Distributed Systems :- More Linearizability

Where we are

- Consistency models
  - ↳ What order of operations can be observed on a concurrent data structure
    (Given by an ADT with a seq. spec.)
- History : <inv ...> <res...>
- Linearizability :

  An implementation is linearizable

  $\iff$

  any history H produced by impl. can be extended to H' s.t we can find a sequential history S where $<_{complete(H')} \subseteq <_S$

- Linearizability - B

$$<_H \subseteq <_S$$

- Locality

  H Linearizable (or other consistency that provides locality) $\iff$ H|o linearizable for all objects o appearing in H

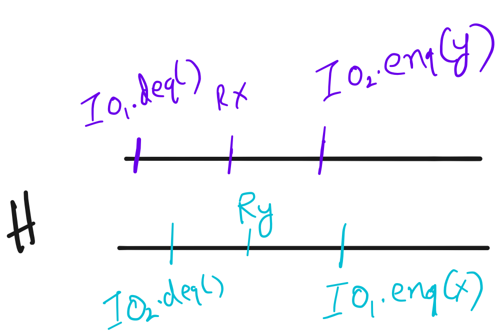
- Non-blocking

  H linearizable $\iff$ H contains ⟨inv o op⟩ w/o ⟨res o v⟩ $\Rightarrow$ we can find v s.t.

  H · ⟨res o v⟩ is linearizable.

- Linearizable-B is not local & not non-blocking.

  ↳ Saw two counter examples last class



$H|O_1$: Lin-B ✓
$H|O_2$: Lin-B ✓
H : Lin-B ✗

} NOT LOCAL

H deq Rx

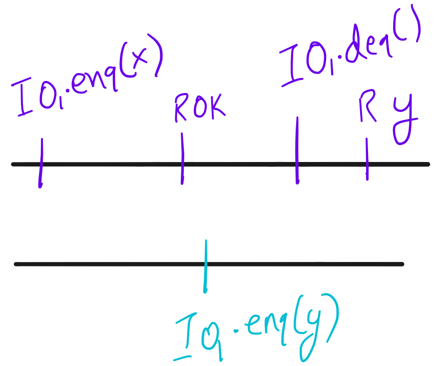
$H|O_1$: Lin ✗

enq(x) HIO₂: Lin ✗

H' deq Rx enq OK

S

I



H: Lin-B ✓

Ho ⟨R Ok⟩: Lin-B ✗

} Not non-blocking

H: Lin ?

Today

- A couple of other consistency models
  - ↳ Seq Cst
  - → Strict Serializability

  Comparing the strength of consistency models.

- Building linearizable "things"

- CAP

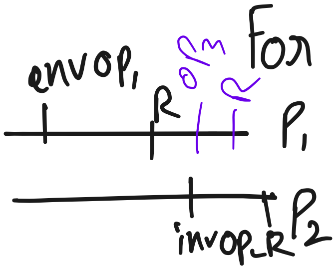
Sequential Consistency / Seq Cst

- Often the strongest memory model supported by language specs: C++ `std::memory_order::seq_cst`
  Rust `Ordering::SeqCst`

- Guarantees
  - Can find a total order for all processes
  - Total order does not violate process order for any process.

Translating to our terminology

H: History. [But $<_H$ is unused]

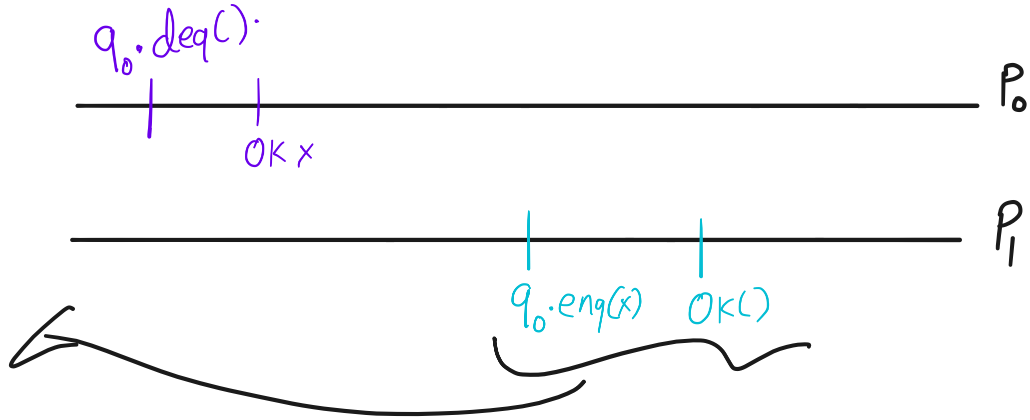


For any process $p \in \Pi$, $<_p$: order of operations.

