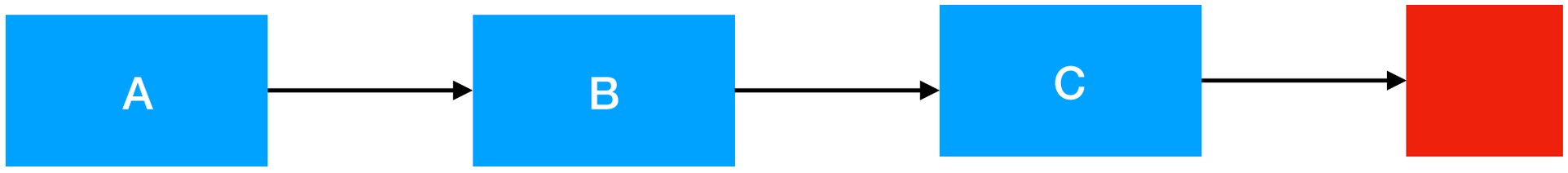
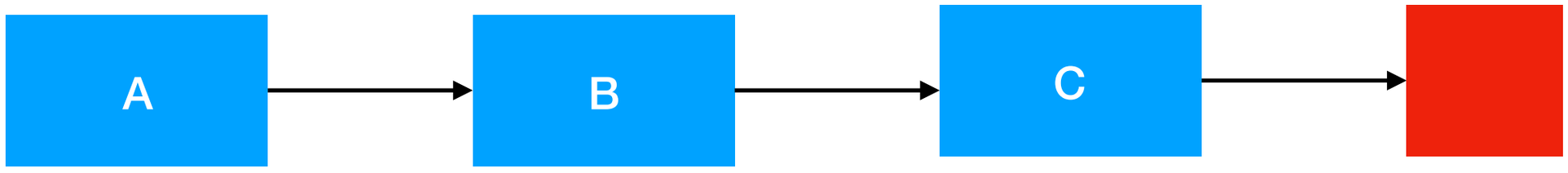
Lab 2: Non-blocking Stack



- Working with Pthreads on multicore CPU
- Using atomic operations (CAS)
- Implementing efficient parallel data structures

Unbounded Stacks



- Stacks implemented as **linked lists**
- Non-blocking: **NO LOCKS**!
- **Push** and **Pop** operations with atomic instructions

Compare-and-Swap

- Do atomically:

  - If *pointer* != *old pointer*: do nothing
    Else: swap *pointer* to *new pointer*

- Typically used only for compare + assign, no swap

```
CAS(void** pointer, void* old, void* new)
  {
     atomic {
        if(*pointer == old)
           *pointer = new;
     }
     return old;
  }
```

CAS for Stack

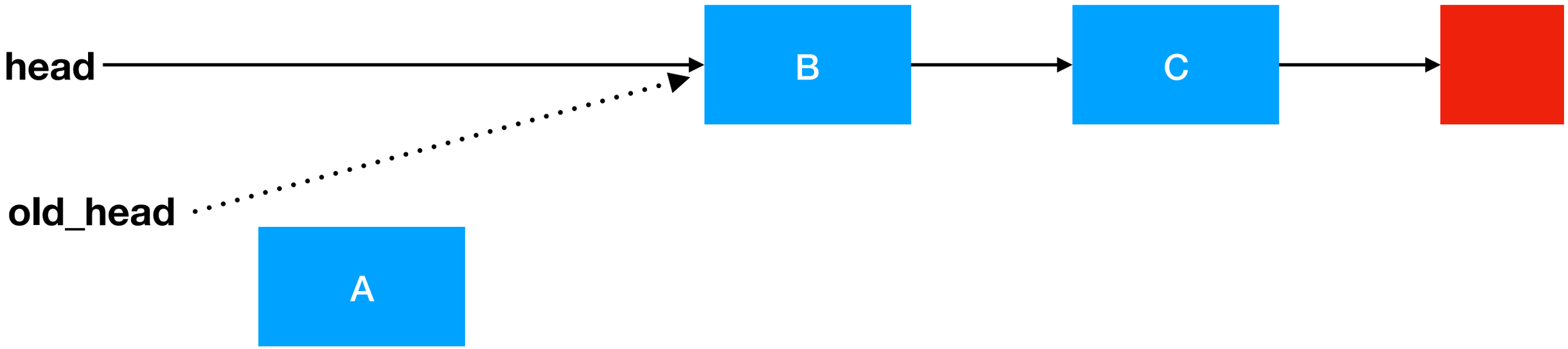
- Push
  - Keep track of old head
  - Set new elements next pointer to old head
  - **Atomically**:
    - Compare current head with saved old head
    - If still equal, set list head to new element

```
do {
    old = head; elem.next = old;
} while(CAS(head, old, elem) != old);
```

