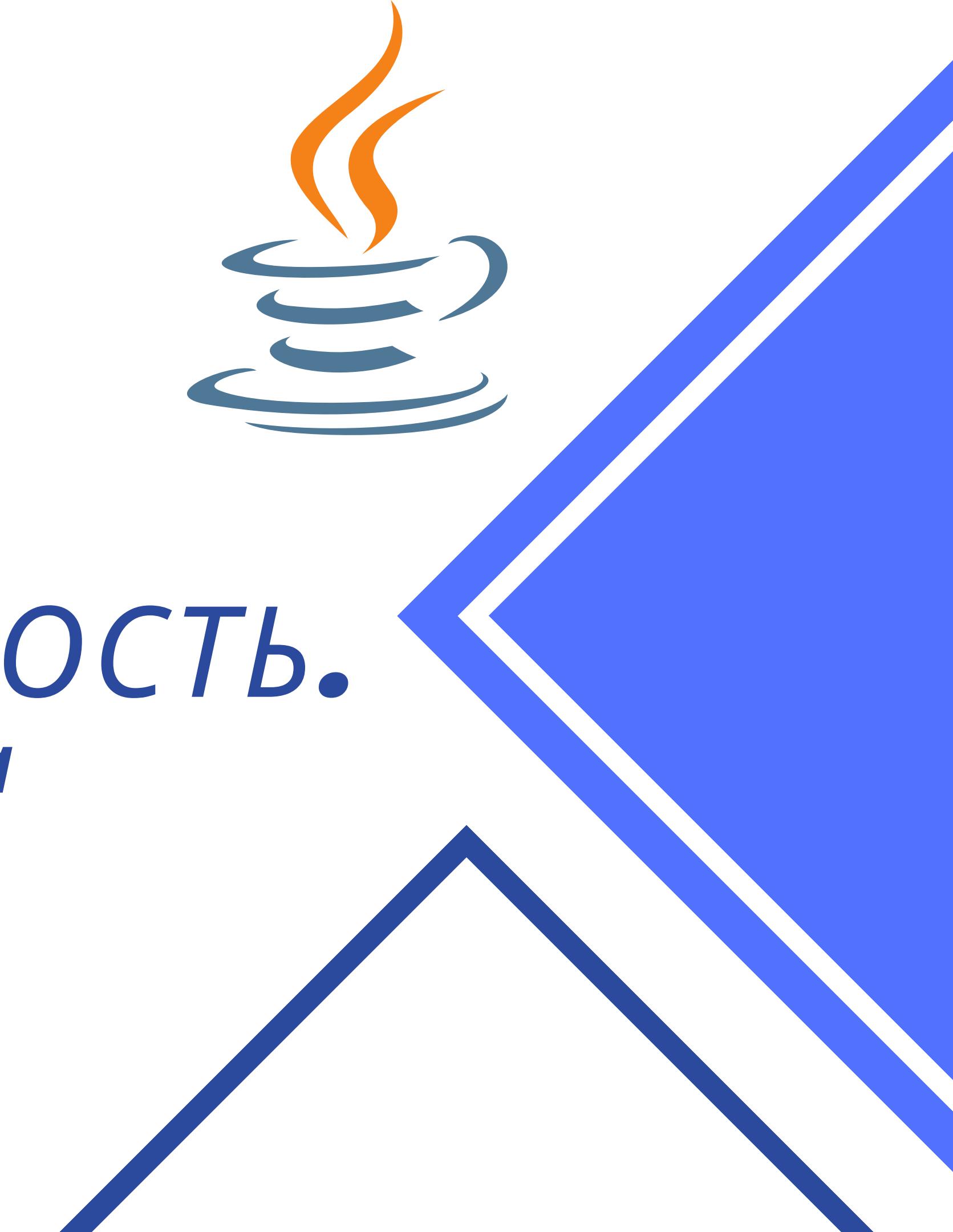
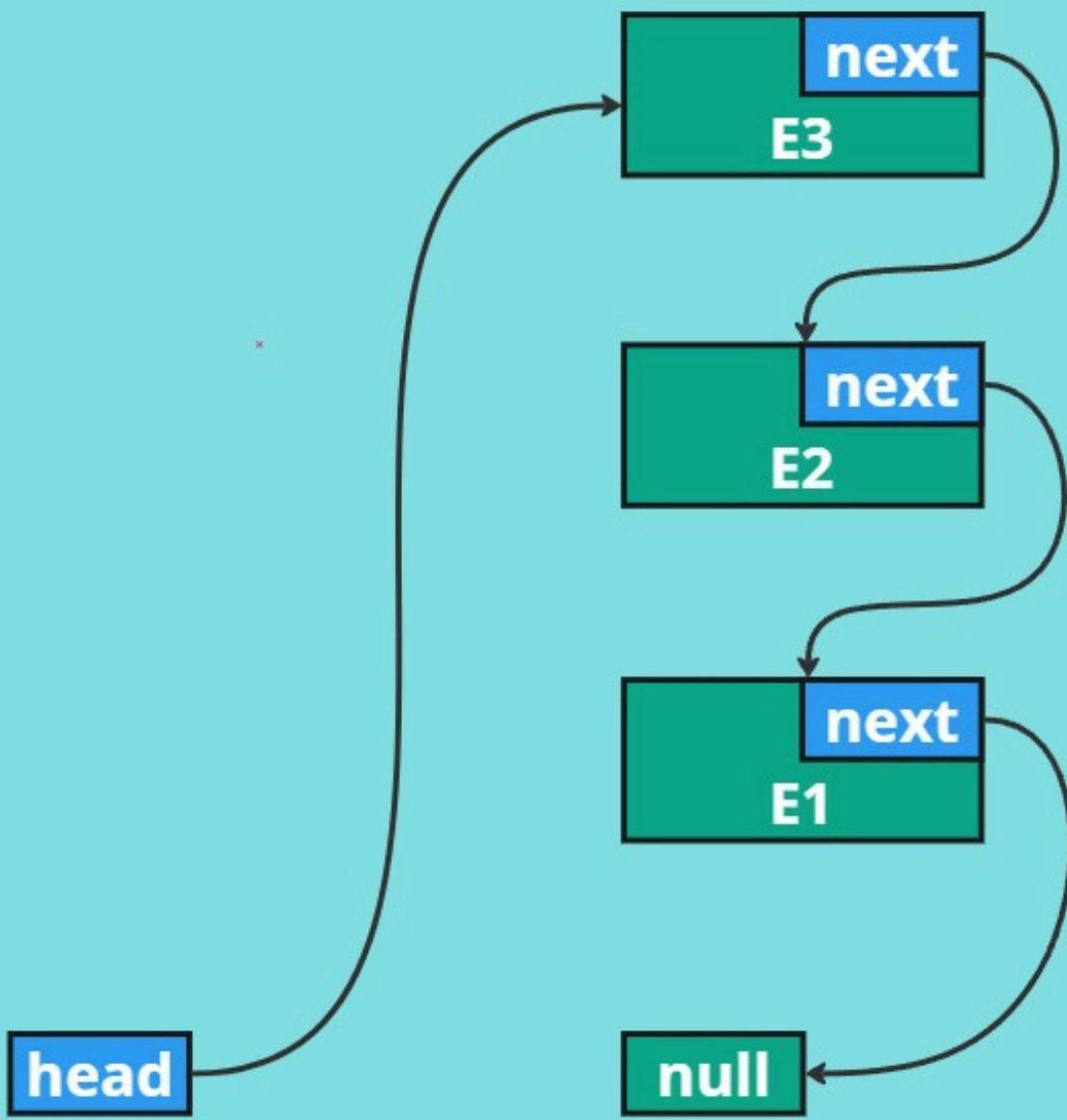


JAVA

многопоточность.
Стек Трайбера



ConcurrentStack



synchronized

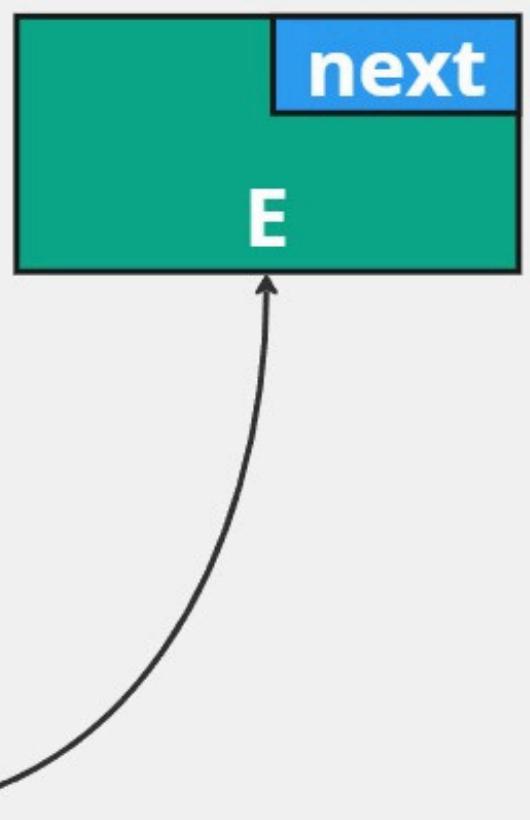


volatile

ReentrantLock

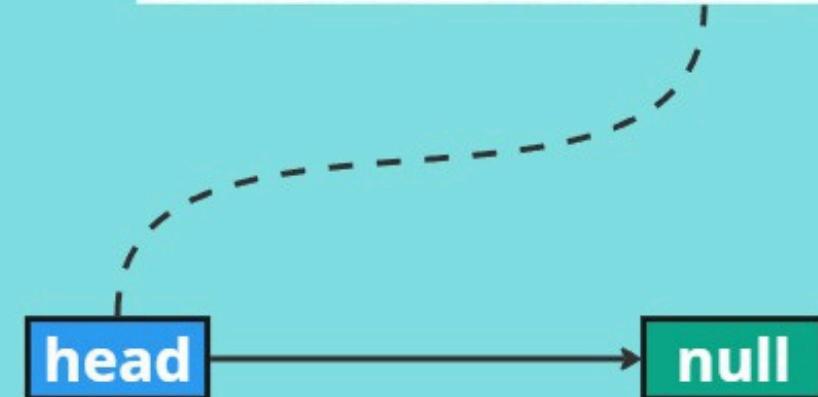
AtomicReference

```
private static final class Node<T> {  
    private final T value;  
    private final Node<T> next;  
  
    public Node(final T value, final Node<T> next) {  
        this.value = value;  
        this.next = next;  
    }  
}
```



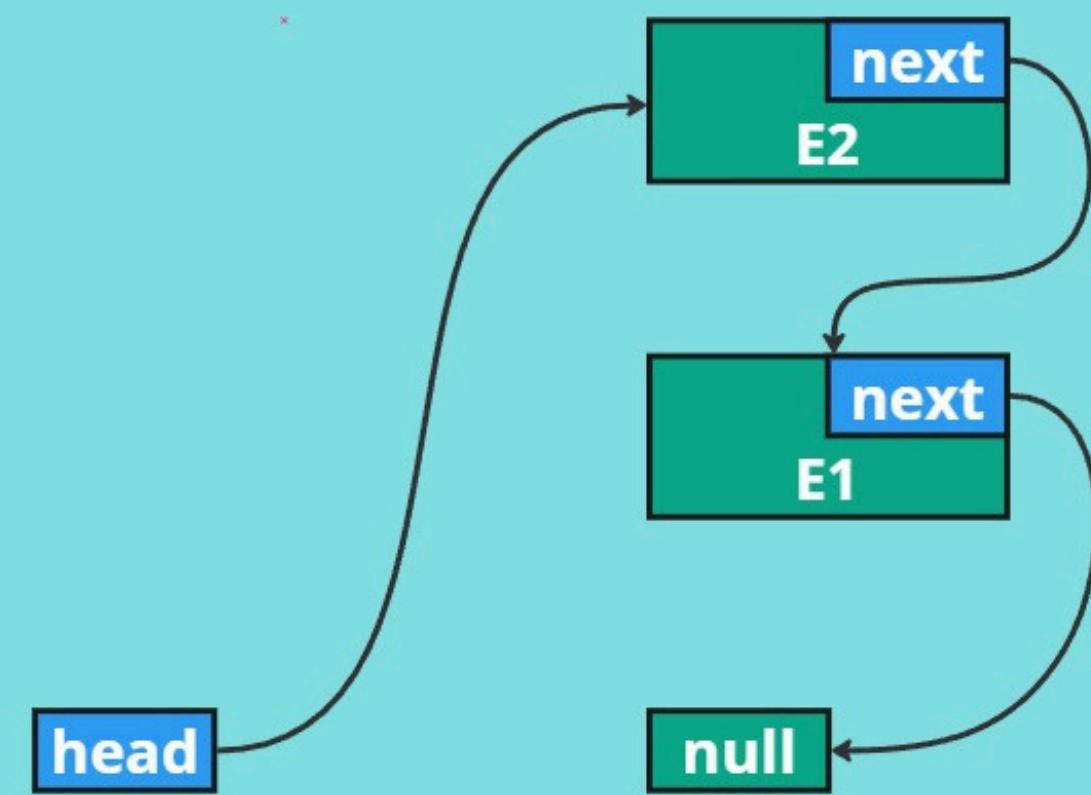
ConcurrentStack

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

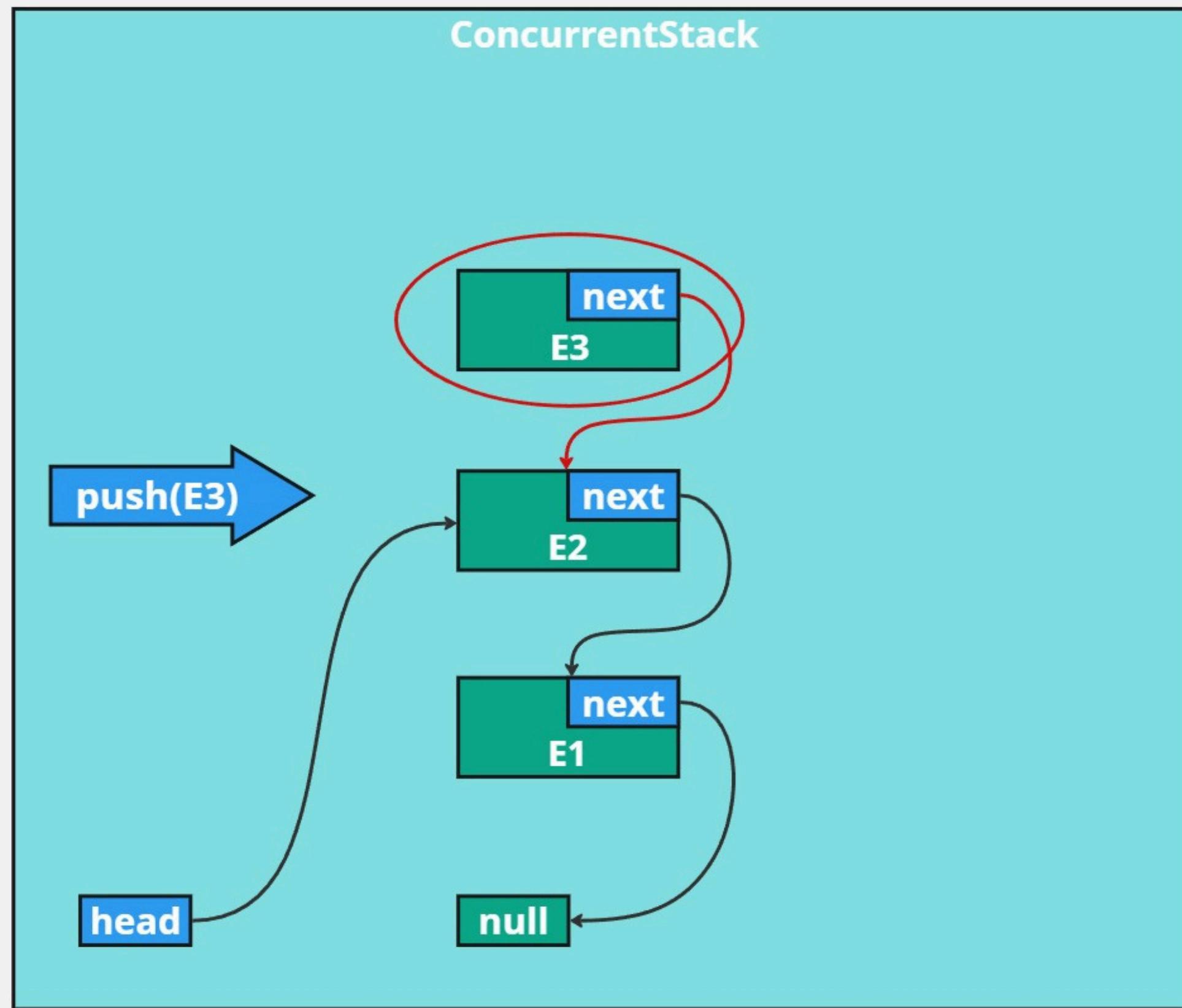


ConcurrentStack

push(E3) →

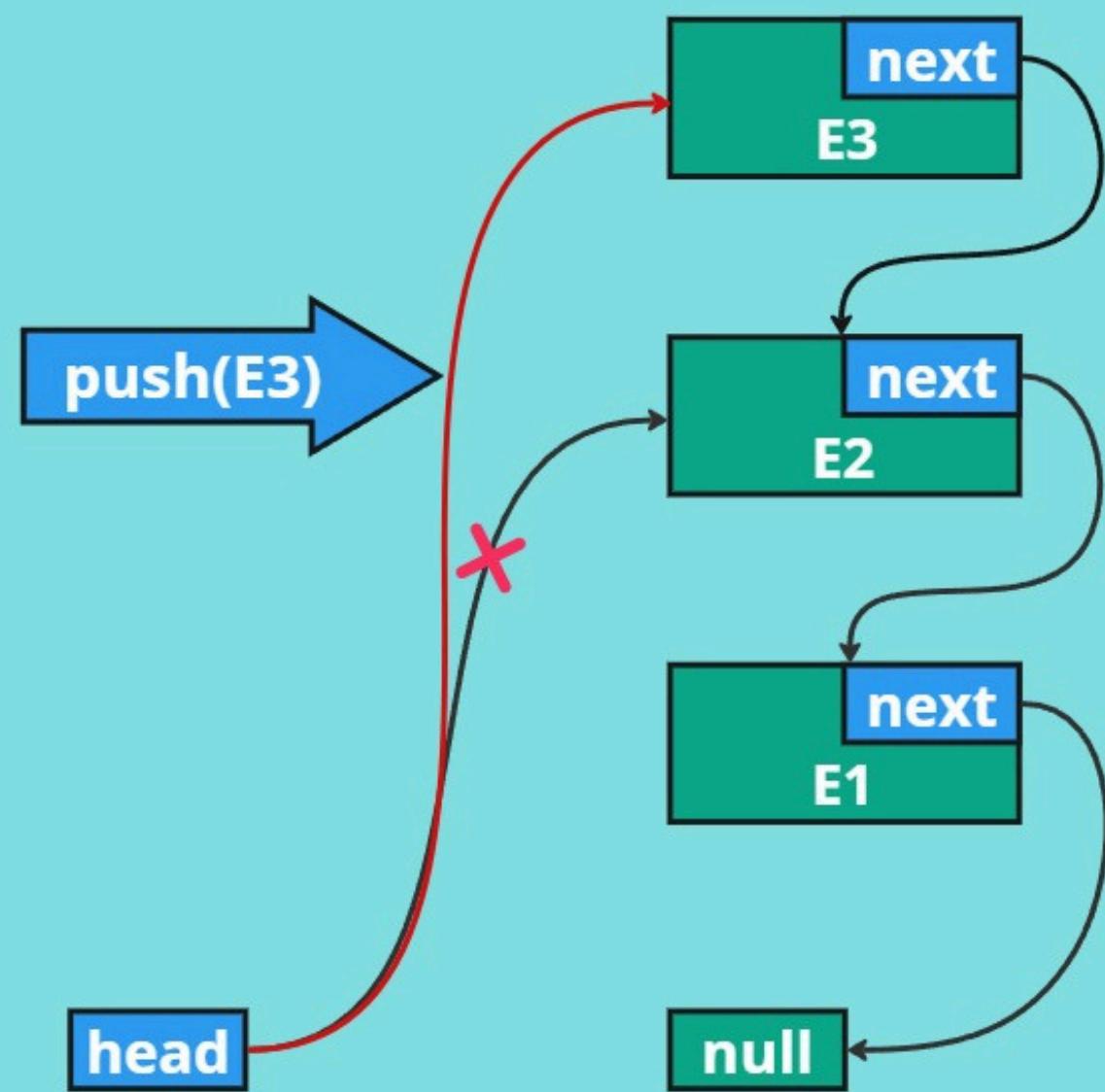


```
public void push(final T element) {  
    //create new node  
    //head should be assigned new node  
}
```



