# On restricted-use objects and beyond

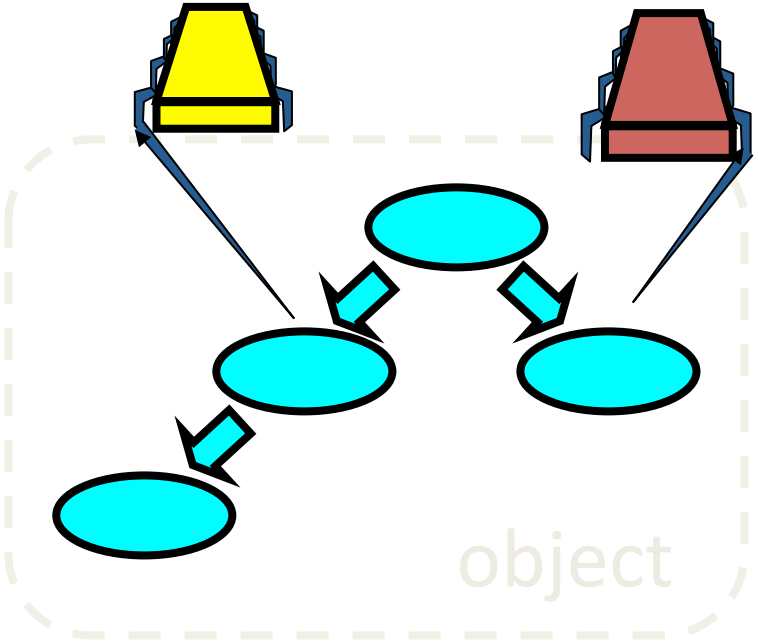
Alessia Milani

LaBRI - Université de Bordeaux

# Concurrent Objects

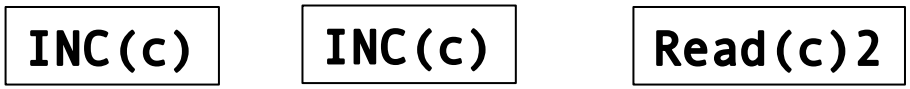
Data object shared by concurrent processes

to coordinate to solve a common task

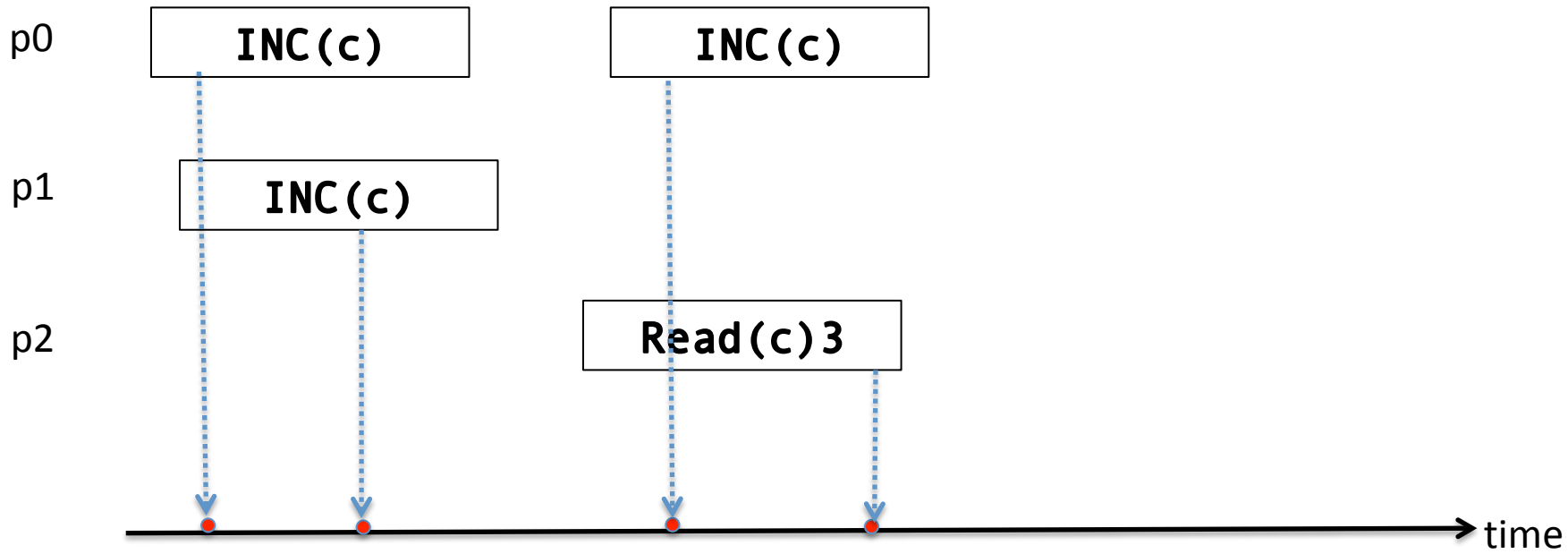


# Counter

- Sequential specification
  - INCrement operation : increases by 1 the current value of the counter
  - READ operation : returns the number of INCrement operations before it
- Example



