Failure Detectors

Seif Haridi

haridi@kth.se

Modeling Timing Assumptions

- Tedious to model eventual synchrony (partial synchrony)
- Timing assumptions mostly needed to detect failures
  - Heartbeats, timeouts, etc…

- Use failure detectors to encapsulate timing assumptions
  - Black box giving suspicions regarding process failures
  - Accuracy of suspicions depends on model strength

Implementation of Failure Detectors

Typical Implementation

- 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 requirements
    - Requirements regarding actually crashed nodes
      - When do they have to be detected?
  - 2. Accuracy requirements
    - Requirements regarding actually alive nodes
      - When are they allowed to be suspected?

Completeness and Accuracy

- In asynchronous system
  - Is it possible to achieve completeness?
    - Yes, suspect all processes

  - Is it possible to achieve accuracy?
    - Yes, refrain from suspecting any process!

  - Is it possible to achieve both?
    - NO!
- 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

- Is it realistic? [d]

Requirements: Completeness

- *Weak Completeness*
  - Every crashed process is *eventually* detected by some correct process

- There exists a time after which all crashed nodes are detected by some correct nodes
  - Possibly detected by different correct nodes

Requirements: Accuracy

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

Requirements: Accuracy

- *Weak Accuracy*
  - There exists a correct process which is never suspected by any process

- There exists a correct node P
  - All nodes will never suspect P

- Still strong assumption
  - One node is always “well-connected”

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
- After some time, the requirements are fulfilled
  - Prior to that, any behavior is possible!

- Quite weak assumptions [d]
  - When can eventual weak accuracy be achieved?

Failure Detectors Classes

Four Main Established Detectors

- Four detectors with strong completeness
  - Perfect Detector (P)
    - Strong Accuracy
  - Strong Detector (S)
    - Weak Accuracy

Synchronous Systems

-
  - Eventually Perfect Detector (◊P)
    - Eventual Strong Accuracy
  - Eventually Strong Detector (◊S)
    - Eventual Weak Accuracy

Partially Synchronous Systems

Four Less Interesting Detectors

- Four detectors with weak completeness
  - Detector Q
    - Strong Accuracy
  - Weak Detector (W)
    - Weak Accuracy

Synchronous Systems

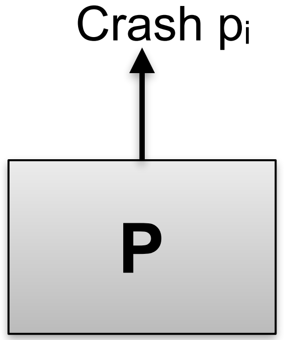
  - Eventually Detector Q (◊Q)
    - Eventual Strong Accuracy
  - Eventually Weak Detector (◊W)
    - Eventual Weak Accuracy

Partially Synchronous Systems

Prefect Failure Detector P

Interface of Perfect Failure Detector

- **Module:**
  - Name: PerfectFailureDetector, instance **P**
- **Events:**
  - **Indication (out):** ⟨**P**, Crash | p_i⟩
    - 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

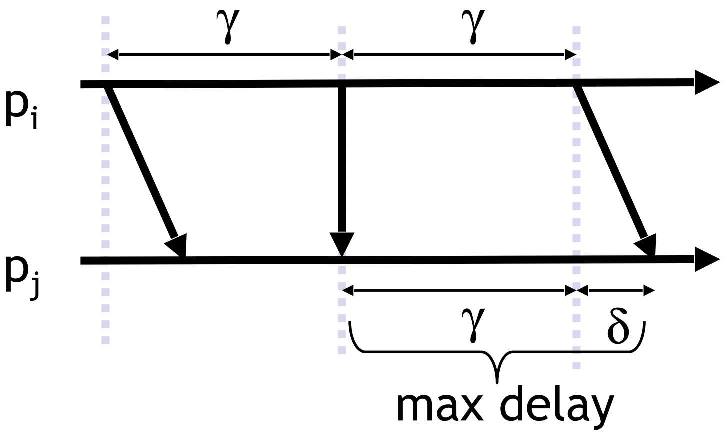
      (safety)
- Safety or Liveness?

