THE PIT AND THE PENDULUM

Lorenzo Alvisi
Cornell University

A CLASSIC HORROR STORY

Ease of Programming

Performance

A CLASSIC HORROR STORY

Ease of Programming

Performance

A CLASSIC HORROR STORY

Ease of Programming

Performance

CONCURRENCY

CORRECTNESS

- Safety
  - "nothing bad happens"

No two processes in the critical section at the same time

CORRECTNESS

- Safety
  - "nothing bad happens"
- Liveness
  - "something good eventually happens"

No two processes in the critical section at the same time

A process that wants to enter the critical section will be able to do so eventually

SEQUENTIAL OBJECTS

Thanks to Maurice Herlihy
"The Art of Multiprocessor Programming"

- Each object has a **state**
  - Register: the value it stores
  - Queue: the sequence of objects it holds

SEQUENTIAL OBJECTS

Thanks to Maurice Herlihy
"The Art of Multiprocessor Programming"

- Each object has a **state**
  - Register: the value it stores
  - Queue: the sequence of objects it holds
- Each object has a set of **methods**
  - Register: Read/Write
  - Queue: Enq/Deq/Head

SEQUENTIAL SPECIFICATIONS

Thanks to Maurice Herlihy

- **If** (precondition)
  - the object is in such-and-such-state before method is called
- **Then** (postcondition)
  - the method will return a particular value
  - or throw a particular exception
- **and** (postcondition continued)
  - the object will be in some other state when method returns

PRE AND POST CONDITIONS FOR DEQ

Thanks to Maurice Herlihy

- Precondition
  - Queue is non-empty

- Postcondition
  - Returns first item in queue
- Postcondition
  - First item no longer in queue

PRE AND POST CONDITIONS FOR DEQ

Thanks to Maurice Herlihy

- Precondition
  - Queue is non-empty

- Postcondition
  - Returns first item in queue

- Postcondition
  - First item no longer in queue

PRE AND POST CONDITIONS FOR DEQ

Thanks to Maurice Herlihy

- Precondition
  - Queue is empty

- Postcondition
  - Throws Empty exception
- Postcondition
  - Queue state unchanged

SEQUENTIAL SPECIFICATIONS ARE AWESOME

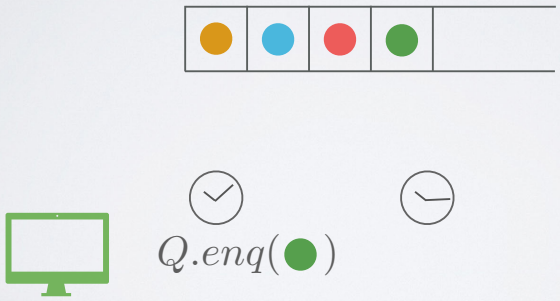
So is Maurice

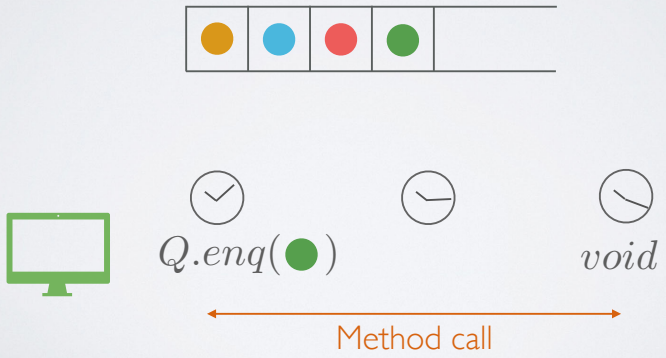
- Interactions among methods captured by side-effects on object state
  - State **between** method calls is meaningful
- Documentation size linear in the number of methods
  - Separation of concerns: each method described in isolation
- Can add new methods
  - Without changing description of old methods

WHAT ABOUT CONCURRENT SPECIFICATIONS?

- Methods?
- Documentation?
- Adding new methods?

METHODS TAKE TIME





METHODS TAKE TIME

- If you are Sequential
  - Really? Never noticed!
- …but if you are Concurrent
  - Method call is not an event
  - Method call is an interval

★ Concurrent method calls overlap! ★

WHAT DOES IT MEAN FOR CORRECTNESS?

- Sequential
  - Object needs meaningful states only between method calls
- Concurrent
  - Because method calls overlap, object may never be between method calls

WHAT DOES IT MEAN FOR CORRECTNESS?

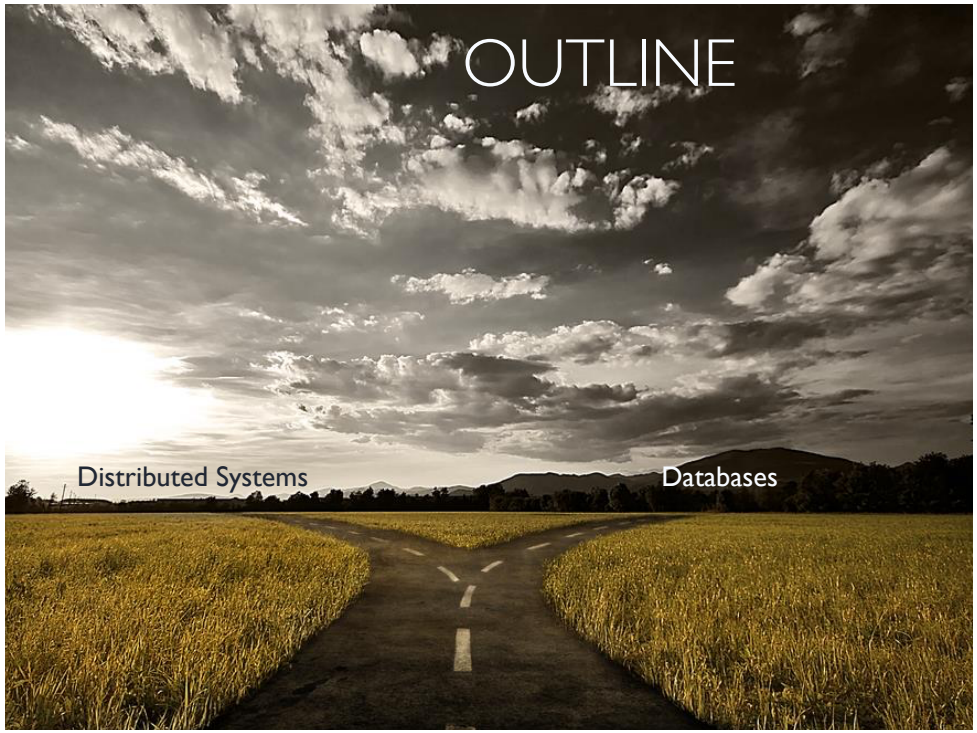
- Sequential
  - Each method described in isolation
- Concurrent
  - Must consider all possible interactions between concurrent calls
    - What if two enq() overlap?
    - What if enq() and deq() overlap?

WHAT DOES IT MEAN FOR CORRECTNESS?

- Sequential
  - New methods do not affect existing methods
- Concurrent
  - **Everything** can potentially interact **with everything else**

WHAT ABOUT DATABASES?

TRANSACTIONS TAKE TIME



REGISTERS

- Sequential specification
  - A read returns the result of the latest completed write

REGISTERS

- Sequential specification
  - A read returns the result of the latest completed write
- What if reads and writes can be concurrent?

REGISTERS

- Sequential specification
  - A read returns the result of the latest completed write
- What if reads and writes can be concurrent?
  - A read not concurrent with a write returns the result of the latest completed write

REGISTERS

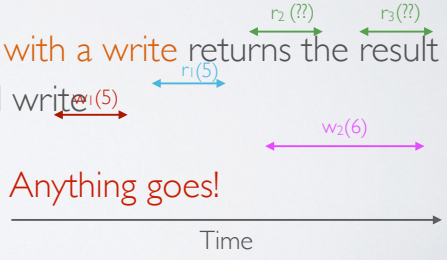
- Sequential specification
  - A read returns the result of the latest completed write
- What if reads and writes can be concurrent?
  - A read not concurrent with a write returns the result of the latest completed write
- And if they are concurrent?

