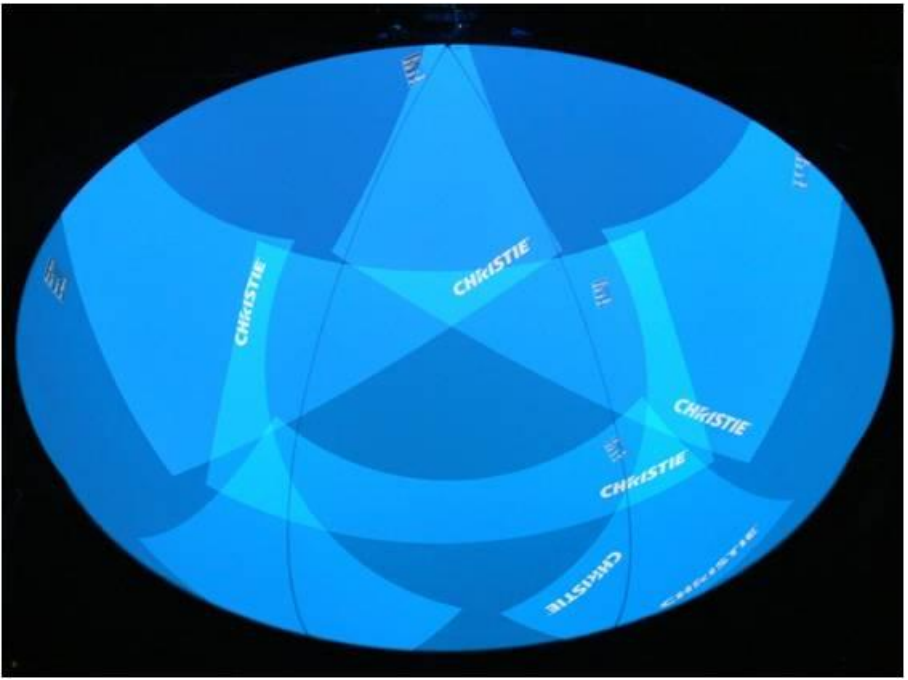
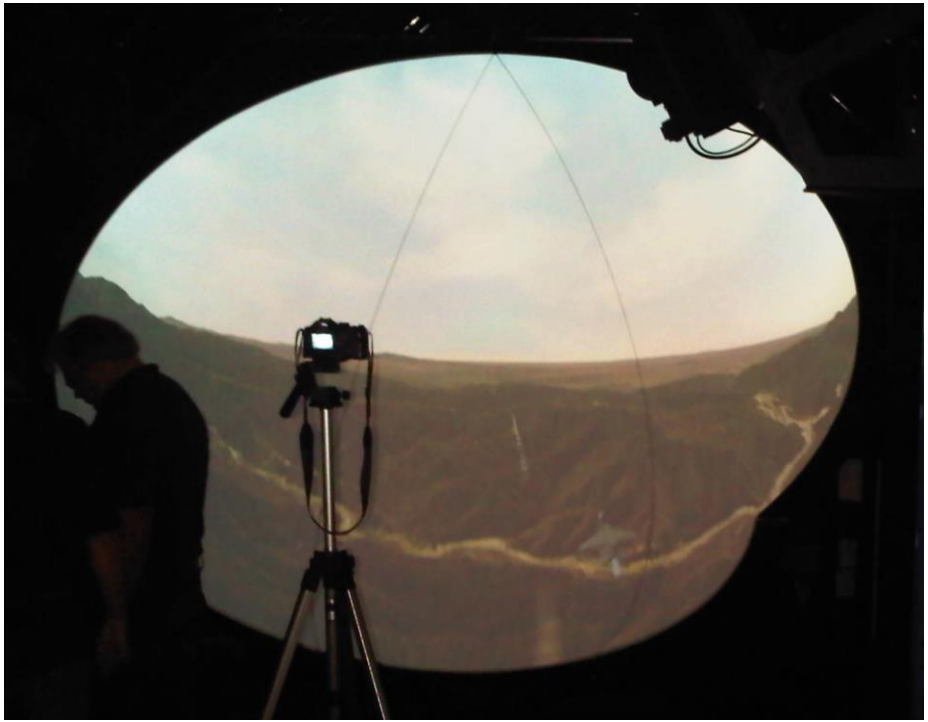
The Continuing Saga of the Lock-free Queue
Part 3 of N

Tony Van Eerd C++Now May 2018

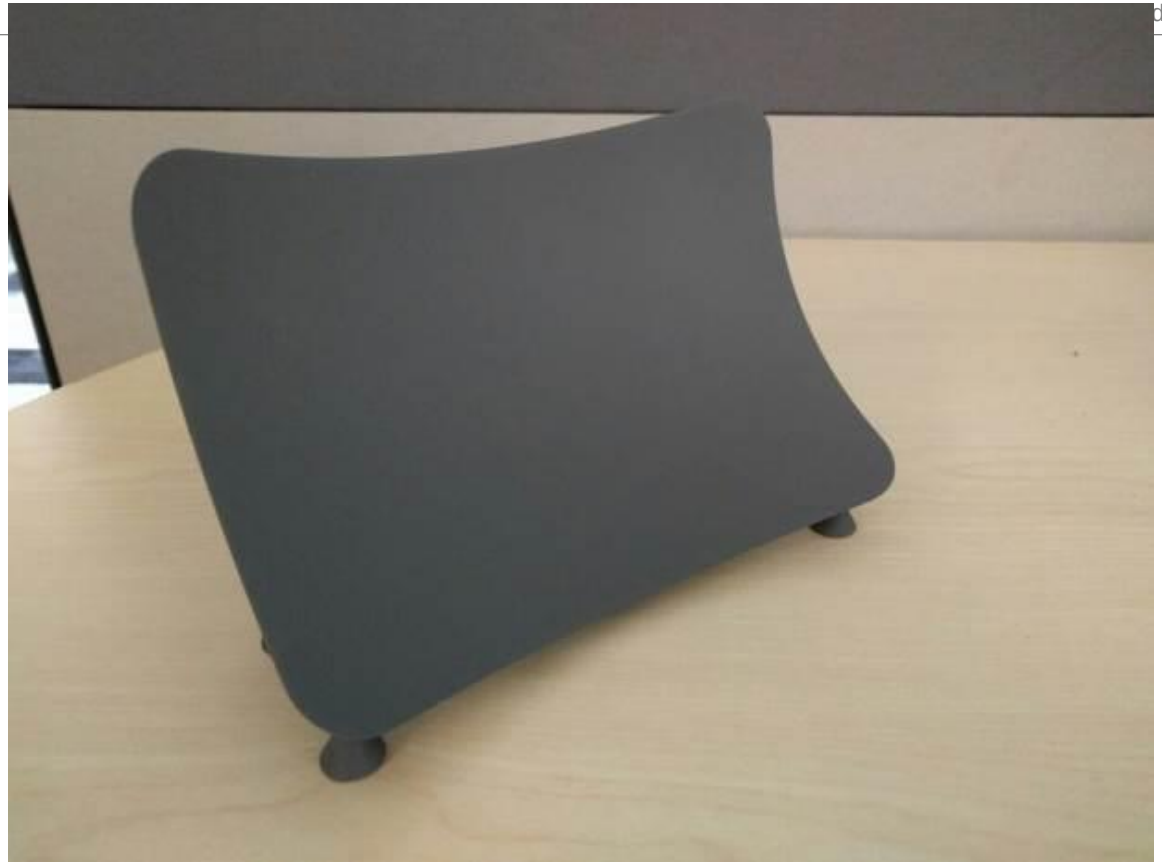
CHRISTIE

CHRISTIE®


CHRISTIE
CHRISTIE
CHRISTIE
CHRISTIE
CHRISTIE




KISTIE




ISTIE


CHRISTIE
CHRISTIE
Christie Mystique
Spectacular experiences. Simple solutions.
CHRISTIE CENTER
BOEHME

youtu.be/sql1kD6_Uok


CHRISTIE
easy

CHRISTIE®

Guide to Threaded Coding

Guide to Threaded Coding

1. Stop Sharing (forget what you learned in kindergarten)
2. OK, well Use Locks then
   (don't call unknown code while holding a lock)
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. Stop Sharing (forget what you learned in kindergarten)
2. OK, well Use Locks then
   (don't call unknown code while holding a lock)
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

Rules of Lock-free Coding

1.

Rules of Lock-free Coding

1. Don’t talk about lock-free coding

Guide to Coding

Guide to Coding

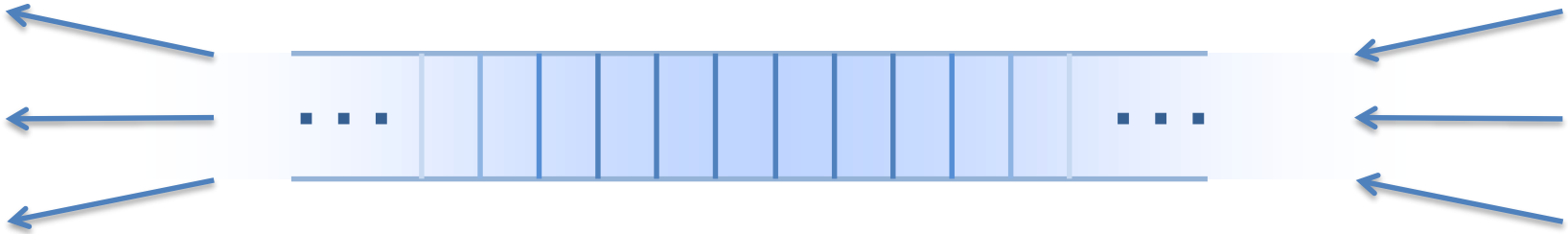
MACROS_ARE_EVIL

Notes:

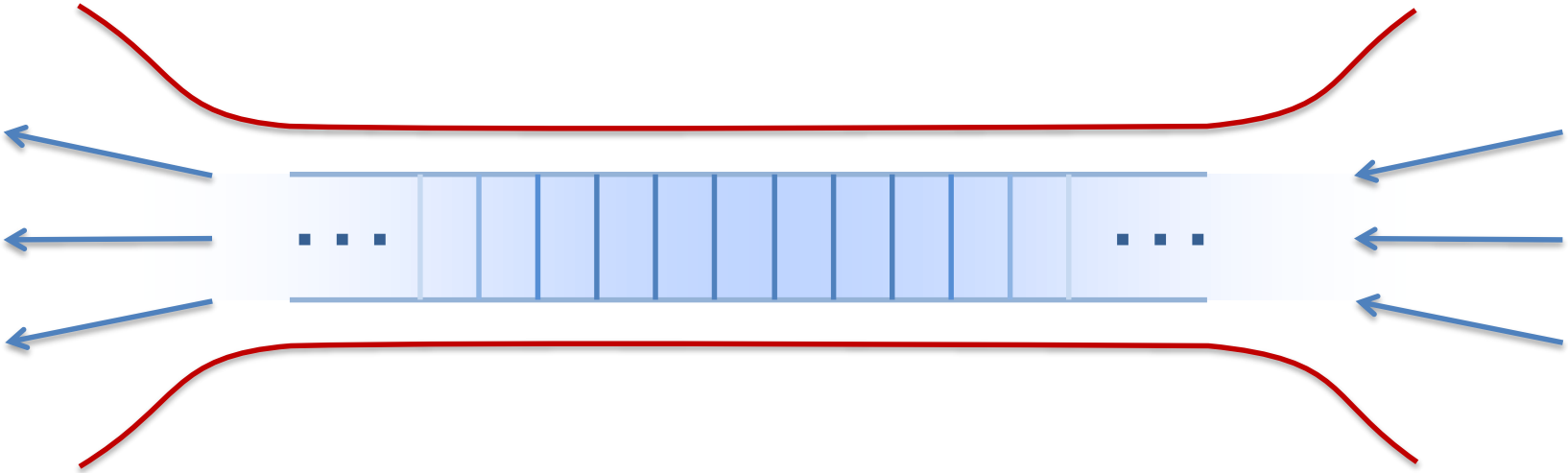
acquire = std::memory_order_acquire
release = std::memory_order_release
(etc)
CAS = “Compare and Set/Swap” = std::compare_exchange() and friends

Not my coding style/structure

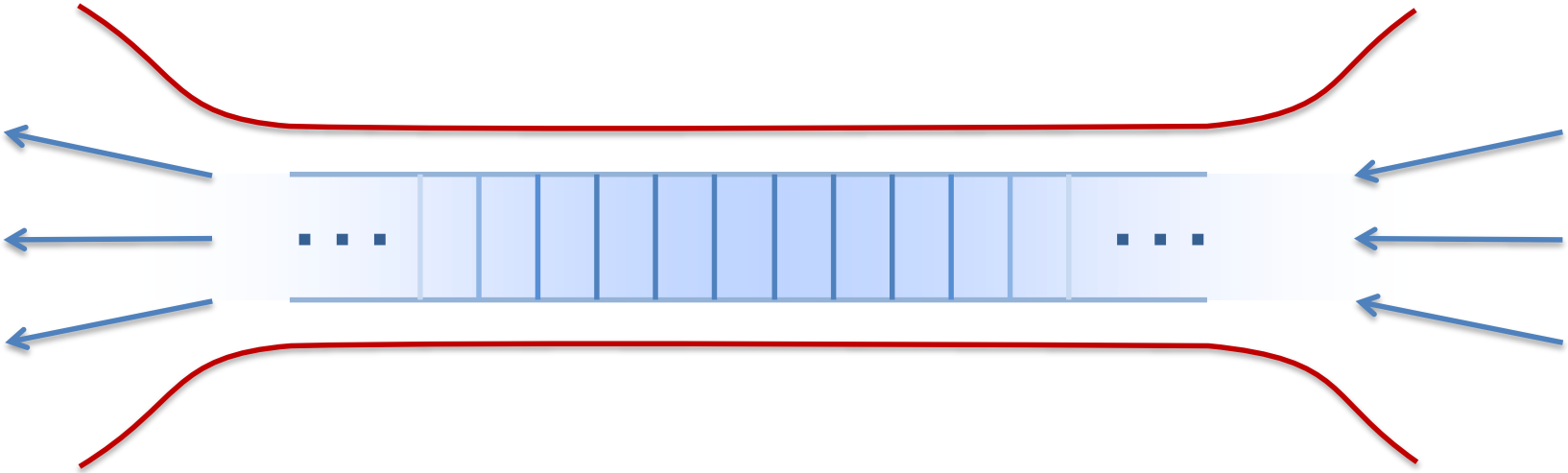
MPMC Queue

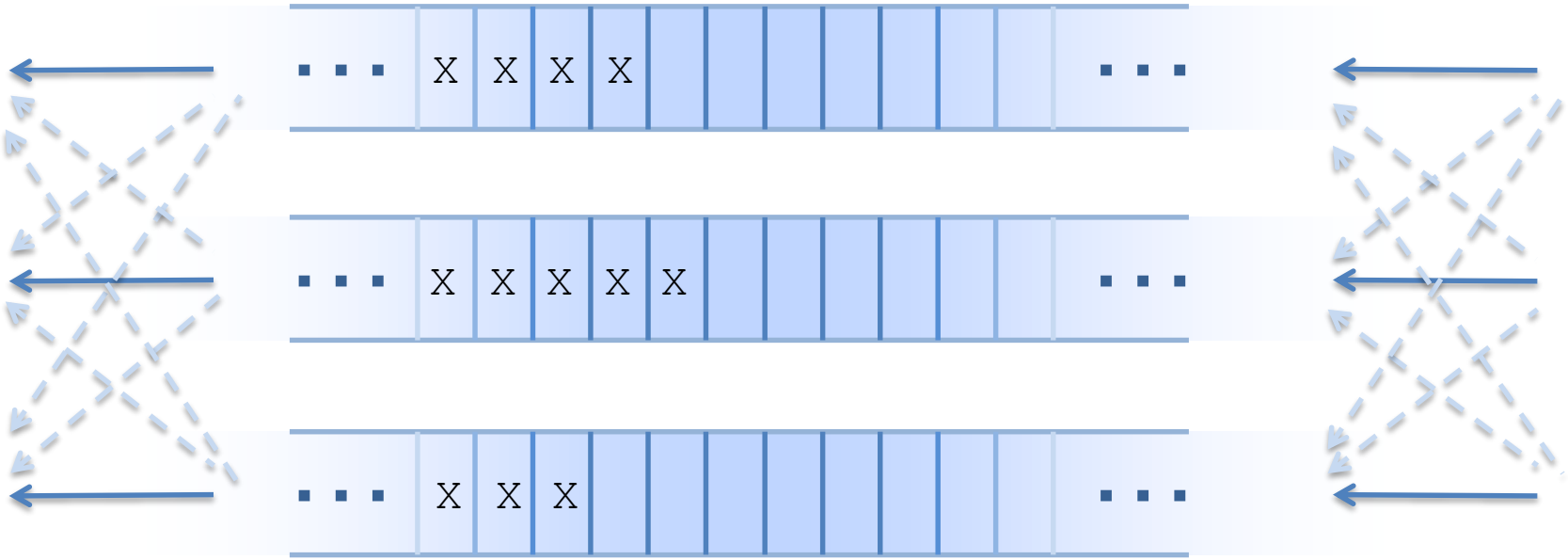


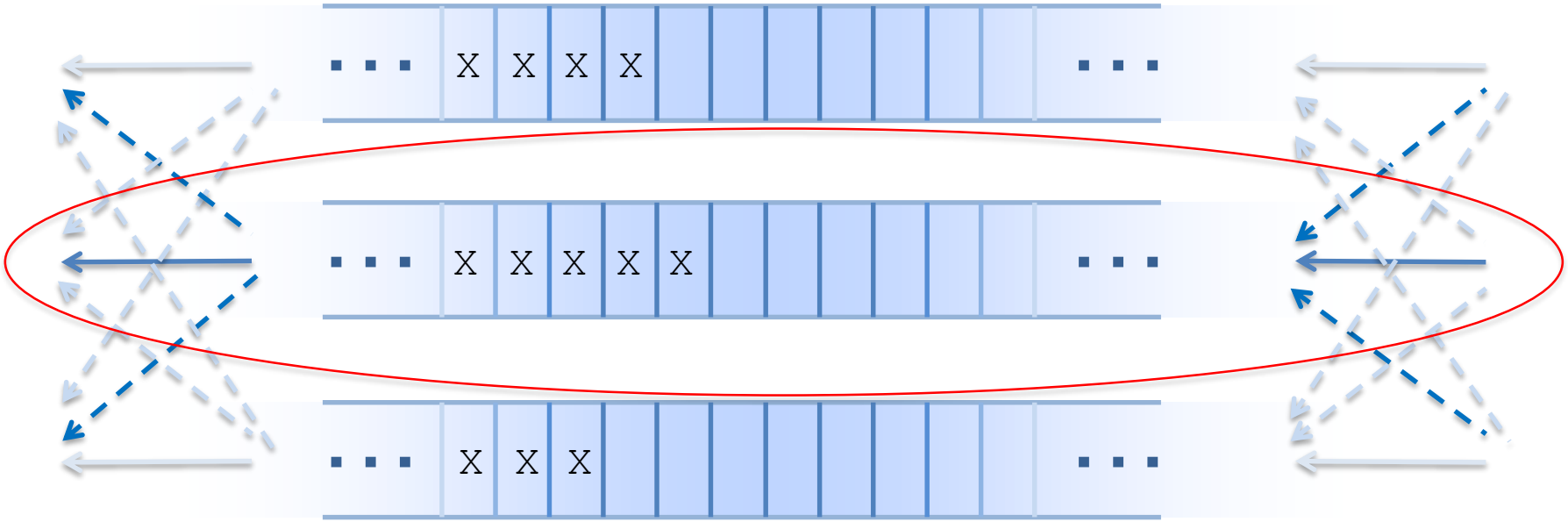
MPMC Queue

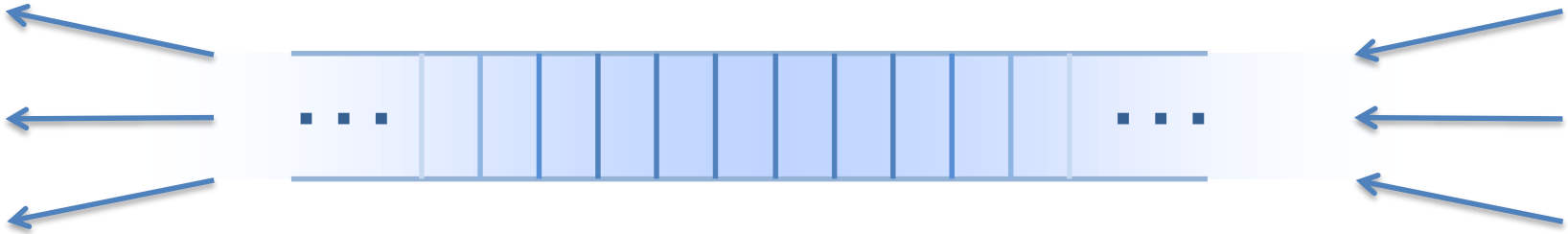


Bottleneck

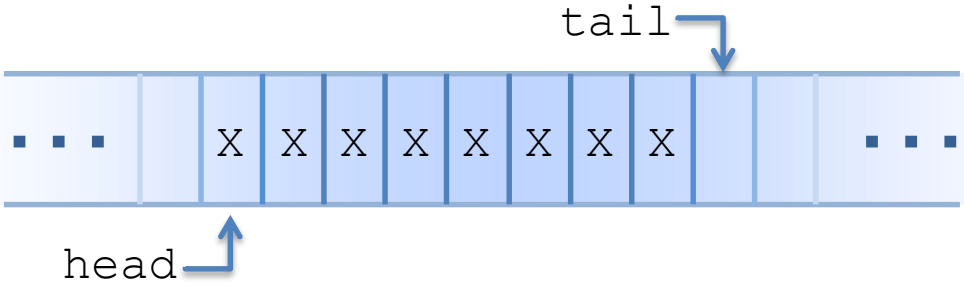



X X X X
X X X X X
X X X


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

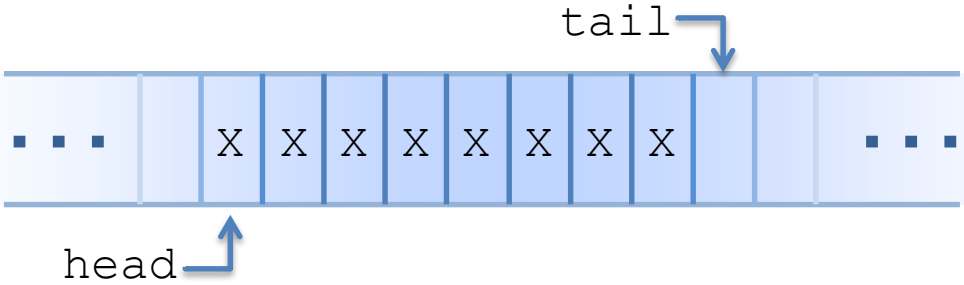


Review



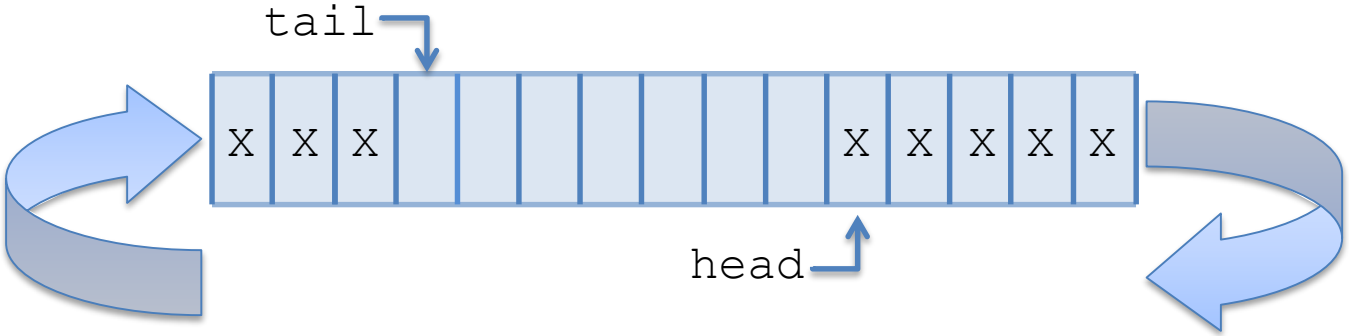
Review

```
class Queue {
    ?
};
```



Review

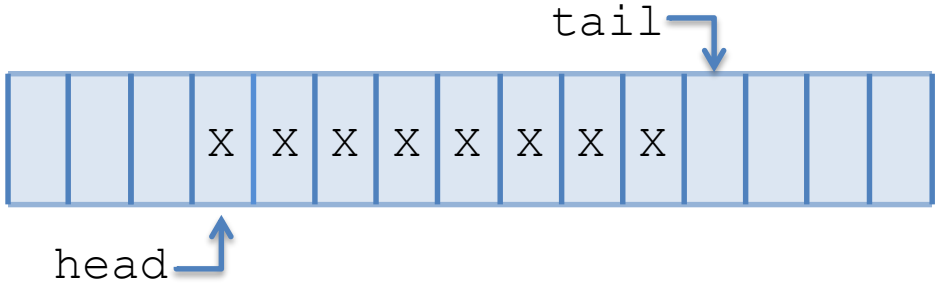
```
class Queue {
 T buffer[SIZE];
 int head;
 int tail;
};
```



