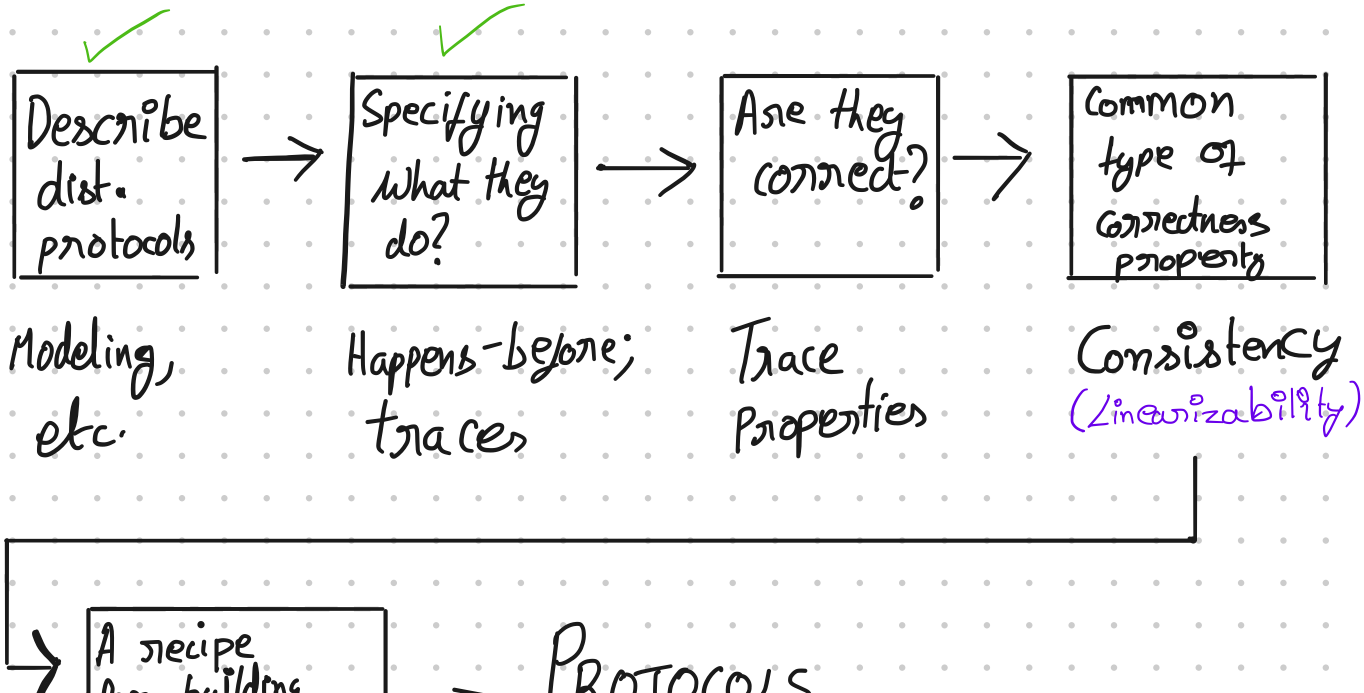
Distributed Systems Lecture 3

Linearizability

Why trace properties; happens before; etc.

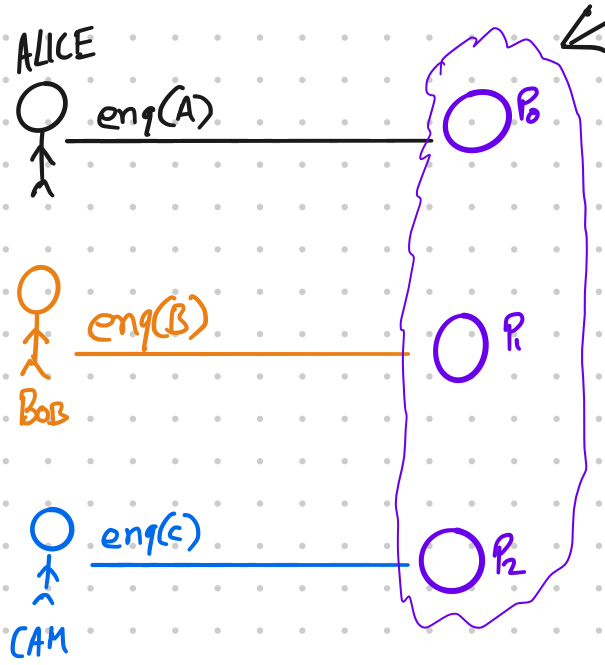
- Came up in last class.
- Today's topic might raise similar questions

Class structure



fault tolerant systems

CONSISTENCY



One system implementing a queue.

ABC
CAB
...
} In what order do operations get performed?

Note: What consistency model is a choice made by the protocol designer / distributed system builder !! A correctness requirement

- Safety or Liveness ?

