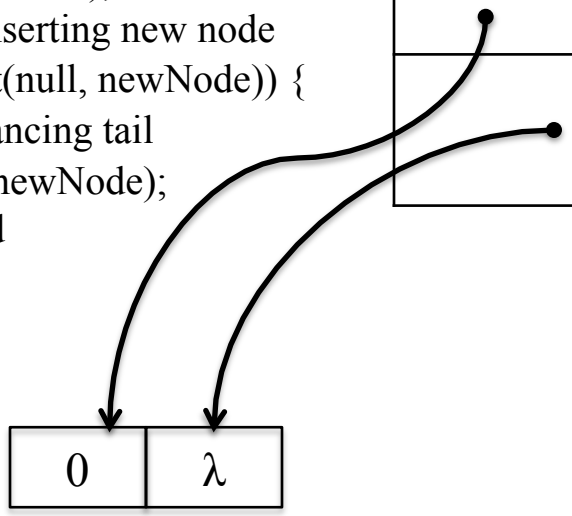
Trace 1

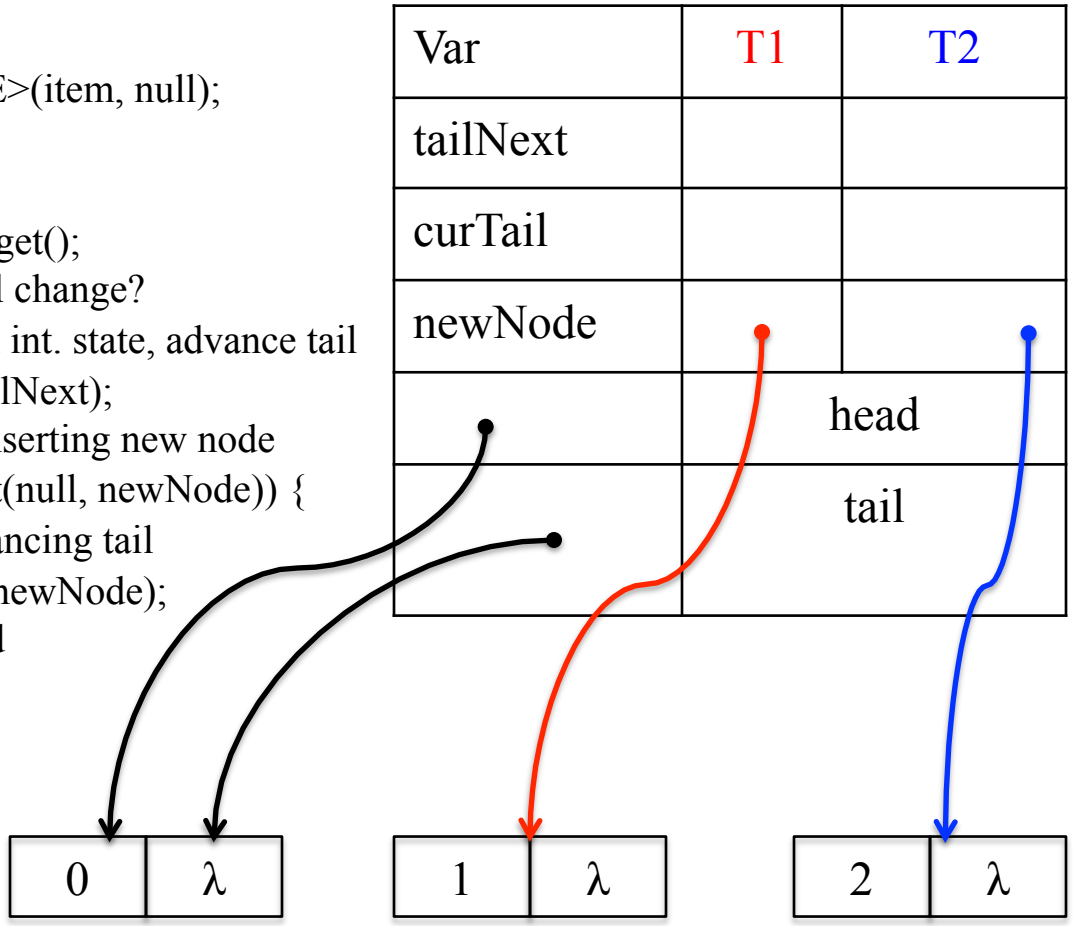
```
    public boolean put(E item) {
1     Node<E> newNode = new Node<E>(item, null);
2     while (true) {
3       Node<E> curTail = tail.get();
4       Node<E> tailNext = curTail.next.get();
5       if (curTail == tail.get()) {// did tail change?
6         if (tailNext != null) { // Queue in int. state, advance tail
7           tail.compareAndSet(curTail, tailNext);
8         else {  // In quiescent state, try inserting new node
9           if (curTail.next.compareAndSet(null, newNode)) {
            // Insertion succeeded, try advancing tail
10            tail.compareAndSet(curTail, newNode);
            // will fail if tail already moved
11          return true;
            }
          }
        }
      }
    }
```

Var	T1	T2
tailNext		
curTail		
newNode		
	head	
	tail	



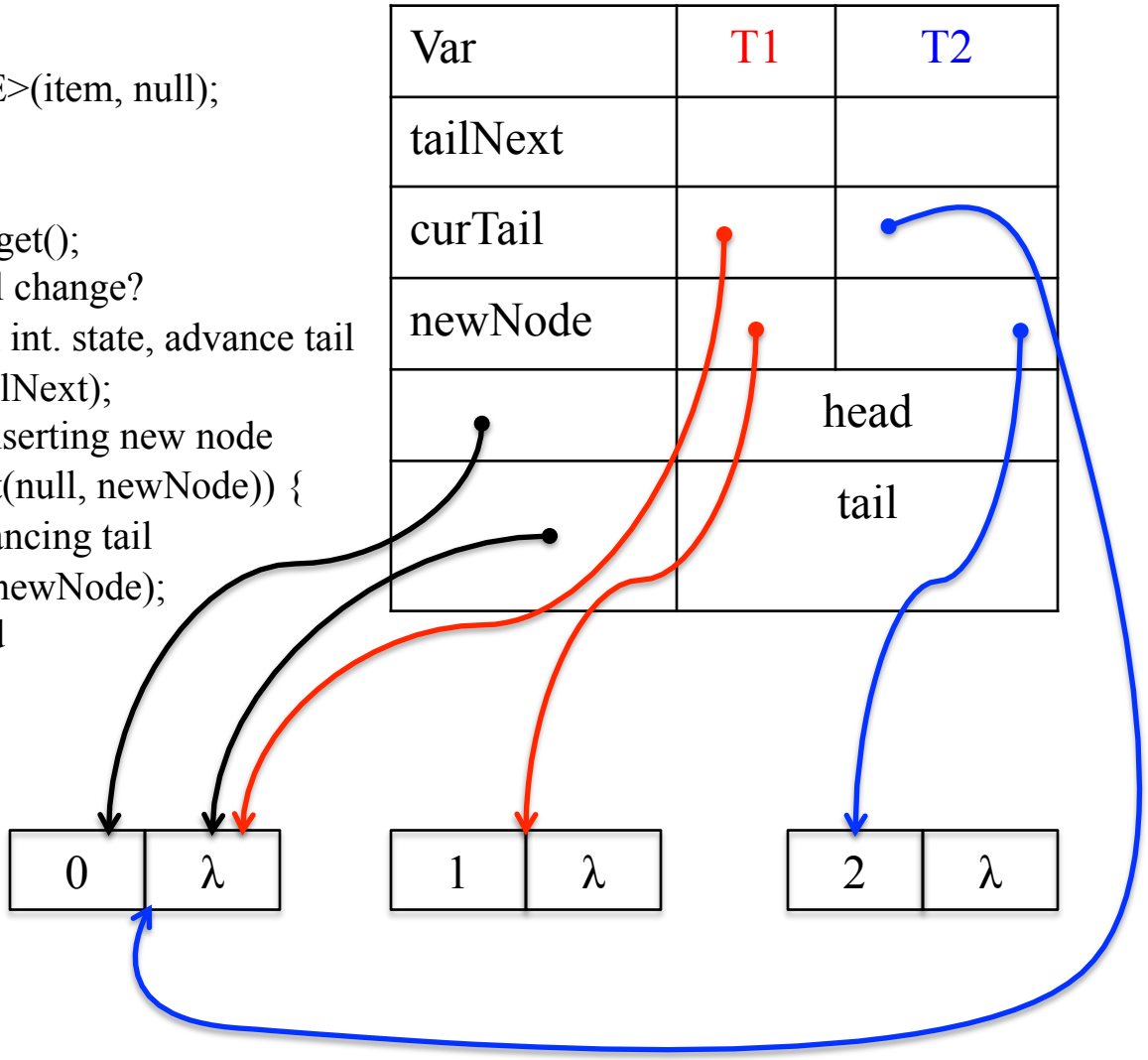
Trace 1

```
   public boolean put(E item) {
1    Node<E> newNode = new Node<E>(item, null);
2    while (true) {
3      Node<E> curTail = tail.get();
4      Node<E> tailNext = curTail.next.get();
5      if (curTail == tail.get()) {// did tail change?
6        if (tailNext != null) { // Queue in int. state, advance tail
7          tail.compareAndSet(curTail, tailNext);
8        else {  // In quiescent state, try inserting new node
9          if (curTail.next.compareAndSet(null, newNode)) {
             // Insertion succeeded, try advancing tail
10             tail.compareAndSet(curTail, newNode);
             // will fail if tail already moved
11           return true;
           }
         }
       }
     }
   }
```



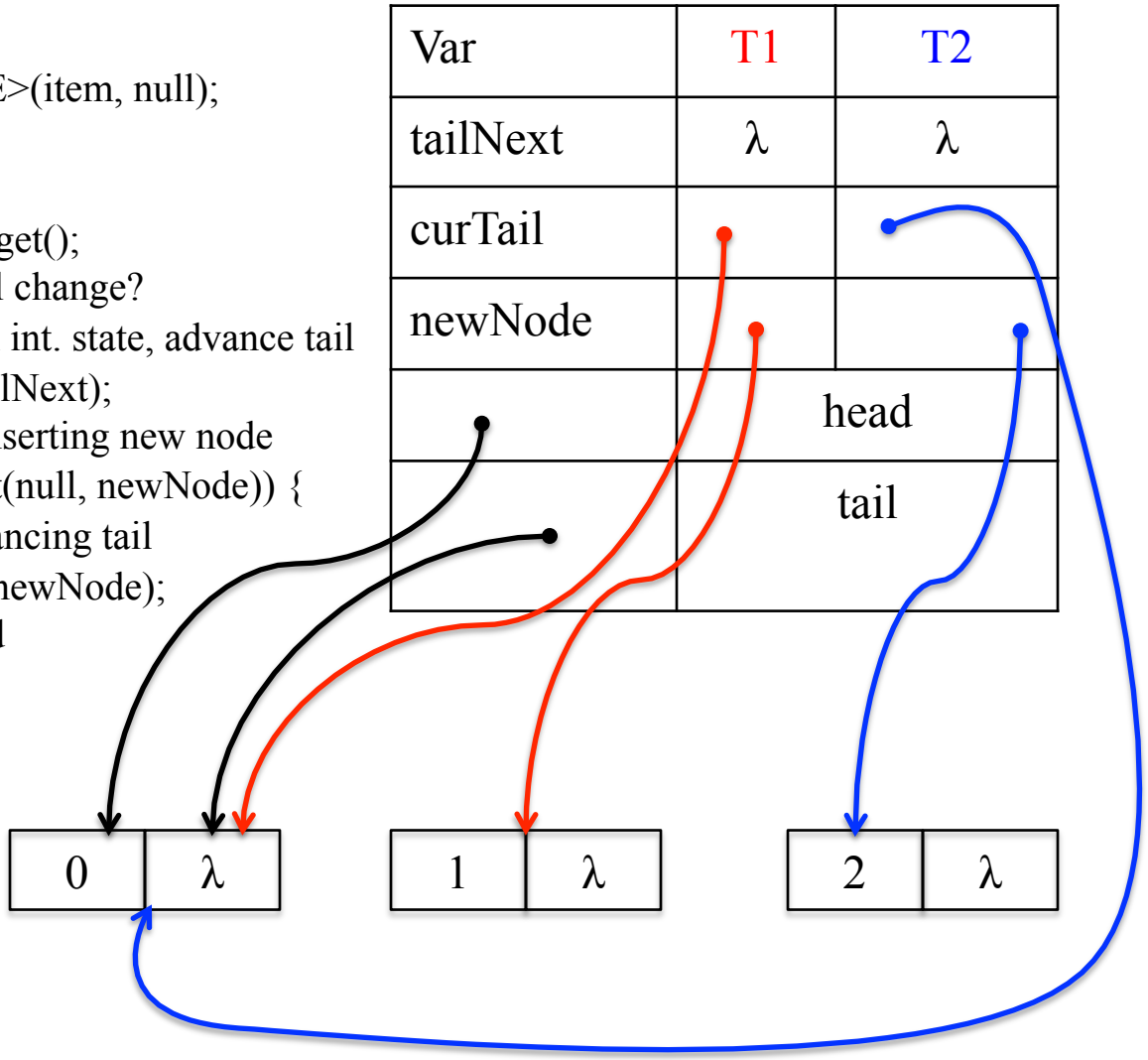
Trace 1

```
    public boolean put(E item) {
1     Node<E> newNode = new Node<E>(item, null);
2     while (true) {
3       Node<E> curTail = tail.get();
4       Node<E> tailNext = curTail.next.get();
5       if (curTail == tail.get()) {// did tail change?
6         if (tailNext != null) { // Queue in int. state, advance tail
7           tail.compareAndSet(curTail, tailNext);
8         else {  // In quiescent state, try inserting new node
9           if (curTail.next.compareAndSet(null, newNode)) {
            // Insertion succeeded, try advancing tail
10            tail.compareAndSet(curTail, newNode);
            // will fail if tail already moved
11          return true;
            }
          }
        }
      }
    }
```



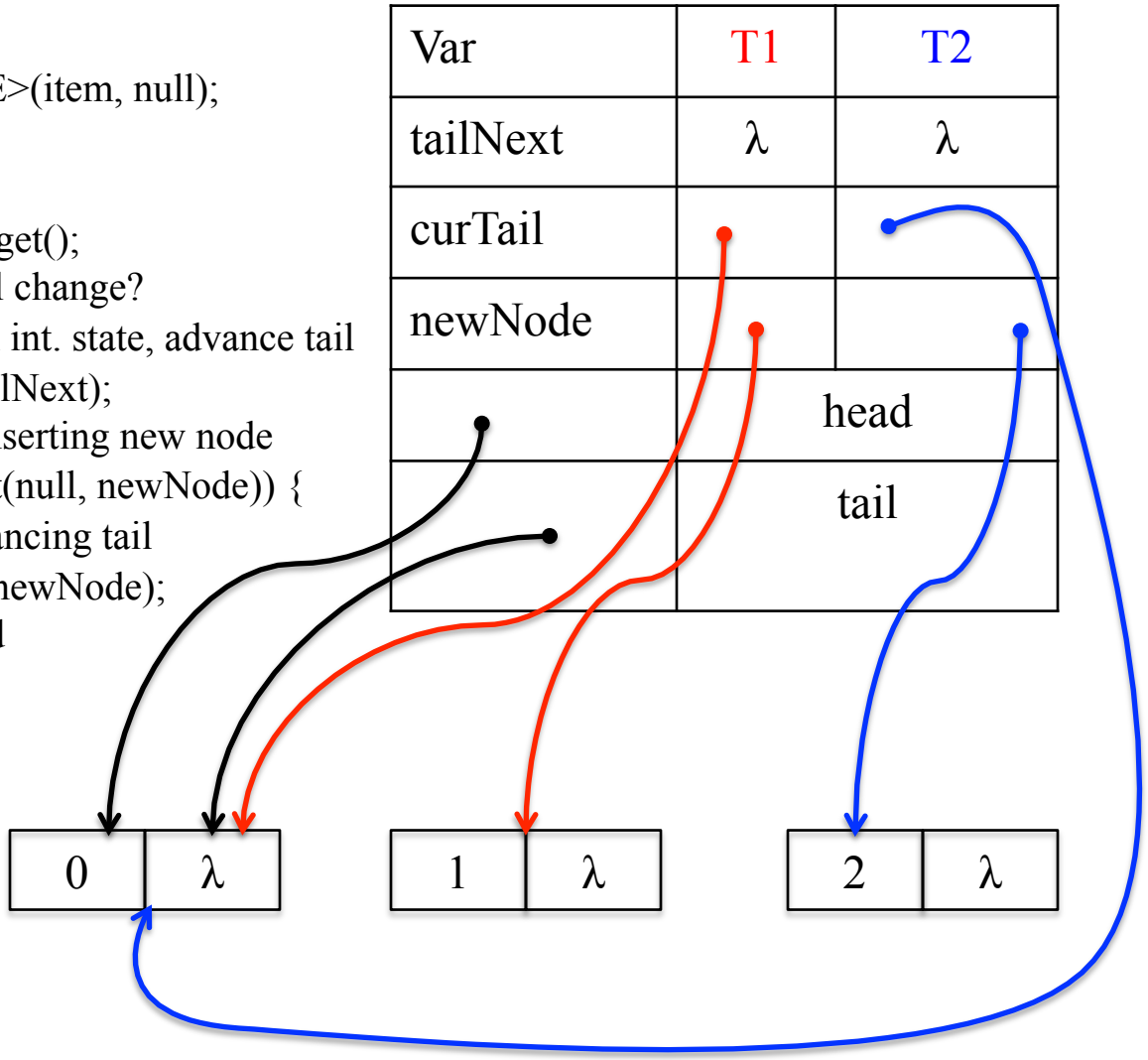
Trace 1

```
    public boolean put(E item) {
1     Node<E> newNode = new Node<E>(item, null);
2     while (true) {
3       Node<E> curTail = tail.get();
4       Node<E> tailNext = curTail.next.get();
5       if (curTail == tail.get()) {// did tail change?
6         if (tailNext != null) { // Queue in int. state, advance tail
7           tail.compareAndSet(curTail, tailNext);
8         else {  // In quiescent state, try inserting new node
9           if (curTail.next.compareAndSet(null, newNode)) {
            // Insertion succeeded, try advancing tail
10            tail.compareAndSet(curTail, newNode);
            // will fail if tail already moved
11          return true;
            }
          }
        }
      }
    }
```



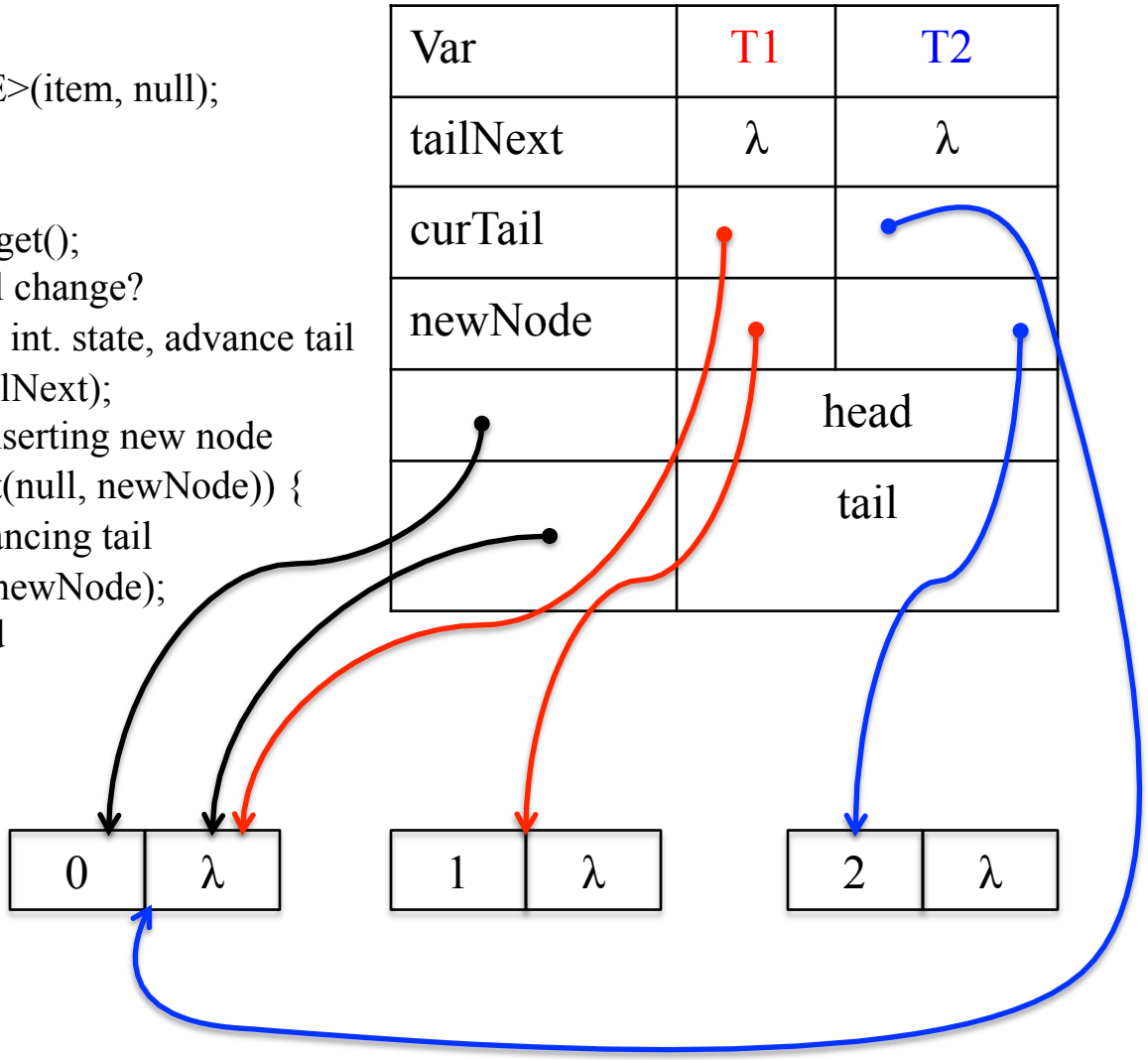
Trace 1

```
    public boolean put(E item) {
1     Node<E> newNode = new Node<E>(item, null);
2     while (true) {
3       Node<E> curTail = tail.get();
4       Node<E> tailNext = curTail.next.get();
5       if (curTail == tail.get()) {// did tail change?
6         if (tailNext != null) { // Queue in int. state, advance tail
7           tail.compareAndSet(curTail, tailNext);
8         else {  // In quiescent state, try inserting new node
9           if (curTail.next.compareAndSet(null, newNode)) {
            // Insertion succeeded, try advancing tail
10            tail.compareAndSet(curTail, newNode);
            // will fail if tail already moved
11          return true;
            }
          }
        }
      }
    }
```