CAS push

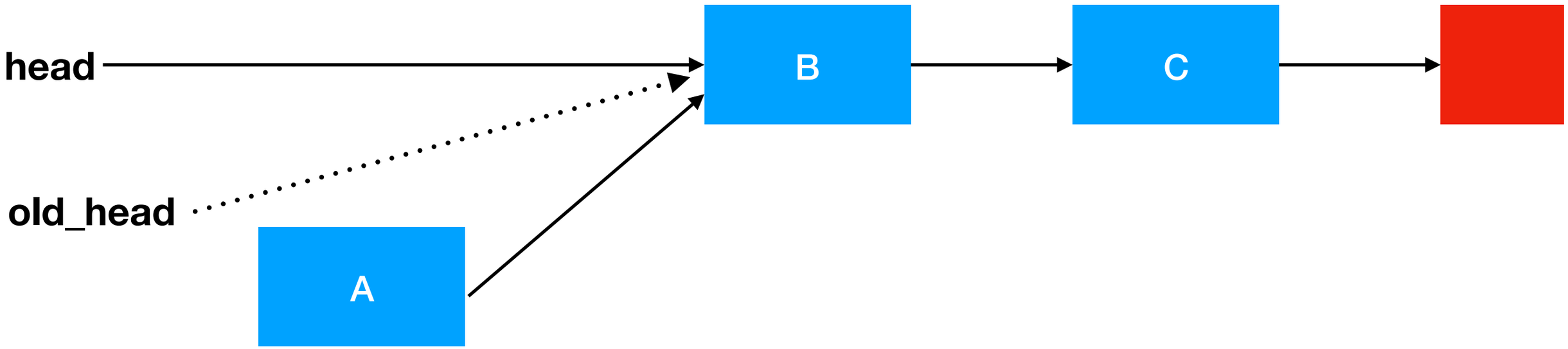
CAS push

Keep track of old head



CAS push

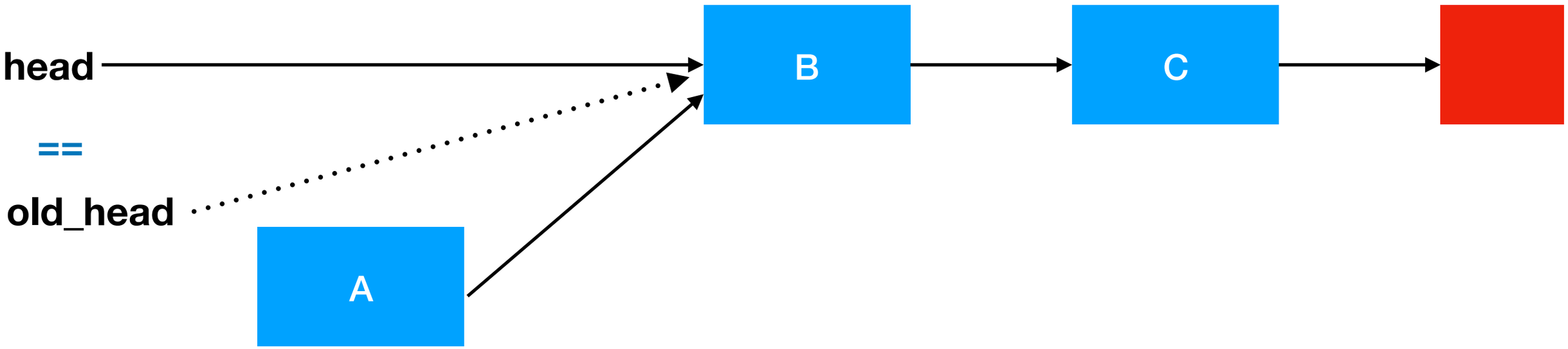
set new elements next pointer to old head