Implementing P in Synchrony

- Assume synchronous system
  - Max transmission delay between 0 and δ time units

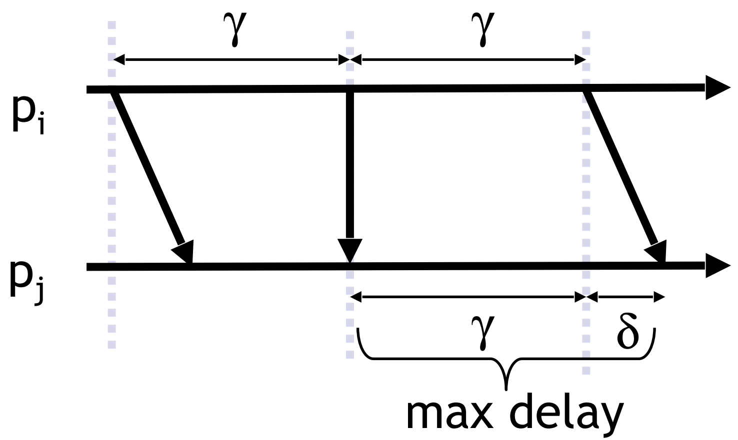
- Each process every γ time units
  - Send <heartbeat> to all processes

- Each process waits $\gamma+\delta$ time units
  - If did not get <heartbeat> from p_i
    - Detect <crash | p_i>



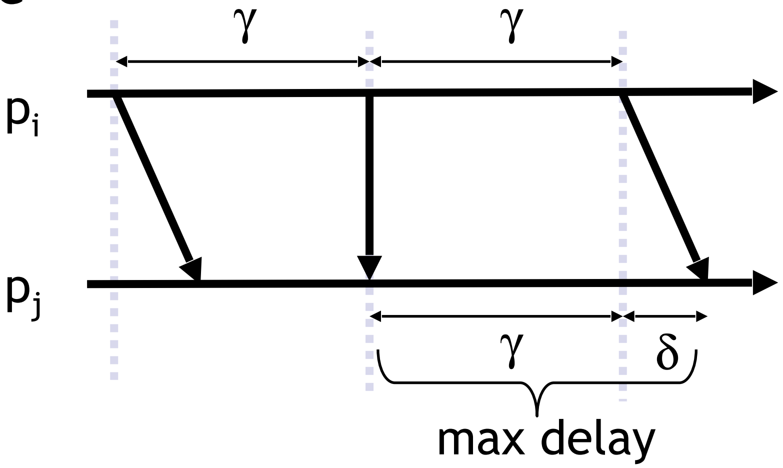
Correctness of P

- ***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

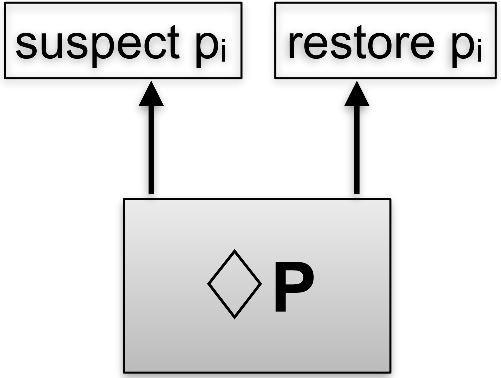


Eventually Prefect Failure Detector ◇P

Interface of ◊P

- **Module:**
  - Name: EventuallyPerfectFailureDetector, **instance** ◊P

- **Events:**
  - Indication: ⟨◊P, suspect | p_i⟩
    - Notifies that process p_i is suspected to have crashed
  - Indication: ⟨◊P, restore | p_i⟩
    - 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 ◇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 ◇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 versus Failure Detection

- Failure detection captures failure behavior
  - Detect failed processes

- Leader election (LE) also captures failure behavior
  - 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 abstraction and algorithms

