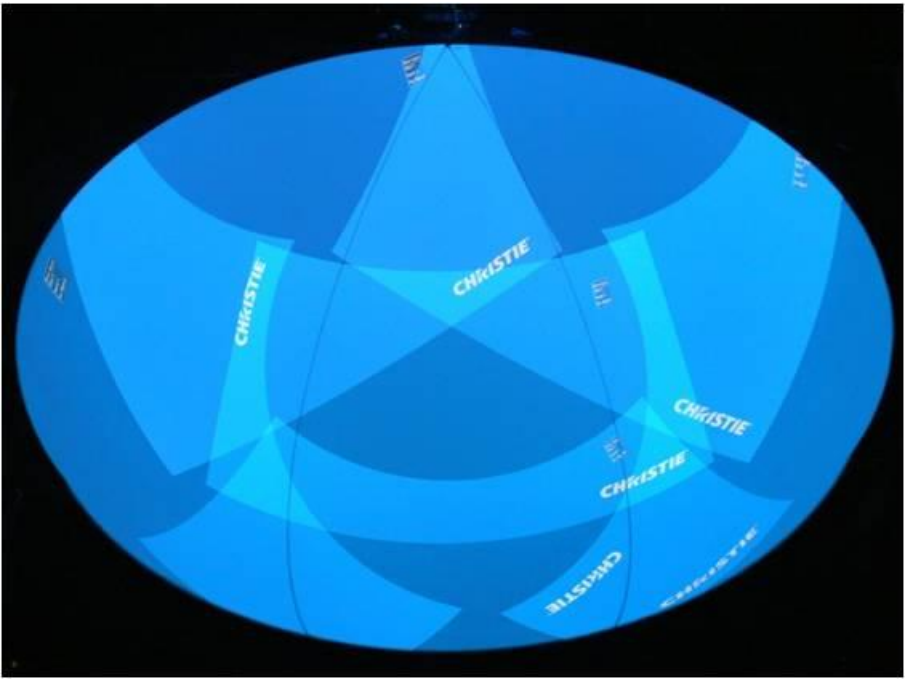
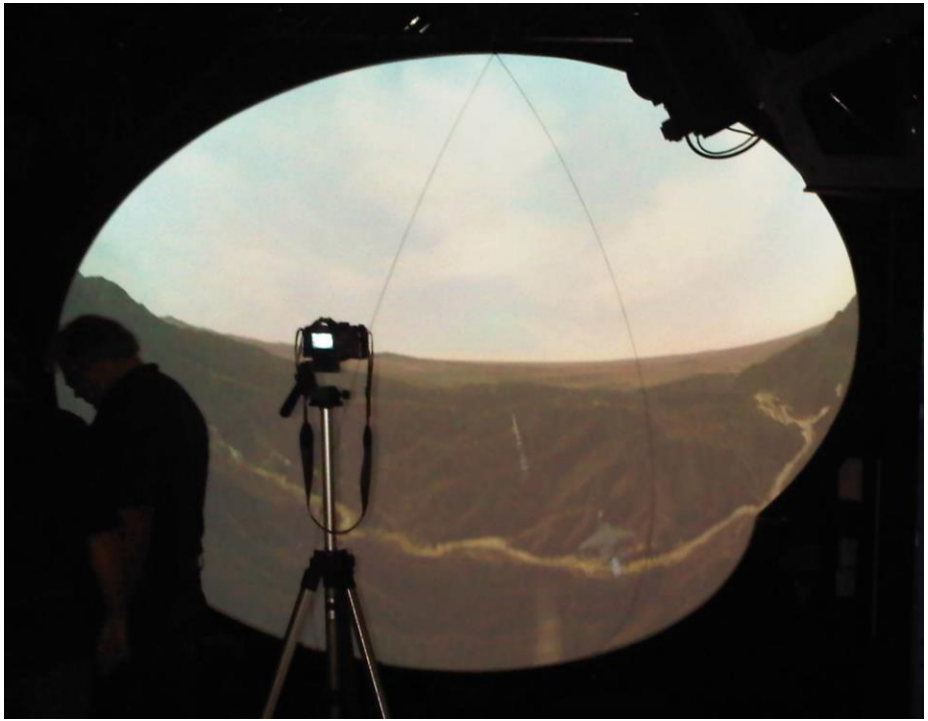
An ~~Overly~~ Not so Complicated Lock-free Queue
Part 2 of N

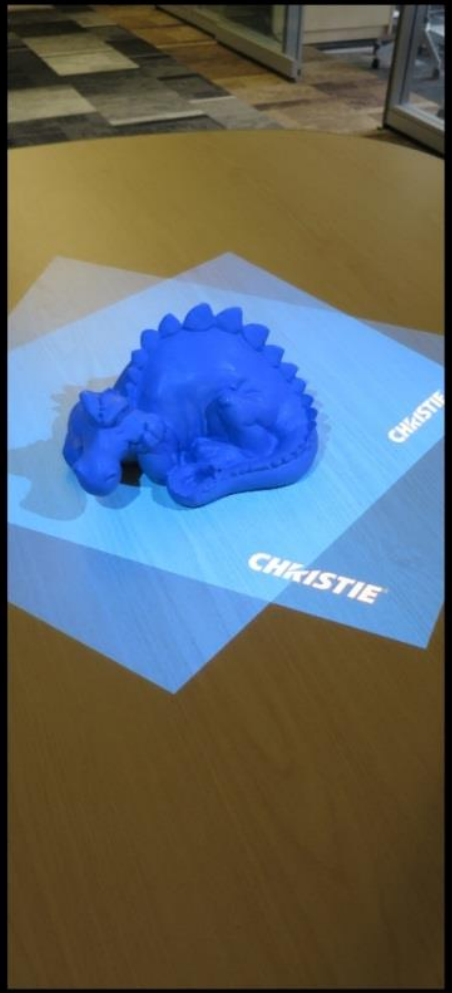
Tony Van Eerd C++Now May 2016

CHRISTIE


CHRISTIE®


CHRISTIE
CHRISTIE
CHRISTIE
CHRISTIE
CHRISTIE




CHRISTIE
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

Guide to Coding

Guide to Coding

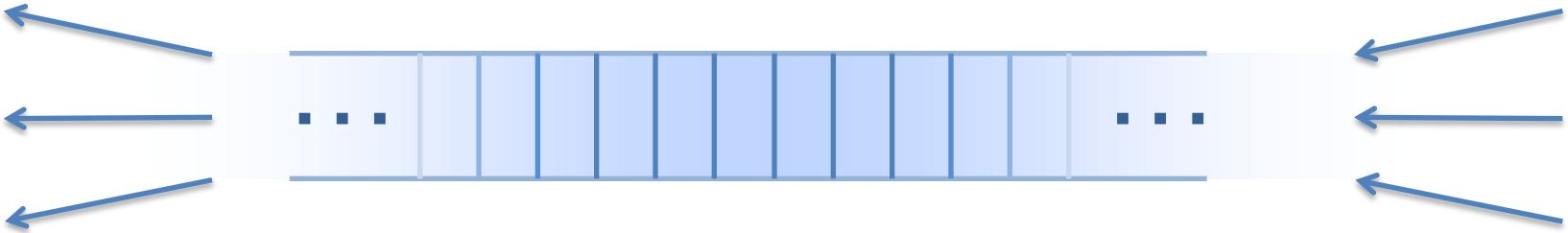
MACROS_ARE_EVIL

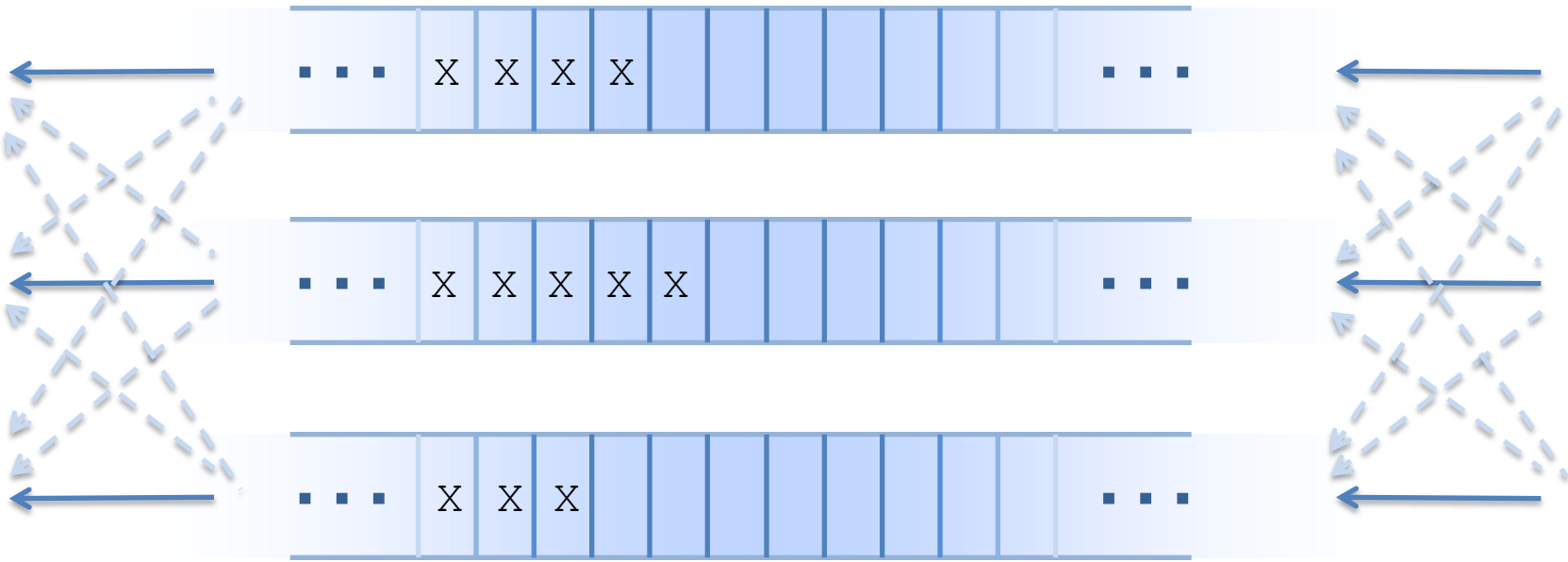
Notes:

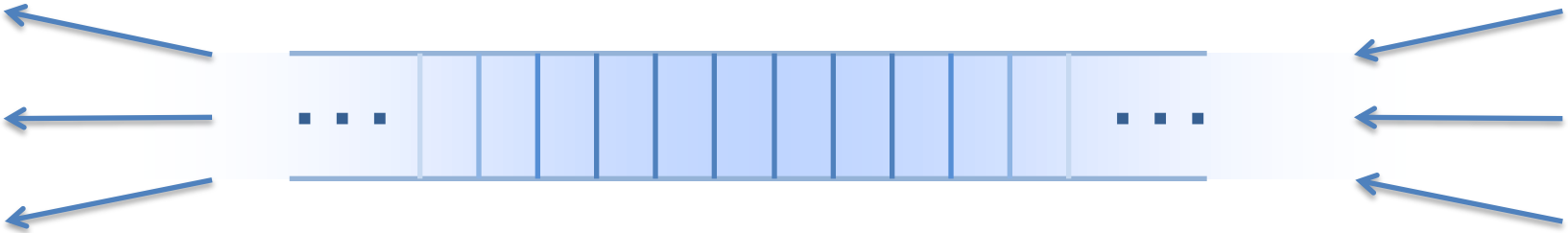
CAS = compare_exchange (_weak or _strong)

Not my coding style/structure

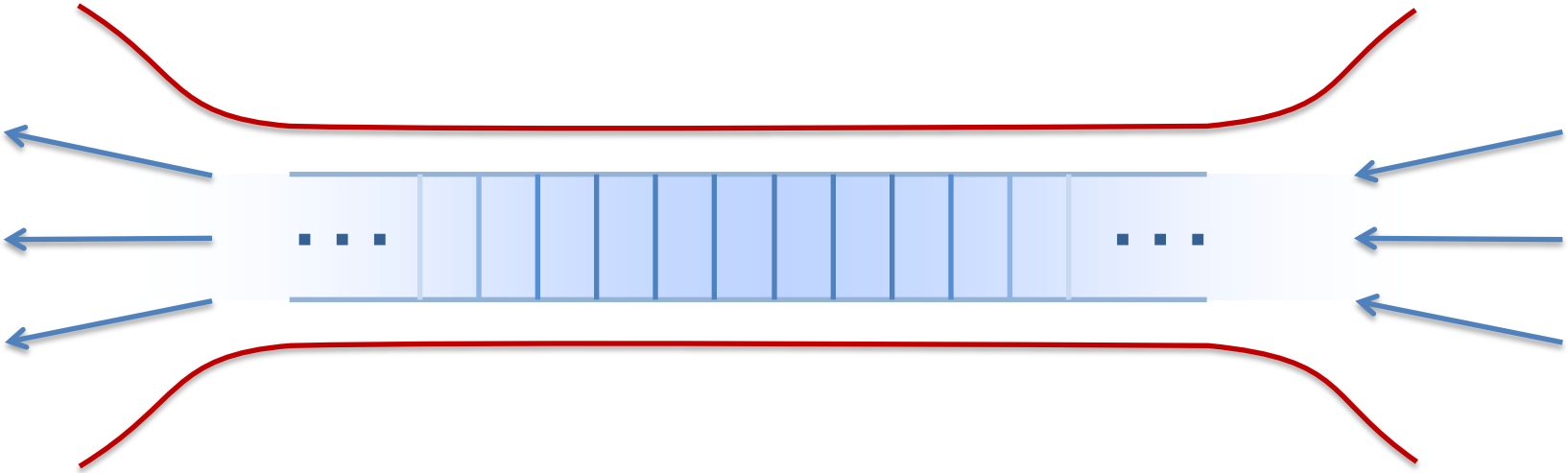
MPMC Queue

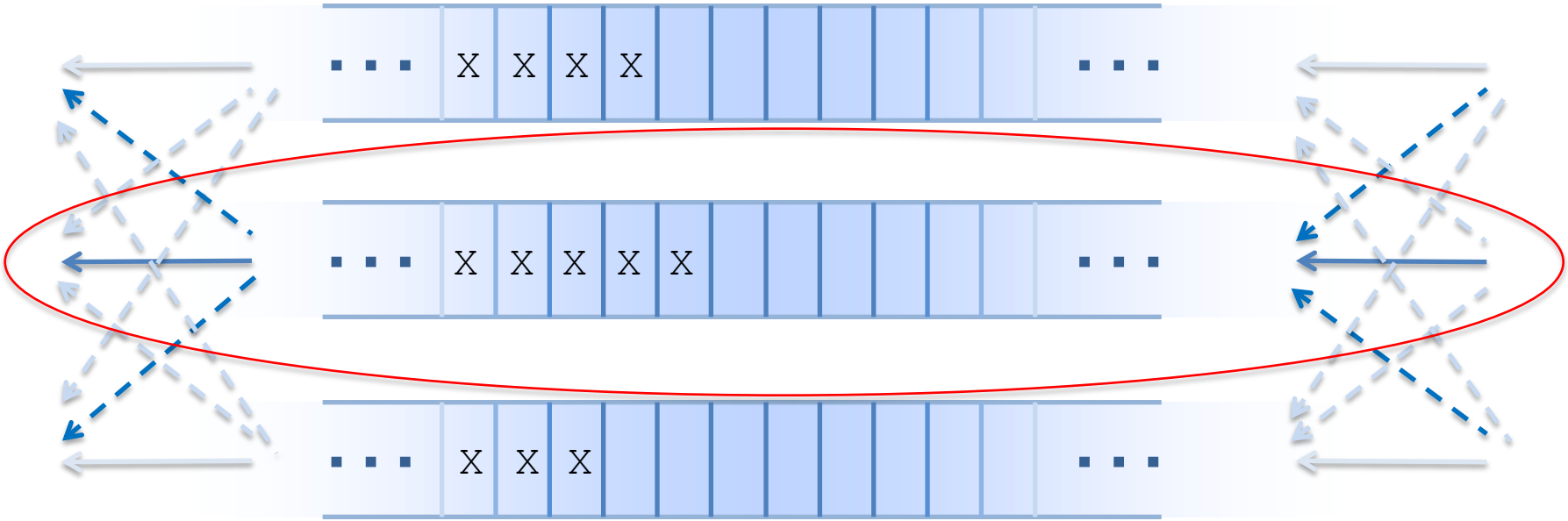


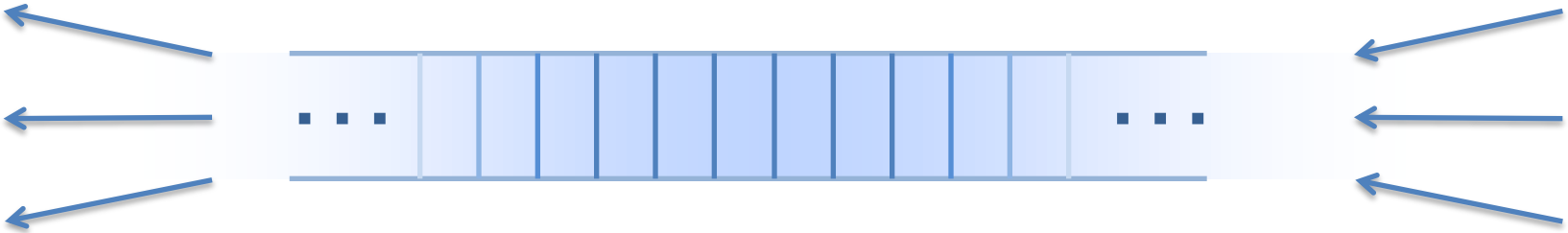

X X X X
X X X X X
X X X



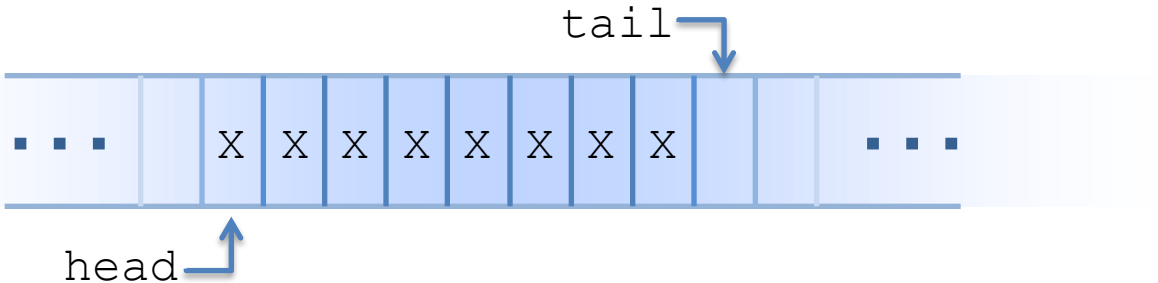
Bottleneck




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

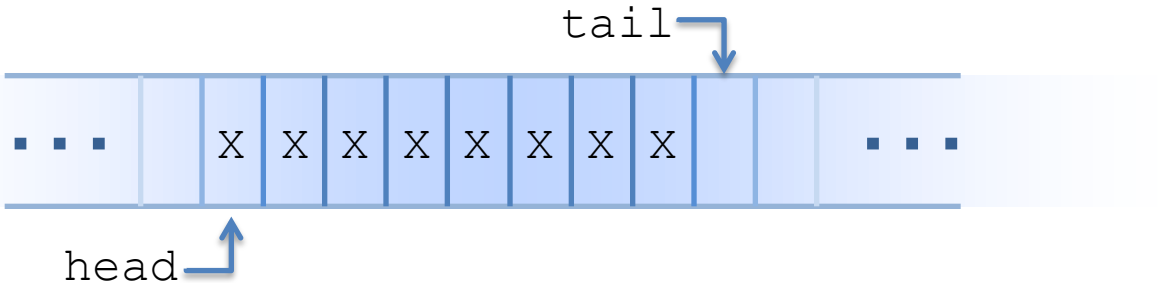


Review



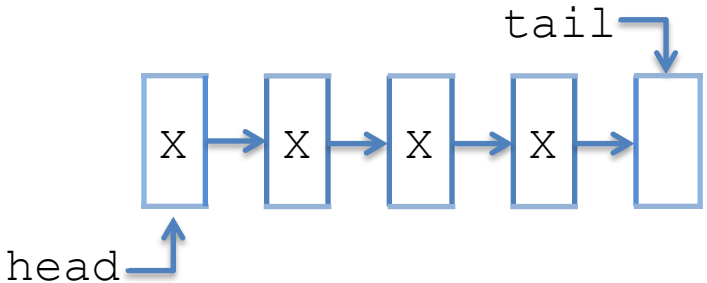
Review

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



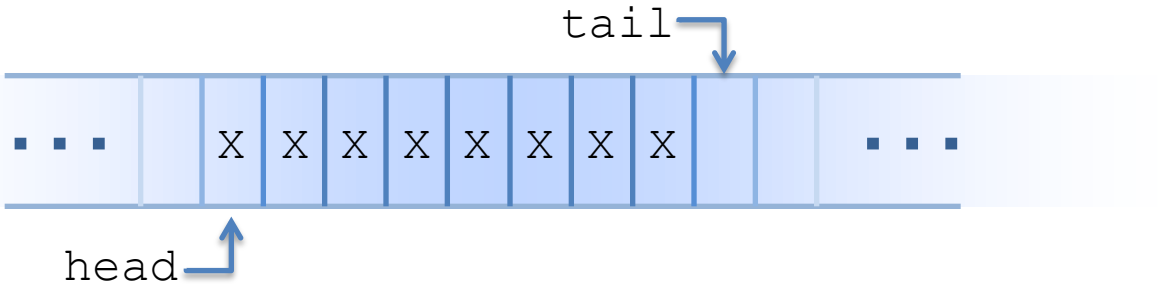
Review

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



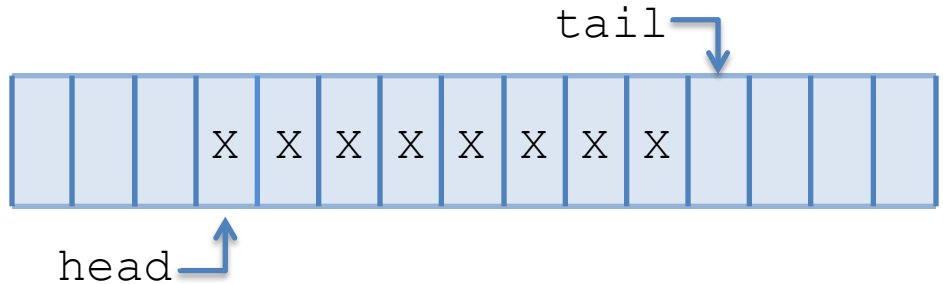
Review

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



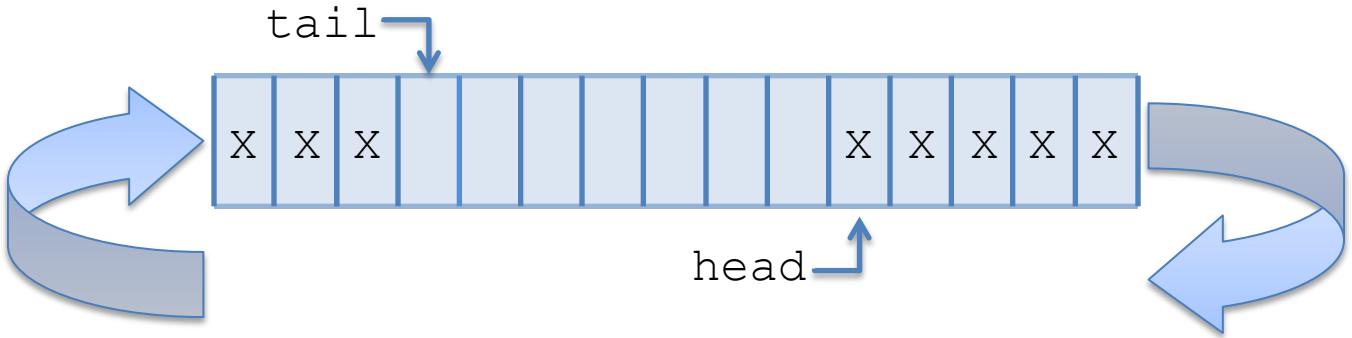
Review

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



Review

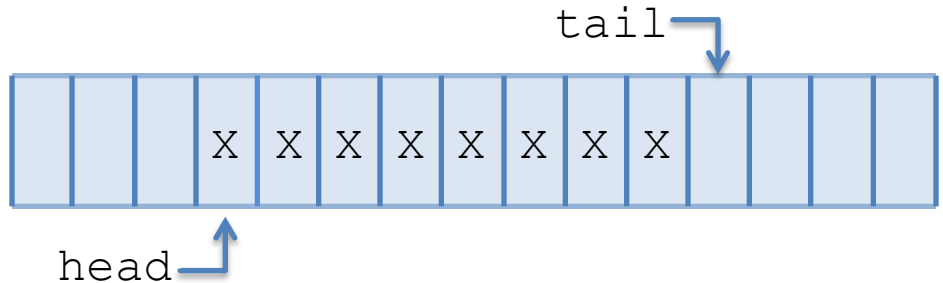
```
class Queue {
 T buffer[SIZE];
 int head;
 int tail;
};
```



Review

Compromise

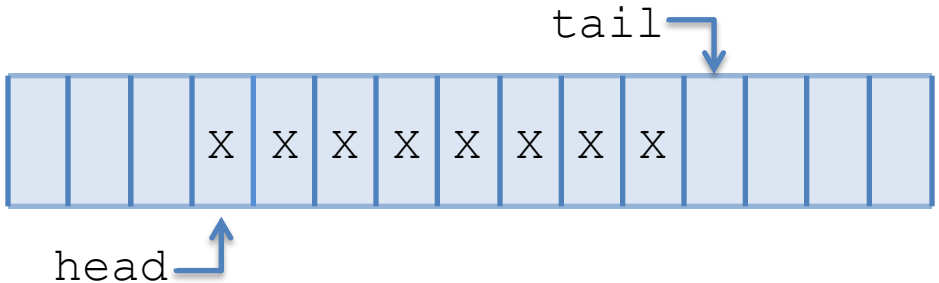
```
class Queue {
 int buffer[SIZE];
 int head;
 int tail;
};
```



Review

Comptomisation

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



Theorizing a New Agenda for Architecture:: An Anthology of ...

https://books.google.com/books?isbn=156898054X

Kate Nesbitt - 1996 - Architecture

At the same time it would have emphasi2ed an indistinct uthan houndaty without **comptomising** the histotic ptofile of the existing tetminus. In the case of the ...

Virgil Thomson: A Reader: Selected Writings, 1924-1984

https://books.google.com/books?isbn=1135360839

Richard Kostelanetz - 2013 - Music

Successful men are often accused of "**comptomising**," too, of **comptomising** with public tasre (which is assumed to be bad tasre and ptofitable to carer to).

About 1,620 results (0.49 seconds)

Did you mean: ***compromisation***

Comptomization and radiation spectra of X-ray sources. Calculation of ...

ntrs.nasa.gov/search.jsp?R=19800019692 ▼ NASA ▼

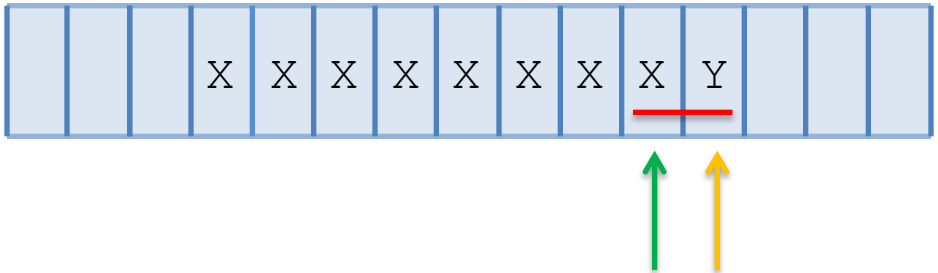
by LA Pozdnyakov - 1980 - Related articles

Jun 1, 1980 - The results of computations of the **Comptomization** of low frequency radiation in weakly relativistic plasma are presented.

Review

false sharing?

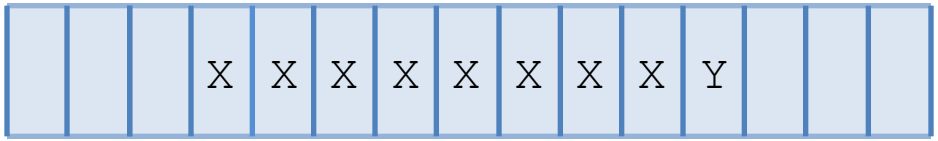
```
class Queue {
 int buffer[SIZE];
 int head;
 int tail;
};
```



Review

false sharing?

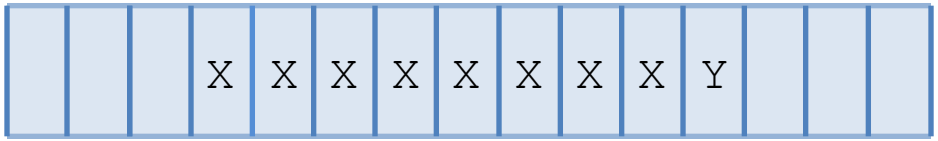
```
class Queue {
 int buffer[SIZE];
 int head;
 int tail;
};
```



Review

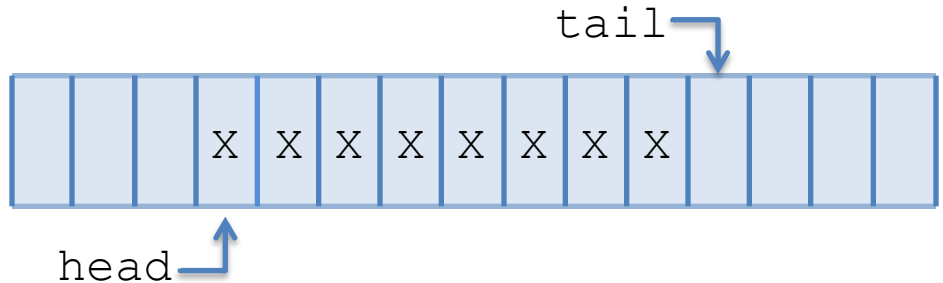
false sharing?