Review

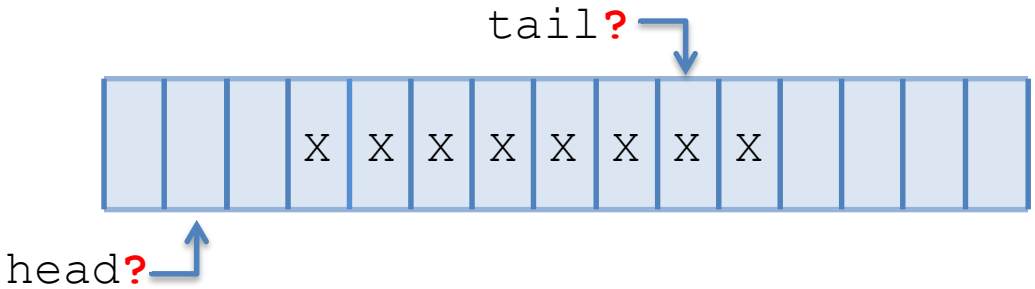
Compromise

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



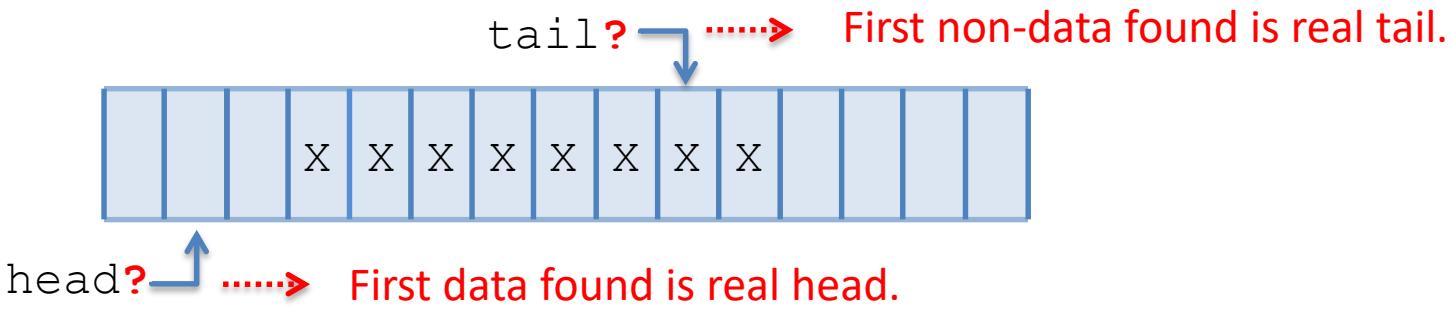
Review

```
class Queue {
 int buffer[SIZE];
 int headish;
 int tailish;
};
```



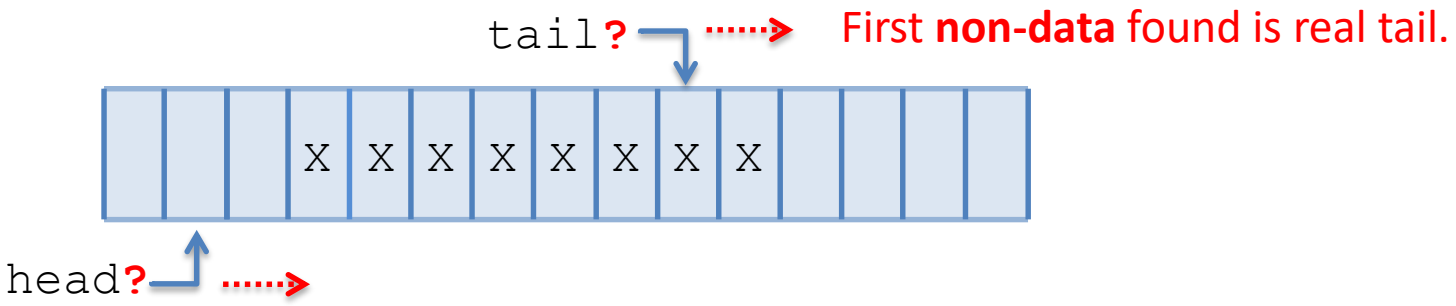
Review

```
class Queue {
 int buffer[SIZE];
 int headish;
 int tailish;
};
```



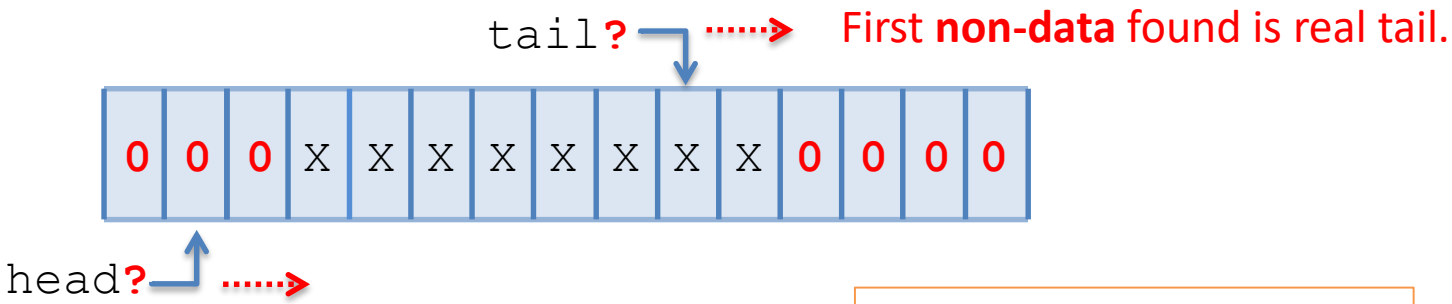
Review

```
class Queue {
 int buffer[SIZE];
 int headish;
 int tailish;
};
```



Review

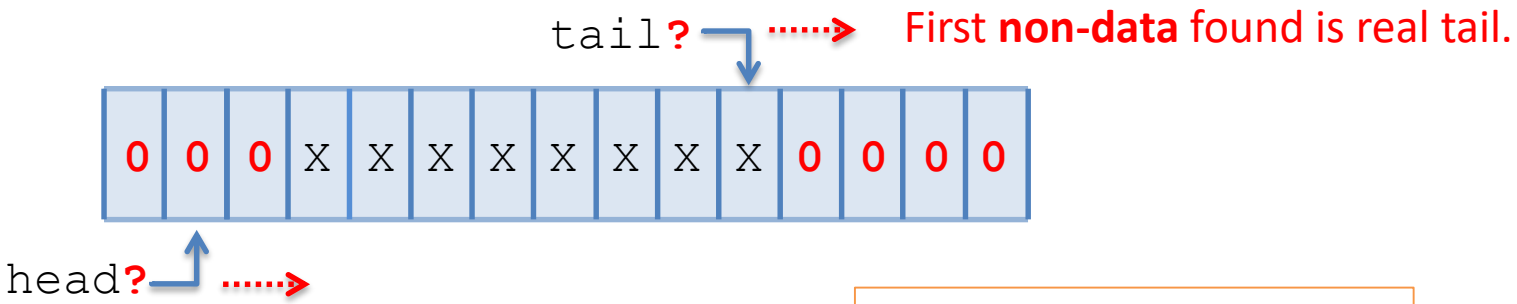
```
class Queue {
 int buffer[SIZE];
 int headish;
 int tailish;
};
```



Data == 0

Review

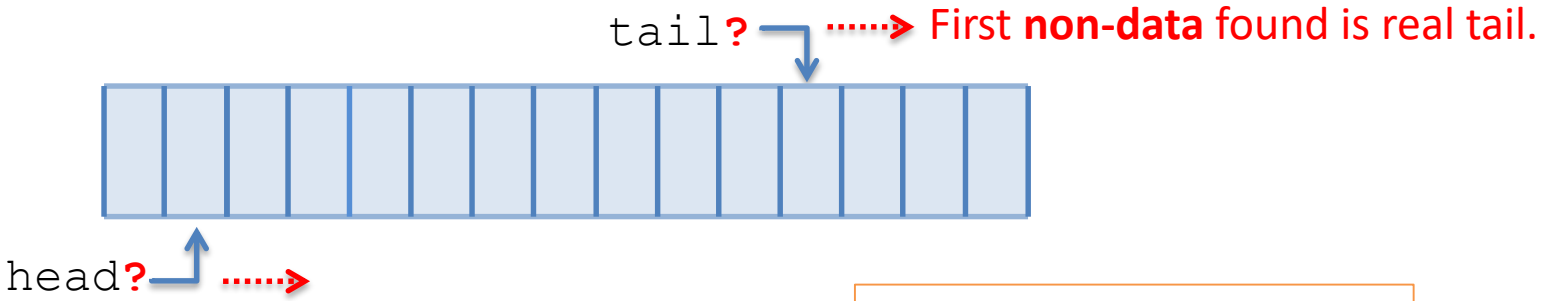
```
class Queue {
 int buffer[SIZE];
 int headish;
 int tailish;
};
```



Data == 0

Review

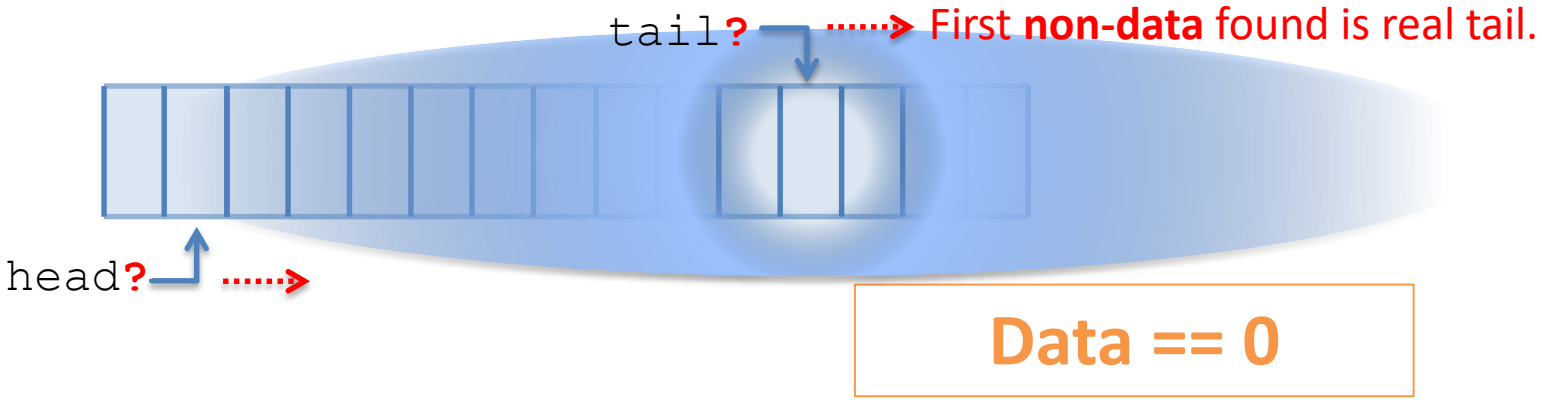
```
class Queue {
 int buffer[SIZE];
 int headish;
 int tailish;
};
```



Data == 0

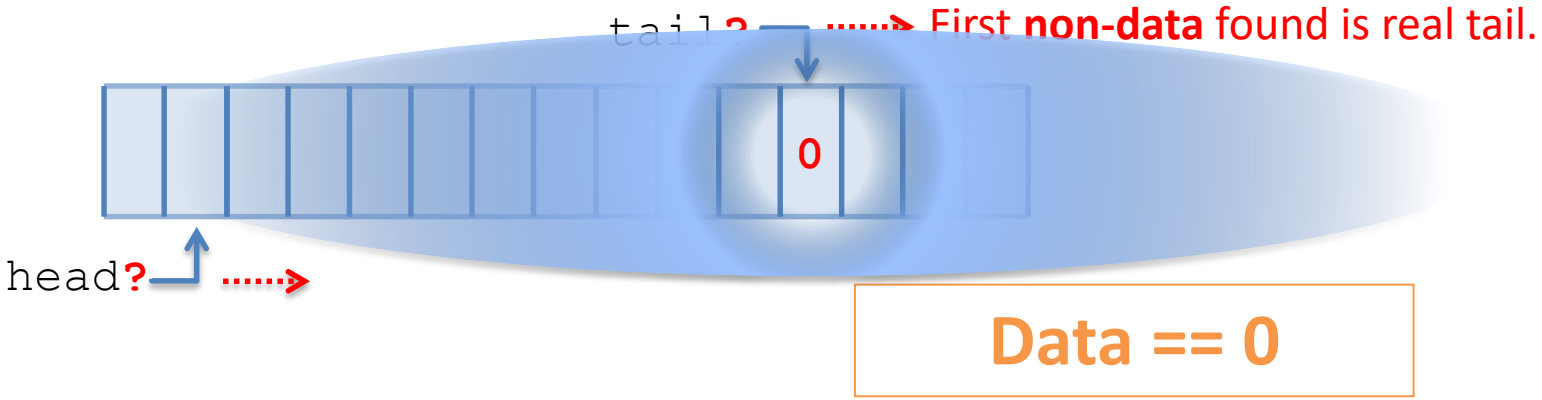
Review

```
class Queue {
 int buffer[SIZE];
 int headish;
 int tailish;
};
```



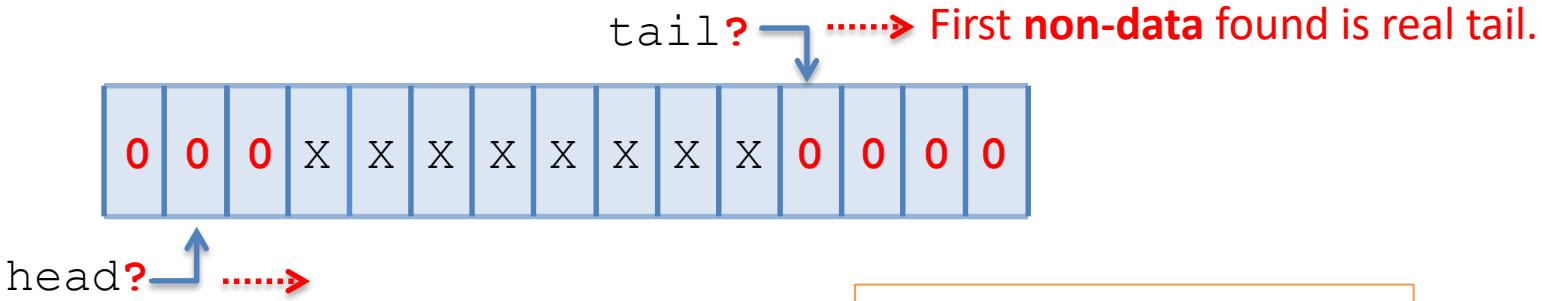
Review

```
class Queue {
 int buffer[SIZE];
 int headish;
 int tailish;
};
```



Review

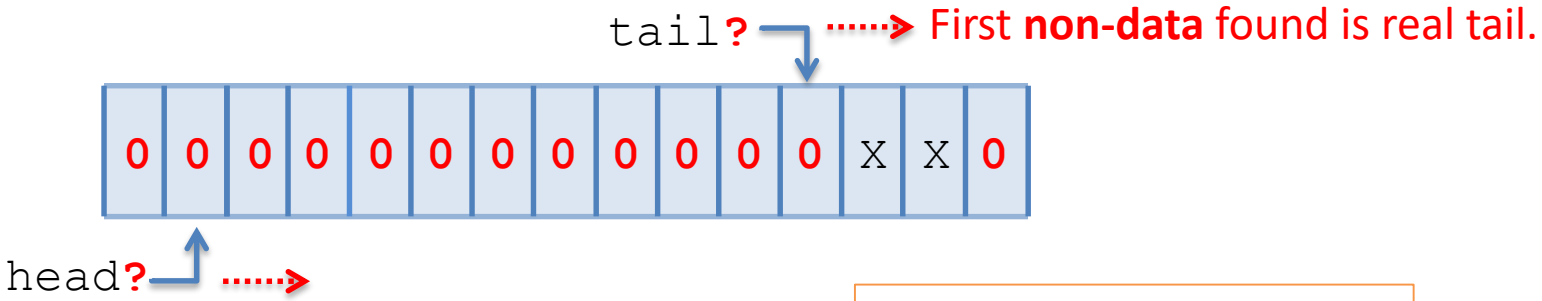
```
class Queue {
 int buffer[SIZE];
 int headish;
 int tailish;
};
```



Data == 0

Review

```
class Queue {
 int buffer[SIZE];
 int headish;
 int tailish;
};
```

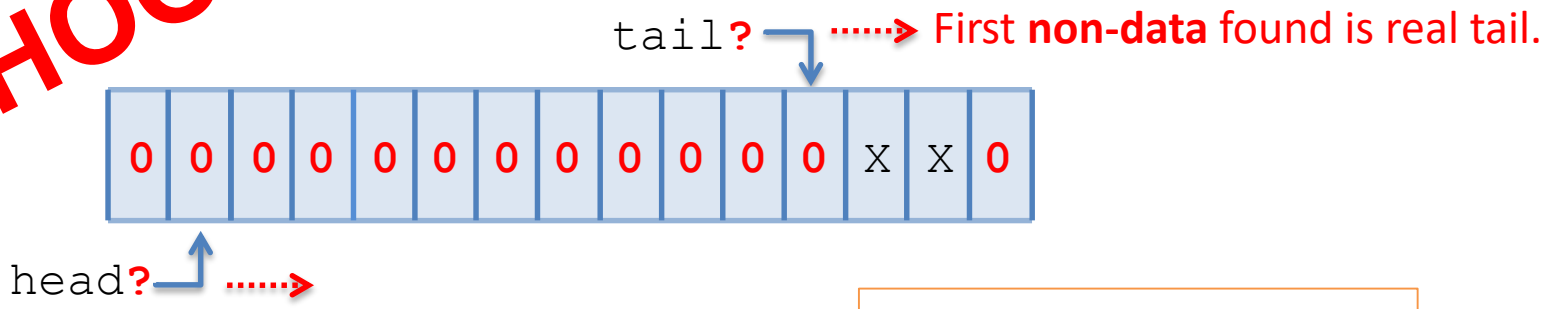


Data == 0

Review

```
class Queue {
 int buffer[SIZE];
 int headish;
 int tailish;
};
```

WHOOPS!

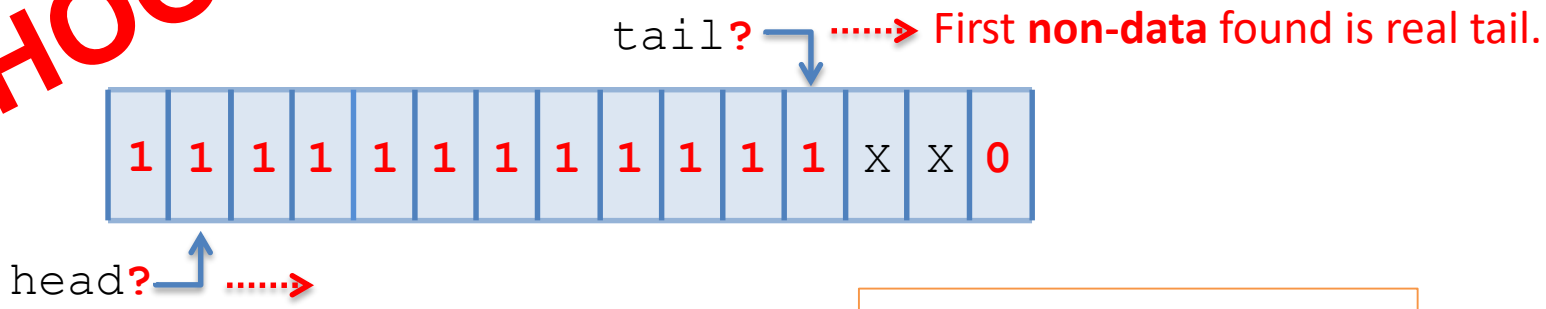


Data == 0

Review

```
class Queue {
 int buffer[SIZE];
 int headish;
 int tailish;
};
```

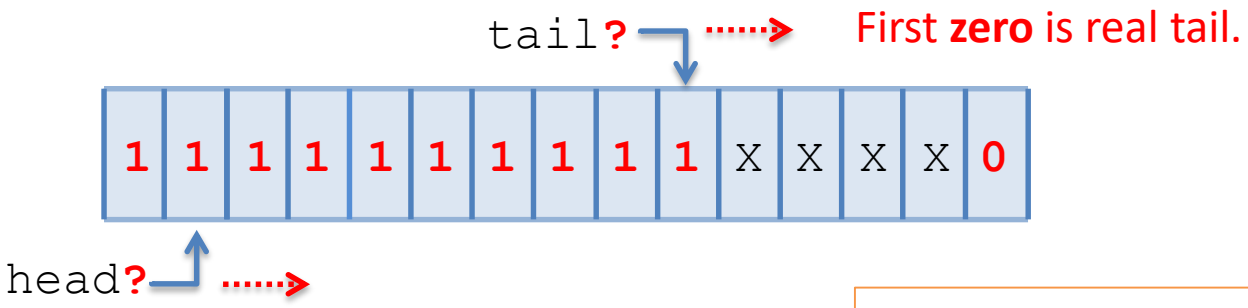
WHOOPS!