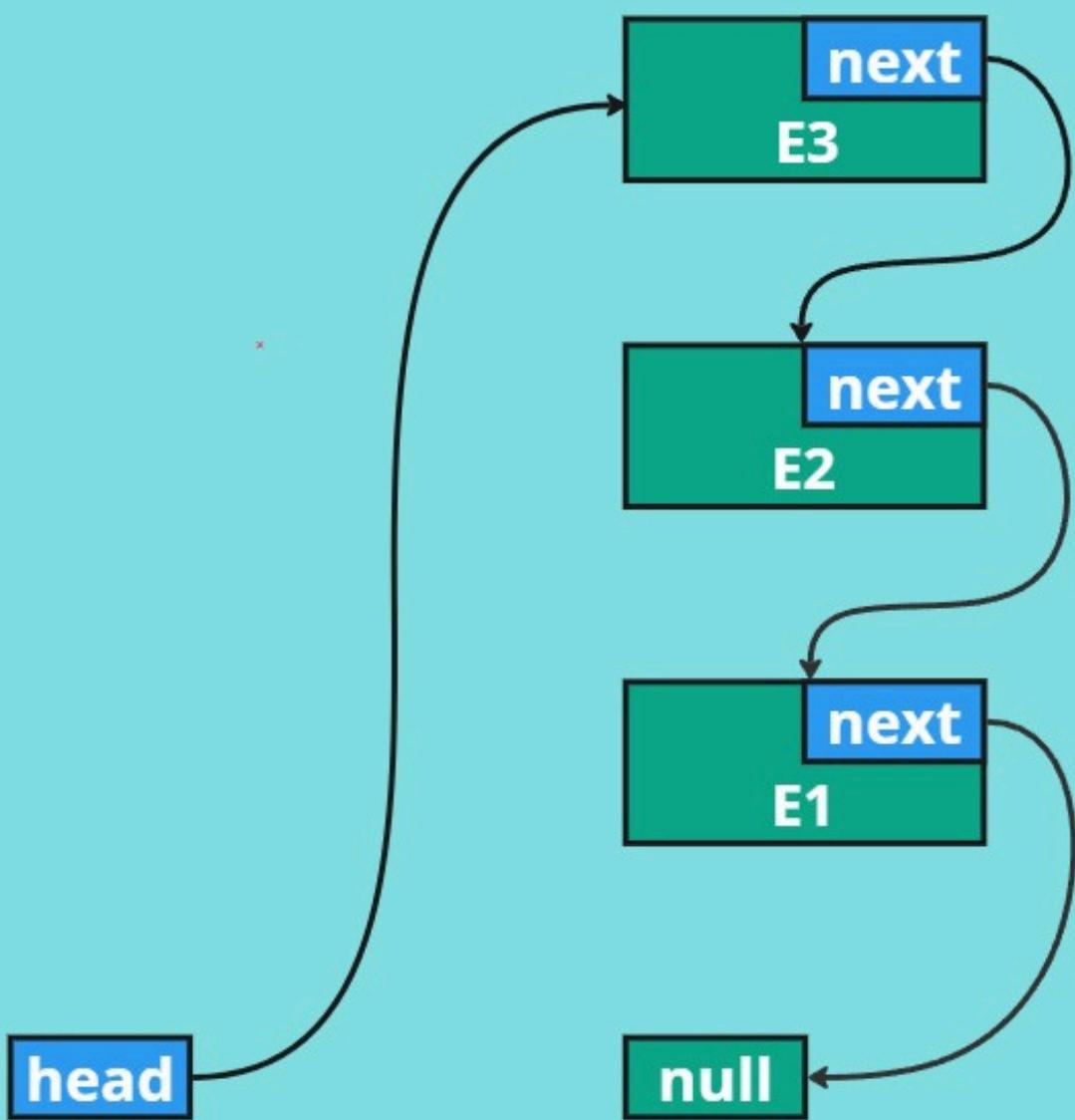
```
public void push(final T element) {  
    //create new node  
    //head should be assigned new node  
}
```

ConcurrentStack



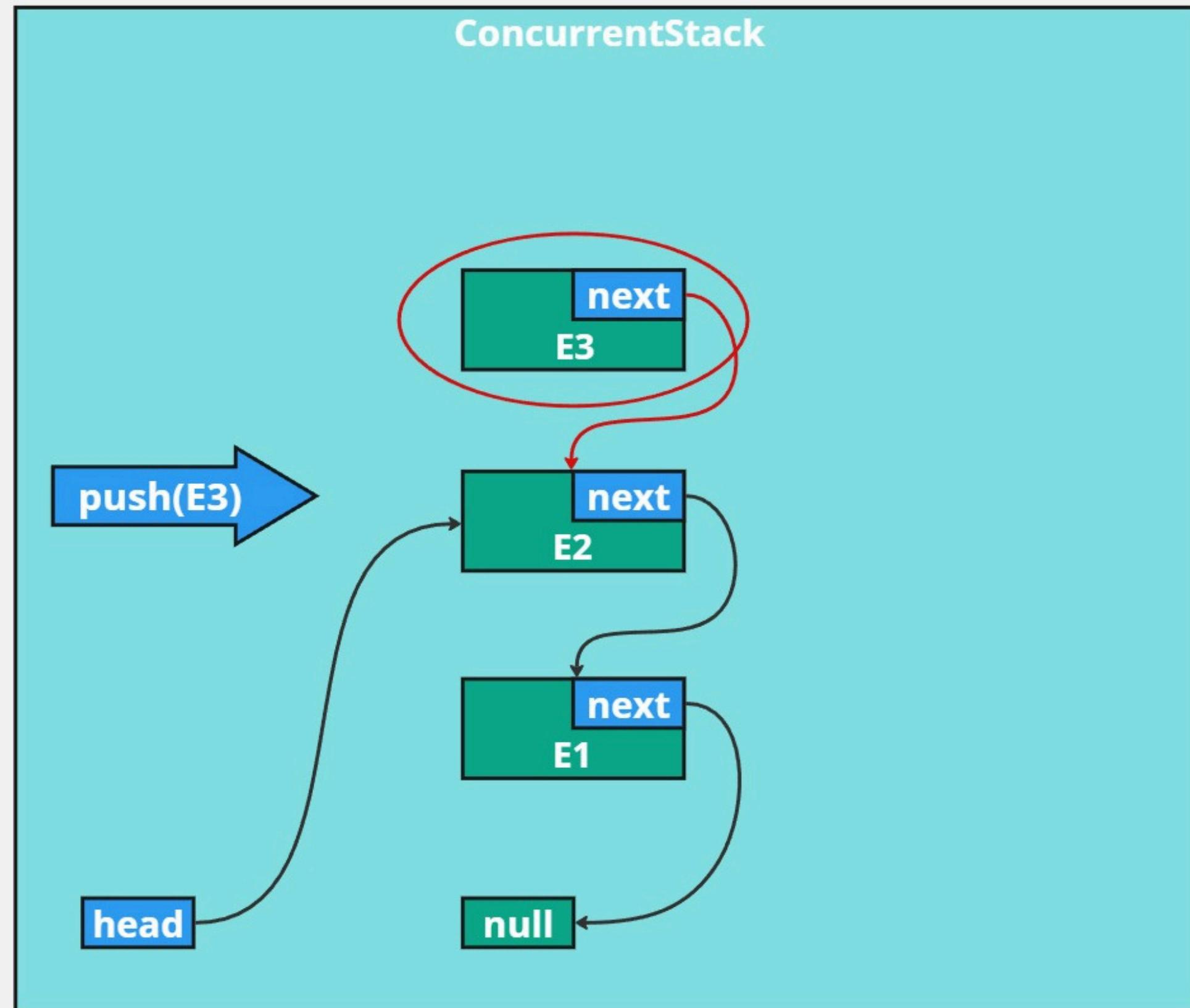
```
public void push(final T element) {  
    //create new node  
    //head should be assigned new node  
}
```

ConcurrentStack



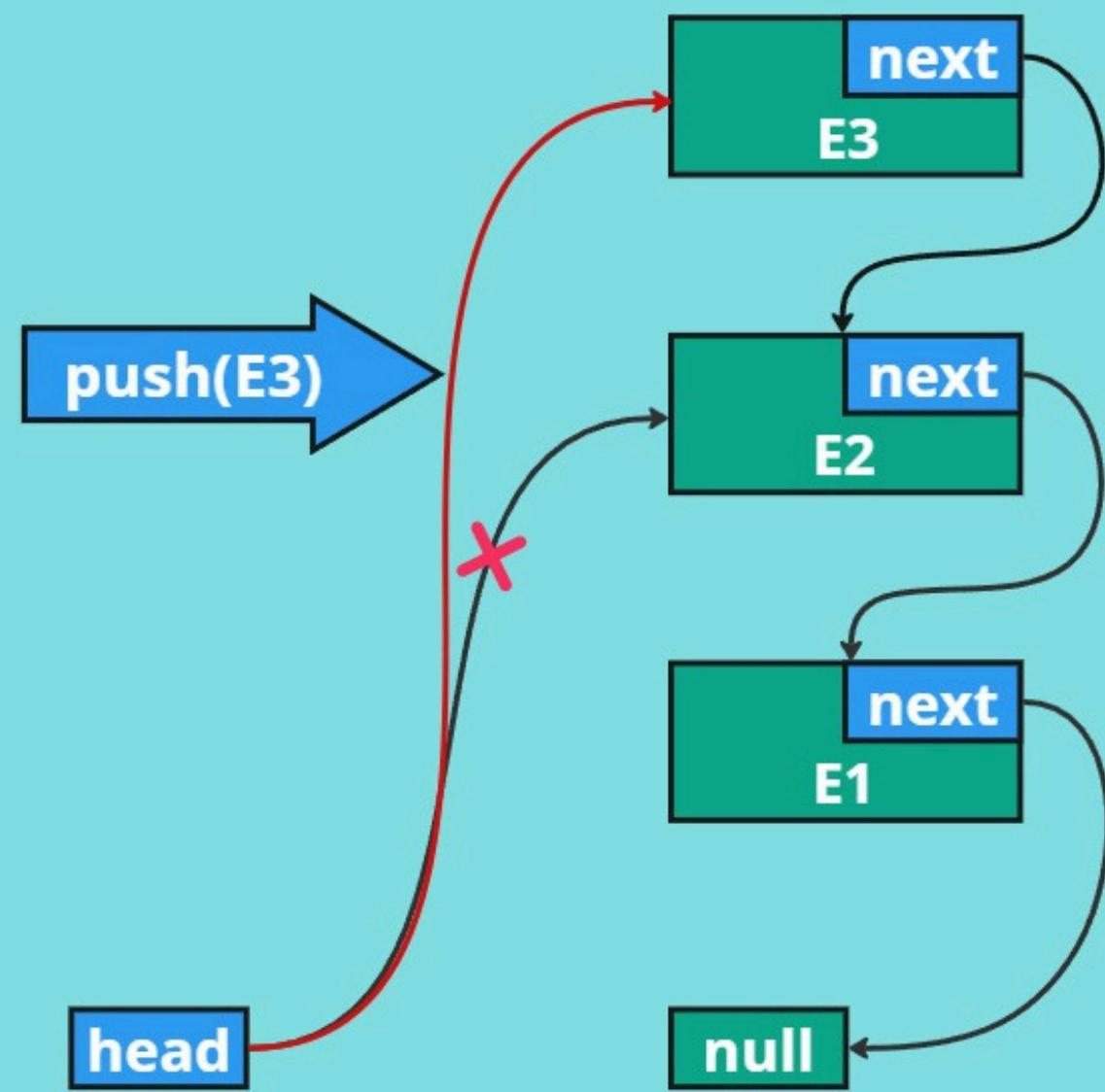
push(E3)

```
public void push(final T element) {  
    //create new node  
    //head should be assigned new node  
}
```



```
public void push(final T element) {  
    head = [new Node<>(element, head)];  
}
```

ConcurrentStack



```
public void push(final T element) {  
    head = new Node<>(element, head);  
}
```

ConcurrentStack

- push(E1) →
- push(E2) →
- push(E3) →



push(E1)

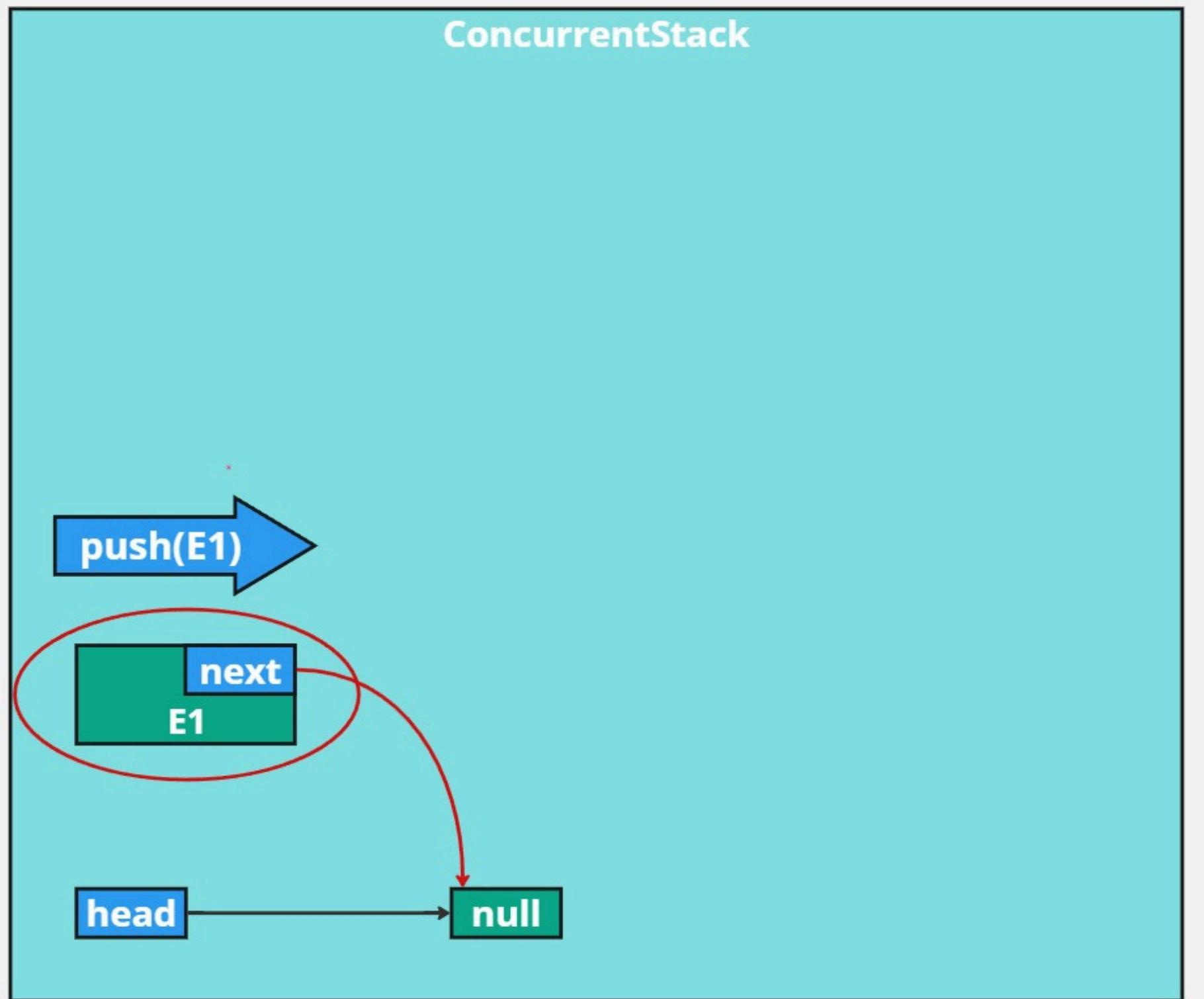
```
public void push(final T element) {  
    head = new Node<>(element, head);  
}
```

push(E2)

```
public void push(final T element) {  
    head = new Node<>(element, head);  
}
```

push(E3)

```
public void push(final T element) {  
    head = new Node<>(element, head);  
}
```



push(E1)

