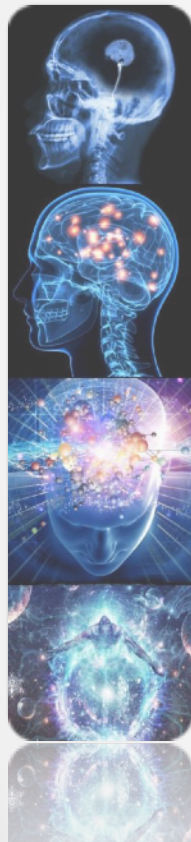


Advanced Course Distributed Systems

Failure Detectors

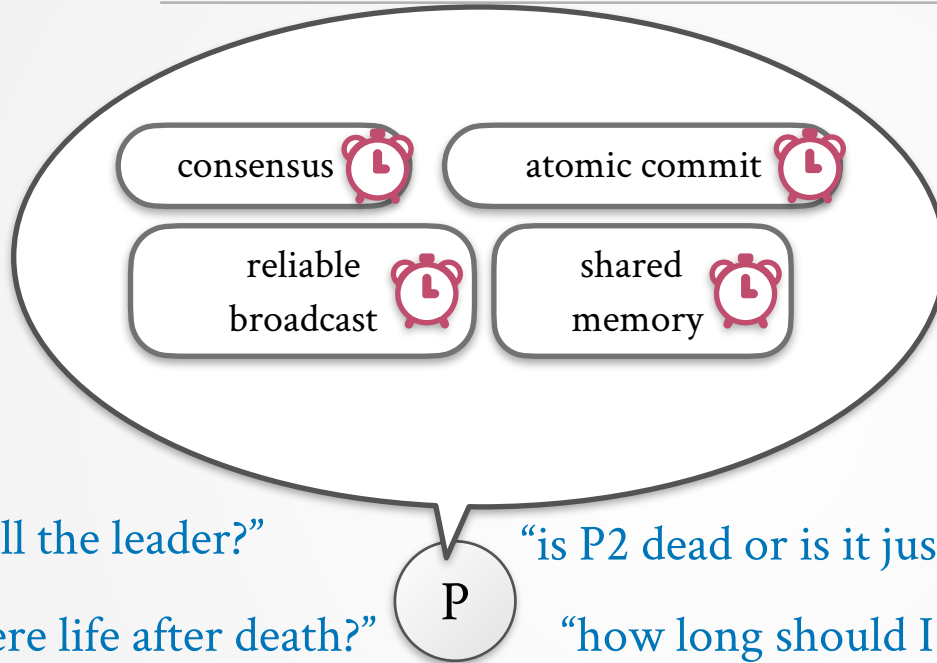


COURSE TOPICS



- ▶ Intro to Distributed Systems
- ▶ Fundamental Abstractions and Failure Detectors
- ▶ Reliable and Causal Order Broadcast
- ▶ Distributed Shared Memory-CRDTs
- ▶ Consensus (Paxos)
- ▶ Replicated State Machines (OmniPaxos, Raft, Zab etc.)
- ▶ Time Abstractions and Interval Clocks (Spanner etc.)
- ▶ Consistent Snapshotting (Stream Data Management)
- ▶ Distributed ACID Transactions (Cloud DBs)

