

# JAVA

МНОГОПОТОЧНОСТЬ.  
ОЧЕРЕДЬ МАЙКЛА-СКОТТА



## ConcurrentQueue



synchronized

ReentrantLock



volatile

AtomicReference<E>

```
private static final class Node<E> {
```

1 usage

```
    private volatile E value;
```

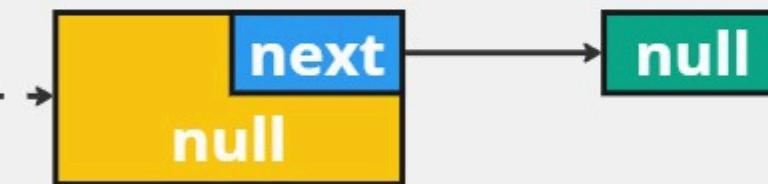
no usages

```
    private volatile Node<E> next;
```

1 usage

```
public Node() {
```

```
}
```



no usages

```
public Node(final E value) {
```

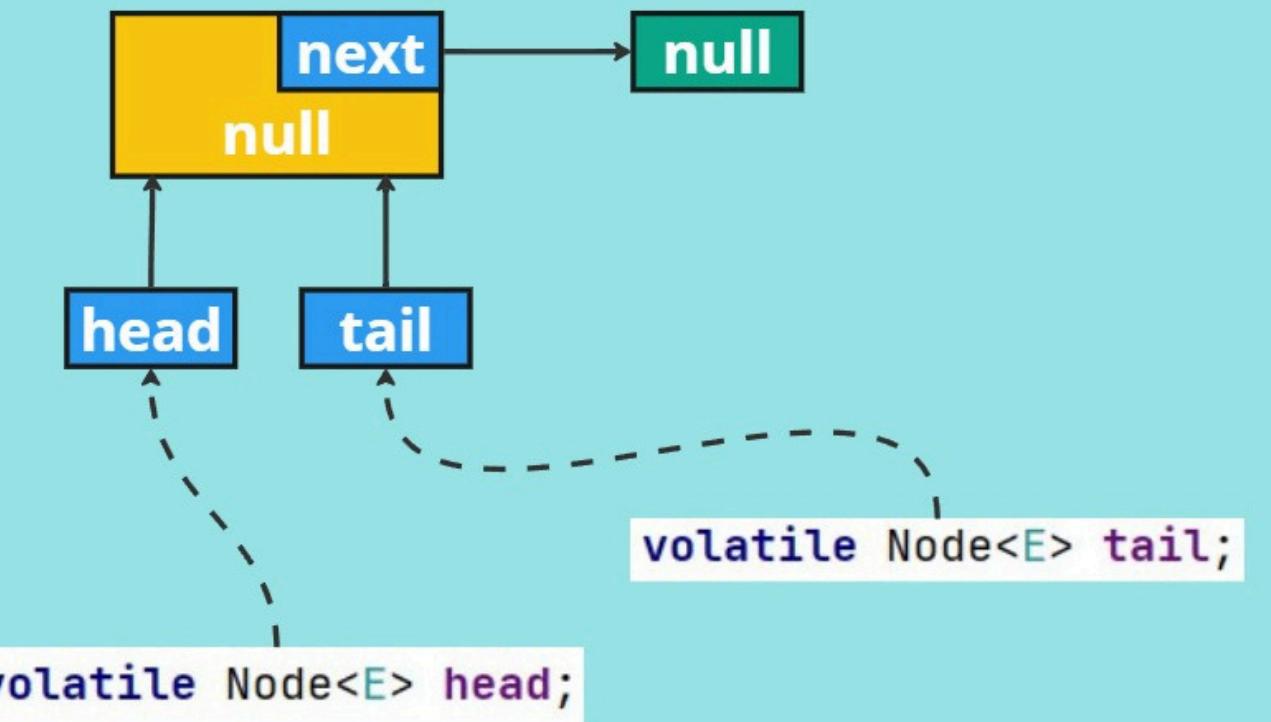
```
    this.value = value;
```

```
}
```



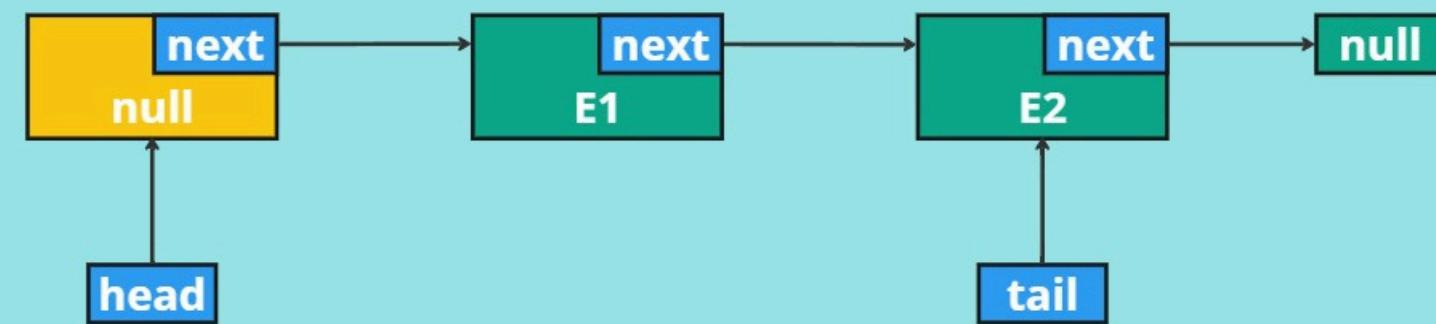
```
}
```

## ConcurrentQueue



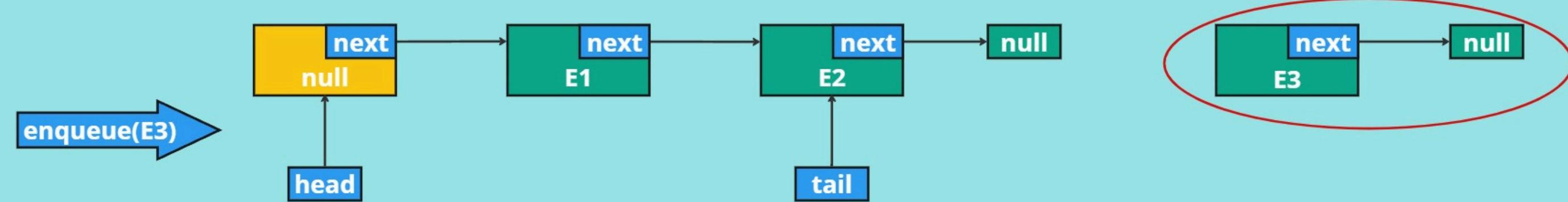
```
public ConcurrentQueue() {
    final Node<E> dummyNode = new Node<>();
    head = dummyNode;
    tail = dummyNode;
}
```

## ConcurrentQueue



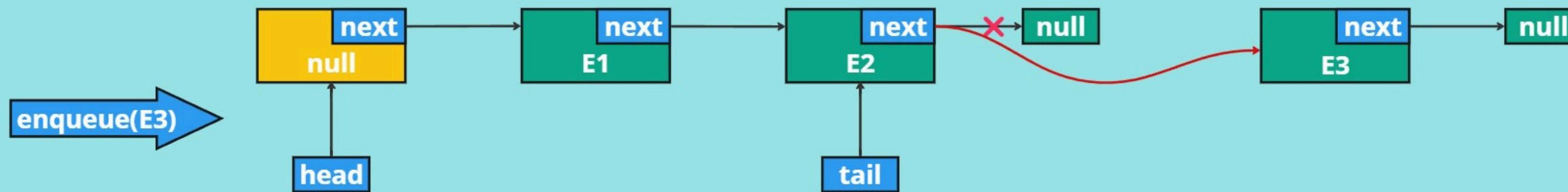
```
public void enqueue(final E element) {  
    //create new node  
    //tail.next should be assigned new node  
    //tail should be assigned new node  
}
```

## ConcurrentQueue



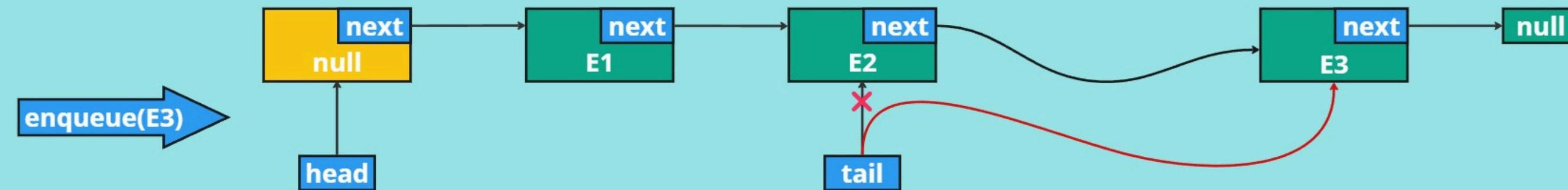
```
public void enqueue(final E element) {  
    //create new node  
    //tail.next should be assigned new node  
    //tail should be assigned new node  
}
```

## ConcurrentQueue



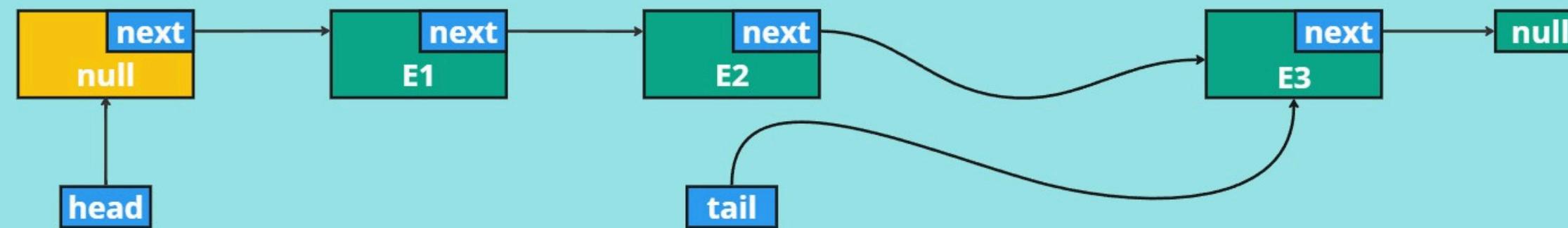
```
public void enqueue(final E element) {  
    //create new node  
    //tail.next should be assigned new node  
    //tail should be assigned new node  
}
```

## ConcurrentQueue



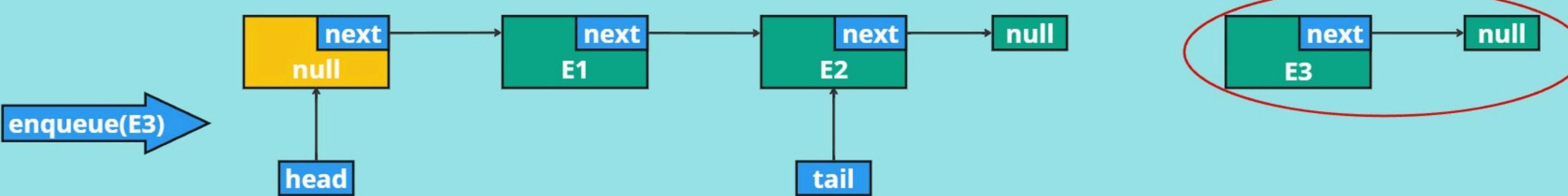
```
public void enqueue(final E element) {  
    //create new node  
    //tail.next should be assigned new node  
    //tail should be assigned new node  
}
```

## ConcurrentQueue

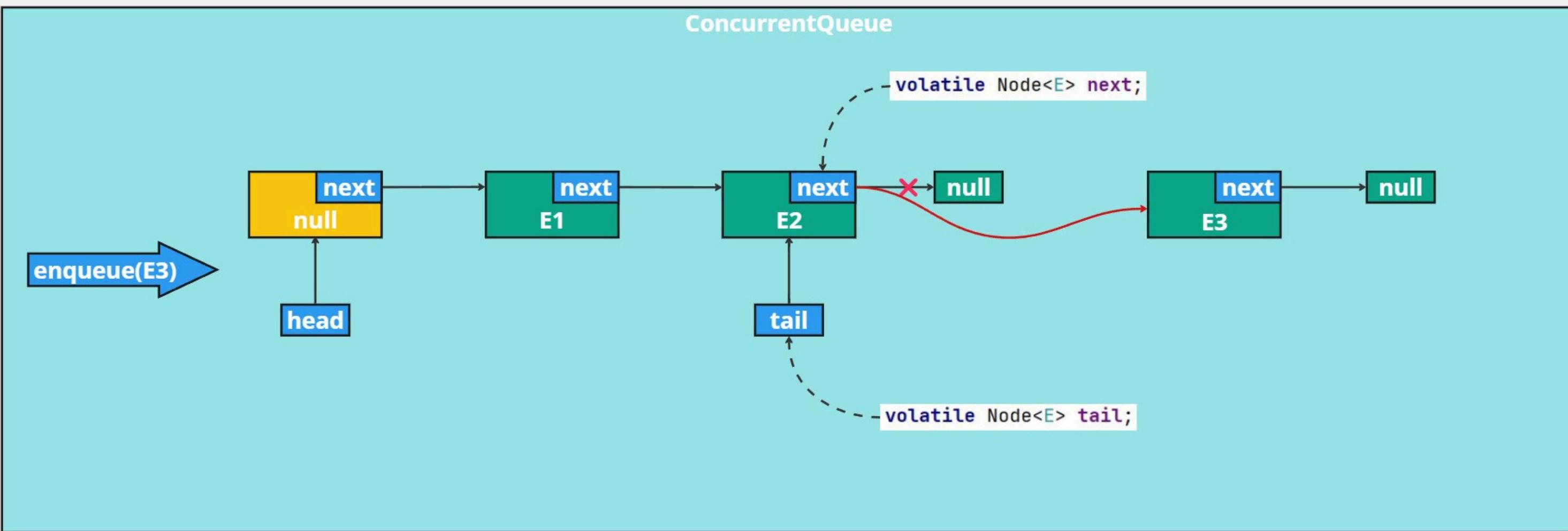


```
public void enqueue(final E element) {  
    //create new node  
    //tail.next should be assigned new node  
    //tail should be assigned new node  
}
```

## ConcurrentQueue

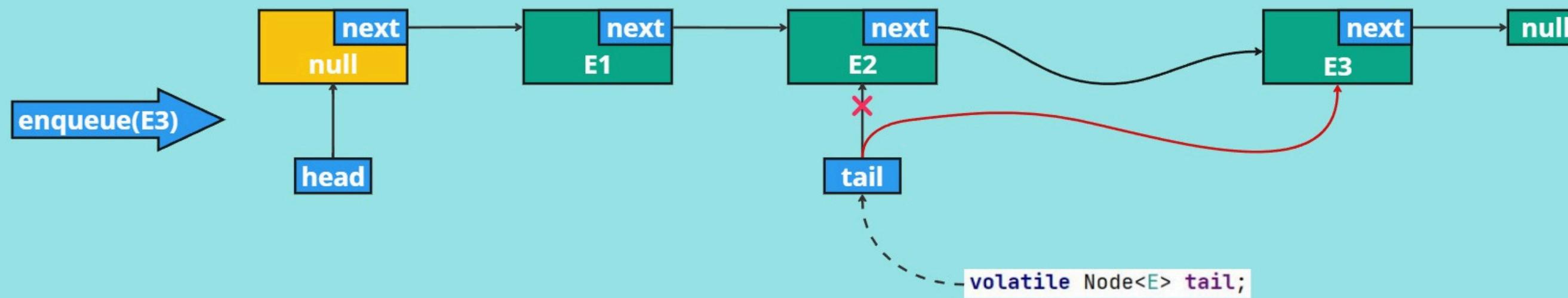


```
public void enqueue(final E element) {  
    final Node<E> newNode = new Node<>(element);  
    //tail.next should be assigned new node  
    //tail should be assigned new node  
}
```



```
public void enqueue(final E element) {  
    final Node<E> newNode = new Node<>(element);  
    tail.next = newNode;  
    //tail should be assigned new node  
}
```

## ConcurrentQueue



```
public void enqueue(final E element) {  
    final Node<E> newNode = new Node<>(element);  
    tail.next = newNode;  
    tail = newNode;  
}
```

## ConcurrentQueue

enqueue(E3) →  
enqueue(E4) →  
enqueue(E5) →



enqueue(E3)

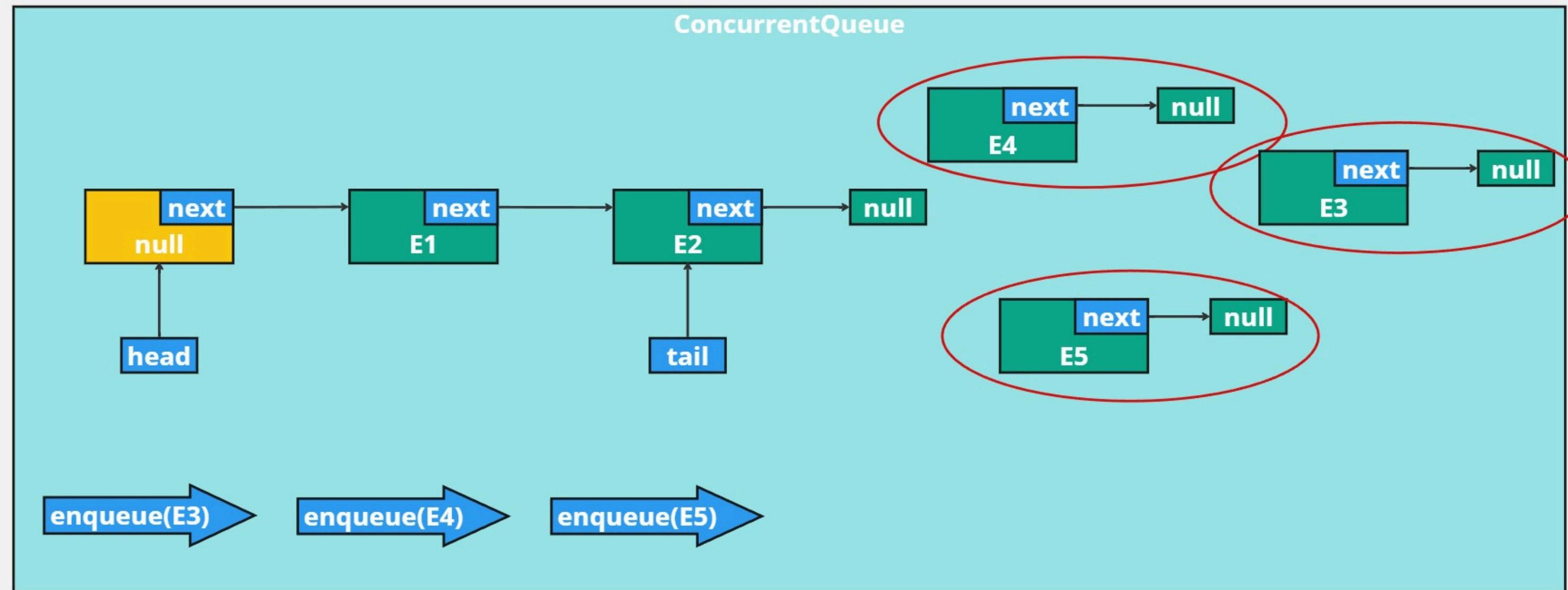
```
public void enqueue(final E element) {  
    final Node<E> newNode = new Node<>(element);  
    tail.next = newNode;  
    tail = newNode;  
}
```

enqueue(E5)

```
public void enqueue(final E element) {  
    final Node<E> newNode = new Node<>(element);  
    tail.next = newNode;  
    tail = newNode;  
}
```

enqueue(E4)

```
public void enqueue(final E element) {  
    final Node<E> newNode = new Node<>(element);  
    tail.next = newNode;  
    tail = newNode;  
}
```



**enqueue(E3)**

```

public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    tail.next = newNode;
    tail = newNode;
}
  
```

**enqueue(E4)**

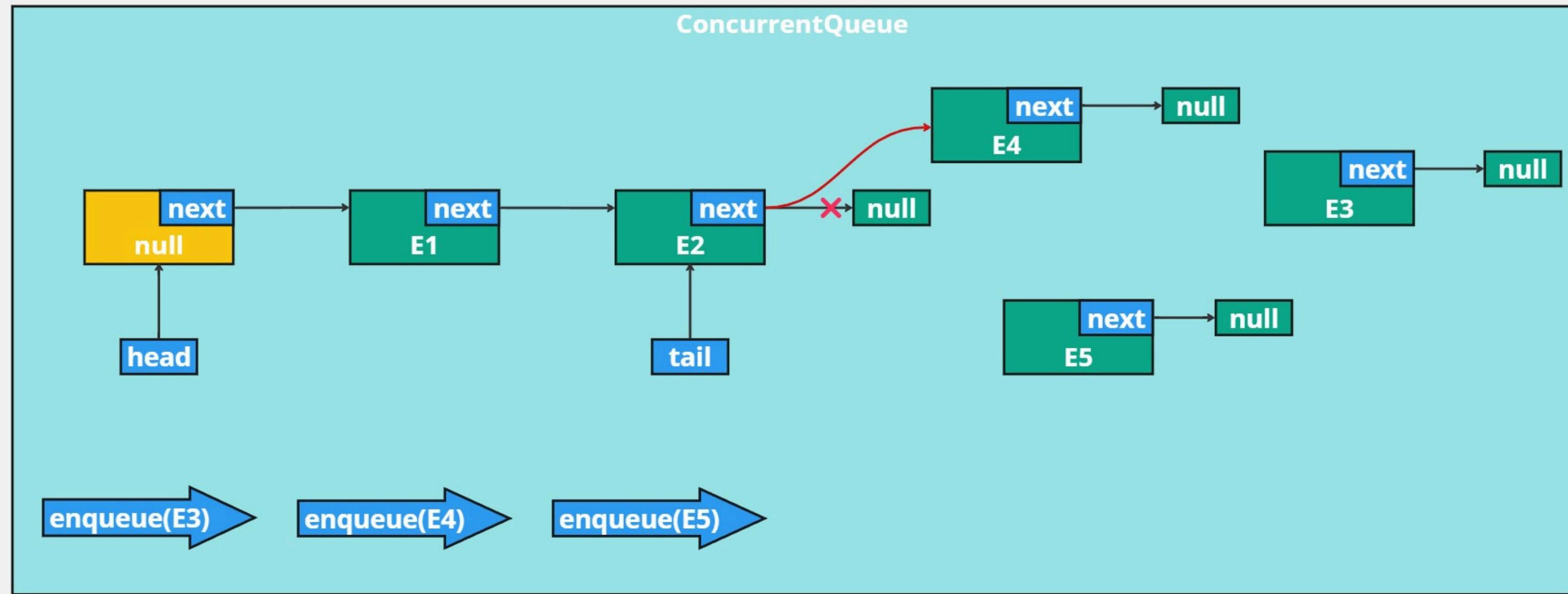
```

public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    tail.next = newNode;
    tail = newNode;
}
  
```

**enqueue(E5)**

```

public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    tail.next = newNode;
    tail = newNode;
}
  
```



**enqueue(E3)**

```

public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    tail.next = newNode;
    tail = newNode;
}
  
```

**enqueue(E4)**

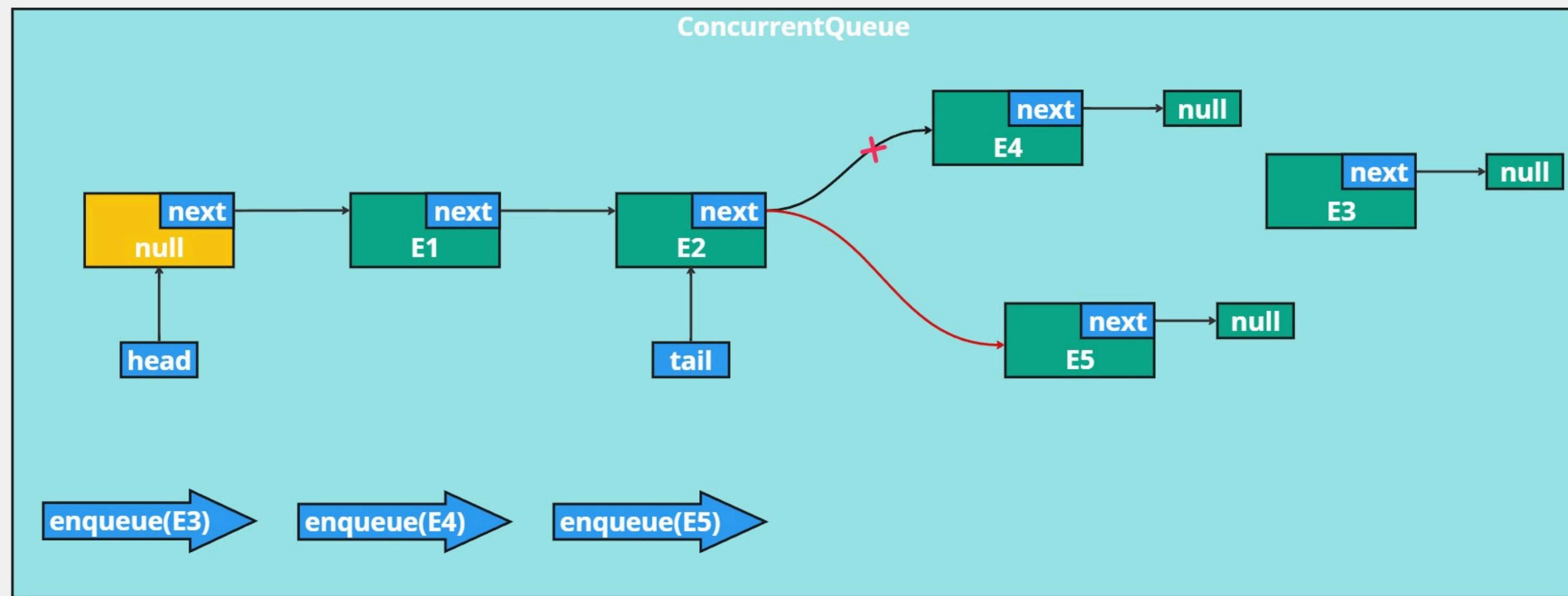
```

public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    tail.next = newNode;
    tail = newNode;
}
  
```

**enqueue(E5)**

```

public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    tail.next = newNode;
    tail = newNode;
}
  
```



**enqueue(E3)**

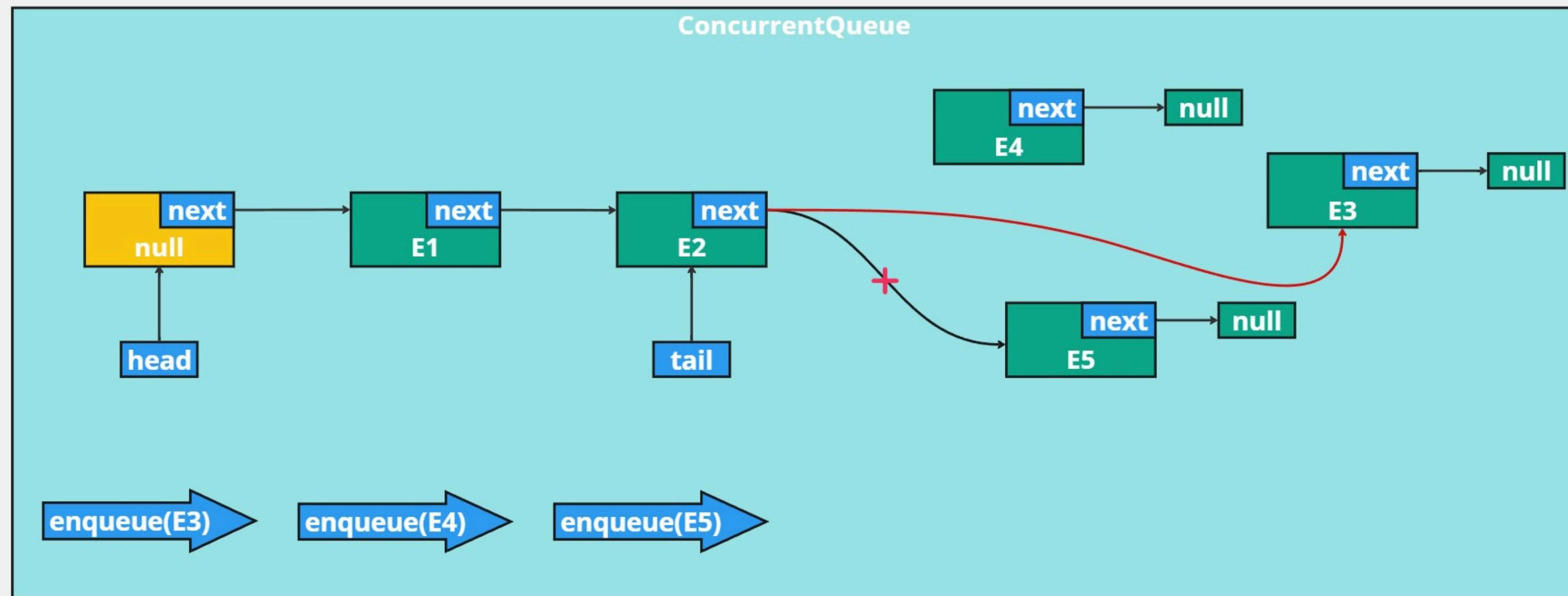
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    tail.next = newNode;
    tail = newNode;
}
```

**enqueue(E)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    tail.next = newNode;
    tail = newNode;
}
```

**enqueue(E4)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    tail.next = newNode;
    tail = newNode;
}
```



**enqueue(E3)**

```

public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    tail.next = newNode;
    tail = newNode;
}

```

**enqueue(E4)**

```

public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    tail.next = newNode;
    |tail = newNode;
}

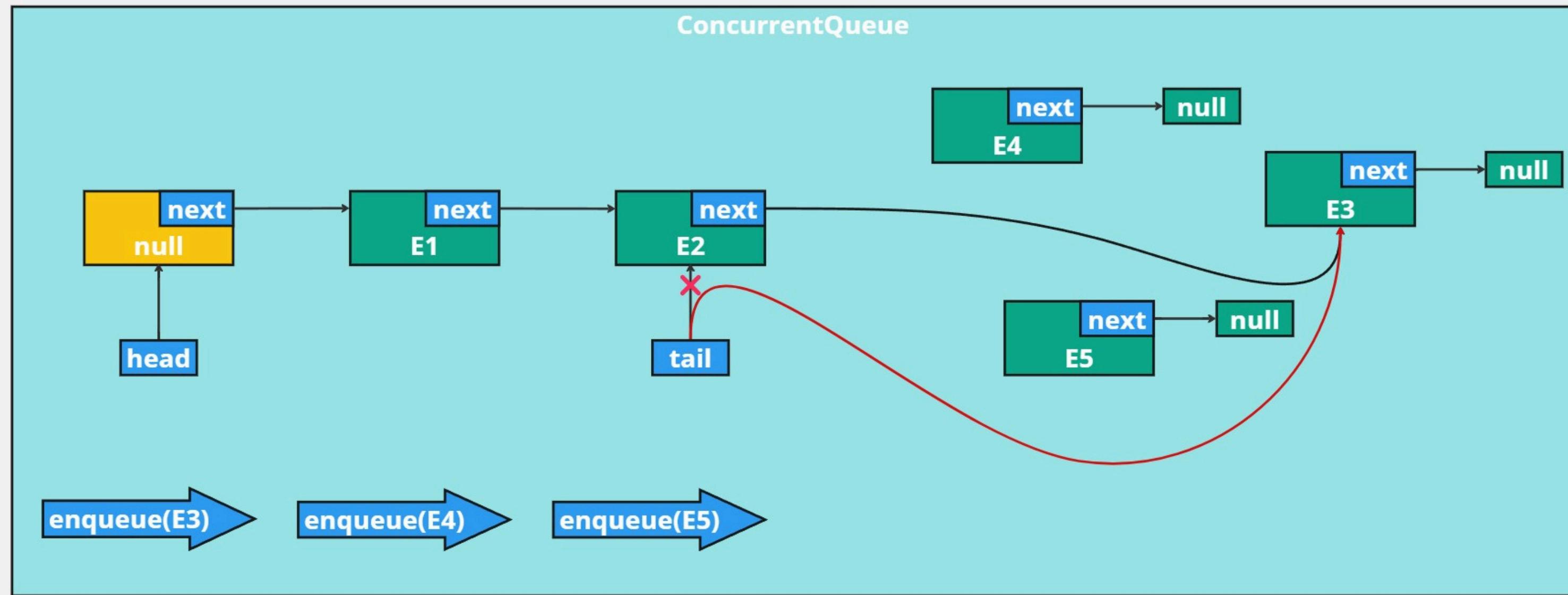
```

**enqueue(E5)**

```

public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    tail.next = newNode;
    |tail = newNode;
}

```



**enqueue(E3)**

```

public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    tail.next = newNode;
tail = newNode;
}
  
```

**enqueue(E5)**

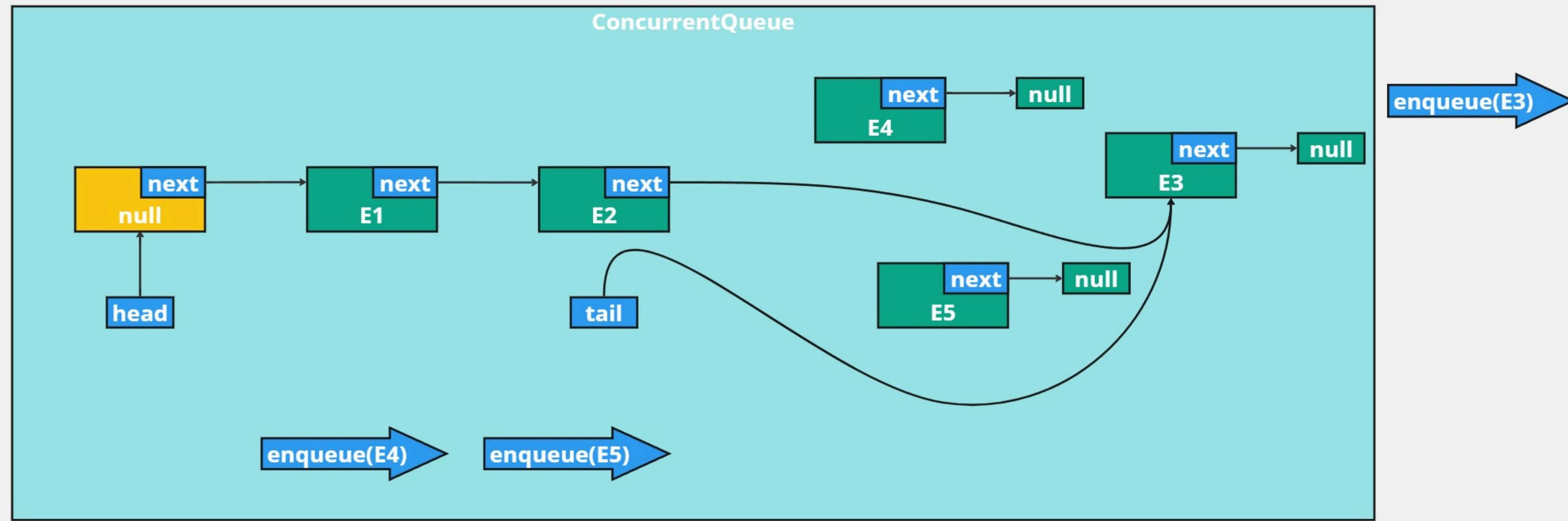
```

public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    tail.next = newNode;
|tail = newNode;
}
  
```

**enqueue(E4)**

```

public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    tail.next = newNode;
|tail = newNode;
}
  
```



**enqueue(E3)**

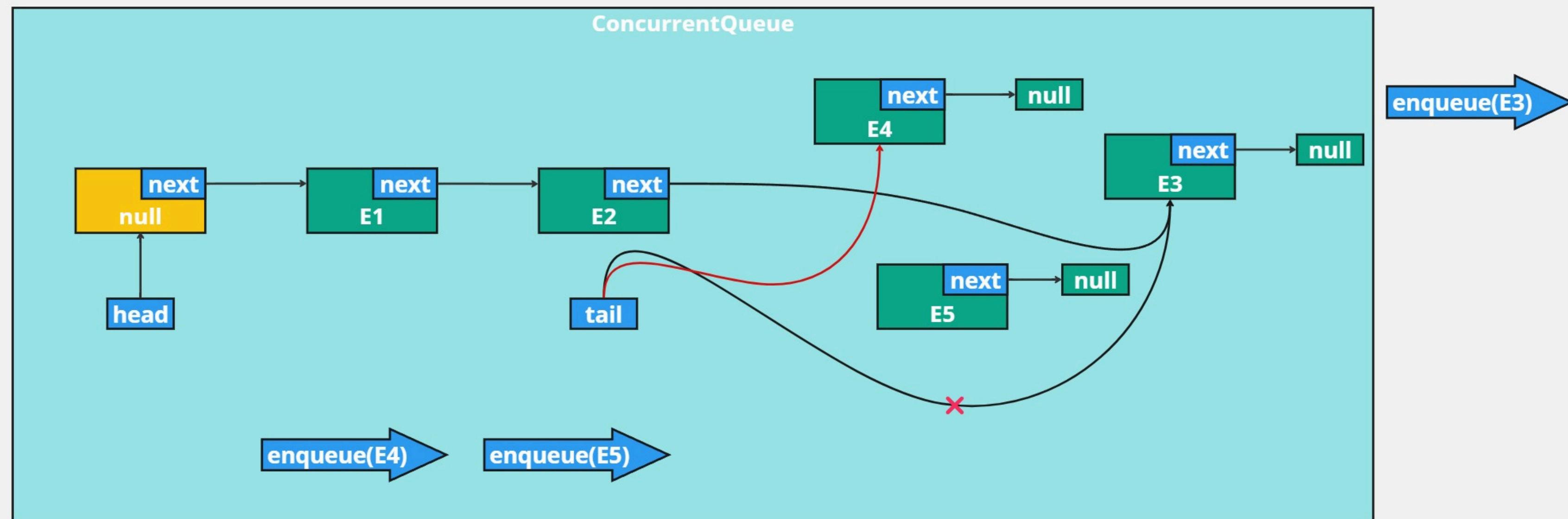
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    tail.next = newNode;
    tail = newNode;
}
```

**enqueue(E4)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    tail.next = newNode;
|tail = newNode;
```

**enqueue(E5)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    tail.next = newNode;
|tail = newNode;
```



**enqueue(E3)**

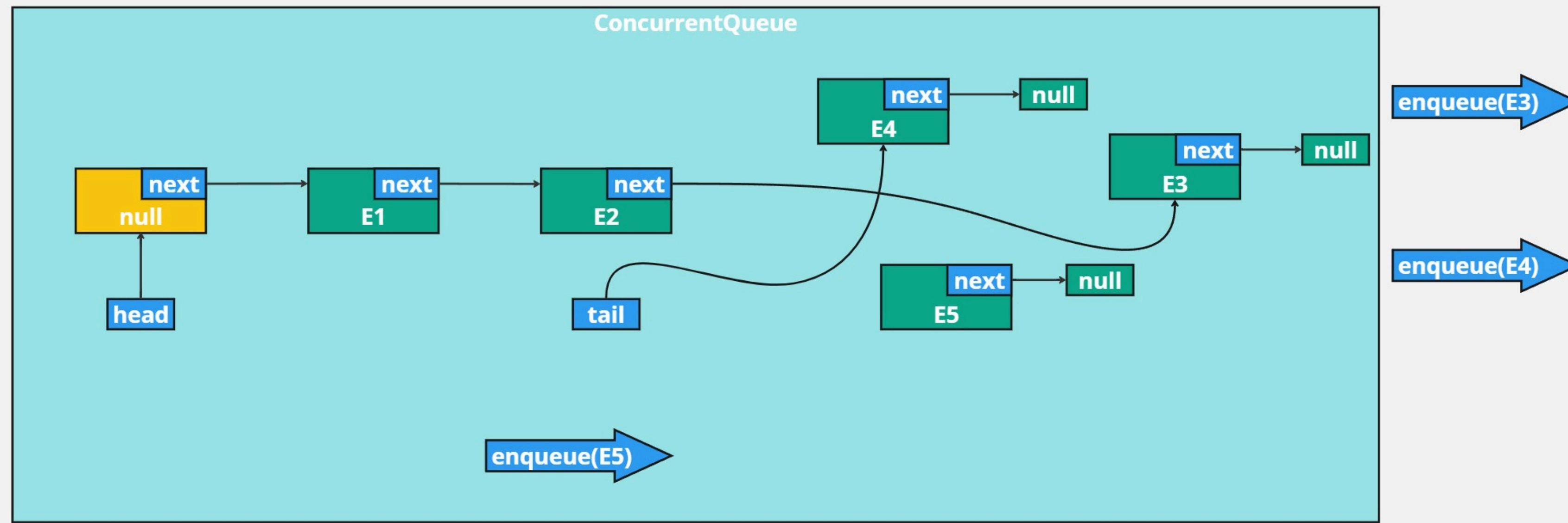
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    tail.next = newNode;
    tail = newNode;
}
```

**enqueue(E4)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    tail.next = newNode;
    tail = newNode;
}
```

**enqueue(E5)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    tail.next = newNode;
    tail = newNode;
}
```



**enqueue(E3)**

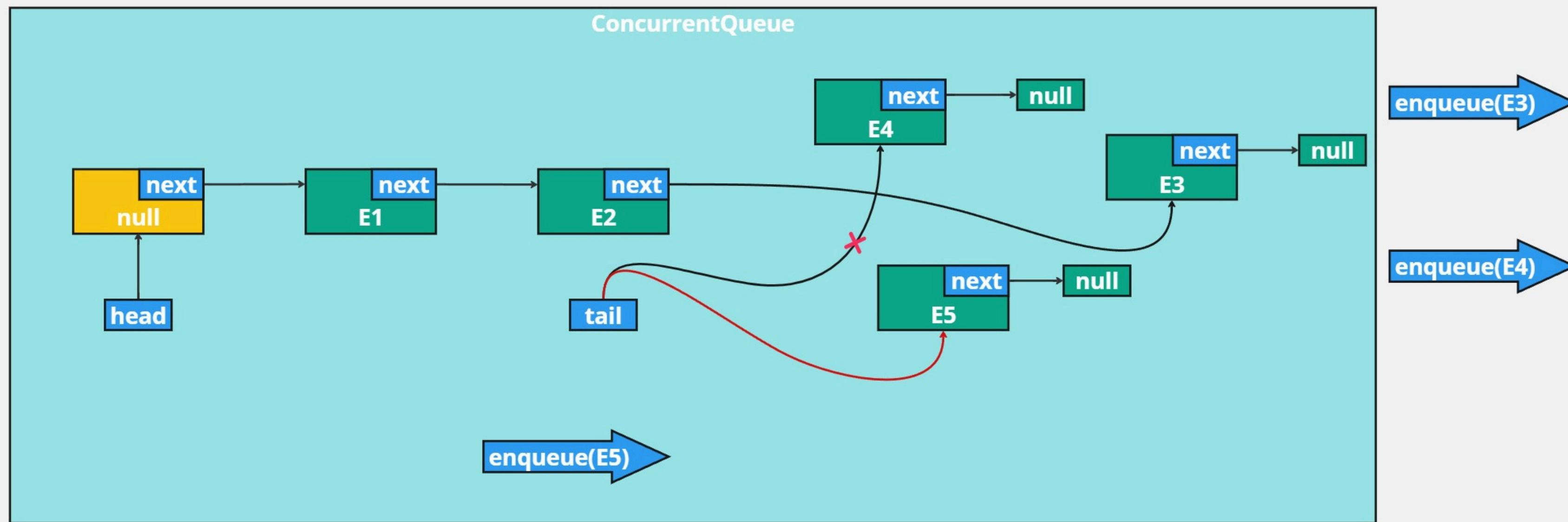
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    tail.next = newNode;
    tail = newNode;
}
```

**enqueue(E4)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    tail.next = newNode;
    tail = newNode;
}
```

**enqueue(E5)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    tail.next = newNode;
    tail = newNode;
}
```



**enqueue(E3)**

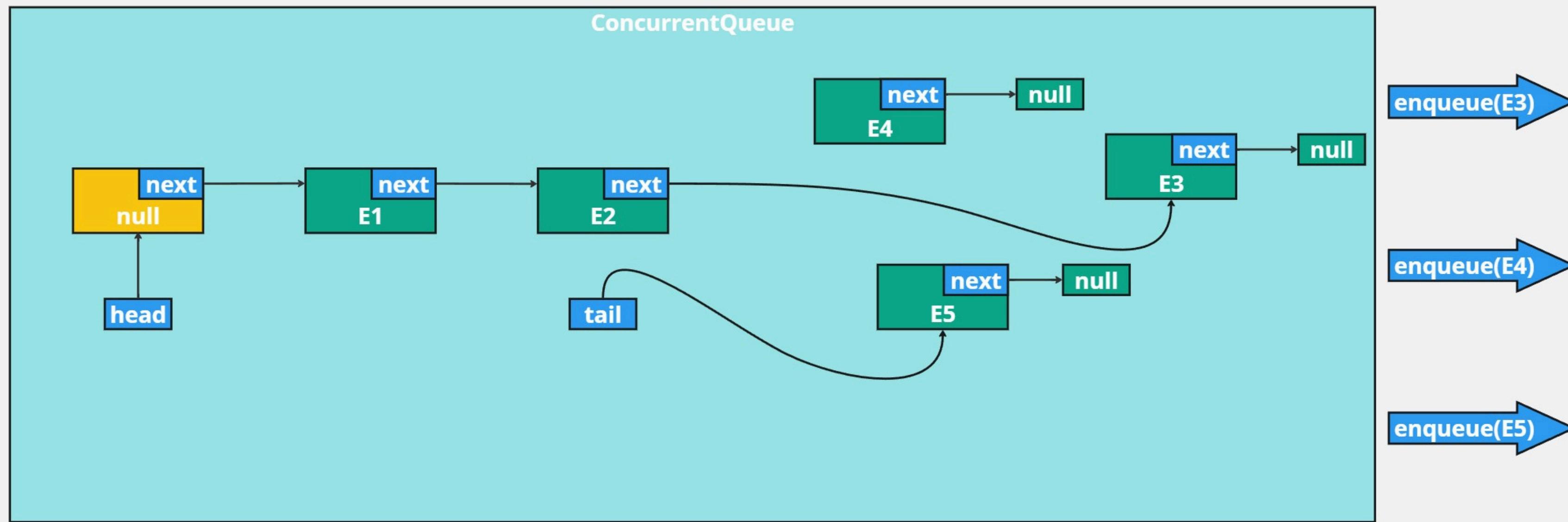
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    tail.next = newNode;
    tail = newNode;
}
```

**enqueue(E4)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    tail.next = newNode;
    tail = newNode;
}
```

**enqueue(E5)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    tail.next = newNode;
    tail = newNode;
}
```



**enqueue(E3)**

```

public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    tail.next = newNode;
    tail = newNode;
}
  
```

**enqueue(E4)**

```

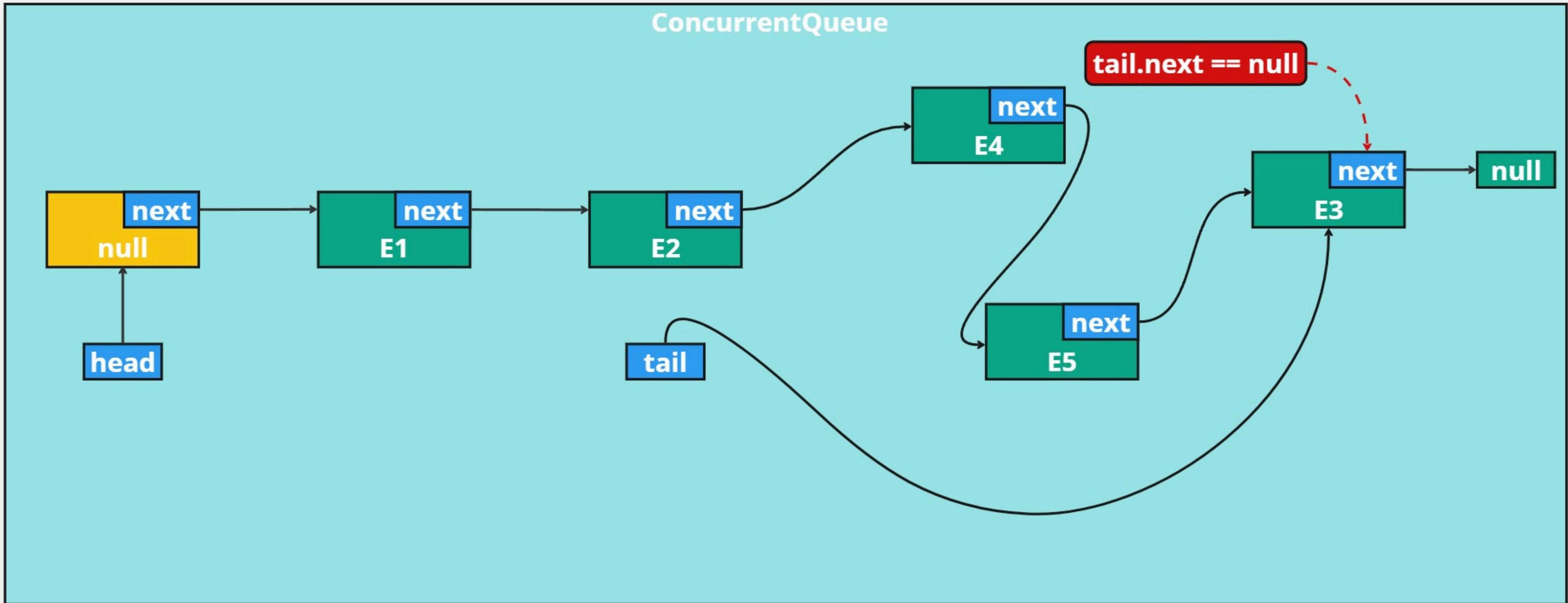
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    tail.next = newNode;
    tail = newNode;
}
  
```

**enqueue(E5)**

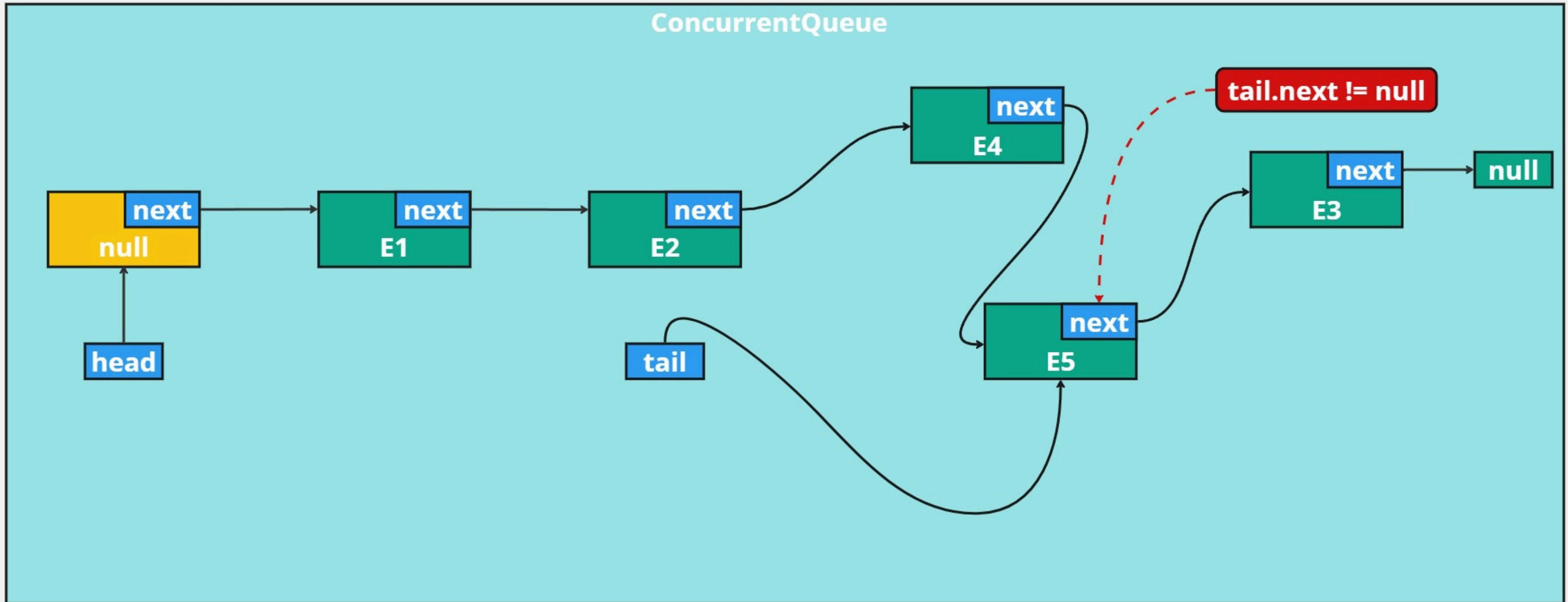
```

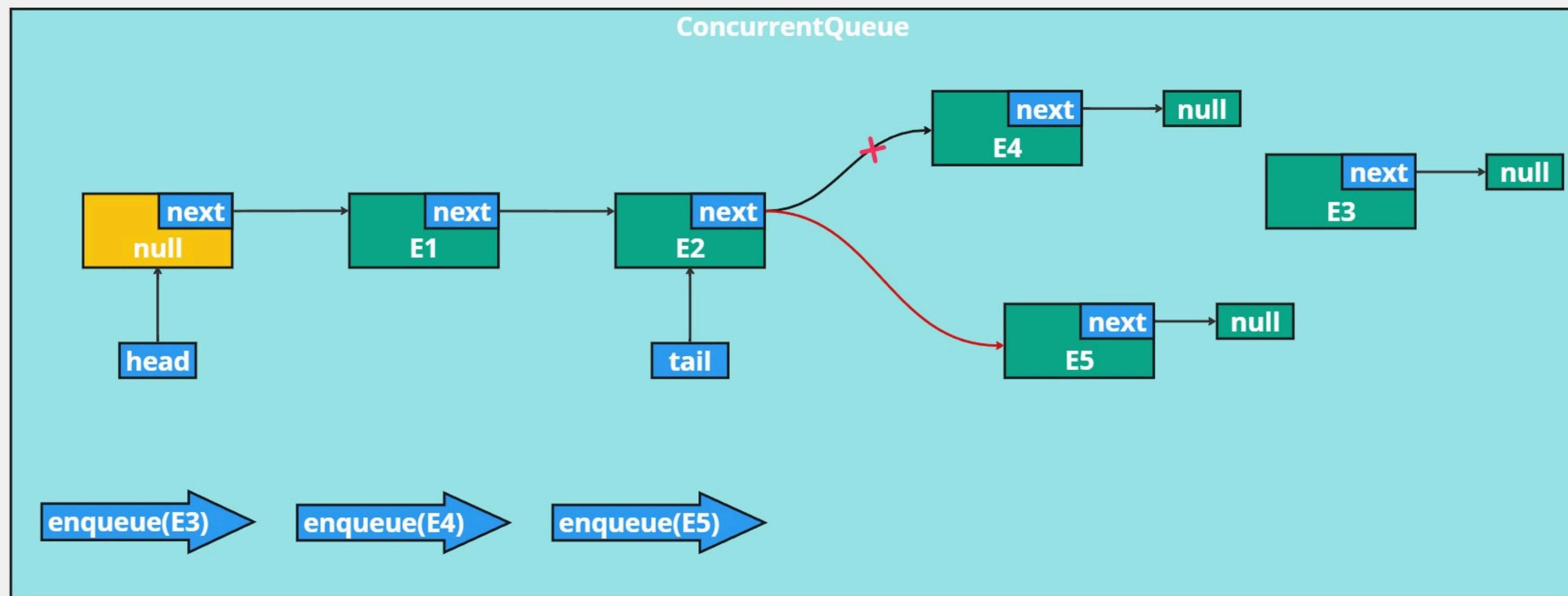
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    tail.next = newNode;
    tail = newNode;
}
  
```

## *Consistency state*



## *Not consistency state*





**enqueue(E3)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    tail.next = newNode;
    tail = newNode;
}
```

**enqueue(E4)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    tail.next = newNode;
    tail = newNode;
}
```

**enqueue(E5)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    tail.next = newNode;
    tail = newNode;
}
```

→ **tail.next != null**

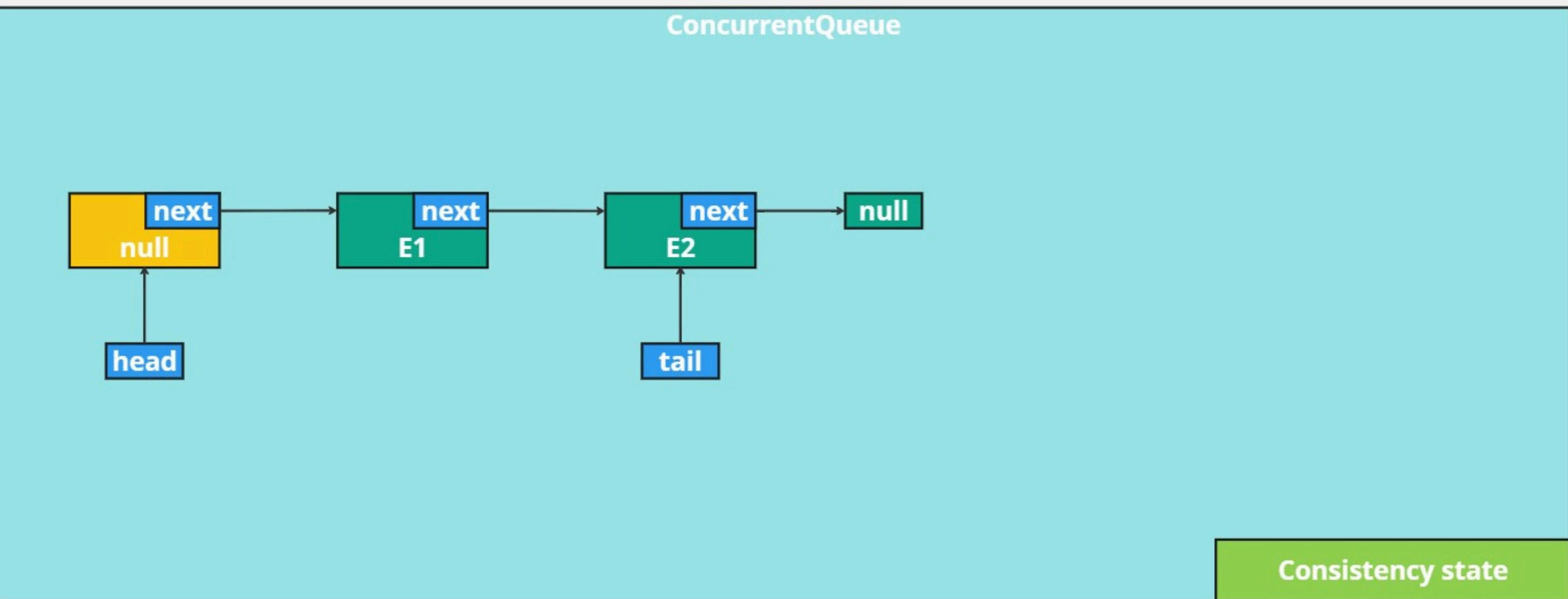
```
public void enqueue(final E element) {  
    final Node<E> newNode = new Node<>(element);  
    tail.next = newNode;  
    tail = newNode;  
}
```



```
public void enqueue(final E element) {  
    final Node<E> newNode = new Node<>(element);  
    while (true) {  
        if (tail.next == null) {  
            tail.next = newNode;  
            break;  
        }  
    }  
    tail = newNode;  
}
```

## ConcurrentQueue

enqueue(E3) →  
enqueue(E4) →  
enqueue(E5) →



### enqueue(E3)

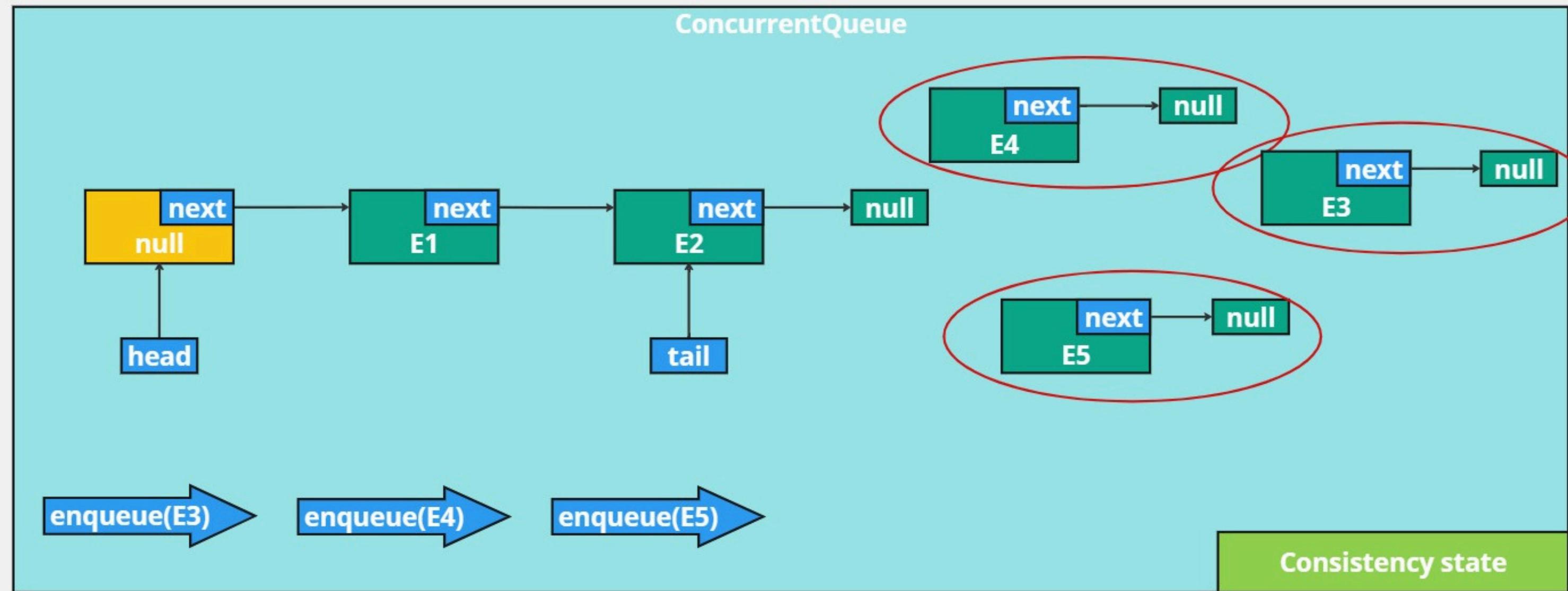
```
public void enqueue(final E element) {  
    final Node<E> newNode = new Node<>(element);  
    while (true) {  
        if (tail.next == null) {  
            tail.next = newNode;  
            break;  
        }  
    }  
    tail = newNode;  
}
```

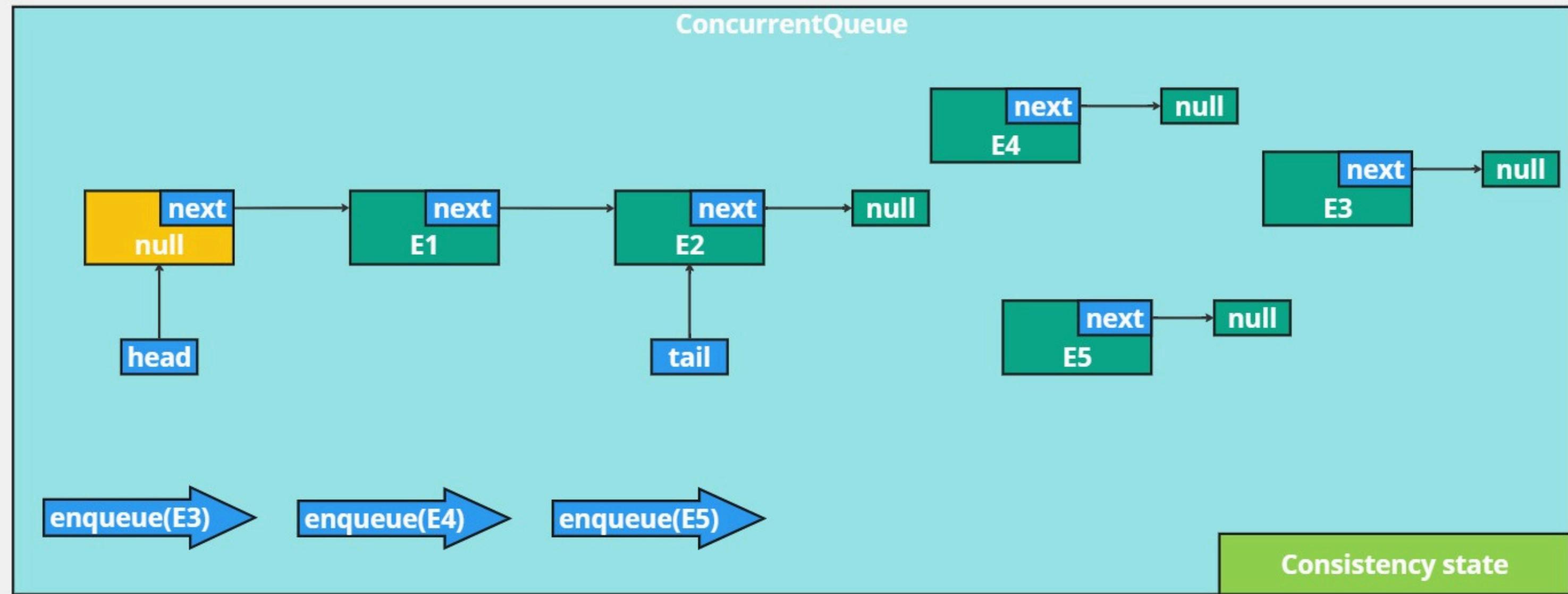
### enqueue(E4)

```
public void enqueue(final E element) {  
    final Node<E> newNode = new Node<>(element);  
    while (true) {  
        if (tail.next == null) {  
            tail.next = newNode;  
            break;  
        }  
    }  
    tail = newNode;  
}
```

### enqueue(E5)

```
public void enqueue(final E element) {  
    final Node<E> newNode = new Node<>(element);  
    while (true) {  
        if (tail.next == null) {  
            tail.next = newNode;  
            break;  
        }  
    }  
    tail = newNode;  
}
```





**enqueue(E3)**

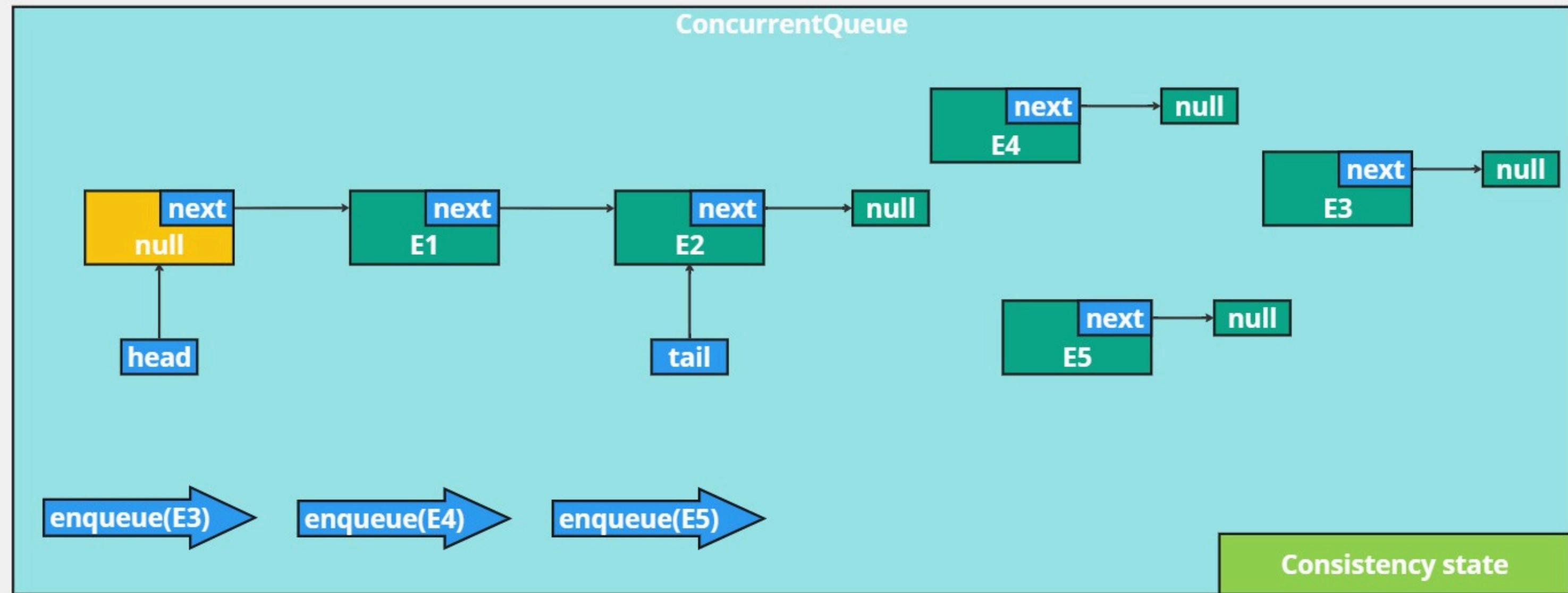
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next == null) {
            tail.next = newNode;
            break;
        }
    }
    tail = newNode;
}
```

**enqueue(E4)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next == null) {
            tail.next = newNode;
            break;
        }
    }
    tail = newNode;
}
```

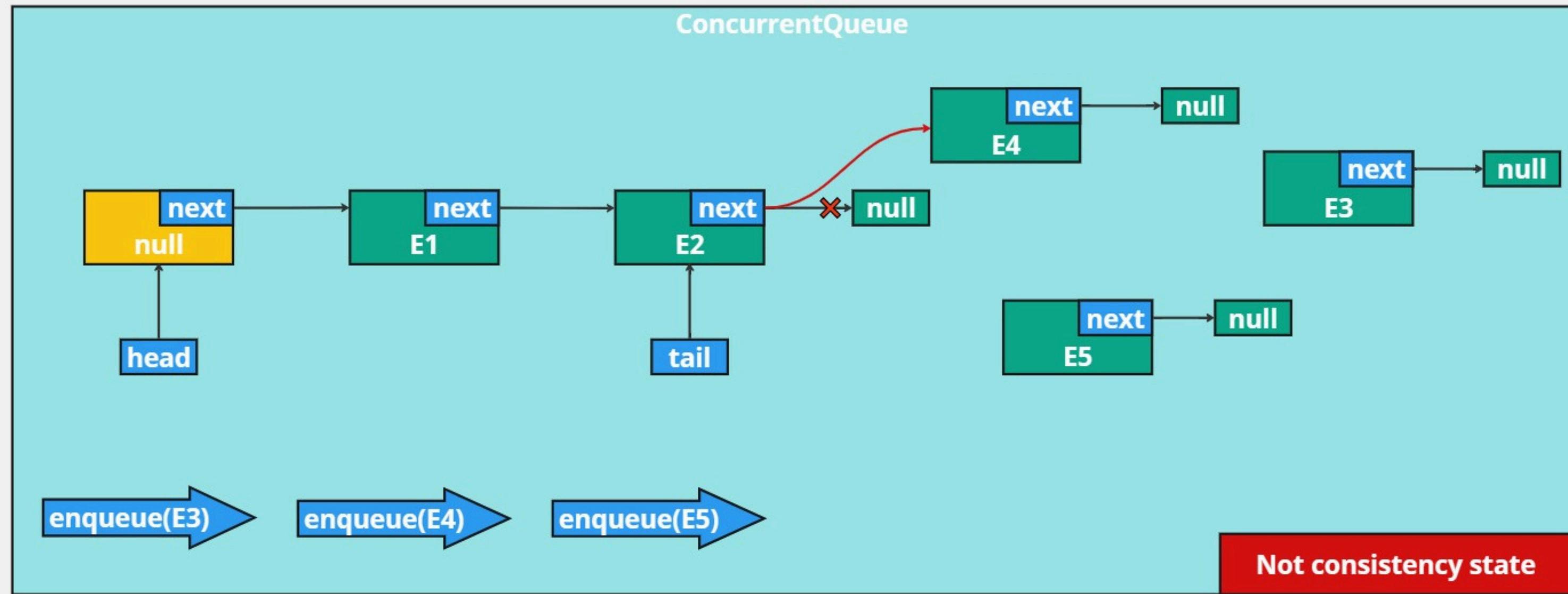
**enqueue(E5)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next == null) {
            tail.next = newNode;
            break;
        }
    }
    tail = newNode;
}
```



Consistency state

enqueue(E3)    enqueue(E4)    enqueue(E5)



**enqueue(E3)**

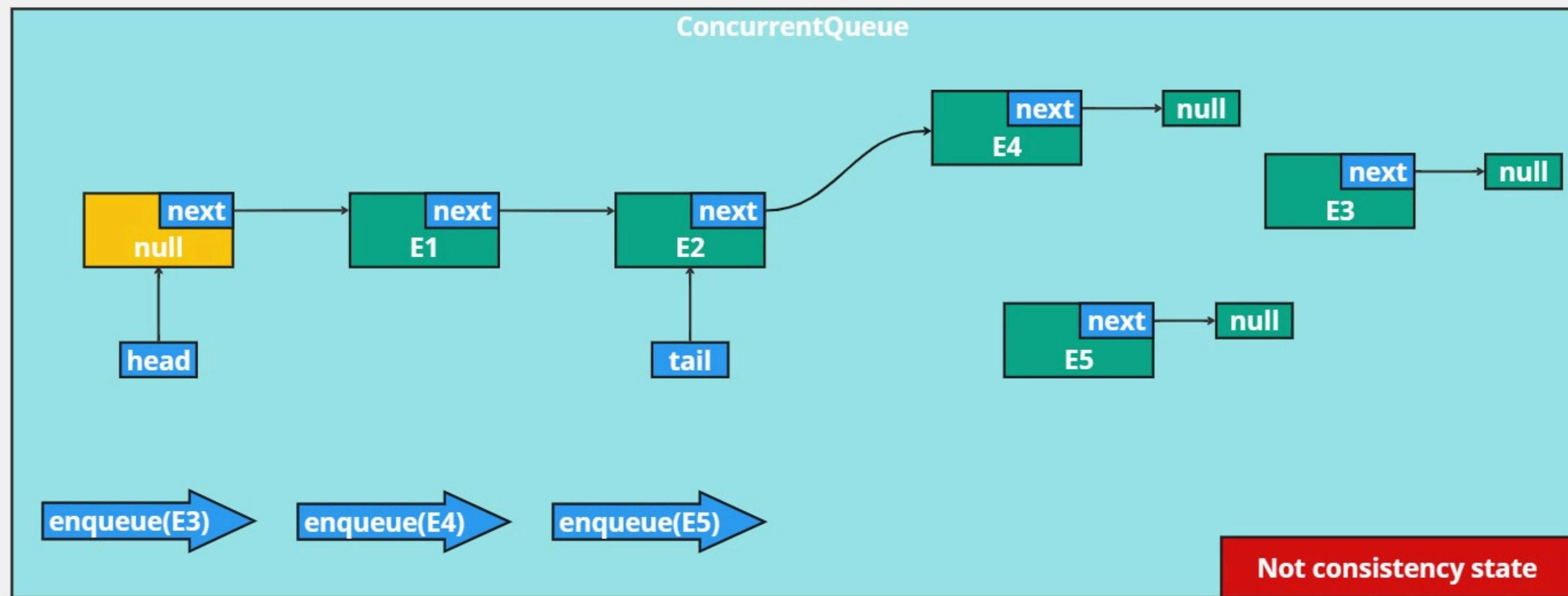
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next == null) {
            tail.next = newNode;
            break;
        }
    }
    tail = newNode;
}
```

**enqueue(E4)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next == null) {
            tail.next = newNode;
            break;
        }
    }
    tail = newNode;
}
```

**enqueue(E5)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next == null) {
            tail.next = newNode;
            break;
        }
    }
    tail = newNode;
}
```



**enqueue(E3)**

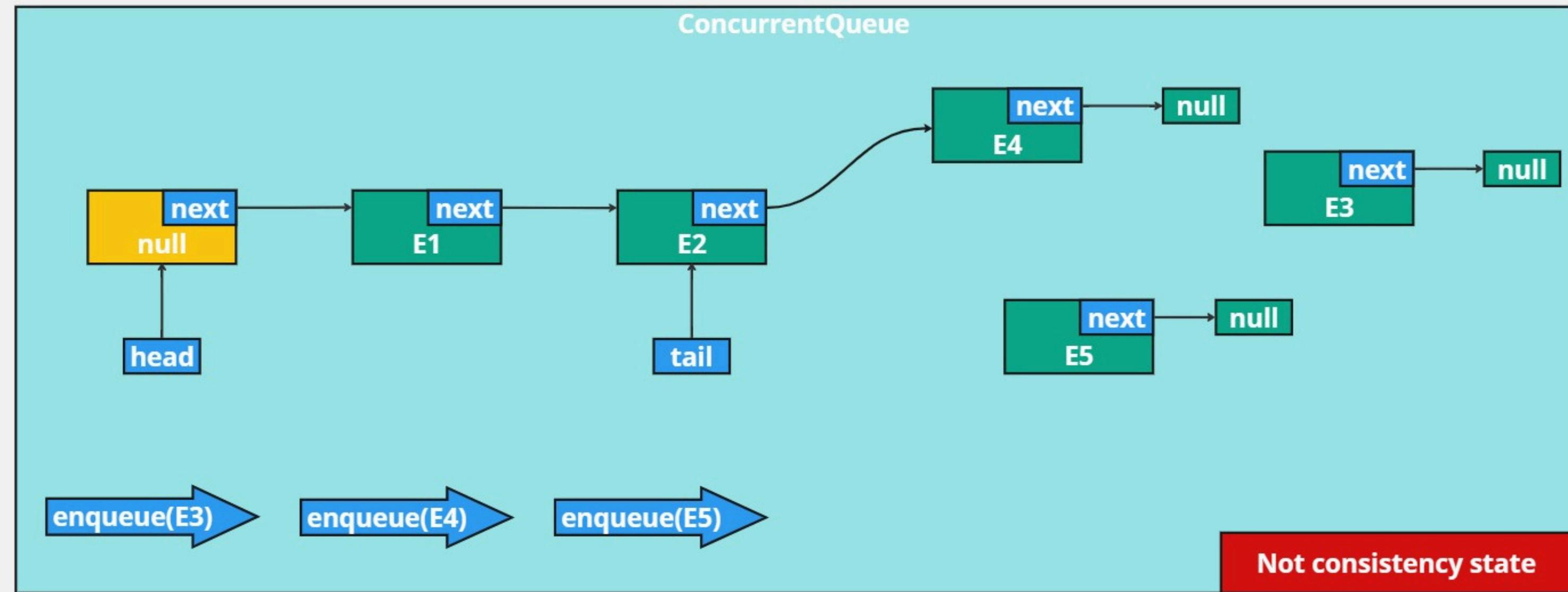
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next == null) {
            tail.next = newNode;
            break;
        }
    }
    tail = newNode;
}
```

**enqueue(E4)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next == null) {
            tail.next = newNode;
            break;
        }
    }
    tail = newNode;
}
```

**enqueue(E5)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next == null) {
            tail.next = newNode;
            break;
        }
    }
    tail = newNode;
}
```



```

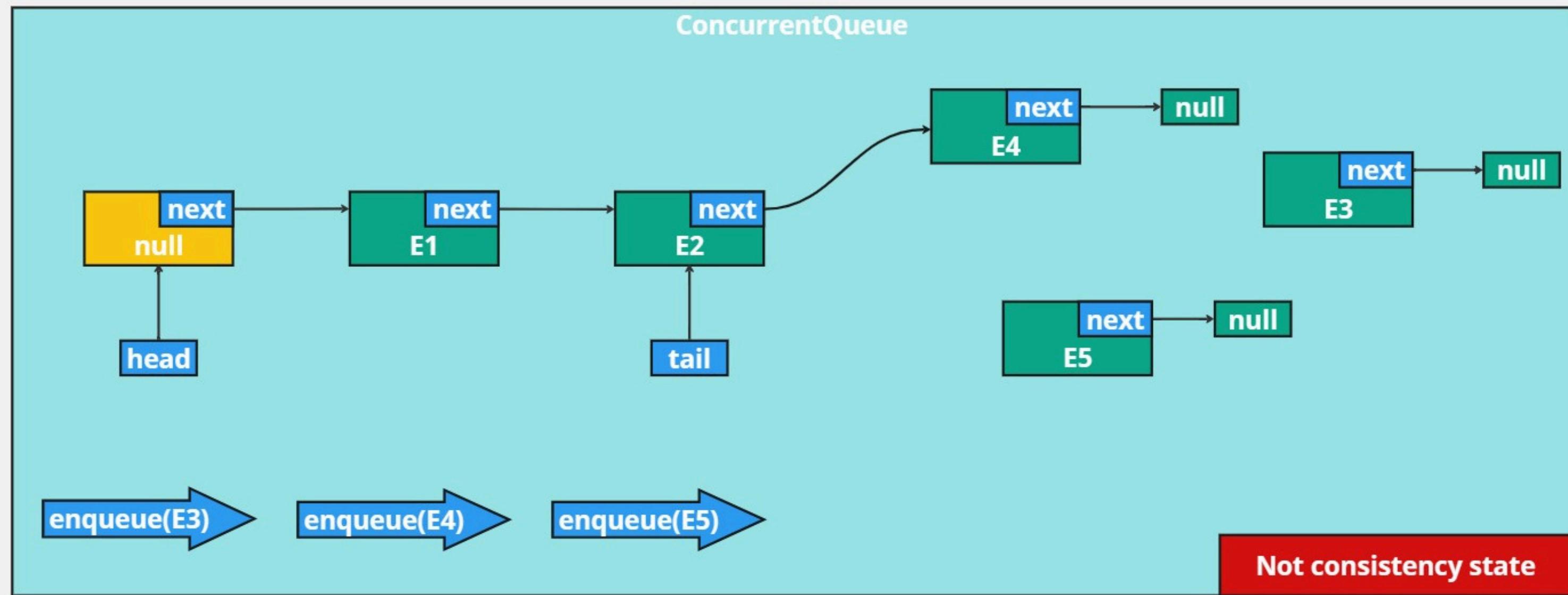
enqueue(E3)
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next == null) {
            tail.next = newNode;
            break;
        }
    }
    tail = newNode;
}
  
```

```

enqueue(E4)
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next == null) {
            tail.next = newNode;
            break;
        }
    }
    tail = newNode;
}
  
```

```

enqueue(E5)
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next == null) {
            tail.next = newNode;
            break;
        }
    }
    tail = newNode;
}
  
```



**enqueue(E3)**

```

public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next == null) {
            tail.next = newNode;
            break;
        }
    }
    tail = newNode;
}

```

**enqueue(E4)**

```

public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next == null) {
            tail.next = newNode;
            break;
        }
    }
    tail = newNode;
}

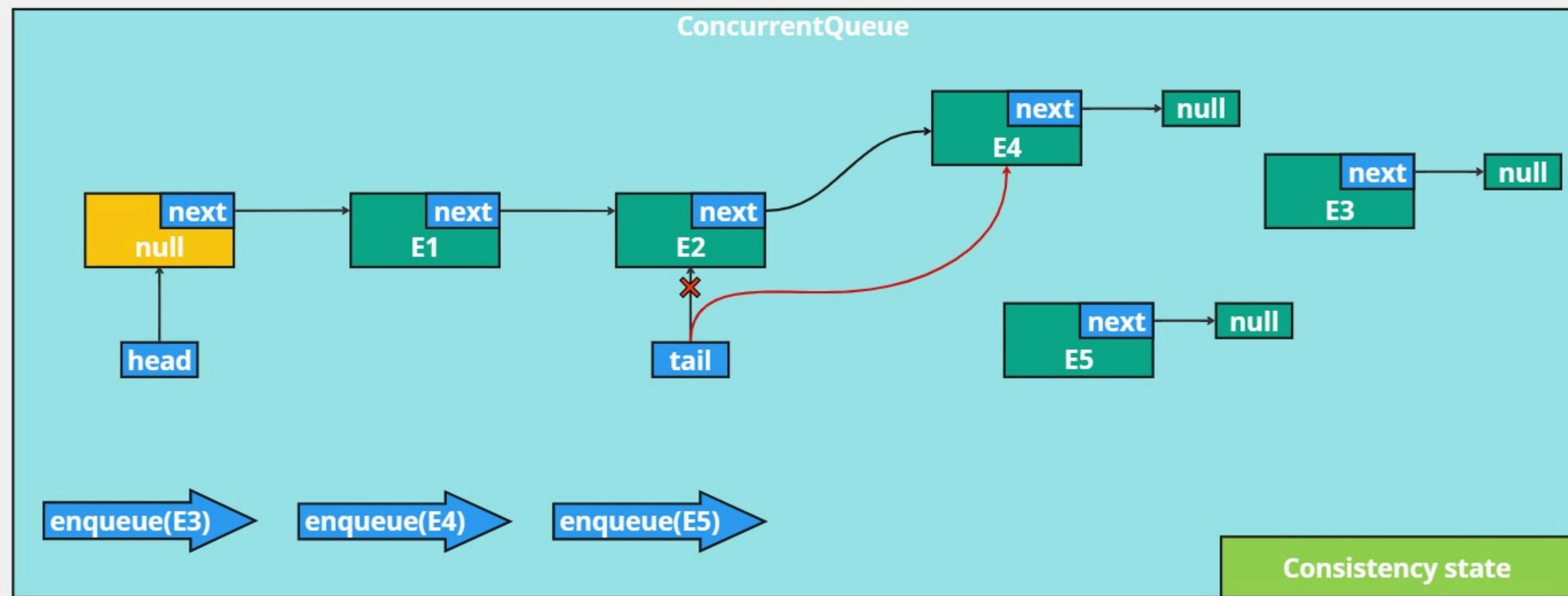
```

**enqueue(E5)**

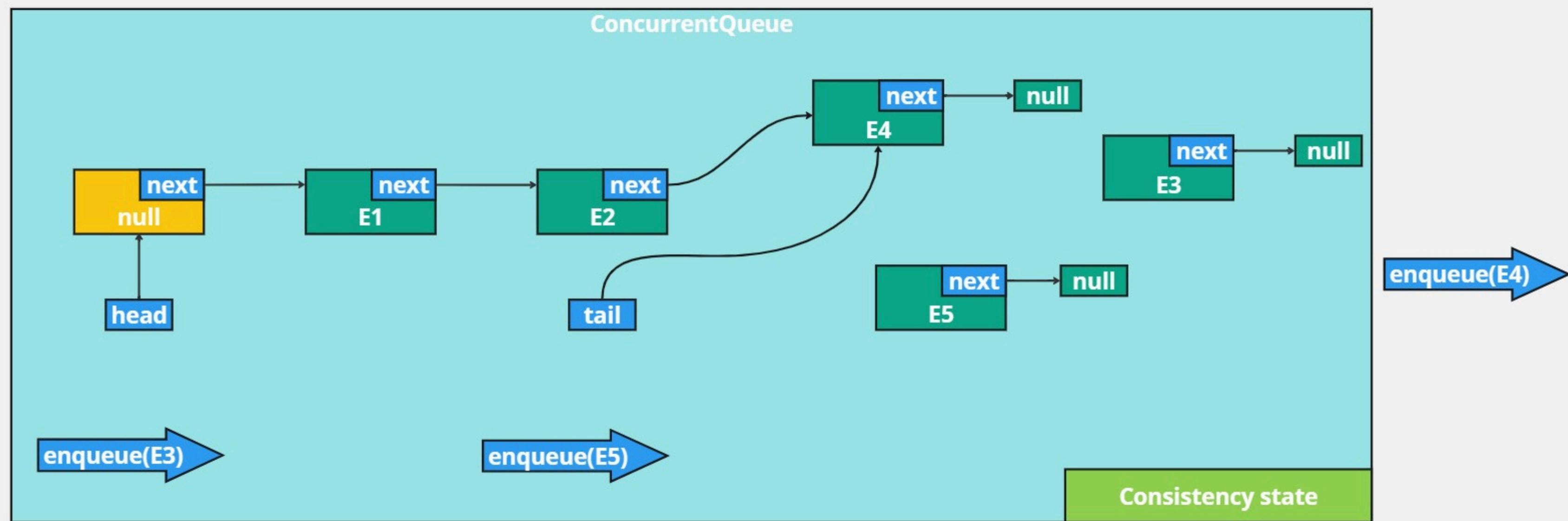
```

public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next == null) {
            tail.next = newNode;
            break;
        }
    }
    tail = newNode;
}

```



Consistency state



**enqueue(E3)**

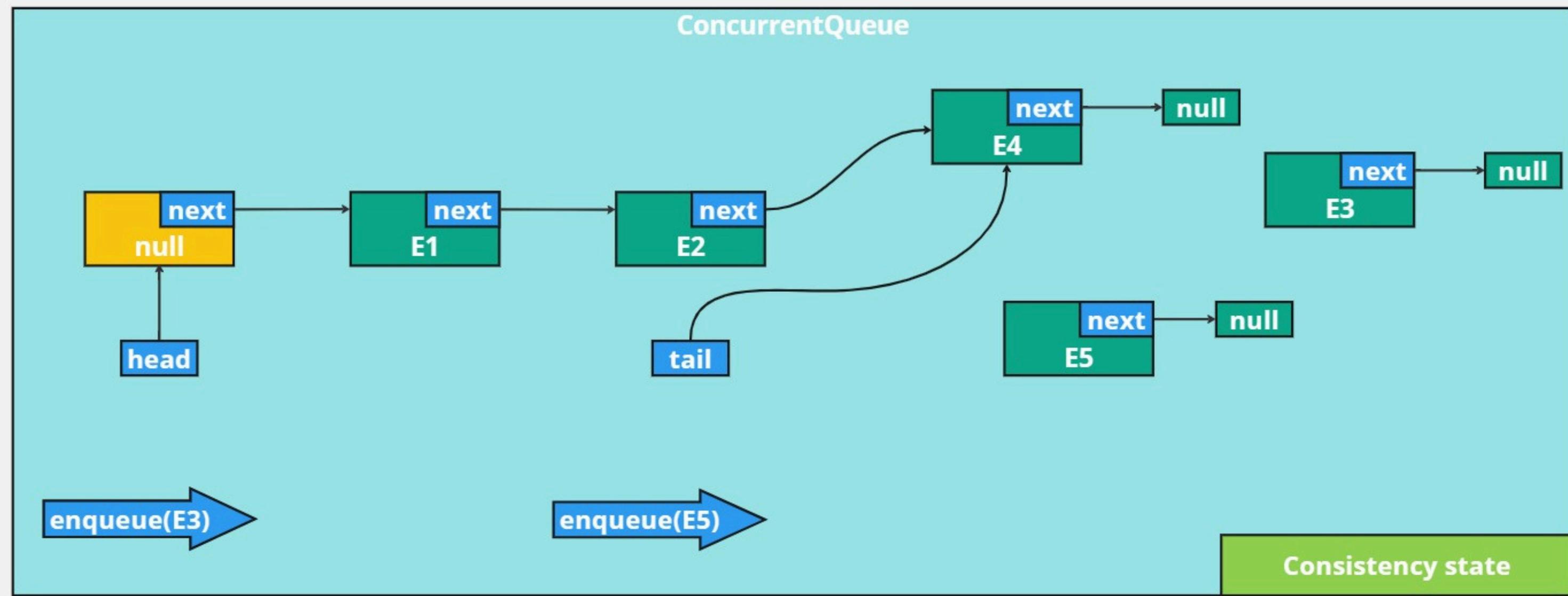
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next == null) {
            tail.next = newNode;
            break;
        }
    }
    tail = newNode;
}
```

**enqueue(E4)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next == null) {
            tail.next = newNode;
            break;
        }
    }
    tail = newNode;
}
```

**enqueue(E5)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next == null) {
            tail.next = newNode;
            break;
        }
    }
    tail = newNode;
}
```



**enqueue(E3)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next == null) {
            tail.next = newNode;
            break;
        }
    }
    tail = newNode;
}
```

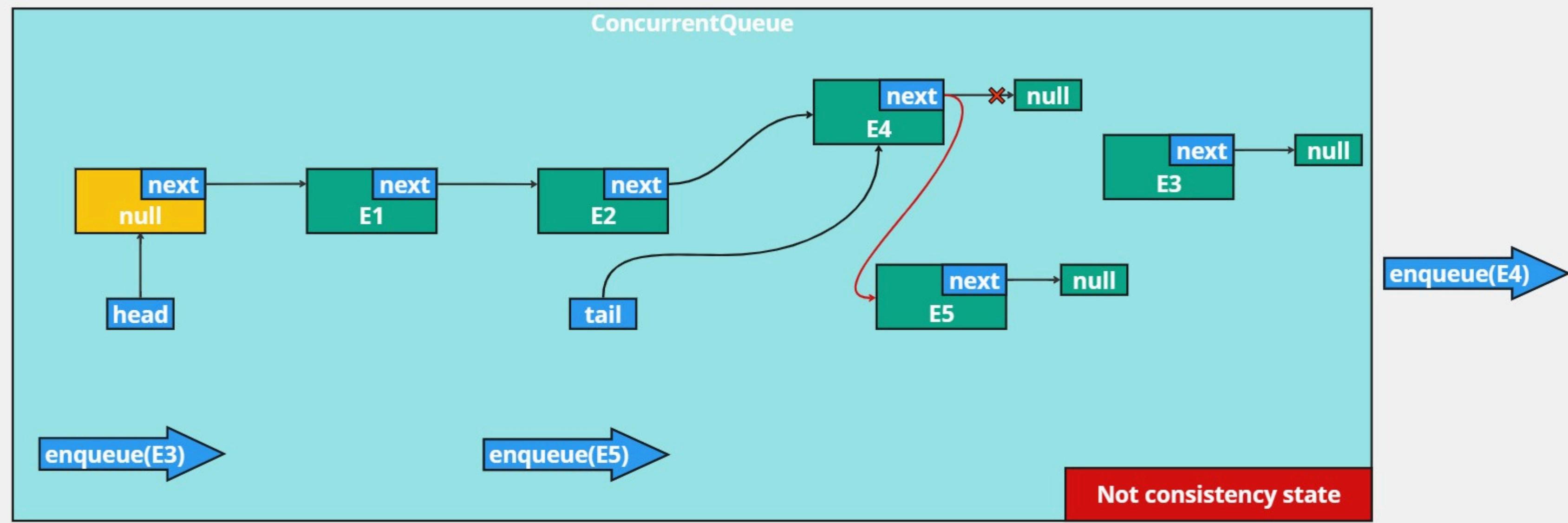
**enqueue(E4)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next == null) {
            tail.next = newNode;
            break;
        }
    }
    tail = newNode;
}
```

