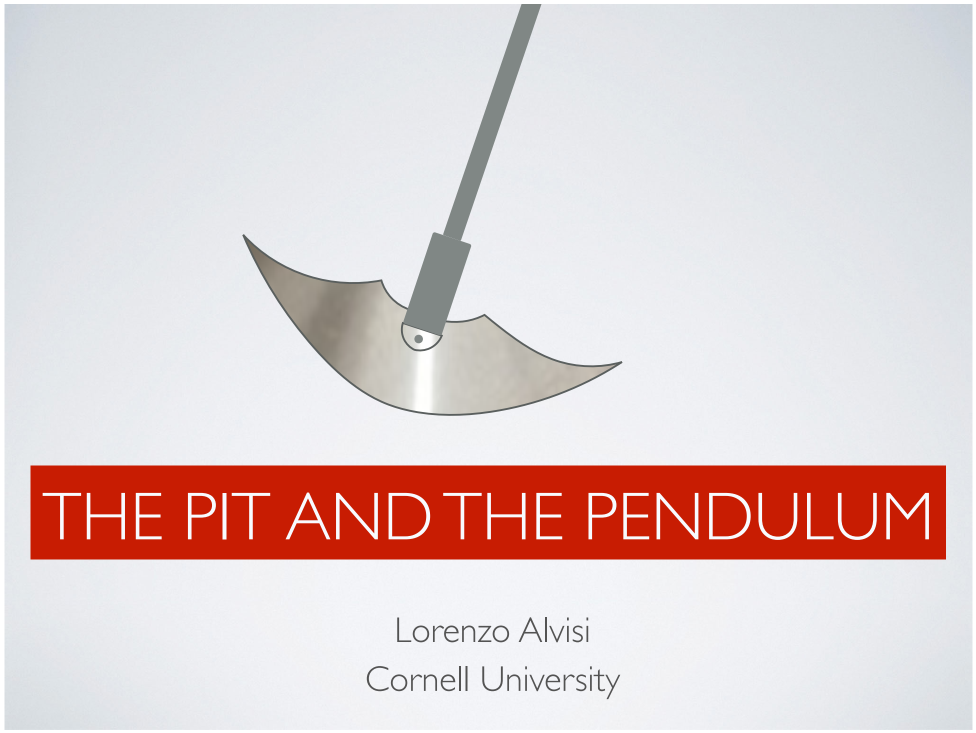


THE PIT AND THE PENDULUM

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

I KNOW WHAT YOU DID LAST SUMMER...

INTERNSHIP!

Ease of Programming

Google Cloud Platform

Microsoft Azure

ORACLE

Performance

amazon DynamoDB

mongoDB

CONCURRENCY

Pierre Franc Lamy (1855-1919)
Young girl on a balcony

Carlo Carrà (1912)
Concurrency, Woman on a balcony

CORRECTNESS

- Safety
  - "nothing bad happens"

CORRECTNESS

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

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

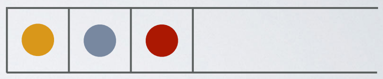
Thanks to Maurice Herlihy

- 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

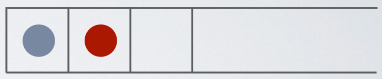


- Postcondition
  - Returns first item in queue
- Postcondition
  - Removes first item in queue

PRE AND POST CONDITIONS FOR DEQ

Thanks to Maurice Herlihy

- Precondition
  - Queue is non-empty



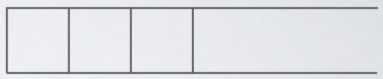
- Postcondition
  - Returns first item in queue

- Postcondition
  - Removes first item 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

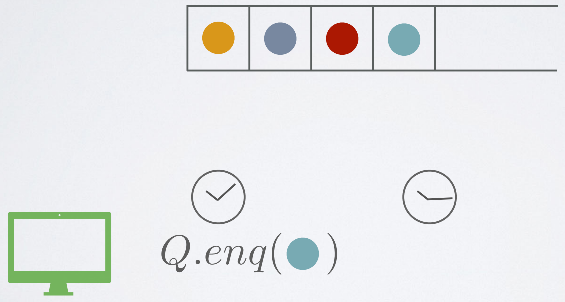
So is Maurice Herlihy

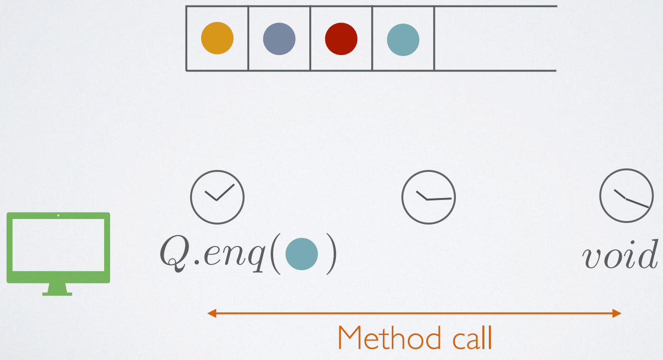
- Interactions among methods captured by side-effects on object state
  - State between method calls is meaningful
- Documentation size linear in the number of methods
  - Separation of concerns: each method described in isolation
- Can add new methods
  - Without changing description of old methods

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?

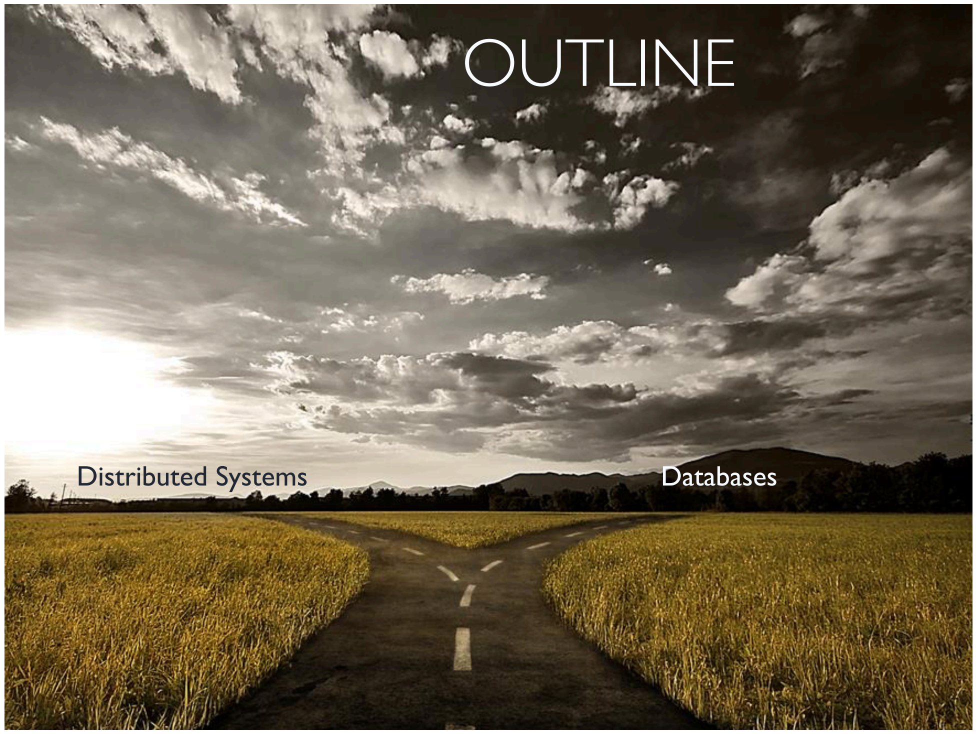
WHAT DOES IT MEAN FOR CORRECTNESS?

