Lock-free by Example

(one very complicated example)

Tony Van Eerd

C++Now, 2015

Guide to Threaded Coding

Guide to Threaded Coding

Use Locks

Guide to Threaded Coding

1. Forget what you learned in Kindergarten
   (ie stop Sharing)
2. Use Locks
3. Measure
4. Measure
5. Change your Algorithm
6. GOTO 1

∞. **Lock-free**

Lock-free coding is the last thing you want to do.

Guide to Threaded Coding

1. Forget what you learned in Kindergarten
   (ie stop Sharing)
2. Use Locks
3. Measure
4. Measure
5. Change your Algorithm
6. GOTO 1

∞. **Lock-free**
∞+1. Measure. Measure.

Lock-free coding is the last thing you want to do.

Guide to Threaded Coding

Don’t Share
Use Locks

NOTE:

CAS = compare_exchange (_weak or _strong)

NOTE:

CAS = compare_exchange

Not my coding style/structure

NOTE:

CAS = compare_exchange

Not my coding style/structure

Remember to lower the audience's expectations:

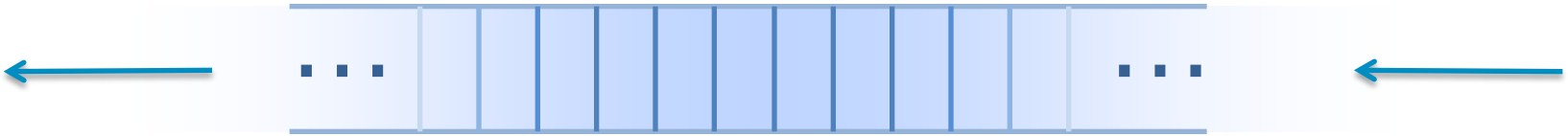
NOTE:

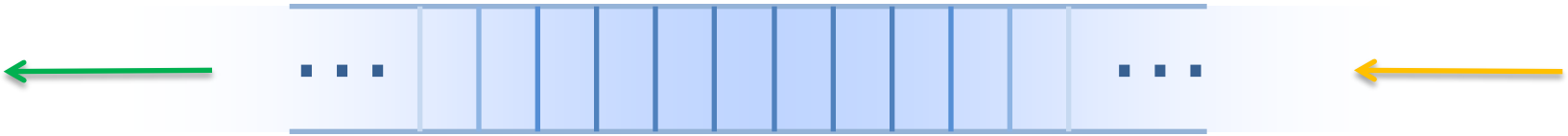
CAS = compare_exchange

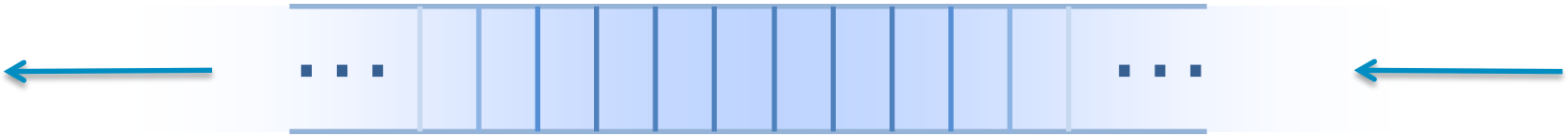
Not my coding style/structure

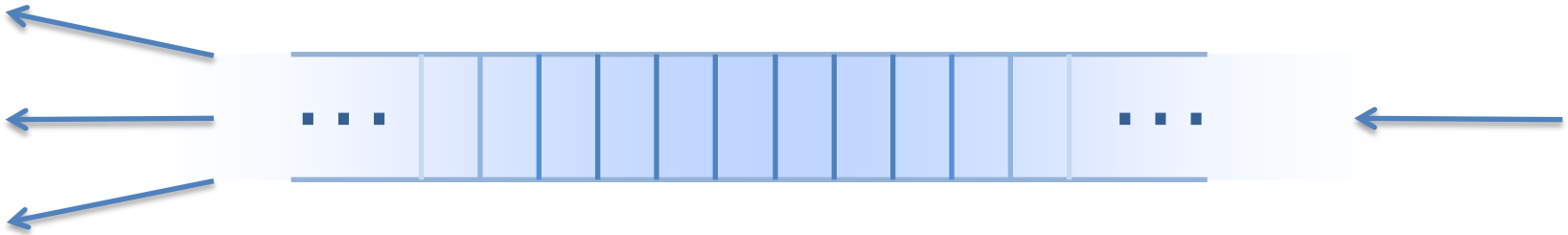
Remember to lower the audience's expectations:

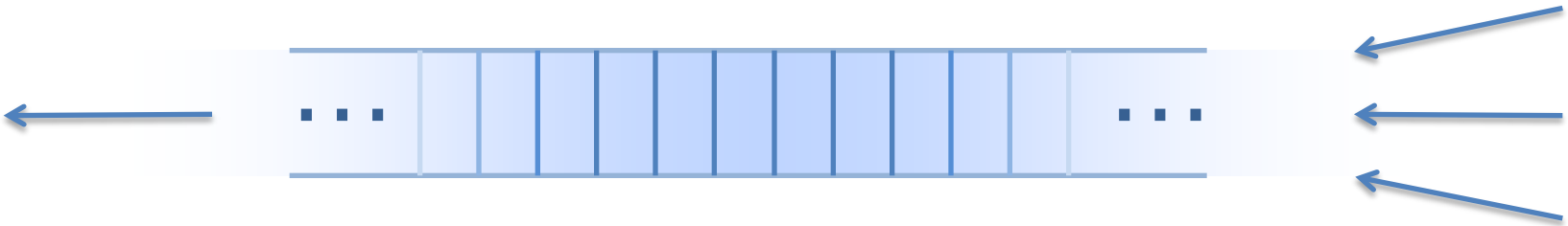
I'm not an expert.

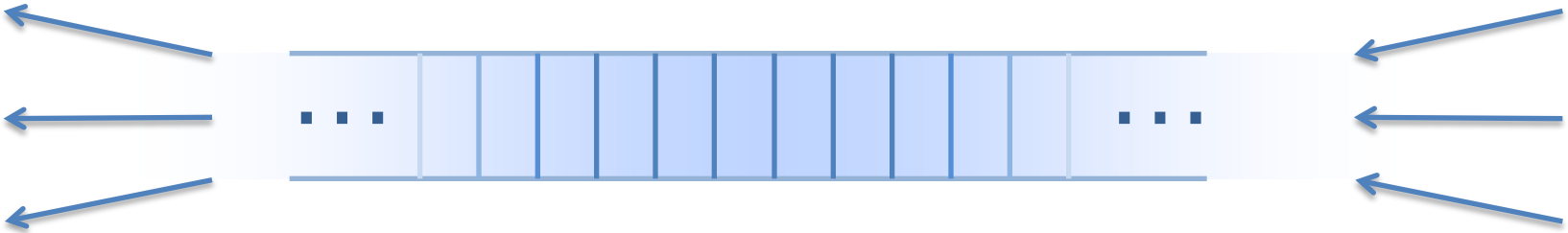




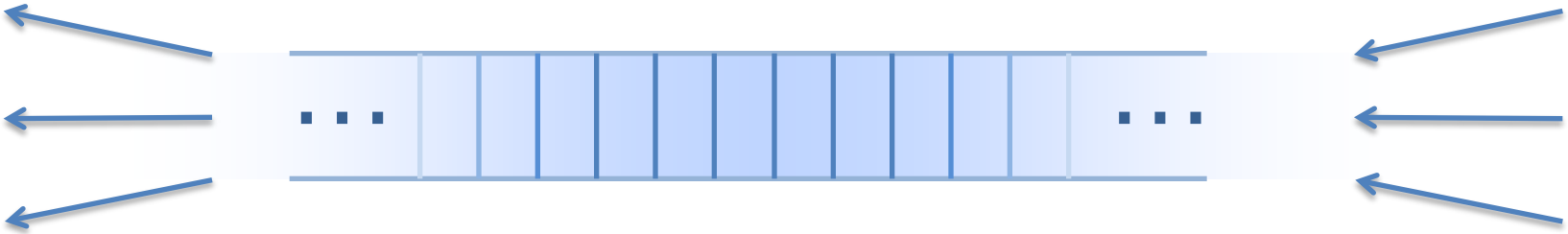




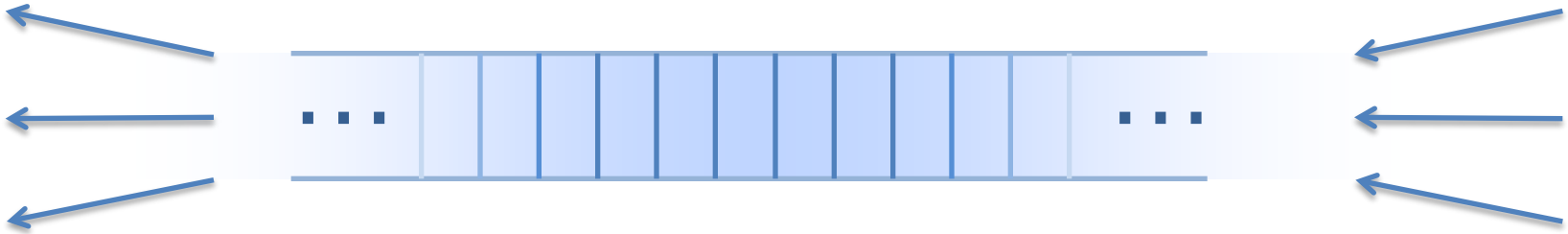




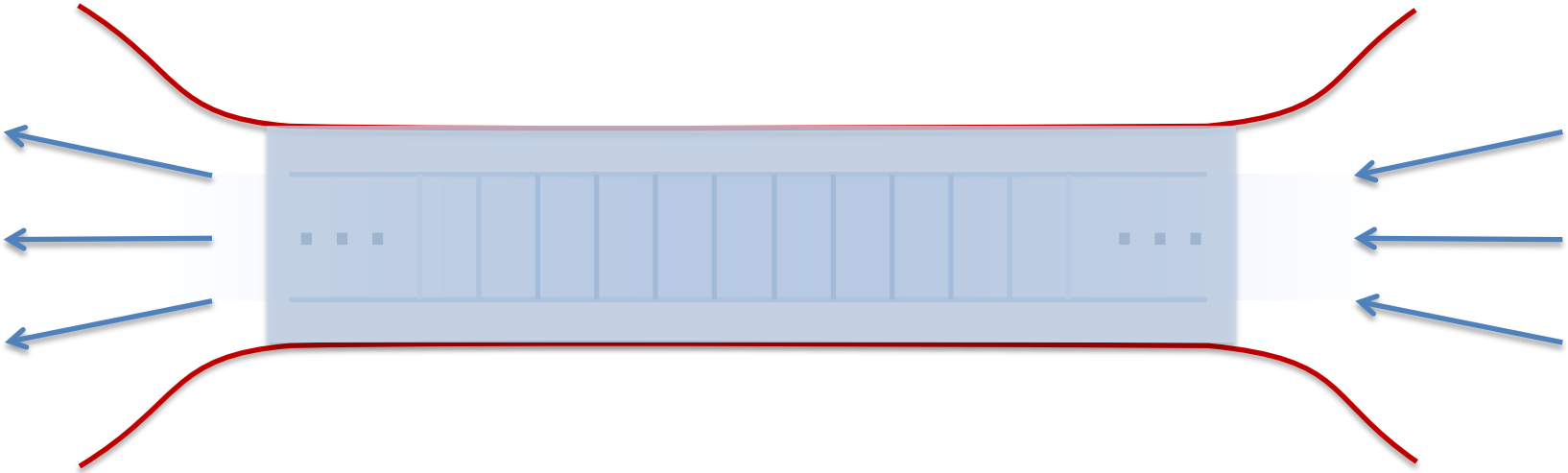
Multi-Producer Multi-Consumer Queue



MPMC Queue

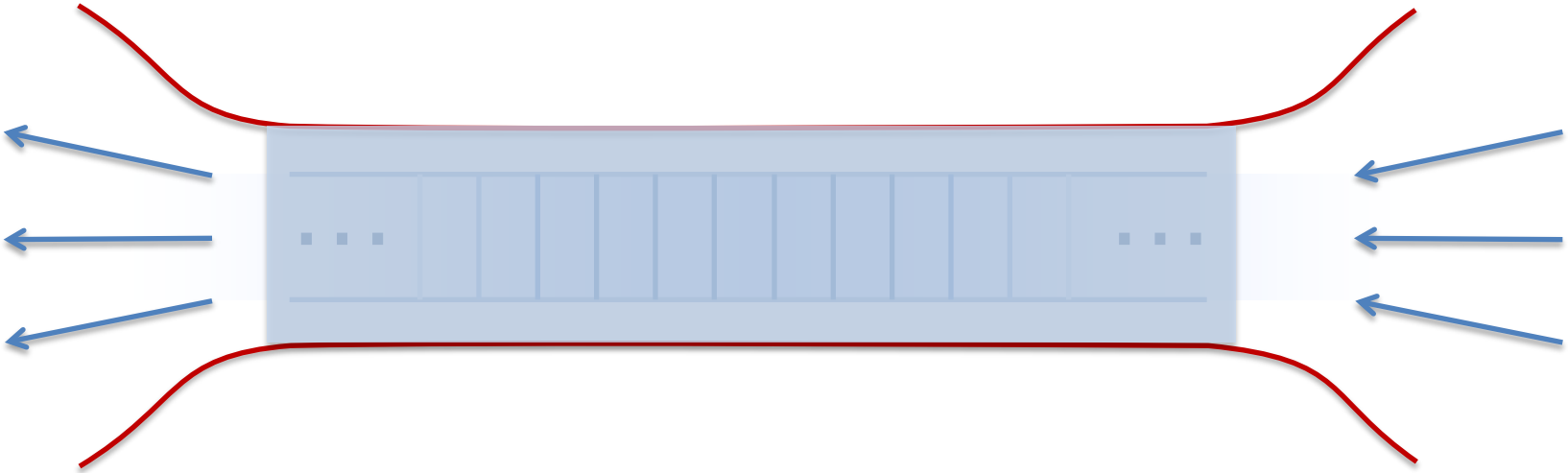


MPMC Queue

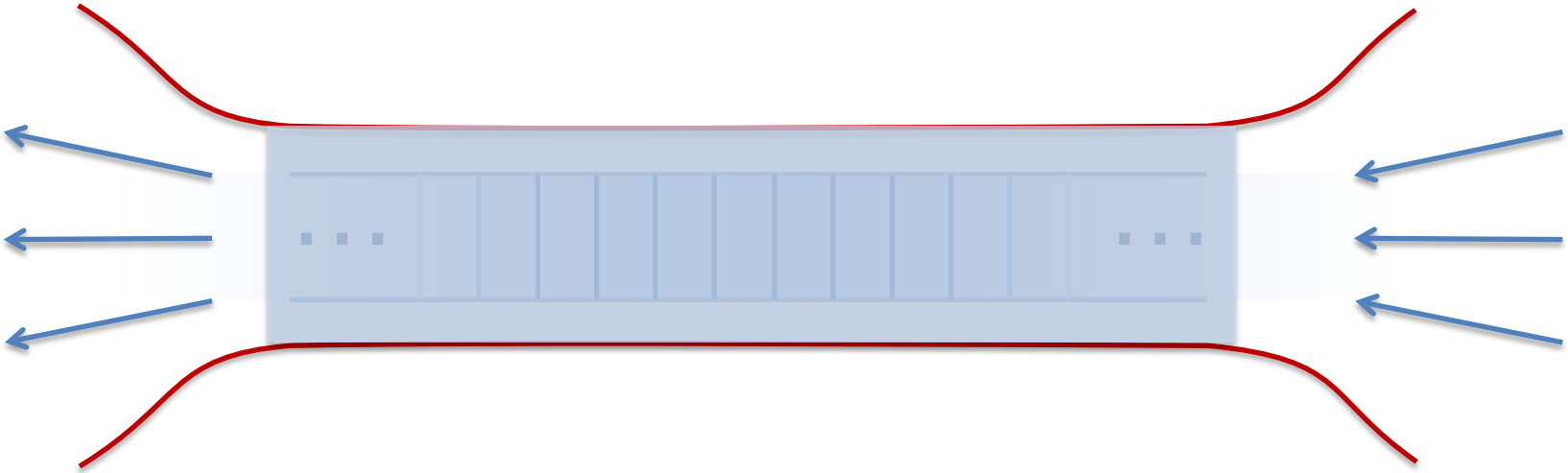


MPMC Queue

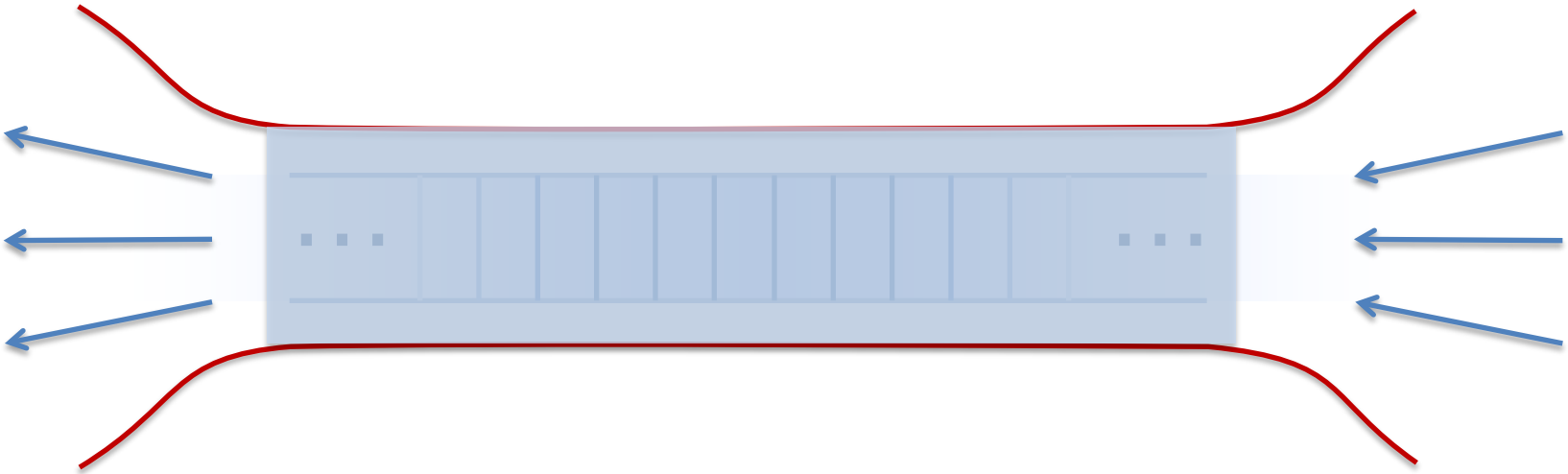
SPSC
SPMC
MPSC
MPMC

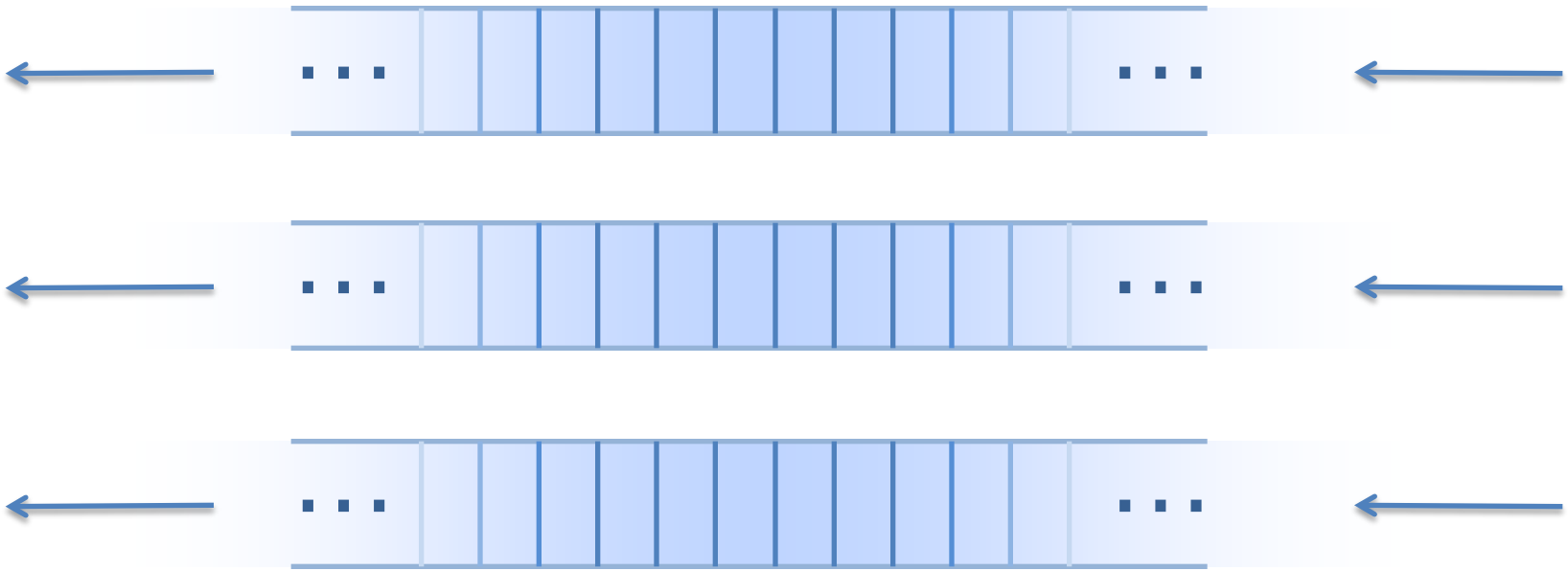


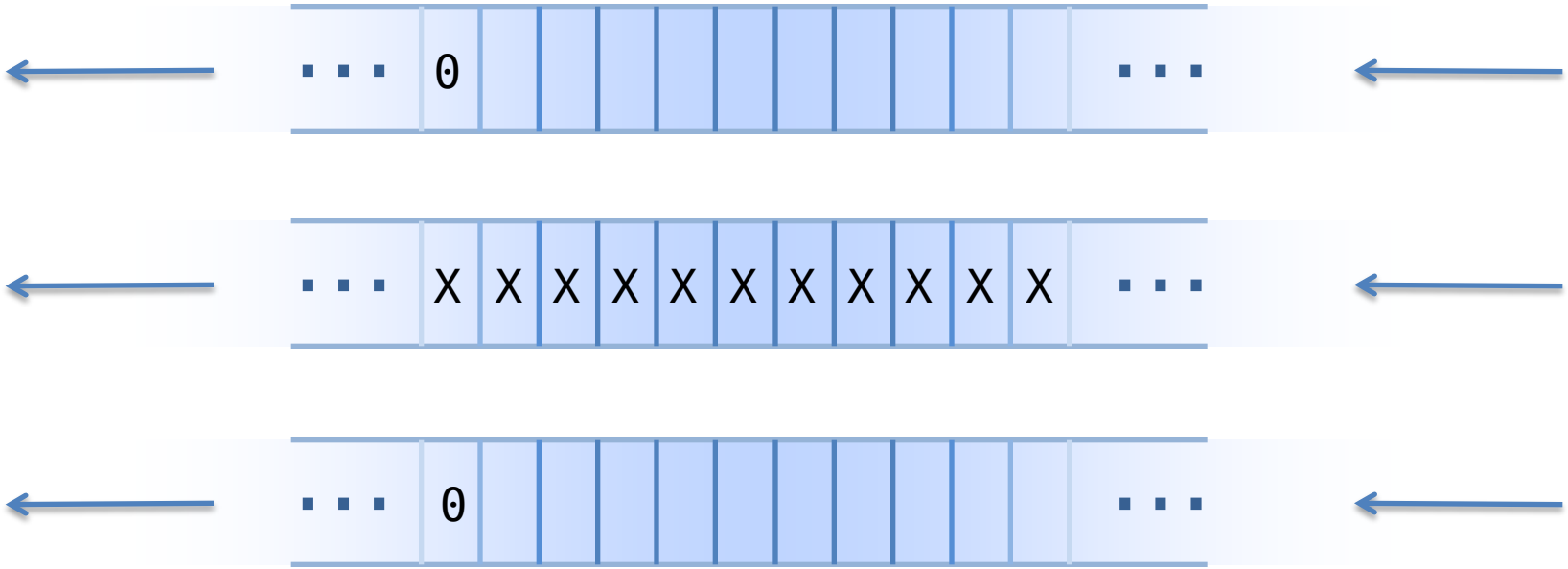
MPMC Queue

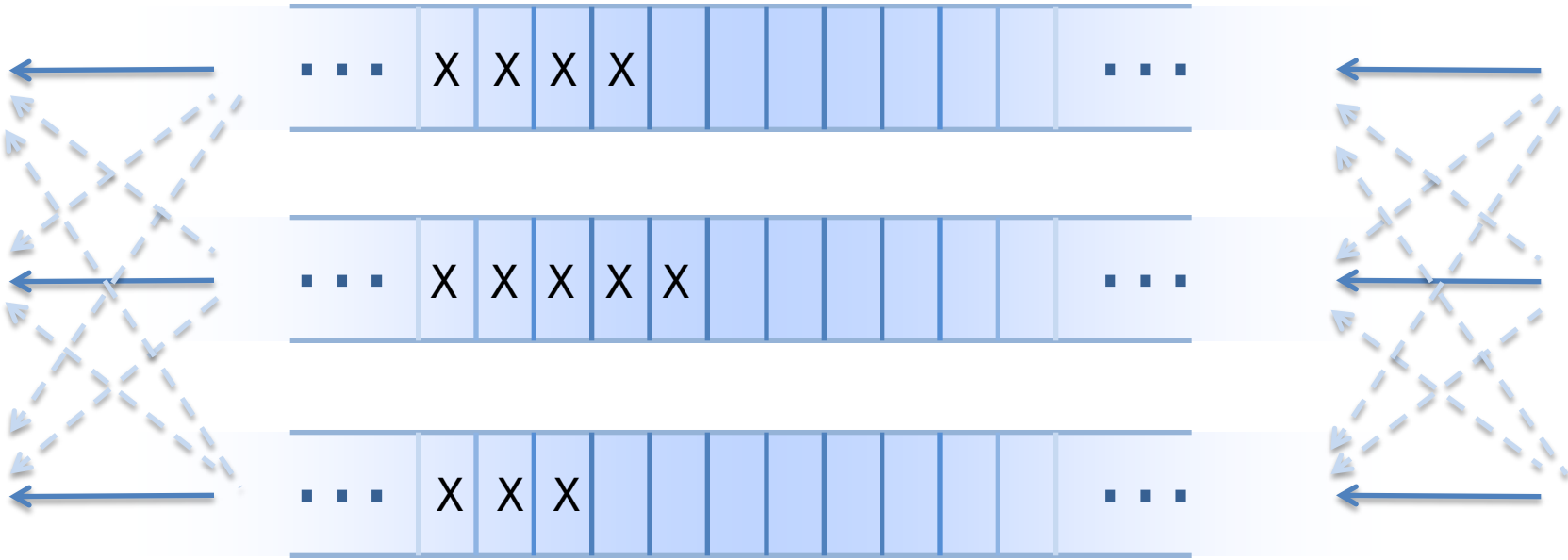


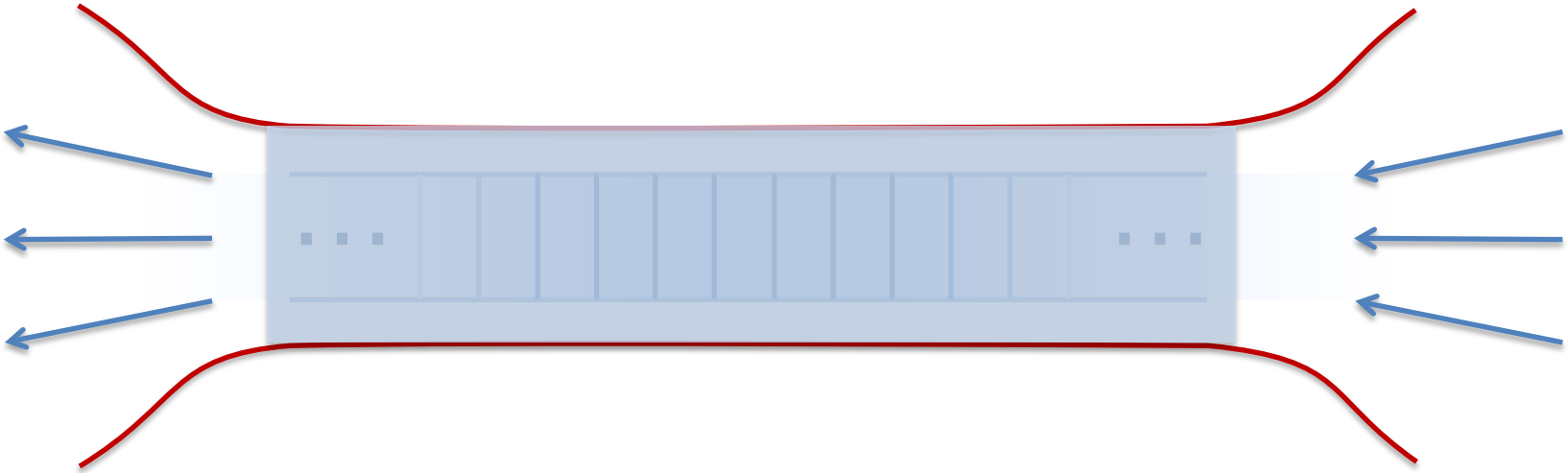
Bottleneck

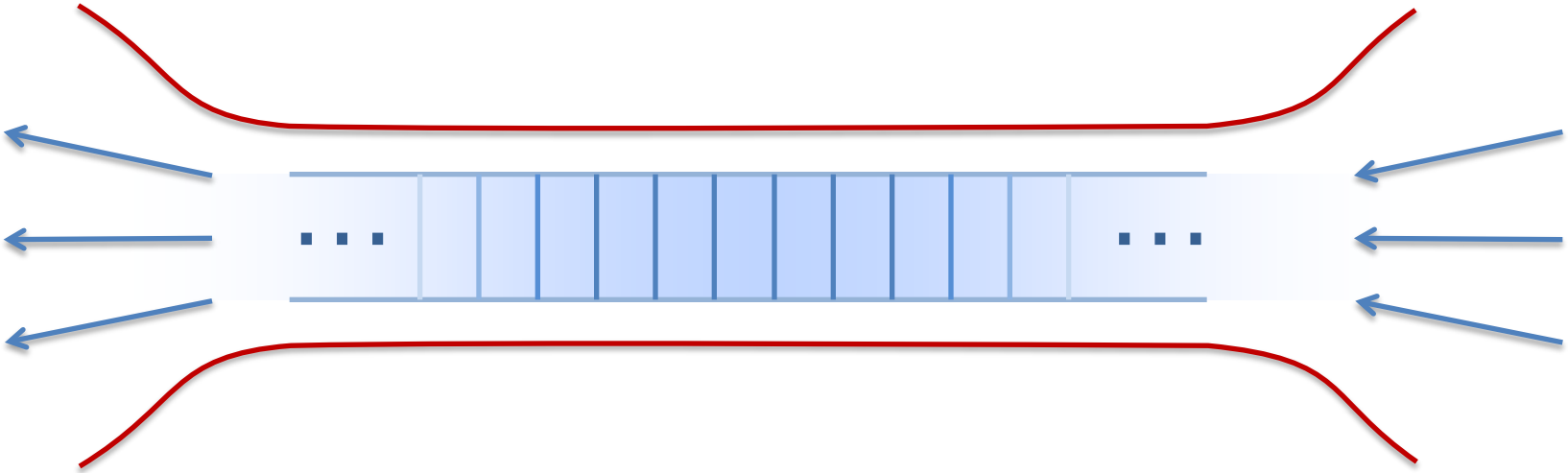


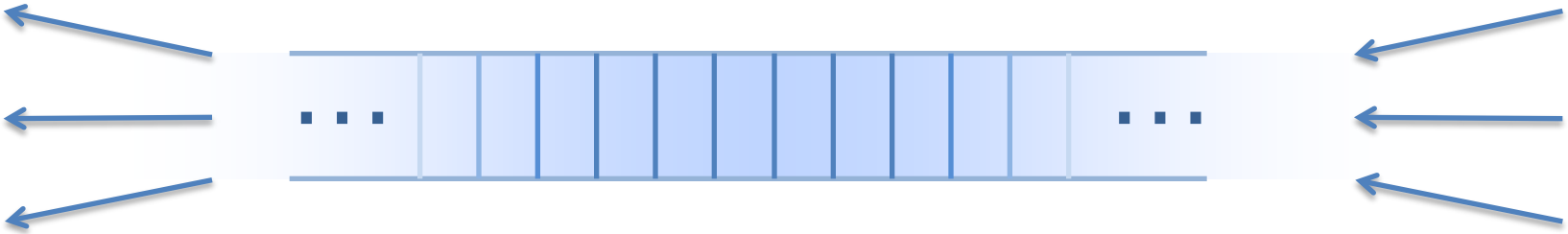


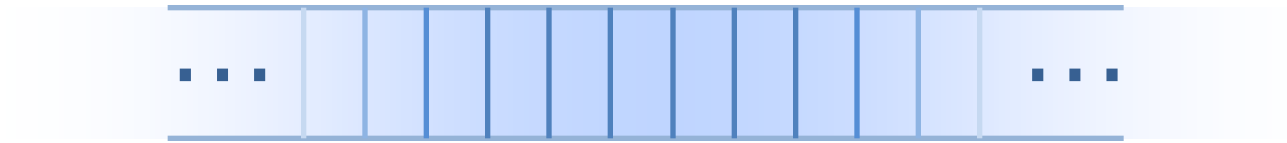

...
0
...
...
X X X X X X X X X X
...
...
0
...

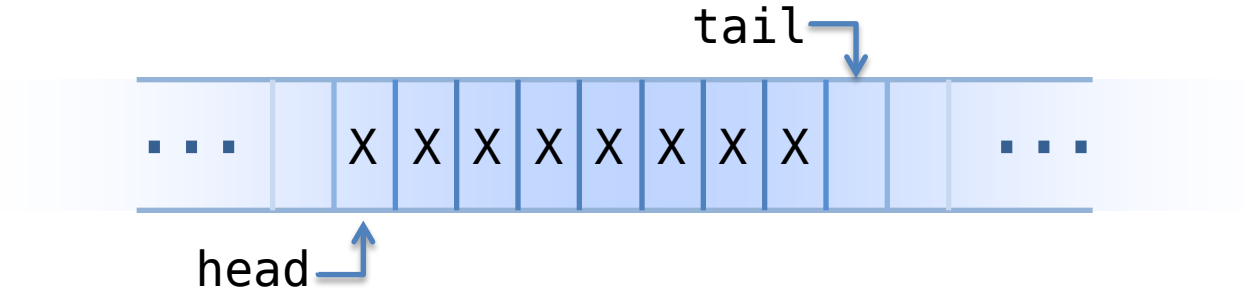

. . .
X X X X
. . .
. . .
X X X X X
. . .
. . .
X X X
. . .



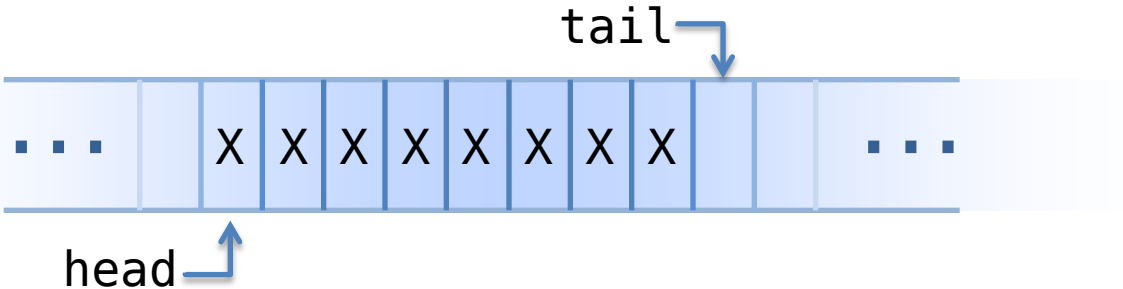




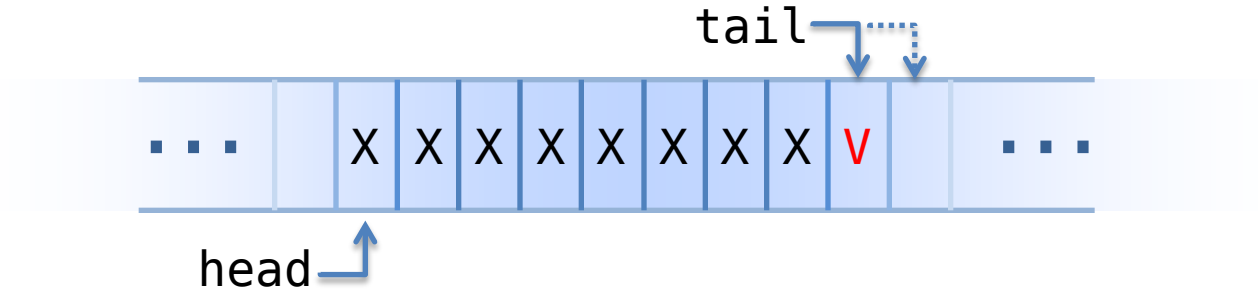



tail
...
X X X X X X X X
...
head

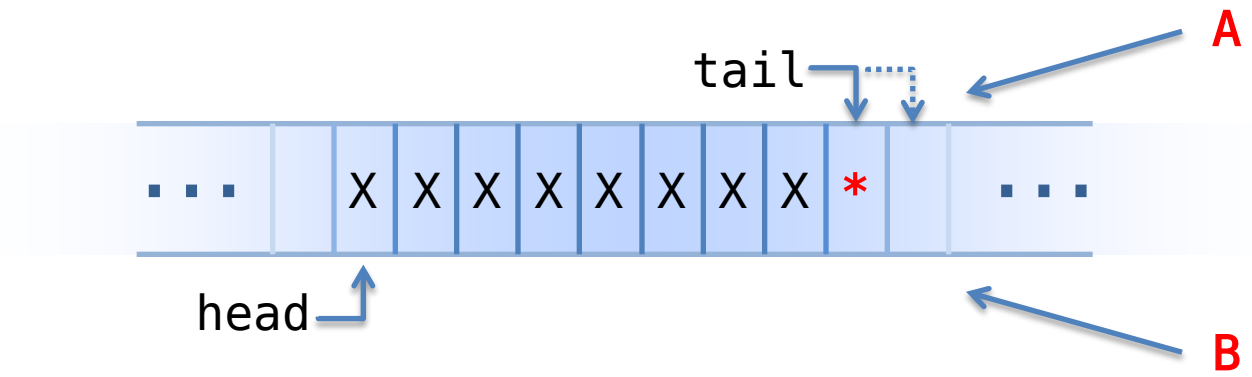
```
class Queue
{
    int buffer[some_size];
    size_t head;
    size_t tail;
};
```



```
void push(int val)
{
    buffer[tail++] = val;
}
```

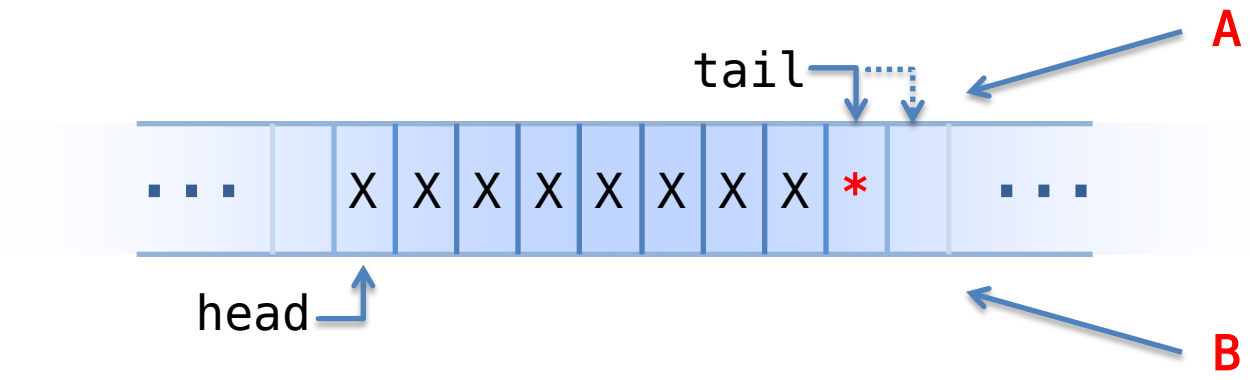


```
void push(int val)
{
    buffer[tail++] = val;
}
```

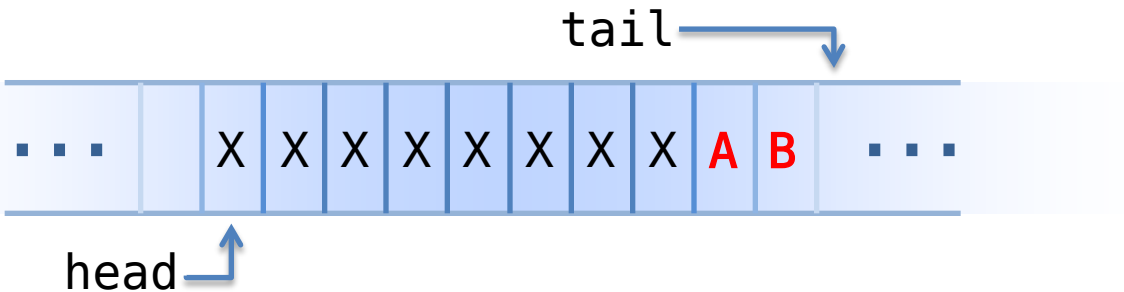


```
void push(int val)
{
    buffer[tail++] = val;
}
```

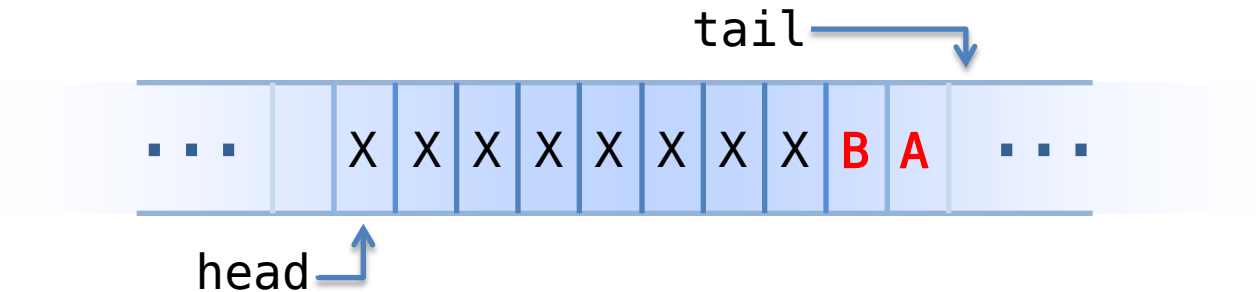
Possible Outcomes?