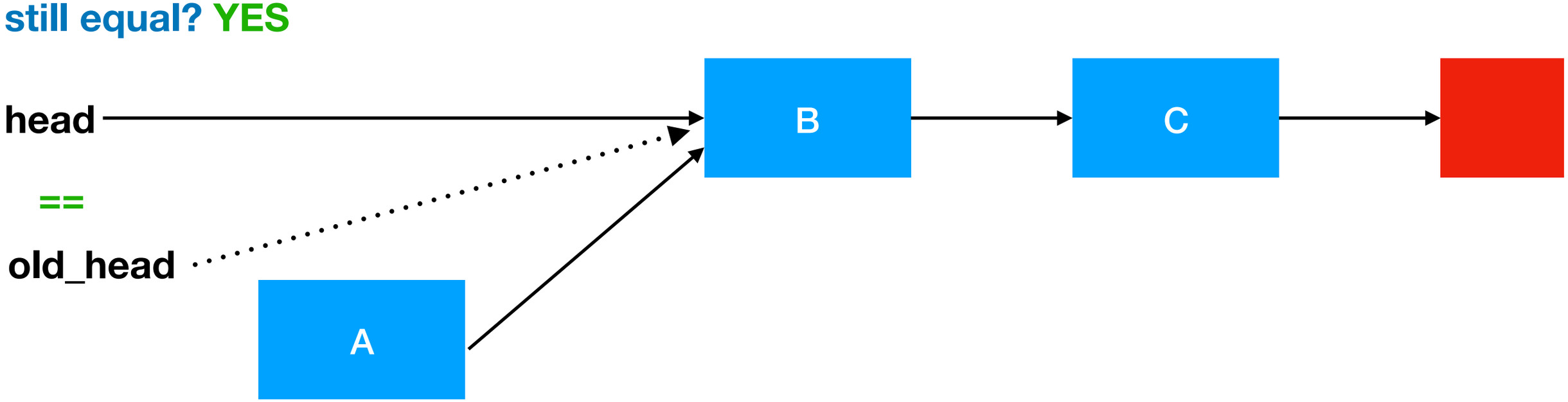
CAS push, success

start atomic operation

still equal?

