Reactive Garbage Collector

Frédéric Boussinot
INRIA Méditerranée
http://www-sop.inria.fr/members/Frederic.Boussinot

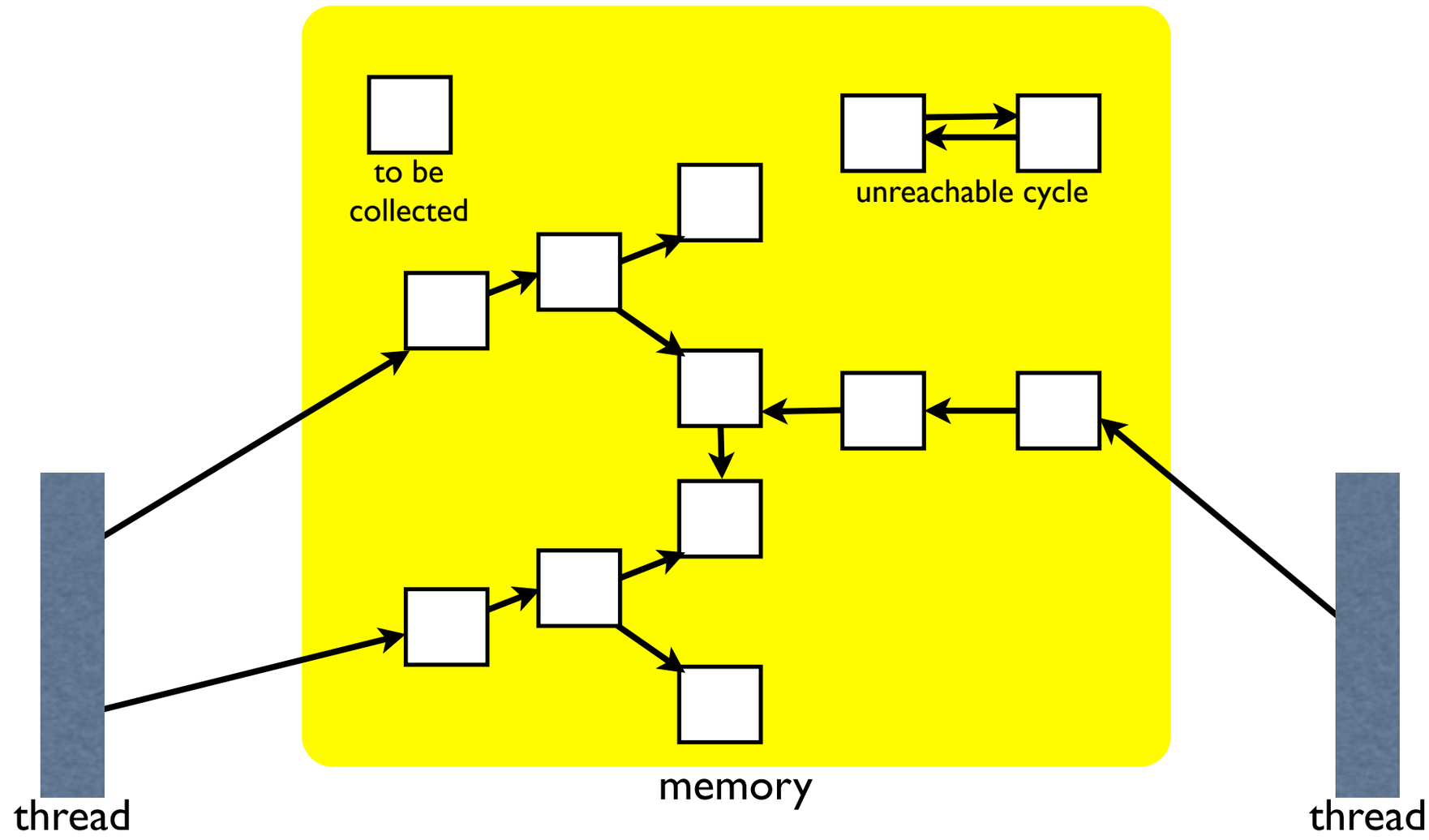
May 2009

ANR-08-EMER-010

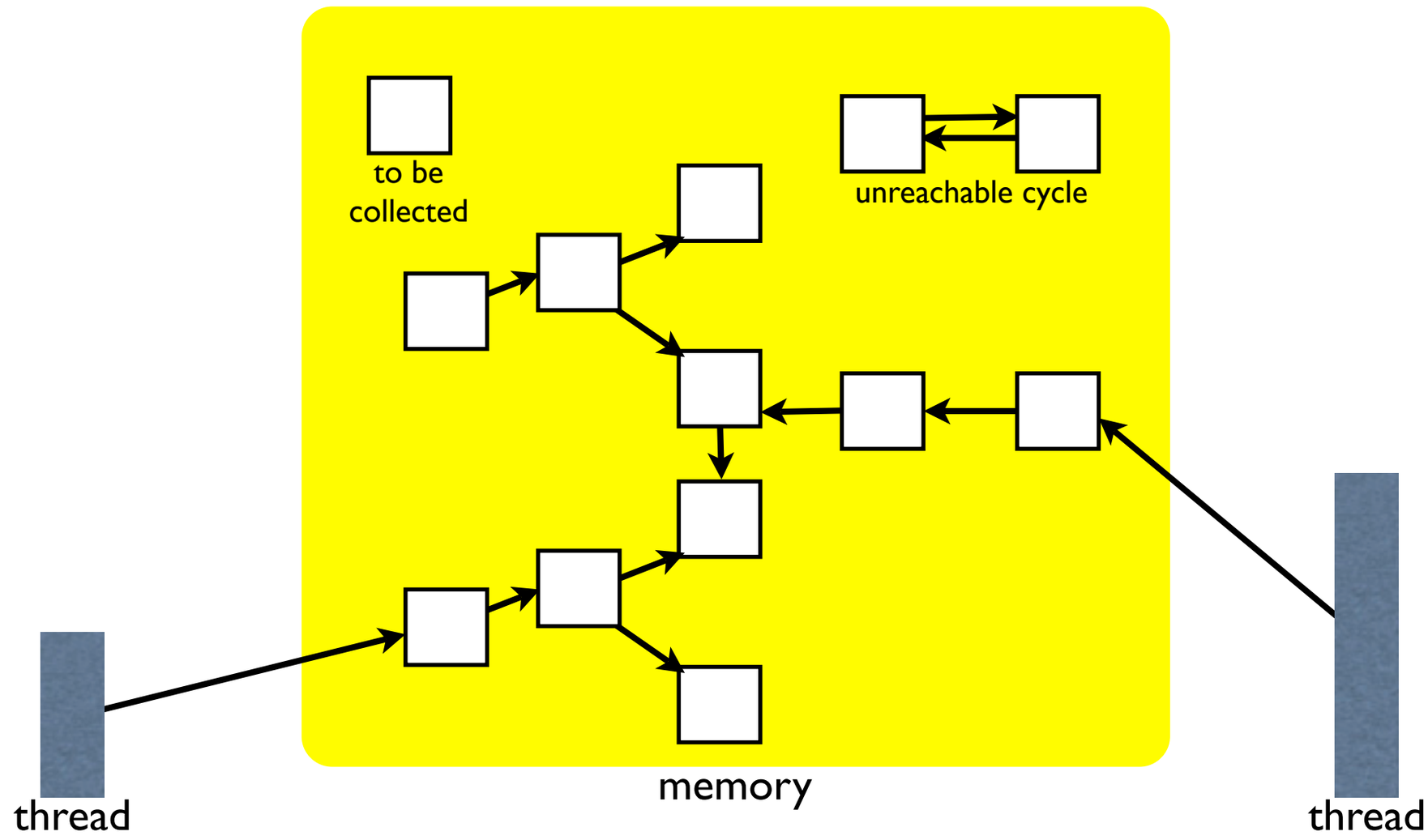
Standard GC

- Tracing / Reference counter based algorithms
  - Tracing-based GCs trace the reachable objects through the system memory and collect unreachable objects
  - Reference-counter GCs check the number of references to an object and collect it as soon as the number falls to zero
- Issues with reference counting:
  - Run-time overhead of reference counters management
  - Difficulties to handle cyclic data structures
- Issues with tracing:
  - Entire system must be suspended: "*stop-the-world*" phase
  - Whole memory may have to be considered
- Tracing usually prefered (Boehm's GC)

