Lock-free by Example

(one very complicated example)

Tony Van Eerd

CppCon, September 2014

BlackBerry

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

Use Locks

Guide to ~~Threaded~~ Coding

Guide to ~~Threaded~~ Coding

MACROS are EVIL

NOTE:

CAS = compare_exchange

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:

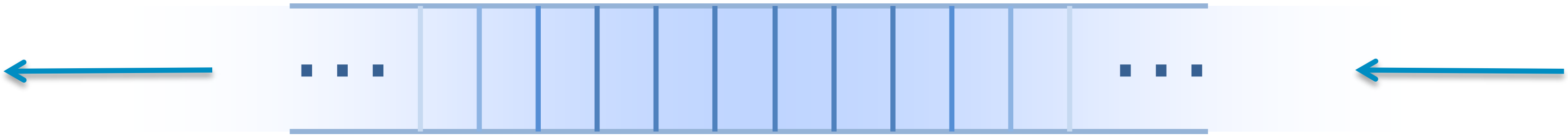
NOTE:

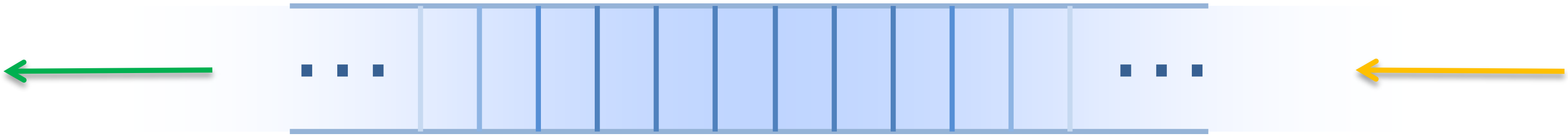
CAS = compare_exchange

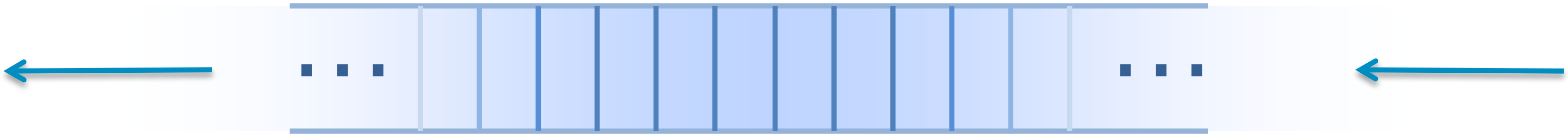
Not my coding style/structure

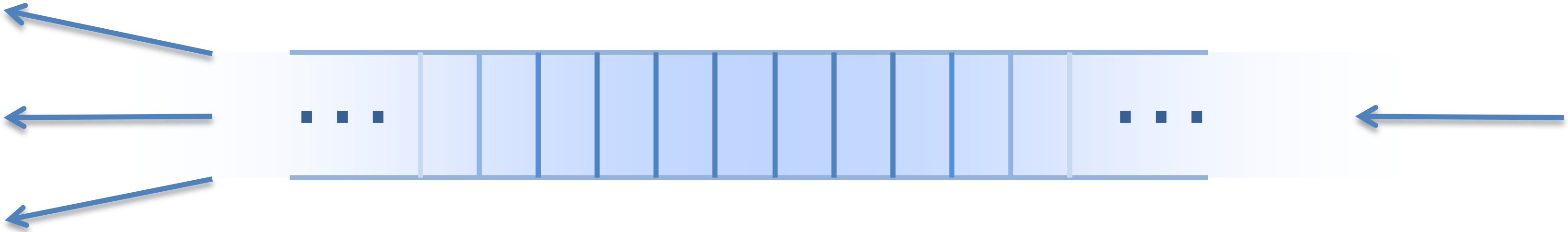
Remember to lower the audience’s expectations:

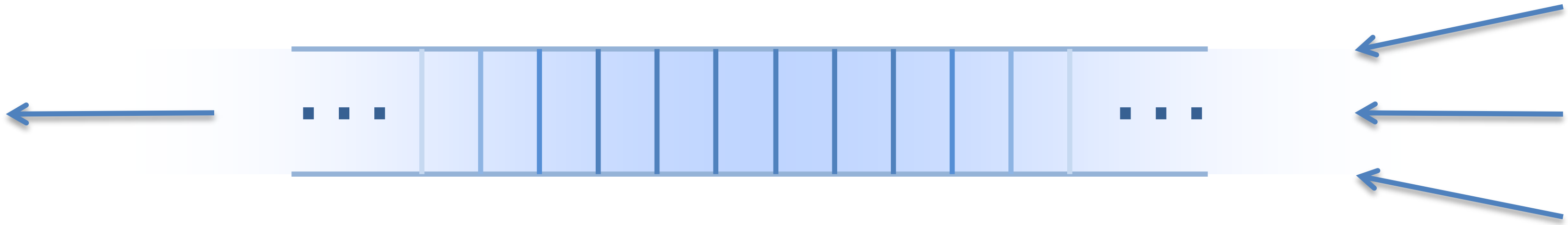
I’m no Paul McKenney

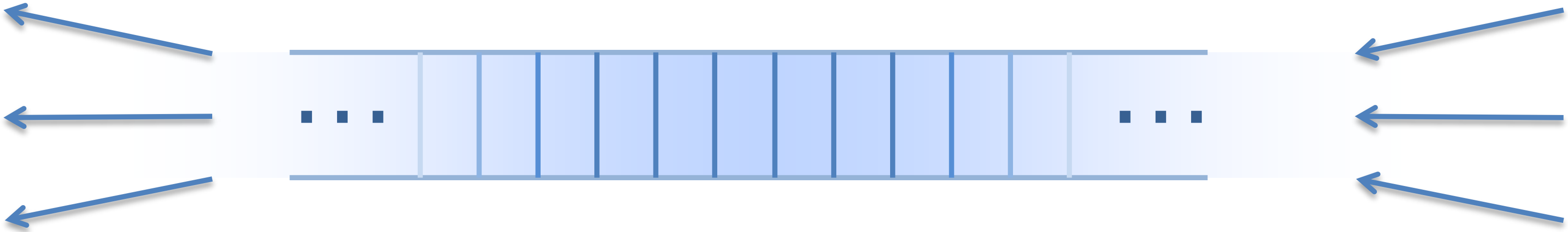




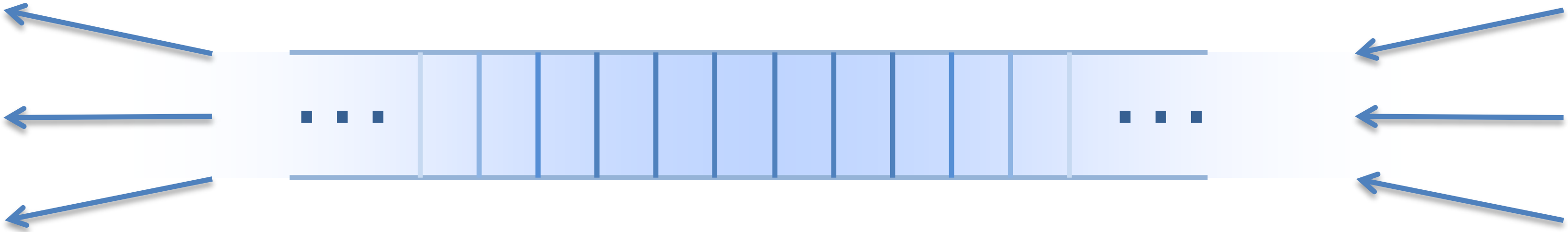




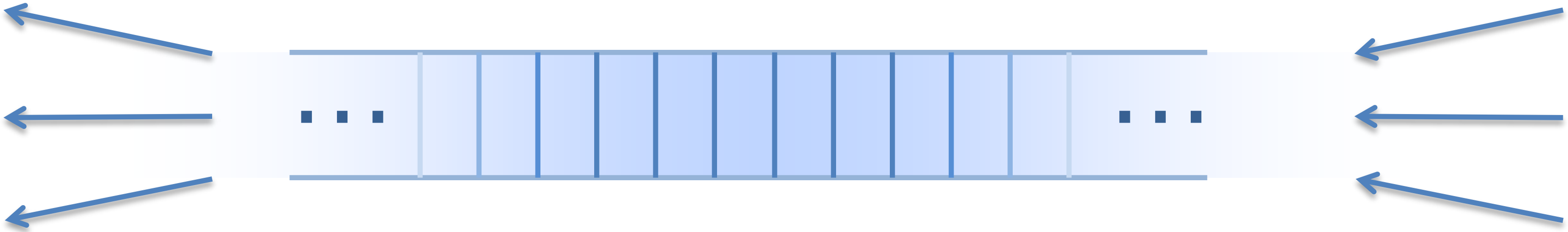




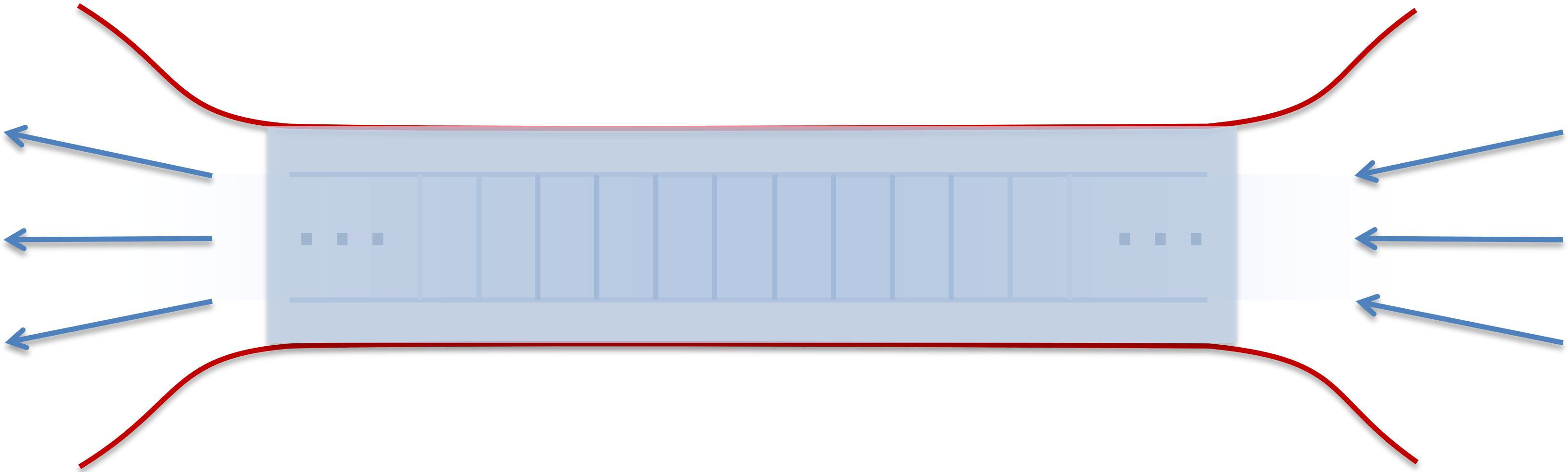
Multi-Producer Multi-Consumer Queue



MPMC Queue

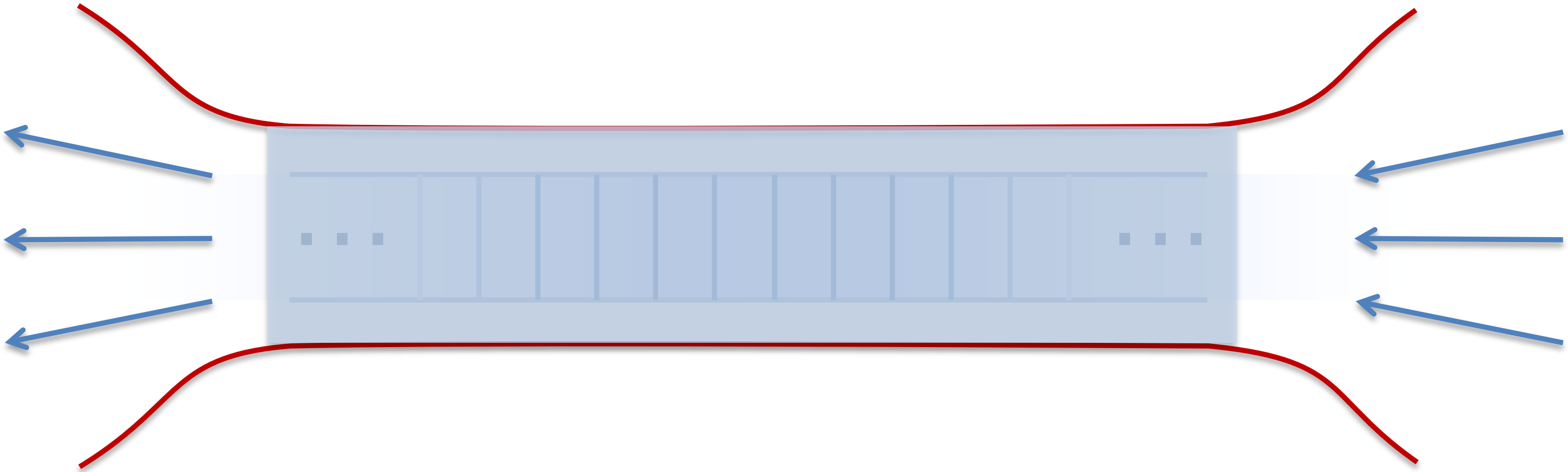


MPMC Queue

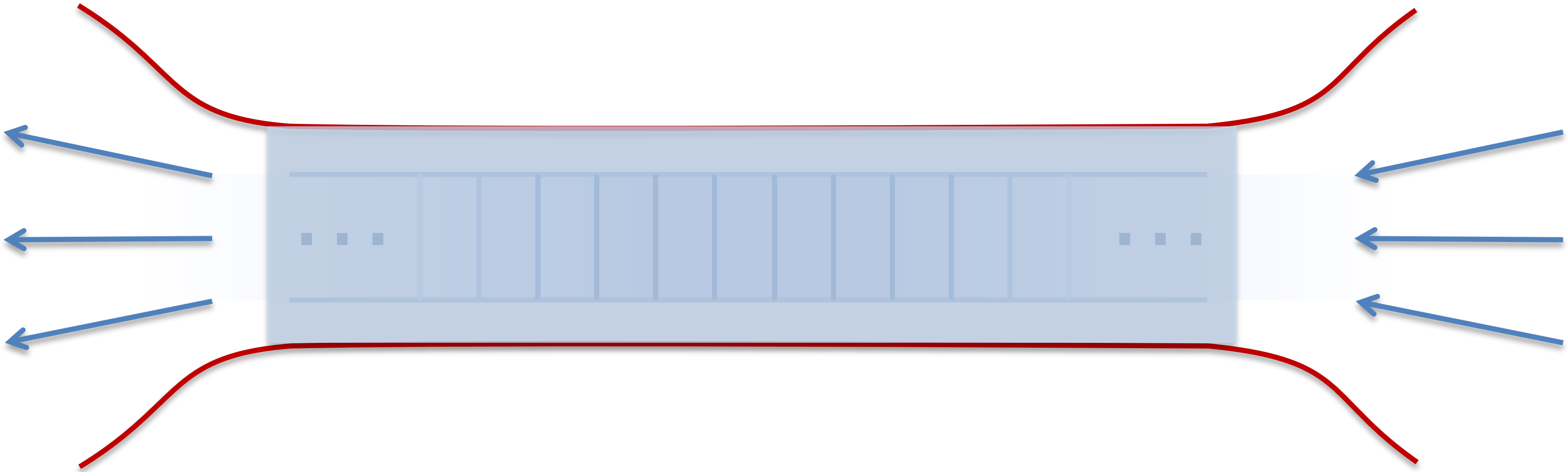


MPMC Queue

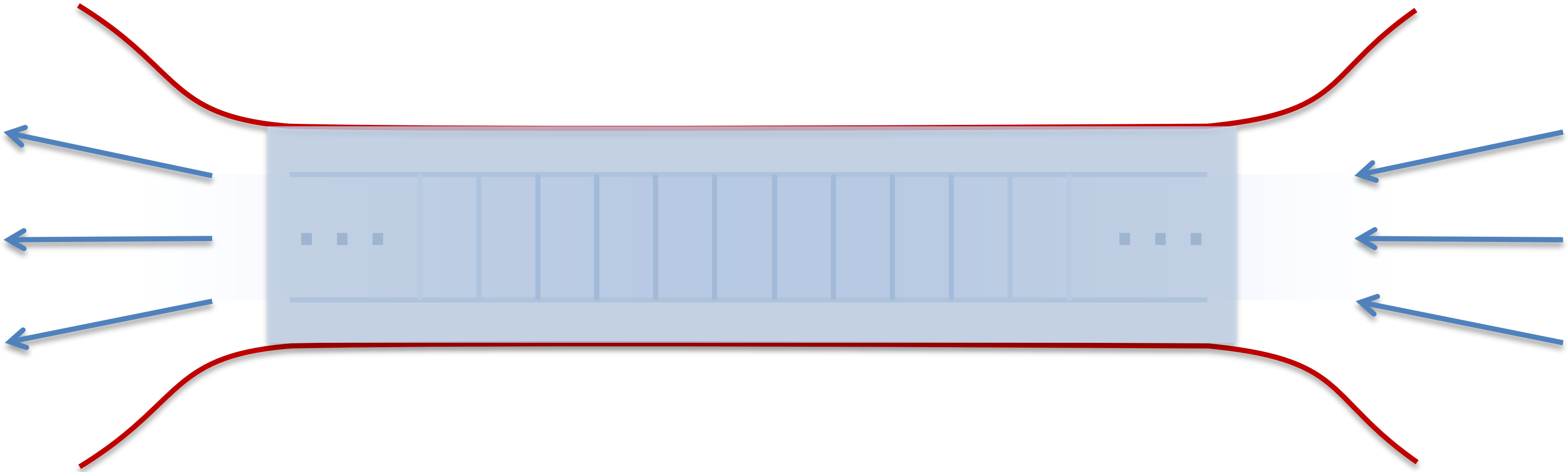
SPSC
SPMC
MPSC
MPMC

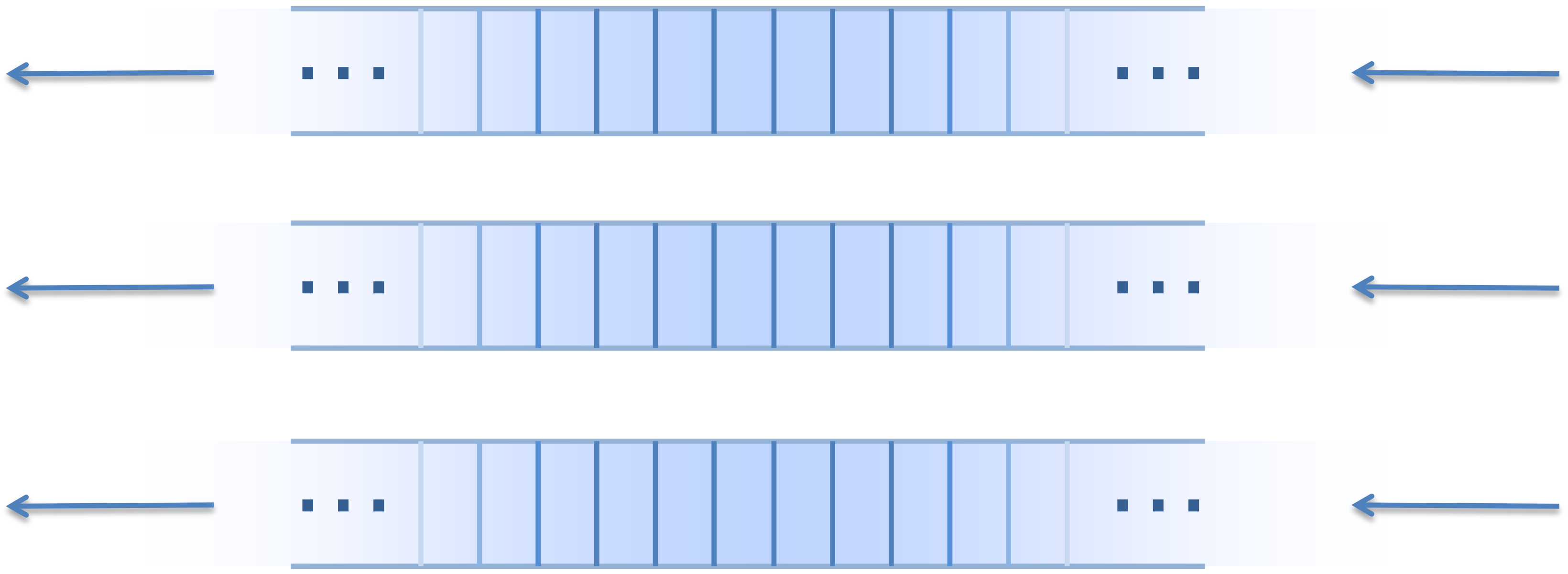


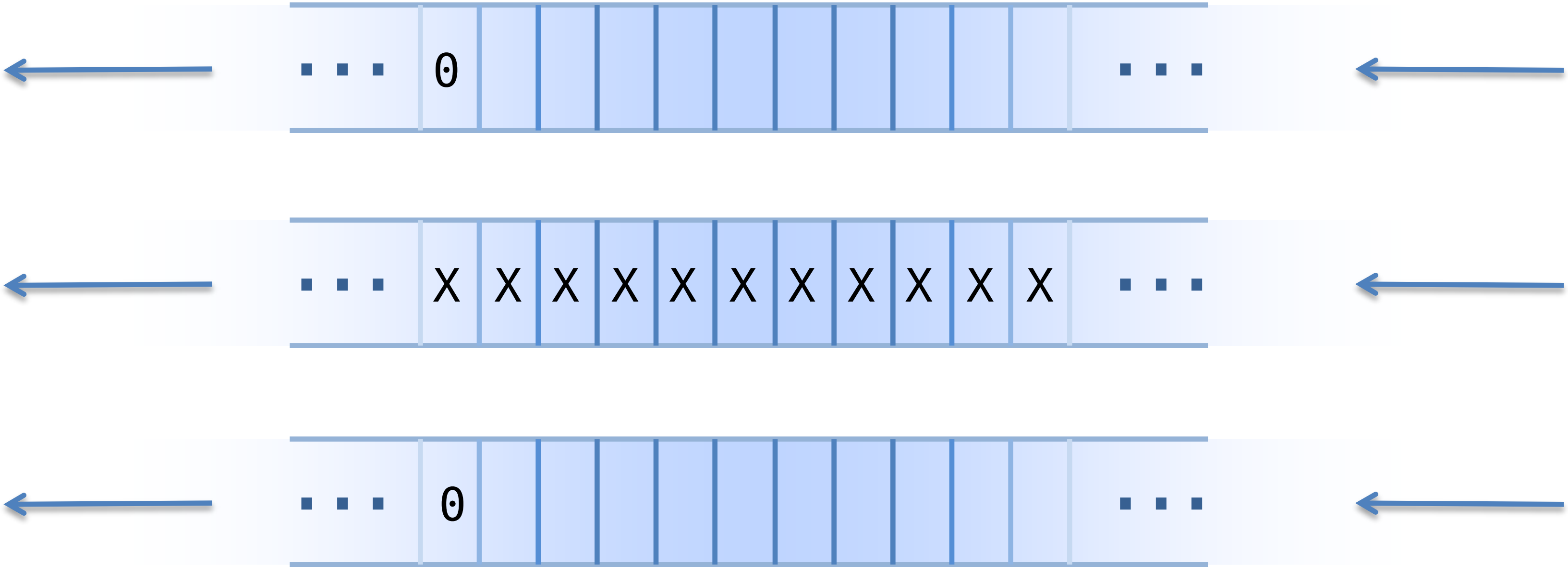
MPMC Queue



Bottleneck



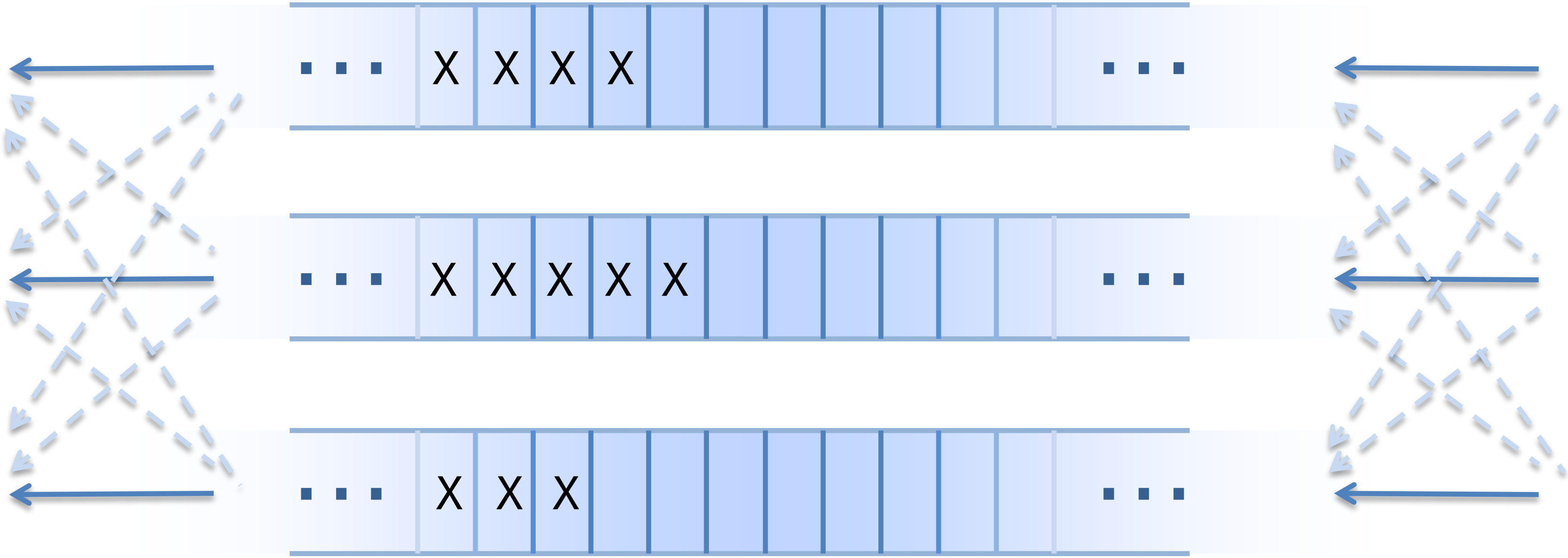


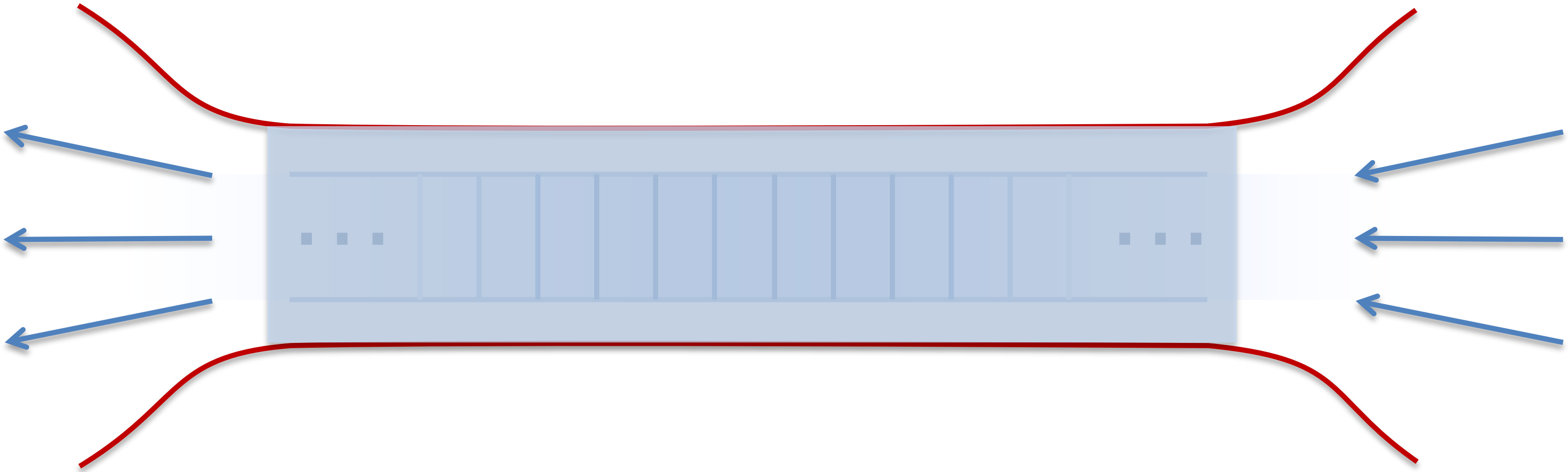


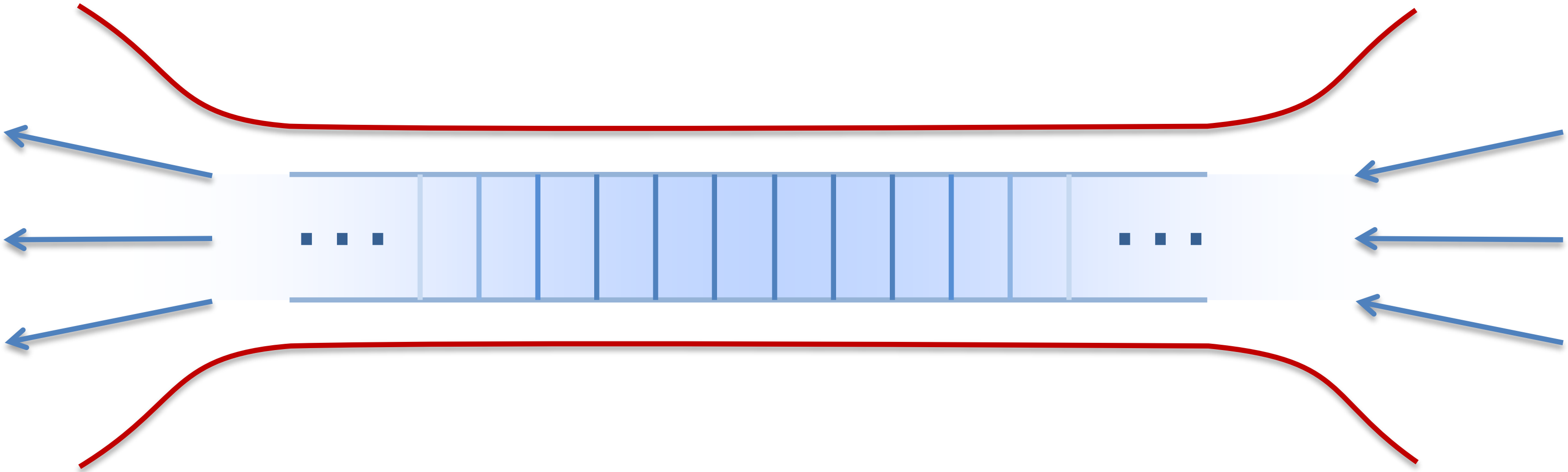
... 0 ...

