

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)) {
10                 // Insertion succeeded, try advancing tail
11                 tail.compareAndSet(curTail, newNode);
12                 // will fail if tail already moved
13                 return true;
14             }
15         }
16     }
17 }

```

Var	T1	T2
tailNext		
curTail		
newNode		
	head	
	tail	



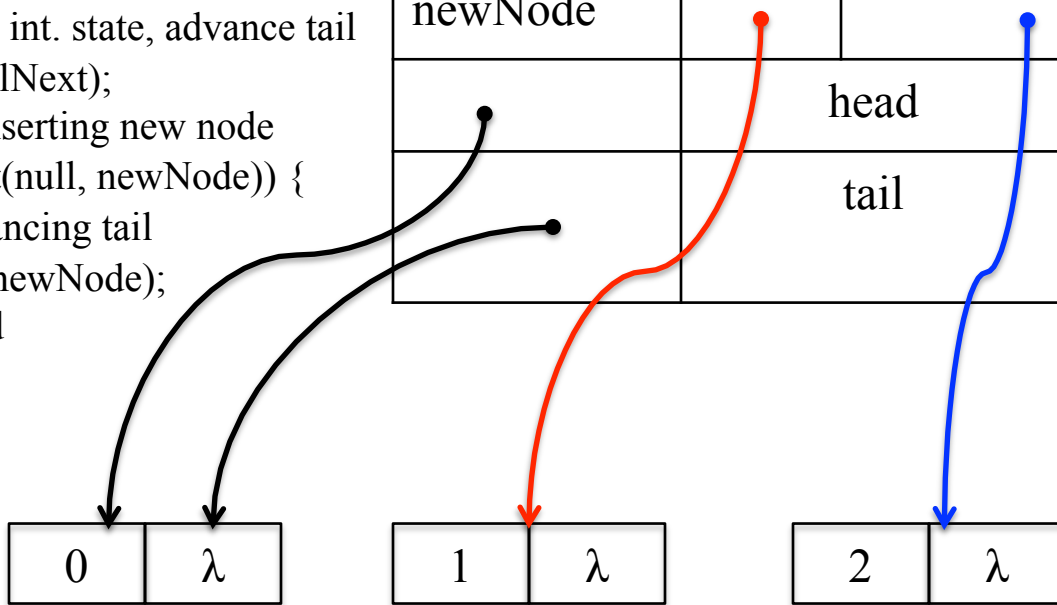
Trace 1

```

1 → public boolean put(E item) {
2   Node<E> newNode = new Node<E>(item, null);
3   while (true) {
4     Node<E> curTail = tail.get();
5     Node<E> tailNext = curTail.next.get();
6     if (curTail == tail.get()) { // did tail change?
7       if (tailNext != null) { // Queue in int. state, advance tail
8         tail.compareAndSet(curTail, tailNext);
9       } else { // In quiescent state, try inserting new node
10        if (curTail.next.compareAndSet(null, newNode)) {
11          // Insertion succeeded, try advancing tail
12          tail.compareAndSet(curTail, newNode);
13          // will fail if tail already moved
14          return true;
15        }
16      }
17    }
18  }
19 }

```

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)) {
10 // Insertion succeeded, try advancing tail
10 tail.compareAndSet(curTail, newNode);
10 // 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)) {
10                  // Insertion succeeded, try advancing tail
11                  tail.compareAndSet(curTail, newNode);
12                  // will fail if tail already moved
13                  return true;
14              }
15          }
16      }
17  }
18  }
19  }

```

Var	T1	T2
tailNext	λ	λ
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)) {
10                 // Insertion succeeded, try advancing tail
11                 tail.compareAndSet(curTail, newNode);
12                 // will fail if tail already moved
13                 return true;
14             }
15         }
16     }
17 }
18 }

```

Var	T1	T2
tailNext	λ	λ
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)) {
10             // Insertion succeeded, try advancing tail
11             tail.compareAndSet(curTail, newNode);
12             // will fail if tail already moved
13             return true;
14         }
15     }
16 }
17 }
18 }

```

Var	T1	T2
tailNext	λ	λ
curTail		
newNode		
head		
tail		



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          }
9          else { // In quiescent state, try inserting new node
10             if (curTail.next.compareAndSet(null, newNode)) {
11                 // Insertion succeeded, try advancing tail
12                 tail.compareAndSet(curTail, newNode);
13                 // will fail if tail already moved
14                 return true;
15             }
16         }
17     }
18 }

```

Var	T1	T2
tailNext	λ	λ
curTail		
newNode		
head		
tail		



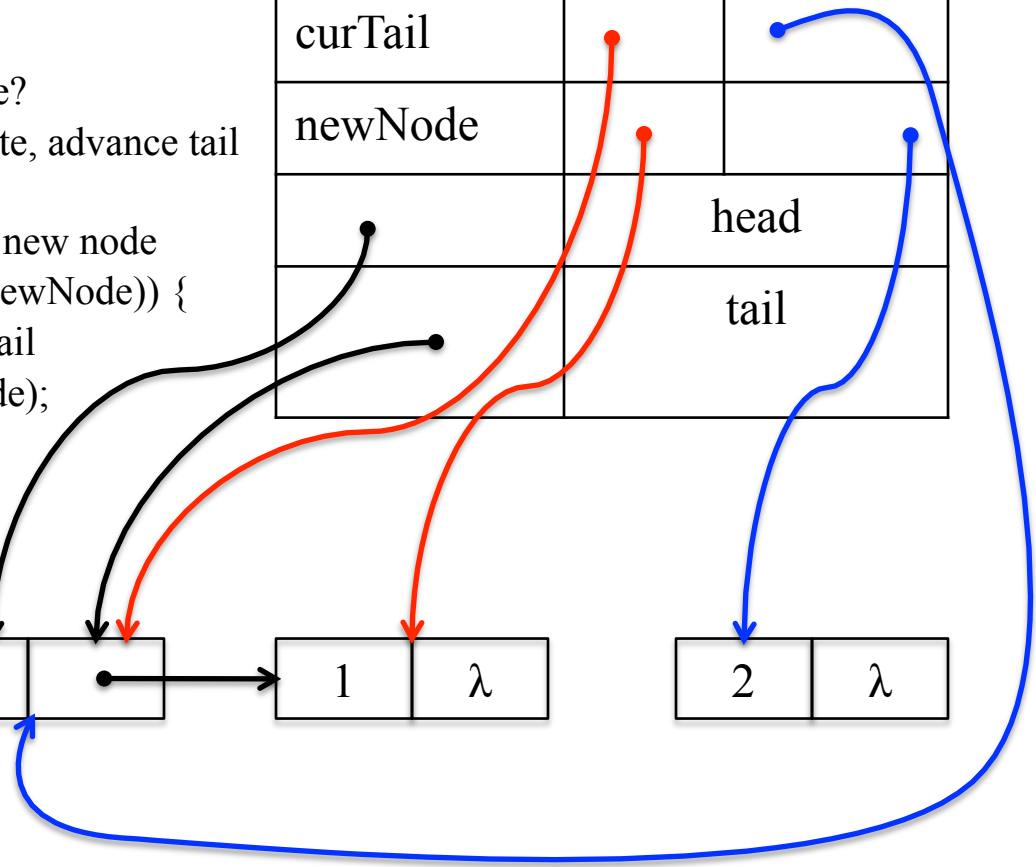
Trace 1: After CAS

```

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          }
9          else { // In quiescent state, try inserting new node
10             if (curTail.next.compareAndSet(null, newNode)) {
11                 // Insertion succeeded, try advancing tail
12                 tail.compareAndSet(curTail, newNode);
13                 // will fail if tail already moved
14                 return true;
15             }
16         }
17     }
18 }

```

Var	T1	T2
tailNext	λ	λ
curTail		
newNode		
head		
tail		



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          }
9          else { // In quiescent state, try inserting new node
10             if (curTail.next.compareAndSet(null, newNode)) {
11                 // Insertion succeeded, try advancing tail
12                 tail.compareAndSet(curTail, newNode);
13                 // will fail if tail already moved
14                 return true;
15             }
16         }
17     }
18 }

```

Var	T1	T2
tailNext	λ	λ
curTail		
newNode		
head		
tail		



Trace 1: T1 Continues

```

1  public boolean put(E item) {
2      Node<E> newNode = new Node<E>(item, null);
3      while (true) {
4          Node<E> curTail = tail.get();
5          Node<E> tailNext = curTail.next.get();
6          if (curTail == tail.get()) { // did tail change?
7              if (tailNext != null) { // Queue in int. state, advance tail
8                  tail.compareAndSet(curTail, tailNext);
9              } else { // In quiescent state, try inserting new node
10                 if (curTail.next.compareAndSet(null, newNode)) {
11                     // Insertion succeeded, try advancing tail
12                     tail.compareAndSet(curTail, newNode);
13                     // will fail if tail already moved
14                     return true;
15                 }
16             }
17         }
18     }
19 }

```

Var	T1	T2
tailNext	λ	λ
curTail		
newNode		
head		
tail		



Trace 1: After CAS

```

1  public boolean put(E item) {
2      Node<E> newNode = new Node<E>(item, null);
3      while (true) {
4          Node<E> curTail = tail.get();
5          Node<E> tailNext = curTail.next.get();
6          if (curTail == tail.get()) { // did tail change?
7              if (tailNext != null) { // Queue in int. state, advance tail
8                  tail.compareAndSet(curTail, tailNext);
9              }
10             if (curTail.next.compareAndSet(null, newNode)) {
11                 // Insertion succeeded, try advancing tail
12                 tail.compareAndSet(curTail, newNode);
13                 // will fail if tail already moved
14                 return true;
15             }
16         }
17     }
18 }

```

Var	T1	T2
tailNext	λ	λ
curTail		
newNode		
head		
tail		



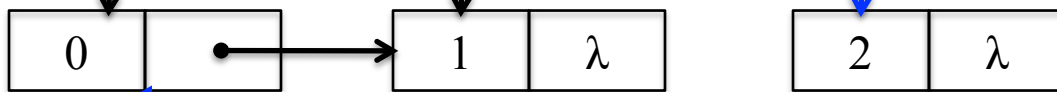
Trace 1: T1 exits

```

1  public boolean put(E item) {
2      Node<E> newNode = new Node<E>(item, null);
3      while (true) {
4          Node<E> curTail = tail.get();
5          Node<E> tailNext = curTail.next.get();
6          if (curTail == tail.get()) { // did tail change?
7              if (tailNext != null) { // Queue in int. state, advance tail
8                  tail.compareAndSet(curTail, tailNext);
9              }
10             if (curTail.next.compareAndSet(null, newNode)) {
11                 // Insertion succeeded, try advancing tail
12                 tail.compareAndSet(curTail, newNode);
13                 // will fail if tail already moved
14                 return true;
15             }
16         }
17     }
18 }