When making the choice, need to balance (TRADEOFF BETWEEN)

- Ease of use
  - ↳ Do programmers need to be extra cautious when using
  - → Do users understand the behavior

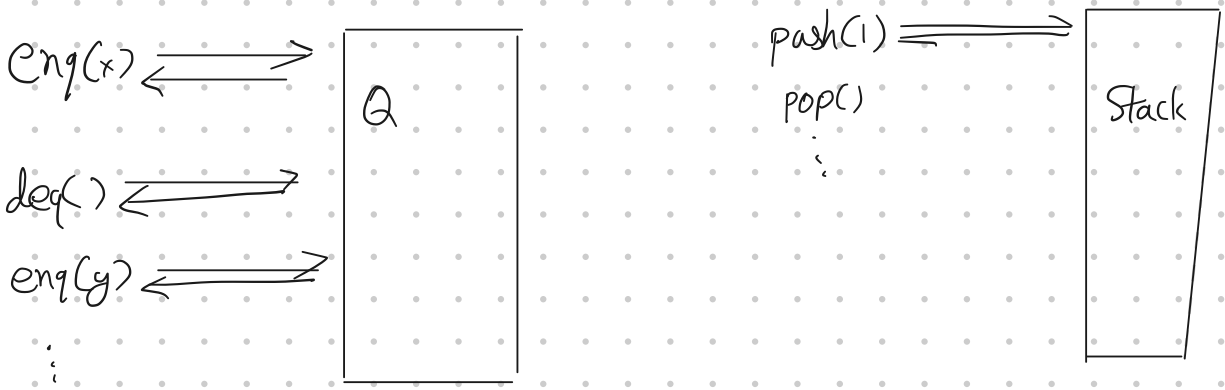
} Tied to each other.

- Performance

- Fault tolerance }

How to think about an ABSTRACT DATA TYPE

Q1. Why is there a DATA TYPE in this DISTRIBUTED Sys. class?



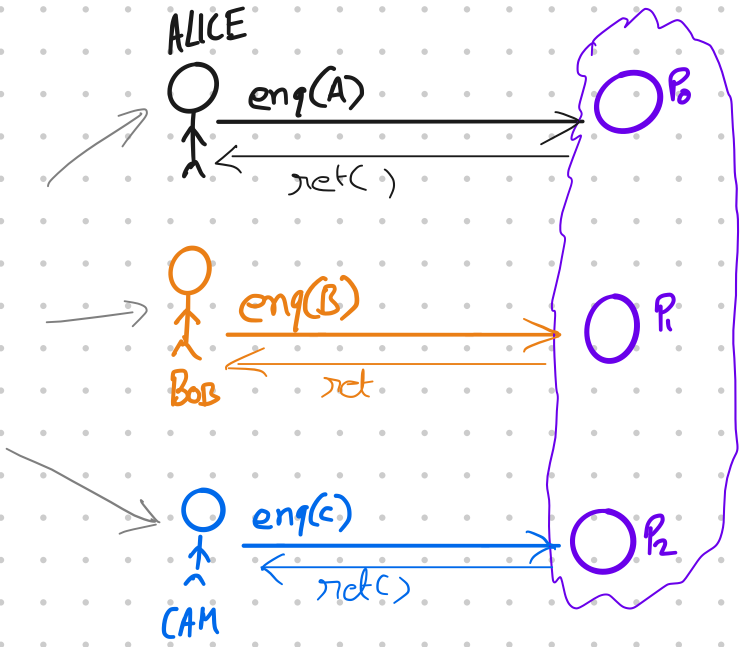
Q2. What is so abstract about this?

NIR SHAVITS ~2011: (older wisdom)

> ...infinitely easier and more intuitive for us humans to specify how abstract data structures behave in a sequential setting, where there are no interleavings.

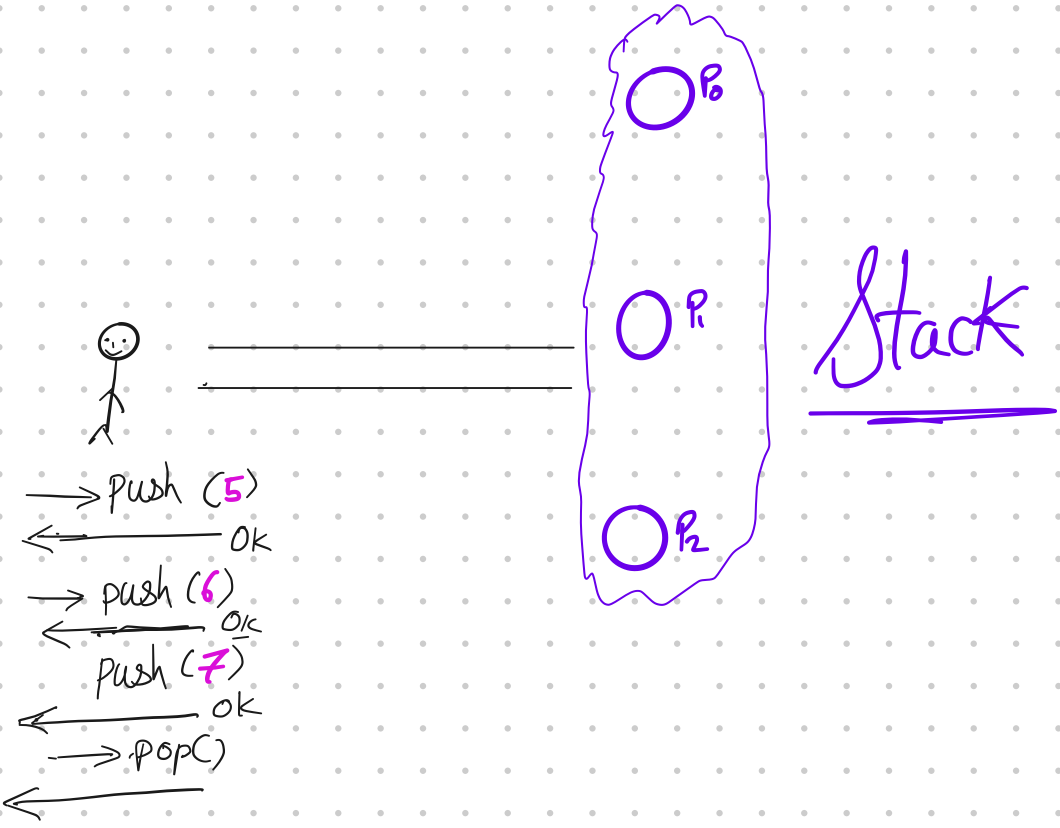
Specify sequential Behavior AND Mapping from concurrent to sequential.

