



Introduction to Non-Blocking Algorithms

Alexandre David

1.2.05

adavid@cs.aau.dk



Concurrent Non-Blocking Algorithms

- Concurrent: Several threads can execute the algorithms simultaneously.
- Blocking algorithms: Algorithms for which processes may isolate or block part of the data-structure to access it without interference. **May cause deadlocks.**
- Non-blocking algorithms: They ensure that the data-structure is always accessible to all processes. **Independent from other halted/delayed processes.**

Compare and swap (CAS)

- **Atomic** instruction available on most processors.
- Most common building block for non-blocking algorithms.
- Available in Java
`AtomicInteger.compareAndSet(int,int) -> bool`
- If the memory is equal to some expected value (compare) then set the memory to a new value.
- Intel:
`cmpxchg r/m, r` (needs lock prefix)

if `eax == r` then `r/m = r`, `ZF=0`
else `eax = r/m`, `ZF=1`



Other Atomic Instructions (Intel)

- Increment.
(lock inc r/m)
- Decrement.
(lock dec r/m)
- Exchange.
(xchg r/m, r)
- Fetch and add.
(lock xadd r/m, r)
- They can be used to implement simple and efficient synchronizations primitives.



Non-Blocking Algorithms

- The key:
 - Try to compute speculatively.
 - CAS before committing the result.
 - Retry if CAS fails.
- Good practice:
 - Work with a state-machine.
 - Every state must be consistent.
 - States = committed (intermediate) results.



Non-Blocking Counter

Standard blocking algorithm

```
proc inc(A)
lock
  tmp = A
  tmp = tmp+1
  A = tmp
unlock
end
```

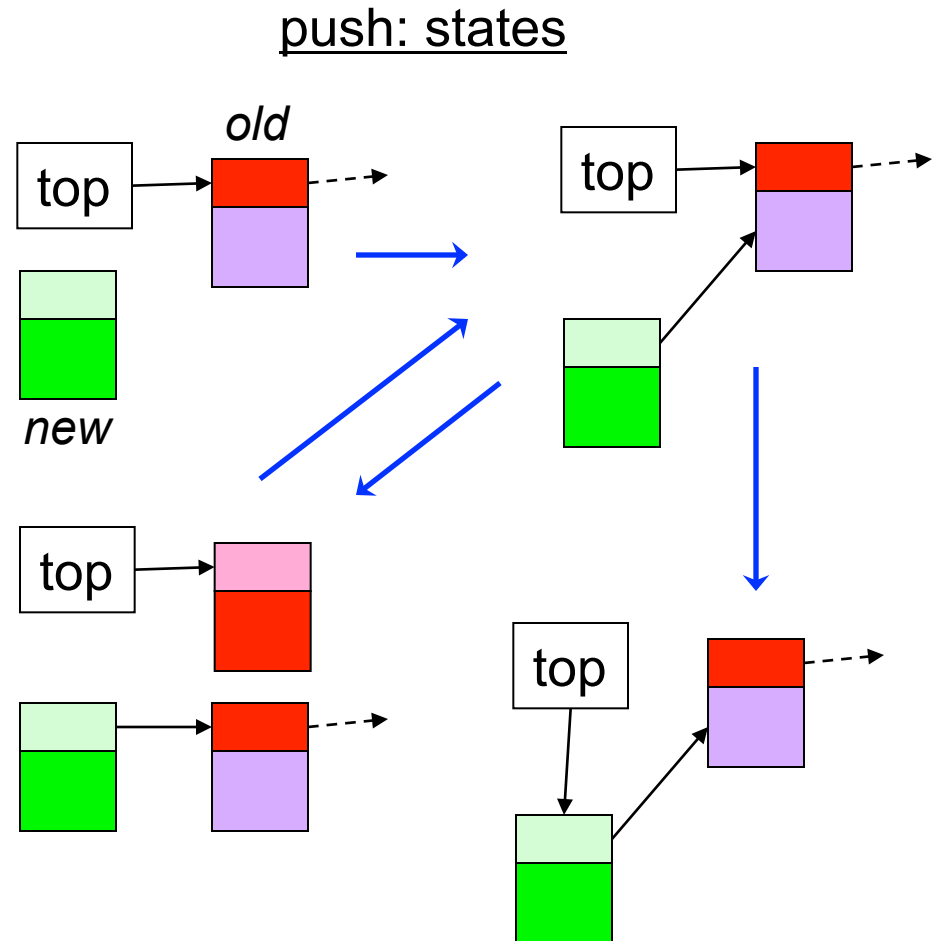
Non-blocking algorithm

```
proc inc(A)
do
  tmp = A
  while not CAS(A, tmp, tmp+1)
end
```

