Distributed Systems

Lecture 2

- Ordering Events
- Safety
- Liveness

Announcements

- Final project ideas
  ↳ Posted on the class website
  → Will ask you to submit a proposal in ~ 1 month

  BUT - Posting this early so You can investigate options before choosing

  Fine to start working now

- Campuswire — JOIN

Last Week

- Asynchronous model

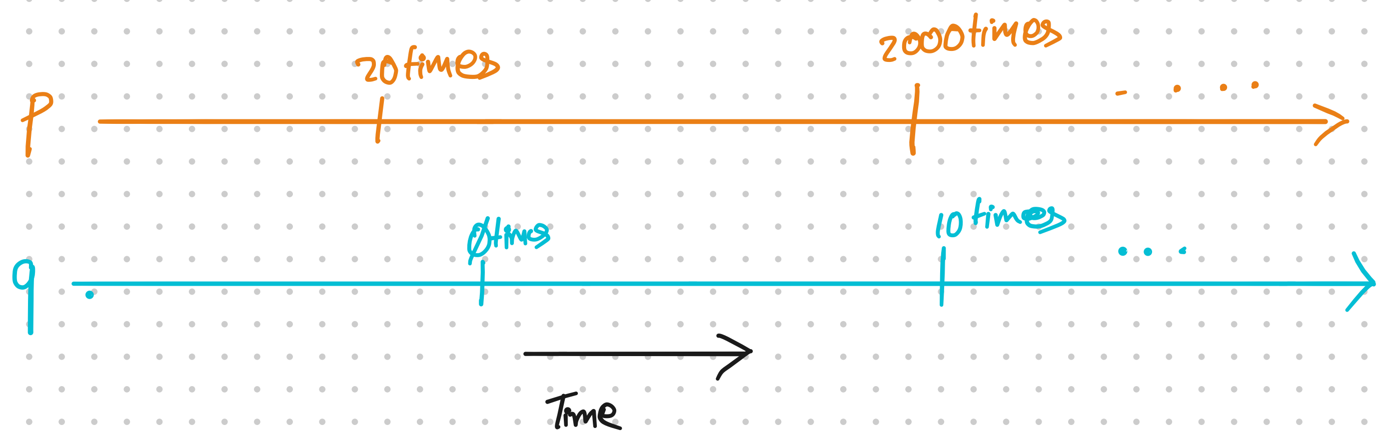
- I/O Automata

Clarifying Fairness

E occurs infinitely often

$\forall n \in \mathbb{Z}^+ \; \exists$ time t after which $|E| \geq n$

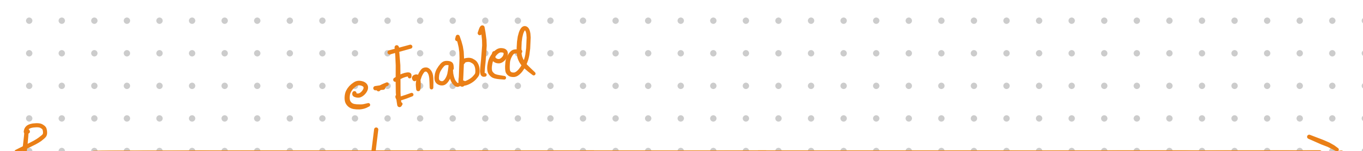
From last class: If process p sends m to q infinitely often then q receives m infinitely often



Do not need all the messages to get through.

Fairness $\not\Rightarrow$ Reliable.

But really the bit of fairness that will be important to us for most of the semester is the simpler form (weak fairness)



- Resets a timer
- Message sent to P over a reliable channel
- ...

$\Rightarrow \exists$ Time t e is delivered

- Timer goes off
- P receives & processes message
- ...

Some Math Preliminaries

- Partial & Total Orders
  ↳ Feature in today's discussion
  ⇒ Play a starring role next week when we discuss LINEARIZABILITY.

- $\leq$: Relation defining a reflexive/weak/non-strict order

  Defined over some set X
  - $\mathbb{Z}$
  - Events
  - ...

  $a, b, c \in X$

  1. $a \leq a$ [Reflexive]
  2. $a \leq b \wedge b \leq a \Rightarrow a = b$ [Antisymmetry]
  3. $a \leq b \wedge b \leq c \Rightarrow a \leq c$ [Transitive]

- Total order

  $\forall a, b \in X \quad a \leq b$ or $b \leq a$

  - Natural numbers $\{ \leq$

Reals

- $X \equiv \mathbb{Z} \times \mathbb{Z}$ $(0,0)$, $(1,0)$, $(1,2) \ldots$
  - Lexicographic order

$$(a_0, b_0) \leq (a_1, b_1) \equiv$$

$$a_0 < a_1 \text{ OR}$$

$$(a_0 = a_1 \text{ AND } b_0 \leq b_1)$$

- Partial Order
  - Not all elements in X can be ordered

$$X \equiv \mathbb{Z} \times \mathbb{Z}$$

$$(a_0, b_0) \leq (a_1, b_1) \equiv$$

$$a_0 \leq a_1 \text{ AND } b_0 \leq b_1$$

- $(0, 0) \leq (0, 0)$?
- $(1, 2) \leq (2, 3)$?
- $(1, 2) \leq (2, 1)$?
- $(2, 1) \leq (1, 2)$?