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



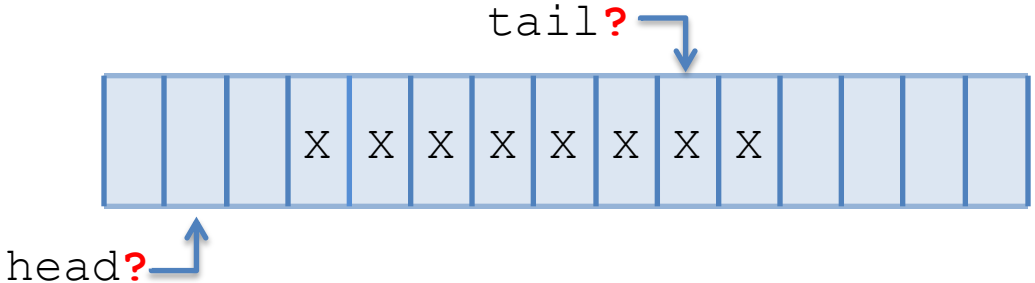
Review

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



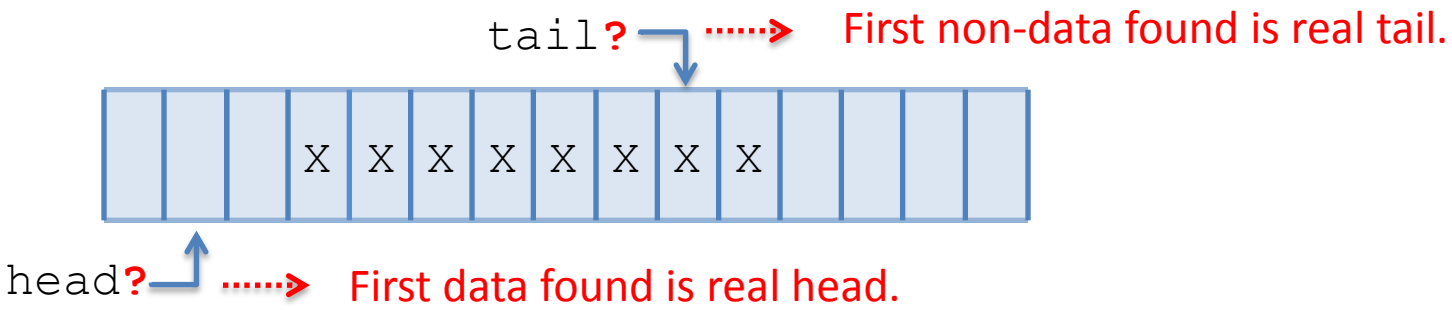
Review

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



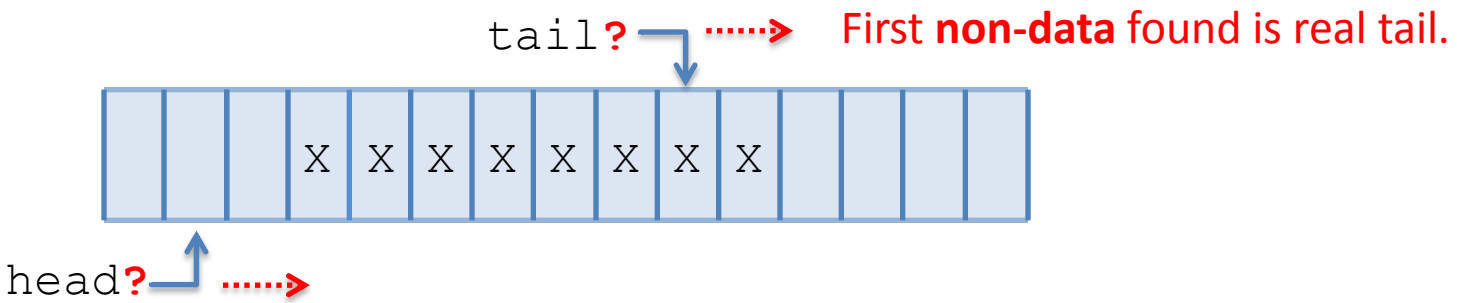
Review

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



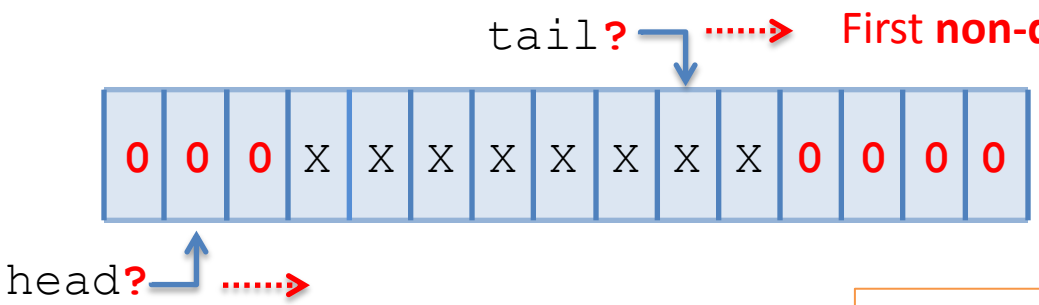
Review

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



Review

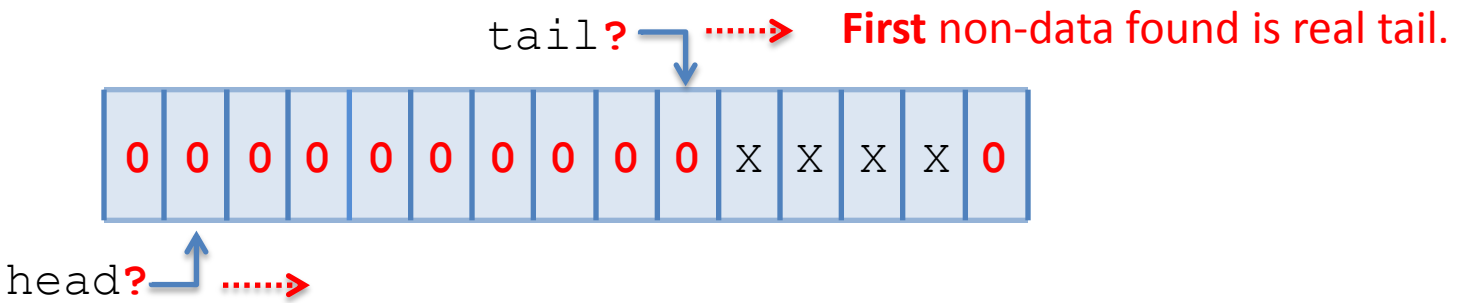
```
class Queue {
 int buffer[SIZE];
 int headish;
 int tailish;
};
```



Comptomisation

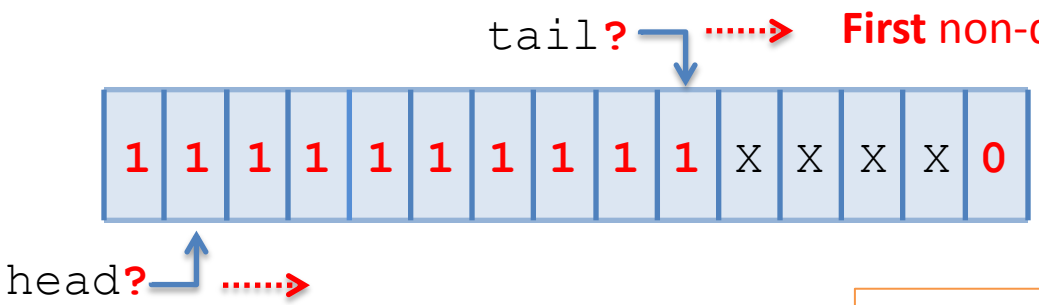
Review

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



Review

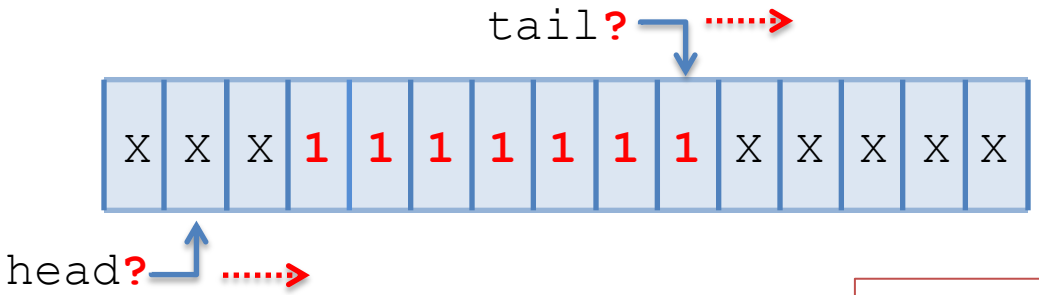
```
class Queue {
 int buffer[SIZE];
 int headish;
 int tailish;
};
```



Comptomisation

Review

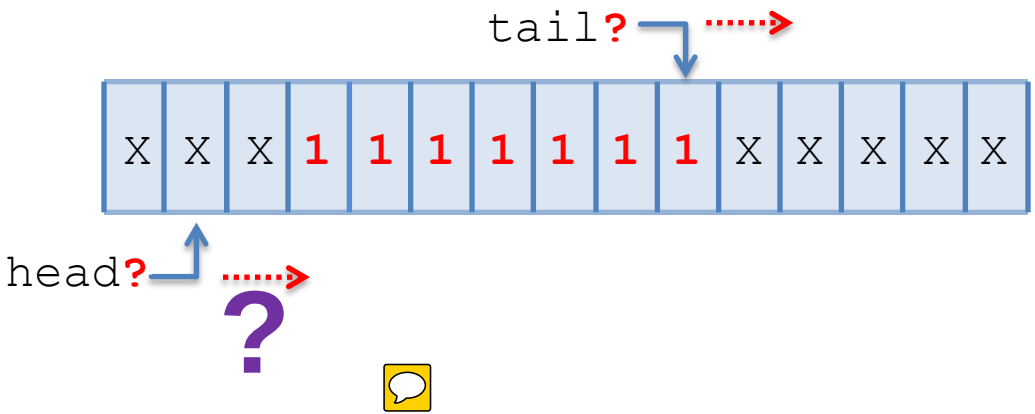
```
class Queue {
 int buffer[SIZE];
 int headish;
 int tailish;
};
```



Circular Buffer

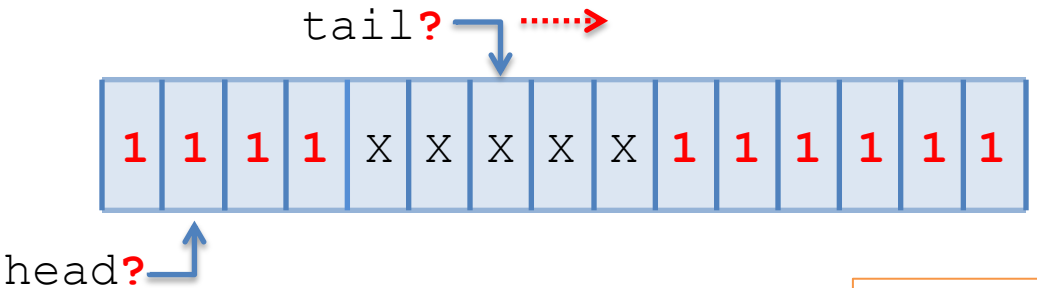
Review

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



Review

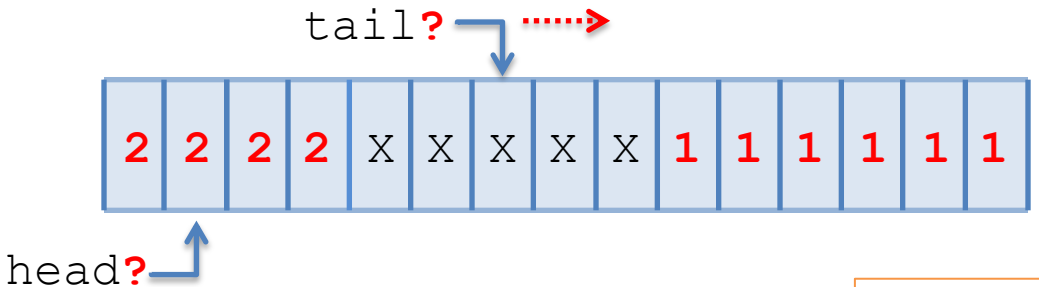
```
class Queue {
 int buffer[SIZE];
 int headish;
 int tailish;
};
```



Comptomisation

Review

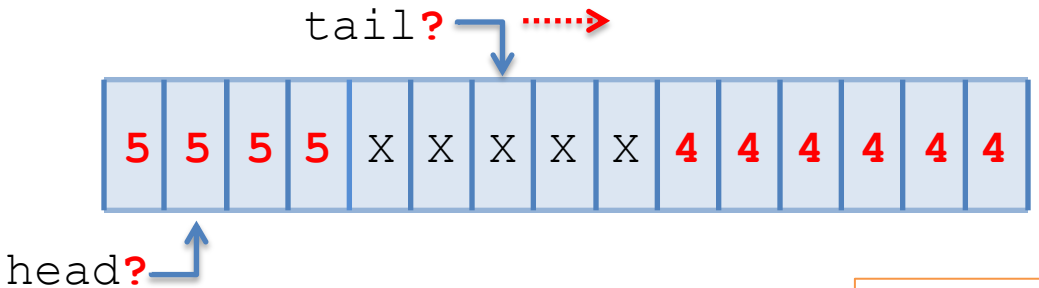
```
class Queue {
 int buffer[SIZE];
 int headish;
 int tailish;
};
```



Comptomisation

Review

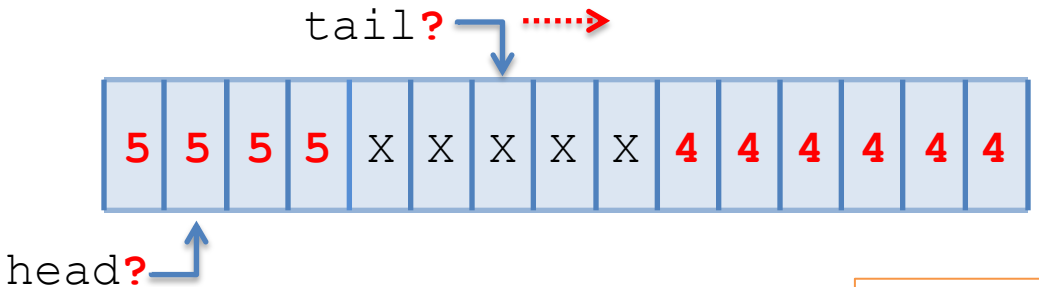
```
class Queue {
 int buffer[SIZE];
 int headish;
 int tailish;
 int generation;
};
```



Comptomisation

Review

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

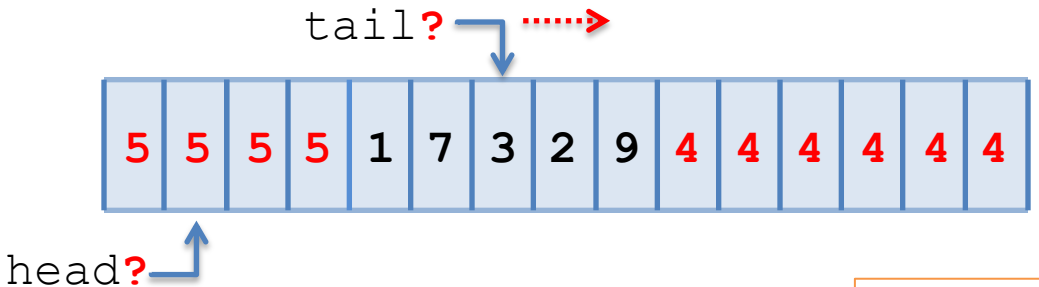


big?

Comptomisation

Review

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

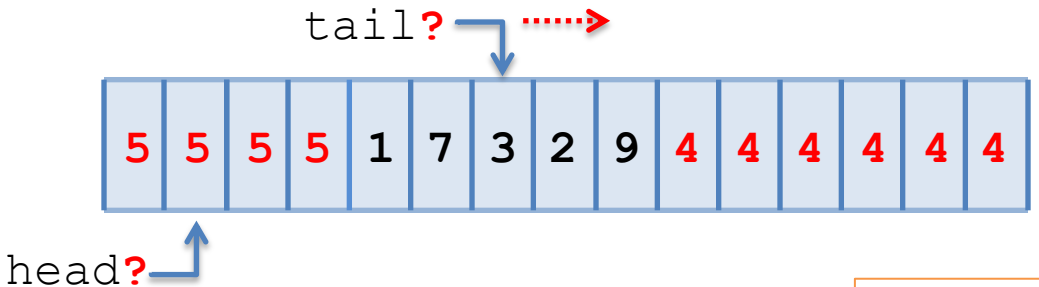


big?

Comptomisation

Review

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



big?

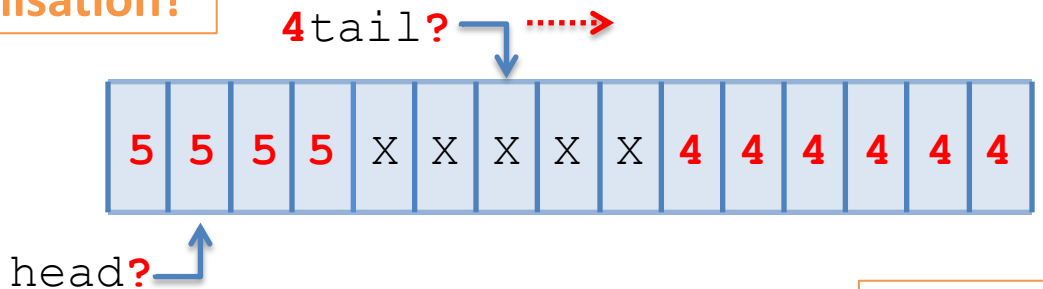
Comptomisation

One bit.
Only positive numbers allowed.

Review

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

Comptomisation?

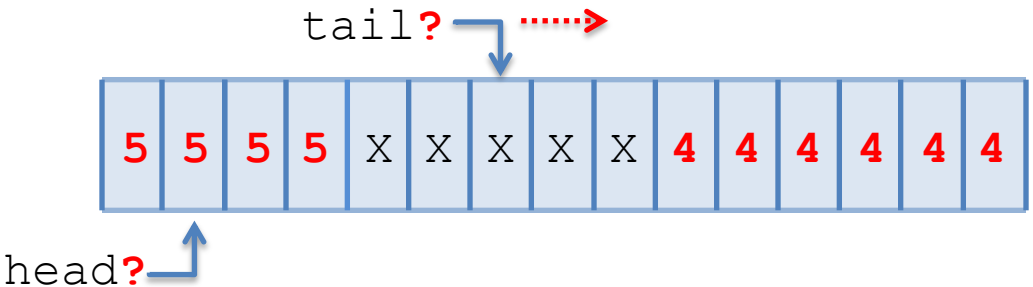


Comptomisation


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

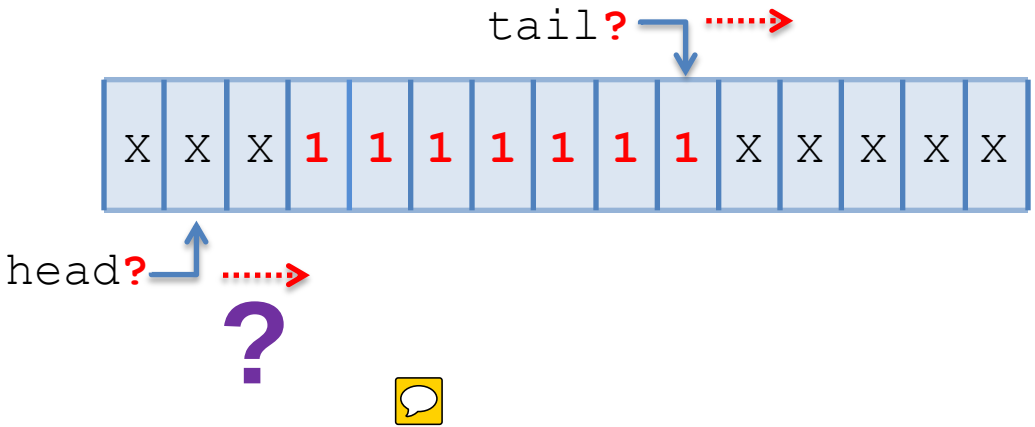
Review

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



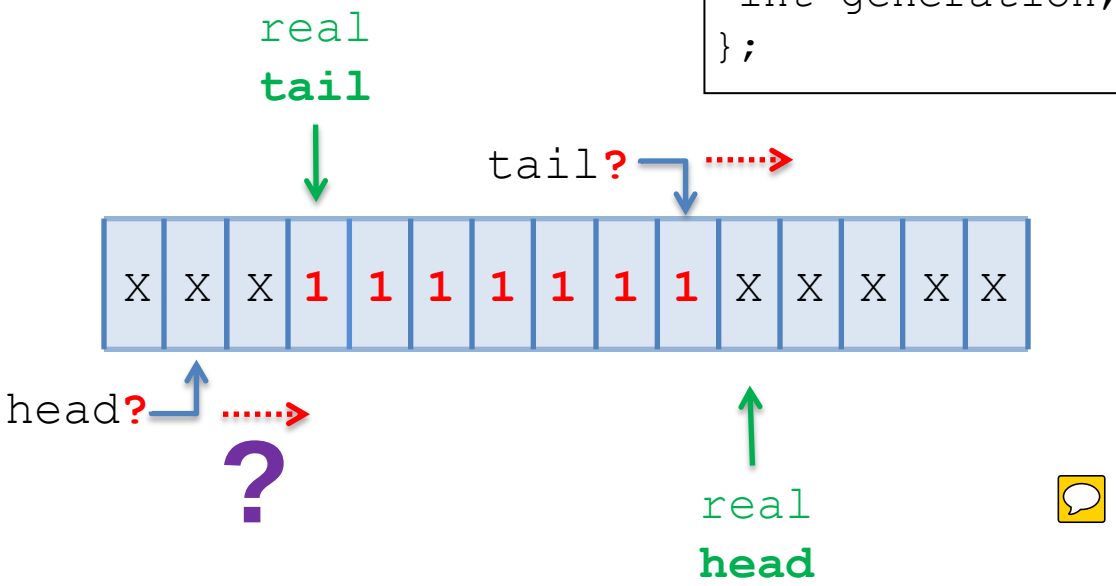
Review

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



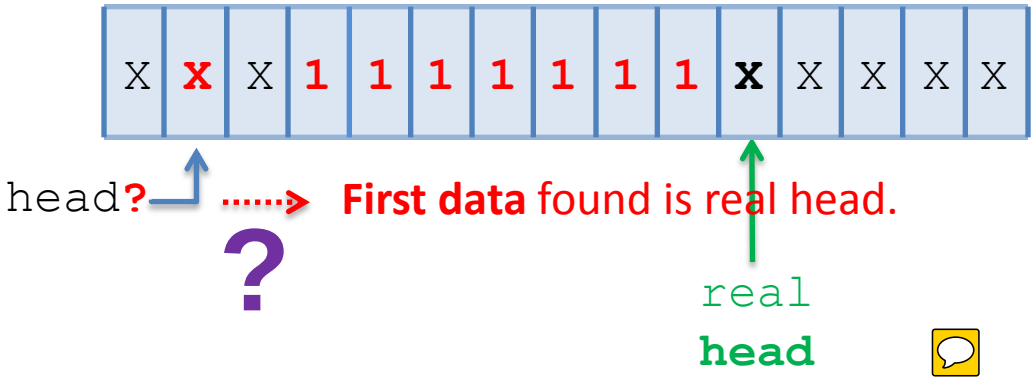
Review

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



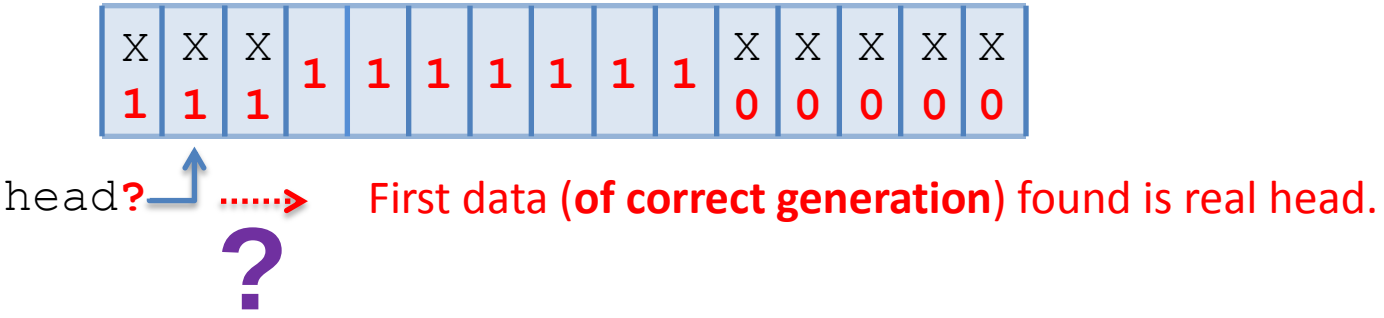
Review

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



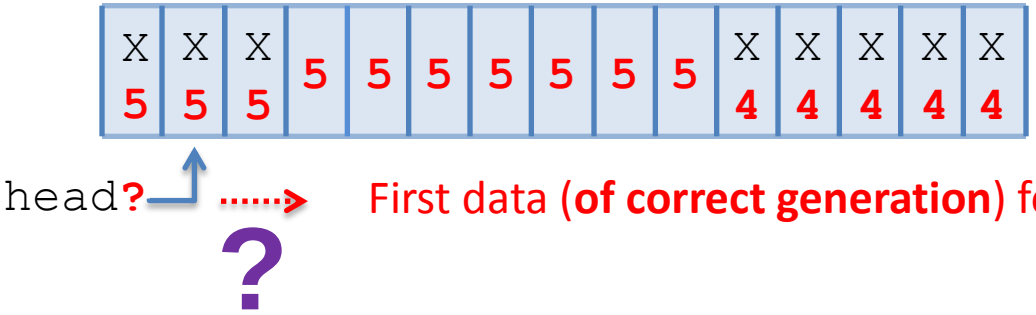
New Path

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



New Path

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

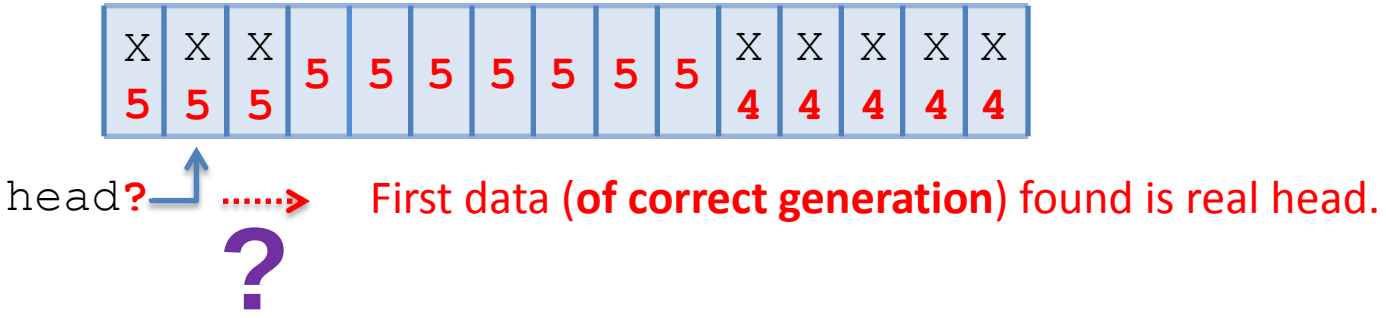


First data (**of correct generation**) found is real head.

New Path

Comptomisation?

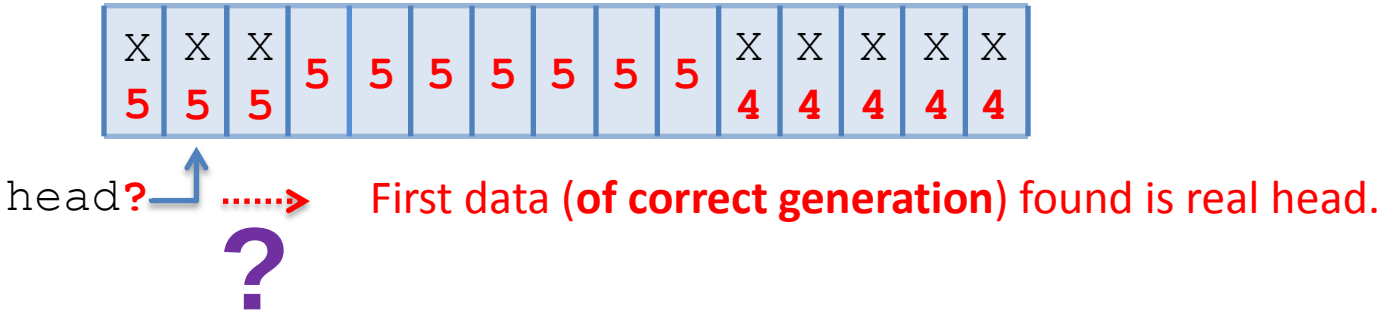
```
class Queue {
 int buffer[SIZE];
 int headish;
 int tailish;
 int generation;
};
```



New Path

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

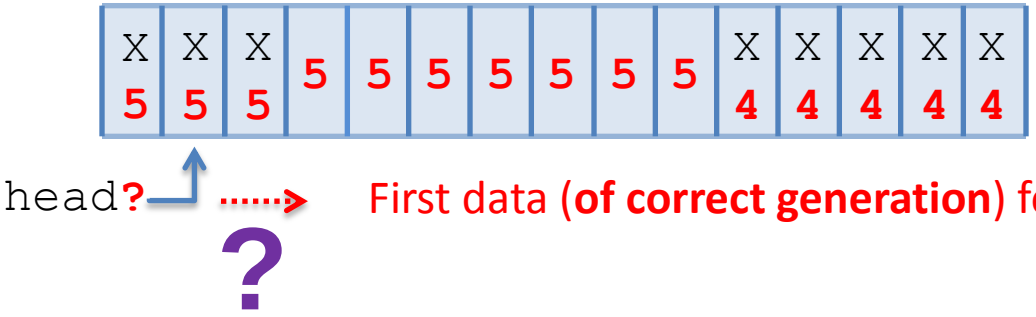
Comptomisation?
No (usually)



New Path

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

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



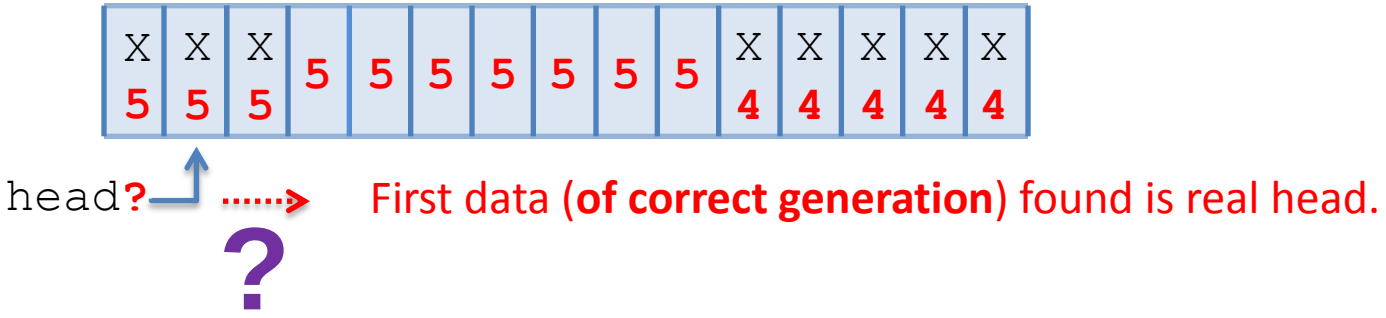
First data (**of correct generation**) found is real head.

New Path

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

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

Comptomisation:

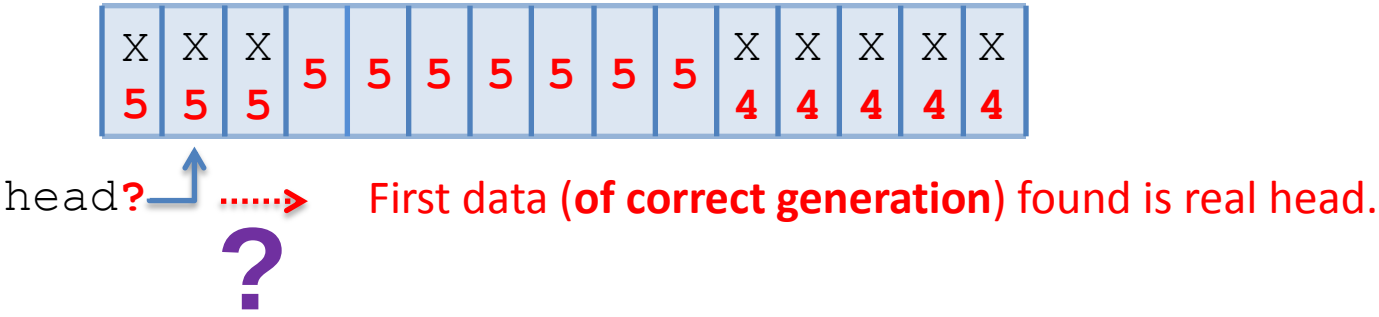


New Path

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

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

Comptomisation: data != 0



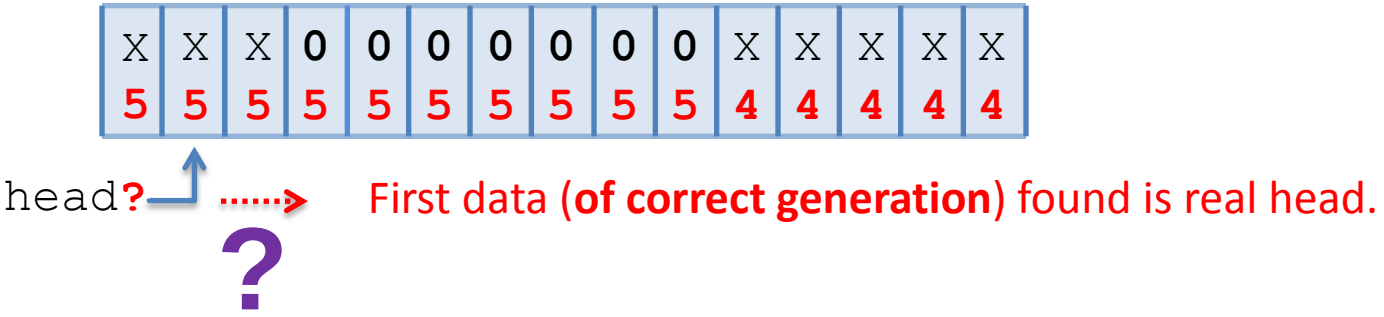
First data (**of correct generation**) found is real head.

