# Universal constructions

***R. Guerraoui***
***Distributed Programming Laboratory***

© R. Guerraoui

# Universality [Her91]

- ***Definition 1 :*** A type T is *universal* if, together with registers, instances of T can be used to provide a wait-free linearizable implementation of any other type (with a sequential specification)

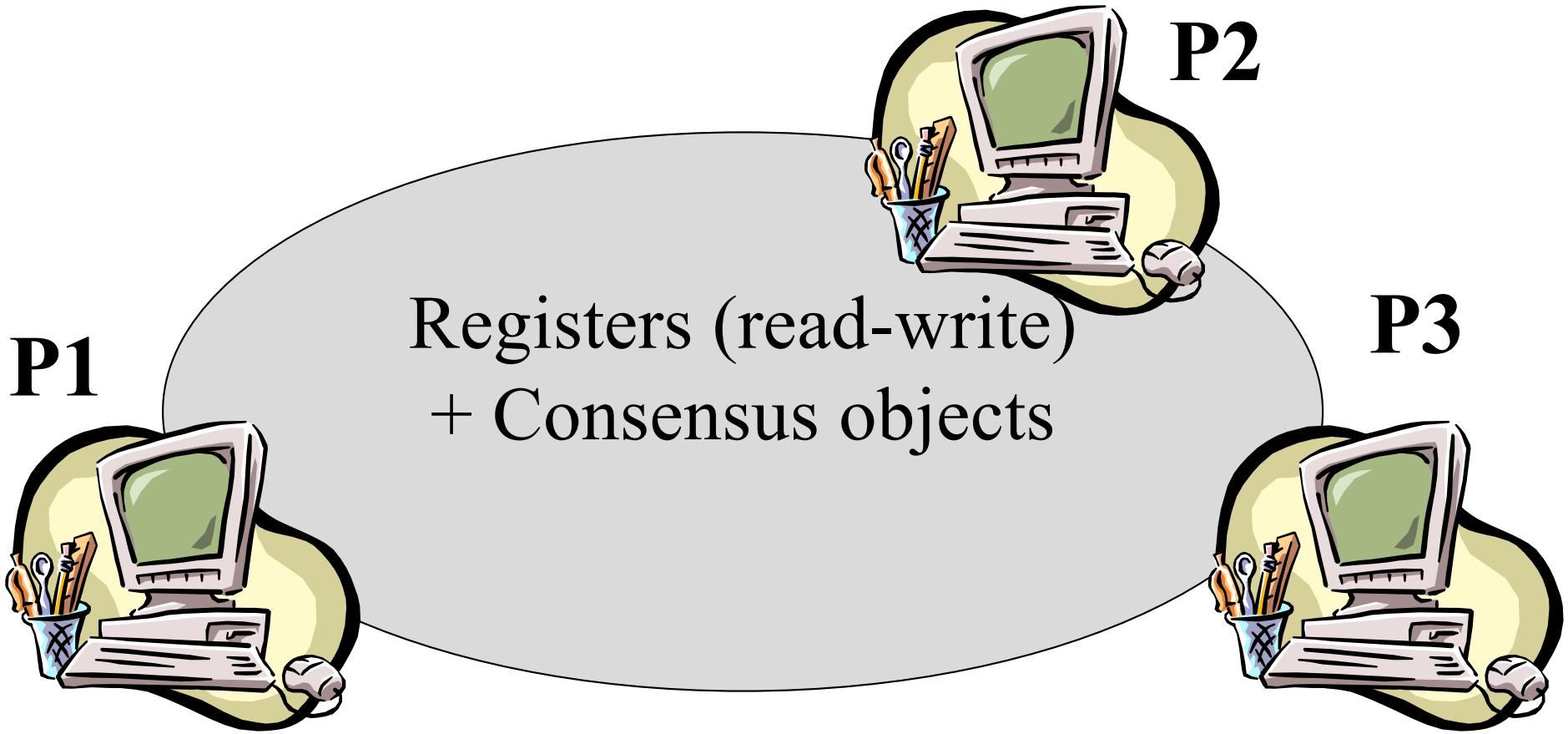
- ***Definition 2:*** The implementation is called a *universal construction*

