Distributed Systems

Lecture 4

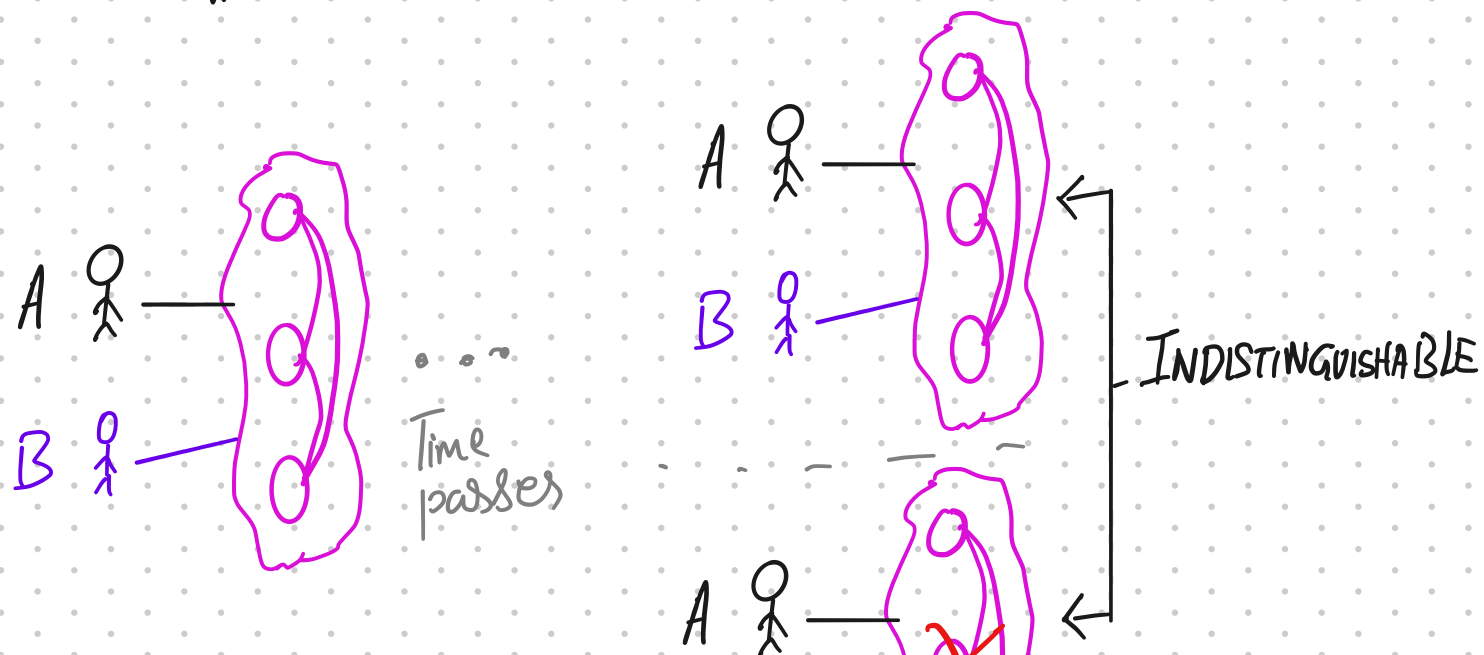
RSM

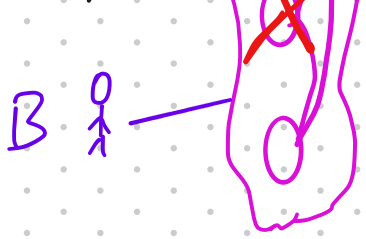
Replicated State Machines

- Problem: How to Build A Fault Tolerant ???

??? ≡ Airline reservation system (Lamport)
Key value store (Chubby, Etcd, ...)
Database (...)
Ice cream stand (...)

What we want





Why?

- System is an abstraction BOUNDARY

Good Design → Don't expose internals.