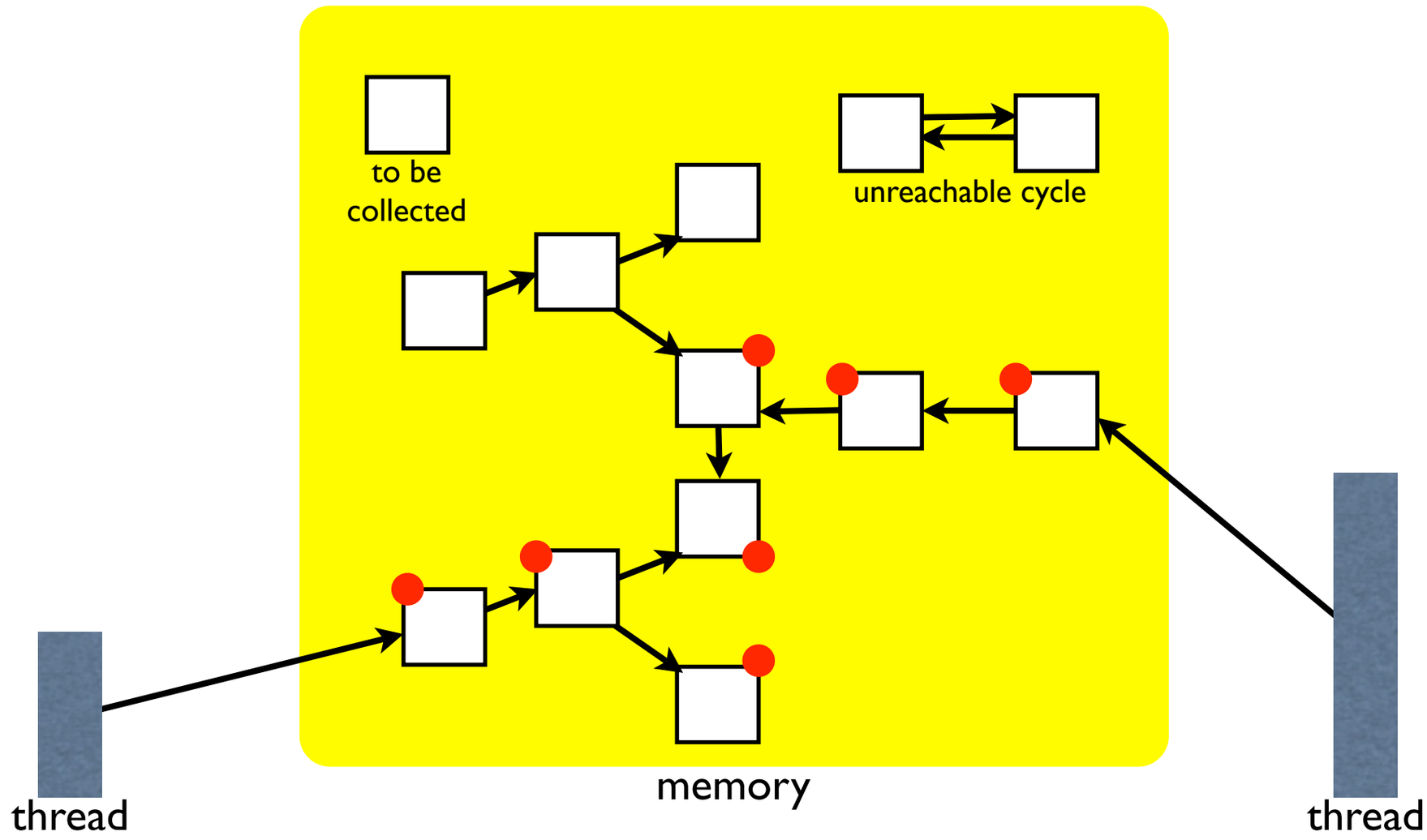
Tracing Example



Tracing Example

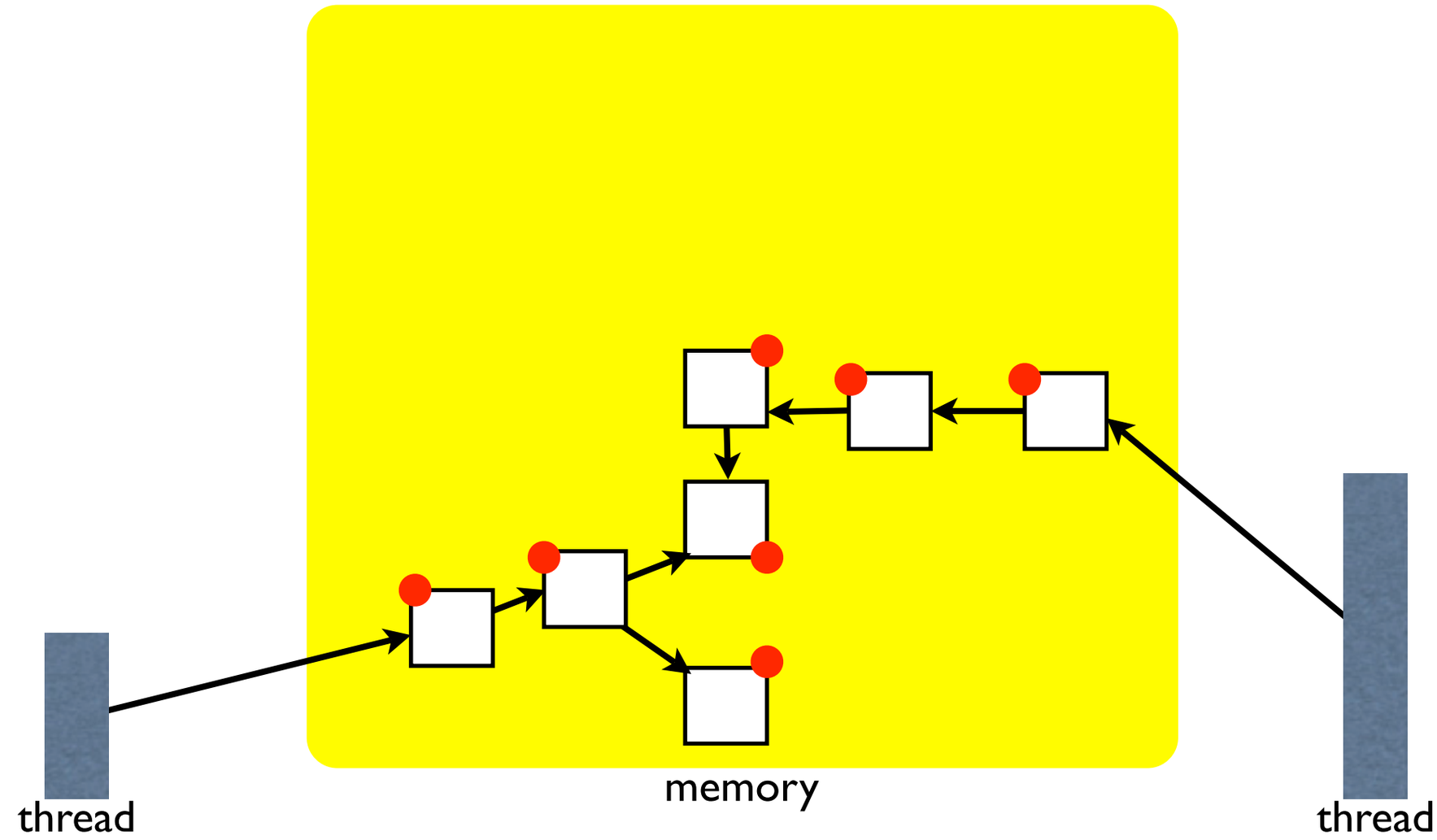


Tracing Example



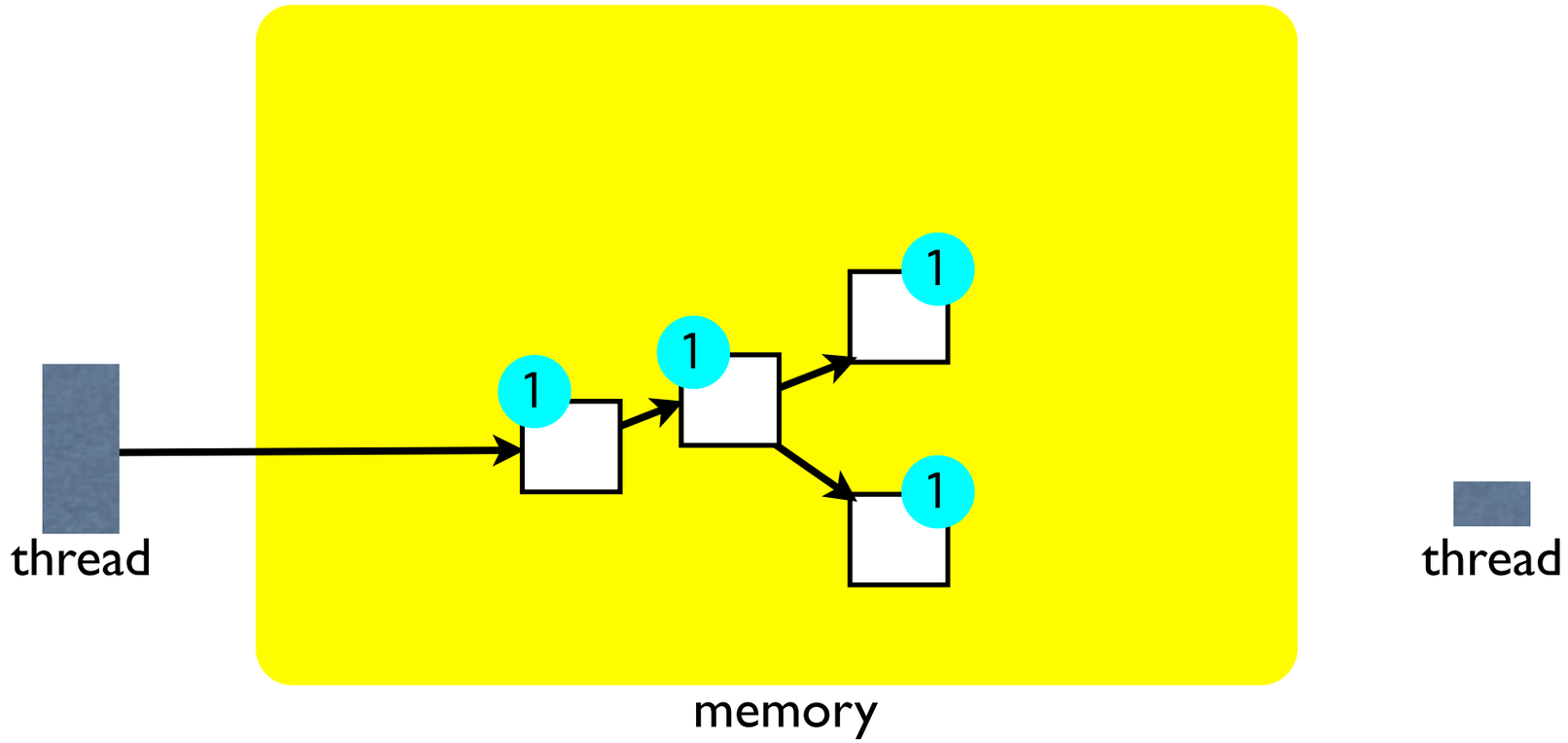
Marking of reachable objects

Tracing Example



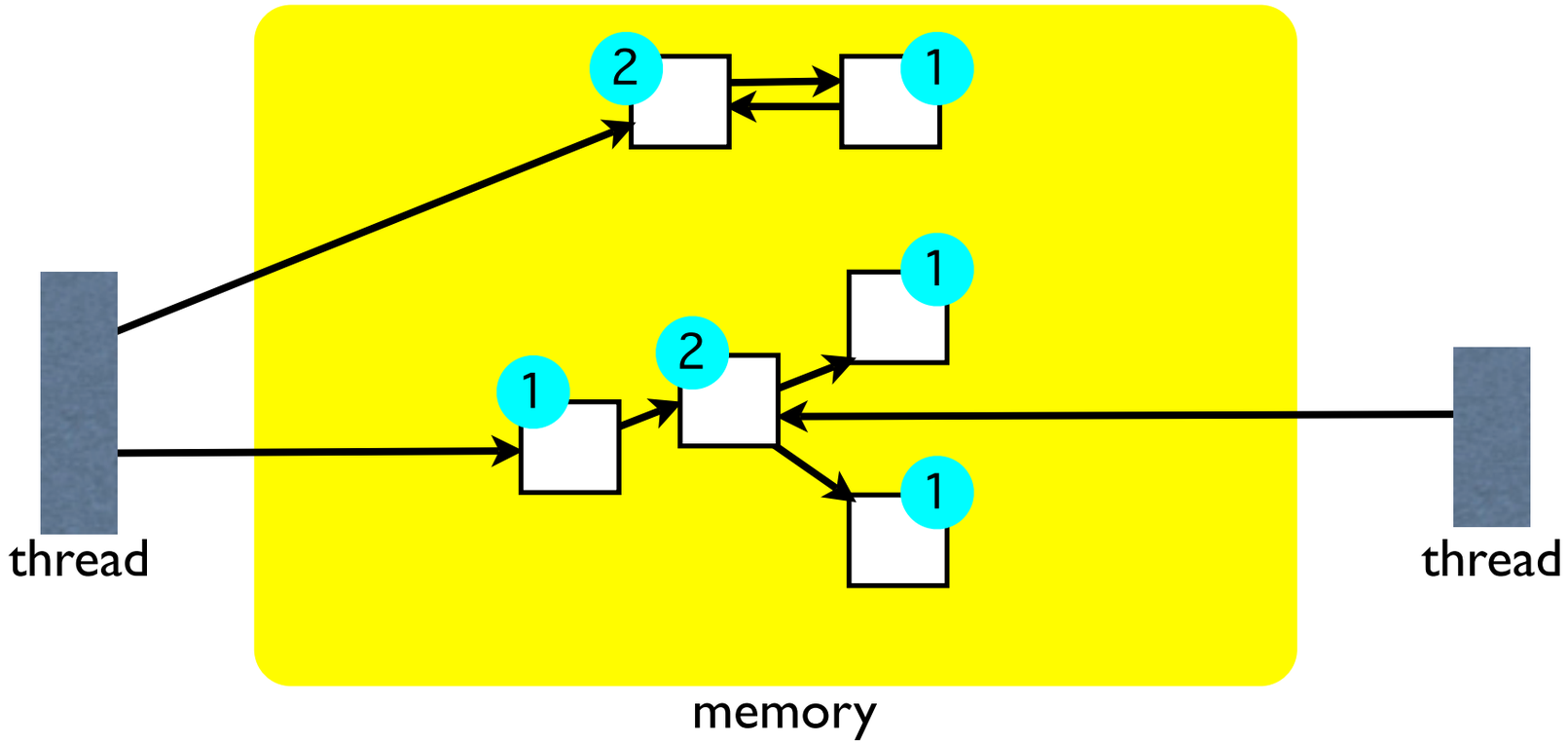
Marking from both threads before collection
Both threads suspended during collection