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

WHAT ABOUT DATABASES?

TRANSACTIONS TAKE TIME

OUTLINE

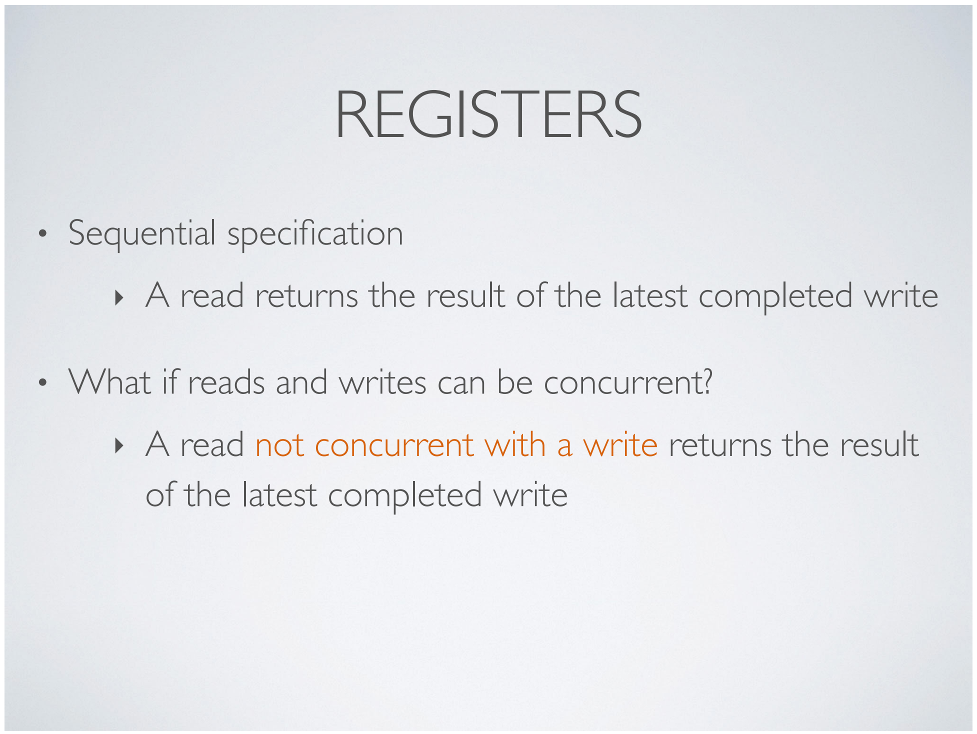


REGISTERS

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

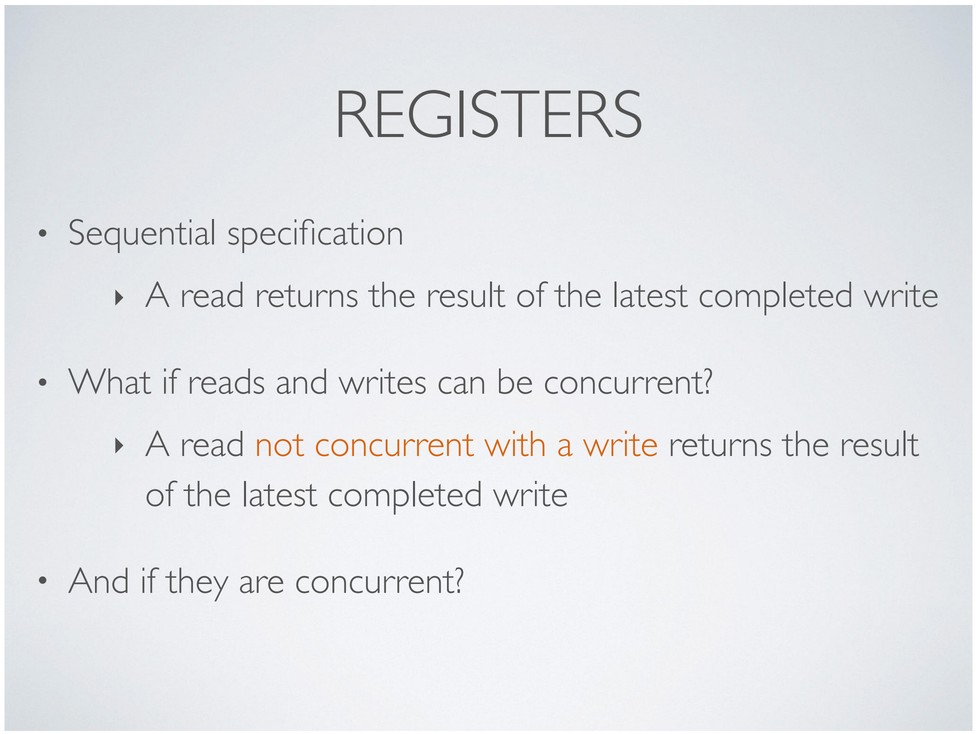
REGISTERS

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



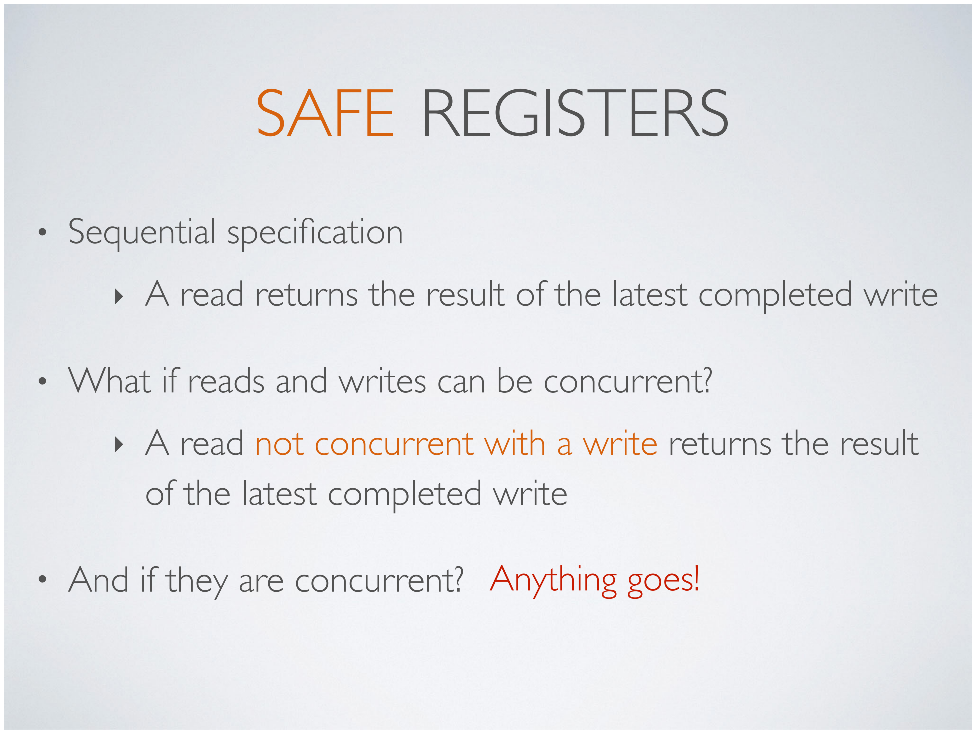
REGISTERS

- 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



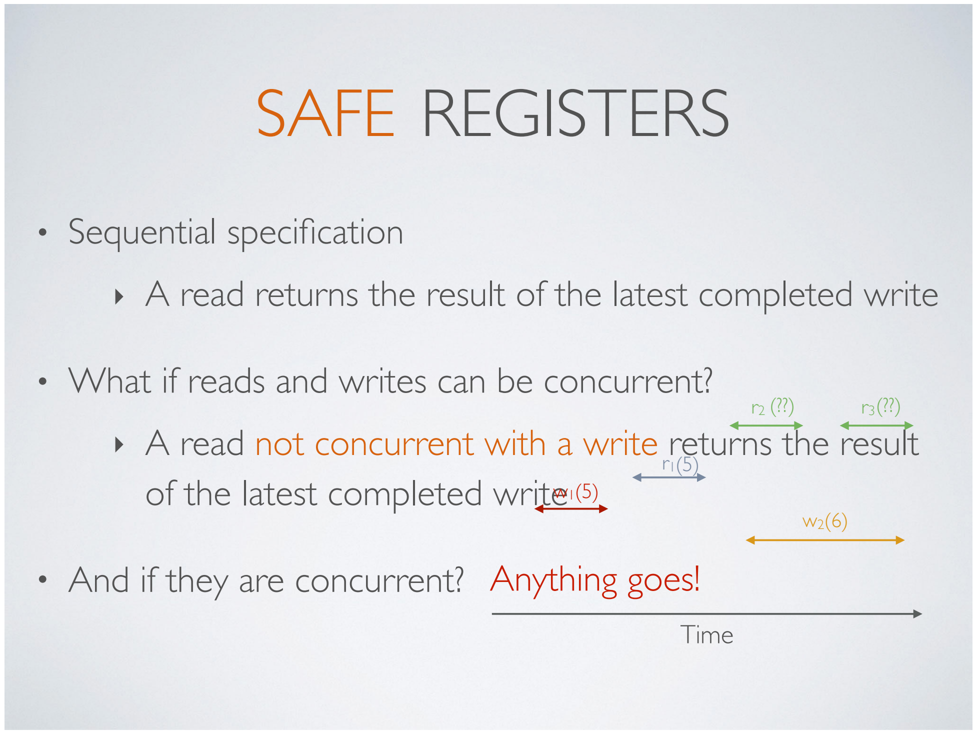
REGISTERS

- 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?



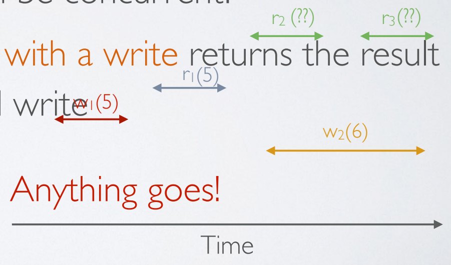
SAFE REGISTERS

- 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

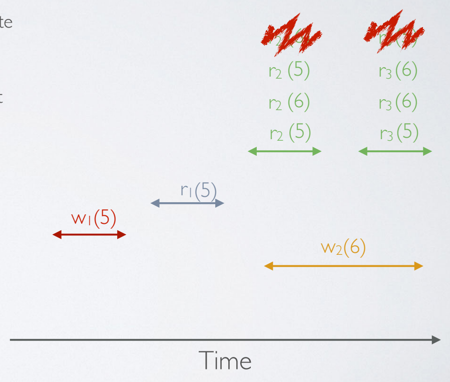
- Sequential specification
  - A read returns the result of the latest completed write
- What if reads and writes can be concurrent?
