TIME ASSUMPTIONS
&
TIME ABSTRACTIONS

TIME ASSUMPTIONS

- Previously: *asynchronous systems*.
- No timing assumptions.
- No physical clock.
- No bounds on process or communication delays.
- First a parenthesis ...

LOGICAL TIME

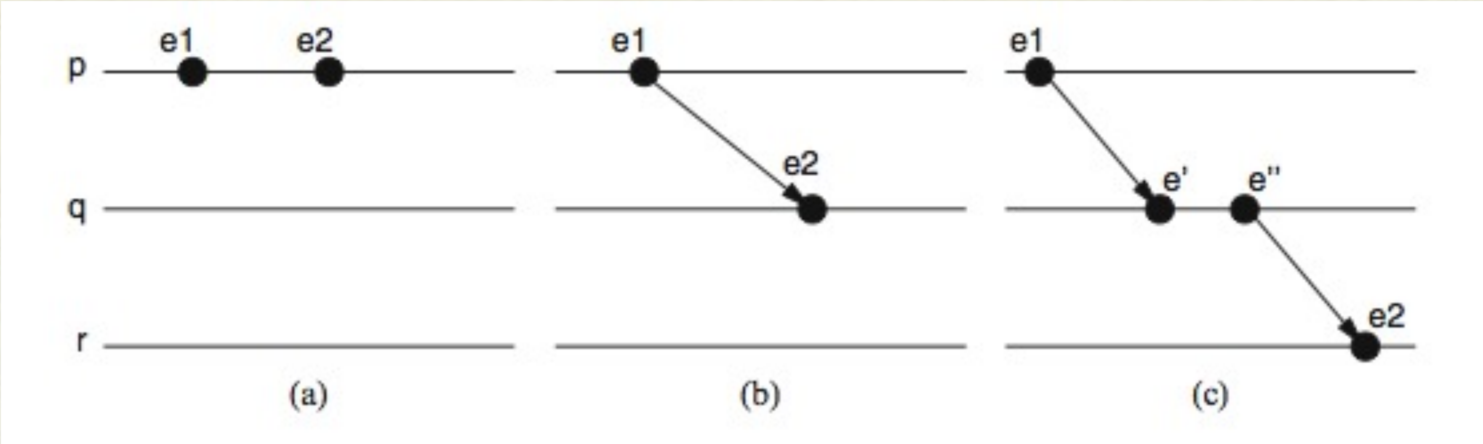
- Asynchronous systems.
- No physical clock!
- Time units: transmission and delivery of messages.

LOGICAL TIME - ALGORITHM

Each process *p* keeps an integer called logical clock *lp* = 0.

1. Whenever an event occurs at process *p*, the logical clock *lp* is incremented by one unit.
2. *p* sends a message, adds timestamp *t(e)*.
3. *p* receives a message *m* with timestamp *tm*, sets *lp* := max{*lp*, *tm*} + 1.

LOGICAL TIME -CAUSALITY



Event *e1* may have *potentially caused* event *e2*, $e1 \rightarrow e2$ if:

(a) *e1* and *e2* occurred at the same process *p* and *e1* occurred before *e2*;

(b) *e1* corresponds to the transmission of a message *m* at a process *p* and *e2* to the reception of *m* at some other process *q*; or

(c) there exists some event e', such that $e1 \rightarrow e'$ and $e' \rightarrow e2$.

SYNCHRONOUS SYSTEM

Synchronous computation: known upper bound on process delays.

AND

Synchronous communication: known upper bound on communication delays.