SAFE REGISTERS

- Sequential specification
  - A read returns the result of the latest completed write
- What if reads and writes can be concurrent?
  - A read not concurrent with a write returns the result of the latest completed write
- And if they are concurrent? Anything goes!

SAFE REGISTERS

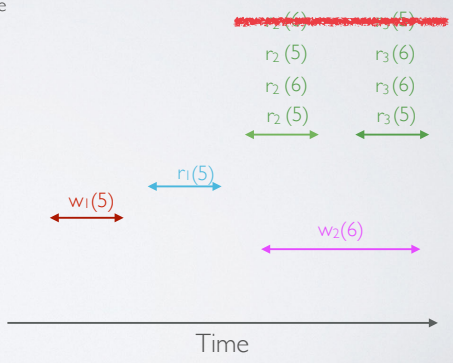
- Sequential specification
  - A read returns the result of the latest completed write
- What if reads and writes can be concurrent?
  - A read not concurrent with a write returns the result of the latest completed write
- And if they are concurrent? Anything goes!
  -



REGULAR REGISTERS

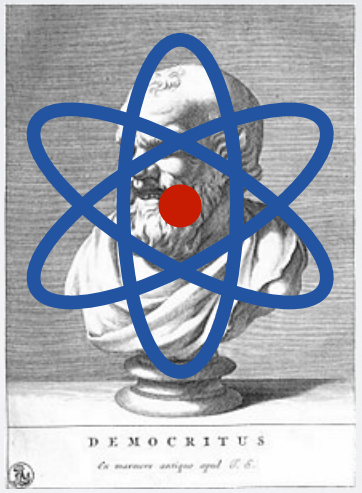
- Sequential specification
  - A read returns the result of the latest completed write
- What if reads and writes can be concurrent?
  - A read not concurrent with a write returns the result of the latest completed write
- And if they are concurrent?

  A read overlapping with a write returns either the old or the new value!



CAN WE DO BETTER?

ATOMIC REGISTERS



A LITTLE MORE FORMALLY

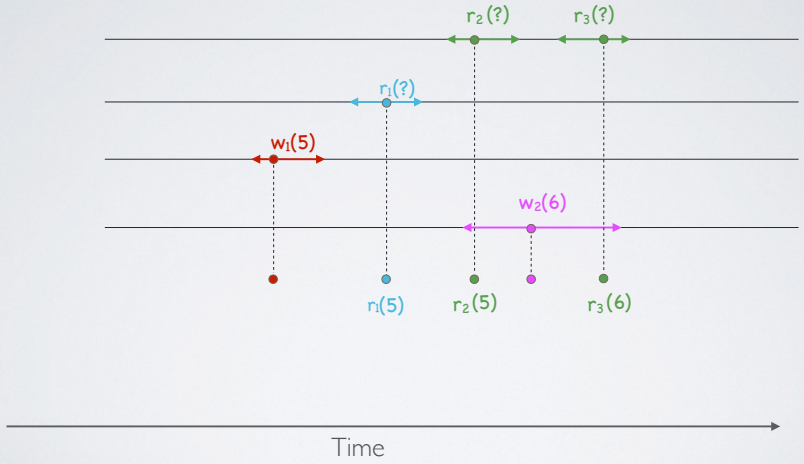
- Execution of a concurrent object modeled by a history
  - a finite sequence of operation invocation and responses
- A history H is sequential if:
  - the first event of H is an invocation
  - each invocation (but possibly the last) is immediately followed by a matching response, followed by a matching invocation
- A history that is not sequential is concurrent

LINEARIZABILITY

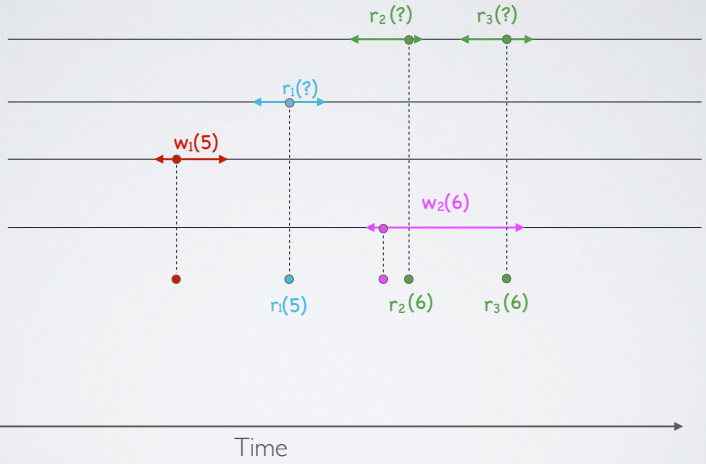
(HERLIHY AND WING 1990)

- A history H is linearizable if there exists a permutation π of the operations in H such that
  - For each object o, the sub-history $\pi|o$ respects the sequential specification of o
  - If the response of operation o_1 occurs in H before the invocation of operation o_2, then o_1 appears before o_2 in π
    - in other words, π respects the real-time ordering of non-overlapping operations

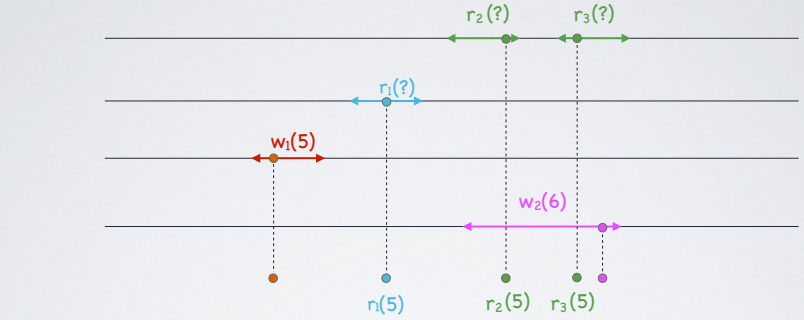
EXAMPLE: REGISTERS



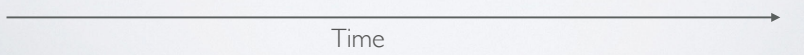
EXAMPLE: REGISTERS



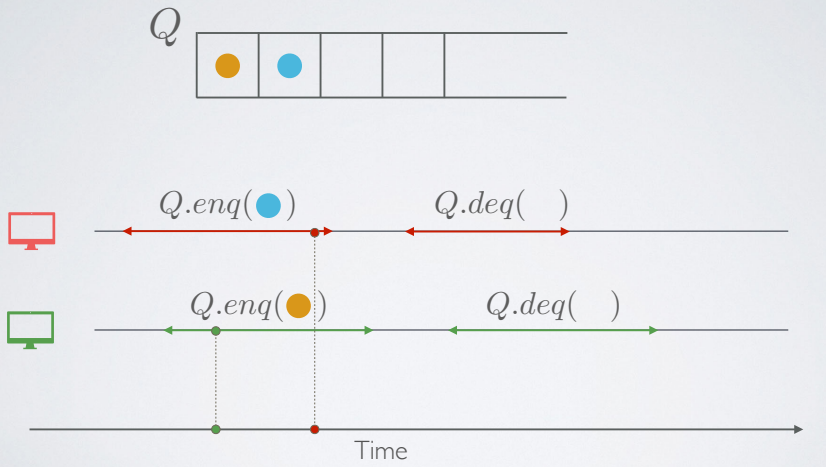
EXAMPLE: REGISTERS



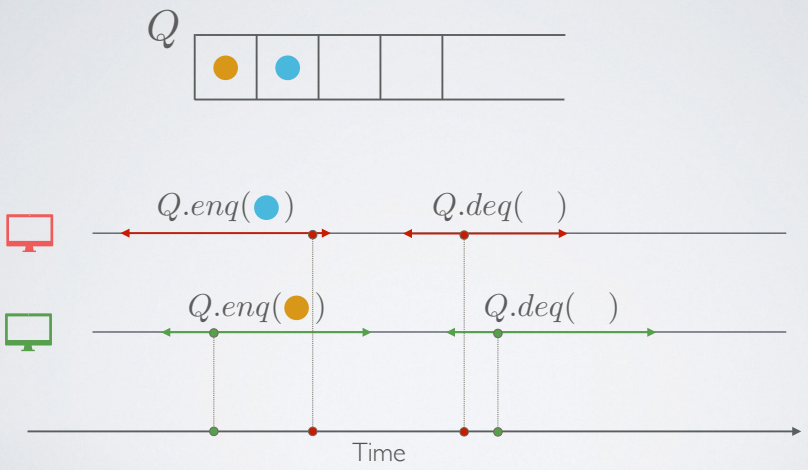
Linearizable registers are called **atomic**



