departamento de informática
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

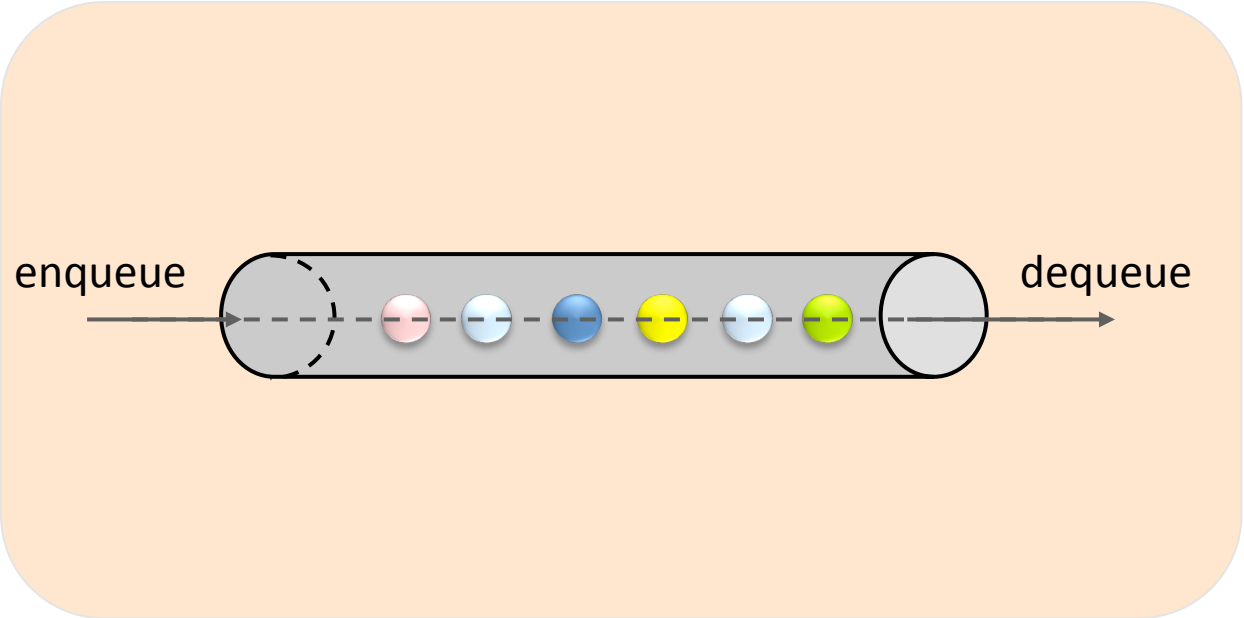
Concurrency and Parallelism
(Concorrência e Paralelismo – CP 11158)

Lecture 4

— Lock-free Data Structures—

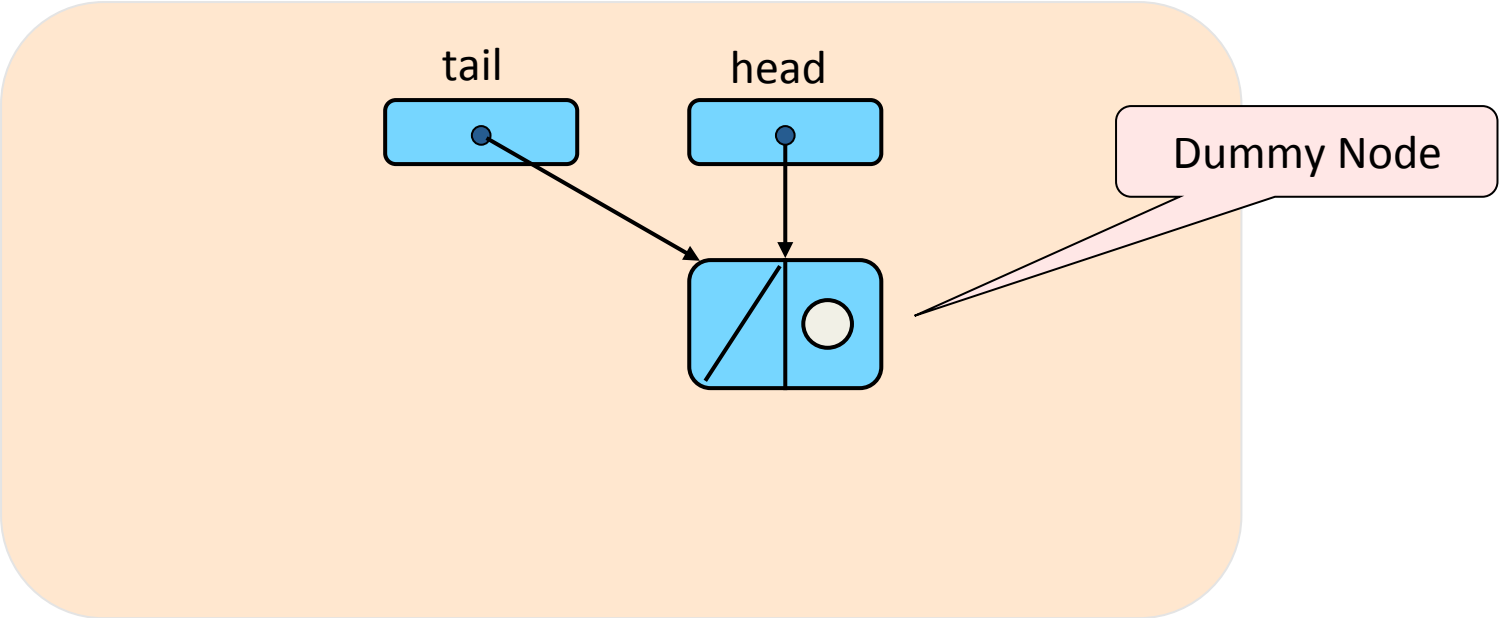
Slides based in material from:
Gadi Taubenfeld (http://www.faculty.idc.ac.il/gadi/book.htm)

