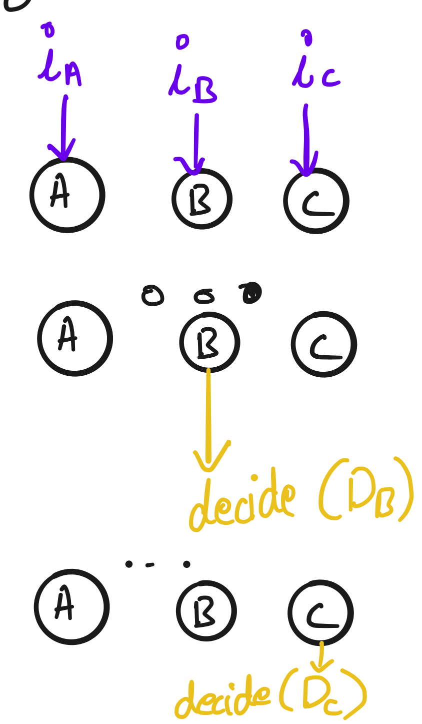
FLP + Partial Synchrony

Where we are

RSMs
- → Consensus
  - → Hard because FLP Impossibility
    - → Circumvent by assuming Partial Synch → Raft, → Paxos
    - → Circumvent with Randomness → Rabia

Today : FLP, Partial Synchrony.

- Defining Consensus



Agreement

→ Process p_0 decides d_0 & process p_1 decides d_1

$\Rightarrow d_0 = d_1$

Validity

- Process p decides d

$\Rightarrow$ Some process q had input $= d$

...   termination

- Eventually some process decides.

Impossibility Result:

No fault-tolerant, deterministic algorithm in the asynchronous model.

Aside: From an award speech in 2001
- Lynch wanted to prove an asynch consensus protocol (designed by Lampson) correct in 82.
- Kept running into trouble, started working with Fischer & Paterson. No luck
- Led to argument in this paper.

Going to walk through proof a bit

Why? - Ideas are interesting & at the core of how we think of many distributed algorithms
- Origin of the model we have been using.

How? Assume a simple setting

- Binary Consensus

- No message loss (only delays)

- At most 1 failure

System Model

○ Process – state
- Recv message ⟶ update state ⟶ send ∅ or more messages
- On Timeout ⟶ update state

Configuration C $\boxed{Ⓐ\ Ⓑ\ Ⓒ\ \boxed{\text{Pending}}}$ ← State of all processes + set of pending messages

Event e : recv (p, m)

p ← Process

m ← or ⊥ to represent timeout

$C \xrightarrow{e\,=\,recv(p,\,m)} C'$

Can differ in state of process p & set of pending messages

} Enabled e?

Schedule $\sigma = e_1; e_2; e_3; \dots; e_n$ } Valid σ?

$C \longrightarrow \ \longrightarrow \ \longrightarrow \dots \longrightarrow C'$

$$\underbrace{e_1 \quad e_2 \quad e_3 \quad e_n}_{\xrightarrow{\quad \sigma \quad}}$$

"Partially" Correctness (or safe correctness)
- Agreement
- Validity

"Totally" Correct = partial correct + Liveness
Termination

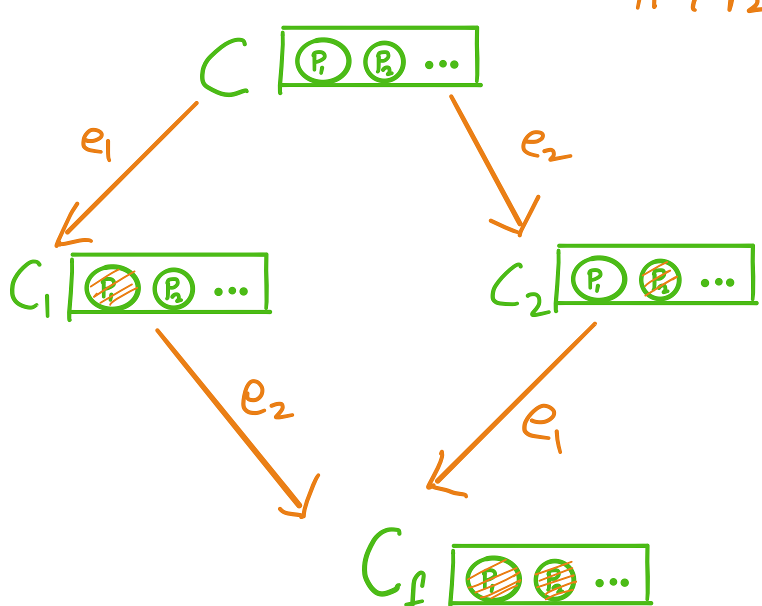
No totally correct fault-tolerant, deterministic algorithm in the asynchronous model.