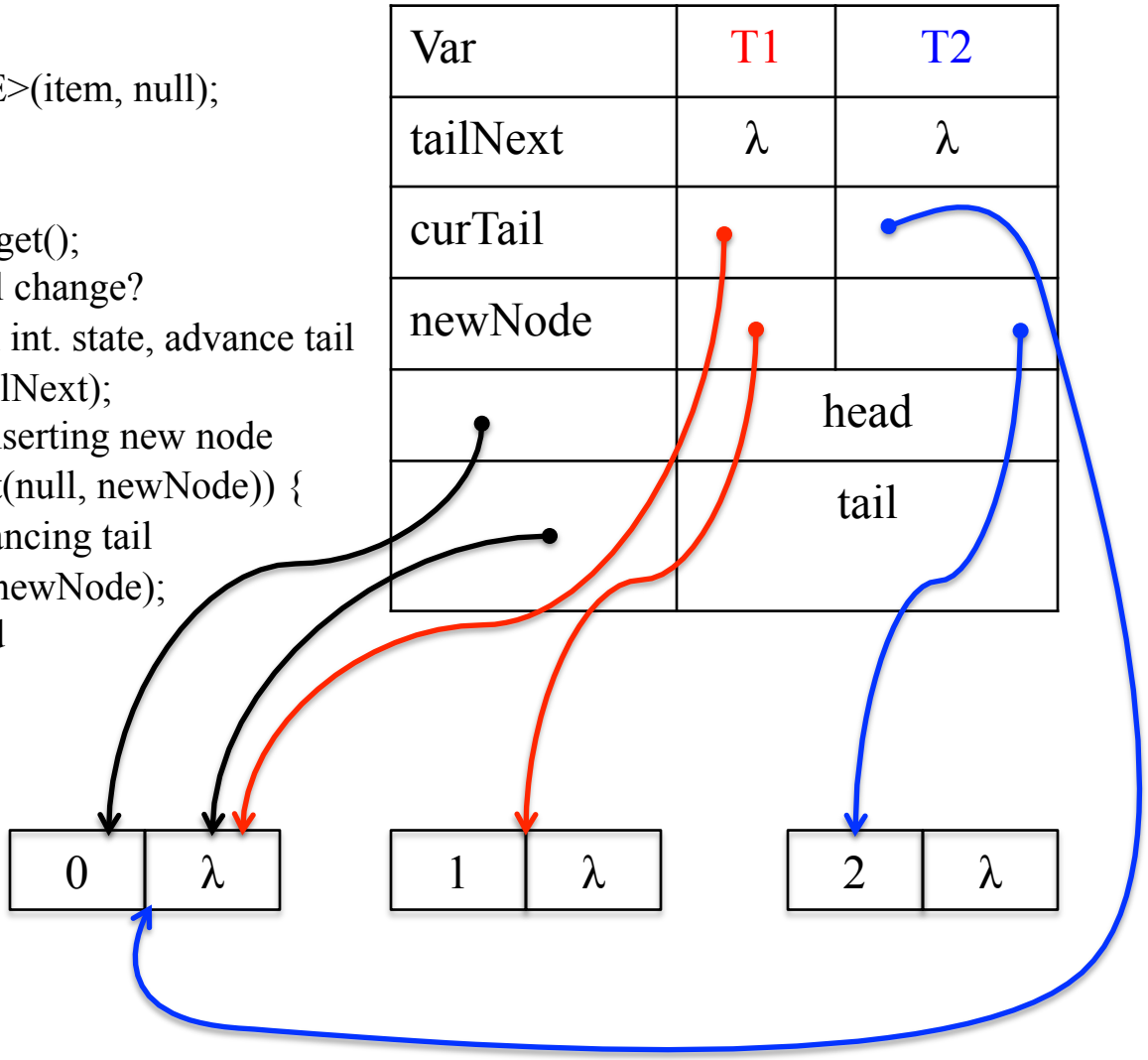
Trace 1

```
    public boolean put(E item) {
1     Node<E> newNode = new Node<E>(item, null);
2     while (true) {
3       Node<E> curTail = tail.get();
4       Node<E> tailNext = curTail.next.get();
5       if (curTail == tail.get()) {// did tail change?
6         if (tailNext != null) { // Queue in int. state, advance tail
7           tail.compareAndSet(curTail, tailNext);
8         else {  // In quiescent state, try inserting new node
9           if (curTail.next.compareAndSet(null, newNode)) {
            // Insertion succeeded, try advancing tail
10            tail.compareAndSet(curTail, newNode);
            // will fail if tail already moved
11          return true;
            }
          }
        }
      }
    }
```



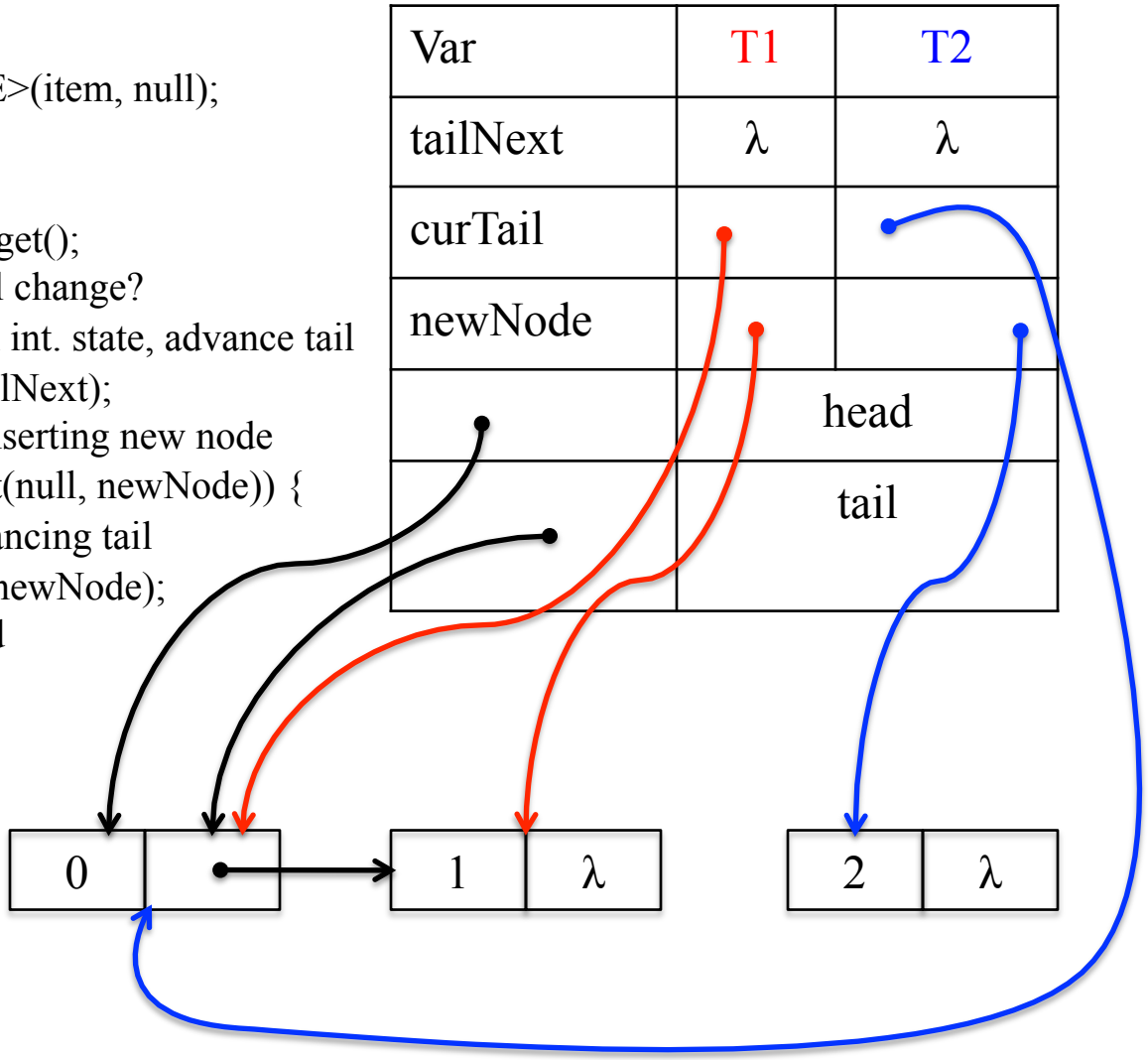
Trace 1: T1 wins

```
public boolean put(E item) {
1     Node<E> newNode = new Node<E>(item, null);
2     while (true) {
3       Node<E> curTail = tail.get();
4       Node<E> tailNext = curTail.next.get();
5       if (curTail == tail.get()) {// did tail change?
6         if (tailNext != null) { // Queue in int. state, advance tail
7           tail.compareAndSet(curTail, tailNext);
8         else {  // In quiescent state, try inserting new node
9           if (curTail.next.compareAndSet(null, newNode)) {
            // Insertion succeeded, try advancing tail
10            tail.compareAndSet(curTail, newNode);
            // will fail if tail already moved
11          return true;
            }
          }
        }
      }
    }
```



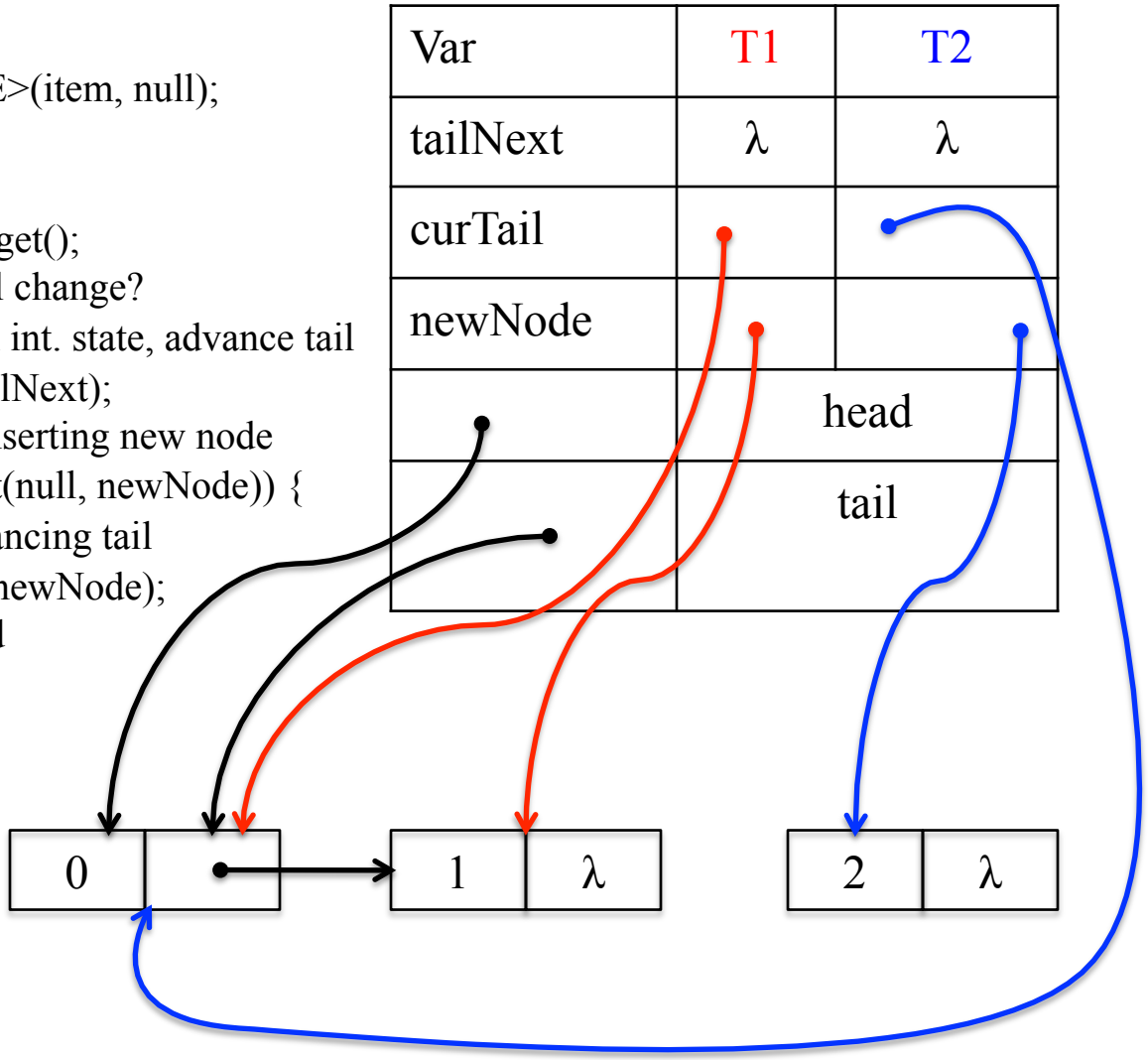
Trace 1: After CAS

```
    public boolean put(E item) {
1     Node<E> newNode = new Node<E>(item, null);
2     while (true) {
3       Node<E> curTail = tail.get();
4       Node<E> tailNext = curTail.next.get();
5       if (curTail == tail.get()) {// did tail change?
6         if (tailNext != null) { // Queue in int. state, advance tail
7           tail.compareAndSet(curTail, tailNext);
8         else {  // In quiescent state, try inserting new node
9           if (curTail.next.compareAndSet(null, newNode)) {
            // Insertion succeeded, try advancing tail
10            tail.compareAndSet(curTail, newNode);
            // will fail if tail already moved
11          return true;
            }
          }
        }
      }
    }
```



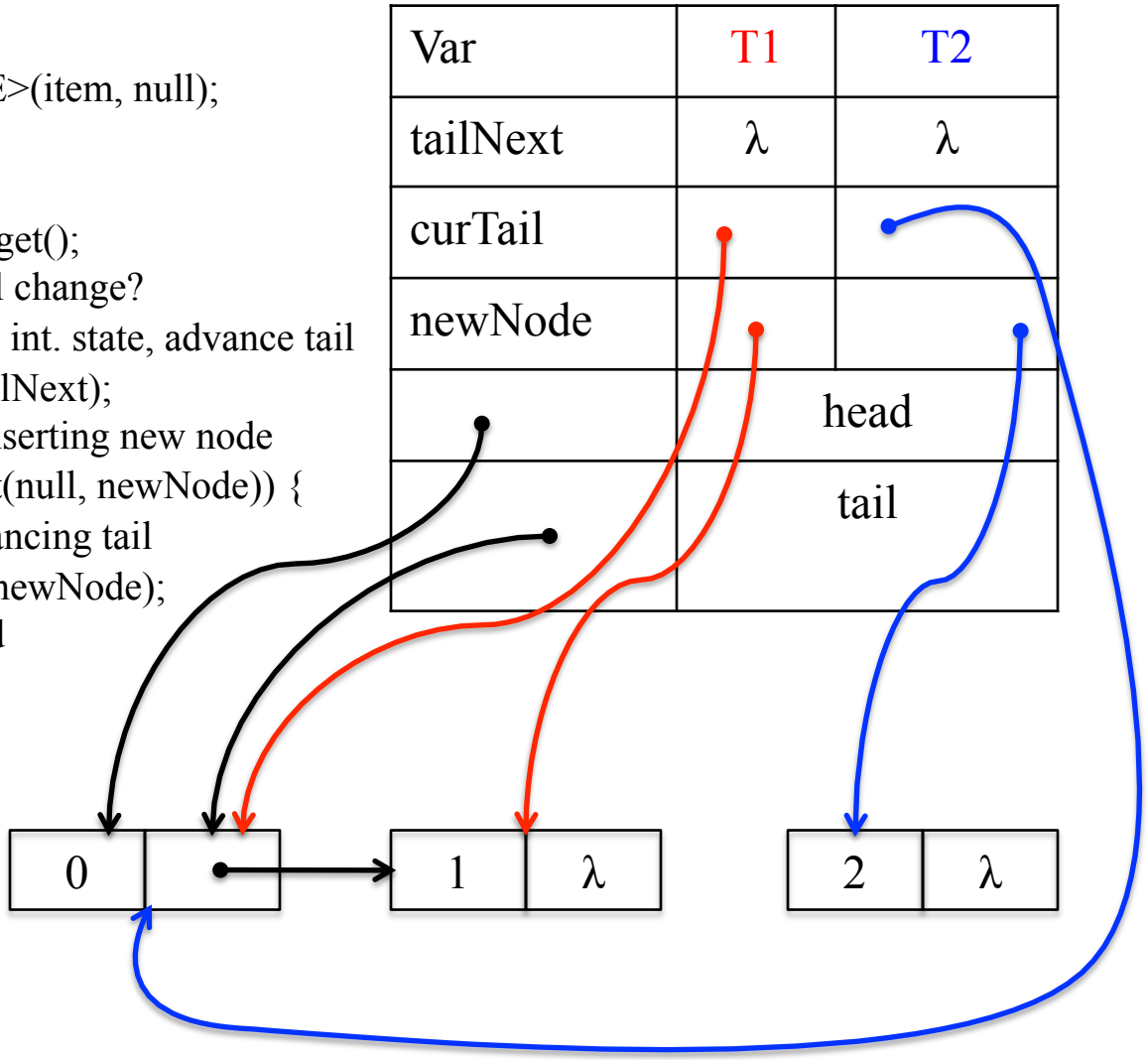
Trace 1: T2 Continues & CAS Fails

```
    public boolean put(E item) {
1     Node<E> newNode = new Node<E>(item, null);
2     while (true) {
3       Node<E> curTail = tail.get();
4       Node<E> tailNext = curTail.next.get();
5       if (curTail == tail.get()) {// did tail change?
6         if (tailNext != null) { // Queue in int. state, advance tail
7           tail.compareAndSet(curTail, tailNext);
8         else {  // In quiescent state, try inserting new node
9           if (curTail.next.compareAndSet(null, newNode)) {
            // Insertion succeeded, try advancing tail
10            tail.compareAndSet(curTail, newNode);
            // will fail if tail already moved
11          return true;
            }
          }
        }
      }
    }
```



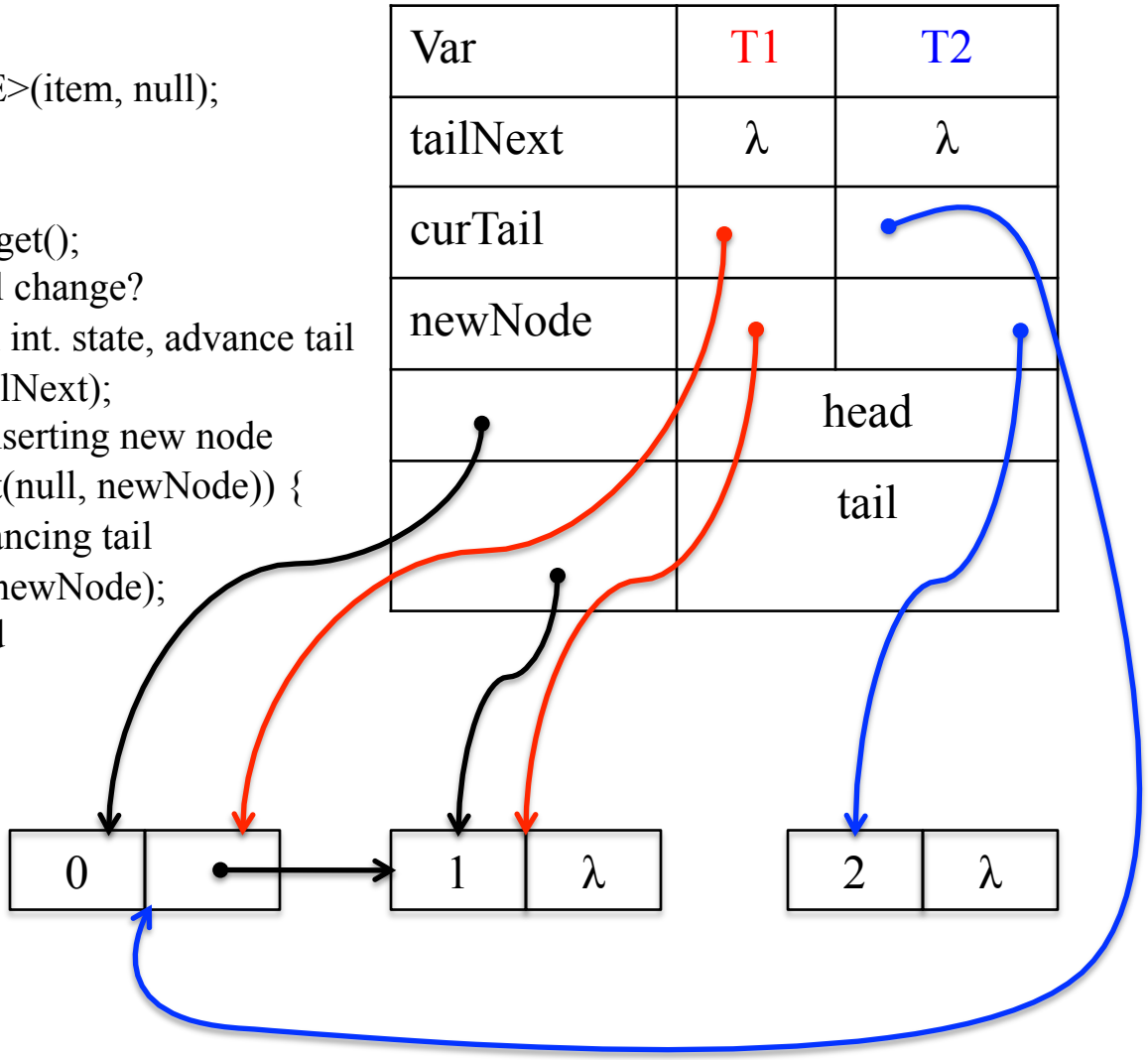
Trace 1: T1 Continues

```
public boolean put(E item) {
1     Node<E> newNode = new Node<E>(item, null);
2     while (true) {
3       Node<E> curTail = tail.get();
4       Node<E> tailNext = curTail.next.get();
5       if (curTail == tail.get()) {// did tail change?
6         if (tailNext != null) { // Queue in int. state, advance tail
7           tail.compareAndSet(curTail, tailNext);
8         else {  // In quiescent state, try inserting new node
9           if (curTail.next.compareAndSet(null, newNode)) {
            // Insertion succeeded, try advancing tail
10            tail.compareAndSet(curTail, newNode);
            // will fail if tail already moved
11          return true;
            }
          }
        }
      }
    }
```