Note: What we are FOCUSED ON WHAT CONCURRENT USERS observe.



Sequential Specification

- Behavior without concurrency



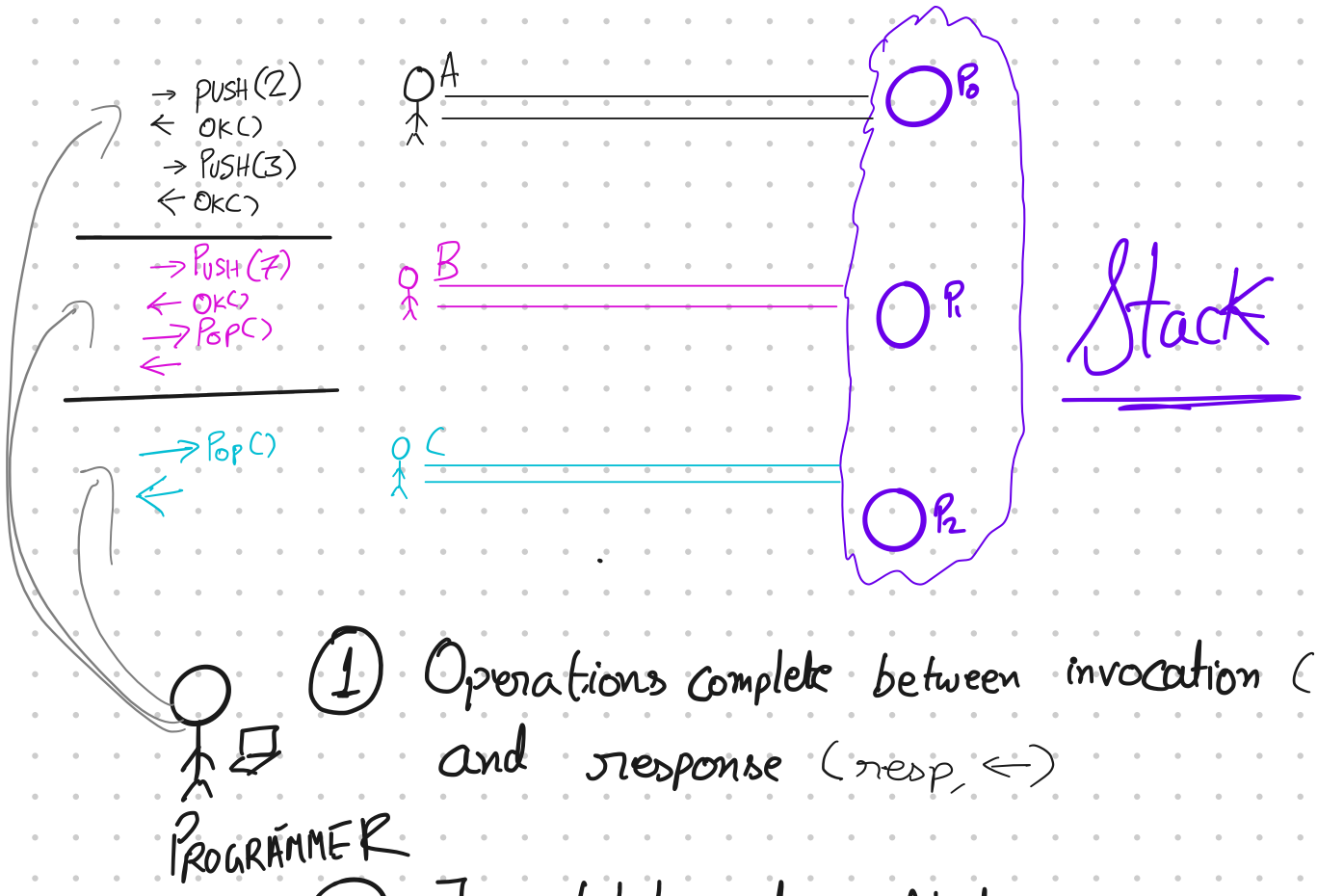
① Exactly one operation at a time.