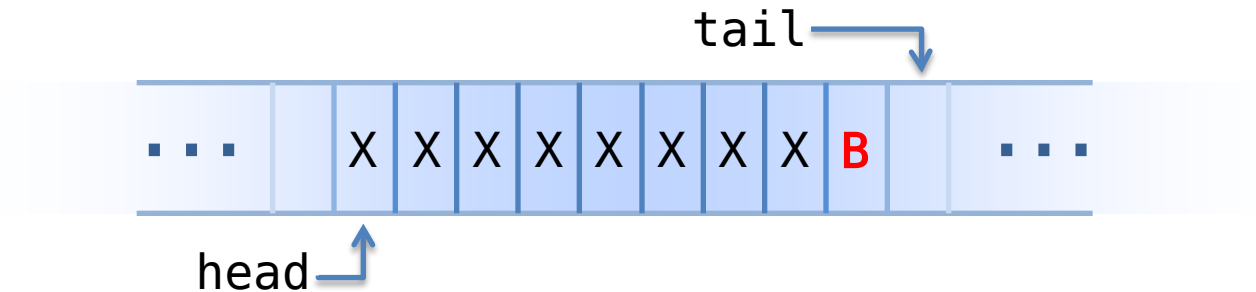
```
void push(int val)
{
    buffer[tail++] = val;
}
```



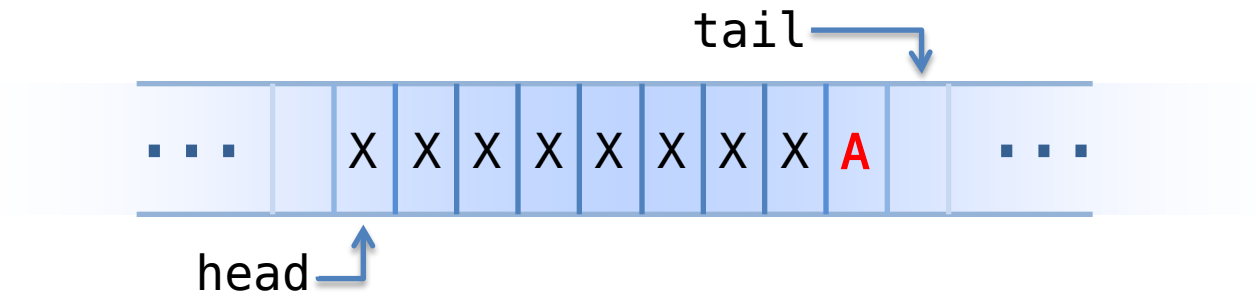
```
void push(int val)
{
    buffer[tail++] = val;
}
```



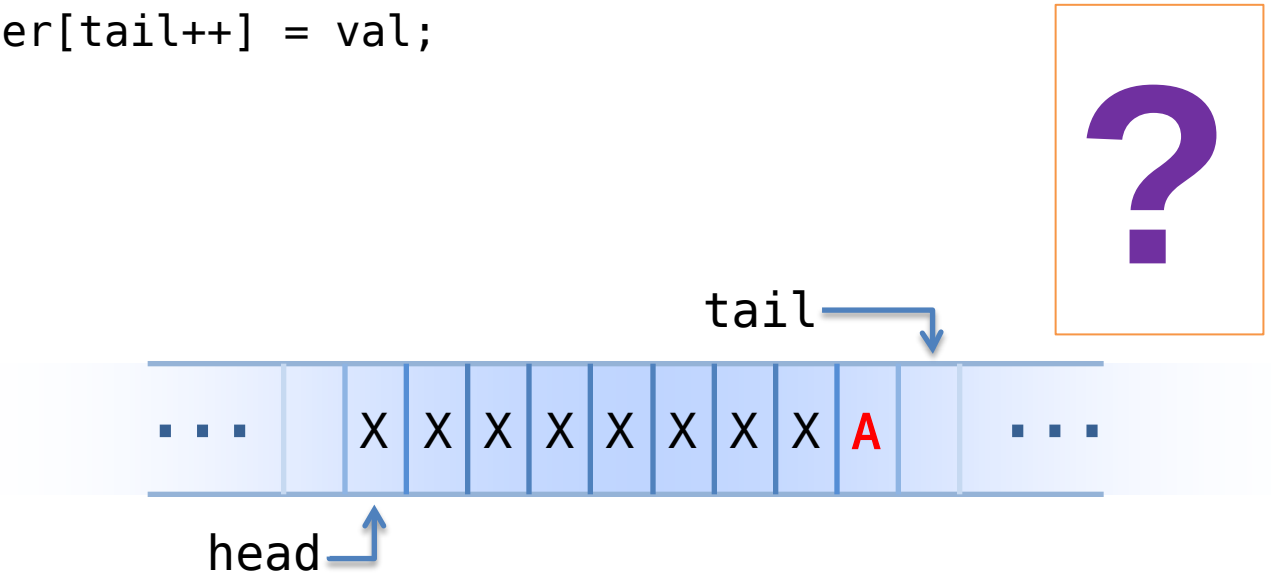
```
void push(int val)
{
    buffer[tail++] = val;
}
```



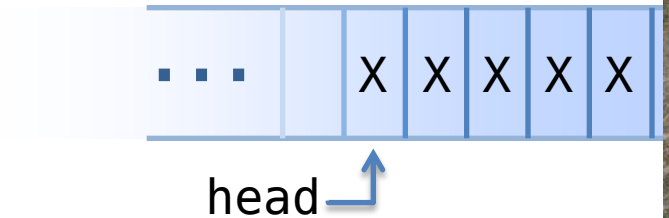
```
void push(int val)
{
    buffer[tail++] = val;
}
```



```
void push(int val)
{
    buffer[tail++] = val;
}
```



```
void push(int val)
{
    buffer[tail++] = val;
}
```

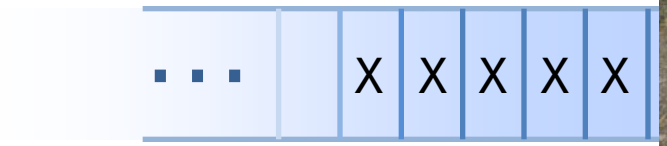


```
void push(int val)
{
    buffer[tail++] = val;
}
```

```
class Queue
{
    int buffer[some_size];
    size_t head;
    size_t tail;
};
```

A

B



```
void push(int val)
{
    buffer[tail++] = val;
}
```

```
class Queue
{
    int buffer[some_size];
    size_t head;
    size_t tail;
};
```

A

B

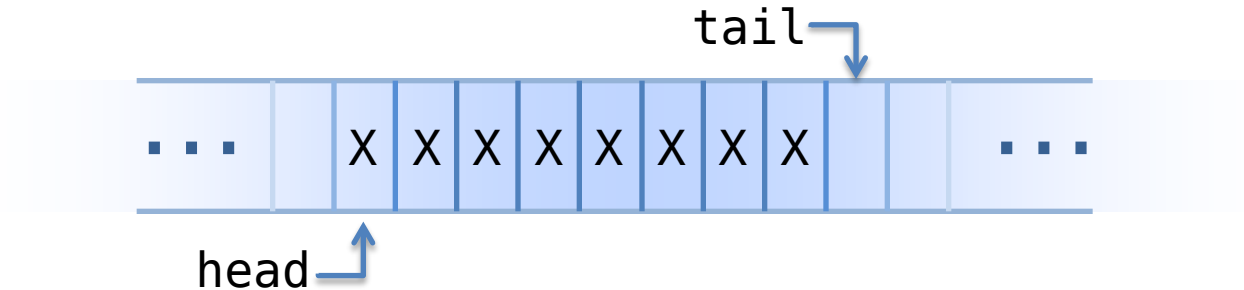
UNDEFINED BEHAVIOUR

```
void push(int val)
{
    buffer[tail++] = val;
}
```

```
class Queue
{
    int buffer[some_size];
    size_t head;
    size_t tail;
};
```

A

B

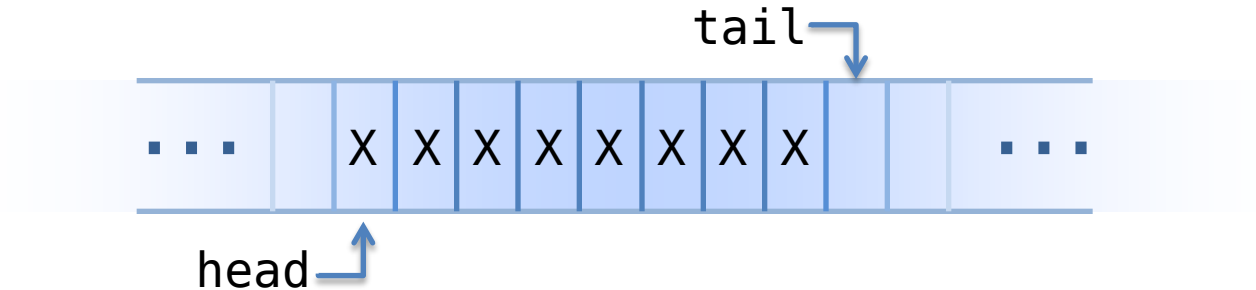


```
void push(int val)
{
    buffer[tail++] = val;
}
```

```
class Queue
{
    atomic<int> buffer[SZ];
    atomic<size_t> head;
    atomic<size_t> tail;
};
```

A

B

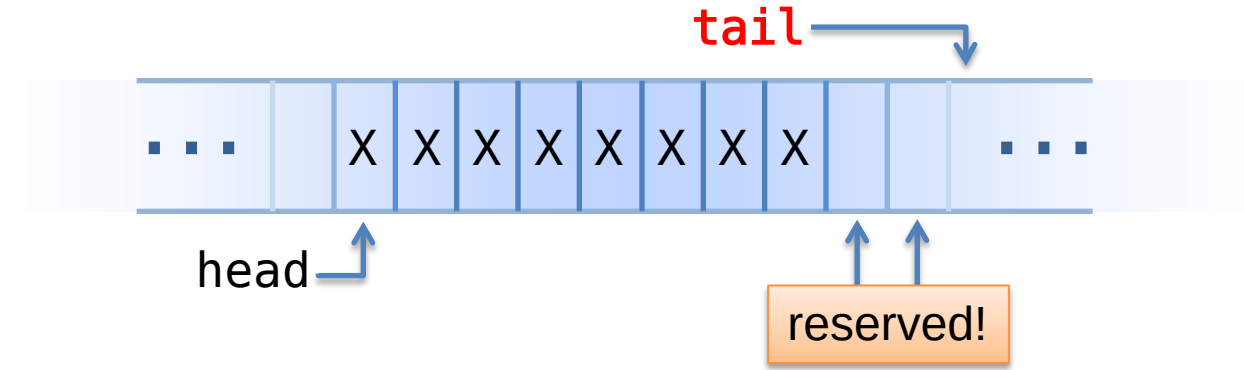


```
void push(int val)
{
    buffer[tail++] = val;
}
```

```
class Queue
{
    atomic<int> buffer[SZ];
    atomic<size_t> head;
    atomic<size_t> tail;
};
```

A

B

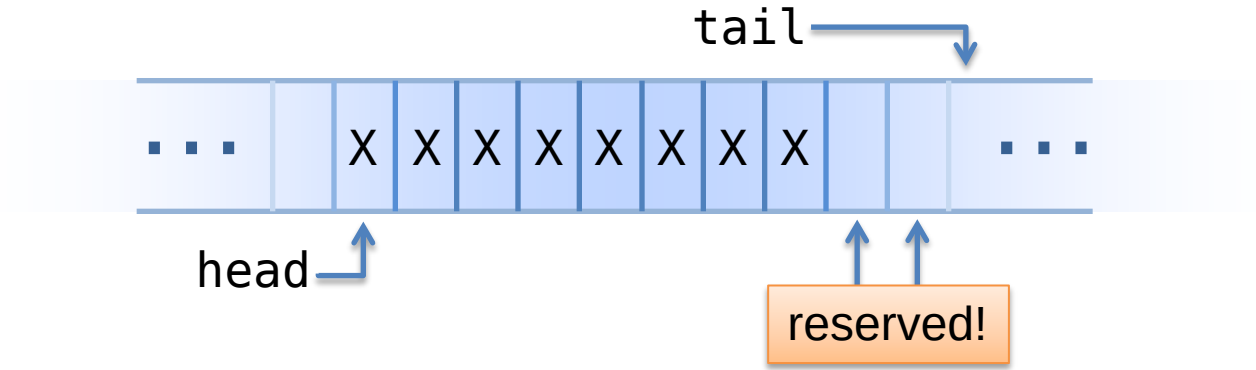


```
void push(int val)
{
    buffer[tail++] = val;
}
```

```
class Queue
{
    int buffer[SZ];
    atomic<size_t> head;
    atomic<size_t> tail;
};
```

A

B

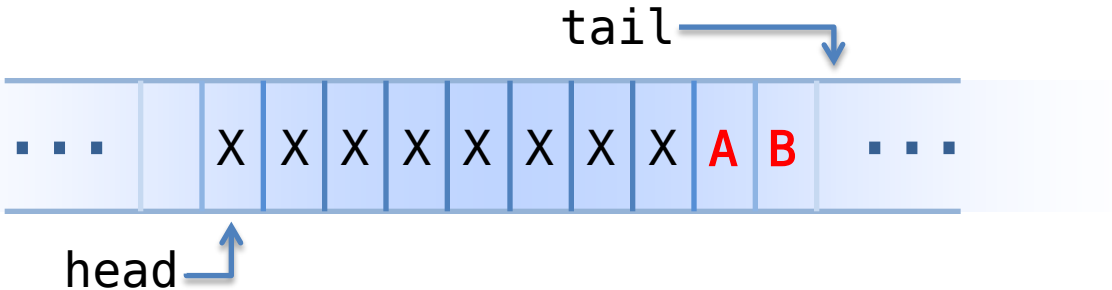


```
void push(int val)
{
    buffer[tail++] = val;
}
```

```
class Queue
{
    int buffer[SZ];
    atomic<size_t> head;
    atomic<size_t> tail;
};
```

A

B

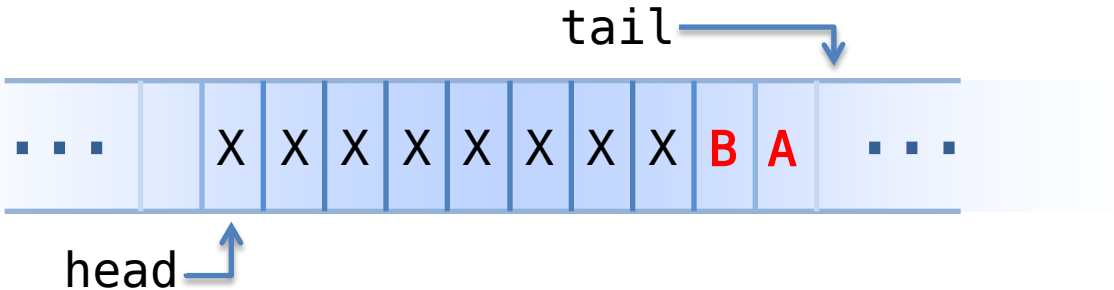


```
void push(int val)
{
    buffer[tail++] = val;
}
```

```
class Queue
{
    int buffer[SZ];
    atomic<size_t> head;
    atomic<size_t> tail;
};
```

A

B

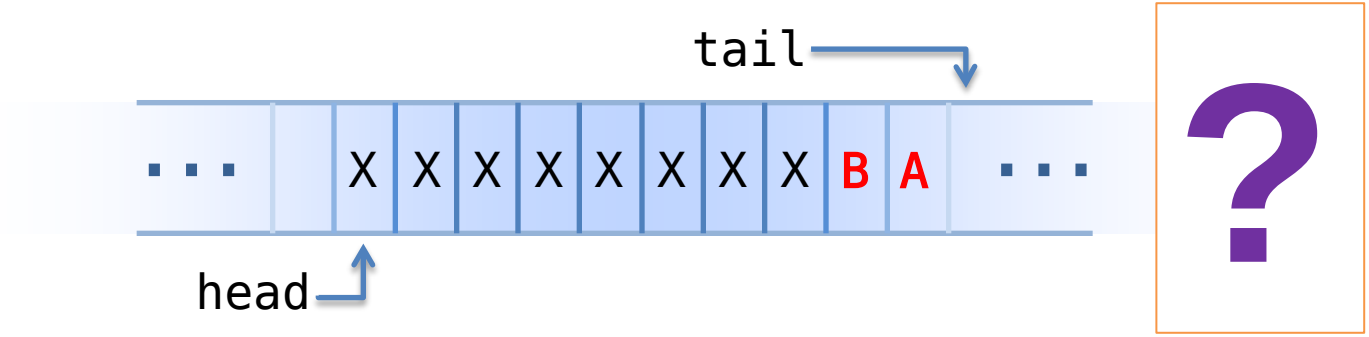


```
void push(int val)
{
    buffer[tail++] = val;
}
```

```
class Queue
{
    int buffer[SZ];
    atomic<size_t> head;
    atomic<size_t> tail;
};
```

A

B

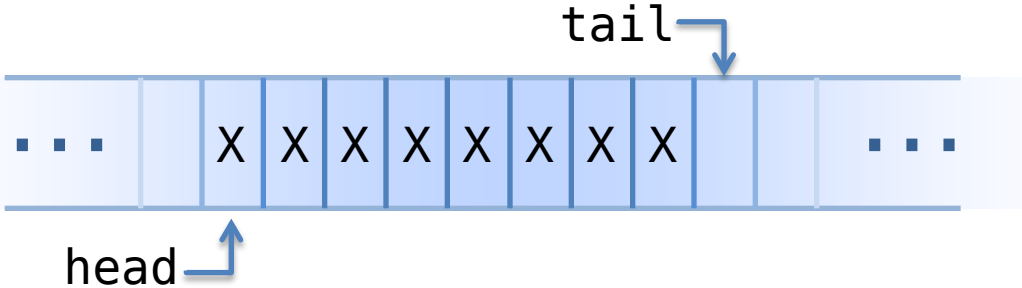


```
void push(int val)
{
    buffer[tail++] = val;
}
```

```
class Queue
{
    int buffer[SZ];
    atomic<size_t> head;
    atomic<size_t> tail;
};
```

A

B

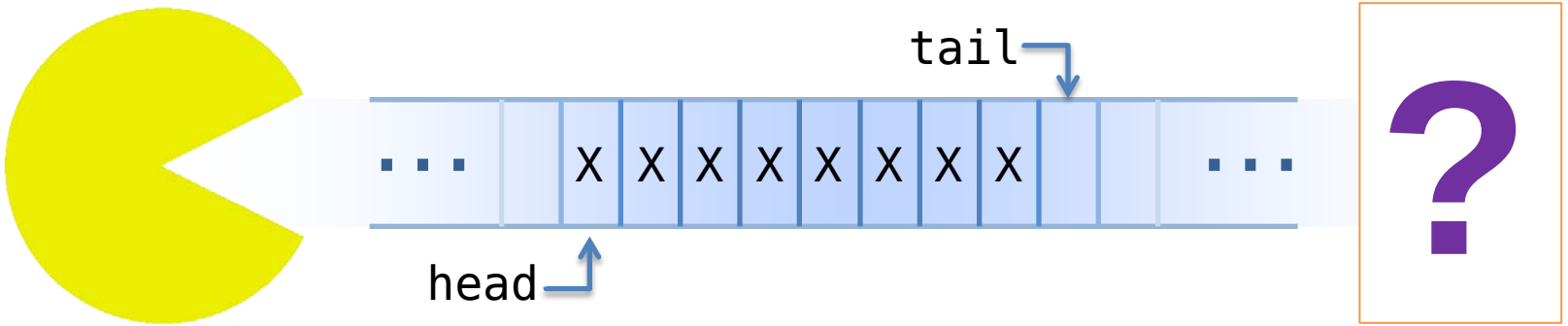


```
void push(int val)
{
    buffer[tail++] = val;
}
```

```
class Queue
{
    int buffer[SZ];
    atomic<size_t> head;
    atomic<size_t> tail;
};
```

A

B

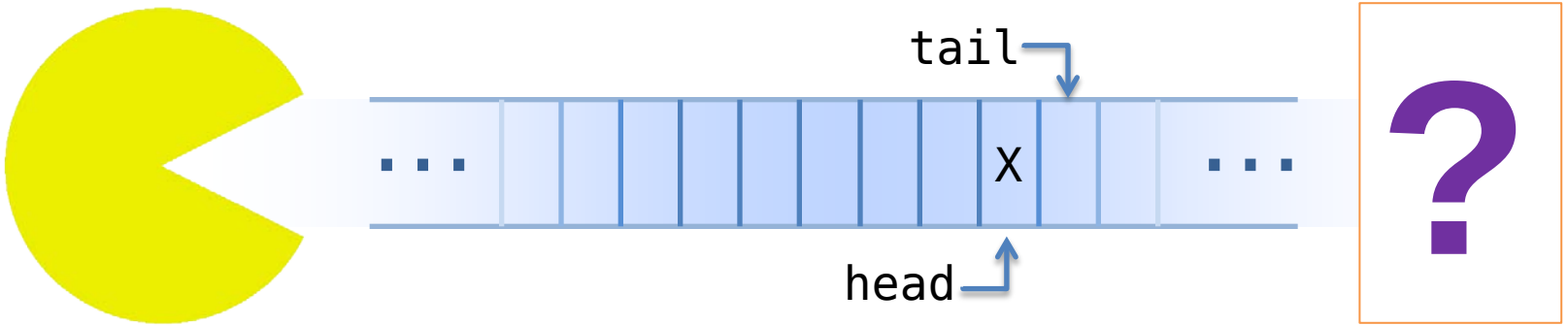


```
void push(int val)
{
    buffer[tail++] = val;
}
```

```
class Queue
{
    int buffer[SZ];
    atomic<size_t> head;
    atomic<size_t> tail;
};
```

A

B



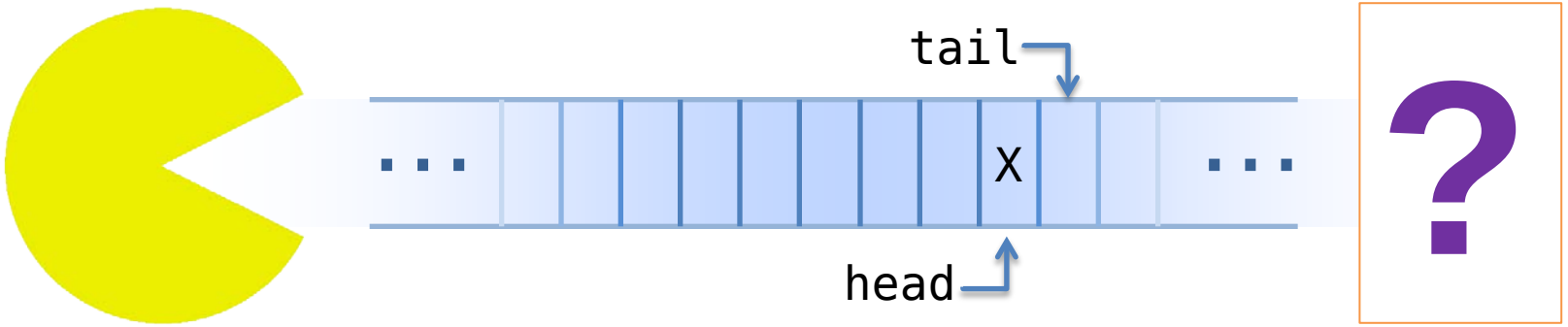
```
void push(int val)
{
    buffer[tail++] = val;
}
```

```
class Queue
{
    int buffer[SZ];
    atomic<size_t> head;
    atomic<size_t> tail;
};
```

X...

A

B



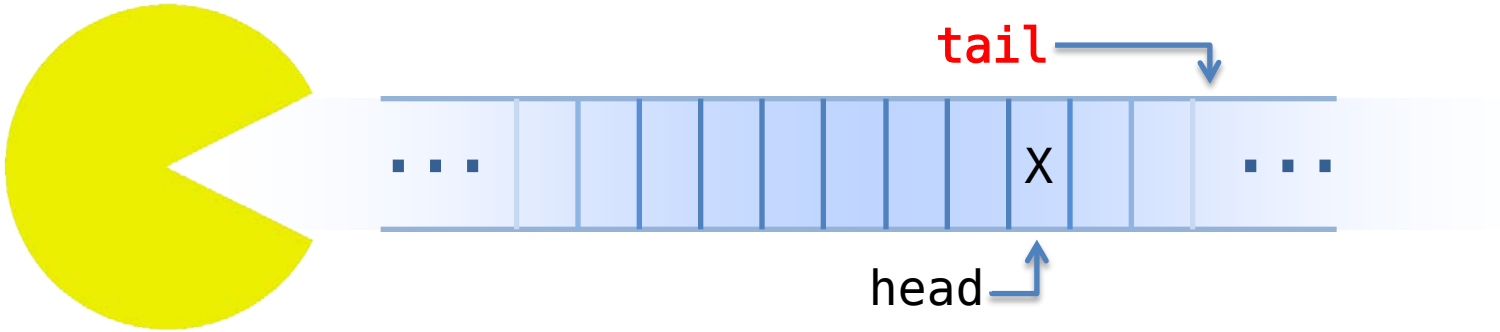
```
void push(int val)
{
    buffer[tail++] = val;
}
```

```
class Queue
{
    int buffer[SZ];
    atomic<size_t> head;
    atomic<size_t> tail;
};
```

X...

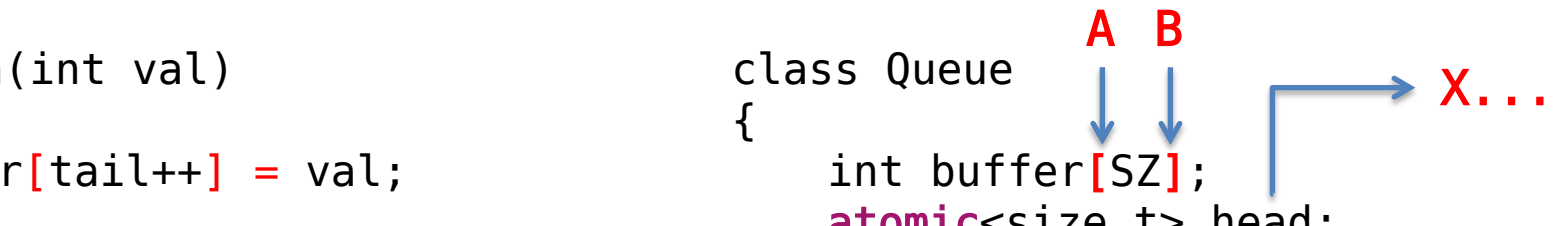
A

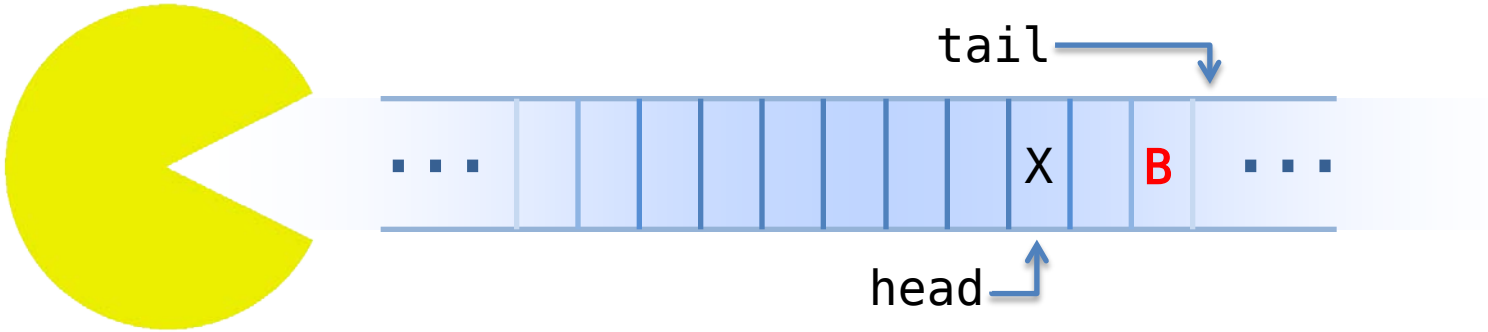
B



```
void push(int val)
{
    buffer[tail++] = val;
}
```

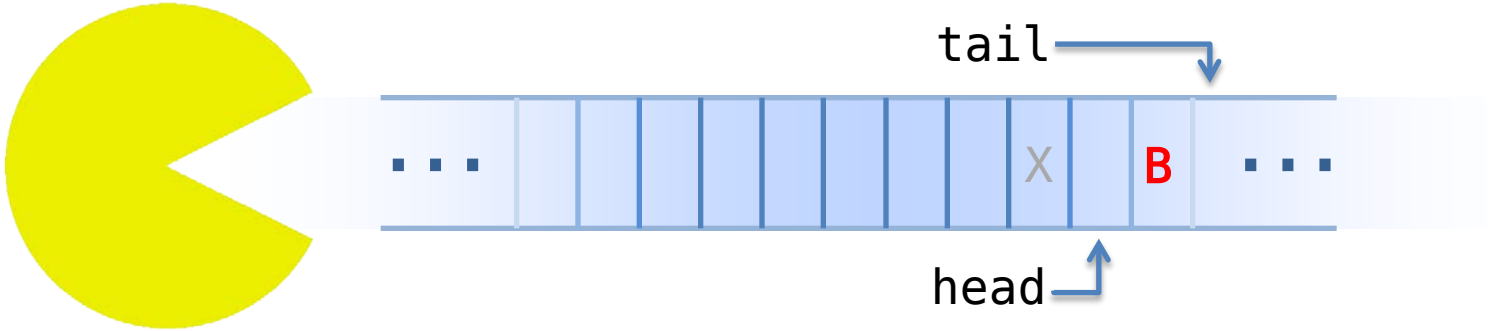
```
class Queue
{
    int buffer[SZ];
    atomic<size_t> head;
    atomic<size_t> tail;
};
```





```
void push(int val)
{
    buffer[tail++] = val;
}
```

```
class Queue
{
    int buffer[SZ];
    atomic<size_t> head;
    atomic<size_t> tail;
};
```

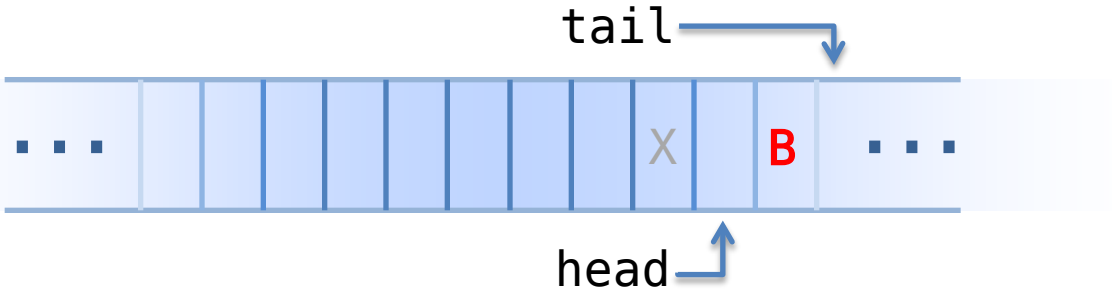


```
void push(int val)
{
    buffer[tail++] = val;
}
```

```
class Queue
{
    int buffer[SZ];
    atomic<size_t> head;
    atomic<size_t> tail;
};
```

A B

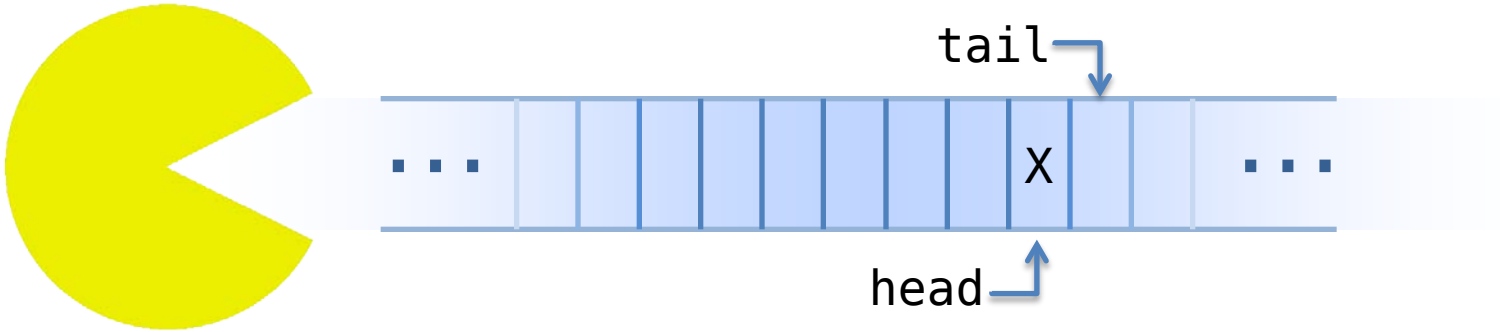
X...



```
void push(int val)
{
    buffer[tail++] = val;
}
```

```
class Queue
{
    int buffer[SZ];
    atomic<size_t> head;
    atomic<size_t> tail;
};
```

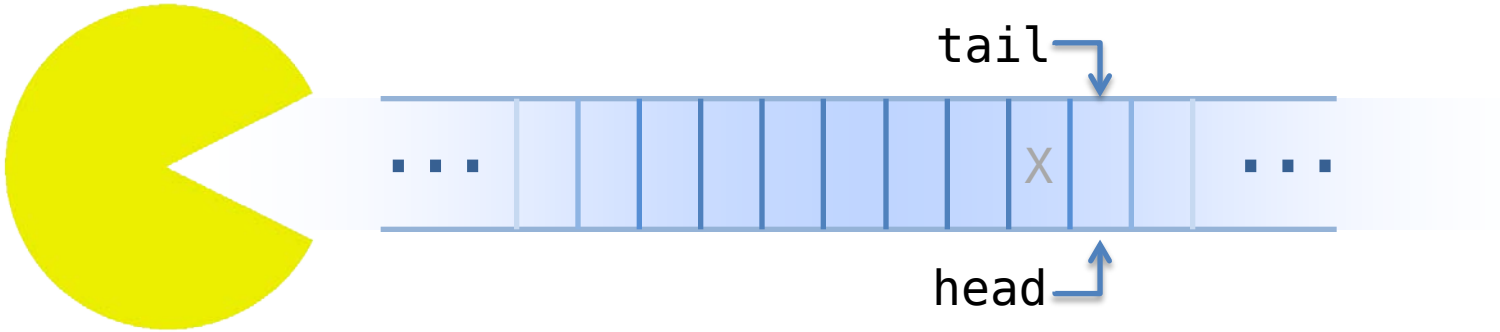
X...



```
void push(int val)
{
    buffer[tail++] = val;
}
```

```
class Queue
{
    int buffer[SZ];
    atomic<size_t> head;
    atomic<size_t> tail;
};
```

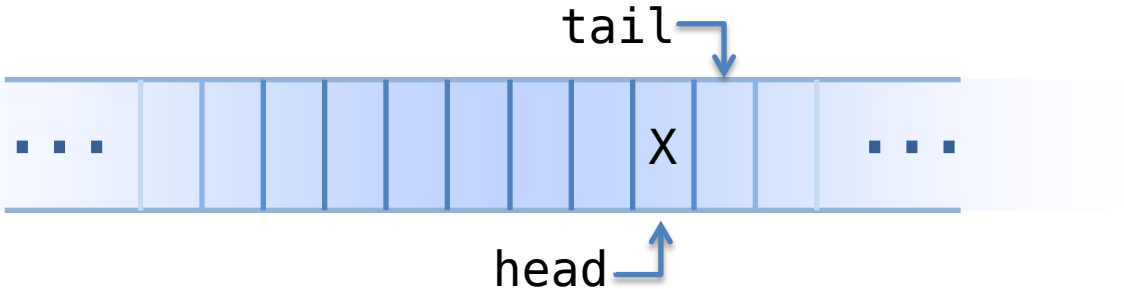
X...



```
void push(int val)
{
    buffer[tail++] = val;
}
```

```
class Queue
{
    int buffer[SZ];
    atomic<size_t> head;
    atomic<size_t> tail;
};
```

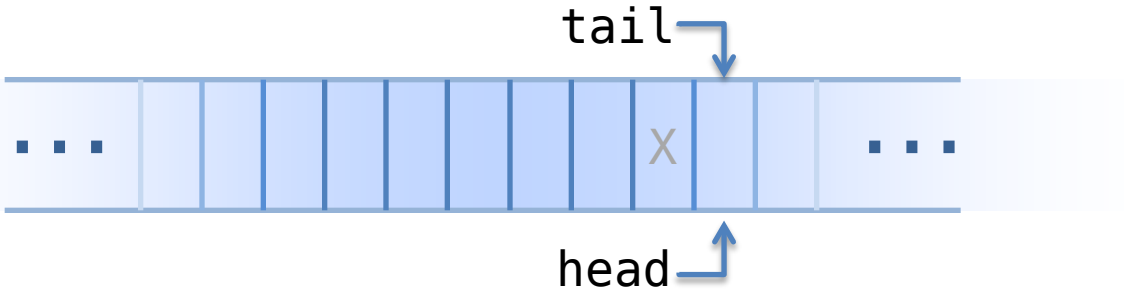
X...



```
void push(int val)
{
    buffer[tail++] = val;
}
```

```
class Queue
{
    int buffer[SZ];
    atomic<size_t> head;
    atomic<size_t> tail;
};
```

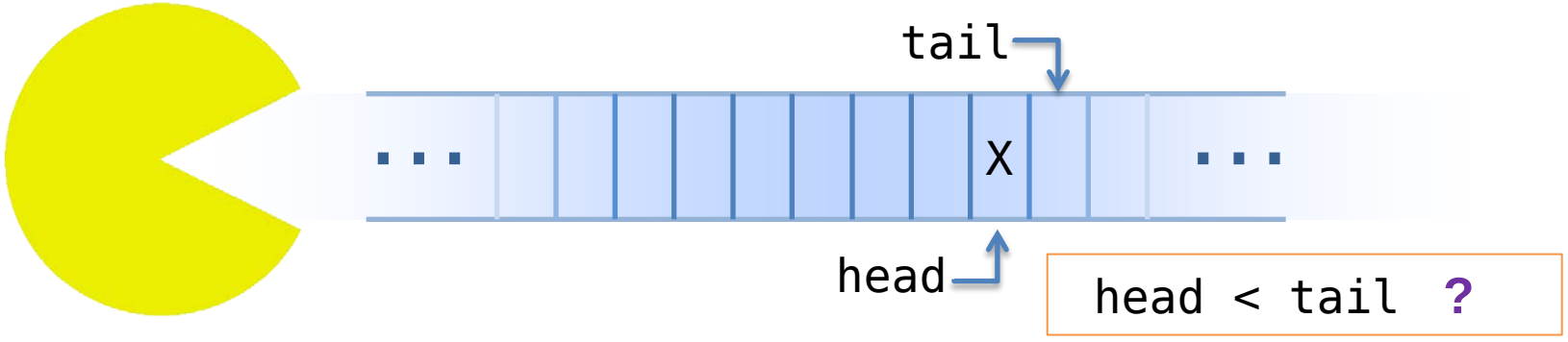
X...



```
void push(int val)
{
    buffer[tail++] = val;
}
```

```
class Queue
{
    int buffer[SZ];
    atomic<size_t> head;
    atomic<size_t> tail;
};
```

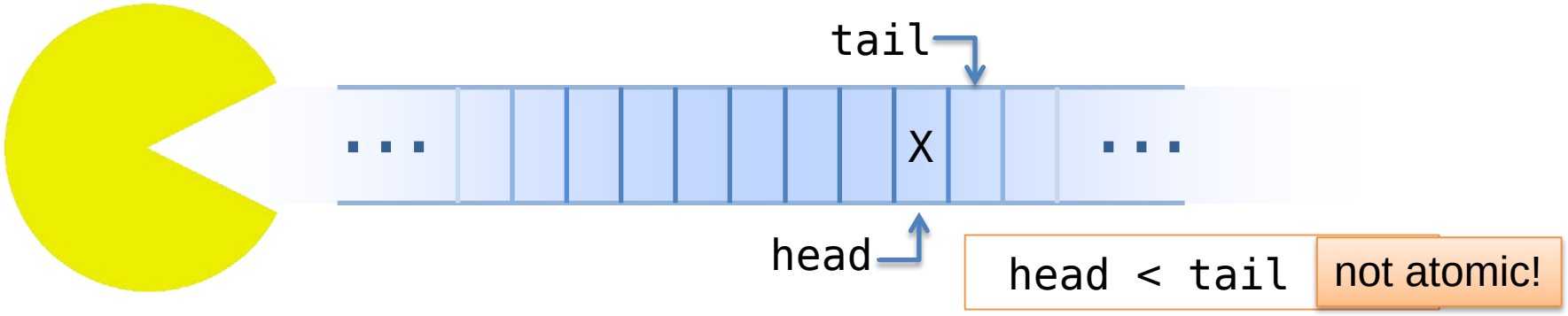
X...



```
void push(int val)
{
    buffer[tail++] = val;
}
```

```
class Queue
{
    int buffer[SZ];
    atomic<size_t> head;
    atomic<size_t> tail;
};
```

X...

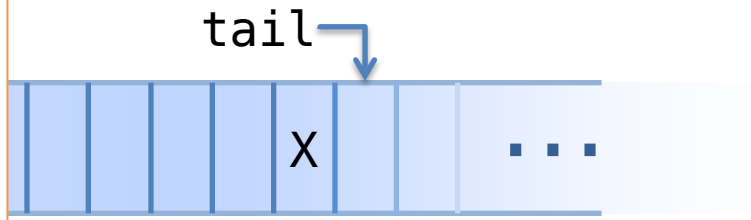


```
void push(int val)
{
    buffer[tail++] = val;
}
```

```
class Queue
{
    int buffer[SZ];
    atomic<size_t> head;
    atomic<size_t> tail;
};
```

X...

```
if (atomic_less(head, tail))
{
    read_head();
};
```



head < tail not atomic!

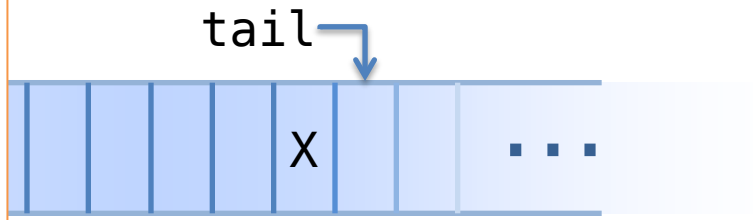
```
void push(int val)
{
    buffer[tail++] = val;
}
```

```
class Queue
{
    int buffer[SZ];
    atomic<size_t> head;
    atomic<size_t> tail;
};
```

X...

```
if (atomic_less(head, tail))
{
    read_head();
};
```

THEN



head < tail

not atomic!

```
void push(int val)
{
    buffer[tail++] = val;
}
```

```
class Queue
{
    int      [SZ]
    atom     si      d;
    atomi    ize_t   ;
};
```

```
if (atomic_less(head, tail
{
    read_he      );
};
```

THEN

X...

head < tail not atomic!

THEN

```
void pu
{
    buf
}
```

```
if (atom
{
    rea
};
```

THEN

X...

atomic!

THEN

is a 4-letter word

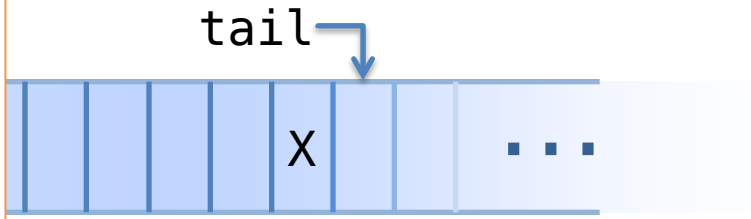
```
void push(int val)
{
    buffer[tail++] = val;
}
```

```
class Queue
{
    int buffer[SZ];
    atomic<size_t> head;
    atomic<size_t> tail;
};
```

X...

```
if (atomic_less(head, tail))
{
    read_head();
};
```

THEN



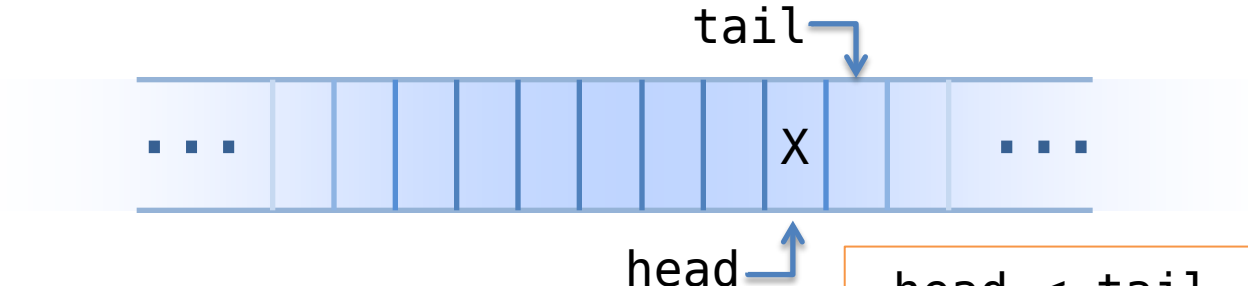
head < tail

not atomic!

```
void push(int val)
{
    buffer[tail++] = val;
}
```

```
class Queue
{
    int buffer[SZ];
    atomic<size_t> head;
    atomic<size_t> tail;
};
```

X...



head < tail ?