Trace 1: After CAS

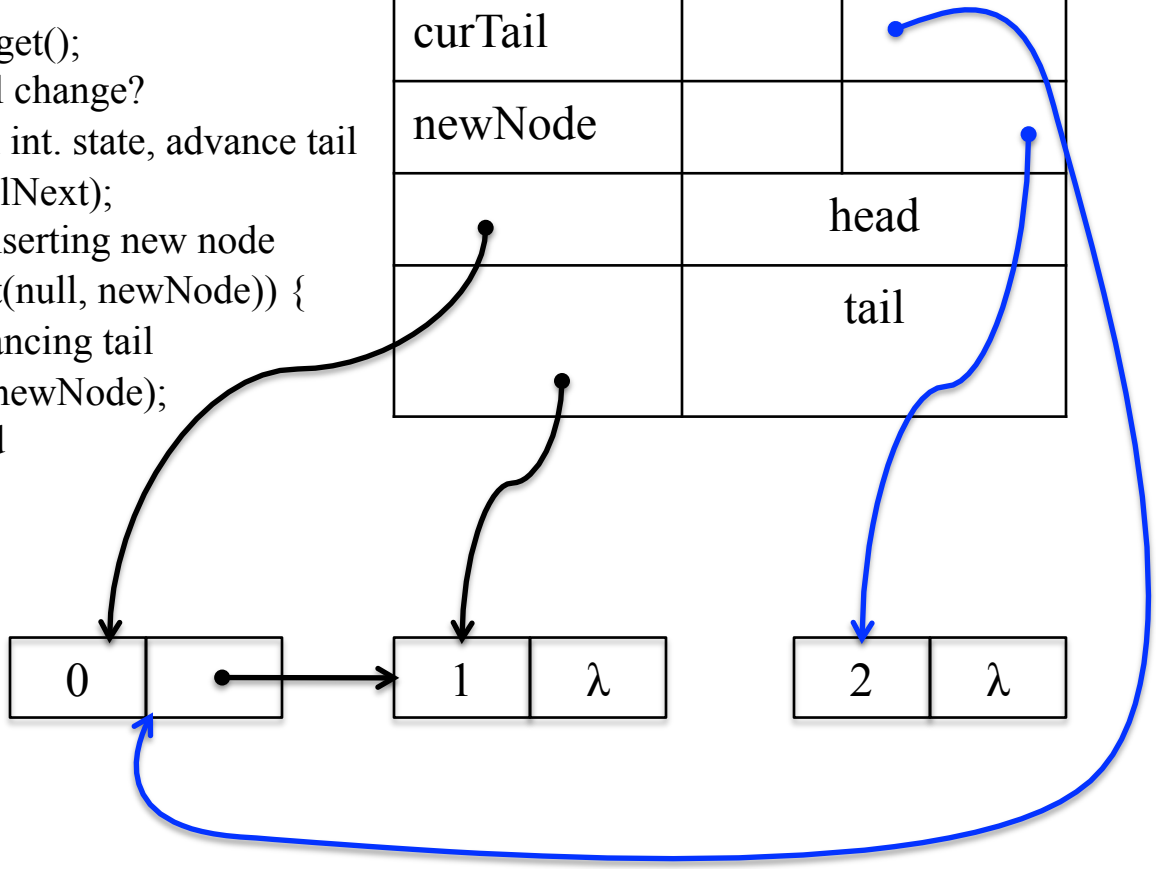
```
public boolean put(E item) {
1     Node<E> newNode = new Node<E>(item, null);
2     while (true) {
3       Node<E> curTail = tail.get();
4       Node<E> tailNext = curTail.next.get();
5       if (curTail == tail.get()) {// did tail change?
6         if (tailNext != null) { // Queue in int. state, advance tail
7           tail.compareAndSet(curTail, tailNext);
8         else {  // In quiescent state, try inserting new node
9           if (curTail.next.compareAndSet(null, newNode)) {
            // Insertion succeeded, try advancing tail
10            tail.compareAndSet(curTail, newNode);
            // will fail if tail already moved
11          return true;
            }
          }
        }
      }
    }
```



Trace 1: T1 exits

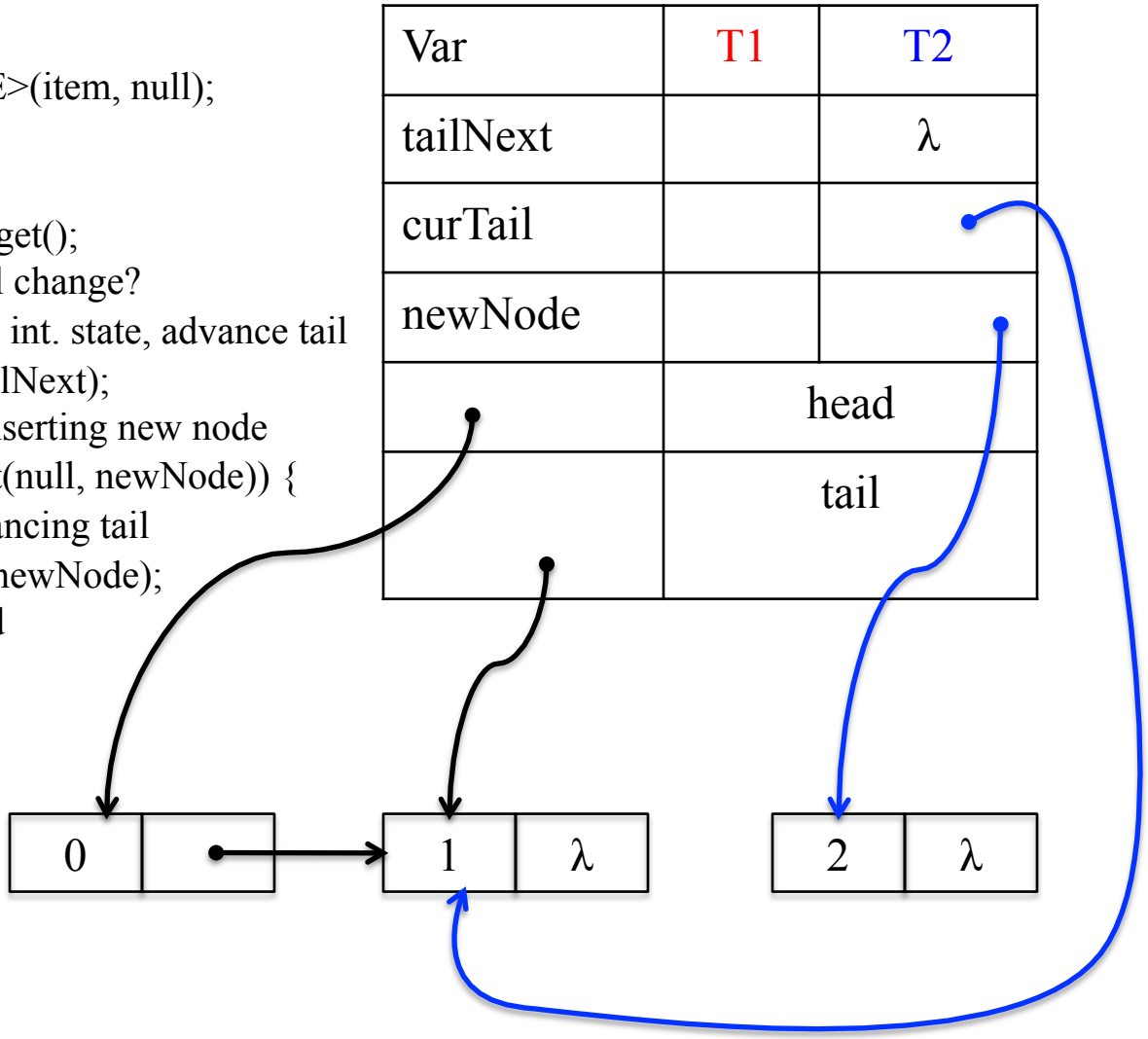
```
public boolean put(E item) {
1   Node<E> newNode = new Node<E>(item, null);
2   while (true) {
3     Node<E> curTail = tail.get();
4     Node<E> tailNext = curTail.next.get();
5     if (curTail == tail.get()) {// did tail change?
6       if (tailNext != null) { // Queue in int. state, advance tail
7         tail.compareAndSet(curTail, tailNext);
8       else {  // In quiescent state, try inserting new node
9         if (curTail.next.compareAndSet(null, newNode)) {
          // Insertion succeeded, try advancing tail
10          tail.compareAndSet(curTail, newNode);
          // will fail if tail already moved
11        return true;
          }
        }
      }
    }
  }
```

Var	T1	T2
tailNext		λ
curTail		
newNode		
	head	
	tail	



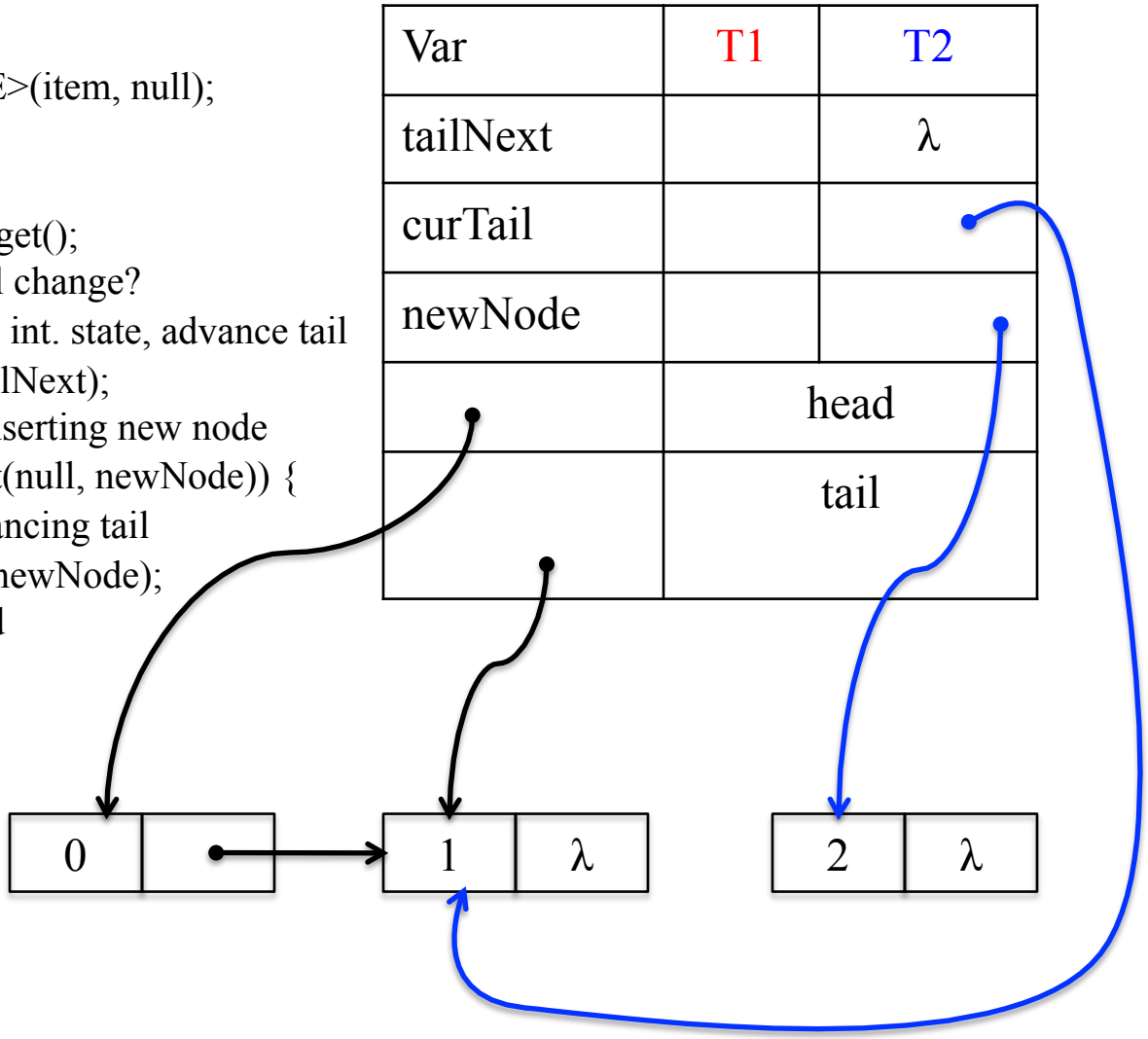
Trace 1: T2 Continues

```
    public boolean put(E item) {
1     Node<E> newNode = new Node<E>(item, null);
2     while (true) {
3       Node<E> curTail = tail.get();
4       Node<E> tailNext = curTail.next.get();
5       if (curTail == tail.get()) {// did tail change?
6         if (tailNext != null) { // Queue in int. state, advance tail
7           tail.compareAndSet(curTail, tailNext);
8         else {  // In quiescent state, try inserting new node
9           if (curTail.next.compareAndSet(null, newNode)) {
            // Insertion succeeded, try advancing tail
10            tail.compareAndSet(curTail, newNode);
            // will fail if tail already moved
11          return true;
            }
          }
        }
      }
    }
```



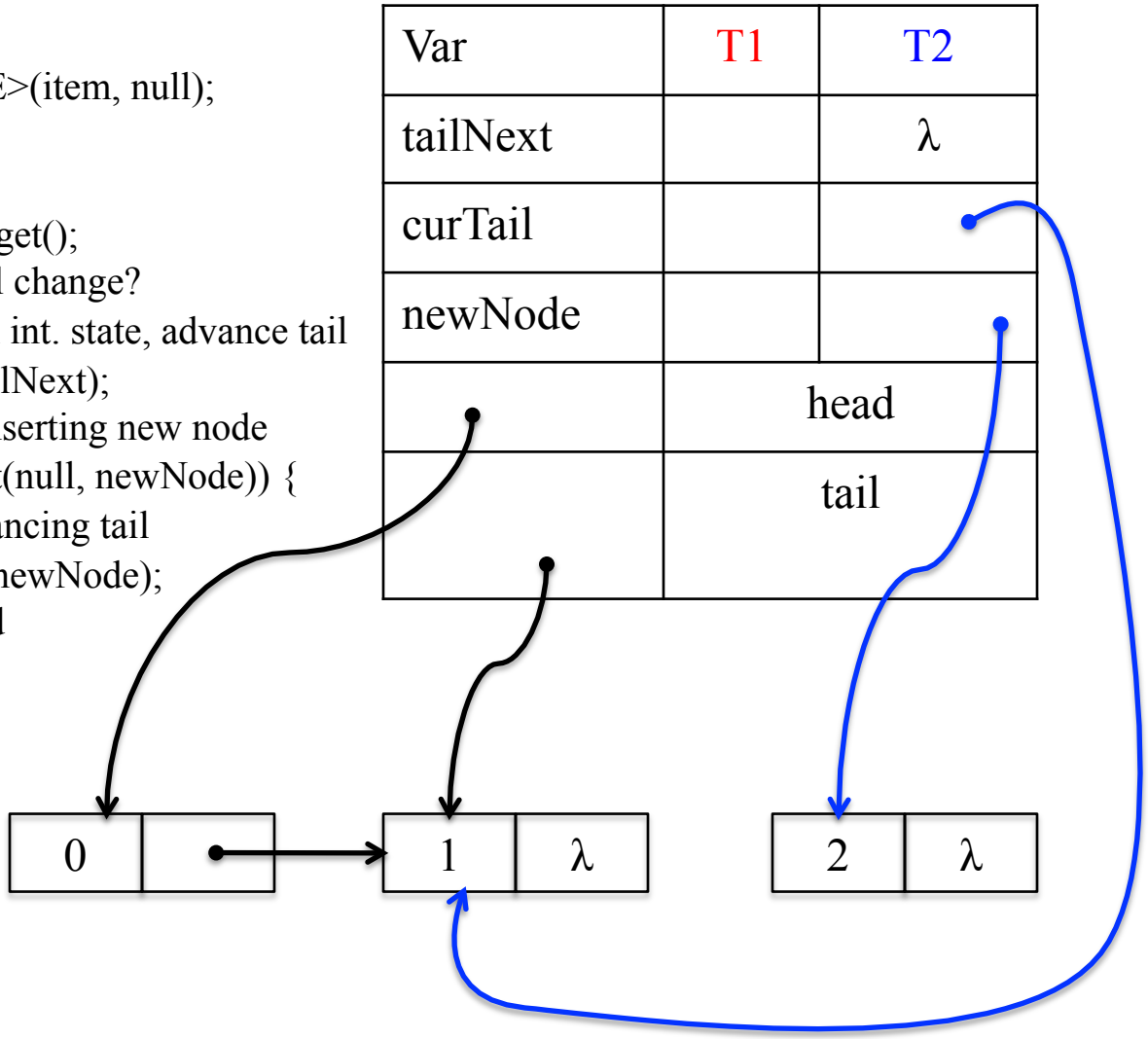
Trace 1: T2 Continues

```
    public boolean put(E item) {
1     Node<E> newNode = new Node<E>(item, null);
2     while (true) {
3       Node<E> curTail = tail.get();
4       Node<E> tailNext = curTail.next.get();
5       if (curTail == tail.get()) {// did tail change?
6         if (tailNext != null) { // Queue in int. state, advance tail
7           tail.compareAndSet(curTail, tailNext);
8         else {  // In quiescent state, try inserting new node
9           if (curTail.next.compareAndSet(null, newNode)) {
            // Insertion succeeded, try advancing tail
10            tail.compareAndSet(curTail, newNode);
            // will fail if tail already moved
11          return true;
            }
          }
        }
      }
    }
```



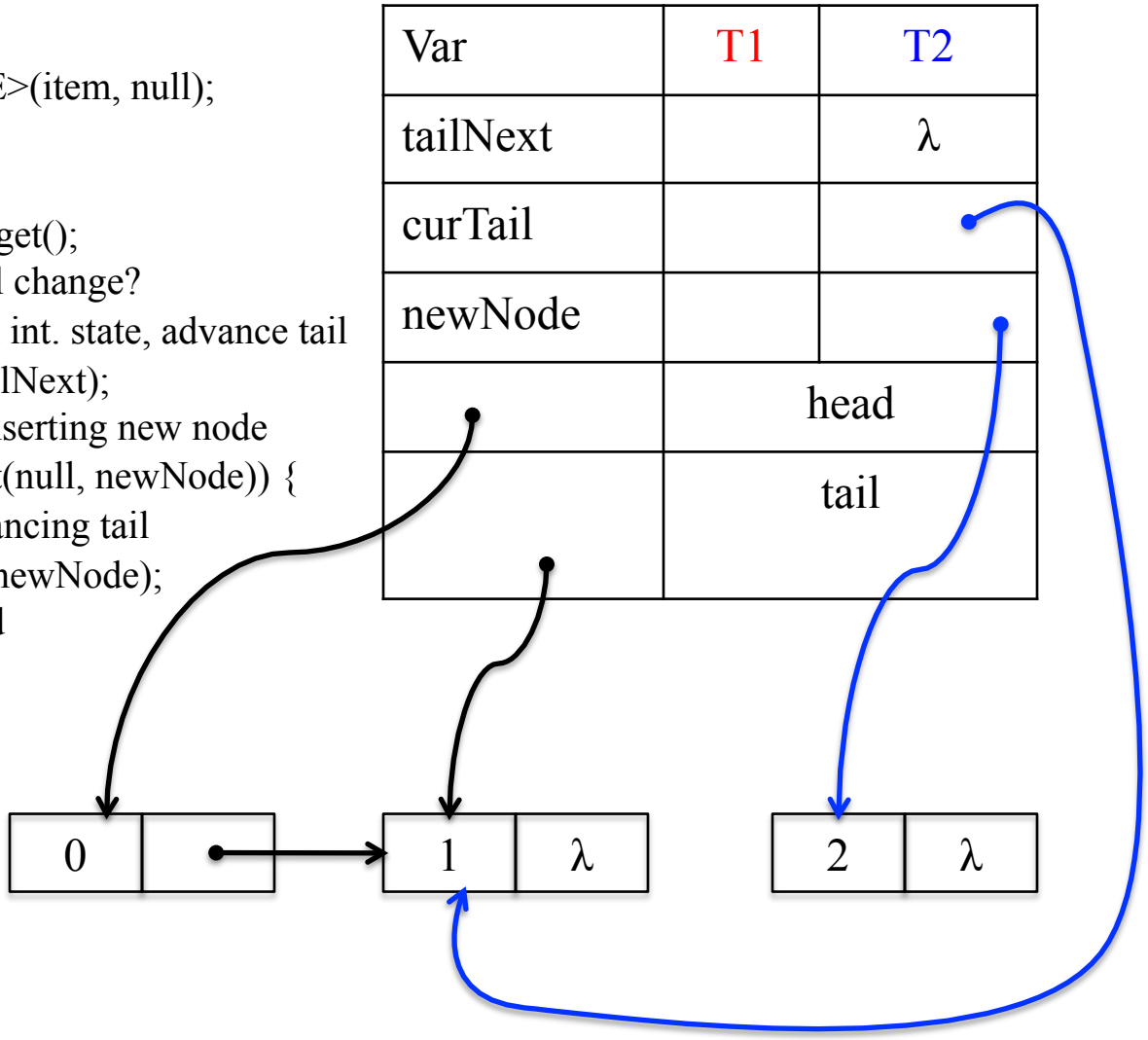
Trace 1: T2 Continues

```
    public boolean put(E item) {
1     Node<E> newNode = new Node<E>(item, null);
2     while (true) {
3       Node<E> curTail = tail.get();
4       Node<E> tailNext = curTail.next.get();
5       if (curTail == tail.get()) {// did tail change?
6         if (tailNext != null) { // Queue in int. state, advance tail
7           tail.compareAndSet(curTail, tailNext);
8         else {  // In quiescent state, try inserting new node
9           if (curTail.next.compareAndSet(null, newNode)) {
            // Insertion succeeded, try advancing tail
10            tail.compareAndSet(curTail, newNode);
            // will fail if tail already moved
11          return true;
            }
          }
        }
      }
    }
```



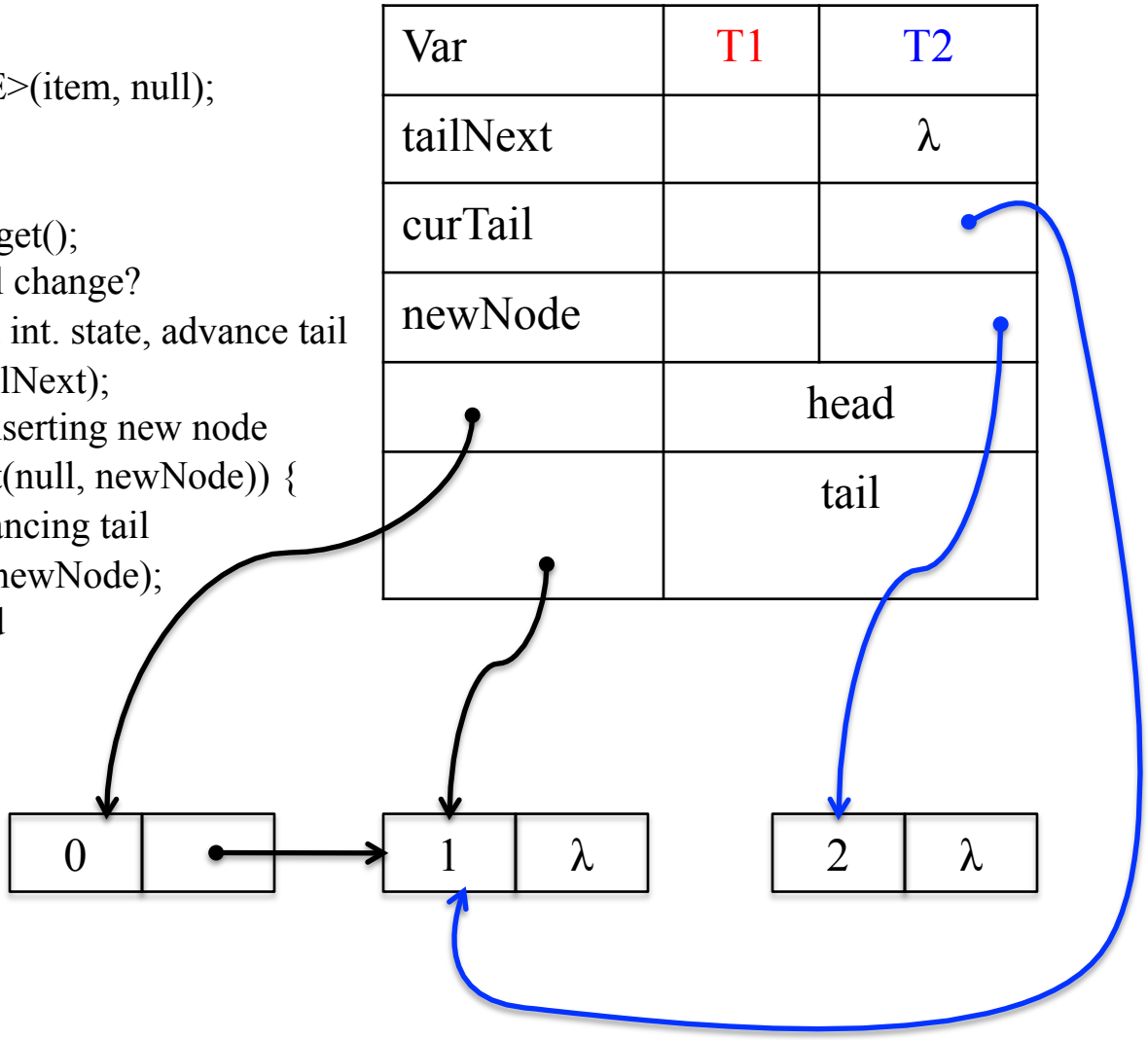
Trace 1: T2 Continues

```
    public boolean put(E item) {
1     Node<E> newNode = new Node<E>(item, null);
2     while (true) {
3       Node<E> curTail = tail.get();
4       Node<E> tailNext = curTail.next.get();
5       if (curTail == tail.get()) {// did tail change?
6         if (tailNext != null) { // Queue in int. state, advance tail
7           tail.compareAndSet(curTail, tailNext);
8         else {  // In quiescent state, try inserting new node
9           if (curTail.next.compareAndSet(null, newNode)) {
            // Insertion succeeded, try advancing tail
10            tail.compareAndSet(curTail, newNode);
            // will fail if tail already moved
11          return true;
            }
          }
        }
      }
    }
```