② Wait for response before issuing next operation

③ Says what each operation can return.

What we are used to: Queues are FIFO, Stacks are LIFO, Text editors ...

Mapping from Concurrent Execution



(1) Operations complete between invocation (inv, →, ...) and response (resp, ←)

(2) ∃ a total order that

(a) All users can agree on

(b) Is consistent with seq. specification

(c) Is consistent with invo. order

Correct might have error

Linearizability

inv ret

Now more formally

(a) Notation from paper

object operation process : Invocation

object OK(value) process : Return

Remember, a process has at most 1 operation in progress at a time.

(b) $<_H$ PARTIAL ORDER

$op_1 <_H op_2 \Rightarrow$ Return op_1 before invocation op_2

Operations complete between invocation (inv, $\rightarrow$, ...) and response (resp, $\leftarrow$)

$\hookrightarrow$ op_2 should observe effects from op_1

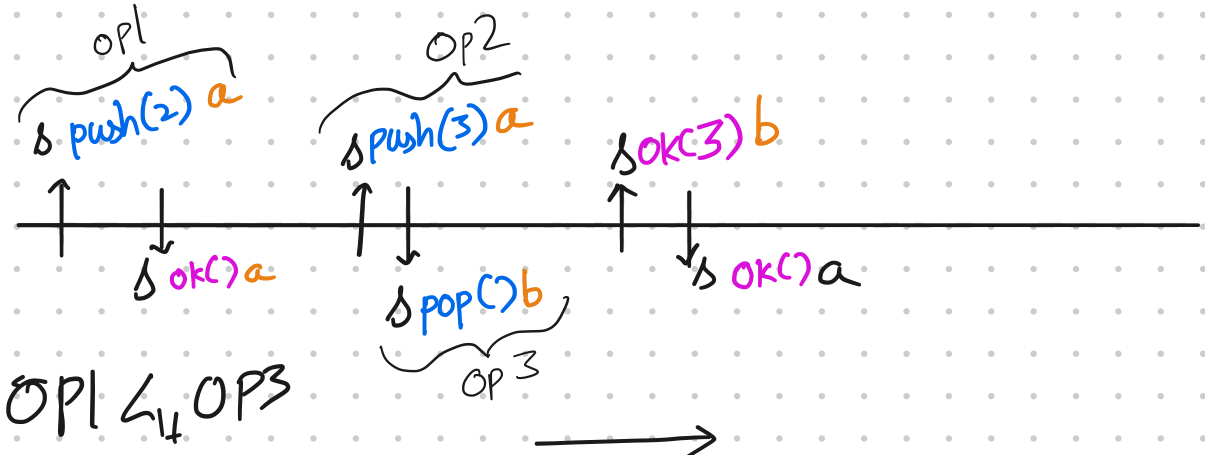
But remember, the total order(s) we will look for are SUPERSETS of $<_H$

Observe

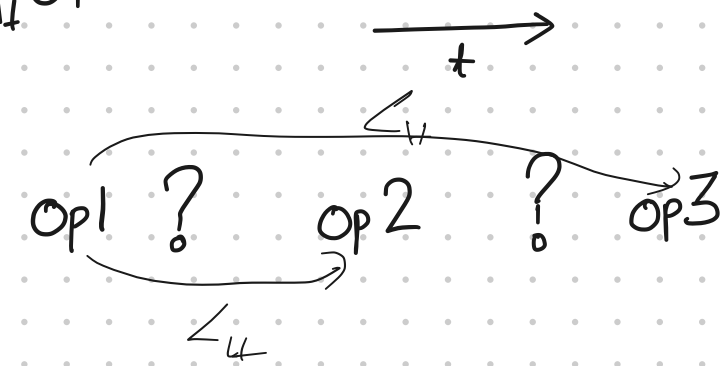
$<_H$ orders all operations issued by

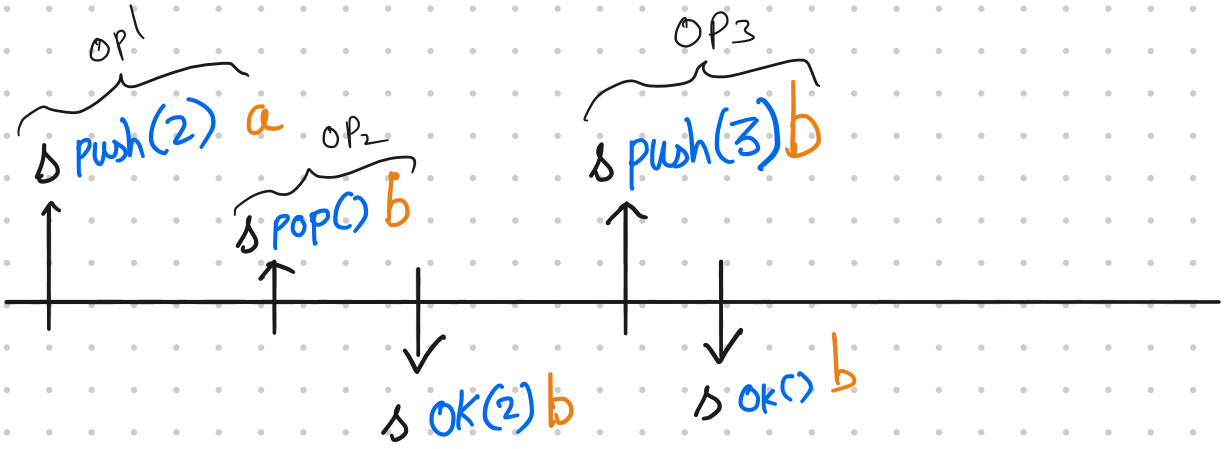
the same process.

