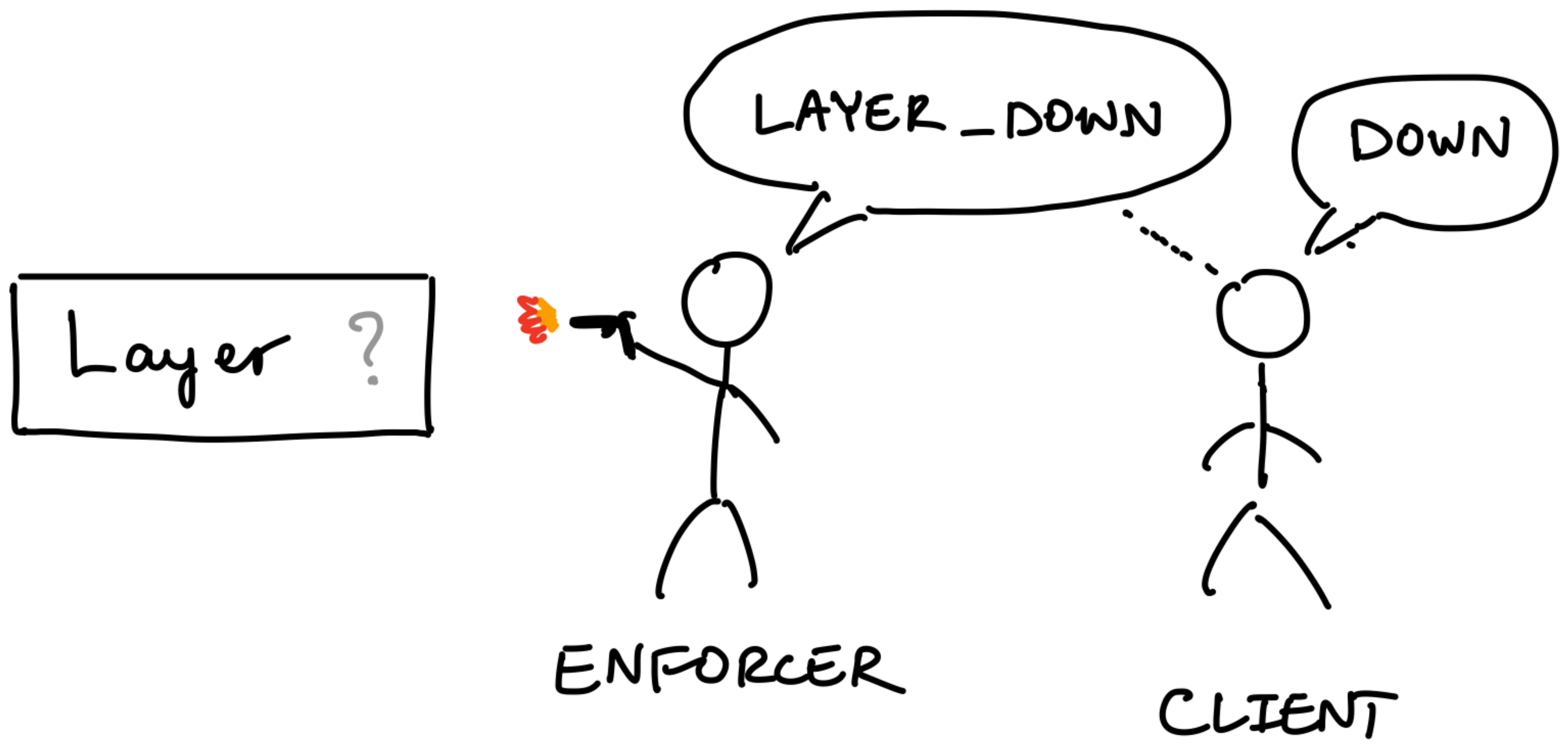
- If any enforcer confident of crash:
  - It reports **LAYER_DOWN** to FALCON
  - FALCON reports final decision **DOWN**
    (works because **any layer down** $\implies$ **process is down**)



The Failure Detection Algorithm

- If any enforcer confident of crash:
  - It reports **LAYER_DOWN** to FALCON
  - FALCON reports final decision **DOWN**
    (works because **any layer down** $\implies$ **process is down**)