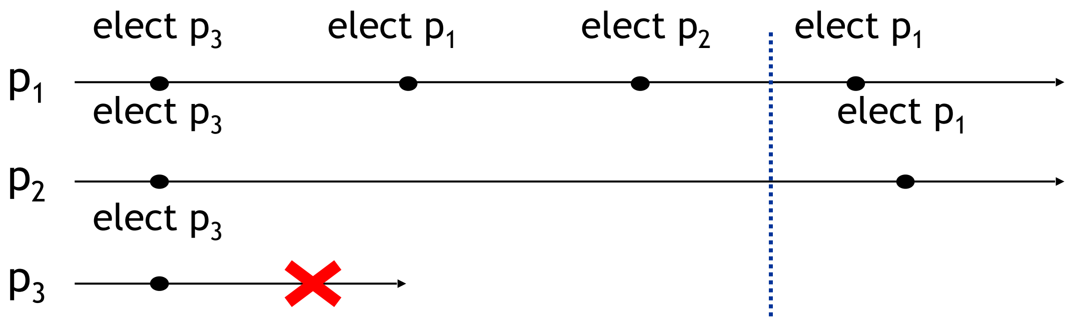
- Leader election (LE) which "matches" P
- Eventual leader election (Ω) which "matches" ◇P

Matching LE and P

- P's properties
  - P always eventually detects failures (strong completeness)
  - P never suspects correct nodes (strong accuracy)
- Completeness of LE
  - Informally: eventually ditch failed leaders
  - Formally: eventually every correct process trusts some correct node
- Accuracy of LE
  - Informally: never ditch a correct leader
  - Formally: No two correct processes trust different correct nodes
    - Is this really accuracy? [d]
    - Yes! Assume two processes trust different correct processes
      - One of them must eventually switch, i.e. leaving a correct node

LE desirable properties

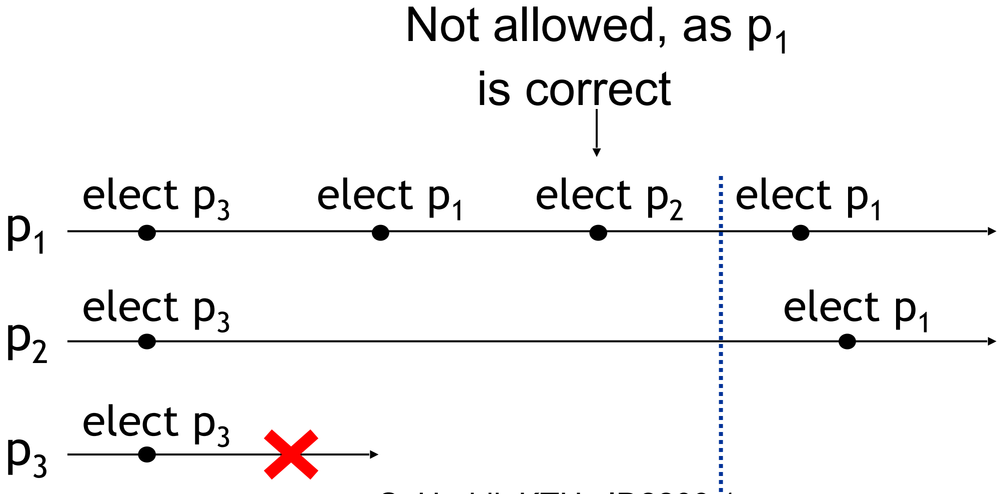
- LE always eventually detects failures
  - Eventually every correct process trusts some correct node
- LE is always accurate
  - No two correct processes trust different correct processes
- But the above two permit the following



- 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**: ⟨leLeader | p_i⟩
    - 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 $rank(p_1)>rank(p_2)>rank(p_3)> \ldots$
- maxrank(S)
  - The process $p \in S$, with the largest rank

Implementing LE

- LeaderElection, **instance** le
- **Uses:**
  - PerfectFailureDetector, **instance P**
- **upon event** $\langle$le, Init$\rangle$ **do**
  - suspected := $\emptyset$
  - leader := $\perp$
- **upon event** $\langle$**P**, Crash | p$\rangle$ **do**
  - suspected := suspected $\cup$ {p}
- **upon** leader $\neq$ maxrank(Π \ suspected) **do**
  - leader := maxrank(Π \ suspected)
  - **trigger** $\langle$le, Leader | leader$\rangle$

Eventual Leader Election Ω

Matching Ω and ◊P

- ◊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):** $\langle\Omega$, Trust | $p_i\rangle$
    - 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 ◊P
  - Each process trusts the process with highest rank among all processes not suspected by ◊P
  - Eventually, exactly one correct process will be trusted by all correct processes