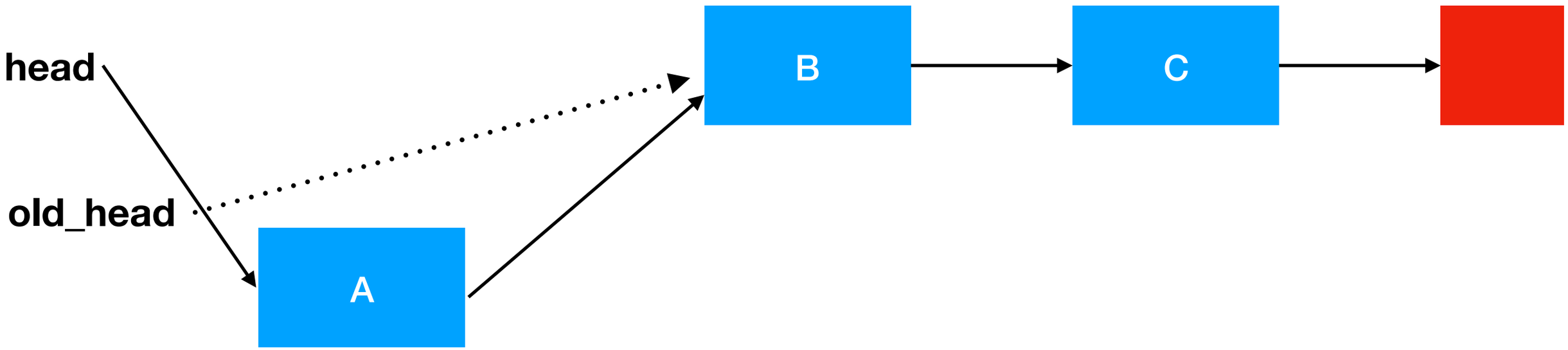


CAS push, success



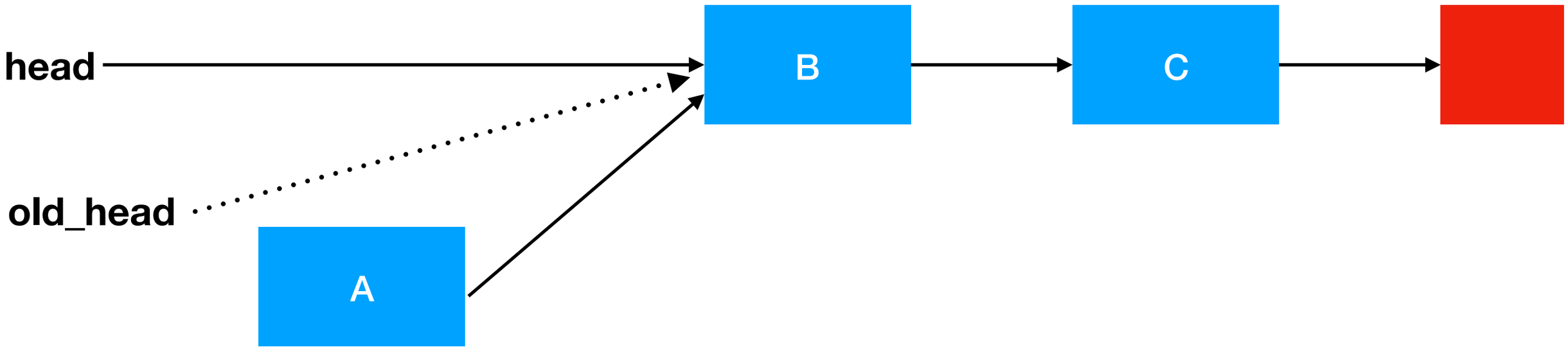
CAS push, success

set list head to new element



