public E take() {
  for (;;) { // Keep trying
    Node<E> oldHead = head.get(); // current head
    Node<E> oldTail = tail.get(); // current tail
    Node<E> oldHeadNext = oldHead.next.get(); // curr head.next
    if (oldHead == head.get()) { //head, tail, and next changed?
      if (oldHead == oldTail) { //Queue empty or tail updated?
        if (oldHeadNext == null) { // Is queue empty?
          return null; //Queue is empty, can't take
        }
        tail.compareAndSet(oldTail, oldHeadNext); // tail updating
      }
    } else { // No need to deal with tail
      if (head.compareAndSet(oldHead, oldHeadNext)) {
        return oldHeadNext.item;
      }
    }
  }
}
public E take() {
    for (;;) { // Keep trying
        Node<E> oldHead = head.get(); // current head
        Node<E> oldTail = tail.get(); // current tail
        Node<E> oldHeadNext = oldHead.next.get(); // curr head.next
        if (oldHead == head.get()) { //head, tail, and next changed?
            if (oldHead == oldTail) { //Queue empty or tail updated?
                if (oldHeadNext == null) { // Is queue empty?
                    return null; //Queue is empty, can't take
                } else { // No need to deal with tail
                    tail.compareAndSet(oldTail, oldHeadNext); // tail updating
                }
            } else { // No need to deal with tail
                if (head.compareAndSet(oldHead, oldHeadNext)) {
                    return oldHeadNext.item;
                }
            }
        }
    }
}
public E take() {
    for (;;) { // Keep trying
        Node<E> oldHead = head.get(); // current head
        Node<E> oldTail = tail.get(); // current tail
        Node<E> oldHeadNext = oldHead.next.get(); // curr head.next
        if (oldHead == head.get()) { //head, tail, and next changed?
            if (oldHead == oldTail) { //Queue empty or tail updated?
                if (oldHeadNext == null) { // Is queue empty?
                    return null; //Queue is empty, can't take
                }
            }
            tail.compareAndSet(oldTail, oldHeadNext); // tail updating
        } else { // No need to deal with tail
            if (head.compareAndSet(oldHead, oldHeadNext)) {
                return oldHeadNext.item;
            }
        }
    }
}
public E take() {
    for (;;) {
        // Keep trying
        Node<E> oldHead = head.get(); // current head
        Node<E> oldTail = tail.get(); // current tail
        Node<E> oldHeadNext = oldHead.next.get(); // curr head.next
        if (oldHead == head.get()) { //head, tail, and next changed?
            if (oldHead == oldTail) { //Queue empty or tail updated?
                if (oldHeadNext == null) { // Is queue empty?
                    return null; //Queue is empty, can't take
                }
                tail.compareAndSet(oldTail, oldHeadNext); // tail updating
            } else { // No need to deal with tail
                if (head.compareAndSet(oldHead, oldHeadNext)) {
                    return oldHeadNext.item;
                }
            }
        }
    }
}
public E take() {
    for (;;) { // Keep trying
        Node<E> oldHead = head.get(); // current head
        Node<E> oldTail = tail.get(); // current tail
        Node<E> oldHeadNext = oldHead.next.get(); // curr head.next
        if (oldHead == head.get()) { //head, tail, and next changed?
            if (oldHead == oldTail) { //Queue empty or tail updated?
                if (oldHeadNext == null) { // Is queue empty?
                    return null; //Queue is empty, can't take
                } else { // No need to deal with tail
                    tail.compareAndSet(oldTail, oldHeadNext); // tail updating
                }
            } else { //head, tail, and next changed?
                if (head.compareAndSet(oldHead, oldHeadNext)) {
                    return oldHeadNext.item;
                }
            }
        }
    }
}
public E take() {
    for (;;) { // Keep trying
        Node<E> oldHead = head.get(); // current head
        Node<E> oldTail = tail.get(); // current tail
        Node<E> oldHeadNext = oldHead.next.get(); // curr head.next
        if (oldHead == head.get()) { //head, tail, and next changed?
            if (oldHead == oldTail) { //Queue empty or tail updated?
                if (oldHeadNext == null) { // Is queue empty?
                    return null; //Queue is empty, can't take
                }
            } else { // No need to deal with tail
                tail.compareAndSet(oldTail, oldHeadNext); // tail updating
            }
        } else { // No need to deal with tail
            if (head.compareAndSet(oldHead, oldHeadNext)) {
                return oldHeadNext.item;
            }
        }
    }
}
public E take() {
    for (;;) {                         // Keep trying
        Node<E> oldHead = head.get();   // current head
        Node<E> oldTail = tail.get();   // current tail
        Node<E> oldHeadNext = oldHead.next.get();  // curr head.next
        if (oldHead == head.get()) {   //head, tail, and next changed?
            if (oldHead == oldTail) {   //Queue empty or tail updated?
                if (oldHeadNext == null) {  // Is queue empty?
                    return null;         //Queue is empty, can't take
                } else {                 // No need to deal with tail
                    tail.compareAndSet(oldTail, oldHeadNext);  // tail updating
                    } else {
                        if (head.compareAndSet(oldHead, oldHeadNext)) {
                            return oldHeadNext.item;
                        }
                    }
                }
        }
    }
}
public E take() {
    for (; ;) { // Keep trying
        Node<E> oldHead = head.get(); // current head
        Node<E> oldTail = tail.get(); // current tail
        Node<E> oldHeadNext = oldHead.next.get(); // curr head.next
        if (oldHead == head.get()) { //head, tail, and next changed?
            if (oldHead == oldTail) { //Queue empty or tail updated?
                if (oldHeadNext == null) { // Is queue empty?