Some tools

(1) C e_1, e_2 | e_1, e_2 enabled in C

$e_1 = recv(P_1, m)$ $e_2 = recv(P_2, m')$

$P_1 \neq P_2$



Diamond Lemma

② Valence: Assume ∃ partially correct consensus protocol P implemented by system

C $\emptyset$-valent iff $\forall$ valid schedules σ from C

$C \xrightarrow[\sigma]{}$
- $\emptyset$-decided (some process p decided $\emptyset$)

OR
- Undecided

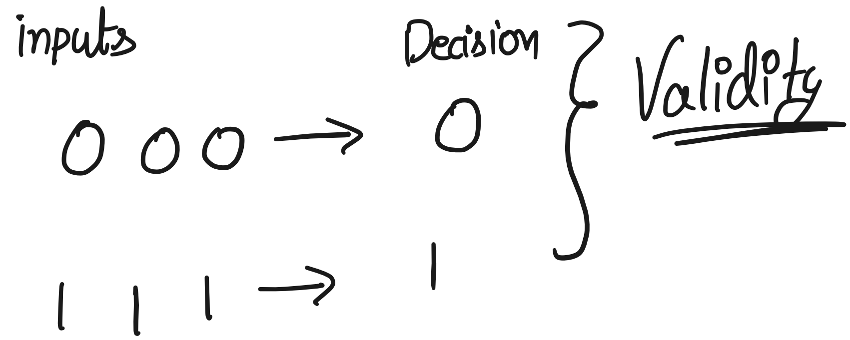
Similarly 1-valent.

C bivalent → C is not $\emptyset$-valent &
C is not ~~1-valent~~ &
some configuration reachable from C by an enabled schedule decides