Compromise

Review

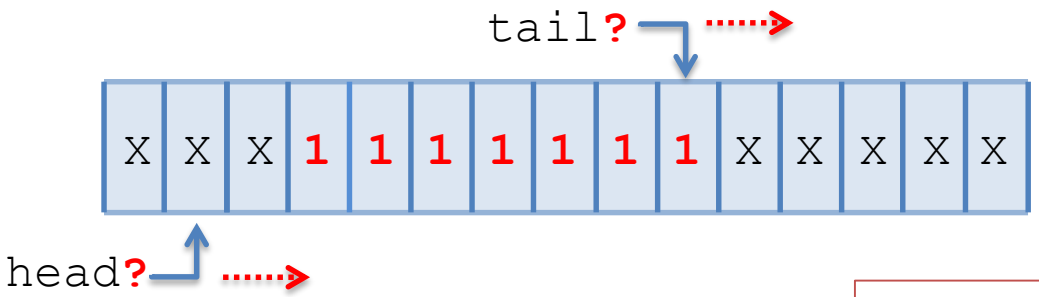
```
class Queue {
 int buffer[SIZE];
 int headish;
 int tailish;
};
```



Compromise

Review

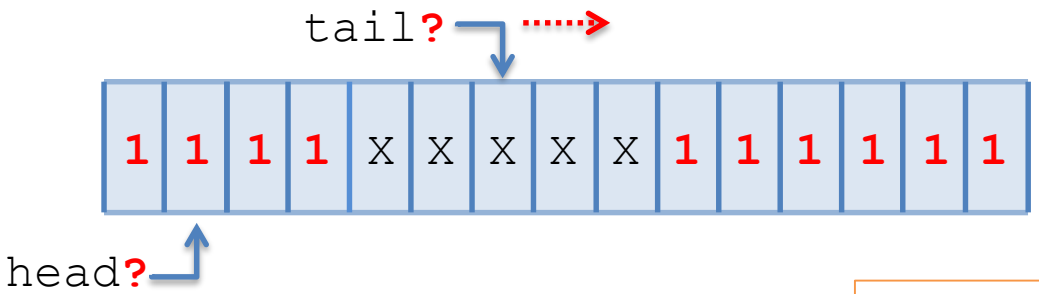
```
class Queue {
 int buffer[SIZE];
 int headish;
 int tailish;
};
```



Circular Buffer

Review

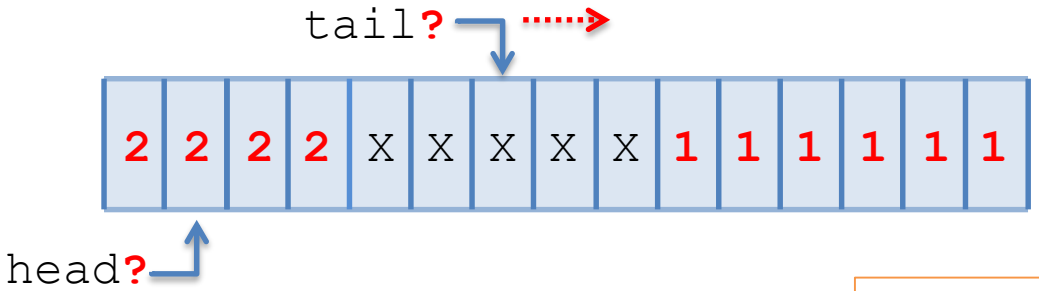
```
class Queue {
 int buffer[SIZE];
 int headish;
 int tailish;
};
```



Compromise

Review

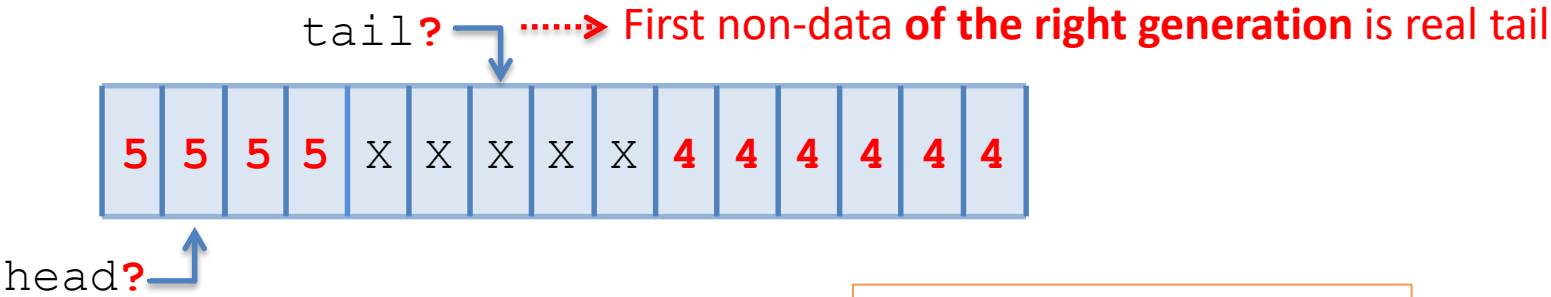
```
class Queue {
 int buffer[SIZE];
 int headish;
 int tailish;
};
```



Compromise

Review

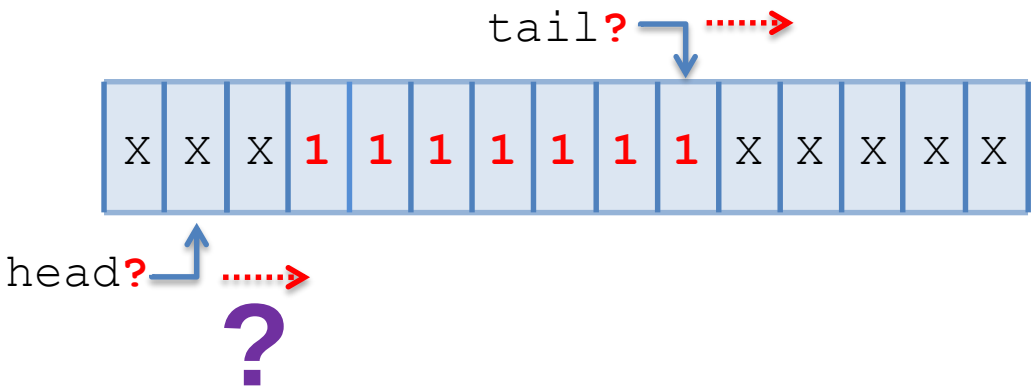
```
class Queue {
 int buffer[SIZE];
 int headish;
 int tailish;
 int generation;
};
```



Compromise

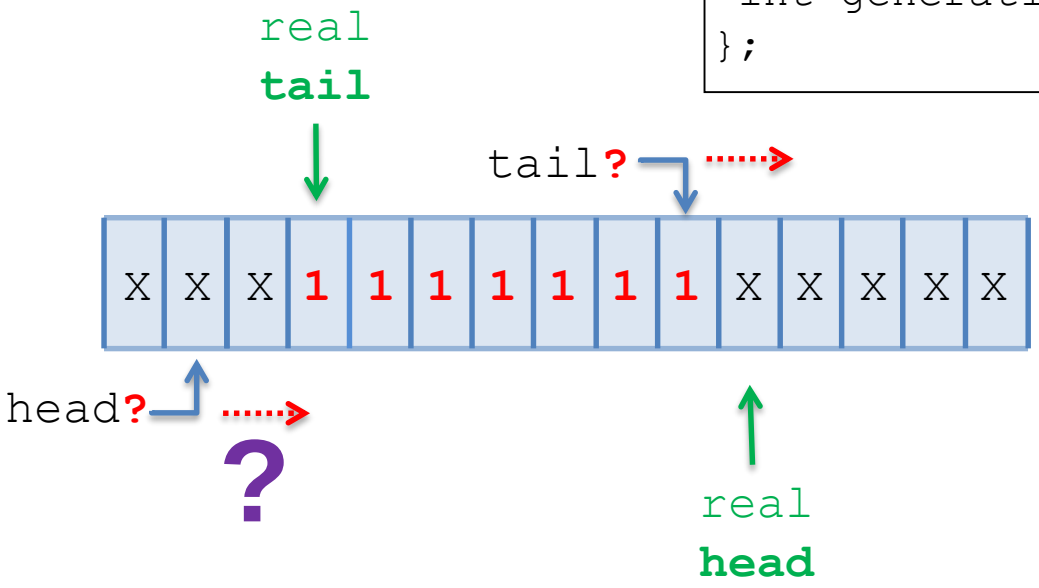
Review

```
class Queue {
 int buffer[SIZE];
 int headish;
 int tailish;
 int generation;
};
```



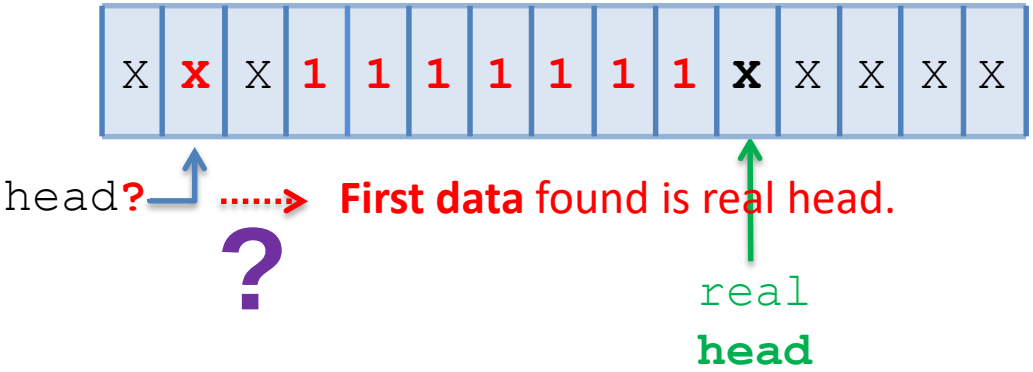
Review

```
class Queue {
 int buffer[SIZE];
 int headish;
 int tailish;
 int generation;
};
```



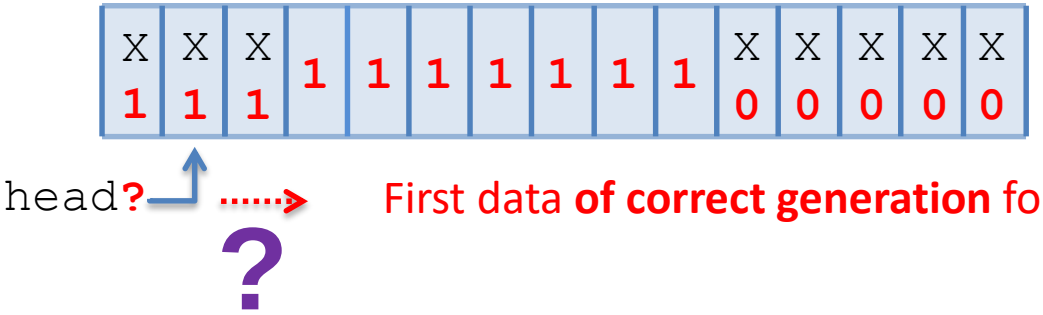
Review

```
class Queue {
 int buffer[SIZE];
 int headish;
 int tailish;
 int generation;
};
```



Review

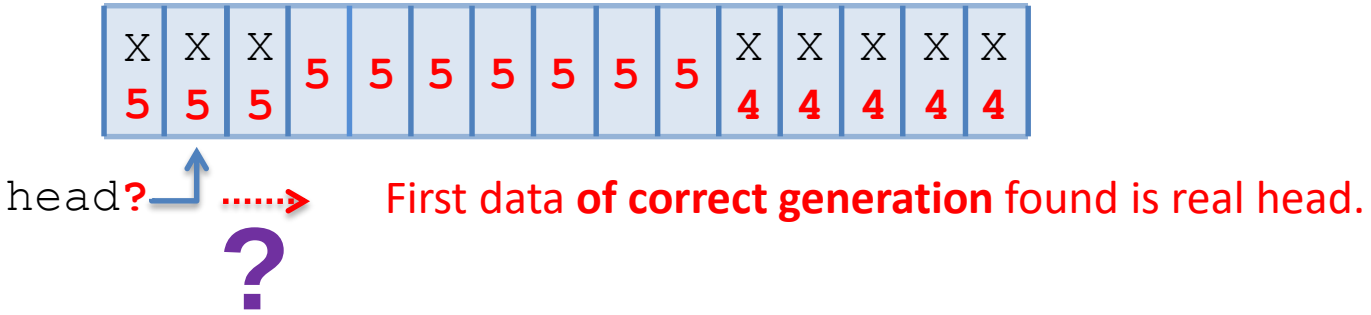
```
class Queue {
 int buffer[SIZE];
 int headish;
 int tailish;
 int generation;
};
```



First data **of correct generation** found is real head.

Review

```
class Queue {
 int buffer[SIZE];
 int headish;
 int tailish;
 int generation;
};
```

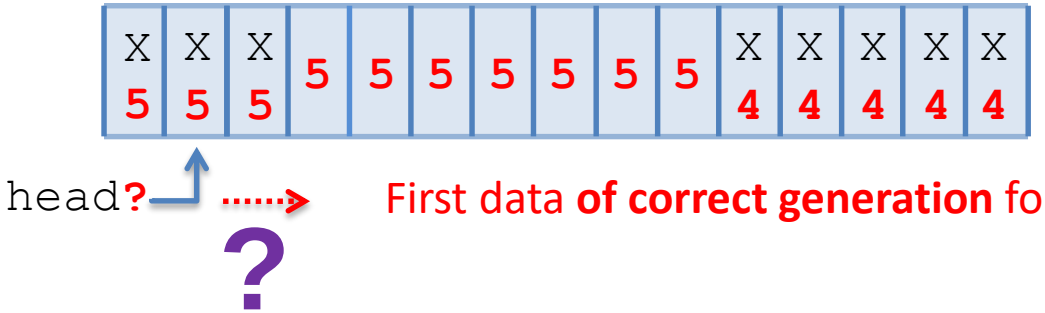


Review

```
struct entry {
 int data;
 int generation;
};
```

```
class Queue {
 entry buffer[SIZE];
 int headish;
 int tailish;
 int generation;
};
```

Compromise



First data **of correct generation** found is real head.

Review

```
struct entry {
 int data;
 int generation;
};
```

```
class Queue {
 entry buffer[SIZE];
 int headish;
 int tailish;
 int generation;
};
```

Also: data != 0

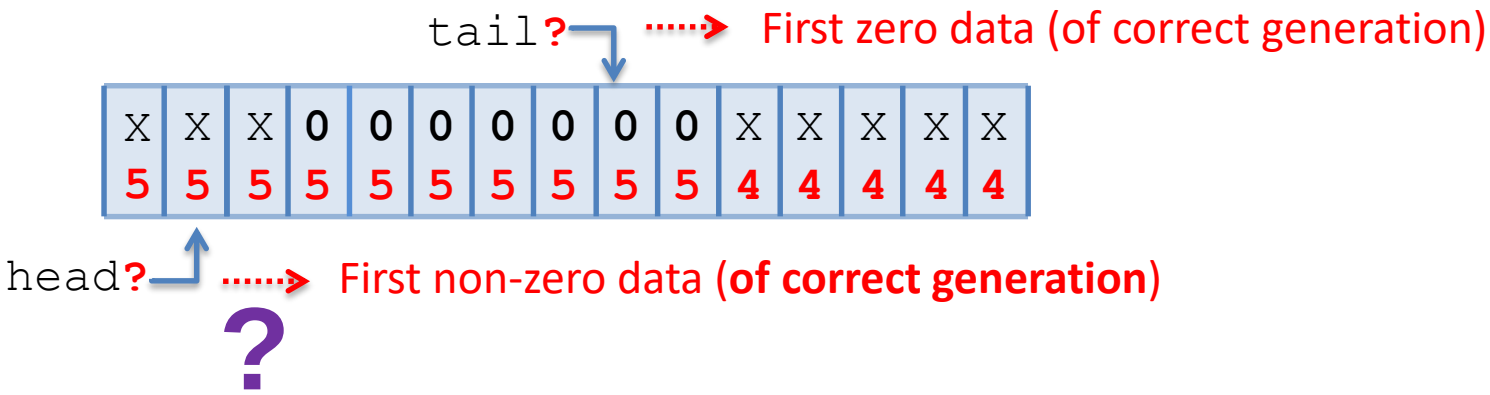
X	X	X	0	0	0	0	0	0	0	X	X	X	X	X
5	5	5	5	5	5	5	5	5	5	4	4	4	4	4

head? ?

First data **of correct generation** found is real head.

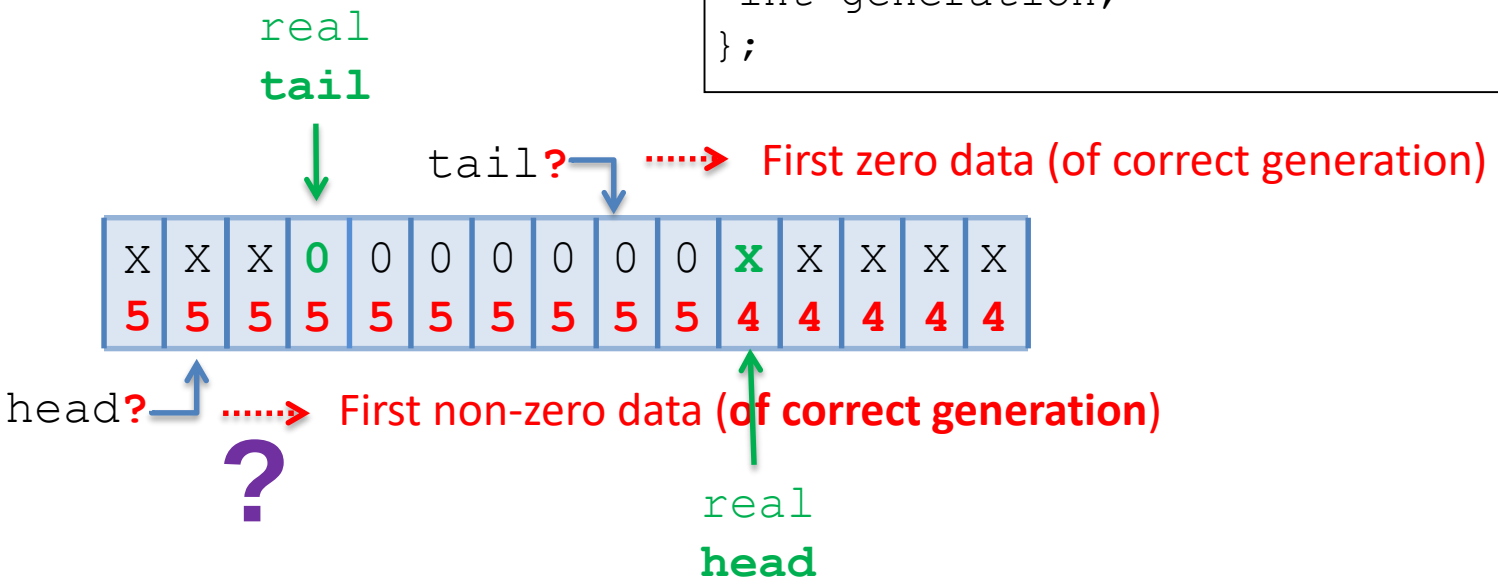
Review

```
class Queue {
 entry buffer[SIZE];
 int headish;
 int tailish;
 int generation;
};
```



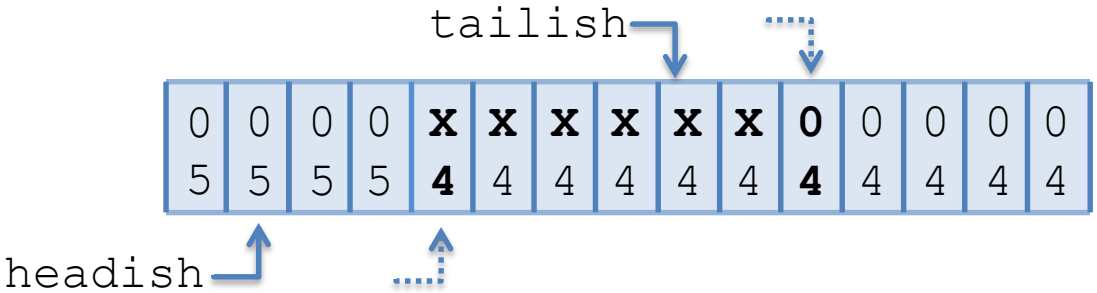
Review

```
class Queue {
 entry buffer[SIZE];
 int headish;
 int tailish;
 int generation;
};
```



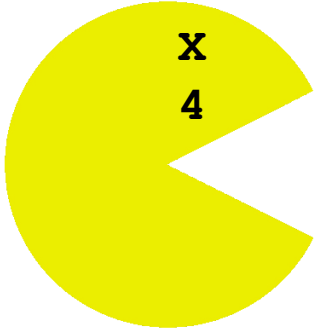
Examples

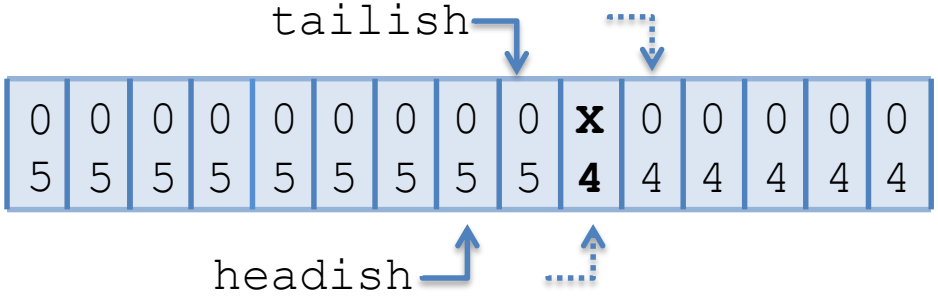
```
class Queue {
 entry buffer[SIZE];
 int headish;
 int tailish;
 int generation;
};
```



Examples

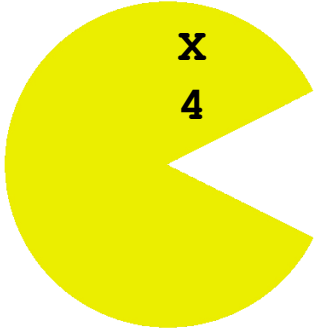
```
class Queue {
 entry buffer[SIZE];
 int headish;
 int tailish;
 int generation;
};
```