New Path

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

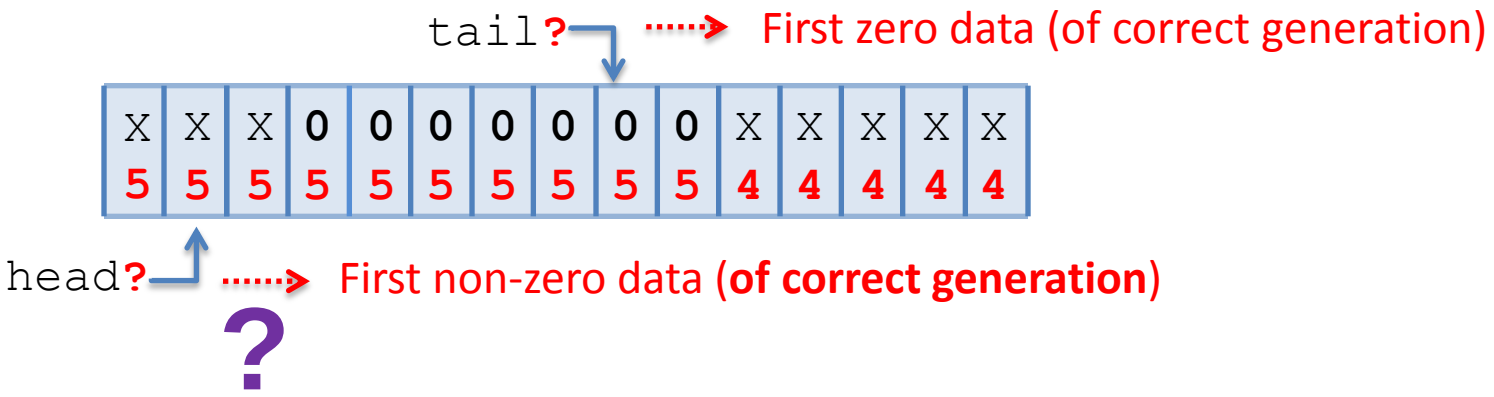
```
class Queue {
 entry buffer[SIZE];
 int headish;
 int tailish;
 int generation;
};
```

Comptomisation: data != 0



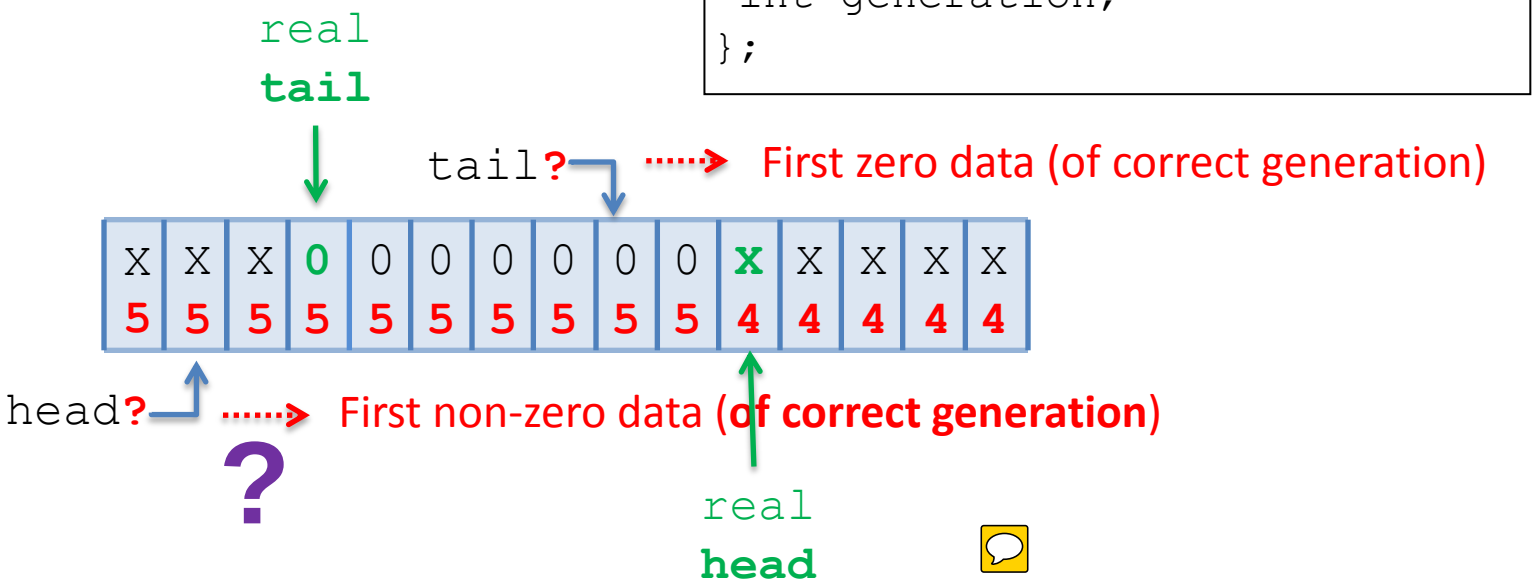
New Path

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



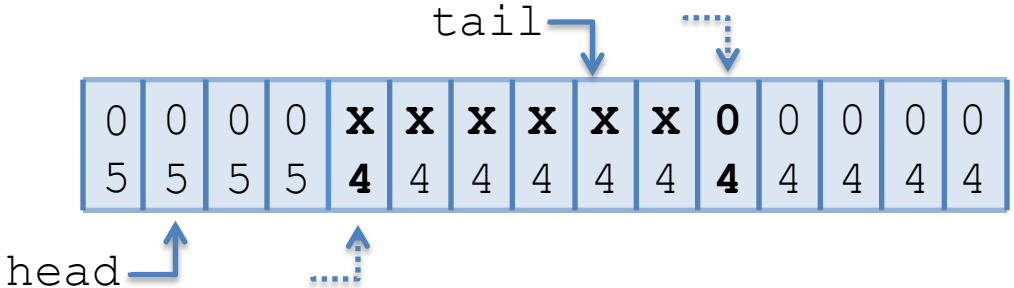
New Path

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



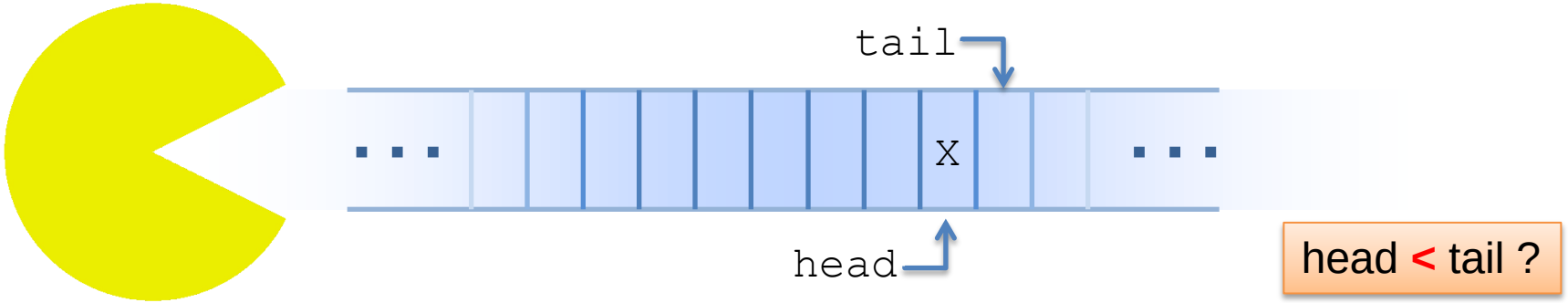
New Path

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



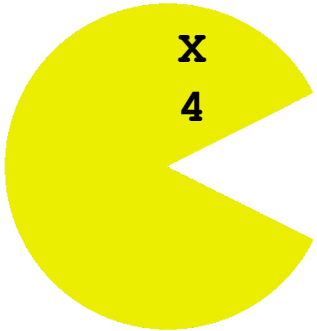
New Path

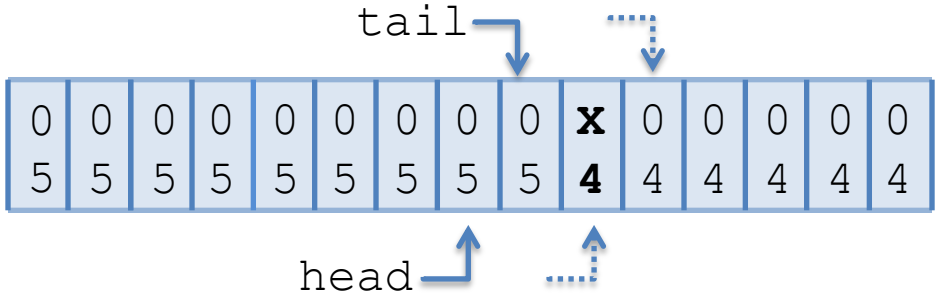
```
class Queue {
 entry buffer[SIZE];
 int headish;
 int tailish;
 int generation;
};
```



New Path

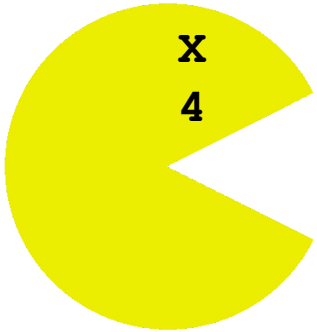
```
class Queue {
 entry buffer[SIZE];
 int headish;
 int tailish;
 int generation;
};
```

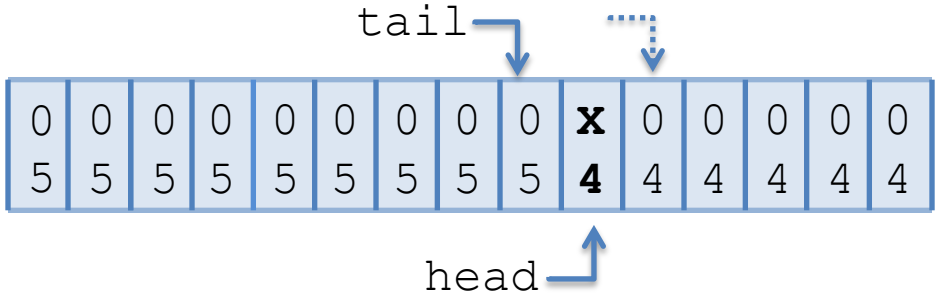




New Path

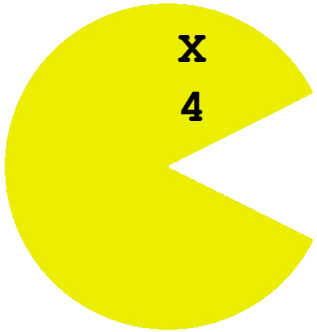
```
class Queue {
 entry buffer[SIZE];
 int headish;
 int tailish;
 int generation;
};
```

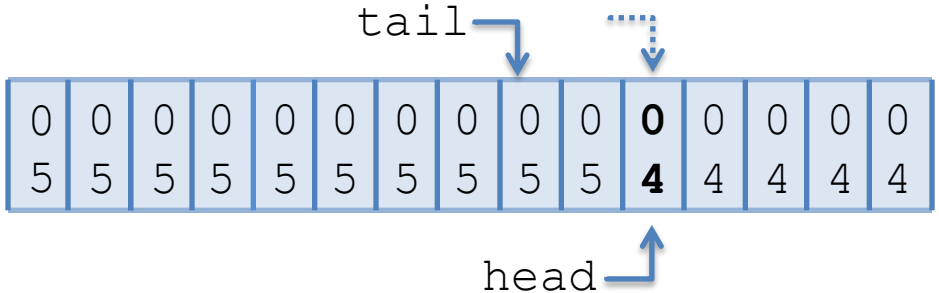




New Path

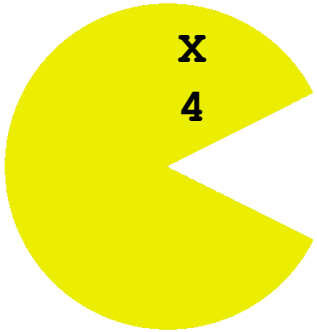
```
class Queue {
 entry buffer[SIZE];
 int headish;
 int tailish;
 int generation;
};
```

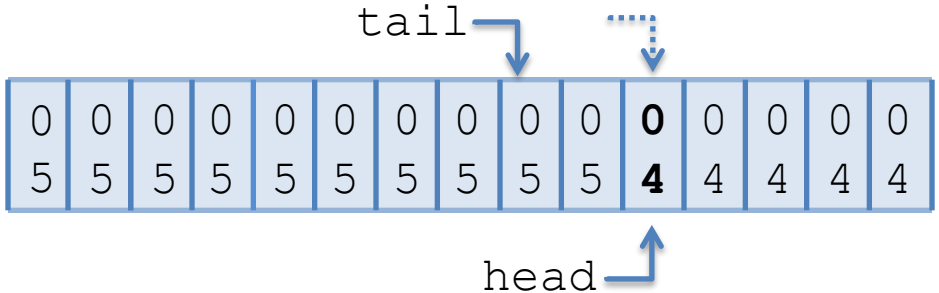




New Path

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

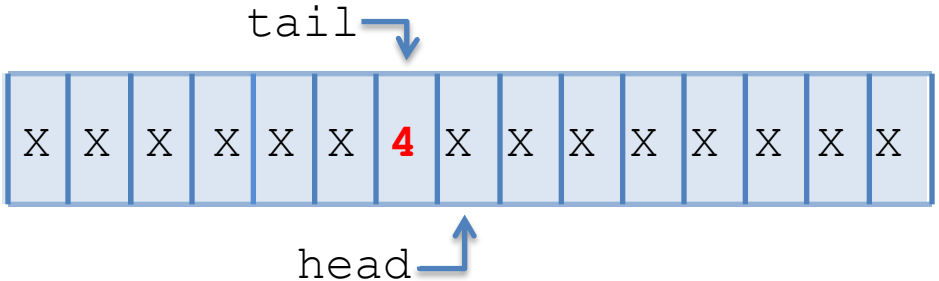




Queue is Empty!

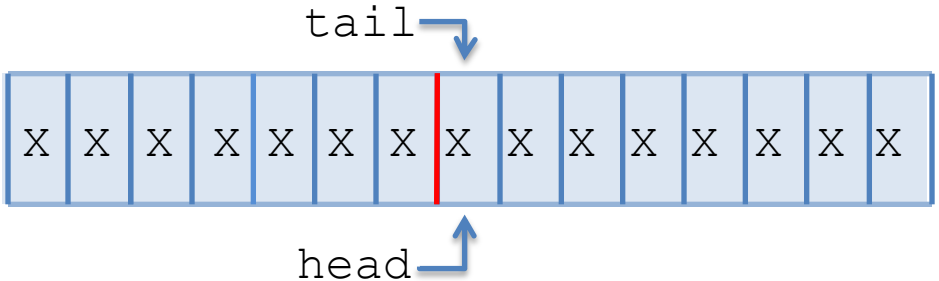
New Path

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



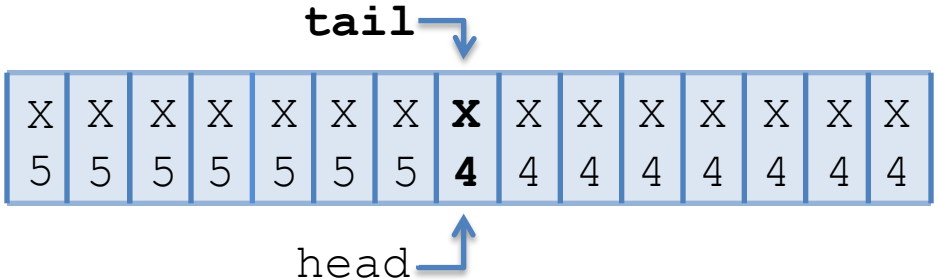
New Path

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



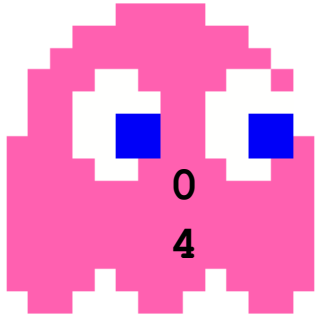
New Path

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

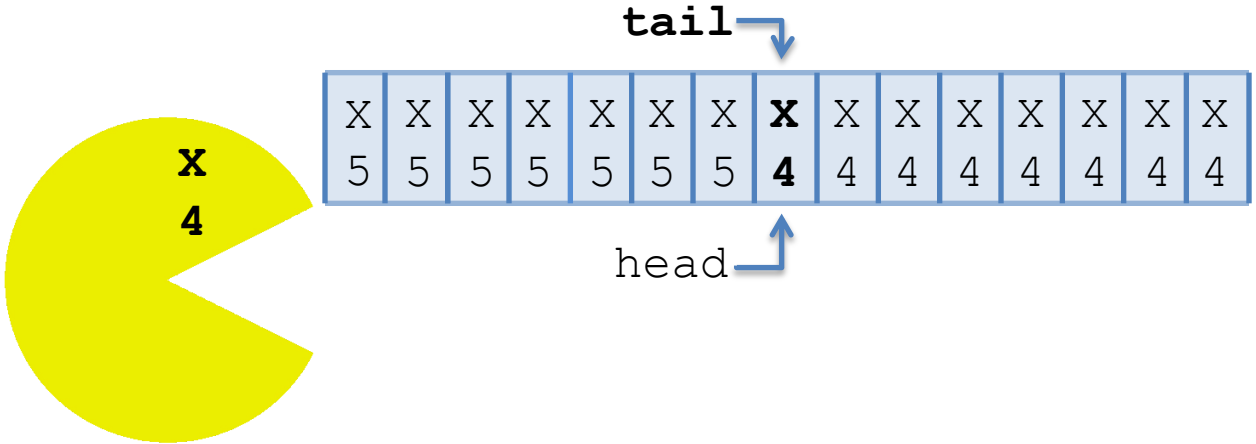


Queue is Full!

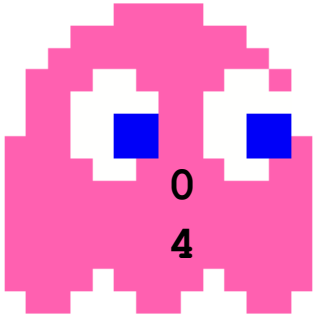
New Path



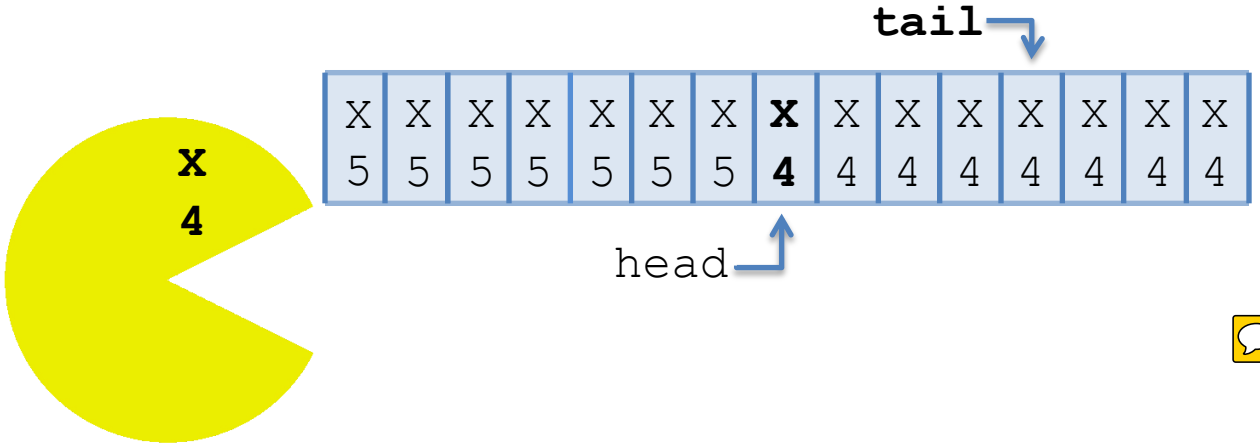

```
class Queue {
 entry buffer[SIZE];
 int headish;
 int tailish;
 int generation;
};
```



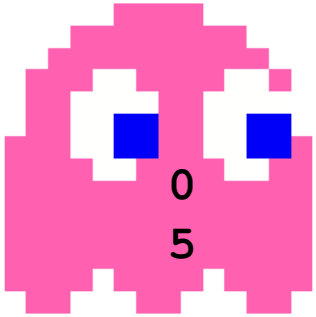
New Path



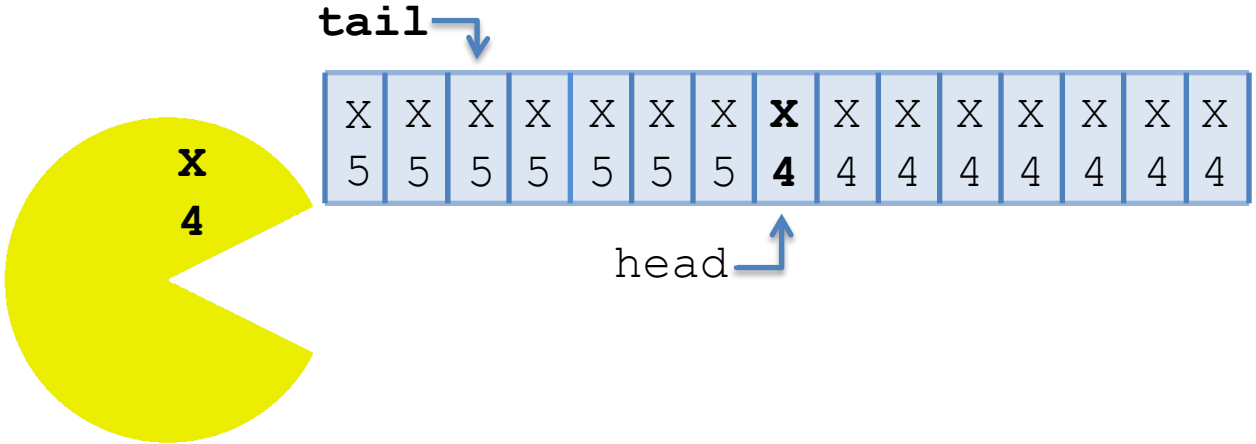

```
class Queue {
 entry buffer[SIZE];
 int headish;
 int tailish;
 int generation;
};
```



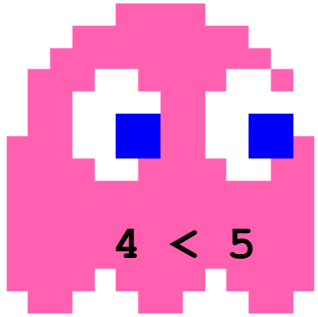
New Path



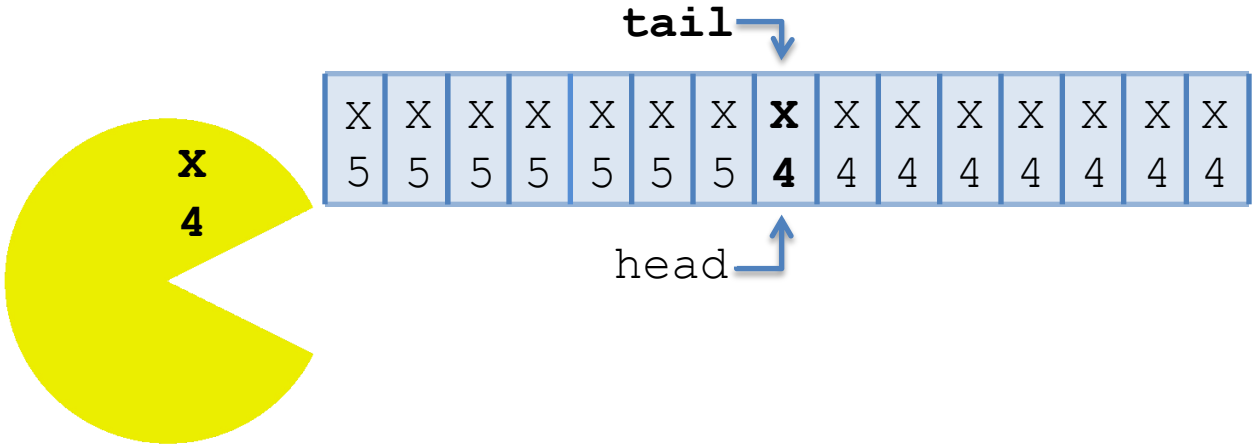

```
class Queue {
 entry buffer[SIZE];
 int headish;
 int tailish;
 int generation;
};
```



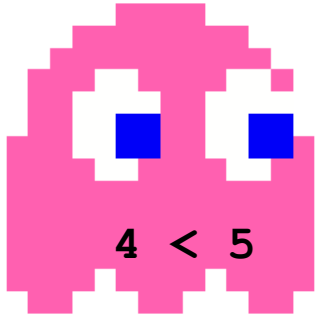
New Path



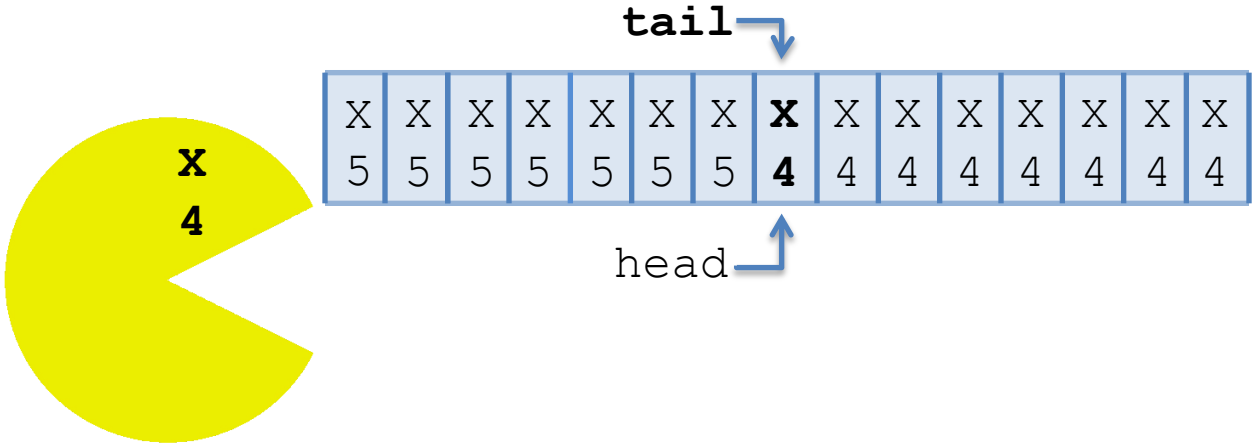

```
class Queue {
 entry buffer[SIZE];
 int headish;
 int tailish;
 int generation;
};
```



New Path




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



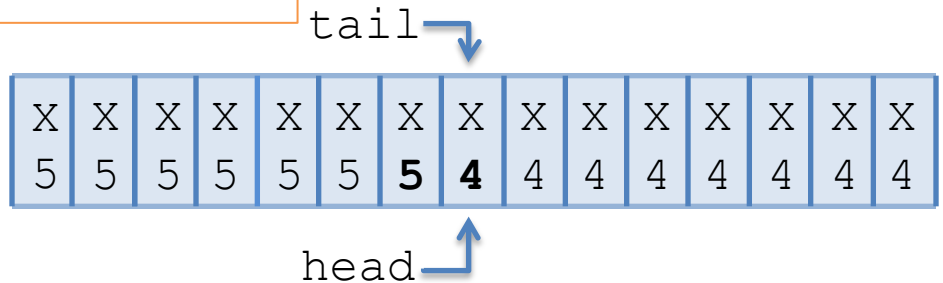
Queue is Full!

New Path

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

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

So…Comptomisation?

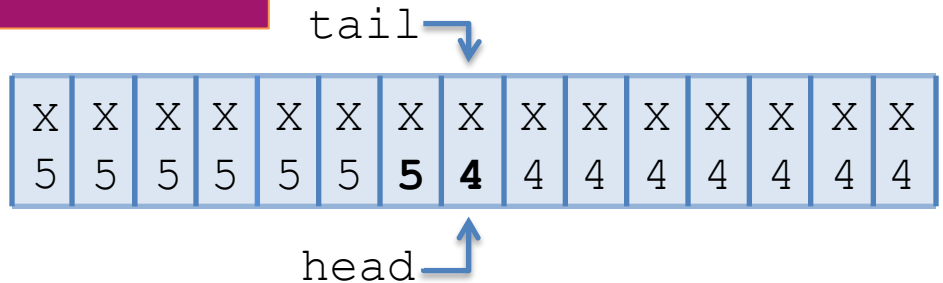


New Path

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

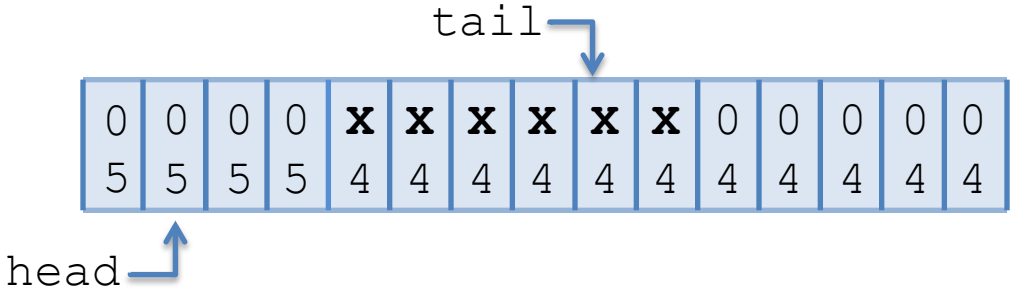
```
class Queue {
 entry buffer[SIZE];
 int headish;
 int tailish;
 int generation;
};
```

AWESOMIZATION!



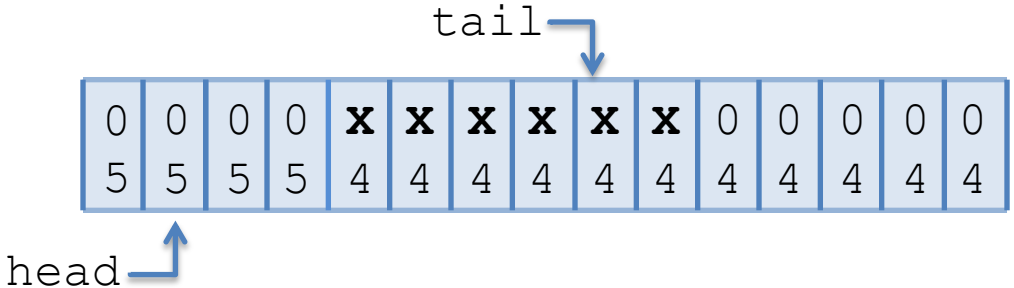
Atomicization!

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



Atomicization

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

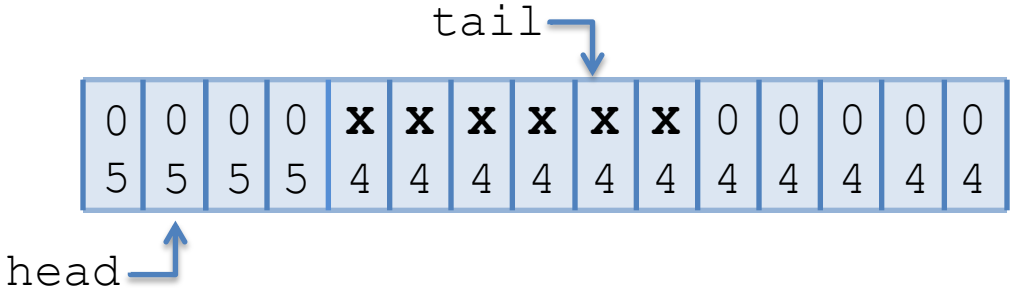


Atomicization

small enough

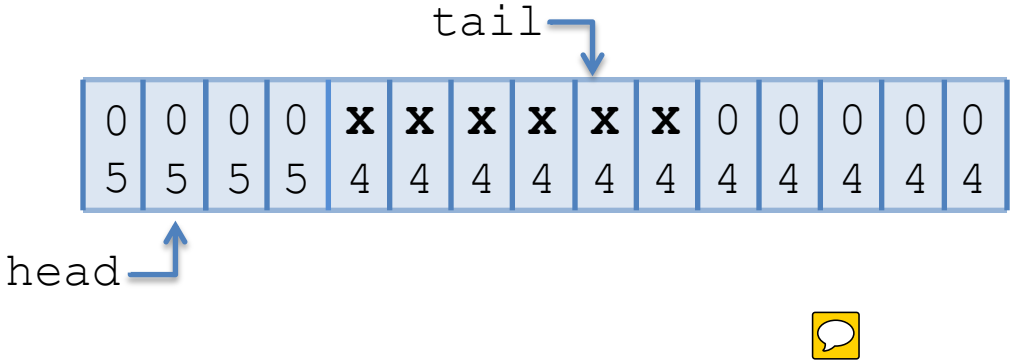
```
struct entry {
 int data;
 int generation;
};
```

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



Atomicization

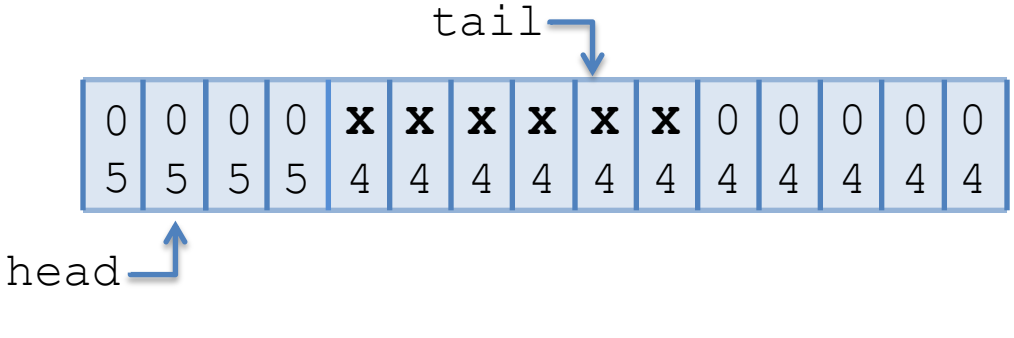
```
class Queue {
 atomic<entry> buffer[SIZE];
 laxtomic<int> headish;
 laxtomic<int> tailish;
 int generation;
};
```



Atomicization

?