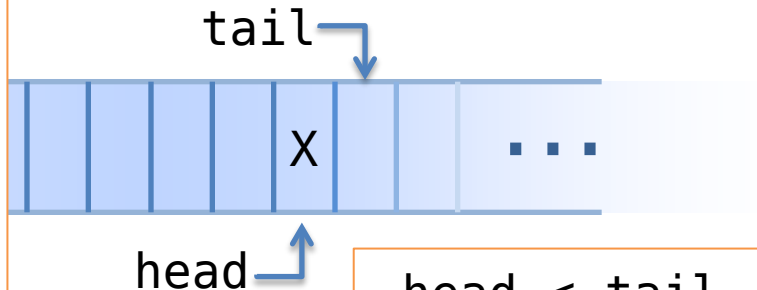
read head **THEN** read tail

```
void push(int val)
{
    buffer[tail++] = val;
}
```

```
class Queue
{
    int buffer[SZ];
    atomic<size_t> head;
    atomic<size_t> tail;
};
```

X...

```
statement1;
THEN
statement2;
THEN
statement3;
...
```



head < tail ?

read head **THEN** read tail

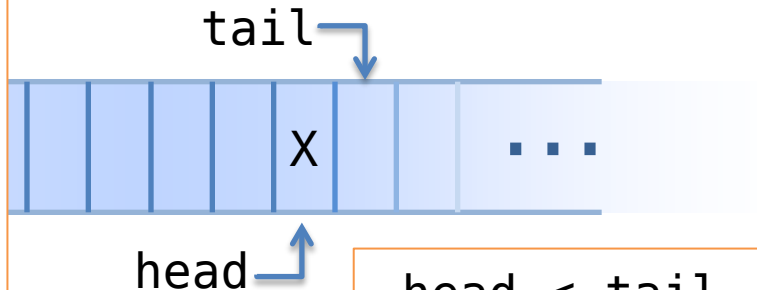
```
void push(int val)
{
    buffer[tail++] = val;
}
```

```
class Queue
{
    int buffer[SZ];
    atomic<size_t> head;
    atomic<size_t> tail;
};
```

X...

```
statement1;
THEN
statement2;
THEN
statement3;
...
```

Local vs Shared



head < tail ?

read head **THEN** read tail

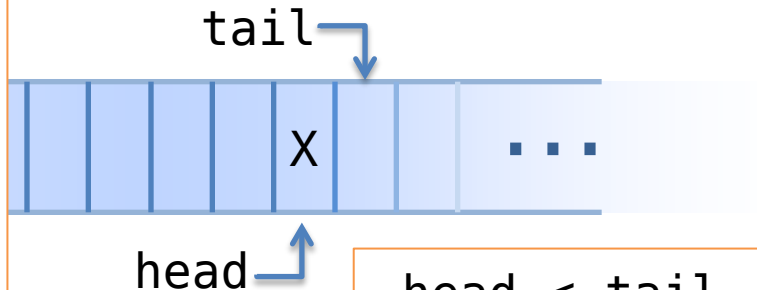
```
void push(int val)
{
    buffer[tail++] = val;
}
```

```
class Queue
{
    int buffer[SZ];
    atomic<size_t> head;
    atomic<size_t> tail;
};
```

X...

```
statement1;
THEN
statement2;
THEN
statement3;
...
```

Local vs Shared



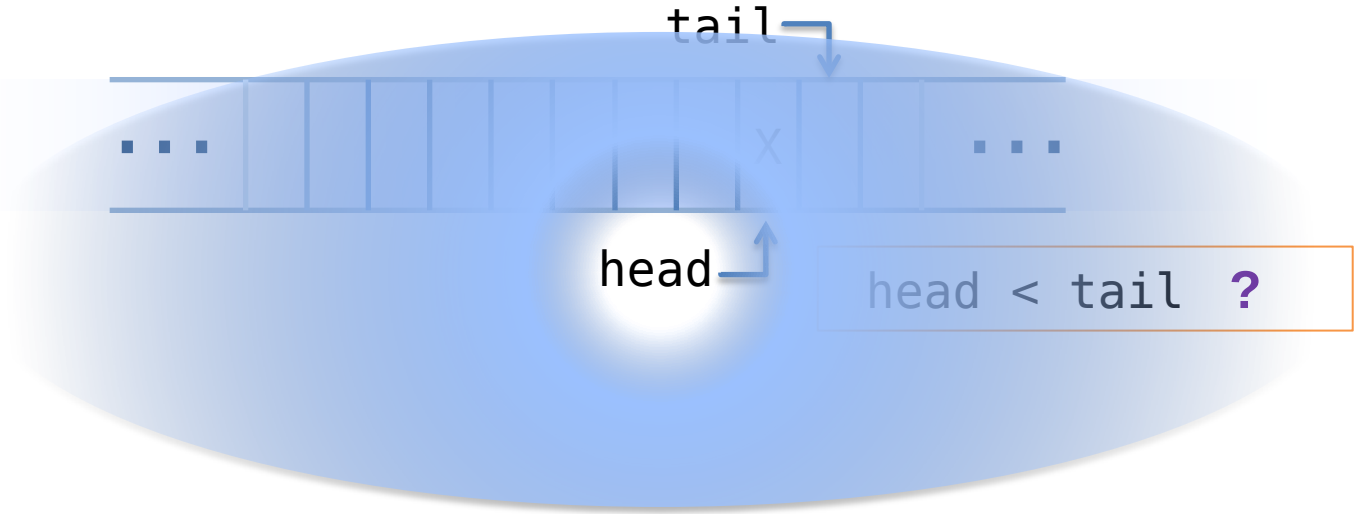
head < tail ?

don't assume **STATE**

```
void push(int val)
{
    buffer[tail++] = val;
}
```

```
class Queue
{
    int buffer[SZ];
    atomic<size_t> head;
    atomic<size_t> tail;
};
```

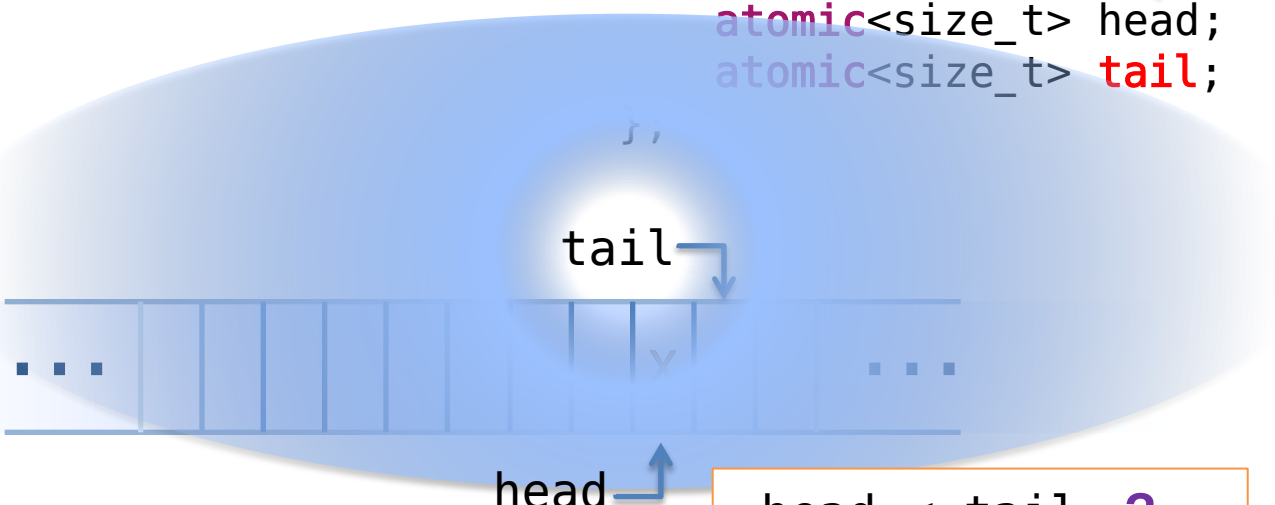
X...



```
void push(int val)
{
    buffer[tail++] = val;
}
```

```
class Queue
{
    int buffer[SZ];
    atomic<size_t> head;
    atomic<size_t> tail;
};
```

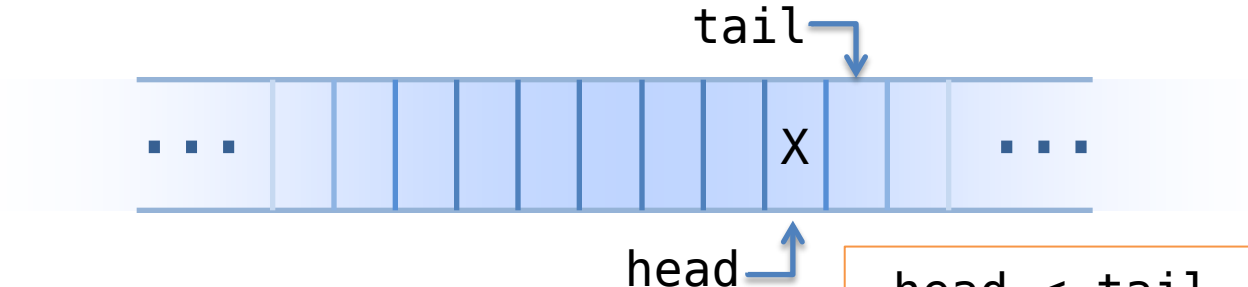
X...



```
void push(int val)
{
    buffer[tail++] = val;
}
```

```
class Queue
{
    int buffer[SZ];
    atomic<size_t> head;
    atomic<size_t> tail;
};
```

X...



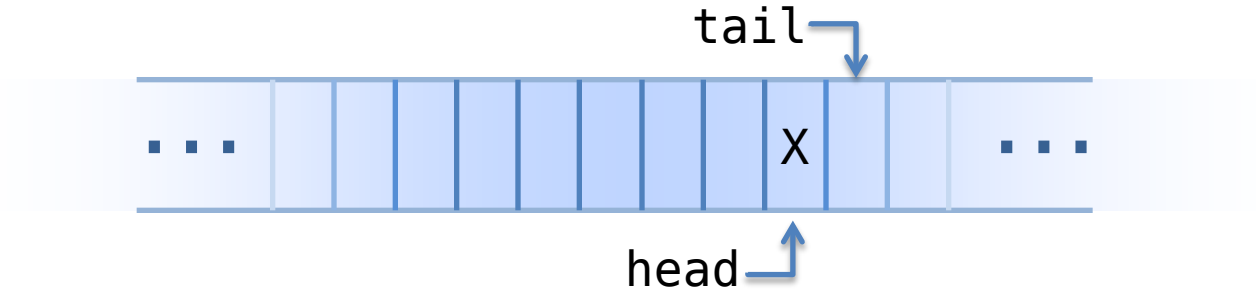
head < tail ?

don't assume **STATE**

```
void push(int val)
{
    buffer[tail++] = val;
}
```

```
class Queue
{
    int buffer[SZ];
    atomic<size_t> head;
    atomic<size_t> tail;
};
```

X...



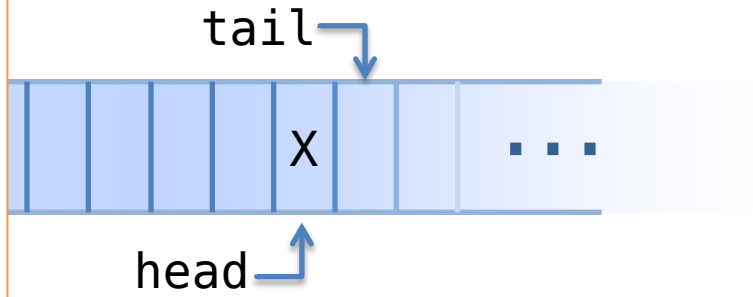
every **STATE** is a good **STATE**

```
void push(int val)
{
    buffer[tail++] = val;
}
```

```
class Queue
{
    int buffer[SZ];
    atomic<size_t> head;
    atomic<size_t> tail;
};
```

X...

```
statement1;
THEN
statement2;
THEN
statement3;
...
```



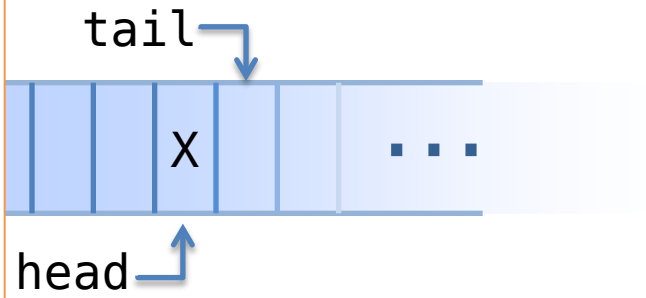
every **STATE** is a good **STATE**

```
void push(int val)
{
    buffer[tail++] = val;
}
```

```
class Queue
{
    int buffer[SZ];
    atomic<size_t> head;
    atomic<size_t> tail;
};
```

X...

```
if (some_state) {
  // some_state is still true(?)
  then_do_stuff();
}
```



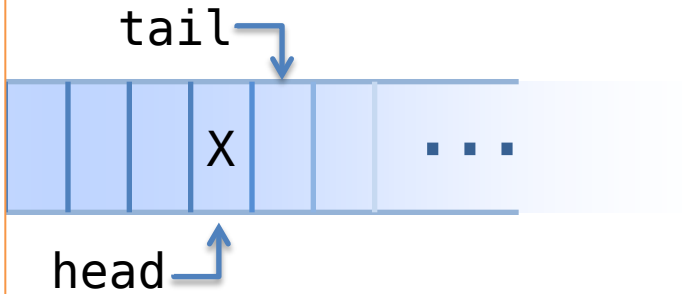
every **STATE** is a good **STATE**

```
void push(int val)
{
    buffer[tail++] = val;
}
```

```
class Queue
{
    int buffer[SZ];
    atomic<size_t> head;
    atomic<size_t> tail;
};
```

X...

```
void member_function {
   break_invariants;
   do_stuff;
   restore_invariants;
}
```

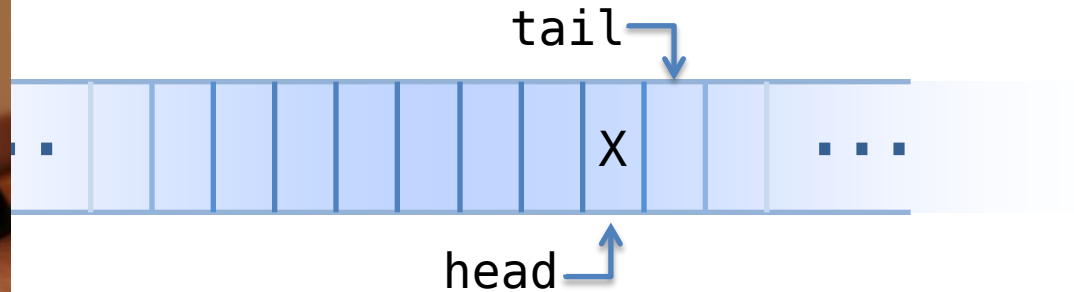


no “temporary suspension” of invariants

```
void push(int val)
{
    buffer[tail++] = val;
}
```

```
class Queue
{
    int buffer[SZ];
    atomic<size_t> head;
    atomic<size_t> tail;
};
```

X...



no “temporary suspension” of invariants

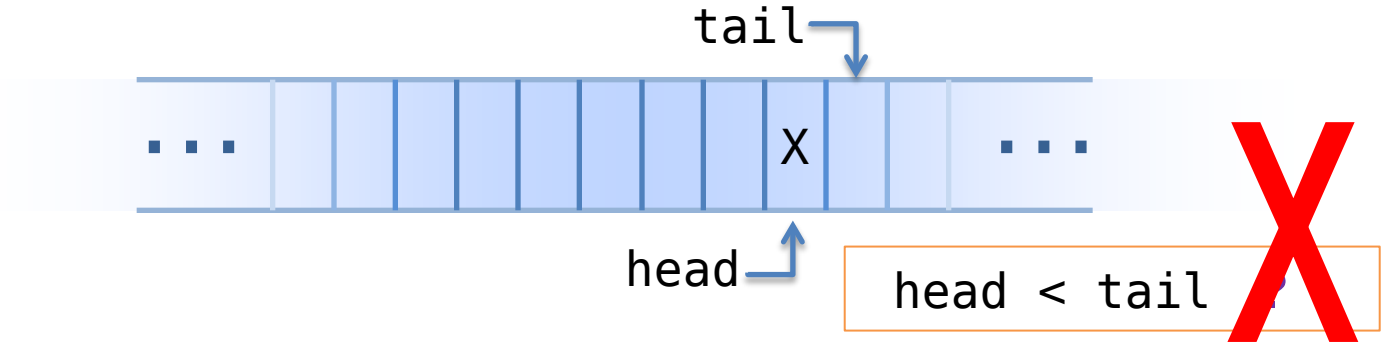
```
void push(int val)
{
    buffer[tail++] = val;
}
```

```
class Queue
{
    int buffer[SZ];
    atomic<size_t> head;
    atomic<size_t> tail;
};
```

X...

A

B



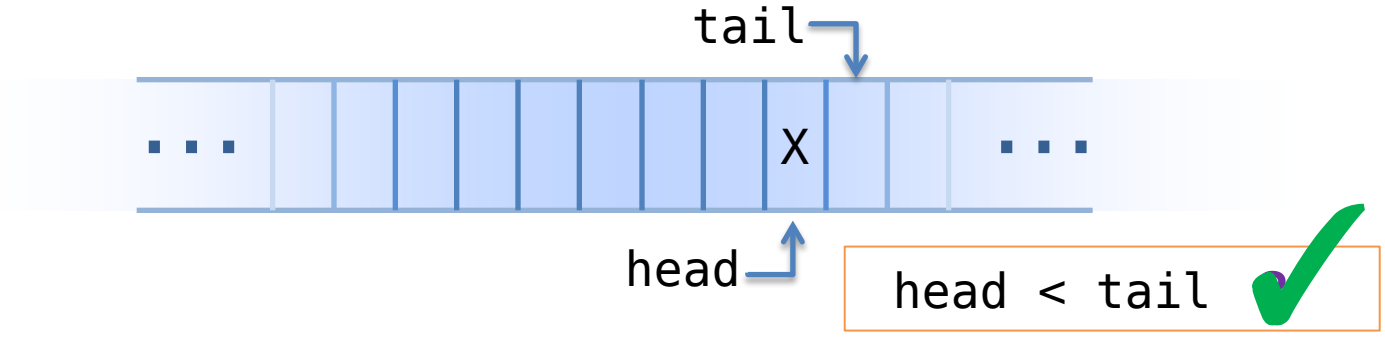
```
void push(int val)
{
    buffer[tail++] = val;
}
```

```
class Queue
{
    int buffer[SZ];
    atomic<size_t> head;
    atomic<size_t> tail;
};
```

X...

A

B



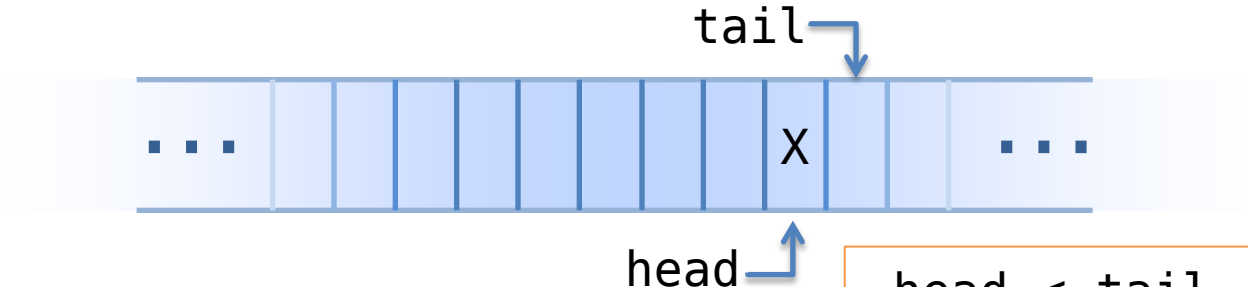
```
void push(int val)
{
    buffer[tail++] = val;
}
```

```
class Queue
{
    int buffer[SZ];
    atomic<size_t> head;
    atomic<size_t> tail;
};
```

X...

A

B



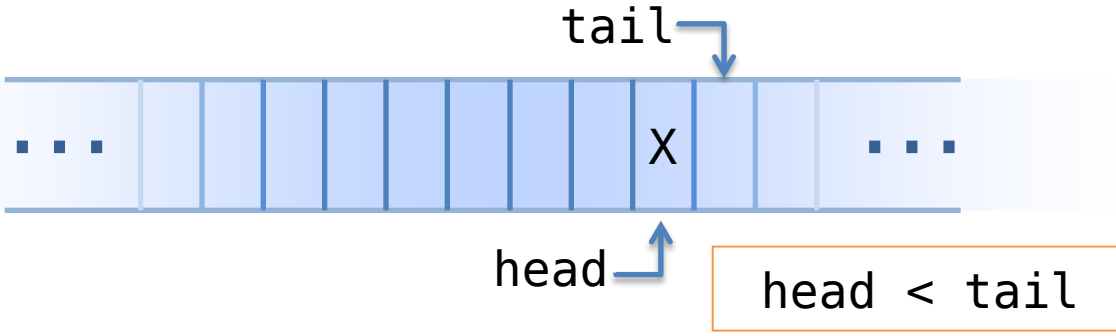
head < tail ✓

ensure tail is always increasing

```
void push(int val)
{
    buffer[tail++] = val;
}
```

```
class Queue
{
    int buffer[SZ];
    atomic<size_t> head;
    atomic<size_t> tail;
};
```

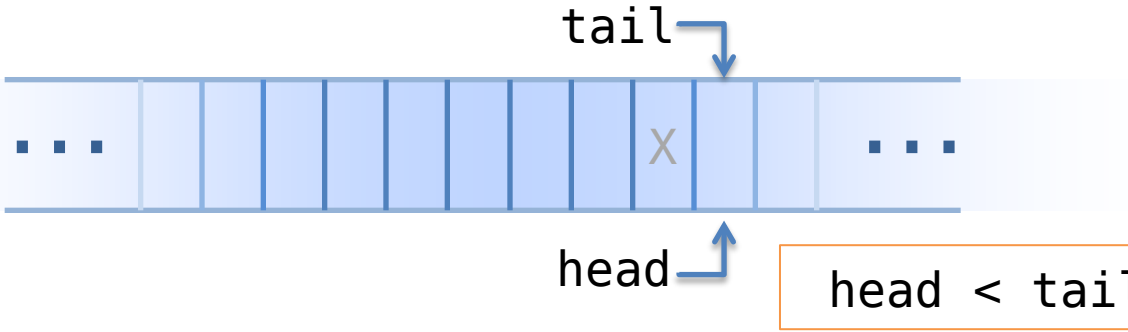
X...



```
void push(int val)
{
    buffer[tail++] = val;
}
```

```
class Queue
{
    int buffer[SZ];
    atomic<size_t> head;
    atomic<size_t> tail;
};
```

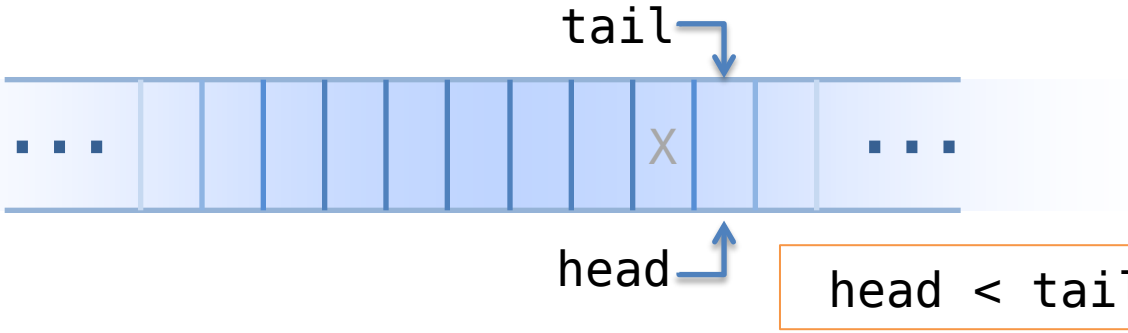
X...



```
void push(int val)
{
    buffer[tail++] = val;
}
```

```
class Queue
{
    int buffer[SZ];
    atomic<size_t> head;
    atomic<size_t> tail;
};
```

X...

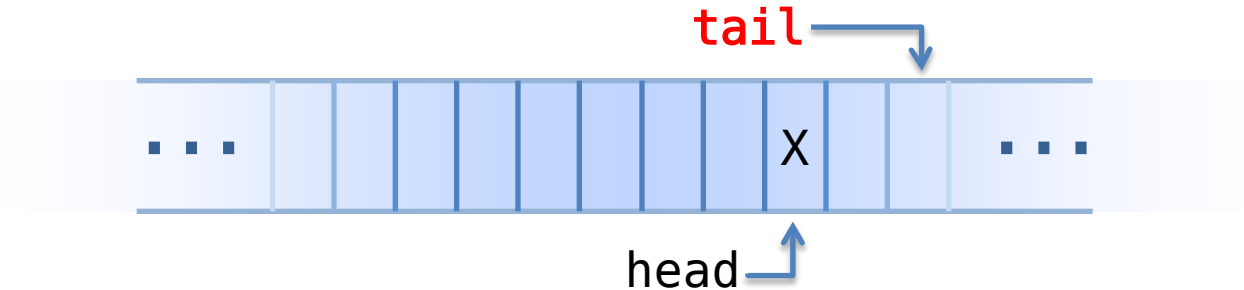


head < tail ?

```
void push(int val)
{
    buffer[tail++] = val;
}
```

```
class Queue
{
    int buffer[SZ];
    atomic<size_t> head;
    atomic<size_t> tail;
};
```

A

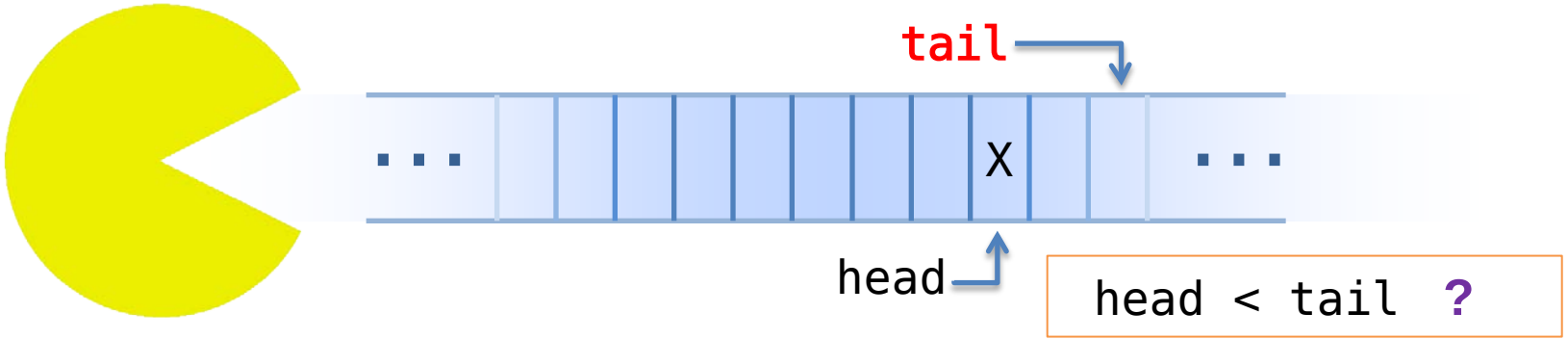


```
void push(int val)
{
    buffer[tail++] = val;
}
```

```
class Queue
{
    int buffer[SZ];
    atomic<size_t> head;
    atomic<size_t> tail;
};
```

X...

A

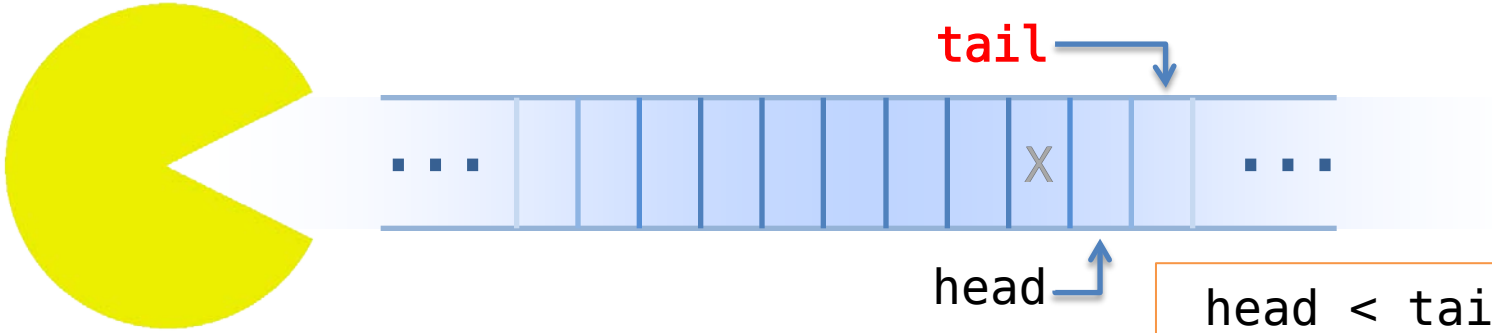


```
void push(int val)
{
    buffer[tail++] = val;
}
```

```
class Queue
{
    int buffer[SZ];
    atomic<size_t> head;
    atomic<size_t> tail;
};
```

X...

A

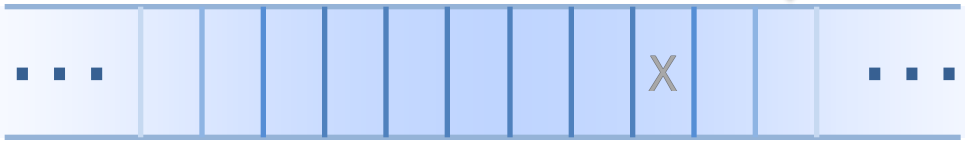


```
void push(int val)
{
    buffer[tail++] = val;
}
```

```
class Queue
{
    int buffer[SZ];
    atomic<size_t> head;
    atomic<size_t> tail;
};
```

X...

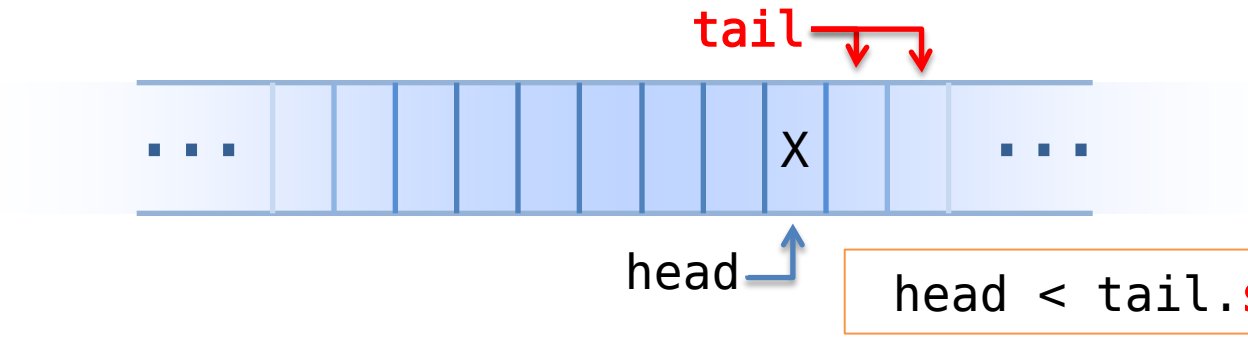
A



head < tail ?

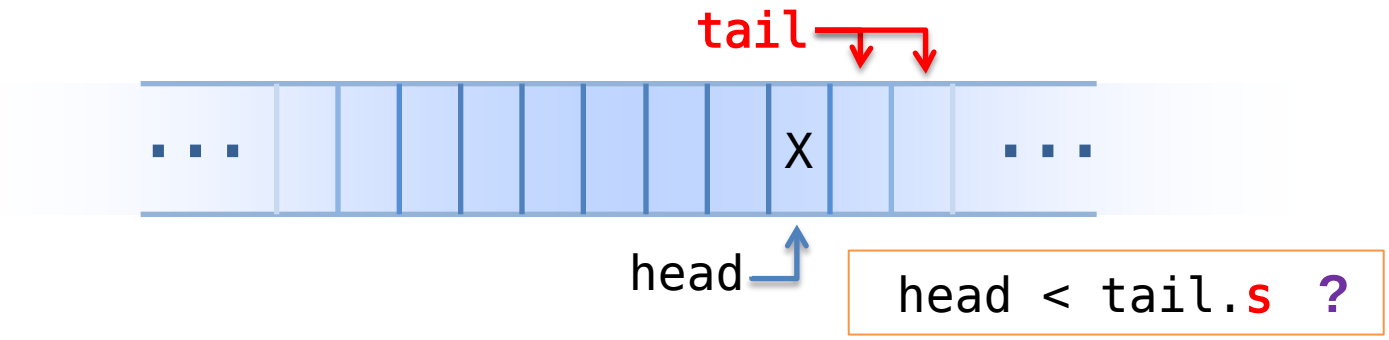
```
void push(int val)
{
    …
}
```

```
class Queue {
    int buffer[SZ];
    atomic<size_t> head;
    struct {    atomic<size_t> s;
                atomic<size_t> e;
    } tail;
};
```



```
void push(int val)
{
    size_t tmp = tail.e++;
    buffer[tmp] = val;
    tail.s++;
}
```

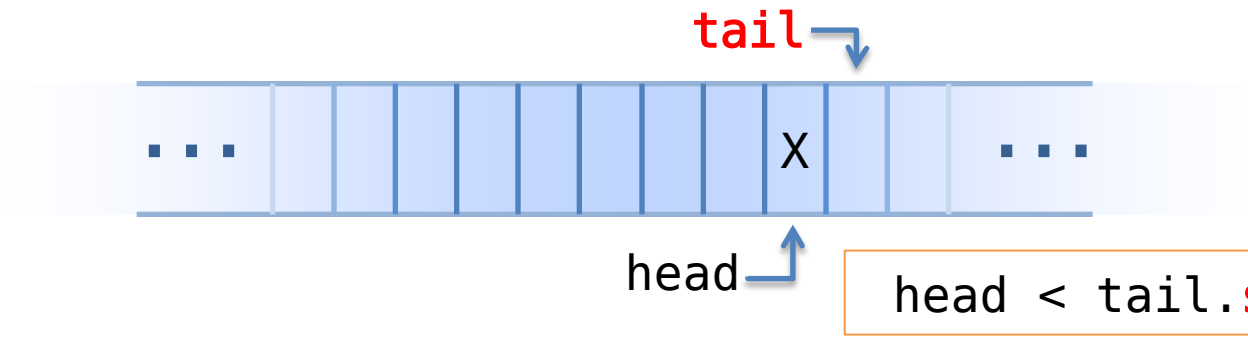
```
class Queue {
    int buffer[SZ];
    atomic<size_t> head;
    struct {    atomic<size_t> s;
                atomic<size_t> e;
    } tail;
};
```



```
void push(int val)
{
    size_t tmp = tail.e++;
    buffer[tmp] = val;
    tail.s++;
}
```

A

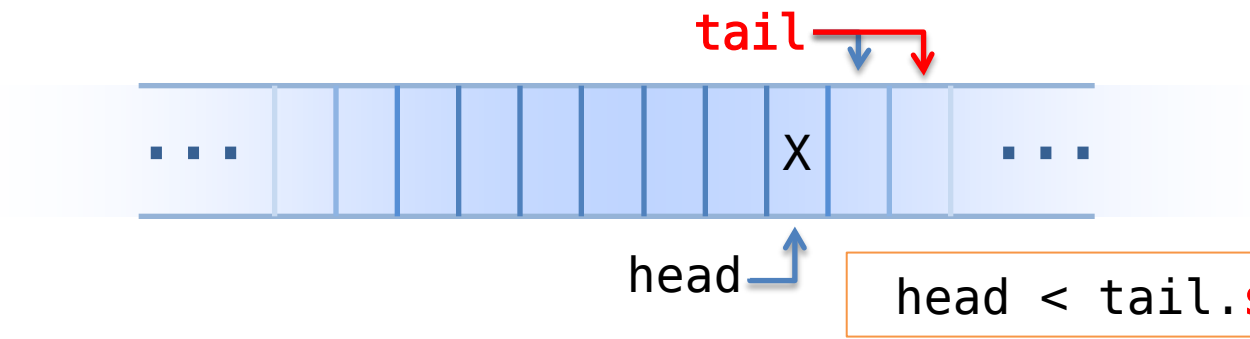
```
class Queue {
    int buffer[SZ];
    atomic<size_t> head;
    struct {    atomic<size_t> s;
                atomic<size_t> e;
    } tail;
};
```



```
void push(int val)
{
    size_t tmp = tail.e++;
    buffer[tmp] = val;
    tail.s++;
}
```

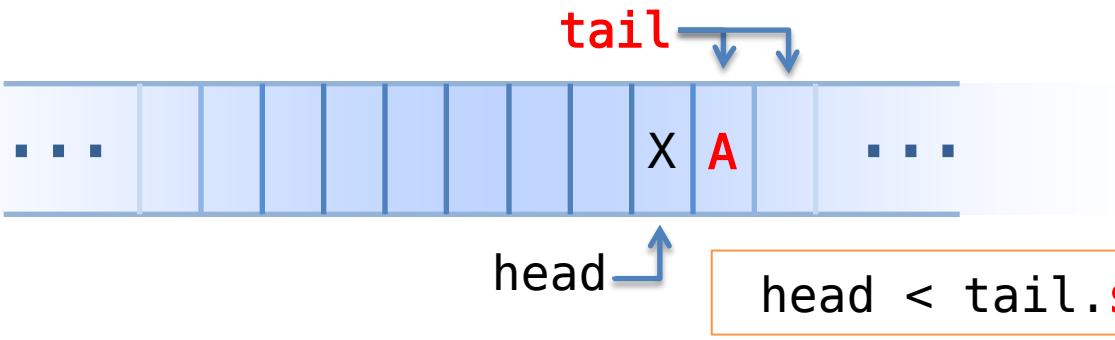
A

```
class Queue {
    int buffer[SZ];
    atomic<size_t> head;
    struct {    atomic<size_t> s;
                atomic<size_t> e;
    } tail;
};
```



```
void push(int val)
{
    size_t tmp = tail.e++;
A   buffer[tmp] = val;
    tail.s++;
}
```

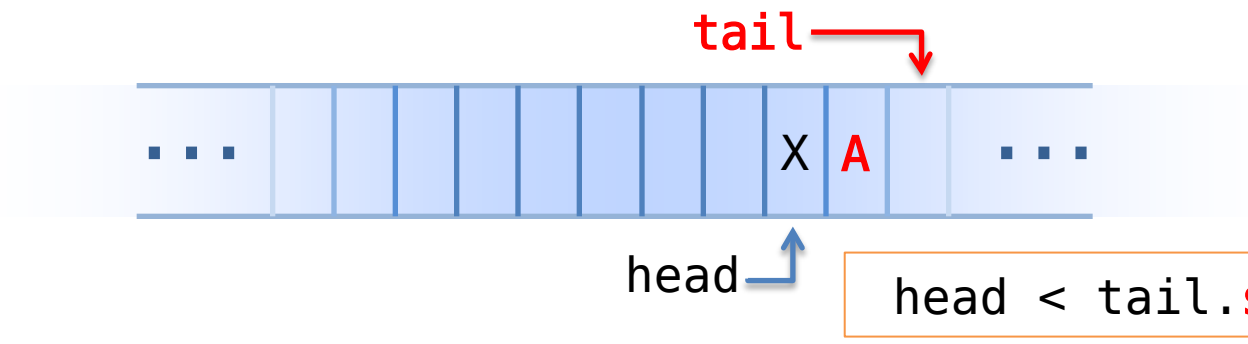
```
class Queue {
    int buffer[SZ];
    atomic<size_t> head;
    struct {    atomic<size_t> s;
                atomic<size_t> e;
    } tail;
};
```



```
void push(int val)
{
    size_t tmp = tail.e++;
    buffer[tmp] = val;
    tail.s++;
}
```

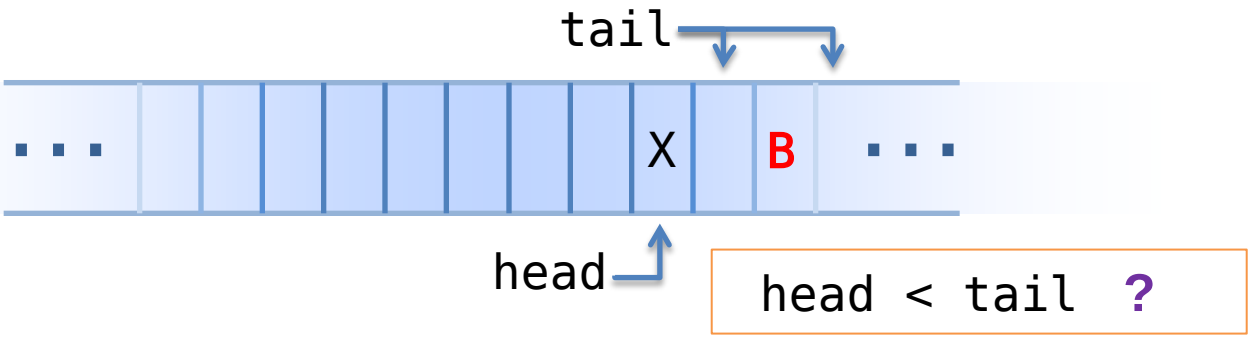
A

```
class Queue {
    int buffer[SZ];
    atomic<size_t> head;
    struct {    atomic<size_t> s;
                atomic<size_t> e;
    } tail;
};
```



```
void push(int val)
{
    size_t tmp = tail.e++;
    buffer[tmp] = val;
    tail.s++;
}
```

```
class Queue {
    int buffer[SZ];
    atomic<size_t> head;
    struct {   atomic<size_t> s;
               atomic<size_t> e;
    } tail;
};
```

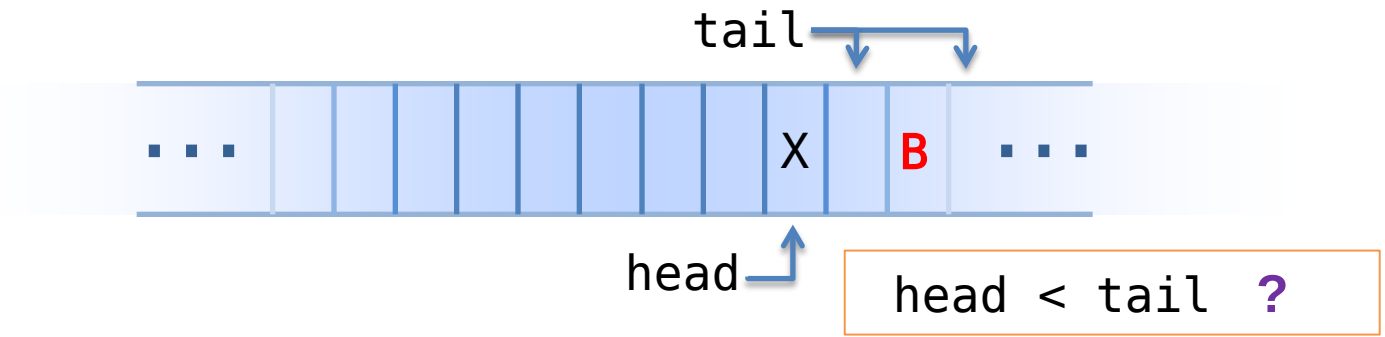


```
void push(int val)
{
    size_t tmp = tail.e++;
    buffer[tmp] = val;
    tail.s++;
}
```

A

B

```
class Queue {
    int buffer[SZ];
    atomic<size_t> head;
    struct {    atomic<size_t> s;
                atomic<size_t> e;
    } tail;
};
```

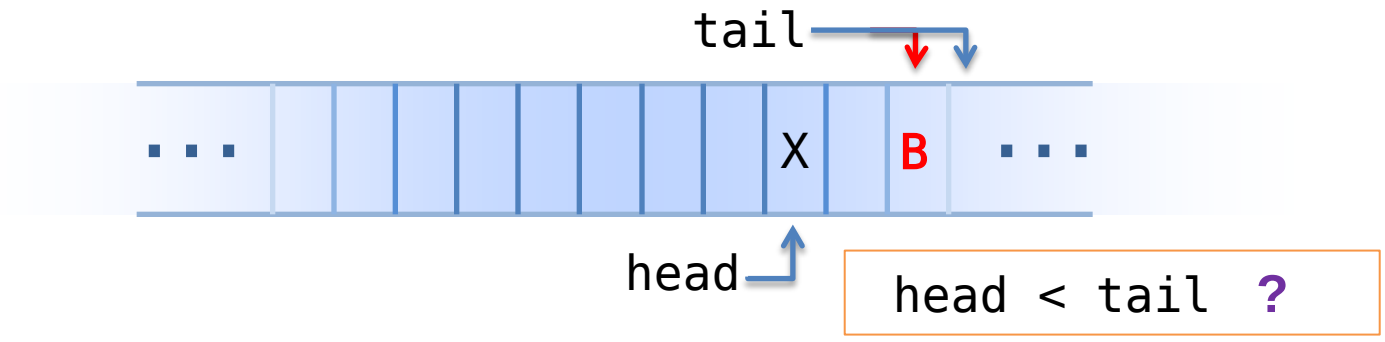


```
void push(int val)
{
    size_t tmp = tail.e++;
    buffer[tmp] = val;
    tail.s++;
}
```

A

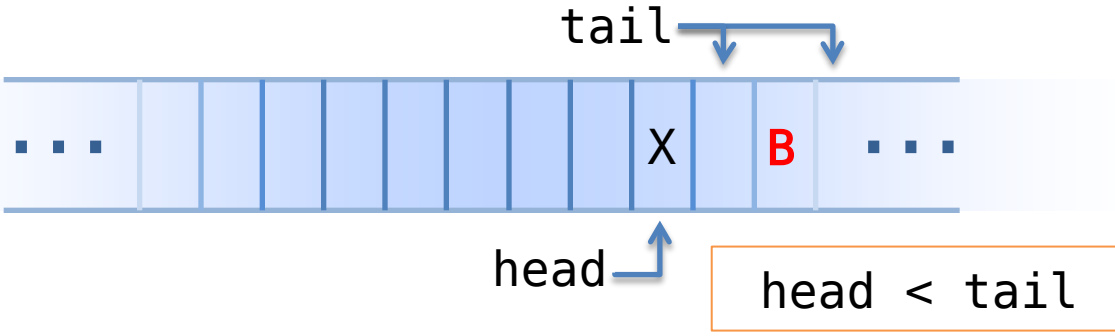
B

```
class Queue {
    int buffer[SZ];
    atomic<size_t> head;
    struct {    atomic<size_t> s;
                atomic<size_t> e;
    } tail;
};
```



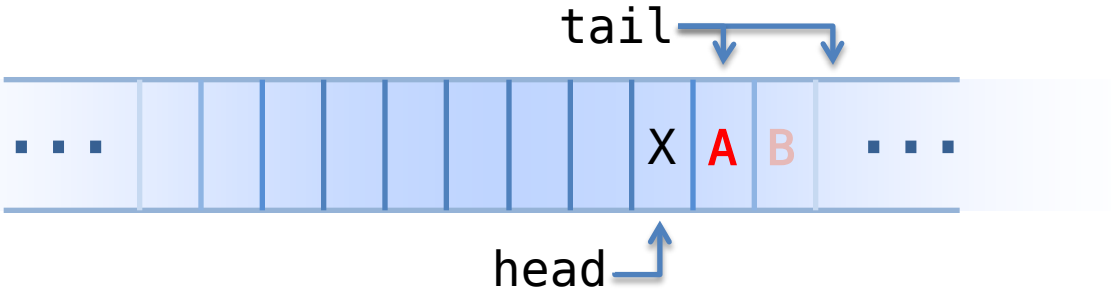
```
void push(int val)
{
    size_t tmp = tail.e++;
    buffer[tmp] = val;
    if(tmp == tail.s) {
        tail.s = …
    }
}
```

```
class Queue {
    int buffer[SZ];
    atomic<size_t> head;
    struct {    atomic<size_t> s;
                atomic<size_t> e;
    } tail;
};
```



```
void push(int val)
{
    size_t tmp = tail.e++;
    buffer[tmp] = val;
    if(tmp == tail.s) {
        tail.s = …
    }
}
```

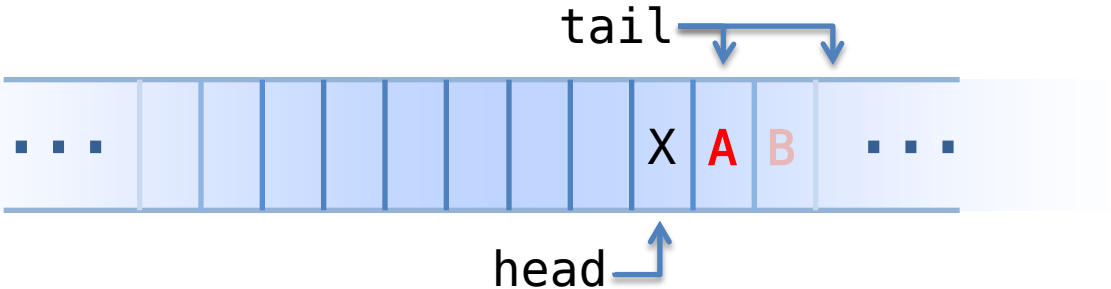
```
class Queue {
    int buffer[SZ];
    atomic<size_t> head;
    struct {    atomic<size_t> s;
                atomic<size_t> e;
    } tail;
};
```



```
void push(int val)
{
    size_t tmp = tail.e++;
    buffer[tmp] = val;
    if(tmp == tail.s) {
        tail.s = …
    }
}
```