$H|_p \quad <_p \equiv <_{H|p}$

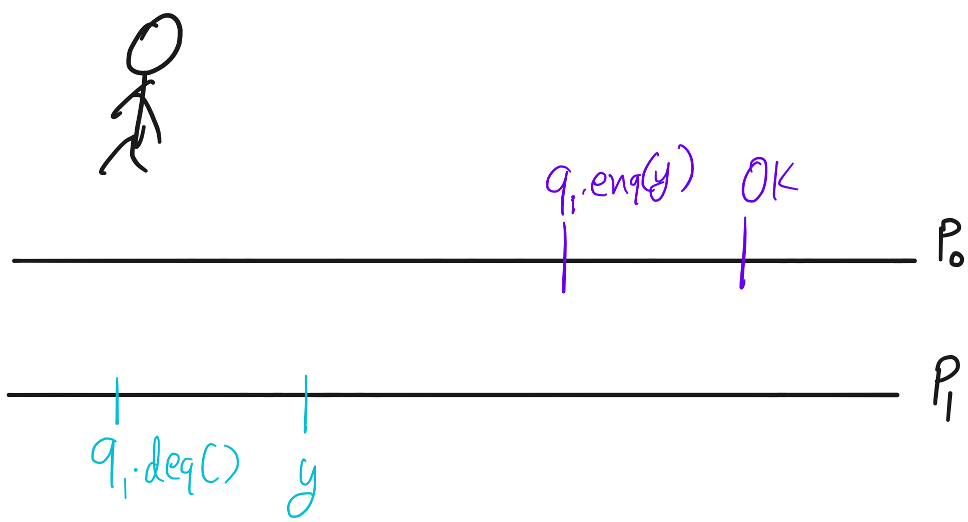
[Remember, we assumed processes are sequential]

$$H \longrightarrow S \quad \text{s.t.} \quad <_p \subseteq <_S \ \forall p \in \Pi$$



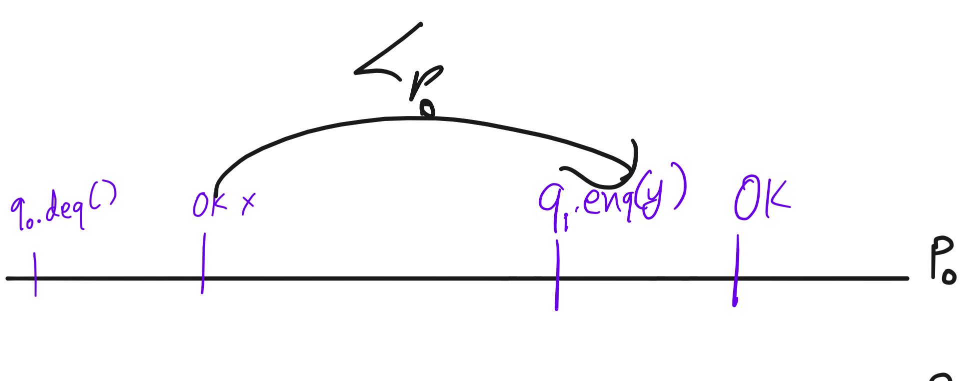
Seq cst ? ✓

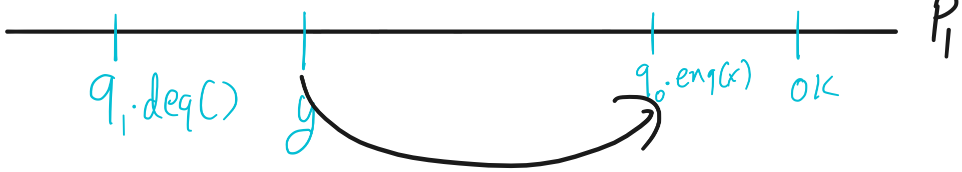
Linearizable ? X



Seq cst ? ✓

Linearizable ? X





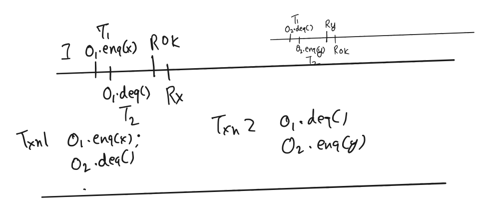
$<_{P_1}$

Seq Cst?