```

Var	T1	T2
tailNext		λ
curTail		
newNode		
		head
		tail



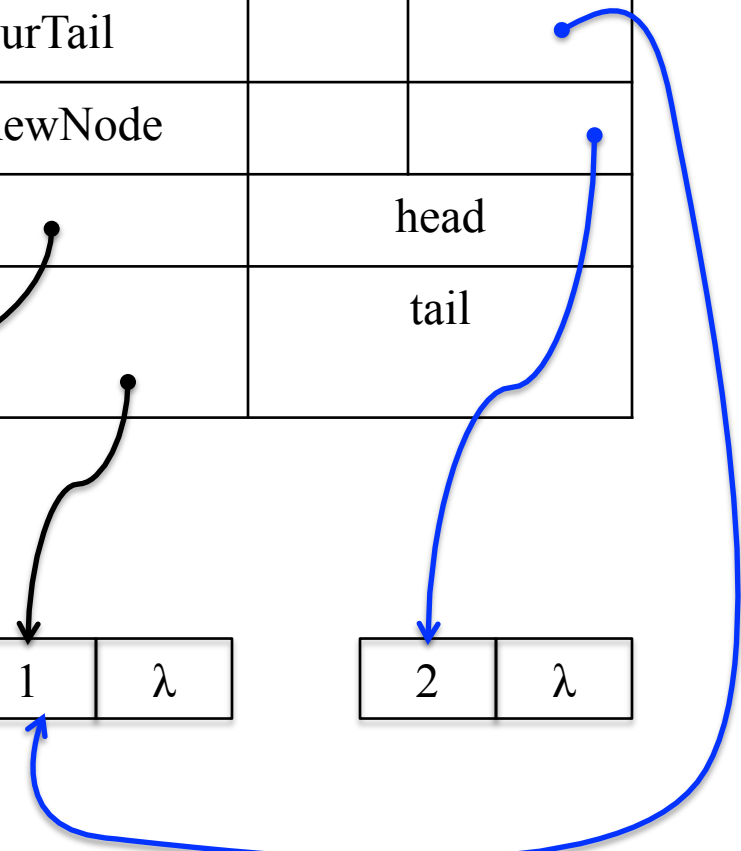
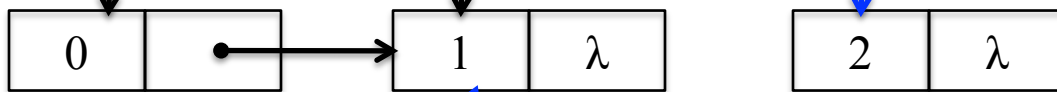
Trace 1: T2 Continues

```

1  public boolean put(E item) {
2      Node<E> newNode = new Node<E>(item, null);
3      while (true) {
4          Node<E> curTail = tail.get();
5          Node<E> tailNext = curTail.next.get();
6          if (curTail == tail.get()) { // did tail change?
7              if (tailNext != null) { // Queue in int. state, advance tail
8                  tail.compareAndSet(curTail, tailNext);
9              }
10             else { // In quiescent state, try inserting new node
11                 if (curTail.next.compareAndSet(null, newNode)) {
12                     // Insertion succeeded, try advancing tail
13                     tail.compareAndSet(curTail, newNode);
14                     // will fail if tail already moved
15                     return true;
16                 }
17             }
18         }
19     }
20 }

```

Var	T1	T2
tailNext		λ
curTail		
newNode		
	head	
	tail	



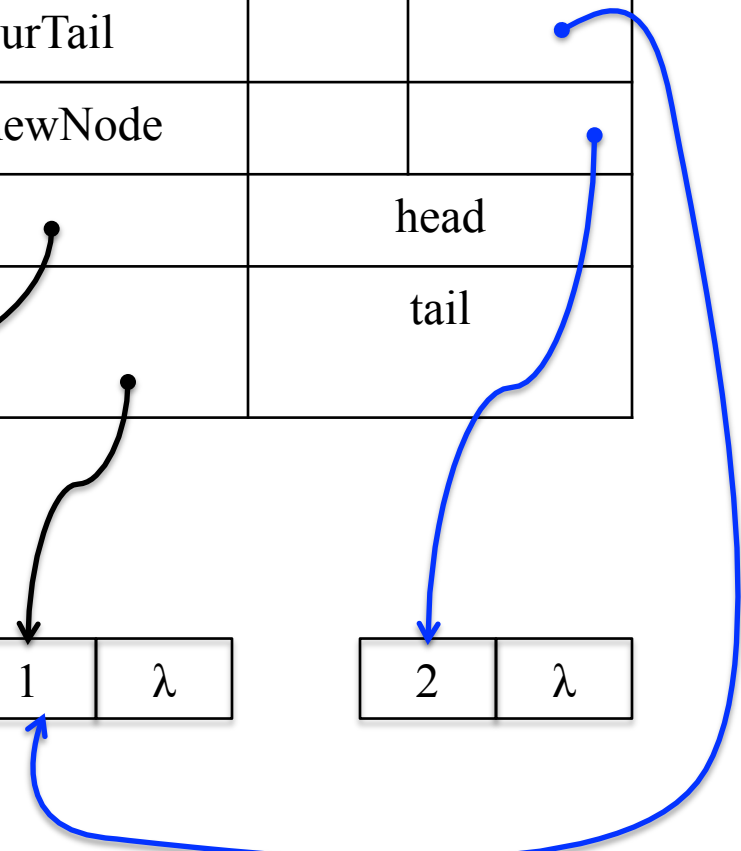
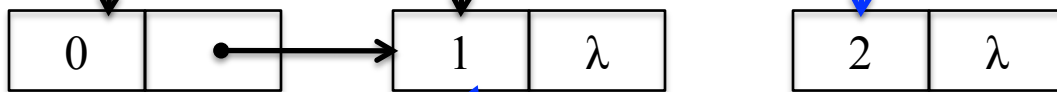
Trace 1: T2 Continues

```

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)) {
10         // Insertion succeeded, try advancing tail
11         tail.compareAndSet(curTail, newNode);
12         // will fail if tail already moved
13         return true;
14       }
15     }
16   }
17 }

```

Var	T1	T2
tailNext		λ
curTail		
newNode		
	head	
	tail	

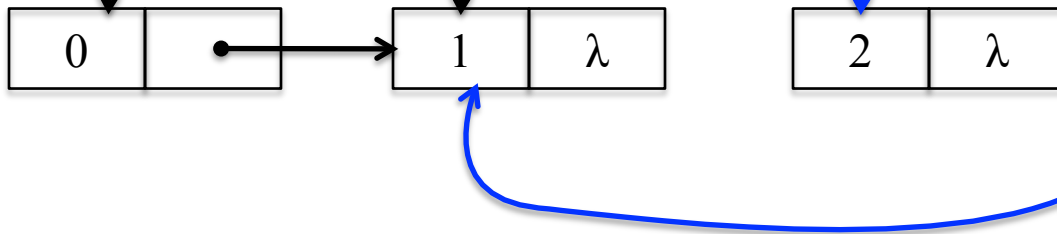


```

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)) {
10                  // Insertion succeeded, try advancing tail
11                  tail.compareAndSet(curTail, newNode);
12                  // will fail if tail already moved
13                  return true;
14              }
15          }
16      }
17  }
18 }

```

Var	T1	T2
tailNext		λ
curTail		
newNode		
	head	
	tail	



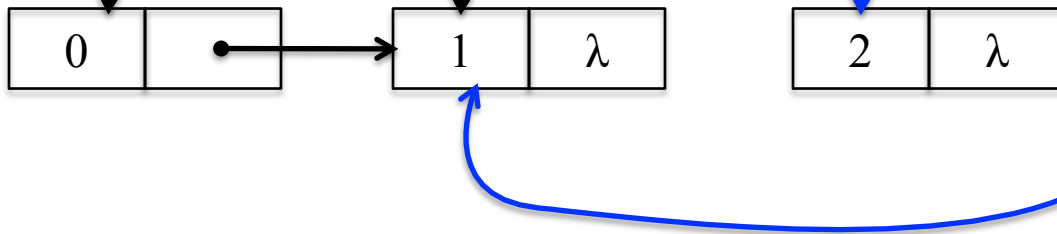
```

1  public boolean put(E item) {
2      Node<E> newNode = new Node<E>(item, null);
3      while (true) {
4          Node<E> curTail = tail.get();
5          Node<E> tailNext = curTail.next.get();
6          if (curTail == tail.get()) { // did tail change?
7              if (tailNext != null) { // Queue in int. state, advance tail
8                  tail.compareAndSet(curTail, tailNext);
9              }
10             else { // In quiescent state, try inserting new node
11                 if (curTail.next.compareAndSet(null, newNode)) {
12                     // Insertion succeeded, try advancing tail
13                     tail.compareAndSet(curTail, newNode);
14                     // will fail if tail already moved
15                 }
16             }
17             return true;
18         }
19     }
20 }