- Last bit of math I promise.
  - Can view Relations as sets

$\mathbb{R}, \leq : \mathbb{R} \times \mathbb{R}$ $(r_0, r_1) \in \leq \iff r_0 \leq r_1$

$$((1,2), (2,3)) \in \leq$$

$((1,2),(2,1)) \notin \leq$

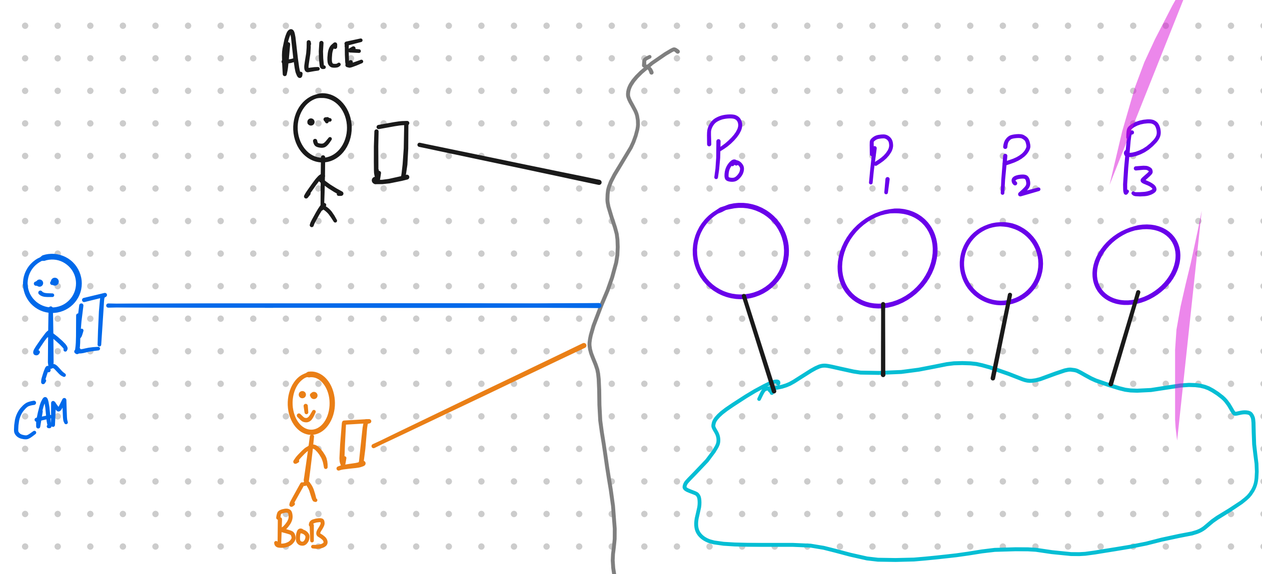
- Paper will talk about finding $\leq_A$ s.t.

$$\leq_A \supseteq \leq_B$$

Meaning $x \leq_B y \Rightarrow x \leq_A y$

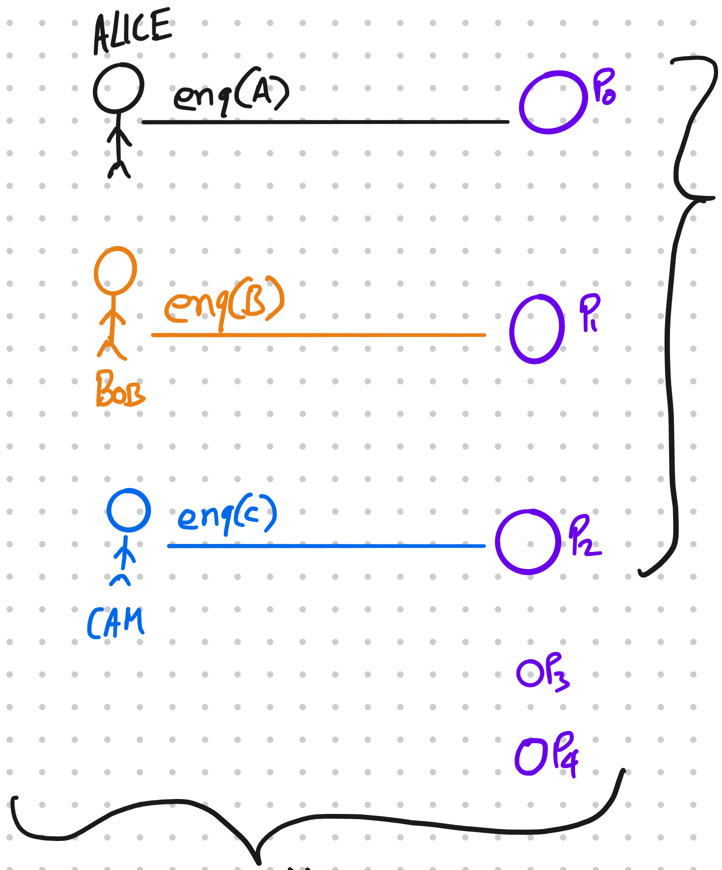
But might have $c \leq_A d$ where $\underline{(c,d) \notin \leq_B}$