Non-Blocking Stack [Treiber's Algorithm]

```
proc push(new)
do
  old = top
  new.next = old
while not CAS(top, old, new)
end

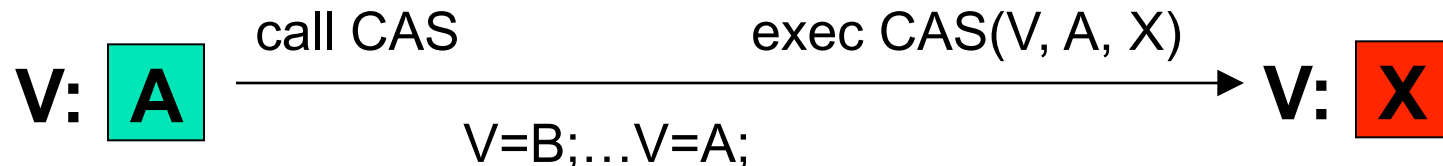
proc pop
do
  old = top
  return null if old == null
  new = old.next
while not CAS(top, old, new)
return old
end
```



The diagram is divided into two horizontal sections. The top section, titled "pop: states", shows a sequence of states. On the left, a box labeled "top" points to a purple circle labeled "old". Below it, an arrow labeled "return" points to another purple circle labeled "0". A blue arrow points to the right. On the right, a box labeled "top" points to a stack of two boxes: a red one on top and a purple one on the bottom. The red box is labeled "old" with a purple "0" above it. A dashed arrow points from the red box to the right. Below the stack, an arrow labeled "return" points to a purple circle labeled "0". The bottom section shows a more complex state transition. On the left, a box labeled "top" points to a stack of a red box (labeled "old") and a purple box (labeled "new"). A dashed red arrow points from the red box to the right. A blue arrow points down to another box labeled "top", which has a dashed red arrow pointing to the right. Below this, an arrow labeled "return" points to a stack of a red box (labeled "old") and a purple box. In the center, two blue arrows point horizontally between the left and right stacks. On the right, a box labeled "top" points to a stack of a pink box (labeled "old") and a red box. Below it, a stack of a red box (labeled "old") and a purple box has a dashed red arrow pointing from the red box to the right. A red box in the bottom right corner contains the text "Careful with memory!".

The ABA Problem

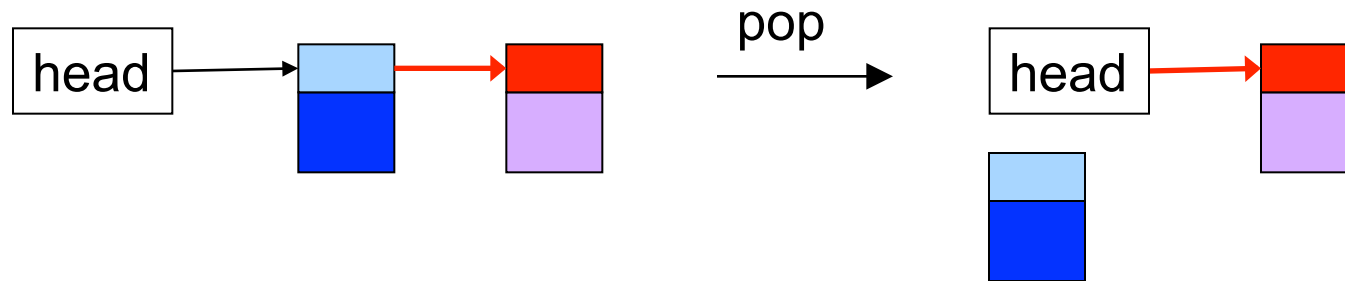
- Suppose that the value of V is A .
- Try a CAS to change A to X .
- Another thread can change A to B and back to A .
- The CAS won't see it and will succeed.
- Usual solution: Add a version number to V .