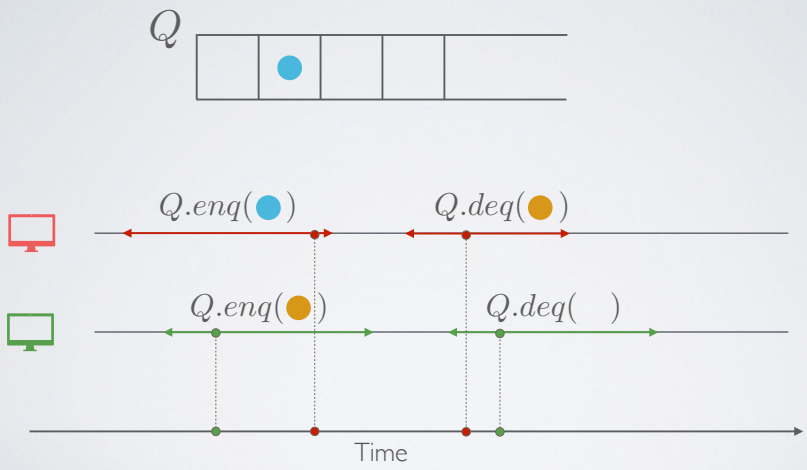
LINEARIZABLE QUEUE?



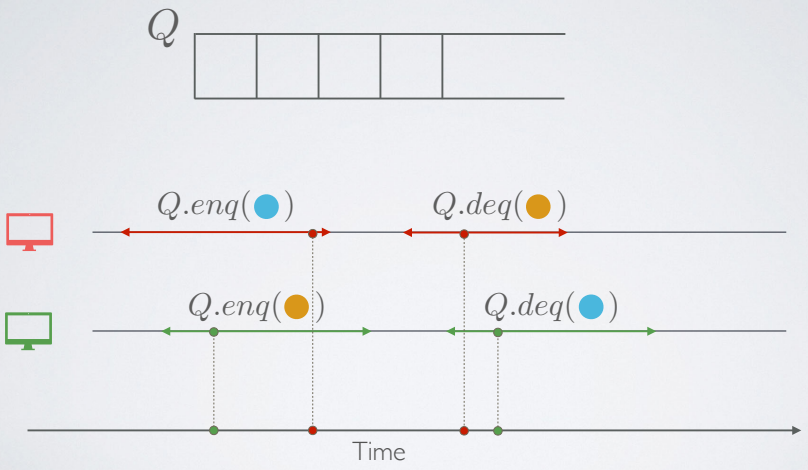
LINEARIZABLE QUEUE?



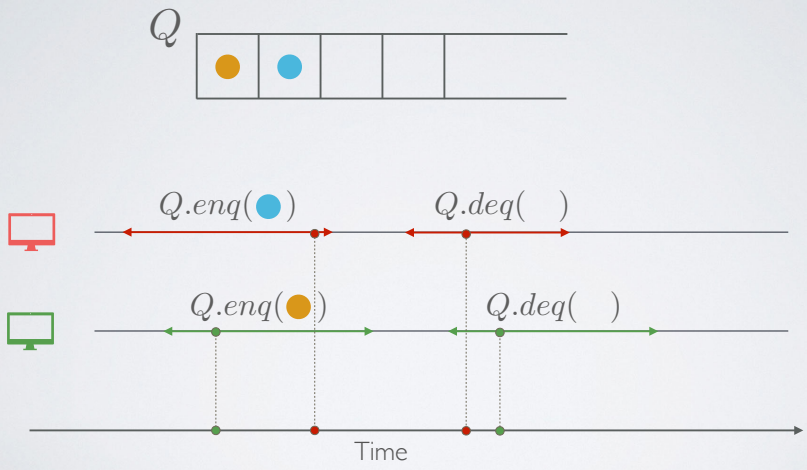
LINEARIZABLE QUEUE?



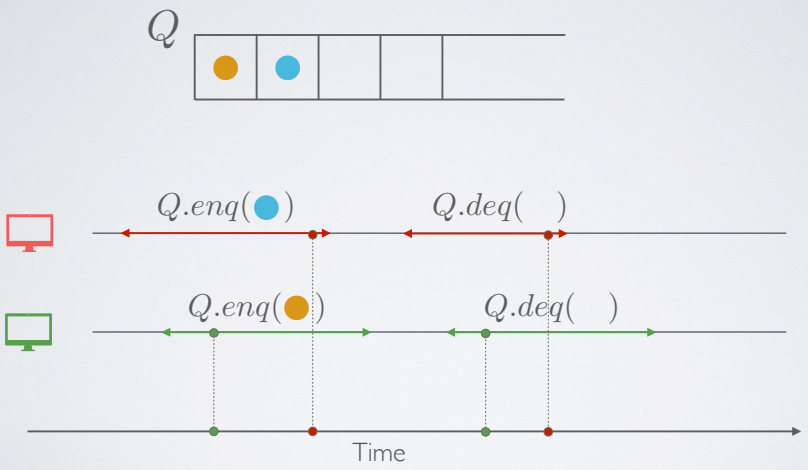
LINEARIZABLE QUEUE?



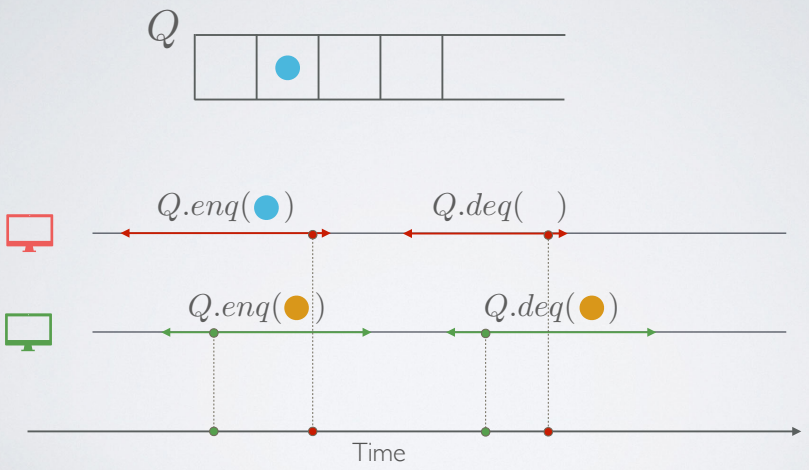
LINEARIZABLE QUEUE?



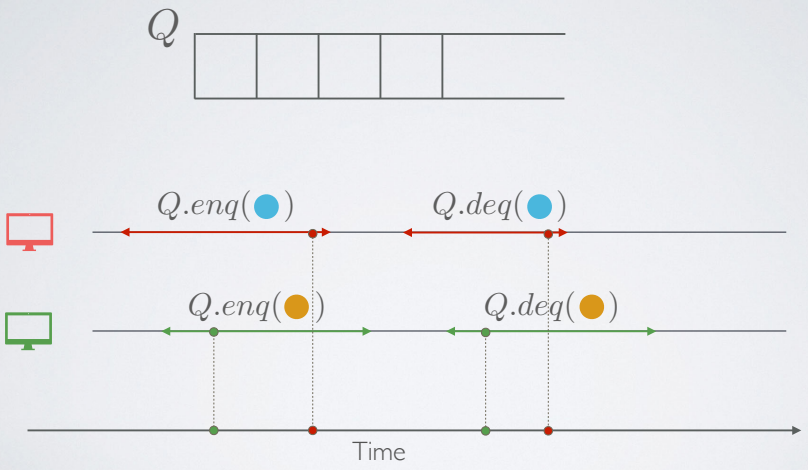
LINEARIZABLE QUEUE?