Implementing Ω

- EventualLeaderElection, **instance** Ω
- **Uses:** EventuallyPerfectFailureDetector, **instance ◊P**
- **upon event** ⟨ Ω, Init⟩ **do**
  - suspected := ∅; leader := ⊥
- **upon event** ⟨**◊P**, Suspect | p⟩ **do**
  - suspected := suspected ∪ {p}
- **upon event** ⟨**◊P,** Restore | p⟩ **do**
  - suspected := suspected \ {p}
- **upon** leader ≠ maxrank(Π \ suspected) **do**
  - leader := maxrank(Π \ suspected)
  - **trigger** ⟨ Ω, Trust | leader⟩

Ω for Crash Recovery

- Can we elect a recovered process? [d]
  - 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 ◊P and Ω for crash-stop
  - Piggyback **epoch** with heartbeats
  - Store **epoch**, upon recovery load **epoch** and increment

Reductions

Reductions

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

Reducibility ≼ is a preorder

- ≼ is a preorder
  - Reflexivity. X≼X
    - X can be solved given a solution to X
  - Transitivity. X≼Y and Y≼Z implies X≼Z
    - Since Y≼Z, use implementation of Z to implement Y.
      use that implementation of Y to implement X.
    Hence we implemented X from Z's implementation

- ≼ is not anti-symmetric, thus not a partial order
  - Two different X and Y can be equivalent
    - Distinct problems X and Y can be solved from the other's solution

Shortcut definitions

- We write $X \simeq Y$ if
  - $X \preccurlyeq Y$ and $Y \preccurlyeq X$
  - Problem X is equivalent to Y

- We write $X \prec Y$ if
  - $X \preccurlyeq Y$ and not $X \simeq Y$
  - or equivalently, $X \preccurlyeq Y$ and not $Y \preccurlyeq X$

  - Problem X is strictly weaker than Y, or
  - Problem Y is strictly stronger than X

Example

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

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 ◊P?
    - Sometimes a weaker FD than P will not solve X
      - Proven using reductions

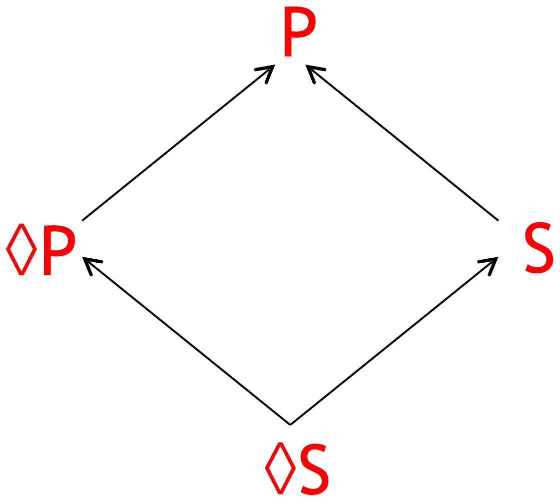
Weakest FD for a problem

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

How are the detectors related

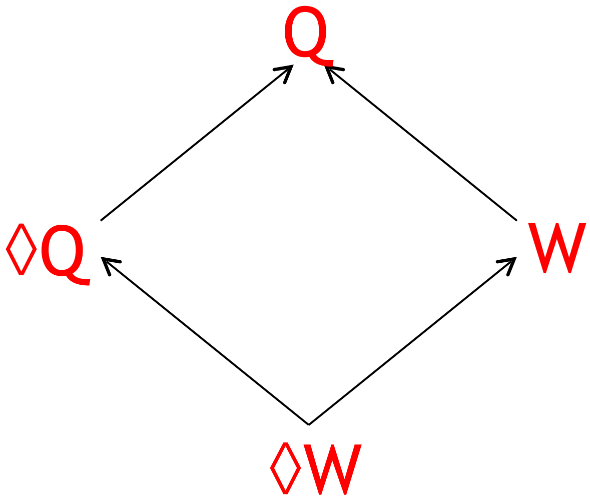
Trivial Reductions

- Strongly complete
  - $\Diamond P \preccurlyeq P$
    - P is always strongly accurate, thus also eventually strongly accurate
  - $\Diamond S \preccurlyeq S$
    - S is always weakly accurate, thus also eventually weakly accurate
  - $S \preccurlyeq P$
    - P is always strongly accurate, thus also always weakly accurate
  - $\Diamond S \preccurlyeq \Diamond P$
    - ◊P is always eventually strongly accurate, thus also always eventually weakly accurate