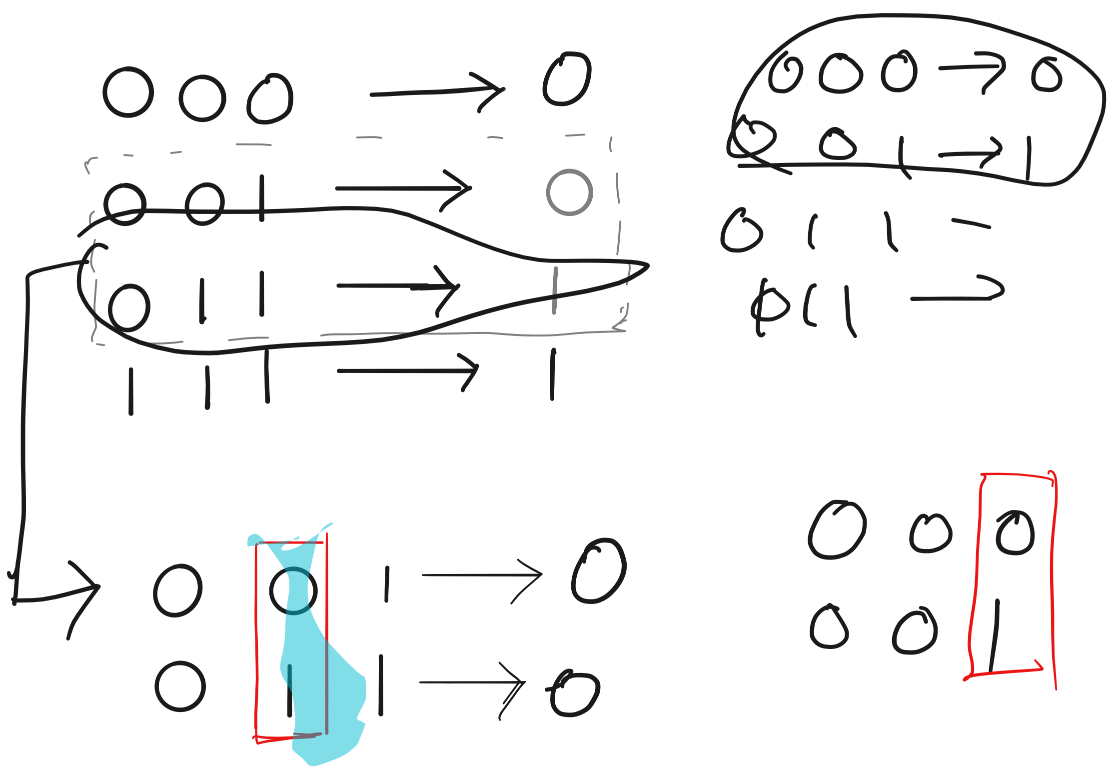
FLP proof → Proof by contradiction

Assume ∃ totally correct protocol P that is 1-fault tolerant.

① ∃ 0- and 1-valent initial configurations



② ∃ Bivalent initial configuration



③ For any enabled event e in bivalent configuration C can find valid σ (where at most 1 process is silent) s.t

$$C \xrightarrow{\sigma} \xrightarrow{e} C'$$

C' is Bivalent.

④ Start from initial bivalent configuration

C_i

Why not pick conven-ient e?

$C_i \rightarrow$ $\boxed{\text{pick enabled } e \text{ at random}}$ $\rightarrow$ Apply ③ find σ

$$C_i \xrightarrow{\sigma} \xrightarrow{e} C_1 \xrightarrow{\sigma_1} \xrightarrow{e} C_2 \ldots$$

Schedule of unbounded length going from one BIVALENT CONFIG to another

$\Rightarrow$ Unbounded schedule where no process decides

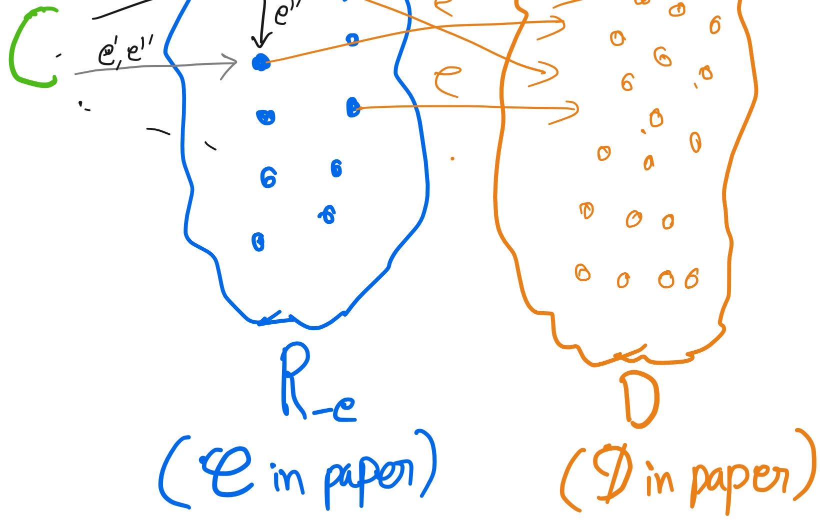
$\Rightarrow$ No termination.

Look at ③

Given bivalent configuration C, event e want to show $\exists \sigma$ s.t. $C \xrightarrow{\sigma} \xrightarrow{e}$ Bivalent

- If $C \xrightarrow{e}$ bivalent : Done $\sigma = [\,]$

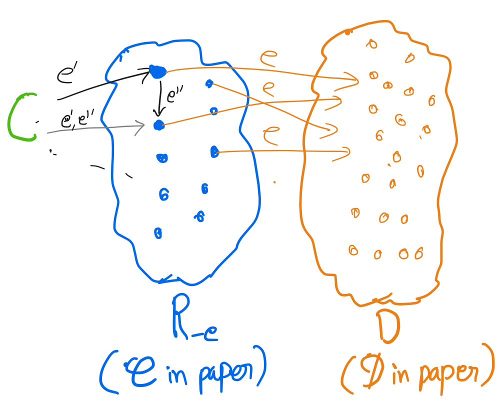
- Otherwise,



Observation: Claim is equivalent to

claim that D contains bivalent configuration.

