Consensus when failstop doesn't hold

FLP shows that can't solve consensus in an asynchronous system with no other facility.

It *can* be solved with a perfect failure detector.

- If *p* suspects *q* then *q* has crashed.
- If *q* crashes, then *p* eventually suspects *q*.

Can consensus be solved with something weaker than a perfect failure detector?

Failure Detectors

A *failure detector* is a routine (an *oracle*) that gives information about failures of processes.

F: $T \rightarrow 2^P$ is the set of crashed processes.

F is monotonic: $(p \in F(t)) \Rightarrow (p \in F(t' > t))$.

crashed(F) are the processes that crash at some time

correct(F) = P – crashed(F)

H: $P \times T \rightarrow 2^P$ where $H(p, t)$ is the set of processes that p suspects as being crashed at time t.

We read $q \in H(p, t)$ as "p suspects q at time t".

A failure detector D maps F to a set of H.

Failure Detectors: Completeness

How good is the failure detector in detecting a crashed process?

strong: $\forall F, \forall H \in D(F): \exists\, t \in T$:

$\quad \forall p \in crashed(F), \forall q \in correct(F): \forall t' \geq t: p \in H(q, t')$

(every process that never crashes eventually suspects every process that does crash)

weak: $\forall F, \forall H \in D(F): \exists\, t \in T$:

$\quad \forall p \in crashed(F), \exists q \in correct(F): \forall t' \geq t: p \in H(q, t')$

(there is a process that never crashes and that eventually suspects every process that does crash)

Failure Detectors: Accuracy

How good is the failure detector in not detecting a correct process?

strong: $\forall F, \forall H \in D(F): \forall\, t \in T: \forall p, q \in P - F(t):$
$$p \notin H(q, t)$$

(No process ever suspects an uncrashed process)

weak: $\forall F, \forall H \in D(F): \exists p \in \text{correct}(F): \forall\, t \in T:$
$$\forall q \in P - F(t): p \notin H(q, t)$$

(There is a process that never crashes and that is never suspected)

Failure Detectors: Eventual Accuracy

eventually strong:

$\forall F, \forall H \in D(F)$: $\exists\, t \in T$: $\forall t' \geq t$:

$\forall p,q \in correct(F)$: $p \notin H(q, t')$

eventually weak:

$\forall F, \forall H \in D(F)$: $\exists\, t \in T$: $\exists p \in \text{correct}(F)$:

$\forall\ t' \geq t$: $\forall q \in correct(F)$: $p \notin H(q, t')$

Failure Detectors: Summary

	strong completeness	*weak completeness*
strong accuracy	P	Q
weak accuracy	S	W
◊-*strong accuracy*	◊P	◊Q
◊-*weak accuracy*	◊S	◊W

Completeness results

Given weak completeness, one can implement strong completeness.

- Let $suspects_p$ be the output of $H(p, t)$.
- Each process p implements $output_p$ which is initially $\{\}$. This is the suspicion set from which p operates.
- Periodically, p sends $\langle p, suspects_p \rangle$ to all.
- When p receives $\langle q, suspects_q \rangle$, p sets
$$output_p = output_p \cup suspects_q - \{q\}$$

... so, if some correct process permanently suspects a crashed process x, then all correct processes will eventually permanently suspect x.

Failure Detectors: Summary

	strong completeness	~~*weak completeness*~~
strong accuracy	P	~~Q~~
weak accuracy	S	~~W~~
$\Diamond$-*strong accuracy*	$\Diamond$P	~~$\Diamond$Q~~
$\Diamond$-*weak accuracy*	$\Diamond$S	~~$\Diamond$W~~

Consensus with Failure Detectors

- propose(v) propose a value v for consensus
- decide(v) decide on a consensus value v

termination: each correct process eventually decides on a value.

uniform integrity: each process decides at most once.

agreement: no two correct processes decide differently.

uniform validity: if a process decides on v, then some process proposed v.

uniform agreement: no two processes decide differently.

Reliable Broadcast

In a synchronous model, consensus ≡ reliable broadcast.

In an asynchronous model, they're different: no failure detector is needed at all!

R-broadcast(m) { send m to all; }

when receive m for the first time {
    if (sender(m) $\neq$ me) then send m to all;
    R-deliver(m);
    }

S Consensus

Processs will send and forward each other's initial values.

$V_p[q]$ will be p's knowledge of q's initial value v_q:

$V_p[q] \in \{v_q, \perp\}$

For each process q, p will wait to either receive an expected message or detect q's failure.

There is some (unknown) correct process c that is never suspected

… so each process will eventually receive c's forwarded values.