                    return null; //Queue is empty, can't take
                } else { // No need to deal with tail
                    tail.compareAndSet(oldTail, oldHeadNext); // tail updating
                }
            } else { //No need to deal with tail
                if (head.compareAndSet(oldHead, oldHeadNext)) {
                    return oldHeadNext.item;
                }
            }
        }
    }
}
public E take() {
    for (; ;) {  // Keep trying
        Node<E> oldHead = head.get();  // current head
        Node<E> oldTail = tail.get();  // current tail
        Node<E> oldHeadNext = oldHead.next.get();  // curr head.next
        if (oldHead == head.get()) {  //head, tail, and next changed?
            if (oldHead == oldTail) {  //Queue empty or tail updated?
                if (oldHeadNext == null) {  // Is queue empty?
                    return null;  //Queue is empty, can't take
                }
                tail.compareAndSet(oldTail, oldHeadNext);  // tail updating
            } else {  // No need to deal with tail
                if (head.compareAndSet(oldHead, oldHeadNext)) {
                    return oldHeadNext.item;
                }
            }
        } else {  // No need to deal with tail
            return null;  //Queue is empty, can't take
        }
    }
}
Trace 1: T2 continues, CAS fails

public E take() {
    for (;;) {  // Keep trying
        Node<E> oldHead = head.get();  // current head
        Node<E> oldTail = tail.get();  // current tail
        Node<E> oldHeadNext = oldHead.next.get(); // curr head.next
        if (oldHead == head.get()) {  //head, tail, and next changed?
            if (oldHead == oldTail) {  //Queue empty or tail updated?
                if (oldHeadNext == null) {  // Is queue empty?
                    return null;  //Queue is empty, can't take
                }
            }
            else {  // No need to deal with tail
                tail.compareAndSet(oldTail, oldHeadNext);  // tail updating
                if (head.compareAndSet(oldHead, oldHeadNext)) {
                    return oldHeadNext.item;
                }
            }
        }
    }
}
public E take() {
    for (;;) { // Keep trying
        Node<E> oldHead = head.get(); // current head
        Node<E> oldTail = tail.get(); // current tail
        Node<E> oldHeadNext = oldHead.next.get(); // curr head.next
        if (oldHead == head.get()) { //head, tail, and next changed?
            if (oldHead == oldTail) { //Queue empty or tail updated?
                if (oldHeadNext == null) { // Is queue empty?
                    return null; //Queue is empty, can't take
                } else { // No need to deal with tail
                    tail.compareAndSet(oldTail, oldHeadNext); // tail updating
                }
            } else { // No need to deal with tail
                if (head.compareAndSet(oldHead, oldHeadNext)) {
                    return oldHeadNext.item;
                }
            }
        }
    }
}
public E take() {
    for (;;) {        // Keep trying
1    Node<E> oldHead = head.get();    // current head
2    Node<E> oldTail = tail.get();    // current tail
3    Node<E> oldHeadNext = oldHead.next.get(); // curr head.next
4    if (oldHead == head.get()) {    //head, tail, and next changed?
5        if (oldHead == oldTail) {    //Queue empty or tail updated?
6            if (oldHeadNext == null) {        // Is queue empty?
7                return null;        //Queue is empty, can't take
8            } else {    // No need to deal with tail
9                tail.compareAndSet(oldTail, oldHeadNext); // tail updating
10            } else {
11                head.compareAndSet(oldHead, oldHeadNext); {        // No need to deal with tail
12                return oldHeadNext.item;
13            }
14        }
15    }
16}
}
public E take() {
    for (;;) { // Keep trying
        Node<E> oldHead = head.get(); // current head
        Node<E> oldTail = tail.get(); // current tail
        Node<E> oldHeadNext = oldHead.next.get(); // curr head.next
        if (oldHead == head.get()) { //head, tail, and next changed?
            if (oldHead == oldTail) { //Queue empty or tail updated?
                if (oldHeadNext == null) { // Is queue empty?
                    return null; //Queue is empty, can't take
                }
            } else { // No need to deal with tail
                tail.compareAndSet(oldTail, oldHeadNext); // tail updating
            }
        } else { // No need to deal with tail
            if (head.compareAndSet(oldHead, oldHeadNext)) {
                return oldHeadNext.item;
            }
        }
    }
}
public E take() {
    for (;;) {  // Keep trying
        Node<E> oldHead = head.get();  // current head
        Node<E> oldTail = tail.get();  // current tail
        Node<E> oldHeadNext = oldHead.next.get();  // curr head.next
        if (oldHead == head.get()) {  //head, tail, and next changed?
            if (oldHead == oldTail) {  //Queue empty or tail updated?
                if (oldHeadNext == null) {  // Is queue empty?
                    return null;  //Queue is empty, can't take
                }
                tail.compareAndSet(oldTail, oldHeadNext);  // tail updating
            } else {  // No need to deal with tail
                if (head.compareAndSet(oldHead, oldHeadNext)) {
                    return oldHeadNext.item;
                }
            }
        }
    }
}
public E take() {
    for (;;) { // Keep trying
        Node<E> oldHead = head.get(); // current head
        Node<E> oldTail = tail.get(); // current tail
        Node<E> oldHeadNext = oldHead.next.get(); // curr head.next
        if (oldHead == head.get()) { //head, tail, and next changed?
            if (oldHead == oldTail) { //Queue empty or tail updated?
                if (oldHeadNext == null) { // Is queue empty?
                    return null; //Queue is empty, can't take
                }
            } else { // No need to deal with tail
                tail.compareAndSet(oldTail, oldHeadNext); // tail updating
            }
        } else { // No need to deal with tail
            if (head.compareAndSet(oldHead, oldHeadNext)) {
                return oldHeadNext.item;
            }
        }
    }
}
public E take() {
    for (;;) {                      // Keep trying
        Node<E> oldHead = head.get(); // current head
        Node<E> oldTail = tail.get(); // current tail
        Node<E> oldHeadNext = oldHead.next.get(); // curr head.next
        if (oldHead == head.get()) {  //head, tail, and next changed?