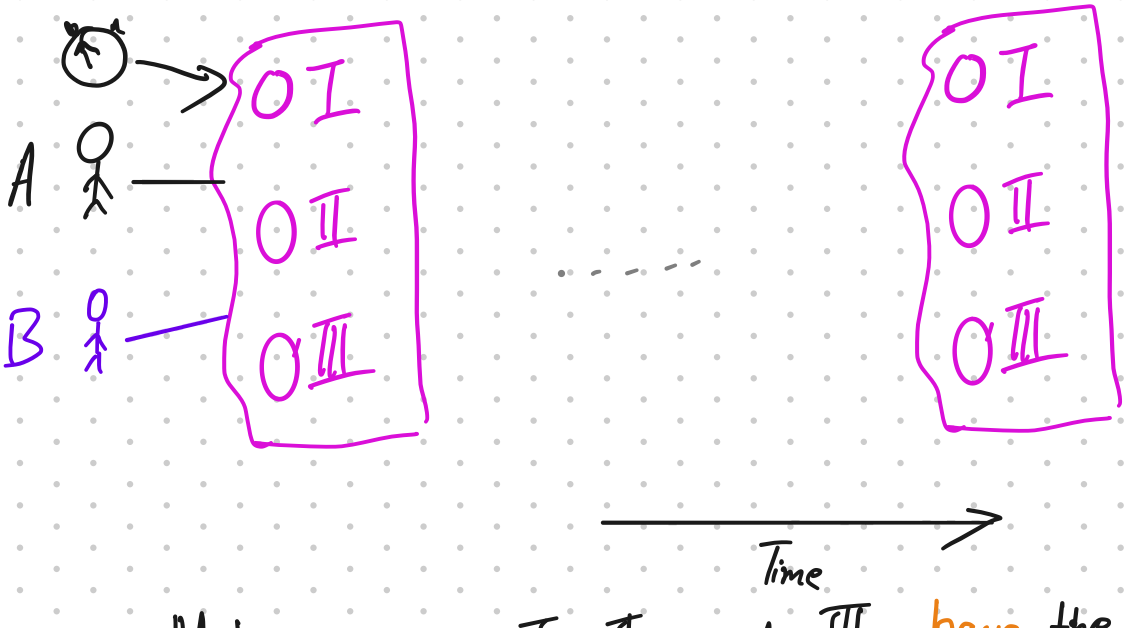
How?

Many ways

↳ Several that are task specific

But hard to get right.