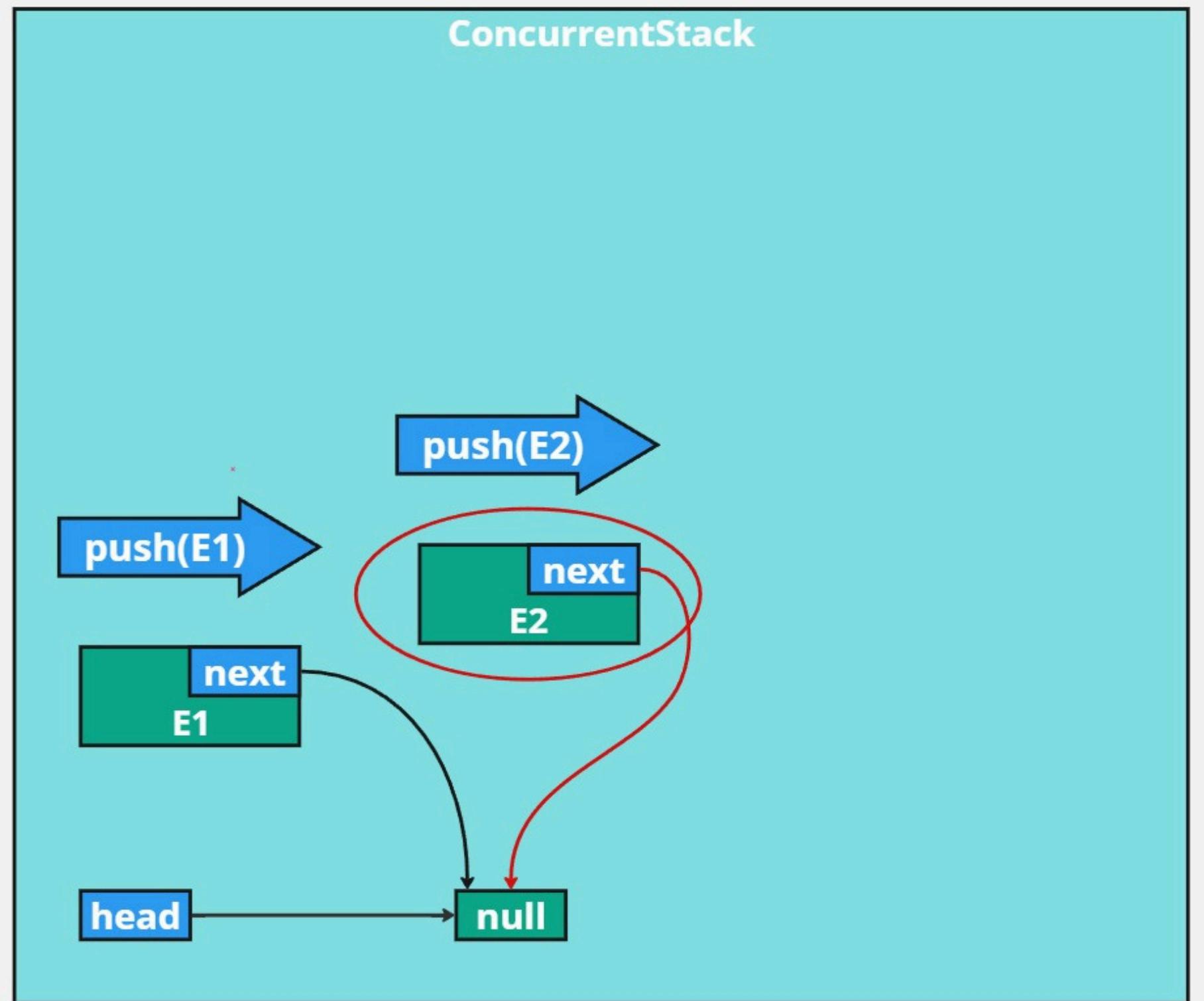
```
public void push(final T element) {  
    head = new Node<>(element, head);  
}
```

push(E2)

```
public void push(final T element) {  
    head = new Node<>(element, head);  
}
```

push(E3)

```
public void push(final T element) {  
    head = new Node<>(element, head);  
}
```



push(E1)

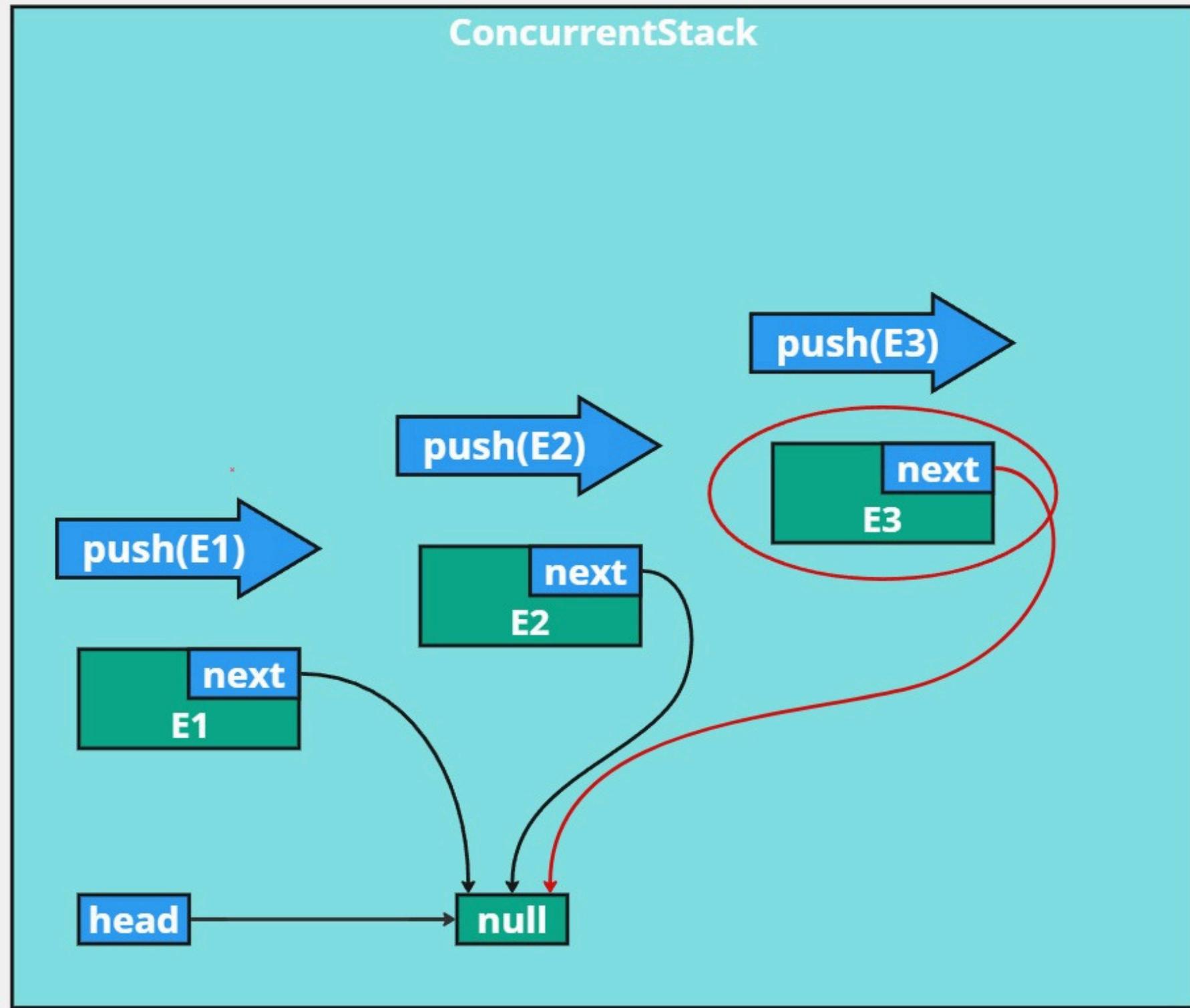
```
public void push(final T element) {  
    head = new Node<>(element, head);  
}
```

push(E2)

```
public void push(final T element) {  
    head = new Node<>(element, head);  
}
```

push(E3)

```
public void push(final T element) {  
    head = new Node<>(element, head);  
}
```



push(E1)

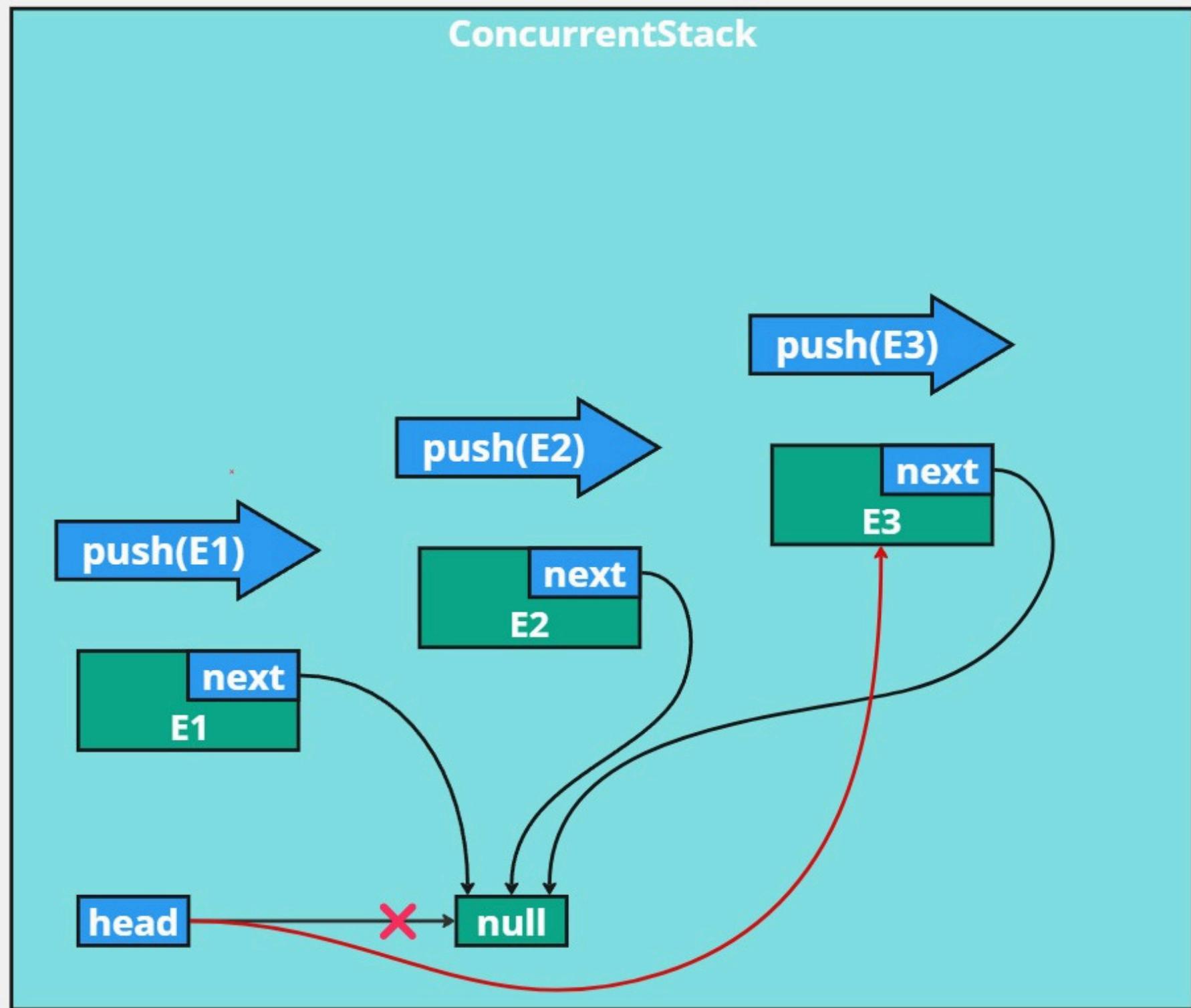
```
public void push(final T element) {
    |head = new Node<>(element, head);
}
```

push(E2)

```
public void push(final T element) {
    |head = new Node<>(element, head);
}
```

push(E3)

```
public void push(final T element) {
    head = new Node<>(element, head);
}
```



push(E1)

```

public void push(final T element) {
    |head = new Node<>(element, head);
}
  
```

push(E2)

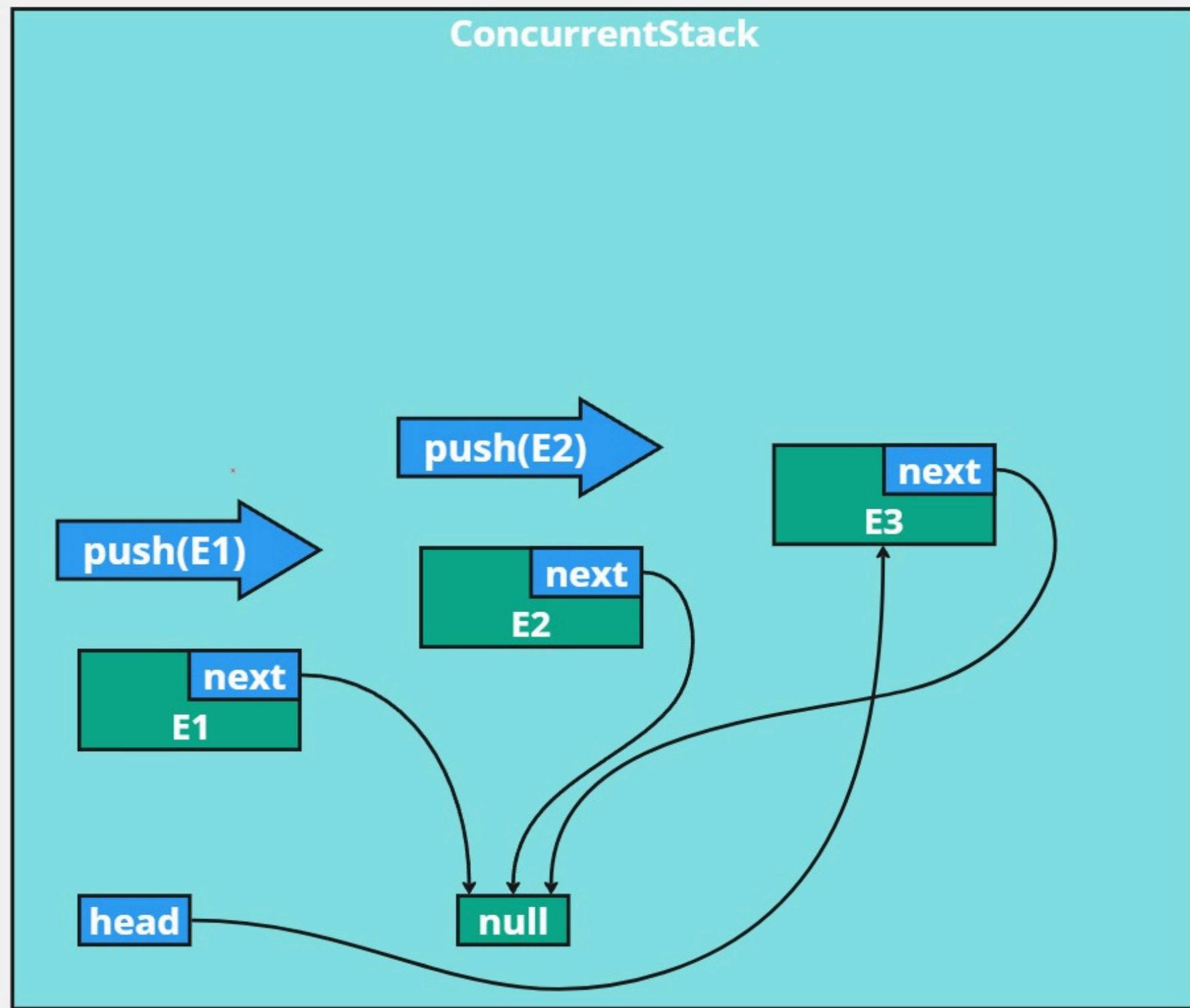
```

public void push(final T element) {
    |head = new Node<>(element, head);
}
  
```

push(E3)

```

public void push(final T element) {
    |head = new Node<>(element, head);
}
  
```



push(E1)

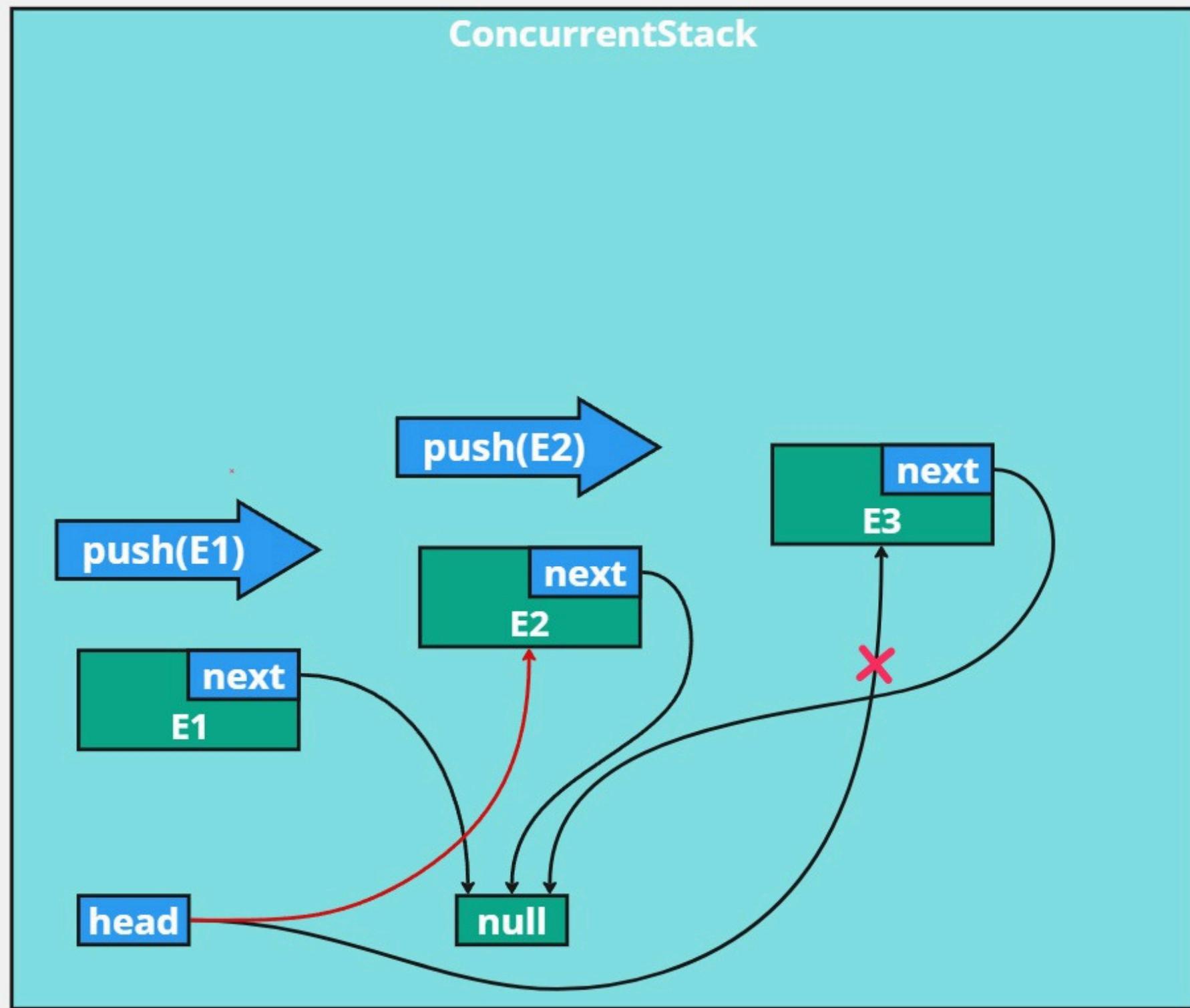
```
public void push(final T element) {  
    head = new Node<>(element, head);  
}
```

push(E2)

```
public void push(final T element) {  
    head = new Node<>(element, head);  
}
```

push(E3)

```
public void push(final T element) {  
    head = new Node<>(element, head);  
}
```



push(E1)