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

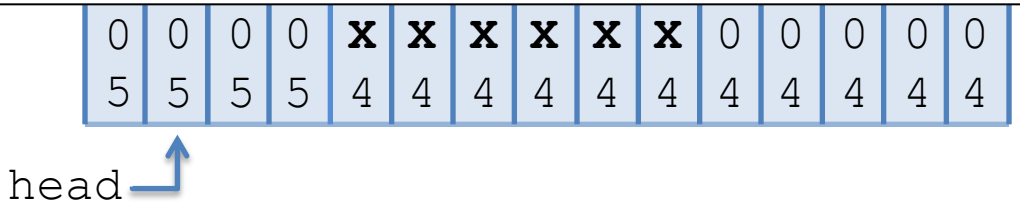


Atomicization

?

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

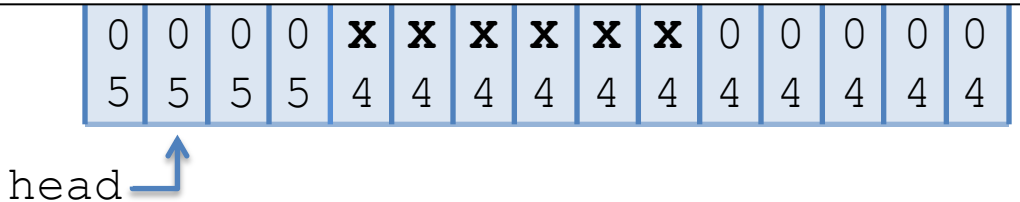
Sorry Herb…



Atomicization

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

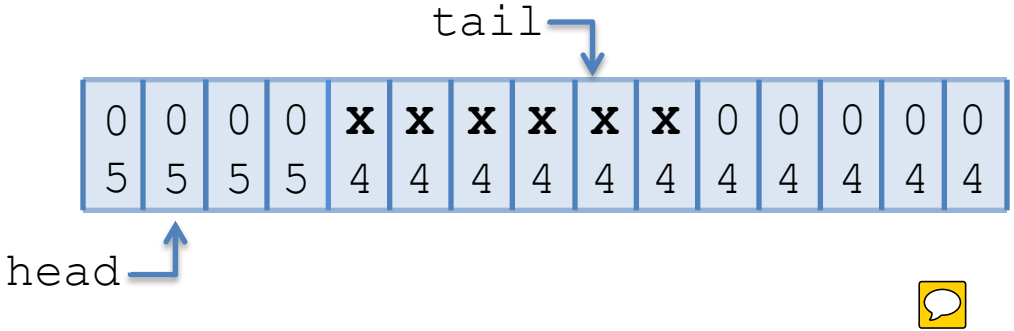
```
template <typename T> struct laxtomic : atomic<T> {
 ...function...(... memory_order = memory_order_relaxed);
};
```



Atomicization

OK?

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



Atomicization

OK?

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

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

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

Atomicization

OK?

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

```
data.x = 10;
data.y = 20;
data.ready.store(true,mo_release);
data.x = 10; // no!
```

release:
before means before

```
    x = data.x; // no!
if (data.ready.load(mo_acquire))
{
    x = data.x;
    y = data.y;
    ...
}
```

acquire:
after means after

Atomicization

OK?

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

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

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

Atomicization

OK?

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

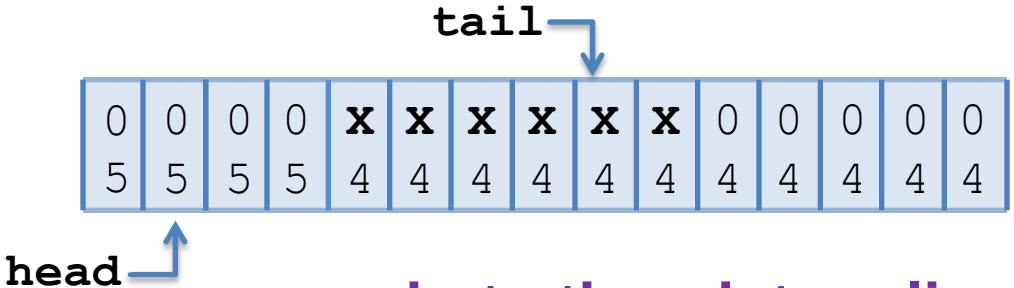
```
data.x = 10;
data.y = 20;
data.ready.store(true,mo_relaxed);
```

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

Atomicization

OK?

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

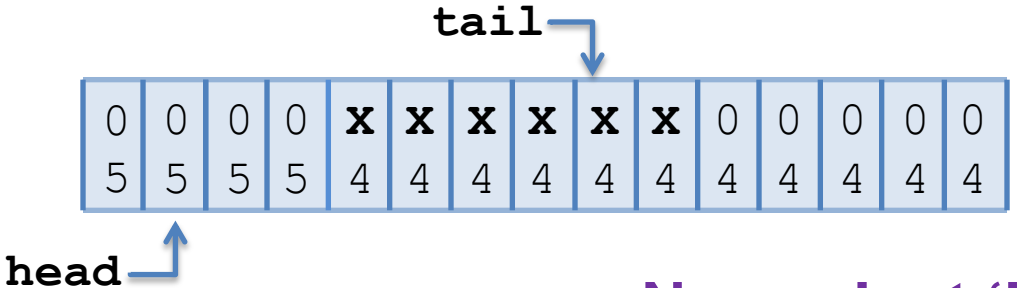


what other data relies on head/tail?

Atomicization

OK

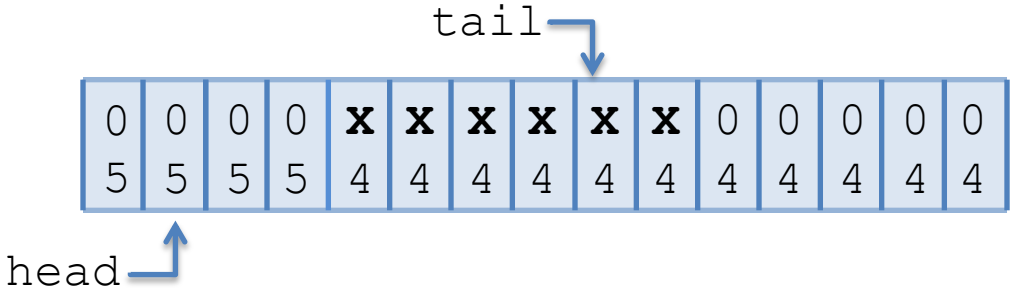
```
class Queue {
 atomic<entry> buffer[SIZE];
 laxtomic<int> headish;
 laxtomic<int> tailish;
 int generation;
};
```



None. Just ‘hints’.

Atomicization

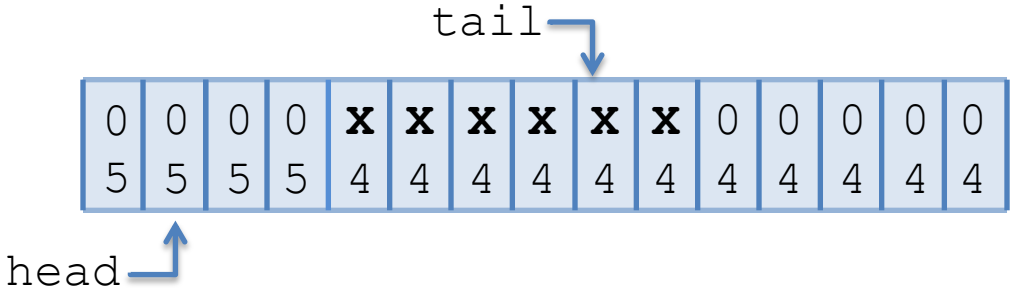
```
class Queue {
 atomic<entry> buffer[SIZE];
 laxtomic<int> head;
 laxtomic<int> tail;
 int generation;
};
```



Atomicization

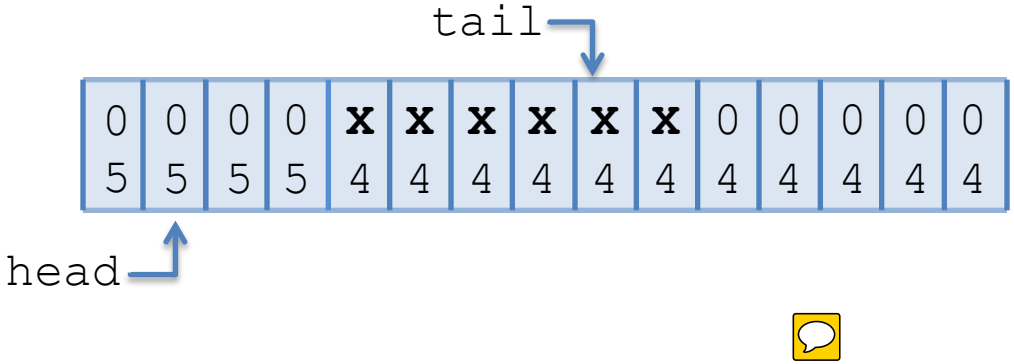
?

```
class Queue {
 atomic<entry> buffer[SIZE];
 laxtomic<int> head;
 laxtomic<int> tail;
 int generation;
};
```



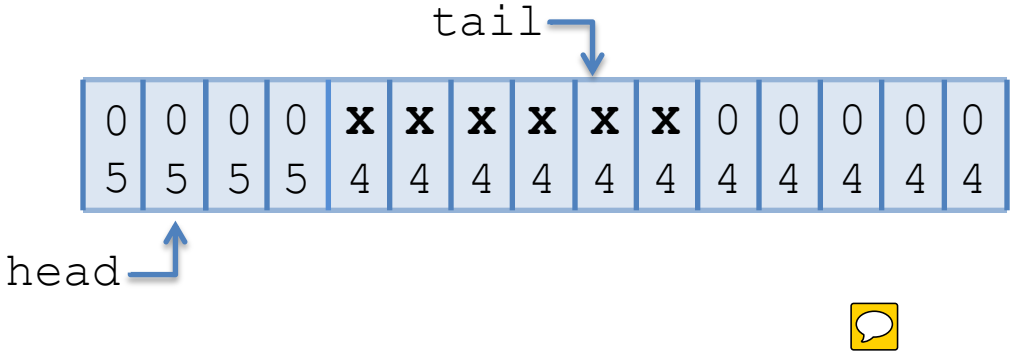
Atomicization

```
class Queue {
 atomic<entry> buffer[SIZE];
 laxtomic<int> head;
 laxtomic<int> tail;
 laxtomic<int> generation;
};
```



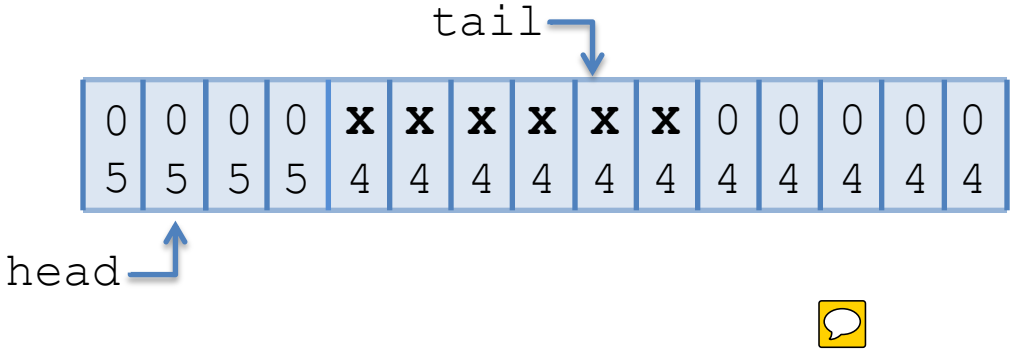
Atomicization

```
class Queue {
 atomic<entry> buffer[SIZE];
 laxtomic<int> head;
 laxtomic<int> tail;
 laxtomic<int> generation;
};
```



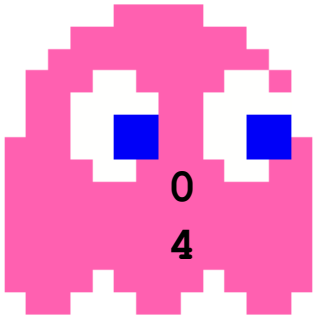
Atomicization

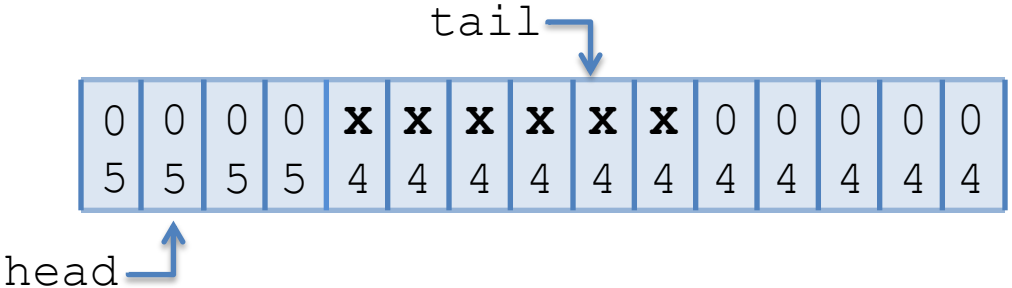
```
class Queue {
 atomic<entry> buffer[SIZE];
 laxtomic<int> head;
 laxtomic<int> tail;
 laxtomic<int> generation;
};
```

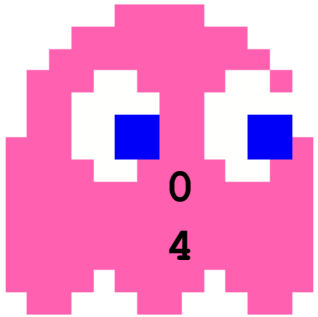


Codization

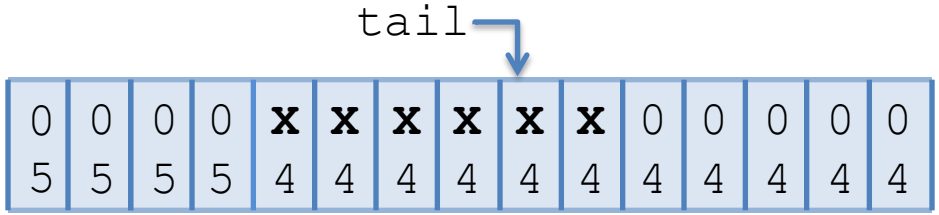
```
void push(int val) {
  …
}
```





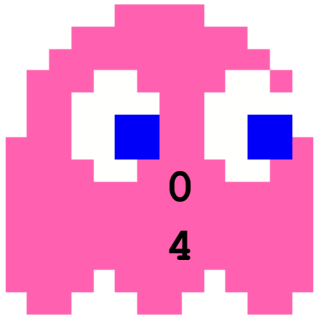



```
bool push(int val) {   entry ent;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )
        incr(tmp, gen);
 } while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
```




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

```
void incr(int & tmp, int &gen)
{
   if (++tmp == SIZE) {
      tmp = 0;
      gen++;
   }
}
```



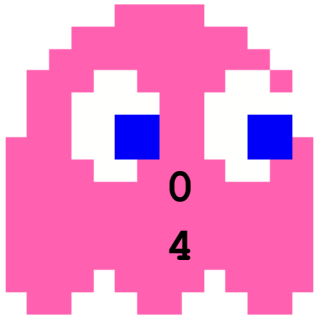

```
bool push(int val) {   entry ent;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )
        incr(tmp, gen);
 } while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
```

tmp

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

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

```
void incr(int & tmp, int &gen)
{
   if (++tmp == SIZE) {
      tmp = 0;
      gen++;
   }
}
```



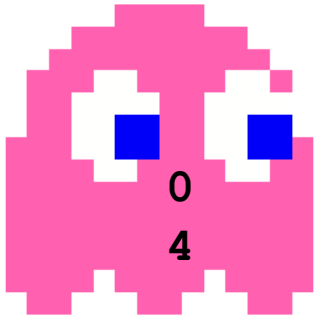

```
bool push(int val) {   entry ent;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )
        incr(tmp, gen);
 } while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
```

tmp

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

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

```
void incr(int & tmp, int &gen)
{
   if (++tmp == SIZE) {
      tmp = 0;
      gen++;
   }
}
```



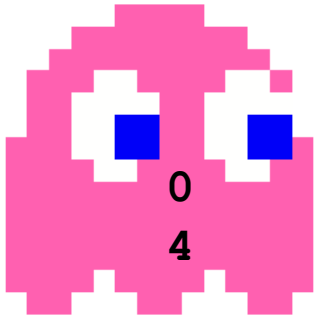

```
bool push(int val) {   entry ent;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )
        incr(tmp, gen);
 } while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
```

tmp

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

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

```
void incr(int & tmp, int &gen)
{
   if (++tmp == SIZE) {
      tmp = 0;
      gen++;
   }
}
```



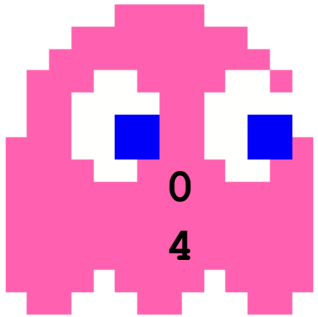

```
bool push(int val) {   entry ent;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )
        incr(tmp, gen);
 } while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
```

tmp

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

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

```
void incr(int & tmp, int &gen)
{
   if (++tmp == SIZE) {
      tmp = 0;
      gen++;
   }
}
```



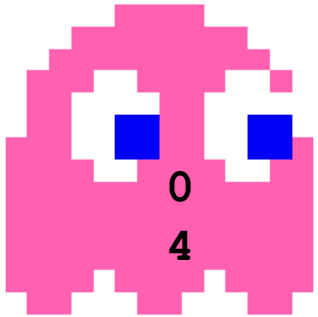

```
bool push(int val) {   entry ent;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )
        incr(tmp, gen);
 } while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
```

tmp

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

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

```
void incr(int & tmp, int &gen)
{
   if (++tmp == SIZE) {
      tmp = 0;
      gen++;
   }
}
```



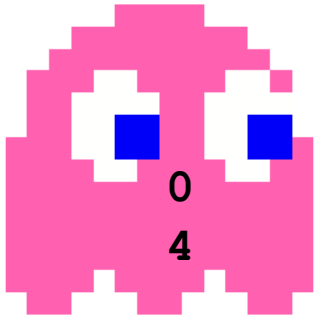

```
bool push(int val) {   entry ent;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )
        incr(tmp, gen);
 } while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
```

tmp

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

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

```
void incr(int & tmp, int &gen)
{
   if (++tmp == SIZE) {
      tmp = 0;
      gen++;
   }
}
```



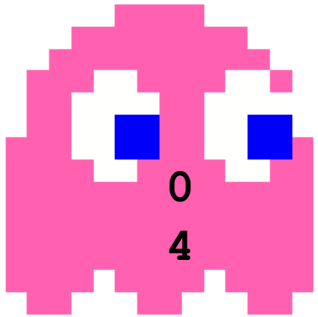

```
bool push(int val) {   entry ent;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )
        incr(tmp, gen);
} while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
```

tmp

0	0	0	0	x	x	x	x	x	x	?	0	0	0	0
5	5	5	5	4	4	4	4	4	4	4	4	4	4	4

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

```
void incr(int & tmp, int &gen)
{
   if (++tmp == SIZE) {
      tmp = 0;
      gen++;
   }
}
```



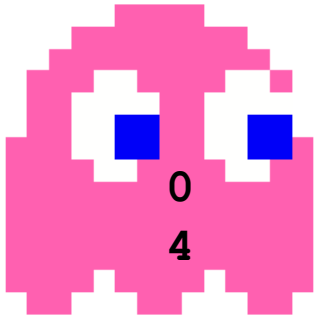

```
bool push(int val) {   entry ent;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )
        incr(tmp, gen);
 } while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
```

tmp

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

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

```
void incr(int & tmp, int &gen)
{
   if (++tmp == SIZE) {
      tmp = 0;
      gen++;
   }
}
```



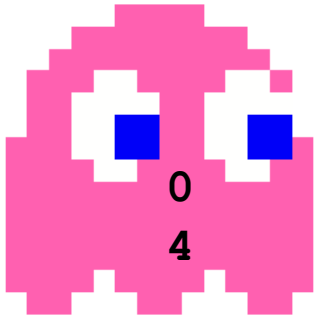

```
bool push(int val) {   entry ent;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )
        incr(tmp, gen);
} while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
```

tmp

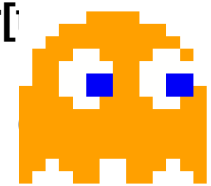
0	0	0	0	x	x	x	x	x	x	?	0	0	0	0
5	5	5	5	4	4	4	4	4	4	4	4	4	4	4

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

```
void incr(int & tmp, int &gen)
{
   if (++tmp == SIZE) {
      tmp = 0;
      gen++;
   }
}
```




```
bool push(int val) {   entry ent;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[                relaxed), gen) )
        incr(tmp, gen);
 } while ( ! buffer[tmp].CAS(ent,                 en}) ); // NOT relaxed
```

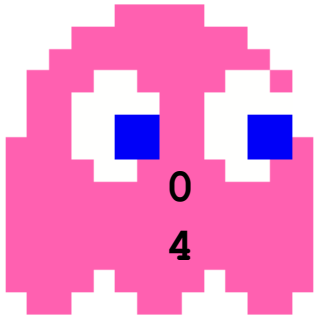


tmp

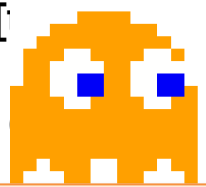
0 5	0 5	0 5	0 5	**x** 4	**x** 4	**x** 4	**x** 4	**x** 4	**x** 4	**C** 4	0 4	0 4	0 4	0 4

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

```
void incr(int & tmp, int &gen)
{
   if (++tmp == SIZE) {
      tmp = 0;
      gen++;
   }
}
```




```
bool push(int val) {   entry ent;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[                relaxed), gen) )
        incr(tmp, gen);
 } while ( ! buffer[tmp].CAS(ent,                 en}) ); // NOT relaxed
```

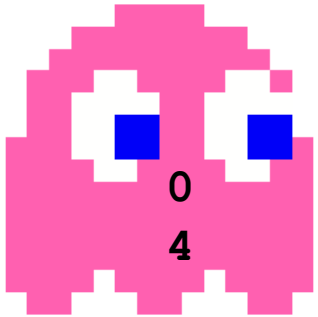


The nature of “lock-free” – you fail *only* when someone else makes progress

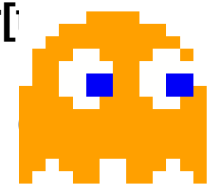
0	0	0	0	x	x	x	x	x	x	c	0	0	0	0
5	5	5	5	4	4	4	4	4	4	4	4	4	4	4

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

```
void incr(int & tmp, int &gen)
{
   if (++tmp == SIZE) {
      tmp = 0;
      gen++;
   }
}
```




```
bool push(int val) {   entry ent;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[                relaxed), gen) )
        incr(tmp, gen);
 } while ( ! buffer[tmp].CAS(ent,                 en}) ); // NOT relaxed
```



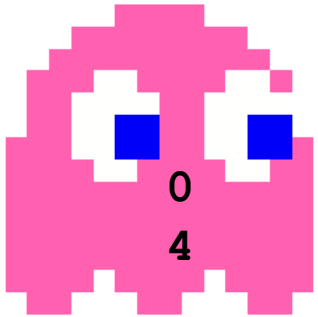
CAS loop: loops on progress
Spin-lock: loops on NON progress

tmp

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

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

```
void incr(int & tmp, int &gen)
{
   if (++tmp == SIZE) {
      tmp = 0;
      gen++;
   }
}
```



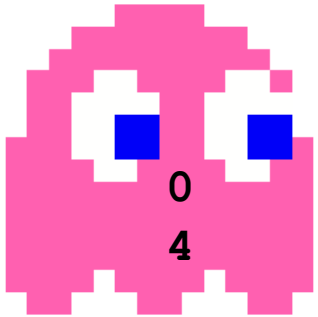

```
bool push(int val) {   entry ent;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )
        incr(tmp, gen);
 } while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
```

tmp

0	0	0	0	x	x	x	x	x	x	c	0	0	0	0
5	5	5	5	4	4	4	4	4	4	4	4	4	4	4

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

```
void incr(int & tmp, int &gen)
{
   if (++tmp == SIZE) {
      tmp = 0;
      gen++;
   }
}
```



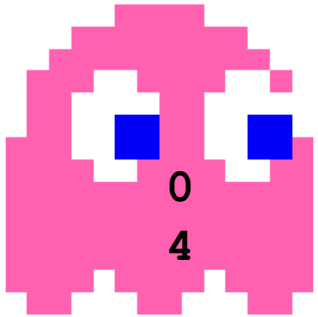

```
bool push(int val) {   entry ent;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )
        incr(tmp, gen);
 } while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
```

tmp

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

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

```
void incr(int & tmp, int &gen)
{
   if (++tmp == SIZE) {
      tmp = 0;
      gen++;
   }
}
```



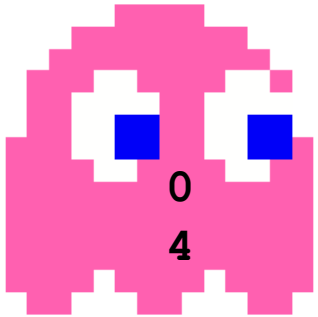

```
bool push(int val) {   entry ent;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )
        incr(tmp, gen);
 } while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
```

tmp

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

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

```
void incr(int & tmp, int &gen)
{
   if (++tmp == SIZE) {
      tmp = 0;
      gen++;
   }
}
```



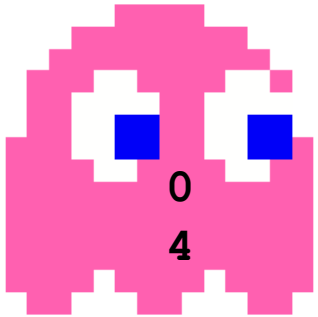

```
bool push(int val) {   entry ent;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )
        incr(tmp, gen);
 } while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
```

tmp

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

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

```
void incr(int & tmp, int &gen)
{
   if (++tmp == SIZE) {
      tmp = 0;
      gen++;
   }
}
```



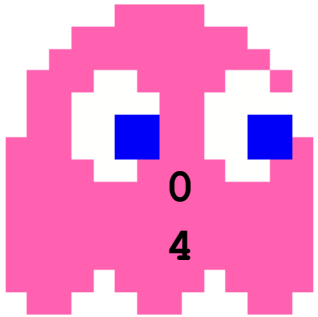

```
bool push(int val) {   entry ent;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )
        incr(tmp, gen);
} while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
```

tmp

0	0	0	0	X	X	X	X	X	X	C	?	0	0	0
5	5	5	5	4	4	4	4	4	4	4	4	4	4	4

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

```
void incr(int & tmp, int &gen)
{
   if (++tmp == SIZE) {
      tmp = 0;
      gen++;
   }
}
```



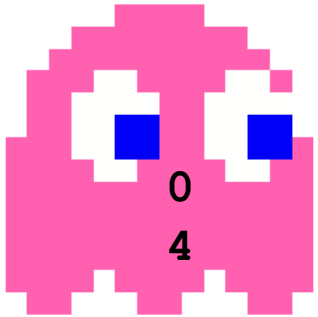

```
bool push(int val) {   entry ent;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )
        incr(tmp, gen);
 } while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
```

tmp

0	0	0	0	X	X	X	X	X	X	C	P	0	0	0
5	5	5	5	4	4	4	4	4	4	4	4	4	4	4

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

```
void incr(int & tmp, int &gen)
{
   if (++tmp == SIZE) {
      tmp = 0;
      gen++;
   }
}
```




```
bool push(int val) {   entry ent;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )
        incr(tmp, gen);
 } while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
```

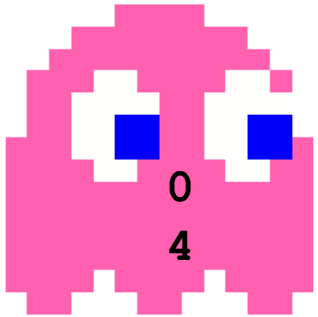
?

tmp

0	0	0	0	X	X	X	X	X	X	C	P	0	0	0
5	5	5	5	4	4	4	4	4	4	4	4	4	4	4

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

```
void incr(int & tmp, int &gen)
{
   if (++tmp == SIZE) {
      tmp = 0;
      gen++;
   }
}
```




```
bool push(int val) {   entry ent;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )
        incr(tmp, gen);
 } while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
```

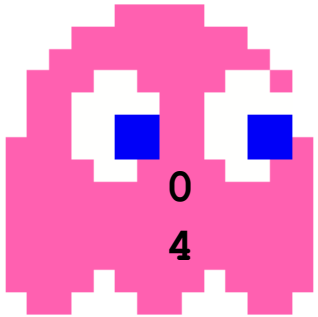
***Should* update generation & tail**

tmp

0	0	0	0	X	X	X	X	X	X	C	P	0	0	0
5	5	5	5	4	4	4	4	4	4	4	4	4	4	4

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

```
void incr(int & tmp, int &gen)
{
   if (++tmp == SIZE) {
      tmp = 0;
      gen++;
   }
}
```




```
bool push(int val) {   entry ent;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )
        incr(tmp, gen);
 } while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
 update(tmp, gen);
```

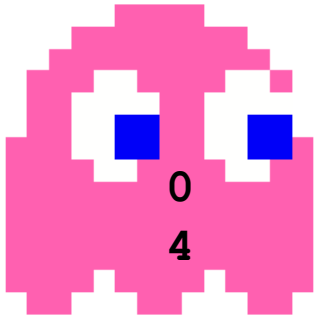
***Should* update generation & tail**

tmp

0	0	0	0	X	X	X	X	X	X	C	P	0	0	0
5	5	5	5	4	4	4	4	4	4	4	4	4	4	4

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

```
void incr(int & tmp, int &gen)
{
   if (++tmp == SIZE) {
      tmp = 0;
      gen++;
   }
}
```




```
bool push(int val) {   entry ent;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )
        incr(tmp, gen);
 } while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
 update(tmp, gen);
```

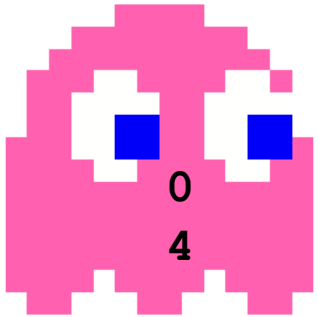
But first…

tmp

0	0	0	0	x	x	x	x	x	x	C	P	0	0	0
5	5	5	5	4	4	4	4	4	4	4	4	4	4	4

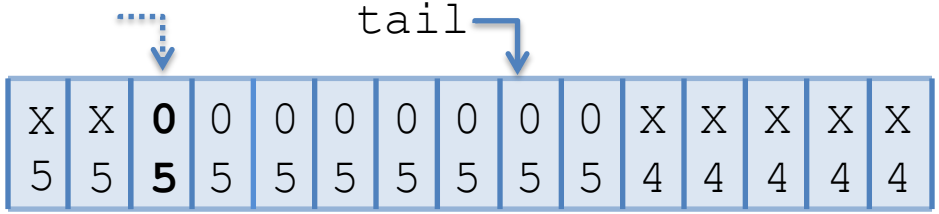
```
bool is_zero(entry e, int gen) {
   return e.data == 0 && e.gen == gen;
}
```

```
void incr(int & tmp, int &gen)
{
   if (++tmp == SIZE) {
      tmp = 0;
      gen++;
   }
}
```