Implementing a lock-free Queue

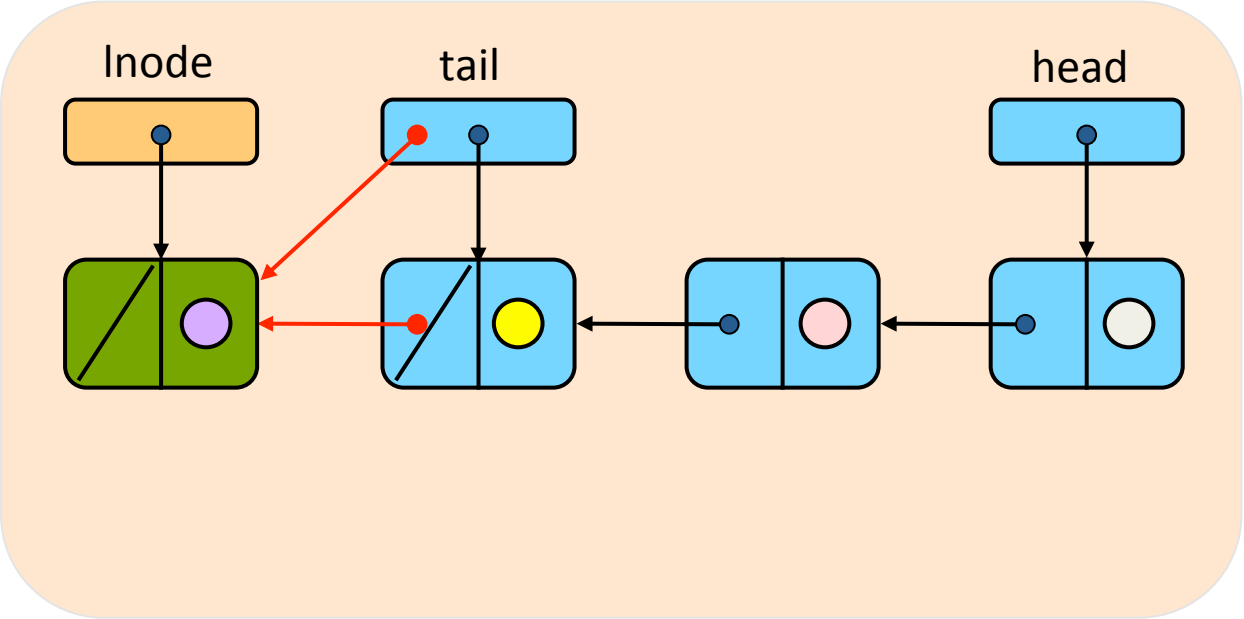


Implementing a lock-free Queue

Empty queue

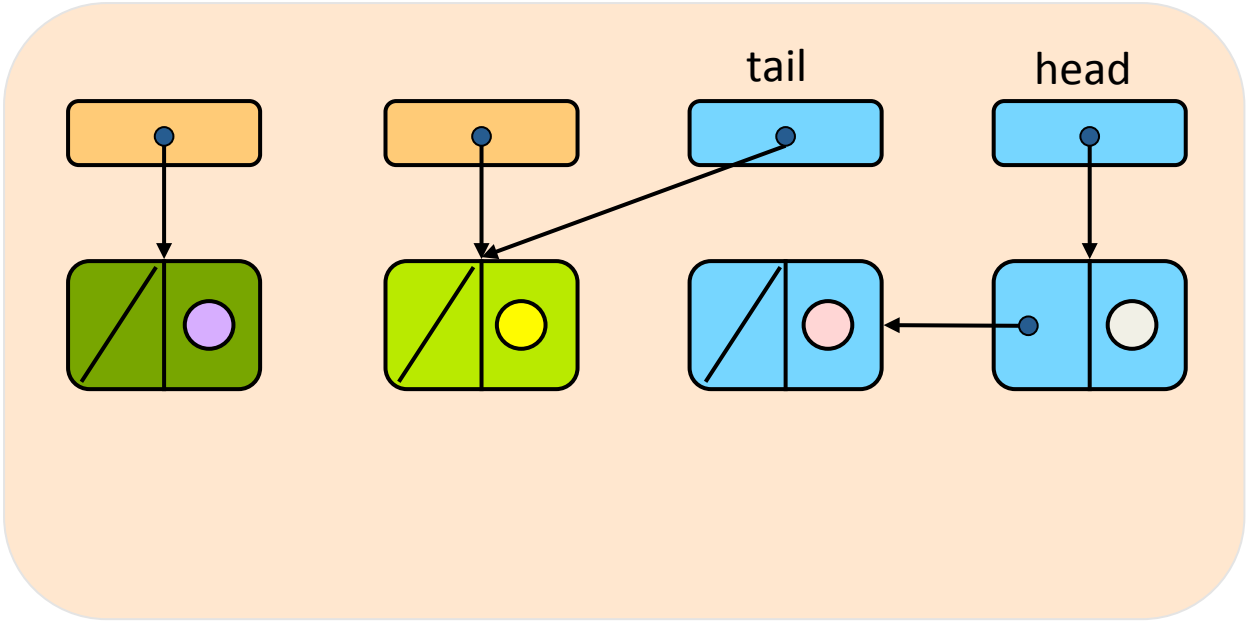