Proof by contradiction:



(i) $\exists$ reachable $\emptyset$ & 1-valent configuration from C

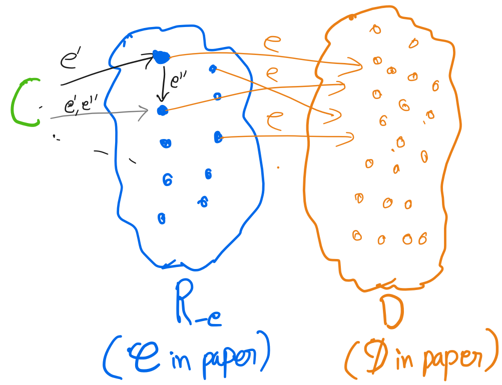
[Reachable: $\exists$ valid σ from C to config]

- $\mathbb{P}$ totally correct $\Rightarrow$ terminates
- C bivalent $\Rightarrow \exists\ \sigma_0, \sigma_1$

$C \xrightarrow[\sigma_0]{} \text{decide } \emptyset$

$\xrightarrow[\sigma_1]{} \text{decide } 1$

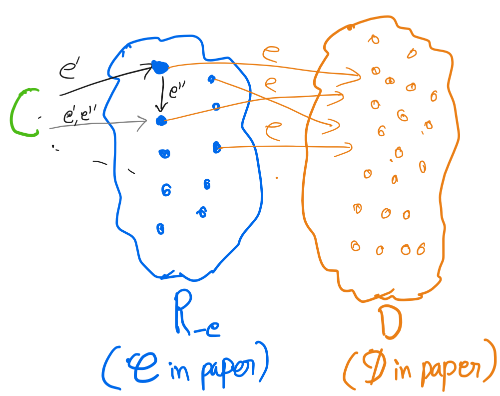
(ii) $\exists$ 0 & 1-valent configurations in D



Let C_0 be 0 valent config reachable from C

- If $C_0 \in R_{-e}$, then

  $e(C_0) \in D$ is a 0-valent config.



- If $C_0 \notin R_{-e}$, the

  $e \in$ schedule σ_0 from $C \xrightarrow[\sigma_0]{} C_0$

$\Rightarrow \exists\ C_0'$ s.t

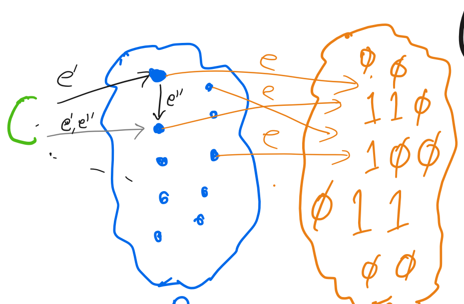
$$C \xrightarrow[\sigma_0\,[\text{before } e]]{} C' \xrightarrow{e} C_0' \xrightarrow[\sigma_0\,(\text{after})]{} C_0$$

& $C_0' \in D$

But by assumption no bivalent conf in $D \Rightarrow C_0'$ is 0-valent

By symmetry for 1-valent

(iii) Hand-wavy:



$\exists\ C_0, C_1 \in R_{-e}$ s.t.

(a) $\exists\ e'$ $C_0 \xrightarrow{e'} C_1$

R_{-e} ($\mathscr{e}$ in paper) D ($\mathscr{D}$ in paper)

ⓑ $e(C_0)$ – $\emptyset$-valent &
$e(C_1)$ – 1-valent

(iv) $e = \text{recv}(p, m)$

$e' = \text{recv}(p', m')$

If $p \neq p'$

$$\begin{tikzcd}
 & C_0 \arrow[dl, "e"'] \arrow[dr, "e'"] & \\
\emptyset\text{-valent} \arrow[dr, "e'"'] & & C_1 \arrow[dl, "e"] \\
 & 1\text{-valent} &
\end{tikzcd}$$

