Distributed Systems

Lecture 4

- Finish Linearizability
- CAP
- Replicated State Machine

Checking linearizability / Seq cst / ...

The rules:

(I) Given: A history

- Total order

(II) Extract a partial order

- Linearizability

$op_1 <_H op_2$

$\iff$ ret $op_1 <$ inv op_2

i $<_H$ ii [2 < 3]

v vi

(i)
1. p Enq(x) A
2. p OK() A

(ii)
3. q Enq(y) B
4. q OK() B

(iii)
5. q Enq(x) A
6. q OK() A

(iv)
7. p Enq(y) B
8. p OK() B

(v)
9. p Deq() A
10. p OK(y) A

(vi)
11. q Deq() B
12. q OK(x) B

Reminder

Inv. object op proc

Ret: object OK(-) proc

(i)
1. p Enq(x) A
2. p OK() A

3. q Enq(y) B
(ii) 4. q OK() B
5. q Enq(x) A
(iii) 6. q OK() A
7. p Enq(y) B
(iv) 8. p OK() B
9. p Deq() A
(v) 10. p OK(y) A
11. q Deq() B
(vi) 12. q OK(x) B

- Seq cst

$op_1 <_s op_2$

$\iff$ op_1, op_2 from same process & ret $op_1 <_{inv} op$

i $<_s$ iii [A, 2<5]

(i) X (ii)

(ii) $<_s$ (iv)

(III) Find an extension (SUPERSET) to partial order that is a total order on operations & meets ADT's sequential spec

Linearizability

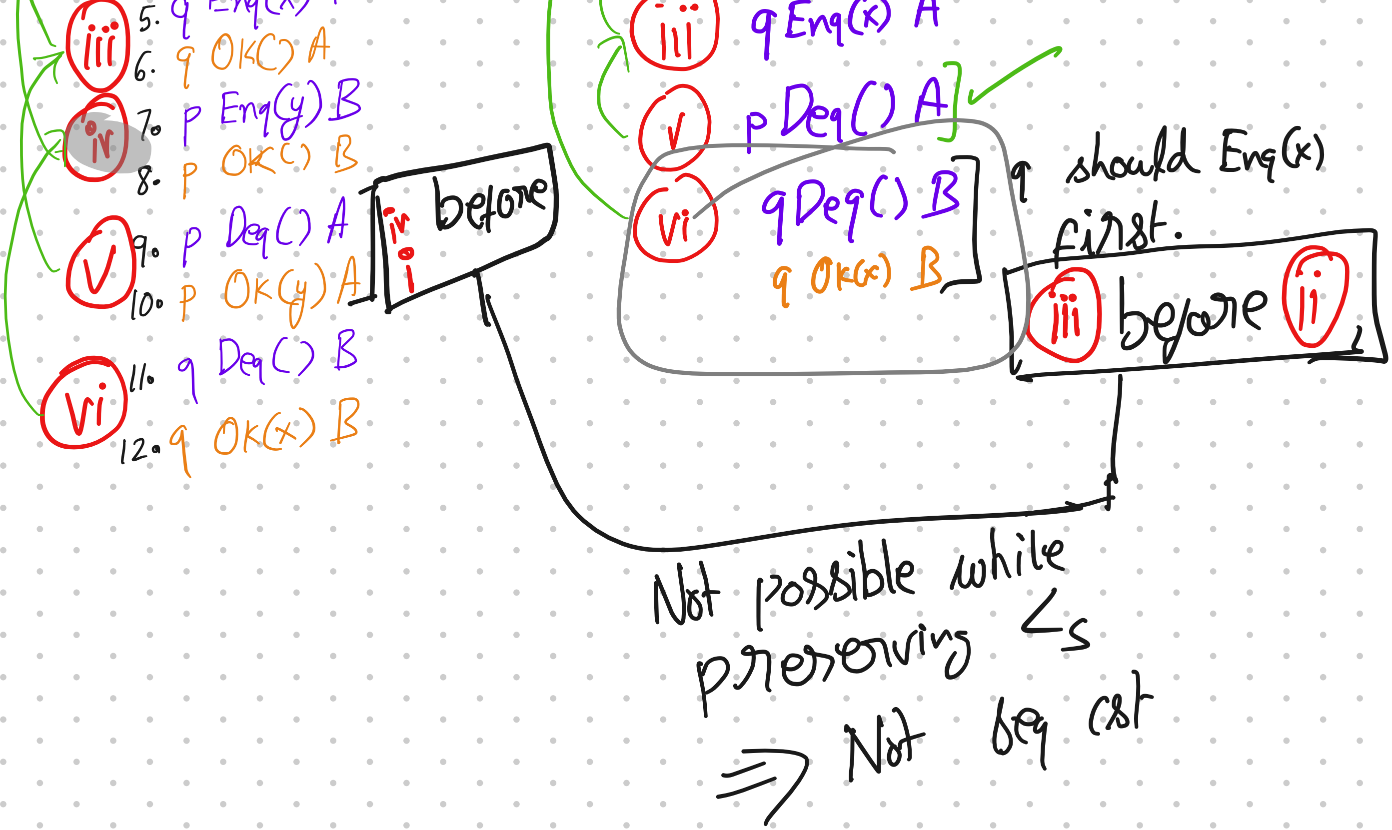
(i) 1. p Enq(x) A
2. p OK() A
(ii) 3. q Enq(y) B
4. q OK() B
(iii) 5. q Enq(x) A
6. q OK() A
(iv) 7. p Enq(y) B
8. p OK() B
(v) 9. p Deq() A
10. p OK(y) A ← p should enq (y) first
(vi) 11. q Deq() B
12. q OK(x) B