Enqueue

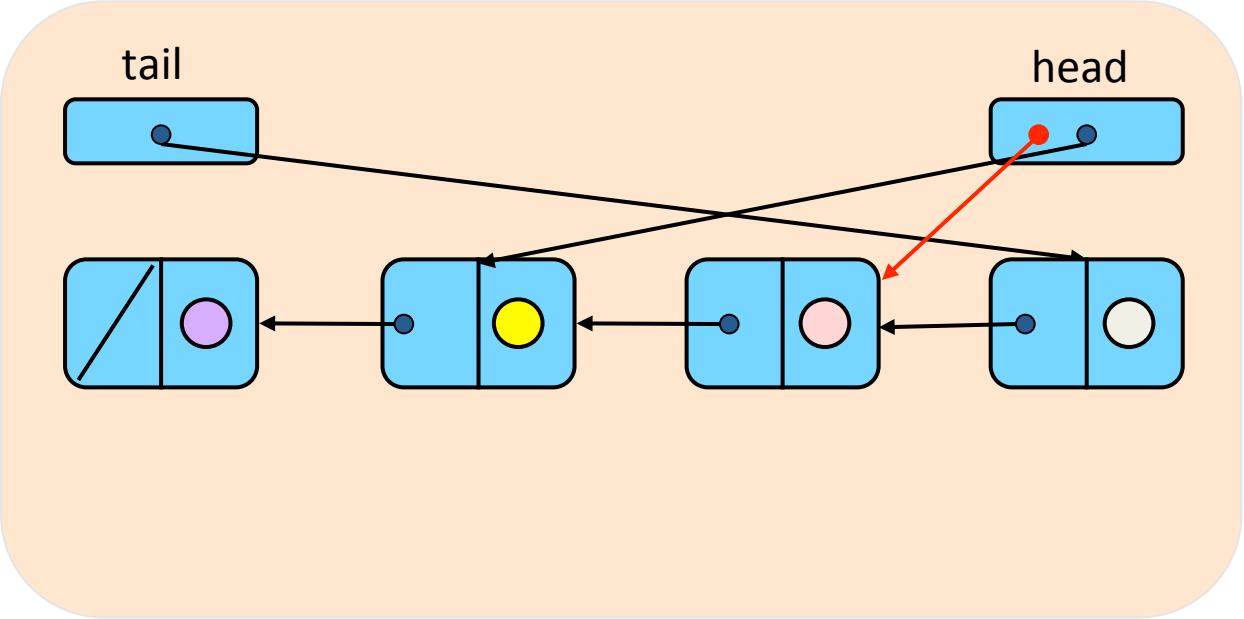


2 concurrent enqueue operations

What can go wrong when using registers?