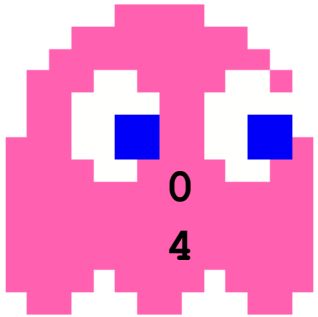

```
bool push(int val) {   entry ent;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )
        incr(tmp, gen);
 } while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
 update(tmp, gen);
```

But first…

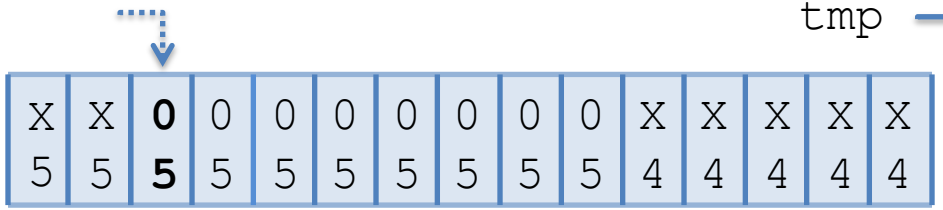


Wrapped

```
void incr(int & tmp, int &gen)
{
   if (++tmp == SIZE) {
      tmp = 0;
      gen++;
   }
}
```

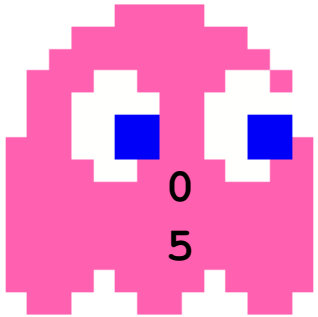



```
bool push(int val) {   entry ent;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )
        incr(tmp, gen);
 } while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
 update(tmp, gen);
```



Wrapped

```
void incr(int & tmp, int &gen)
{
   if (++tmp == SIZE) {
      tmp = 0;
      gen++;
   }
}
```



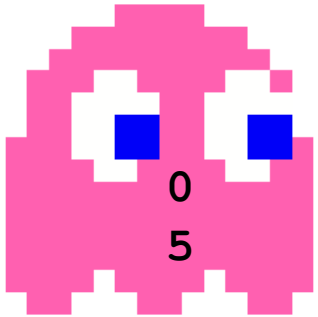

```
bool push(int val) {   entry ent;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )
        incr(tmp, gen);
 } while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
 update(tmp, gen);
```

tmp

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

Wrapped

```
void incr(int & tmp, int &gen)
{
   if (++tmp == SIZE) {
      tmp = 0;
      gen++;
   }
}
```



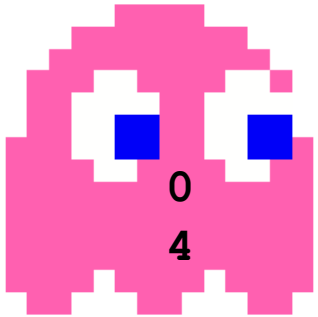

```
bool push(int val) {   entry ent;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )
        incr(tmp, gen);
} while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
 update(tmp, gen);
```

tmp

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

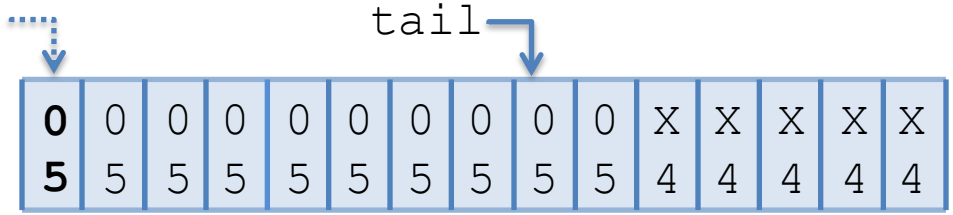
Wrapped

```
void incr(int & tmp, int &gen)
{
   if (++tmp == SIZE) {
      tmp = 0;
      gen++;
   }
}
```



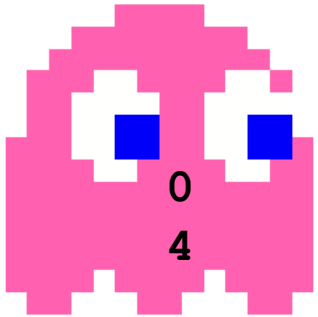

```
bool push(int val) {   entry ent;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )
        incr(tmp, gen);
 } while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
 update(tmp, gen);
```

But first…



Wrapped

```
void incr(int & tmp, int &gen)
{
   if (++tmp == SIZE) {
      tmp = 0;
      gen++;
   }
}
```




```
bool push(int val) {   entry ent;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )
        incr(tmp, gen);
 } while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
 update(tmp, gen);
```

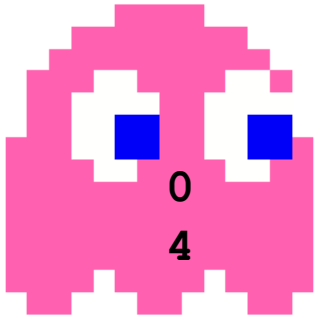
tail

But first…

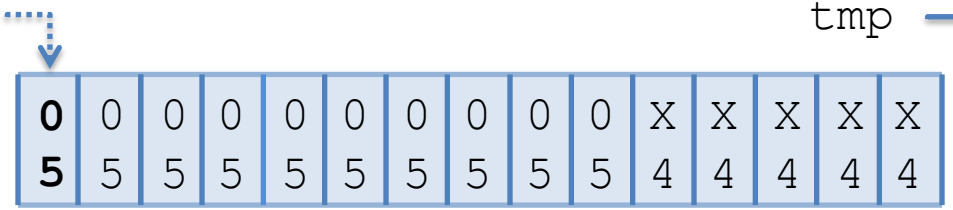
0 **5**	0 5	0 5	0 5	0 5	0 5	0 5	0 5	0 5	0 5	X 4	X 4	X 4	X 4	X 4

Wrapped

```
void incr(int & tmp, int &gen)
{
   if (++tmp == SIZE) {
      tmp = 0;
      gen++;
   }
}
```

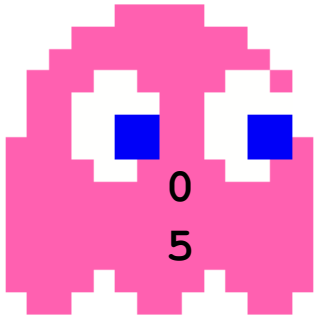



```
bool push(int val) {   entry ent;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )
        incr(tmp, gen);
 } while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
 update(tmp, gen);
```



Wrapped

```
void incr(int & tmp, int &gen)
{
   if (++tmp == SIZE) {
      tmp = 0;
      gen++;
   }
}
```



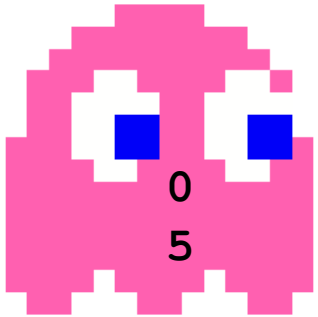

```
bool push(int val) {   entry ent;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )
        incr(tmp, gen);
 } while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
 update(tmp, gen);
```

tmp

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

Wrapped

```
void incr(int & tmp, int &gen)
{
   if (++tmp == SIZE) {
      tmp = 0;
      gen++;
   }
}
```



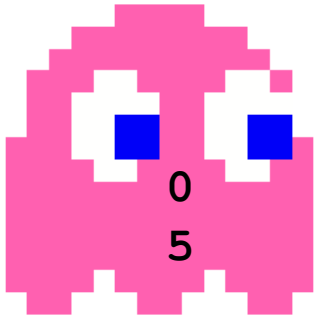

```
bool push(int val) {   entry ent;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )
        incr(tmp, gen);
} while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
 update(tmp, gen);
```

tmp

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

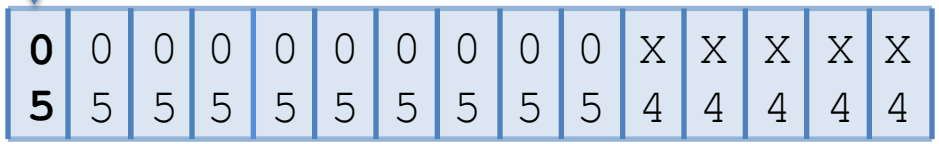
Wrapped

```
void incr(int & tmp, int &gen)
{
   if (++tmp == SIZE) {
      tmp = 0;
      gen++;
   }
}
```



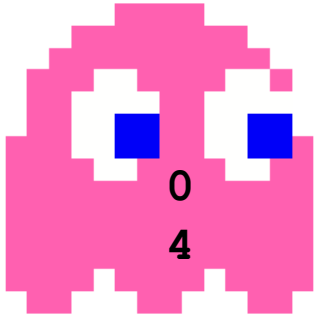

```
bool push(int val) {   entry ent;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )
        incr(tmp, gen);
} while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
 update(tmp, gen);
```

tmp



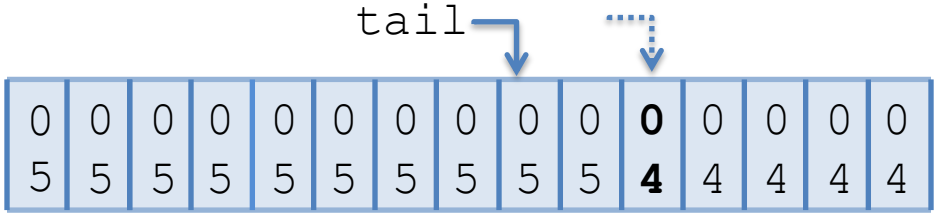
Wrapped

```
void incr(int & tmp, int &gen)
{
   if (++tmp == SIZE) {
      tmp = 0;
      gen++;
   }
}
```

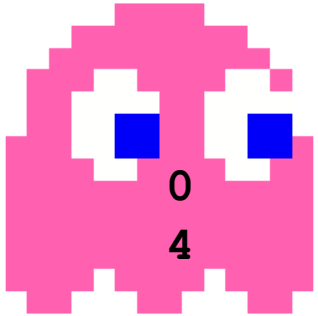



```
bool push(int val) {   entry ent;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )
        incr(tmp, gen);
 } while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
 update(tmp, gen);
```

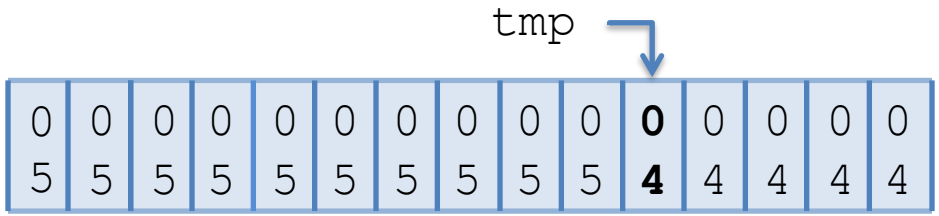
But first…



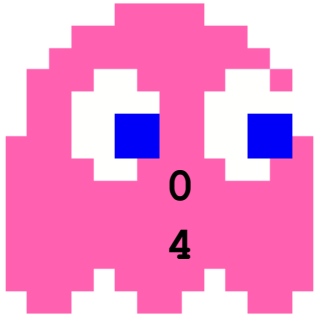
Queue is Empty




```
bool push(int val) {   entry ent;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )
        incr(tmp, gen);
 } while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
 update(tmp, gen);
```

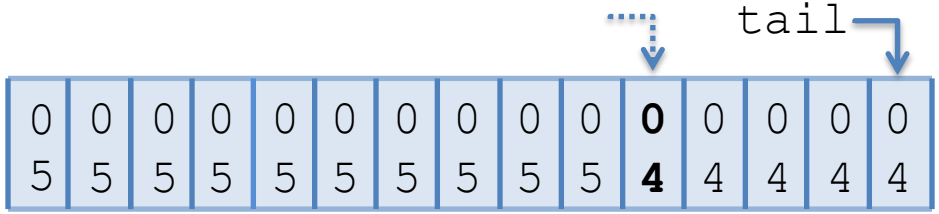


Queue is Empty

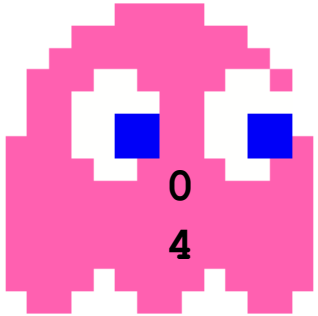



```
bool push(int val) {   entry ent;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )
        incr(tmp, gen);
 } while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
 update(tmp, gen);
```

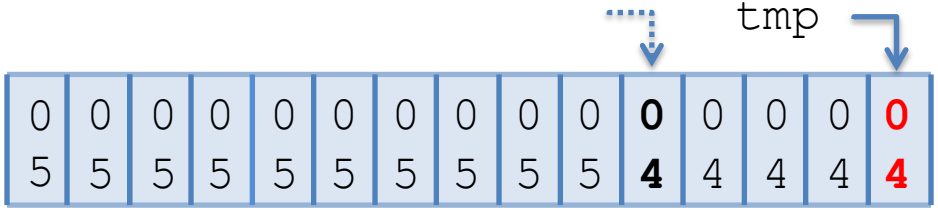
But first…



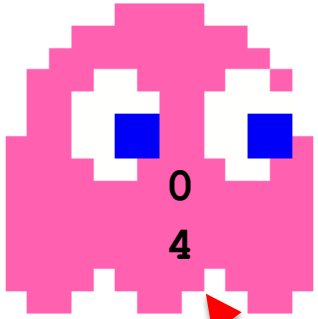
Queue is Empty




```
bool push(int val) {   entry ent;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )
        incr(tmp, gen);
 } while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
 update(tmp, gen);
```



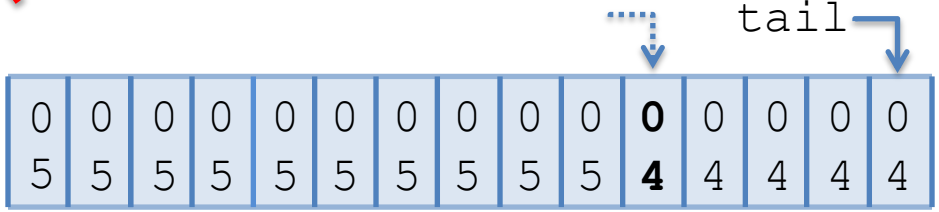
Queue is Empty



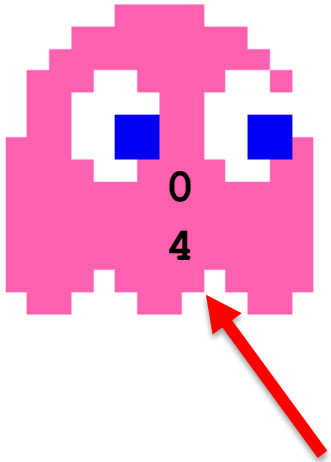

```
bool push(int val) {   entry ent;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )
        incr(tmp, gen);
 } while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
 update(tmp, gen);
```

But first…

?

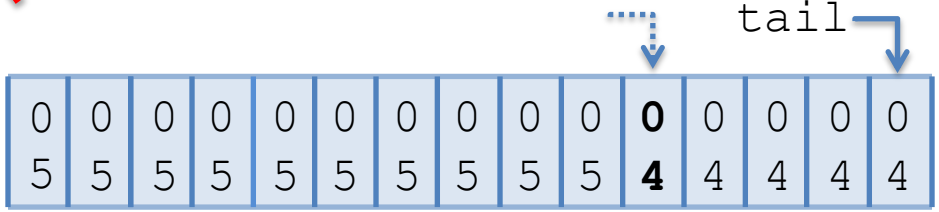


Queue is Empty



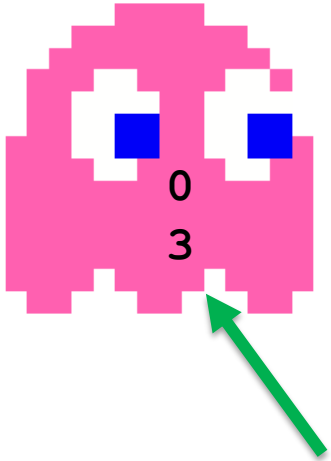

```
bool push(int val) {   entry ent;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )
        incr(tmp, gen);
 } while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
 update(tmp, gen);
```

!



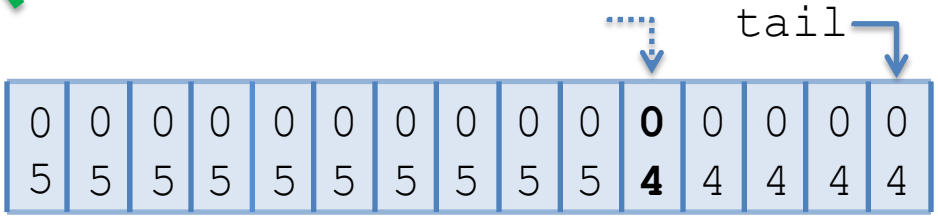
Queue is Empty

INVARIANT: tail@gen is "<=" real tail

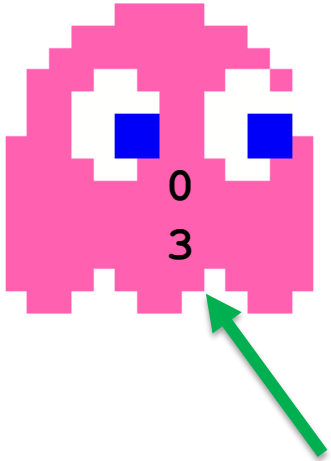



```
bool push(int val) {   entry ent;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )
        incr(tmp, gen);
 } while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
 update(tmp, gen);
```

But first…

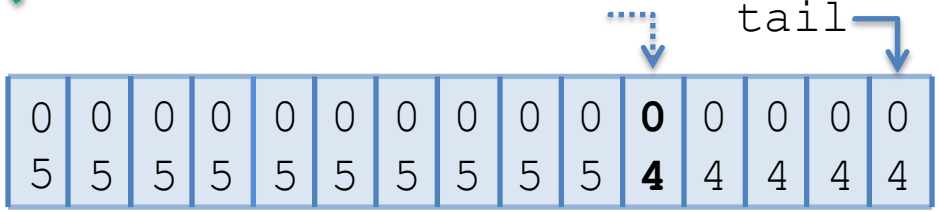


Queue is Empty



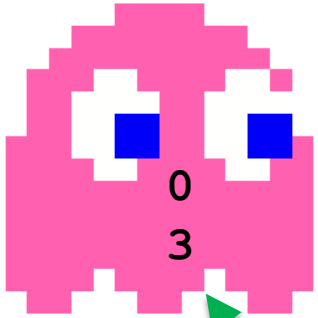

```
bool push(int val) {   entry ent;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )
        incr(tmp, gen);
 } while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
 update(tmp, gen);
```

But first…



Queue is Empty

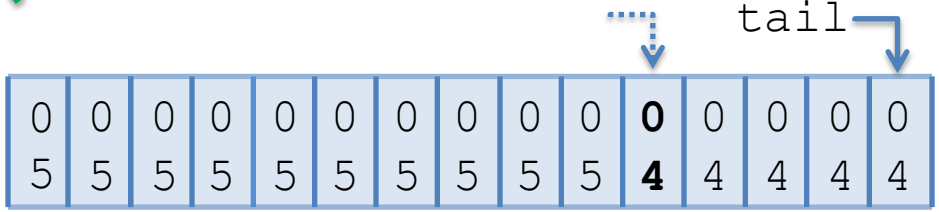
INVARIANT? generation >= 4 ?



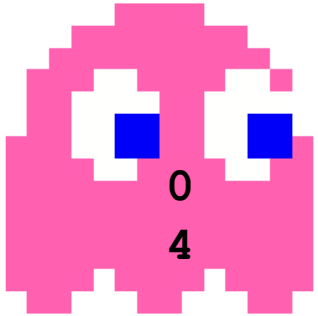

```
bool push(int val) {   entry ent;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )
        incr(tmp, gen);
 } while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
 update(tmp, gen);
```

But first…

✓

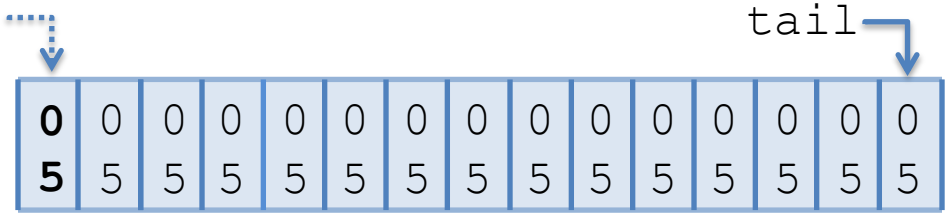


Queue is Empty

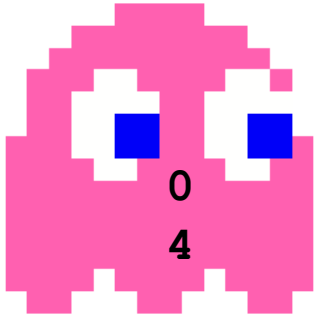



```
bool push(int val) {   entry ent;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )
        incr(tmp, gen);
 } while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
 update(tmp, gen);
```

But first…

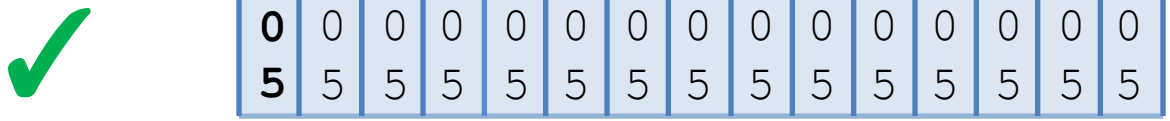


Queue is Empty

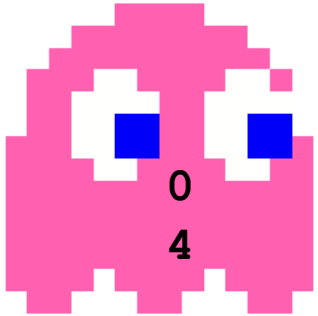



```
bool push(int val) {   entry ent;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )
        incr(tmp, gen);
 } while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
 update(tmp, gen);
```

tmp



Queue is Empty




```
bool push(int val) {   entry ent;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )
        incr(tmp, gen);
 } while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
 update(tmp, gen);
```

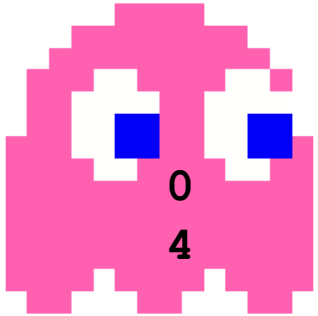
tail

But first…

✓

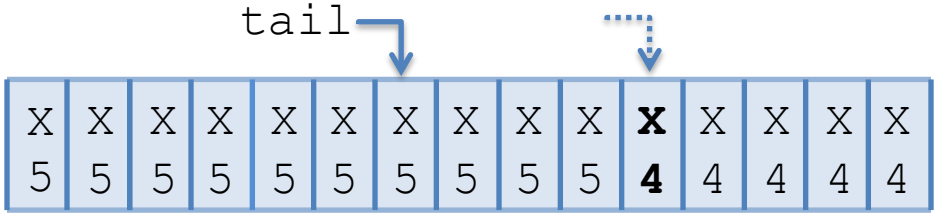
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Initialized state

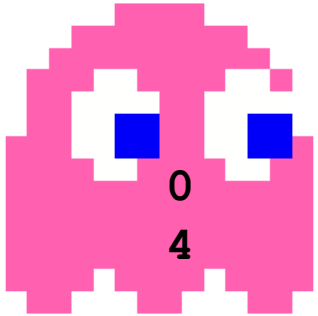



```
bool push(int val) {   entry ent;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )
        incr(tmp, gen);
 } while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
 update(tmp, gen);
```

But first…

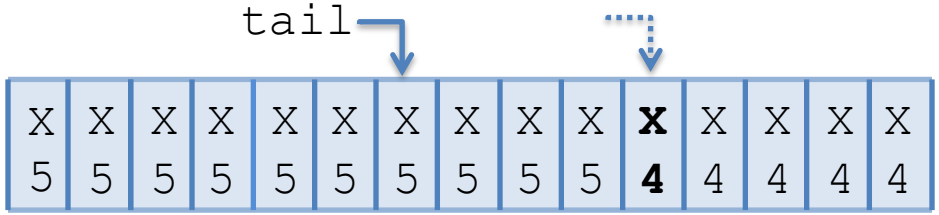


Queue is FULL



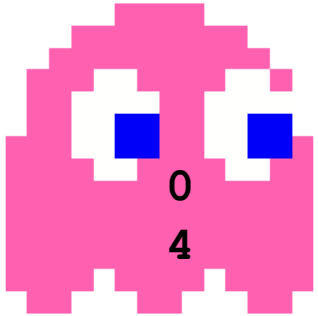

```
bool push(int val) {   entry ent;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )
        incr(tmp, gen);
 } while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
 update(tmp, gen);
```

But first…



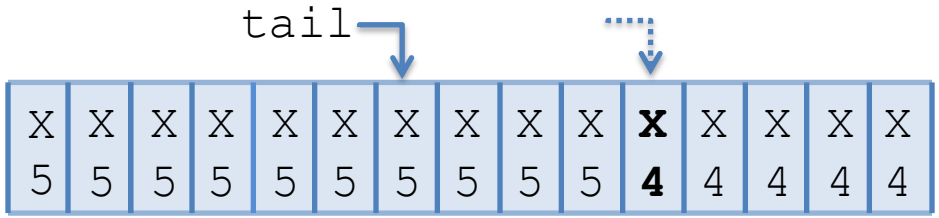
Queue is FULL

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



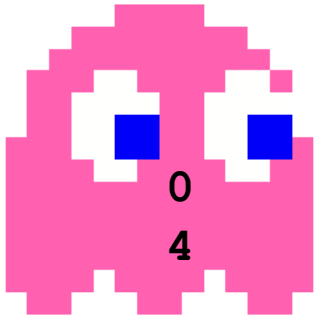

```
bool push(int val) {   entry ent;   int prev = 0;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )  {
        if (ent.gen < prev) return false;  // FULL
        else incr(tmp, gen);
        if (ent.data) prev = ent.gen;       }
 } while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
```

But first…



Queue is FULL

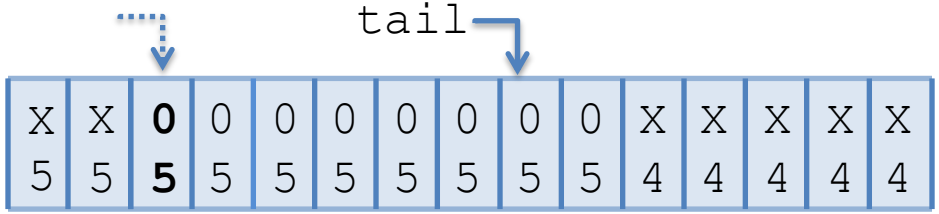
```
bool is_zero(entry e, int gen) {
   return e.data == 0 && e.gen == gen;
}
```




```
bool push(int val) {   entry ent;   int prev = 0;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )  {
        if (ent.gen < prev) return false;  // FULL
        else incr(tmp, gen);
        if (ent.data) prev = ent.gen;       }
 } while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
```

But first…

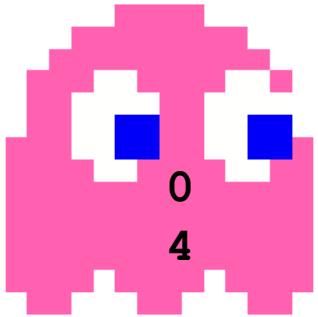
?