Trivial Reductions (2)

- Weakly complete
  - ◊Q≼Q
    - Q is always strongly accurate, thus also eventually strongly accurate
  - ◊W≼W
    - W is always weakly accurate, thus also eventually weakly accurate
  - W≼Q
    - Q is always strongly accurate, thus also always weakly accurate
  - ◊W≼◊Q
    - ◊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≼Q, S≼W, ◊P≼◊Q, ◊S≼◊W,
- They're equivalent!
  P≃Q, S≃W, ◊P≃◊Q, ◊S≃◊W

Completeness \ Accuracy	Strong	Weak	Eventual Strong	Eventual Weak
Strong	P	S	◊P	◊S
Weak	Q	W	◊Q	◊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 <**S**,q>
    - Susp := (Susp $\cup$ S) – {q}

also works like a heartbeat

- Every crash is eventually detected by all correct p
  - Can this violate some accuracy properties?

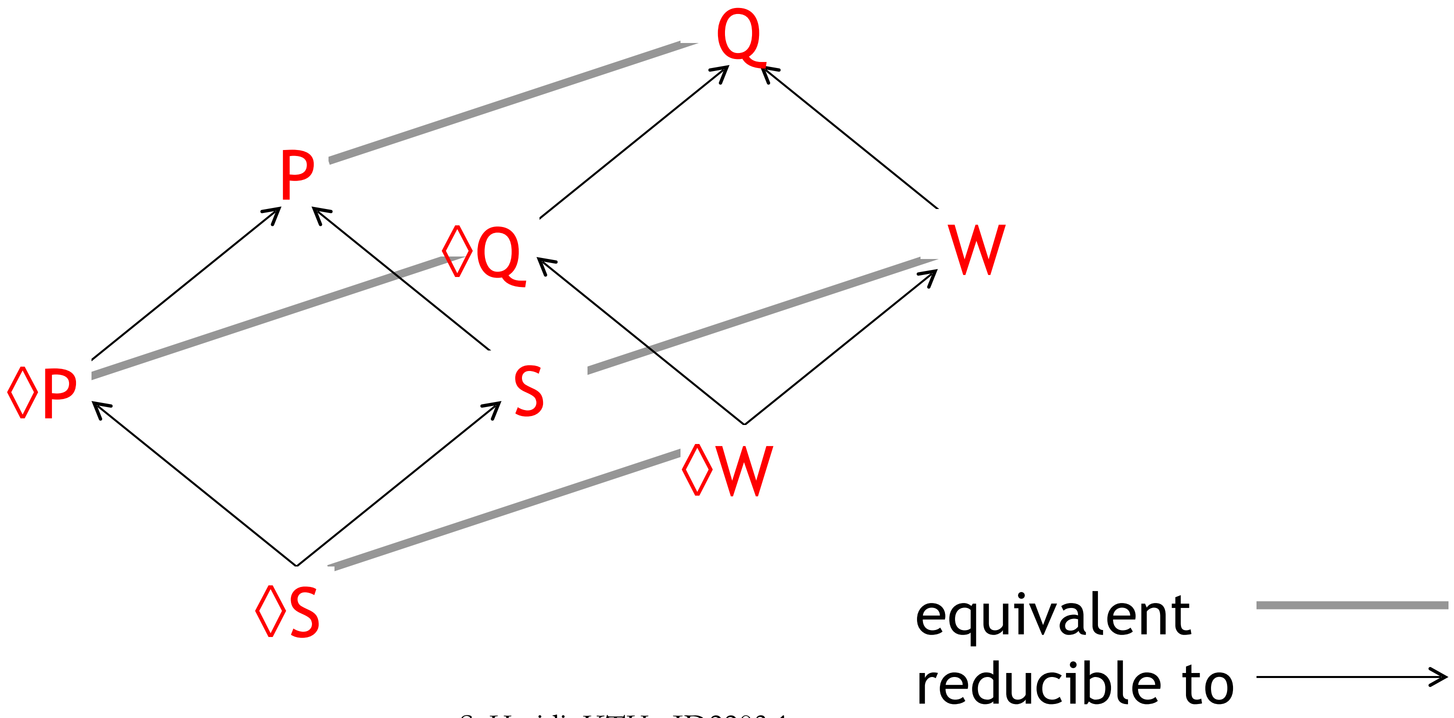
Maintaining Accuracy

- 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



Omega also a FD

- Can we implement ◊S with Ω? [d]
  - I.e. is it true that ◊S≼Ω
  - 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, ◊S≼Ω

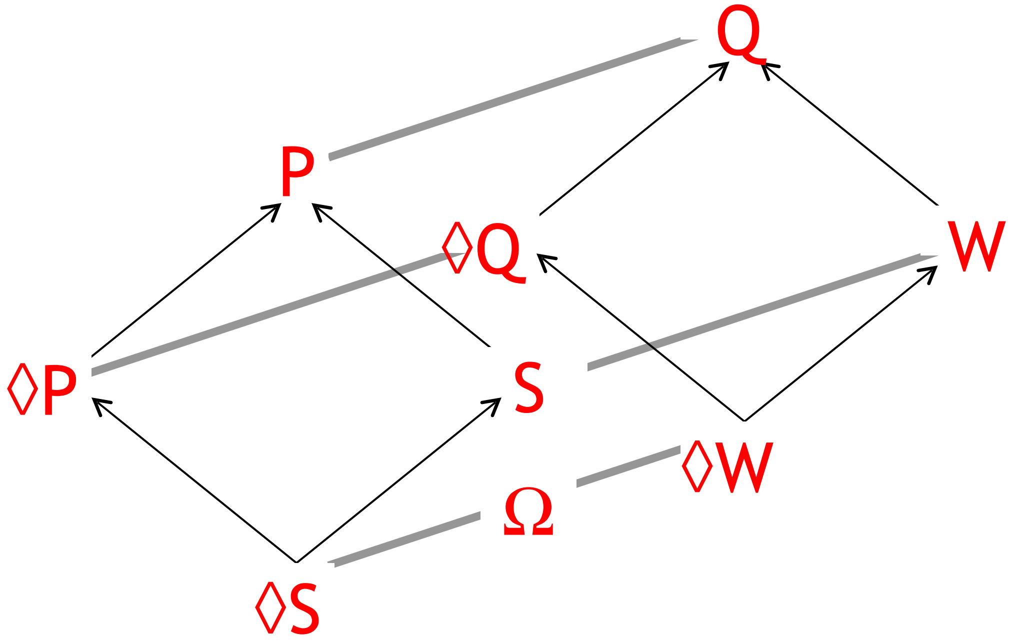
Ω equivalent to ◊S (and ◊W)

- We showed ◊S≼Ω, it turns out we also have Ω≼◊S
  - I.e. Ω≃◊S

- The famous CHT (Chandra, Hadzilocas, Toueg) result
  - If consensus implementable with detector D
    Then Omega can be implemented using D

  - I.e. if Consensus≼D, then Ω≼D
    - Since ◊S can be used to solve consensus, we have Ω≼D
  - Implies ◊W is weakest detector to solve consensus

Relation between FDs (2)



equivalent ▬▬▬
reducible to ⟶

Combining Abstractions

Combining Abstractions

- **Fail-stop** **(synchronous)**
  - Crash-stop process model
  - Perfect links + Perfect failure detector (P)
- **Fail-silent** **(asynchronous)**
  - Crash-stop process model
  - Perfect links
- **Fail-noisy** **(partially synchronous)**
  - Crash-stop process model
  - Perfect links + Eventually Perfect failure detector (◊P)

- **Fail-recovery**
  - Crash-recovery process model
  - Stubborn links + …

The rest of course

- Assume crash-stop system with a perfect failure detector (fail-stop)
  - Give algorithms

- Try to make a weaker assumption
  - Revisit the algorithms