# Consensus

- ***Theorem 1:*** Consensus is universal [Her91]
- ***Corollary 1:*** Compare&swap is universal
- ***Corollary 2:*** Test&set is universal in a system of 2 processes (it has consensus number 2)

- ***Corollary to FLP/LA:*** Register is not universal in a system of at least 2 processes

# Shared memory model



# The consensus object

- One operation ***propose()*** which returns a value. When a propose returns, the process decides

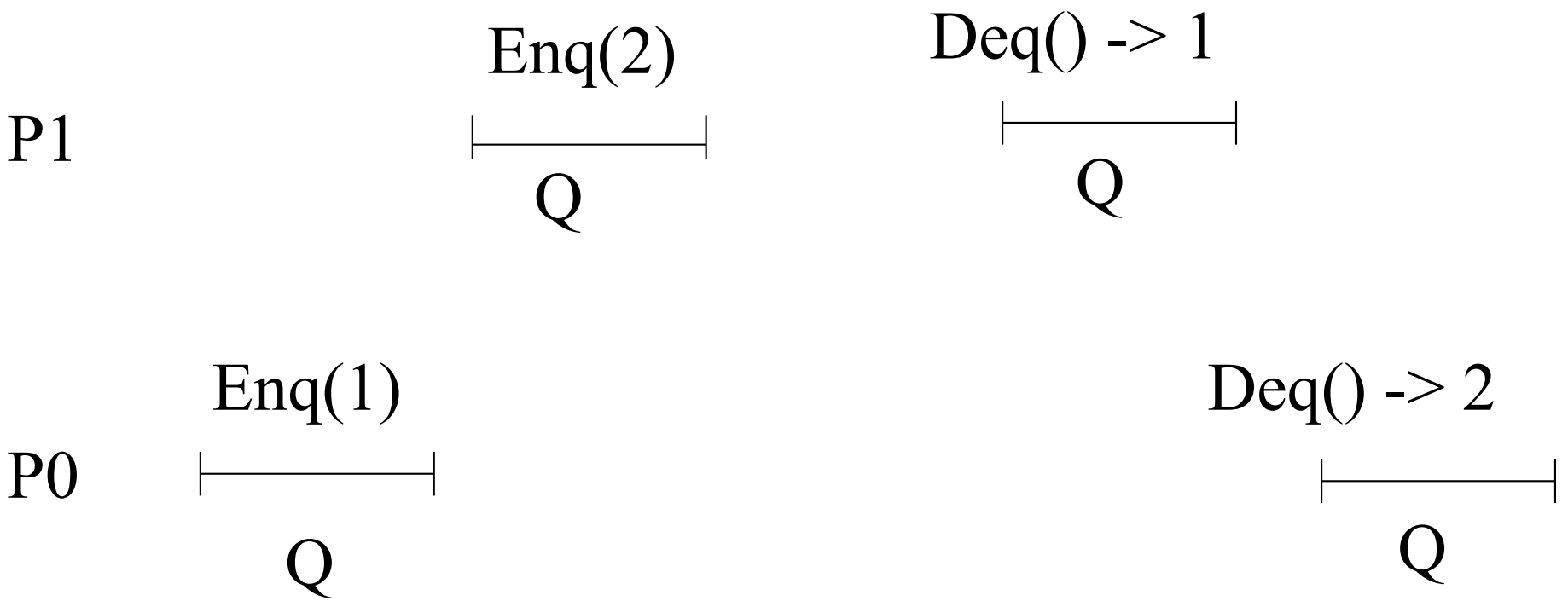
- *Agreement:* No two processes decide differently
- *Validity:* Every decided value is a proposed value
- *Termination (wait-free):* Every correct process that proposes a value eventually decides

# Universality

- We consider first ***deterministic*** objects and then ***non-deterministic*** ones