LINEARIZABLE QUEUE?



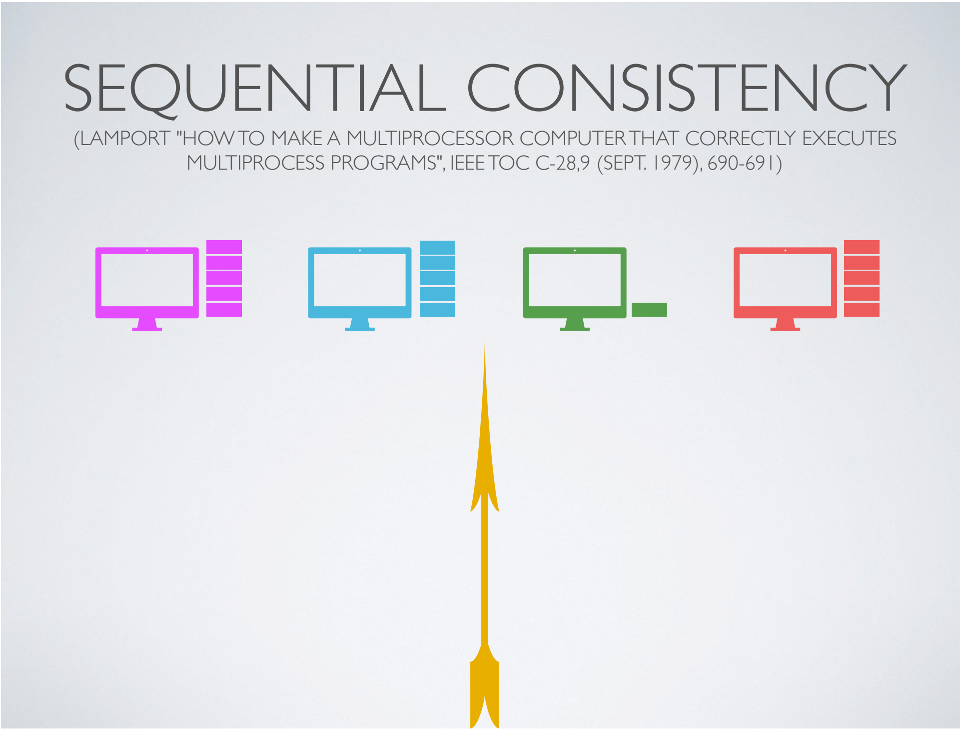
LINEARIZABLE QUEUE?