Ref Counting Example

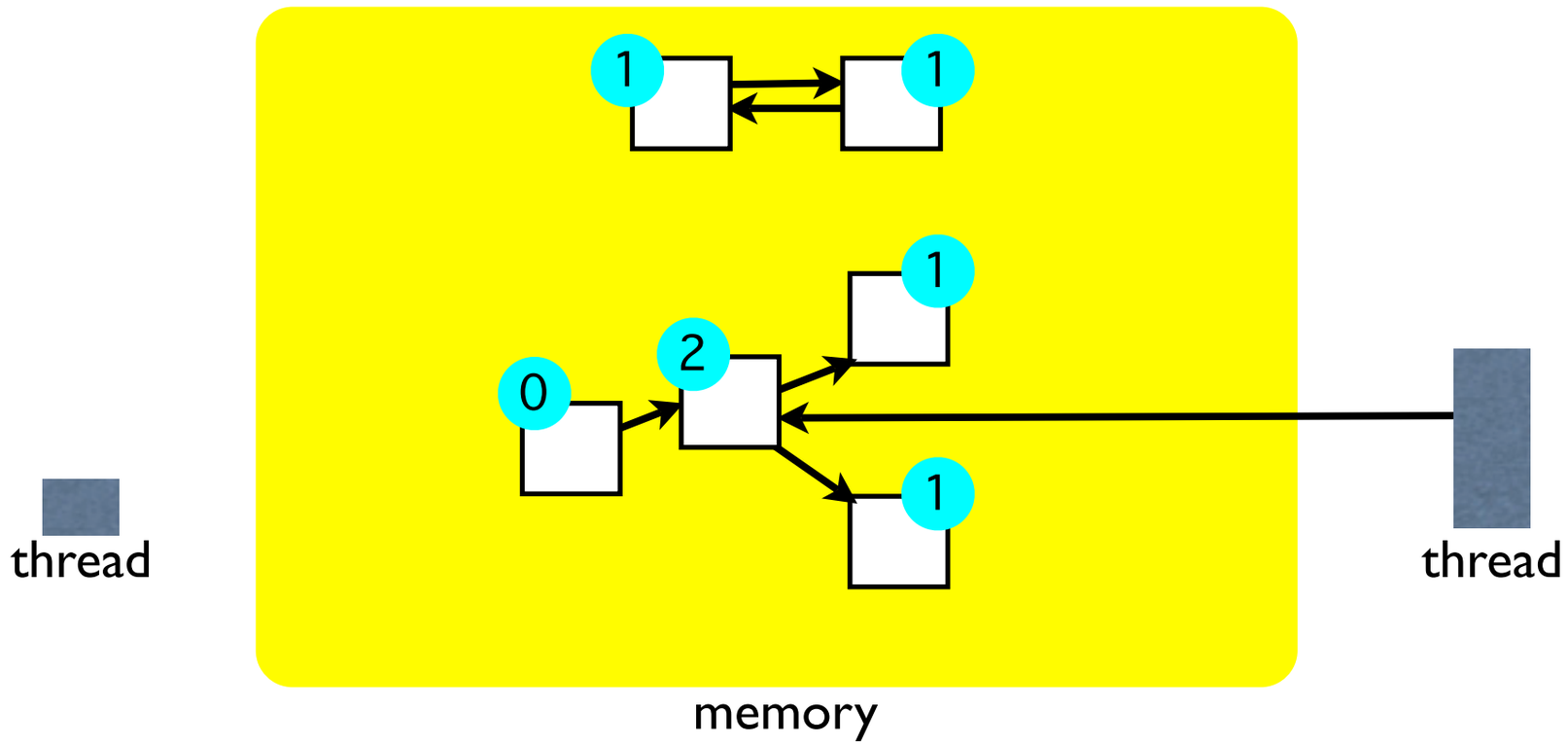


Count = number of objects pointing the target

Ref Counting Example

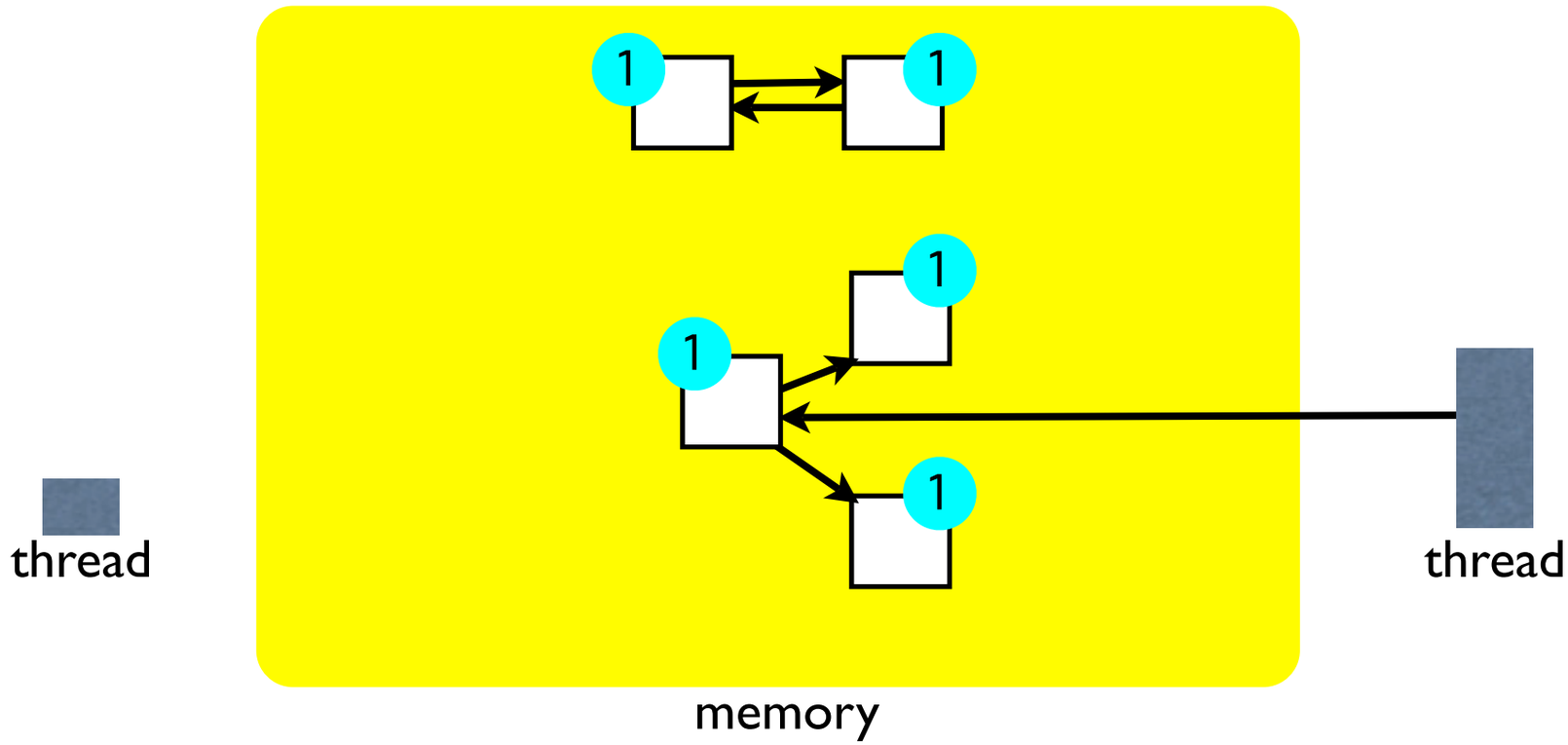


Ref Counting Example

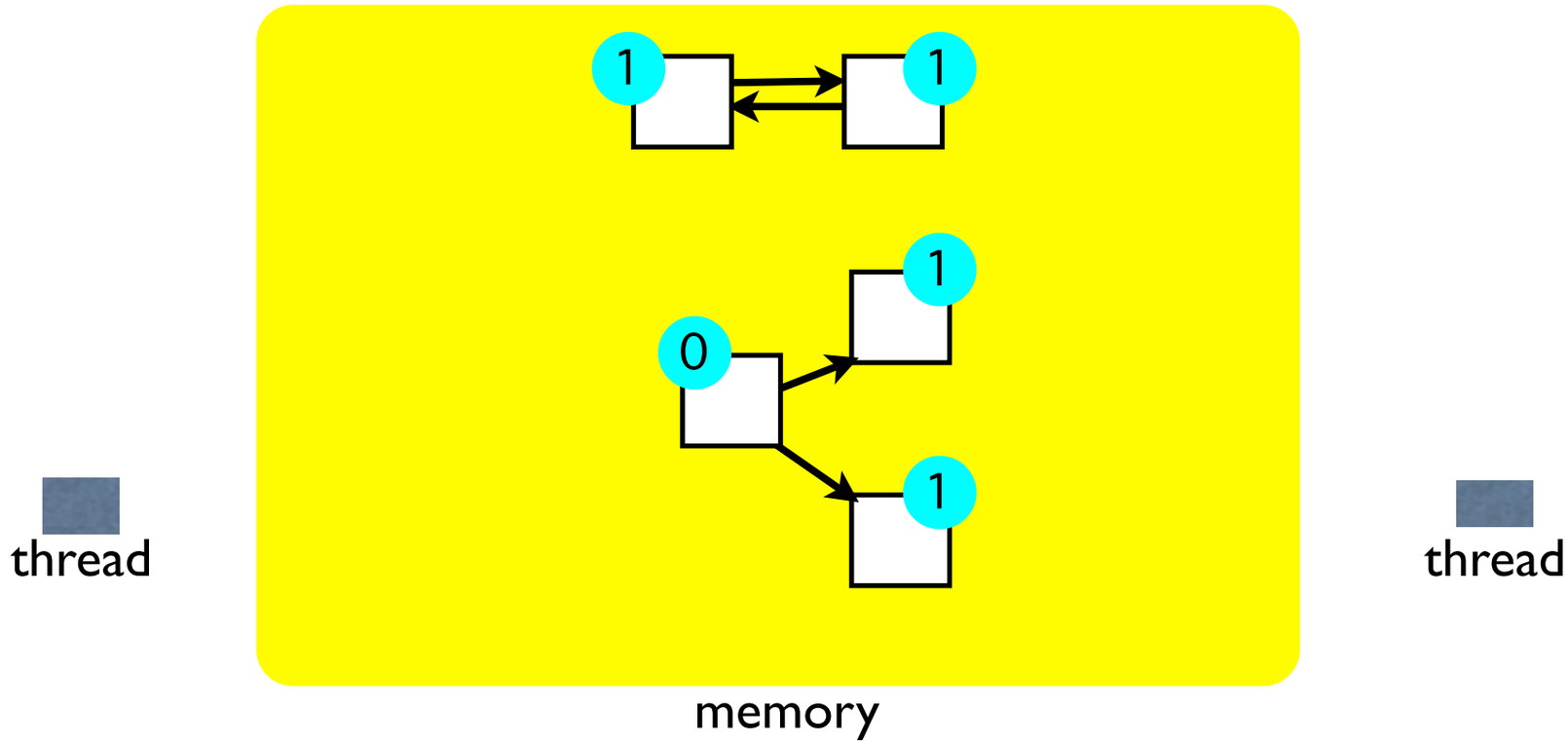


Collection when ref-counter falls to 0

Ref Counting Example

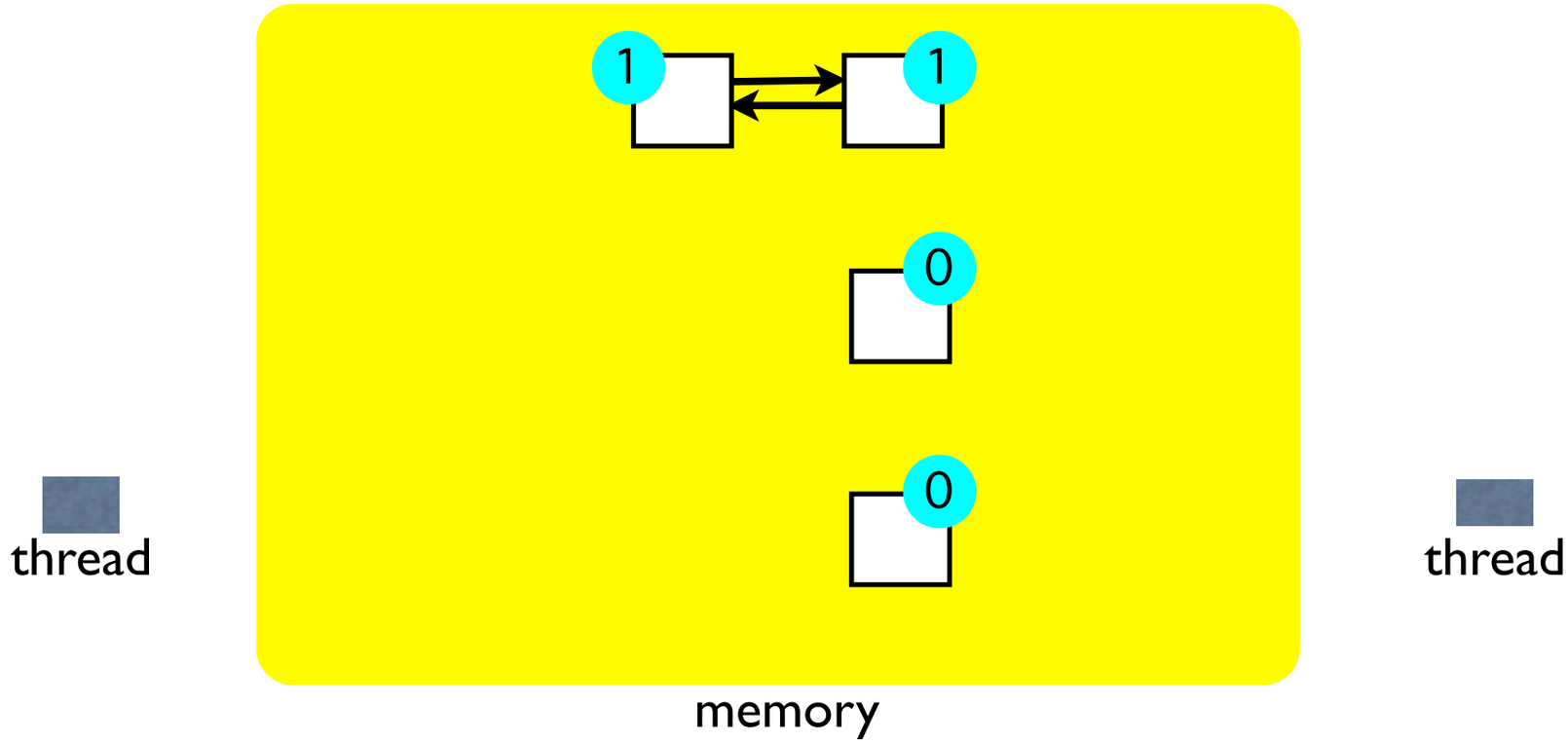


Ref Counting Example

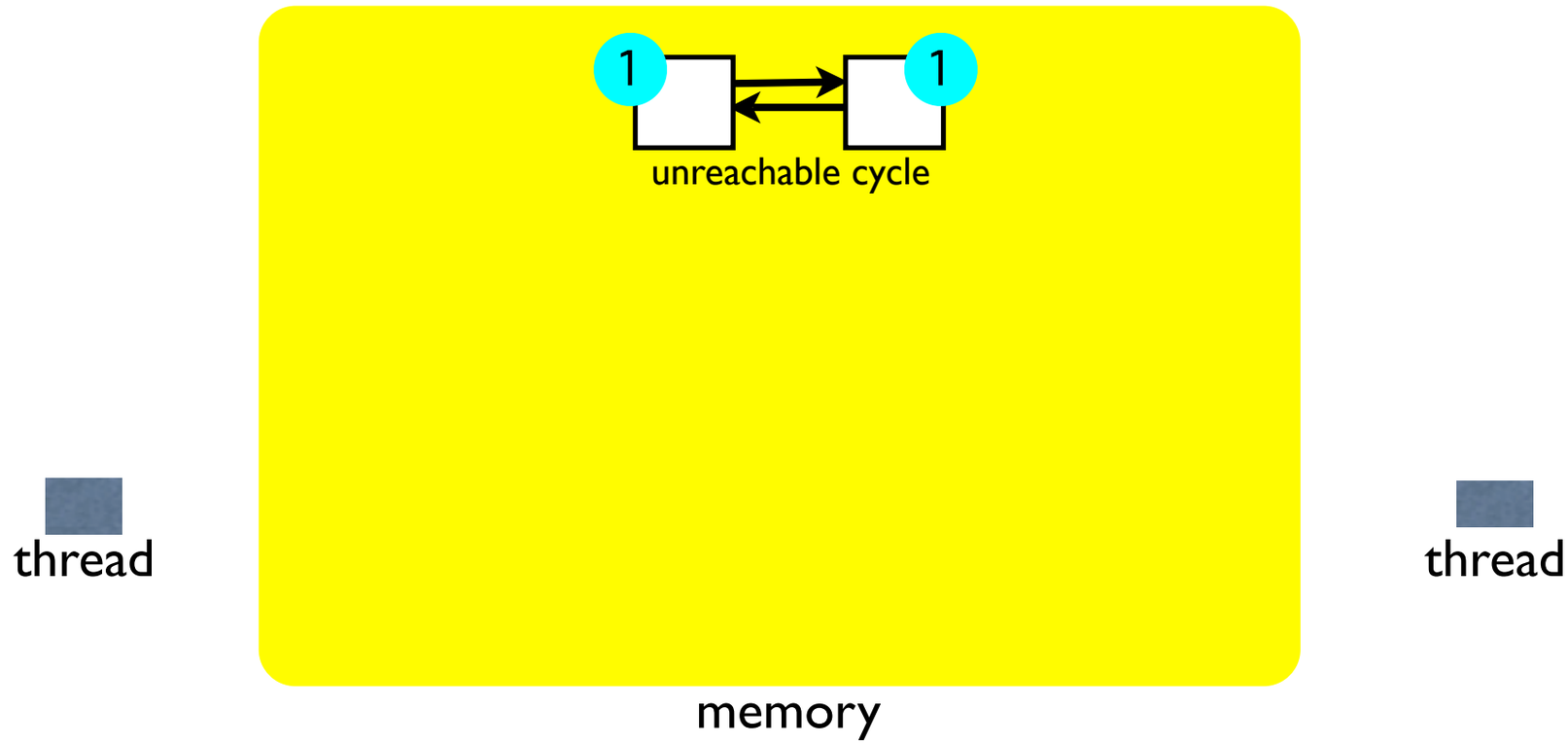


Transitive collection when counter = 0

Ref Counting Example

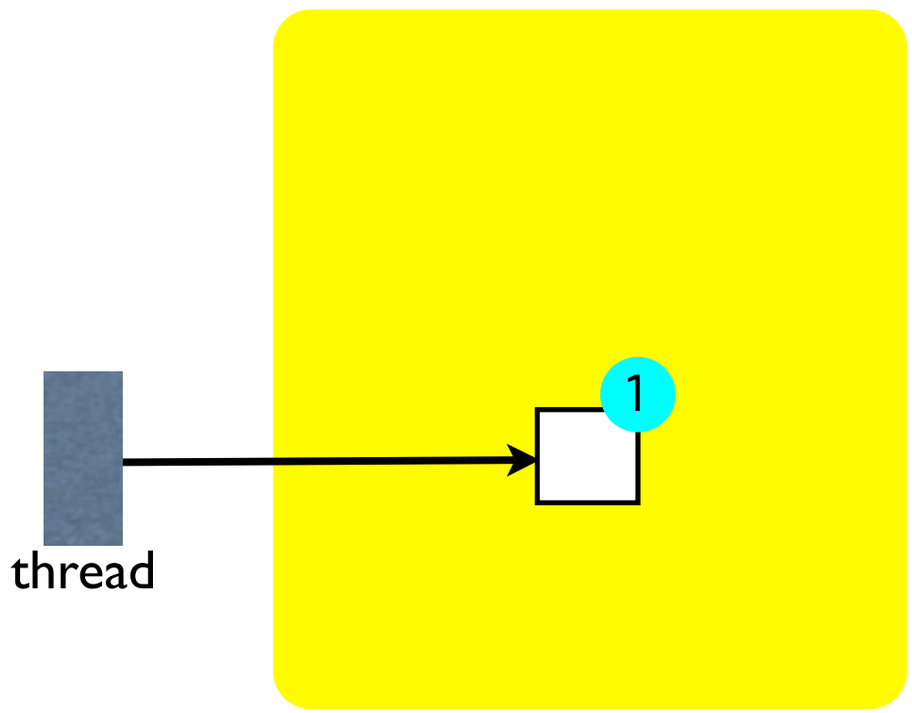


Ref Counting Example



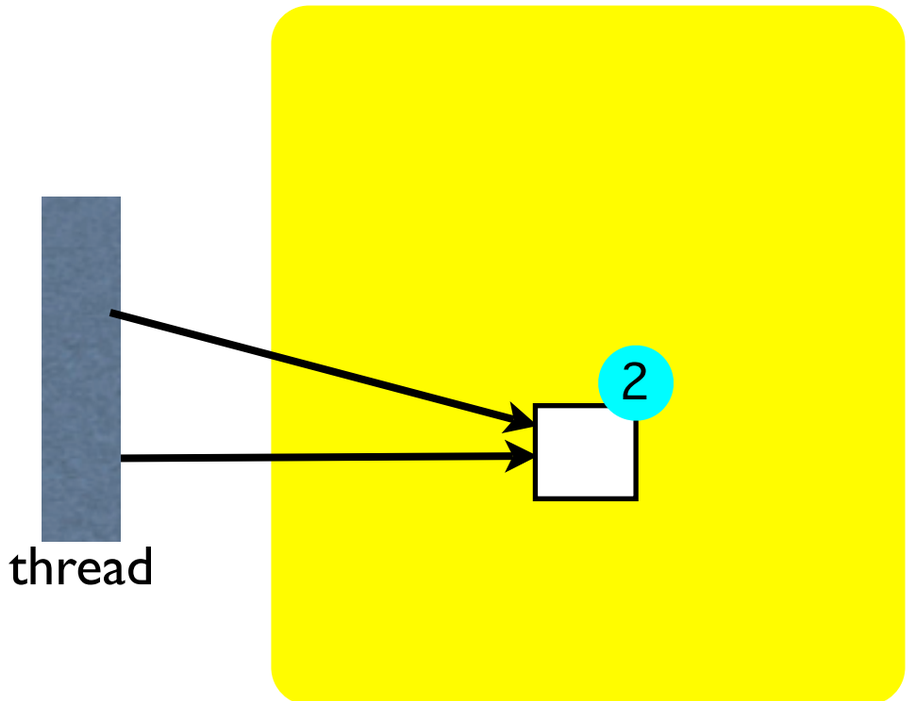
Uncollected cyclic structure

Ref Counting Example



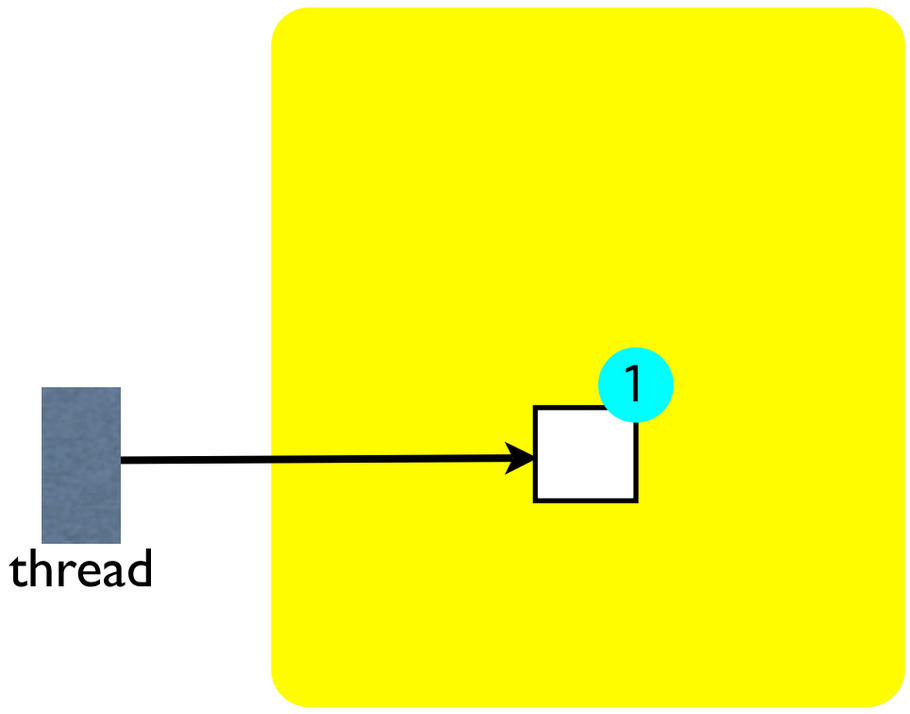
Overhead of ref-counter managment: useless operations performed

Ref Counting Example