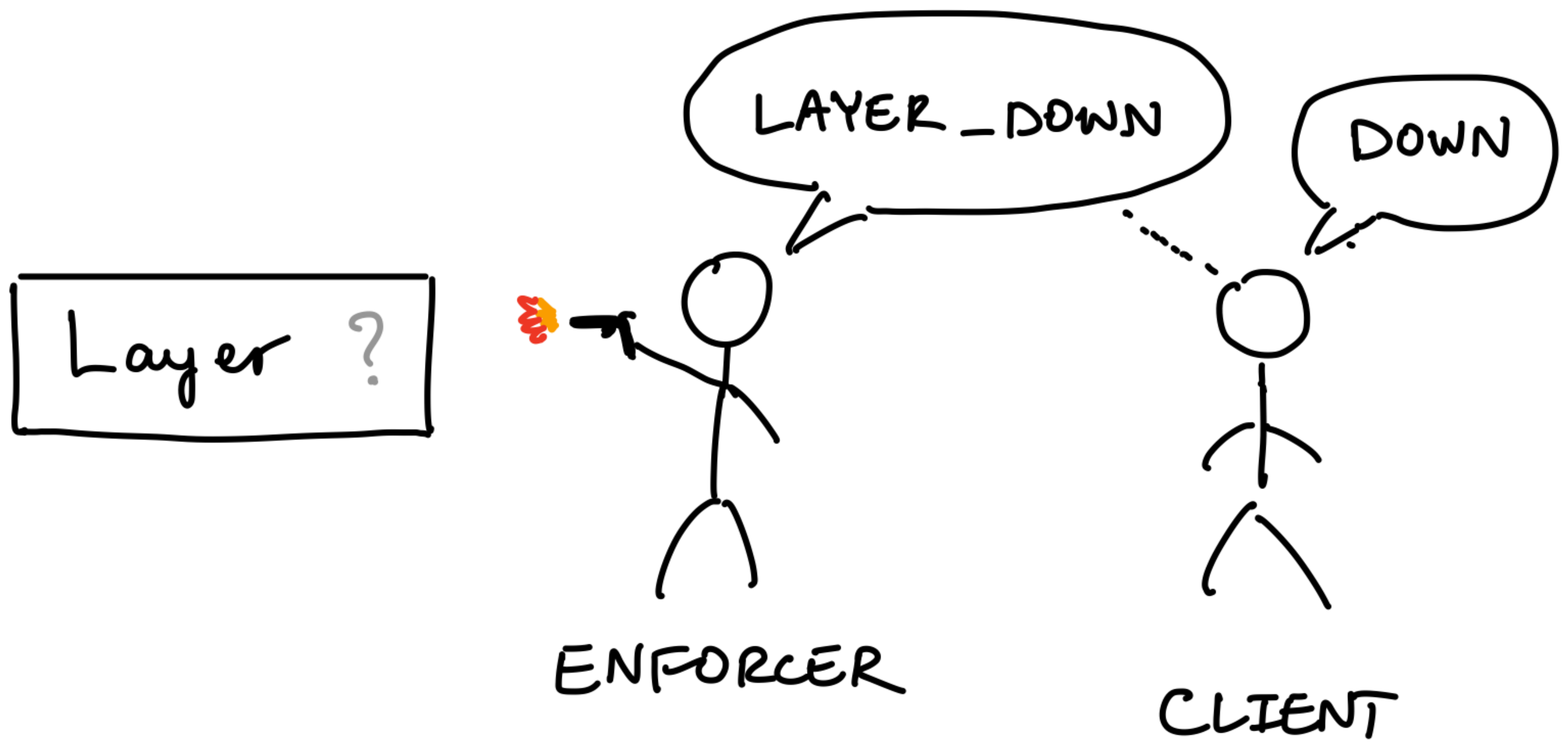
- If any enforcer suspicious:
  - Kill layer and be confident :)



The Failure Detection Algorithm

- If any enforcer confident of crash:
    - It reports **LAYER_DOWN** to FALCON
    - FALCON reports final decision **DOWN**
      (works because **any layer down** $\implies$ **process is down**)

- If any enforcer suspicious:
    - Kill layer and be confident :)



- If timeout expires:
    - FALCON issues surgical kill orders and says **DOWN**

Corner Case

Corner Case

- **Problem:** haven’t heard from network spy

Corner Case

- **Problem:** haven't heard from network spy
- **Solution:** block, because this means either:
    - extreme asynchrony
    - network layer crash (indistinguishable from asynchrony)

