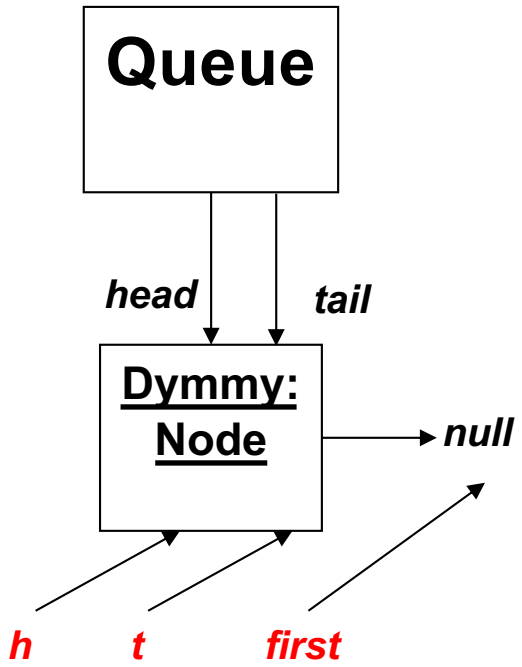
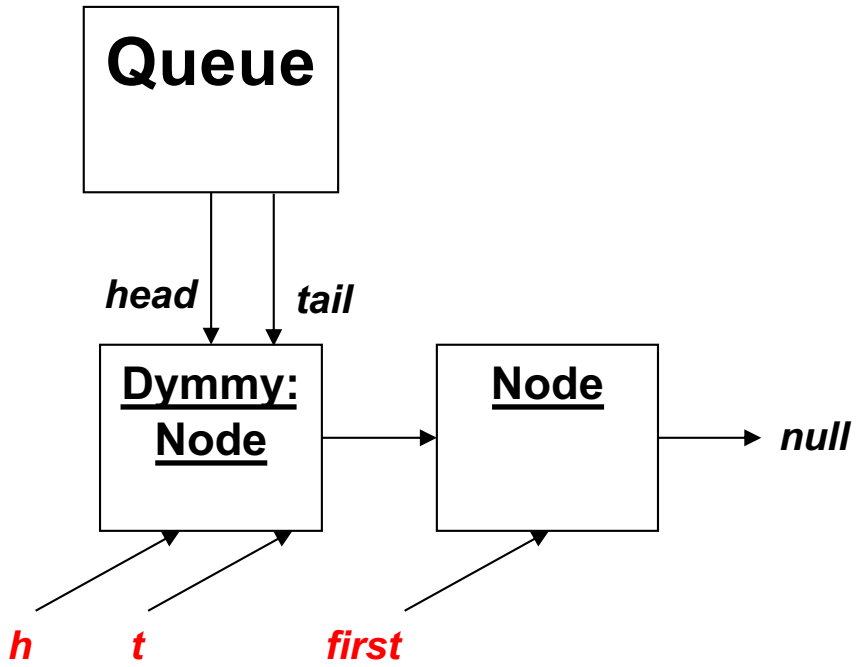
A7: Non-blocking Queue