**enqueue(E5)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next == null) {
            tail.next = newNode;
            break;
        }
    }
    tail = newNode;
}
```

Consistency state



**enqueue(E3)**

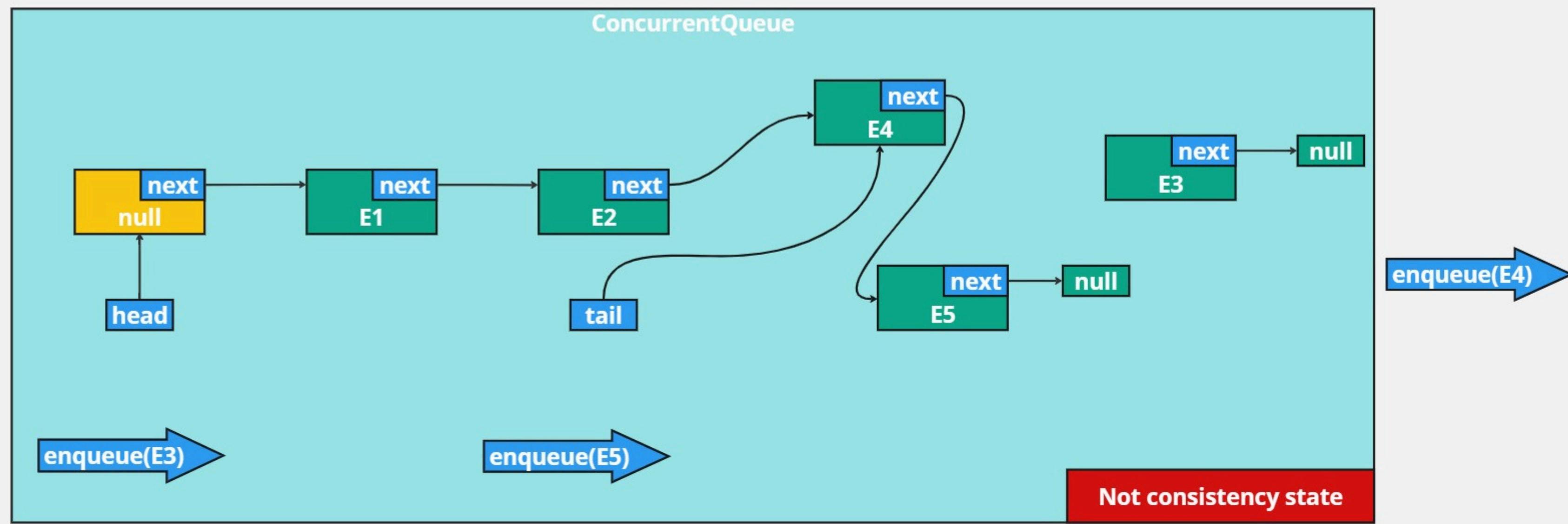
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next == null) {
            tail.next = newNode;
            break;
        }
    }
    tail = newNode;
}
```

**enqueue(E4)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next == null) {
            tail.next = newNode;
            break;
        }
    }
    tail = newNode;
}
```

**enqueue(E5)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next == null) {
            tail.next = newNode;
            break;
        }
    }
    tail = newNode;
}
```



**enqueue(E3)**

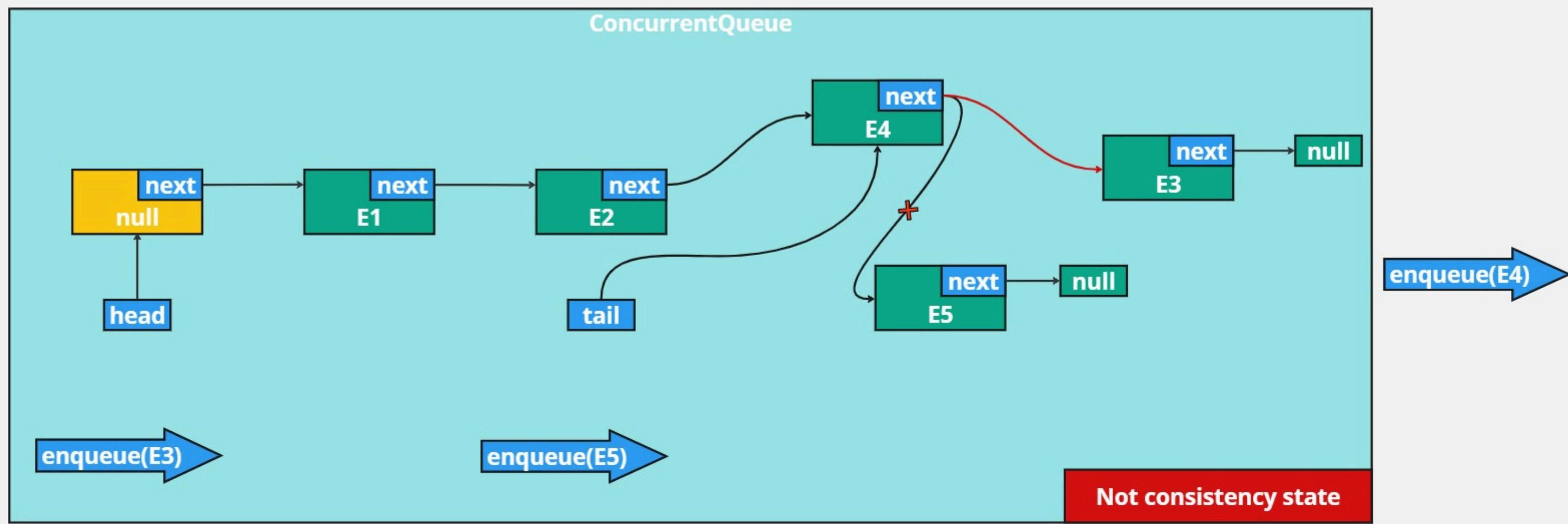
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next == null) {
            tail.next = newNode;
            break;
        }
    }
    tail = newNode;
}
```

**enqueue(E4)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next == null) {
            tail.next = newNode;
            break;
        }
    }
    tail = newNode;
}
```

**enqueue(E5)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next == null) {
            tail.next = newNode;
            break;
        }
    }
    tail = newNode;
}
```



**enqueue(E3)**

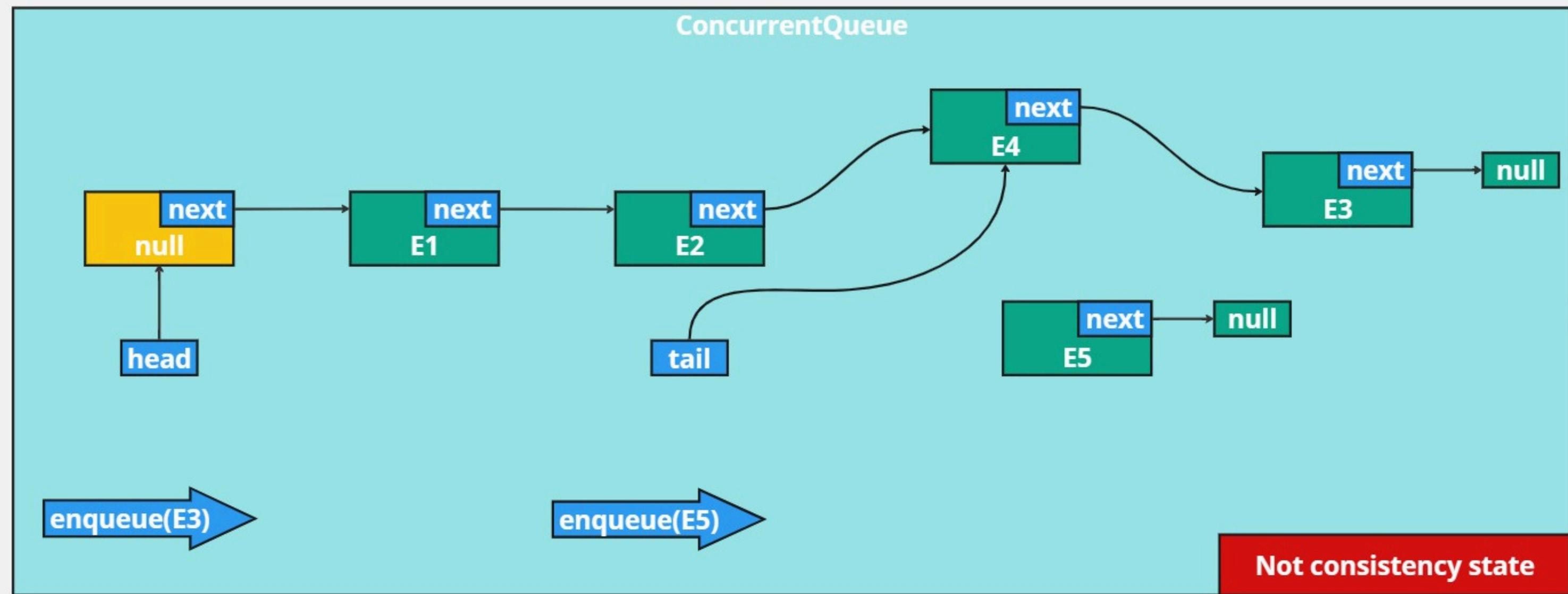
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next == null) {
            tail.next = newNode;
            break;
        }
    }
    tail = newNode;
}
```

**enqueue(E4)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next == null) {
            tail.next = newNode;
            break;
        }
    }
    tail = newNode;
}
```

**enqueue(E5)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next == null) {
            tail.next = newNode;
            break;
        }
    }
    tail = newNode;
}
```



**enqueue(E3)**

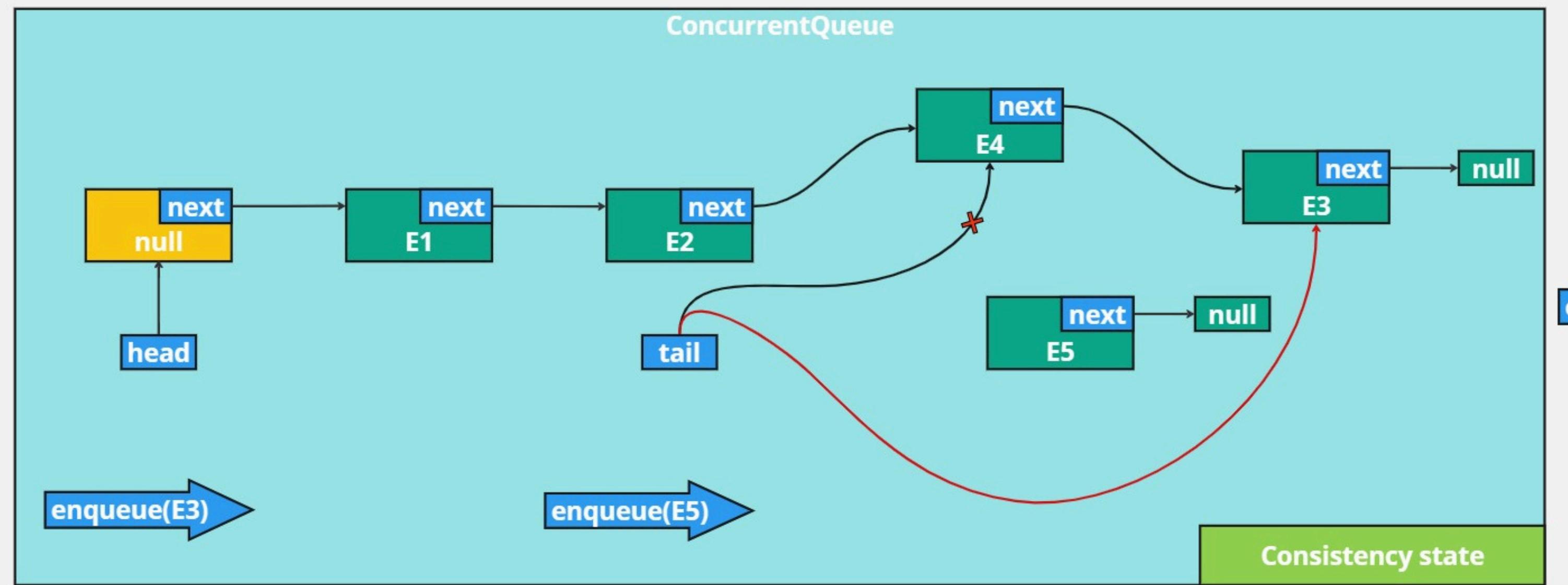
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next == null) {
            tail.next = newNode;
            break;
        }
    }
    tail = newNode;
}
```

**enqueue(E4)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next == null) {
            tail.next = newNode;
            break;
        }
    }
    tail = newNode;
}
```

**enqueue(E5)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next == null) {
            tail.next = newNode;
            break;
        }
    }
    tail = newNode;
}
```



**enqueue(E3)**

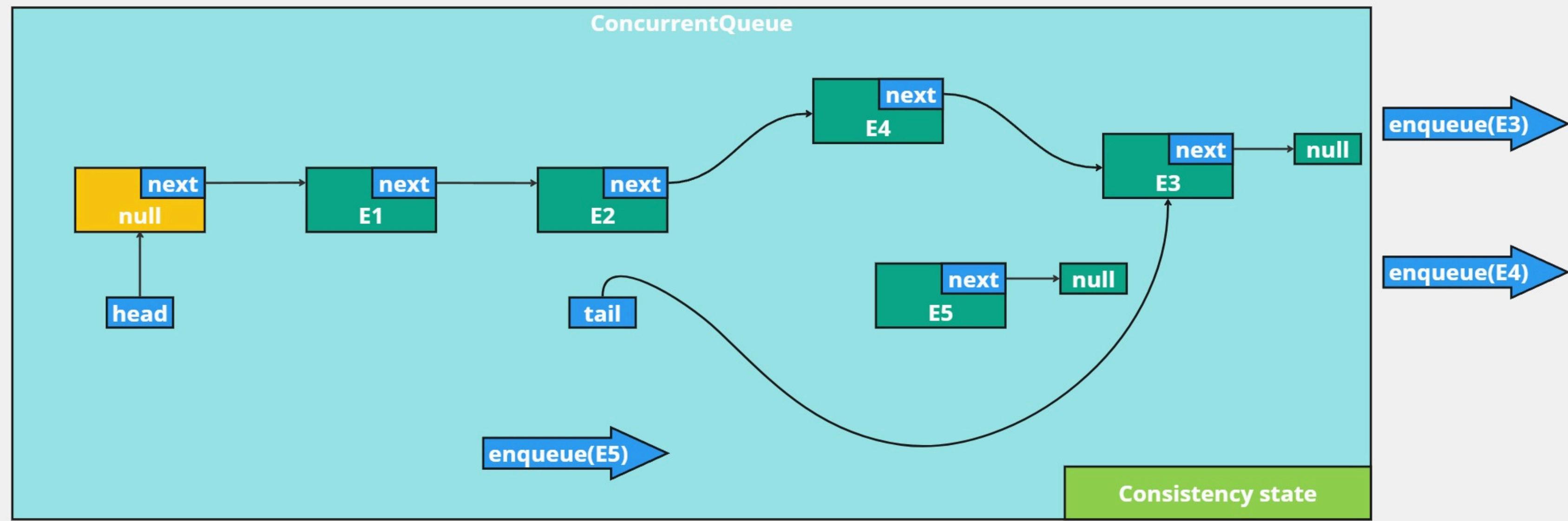
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next == null) {
            tail.next = newNode;
            break;
        }
    }
    tail = newNode;
}
```

**enqueue(E4)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next == null) {
            tail.next = newNode;
            break;
        }
    }
    tail = newNode;
}
```

**enqueue(E5)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next == null) {
            tail.next = newNode;
            break;
        }
    }
    tail = newNode;
}
```



### enqueue(E3)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next == null) {
            tail.next = newNode;
            break;
        }
    }
    tail = newNode;
}
```

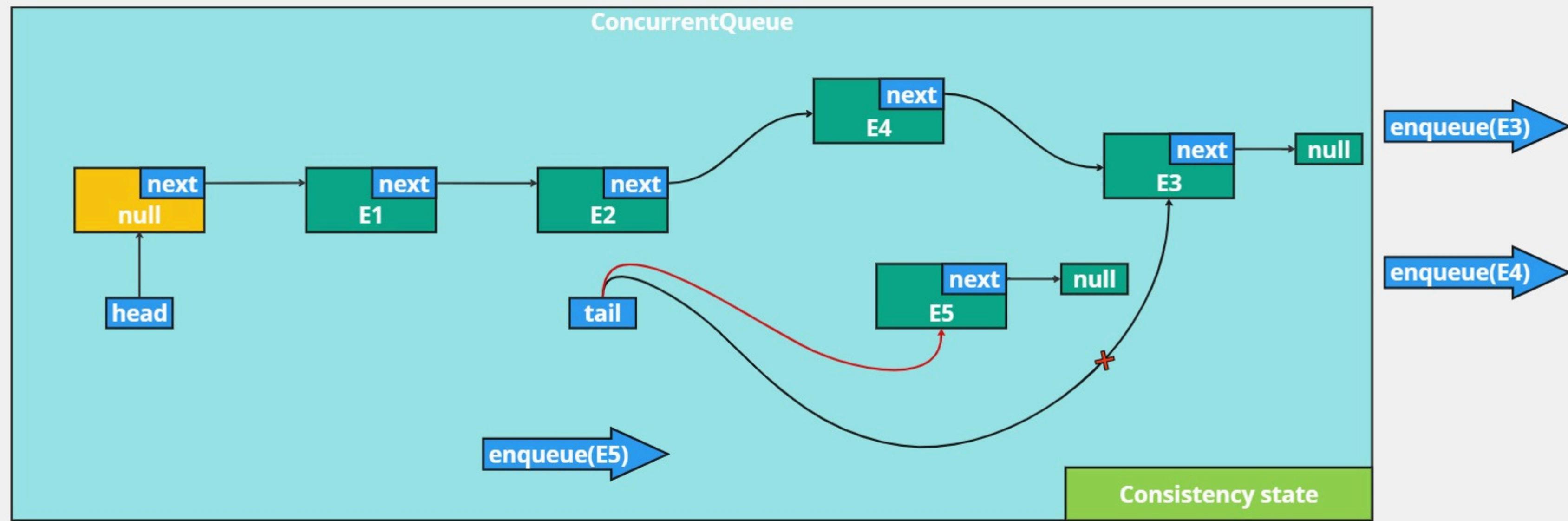
### enqueue(E4)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next == null) {
            tail.next = newNode;
            break;
        }
    }
    tail = newNode;
}
```

### enqueue(E5)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next == null) {
            tail.next = newNode;
            break;
        }
    }
    tail = newNode;
}
```

**Consistency state**



`enqueue(E3)`

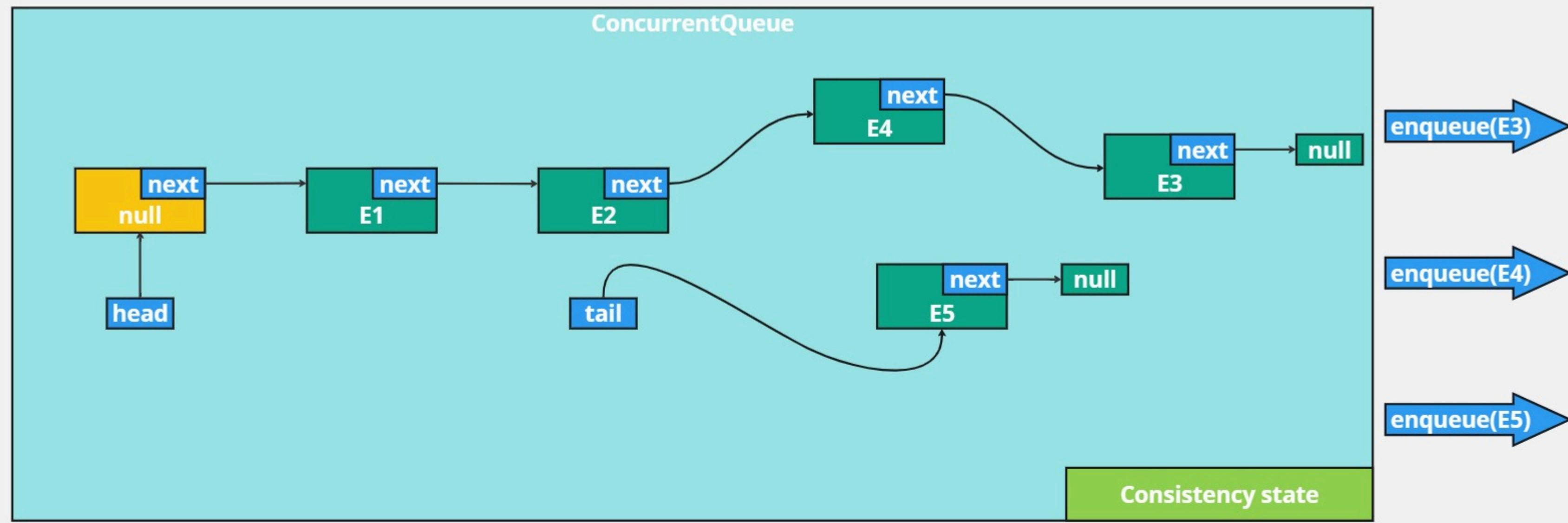
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next == null) {
            tail.next = newNode;
            break;
        }
    }
    tail = newNode;
}
```

`enqueue(E4)`

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next == null) {
            tail.next = newNode;
            break;
        }
    }
    tail = newNode;
}
```

`enqueue(E5)`

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next == null) {
            tail.next = newNode;
            break;
        }
    }
    tail = newNode;
}
```



**enqueue(E3)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next == null) {
            tail.next = newNode;
            break;
        }
    }
    tail = newNode;
}
```

**enqueue(E4)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next == null) {
            tail.next = newNode;
            break;
        }
    }
    tail = newNode;
}
```

**enqueue(E5)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next == null) {
            tail.next = newNode;
            break;
        }
    }
    tail = newNode;
}
```

```
public void enqueue(final E element) {  
    final Node<E> newNode = new Node<>(element);  
    while (true) {  
        if (tail.next == null) {  
            tail.next = newNode;  
            break;  
        }  
    }  
    tail = newNode;  
}
```

*not atomic*

```
private static final class Node<E> {  
    1 usage  
    private volatile E value;  
    no usages  
private volatile Node<E> next;  
  
    1 usage  
    public Node() {  
    }  
  
    no usages  
    public Node(final E value) {  
        this.value = value;  
    }  
}
```



```
private static final class Node<E> {  
    1 usage  
    private volatile E value;  
    3 usages  
private final AtomicReference<Node<E>> next;  
  
    1 usage  
    public Node() {  
        next = new AtomicReference<>();  
    }  
  
    1 usage  
    public Node(final E value) {  
        this.value = value;  
next = new AtomicReference<>();  
    }  
}
```

```
public void enqueue(final E element) {  
    final Node<E> newNode = new Node<>(element);  
    while (true) {  
        if (tail.next == null) {  
            tail.next = newNode;  
            break;  
        }  
        tail = newNode;  
    }  
}
```

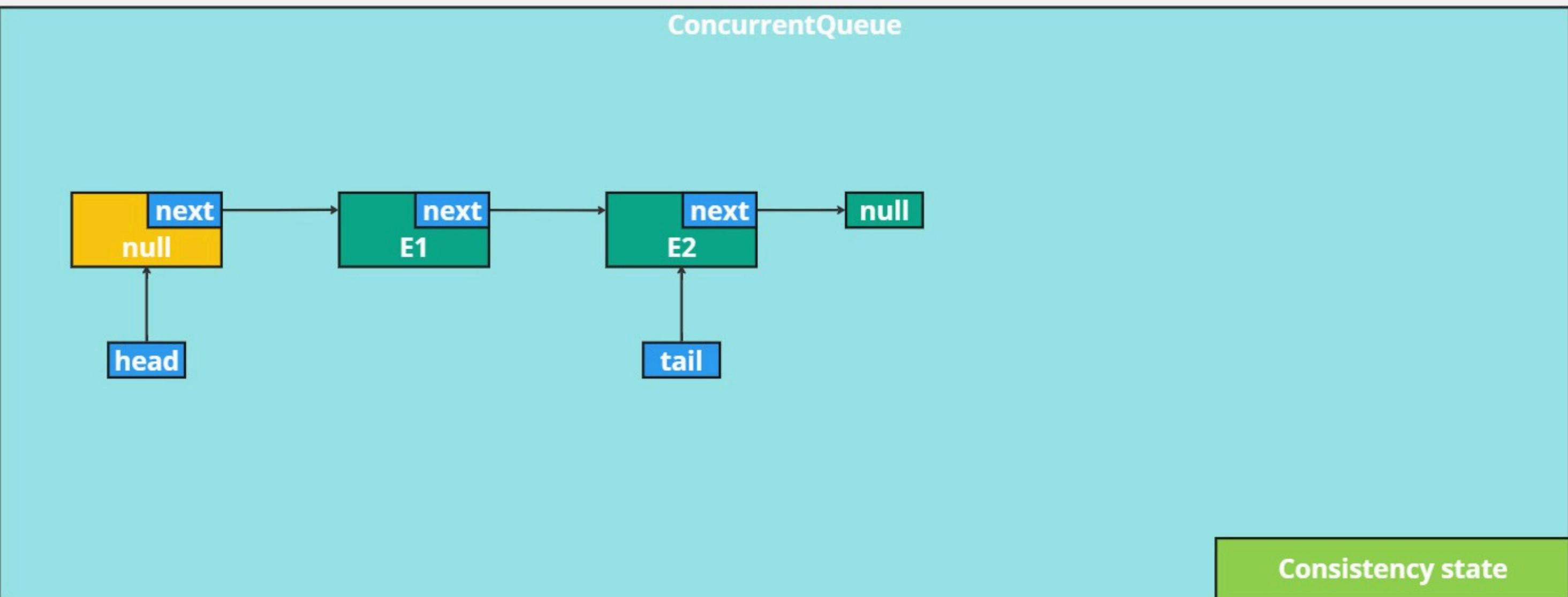


```
public void enqueue(final E element) {  
    final Node<E> newNode = new Node<>(element);  
    while (true) {  
        if (tail.next.compareAndSet(  
            expectedValue: null,  
            newValue: newNode)) {  
            break;  
        }  
        tail = newNode;  
    }  
}
```

atomic

## ConcurrentQueue

enqueue(E3) →  
enqueue(E4) →  
enqueue(E5) →



### enqueue(E3)

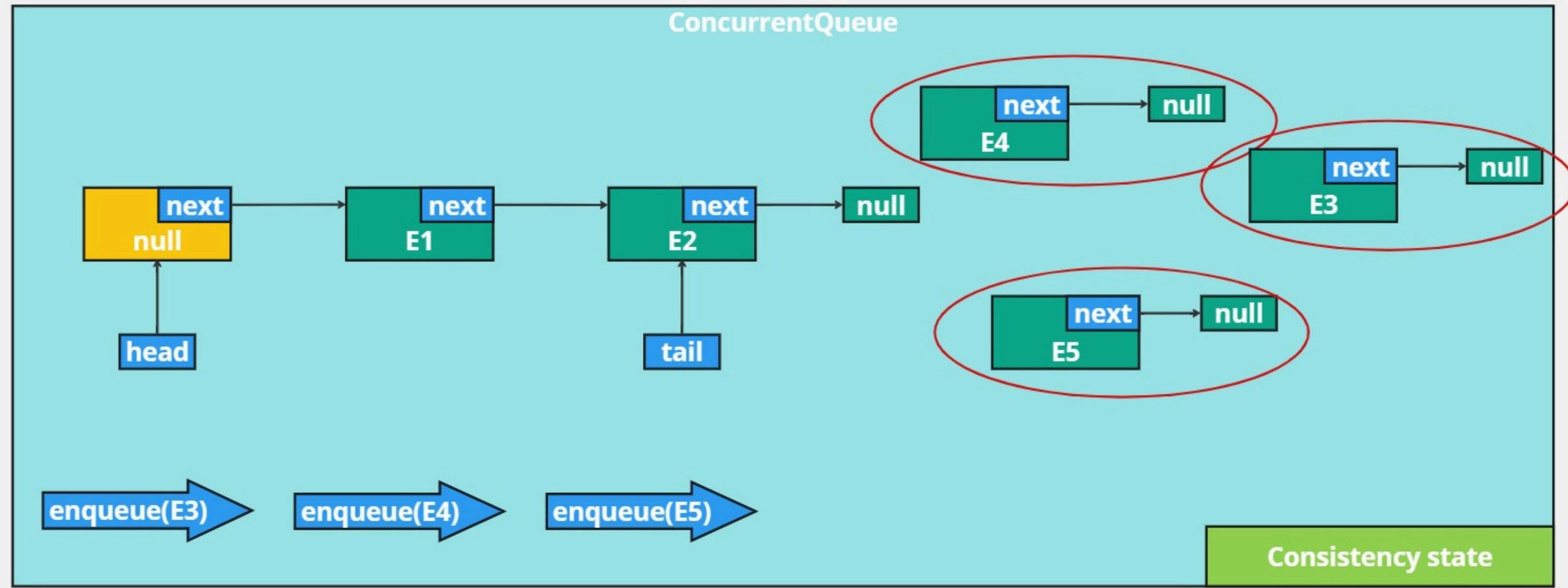
```
public void enqueue(final E element) {  
    final Node<E> newNode = new Node<>(element);  
    while (true) {  
        if (tail.next.compareAndSet(null, newNode)) {  
            break;  
        }  
    }  
    tail = newNode;  
}
```

### enqueue(E4)

```
public void enqueue(final E element) {  
    final Node<E> newNode = new Node<>(element);  
    while (true) {  
        if (tail.next.compareAndSet(null, newNode)) {  
            break;  
        }  
    }  
    tail = newNode;  
}
```

### enqueue(E5)

```
public void enqueue(final E element) {  
    final Node<E> newNode = new Node<>(element);  
    while (true) {  
        if (tail.next.compareAndSet(null, newNode)) {  
            break;  
        }  
    }  
    tail = newNode;  
}
```



### enqueue(E3)

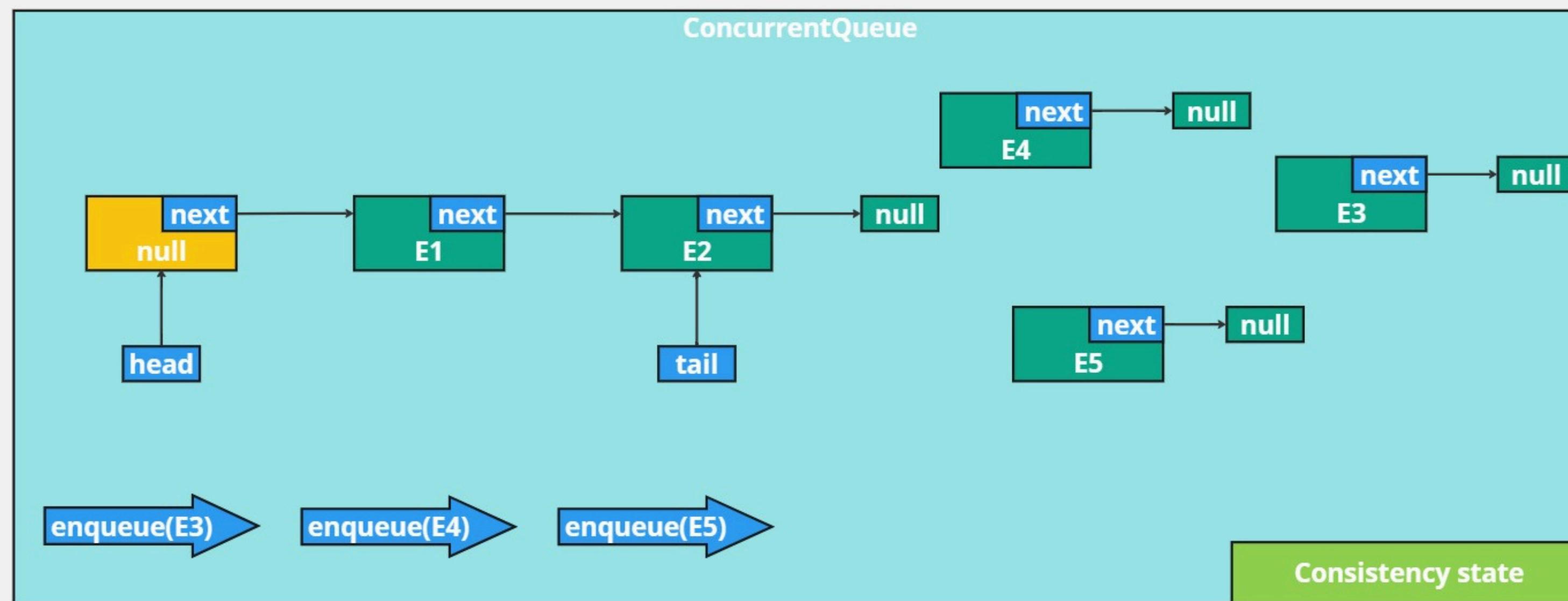
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```

### enqueue(E4)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```

### enqueue(E5)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```



**enqueue(E3)**

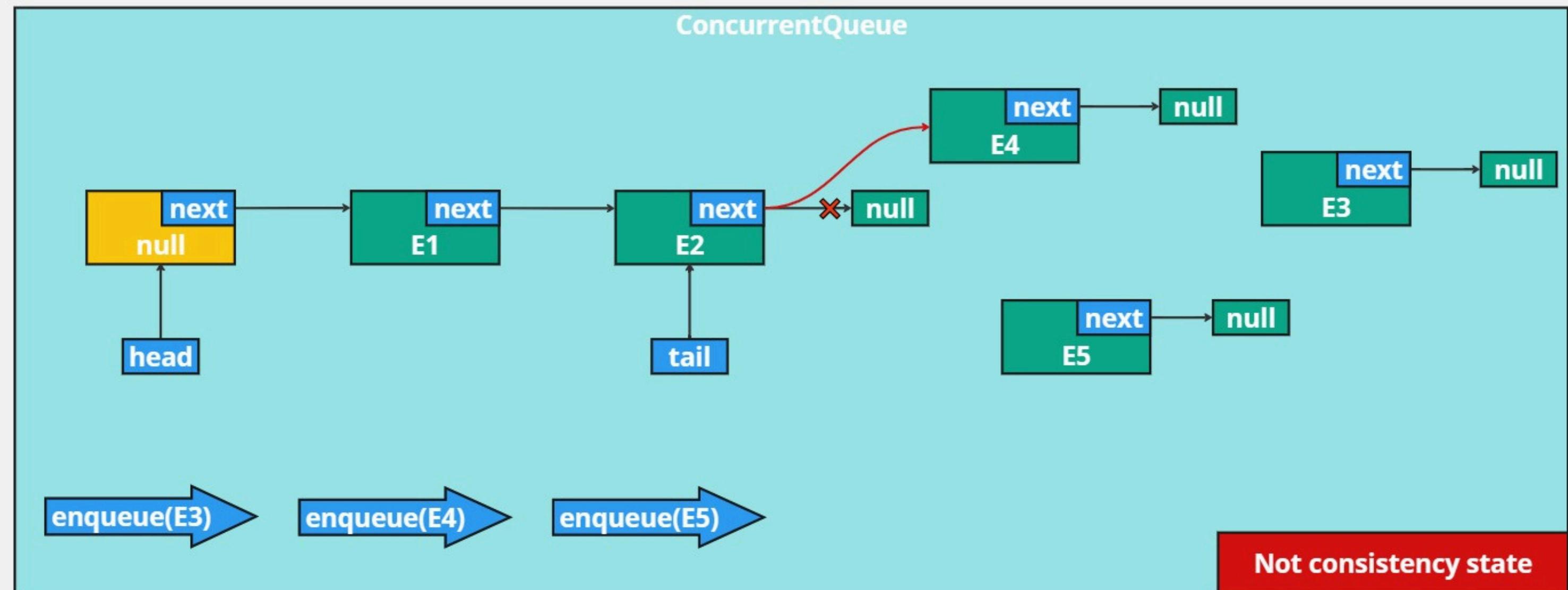
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```

**enqueue(E4)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```

**enqueue(E5)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```



**enqueue(E3)**

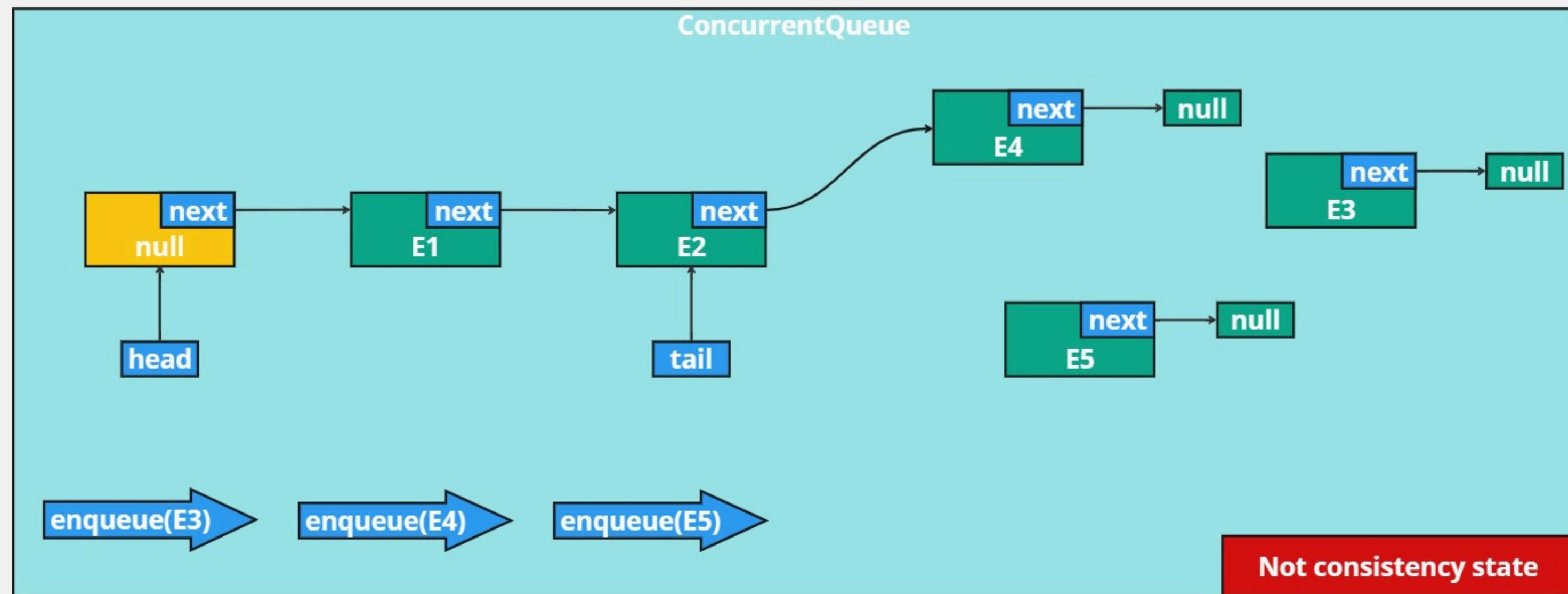
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```

**enqueue(E4)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```

**enqueue(E5)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```



### enqueue(E3)

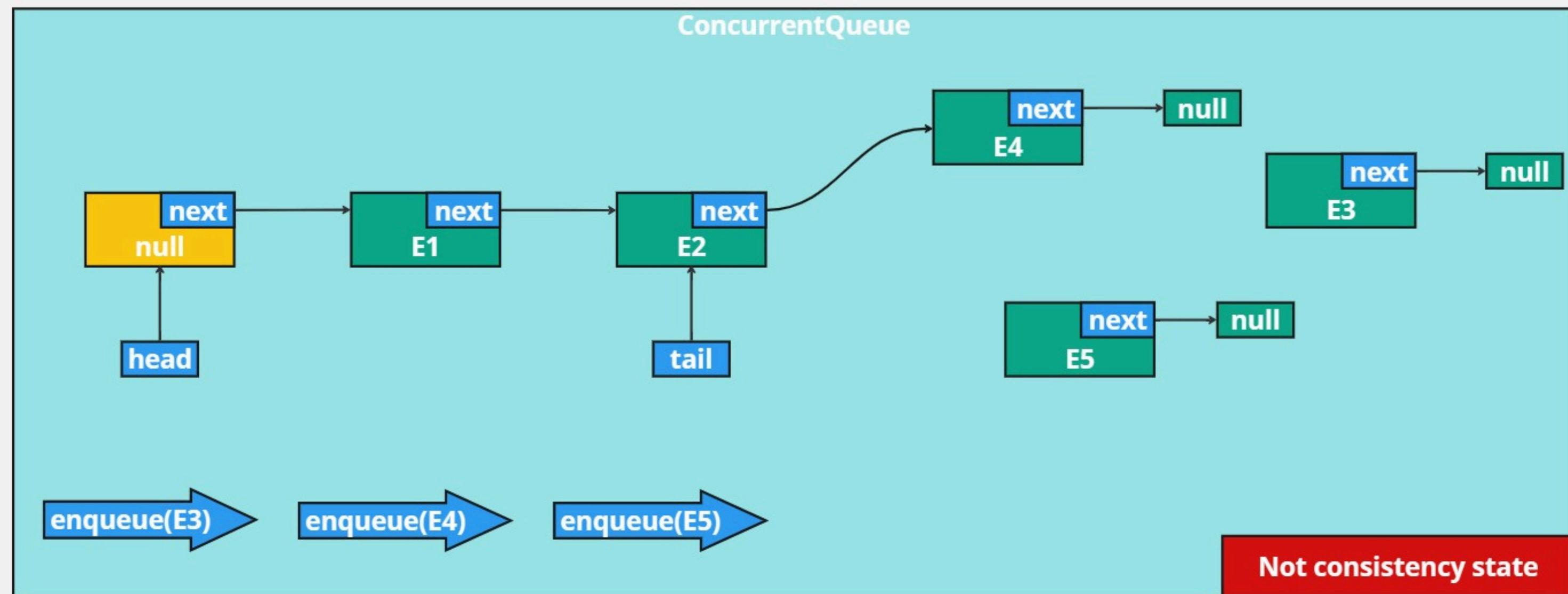
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```

### enqueue(E4)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```

### enqueue(E5)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```



### **enqueue(E3)**

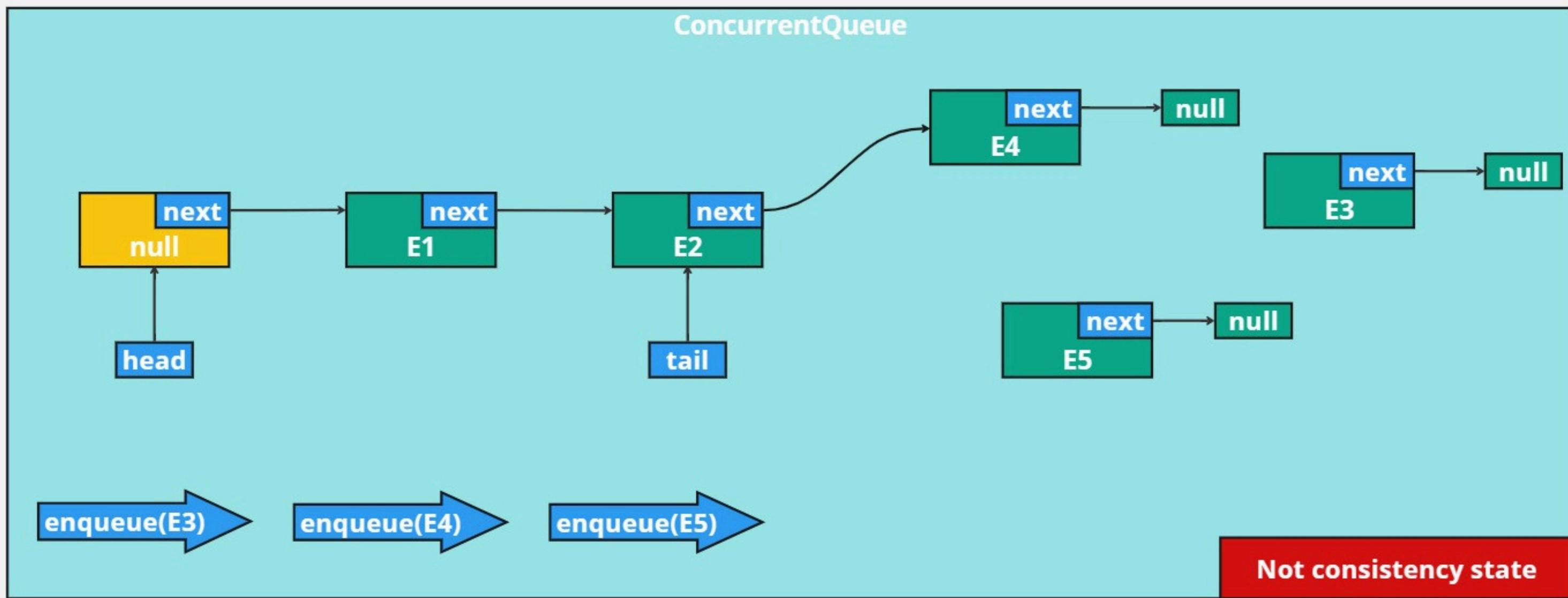
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```

### **enqueue(E4)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```

### **enqueue(E5)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```



**enqueue(E3)**

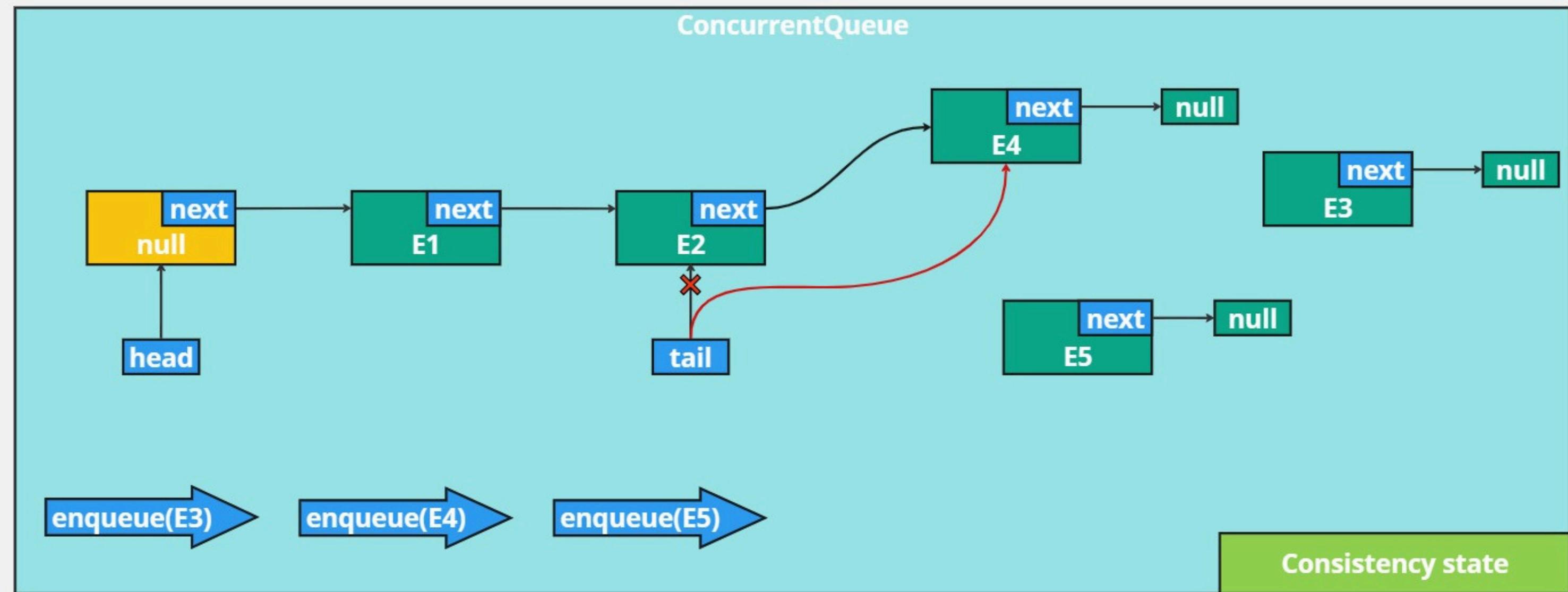
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```

**enqueue(E4)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```

**enqueue(E5)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```



### `enqueue(E3)`

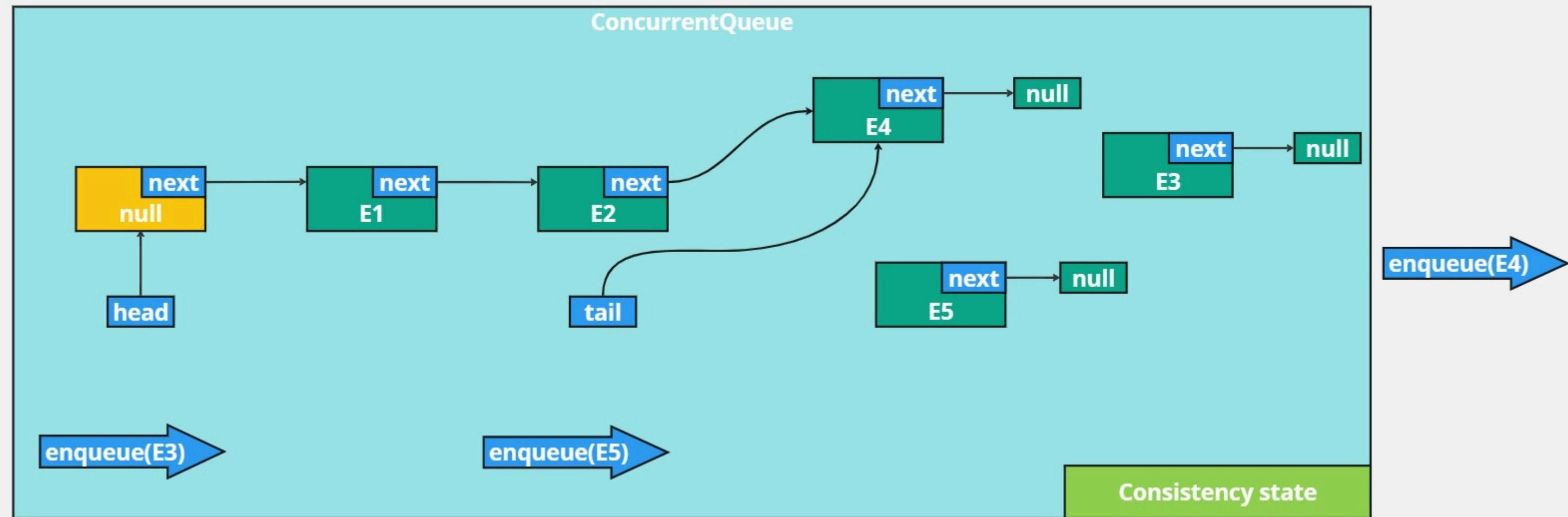
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```

### `enqueue(E4)`

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```

### `enqueue(E5)`

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```



**enqueue(E3)**

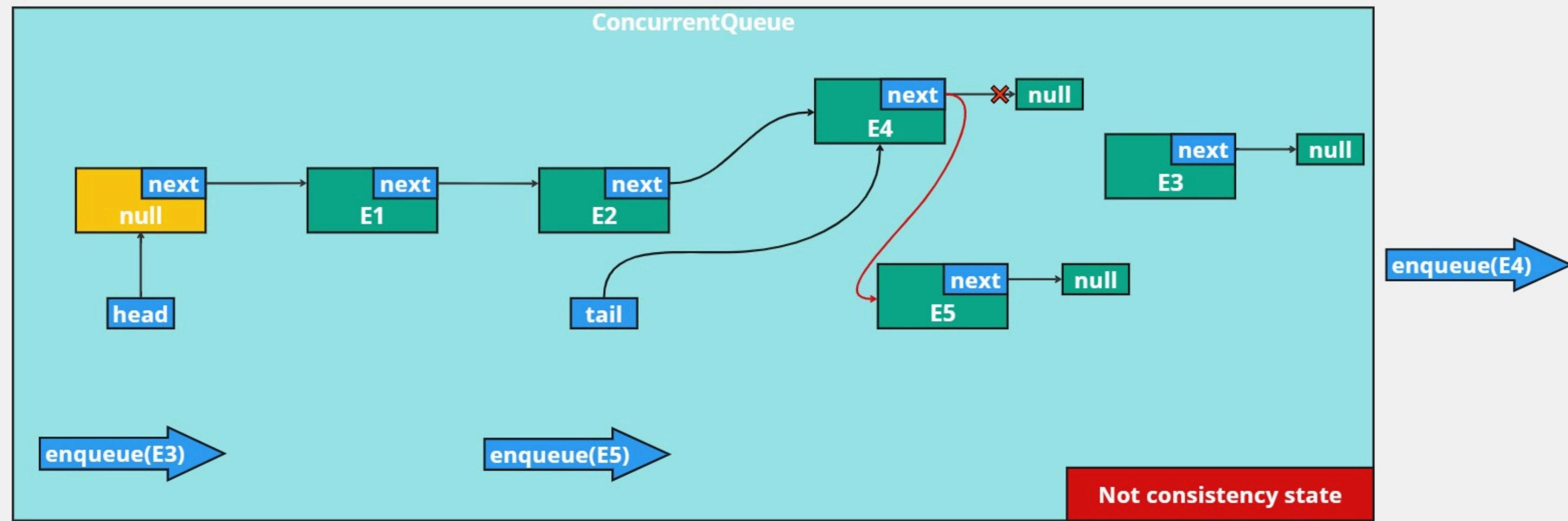
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```

**enqueue(E4)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```

**enqueue(E5)**

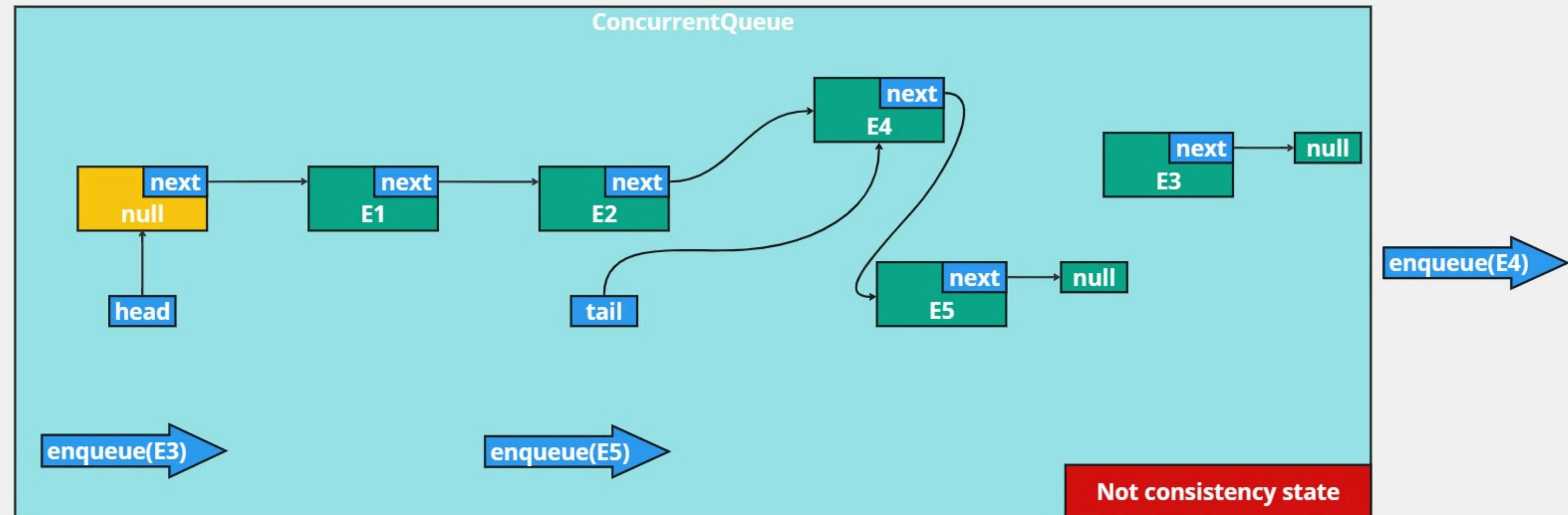
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```



```
enqueue(E3)
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```

```
enqueue(E4)
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```

```
enqueue(E5)
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```



`enqueue(E3)`

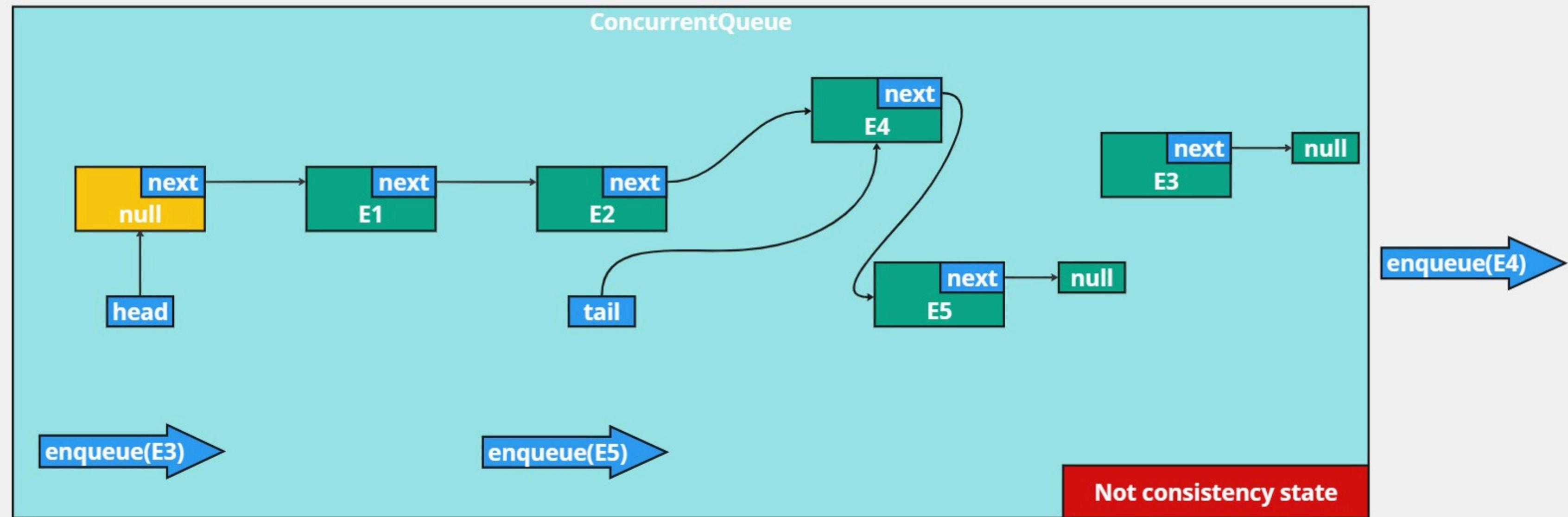
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```

`enqueue(E4)`

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```

`enqueue(E5)`

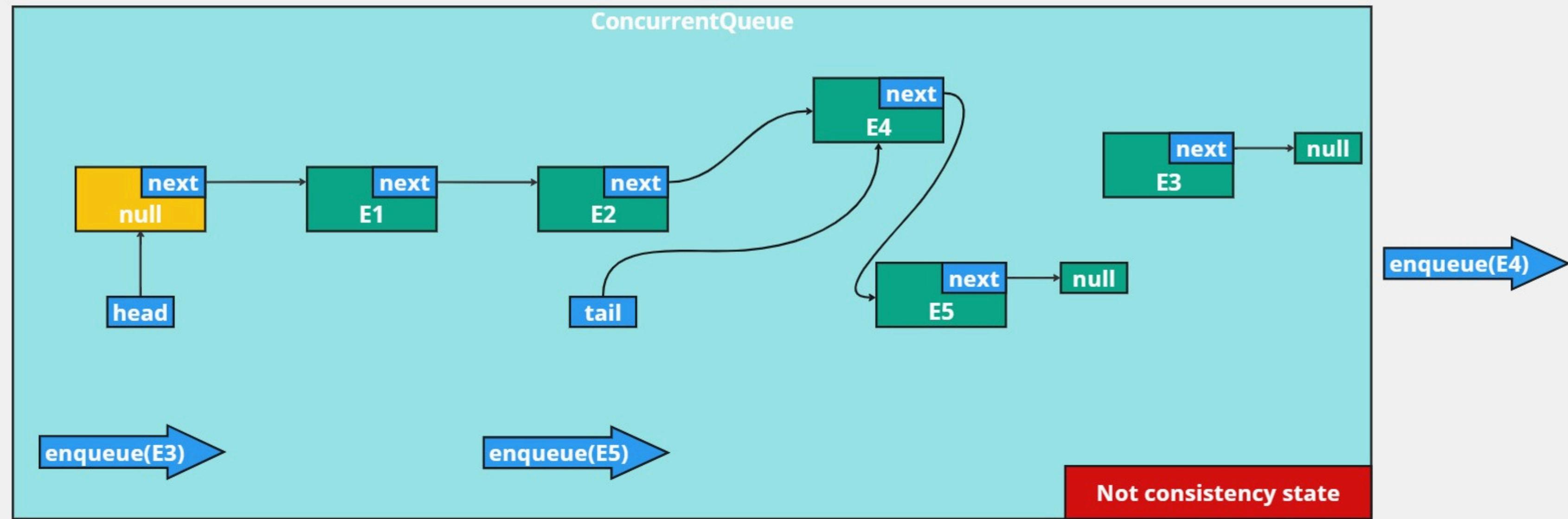
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```



```
enqueue(E3)
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```

```
enqueue(E4)
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```

```
enqueue(E5)
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```



**enqueue(E3)**

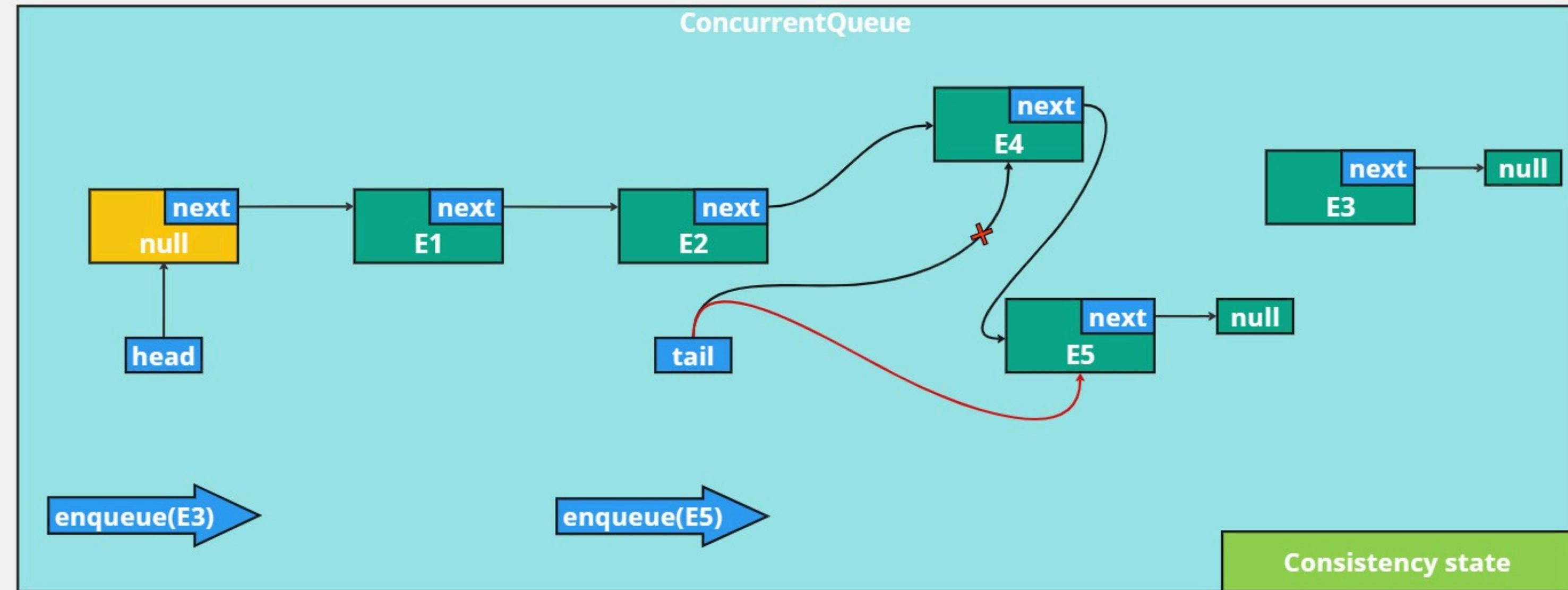
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```

**enqueue(E4)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```

**enqueue(E5)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```



### enqueue(E3)

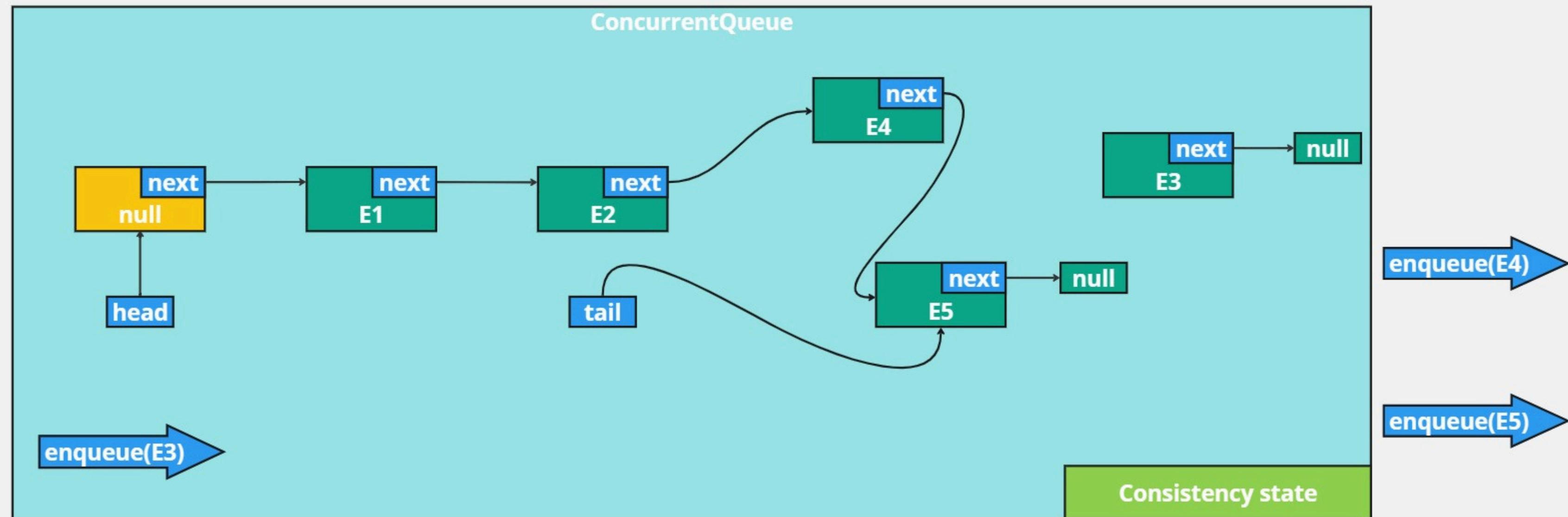
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```

### enqueue(E4)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```

### enqueue(E5)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```



**enqueue(E3)**

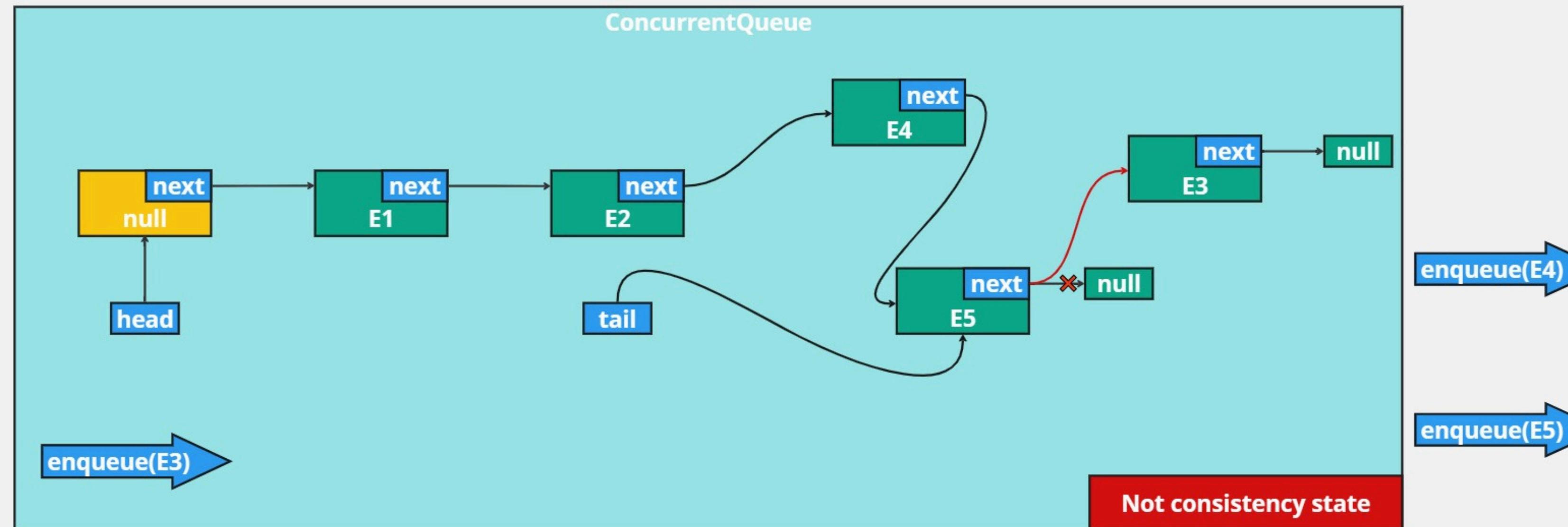
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```

**enqueue(E4)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```

**enqueue(E5)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```



**enqueue(E3)**

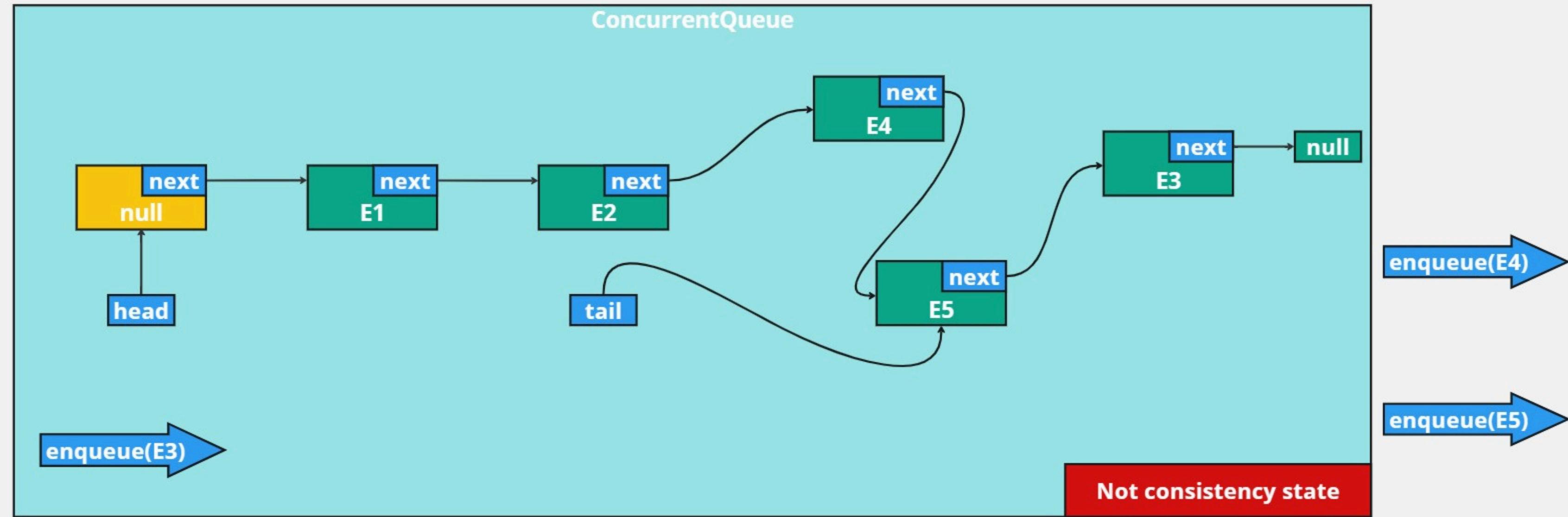
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```

**enqueue(E4)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```

**enqueue(E5)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```



### enqueue(E3)

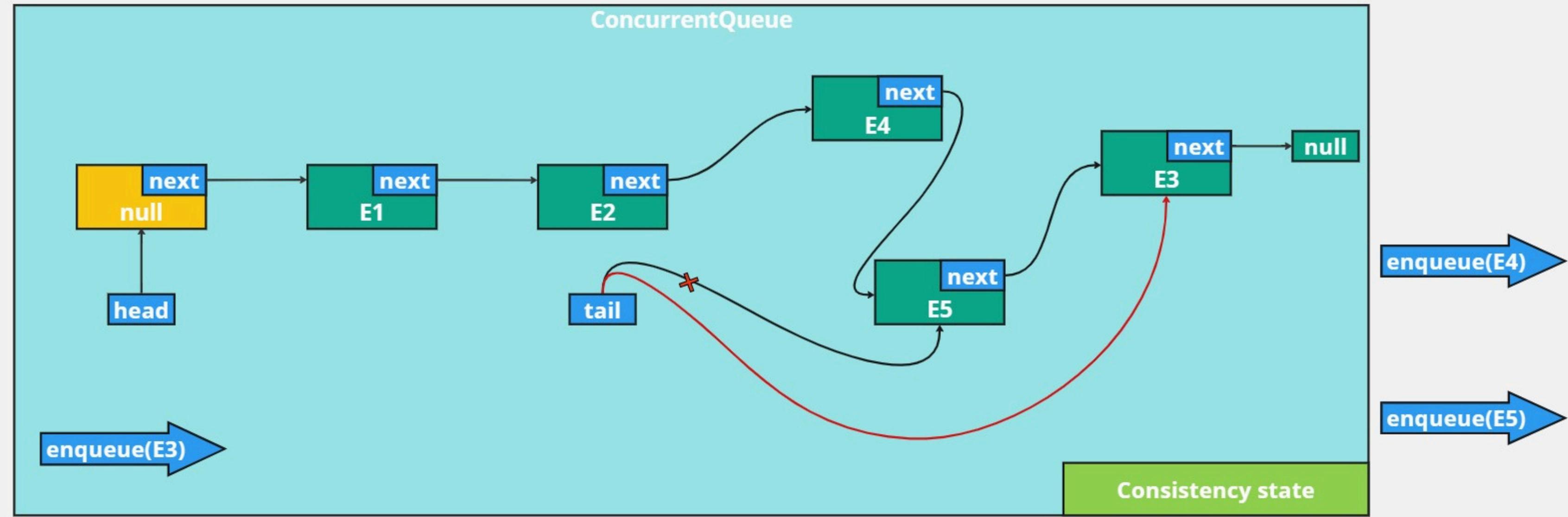
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```

### enqueue(E4)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```

### enqueue(E5)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```



### enqueue(E3)

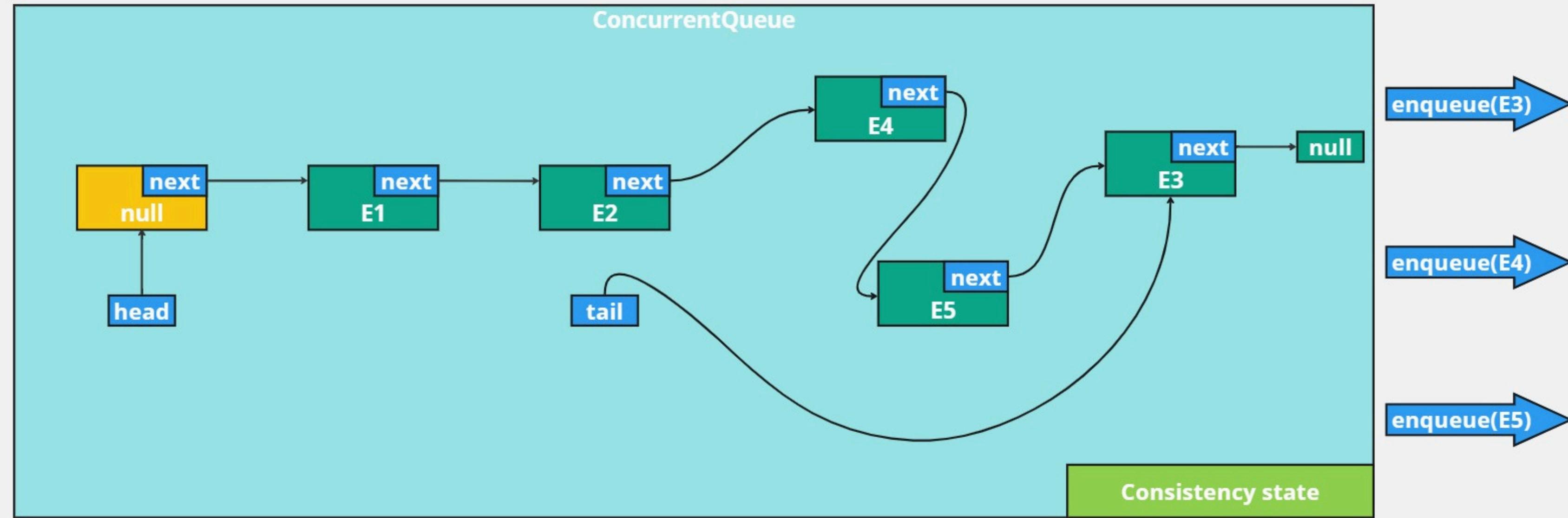
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```

### enqueue(E4)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```

### enqueue(E5)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```



### enqueue(E3)

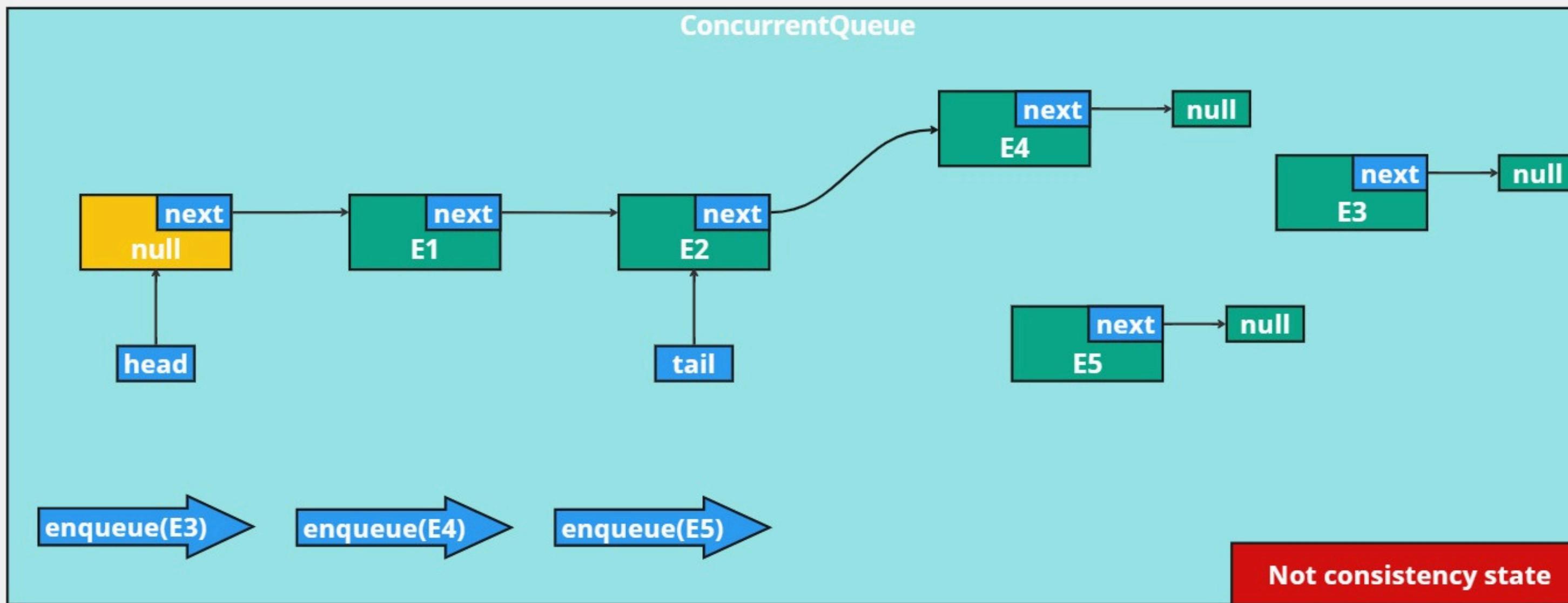
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```

### enqueue(E4)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```

### enqueue(E5)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```



**enqueue(E3)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```

**enqueue(E4)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```

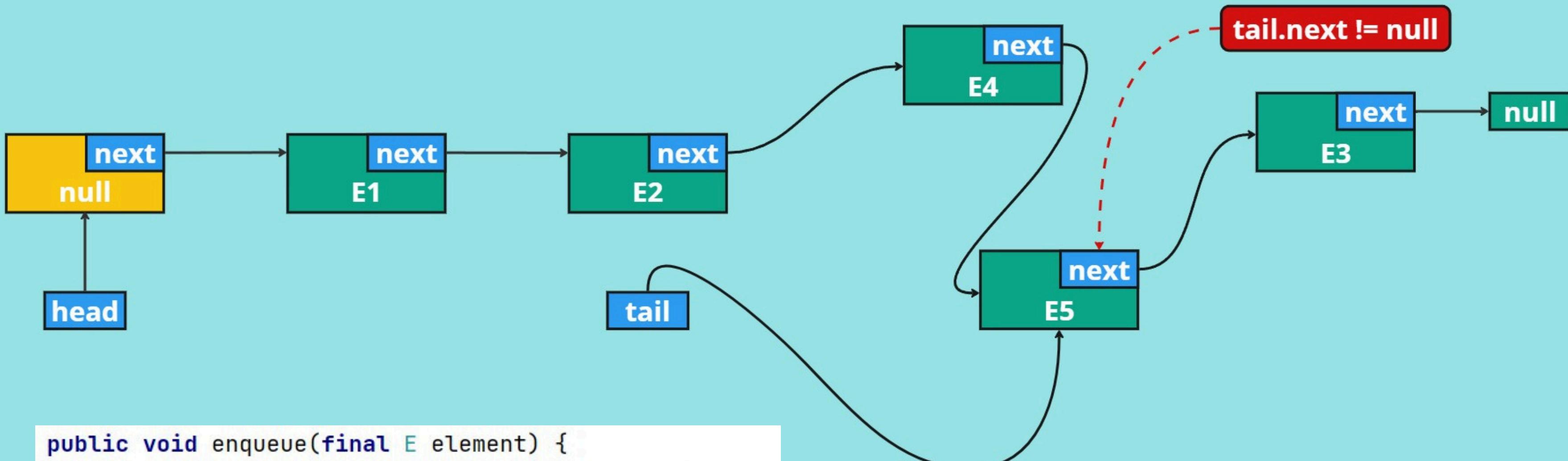
**enqueue(E5)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        }
    }
    tail = newNode;
}
```

```
public void enqueue(final E element) {  
    final Node<E> newNode = new Node<>(element);  
    while (true) {  
        if (tail.next.compareAndSet(null, newNode)) {  
            break;  
        }  
    }  
    tail = newNode;  
}  
  
public void enqueue(final E element) {  
    final Node<E> newNode = new Node<>(element);  
    while (true) {  
        if (tail.next.compareAndSet(null, newNode)) {  
            break;  
        }  
    }  
    tail = newNode;  
}  
  
public void enqueue(final E element) {  
    final Node<E> newNode = new Node<>(element);  
    while (true) {  
        if (tail.next.compareAndSet(null, newNode)) {  
            break;  
        }  
    }  
    tail = newNode;  
}  
  
public void enqueue(final E element) {  
    final Node<E> newNode = new Node<>(element);  
    while (true) {  
        if (tail.next.compareAndSet(null, newNode)) {  
            break;  
        }  
    }  
    tail = newNode;  
}  
  
public void enqueue(final E element) {  
    final Node<E> newNode = new Node<>(element);  
    while (true) {  
        if (tail.next.compareAndSet(null, newNode)) {  
            break;  
        }  
    }  
    tail = newNode;  
}
```

## *Not consistency state*

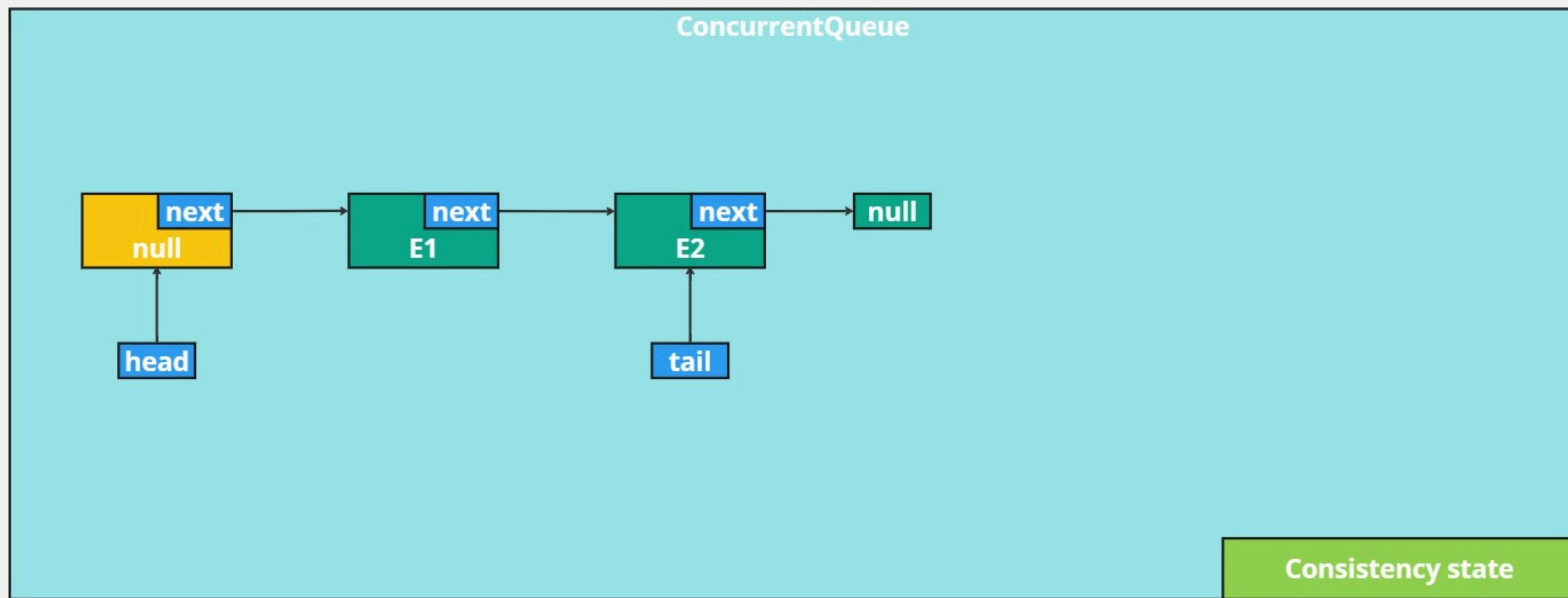
ConcurrentQueue



```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail = tail.next.get();
        }
    }
    tail = newNode;
}
```

## ConcurrentQueue

enqueue(E3) →  
 enqueue(E4) →  
 enqueue(E5) →



### enqueue(E3)

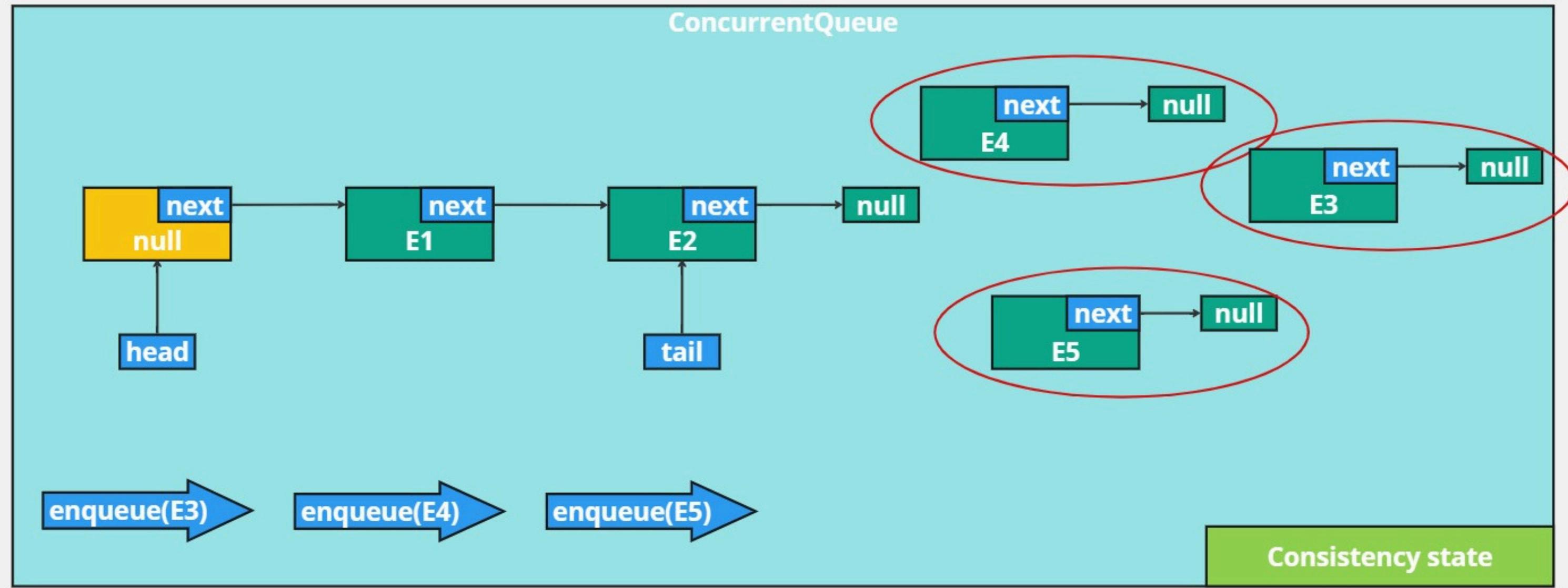
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail = tail.next.get();
        }
    }
    tail = newNode;
}
```

### enqueue(E4)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail = tail.next.get();
        }
    }
    tail = newNode;
}
```

### enqueue(E5)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail = tail.next.get();
        }
    }
    tail = newNode;
}
```



### `enqueue(E3)`

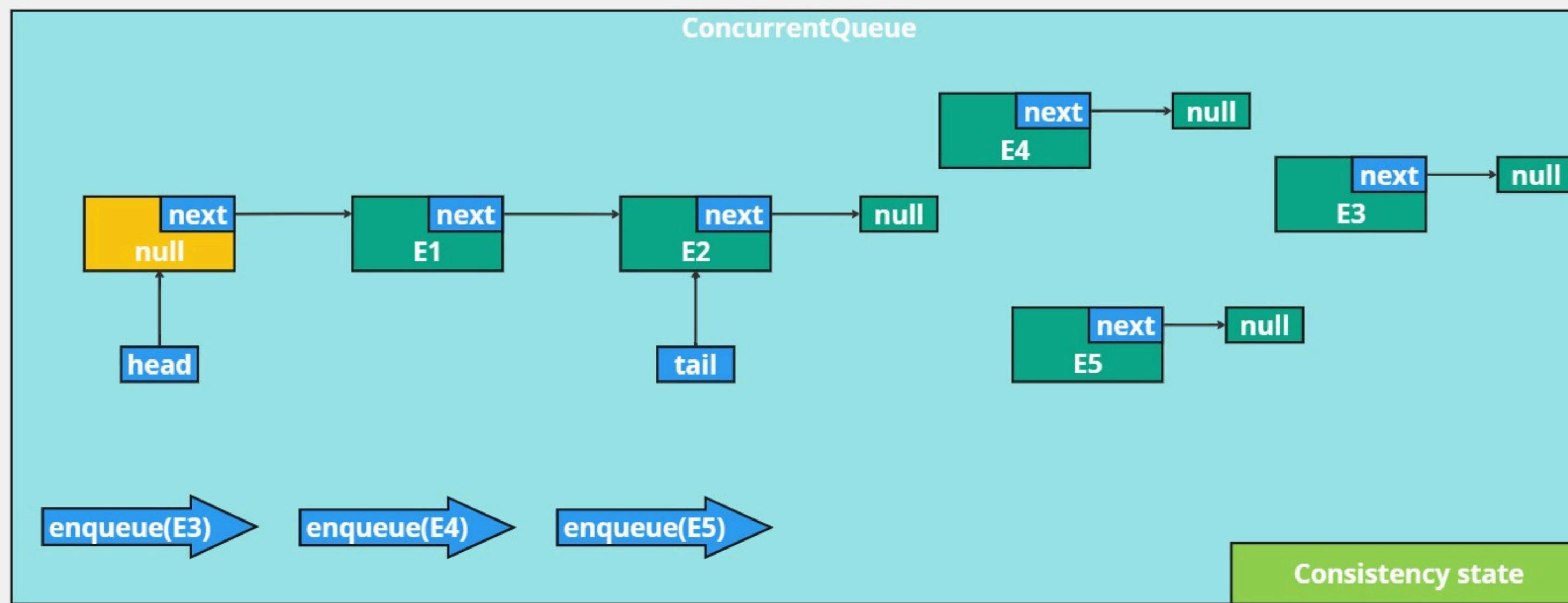
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail = tail.next.get();
        }
    }
    tail = newNode;
}
```

### `enqueue(E4)`

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail = tail.next.get();
        }
    }
    tail = newNode;
}
```

### `enqueue(E5)`

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail = tail.next.get();
        }
    }
    tail = newNode;
}
```



### `enqueue(E3)`

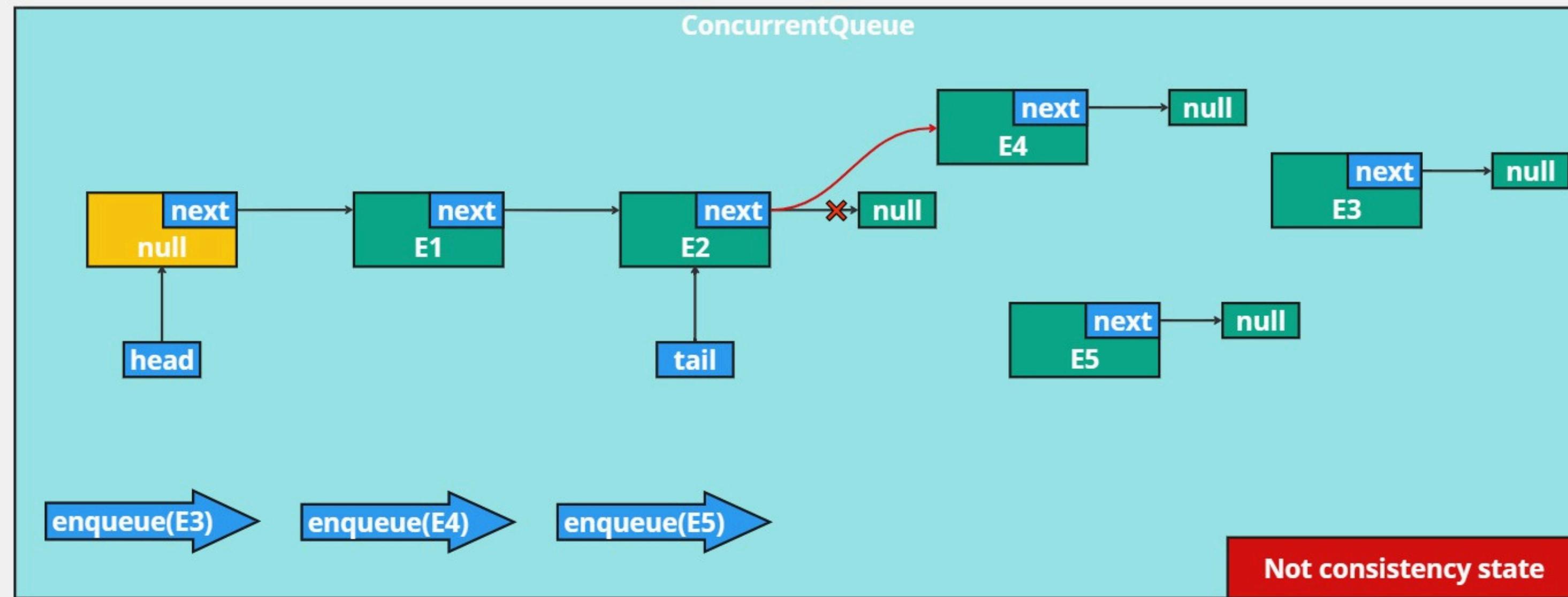
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail = tail.next.get();
        }
    }
    tail = newNode;
}
```

### `enqueue(E4)`

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail = tail.next.get();
        }
    }
    tail = newNode;
}
```