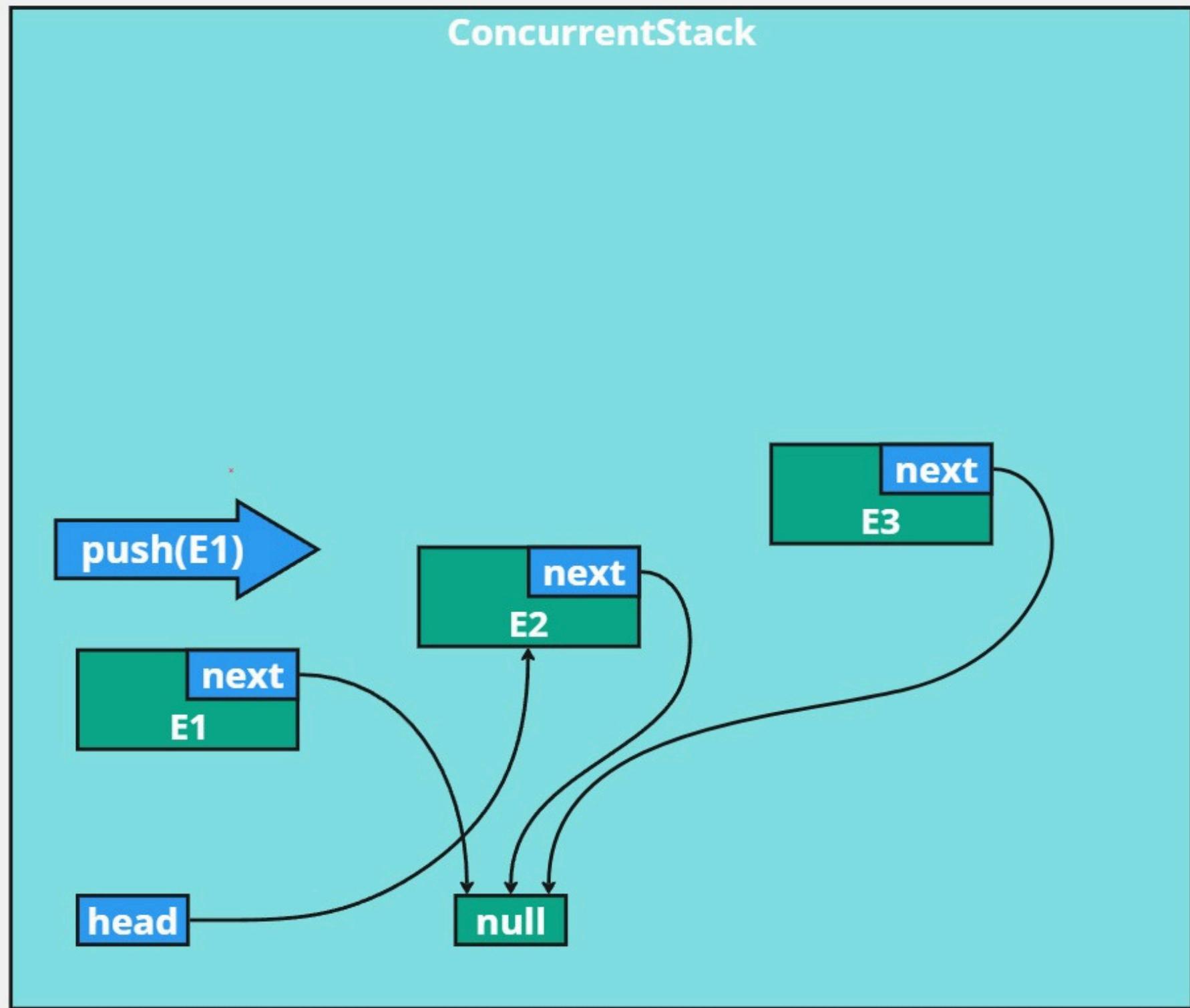
```
public void push(final T element) {
    head = new Node<>(element, head);
}
```

push(E2)

```
public void push(final T element) {
    head = new Node<>(element, head);
}
```

push(E3)

```
public void push(final T element) {
    head = new Node<>(element, head);
}
```



push(E1)

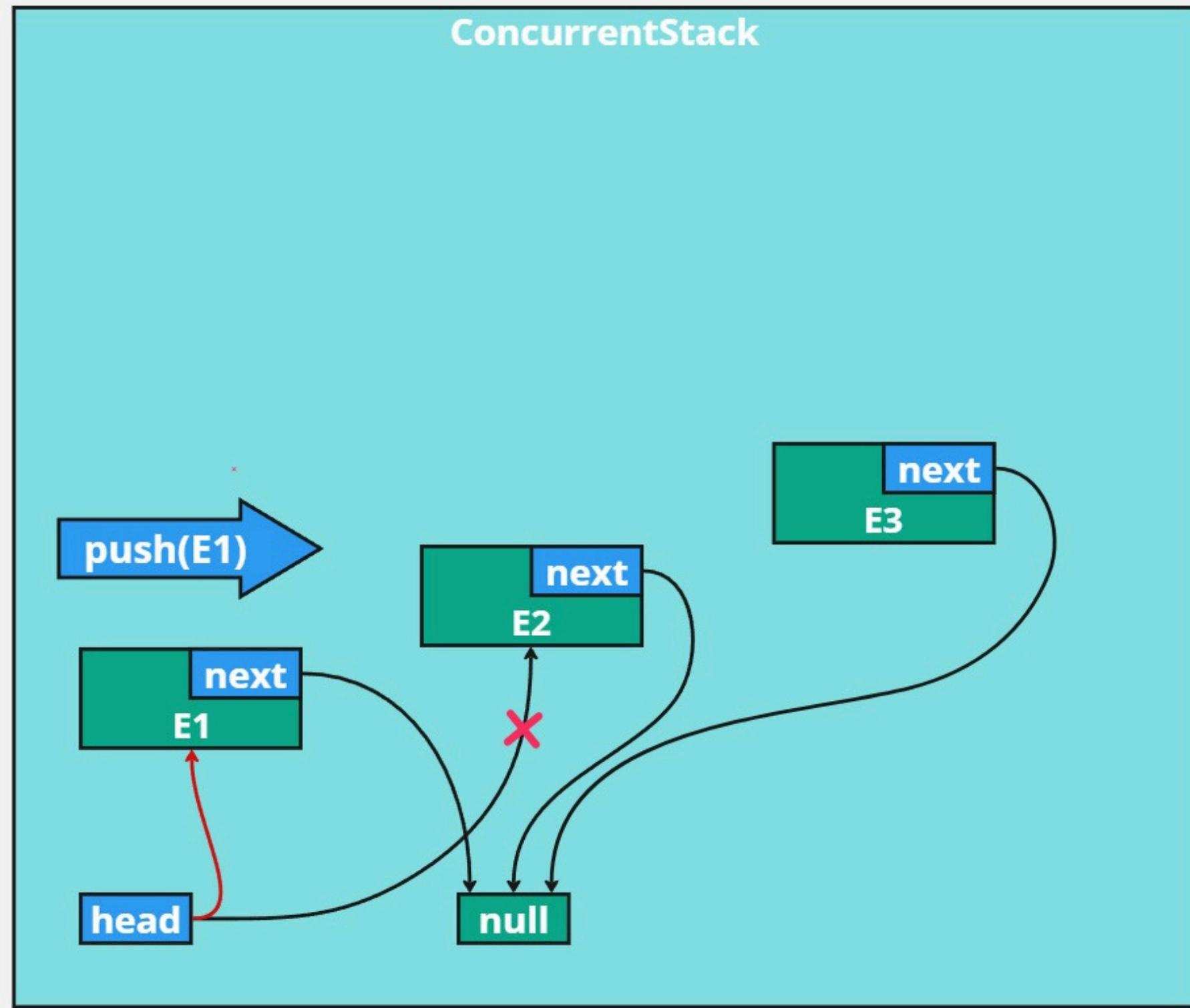
```
public void push(final T element) {  
    head = new Node<>(element, head);  
}
```

push(E2)

```
public void push(final T element) {  
    head = new Node<>(element, head);  
}
```

push(E3)

```
public void push(final T element) {  
    head = new Node<>(element, head);  
}
```



push(E1)

```

public void push(final T element) {
    head = new Node<>(element, head);
}
  
```

push(E2)

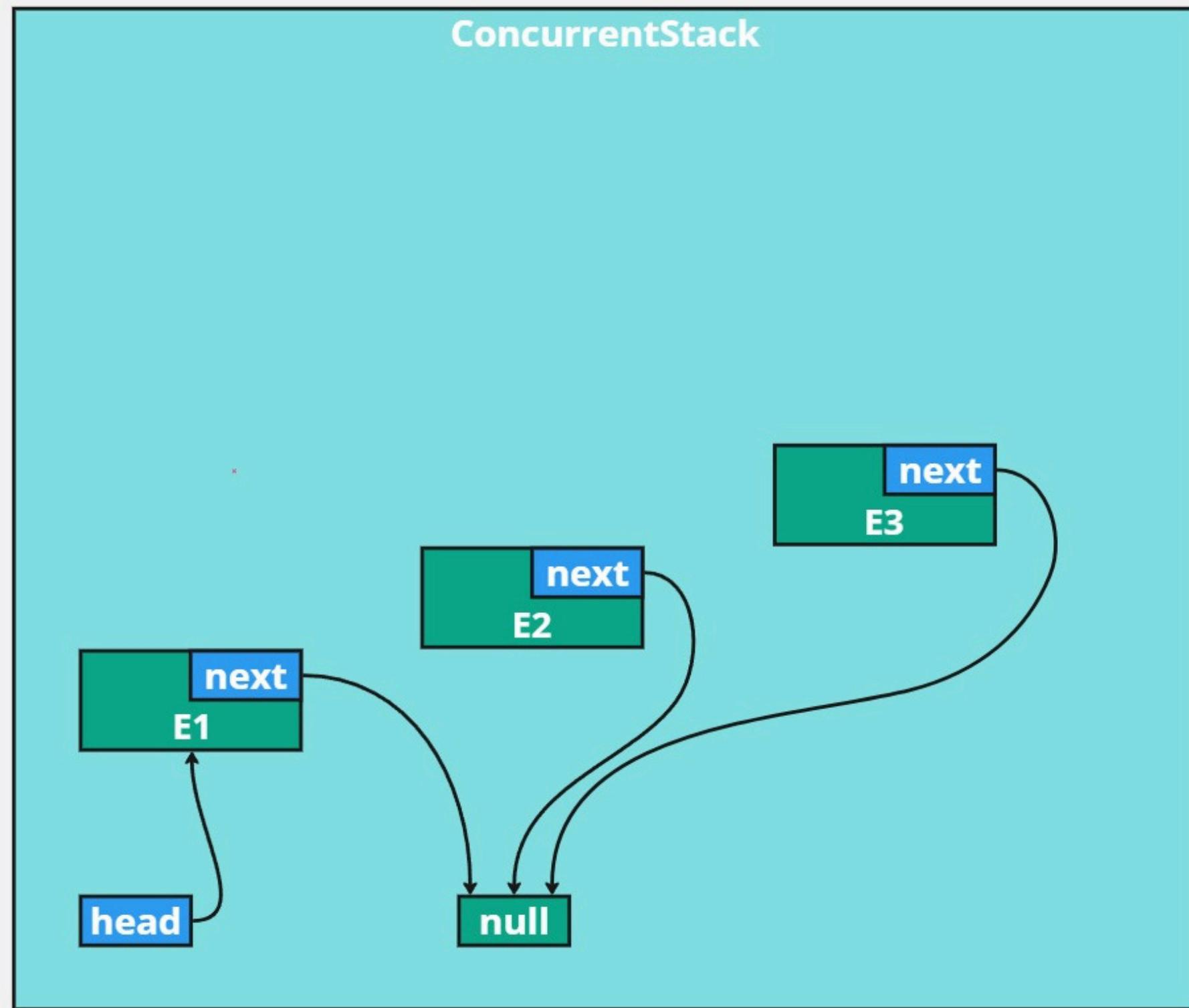
```

public void push(final T element) {
    head = new Node<>(element, head);
}
  
```

push(E3)

```

public void push(final T element) {
    head = new Node<>(element, head);
}
  
```



push(E1)

```
public void push(final T element) {  
    head = new Node<>(element, head);  
}
```

push(E2)

```
public void push(final T element) {  
    head = new Node<>(element, head);  
}
```

push(E3)

```
public void push(final T element) {  
    head = new Node<>(element, head);  
}
```

#1

```
public void push(final T element) {
```

#2

```
--> head = new Node<>(element, head);
```

}

head should be the same

#1

#2

```
public void push(final T element) {  
    head = new Node<T>(element, head);  
}
```



```
public void push(final T element) {  
    while (true) {  
        final Node<T> previousHead = head;  
        final Node<T> newNode = new Node<T>(element, previousHead);  
        if (head == previousHead) {  
            head = newNode;  
            return;  
        }  
    }  
}
```

```
public void push(final T element) {  
    while (true) {  
        final Node<T> previousHead = head;  
        final Node<T> newNode = new Node<>(element, previousHead);  
        if (head == previousHead) {  
            head = newNode;  
            return;  
        }  
    }  
}
```

not atomic

```
private volatile Node<T> head;

public void push(final T element) {
    while (true) {
        final Node<T> previousHead = head;
        final Node<T> newNode = new Node<>(element, previousHead);
        if (head == previousHead) {
            head = newNode;
            return;
        }
    }
}
```

private final AtomicReference<Node<T>> head = new AtomicReference<>();

```
public void push(final T element) {
    while (true) {
        final Node<T> previousHead = head.get();
        final Node<T> newNode = new Node<T>(element, previousHead);
        if (head.compareAndSet(previousHead, newNode)) {
            return;
        }
    }
}
```



ConcurrentStack

push(E3)

push(E2)

push(E1)



push(E1)

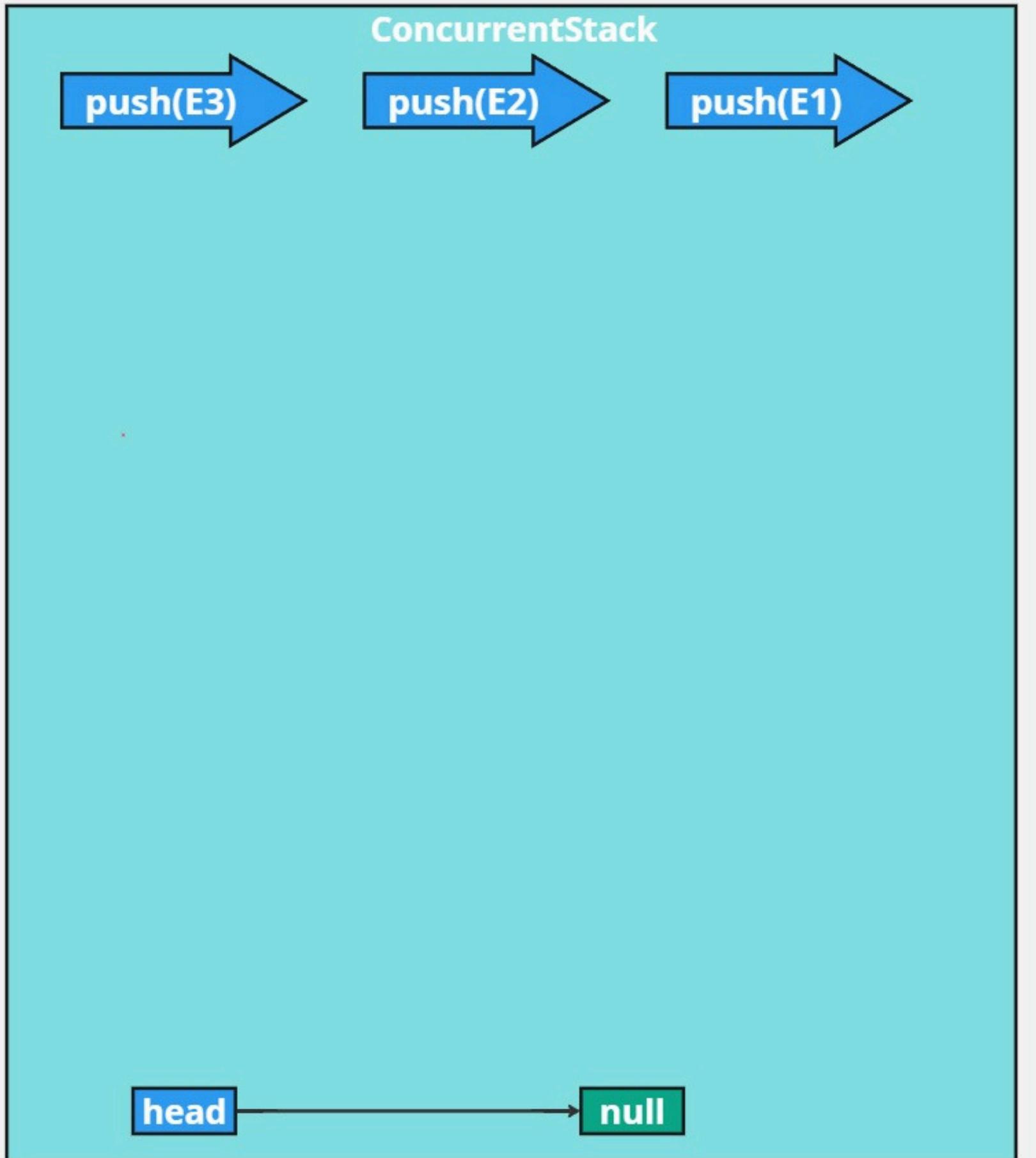
```
public void push(final T element) {  
    while (true) {  
        final Node<T> previousHead = head.get();  
        final Node<T> newNode = new Node<>(element, previousHead);  
        if (head.compareAndSet(previousHead, newNode)) {  
            return;  
        }  
    }  
}
```

push(E2)

```
public void push(final T element) {  
    while (true) {  
        final Node<T> previousHead = head.get();  
        final Node<T> newNode = new Node<>(element, previousHead);  
        if (head.compareAndSet(previousHead, newNode)) {  
            return;  
        }  
    }  
}
```

push(E3)

```
public void push(final T element) {  
    while (true) {  
        final Node<T> previousHead = head.get();  
        final Node<T> newNode = new Node<>(element, previousHead);  
        if (head.compareAndSet(previousHead, newNode)) {  
            return;  
        }  
    }  
}
```



push(E1)