            if (oldHead == oldTail) {   //Queue empty or tail updated?
                if (oldHeadNext == null) { // Is queue empty?
                    return null;          //Queue is empty, can't take
                }
            }
            tail.compareAndSet(oldTail, oldHeadNext); // tail updating
        } else {                      // No need to deal with tail
            if (head.compareAndSet(oldHead, oldHeadNext)) {
                return oldHeadNext.item;
            }
        }
    }
}
public E take() {
    for (;;) { // Keep trying
        Node<E> oldHead = head.get(); // current head
        Node<E> oldTail = tail.get(); // current tail
        Node<E> oldHeadNext = oldHead.next.get(); // curr head.next
        if (oldHead == head.get()) { //head, tail, and next changed?
            if (oldHead == oldTail) { //Queue empty or tail updated?
                if (oldHeadNext == null) { // Is queue empty?
                    return null; //Queue is empty, can't take
                } else { // No need to deal with tail
                    tail.compareAndSet(oldTail, oldHeadNext); // tail updating
                }
            } else { // No need to deal with tail
                if (head.compareAndSet(oldHead, oldHeadNext)) {
                    return oldHeadNext.item;
                }
            }
        }
    }
}
public E take() {
    for (;;) { // Keep trying
        Node<E> oldHead = head.get(); // current head
        Node<E> oldTail = tail.get(); // current tail
        Node<E> oldHeadNext = oldHead.next.get(); // curr head.next
        if (oldHead == head.get()) { //head, tail, and next changed?
            if (oldHead == oldTail) { //Queue empty or tail updated?
                if (oldHeadNext == null) { // Is queue empty?
                    return null; //Queue is empty, can't take
                }
            }
        }
        tail.compareAndSet(oldTail, oldHeadNext); // tail updating
    } else { // No need to deal with tail
        if (head.compareAndSet(oldHead, oldHeadNext)) {
            return oldHeadNext.item;
        }
    }
}
public E take() {
    for (;;) { // Keep trying
        Node<E> oldHead = head.get(); // current head
        Node<E> oldTail = tail.get(); // current tail
        Node<E> oldHeadNext = oldHead.next.get(); // curr head.next
        if (oldHead == head.get()) { //head, tail, and next changed?
            if (oldHead == oldTail) { //Queue empty or tail updated?
                if (oldHeadNext == null) { // Is queue empty?
                    return null; //Queue is empty, can't take
                }
            }
        } else { // No need to deal with tail
            tail.compareAndSet(oldTail, oldHeadNext); // tail updating
        }
        if (head.compareAndSet(oldHead, oldHeadNext)) {
            return oldHeadNext.item;
        }
    }
}
```java
public E take() {
    for (;;) { // Keep trying
        Node<E> oldHead = head.get(); // current head
        Node<E> oldTail = tail.get(); // current tail
        Node<E> oldHeadNext = oldHead.next.get(); // curr head.next
        if (oldHead == head.get()) { //head, tail, and next changed?
            if (oldHead == oldTail) { //Queue empty or tail updated?
                if (oldHeadNext == null) { // Is queue empty?
                    return null; //Queue is empty, can't take
                } else { // No need to deal with tail
                    tail.compareAndSet(oldTail, oldHeadNext); // tail updating
                }
            } else { // No need to deal with tail
                if (head.compareAndSet(oldHead, oldHeadNext)) {
                    return oldHeadNext.item;
                }
            }
        }
    }
}
```
public E take() {
    for (;;) { // Keep trying
        Node<E> oldHead = head.get(); // current head
        Node<E> oldTail = tail.get(); // current tail
        Node<E> oldHeadNext = oldHead.next.get(); // curr head.next
        if (oldHead == head.get()) { //head, tail, and next changed?
            if (oldHead == oldTail) { //Queue empty or tail updated?
                if (oldHeadNext == null) { // Is queue empty?
                    return null; //Queue is empty, can't take
                }
            } else { // No need to deal with tail
                tail.compareAndSet(oldTail, oldHeadNext); // tail updating
            }
        } else { // No need to deal with tail
            if (head.compareAndSet(oldHead, oldHeadNext)) {
                return oldHeadNext.item;
            }
        }
    }
}
public E take() {
    for (;;) { // Keep trying
        Node<E> oldHead = head.get(); // current head
        Node<E> oldTail = tail.get(); // current tail
        Node<E> oldHeadNext = oldHead.next.get(); // curr head.next
        if (oldHead == head.get()) { //head, tail, and next changed?
            if (oldHead == oldTail) { //Queue empty or tail updated?
                if (oldHeadNext == null) { // Is queue empty?
                    return null; //Queue is empty, can't take
                }
            }
            else { // No need to deal with tail
                tail.compareAndSet(oldTail, oldHeadNext); // tail updating
            }
        }
        else { // No need to deal with tail
            if (head.compareAndSet(oldHead, oldHeadNext)) {
                return oldHeadNext.item;
            }
        }
    }
}
public E take() {
    for (; ; ) { // Keep trying
        Node<E> oldHead = head.get(); // current head
        Node<E> oldTail = tail.get(); // current tail
        Node<E> oldHeadNext = oldHead.next.get(); // curr head.next
        if (oldHead == head.get()) { //head, tail, and next changed?
            if (oldHead == oldTail) { //Queue empty or tail updated?
                if (oldHeadNext == null) { // Is queue empty?
                    return null; //Queue is empty, can't take
                }
            }
            else { // No need to deal with tail
                tail.compareAndSet(oldTail, oldHeadNext); // tail updating
            }
        }
    else { // No need to deal with tail
        if (head.compareAndSet(oldHead, oldHeadNext)) { // Head, tail, and next changed?