Trace 1: T2 Continues

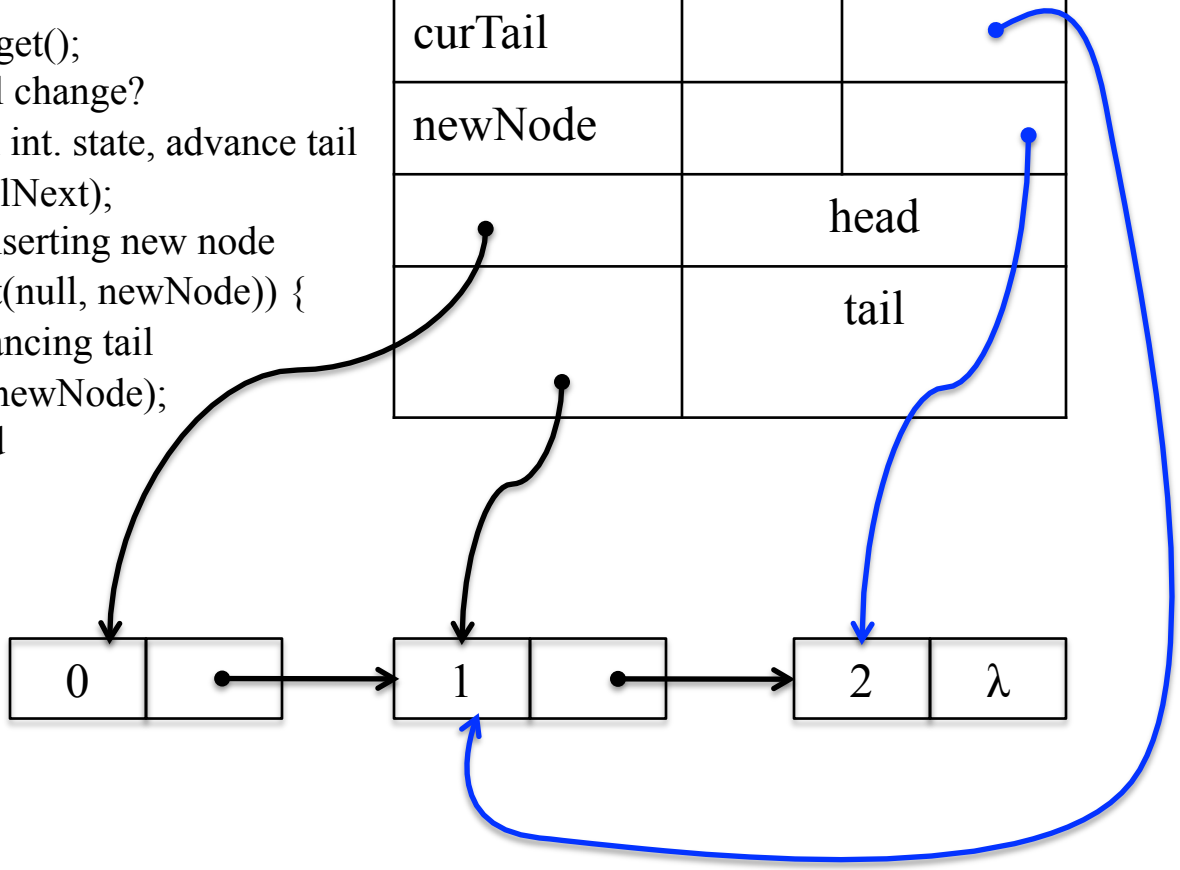
```
    public boolean put(E item) {
1     Node<E> newNode = new Node<E>(item, null);
2     while (true) {
3       Node<E> curTail = tail.get();
4       Node<E> tailNext = curTail.next.get();
5       if (curTail == tail.get()) {// did tail change?
6         if (tailNext != null) { // Queue in int. state, advance tail
7           tail.compareAndSet(curTail, tailNext);
8         else {  // In quiescent state, try inserting new node
9           if (curTail.next.compareAndSet(null, newNode)) {
            // Insertion succeeded, try advancing tail
10            tail.compareAndSet(curTail, newNode);
            // will fail if tail already moved
11          return true;
            }
          }
        }
      }
    }
```



Trace 1: After CAS

```
    public boolean put(E item) {
1     Node<E> newNode = new Node<E>(item, null);
2     while (true) {
3       Node<E> curTail = tail.get();
4       Node<E> tailNext = curTail.next.get();
5       if (curTail == tail.get()) {// did tail change?
6         if (tailNext != null) { // Queue in int. state, advance tail
7           tail.compareAndSet(curTail, tailNext);
8         else {  // In quiescent state, try inserting new node
9           if (curTail.next.compareAndSet(null, newNode)) {
            // Insertion succeeded, try advancing tail
10            tail.compareAndSet(curTail, newNode);
            // will fail if tail already moved
11          return true;
            }
          }
        }
      }
    }
```

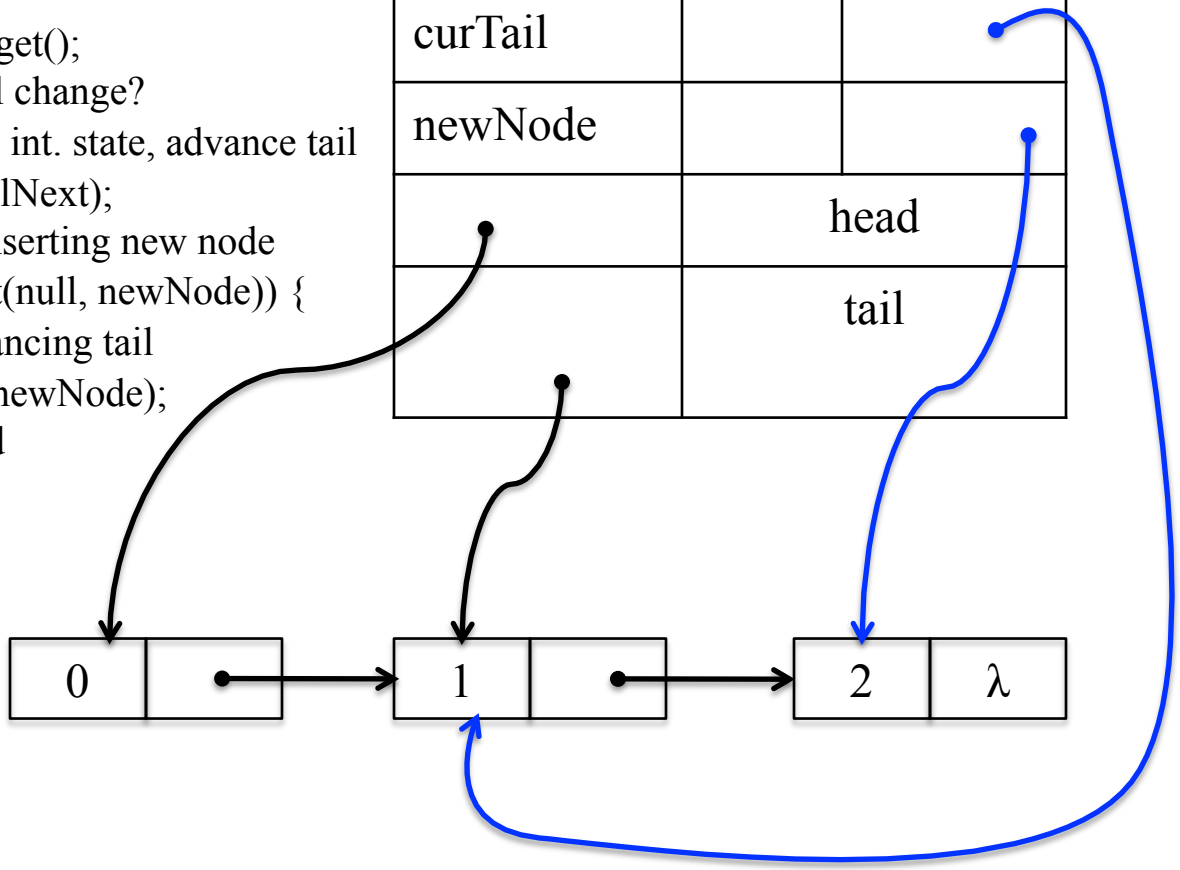
Var	T1	T2
tailNext		λ
curTail		
newNode		
	head	
	tail	



Trace 1: T2 Continues

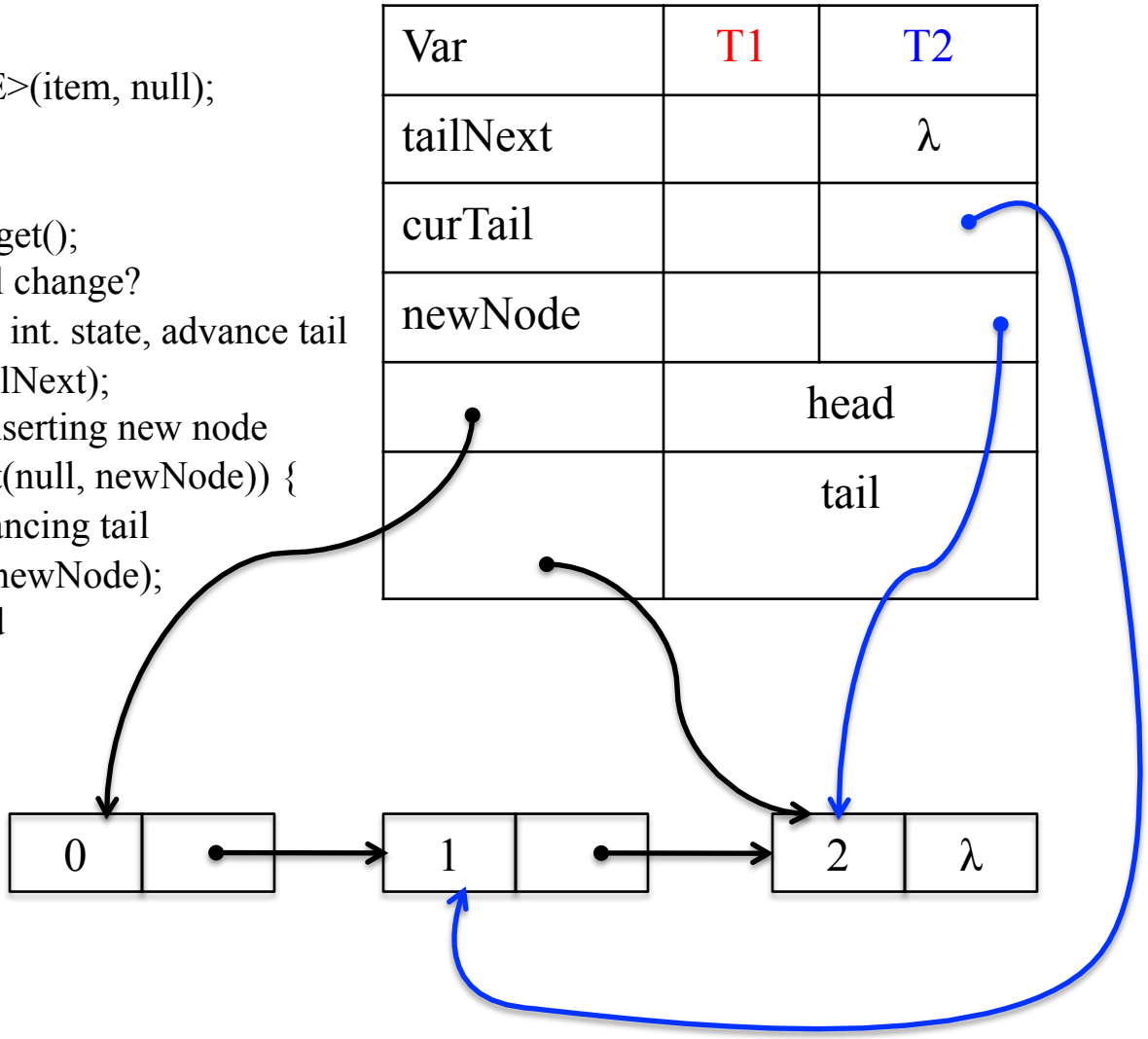
```
    public boolean put(E item) {
1     Node<E> newNode = new Node<E>(item, null);
2     while (true) {
3       Node<E> curTail = tail.get();
4       Node<E> tailNext = curTail.next.get();
5       if (curTail == tail.get()) {// did tail change?
6         if (tailNext != null) { // Queue in int. state, advance tail
7           tail.compareAndSet(curTail, tailNext);
8         else {  // In quiescent state, try inserting new node
9           if (curTail.next.compareAndSet(null, newNode)) {
            // Insertion succeeded, try advancing tail
10            tail.compareAndSet(curTail, newNode);
            // will fail if tail already moved
11          return true;
            }
          }
        }
      }
    }
```

Var	T1	T2
tailNext		λ
curTail		
newNode		
	head	
	tail	



Trace 1: T2 Continues

```
    public boolean put(E item) {
1     Node<E> newNode = new Node<E>(item, null);
2     while (true) {
3       Node<E> curTail = tail.get();
4       Node<E> tailNext = curTail.next.get();
5       if (curTail == tail.get()) {// did tail change?
6         if (tailNext != null) { // Queue in int. state, advance tail
7           tail.compareAndSet(curTail, tailNext);
8         else {  // In quiescent state, try inserting new node
9           if (curTail.next.compareAndSet(null, newNode)) {
            // Insertion succeeded, try advancing tail
10            tail.compareAndSet(curTail, newNode);
            // will fail if tail already moved
11          return true;
            }
          }
        }
      }
    }
```



Trace 1: T2 Exits

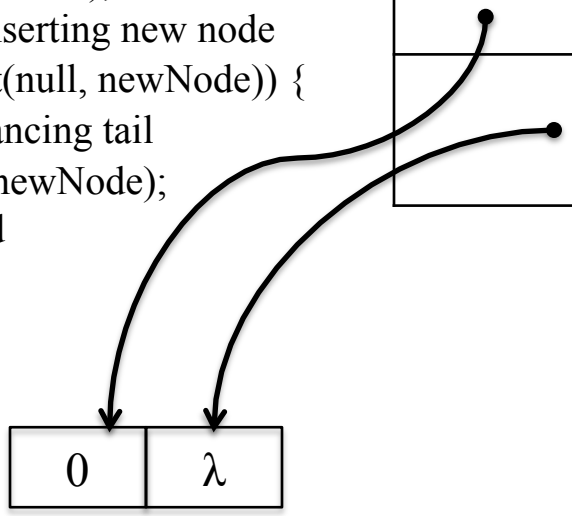
```
    public boolean put(E item) {
1     Node<E> newNode = new Node<E>(item, null);
2     while (true) {
3       Node<E> curTail = tail.get();
4       Node<E> tailNext = curTail.next.get();
5       if (curTail == tail.get()) {// did tail change?
6         if (tailNext != null) { // Queue in int. state, advance tail
7           tail.compareAndSet(curTail, tailNext);
8         else {  // In quiescent state, try inserting new node
9           if (curTail.next.compareAndSet(null, newNode)) {
            // Insertion succeeded, try advancing tail
10            tail.compareAndSet(curTail, newNode);
            // will fail if tail already moved
11          return true;
          }
        }
      }
    }
  }
```

Var	T1	T2
tailNext		
curTail		
newNode		
	head	
	tail	

Trace 2

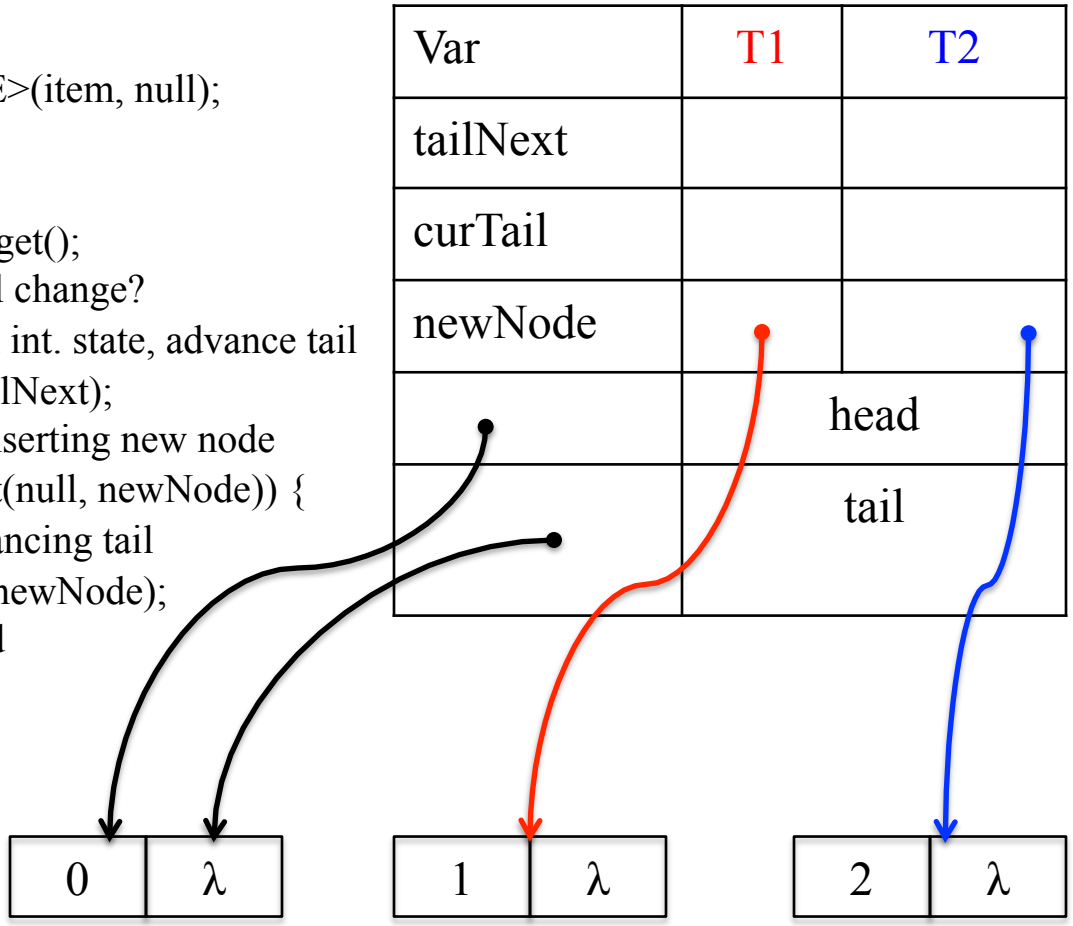
```
    public boolean put(E item) {
1     Node<E> newNode = new Node<E>(item, null);
2     while (true) {
3       Node<E> curTail = tail.get();
4       Node<E> tailNext = curTail.next.get();
5       if (curTail == tail.get()) {// did tail change?
6         if (tailNext != null) { // Queue in int. state, advance tail
7           tail.compareAndSet(curTail, tailNext);
8         else {  // In quiescent state, try inserting new node
9           if (curTail.next.compareAndSet(null, newNode)) {
            // Insertion succeeded, try advancing tail
10            tail.compareAndSet(curTail, newNode);
            // will fail if tail already moved
11          return true;
            }
          }
        }
      }
    }
```

Var	T1	T2
tailNext		
curTail		
newNode		
	head	
	tail	



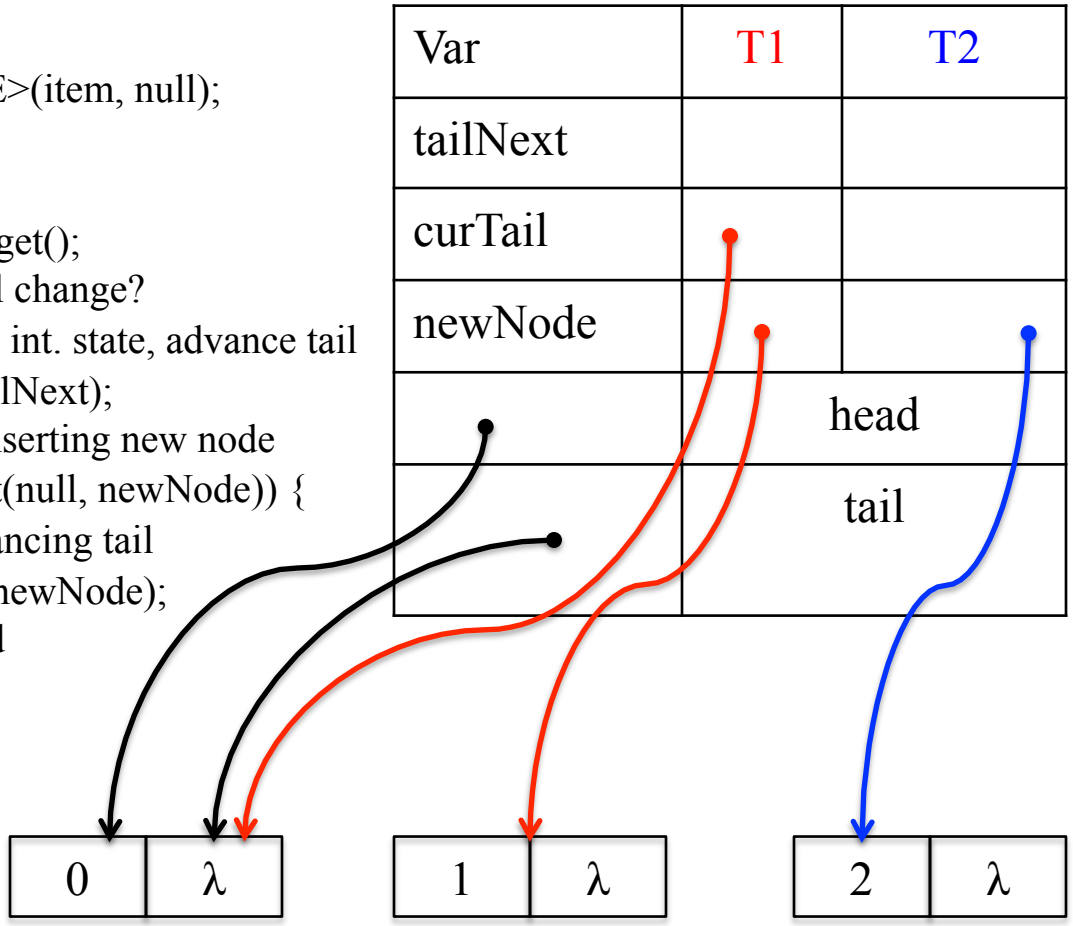
Trace 2

```
    public boolean put(E item) {
1     Node<E> newNode = new Node<E>(item, null);
2     while (true) {
3       Node<E> curTail = tail.get();
4       Node<E> tailNext = curTail.next.get();
5       if (curTail == tail.get()) {// did tail change?
6         if (tailNext != null) { // Queue in int. state, advance tail
7           tail.compareAndSet(curTail, tailNext);
8         else {  // In quiescent state, try inserting new node
9           if (curTail.next.compareAndSet(null, newNode)) {
            // Insertion succeeded, try advancing tail
10            tail.compareAndSet(curTail, newNode);
            // will fail if tail already moved
11          return true;
            }
          }
        }
      }
    }
```