Construct protocol so that if $V_c[q] \neq \perp$ then, for all noncrashed processes p, $V_p[q] \neq \perp$.

That is, if any noncrashed p has $V_p[q] = \perp$, then c has $V_c[q] = \perp$.

The processes then exchange V and choose the first non-$\perp$ value.

Since all get V_c they will agree on the value.

S Consensus

- Phase 1 consists of $n - 1$ rounds
  - In each round a process forwards values it learned in the last round and receives these values from other processes.
  - The rounds aren't necessarily synchronized because the failure detector can have false suspicions.
- Phase 2 processes exchange vector of values they've received during phase 1.
- In phase 3 they decide.

S Consensus (protocol)

variables for process p:

V_p: array of learned proposed values ($\perp$ if none).

Δ_p: array of proposed values learned in this round (for phase 1).

$msgs_p[r]$: messages p received in round r of phase 1.

$lastmsg_p$: messages p received in phase 2.

propose(v_p):

$V_p = \langle \perp, \perp, \ldots, \perp \rangle$;

$V_p[p] = v_p$;

$\Delta_p = V_p$;

S Consensus (protocol, continued)

Phase 1 *// repeatedly forward values newly learned*

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

  send(r_p, Δ_p, p) to all;

  for each q: receive (r_p, Δ_q, q) or suspect q;

  $msgs_p[r_p]$ = messages received with round r_p;

  $\Delta_p = \langle \perp, \perp, \ldots, \perp \rangle$;

  for $k = 1$ to n

    if ($V_p[k] == \perp \wedge \exists\ (r_p, \Delta_q, q) \in msgs_p[r_p]$: $\Delta_q[k] \neq \perp$)

      $V_p[k] = \Delta_p[k] = \Delta_q[k]$;

S Consensus (protocol, continued)

Phase 2 // *agree on vectors*

send V_p to all;

for each q: received V_q or suspect q;

$lastmsgs_p$ = messages received in phase 2;

for $k = 1$ to n

    if ($\exists\ V_q$ in $lastmsgs_p$: $V_q[k] == \perp$) $V_p[k] = \perp$;

Phase 3

Decide on the first non-$\perp$ component of V_p;

S-consensus (proof, $n > t$)

Lemma 1: $\forall p, q$: $V_p[q] \in \{v_q, \perp\}$

Straightforward observation based on how values are assigned.

Lemma 2: Each correct process eventually reaches phase 3.

From strong completeness.

P_1 = the set of processes that complete phase 1

P_2 = the set of processes that complete phase 2

c = a correct process that is never suspected

S-consensus (proof, continued)

Lemma 3: In each round r of phase 1, $\forall\, p \in P_1$: $msgs_p[r]$ contains (r, Δ_c, c).

Because c is correct and never suspected.

Define $V \leq V' \equiv \forall\, q$: $V[q] \in \{V'[q], \perp\}$

Lemma 4: $\forall\, p \in P_1$: $V_c \leq V_p$ at end of phase 1.

Let c first set $V_c[q]$ to v_a in round r.

If $r < n - 1$ then c will set $\Delta_c[q]$ to v_q in r

… by Lemma 3 p will get v_q in $r + 1$.

If $r = n - 1$ then each other process has already forwarded v_q (each process forwards a value no more than once).

S-consensus (proof, continued)

Lemma 5: $\forall\, p \in P_2$: $V_c = V_p$ at end of phase 2.

- Consider $V_p[q]$ and $V_c[q]$.
- If $V_p[q]=v_q$ then from Lemma 4, at the end of phase 1, each process p has $V_p[q]=v_q$. Hence, $V_p[q]=V_c[q]=v_q$ at the end of phase 2.
- If $V_c[q]=\perp$, then since c is never suspected as being faulty, by the end of Phase 2 each process will have received V_c and set $V_p[q]=\perp$.

So, uniform agreement holds.