THEN

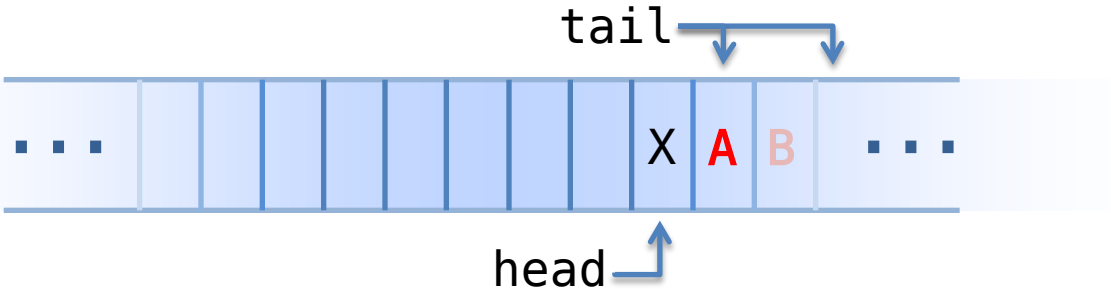
```
class Queue {
    int buffer[SZ];
    atomic<size_t> head;
    struct {    atomic<size_t> s;
                atomic<size_t> e;
    } tail;
};
```



```
void push(int val)
{
    size_t tmp = tail.e++;
    buffer[tmp] = val;
    if(tmp == tail.s) {
        tail.s = …
    }
}
```

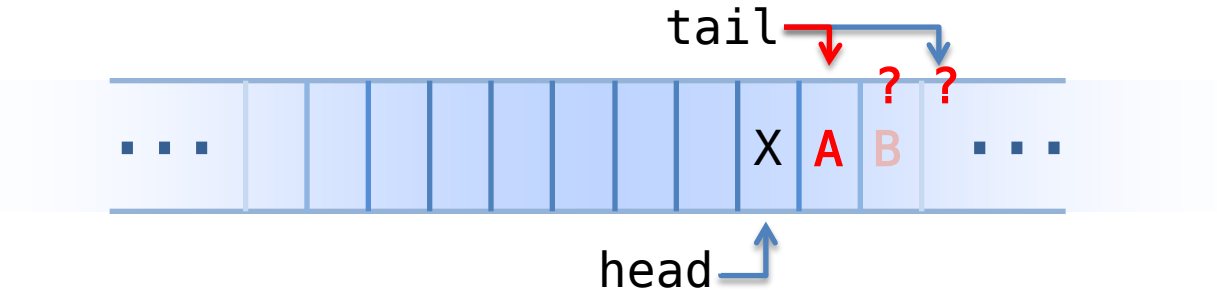
THEN

```
class Queue {
    int buffer[SZ];
    atomic<size_t> head;
    struct {    atomic<size_t> s;
                atomic<size_t> e;
    } tail;
};
```



```
void push(int val)
{
    size_t tmp = tail.e++;
    buffer[tmp] = val;
    if(tmp == tail.s) {
        tail.s = ???
    }
}
```

```
class Queue {
    int buffer[SZ];
    atomic<size_t> head;
    struct {    atomic<size_t> s;
                atomic<size_t> e;
    } tail;
};
```



```
void push(int val)
{
    size_t tmp = tail.e++;
    buffer[tmp] = val;
    if(tmp == tail.s) {
        tail.s = ???
    }
}
```

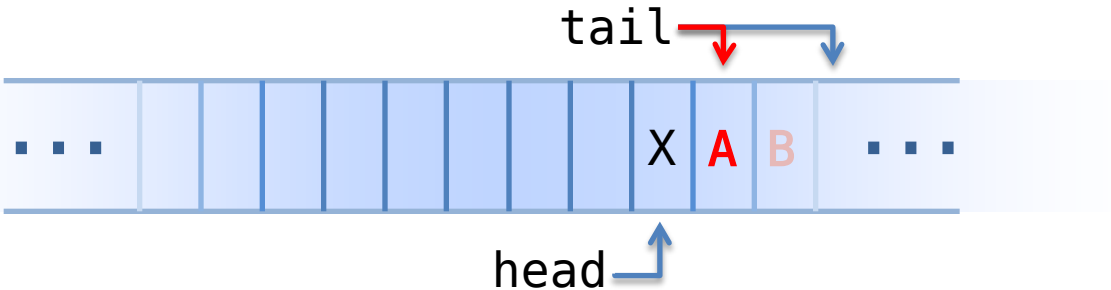
```
class Queue {
    int buffer[SZ];
    atomic<size_t> head;
    struct {    atomic<size_t> s;
                atomic<size_t> e;
    } tail;
};
```

Compromise…

Queue of int -> Queue of int != 0

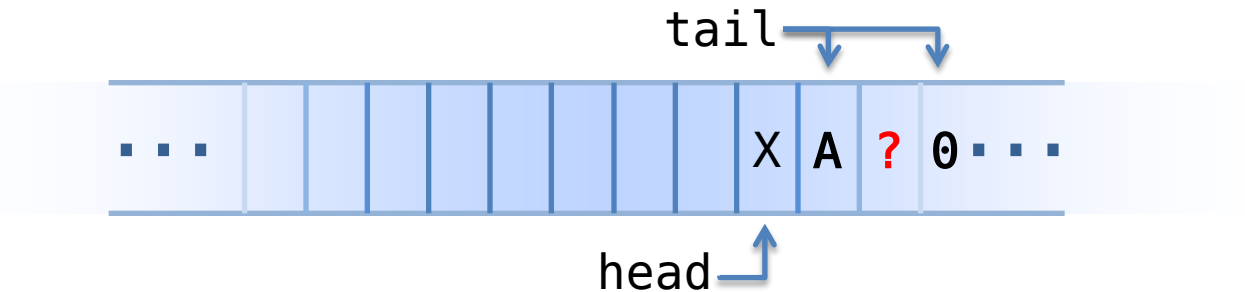
```
void push(int val)
{
    size_t tmp = tail.e++;
    buffer[tmp] = val;
    if(tmp == tail.s) {
        tail.s = ???
    }
}
```

```
class Queue {
    int buffer[SZ];
    atomic<size_t> head;
    struct {    atomic<size_t> s;
                atomic<size_t> e;
    } tail;
};
```



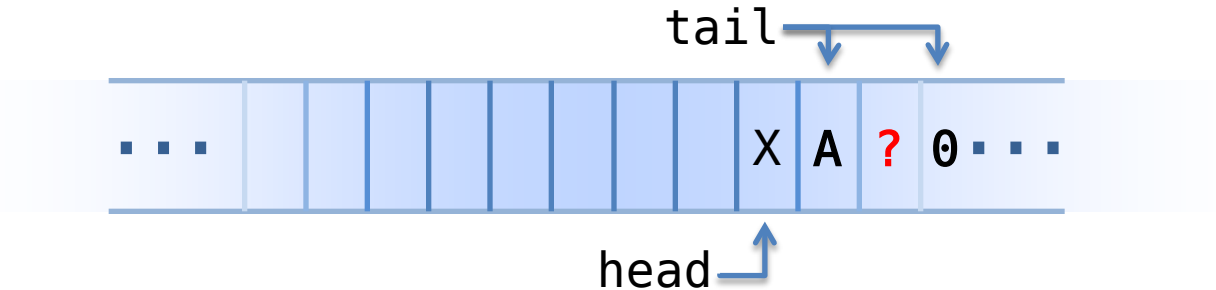
```
void push(int val) {
    size_t tmp = tail.e++;
    buffer[tmp] = val;
    if(tmp == tail.s) {
        do
            tail.s++;
        while (buffer[tail.s]);
    }
}
```

```
class Queue {
    int buffer[SZ];
    atomic<size_t> head;
    struct {    atomic<size_t> s;
                atomic<size_t> e;
    } tail;
};
```



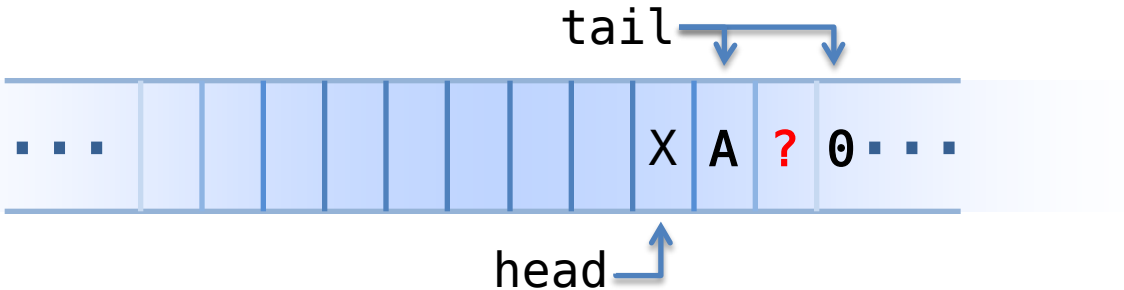
```
void push(int val) {
    size_t tmp = tail.e++;
B   buffer[tmp] = val;
    if(tmp == tail.s) {
        do
            tail.s++;
A       while (buffer[tail.s]);
    }
}
```

```
class Queue {
    int buffer[SZ];
    atomic<size_t> head;
    struct {    atomic<size_t> s;
                atomic<size_t> e;
    } tail;
};
```



```
void push(int val) {
    size_t tmp = tail.e++;
B   buffer[tmp] = val;
    if(tmp == tail.s) {
        do
            tail.s++;
A       while (buffer[tail.s]);
    }
}
```

```
class Queue {
    atomic<int> buffer[SZ];
    atomic<size_t> head;
    struct {    atomic<size_t> s;
                atomic<size_t> e;
    } tail;
};
```

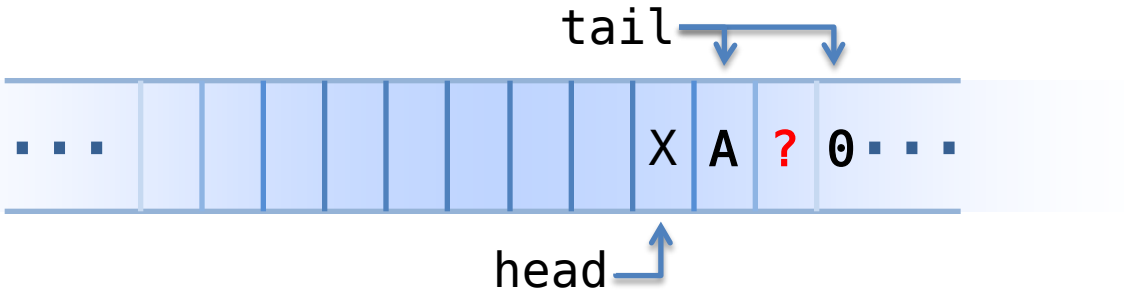


```
void push(int val) {
    size_t tmp = tail.e++;
    buffer[tmp] = val;
    if(tmp == tail.s) {
        do
            tail.s++;
        while (buffer[tail.s]);
    }
}
```

THEN

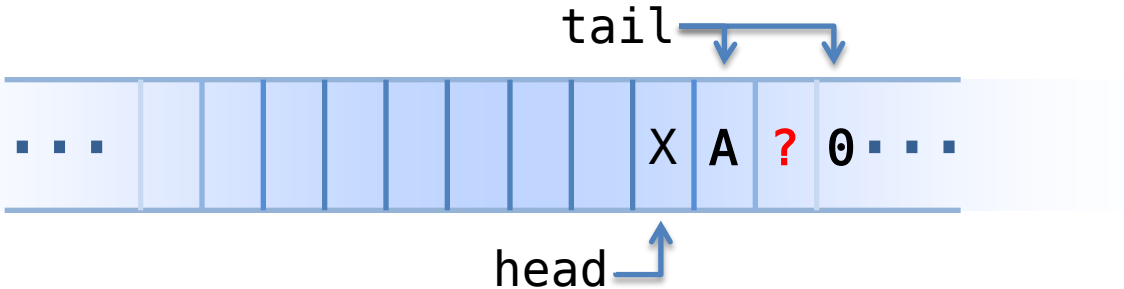
THEN

```
class Queue {
    atomic<int> buffer[SZ];
    atomic<size_t> head;
    struct {    atomic<size_t> s;
                atomic<size_t> e;
    } tail;
};
```



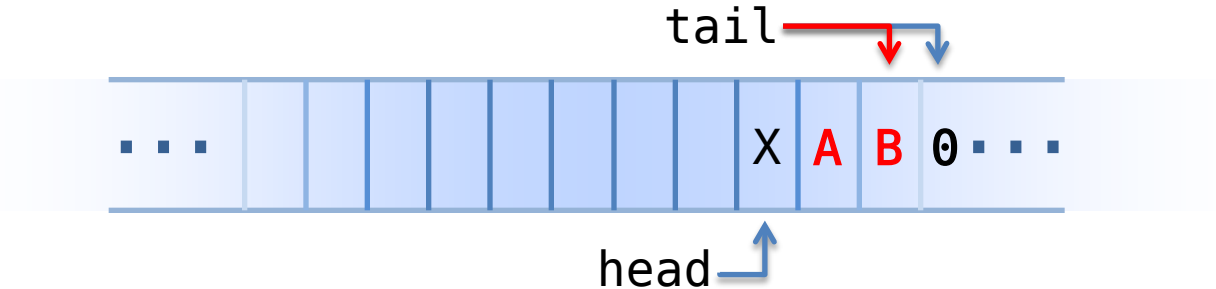
```
void push(int val) {
    size_t tmp = tail.e++;
    buffer[tmp] = val;
    if(tmp == tail.s) {
        do
A ------>   tail.s++;
        while (buffer[tail.s]);
    }
}
```

```
class Queue {
    atomic<int> buffer[SZ];
    atomic<size_t> head;
    struct {    atomic<size_t> s;
                atomic<size_t> e;
    } tail;
};
```



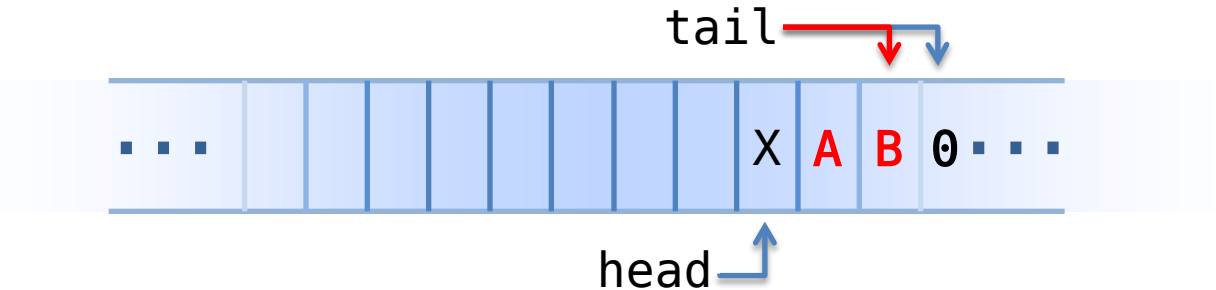
```
void push(int val) {
    size_t tmp = tail.e++;
    buffer[tmp] = val;
B   if(tmp == tail.s) {
        do
A           tail.s++;
        while (buffer[tail.s]);
    }
}
```

```
class Queue {
    atomic<int> buffer[SZ];
    atomic<size_t> head;
    struct {    atomic<size_t> s;
                atomic<size_t> e;
    } tail;
};
```



```
void push(int val) {
    size_t tmp = tail.e++;
    buffer[tmp] = val;
    if(tmp == tail.s) {
B       do
A           tail.s++;
        while (buffer[tail.s]);
    }
}
```

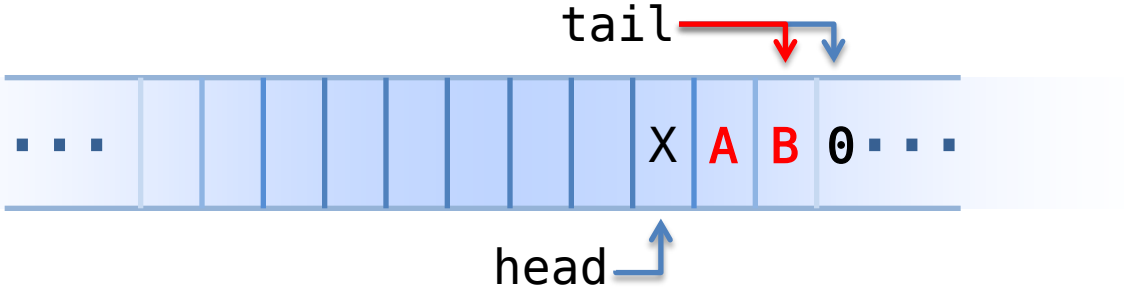
```
class Queue {
    atomic<int> buffer[SZ];
    atomic<size_t> head;
    struct {    atomic<size_t> s;
                atomic<size_t> e;
    } tail;
};
```



```
void push(int val) {
    size_t tmp = tail.e++;
    buffer[tmp] = val;
    if(tmp == tail.s) {
        do
            CAS(tail.s,tmp,tmp+1);
        while (buffer[++tmp]);
    }
}
```

B
A

```
class Queue {
    atomic<int> buffer[SZ];
    atomic<size_t> head;
    struct {    atomic<size_t> s;
                atomic<size_t> e;
    } tail;
};
```

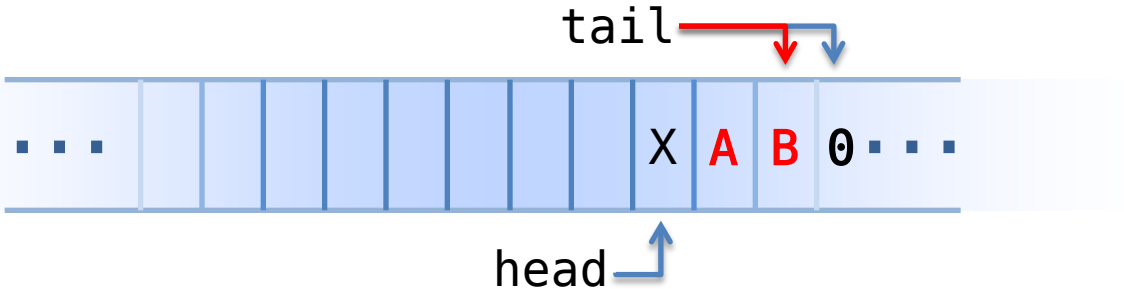


```
void push(int val) {
    size_t tmp = tail.e++;
    buffer[tmp] = val;
    if(tmp == tail.s) {
        do
            CAS(tail.s,tmp,tmp+1);
        while (buffer[++tmp]);
    }
}
```

same

B
A

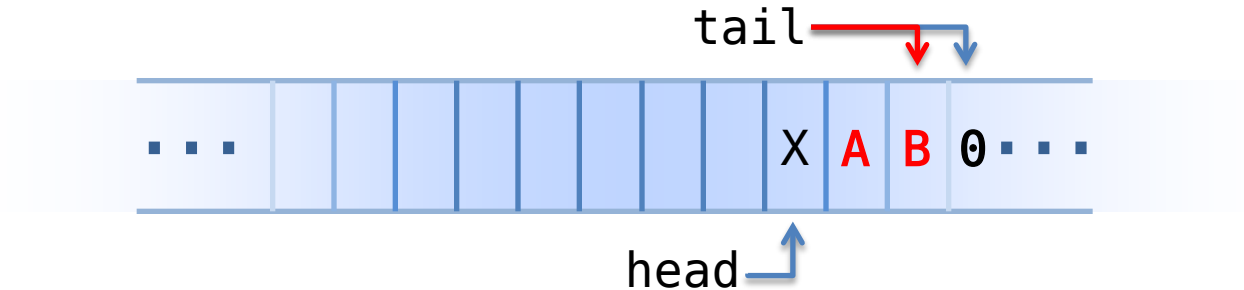
```
class Queue {
    atomic<int> buffer[SZ];
    atomic<size_t> head;
    struct {    atomic<size_t> s;
                atomic<size_t> e;
    } tail;
};
```



```
void push(int val) {
    size_t tmp = tail.e++;
    buffer[tmp] = val;
    do
        CAS(tail.s,tmp,tmp+1);
    while (buffer[++tmp]);
}
```

B
A

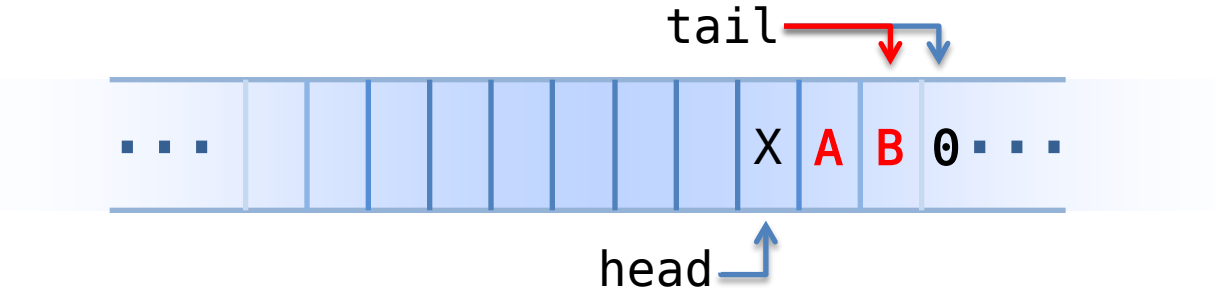
```
class Queue {
    atomic<int> buffer[SZ];
    atomic<size_t> head;
    struct {    atomic<size_t> s;
                atomic<size_t> e;
    } tail;
};
```



```
void push(int val) {
    size_t tmp = tail.e++;
    buffer[tmp] = val;
    bool r;
    do
        r = CAS(tail.s,tmp,tmp+1);
    while (r && buffer[++tmp]);
}
```

B
A

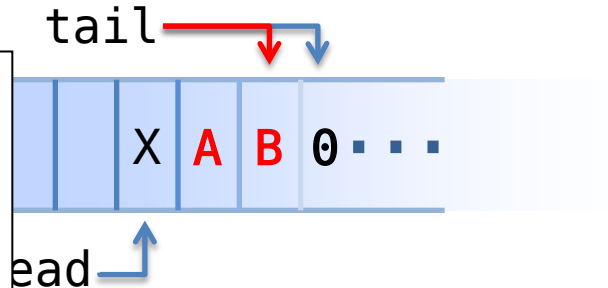
```
class Queue {
    atomic<int> buffer[SZ];
    atomic<size_t> head;
    struct {    atomic<size_t> s;
                atomic<size_t> e;
    } tail;
};
```



```
void push(int val) {
    size_t tmp = tail.e++;
    buffer[tmp] = val;
    bool r;
    do
        r = CAS(tail.s,tmp,tmp+1);
    while (r && buffer[++tmp]);
}
```

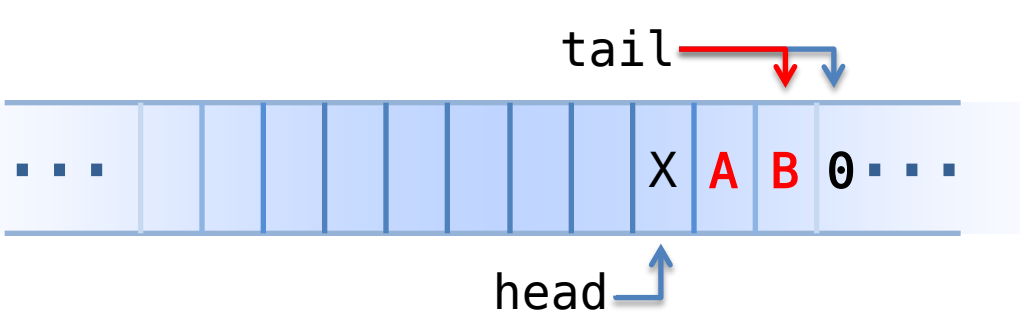
```
class Queue {
    atomic<int> buffer[SZ];
    atomic<size_t> head;
    struct {    atomic<size_t> s;
                atomic<size_t> e;
    } tail;
};
```

```
void push(int val) {
    size_t tmp = tail.e++;
    buffer[tmp] = val;
    do
        CAS(tail.s,tmp,tmp+1);
    while (buffer[++tmp]);
}
```



```
void push(int val) {
    size_t tmp = tail.e++;
    buffer[tmp] = val;
    do
        CAS(tail.s,tmp,tmp+1);
    while (buffer[++tmp]);
}
```

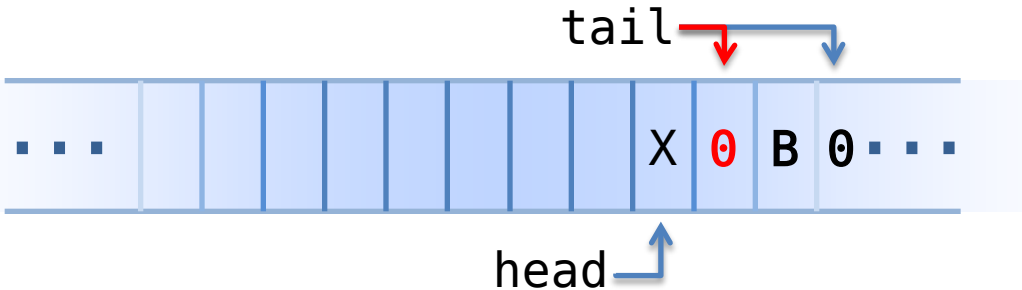
```
class Queue {
    atomic<int> buffer[SZ];
    atomic<size_t> head;
    struct {    atomic<size_t> s;
                atomic<size_t> e;
    } tail;
};
```



```
void push(int val) {
    size_t tmp = tail.e++;
    buffer[tmp] = val;
    do
        CAS(tail.s,tmp,tmp+1);
    while (buffer[++tmp]);
}
```

A → PAUSE

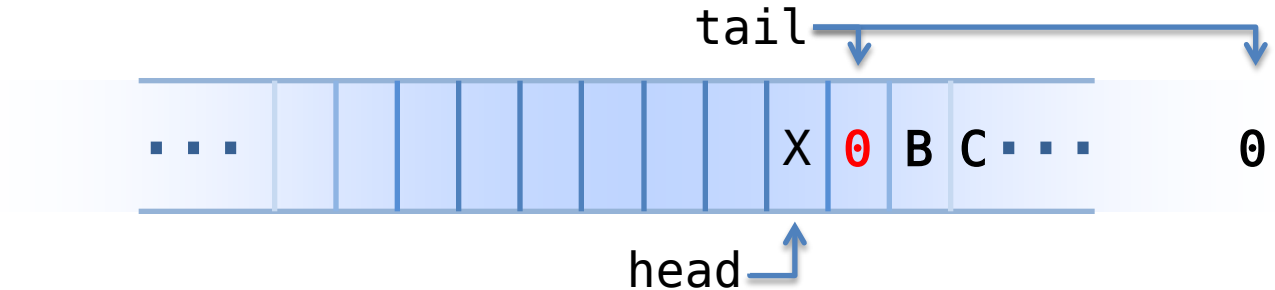
```
class Queue {
    atomic<int> buffer[SZ];
    atomic<size_t> head;
    struct {    atomic<size_t> s;
                atomic<size_t> e;
    } tail;
};
```



```
void push(int val) {
    size_t tmp = tail.e++;
    buffer[tmp] = val;
    do
        CAS(tail.s,tmp,tmp+1);
    while (buffer[++tmp]);
}
```

A → PAUSE

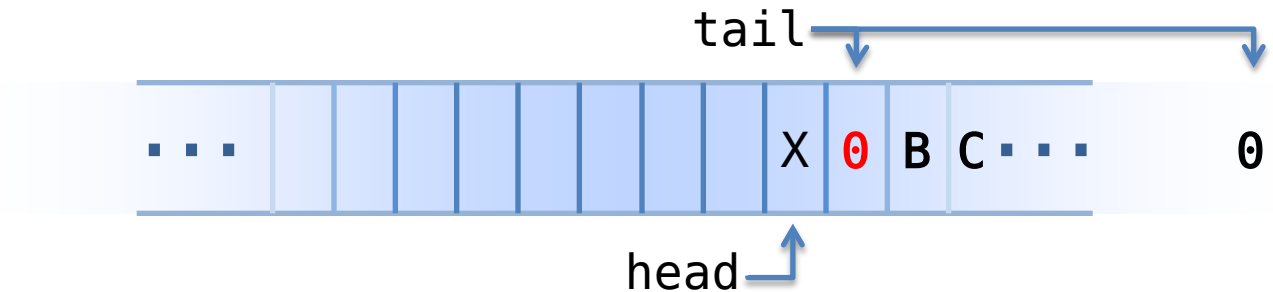
```
class Queue {
    atomic<int> buffer[SZ];
    atomic<size_t> head;
    struct {    atomic<size_t> s;
                atomic<size_t> e;
    } tail;
};
```



```
void push(int val) {
    size_t tmp = tail.e++;
    buffer[tmp] = val;
    do
        CAS(tail.s,tmp,tmp+1);
    while (buffer[++tmp]);
}
```

A → PAUSE

```
class Queue {
    atomic<int> buffer[SZ];
    atomic<size_t> head;
    struct {    atomic<size_t> s;
                atomic<size_t> e;
    } tail;
};
```

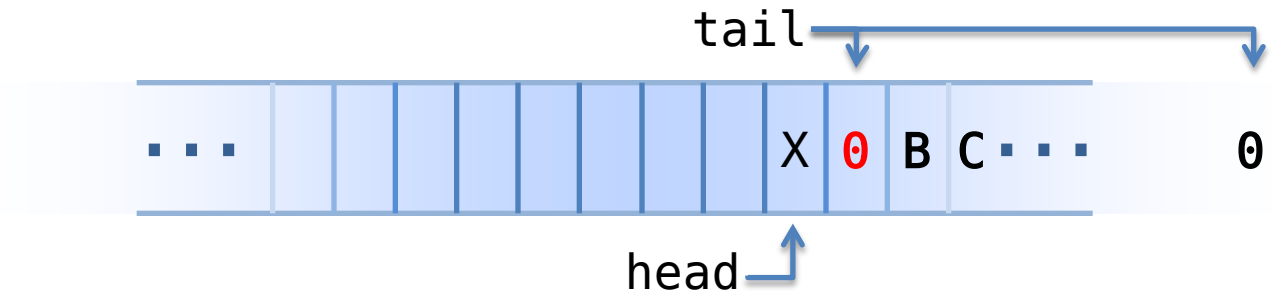


OK ?

```
void push(int val) {
    size_t tmp = tail.e++;
    buffer[tmp] = val;
    do
        CAS(tail.s,tmp,tmp+1);
    while (buffer[++tmp]);
}
```

A → PAUSE

```
class Queue {
    atomic<int> buffer[SZ];
    atomic<size_t> head;
    struct {    atomic<size_t> s;
                atomic<size_t> e;
    } tail;
};
```

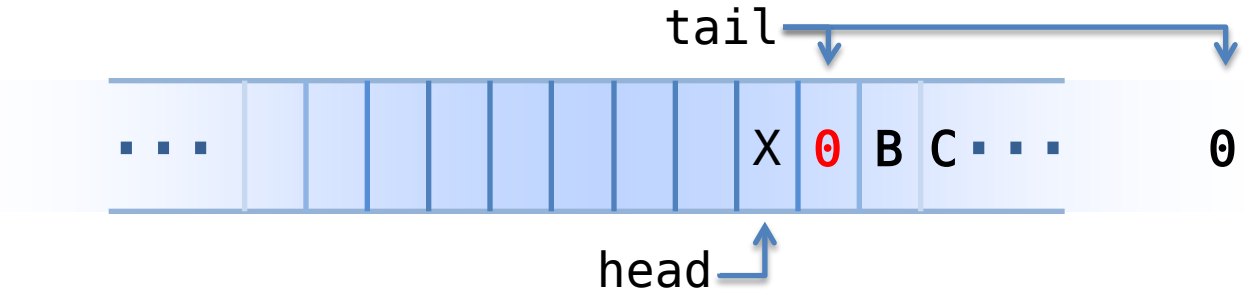


lock-free ?

```
void push(int val) {
    size_t tmp = tail.e++;
    buffer[tmp] = val;
    do
        CAS(tail.s,tmp,tmp+1);
    while (buffer[++tmp]);
}
```

A → PAUSE

```
class Queue {
    atomic<int> buffer[SZ];
    atomic<size_t> head;
    struct {   atomic<size_t> s;
               atomic<size_t> e;
    } tail;
};
```



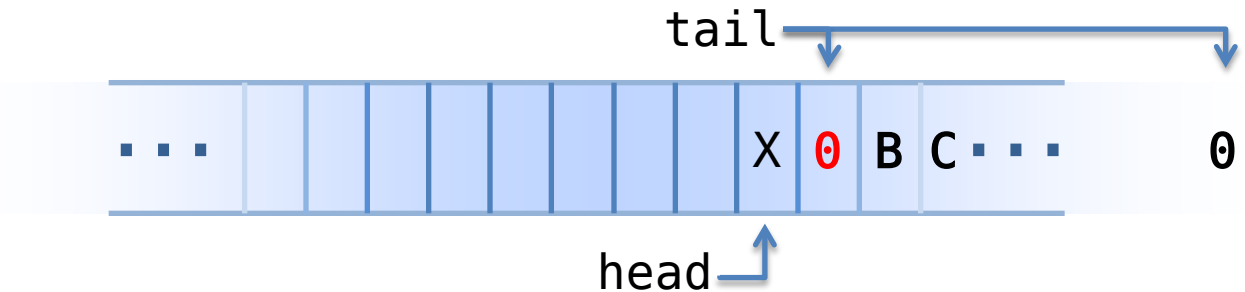

```
int pop() {
    if/while(!(head < tail.s))
        return/wait;
    …
}
```

lock-free ?

```
void push(int val) {
    size_t tmp = tail.e++;
    buffer[tmp] = val;
    do
        CAS(tail.s,tmp,tmp+1);
    while (buffer[++tmp]);
}
```

A → PAUSE

```
class Queue {
    atomic<int> buffer[SZ];
    atomic<size_t> head;
    struct {   atomic<size_t> s;
               atomic<size_t> e;
    } tail;
};
```



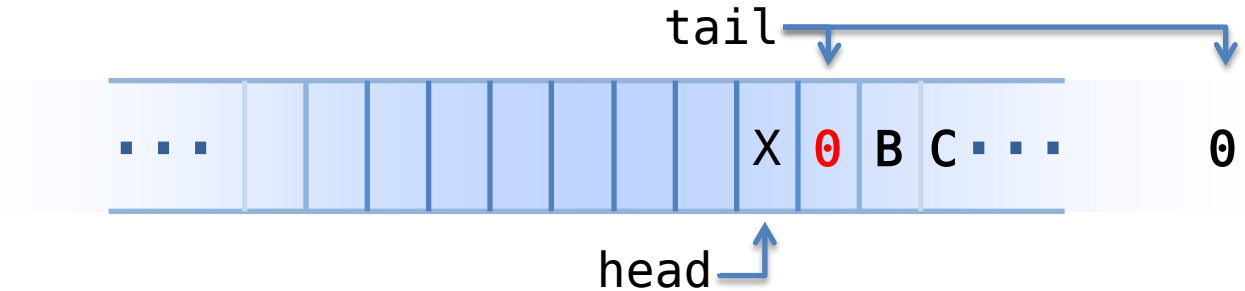

```
int pop() {
    if/while(!(head < tail.s))
        return/wait;
    …
}
```

"**block**-free" ?

```
void push(int val) {
    size_t tmp = tail.e++;
    buffer[tmp] = val;
    do
        CAS(tail.s,tmp,tmp+1);
    while (buffer[++tmp]);
}
```

A → PAUSE

```
class Queue {
    atomic<int> buffer[SZ];
    atomic<size_t> head;
    struct {    atomic<size_t> s;
                atomic<size_t> e;
    } tail;
};
```



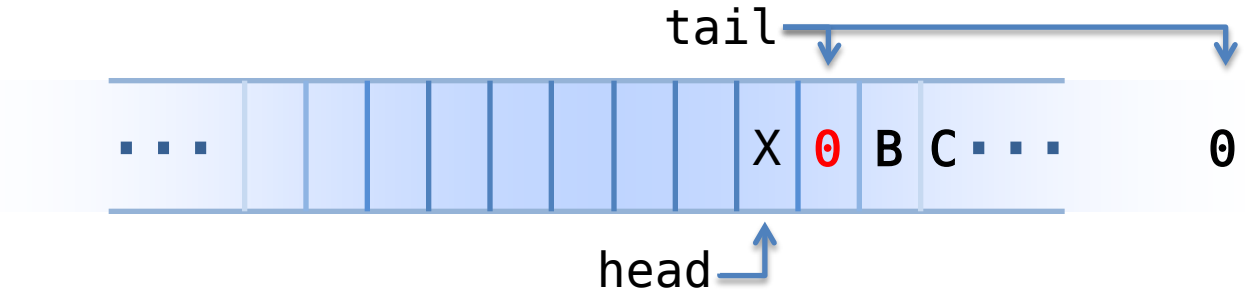

```
int pop() {
    if/while(
        retur
    …
}
```

An algorithm is ***lock-free*** if at all times **at least one thread** is guaranteed to be **making progress**.

```
void push(int val) {
    size_t tmp = tail.e++;
    buffer[tmp] = val;
    do
        CAS(tail.s,tmp,tmp+1);
    while (buffer[++tmp]);
}
```

A → PAUSE

```
class Queue {
    atomic<int> buffer[SZ];
    atomic<size_t> head;
    struct {    atomic<size_t> s;
                atomic<size_t> e;
    } tail;
};
```



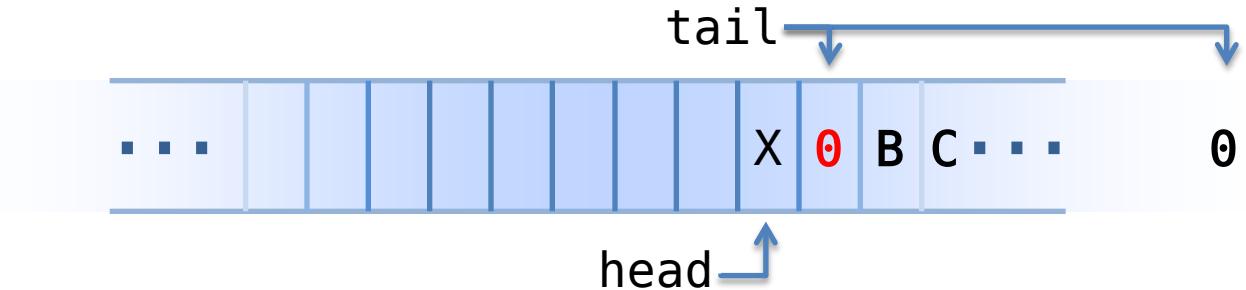

```
int pop() {
    if/
    …
}
```

If I suspended a certain thread at the worst time, for a long time or forever, do bad things happen?
Yes -> not lockfree.

```
void push(int val) {
    size_t tmp = tail.e++;
    buffer[tmp] = val;
    do
        CAS(tail.s,tmp,tmp+1);
    while (buffer[++tmp]);
}
```

A → PAUSE

```
class Queue {
    atomic<int> buffer[SZ];
    atomic<size_t> head;
    struct {   atomic<size_t> s;
               atomic<size_t> e;
    } tail;
};
```



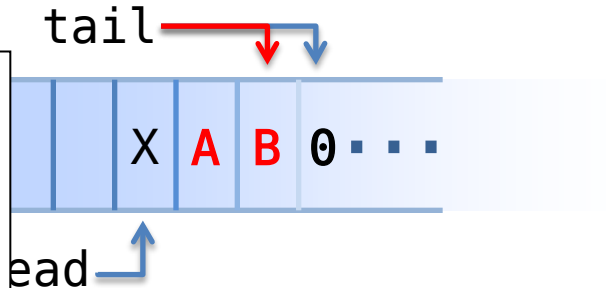

```
int pop() {
    if/while(!(head < tail.s))
        return/wait;
    …
}
```

!= lock-free

```
void push(int val) {
    size_t tmp = tail.e++;
    buffer[tmp] = val;
    bool r;
    do
        r = CAS(tail.s,tmp,tmp+1);
    while (r && buffer[++tmp]);
}
```

```
class Queue {
    atomic<int> buffer[SZ];
    atomic<size_t> head;
    struct {    atomic<size_t> s;
                atomic<size_t> e;
    } tail;
};
```

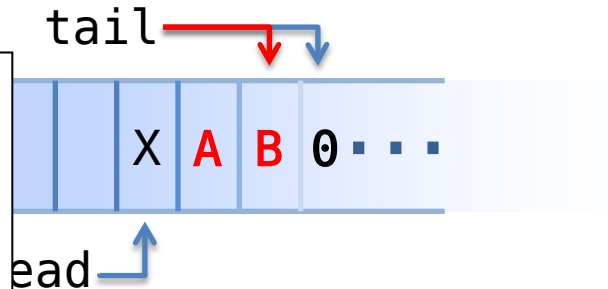
```
void push(int val) {
    size_t tmp = tail.e++;
    buffer[tmp] = val;
    do
        CAS(tail.s,tmp,tmp+1);
    while (buffer[++tmp]);
}
```



```
void push(int val) {
    size_t tmp = tail.e++;
    buffer[tmp] = val;
    bool r;
    do
        r = CAS(tail.s,tmp,tmp+1);
    while (r && buffer[++tmp]);
}
```

A → PAUSE

```
class Queue {
    atomic<int> buffer[SZ];
    atomic<size_t> head;
    struct {    atomic<size_t> s;
                atomic<size_t> e;
    } tail;
};
```




```
void push(int val) {
    size_t tmp = tail.e++;
    buffer[tmp] = val;
    do
        CAS(tail.s,tmp,tmp+1);
    while (buffer[++tmp]);
}
```

```
void push(int val) {
    size_t tmp = tail.e++;
    buffer[tmp] = val;
    bool r;
    do
        r = CAS(tail.s,tmp,tmp+1);
    while (r && buffer[++tmp]);
}
```

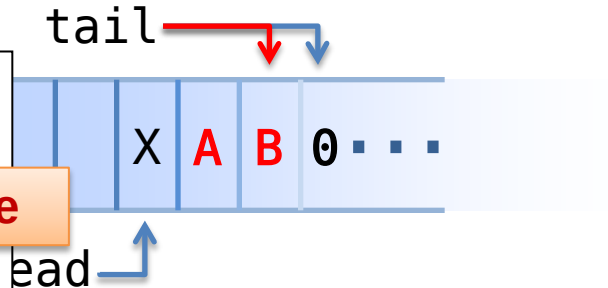
!= lock-free

A → PAUSE

```
void push(int val) {
    size_t tmp = tail.e++;
    buffer[tmp] = val;
    do
        CAS(tail.s,tmp,tmp+1);
    while (buffer[++tmp]);
}
```

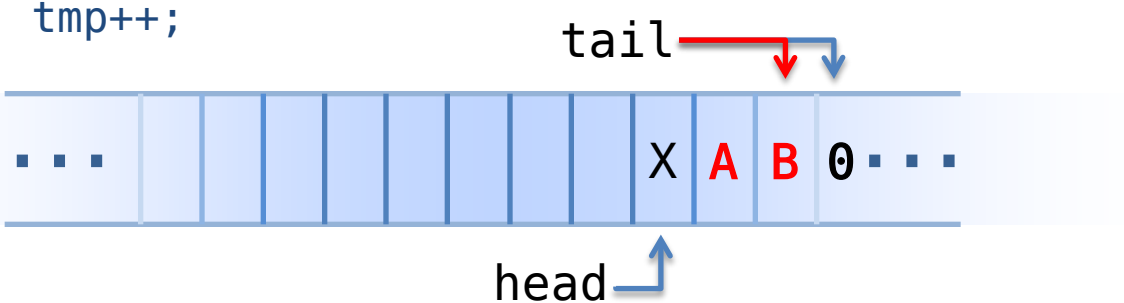
~= lock-free

```
class Queue {
    atomic<int> buffer[SZ];
    atomic<size_t> head;
    struct {    atomic<size_t> s;
                atomic<size_t> e;
    } tail;
};
```



```
void push(int val) {
    size_t tmp = tail.e++;
    buffer[tmp] = val;
    bool r;
    tmp = tail.s;
    while (buffer[tmp]) {
        r = tail.s.CAS(tmp,tmp+1);
        if (r) tmp++;
    }
}
```

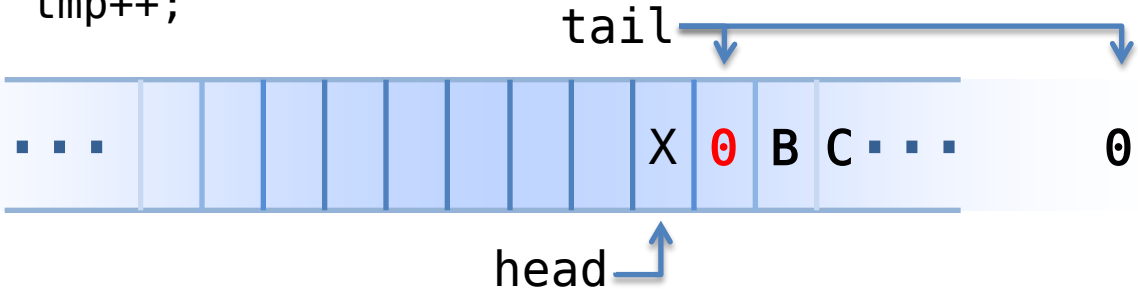
```
class Queue {
    atomic<int> buffer[SZ];
    atomic<size_t> head;
    struct {    atomic<size_t> s;
                atomic<size_t> e;
    } tail;
};
```



```
void push(int val) {
    size_t tmp = tail.e++;
    buffer[tmp] = val;
    bool r;
    tmp = tail.s;
    while (buffer[tmp]) {
        r = tail.s.CAS(tmp,tmp+1);
        if (r) tmp++;
    }
}
```

A → PAUSE

```
class Queue {
    atomic<int> buffer[SZ];
    atomic<size_t> head;
    struct {   atomic<size_t> s;
               atomic<size_t> e;
    } tail;
};
```




```
int pop() {
    if/while(!(head < tail.s))
        return/wait;
    …
}
```