ALTERNATIVELY ONLY

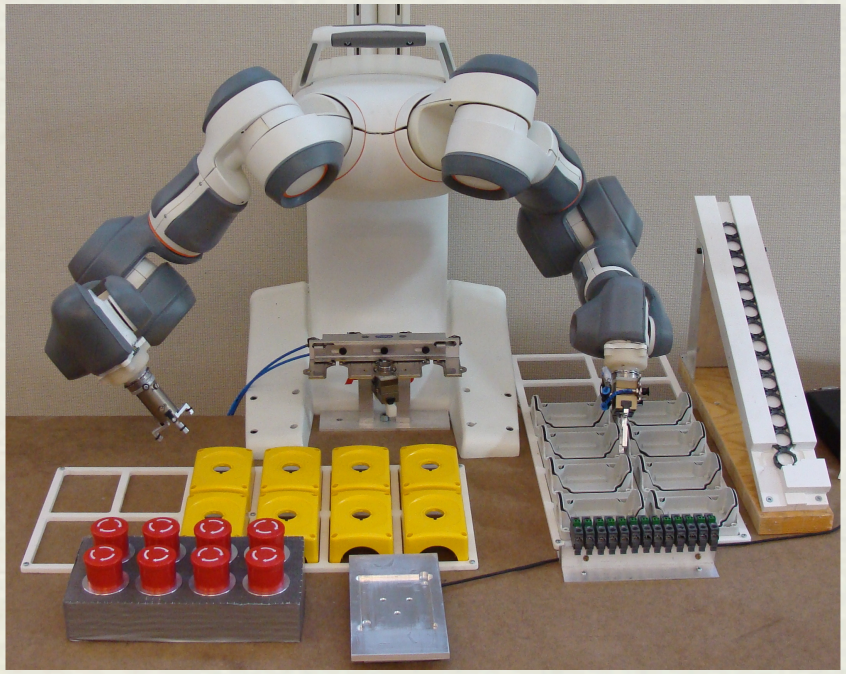
Synchronous physical clocks: local physical clocks + upper bound on deviation from global physical clock.

SYNCHRONOUS SYSTEM - SERVICES

- Timed failure detection - heartbeats.
- Measure of transit delays.
- Coordination based on time. *Lease*.
- Worst-case performance.
- Synchronized clocks. Time stamp events + order (within sync precision).

PROBLEMS

- *Coverage*: When the timing assumptions hold.



Source: screenshot reddit.com

Controlled network load ... ... worst case scenario.

PARTIAL SYNCHRONY

- Periodically overloaded.
- No bound on the period that is asynchronous.
- After some time the timing assumptions hold "forever".
- Eventually they will hold!

ABSTRACTING TIME

- Add the timing assumptions to ...
- ... links and processes? Messy.
- Introduce: *failure detection!*
- Crash: heartbeats.

PERFECT FAILURE DETECTION

- Timeouts: 2 x transmission time + worst-case process time.
- No response? Crash!
- Final judgement.

PERFECT FAILURE DETECTION

Module 2.6: Interface and properties of the perfect failure detector

Module:

Name: PerfectFailureDetector, **instance** $\mathcal{P}$.

Events:

Indication: $\langle \mathcal{P}, Crash \mid p \rangle$: Detects that process p has crashed.

Properties:

PFD1: *Strong completeness:* Eventually, every process that crashes is permanently detected by every correct process.

PFD2: *Strong accuracy:* If a process p is detected [to have crashed] by any process, then p has crashed.

Algorithm 2.5: Exclude on Timeout

Implements:
PerfectFailureDetector, **instance** $\mathcal{P}$.

Uses:
PerfectPointToPointLinks, **instance** *pl*.

upon event $\langle$ $\mathcal{P}$, *Init* $\rangle$ **do**
alive := Π;
detected := $\emptyset$;
starttimer(Δ);

Large enough so that every process can send and deliver a heartbeat to all.

upon event $\langle$ *Timeout* $\rangle$ **do**
forall $p \in \Pi$ **do**
if $(p \notin alive) \wedge (p \notin detected)$ **then**
detected := *detected* $\cup \{p\}$;
trigger $\langle$ $\mathcal{P}$, *Crash* | p $\rangle$;
trigger $\langle$ *pl*, *Send* | p, [HEARTBEATREQUEST] $\rangle$;
alive := $\emptyset$;
starttimer(Δ);