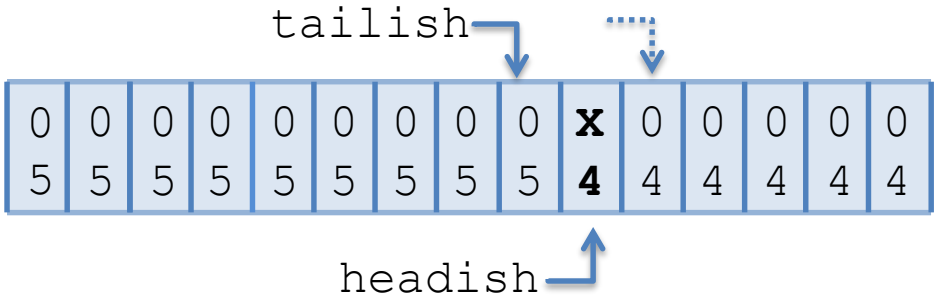




Examples

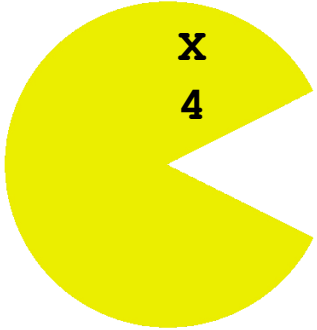
```
class Queue {
 entry buffer[SIZE];
 int headish;
 int tailish;
 int generation;
};
```

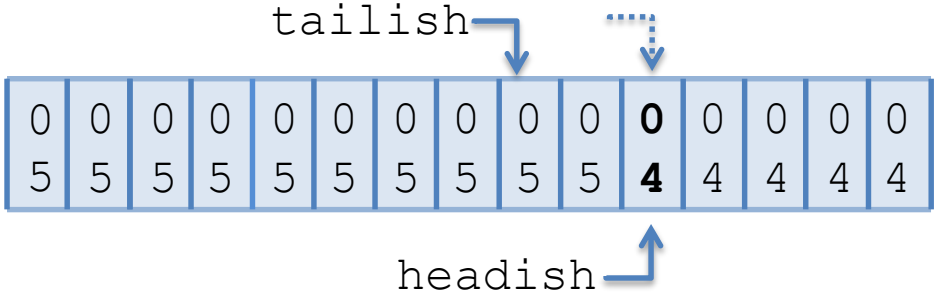




Examples

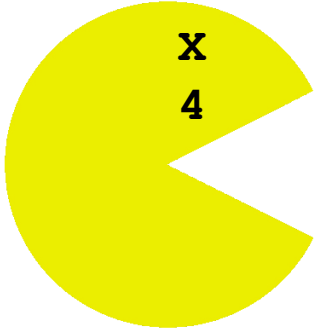
```
class Queue {
 entry buffer[SIZE];
 int headish;
 int tailish;
 int generation;
};
```

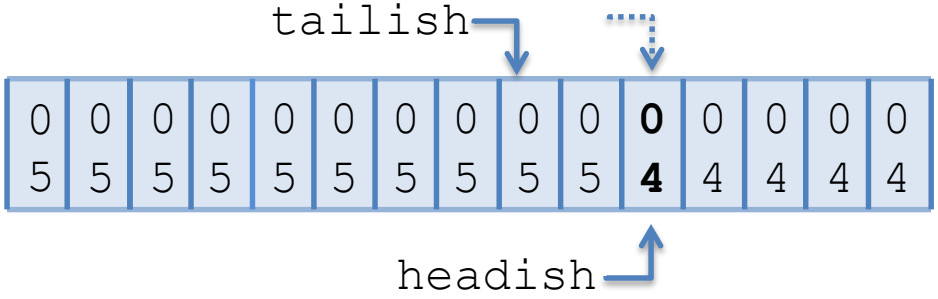




Examples

```
class Queue {
 entry buffer[SIZE];
 int headish;
 int tailish;
 int generation;
};
```

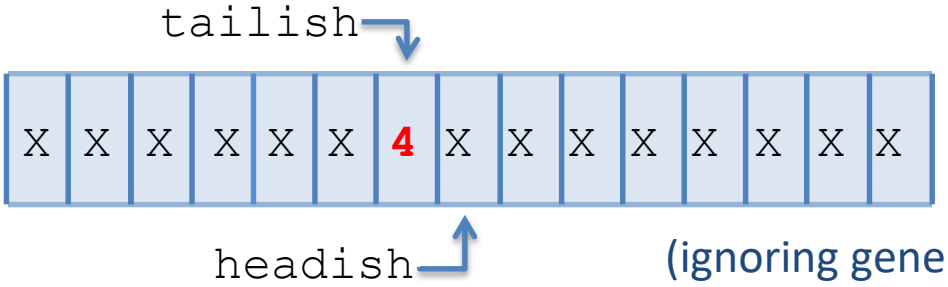




Queue is Empty!

Examples

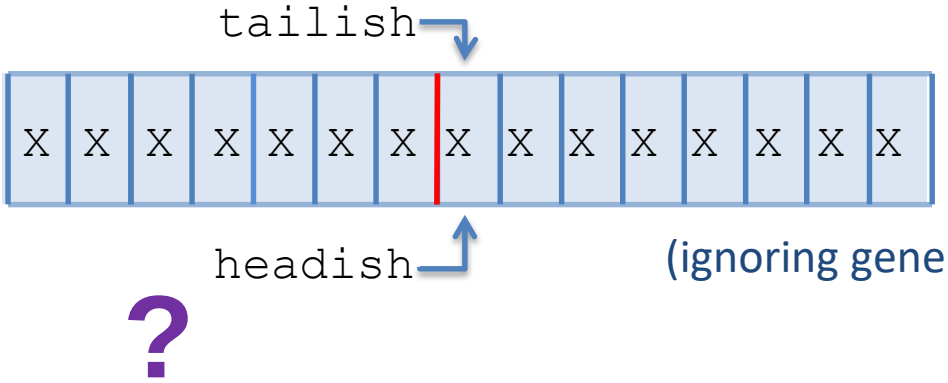
```
class Queue {
 entry buffer[SIZE];
 int headish;
 int tailish;
 int generation;
};
```



(ignoring generation for a moment)

Examples

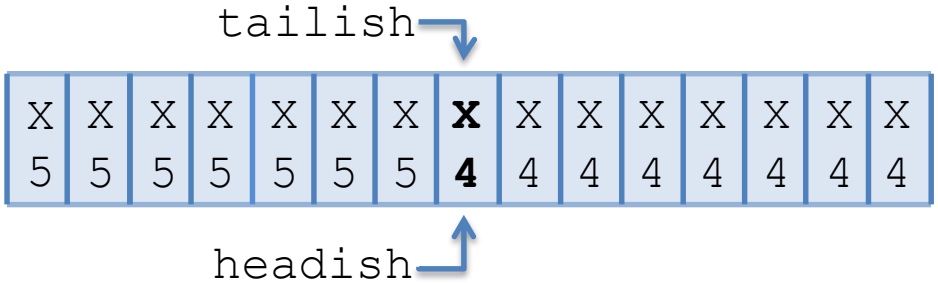
```
class Queue {
 entry buffer[SIZE];
 int headish;
 int tailish;
 int generation;
};
```



(ignoring generation for a moment)

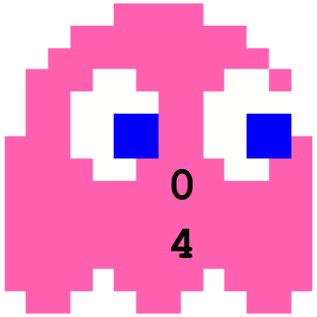
Examples

```
class Queue {
 entry buffer[SIZE];
 int headish;
 int tailish;
 int generation;
};
```

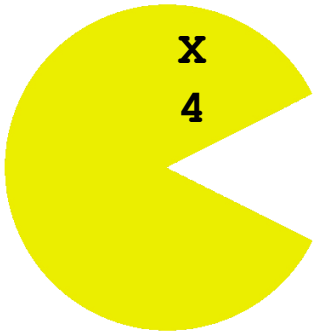


Queue is Full!

Examples




```
class Queue {
 entry buffer[SIZE];
 int headish;
 int tailish;
 int generation;
};
```

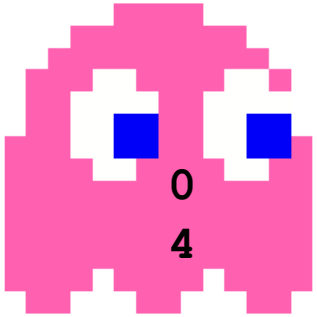


tailish

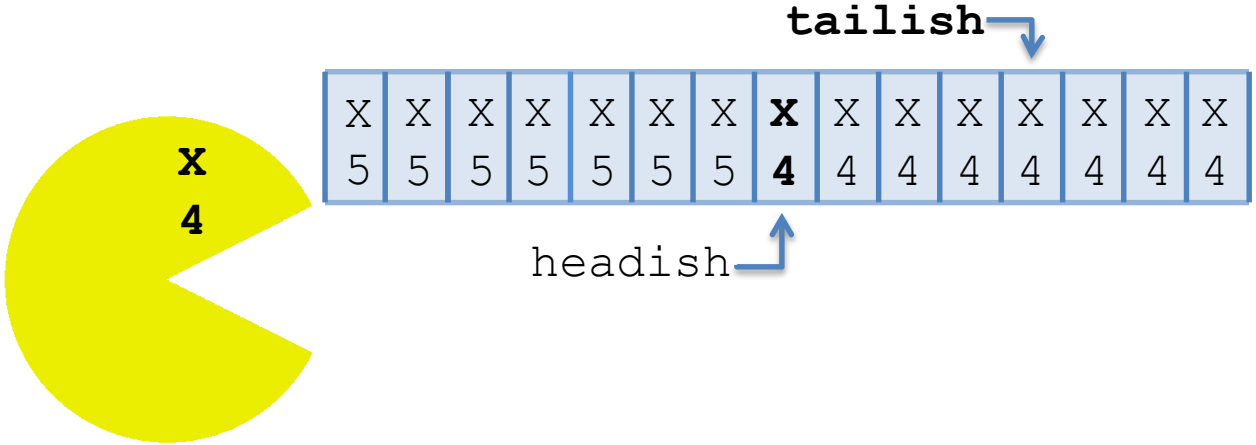
X	X	X	X	X	X	X	**X**	X	X	X	X	X	X	X
5	5	5	5	5	5	5	**4**	4	4	4	4	4	4	4

headish

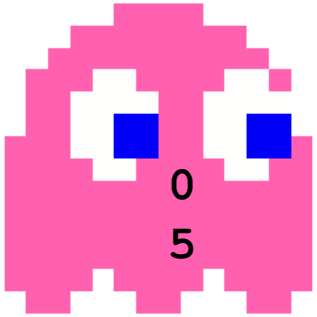
Examples



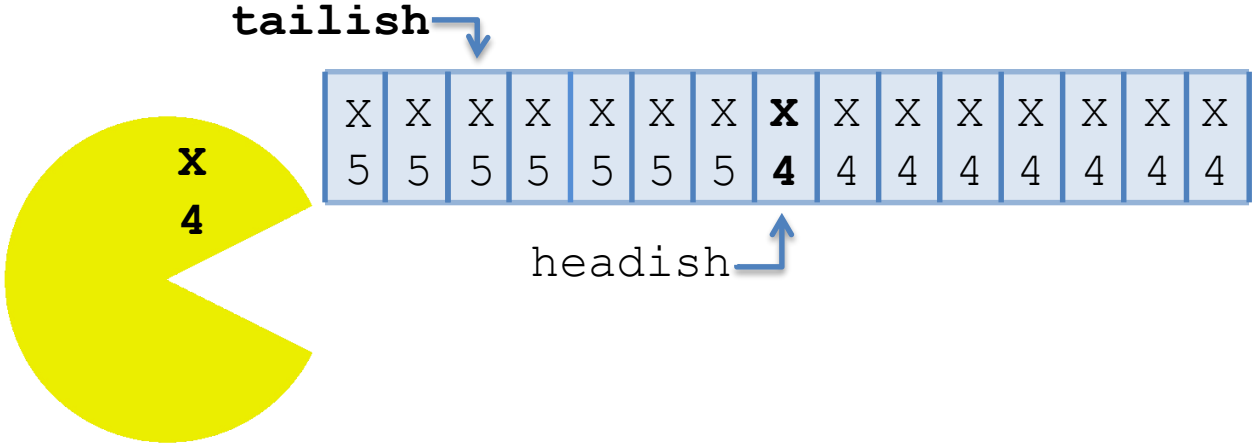

```
class Queue {
 entry buffer[SIZE];
 int headish;
 int tailish;
 int generation;
};
```



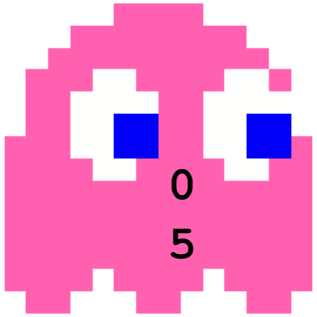
Examples




```
class Queue {
 entry buffer[SIZE];
 int headish;
 int tailish;
 int generation;
};
```



Examples



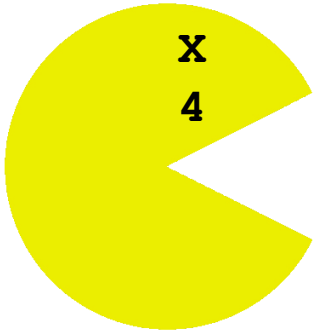

```
class Queue {
 entry buffer[SIZE];
 int headish;
 int tailish;
 int generation;
};
```

tailish

4 < 5

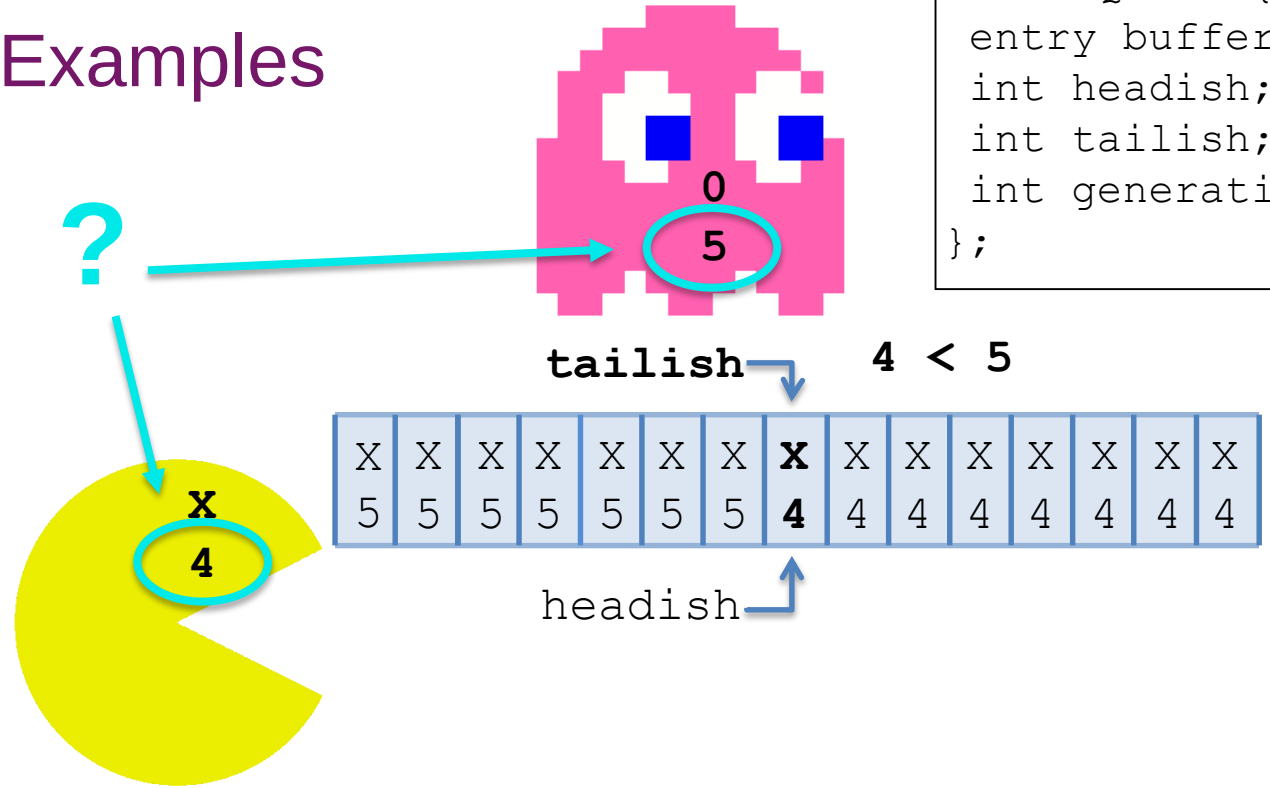
X	X	X	X	X	X	X	**X**	X	X	X	X	X	X	X
5	5	5	5	5	5	5	**4**	4	4	4	4	4	4	4

headish



Queue is Full!

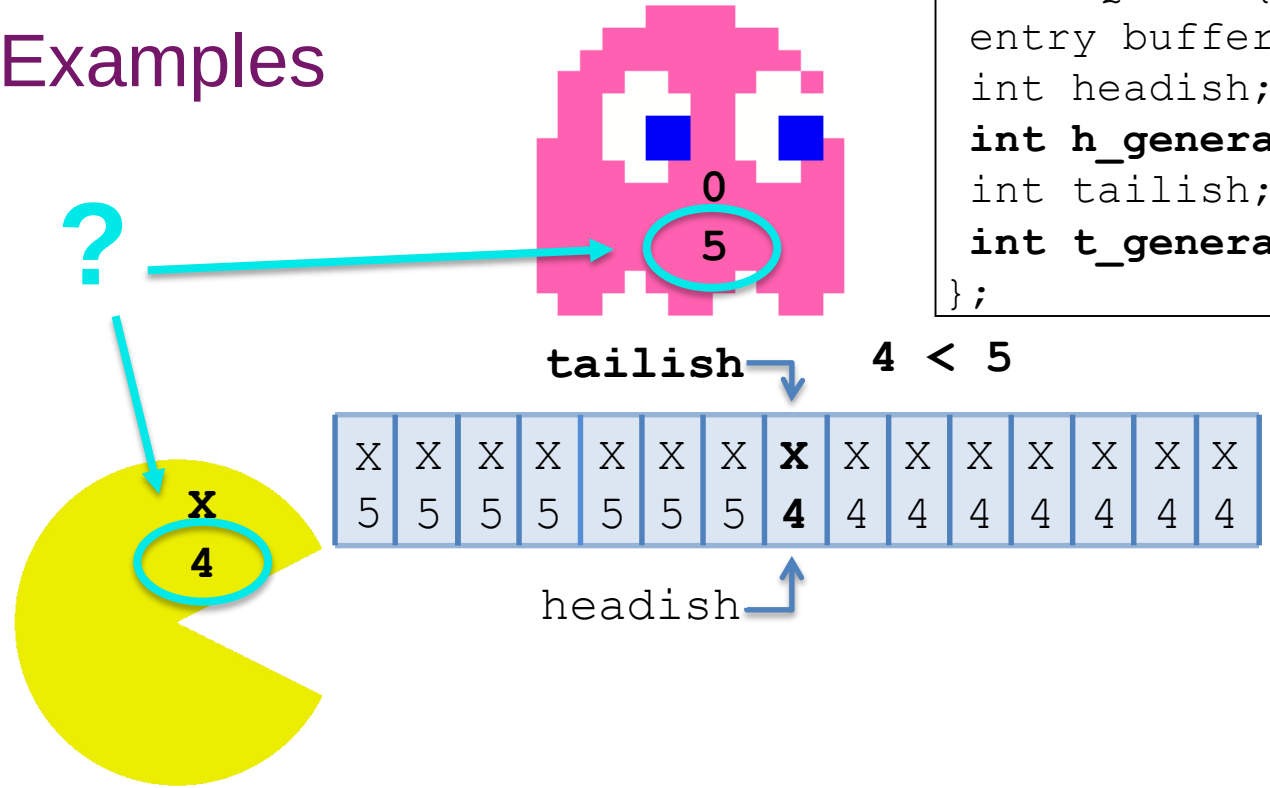
Examples




```
class Queue {
 entry buffer[SIZE];
 int headish;
 int tailish;
 int generation;
};
```

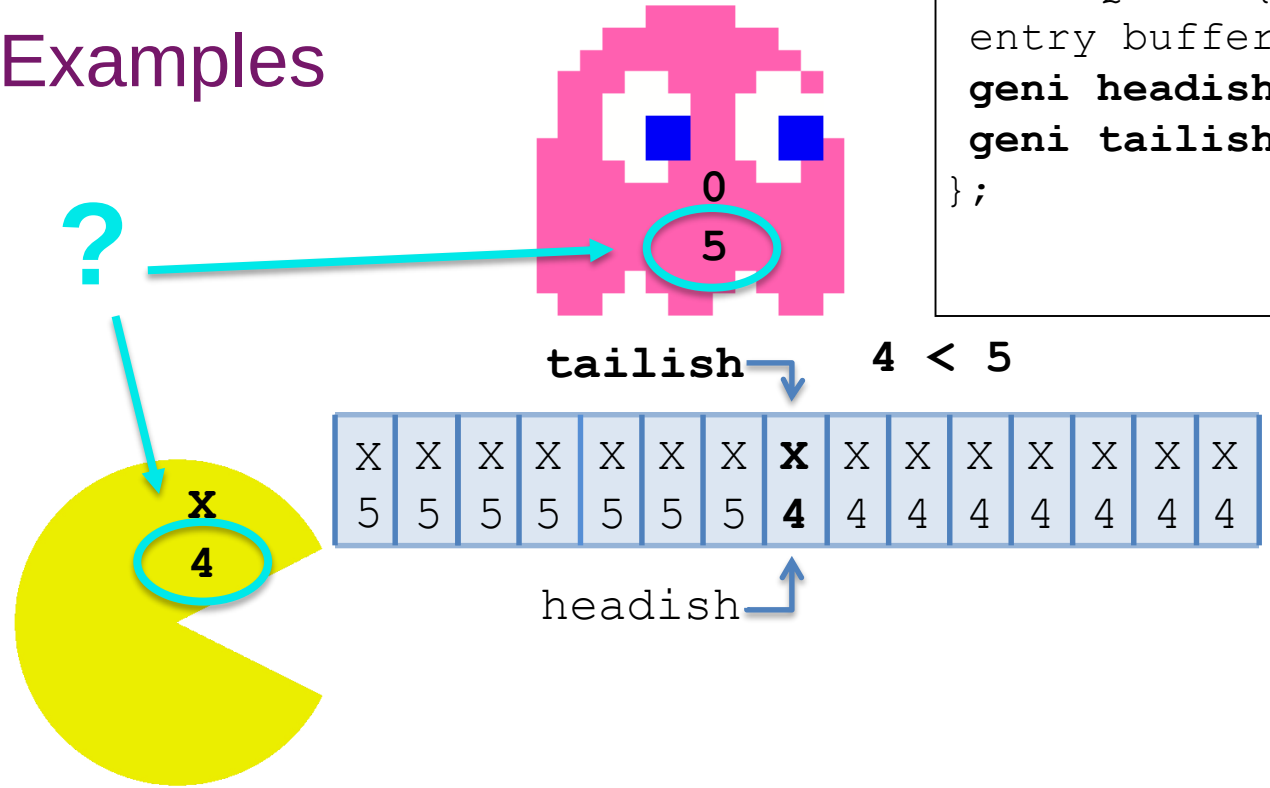
Queue is Full!