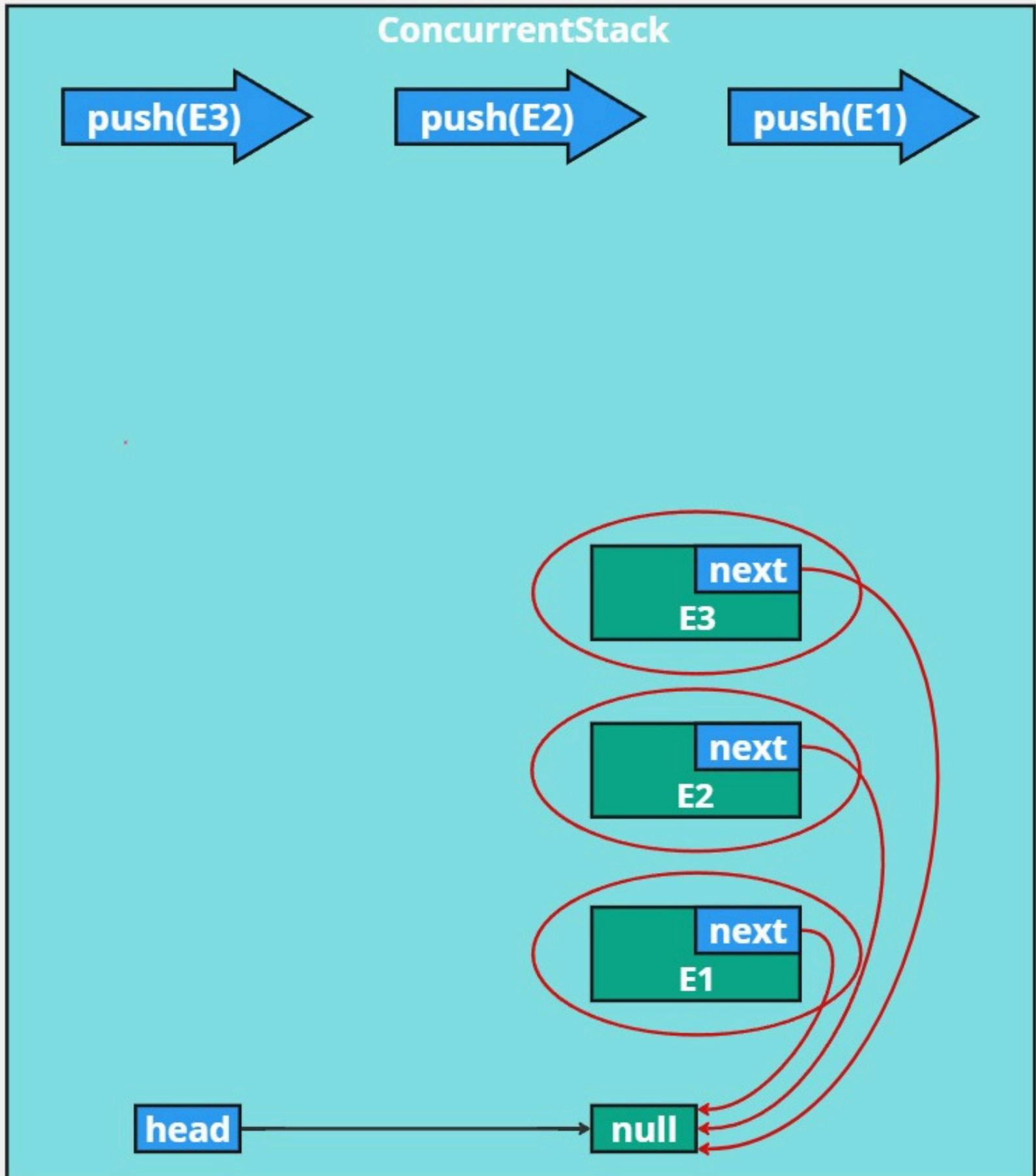
```
public void push(final T element) {
    while (true) {
        final Node<T> previousHead = head.get();
        final Node<T> newNode = new Node<>(element, previousHead);
        if (head.compareAndSet(previousHead, newNode)) {
            return;
        }
    }
}
```

push(E2)

```
public void push(final T element) {
    while (true) {
        final Node<T> previousHead = head.get();
        final Node<T> newNode = new Node<>(element, previousHead);
        if (head.compareAndSet(previousHead, newNode)) {
            return;
        }
    }
}
```

push(E3)

```
public void push(final T element) {
    while (true) {
        final Node<T> previousHead = head.get();
        final Node<T> newNode = new Node<>(element, previousHead);
        if (head.compareAndSet(previousHead, newNode)) {
            return;
        }
    }
}
```



push(E1)

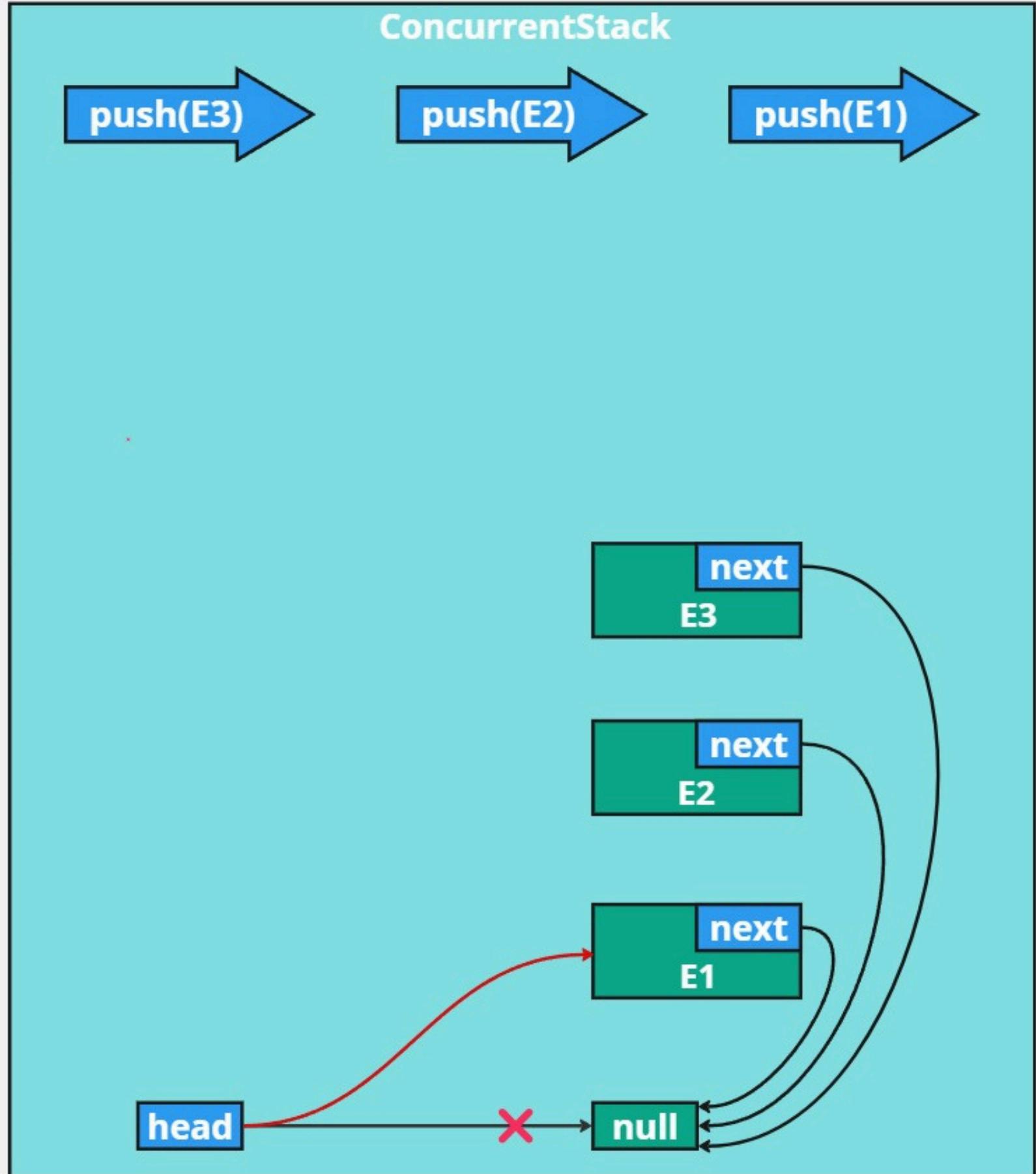
```
public void push(final T element) {
    while (true) { , ----- null
        final Node<T> previousHead = head.get();
        final Node<T> newNode = new Node<>(element, previousHead);
        if (head.compareAndSet(previousHead, newNode)) {
            return;
        }
    }
}
```

push(E2)

```
public void push(final T element) {
    while (true) { , ----- null
        final Node<T> previousHead = head.get();
        final Node<T> newNode = new Node<>(element, previousHead);
        if (head.compareAndSet(previousHead, newNode)) {
            return;
        }
    }
}
```

push(E3)

```
public void push(final T element) {
    while (true) { , ----- null
        final Node<T> previousHead = head.get();
        final Node<T> newNode = new Node<>(element, previousHead);
        if (head.compareAndSet(previousHead, newNode)) {
            return;
        }
    }
}
```



push(E1)

```

public void push(final T element) {
    while (true) { ,----- null
        final Node<T> previousHead = head.get();
        final Node<T> newNode = new Node<>(element, previousHead);
        if (head.compareAndSet(previousHead, newNode)) {
            return;
        }
    }
}

```

push(E2)

```

public void push(final T element) {
    while (true) { ,----- null
        final Node<T> previousHead = head.get();
        final Node<T> newNode = new Node<>(element, previousHead);
        if (head.compareAndSet(previousHead, newNode)) {
            return;
        }
    }
}

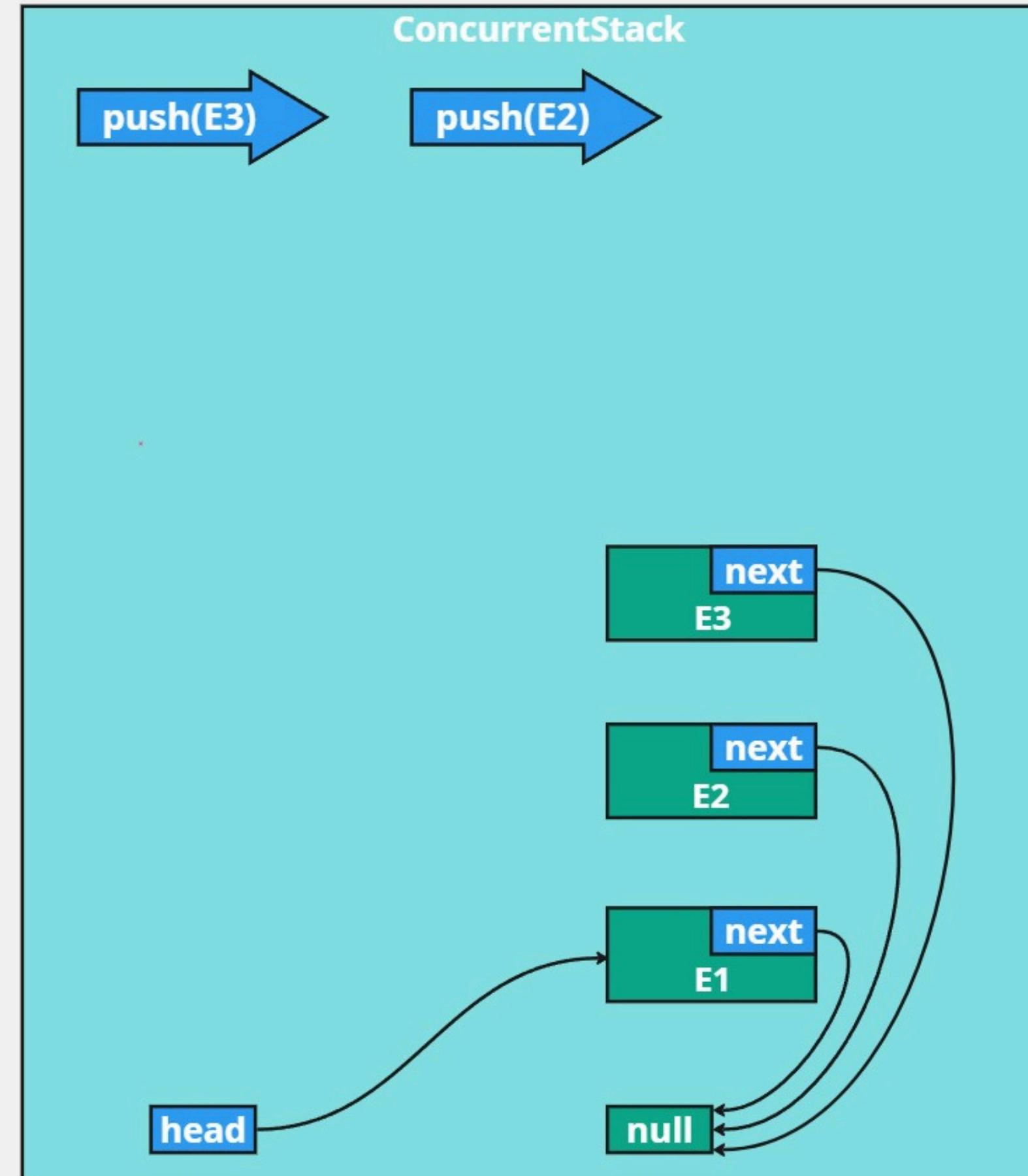
```

push(E3)

```

public void push(final T element) {
    while (true) { ,----- null
        final Node<T> previousHead = head.get();
        final Node<T> newNode = new Node<>(element, previousHead);
        if (head.compareAndSet(previousHead, newNode)) {
            return;
        }
    }
}

```



push(E1)

```

public void push(final T element) {
    while (true) { ,----- null
        final Node<T> previousHead = head.get();
        final Node<T> newNode = new Node<>(element, previousHead);
        if (head.compareAndSet(previousHead, newNode)) {
            return;
        }
    }
}
  
```

push(E2)

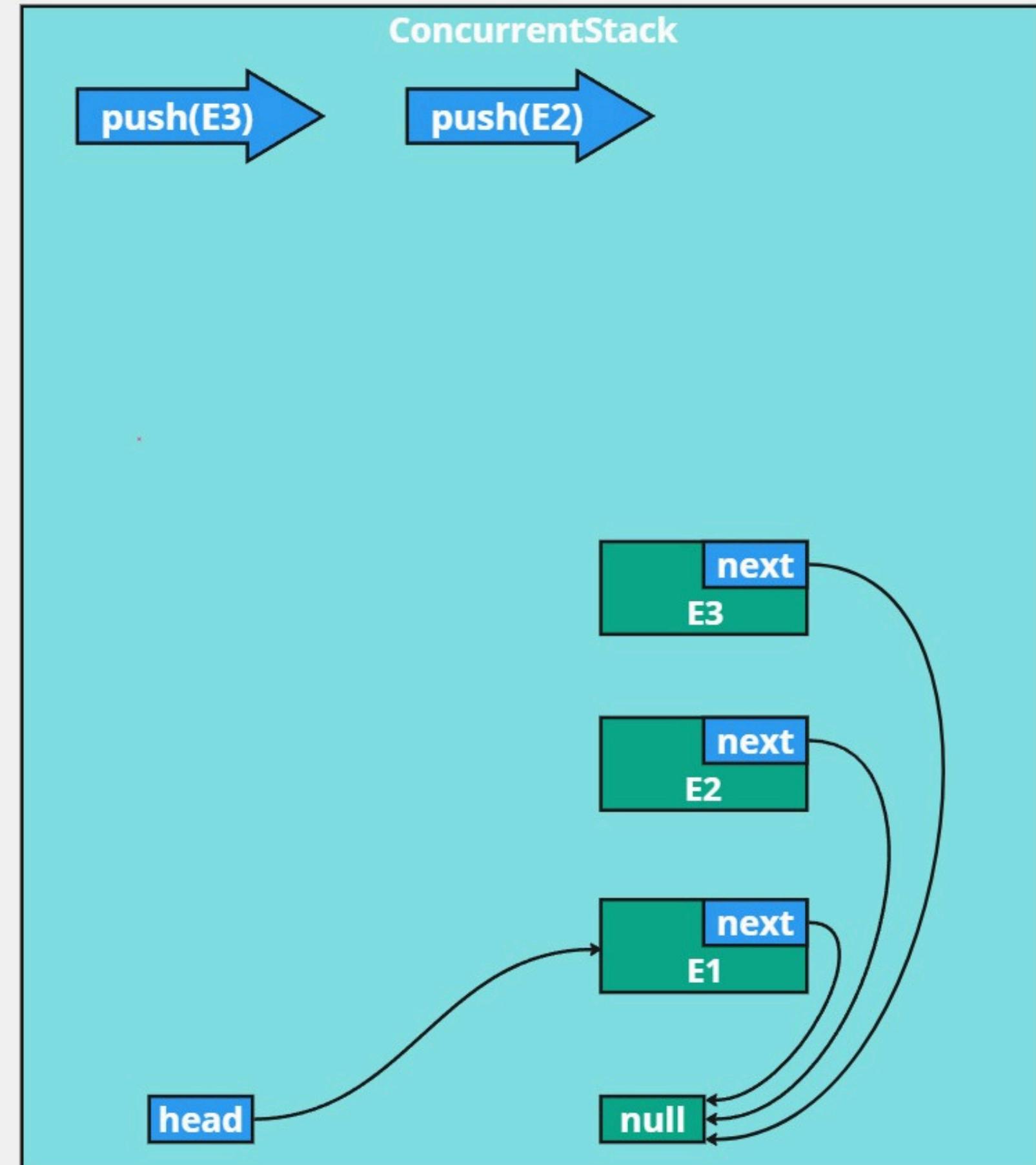
```

public void push(final T element) {
    while (true) { ,----- null
        final Node<T> previousHead = head.get();
        final Node<T> newNode = new Node<>(element, previousHead);
        if (head.compareAndSet(previousHead, newNode)) {
            return;
        }
    }
}
  
```

push(E3)

```

public void push(final T element) {
    while (true) { ,----- null
        final Node<T> previousHead = head.get();
        final Node<T> newNode = new Node<>(element, previousHead);
        if (head.compareAndSet(previousHead, newNode)) {
            return;
        }
    }
}
  