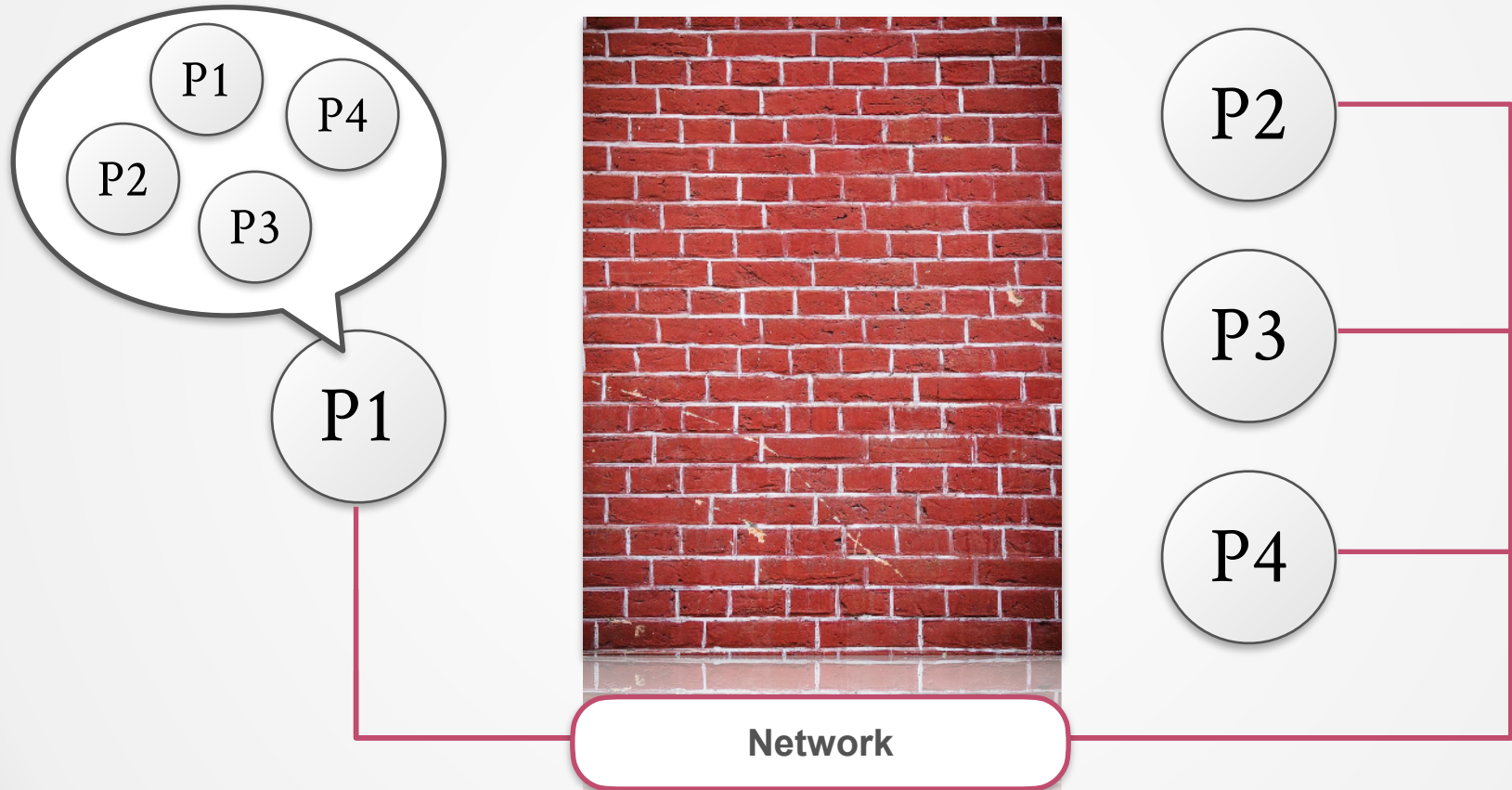
MOTIVATION



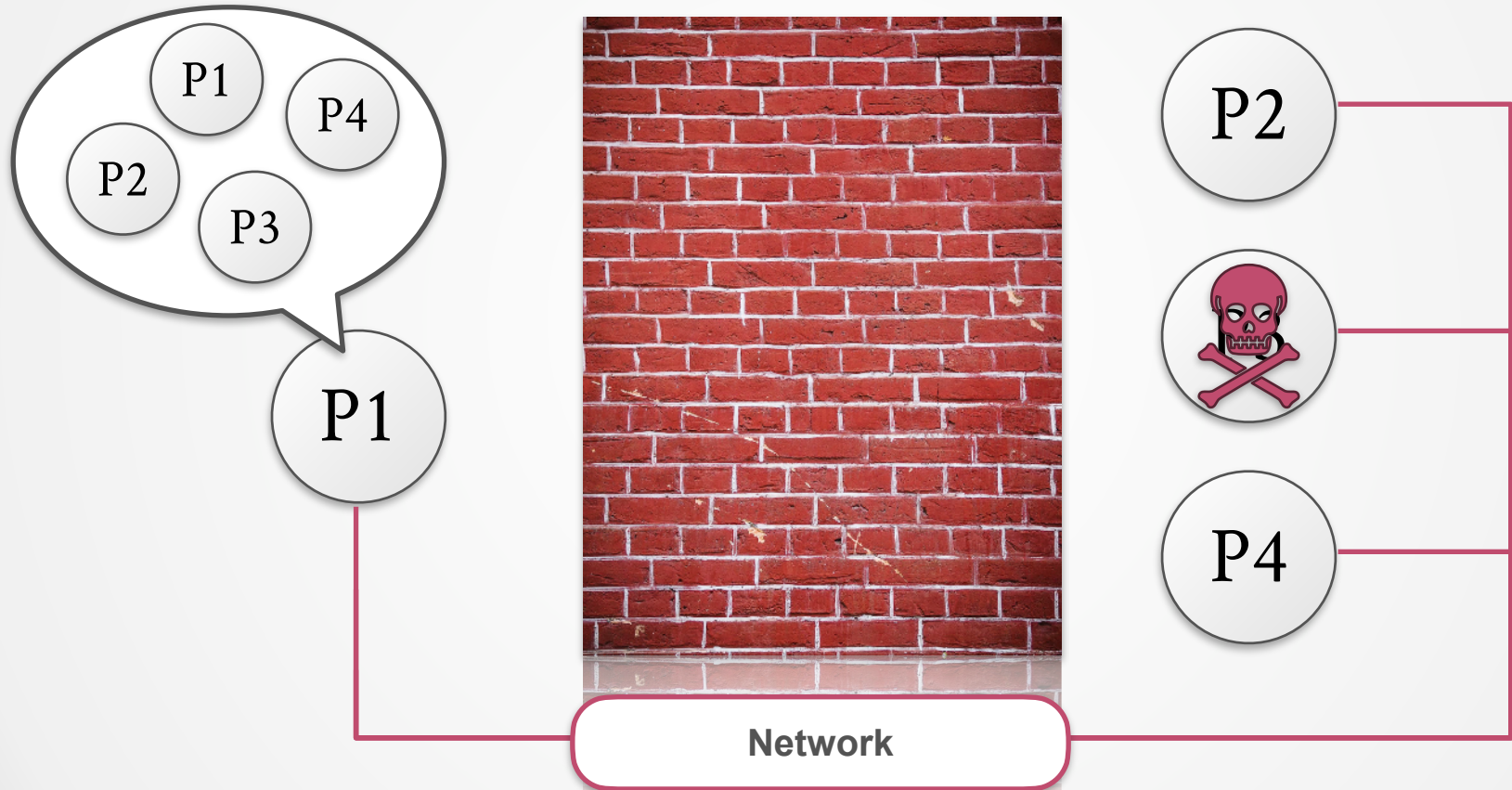
- Protocols in synchronous/ eventually synchronous models require heavy use of **timers** (Tideous).