### `enqueue(E5)`

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail = tail.next.get();
        }
    }
    tail = newNode;
}
```



### `enqueue(E3)`

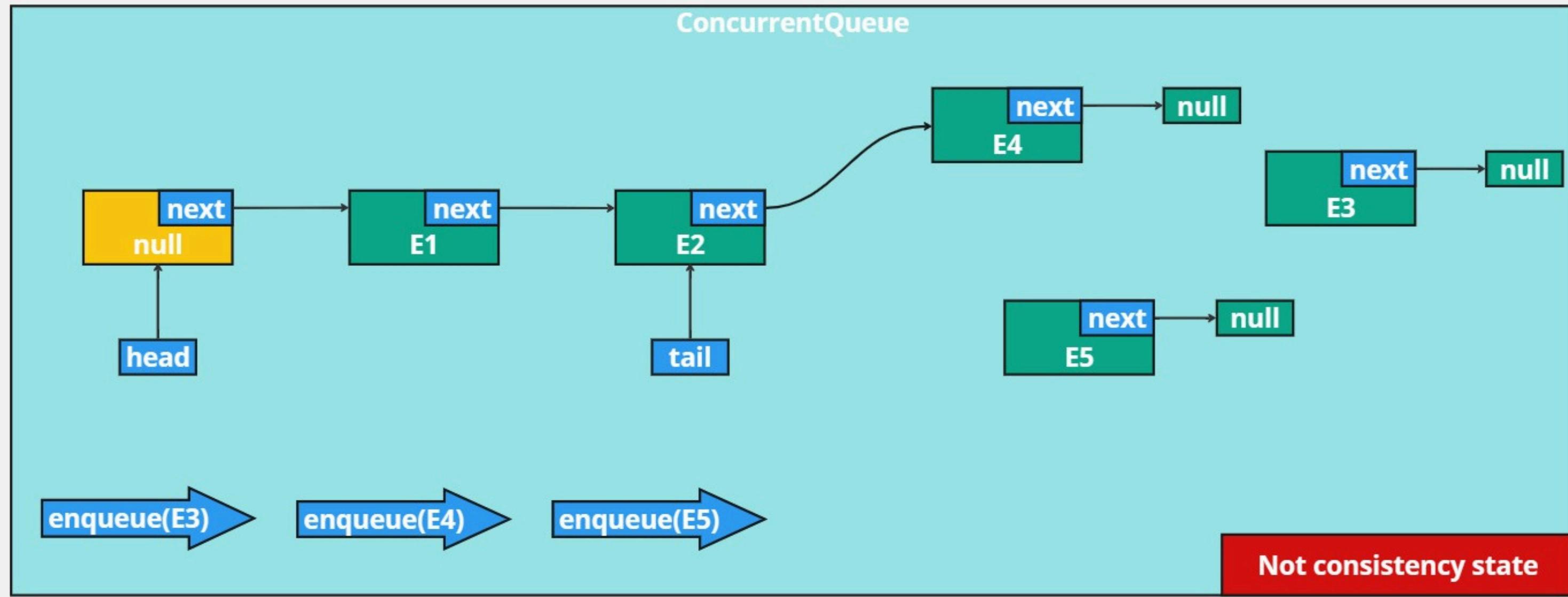
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail = tail.next.get();
        }
    }
    tail = newNode;
}
```

### `enqueue(E4)`

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail = tail.next.get();
        }
    }
    tail = newNode;
}
```

### `enqueue(E5)`

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail = tail.next.get();
        }
    }
    tail = newNode;
}
```



### enqueue(E3)

```

public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail = tail.next.get();
        }
    }
    tail = newNode;
}

```

### enqueue(E4)

```

public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail = tail.next.get();
        }
    }
    tail = newNode;
}

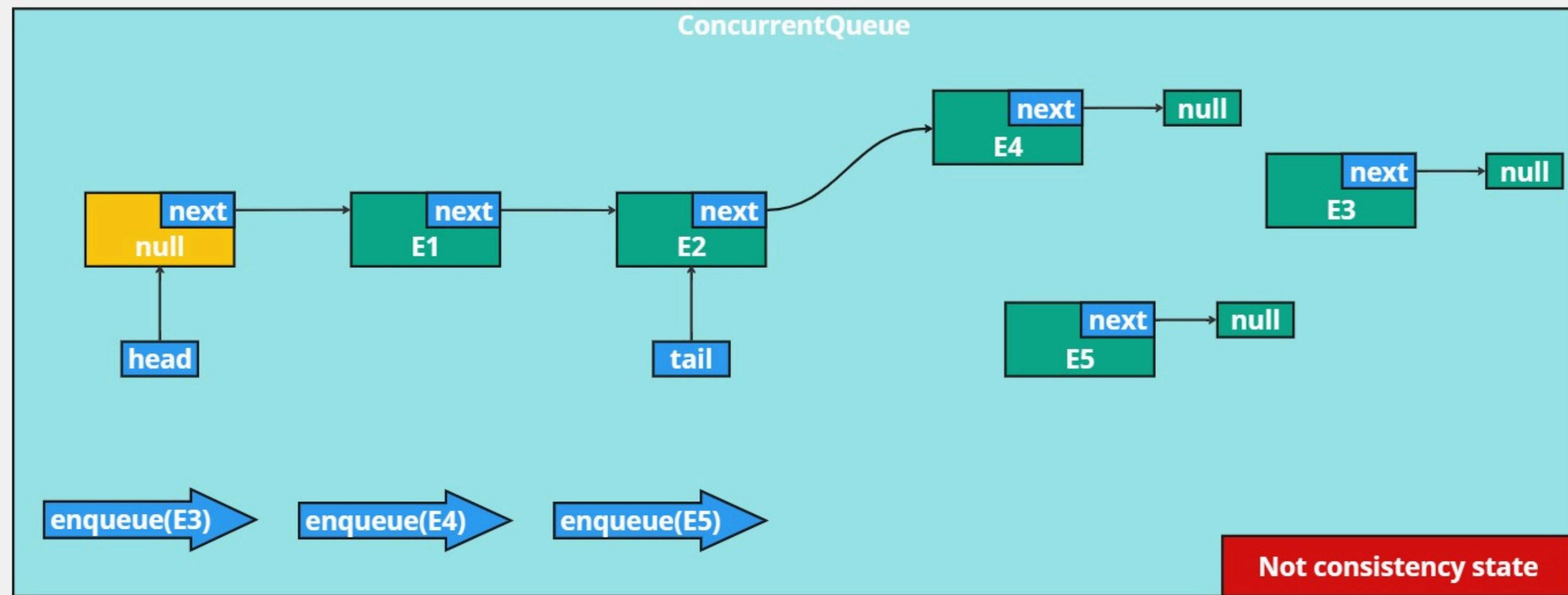
```

### enqueue(E5)

```

public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail = tail.next.get();
        }
    }
    tail = newNode;
}

```



### enqueue(E3)

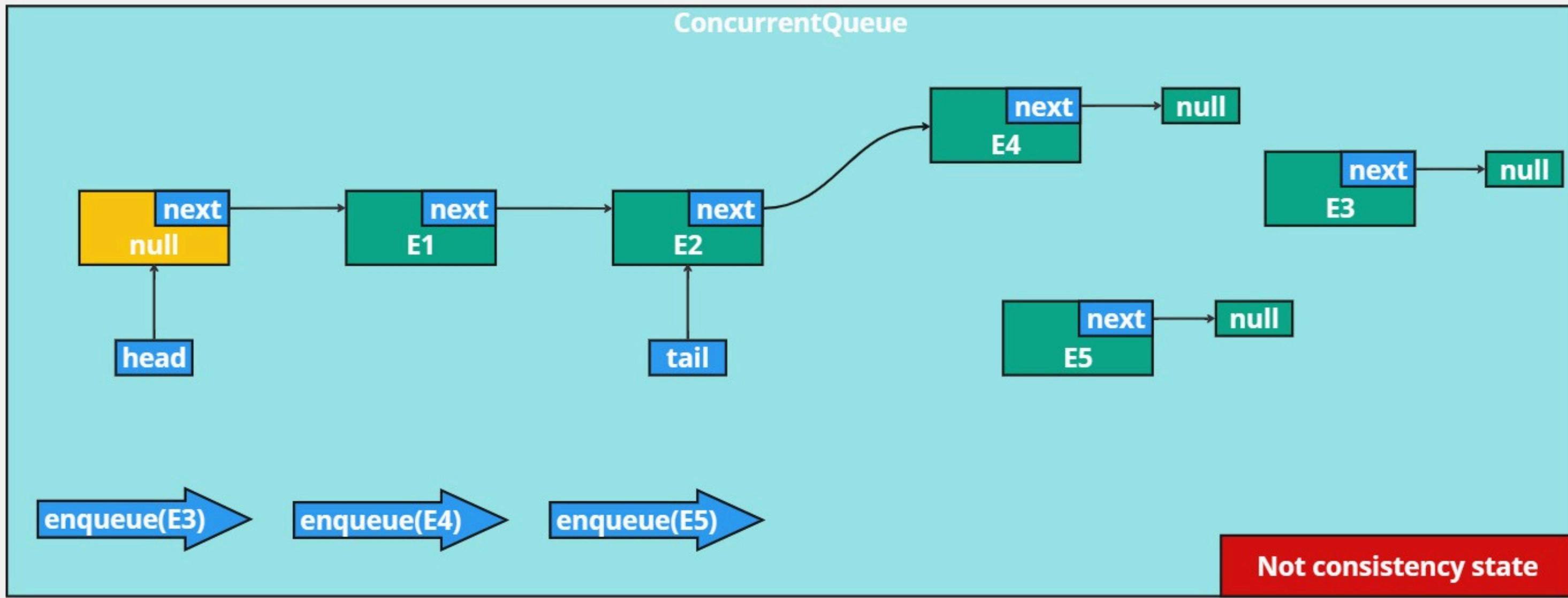
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail = tail.next.get();
        }
    }
    tail = newNode;
}
```

### enqueue(E4)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail = tail.next.get();
        }
    }
    tail = newNode;
}
```

### enqueue(E5)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail = tail.next.get();
        }
    }
    tail = newNode;
}
```



### enqueue(E3)

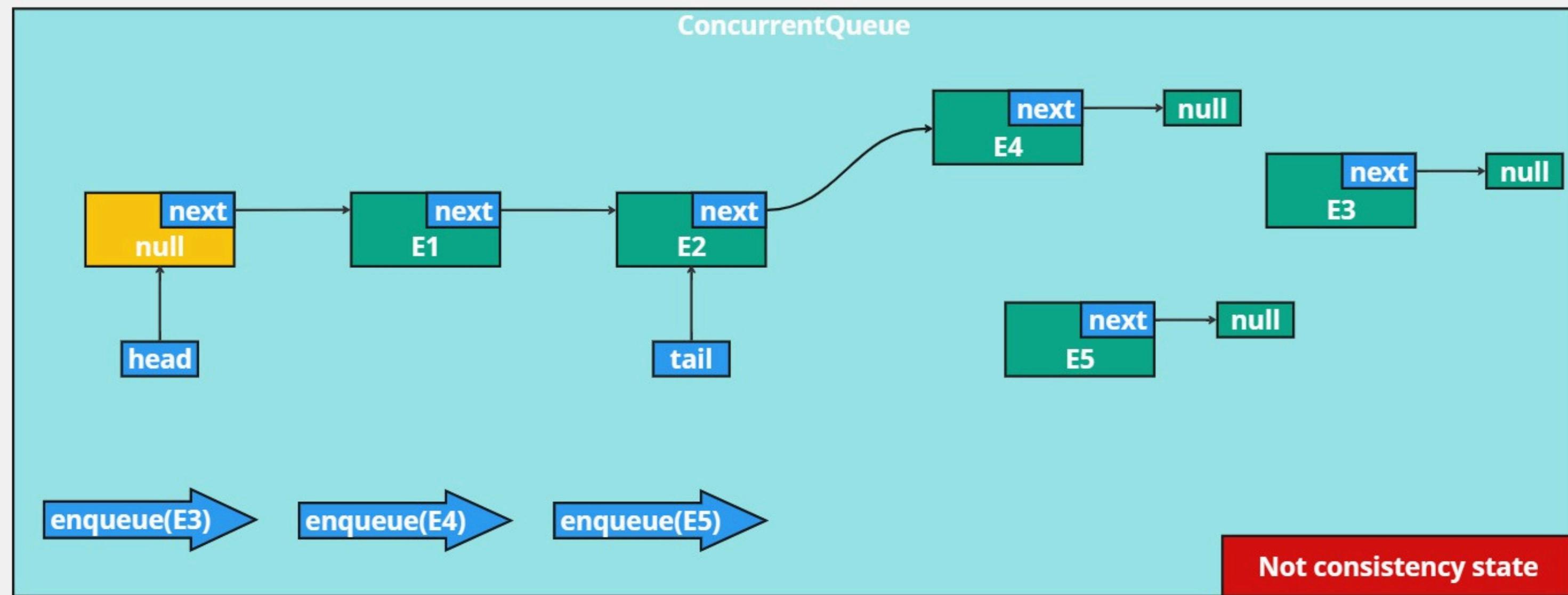
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail = tail.next.get();
        }
    }
    tail = newNode;
}
```

### enqueue(E4)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail = tail.next.get();
        }
    }
    tail = newNode;
}
```

### enqueue(E5)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail = tail.next.get();
        }
    }
    tail = newNode;
}
```



### enqueue(E3)

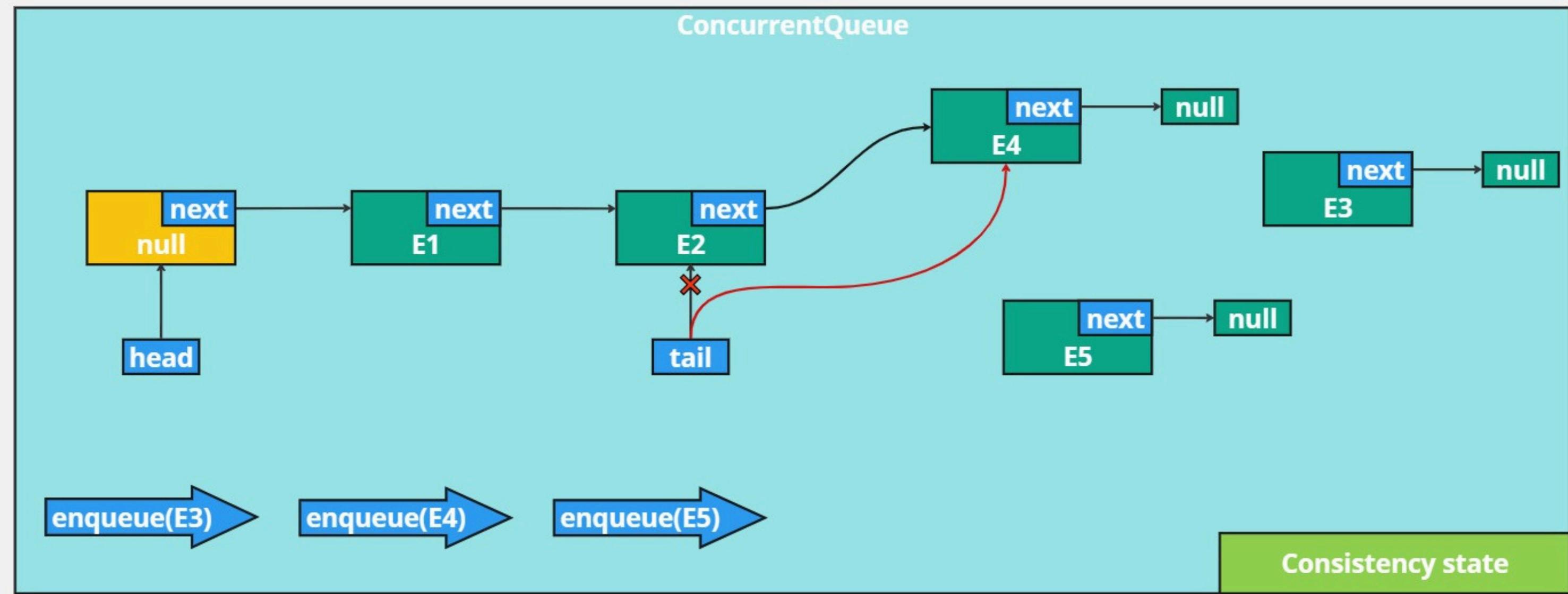
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail = tail.next.get();
        }
    }
    tail = newNode;
}
```

### enqueue(E4)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail = tail.next.get();
        }
    }
    tail = newNode;
}
```

### enqueue(E5)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail = tail.next.get();
        }
    }
    tail = newNode;
}
```



### `enqueue(E3)`

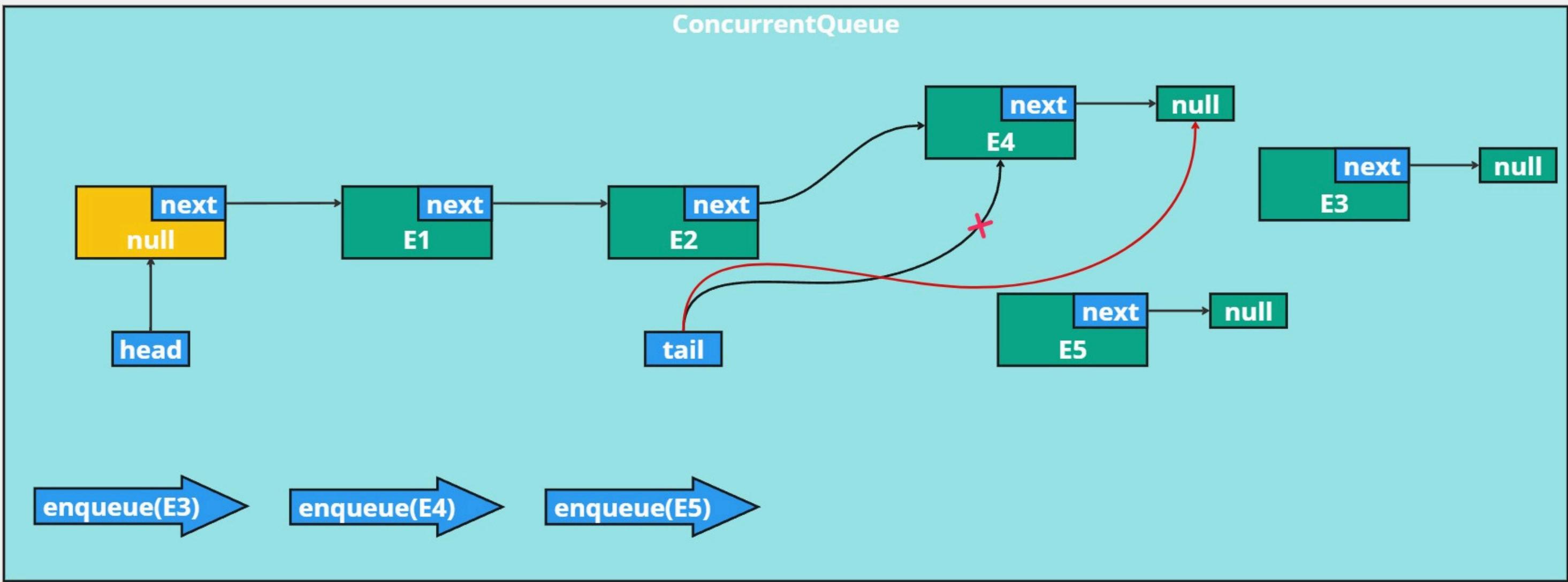
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail = tail.next.get();
        }
    }
    tail = newNode;
}
```

### `enqueue(E4)`

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail = tail.next.get();
        }
    }
    tail = newNode;
}
```

### `enqueue(E5)`

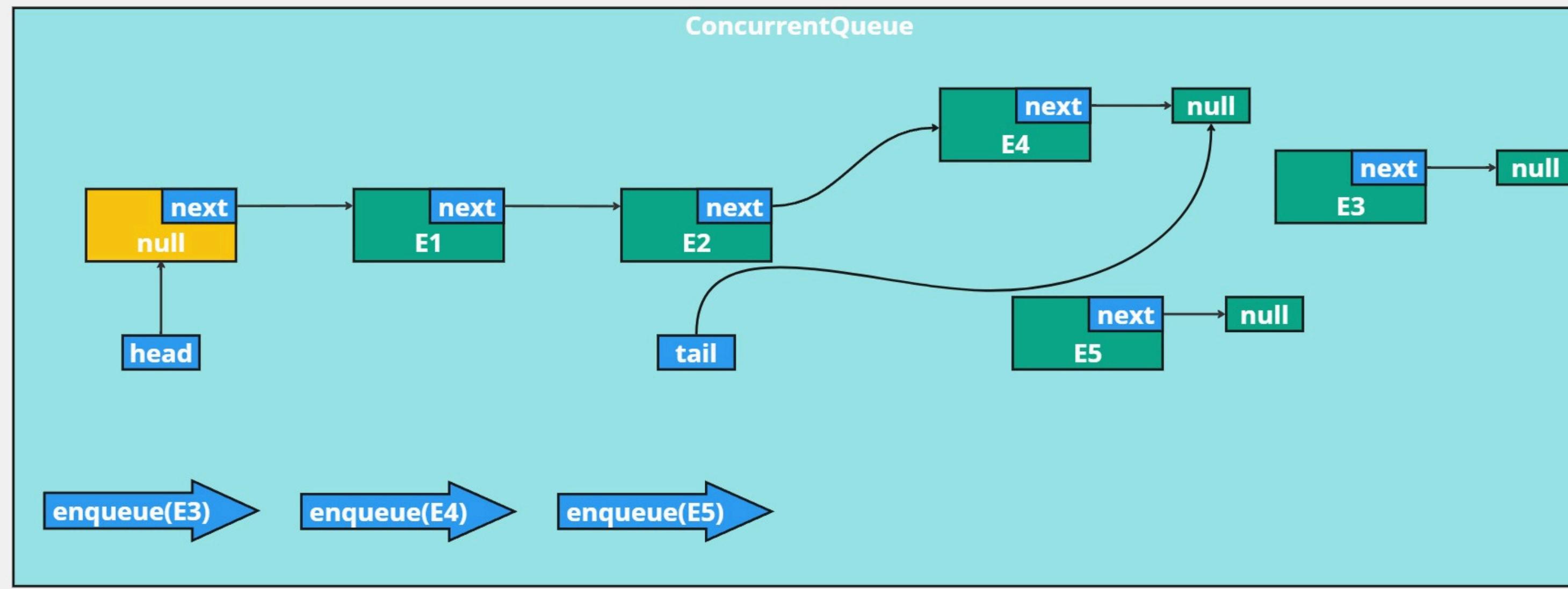
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail = tail.next.get();
        }
    }
    tail = newNode;
}
```



```
enqueue(E3)
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail = tail.next.get();
        }
    }
    tail = newNode;
}
```

```
enqueue(E4)
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail = tail.next.get();
        }
    }
    tail = newNode;
}
```

```
enqueue(E5)
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail = tail.next.get();
        }
    }
    tail = newNode;
}
```



### enqueue(E3)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail = tail.next.get();
        }
    }
    tail = newNode;
}
```

**✗ NullPointerException**

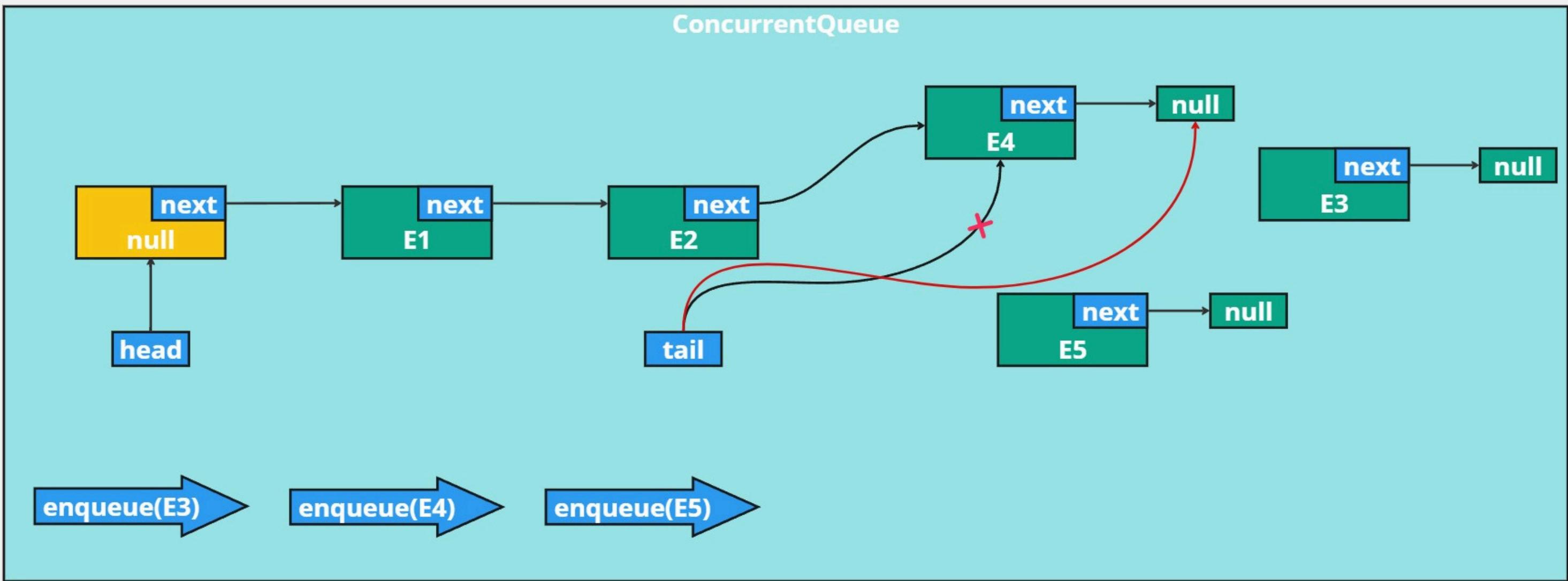
### enqueue(E4)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail = tail.next.get();
        }
    }
    tail = newNode;
}
```

### enqueue(E5)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail = tail.next.get();
        }
    }
    tail = newNode;
}
```

**✗ NullPointerException**



```
enqueue(E3)
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail = tail.next.get();
        }
    }
    tail = newNode;
}
```

```
enqueue(E4)
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail = tail.next.get();
        }
    }
    tail = newNode;
}
```

```
enqueue(E5)
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        if (tail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail = tail.next.get();
        }
    }
    tail = newNode;
}
```

```
public void enqueue(final E element) {  
    final Node<E> newNode = new Node<>(element);  
    while (true) {  
        if (tail.next.compareAndSet(null, newNode)) {  
            break;  
        } else {  
            tail = tail.next.get();  
        }  
    }  
    tail = newNode;  
}
```



```
public void enqueue(final E element) {  
    final Node<E> newNode = new Node<>(element);  
    while (true) {  
        final Node<E> previousTail = tail;  
        if (previousTail.next.compareAndSet(null, newNode)) {  
            break;  
        } else {  
            if (tail == previousTail) {  
                tail = previousTail.next.get();  
            }  
        }  
    }  
    tail = newNode;  
}
```

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail;
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            if (tail == previousTail) {
                tail = previousTail.next.get();
            }
        }
    }
    tail = newNode;
}
```

*not atomic*

```
private volatile Node<E> tail;
```

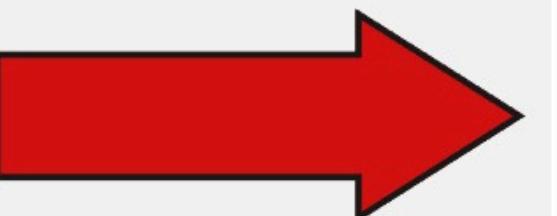
```
public ConcurrentQueue() {  
    final Node<E> dummyNode = new Node<>();  
    head = dummyNode;  
tail = dummyNode;  
}
```



```
private final AtomicReference<Node<E>> tail;
```

```
public ConcurrentQueue() {  
    final Node<E> dummyNode = new Node<>();  
    head = dummyNode;  
tail = new AtomicReference<>(dummyNode);  
}
```

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail;
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            if (tail == previousTail) {
                tail = previousTail.next.get();
            }
        }
    }
    tail = newNode;
}
```

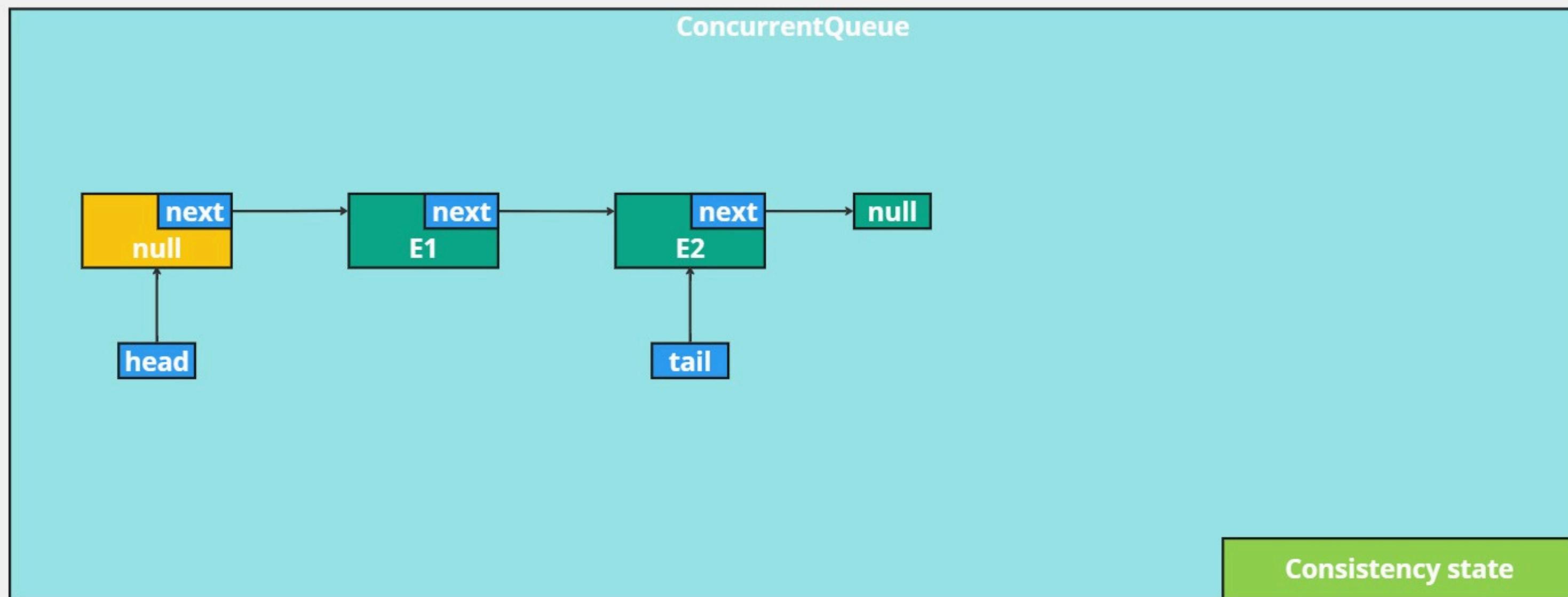


```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```

*atomic*

## ConcurrentQueue

**enqueue(E3)**  
**enqueue(E4)**  
**enqueue(E5)**



### enqueue(E3)

```

public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}

```

### enqueue(E4)

```

public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}

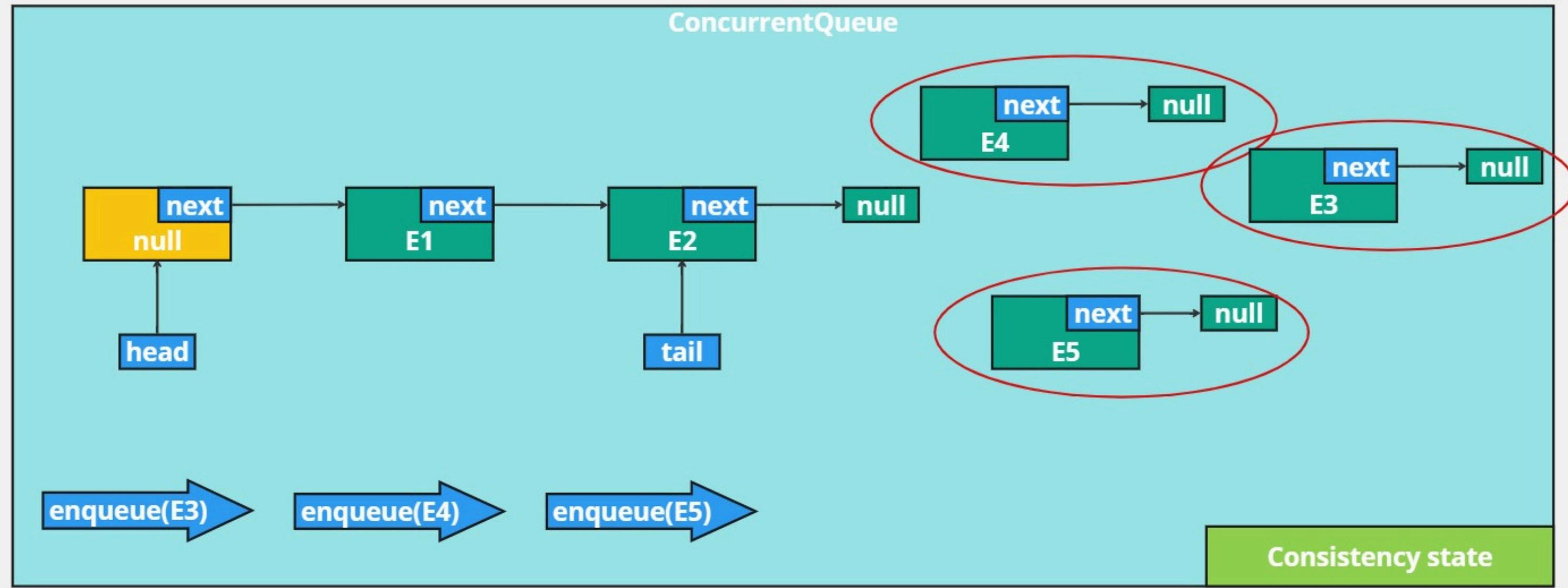
```

### enqueue(E5)

```

public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}

```



### enqueue(E3)

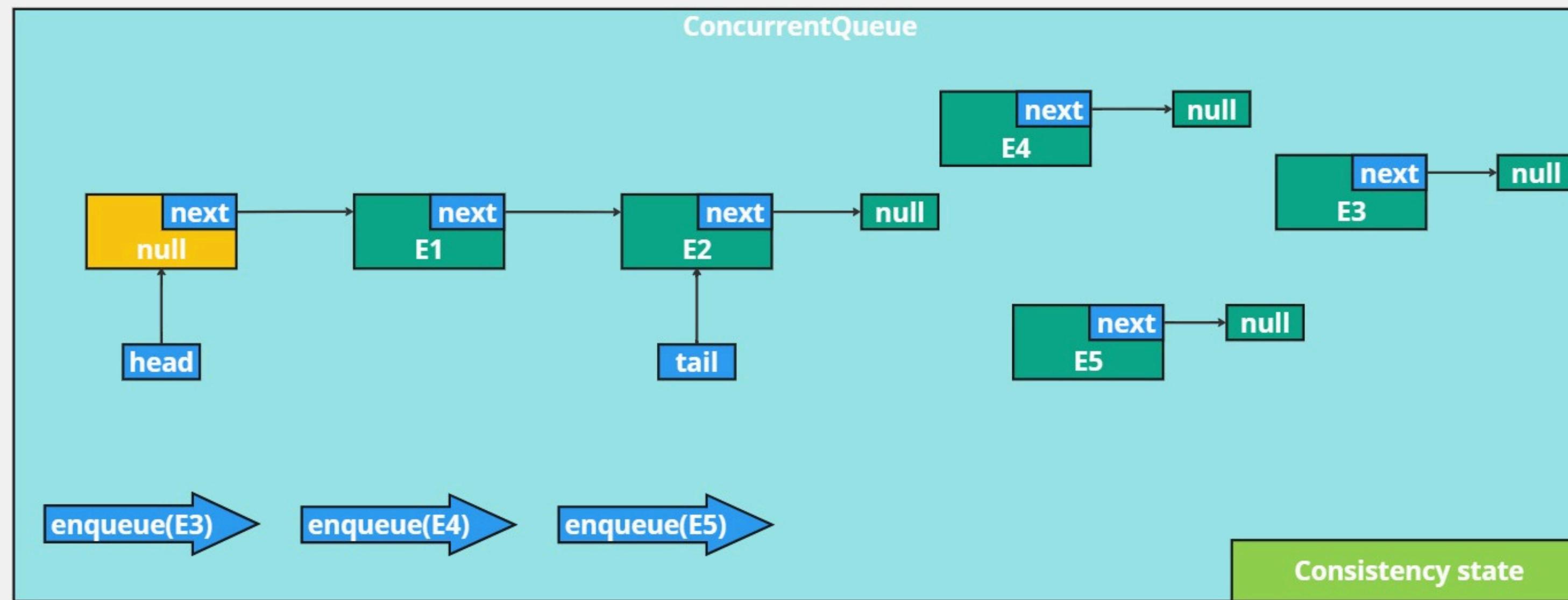
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```

### enqueue(E4)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```

### enqueue(E5)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```



### enqueue(E3)

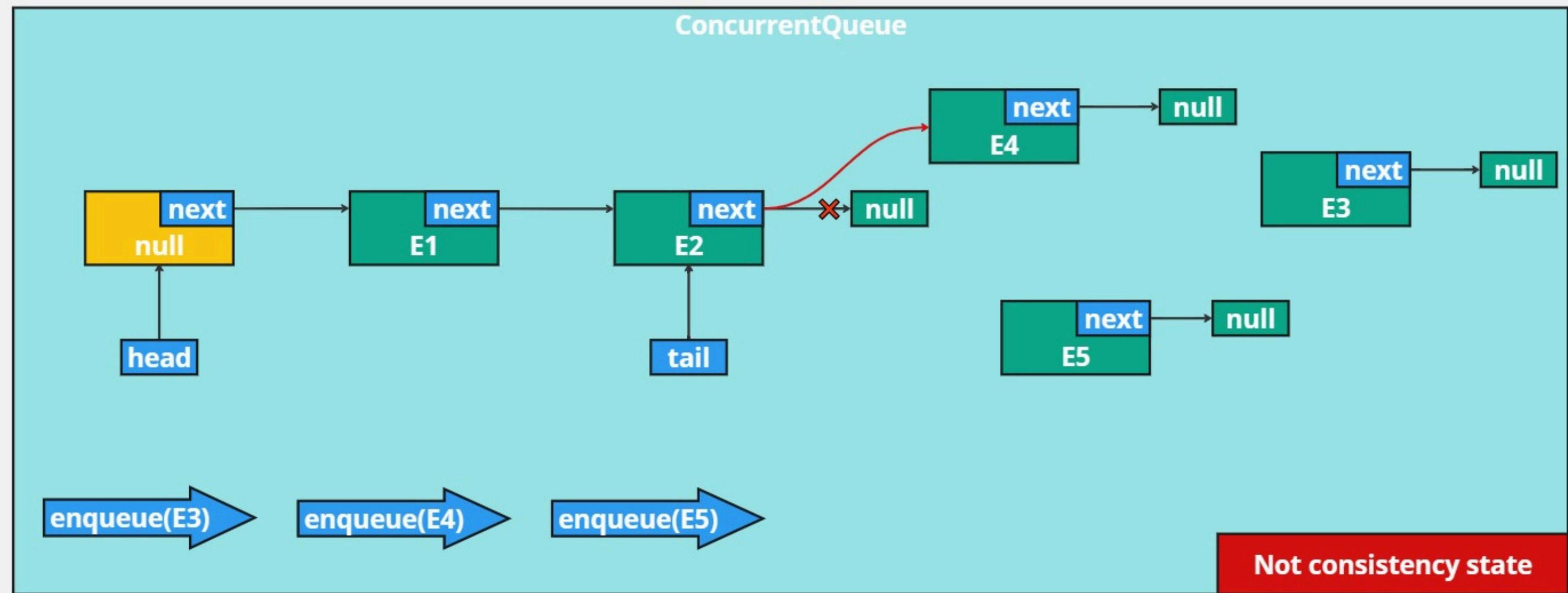
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```

### enqueue(E4)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```

### enqueue(E5)

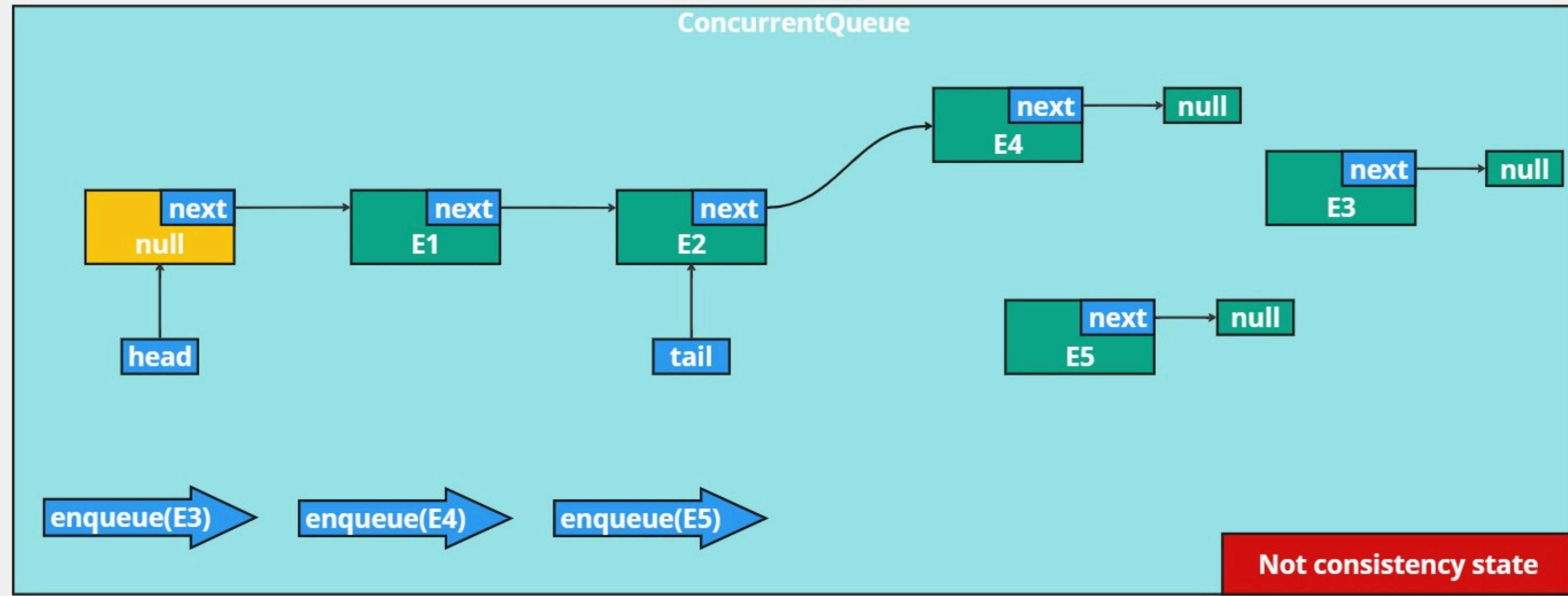
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```



```
enqueue(E3)
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```

```
enqueue(E4)
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```

```
enqueue(E5)
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```



### enqueue(E3)

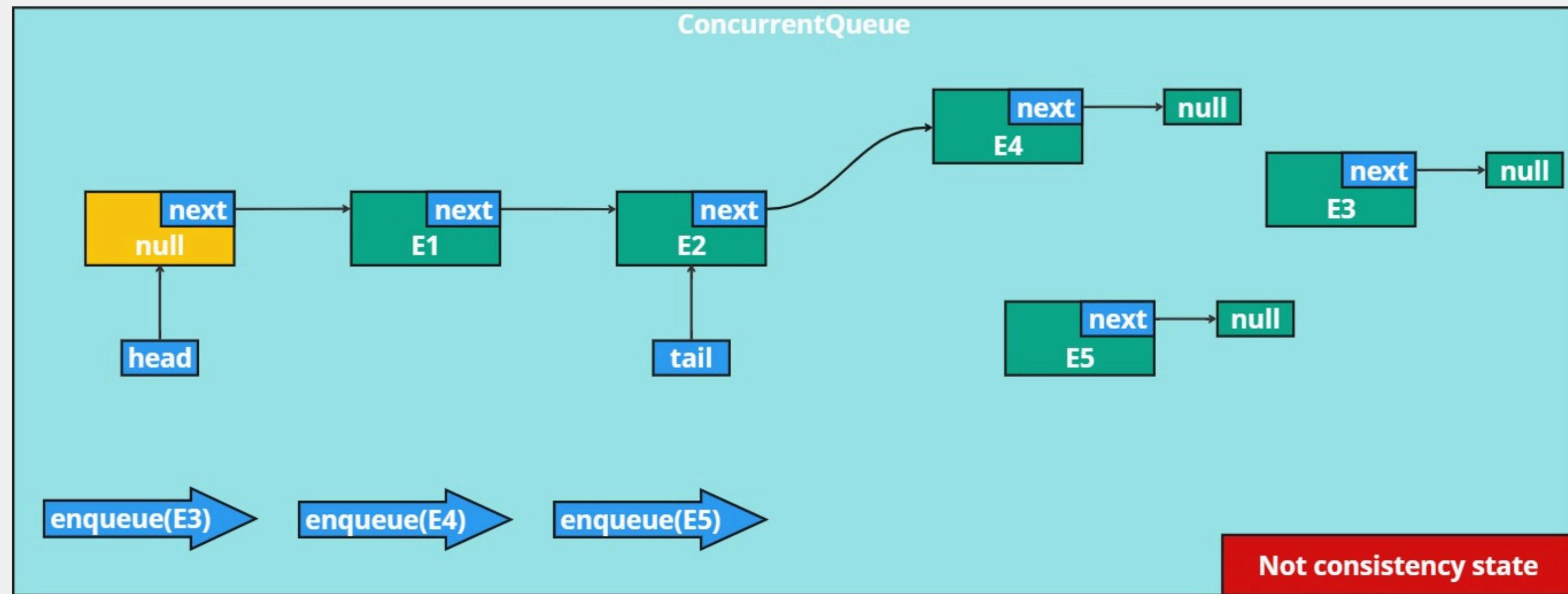
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```

### enqueue(E4)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```

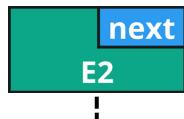
### enqueue(E5)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```



### enqueue(E3)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```



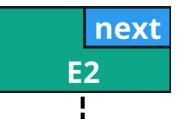
### enqueue(E4)

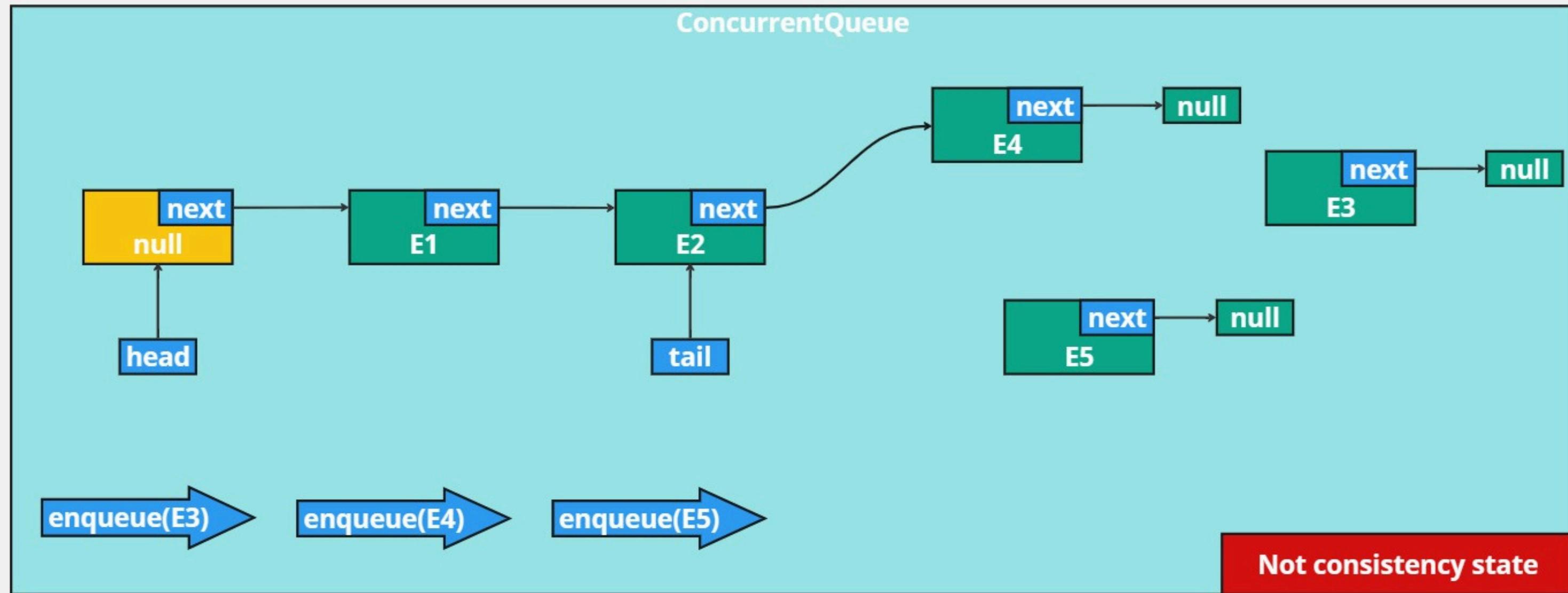
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```



### enqueue(E5)

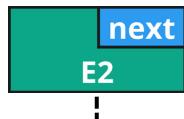
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```





### enqueue(E3)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```



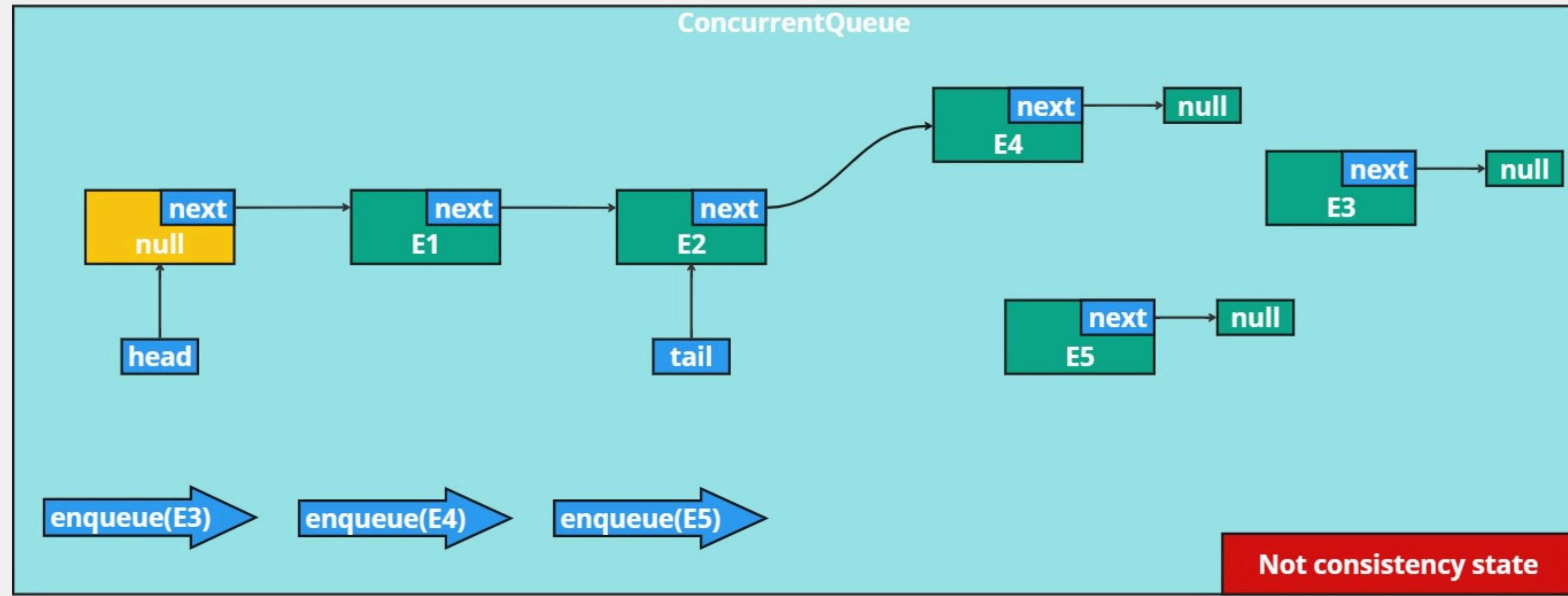
### enqueue(E4)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```



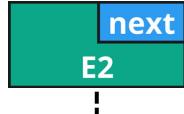
### enqueue(E5)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```



### enqueue(E3)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```



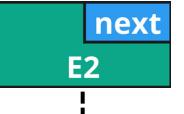
### enqueue(E4)

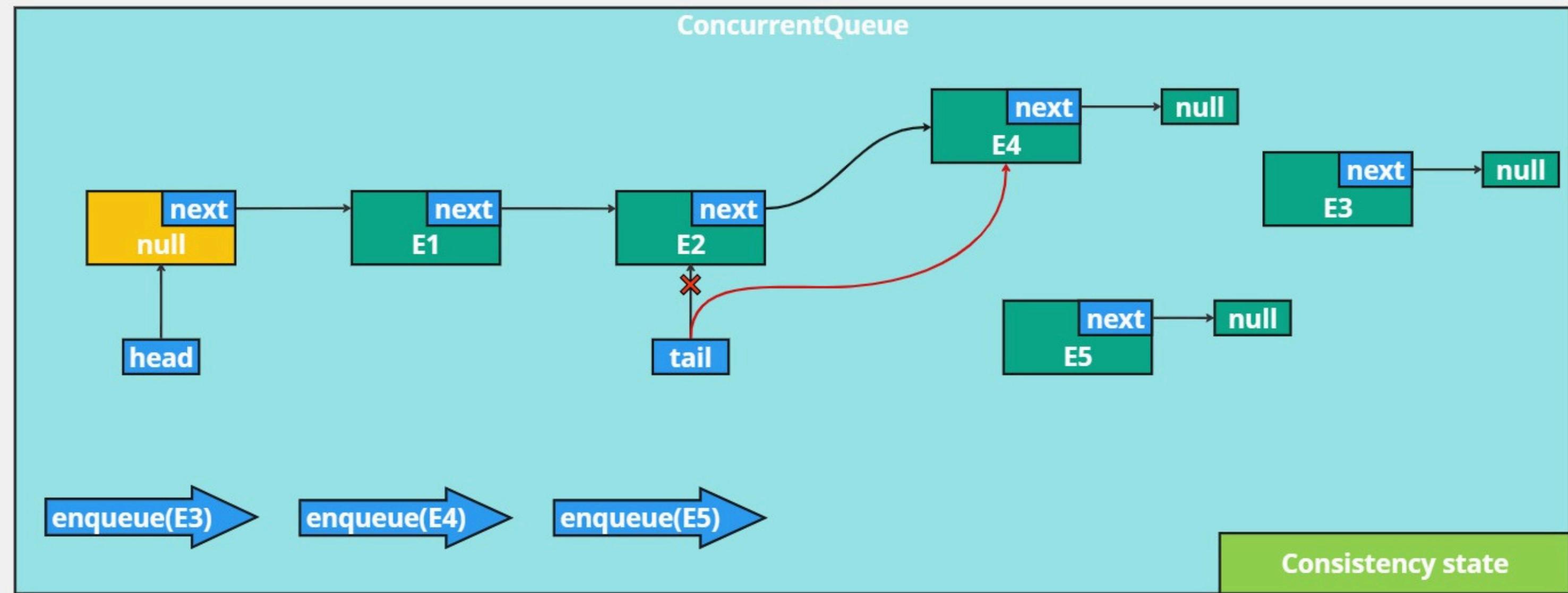
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```



### enqueue(E5)

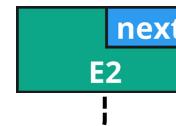
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```





### enqueue(E3)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```



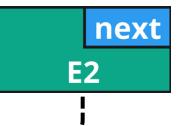
### enqueue(E4)

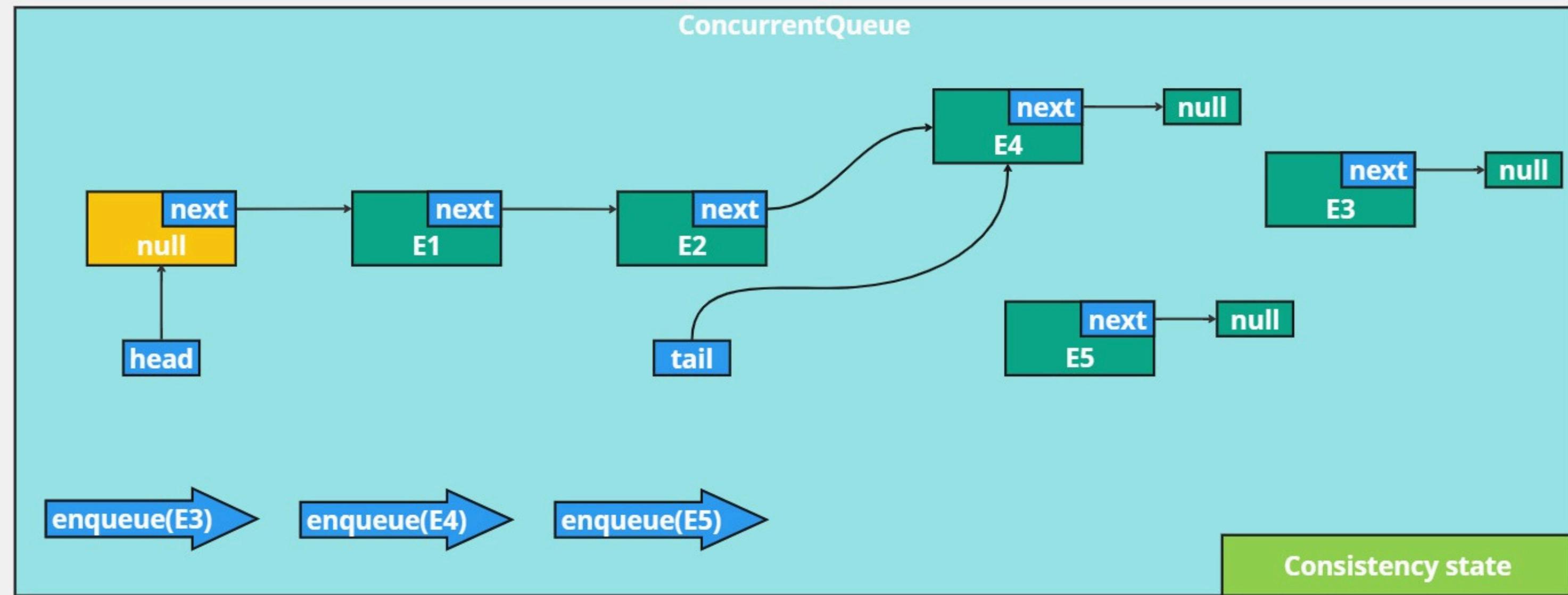
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```



### enqueue(E5)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```





### enqueue(E3)

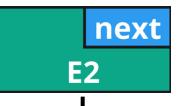
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```

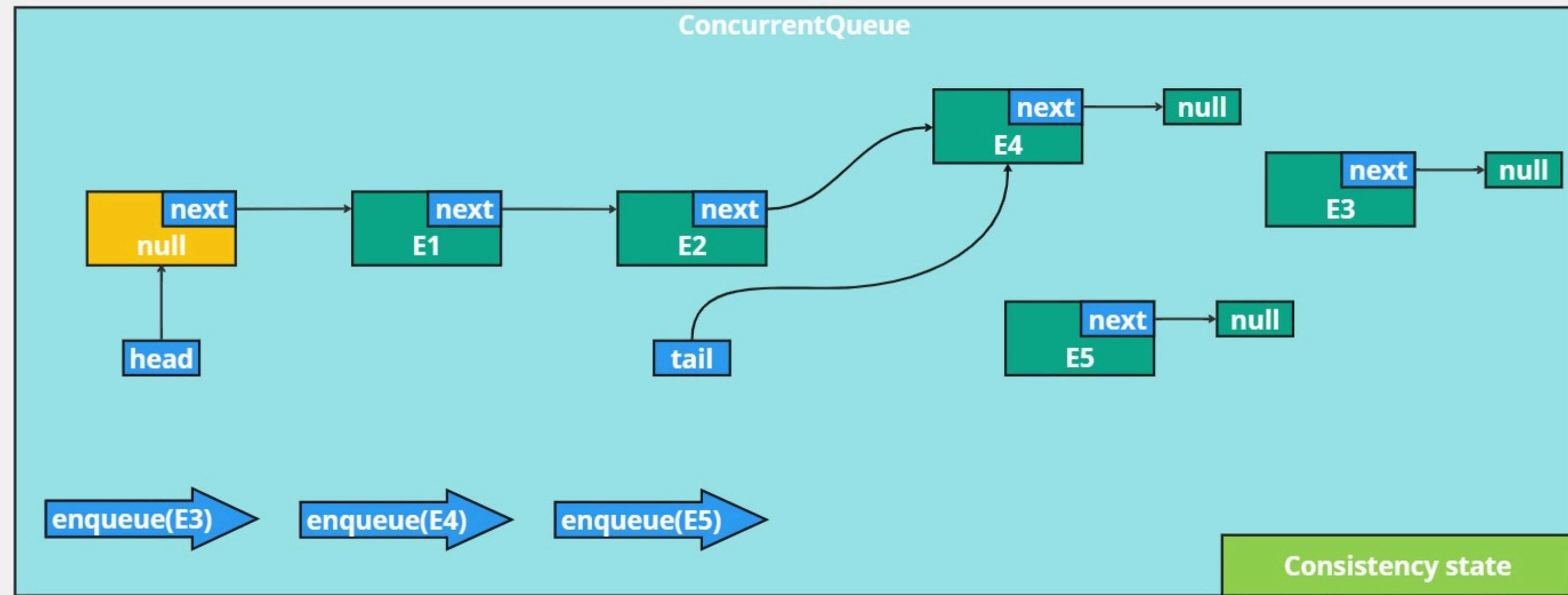
### enqueue(E4)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```

### enqueue(E5)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```





### enqueue(E3)

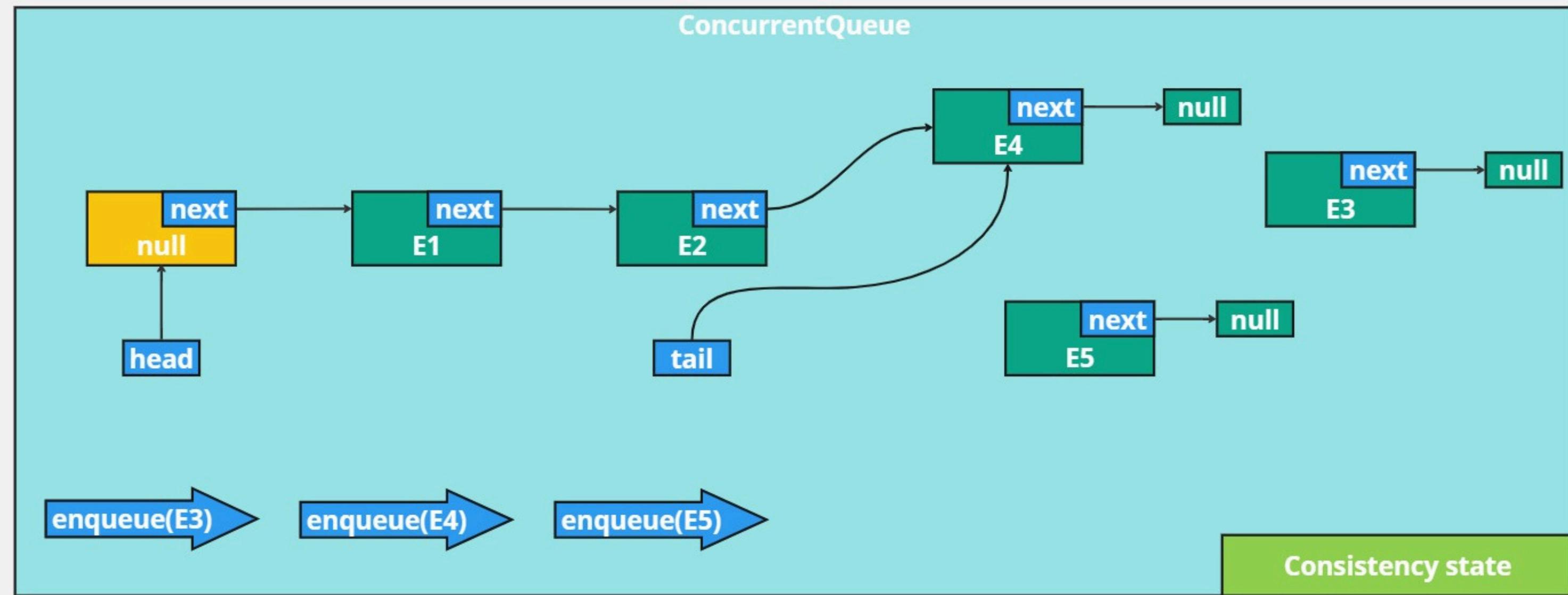
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```

### enqueue(E4)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```

### enqueue(E5)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```



**enqueue(E3)**

```

public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}

```

**enqueue(E4)**

```

public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}

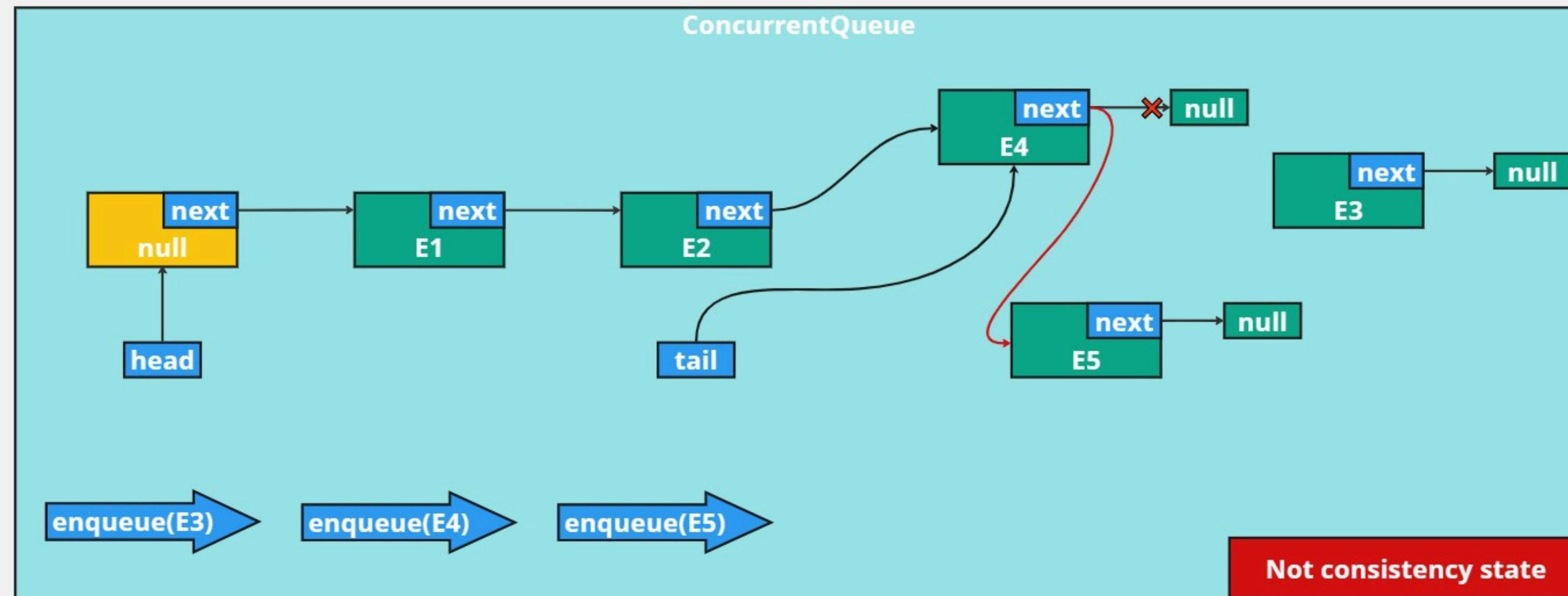
```

**enqueue(E5)**

```

public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}

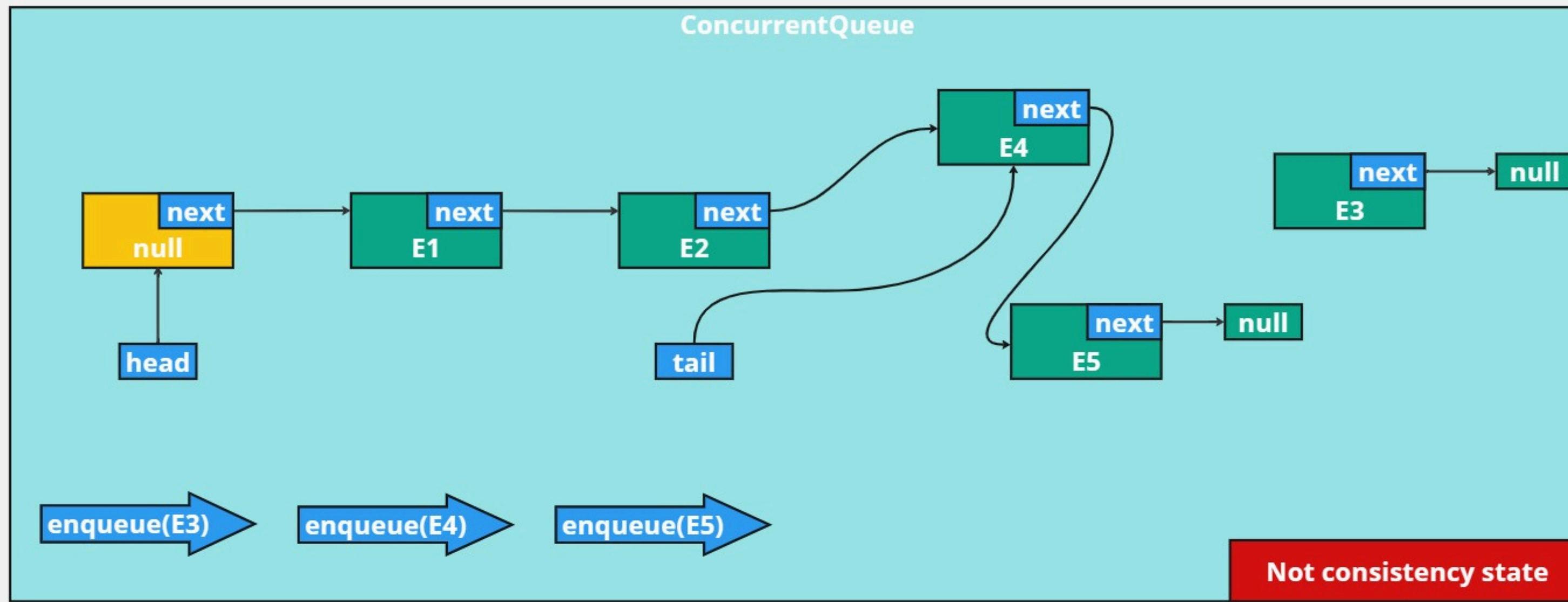
```



```
enqueue(E3)
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```

```
enqueue(E4)
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```

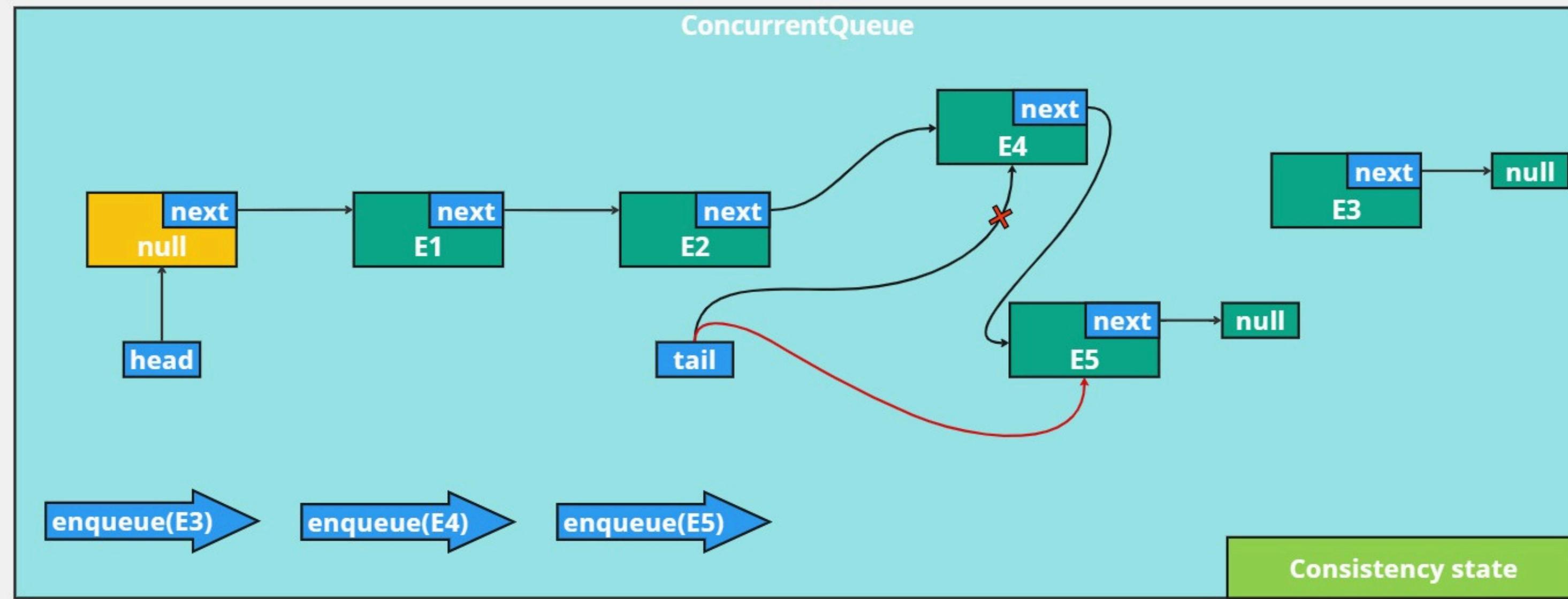
```
enqueue(E5)
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```



```
enqueue(E3)
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```

```
enqueue(E4)
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```

```
enqueue(E5)
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```



#### enqueue(E3)

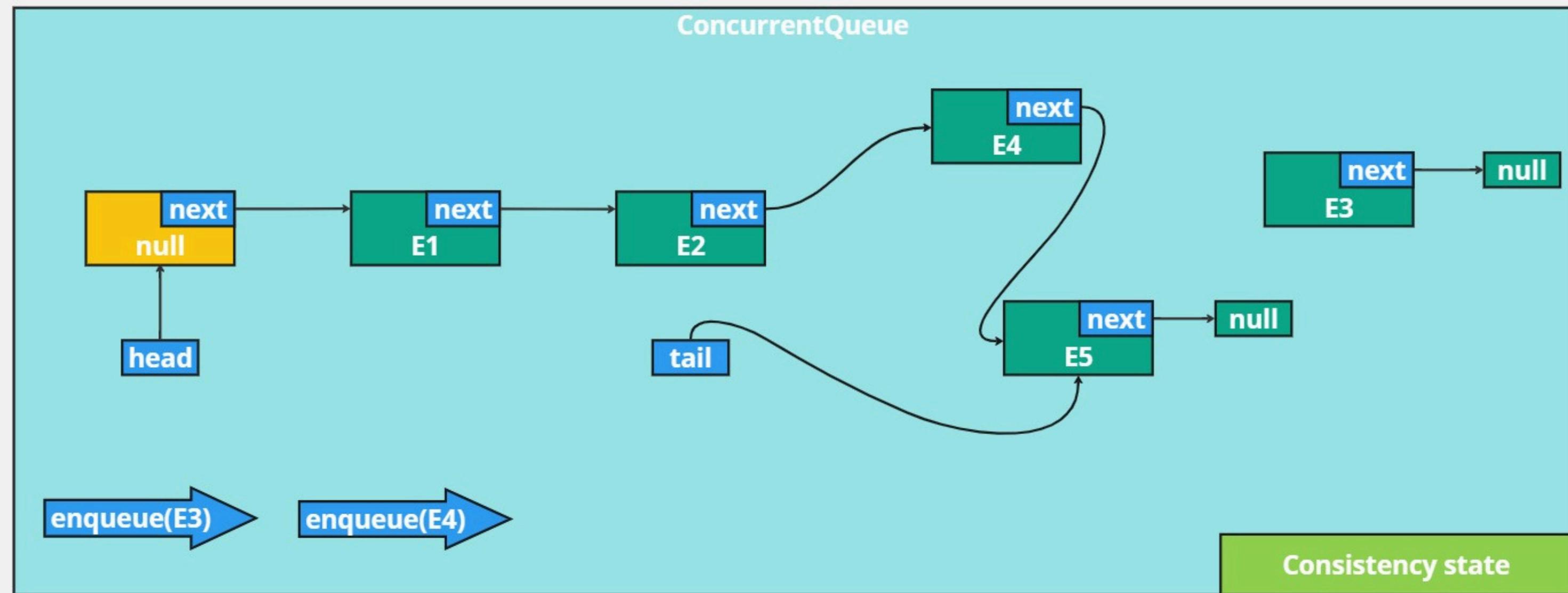
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```

#### enqueue(E4)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```

#### enqueue(E5)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```



### enqueue(E3)

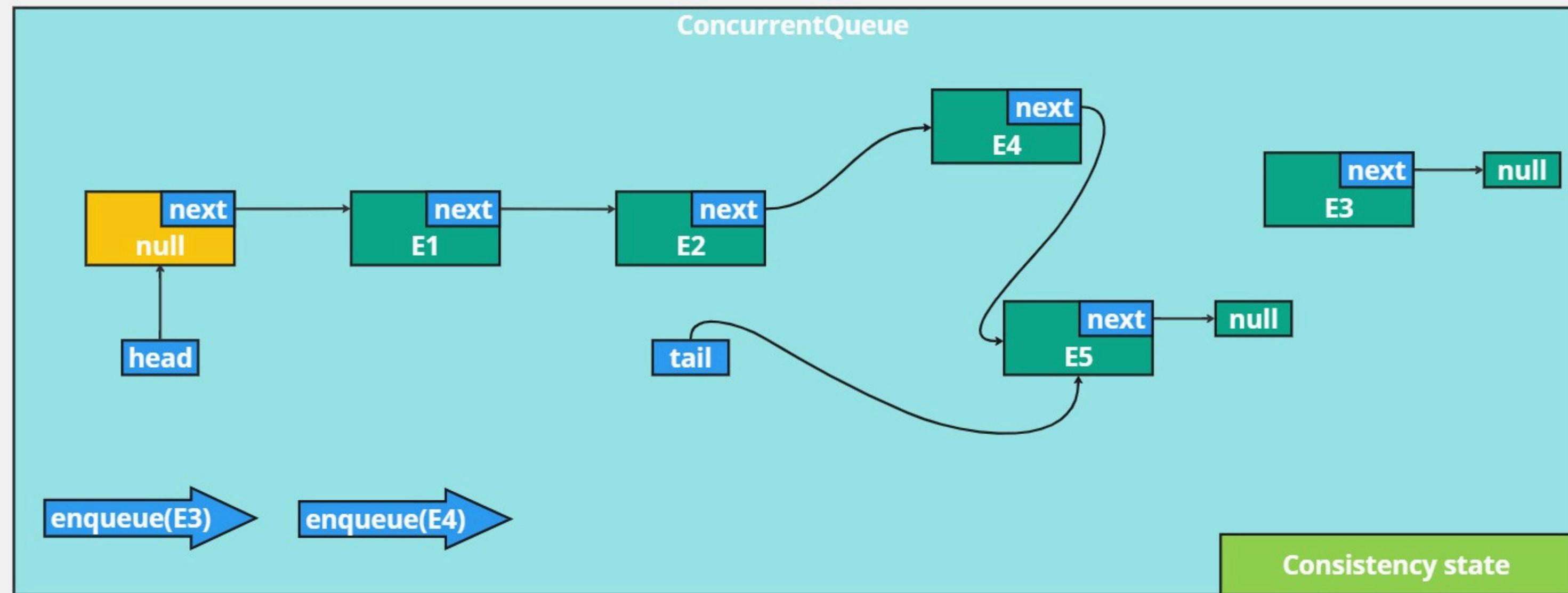
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```

### enqueue(E4)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```

### enqueue(E5)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```



**enqueue(E3)**

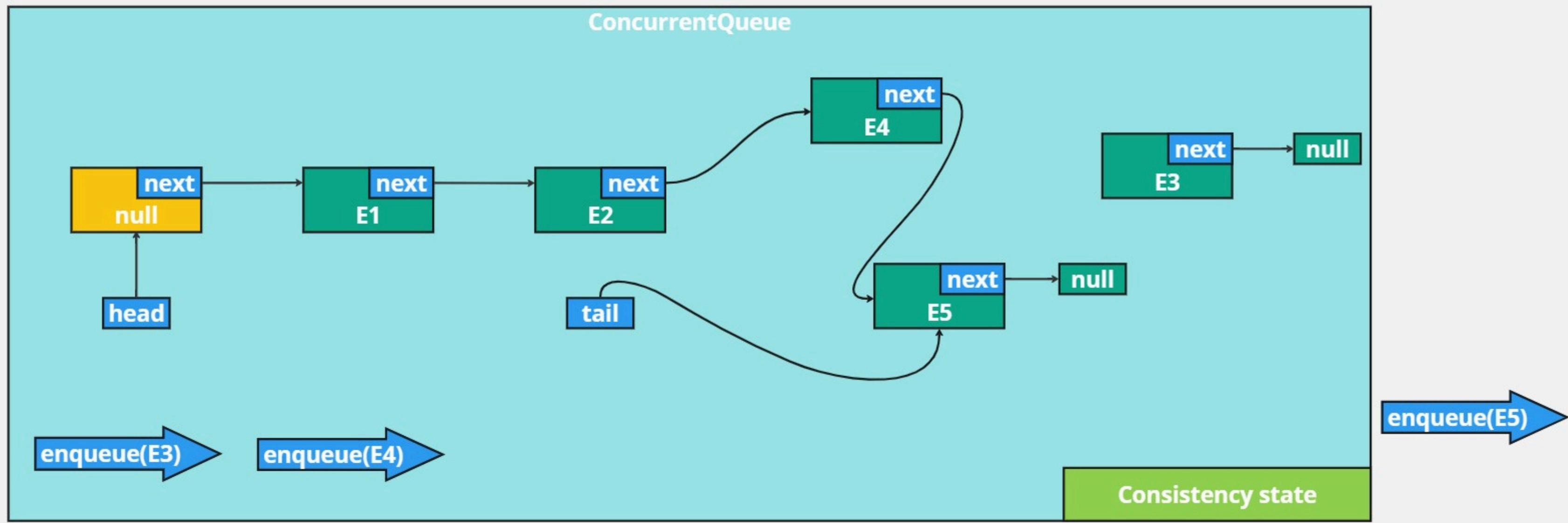
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```

**enqueue(E4)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```

**enqueue(E5)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```



**enqueue(E3)**

```

public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}

```

**enqueue(E4)**

```

public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}

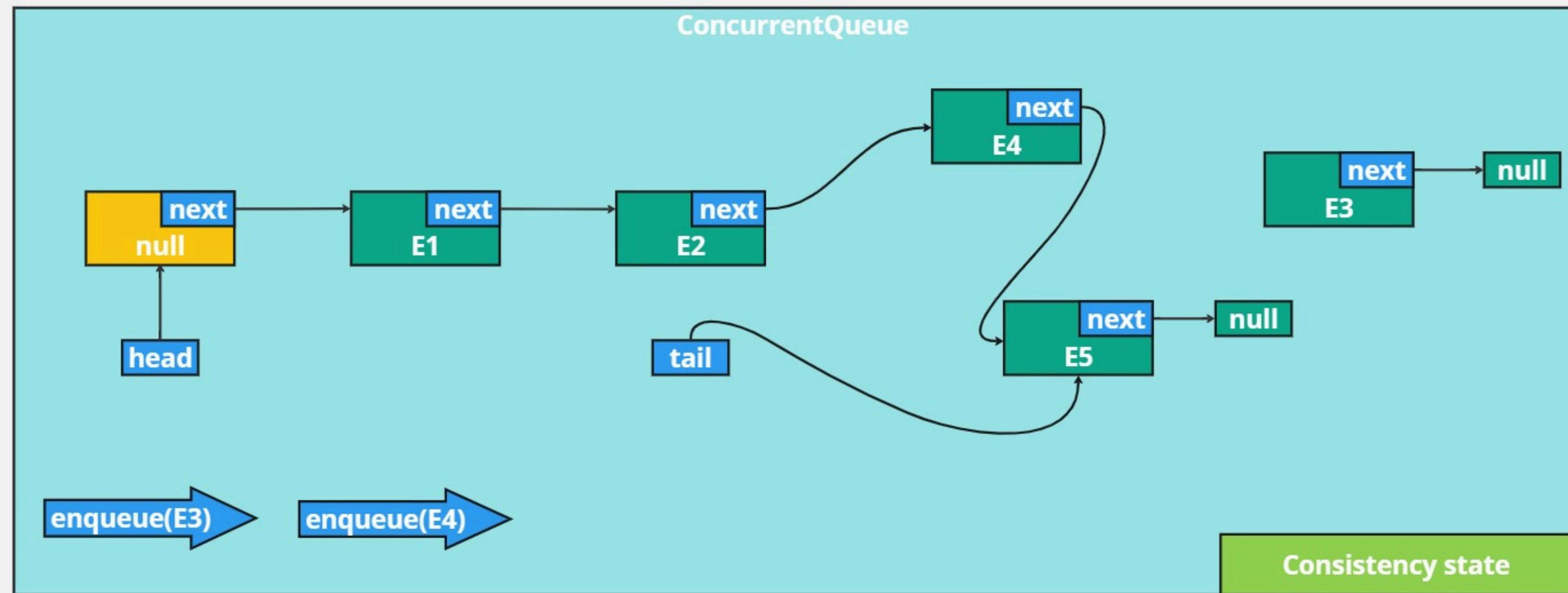
```

**enqueue(E5)**

```

public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}

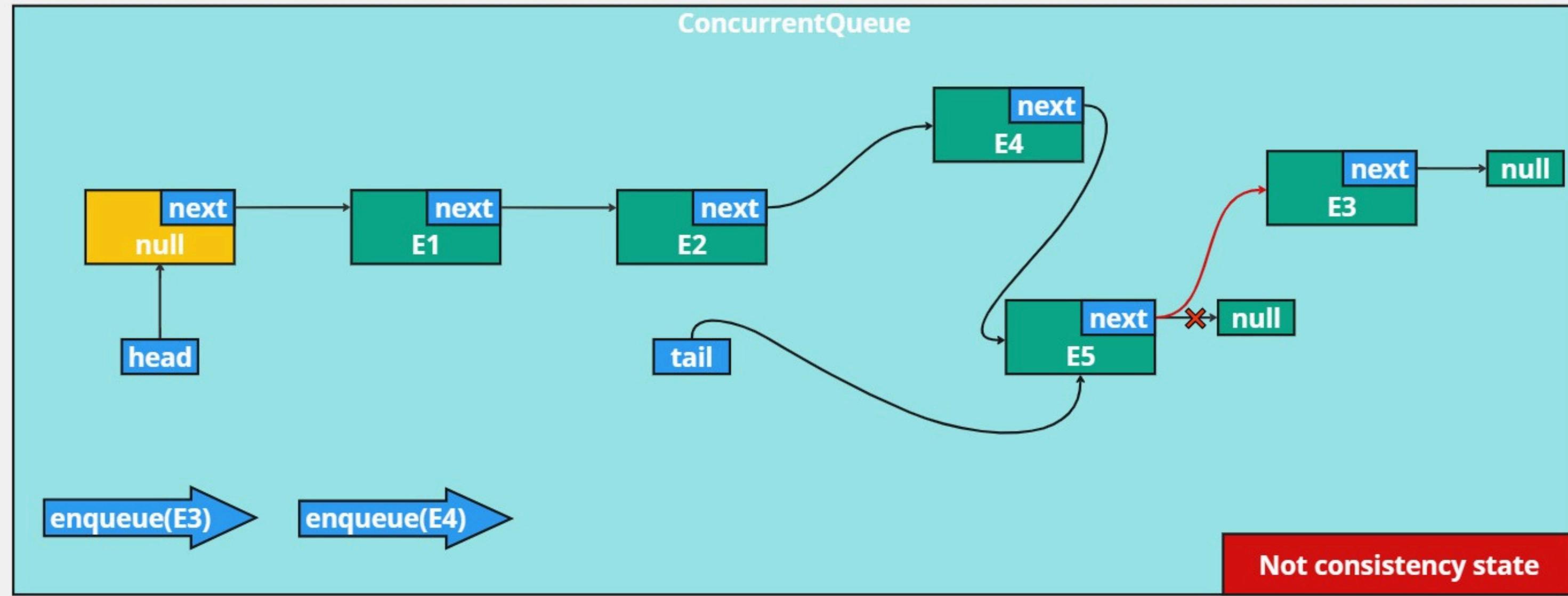
```



```
enqueue(E3)
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```

```
enqueue(E4)
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```

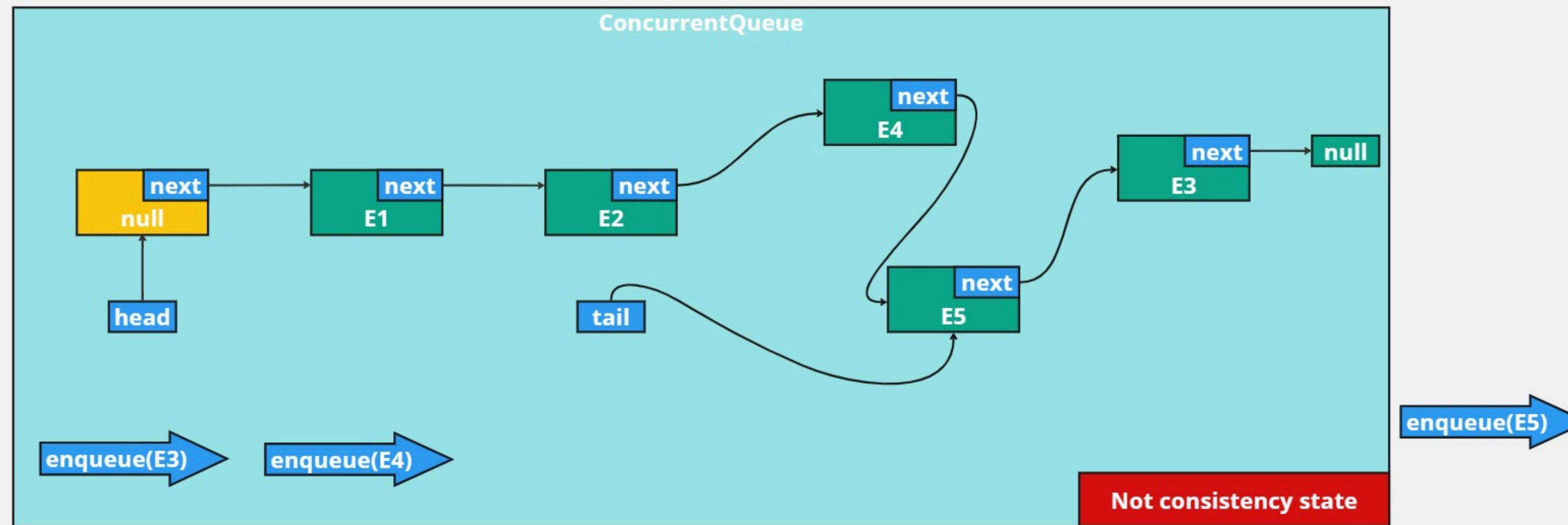
```
enqueue(E5)
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```



```
enqueue(E3)
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```