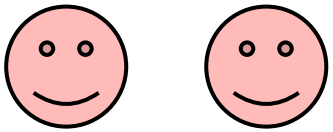


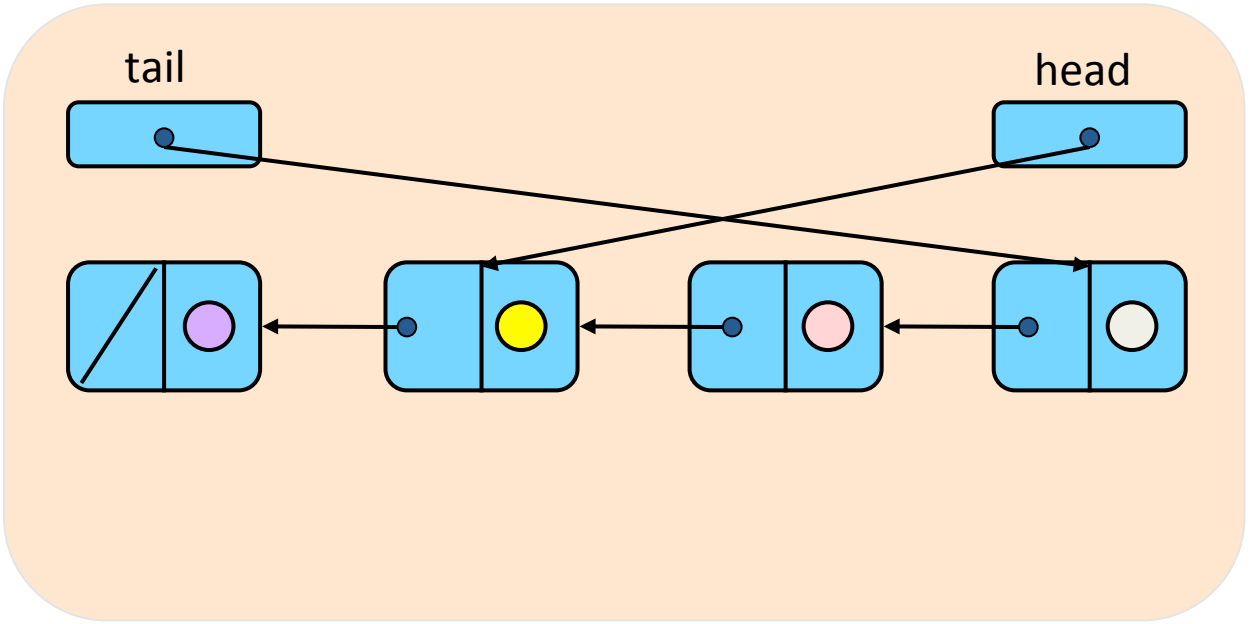
Dequeue



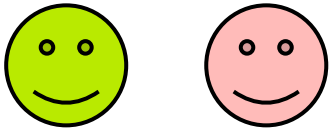
2 concurrent dequeue operations

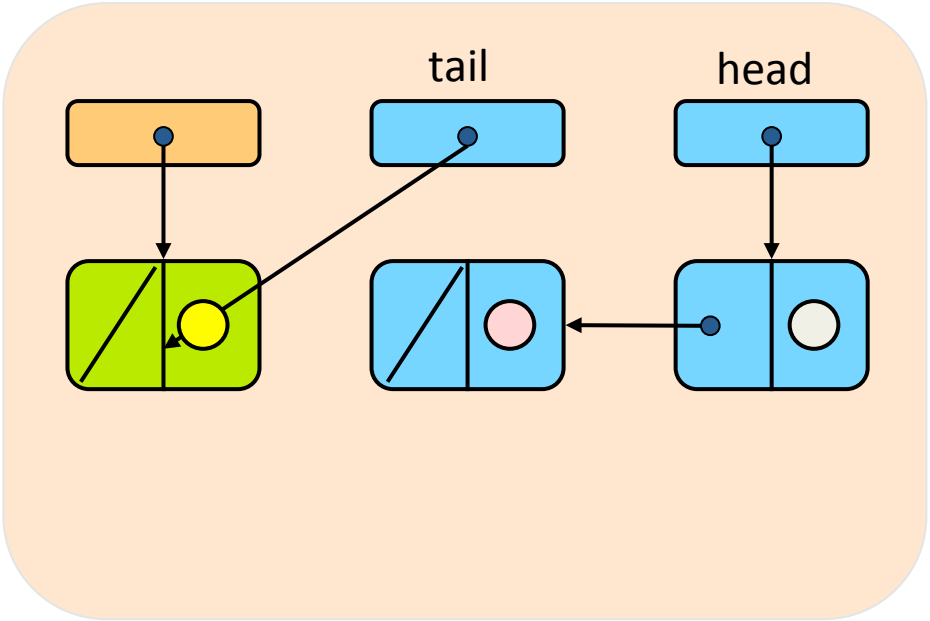
What can go wrong when using registers?