            return oldHeadNext.item;
        }
    }
}
public E take() {
    for (; ; ) { // Keep trying
        Node<E> oldHead = head.get(); // current head
        Node<E> oldTail = tail.get(); // current tail
        Node<E> oldHeadNext = oldHead.next.get(); // curr head.next
        if (oldHead == head.get()) { //head, tail, and next changed?
            if (oldHead == oldTail) { //Queue empty or tail updated?
                if (oldHeadNext == null) { // Is queue empty?
                    return null; //Queue is empty, can't take
                }
            } else { // No need to deal with tail
                tail.compareAndSet(oldTail, oldHeadNext); // tail updating
            }
        } else { // No need to deal with tail
            if (head.compareAndSet(oldHead, oldHeadNext)) {
                return oldHeadNext.item;
            }
        }
    }
}

<table>
<thead>
<tr>
<th>Var</th>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>oldHead</td>
<td></td>
<td></td>
</tr>
<tr>
<td>oldTail</td>
<td></td>
<td></td>
</tr>
<tr>
<td>oldHeadNext</td>
<td></td>
<td></td>
</tr>
<tr>
<td>head</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tail</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1 2 λ
public E take() {
    for (;;) { // Keep trying
        Node<E> oldHead = head.get(); // current head
        Node<E> oldTail = tail.get(); // current tail
        Node<E> oldHeadNext = oldHead.next.get(); // curr head.next
        if (oldHead == head.get()) { //head, tail, and next changed?
            if (oldHead == oldTail) { //Queue empty or tail updated?
                if (oldHeadNext == null) { // Is queue empty?
                    return null; //Queue is empty, can't take
                }
                tail.compareAndSet(oldTail, oldHeadNext); // tail updating
            } else { // No need to deal with tail
                if (head.compareAndSet(oldHead, oldHeadNext)) {
                    return oldHeadNext.item;
                }
            }
        }
    }
}
public E take() {
    for (;;) { // Keep trying
        Node<E> oldHead = head.get(); // current head
        Node<E> oldTail = tail.get(); // current tail
        Node<E> oldHeadNext = oldHead.next.get(); // curr head.next
        if (oldHead == head.get()) { // head, tail, and next changed?
            if (oldHead == oldTail) { //Queue empty or tail updated?
                if (oldHeadNext == null) { // Is queue empty?
                    return null; //Queue is empty, can't take
                } else { // No need to deal with tail
                    tail.compareAndSet(oldTail, oldHeadNext); // tail updating
                }
            } else { // No need to deal with tail
                if (head.compareAndSet(oldHead, oldHeadNext)) {
                    return oldHeadNext.item;
                }
            }
        }
    }
}
public E take() {
    for (;;) { // Keep trying
        Node<E> oldHead = head.get(); // current head
        Node<E> oldTail = tail.get(); // current tail
        Node<E> oldHeadNext = oldHead.next.get(); // curr head.next
        if (oldHead == head.get()) { //head, tail, and next changed?
            if (oldHead == oldTail) { //Queue empty or tail updated?
                if (oldHeadNext == null) { // Is queue empty?
                    return null; //Queue is empty, can't take
                } else { // No need to deal with tail
                    tail.compareAndSet(oldTail, oldHeadNext); // tail updating
                }
            }
            else { // No need to deal with tail
                if (head.compareAndSet(oldHead, oldHeadNext)) {
                    return oldHeadNext.item;
                }
            }
        }
    }
}
public E take() {
    for (;;) {  // Keep trying
        Node<E> oldHead = head.get();  // current head
        Node<E> oldTail = tail.get();  // current tail
        Node<E> oldHeadNext = oldHead.next.get();  // curr head.next
        if (oldHead == head.get()) {  // head, tail, and next changed?
            if (oldHead == oldTail) {  // Queue empty or tail updated?
                if (oldHeadNext == null) {  // Is queue empty?
                    return null;  // Queue is empty, can't take
                }
            } else {  // No need to deal with tail
                tail.compareAndSet(oldTail, oldHeadNext);  // tail updating
            }
        } else {  // No need to deal with tail
            if (head.compareAndSet(oldHead, oldHeadNext)) {
                return oldHeadNext.item;
            }
        }
    }
}
Trace 2: T2 continues, retry

```java
public E take() {
    for (;;) {                        // Keep trying
        Node<E> oldHead = head.get(); // current head
        Node<E> oldTail = tail.get(); // current tail
        Node<E> oldHeadNext = oldHead.next.get(); // curr head.next
        if (oldHead == head.get()) {      //head, tail, and next changed?
            if (oldHead == oldTail) {     //Queue empty or tail updated?
                if (oldHeadNext == null) { // Is queue empty?
                    return null;            //Queue is empty, can't take
                }
            } else {                     // No need to deal with tail
                if (head.compareAndSet(oldHead, oldHeadNext)) {
                    return oldHeadNext.item;
                }
            }
        }
        tail.compareAndSet(oldTail, oldHeadNext); // tail updating
    }
}
```
public E take() {
    for (;;) { // Keep trying
        Node<E> oldHead = head.get(); // current head
        Node<E> oldTail = tail.get(); // current tail
        Node<E> oldHeadNext = oldHead.next.get(); // cur head.next
        if (oldHead == head.get()) { //head, tail, and next changed?
            if (oldHead == oldTail) { //Queue empty or tail updated?
                if (oldHeadNext == null) { // Is queue empty?
                    return null; //Queue is empty, can't take
                }
            } else { // No need to deal with tail
                tail.compareAndSet(oldTail, oldHeadNext); // tail updating
            }
        } else { // No need to deal with tail
            if (head.compareAndSet(oldHead, oldHeadNext)) {
                return oldHeadNext.item;
            }
        }
    }
}
public E take() {
    for (;;) {
        // Keep trying
        Node<E> oldHead = head.get();  // current head
        Node<E> oldTail = tail.get();  // current tail
        Node<E> oldHeadNext = oldHead.next.get(); // curr head.next
        if (oldHead == head.get()) {  // head, tail, and next changed?
