Automatic Realizations of Statically Safe Intra-Object Synchronization Schemes in MP-Eiffel

Miguel Oliveira e Silva (mos@det.ua.pt)

IEETA - Department of Electronics, Telecommunications, and Informatics
University of Aveiro

First International Symposium on Concurrency, Real-Time and Distribution in Eiffel-like Languages, 2006

Outline

Why an Alternative Approach to SCOOP?

- Provide both models of inter-processor communication:

```
x.f(y)
```

  - Shared memory (Current = processor(x))
  - Message passing (Current /= processor(x))
- Intra-object concurrency
- Concurrent contracts
- Avoid redundancy within concurrent calls (wrapper routines with duplicated preconditions)
- Interesting and fun work for a PhD

Basic Definitions

- Processor (Writer, Reader)
- Concurrent objects
- Synchronization requirements:
  - Intra-object synchronization (server synchronization)
  - Conditional synchronization
  - Inter-object synchronization (client synchronization)
- Concurrent condition
- Concurrent assertion
- Concurrent object availability

$$COA_x = \frac{N_c}{N_x}$$

x: readers/writers/all, c: maximum concurrent processors

MP-Eiffel

- Explicit concurrent objects
- Static safety
- Abstract processors
- Both models of inter-processor communication
- Abstract synchronization
- Concurrency control language

Shared Memory Inter-Processor Communication

- **shared** and **remote** objects
- shared entities:

  `job_queue: shared QUEUE[JOB]`
- remote entities:

  `weather: remote WEATHER`
  - Identify pure query services
  - attributes
  - functions without assignments to attributes and calls to impure routines.
  - all possible redefinitions are also required to be pure
- Synchronous exceptions (same processor)

Message Passing Inter-Processor Communication

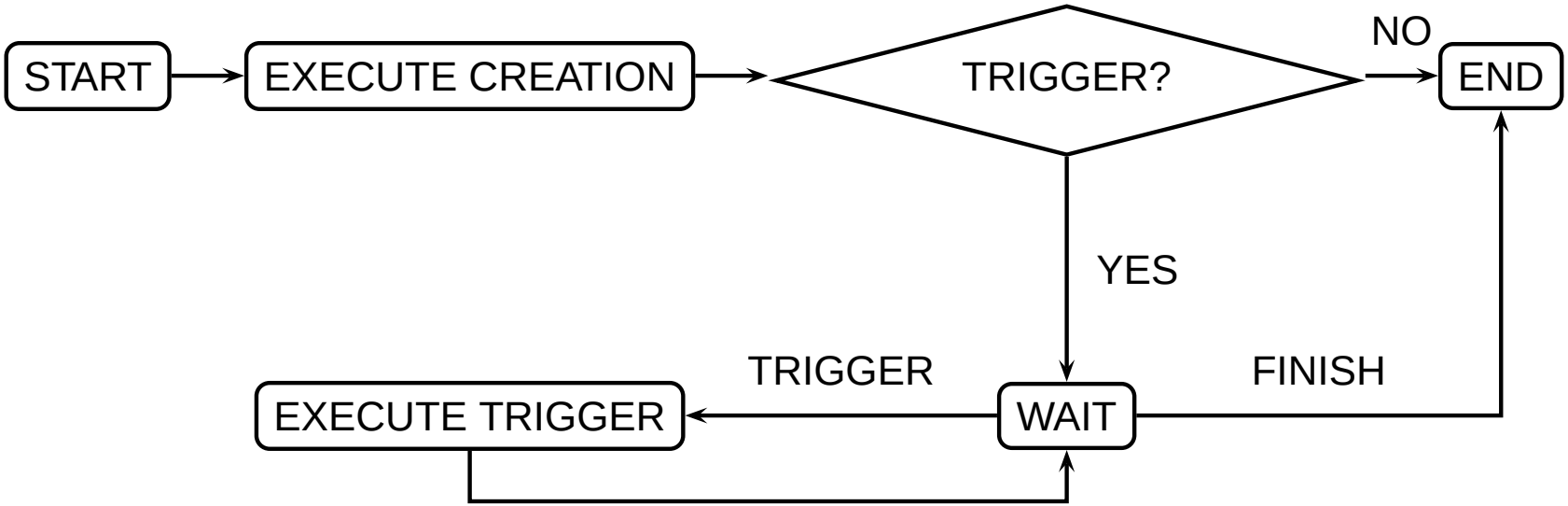
- triggers
- message sender:

  trigger `x.f`
- x is required to be a remote entity
- message receiver: explicit trigger interface