- Time-related questions need to be abstracted..somehow

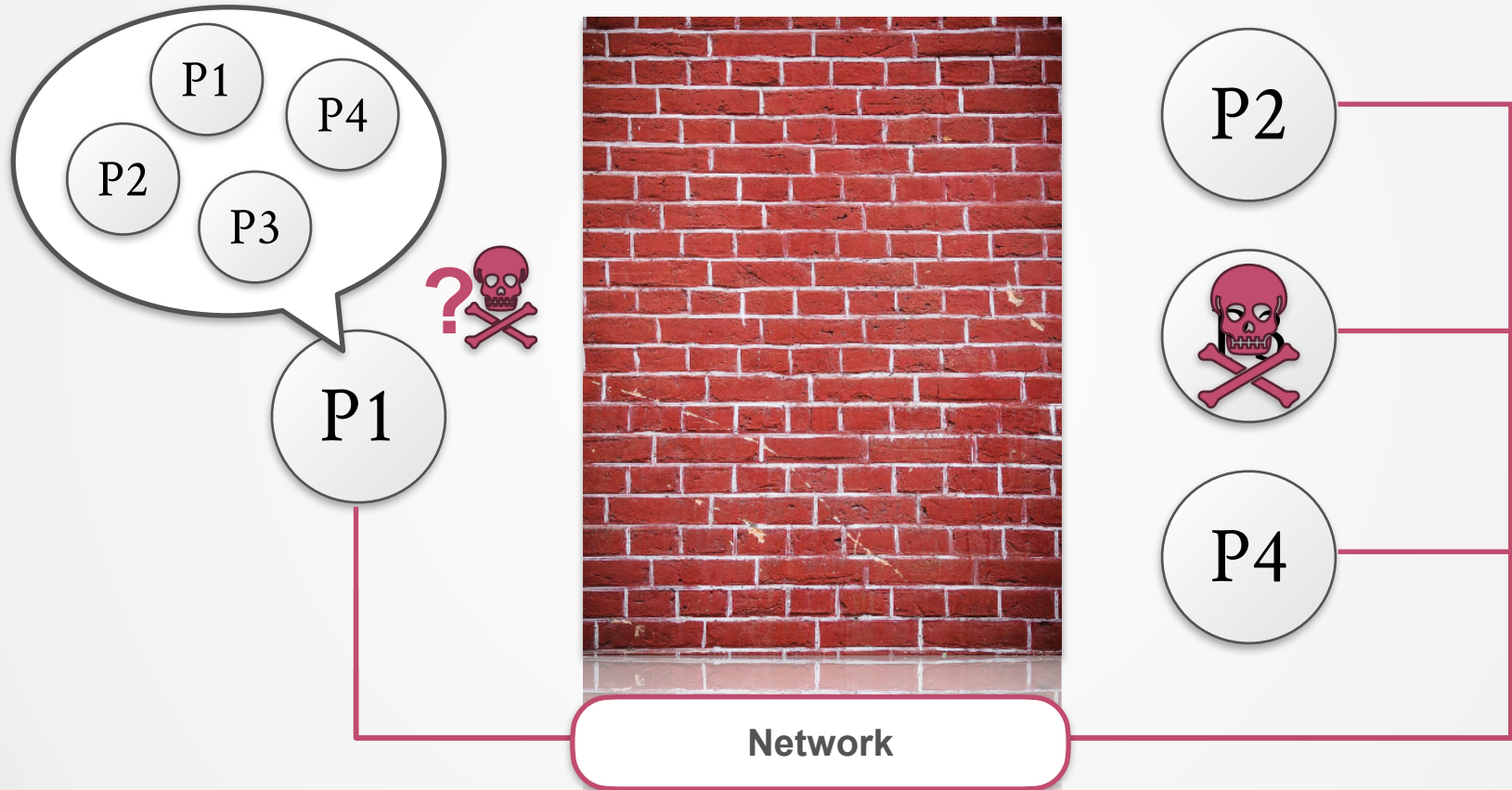
THE PROBLEM



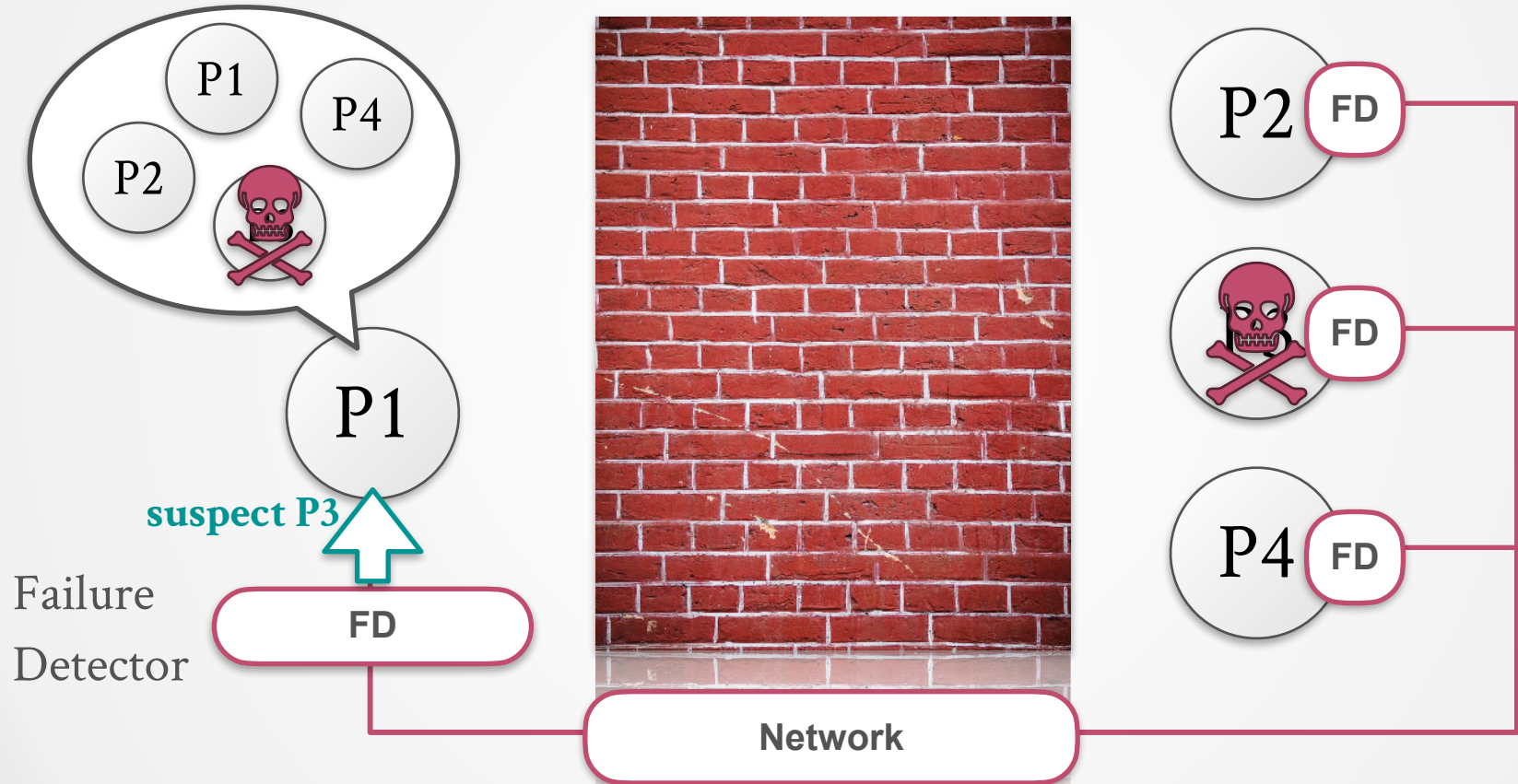
THE PROBLEM



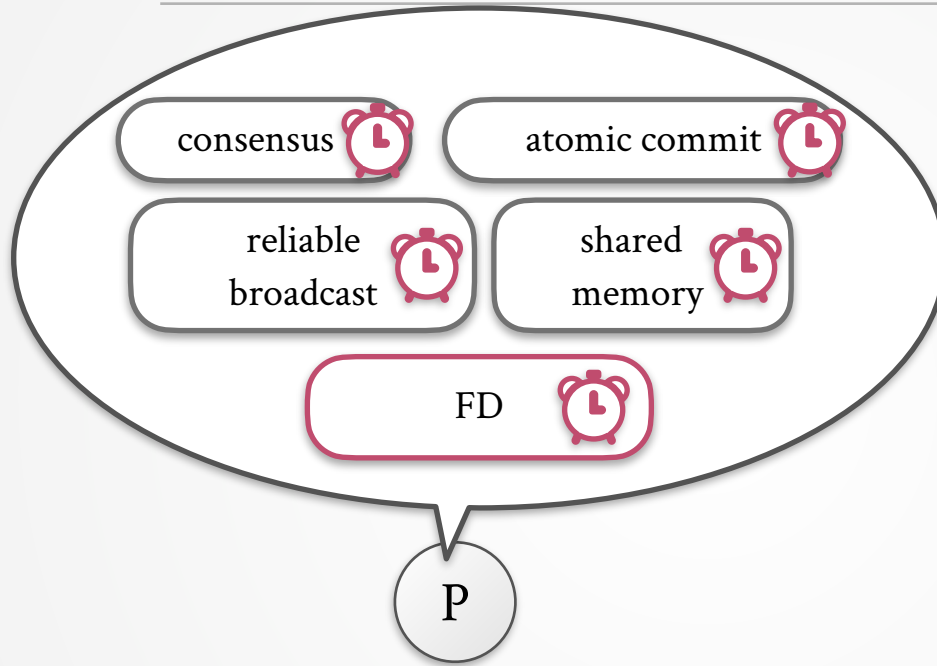
THE PROBLEM



THE PROBLEM



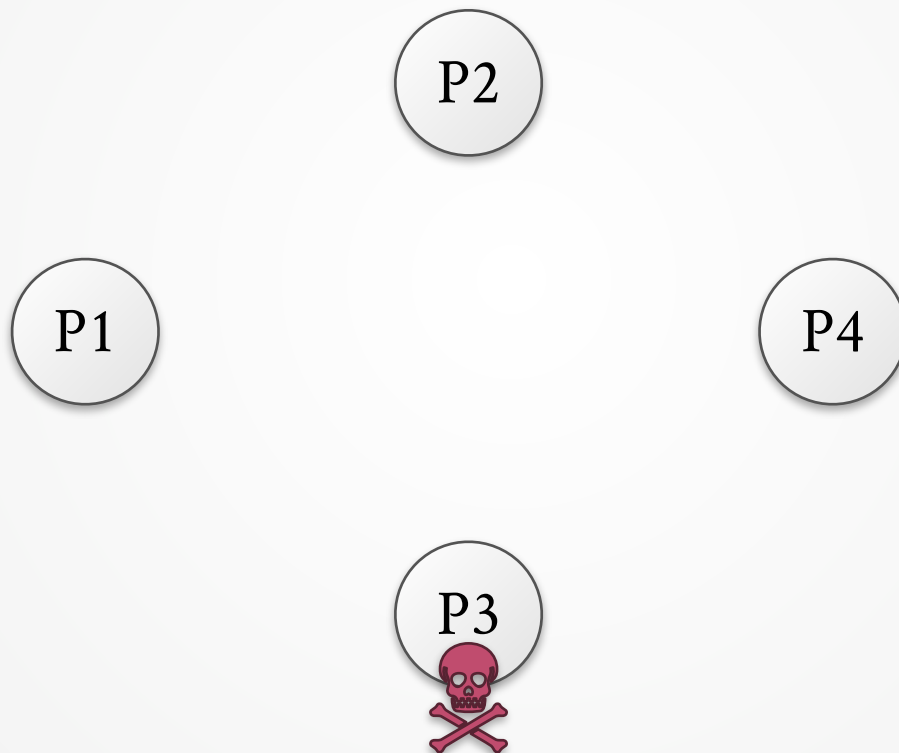
MOTIVATION



- A Failure Detector can substitute timing assumptions.

- **Spoiler Alert:** The Accuracy of a FD relates to the strength of the underlying model.

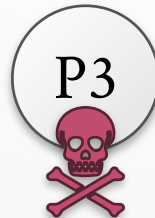
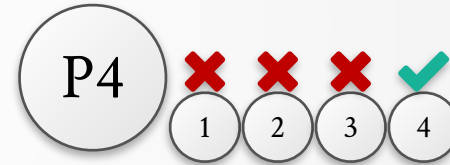
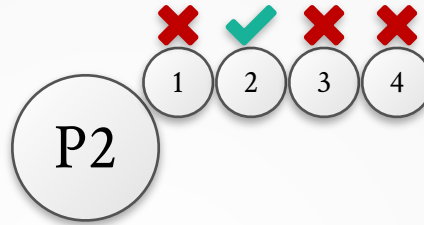
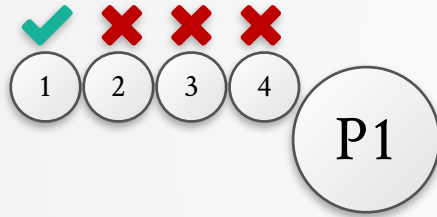
IDEA SKETCH