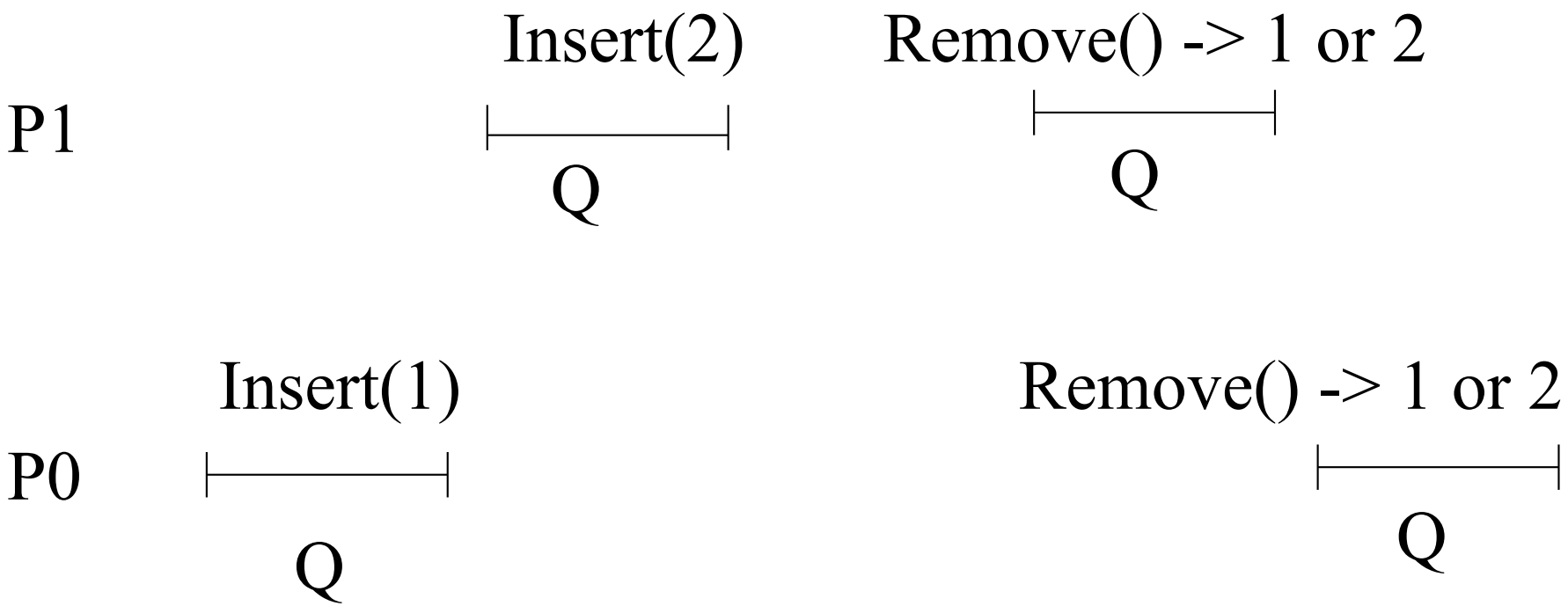
- An object is ***deterministic*** if the result and final state of any operation depends solely on the initial state and the arguments of the operation