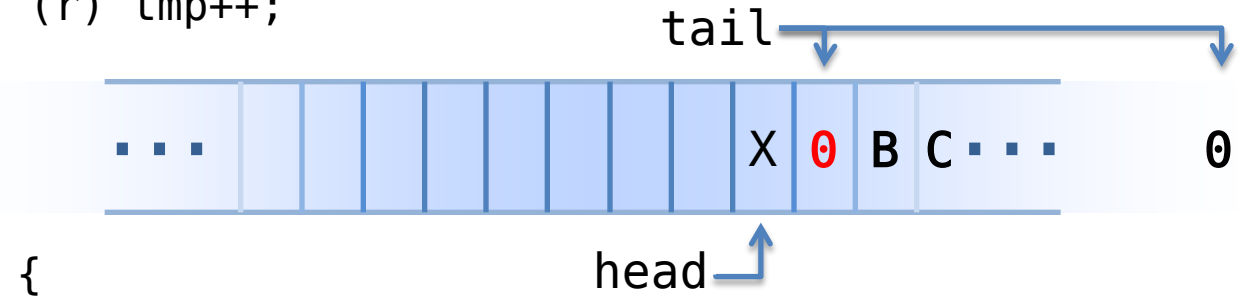
!= lock-free


BlackBerry
HERE
TO

```
void push(int val) {
    size_t tmp = tail.e++;
    buffer[tmp] = val;
    bool r;
    tmp = tail.s;
    while (buffer[tmp]) {
        r = tail.s.CAS(tmp,tmp+1);
        if (r) tmp++;
    }
}
```

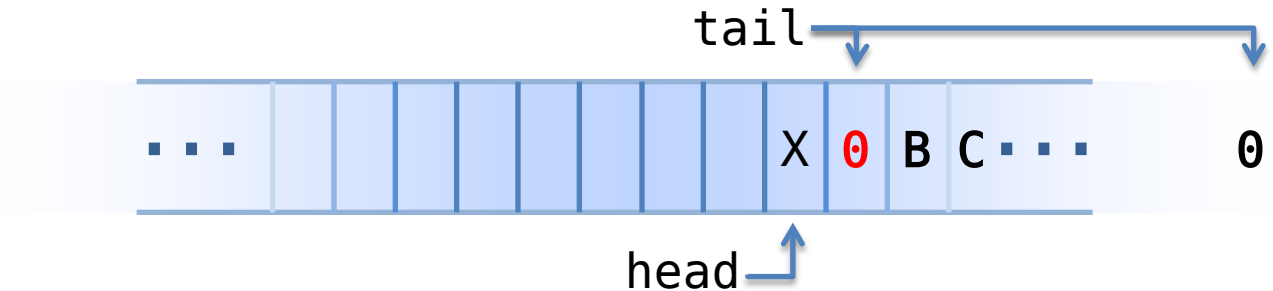
A → PAUSE

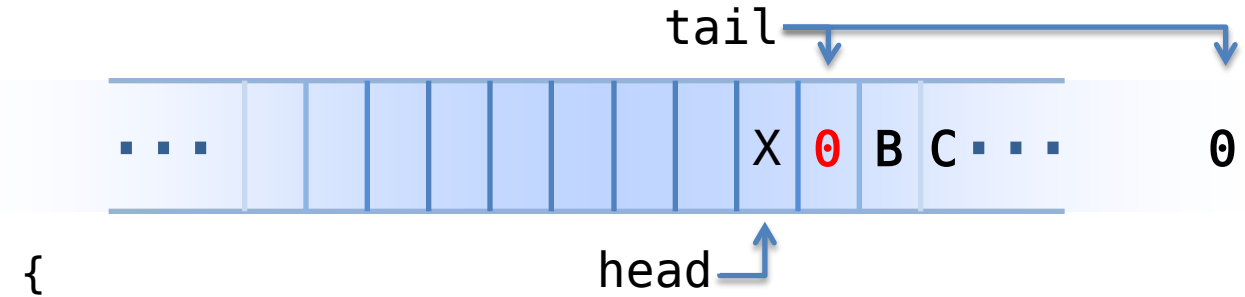
```
class Queue {
    atomic<int> buffer[SZ];
    atomic<size_t> head;
    struct {    atomic<size_t> s;
                atomic<size_t> e;
    } tail;
};
```




```
int pop() {
    while
        ;
    …
}
```

An algorithm is ***lock-free*** if at all times **at least one thread** is guaranteed to be **making progress**.


tail
...
X
0
B
C
...
0
head

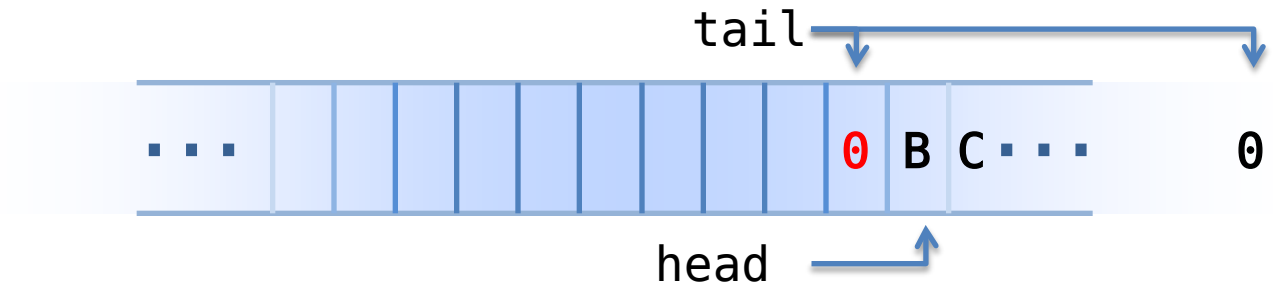



```
int pop() {
    while (!(head < tail.s))
        ;
    …
}
```

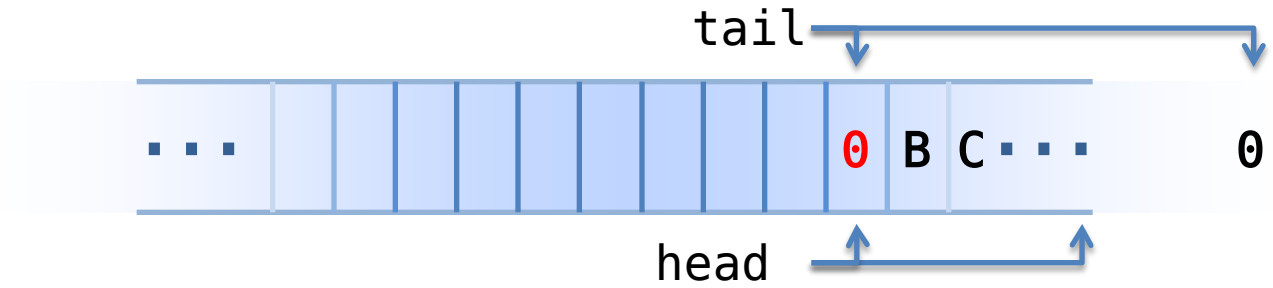
don’t want to wait


```
int pop() {
    …
}
```

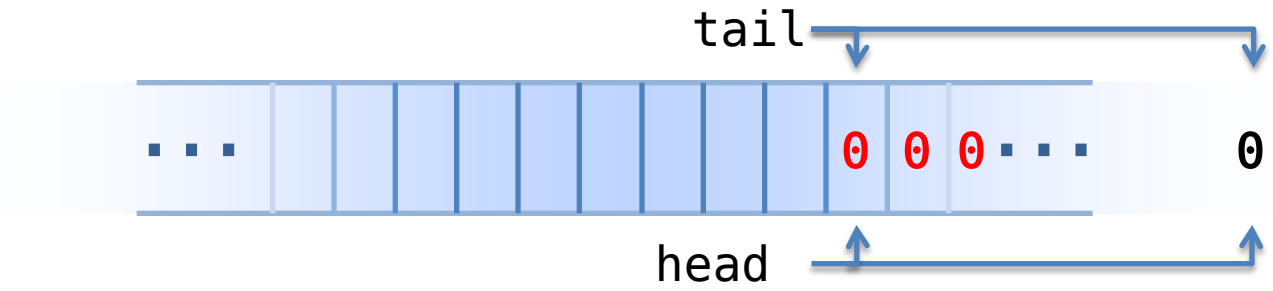
don’t wait



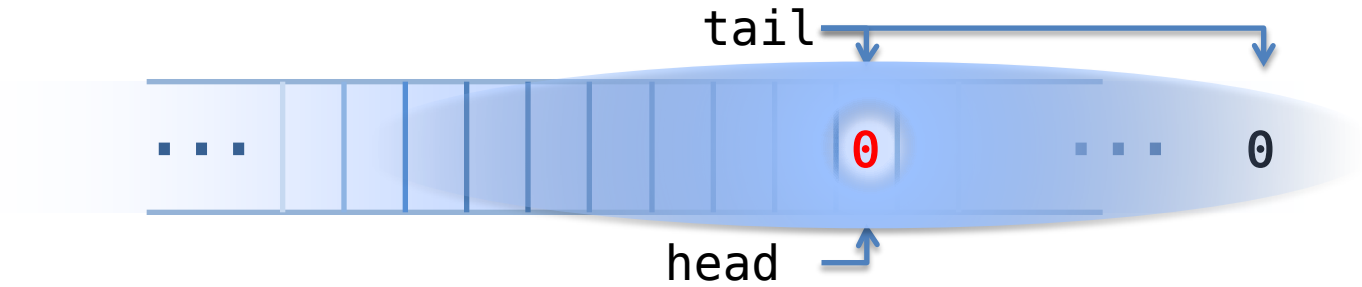
don’t wait! (?)

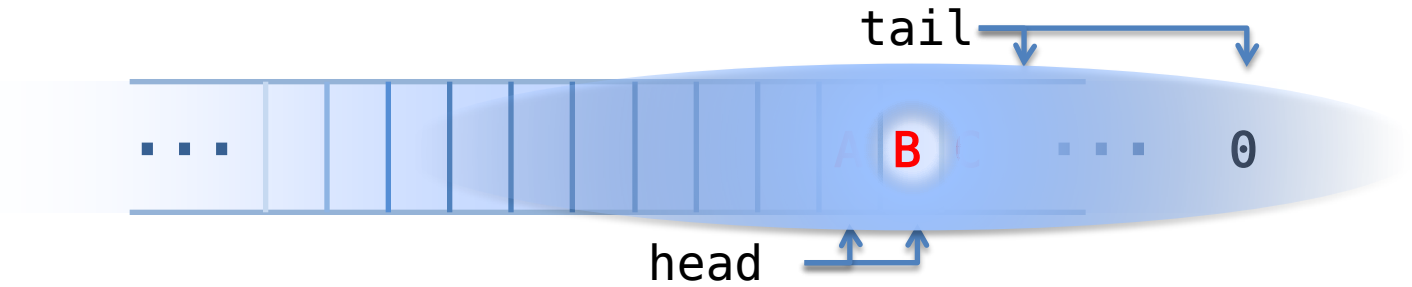


come back later (when?)

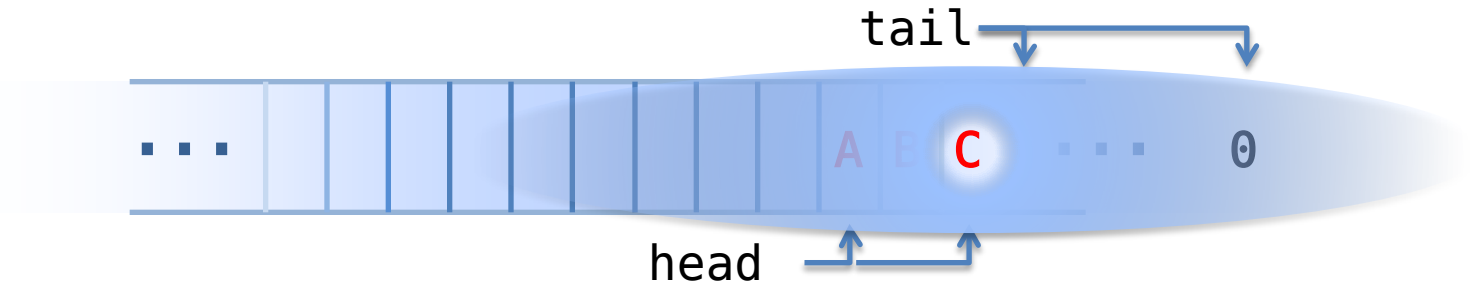


spin?

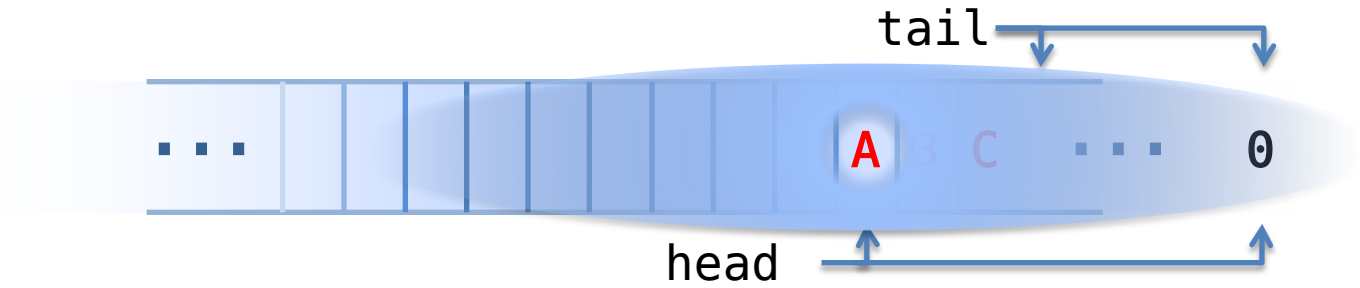

tail
0
0
head



pop: B

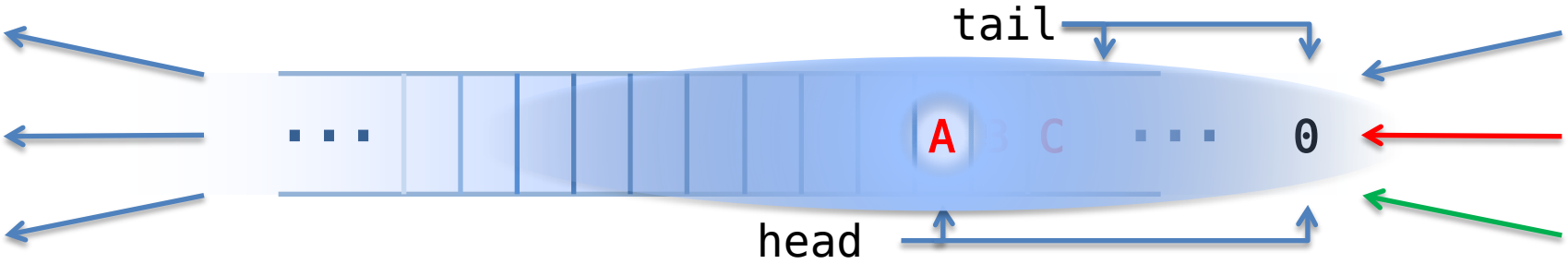


pop: B C



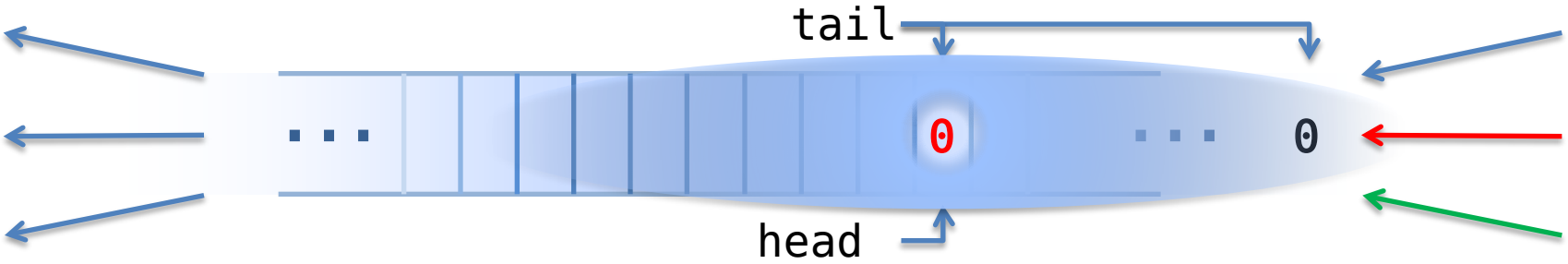
pop: **B C A**

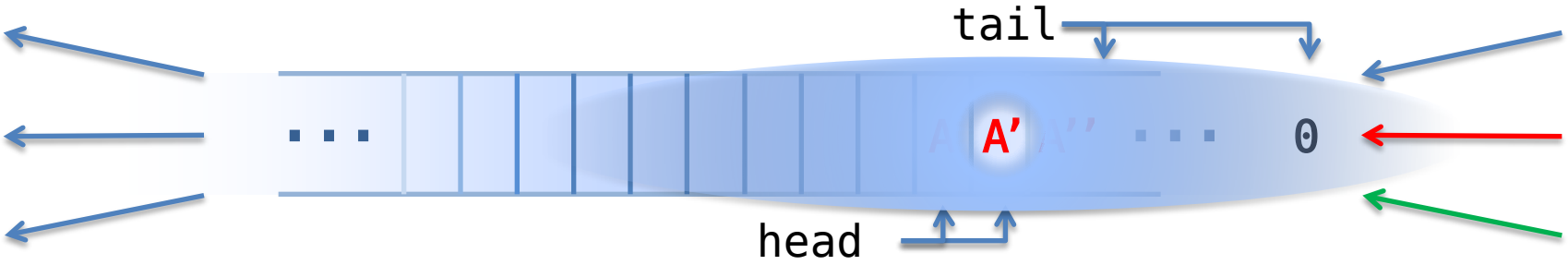
pop order ?



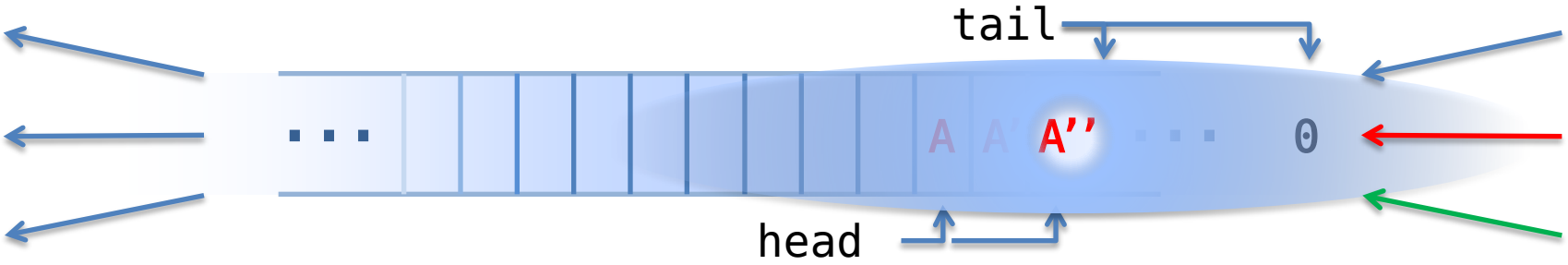
pop: B C A

pop order ?

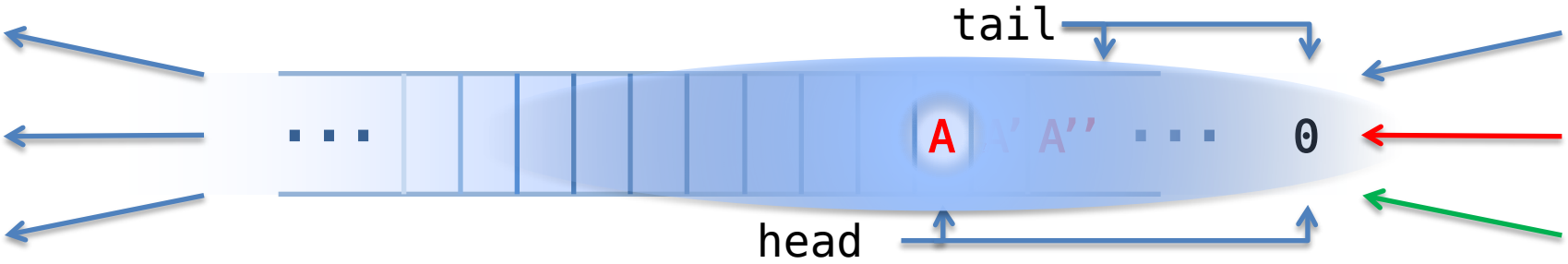

tail
0
0
head



pop: A'

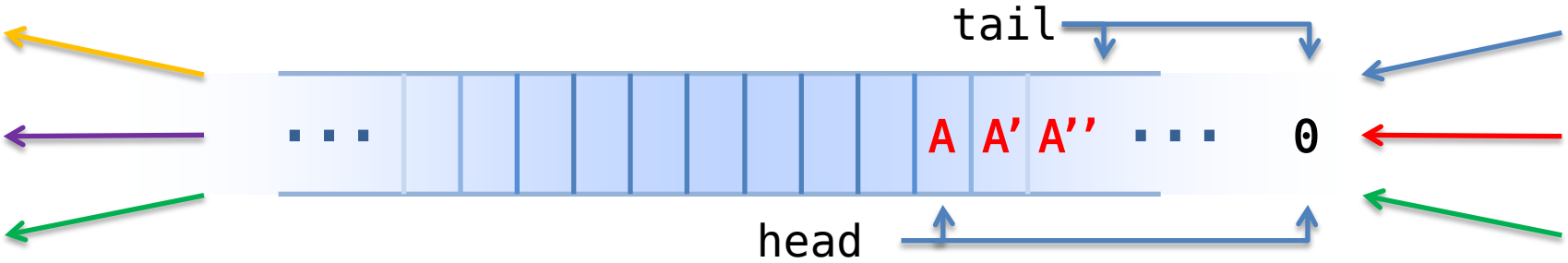


pop: A' A''



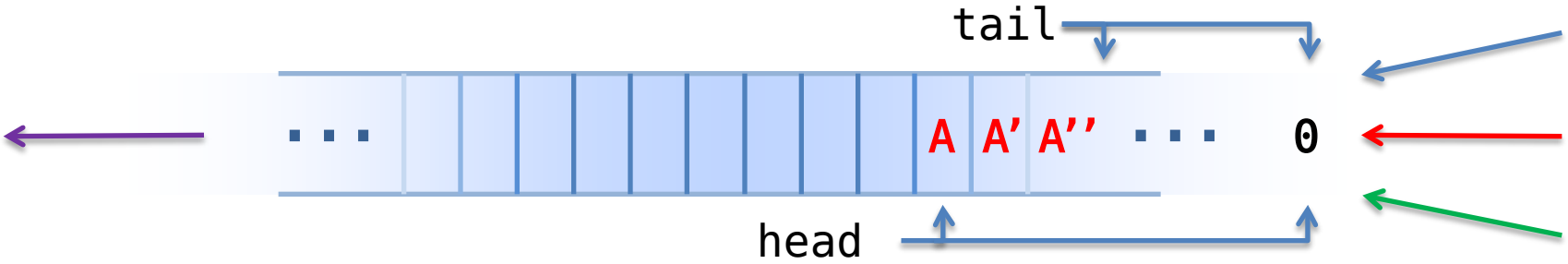
pop: A' A'' A

pop order ?



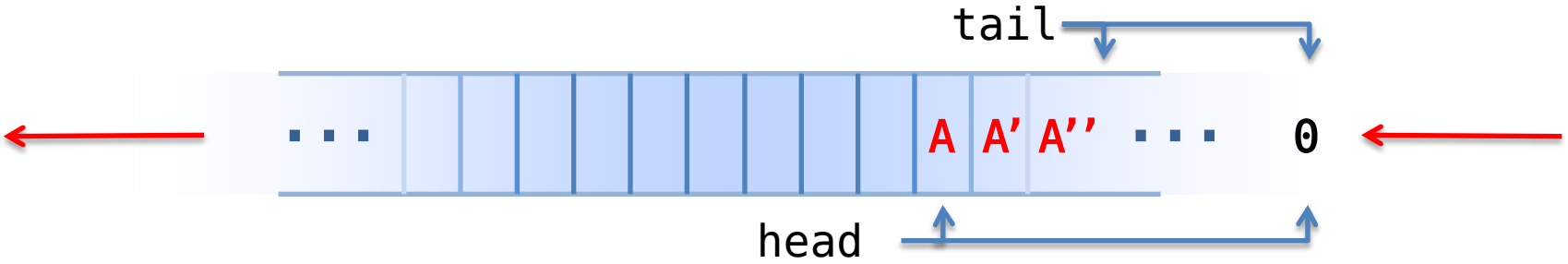
pop: A' A'' A

pop order ?



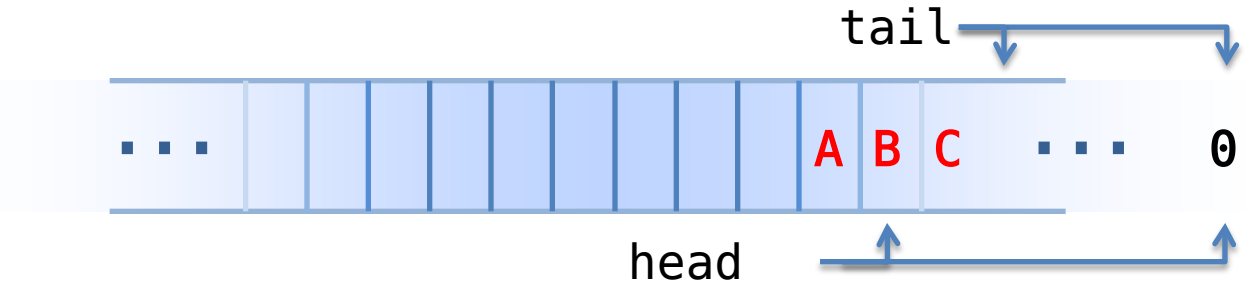
pop: A' A'' A

pop order ?



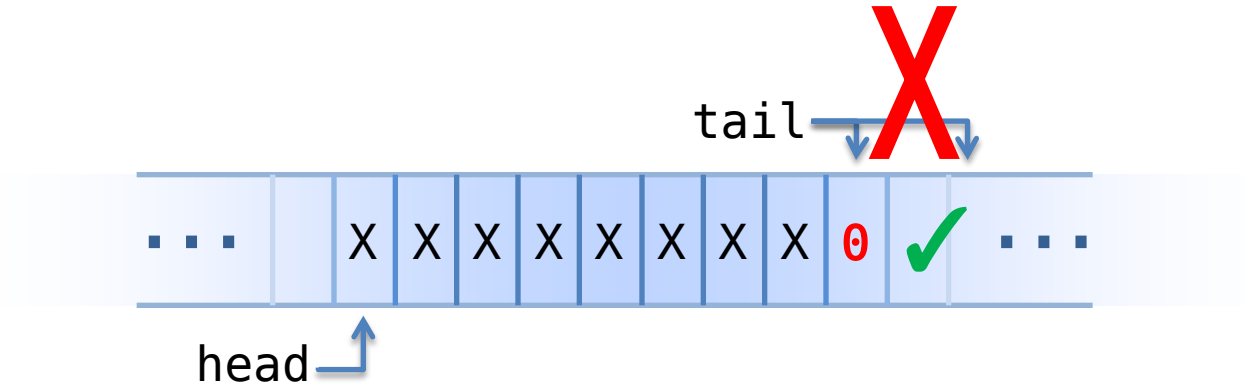
pop: A' A'' A

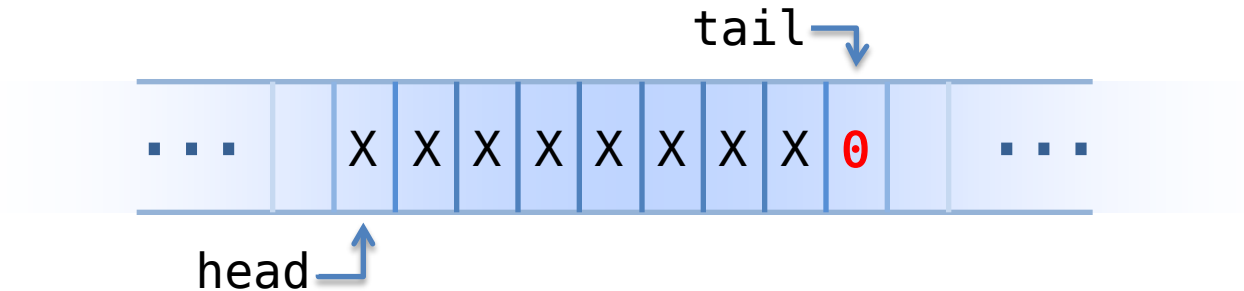
pop order ?

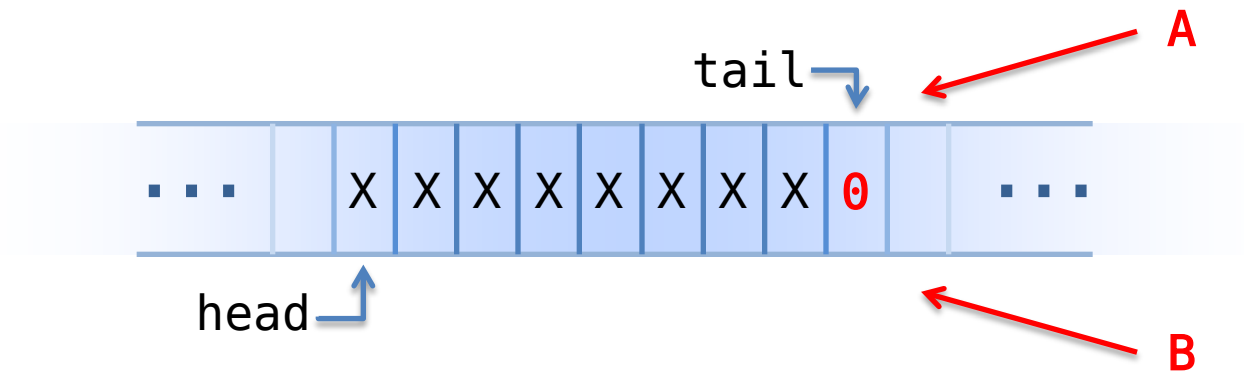


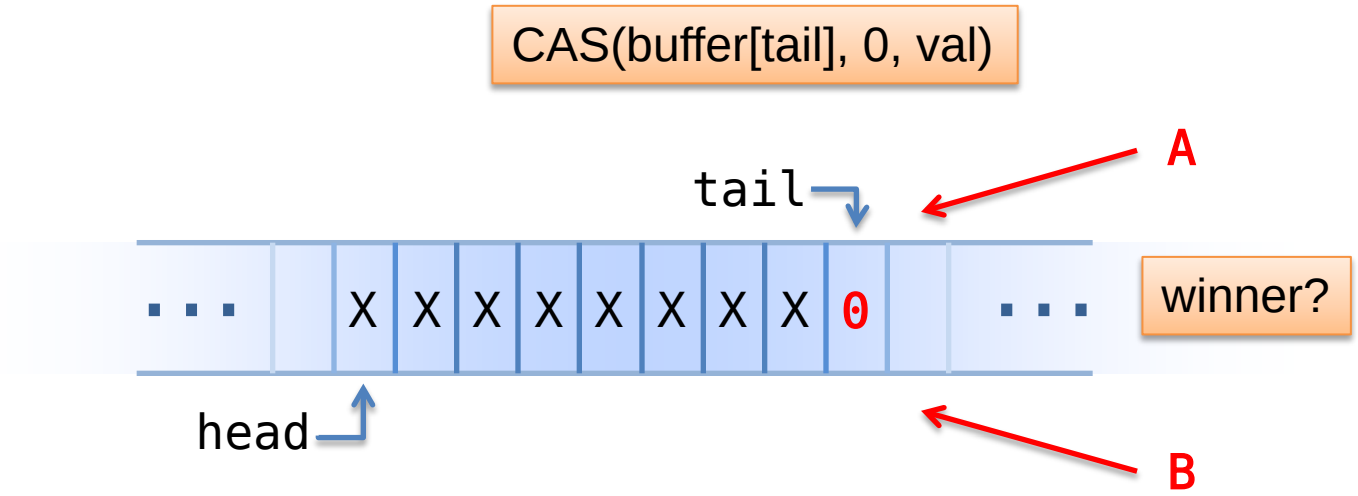
pop: B C A … B?

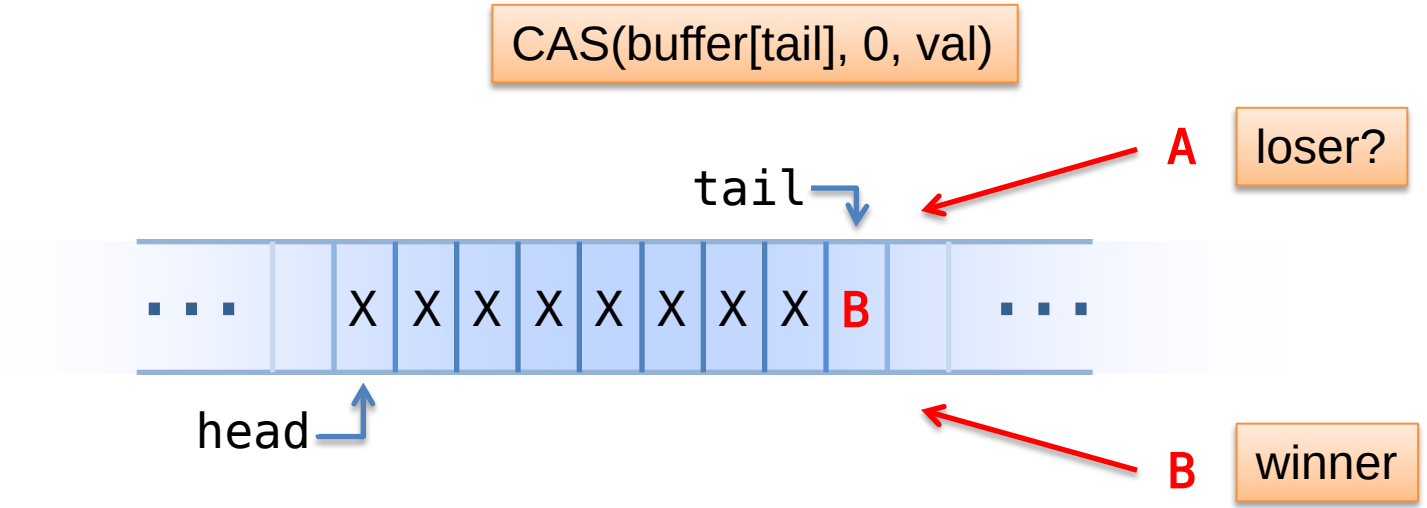
did I read B already ?


tail
X
. . .
X X X X X X X X 0
. . .
head

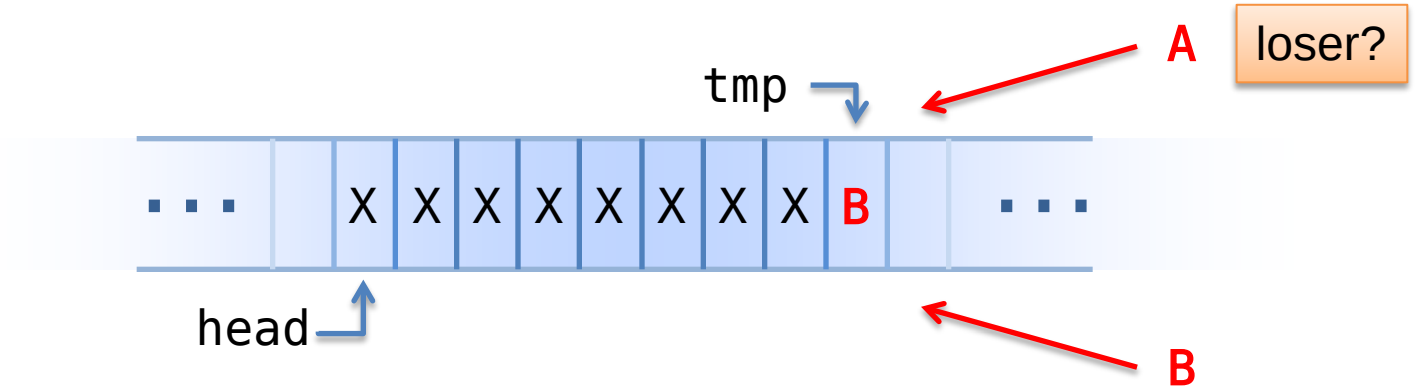

tail
...
X X X X X X X X 0
...
head


A
tail
. . .
X X X X X X X X 0
. . .
head
B


CAS(buffer[tail], 0, val)
A
tail
. . .
X X X X X X X X 0
. . .
winner?
head
B

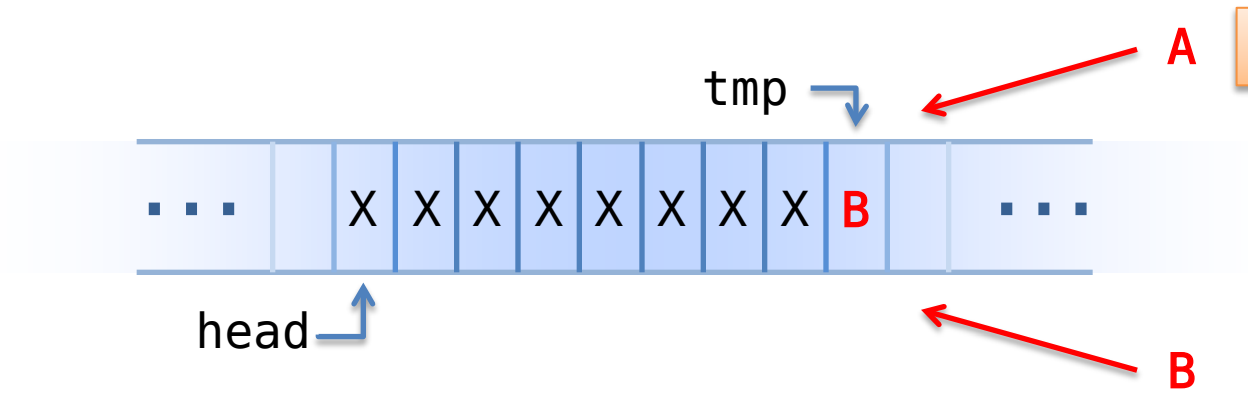

CAS(buffer[tail], 0, val)
A
loser?
tail
. . .
X X X X X X X X B
. . .
head
B
winner

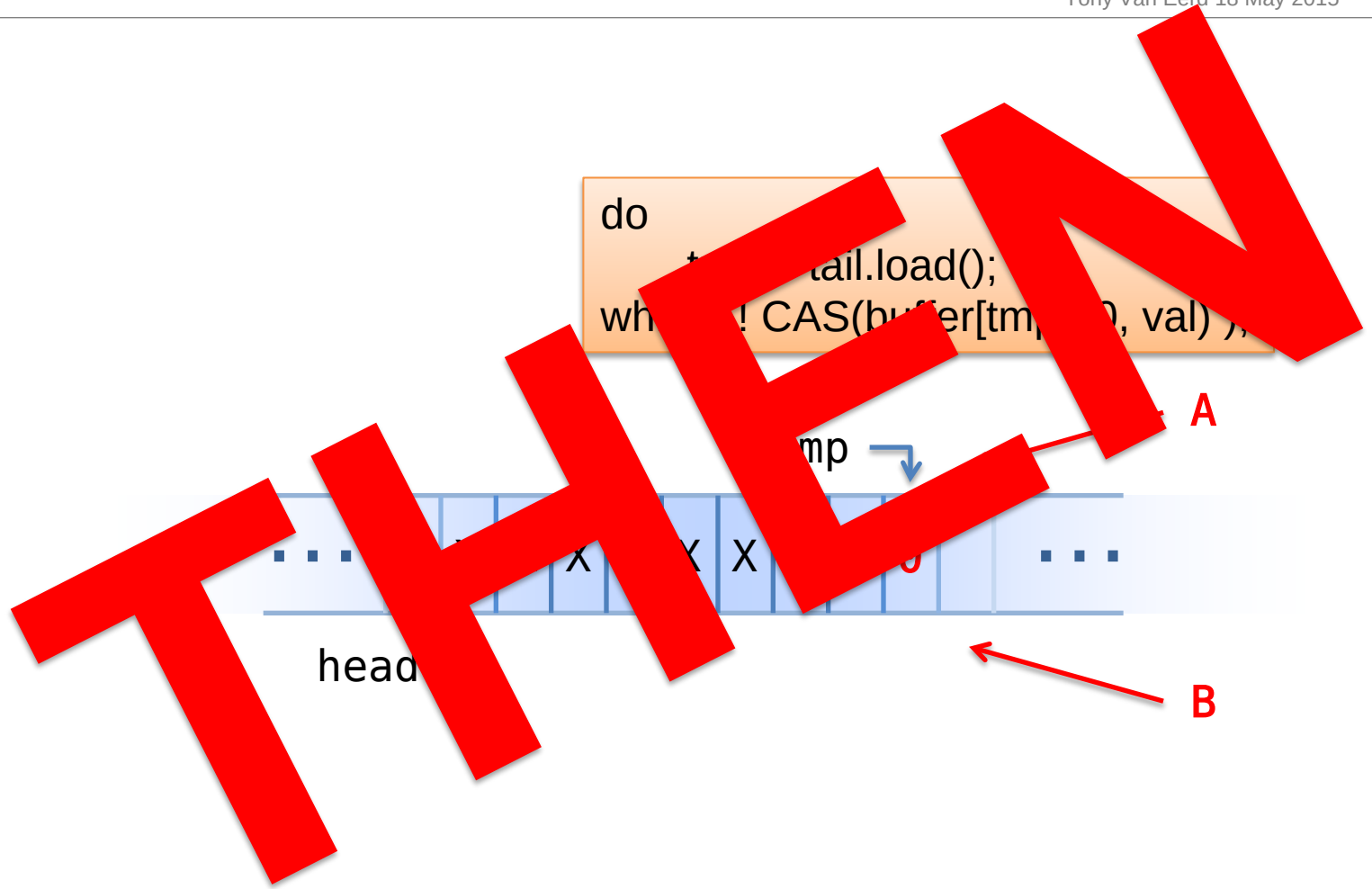
```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], 0, val) );
```



```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], 0, val) );
```

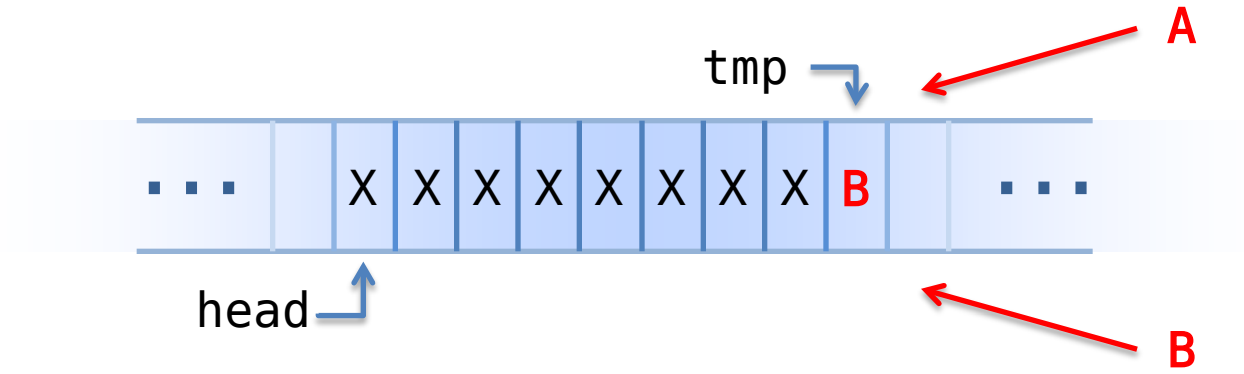
wait for tail?