No

~~Serializable~~ Seq Cst

(i) 1. p Enq(x) A
2. p OK() A
(ii) 3. q Enq(y) B
4. q OK() B
5. q Enq(x) A

(ii) q Enq(y) B
(iv) p Enq(y) B
(i) p Enq(x) A



Observe: Seq cst – Fine for computed total order to not preserve history total order

H total order

Linearizable – Real-Time

x inv op₁ A

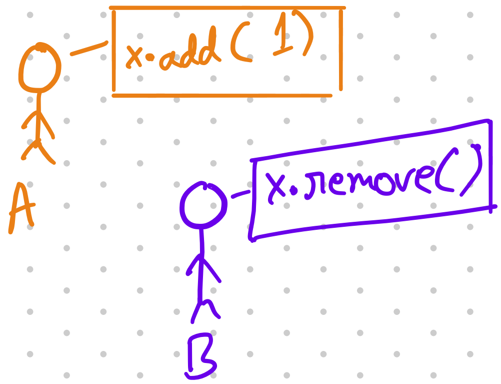
x+y inv op₂ –

x+y+ → ok

Locality

```
add(item) {
    ctr.inc()
    q.enq(item)
    ret OK()
}

remove() {
    if ctr.count() > 0 {
        ctr.dec()
        return q.deq()      ← ⊥ or item
    } else {
        return OK(empty)
    }
}
```



ctr inc() A
ctr OK() A
ctr count() B
ctr OK(1) B
ctr dec() B
ctr OK() B
q deq() B ←
q OK(1) B ←
q enq(1) A
q OK() A

System with counter + queue.
- Linearizable?