```
enqueue(E4)
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```

```
enqueue(E5)
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```



### enqueue(E3)

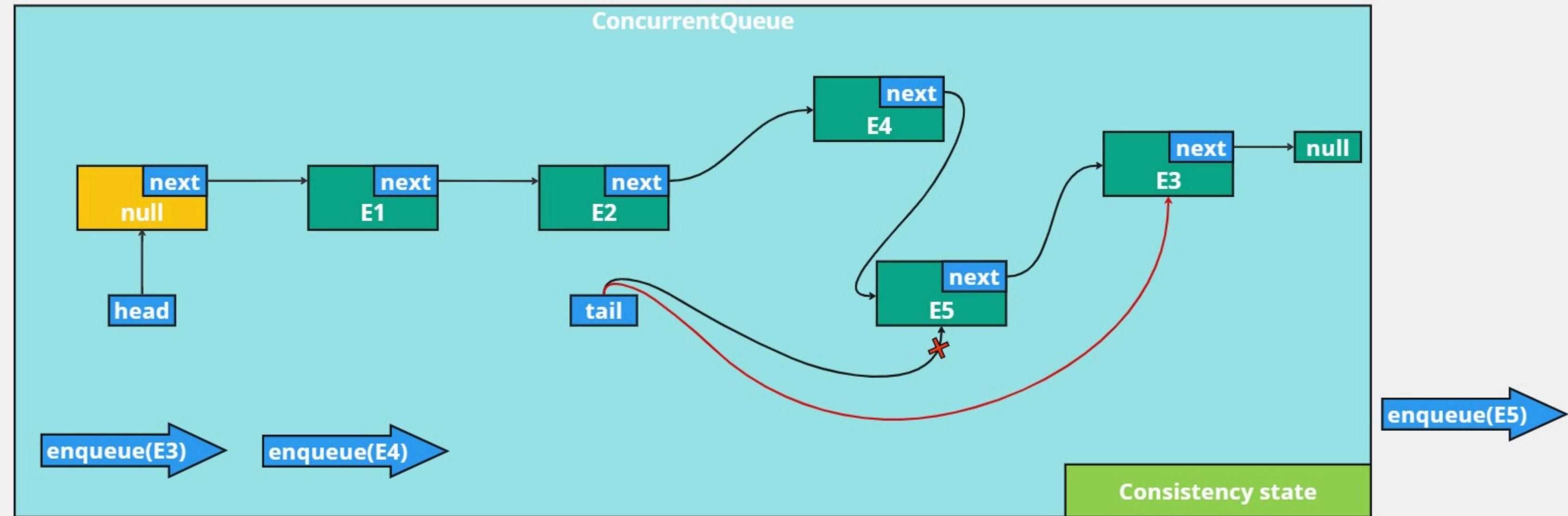
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```

### enqueue(E4)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```

### enqueue(E5)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```



### enqueue(E3)

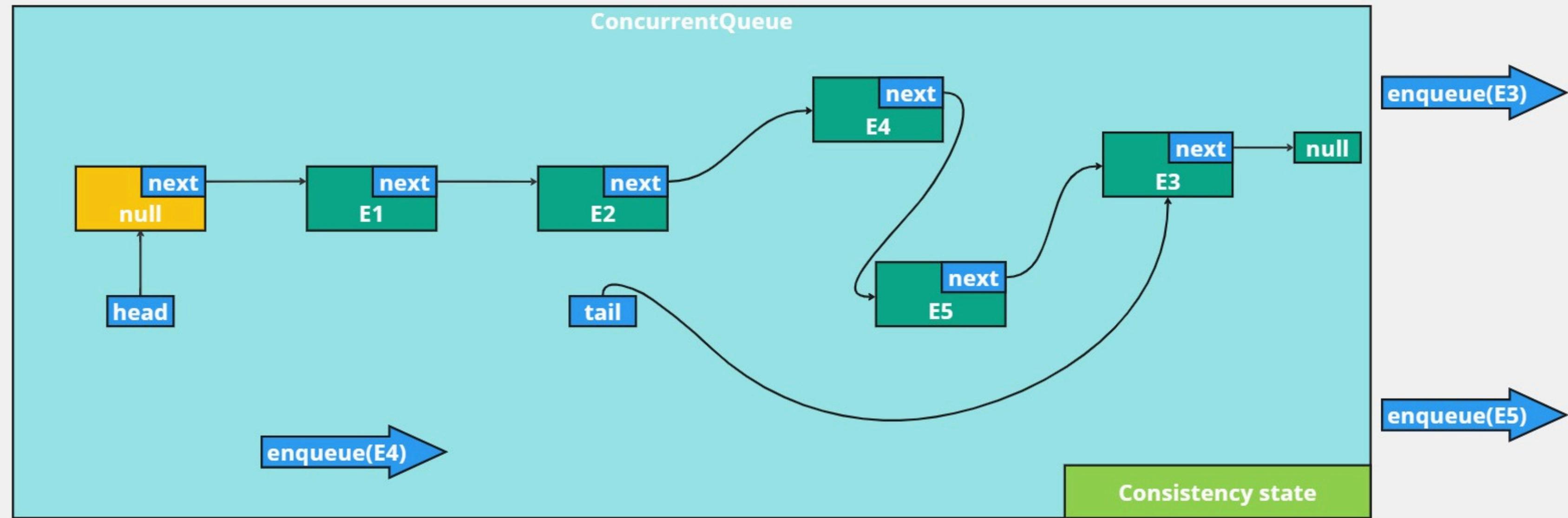
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```

### enqueue(E4)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```

### enqueue(E5)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```



### enqueue(E3)

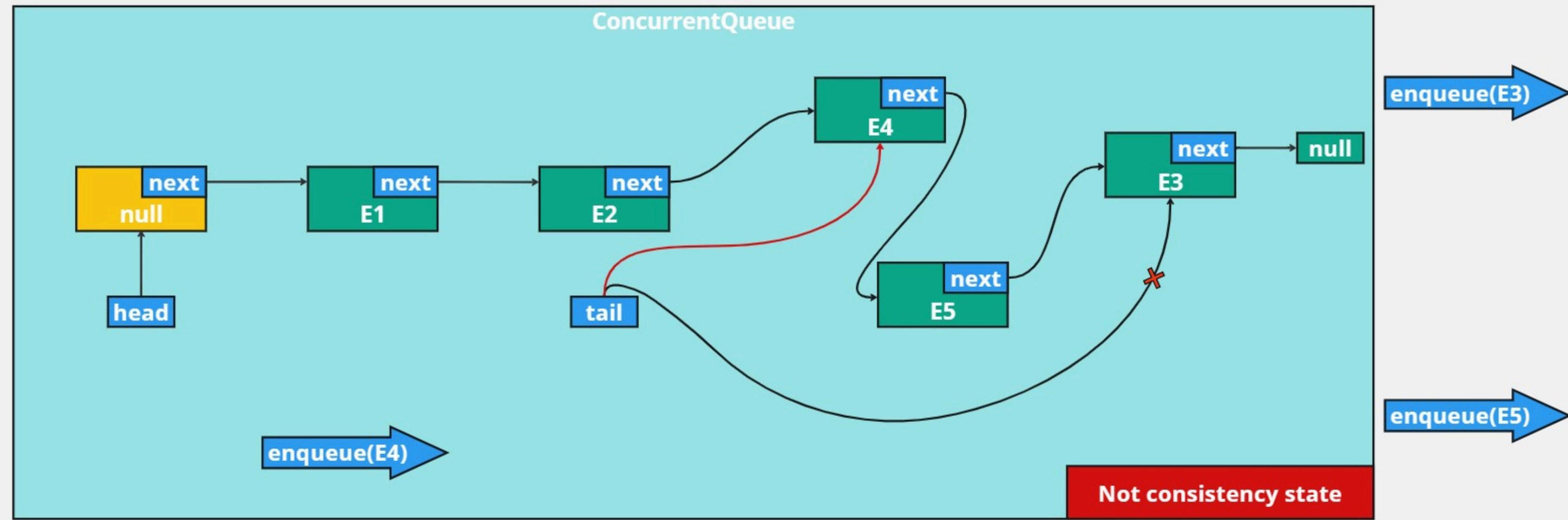
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```

### enqueue(E4)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```

### enqueue(E5)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```



### enqueue(E3)

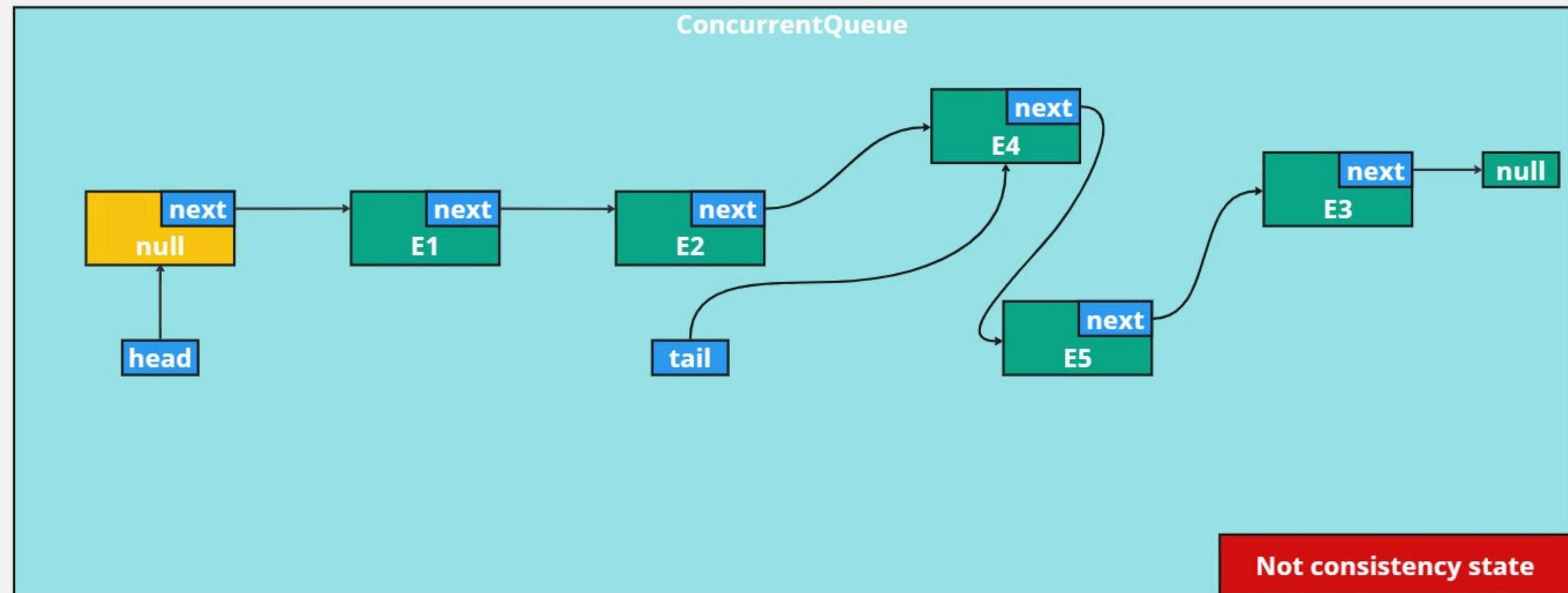
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```

### enqueue(E4)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```

### enqueue(E5)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```



**enqueue(E3)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```

**enqueue(E4)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```

**enqueue(E5)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            break;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
    tail.set(newNode);
}
```

```
public void enqueue(final E element) {  
    final Node<E> newNode = new Node<>(element);  
    while (true) {  
        final Node<E> previousTail = tail.get();  
        if (previousTail.next.compareAndSet(null, newNode)) {  
            break;  
        } else {  
            tail.compareAndSet(previousTail, previousTail.next.get());  
        }  
    }  
tail.set(newNode);  
}
```



```
public void enqueue(final E element) {  
    final Node<E> newNode = new Node<>(element);  
    while (true) {  
        final Node<E> previousTail = tail.get();  
        if (previousTail.next.compareAndSet(null, newNode)) {  
            if (tail.get() == previousTail) {  
                tail.set(newNode);  
            }  
            return;  
        } else {  
            tail.compareAndSet(previousTail, previousTail.next.get());  
        }  
    }  
}
```

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            if (tail.get() == previousTail) {
                tail.set(newNode);
            }
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

**not atomic**

```
public void enqueue(final E element) {  
    final Node<E> newNode = new Node<>(element);  
    while (true) {  
        final Node<E> previousTail = tail.get();  
        if (previousTail.next.compareAndSet(null, newNode)) {  
            if (tail.get() == previousTail) {  
                tail.set(newNode);  
            }  
            return;  
        } else {  
            tail.compareAndSet(previousTail, previousTail.next.get());  
        }  
    }  
}
```

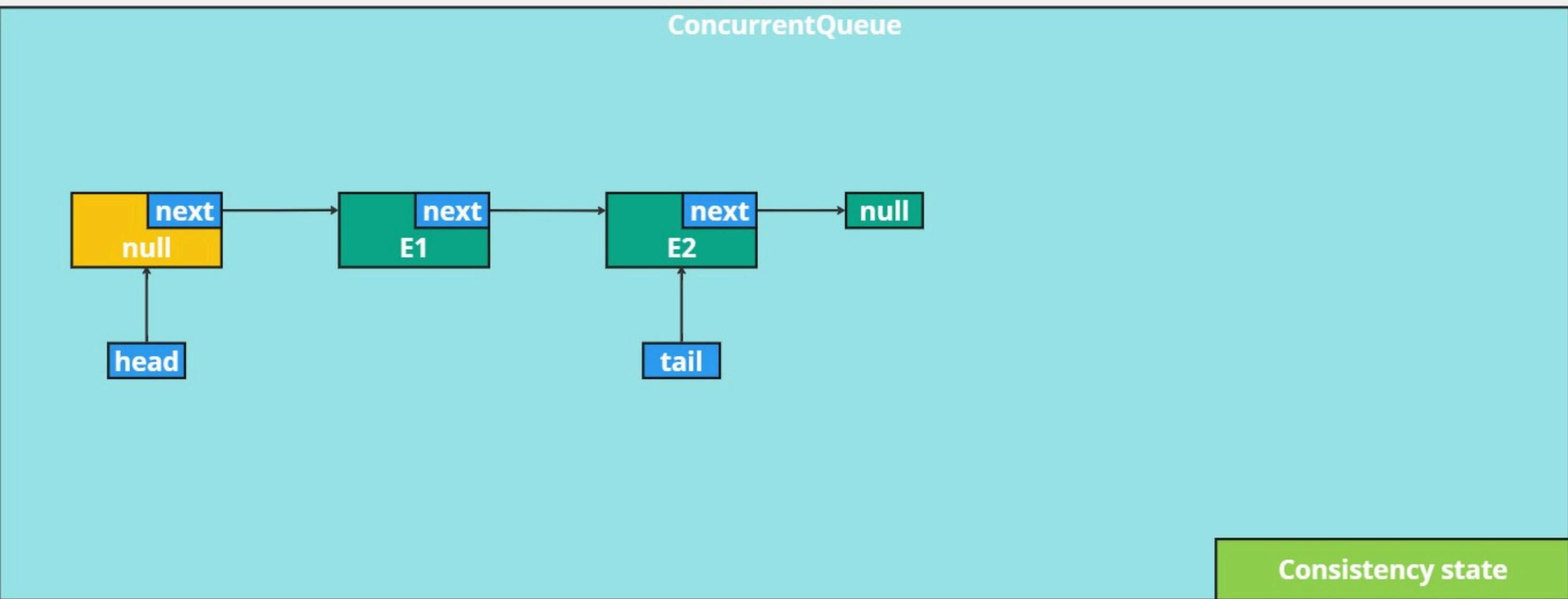


```
public void enqueue(final E element) {  
    final Node<E> newNode = new Node<>(element);  
    while (true) {  
        final Node<E> previousTail = tail.get();  
        if (previousTail.next.compareAndSet(null, newNode)) {  
            tail.compareAndSet(previousTail, newNode);  
            return;  
        } else {  
            tail.compareAndSet(previousTail, previousTail.next.get());  
        }  
    }  
}
```

atomic

## ConcurrentQueue

enqueue(E3) →  
 enqueue(E4) →  
 enqueue(E5) →



### enqueue(E3)

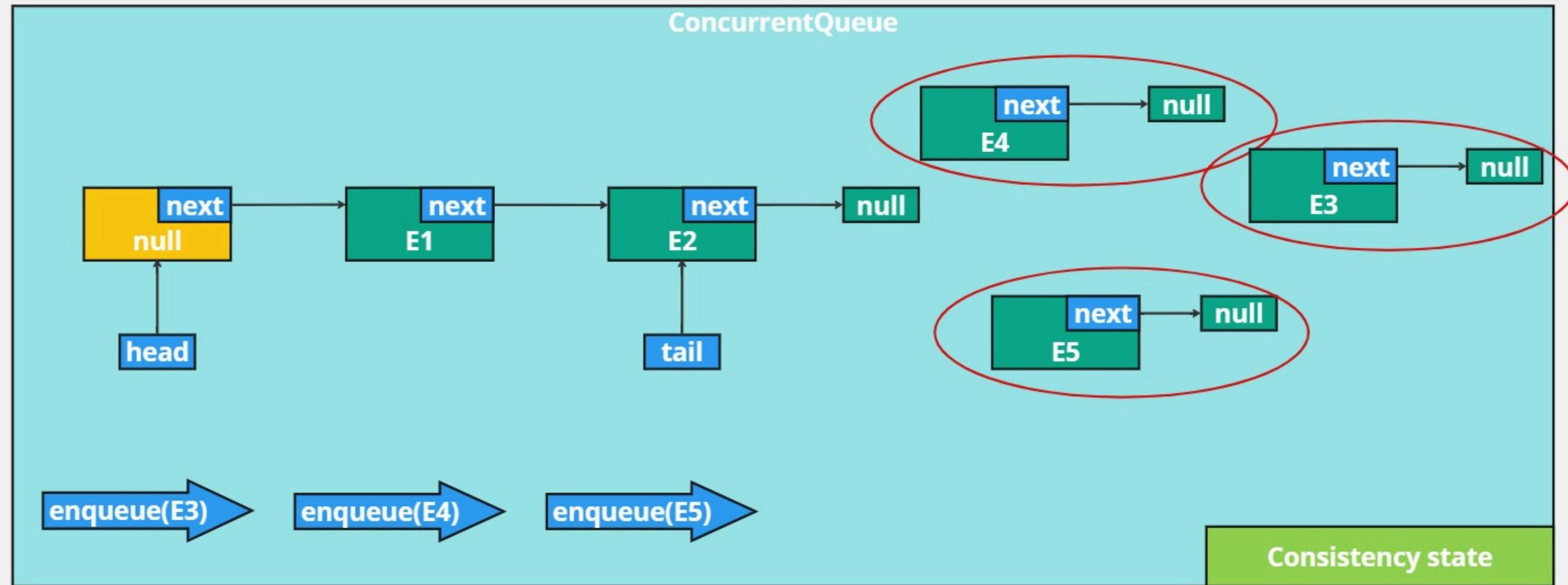
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

### enqueue(E4)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

### enqueue(E5)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```



```

public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}

```

```

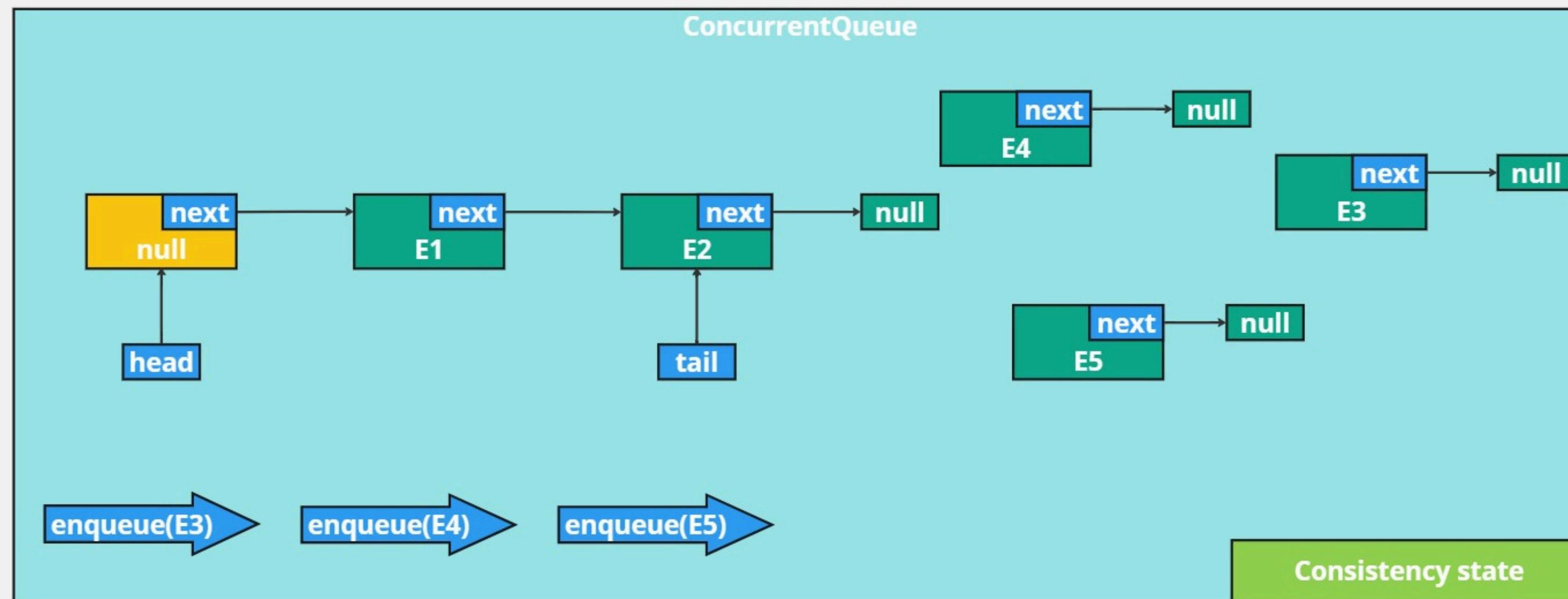
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}

```

```

public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}

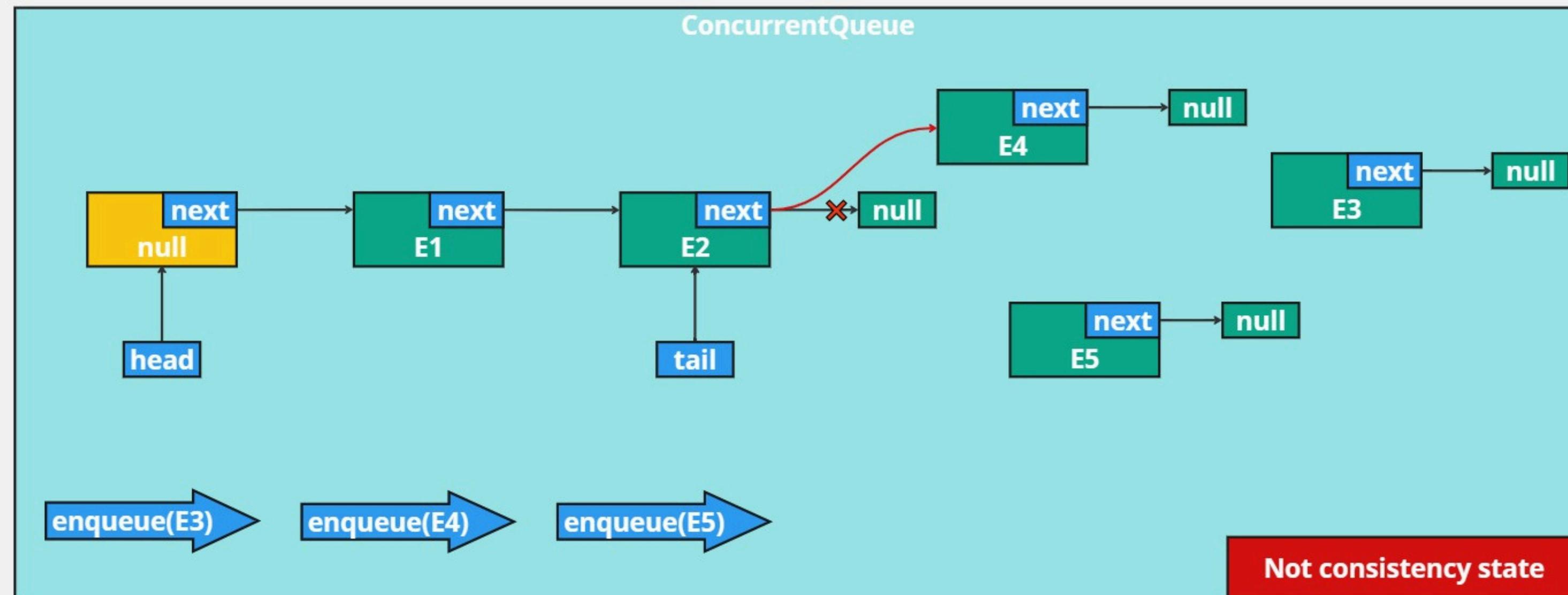
```



```
enqueue(E3)
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

```
enqueue(E4)
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

```
enqueue(E5)
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```



**enqueue(E3)**

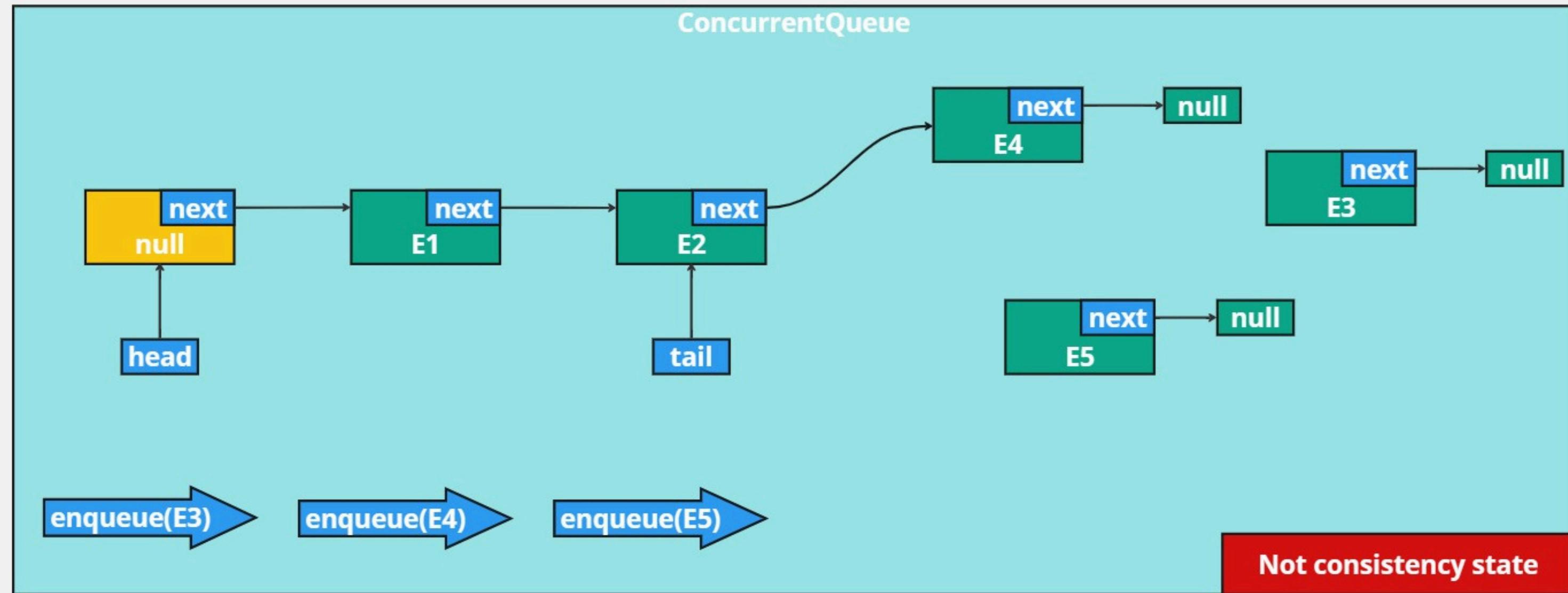
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

**enqueue(E4)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

**enqueue(E5)**

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```



**enqueue(E3)**

```

public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
  
```

**enqueue(E4)**

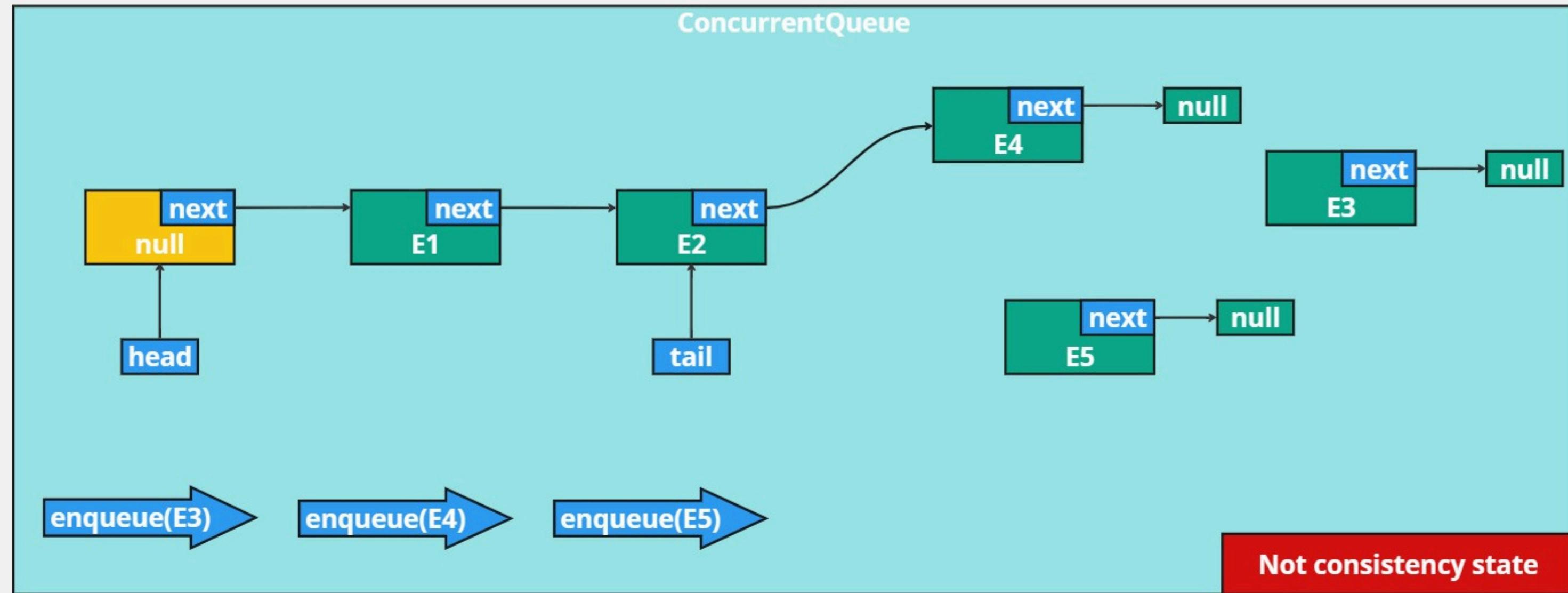
```

public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
  
```

**enqueue(E5)**

```

public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
  
```



### enqueue(E3)

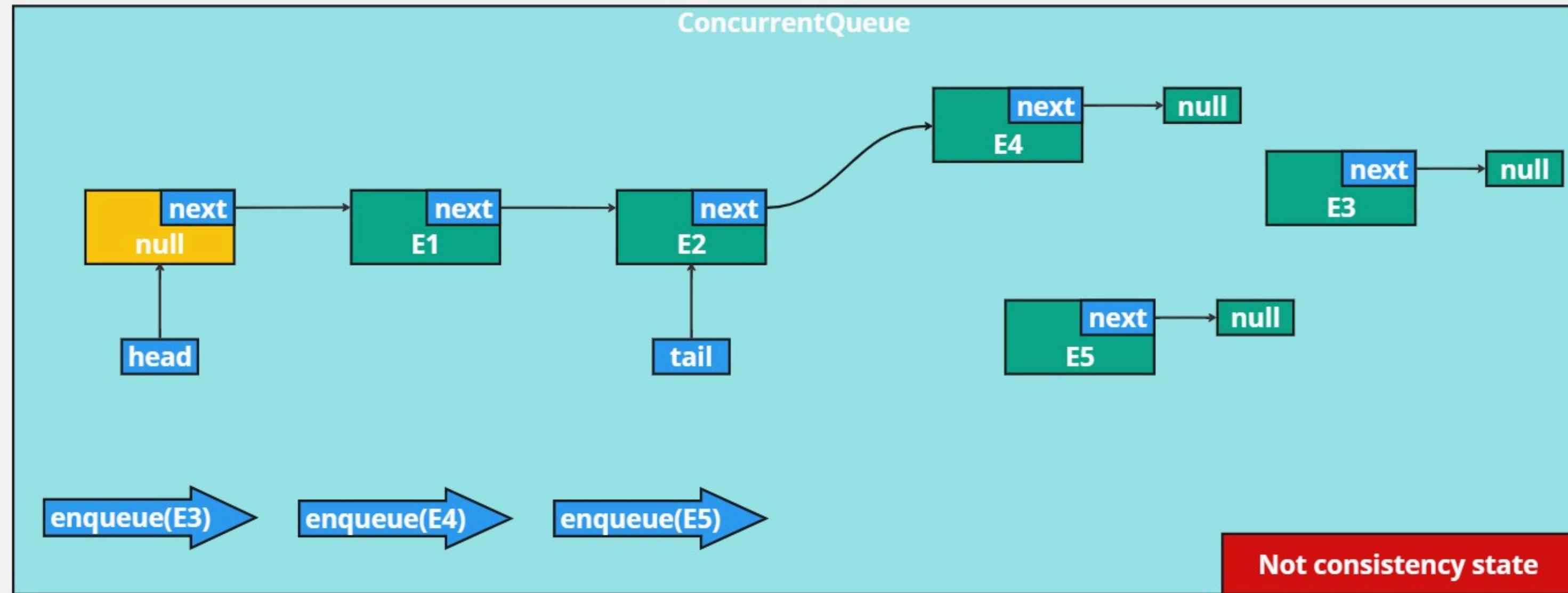
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

### enqueue(E4)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

### enqueue(E5)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```



### enqueue(E3)

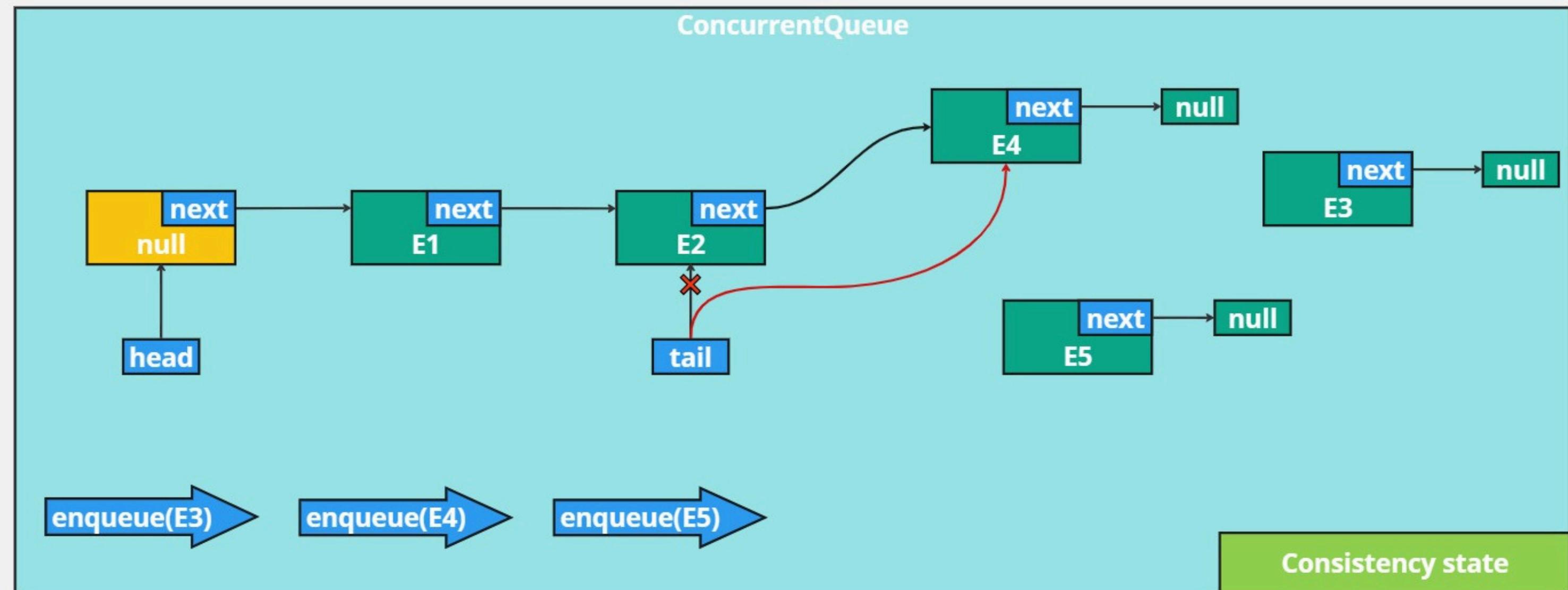
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    final Node<E> previousTail = tail.get();
    while (true) {
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

### enqueue(E4)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    final Node<E> previousTail = tail.get();
    while (true) {
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

### enqueue(E5)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    final Node<E> previousTail = tail.get();
    while (true) {
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```



### enqueue(E3)

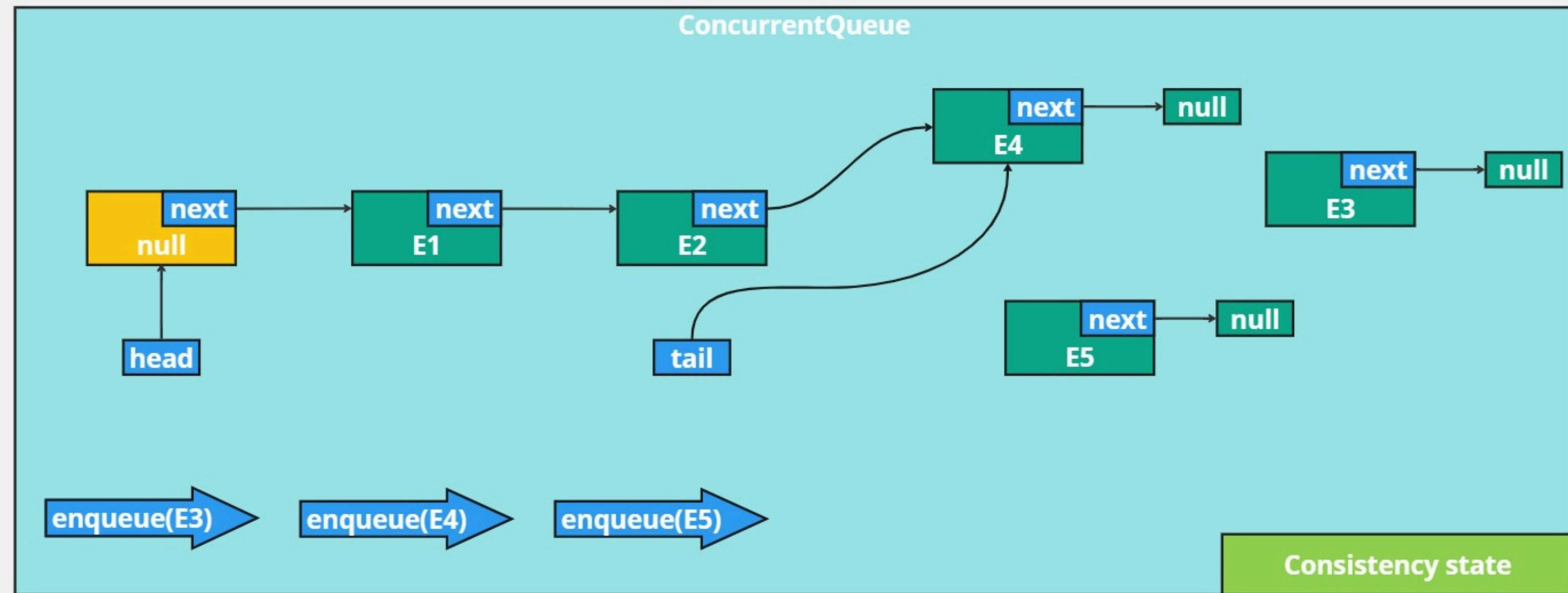
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

### enqueue(E4)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

### enqueue(E5)

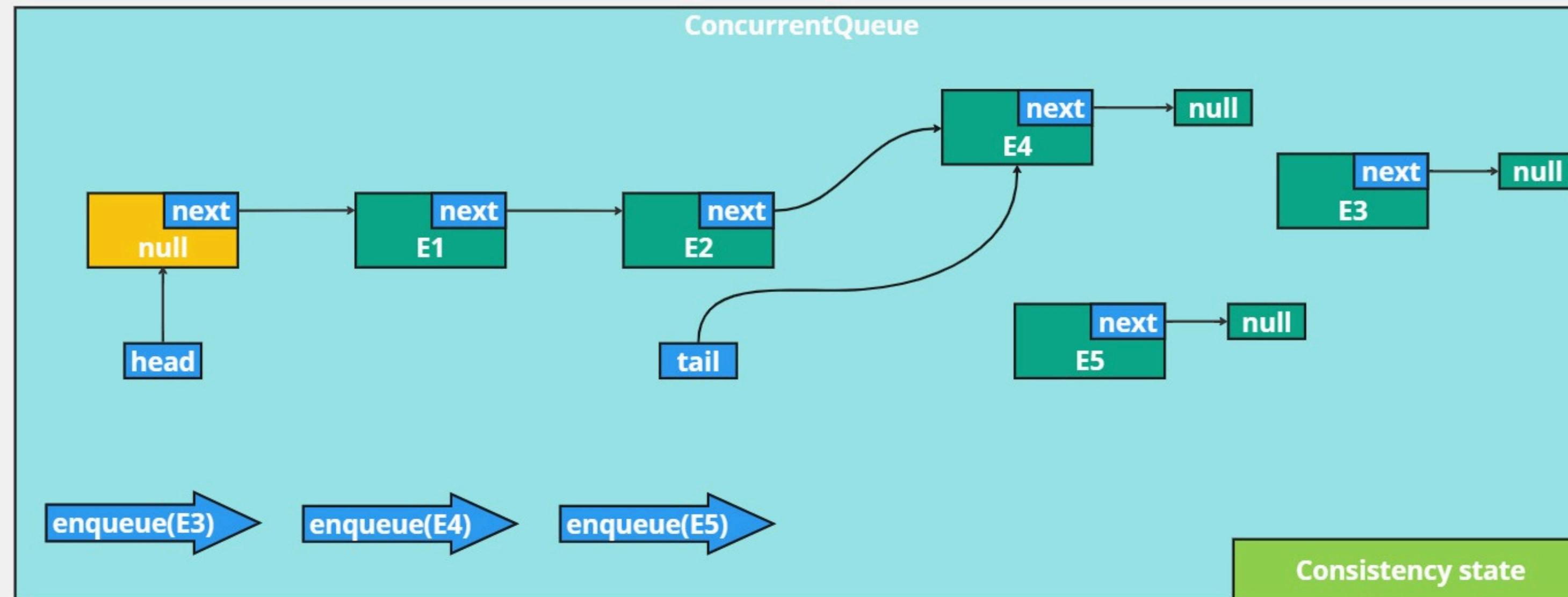
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```



```
enqueue(E3)
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

```
enqueue(E4)
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

```
enqueue(E5)
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```



### enqueue(E3)

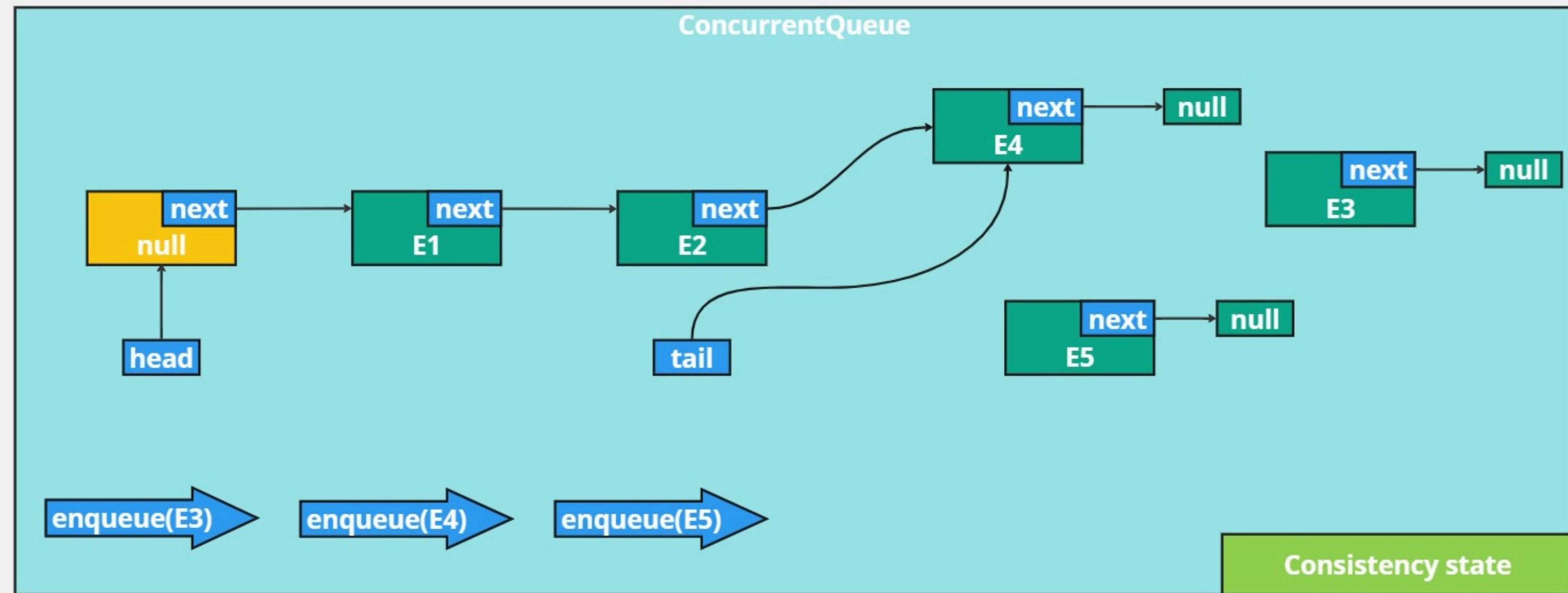
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

### enqueue(E4)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

### enqueue(E5)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```



### enqueue(E3)

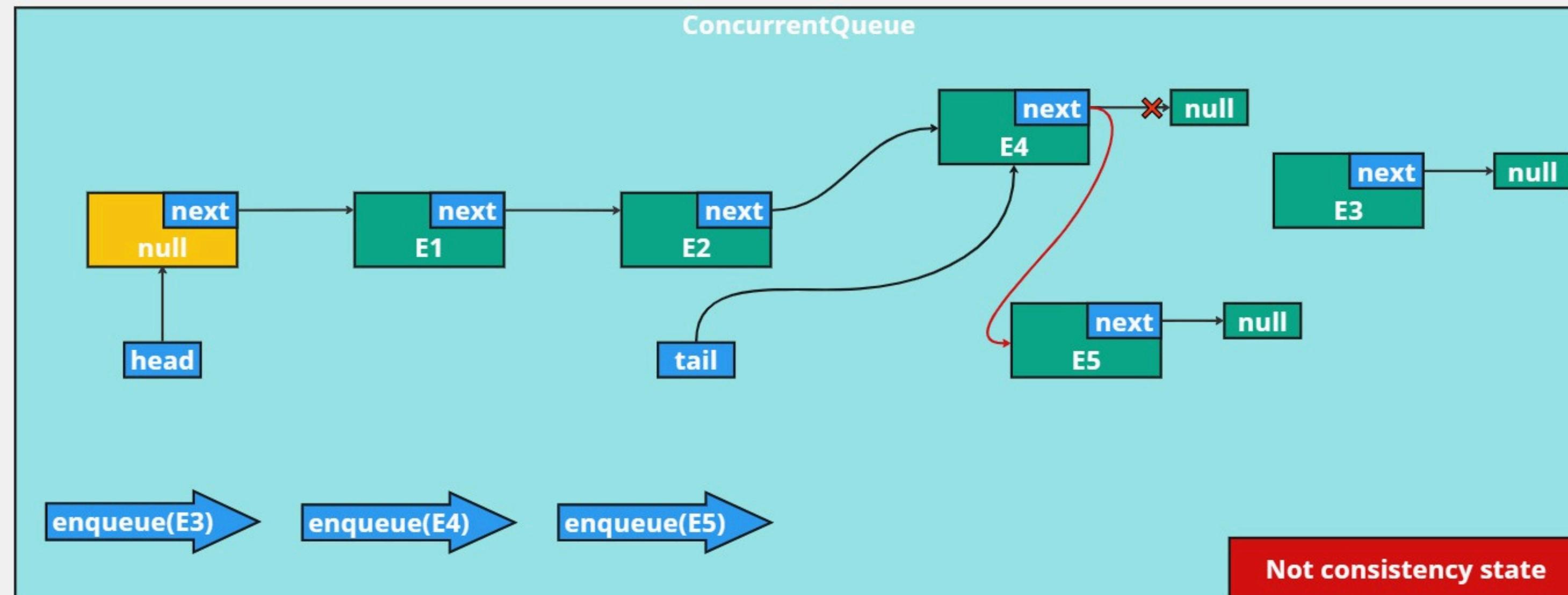
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

### enqueue(E4)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

### enqueue(E5)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```



### enqueue(E3)

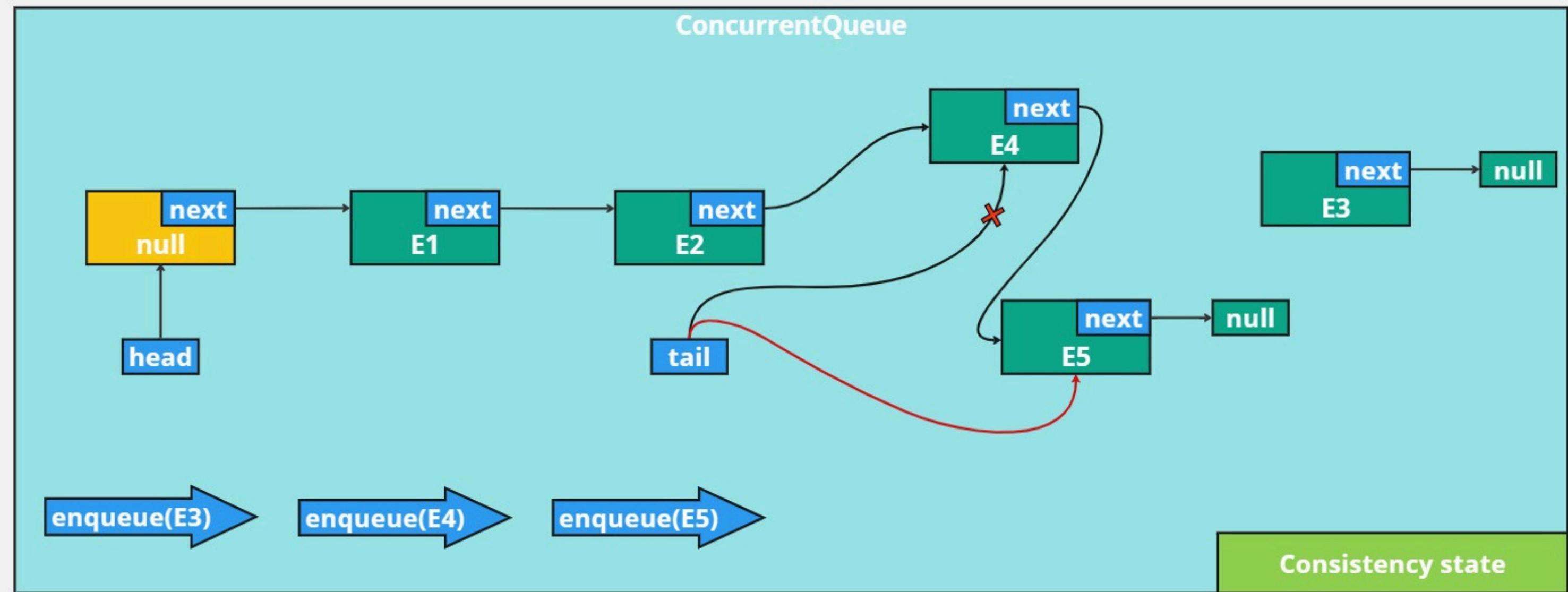
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

### enqueue(E4)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

### enqueue(E5)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```



### enqueue(E3)

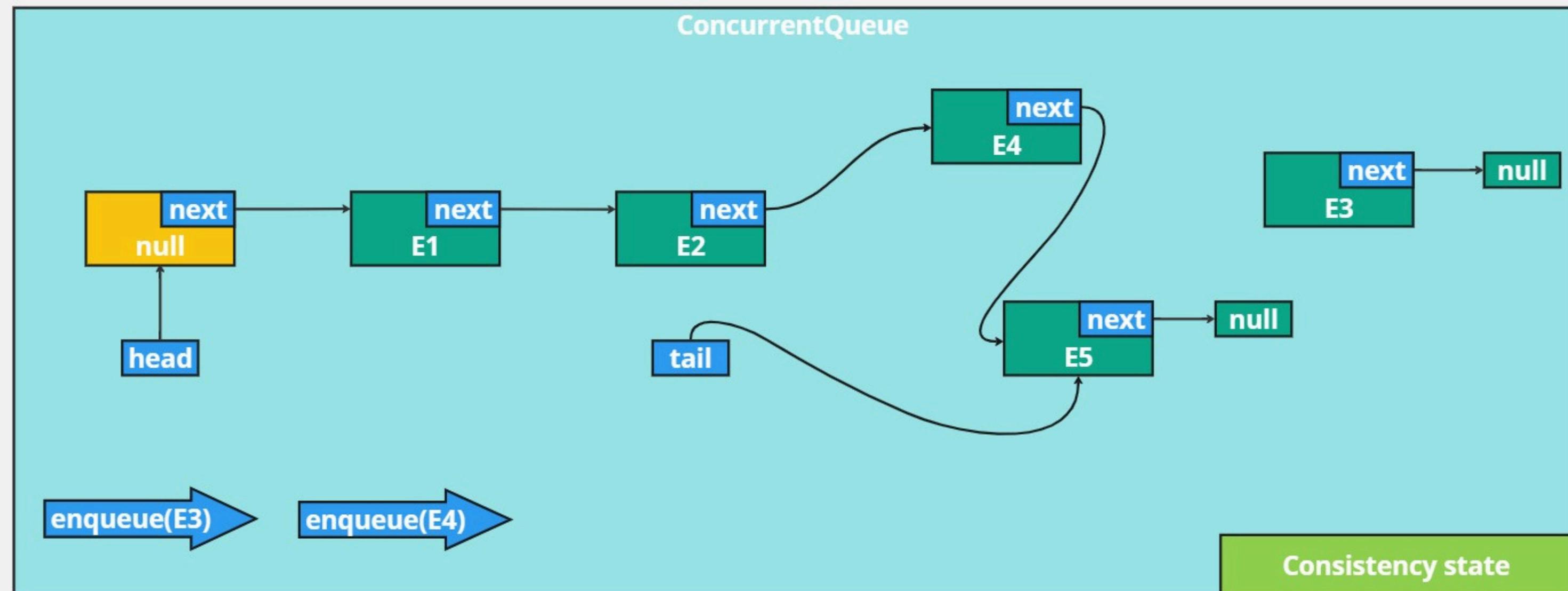
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

### enqueue(E4)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

### enqueue(E5)

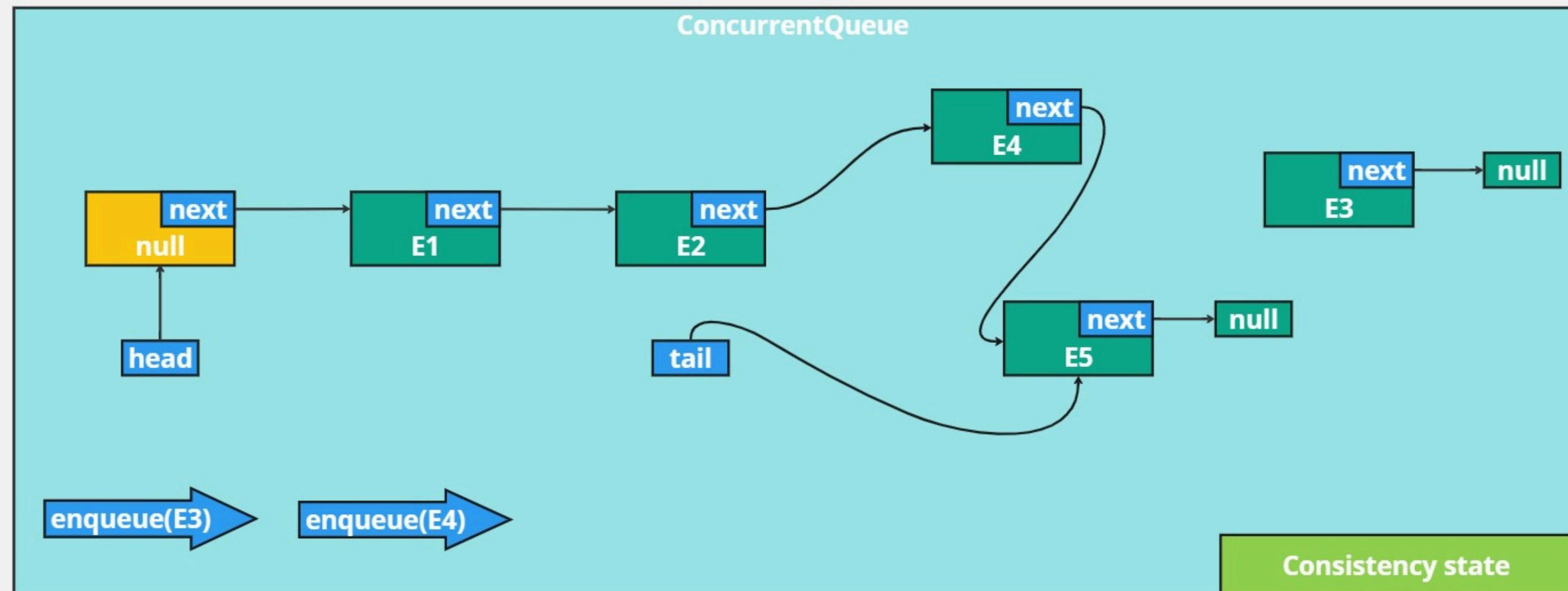
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```



```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```



### enqueue(E3)

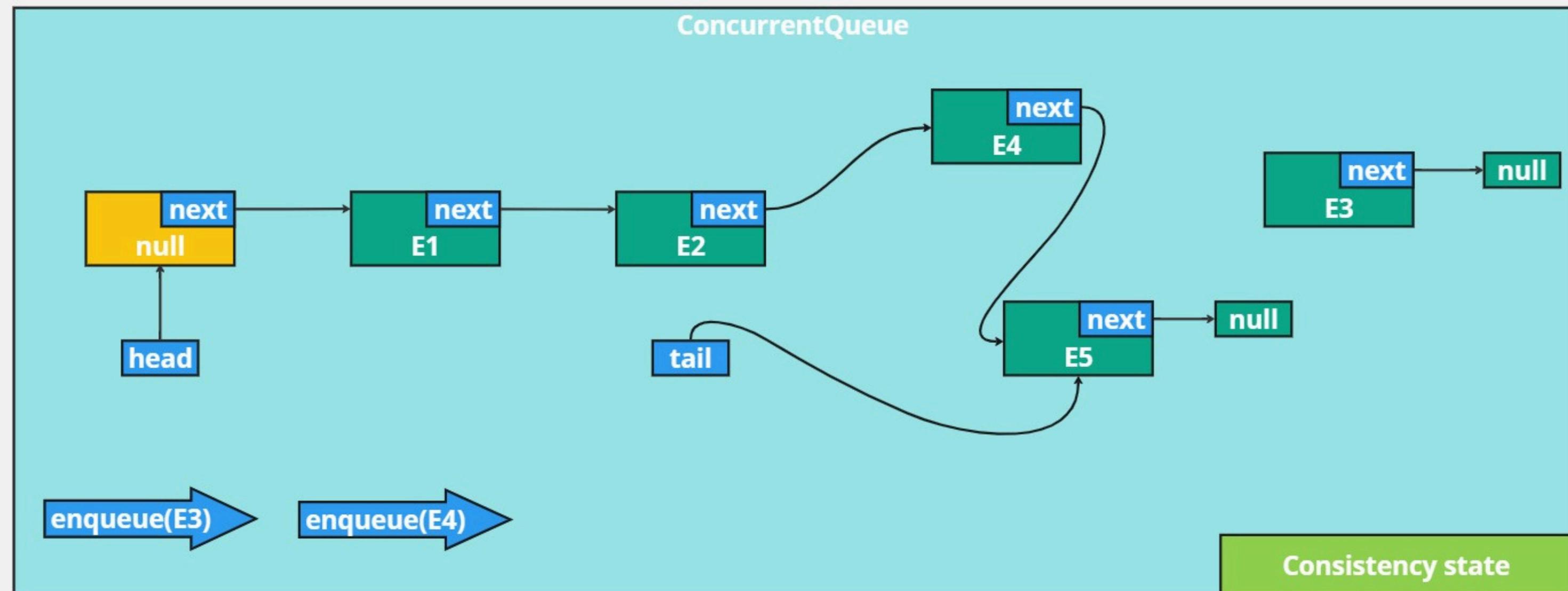
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

### enqueue(E4)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

### enqueue(E5)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```



### enqueue(E3)

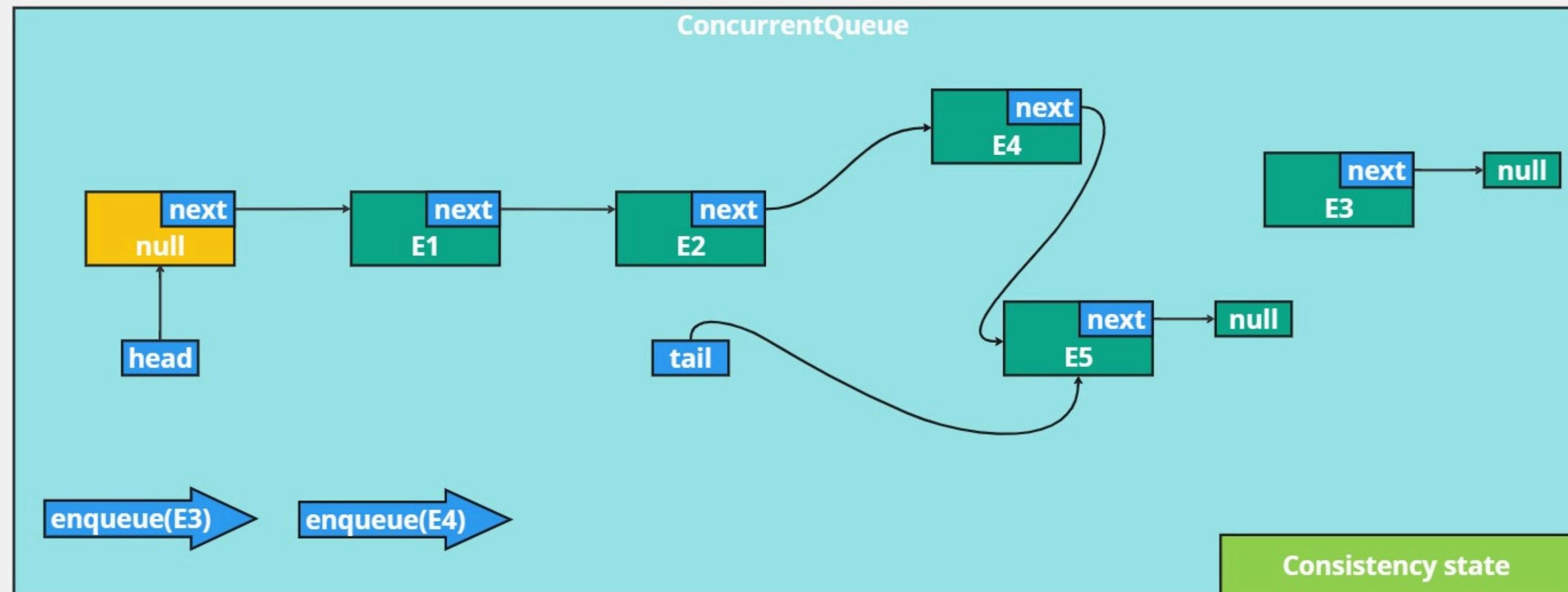
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

### enqueue(E4)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

### enqueue(E5)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```



### enqueue(E3)

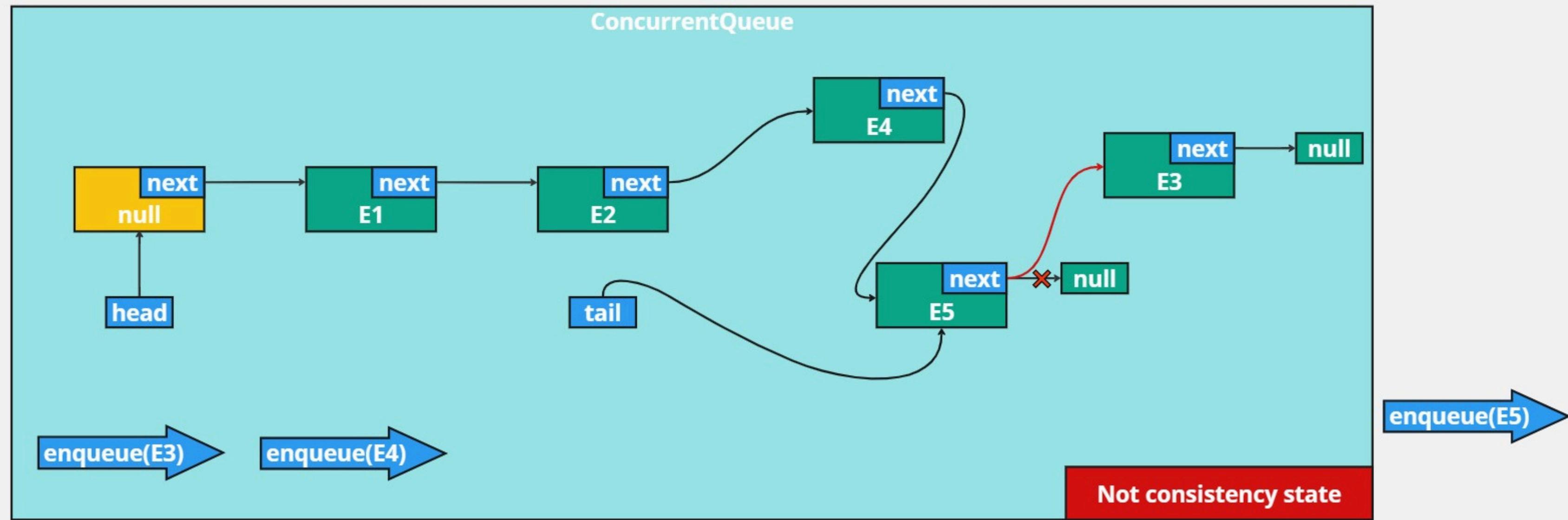
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

### enqueue(E4)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

### enqueue(E5)

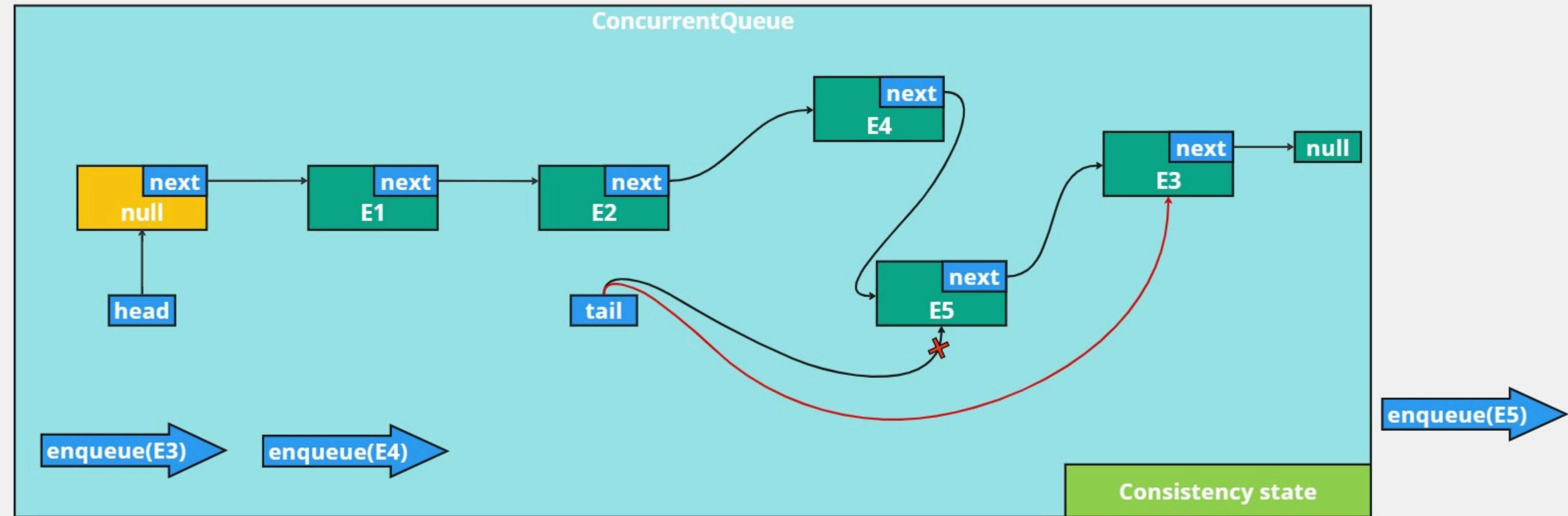
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```



```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```



### enqueue(E3)

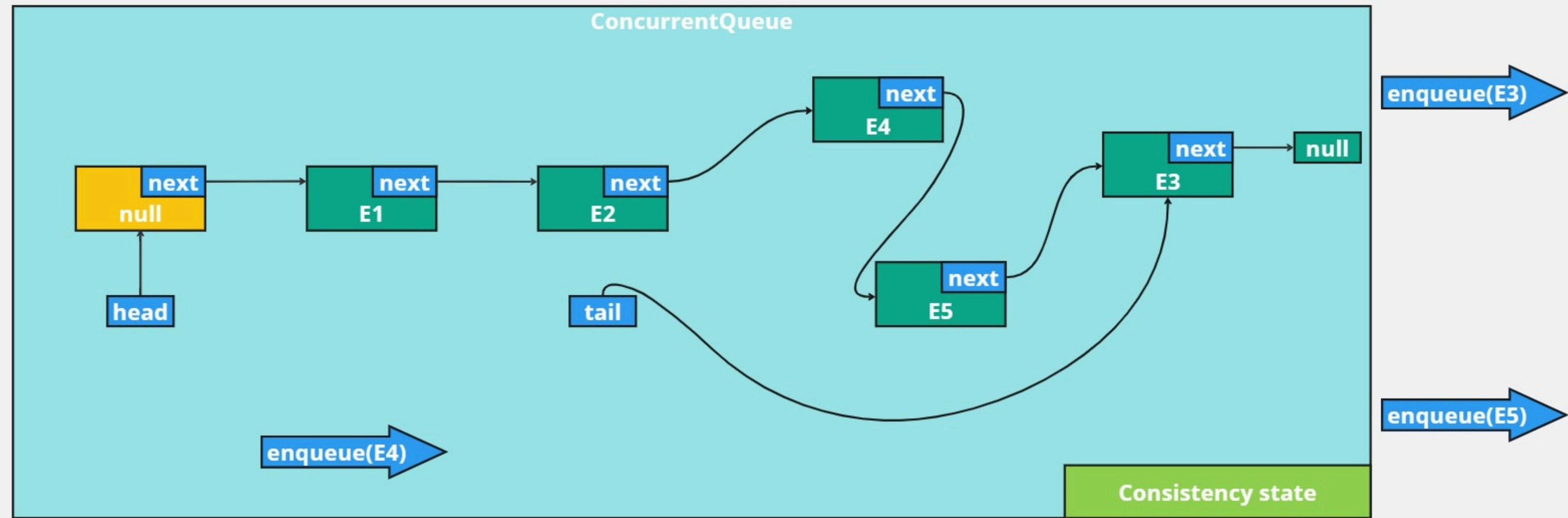
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

### enqueue(E4)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

### enqueue(E5)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```



### enqueue(E3)

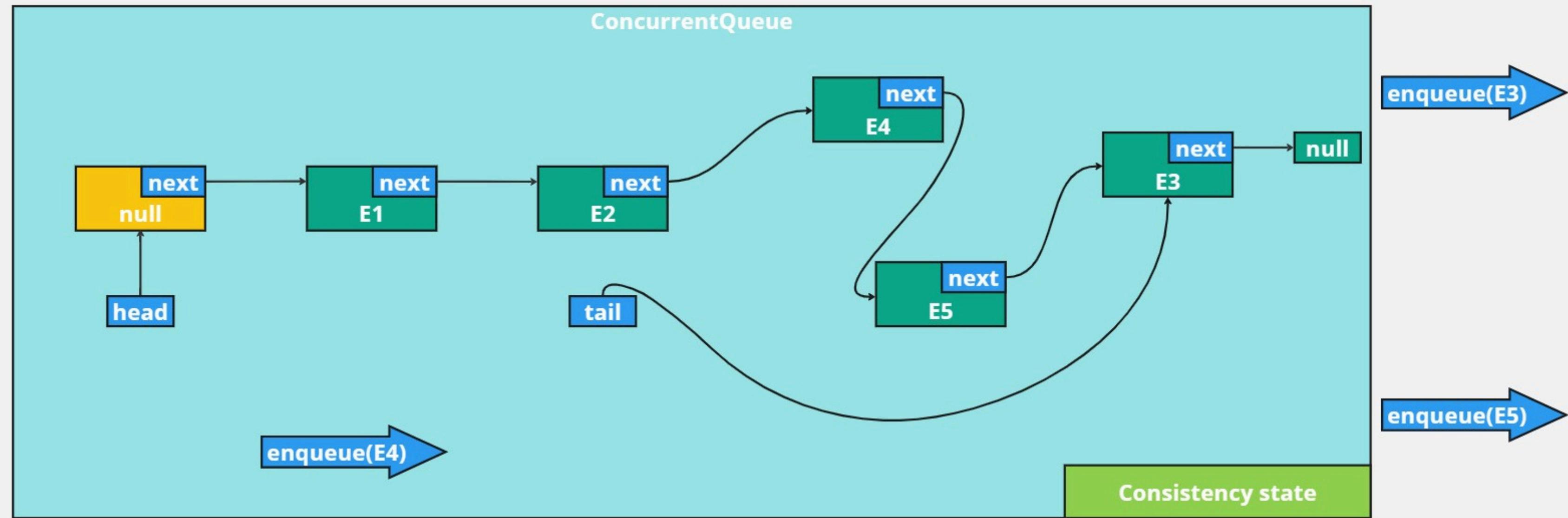
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

### enqueue(E4)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

### enqueue(E5)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```



### enqueue(E3)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

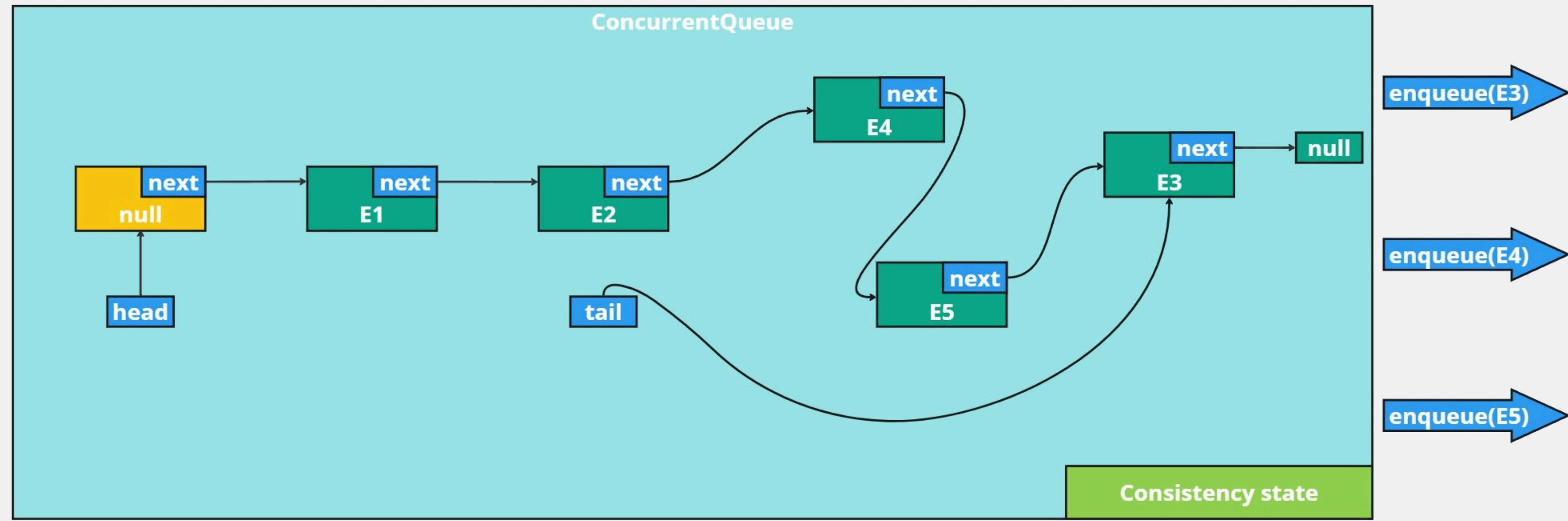
### enqueue(E4)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

A callout box highlights the line `tail.compareAndSet(previousTail, newNode);` in red.

### enqueue(E5)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```



### enqueue(E3)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

### enqueue(E4)

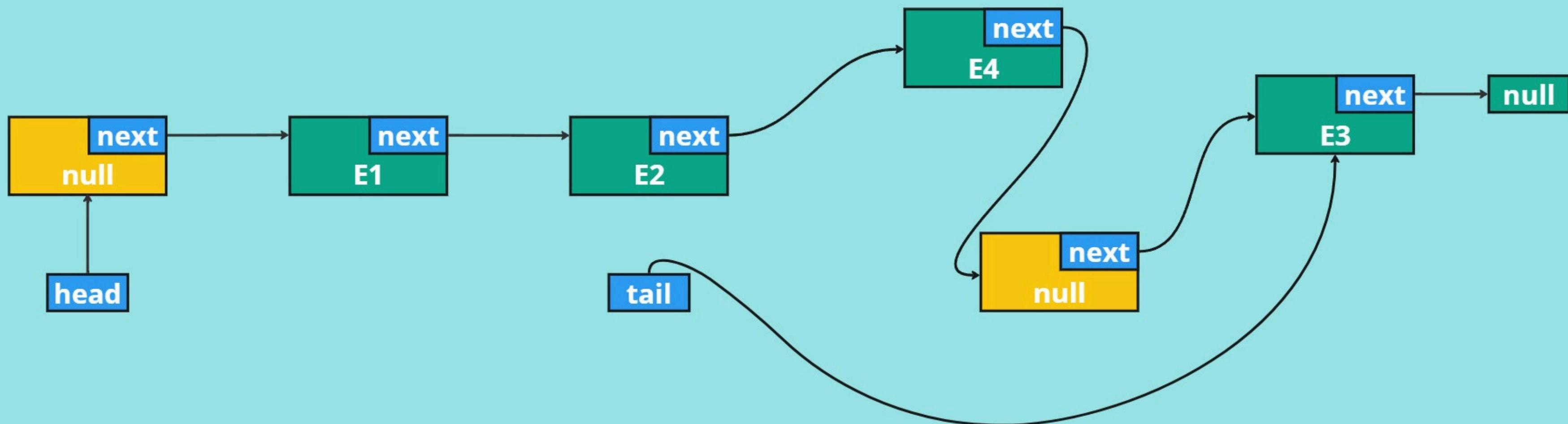
```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

A red box highlights the 'return' statement in the code.

### enqueue(E5)

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

## ConcurrentQueue



```
public void enqueue(final E element) {  
    final Node<E> newNode = new Node<>(element);  
    while (true) {  
        final Node<E> previousTail = tail.get();  
        if (previousTail.next.compareAndSet(null, newNode)) {  
            tail.compareAndSet(previousTail, newNode);  
            return;  
        } else {  
            tail.compareAndSet(previousTail, previousTail.next.get());  
        }  
    }  
}
```



```
public void enqueue(final E element) {  
    final Node<E> newNode = new Node<>(requireNonNull(element));  
    while (true) {  
        final Node<E> previousTail = tail.get();  
        if (previousTail.next.compareAndSet(null, newNode)) {  
            tail.compareAndSet(previousTail, newNode);  
            return;  
        } else {  
            tail.compareAndSet(previousTail, previousTail.next.get());  
        }  
    }  
}
```

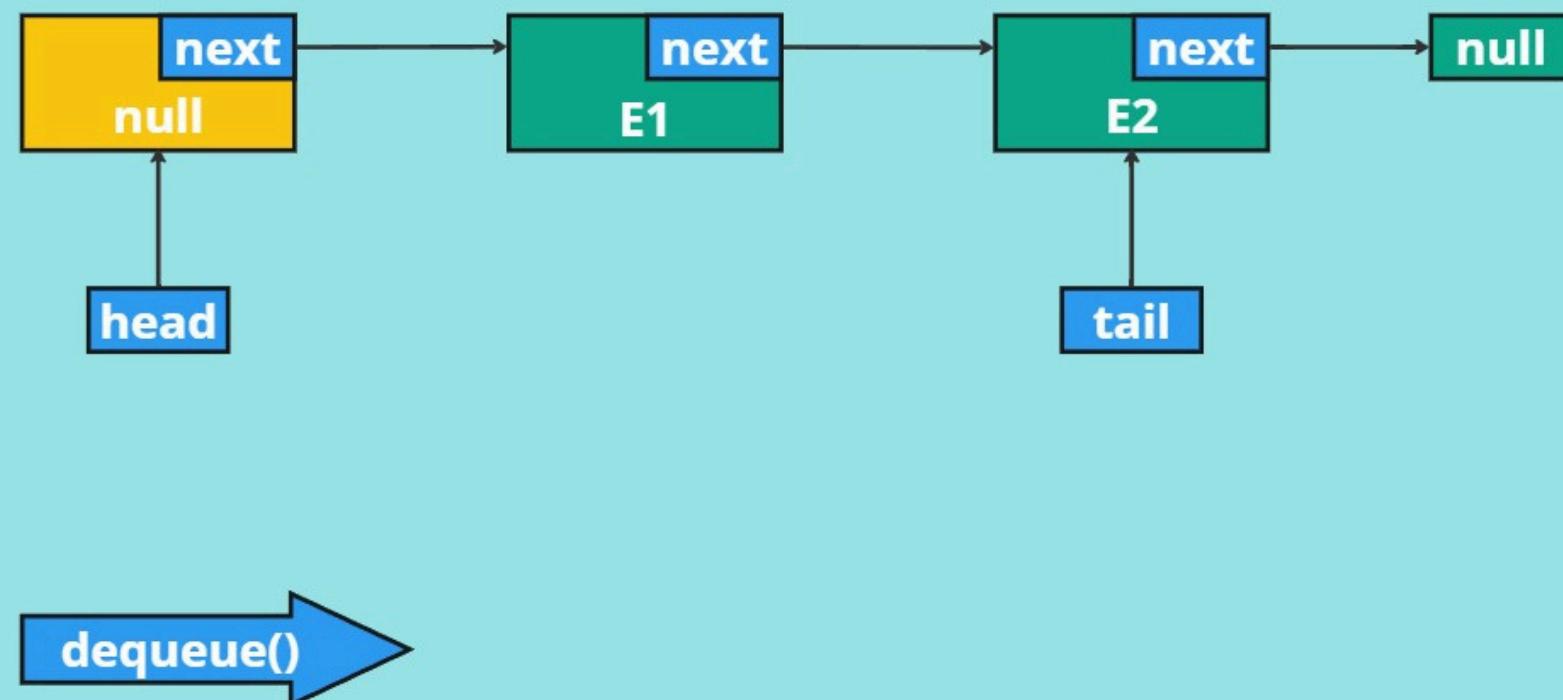
## ConcurrentQueue



dequeue()

```
public Optional<E> dequeue() {  
    //check if queue is empty  
    //extract first node's value  
    //first node's value should be assigned null  
    //head should be assigned first node  
    //previous head's next should be assigned previous head  
    //return element  
}
```

## ConcurrentQueue



```
public Optional<E> dequeue() {  
    //check if queue is empty  
    //extract first node's value  
    //first node's value should be assigned null  
    //head should be assigned first node  
    //previous head's next should be assigned previous head  
    //return element  
}
```

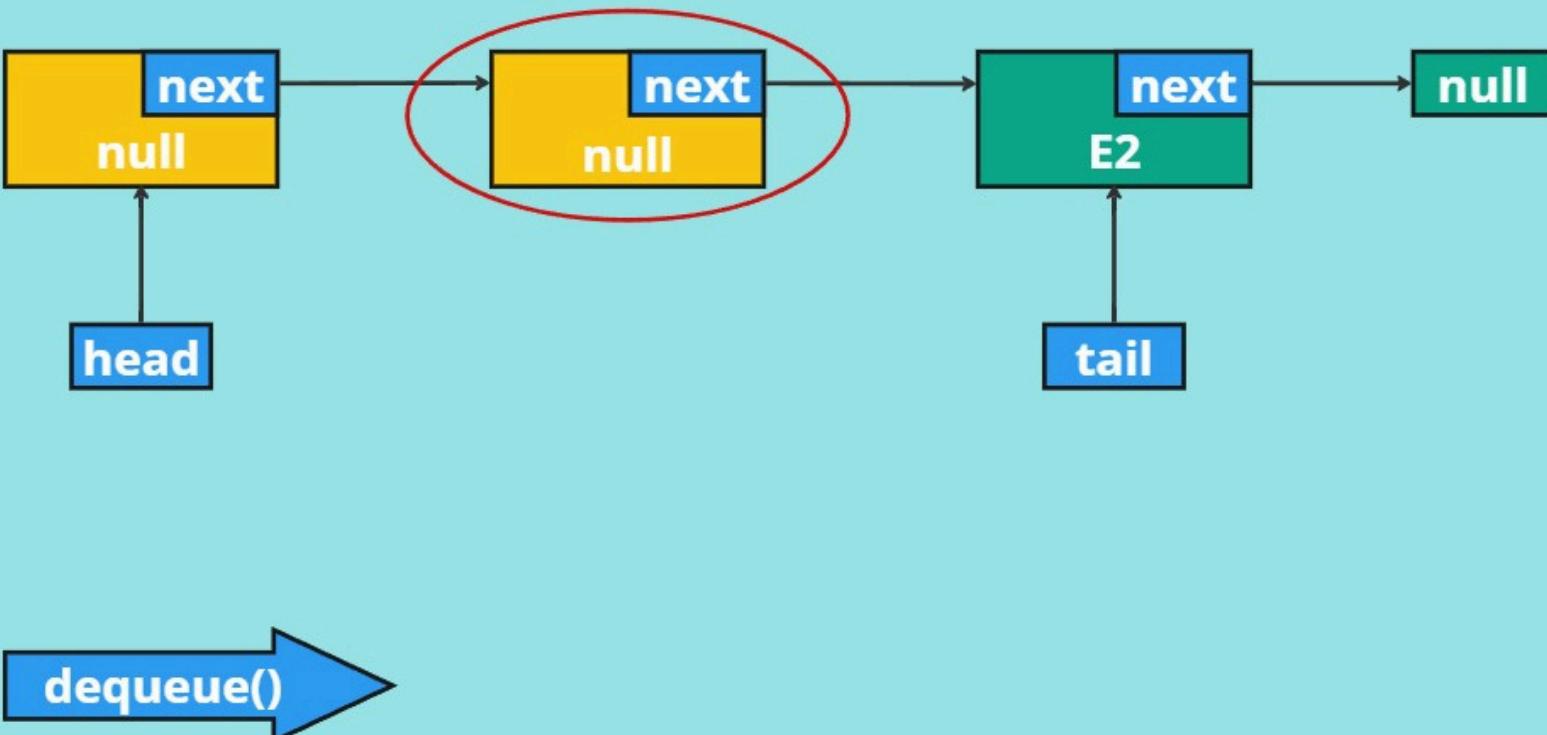
## ConcurrentQueue



**dequeue()**

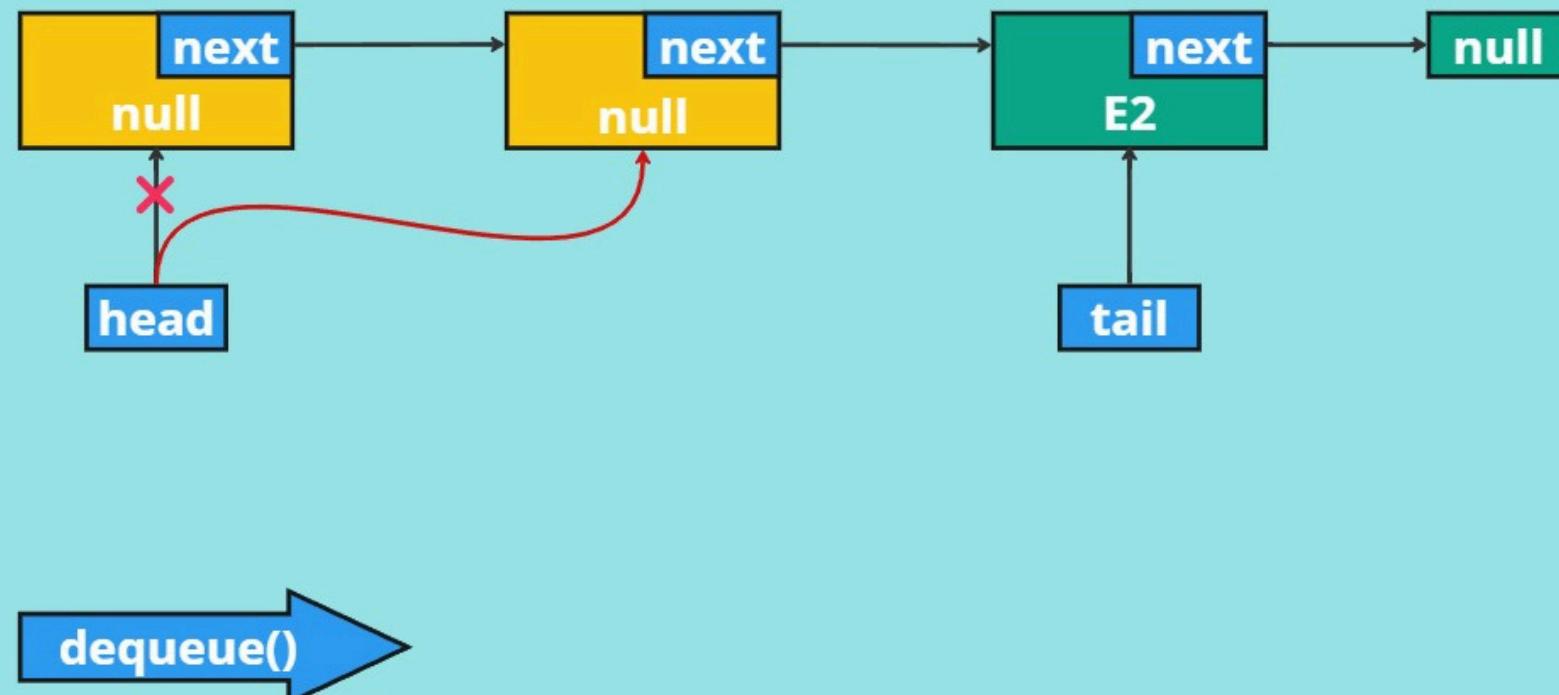
```
public Optional<E> dequeue() {  
    //check if queue is empty  
    //extract first node's value  
    //first node's value should be assigned null  
    //head should be assigned first node  
    //previous head's next should be assigned previous head  
    //return element  
}
```

## ConcurrentQueue



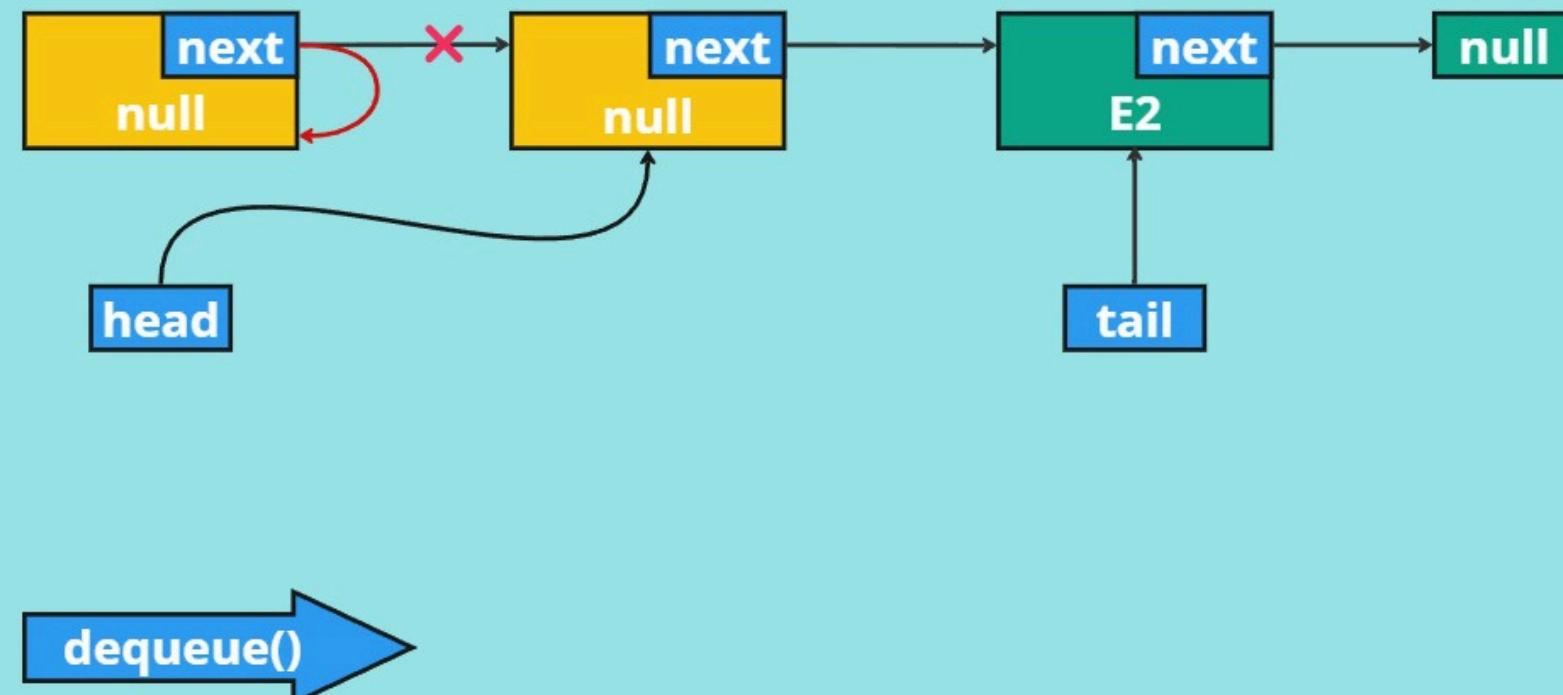
```
public Optional<E> dequeue() {  
    //check if queue is empty  
    //extract first node's value  
    //first node's value should be assigned null  
    //head should be assigned first node  
    //previous head's next should be assigned previous head  
    //return element  
}
```

## ConcurrentQueue



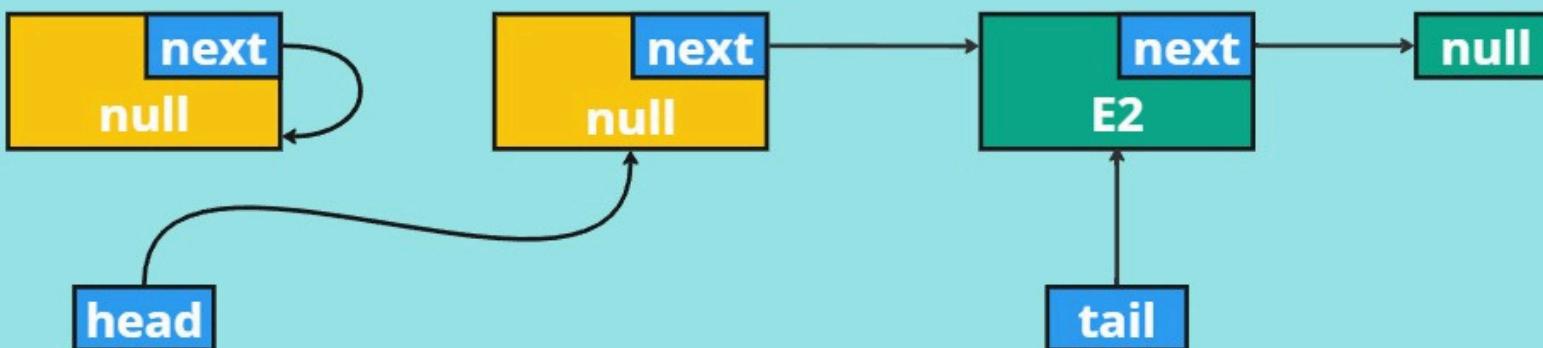
```
public Optional<E> dequeue() {  
    //check if queue is empty  
    //extract first node's value  
    //first node's value should be assigned null  
    //head should be assigned first node  
    //previous head's next should be assigned previous head  
    //return element  
}
```

## ConcurrentQueue



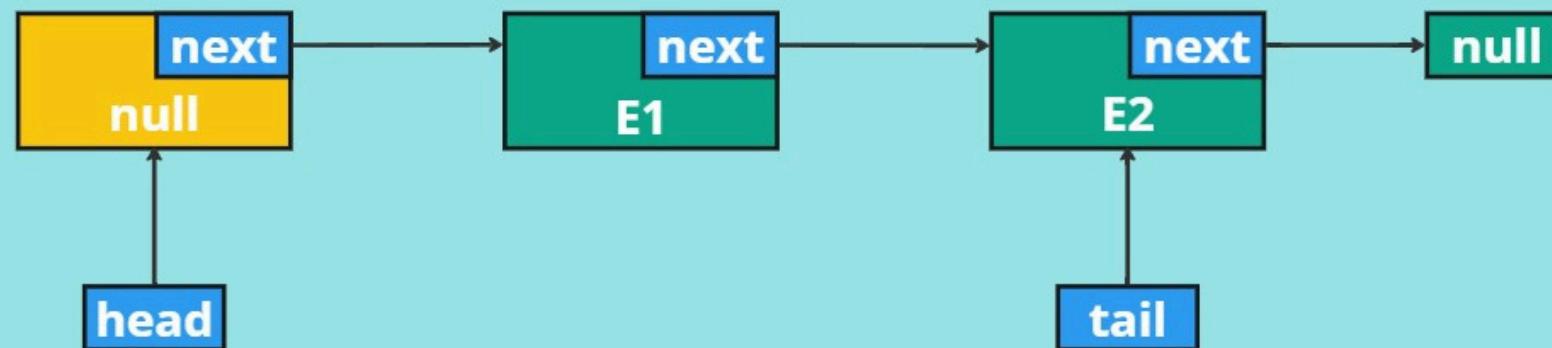
```
public Optional<E> dequeue() {  
    //check if queue is empty  
    //extract first node's value  
    //first node's value should be assigned null  
    //head should be assigned first node  
    //previous head's next should be assigned previous head  
    //return element  
}
```

## ConcurrentQueue