New heartbeats are triggered

upon event $\langle$ *pl*, *Deliver* | q, [HEARTBEATREQUEST] $\rangle$ **do**
trigger $\langle$ *pl*, *Send* | q, [HEARTBEATREPLY] $\rangle$;

upon event $\langle$ *pl*, *Deliver* | p, [HEARTBEATREPLY] $\rangle$ **do**
alive := *alive* $\cup \{p\}$;

LEADER ELECTION

- Detect *living* process -> *Leader*.
- Leader that coordinates the others.
- Only for crash-stop!
- Useful for backup processes.

LEADER ELECTION

Module 2.7: Interface and properties of leader election

Module:

Name: LeaderElection, **instance** *le*.

Events:

Indication: $\langle$ *le*, *Leader* | p $\rangle$: Indicates that process p is elected as leader.

Properties:

LE1: *Eventual detection:* Either there is no correct process, or some correct process is eventually elected as the leader.

LE2: *Accuracy:* If a process is leader, then all previously elected leaders have crashed.

Algorithm 2.6: Monarchical Leader Election

Implements:
LeaderElection, **instance** *le*.

Uses:
PerfectFailureDetector, **instance** $\mathcal{P}$.

upon event $\langle$ *le*, *Init* $\rangle$ **do**
  $suspected := \emptyset$;
  $leader := \perp$;

upon event $\langle$ $\mathcal{P}$, *Crash* $|$ p $\rangle$ **do**
  $suspected := suspected \cup \{p\}$;

upon $leader \neq \text{maxrank}(\Pi \setminus suspected)$ **do**
  $leader := \text{maxrank}(\Pi \setminus suspected)$;
  trigger $\langle$ *le*, *Leader* $|$ *leader* $\rangle$;

#1

Source: http://en.wikipedia.org/wiki/File:King_Carl_XVI_Gustaf_at_National_Day_2009_Cropped.png

#2

Source: http://en.wikipedia.org/wiki/File:Crown_Princess_Victoria.jpg

EVENTUAL FAILURE DETECTION

- Partial synchronous systems.
- False suspicions.
- Change judgement.
- Small timeouts -> increased.
- After increased timeouts -> system synchronous.

Source: http://en.wikipedia.org/wiki/File:Noel-coypel-the-resurrection-of-christ-1700.jpg

EVENTUAL FAILURE DETECTION

Module 2.8: Interface and properties of the eventually perfect failure detector

Module:

Name: EventuallyPerfectFailureDetector, **instance** $\Diamond\mathcal{P}$.

Events:

Indication: $\langle\, \Diamond\mathcal{P}$, *Suspect* $\mid p \,\rangle$: Notifies that process p is suspected to have crashed.

Indication: $\langle\, \Diamond\mathcal{P}$, *Restore* $\mid p \,\rangle$: Notifies that process p is not suspected anymore.

Properties:

EPFD1: *Strong completeness:* Eventually, every process that crashes is permanently suspected by every correct process.

EPFD2: *Eventual strong accuracy:* Eventually, no correct process is suspected by any correct process.

Algorithm 2.7: Increasing Timeout

Implements:
EventuallyPerfectFailureDetector, **instance** $\diamond\mathcal{P}$.