  - A read not concurrent with a write returns the result of the latest completed write
- And if they are concurrent? Anything goes!



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!



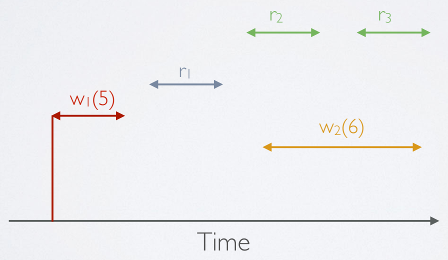
CAN WE DO BETTER?

LINEARIZABILITY

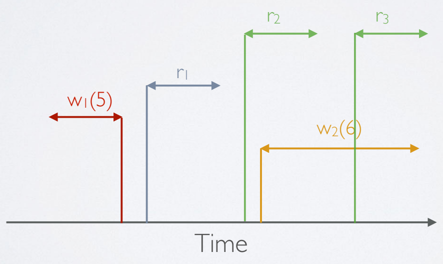
Herlihy & Wing '87

- Each method
  - Takes effect instantenously
  - Between invocation and response
- Object is correct (*linearizable*) if this "sequential" behavior is correct
  - All executions of a linearizable object are linearizable

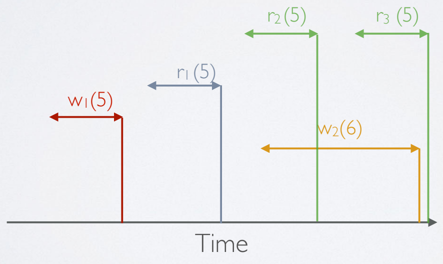
LINEARIZABLE REGISTERS



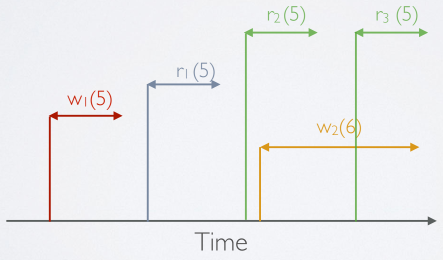
LINEARIZABLE REGISTERS



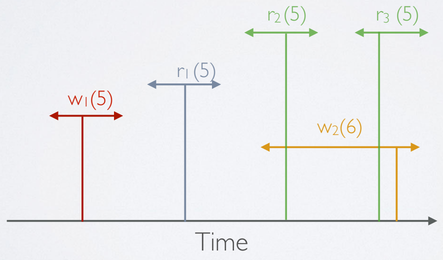
LINEARIZABLE REGISTERS



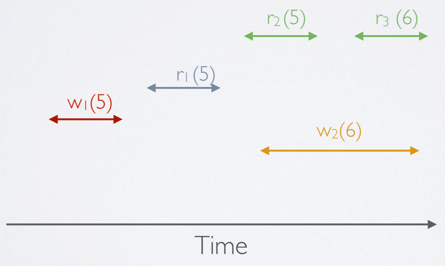
LINEARIZABLE REGISTERS



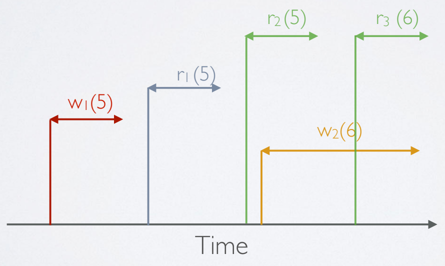
LINEARIZABLE REGISTERS (ATOMIC)