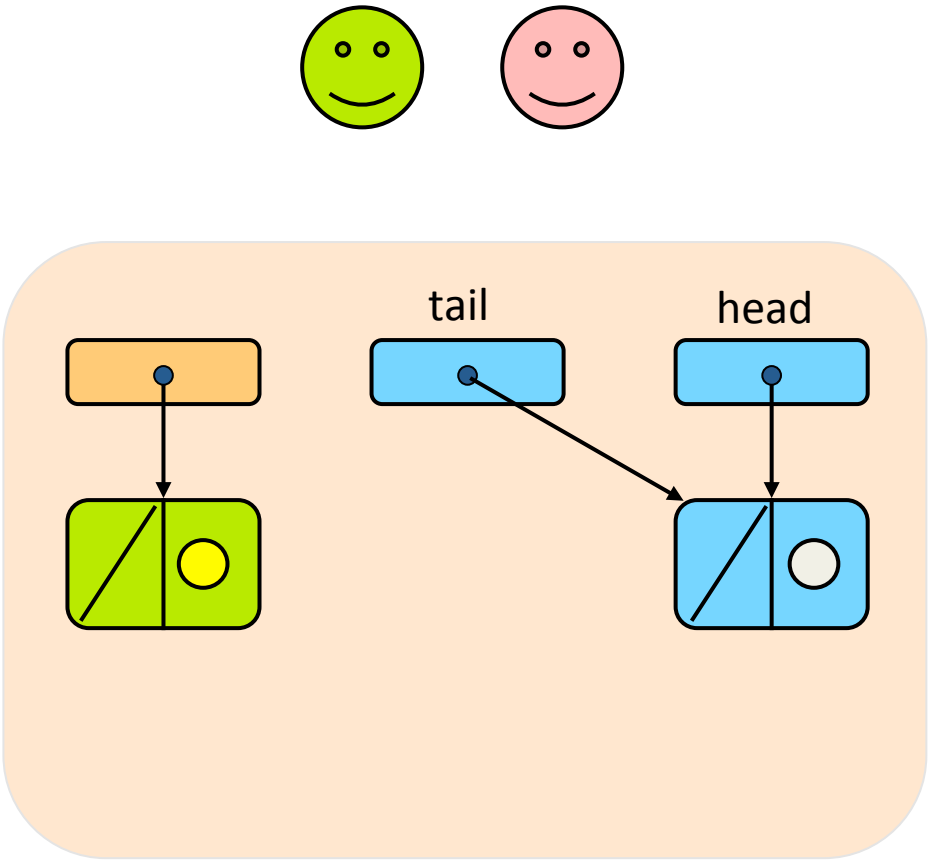


Enqueue + Dequeue (Non-empty queue)

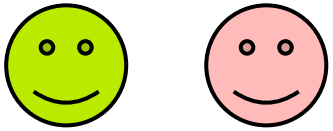


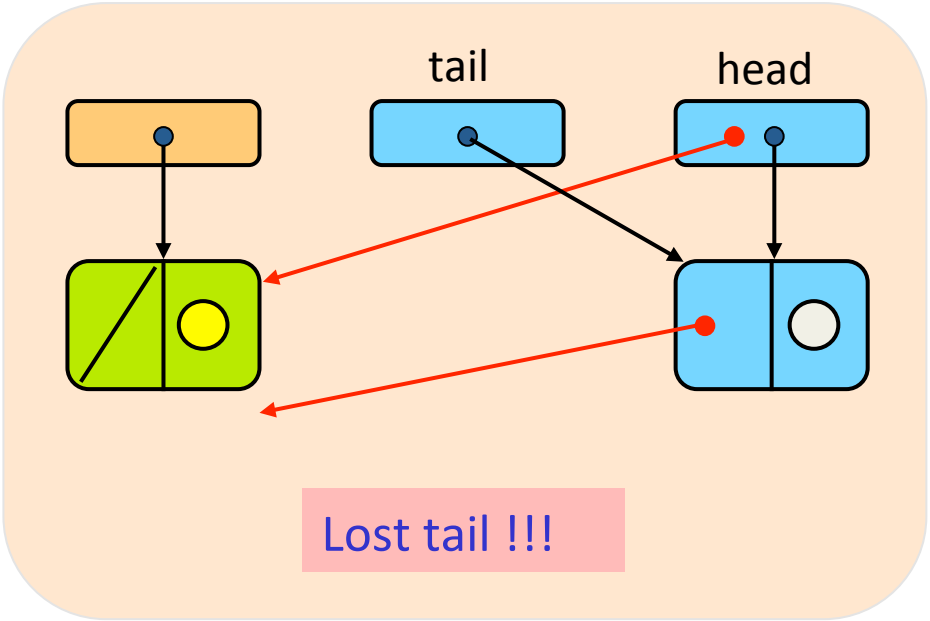


Enqueue + Dequeue (Empty queue)



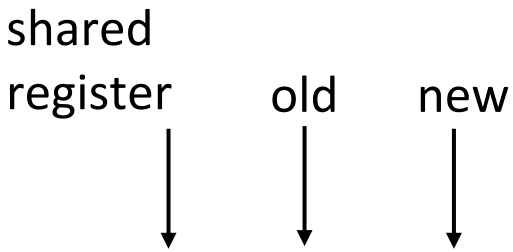
Enqueue + Dequeue (Empty queue)





The dequeuer must advance tail before redirecting head

Compare & swap (CAS)

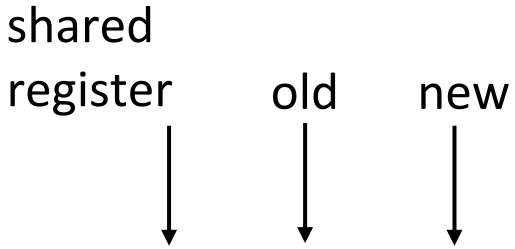


CAS (A, B, C)

if A=B then A:=C; return(true)
else return(false)