increment the counter

Ref Counting Example



decrement the counter: previous increment was useless

GC Characteristics

- Concurrent: several concurrent mutators (avoid one big lock...)
- Generational: long living/short living objects processed differently
- Parallel: several parallel mutators/collectors (avoid data-races)
- *On-the-fly*: no “stop-the-world” phase (need to stop all the cores...)
- Distributed: collection of remote references

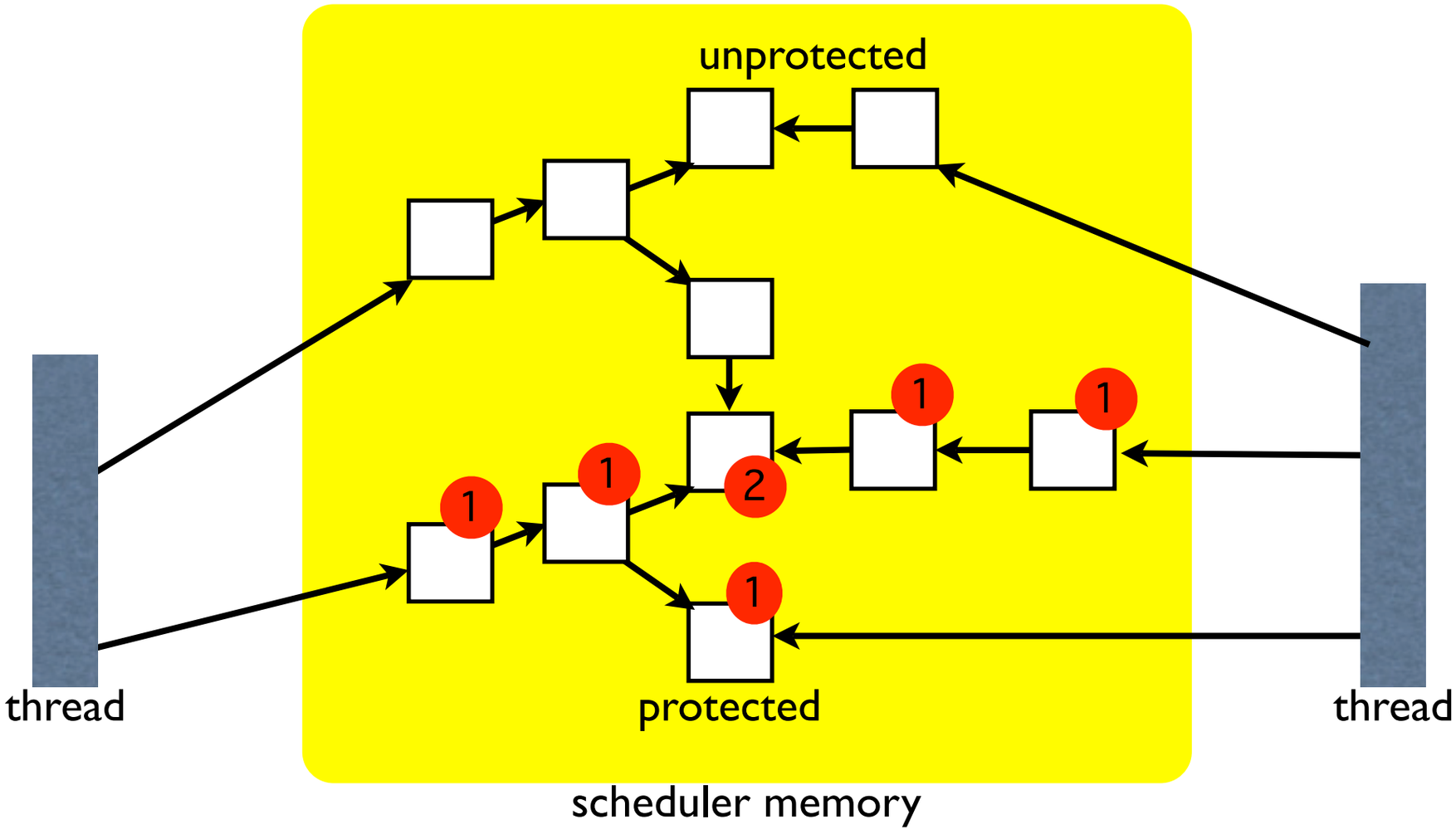
Reactive GC

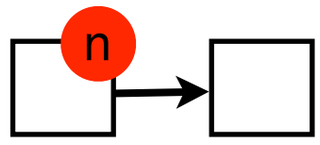
- Main objective: responsiveness
- Based on the notion of instant
- Mix of tracing and reference counting techniques
- Concurrent, parallel, generational
- On-the-fly: no “stop-the-world” phase
- Unable to handle general cyclic data structures