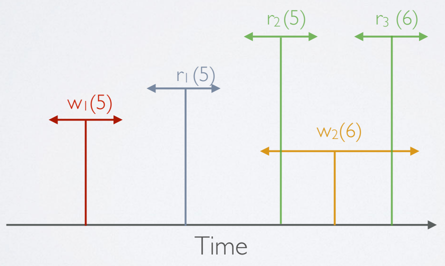
LINEARIZABLE REGISTERS



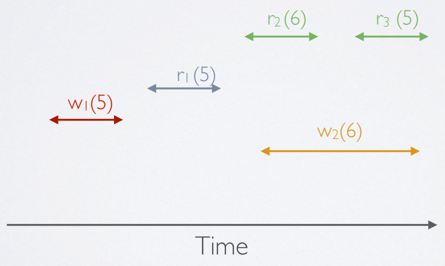
LINEARIZABLE REGISTERS



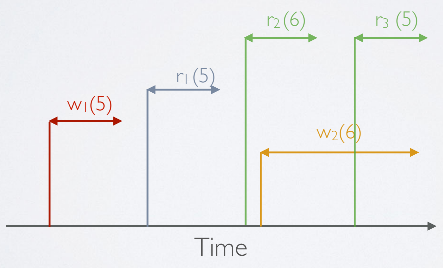
LINEARIZABLE REGISTERS



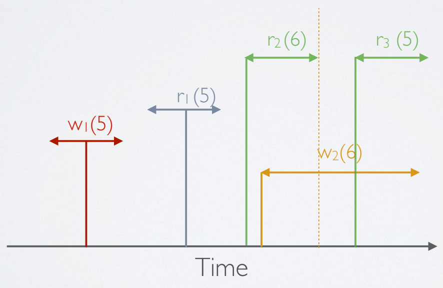
LINEARIZABLE REGISTERS



LINEARIZABLE REGISTERS

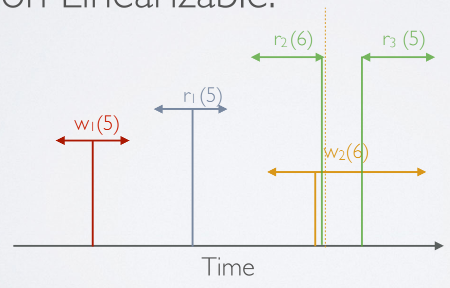


LINEARIZABLE REGISTERS

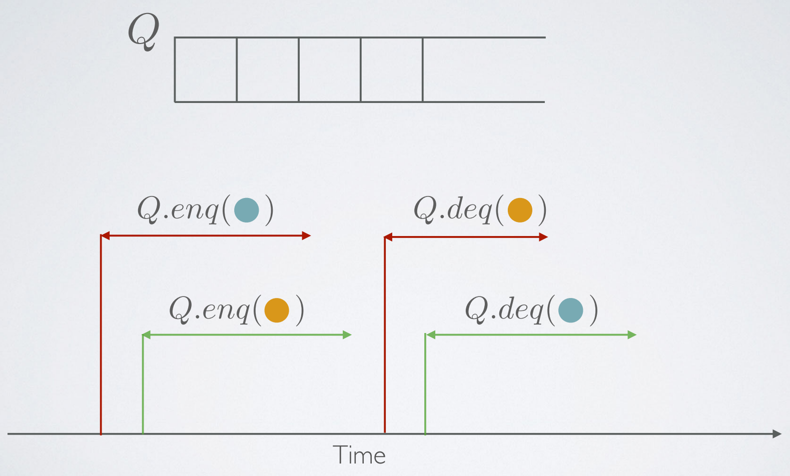


LINEARIZABLE REGISTERS

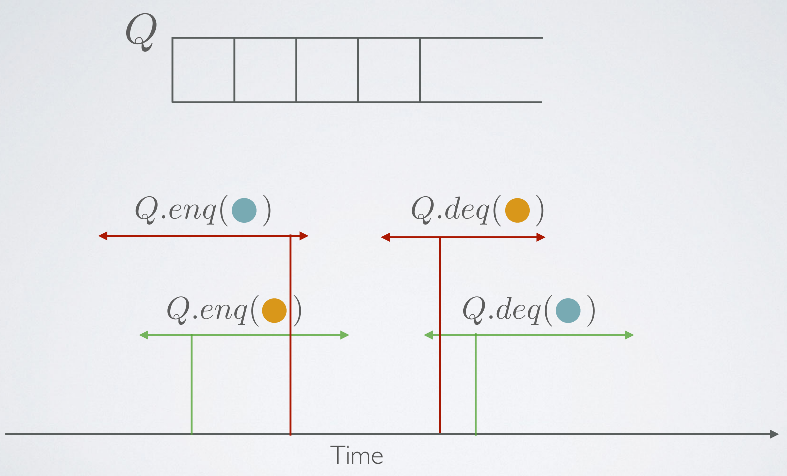
Non Linearizable!



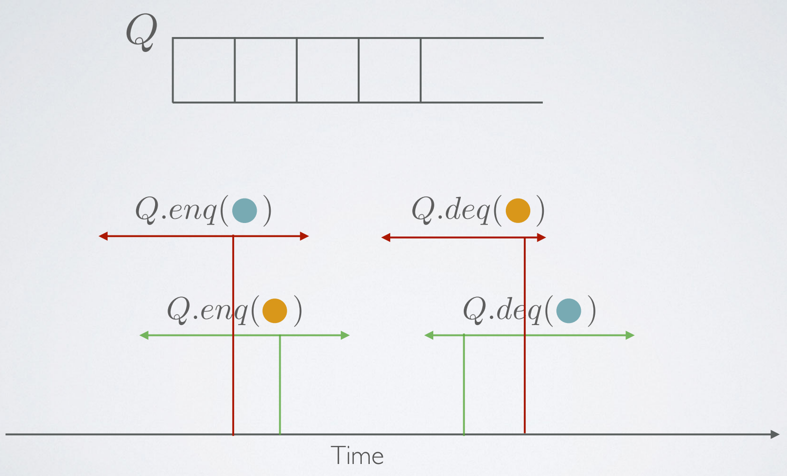
LINEARIZABLE QUEUE



LINEARIZABLE QUEUE



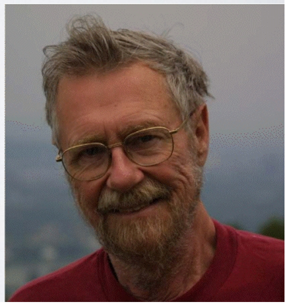
LINEARIZABLE QUEUE



LINEARIZABILITY

Herlihy & Wing '87

- Allows us to capture the notion of an object supporting atomic operations
- Is composable: executions involving linearizable objects are linearizable!
  - Separation of concerns