# Example (FIFO Queue)
## Sequential deterministic specification



# Example (Set)

## Sequential non-deterministic specification

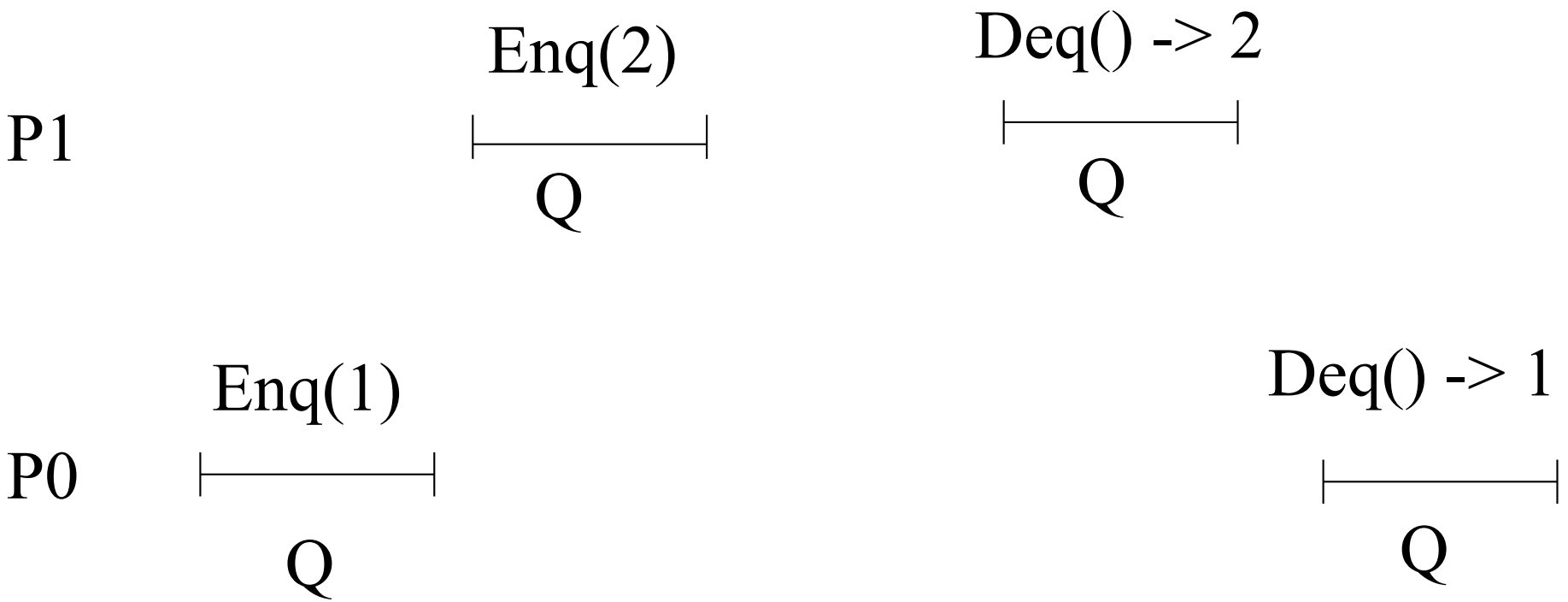


# Universal construction (1)

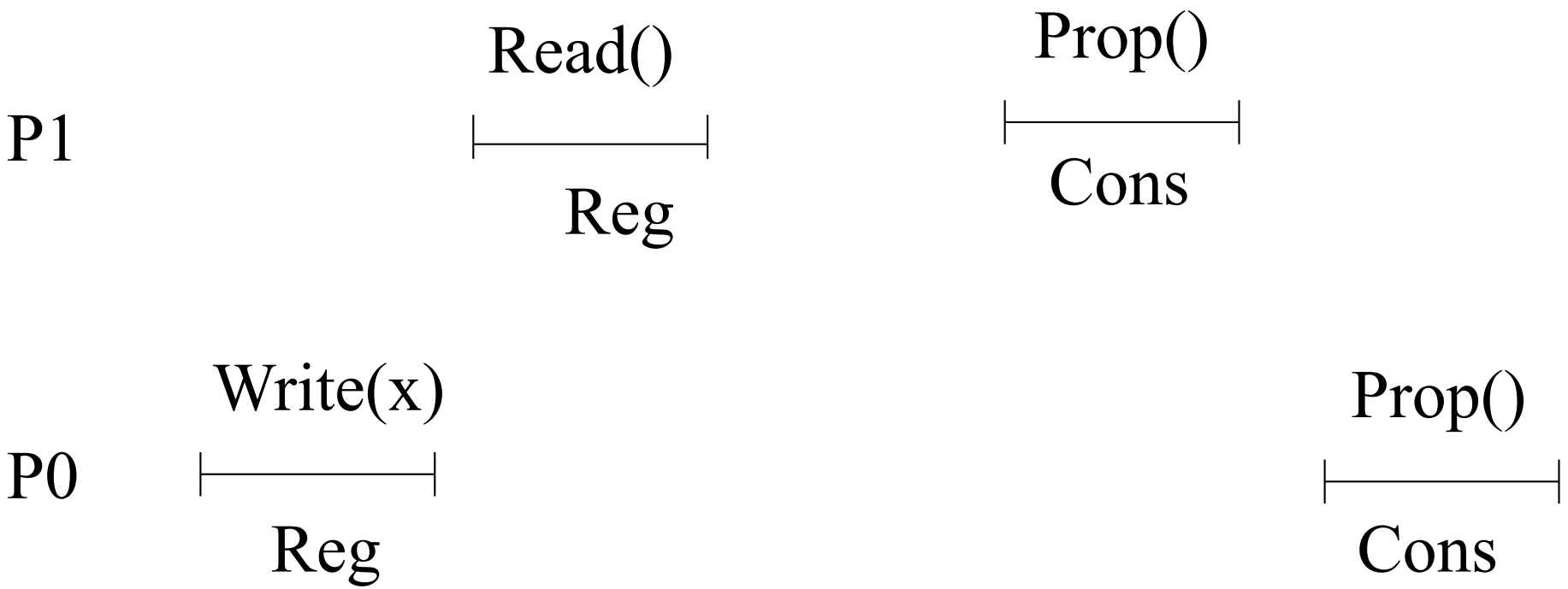
- We assume a deterministic object

- We give an algorithm where
  - ✓every process has a copy of the object (inherent for wait-freedom)
  - ✓processes communicate through registers and consensus objects (linearizability)

# Example (FIFO Queue)
# Non-linearizable execution



# Universal algorithm (1)



# Shared objects

- The processes share an array of  n SWMR registers ***Lreq*** (theoretically of infinite size)

- This is used to ***inform*** all processes about which requests need to be performed

# Shared objects

- The processes also share a consensus list ***Lcons*** (also of infinite size)

- This is used to ensure that the processes agree on a **total order** to perform the requests (on their local copies)

  ✓We use an ordered list of consensus objects

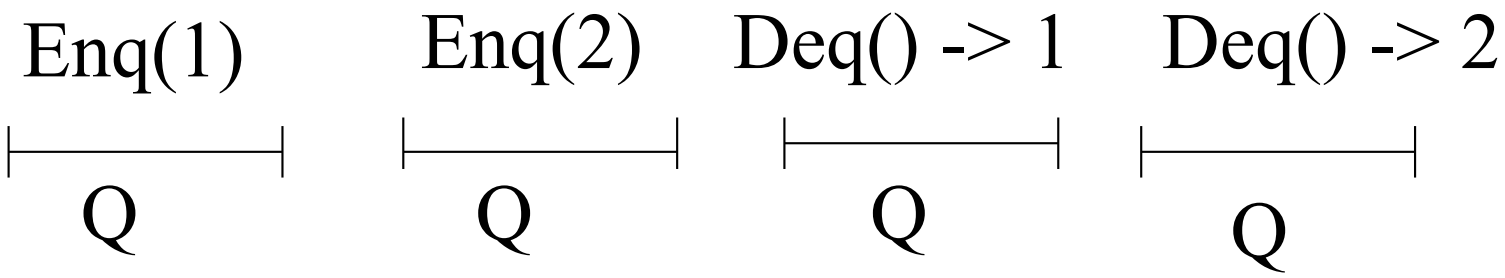
  ✓Every such object is uniquely identified by an integer

  ✓Every consensus object is used to agree on a set of requests (the integer is associated to this set)

# Universal algorithm (1)

- The algorithm combines the shared registers ***Lreq[l]*** and the consensus object list ***Lcons*** to ensure that:
  - ✓Every request invoked by a correct process is performed and a result is eventually returned (wait-free)
  - ✓Requests are executed in the same total order at all processes (i.e., there is a linearization point)
  - ✓This order reflects the real-time order (the linearization point is within the interval of the operation)

# Linearization (FIFO Queue)



# Local data structures

- Every process also uses two local data structures:
  - ✓ A list of requests that the process has performed (on its local copy): ***lPerf***
  - ✓ A list of requests that the process has to perform: ***llnv***
- Every request is uniquely identified

# Universal algorithm (1)

- Every process pI executes three // tasks:
  - ✓Task 1: whenever pI has a new request, pI adds it to ***Lreq[I]***
  - ✓Task 2: periodically, pI adds the new elements of every ***Lreq[J]*** into *llnv*
  - ✓Task 3: while (*llnv* – *lPerf*) is not empty, pI performs requests using ***Lcons***