```



push(E1)

```

public void push(final T element) {
    while (true) {
        final Node<T> previousHead = head.get();
        final Node<T> newNode = new Node<>(element, previousHead);
        if (head.compareAndSet(previousHead, newNode)) {
            return;
        }
    }
}
  
```

push(E2)

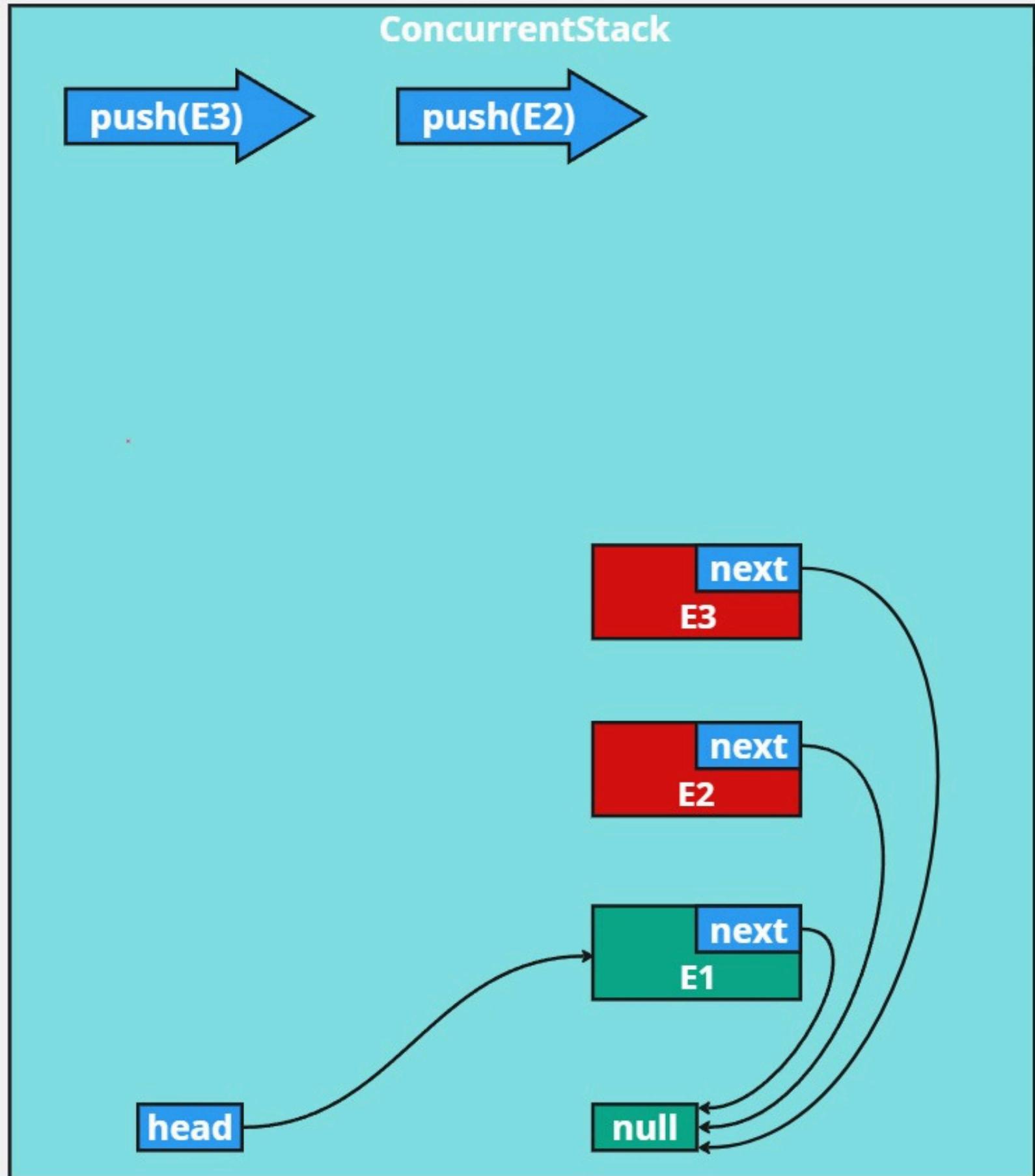
```

public void push(final T element) {
    while (true) {
        final Node<T> previousHead = head.get();
        final Node<T> newNode = new Node<>(element, previousHead);
        if (head.compareAndSet(previousHead, newNode)) {
            return;
        }
    }
}
  
```

push(E3)

```

public void push(final T element) {
    while (true) {
        final Node<T> previousHead = head.get();
        final Node<T> newNode = new Node<>(element, previousHead);
        if (head.compareAndSet(previousHead, newNode)) {
            return;
        }
    }
}
  
```



push(E1)

```

public void push(final T element) {
    while (true) {
        final Node<T> previousHead = head.get();
        final Node<T> newNode = new Node<>(element, previousHead);
        if (head.compareAndSet(previousHead, newNode)) {
            return;
        }
    }
}

```

push(E2)

```

public void push(final T element) {
    while (true) {
        final Node<T> previousHead = head.get();
        final Node<T> newNode = new Node<>(element, previousHead);
        if (head.compareAndSet(previousHead, newNode)) {
            return;
        }
    }
}

```

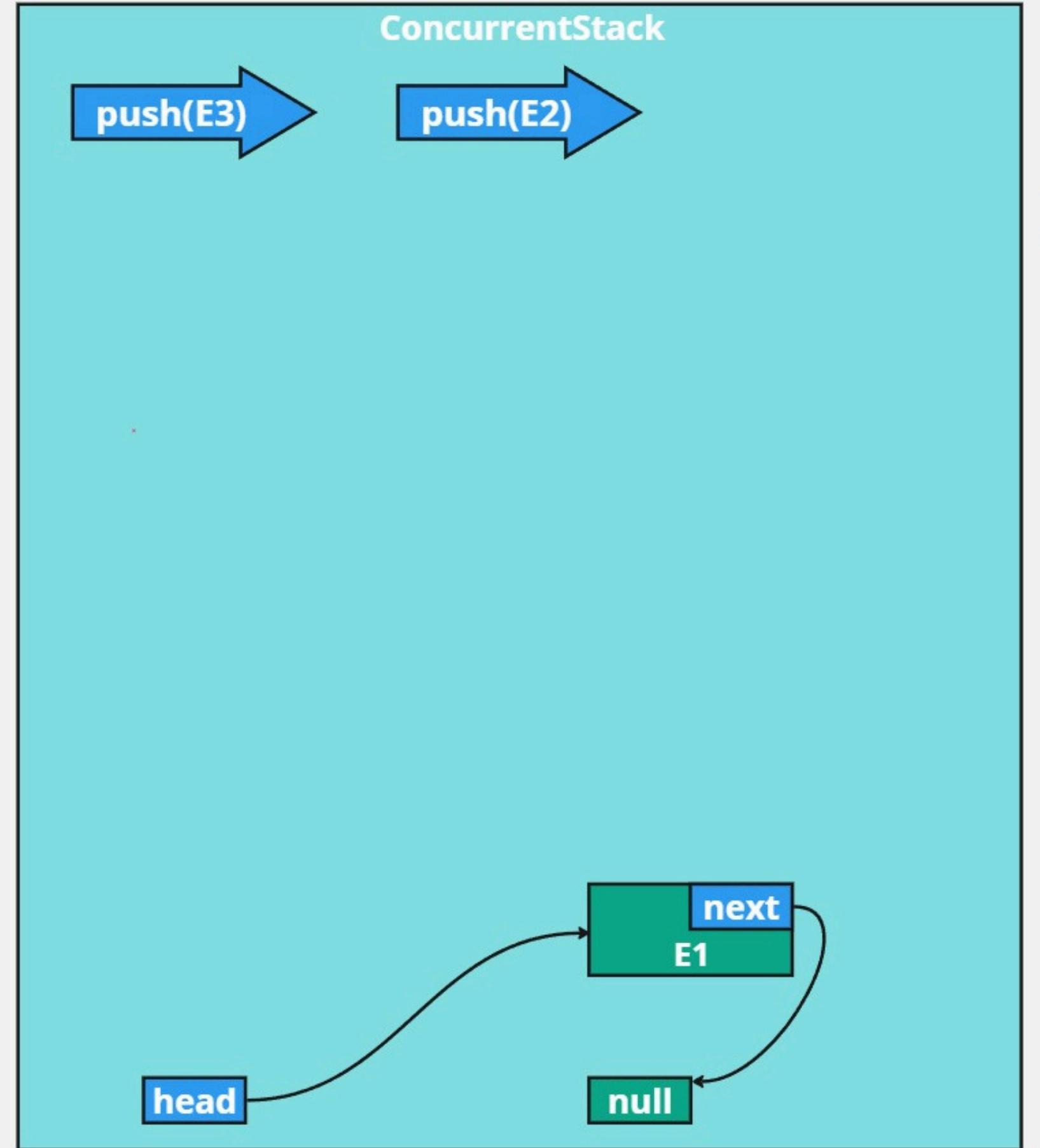
push(E3)

```

public void push(final T element) {
    while (true) {
        final Node<T> previousHead = head.get();
        final Node<T> newNode = new Node<>(element, previousHead);
        if (head.compareAndSet(previousHead, newNode)) {
            return;
        }
    }
}

```

The code snippets show the implementation of the `push` method for each element (E1, E2, and E3). The method uses a loop with a `while (true)` condition. Inside the loop, it retrieves the current head node using `head.get()`, creates a new node with the given element and the current head as its previous node, and then attempts to update the head using `head.compareAndSet`. If successful, it returns immediately. If unsuccessful, it loops back to try again.



push(E1)

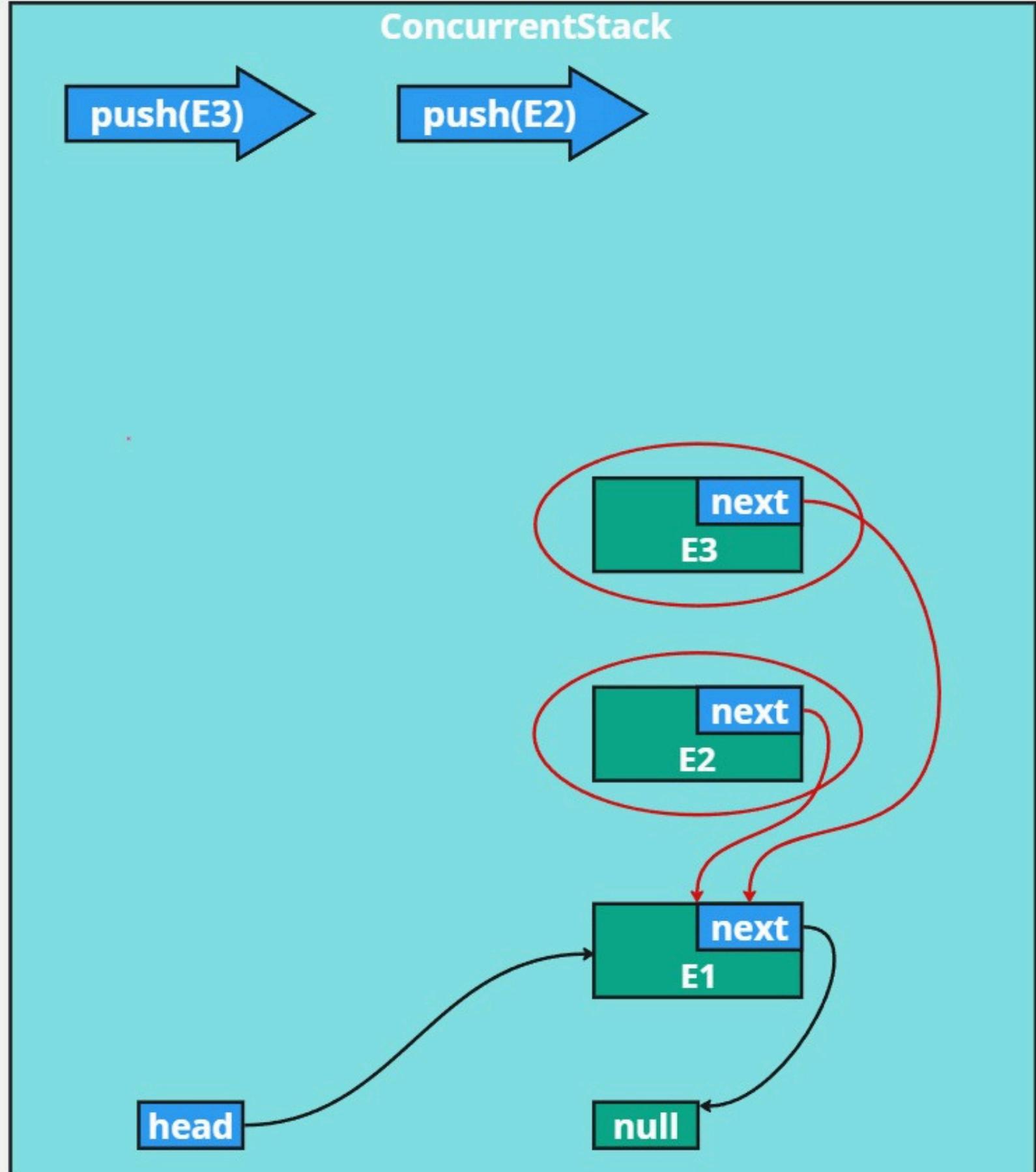
```
public void push(final T element) {
    while (true) {
        final Node<T> previousHead = head.get();
        final Node<T> newNode = new Node<>(element, previousHead);
        if (head.compareAndSet(previousHead, newNode)) {
            return;
        }
    }
}
```

push(E2)

```
public void push(final T element) {
    while (true) {
        final Node<T> previousHead = head.get();
        final Node<T> newNode = new Node<>(element, previousHead);
        if (head.compareAndSet(previousHead, newNode)) {
            return;
        }
    }
}
```

push(E3)

```
public void push(final T element) {
    while (true) {
        final Node<T> previousHead = head.get();
        final Node<T> newNode = new Node<>(element, previousHead);
        if (head.compareAndSet(previousHead, newNode)) {
            return;
        }
    }
}
```



push(E1)

```

public void push(final T element) {
    while (true) {
        final Node<T> previousHead = head.get();
        final Node<T> newNode = new Node<>(element, previousHead);
        if (head.compareAndSet(previousHead, newNode)) {
            return;
        }
    }
}
  
```

push(E2)

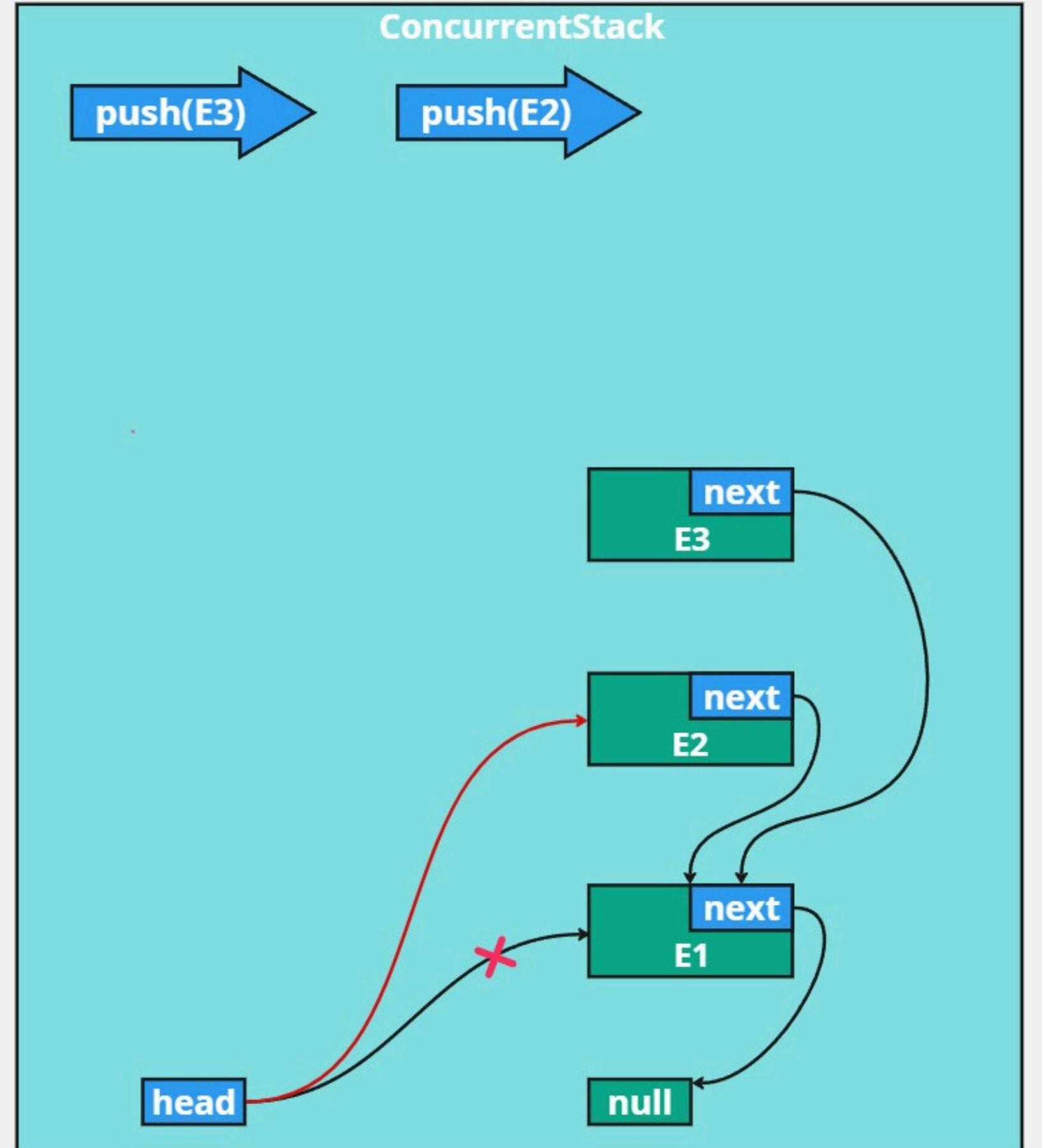
```

public void push(final T element) {
    while (true) {
        final Node<T> previousHead = head.get();
        final Node<T> newNode = new Node<>(element, previousHead);
        if (head.compareAndSet(previousHead, newNode)) {
            return;
        }
    }
}
  
```

push(E3)

```

public void push(final T element) {
    while (true) {
        final Node<T> previousHead = head.get();
        final Node<T> newNode = new Node<>(element, previousHead);
        if (head.compareAndSet(previousHead, newNode)) {
            return;
        }
    }
}
  
```



push(E1)

```
public void push(final T element) {
    while (true) {
        final Node<T> previousHead = head.get();
        final Node<T> newNode = new Node<>(element, previousHead);
        if (head.compareAndSet(previousHead, newNode)) {
            return;
        }
    }
}
```

push(E2)

```
public void push(final T element) {
    while (true) {
        final Node<T> previousHead = head.get();
        final Node<T> newNode = new Node<>(element, previousHead);
        if (head.compareAndSet(previousHead, newNode)) {
            return;
        }
    }
}
```

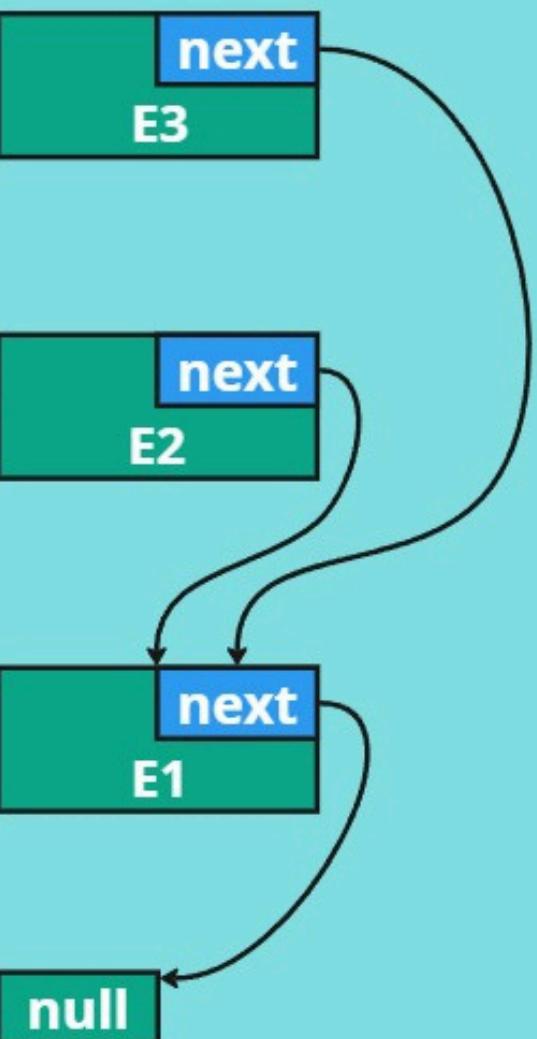
push(E3)

```
public void push(final T element) {
    while (true) {
        final Node<T> previousHead = head.get();
        final Node<T> newNode = new Node<>(element, previousHead);
        if (head.compareAndSet(previousHead, newNode)) {
            return;
        }
    }
}
```

Diagram illustrating the state of the concurrent stack after multiple pushes. The stack consists of three nodes: E3 (top), E2 (middle), and E1 (bottom). Each node has a 'next' field. A red arrow points from the 'head' pointer to the 'next' field of node E1, which is marked with a red asterisk (*). This indicates that the current head node (E1) is being updated by a new node (newNode), but the update has not yet been committed due to a race condition.