Wrapped

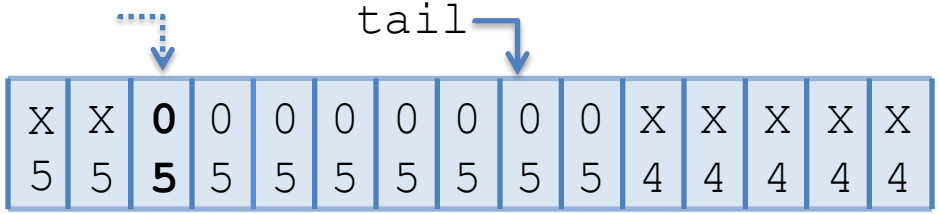
Recheck!?

```
void incr(int & tmp, int &gen)
{
   if (++tmp == SIZE) {
      tmp = 0;
      gen++;
   }
}
```



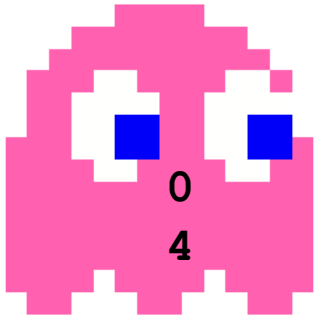

```
bool push(int val) {   entry ent;   int prev = 0;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )  {
        if (ent.gen < prev) return false;  // FULL
        else incr(tmp, gen);
        if (ent.data) prev = ent.gen;       }
 } while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
```

But first…



Wrapped

```
void incr(int & tmp, int &gen)
{
   if (++tmp == SIZE) {
      tmp = 0;
      gen++;
   }
}
```




```
bool push(int val) {   entry ent;   int prev = 0;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )  {
        if (ent.gen < prev) return false;  // FULL
        else incr(tmp, gen);
        if (ent.data) prev = ent.gen;       }
 } while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
```

tail

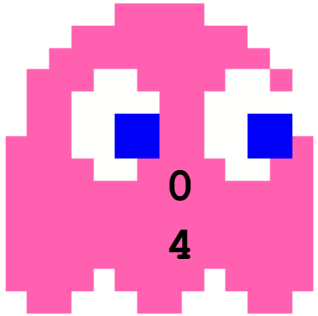
But first…

✓

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

Wrapped

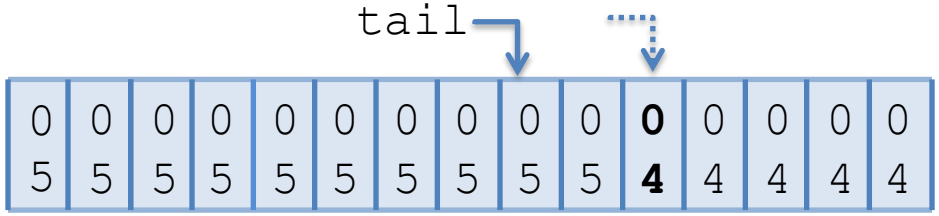
```
void incr(int & tmp, int &gen)
{
   if (++tmp == SIZE) {
      tmp = 0;
      gen++;
   }
}
```



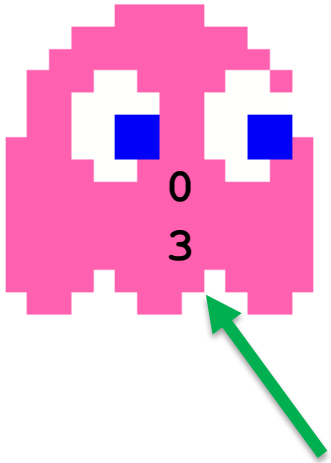

```
bool push(int val) {   entry ent;   int prev = 0;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )  {
        if (ent.gen < prev) return false;  // FULL
        else incr(tmp, gen);
        if (ent.data) prev = ent.gen;       }
 } while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
```

But first…

✓

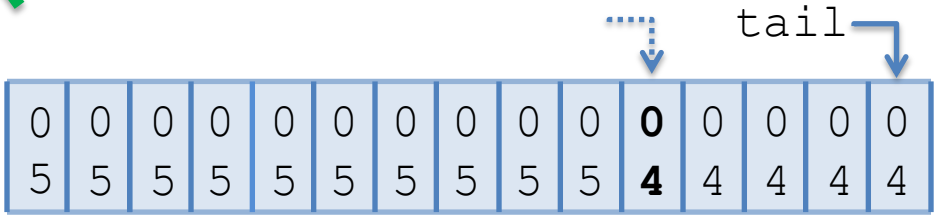


Queue is Empty

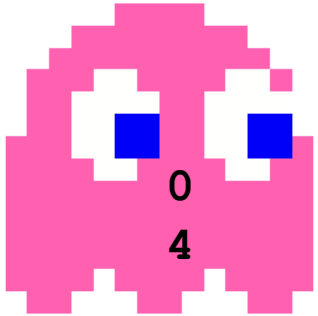



```
bool push(int val) {   entry ent;   int prev = 0;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )  {
        if (ent.gen < prev) return false;  // FULL
        else incr(tmp, gen);
        if (ent.data) prev = ent.gen;       }
 } while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
```

But first…

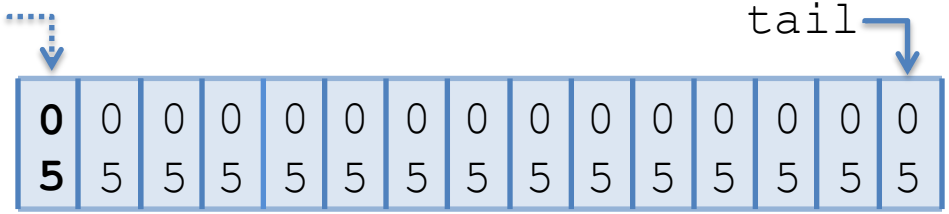


Queue is Empty

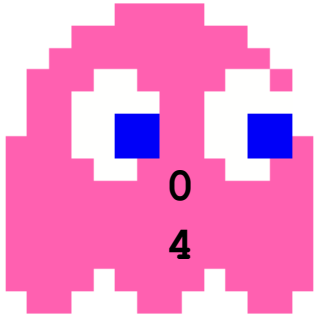



```
bool push(int val) {   entry ent;   int prev = 0;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )  {
        if (ent.gen < prev) return false;  // FULL
        else incr(tmp, gen);
        if (ent.data) prev = ent.gen;       }
 } while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
```

But first…



Queue is Empty

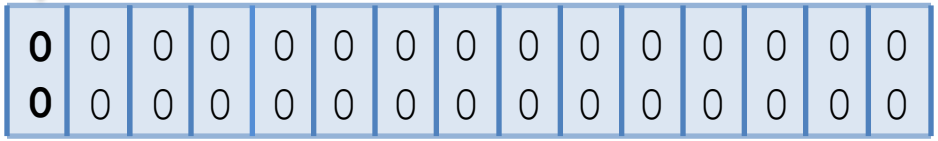



```
bool push(int val) {   entry ent;   int prev = 0;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )  {
        if (ent.gen < prev) return false;  // FULL
        else incr(tmp, gen);
        if (ent.data) prev = ent.gen;       }
 } while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
```

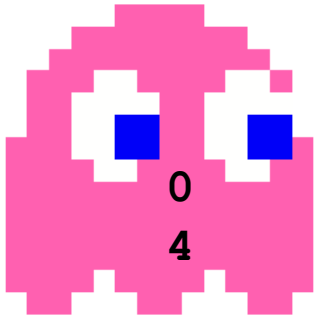
tail

But first…

✓



Initialized state




```
bool push(int val) {   entry ent;   int prev = 0;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )  {
        if (ent.gen < prev) return false;  // FULL
        else incr(tmp, gen);
        if (ent.data) prev = ent.gen;       }
 } while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
 update(tmp, gen);
```

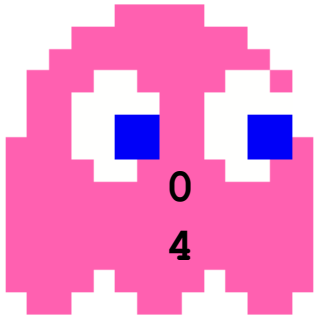
***Should* update generation & tail**

tmp

0	0	0	0	**X**	**X**	**X**	**X**	**X**	**X**	**C**	**P**	0	0	0
5	5	5	5	4	4	4	4	4	4	4	4	4	4	4

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

```
void incr(int & tmp, int &gen)
{
   if (++tmp == SIZE) {
      tmp = 0;
      gen++;
   }
}
```




```
bool push(int val) {   entry ent;   int prev = 0;
 int gen = generation;  // laxtomic load
 int tmp = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp].load(relaxed), gen) )  {
        if (ent.gen < prev) return false;  // FULL
        else incr(tmp, gen);
        if (ent.data) prev = ent.gen;       }
 } while ( ! buffer[tmp].CAS(ent, entry{val,gen}) ); // NOT relaxed
 update(oldtail, tmp, oldgen, gen);
```

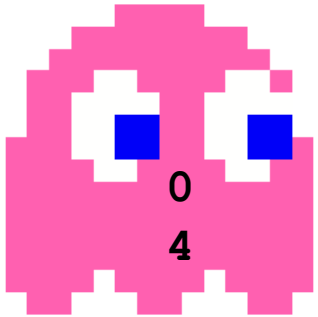
***Should* update generation & tail**

tmp

0	0	0	0	**X**	**X**	**X**	**X**	**X**	**X**	**C**	**P**	0	0	0
5	5	5	5	4	4	4	4	4	4	4	4	4	4	4

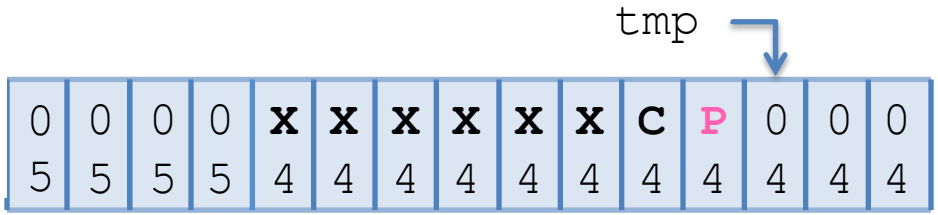
```
bool is_zero(entry e, int gen) {
   return e.data == 0 && e.gen == gen;
}
```

```
void incr(int & tmp, int &gen)
{
   if (++tmp == SIZE) {
      tmp = 0;
      gen++;
   }
}
```

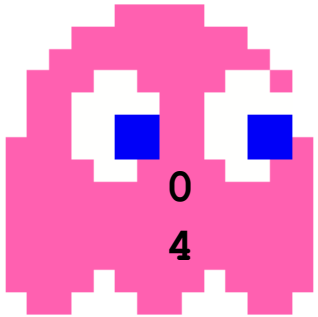



```
bool update(int oldtail, int tmp, oldgen, gen) {
   incr(tmp, gen); // we just filled a spot, go to next spot
   tail.CAS(oldtail, tmp);
   generation.CAS(oldgen, gen);
}
```

***Should* update generation & tail**

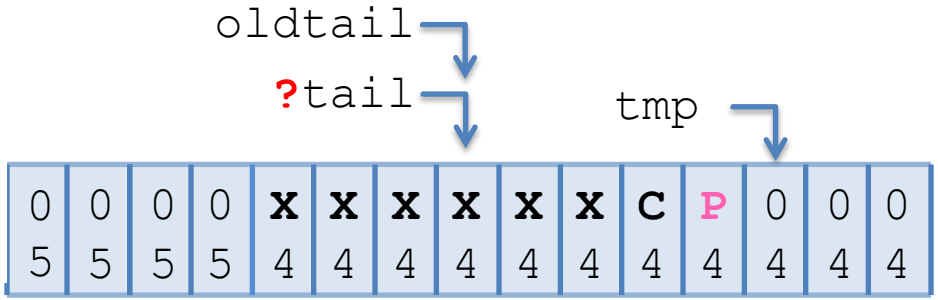


```
void incr(int & tmp, int &gen)
{
   if (++tmp == SIZE) {
      tmp = 0;
      gen++;
   }
}
```

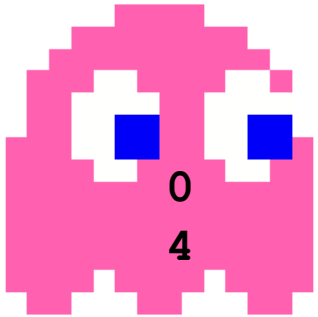



```
bool update(int oldtail, int tmp, oldgen, gen) {
   incr(tmp, gen); // we just filled a spot, go to next spot
   tail.CAS(oldtail, tmp);
   generation.CAS(oldgen, gen);
}
```

***Should* update generation & tail**

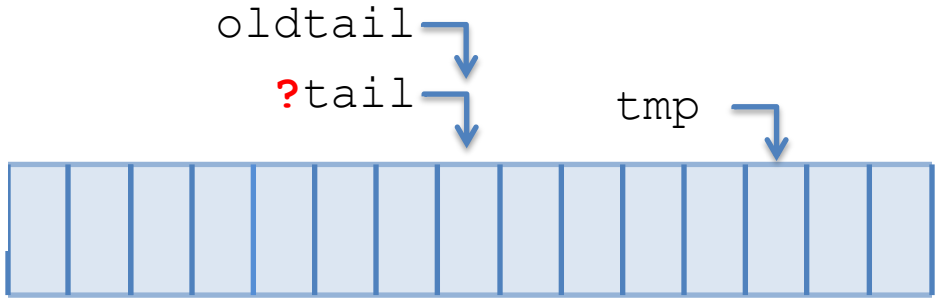


```
void incr(int & tmp, int &gen)
{
   if (++tmp == SIZE) {
      tmp = 0;
      gen++;
   }
}
```

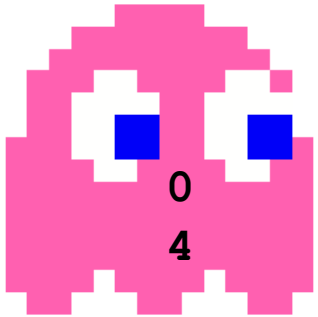



```
bool update(int oldtail, int tmp, oldgen, gen) {
   incr(tmp, gen); // we just filled a spot, go to next spot
   tail.CAS(oldtail, tmp);
   generation.CAS(oldgen, gen);
}
```

?

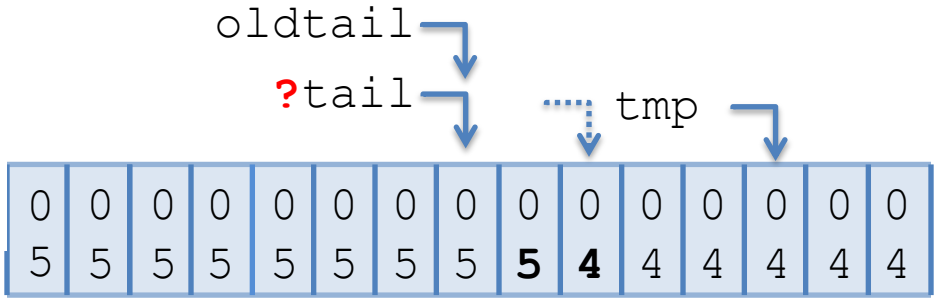


```
void incr(int & tmp, int &gen)
{
   if (++tmp == SIZE) {
      tmp = 0;
      gen++;
   }
}
```

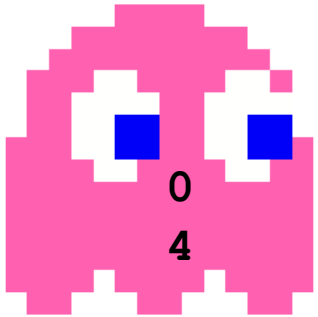



```
bool update(int oldtail, int tmp, oldgen, gen) {
   incr(tmp, gen); // we just filled a spot, go to next spot
   tail.CAS(oldtail, tmp);
   generation.CAS(oldgen, gen);
}
```

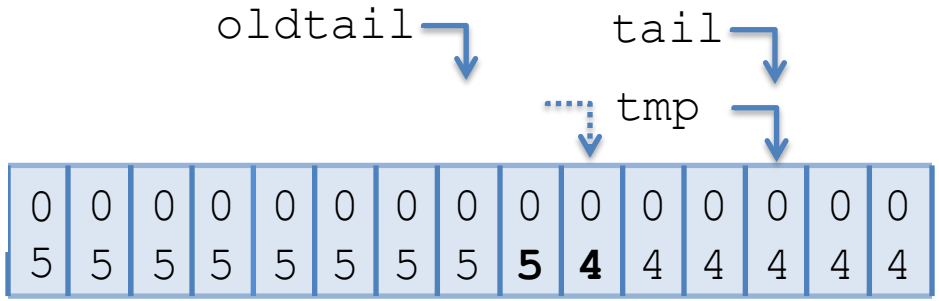
?



```
void incr(int & tmp, int &gen)
{
   if (++tmp == SIZE) {
      tmp = 0;
      gen++;
   }
}
```



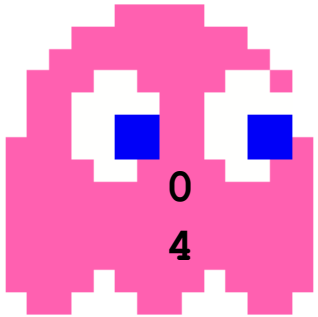

```
bool update(int oldtail, int tmp, oldgen, gen) {
   incr(tmp, gen); // we just filled a spot, go to next spot
   tail.CAS(oldtail, tmp);
   generation.CAS(oldgen, gen);
}
```



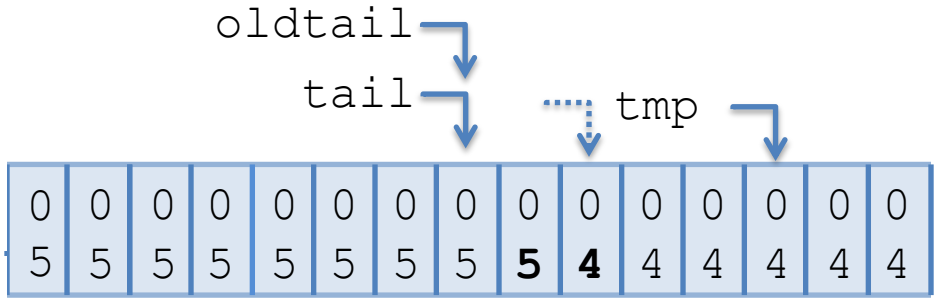
?

INVARIANT: tail@gen is "<=" real tail

```
void incr(int & tmp, int &gen)
{
   if (++tmp == SIZE) {
      tmp = 0;
      gen++;
   }
}
```



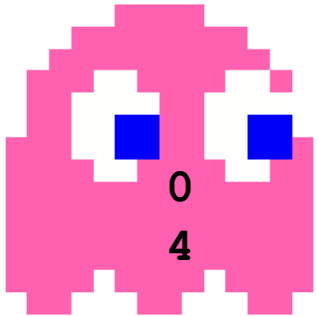

```
bool update(int oldtail, int tmp, oldgen, gen) {
   incr(tmp, gen); // we just filled a spot, go to next spot
   tail.CAS(oldtail, tmp);
   generation.CAS(oldgen, gen);
}
```



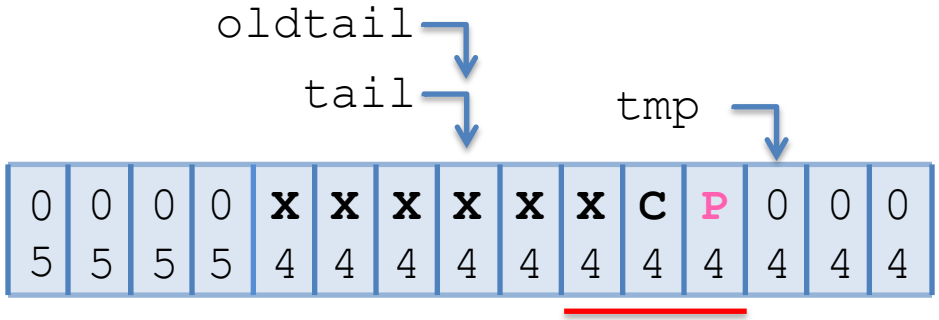
?

Can't happen

```
void incr(int & tmp, int &gen)
{
   if (++tmp == SIZE) {
      tmp = 0;
      gen++;
   }
}
```



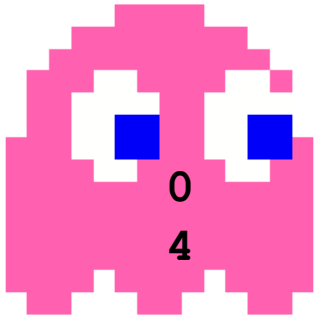

```
bool update(int oldtail, int tmp, oldgen, gen) {
   incr(tmp, gen); // we just filled a spot, go to next spot
   tail.CAS(oldtail, tmp);
   generation.CAS(oldgen, gen);
}
```



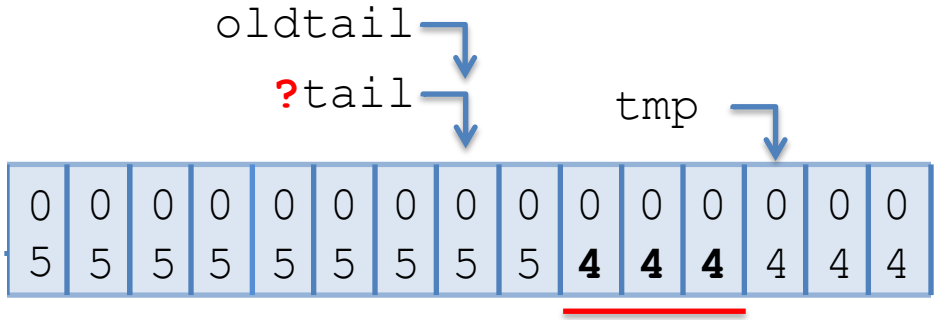
?

Can't happen

```
void incr(int & tmp, int &gen)
{
   if (++tmp == SIZE) {
      tmp = 0;
      gen++;
   }
}
```



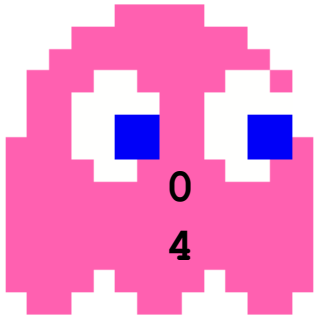

```
bool update(int oldtail, int tmp, oldgen, gen) {
   incr(tmp, gen); // we just filled a spot, go to next spot
   tail.CAS(oldtail, tmp);
   generation.CAS(oldgen, gen);
}
```



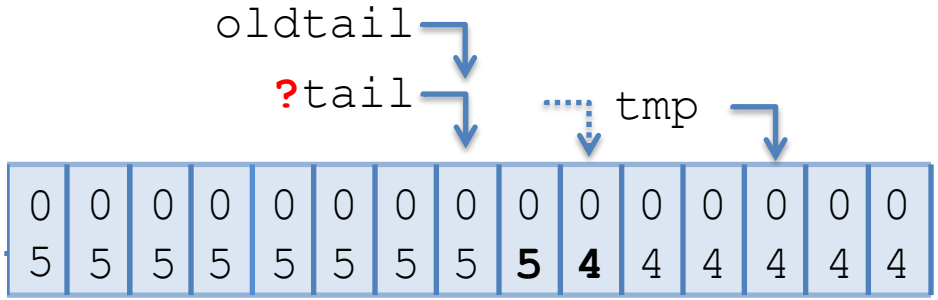
?

Can't happen

```
void incr(int & tmp, int &gen)
{
   if (++tmp == SIZE) {
      tmp = 0;
      gen++;
   }
}
```




```
bool update(int oldtail, int tmp, oldgen, gen) {
   incr(tmp, gen); // we just filled a spot, go to next spot
   tail.CAS(oldtail, tmp);
   generation.CAS(oldgen, gen);
}
```

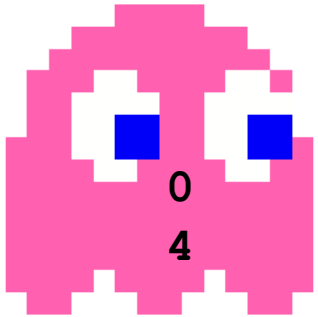


?

Can't happen

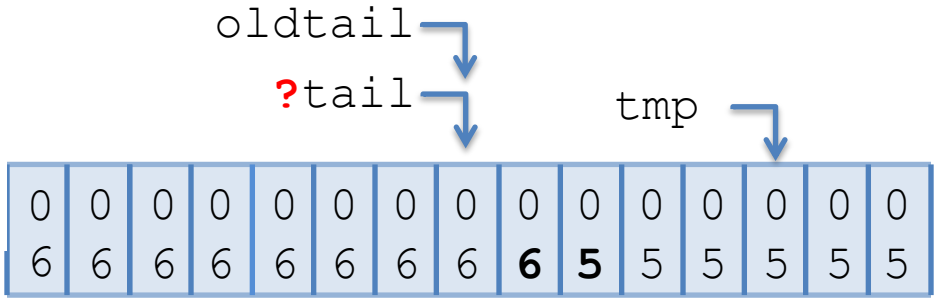
INVARIANT: tail@gen is "<=" real tail

```
void incr(int & tmp, int &gen)
{
   if (++tmp == SIZE) {
      tmp = 0;
      gen++;
   }
}
```

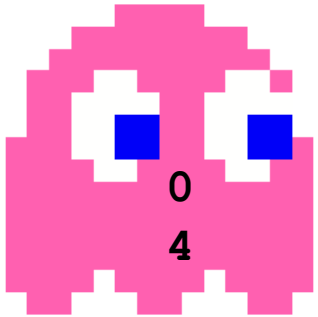



```
bool update(int oldtail, int tmp, oldgen, gen) {
   incr(tmp, gen); // we just filled a spot, go to next spot
   tail.CAS(oldtail, tmp);
   generation.CAS(oldgen, gen);
}
```

?

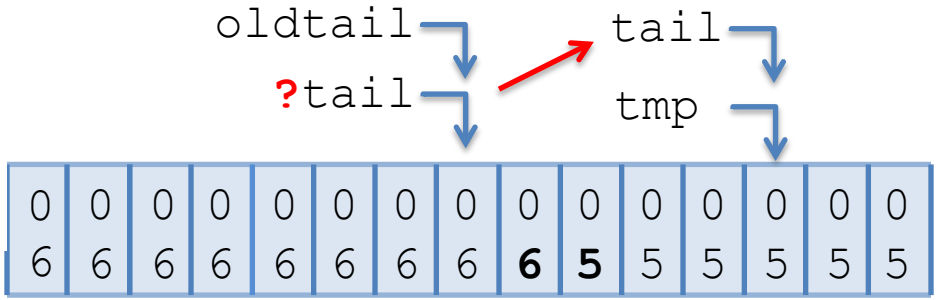



```
void incr(int & tmp, int &gen)
{
   if (++tmp == SIZE) {
      tmp = 0;
      gen++;
   }
}
```



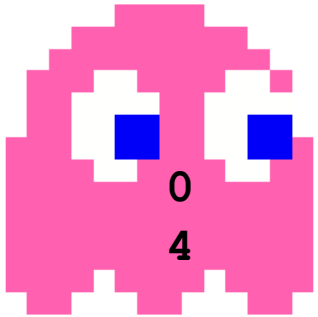

```
bool update(int oldtail, int tmp, oldgen, gen) {
   incr(tmp, gen); // we just filled a spot, go to next spot
   tail.CAS(oldtail, tmp);
   generation.CAS(oldgen, gen);
}
```

?



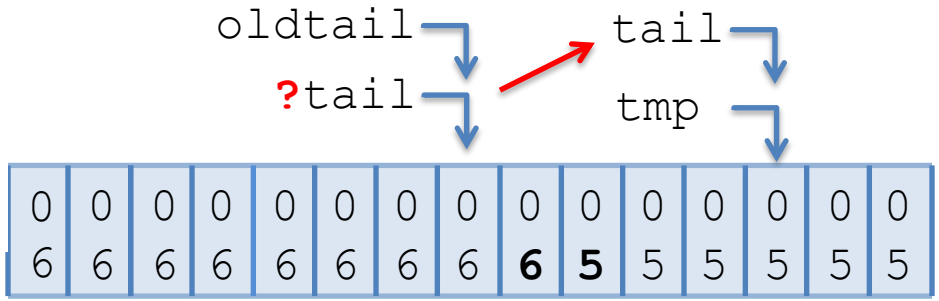
INVARIANT: tail@gen is "<=" real tail

```
void incr(int & tmp, int &gen)
{
   if (++tmp == SIZE) {
      tmp = 0;
      gen++;
   }
}
```




```
bool update(int oldtail, int tmp, oldgen, gen) {
   incr(tmp, gen); // we just filled a spot, go to next spot
   tail.CAS(oldtail, tmp);
   generation.CAS(oldgen, gen);
}
```

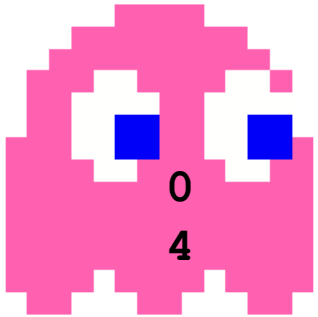
?



ABA

```
void incr(int & tmp, int &gen)
{
   if (++tmp == SIZE) {
      tmp = 0;
      gen++;
   }
}
```

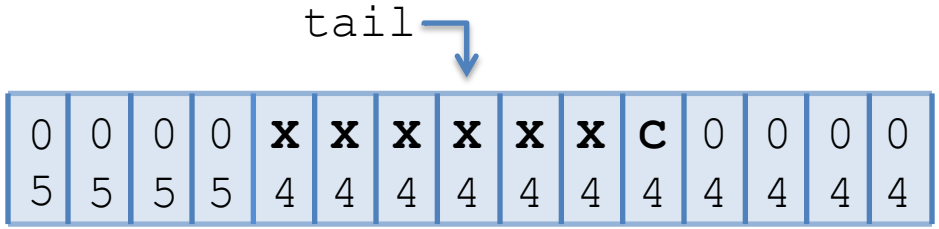
INVARIANT: tail@gen is "<=" real tail




```
bool update(int oldtail, int tmp, oldgen, gen) {
   incr(tmp, gen); // we just filled a spot, go to next spot
   tail.CAS(oldtail, tmp);
   generation.CAS(oldgen, gen);
}
```

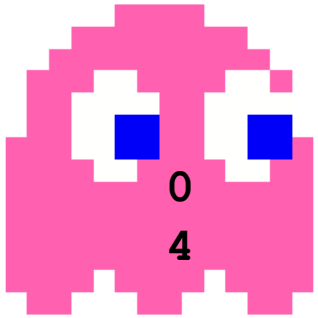
ABA

?



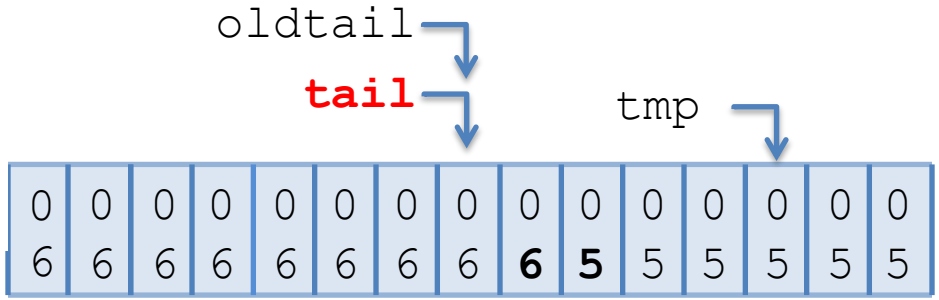
INVARIANT: tail@gen is "<=" real tail

```
void incr(int & tmp, int &gen)
{
   if (++tmp == SIZE) {
      tmp = 0;
      gen++;
   }
}
```




```
bool update(int oldtail, int tmp, oldgen, gen) {
   incr(tmp, gen); // we just filled a spot, go to next spot
   tail.CAS(oldtail, tmp);
   generation.CAS(oldgen, gen);
}
```

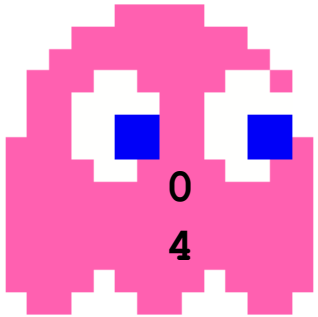
?



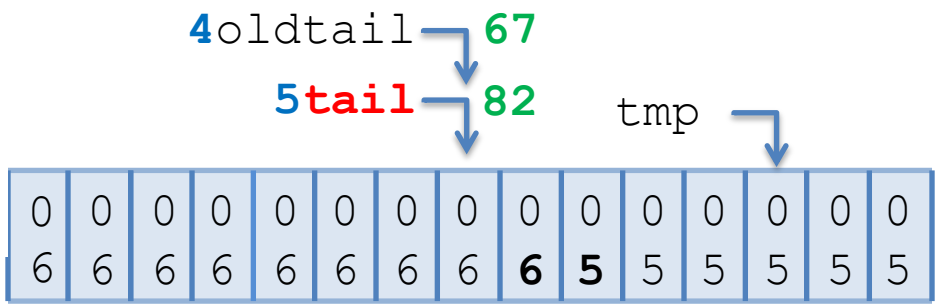
ABA

INVARIANT: tail@gen is "<=" real tail

```
void incr(int & tmp, int &gen)
{
   if (++tmp == SIZE) {
      tmp = 0;
      gen++;
   }
}
```




```
bool update(int oldtail, int tmp, oldgen, gen) {
   incr(tmp, gen); // we just filled a spot, go to next spot
   tail.CAS(oldtail, tmp);
   generation.CAS(oldgen, gen);
}
```

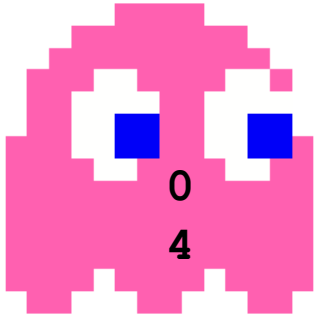


?

- forever incr tail
or
- merge tail + gen

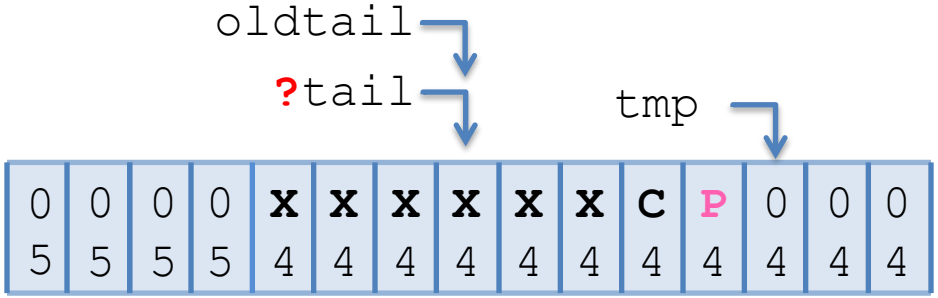
```
void incr(int & tmp, int &gen)
{
   if (++tmp % SIZE == 0)
      gen++;
}

…buffer[tmp%SIZE]…
```

INVARIANT: tail@gen is "<=" real tail

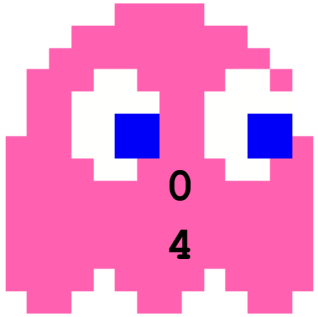



```
bool update(int oldtail, int tmp, oldgen, gen) {
   incr(tmp, gen); // we just filled a spot, go to next spot
   tail.CAS(oldtail, tmp);
   generation.CAS(oldgen, gen);
}
```

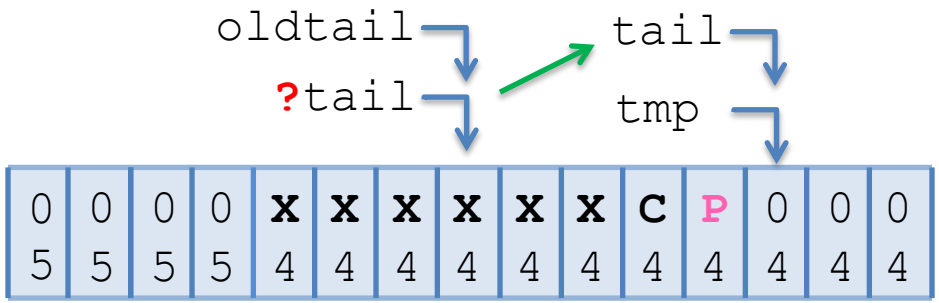
?