Why?



op1 <H op3





op1 op2 op3

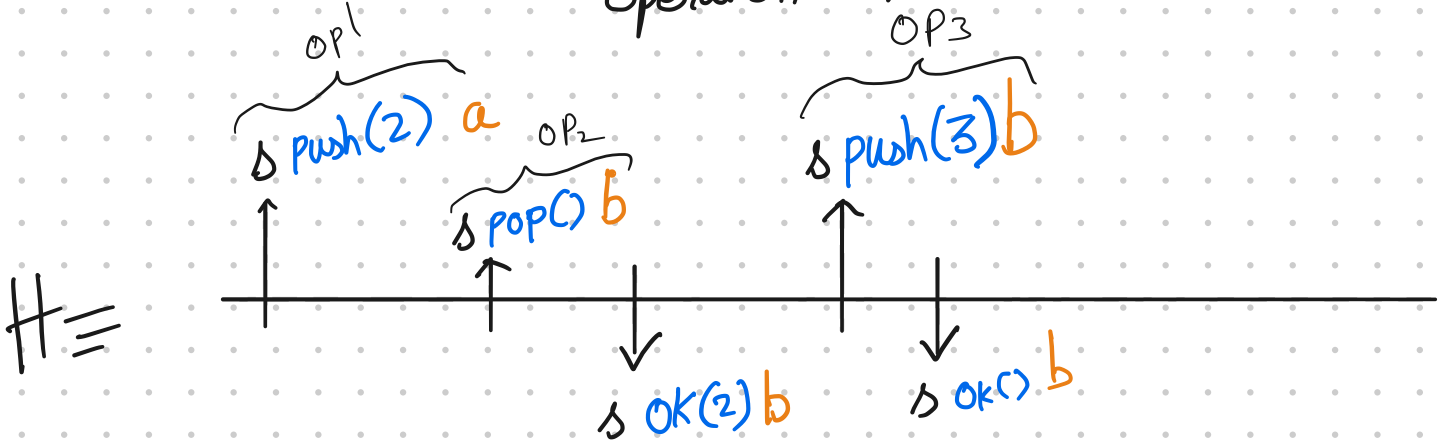
© History H linearizable if and only if

$\exists$ H that extends H (by ∅ or ... returns) s.t.

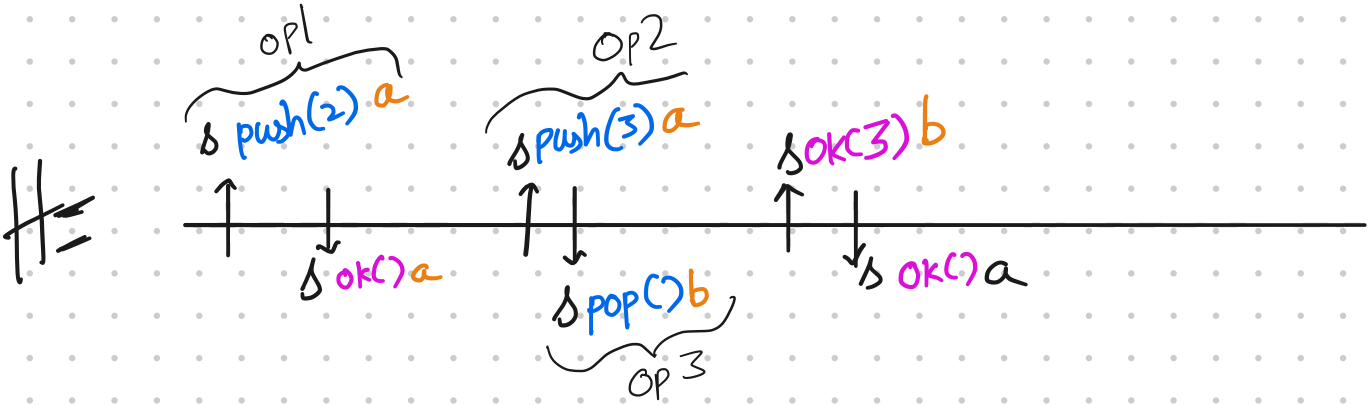
there is total order $<_S$ where $<_{complete(H')} \subseteq <_S$

& $<_S$ meets SEQUENTIAL SPEC.

- complete(H): H with all incomplete/inprogress operations removed



Operations in COMPLETE(H)?



Operations in COMPLETE(H)?