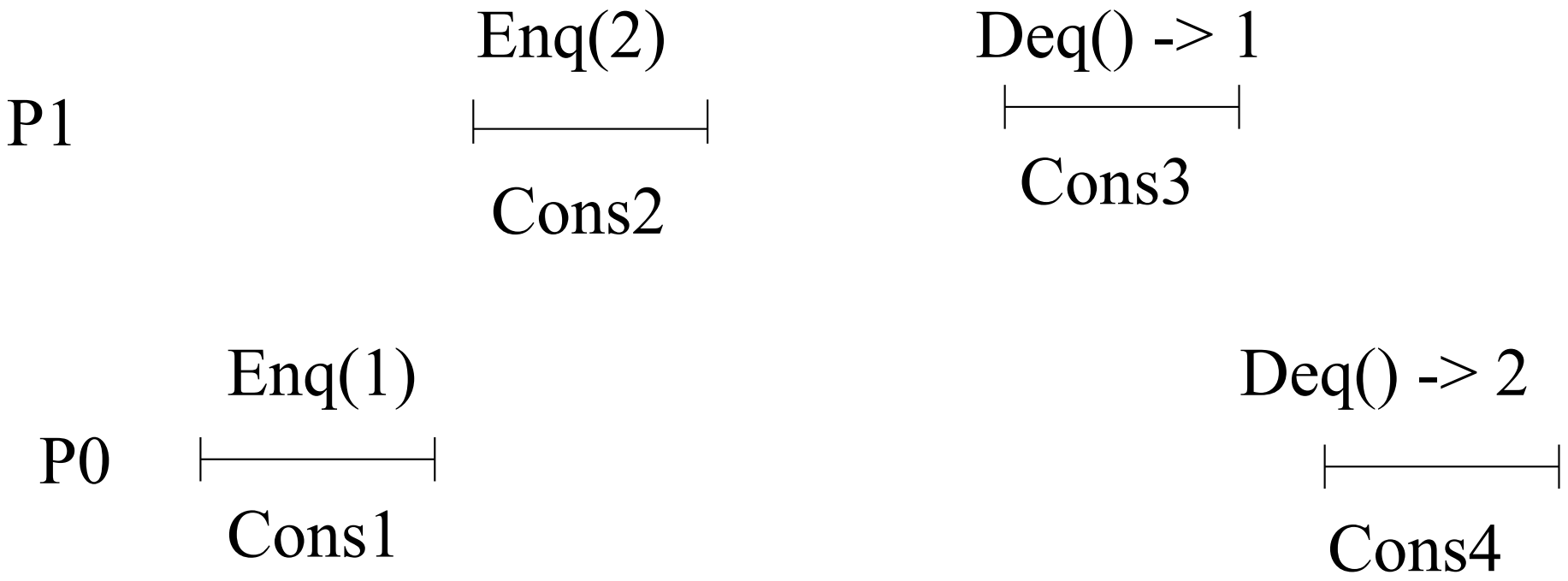
# Task 3

- While *lInv – lPerf* is not empty
  - pI proposes lI*nv* – lP*erf* for a new consensus in ***Lcons*** (increasing the consensus integer)
  - pI performs the requests decided (that are not in *Lperf*) on the local copy
  - For every performed request:

    **pI returns the result if the request is in Lreq[I]**

    **pI puts the request in lPerf**

# Example (FIFO Queue)



# Correctness

- ***Lemma 1*** (wait-free): every correct process pI that invokes req eventually returns from that invocation
- Proof (sketch): Assume by contradiction that pI does not return from that invocation; pI puts req into ***Lreq*** (Task 1); eventually, every proposed *lInv - lPerf* contains req (Task 2); and the consensus decision contains req (Task 3); the result is then eventually returned (Task 3)

# Correctness

- ***Lemma 2*** (order): the processes execute the requests in the same total order

- Proof (sketch): the processes agree on the same total order for sets of requests and then use the same order within every set of requests (the linearization order is determined by the integers associated with the consensus)

# Correctness

- ***Lemma 3*** (real-time): if a request req1 precedes in real-time a request req2, then req2 appears in the linearization after req1

- Proof (sketch): it directly follows from the algorithm that the result of req2 is based on the state of req1