```



Var	T1	T2
tailNext		λ
curTail		
newNode		
	head	
	tail	



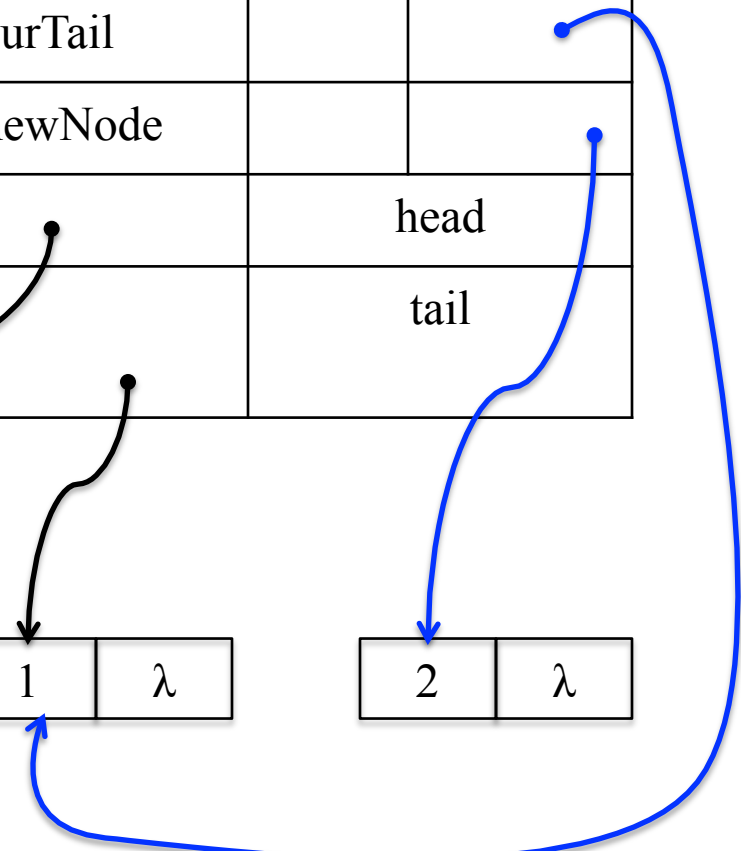
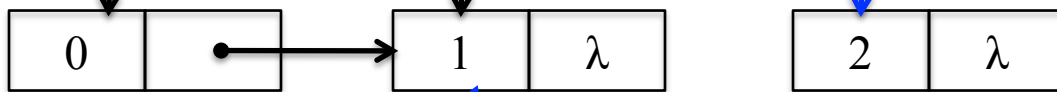
Trace 1: T2 Continues

```

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)) {
10                 // Insertion succeeded, try advancing tail
11                 tail.compareAndSet(curTail, newNode);
12                 // will fail if tail already moved
13                 return true;
14             }
15         }
16     }
17 }

```

Var	T1	T2
tailNext		λ
curTail		
newNode		
	head	
	tail	

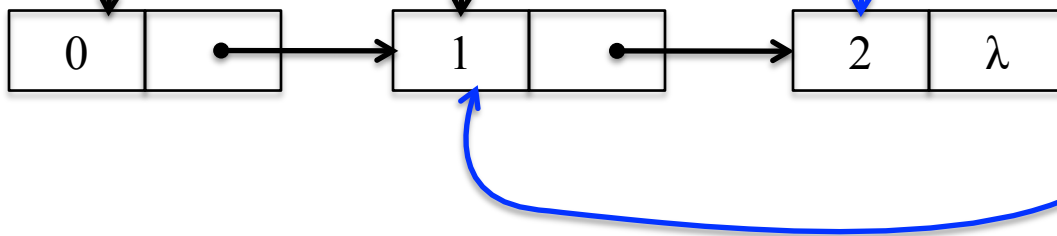


```

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)) {
10                  // Insertion succeeded, try advancing tail
11                  tail.compareAndSet(curTail, newNode);
12                  // will fail if tail already moved
13              }
14          }
15      }
16  }
17  }
18  }
19  }
20  }
21  }
22  }
23  }
24  }
25  }
26  }
27  }
28  }
29  }
30  }
31  }
32  }
33  }
34  }
35  }
36  }
37  }
38  }
39  }
40  }
41  }
42  }
43  }
44  }
45  }
46  }
47  }
48  }
49  }
50  }
51  }
52  }
53  }
54  }
55  }
56  }
57  }
58  }
59  }
60  }
61  }
62  }
63  }
64  }
65  }
66  }
67  }
68  }
69  }
70  }
71  }
72  }
73  }
74  }
75  }
76  }
77  }
78  }
79  }
80  }
81  }
82  }
83  }
84  }
85  }
86  }
87  }
88  }
89  }
90  }
91  }
92  }
93  }
94  }
95  }
96  }
97  }
98  }
99  }
100 }

```

Var	T1	T2
tailNext		λ
curTail		
newNode		
	head	
	tail	



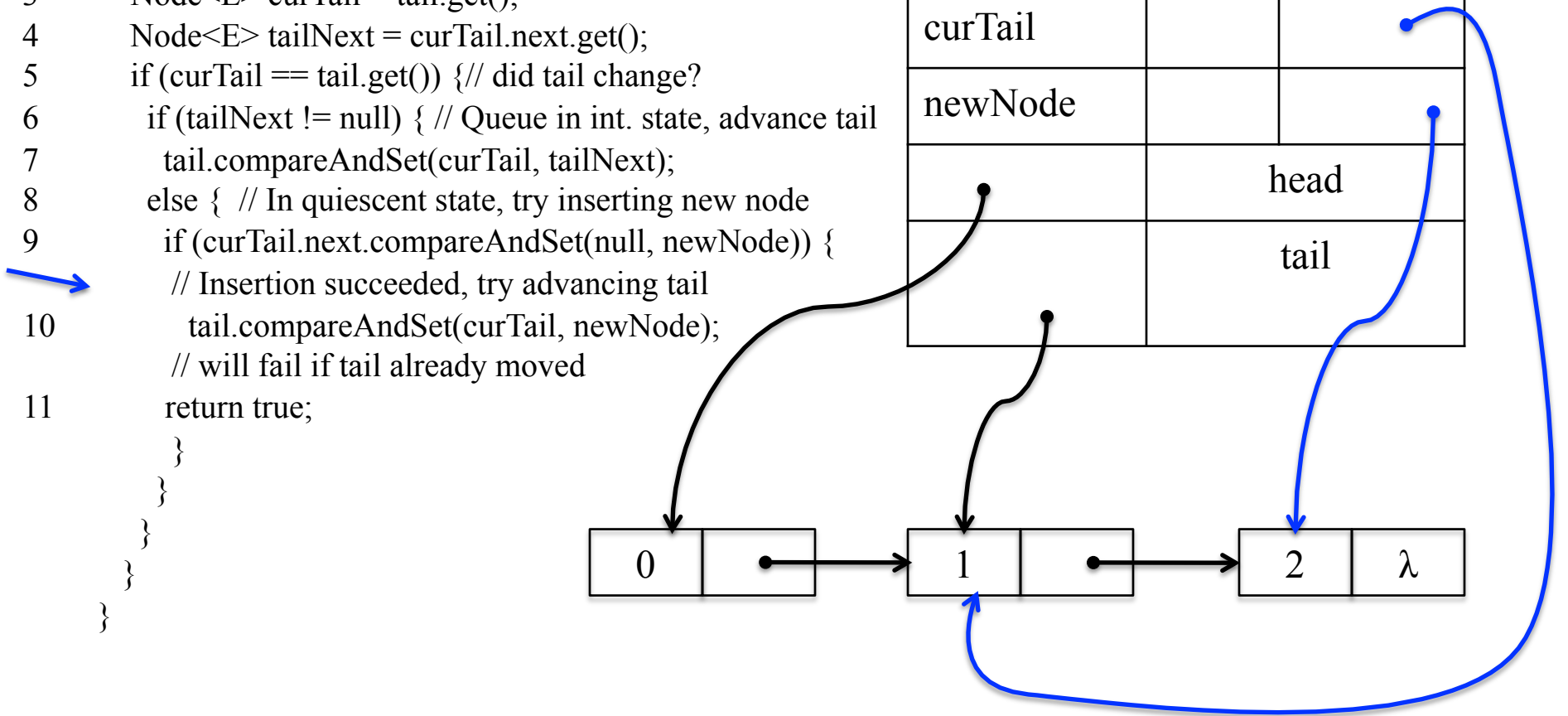
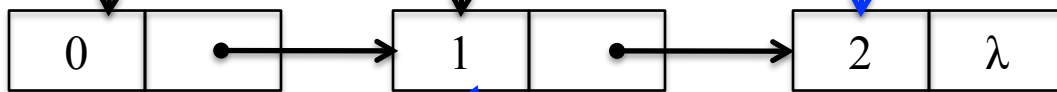
Trace 1: T2 Continues

```

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)) {
10                 // Insertion succeeded, try advancing tail
11                 tail.compareAndSet(curTail, newNode);
12                 // will fail if tail already moved
13                 return true;
14             }
15         }
16     }
17 }
18 }

```

Var	T1	T2
tailNext		λ
curTail		
newNode		
	head	
	tail	



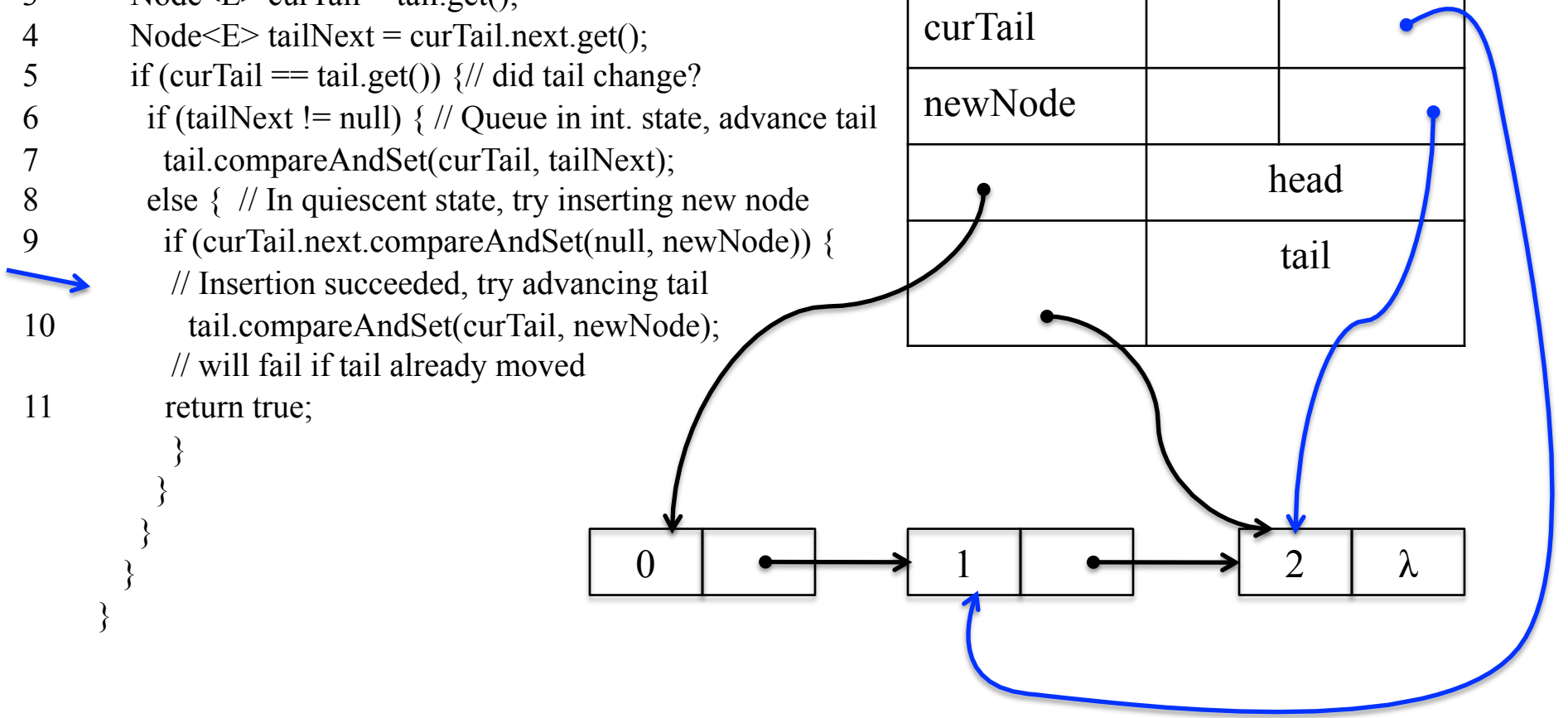
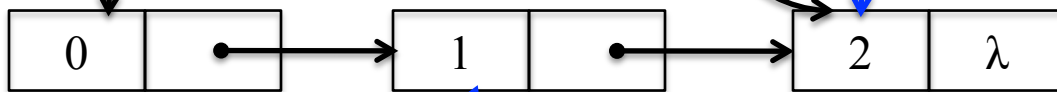
Trace 1: T2 Continues

```

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)) {
10                 // Insertion succeeded, try advancing tail
11                 tail.compareAndSet(curTail, newNode);
12                 // will fail if tail already moved
13                 return true;
14             }
15         }
16     }
17 }
18 }