Supported by Sun, Intel, AMD, …

CAS: The ABA problem

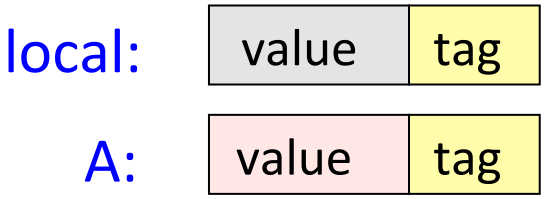


CAS (A, B, C)

if A=B then A:=C

local:= A

CAS (A, local, 100)



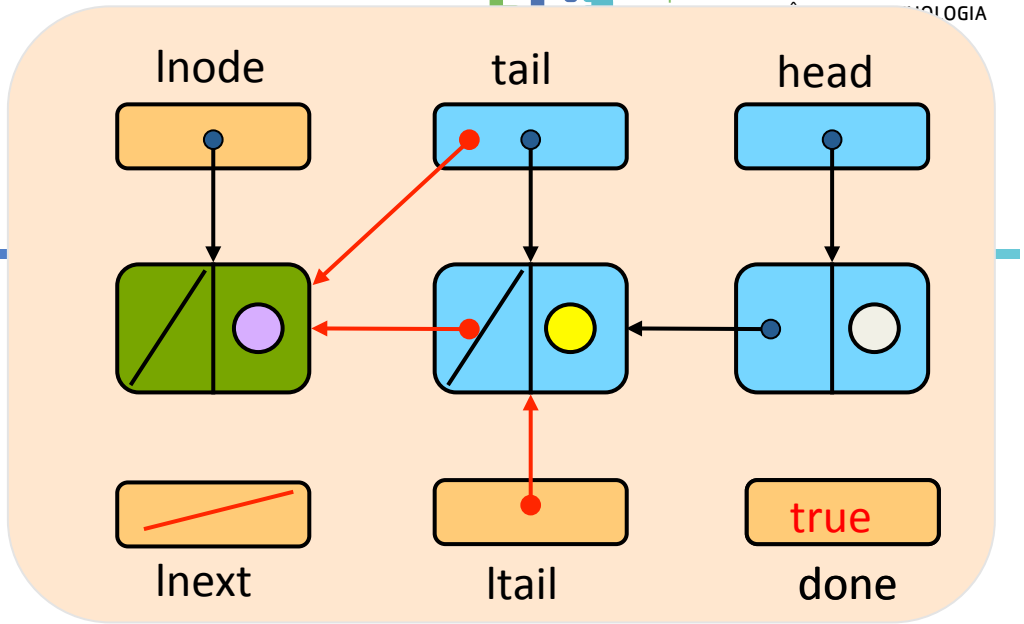
CAS (A, local, <100,local.tag+1>)

In the following, we will ignore the ABA problem and assume that it is resolved using additional tag fields.

A Lock-Free (non-blocking) Queue

- The algorithm is due to M. M. Michael and M. L. Scott (1996).
- The algorithm is included in the Standard Java Concurrency Package.
- Every process must be prepared to encounter a half-finished enqueue operation, and help to finish it.
- Invariant: The tail refers to either the last node or to the node just before the last node.

Enqueue



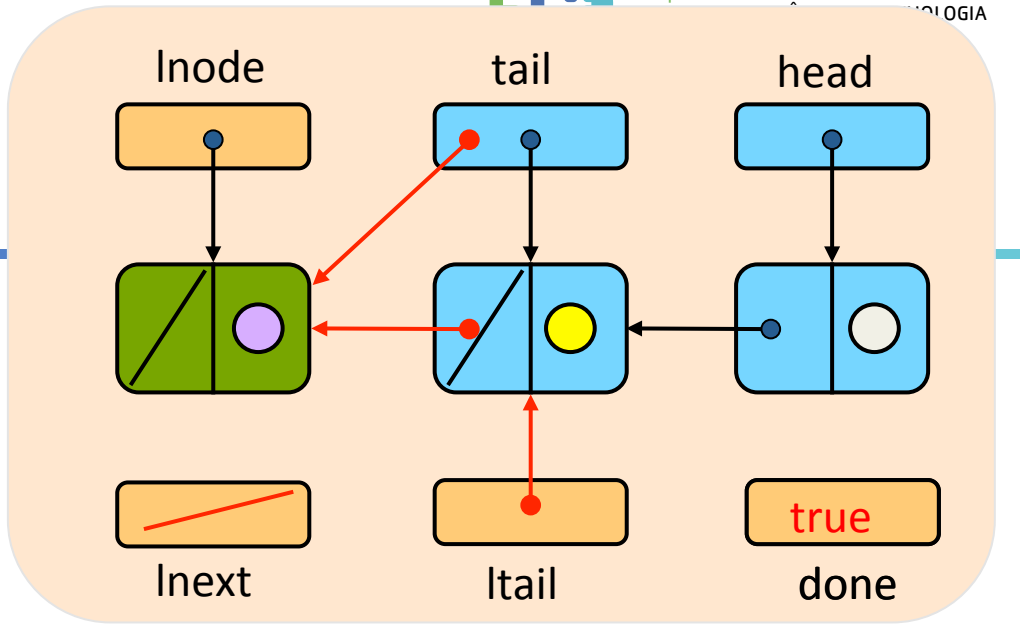

```
repeat
ltail ← tail
lnext ← ltail.next
if ltail = tail then
    if lnext = NULL then
        if CAS (ltail.next, lnext, lnode) then done ← true fi
    else CAS (tail, ltail, lnext) fi fi
until done = true
CAS (tail, ltail, lnode)
```

try to link lnode to the

enqueue failed: try to swing tail to a

enqueue done: try to swing tail to the inserted node.