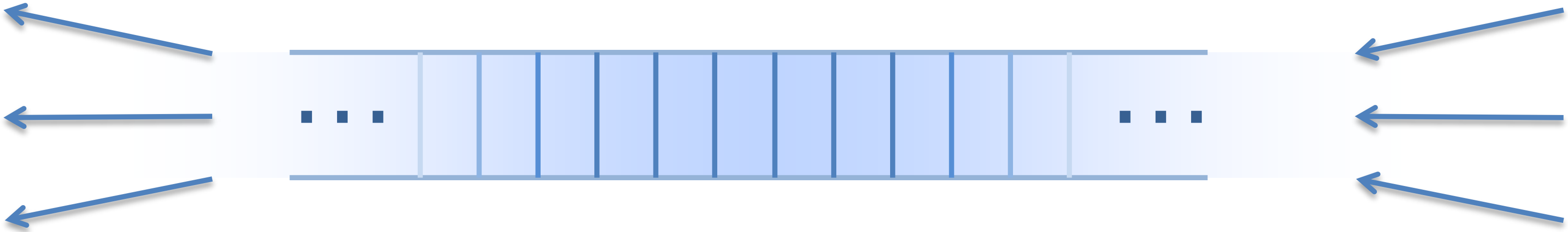
... X X X X X X X X X X X ...

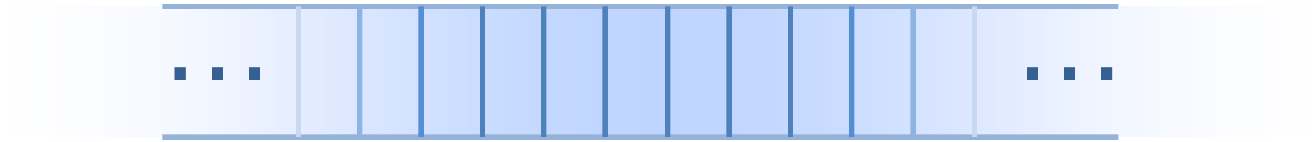
... 0 ...

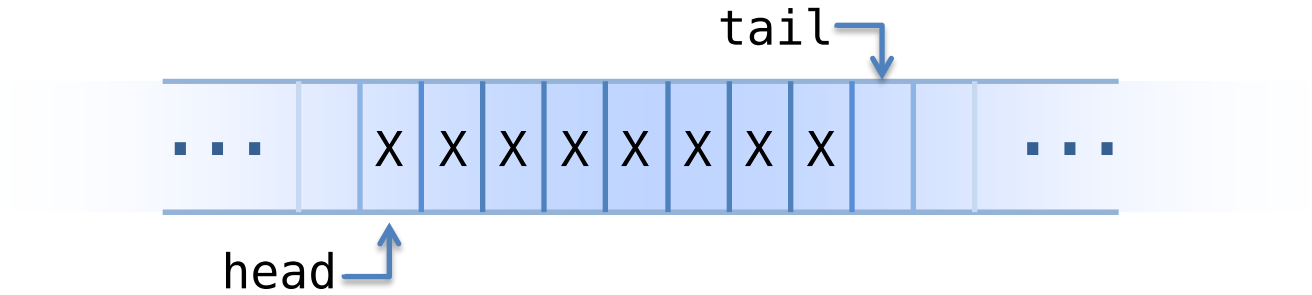

. . .
X X X X
. . .
. . .
X X X X X
. . .
. . .
X X X
. . .



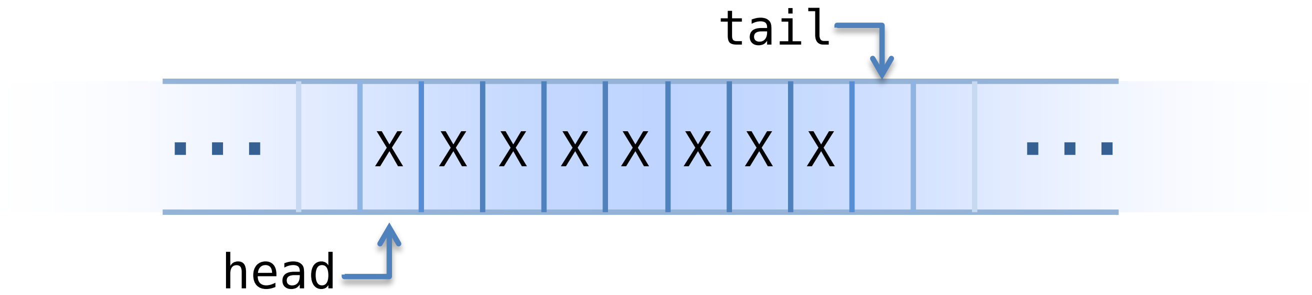




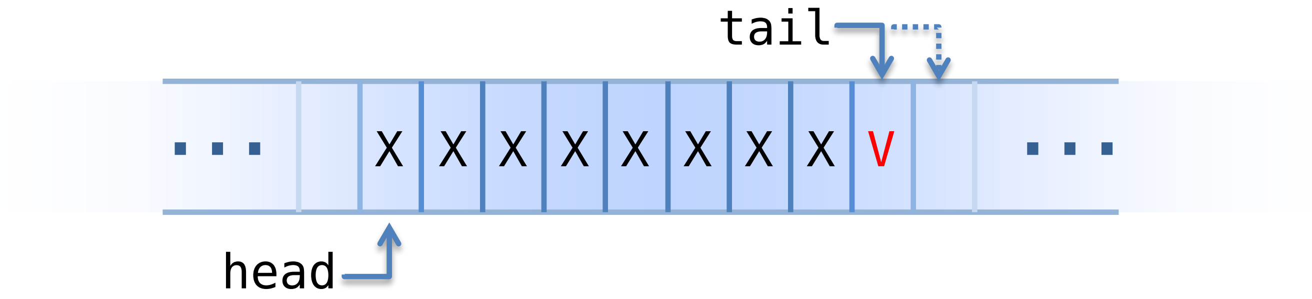



tail
...
X
X
X
X
X
X
X
X
...
head

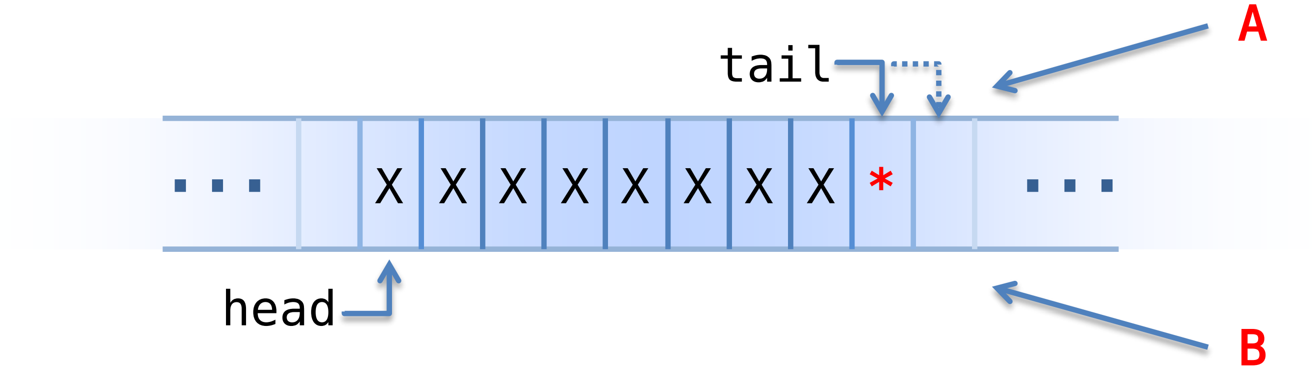
```
class Queue
{
    int buffer[some_size];
    size_t head;
    size_t tail;
};
```



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

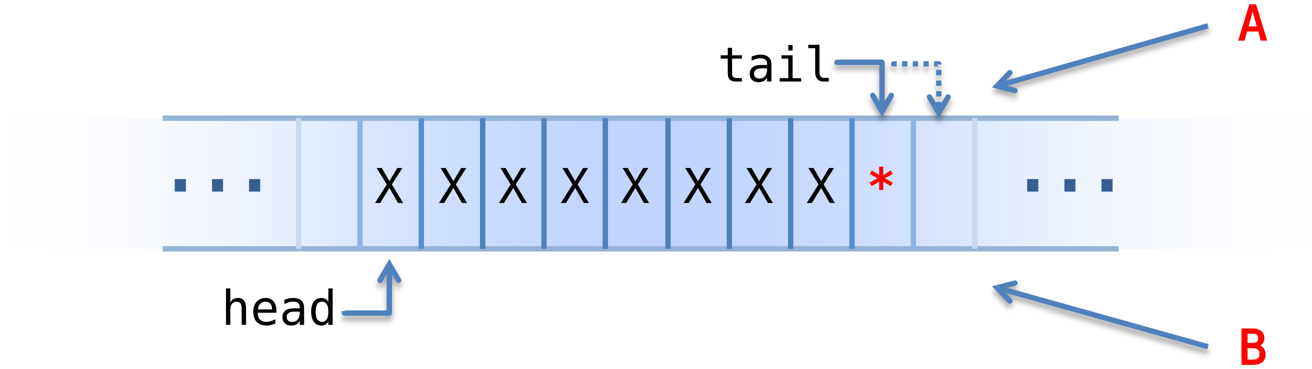


```
void push(int val)
{
    buffer[tail++] = val;
}
```

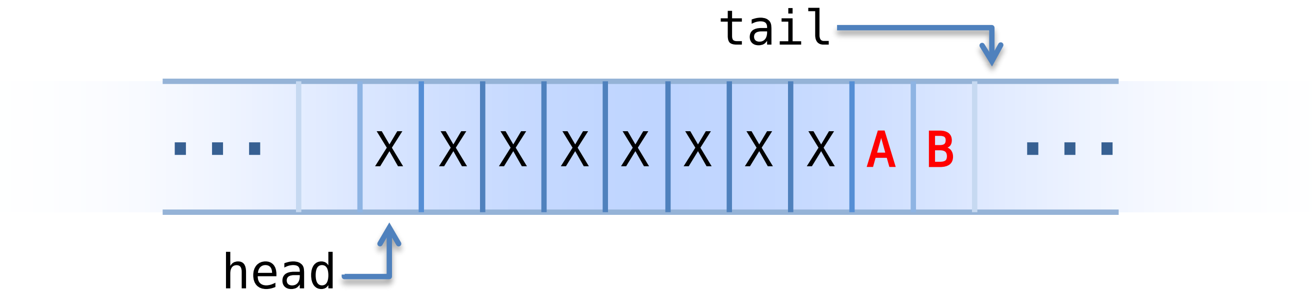


```
void push(int val)
{
    buffer[tail++] = val;
}
```

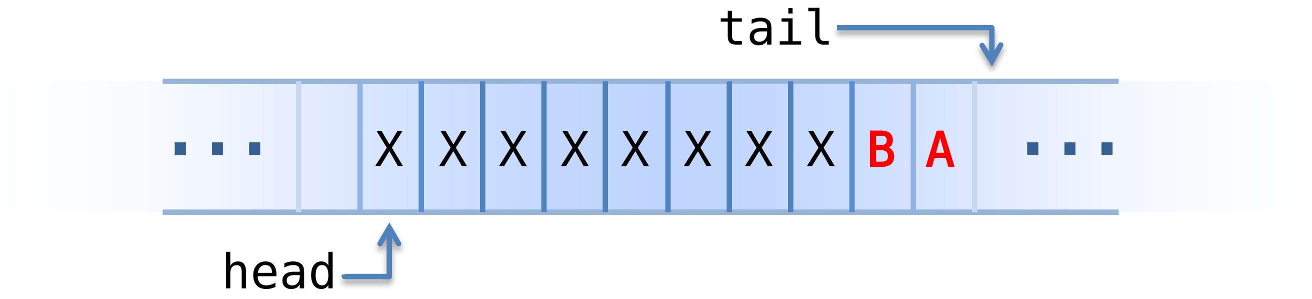
Possible Outcomes?



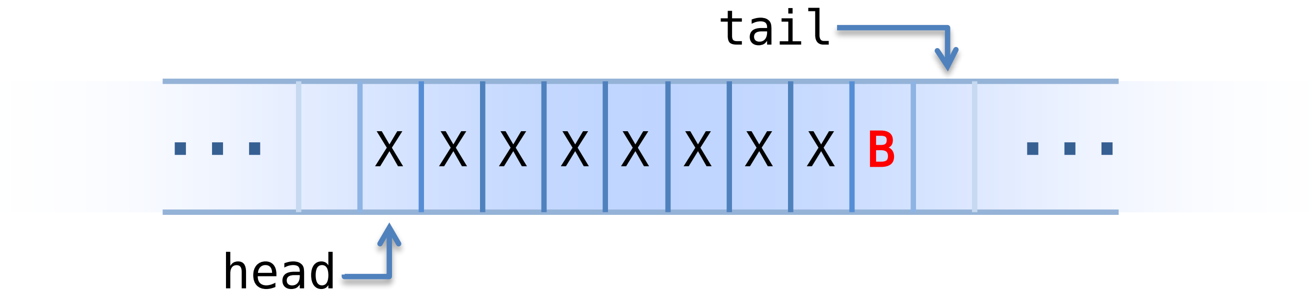
```
void push(int val)
{
    buffer[tail++] = val;
}
```



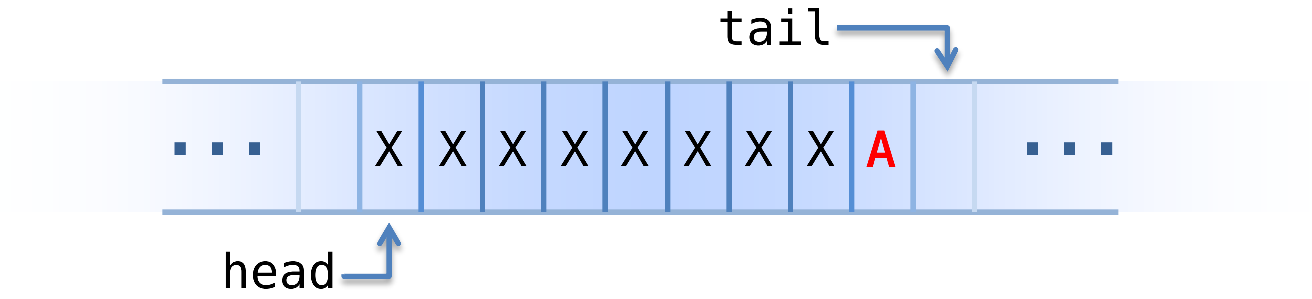
```
void push(int val)
{
    buffer[tail++] = val;
}
```



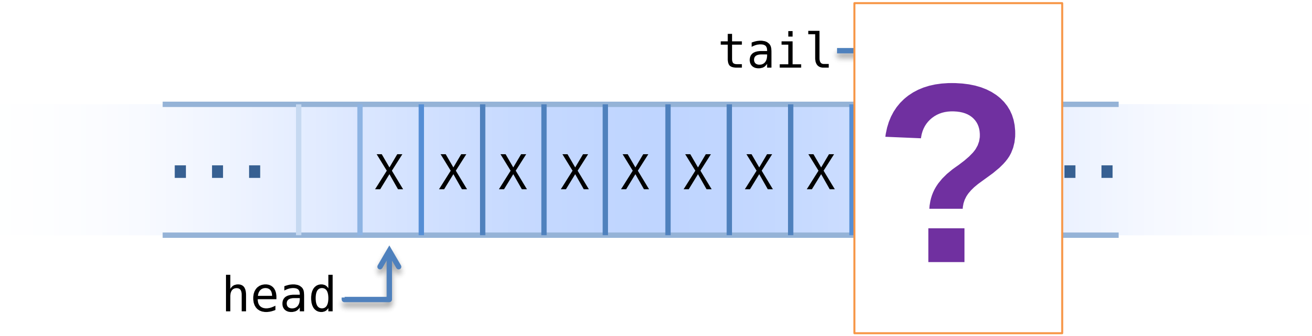
```
void push(int val)
{
    buffer[tail++] = val;
}
```



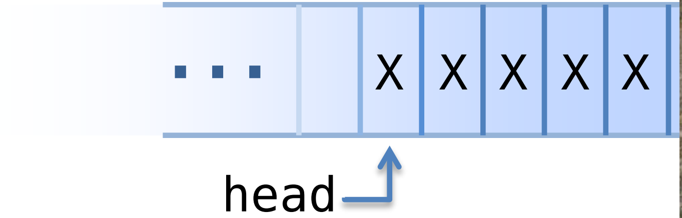
```
void push(int val)
{
    buffer[tail++] = val;
}
```



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



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

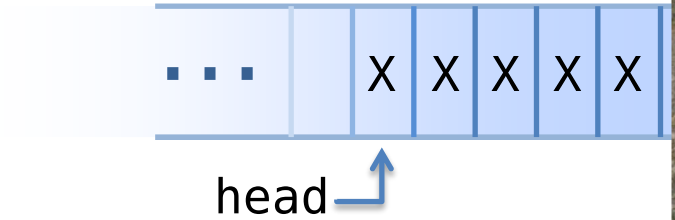


```
void push(int val)
{
    buffer[tail++] = val;
}
```

```
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

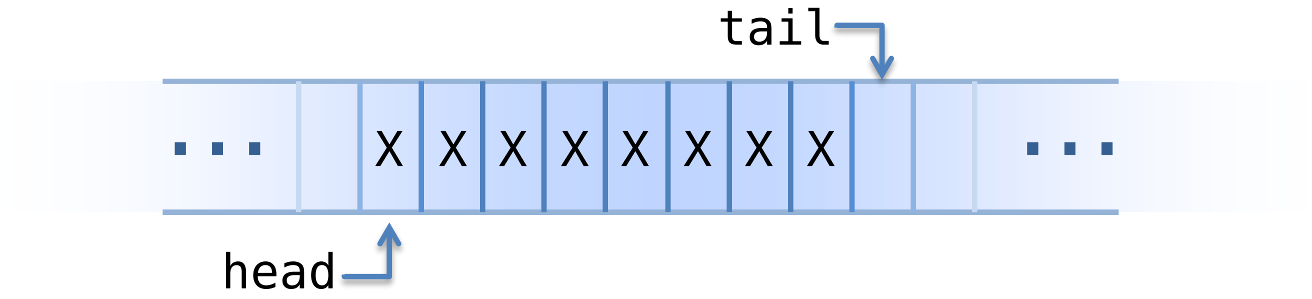
UNDEFINED BEHAVIOUR

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

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

A

B

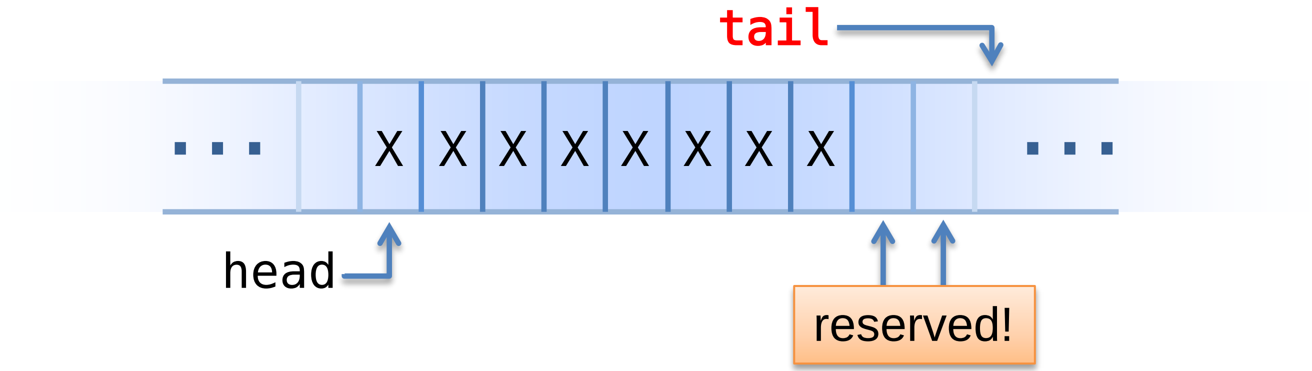


```
void push(int val)
{
    buffer[tail++] = val;
}
```

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

A

B

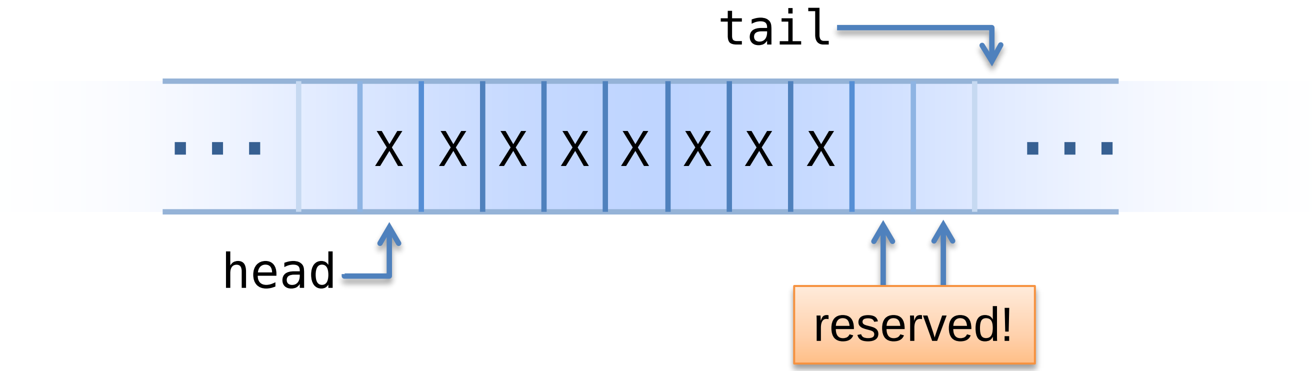


```
void push(int val)
{
    buffer[tail++] = val;
}
```

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

A

B



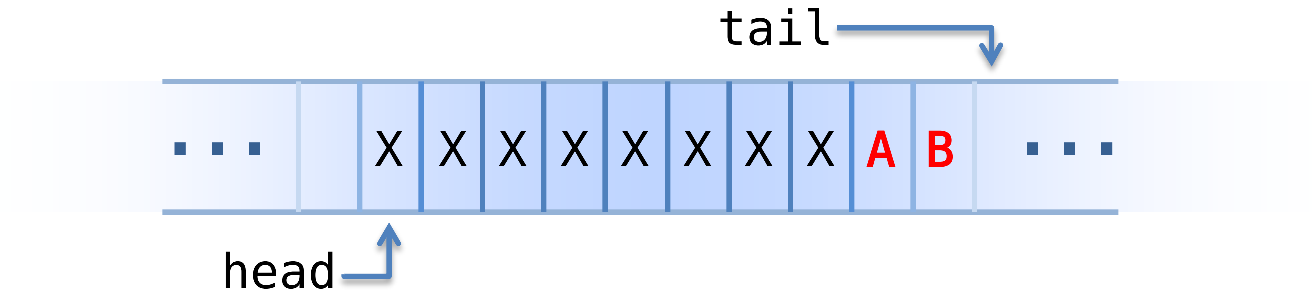
reserved!

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

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

A

B

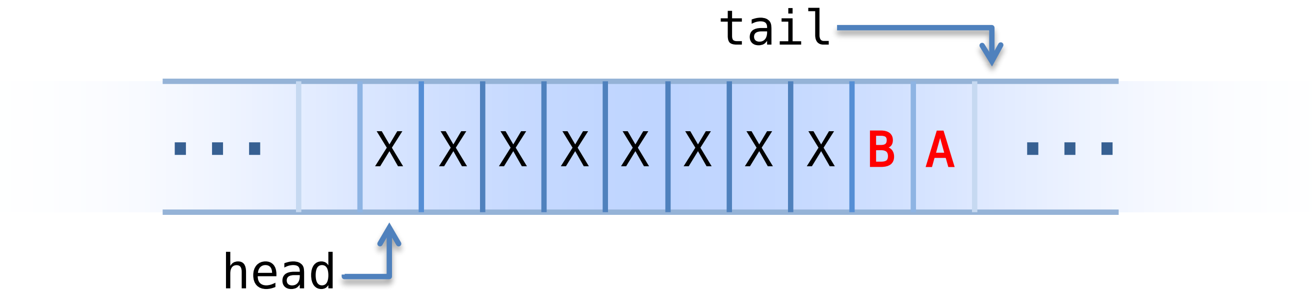


```
void push(int val)
{
    buffer[tail++] = val;
}
```

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

A

B

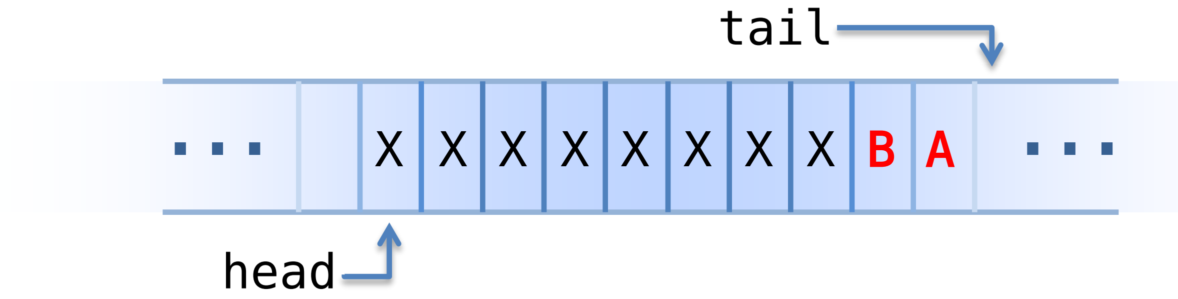


```
void push(int val)
{
    buffer[tail++] = val;
}
```

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

A

B

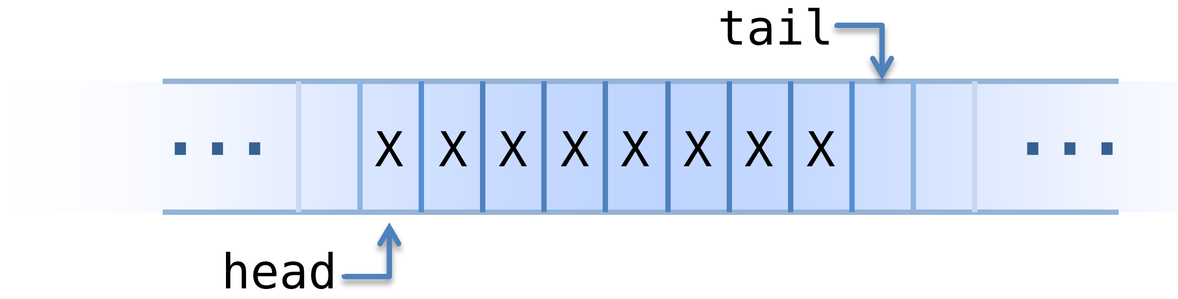