# Why not?

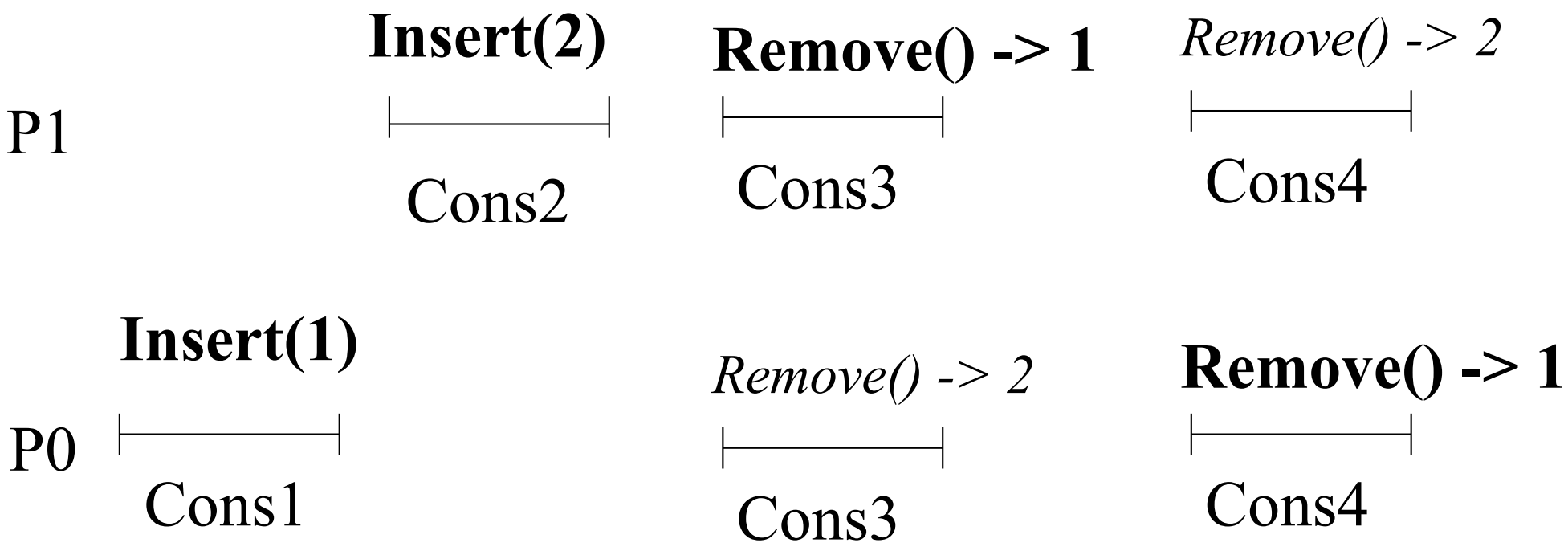
- Every process pI executes three // tasks:
  - ✓Task 1: whenever pI has a new request, pI adds it to *lInv*
  - ✓Task 3: while (*lInv* – *lPerf*) is not empty, pI performs requests using ***Lcons***

# Universality (1 + 2)

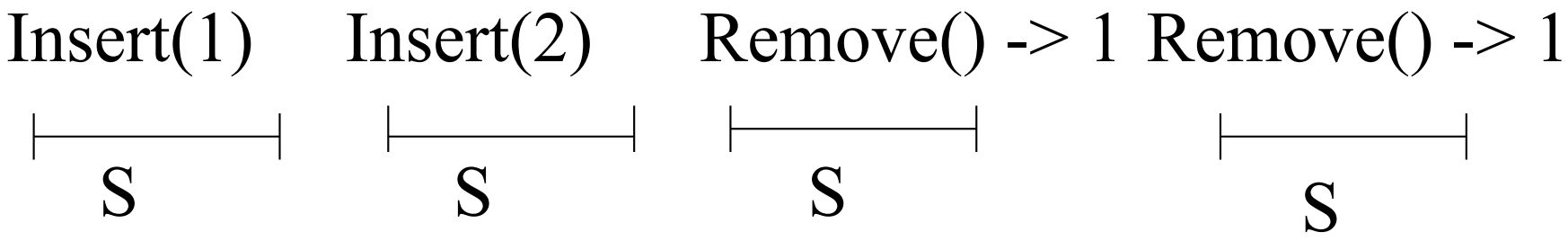
- We consider first deterministic objects and then non-deterministic ones

- An object is non-deterministic if the result and final state of an operation might differ even with the same initial state and the same arguments

# Example (Set)


Insert(2)
Remove() -> 1
Remove() -> 2
P1
Cons2
Cons3
Cons4
Insert(1)
Remove() -> 2
Remove() -> 1
P0
Cons1
Cons3
Cons4

# Non-linearization



# A restricted deterministic type

- Assume that a non-deterministic type T is defined by a relation $\delta$ that maps each state *s* and each request *o* to **a set** of pairs *(s',r)*, where *s'* is a new state and *r* is the returned result after applying request *o* to an object of T in state *s*.