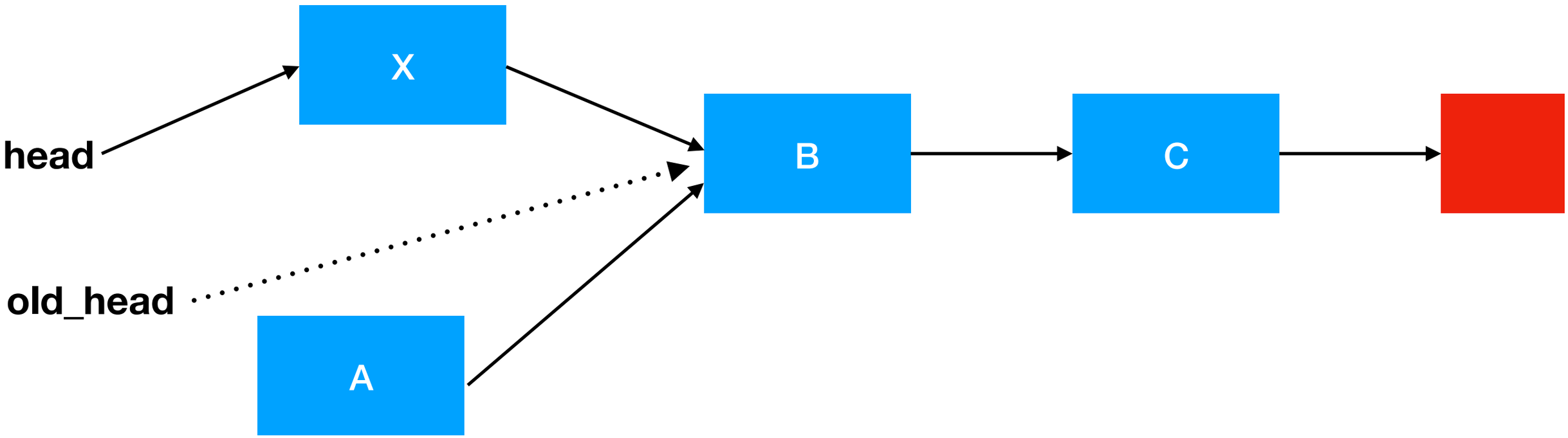
end atomic operation

CAS push

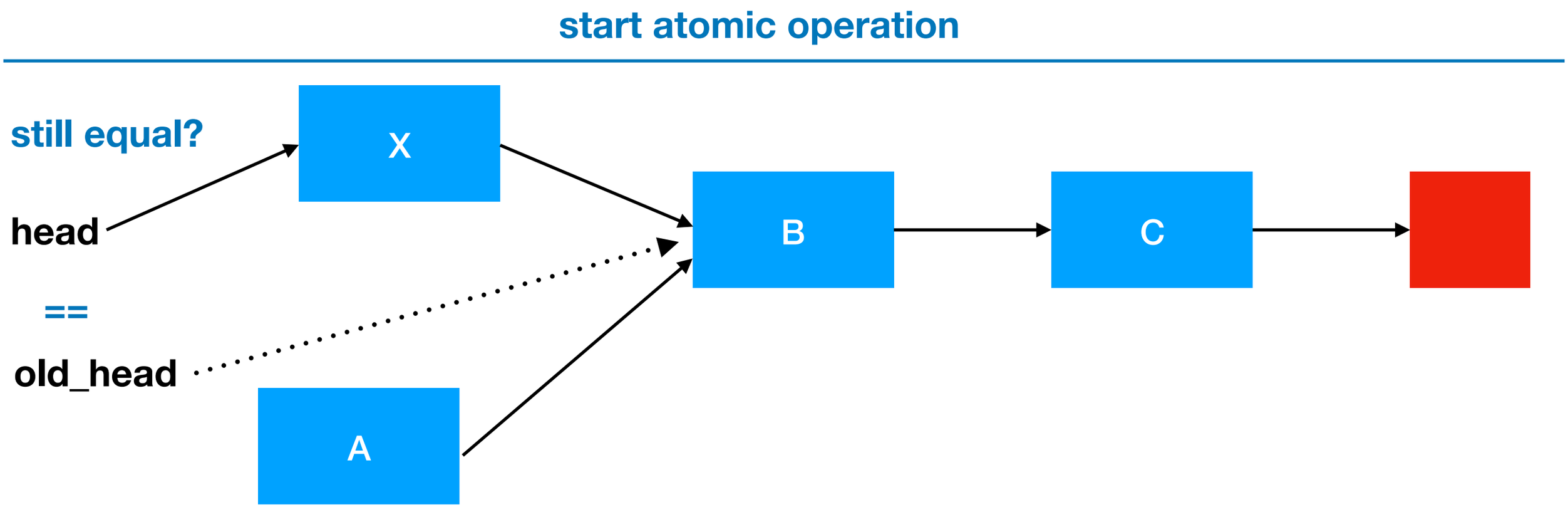


CAS push

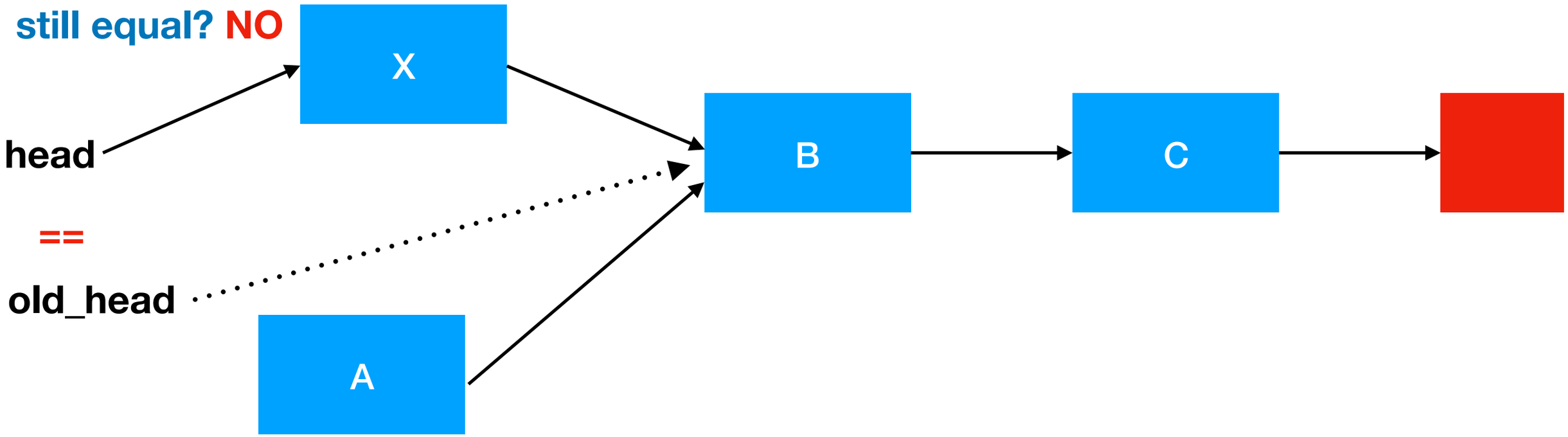
Another thread pushed X!