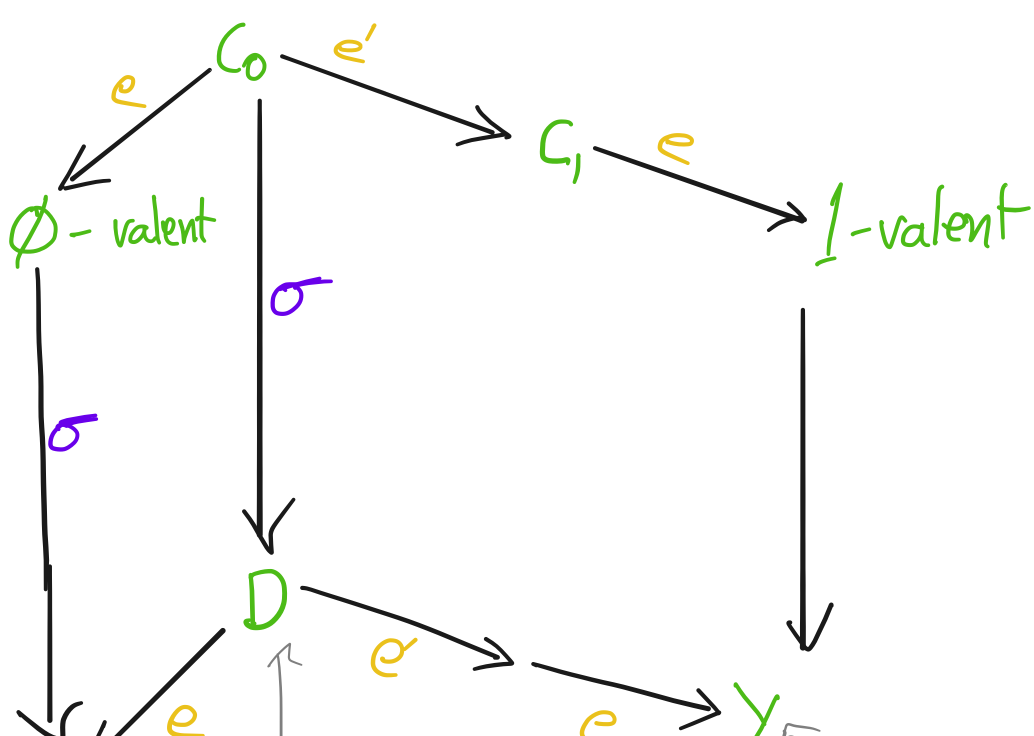
$\times$

If $p = p'$

$\exists \sigma$ where $C_0 \xrightarrow[\sigma]{}$ some process decides;

σ finite & p takes no steps in σ

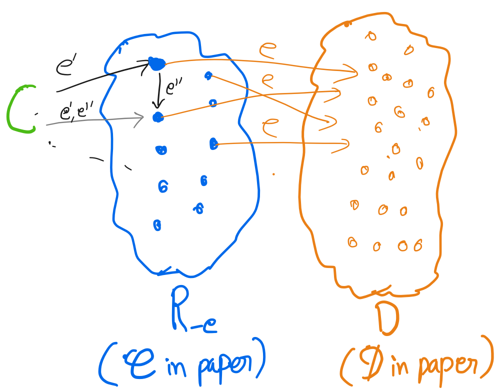
[Because F.T.]



must be ϕ ?

Reached a decision.
Agreement $\Rightarrow$ must hold

must be 1 ?



Thus D must contain a bivalent config.

Given bivalent configuration C, event e
$\exists\, \sigma$ s.t. $C \xrightarrow[\sigma]{} \xrightarrow[e]{}$ Bivalent

$\rightarrow$ For any consensus protocol that is 1-FT, $\exists$ unbounded schedule w/o decision

$\Rightarrow$ No totally correct protocol.

So where does that leave us

FT-consensus	(Real world??)	No consensus
synchronous		asynch

(1-fail stop)

(Fail-stop,

Omission,

Byz.,

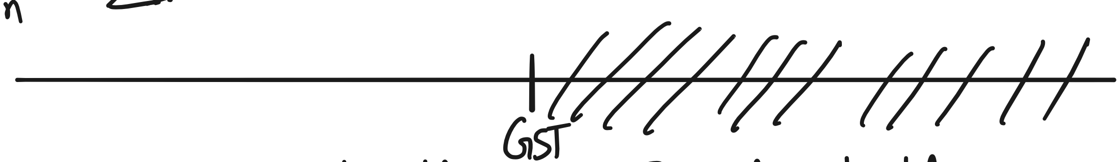
Auth. Byz.)