RGC Algorithm

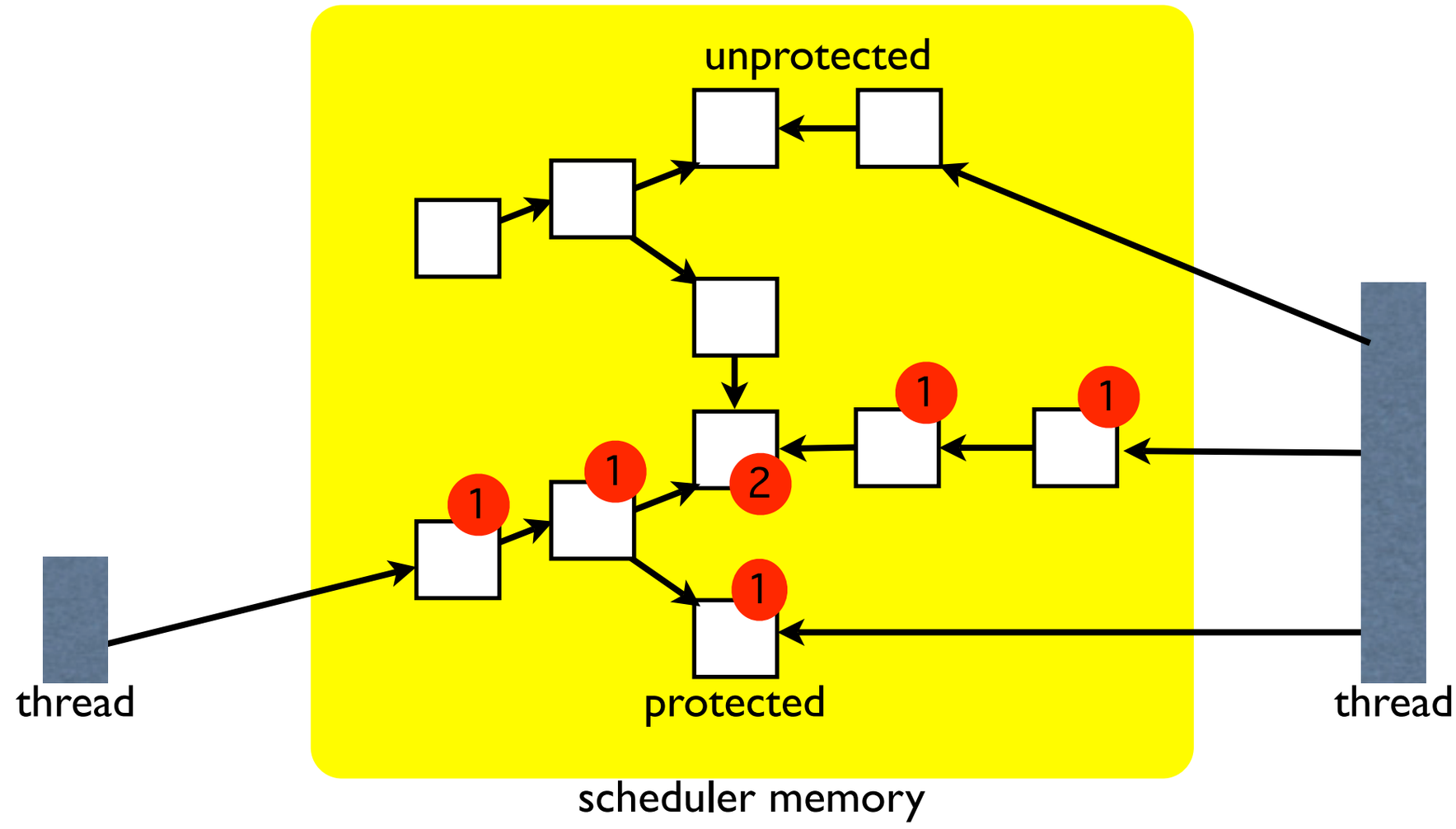
- References have a status protected/unprotected (initially unprotected)
- Turn protected when need to survive at the next instant (ex: created by a thread)
- Unprotected objects (ex: created by a function) can be safely collected at the end of the current instant by the scheduler to which they belong
- Protection level associated to protected objects. Collection when the protection level falls to 0 (as in reference counting based GCs)

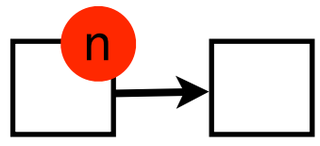
RGC Example



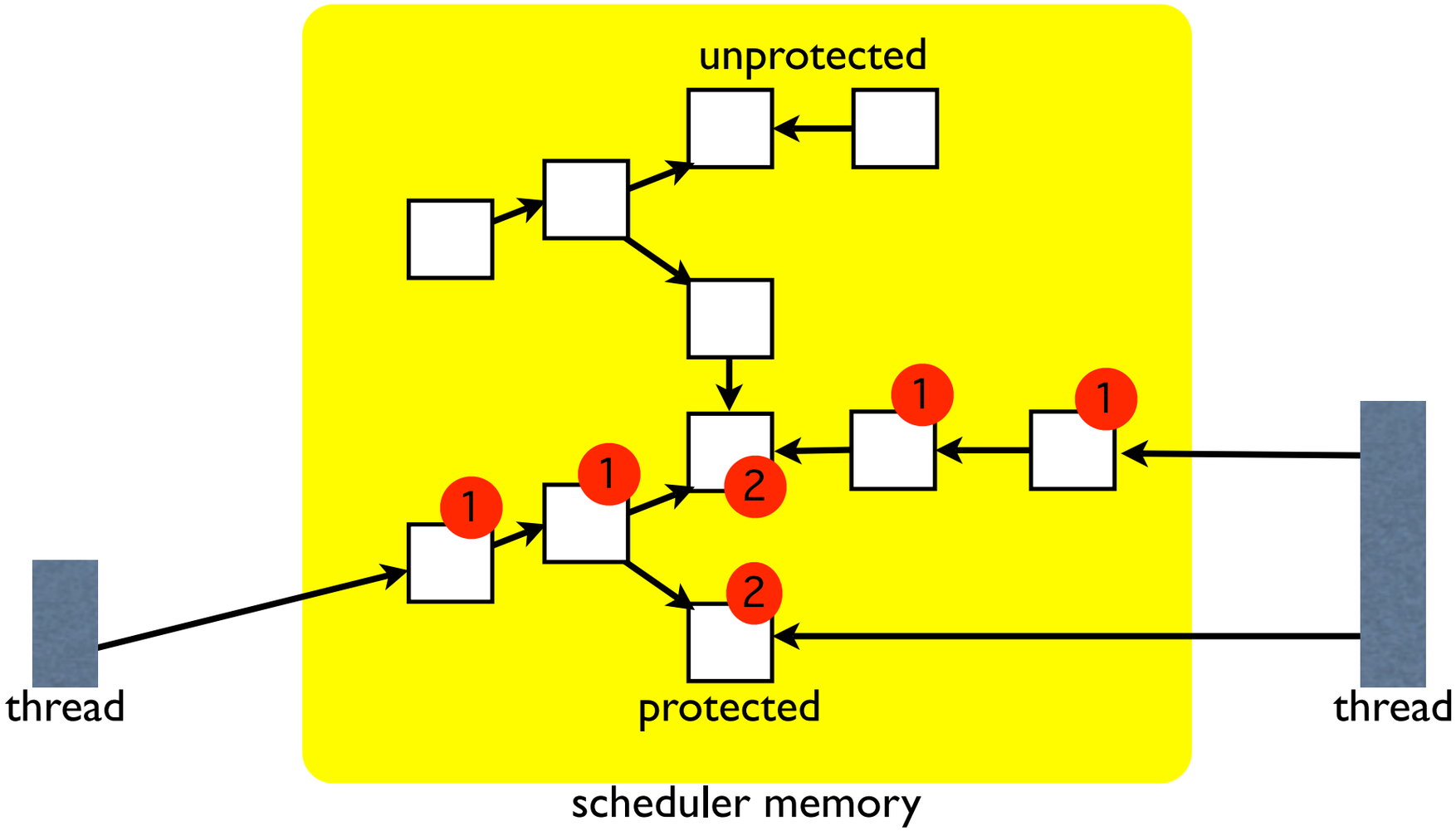
Not possible: 

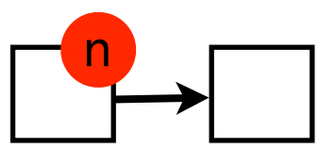
RGC Example



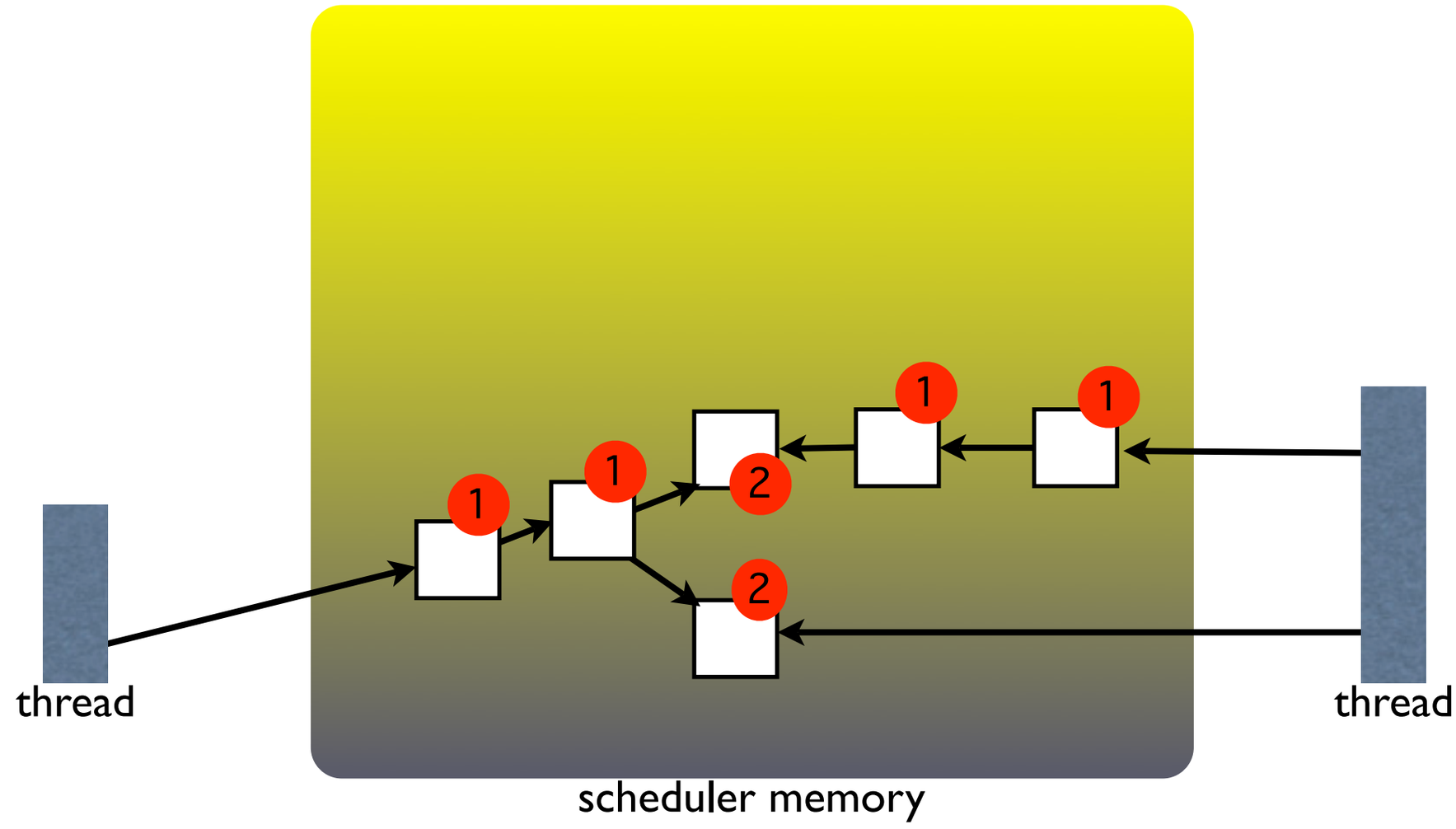
Not possible: 