do
tail.load();
wh ! CAS(buffer[tm , val));
mp
A
X X X
head
B
THEN

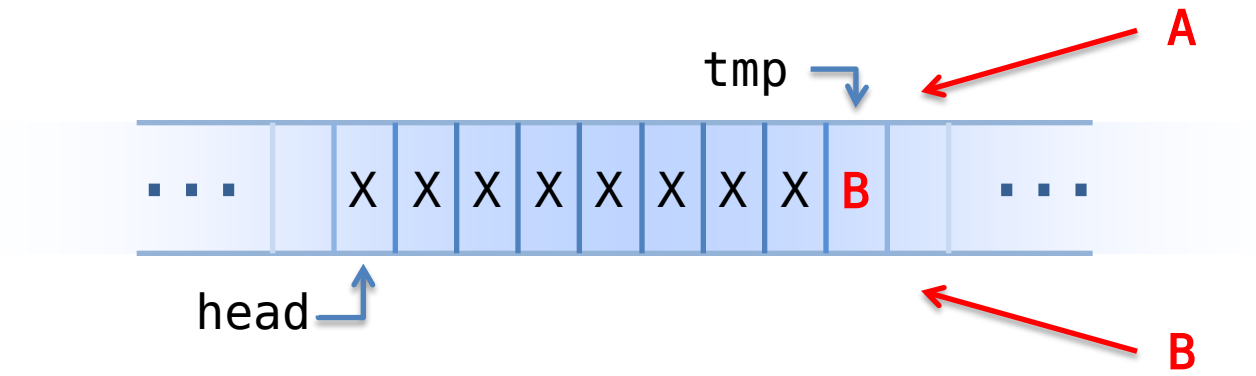
```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], 0, val) );
```

if CAS fails **THEN** try again



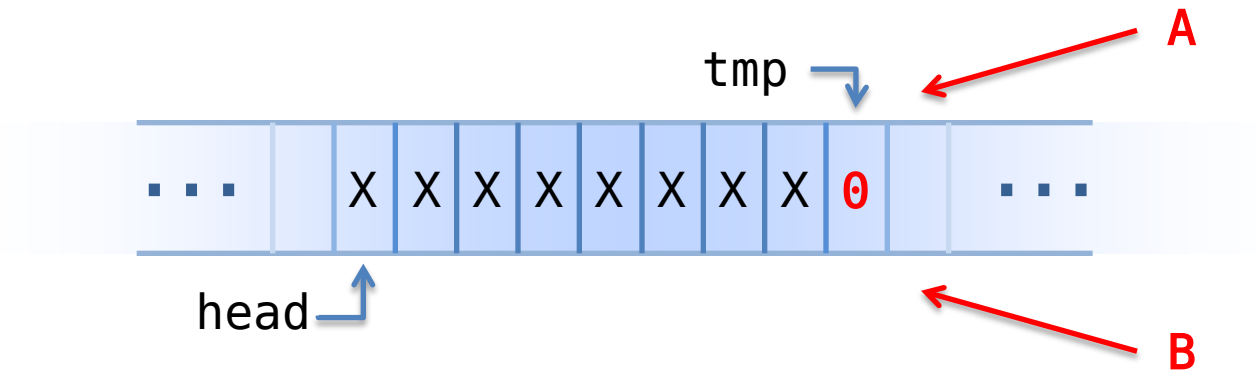
```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], 0, val) );
```

read tail
THEN read buffer



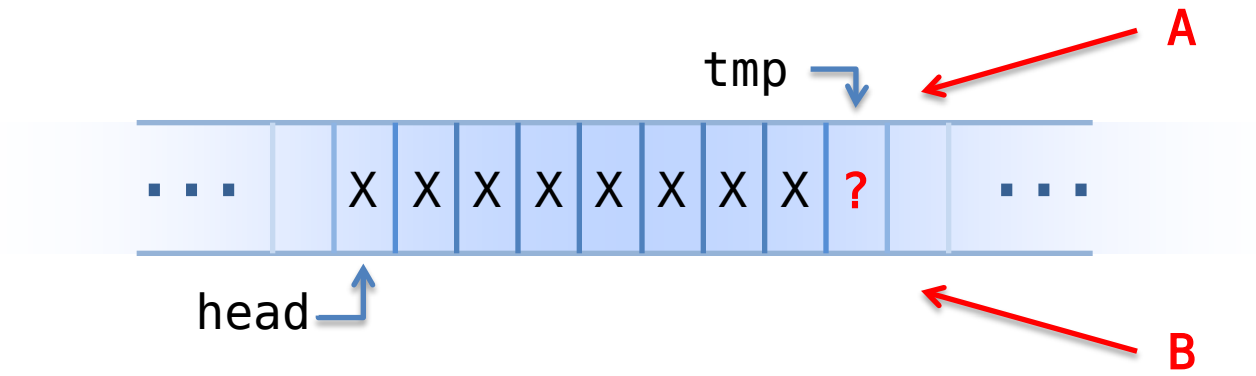
```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], 0, val) );
```

read tail
THEN read buffer



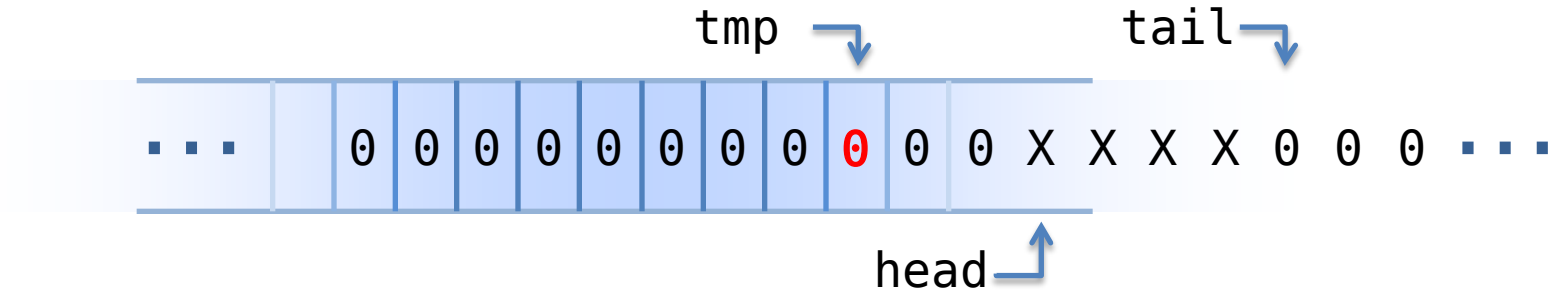
```
do
      tmp = tail.load();
while ( ! CAS(buffer[tmp], 0, val) );
```

read tail
THEN read buffer



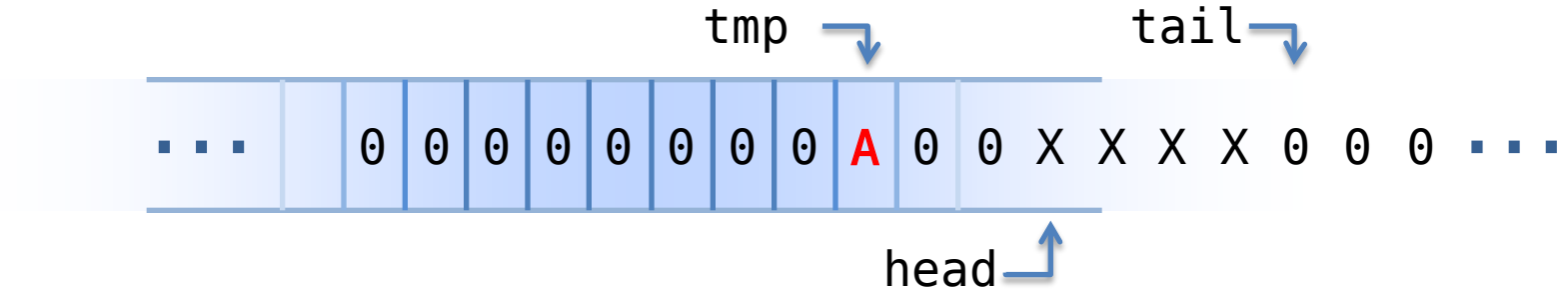
```
do
      tmp = tail.load();
while ( ! CAS(buffer[tmp], 0, val) );
```

read tail
THEN read buffer

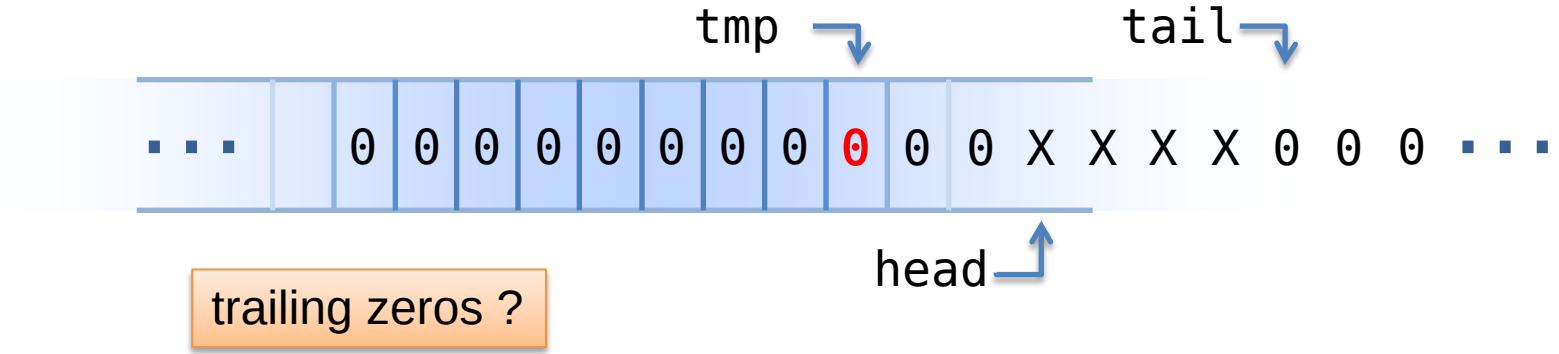


```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], 0, val) );
```

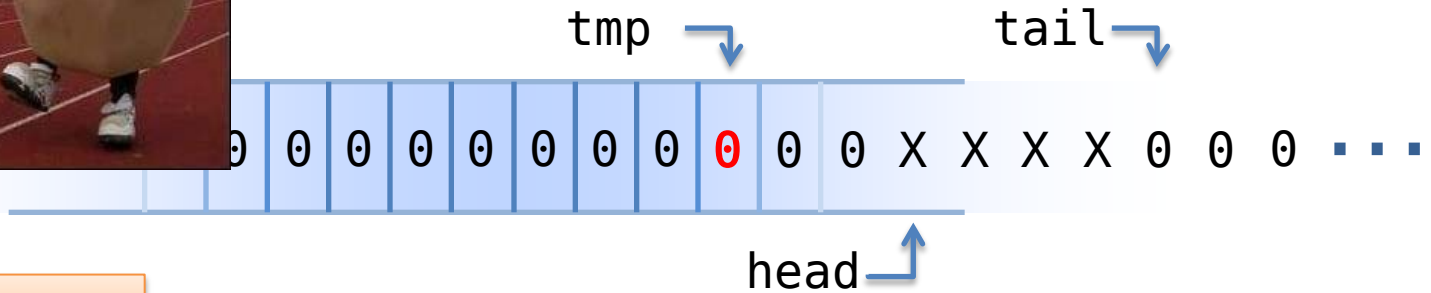
read tail
THEN read buffer



```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], 0, val) );
```

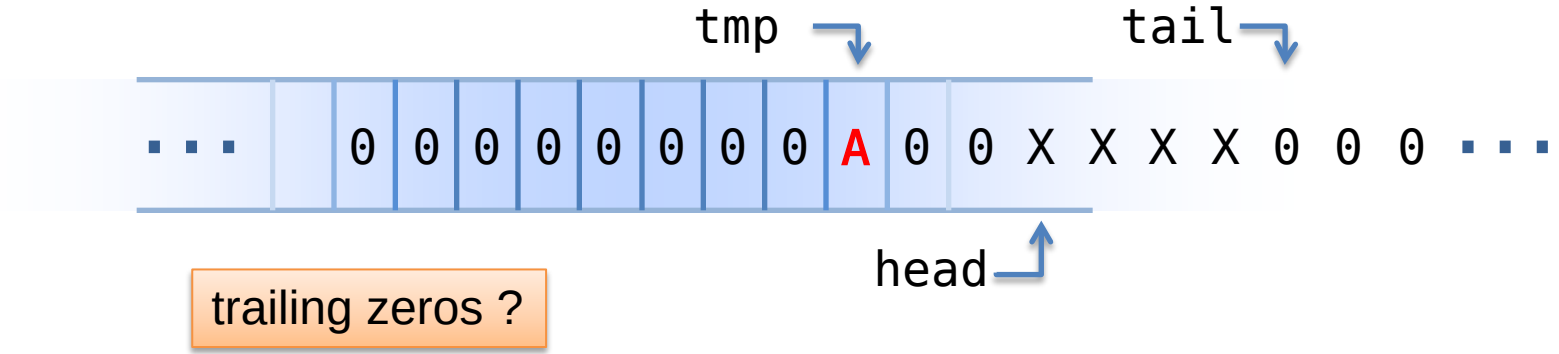


```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], 0, val) );
```



pop() ?

```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], 0, val) );
```



```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], 0, val) );
```

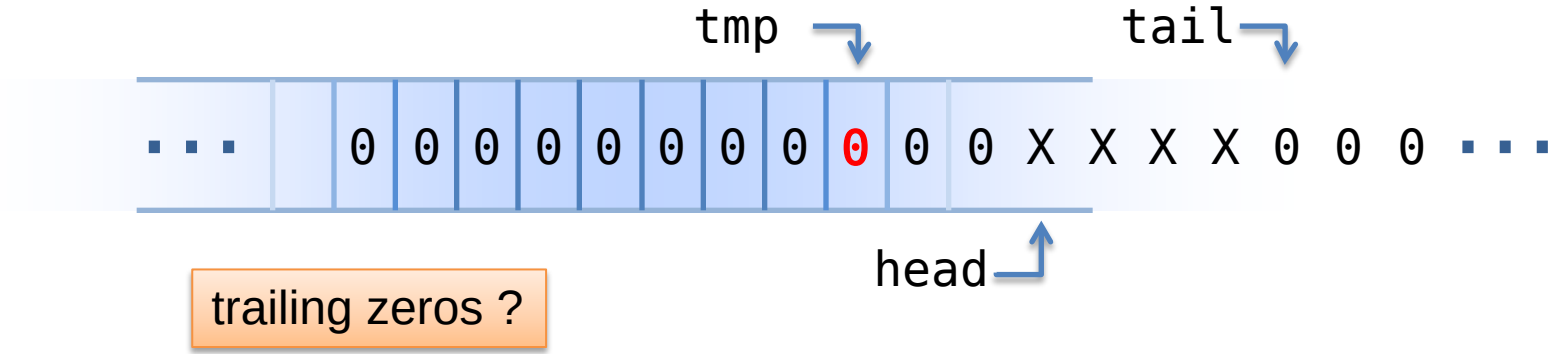
Compromise…

Queue of int

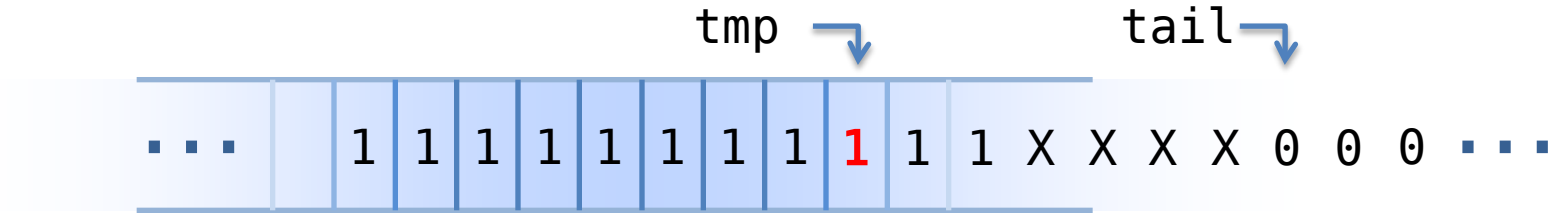
-> Queue of int != 0

-> Queue of int != 0 or 1

```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], 0, val) );
```

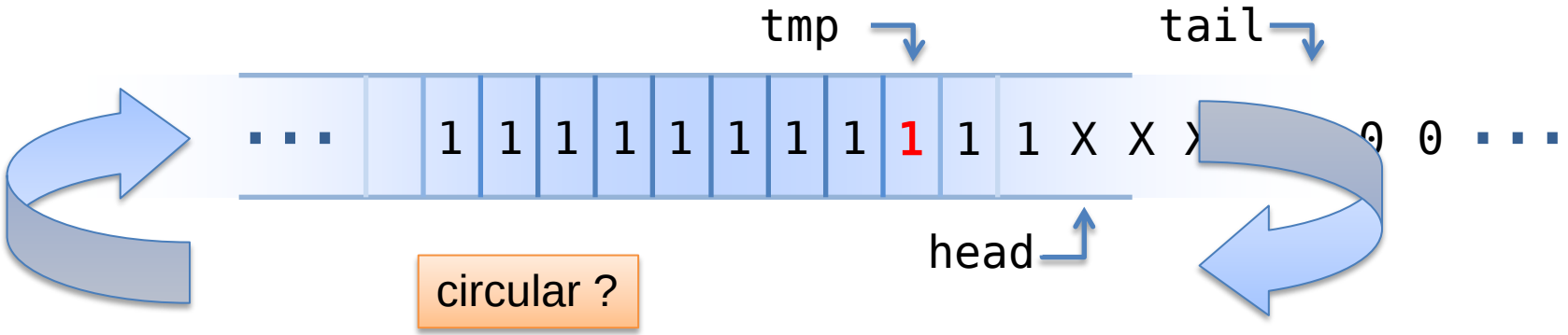


```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], 0, val) );
```

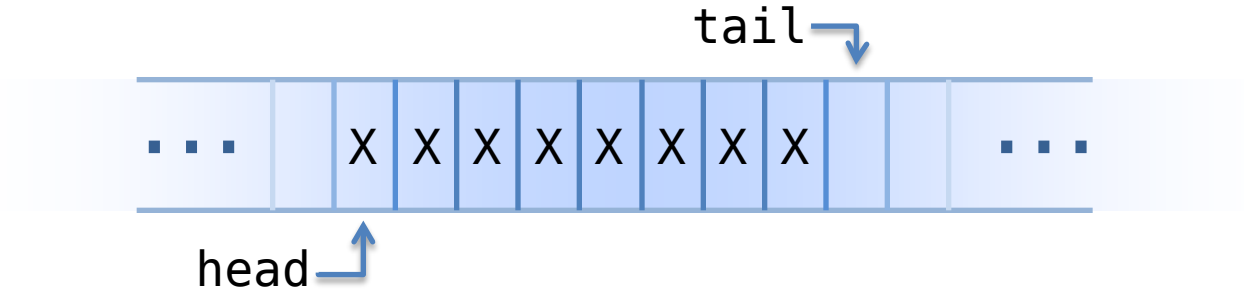


pop() ?

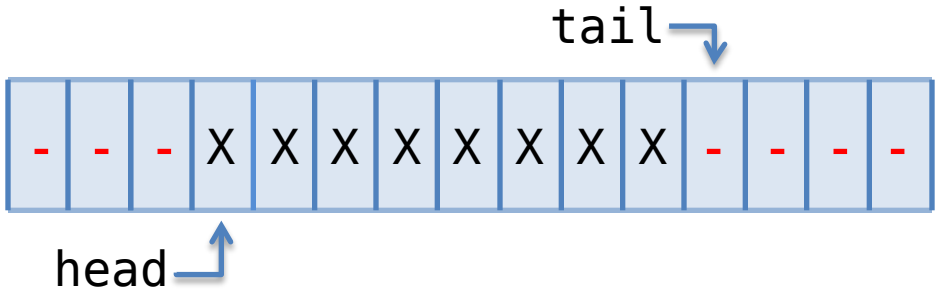
```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], 0, val) );
```



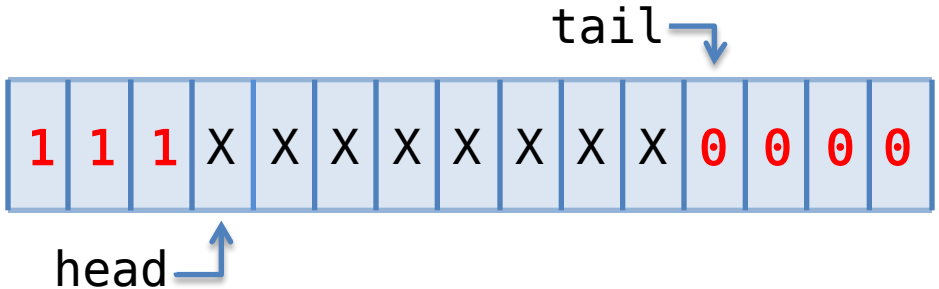
```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], 0, val) );
```

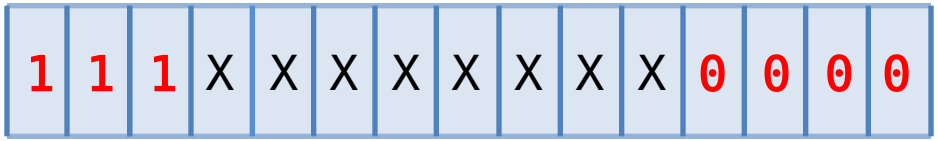


```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], 0, val) );
```

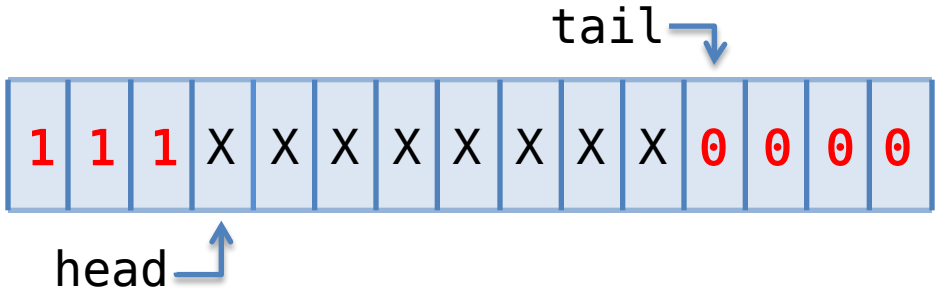


```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], 0, val) );
```

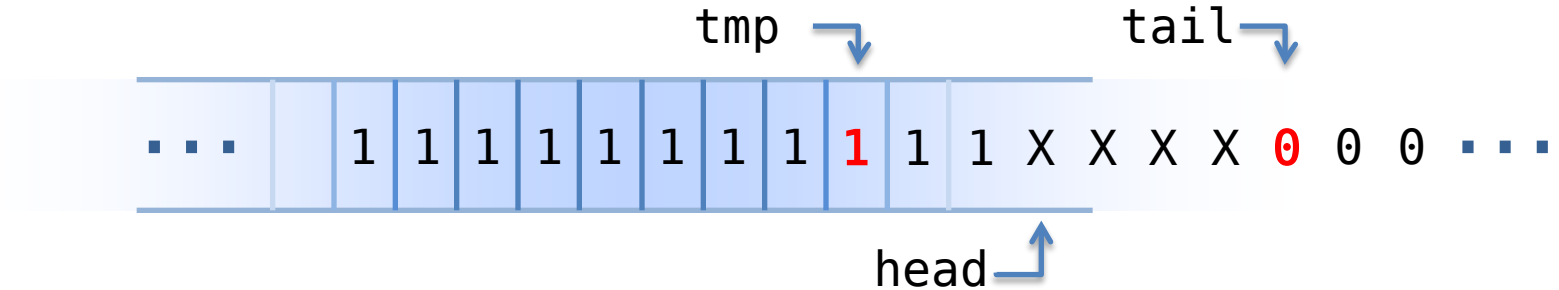



1 1 1 X X X X X X X X 0 0 0 0

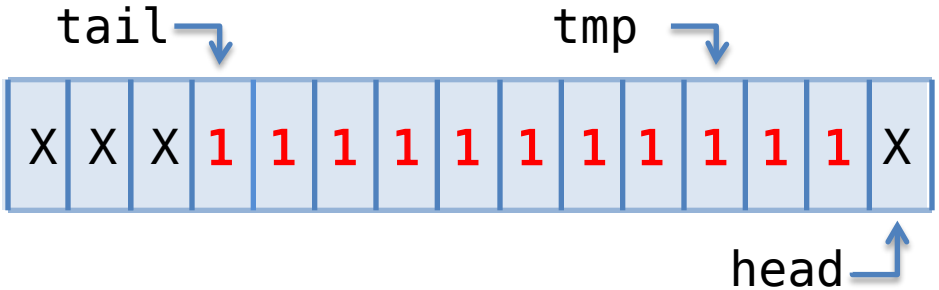
```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], 0, val) );
```



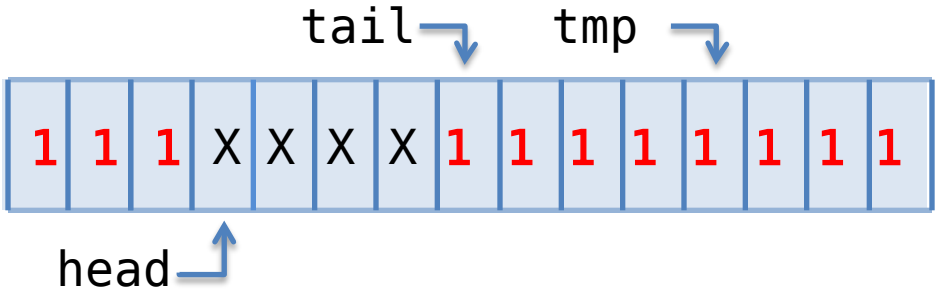
```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], 0, val) );
```



```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], 0, val) );
```



```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], 0, val) );
```



```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], 0, val) );
```

Compromise…

```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], 0, val) );
```

Compromise…

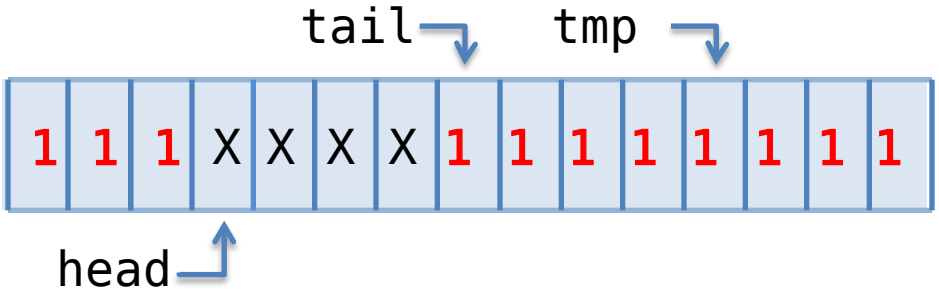
Queue of int

-> Queue of int != 0

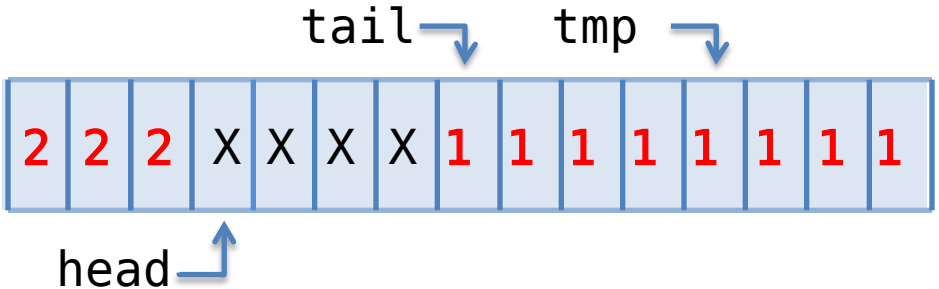
-> Queue of int != 0 or 1

-> Queue of int < 0

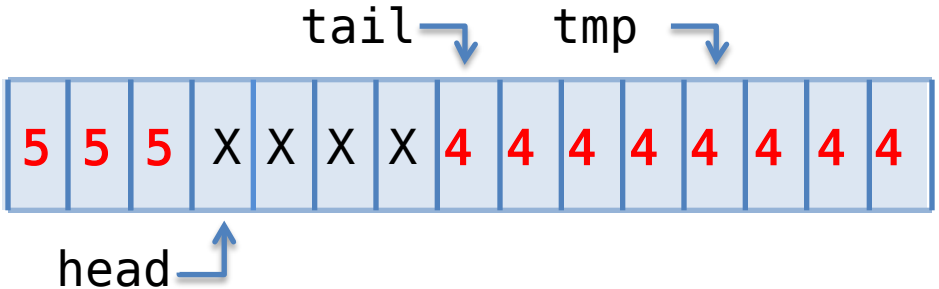
```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], 0, val) );
```



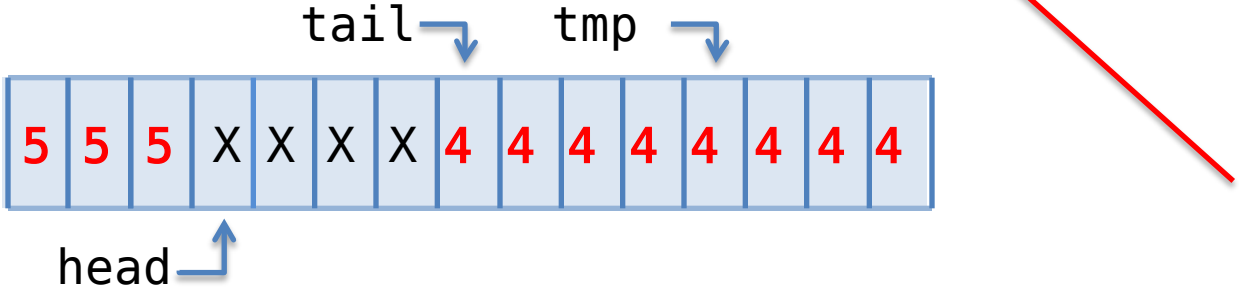
```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], 0, val) );
```



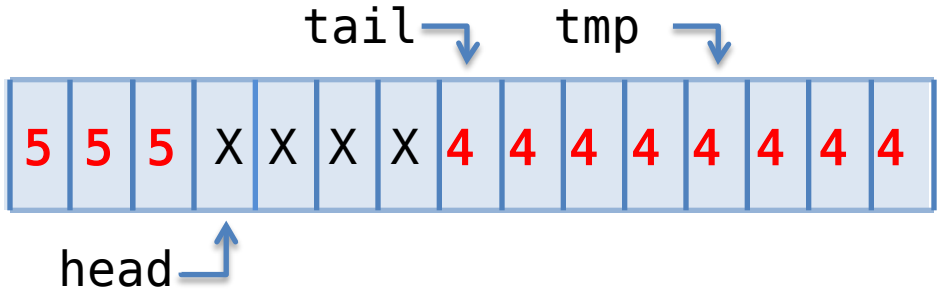
```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], 0, val) );
```



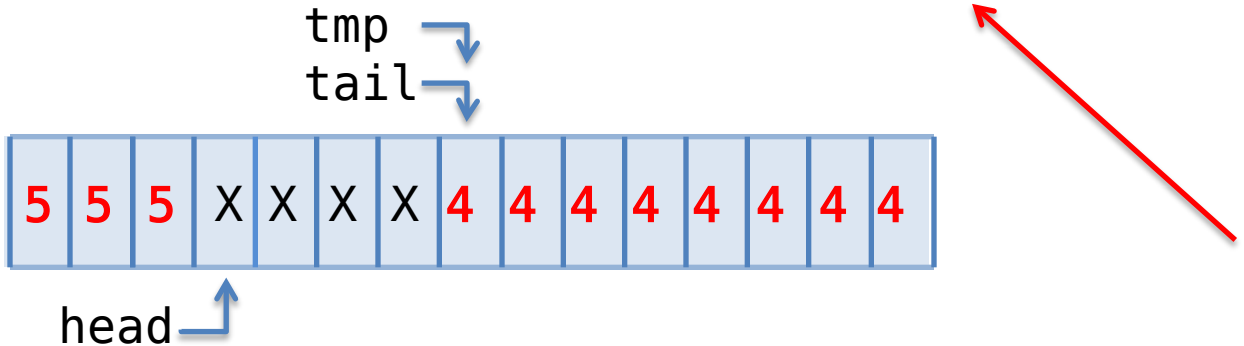
```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], 0, val) );
```



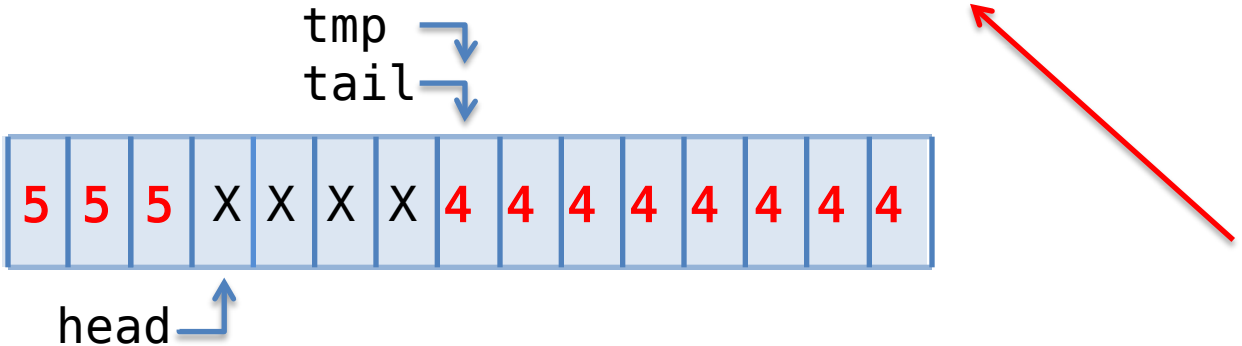
```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], 0, val) );
```



```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], 0, val) );
```

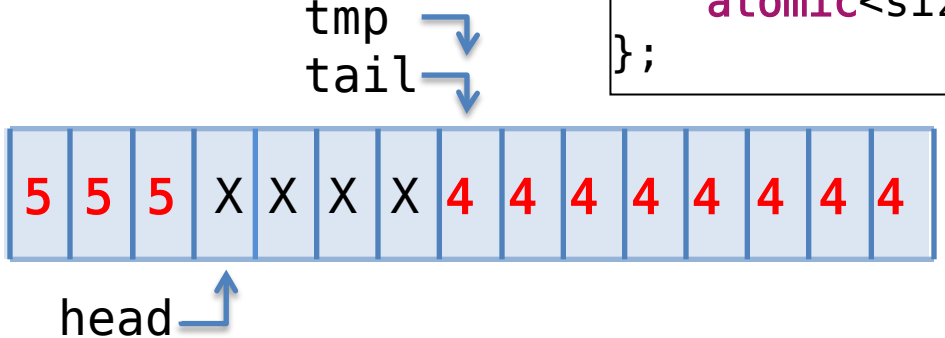


```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], 4, val) );
```



```
class Queue {
    atomic<int> buffer[SZ];
    atomic<size_t> head;
    struct {    atomic<size_t> s;
                atomic<size_t> e;
    } tail;
    atomic<size_t> generation;
};
```

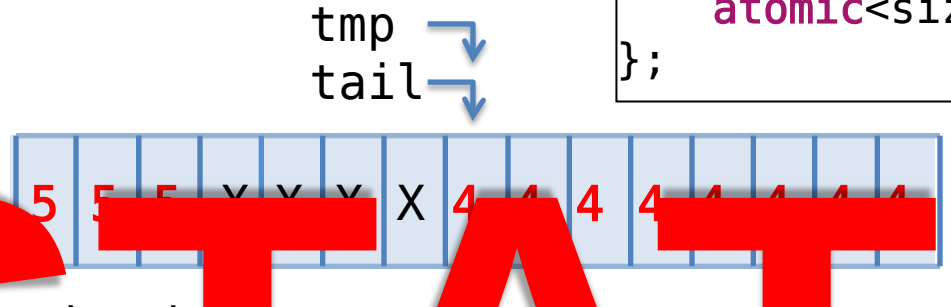
```
do
    tmp = t
while ( ! CA
```



MOAR STATE

```
cl  Queue
    at ic<in   buffe  SZ];
    at ic<si   d;
       t {   atom <size_t> s;
             atomi size_t> e;
    } ta  ;
    atomic<size_t> generation;
};
```

```
do
   p =
  (!C
```



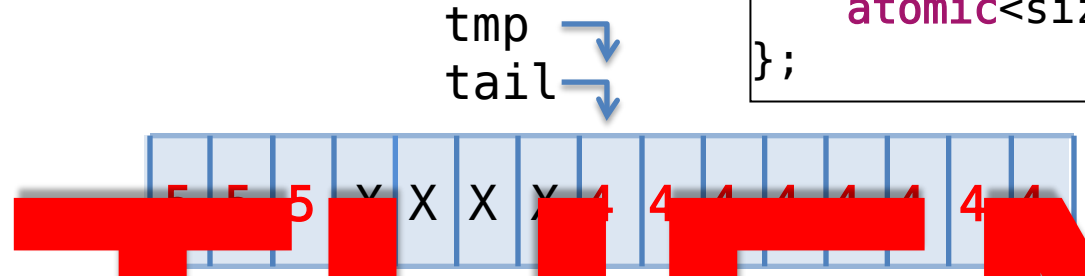
MOAR

```
cl  Queue
    at ic<in   buffe  SZ];
    at ic<si   d;
       t {   atom <size_t> s;
             atomi size_t> e;
    } ta  ;
    atomic<size_t> generation;
};
```

do

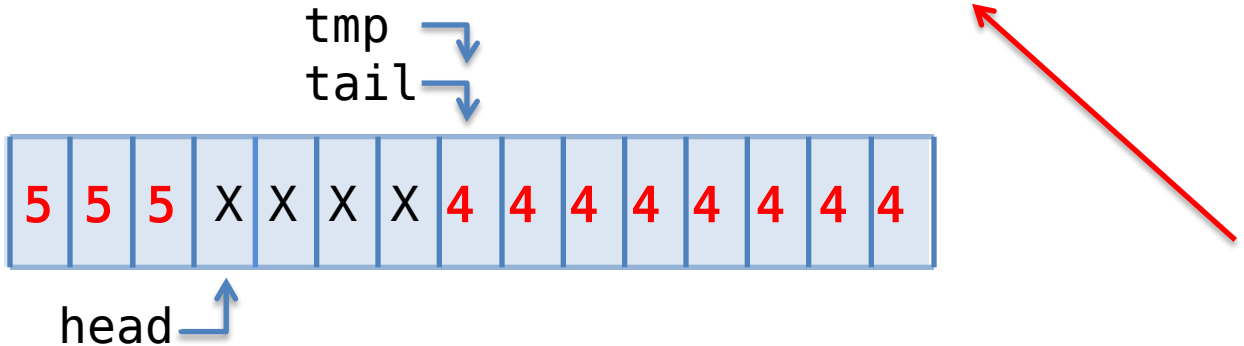
p =

(!C

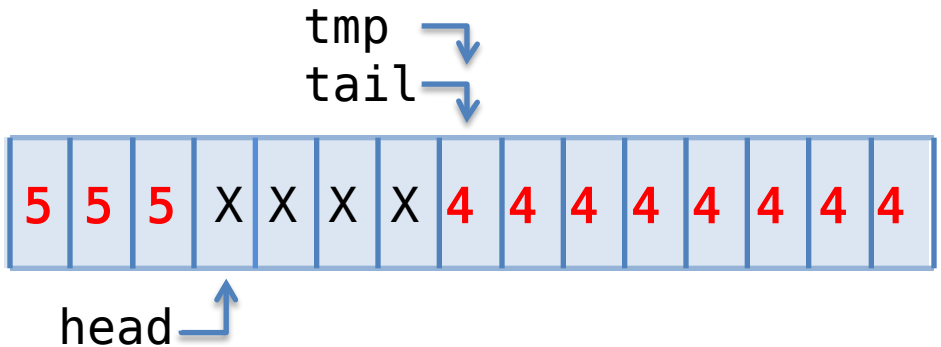


THEN

```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], 4, val) );
```



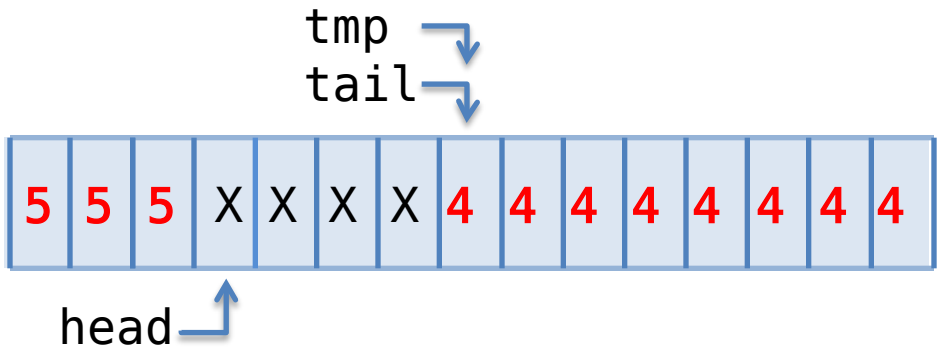
```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], gen(tmp), val) );
```



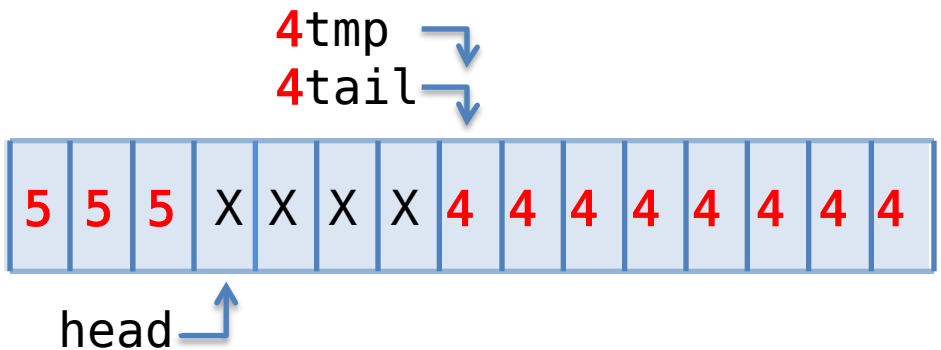
```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], gen(tmp), val) );
```

tmp

```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], gen(tmp), val) );
```

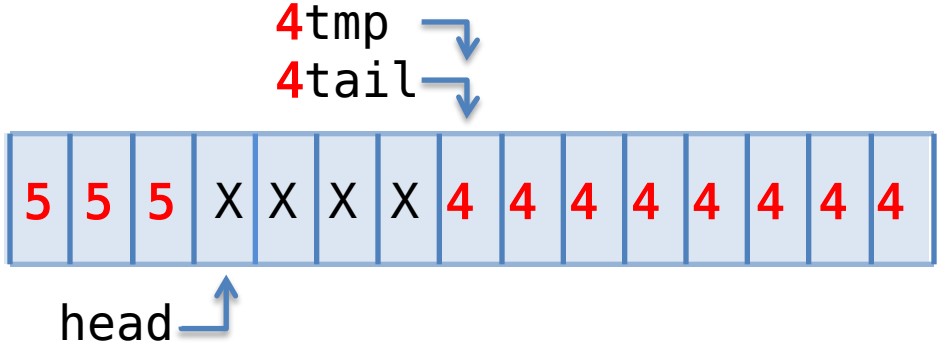


```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], gen(tmp), val) );
```



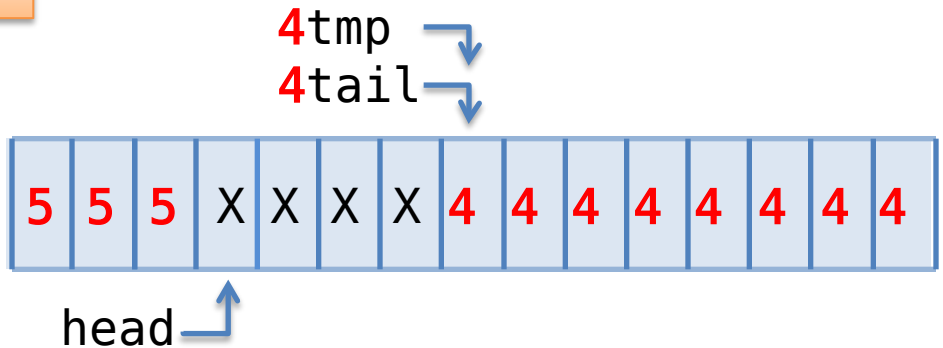
```
class index {
    size_t value;  // gen | idx
    size_t generation();
    operator size_t();
    index& operator++();
    //etc
};
```

```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], gen(tmp), val) );
```

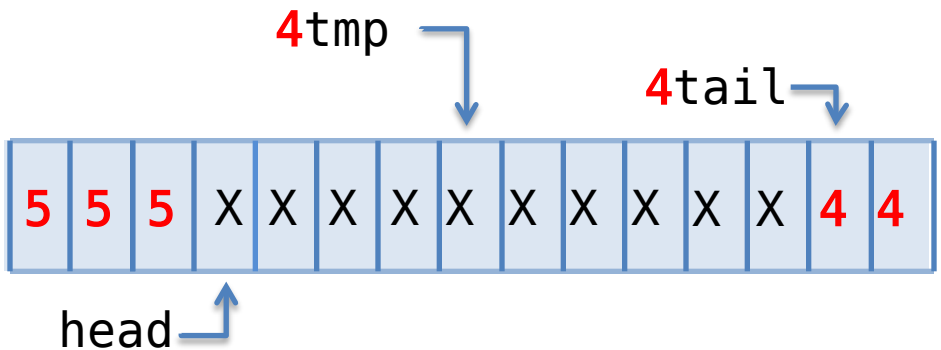


```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], gen(tmp), val) );
```

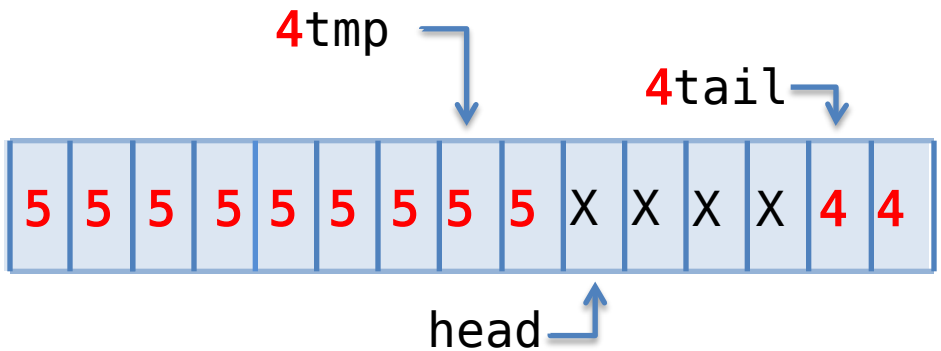
snapshot



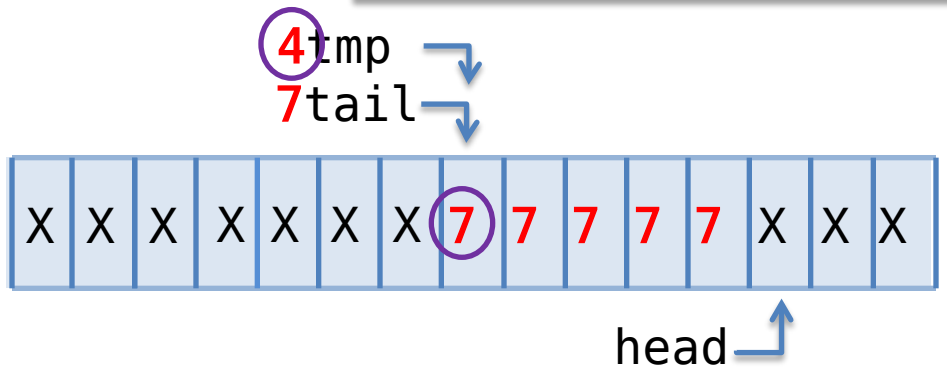
```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], gen(tmp), val) );
```



```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], gen(tmp), val) );
```