RSM: A generic recipe



Make sure I, II and III have the

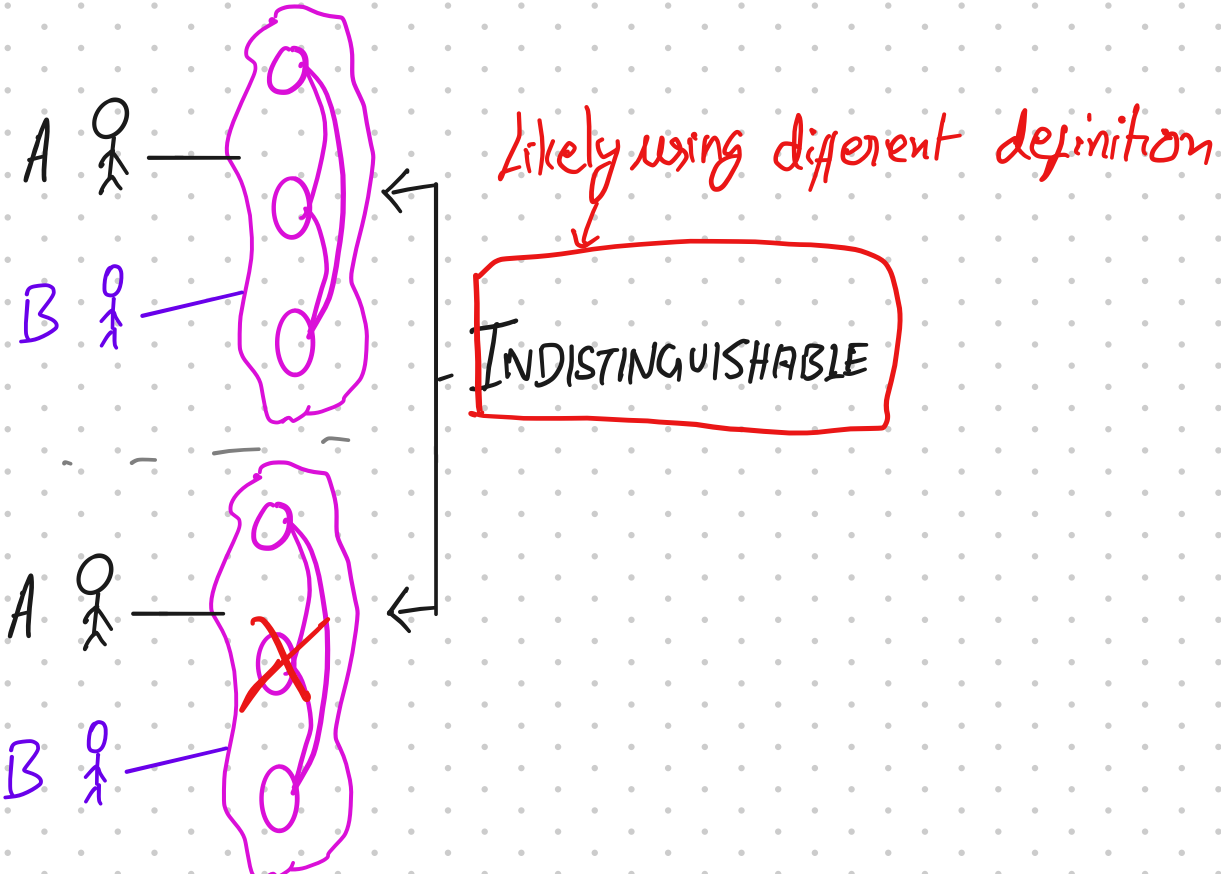
Same state & behave identically

Behave identically : Deterministic

Out can depend on
- Current state
- In

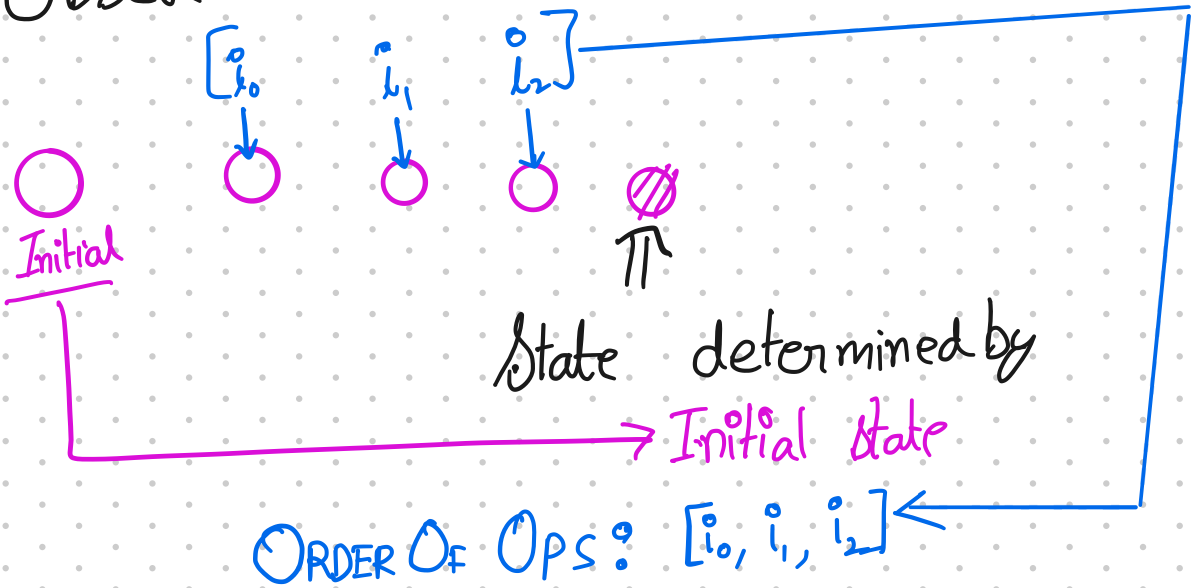
- No dependence on time
- No dependence on ANYTHING OTHER THAN INPUTS + STATE
- No reading /dev/random
...

Be suspicious of fault tolerance mechanisms that do not assume DETERMINISM

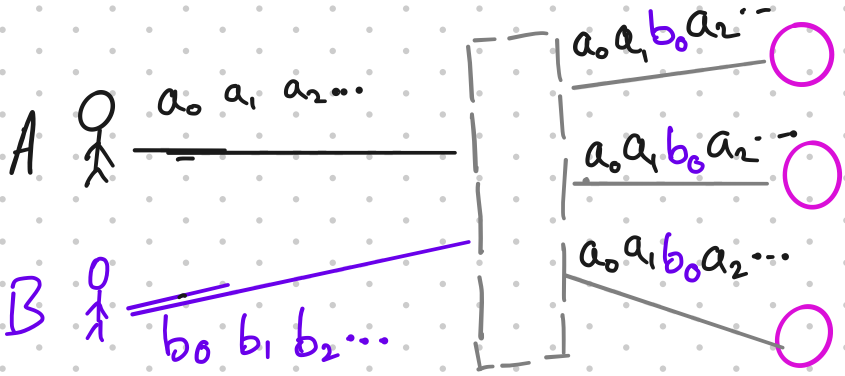


Why Determinism helps.