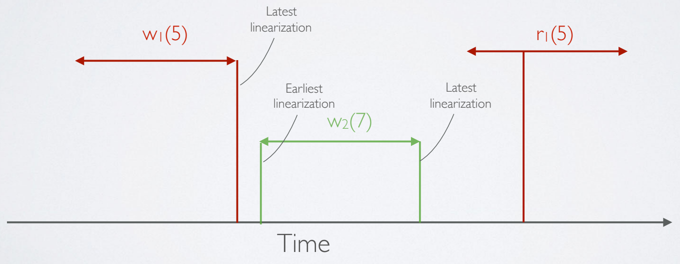
ALTERNATIVE: SEQUENTIAL CONSISTENCY

Lamport '79

- "The result of any execution is the same as if the operations of all processes were executed in some sequential order and the operations of each process appear in this sequence in the order specified by its program"
- Often used to describe multiprocessor memory architectures
- Unlike linearizability, SC's total order need not respect real time
  - Operations from the same thread cannot be reordered
  - Non-overlapping operations from different threads can be reordered

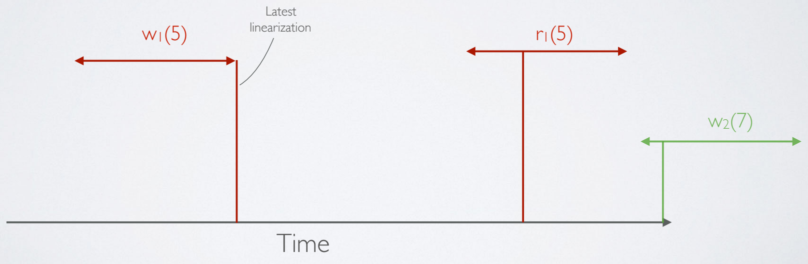
EXAMPLE

Non Linearizable…



EXAMPLE

… but Sequentially Consistent!

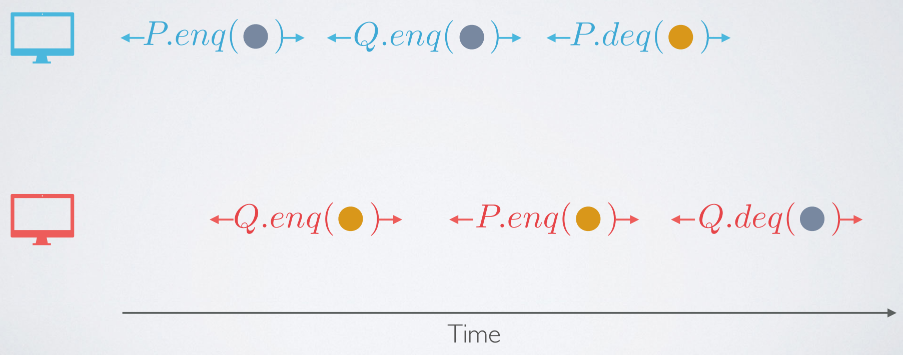


THEOREM

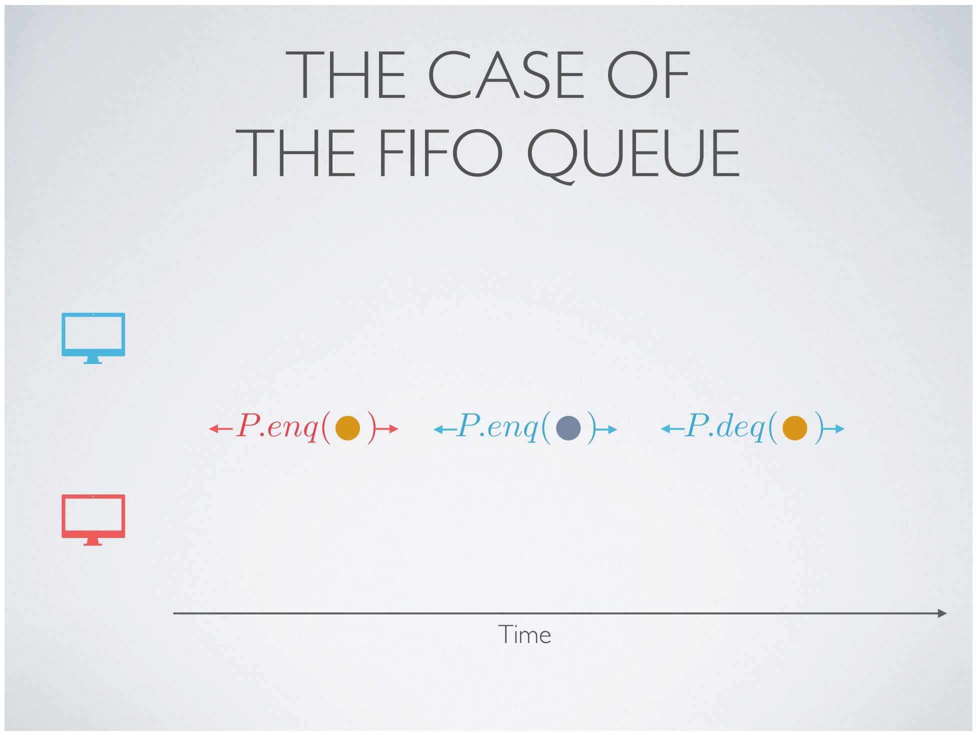
Sequential Consistency
Is Not Composable