System with new adt remove ⇒
if empty return empty
else

Comparing Consistency Models

A stronger than B

For any history H: H⊢A ⇒ H⊢B
but not H⊢B ⇒ H⊢A

A equivalent to B

H⊢A ⇔ H⊢B

Linearizability

$op_1 <_H op_2$

$\iff$ ret op_1 < inv op_2

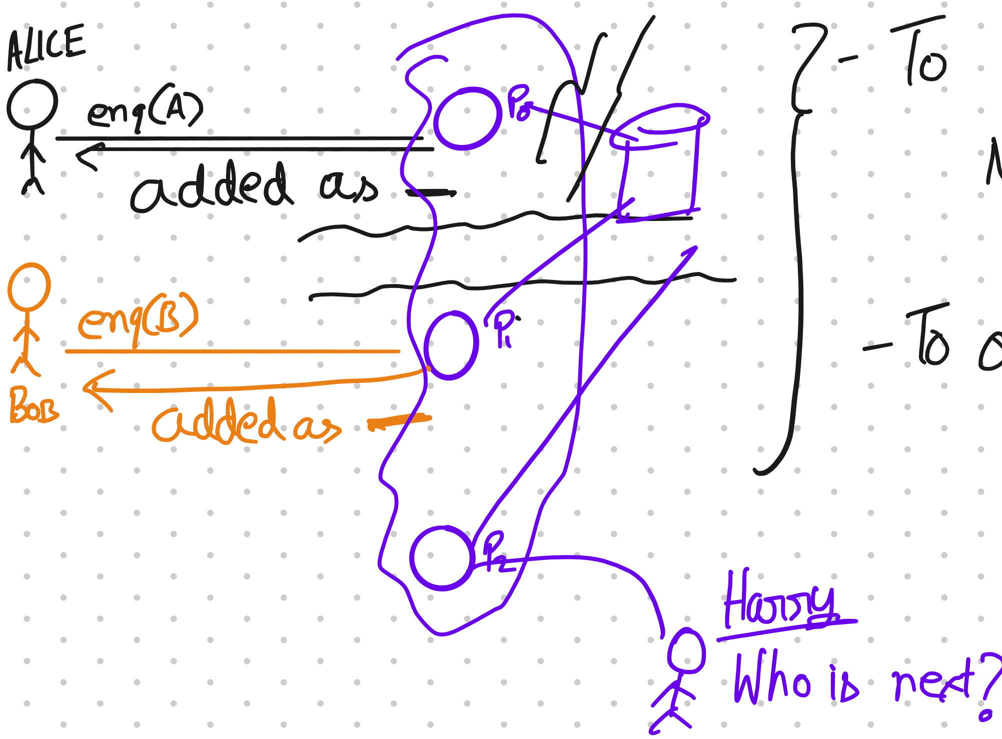
Seq Cst

$op_1 <_S op_2$

$\iff$ op_1, op_2 from same process ret op_1 < inv op

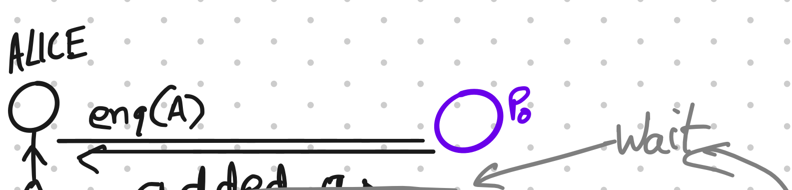
CAP