OK! Back to Correctness OR Time OR...



1. Messages/orders/... ordered first come first serve
2. FIFO Queue
3. ...

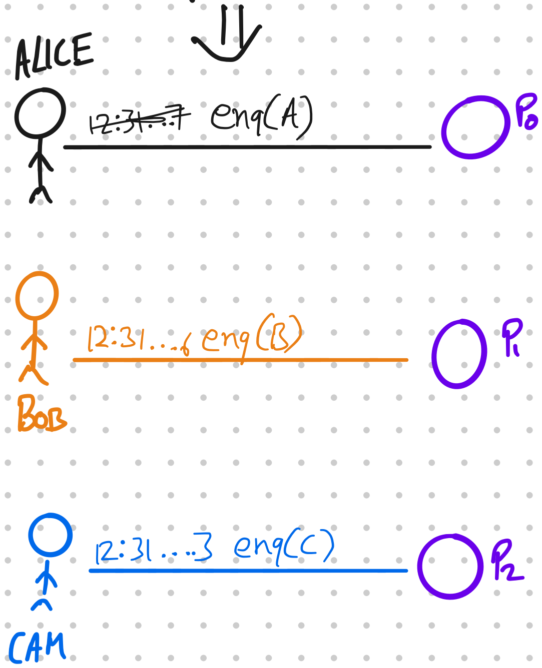
NEED TO ORDER OPERATIONS/EVENTS/THINGS/...



ABC
CAB
...

What is the correct queue?

⇓



PROBLEMS

① Relativity: No notion of a universal time

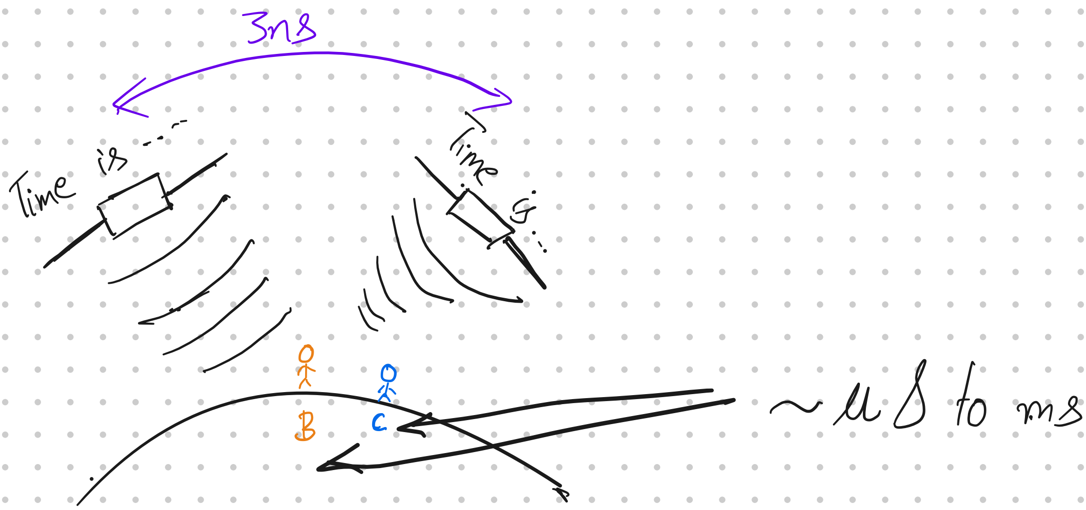
Alice's clock runs at a diff rate than Bob's or Charlie's, etc.

FUN BLOG: EXPERIMENTAL CONFIRMATION:

② Limits on time synchronization

- Impossible in asynchronous model

- Perfect sync impossible even with
assumptions about message delays
(See optional paper for this week)



Implication: To use clocks to sync
- Assume min interval b/w events that need to be synced

AND/OR – How to order events that occur too close to each other.