```

Var	T1	T2
tailNext		λ
curTail		
newNode		
	head	
	tail	



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)) {
10                 // Insertion succeeded, try advancing tail
11                 tail.compareAndSet(curTail, newNode);
12                 // will fail if tail already moved
13                 return true;
14             }
15         }
16     }
17 }

```

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)) {
10                 // Insertion succeeded, try advancing tail
11                 tail.compareAndSet(curTail, newNode);
12                 // will fail if tail already moved
13                 return true;
14             }
15         }
16     }
17 }

```

Var	T1	T2
tailNext		
curTail		
newNode		
	head	
	tail	



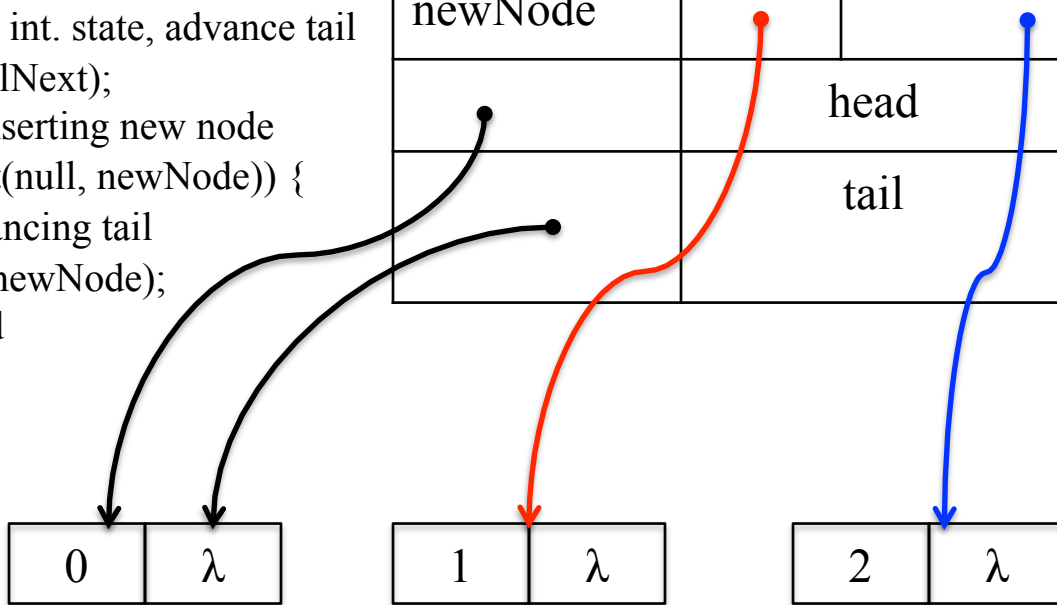
Trace 2

```

1 → public boolean put(E item) {
2   Node<E> newNode = new Node<E>(item, null);
3   while (true) {
4     Node<E> curTail = tail.get();
5     Node<E> tailNext = curTail.next.get();
6     if (curTail == tail.get()) { // did tail change?
7       if (tailNext != null) { // Queue in int. state, advance tail
8         tail.compareAndSet(curTail, tailNext);
9       } else { // In quiescent state, try inserting new node
10        if (curTail.next.compareAndSet(null, newNode)) {
11          // Insertion succeeded, try advancing tail
12          tail.compareAndSet(curTail, newNode);
13          // will fail if tail already moved
14          return true;
15        }
16      }
17    }
18  }
19 }

```

Var	T1	T2
tailNext		
curTail		
newNode		
		head
		tail



Trace 2

```

1 → public boolean put(E item) {
2 →   Node<E> newNode = new Node<E>(item, null);
3   while (true) {
4     Node<E> curTail = tail.get();
5     Node<E> tailNext = curTail.next.get();
6     if (curTail == tail.get()) { // did tail change?
7       if (tailNext != null) { // Queue in int. state, advance tail
8         tail.compareAndSet(curTail, tailNext);
9       } else { // In quiescent state, try inserting new node
10        if (curTail.next.compareAndSet(null, newNode)) {
11          // Insertion succeeded, try advancing tail
12          tail.compareAndSet(curTail, newNode);
13          // will fail if tail already moved
14          return true;
15        }
16      }
17    }
18  }
19 }

```

Var	T1	T2
tailNext		
curTail		
newNode		
		head
		tail



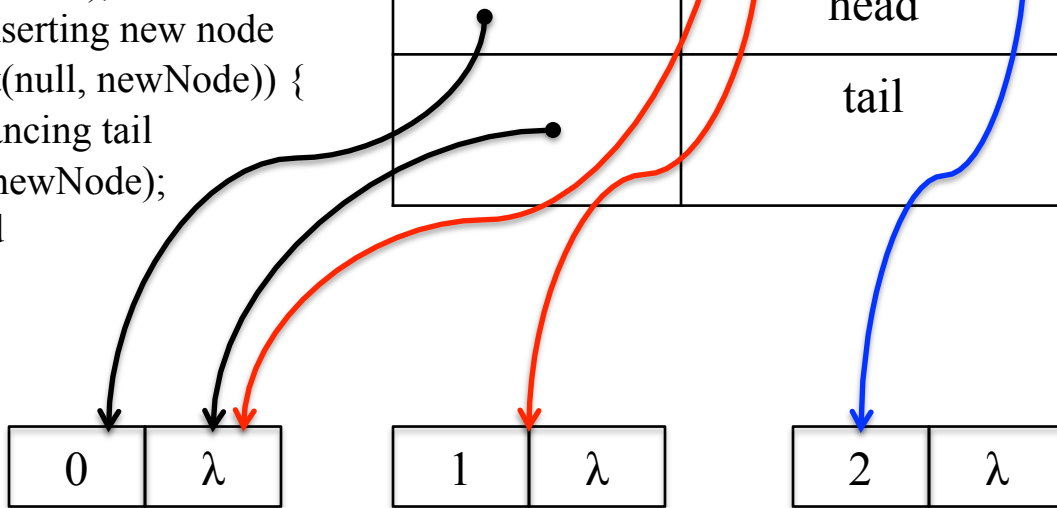
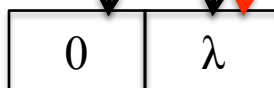
Trace 2

```

1 → public boolean put(E item) {
2   Node<E> newNode = new Node<E>(item, null);
3   while (true) {
4     Node<E> curTail = tail.get();
5     Node<E> tailNext = curTail.next.get();
6     if (curTail == tail.get()) { // did tail change?
7       if (tailNext != null) { // Queue in int. state, advance tail
8         tail.compareAndSet(curTail, tailNext);
9       } else { // In quiescent state, try inserting new node
10        if (curTail.next.compareAndSet(null, newNode)) {
11          // Insertion succeeded, try advancing tail
12          tail.compareAndSet(curTail, newNode);
13          // will fail if tail already moved
14          return true;
15        }
16      }
17    }
18  }
19 }

```

Var	T1	T2
tailNext	λ	
curTail		
newNode		
		head
		tail



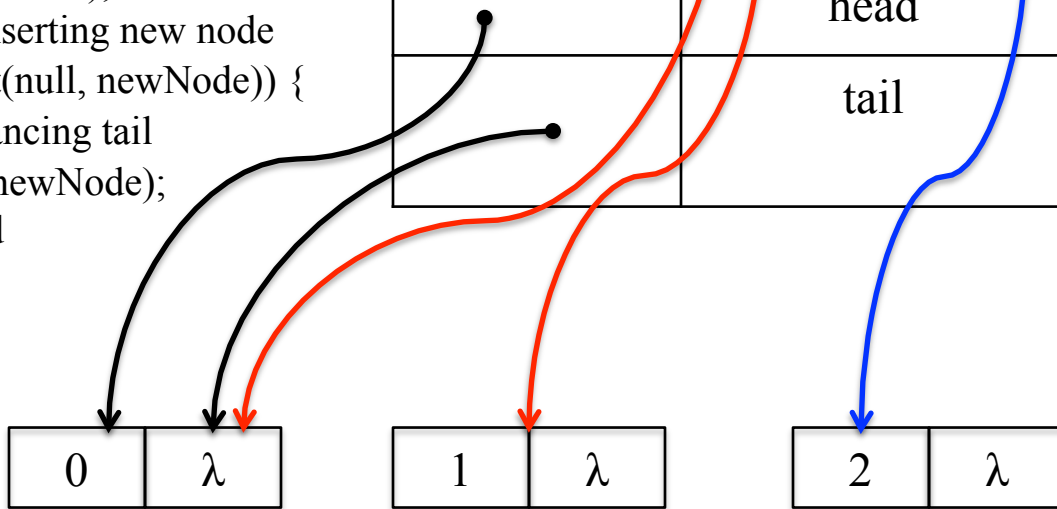
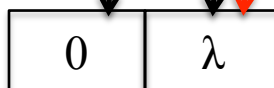
Trace 2

```

1 → public boolean put(E item) {
2   Node<E> newNode = new Node<E>(item, null);
3   while (true) {
4     Node<E> curTail = tail.get();
5     Node<E> tailNext = curTail.next.get();
6     if (curTail == tail.get()) { // did tail change?
7       if (tailNext != null) { // Queue in int. state, advance tail
8         tail.compareAndSet(curTail, tailNext);
9       } else { // In quiescent state, try inserting new node
10        if (curTail.next.compareAndSet(null, newNode)) {
11          // Insertion succeeded, try advancing tail
12          tail.compareAndSet(curTail, newNode);
13          // will fail if tail already moved
14          return true;
15        }
16      }
17    }
18  }
19 }