CORE MESSAGE: Linearizability comes at a cost

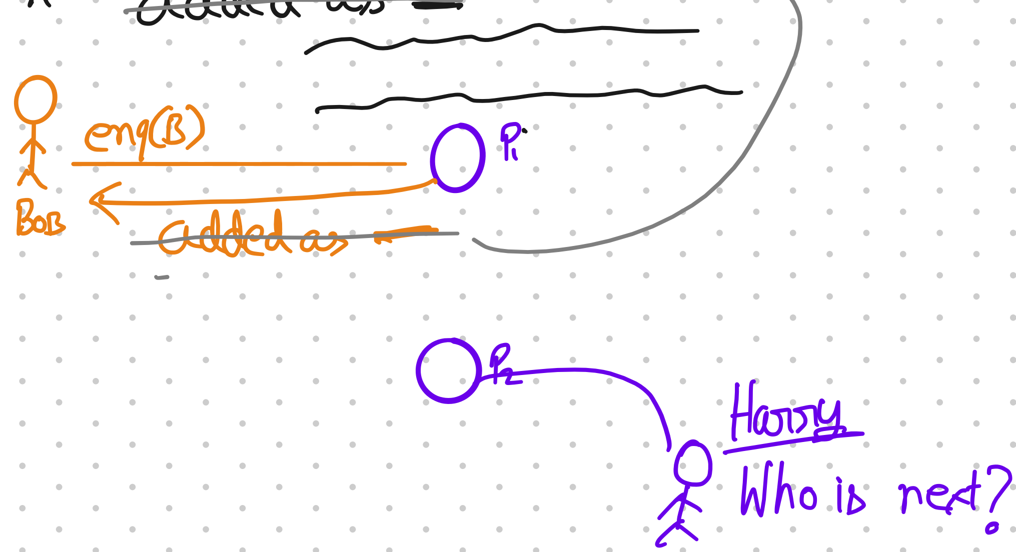


- To be linearizable Need to order enq(A), enq(B)
- To order, need to know about the other

Choices



Wait ⇒ Not available



[Not sure how long it will be]

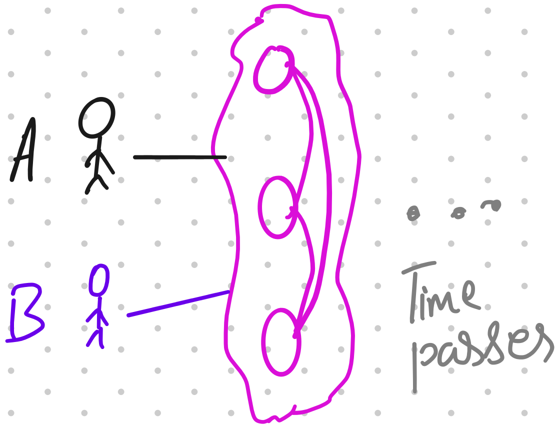
Give up on linearizability

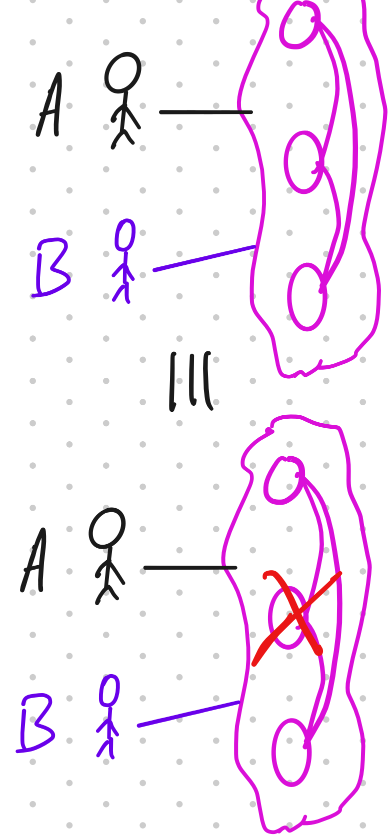
Replicated State Machines

- PROBLEM: HOW TO BUILD A FAULT TOLERANT ???