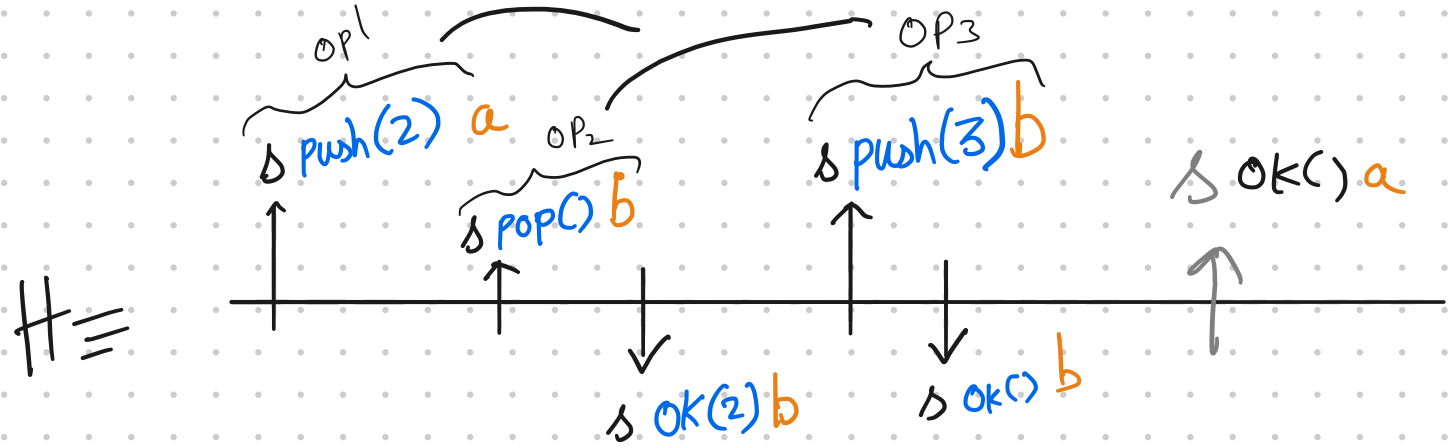
- H is legal if and only if

History H linearizable if

$\exists$ H' that extends H (by $\emptyset$ or more returns) s.t.

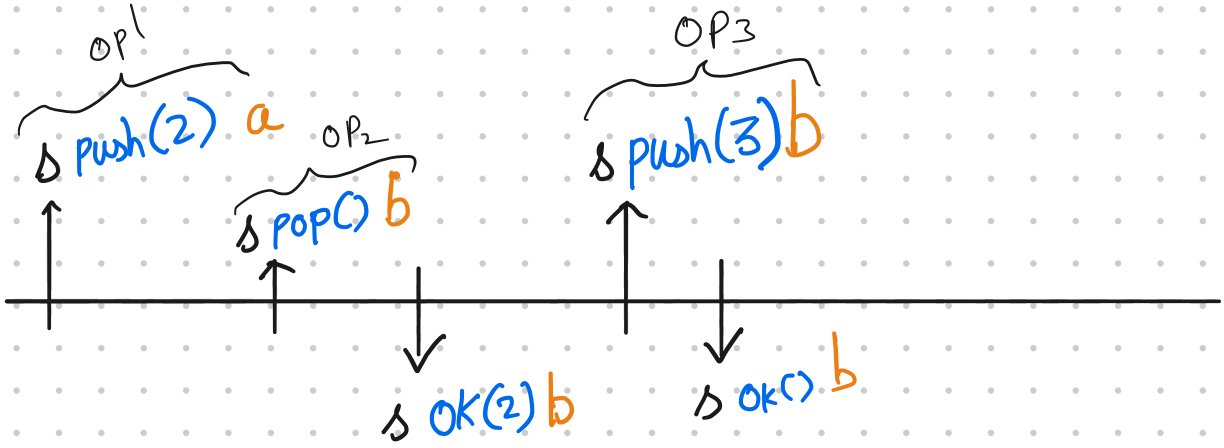
there is total order $<_S$ where $<_{complete(H')} \subseteq <_S$