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

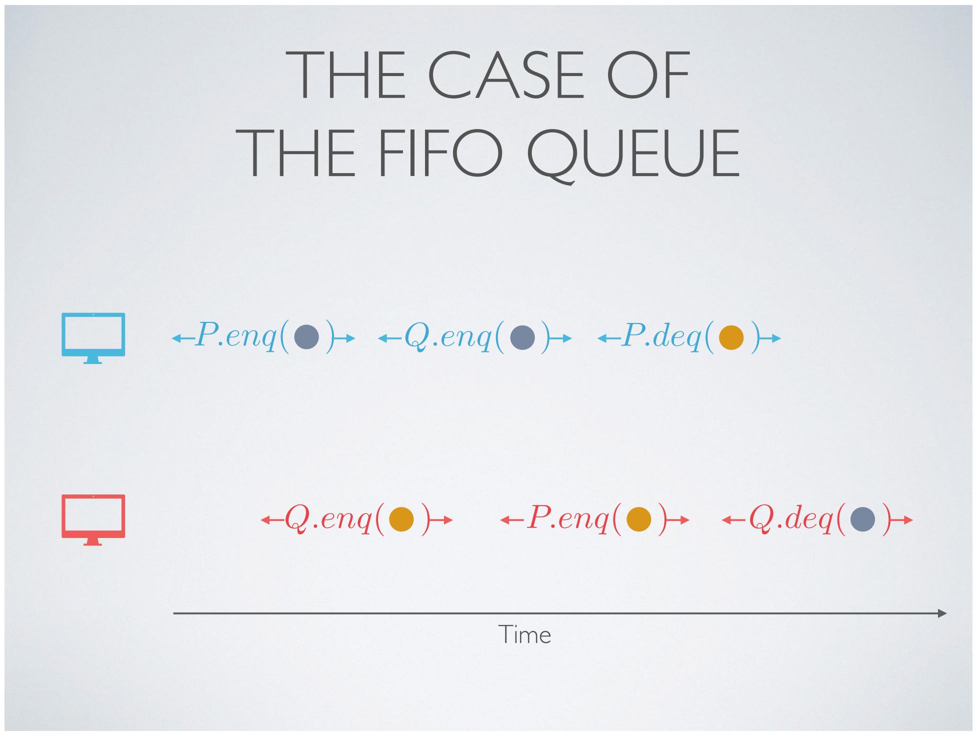
THE CASE OF THE FIFO QUEUE



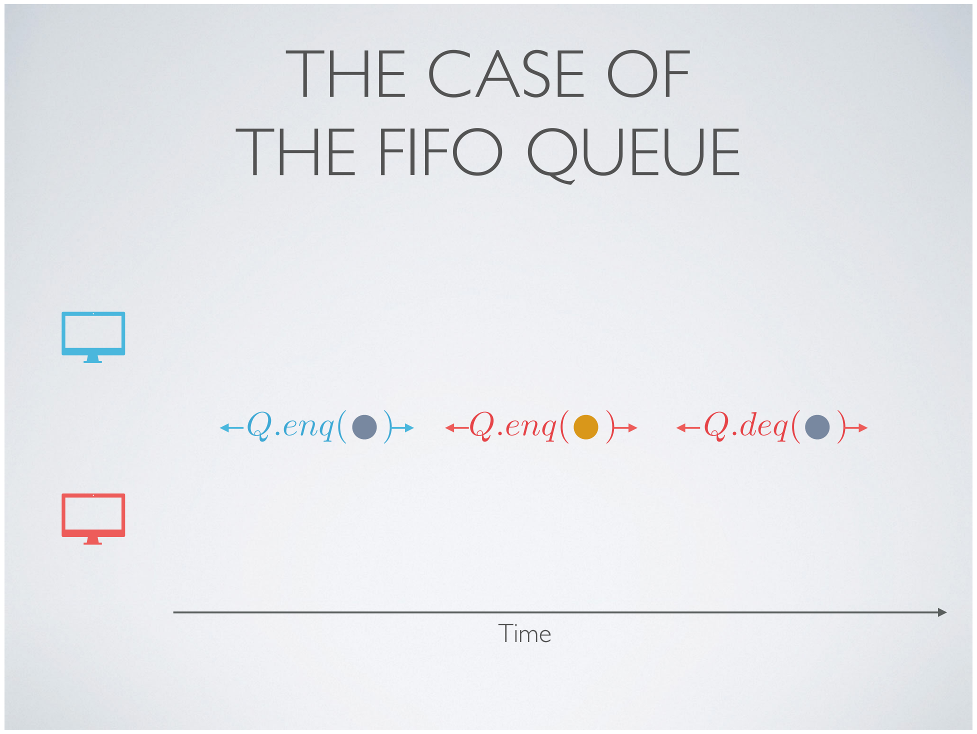
THE CASE OF THE FIFO QUEUE



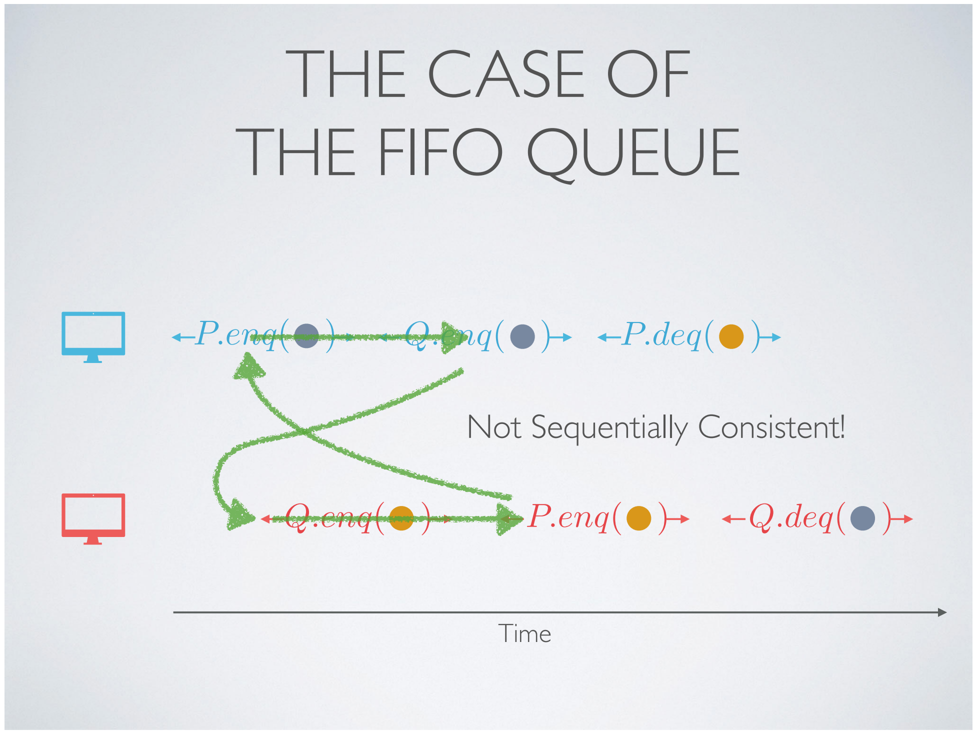
THE CASE OF THE FIFO QUEUE



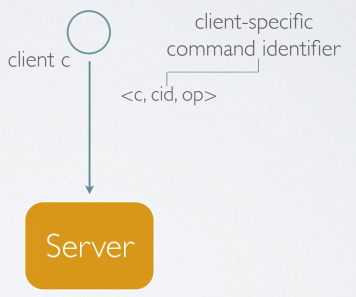
THE CASE OF THE FIFO QUEUE



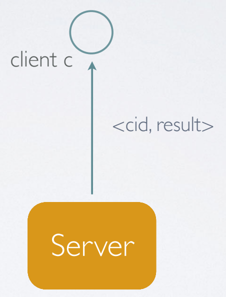
THE CASE OF THE FIFO QUEUE

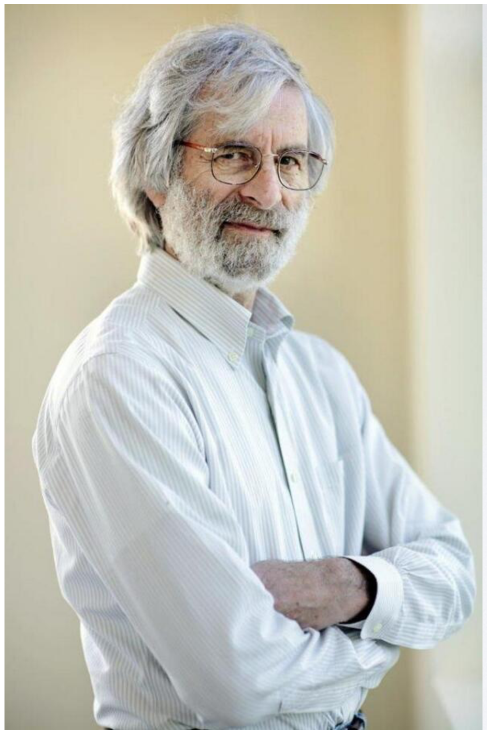


THE BIG PICTURE



THE BIG PICTURE

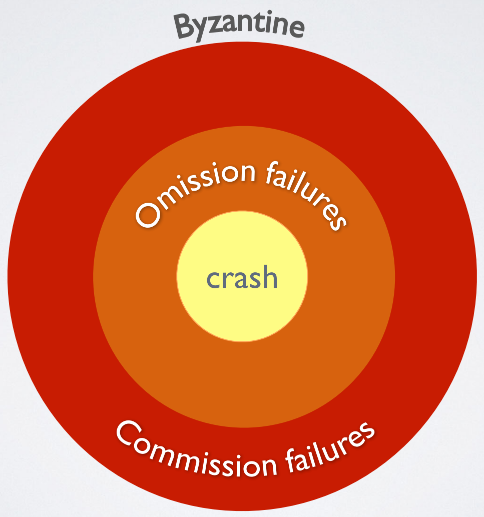




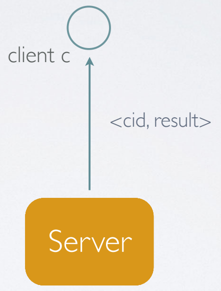
"A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable."