Enqueue

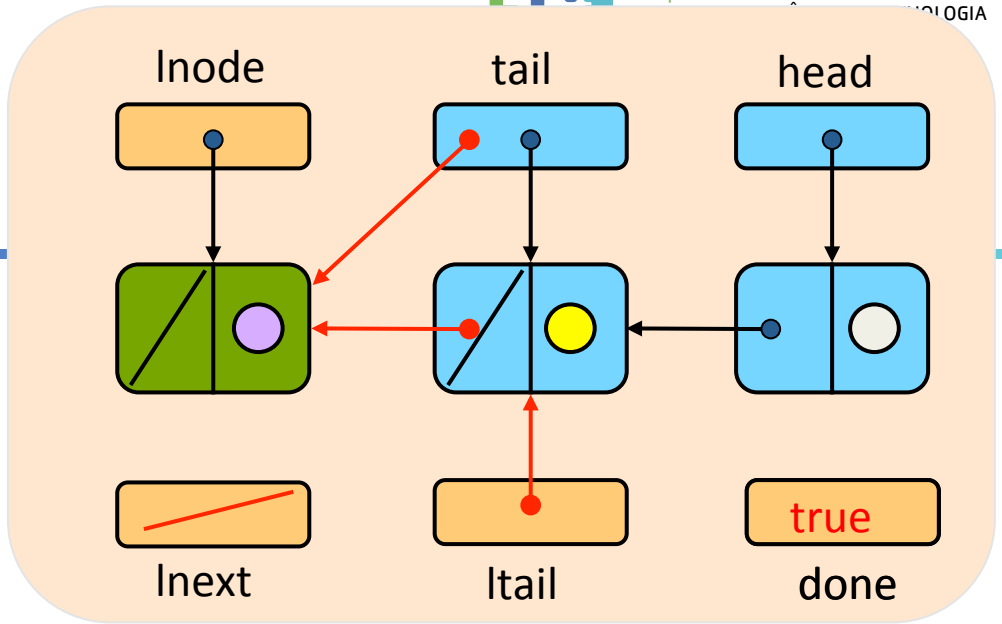



```
repeat
ltail ← tail
lnext ← ltail.next
if ltail = tail then
    if lnext = NULL then
        if CAS (ltail.next, lnext, node) then done ← true fi
    else CAS (tail, ltail, lnext) fi fi
until done = true
CAS (tail, ltail, lnode)
```

Question: Would the enqueue operation be correct if the last line is omitted?

Answer: Yes, but…

Enqueue

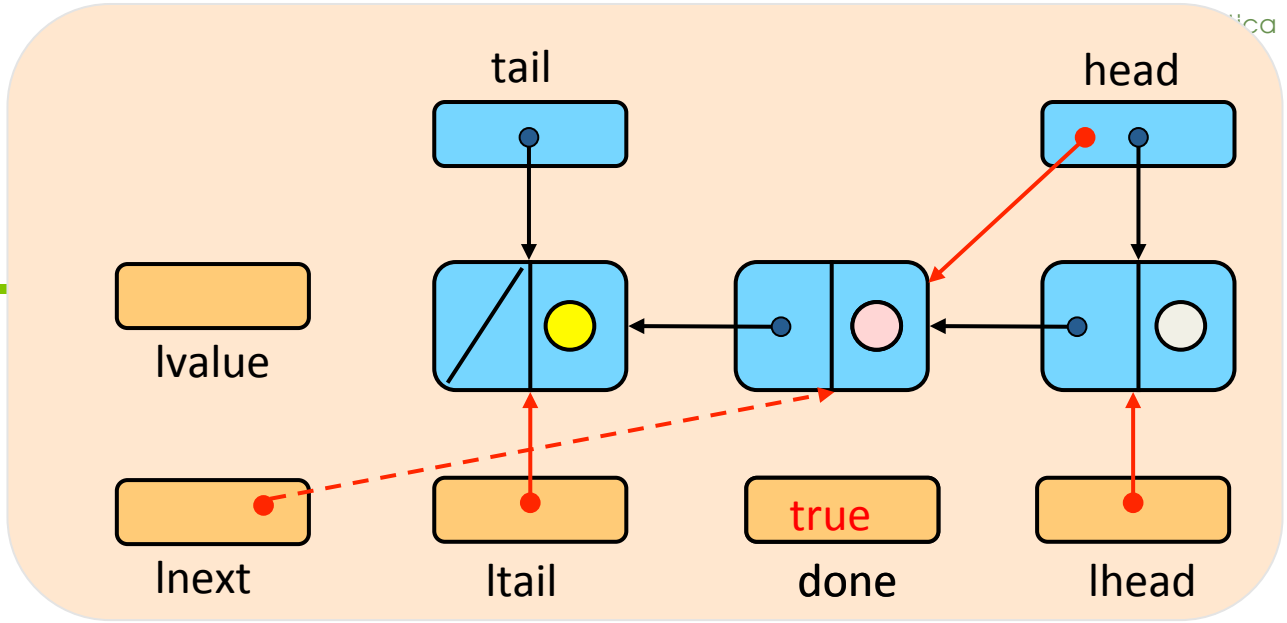



```
repeat
ltail ← tail
lnext ← ltail.next
if ltail = tail then
    if lnext = NULL then
        if CAS (ltail.next, lnext, node) then done ← true fi
    else CAS (tail, ltail, lnext) fi fi
until done = true
CAS (tail, ltail, lnode)
```