```
void push(int val)
{
    buffer[tail++] = val;
}
```

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

A

B



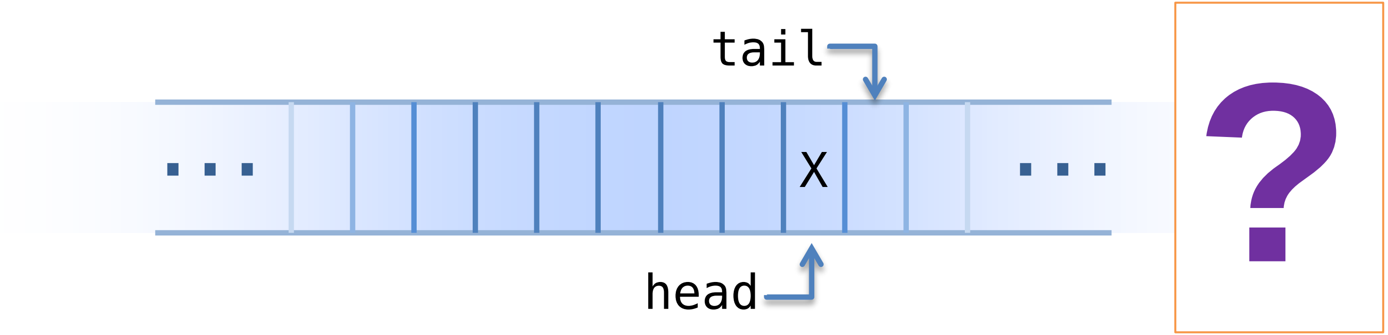
?

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

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

A

B



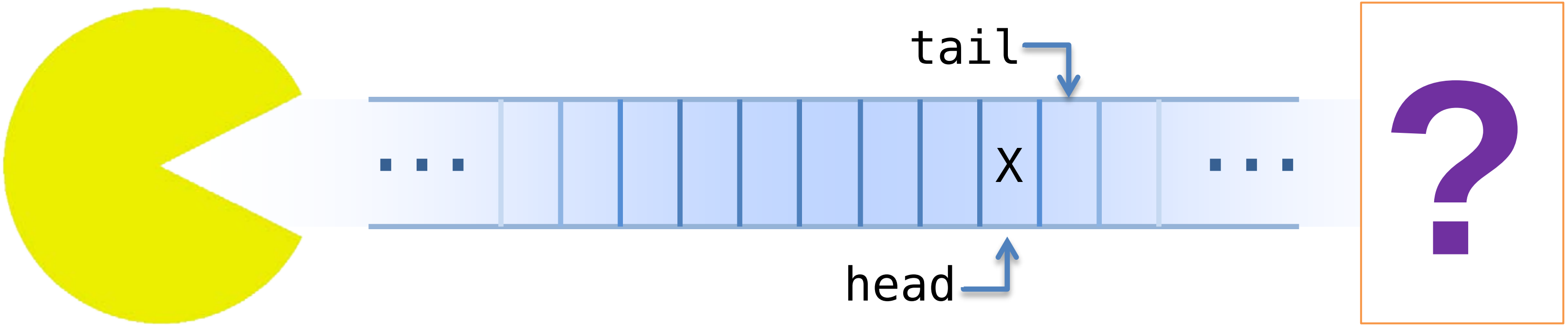
```
void push(int val)
{
    buffer[tail++] = val;
}
```

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

X...

A

B



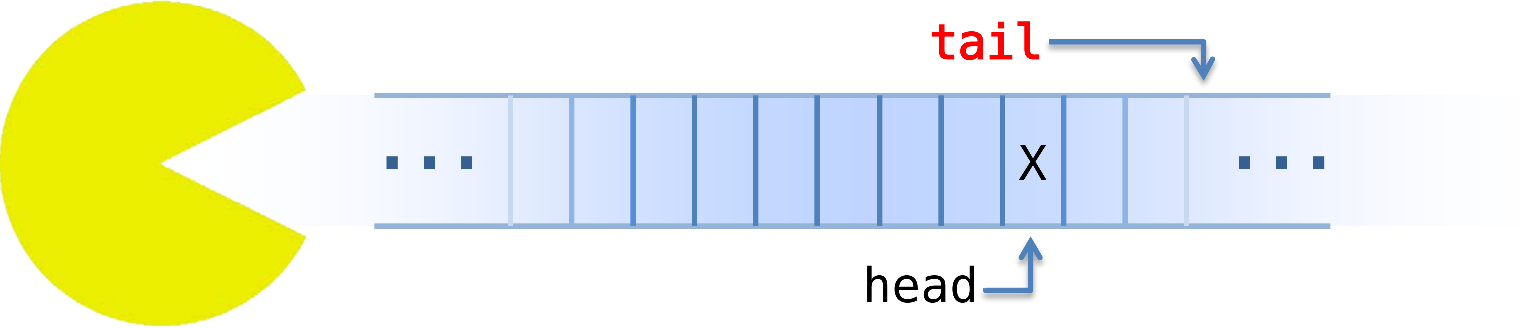
```
void push(int val)
{
    buffer[tail++] = val;
}
```

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

X...

A

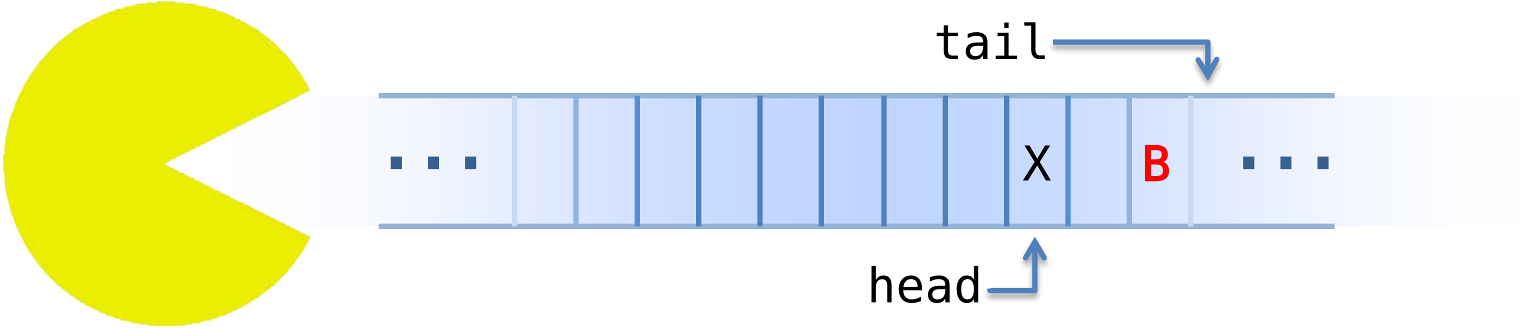
B



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

A B X...

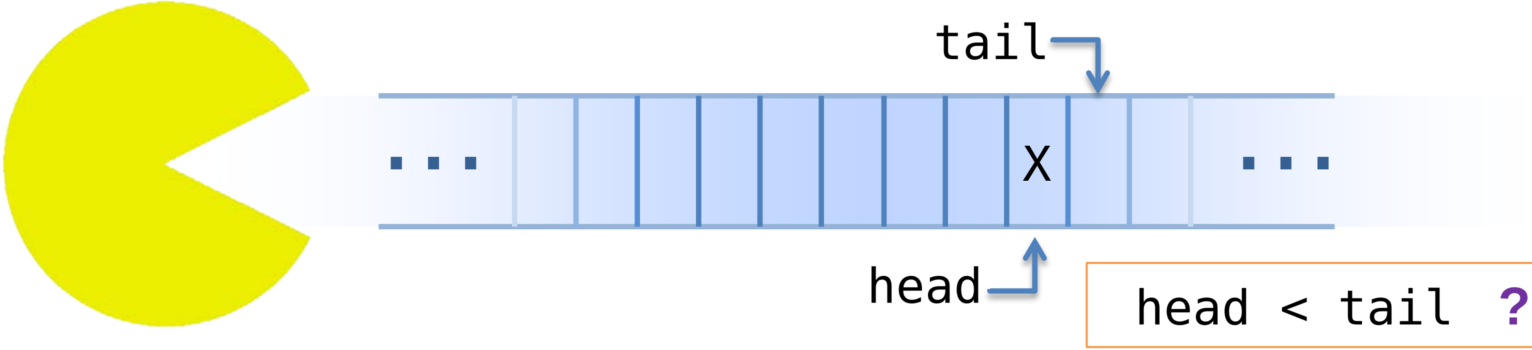
```
class Queue
{
    int buffer[SZ];
    atomic<size_t> head;
    atomic<size_t> tail;
};
```



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

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

X...

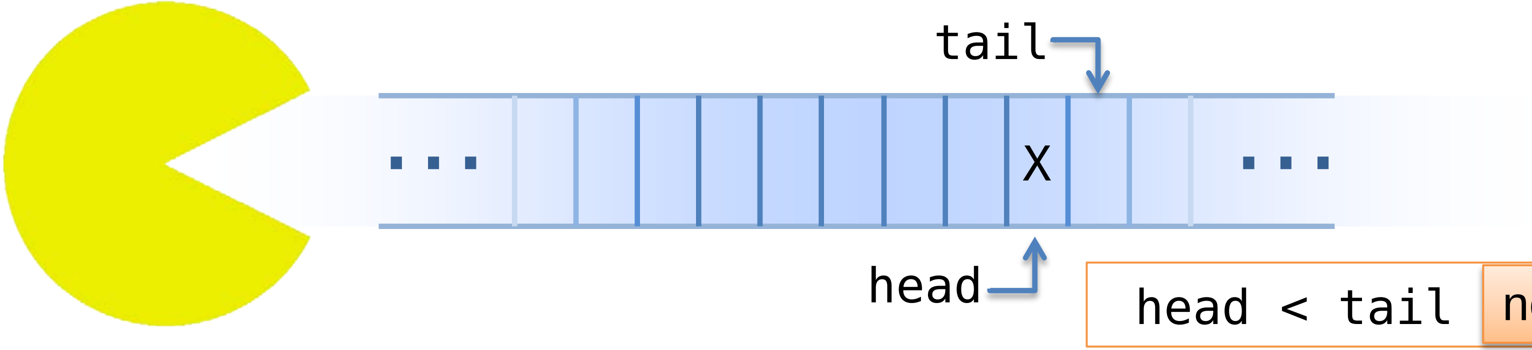


head < tail ?

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

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

X...



head < tail

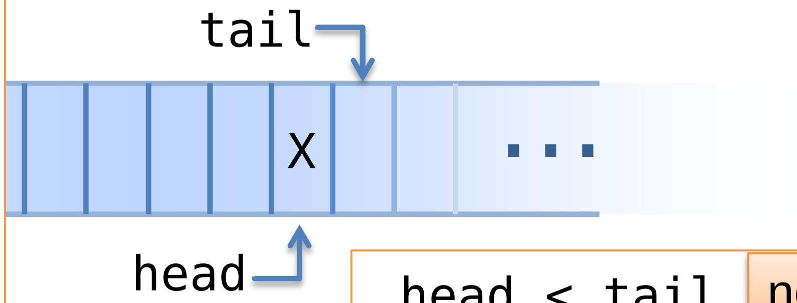
not atomic!

```
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))
{
    do_something();
};
```



head < tail

not atomic!

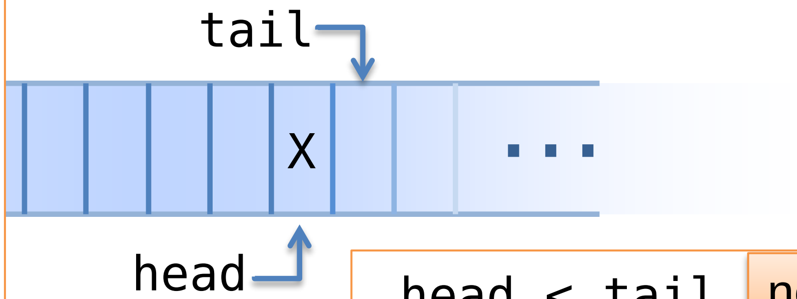
```
void push(int val)
{
    buffer[tail++] = val;
}
```

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

X...

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

THEN



head < tail

not atomic!

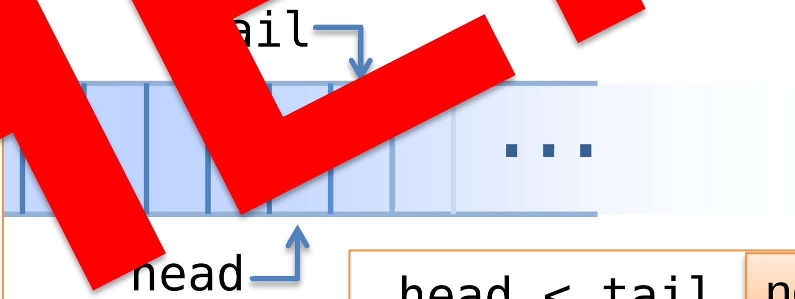
```
void push(int val)
{
    buffer[tail++] = val;
}
```

```
class Queue
{
    int          [SZ]
    atom   siz         d;
    atomi   ize_        ;
};
```

X...

```
if (atomic_less(head, tail
{
    do_somet   g()
};
```

THEN



head < tail

not atomic!

THEN

```
void pu
{
    buf
}
```

X...

;
;

```
if (atom
{
      do_s
};
```

THEN

THEN

is a 4-letter word

atomic!

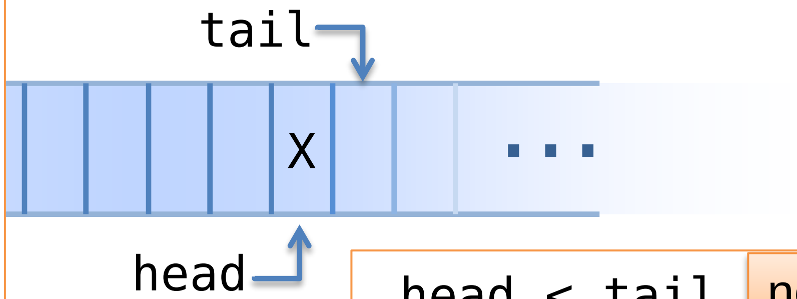
```
void push(int val)
{
    buffer[tail++] = val;
}
```

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

X...

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

THEN



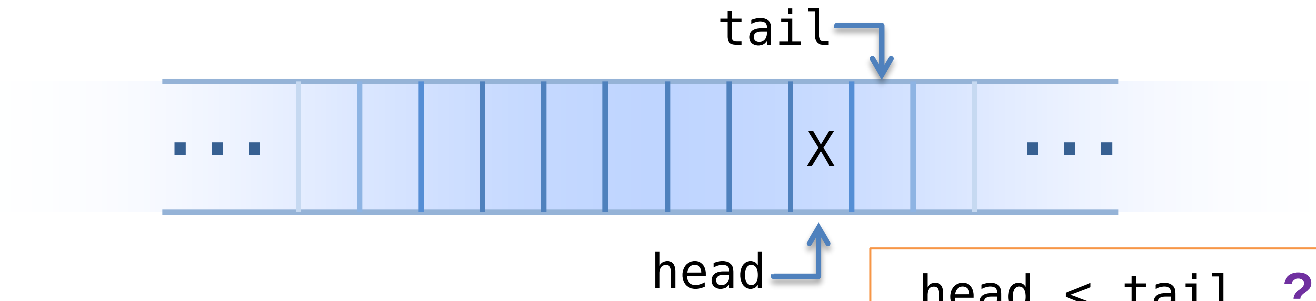
head < tail

not atomic!

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

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

X...



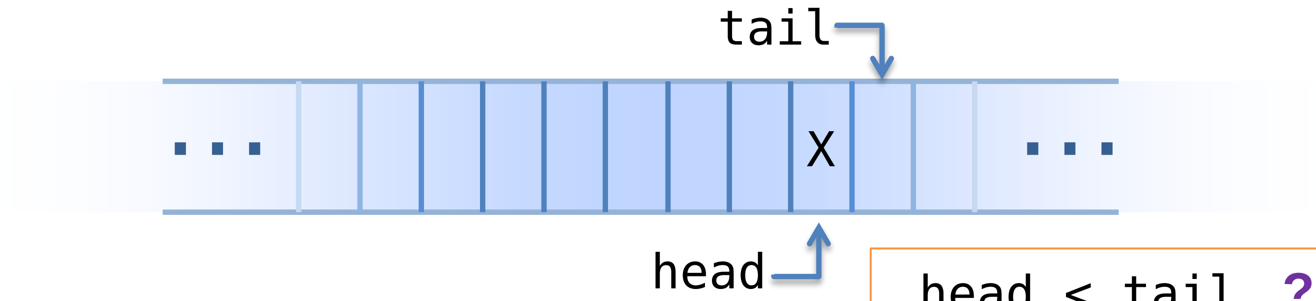
head < tail ?

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...



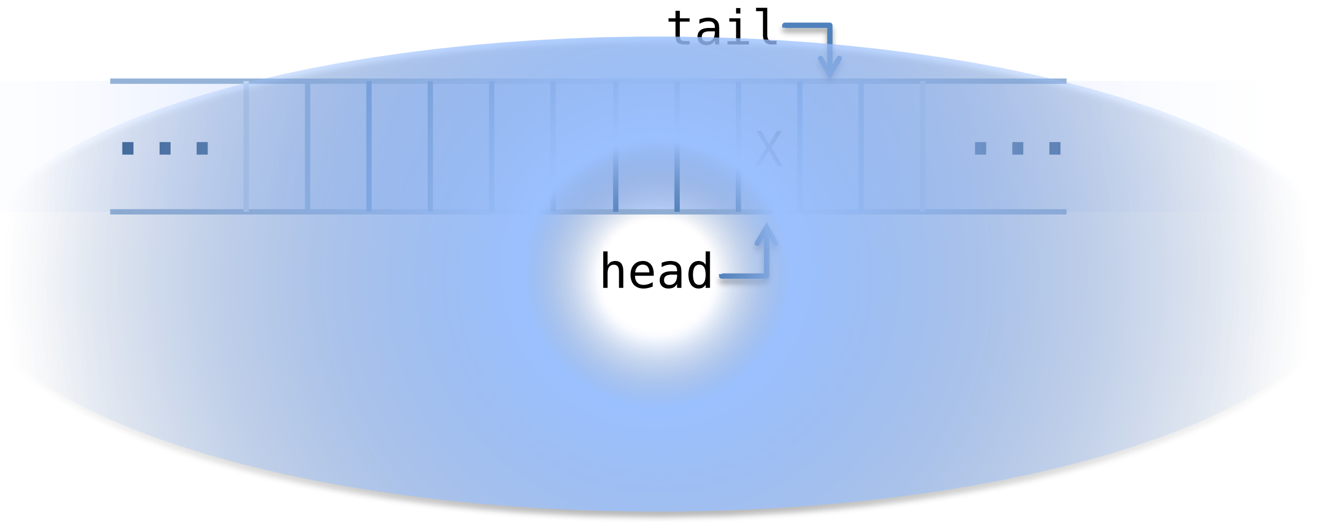
head < tail ?

don’t assume **STATE**

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

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

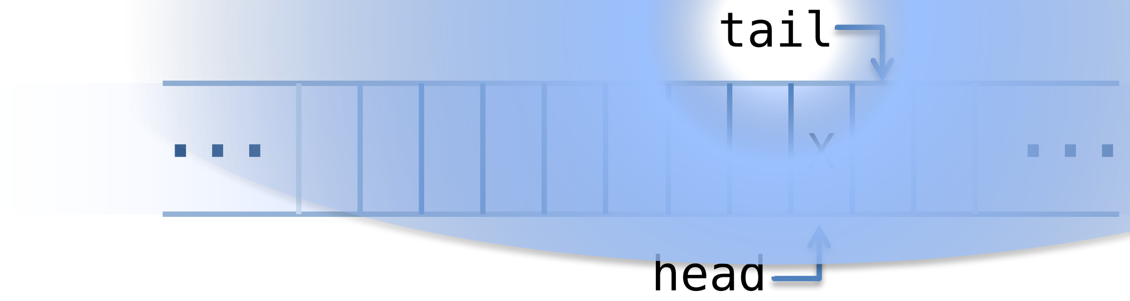
X...



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

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

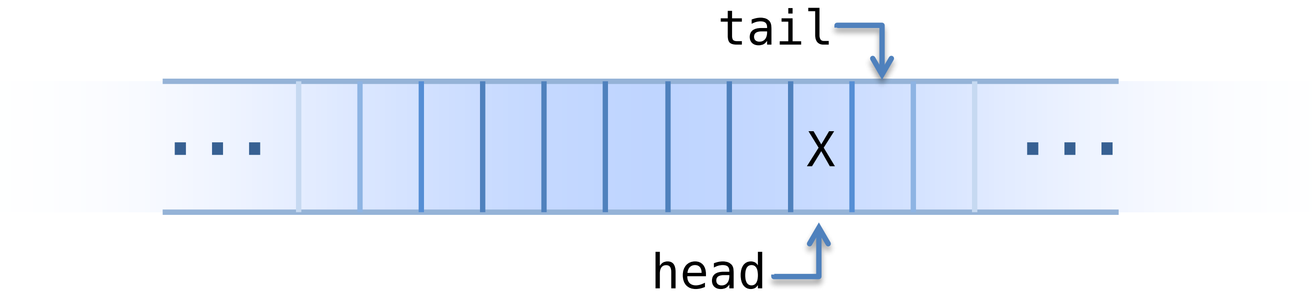
X...



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

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

X...

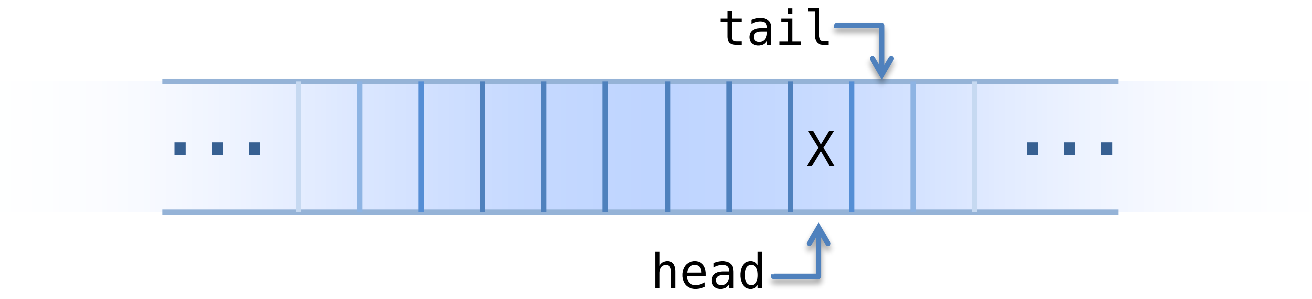


don’t assume **STATE**

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

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

X...

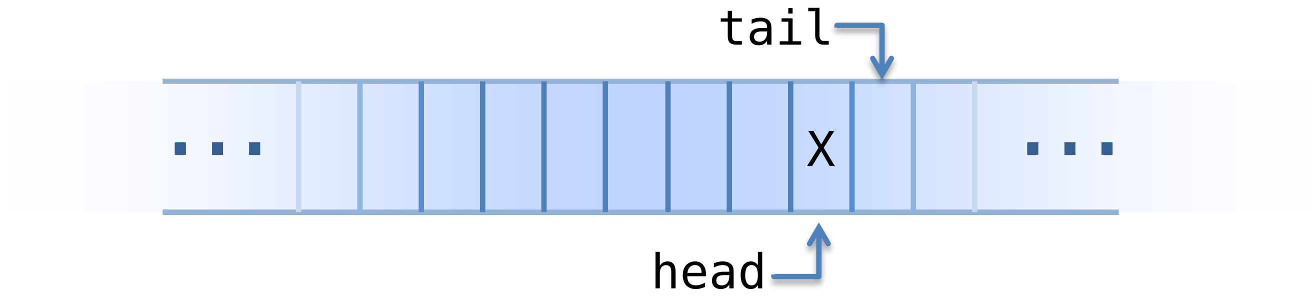


every **STATE** is a good **STATE**

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

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

X...

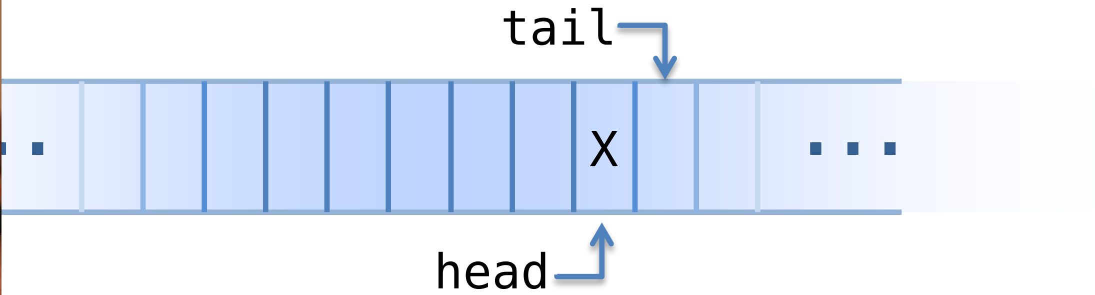


no “temporary suspension” of invariants

```
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

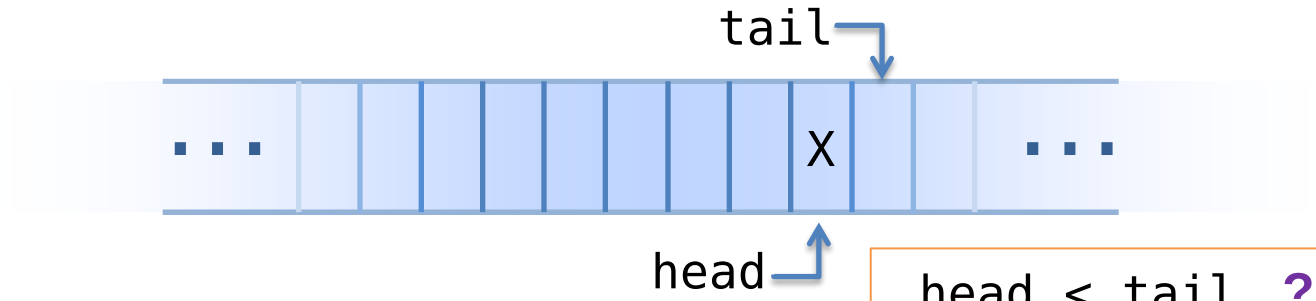
```
void push(int val)
{
    buffer[tail++] = val;
}
```

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

X...

A

B



head < tail ?

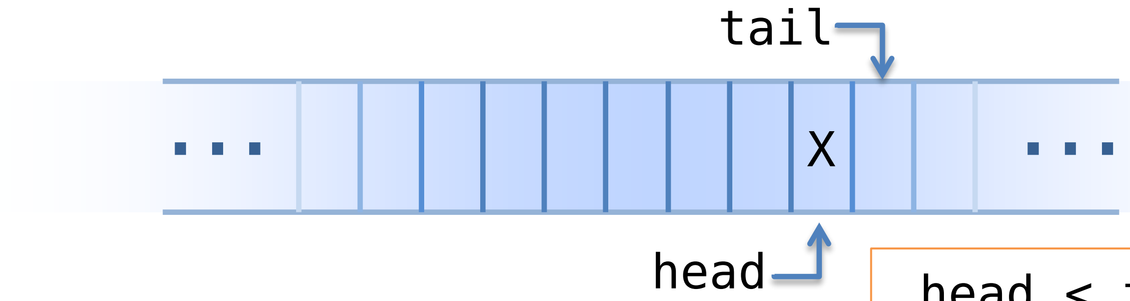
```
void push(int val)
{
    buffer[tail++] = val;
}
```

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

X...

A

B



head < tail ?

ensure tail is always increasing

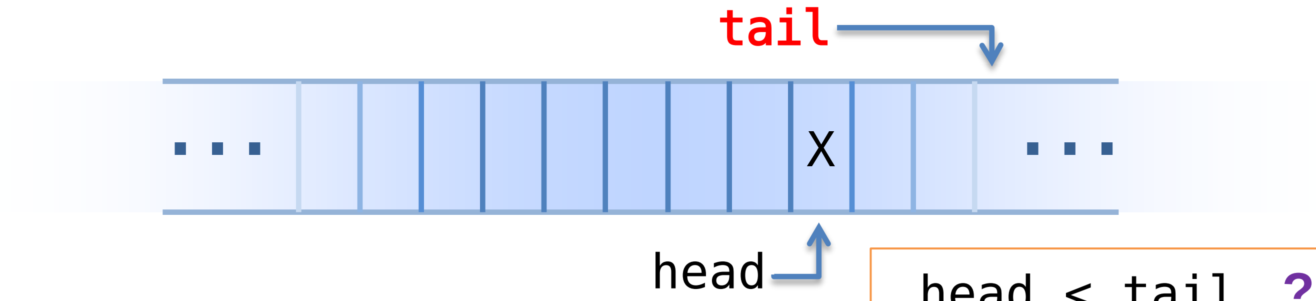
```
void push(int val)
{
    buffer[tail++] = val;
}
```

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

X...

A

B



head < tail ?

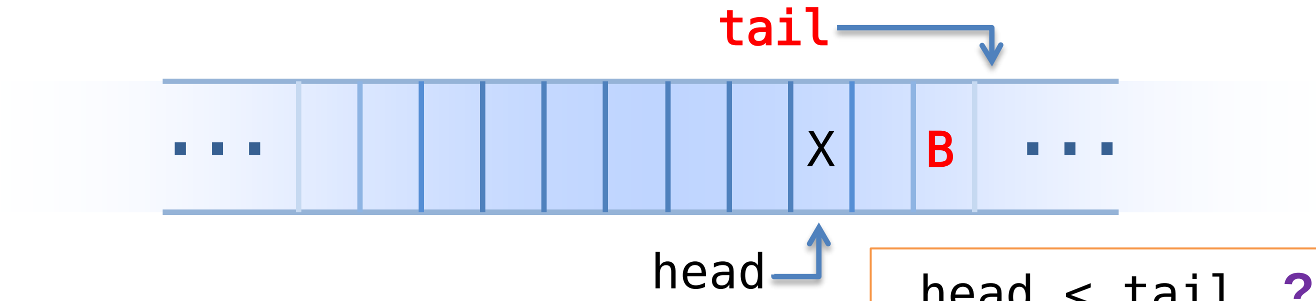
```
void push(int val)
{
    buffer[tail++] = val;
}
```

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

X...