What we will assume: TIME-FREE MODEL

- Will not read clocks/time in our protocols, not attach time to messages, etc.
- Will use timers (for timeout) but

Reflects reality

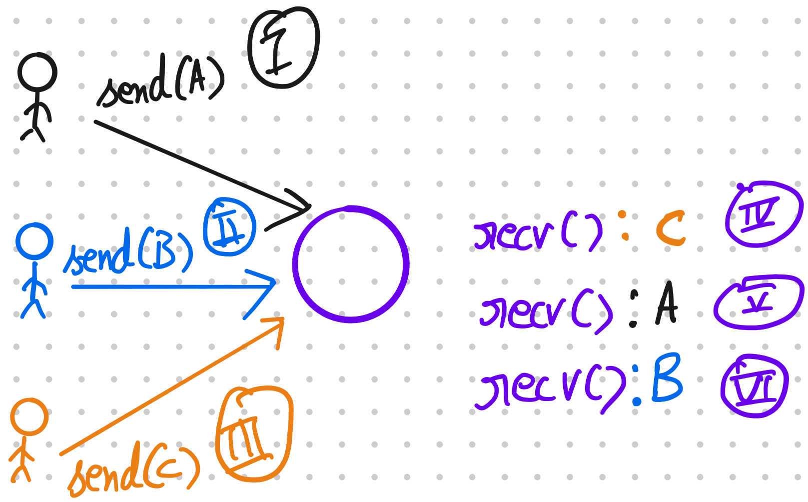
Timer goes off non-det. some time after timeout expires

set timer for 5 sec

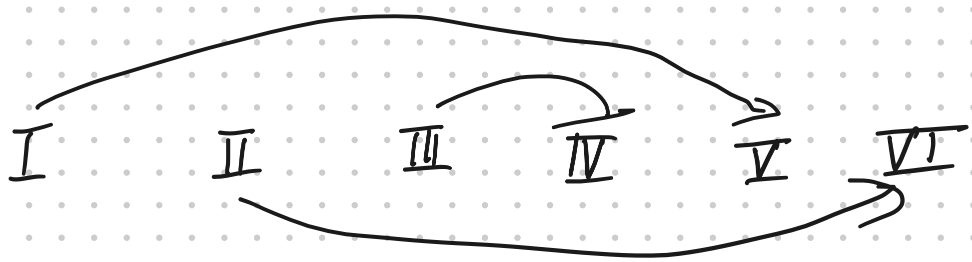
in most cases!

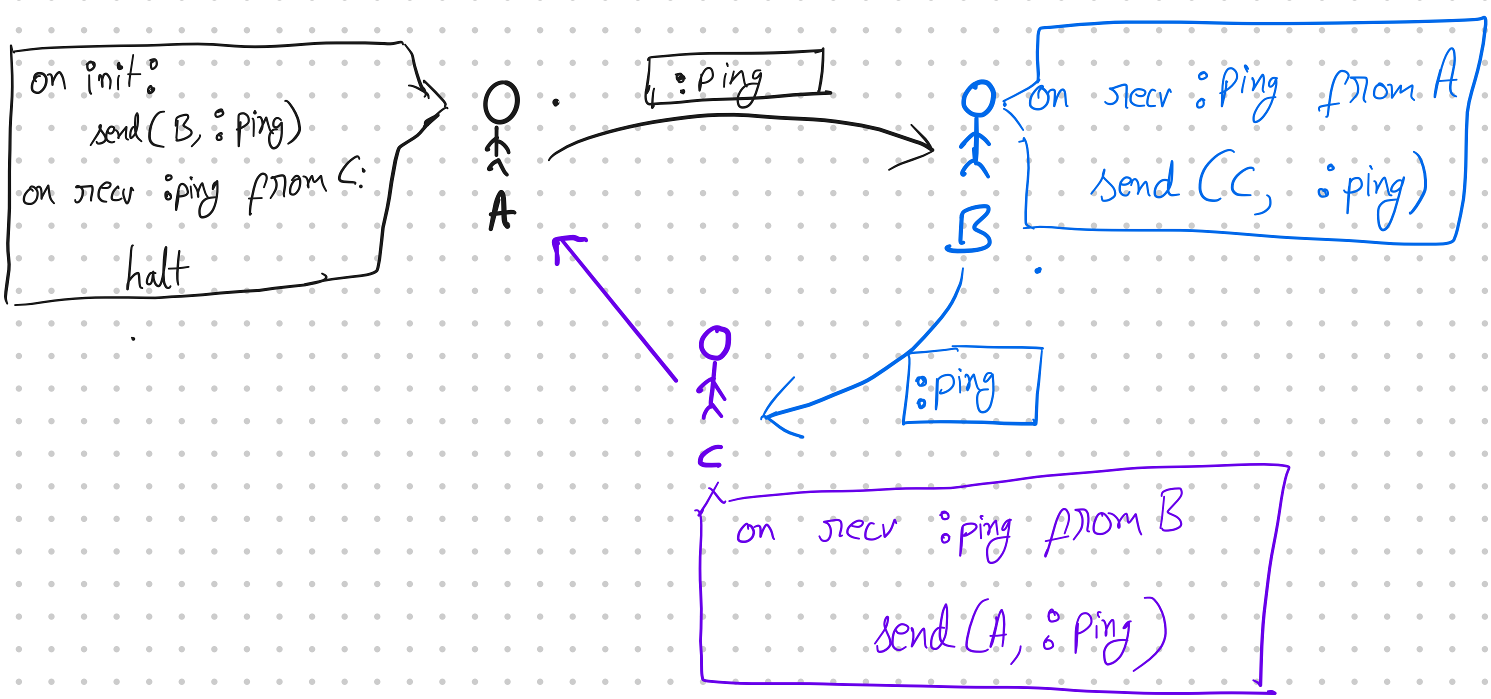
⇒ Protocol sees timer event in $[5s, \infty)$

What can we order without clocks?