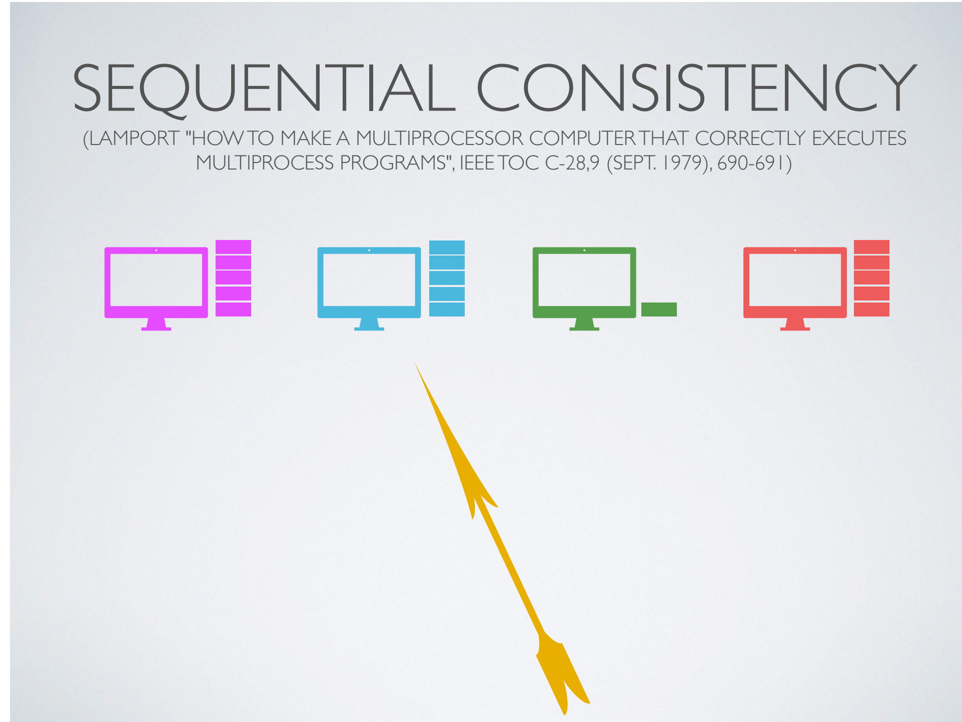


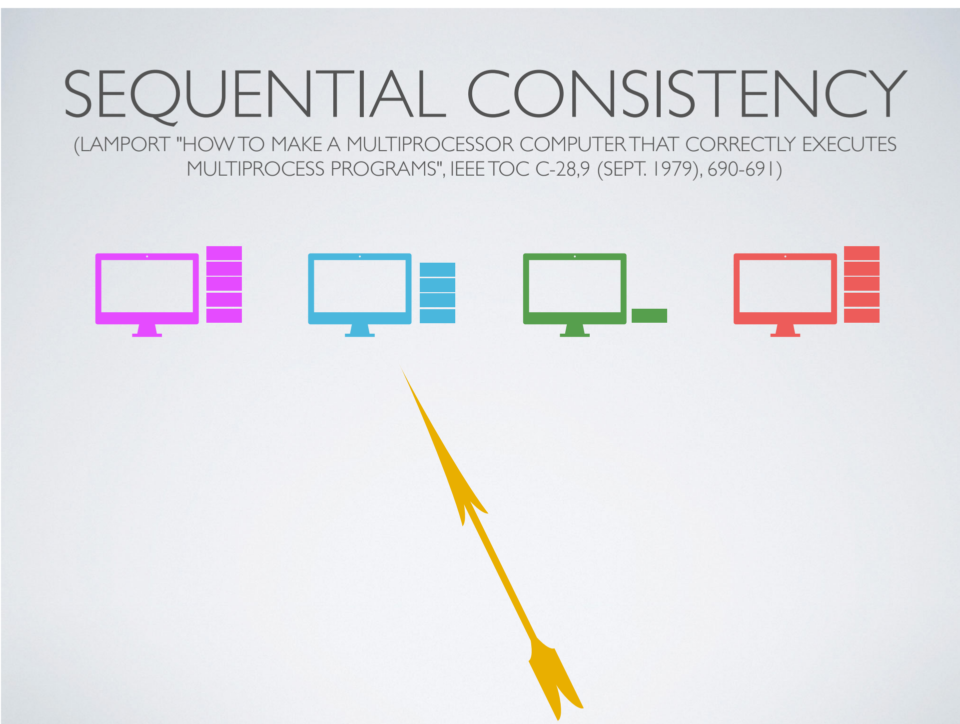
SEQUENTIAL CONSISTENCY

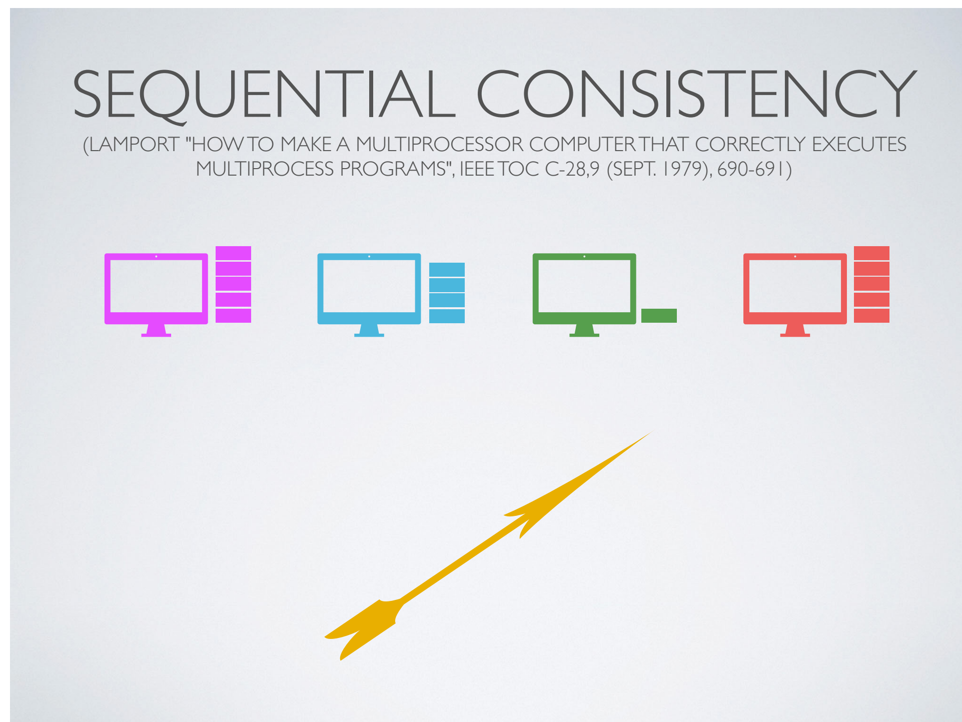
(LAMPORT "HOW TO MAKE A MULTIPROCESSOR COMPUTER THAT CORRECTLY EXECUTES MULTIPROCESS PROGRAMS", IEEE TOC C-28,9 (SEPT. 1979), 690-691)

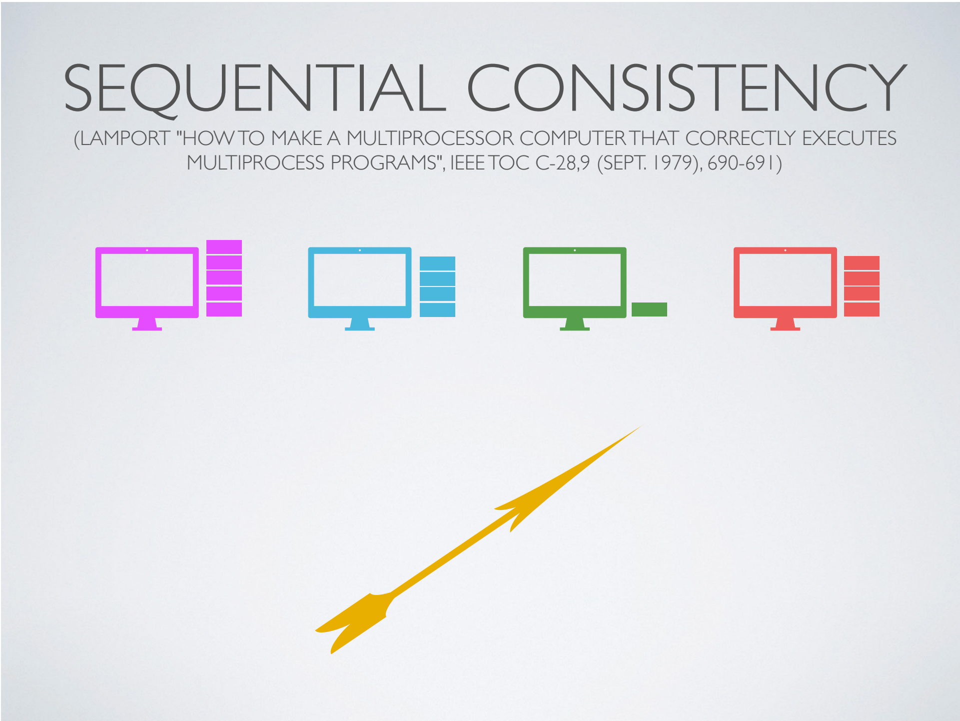
- A history H is sequentially consistent if there exists a permutation π of the operations in H such that
  - $\pi|o$ respects the sequential specification of each object o
  - If the response for operation o_1 at p_i occurs in H before the invocation for operation o_2 at p_i, then o_1 appears before o_2 in π

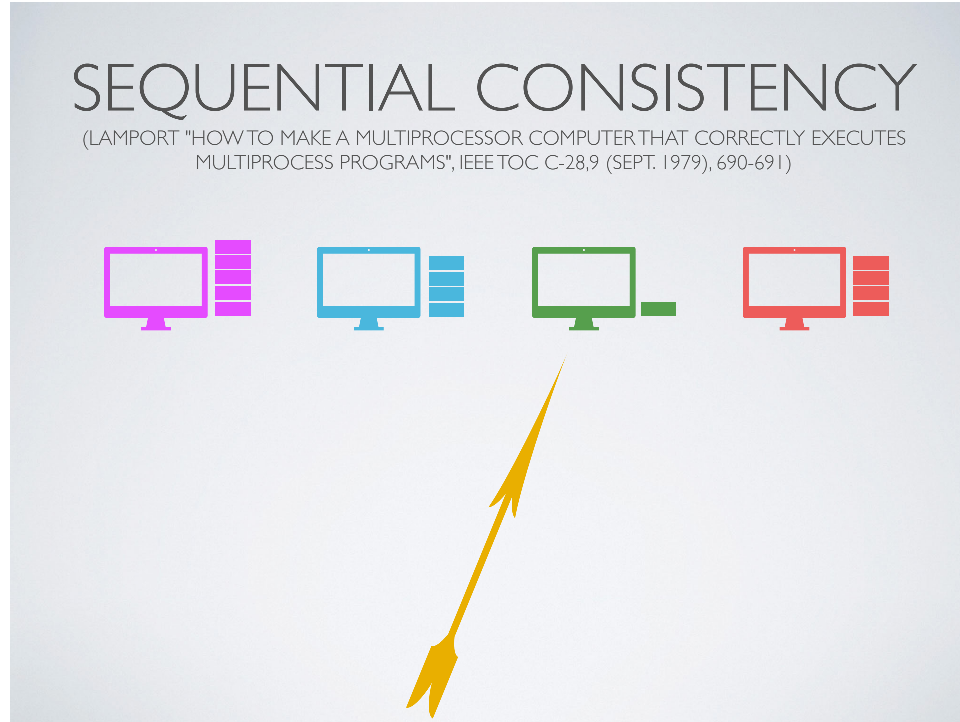

SEQUENTIAL CONSISTENCY
(LAMPORT "HOW TO MAKE A MULTIPROCESSOR COMPUTER THAT CORRECTLY EXECUTES MULTIPROCESS PROGRAMS", IEEE TOC C-28,9 (SEPT. 1979), 690-691)