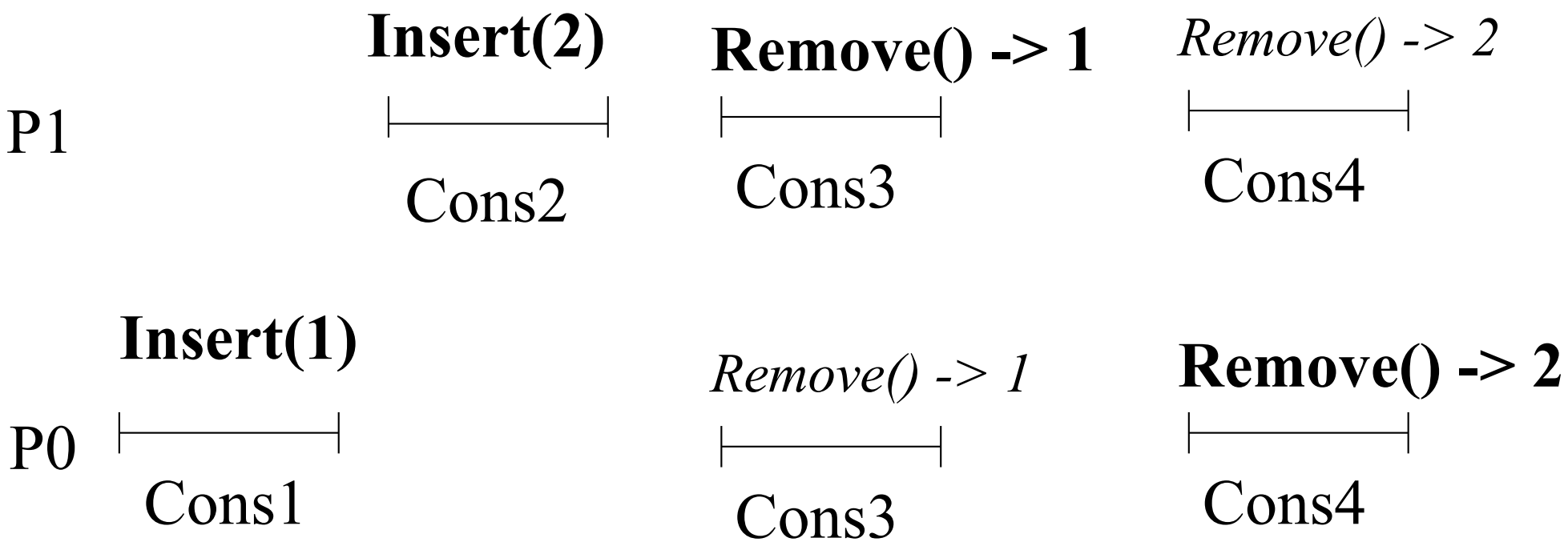
- Define a function $\delta'$ as follows:

  For any s and o, $\delta'(s,o) \in \delta(s,o)$.

The type defined by $\delta'$ is deterministic

# It is sufficient to implement a type defined by δ' !

- Every execution of the resulting (deterministic) object will satisfy the specification of T.



# Task 3 (Preserving non-determinism)

- While *IInv – IPerf* is not empty
  - pI produces the reply and new state (update) from request by performing:

    *(reply,update):= object.exec(request)*

999415138

  - pI proposes (request,reply,update) to a new consensus in ***Lcons*** (increasing the consensus integer) producing (re,rep,up)
  - pI updates the local copy: object.update(up)
  - pI returns the result if the request is in Lreq[I]
  - pI puts (req,rep,up) in IPerf