FLP

Evaluating FALCON

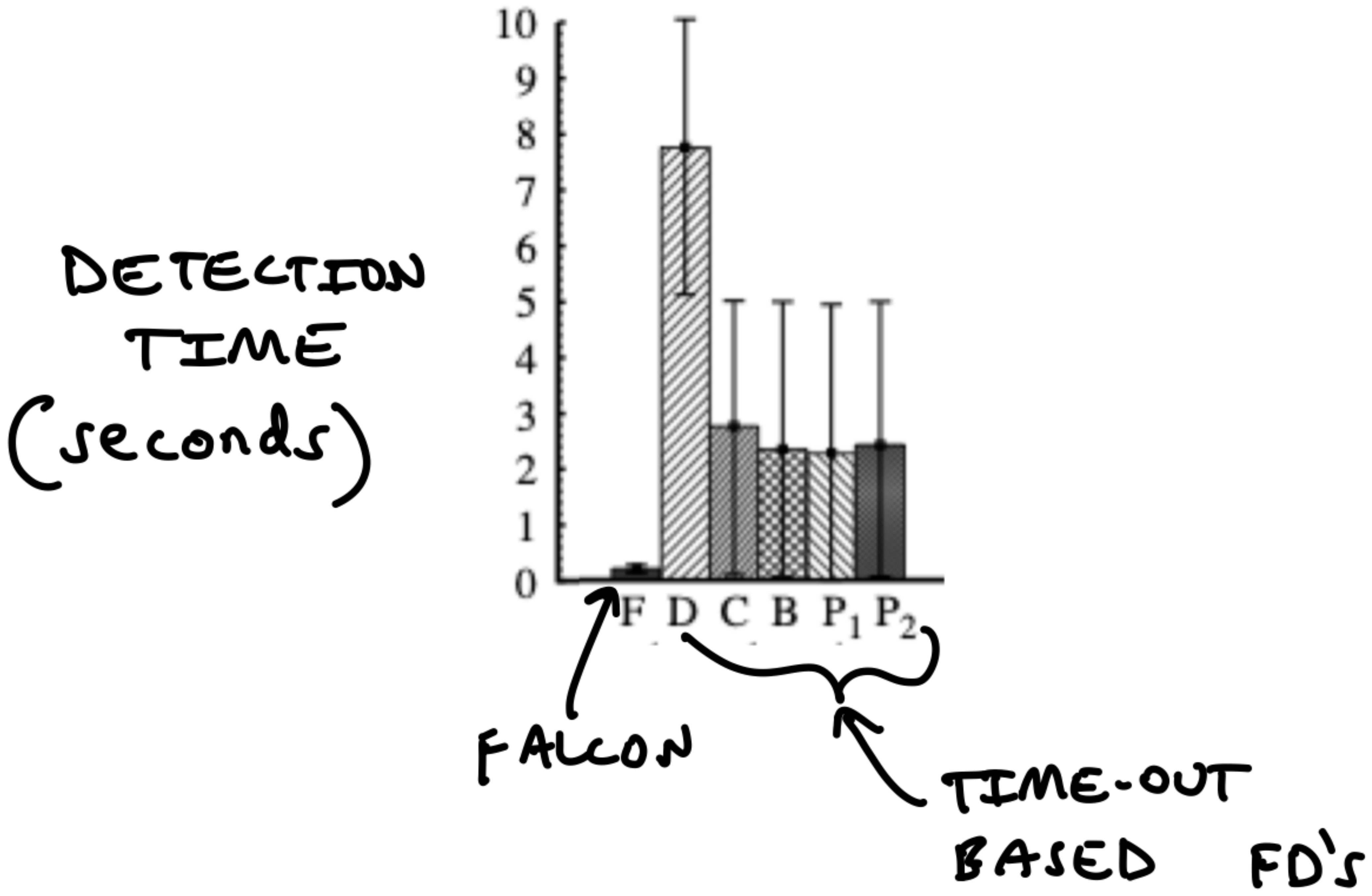
Evaluating FALCON

- **Criteria**:
  1. Failure detection time (and thus system availability)
  2. Disruption (when and how much killing)
  3. Computational complexity
  4. Code complexity