A

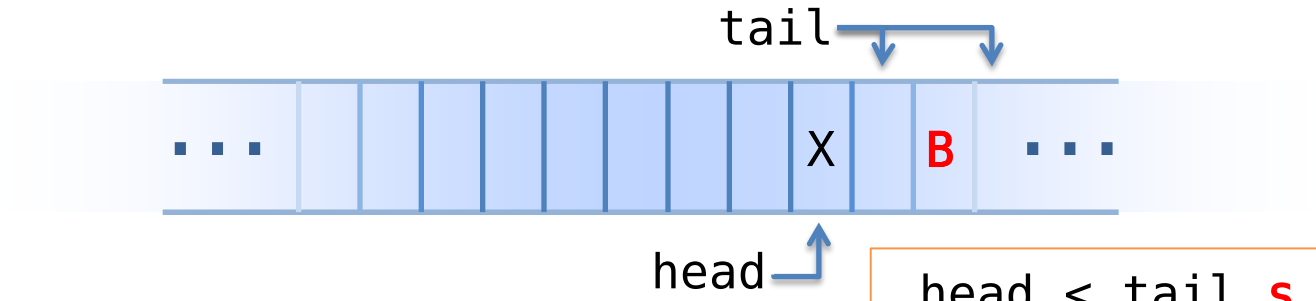
B



head < tail ?

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

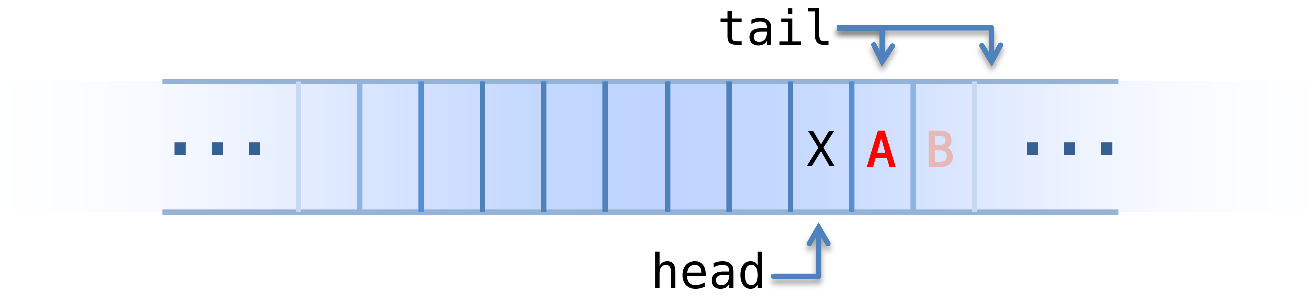
```
class Queue {
    int buffer[SZ];
    atomic<size_t> head;
    struct {   atomic<size_t> s;
               atomic<size_t> e;
    } tail;
};
```



head < tail.s ?

```
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 = ???
    }
}
```

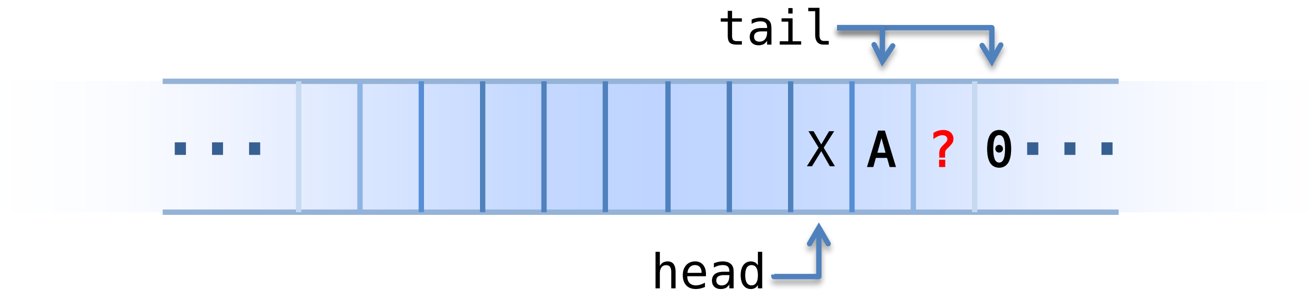
```
class Queue {
    int buffer[SZ];
    atomic<size_t> head;
    struct {    atomic<size_t> s;
                atomic<size_t> e;
    } tail;
};
```

Compromise…

Queue of int -> Queue of int != 0

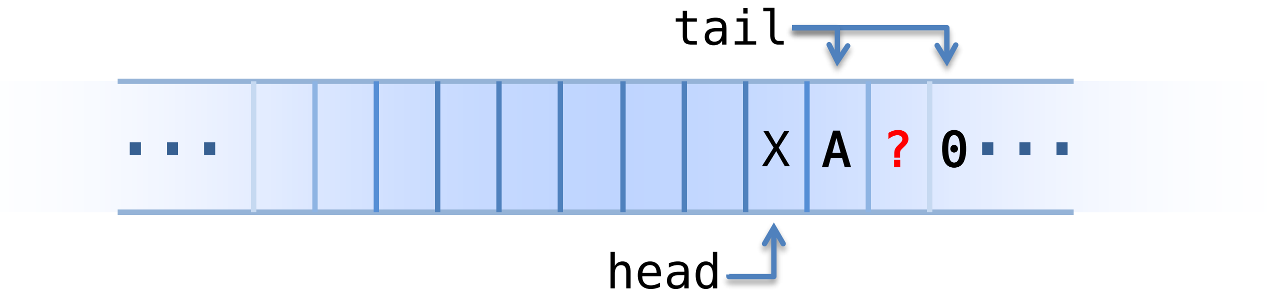
```
void push(int val) {
    size_t tmp = tail.e++;
    buffer[tmp] = val;
    if(tmp = tail.s) {
        do
            tail.s++;
        while (buffer[tail.s]);
    }
}
```

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



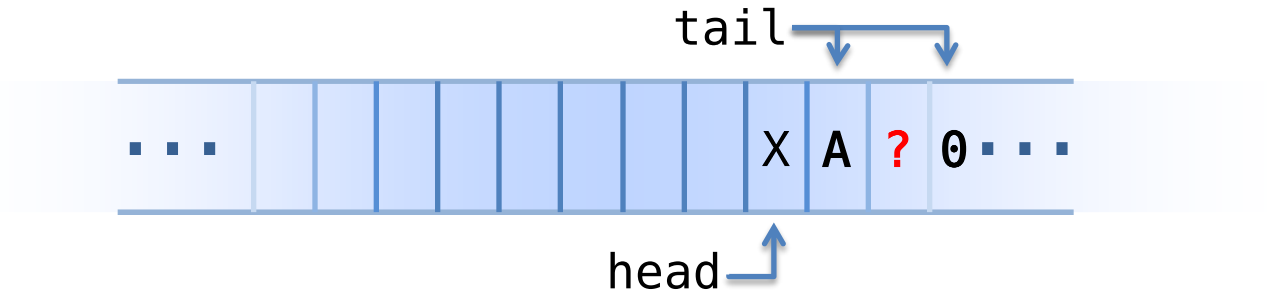
```
void push(int val) {
    size_t tmp = tail.e++;
    buffer[tmp] = val;
    if(tmp = tail.s) {
        do
            tail.s++;
        while (buffer[tail.s]);
    }
}
```

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



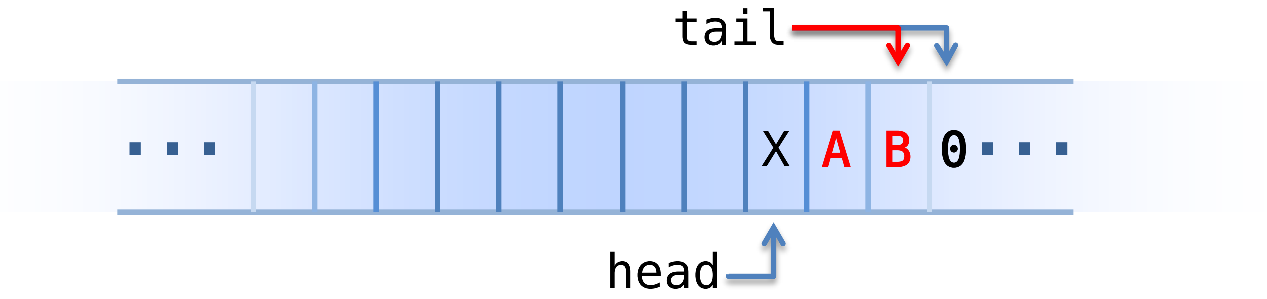
```
void push(int val) {
    size_t tmp = tail.e++;
    buffer[tmp] = val;
    if(tmp = tail.s) {
        do
            tail.s++;
        while (buffer[tail.s]);
    }
}
```

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



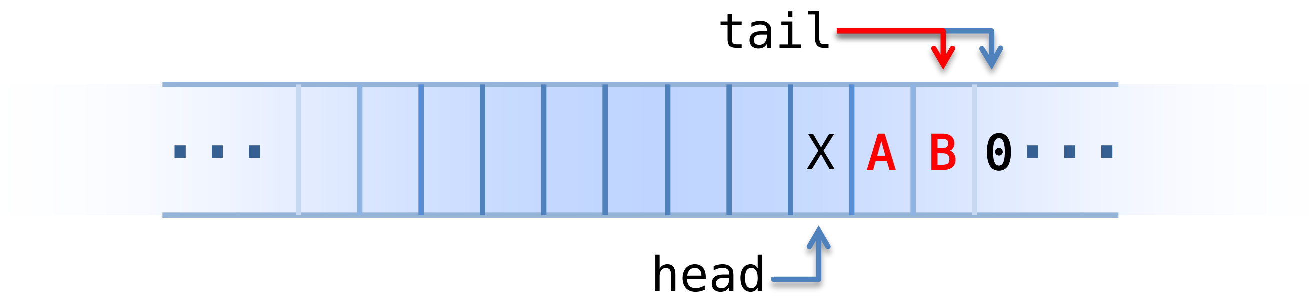
```
    void push(int val) {
        size_t tmp = tail.e++;
        buffer[tmp] = val;
B →     if(tmp = tail.s) {  THEN
            do
A →             tail.s++;
            while (buffer[tail.s]);
        }
    }
```

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



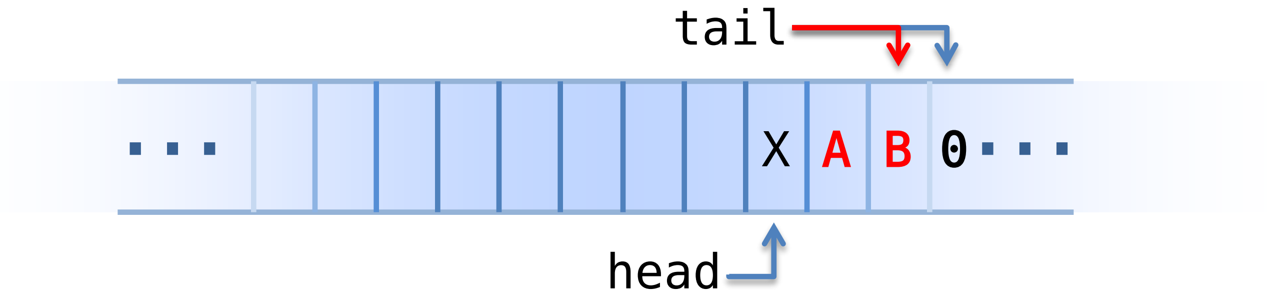
```
  void push(int val) {
      size_t tmp = tail.e++;
      buffer[tmp] = val;
B →   if(tmp = tail.s) {   THEN
          do
A →           CAS(tail.s,tmp,tmp+1);
          while (buffer[++tmp]);
      }
  }
```

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



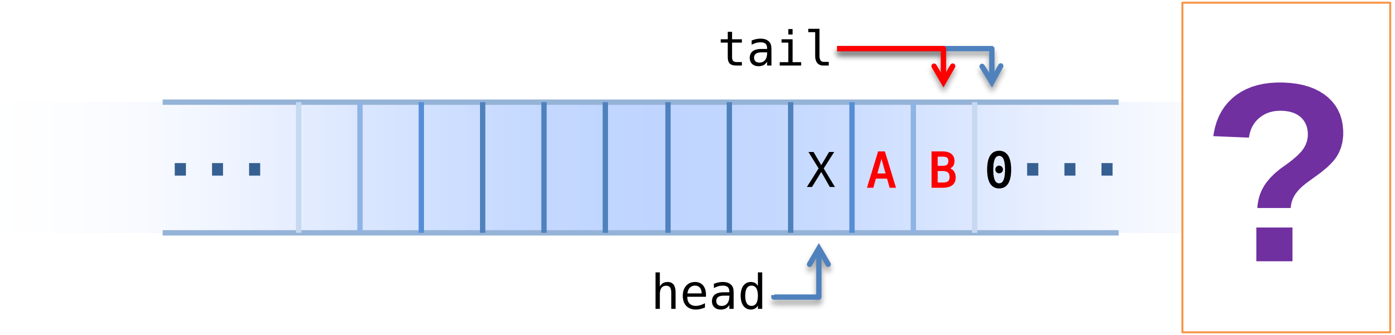
```
    void push(int val) {
        size_t tmp = tail.e++;
        buffer[tmp] = val;
        bool r;
        do
B
A →         r = CAS(tail.s,tmp,tmp+1);
        while (r && buffer[++tmp]);
    }
```

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



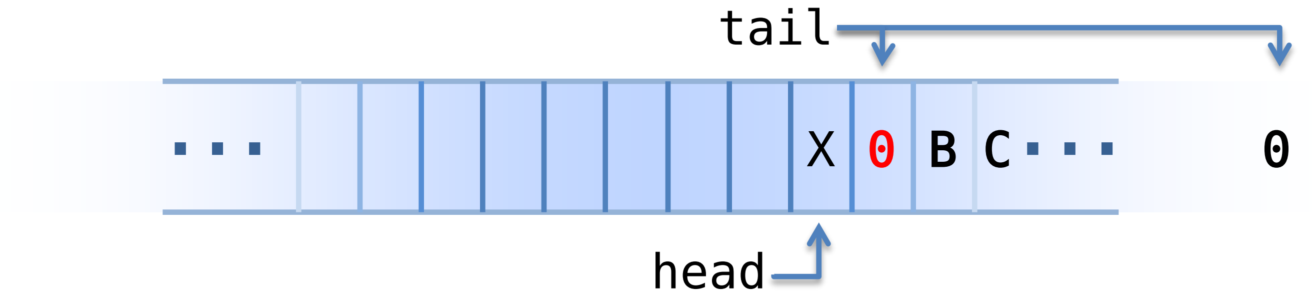
```
void push(int val) {
    size_t tmp = tail.e++;
    buffer[tmp] = val;
    bool r;
    do
        r = CAS(tail.s,tmp,tmp+1);
    while (r && buffer[++tmp]);
}
```

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



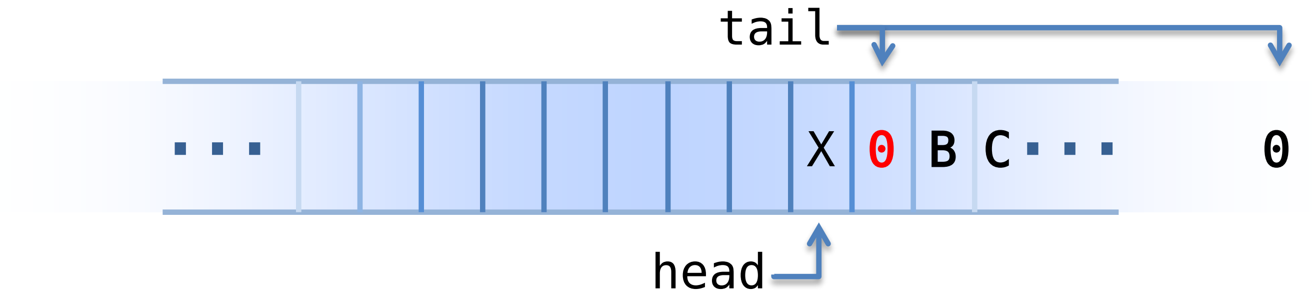
```
void push(int val) {
    size_t tmp = tail.e++;
    buffer[tmp] = val;
    bool r;
    do
        r = CAS(tail.s,tmp,tmp+1);
    while (r && buffer[++tmp]);
}
```

```
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;
    bool r;
    do
        r = CAS(tail.s,tmp,tmp+1);
    while (r && buffer[++tmp]);
}
```

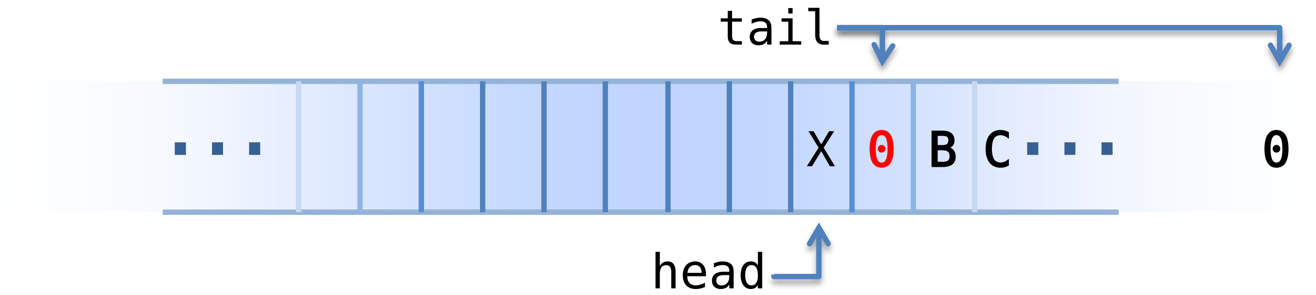
```
class Queue {
    atomic<int> buffer[SZ];
    atomic<size_t> head;
    struct {    atomic<size_t> s;
                atomic<size_t> e;
    } tail;
};
```



!= 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]);
}
```

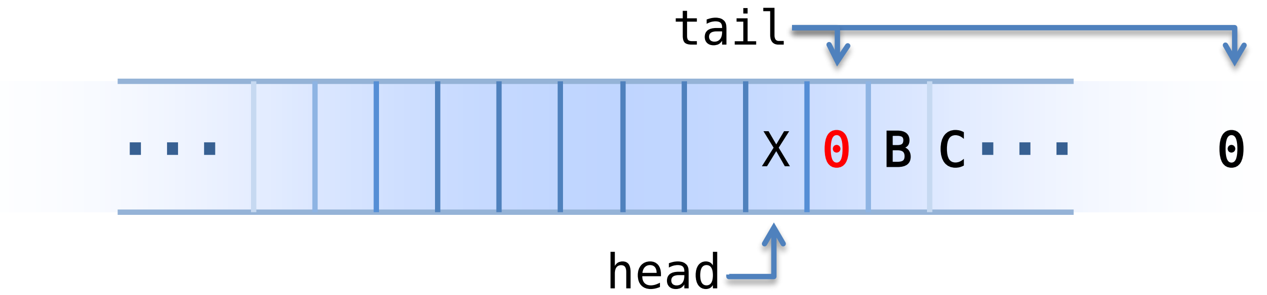
```
class Queue {
    atomic<int> buffer[SZ];
    atomic<size_t> head;
    struct {    atomic<size_t> s;
                atomic<size_t> e;
    } tail;
};
```



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;
    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;
};
```

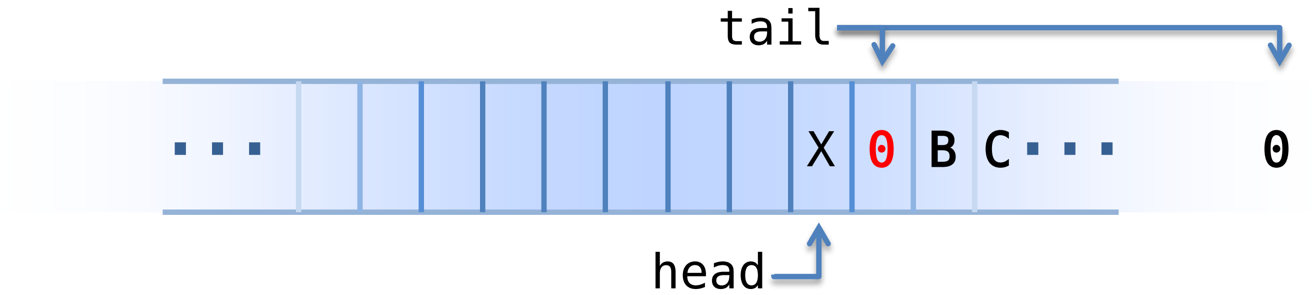


If I suspended a certain thread at the worst time, for a long time or forever, do bad things happen?
Yes -> not lockfree.

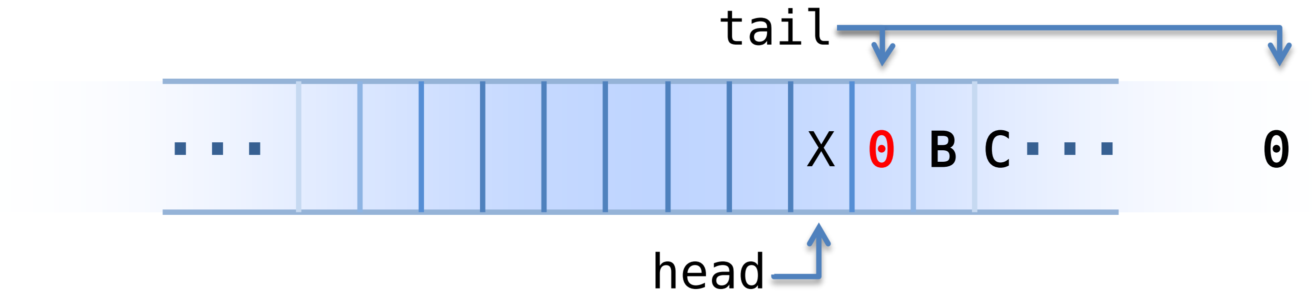
BlackBerry
Blac
HERE
TO
Building C

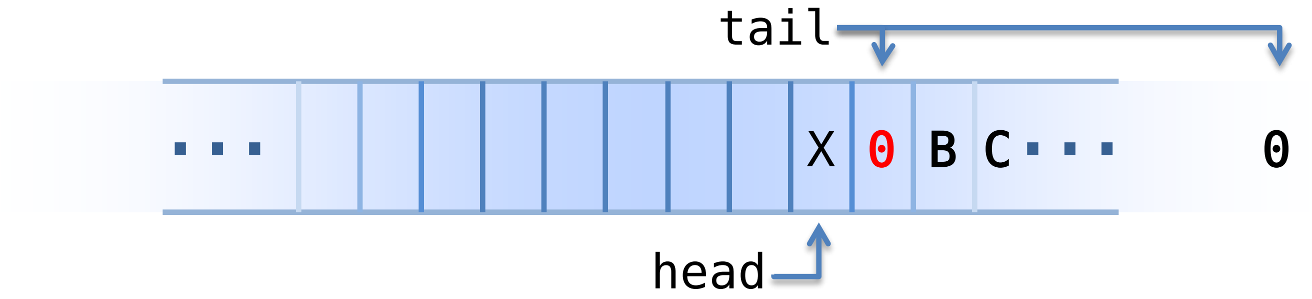
```
void push(int val) {
    size_t tmp = tail.e++;
    buffer[tmp] = val;
    bool r;
    do
        r = CAS(tail.s,tmp,tmp+1);
    while (r && buffer[++tmp]);
}
```

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

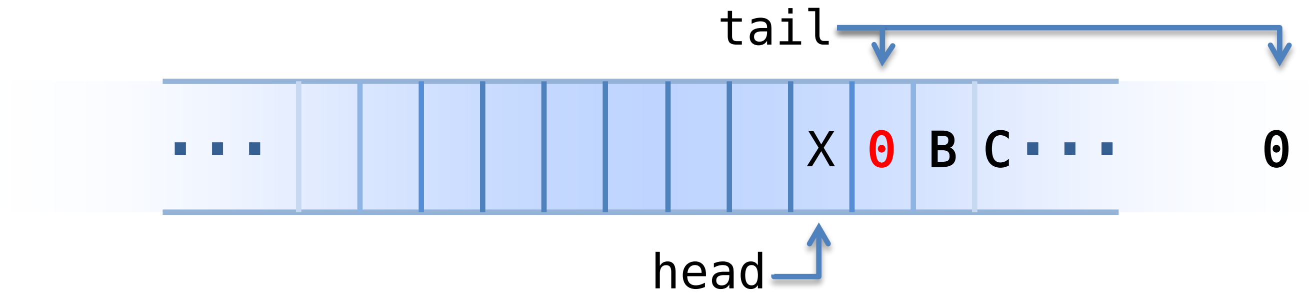


An algorithm is ***lock-free*** if at all times **at least one thread** is guaranteed to be **making progress**.

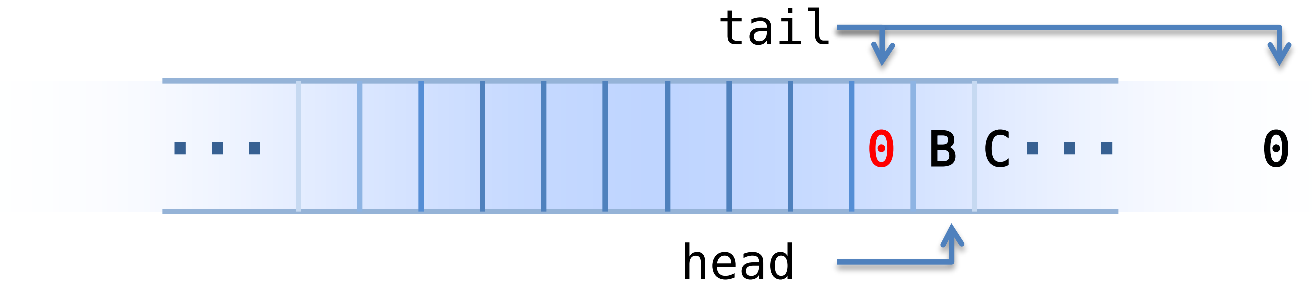

tail
...
X
0
B
C...
0
head



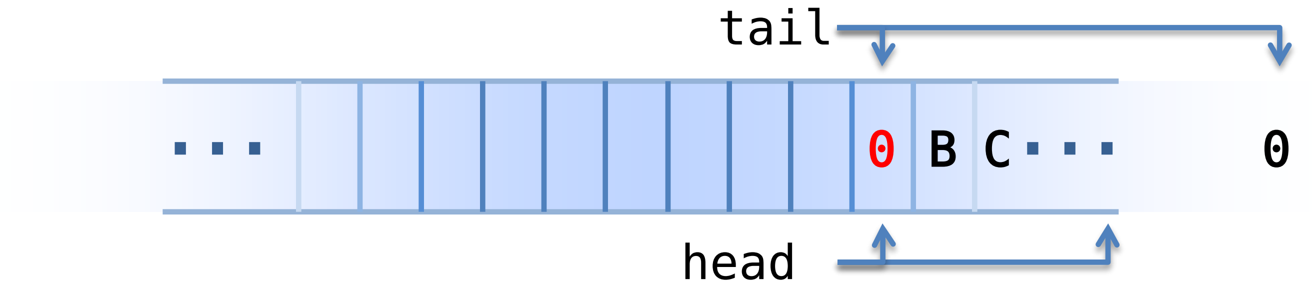
don’t want to wait



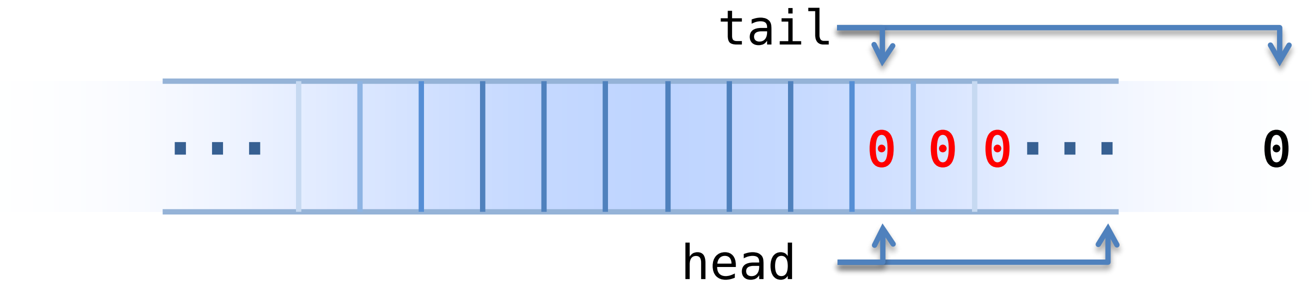
don't wait



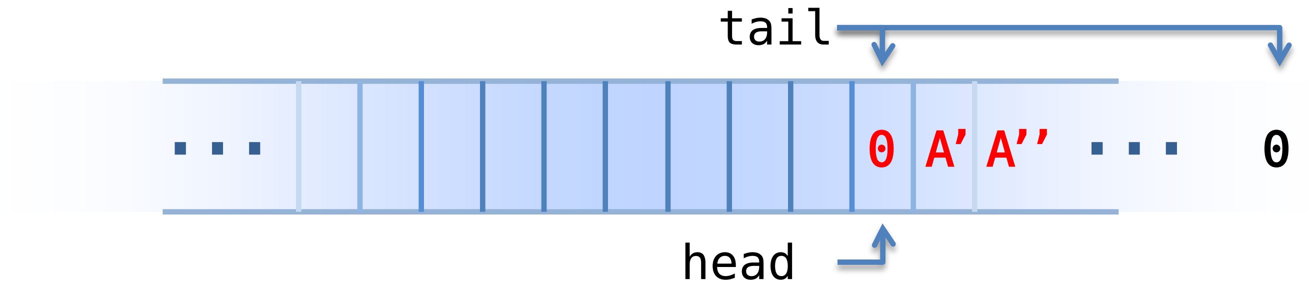
don’t wait! (?)