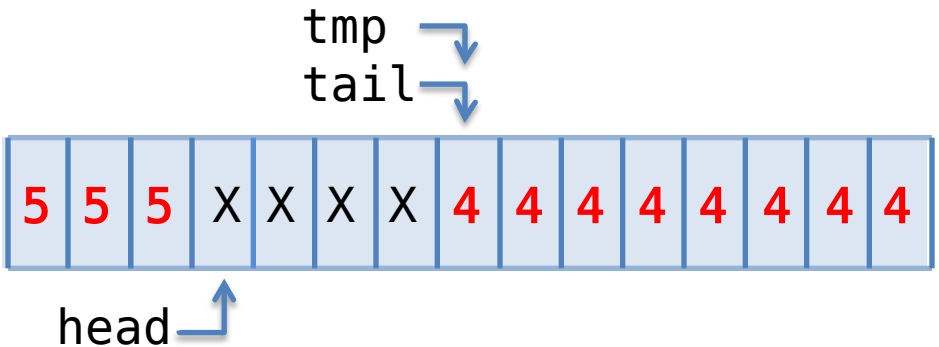


```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], gen(tmp), val) );
```

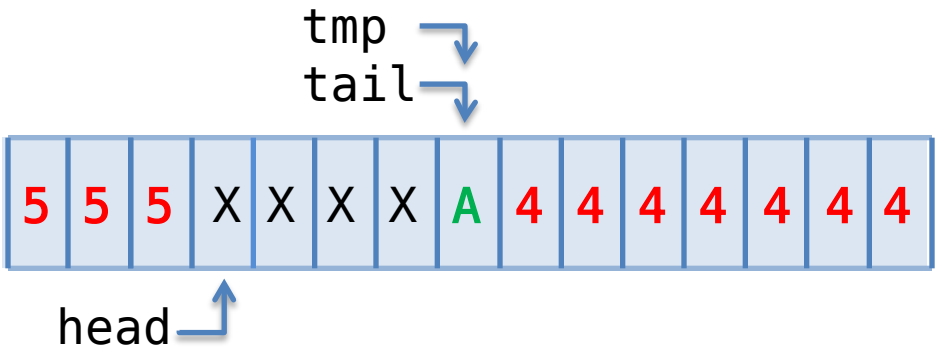


All states are valid states for all lines of code (*)

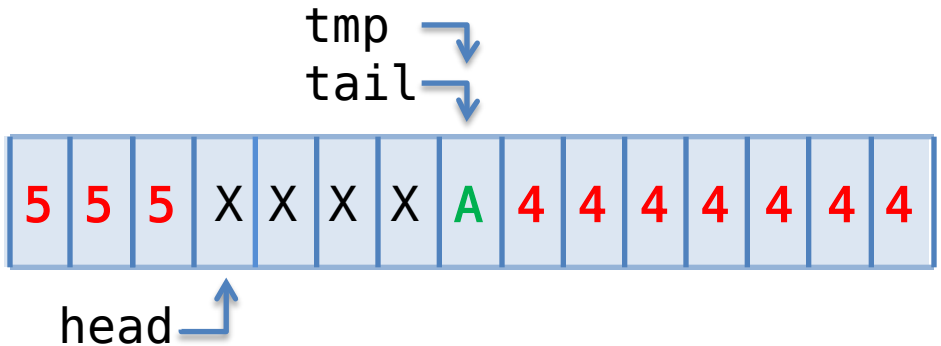
```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], gen(tmp), val) );
```



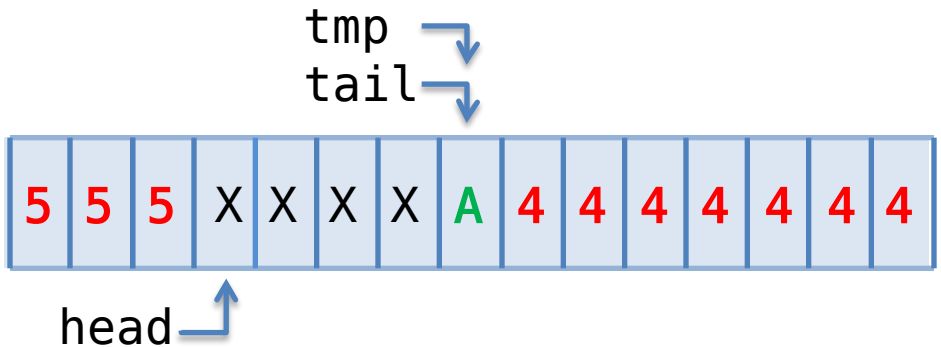
```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], gen(tmp), val) );
```



```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], gen(tmp), val) );
tail++; //???
```

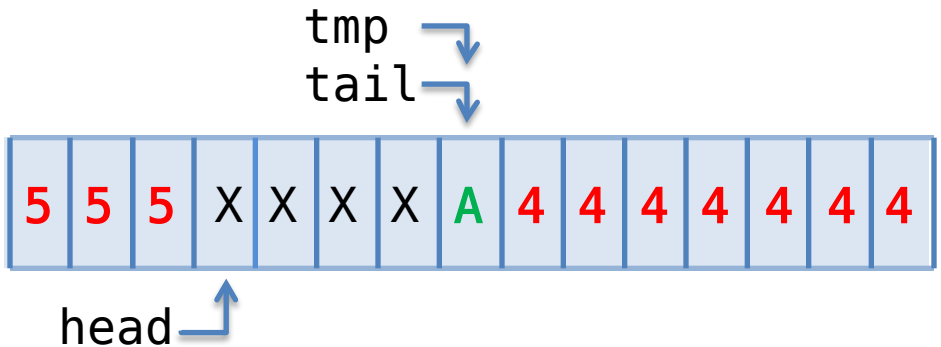


```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], gen(tmp), val) );
tail++; //???
```

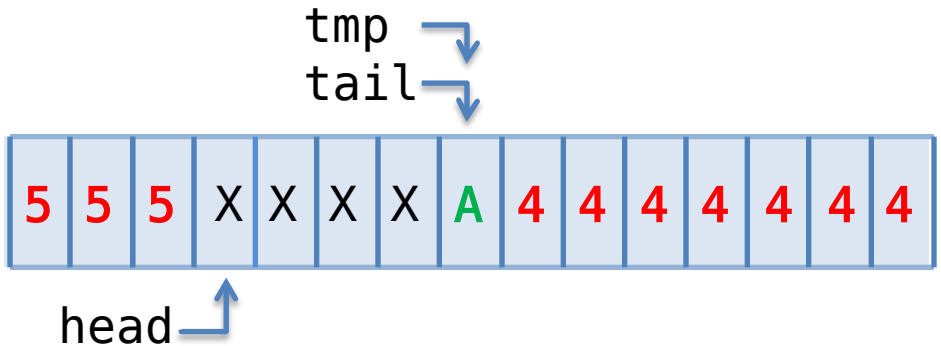


spinlock ?

```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], gen(tmp), val) );
tail++; //???
```

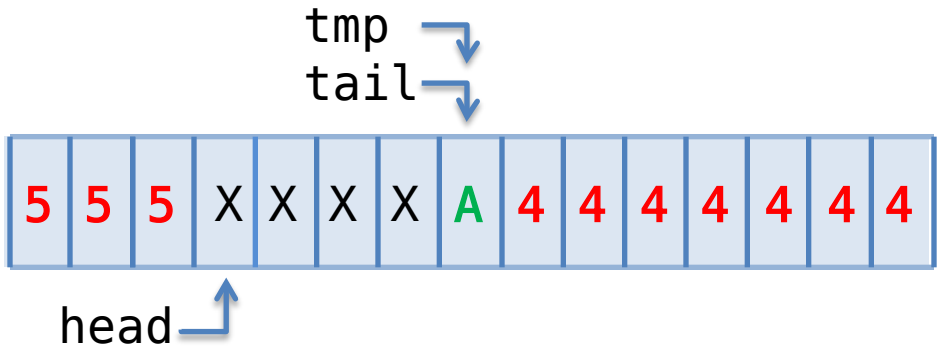


```
do {
        tmp = tail.load();
        while (buffer[tmp] != gen(tmp))
                tmp++;
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
tail++; // yes!
```



```
do {
        tmp = tail.load();
        while (buffer[tmp] != gen(tmp))
                tmp++;
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
tail++; // yes!
```

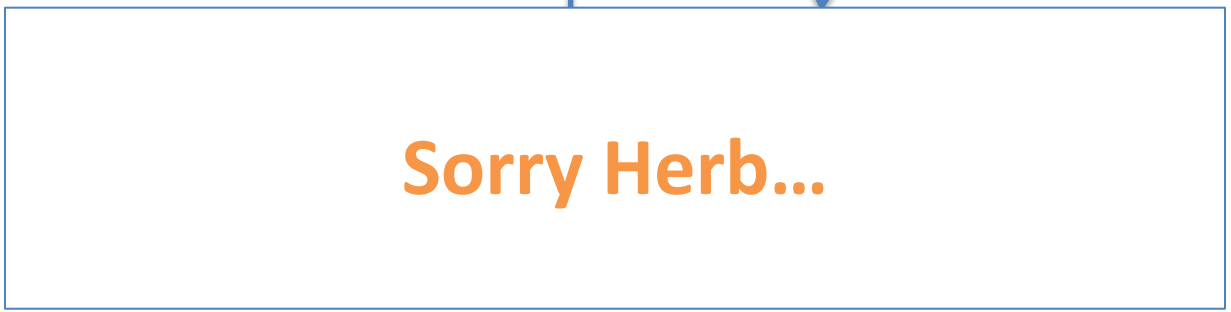
same



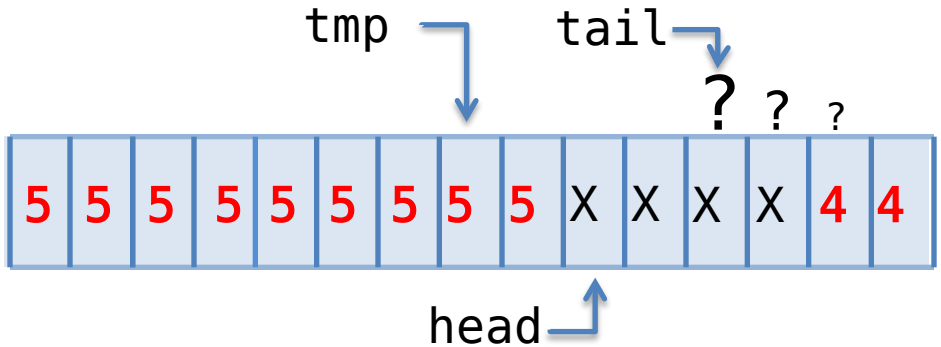
```
do {
        tmp = tail.load();
        while (buffer[tmp] != gen(tmp))
                tmp++;
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
tail++; // yes!
```

same

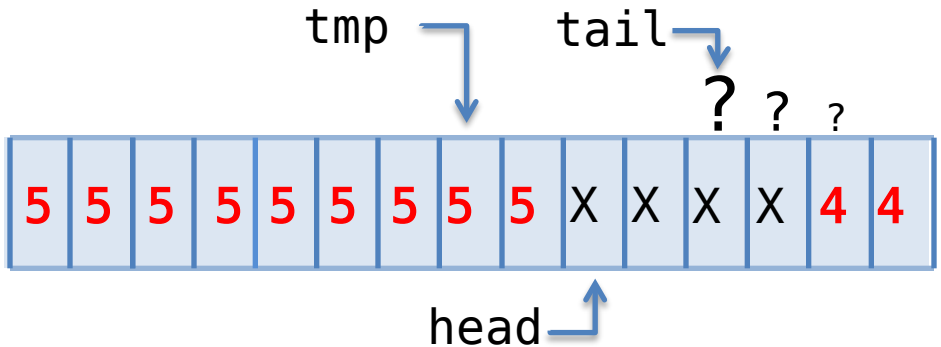
tmp tail



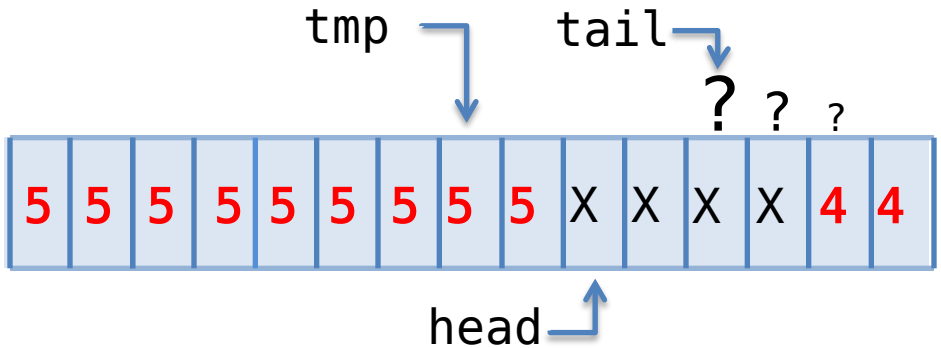
```
do {
        tmp = tail.load(memory_order_relaxed);
        while (buffer[tmp].load(relaxed) != gen(tmp))
                tmp++;
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
tail++; // yes!
```



```
do {
        tmp = tail.load(memory_order_relaxed);
        while (buffer[tmp].load(relaxed) != gen(tmp))
                tmp++;
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
tail = tmp + 1;
```

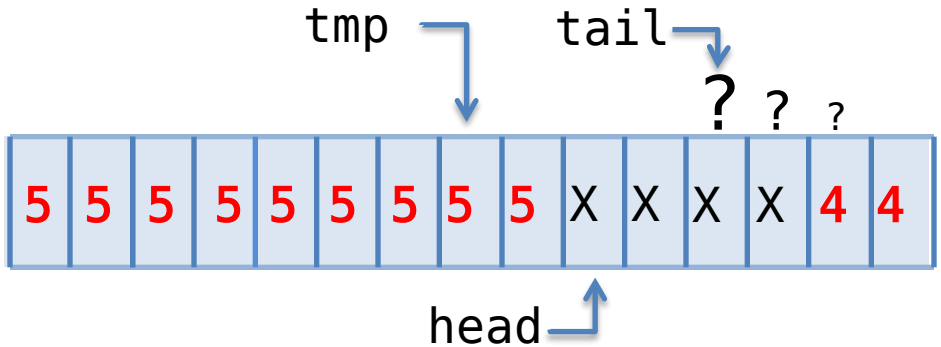


```
do {
        tmp = oldtail = tail.load(relaxed);
        while (buffer[tmp].load(relaxed) != gen(tmp))
                tmp++;
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
CAS(tail, oldtail, tmp+1);
```



```
do {
        tmp = oldtail = tail.load(relaxed);
        while (buffer[tmp].load(relaxed) != gen(tmp))
                tmp++;
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
CAS(tail, oldtail, tmp+1);
```

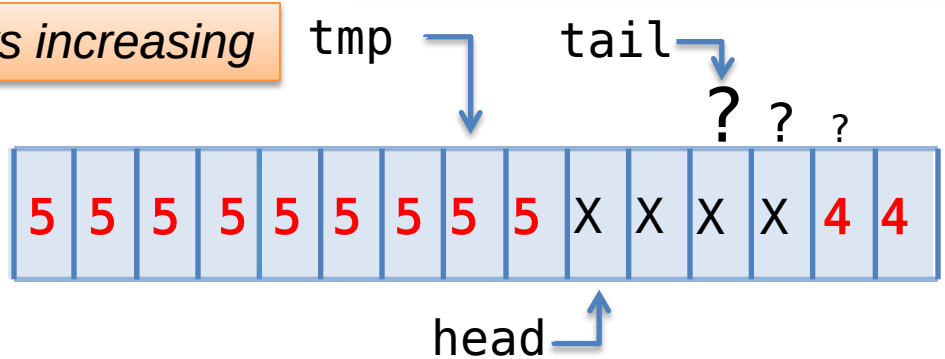
ABA?



```
do {
        tmp = oldtail = tail.load(relaxed);
        while (buffer[tmp].load(relaxed) != gen(tmp))
                tmp++;
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
CAS(tail, oldtail, tmp+1);
```

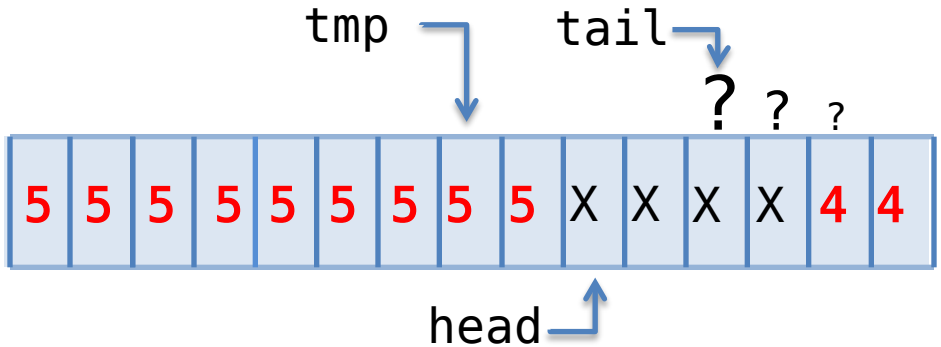
ABA?

ensure tail is always increasing



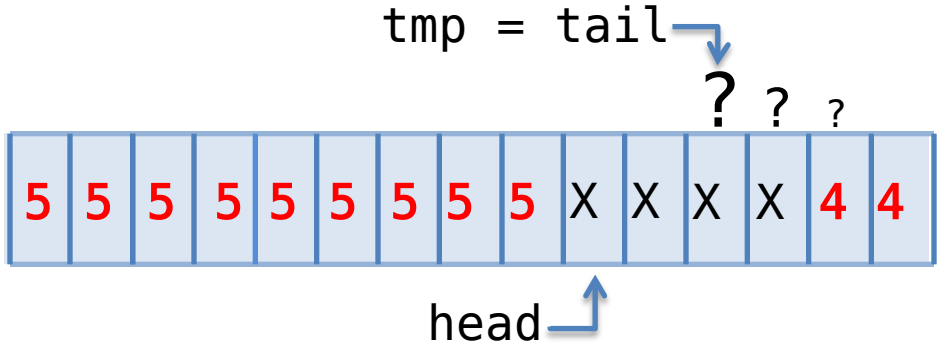
```
do {
        tmp = oldtail = tail.load(relaxed);
        while (buffer[tmp].load(relaxed) != gen(tmp))
                tmp++;
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
CAS(tail, oldtail, tmp+1);
```

Is tail up to date “now”?



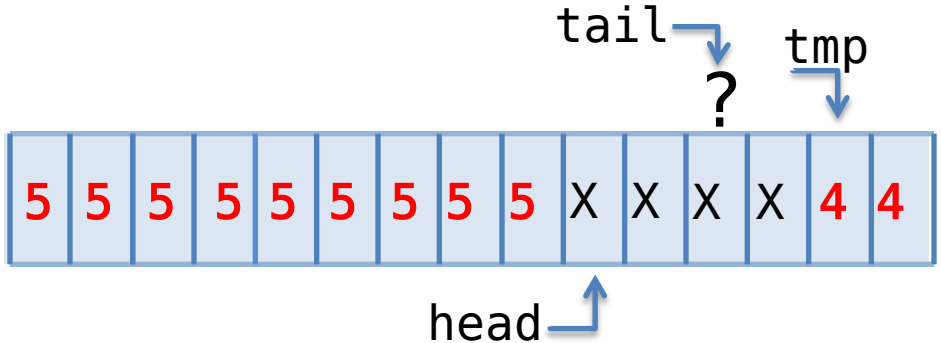
```
do {
      tmp = oldtail = tail.load(relaxed);
      while (buffer[tmp].load(relaxed) != gen(tmp))
            tmp++;
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
CAS(tail, oldtail, tmp+1);
```

Is tail up to date “now”?



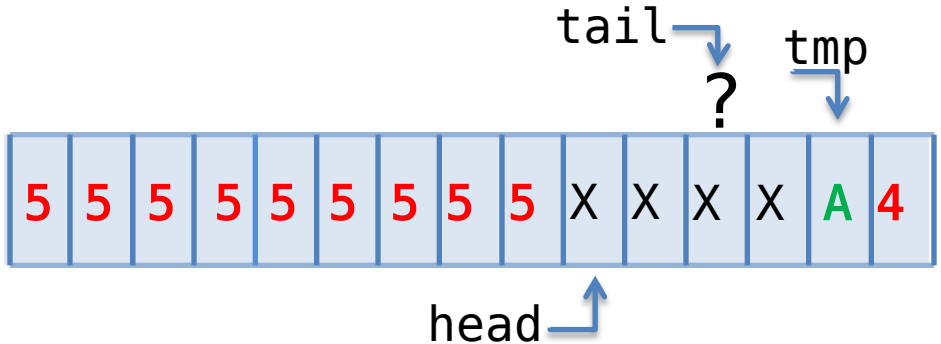
```
do {
        tmp = oldtail = tail.load(relaxed);
        while (buffer[tmp].load(relaxed) != gen(tmp))
                tmp++;
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
CAS(tail, oldtail, tmp+1);
```

Is tail up to date “now”?



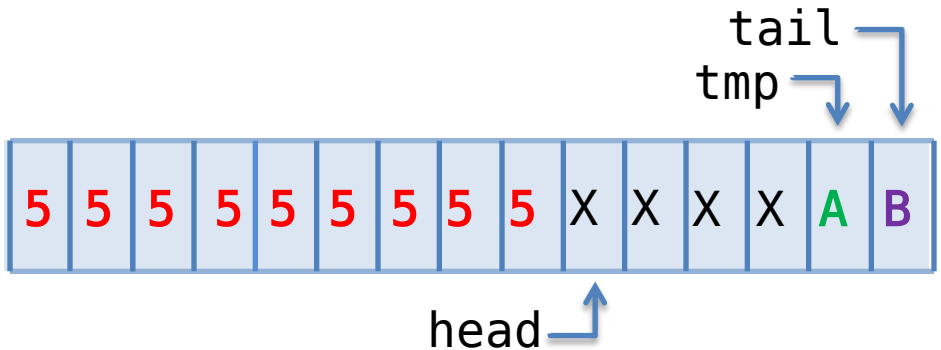
```
do {
        tmp = oldtail = tail.load(relaxed);
        while (buffer[tmp].load(relaxed) != gen(tmp))
                tmp++;
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
CAS(tail, oldtail, tmp+1);
```

Is tail up to date “now”?



```
do {
        tmp = oldtail = tail.load(relaxed);
        while (buffer[tmp].load(relaxed) != gen(tmp))
                tmp++;
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
CAS(tail, oldtail, tmp+1);
```

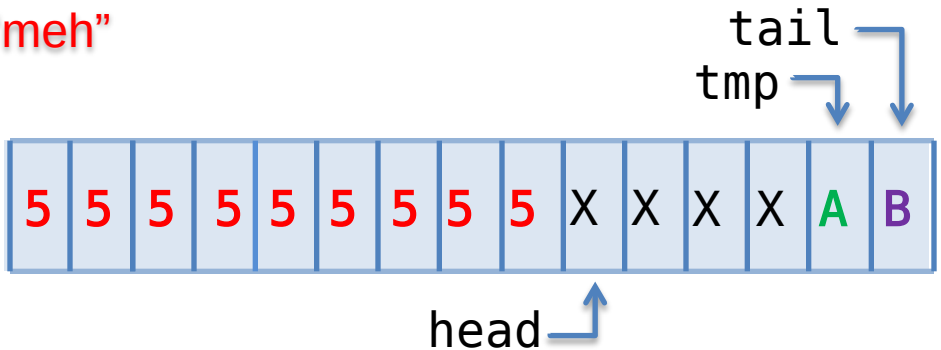
Is tail up to date “now”?



```
do {
        tmp = oldtail = tail.load(relaxed);
        while (buffer[tmp].load(relaxed) != gen(tmp))
                tmp++;
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
CAS(tail, oldtail, tmp+1);
```

Is tail up to date “now”?

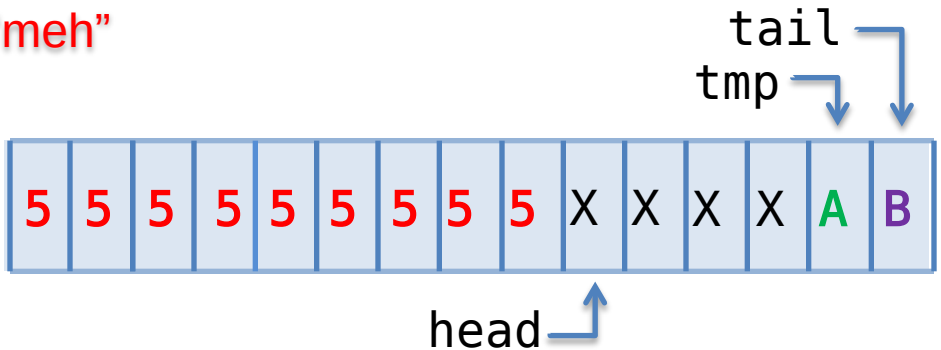
“meh”



```
do {
        tmp = oldtail = tail.load(relaxed);
        while (buffer[tmp].load(relaxed) != gen(tmp))
                tmp++;
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
CAS(tail, oldtail, tmp+1, relaxed);
```

Is tail up to date “now”?

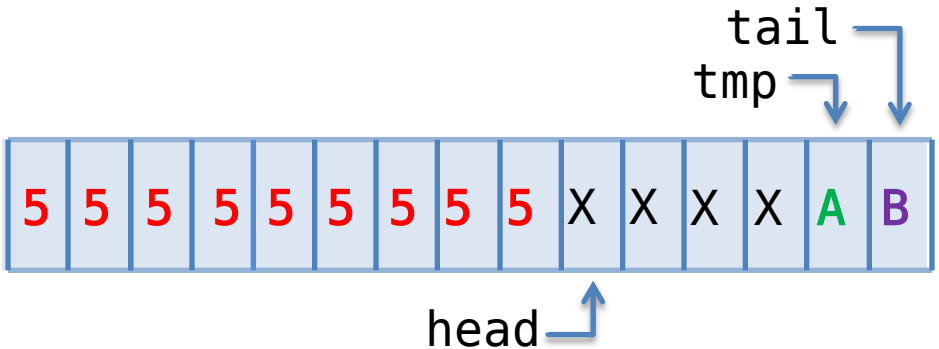
“meh”



push(val)

?

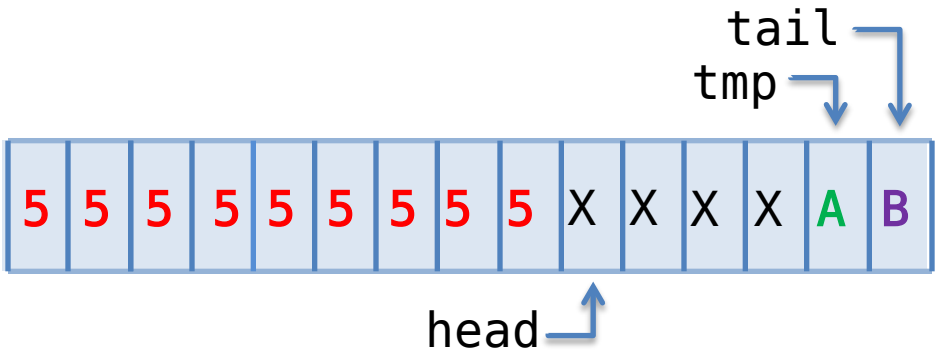
```
do {
        tmp = oldtail = tail.load(relaxed);
        while (buffer[tmp].load(relaxed) != gen(tmp))
                tmp++;
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
CAS(tail, oldtail, tmp+1);
```



push(val)

?

```
do {
        tmp = oldtail = tail.load(relaxed);
        while (buffer[tmp].load(relaxed) != gen(tmp))
                tmp++;
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
CAS(tail, oldtail, tmp+1);
```

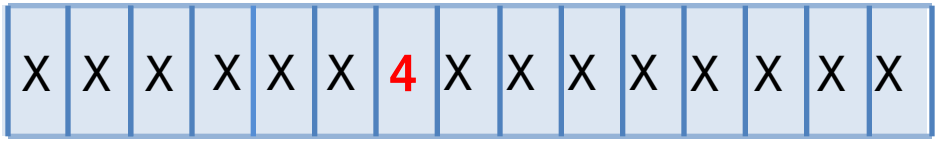


All states are valid states for all lines of code?

push(val)

?

```
do {
        tmp = oldtail = tail.load(relaxed);
        while (buffer[tmp].load(relaxed) != gen(tmp))
                tmp++;
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
CAS(tail, oldtail, tmp+1);
```

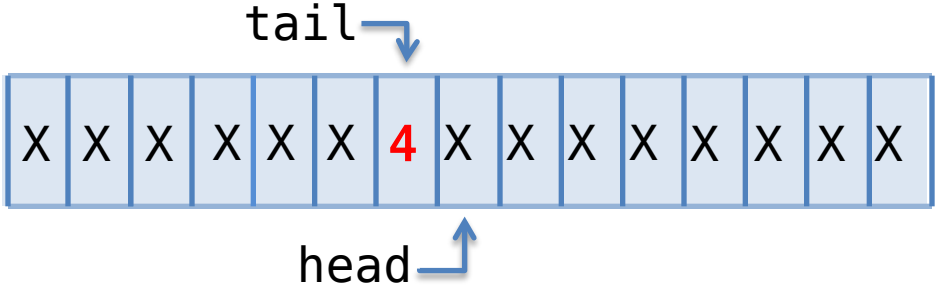


All states are valid states for all lines of code?

push(val)

?

```
do {
        tmp = oldtail = tail.load(relaxed);
        while (buffer[tmp].load(relaxed) != gen(tmp))
                tmp++;
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
CAS(tail, oldtail, tmp+1);
```

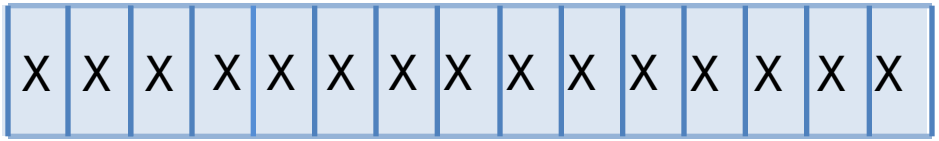


All states are valid states for all lines of code?

push(val)

?

```
do {
        tmp = oldtail = tail.load(relaxed);
        while (buffer[tmp].load(relaxed) != gen(tmp))
                tmp++;
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
CAS(tail, oldtail, tmp+1);
```

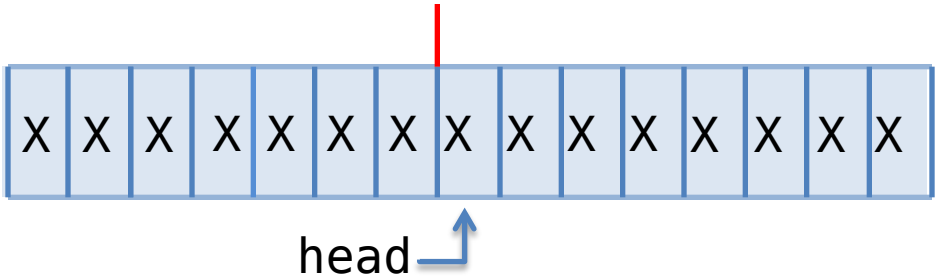


All states are valid states for all lines of code?

push(val)

?

```
do {
        tmp = oldtail = tail.load(relaxed);
        while (buffer[tmp].load(relaxed) != gen(tmp))
                tmp++;
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
CAS(tail, oldtail, tmp+1);
```

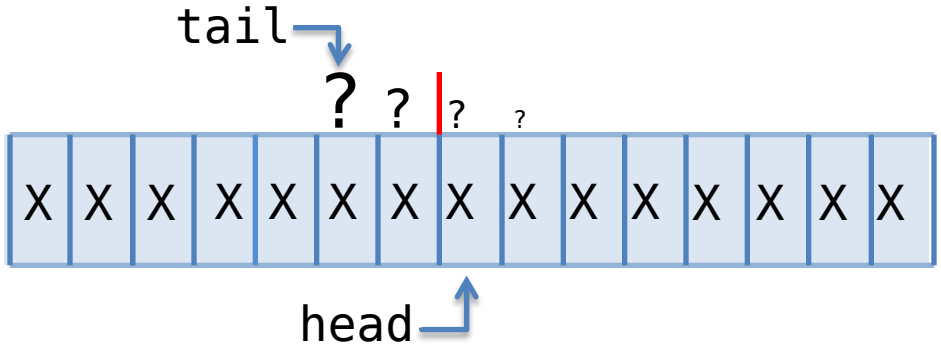


All states are valid states for all lines of code?

push(val)

?

```
do {
        tmp = oldtail = tail.load(relaxed);
        while (buffer[tmp].load(relaxed) != gen(tmp))
                tmp++;
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
CAS(tail, oldtail, tmp+1);
```

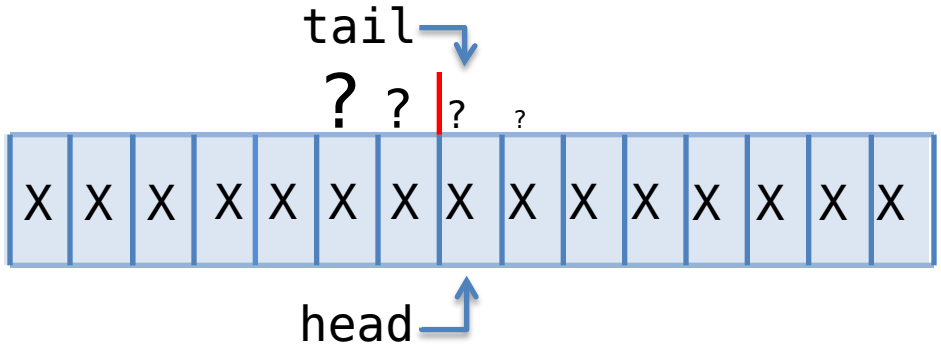


All states are valid states for all lines of code?

push(val)

?

```
do {
        tmp = oldtail = tail.load(relaxed);
        while (buffer[tmp].load(relaxed) != gen(tmp))
                tmp++;
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
CAS(tail, oldtail, tmp+1);
```

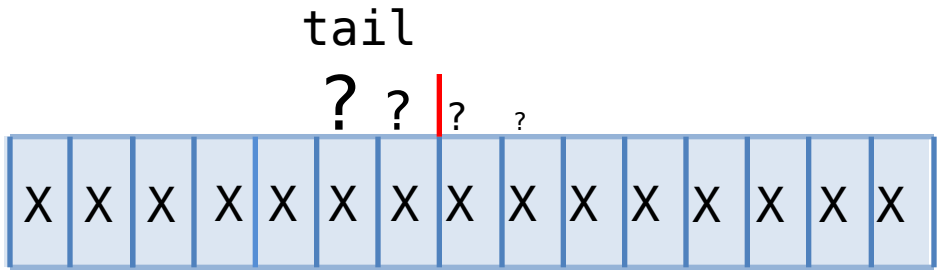


All states are valid states for all lines of code?

push(val)

?

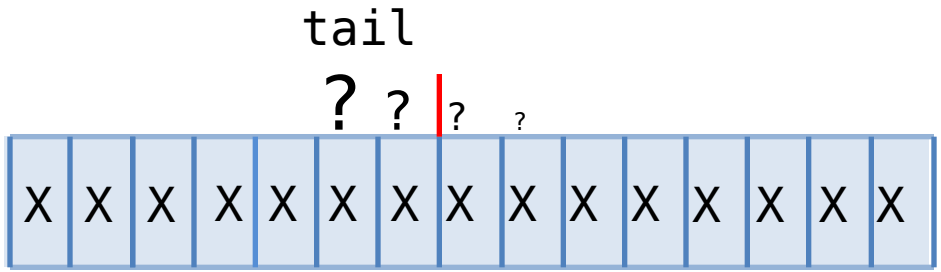
```
do {
        tmp = oldtail = tail.load(relaxed);
        while (buffer[tmp].load(relaxed) != gen(tmp))
                tmp++;
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
CAS(tail, oldtail, tmp+1);
```



All states are valid states for all lines of code?

(worse?) spinlock ?

```
do {
        tmp = oldtail = tail.load(relaxed);
        while (buffer[tmp].load(relaxed) != gen(tmp))
                tmp++;
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
CAS(tail, oldtail, tmp+1);
```



All states are valid states for all lines of code?

(worse?) spinlock ?

```
do {
        tmp = oldtail = tail.load(relaxed);
        while (buffer[tmp].load(relaxed) != gen(tmp))
                tmp++;
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
CAS(tail, oldtail, tmp+1);
```

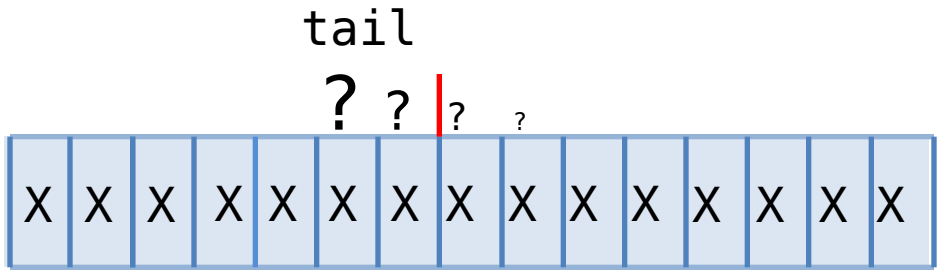
tail

Compromise…

All states are valid states for all lines of code?

(worse?) spinlock ?
overflow ?

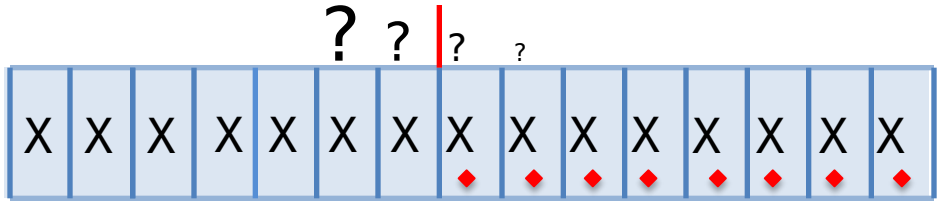
```
do {
        tmp = oldtail = tail.load(relaxed);
        while (buffer[tmp].load(relaxed) != gen(tmp))
                tmp++;
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
CAS(tail, oldtail, tmp+1);
```



All states are valid states for all lines of code?

```
do {
        tmp = oldtail = tail.load(relaxed);
        g = gen(tmp);
        while ((b = buffer[tmp]) != g && odd(g)==odd(b))
                tmp++;
 } while ( ! CAS(buffer[tmp], gen(tmp), val | odd(gen
CAS(tail, oldtail, tmp+1);
```

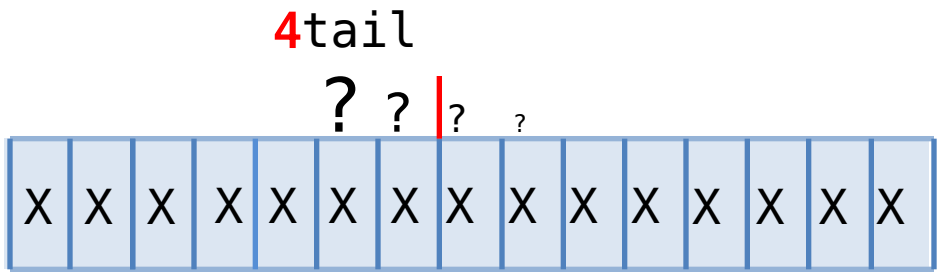
4tail



All states are valid states for all lines of code?

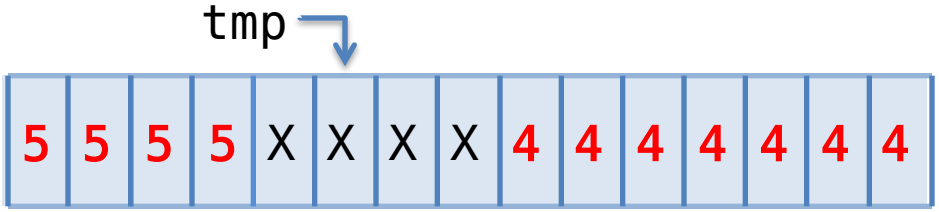
spinlock on full ?

```
do {
        tmp = oldtail = tail.load(relaxed);
        while (buffer[tmp] != gen(tmp) && tmp - oldtail < size)
                tmp++;
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
CAS(tail, oldtail, tmp+1);
```



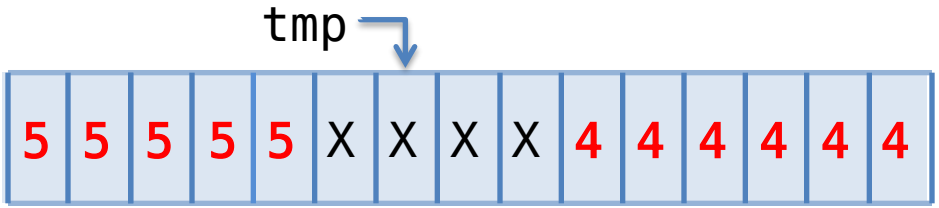
All states are valid states for all lines of code?

```
do {
        tmp = oldtail = tail.load(relaxed);
        while (buffer[tmp] != gen(tmp) && tmp - oldtail < size)
                tmp++;
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
CAS(tail, oldtail, tmp+1);
```



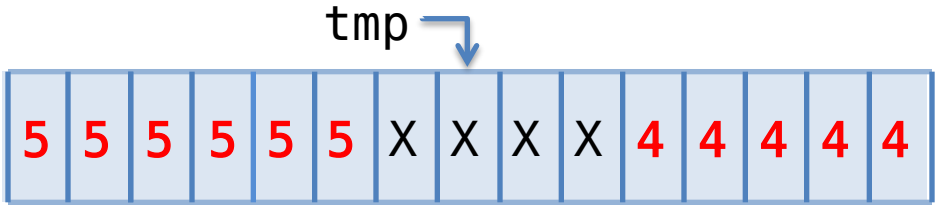
All states are valid states for all lines of code?

```
do {
        tmp = oldtail = tail.load(relaxed);
        while (buffer[tmp] != gen(tmp) && tmp - oldtail < size)
                tmp++;
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
CAS(tail, oldtail, tmp+1);
```



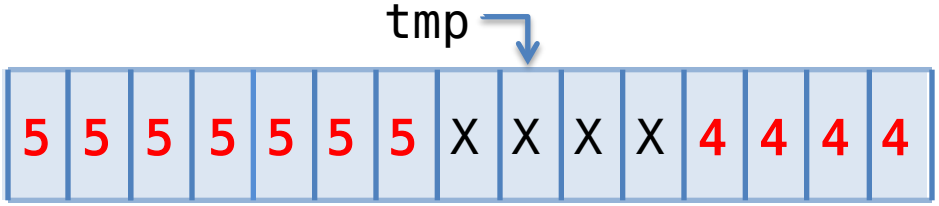
All states are valid states for all lines of code?

```
do {
        tmp = oldtail = tail.load(relaxed);
        while (buffer[tmp] != gen(tmp) && tmp - oldtail < size)
                tmp++;
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
CAS(tail, oldtail, tmp+1);
```



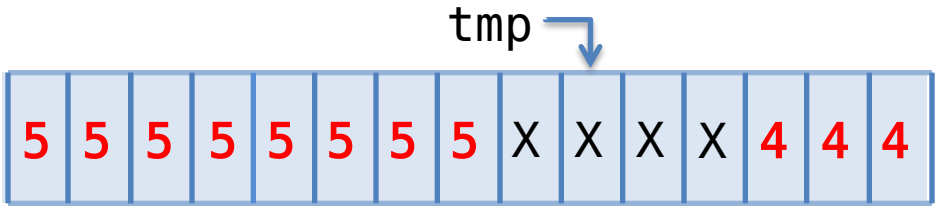
All states are valid states for all lines of code?

```
do {
        tmp = oldtail = tail.load(relaxed);
        while (buffer[tmp] != gen(tmp) && tmp - oldtail < size)
                tmp++;
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
CAS(tail, oldtail, tmp+1);
```



All states are valid states for all lines of code?

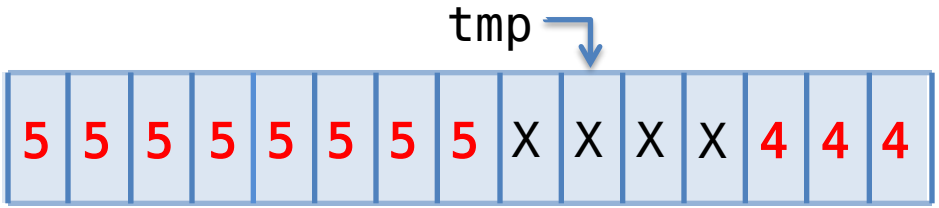
```
do {
        tmp = oldtail = tail.load(relaxed);
        while (buffer[tmp] != gen(tmp) && tmp - oldtail < size)
                tmp++;
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
CAS(tail, oldtail, tmp+1);
```



All states are valid states for all lines of code?

spinlock **NOT** full ?

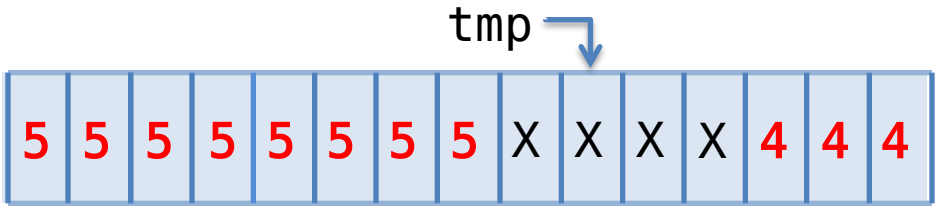
```
do {
        tmp = oldtail = tail.load(relaxed);
        while (buffer[tmp] != gen(tmp) && tmp - oldtail < size)
                tmp++;
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
CAS(tail, oldtail, tmp+1);
```



All states are valid states for all lines of code?