HEARTBEATS

 = heartbeat timer

 = max waiting time

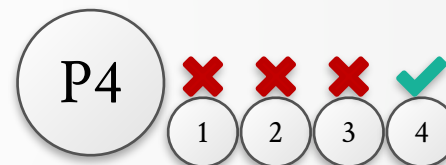
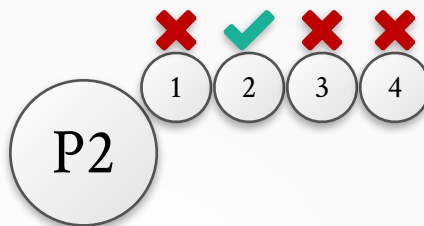
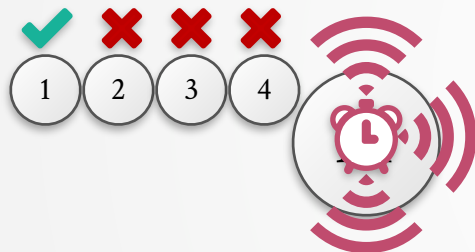




 not suspect
 suspect

HEARTBEATS

 = heartbeat timer

 = max waiting time

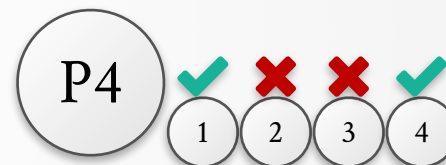
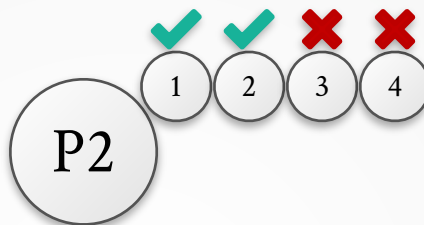
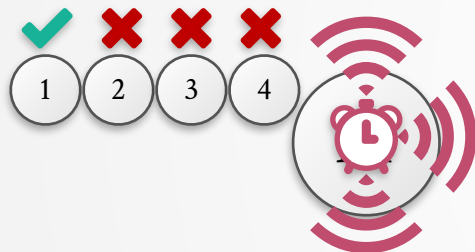




 not suspect
 suspect

HEARTBEATS

 = heartbeat timer

 = max waiting time

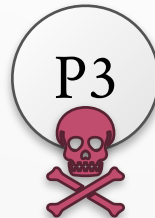
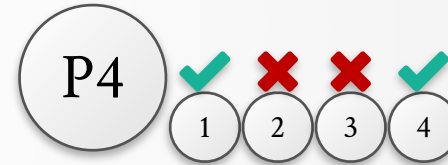
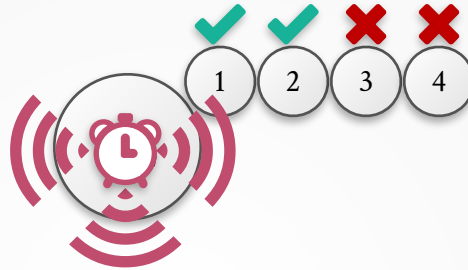
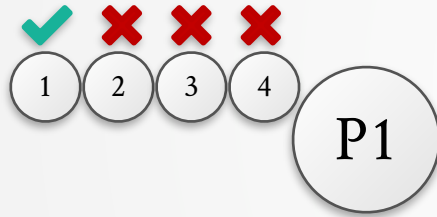




 not suspect
 suspect

HEARTBEATS

 = heartbeat timer

 = max waiting time

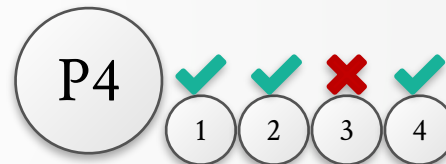
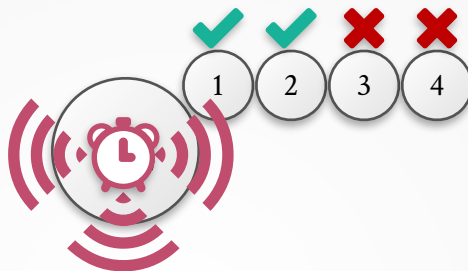
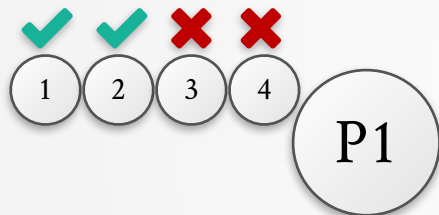




 not suspect
 suspect

HEARTBEATS

 = heartbeat timer

 = max waiting time

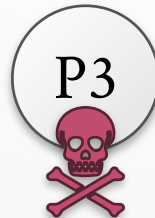
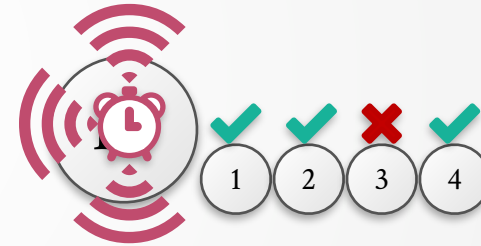
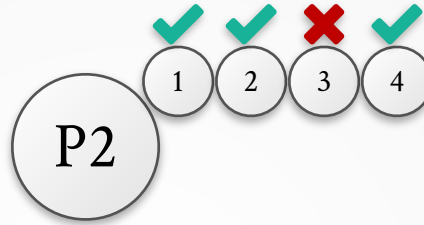
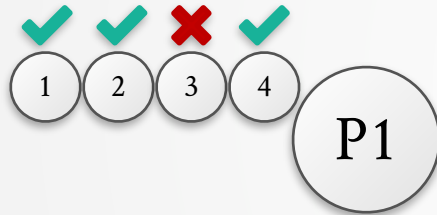




 not suspect
 suspect

HEARTBEATS

 = heartbeat timer

 = max waiting time

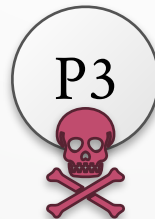
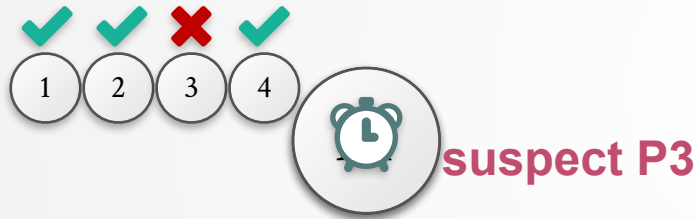




 not suspect
 suspect

HEARTBEATS

 = heartbeat timer

 = max waiting time



 not suspect
 suspect

IMPLEMENTATION IDEA

- Periodically exchange **heartbeat** messages
- **Timeout** based on **worst case** message round trip
 - If timeout, then **suspect** process
 - If received message from suspected node, **revise suspicion** and increase time-out

COMPLETENESS AND ACCURACY

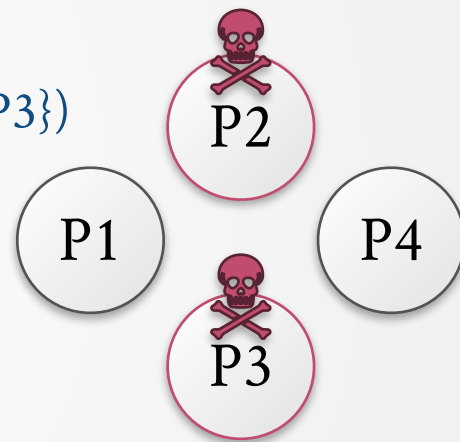
- Two important types of requirements

1. Completeness

- No False-Negatives! (i.e., should suspect at least $\{P2, P3\}$)
- When do they have to be suspected?

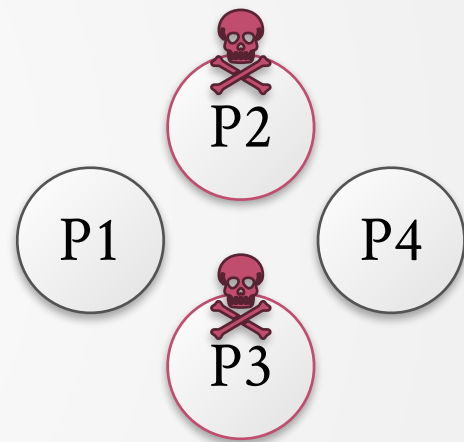
2. Accuracy

- No False-Positives! (i.e., should not suspect $\{P1, P4\}$)
- When are they allowed to be suspected?