```

Var	T1	T2
tailNext	λ	
curTail		
newNode		
		head
		tail



Trace 2

```

1 → public boolean put(E item) {
2   Node<E> newNode = new Node<E>(item, null);
3   while (true) {
4     Node<E> curTail = tail.get();
5     Node<E> tailNext = curTail.next.get();
6     if (curTail == tail.get()) { // did tail change?
7       if (tailNext != null) { // Queue in int. state, advance tail
8         tail.compareAndSet(curTail, tailNext);
9       } else { // In quiescent state, try inserting new node
10        if (curTail.next.compareAndSet(null, newNode)) {
11          // Insertion succeeded, try advancing tail
12          tail.compareAndSet(curTail, newNode);
13          // will fail if tail already moved
14          return true;
15        }
16      }
17    }
18  }
19 }

```

Var	T1	T2
tailNext	λ	
curTail		
newNode		
	head	
	tail	



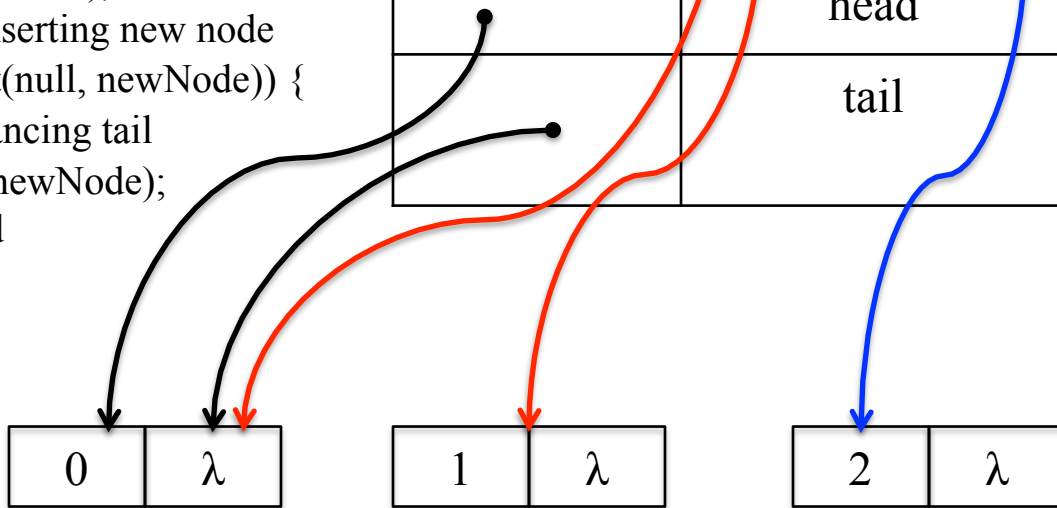
Trace 2

```

1 → public boolean put(E item) {
2   Node<E> newNode = new Node<E>(item, null);
3   while (true) {
4     Node<E> curTail = tail.get();
5     Node<E> tailNext = curTail.next.get();
6     if (curTail == tail.get()) { // did tail change?
7       if (tailNext != null) { // Queue in int. state, advance tail
8         tail.compareAndSet(curTail, tailNext);
9       } else { // In quiescent state, try inserting new node
10        if (curTail.next.compareAndSet(null, newNode)) {
11          // Insertion succeeded, try advancing tail
12          tail.compareAndSet(curTail, newNode);
13          // will fail if tail already moved
14          return true;
15        }
16      }
17    }
18  }
19 }

```

Var	T1	T2
tailNext	λ	
curTail		
newNode		
	head	
	tail	



Trace 2: After CAS

```

1  public boolean put(E item) {
2      Node<E> newNode = new Node<E>(item, null);
3      while (true) {
4          Node<E> curTail = tail.get();
5          Node<E> tailNext = curTail.next.get();
6          if (curTail == tail.get()) { // did tail change?
7              if (tailNext != null) { // Queue in int. state, advance tail
8                  tail.compareAndSet(curTail, tailNext);
9              }
10             else { // In quiescent state, try inserting new node
11                 if (curTail.next.compareAndSet(null, newNode)) {
12                     // Insertion succeeded, try advancing tail
13                     tail.compareAndSet(curTail, newNode);
14                     // will fail if tail already moved
15                     return true;
16                 }
17             }
18         }
19     }
20 }

```

Var	T1	T2
tailNext	λ	
curTail		
newNode		
		head
		tail



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)) {
10                 // Insertion succeeded, try advancing tail
11                 tail.compareAndSet(curTail, newNode);
12                 // will fail if tail already moved
13                 return true;
14             }
15         }
16     }
17 }
18 }

```

Var	T1	T2
tailNext	λ	
curTail		
newNode		
		head
		tail



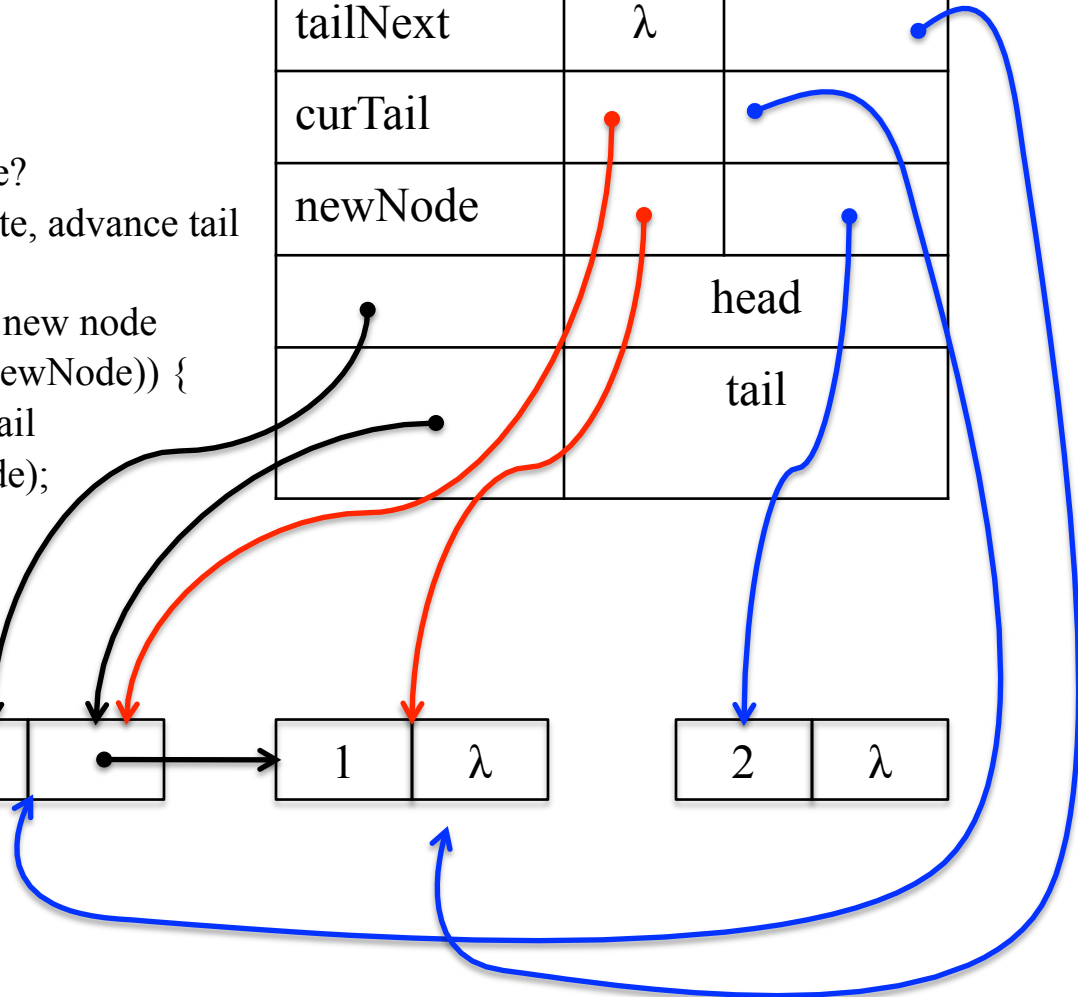
Trace 2