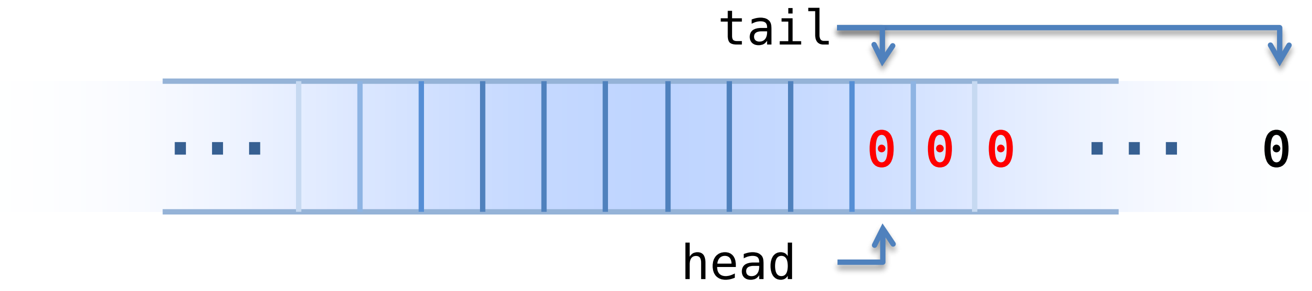
come back later (when?)



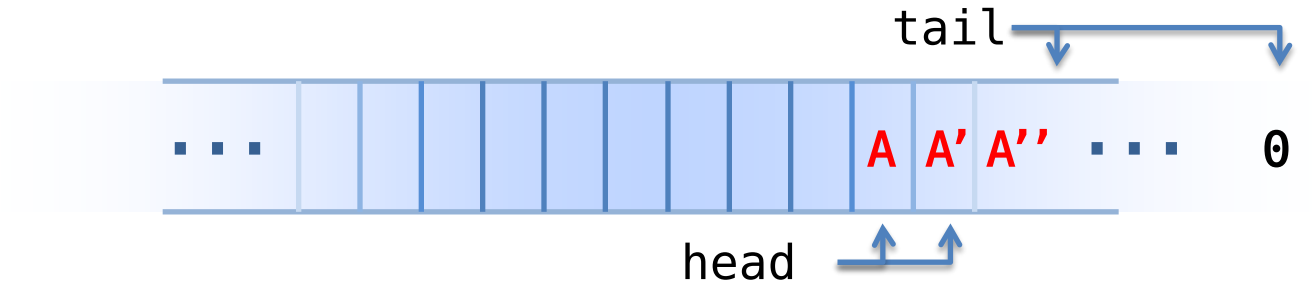
spin?



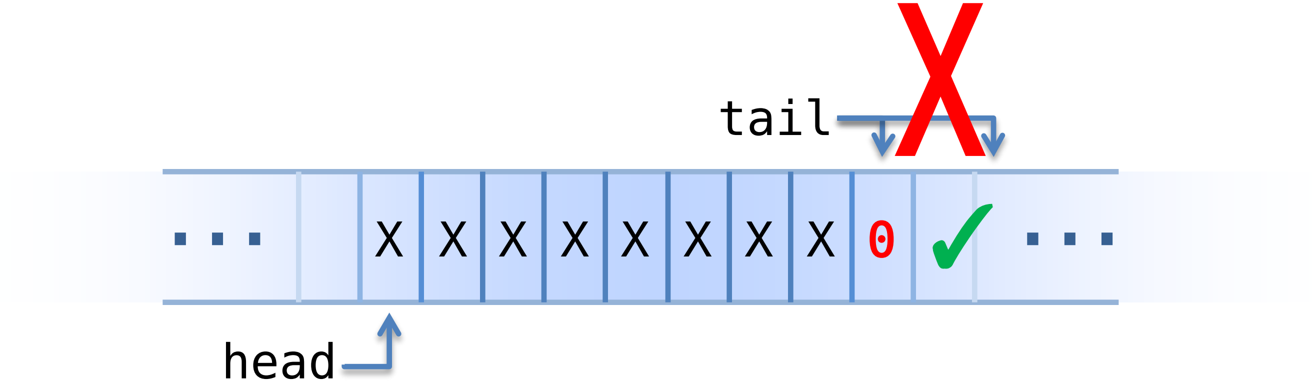
pop order ?

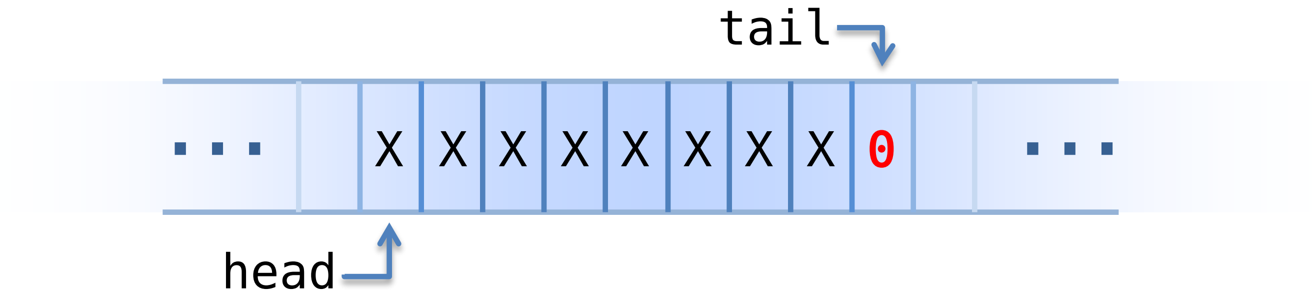


pop order ?

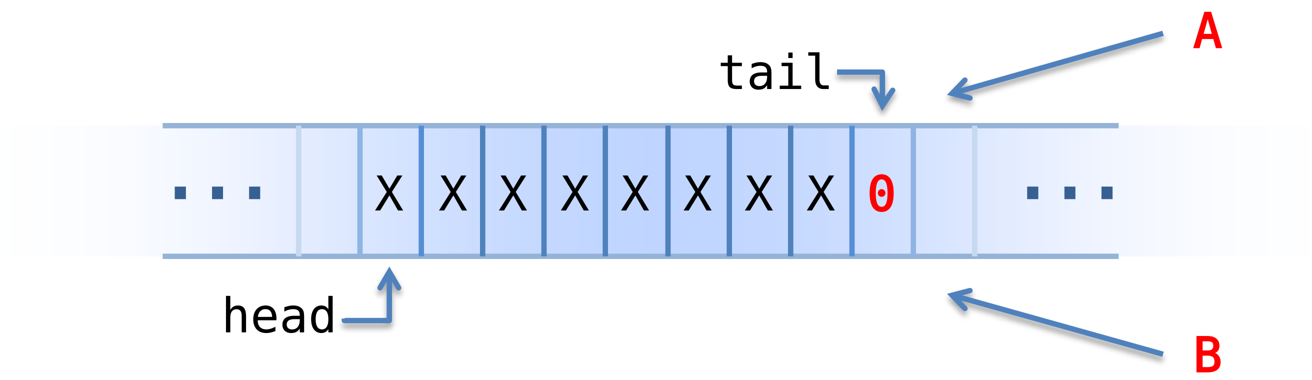


pop order ?

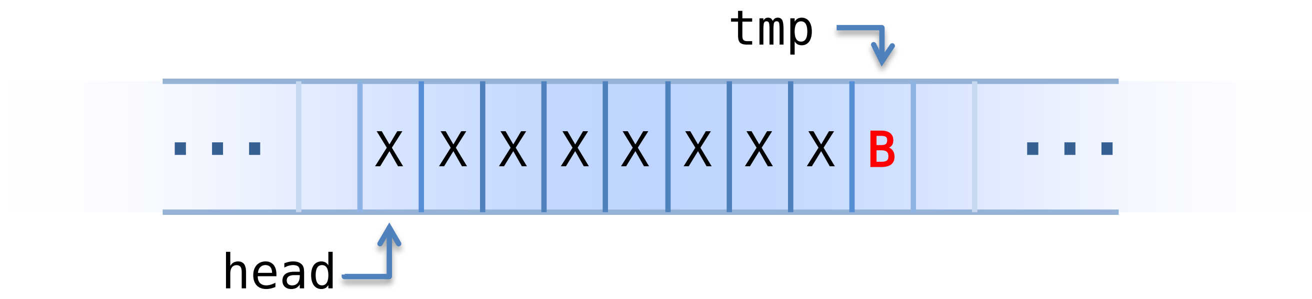

tail
...
X
X
X
X
X
X
X
X
0
✓
...
head


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

CAS(buffer[tail], 0, val)

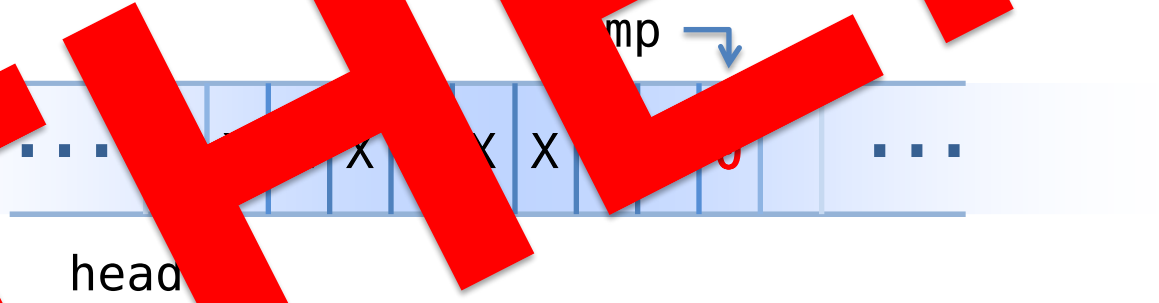


```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], 0, val) );
```



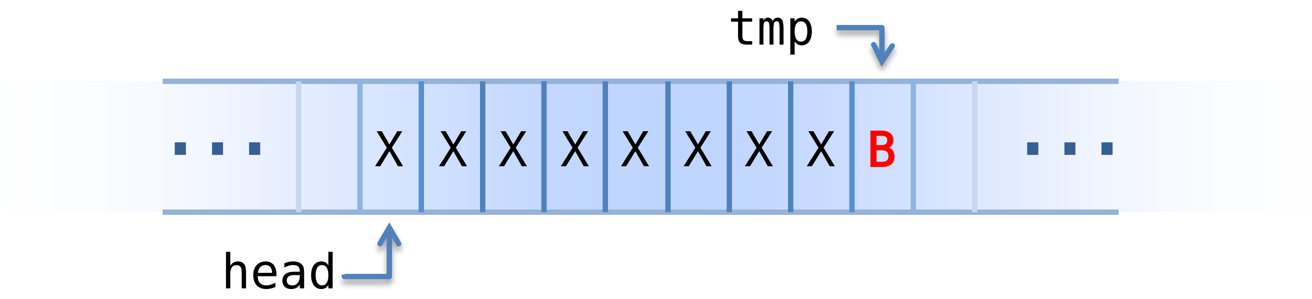
THEN

```
do
    t        tail.load();
wh     ! CAS(bu  er[tm   0, val) );
```



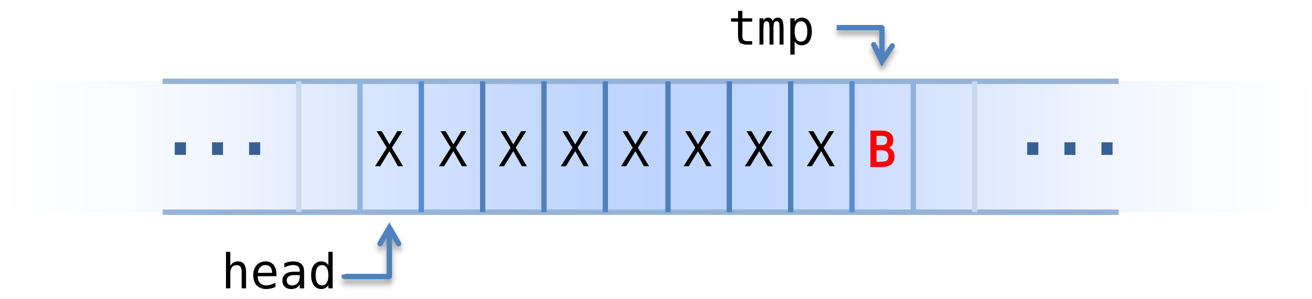
```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], 0, val) );
```

if it fails
THEN try again



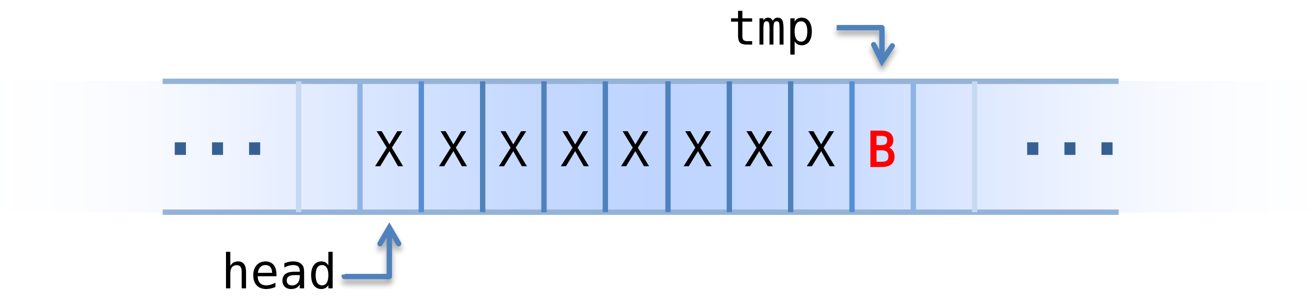
```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], 0, val) );
```

read tail
THEN read buffer



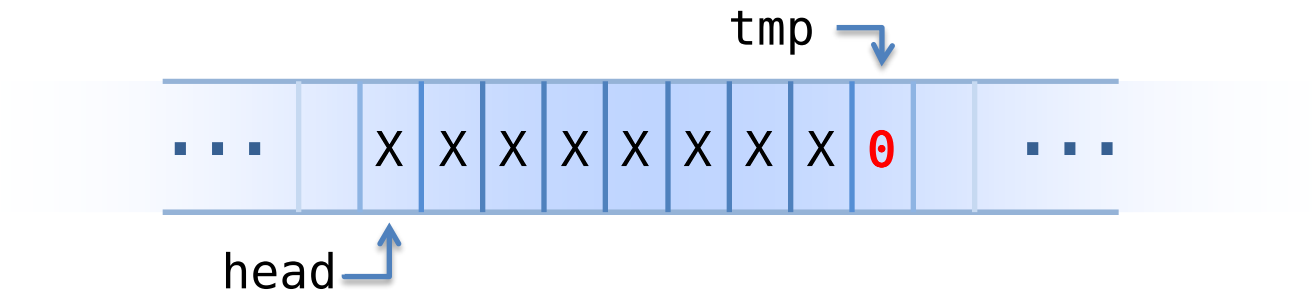
```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], 0, val) );
```

read tail
THEN read buffer



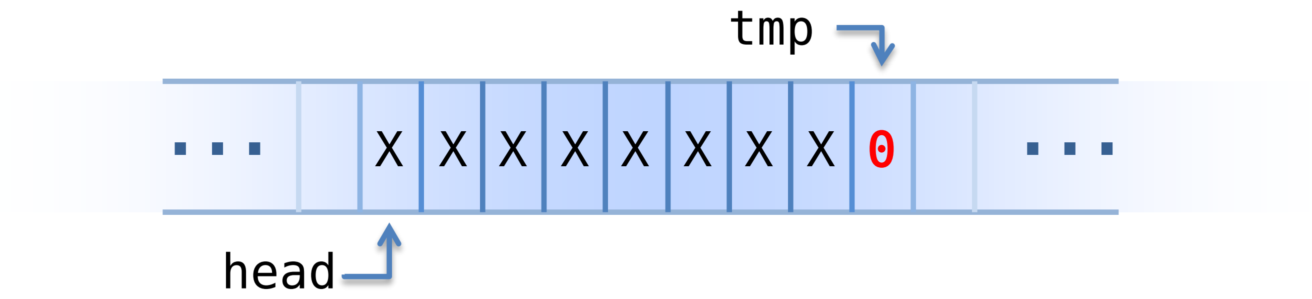
```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], 0, val) );
```

read tail
THEN read buffer



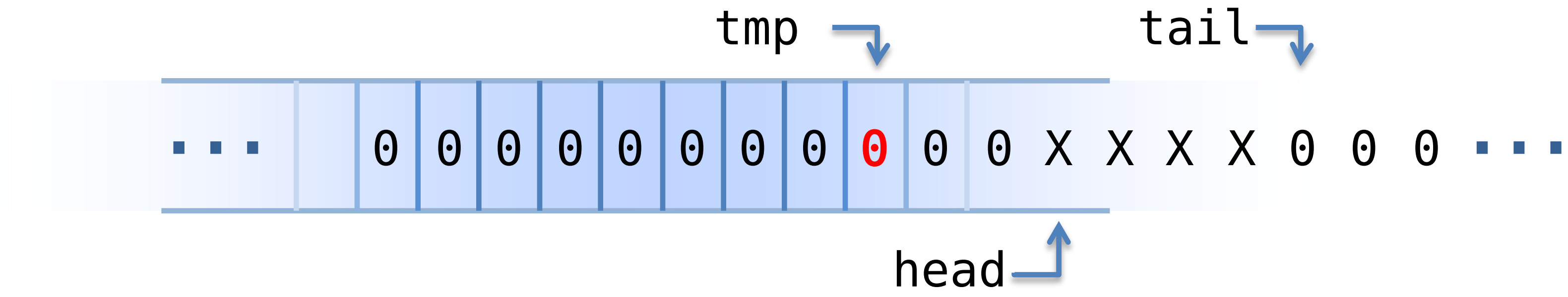
```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], 0, val) );
```

read tail
THEN read buffer



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

read tail
THEN read buffer



tmp

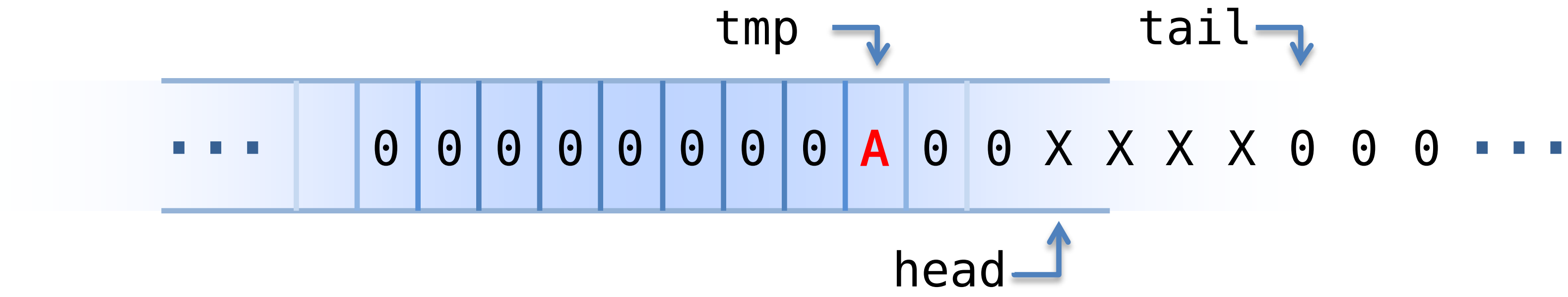
tail

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

head

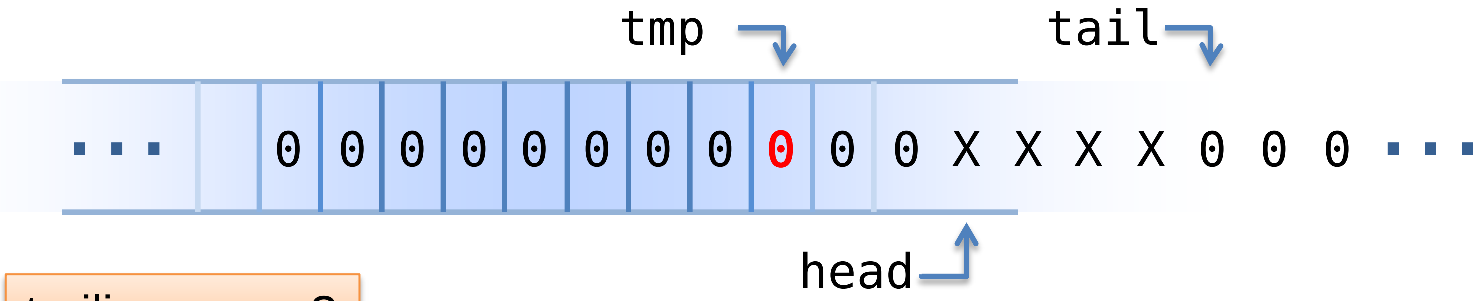
```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], 0, val) );
```

read tail
THEN read buffer



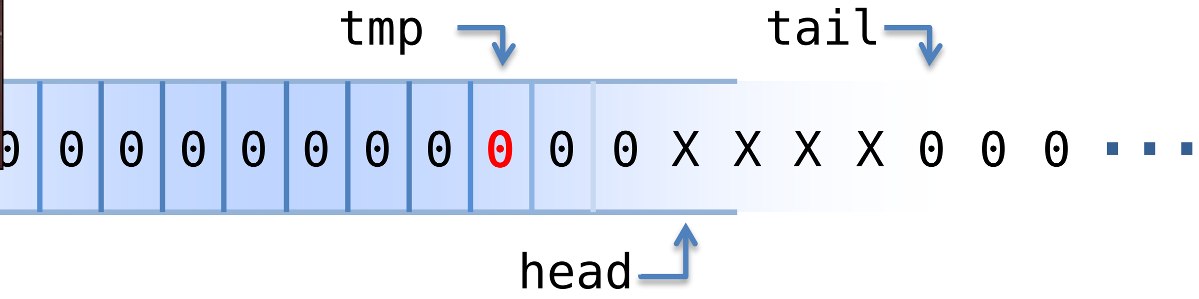
... 0 0 0 0 0 0 0 0 A 0 0 X X X X 0 0 0 ...

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



trailing zeros ?

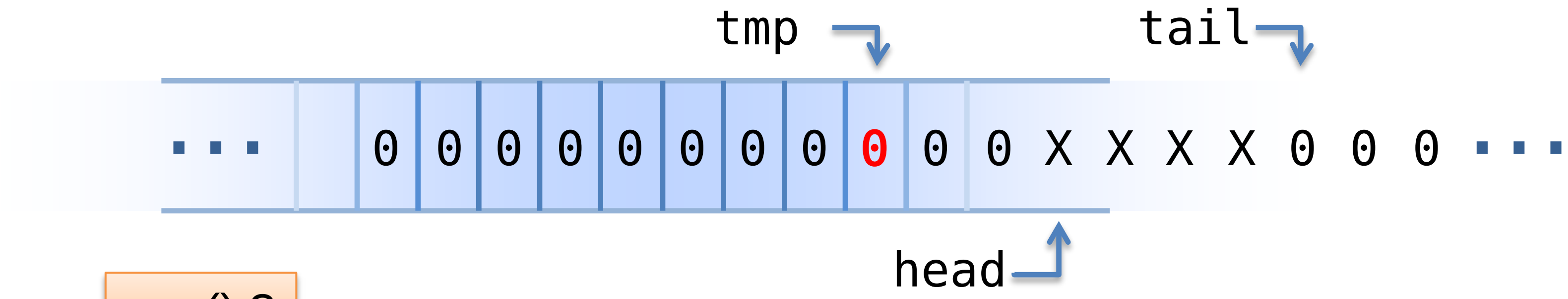
```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], 0, val) );
```



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

pop() ?

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



pop() ?

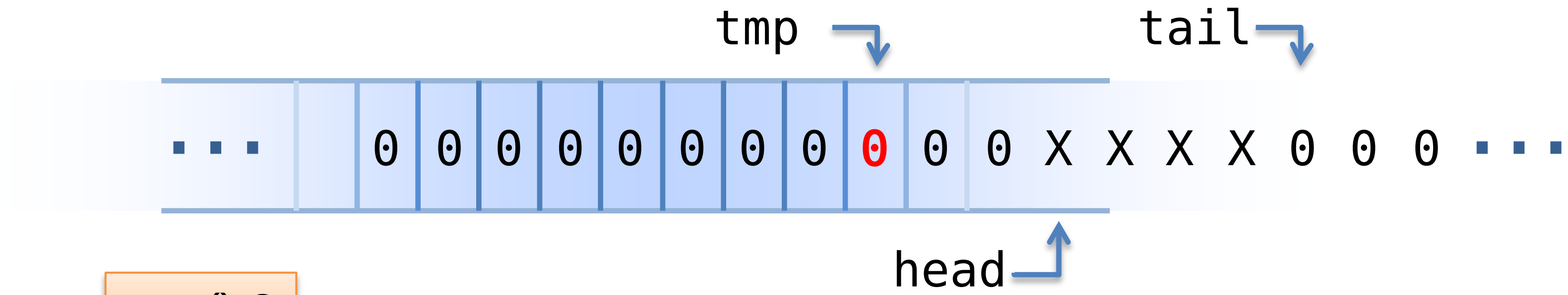
```
do
	tmp = tail.load();
while ( ! CAS(buffer[tmp], 0, val) );
```

Compromise…

0 0 0 ...

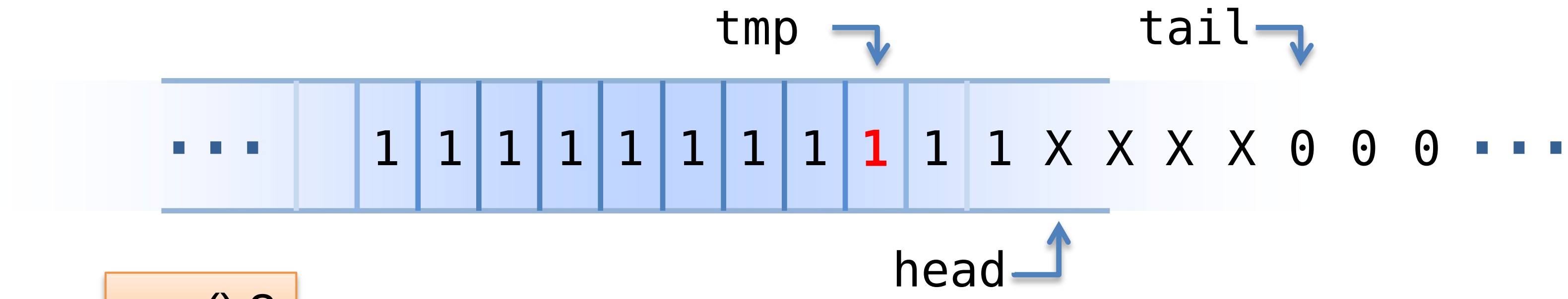
pop() ?

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



pop() ?

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



tmp

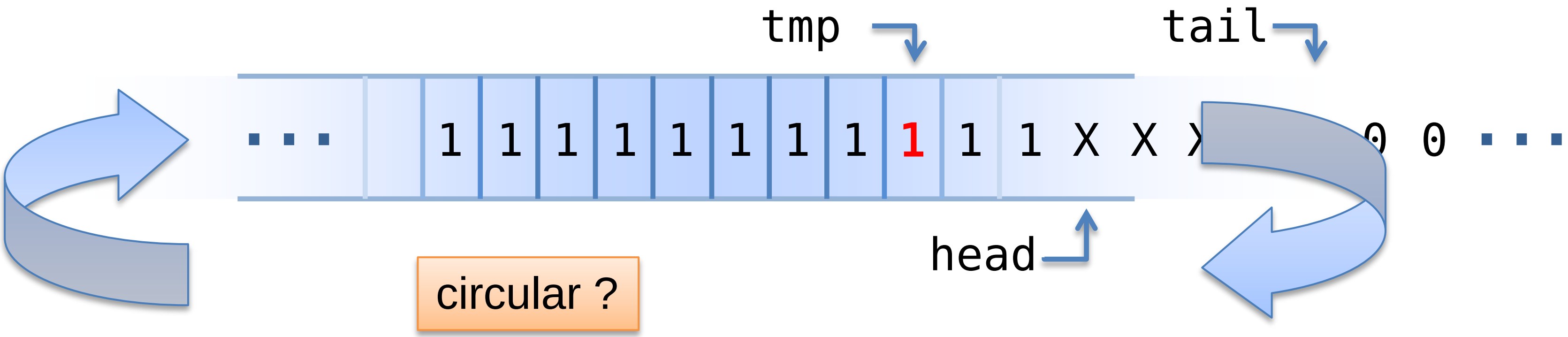
tail

... 1 1 1 1 1 1 1 1 1 1 1 X X X X 0 0 0 ...

head

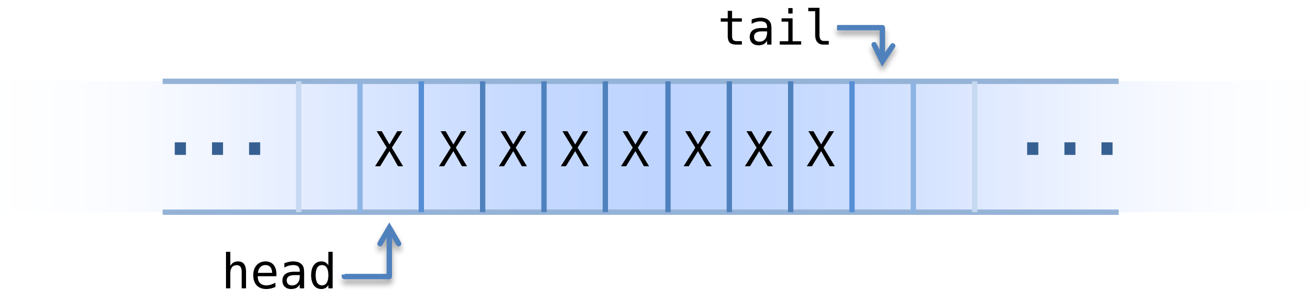
pop() ?