```
public Optional<E> dequeue() {  
    //check if queue is empty  
    //extract first node's value  
    //first node's value should be assigned null  
    //head should be assigned first node  
    //previous head's next should be assigned previous head  
    //return element  
}
```

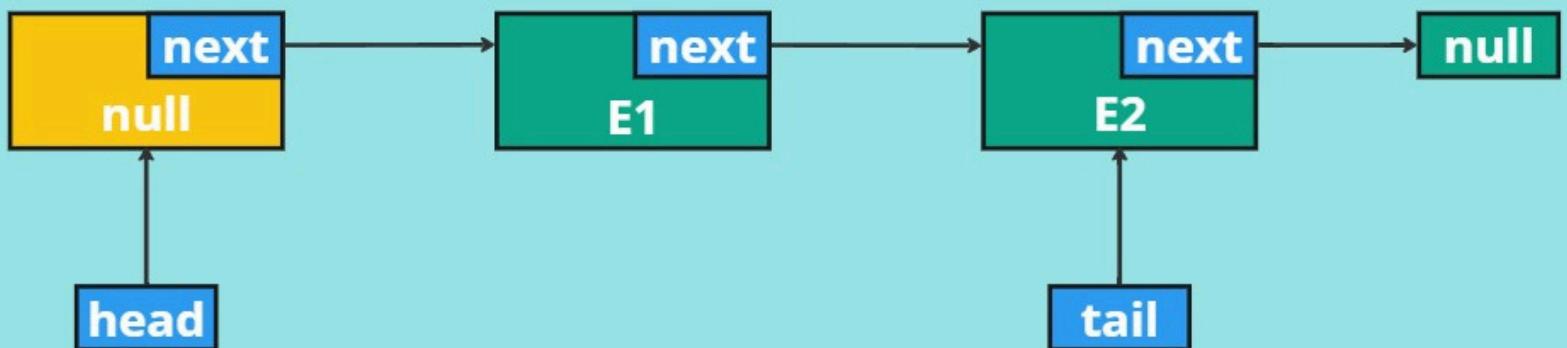
## ConcurrentQueue



**dequeue()**

```
public Optional<E> dequeue() {  
    final Node<E> previousHead = head;  
    final Node<E> nextHead = previousHead.next.get();  
    //check if queue is empty  
    //extract first node's value  
    //first node's value should be assigned null  
    //head should be assigned first node  
    //previous head's next should be assigned previous head  
    //return element  
}
```

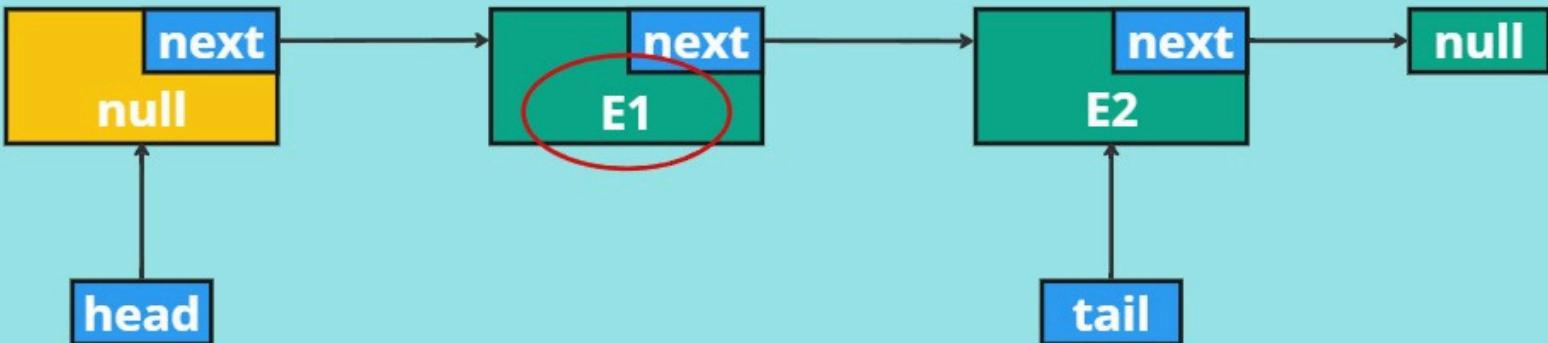
## ConcurrentQueue



dequeue()

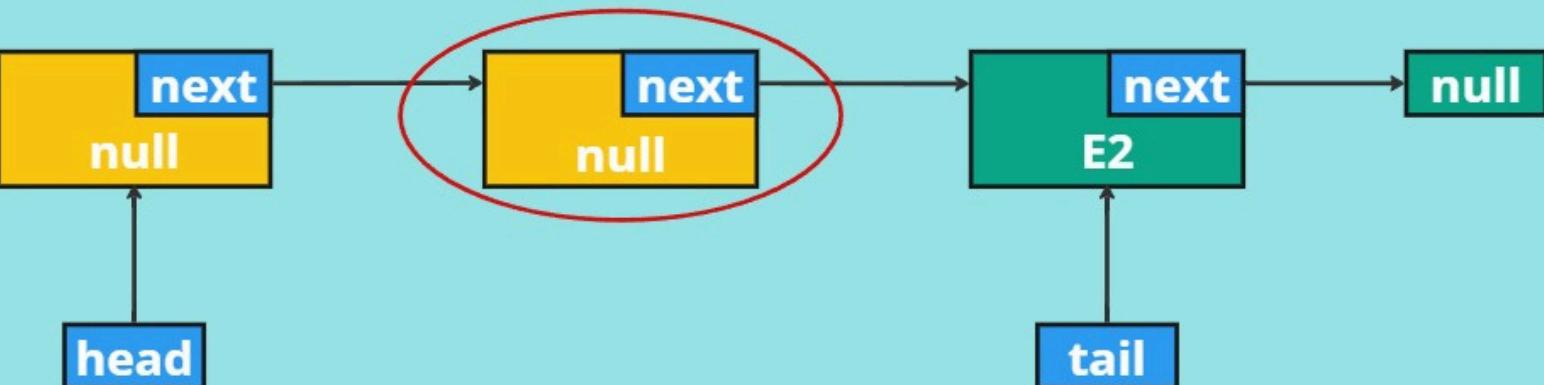
```
public Optional<E> dequeue() {
    final Node<E> previousHead = head;
    final Node<E> nextHead = previousHead.next.get();
    if (previousHead == tail.get()) {
        return empty();
    }
    //extract first node's value
    //first node's value should be assigned null
    //head should be assigned first node
    //previous head's next should be assigned previous head
    //return element
}
```

## ConcurrentQueue



```
public Optional<E> dequeue() {
    final Node<E> previousHead = head;
    final Node<E> nextHead = previousHead.next.get();
    if (previousHead == tail.get()) {
        return empty();
    }
    final E element = nextHead.value;
    //first node's value should be assigned null
    //head should be assigned first node
    //previous head's next should be assigned previous head
    //return element
}
```

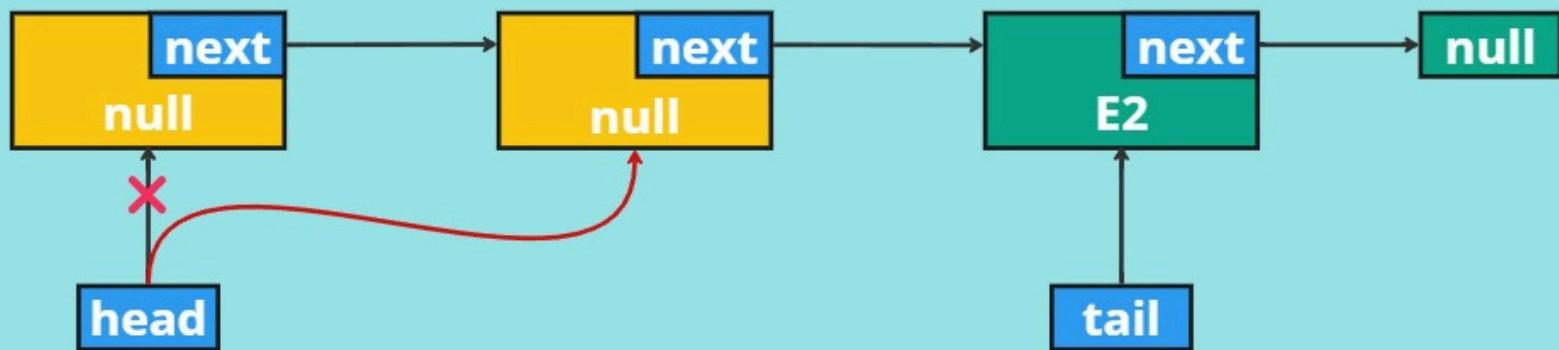
## ConcurrentQueue



**dequeue()**

```
public Optional<E> dequeue() {
    final Node<E> previousHead = head;
    final Node<E> nextHead = previousHead.next.get();
    if (previousHead == tail.get()) {
        return empty();
    }
    final E element = nextHead.value;
    nextHead.value = null;
    //head should be assigned first node
    //previous head's next should be assigned previous head
    //return element
}
```

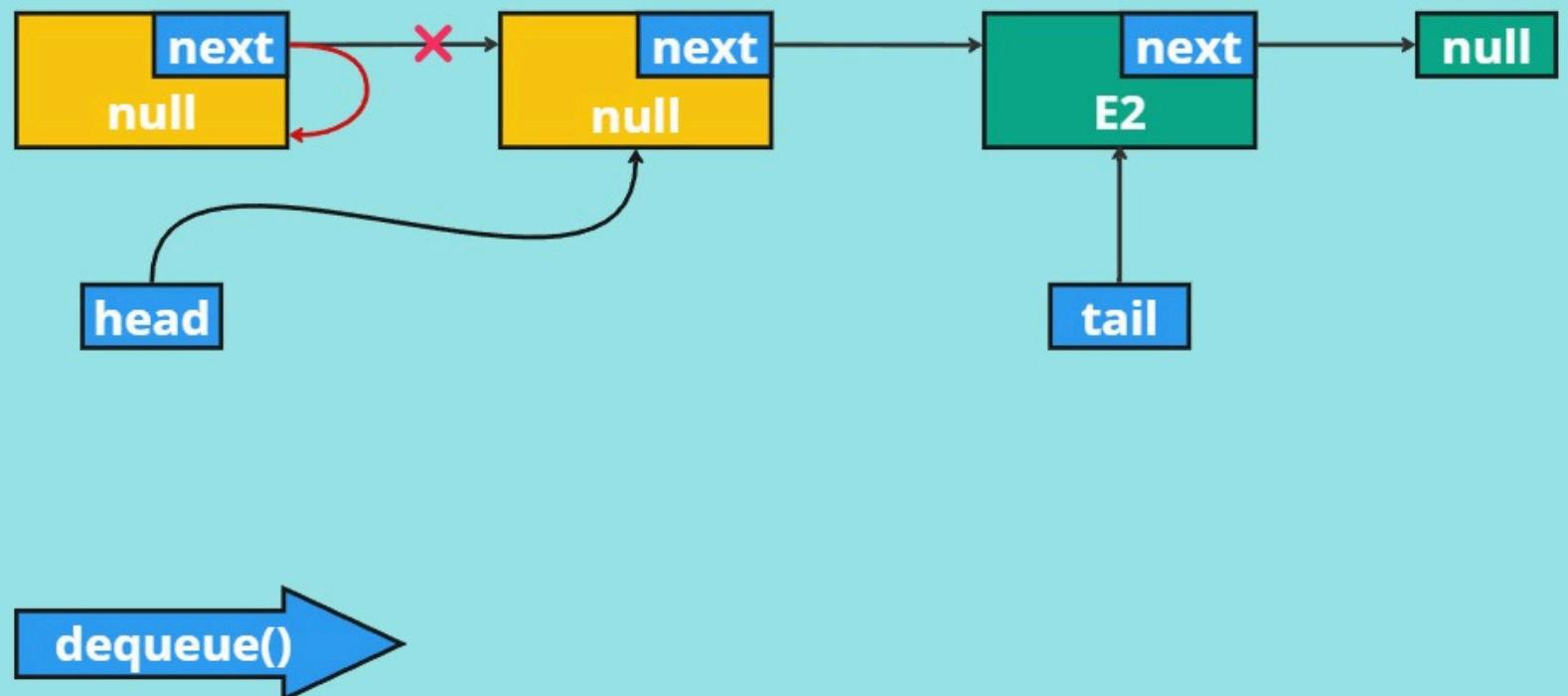
## ConcurrentQueue



dequeue()

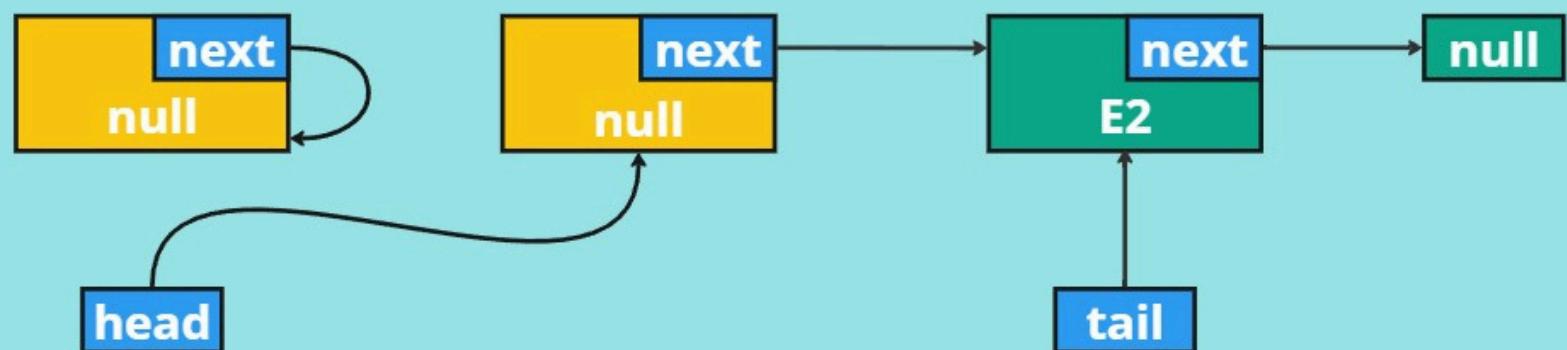
```
public Optional<E> dequeue() {
    final Node<E> previousHead = head;
    final Node<E> nextHead = previousHead.next.get();
    if (previousHead == tail.get()) {
        return empty();
    }
    final E element = nextHead.value;
    nextHead.value = null;
    head = nextHead;
    //previous head's next should be assigned previous head
    //return element
}
```

## ConcurrentQueue



```
public Optional<E> dequeue() {
    final Node<E> previousHead = head;
    final Node<E> nextHead = previousHead.next.get();
    if (previousHead == tail.get()) {
        return empty();
    }
    final E element = nextHead.value;
    nextHead.value = null;
    head = nextHead;
    previousHead.next.set(previousHead);
    //return element
}
```

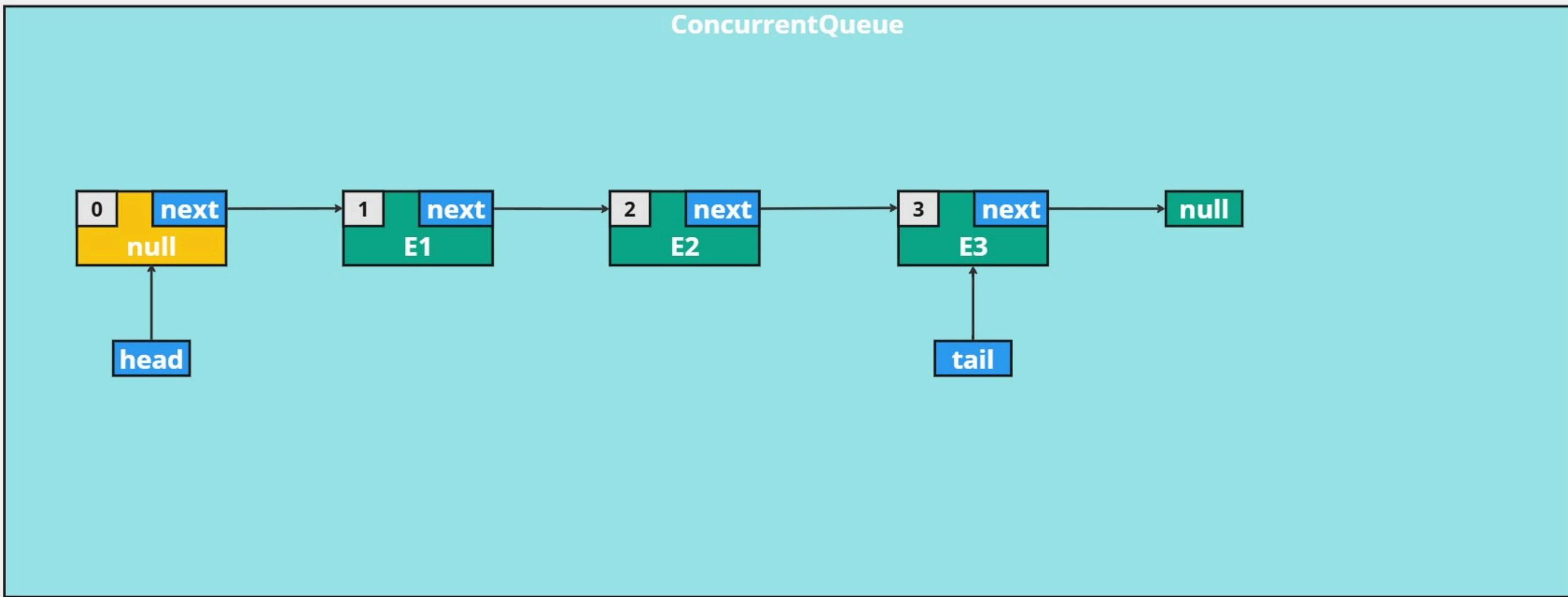
## ConcurrentQueue



```
public Optional<E> dequeue() {
    final Node<E> previousHead = head;
    final Node<E> nextHead = previousHead.next.get();
    if (previousHead == tail.get()) {
        return empty();
    }
    final E element = nextHead.value;
    nextHead.value = null;
    head = nextHead;
    previousHead.next.set(previousHead);
    return Optional.of(element);
}
```

## ConcurrentQueue

thread-1 →  
thread-2 →  
thread-3 →



### thread-1

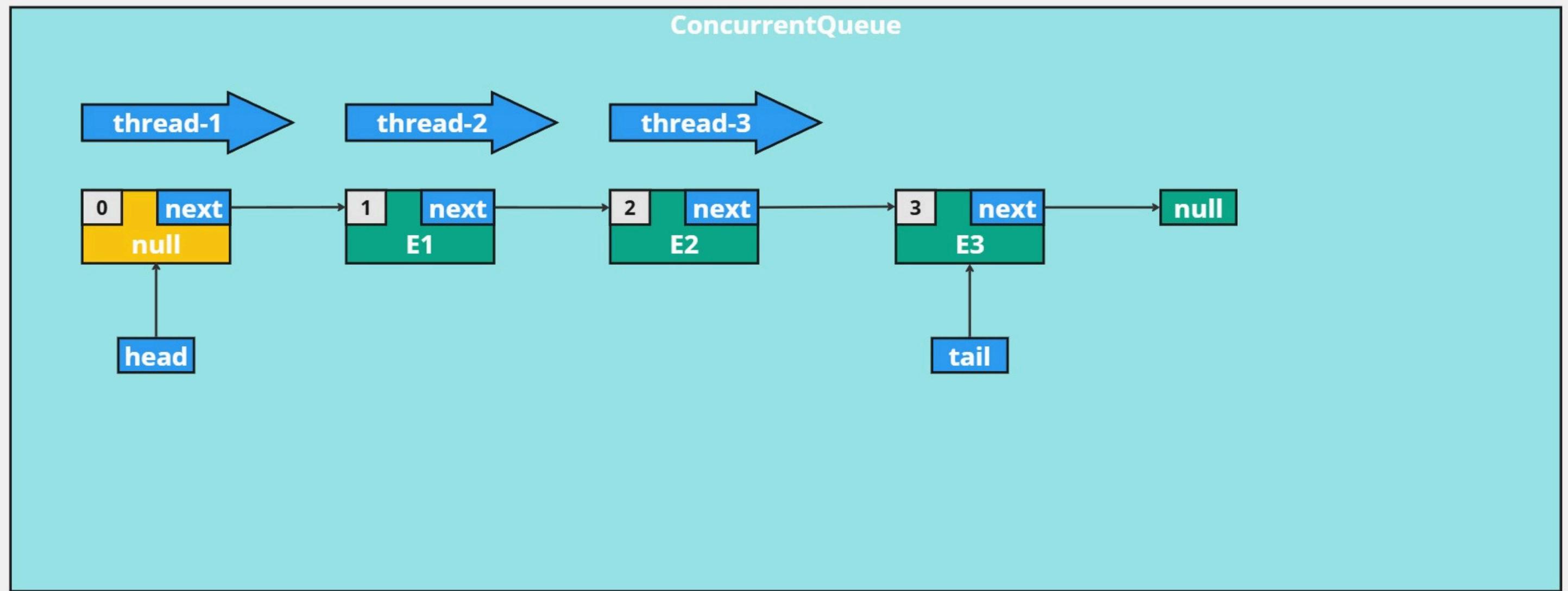
```
public Optional<E> dequeue() {  
    final Node<E> previousHead = head;  
    final Node<E> nextHead = previousHead.next.get();  
    if (previousHead == tail.get()) {  
        return empty();  
    }  
    final E element = nextHead.value;  
    nextHead.value = null;  
    head = nextHead;  
    previousHead.next.set(previousHead);  
    return Optional.of(element);  
}
```

### thread-2

```
public Optional<E> dequeue() {  
    final Node<E> previousHead = head;  
    final Node<E> nextHead = previousHead.next.get();  
    if (previousHead == tail.get()) {  
        return empty();  
    }  
    final E element = nextHead.value;  
    nextHead.value = null;  
    head = nextHead;  
    previousHead.next.set(previousHead);  
    return Optional.of(element);  
}
```

### thread-3

```
public Optional<E> dequeue() {  
    final Node<E> previousHead = head;  
    final Node<E> nextHead = previousHead.next.get();  
    if (previousHead == tail.get()) {  
        return empty();  
    }  
    final E element = nextHead.value;  
    nextHead.value = null;  
    head = nextHead;  
    previousHead.next.set(previousHead);  
    return Optional.of(element);  
}
```



**thread-1**

```

public Optional<E> dequeue() {
    final Node<E> previousHead = head;
    final Node<E> nextHead = previousHead.next.get();
    if (previousHead == tail.get()) {
        return empty();
    }
    final E element = nextHead.value;
    nextHead.value = null;
    head = nextHead;
    previousHead.next.set(previousHead);
    return Optional.of(element);
}

```

**thread-2**

```

public Optional<E> dequeue() {
    final Node<E> previousHead = head;
    final Node<E> nextHead = previousHead.next.get();
    if (previousHead == tail.get()) {
        return empty();
    }
    final E element = nextHead.value;
    nextHead.value = null;
    head = nextHead;
    previousHead.next.set(previousHead);
    return Optional.of(element);
}

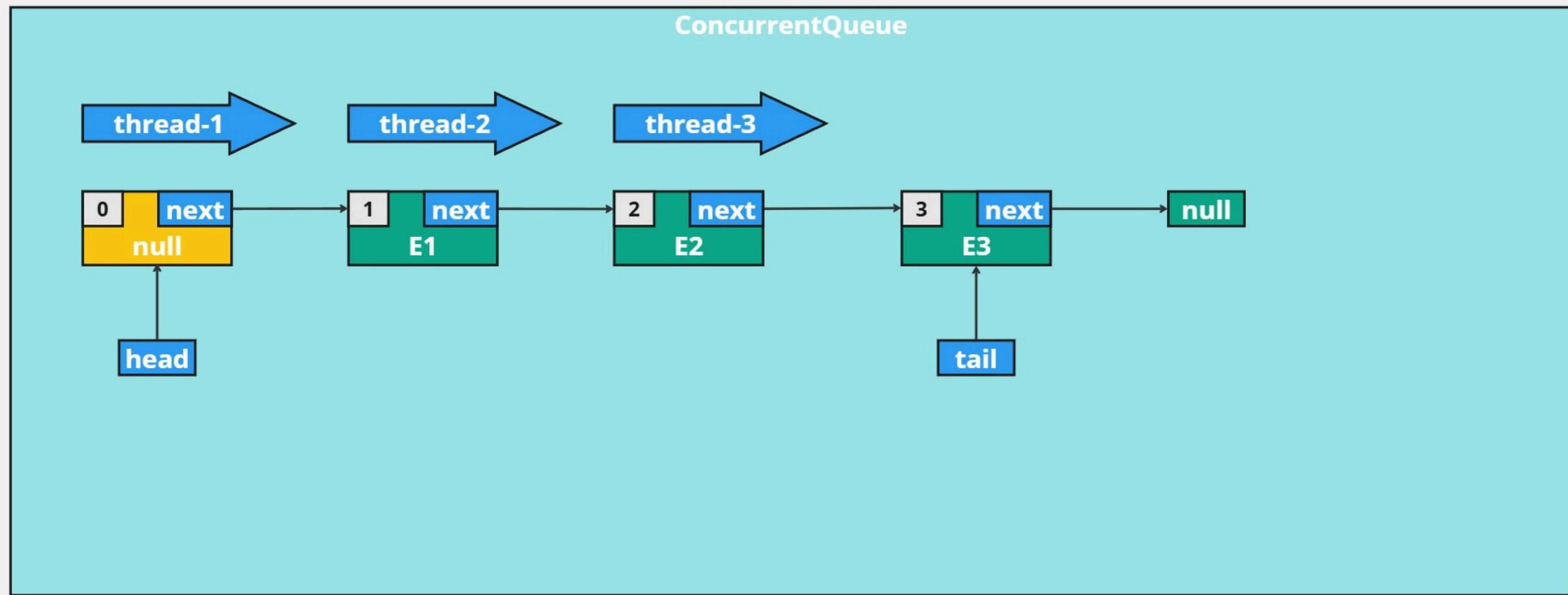
```

**thread-3**

```

public Optional<E> dequeue() {
    final Node<E> previousHead = head;
    final Node<E> nextHead = previousHead.next.get();
    if (previousHead == tail.get()) {
        return empty();
    }
    final E element = nextHead.value;
    nextHead.value = null;
    head = nextHead;
    previousHead.next.set(previousHead);
    return Optional.of(element);
}

```



**thread-1**

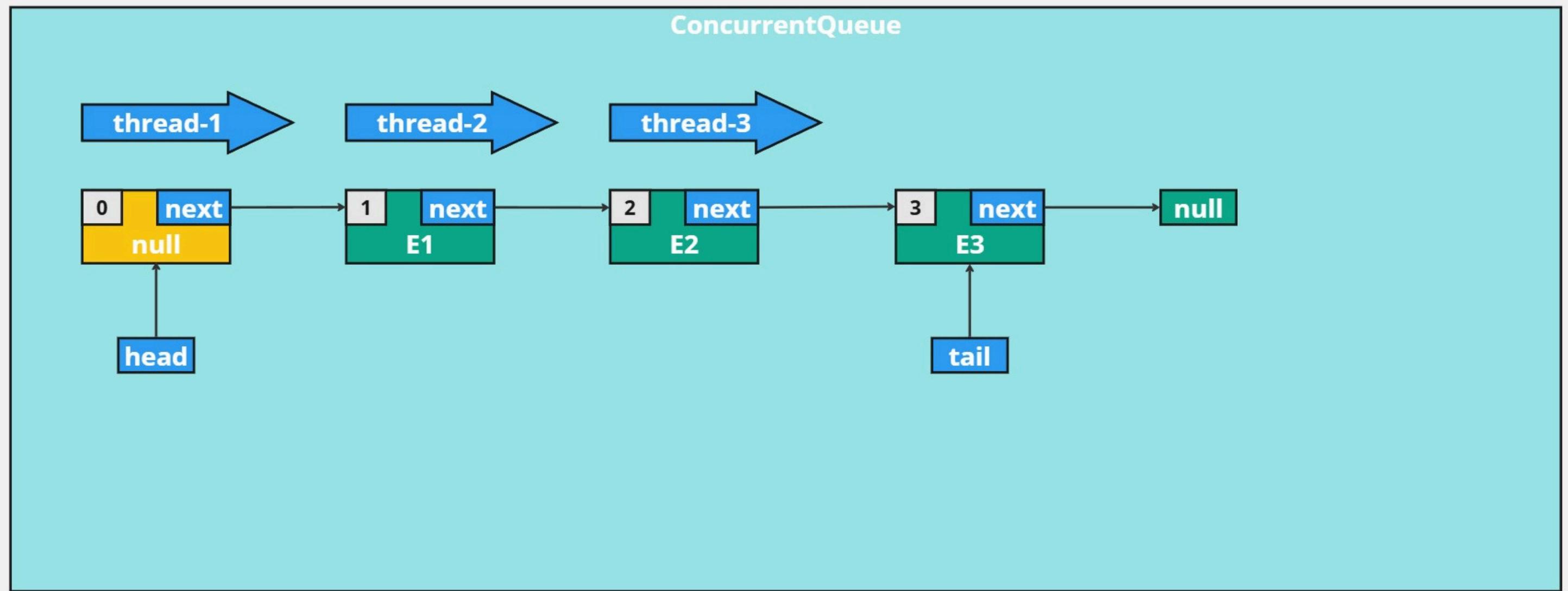
```
public Optional<E> dequeue() {
    final Node<E> previousHead = head;
    final Node<E> nextHead = previousHead.next.get();
    if (previousHead == tail.get()) {
        return empty();
    }
    final E element = nextHead.value;
    nextHead.value = null;
    head = nextHead;
    previousHead.next.set(previousHead);
    return Optional.of(element);
}
```

**thread-2**

```
public Optional<E> dequeue() {
    final Node<E> previousHead = head;
    final Node<E> nextHead = previousHead.next.get();
    if (previousHead == tail.get()) {
        return empty();
    }
    final E element = nextHead.value;
    nextHead.value = null;
    head = nextHead;
    previousHead.next.set(previousHead);
    return Optional.of(element);
}
```

**thread-3**

```
public Optional<E> dequeue() {
    final Node<E> previousHead = head;
    final Node<E> nextHead = previousHead.next.get();
    if (previousHead == tail.get()) {
        return empty();
    }
    final E element = nextHead.value;
    nextHead.value = null;
    head = nextHead;
    previousHead.next.set(previousHead);
    return Optional.of(element);
}
```



**thread-1**

```

public Optional<E> dequeue() {
    final Node<E> previousHead = head;
    final Node<E> nextHead = previousHead.next.get();
    if (previousHead == tail.get()) {
        return empty();
    }
    final E element = nextHead.value;
    nextHead.value = null;
    head = nextHead;
    previousHead.next.set(previousHead);
    return Optional.of(element);
}

```

**thread-2**

```

public Optional<E> dequeue() {
    final Node<E> previousHead = head;
    final Node<E> nextHead = previousHead.next.get();
    if (previousHead == tail.get()) {
        return empty();
    }
    final E element = nextHead.value;
    nextHead.value = null;
    head = nextHead;
    previousHead.next.set(previousHead);
    return Optional.of(element);
}

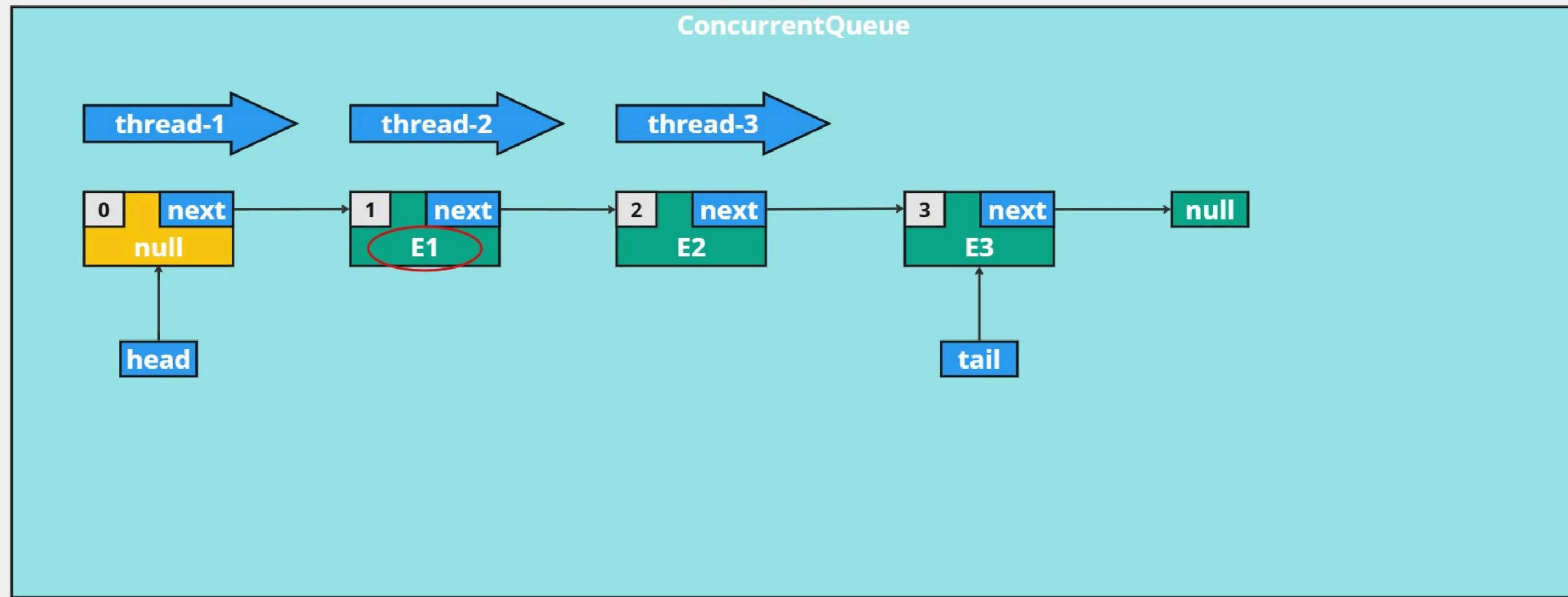
```

**thread-3**

```

public Optional<E> dequeue() {
    final Node<E> previousHead = head;
    final Node<E> nextHead = previousHead.next.get();
    if (previousHead == tail.get()) {
        return empty();
    }
    final E element = nextHead.value;
    nextHead.value = null;
    head = nextHead;
    previousHead.next.set(previousHead);
    return Optional.of(element);
}

```



### thread-1

```

public Optional<E> dequeue() {
    final Node<E> previousHead = head;
    final Node<E> nextHead = previousHead.next.get();
    if (previousHead == tail.get()) {
        return empty();
    }
    final E element = nextHead.value;
    nextHead.value = null;
    head = nextHead;
    previousHead.next.set(previousHead);
    return Optional.of(element);
}

```

### thread-2

```

public Optional<E> dequeue() {
    final Node<E> previousHead = head;
    final Node<E> nextHead = previousHead.next.get();
    if (previousHead == tail.get()) {
        return empty();
    }
    final E element = nextHead.value;
    nextHead.value = null;
    head = nextHead;
    previousHead.next.set(previousHead);
    return Optional.of(element);
}

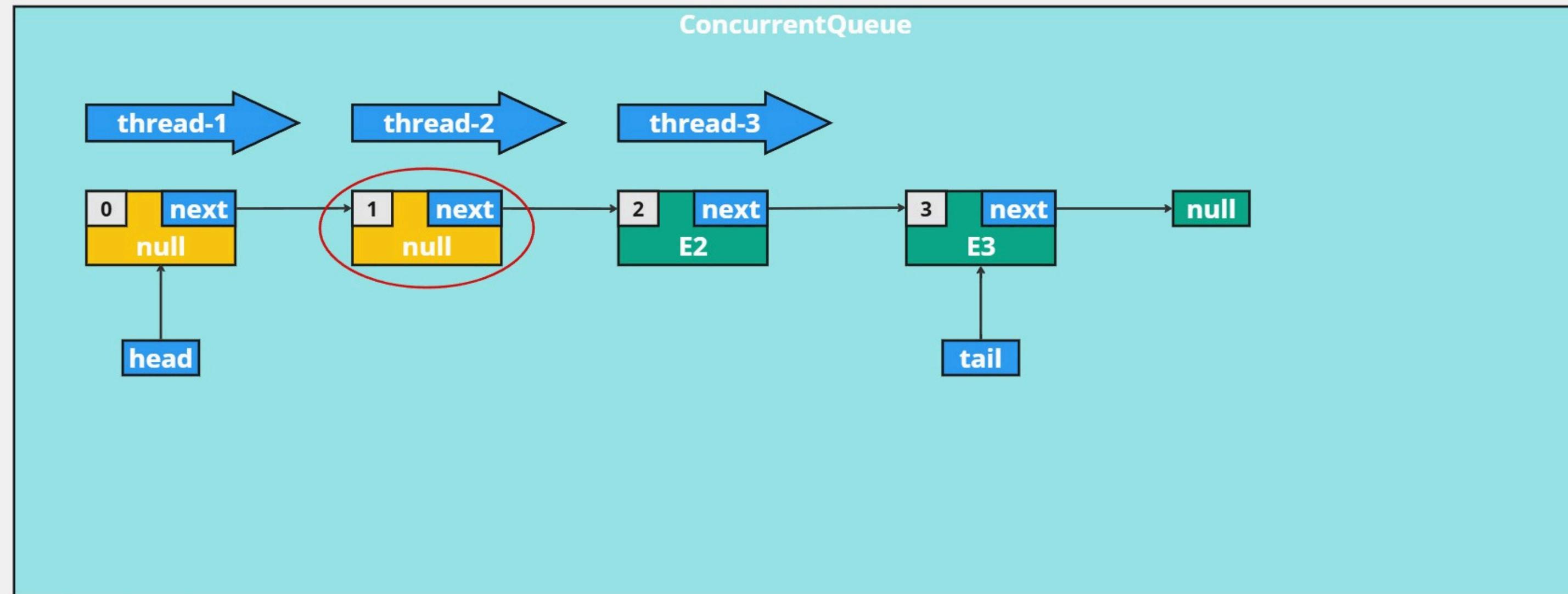
```

### thread-3

```

public Optional<E> dequeue() {
    final Node<E> previousHead = head;
    final Node<E> nextHead = previousHead.next.get();
    if (previousHead == tail.get()) {
        return empty();
    }
    final E element = nextHead.value;
    nextHead.value = null;
    head = nextHead;
    previousHead.next.set(previousHead);
    return Optional.of(element);
}

```



### thread-1

```

public Optional<E> dequeue() {
    final Node<E> previousHead = head;
    final Node<E> nextHead = previousHead.next.get();
    if (previousHead == tail.get()) {
        return empty();
    }
    final E element = nextHead.value;
    nextHead.value = null;
    head = nextHead;
    previousHead.next.set(previousHead);
    return Optional.of(element);
}

```

### thread-2

```

public Optional<E> dequeue() {
    final Node<E> previousHead = head;
    final Node<E> nextHead = previousHead.next.get();
    if (previousHead == tail.get()) {
        return empty();
    }
    final E element = nextHead.value;
    nextHead.value = null;
    head = nextHead;
    previousHead.next.set(previousHead);
    return Optional.of(element);
}

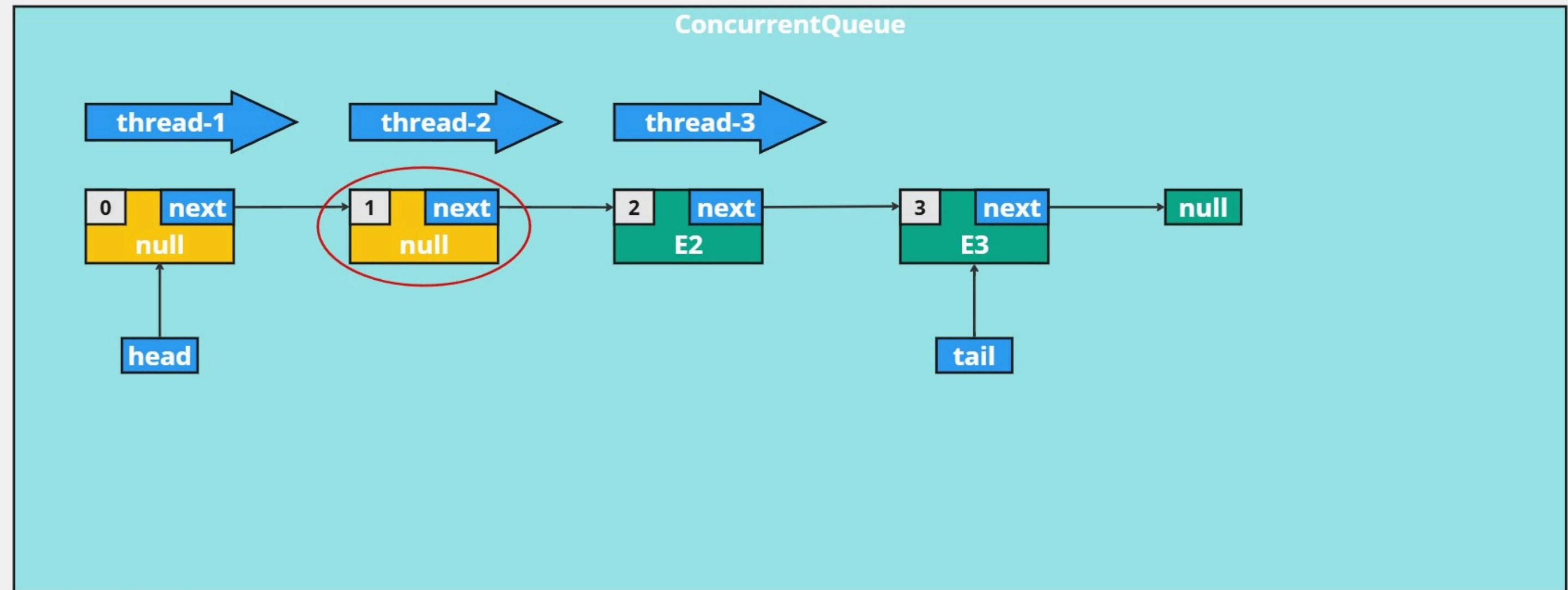
```

### thread-3

```

public Optional<E> dequeue() {
    final Node<E> previousHead = head;
    final Node<E> nextHead = previousHead.next.get();
    if (previousHead == tail.get()) {
        return empty();
    }
    final E element = nextHead.value;
    nextHead.value = null;
    head = nextHead;
    previousHead.next.set(previousHead);
    return Optional.of(element);
}

```



**thread-1**

```

public Optional<E> dequeue() {
    final Node<E> previousHead = head;
    final Node<E> nextHead = previousHead.next.get();
    if (previousHead == tail.get()) {
        return empty();
    }
    final E element = nextHead.value;
    nextHead.value = null;
    head = nextHead;
    previousHead.next.set(previousHead);
    return Optional.of(element);
}

```

**thread-2**

```

public Optional<E> dequeue() {
    final Node<E> previousHead = head;
    final Node<E> nextHead = previousHead.next.get();
    if (previousHead == tail.get()) {
        return empty();
    }
    final E element = nextHead.value;
    nextHead.value = null;
    head = nextHead;
    previousHead.next.set(previousHead);
    return Optional.of(element);
}

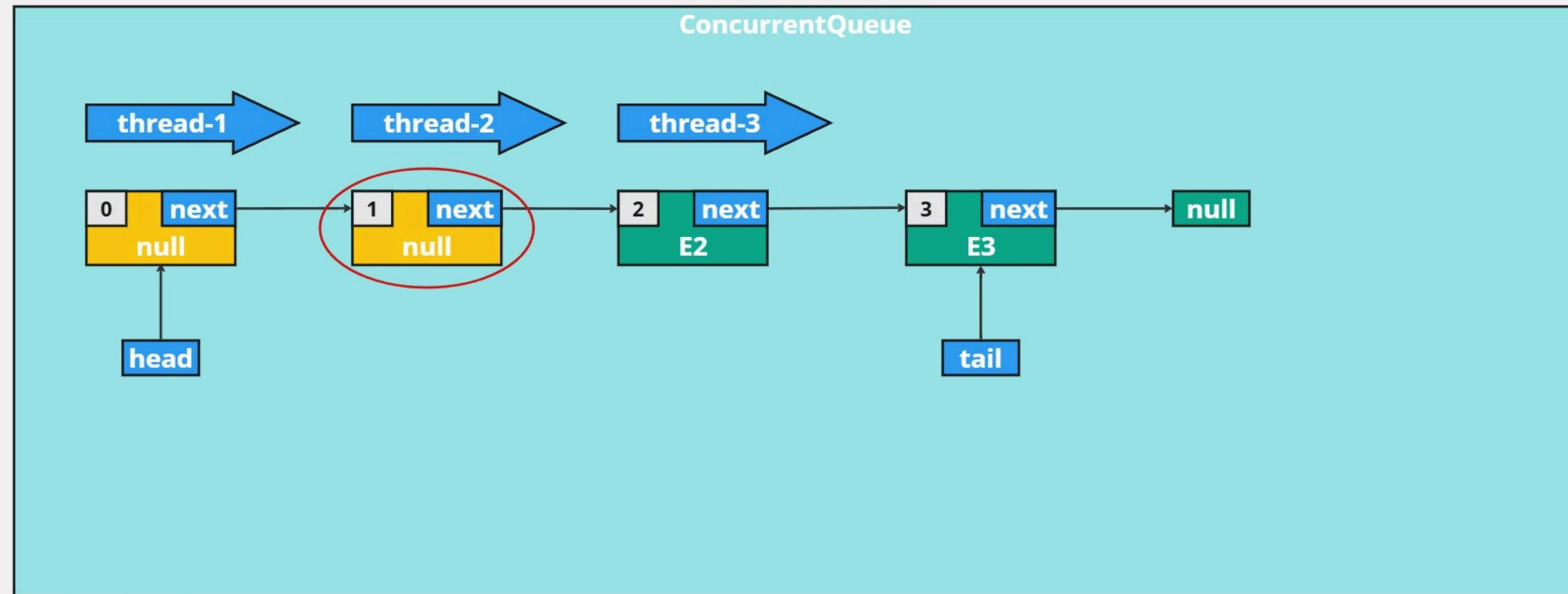
```

**thread-3**

```

public Optional<E> dequeue() {
    final Node<E> previousHead = head;
    final Node<E> nextHead = previousHead.next.get();
    if (previousHead == tail.get()) {
        return empty();
    }
    final E element = nextHead.value;
    nextHead.value = null;
    head = nextHead;
    previousHead.next.set(previousHead);
    return Optional.of(element);
}

```



**thread-1**

```

public Optional<E> dequeue() {
    final Node<E> previousHead = head;
    final Node<E> nextHead = previousHead.next.get();
    if (previousHead == tail.get()) {
        return empty();
    }
    final E element = nextHead.value;
    nextHead.value = null;
    head = nextHead;
    previousHead.next.set(previousHead);
    return Optional.of(element);
}

```

**thread-2**

```

public Optional<E> dequeue() {
    final Node<E> previousHead = head;
    final Node<E> nextHead = previousHead.next.get();
    if (previousHead == tail.get()) {
        return empty();
    }
    final E element = nextHead.value;
    nextHead.value = null;
    head = nextHead;
    previousHead.next.set(previousHead);
    return Optional.of(element);
}

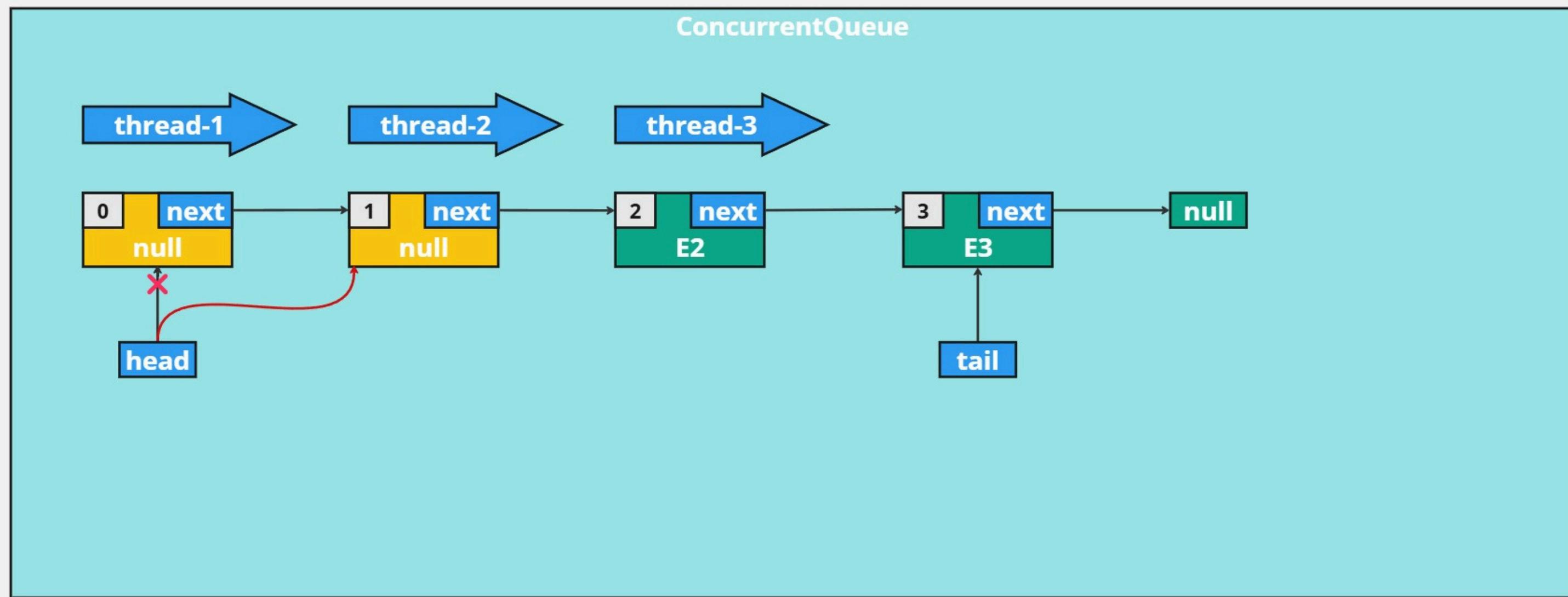
```

**thread-3**

```

public Optional<E> dequeue() {
    final Node<E> previousHead = head;
    final Node<E> nextHead = previousHead.next.get();
    if (previousHead == tail.get()) {
        return empty();
    }
    final E element = nextHead.value;
    nextHead.value = null;
    head = nextHead;
    previousHead.next.set(previousHead);
    return Optional.of(element);
}

```



### thread-1

```

public Optional<E> dequeue() {
    final Node<E> previousHead = head;
    final Node<E> nextHead = previousHead.next.get();
    if (previousHead == tail.get()) {
        return empty();
    }
    final E element = nextHead.value;
    nextHead.value = null;
    head = nextHead;
    previousHead.next.set(previousHead);
    return Optional.of(element);
}

```

### thread-2

```

public Optional<E> dequeue() {
    final Node<E> previousHead = head;
    final Node<E> nextHead = previousHead.next.get();
    if (previousHead == tail.get()) {
        return empty();
    }
    final E element = nextHead.value;
    nextHead.value = null;
    head = nextHead;
    previousHead.next.set(previousHead);
    return Optional.of(element);
}

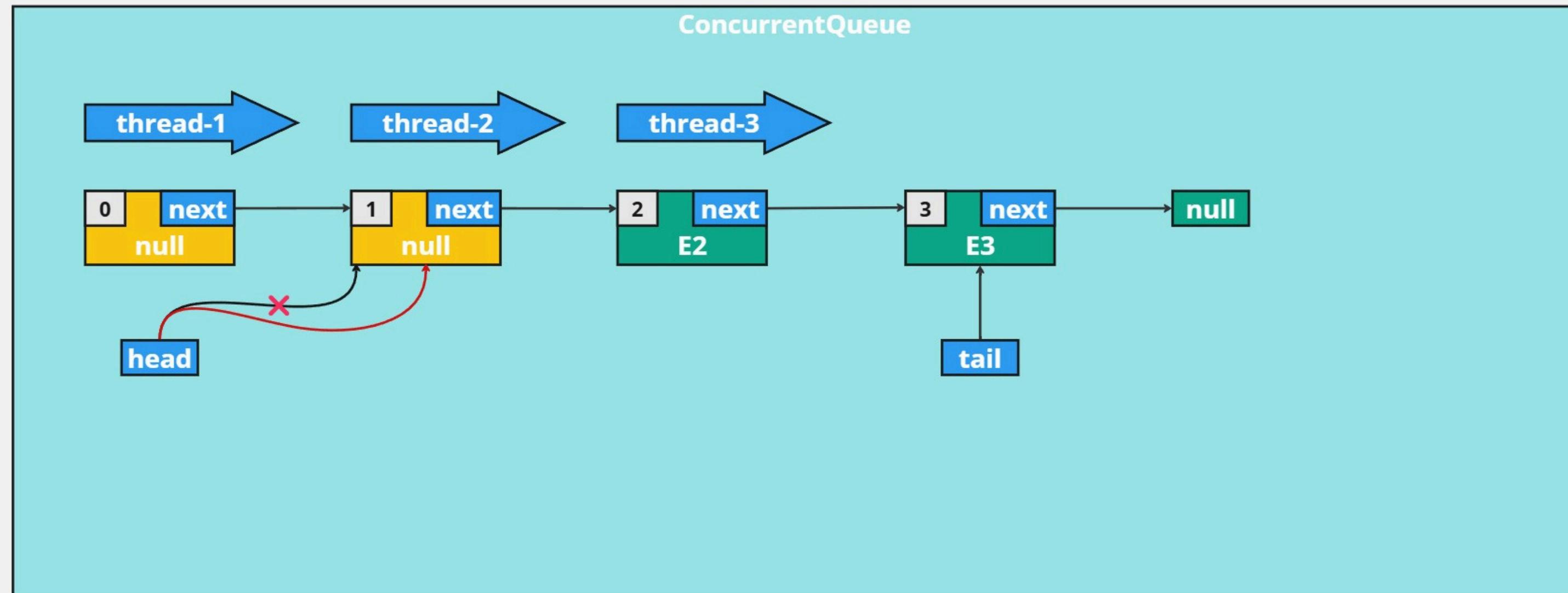
```

### thread-3

```

public Optional<E> dequeue() {
    final Node<E> previousHead = head;
    final Node<E> nextHead = previousHead.next.get();
    if (previousHead == tail.get()) {
        return empty();
    }
    final E element = nextHead.value;
    nextHead.value = null;
    head = nextHead;
    previousHead.next.set(previousHead);
    return Optional.of(element);
}

```



**thread-1**

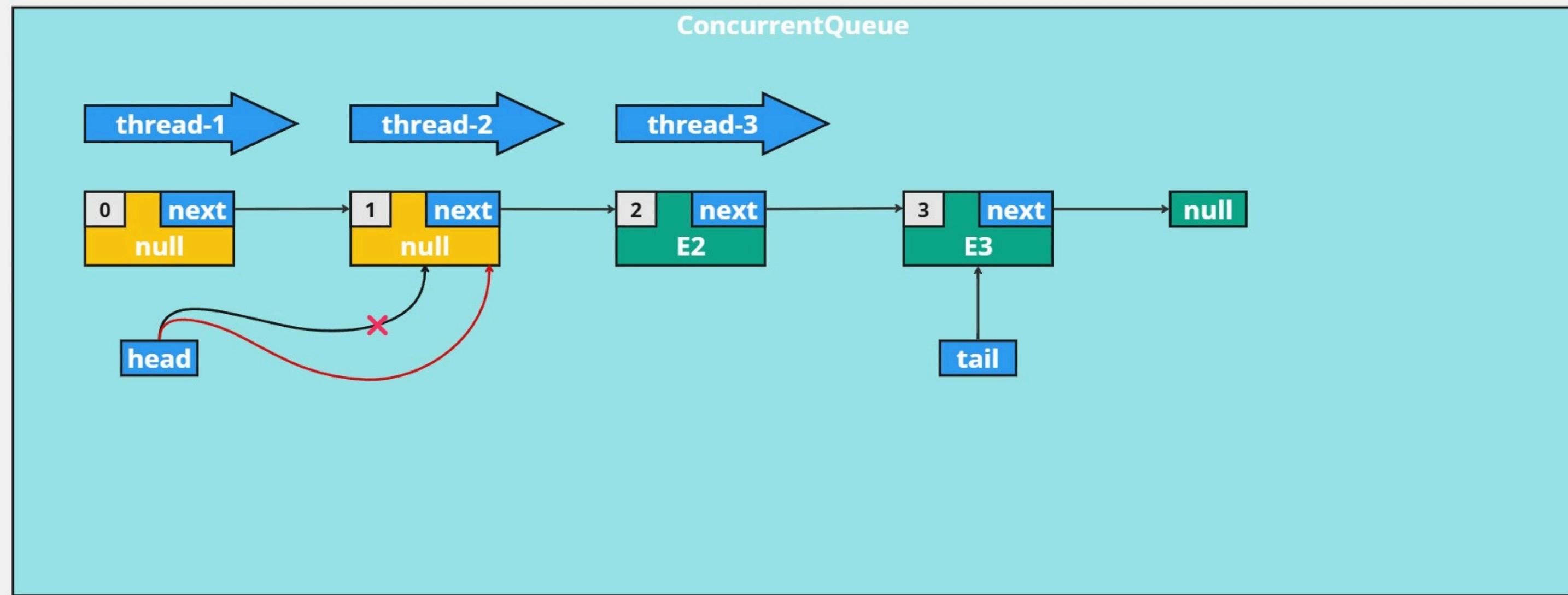
```
public Optional<E> dequeue() {
    final Node<E> previousHead = head;
    final Node<E> nextHead = previousHead.next.get();
    if (previousHead == tail.get()) {
        return empty();
    }
    final E element = nextHead.value;
    nextHead.value = null;
    head = nextHead;
    previousHead.next.set(previousHead);
    return Optional.of(element);
}
```

**thread-2**

```
public Optional<E> dequeue() {
    final Node<E> previousHead = head;
    final Node<E> nextHead = previousHead.next.get();
    if (previousHead == tail.get()) {
        return empty();
    }
    final E element = nextHead.value;
    nextHead.value = null;
    head = nextHead;
    previousHead.next.set(previousHead);
    return Optional.of(element);
}
```

**thread-3**

```
public Optional<E> dequeue() {
    final Node<E> previousHead = head;
    final Node<E> nextHead = previousHead.next.get();
    if (previousHead == tail.get()) {
        return empty();
    }
    final E element = nextHead.value;
    nextHead.value = null;
    head = nextHead;
    previousHead.next.set(previousHead);
    return Optional.of(element);
}
```



### thread-1

```

public Optional<E> dequeue() {
    final Node<E> previousHead = head;
    final Node<E> nextHead = previousHead.next.get();
    if (previousHead == tail.get()) {
        return empty();
    }
    final E element = nextHead.value;
    nextHead.value = null;
    head = nextHead;
    previousHead.next.set(previousHead);
    return Optional.of(element);
}

```

### thread-2

```

public Optional<E> dequeue() {
    final Node<E> previousHead = head;
    final Node<E> nextHead = previousHead.next.get();
    if (previousHead == tail.get()) {
        return empty();
    }
    final E element = nextHead.value;
    nextHead.value = null;
    head = nextHead;
    previousHead.next.set(previousHead);
    return Optional.of(element);
}

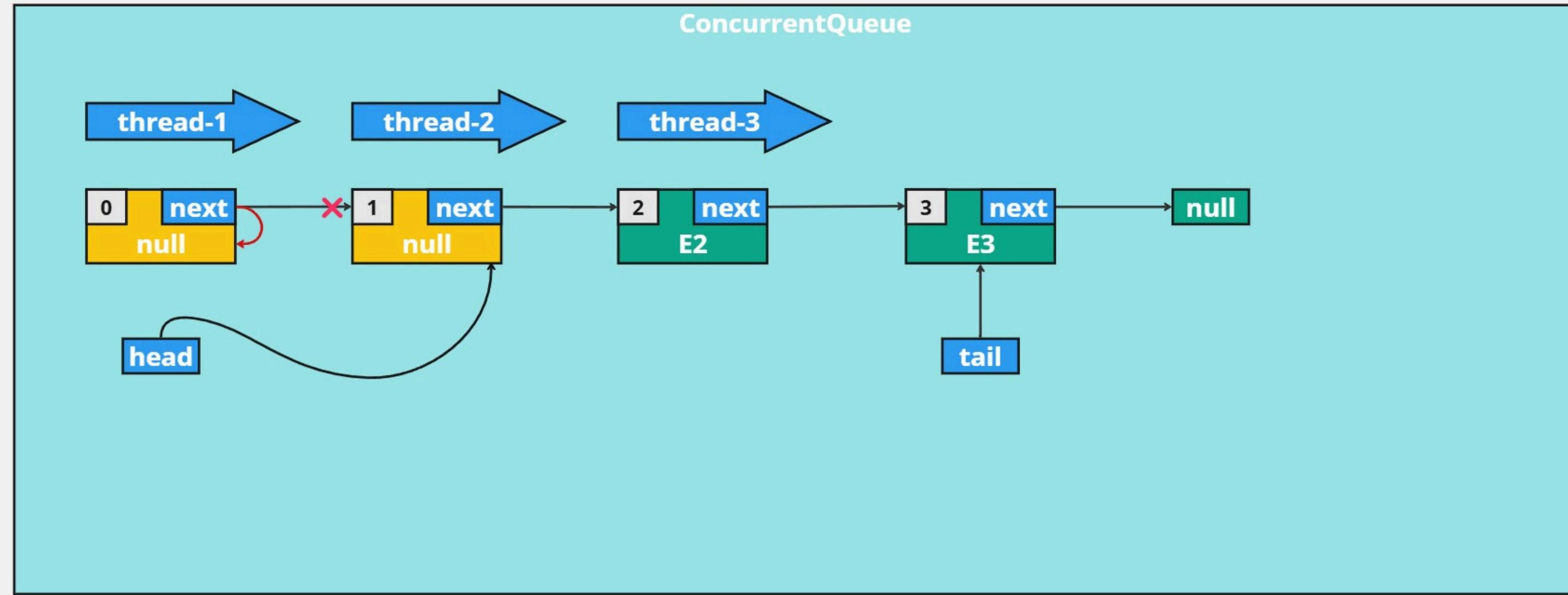
```

### thread-3

```

public Optional<E> dequeue() {
    final Node<E> previousHead = head;
    final Node<E> nextHead = previousHead.next.get();
    if (previousHead == tail.get()) {
        return empty();
    }
    final E element = nextHead.value;
    nextHead.value = null;
    head = nextHead;
    previousHead.next.set(previousHead);
    return Optional.of(element);
}

```



### thread-1

```

public Optional<E> dequeue() {
    final Node<E> previousHead = head;
    final Node<E> nextHead = previousHead.next.get();
    if (previousHead == tail.get()) {
        return empty();
    }
    final E element = nextHead.value;
    nextHead.value = null;
    head = nextHead;
    previousHead.next.set(previousHead);
    return Optional.of(element);
}

```

### thread-2

```

public Optional<E> dequeue() {
    final Node<E> previousHead = head;
    final Node<E> nextHead = previousHead.next.get();
    if (previousHead == tail.get()) {
        return empty();
    }
    final E element = nextHead.value;
    nextHead.value = null;
    head = nextHead;
    previousHead.next.set(previousHead);
    return Optional.of(element);
}

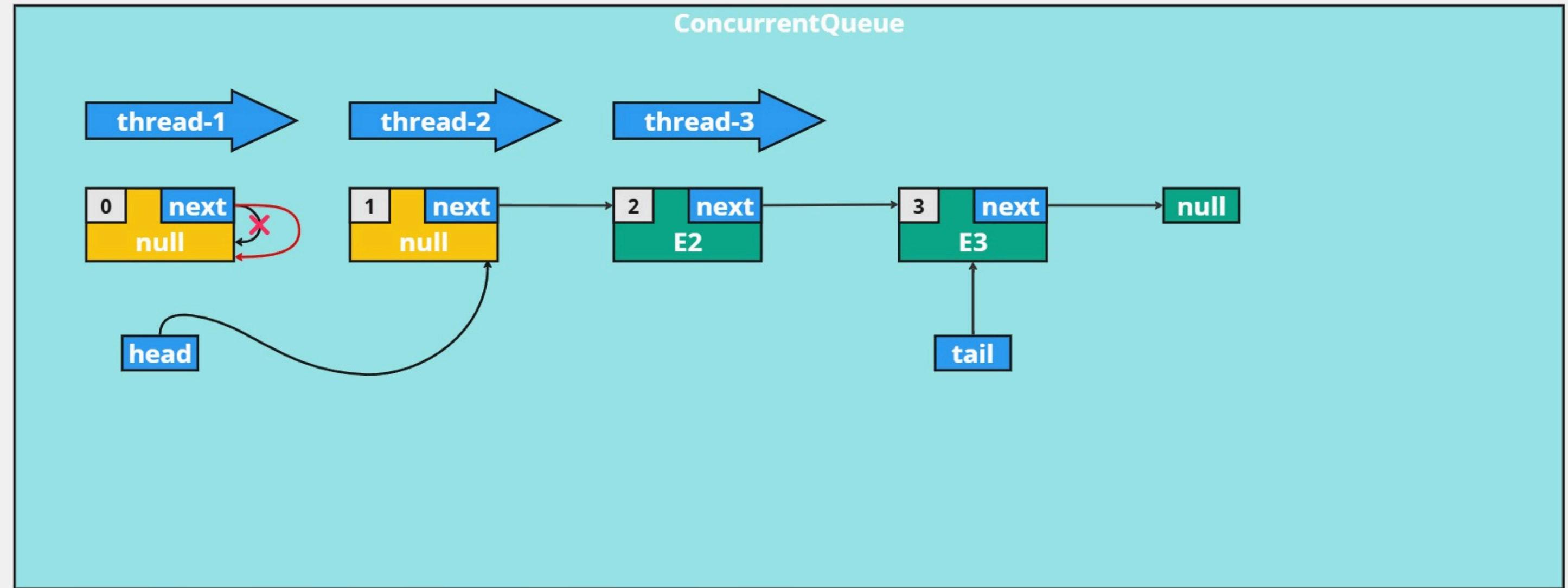
```

### thread-3

```

public Optional<E> dequeue() {
    final Node<E> previousHead = head;
    final Node<E> nextHead = previousHead.next.get();
    if (previousHead == tail.get()) {
        return empty();
    }
    final E element = nextHead.value;
    nextHead.value = null;
    head = nextHead;
    previousHead.next.set(previousHead);
    return Optional.of(element);
}

```



### thread-1

```

public Optional<E> dequeue() {
    final Node<E> previousHead = head;
    final Node<E> nextHead = previousHead.next.get();
    if (previousHead == tail.get()) {
        return empty();
    }
    final E element = nextHead.value;
    nextHead.value = null;
    head = nextHead;
    previousHead.next.set(previousHead);
    return Optional.of(element);
}

```

### thread-2

```

public Optional<E> dequeue() {
    final Node<E> previousHead = head;
    final Node<E> nextHead = previousHead.next.get();
    if (previousHead == tail.get()) {
        return empty();
    }
    final E element = nextHead.value;
    nextHead.value = null;
    head = nextHead;
    previousHead.next.set(previousHead);
    return Optional.of(element);
}

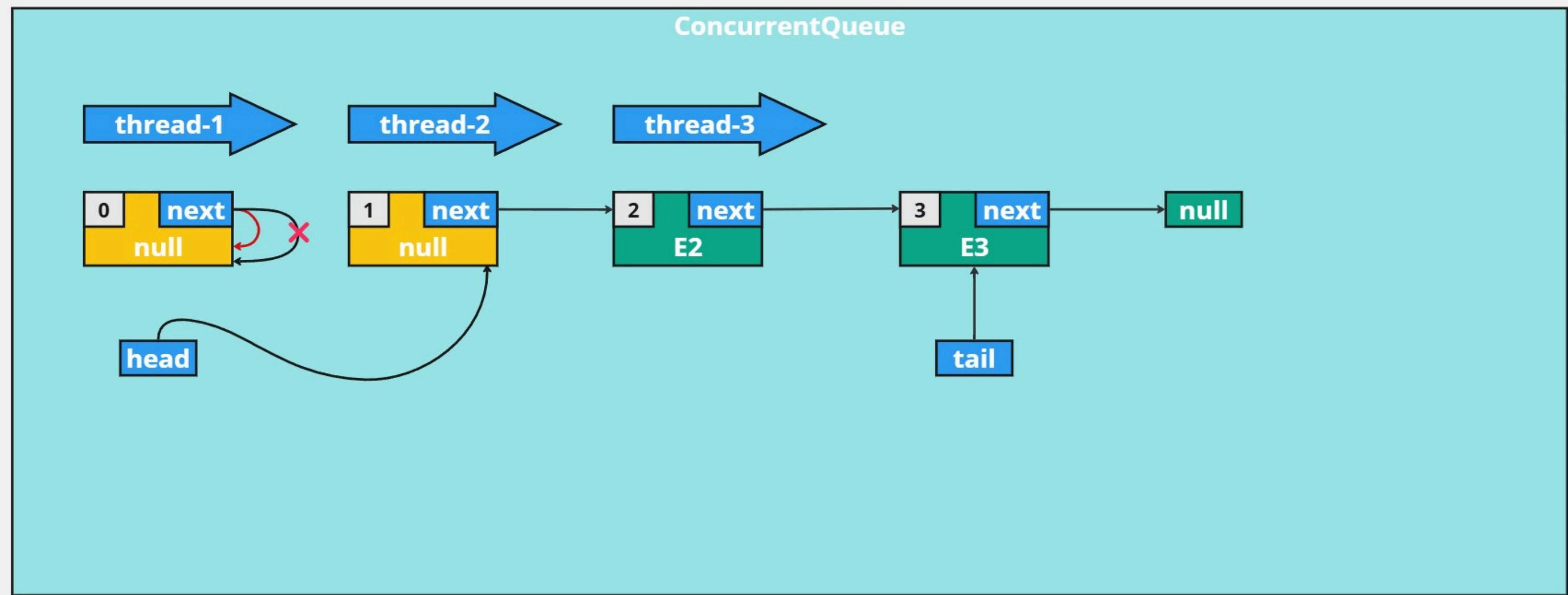
```

### thread-3

```

public Optional<E> dequeue() {
    final Node<E> previousHead = head;
    final Node<E> nextHead = previousHead.next.get();
    if (previousHead == tail.get()) {
        return empty();
    }
    final E element = nextHead.value;
    nextHead.value = null;
    head = nextHead;
    previousHead.next.set(previousHead);
    return Optional.of(element);
}

```



**thread-1**

```

public Optional<E> dequeue() {
    final Node<E> previousHead = head;
    final Node<E> nextHead = previousHead.next.get();
    if (previousHead == tail.get()) {
        return empty();
    }
    final E element = nextHead.value;
    nextHead.value = null;
    head = nextHead;
    previousHead.next.set(previousHead);
    return Optional.of(element);
}

```

**thread-2**

```

public Optional<E> dequeue() {
    final Node<E> previousHead = head;
    final Node<E> nextHead = previousHead.next.get();
    if (previousHead == tail.get()) {
        return empty();
    }
    final E element = nextHead.value;
    nextHead.value = null;
    head = nextHead;
    previousHead.next.set(previousHead);
    return Optional.of(element);
}

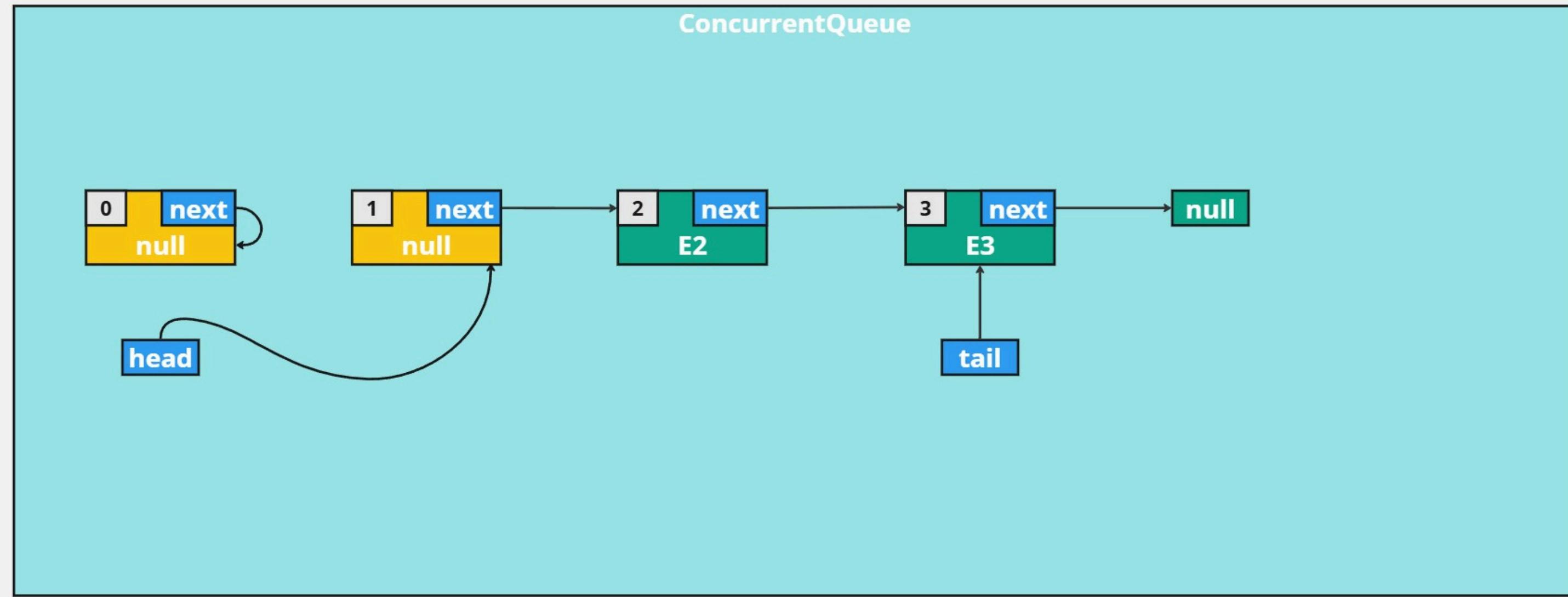
```

**thread-3**

```

public Optional<E> dequeue() {
    final Node<E> previousHead = head;
    final Node<E> nextHead = previousHead.next.get();
    if (previousHead == tail.get()) {
        return empty();
    }
    final E element = nextHead.value;
    nextHead.value = null;
    head = nextHead;
    previousHead.next.set(previousHead);
    return Optional.of(element);
}

```



**thread-1**

```
public Optional<E> dequeue() {
    final Node<E> previousHead = head;
    final Node<E> nextHead = previousHead.next.get();
    if (previousHead == tail.get()) {
        return empty();
    }
    final E element = nextHead.value;
    nextHead.value = null;
    head = nextHead;
    previousHead.next.set(previousHead);
    return Optional.of(element);
}
```

**thread-2**

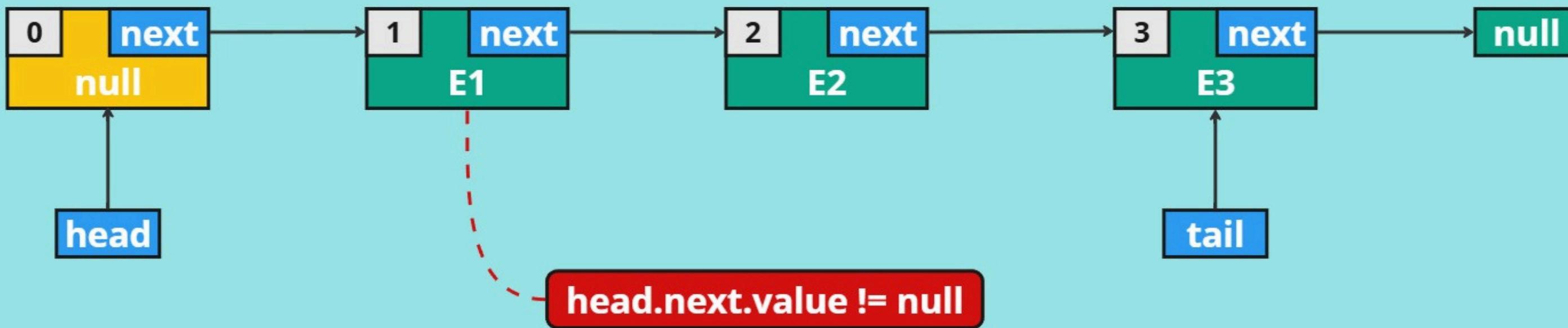
```
public Optional<E> dequeue() {
    final Node<E> previousHead = head;
    final Node<E> nextHead = previousHead.next.get();
    if (previousHead == tail.get()) {
        return empty();
    }
    final E element = nextHead.value;
    nextHead.value = null;
    head = nextHead;
    previousHead.next.set(previousHead);
    return Optional.of(element);
}
```

**thread-3**

```
public Optional<E> dequeue() {
    final Node<E> previousHead = head;
    final Node<E> nextHead = previousHead.next.get();
    if (previousHead == tail.get()) {
        return empty();
    }
    final E element = nextHead.value;
    nextHead.value = null;
    head = nextHead;
    previousHead.next.set(previousHead);
    return Optional.of(element);
}
```

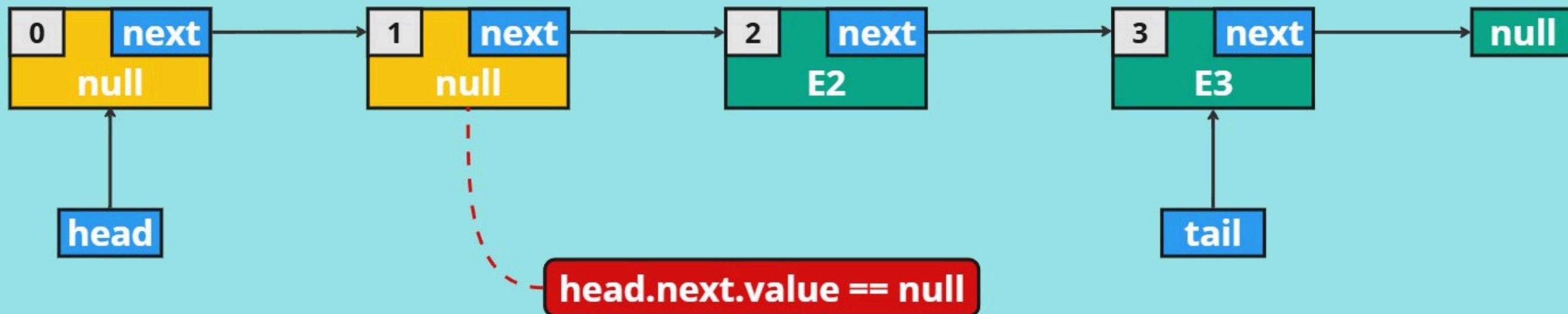
## *Consistency state*

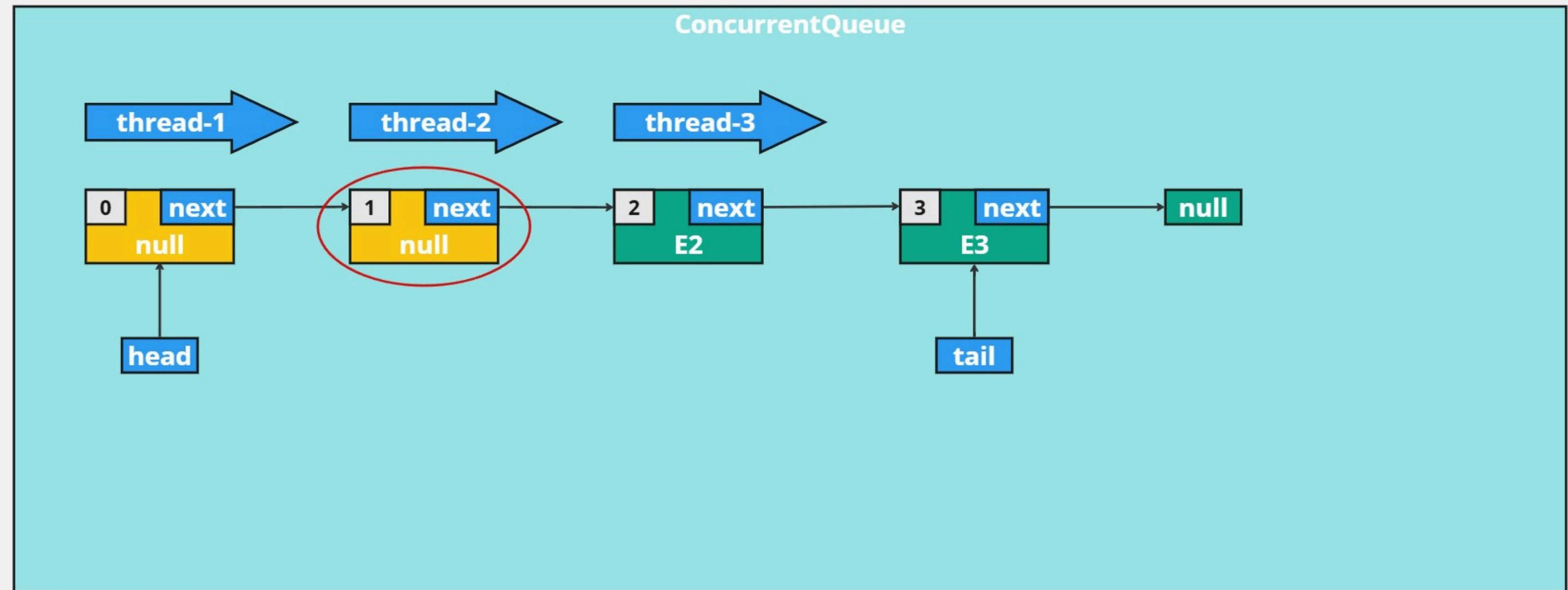
ConcurrentQueue



## *Not consistency state*

ConcurrentQueue





### thread-1

```

public Optional<E> dequeue() {
    final Node<E> previousHead = head;
    final Node<E> nextHead = previousHead.next.get();
    if (previousHead == tail.get()) {
        return empty();
    }
    final E element = nextHead.value;
    nextHead.value = null;
    head = nextHead;
    previousHead.next.set(previousHead);
    return Optional.of(element);
}

```

### thread-2

```

public Optional<E> dequeue() {
    final Node<E> previousHead = head;
    final Node<E> nextHead = previousHead.next.get();
    if (previousHead == tail.get()) {
        return empty();
    }
    final E element = nextHead.value;
    nextHead.value = null;
    head = nextHead;
    previousHead.next.set(previousHead);
    return Optional.of(element);
}

```

### thread-3

```

public Optional<E> dequeue() {
    final Node<E> previousHead = head;
    final Node<E> nextHead = previousHead.next.get();
    if (previousHead == tail.get()) {
        return empty();
    }
    final E element = nextHead.value;
    nextHead.value = null;
    head = nextHead;
    previousHead.next.set(previousHead);
    return Optional.of(element);
}

```

```
public Optional<E> dequeue() {  
    final Node<E> previousHead = head;  
    final Node<E> nextHead = previousHead.next.get();  
    if (previousHead == tail.get()) {  
        return empty();  
    }  
    final E element = nextHead.value;  
    nextHead.value = null;  
    head = nextHead;  
    previousHead.next.set(previousHead);  
    return Optional.of(element);  
}
```

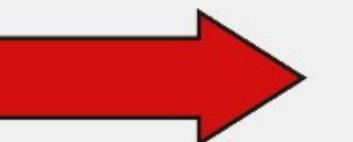


```
public Optional<E> dequeue() {  
    while (true) {  
        final Node<E> previousHead = head;  
        final Node<E> nextHead = previousHead.next.get();  
        if (previousHead == tail.get()) {  
            return empty();  
        }  
        final E element = nextHead.value;  
        if (element != null) {  
            nextHead.value = null;  
            head = nextHead;  
            previousHead.next.set(previousHead);  
            return Optional.of(element);  
        }  
    }  
}
```

```
public Optional<E> dequeue() {  
    while (true) {  
        final Node<E> previousHead = head;  
        final Node<E> nextHead = previousHead.next.get();  
        if (previousHead == tail.get()) {  
            return empty();  
        }  
        final E element = nextHead.value;  
        if (element != null) {  
            nextHead.value = null;  
            head = nextHead;  
            previousHead.next.set(previousHead);  
            return Optional.of(element);  
        }  
    }  
}
```

*not atomic*

```
private static final class Node<E> {  
    1 usage  
    private volatile E value;  
    3 usages  
    private final AtomicReference<Node<E>> next;  
  
    1 usage  
    public Node() {  
        next = new AtomicReference<>();  
    }  
  
    1 usage  
    public Node(final E value) {  
        this.value = value;  
        next = new AtomicReference<>();  
    }  
}
```



```
private static final class Node<E> {  
    4 usages  
    private final AtomicReference<E> value;  
    6 usages  
    private final AtomicReference<Node<E>> next;  
  
    1 usage Vlad Zuev  
    public Node() {  
        value = new AtomicReference<>();  
        next = new AtomicReference<>();  
    }  
  
    1 usage Vlad Zuev  
    public Node(final E value) {  
        this.value = new AtomicReference<>(value);  
        next = new AtomicReference<>();  
    }  
}
```

```
public Optional<E> dequeue() {  
    while (true) {  
        final Node<E> previousHead = head;  
        final Node<E> nextHead = previousHead.next.get();  
        if (previousHead == tail.get()) {  
            return empty();  
        }  
        final E element = nextHead.value;  
        if (element != null) {  
            nextHead.value = null;  
            head = nextHead;  
            previousHead.next.set(previousHead);  
            return Optional.of(element);  
        }  
    }  
}
```

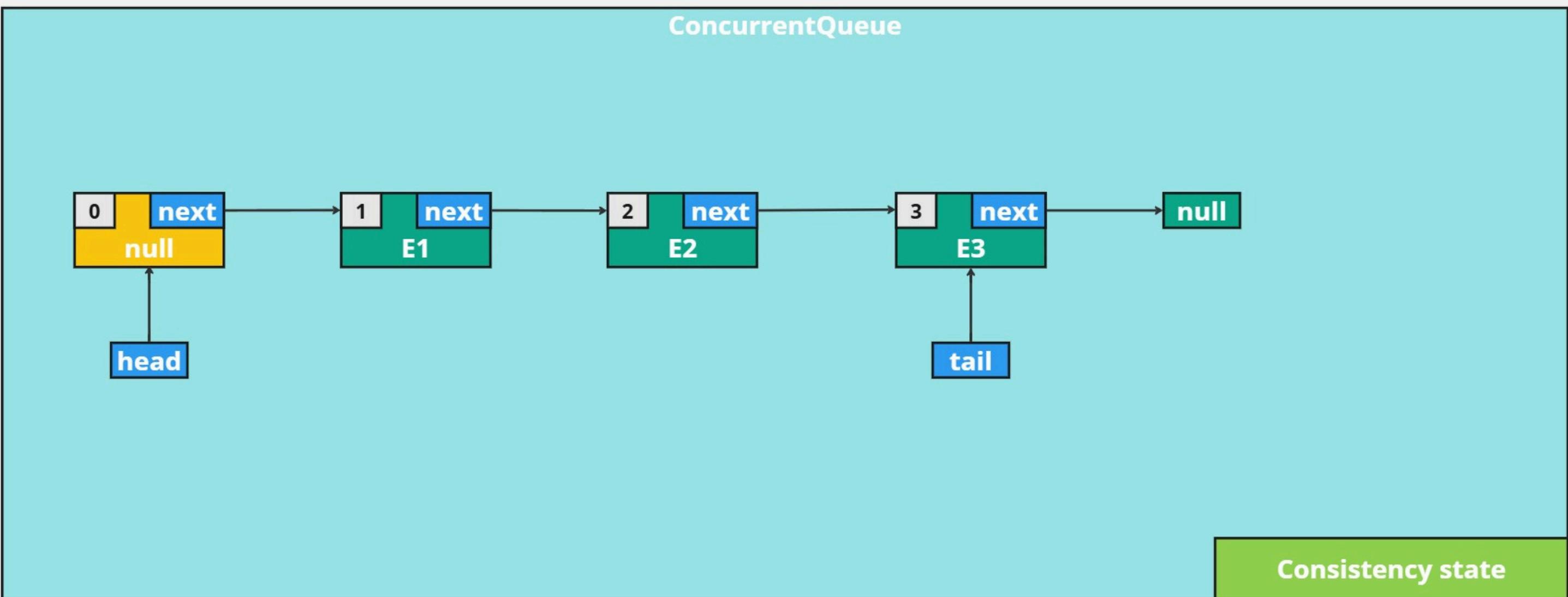


```
public Optional<E> dequeue() {  
    while (true) {  
        final Node<E> previousHead = head;  
        final Node<E> nextHead = previousHead.next.get();  
        if (previousHead == tail.get()) {  
            return empty();  
        }  
        final E element = nextHead.value.get();  
        if (element != null && nextHead.value.compareAndSet(element, null)) {  
            head = nextHead;  
            previousHead.next.set(previousHead);  
            return of(element);  
        }  
    }  
}
```

atomic

## ConcurrentQueue

thread-1 →  
thread-2 →  
thread-3 →



### thread-1

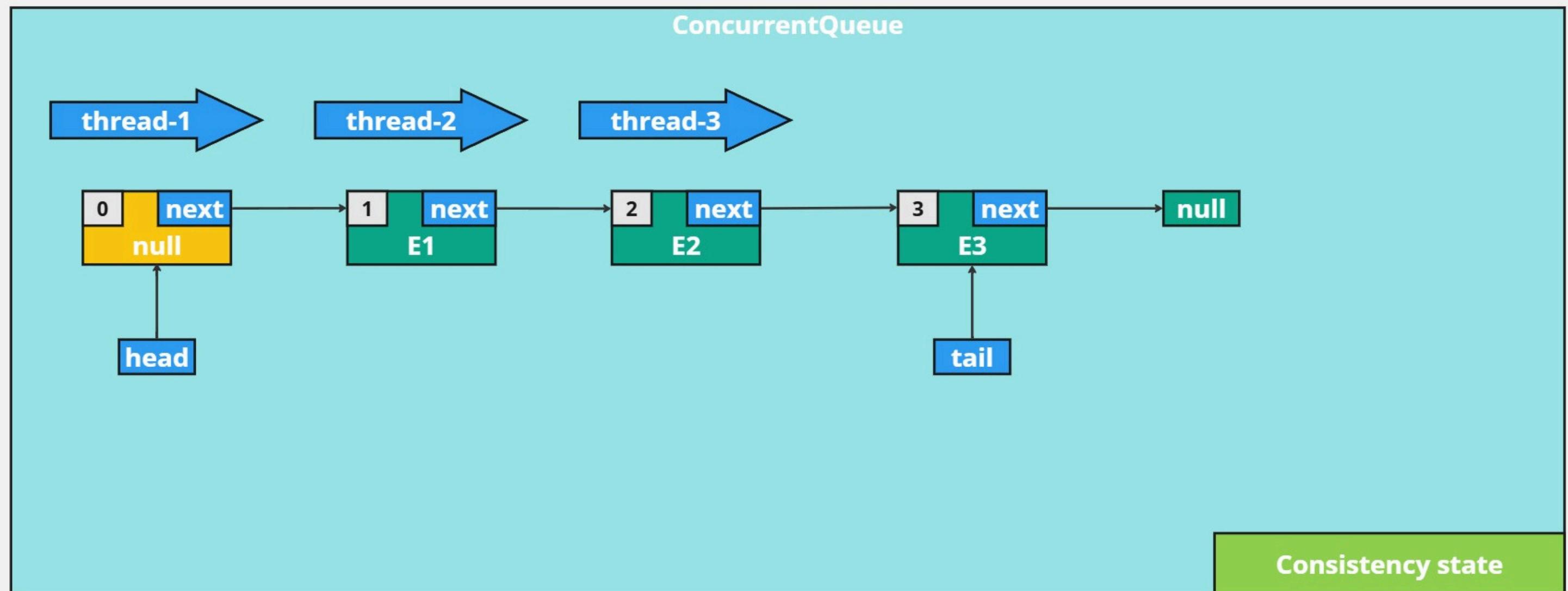
```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

### thread-2

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

### thread-3

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```



### thread-1

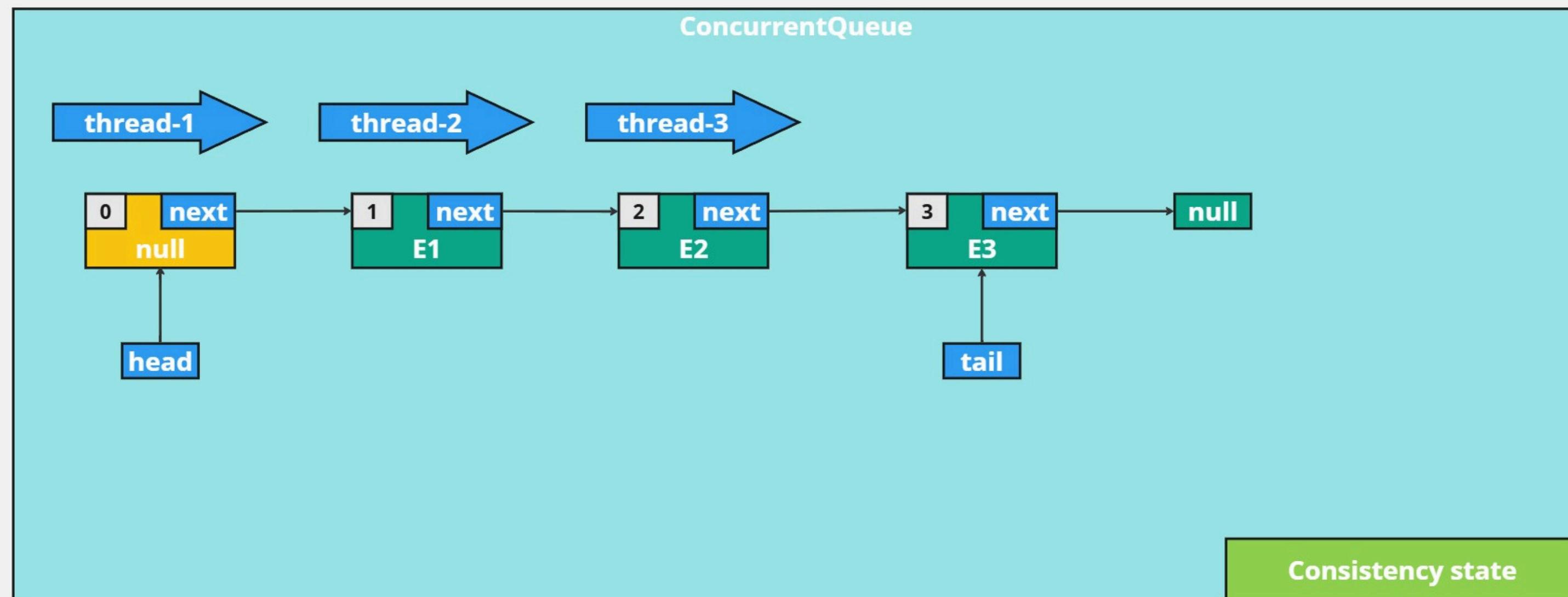
```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

### thread-2

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

### thread-3

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```



### thread-1

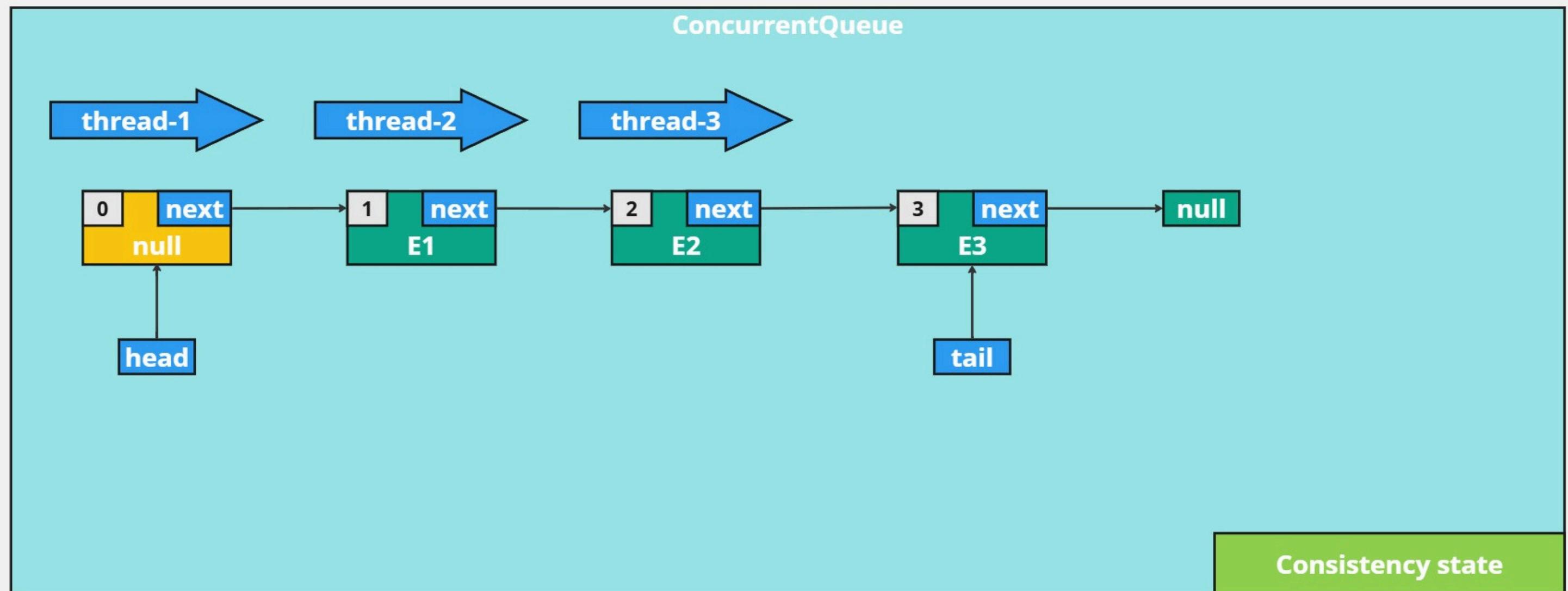
```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