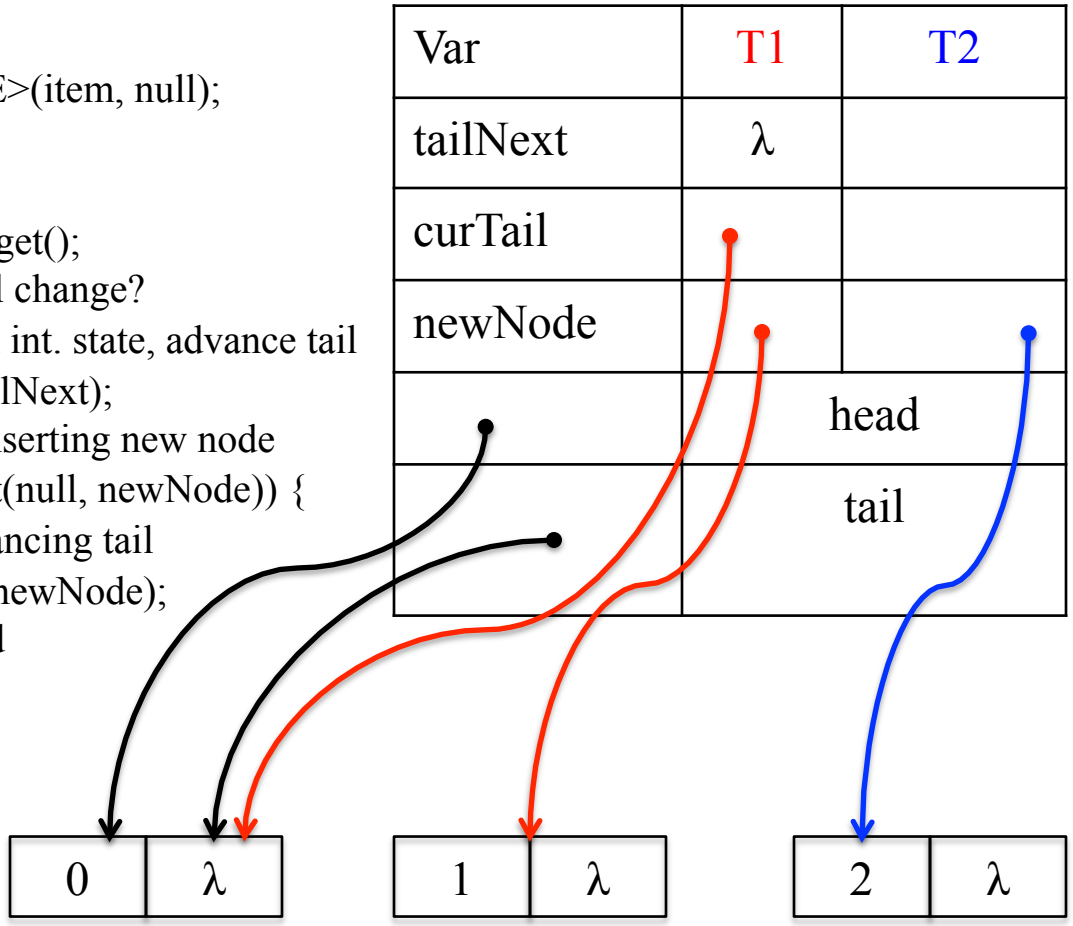
Trace 2

```
    public boolean put(E item) {
1     Node<E> newNode = new Node<E>(item, null);
2     while (true) {
3       Node<E> curTail = tail.get();
4       Node<E> tailNext = curTail.next.get();
5       if (curTail == tail.get()) {// did tail change?
6         if (tailNext != null) { // Queue in int. state, advance tail
7           tail.compareAndSet(curTail, tailNext);
8         else {  // In quiescent state, try inserting new node
9           if (curTail.next.compareAndSet(null, newNode)) {
              // Insertion succeeded, try advancing tail
10              tail.compareAndSet(curTail, newNode);
              // will fail if tail already moved
11            return true;
            }
          }
        }
      }
    }
```



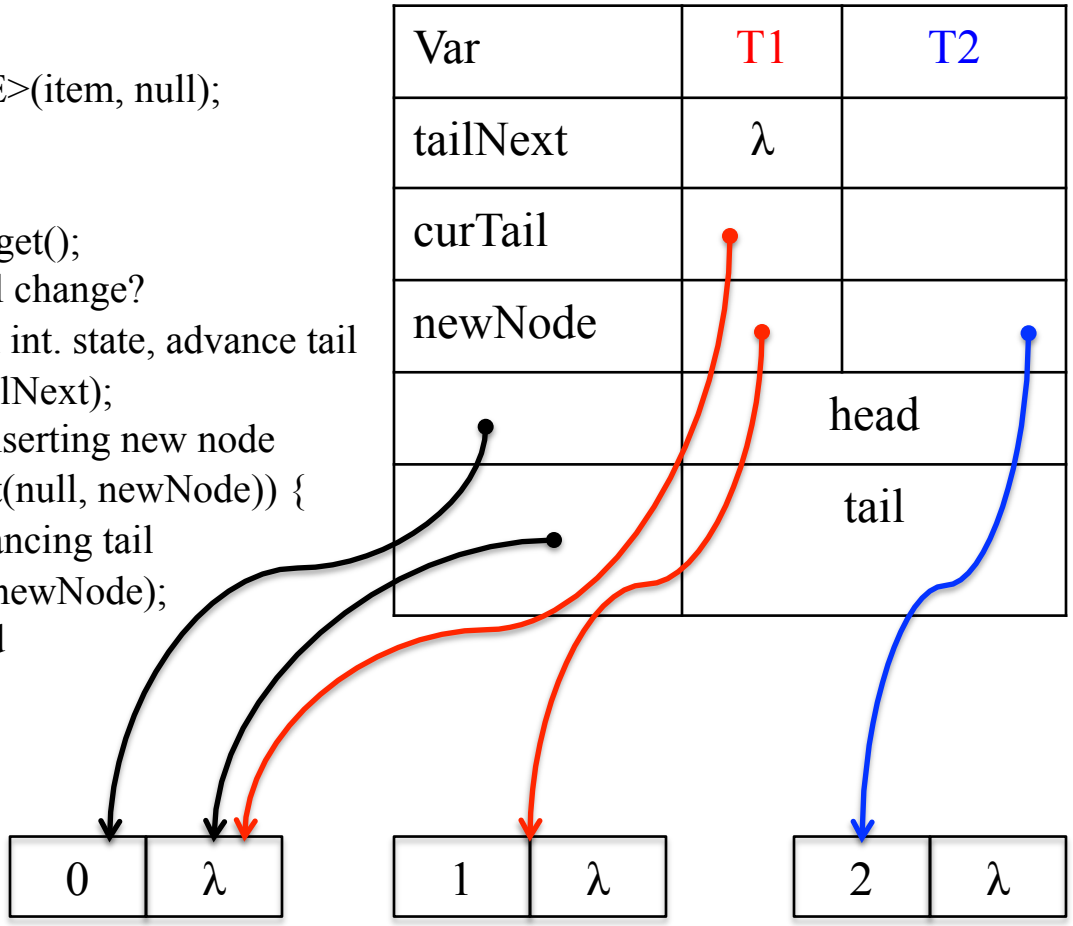
Trace 2

```
    public boolean put(E item) {
1     Node<E> newNode = new Node<E>(item, null);
2     while (true) {
3       Node<E> curTail = tail.get();
4       Node<E> tailNext = curTail.next.get();
5       if (curTail == tail.get()) {// did tail change?
6         if (tailNext != null) { // Queue in int. state, advance tail
7           tail.compareAndSet(curTail, tailNext);
8         else {  // In quiescent state, try inserting new node
9           if (curTail.next.compareAndSet(null, newNode)) {
              // Insertion succeeded, try advancing tail
10              tail.compareAndSet(curTail, newNode);
              // will fail if tail already moved
11            return true;
            }
          }
        }
      }
    }
```



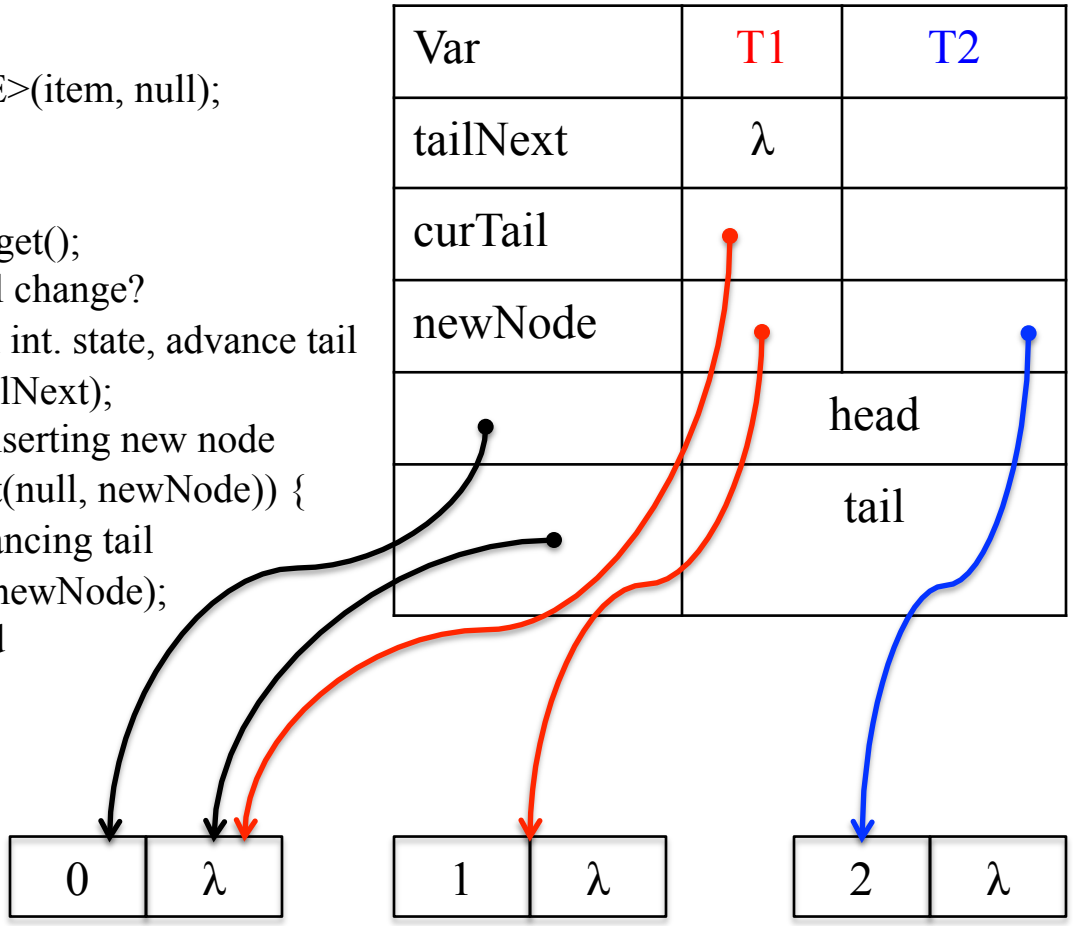
Trace 2

```
    public boolean put(E item) {
1     Node<E> newNode = new Node<E>(item, null);
2     while (true) {
3       Node<E> curTail = tail.get();
4       Node<E> tailNext = curTail.next.get();
5       if (curTail == tail.get()) {// did tail change?
6         if (tailNext != null) { // Queue in int. state, advance tail
7           tail.compareAndSet(curTail, tailNext);
8         else {  // In quiescent state, try inserting new node
9           if (curTail.next.compareAndSet(null, newNode)) {
              // Insertion succeeded, try advancing tail
10              tail.compareAndSet(curTail, newNode);
              // will fail if tail already moved
11            return true;
            }
          }
        }
      }
    }
```



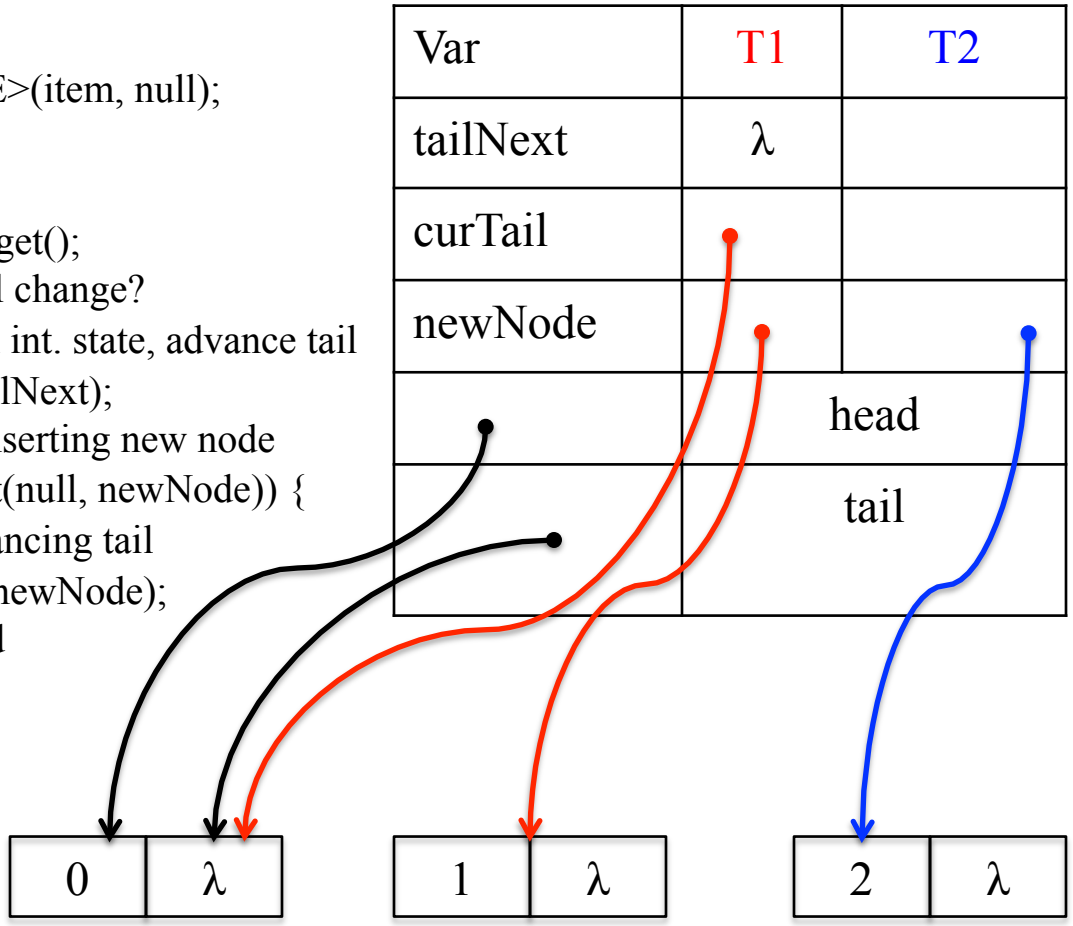
Trace 2

```
    public boolean put(E item) {
1     Node<E> newNode = new Node<E>(item, null);
2     while (true) {
3       Node<E> curTail = tail.get();
4       Node<E> tailNext = curTail.next.get();
5       if (curTail == tail.get()) {// did tail change?
6         if (tailNext != null) { // Queue in int. state, advance tail
7           tail.compareAndSet(curTail, tailNext);
8         else {  // In quiescent state, try inserting new node
9           if (curTail.next.compareAndSet(null, newNode)) {
              // Insertion succeeded, try advancing tail
10              tail.compareAndSet(curTail, newNode);
              // will fail if tail already moved
11            return true;
            }
          }
        }
      }
    }
```



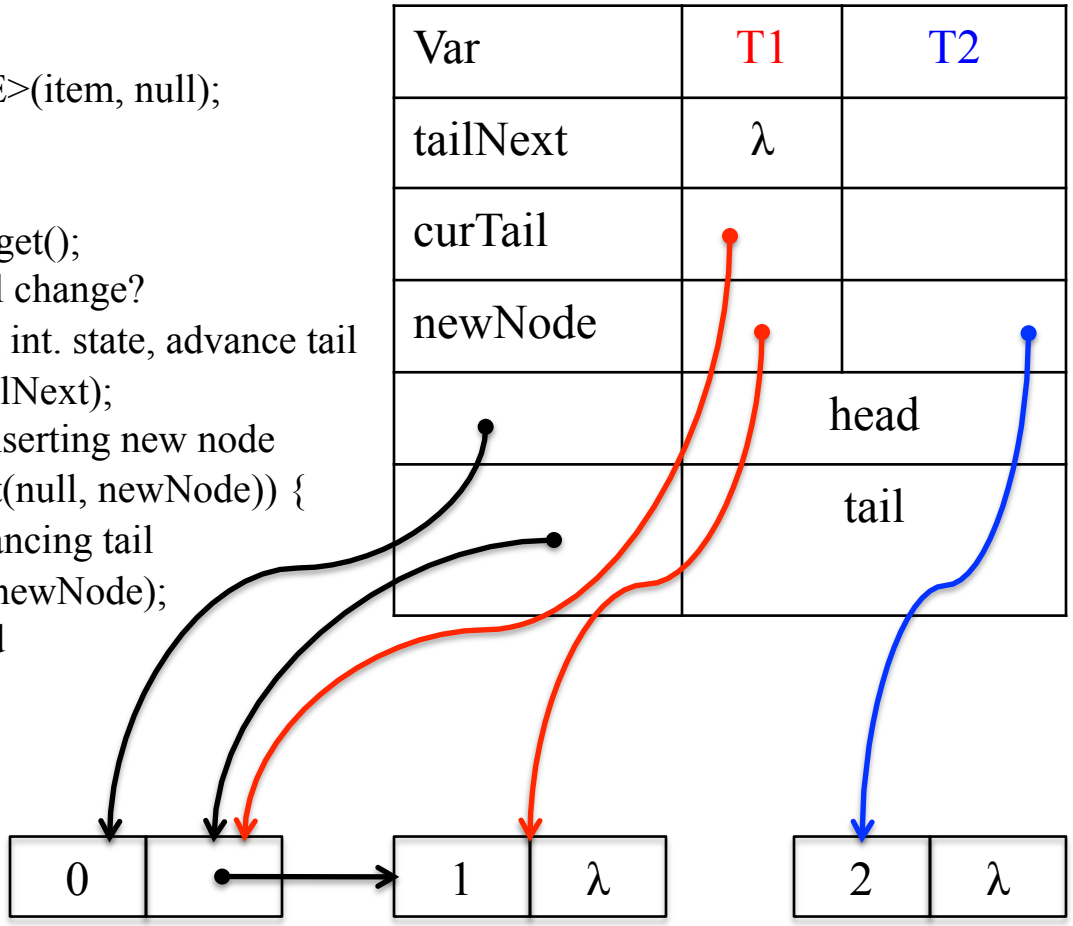
Trace 2

```
    public boolean put(E item) {
1     Node<E> newNode = new Node<E>(item, null);
2     while (true) {
3       Node<E> curTail = tail.get();
4       Node<E> tailNext = curTail.next.get();
5       if (curTail == tail.get()) {// did tail change?
6         if (tailNext != null) { // Queue in int. state, advance tail
7           tail.compareAndSet(curTail, tailNext);
8         else {  // In quiescent state, try inserting new node
9           if (curTail.next.compareAndSet(null, newNode)) {
              // Insertion succeeded, try advancing tail
10              tail.compareAndSet(curTail, newNode);
              // will fail if tail already moved
11            return true;
            }
          }
        }
      }
    }
```



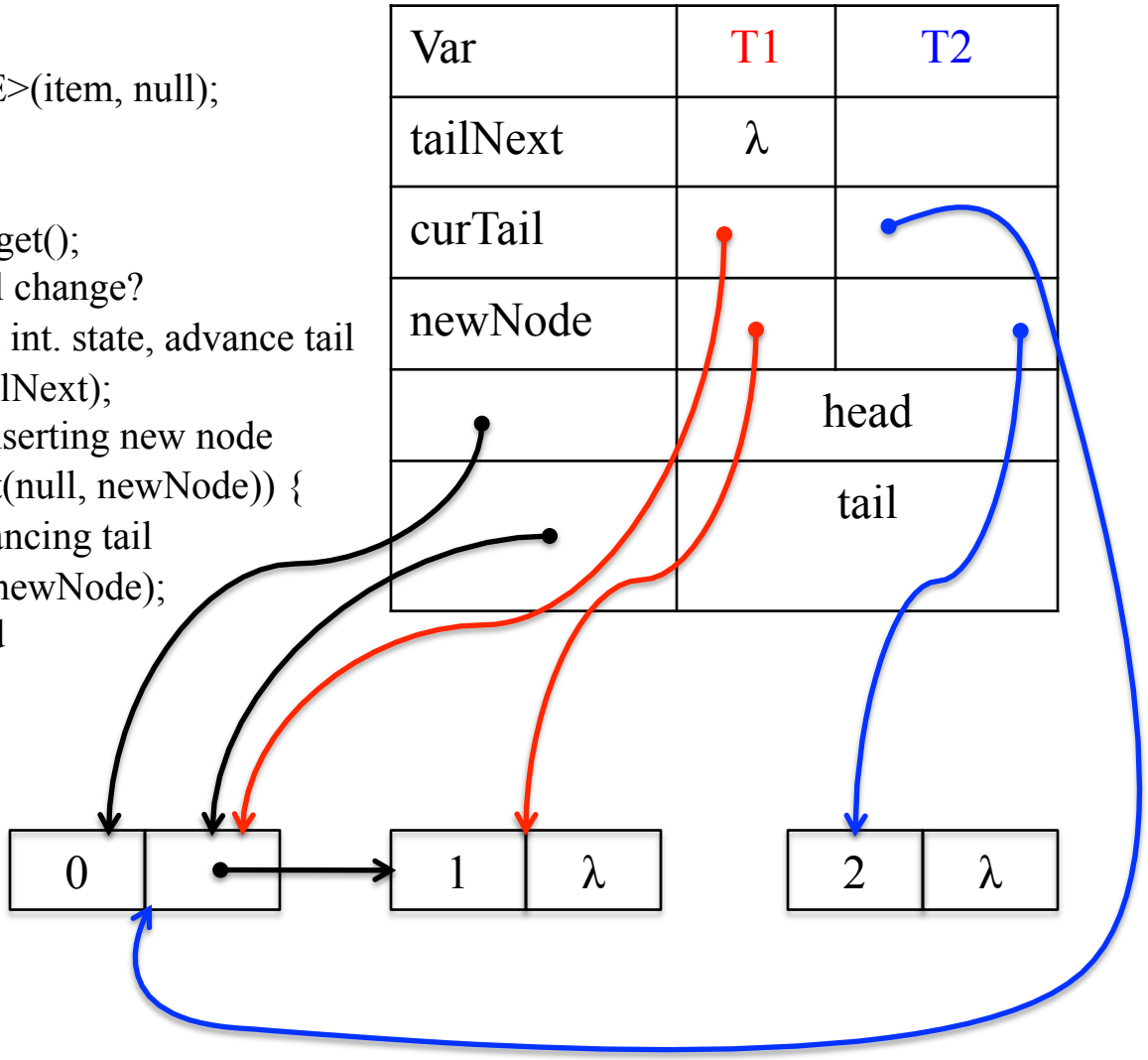
Trace 2: After CAS

```
    public boolean put(E item) {
1     Node<E> newNode = new Node<E>(item, null);
2     while (true) {
3       Node<E> curTail = tail.get();
4       Node<E> tailNext = curTail.next.get();
5       if (curTail == tail.get()) {// did tail change?
6         if (tailNext != null) { // Queue in int. state, advance tail
7           tail.compareAndSet(curTail, tailNext);
8         else {  // In quiescent state, try inserting new node
9           if (curTail.next.compareAndSet(null, newNode)) {
              // Insertion succeeded, try advancing tail
10              tail.compareAndSet(curTail, newNode);
              // will fail if tail already moved
11            return true;
            }
          }
        }
      }
    }
```