"fullish"

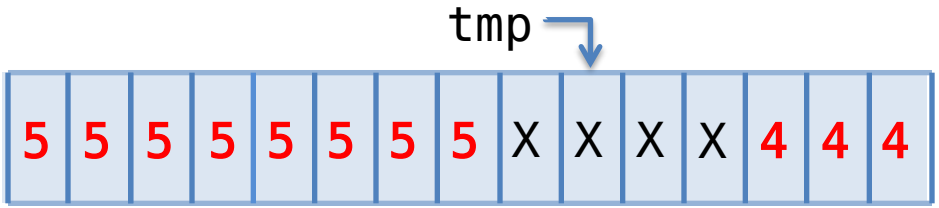
```
do {
        tmp = oldtail = tail.load(relaxed);
        tmp = find_tail(tmp, &oldtail);
        if (tmp == FULL) …???;
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
CAS(tail, oldtail, tmp+1);
```



All states are valid states for all lines of code?

```
if (some_state)
{
   // still some_state???
   ...
}
```

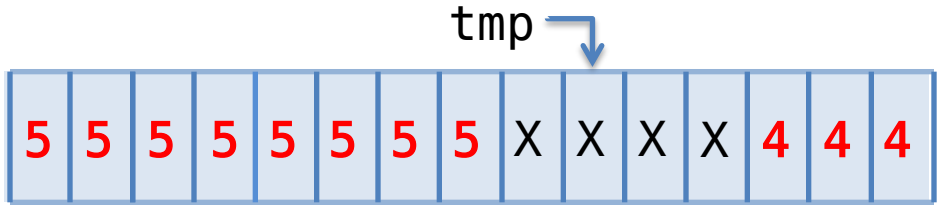
```
do {
      tmp = oldtail = tail.load(relaxed);
      tmp = find_tail(tmp, &oldtail);
      if (tmp == FULL) …???;
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
CAS(tail, oldtail, tmp+1);
```



All states are valid states for all lines of code?

```
if (full)
{
    // still full???
    // or now empty?
    ...
}
```

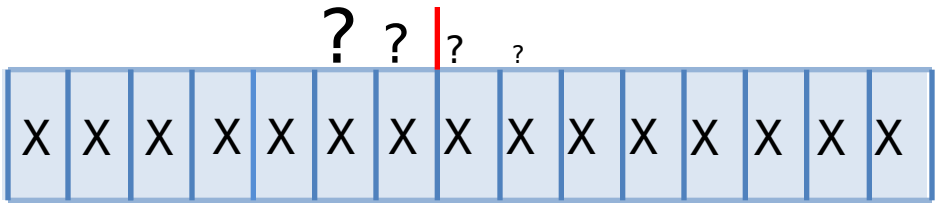
```
do {
        tmp = oldtail = tail.load(relaxed);
        tmp = find_tail(tmp, &oldtail);
        if (tmp == FULL) ...???;
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
CAS(tail, oldtail, tmp+1);
```



All states are valid states for all lines of code?

```
if (full)
{
    // still full???
    // or now empty?
    ...
}
```

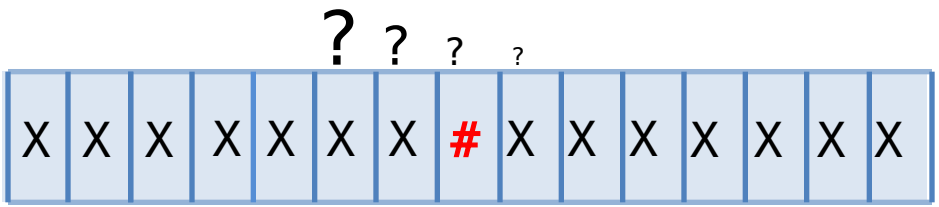
```
do {
        tmp = oldtail = tail.load(relaxed);
        tmp = find_tail(tmp, &oldtail);
        if (tmp == FULL) ...???;
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
CAS(tail, oldtail, tmp+1);
```



All states are valid states for all lines of code?

```
if (full)
{
    // still full???
    // or now empty?
    ...
}
```

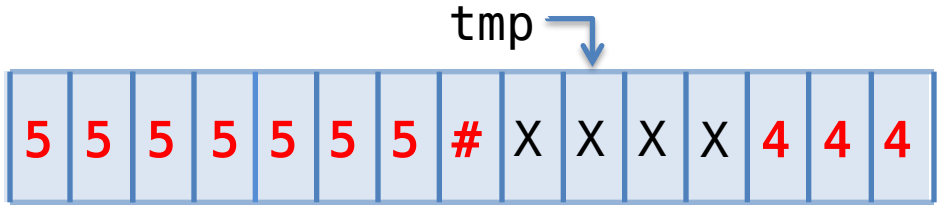
```
do {
        tmp = oldtail = tail.load(relaxed);
        tmp = find_tail(tmp, &oldtail);
        if (tmp == FULL) ...???;
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
CAS(tail, oldtail, tmp+1);
```



All states are valid states for all lines of code?

```
if (full)
{
    // still full???
    // or now empty?
    ...
}
```

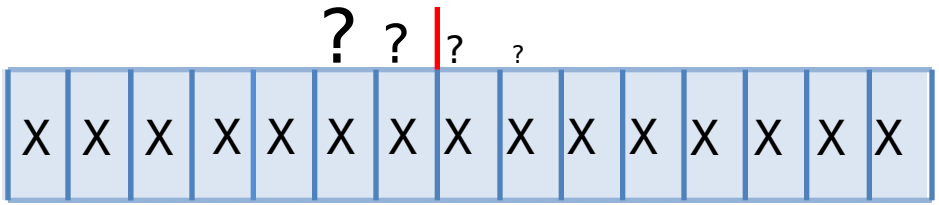
```
do {
        tmp = oldtail = tail.load(relaxed);
        tmp = find_tail(tmp, &oldtail);
        if (tmp == FULL) ...???;
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
CAS(tail, oldtail, tmp+1);
```



All states are valid states for all lines of code?

```
if (full)
{
    // still full???
    // or now empty?
    ...
}
```

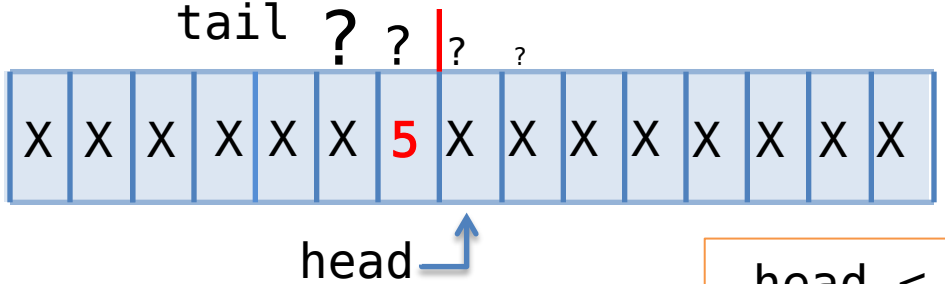
```
do {
        tmp = oldtail = tail.load(relaxed);
        tmp = find_tail(tmp, &oldtail);
        if (tmp == FULL) ...???;
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
CAS(tail, oldtail, tmp+1);
```



All states are valid states for all lines of code?

```
if (full)
{
    // still full???
    // or now empty?
    ...
}
```

```
do {
        tmp = oldtail = tail.load(relaxed);
        tmp = find_tail(tmp, &oldtail);
        if (tmp == FULL) ...???;
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
CAS(tail, oldtail, tmp+1);
```

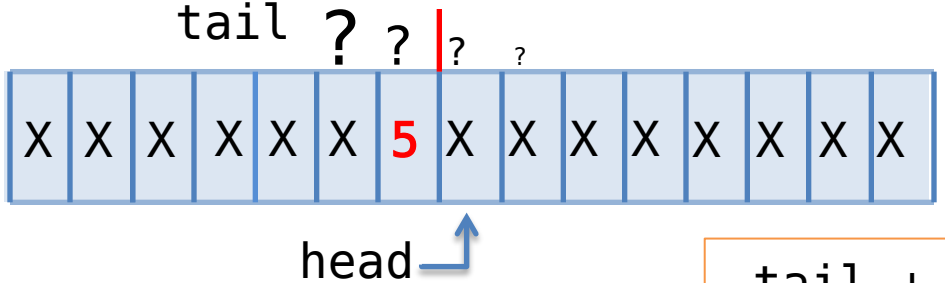


head < tail ?

All states are valid states for all lines of code?

```
if (full)
{
    // still full???
    // or now empty?
    ...
}
```

```
do {
        tmp = oldtail = tail.load(relaxed);
        tmp = find_tail(tmp, &oldtail);
        if (tmp == FULL) ...???;
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
CAS(tail, oldtail, tmp+1);
```

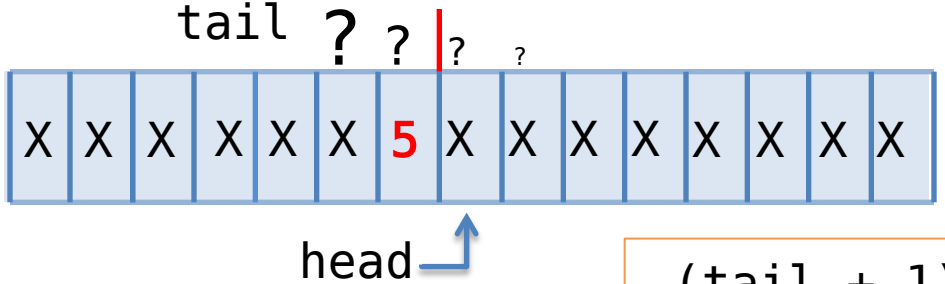



```
tail + 1 != head?
```

All states are valid states for all lines of code?

```
if (full)
{
    // still full???
    // or now empty?
    ...
}
```

```
do {
        tmp = oldtail = tail.load(relaxed);
        tmp = find_tail(tmp, &oldtail);
        if (tmp == FULL) ...???;
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
CAS(tail, oldtail, tmp+1);
```

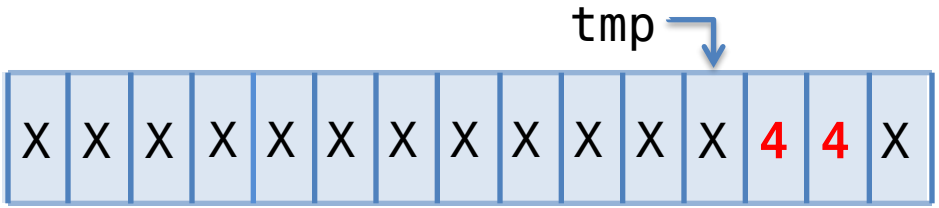


All states are valid states for all lines of code?

```
(tail + 1)%SZ != head%SZ ?
```

"fullish"

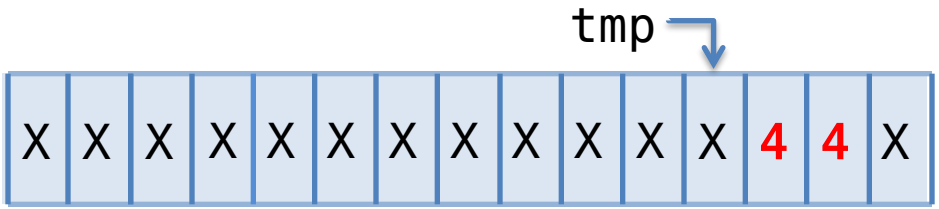
```
do {
        tmp = oldtail = tail.load(relaxed);
        tmp = find_tail(tmp, &oldtail);
        if (tmp == FULL) …???;
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
CAS(tail, oldtail, tmp+1);
```



All states are valid states for all lines of code?

"fullish"

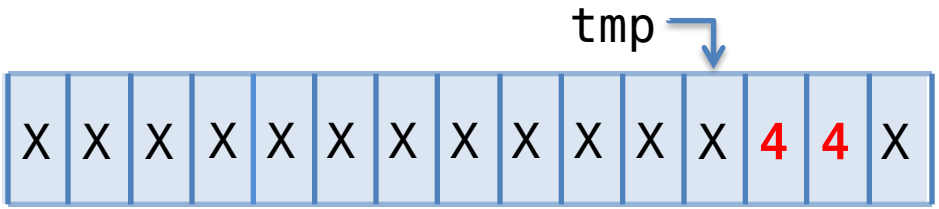
```
do {
        tmp = oldtail = tail.load(relaxed);
        tmp = find_tail(tmp, &oldtail);
        if (tmp == FULL) wait_for_space();
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
CAS(tail, oldtail, tmp+1);
```



All states are valid states for all lines of code?

“fullish” →

```
do {
        tmp = oldtail = tail.load(relaxed);
        tmp = find_tail(tmp, &oldtail);
        if (tmp == FULL) { wait_for_space(); continue;}
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
CAS(tail, oldtail, tmp+1);
```

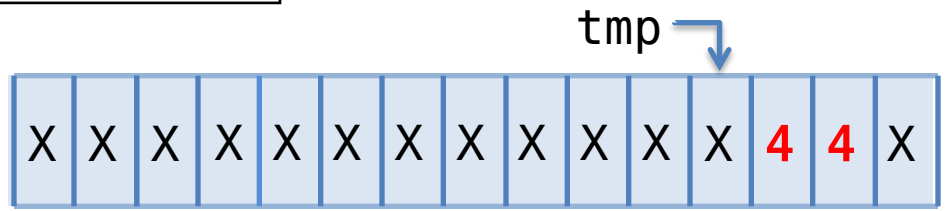


All states are valid states for all lines of code?

```
wait_for_space()
{
  unique_lock lock(mutex);

  while (still_fullish())
     cond_full.wait(lock);
}
```

```
do {
      tmp = oldtail = tail.load(relaxed);
      tmp = find_tail(tmp, &oldtail);
      if (tmp == FULL) { wait_for_space(); continue;}
 } while ( ! CAS(buffer[tmp], gen(tmp), va) );
CAS(tail, oldtail, tmp+1);
```



All states are valid states for all lines of code?

Lock-free by Example

(one very complicated example)

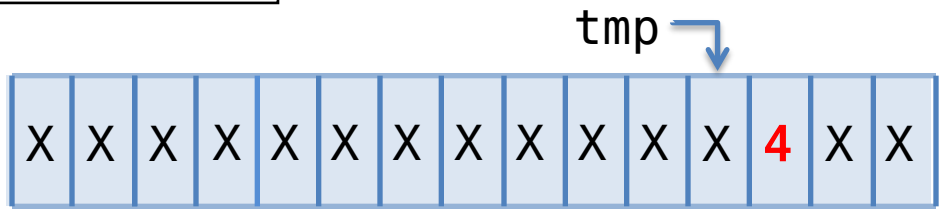
Tony Van Eerd

C++Now, 2015

```
{
  unique_lock lock(mutex);

  while (still_fullish())
     cond_full.wait(lock);
}
```

```
do {
      tmp = oldtail = tail.load(relaxed);
      tmp = find_tail(tmp, &oldtail);
      if (tmp == FULL) { wait_for_space(); continue;}
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
CAS(tail, oldtail, tmp+1);
```

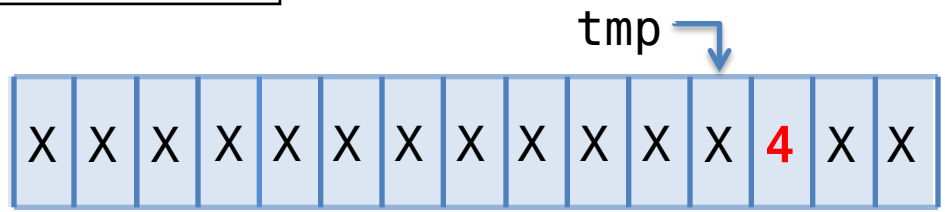


All states are valid states for all lines of code?

```
{
  unique_lock lock(mutex);

  while ( ! …find_tail… )
     cond_full.wait(lock);
}
```

```
do {
      tmp = oldtail = tail.load(relaxed);
      tmp = find_tail(tmp, &oldtail);
      if (tmp == FULL) { wait_for_space(); continue;}
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
CAS(tail, oldtail, tmp+1);
```

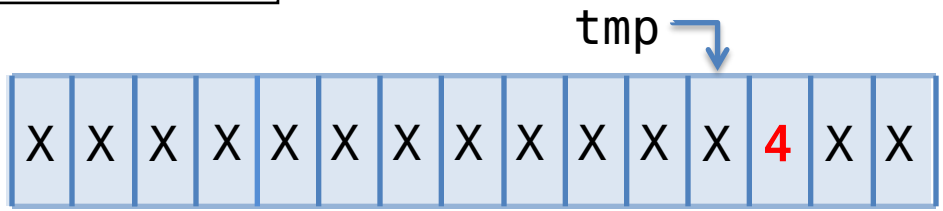


All states are valid states for all lines of code?

```
{
  unique_lock lock(mutex);

  while ( ! …find_tail… )
     cond_full.wait(lock);
}
```

```
do {
      tmp = oldtail = tail.load(relaxed);
      tmp = find_tail(tmp, &oldtail);
      if(tmp == FULL)wait_for_space(&tmp,&oldtail);
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
CAS(tail, oldtail, tmp+1);
```



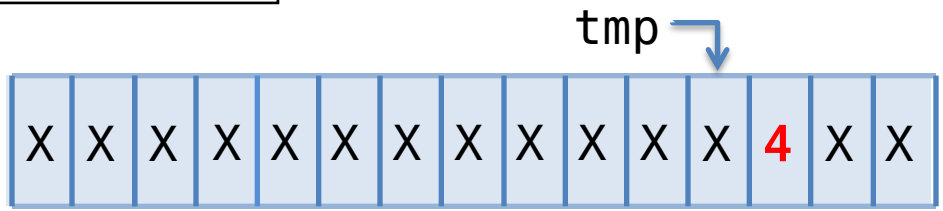
All states are valid states for all lines of code?

```
{
  unique_lock lock(mutex);

  while ( ! …find_tail… )
     cond_full.wait(lock);
}
```

?

who calls notify() “queue not full“ ?

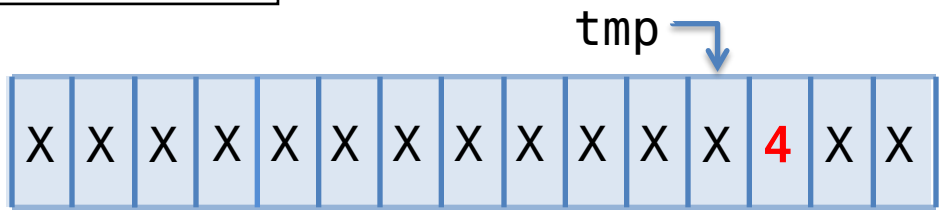


```
{
  unique_lock lock(mutex);

  while ( ! …find_tail… )
     cond_full.wait(lock);
}
```

?

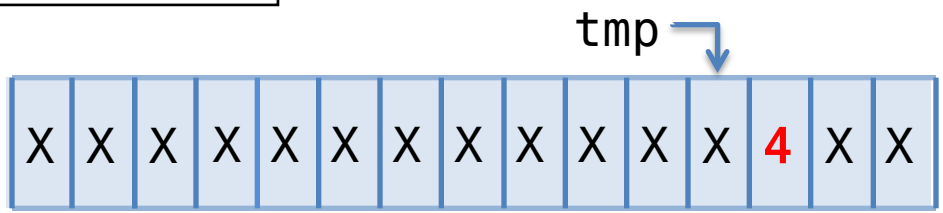
who calls notify() “queue not full“ ?




```
int pop() {
  ...
  cond_full.notify();
}
```

```
{
  unique_lock lock(mutex);

  while ( ! …find_tail… )
     cond_full.wait(lock);
}
```



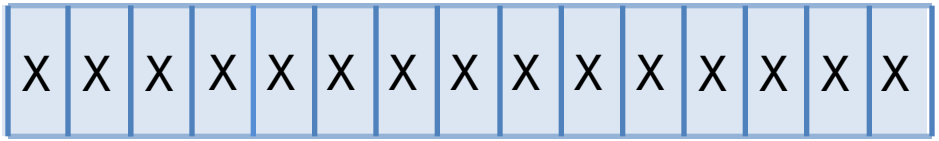

```
int pop() {
  ...
  cond_full.notify();
}
```

?

```
int pop() {
  ...
  unique_lock lock(mutex);
  cond_full.notify();
}
```

```
{
  unique_lock lock(mutex);

  while ( ! …find_tail… )
      cond_full.wait(lock);
}
```




```
int pop() {
  ...
  cond_full.notify();
}
```

?

```
int pop() {
  ...
  unique_lock lock(mutex);
  cond_full.notify();
}
```

```
{
  unique_lock lock(mutex);

  while ( ! …find_tail… )
      cond_full.wait(lock);
}
```


```
int pop() {
  ...
  cond_full.notify();
}
```

?

```
int pop() {
  ...
  unique_lock lock(mutex);
  cond_full.notify();
}
```

Lock-free by Example

(one very complicated example)

Tony Van Eerd

C++Now, 2015

```
{
  unique_lock lock(mutex);

  while ( ! …find_tail… )
      cond_full.wait(lock);
}
```


```
int pop() {
  ...
  cond_full.notify();
}
```

?

```
int pop() {
  ...
  unique_lock lock(mutex);
  cond_full.notify();
}
```

```
{
  unique_lock lock(mutex);

  while ( ! …find_tail… )
      cond_full.wait(lock);
}
```


```
int pop() {
  ...
  cond_full.notify();
}
```

?

```
int pop() {
  ...
  unique_lock lock(mutex);
  cond_full.notify();
}
```

```
waiting = true;
{
  unique_lock lock(mutex);

  while ( ! …find_tail… )
     cond_full.wait(lock);
}
```

I'm waiting!


```
int pop() {
  ...
  cond_full.notify();
}
```

?

```
int pop() {
  ...
  unique_lock lock(mutex);
  cond_full.notify();
}
```

```
waiting = true;
{
  unique_lock lock(mutex);

  while ( ! …find_tail… )
     cond_full.wait(lock);
}
```

I'm waiting!

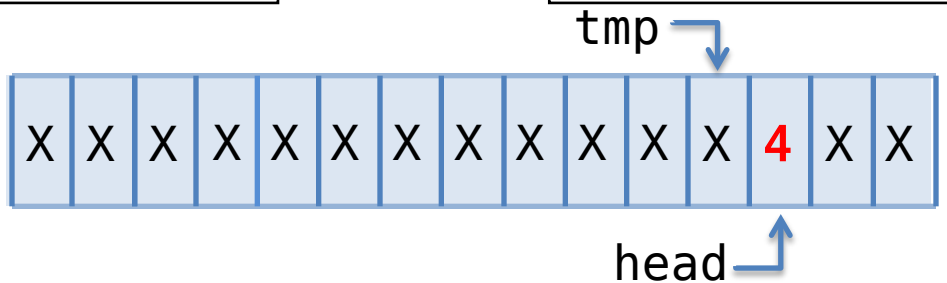
```
int pop() {
  ...
  unique_lock lock(mutex);
  cond_full.notify();
}
```

```
waiting = true;
{
  unique_lock lock(mutex);

  while ( ! …find_tail… )
     cond_full.wait(lock);
}
```

I'm waiting!

```
int pop() {
  ...
  if (waiting) {
    unique_lock lock(mutex);
    cond_full.notify();
  }
}
```

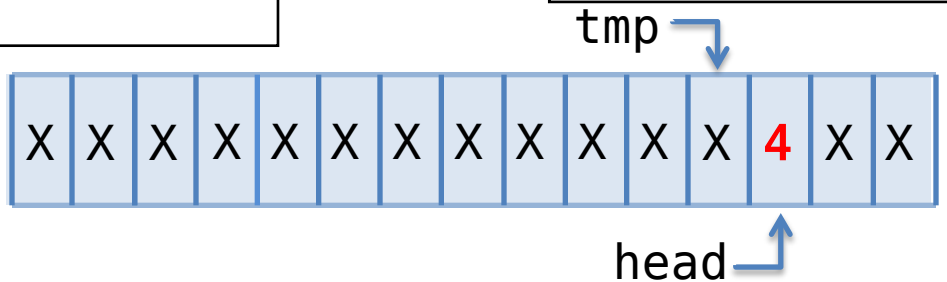


```
waiting = true;
{
  unique_lock lock(mutex);

  while ( ! …find_tail… )
     cond_full.wait(lock);
}
waiting = false;
```

I'm waiting!

```
int pop() {
  ...
  if (waiting) {
    unique_lock lock(mutex);
    cond_full.notify();
  }
}
```

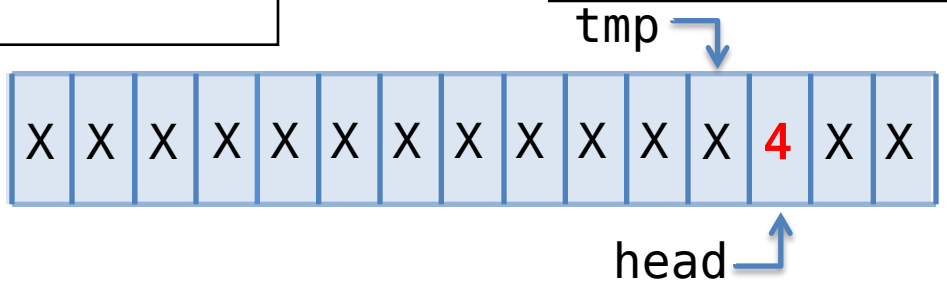


```
waiting++;
{
  unique_lock lock(mutex);

  while ( ! …find_tail… )
     cond_full.wait(lock);
}
waiting--;
```

I'm waiting!

```
int pop() {
  ...
  if (waiting) {
    unique_lock lock(mutex);
    cond_full.notify();
  }
}
```

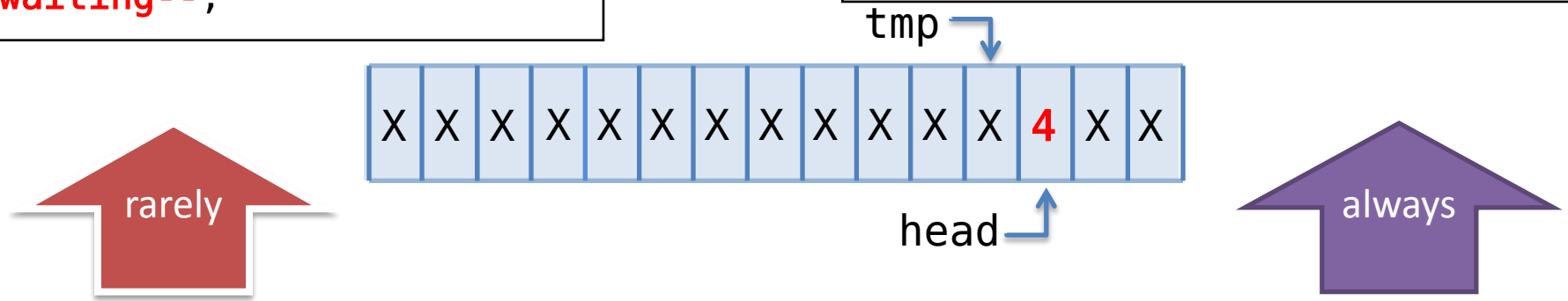


```
waiting++;
{
  unique_lock lock(mutex);

  while ( ! …find_tail… )
     cond_full.wait(lock);
}
waiting--;
```

I'm waiting!

```
int pop() {
  ...
  if (waiting) {
    unique_lock lock(mutex);
    cond_full.notify();
  }
}
```

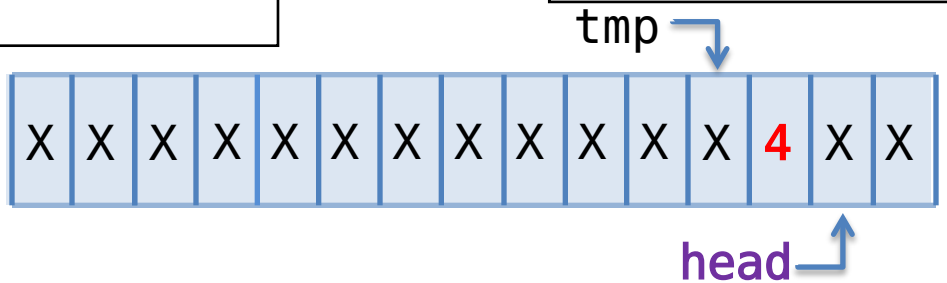


```
waiting++;
{
  unique_lock lock(mutex);

  while ( ! …find_tail… )
     cond_full.wait(lock);
}
waiting--;
```

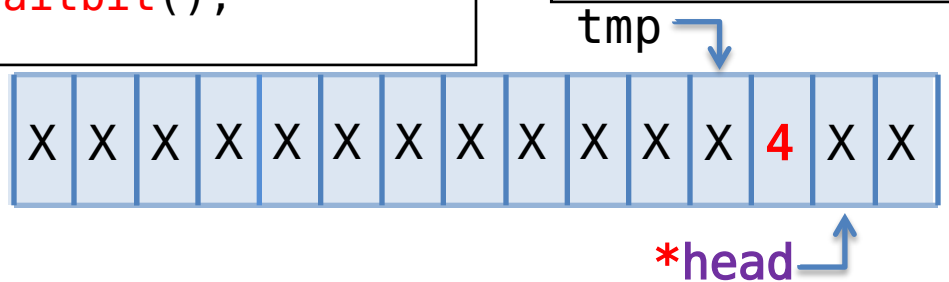
I'm waiting!

```
int pop() {
  …CAS(head, oldhead, tmp+1);
  if (waiting) {
    unique_lock lock(mutex);
    cond_full.notify();
  }
}
```



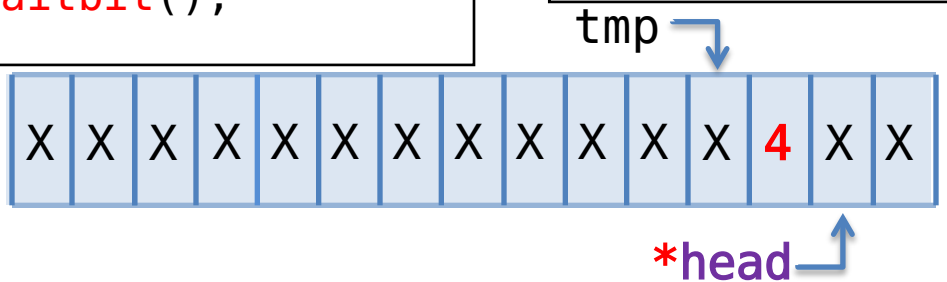
```
{
  unique_lock lock(mutex);
  if (waiting++ == 0)
    head.set_waitbit();
  while ( ! …find_tail… )
     cond_full.wait(lock);
  if (--waiting == 0)
    head.clear_waitbit();
}
```

```
int pop() {
  …CAS(head, oldhead, tmp+1);
  if (oldhead.waitbit()) {
    unique_lock lock(mutex);
    cond_full.notify();
  }
}
```



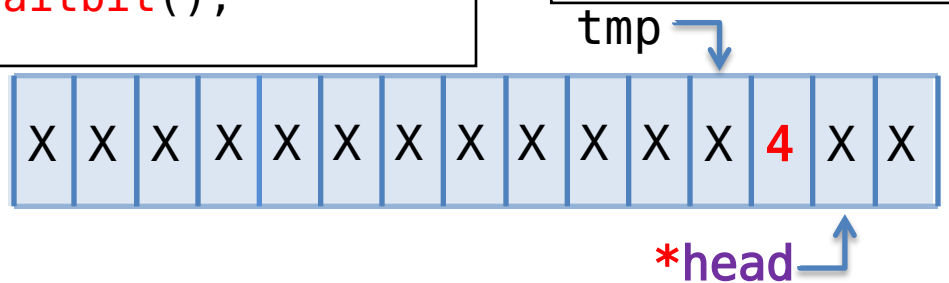
```
{
  unique_lock lock(mutex);
  if (waiting++ == 0)
    head.set_waitbit();
  while ( ! …find_tail… )
     cond_full.wait(lock);
  if (--waiting == 0)
    head.clear_waitbit();
}
```

```
int pop() {
  …CAS(head, oldhead, tmp+1);
  if (oldhead.waitbit()) {
    unique_lock lock(mutex);
    cond_full.notify();
  }
}
```



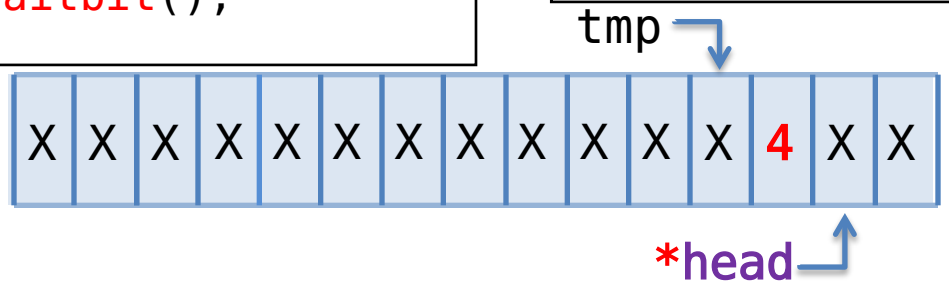
```
{
  unique_lock lock(mutex);
  if (waiting++ == 0)
    head.set_waitbit();
  while ( ! …find_tail… )
     cond_full.wait(lock);
  if (--waiting == 0)
    head.clear_waitbit();
}
```

```
int pop() {
  …CAS(head, oldhead, tmp+1);
  if (oldhead.waitbit()) {
    unique_lock lock(mutex);
    cond_full.notify();
  }
}
```



```
{
  unique_lock lock(mutex);
  if (waiting++ == 0)
    head.set_waitbit();
  while ( ! …find_tail… )
     cond_full.wait(lock);
  if (--waiting == 0)
    head.clear_waitbit();
}
```

```
int pop() {
  …CAS(head, oldhead, tmp+1);
  if (oldhead.waitbit()) {
    unique_lock lock(mutex);
    cond_full.notify();
  }
}
```



NOTE: **waiting** is NOT atomic

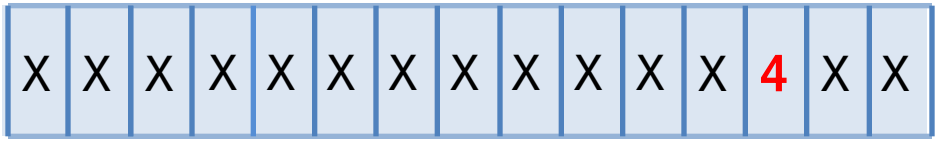

AUTION !
oison Ivy
ay on Trail

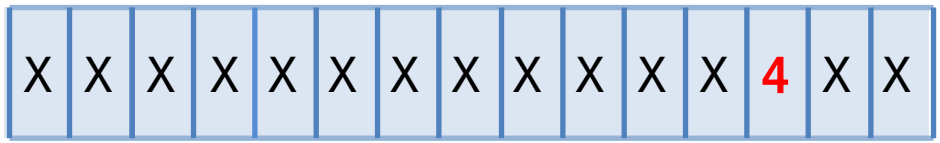
Looking Back

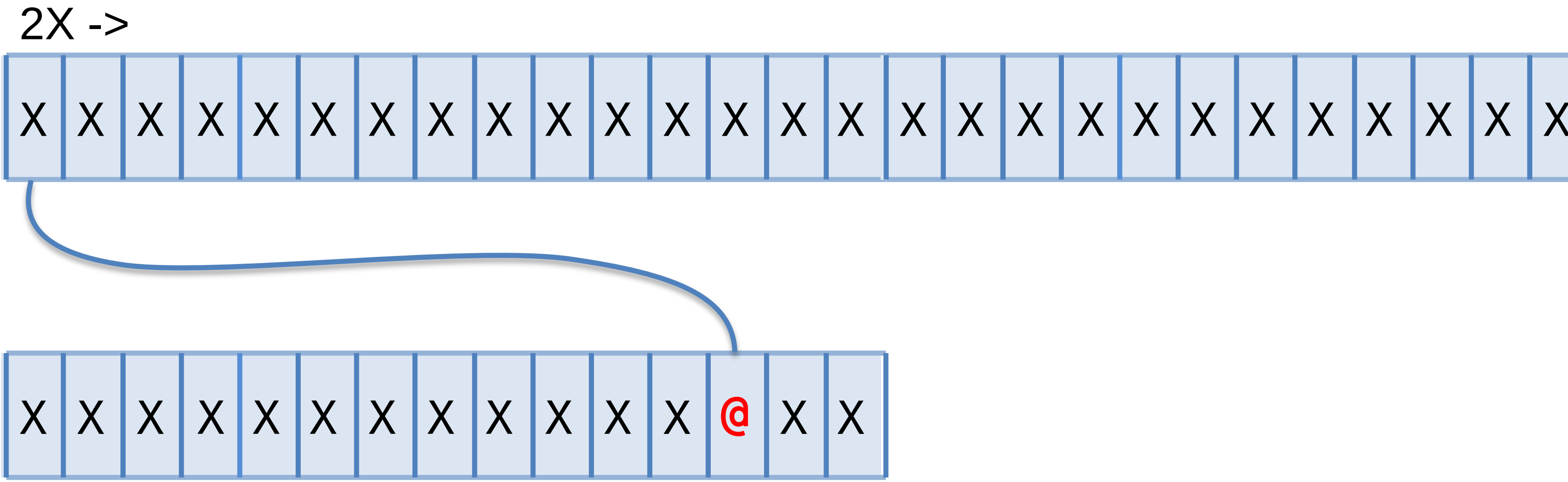
Looking Back

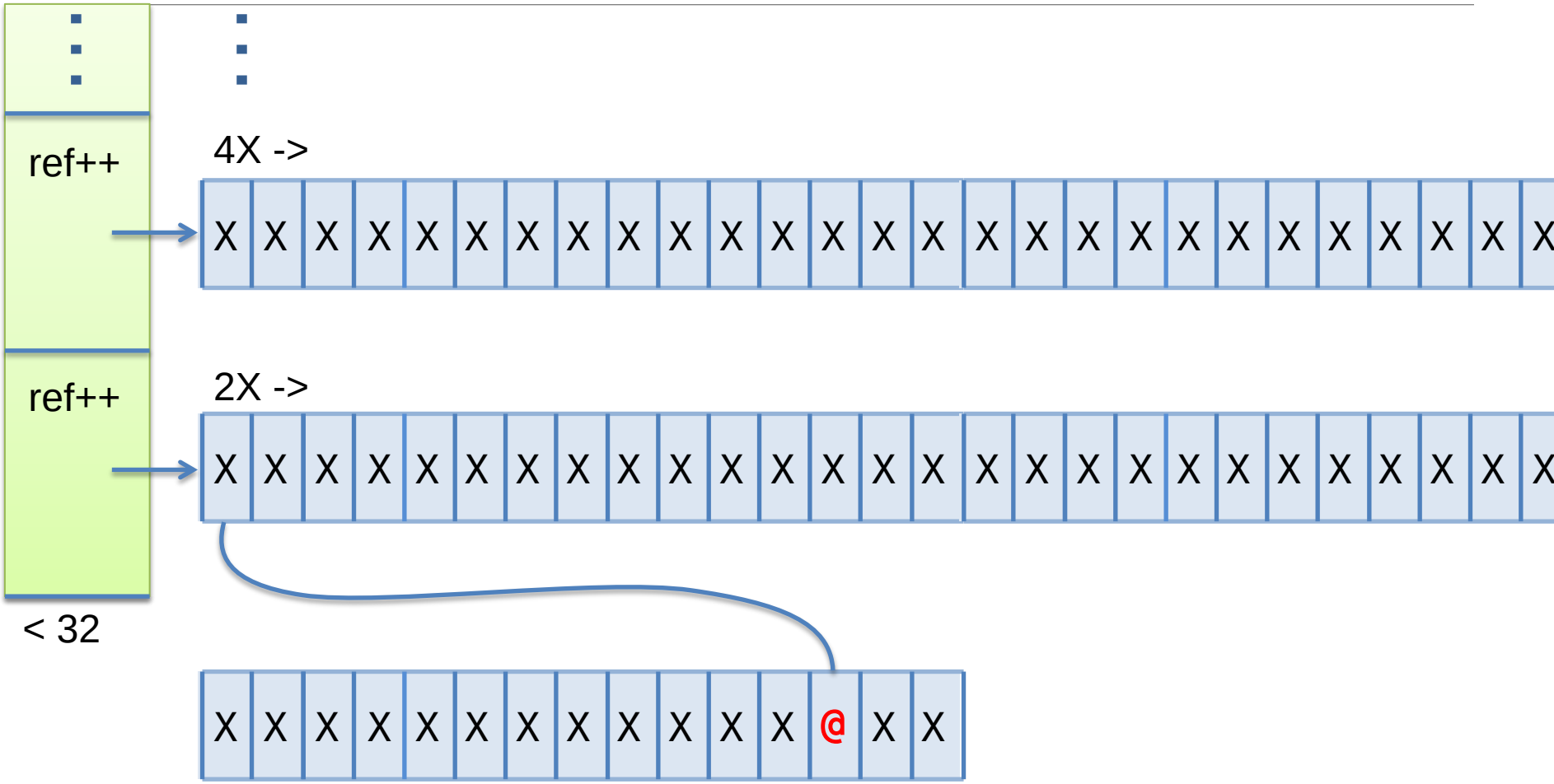
push()

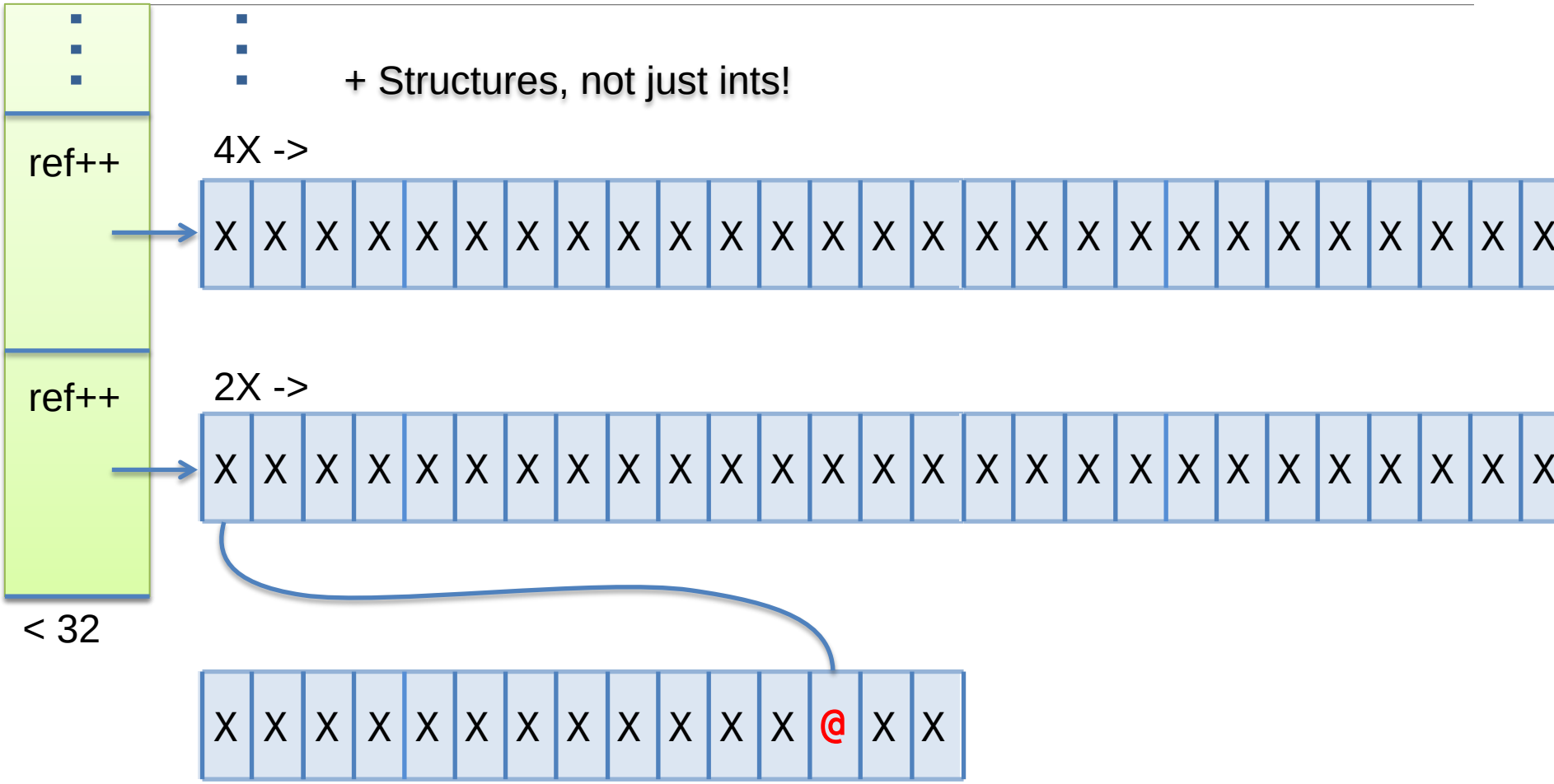
Looking Ahead


X X X X X X X X X X X X 4 X X


X X X X X X X X X X X X 4 X X


2X ->
X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X X
X X X X X X X X X X X X @ X X


ref++
4X ->
ref++
2X ->
< 32
@


+ Structures, not just ints!
4X ->
ref++
2X ->
ref++
< 32
@

“The Problem with Threads”

http://ptolemy.eecs.berkeley.edu/

http://ptolemy.eecs.berkeley.edu/publications/papers/06/problemwithThreads/

“A part of the Ptolemy Project experiment was to see whether **effective software engineering practices** could be developed for an academic research setting. We developed a process that included a code maturity rating system (with four levels, red, yellow, green, and blue), **design reviews**, **code reviews**, **nightly builds**, **regression tests**, and **automated code coverage metrics**. The portion of the kernel that ensured a consistent view of the program structure was written in early 2000, design reviewed to yellow, and code reviewed to green. The **reviewers included concurrency experts**, not just inexperienced graduate students (Christopher Hylands (now Brooks), Bart Kienhuis, John Reekie, and myself were all reviewers). We wrote **regression tests that achieved 100 percent code coverage**. The nightly build and regression tests ran on a two processor SMP machine, which exhibited different thread behavior than the development machines, which all had a single processor. The Ptolemy II **system** itself began to be **widely used**, and every use of the system exercised this code. **No problems were observed until the code deadlocked on April 26, 2004, four years later**.”

All states are valid states for all lines of code!