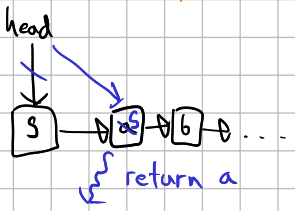
Lock-free unbounded queue:

enq(x):

① a.next = x
② tail = x

sentinel lets avoids the case where head = null.

deq():



head = head.next

First naive implementation of enq:

```
enq(v) {
   Node x = new Node(v);
   Node last;
   do {
      last = tail.get();
   } while(!last.next.compareAndSet(null, x));
   tail.set(x);
}
```

only one thread can pass here (because tail.next != null)
BUT all other threads will spin until tail is updated ☹
⇒ not lock-free

To ensure lock-freedom, other threads must try and advance tail themselves.

```
enq(v) {
   Node x = new Node(v);
   Node last, next;
   while (true) {
      last = tail.get();
      next = last.next.get();
      if(next == null) {
         if (last.next.compareAndSet(null, x)) {
            tail.compareAndSet(last, x);
            return;
         }
      else {
         tail.compareAndSet(last, next);
      }
   }
```

new

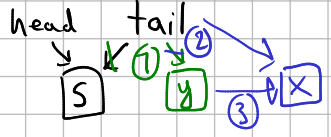
before we did tail.set(x), but now tail could be changed by other threads.

we know that a node's "next" will never change if next != null.

What if we enq did ② first and then ①?

Then the execution

A	B
enq(x)	enq(y)
deq() → empty	



(enq(y) not yet complete...)

④ deq must fail because it can't see y ☹

is possible which is not sequentially consistent (SC) ☹ (the queue cannot be empty for A with SC, since A just enqueued something and there was no other call to deq.)