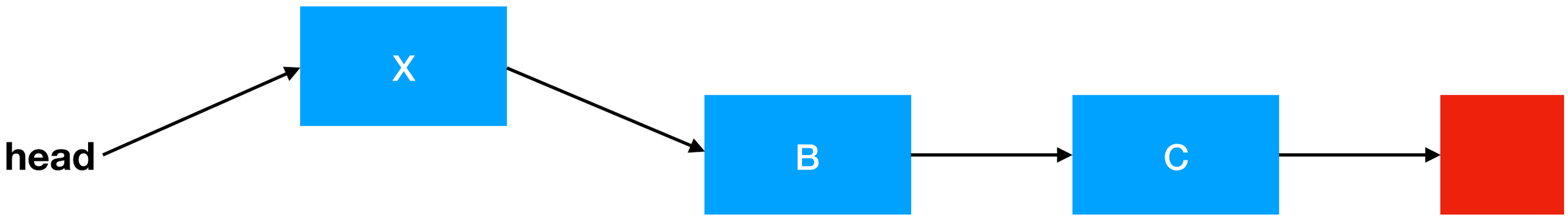
CAS push, failure



CAS push, failure



CAS push, failure



old_head

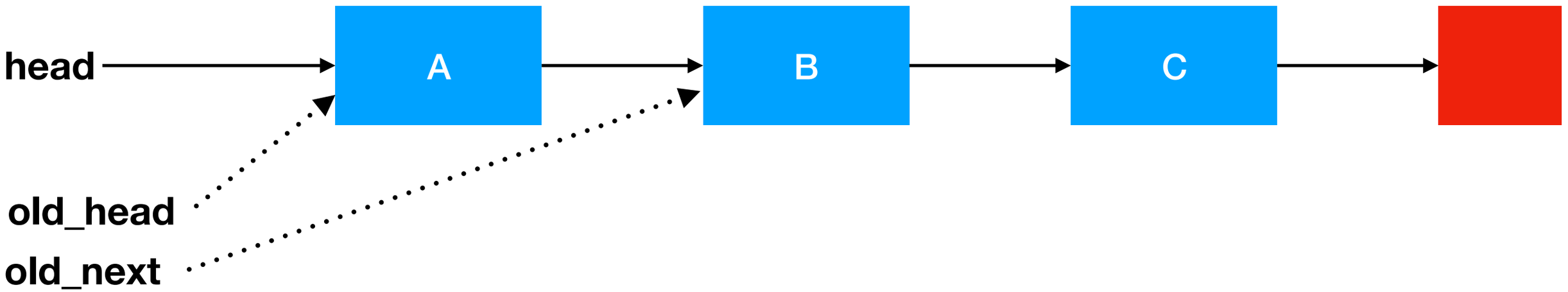
end atomic operation

ABA problem

- List elements can be re-used
  - Memory is limited, pointers can reappear => still low risk
  - Improve performance by keeping a pool of unused list elements
    => much greater risk of re-use!
- What if a list element is
  - popped,
  - pushed (with new content),
  - during the non-atomic part of a Pop?