            if (oldHead == oldTail) {  // Queue empty or tail updated?
                if (oldHeadNext == null) {  // Is queue empty?
                    return null;  // Queue is empty, can't take
                } else {  // No need to deal with tail
                    tail.compareAndSet(oldTail, oldHeadNext); // tail updating
                }
            } else {  // No need to deal with tail
                if (head.compareAndSet(oldHead, oldHeadNext)) {
                    return oldHeadNext.item;
                }
            }
        }
    }
}
public E take() {
    for (;;) { // Keep trying
        Node<E> oldHead = head.get(); // current head
        Node<E> oldTail = tail.get(); // current tail
        Node<E> oldHeadNext = oldHead.next.get(); // curr head.next
        if (oldHead == head.get()) { //head, tail, and next changed?
            if (oldHead == oldTail) { //Queue empty or tail updated?
                if (oldHeadNext == null) { // Is queue empty?
                    return null; //Queue is empty, can't take
                }
                tail.compareAndSet(oldTail, oldHeadNext); // tail updating
            } else { // No need to deal with tail
                if (head.compareAndSet(oldHead, oldHeadNext)) {
                    return oldHeadNext.item;
                }
            }
        }
    }
}
public E take() {
    for (;;) { // Keep trying
        Node<E> oldHead = head.get(); // current head
        Node<E> oldTail = tail.get(); // current tail
        Node<E> oldHeadNext = oldHead.next.get(); // curr head.next
        if (oldHead == head.get()) { //head, tail, and next changed?
            if (oldHead == oldTail) { //Queue empty or tail updated?
                if (oldHeadNext == null) { // Is queue empty?
                    return null; //Queue is empty, can't take
                }
                tail.compareAndSet(oldTail, oldHeadNext); // tail updating
            } else { // No need to deal with tail
                if (head.compareAndSet(oldHead, oldHeadNext)) {
                    return oldHeadNext.item;
                }
            }
        }
    }
}
public E take() {
    for (; ;) {  // Keep trying
        Node<E> oldHead = head.get();  // current head
        Node<E> oldTail = tail.get();  // current tail
        Node<E> oldHeadNext = oldHead.next.get(); // curr head.next
        if (oldHead == head.get()) {  //head, tail, and next changed?
            if (oldHead == oldTail) {  //Queue empty or tail updated?
                if (oldHeadNext == null) {  // Is queue empty?
                    return null;  //Queue is empty, can't take
                } else {  // No need to deal with tail
                    tail.compareAndSet(oldTail, oldHeadNext);  // tail updating
                }
            } else {  // No need to deal with tail
                if (head.compareAndSet(oldHead, oldHeadNext)) {
                    return oldHeadNext.item;
                }
            }
        }
    }
}
public E take() {
    for (;;) { // Keep trying
        Node<E> oldHead = head.get(); // current head
        Node<E> oldTail = tail.get(); // current tail
        Node<E> oldHeadNext = oldHead.next.get(); // curr head.next
        if (oldHead == head.get()) { //head, tail, and next changed?
            if (oldHead == oldTail) { //Queue empty or tail updated?
                if (oldHeadNext == null) { // Is queue empty?
                    return null; //Queue is empty, can't take
                }
                tail.compareAndSet(oldTail, oldHeadNext); // tail updating
            } else { // No need to deal with tail
                if (head.compareAndSet(oldHead, oldHeadNext)) {
                    return oldHeadNext.item;
                }
            }
        }
    }
}
public E take() {
    for (;;) {
        // Keep trying
        Node<E> oldHead = head.get(); // current head
        Node<E> oldTail = tail.get(); // current tail
        Node<E> oldHeadNext = oldHead.next.get(); // curr head.next
        if (oldHead == head.get()) { //head, tail, and next changed?
            if (oldHead == oldTail) { //Queue empty or tail updated?
                if (oldHeadNext == null) { // Is queue empty?
                    return null; //Queue is empty, can't take
                } else {
                } // No need to deal with tail
            } else {
                tail.compareAndSet(oldTail, oldHeadNext); // tail updating
            } else {
                if (head.compareAndSet(oldHead, oldHeadNext)) {
                    return oldHeadNext.item;
                }
            }
    }
}
public boolean put(E item) {
    Node<E> newNode = new Node<E>(item, null);
    while (true) {
        Node<E> curTail = tail.get();
        Node<E> tailNext = curTail.next.get();
        if (curTail == tail.get()) { // did tail change?
            if (tailNext != null) { // Queue in int. state, advance tail
                tail.compareAndSet(curTail, tailNext);
            } else { // In quiescent state, try inserting new node
                if (curTail.next.compareAndSet(null, newNode)) {
                    // Insertion succeeded, try advancing tail
                    tail.compareAndSet(curTail, newNode);
                    // will fail if tail already moved
                }
            }
        }
        return true;
    }
}
public E take() {
    for (; ; ) { // Keep trying
        Node<E> oldHead = head.get(); // current head
        Node<E> oldTail = tail.get(); // current tail
        Node<E> oldHeadNext = oldHead.next.get(); // curr head.next
        if (oldHead == head.get()) { // head, tail, and next changed?
            if (oldHead == oldTail) { //Queue empty or tail updated?
                if (oldHeadNext == null) { // Is queue empty?
                    return null; //Queue is empty, can't take
                } else { // No need to deal with tail
                    tail.compareAndSet(oldTail, oldHeadNext); // tail updating
                }
            } else { // No need to deal with tail
                if (head.compareAndSet(oldHead, oldHeadNext)) {
                    return oldHeadNext.item;
                }
            }
        }
    }
}
public E take() {
    for (;;) {  // Keep trying
        Node<E> oldHead = head.get();  // current head
        Node<E> oldTail = tail.get();  // current tail
        Node<E> oldHeadNext = oldHead.next.get();  // curr head.next
        if (oldHead == head.get()) {  //head, tail, and next changed?