Is Seq Cst local? No

Strict Serializability

- Transaction
  - ↳ Sequence of operations issued by one process
  - → Operations might target different objects
  - → Must be ISOLATED



- H strict serializable
  - ↳ H linearizable
  - → s.t in linearization S

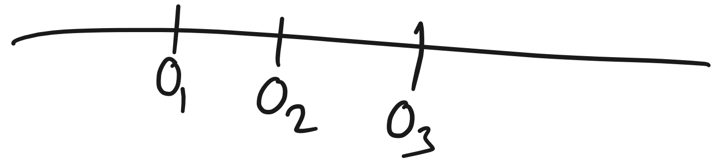
    cannot find O_1, O_2, O_3

    s.t. $\nexists$ $TxN(O_1) = TxN(O_3) \neq TxN(O_2)$

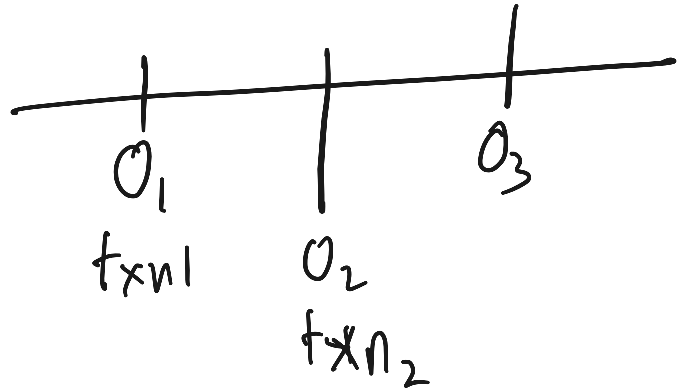
    $\forall$ $O_1 <_S O_2 \wedge O_2 <_S O_3$ &

    $O_1 <_S O_3$

Non-blocking?



Local?



Observe

Linearizable $\Rightarrow$ Seq Cst

Why? - Processes are sequential

$\Rightarrow$ For any history H, process P

$$<_P \subseteq <_H \subseteq <_S$$

LINEARIZABILITY $\Longleftarrow$ STRICT SERIALIZABILITY

Notion of stronger/weaker consistency model

Implications for the readings?

RIFL

BUILDING LINEARIZABLE SYSTEMS

Why?

- The main reason we read the Herlihy & Wing paper.

- Argument: - Linearizability is easier for programmers to reason about
  ↳ When do effects become visible?

Two Ends of the Spectrum

- Assume you are given a linearizable impl. of an ADT
  - Queue
  - Register — get, set } get: Returns value of last set (or $\perp$ if no set has been called)

Sequential Spec

$$\begin{cases} get() \rightarrow \perp & ✓ \\ set(5)\ set(6)\ get() \rightarrow 6 & ✓ \\ set(5)\ set(6)\ set(7)\ get() \rightarrow 6 & ✗ \end{cases}$$

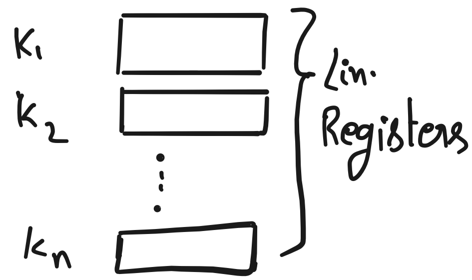
Types of systems

- Collection of independent lin. impl. of ADT

  FOR EXAMPLE, KV Store

  (No txns, only single key operations,

roughly what most people think
Redis provides.)



Sequential spec.
get(k) returns v from
last put(k, v) (or $\perp$)

Observe: Sequential spec. does not impose
any ordering requirement b/w Keys

Just use a set of lin. register

Relation to locality

- Collection of dependent ADTs

Counter
- Init value ϕ
- Inc $\rightarrow$ increment value, return Ok
- Dec $\rightarrow$ decrement value, return Ok
- Get $\rightarrow$ Current value

Maybe track value in a lin. register

(get, set)

val 

```
init:
    val.set(0)
get:
    return val.get()

inc:
    c ← val.get()
    val.set(c+1)

dec:
    c ← val.get()
    val.set(c-1)
```

Does this work?

Can we fix?

Practice
- One common approach in distributed system
    - Maintain a totally ordered log of operations } RSM

- Apply operations in logical order }

- Why?
  - Need to replicate data for F.T.
  - Need to ensure replicas agree on order even for CONCURRENT Ops

    [Failures cannot break abstraction]

CRITICALLY ANALYZING RIFL

- Assumptions

  RPCs; Runtime

  - System — Durable Storage ← Rel. md

  - Client

- Importance of exactly once to linearizability?

  Does exactly once delivery ⟹ Linearizability?

Does linearizability ⇒ exactly once delivery?

- What does RIFL Provide?

CAP:

- Why?
  - Need to replicate data for F.T.
  - Need to ensure replicas agree on order even for CONCURRENT Ops

[Failures cannot break abstraction]

Q) Can replicas agree on order of concurrent ops without communicating?

Problem: Remember R_0 cannot distinguish between

- R_1 has failed
- Messages from R_1 are delayed (indef.) } Partition

In the absence of messages from R_1, R_0 has two options

- Wait "Not available"/ Processing might take unbounded time
- Process assuming R_1 has failed

- process, assuming

R_0 & R_1 might disagree on results

But R_0, R_1 are impl. details

↳ Violate linearizability

"Not consistent"