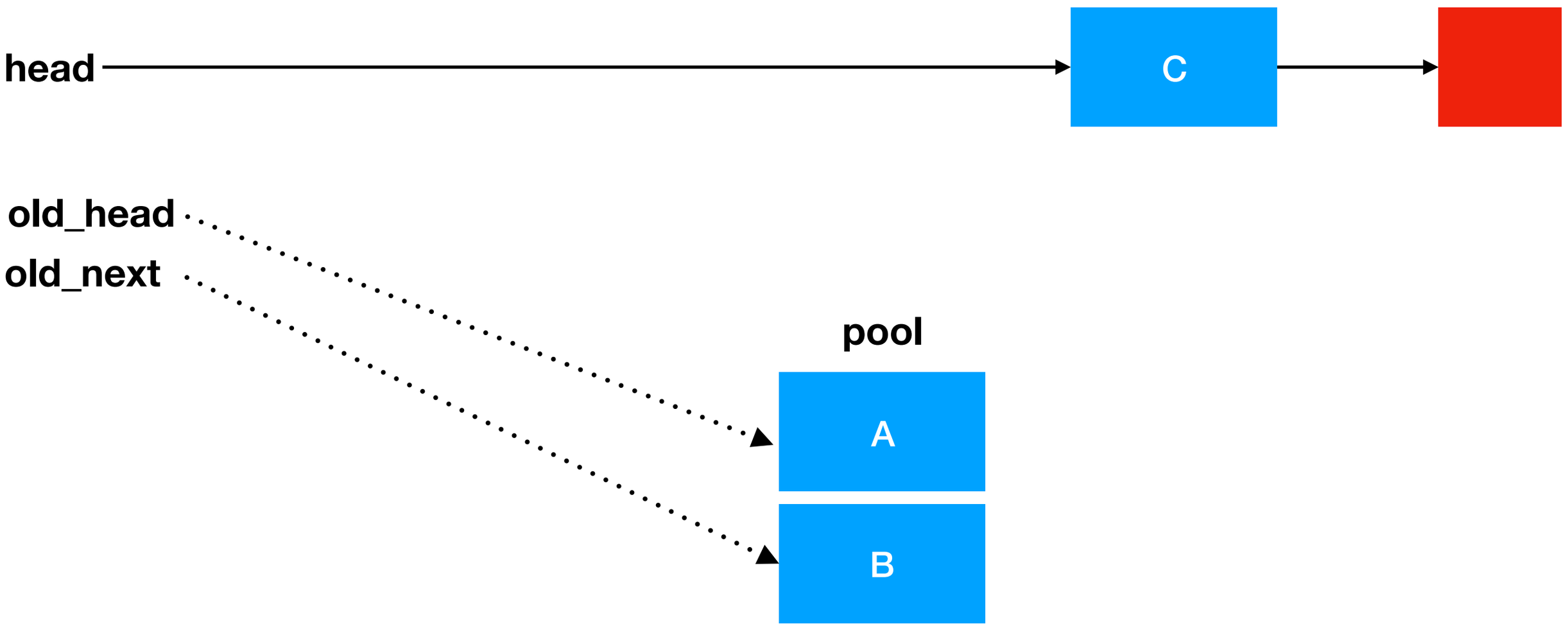
ABA problem

thread 0 starting pop



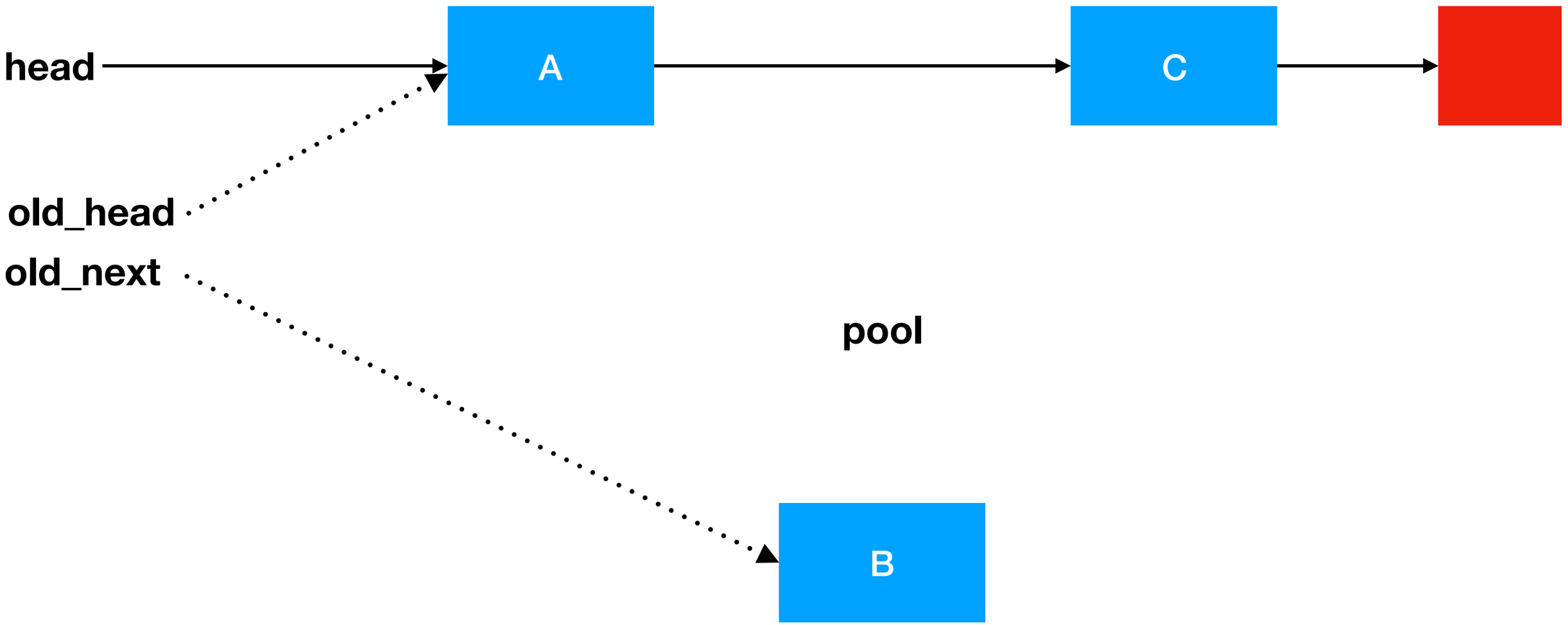
ABA problem

thread 1 pops A, thread 2 pops B



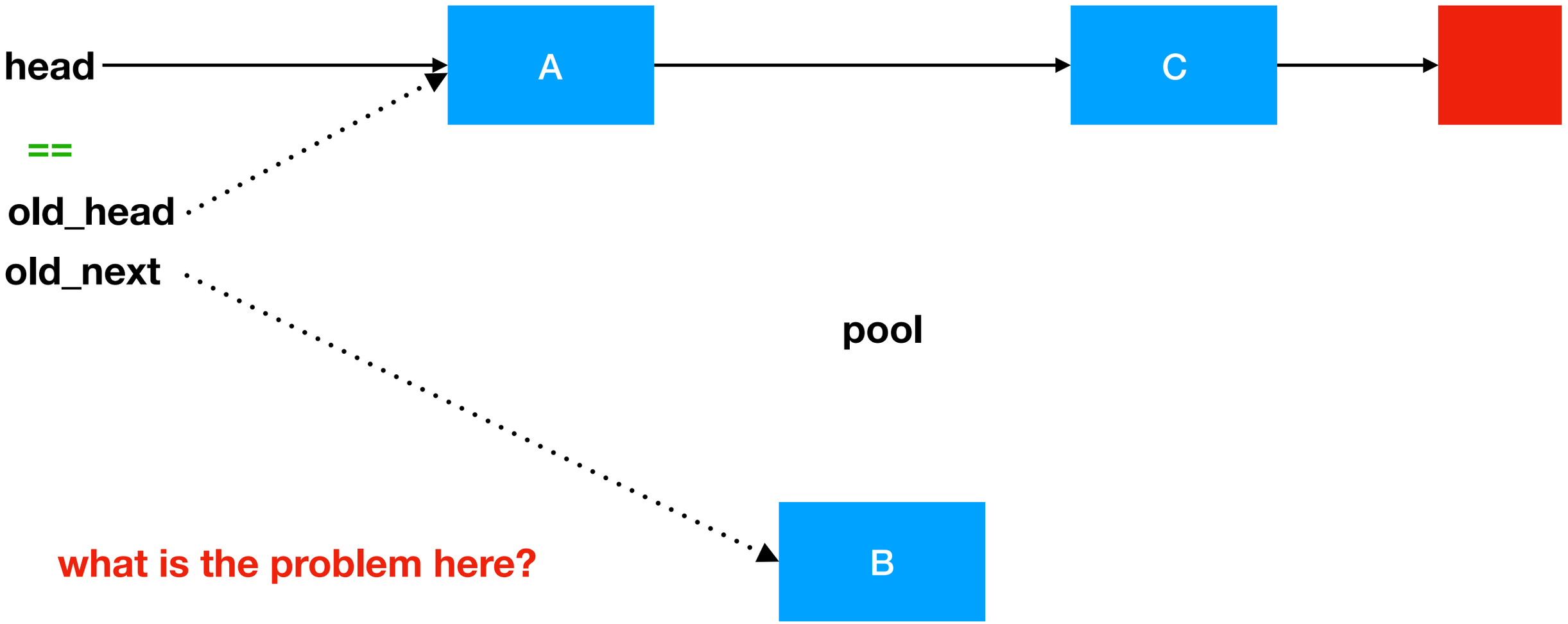
ABA problem

thread 1 pushes A



ABA problem

thread 0 resumes pop; enters atomic region:
compares head and old_head