Observe



Leads to RSMs



Two requirements

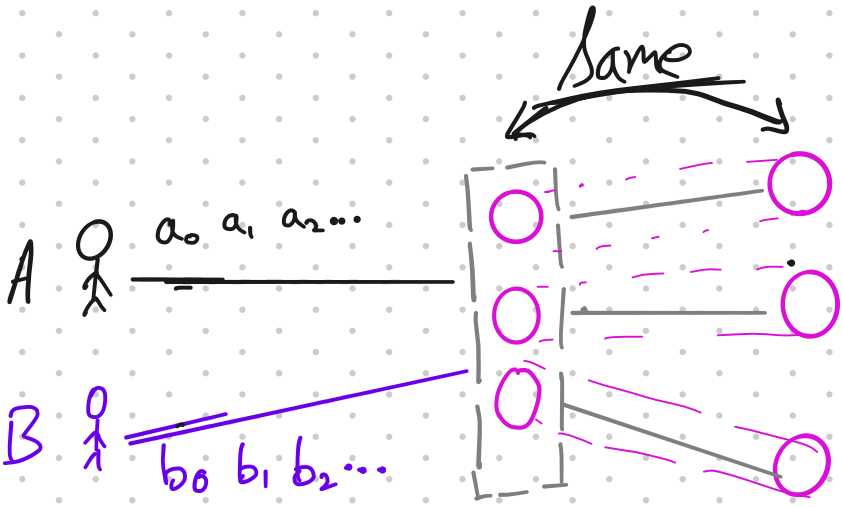
Agreement: All correct processes get the same set of requests (commands)

Ordering: All correct processes execute request in the same order.

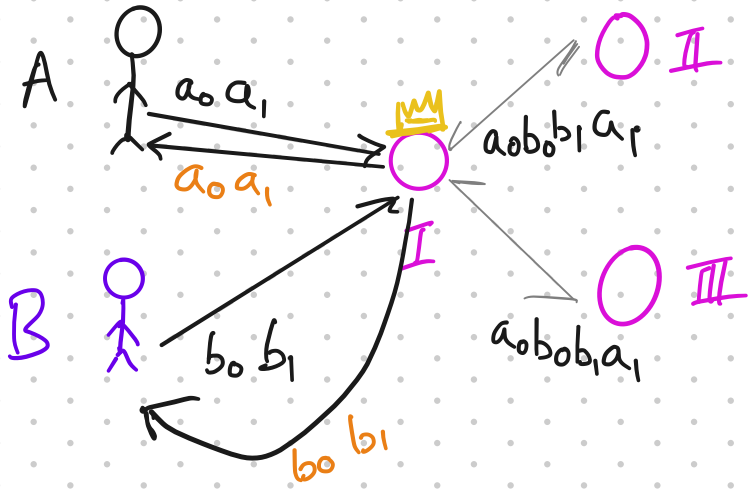
model makes this hard!

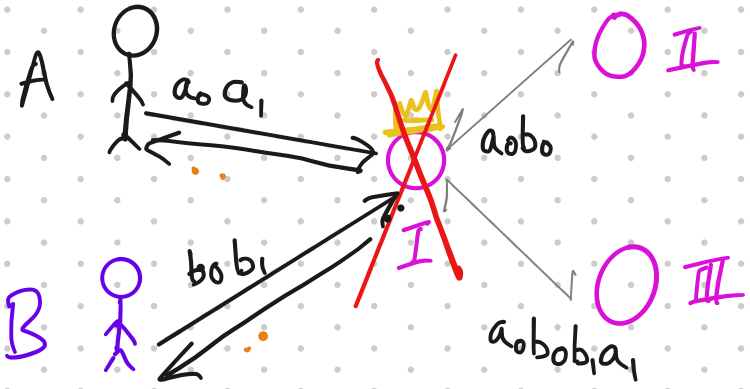
The asynchronous model makes...

FLP (in several weeks) formalizes this notion.



"Leader" based





Questions when designing

→ When can leader respond to a client? [Commit Point]

→ Who becomes leader? [Election]

→ How is leader failure detected?