- **Case 1: get on empty queue**



A7: Non-blocking Queue

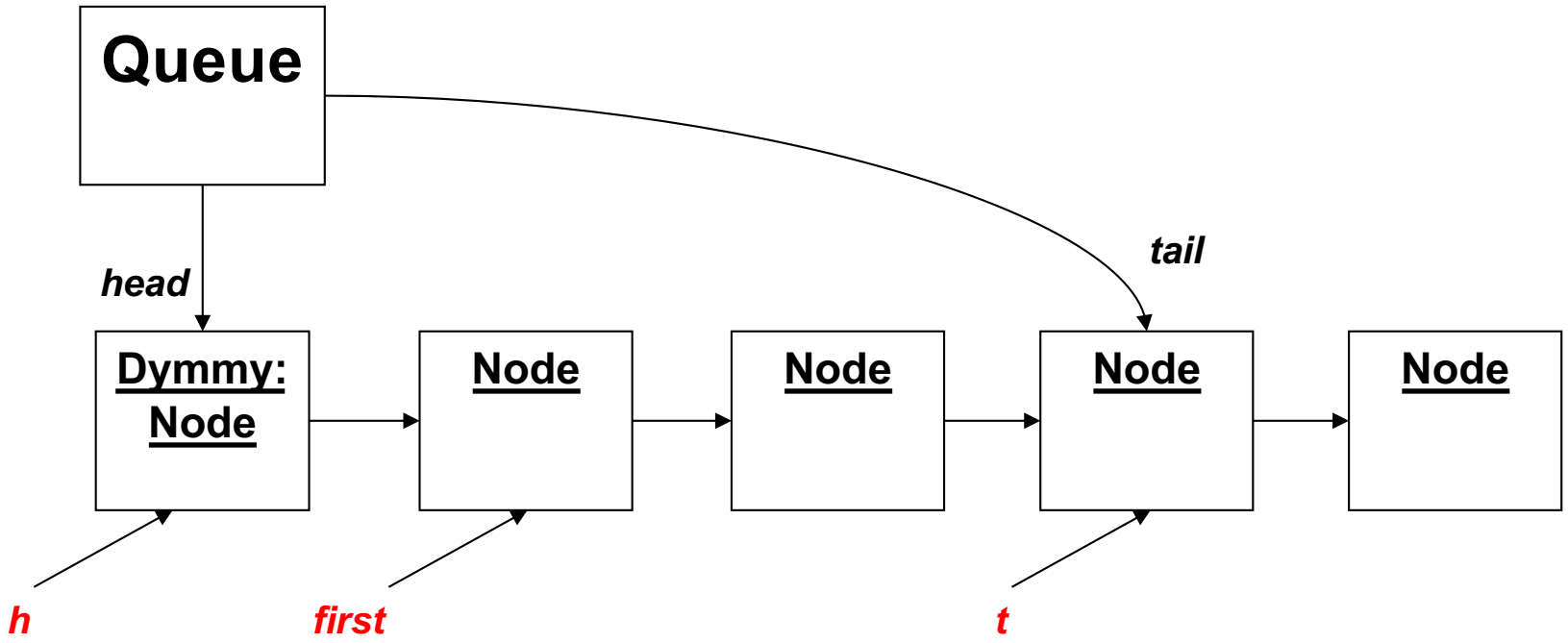
- **Case 2: get on empty queue with half inserted element**



Invariant
- tail refers to dummy OR
- tail refers to the last element OR
- tail refers to second-last element (in the middle of an update)

A7: Non-blocking Queue

- **Case 3: get on queue with >= 1 element**



A7: Non-blocking Queue

```
public E get() {
   while (true) {
      Node<E> h = head.get();
      Node<E> t = tail.get();
      Node<E> first = h.next.get();

      if (h == t) { // Case 1 or 2
          if (first == null) { // Case 1
             throw new NoSuchElementException();
          } else { //
             // Case 2: Move tail towards the end
             tail.compareAndSet(t, first);
          }
       } else if (head.compareAndSet(h, first)) { // Case 3
          E item = first.item;
          first.item = null; // GC reasons
          return item;
       }
   }
}
```

A7: CopyOnWrite List

```
public class CoWLinkedListSolution<E> implements CoWList<E> {
  private final AtomicReference<List<E>> atomic =
    new AtomicReference<List<E>>(new LinkedList<E>());

  public int size() { return atomic.get().size(); }

  public void addFirst(E e) {
    while(true) {
      List<E> current = atomic.get();
      LinkedList<E> modified = new LinkedList<>(current);
      modified.addFirst(e);
      if(atomic.compareAndSet(current, modified)) return;
    }
  }

  …
```

A7: CopyOnWrite List

```
…

public void removeFirst() {
  while(true) {
    List<E> current = atomic.get();
    LinkedList<E> modified = new LinkedList<>(current);
    modified.removeFirst();
    if(atomic.compareAndSet(current, modified)) return;
  }
}

…
```

A7: CopyOnWrite List

```
  …

  public Iterator<E> iterator() {
    final Iterator<E> it = atomic.get().iterator();
    return new Iterator<E>() {
      @Override
      public boolean hasNext() { return it.hasNext(); }
      @Override
      public E next() { return it.next(); }
      @Override
      public void remove() {
        throw new UnsupportedOperationException();
      }
    };
  }

}
```