```

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)) {
10          // Insertion succeeded, try advancing tail
11          tail.compareAndSet(curTail, newNode);
          // will fail if tail already moved
        }
        return true;
      }
    }
  }
}

```

Var	T1	T2
tailNext	λ	
curTail		
newNode		
head		
tail		



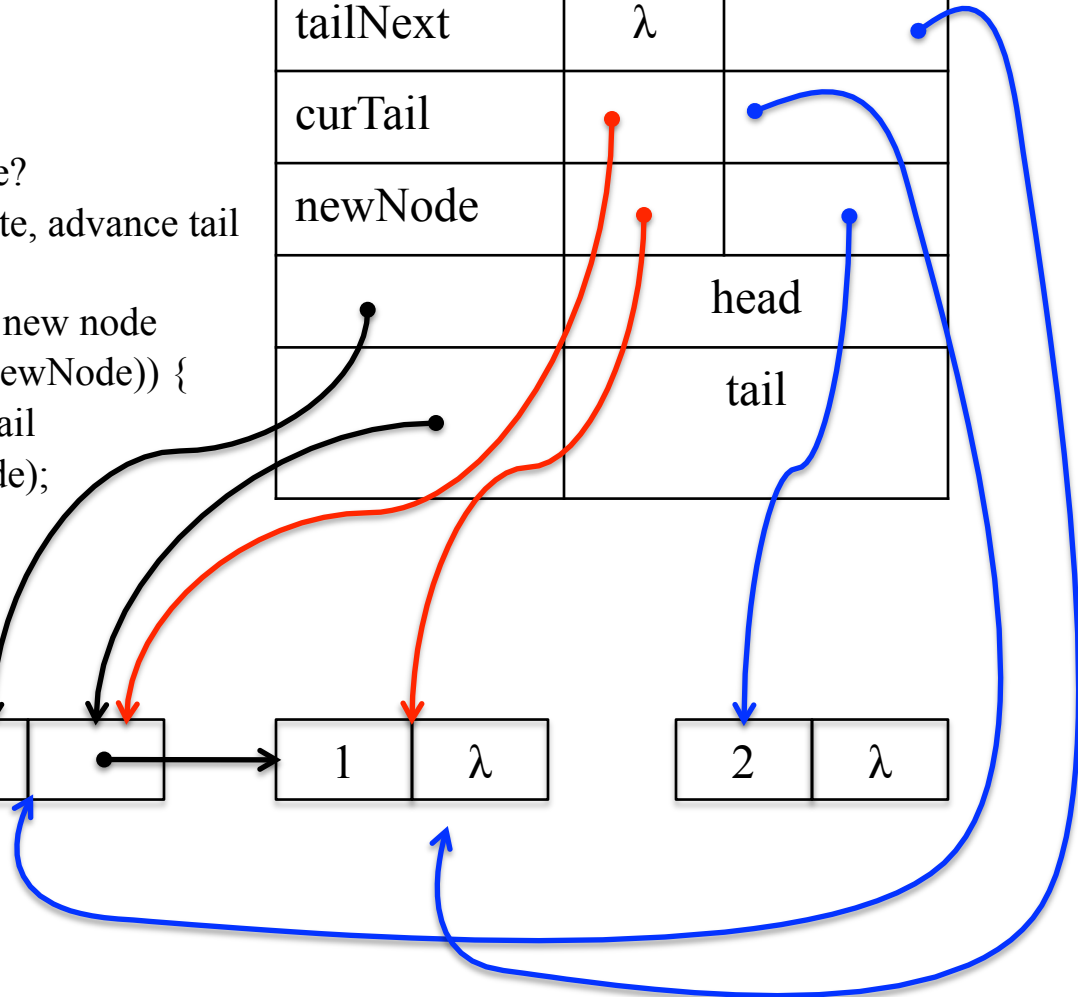
Trace 2

```

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)) {
10                 // Insertion succeeded, try advancing tail
11                 tail.compareAndSet(curTail, newNode);
12                 // will fail if tail already moved
13                 return true;
14             }
15         }
16     }
17 }
18 }

```

Var	T1	T2
tailNext	λ	
curTail		
newNode		
head		
tail		



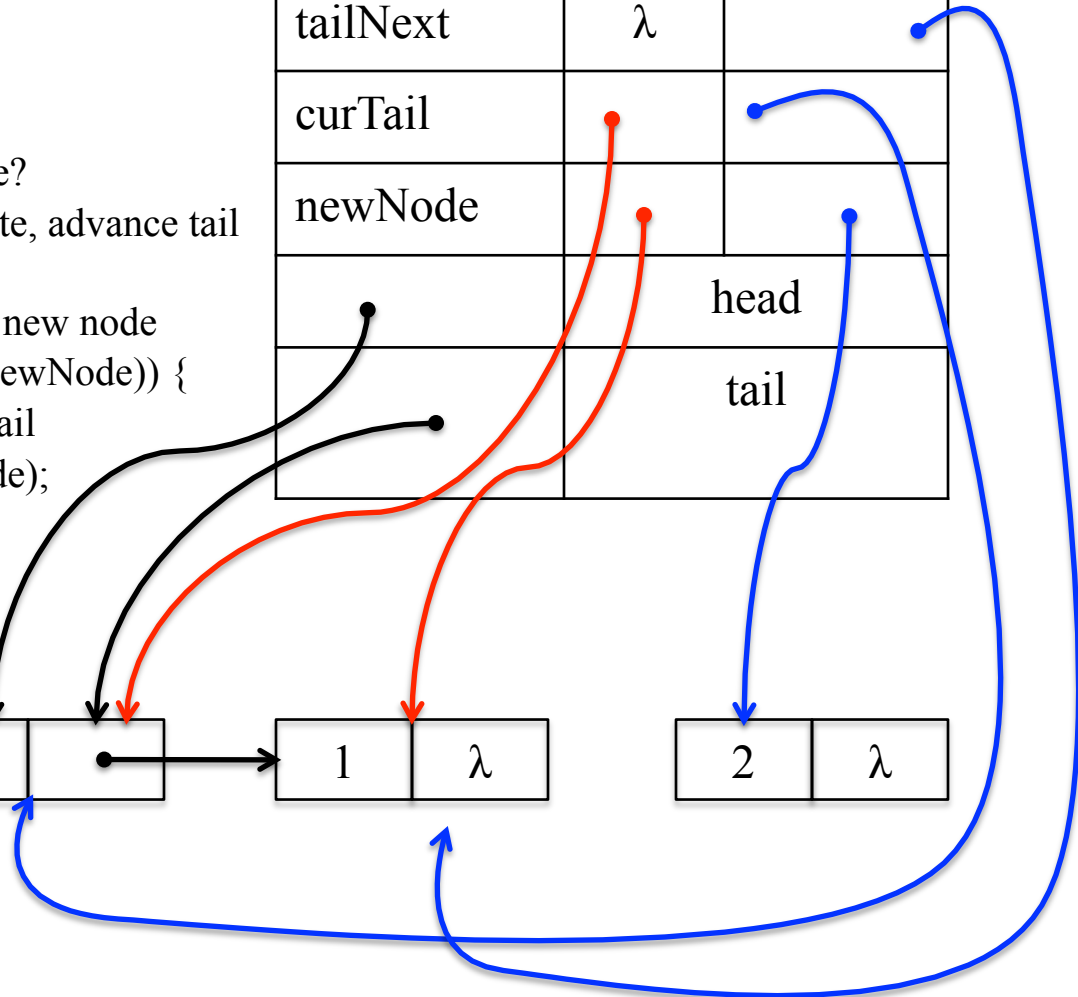
Trace 2

```

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)) {
10                 // Insertion succeeded, try advancing tail
11                 tail.compareAndSet(curTail, newNode);
12                 // will fail if tail already moved
13                 return true;
14             }
15         }
16     }
17 }

```

Var	T1	T2
tailNext	λ	
curTail		
newNode		
head		
tail		



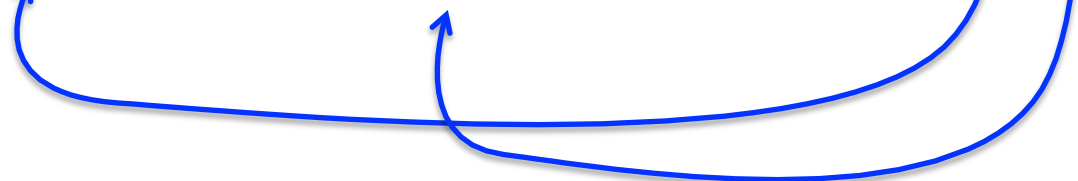
Trace 2: T2 wins

```

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          }
9          else { // In quiescent state, try inserting new node
10             if (curTail.next.compareAndSet(null, newNode)) {
11                 // Insertion succeeded, try advancing tail
12                 tail.compareAndSet(curTail, newNode);
13                 // will fail if tail already moved
14                 return true;
15             }
16         }
17     }
18 }

```

Var	T1	T2
tailNext	λ	
curTail		
newNode		
		head
		tail



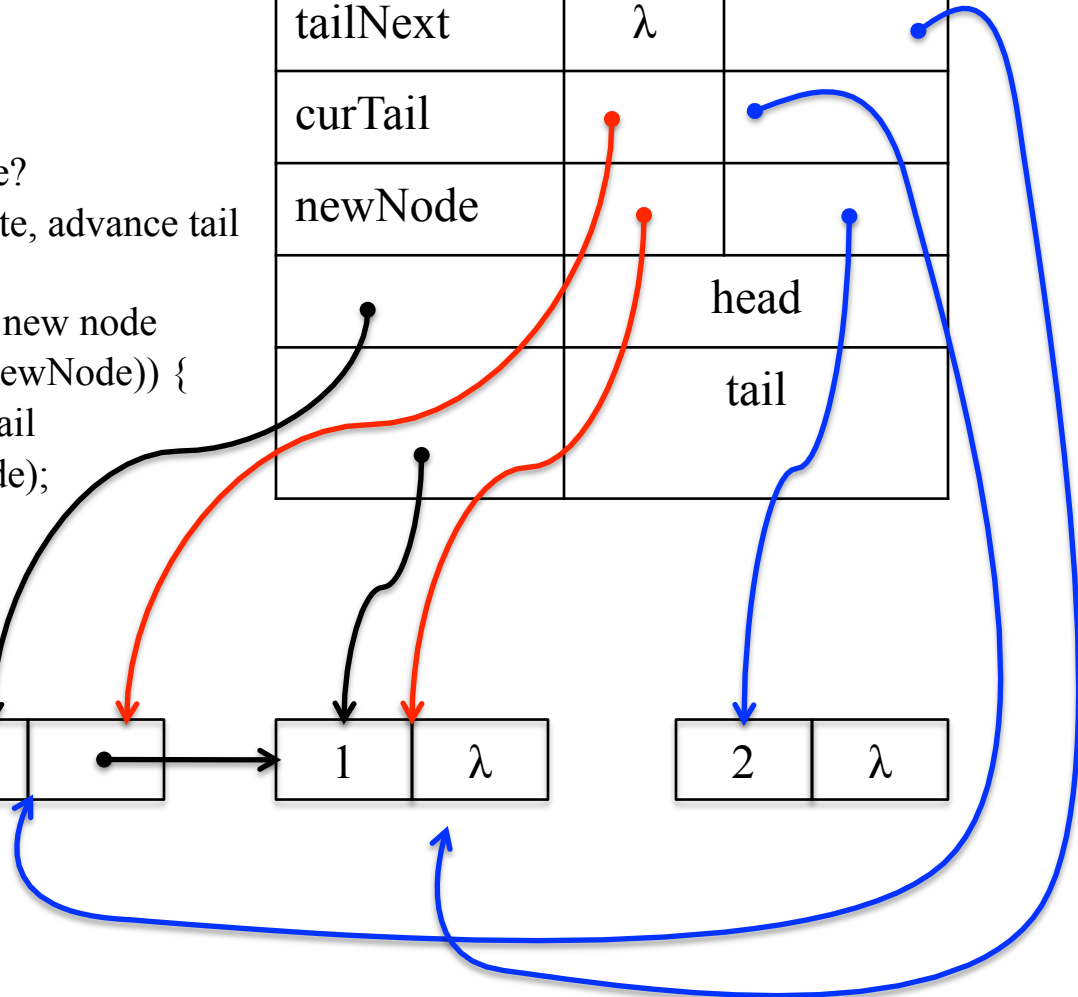
Trace 2: After CAS