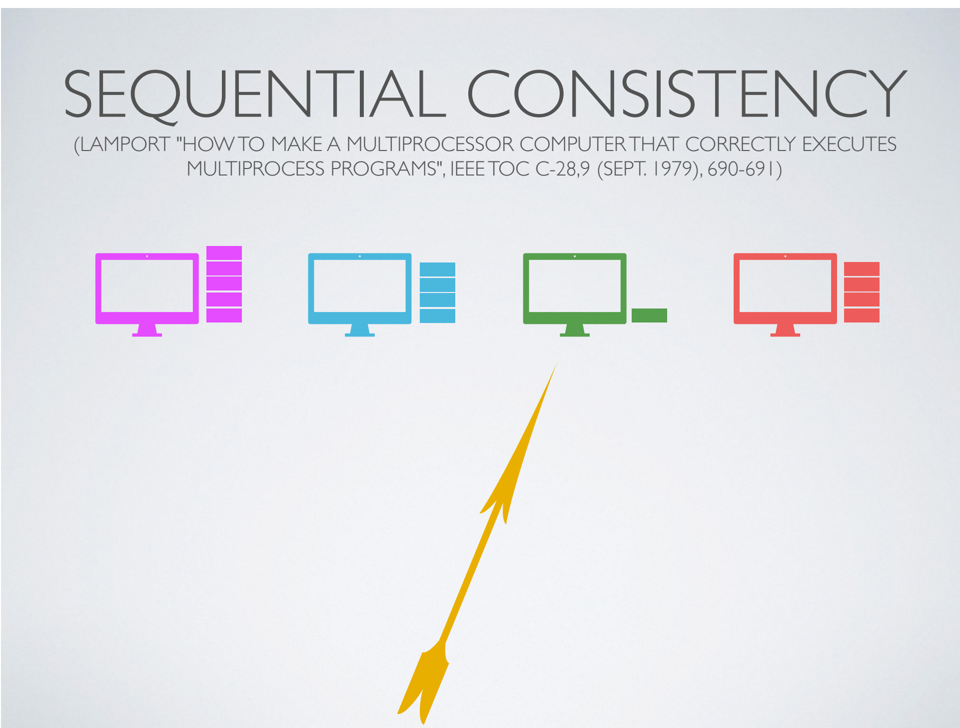

SEQUENTIAL CONSISTENCY
(LAMPORT "HOW TO MAKE A MULTIPROCESSOR COMPUTER THAT CORRECTLY EXECUTES MULTIPROCESS PROGRAMS", IEEE TOC C-28,9 (SEPT. 1979), 690-691)

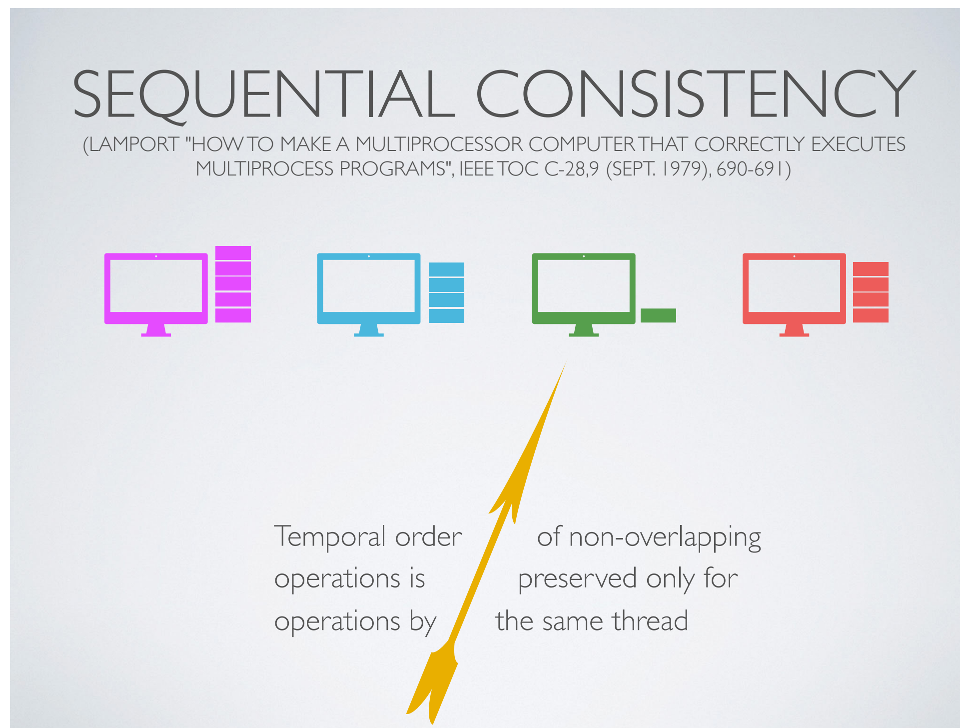

SEQUENTIAL CONSISTENCY
(LAMPORT "HOW TO MAKE A MULTIPROCESSOR COMPUTER THAT CORRECTLY EXECUTES MULTIPROCESS PROGRAMS", IEEE TOC C-28,9 (SEPT. 1979), 690-691)


SEQUENTIAL CONSISTENCY
(LAMPORT "HOW TO MAKE A MULTIPROCESSOR COMPUTER THAT CORRECTLY EXECUTES MULTIPROCESS PROGRAMS", IEEE TOC C-28,9 (SEPT. 1979), 690-691)

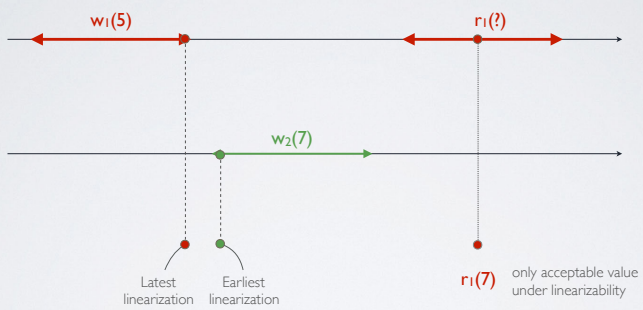

SEQUENTIAL CONSISTENCY
(LAMPORT "HOW TO MAKE A MULTIPROCESSOR COMPUTER THAT CORRECTLY EXECUTES MULTIPROCESS PROGRAMS", IEEE TOC C-28,9 (SEPT. 1979), 690-691)


SEQUENTIAL CONSISTENCY
(LAMPORT "HOW TO MAKE A MULTIPROCESSOR COMPUTER THAT CORRECTLY EXECUTES MULTIPROCESS PROGRAMS", IEEE TOC C-28,9 (SEPT. 1979), 690-691)


SEQUENTIAL CONSISTENCY
(LAMPORT "HOW TO MAKE A MULTIPROCESSOR COMPUTER THAT CORRECTLY EXECUTES MULTIPROCESS PROGRAMS", IEEE TOC C-28,9 (SEPT. 1979), 690-691)

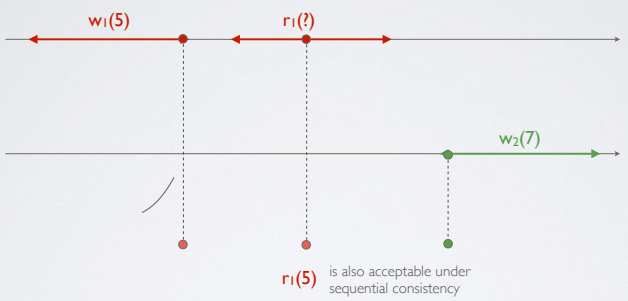

SEQUENTIAL CONSISTENCY
(LAMPORT "HOW TO MAKE A MULTIPROCESSOR COMPUTER THAT CORRECTLY EXECUTES MULTIPROCESS PROGRAMS", IEEE TOC C-28,9 (SEPT. 1979), 690-691)
Temporal order of non-overlapping
operations is preserved only for
operations by the same thread

EXAMPLE



but under sequential consistency…

EXAMPLE



…this is a valid permutation of the same history

THINK GLOBAL, ACT LOCAL

- A property P of a concurrent system is

  local

  if the system satisfies P

  whenever each individual object satisfies P

  - Given two objects o_1 and o_2 each satisfying P, the composite object $[o_1, o_2]$ satisfies P