& $<_S$ meets SEQUENTIAL SPEC

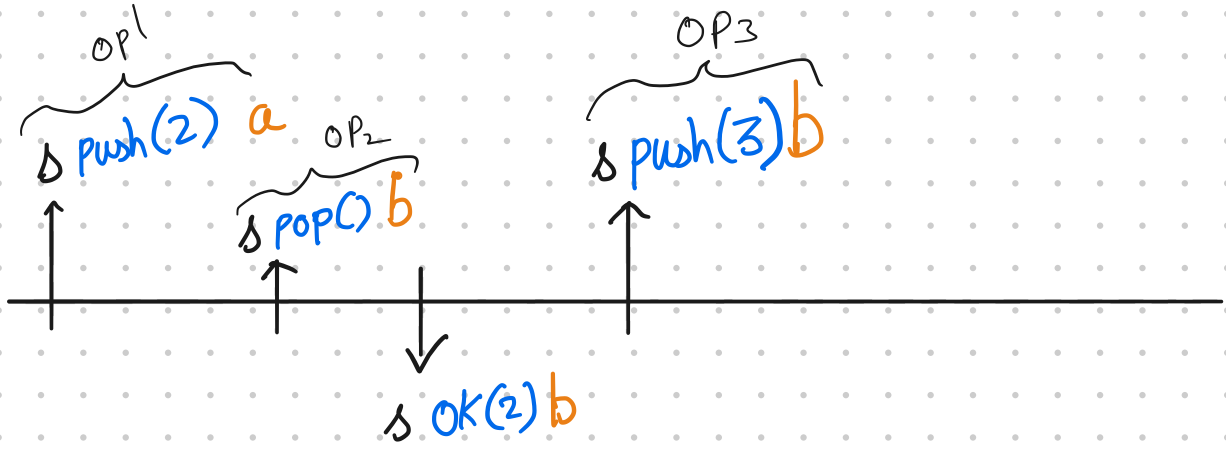


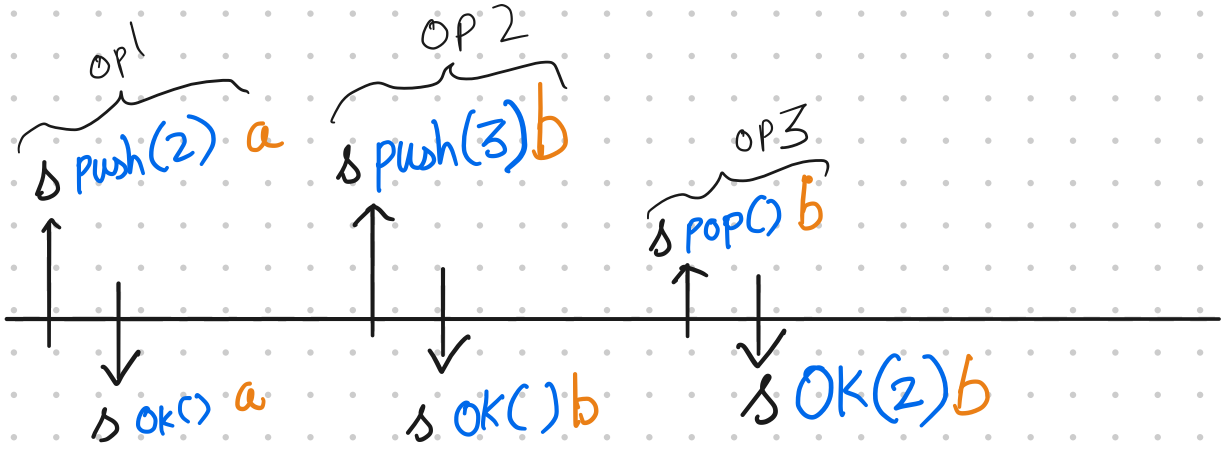
$<_S = op_1 <_S op_2 <_S op_3$

Is H a linearizable history for a stack?



$H \rightarrow H'$ $H' \rightarrow$ Complete (H')





Ok. But Are There Other Options?

Seq. consistency

① ~~Operations complete between invocation (inv, →, ...) and response (→ resp, ←)~~

② ∃ a total order that

(a) All users can agree on

(b) Is consistent with seq. specification

$<_p$: Order of ops from process p

Equivalently $<_{H|p}$ [$H|p$: Only invocations & returns for process p]

History H seq cst if and only if

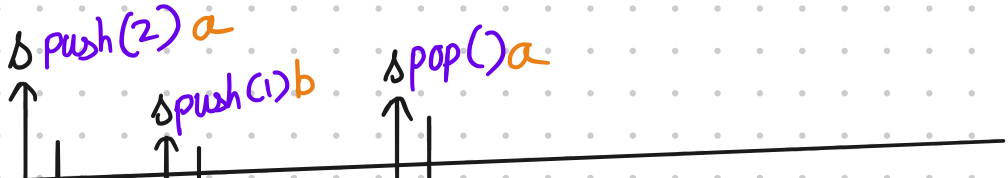
$\exists$ H' that extends H (by $\emptyset$ or more returns) s.t.

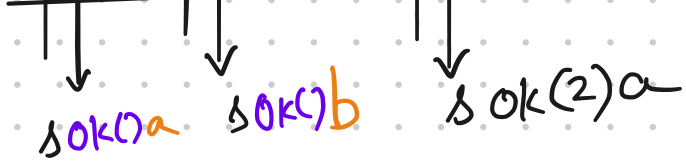
there is total order

$<_S$ where $<_{\text{complete}(H'|p)} \subseteq <_S \forall_p$

& $<_S$ meets SEQUENTIAL SPEC

- Common: Strongest guarantee provided by C++, Rust, etc.
- Might appear that operations move in time





- Linearizable?

- Seq cst ?