Commit Req. — Make sure inputs are replicated enough so they are not lost on failure

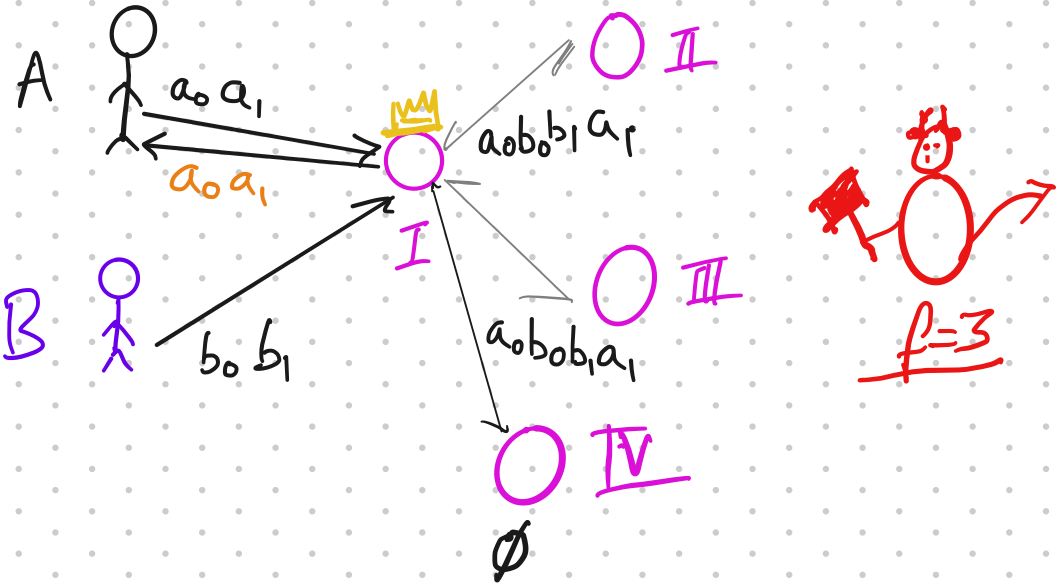
What is enough?

Assume up to f fail-stop failures

- Lower bound:

$$\geq f+1$$

Why?



- Async. model
  - → Things get tricky

- Cannot distinguish b/w failure & msg delay

Decides both what nodes fail & how msgs are delayed

Req/Desirable: Termination

↳ Every request eventually processed

Election req (below) + async → No termination

- Partially synch

↳ Define later in class but roughly:

eventually deployment looks synchronous

→ eventually Can (sort of) distinguish b/w failure & delay

Tricky bit: eventually

- Can be a long time off
- Don't know if we have hit it

But... bound (for replication)

$\geq f+1$ assuming $2f+1$ nodes

More generally – quorum intersection

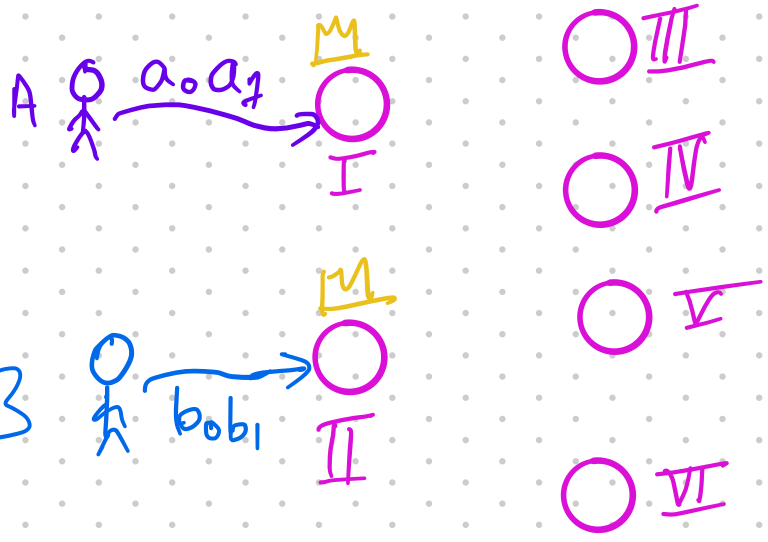
[$n/2+1$ ← But be suspicious]

ELECTION REQ.