COMPLETENESS AND ACCURACY

- Assume the **asynchronous** system model
 - Is it possible to achieve completeness?
 - Yep, suspect all processes (i.e., $\{P1, P2, P3, P4\}$)
 - Is it possible to achieve accuracy?
 - Yep, suspect none (i.e., $\{\}$)
 - How about achieving both?
 - Bad News 🙄 : Failure detectors are feasible **only** in synchronous and partially synchronous systems



REQUIREMENTS: COMPLETENESS

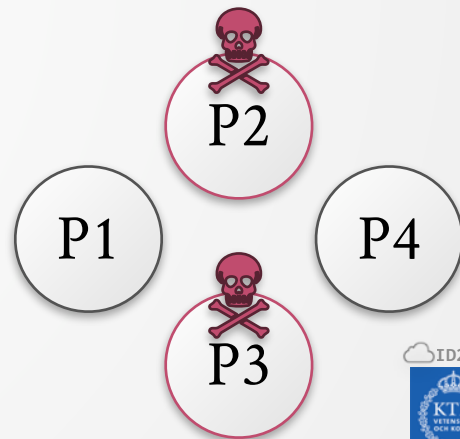
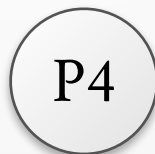
- Strong Completeness

- Every crashed process is **eventually** detected by all **correct** processes
- There exists a time after which...**all** crashed processes are detected by **all** correct processes
 - We only study failure detectors with this property

$\{P2, P3\}$



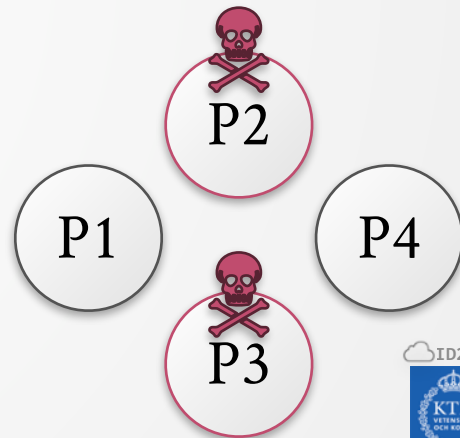
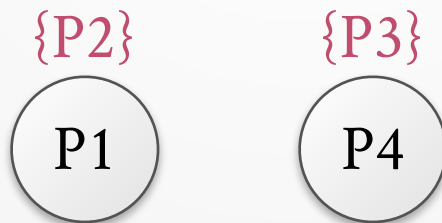
$\{P2, P3\}$



REQUIREMENTS: COMPLETENESS

- Weak Completeness

- Every crashed process is **eventually** detected by some **correct** process
- There exists a time after which...**all** crashed processes are detected by **some** correct processes
 - Possibly detected by **different** correct processes



REQUIREMENTS: ACCURACY

- Strong Accuracy
 - No correct process is ever suspected
 - For any process pair **p and q**,
 - p does not suspect q, unless q has crashed
 - Is it realistic?
 - Strong assumption, requires synchrony
 - I.e. no premature timeouts

REQUIREMENTS: ACCURACY

- **Weak Accuracy**
- There exists a correct process P which is never suspected by any process.
- Still strong assumption
 - One “well-connected” process in the system.

REQUIREMENTS: ACCURACY

Eventual Strong Accuracy

After some finite time the FD provides **strong accuracy**

Eventual Weak Accuracy

After some finite time the detector provides **weak accuracy**

...Prior to that, any behaviour is possible.

Quite weak assumptions

When can eventual weak accuracy be achieved?

Classes of Failure Detectors

THE PRACTICAL FDs

Four detectors with **strong completeness**

Perfect Detector (P)

Strong Accuracy

Strong Detector (S)

Weak Accuracy

Synchronous Systems

Eventually Perfect Detector ($\diamond P$)

Eventual Strong Accuracy

Eventually Strong Detector ($\diamond S$)

Eventual Weak Accuracy

Partially Synchronous
Systems

LESS INTERESTING FDs

Four detectors with **weak completeness**

Detector Q

Strong Accuracy

Weak Detector (W)

Weak Accuracy

Synchronous Systems

Eventually Detector Q ($\diamond Q$)

Eventual Strong Accuracy

Eventually Weak Detector ($\diamond W$)

Eventual Weak Accuracy

Partially Synchronous
Systems

Perfect Failure Detector - P

INTERFACE OF PERFECT FAILURE DETECTOR

Module:

Name: PerfectFailureDetector, instance **P**

Events:

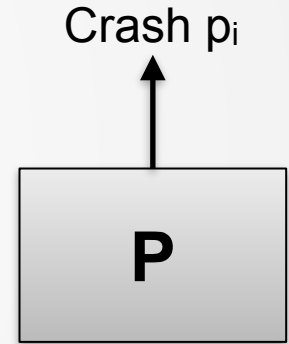
Indication (out): $\langle \mathbf{P}, \text{Crash} \mid p_i \rangle$

Notifies that process p_i has crashed

Properties:

PFD1 (strong completeness)

PFD2 (strong accuracy)

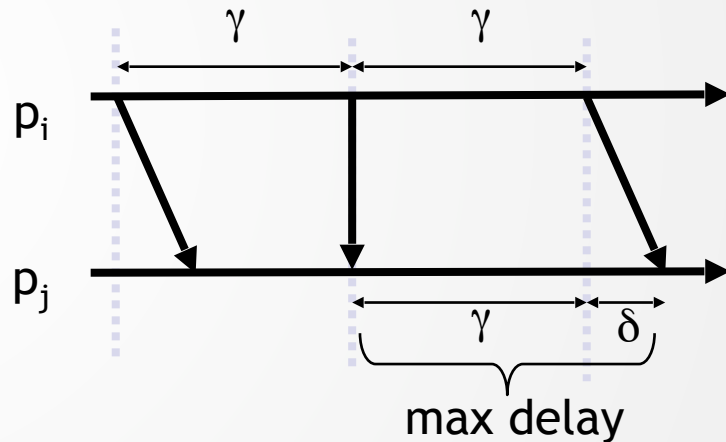


PROPERTIES OF P

- Properties:
 - *PFD1 (strong completeness)*
 - Eventually every process that **crashes** is permanently detected by every correct process (liveness)
 - *PFD2 (strong accuracy)*
 - If a node p is detected by any node, then p has crashed (safety)
- Safety or Liveness?

IMPLEMENTING P IN SYNCHRONY

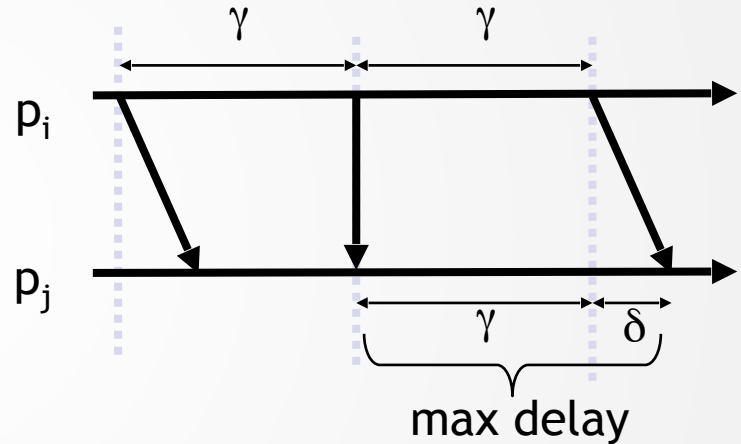
- Assume **synchronous system**
 - Max transmission delay between 0 and δ time units
- Each process every γ time units
 - Send <heartbeat> to all processes
- Each process waits $\gamma + \delta$ time units
 - If did not get <heartbeat> from p_i
 - Detect <crash | p_i >



CORRECTNESS OF P

PFD1 (strong completeness)

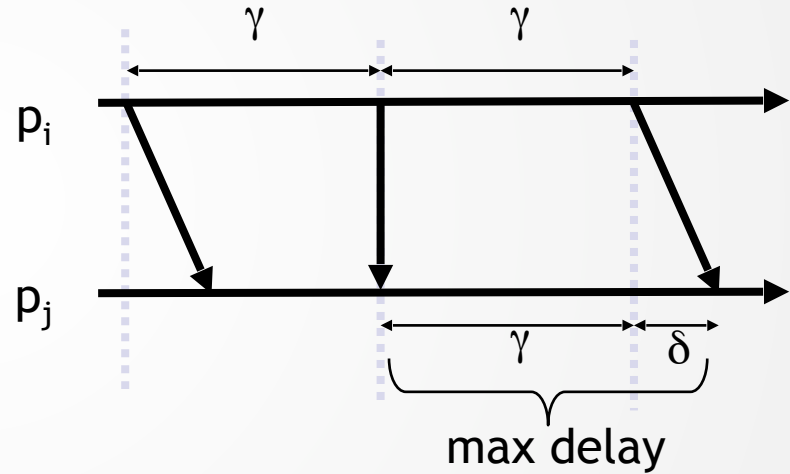
- A crashed process doesn't send <heartbeat>
- Eventually every process will notice the absence of <heartbeat>