Examples




```
class Queue {
 entry buffer[SIZE];
 int headish;
 int h_generation;
 int tailish;
 int t_generation;
};
```

Examples

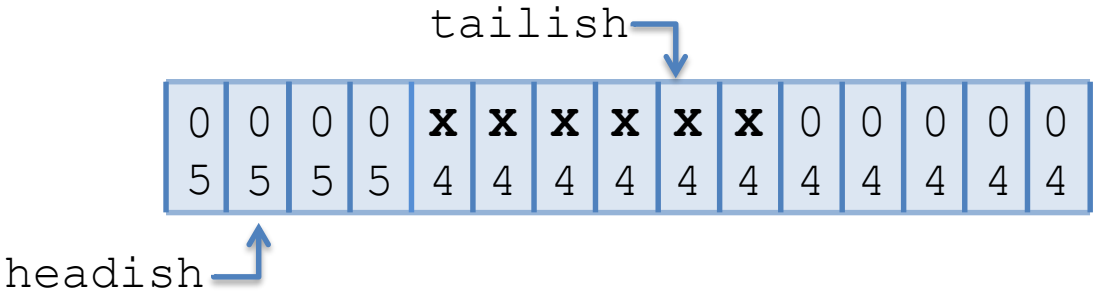



```
class Queue {
 entry buffer[SIZE];
 geni headish;
 geni tailish;
};
```

```
struct geni {
 int val, gen;
 void incr()
 { if (++val%SIZE == 0)
   {   val = 0;
       gen++;
   }
 }
 operator int() {
    return val; }
 operator<() = default;
 bool is_data(int g)
 { return val != 0
       && gen == g; }
 bool is_zero(int g)
 { return val == 0
       && gen == g; }
};
```

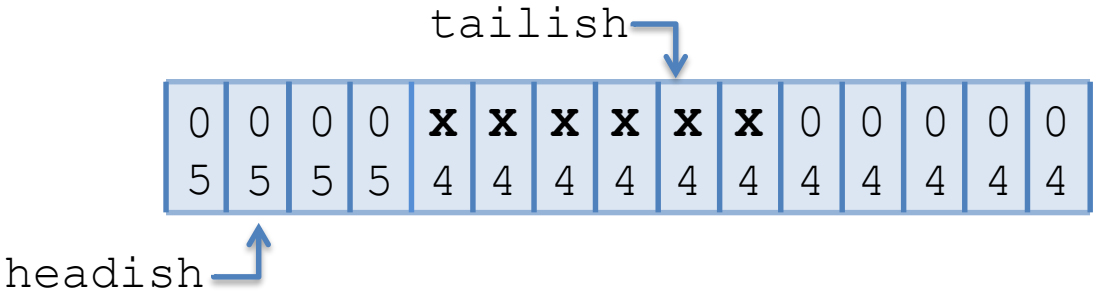
Atomicization!

```
class Queue {
 entry buffer[SIZE];
 geni headish;
 geni tailish;
};
```



Atomicization

```
class Queue {
 atomic<entry> buffer[SIZE];
 geni headish;
 geni tailish;
};
```

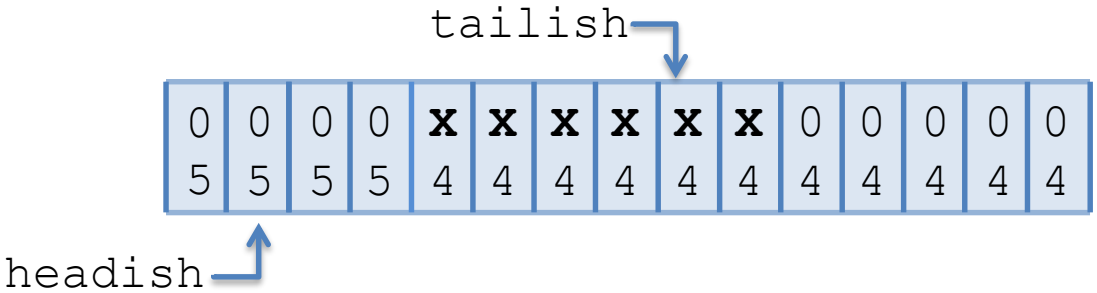


Atomicization

small enough

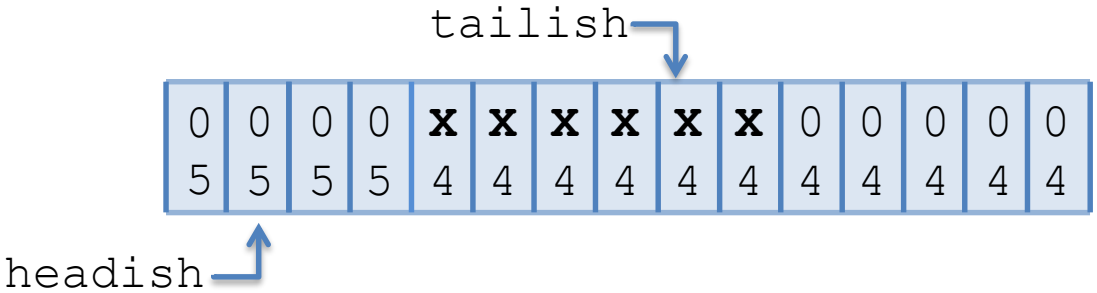
```
struct entry {
 int data;
 int generation;
};
```

```
class Queue {
 atomic<entry> buffer[SIZE];
 geni headish;
 geni tailish;
};
```



Atomicization

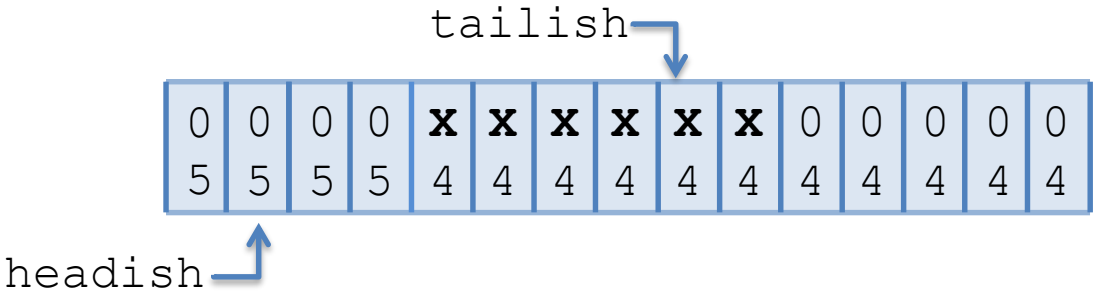
```
class Queue {
 atomic<entry> buffer[SIZE];
 laxtomic<geni> headish;
 laxtomic<geni> tailish;
};
```



Atomicization

?

```
class Queue {
 atomic<entry> buffer[SIZE];
 laxtomic<geni> headish;
 laxtomic<geni> tailish;
};
```



Atomicization

?

```
class Queue {
 atomic<entry> buffer[SIZE];
 laxtomic<geni> headish;
 laxtomic<geni> tailish;
};
```

Sorry Herb…

0	0	0	0	**X**	**X**	**X**	**X**	**X**	**X**	0	0	0	0	0
5	5	5	5	4	4	4	4	4	4	4	4	4	4	4

headish

Atomicization

```
class Queue {
 atomic<entry> buffer[SIZE];
 laxtomic<geni> headish;
 laxtomic<geni> tailish;
};
```

```
template <typename T> struct laxtomic : atomic<T> {
   …function name…(…, memory_order = memory_order_relaxed){…}
};
```

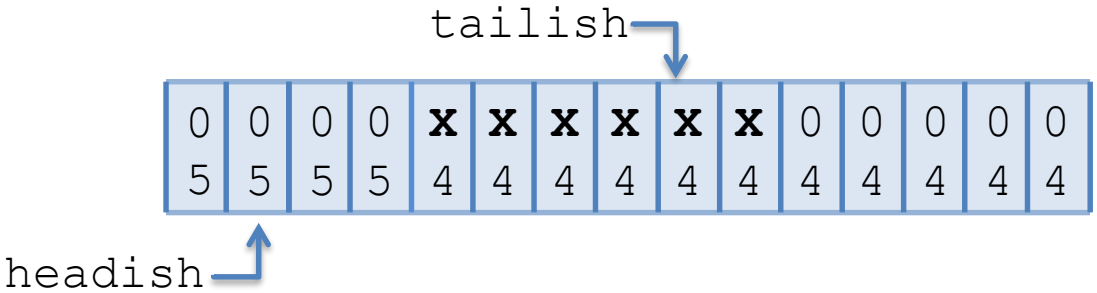
0	0	0	0	**X**	**X**	**X**	**X**	**X**	**X**	0	0	0	0	0
5	5	5	5	4	4	4	4	4	4	4	4	4	4	4

headish

Atomicization

OK?

```
class Queue {
 atomic<entry> buffer[SIZE];
 laxtomic<geni> headish;
 laxtomic<geni> tailish;
};
```



Atomicization

```
class Queue {
 atomic<entry> buffer[SIZE];
 laxtomic<geni> headish;
 geni> tailish;
```

“normal” lock free code

```
data.x = 10;
data.y = 20;
data.ready.store(true,release);
```

x
4

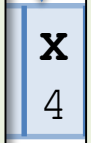
```
if (data.ready.load(acquire))
{
    x = data.x;
    y = data.y;
    ...
}
```

Atomicization

```
class Queue {
 atomic<entry> buffer[SIZE];
 laxtomic<geni> headish;
 geni> tailish;
```

“normal” lock free code

```
data.x = 10;
data.y = 20;
data.ready.store(true,release);
```




```
if (data.ready.load(acquire))
{
   x = data.x;
   y = data.y;
   ...
}
```

Atomicization

OK?

"normal" lock free code

```
class Queue {
 atomic<entry> buffer[SIZE];
 laxtomic<geni> headish;
 geni> tailish;
```

```
data.x = 10;
data.y = 20;
data.ready.store(true,release);
data.x = 10; // can't happen!
```

x
4

```
   x = data.x; // can't happen!
if (data.ready.load(acquire))
{
   x = data.x;
   y = data.y;
   ...
}
```

Atomicization

```
class Queue {
 atomic<entry> buffer[SIZE];
 laxtomic<geni> headish;
 geni> tailish;
```

“normal” lock free code

```
data.x = 10;
data.y = 20;
data.ready.store(true,release);
data.x = 10; // can't happen!
```

release:
before means before

x
4

```
   x = data.x; // can't happen!
if (data.ready.load(acquire))
{
   x = data.x;
   y = data.y;
   ...
}
```

acquire:
after means after

Atomicization

```
class Queue {
 atomic<entry> buffer[SIZE];
 laxtomic<geni> headish;
 geni> tailish;
```

relaxed lock free code

```
data.x = 10;
data.y = 20;
data.ready.store(true,mo_relax);
data.x = 10; // CAN happen!
```

x
4

```
   x = data.x; // CAN happen!
if (data.ready.load(mo_relax))
{
   x = data.x;
   y = data.y;
   ...
}
```

Atomicization

```
class Queue {
 atomic<entry> buffer[SIZE];
 laxtomic<geni> headish;
 geni> tailish;
```

“normal” lock free code

```
data.x = 10;
data.y = 20;
data.ready.store(true,release);
data.x = 10; // can't happen!
```

x
4

```
x = data.x; // can't happen!
if (data.ready.load(acquire))
{
  x = data.x;
  y = data.y;
  ...
```

there is a *relationship* between `ready` and the other data

Atomicization

```
class Queue {
 atomic<entry> buffer[SIZE];
 laxtomic<geni> headish;
 geni> tailish;
```

relaxed lock free code

```
data.x = 10;
data.y = 20;
data.ready.store(true,mo_relax);
data.x = 10; // CAN happen!
```

x
4

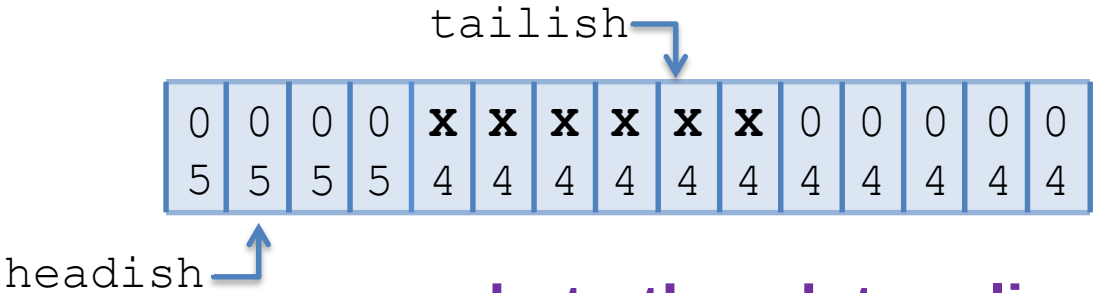
```
x = data.x; // CAN happen!
if (data.ready.load(mo_relax))
{
 x = data.x;
 y = data.y;
 ...
```

there is *no relationship* between `ready` and the other data

Atomicization

OK?

```
class Queue {
 atomic<entry> buffer[SIZE];
 laxtomic<geni> headish;
 laxtomic<geni> tailish;
};
```

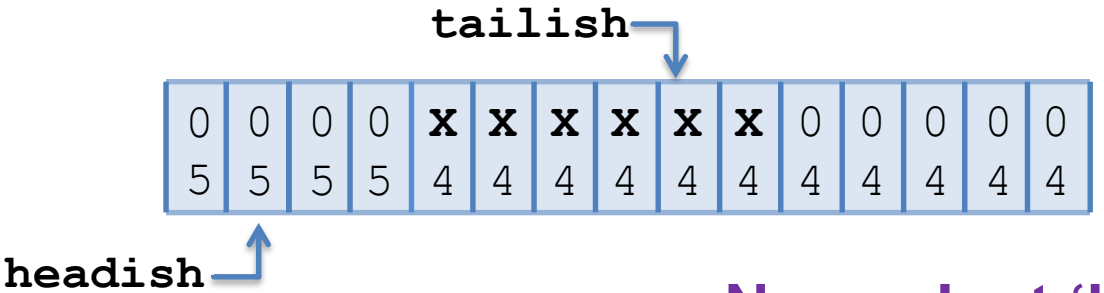


what other data relies on head/tail?

Atomicization

OK

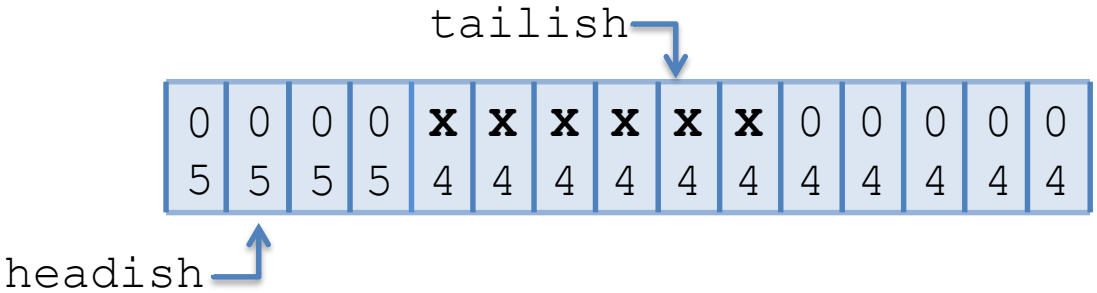
```
class Queue {
 atomic<entry> buffer[SIZE];
 laxtomic<geni> headish;
 laxtomic<geni> tailish;
};
```



None. Just ‘hints’.

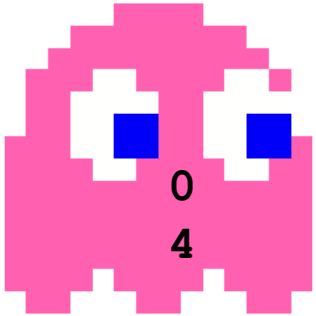
Atomicization

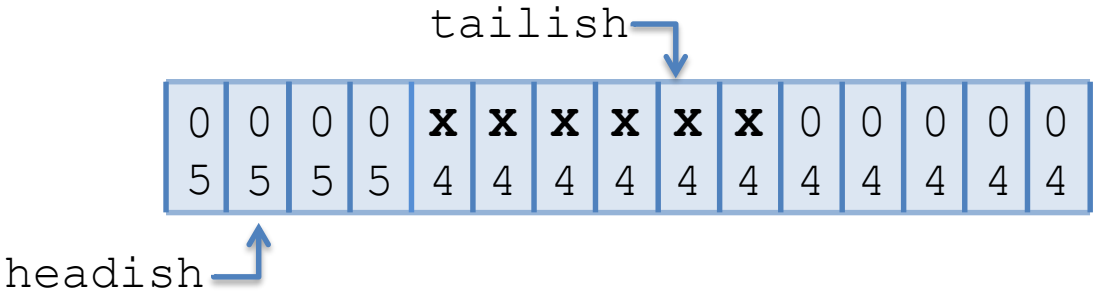
```
class Queue {
 atomic<entry> buffer[SIZE];
 laxtomic<geni> headish;
 laxtomic<geni> tailish;
};
```



Codization

```
void push(int val) {
  …
}
```





0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	4	4	4	4	4	4	4	4	4	4	4	4	4	4

0	0	0	0	0	0	0	0	0	0	**x**	**x**	**x**	**x**	**x**
5	5	5	5	5	5	5	5	5	5	**4**	4	4	4	4

x	**x**	**x**	**0**	0	0	0	0	0	0	0	0	0	0	0
4	4	4	**4**	4	4	4	4	4	4	4	4	4	4	4

x	**x**	**x**	**0**	0	0	0	0	0	0	**x**	**x**	**x**	**x**	**x**
5	5	5	**5**	5	5	5	5	5	5	**4**	4	4	4	4

0	0	0	0	**x**	**x**	**x**	**x**	**x**	**x**	**0**	0	0	0	0
5	5	5	5	**4**	4	4	4	4	4	**4**	4	4	4	4

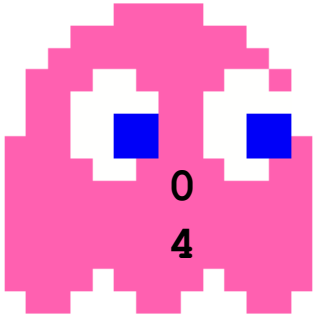
x	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**
5	5	5	5	5	5	5	5	5	5	**4**	4	4	4	4

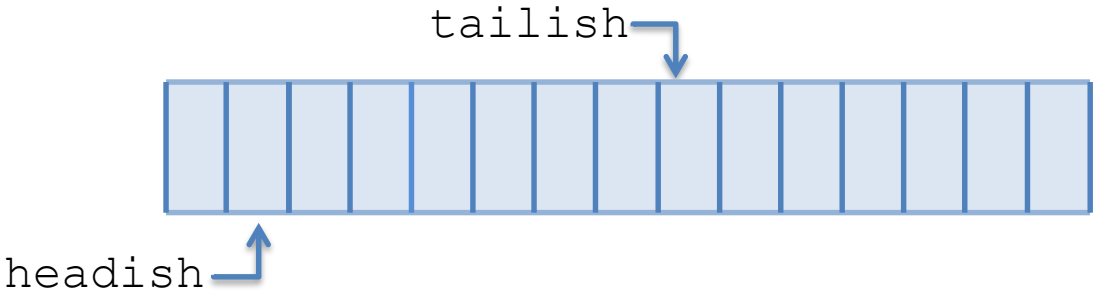
0	0	0	0	0	0	0	0	0	0	**0**	0	0	0	0
5	5	5	5	5	5	5	5	5	5	**4**	4	4	4	4

x	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5

Codization

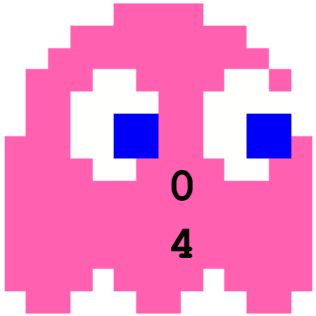
```
void push(int val) {
  …
}
```

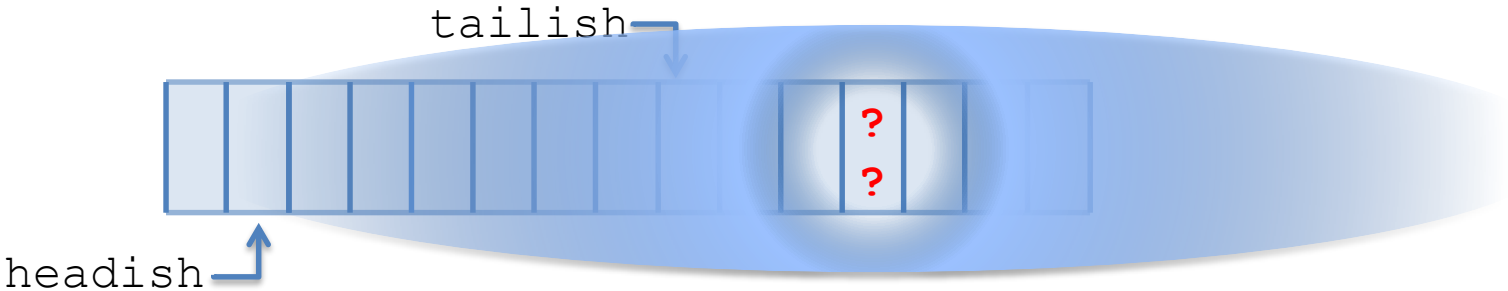




Codization

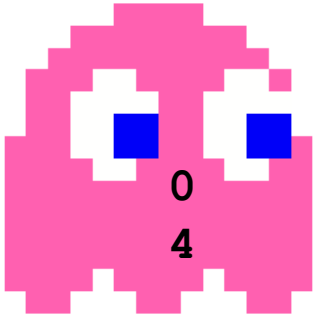
```
void push(int val) {
  …
}
```

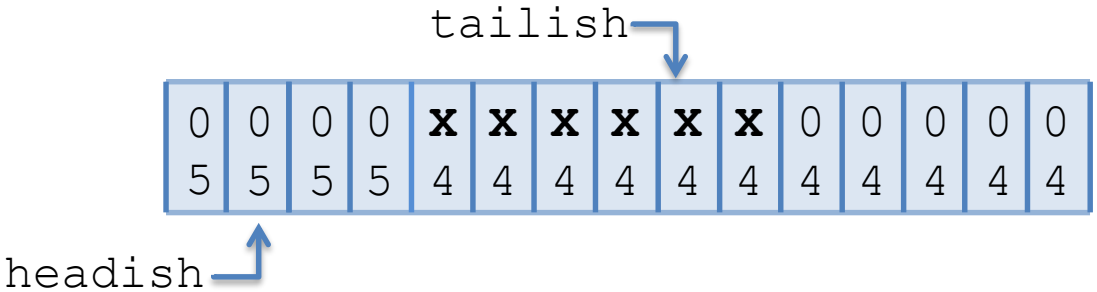




Codization

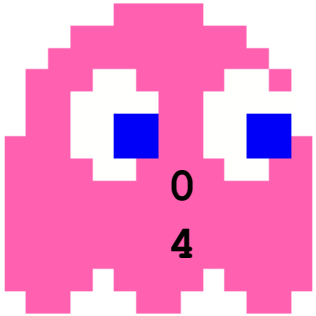
```
void push(int val) {
  …
}
```

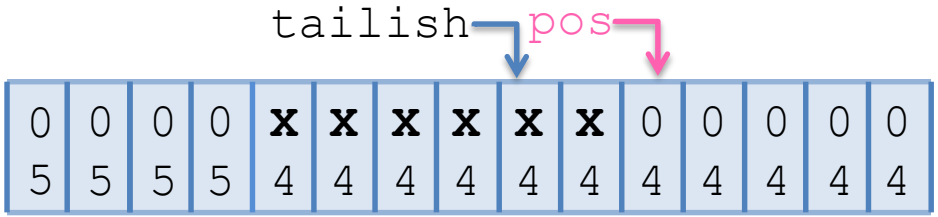




Codization

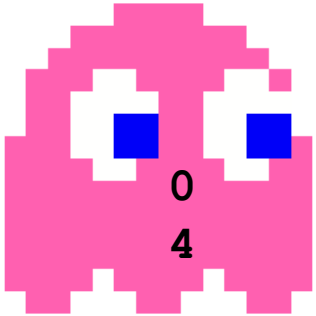
```
void push(int val) {
  geni pos = find_tail(tailish);
  write_value(pos, val);
}
```