$\preceq$ Lamport's Happens Before





A ... B receive

A: send → B: receive

B: send → C: receive

C: send → A: receive

OBSERVE: ① Happens-Before is a partial order

- Orders events that are CAUSALLY CONNECTED

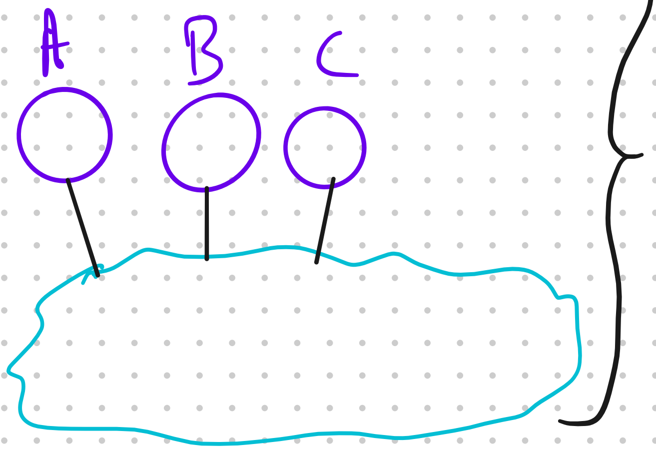
② Can equivalently consider how to order states

A.send	B.receive + B.send	C.receive + C.send	A.receive
A after init	B after receiving ping from A	C after	A after

CORRECTNESS

Specify how state evolves during execution

STATE:

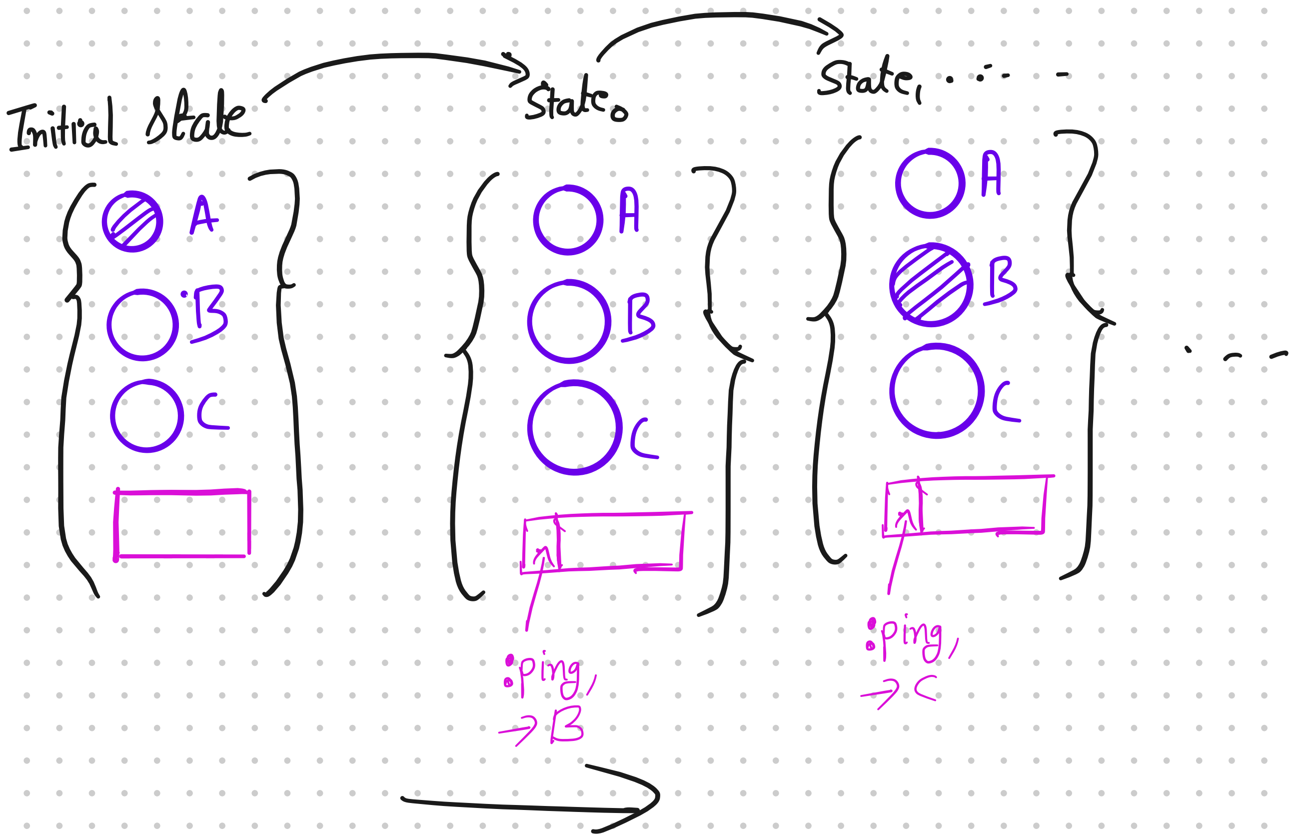


A, B, C's memory

+

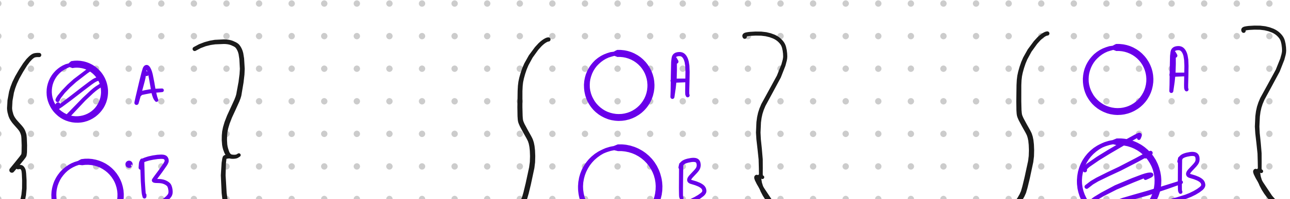
Messages that have been sent but not received or dropped