Partial Synchrony

- Separates out comm. & processing

Talks about two variants

(a) Known Δ



Bounds don't hold   Bounds hold

(b) Unknown Δ, but it always holds.

Results are the same, but most people mean (a)

Approach

- Design protocols for a model where

- Communication & processing are synchronous

- Messages can be lost

... But ∃ time (GST) after which message from correct process p to correct process q is received

- Show they are totally correct

→ Design

- Rounds where processes send & receive messages

- Decisions require quorum of CORRECT processes to be involved

Why termination

- Each round takes bounded time (synchrony assumption)

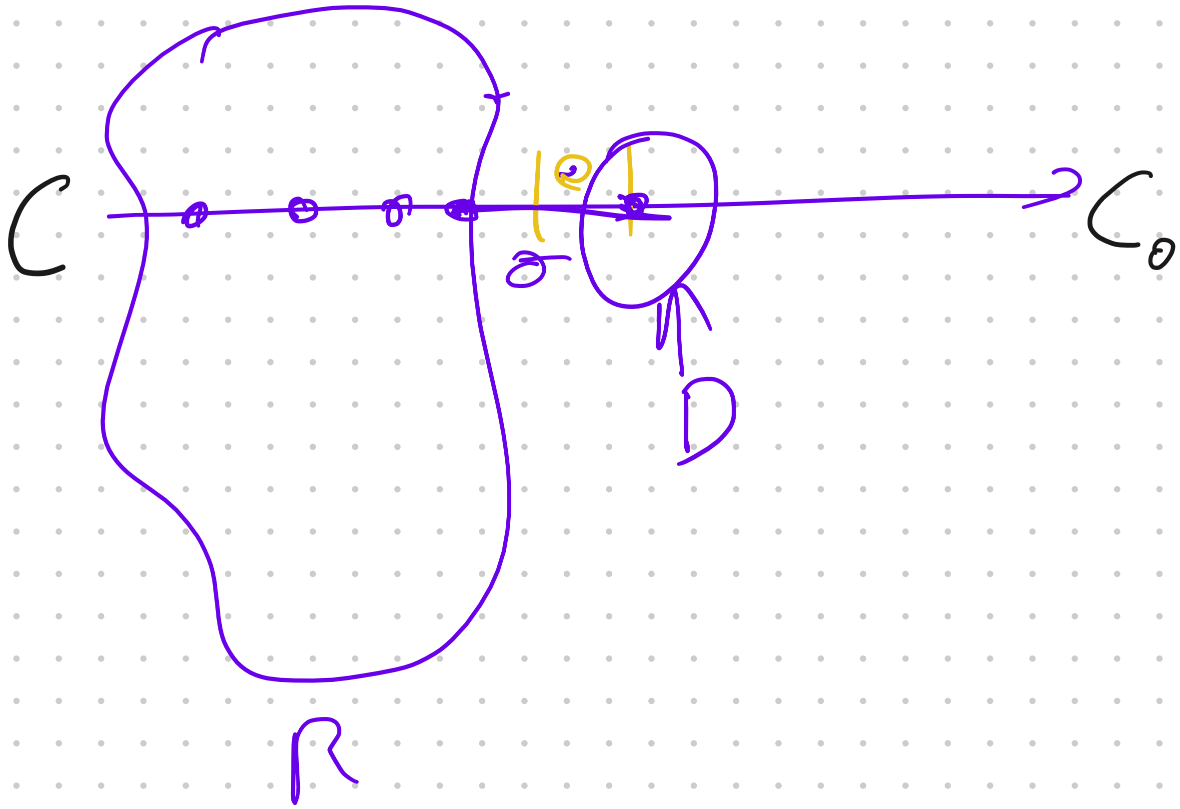
- After GST, all correct processes can communicate w/ each other

Assume large enough set of processes to ensure t-failures still allow decisions

→ Bounded rounds to reach consensus

→ bounded time to reach consensus.

- Show that synchronous rounds can be emulated in partially synchronous model
  - ↳ When Δ is known, use that to decide how long to wait for communication
  - → When Δ is unknown, keep increasing until it hits the right value.



C
C_0
R
D