ConcurrentStack

push(E3)



push(E1)

```
public void push(final T element) {  
    while (true) {  
        final Node<T> previousHead = head.get();  
        final Node<T> newNode = new Node<>(element, previousHead);  
        if (head.compareAndSet(previousHead, newNode)) {  
            return;  
        }  
    }  
}
```

push(E1)

push(E2)

```
public void push(final T element) {  
    while (true) {  
        final Node<T> previousHead = head.get();  
        final Node<T> newNode = new Node<>(element, previousHead);  
        if (head.compareAndSet(previousHead, newNode)) {  
            return;  
        }  
    }  
}
```

push(E2)

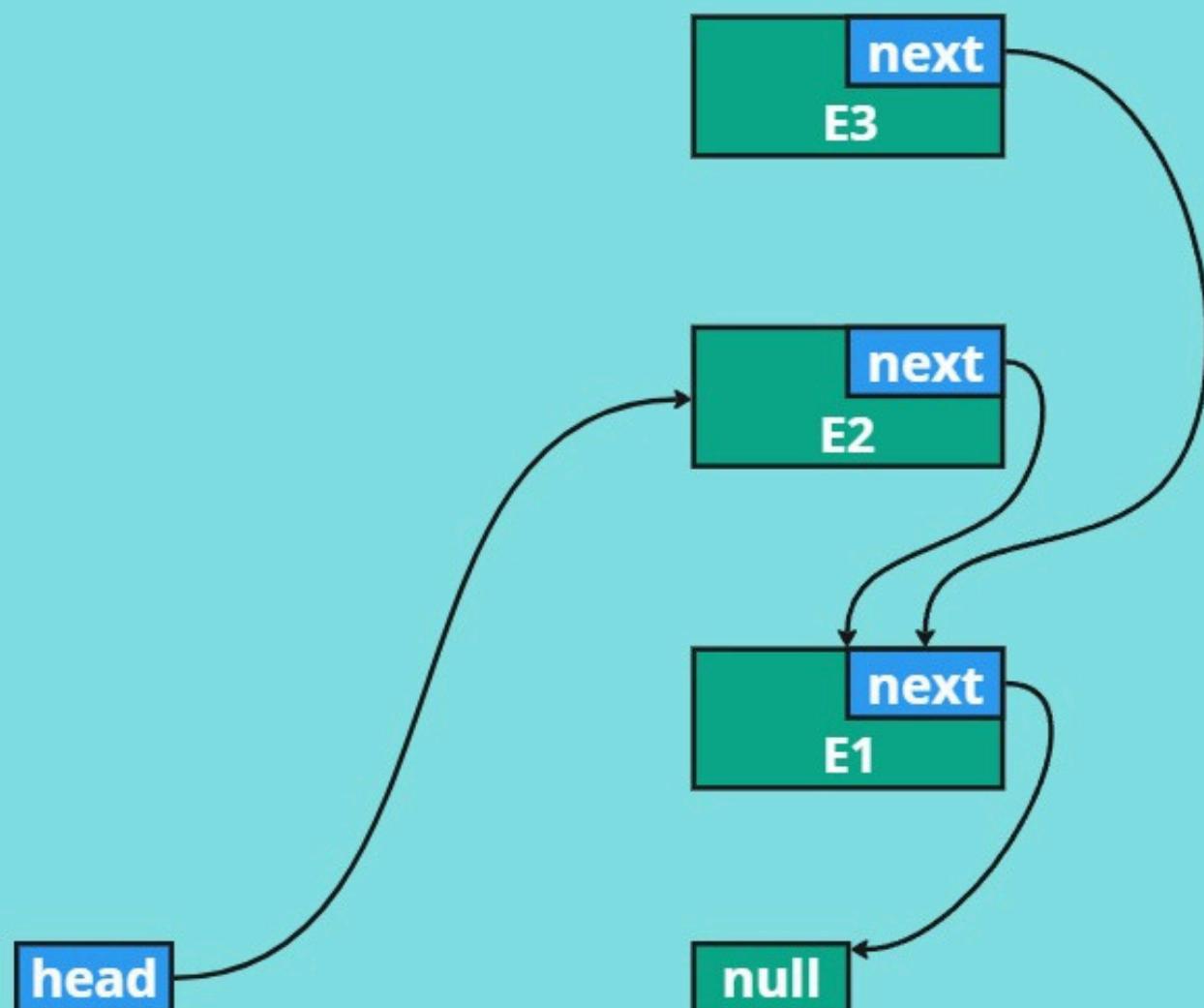
push(E3)

```
public void push(final T element) {  
    while (true) {  
        final Node<T> previousHead = head.get();  
        final Node<T> newNode = new Node<>(element, previousHead);  
        if (head.compareAndSet(previousHead, newNode)) {  
            return;  
        }  
    }  
}
```

push(E3)

ConcurrentStack

push(E3)



push(E1)

```
public void push(final T element) {  
    while (true) {  
        final Node<T> previousHead = head.get();  
        final Node<T> newNode = new Node<>(element, previousHead);  
        if (head.compareAndSet(previousHead, newNode)) {  
            return;  
        }  
    }  
}
```

push(E1)

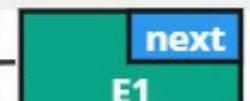
push(E2)

```
public void push(final T element) {  
    while (true) {  
        final Node<T> previousHead = head.get();  
        final Node<T> newNode = new Node<>(element, previousHead);  
        if (head.compareAndSet(previousHead, newNode)) {  
            return;  
        }  
    }  
}
```

push(E2)

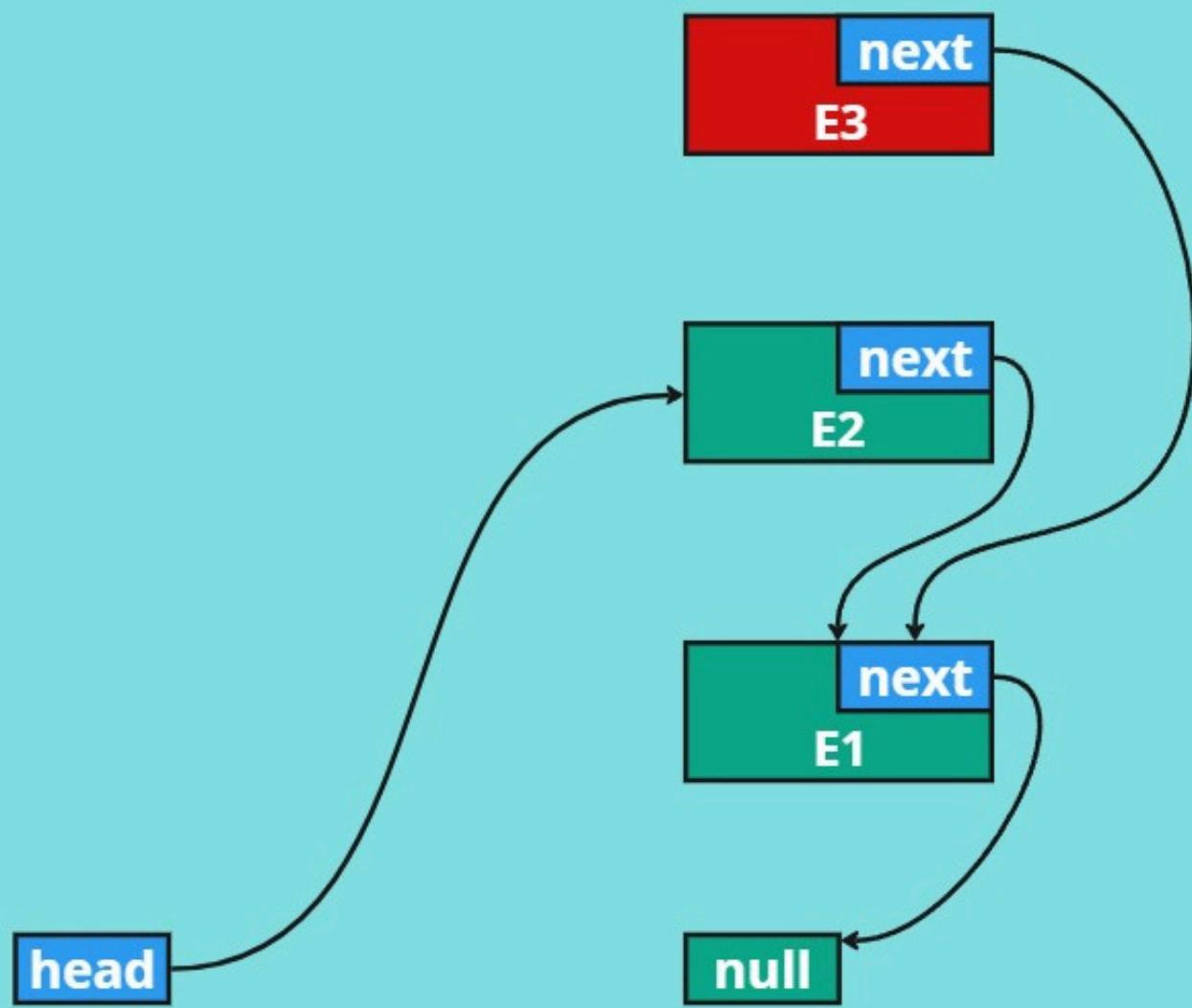
push(E3)

```
public void push(final T element) {  
    while (true) {  
        final Node<T> previousHead = head.get();  
        final Node<T> newNode = new Node<>(element, previousHead);  
        if (head.compareAndSet(previousHead, newNode)) {  
            return;  
        }  
    }  
}
```



ConcurrentStack

push(E3)



push(E1)

```
public void push(final T element) {  
    while (true) {  
        final Node<T> previousHead = head.get();  
        final Node<T> newNode = new Node<>(element, previousHead);  
        if (head.compareAndSet(previousHead, newNode)) {  
            return;  
        }  
    }  
}
```

push(E1)

push(E2)

```
public void push(final T element) {  
    while (true) {  
        final Node<T> previousHead = head.get();  
        final Node<T> newNode = new Node<>(element, previousHead);  
        if (head.compareAndSet(previousHead, newNode)) {  
            return;  
        }  
    }  
}
```

push(E2)

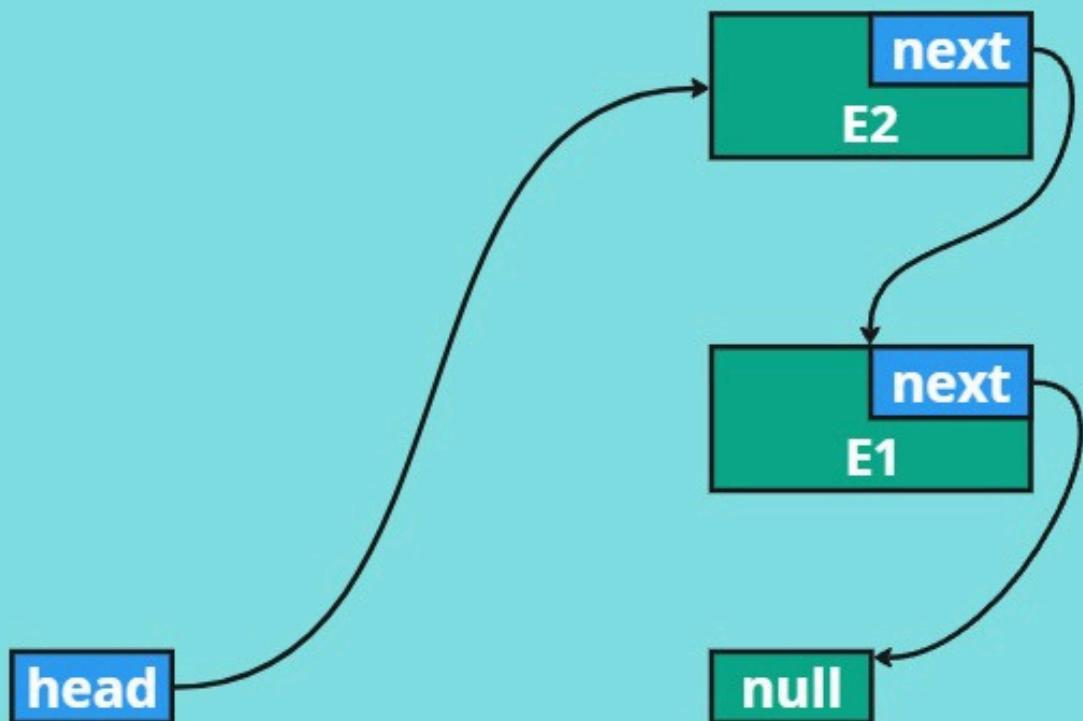
push(E3)

```
public void push(final T element) {  
    while (true) {  
        final Node<T> previousHead = head.get();  
        final Node<T> newNode = new Node<>(element, previousHead);  
        if (head.compareAndSet(previousHead, newNode)) {  
            return;  
        }  
    }  
}
```

push(E3)

ConcurrentStack

push(E3)



push(E1)

push(E1)

```
public void push(final T element) {  
    while (true) {  
        final Node<T> previousHead = head.get();  
        final Node<T> newNode = new Node<>(element, previousHead);  
        if (head.compareAndSet(previousHead, newNode)) {  
            return;  
        }  
    }  
}
```

push(E2)

push(E2)

```
public void push(final T element) {  
    while (true) {  
        final Node<T> previousHead = head.get();  
        final Node<T> newNode = new Node<>(element, previousHead);  
        if (head.compareAndSet(previousHead, newNode)) {  
            return;  
        }  
    }  
}
```

push(E3)

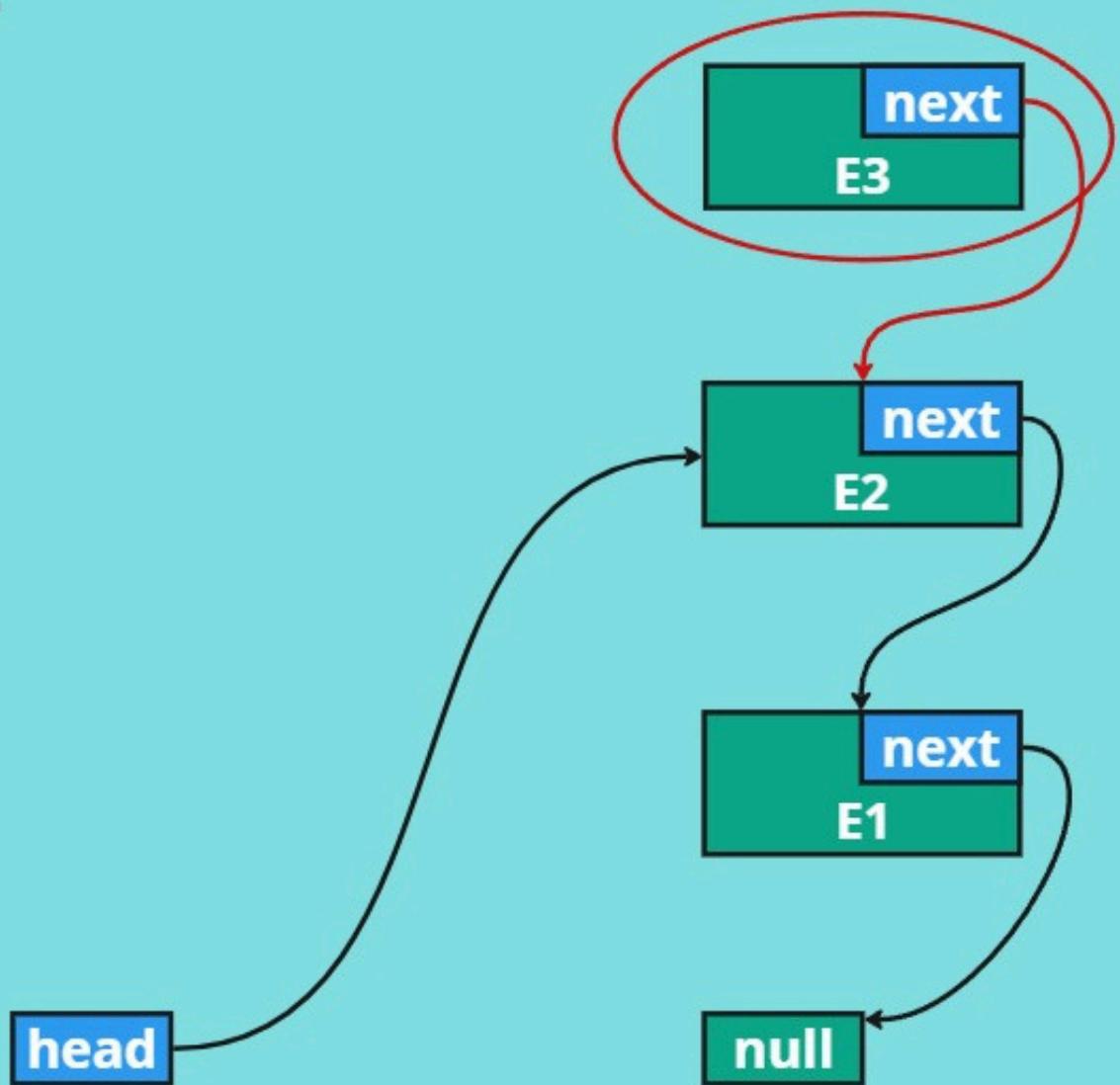
push(E3)

```
public void push(final T element) {  
    while (true) {  
        final Node<T> previousHead = head.get();  
        final Node<T> newNode = new Node<>(element, previousHead);  
        if (head.compareAndSet(previousHead, newNode)) {  
            return;  
        }  
    }  
}
```

A red box highlights the line 'final Node<T> previousHead = head.get();' in the code. A dashed arrow points from this highlighted line to the 'previousHead' field in the 'push(E3)' diagram above, indicating that the current head node (E2) is being used as the previous head for the new node (E3).