- forever incr tail
or
- merge tail + gen

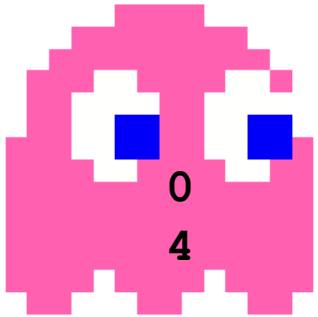



```
bool update(int oldtail, int tmp, oldgen, gen) {
   incr(tmp, gen); // we just filled a spot, go to next spot
   tail.CAS(oldtail, tmp);
   generation.CAS(oldgen, gen);
}
```



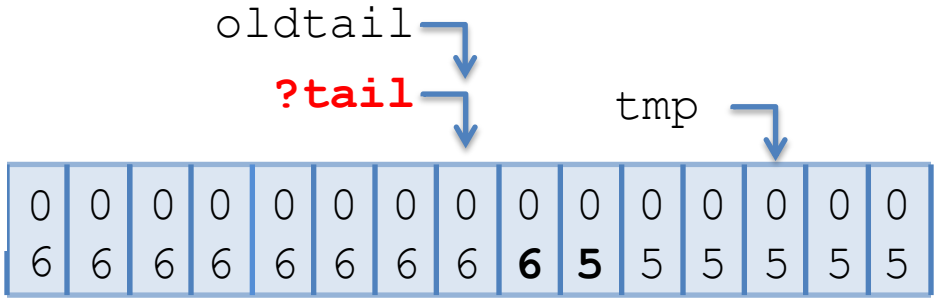
- forever incr tail
or
- merge tail + gen

✓

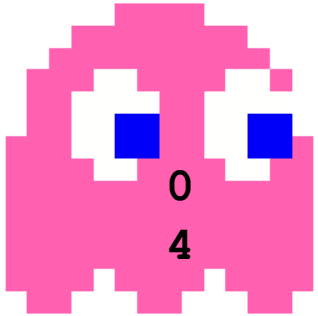



```
bool update(int oldtail, int tmp, oldgen, gen) {
   incr(tmp, gen); // we just filled a spot, go to next spot
   tail.CAS(oldtail, tmp);
   generation.CAS(oldgen, gen);
}
```

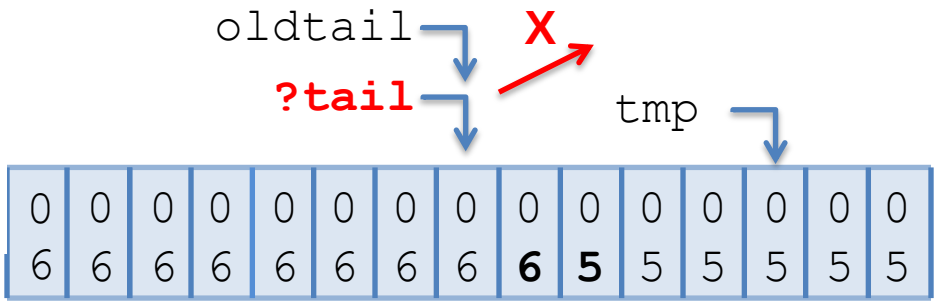
?



- forever incr tail
or
- merge tail + gen

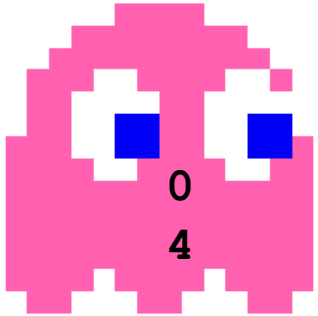



```
bool update(int oldtail, int tmp, oldgen, gen) {
   incr(tmp, gen); // we just filled a spot, go to next spot
   tail.CAS(oldtail, tmp);
   generation.CAS(oldgen, gen);
}
```



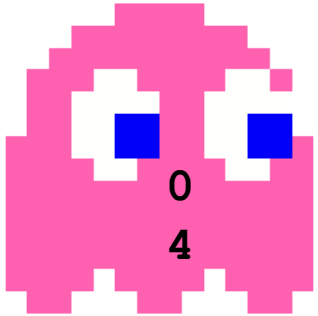
- forever incr tail
or
- merge tail + gen

✓

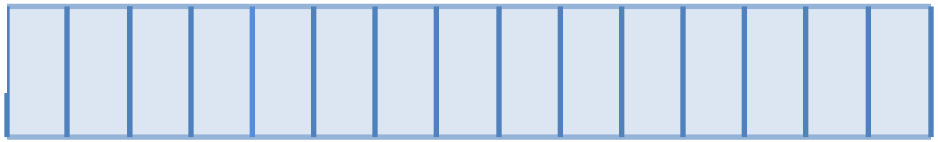



```
bool update(int oldtail, int tmp, oldgen, gen) {
   incr(tmp, gen); // we just filled a spot, go to next spot
   tail.CAS(oldtail, tmp);
   generation.CAS(oldgen, gen);
}
```

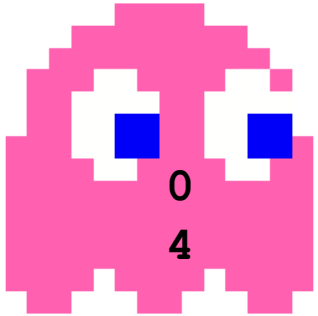
?




```
bool update(int oldtail, int tmp, oldgen, gen) {
   incr(tmp, gen); // we just filled a spot, go to next spot
   tail.CAS(oldtail, tmp);
   generation.CAS(oldgen, gen);
}
```



```
void incr(int & tmp, int &gen)
{
   if (++tmp …)
      …
      gen++;
}
```



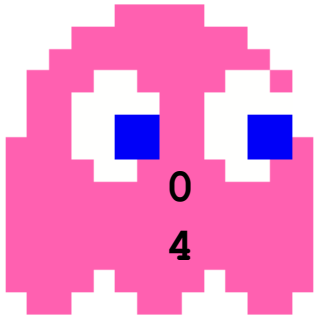

```
bool update(int oldtail, int tmp, oldgen, gen) {
   incr(tmp, gen); // we just filled a spot, go to next spot
   tail.CAS(oldtail, tmp);
   while (!generation.CAS(oldgen, gen) && oldgen < gen)
        ;
}
```

generation gen

2 3 → 5

✓

```
void incr(int & tmp, int &gen)
{
   if (++tmp …)
        …
        gen++;
}
```

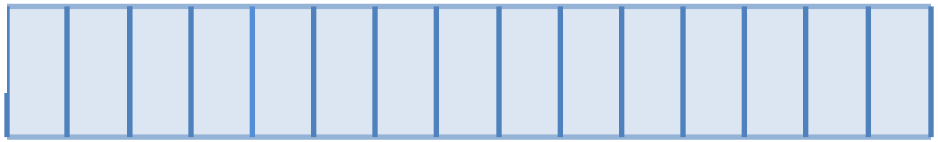



```
bool update(int oldtail, int tmp, oldgen, gen) {
   incr(tmp, gen); // we just filled a spot, go to next spot
   tail.CAS(oldtail, tmp);
   while (!generation.CAS(oldgen, gen) && oldgen < gen)
        ;
}
```

generation gen

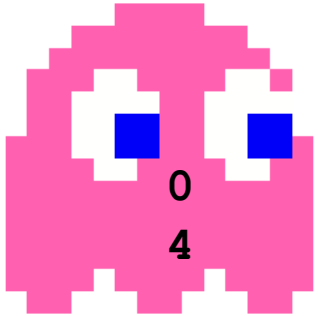
2 3 → 5

✓

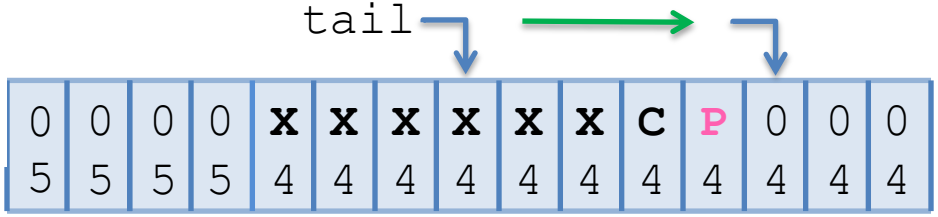


```
void incr(int & tmp, int &gen)
{
   if (++tmp …)
        …
        gen++;
}
```

INVARIANT: tail@gen is "<=" real tail




```
bool push(int val) {   entry ent;   int prev = 0;
 int gen = oldgen = generation;  // laxtomic load
 int tmp = oldtail = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp%SIZE].load(relaxed), gen) ) {
        if (ent.gen < prev) return false;  // FULL
        else incr(tmp, gen);
        if (ent.data) prev = ent.gen;       }
 } while ( ! buffer[tmp%SIZE].CAS(ent, entry{val,gen}, release) );
 update(oldtail, tmp, oldgen, gen);  return true; }  // laxtomic
```

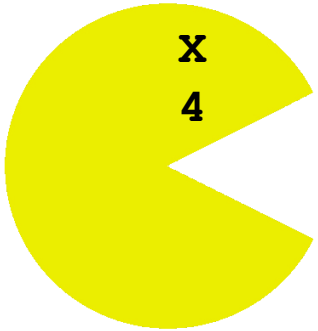


forever incr tail

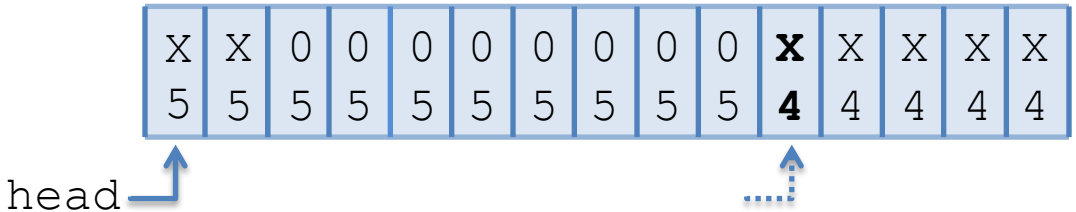
```
void update(int oldtail, int tmp, oldgen, gen) {
   incr(tmp, gen);
   tail.CAS(oldtail, tmp);
   generation.CAS(oldgen, gen);
}
```

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

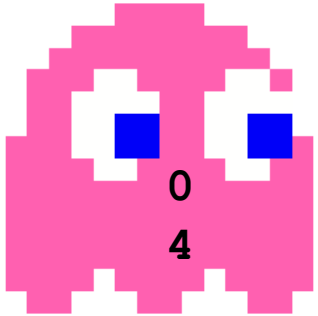
```
void incr(int & tmp, int &gen)
{
   if (++tmp % SIZE == 0)
      gen++;
}
```




```
int pop() {




}
```



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



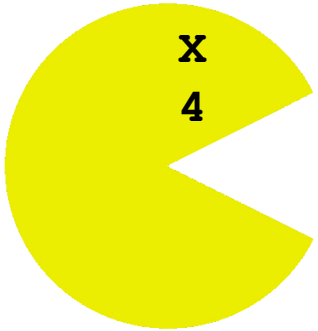

```
bool push(int val) {   entry ent;   int prev = 0;
 int gen = oldgen = generation;  // laxtomic load
 int tmp = oldtail = tail;  // laxtomic load
 do {
    while( ! is_zero(ent = buffer[tmp%SIZE].load(relaxed), gen) ) {
        if (ent.gen < prev) return false;  // FULL
        else incr(tmp, gen);
        if (ent.data) prev = ent.gen;       }
 } while ( ! buffer[tmp%SIZE].CAS(ent, entry{val,gen}, release) );
 update(oldtail, tmp, oldgen, gen); // laxtomic
 return true; }
```

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

head

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

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




```
int pop() {   entry ent;
 int gen = oldgen = generation;  // laxtomic load
 int tmp = oldhead = head;  // laxtomic load
 do {
    while( ! is_data(ent = buffer[tmp%SIZE].load(relaxed), gen) )
       if (ent.gen == gen) return 0;  // EMPTY
       else incr(tmp, gen);

 } while ( ! buffer[tmp%SIZE].CAS(ent, entry{0,gen+1}, acquire) );
 update_head(oldhead, tmp, oldgen, gen); // laxtomic
 return ent.data; }
```

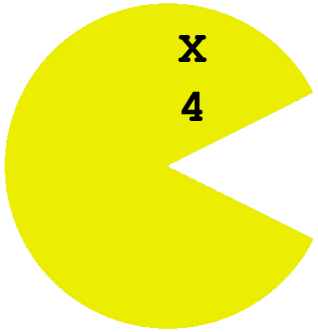
✓ ?

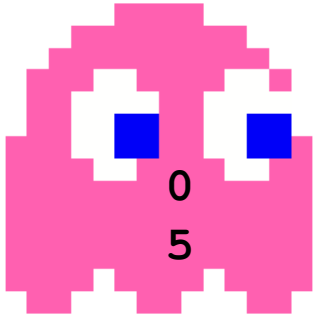
X	X	0	0	0	0	0	0	0	0	**X**	X	X	X	X
5	5	5	5	5	5	5	5	5	5	**4**	4	4	4	4

head

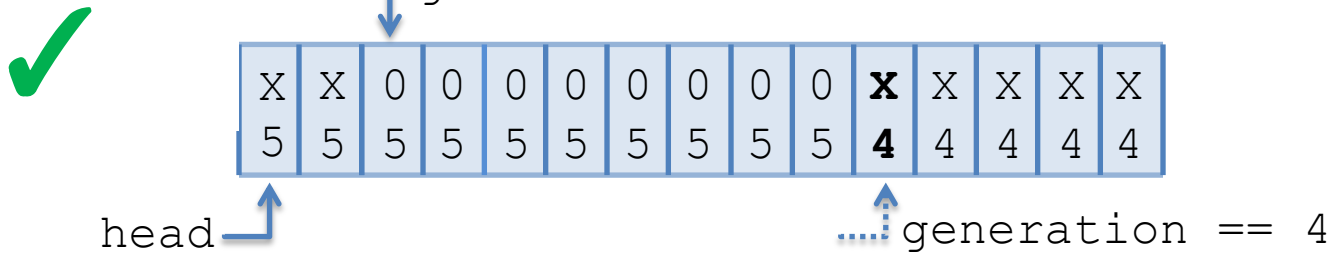
First non-zero data (**of correct generation**)

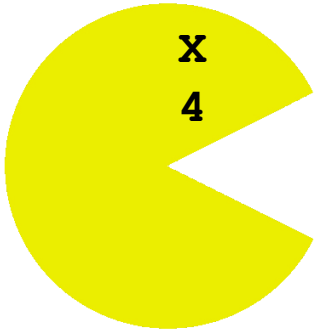
```
bool is_data(entry e, int gen) {
  return      e.data != 0
           && e.gen == gen;
}
```

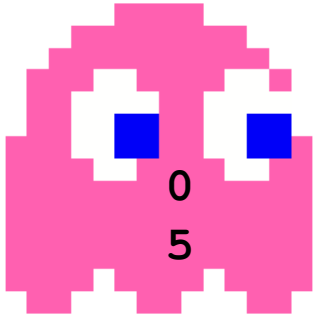





```
class Queue {
 atomic<entry> buffer[SIZE];
 laxtomic<int> head;
 laxtomic<int> h_generation;
 laxtomic<int> tail;
 laxtomic<int> t_generation;
};
```

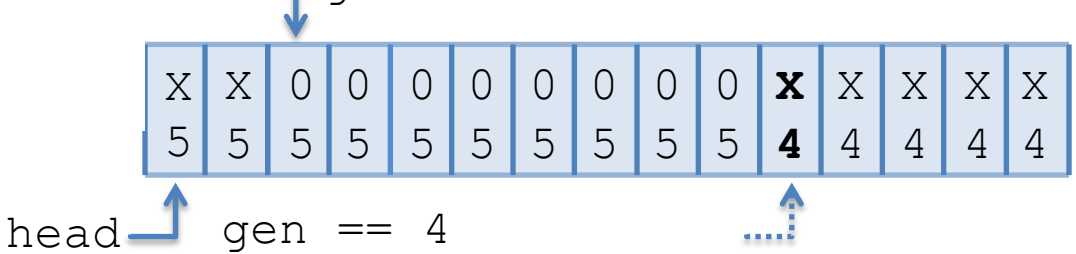







```
class Queue {
 atomic<geni> buffer[SIZE];
 laxtomic<geni> head;
 laxtomic<geni> tail;
};
```

```
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; }
};
```




```
void update(laxtomic<geni>&target, geni old, geni tmp) {
   tmp.incr(); // go to next
   while (!tail.CAS(old, tmp) && old < tmp) { }     }
```

```
bool push(int val)
{ int prev = 0;
 geni ent;
 geni tmp;
 geni old = tmp = tail;  // laxtomic load
 do {
    ent = buffer[tmp].load(relaxed);
    while( ! is_zero(ent, tmp.gen) ) {
        if (ent.gen < prev) return false; // full
        else tmp.incr();
        if (ent.data) prev = ent.gen;  }
    geni newg{val, tmp.gen};
 } while ( ! buffer[tmp].CAS(ent, newg, release));
 tmp.incr(); // go to next
 // update if no one else has gone as far:
 while (!tail.CAS(old, tmp) && old < tmp)  { }
return true;
}
```

```
int pop()
{
 geni ent;
 geni tmp;
 geni old = tmp = head;  // laxtomic load
 do {
    ent = buffer[tmp].load(relaxed);
    while( ! ent.is_data(tmp.gen) )
        if (ent.gen == tmp.gen) return 0; // empty
        else tmp.incr();

    geni zero{0, tmp.gen+1};
 } while ( ! buffer[tmp].CAS(ent, zero, acquire));
 tmp.incr(); // go to next
 // update if no one else has gone as far:
  while (!head.CAS(old, tmp) && old < tmp)  { }
  return ent.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

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

MORE?

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

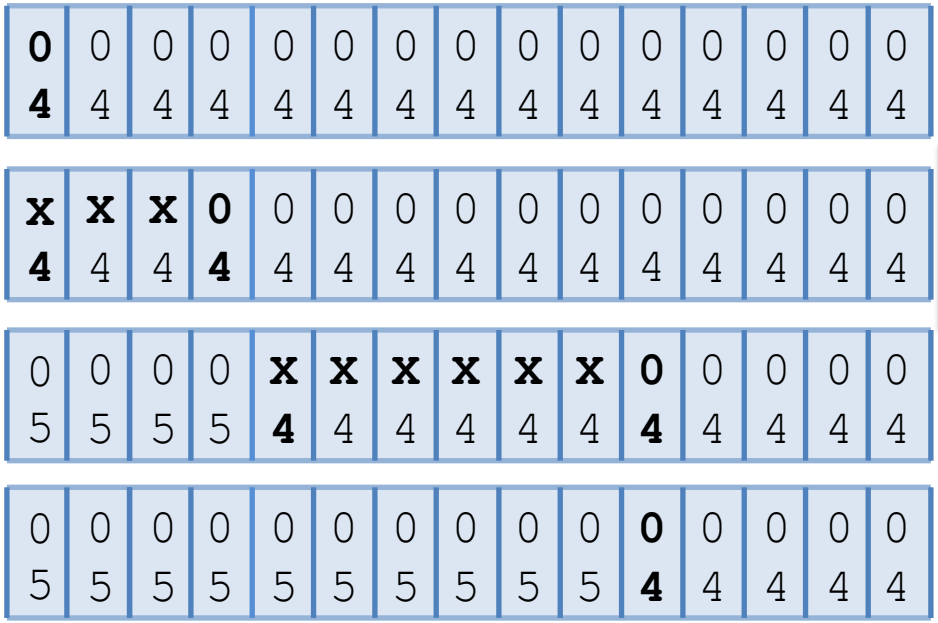
0	0	0	0	**x**	**x**	**x**	**x**	**x**	**x**	**0**	0	0	0	0
5	5	5	5	**4**	4	4	4	4	4	**4**	4	4	4	4

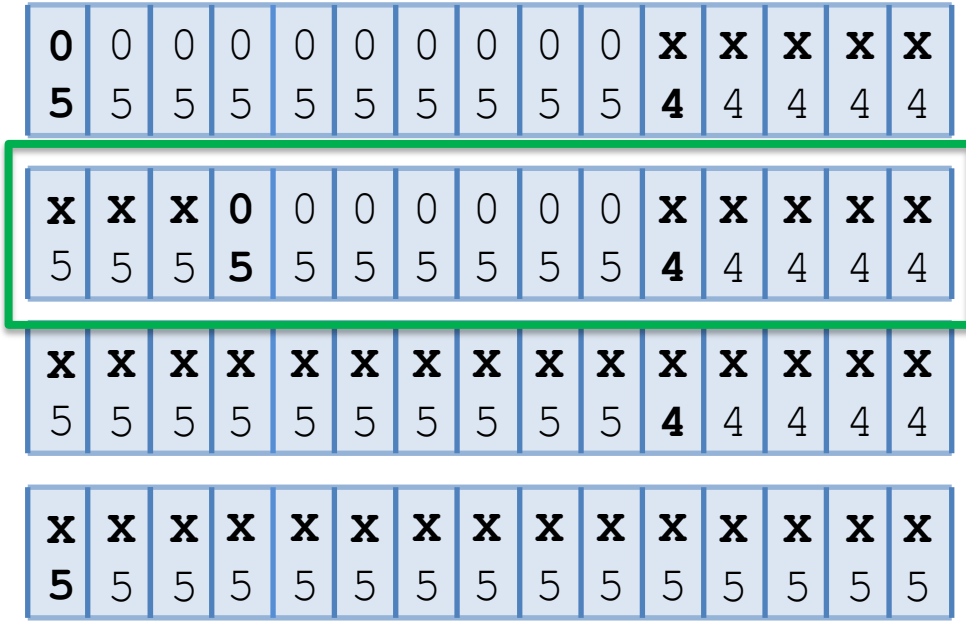
x	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**
5	5	5	5	5	5	5	5	5	5	**4**	4	4	4	4

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

LESS?





LESS?

(Algorithm can't see more than one spot at a time!)

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

LESS?

(Algorithm can't see more than one spot at a time!)

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

So really…

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

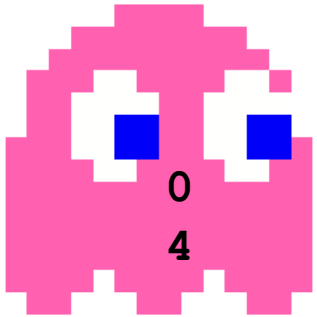
x	**x**	**x**	**0**	0	0	0	0	0	0	**x**	**x**	**x**	**x**	**x**
5	5	5	**5**	5	5	5	5	5	5	**4**	4	4	4	4

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



4tail

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

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

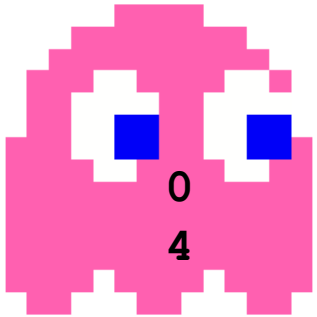
x	**x**	**x**	**0**	0	0	0	0	0	0	**x**	**x**	**x**	**x**	**x**
5	5	5	**5**	5	5	5	5	5	5	**4**	4	4	4	4

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

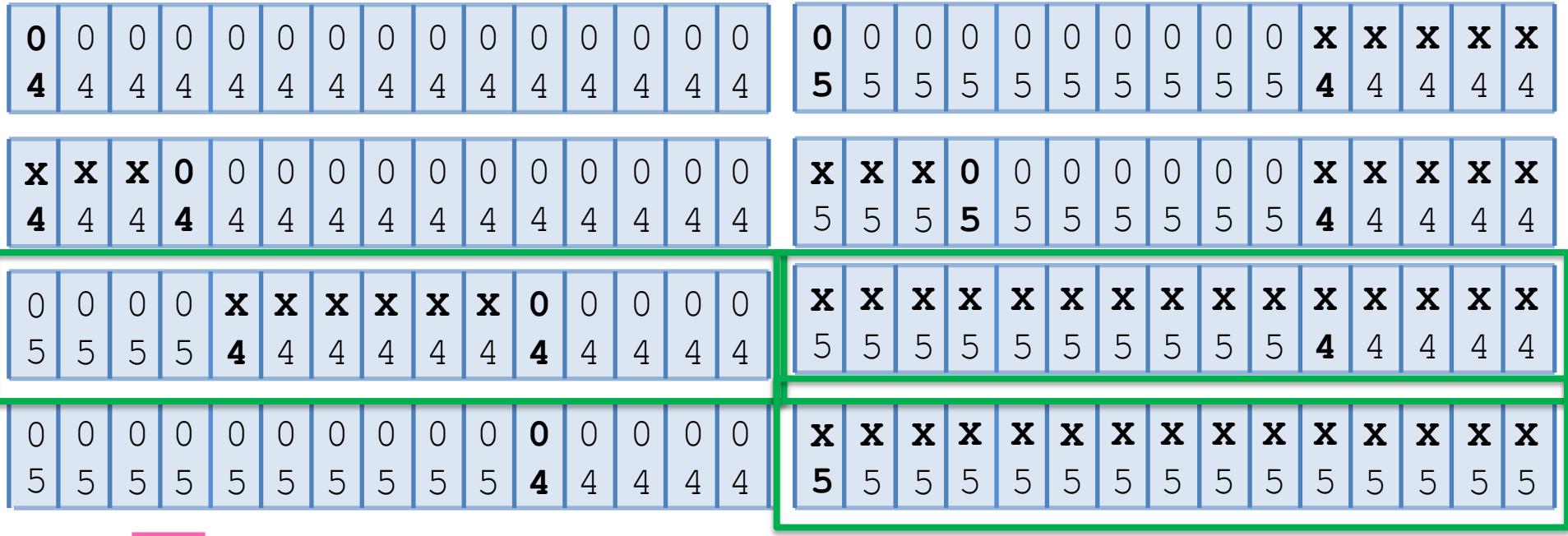
0	0	0	0	0	0	0	0	0	0	**0**	0	0	0	0
5	5	5	5	5	5	5	5	5	5	**4**	4	4	4	4

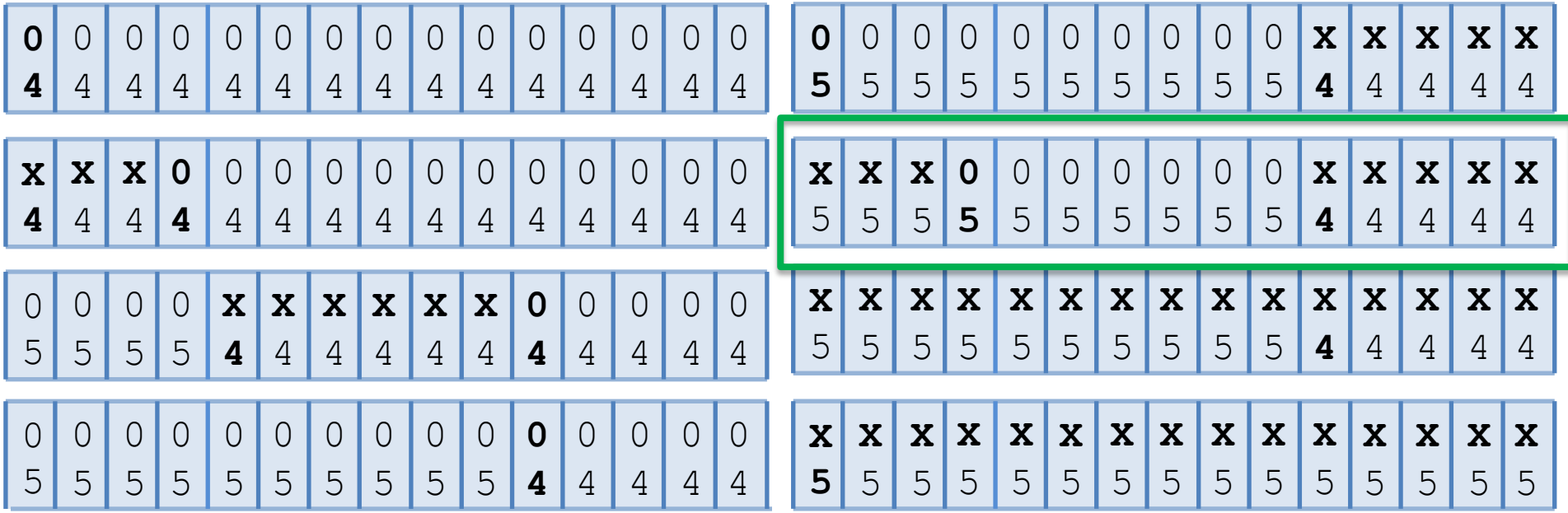
x	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5



4tail

							x							
							4							


0 4
0 5
x 4
x x x 0
x x x 0 5
x 5
4tail
x 4
0 4


0 4
0 5
x x x 0
x x x 0 5
0 5 x 4
x 5 4
0 5 0 4
x 5 0 4
4tail
0
4

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

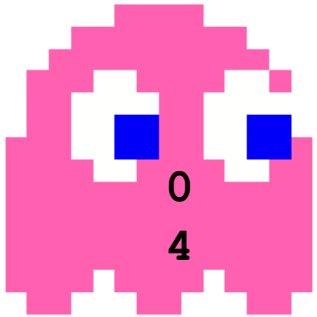
x	**x**	**x**	**0**	0	0	0	0	0	0	**x**	**x**	**x**	**x**	**x**
5	5	5	**5**	5	5	5	5	5	5	**4**	4	4	4	4

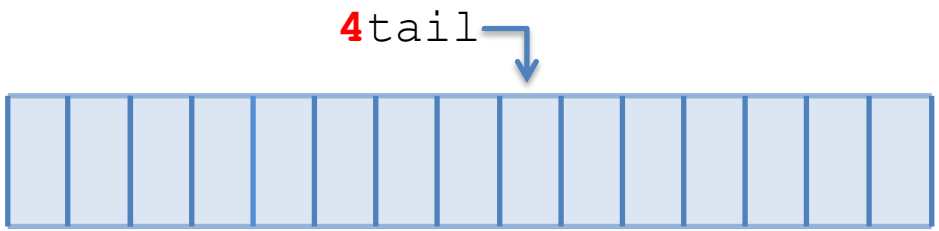
0	0	0	0	**x**	**x**	**x**	**x**	**x**	**x**	**0**	0	0	0	0
5	5	5	5	**4**	4	4	4	4	4	**4**	4	4	4	4

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





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	**x**	**x**	**x**	**x**	**x**
4	4	4	**4**	4	4	4	4	4	4	4	4	4	4	4

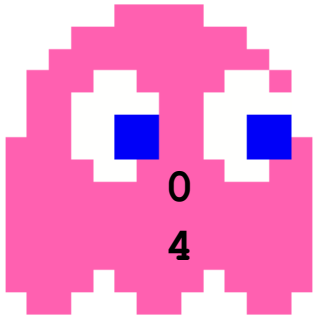
x	**x**	**x**	**0**	0	0	0	0	0	0	**x**	**x**	**x**	**x**	**x**
5	5	5	**5**	5	5	5	5	5	5	**4**	4	4	4	4

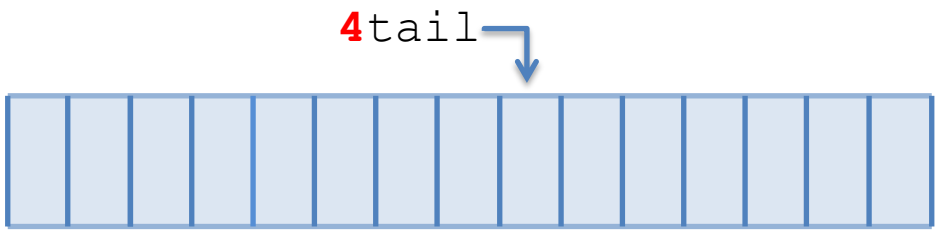
0	0	0	0	**x**	**x**	**x**	**x**	**x**	**x**	**0**	0	0	0	0
5	5	5	5	**4**	4	4	4	4	4	**4**	4	4	4	4

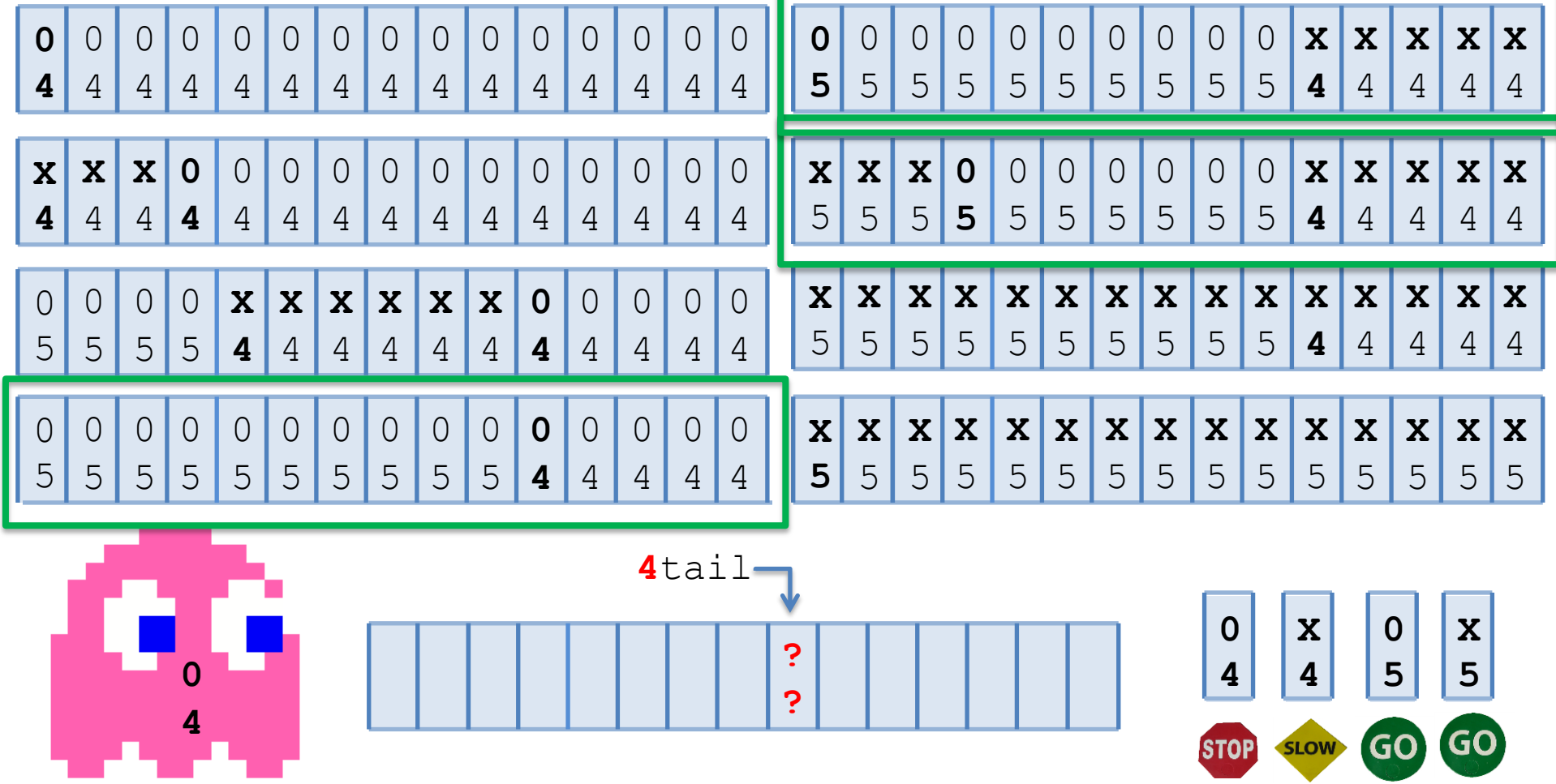
x	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**
5	5	5	5	5	5	5	5	5	5	**4**	4	4	4	4

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






4tail
?
?
STOP
SLOW
GO
GO

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

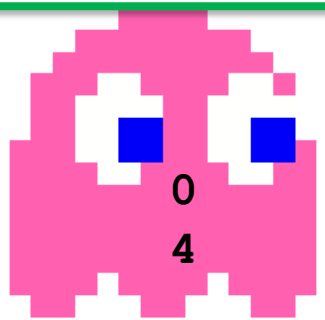
X	**X**	**X**	**0**	0	0	0	0	0	0	**X**	**X**	**X**	**X**	**X**
4	4	4	**4**	4	4	4	4	4	4	4	4	4	4	4

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	**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

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

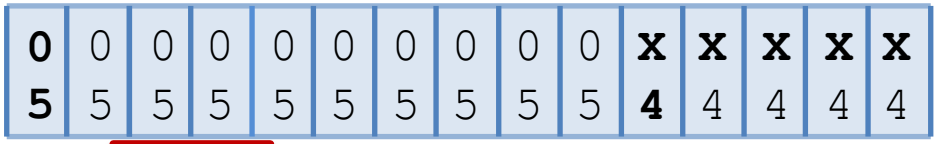
X	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5

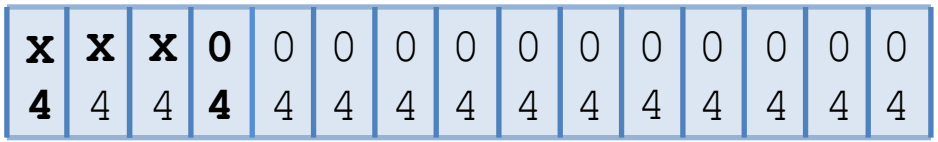


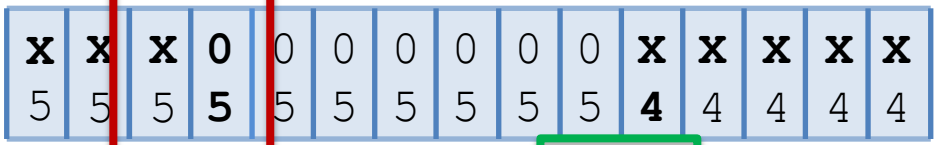
4tail

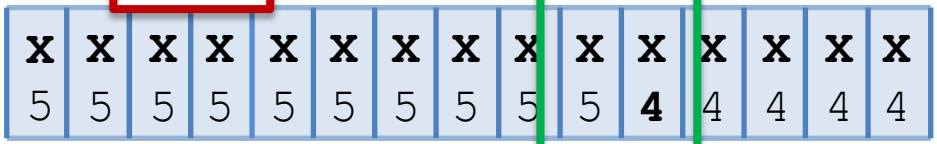
								?						
								?						

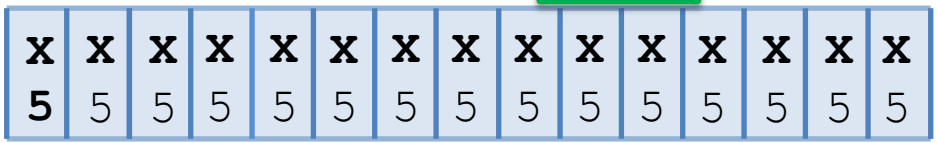
0	X	0	X
4	4	5	5



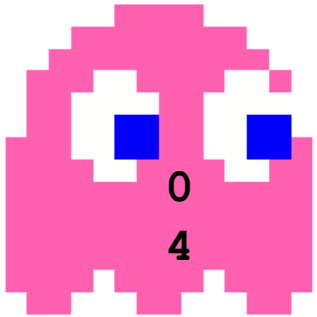


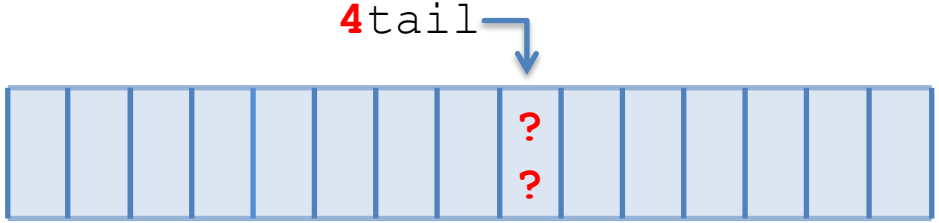


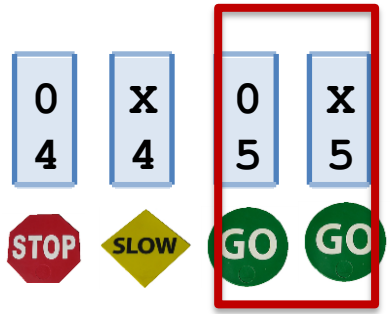