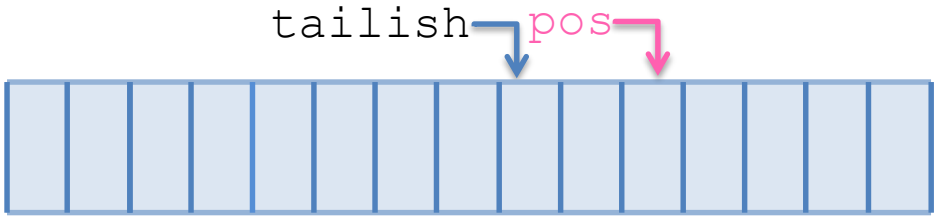




Codization

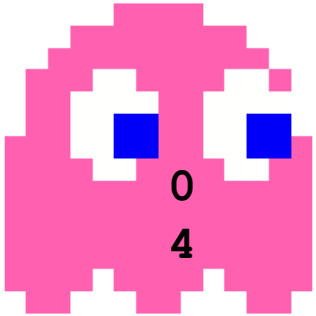
```
void push(int val) {
  geni pos = find_tail(tailish);
  write_value(pos, val);
}
```

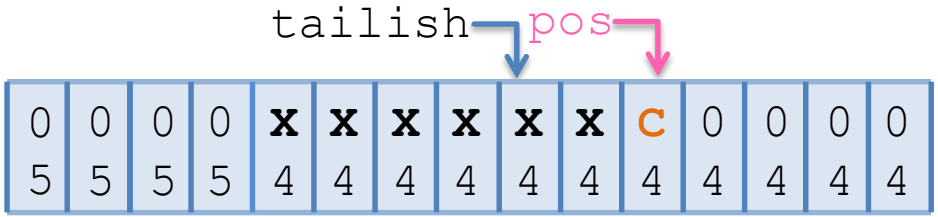


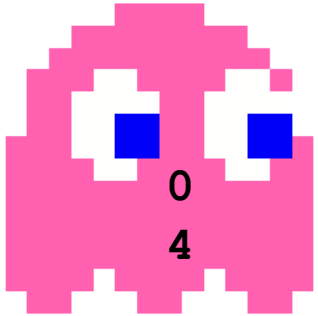


Codization

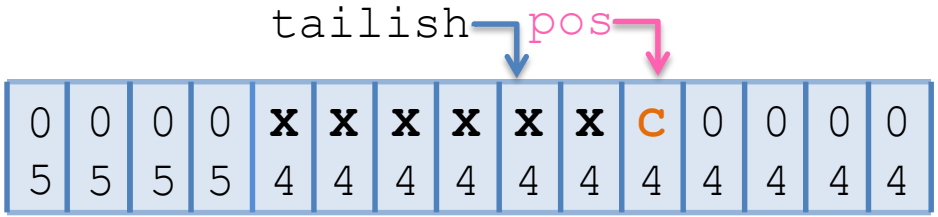
```
void push(int val) {
  geni pos = find_tail(tailish);
  write_value(pos, val);
}
```

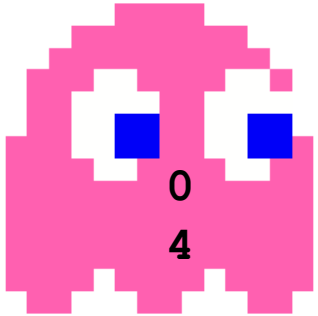




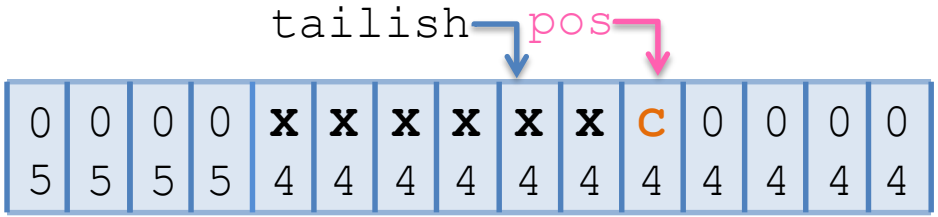



```
void push(int val) {
  do {
    pos = find_tail(tailish);
  } while (!try_write_value(pos, val));
}
```

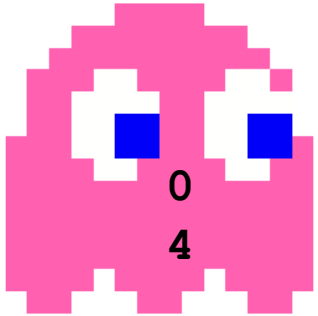




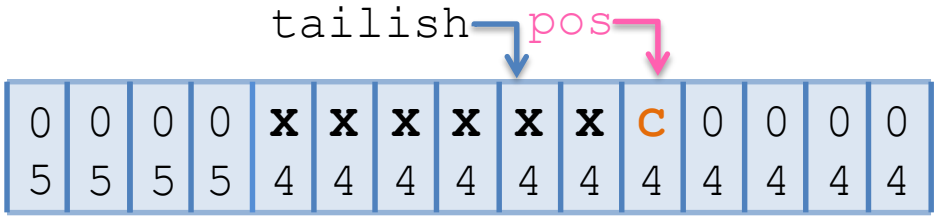

```
void push(int val) {
  do {
    pos = find_tail(tailish);
  } while (!try_write_value(pos, val));
}
```



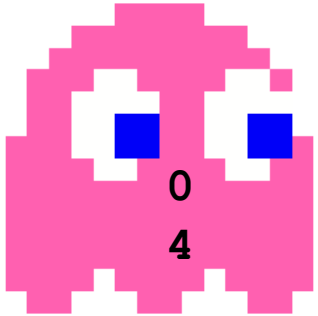
The nature of “lock-free” – you fail ***only*** when someone else makes progress



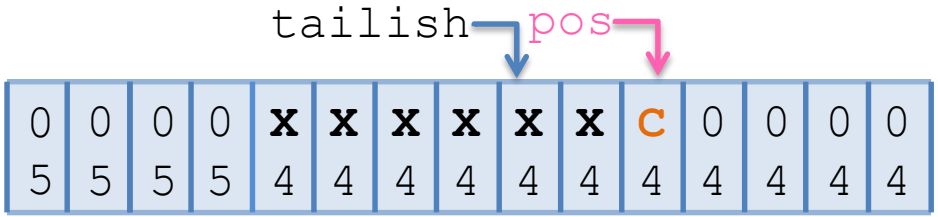

```
void push(int val) {
  do {
    pos = find_tail(tailish);
  } while (!try_write_value(pos, val));
}
```

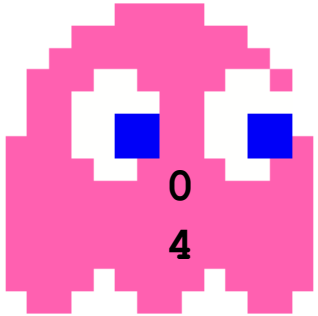


CAS loop: loops on progress
Spin-lock: loops on progress and NON progress

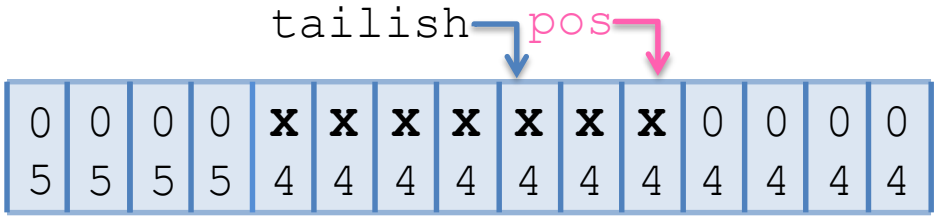



```
void push(int val) {
  geni pos = tailish; // relaxed load
  do {
    pos = find_tail(pos);
  } while (!try_write_value(pos, val));
}
```



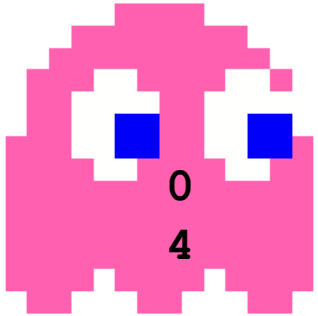



```
void push(int val) {
  geni pos = tailish; // relaxed load
  do {
    pos = find_tail(pos);
  } while (!try_write_value(pos, val));
}
```

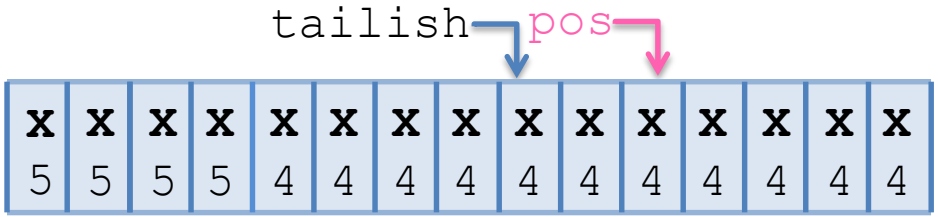



```
bool is_zero(entry e, int gen) {
   return e.data == 0 && e.gen == gen;
}
```

```
geni find_tail(geni pos) {
   while( ! is_zero(buffer[pos.val].load(relaxed), pos.gen) )
       pos++;
   return pos;
}
```



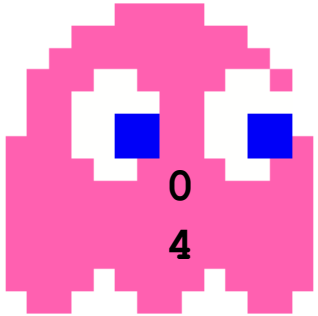

```
void push(int val) {
  geni pos = tailish; // relaxed load
  do {
    pos = find_tail(pos);
  } while (!try_write_value(pos, val));
}
```



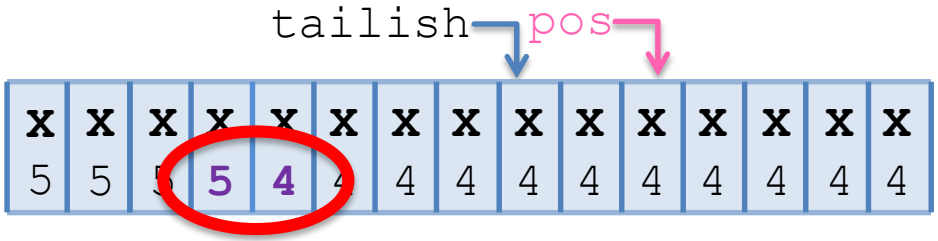
Queue is FULL

```
bool is_zero(entry e, int gen) {
   return e.data == 0 && e.gen == gen;
}
```

```
geni find_tail(geni pos) {
   while( ! is_zero(buffer[pos.val].load(relaxed), pos.gen) )
       pos++;
   return pos;
}
```



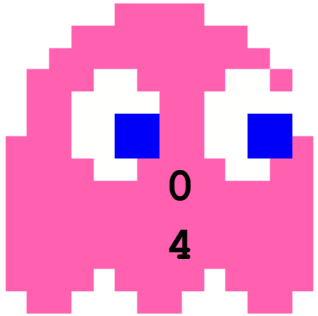

```
void push(int val) {
  geni pos = tailish; // relaxed load
  do {
    pos = find_tail(pos);
  } while (!try_write_value(pos, val));
}
```



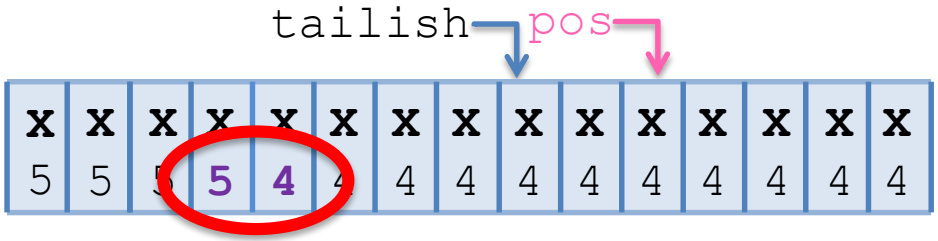
Queue is FULL

```
bool is_tail(entry e, int gen) {
   return e.data == 0 && e.gen == gen
          || e.data  != 0 && e.gen < gen;
}
```

```
geni find_tail(geni pos) {
   while( ! is_tail(buffer[pos.val].load(relaxed), pos.gen) )
       pos++;
   return pos;
}
```



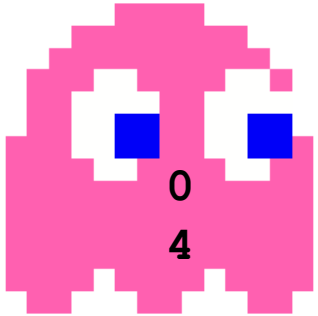

```
void push(int val) {
  geni pos = tailish; // relaxed load
  do {
    pos = find_tail(pos);
  } while (!try_write_value(pos, val));
}
```



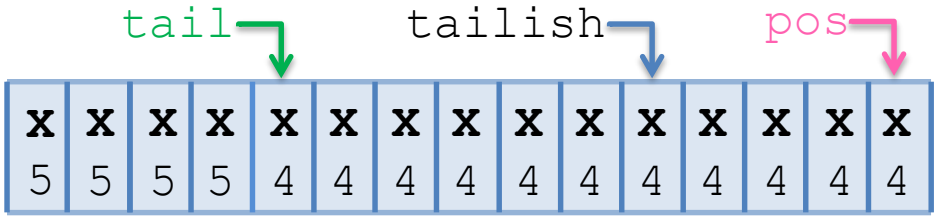
Queue is FULL

```
bool is_tail(entry e, int gen) {
  return e.data == 0  ?
        e.gen == gen  :  e.gen < gen;
}
```

```
geni find_tail(geni pos) {
  while( ! is_tail(buffer[pos.val].load(relaxed), pos.gen) )
      pos++;
  return pos;
}
```



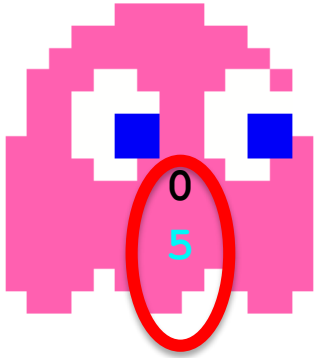

```
void push(int val) {
  geni pos = tailish; // relaxed load
  do {
    pos = find_tail(pos);
  } while (!try_write_value(pos, val));
}
```



Queue is FULL

```
bool is_tail(entry e, int gen) {
   return e.data == 0 && e.gen == gen
          || e.data  != 0 && e.gen < gen;
}
```

```
geni find_tail(geni pos) {
   while( ! is_tail(buffer[pos.val].load(relaxed), pos.gen) )
       pos++;
   return pos;
}
```

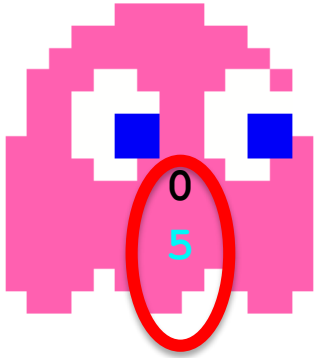



```
void push(int val) {
  geni pos = tailish; // relaxed load
  do {
    pos = find_tail(pos);
  } while (!try_write_value(pos, val));
}
```

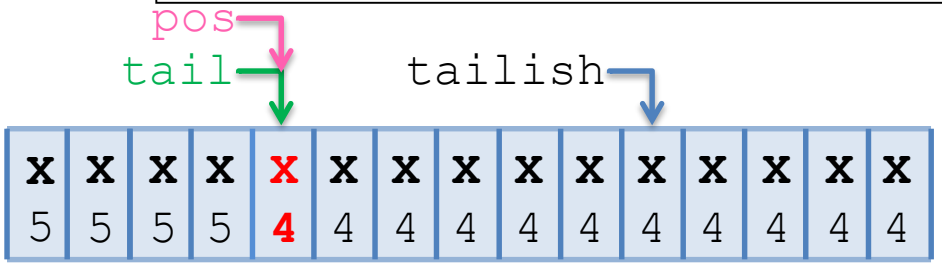
Queue is FULL

```
bool is_tail(entry e, int gen) {
   return e.data == 0 && e.gen == gen
          || e.data  != 0 && e.gen < gen;
}
```

```
geni find_tail(geni pos) {
   while( !is_tail(buffer[pos.val].load(relaxed), pos.gen) )
      pos++;
   return pos;
}
```




```
void push(int val) {
  geni pos = tailish; // relaxed load
  do {
    pos = find_tail(pos);
  } while (!try_write_value(pos, val));
}
```

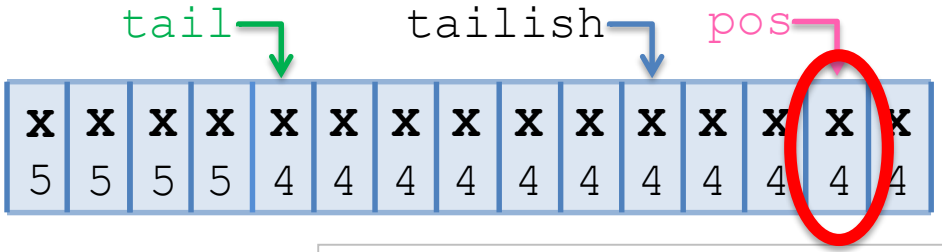


Queue is FULL

```
bool is_tail(entry e, int gen) {
   return e.data == 0 && e.gen == gen
          || e.data  != 0 && e.gen < gen;
}
```

```
geni find_tail(geni pos) {
   while( ! is_tail(buffer[pos.val].load(relaxed), pos.gen) )
       pos++;
   return pos;
}
```


```
void push(int val) {
  geni pos = tailish; // relaxed load
  do {
    pos = find_tail(pos);
  } while (!try_write_value(pos, val));
}
```

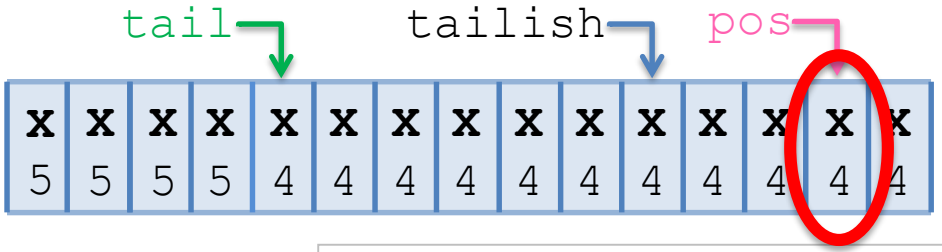


Queue is FULL

```
bool is_tail(entry e, int gen) {
  return e.data == 0 && e.gen == gen
        || e.data  != 0 && e.gen < gen;
}
```

```
geni find_tail(geni pos) {
  while( ! is_tail(buffer[pos.val].load(relaxed), pos.gen) )
      pos++;
  return pos;
}
```


```
void push(int val) {
  geni pos = tailish; // relaxed load
  do {
    pos = find_tail(pos);
  } while (!try_write_value(pos, val));
}
```



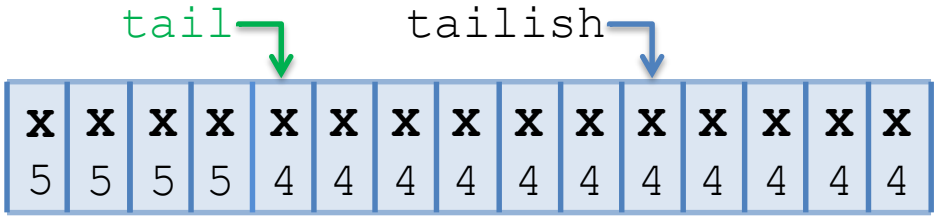
Queue is FULL

```
bool maybe_tail(entry e, int gen) {
  return e.data == 0 && e.gen == gen
         || e.data  != 0 && e.gen < gen;
}
```

```
geni find_tail(geni pos) {
  while(!maybe_tail(buffer[pos.val].load(relaxed), pos.gen))
      pos++;
  return pos;
}
```


```
void push(int val) {
  geni pos = tailish; // relaxed load
  do {
    pos = find_tail(pos);
  } while (!try_write_value(pos, val));
}
```

INVARIANT:
tailish <= real tail



Queue is FULL

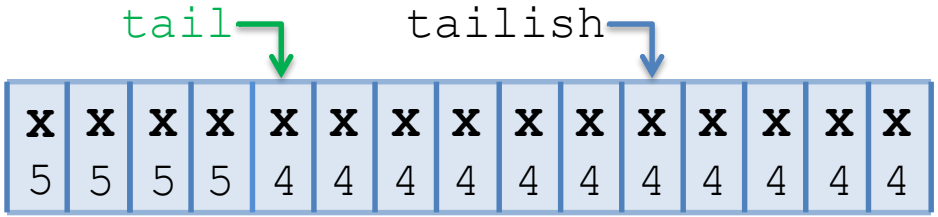
```
bool maybe_tail(entry e, int gen) {
   return e.data == 0 && e.gen == gen
          || e.data  != 0 && e.gen < gen;
}
```

```
geni find_tail(geni pos) { // precond: pos <= tail
   while(!maybe_tail(buffer[pos.val].load(relaxed), pos.gen))
       pos++;
   return pos;
}
```


```
void push(int val) {
  geni pos = tailish; // relaxed load
  do {
    pos = find_tail(pos);
  } while (!try_write_value(pos, val));
}
```

Exercise for the reader:

Don't loop on FULL. Either:
- have push() return false
- *wait* for non full.