RGC Example



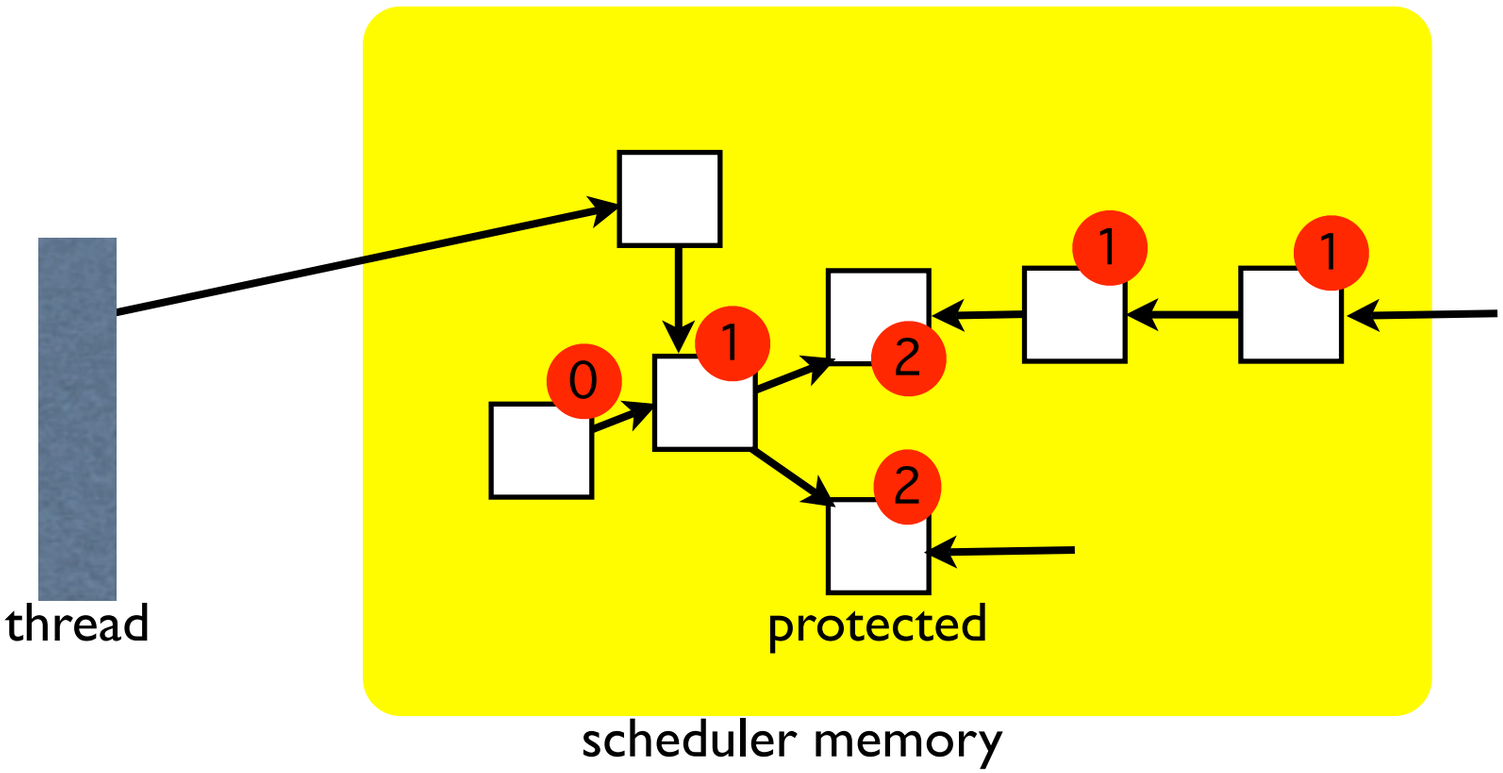
Not possible: 