Detection Time and Availability

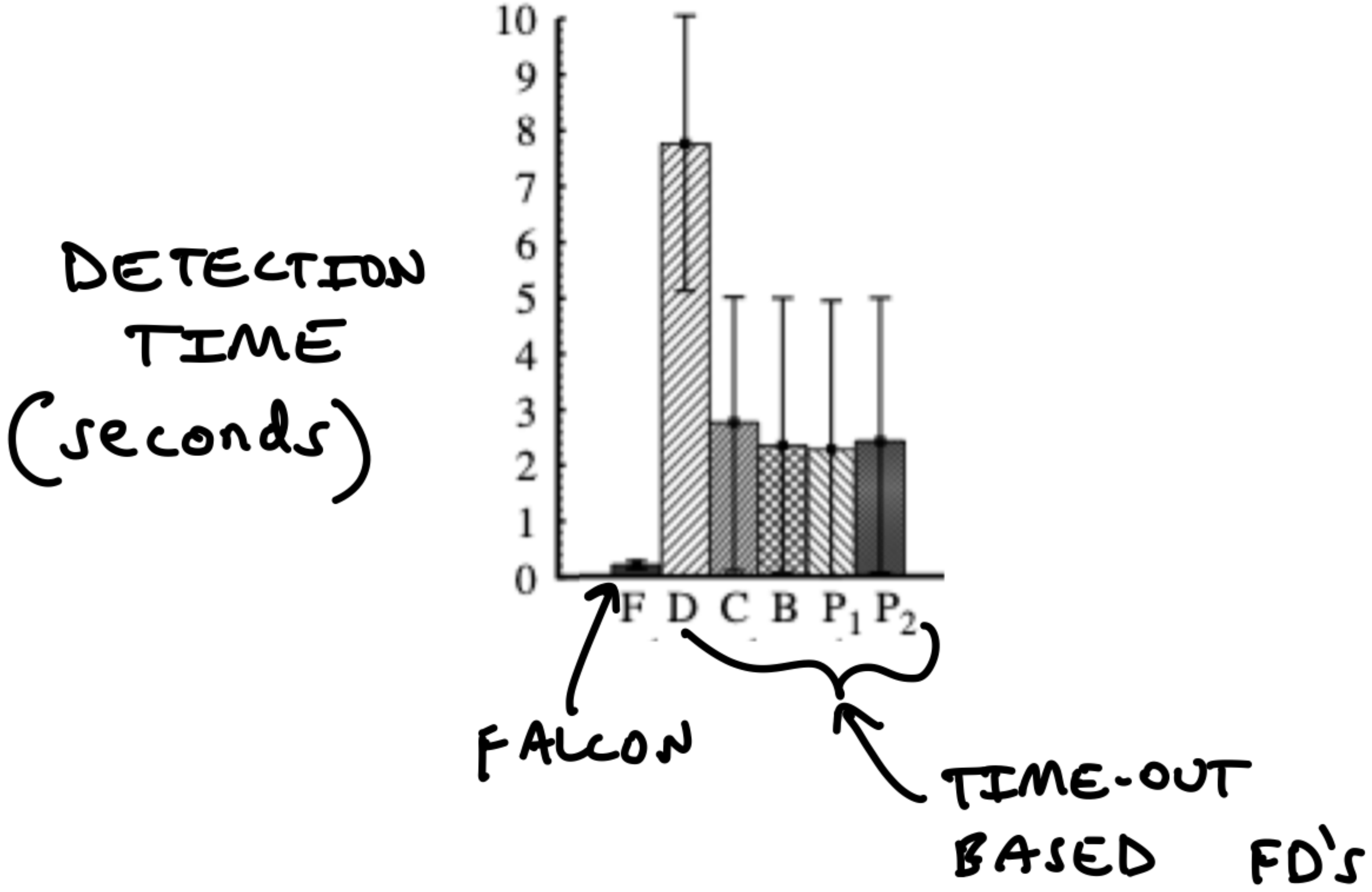
Detection Time and Availability

- Sub-second detection time:



Detection Time and Availability

- Sub-second detection time:



- Reduces median ZooKeeper unavailability time by ~6x (kernel/host crashes)

System Disruption

System Disruption

- Usually **minimal killing**
- Few cases where it goes overboard:
  - E.g., Network load causes VMM spy enforcer to suspect death (and kill VMM)
  - **Can be fixed** by better enforcer logic!

Computational and Code Complexity

Computational and Code Complexity

- Generally **low cost:** < 1% CPU overhead

Computational and Code Complexity

- Generally **low cost:** < 1% CPU overhead
- Reduces code complexity:
  - FALCON is a **R**FD $\Longrightarrow$ don't need to handle failure mistakes
  - **Primary-backup + FALCON = 21% less code** than Paxos + timeout!

More Discussion In The Paper

More Discussion In The Paper

- FALCON after recovery
- FALCON for different platforms
- More evaluation against large failure and stress-test suite