# Linearizable Counter c



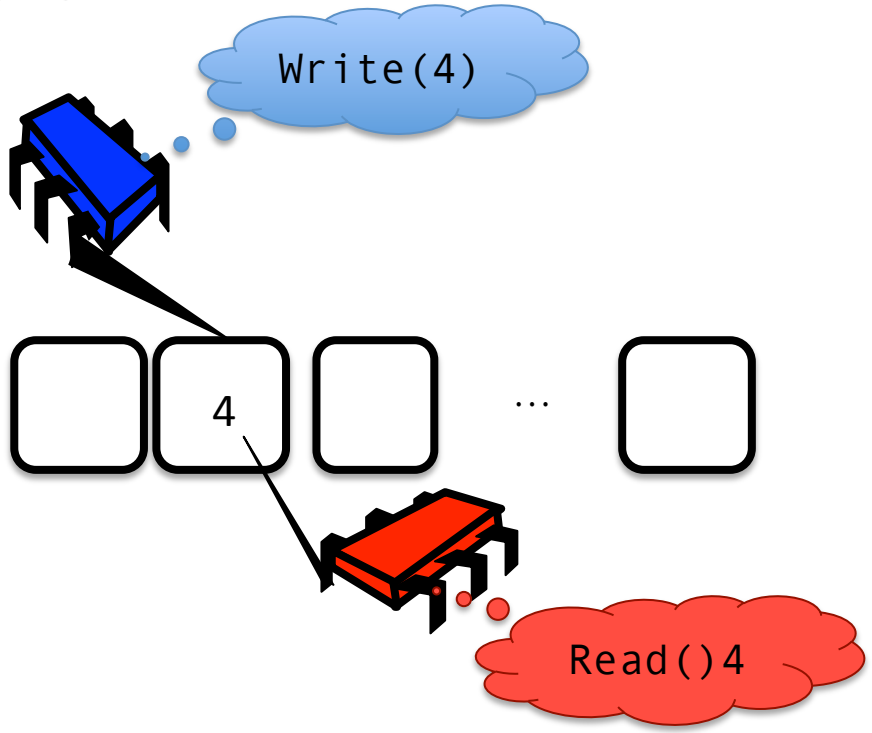
# Counter implementations

- Linearizable
- Wait-free : A process completes any operation in a finite number of steps (accesses to base objects)
- With sublinear step complexity :

the step complexity of an operation (e.g. INC) istance is the number of accesses to base objects (e.g. registers)

# Model

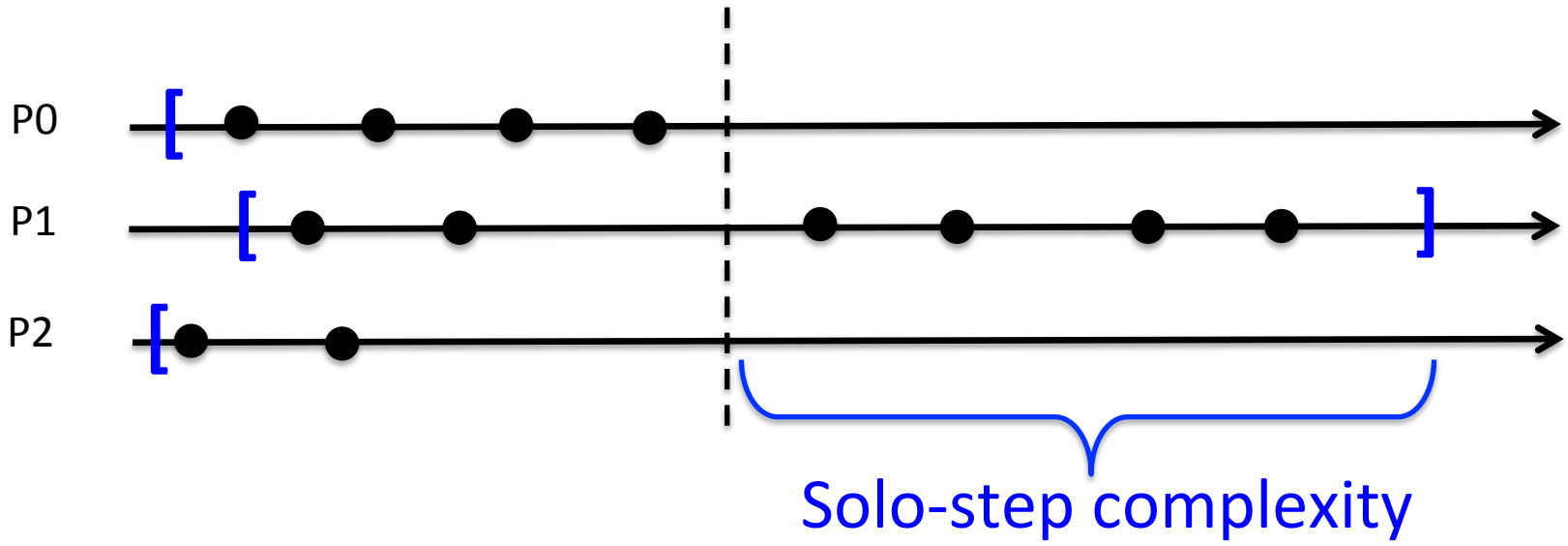
- System of n processes communicating via read/write shared memory (registers)
- Asynchronous
- Crash failure