CORRECTNESS OF P

PFD2 (strong accuracy)

- Assuming local computation is negligible
- Maximum time between 2 heartbeats
 - $\gamma + \delta$ time units
- If alive, all process will receive hb in time
 - No inaccuracy



Eventually Perfect Failure Detector - $\diamond P$

INTERFACE OF $\diamond P$

Module:

Name: EventuallyPerfectFailureDetector, **instance** $\diamond P$

Events:

Indication: $\langle \diamond P, \text{suspect} \mid p_i \rangle$

Notifies that process p_i is suspected to have crashed

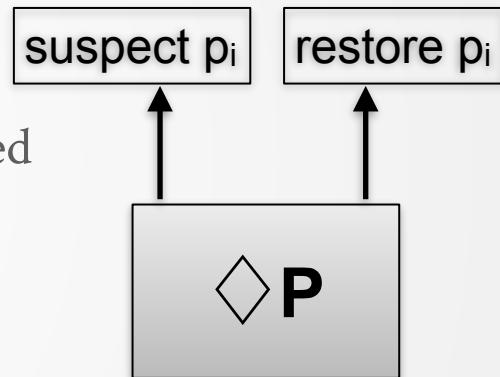
Indication: $\langle \diamond P, \text{restore} \mid p_i \rangle$

Notifies that process p_i is not suspected anymore

Properties:

PFD1 (strong completeness)

PFD2 (eventual strong accuracy). **Eventually**, no correct process is suspected by any correct process

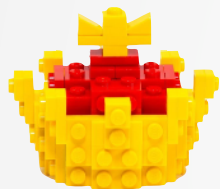


IMPLEMENTING $\diamond P$

- Assume **partially synchronous system**
 - Eventually some bounds exists
- Each process every γ time units
 - Send <heartbeat> to all processes
- Each process waits T time units
 - If did not get <heartbeat> from p_i
 - Indicate <suspect | p_i > if p_i is not in **suspected** set
 - Put p_i in **suspected** set
 - If get HB from p_i , and p_i is in **suspected**
 - Indicate <restore | p_i > and remove p_i from **suspected**
 - Increase timeout T

CORRECTNESS OF $\Diamond P$

- *EPFD1 (strong completeness)*
 - Same as before
- *EPFD2 (eventual strong accuracy)*
 - Each time p is inaccurately suspected by a correct q
 - Timeout T is increased at q
 - Eventually system becomes synchronous, and T becomes larger than the **unknown bound** δ ($T > \gamma + \delta$)
 - q will receive HB on time, and never suspect p again



Leader Election

LEADER ELECTION VS FAILURE DETECTION

- Failure detection captures failure behaviour
 - Detect **failed** processes
- Leader election (LE) also captures failure behaviour
 - Detect **correct** processes (a single **and** same for all)
- Formally, **leader election is a FD**
 - Always suspects all processes except one (leader)
 - Ensures some properties regarding that process

LEADER ELECTION VS. FAILURE DETECTION

We will define two leader election abstractions and algorithms

- Leader election (LE) which “matches” P
- Eventual leader election (Ω) which “matches” $\Diamond P$

MATCHING LE AND P

P's properties

P always eventually detects failures (strong completeness)

P never suspects correct processes (strong accuracy)

Completeness of LE

Informally: eventually ditch **failed leaders**

Formally: **eventually** every correct process trusts **some** correct process

Accuracy of LE

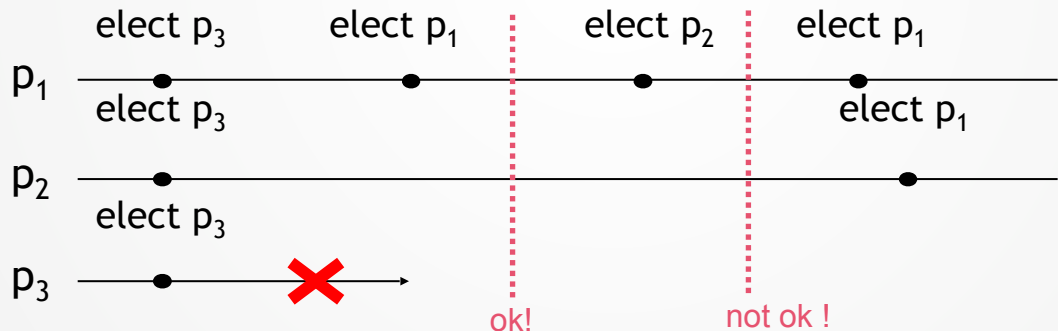
Informally: never ditch a correct leader

Formally: No two **correct** processes trust different **correct** nodes

- Is this really accuracy?
- Yes! Assume two processes trust different correct processes
 - One of them must eventually switch, i.e. leaving a correct node

LE DESIRABLE PROPERTIES

- LE always eventually detects failures
 - Eventually every correct process trusts some correct node
- LE is always accurate
 - No two correct processes trust different correct processes
- But the above two permit the following



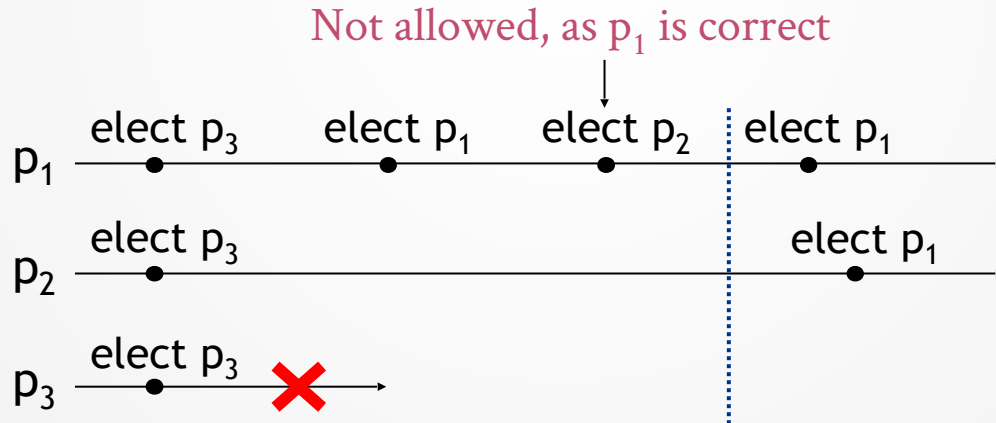
- But P_1 is “inaccurately” leaving a correct leader

LE DESIRABLE PROPERTIES

To avoid “inaccuracy” we add

Local Accuracy:

If a process is elected leader by p_i , all previously elected leaders by p_i have crashed



INTERFACE OF LEADER ELECTION

Module:

Name: LeaderElection (le)

Events:

Indication: $\langle \text{leLeader} \mid p_i \rangle$

Indicate that leader is node p_i

Properties:

- **LE1 (eventual completeness).** Eventually every correct process trusts some correct process
- **LE2 (agreement).** No two correct processes trust different correct processes
- **LE3 (local accuracy).** If a process is elected leader by p_i , all previously elected leaders by p_i have crashed

IMPLEMENTING LE

- Globally rank all processes

E.g. rank ordering $\text{rank}(p_1) > \text{rank}(p_2) > \text{rank}(p_3) > \dots$

- $\text{maxrank}(S)$

The process $p \in S$, with the largest rank

IMPLEMENTING LE

LeaderElection, **instance** le

Uses:

PerfectFailureDetector, **instance** P

upon event $\langle \text{le}, \text{Init} \rangle$ **do**

suspected $:= \emptyset$

leader $:= \perp$

upon event $\langle \text{P}, \text{Crash } |p\rangle$ **do**

suspected $:= \text{suspected} \cup \{p\}$

upon leader $\neq \text{maxrank}(\Pi \setminus \text{suspected})$ **do**

leader $:= \text{maxrank}(\Pi \setminus \text{suspected})$

trigger $\langle \text{le}, \text{Leader } | \text{leader} \rangle$

Eventual Leader Election - Ω

MATCHING Ω AND $\diamond P$

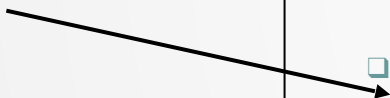
$\diamond P$ weakens P by only providing eventual accuracy