Leslie Lamport

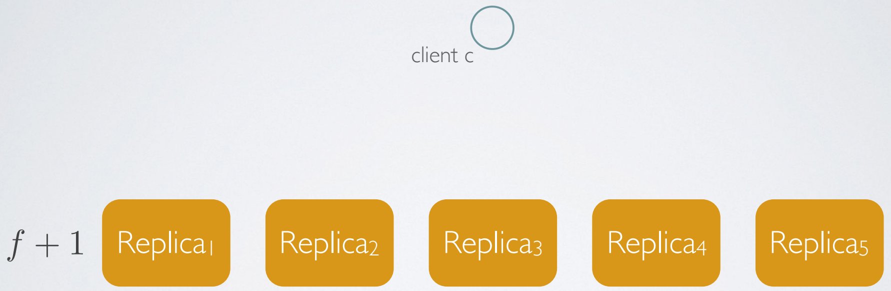
FAILURE MODELS



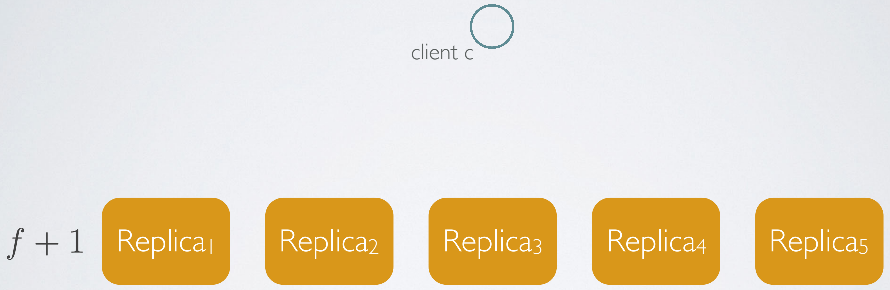
THE BIG PICTURE



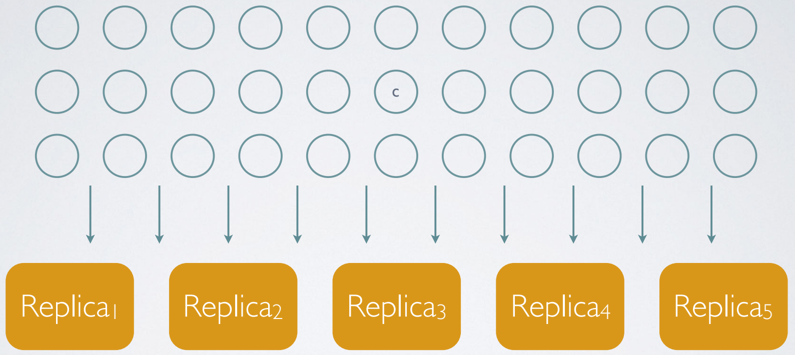
THE BIG PICTURE



THE BIG PICTURE



THE BIG PICTURE

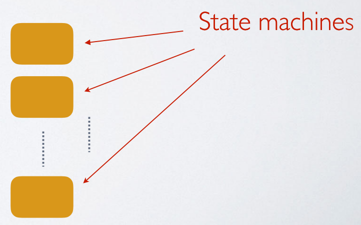


STATE MACHINE REPLICATION

1. Make server deterministic (state machine)

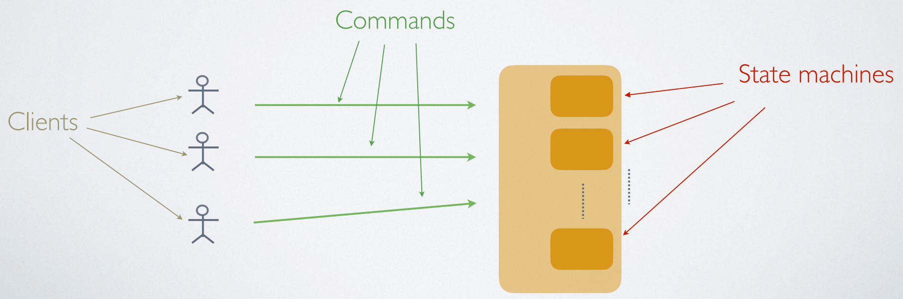
STATE MACHINE REPLICATION

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



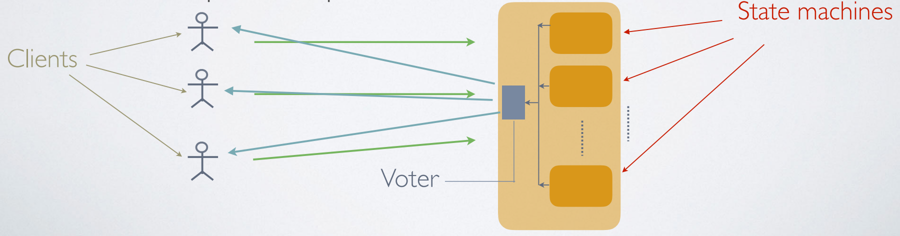
STATE MACHINE REPLICATION

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

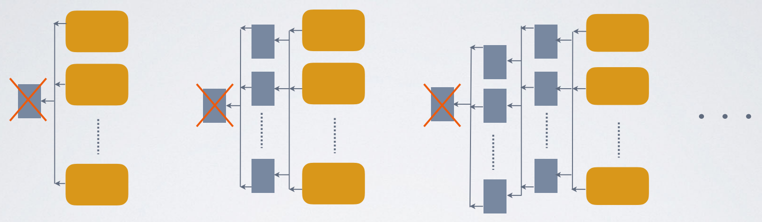


STATE MACHINE REPLICATION

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

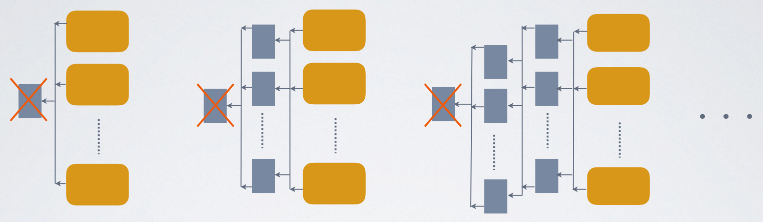


A CONUNDRUM

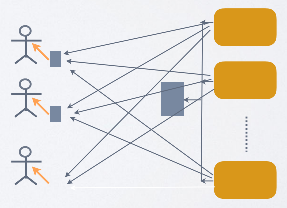


A: voter and
client share fate!