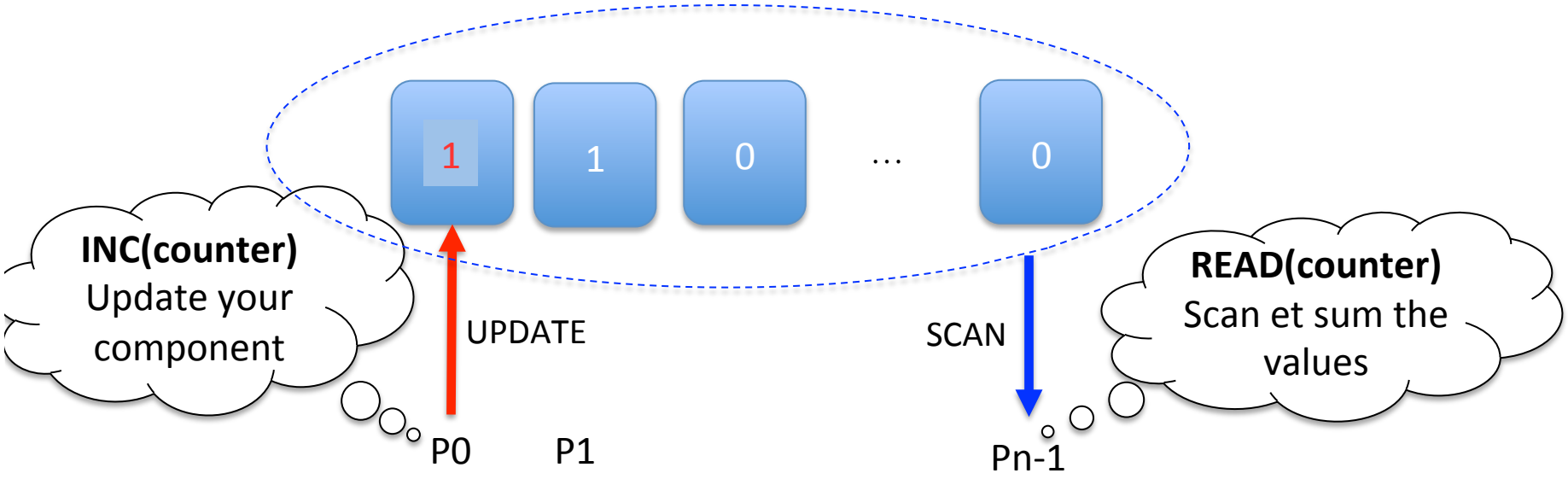
# Counter step compl

An operation is guaranteed to return if it runs in absence of step contention for sufficiently long

- Any linearizable obstruction-free implementation of a counter from read/write registers has Ω(n) solo-step complexity [Jayanti et al. SICOMP 2000]



# A O(n) counter implementation




Based on a snapshot object allowing each process to

- update its own component (by invoking an UPDATE)
- obtain an atomic view of all components (by invoking a SCAN)

O(n) snapshot [Inoue and Chen WDAG1994]

# Restricted-use objects

- Aspnes et al. [AAC12] observed that the linear lower bound can be beaten if we restrict the way objects are used
  - limit the number of operations that can be performed on the object
  - bound the value an object can support
  - …

[AAC12] James Aspnes, Hagit Attiya, and Keren Censor-Hillel, Polylogarithmic concurrent data structures from monotone circuits, J. ACM 59 (2012), no. 1, 2:1–2:24