```

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)) {
10                 // Insertion succeeded, try advancing tail
11                 tail.compareAndSet(curTail, newNode);
12                 // will fail if tail already moved
13                 return true;
14             }
15         }
16     }
17 }
18 }

```

Var	T1	T2
tailNext	λ	
curTail		
newNode		
head		
tail		



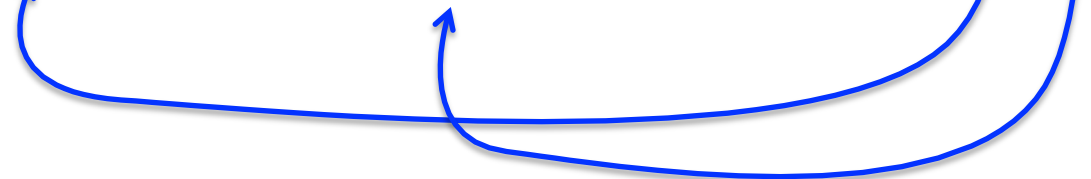
Trace 2: T1 continues: CAS fails

```

1  public boolean put(E item) {
2      Node<E> newNode = new Node<E>(item, null);
3      while (true) {
4          Node<E> curTail = tail.get();
5          Node<E> tailNext = curTail.next.get();
6          if (curTail == tail.get()) { // did tail change?
7              if (tailNext != null) { // Queue in int. state, advance tail
8                  tail.compareAndSet(curTail, tailNext);
9              }
10             if (curTail.next.compareAndSet(null, newNode)) {
11                 // Insertion succeeded, try advancing tail
12                 tail.compareAndSet(curTail, newNode);
13                 // will fail if tail already moved
14                 return true;
15             }
16         }
17     }
18 }

```

Var	T1	T2
tailNext	λ	
curTail		
newNode		
head		
tail		



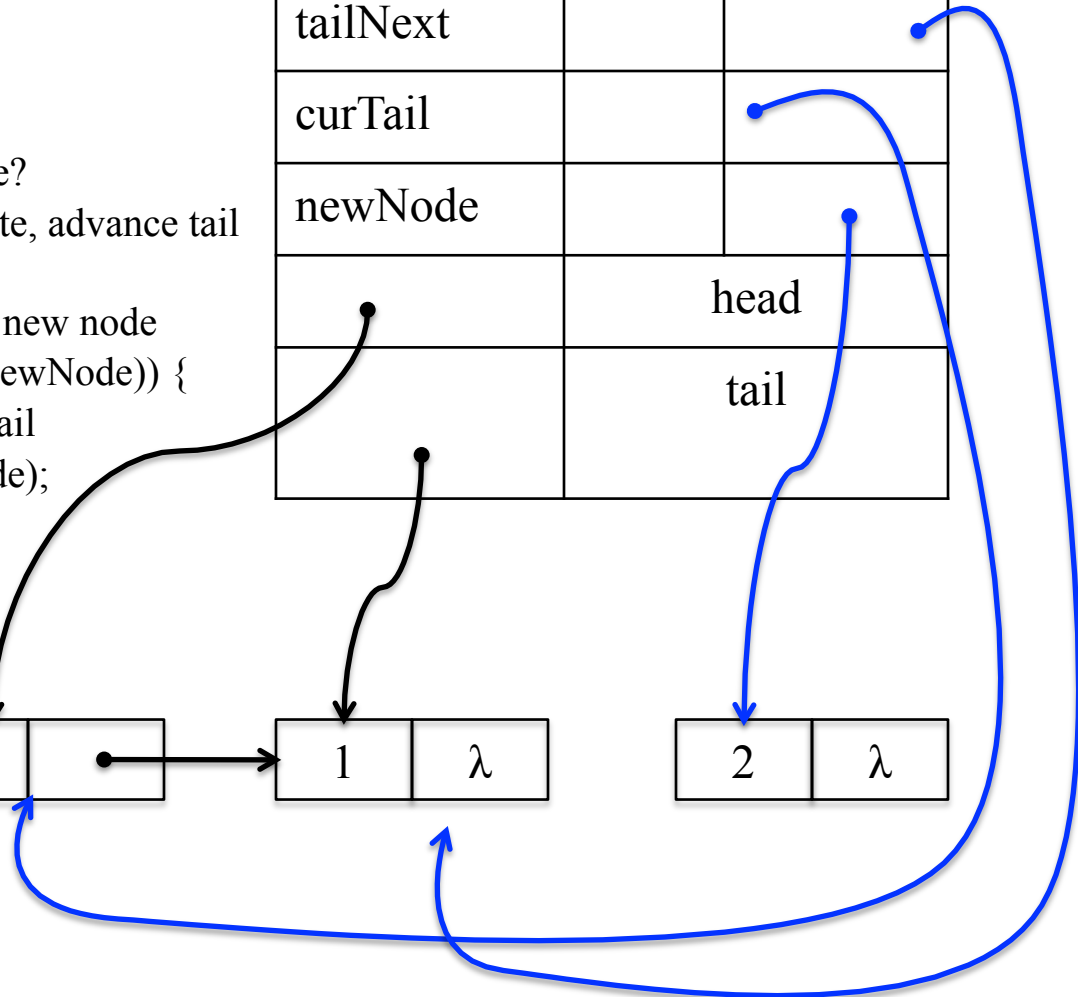
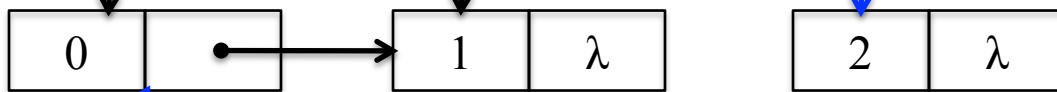
Trace 2: T1 exits

```

1  public boolean put(E item) {
2      Node<E> newNode = new Node<E>(item, null);
3      while (true) {
4          Node<E> curTail = tail.get();
5          Node<E> tailNext = curTail.next.get();
6          if (curTail == tail.get()) { // did tail change?
7              if (tailNext != null) { // Queue in int. state, advance tail
8                  tail.compareAndSet(curTail, tailNext);
9              }
10             else { // In quiescent state, try inserting new node
11                 if (curTail.next.compareAndSet(null, newNode)) {
12                     // Insertion succeeded, try advancing tail
13                     tail.compareAndSet(curTail, newNode);
14                     // will fail if tail already moved
15                     return true;
16                 }
17             }
18         }
19     }
20 }

```

Var	T1	T2
tailNext		
curTail		
newNode		
	head	
	tail	



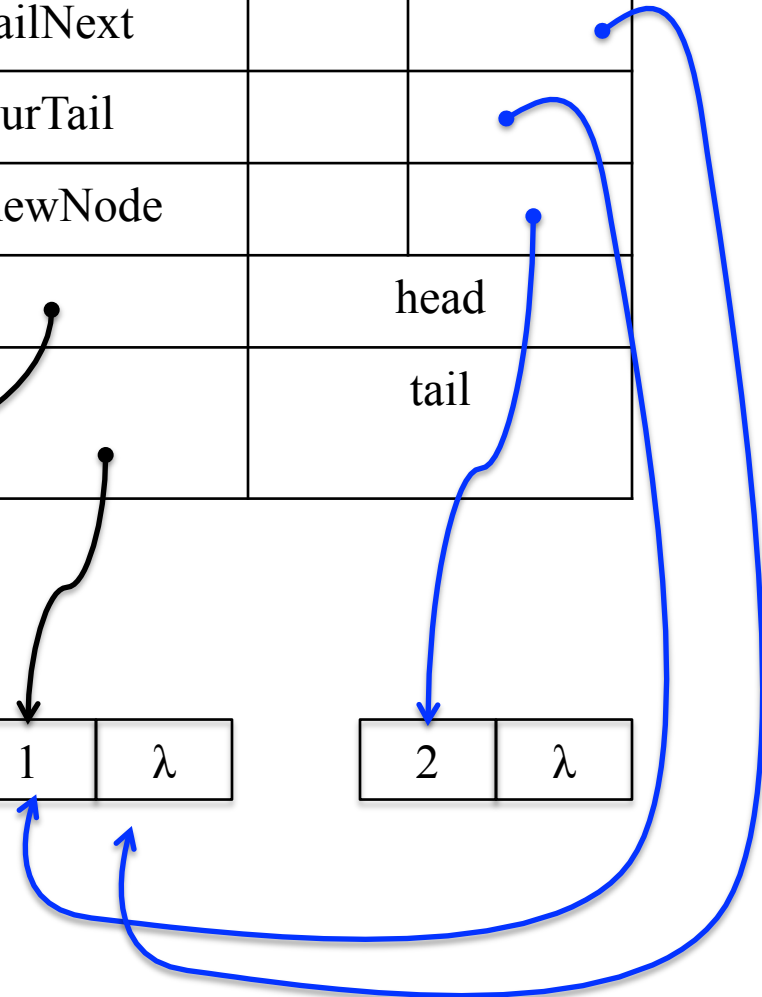
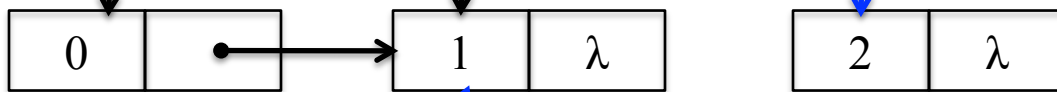
Trace 2: T2 continues