            if (oldHead == oldTail) {  //Queue empty or tail updated?
                if (oldHeadNext == null) {  // Is queue empty?
                    return null;  //Queue is empty, can't take
                }
                tail.compareAndSet(oldTail, oldHeadNext);  // tail updating
            } else {  // No need to deal with tail
                if (head.compareAndSet(oldHead, oldHeadNext)) {
                    return oldHeadNext.item;
                }
            }
        } else {  // No need to deal with tail
            if (head.compareAndSet(oldHead, oldHeadNext)) {
                return oldHeadNext.item;
            }
        }
    }
}
public boolean put(E item) {
    Node<E> newNode = new Node<E>(item, null);
    while (true) {
        Node<E> curTail = tail.get();
        Node<E> tailNext = curTail.next.get();
        if (curTail == tail.get()) {// did tail change?
            if (tailNext != null) {// Queue in int. state, advance tail
                tail.compareAndSet(curTail, tailNext);
            } else { // In quiescent state, try inserting new node
                if (curTail.next.compareAndSet(null, newNode)) {
                    // Insertion succeeded, try advancing tail
                    tail.compareAndSet(curTail, newNode);
                    // will fail if tail already moved
                    return true;
                }
            }
        }
    }
}
public boolean put(E item) {
  Node<E> newNode = new Node<E>(item, null);
  while (true) {
    Node<E> curTail = tail.get();
    Node<E> tailNext = curTail.next.get();
    if (curTail == tail.get()) {// did tail change?
      if (tailNext != null) {// Queue in int. state, advance tail
        tail.compareAndSet(curTail, tailNext);
      } else { // In quiescent state, try inserting new node
        if (curTail.next.compareAndSet(null, newNode)) {
          // Insertion succeeded, try advancing tail
          tail.compareAndSet(curTail, newNode);
        } else { // will fail if tail already moved
          return false;
        }
      }
    }
  }
}
public boolean put(E item) {
    Node<E> newNode = new Node<E>(item, null);
    while (true) {
        Node<E> curTail = tail.get();
        Node<E> tailNext = curTail.next.get();
        if (curTail == tail.get()) {// did tail change?
            if (tailNext != null) { // Queue in int. state, advance tail
                tail.compareAndSet(curTail, tailNext);
            } else { // In quiescent state, try inserting new node
                if (curTail.next.compareAndSet(null, newNode)) {
                    // Insertion succeeded, try advancing tail
                    tail.compareAndSet(curTail, newNode);
                    // will fail if tail already moved
                    return true;
                }
            }
        }
    }
}
Trace 3: T2 continues

```java
public boolean put(E item) {
    Node<E> newNode = new Node<E>(item, null);
    while (true) {
        Node<E> curTail = tail.get();
        Node<E> tailNext = curTail.next.get();
        if (curTail == tail.get()) {// did tail change?
            if (tailNext != null) { // Queue in int. state, advance tail
                tail.compareAndSet(curTail, tailNext);
            } else { // In quiescent state, try inserting new node
                if (curTail.next.compareAndSet(null, newNode)) {
                    // Insertion succeeded, try advancing tail
                    tail.compareAndSet(curTail, newNode);
                    // will fail if tail already moved
                }
            }
        } else {
            return true;
        }
    }
}
```
public boolean put(E item) {
    Node<E> newNode = new Node<E>(item, null);
    while (true) {
        Node<E> curTail = tail.get();
        Node<E> tailNext = curTail.next.get();
        if (curTail == tail.get()) {/* did tail change? */
            if (tailNext != null) { /* Queue in int. state, advance tail */
                tail.compareAndSet(curTail, tailNext);
            } else { /* In quiescent state, try inserting new node */
                if (curTail.next.compareAndSet(null, newNode)) {
                    // Insertion succeeded, try advancing tail
                    tail.compareAndSet(curTail, newNode);
                    // will fail if tail already moved
                }
            }
        }
        return true;
    }
}
public boolean put(E item) {
    Node<E> newNode = new Node<E>(item, null);
    while (true) {
        Node<E> curTail = tail.get();
        Node<E> tailNext = curTail.next.get();
        if (curTail == tail.get()) {// did tail change?
            if (tailNext != null) {// Queue in int. state, advance tail
                tail.compareAndSet(curTail, tailNext);
            } else { // In quiescent state, try inserting new node
                if (curTail.next.compareAndSet(null, newNode)) {
                    // Insertion succeeded, try advancing tail
                    tail.compareAndSet(curTail, newNode);
                    // will fail if tail already moved
                }
            }
        }
        return true;
    }
}
public E take() {
    for (;;) { // Keep trying
        Node<E> oldHead = head.get(); // current head
        Node<E> oldTail = tail.get(); // current tail
        Node<E> oldHeadNext = oldHead.next.get(); // curr head.next
        if (oldHead == head.get()) { //head, tail, and next changed?
            if (oldHead == oldTail) { //Queue empty or tail updated?
                if (oldHeadNext == null) { // Is queue empty?
                    return null; //Queue is empty, can't take
                }
            } else { // No need to deal with tail
                tail.compareAndSet(oldTail, oldHeadNext); // tail updating
            }
        } else { // No need to deal with tail
            if (head.compareAndSet(oldHead, oldHeadNext)) {
                return oldHeadNext.item;
            }
        }
    }
}
Trace 3: T1 continues

```java
public E take() {
    for (;;) { // Keep trying
        Node<E> oldHead = head.get(); // current head
        Node<E> oldTail = tail.get(); // current tail
        Node<E> oldHeadNext = oldHead.next.get(); // curr head.next
        if (oldHead == head.get()) { //head, tail, and next changed?
            if (oldHead == oldTail) { //Queue empty or tail updated?
                if (oldHeadNext == null) { // Is queue empty?