# m-valued counter [AAC12]

- An m-valued counter is a counter that takes values in {0,1,…,m-1} for some integer m
- An m-valued counter for n processes can be implemented with
  - O(lognlogm) step complexity for Increment operations
  - O(logm) step complexity for Read operations

# Max-register r

Sequential specification

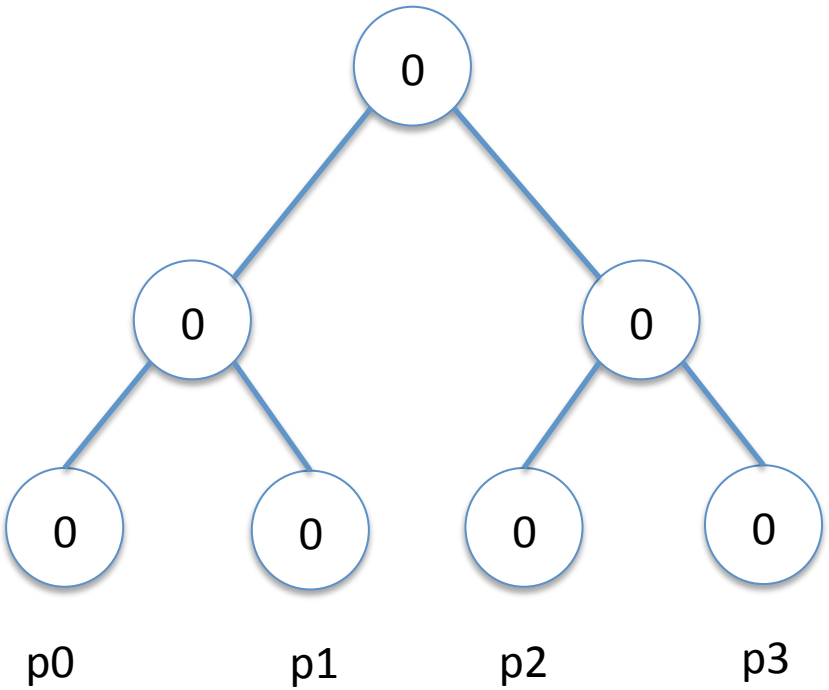
- Write(r,v) : writes the value v to r
- ReadMax(r)v : returns the maximum value previously written into r

`Write(r,4)` `Write(r,1)` `ReadMax(r)4`

# Bounded max-register

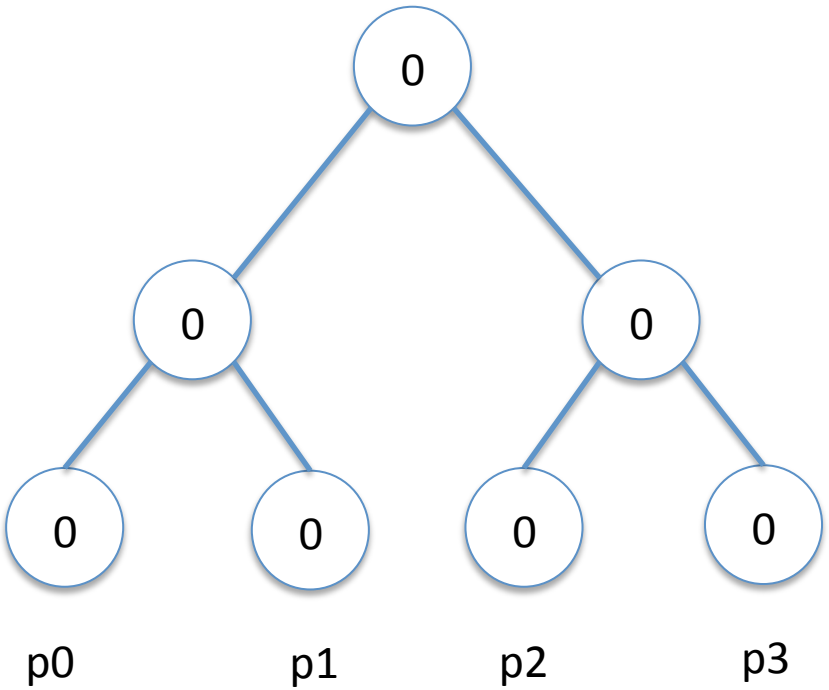
- Bounded max-register : Assume values from {0,1,…,m-1} for some integer m
  - m is called the size of the max-register
- Can be implemented with O(logm) step complexity for both ReadMax and Write operations [AAC12]