```
while( ! is_zero(ent, tmp.gen) ) {
    if (ent.gen < prev) return false; // full
    else tmp.incr();
    if (ent.data) prev = ent.gen;  }
```



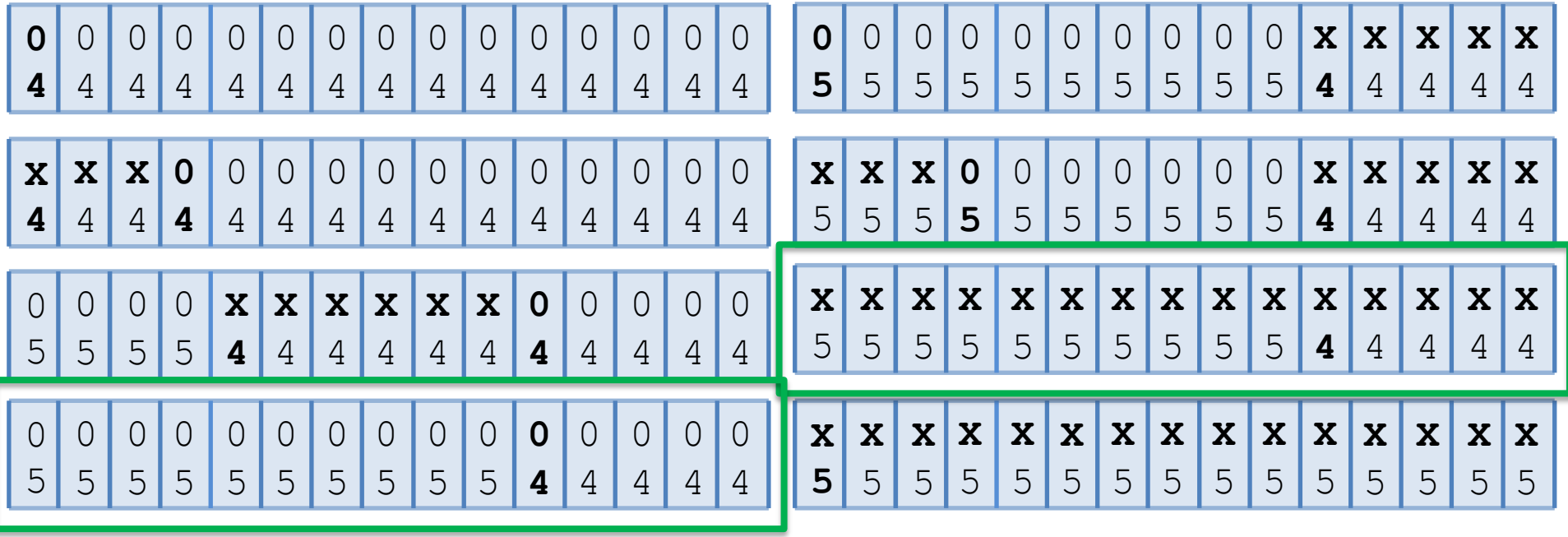




```
bool push(int val)
{ int prev = 0;
 geni ent;
 geni tmp;
 geni old = tmp = tail;  // laxtomic load
 do {
    ent = buffer[tmp].load(relaxed);
    while( ! is_zero(ent, tmp.gen) ) {
       if (ent.gen < prev) return false; // full
       else tmp.incr();
       if (ent.data) prev = ent.gen; }
    geni newg{val, tmp.gen};
 } while ( ! buffer[tmp].CAS(ent, newg, release));
 tmp.incr(); // go to next
 // update if no one else has gone as far:
 while (!tail.CAS(old, tmp) && old < tmp)  { }
return true;
}
```

```
int pop()
{
 geni ent;
 geni tmp;
 geni old = tmp = head;  // laxtomic load
 do {
    ent = buffer[tmp].load(relaxed);
    while( ! ent.is_data(tmp.gen) )
       if (ent.gen == tmp.gen) return 0; // empty
       else tmp.incr();
    geni zero{0, tmp.gen+1};
 } while ( ! buffer[tmp].CAS(ent, zero, acquire));
 tmp.incr(); // go to next
 // update if no one else has gone as far:
  while (!head.CAS(old, tmp) && old < tmp)  { }
  return ent.val;
}
```

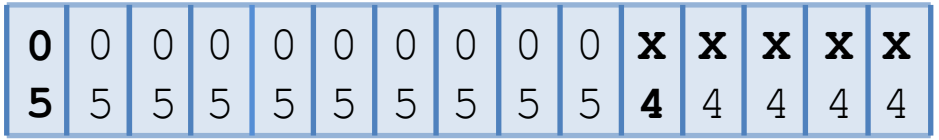
All states are valid states for all lines of code!?

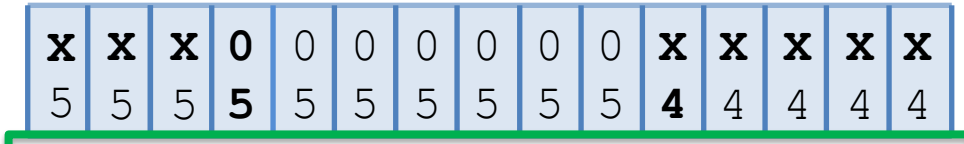


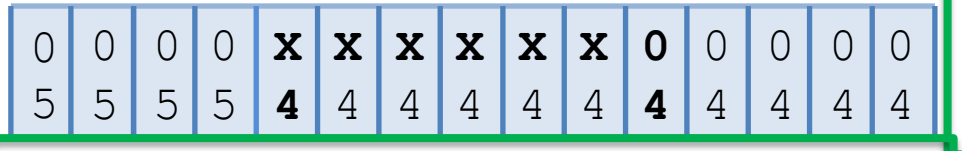
if (ent.gen **==** tmp.gen) return **0**; // empty

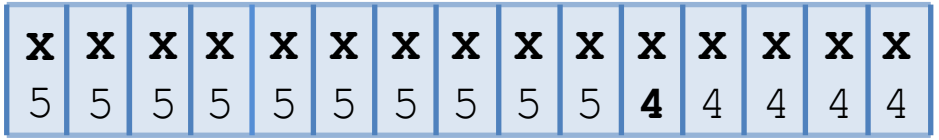
if (ent.gen **<** prev) return **false**; // full

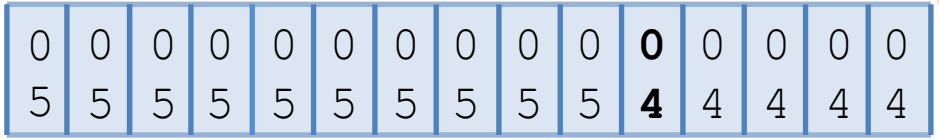
Update tail before leaving?

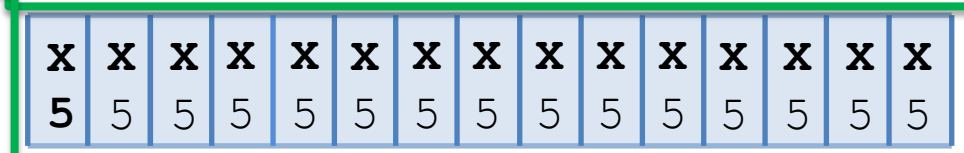













```
if (ent.gen == tmp.gen) {
   // update if no one else has gone as far:
   while (!tail.CAS(old, tmp) && old < tmp)  { }
   return 0; // empty
}
```

```
if (ent.gen < prev) {
   // update if no one else has gone as far:
   while (!head.CAS(old, tmp) && old < tmp)  { }
   return false; // full
}
```

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

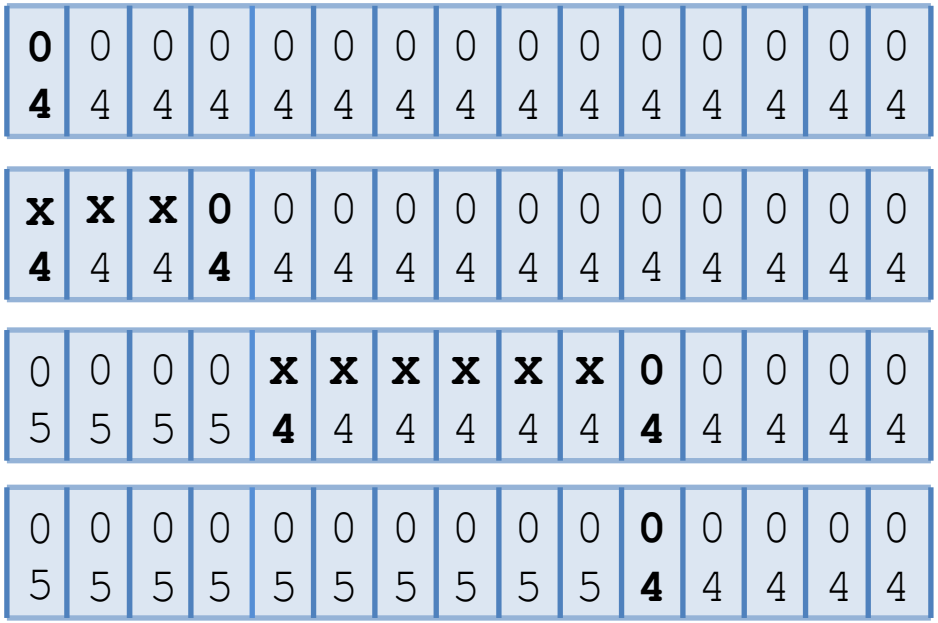
0	0	0	0	**x**	**x**	**x**	**x**	**x**	**x**	**0**	0	0	0	0
5	5	5	5	**4**	4	4	4	4	4	**4**	4	4	4	4

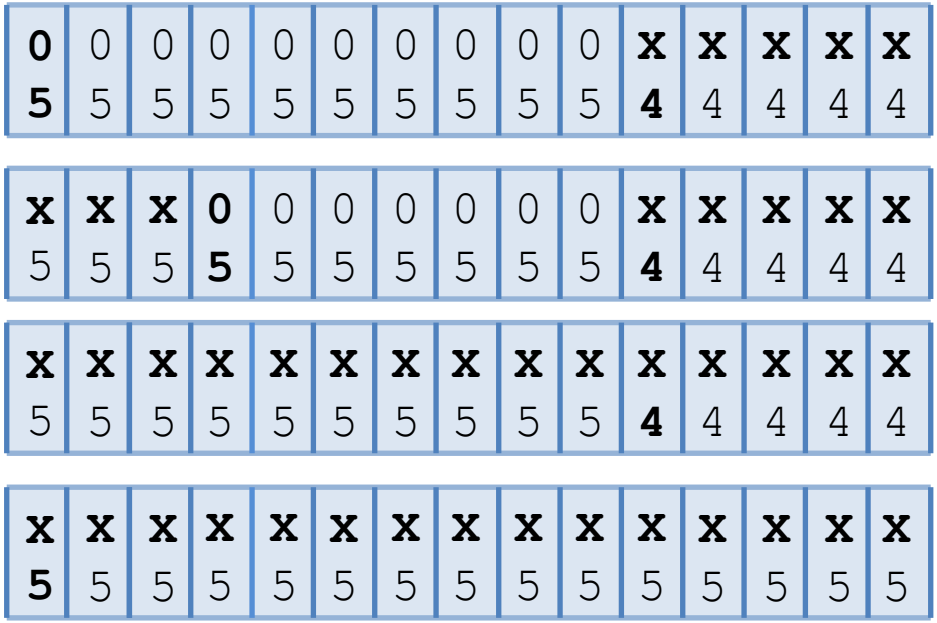
x	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**
5	5	5	5	5	5	5	5	5	5	**4**	4	4	4	4

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

INVARIANT: tail@gen is “<=“ real tail





tail <= real tail
head <= real head
range is circularly contiguous

tail only increases, adds data
head only increases, when data
∴ head <= tail ?
ordering guarantees?

```
do {
   ent = buffer[tmp].load(relaxed);
   while( ! is_zero(ent, tmp.gen) ) {
      if (ent.gen < prev) return false; // full
      else tmp.incr();
      if (ent.data) prev = ent.gen;  }
   geni newg{val, tmp.gen};
} while ( ! buffer[tmp].CAS(ent, newg, release));
tmp.incr(); // go to next
// update if no one else has gone as far:
while (!tail.CAS(old, tmp) && old < tmp)  { }
return true;
```

```
do {
   ent = buffer[tmp].load(relaxed);
   while( ! ent.is_data(tmp.gen) )
      if (ent.gen == tmp.gen) return 0; // empty
      else tmp.incr();

   geni zero{0, tmp.gen+1};
} while ( ! buffer[tmp].CAS(ent, zero, acquire));
tmp.incr(); // go to next
// update if no one else has gone as far:
 while (!head.CAS(old, tmp) && old < tmp)  { }
 return ent.val;
```

tail <= real tail

head <= real head

range is circularly contiguous?

tail only increases, adds data

head only increases, when data

∴ head <= tail ?

ordering guarantees?

Looking Back

push()

Looking Back

push()
pop()

Looking Back

push()
pop()
(started invariants)

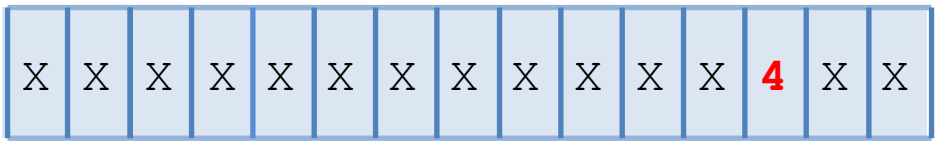
“The Problem with Threads”

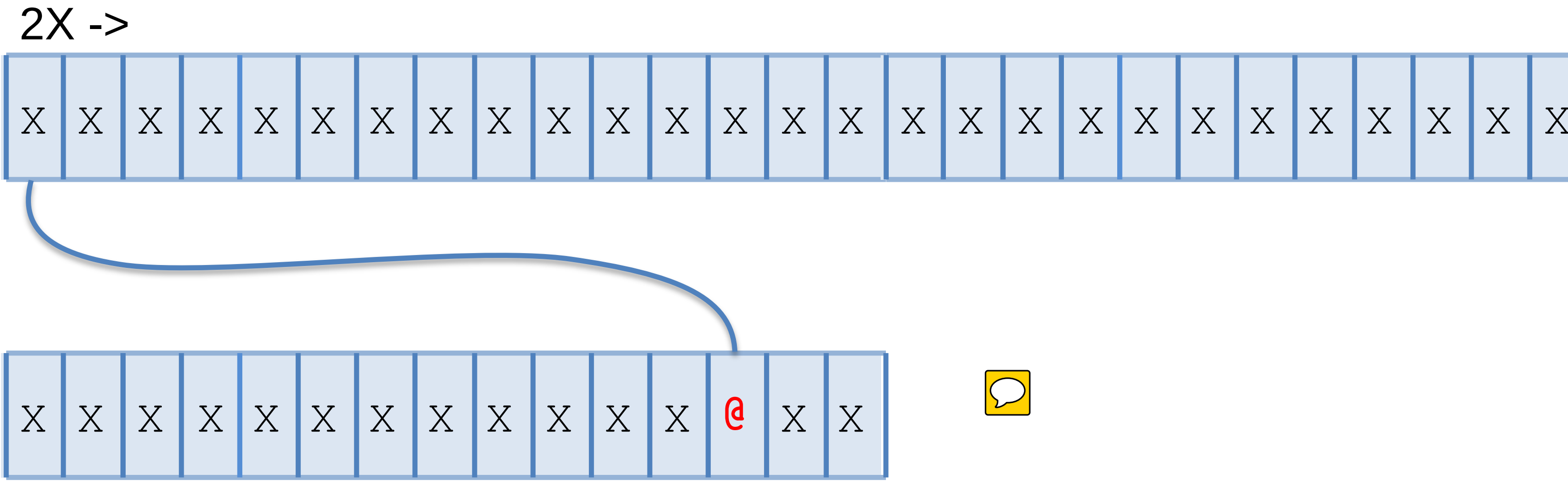
http://ptolemy.eecs.berkeley.edu/

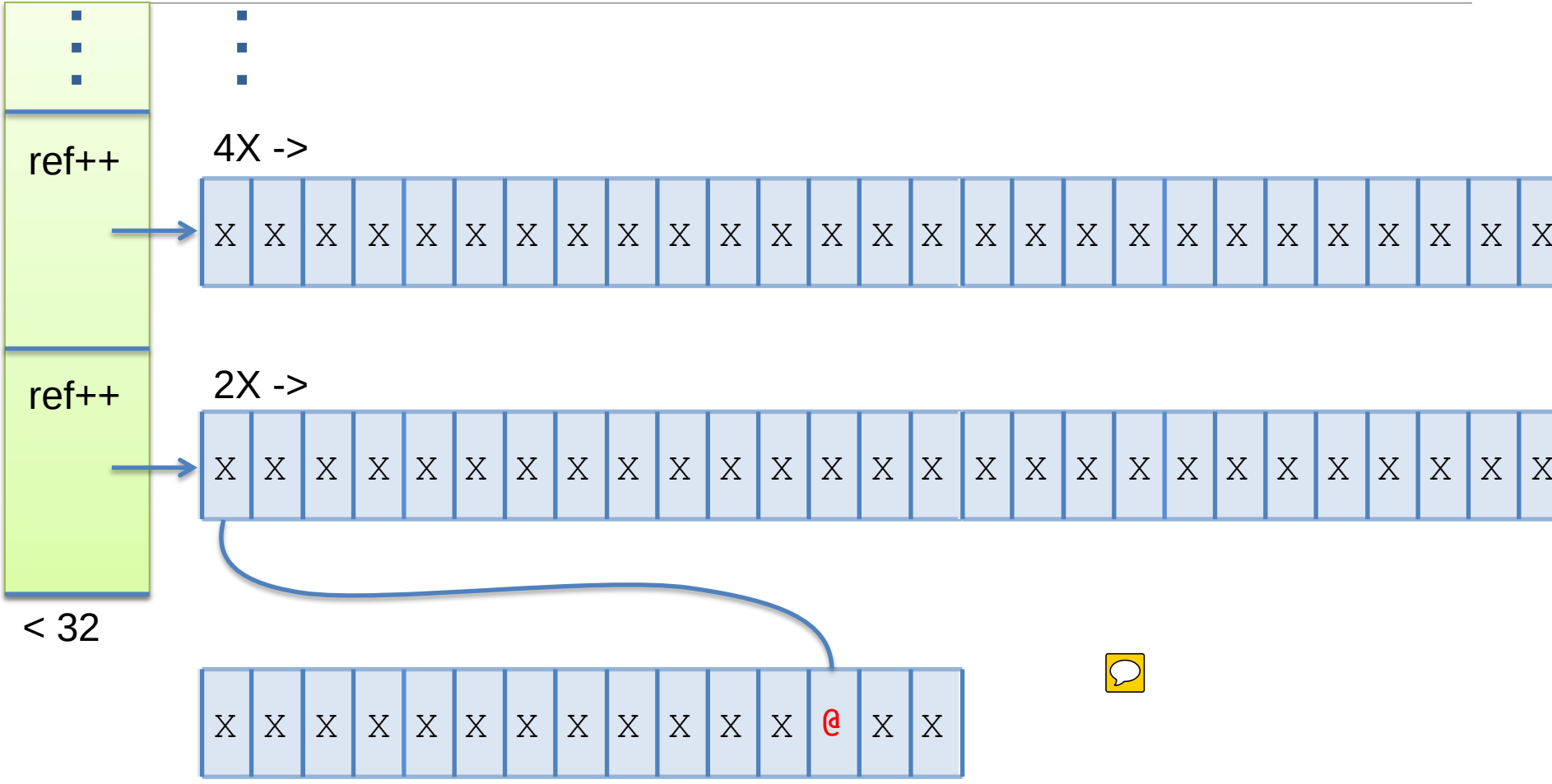
http://ptolemy.eecs.berkeley.edu/publications/papers/06/problemwithThreads/

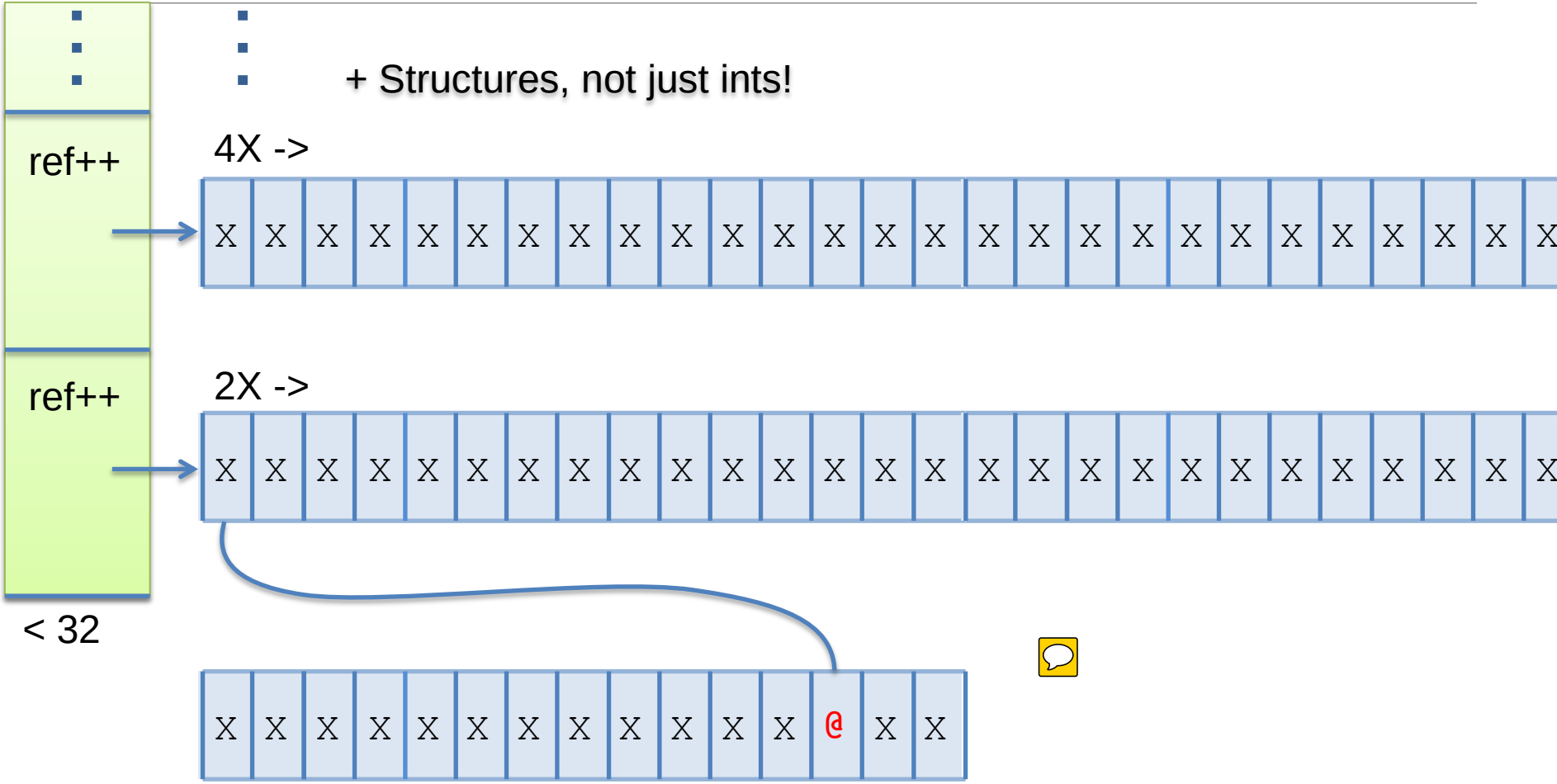
“A part of the Ptolemy Project experiment was to see whether **effective software engineering practices** could be developed for an academic research setting. We developed a process that included a code maturity rating system (with four levels, red, yellow, green, and blue), **design reviews**, **code 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**.”

Looking Ahead


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


2X ->
@


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


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

```
bool push(int val)
{ int prev = 0;
 geni ent;
 geni tmp;
 geni old = tmp = tail;  // laxtomic load
 do {
    ent = buffer[tmp].load(relaxed);
    while( ! is_zero(ent, tmp.gen) ) {
        if (ent.gen < prev) {
           while(!tail.CAS(old,tmp) && old < tmp) { }
           return false; // full
        }  else tmp.incr();
        if (ent.data) prev = ent.gen;  }
    geni newg{val, tmp.gen};
 } while ( ! buffer[tmp].CAS(ent, newg, release));
 tmp.incr(); // go to next
 // update if no one else has gone as far:
 while (!tail.CAS(old, tmp) && old < tmp)  { }
return true;
}
```

```
int pop()
{
 geni ent;
 geni tmp;
 geni old = tmp = head;  // laxtomic load
 do {
    ent = buffer[tmp].load(relaxed);
    while( ! ent.is_data(tmp.gen) )
        if (ent.gen == tmp.gen) {
           while(!head.CAS(old,tmp) && old<tmp){ }
           return 0; // empty
        } else tmp.incr();

    geni zero{0, tmp.gen+1};
 } while ( ! buffer[tmp].CAS(ent, zero, acquire));
 tmp.incr(); // go to next
 // update if no one else has gone as far:
  while (!head.CAS(old, tmp) && old < tmp)  { }
  return ent.val;
}
```