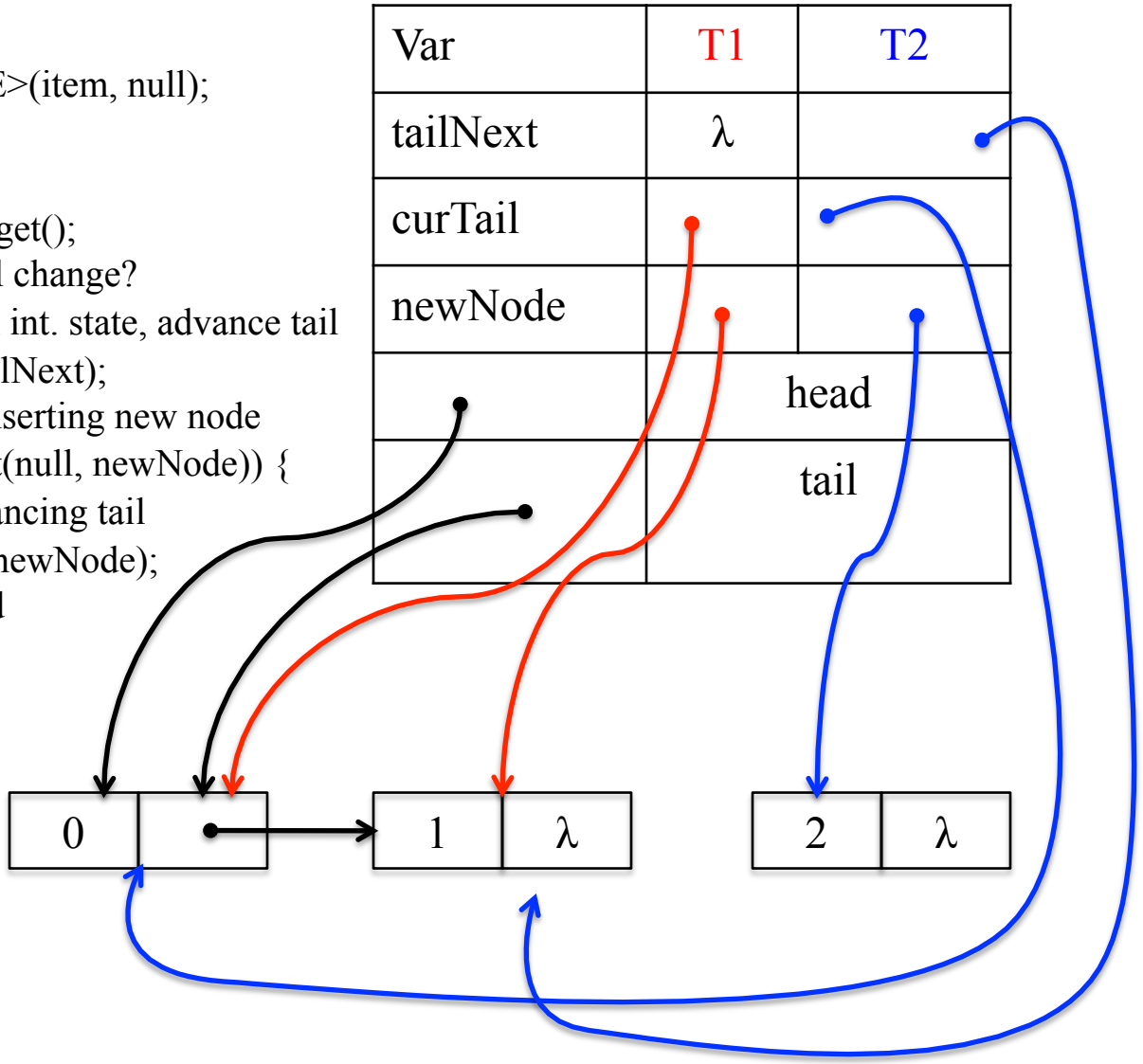
Trace 2: T2 continues

```
    public boolean put(E item) {
1     Node<E> newNode = new Node<E>(item, null);
2     while (true) {
3       Node<E> curTail = tail.get();
4       Node<E> tailNext = curTail.next.get();
5       if (curTail == tail.get()) {// did tail change?
6         if (tailNext != null) { // Queue in int. state, advance tail
7           tail.compareAndSet(curTail, tailNext);
8         else {  // In quiescent state, try inserting new node
9           if (curTail.next.compareAndSet(null, newNode)) {
            // Insertion succeeded, try advancing tail
10            tail.compareAndSet(curTail, newNode);
            // will fail if tail already moved
11          return true;
            }
          }
        }
      }
    }
```



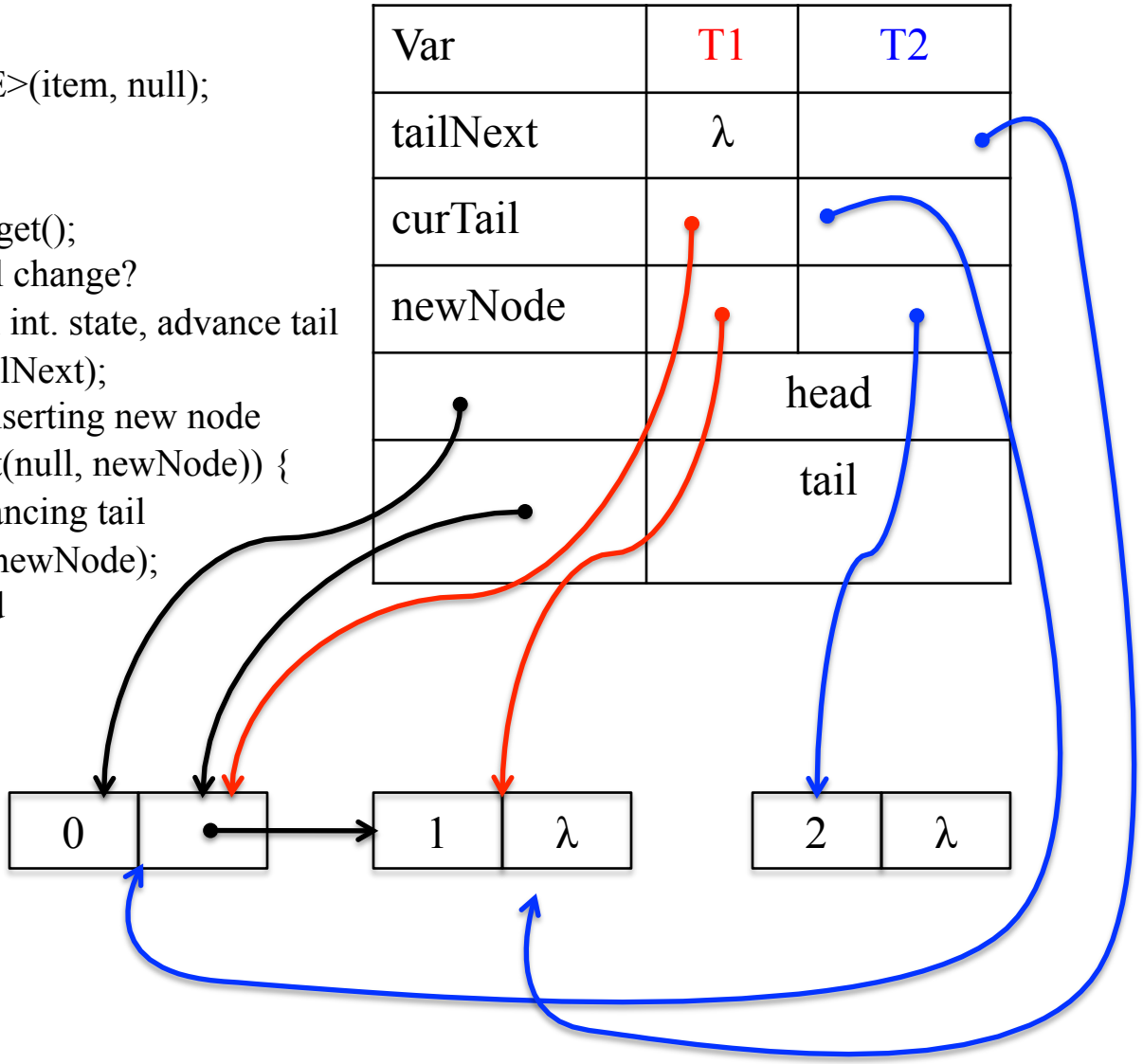
Trace 2

```
    public boolean put(E item) {
1     Node<E> newNode = new Node<E>(item, null);
2     while (true) {
3       Node<E> curTail = tail.get();
4       Node<E> tailNext = curTail.next.get();
5       if (curTail == tail.get()) {// did tail change?
6         if (tailNext != null) { // Queue in int. state, advance tail
7           tail.compareAndSet(curTail, tailNext);
8         else {  // In quiescent state, try inserting new node
9           if (curTail.next.compareAndSet(null, newNode)) {
            // Insertion succeeded, try advancing tail
10            tail.compareAndSet(curTail, newNode);
            // will fail if tail already moved
11          return true;
            }
          }
        }
      }
    }
```



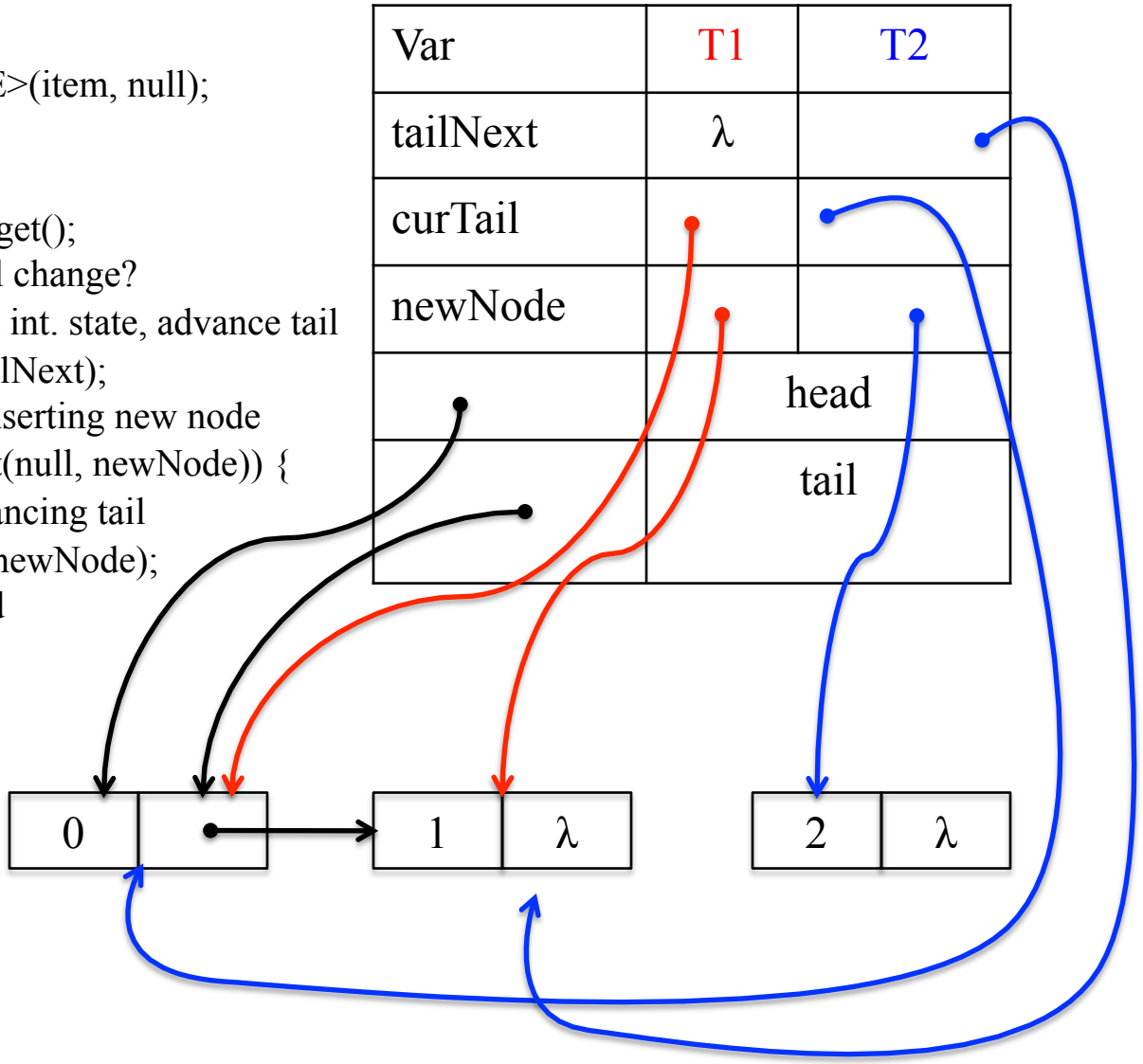
Trace 2

```
    public boolean put(E item) {
1     Node<E> newNode = new Node<E>(item, null);
2     while (true) {
3       Node<E> curTail = tail.get();
4       Node<E> tailNext = curTail.next.get();
5       if (curTail == tail.get()) {// did tail change?
6         if (tailNext != null) { // Queue in int. state, advance tail
7           tail.compareAndSet(curTail, tailNext);
8         else {  // In quiescent state, try inserting new node
9           if (curTail.next.compareAndSet(null, newNode)) {
            // Insertion succeeded, try advancing tail
10            tail.compareAndSet(curTail, newNode);
            // will fail if tail already moved
11          return true;
            }
          }
        }
      }
    }
```



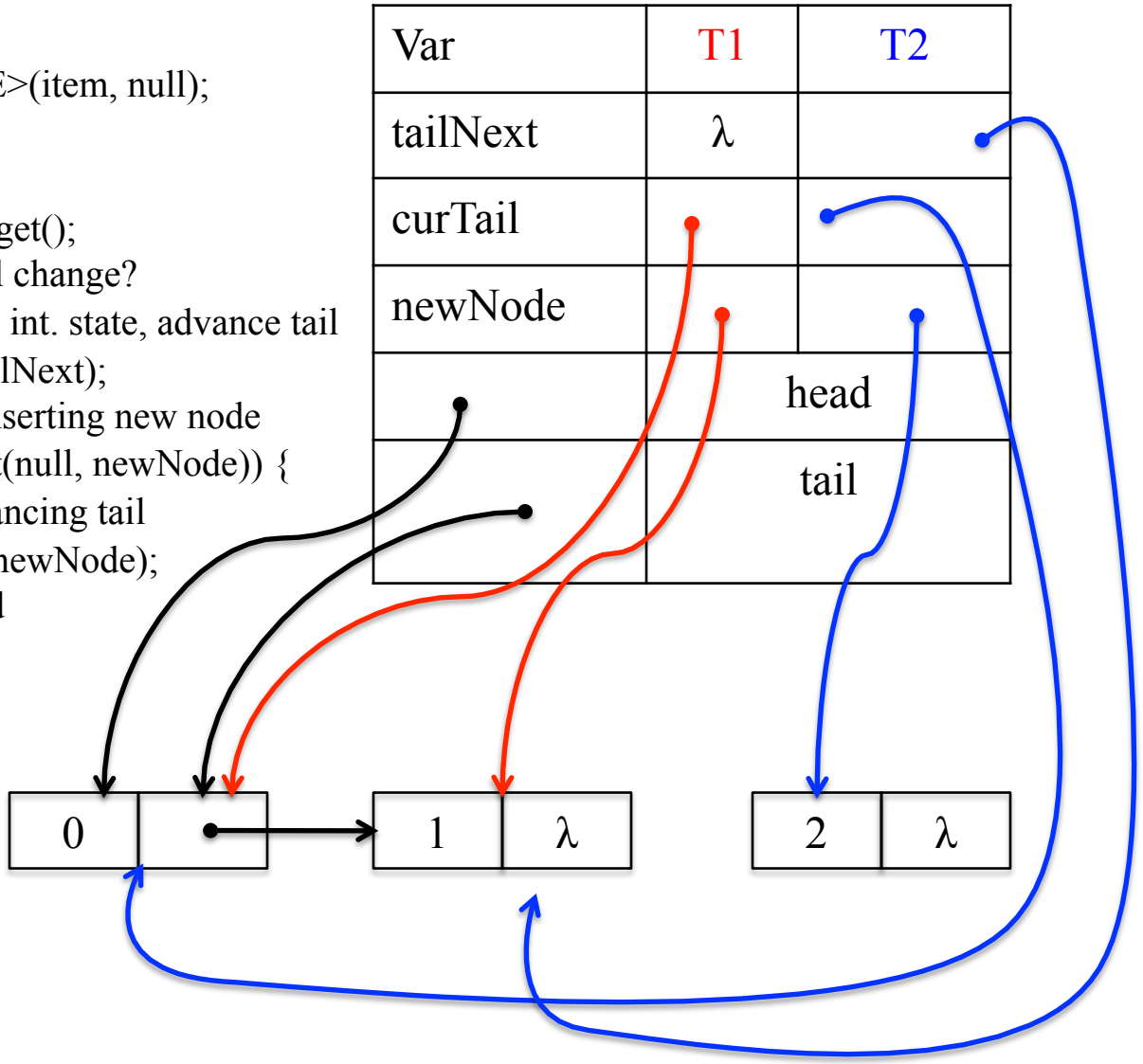
Trace 2

```
    public boolean put(E item) {
1     Node<E> newNode = new Node<E>(item, null);
2     while (true) {
3       Node<E> curTail = tail.get();
4       Node<E> tailNext = curTail.next.get();
5       if (curTail == tail.get()) {// did tail change?
6         if (tailNext != null) { // Queue in int. state, advance tail
7           tail.compareAndSet(curTail, tailNext);
8         else {  // In quiescent state, try inserting new node
9           if (curTail.next.compareAndSet(null, newNode)) {
            // Insertion succeeded, try advancing tail
10            tail.compareAndSet(curTail, newNode);
            // will fail if tail already moved
11          return true;
            }
          }
        }
      }
    }
```



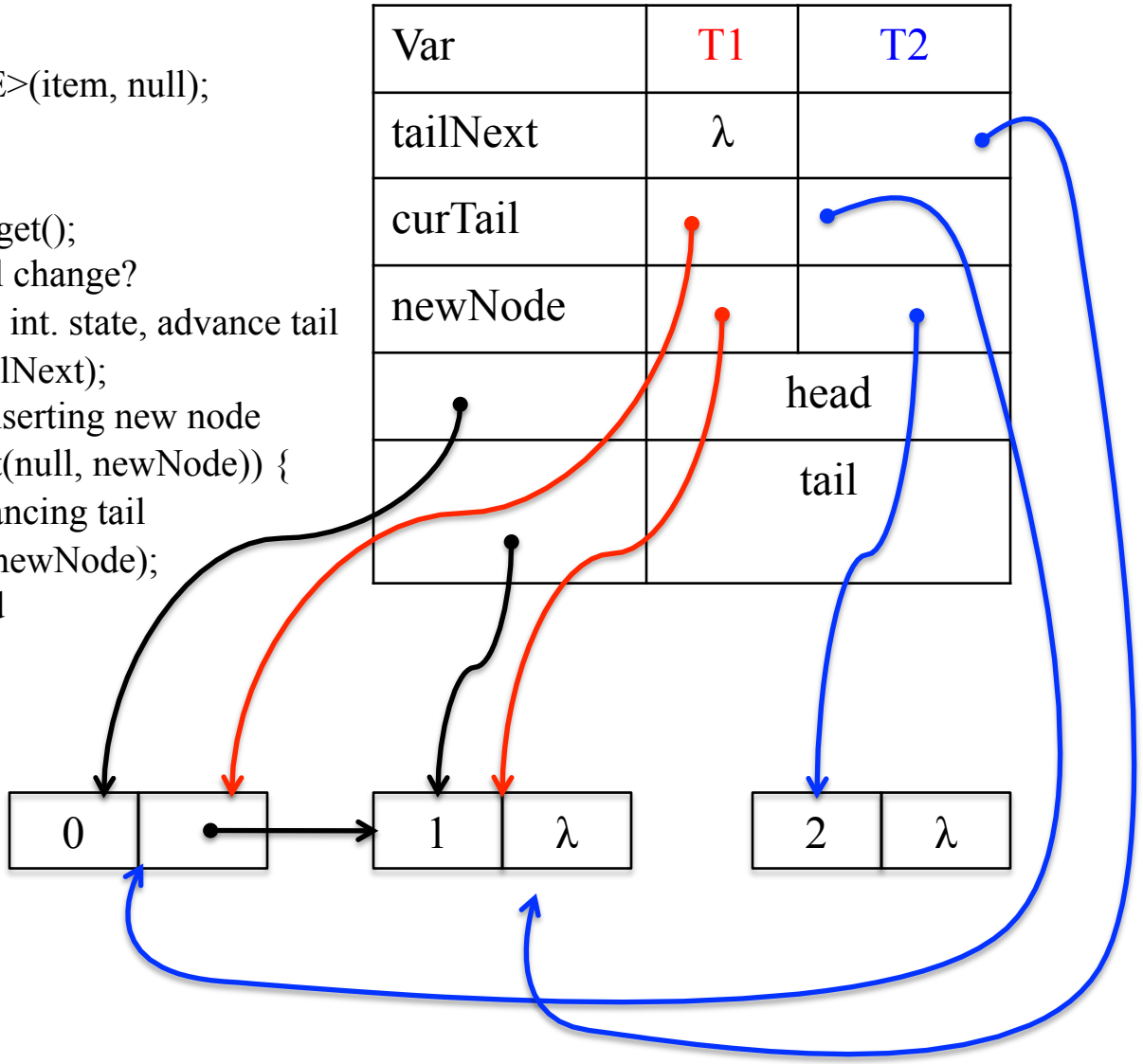
Trace 2:T2 wins

```
    public boolean put(E item) {
1     Node<E> newNode = new Node<E>(item, null);
2     while (true) {
3       Node<E> curTail = tail.get();
4       Node<E> tailNext = curTail.next.get();
5       if (curTail == tail.get()) {// did tail change?
6         if (tailNext != null) { // Queue in int. state, advance tail
7           tail.compareAndSet(curTail, tailNext);
8         else {  // In quiescent state, try inserting new node
9           if (curTail.next.compareAndSet(null, newNode)) {
            // Insertion succeeded, try advancing tail
10            tail.compareAndSet(curTail, newNode);
            // will fail if tail already moved
11          return true;
            }
          }
        }
      }
    }
```