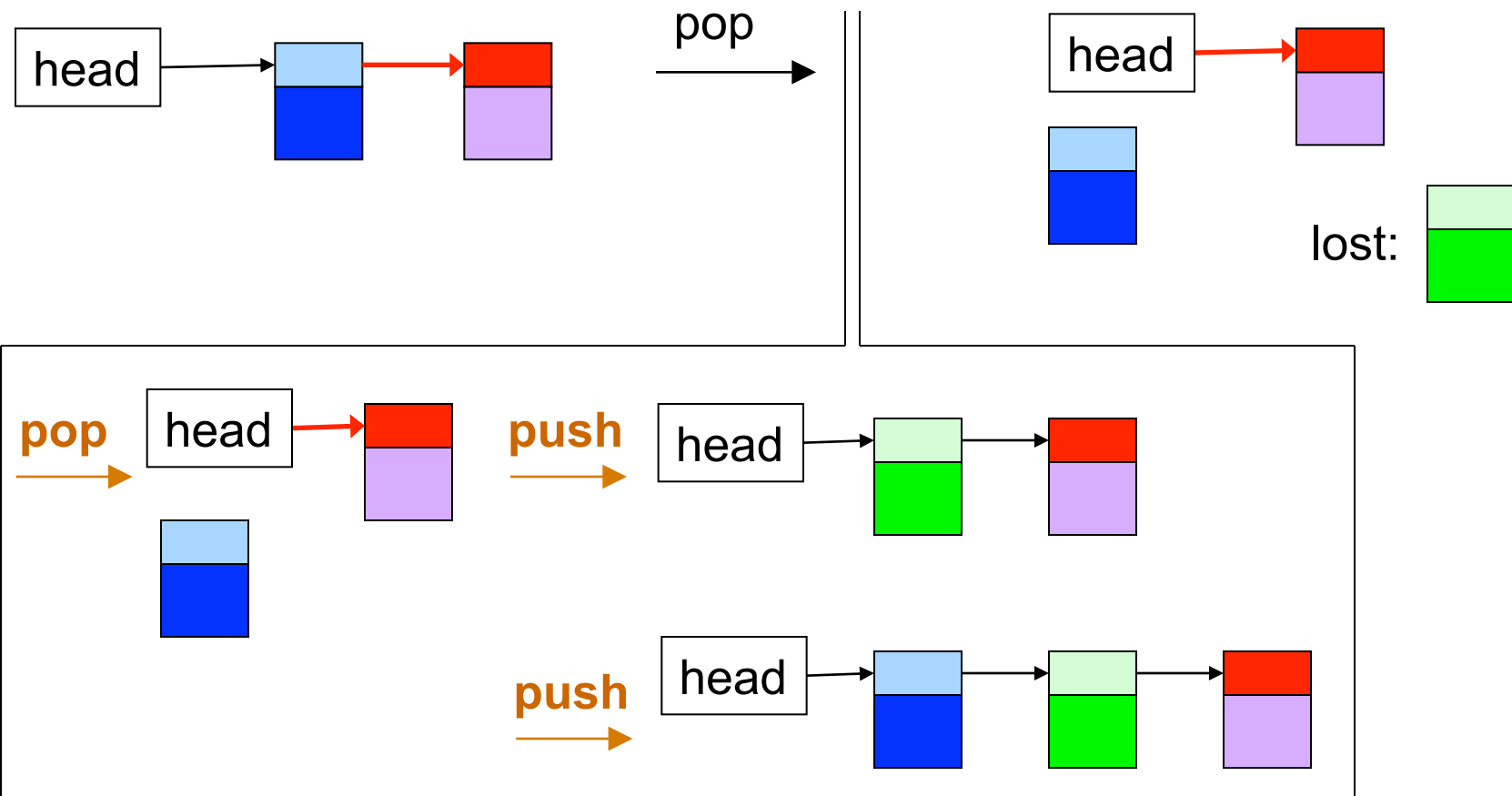
The ABA problem

- **Some** algorithms may suffer from it.
- Example: Linked list.

Expected behavior



The ABA problem





Fixes

- Reference counter (implicit in Java).
 - Allocation/de-allocation problems.
- Version number.
 - ABA problems.

Insertion in a Queue

[Michael-Scott's Algorithm]

```
proc put(new)
do
  last = tail
  nxt = last.next
  if last == tail
    if nxt == null
      if CAS(last.next, null, new)
        CAS(tail, last, new)
        break
    fi
  else
    CAS(tail, last, nxt)
  fi
fi
loop
end
```

ABA problem: use tags.