LINEARIZABILITY IS A LOCAL PROPERTY

Theorem

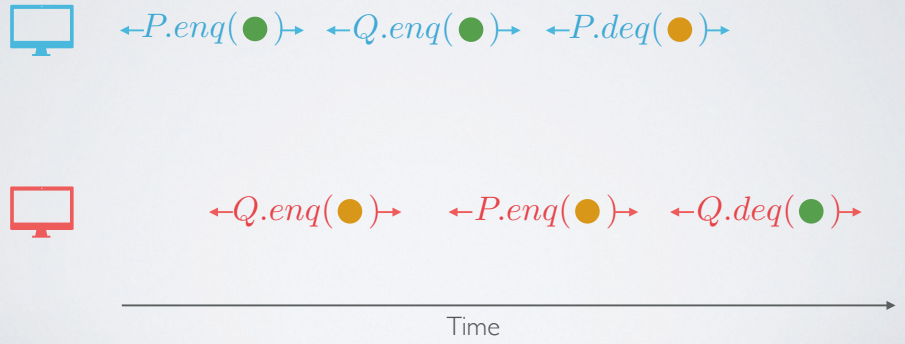
A history H is linearizable iff

for each object o, H restricted to the operations in o is linearizable

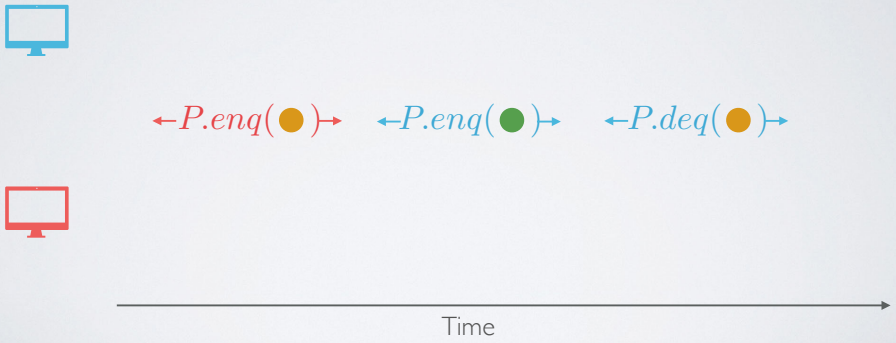
- Because linearizability is a local property, objects can be implemented independently

What about
sequential consistency?

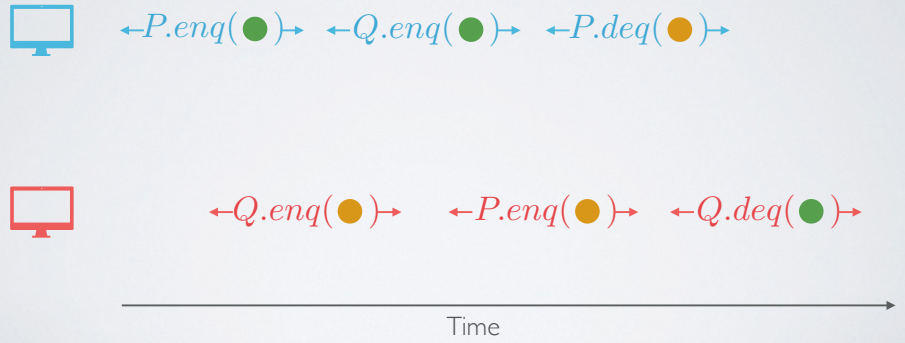
THE CASE OF THE FIFO QUEUES



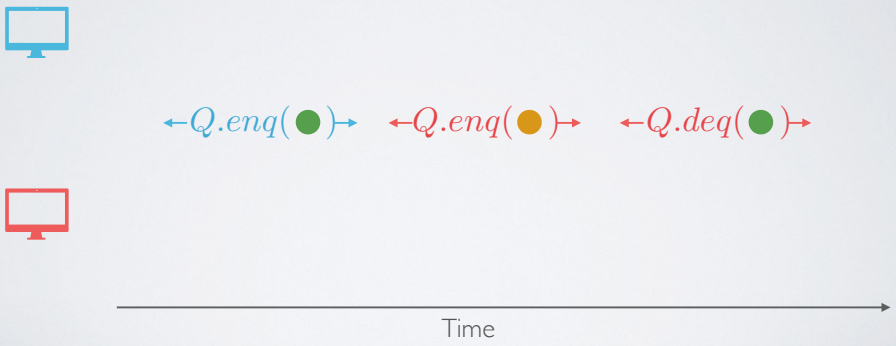
THE CASE OF THE FIFO QUEUES



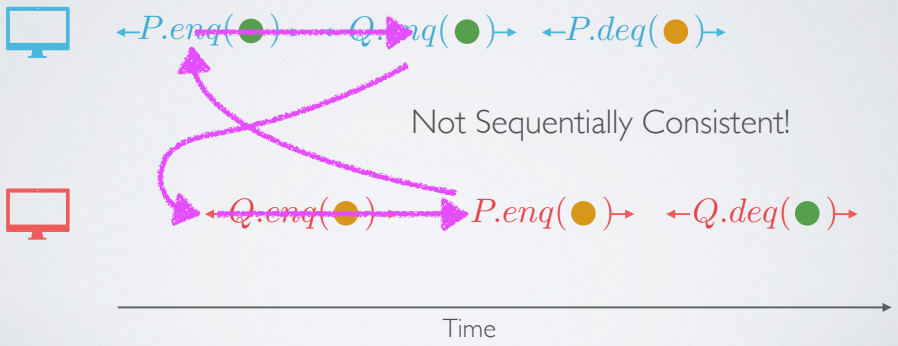
THE CASE OF THE FIFO QUEUES



THE CASE OF THE FIFO QUEUES



THE CASE OF THE FIFO QUEUES



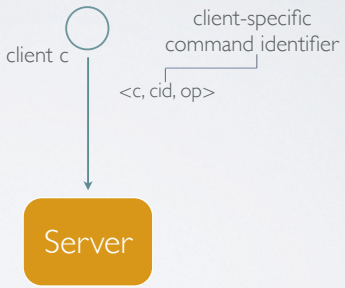
THEOREM

Sequential Consistency is not composable

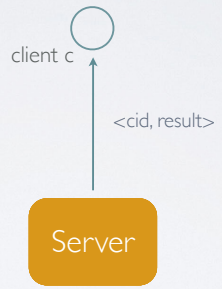
i.e., an execution involving a collection of sequentially consistent objects may not be sequentially consistent

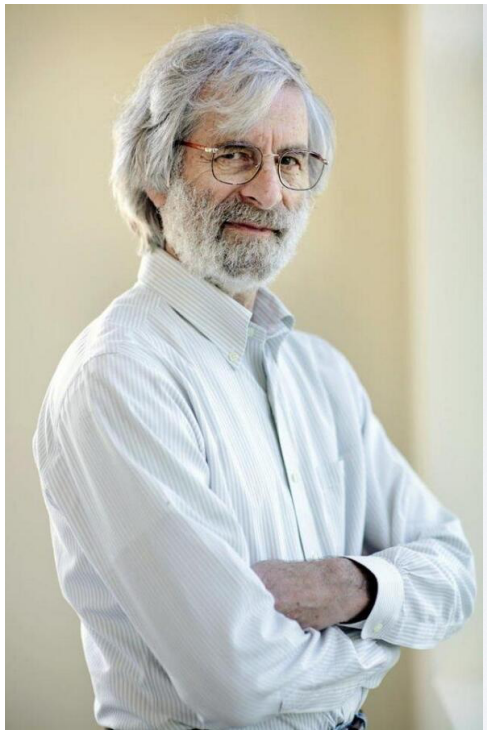
BREAK

THE BIG PICTURE



THE BIG PICTURE

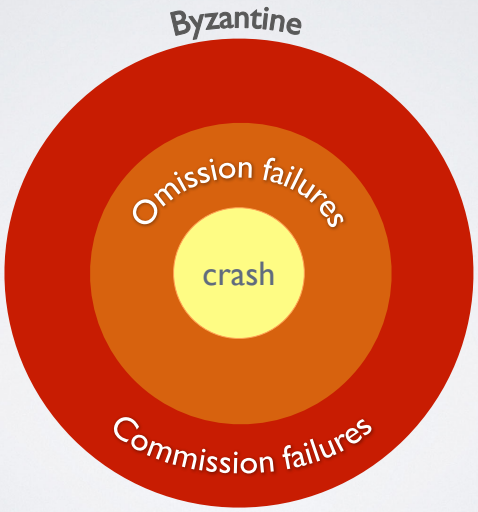




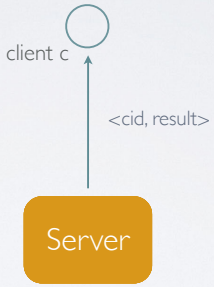
"A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable."