# m-value counter implementation



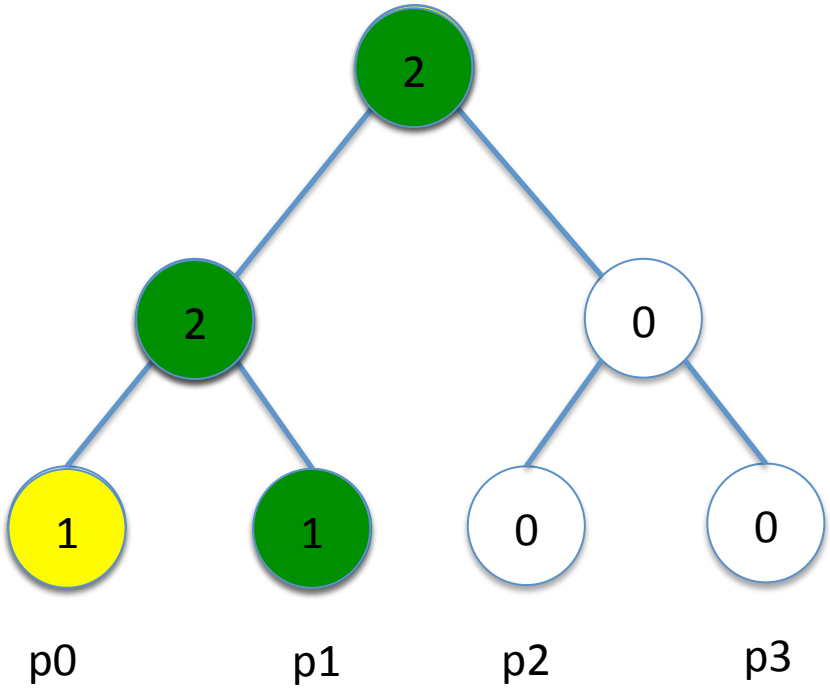

- The counter is structured as a binary tree of max-registers
- Each process is associated with a leaf

# m-value counter implementation




- INC(c)
  1. increment the value in your leaf
  2. Follow the path to the root and for each node r in the path do
     - Read the value of the left and right child of r
     - Write into r the sum of these values
- READ(c) : read the value at the root

# m-value counter implementation : a scenario




p0 **INC(c)**

p1 **INC(c)**

# Bounded max-register implementation

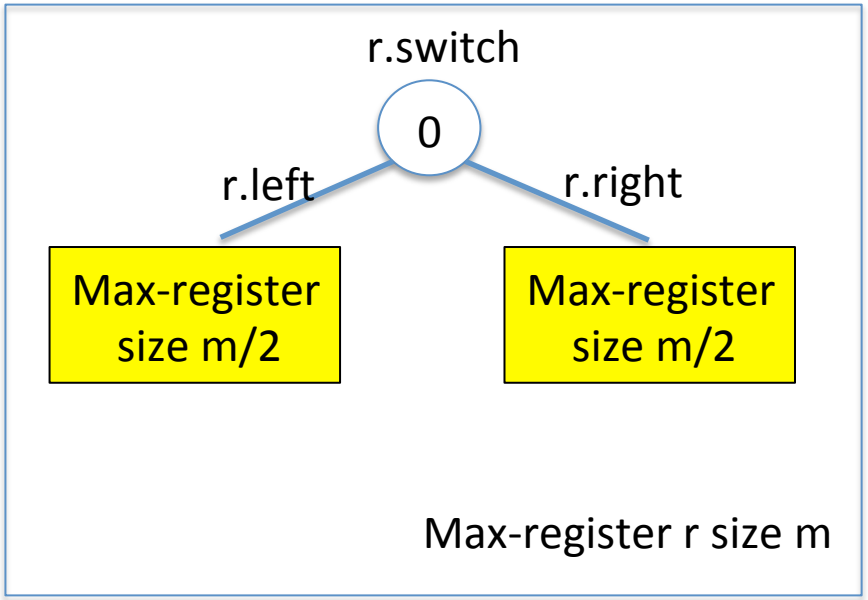
**WriteMax(r,v)**

If v<m/2  WriteMax(r.left,v)

else

WriteMax(r.right,v-m/2)

r.switch ← 1




**ReadMax(r,v)**

If r.switch=0

return ReadMax(r.left)

Else return ReadMax(r.left)+m/2

# Conclusion

- Wait-free read/write counter requires $\Omega(n)$ steps in the worst case [Jayanti et al. SICOMP 2000]
- We can have polylogarithmic step complexity if we restrict the way we use it [AAC12]
- Can we have polylogarithmic amortized step complexity in executions of arbitrary length?