  trigger `{A_CLASS} a,b,c`
- Synchronous sequential preconditions

Processor Life Cycle

- Created by a create call on a remote entity

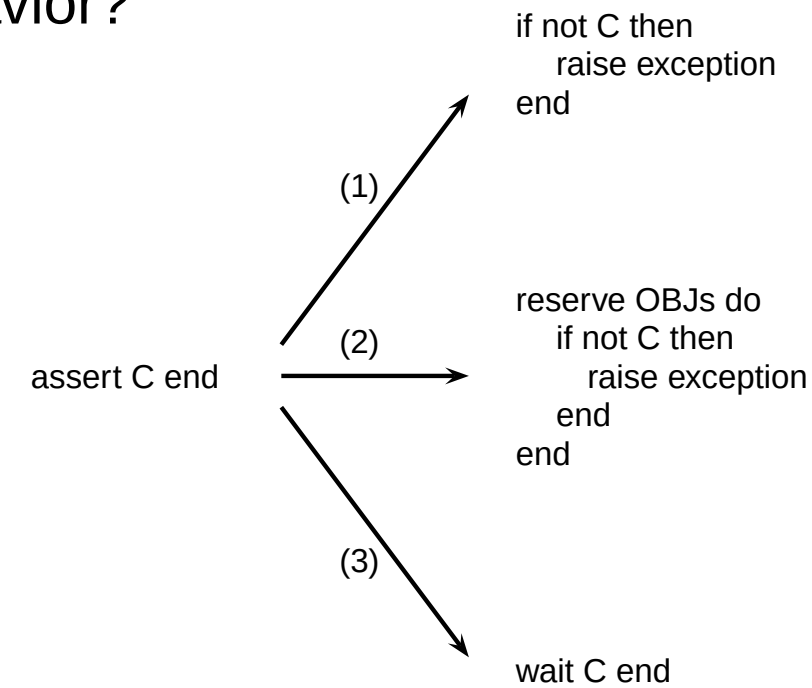


Concurrent Object Correctness

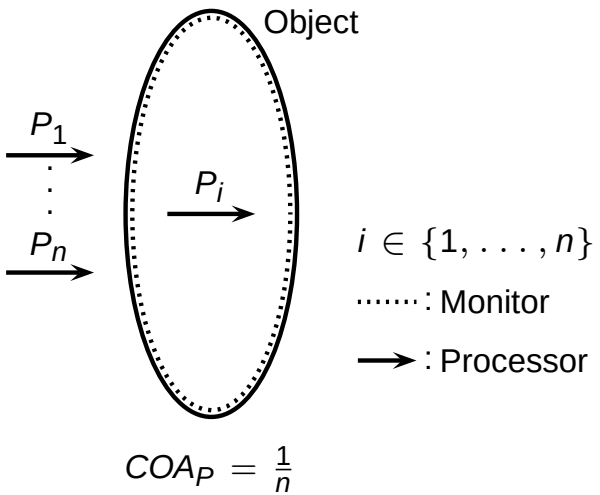
- Abstract Data Type
- Stable times client usage ensure sequential correctness
- Public modifiable attributes forbidden
- Linearizability (Herlihy, 1990)
- Class contracts
- Total object covering
- Processor attributes (not yet adopted)

Concurrent Contracts

- Correct behavior?



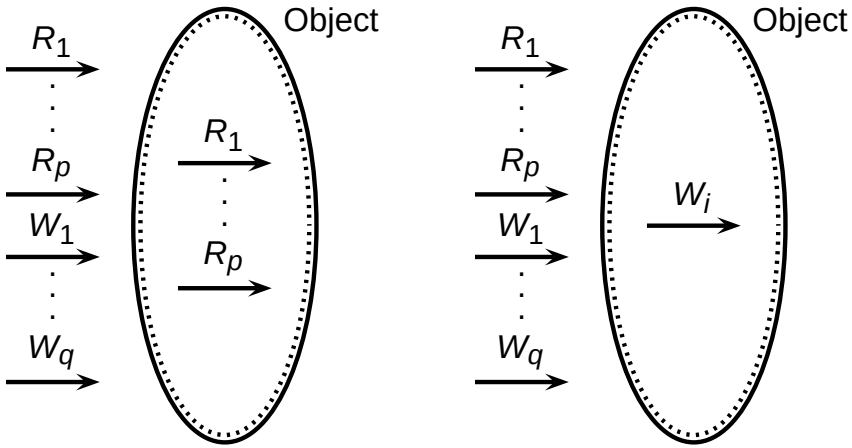
Monitor



$COA_P = \frac{1}{n}$

- Simplest synchronization scheme
- Lowest concurrent availability
- Requirements:
  - Public service identification

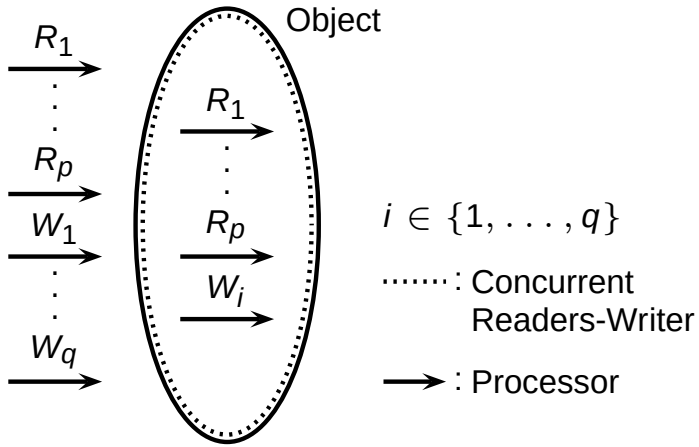
Readers-Writer Exclusion



$COA_R = 100\%$, $COA_W = 0\%$ $COA_R = 0\%$, $COA_W = \frac{1}{q}$

- Taking advantage of command/query separation
- Requirements:
  - Query (pure) services identification