Weaken LE to Ω by only guaranteeing **eventual** agreement

LE Properties:

- ❑ **LE1 (eventual completeness).** Eventually every correct node trusts some correct node
- ❑ **LE2 (agreement).** No two correct nodes trust different correct nodes
- ❑ ~~**LE3 (local accuracy).** If a node is elected leader by p_i , all previously elected leaders by p_i have crashed~~

eventual



INTERFACE OF EVENTUAL LEADER ELECTION

Module:

Name: EventualLeaderElection (Ω)

Events:

Indication (out): $\langle \Omega, \text{Trust} \mid p_i \rangle$

Notify that p_i is trusted to be leader

Properties:

ELD1 (eventual completeness). **Eventually** every correct node trusts some correct node

ELD2 (eventual agreement). **Eventually** no two correct nodes trust different correct node

EVENTUAL LEADER DETECTION Ω

In crash-stop process abstraction

Ω is obtained directly from $\diamond P$

- Each process trusts the process with highest rank among all processes not suspected by $\diamond P$
- Eventually, exactly one correct process will be trusted by all correct processes

IMPLEMENTING Ω

EventualLeaderElection, **instance** Ω

Uses: EventuallyPerfectFailureDetector, **instance** $\diamond P$

upon event $\langle \Omega, \text{Init} \rangle$ **do**

$\text{suspected} := \emptyset; \text{leader} := \perp$

upon event $\langle \diamond P, \text{Suspect } |p\rangle$ **do**

$\text{suspected} := \text{suspected} \cup \{p\}$

upon event $\langle \diamond P, \text{Restore } |p\rangle$ **do**

$\text{suspected} := \text{suspected} \setminus \{p\}$

upon $\text{leader} \neq \text{maxrank}(\Pi \setminus \text{suspected})$ **do**

$\text{leader} := \text{maxrank}(\Pi \setminus \text{suspected})$

trigger $\langle \Omega, \text{Trust } | \text{leader} \rangle$

Ω FOR CRASH RECOVERY

Can we elect a recovered process?

Not if it keeps crash-recovering infinitely often!

Basic idea

Count number of times you've crashed (**epoch**)

Distribute your **epoch** periodically to all nodes

Elect leader with lowest (**epoch, rank(node)**)

Implementation

Similar to $\diamond P$ and Ω for crash-stop

Piggyback **epoch** with heartbeats

Store **epoch**, upon recovery load **epoch** and increment

Reductions

REDUCTIONS

We say $X \leq Y$ if

- X can be solved given a solution of Y
- Read X is reducible to Y
- Informally, problem X is easier or as hard as Y

PREORDERS, PARTIAL ORDERS...

- A relation \preceq is a **preorder** on a set A if for any x, y, z in A
 - $x \preceq x$ (**reflexivity**)
 - $x \preceq y$ and $y \preceq z$ implies $x \preceq z$ (**transitivity**)
- Difference between preorder and partial order
 - Partial order is a preorder with **anti-symmetry**
 - $x \preceq y$ and $y \preceq x$ implies $x = y$
- For **preorder** two different objects x and y can be symmetric
 - It is possible that $x \preceq y$ and $y \preceq x$ for two **different** x and y , ($x \neq y$)

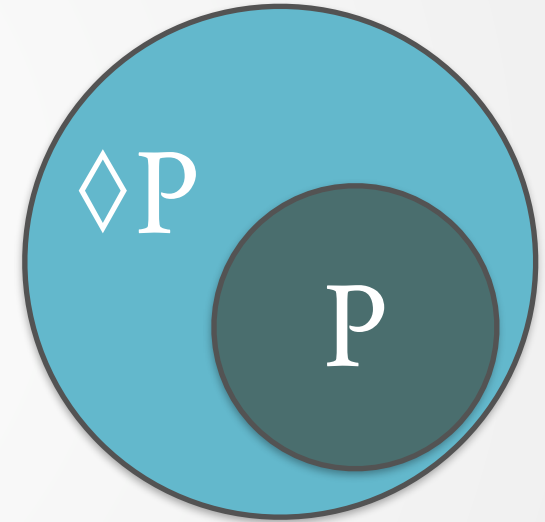
SHORTCUT DEFINITIONS

- We write $X \approx Y$ if
 - $X \leq Y$ and $Y \leq X$
 - Problem X is **equivalent** to Y
- We write $X < Y$ if
 - $X \leq Y$ and not $X \approx Y$
 - or equivalently, $X \leq Y$ and not $Y \leq X$
 - Problem X is **strictly weaker** than Y , or
 - Problem Y is **strictly stronger** than X

EXAMPLE

- It is true that $\Diamond P \leq P$
 - Given P , we can implement $\Diamond P$
 - We just return P 's suspicions.
 - P always satisfies $\Diamond P$'s properties
- In fact, $\Diamond P < P$ in the asynchronous model
 - Because not $P \leq \Diamond P$ is true
- Reductions common in computability theory
 - If $X \leq Y$, and if we know X is **impossible** to solve
 - Then Y is impossible to solve too
 - If $\Diamond P \leq P$, and some problem Z can be solved with $\Diamond P$
 - Then Z can also be solved with P

Possible Traces



WEAKEST FD FOR A PROBLEM?

- Often P is used to solve problem X
 - But P is not very practical (needs synchrony)
 - Is X a “practically” solvable problem?
 - Can we implement X with $\diamond P$?
 - Sometimes a weaker FD than P will not solve X
 - Proven using reductions

WEAKEST FD FOR A PROBLEM

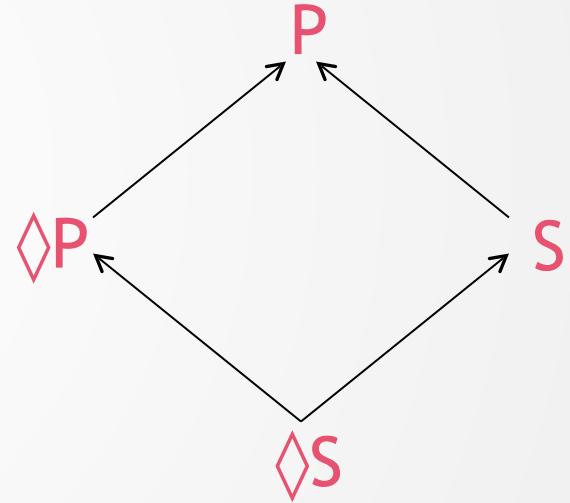
- Common proof to show P is weakest FD for X
 - Prove that $P \leq X$
 - I.e. P can be solved given X
- If $P \leq X$ then $\Diamond P < X$
 - Because we know $\Diamond P < P$ and $P \approx X$, i.e. $\Diamond P < P \approx X$
 - If we can solve X with $\Diamond P$, then
 - we can solve P with $\Diamond P$, which is a contradiction

How are the detectors related

TRIVIAL REDUCTIONS

- Strongly complete

- $\Diamond P \leqslant P$
 - P is always strongly accurate, thus also eventually strongly accurate
- $\Diamond S \leqslant S$
 - S is always weakly accurate, thus also eventually weakly accurate
- $S \leqslant P$
 - P is always strongly accurate, thus also always weakly accurate
- $\Diamond S \leqslant \Diamond P$
 - $\Diamond P$ is always eventually strongly accurate, thus also always eventually weakly accurate



TRIVIAL REDUCTIONS (2)

- Weakly complete

- $\Diamond Q \leqslant Q$

- Q is always strongly accurate, thus also eventually strongly accurate

- $\Diamond W \leqslant W$

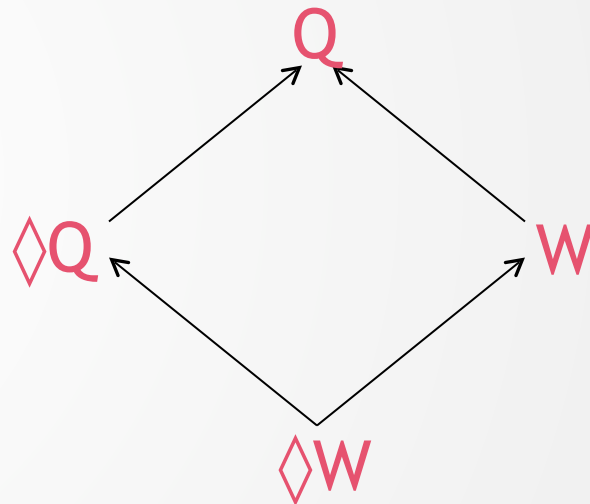
- W is always weakly accurate, thus also eventually weakly accurate