### thread-2

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

### thread-3

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```



### thread-1

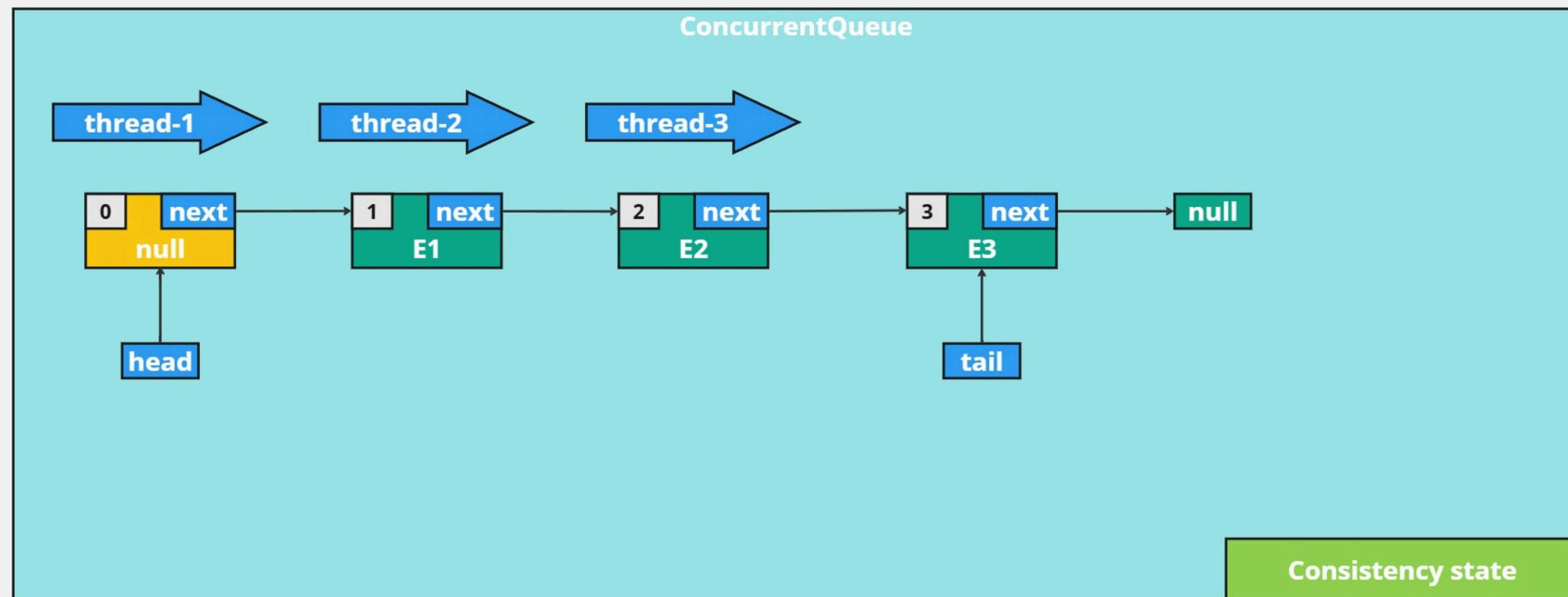
```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

### thread-2

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

### thread-3

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```



### thread-1

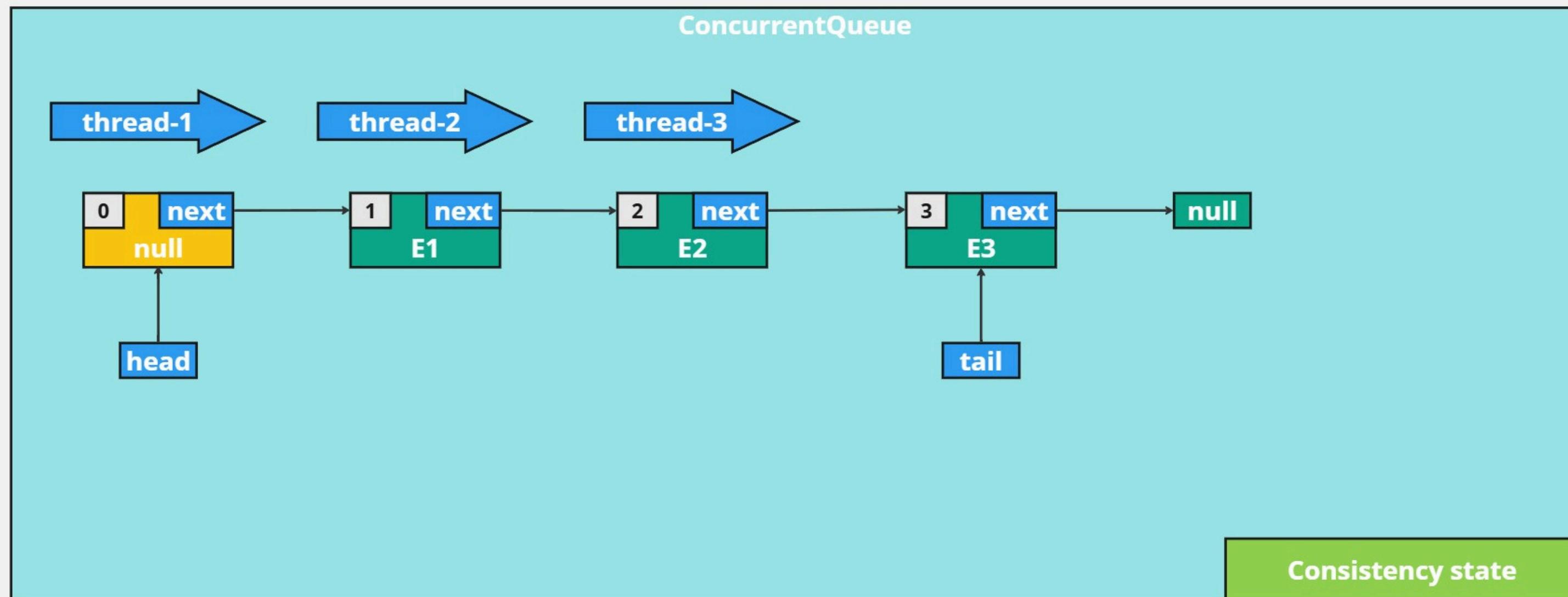
```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

### thread-2

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

### thread-3

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```



### thread-1

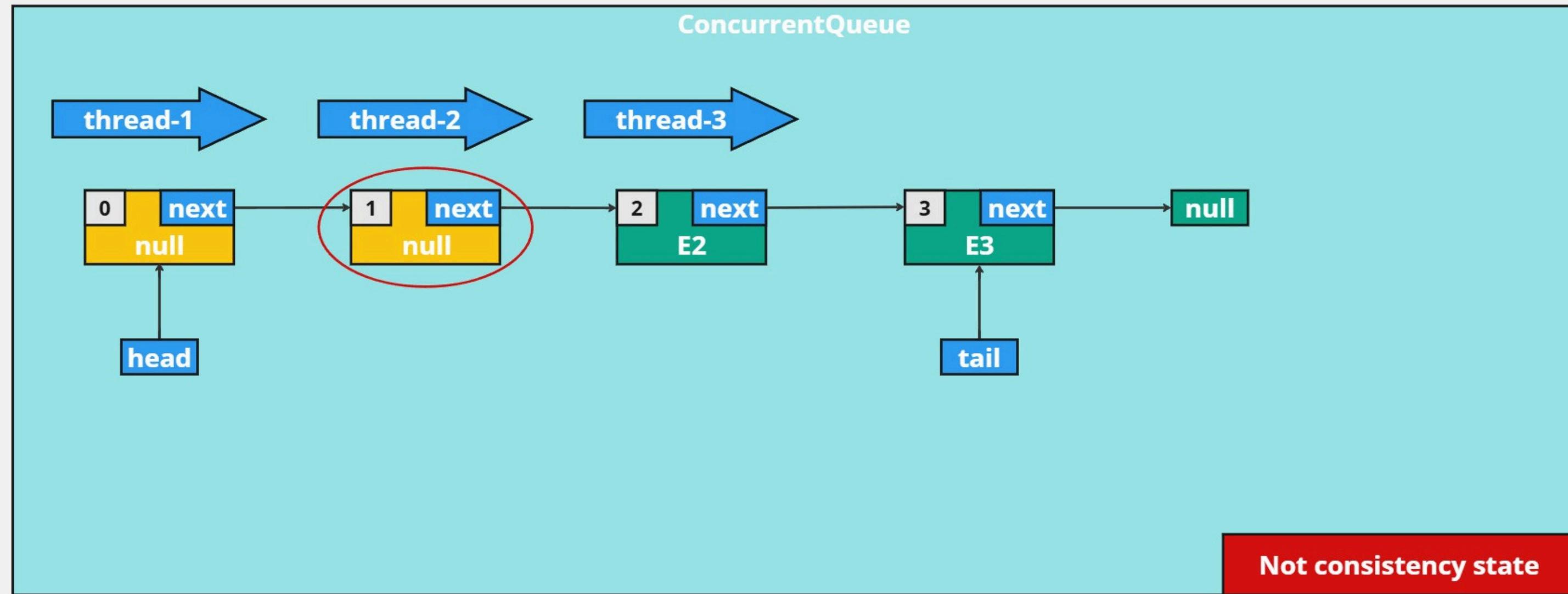
```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

### thread-2

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

### thread-3

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```



### thread-1

```

public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}

```

### thread-2

```

public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}

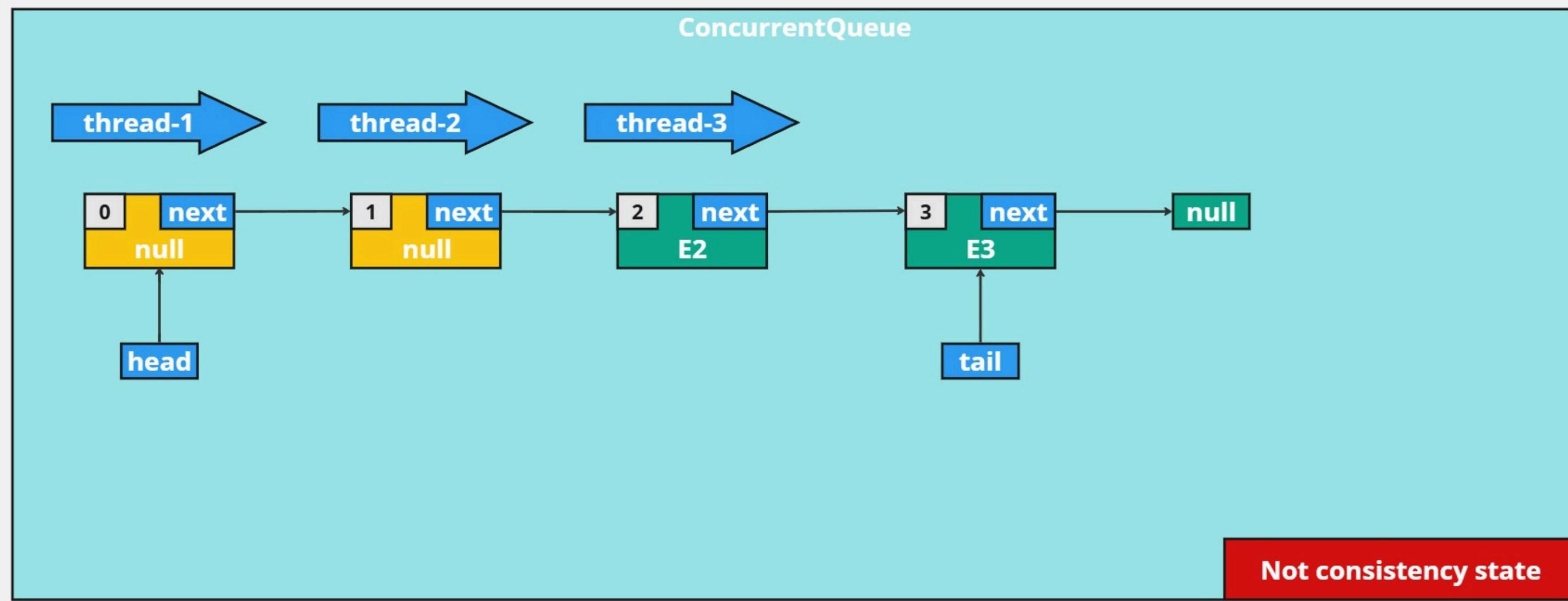
```

### thread-3

```

public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}

```



**thread-1**

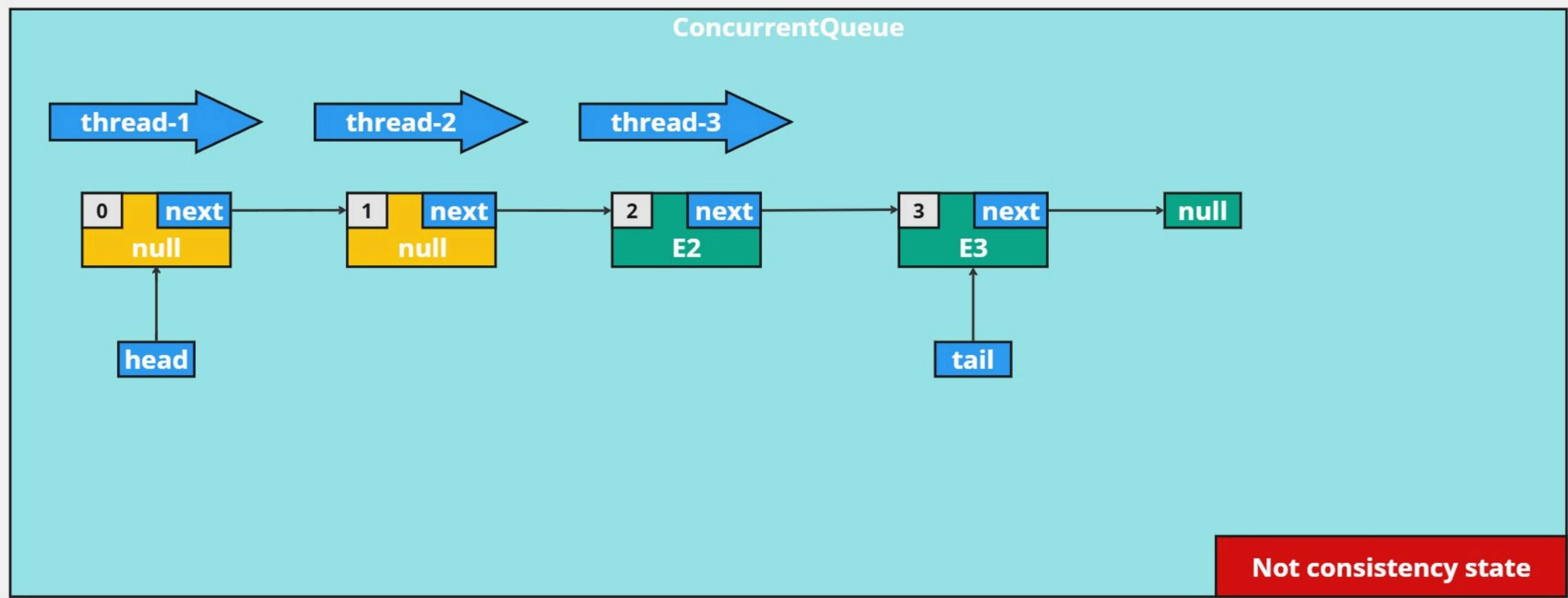
```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

**thread-2**

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

**thread-3**

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```



### thread-1

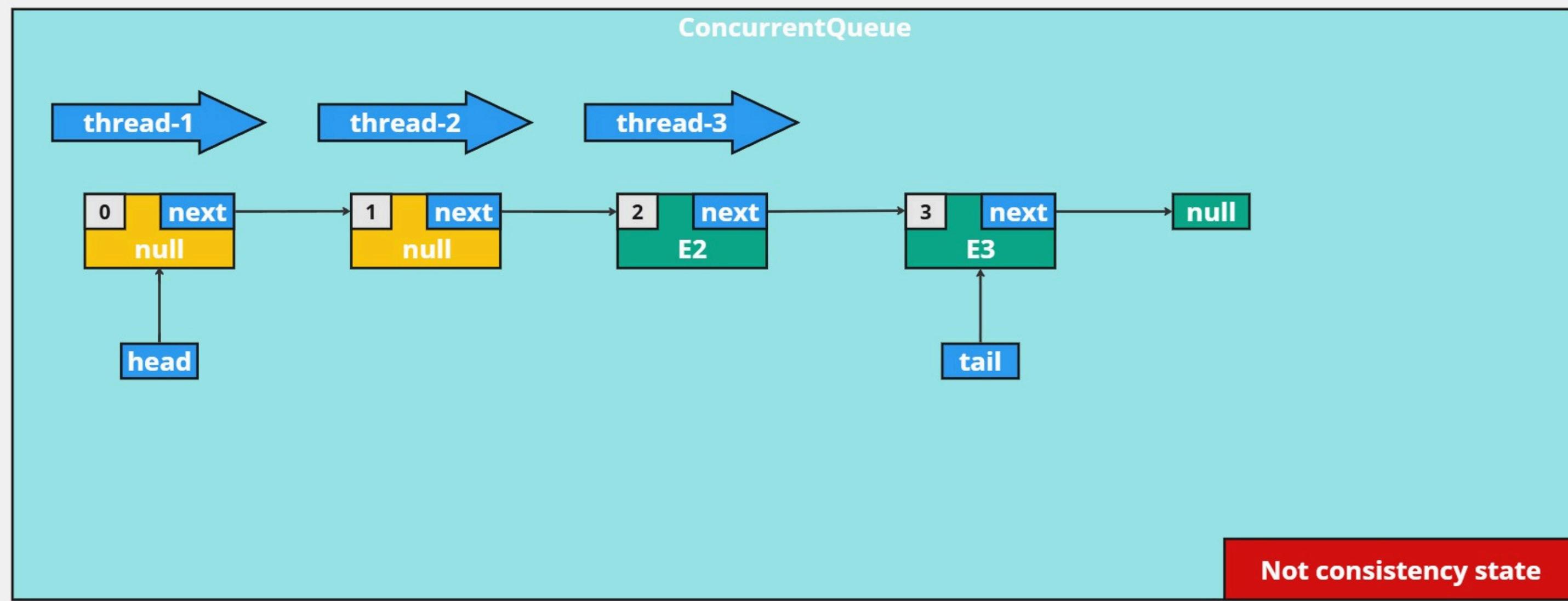
```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

### thread-2

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

### thread-3

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```



### thread-1

```

public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
  
```

Annotations:

- Line 1: Number 1
- Line 2: Box 0
- Line 5: Box E1

### thread-2

```

public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
  
```

Annotations:

- Line 1: Red bracket spanning lines 1-3 of thread-1
- Line 2: Red bracket spanning lines 1-3 of thread-2
- Line 5: Red bracket spanning lines 1-3 of thread-3
- Line 6: Box E1

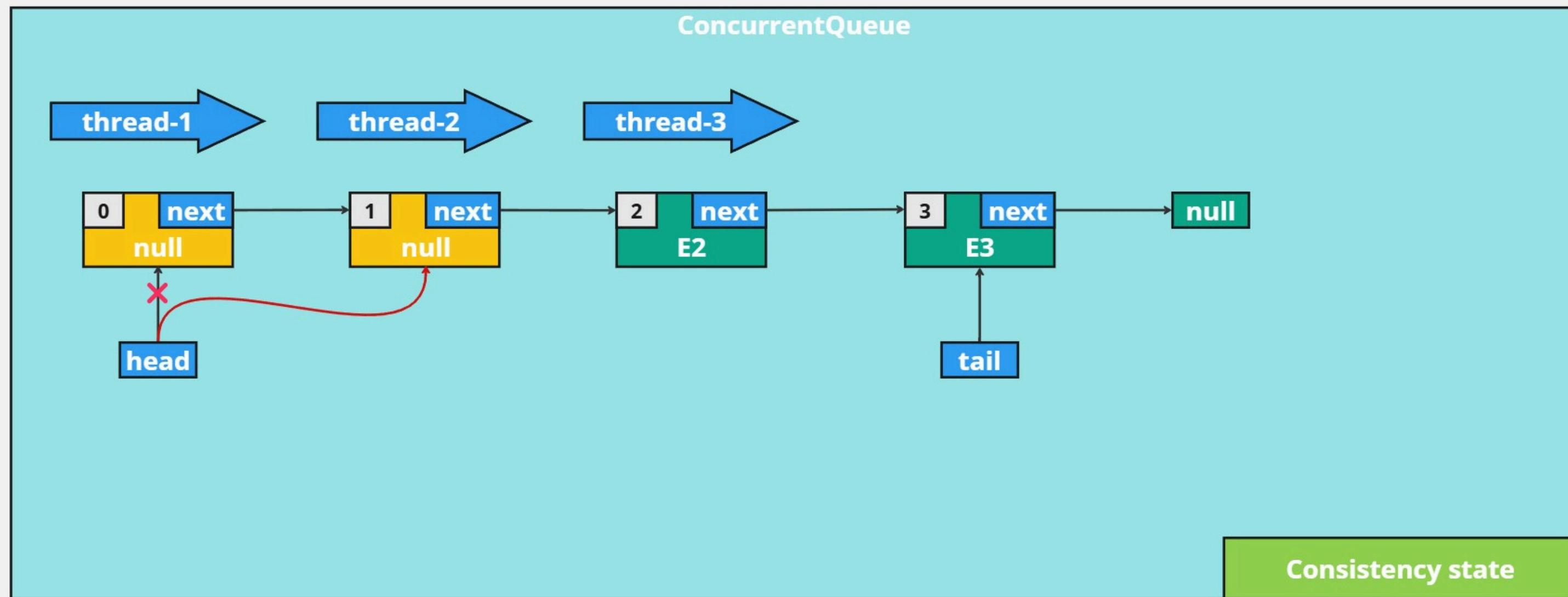
### thread-3

```

public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
  
```

Annotations:

- Line 1: Red bracket spanning lines 1-3 of thread-1
- Line 2: Red bracket spanning lines 1-3 of thread-2
- Line 3: Red bracket spanning lines 1-3 of thread-3



### thread-1

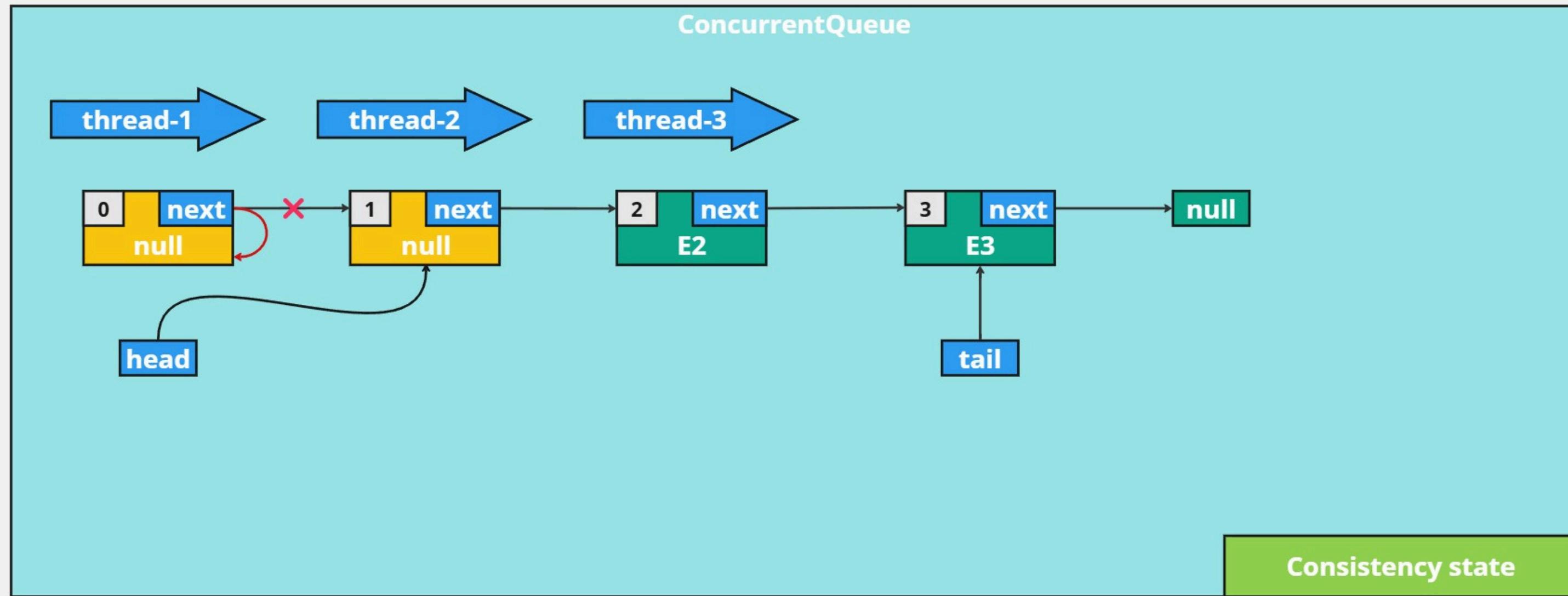
```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

### thread-2

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

### thread-3

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```



### thread-1

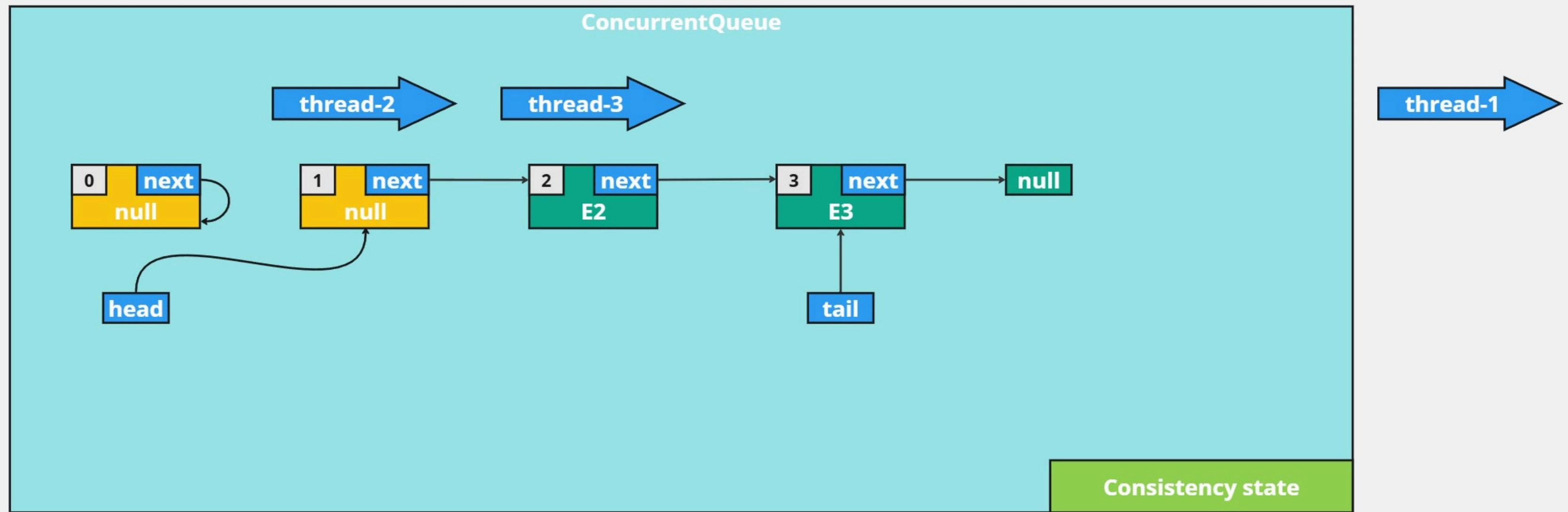
```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

### thread-2

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

### thread-3

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```



### thread-1

```

public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}

```

Annotations:

- Line 1: Box 0
- Line 1: Box 1
- Line 2: Box E1
- Line 10: Box E1

### thread-2

```

public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}

```

Annotations:

- Line 1: Box 0
- Line 1: Box 1
- Line 2: Box E1
- Line 10: Box E1

### thread-3

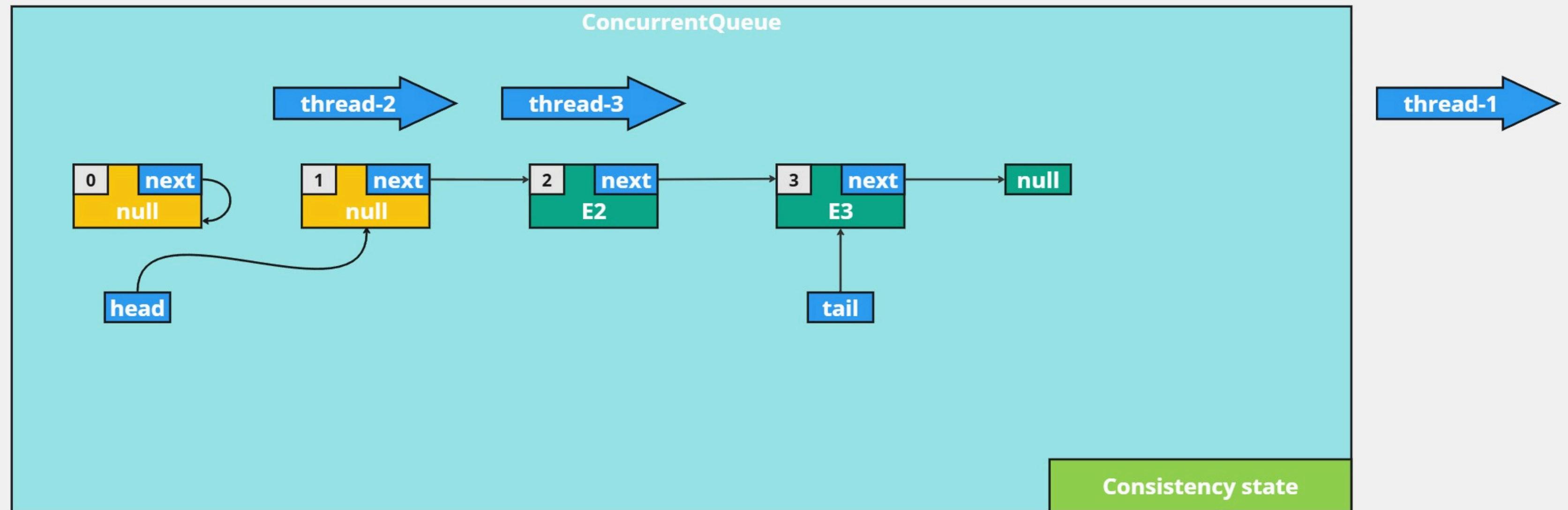
```

public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}

```

Annotations:

- Line 1: Box 0
- Line 1: Box 1
- Line 2: Box E1
- Line 10: Box E1



### thread-1

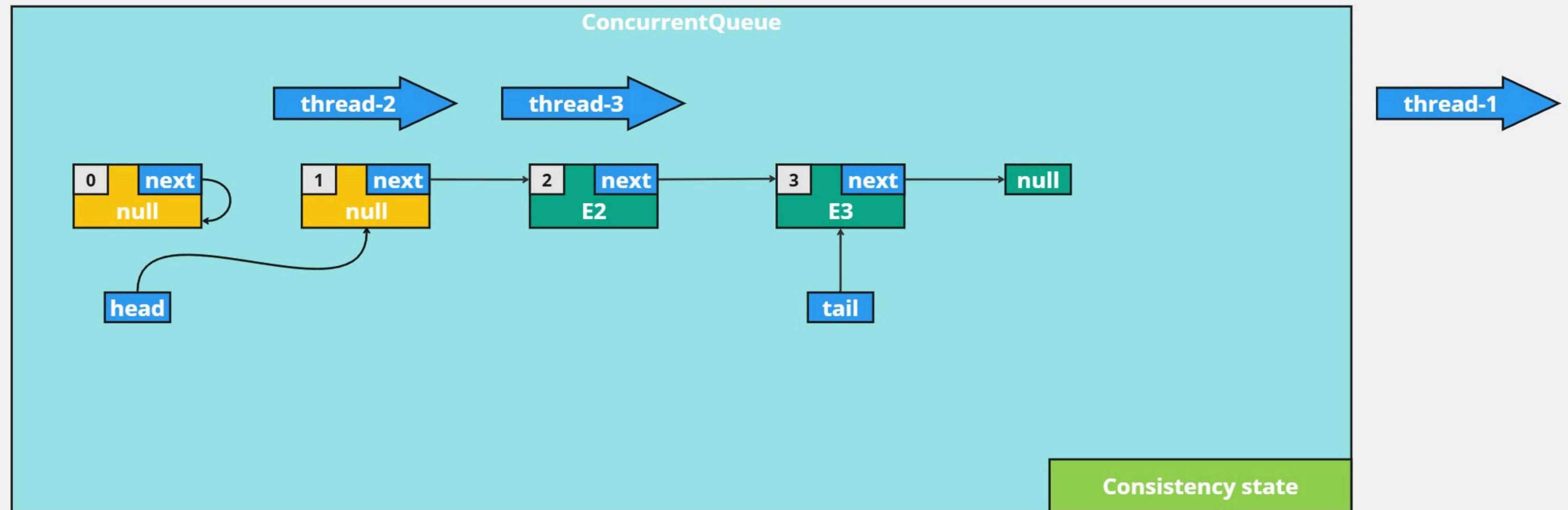
```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

### thread-2

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head; 1
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

### thread-3

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head; 1
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```



### thread-1

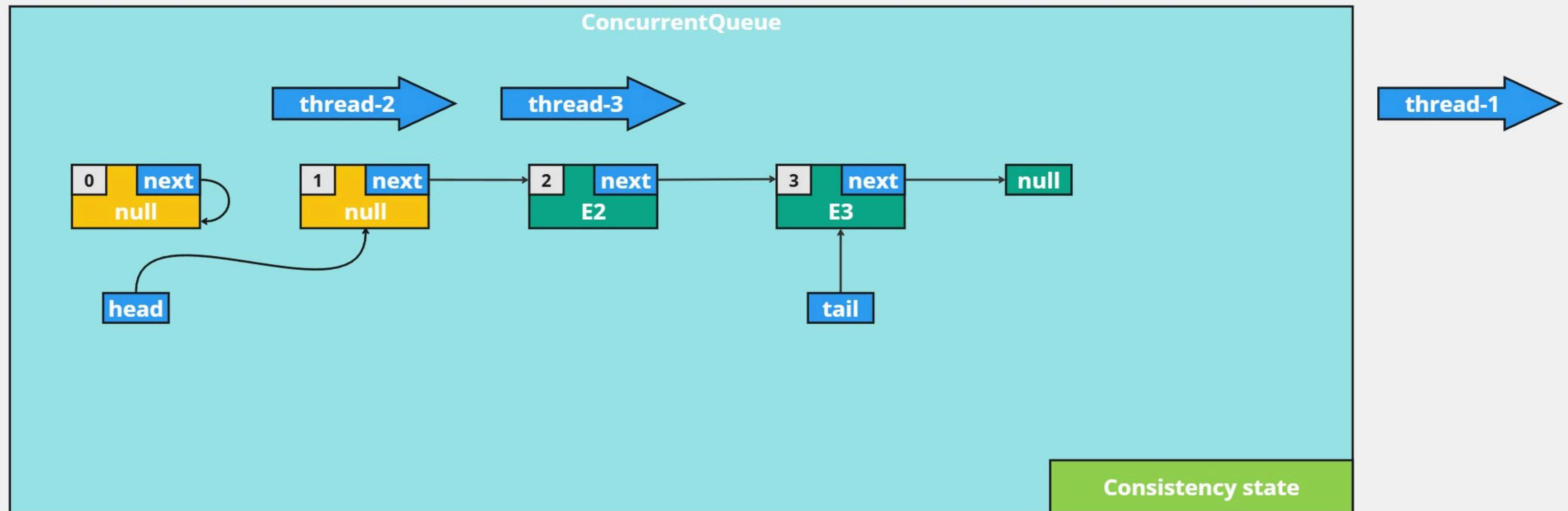
```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

### thread-2

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get(); 1
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) { 2
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

### thread-3

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get(); 1
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) { 2
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```



### thread-1

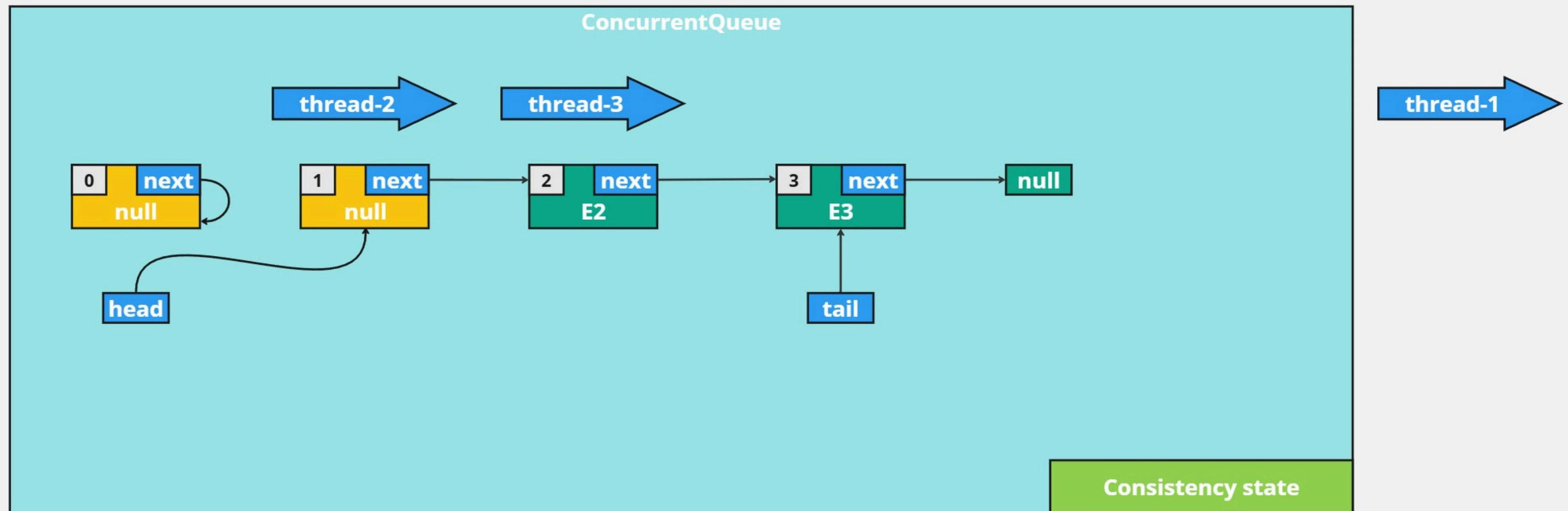
```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

### thread-2

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) { 1
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) { 2
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

### thread-3

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get(); 1
        if (previousHead == tail.get()) { 2
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```



### thread-1

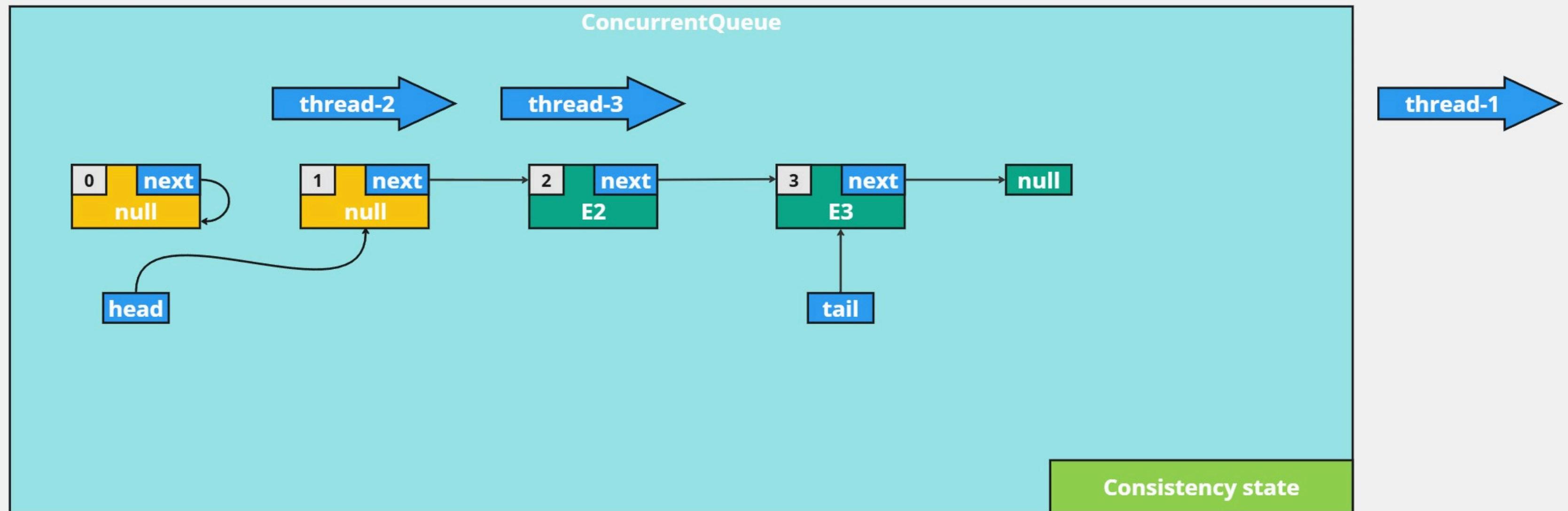
```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

### thread-2

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get(); E2
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

### thread-3

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get(); E2
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```



### thread-1

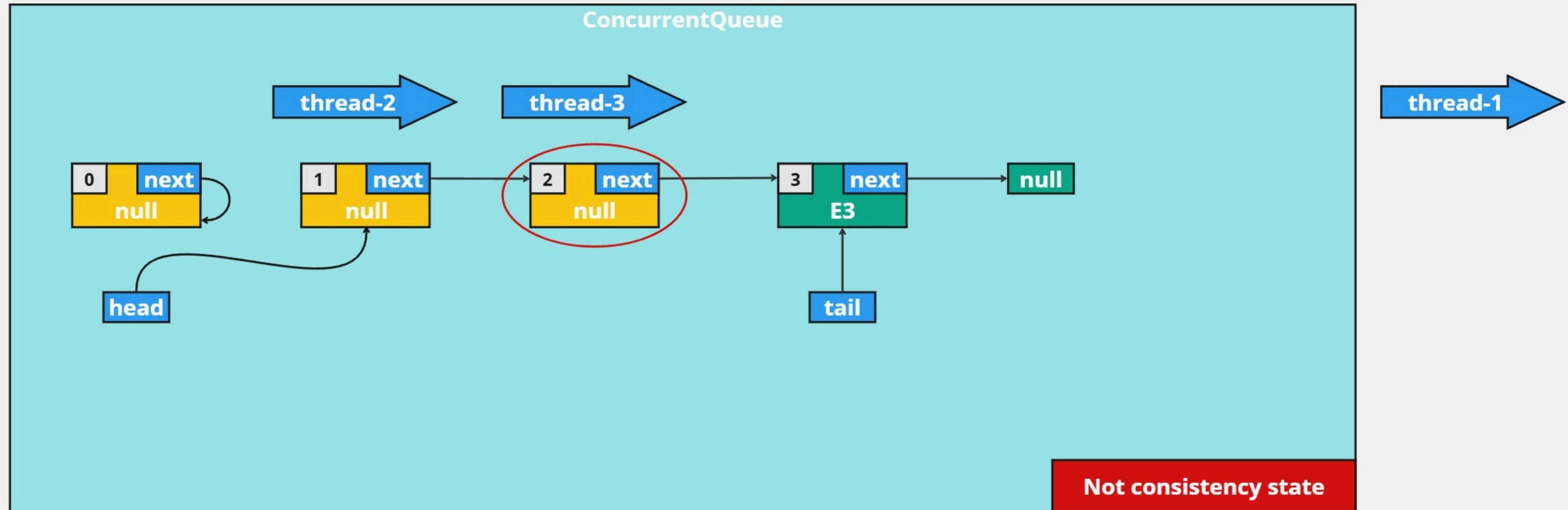
```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

### thread-2

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

### thread-3

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```



### thread-1

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

### thread-2

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

1

2

E2

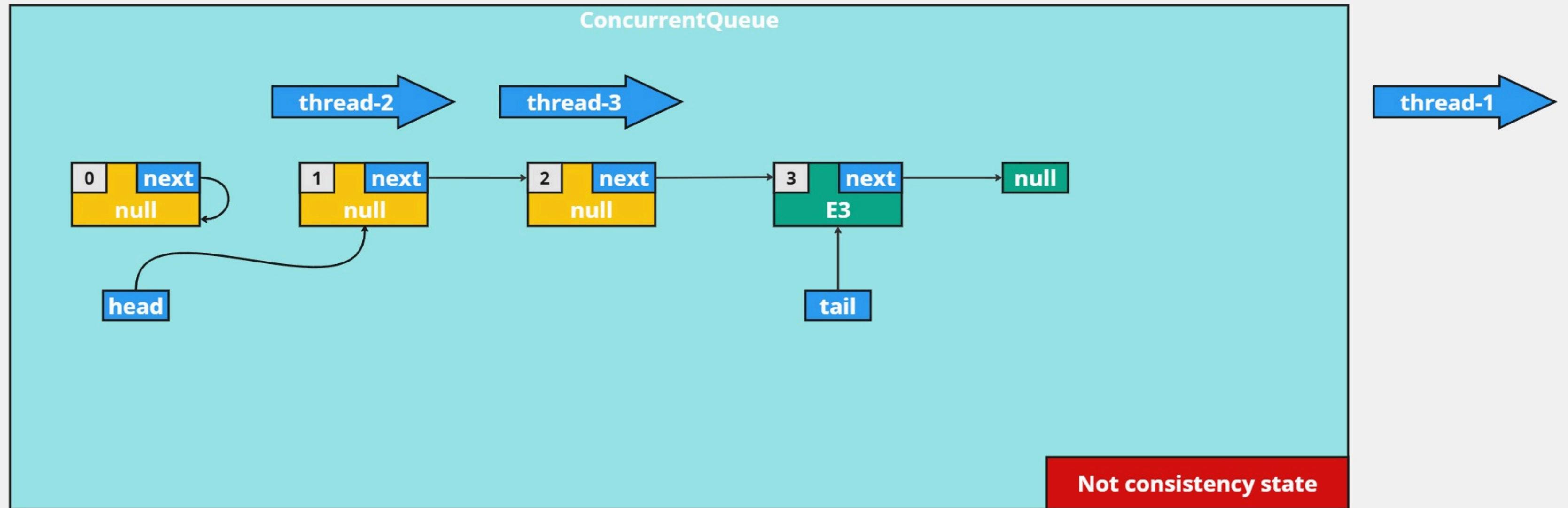
### thread-3

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

1

2

E2



### thread-1

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

### thread-2

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

### thread-3

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

1

2

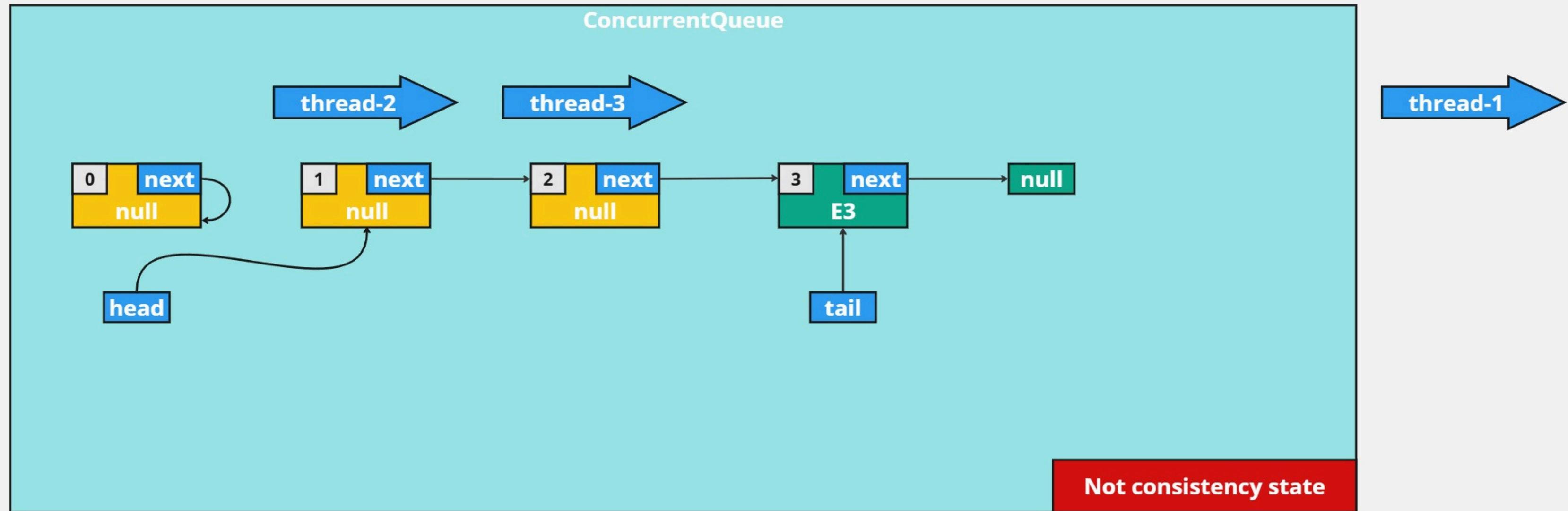
1

E2

2

1

E2



### thread-1

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

### thread-2

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

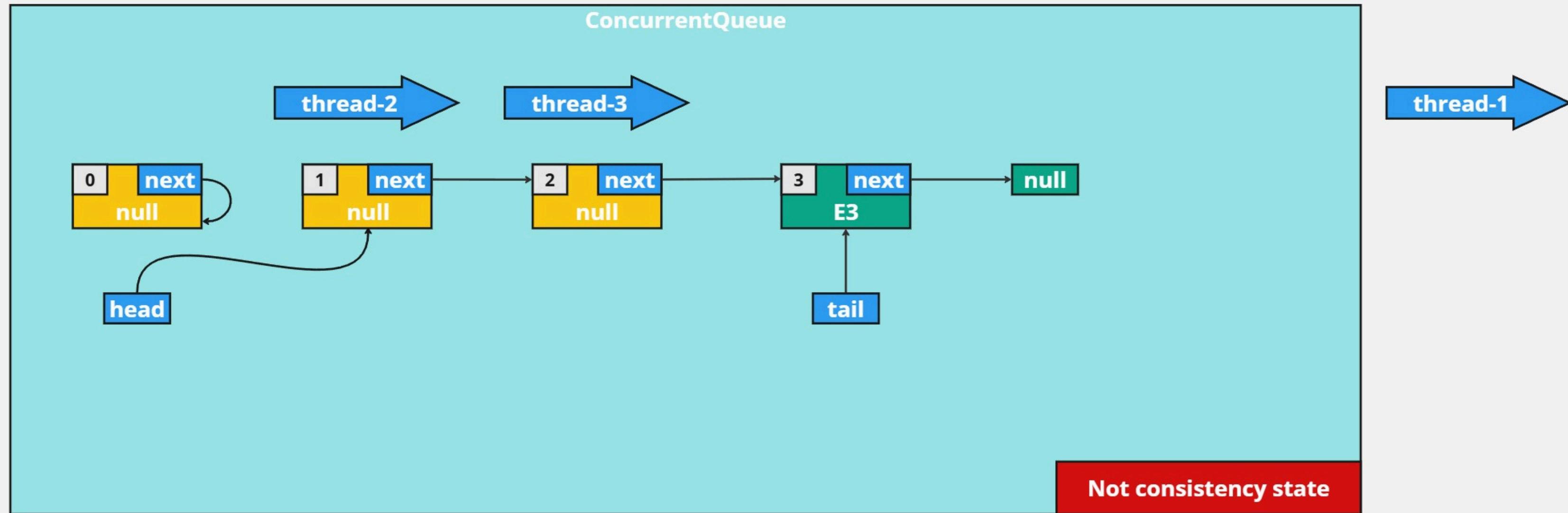
1

2

E2

### thread-3

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```



### thread-1

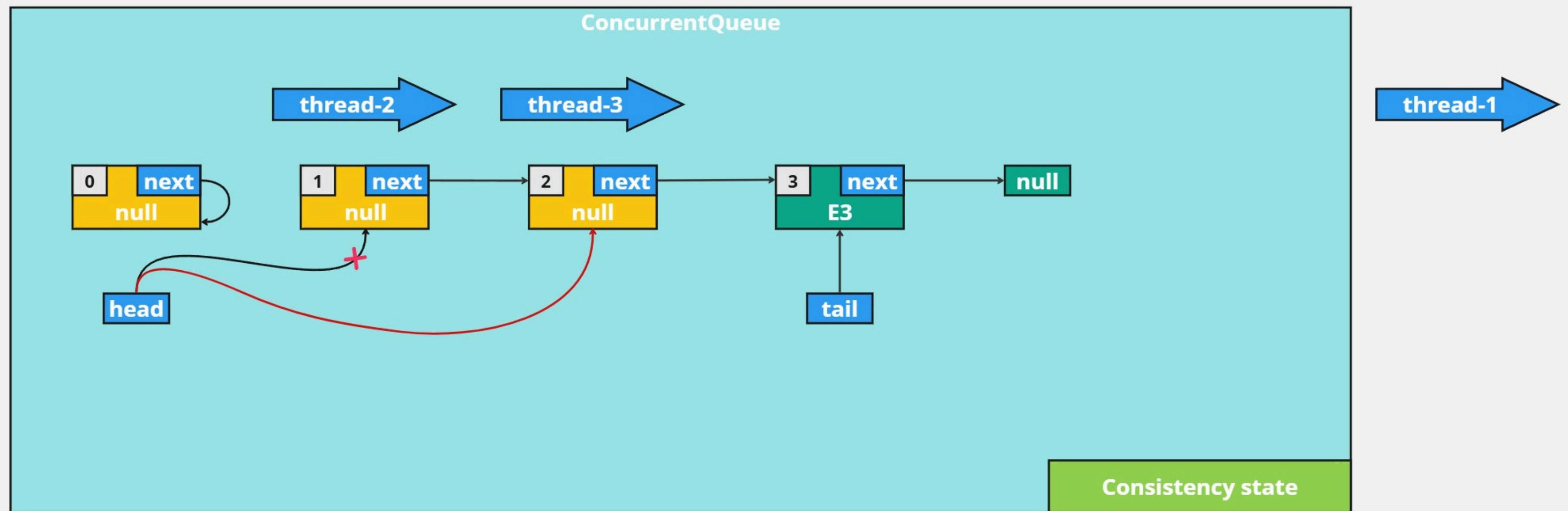
```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

### thread-2

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

### thread-3

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```



### thread-1

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

### thread-2

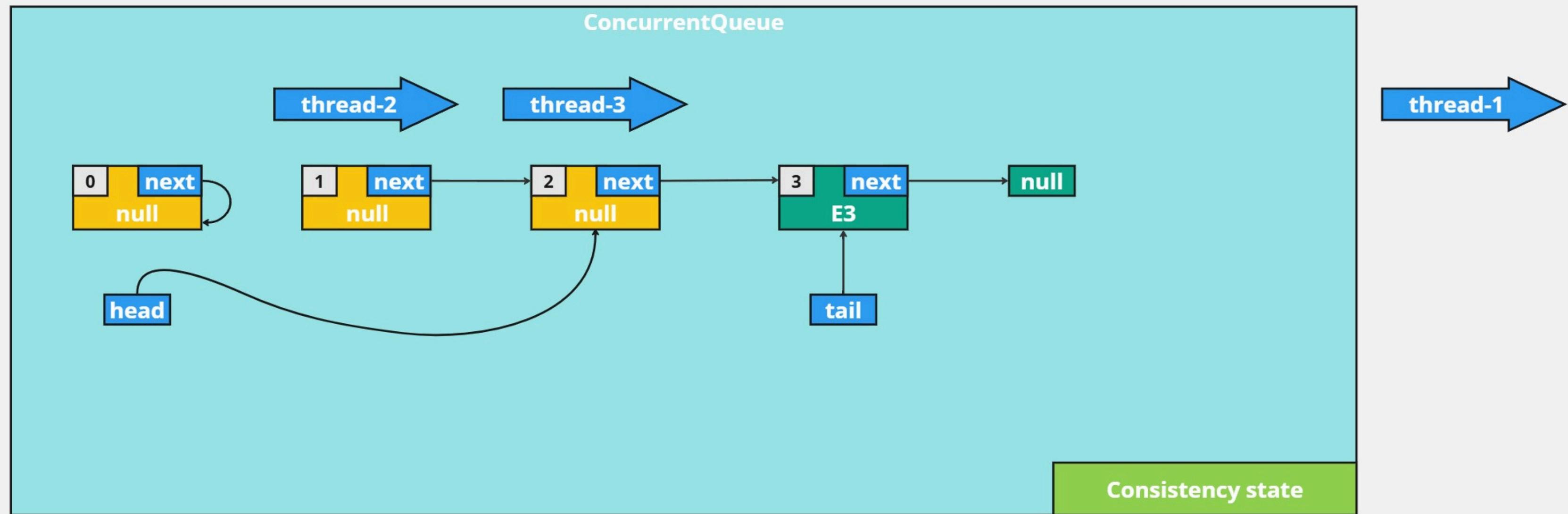
```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

Annotations for thread-2:

- Annotation 1: Points to the line `final Node<E> previousHead = head;`
- Annotation 2: Points to the line `final Node<E> nextHead = previousHead.next.get();`
- Annotation E2: Points to the line `if (element != null && nextHead.value.compareAndSet(element, null)) {`

### thread-3

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```



### thread-1

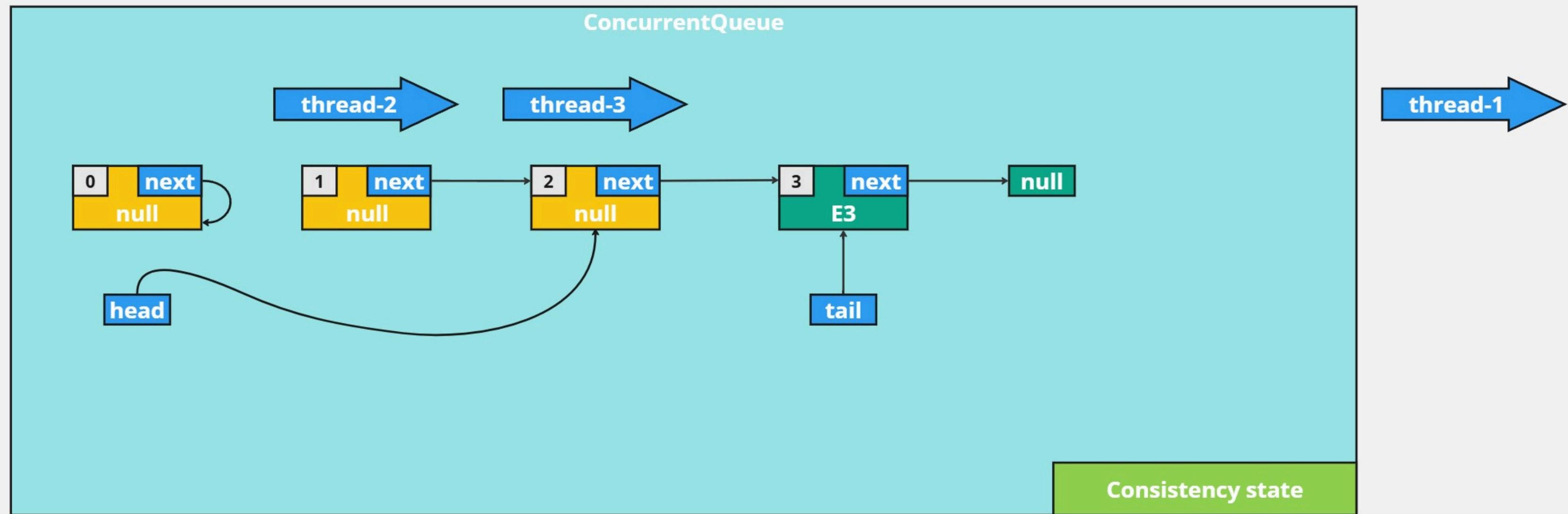
```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

### thread-2

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

### thread-3

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```



### thread-1

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

### thread-2

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

Annotations for thread-2:

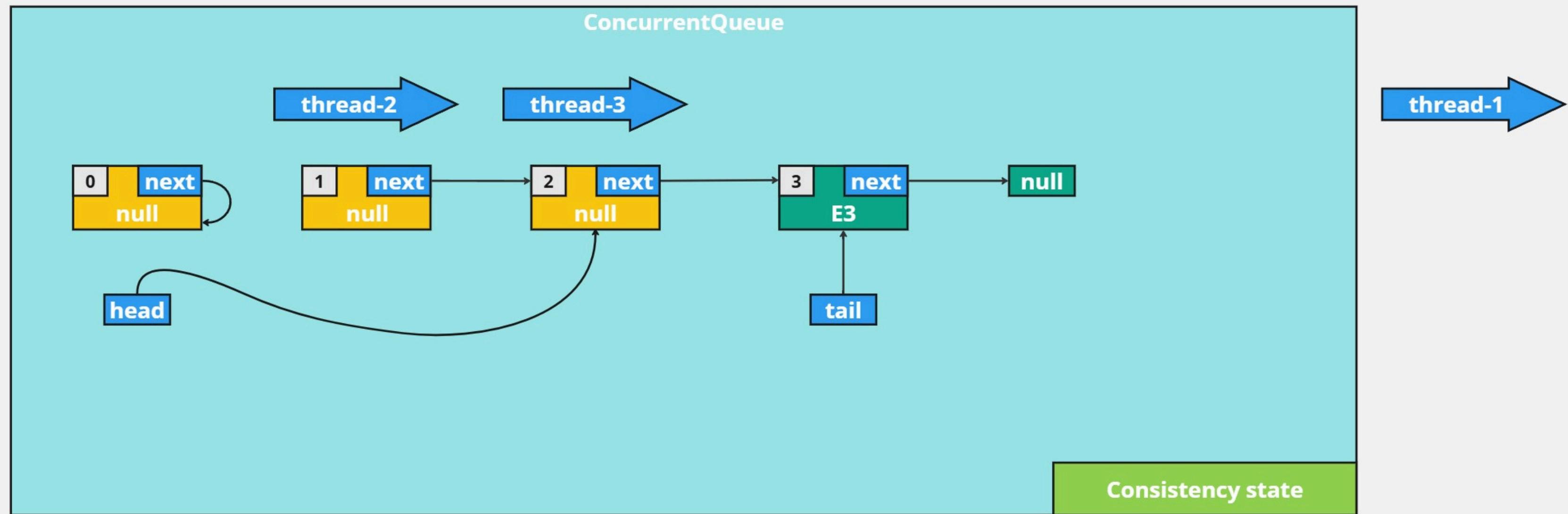
- Annotation 1: Points to the line `final Node<E> nextHead = previousHead.next.get();`
- Annotation 2: Points to the line `if (previousHead == tail.get()) {`
- Annotation E2: Points to the line `final E element = nextHead.value.get();`

### thread-3

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

Annotations for thread-3:

- Annotation 2: Points to the line `final Node<E> nextHead = previousHead.next.get();`
- Annotation 3: Points to the line `if (previousHead == tail.get()) {`



### thread-1

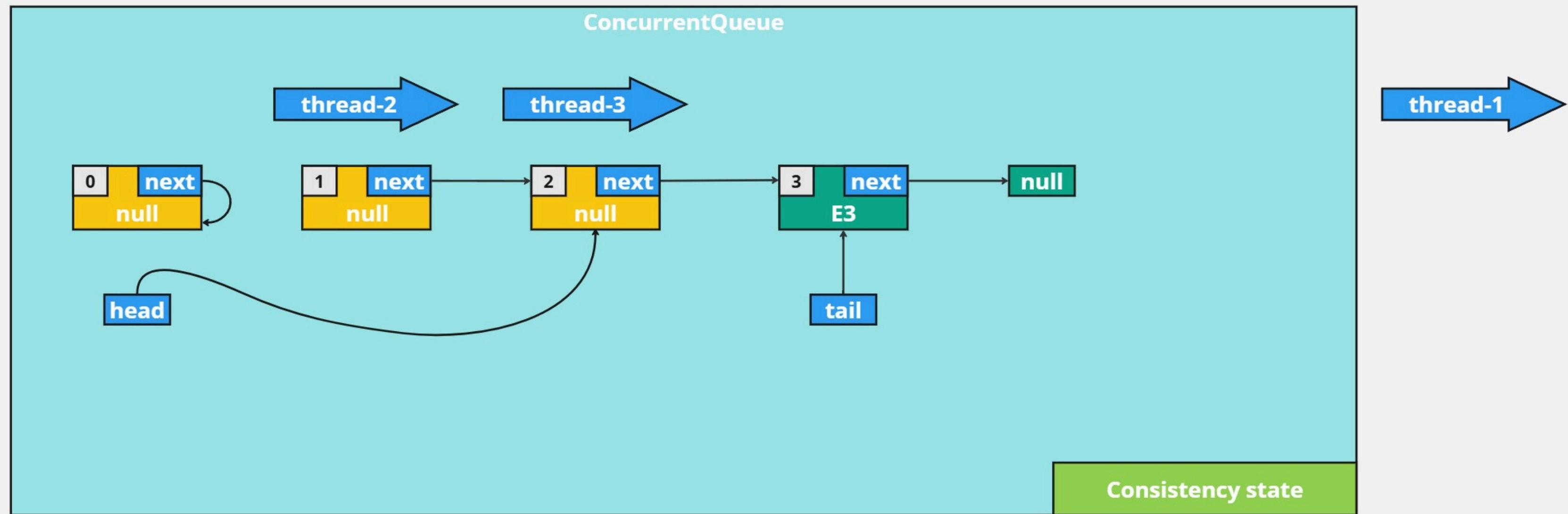
```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

### thread-2

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

### thread-3

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```



### thread-1

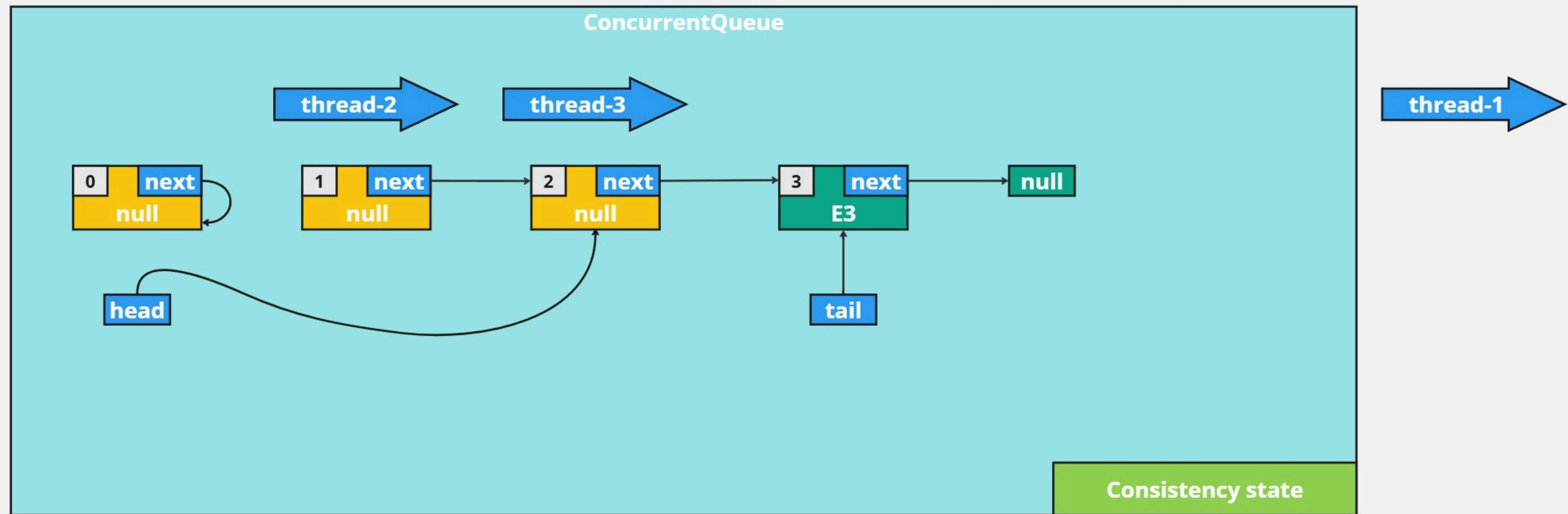
```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

### thread-2

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

### thread-3

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```



### thread-1

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

### thread-2

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

Annotations:

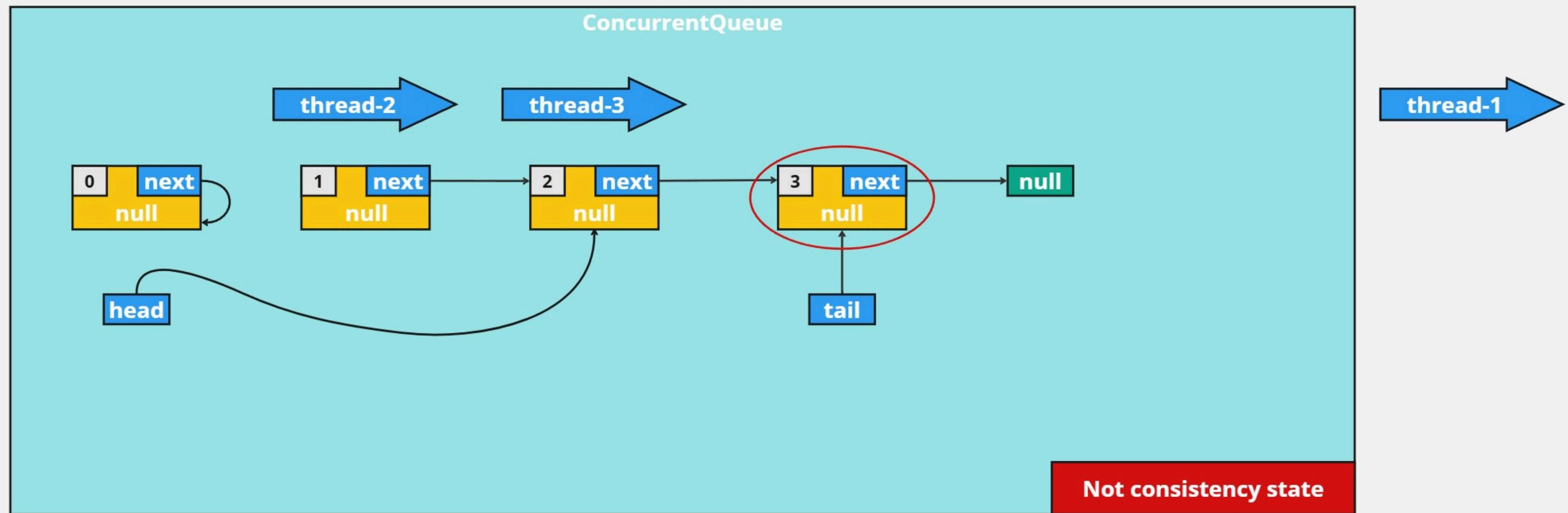
- Annotation 1: Points to the first comparison in the loop condition of thread-2's code.
- Annotation 2: Points to the first comparison in the loop condition of thread-2's code.
- Annotation E2: Points to the line "return of(element);"

### thread-3

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

Annotations:

- Annotation 2: Points to the first comparison in the loop condition of thread-3's code.
- Annotation 3: Points to the first comparison in the loop condition of thread-3's code.
- Annotation E3: Points to the line "return of(element);"



### thread-1

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

### thread-2

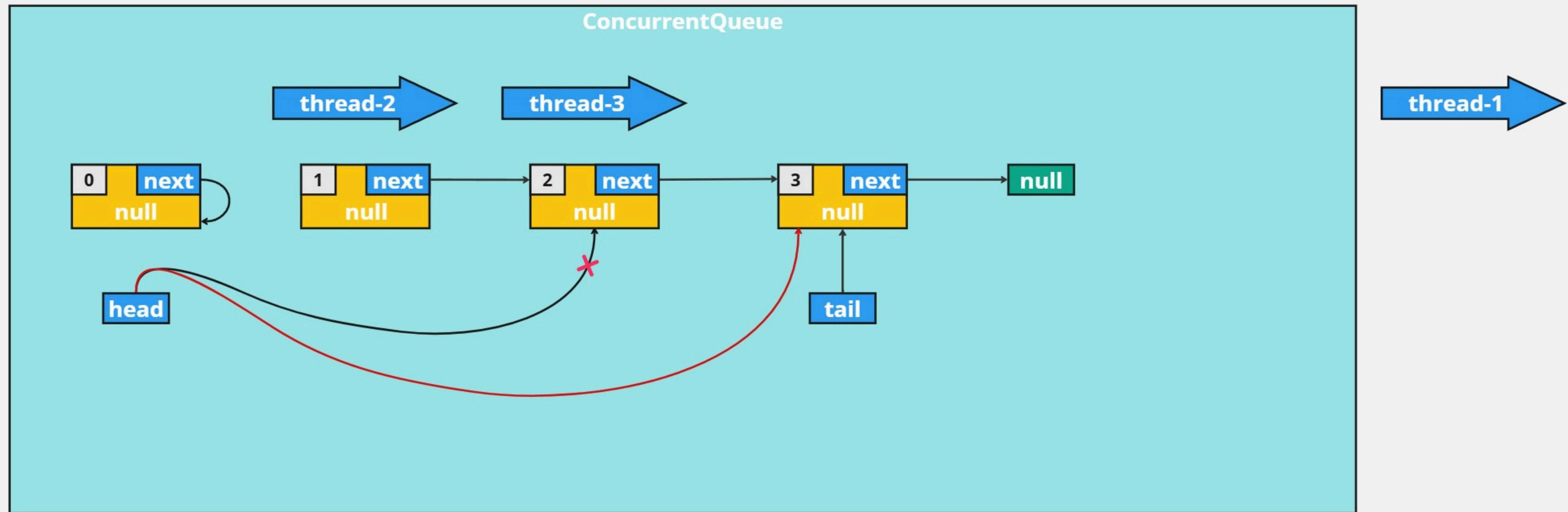
```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

1  
2  
E2

### thread-3

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

2  
3  
E3



### thread-1

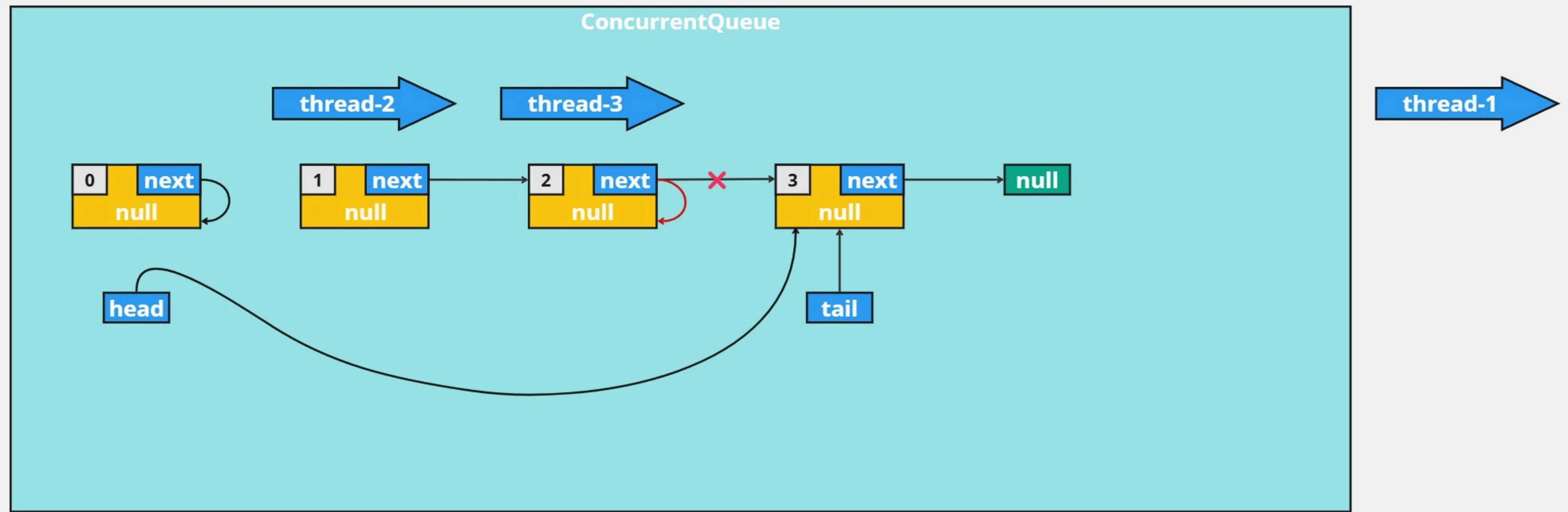
```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

### thread-2

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        1 if (previousHead == tail.get()) {
            return empty();
        }
        2 final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            3 head = nextHead;
            previousHead.next.set(previousHead);
            E2 return of(element);
        }
    }
}
```

### thread-3

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        1 if (previousHead == tail.get()) {
            return empty();
        }
        2 final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            3 head = nextHead;
            previousHead.next.set(previousHead);
            E3 return of(element);
        }
    }
}
```



### thread-1

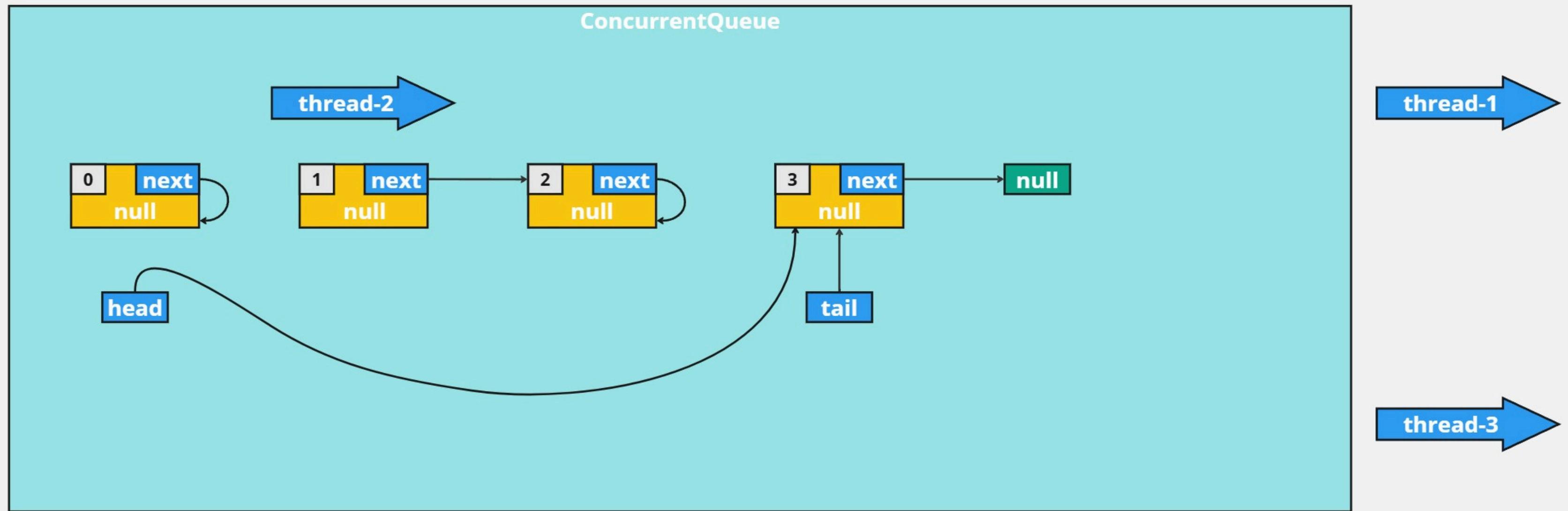
```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

### thread-2

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        1 if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        2 if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            3 previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

### thread-3

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        1 if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        2 if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            3 previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```



### thread-1

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

### thread-2

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

Annotations for thread-2:

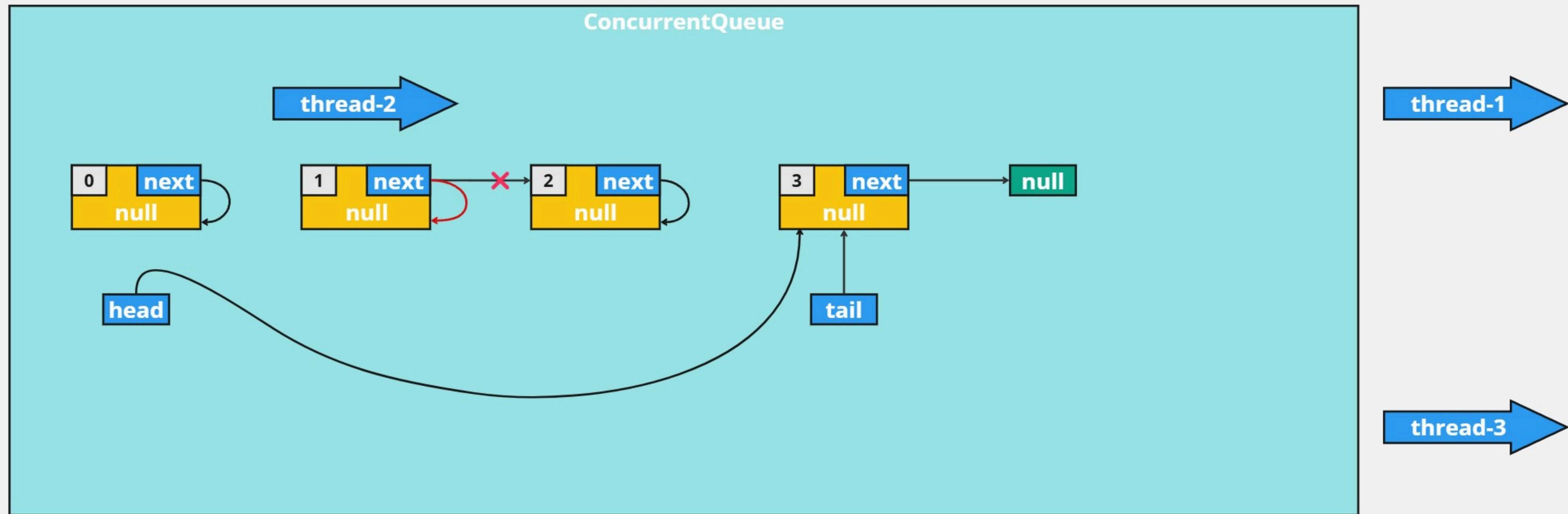
- Annotation 1: Points to the line `final Node<E> previousHead = head;`
- Annotation 2: Points to the line `final Node<E> nextHead = previousHead.next.get();`
- Annotation E2: Points to the line `final E element = nextHead.value.get();`

### thread-3

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

Annotations for thread-3:

- Annotation 2: Points to the line `final Node<E> previousHead = head;`
- Annotation 3: Points to the line `final Node<E> nextHead = previousHead.next.get();`
- Annotation E3: Points to the line `return of(element);`



### thread-1

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

### thread-2

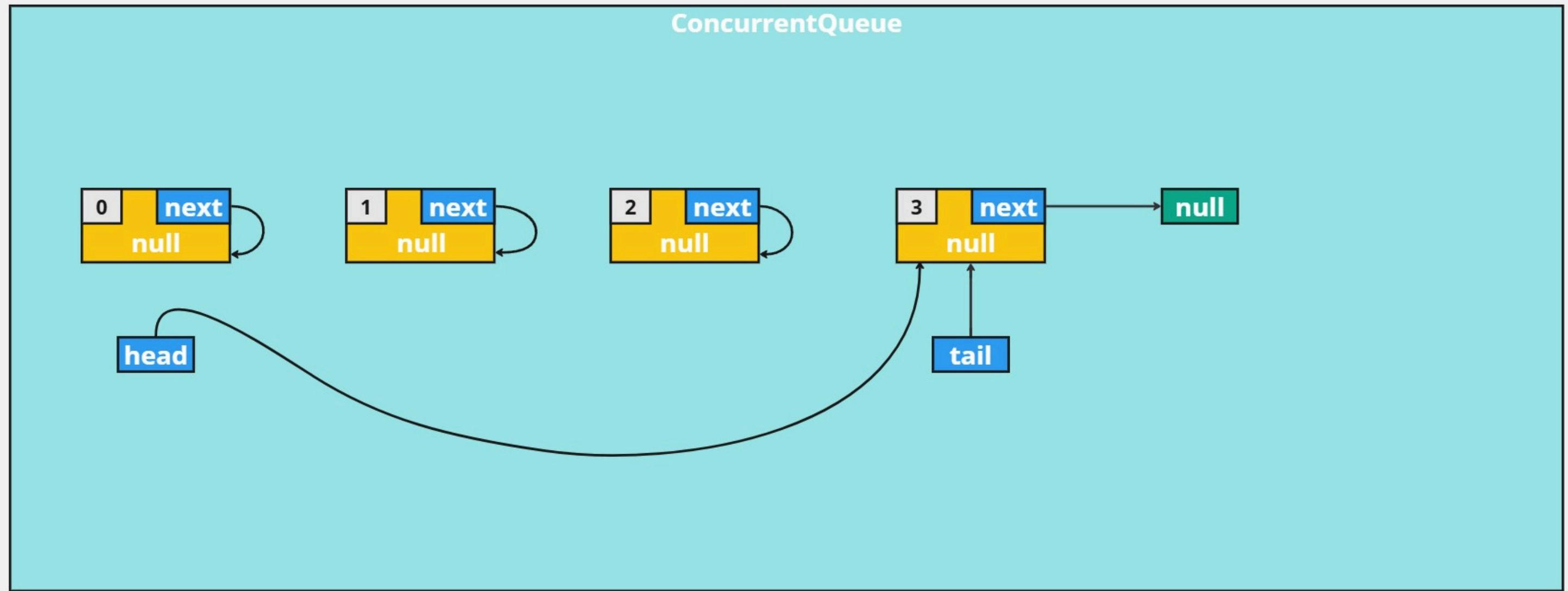
```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

Annotations for thread-2:

- Annotation 1: Points to the line `final Node<E> previousHead = head;`
- Annotation 2: Points to the line `final Node<E> nextHead = previousHead.next.get();`
- Annotation E2: Points to the line `previousHead.next.set(previousHead);`

### thread-3

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```



### thread-1

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

### thread-2

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

Annotations for thread-2:

- Annotation 1: Points to the line `final Node<E> previousHead = head;`
- Annotation 2: Points to the line `final Node<E> nextHead = previousHead.next.get();`
- Annotation E2: Points to the line `return of(element);`

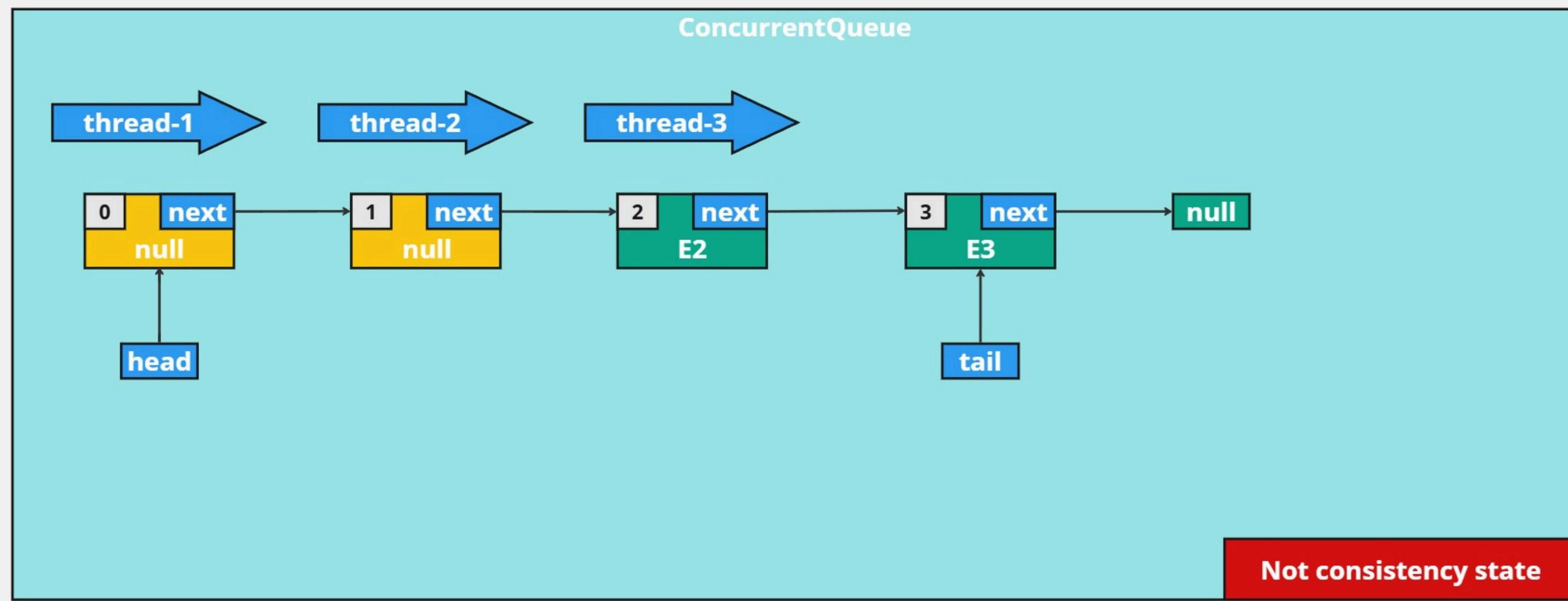
### thread-3

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

thread-1

thread-2

thread-3



### thread-1

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

Annotations:

- Line 1: A red box labeled '1' surrounds the entire loop body.
- Line 2: A red box labeled '0' surrounds the assignment of previousHead.
- Line 5: A red box labeled 'E1' surrounds the assignment of element.

### thread-2

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

Annotations:

- A red bracket on the right side groups the entire loop body.
- A red bracket on the left side groups the assignment of previousHead.

### thread-3

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head;
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            head = nextHead;
            previousHead.next.set(previousHead);
            return of(element);
        }
    }
}
```

Annotations:

- A red bracket on the right side groups the entire loop body.
- A red bracket on the left side groups the assignment of previousHead.



```
private volatile Node<E> head;
```

```
public ConcurrentQueue() {  
    final Node<E> dummyNode = new Node<>();  
head = dummyNode;  
    tail = new AtomicReference<>(dummyNode);  
}
```

```
public Optional<E> dequeue() {  
    while (true) {  
        final Node<E> previousHead = head;  
        final Node<E> nextHead = previousHead.next.get();  
        if (previousHead == tail.get()) {  
            return empty();  
        }  
        final E element = nextHead.value.get();  
        if (element != null && nextHead.value.compareAndSet(element, null)) {  
            head = nextHead;  
previousHead.next.set(previousHead);  
            return of(element);  
        }  
    }  
}
```

```
private final AtomicReference<Node<E>> head;
```

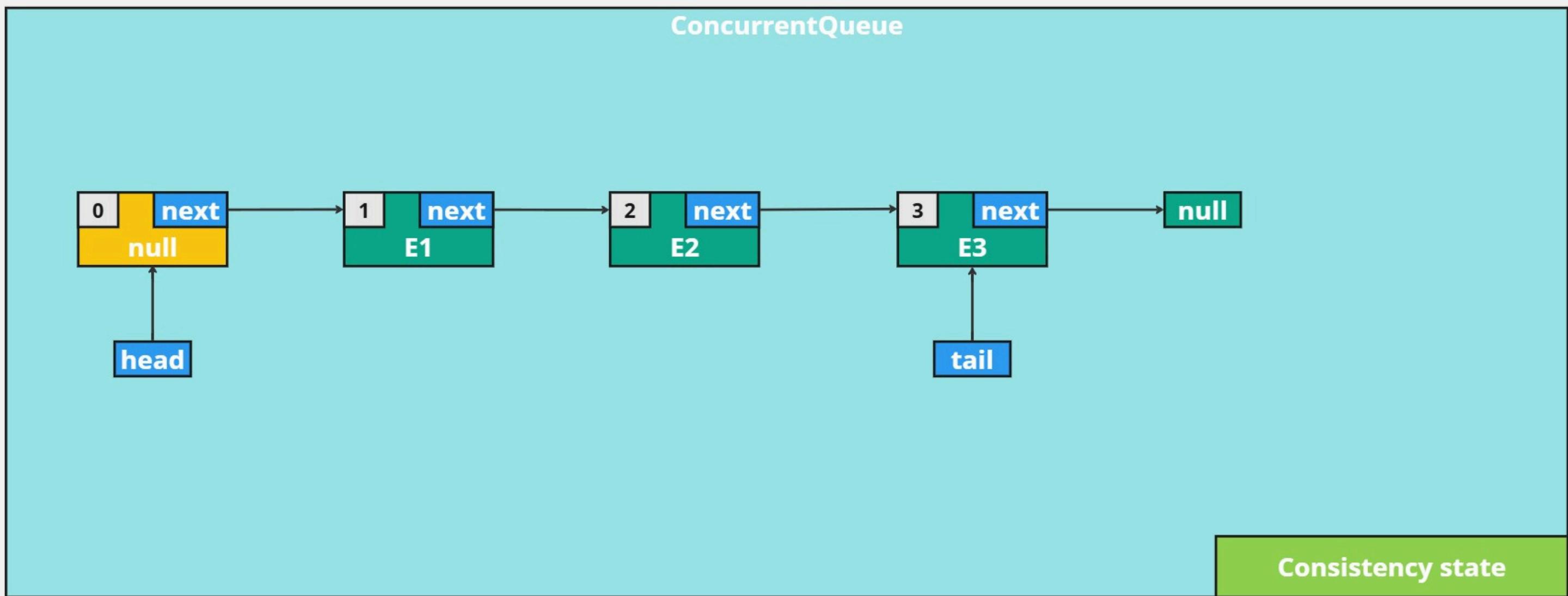
```
public ConcurrentQueue() {  
    final Node<E> dummyNode = new Node<>();  
head = new AtomicReference<>(dummyNode);  
    tail = new AtomicReference<>(dummyNode);  
}
```

```
public Optional<E> dequeue() {  
    while (true) {  
        final Node<E> previousHead = head.get();  
        final Node<E> nextHead = previousHead.next.get();  
        if (previousHead == tail.get()) {  
            return empty();  
        }  
        final E element = nextHead.value.get();  
        if (element != null && nextHead.value.compareAndSet(element, null)) {  
            updateHead(previousHead, nextHead);  
            return of(element);  
        } else {  
            updateHead(previousHead, nextHead);  
        }  
    }  
}  
private void updateHead(final Node<E> previous, final Node<E> next) {  
    if (head.compareAndSet(previous, next)) {  
        previous.next.set(next);  
    }  
}
```