Properties

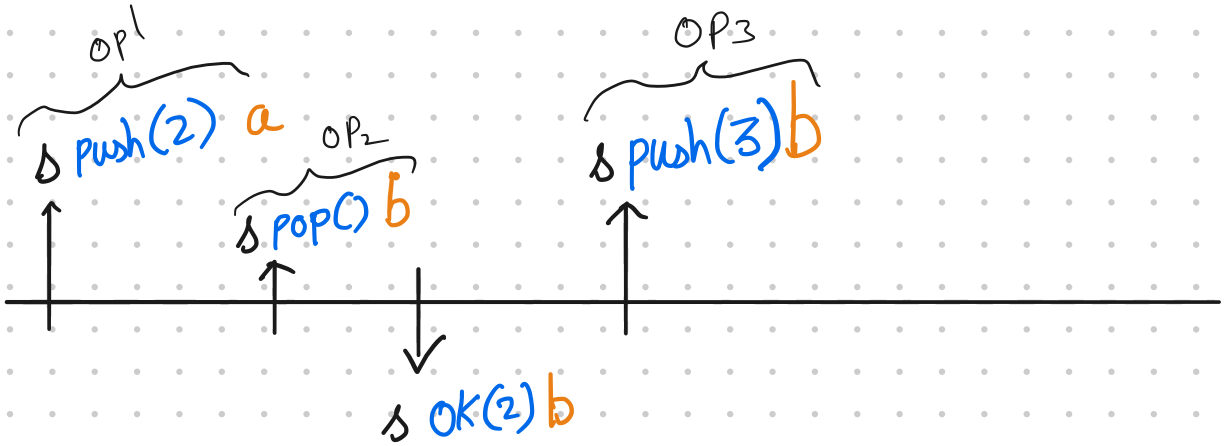
Non-Blocking

If history H is linearizable and

H contains incomplete op O then

$\exists$ response O_R to O such that

$H; O_R$ is linearizable



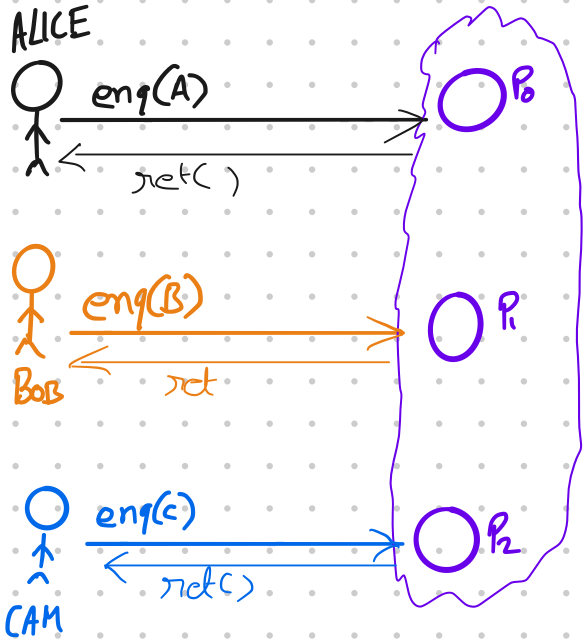
Two incomplete operations: op1, op3

D. . .

responses.

Why is this useful?

- Can delay operation execution without violating consistency



Local

H has operations for two objects x and y

H linearizable $\iff$ $H|_x$ linearizable & $H|_y$ linearizable

H/y

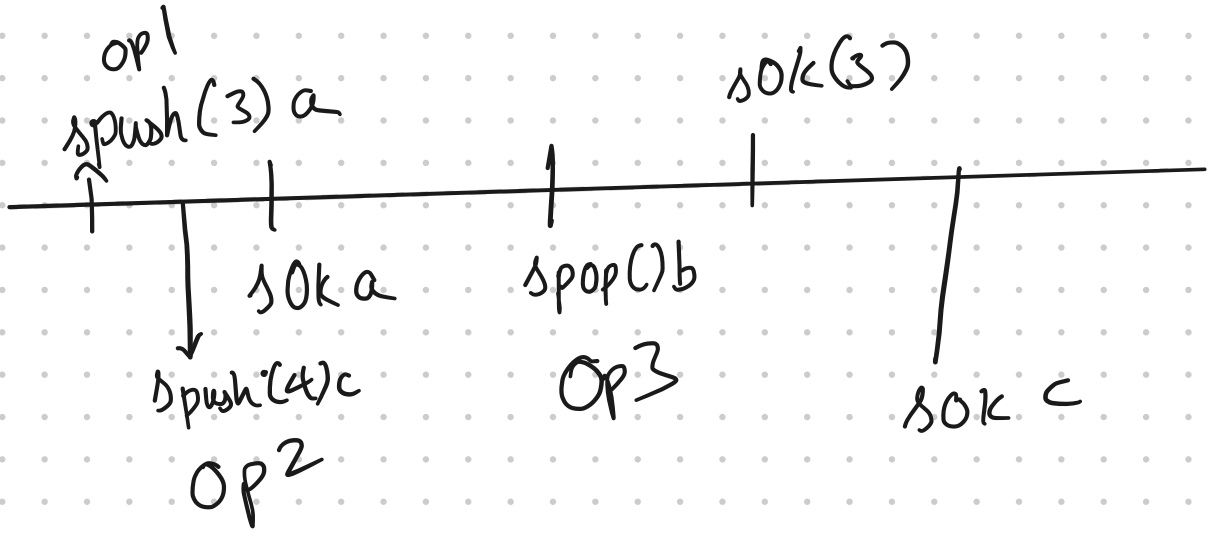
What this means?

- The linearization of H (S or $<_S$) matches x's seq spec $\triangleright$ y's seq spec.

- Cannot say anything about a data structure with a different seq. spec that combines x Δ_y

struct stack_with_len {
  stack x
  int len
} | push
  | pop
  | get_length

- Why useful?


op1
spush(3) a
sOk(3)
sOk a
spop()b
spush(4)c
Op3
sOk c
Op2