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

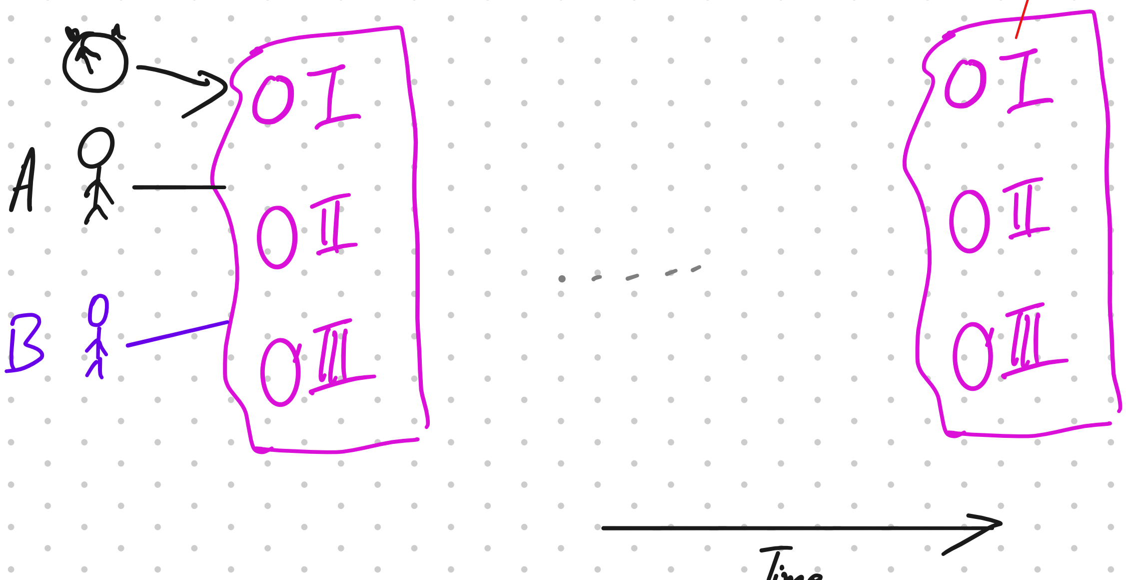
What we want





Note: Many possible solutions BUT

- Hard to design a correct one.
- So instead: identify a recipe.