Queue is FULL

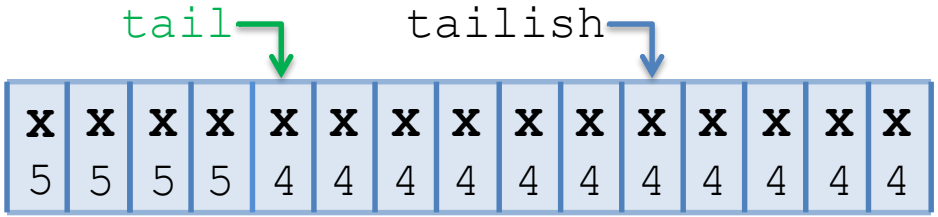
```
bool maybe_tail(entry e, int gen) {
  return e.data == 0 && e.gen == gen
        || e.data  != 0 && e.gen < gen;
}
```

```
geni find_tail(geni pos) { // precond: pos <= tail
  while(!maybe_tail(buffer[pos.val].load(relaxed), pos.gen)
      pos++;
  return pos;
}
```


```
void push(int val) {
  geni pos = tailish; // relaxed load
  do {
    pos = find_tail(pos, full);
    if (full) …
  } while (!try_write_value(pos, val));
}
```

Exercise for the reader:

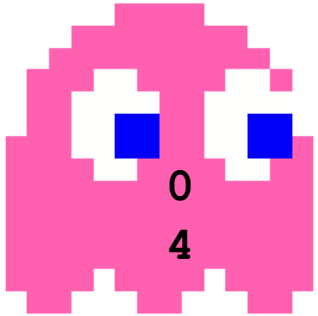
Don't loop on FULL. Either:
- have push() return false
- *wait* for non full.



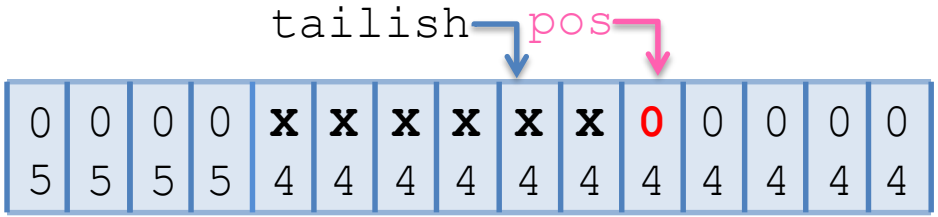
Queue is FULL

```
bool maybe_tail(entry e, int gen) {
  return e.data == 0 && e.gen == gen
        || e.data  != 0 && e.gen < gen;
}
```

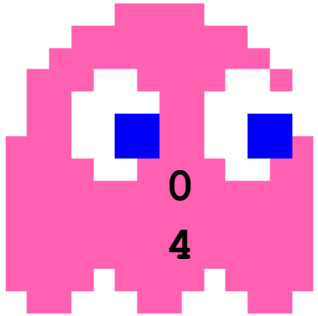
```
geni find_tail(geni pos) { // precond: pos <= tail
  while(!maybe_tail(buffer[pos.val].load(relaxed), pos.gen)
      pos++;
  return pos;
}
```



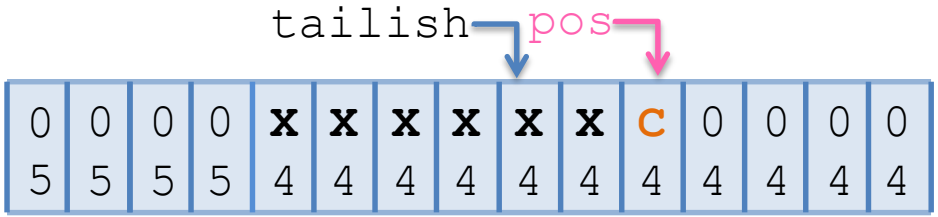

```
void push(int val) {
  geni pos = tailish; // relaxed load
  do {
    pos = find_tail(pos);
  } while (!try_write_value(pos, val));
}
```



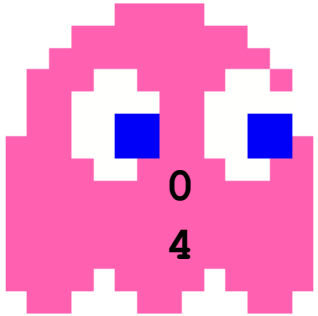

```
bool try_write_value(geni pos, int val) {



}
```



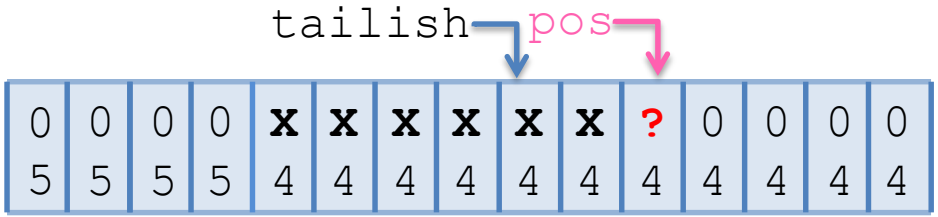

```
void push(int val) {
  geni pos = tailish; // relaxed load
  do {
    pos = find_tail(pos);
  } while (!try_write_value(pos, val));
}
```



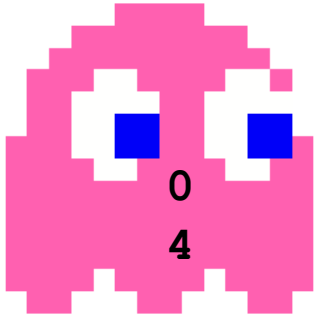
```
bool try_write_value(geni pos, int val) {



}
```




```
void push(int val) {
  geni pos = tailish; // relaxed load
  do {
    pos = find_tail(pos);
  } while (!try_write_value(pos, val));
}
```




```
bool try_write_value(geni pos, int val) {
   entry old{0, pos.gen};
   entry nu{val, pos.gen};
   return buffer[pos].compare_exchange_weak(old, nu, release, relaxed);
}
```

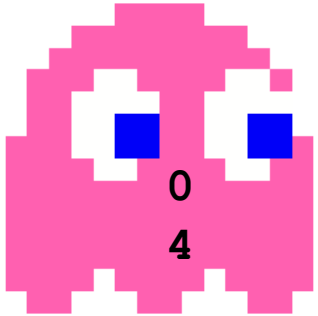



```
void push(int val) {
  geni pos = tailish; // relaxed load
  do {
    pos = find_tail(pos);
  } while (!try_write_value(pos, val));
}
```

```
bool maybe_tail(entry e, int gen) {
   return e.data == 0 && e.gen == gen
          || e.data  != 0 && e.gen < gen;
}
```

```
geni find_tail(geni pos) { // precond: pos <= tail
   while(!maybe_tail(buffer[pos.val].load(relaxed), pos.gen))
       pos++;
   return pos;
}
```

```
bool try_write_value(geni pos, int val) {
   entry old{0, pos.gen};
   entry nu{val, pos.gen};
   return buffer[pos].compare_exchange_weak(old, nu, release, relaxed);
}
```

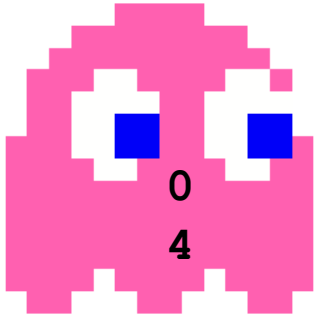



```
void push(int val) {
  geni pos = tailish; // relaxed load
  do {
    pos = find_tail(pos);
  } while (!try_write_value(pos, val));
  // ???
}
```

```
bool maybe_tail(entry e, int gen) {
   return e.data == 0 && e.gen == gen
          || e.data  != 0 && e.gen < gen;
}
```

```
geni find_tail(geni pos) { // precond: pos <= tail
   while(!maybe_tail(buffer[pos.val].load(relaxed), pos.gen))
       pos++;
   return pos;
}
```

```
bool try_write_value(geni pos, int val) {
   entry old{0, pos.gen};
   entry nu{val, pos.gen};
   return buffer[pos].compare_exchange_weak(old, nu, release, relaxed);
}
```

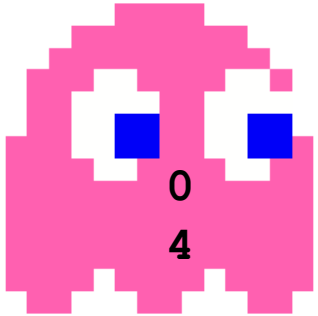



```
void push(int val) {
  geni pos = tailish; // relaxed load
  do {
    pos = find_tail(pos);
  } while (!try_write_value(pos, val));
  update_tailish(tailish, pos);
}
```

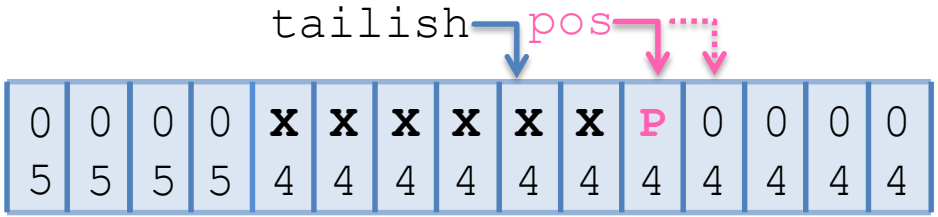
```
bool maybe_tail(entry e, int gen) {
   return e.data == 0 && e.gen == gen
          || e.data  != 0 && e.gen < gen;
}
```

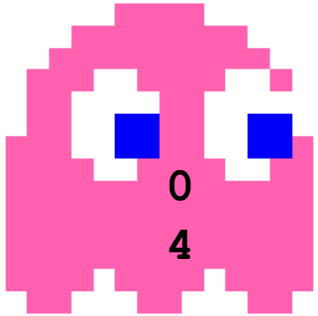
```
geni find_tail(geni pos) { // precond: pos <= tail
   while(!maybe_tail(buffer[pos.val].load(relaxed), pos.gen))
       pos++;
   return pos;
}
```

```
bool try_write_value(geni pos, int val) {
   entry old{0, pos.gen};
   entry nu{val, pos.gen};
   return buffer[pos].compare_exchange_weak(old, nu, release, relaxed);
}
```

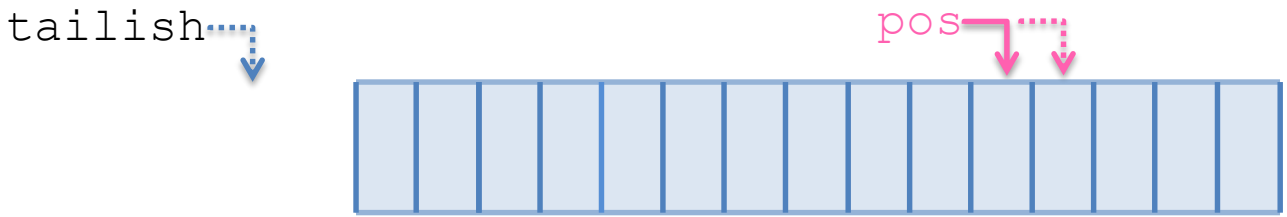


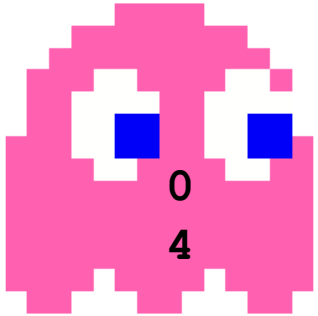

```
void push(int val) {
  geni pos = tailish; // relaxed load
  do {
    pos = find_tail(pos);
  } while (!try_write_value(pos, val));
  update_tailish(tailish, ++pos);
}
```



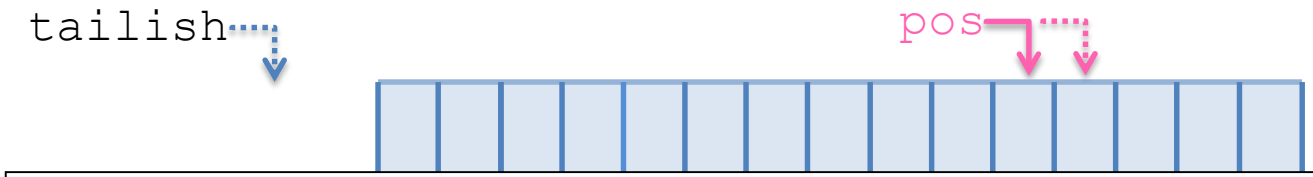



```
void push(int val) {
  geni pos = tailish; // relaxed load
  do {
    pos = find_tail(pos);
  } while (!try_write_value(pos, val));
  update_tailish(tailish, ++pos);
}
```

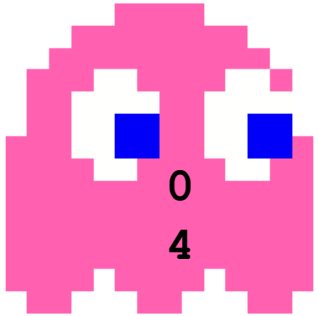




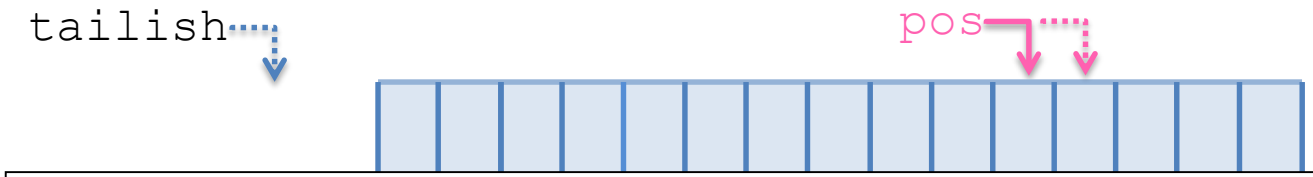

```
void push(int val) {
  geni pos = tailish; // relaxed load
  do {
    pos = find_tail(pos);
  } while (!try_write_value(pos, val));
  update_tailish(tailish, ++pos);
}
```



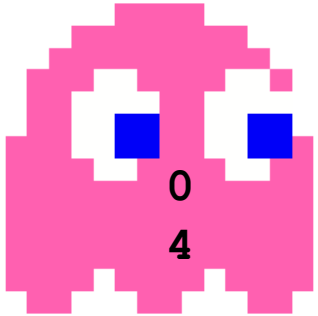

```
tailish.compare_exchange(oldtailish, pos); // relaxed (in theory)
```



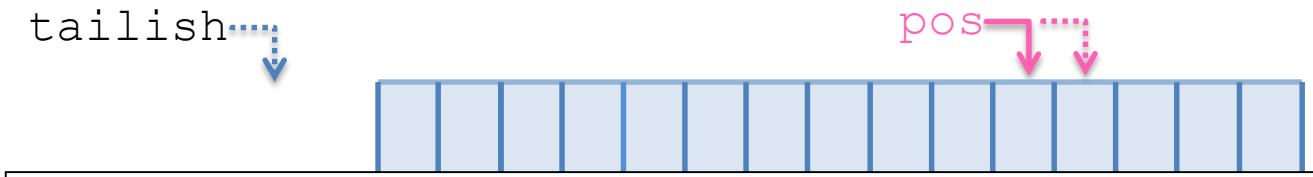

```
void push(int val) {
  geni pos = tailish; // relaxed load
  do {
    pos = find_tail(pos);
  } while (!try_write_value(pos, val));
  update_tailish(tailish, ++pos);
}
```



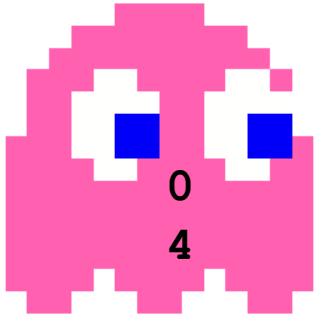

```
tailish.compare_exchange(oldtailish, pos); // relaxed (in theory)
---
geni recent = tailish;
while (recent < pos && !tailish.compare_exchange(recent, pos)) //(recent is updated each loop)
    ;
```



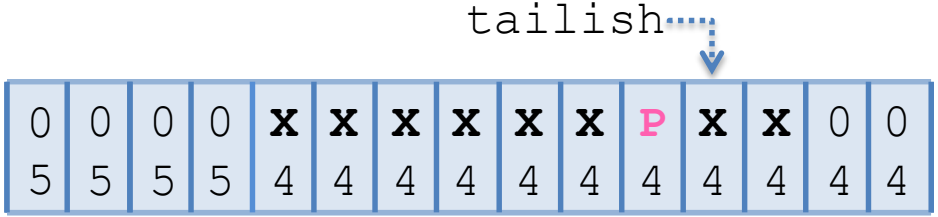

```
void push(int val) {
  geni pos = tailish; // relaxed load
  do {
    pos = find_tail(pos);
  } while (!try_write_value(pos, val));
  update_tailish(tailish, ++pos);
}
```

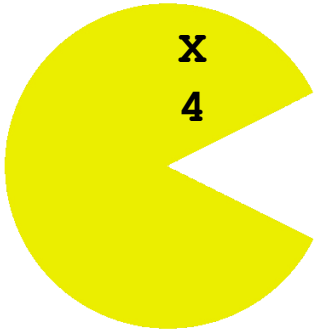



```
tailish.compare_exchange(oldtailish, pos); // relaxed (in theory)
---
geni recent = tailish;
while (recent < pos && !tailish.compare_exchange(recent, pos)) //(recent is updated each loop)
    ;
---
tailish = pos;  // relaxed (from Sebastian Redl)
```

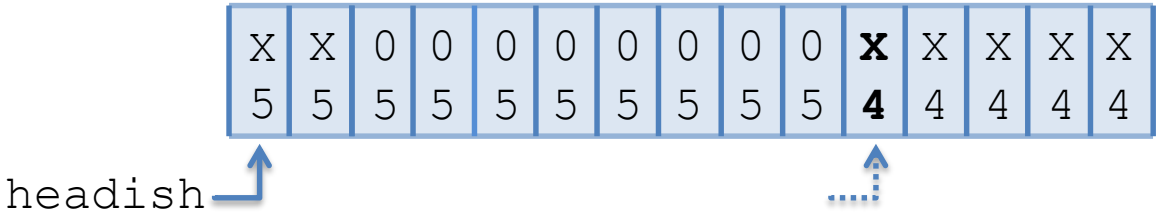



```
void push(int val) {
  geni pos = tailish; // relaxed load
  do {
    pos = find_tail(pos);
  } while (!try_write_value(pos, val));
  tailish = pos+1;  // thanks Sebastian
}
```

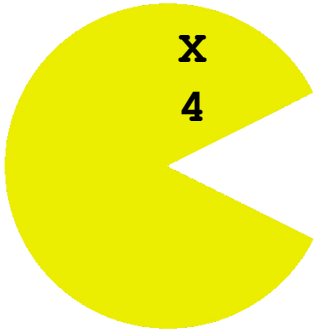




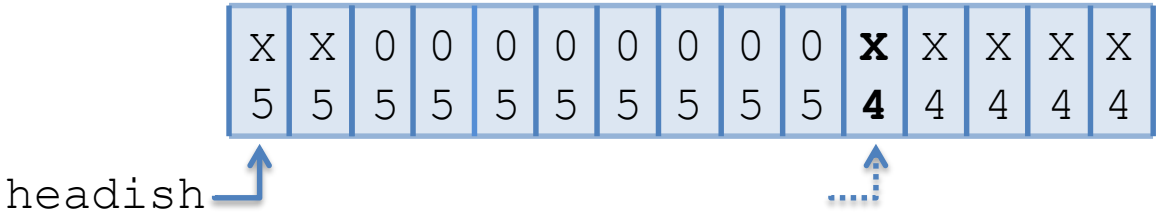

```
int pop() {



}
```



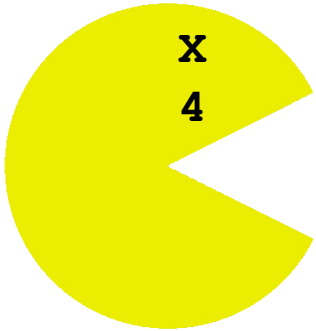
First non-zero data **of correct generation**



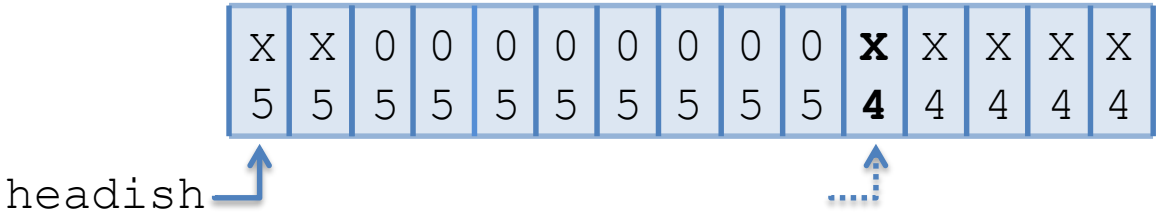

```
void push(int val) {
  geni pos = tailish; // relaxed load
  do {
    pos = find_tail(pos);
  } while (!try_write_value(pos, val));
  tailish = pos+1;  // thanks Sebastian
}
```



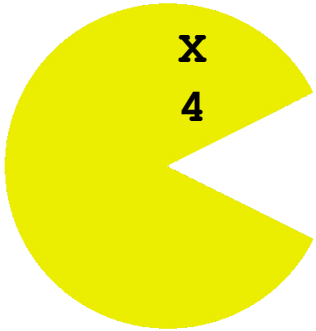
First non-zero data **of correct generation**



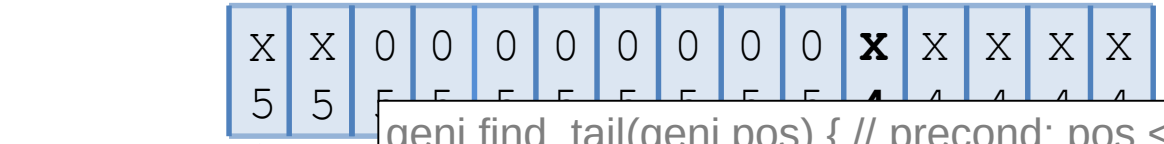

```
int pop() {
  entry ent;
  geni pos = headish; // relaxed load
  do {
    pos = find_head(pos, ent);
  } while (!try_remove_value(pos, ent));
  headish = pos+1;  // thanks Sebastian
  return ent.val;
}
```



First non-zero data **of correct generation**

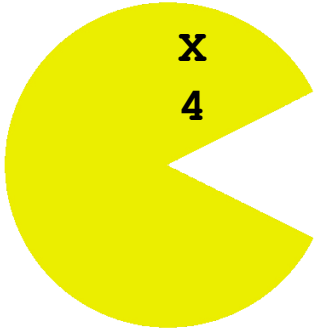



```
int pop() {
  entry ent;
  geni pos = headish; // relaxed load
  do {
    pos = find_head(pos, ent);
  } while (!try_remove_value(pos, ent));
  headish = pos+1;  // thanks Sebastian
  return ent.val;
}
```