- $W \leqslant Q$

- Q is always strongly accurate, thus also always weakly accurate

- $\Diamond W \leqslant \Diamond Q$

- $\Diamond Q$ is always eventually strongly accurate, thus also always eventually weakly accurate




COMPLETENESS “IRRELEVANT”

- Weak completeness **trivially reducible** to strong
- Strong completeness **reducible** to weak
 - i.e. can get strong completeness from weak
 - $P \leq Q, S \leq W, \Diamond P \leq \Diamond Q, \Diamond S \leq \Diamond W,$
- They're **equivalent!**
 - $P \approx Q, S \approx W, \Diamond P \approx \Diamond Q, \Diamond S \approx \Diamond W$

		Accuracy			
		Strong	Weak	Eventual Strong	Eventual Weak
Completeness	Strong	P	S	$\Diamond P$	$\Diamond S$
	Weak	Q	W	$\Diamond Q$	$\Diamond W$

PROVING IRRELEVANCE OF COMPLETENESS

- Weak completeness ensures
 - every crash is eventually detected by some correct node
- Simple idea
 - Every process q broadcast suspicions **Susp** periodically
 - upon event receive $\langle S, q \rangle$ 
 - $\text{Susp} := (\text{Susp} \cup S) - \{q\}$
- Every crash is eventually detected by all correct p
 - Can this violate some accuracy properties?

**also works like a
heartbeat**

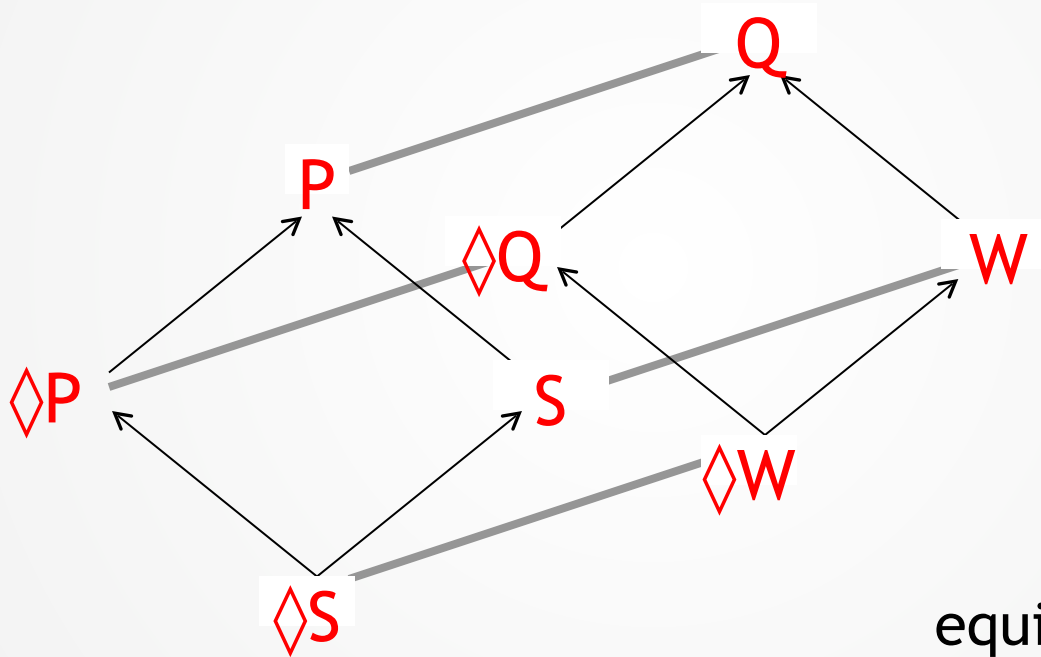
MAINTAINING ACCURACY

- Strong and Weak Accuracy aren't violated
- Strong accuracy
 - No one is ever inaccurate
 - Our reduction never spreads inaccurate suspicions
- Weak accuracy
 - Everyone is accurate about at least one process p
 - No one will spread inaccurate information about p

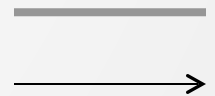
MAINTAINING EVENTUAL ACCURACY

- Eventual Strong and Eventual Weak Accuracy aren't violated
- Proof is almost same as previous page
 - Eventually all faulty processes crash
 - Inaccurate suspicions **undone**
 - Will get heartbeat from correct nodes and revise $(-\{q\})$

RELATION BETWEEN FDs



equivalent
reducible to



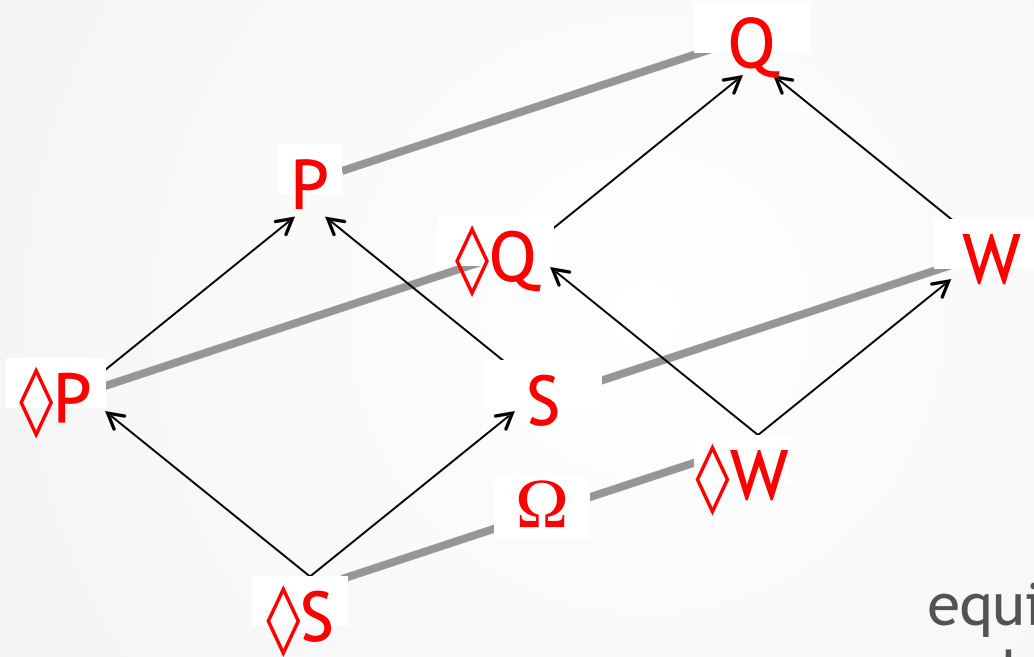
Ω ALSO A FD

- Can we implement $\diamond S$ with Ω ?
 - I.e. is it true that $\diamond S \leq \Omega$
 - Suspect all nodes except the leader given by Ω
 - Eventual Completeness
 - All nodes are suspected except the leader (which is correct)
 - Eventual Weak Accuracy
 - Eventually, one correct node (leader) is not suspected by anyone
- Thus, $\diamond S \leq \Omega$

Ω EQUIVALENT TO $\Diamond S$ (AND $\Diamond W$)

- We showed $\Diamond S \leq \Omega$, it turns out we also have $\Omega \leq \Diamond S$
 - I.e. $\Omega \approx \Diamond S$
- The famous CHT (Chandra, Hadzilocas, Toueg) result
 - If consensus implementable with detector D
 - Then Omega can be implemented using D
 - I.e. if $\text{Consensus} \leq D$, then $\Omega \leq D$
 - Since $\Diamond S$ can be used to solve consensus, we have $\Omega \leq D$
 - Implies $\Diamond W$ is weakest detector to solve consensus

RELATION BETWEEN FDs (2)



Combining Abstractions

COMBINING ABSTRACTIONS

Fail-stop

Crash-stop process model

Perfect links + Perfect failure detector (P)

(synchronous)

Fail-silent

Crash-stop process model

Perfect links

(asynchronous)

Fail-noisy

Crash-stop process model

Perfect links + Eventually Perfect failure detector ($\diamond P$)

(partially synchronous)

Fail-recovery

Crash-recovery process model

Stubborn links + ...