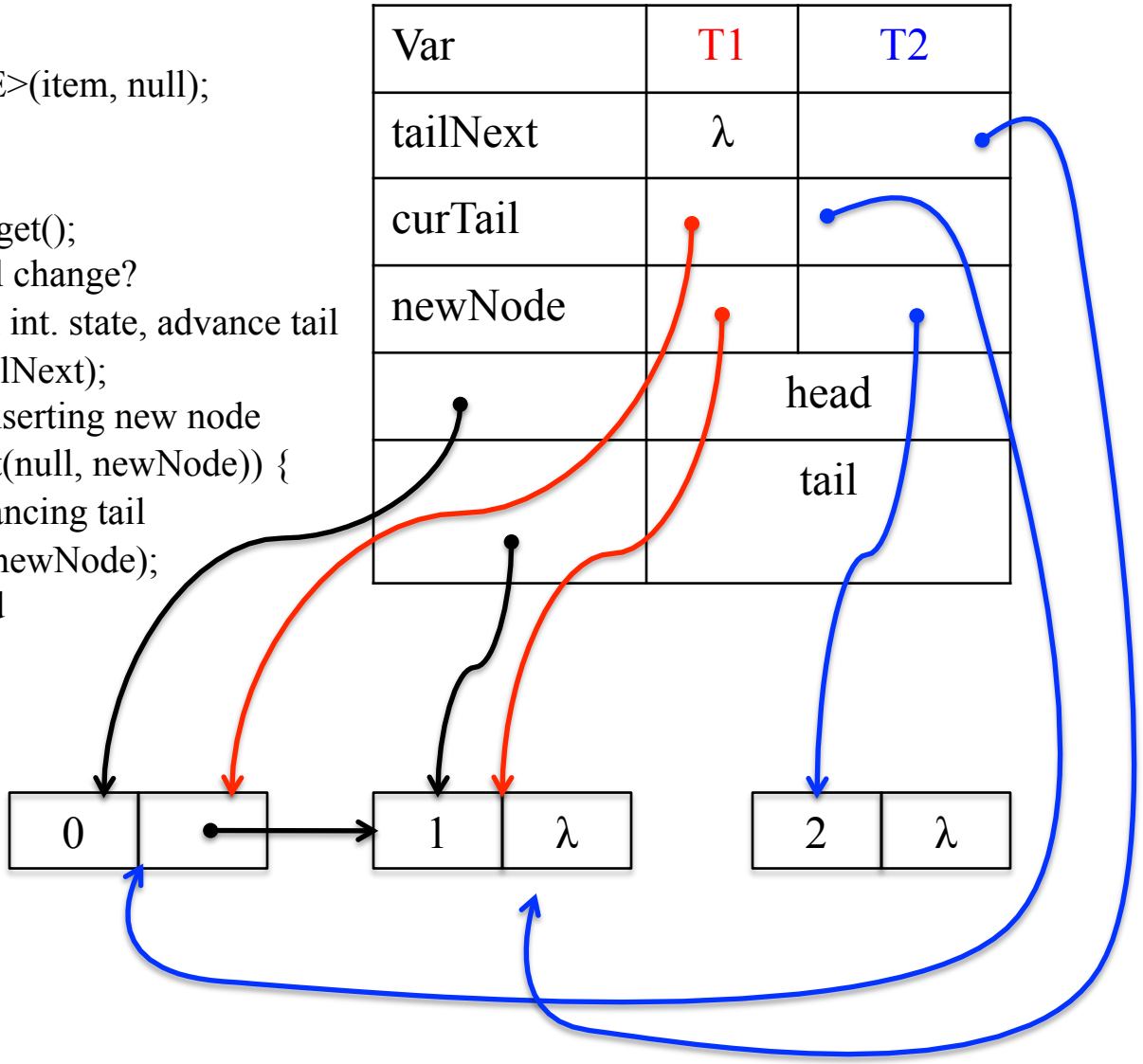
Trace 2: After CAS

```
    public boolean put(E item) {
1     Node<E> newNode = new Node<E>(item, null);
2     while (true) {
3       Node<E> curTail = tail.get();
4       Node<E> tailNext = curTail.next.get();
5       if (curTail == tail.get()) {// did tail change?
6         if (tailNext != null) { // Queue in int. state, advance tail
7           tail.compareAndSet(curTail, tailNext);
8         else {  // In quiescent state, try inserting new node
9           if (curTail.next.compareAndSet(null, newNode)) {
            // Insertion succeeded, try advancing tail
10            tail.compareAndSet(curTail, newNode);
            // will fail if tail already moved
11          return true;
            }
          }
        }
      }
    }
```



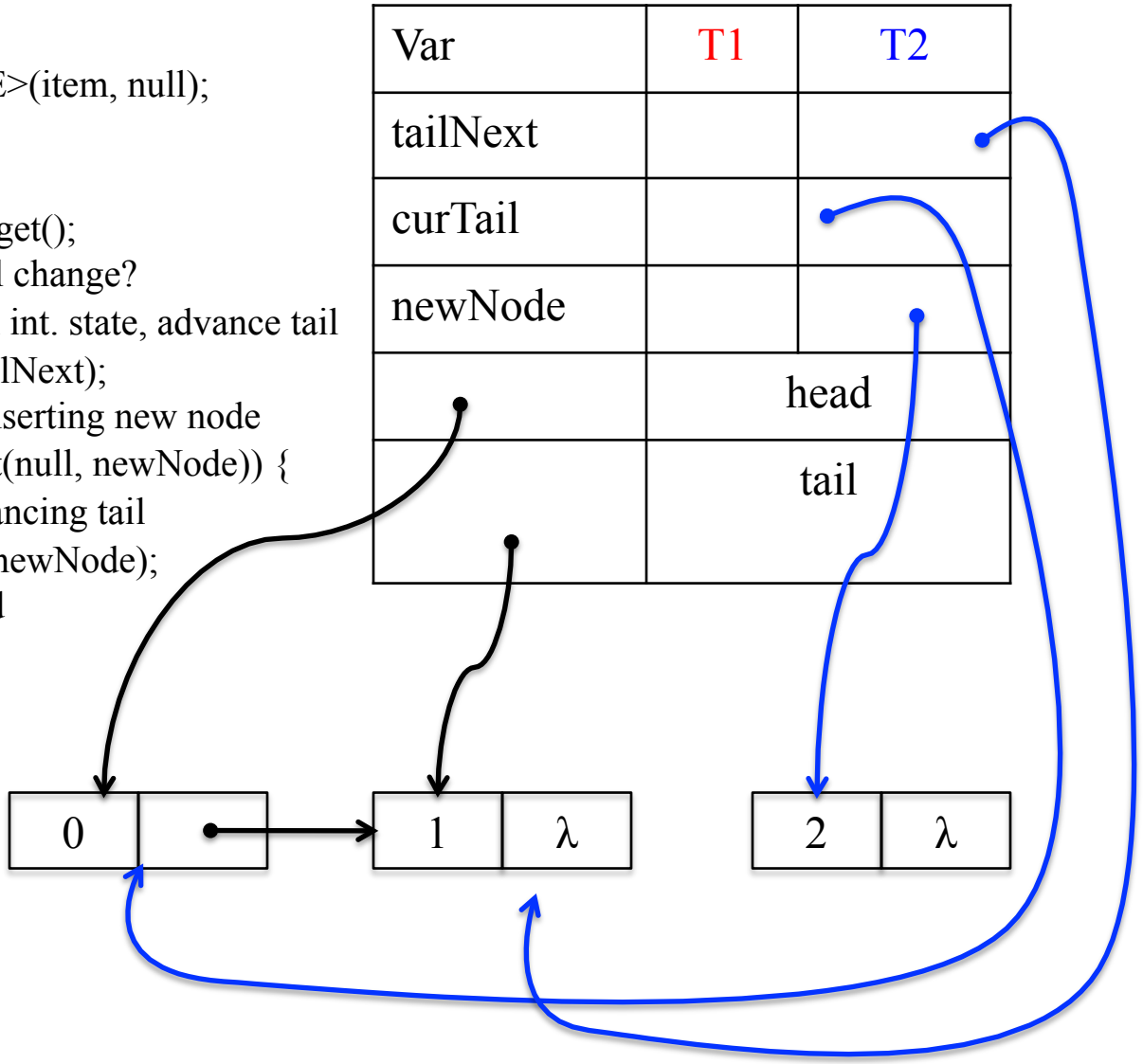
Trace 2: T1 continues: CAS fails

```
public boolean put(E item) {
1     Node<E> newNode = new Node<E>(item, null);
2     while (true) {
3       Node<E> curTail = tail.get();
4       Node<E> tailNext = curTail.next.get();
5       if (curTail == tail.get()) {// did tail change?
6         if (tailNext != null) { // Queue in int. state, advance tail
7           tail.compareAndSet(curTail, tailNext);
8         else {  // In quiescent state, try inserting new node
9           if (curTail.next.compareAndSet(null, newNode)) {
            // Insertion succeeded, try advancing tail
10            tail.compareAndSet(curTail, newNode);
            // will fail if tail already moved
11          return true;
            }
          }
        }
      }
    }
```



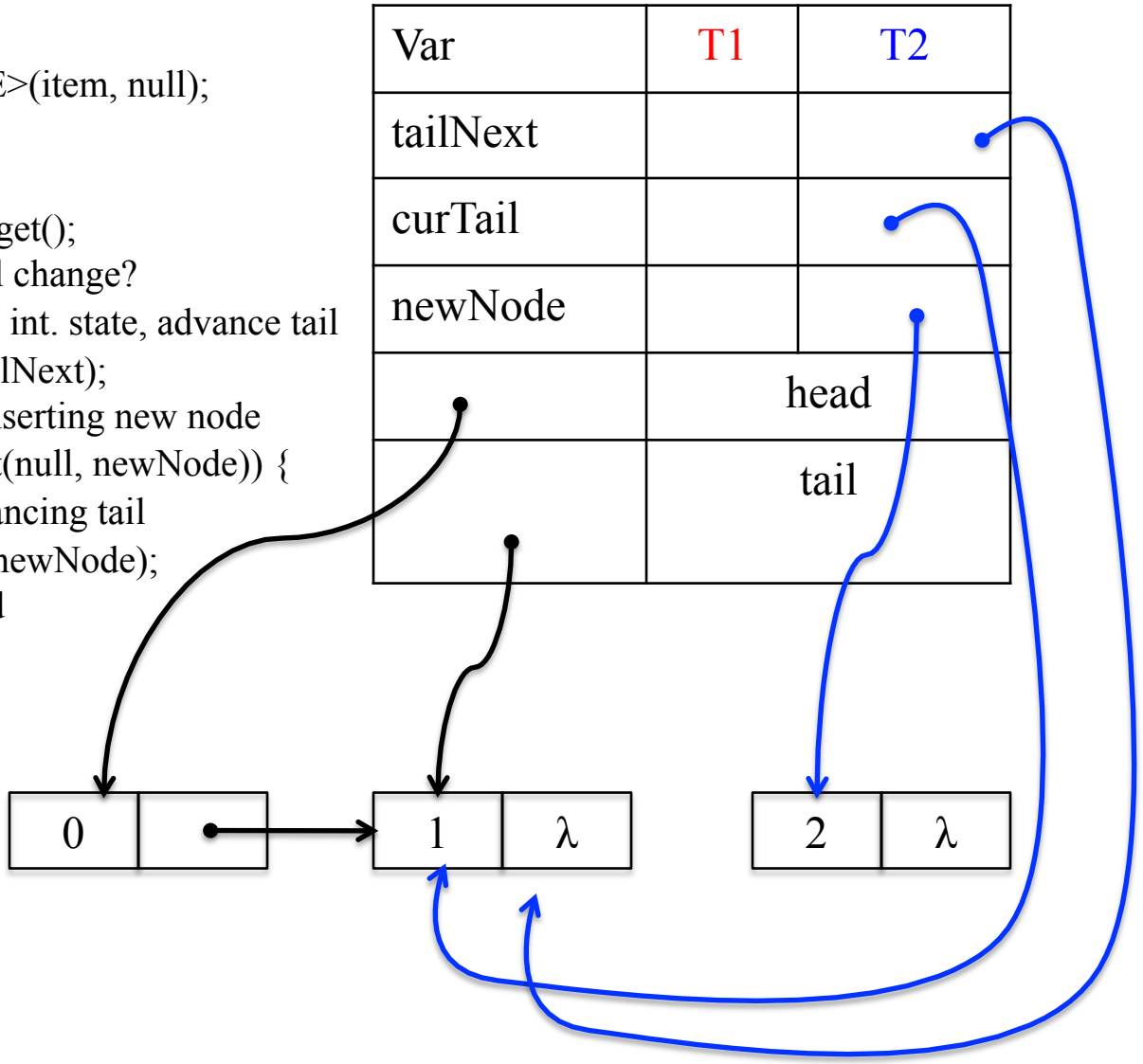
Trace 2: T1 exits

```
    public boolean put(E item) {
1     Node<E> newNode = new Node<E>(item, null);
2     while (true) {
3       Node<E> curTail = tail.get();
4       Node<E> tailNext = curTail.next.get();
5       if (curTail == tail.get()) {// did tail change?
6         if (tailNext != null) { // Queue in int. state, advance tail
7           tail.compareAndSet(curTail, tailNext);
8         else {  // In quiescent state, try inserting new node
9           if (curTail.next.compareAndSet(null, newNode)) {
              // Insertion succeeded, try advancing tail
10              tail.compareAndSet(curTail, newNode);
              // will fail if tail already moved
11            return true;
            }
          }
        }
      }
    }
```



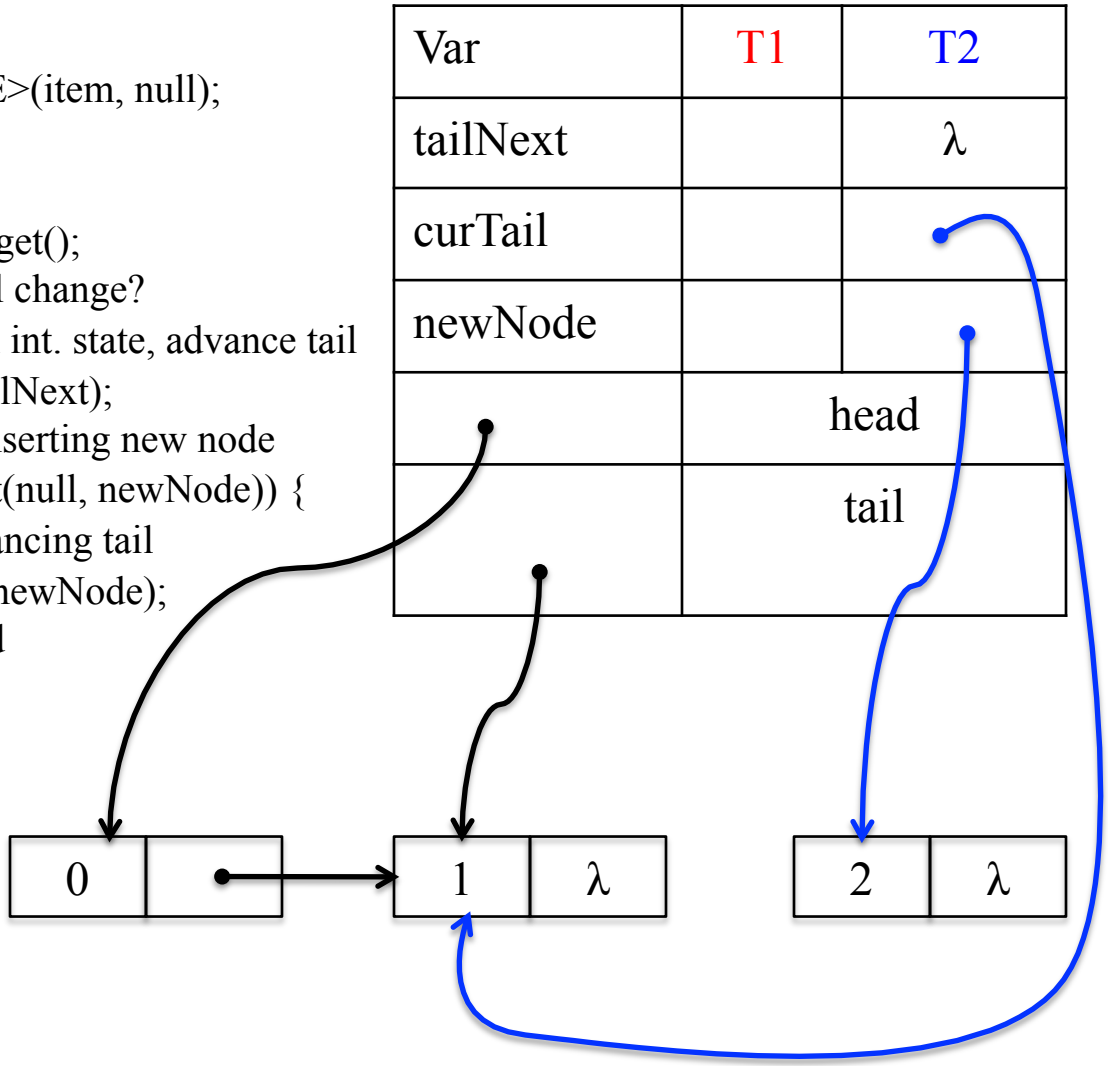
Trace 2: T2 continues

```
    public boolean put(E item) {
1     Node<E> newNode = new Node<E>(item, null);
2     while (true) {
3       Node<E> curTail = tail.get();
4       Node<E> tailNext = curTail.next.get();
5       if (curTail == tail.get()) {// did tail change?
6         if (tailNext != null) { // Queue in int. state, advance tail
7           tail.compareAndSet(curTail, tailNext);
8         else {  // In quiescent state, try inserting new node
9           if (curTail.next.compareAndSet(null, newNode)) {
              // Insertion succeeded, try advancing tail
10              tail.compareAndSet(curTail, newNode);
              // will fail if tail already moved
11            return true;
            }
          }
        }
      }
    }
```



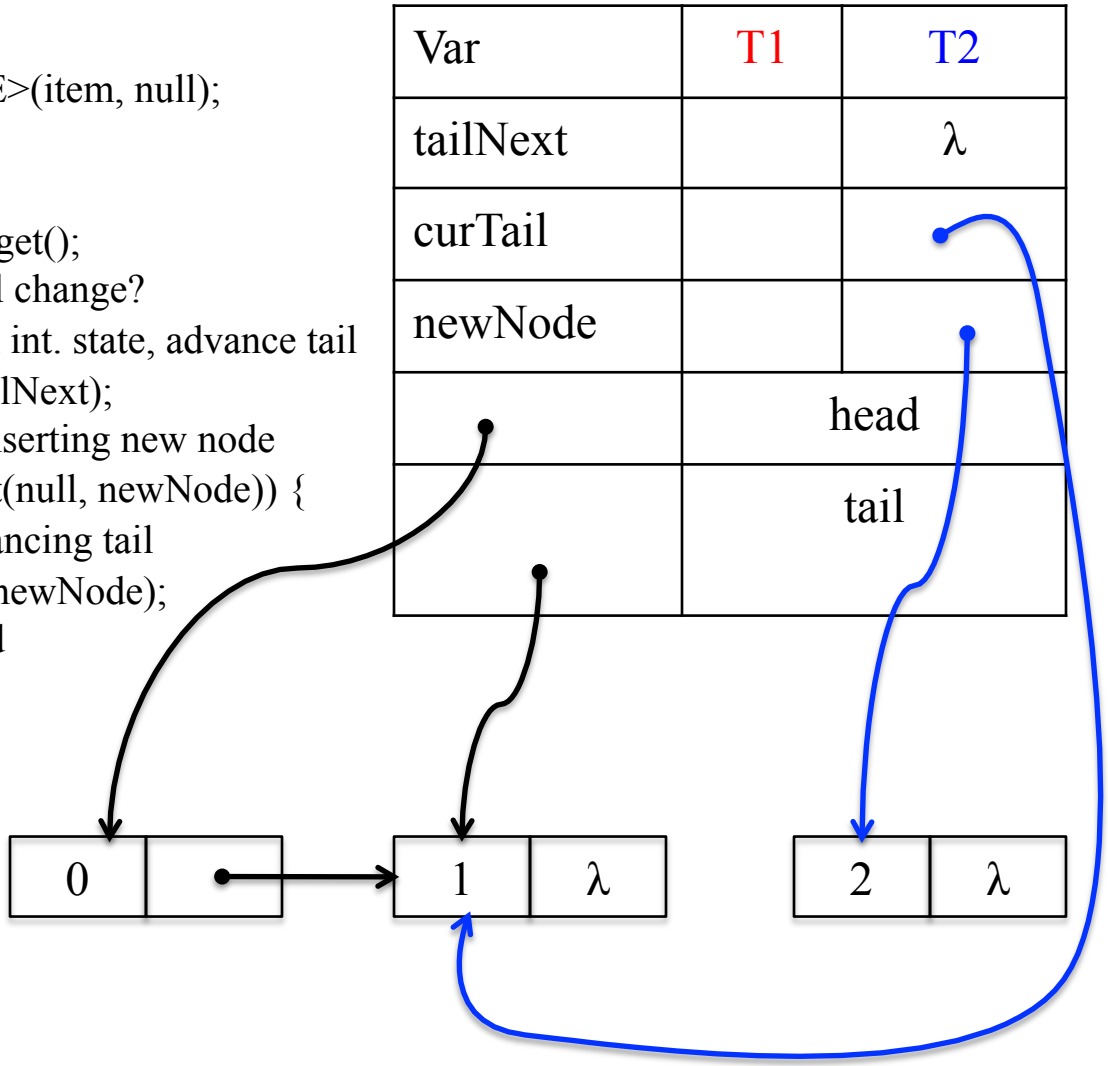
Trace 2: T2 continues

```
    public boolean put(E item) {
1     Node<E> newNode = new Node<E>(item, null);
2     while (true) {
3       Node<E> curTail = tail.get();
4       Node<E> tailNext = curTail.next.get();
5       if (curTail == tail.get()) {// did tail change?
6         if (tailNext != null) { // Queue in int. state, advance tail
7           tail.compareAndSet(curTail, tailNext);
8         else {  // In quiescent state, try inserting new node
9           if (curTail.next.compareAndSet(null, newNode)) {
            // Insertion succeeded, try advancing tail
10            tail.compareAndSet(curTail, newNode);
            // will fail if tail already moved
11          return true;
            }
          }
        }
      }
    }
```