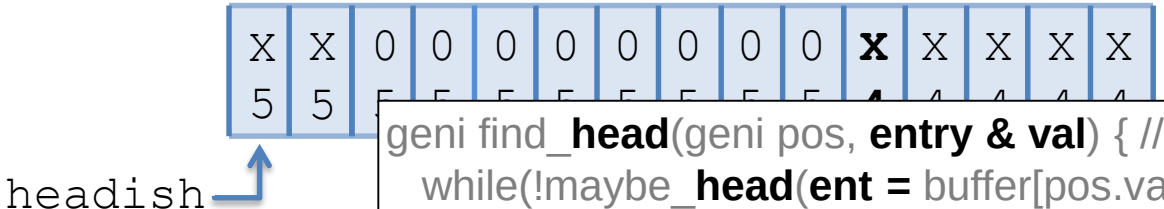


First non-zer

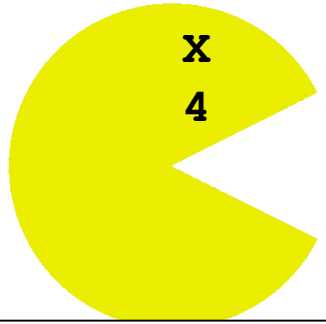
```
geni find_tail(geni pos) { // precond: pos <= tail
  while(!maybe_tail(buffer[pos.val].load(relaxed), pos.gen))
      pos++;
  return pos;
}
```




```
int pop() {
  entry ent;
  geni pos = headish; // relaxed load
  do {
    pos = find_head(pos, ent);
  } while (!try_remove_value(pos, ent));
  headish = pos+1;  // thanks Sebastian
  return ent.val;
}
```




```
geni find_head(geni pos, entry & val) { // precond: pos <= head
  while(!maybe_head(ent = buffer[pos.val].load(relaxed), pos.gen))
      pos++;
  return pos;
}
```




```
int pop() {
  entry ent;
  geni pos = headish; // relaxed load
  do {
    pos = find_head(pos, ent);
  } while (!try_remove_value(pos, ent));
  adish = pos+1;  // thanks Sebastian
  turn ent.val;
```

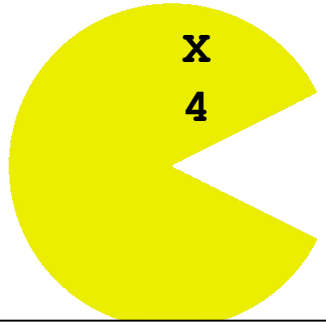
```
bool maybe_tail(entry e, int gen) {
  return e.data == 0 && e.gen == gen
       || e.data  != 0 && e.gen < gen;
}
```

X	X	0	0	0	0	0	0	0	0	**X**	X	X	X	X
5	5	5	5	5	5	5	5	5	5	**4**	4	4	4	4

headish

First non-zer

```
geni find_head(geni pos, entry & val) { // precond: pos <= head
  while(!maybe_head(ent = buffer[pos.val].load(relaxed), pos.gen))
    pos++;
  return pos;
}
```




```
int pop() {
  entry ent;
  geni pos = headish; // relaxed load
  do {
    pos = find_head(pos, ent);
  } while (!try_remove_value(pos, ent));
  adish = pos+1;  // thanks Sebastian
  turn ent.val;
```

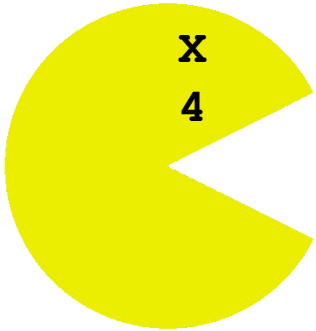
```
bool maybe_head(entry e, int gen) {
  return e.data != 0 && e.gen == gen
       || e.data  == 0 && e.gen < gen;
}
```

X	X	0	0	0	0	0	0	0	0	**X**	X	X	X	X
5	5	5	5	5	5	5	5	5	5	**4**	4	4	4	4

headish

First non-zer

```
geni find_head(geni pos, entry & val) { // precond: pos <= head
  while(!maybe_head(ent = buffer[pos.val].load(relaxed), pos.gen))
    pos++;
  return pos;
}
```



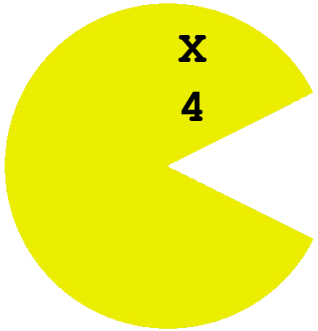

```
int pop() {
  entry ent;
  geni pos = headish; // relaxed load
  do {
    pos = find_head(pos, ent);
  } while (!try_remove_value(pos, ent));
  headish = pos+1;  // thanks Sebastian
  return ent.val;
}
```

X	X	0	0	0	0	0	0	0	0	**X**	X	X	X	X
5	5	5	5	5	5	5	5	5	5	**4**	4	4	4	4

headish

First

```
bool try_remove_value(geni pos, entry old) {

  entry nu{0, pos.gen};
  return buffer[pos].compare_exchange_weak(old, nu, acquire, relaxed);
}
```




```
int pop() {
  entry ent;
  geni pos = headish; // relaxed load
  do {
    pos = find_head(pos, ent);
  } while (!try_remove_value(pos, ent));
  headish = pos+1;  // thanks Sebastian
  return ent.val;
}
```

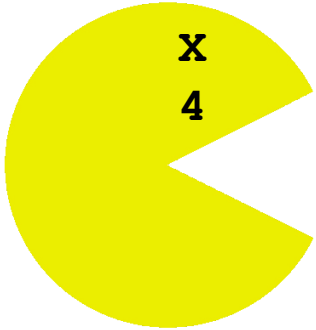
X	X	0	0	0	0	0	0	0	0	**X**	X	X	X	X
5	5	5	5	5	5	5	5	5	5	**4**	4	4	4	4

headish

First

```
bool try_remove_value(geni pos, entry old) {

  entry nu{0, pos.gen};
  return buffer[pos].compare_exchange_weak(old, nu, acquire, relaxed);
}
```

?




```
int pop() {
  entry ent;
  geni pos = headish; // relaxed load
  do {
    pos = find_head(pos, ent);
  } while (!try_remove_value(pos, ent));
  headish = pos+1;  // thanks Sebastian
  return ent.val;
}
```

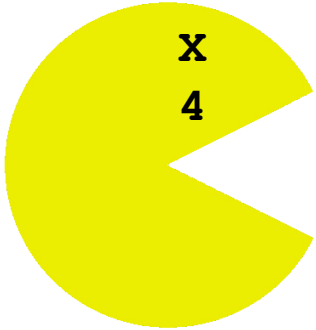
X	X	0	0	0	0	0	0	0	0	**X**	X	X	X	X
5	5	5	5	5	5	5	5	5	5	**4**	4	4	4	4

headish

First

```
bool try_remove_value(geni pos, entry old) {

  entry nu{0, pos.gen};
  return buffer[pos].compare_exchange_weak(old, nu, acq_rel, relaxed);
}
```

?




```
int pop() {
  entry ent;
  geni pos = headish; // relaxed load
  do {
    pos = find_head(pos, ent);
  } while (!try_remove_value(pos, ent));
  headish = pos+1;  // thanks Sebastian
  return ent.val;
}
```

x	x	0	0	0	0	0	0	0	0	**x**	x	x	x	x
5	5	5	5	5	5	5	5	5	5	**4**	4	4	4	4

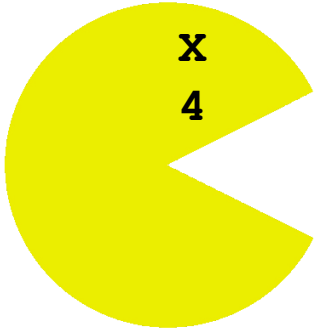
headish

First

Exercise for the reader

```
bool try_remove_value(geni pos, entry old) {

  entry nu{0, pos.gen};
  return buffer[pos].compare_exchange_weak(old, nu, acq_rel, relaxed);
}
```

?




```
int pop() {
  entry ent;
  geni pos = headish; // relaxed lo
  do {
    pos = find_head(pos, ent);
  } while (!try_remove_value(pos,
  headish = pos+1;  // thanks Sebastian
  return ent.val;
}
```

Exercise for the reader:

Don't loop on **EMPTY**. Either:

- have **pop**() return **0**
- *wait* for non **empty**.

X	X	0	0	0	0	0	0	0	0	**X**	X	X	X	X
5	5	5	5	5	5	5	5	5	5	**4**	4	4	4	4

headish

First

Exercise for the reader

```
bool try_remove_value(geni pos, entry old) {

  entry nu{0, pos.gen};
  return buffer[pos].compare_exchange_weak(old, nu, acq_rel, relaxed);
}
```

?

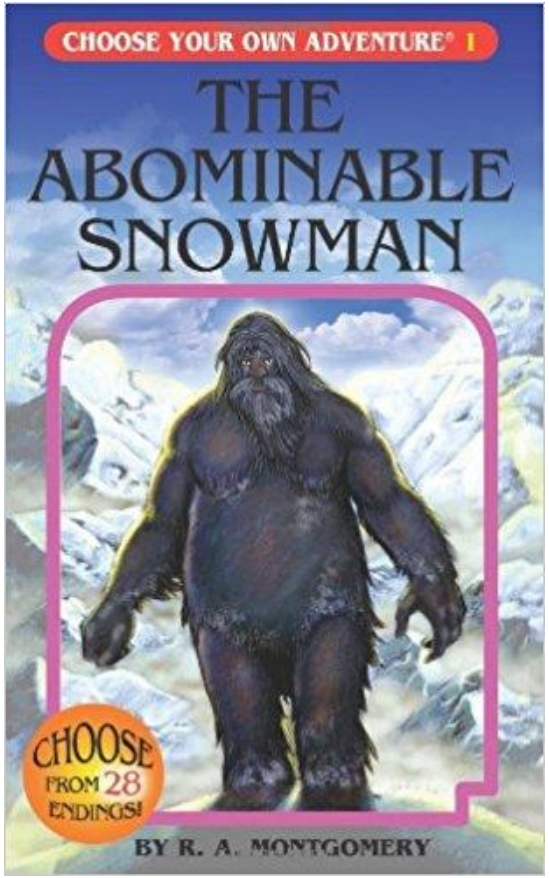
```
void push(int val) {

   geni pos = tailish; // relaxed load
   do {
      pos = find_tail(pos);
      if (FULL) { … }
   } while (!try_write_value(pos, val));
   tailish = pos+1;  // thanks Sebastian

}
geni find_tail(geni pos) { // precond: pos <= tail
  while(!maybe_tail(buff[pos.val].load(relaxed), pos.gen))
      pos++;
  return pos;
}
bool maybe_tail(entry e, int gen) {
   return e.data == 0 && e.gen == gen
         || e.data  != 0 && e.gen < gen;
}
bool try_write_value(geni pos, int val) {
   entry old{0, pos.gen};
   entry nu{val, pos.gen};
   return buffer[pos].c_e_weak(old, nu, release, relaxed);
}
```

```
int pop() {
   entry ent;
   geni pos = headish; // relaxed load
   do {
      pos = find_head(pos, ent);
      if (FULL) { … }
   } while (!try_remove_value(pos, ent));
   headish = pos+1;  // thanks Sebastian
   return ent.val;
}
geni find_head(geni pos, entry & val) { //precond: pos <= head
  while(!maybe_head(ent = buff[pos.val].load(relaxed),pos.gen))
      pos++;
  return pos;
}
bool maybe_head(entry e, int gen) {
   return e.data != 0 && e.gen == gen
         || e.data  == 0 && e.gen < gen;
}
bool try_remove_value(geni pos, entry old) {

   entry nu{0, pos.gen};
   return buffer[pos].c_e_weak(old, nu, acq_rel, relaxed);
}
```

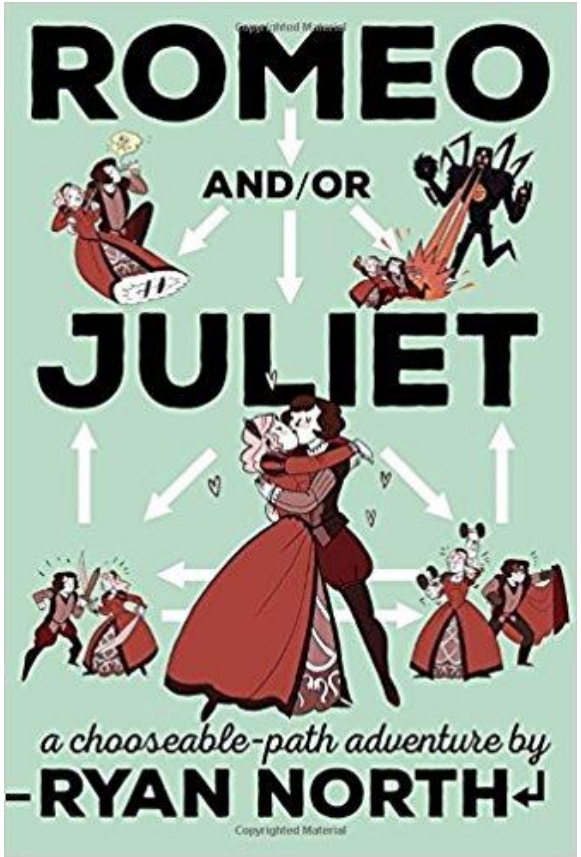
```
void push(int val) {
  geni pos = tailish; // relaxed load
  do {
    pos = find_tail(pos);
    if (FULL) { … }
  } while (!try_write_value(pos, val));
  tailish = pos+1; // thanks Sebastian

}
geni find_tail(geni pos) { // precond: pos <= tail
  while(!maybe_tail(buff[pos.val].load(relaxed), pos.gen))
    pos++;
  return pos;
}
bool maybe_tail(entry e, int gen) {
  return e.data == 0 && e.gen == gen
        || e.data  != 0 && e.gen < gen;
}
bool try_write_value(geni pos, int val) {
  entry old{0, pos.gen};
  entry nu{val, pos.gen};
  return buffer[pos].c_e_weak(old, nu, release, relaxed);
}
```

```
int pop() {
  entry ent;
  geni pos = headish; // relaxed load
  do {
    pos = find_head(pos, ent);
    if (FULL) { … }
  } while (!try_remove_value(pos, ent));
  headish = pos+1;  // thanks Sebastian
  return ent.val;
}
geni find_head(geni pos, entry & val) { //precond: pos <= head
  while(!maybe_head(ent = buff[pos.val].load(relaxed),pos.gen))
    pos++;
  return pos;
}
bool maybe_head(entry e, int gen) {
  return e.data != 0 && e.gen == gen
        || e.data  == 0 && e.gen < gen;
}
bool try_remove_value(geni pos, entry old) {

  entry nu{0, pos.gen};
  return buffer[pos].c_e_weak(old, nu, acq_rel, relaxed);
}
```

Choose Your Own Adventure



Chooseable-path Adventure



Choose Your Own Adventure

- Make head/tail *disappear*
- Store non-ints
- Wait on full/empty (with condvar)
- Grow the queue
- Proofs
- *Something else!*

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	4	4	4	4	4	4	4	4	4	4	4	4	4	4

0	0	0	0	0	0	0	0	0	0	**x**	**x**	**x**	**x**	**x**
5	5	5	5	5	5	5	5	5	5	**4**	4	4	4	4

x	**x**	**x**	**0**	0	0	0	0	0	0	0	0	0	0	0
4	4	4	**4**	4	4	4	4	4	4	4	4	4	4	4

x	**x**	**x**	**0**	0	0	0	0	0	0	**x**	**x**	**x**	**x**	**x**
5	5	5	**5**	5	5	5	5	5	5	**4**	4	4	4	4

0	0	0	0	**x**	**x**	**x**	**x**	**x**	**x**	**0**	0	0	0	0
5	5	5	5	**4**	4	4	4	4	4	**4**	4	4	4	4

x	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**
5	5	5	5	5	5	5	5	5	5	**4**	4	4	4	4

0	0	0	0	0	0	0	0	0	0	**0**	0	0	0	0
5	5	5	5	5	5	5	5	5	5	**4**	4	4	4	4

x	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5

A word about auto

```
auto pos = tailish;
---
geni pos = tailish;
```

A word about auto

```
auto pos = tailish;  // pos is laxtomic<geni>
---
geni pos = tailish;
```

A word about auto

```
auto pos = tailish;  // pos is laxtomic<geni>
---
geni pos = tailish;
---
auto pos = tailish.load();  // pos is geni
```

A word about auto

```
auto pos = tailish;  // pos is laxtomic<geni>
---
geni pos = tailish;
---
auto pos = tailish.load();  // pos is geni

---
Iterator it = v.begin();
```

FCD

FCD

Fear
Certainty
Doubt

Atomicization

```
class Queue {
 atomic<entry> buffer[SIZE];
 laxtomic<geni> headish;
 geni> tailish;
```

“normal” lock free code

```
data.x = 10;
data.y = 20;
data.ready.store(true,release);
```

release:
before means before

x
4

```
if (data.ready.load(acquire))
{
    x = data.x;
    y = data.y;
    ...
}
```

acquire:
after means after

```
int non_atomic_data;
atomic<bool> initialized;


// Thread 1
void init() {
   non_atomic_data = 17;
   initialized.store(true, release);
};
```

```
// User thread
if (initialized.load(acquire)) {
   read(non_atomic_data);
}
```

```
int non_atomic_data;
atomic<bool> initialized;


// Thread 1
void init() {
   non_atomic_data = 17;
   initialized.store(true, release);
};
```

```
// User thread
if (initialized.load(acquire)) {
   read(non_atomic_data);
}
```

```
int non_atomic_data;
atomic<bool> initialized;
atomic<bool> counted; // for accounting

// Thread 1
void init() {
   non_atomic_data = 17;
   initialized.store(true, release);
};

// Thread 2
atomic<int> keep_track;
void accounting() {
   if (initialized && !counted) {//or acquire/relaxed
      keep_track++;
      counted.store(true, relaxed);
   }
}
```

```
// User thread
if (initialized.load(acquire)) {
   read(non_atomic_data);
}
```

```
int non_atomic_data;
atomic<bool> initialized;
atomic<bool> counted; // for accounting

// Thread 1
void init() {
   non_atomic_data = 17;
   initialized.store(true, release);
};

// Thread 2
atomic<int> keep_track;
void accounting() {
   if (initialized && !counted) {//or acquire/relaxed
      keep_track++;
      counted.store(true, relaxed);
   }
}
```

```
int non_atomic_data;
enum State { un, initted, counted };
atomic<State> state; // combined!

// Thread 1
void init() {
   non_atomic_data = 17;
   state.store(initted, release);
};

// Thread 2
atomic<int> keep_track;
void accounting() {
   if (state == initted) {
      keep_track++;
      state.store(counted, relaxed);
   }
}
```

```
int non_atomic_data;
enum State { un, initted, counted };
atomic<State> state; // combined!

// Thread 1
void init() {
   non_atomic_data = 17;
   state.store(initted, release);
};

// Thread 2
atomic<int> keep_track;
void accounting() {
   if (state == initted) {
      keep_track++;
      state.store(counted, relaxed);
   }
}
```

```
int non_atomic_data;
enum State { un, initted, counted };
atomic<State> state; // combined!

// Thread 1
void init() {
   non_atomic_data = 17;
   state.store(initted, release);
};

// Thread 2
atomic<int> keep_track;
void accounting() {
   if (state == initted) {
      keep_track++;
      state.store(counted, relaxed);
   }
}
```

```
// User thread…?
if (initialized.load(acquire)) {
   read(non_atomic_data);
}
```

```
int non_atomic_data;
enum State { un, initted, counted };
atomic<State> state; // combined!

// Thread 1
void init() {
   non_atomic_data = 17;
   state.store(initted, release);
};

// Thread 2
atomic<int> keep_track;
void accounting() {
   if (state == initted) {
      keep_track++;
      state.store(counted, relaxed);
   }
}
```

```
// User thread
if (state.load(acquire) >= initted) {
   read(non_atomic_data);
}
```

```
int non_atomic_data;
enum State { un, initted =0x1, counted =0x3 };
atomic<State> state; // combined!

// Thread 1
void init() {
   non_atomic_data = 17;
   state.store(initted, release);
};

// Thread 2
atomic<int> keep_track;
void accounting() {
   if (state == initted) {
      keep_track++;
      state.store(counted, relaxed);
   }
}
```

Equivalently

```
// User thread
if (state.load(acquire) & initted) {
   read(non_atomic_data);
}
```

```
int non_atomic_data;
enum State { un, initted, counted };
atomic<State> state; // combined!

// Thread 1
void init() {
   non_atomic_data = 17;
   state.store(initted, release);
};

// Thread 2
atomic<int> keep_track;
void accounting() {
   if (state == initted) {
      keep_track++;
      state.store(counted, relaxed);
   }
}
```

```
// User thread
if (state.load(acquire) >= initted) {
   read(non_atomic_data);
}
```

```
int non_atomic_data;
enum State { un, initted, counted };
atomic<State> state; // combined!

// Thread 1
void init() {
   non_atomic_data = 17;
   state.store(initted, release);
};

// Thread 2
atomic<int> keep_track;
void accounting() {
   if (state == initted) {
      keep_track++;
      state.store(counted, relaxed);
   }
}
```

```
// User thread
if (state.load(acquire) >= initted) {
   read(non_atomic_data);
}
```

```
int non_atomic_data;
enum State { un, initted, counted };
atomic<State> state; // combined!

// Thread 1
void init() {
   non_atomic_data = 17;
   state.store(initted, release);
};

// Thread 2
atomic<int> keep_track;
void accounting() {
   if (state == initted) {
      keep_track++;
      state.store(counted, relaxed);
   }
}
```

```
// User thread
if (state.load(acquire) >= initted) {
   read(non_atomic_data);
}
```

UNDEFINED BEHAVIOUR

```
int non_atomic_data;
enum State { un, initted, counted };
atomic<State> state; // combined!

// Thread 1
void init() {
   non_atomic_data = 17;
   state.store(initted, release);
};

// Thread 2
atomic<int> keep_track;
void accounting() {
   if (state == initted) {
      keep_track++;
      state.store(counted, relaxed);
   }
}
```

```
// User thread
if (state.load(acquire) >= initted) {
   read(non_atomic_data);
}
```

UNDEFINED BEHAVIOUR

(probably)

```
int non_atomic_data;
enum State { un, initted, counted };
atomic<State> state; // combined!

// Thread 1
void init() {
   non_atomic_data = 17;
   state.store(initted, release);
};

// Thread 2
atomic<int> keep_track;
void accounting() {
   if (state == initted) {
      keep_track++;
      state.store(counted, relaxed);
   }
}
```

```
// User thread
if (state.load(acquire) >= initted) {
   read(non_atomic_data);
}
```

UNDEFINED BEHAVIOUR

(probably)

```
int r1 = *p;
```

This is 2 separate loads, the second depending on the first:

```
int* ptr = read(p);
int r1 = read_at(ptr);
```

“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**.”

The Continuing Saga of the Lock-free Queue
Part 3 of N

Tony Van Eerd C++Now May 2018

CHRISTIE