                    return null; //Queue is empty, can't take
                }
                tail.compareAndSet(oldTail, oldHeadNext); // tail updating
            } else { // No need to deal with tail
                if (head.compareAndSet(oldHead, oldHeadNext)) {
                    return oldHeadNext.item;
                }
            }
        }
    }
}
```
Trace 3: T1 continues

```java
public E take() {
    for (;;) { // Keep trying
        Node<E> oldHead = head.get(); // current head
        Node<E> oldTail = tail.get(); // current tail
        Node<E> oldHeadNext = oldHead.next.get(); // curr head.next
        if (oldHead == head.get()) { //head, tail, and next changed?
            if (oldHead == oldTail) { //Queue empty or tail updated?
                if (oldHeadNext == null) { // Is queue empty?
                    return null; //Queue is empty, can't take
                } else {
                    tail.compareAndSet(oldTail, oldHeadNext); // tail updating
                }
            } else { // No need to deal with tail
                if (head.compareAndSet(oldHead, oldHeadNext)) {
                    return oldHeadNext.item;
                }
                break;
            }
        } else {
            break;
        }
    }
    return null;
}
```
public E take() {
    for (;;) { // Keep trying
        Node<E> oldHead = head.get(); // current head
        Node<E> oldTail = tail.get(); // current tail
        Node<E> oldHeadNext = oldHead.next.get(); // curr head.next
        if (oldHead == head.get()) { //head, tail, and next changed?
            if (oldHead == oldTail) { //Queue empty or tail updated?
                if (oldHeadNext == null) { // Is queue empty?
                    return null; //Queue is empty, can't take
                }
            } else { // No need to deal with tail
                tail.compareAndSet(oldTail, oldHeadNext); // tail updating
            }
        } else { // No need to deal with tail
            if (head.compareAndSet(oldHead, oldHeadNext)) {
                return oldHeadNext.item;
            }
        }
    }
}
public E take() {
    for (;;) {  // Keep trying
        Node<E> oldHead = head.get();  // current head
        Node<E> oldTail = tail.get();  // current tail
        Node<E> oldHeadNext = oldHead.next.get(); // curr head.next
        if (oldHead == head.get()) {  //head, tail, and next changed?
            if (oldHead == oldTail) {  //Queue empty or tail updated?
                if (oldHeadNext == null) {  // Is queue empty?
                    return null;  //Queue is empty, can't take
                } else {  // No need to deal with tail
                    tail.compareAndSet(oldTail, oldHeadNext);  // tail updating
                }
            }
            else {  // No need to deal with tail
                if (head.compareAndSet(oldHead, oldHeadNext)) {
                    return oldHeadNext.item;
                }
            }
        }
    }
}
Trace 3: after CAS

public E take() {
    for (;;) { // Keep trying
        Node<E> oldHead = head.get(); // current head
        Node<E> oldTail = tail.get(); // current tail
        Node<E> oldHeadNext = oldHead.next.get(); // curr head.next
        if (oldHead == head.get()) { //head, tail, and next changed?
            if (oldHead == oldTail) { //Queue empty or tail updated?
                if (oldHeadNext == null) { // Is queue empty?
                    return null; //Queue is empty, can't take
                } else { // No need to deal with tail
                    tail.compareAndSet(oldTail, oldHeadNext); // tail updating
                }
            } else { // head update
                if (head.compareAndSet(oldHead, oldHeadNext)) {
                    return oldHeadNext.item;
                }
            }
        }
    }
}
public boolean put(E item) {
    Node<E> newNode = new Node<E>(item, null);
    while (true) {
        Node<E> curTail = tail.get();
        Node<E> tailNext = curTail.next.get();
        if (curTail == tail.get()) { // did tail change?
            if (tailNext != null) { // Queue in int. state, advance tail
                tail.compareAndSet(curTail, tailNext);
            } else { // In quiescent state, try inserting new node
                if (curTail.next.compareAndSet(null, newNode)) {
                    // Insertion succeeded, try advancing tail
                    tail.compareAndSet(curTail, newNode);
                // will fail if tail already moved
                return true;
            }
        }
    }
}
public boolean put(E item) {
    Node<E> newNode = new Node<E>(item, null);
    while (true) {
        Node<E> curTail = tail.get();
        Node<E> tailNext = curTail.next.get();
        if (curTail == tail.get()) { // did tail change?
            if (tailNext != null) { // Queue in int. state, advance tail
                tail.compareAndSet(curTail, tailNext);
            } else { // In quiescent state, try inserting new node
                if (curTail.next.compareAndSet(null, newNode)) {
                    // Insertion succeeded, try advancing tail
                    tail.compareAndSet(curTail, newNode);
                    // will fail if tail already moved
                }
            }
        }
        return true;
    }
}
Trace 3: T1 continues

```java
class Node<E> {
    E item;
    Node<E> next;
}

public E take() {
    Node<E> oldHead = head.get();       // current head
    Node<E> oldTail = tail.get();       // current tail
    Node<E> oldHeadNext = oldHead.next.get();  // curr head.next
    if (oldHead == head.get()) {        //head, tail, and next changed?
        if (oldHead == oldTail) {      //Queue empty or tail updated?
            if (oldHeadNext == null) {  // Is queue empty?
                return null;           //Queue is empty, can't take
            } else {                 // No need to deal with tail
                tail.compareAndSet(oldTail, oldHeadNext);  // tail updating
            }
        } else {                    // No need to deal with tail
            if (head.compareAndSet(oldHead, oldHeadNext)) {
                return oldHeadNext.item;
            }
        }
    } else {                         // No need to deal with tail
        if (head.compareAndSet(oldHead, oldHeadNext)) {
            return oldHeadNext.item;
        }
    }
}
```
public E take() {
    for (;;) {  // Keep trying
        Node<E> oldHead = head.get();  // current head
        Node<E> oldTail = tail.get();  // current tail
        Node<E> oldHeadNext = oldHead.next.get();  // curr head.next
        if (oldHead == head.get()) {  //head, tail, and next changed?