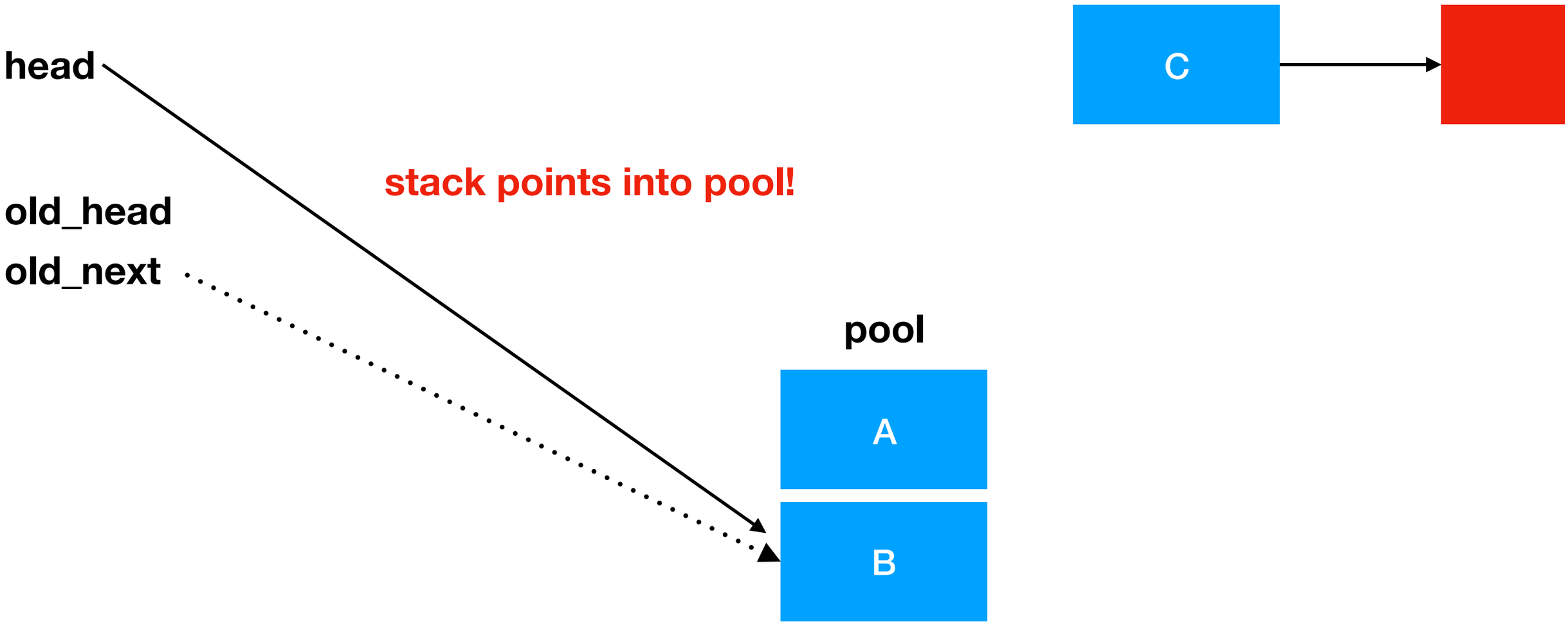


what is the problem here?

ABA problem

A is popped, setting head to old_next (B)

elements have leaked!



Lab 2

- Goal for the lab:
  - Implement non-blocking unbounded stack
  - Use atomic operations
  - Study the ABA problem
    - Detect it or force it to occur
    - Can it be avoided?