A CONUNDRUM



A: voter and
client share fate!

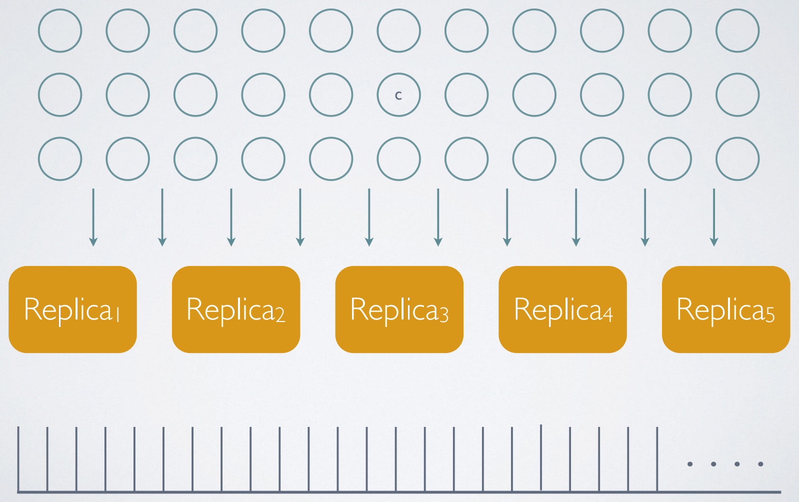


REPLICA COORDINATION

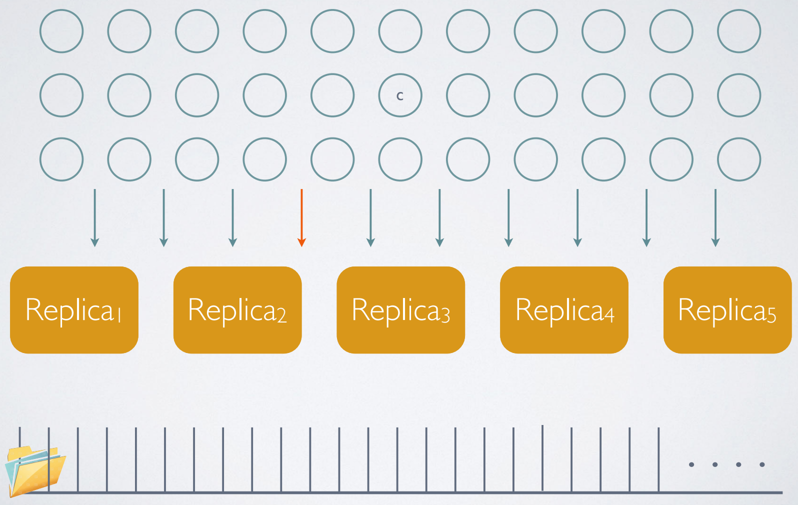
All non-faulty state machines receive
all commands in the same order

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

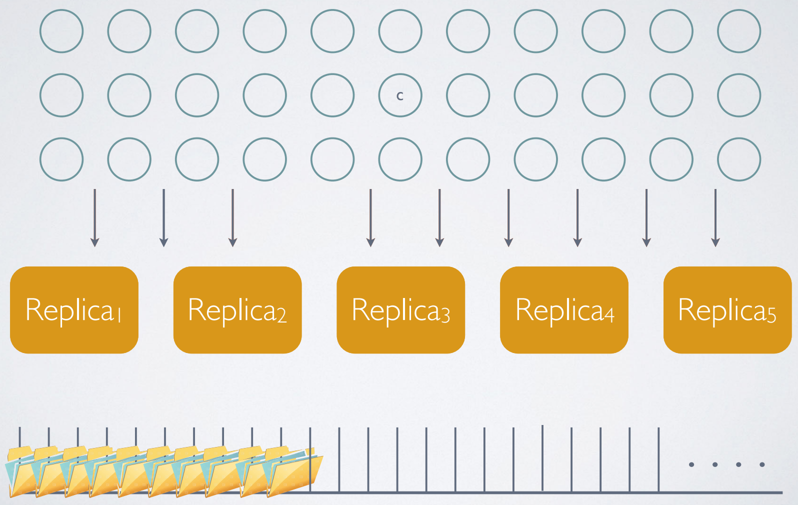
THE BIG PICTURE



THE BIG PICTURE



THE BIG PICTURE



CONSENSUS

- **Validity** – If a process decides v, then v was proposed by some process
- **Agreement** – No two correct process decide differently
- **Integrity** – No correct process decides twice
- **Termination** – Every correct process eventually decides some value

MESSAGES TAKE TIME

Does it matter how much?

OF COURSE!

AND YET...

Should it matter for CORRECTNESS?

Assumptions are vulnerabilities!

ASYNCHRONOUS SYSTEMS

NO centralized clock

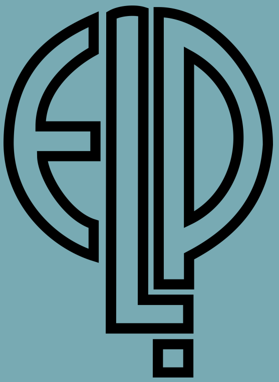
NO upper bound on the relative speed of processes

NO upper bound on message delivery time

CONSENSUS† IS IMPOSSIBLE IN AN ASYNCHRONOUS SYSTEM*

†deterministic

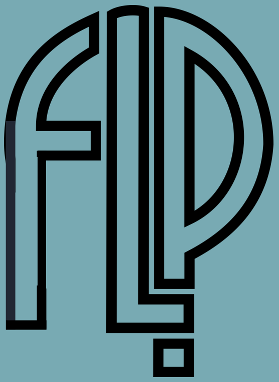
*in the presence of failures

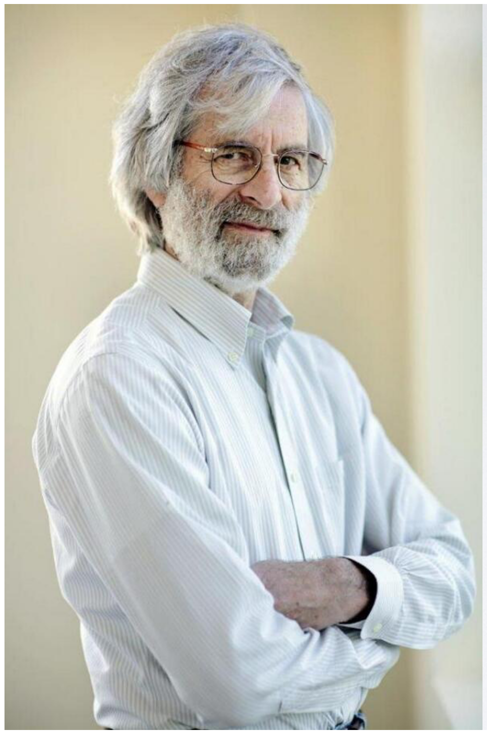


CONSENSUS† IS IMPOSSIBLE IN AN ASYNCHRONOUS SYSTEM*

†deterministic

*in the presence of failures





Paxos

Always safe

Ready to pounce on liveness