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

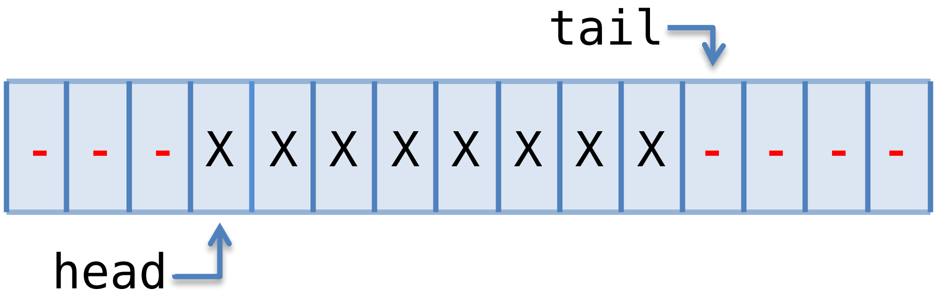


circular ?

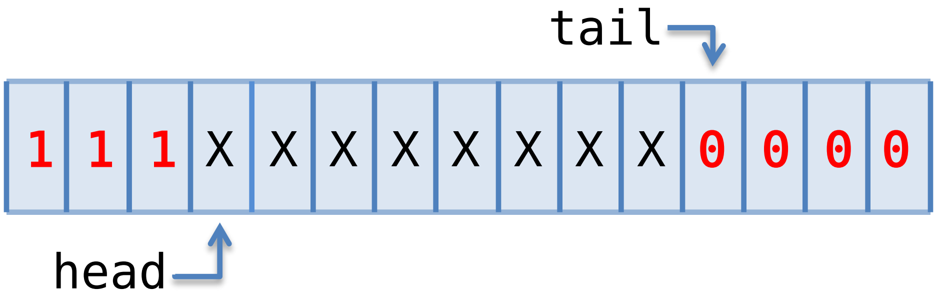
```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], 0, val) );
```



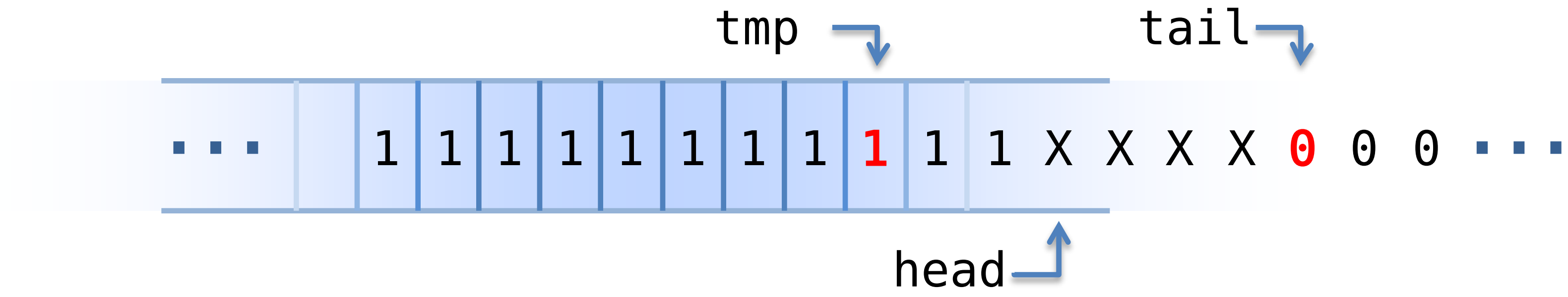
```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], 0, val) );
```



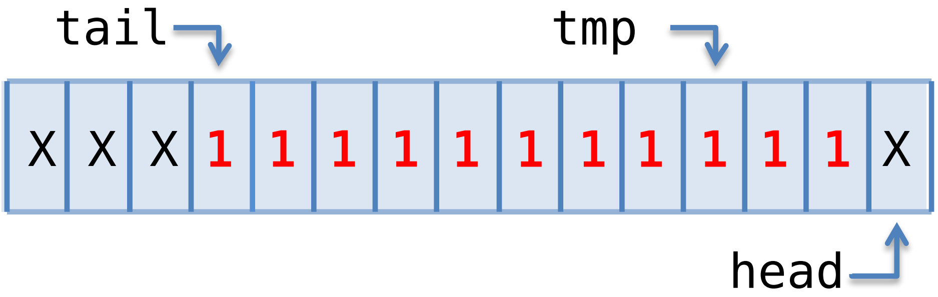
```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], 0, val) );
```



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

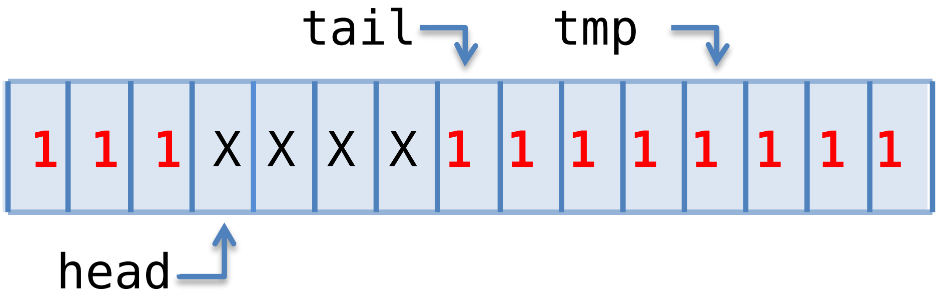


```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], 0, val) );
```



X	X	X	1	1	1	1	1	1	1	1	1	1	1	X

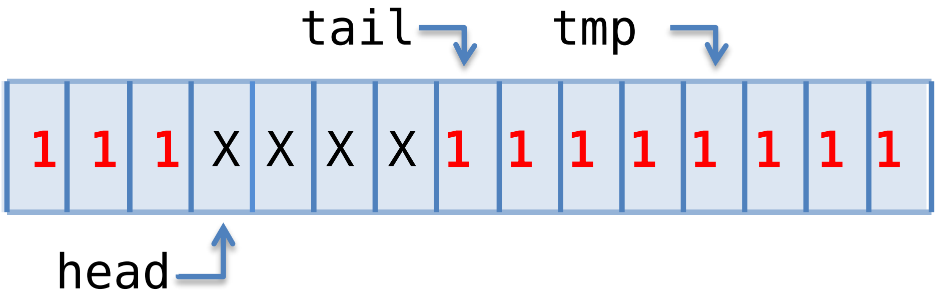
```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], 0, val) );
```



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

Compromise…

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



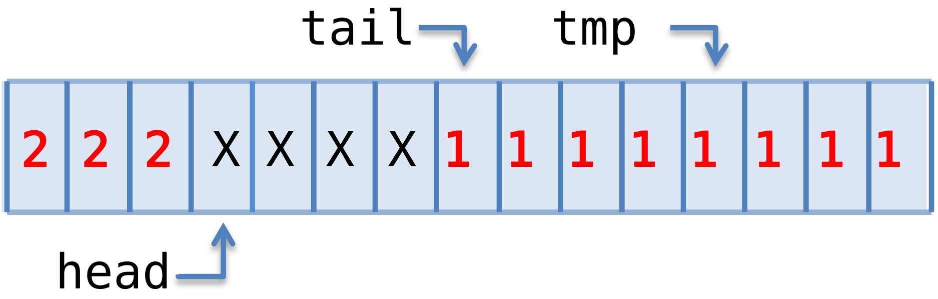
tail

tmp

1	1	1	X	X	X	X	1	1	1	1	1	1	1	1

head

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

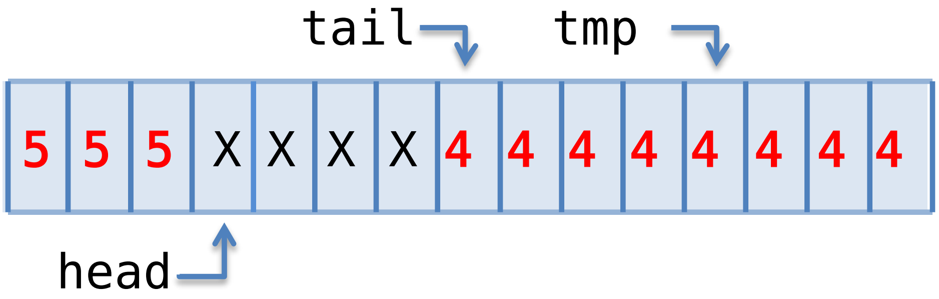


tail tmp

2	2	2	X	X	X	X	1	1	1	1	1	1	1	1

head

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

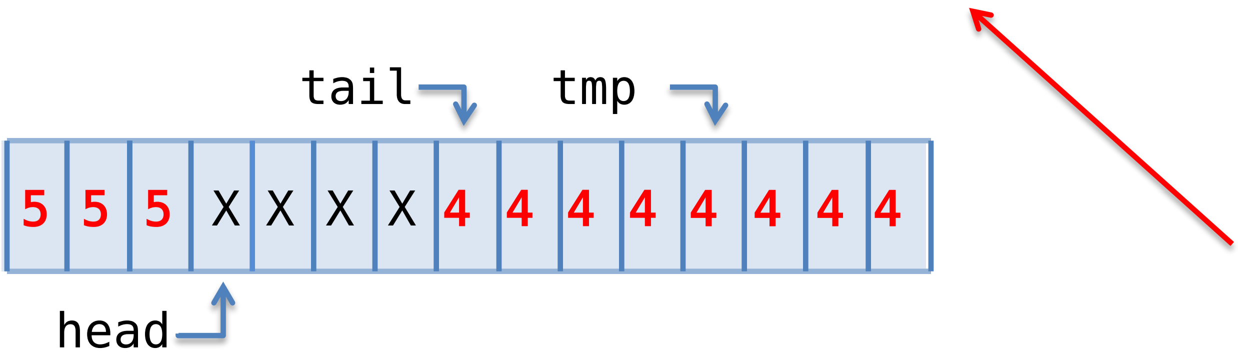


tail tmp

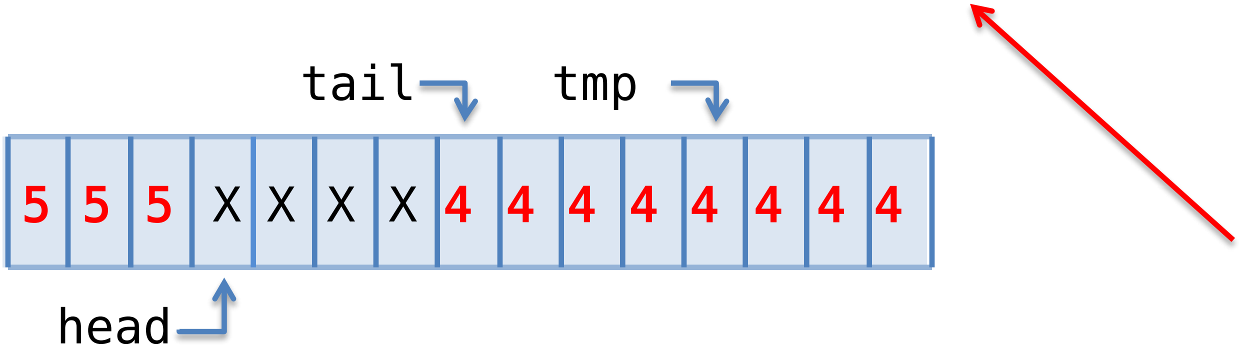
5	5	5	X	X	X	X	4	4	4	4	4	4	4	4

head

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

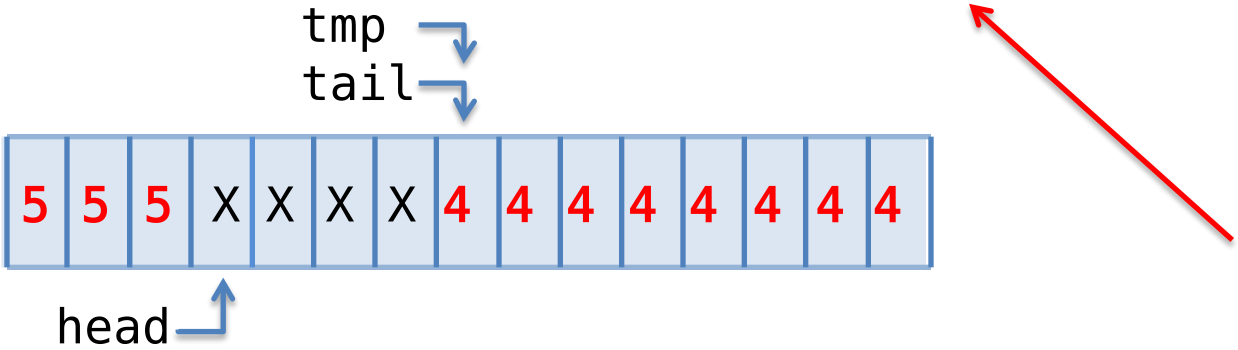


```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], 0, val) );
```



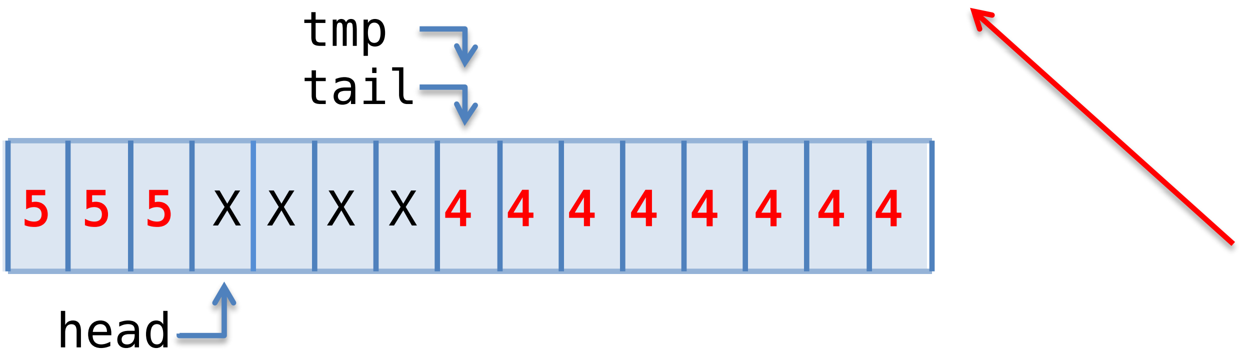
5	5	5	X	X	X	X	4	4	4	4	4	4	4	4

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



5	5	5	X	X	X	X	4	4	4	4	4	4	4	4

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



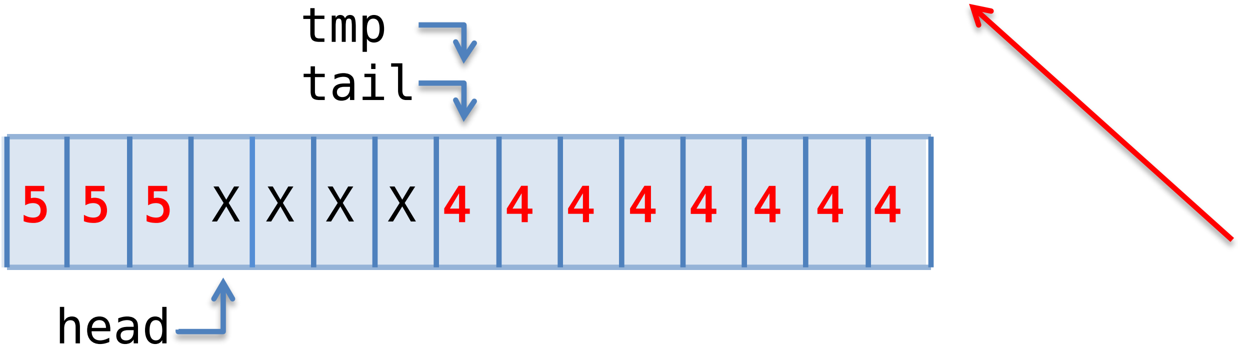
tmp

tail

5	5	5	X	X	X	X	4	4	4	4	4	4	4	4

head

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



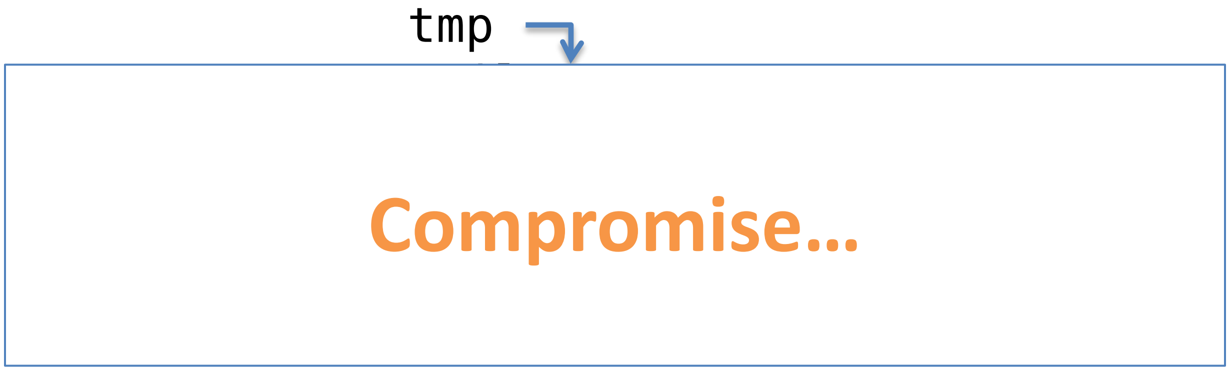
tmp

tail

5	5	5	X	X	X	X	4	4	4	4	4	4	4	4

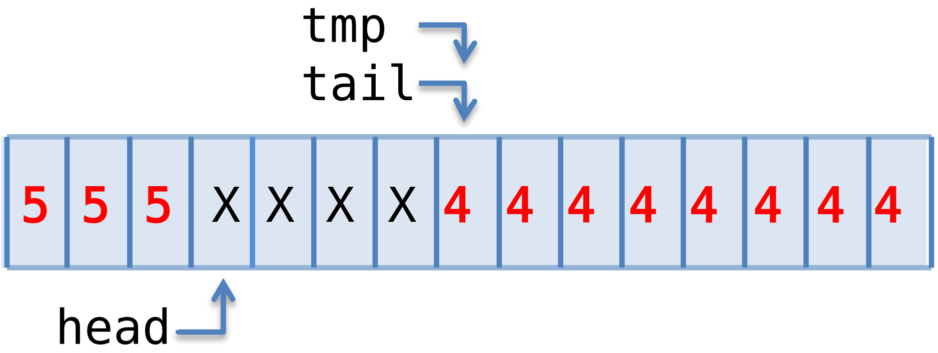
head

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



Compromise…

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



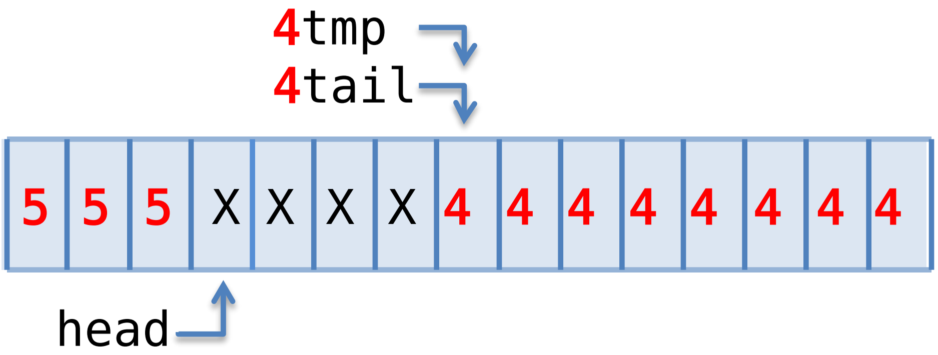
tmp

tail

5	5	5	X	X	X	X	4	4	4	4	4	4	4	4

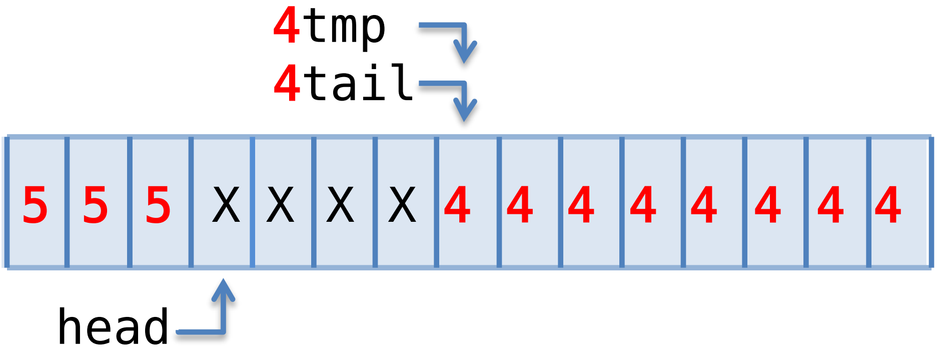
head

```
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
};
```

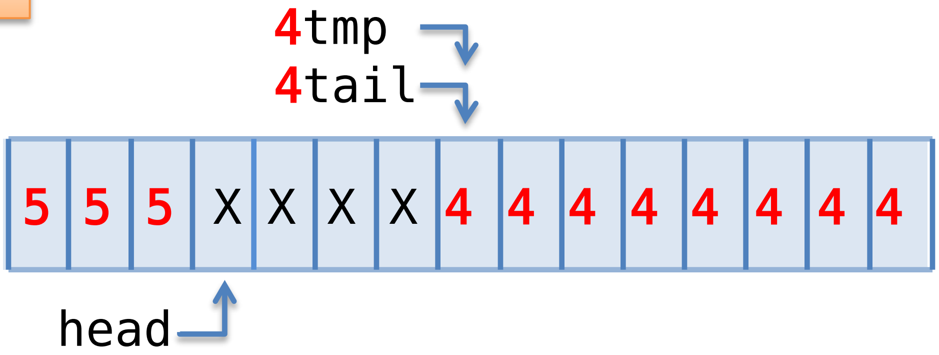
```
do
    tmp = tail.load();
while ( ! CAS(buffer[tmp], gen(tmp), val) );
```



5	5	5	X	X	X	X	4	4	4	4	4	4	4	4

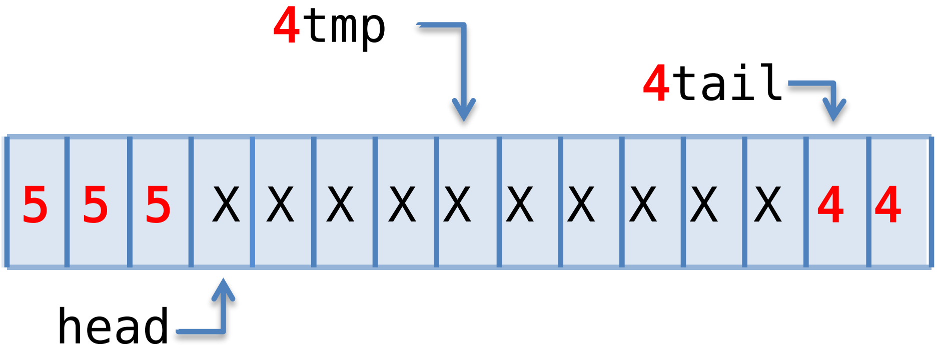
```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], gen(tmp), val) );
```

snapshot



5	5	5	X	X	X	X	4	4	4	4	4	4	4	4

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



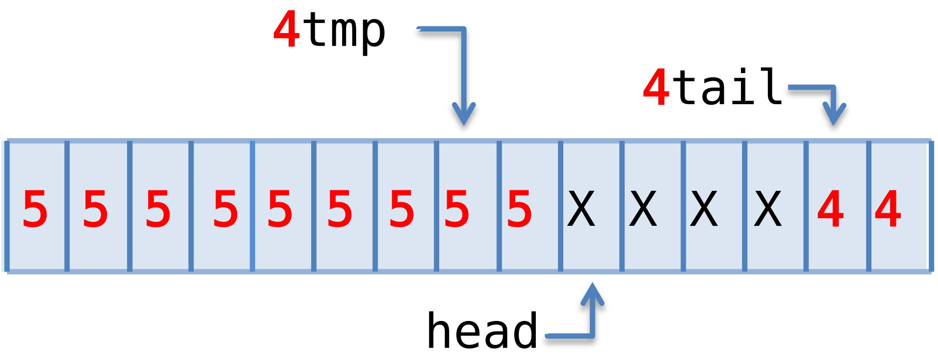
4tmp

4tail

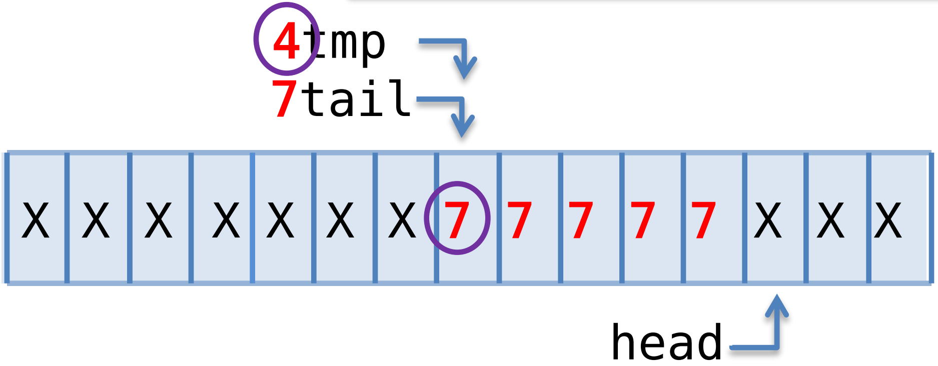
5	5	5	X	X	X	X	X	X	X	X	X	X	4	4

head

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

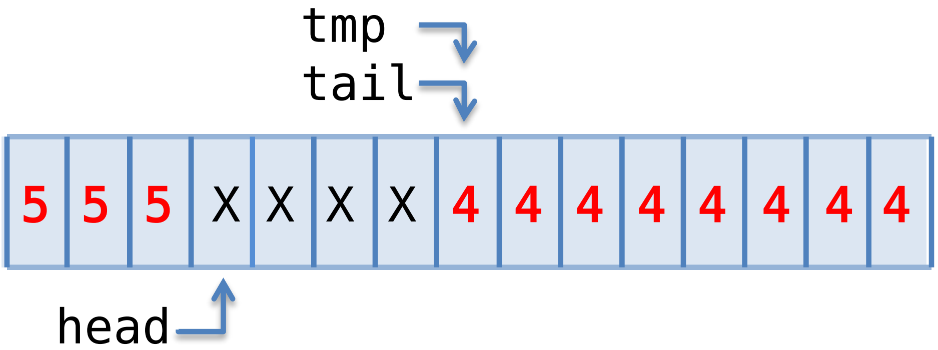


```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], gen(tmp), val) );
```

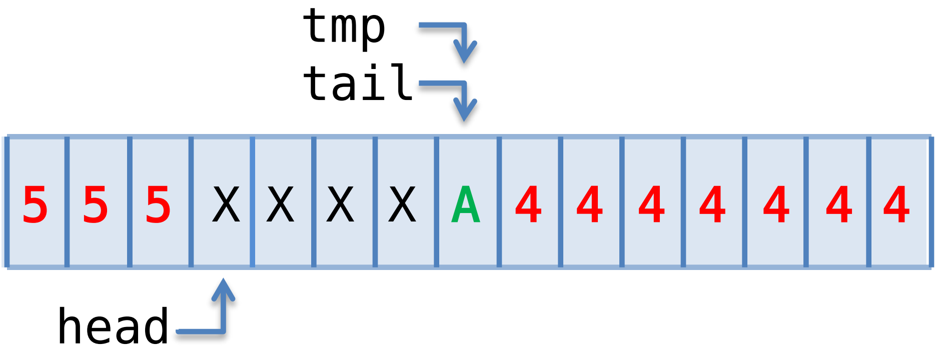


All states are valid states for all lines of code (*)