RGC Example



Collection of unprotected at the end of instant

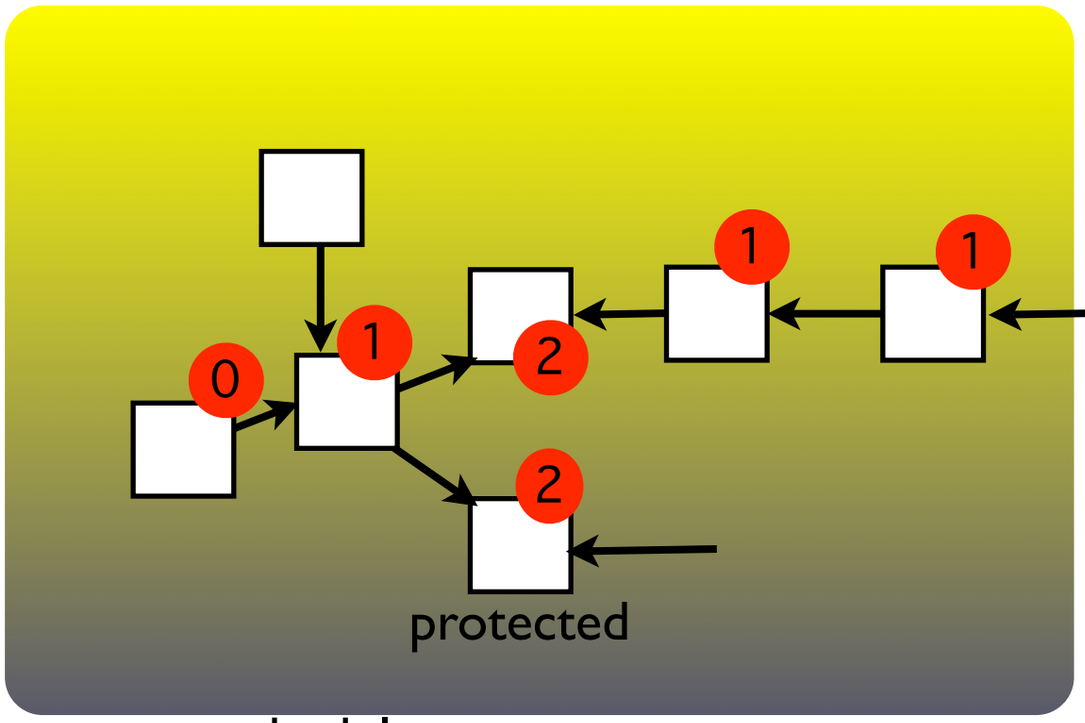
RGC Example



Collection of protected (with counter = 0) not before the end of instant

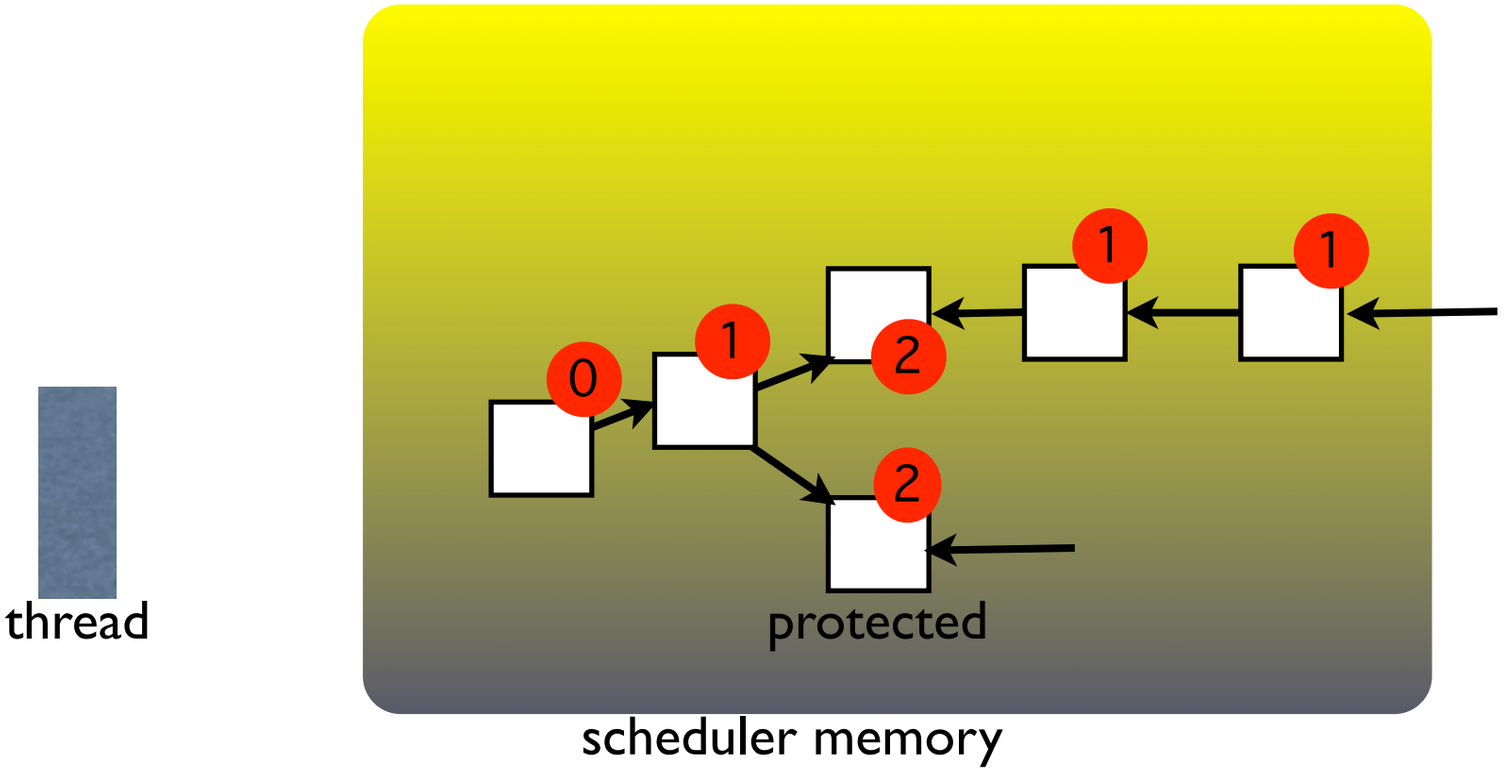
RGC Example

thread

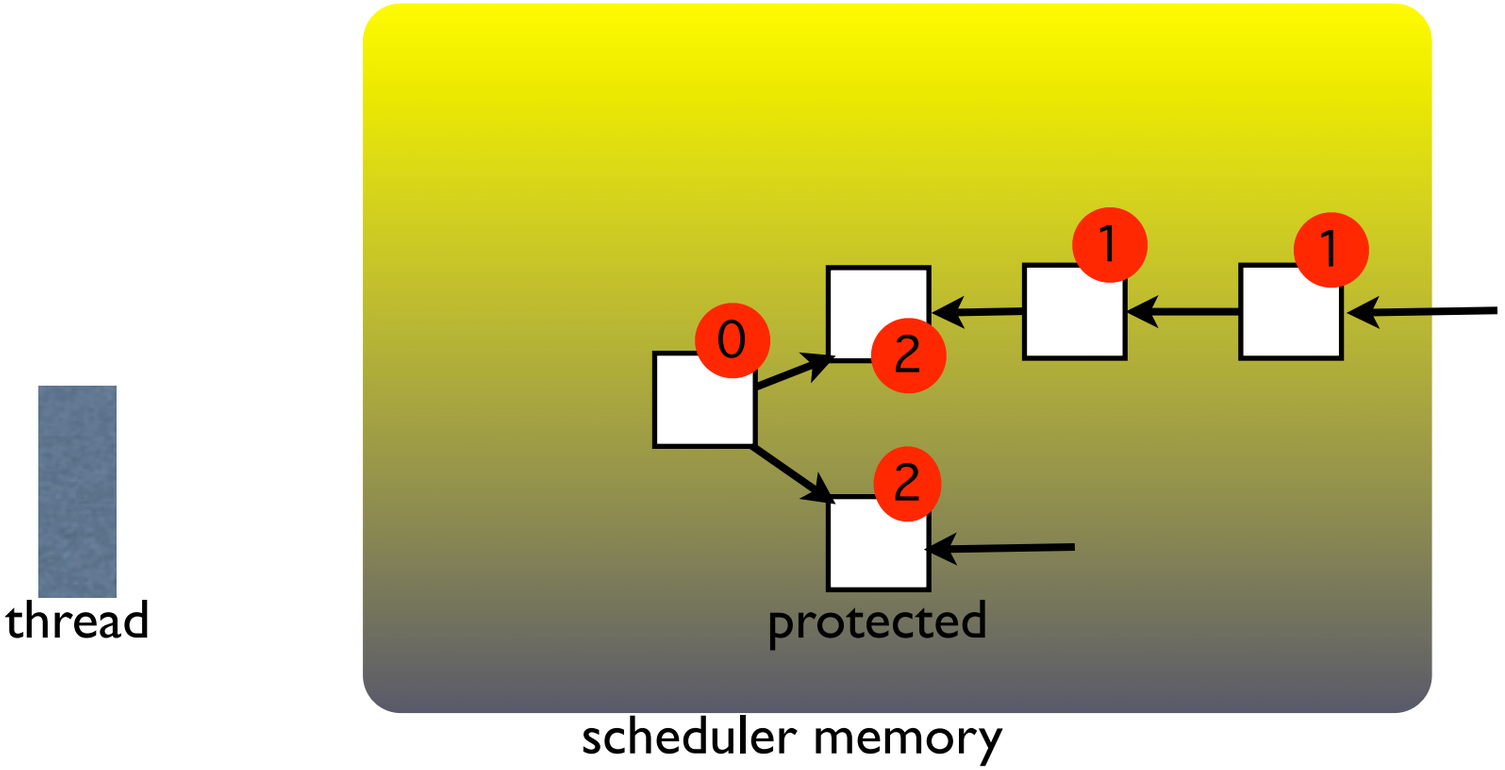


scheduler memory

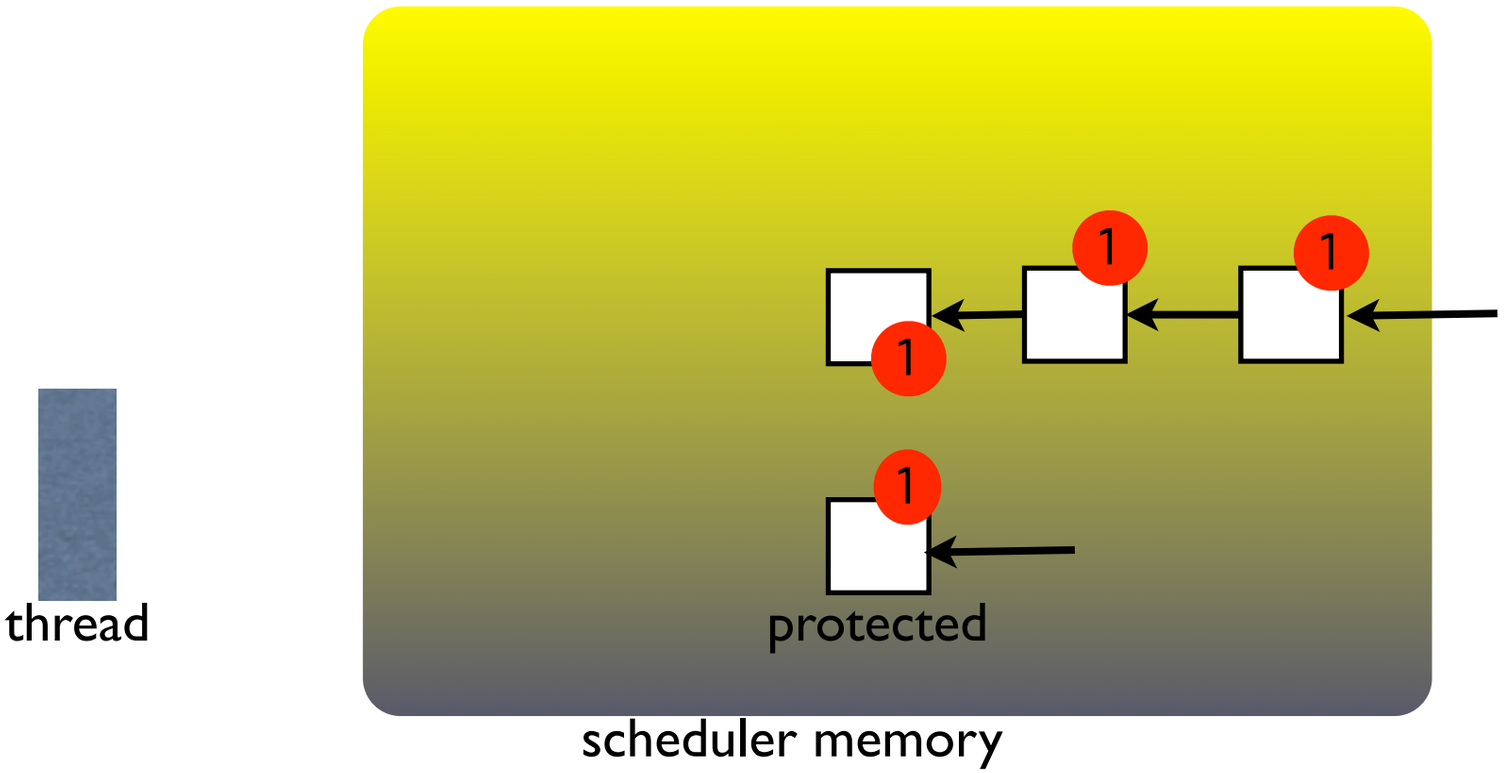
RGC Example



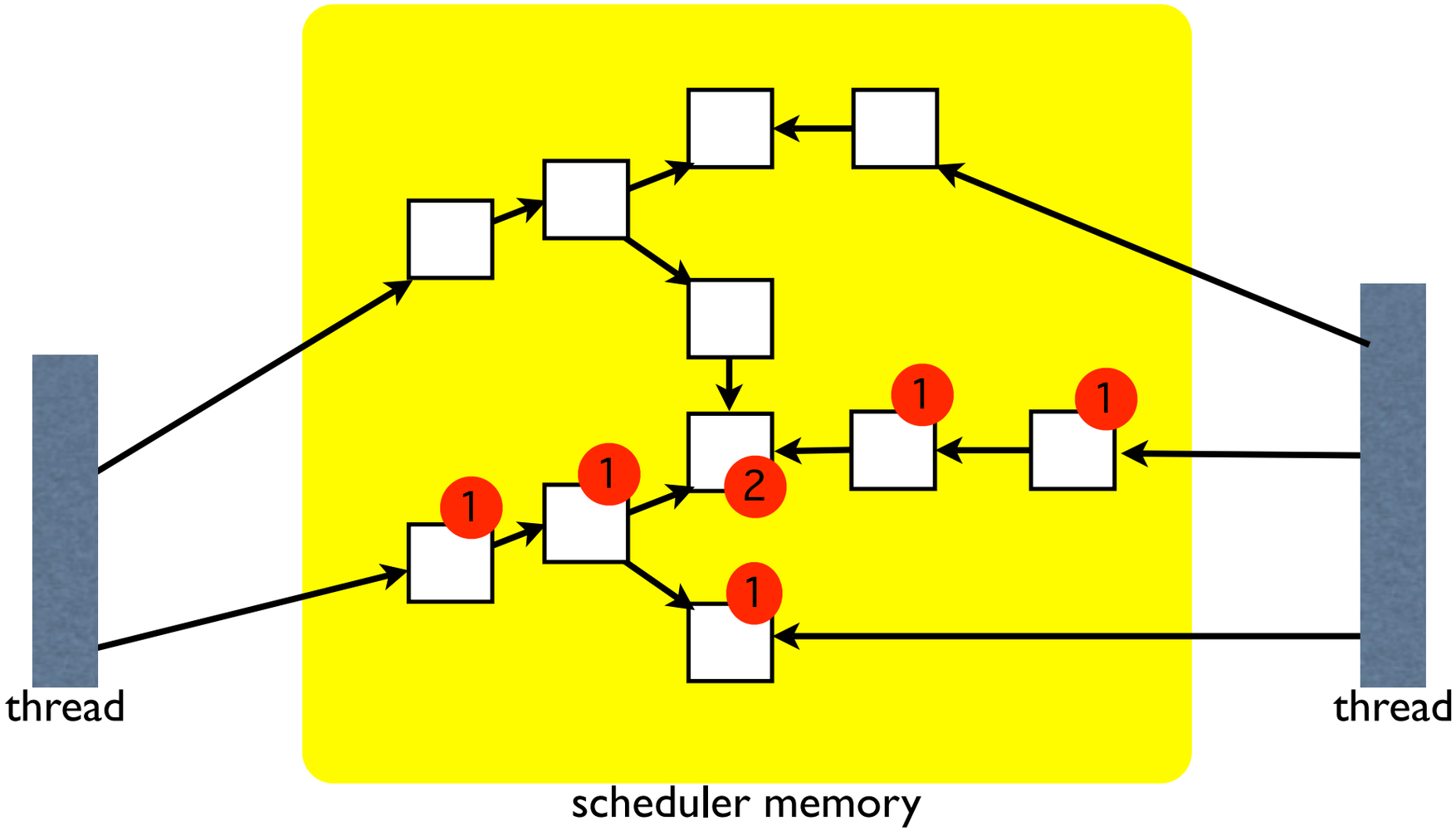
RGC Example



RGC Example

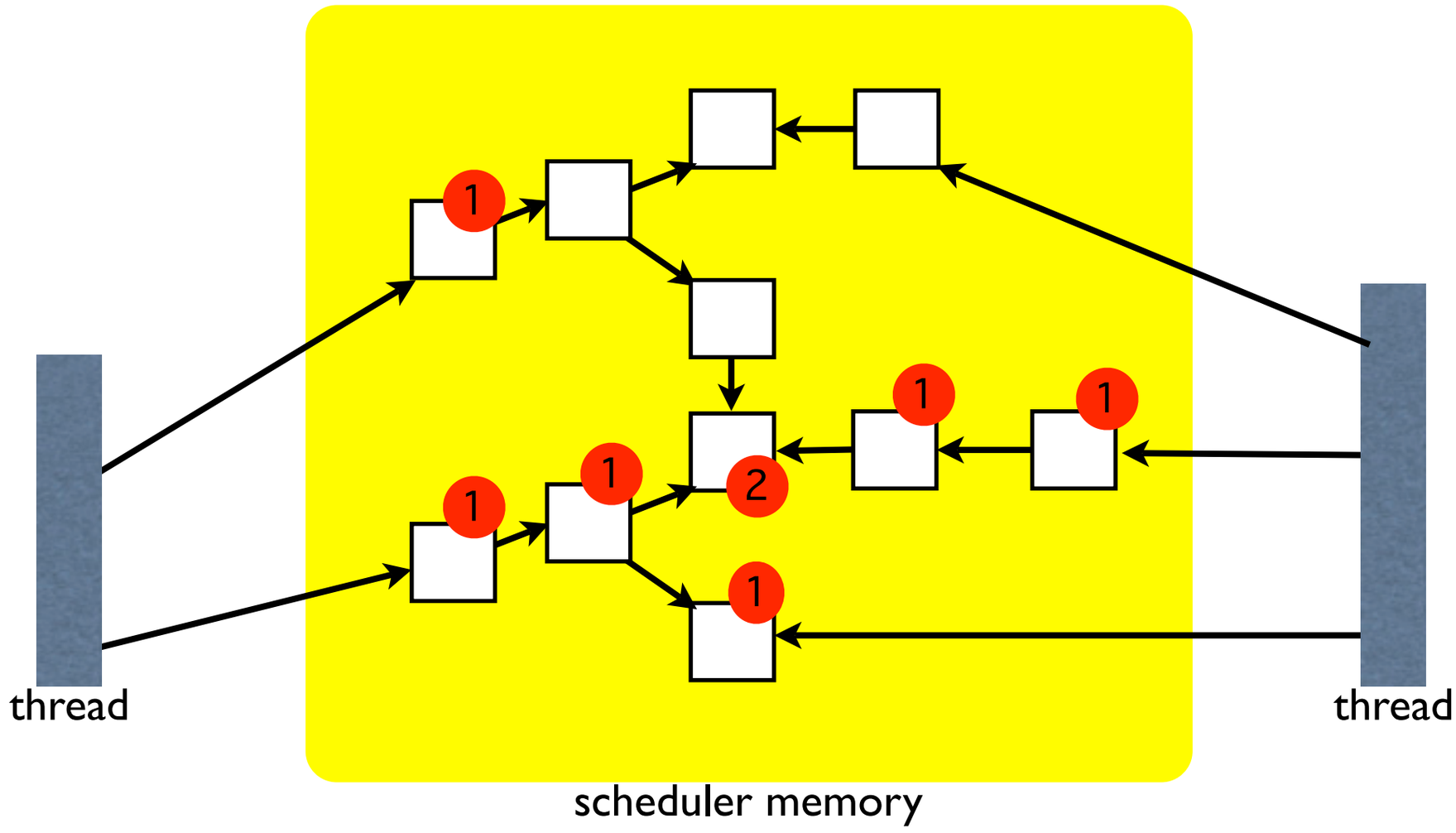


RGC Example

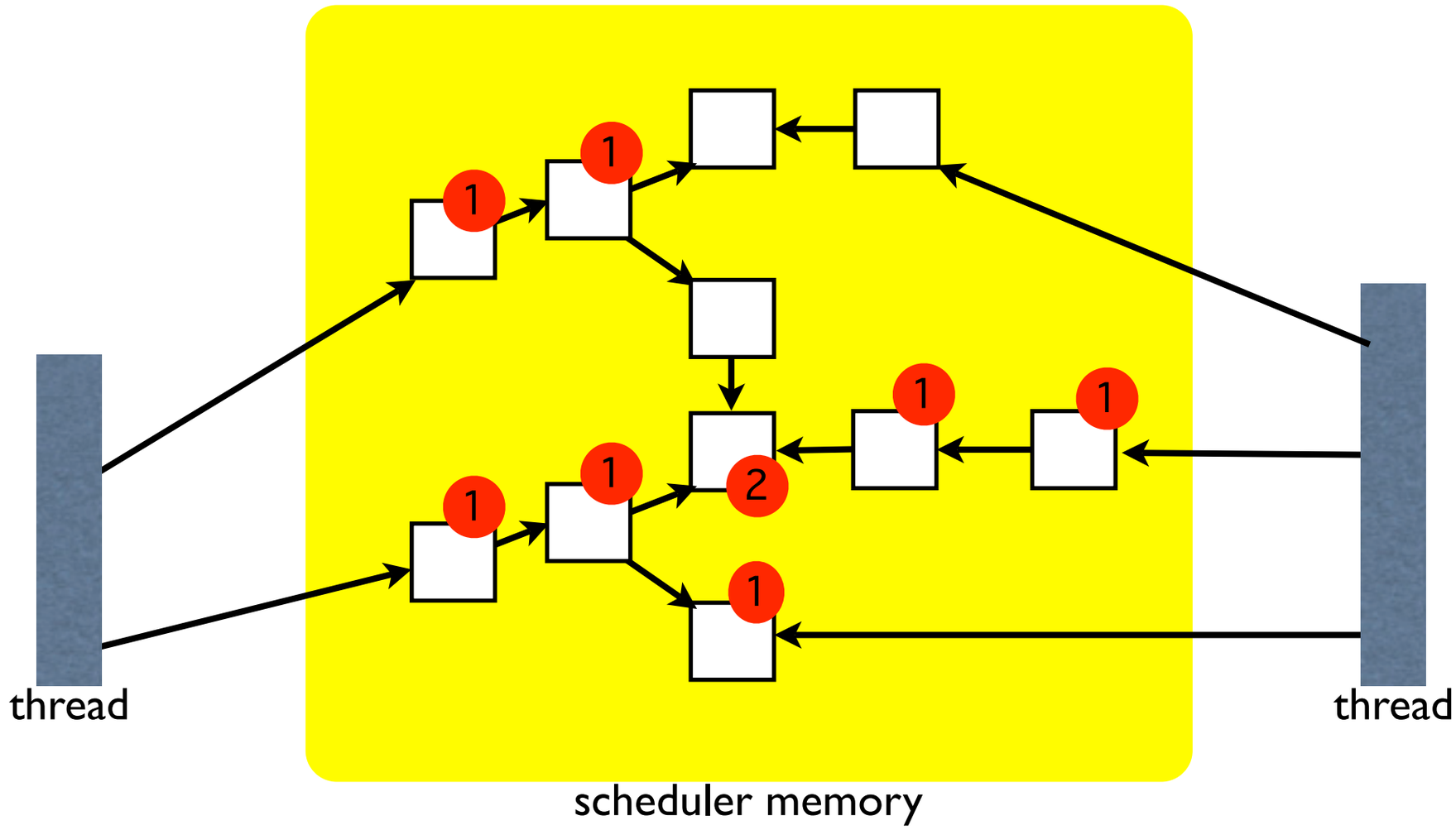


Transitive protection (may be expensive)

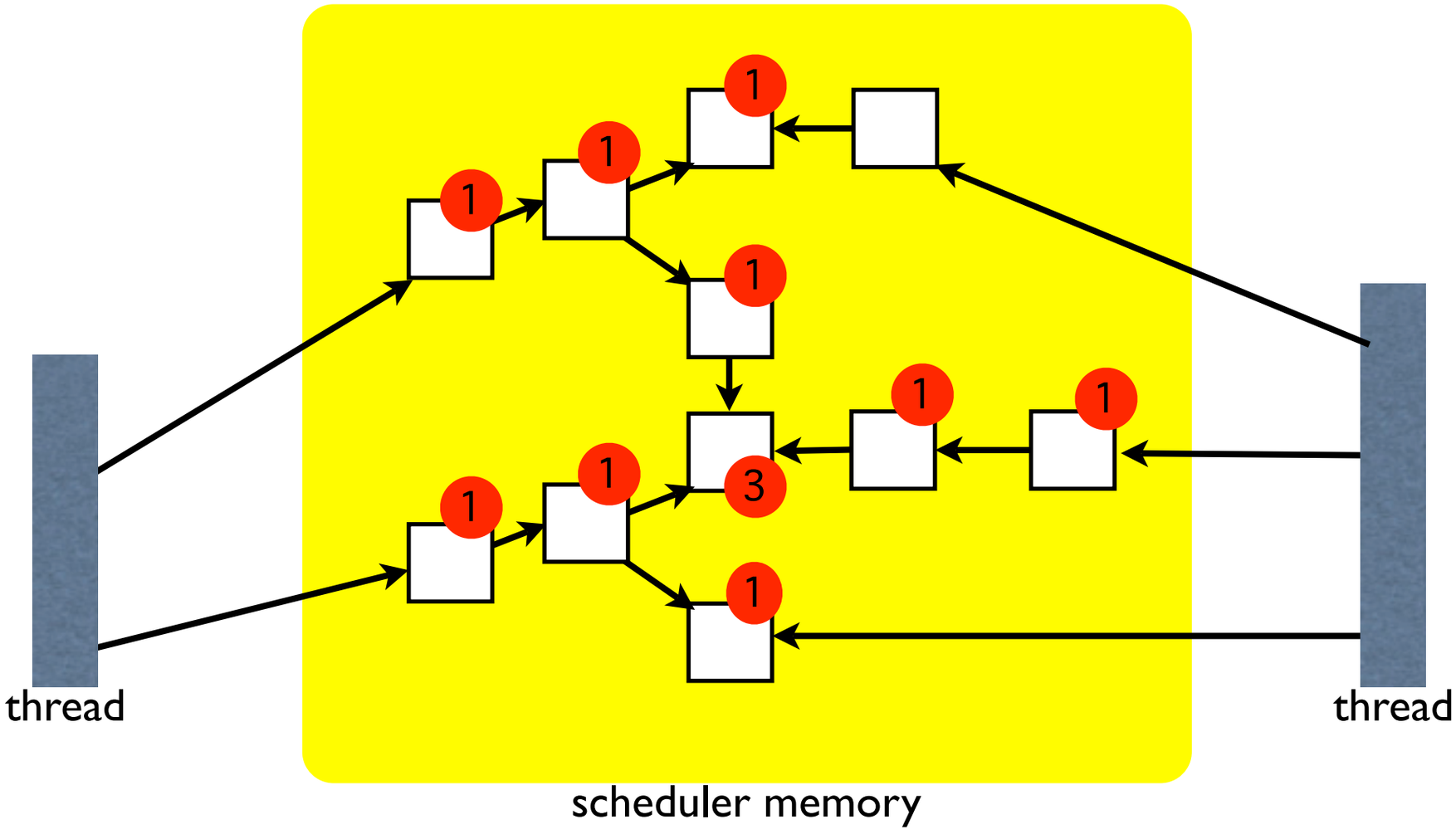
RGC Example



RGC Example

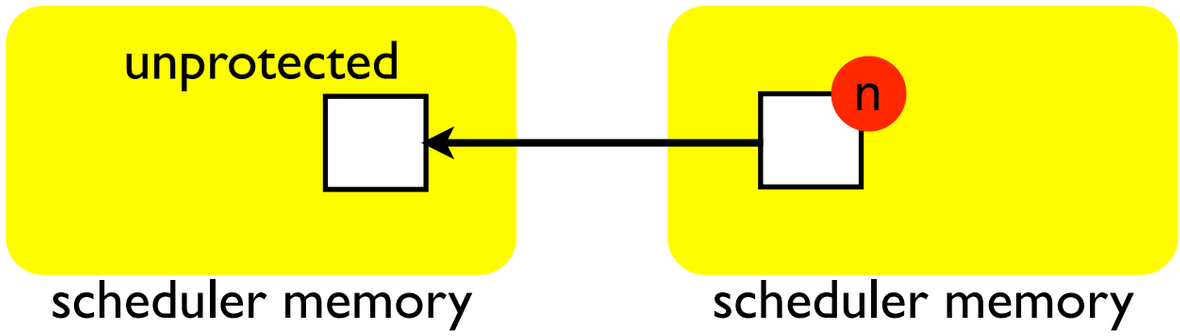


RGC Example

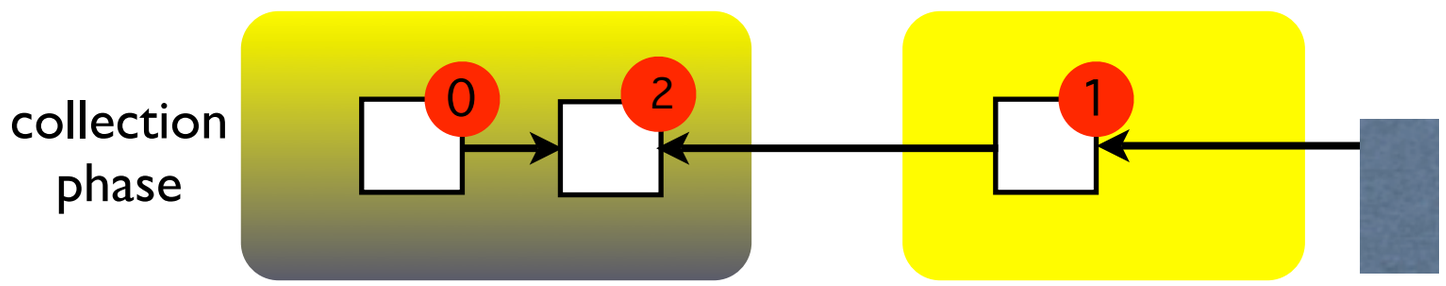


RGC Example

- Migration implies protection. Never occur:

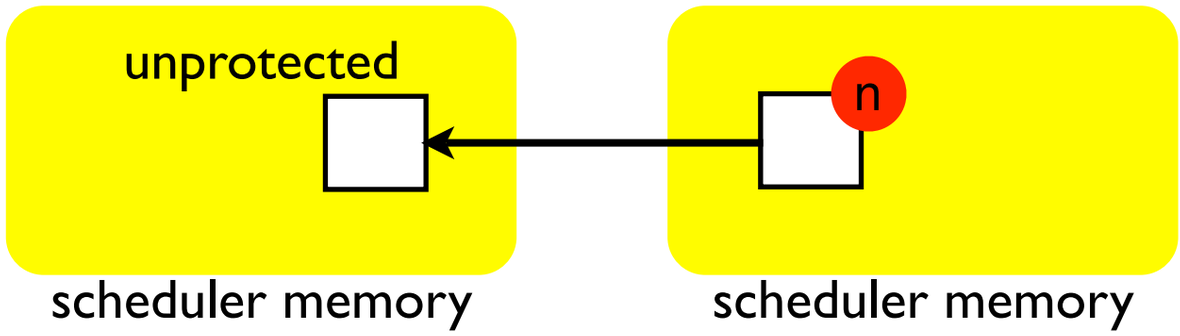


- Parallel collections are possible
- Locks must be associated to ref-counters:

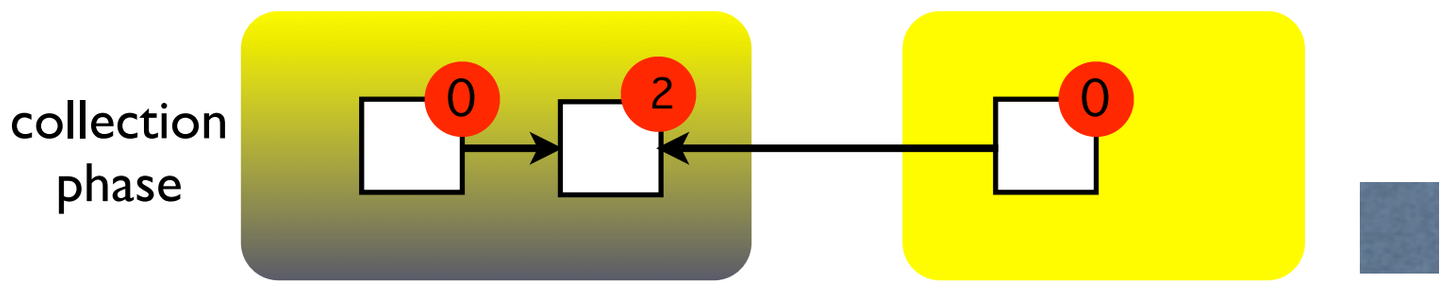


RGC Example

- Migration implies protection. Never occur:



- Parallel collections are possible
- Lock must be associated to ref-counters:



concurrent access to ref counter 2

RGC Algorithm

- Generational aspect: short-living = unprotected = one instant, long-living = protected = several instants
- Concurrent: collection by one scheduler, allocation by another
- Parallel: collection in real parallelism by 2 schedulers run on 2 different cores (protection needs locks, in the general case)
- On-the-fly: does not require full halt of the system ("stop-the-scheduler" instead of "stop-the-world")

Conclusion

- RGC used in the FunLoft compiler
- Absence of freezing effects on the simulations used as benchmarks (collisions of particles, cellular automata, preys-predators)
- Worst case: ~time*3 compared to Boehm's GC
- Actually, RGC is encapsulated in Boehm's GC (for protected cyclic data) in the FunLoft compiler