- Make sure I, II and III always have the same state & behave identically

Same state

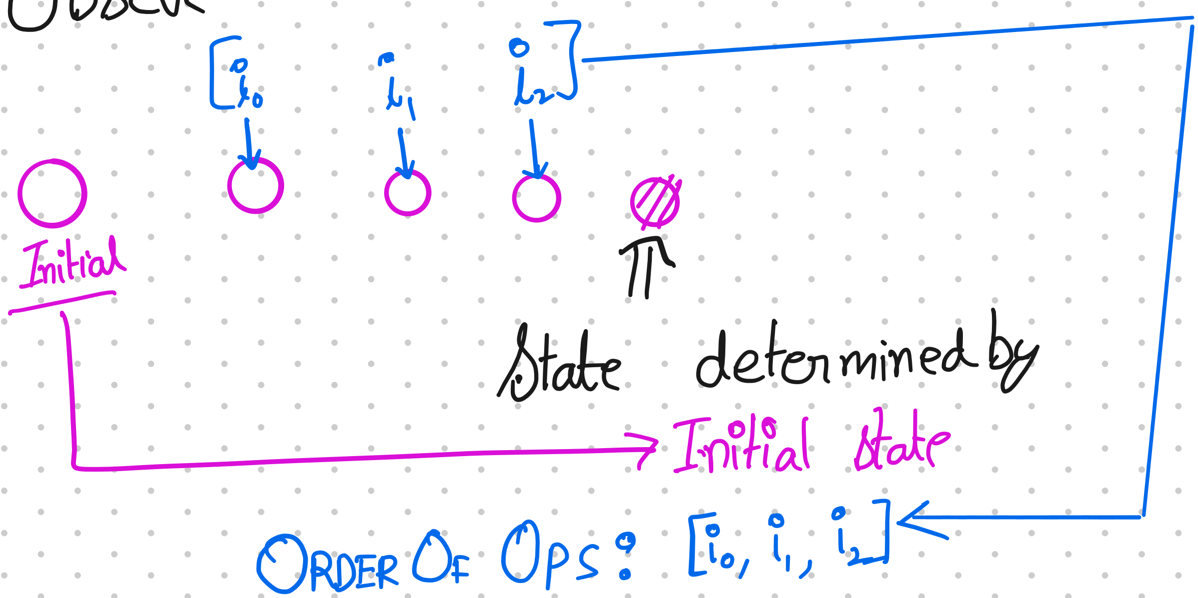
Behave identically : Deterministic

Out can depend on
- Current state
- In

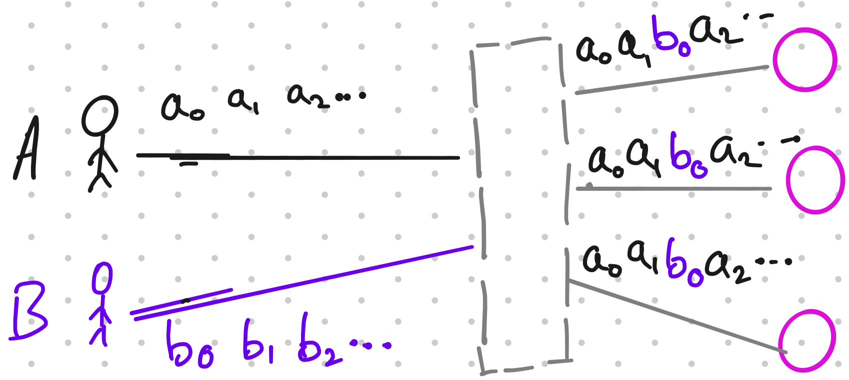
- No dependence on time
- No dependence on observation about the world
- No reading /dev/random
...

Assumed by nearly all fault tolerance mechanisms.

Observe



Leads to RSMs

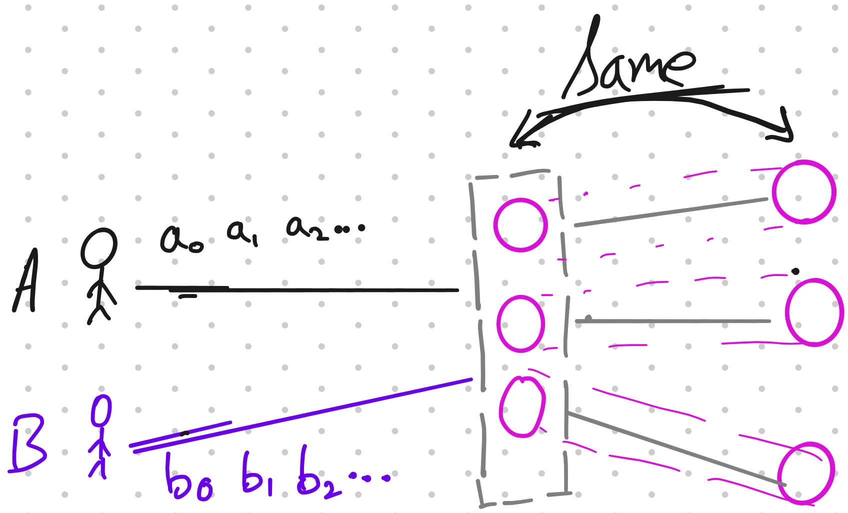


Two requirements

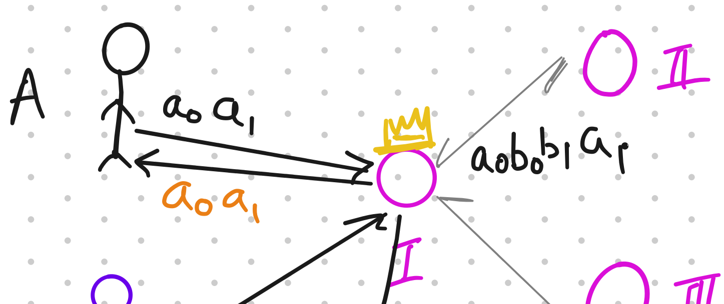
Agreement: All <u>correct</u> processes get the same set of requests (commands)