```

1  public boolean put(E item) {
2      Node<E> newNode = new Node<E>(item, null);
3      while (true) {
4          Node<E> curTail = tail.get();
5          Node<E> tailNext = curTail.next.get();
6          if (curTail == tail.get()) { // did tail change?
7              if (tailNext != null) { // Queue in int. state, advance tail
8                  tail.compareAndSet(curTail, tailNext);
9              } else { // In quiescent state, try inserting new node
10                 if (curTail.next.compareAndSet(null, newNode)) {
11                     // Insertion succeeded, try advancing tail
12                     tail.compareAndSet(curTail, newNode);
13                     // will fail if tail already moved
14                     return true;
15                 }
16             }
17         }
18     }
19 }

```

Var	T1	T2
tailNext		
curTail		
newNode		
	head	
	tail	



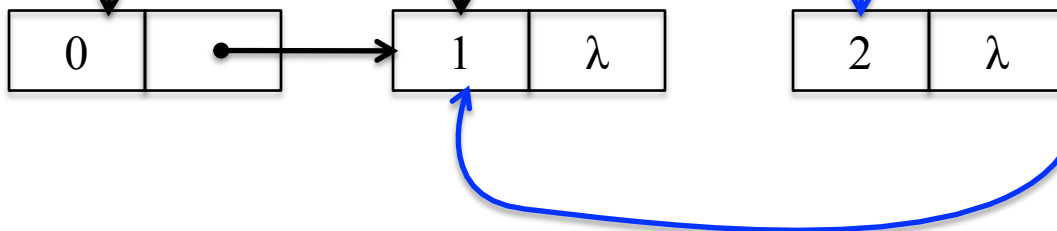
Trace 2: T2 continues

```

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)) {
10          // Insertion succeeded, try advancing tail
11          tail.compareAndSet(curTail, newNode);
12          // will fail if tail already moved
13          return true;
14        }
15      }
16    }
17  }
18 }

```

Var	T1	T2
tailNext		λ
curTail		
newNode		
		head
		tail



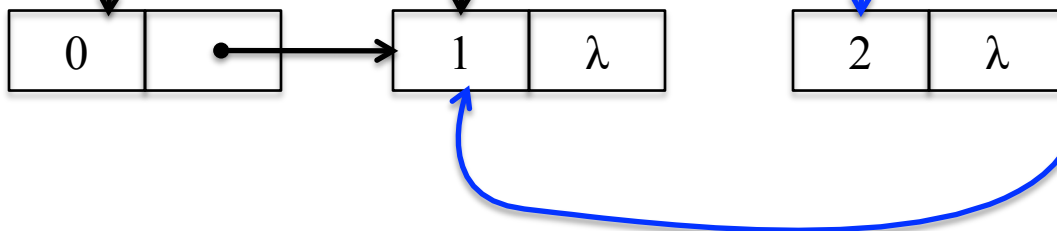
Trace 2: T2 continues

```

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)) {
10                 // Insertion succeeded, try advancing tail
11                 tail.compareAndSet(curTail, newNode);
12                 // will fail if tail already moved
13                 return true;
14             }
15         }
16     }
17 }
18 }

```

Var	T1	T2
tailNext		λ
curTail		
newNode		
		head
		tail



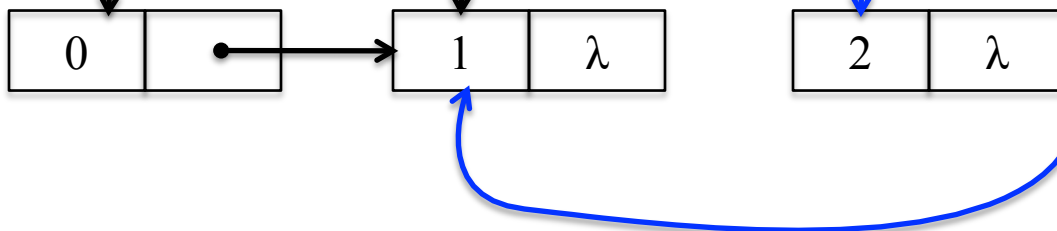
Trace 2: T2 continues

```

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)) {
10                 // Insertion succeeded, try advancing tail
11                 tail.compareAndSet(curTail, newNode);
12                 // will fail if tail already moved
13                 return true;
14             }
15         }
16     }
17 }

```

Var	T1	T2
tailNext		λ
curTail		
newNode		
		head
		tail



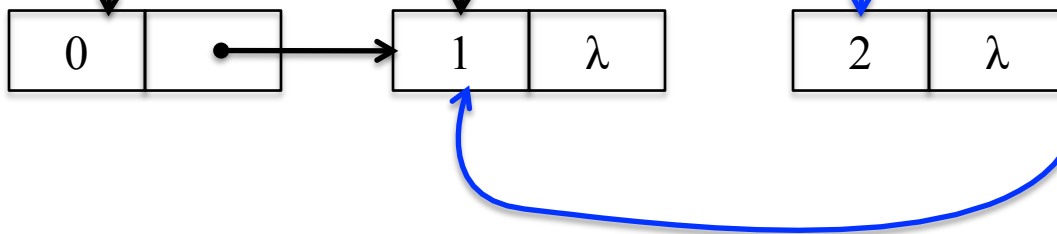
Trace 2: T2 continues

```

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)) {
10                 // Insertion succeeded, try advancing tail
11                 tail.compareAndSet(curTail, newNode);
12                 // will fail if tail already moved
13                 return true;
14             }
15         }
16     }
17 }

```

Var	T1	T2
tailNext		λ
curTail		
newNode		
	head	
	tail	



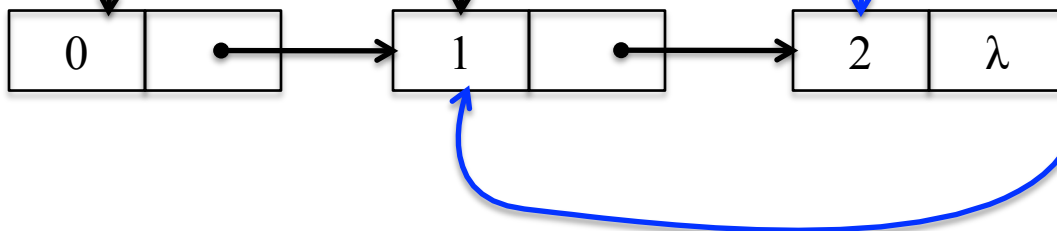
Trace 2: After CAS

```

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)) {
10                 // Insertion succeeded, try advancing tail
11                 tail.compareAndSet(curTail, newNode);
12                 // will fail if tail already moved
13                 return true;
14             }
15         }
16     }
17 }
18 }

```

Var	T1	T2
tailNext		λ
curTail		
newNode		
head		
tail		



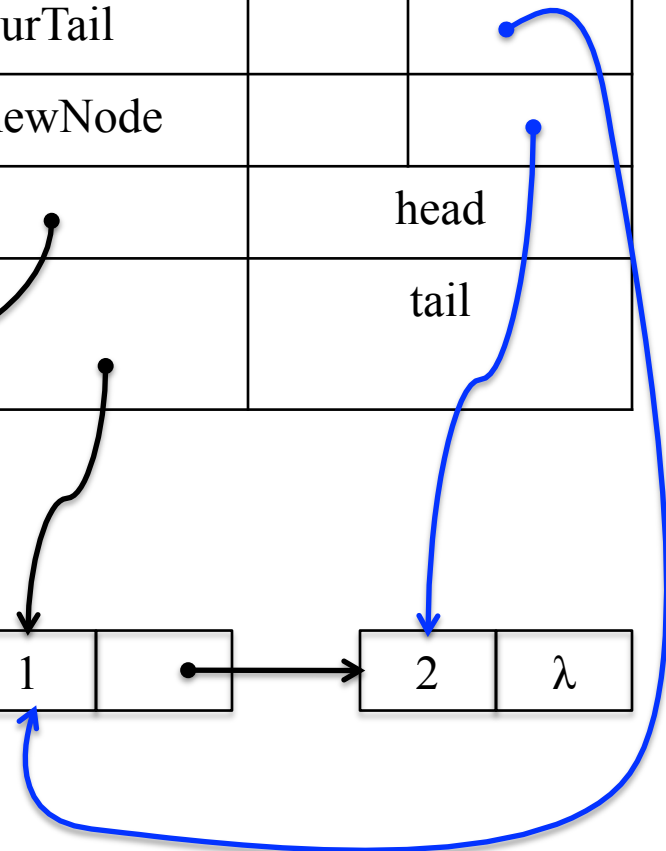
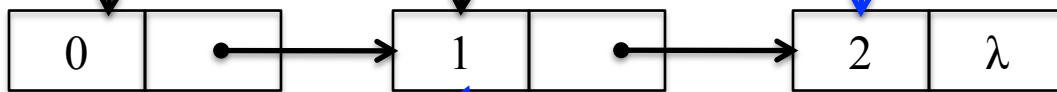
Trace 2: T2 continues

```

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)) {
10                 // Insertion succeeded, try advancing tail
11                 tail.compareAndSet(curTail, newNode);
12                 // will fail if tail already moved
13                 return true;
14             }
15         }
16     }
17 }
18 }

```

Var	T1	T2
tailNext		λ
curTail		
newNode		
		head
		tail



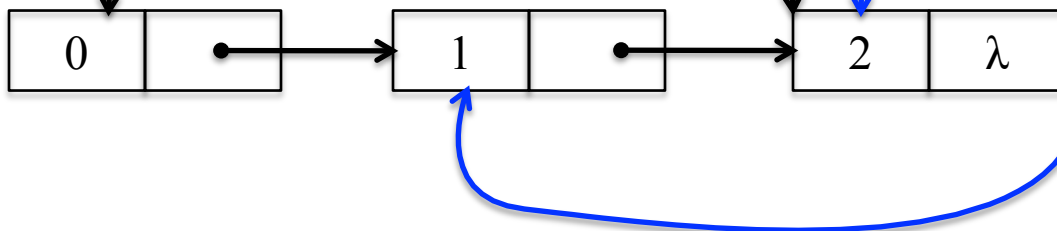
Trace 2: After CAS

```

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)) {
10                 // Insertion succeeded, try advancing tail
11                 tail.compareAndSet(curTail, newNode);
12                 // will fail if tail already moved
13                 return true;
14             }
15         }
16     }
17 }
18 }

```

Var	T1	T2
tailNext		λ
curTail		
newNode		
head		
tail		



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)) {
10                 // Insertion succeeded, try advancing tail
11                 tail.compareAndSet(curTail, newNode);
12                 // will fail if tail already moved
13                 return true;
14             }
15         }
16     }
17 }
18 }

```

Var	T1	T2
tailNext		
curTail		
newNode		
	head	
	tail	