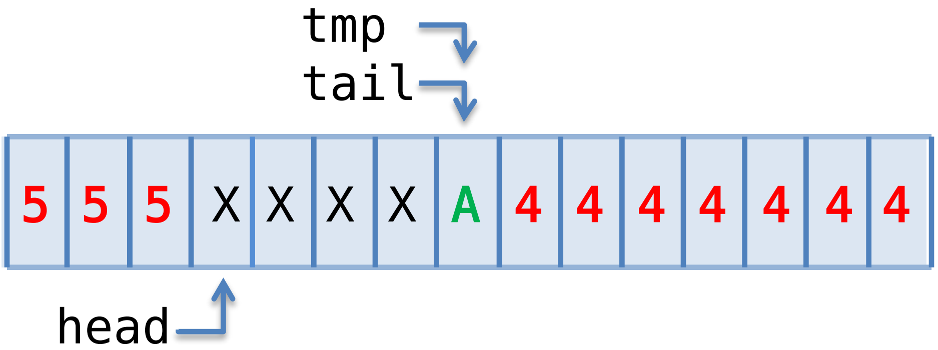
```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], gen(tmp), val) );
```



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

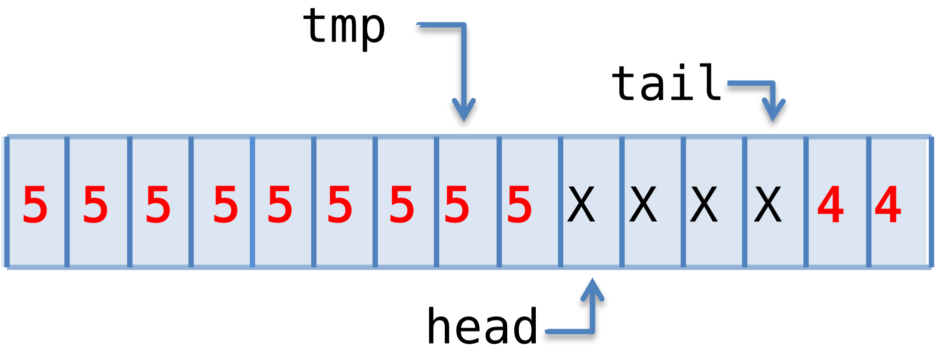


```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], gen(tmp), val) );
tail++; //???
```

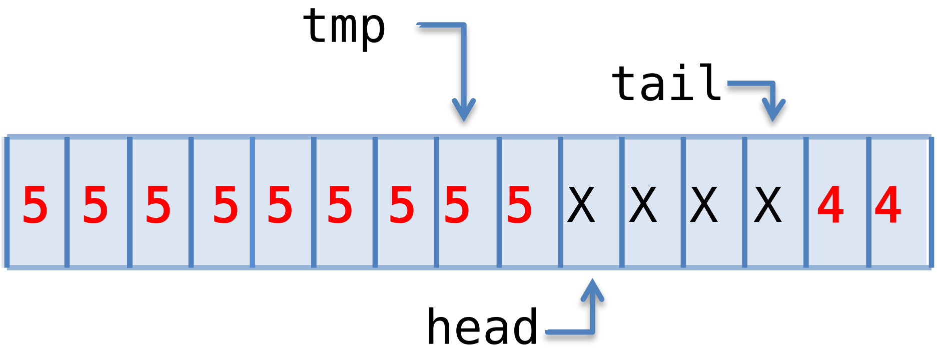


5	5	5	X	X	X	X	A	4	4	4	4	4	4	4

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



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



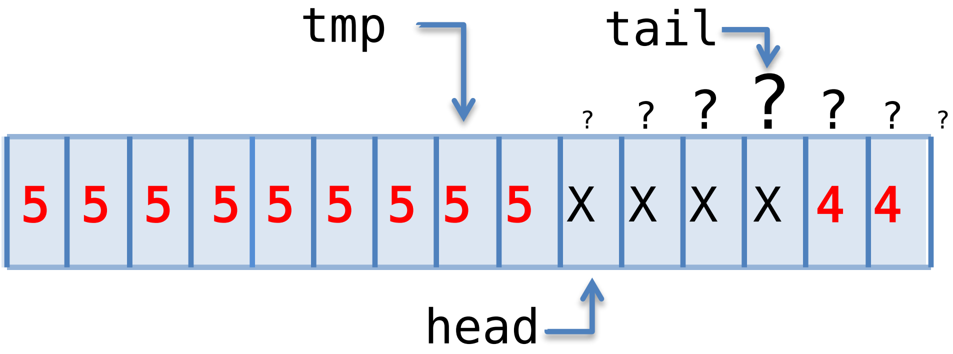
tmp

tail

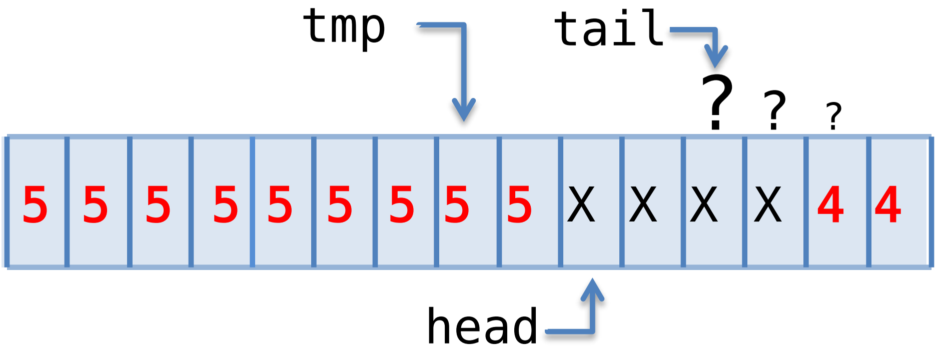
5 5 5 5 5 5 5 5 5 X X X X 4 4

head

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



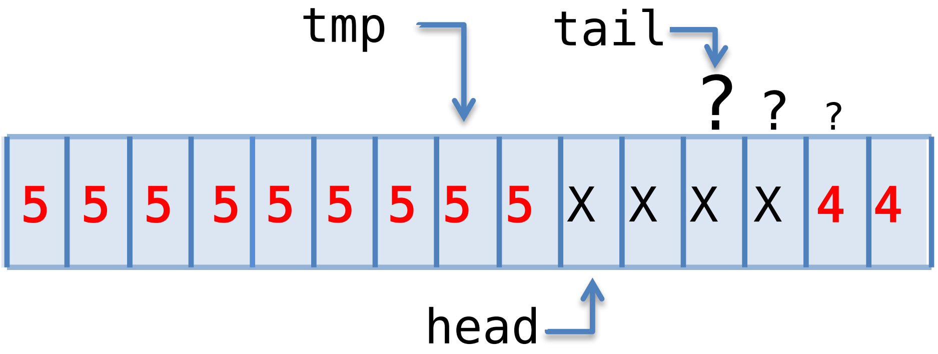
```
do
        tmp = tail.load();
while ( ! CAS(buffer[tmp], gen(tmp), val) );
tail++; // yes!
```



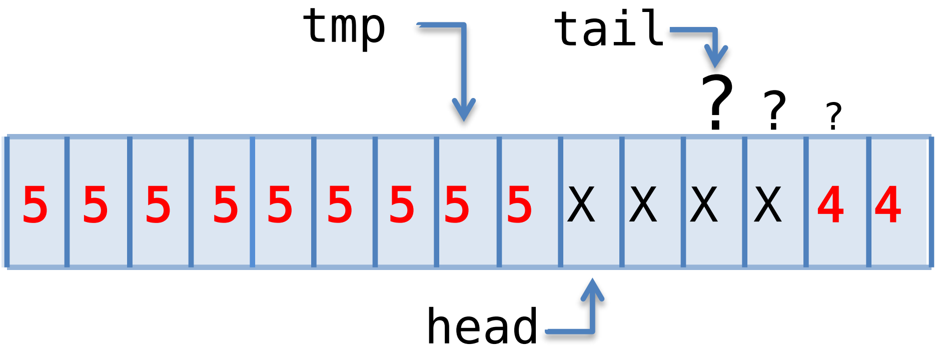
5	5	5	5	5	5	5	5	5	X	X	X	X	4	4

spinlock ?

```
do
        tmp = tail.load();
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!
```

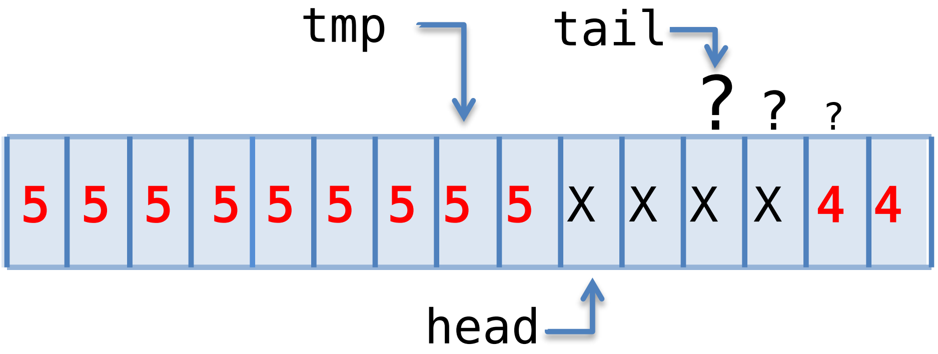


5	5	5	5	5	5	5	5	5	X	X	X	X	4	4

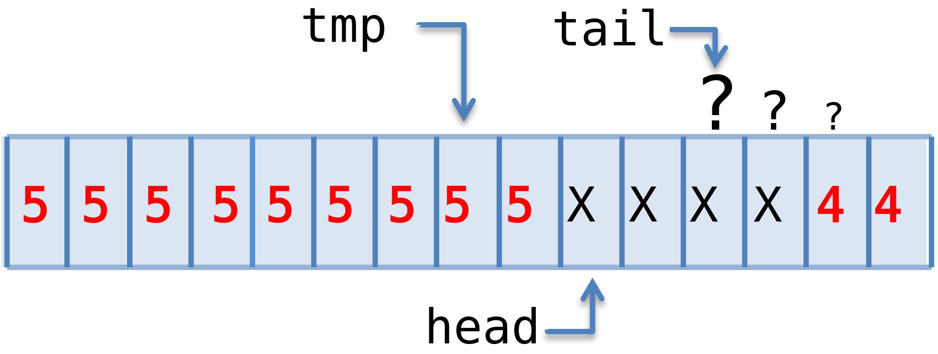
```
do {
        tmp = tail.load();
        while (buffer[tmp] != gen(tmp))
                tmp++;
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
tail++; // yes!
```

Sorry Herb…

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

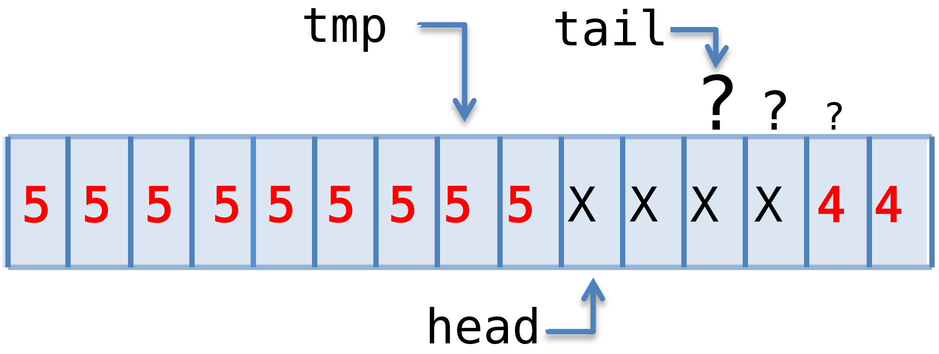


```
do {
        tmp = oldtail = tail.load(relaxed);
        while (buffer[tmp].load(relaxed) != gen(tmp))
                tmp++;
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
CAS(tail, oldtail, tmp+1);
```



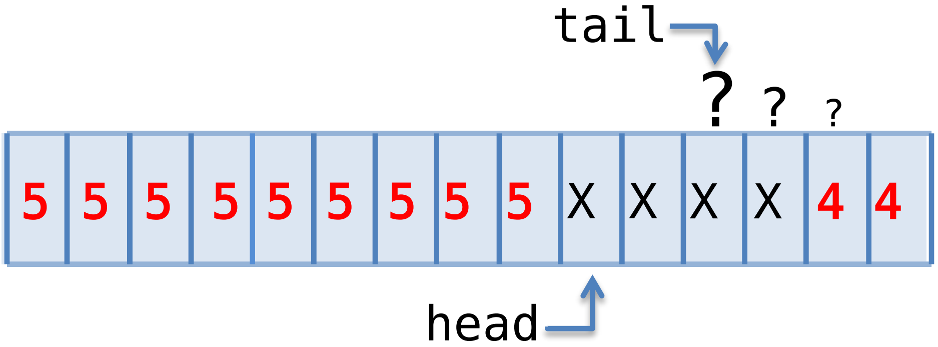
```
do {
        tmp = oldtail = tail.load(relaxed);
        while (buffer[tmp].load(relaxed) != gen(tmp))
                tmp++;
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
CAS(tail, oldtail, tmp+1);
```

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);
```

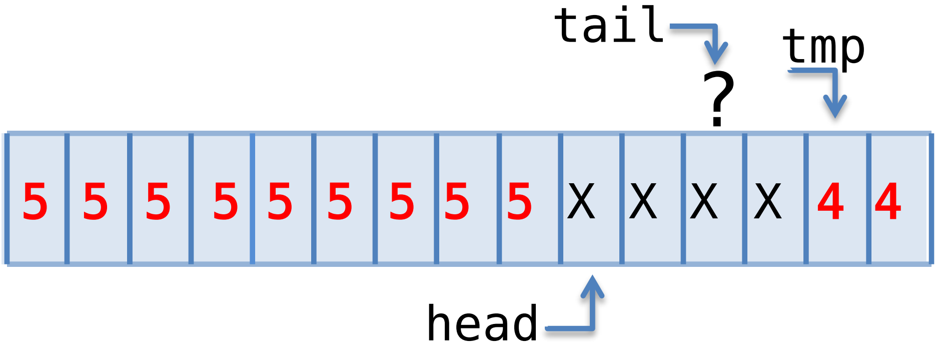
Is tail up to date “now”?



5	5	5	5	5	5	5	5	5	X	X	X	X	4	4

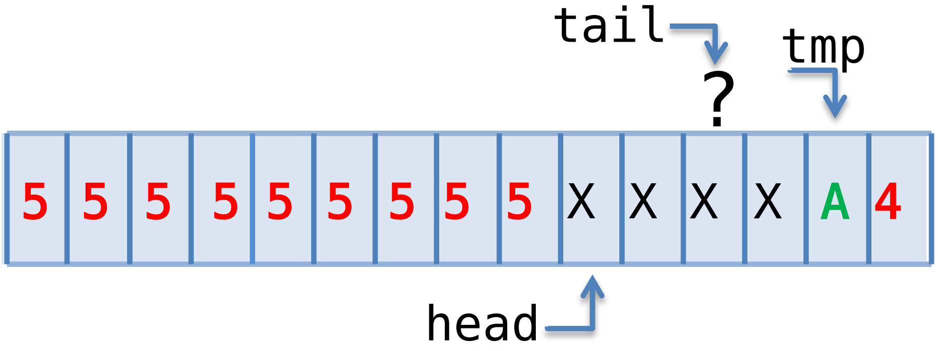
```
do {
    tmp = oldtail = tail.load(relaxed);
    while (buffer[tmp].load(relaxed) != gen(tmp))
        tmp++;
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
CAS(tail, oldtail, tmp+1);
```

Is tail up to date “now”?



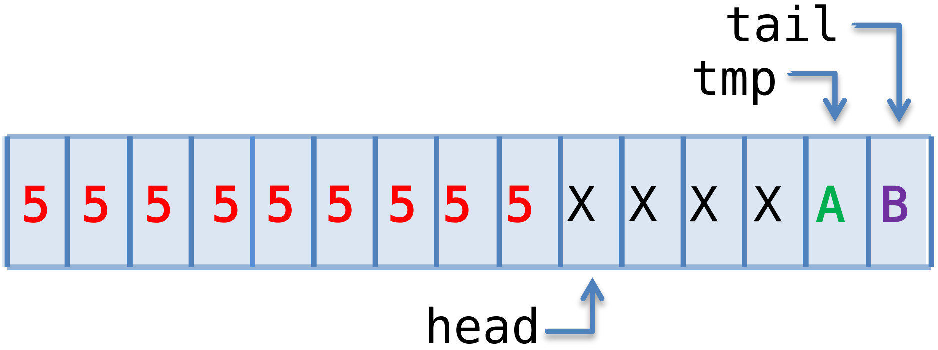
```
do {
        tmp = oldtail = tail.load(relaxed);
        while (buffer[tmp].load(relaxed) != gen(tmp))
                tmp++;
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
CAS(tail, oldtail, tmp+1);
```

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”?



tail

tmp

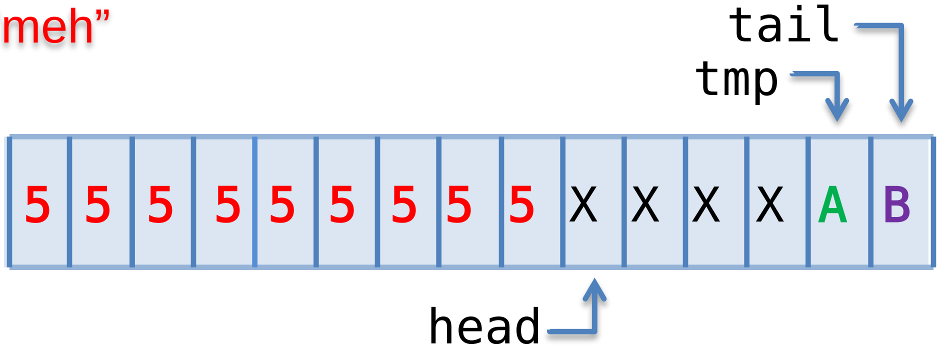
5 5 5 5 5 5 5 5 5 X X X X A B

head

```
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”

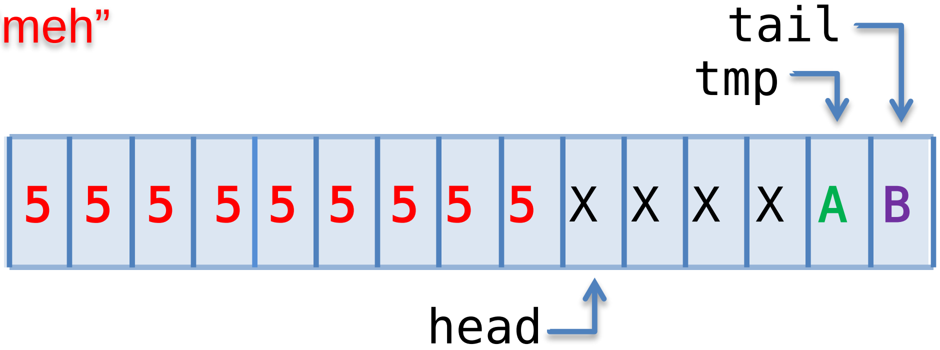


5	5	5	5	5	5	5	5	5	X	X	X	X	A	B

```
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”?

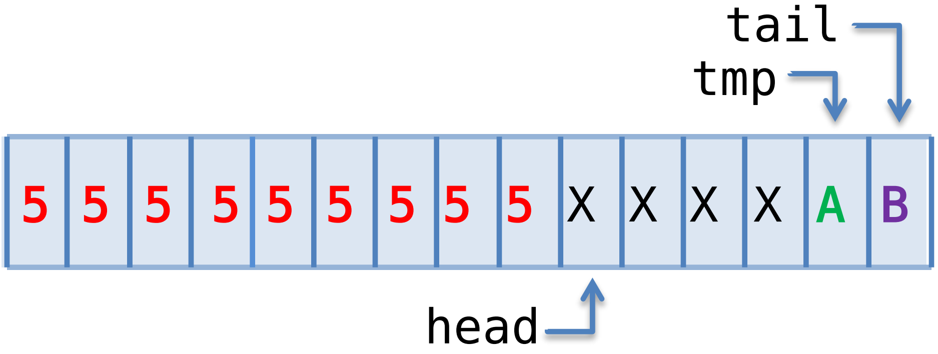
“meh”



push(val)

?

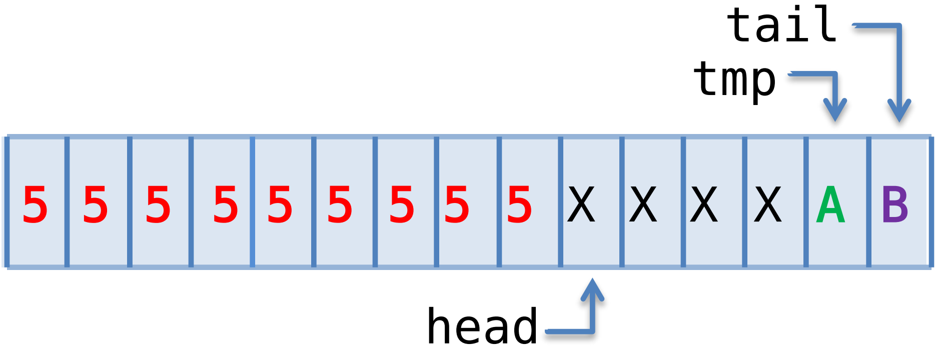
```
do {
        tmp = oldtail = tail.load(relaxed);
        while (buffer[tmp].load(relaxed) != gen(tmp))
                tmp++;
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
CAS(tail, oldtail, tmp+1);
```



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);
```

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

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);
```

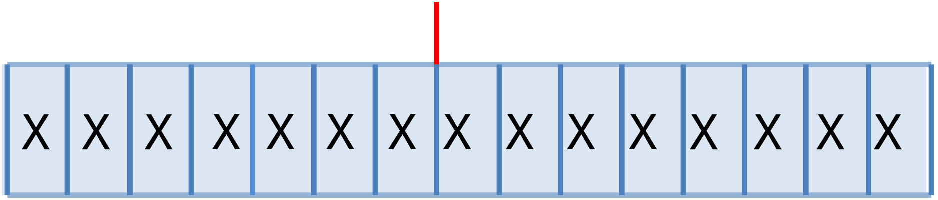
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

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);
```

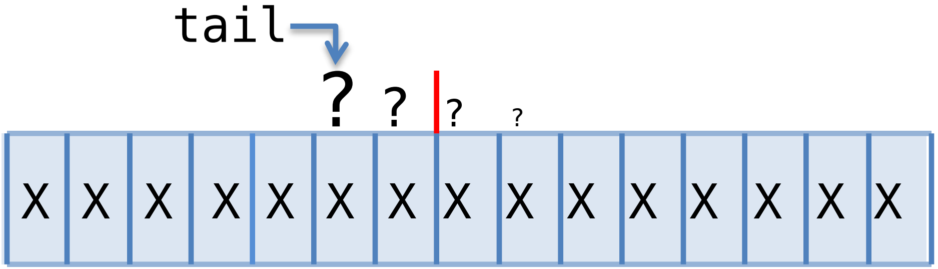


All states are valid states for all lines of code?

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);
```

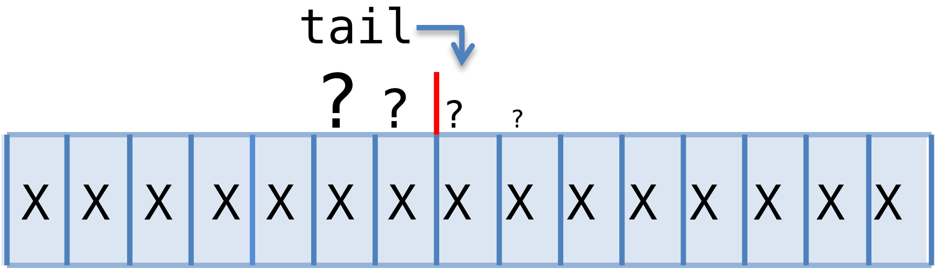


All states are valid states for all lines of code?

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);
```

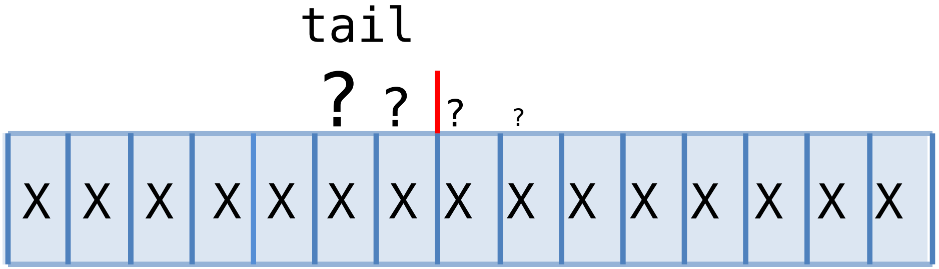


All states are valid states for all lines of code?

push(val)

?

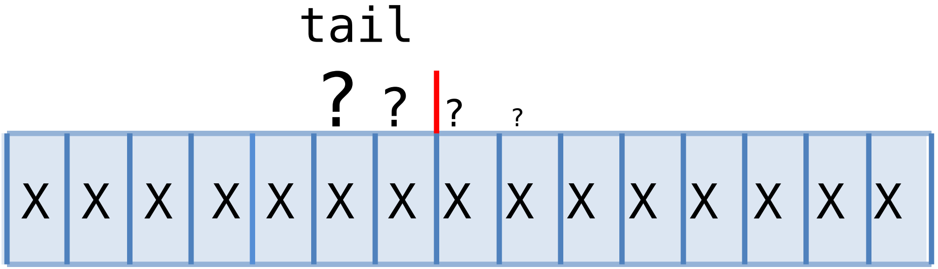
```
do {
        tmp = oldtail = tail.load(relaxed);
        while (buffer[tmp].load(relaxed) != gen(tmp))
                tmp++;
 } while ( ! CAS(buffer[tmp], gen(tmp), val) );
CAS(tail, oldtail, tmp+1);
```



All states are valid states for all lines of code?

```
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);
```

(worse?) spinlock ?



All states are valid states for all lines of code?

```
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);
```

(worse?) spinlock ?

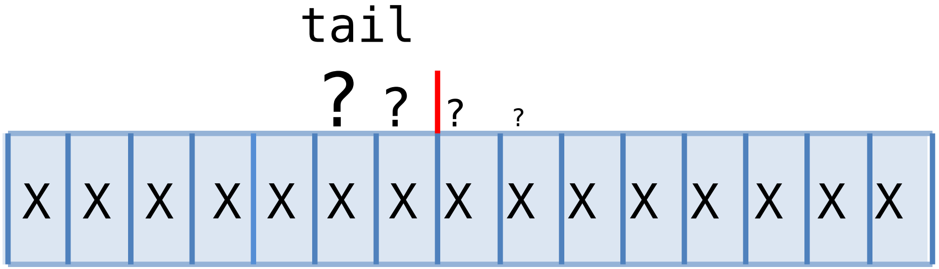
`tail`

Compromise…

All states are valid states for all lines of code?

```
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);
```

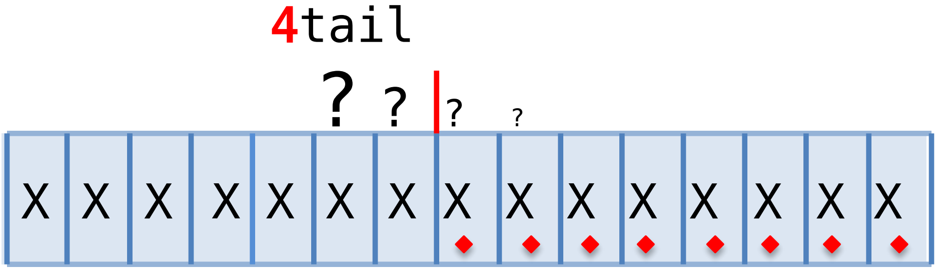
(worse?) spinlock ?



All states are valid states for all lines of code?

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

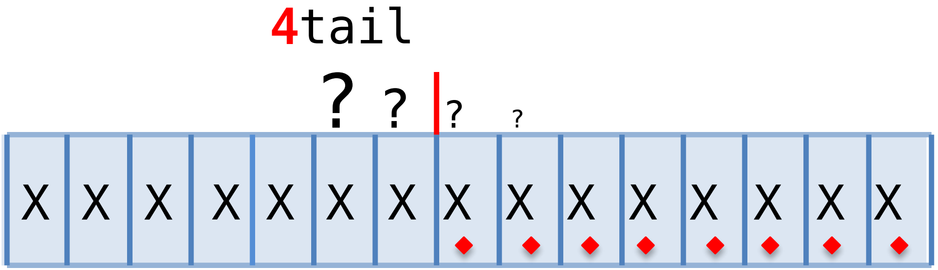
(worse?) spinlock ?



All states are valid states for all lines of code?

(worse?) spinlock ?

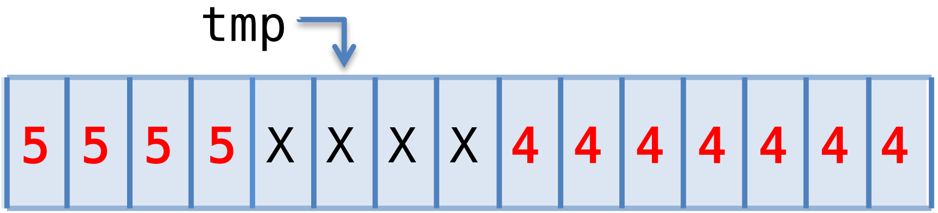
```
do {
        tmp = oldtail = tail.load(relaxed);
        tmp = find_tail(tmp);
 } while ( ! CAS(buffer[tmp], gen(tmp), val | odd(tmp
CAS(tail, oldtail, tmp+1);
```



All states are valid states for all lines of code?