Leslie Lamport

FAILURE MODELS



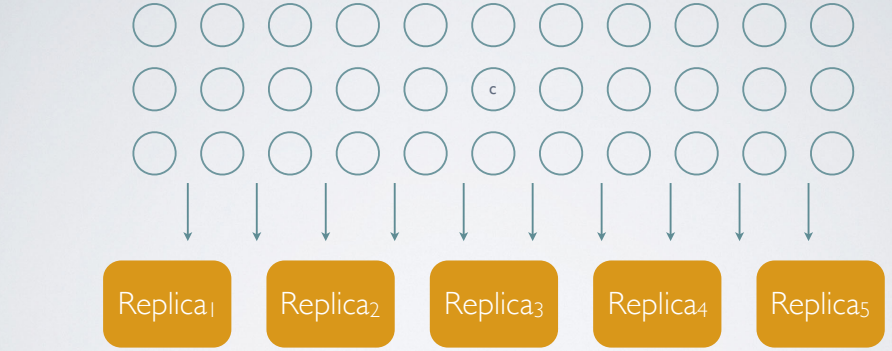
THE BIG PICTURE



THE BIG PICTURE

THE BIG PICTURE

THE BIG PICTURE



STATE MACHINE REPLICATION

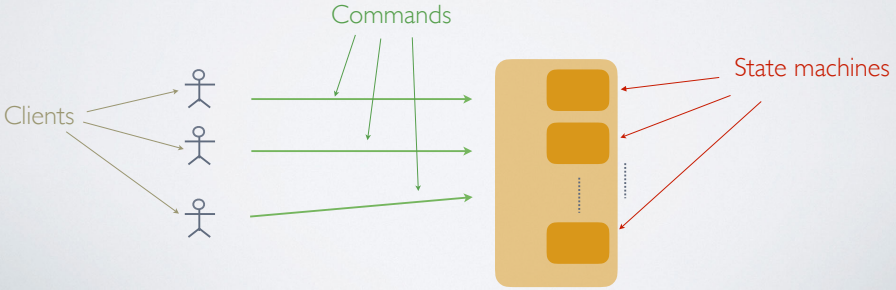
1. Make server deterministic (state machine)

STATE MACHINE REPLICATION

1. Make server deterministic (state machine)
2. Replicate server

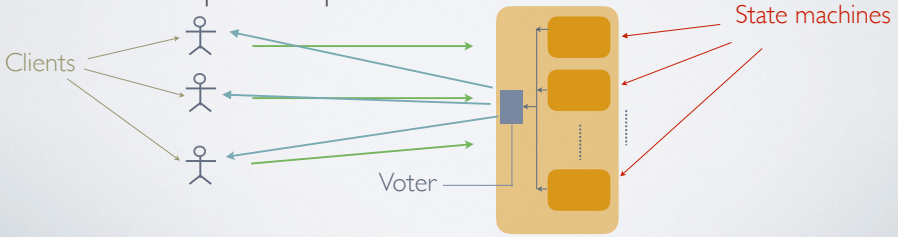
STATE MACHINE REPLICATION

1. Make server deterministic (state machine)
2. Replicate server
3. Ensure correct replicas step through the same sequence of state transitions

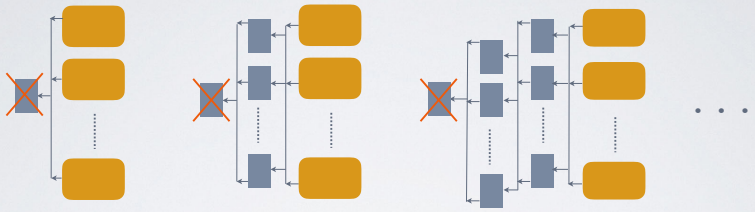


STATE MACHINE REPLICATION

1. Make server deterministic (state machine)
2. Replicate server
3. Ensure correct replicas step through the same sequence of state transitions
4. Vote on replica outputs for fault-tolerance

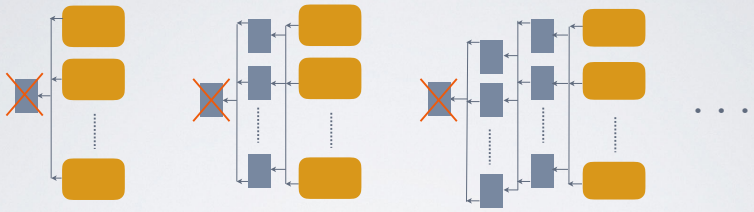


A CONUNDRUM

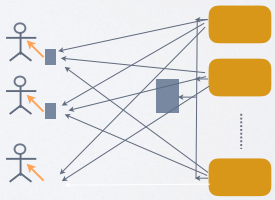


A: voter and
client share fate!

A CONUNDRUM



A: voter and
client share fate!

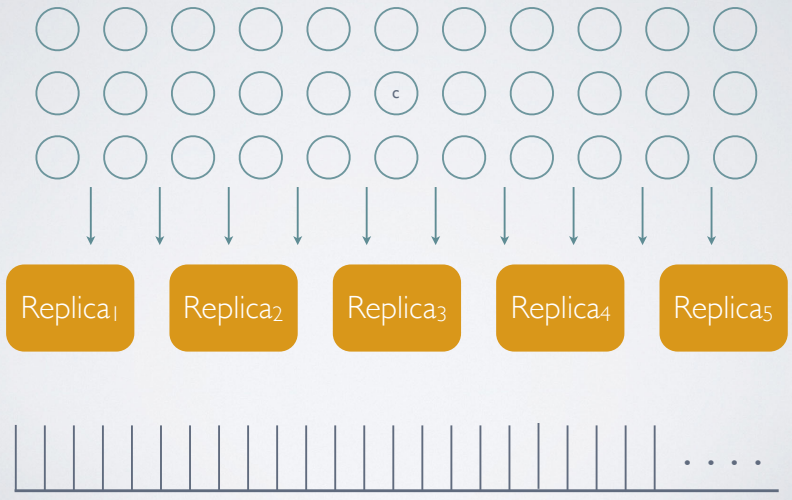


REPLICA COORDINATION

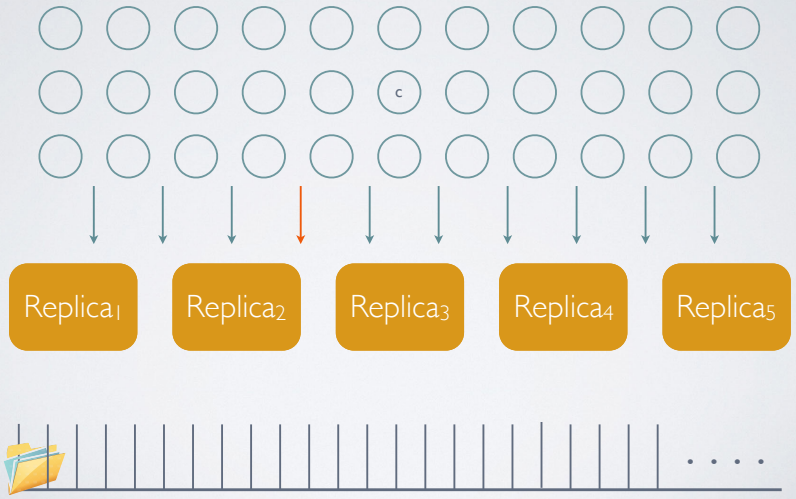
All non-faulty state machines receive
all commands in the same order

- Agreement: Every non-faulty state machine receives every command
- Order: Every non-faulty state machine processes the commands it receives in the same order

THE BIG PICTURE



THE BIG PICTURE



THE BIG PICTURE