## ConcurrentQueue

thread-1 →  
thread-2 →  
thread-3 →



### thread-1

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}
```

### thread-2

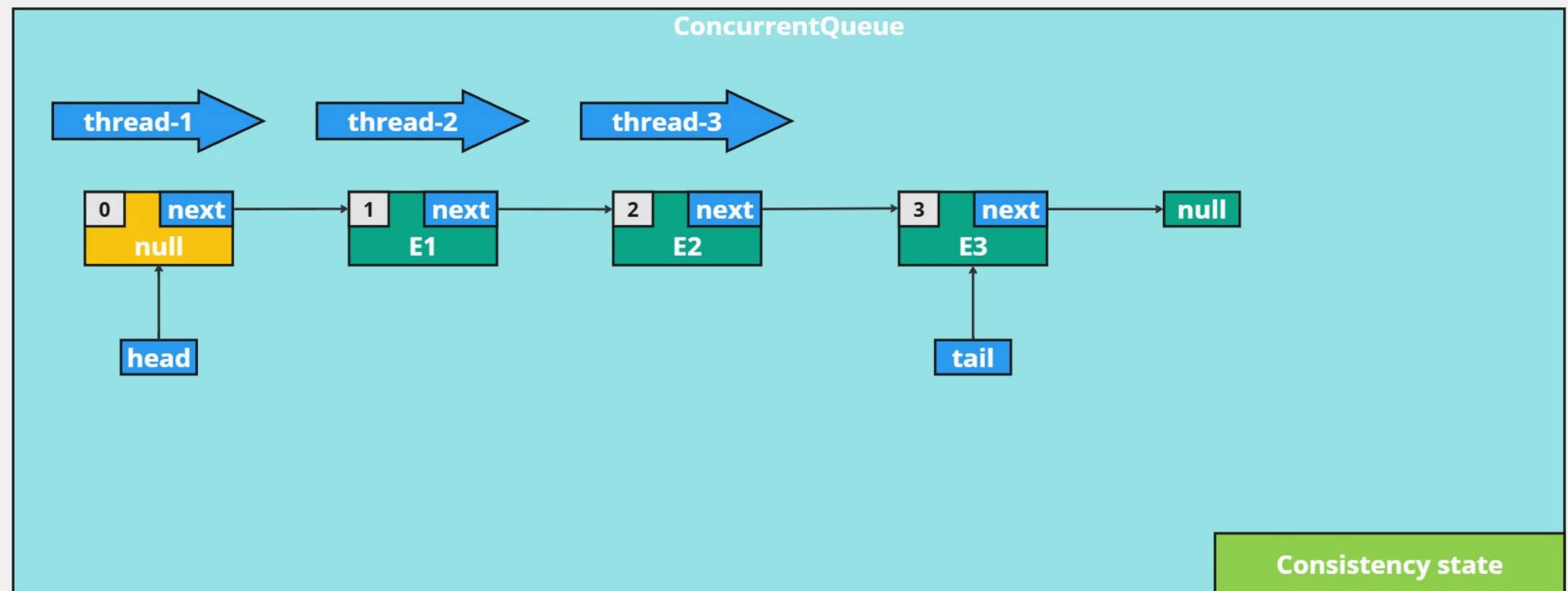
```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}
```

### thread-3

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}
```



### thread-1

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}
```

### thread-2

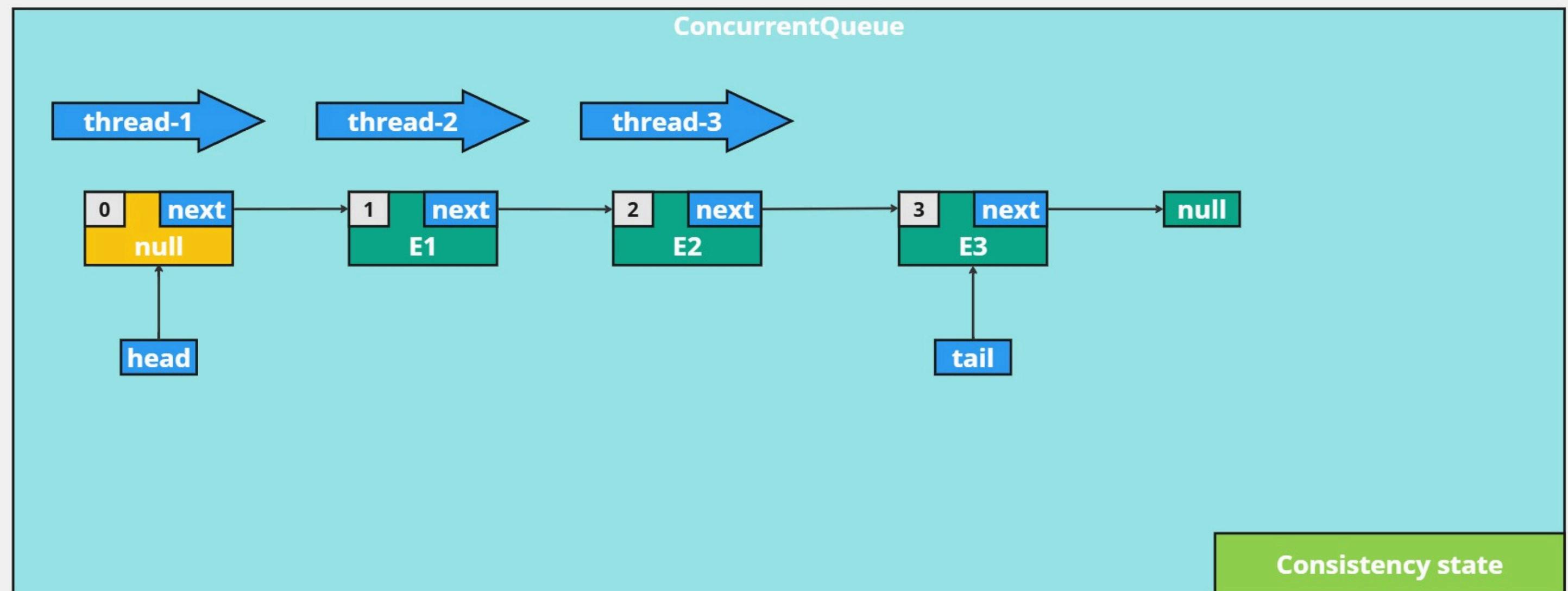
```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}
```

### thread-3

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}
```



### thread-1

```

public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get(); 0
        final Node<E> nextHead = previousHead.next.get(); 1
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}

```

### thread-2

```

public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get(); 0
        final Node<E> nextHead = previousHead.next.get(); 1
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}

```

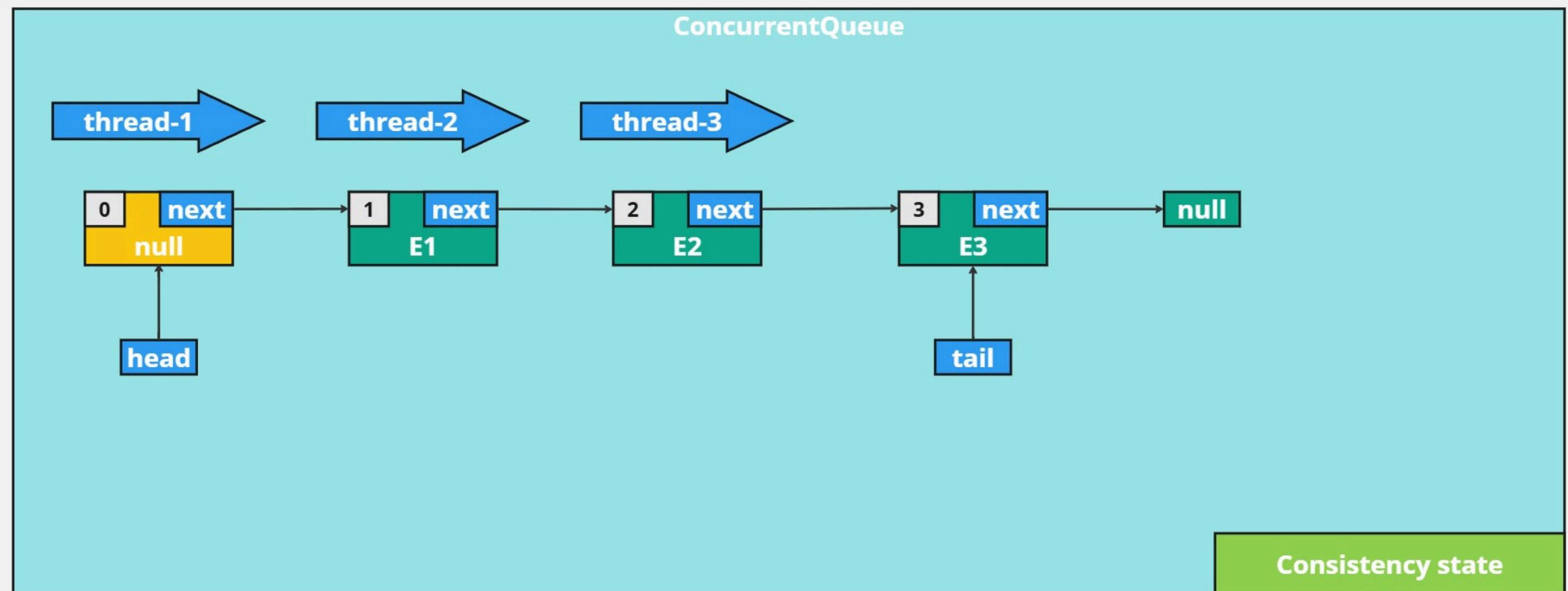
### thread-3

```

public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get(); 0
        final Node<E> nextHead = previousHead.next.get(); 1
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}

```



### thread-1

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}
```

### thread-2

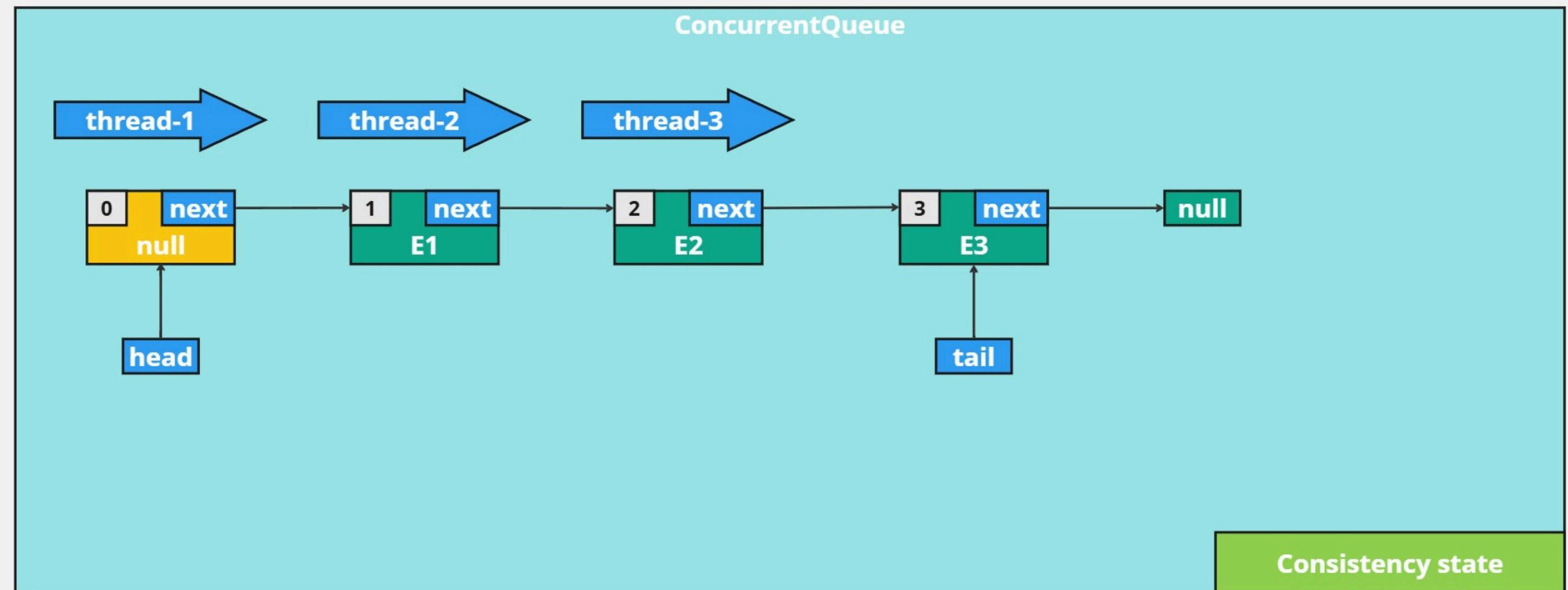
```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}
```

### thread-3

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}
```



#### thread-1

```

public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get(); E1
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}

```

#### thread-2

```

public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get(); E1
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}

```

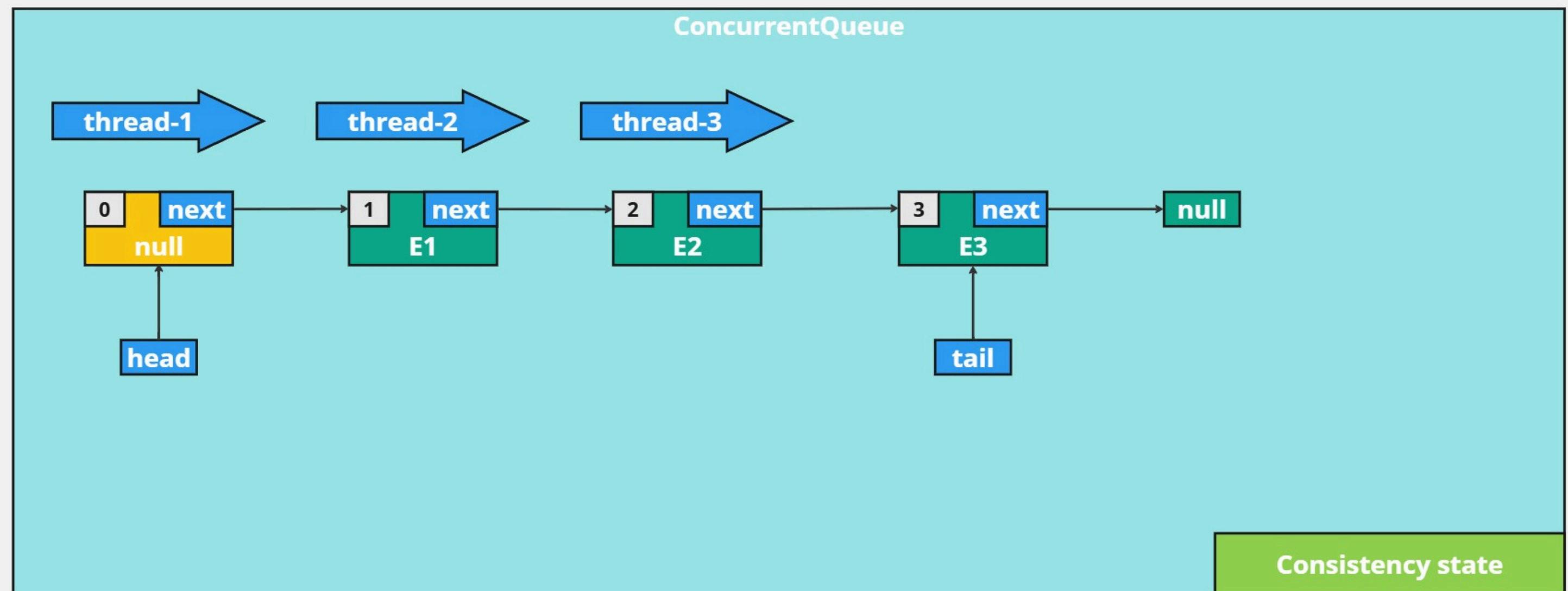
#### thread-3

```

public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get(); E1
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}

```



#### thread-1

```

public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get(); E1
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}

```

#### thread-2

```

public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get(); E1
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}

```

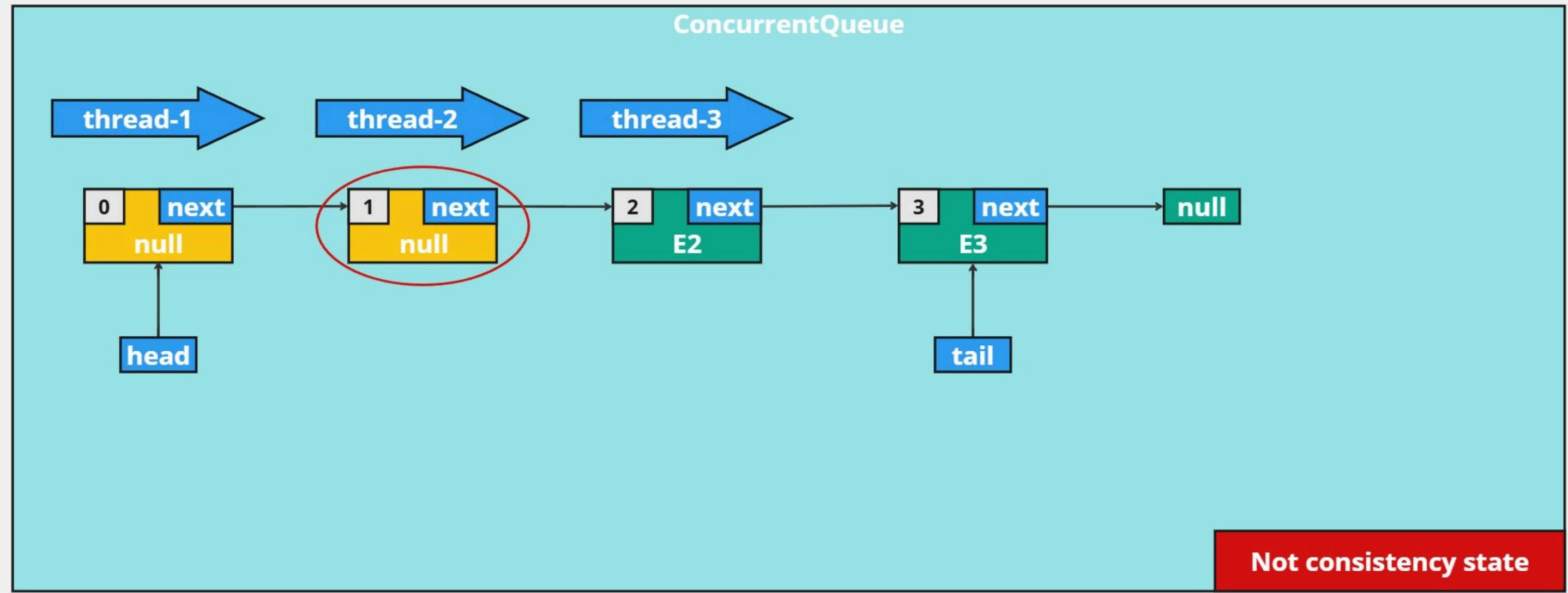
#### thread-3

```

public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get(); E1
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}

```



#### thread-1

```

public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get(); E1
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}

```

#### thread-2

```

public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get(); E1
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}

```

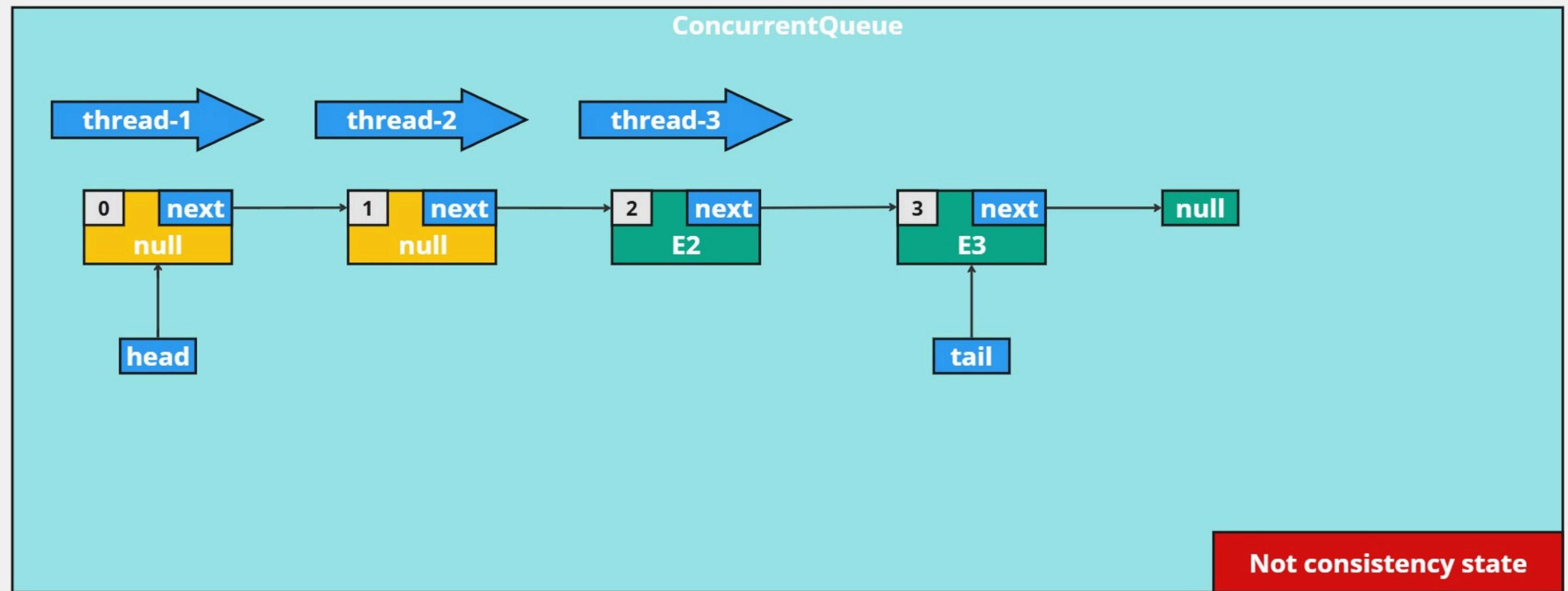
#### thread-3

```

public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get(); E1
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}

```



### thread-1

```

public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get(); E1
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}

```

### thread-2

```

public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get(); E1
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}

```

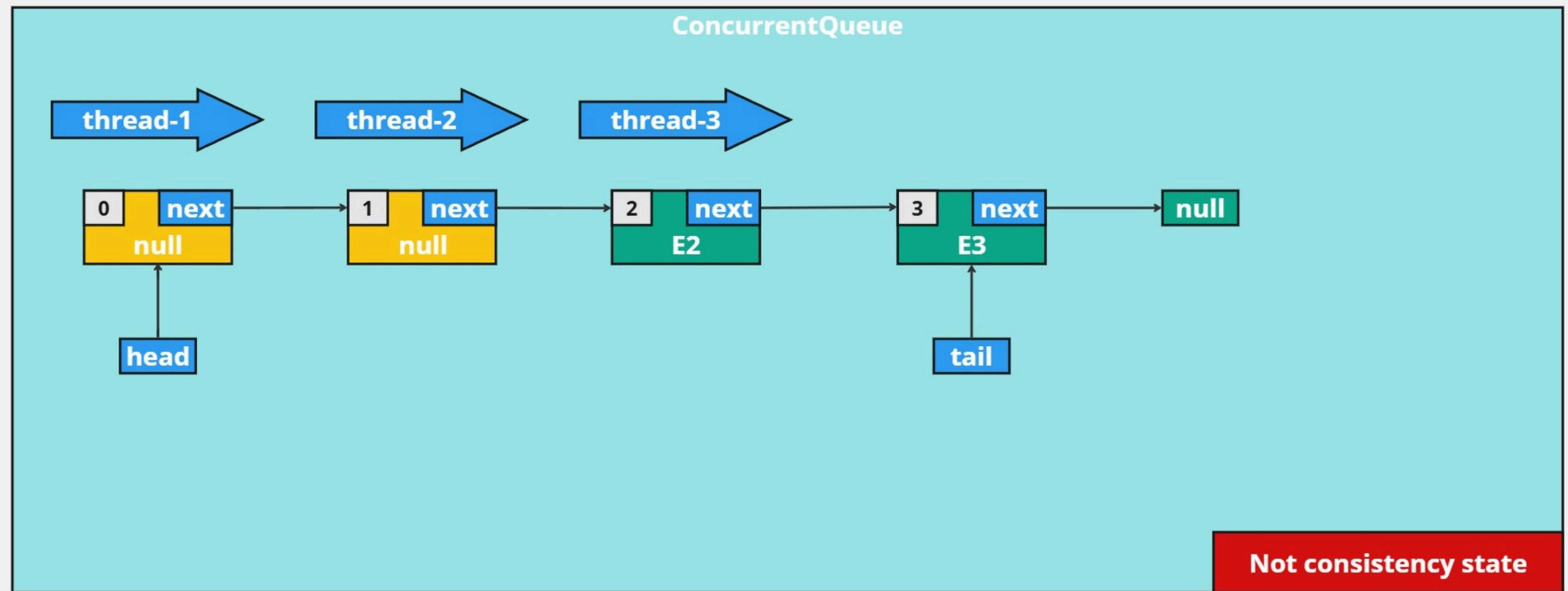
### thread-3

```

public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get(); E1
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}

```



### thread-1

```

public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get(); E1
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}

```

### thread-2

```

public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get(); E1
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead); E1
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}

```

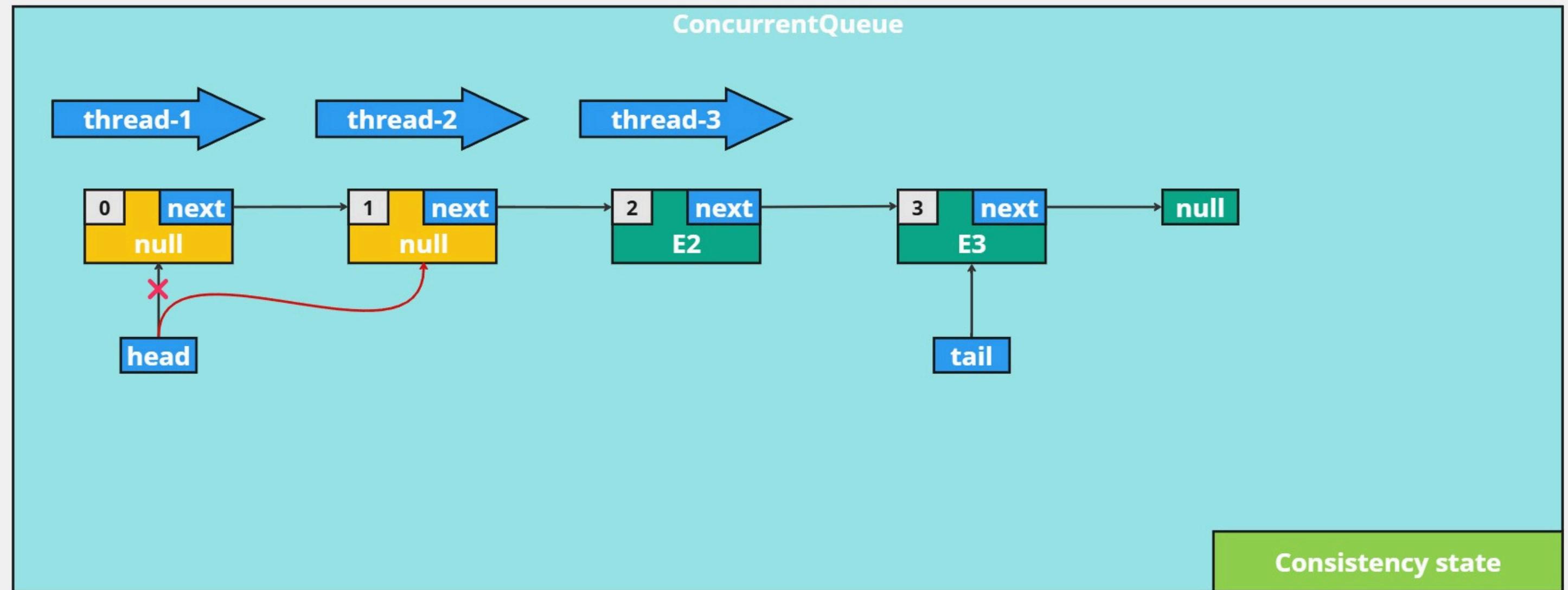
### thread-3

```

public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get(); E1
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead); E1
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}

```



#### thread-1

```

public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get(); E1
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}

```

#### thread-2

```

public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get(); E1
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead); E1
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}

```

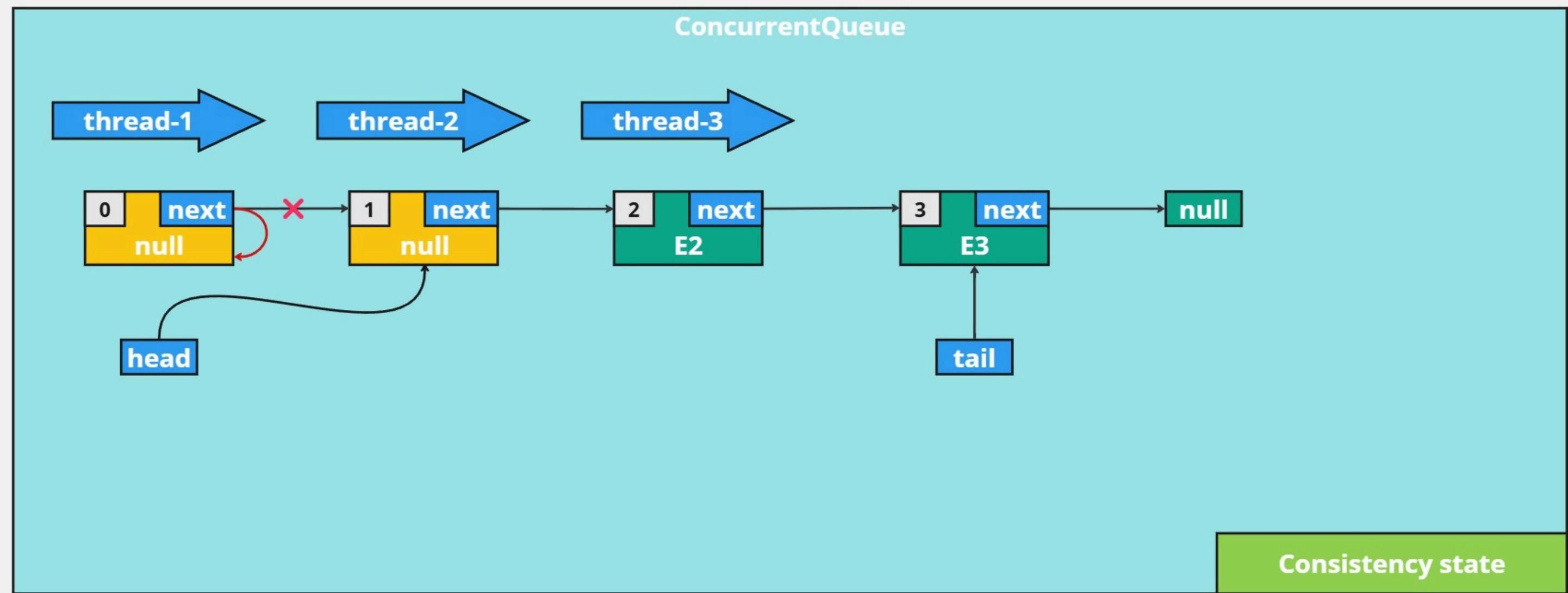
#### thread-3

```

public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get(); E1
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead); E1
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}

```



#### thread-1

```

public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get(); E1
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}

```

#### thread-2

```

public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get(); E1
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead); E1
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}

```

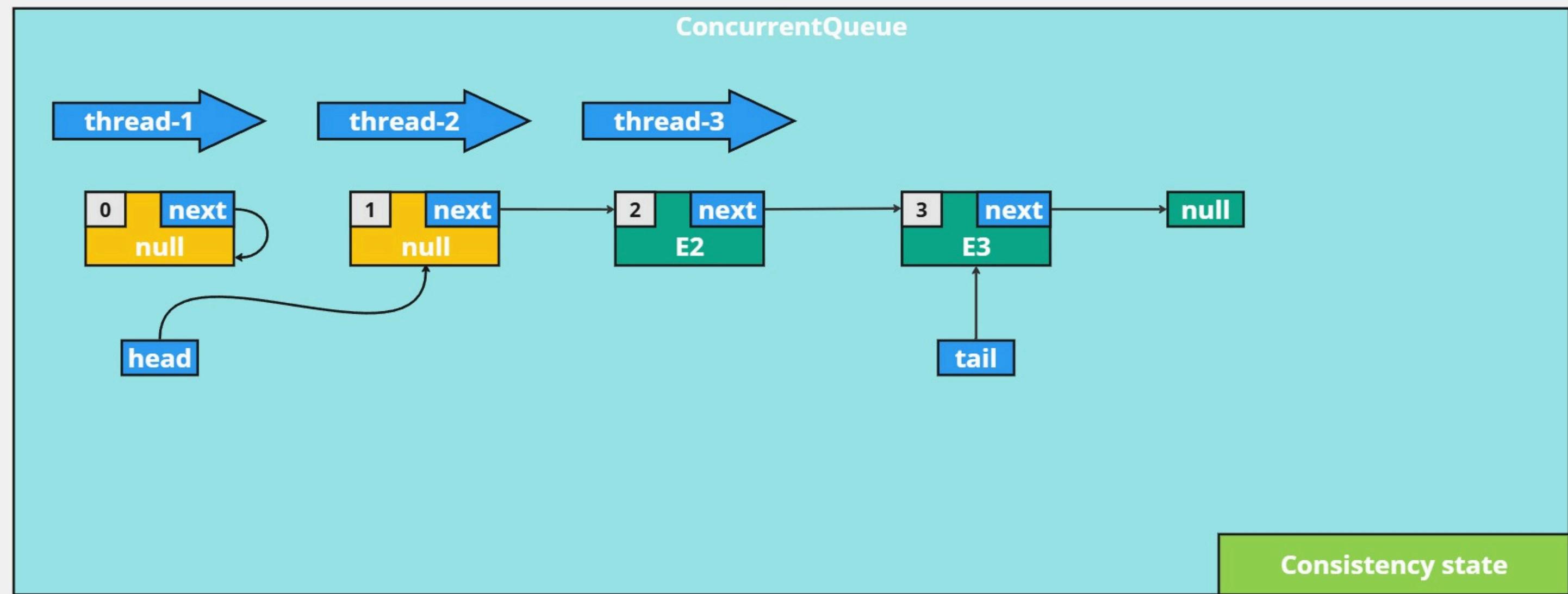
#### thread-3

```

public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get(); E1
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead); E1
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}

```



### thread-1

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get(); E1
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}
```

### thread-2

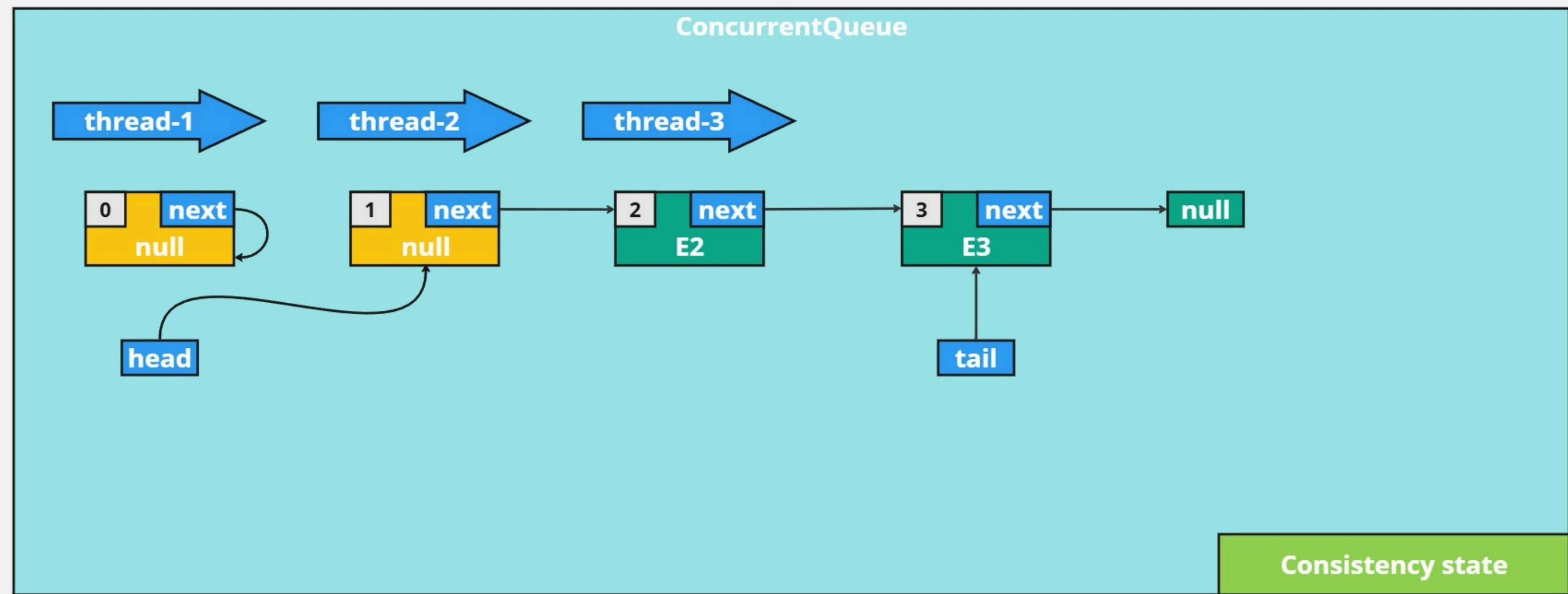
```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get(); E2
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}
```

### thread-3

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get(); E3
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}
```



### thread-1

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get(); E1
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}
```

### thread-2

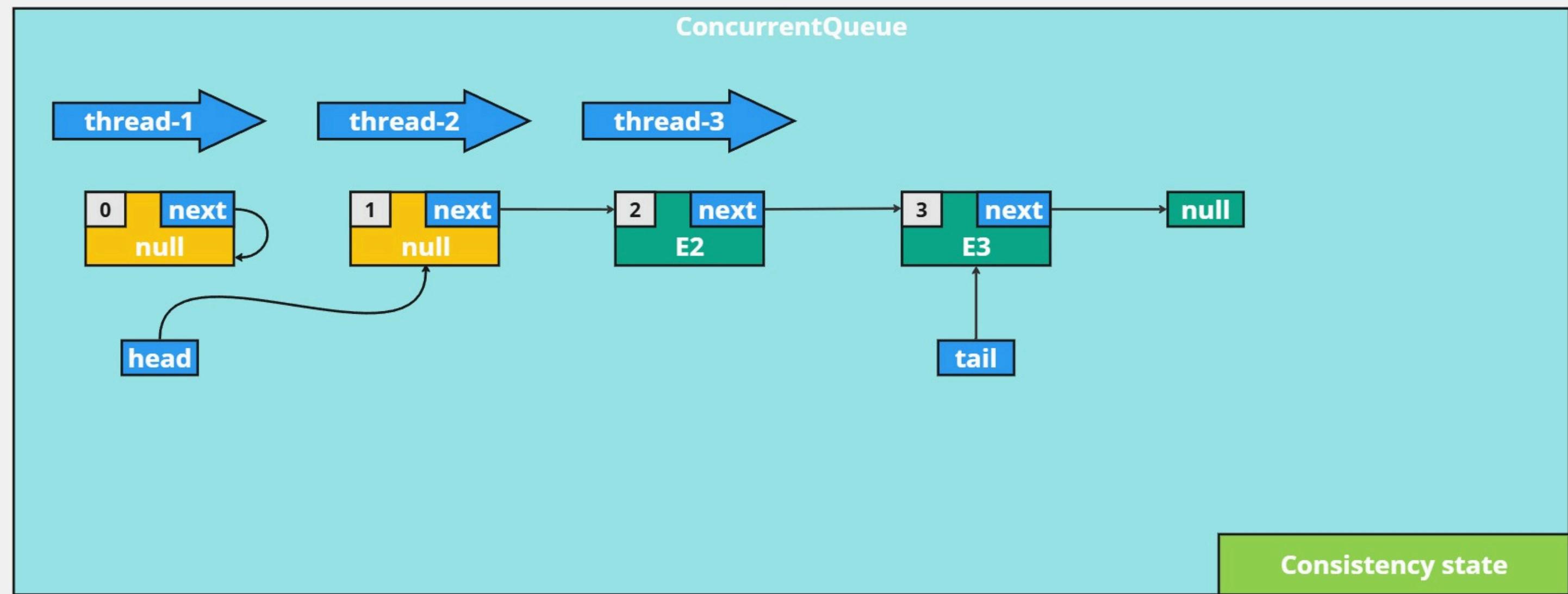
```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get(); E2
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}
```

### thread-3

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get(); E3
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
if (head.compareAndSet(previous, next)) {
    previous.next.set(next);
}
}
```



### thread-1

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get(); E1
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}
```

### thread-2

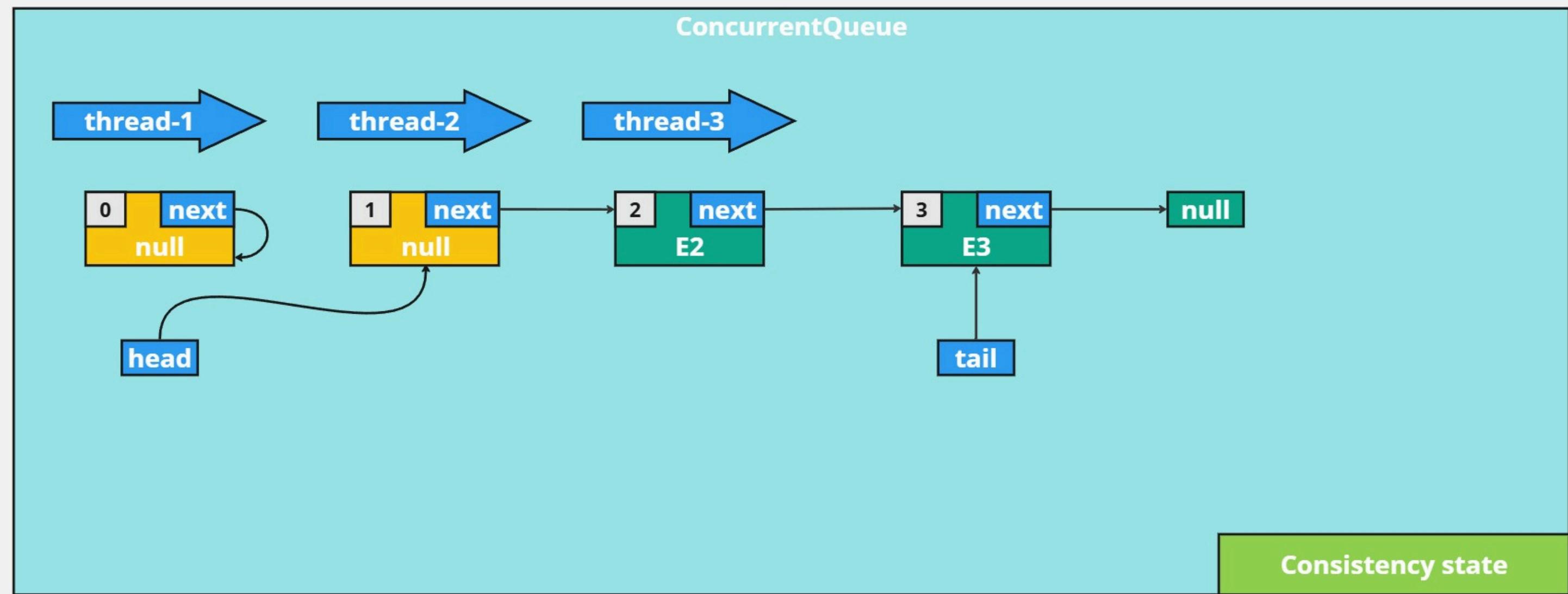
```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get(); E2
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}
```

### thread-3

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get(); E3
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}
```



### thread-1

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get(); E1
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}
```

### thread-2

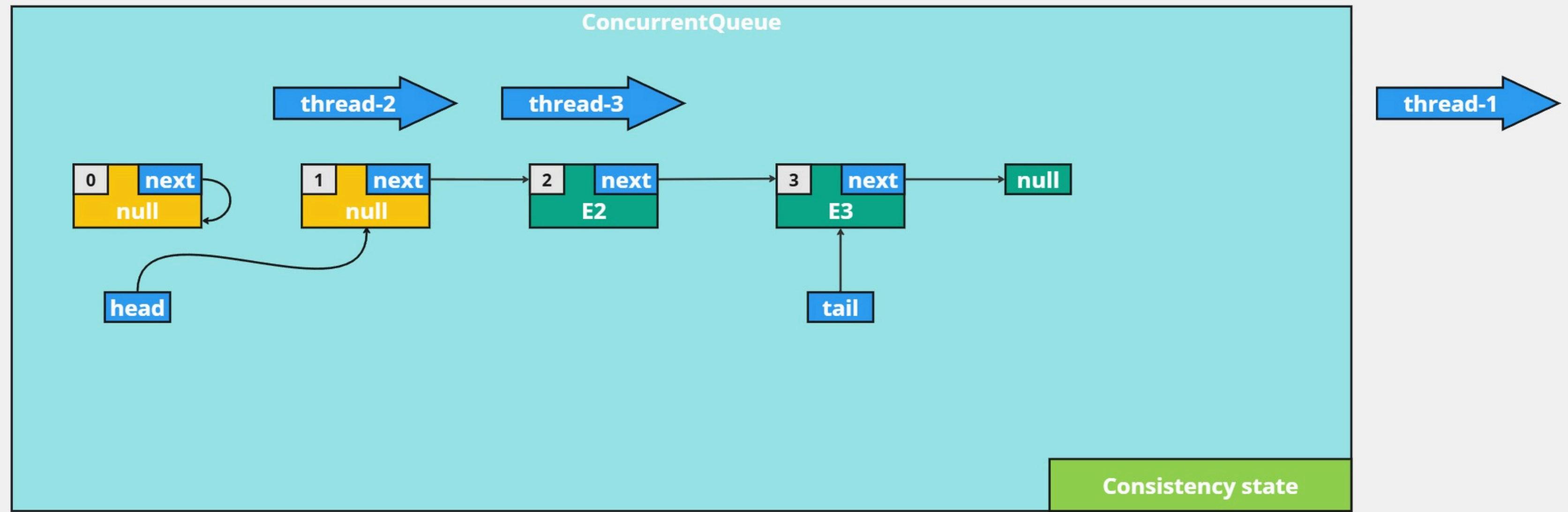
```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get(); E2
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}
```

### thread-3

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get(); E3
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}
```



### thread-1

```

public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get(); E1
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}

```

### thread-2

```

public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}

```

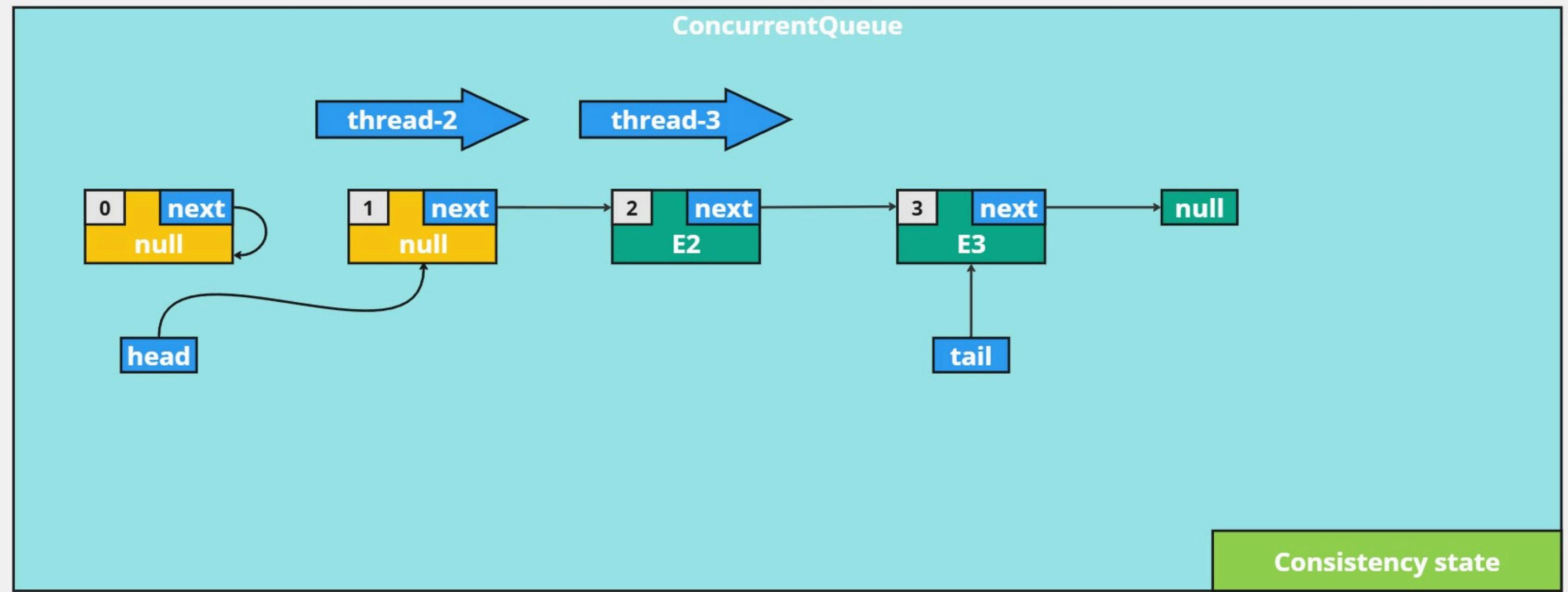
### thread-3

```

public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}

```



### thread-1

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}
```

### thread-2

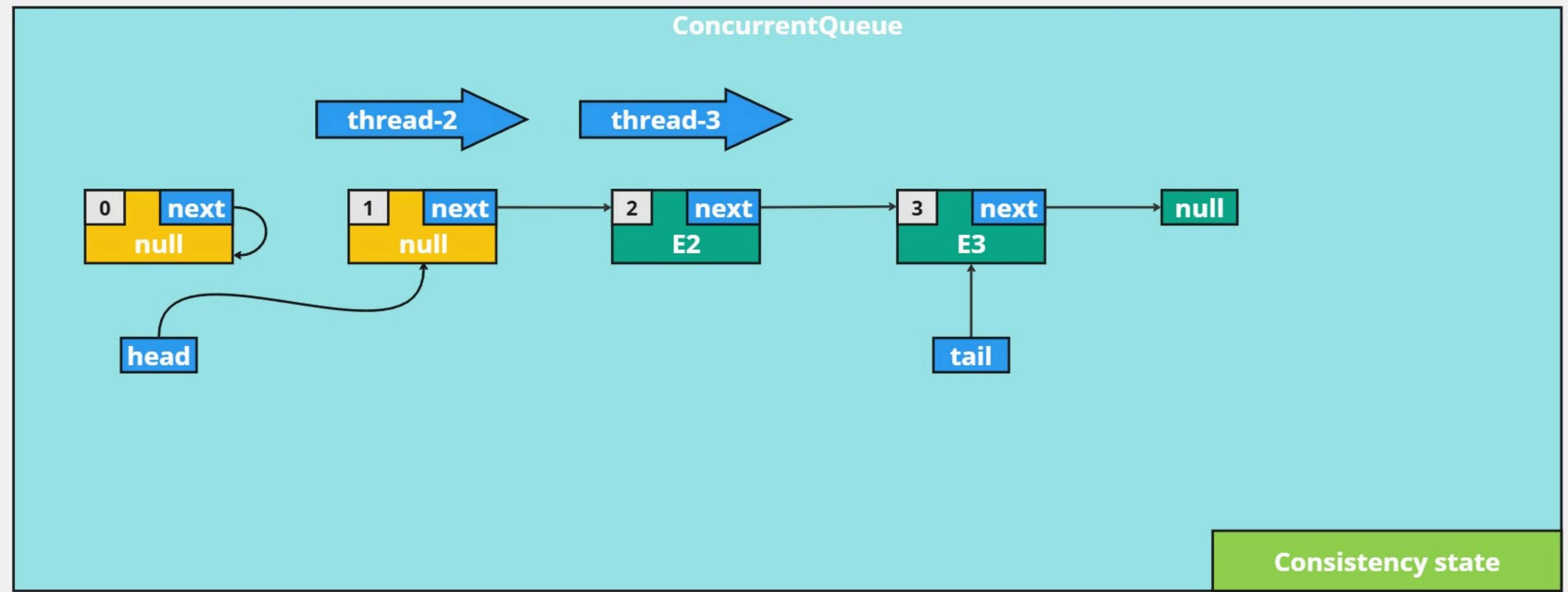
```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}
```

### thread-3

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}
```



### thread-1

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}
```

### thread-2

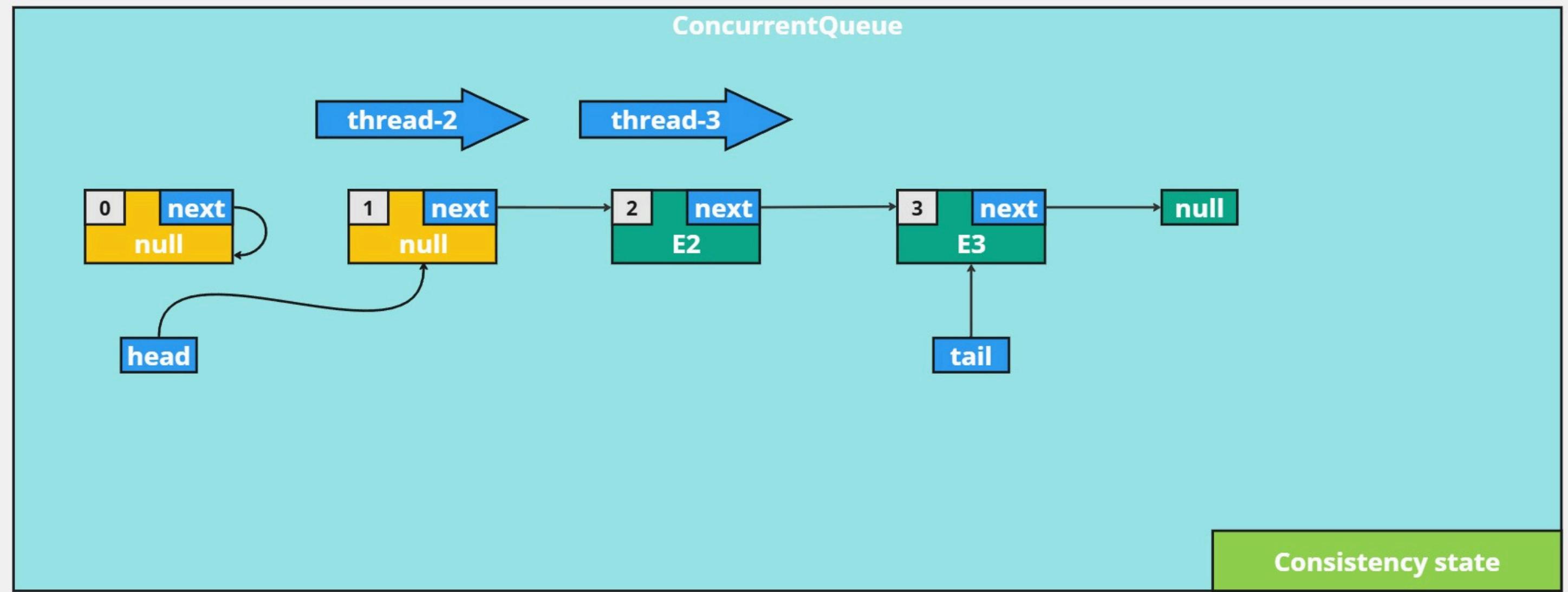
```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get(); 1
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}
```

### thread-3

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get(); 1
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}
```



### thread-1

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}
```

### thread-2

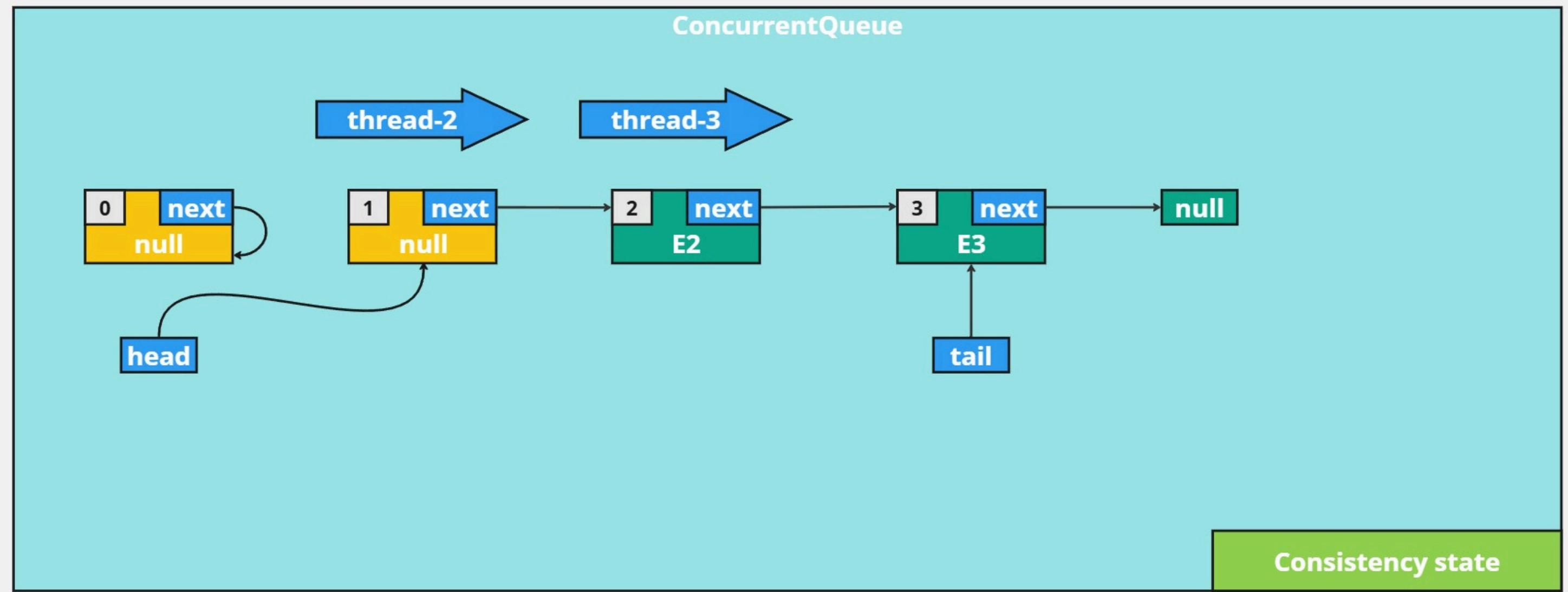
```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}
```

### thread-3

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}
```



### thread-1

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}
```

### thread-2

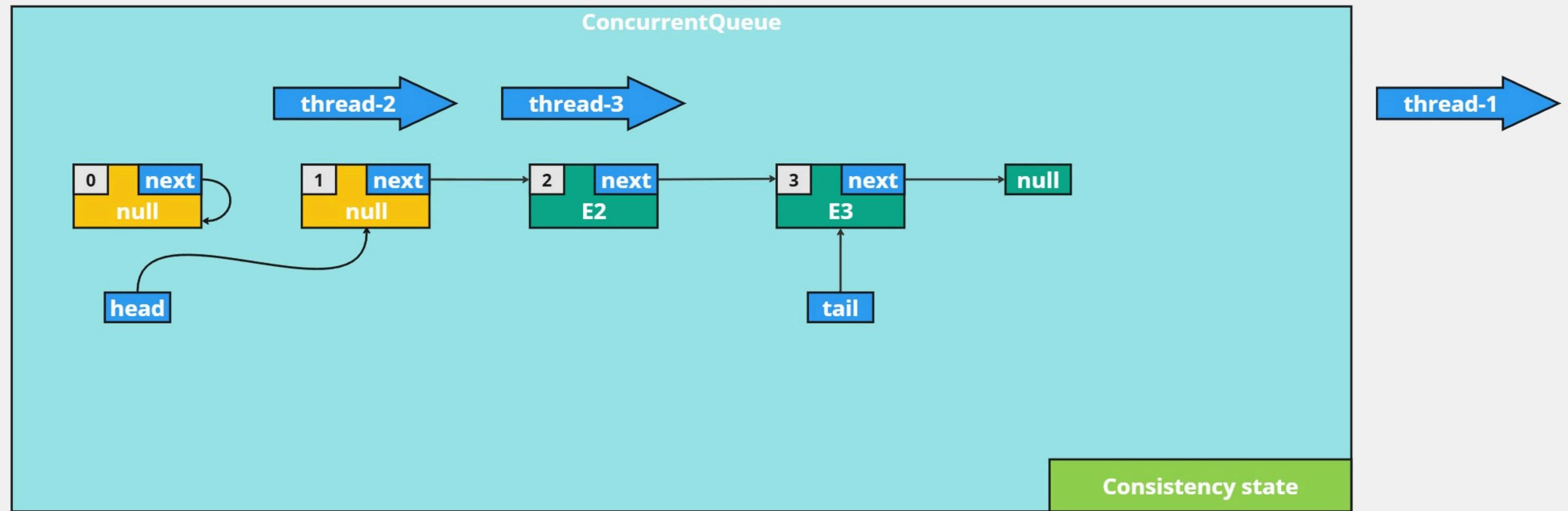
```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) { 1
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}
```

### thread-3

```
public Optional<E> dequeue() {
    while (true) { 1
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) { 2
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}
```



### thread-1

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}
```

### thread-2

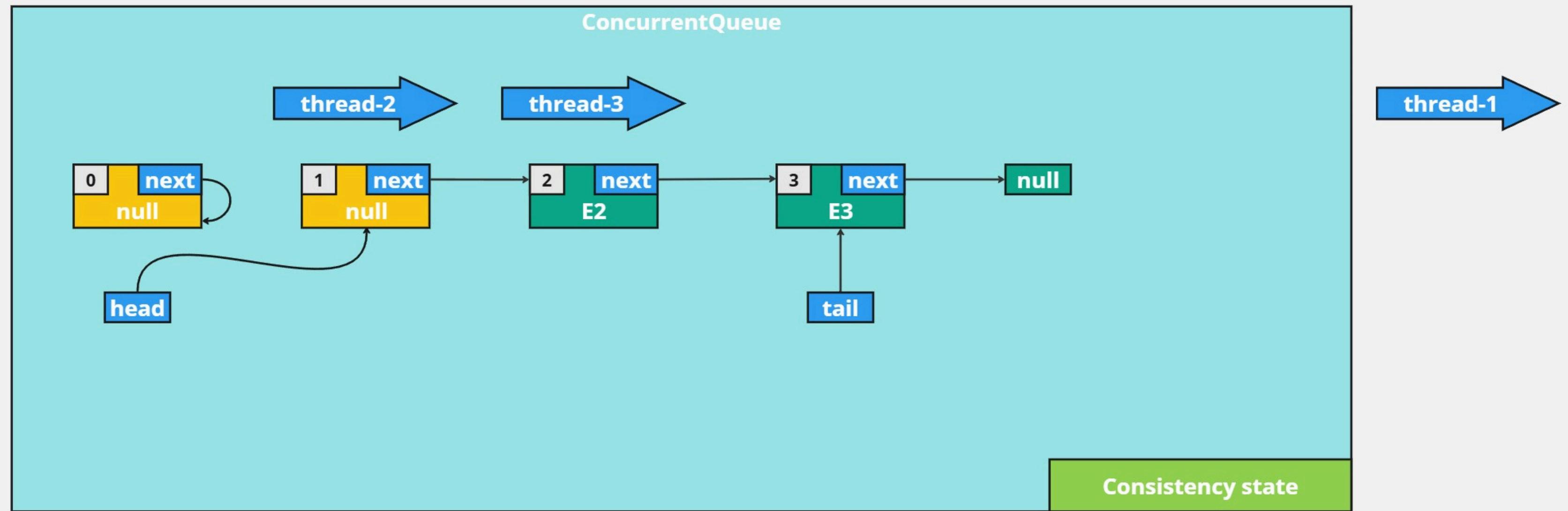
```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get(); E2
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}
```

### thread-3

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get(); E2
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}
```



### thread-1

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}
```

### thread-2

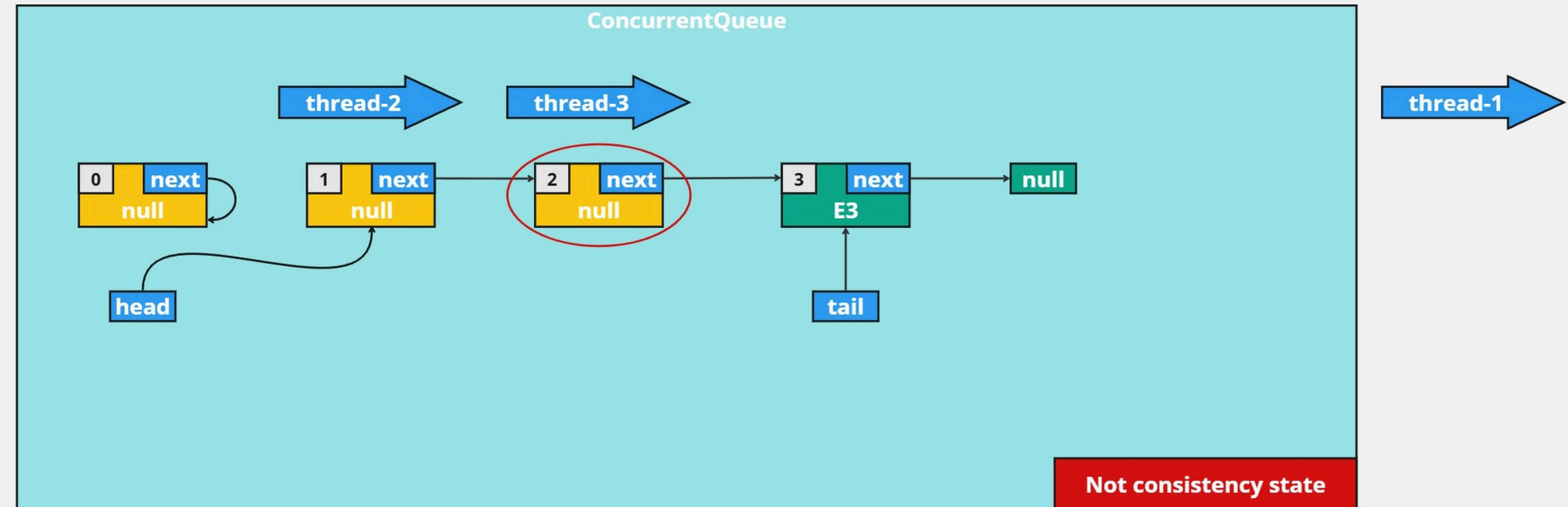
```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}
```

### thread-3

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}
```



#### thread-1

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}
```

#### thread-2

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}
```

Annotations:

- Annotation 1: Points to the line `if (head.compareAndSet(previous, next)) {` in the `dequeue()` method of thread-2.
- Annotation 2: Points to the line `final Node<E> nextHead = previousHead.next.get();` in the `dequeue()` method of thread-2.
- Annotation E2: Points to the line `if (element != null && nextHead.value.compareAndSet(element, null)) {` in the `dequeue()` method of thread-2.

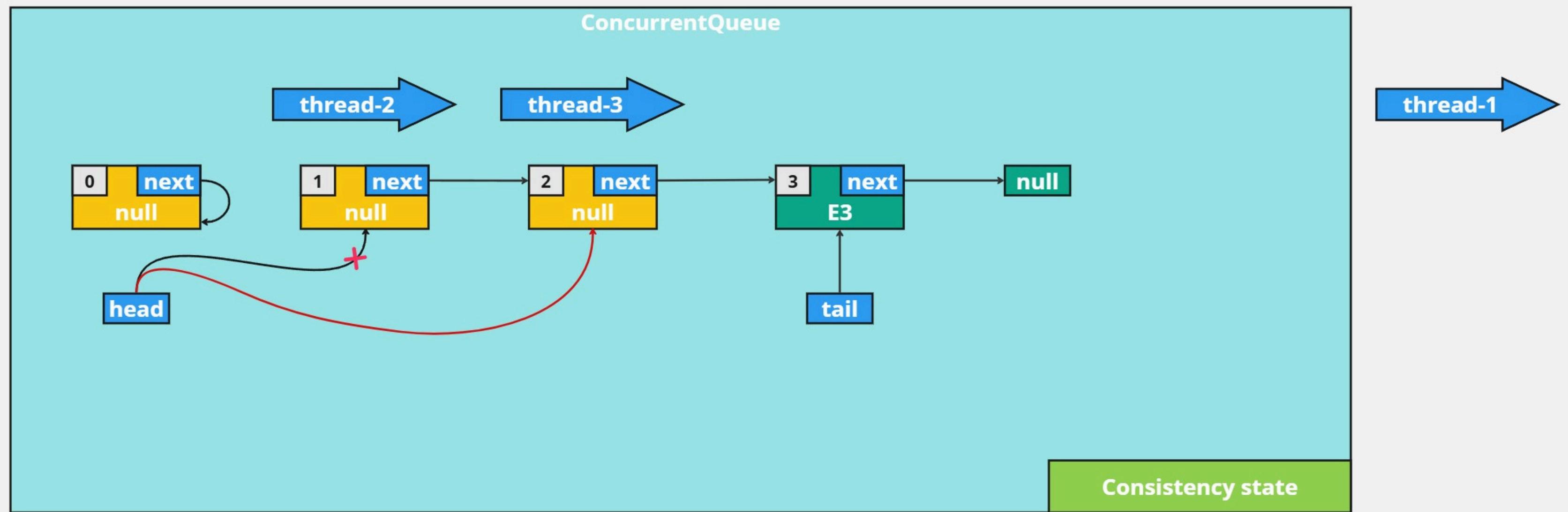
#### thread-3

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}
```

Annotations:

- Annotation 1: Points to the line `if (head.compareAndSet(previous, next)) {` in the `dequeue()` method of thread-3.
- Annotation 2: Points to the line `final Node<E> nextHead = previousHead.next.get();` in the `dequeue()` method of thread-3.
- Annotation E2: Points to the line `if (element != null && nextHead.value.compareAndSet(element, null)) {` in the `dequeue()` method of thread-3.



### thread-1

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}
```

### thread-2

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}
```

1

2

E2

### thread-3

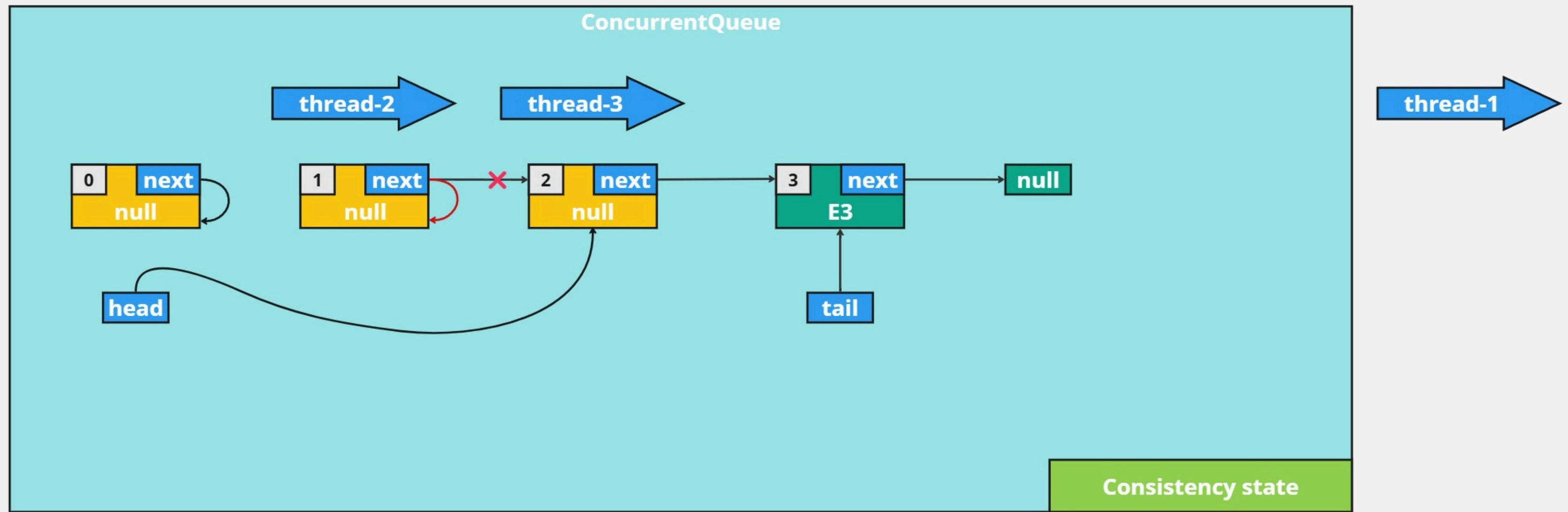
```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}
```

1

2

E2



### thread-1

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}
```

### thread-2

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}
```

Annotations for thread-2:

- Callout 1: Points to the first call to `updateHead`.
- Callout 2: Points to the second call to `updateHead`.
- Callout E2: Points to the `return of(element);` statement.

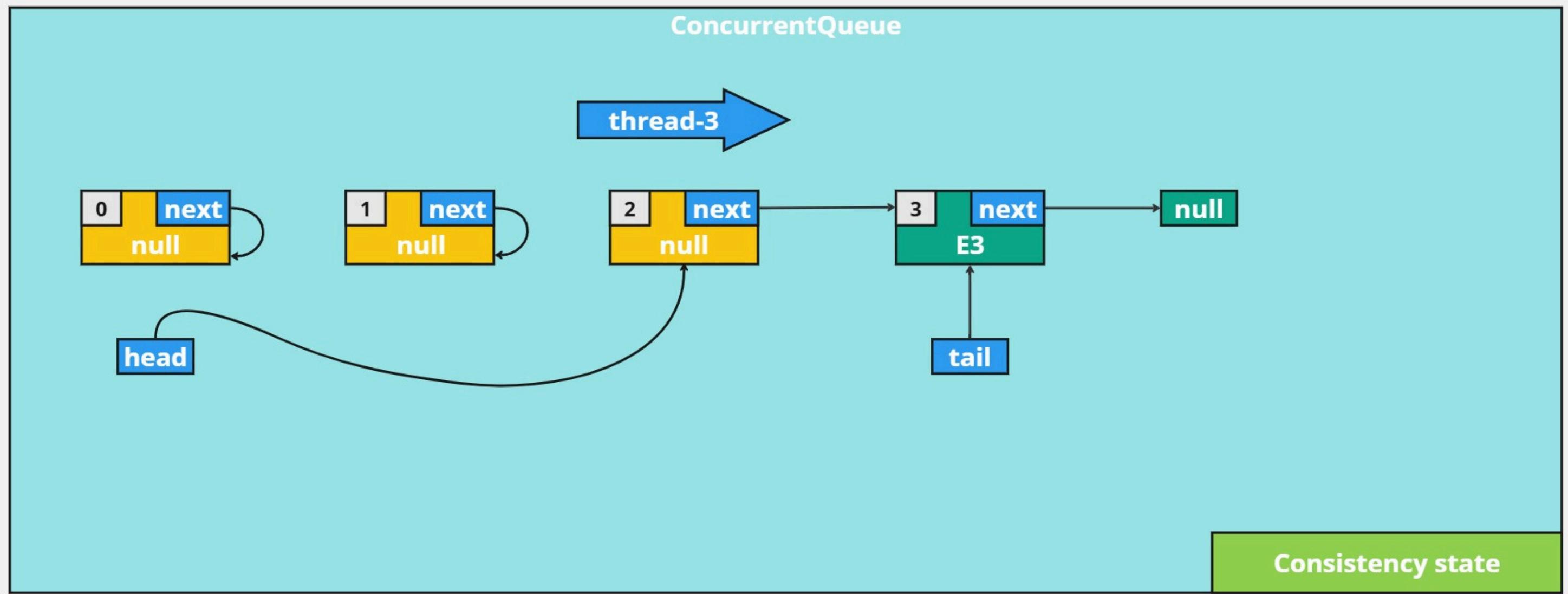
### thread-3

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}
```

Annotations for thread-3:

- Callout 1: Points to the first call to `updateHead`.
- Callout 2: Points to the second call to `updateHead`.
- Callout E2: Points to the `return of(element);` statement.



#### thread-1

```

public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}

```

#### thread-2

```

public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}

```

Annotations:

- Step 1: A dashed arrow labeled '1' points from the 'head.get()' call in thread-2 to the 'head' field in thread-1.
- Step 2: A dashed arrow labeled '2' points from the 'nextHead.get()' call in thread-2 to the 'next' field in thread-1.
- Annotation E2: A dashed arrow labeled 'E2' points from the 'element.get()' call in thread-2 to the 'value' field in thread-1.

#### thread-3

```

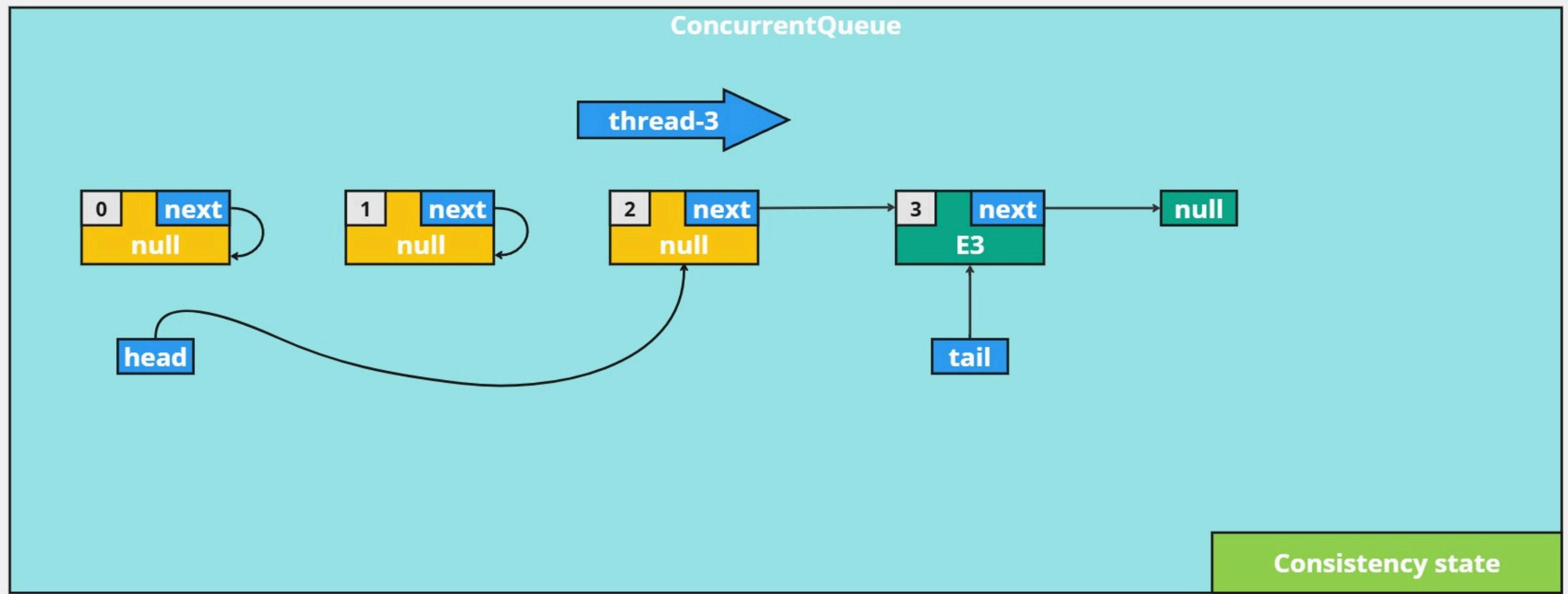
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}

```

Annotations:

- Step 1: A dashed arrow labeled '1' points from the 'head.get()' call in thread-3 to the 'head' field in thread-1.
- Step 2: A dashed arrow labeled '2' points from the 'nextHead.get()' call in thread-3 to the 'next' field in thread-1.
- Annotation E2: A dashed arrow labeled 'E2' points from the 'element.get()' call in thread-3 to the 'value' field in thread-1.



### thread-1

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}
```

### thread-2

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}
```

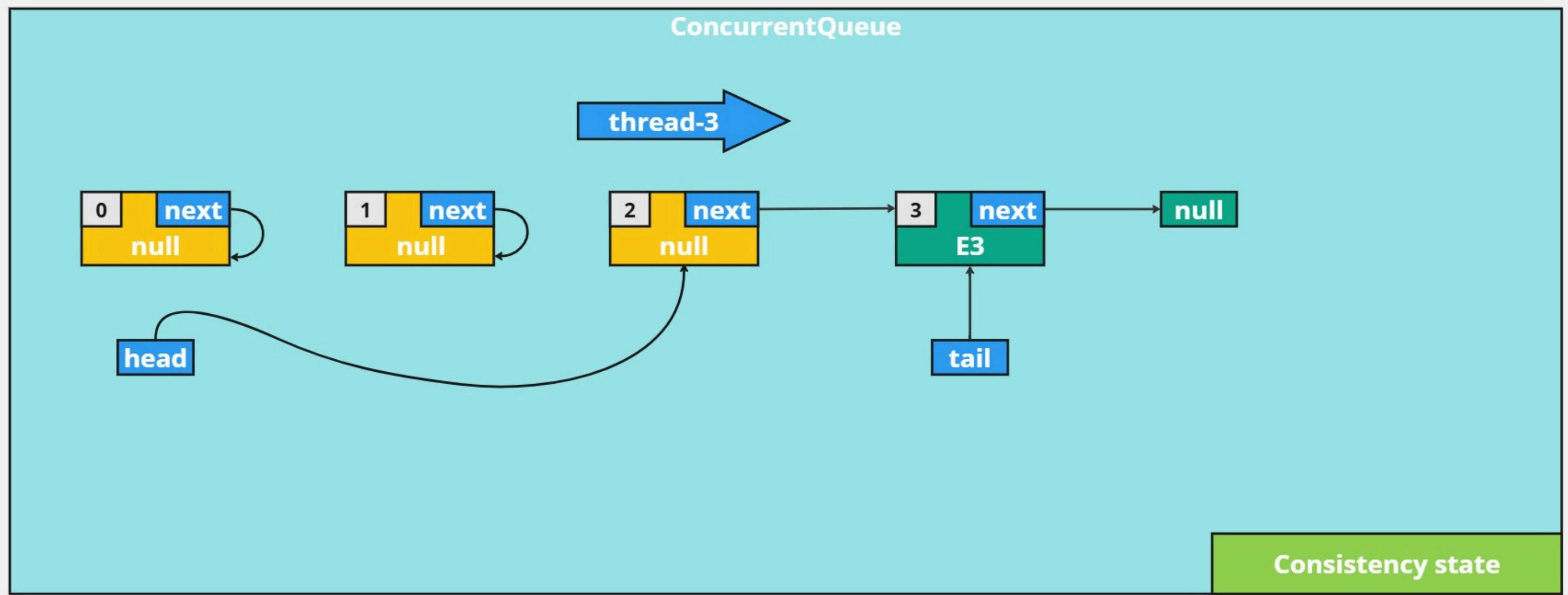
### thread-3

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}
```

Annotations for thread-3 highlight specific code segments:

- 1**: Points to the first `if (head.compareAndSet(previous, next))` check.
- 2**: Points to the second `if (head.compareAndSet(previous, next))` check.
- E2**: Points to the `nextHead.value.get()` call in the second `if` block of thread-2's code.
- E3**: Points to the `nextHead.value.get()` call in the second `if` block of thread-3's code.
- 1**: Points to the `nextHead.value.compareAndSet(element, null)` call in the second `if` block of thread-2's code.
- 2**: Points to the `nextHead.value.compareAndSet(element, null)` call in the second `if` block of thread-3's code.



### thread-1

```

public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}

```

### thread-2

```

public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}

```

### thread-3

```

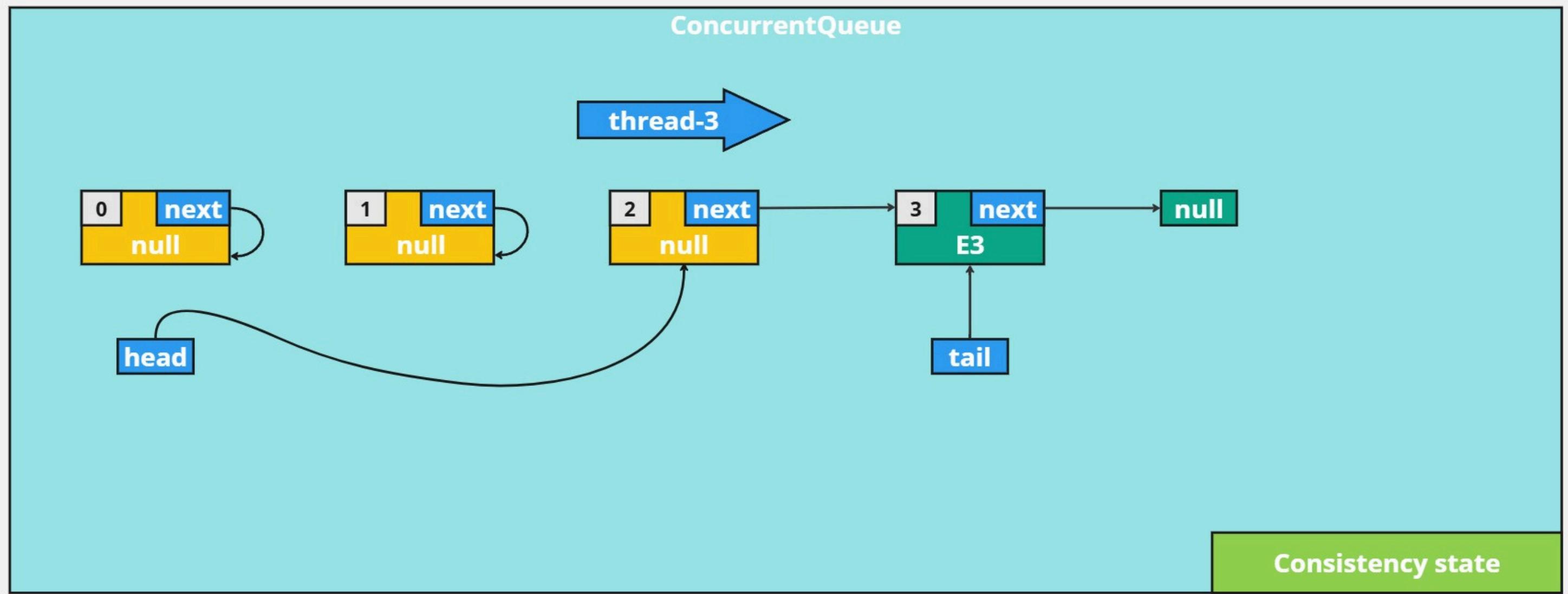
1
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}

```

Annotations:

- Annotation 1: Points to the first call to updateHead in thread-3's dequeue method.
- Annotation 2: Points to the second call to updateHead in thread-3's dequeue method.
- Annotation E2: Points to the second call to updateHead in thread-2's dequeue method.