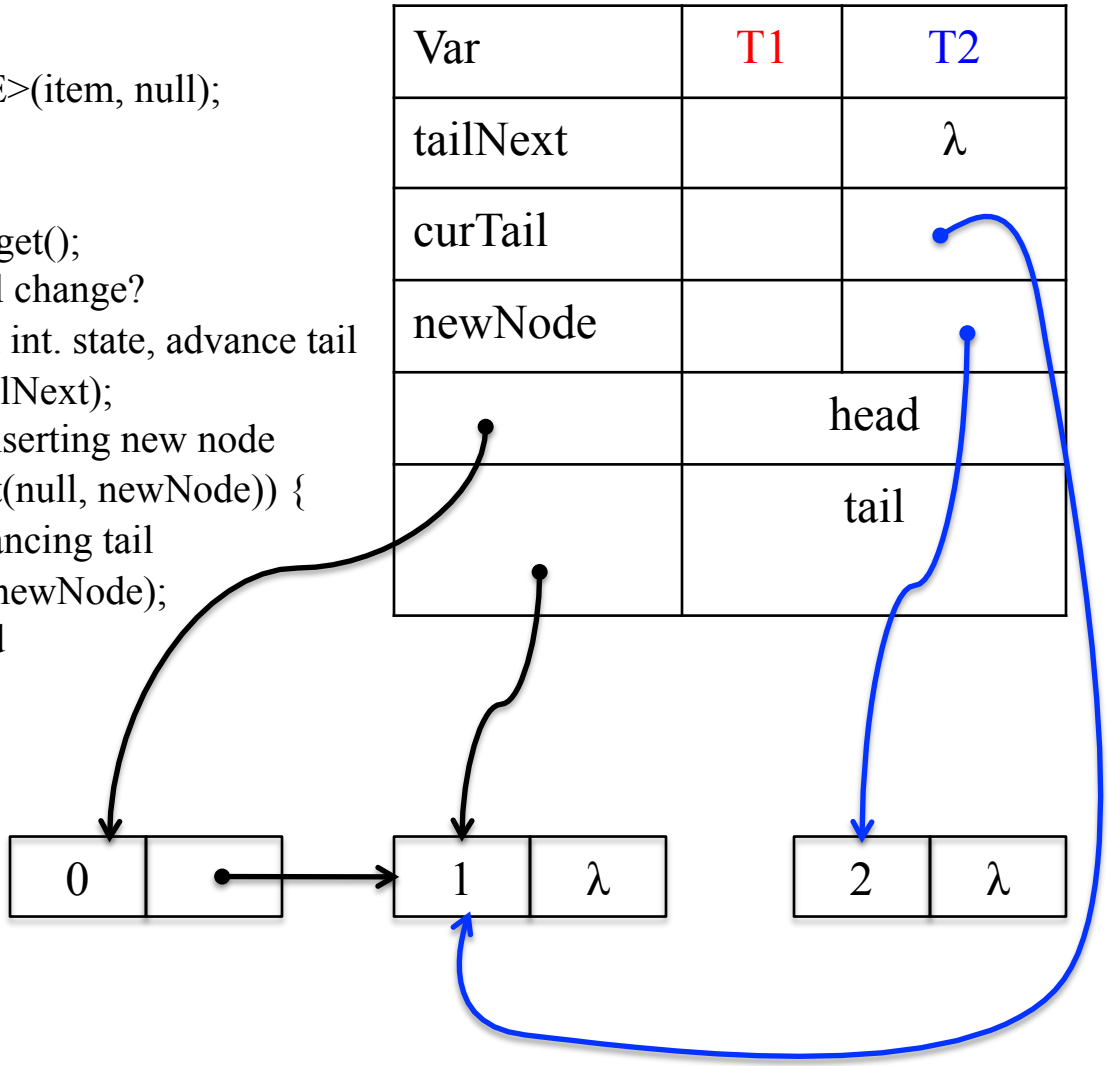
Trace 2: T2 continues

```
    public boolean put(E item) {
1     Node<E> newNode = new Node<E>(item, null);
2     while (true) {
3       Node<E> curTail = tail.get();
4       Node<E> tailNext = curTail.next.get();
5       if (curTail == tail.get()) {// did tail change?
6         if (tailNext != null) { // Queue in int. state, advance tail
7           tail.compareAndSet(curTail, tailNext);
8         else {  // In quiescent state, try inserting new node
9           if (curTail.next.compareAndSet(null, newNode)) {
            // Insertion succeeded, try advancing tail
10            tail.compareAndSet(curTail, newNode);
            // will fail if tail already moved
11          return true;
            }
          }
        }
      }
    }
```



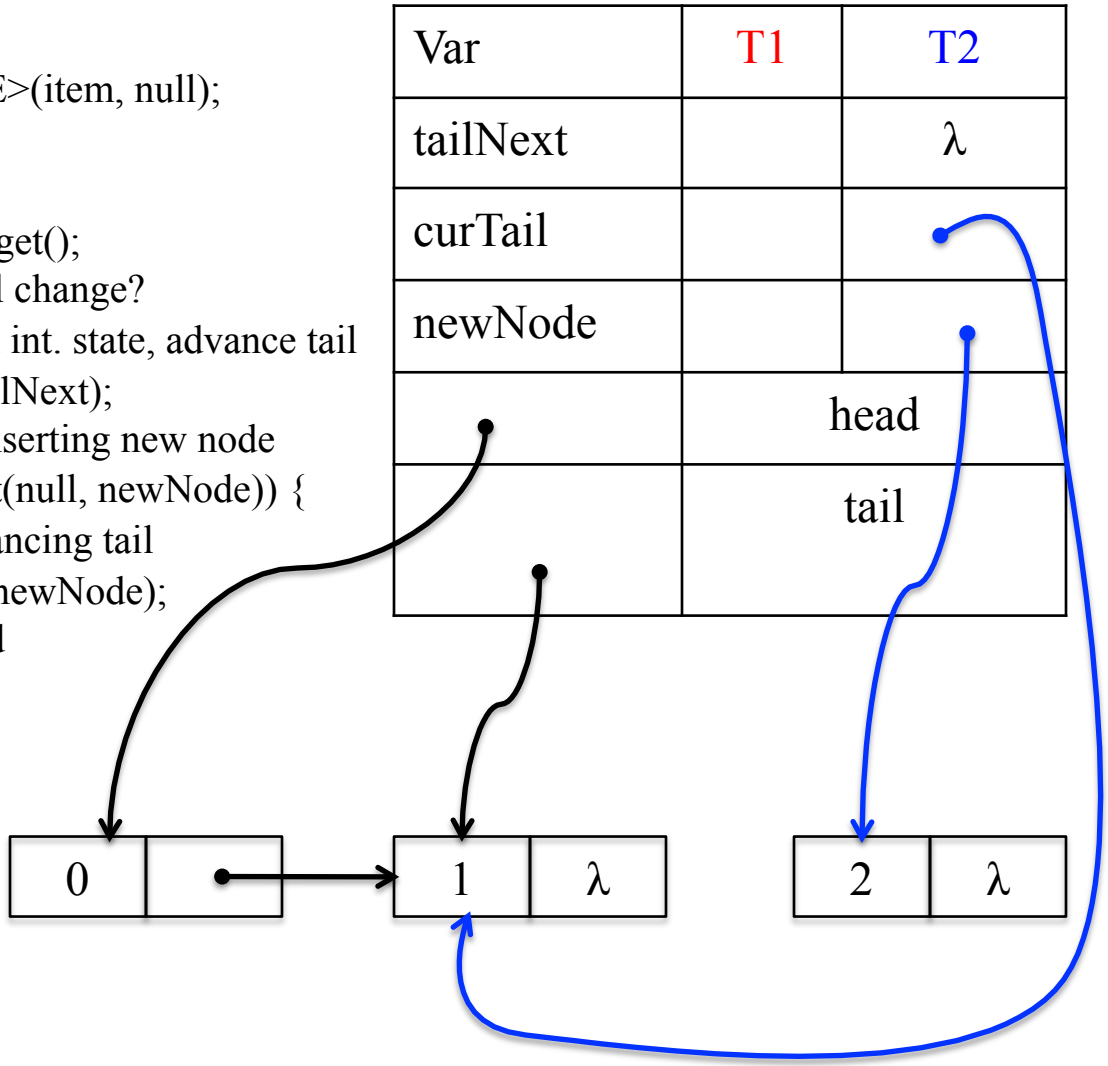
Trace 2: T2 continues

```
    public boolean put(E item) {
1     Node<E> newNode = new Node<E>(item, null);
2     while (true) {
3       Node<E> curTail = tail.get();
4       Node<E> tailNext = curTail.next.get();
5       if (curTail == tail.get()) {// did tail change?
6         if (tailNext != null) { // Queue in int. state, advance tail
7           tail.compareAndSet(curTail, tailNext);
8         else {  // In quiescent state, try inserting new node
9           if (curTail.next.compareAndSet(null, newNode)) {
            // Insertion succeeded, try advancing tail
10            tail.compareAndSet(curTail, newNode);
            // will fail if tail already moved
11          return true;
            }
          }
        }
      }
    }
```



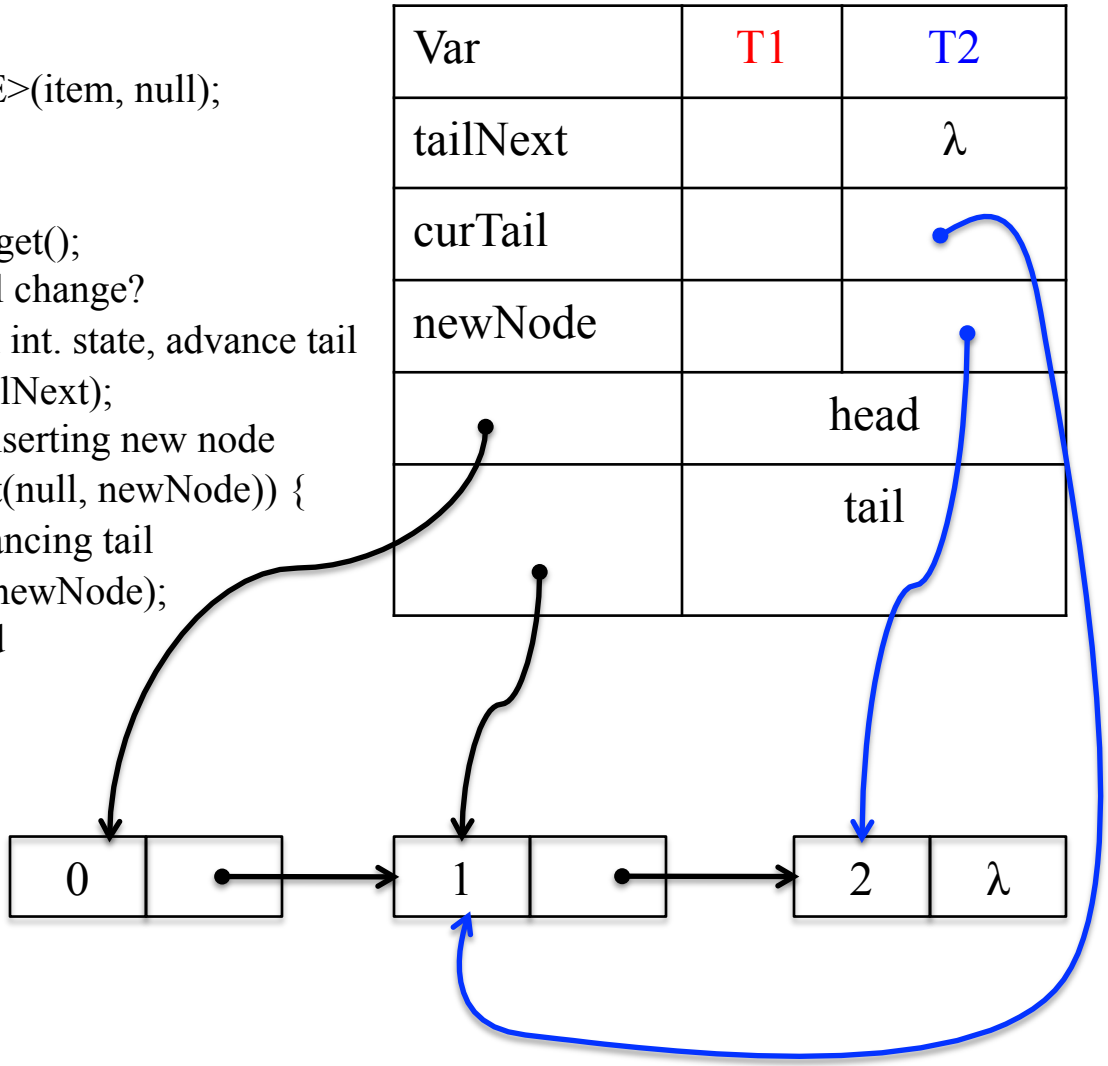
Trace 2: T2 continues

```
    public boolean put(E item) {
1     Node<E> newNode = new Node<E>(item, null);
2     while (true) {
3       Node<E> curTail = tail.get();
4       Node<E> tailNext = curTail.next.get();
5       if (curTail == tail.get()) {// did tail change?
6         if (tailNext != null) { // Queue in int. state, advance tail
7           tail.compareAndSet(curTail, tailNext);
8         else {  // In quiescent state, try inserting new node
9           if (curTail.next.compareAndSet(null, newNode)) {
            // Insertion succeeded, try advancing tail
10            tail.compareAndSet(curTail, newNode);
            // will fail if tail already moved
11          return true;
            }
          }
        }
      }
    }
```



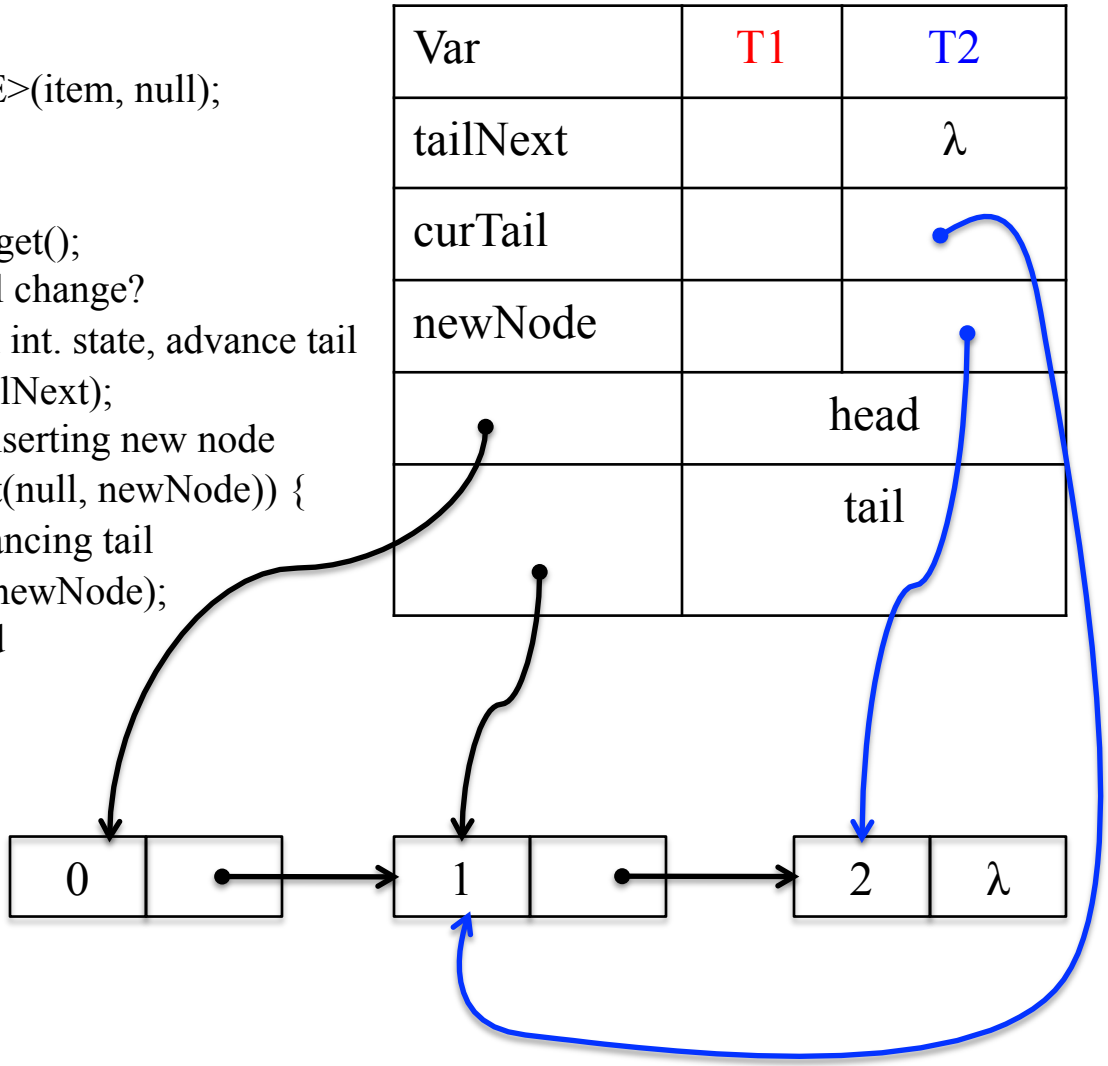
Trace 2: After CAS

```
    public boolean put(E item) {
1     Node<E> newNode = new Node<E>(item, null);
2     while (true) {
3       Node<E> curTail = tail.get();
4       Node<E> tailNext = curTail.next.get();
5       if (curTail == tail.get()) {// did tail change?
6         if (tailNext != null) { // Queue in int. state, advance tail
7           tail.compareAndSet(curTail, tailNext);
8         else {  // In quiescent state, try inserting new node
9           if (curTail.next.compareAndSet(null, newNode)) {
            // Insertion succeeded, try advancing tail
10            tail.compareAndSet(curTail, newNode);
            // will fail if tail already moved
11          return true;
            }
          }
        }
      }
    }
```



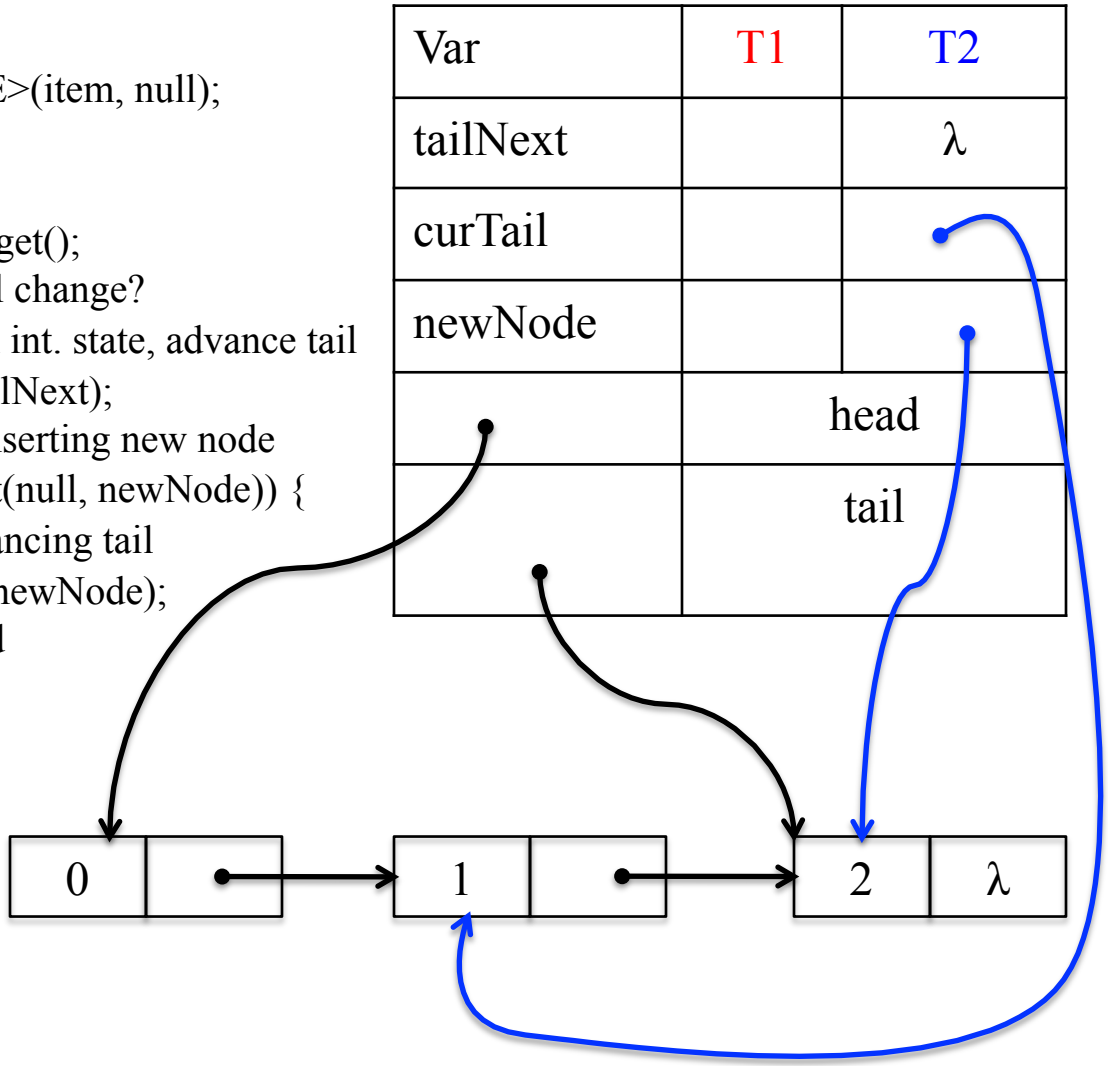
Trace 2: T2 continues

```
    public boolean put(E item) {
1     Node<E> newNode = new Node<E>(item, null);
2     while (true) {
3       Node<E> curTail = tail.get();
4       Node<E> tailNext = curTail.next.get();
5       if (curTail == tail.get()) {// did tail change?
6         if (tailNext != null) { // Queue in int. state, advance tail
7           tail.compareAndSet(curTail, tailNext);
8         else {  // In quiescent state, try inserting new node
9           if (curTail.next.compareAndSet(null, newNode)) {
            // Insertion succeeded, try advancing tail
10            tail.compareAndSet(curTail, newNode);
            // will fail if tail already moved
11          return true;
            }
          }
        }
      }
    }
```



Trace 2: After CAS

```
    public boolean put(E item) {
1     Node<E> newNode = new Node<E>(item, null);
2     while (true) {
3       Node<E> curTail = tail.get();
4       Node<E> tailNext = curTail.next.get();
5       if (curTail == tail.get()) {// did tail change?
6         if (tailNext != null) { // Queue in int. state, advance tail
7           tail.compareAndSet(curTail, tailNext);
8         else {  // In quiescent state, try inserting new node
9           if (curTail.next.compareAndSet(null, newNode)) {
            // Insertion succeeded, try advancing tail
10            tail.compareAndSet(curTail, newNode);
            // will fail if tail already moved
11          return true;
            }
          }
        }
      }
    }
```



Trace 2: T2 Exits

```
    public boolean put(E item) {
1     Node<E> newNode = new Node<E>(item, null);
2     while (true) {
3       Node<E> curTail = tail.get();
4       Node<E> tailNext = curTail.next.get();
5       if (curTail == tail.get()) {// did tail change?
6         if (tailNext != null) { // Queue in int. state, advance tail
7           tail.compareAndSet(curTail, tailNext);
8         else {  // In quiescent state, try inserting new node
9           if (curTail.next.compareAndSet(null, newNode)) {
            // Insertion succeeded, try advancing tail
10            tail.compareAndSet(curTail, newNode);
            // will fail if tail already moved
11          return true;
            }
          }
        }
      }
    }
```

Var	T1	T2
tailNext		
curTail		
newNode		
	head	
	tail