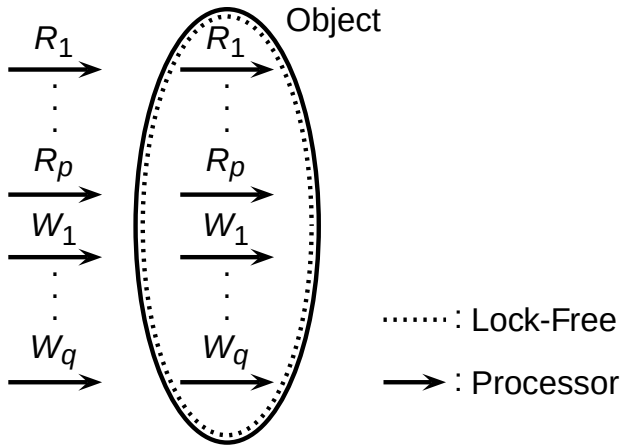
Concurrent Readers-Writer



$COA_R = 100\%$ $COA_W = \frac{1}{q}$

- Similar requirements as readers-writer exclusion
- Starvation of readers is possible
- Wait-free algorithm (Peterson) for objects with a unique writer (concurrent objects attached to remote entities)
- Repeatable pure queries

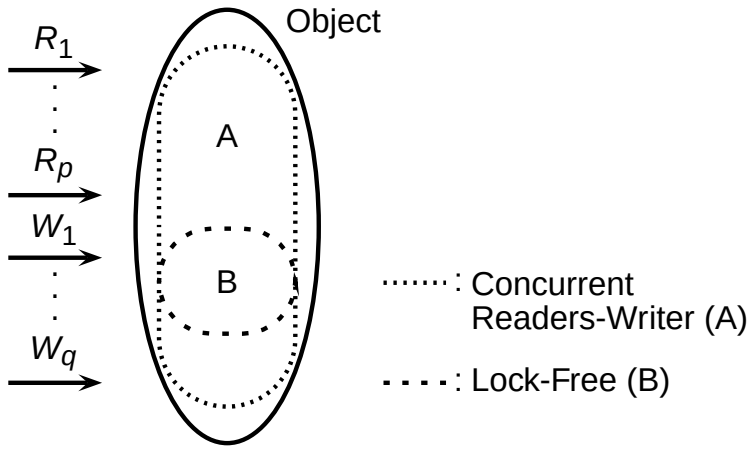
Lock-Free and Software Transactional Memory



$COA_R = 100\%$ $COA_W = 100\%$

- No blocking
- Immune to deadlocks and processor failure
- Complex schemes
- Requirements:
  - Object state replication
  - Repeatable services

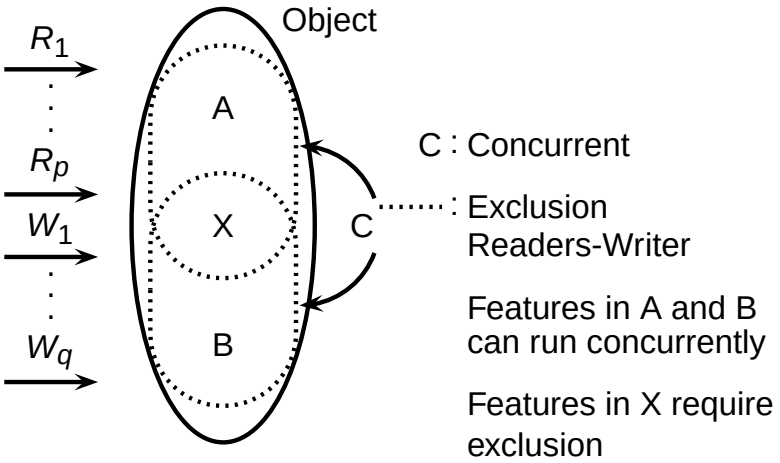
Mixed Exclusion Schemes



$COA_A = COA_{CRW}$ $COA_B = COA_{LF}$

- Mutual exclusion between schemes
- Solves the problem of intra-object and inter-object object synchronization integration
- Requirements:
  - Total object covering
  - Safety of each scheme

Mixed Concurrent Schemes



- Unsafe in general
- It may apply if concurrent services handle don't use the same attributes. However,
- linearizable invariant verification is required!

Inter-Object Synchronization

- SCOOP:
  - Locking is attached with formal arguments
  - It is a redundant approach (wrapping services and possible precondition duplication)
- Certain instructions expect sequential consistency
- Example:

```
if buffer.empty then
   -- (1)
   -- (2)
   ...
else
   -- (4)
   -- (5)
   ...
end
```

- Preconditions, iterative instructions

Other issues

- Conditional synchronization implementation
  - One condition variable per object, and signal all processors at the end of the object's public routines
  - Take advantage of commands and queries (MP-Eiffel)
- Sub-type polymorphism

Summary