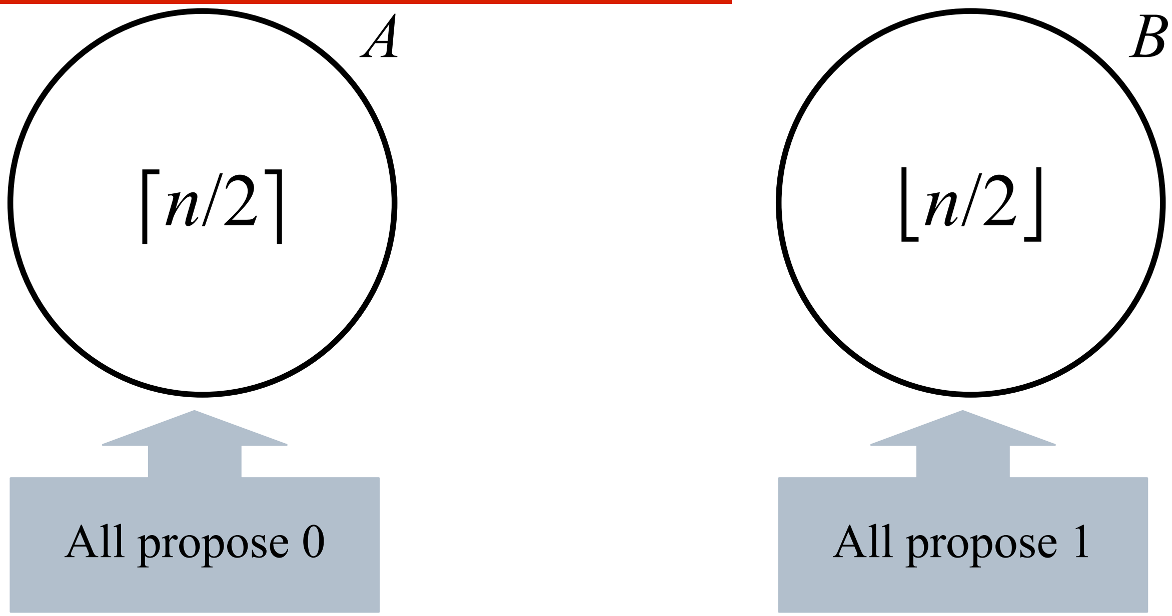
◇S Consensus

There is an unbounded period of time during which all processes may be suspected.

Use a *rotating coordinator* scheme:

- Each process *p* will repeatedly try to establish consensus.
- If *p* is not suspected by anyone for long enough, then it will succeed.
- ◇S guarantees that eventually there will be some process that is not suspected by anyone.

$\Diamond$S Consensus requires $n > 2t$



Run 1: have all in A crash before proposing. By *termination* and *uniform validity*, those in B decide 1.

Run 2: have all in B crash before proposing. By *termination* and *uniform validity*, those in A decide 0.

Run 3: have all in B suspect A and those in A suspect B until *agreement* is violated.

◇S Consensus

- Protocol consists of an unbounded number of rounds
- Each round has a well-known coordinator
- The coordinator obtains values from a quorum, takes the latest value, and writes that value to a quorum.
  - Consensus is reached when a quorum contains the same value.
  - Coordinator knows consensus reached when gets acknowledgements from a quorum.
  - When coordinator knows, it uses reliable broadcast to spread the good news.

$\Diamond$S Consensus (protocol)

Variables

estimate$_p$: p's estimate of the decision value

state$_p$: {**undecided**, **decided**}

r_p: p's current round

ts_p: the last round in which p updated *estimate*$_p$.

c_p: coordinator for round r_p: (r_p **mod** n) + 1.

Assume that the processes are {1, 2, 3, ... n}

$\diamond$S Consensus (protocol, continued)

```
propose(v_p) {
    estimate_p = v_p;
    state_p = undecided;
    r_p = ts_p = 0;
    while (state_p == undecided) {
        r_p = r_p + 1;
        c_p = (r_p mod n) + 1;
        // ----------- phase 1 -----------
        send (p, r_p, estimate_p ts_p) to c_p;
        // ----------- phase 2 -----------
        if (p == c_p)
            receive (q, r_p, estimate_q ts_q) into msgs_p[r_p] until have received from a majority;
            t = largest ts_q in msgs_p[r_p];
            estimate_p = one of the estimate_q in msgs_p[r_p] with ts_q = t;
            send (p, r_p, estimate_p) to all;
```

◇S Consensus (protocol, continued)

```
        // ----------- phase 3 -----------
        wait until suspect c_p or receive (c_p, r_cp, estimate_cp);
        if (received)
                estimate_p = estimate_cp;
                ts_p = r_p;
                send (p, r_p, ack) to c_p;
        else send (p, r_p, nack) to c_p;
        // ----------- phase 4 -----------
        if (p == c_p)
                wait until receive (q, r_p, ack/nack) from majority;
                if (all ack) R-broadcast (p, r_p, estimate_p, decide);
    }

when R-deliver (q, r_q, estimate_q, decide) {
    if (state_p == undecided)
        decide(estimate_q);
        state_p = decided;
    }
```

Asynchronous consensus…

- ◇W is the weakest failure detector that solves consensus.
- It's equivalent to ◇S.
- It's also equivalent to Ω:
  - Each process *p*'s failure detector outputs $trust_p$: a single process *p* believes is correct.
  - Ω ensures that eventually all correct processes always trust the same correct process.