### thread-1

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}
```

### thread-2

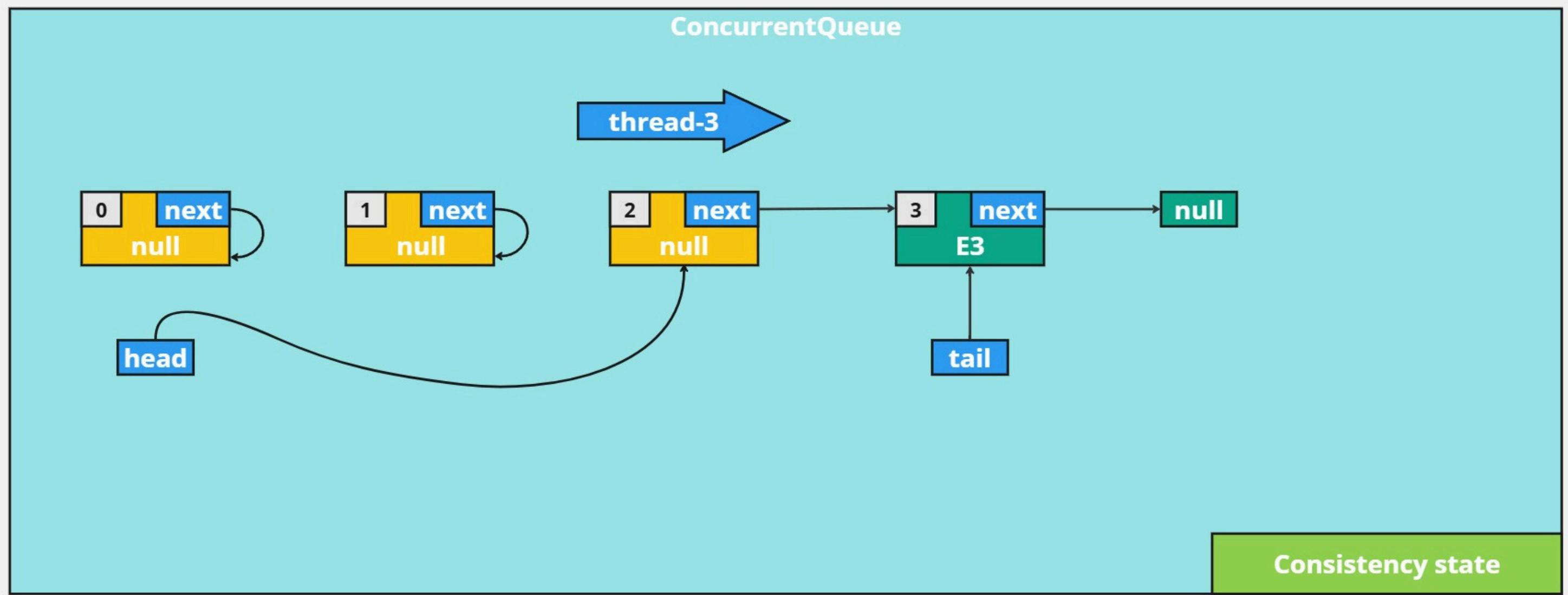
```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}
```

### thread-3

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get(); 2
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}
```



### thread-1

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}
```

### thread-2

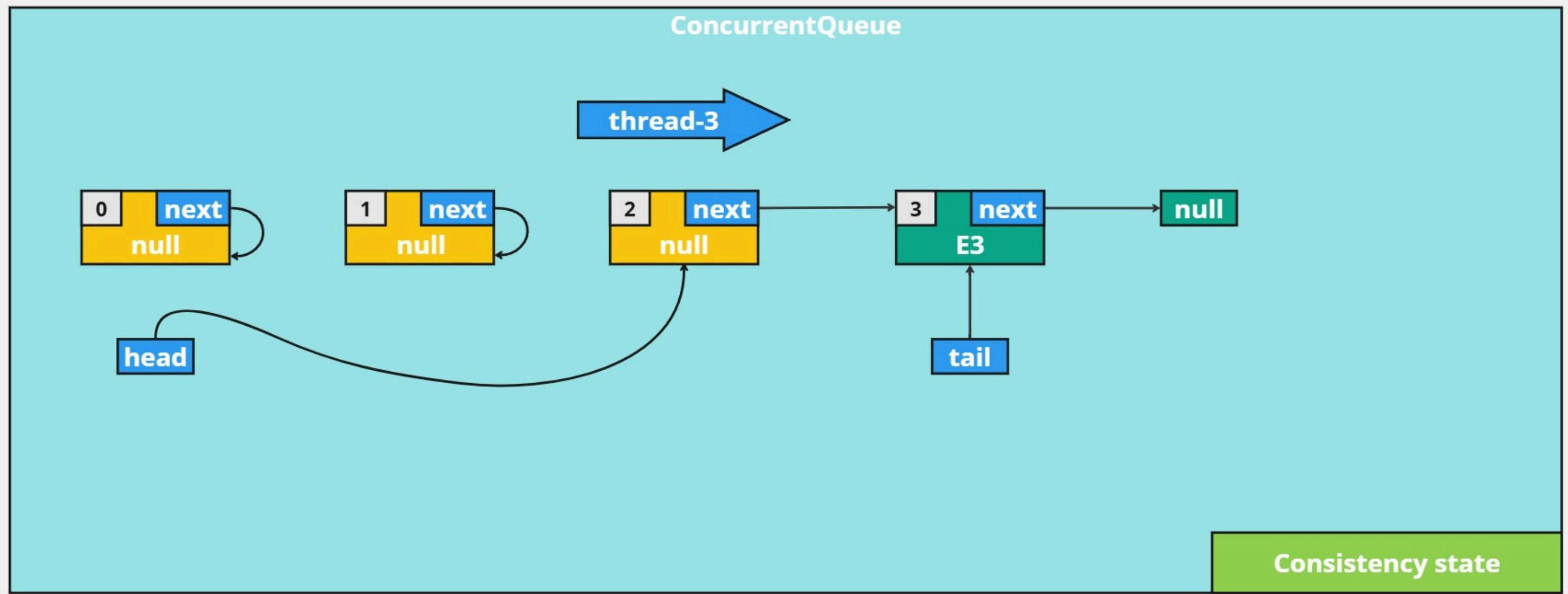
```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}
```

### thread-3

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}
```



### thread-1

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}
```

### thread-2

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}
```

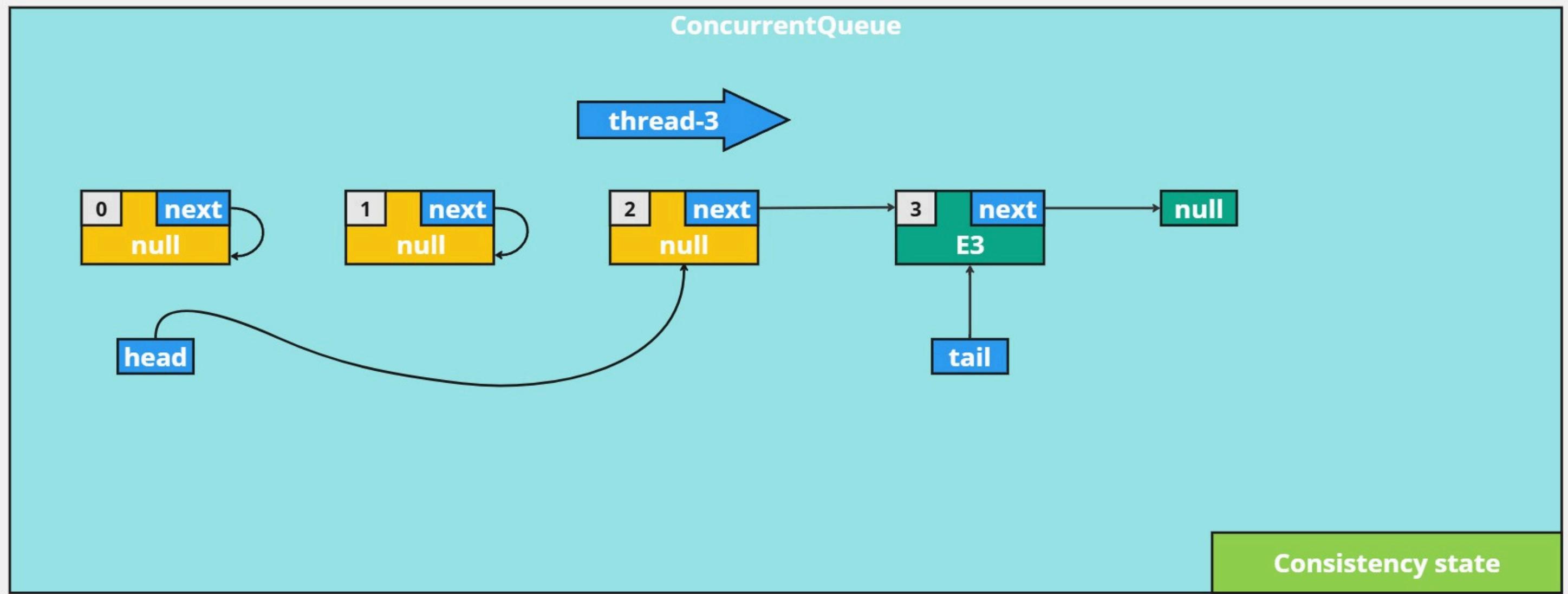
### thread-3

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}
```

thread-1

thread-2



### thread-1

```

public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}

```

### thread-2

```

public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}

```

### thread-3

```

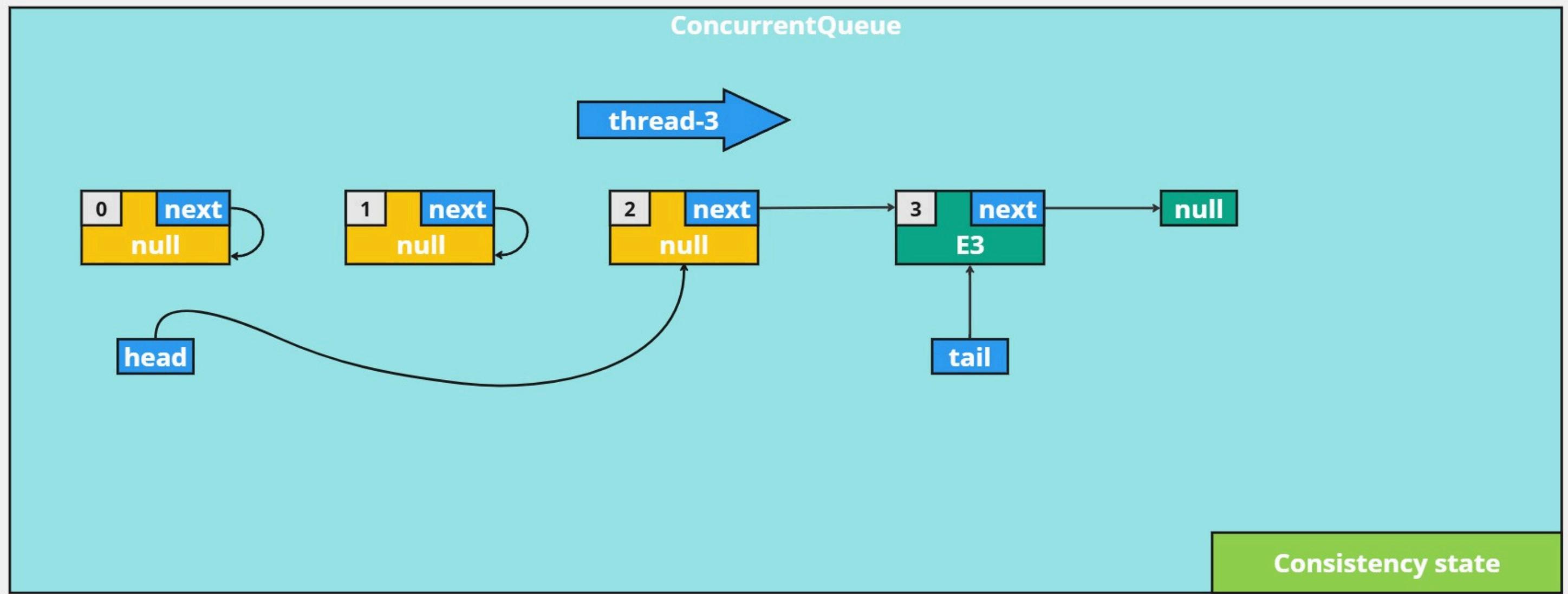
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}

```

Annotations:

- Thread-1: A dashed line connects the 'head' field in thread-1's code to the 'head' pointer in the diagram.
- Thread-2: A dashed line connects the 'head' field in thread-2's code to the 'head' pointer in the diagram.
- Thread-3:
  - A box labeled '2' is placed over the 'head' field in thread-3's code.
  - A box labeled '3' is placed over the 'tail' pointer in thread-3's code.
  - A red box highlights the line `final E element = nextHead.value.get();`.
  - A box labeled 'E3' is placed over the value of node 3 in the diagram.



### thread-1

```

public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}

```

### thread-2

```

public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}

```

### thread-3

```

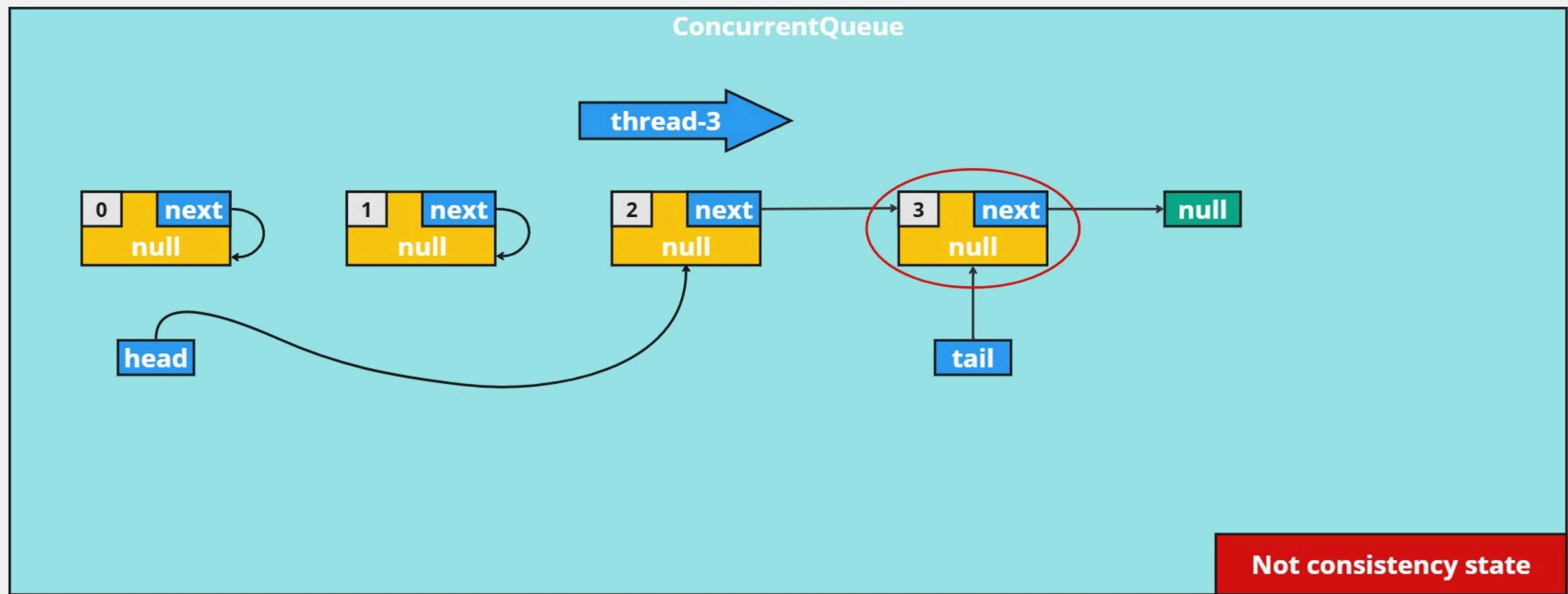
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}

```

Annotations for thread-3:

- Annotation 2: A dashed line connects the 'head' field of the code to the 'head' pointer in the diagram.
- Annotation 3: A dashed line connects the 'tail' field of the code to the 'tail' pointer in the diagram.
- Annotation E3: A dashed line connects the 'value' field of node 3 in the code to the value 'E3' in the diagram.



### thread-1

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}
```

### thread-2

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}
```

### thread-3

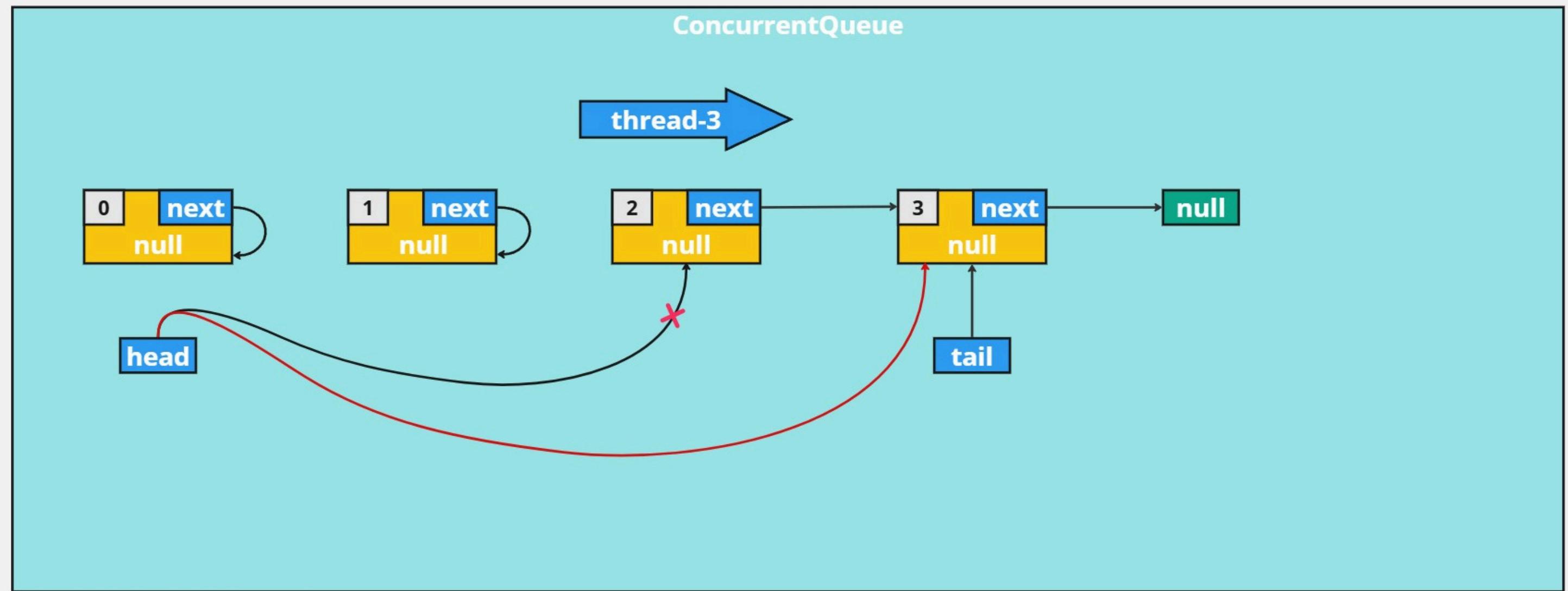
```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}
```

2

3

E3



### thread-1

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}
```

### thread-2

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

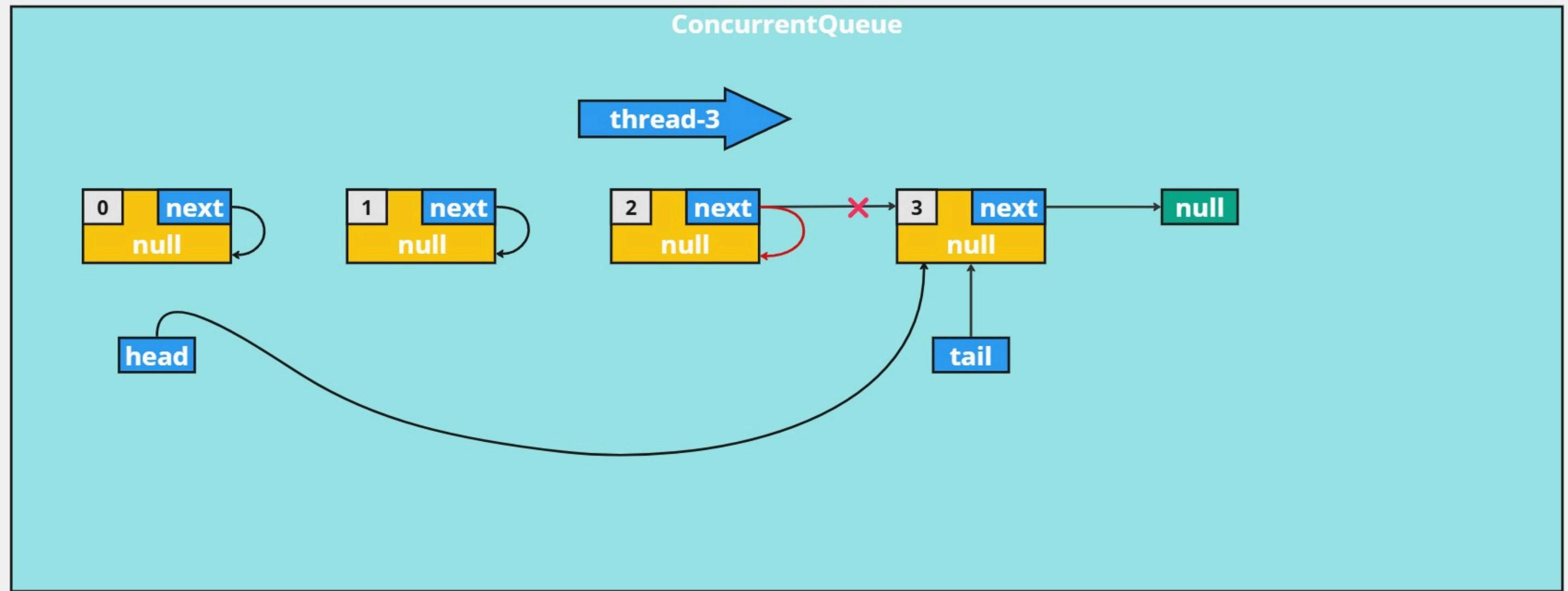
private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}
```

### thread-3

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}
```

2  
3  
E3



### thread-1

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}
```

### thread-2

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}
```

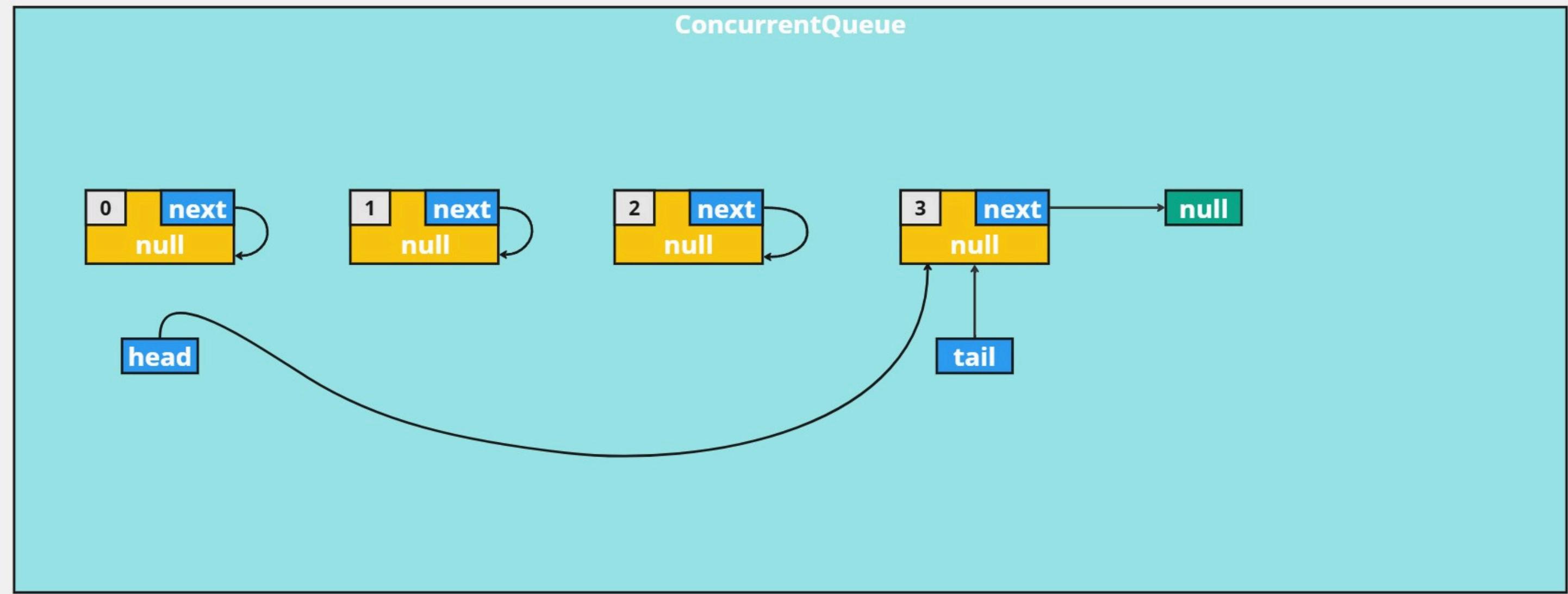
### thread-3

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(previous);
    }
}
```

Annotations for thread-3:

- Step 2: A dashed line connects the 'head' field of the code to the 'head' pointer in the diagram.
- Step 3: A dashed line connects the 'tail' field of the code to the 'tail' pointer in the diagram.
- E3: A red box highlights the line 'return of(element);' in the code.



### thread-1

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}
```

### thread-2

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}
```

### thread-3

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

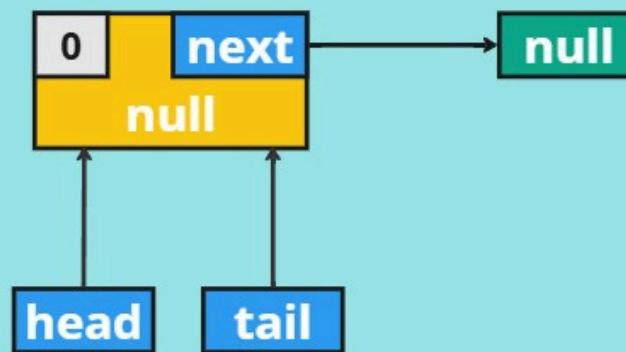
private void updateHead(final Node<E> previous, final Node<E> next) {
    if (head.compareAndSet(previous, next)) {
        previous.next.set(next);
    }
}
```

Annotations for thread-3:

- Annotation 2: A dashed line connects the 'head.get()' call in the first iteration of the loop to the 'head.get()' call in the second iteration.
- Annotation 3: A dashed line connects the 'tail.get()' call in the first iteration of the loop to the 'tail.get()' call in the second iteration.
- Annotation E3: A dashed line connects the 'nextHead.value.get()' call in the first iteration of the loop to the 'nextHead.value.get()' call in the second iteration.

## ConcurrentQueue

dequeue()



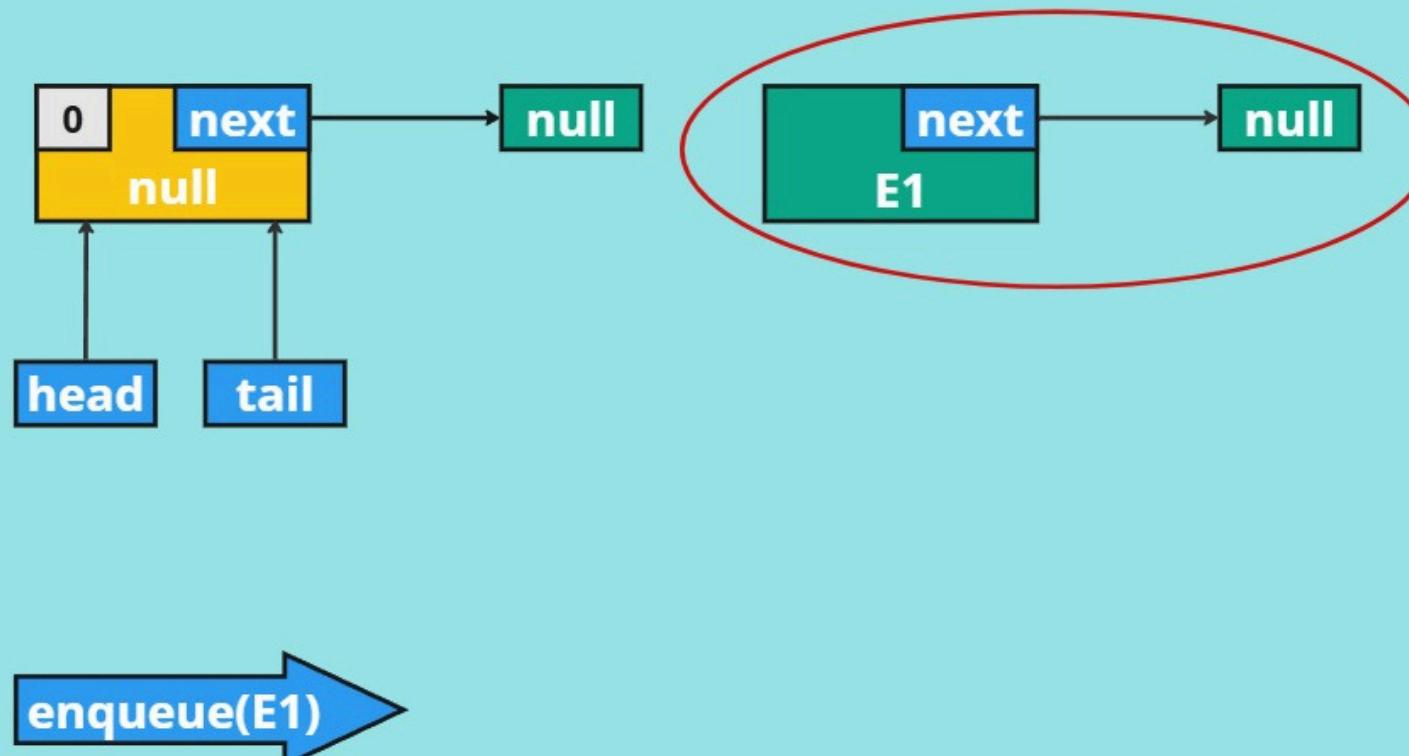
enqueue(E1)

```
public Optional<E> dequeue() {  
    while (true) {  
        final Node<E> previousHead = head.get();  
        final Node<E> nextHead = previousHead.next.get();  
        if (previousHead == tail.get()) {  
            return empty();  
        }  
        final E element = nextHead.value.get();  
        if (element != null && nextHead.value.compareAndSet(element, null)) {  
            updateHead(previousHead, nextHead);  
            return of(element);  
        } else {  
            updateHead(previousHead, nextHead);  
        }  
    }  
}
```

```
public void enqueue(final E element) {  
    final Node<E> newNode = new Node<E>(element);  
    while (true) {  
        final Node<E> previousTail = tail.get();  
        if (previousTail.next.compareAndSet(null, newNode)) {  
            tail.compareAndSet(previousTail, newNode);  
            return;  
        } else {  
            tail.compareAndSet(previousTail, previousTail.next.get());  
        }  
    }  
}
```

## ConcurrentQueue

**dequeue()**



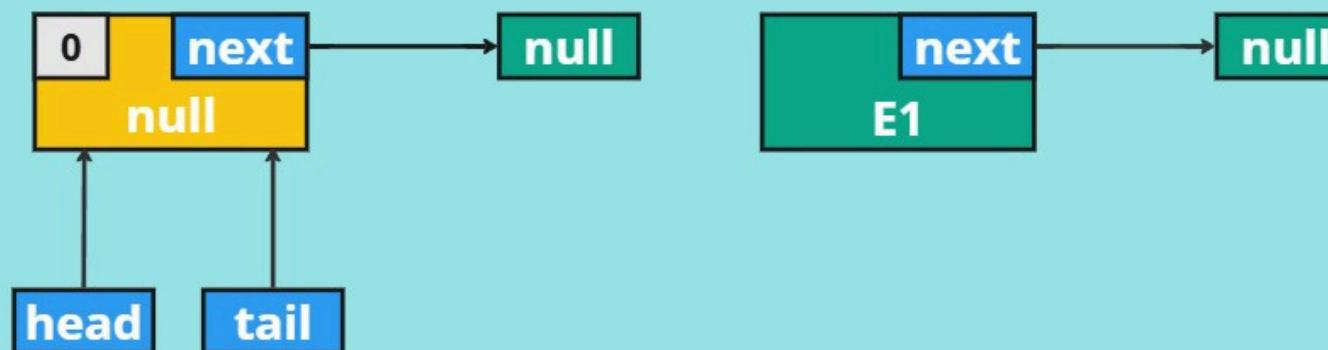
**enqueue(E1)**

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}
```

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

## ConcurrentQueue

dequeue()



enqueue(E1)

```

public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

```

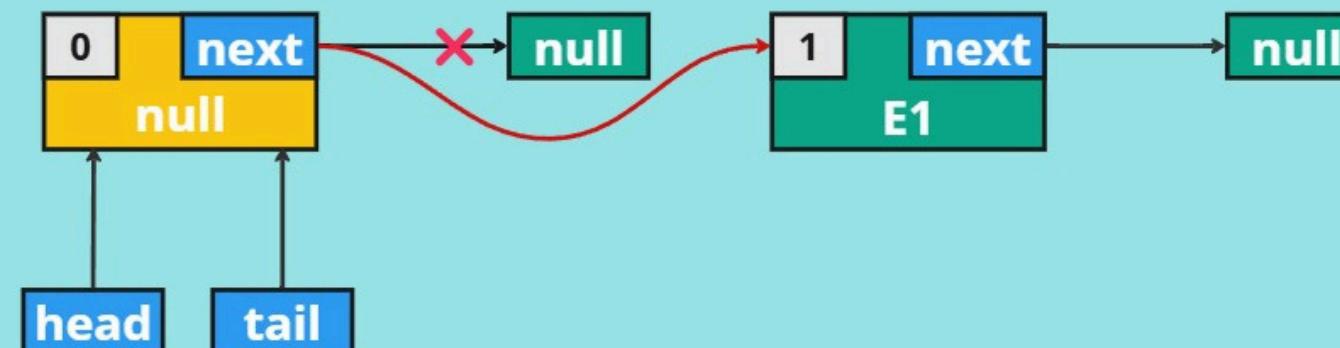
```

public void enqueue(final E element) {
    final Node<E> newNode = new Node<E>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}

```

## ConcurrentQueue

**dequeue()**



**enqueue(E1)**

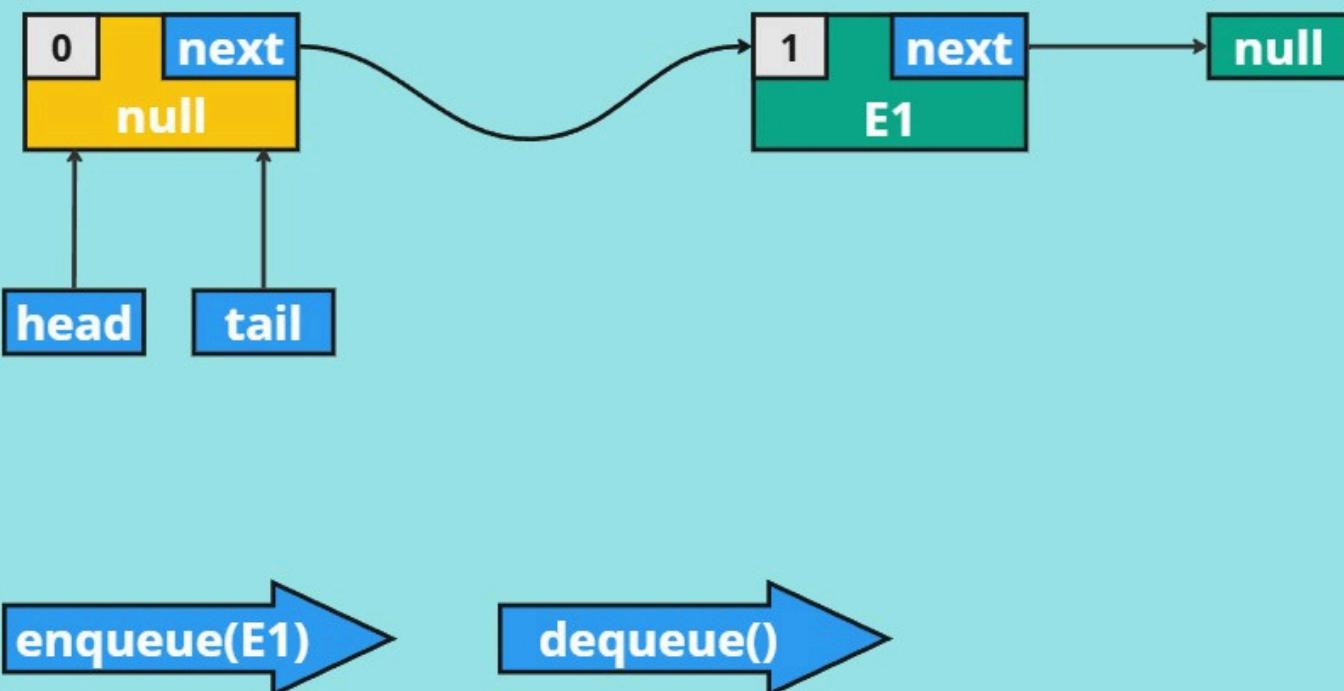
```

public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}
  
```

```

public void enqueue(final E element) {
    final Node<E> newNode = new Node<E>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
  
```

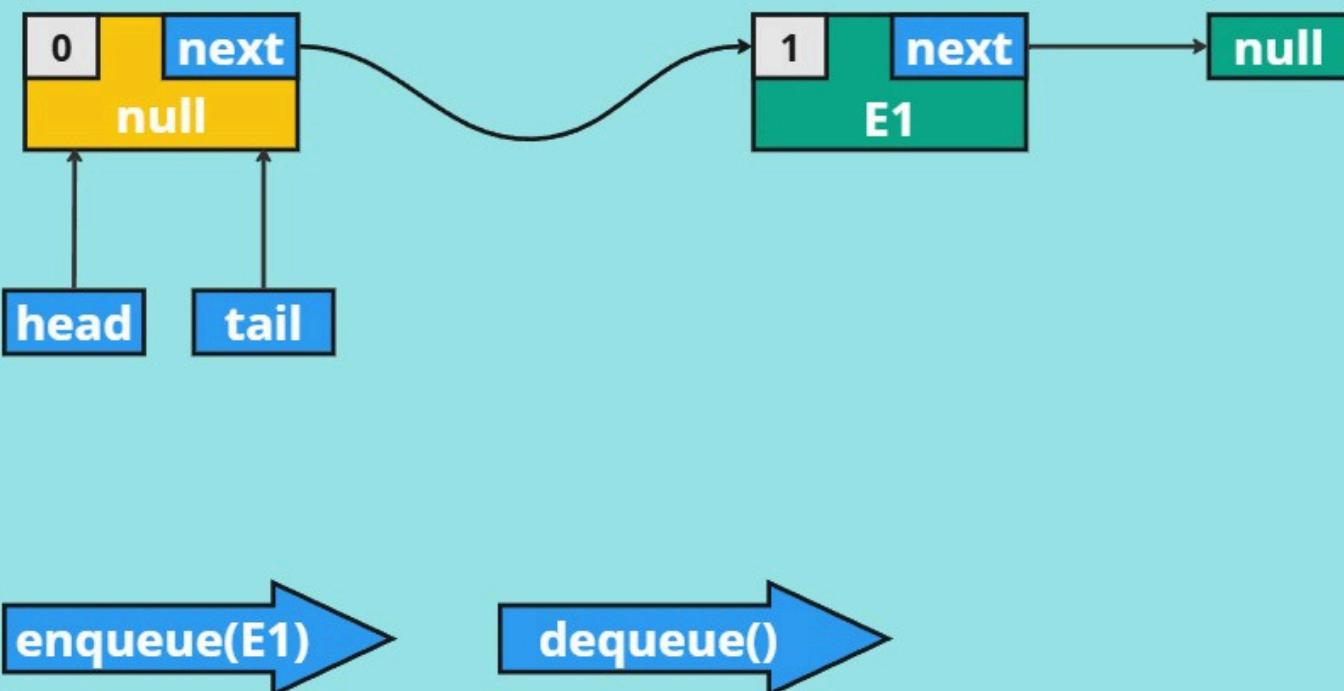
## ConcurrentQueue



```
public Optional<E> dequeue() {  
    while (true) {  
        final Node<E> previousHead = head.get(); ----- 0  
        final Node<E> nextHead = previousHead.next.get();  
        if (previousHead == tail.get()) {  
            return empty();  
        }  
        final E element = nextHead.value.get();  
        if (element != null && nextHead.value.compareAndSet(element, null)) {  
            updateHead(previousHead, nextHead);  
            return of(element);  
        } else {  
            updateHead(previousHead, nextHead);  
        }  
    }  
}
```

```
public void enqueue(final E element) {  
    final Node<E> newNode = new Node<E>(element);  
    while (true) { ----- 0  
        final Node<E> previousTail = tail.get();  
        if (previousTail.next.compareAndSet(null, newNode)) {  
            tail.compareAndSet(previousTail, newNode);  
            return;  
        } else {  
            tail.compareAndSet(previousTail, previousTail.next.get());  
        }  
    }  
}
```

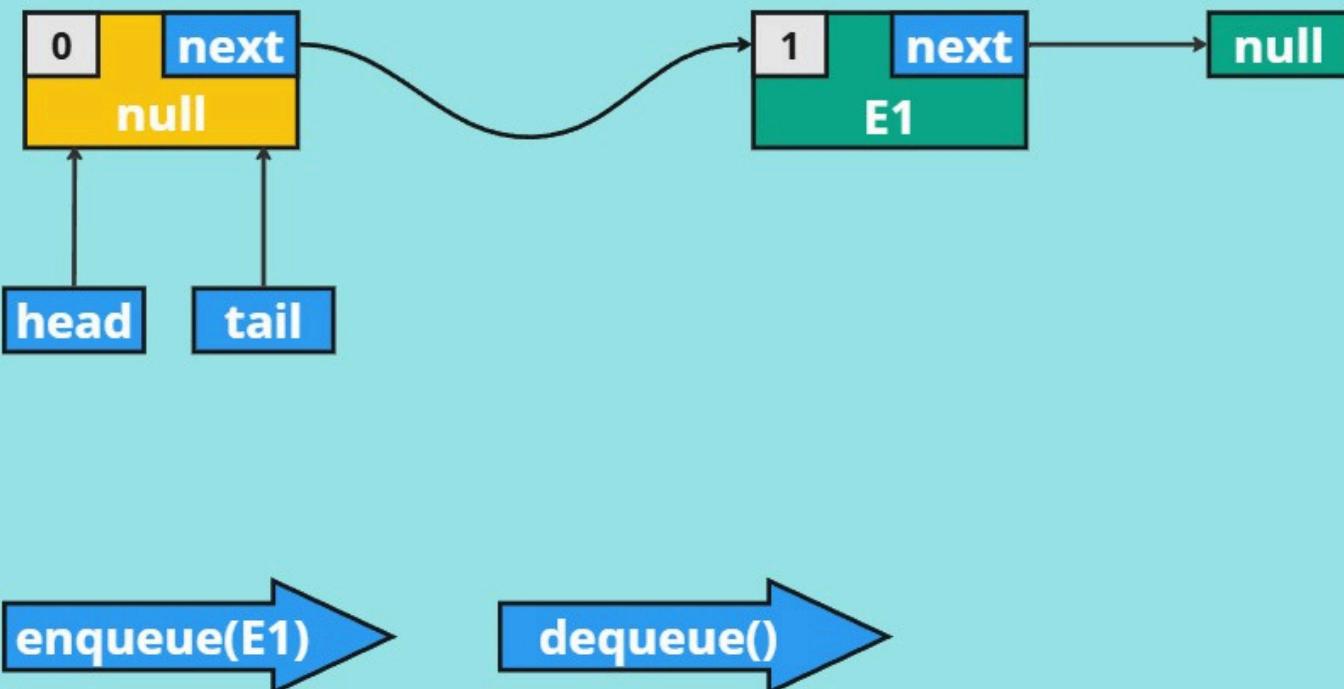
## ConcurrentQueue



```
public Optional<E> dequeue() {  
    while (true) {  
        final Node<E> previousHead = head.get();  
        final Node<E> nextHead = previousHead.next.get();  
        if (previousHead == tail.get()) {  
            return empty();  
        }  
        final E element = nextHead.value.get();  
        if (element != null && nextHead.value.compareAndSet(element, null)) {  
            updateHead(previousHead, nextHead);  
            return of(element);  
        } else {  
            updateHead(previousHead, nextHead);  
        }  
    }  
}
```

```
public void enqueue(final E element) {  
    final Node<E> newNode = new Node<E>(element);  
    while (true) {  
        final Node<E> previousTail = tail.get();  
        if (previousTail.next.compareAndSet(null, newNode)) {  
            tail.compareAndSet(previousTail, newNode);  
            return;  
        } else {  
            tail.compareAndSet(previousTail, previousTail.next.get());  
        }  
    }  
}
```

## ConcurrentQueue



```

public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) { 0
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

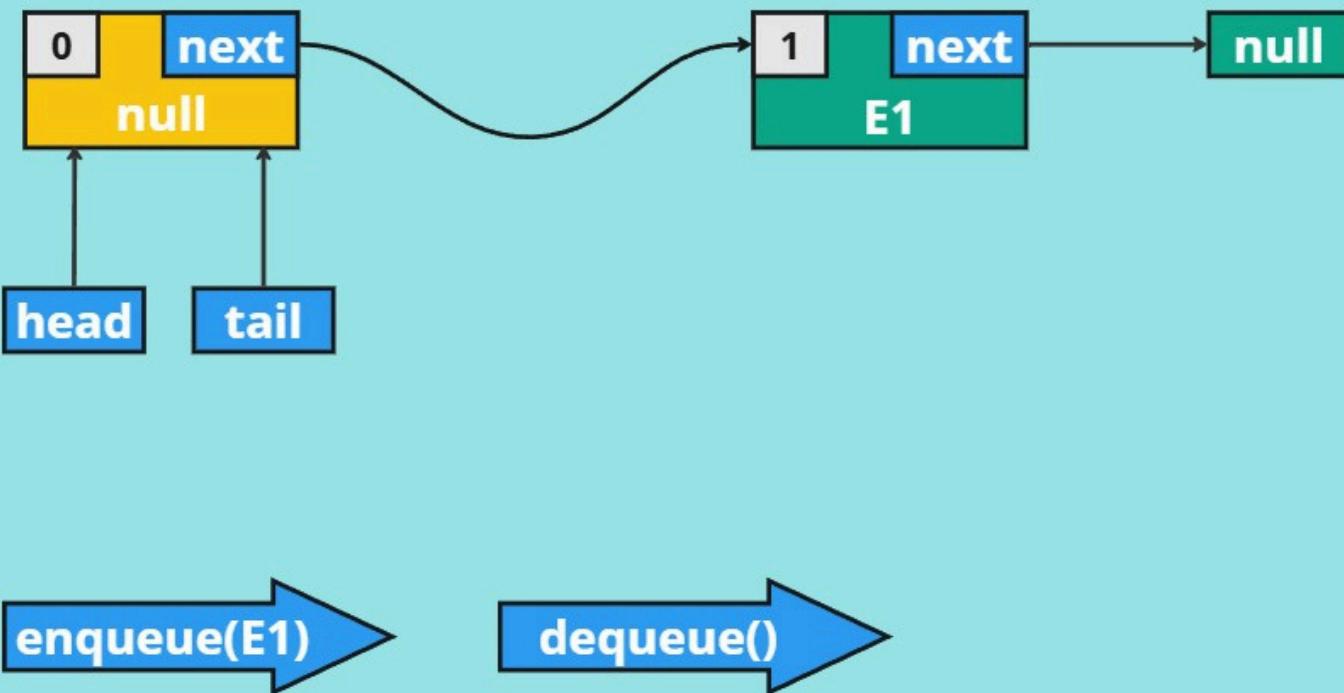
```

```

public void enqueue(final E element) {
    final Node<E> newNode = new Node<E>(element);
    while (true) { 0
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}

```

## ConcurrentQueue



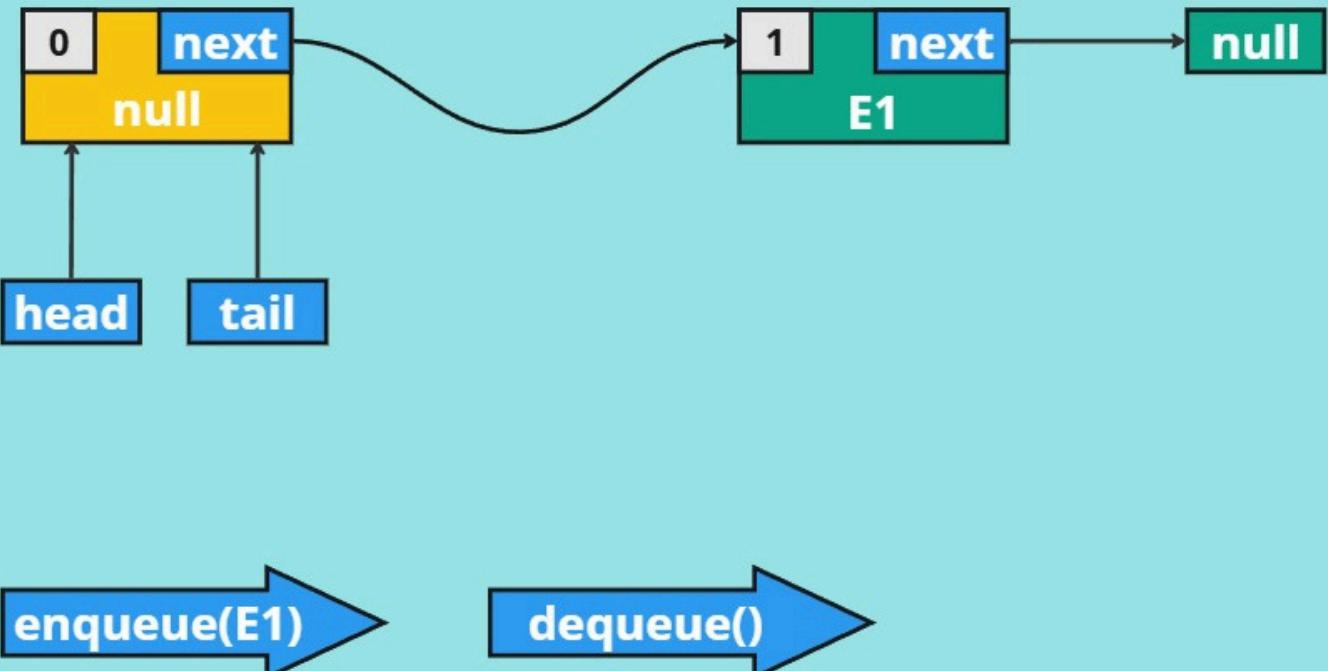
```

public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get(); ----- 0
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get()) { 1
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}
  
```

```

public void enqueue(final E element) {
    final Node<E> newNode = new Node<E>(element);
    while (true) { ----- 0
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
  
```

## ConcurrentQueue

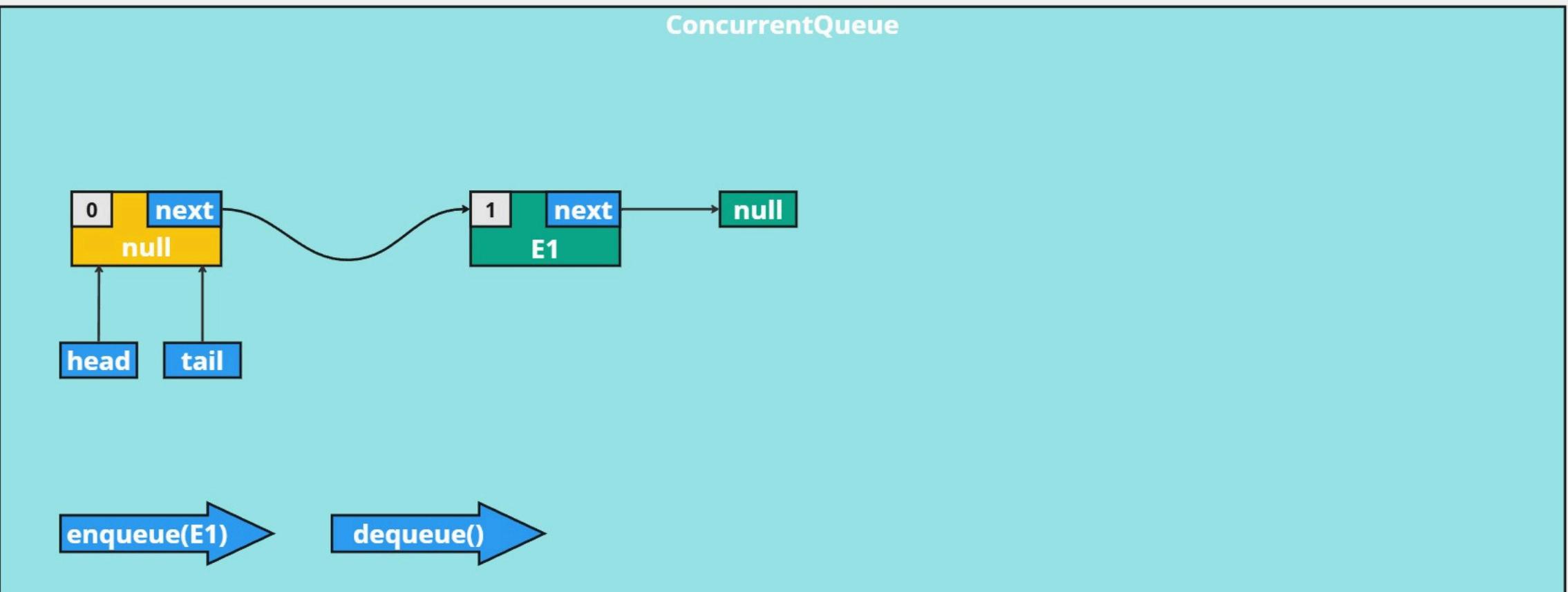


```
public Optional<E> dequeue() {  
    while (true) {  
        final Node<E> previousHead = head.get();  
        final Node<E> nextHead = previousHead.next.get();  
        if (previousHead == tail.get()) {  
            return empty();  
        }  
        final E element = nextHead.value.get();  
        if (element != null && nextHead.value.compareAndSet(element, null)) {  
            updateHead(previousHead, nextHead);  
            return of(element);  
        } else {  
            updateHead(previousHead, nextHead);  
        }  
    }  
}
```



```
public Optional<E> dequeue() {  
    while (true) {  
        final Node<E> previousHead = head.get();  
        final Node<E> nextHead = previousHead.next.get();  
        if (previousHead == tail.get() && nextHead == null) {  
            return empty();  
        }  
        final E element = nextHead.value.get();  
        if (element != null && nextHead.value.compareAndSet(element, null)) {  
            updateHead(previousHead, nextHead);  
            return of(element);  
        } else {  
            updateHead(previousHead, nextHead);  
        }  
    }  
}
```

## ConcurrentQueue



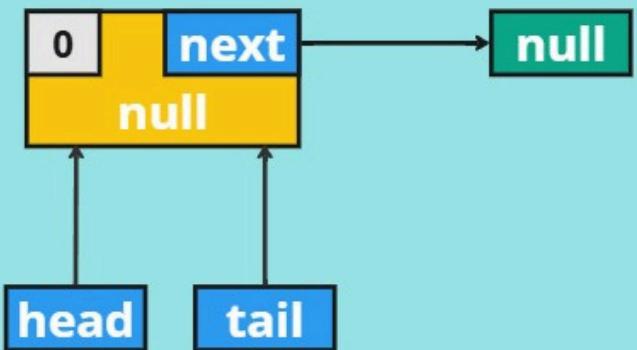
```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == tail.get() && nextHead == null) {
            return empty();
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}
```



```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> previousTail = tail.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == previousTail) {
            if (nextHead == null) {
                return empty();
            } else {
                tail.compareAndSet(previousTail, nextHead);
            }
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}
```

## ConcurrentQueue

dequeue()



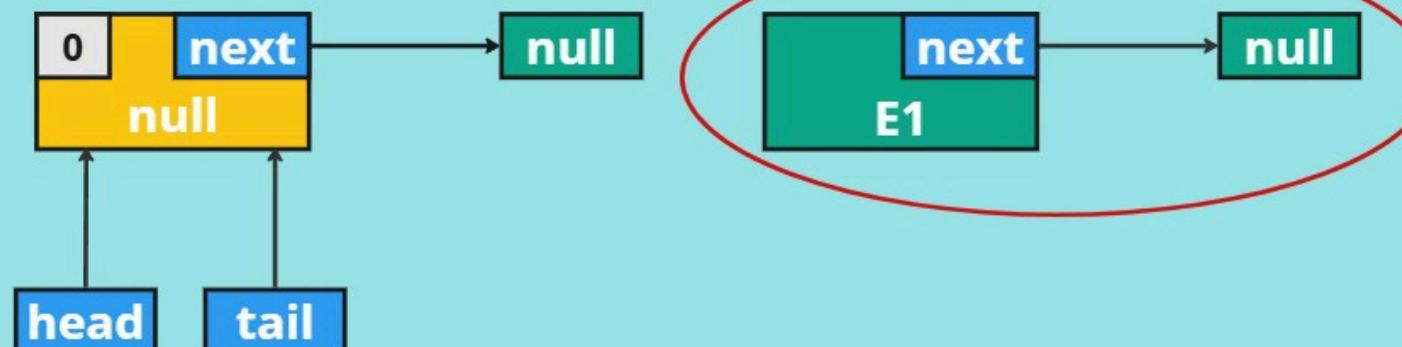
enqueue(E1)

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> previousTail = tail.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == previousTail) {
            if (nextHead == null) {
                return empty();
            } else {
                tail.compareAndSet(previousTail, nextHead);
            }
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}
```

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

## ConcurrentQueue

dequeue()



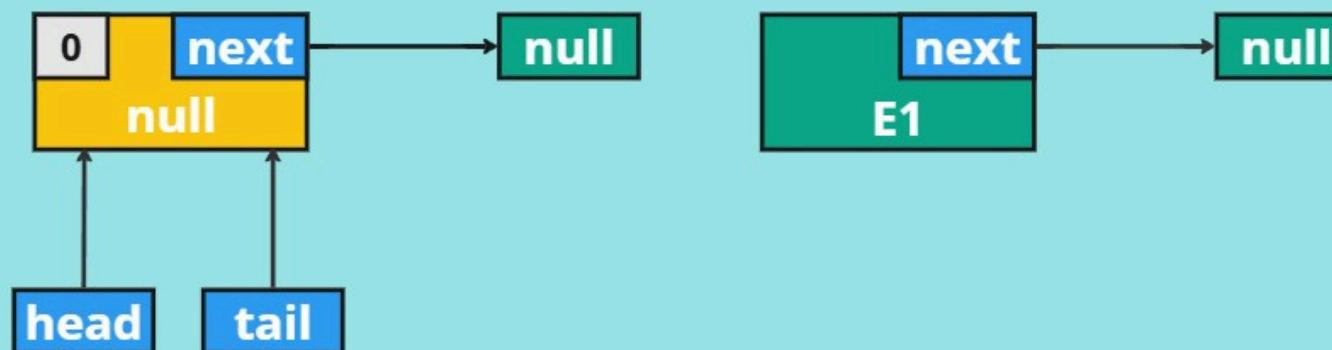
enqueue(E1)

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> previousTail = tail.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == previousTail) {
            if (nextHead == null) {
                return empty();
            } else {
                tail.compareAndSet(previousTail, nextHead);
            }
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}
```

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

## ConcurrentQueue

dequeue()



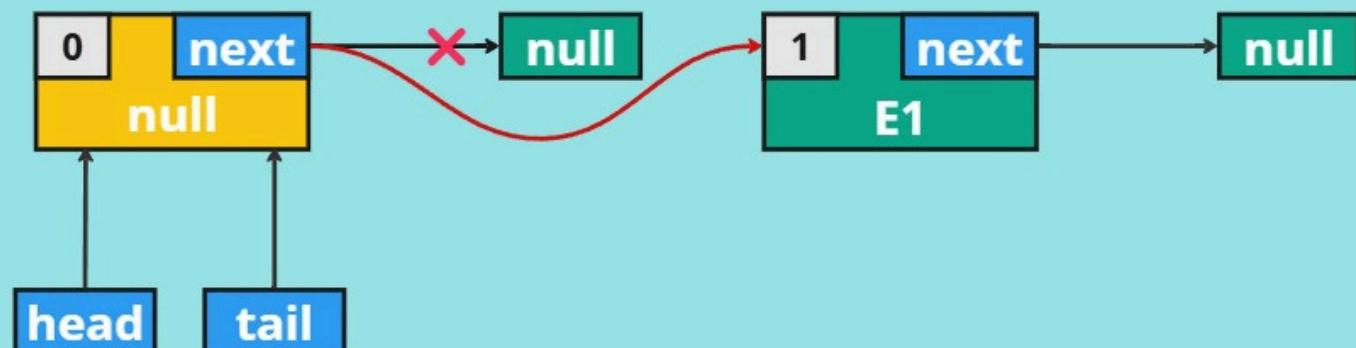
enqueue(E1)

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> previousTail = tail.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == previousTail) {
            if (nextHead == null) {
                return empty();
            } else {
                tail.compareAndSet(previousTail, nextHead);
            }
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}
```

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

## ConcurrentQueue

**dequeue()**

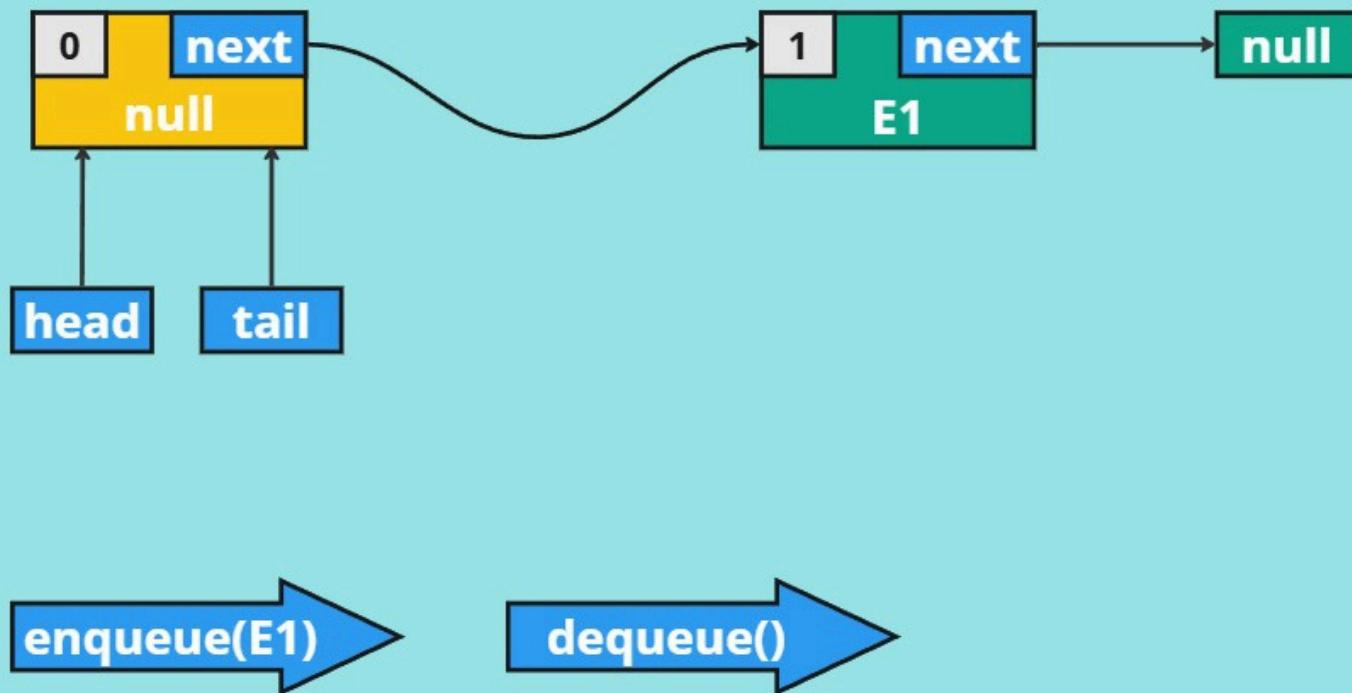


**enqueue(E1)**

```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> previousTail = tail.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == previousTail) {
            if (nextHead == null) {
                return empty();
            } else {
                tail.compareAndSet(previousTail, nextHead);
            }
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}
```

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

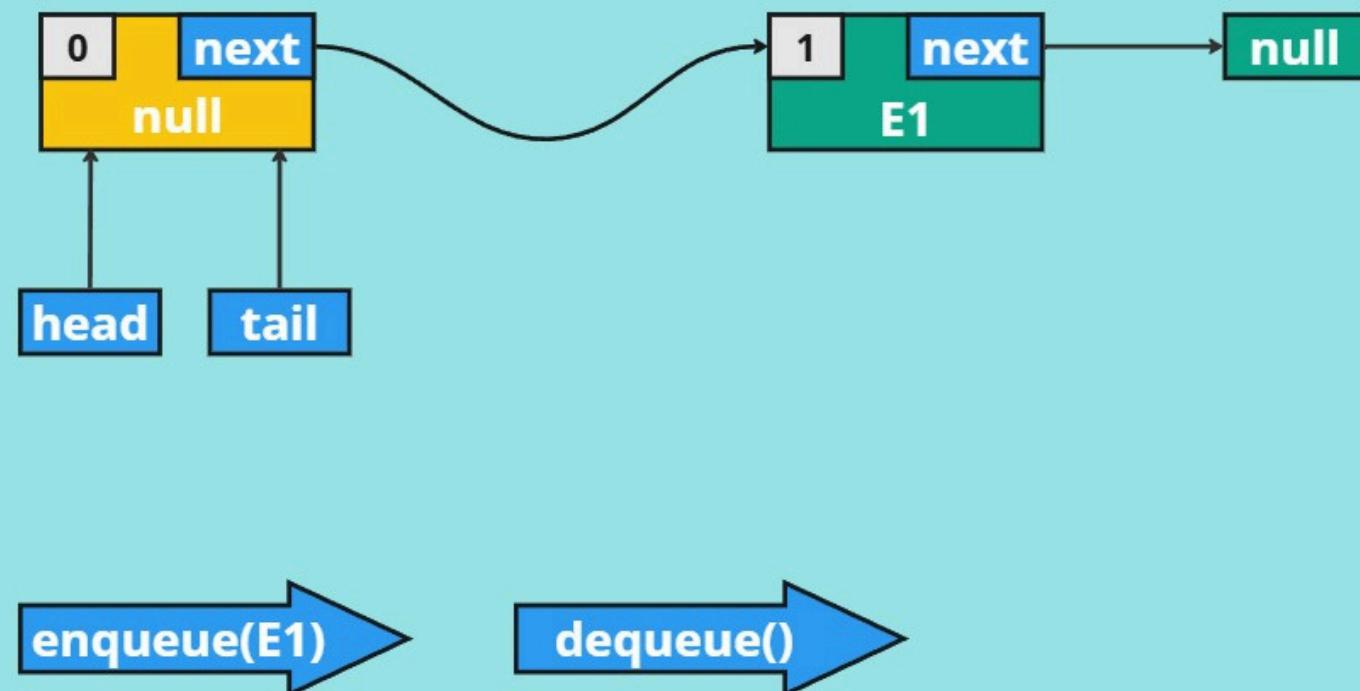
## ConcurrentQueue



```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> previousTail = tail.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == previousTail) {
            if (nextHead == null) {
                return empty();
            } else {
                tail.compareAndSet(previousTail, nextHead);
            }
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}
```

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

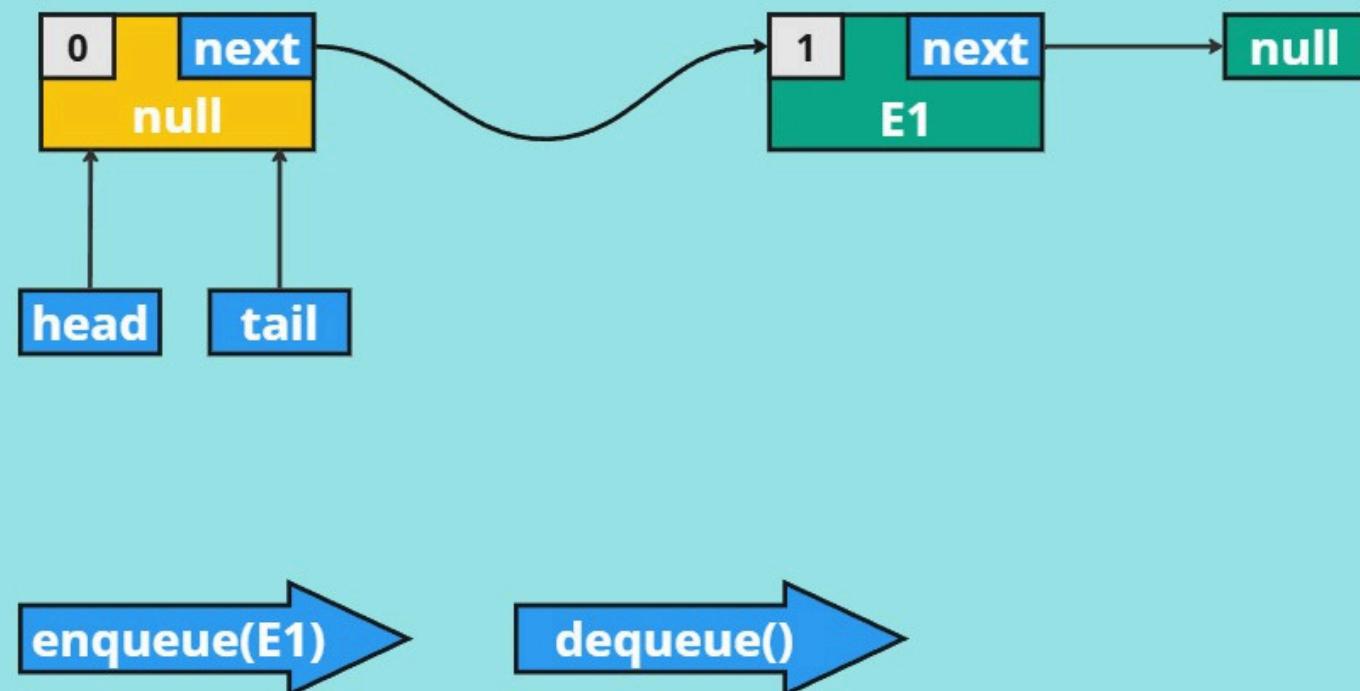
## ConcurrentQueue



```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get(); 0
        final Node<E> previousTail = tail.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == previousTail) {
            if (nextHead == null) {
                return empty();
            } else {
                tail.compareAndSet(previousTail, nextHead);
            }
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}
```

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) { 0
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

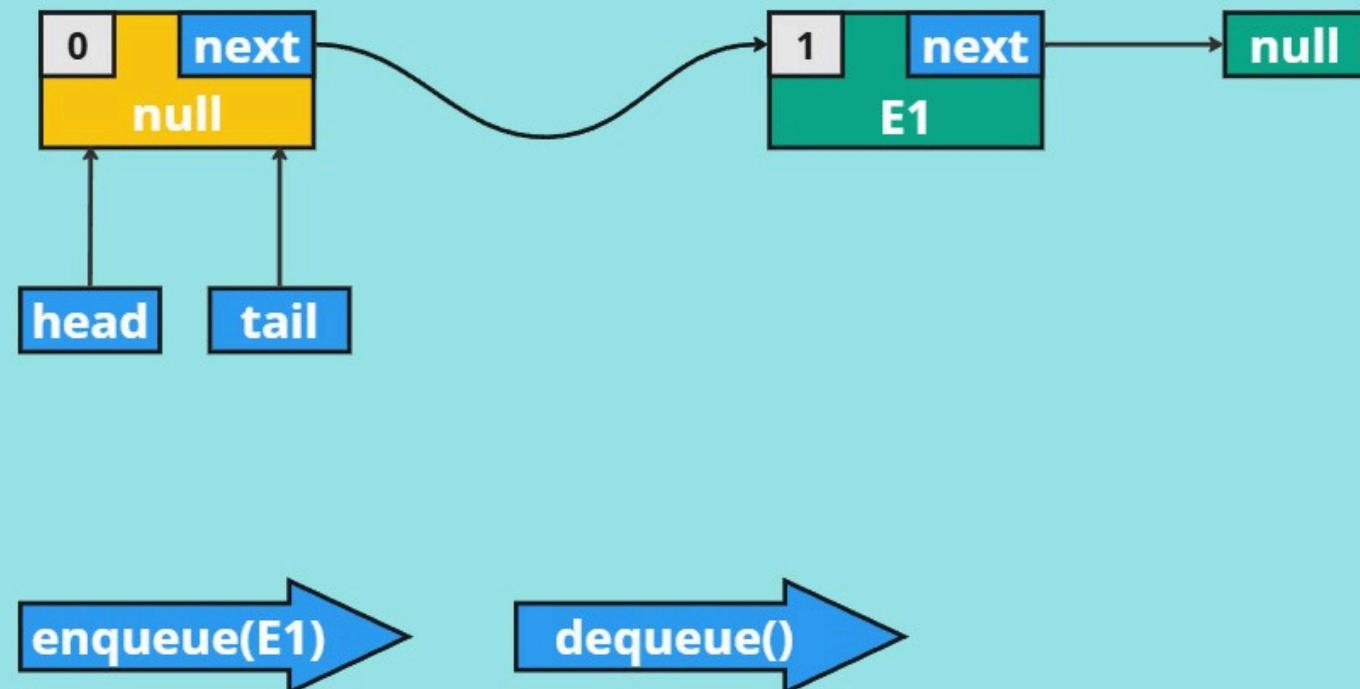
## ConcurrentQueue



```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> previousTail = tail.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == previousTail) {
            if (nextHead == null) {
                return empty();
            } else {
                tail.compareAndSet(previousTail, nextHead);
            }
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}
```

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

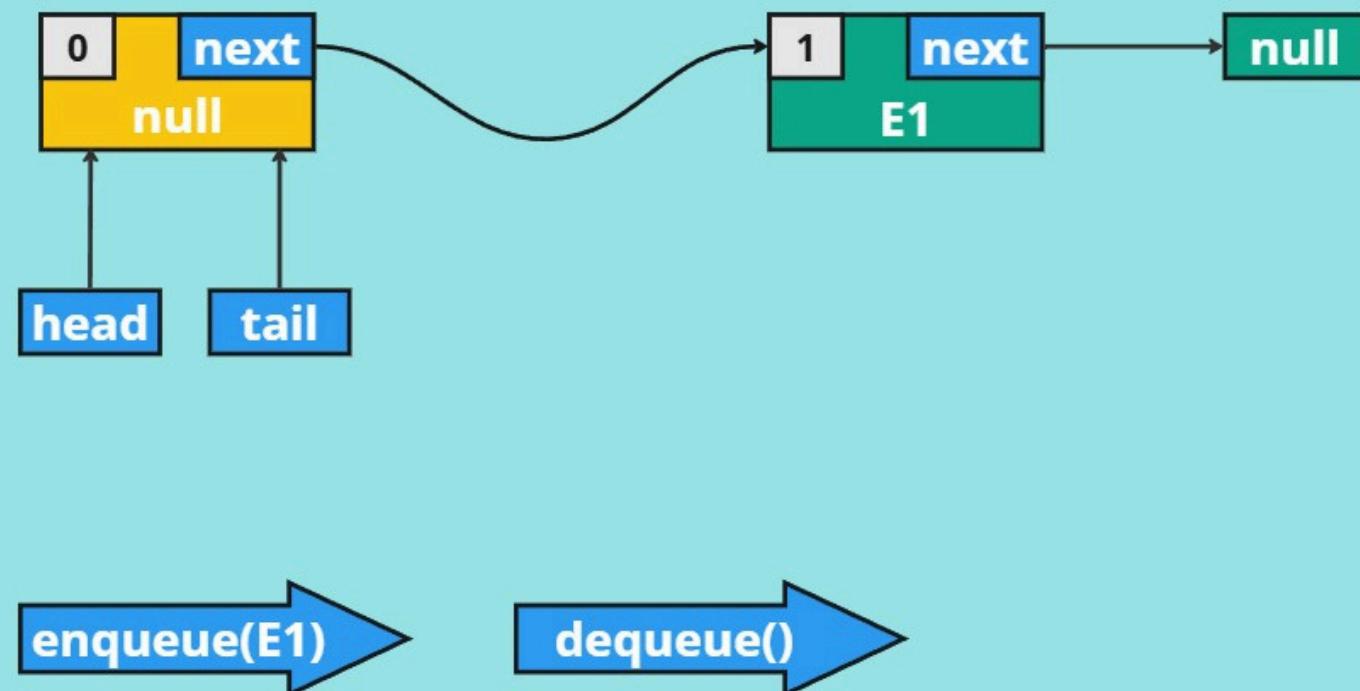
## ConcurrentQueue



```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> previousTail = tail.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == previousTail) {
            if (nextHead == null) {
                return empty();
            } else {
                tail.compareAndSet(previousTail, nextHead);
            }
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}
```

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

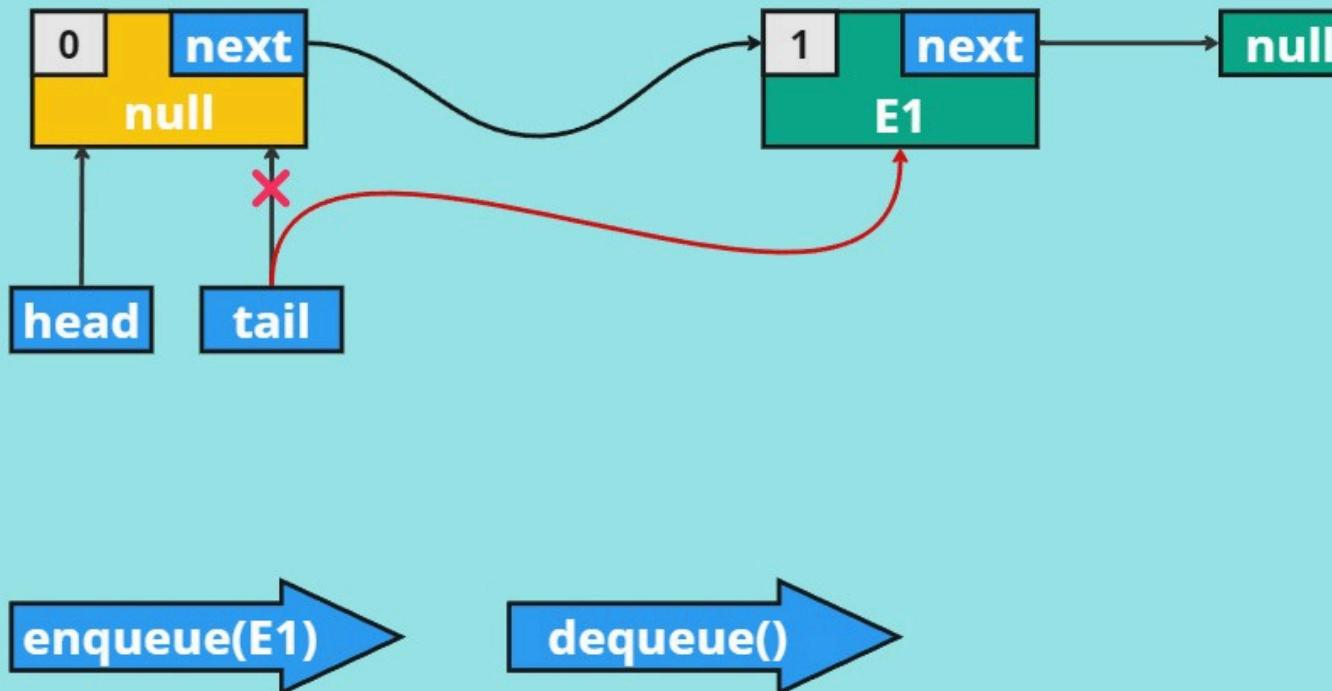
## ConcurrentQueue



```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> previousTail = tail.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == previousTail) {
            if (nextHead == null) {
                return empty();
            } else {
                tail.compareAndSet(previousTail, nextHead);
            }
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}
```

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

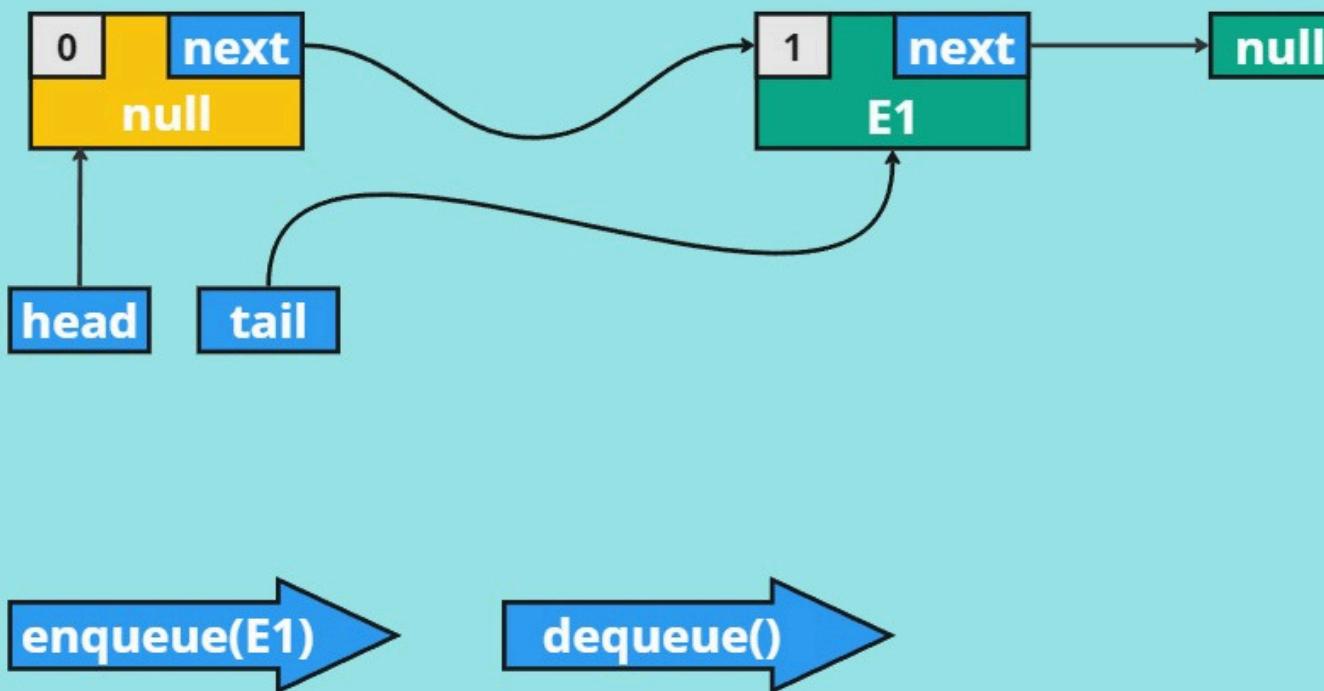
## ConcurrentQueue



```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get(); 0
        final Node<E> previousTail = tail.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == previousTail) { 1
            if (nextHead == null) {
                return empty();
            } else {
                tail.compareAndSet(previousTail, nextHead); 0
            }
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}
```

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get(); 0
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

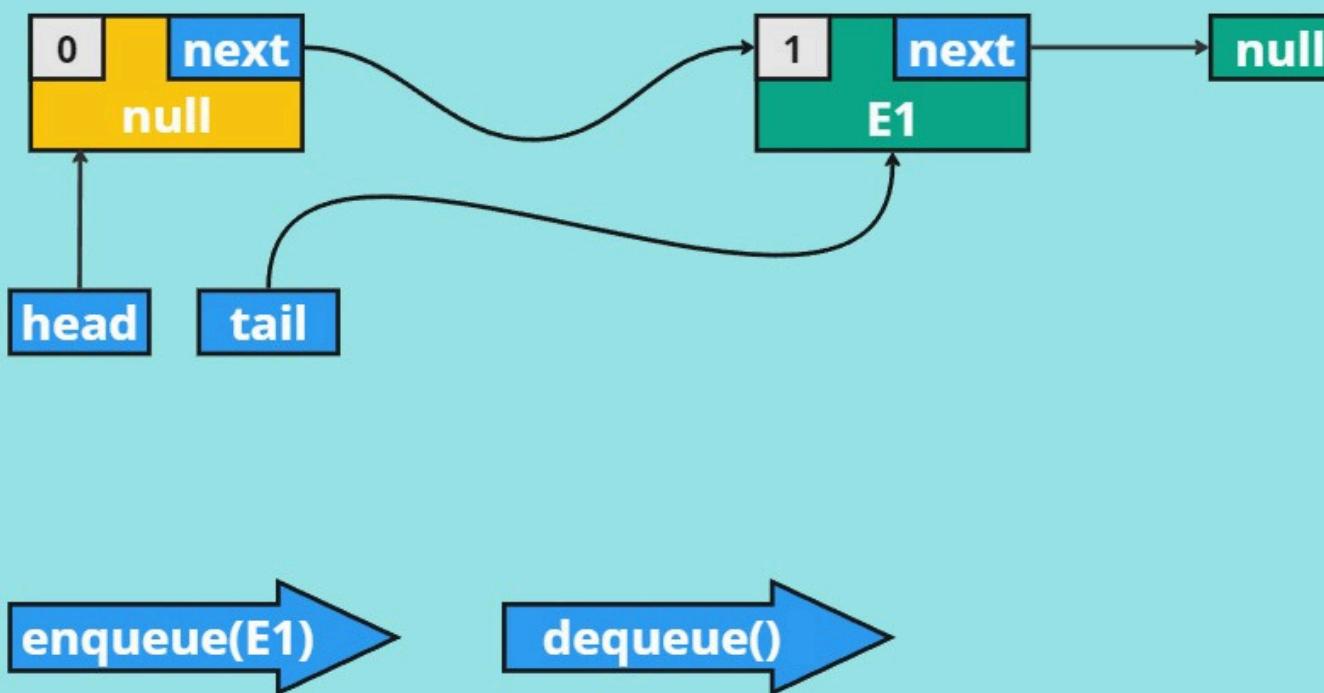
## ConcurrentQueue



```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get(); 0
        final Node<E> previousTail = tail.get();
        final Node<E> nextHead = previousHead.next.get(); 1
        if (previousHead == previousTail) {
            if (nextHead == null) {
                return empty();
            } else {
                tail.compareAndSet(previousTail, nextHead);
            }
        }
        final E element = nextHead.value.get(); E1
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}
```

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get(); 0
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

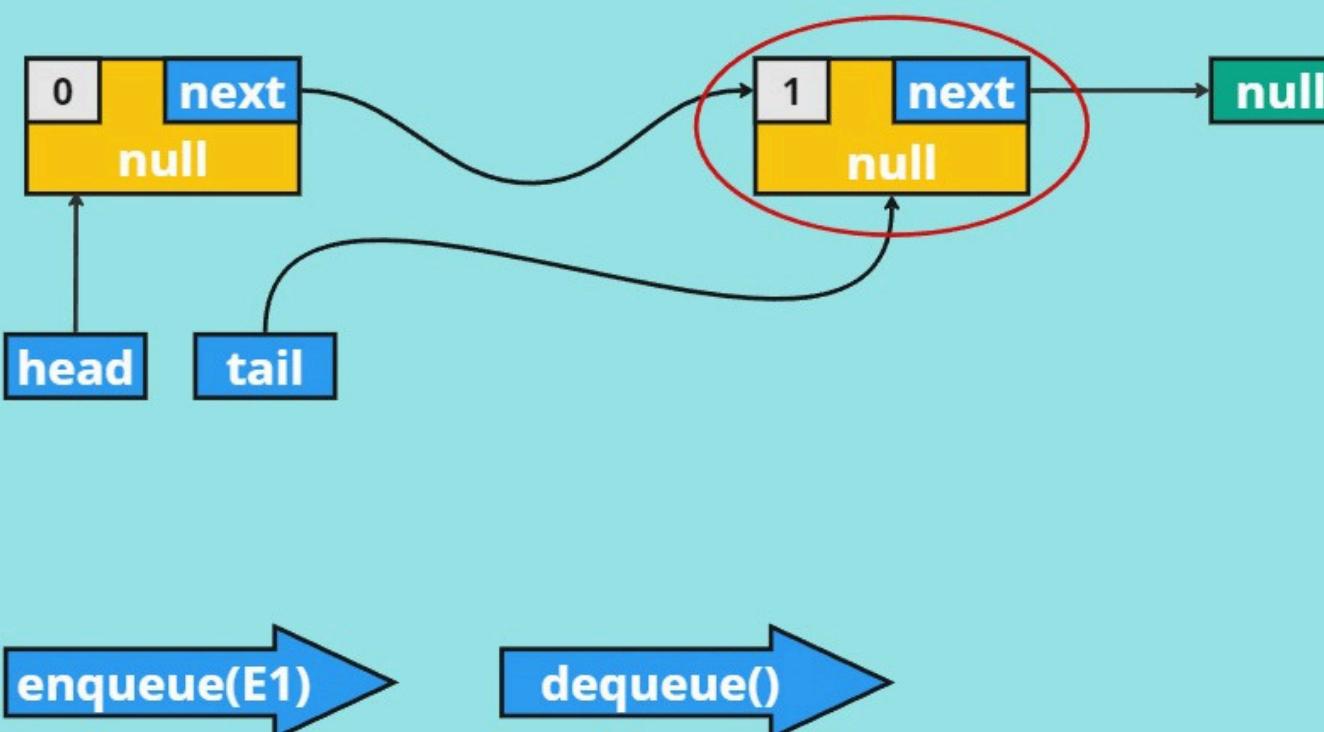
## ConcurrentQueue



```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> previousTail = tail.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == previousTail) {
            if (nextHead == null) {
                return empty();
            } else {
                tail.compareAndSet(previousTail, nextHead);
            }
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}
```

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

## ConcurrentQueue



```

public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get(); 0
        final Node<E> previousTail = tail.get();
        final Node<E> nextHead = previousHead.next.get(); 1
        if (previousHead == previousTail) {
            if (nextHead == null) {
                return empty();
            } else {
                tail.compareAndSet(previousTail, nextHead);
            }
        }
        final E element = nextHead.value.get(); E1
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

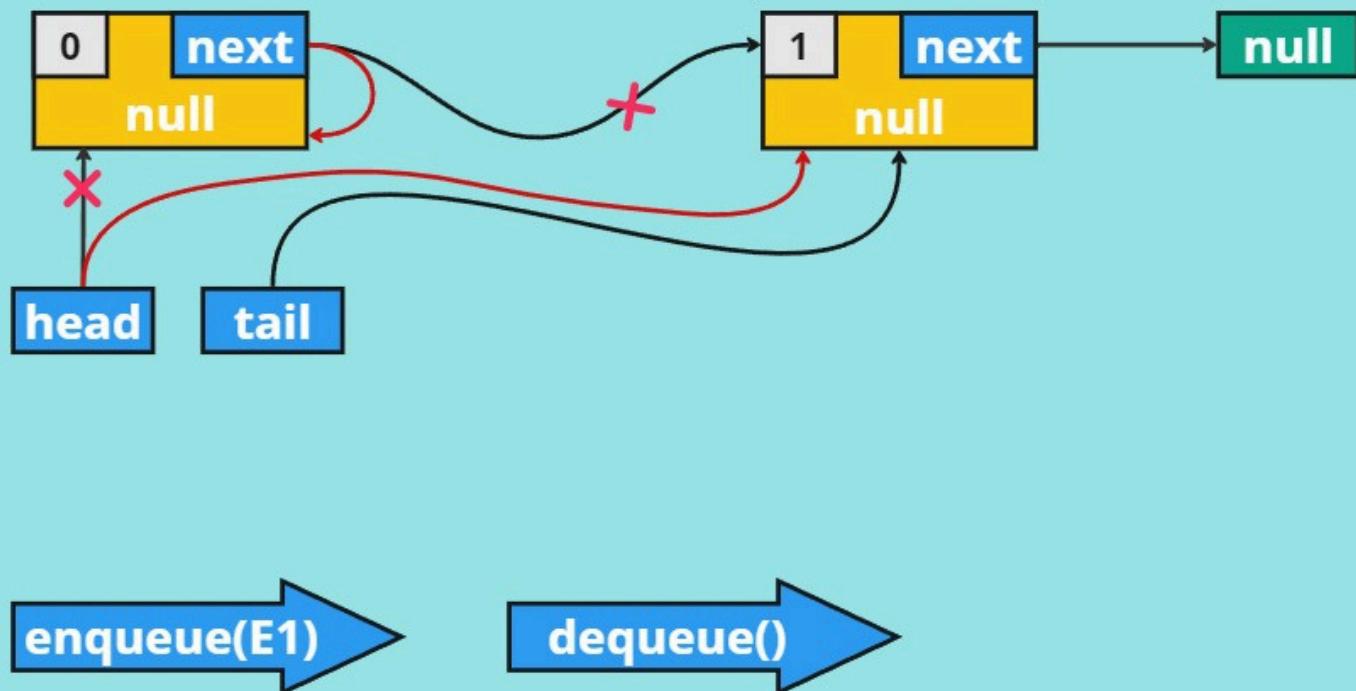
```

```

public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get(); 0
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}

```

## ConcurrentQueue



enqueue(E1) → dequeue()

```

public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get(); 0
        final Node<E> previousTail = tail.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == previousTail) { 1
            if (nextHead == null) {
                return empty();
            } else {
                tail.compareAndSet(previousTail, nextHead);
            }
        }
        final E element = nextHead.value.get(); E1
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

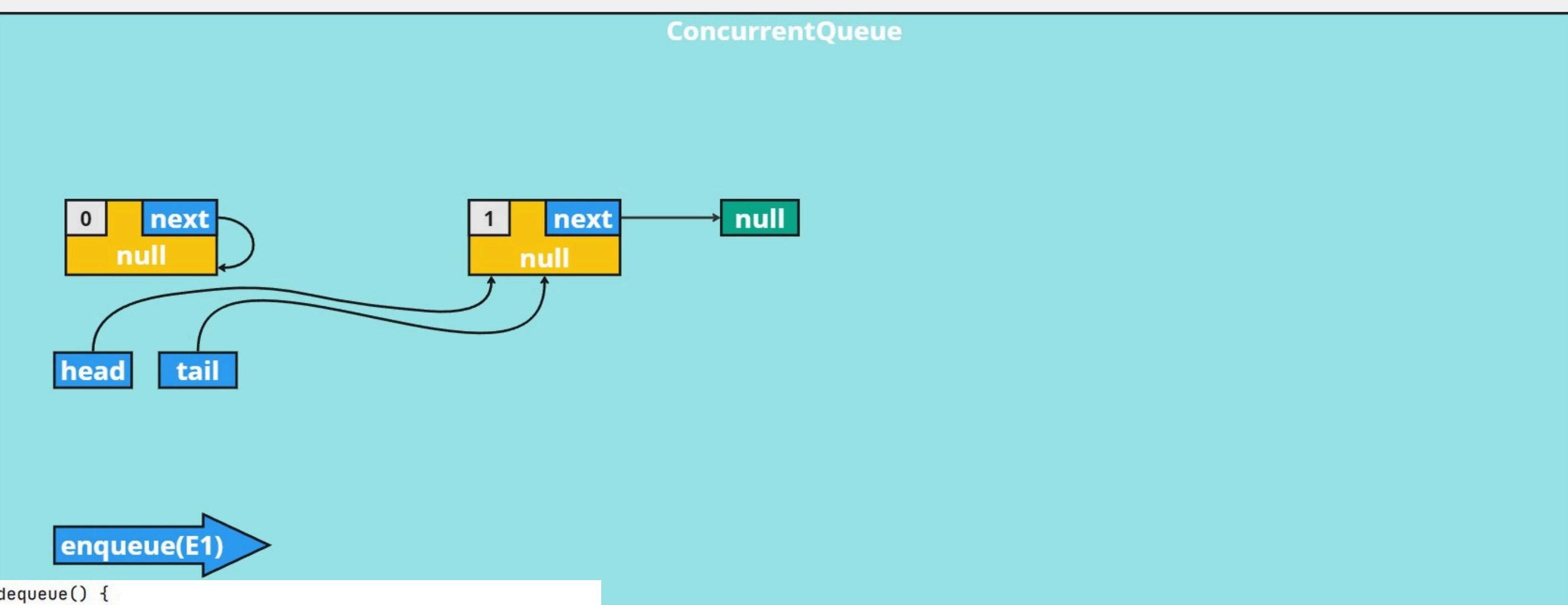
```

```

public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get(); 0
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}

```

## ConcurrentQueue



```

public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get(); 0
        final Node<E> previousTail = tail.get();
        final Node<E> nextHead = previousHead.next.get(); 1
        if (previousHead == previousTail) {
            if (nextHead == null) {
                return empty();
            } else {
                tail.compareAndSet(previousTail, nextHead);
            }
        }
        final E element = nextHead.value.get(); E1
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}

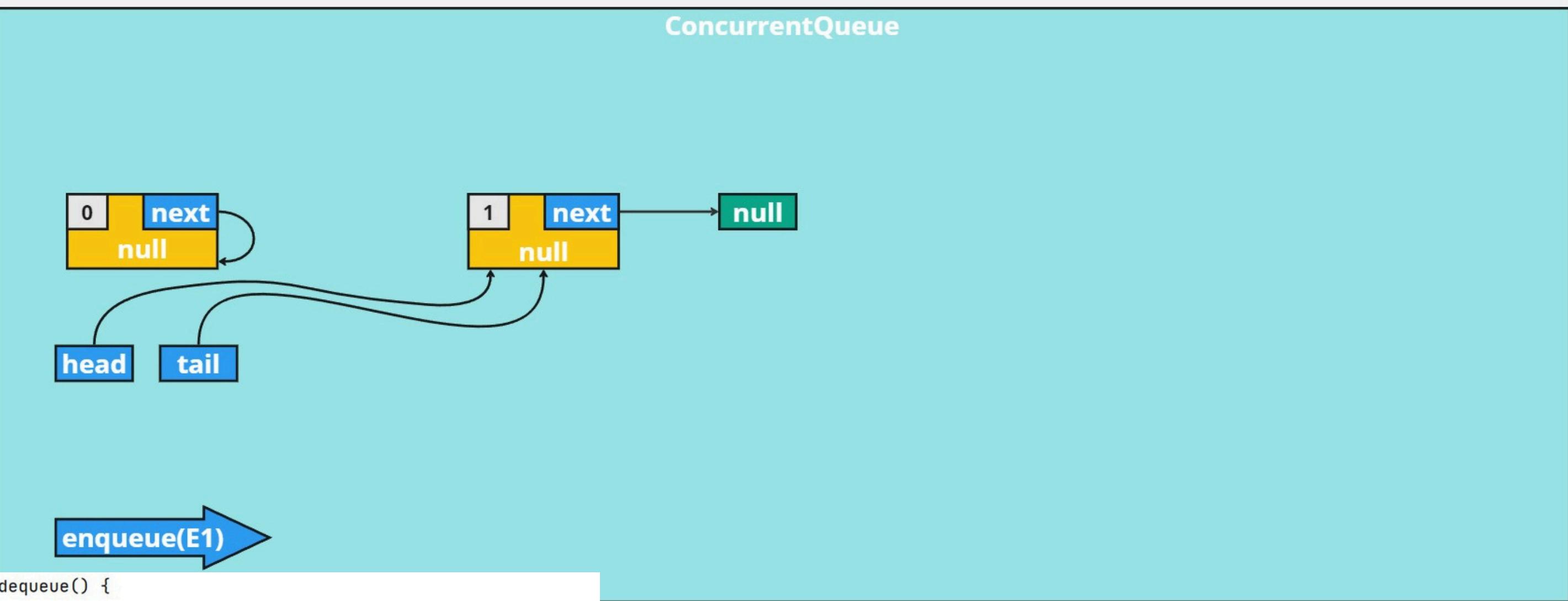
```

```

public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get(); 0
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}

```

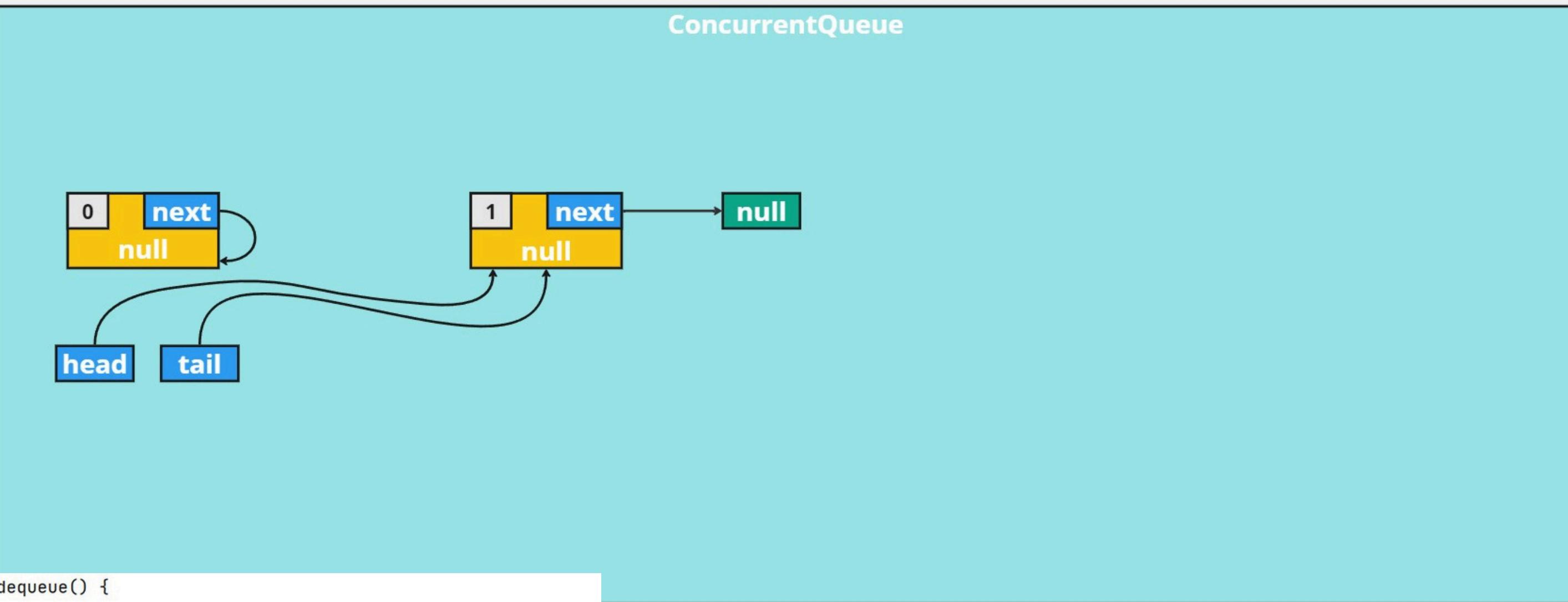
## ConcurrentQueue



```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> previousTail = tail.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == previousTail) {
            if (nextHead == null) {
                return empty();
            } else {
                tail.compareAndSet(previousTail, nextHead);
            }
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}
```

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

## ConcurrentQueue



```
public Optional<E> dequeue() {
    while (true) {
        final Node<E> previousHead = head.get();
        final Node<E> previousTail = tail.get();
        final Node<E> nextHead = previousHead.next.get();
        if (previousHead == previousTail) {
            if (nextHead == null) {
                return empty();
            } else {
                tail.compareAndSet(previousTail, nextHead);
            }
        }
        final E element = nextHead.value.get();
        if (element != null && nextHead.value.compareAndSet(element, null)) {
            updateHead(previousHead, nextHead);
            return of(element);
        } else {
            updateHead(previousHead, nextHead);
        }
    }
}
```

```
public void enqueue(final E element) {
    final Node<E> newNode = new Node<>(element);
    while (true) {
        final Node<E> previousTail = tail.get();
        if (previousTail.next.compareAndSet(null, newNode)) {
            tail.compareAndSet(previousTail, newNode);
            return;
        } else {
            tail.compareAndSet(previousTail, previousTail.next.get());
        }
    }
}
```

**java.util.concurrent.ConcurrentLinkedQueue**