Question: Would the enqueue operation be correct if this line is omitted?

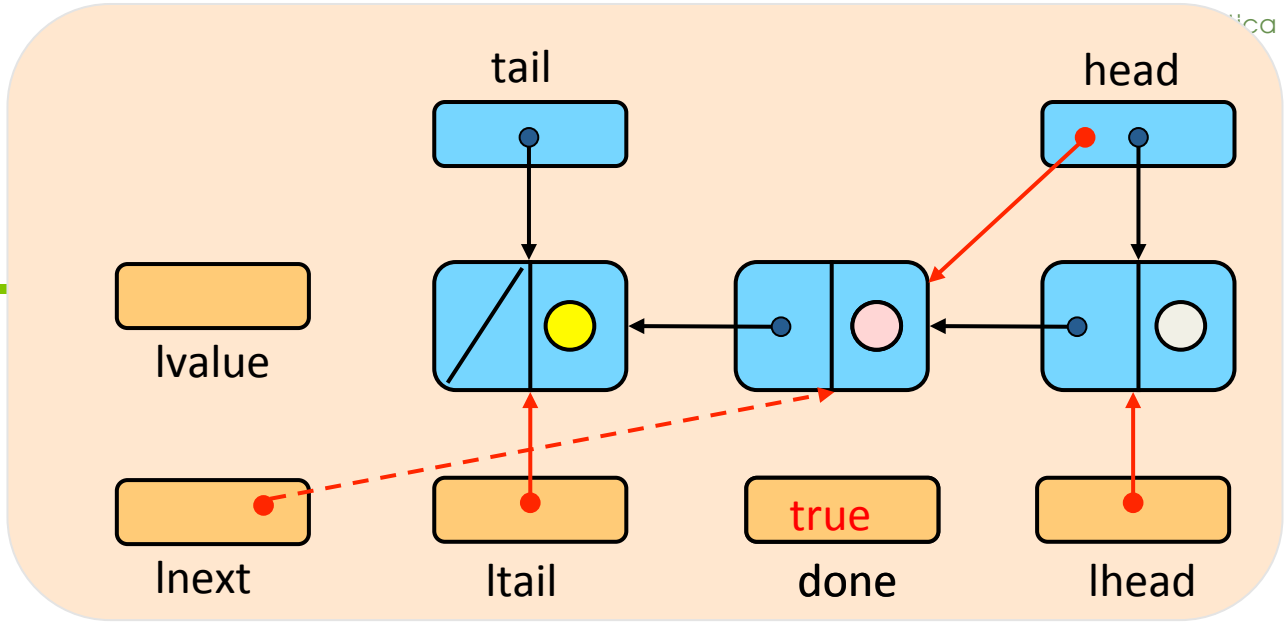
Yes.

Dequeue




```
repeat
lhead ← head
ltail ← tail
lnext ← lhead.next
if lhead = head then
    if lhead = ltail then                    /* empty or tail behind?
        if lnext = NULL then return(NULL) fi ;        /* empty
        CAS (tail, ltail, lnext)                 /* try to advance tail
    else lvalue ← lnext.value                /* no need to deal with tail
        if CAS (head, lhead, lnext) then done ← true fi fi fi
until done = true
free(lhead) ; return(lvalue)
```

Dequeue




```
repeat
lhead ← head
ltail ← tail
lnext ← lhead.next
if lhead = head then
    if lhead = ltail then                    /* empty or tail behind?
        if lnext = NULL then return(NULL) fi ;        /* empty
        CAS (tail, ltail, lnext)                 /* try to advance tail
    else lvalue ← lnext.value                /* no need to deal with tail
        if CAS (head, lhead, lnext) then done ← true fi fi fi
until done = true
free(lhead) ; return(lvalue)
```

Question: Would the dequeue operation be correct if this line is omitted?

departamento de informática
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

The End