            if (oldHead == oldTail) {  //Queue empty or tail updated?
                if (oldHeadNext == null) {  // Is queue empty?
                    return null;  //Queue is empty, can't take
                } else {  // No need to deal with tail
                }
            }
            tail.compareAndSet(oldTail, oldHeadNext);  // tail updating
        } else {  // No need to deal with tail
            if (head.compareAndSet(oldHead, oldHeadNext)) {
                return oldHeadNext.item;
            }
        }
    }
}
public E take() {
    for (; ; ) { // Keep trying
        Node<E> oldHead = head.get(); // current head
        Node<E> oldTail = tail.get(); // current tail
        Node<E> oldHeadNext = oldHead.next.get(); // cur head.next
        if (oldHead == head.get()) { //head, tail, and next changed?
            if (oldHead == oldTail) { //Queue empty or tail updated?
                if (oldHeadNext == null) { // Is queue empty?
                    return null; //Queue is empty, can't take
                } else {
                    tail.compareAndSet(oldTail, oldHeadNext); // tail updating
                }
            }
        } else { // No need to deal with tail
            if (head.compareAndSet(oldHead, oldHeadNext)) {
                return oldHeadNext.item;
            }
        }
    }
}
Trace 3: T1 continues

```java
public E take() {
    for (;;) { // Keep trying
        Node<E> oldHead = head.get(); // current head
        Node<E> oldTail = tail.get(); // current tail
        Node<E> oldHeadNext = oldHead.next.get(); // curr head.next
        if (oldHead == head.get()) { //head, tail, and next changed?
            if (oldHead == oldTail) { //Queue empty or tail updated?
                if (oldHeadNext == null) { // Is queue empty?
                    return null; //Queue is empty, can't take
                } else { // No need to deal with tail
                    tail.compareAndSet(oldTail, oldHeadNext); // tail updating
                }
            } else { // No need to deal with tail
                if (head.compareAndSet(oldHead, oldHeadNext)) {
                    return oldHeadNext.item;
                }
            }
        }
    }
}
```
public E take() {
    for (; ;) { // Keep trying
        Node<E> oldHead = head.get(); // current head
        Node<E> oldTail = tail.get(); // current tail
        Node<E> oldHeadNext = oldHead.next.get(); // curr head.next
        if (oldHead == head.get()) { // head, tail, and next changed?
            if (oldHead == oldTail) { //Queue empty or tail updated?
                if (oldHeadNext == null) { // Is queue empty?
                    return null; //Queue is empty, can't take
                }
                tail.compareAndSet(oldTail, oldHeadNext); // tail updating
            }
        } else { // No need to deal with tail
            if (head.compareAndSet(oldHead, oldHeadNext)) {
                return oldHeadNext.item;
            }
        }
    }
}

<table>
<thead>
<tr>
<th>Var</th>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>oldHead</td>
<td></td>
<td></td>
</tr>
<tr>
<td>oldTail</td>
<td></td>
<td></td>
</tr>
<tr>
<td>oldHeadNext</td>
<td></td>
<td></td>
</tr>
<tr>
<td>head</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tail</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2 3 \(\lambda\)
public E take() {
    for (; true;) { // Keep trying
        Node<E> oldHead = head.get(); // current head
        Node<E> oldTail = tail.get(); // current tail
        Node<E> oldHeadNext = oldHead.next.get(); // curr head.next
        if (oldHead == head.get()) { //head, tail, and next changed?
            if (oldHead == oldTail) { //Queue empty or tail updated?
                if (oldHeadNext == null) { // Is queue empty?
                    return null; //Queue is empty, can't take
                }
            }
            tail.compareAndSet(oldTail, oldHeadNext); // tail updating
        } else { // No need to deal with tail
            if (head.compareAndSet(oldHead, oldHeadNext)) {
                return oldHeadNext.item;
            }
        }
    }
}
Trace 3: T1 after CAS

```java
public E take() {
    for (; ;) { // Keep trying
        Node<E> oldHead = head.get(); // current head
        Node<E> oldTail = tail.get(); // current tail
        Node<E> oldHeadNext = oldHead.next.get(); // curr head.next
        if (oldHead == head.get()) { //head, tail, and next changed?
            if (oldHead == oldTail) { //Queue empty or tail updated?
                if (oldHeadNext == null) { // Is queue empty?
                    return null; //Queue is empty, can't take
                } else {
                    tail.compareAndSet(oldTail, oldHeadNext); // tail updating
                }
            } else { // No need to deal with tail
                if (head.compareAndSet(oldHead, oldHeadNext)) {
                    return oldHeadNext.item;
                }
            }
        } else {
        }
    }
}
```

```
<table>
<thead>
<tr>
<th>Var</th>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>oldHead</td>
<td></td>
<td></td>
</tr>
<tr>
<td>oldTail</td>
<td></td>
<td></td>
</tr>
<tr>
<td>oldHeadNext</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Trace 3: T1 exits

```java
public E take() {
    for (;;) { // Keep trying
        Node<E> oldHead = head.get(); // current head
        Node<E> oldTail = tail.get(); // current tail
        Node<E> oldHeadNext = oldHead.next.get(); // curr head.next
        if (oldHead == head.get()) { //head, tail, and next changed?
            if (oldHead == oldTail) { //Queue empty or tail updated?
                if (oldHeadNext == null) { // Is queue empty?
                    return null; //Queue is empty, can't take
                }
            } else { // No need to deal with tail
                tail.compareAndSet(oldTail, oldHeadNext); // tail updating
            }
        } else { // No need to deal with tail
            if (head.compareAndSet(oldHead, oldHeadNext)) {
                return oldHeadNext.item;
            }
        }
    }
}
```