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

(worse?) spinlock ?

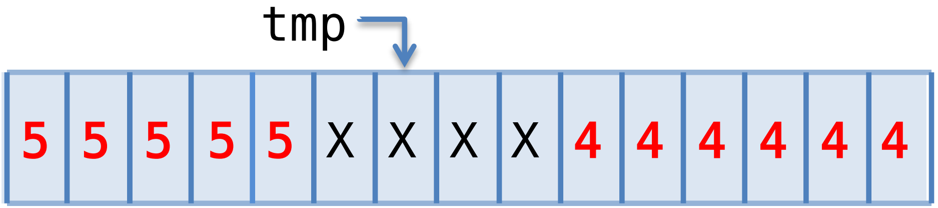


5	5	5	5	X	X	X	X	4	4	4	4	4	4	4

All states are valid states for all lines of code?

(worse?) spinlock ?

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

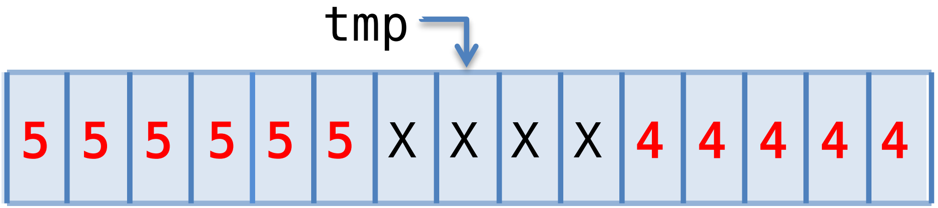


5	5	5	5	5	X	X	X	X	4	4	4	4	4	4

All states are valid states for all lines of code?

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

(worse?) spinlock ?

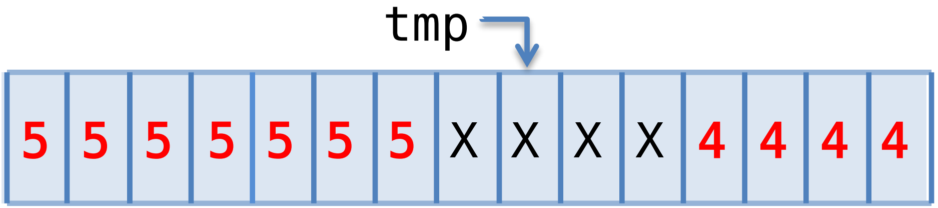


5	5	5	5	5	5	X	X	X	X	4	4	4	4	4

All states are valid states for all lines of code?

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

(worse?) spinlock ?



5	5	5	5	5	5	5	X	X	X	X	4	4	4	4

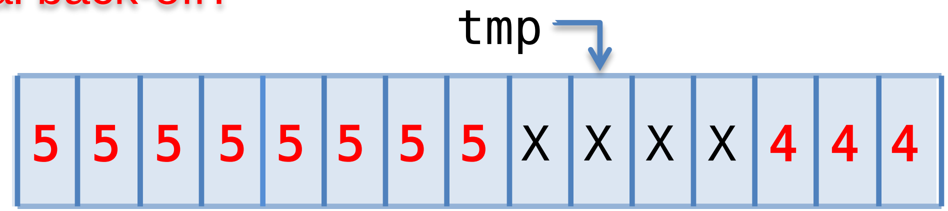
All states are valid states for all lines of code?

(worse?) spinlock ?

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

unlikely, however…

exponential back-off?



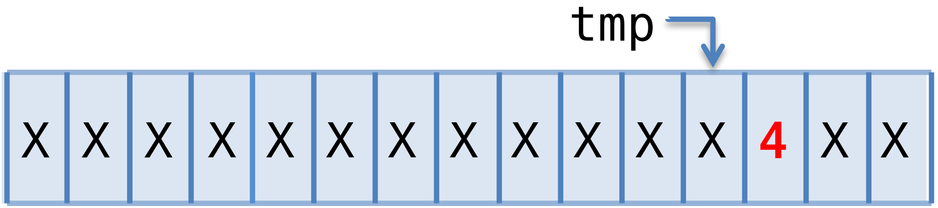
tmp

5	5	5	5	5	5	5	5	X	X	X	X	4	4	4

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 | odd(tmp)
CAS(tail, oldtail, tmp+1);
```



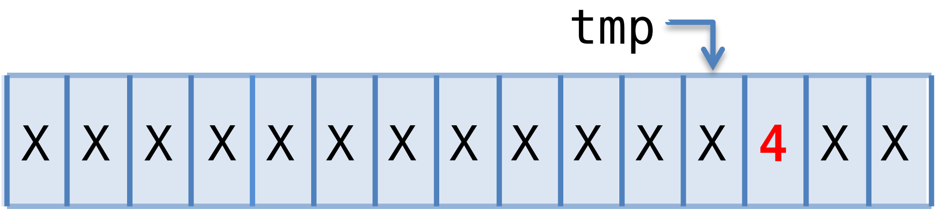
tmp

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

All states are valid states for all lines of code?

“fullish”

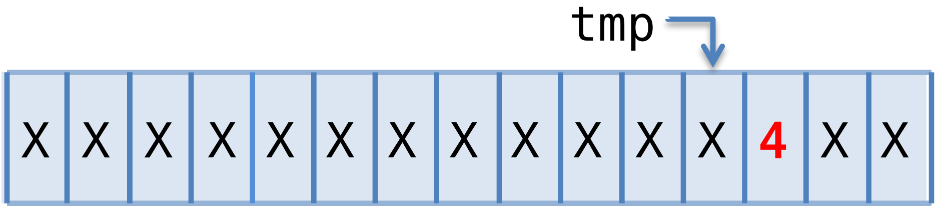
```
do {
        tmp = oldtail = tail.load(relaxed);
        tmp = find_tail(tmp, &oldtail);
        if (tmp == FULL) wait_for_space();
 } while ( ! CAS(buffer[tmp], gen(tmp), val | odd(tmp)
CAS(tail, oldtail, tmp+1);
```



All states are valid states for all lines of code?

```
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 | odd(tmp)
CAS(tail, oldtail, tmp+1);
```

“fullish” →



tmp

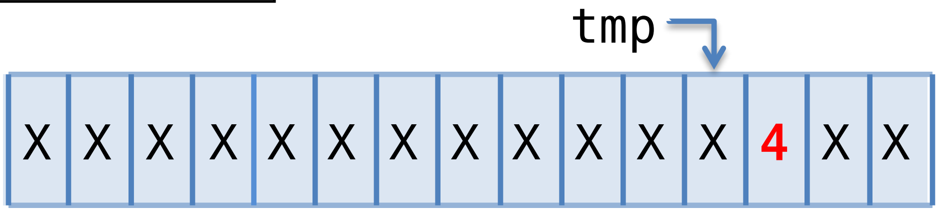
X X X X X X X X X X X X 4 X X

All states are valid states for all lines of code?

```
{
  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 | odd(tmp)
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

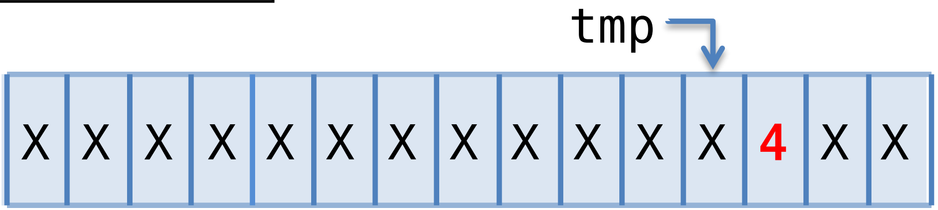
CppCon, September 2014

BlackBerry

```
{
  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 | odd(tmp)
CAS(tail, oldtail, tmp+1);
```

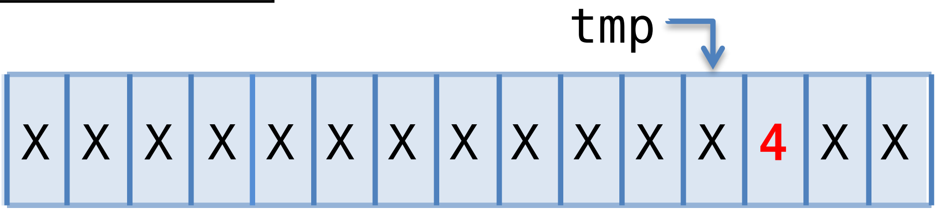


All states are valid states for all lines of code?

```
{
   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 | odd(tmp)
CAS(tail, oldtail, tmp+1);
```



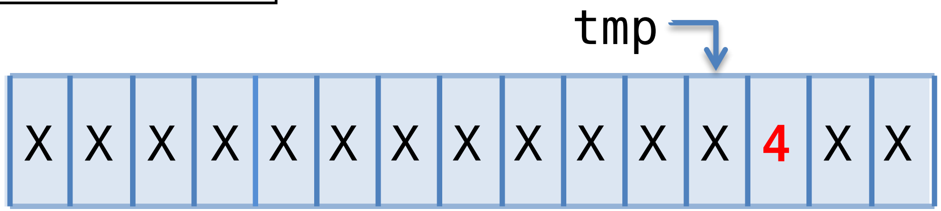
X	X	X	X	X	X	X	X	X	X	X	X	4	X	X

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 | odd(tmp)
CAS(tail, oldtail, tmp+1);
```



tmp

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

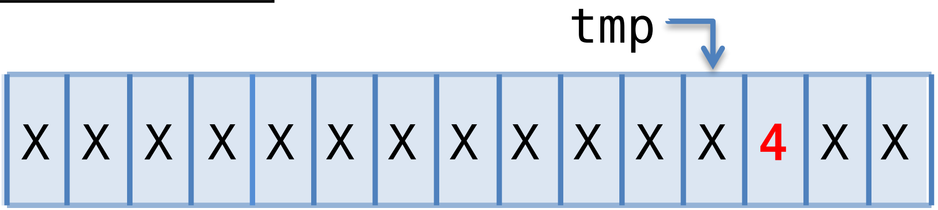
All states are valid states for all lines of code?

```
{
  unique_lock lock(mutex);

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

?

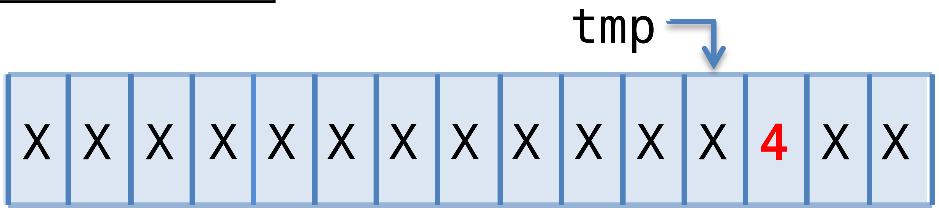
who calls notify()?



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

```
{
   unique_lock lock(mutex);

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



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

```
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);
}
```

X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

```
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);
}
```

5	5	5	5	5	5	5	5	5	5	5	5	4	4	4

```
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);
}
```

5	5	5	5	5	5	5	5	5	5	5	5	4	4	4

```
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!

5	5	5	5	5	5	5	5	5	5	5	5	4	4	4

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

?

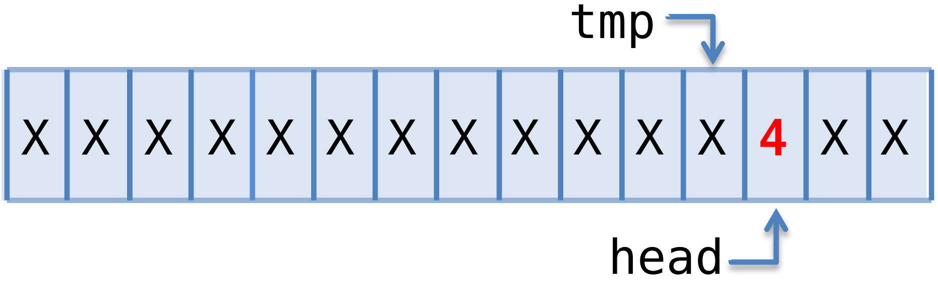
```
int pop() {
  ...
  unique_lock lock(mutex);
  cond_full.notify();
}
```

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

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

I'm waiting!

```
int pop() {
  …CAS(buffer[x], val, gen);//4
  if (waiting) {
    unique_lock lock(mutex);
    cond_full.notify();
  }
}
```



tmp

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

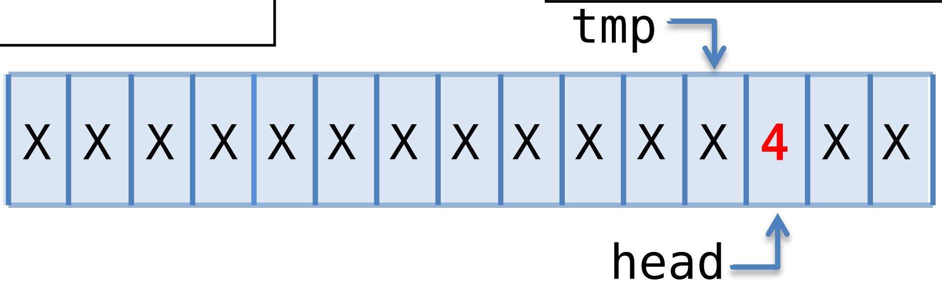
head

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

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

I’m waiting!

```
int pop() {
 …CAS(buffer[x], val, gen);//4
 if (waiting) {
  unique_lock lock(mutex);
  cond_full.notify();
 }
}
```

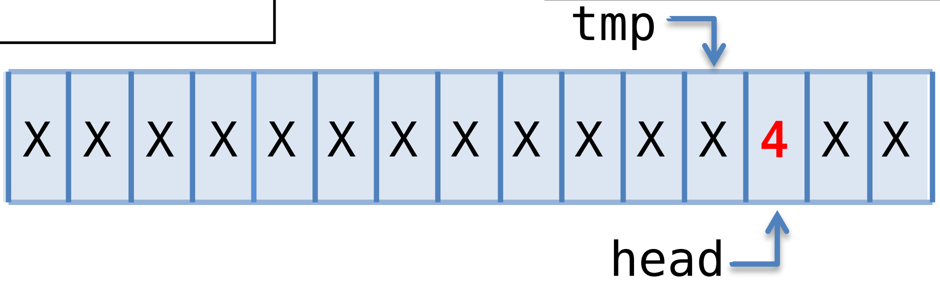


```
waiting++;
{
  unique_lock lock(mutex);

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

I’m waiting!

```
int pop() {
  …CAS(buffer[x], val, gen);//4
  if (waiting) {
    unique_lock lock(mutex);
    cond_full.notify();
  }
}
```



tmp

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

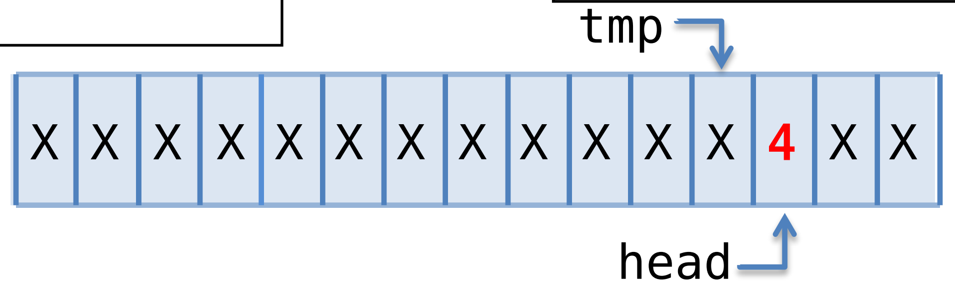
head

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

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

I'm waiting!

```
int pop() {
  …CAS(buffer[x], val, gen);//4
  if (waiting) {
    unique_lock lock(mutex);
    cond_full.notify();
  }
}
```



rarely

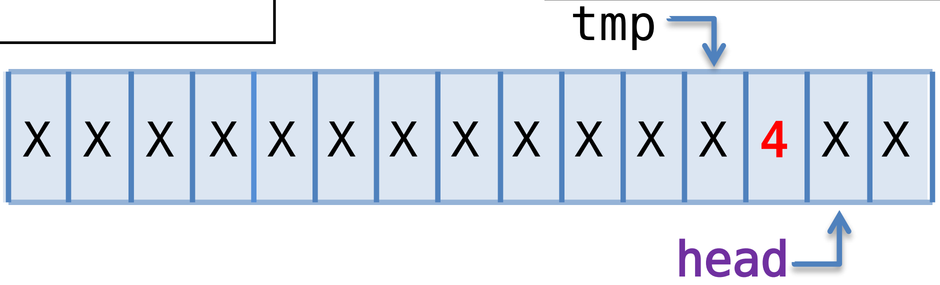
always

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

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

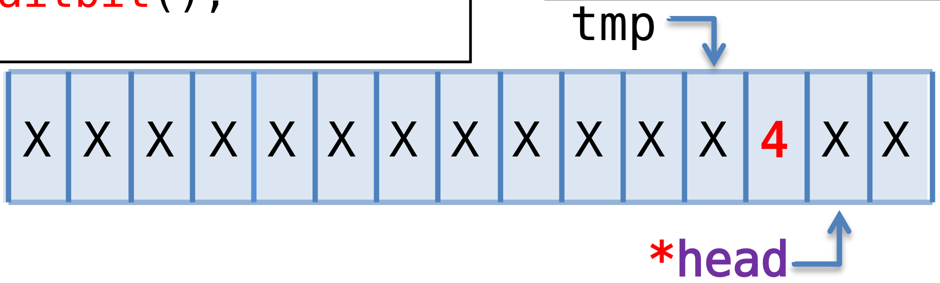
I'm waiting!

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



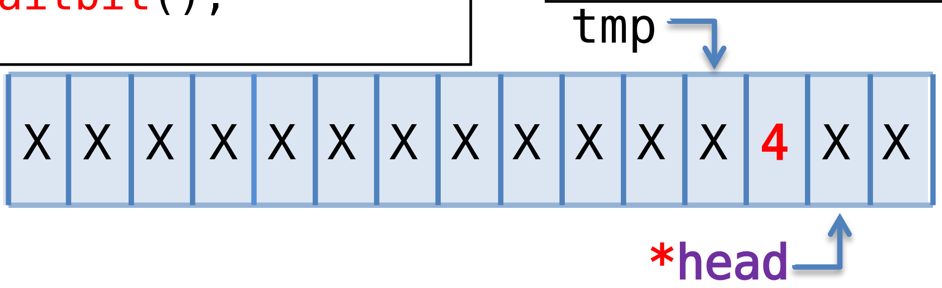
```
{
  unique_lock lock(mutex);
  if (waiting++ == 0)
    head.set_waitbit();
  while ( ! …find_tail… )
     cond_full.wait(lock);
  if (--waiting == 0)
    head.clear_waitbit();
}
```

```
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();
  }
}
```



NOTE: **waiting** is NOT atomic

AUTION !
oison Ivy
ay on Trail
E PET OWNER'S CODE
rloo

Looking Back

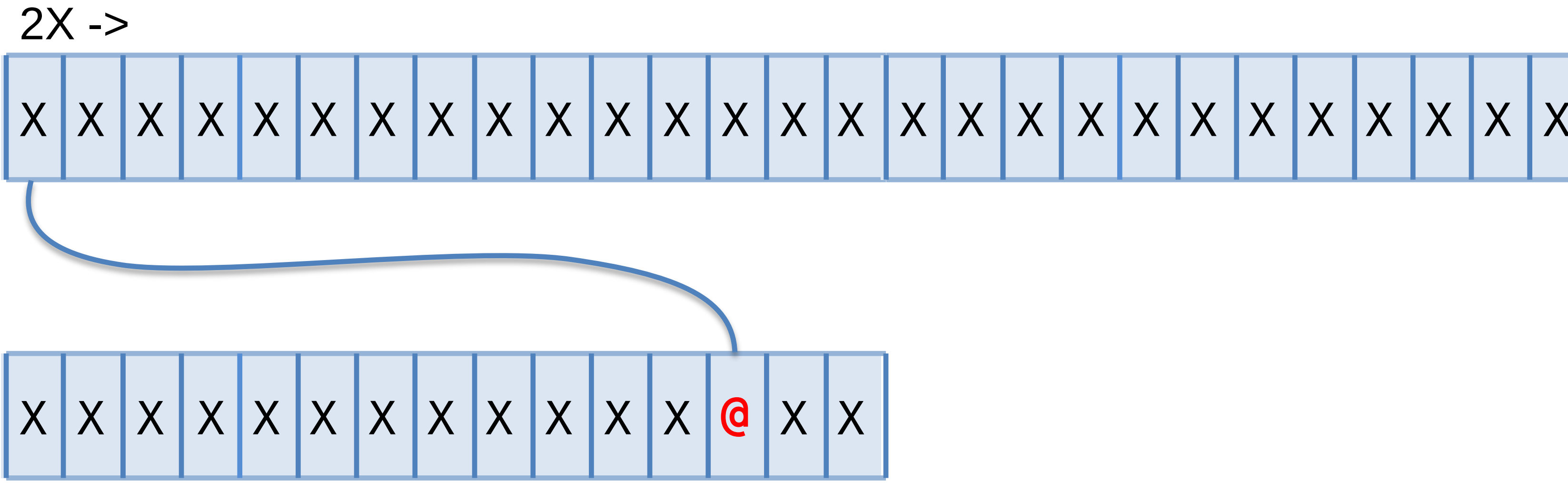
Looking Back

push()

Looking Ahead

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

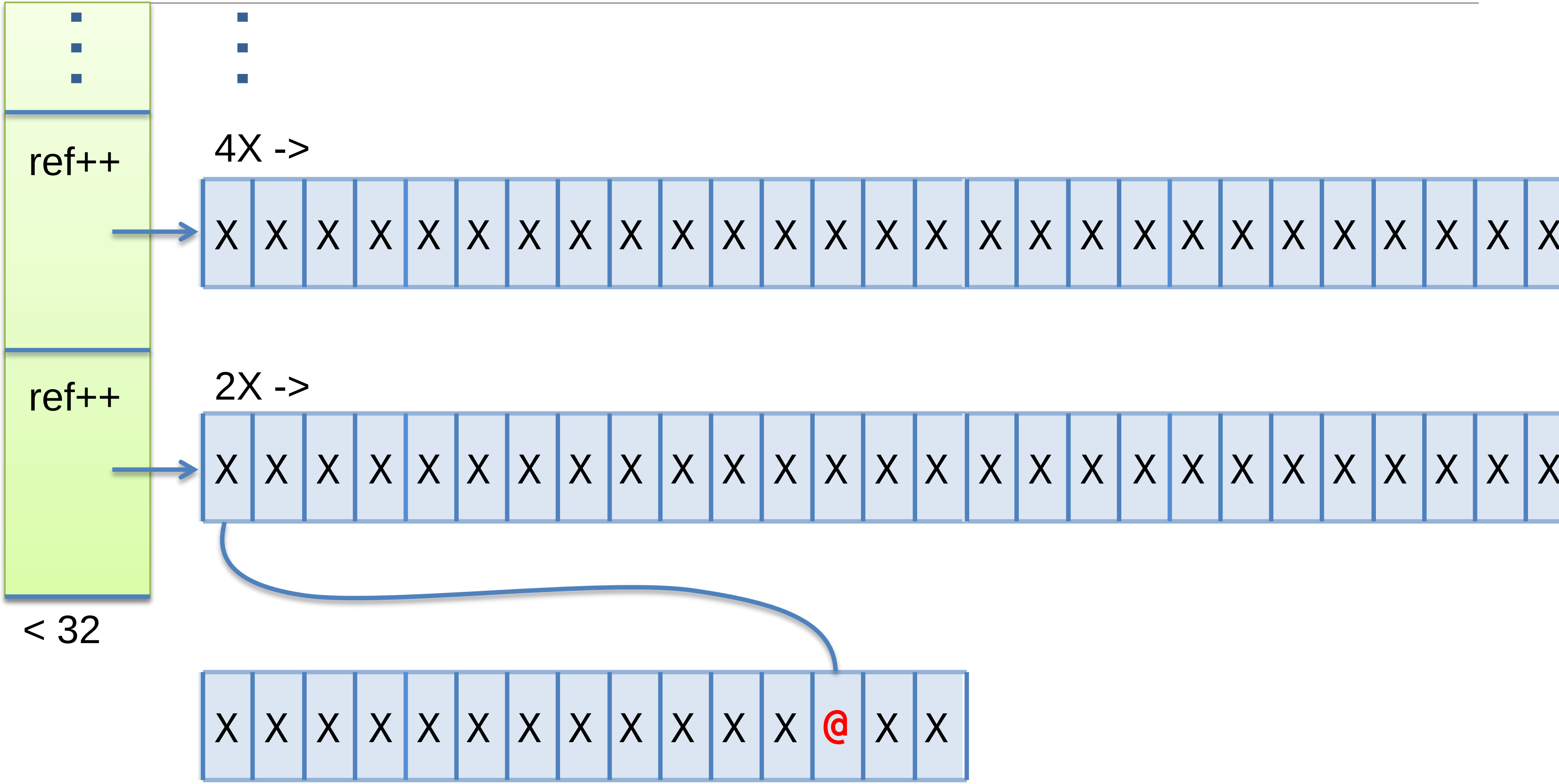
X	X	X	X	X	X	X	X	X	X	X	X	4	X	X



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



ref++

4X ->

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++

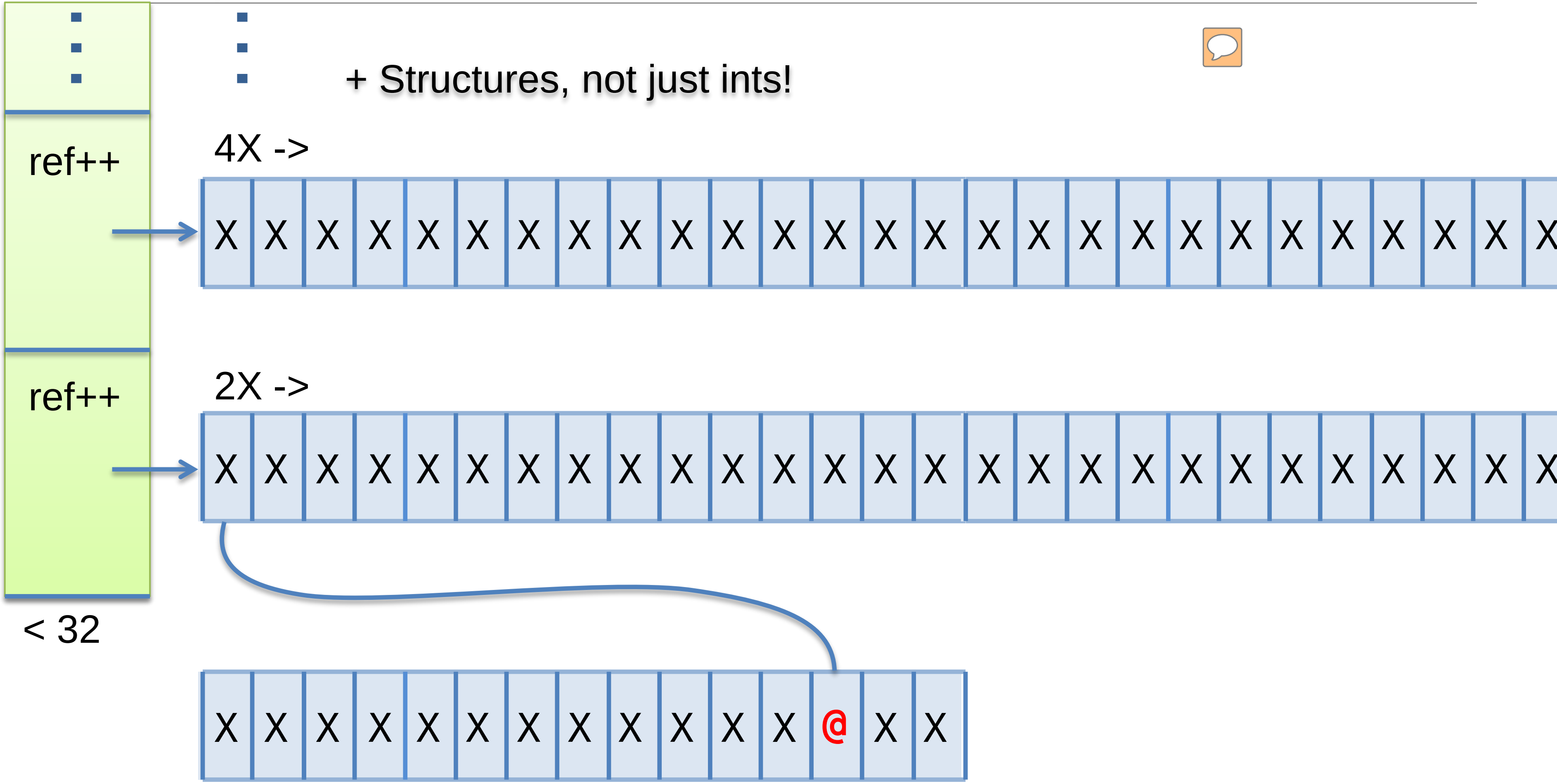
2X ->

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

< 32

X X X X X X X X X X X X @ X X

+ Structures, not just ints!


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

“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!