Uses:
PerfectPointToPointLinks, **instance** *pl*.

upon event $\langle\ \diamond\mathcal{P}, \mathit{Init}\ \rangle$ **do**
    $\mathit{alive} := \Pi$;
    $\mathit{suspected} := \emptyset$;
    $\mathit{delay} := \Delta$;
    starttimer(*delay*);

upon event $\langle\ \mathit{Timeout}\ \rangle$ **do**
    if $\mathit{alive} \cap \mathit{suspected} \neq \emptyset$ **then**
        $\mathit{delay} := \mathit{delay} + \Delta$; ← Performance goes down.
    forall $p \in \Pi$ **do**
        if $(p \notin \mathit{alive}) \wedge (p \notin \mathit{suspected})$ **then**
            $\mathit{suspected} := \mathit{suspected} \cup \{p\}$;
            trigger $\langle\ \diamond\mathcal{P}, \mathit{Suspect} \mid p\ \rangle$;
        else if $(p \in \mathit{alive}) \wedge (p \in \mathit{suspected})$ **then**
            $\mathit{suspected} := \mathit{suspected} \setminus \{p\}$;
            trigger $\langle\ \diamond\mathcal{P}, \mathit{Restore} \mid p\ \rangle$;
        trigger $\langle\ \mathit{pl}, \mathit{Send} \mid p, [\text{HEARTBEATREQUEST}]\ \rangle$;
    $\mathit{alive} := \emptyset$;
    starttimer(*delay*);

upon event $\langle\ \mathit{pl}, \mathit{Deliver} \mid q, [\text{HEARTBEATREQUEST}]\ \rangle$ **do**
    trigger $\langle\ \mathit{pl}, \mathit{Send} \mid q, [\text{HEARTBEATREPLY}]\ \rangle$;

upon event $\langle\ \mathit{pl}, \mathit{Deliver} \mid p, [\text{HEARTBEATREPLY}]\ \rangle$ **do**
    $\mathit{alive} := \mathit{alive} \cup \{p\}$;

EVENTUAL LEADER ELECTION

- Living dead!
- Eventually correct processes elect the same leader.
- Crash-recovery.
- Two algorithms.

EVENTUAL LEADER ELECTION

Module 2.9: Interface and properties of the eventual leader detector

Module:

Name: EventualLeaderDetector, **instance** Ω.

Events:

Indication: $\langle\ \Omega, \mathit{Trust} \mid p\ \rangle$: Indicates that process p is trusted to be leader.

Properties:

ELD1: *Eventual accuracy:* There is a time after which every correct process trusts some correct process.

ELD2: *Eventual agreement:* There is a time after which no two correct processes trust different correct processes.

Algorithm 2.8: Monarchical Eventual Leader Detection

Implements:
EventualLeaderDetector, **instance** Ω.

Uses:
EventuallyPerfectFailureDetector, **instance** $\Diamond\mathcal{P}$.

upon event $\langle\, \Omega, \mathit{Init} \,\rangle$ **do**
  $\mathit{suspected} := \emptyset$;
  $\mathit{leader} := \perp$;

upon event $\langle\, \Diamond\mathcal{P}, \mathit{Suspect} \mid p \,\rangle$ **do**
  $\mathit{suspected} := \mathit{suspected} \cup \{p\}$;

upon event $\langle\, \Diamond\mathcal{P}, \mathit{Restore} \mid p \,\rangle$ **do**
  $\mathit{suspected} := \mathit{suspected} \setminus \{p\}$; ← New

upon $\mathit{leader} \neq \mathrm{maxrank}(\Pi \setminus \mathit{suspected})$ **do**
  $\mathit{leader} := \mathrm{maxrank}(\Pi \setminus \mathit{suspected})$;
  trigger $\langle\, \Omega, \mathit{Trust} \mid \mathit{leader} \,\rangle$;

Algorithm 2.9: Elect Lower Epoch

Implements:
EventualLeaderDetector, **instance** Ω.