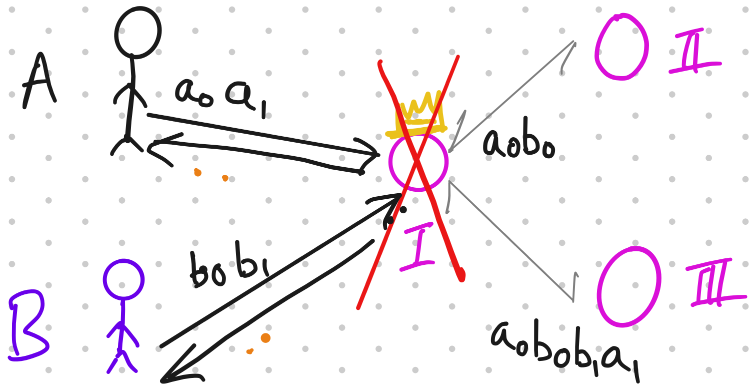
Ordering: All <u>correct</u> processes execute request in the same order.

The asynchronous model makes this hard!
Formally show this later in class



"Leader" based

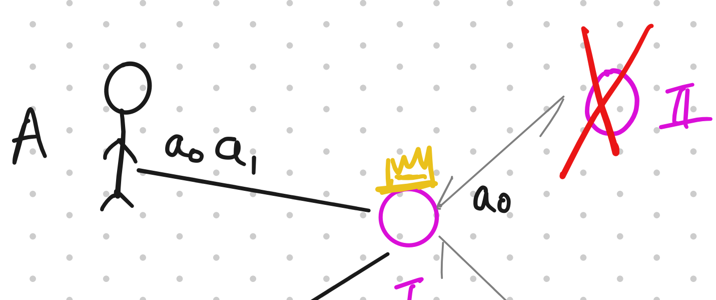


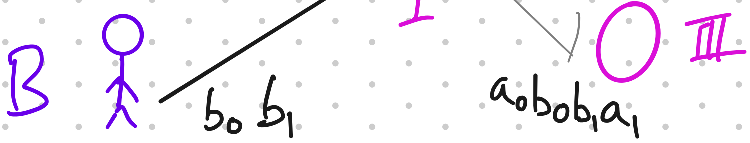


- How to detect leader has failed?
  - Time out

- Who becomes the next leader?
  - Req: At most one active leader.

  Agree on leader



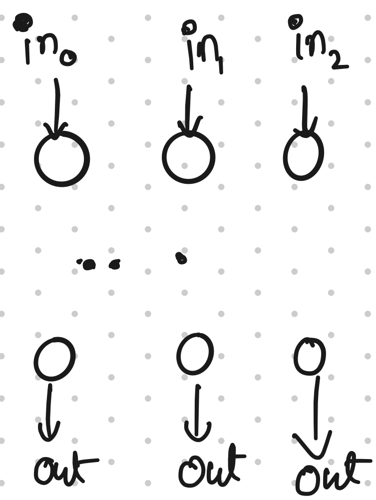


- When is it safe to execute request & return result?
  - ↳ COMMIT POINT

When? Enough processes agree that they know command.

To recover from failures

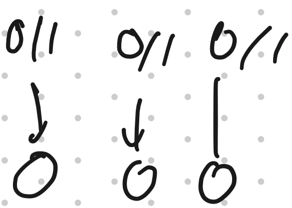
Agreement: Solved by CONSENSUS PROTOCOLS

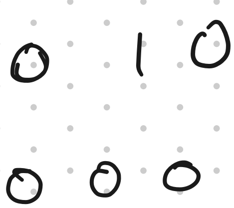


Agreement: If process p & q output v_p & v_q then $v_p = v_q$

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

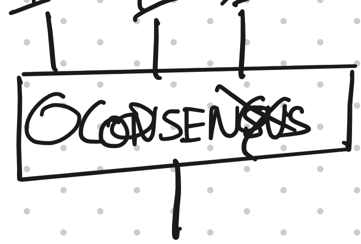
Termination: Eventually ~~all~~ one correct processes output a decision





recv m: do
return / one
end

- Leader Election



- Sufficiently Replicated

(idx, ref