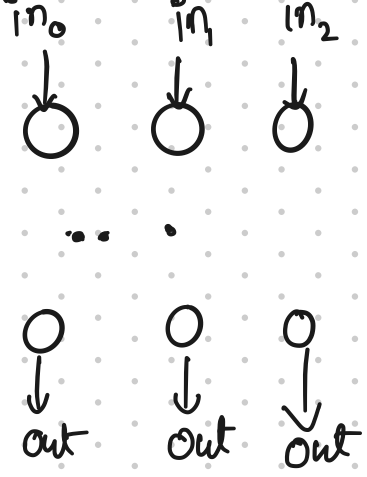
At most one active leader

Agree on leader

Why?



Agreement: Consensus

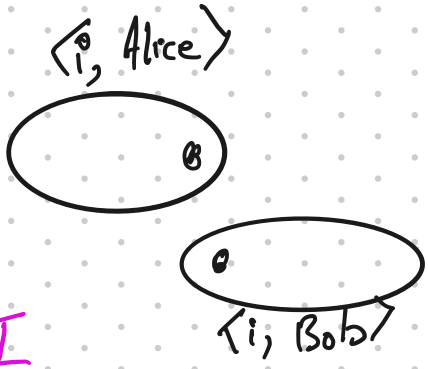


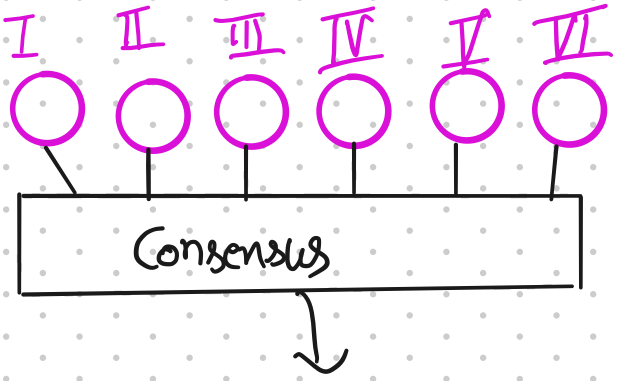
Agreement: If process p & q output v_p & v_q then $v_p = v_q$

Validity: If process p outputs v_p then $v_p \in \{in_0, in_1, \ldots\}$

Termination: Eventually one correct process output a decision

Leader election as consensus





$\frac{n}{2} + 1$

'Leaderless' RSMs

- Will look at a protocol (Rabia) later in the semester
- Not really popular/used

↳ Why?

Tradeoffs

↳ Consensus each round

vs

?

Decoupling agreement & ordering.

Two requirements

Agreement: All correct processes get the same set of requests (commands)

Ordering: All correct processes execute request in the same order.

Above:

agreement + ordering

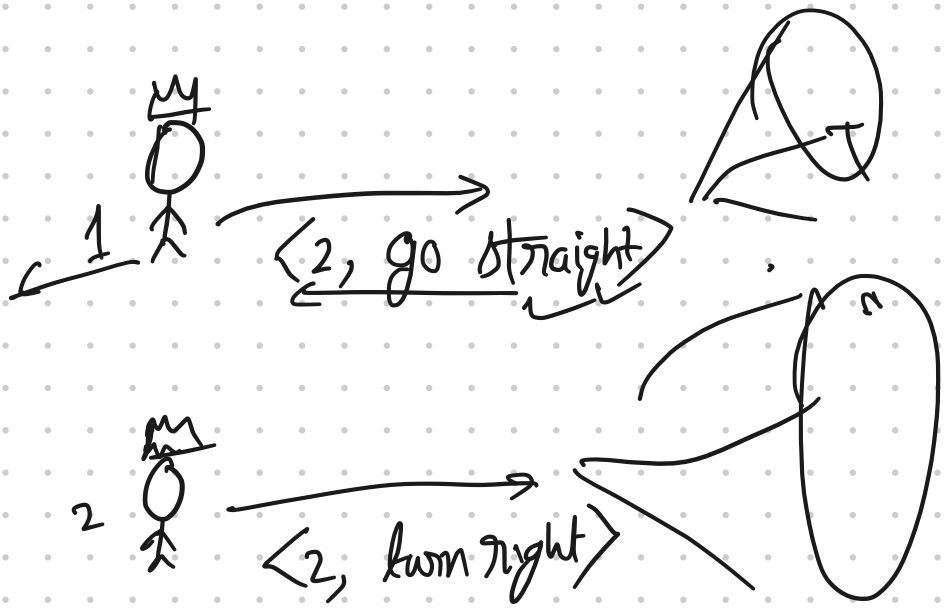
appear coupled

Same mech. for both. etc

Some systems (e.g., Scalog) Decouple them

Why?

How?


1
<2, go straight>
2
<2, turn right>