Uses:
FairLossPointToPointLinks, **instance** *fll*. ← Note!

upon event ⟨ Ω, *Init* ⟩ **do**
epoch := 0;
store(*epoch*);
candidates := ∅;
trigger ⟨ Ω, *Recovery* ⟩; // recovery procedure completes the initialization

upon event ⟨ Ω, *Recovery* ⟩ **do**
leader := maxrank(Π);
trigger ⟨ Ω, *Trust* | *leader* ⟩;
delay := Δ;
retrieve(*epoch*);
epoch := *epoch* + 1;
store(*epoch*);
forall $p \in \Pi$ **do**
trigger ⟨ *fll*, *Send* | p, [HEARTBEAT, *epoch*] ⟩;
candidates := ∅;
starttimer(*delay*);

upon event ⟨ *Timeout* ⟩ **do**
newleader := *select*(*candidates*); ← Select the one with highest rank, among those with lowest epoch.
if *newleader* ≠ *leader* **then**
delay := *delay* + Δ; ← Longer time
leader := *newleader*;
trigger ⟨ Ω, *Trust* | *leader* ⟩;
forall $p \in \Pi$ **do**
trigger ⟨ *fll*, *Send* | p, [HEARTBEAT, *epoch*] ⟩;
candidates := ∅;
starttimer(*delay*);

upon event ⟨ *fll*, *Deliver* | q, [HEARTBEAT, *ep*] ⟩ **do**
if exists $(s, e) \in$ *candidates* such that $s = q \wedge e < ep$ **then**
candidates := *candidates* \ {(q, e)};
candidates := *candidates* ∪ (q, ep);

BYZANTINE LEADER ELECTION

- Trust, but verify!
- Complain if wrong actions or too slow.
- Progressively more time to prove yourself.

BYZANTINE LEADER ELECTION

Module 2.10: Interface and properties of the Byzantine eventual leader detector

Module:

Name: ByzantineLeaderDetector, **instance** *bld*.

Events:

Can be byzantine!

Indication: ⟨ *bld*, *Trust* | p ⟩: Indicates that process p is trusted to be leader.

Request: ⟨ *bld*, *Complain* | p ⟩: Receives a complaint about process p.

Triggered by higher level algorithm

Properties:

BLD1: *Eventual succession:* If more than f correct processes that trust some process p complain about p, then every correct process eventually trusts a different process than p.

BLD2: *Putsch resistance:* A correct process does not trust a new leader unless at least one correct process has complained against the previous leader.

BLD3: *Eventual agreement:* There is a time after which no two correct processes trust different processes.

BYZANTINE LEADER ELECTION

- Max f Byzantine processes.
- $N > 3f$.
- $leader(r) = p$ when $rank(p) = r \bmod N$, $r \bmod N \neq 0$. Otherwise q, such that $rank(q) = N$.
- More than $2f$ complaints needed!
- When more than f complaints for a round, processes than haven't complained, do it!

Algorithm 2.10: Rotating Byzantine Leader Detection

Implements:
    ByzantineLeaderDetector, **instance** *bld*.

Uses:
    AuthPerfectPointToPointLinks, **instance** *al*.

upon event ⟨ *bld*, *Init* ⟩ **do**
    round := 1;
    complainlist := $[\perp]^N$;
    complained := FALSE;
    trigger ⟨ *bld*, *Trust* | *leader*(*round*) ⟩;

upon event ⟨ *bld*, *Complain* | p ⟩ **such that** p = *leader*(*round*) **and**
        complained = FALSE **do**
    complained := TRUE;
    forall $q \in \Pi$ **do**
        trigger ⟨ *al*, *Send* | q, [COMPLAINT, *round*] ⟩;

upon event ⟨ *al*, *Deliver* | p, [COMPLAINT, r] ⟩ **such that** r = *round* **and**
        complainlist[p] = $\perp$ **do**
    complainlist[p] := COMPLAINT;
    if #(*complainlist*) $> f \wedge$ *complained* = FALSE **then**
        complained := TRUE;
        forall $q \in \Pi$ **do**
            trigger ⟨ *al*, *Send* | q, [COMPLAINT, *round*] ⟩;
    else if #(*complainlist*) $> 2f$ **then**
        round := *round* + 1;
        complainlist := $[\perp]^N$;
        complained := FALSE;
        trigger ⟨ *bld*, *Trust* | *leader*(*round*) ⟩;