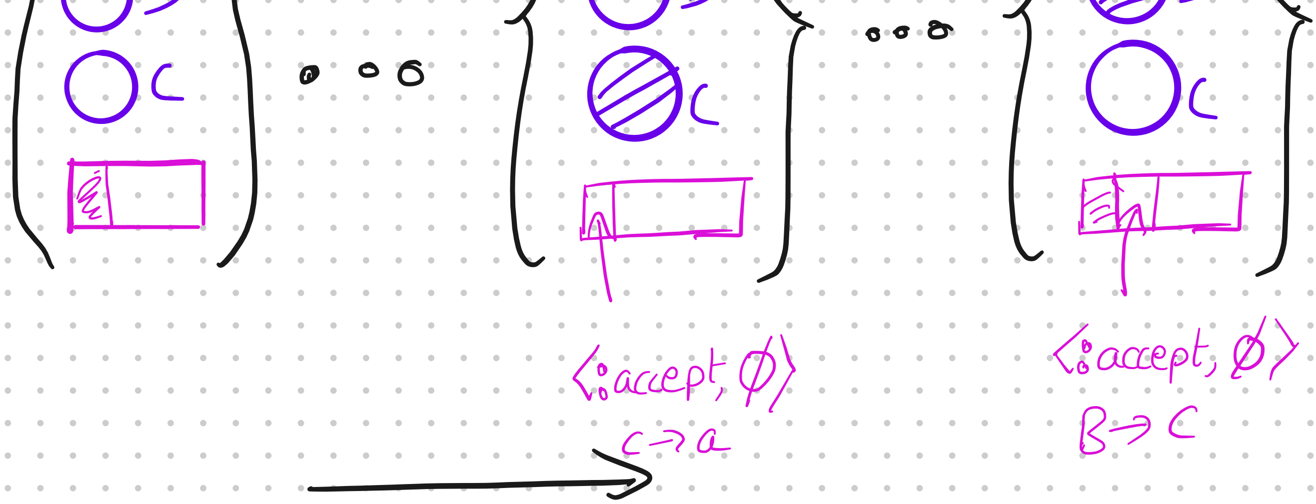
Pending messages

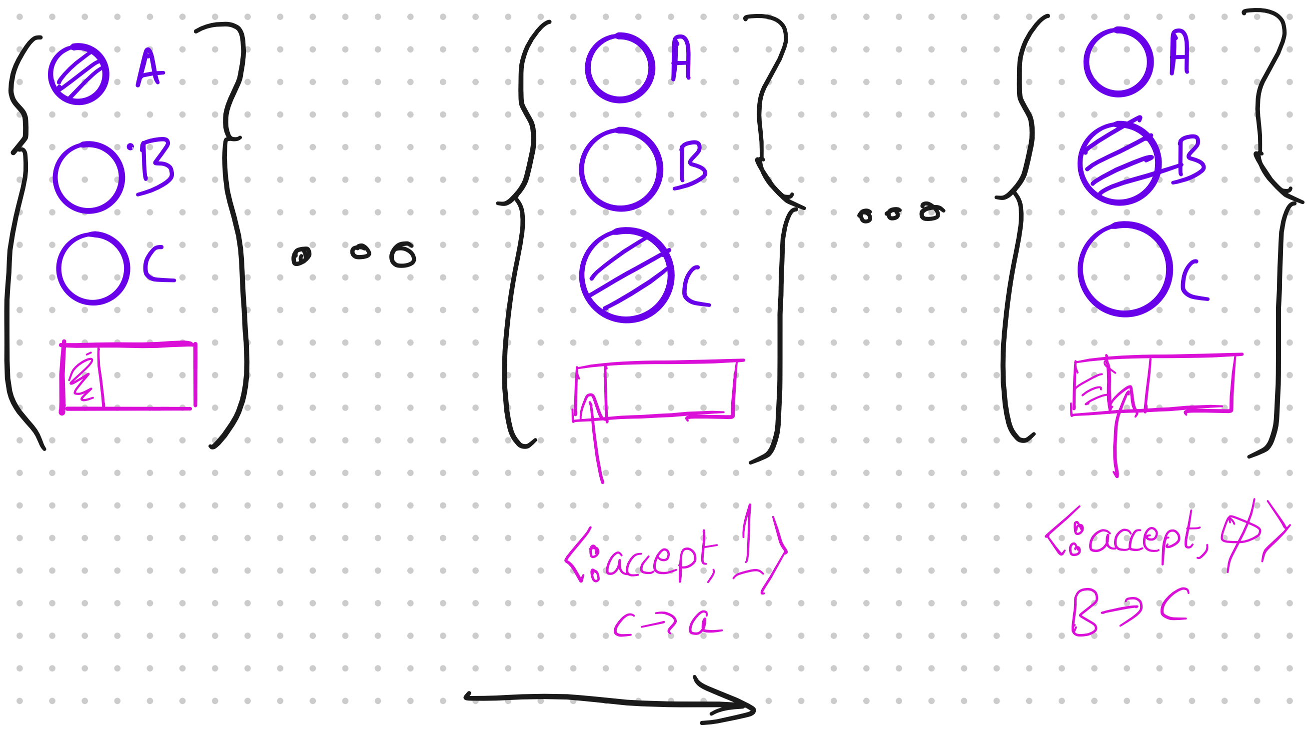


P Safety: Some bad state evolution never occurs OR some predicate never holds.

- If a process p sends ⟨:accept, ∅⟩ then after that no process sends ⟨:accept, 1⟩







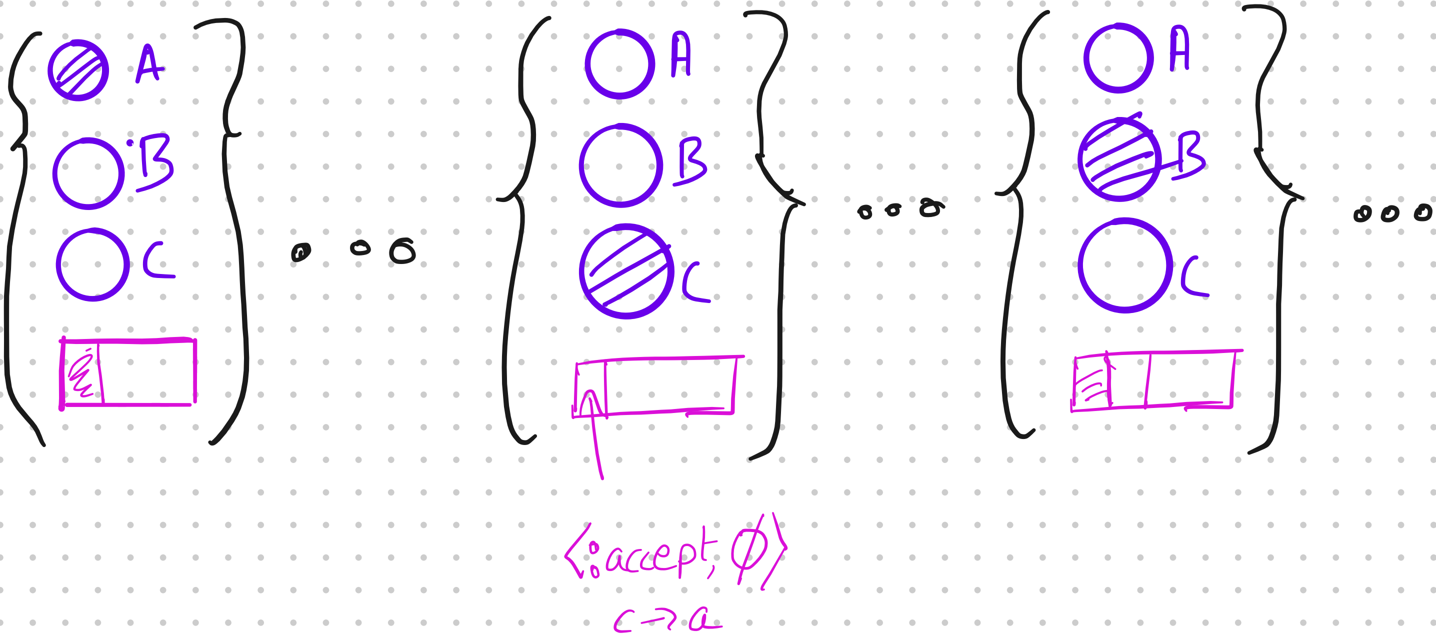
OBSERVATIONS:

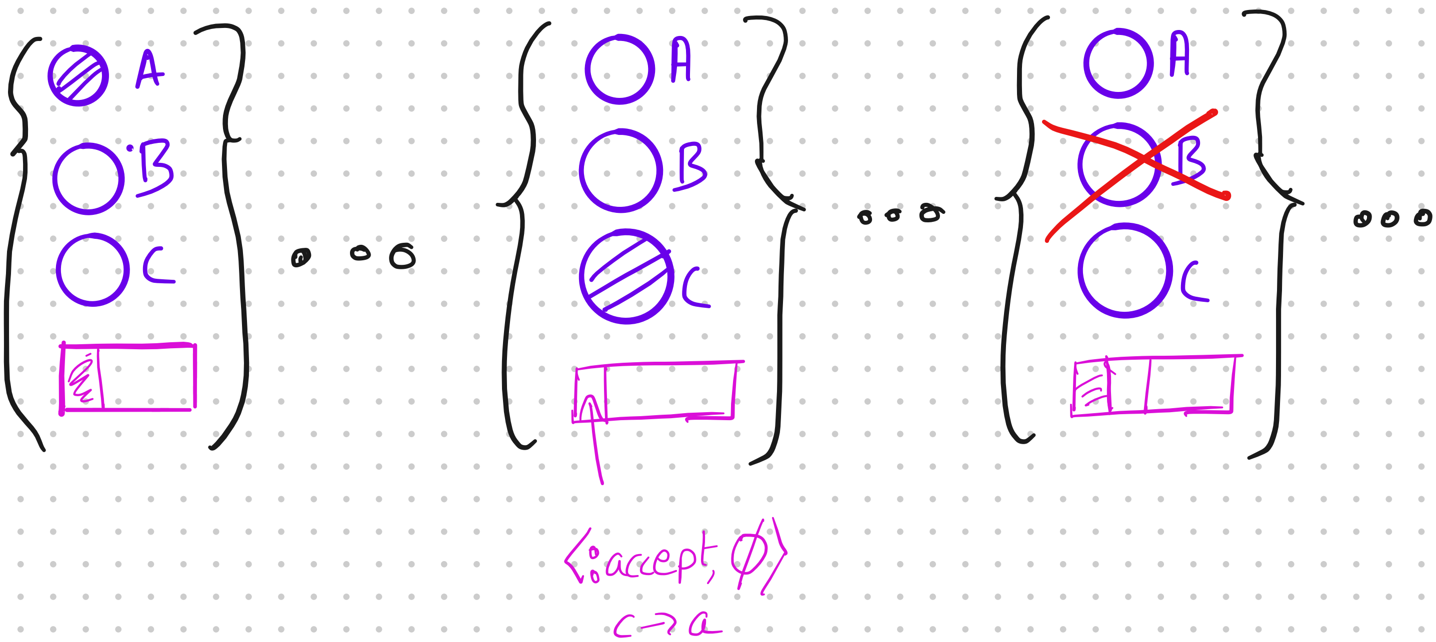
- Given a sequence of states (TRACE) can determine if safety condition has been VIOLATED by evaluating predicate.

- Once a safety condition is violated, no subsequent change is going to fix it/make it hold.

Liveness : Some good thing eventually happens OR some predicate eventually holds

- If a process p sends ⟨:accept, ∅⟩ then EVENTUALLY all processes send ⟨:accept, ∅⟩





OBSERVATION: Just evaluating predicate is not enough

Instead given a trace $\underbrace{T}_{s_0,\, s_1,\, s_2,\, \ldots,\, s_n}$ need to ask

if $\exists$ a sequence σ of states that

ⓐ Is valid as an extension to T

↳ (OR VALID STARTING FROM S_N)

E.G.,- If in fail-stop model, no failed process acts.

- . . .

ⓑ Predicate holds for .

$$\boxed{T\sigma \models P}$$

Paper: Uniform and Absolute liveness

↳ Additional properties that hold for σ

- Uniform: $\exists \sigma$ THAT APPLIES TO ALL EXECUTIONS

- ABSOLUTE: If $\sigma \models P$ then $\forall \tau$, $\tau\sigma \models P$