ConcurrentStack

push(E3)



push(E1)

```
public void push(final T element) {  
    while (true) {  
        final Node<T> previousHead = head.get();  
        final Node<T> newNode = new Node<>(element, previousHead);  
        if (head.compareAndSet(previousHead, newNode)) {  
            return;  
        }  
    }  
}
```

push(E2)

```
public void push(final T element) {  
    while (true) {  
        final Node<T> previousHead = head.get();  
        final Node<T> newNode = new Node<>(element, previousHead);  
        if (head.compareAndSet(previousHead, newNode)) {  
            return;  
        }  
    }  
}
```

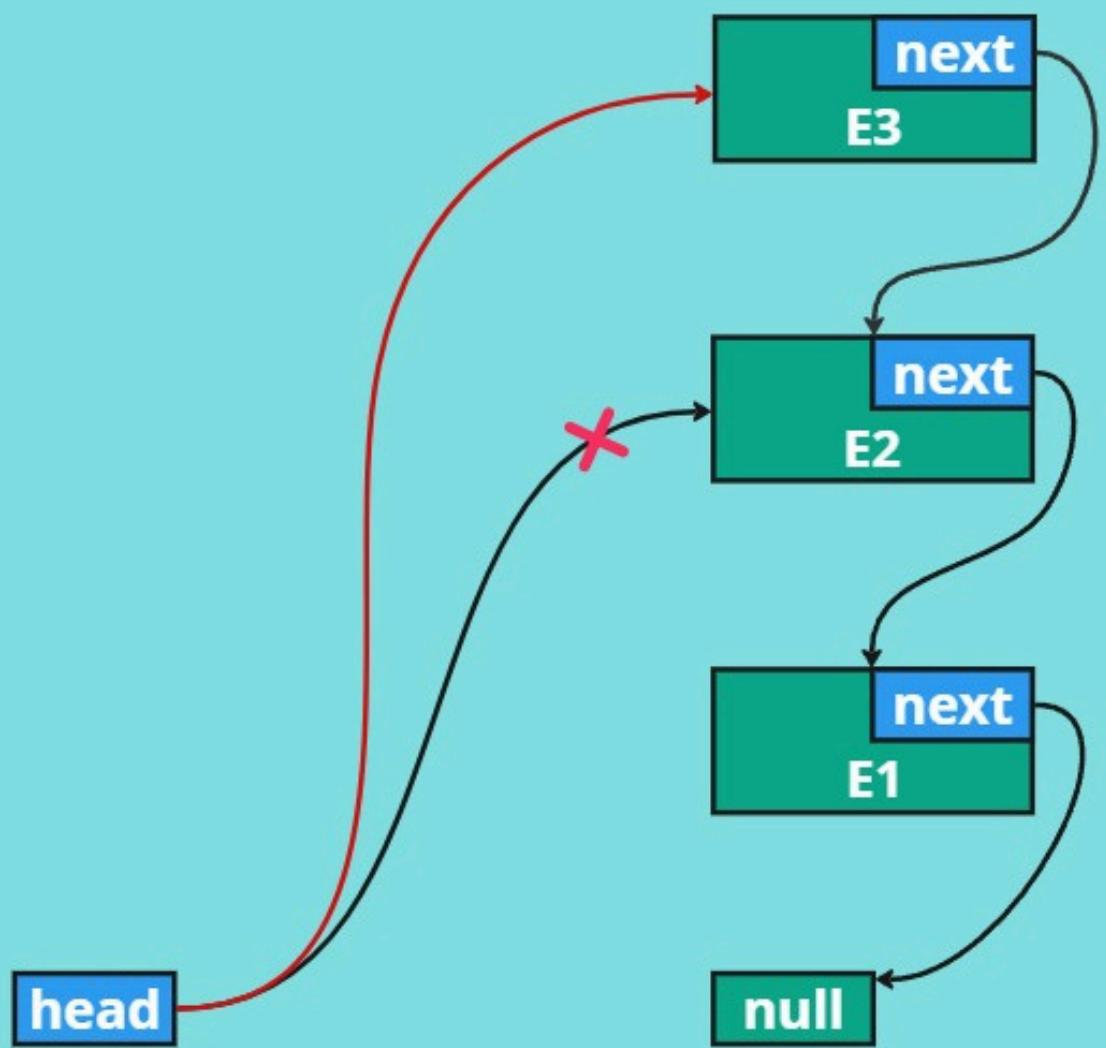
push(E3)

```
public void push(final T element) {  
    while (true) {  
        final Node<T> previousHead = head.get();  
        final Node<T> newNode = new Node<>(element, previousHead);  
        if (head.compareAndSet(previousHead, newNode)) {  
            return;  
        }  
    }  
}
```

A dashed line connects the 'push(E3)' label to the 'newNode = new Node<>(element, previousHead);' line in the code block. A red box highlights this line.

ConcurrentStack

push(E3)



push(E1)

```
public void push(final T element) {  
    while (true) {  
        final Node<T> previousHead = head.get();  
        final Node<T> newNode = new Node<>(element, previousHead);  
        if (head.compareAndSet(previousHead, newNode)) {  
            return;  
        }  
    }  
}
```

push(E1)

push(E2)

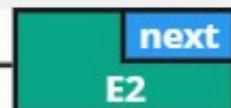
```
public void push(final T element) {  
    while (true) {  
        final Node<T> previousHead = head.get();  
        final Node<T> newNode = new Node<>(element, previousHead);  
        if (head.compareAndSet(previousHead, newNode)) {  
            return;  
        }  
    }  
}
```

push(E2)

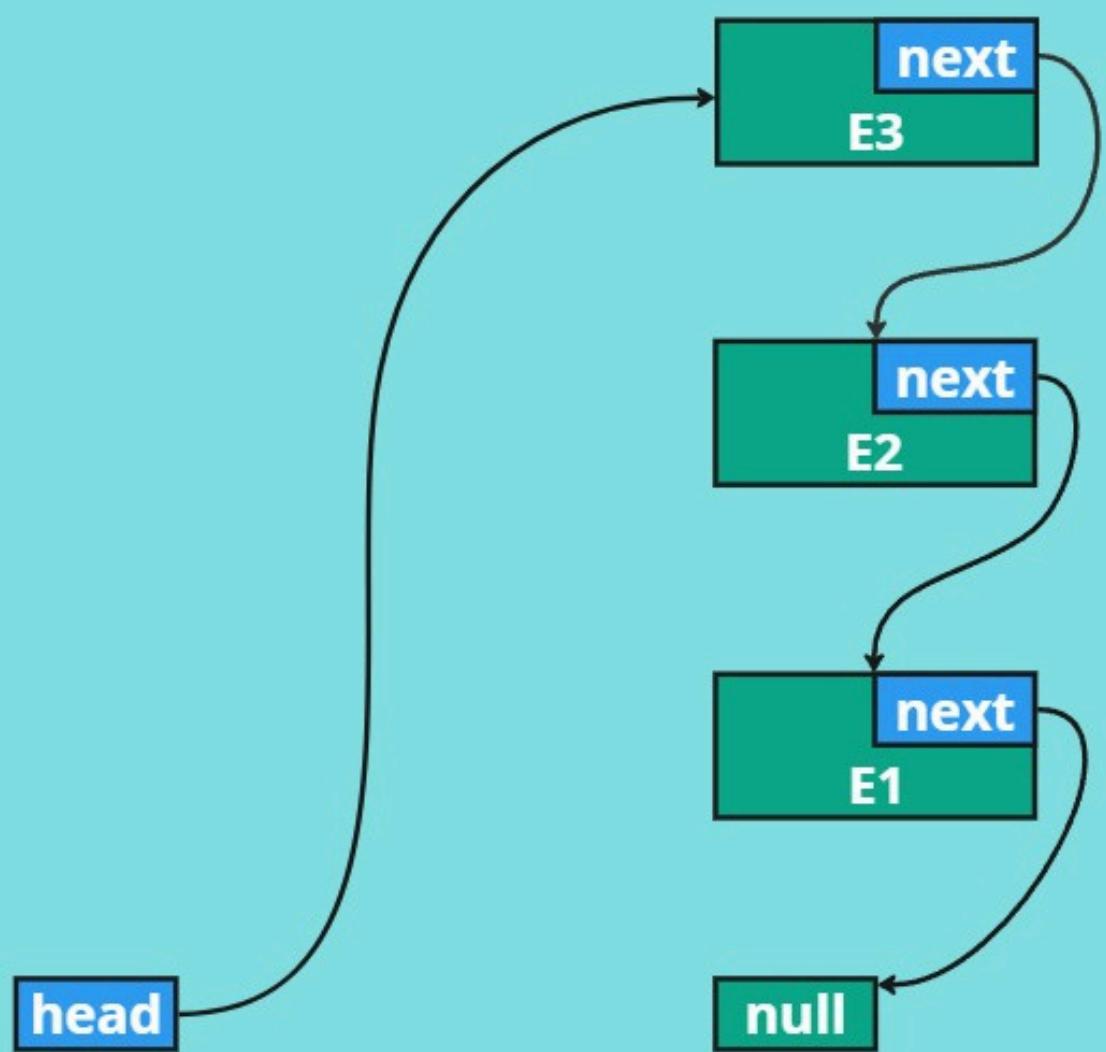
push(E3)

```
public void push(final T element) {  
    while (true) {  
        final Node<T> previousHead = head.get();  
        final Node<T> newNode = new Node<>(element, previousHead);  
        if (head.compareAndSet(previousHead, newNode)) {  
            return;  
        }  
    }  
}
```

A red box highlights the line of code where the compareAndSet operation is performed on the head node.



ConcurrentStack



push(E1)

```
public void push(final T element) {  
    while (true) {  
        final Node<T> previousHead = head.get();  
        final Node<T> newNode = new Node<>(element, previousHead);  
        if (head.compareAndSet(previousHead, newNode)) {  
            return;  
        }  
    }  
}
```

push(E2)

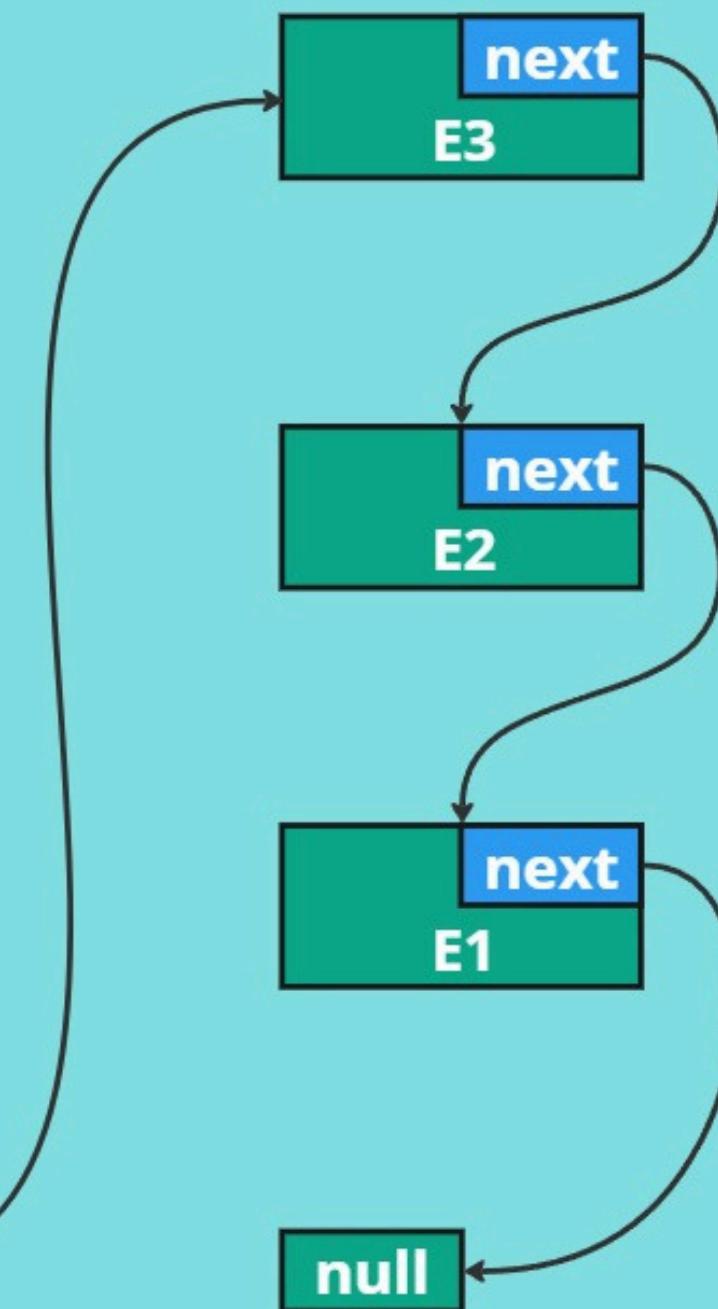
```
public void push(final T element) {  
    while (true) {  
        final Node<T> previousHead = head.get();  
        final Node<T> newNode = new Node<>(element, previousHead);  
        if (head.compareAndSet(previousHead, newNode)) {  
            return;  
        }  
    }  
}
```

push(E3)

```
public void push(final T element) {  
    while (true) {  
        final Node<T> previousHead = head.get();  
        final Node<T> newNode = new Node<>(element, previousHead);  
        if (head.compareAndSet(previousHead, newNode)) {  
            return;  
        }  
    }  
}
```

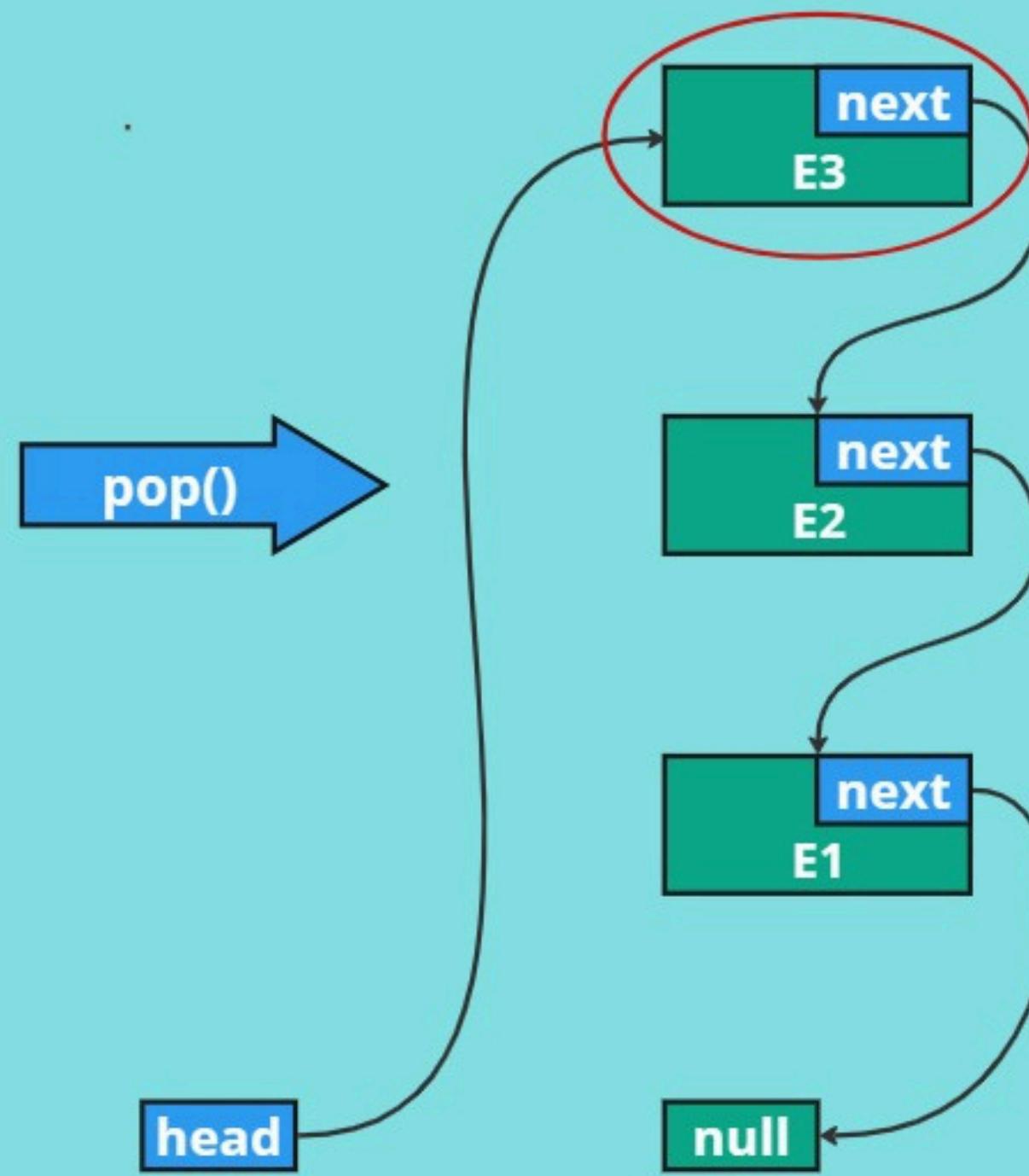
ConcurrentStack

pop()



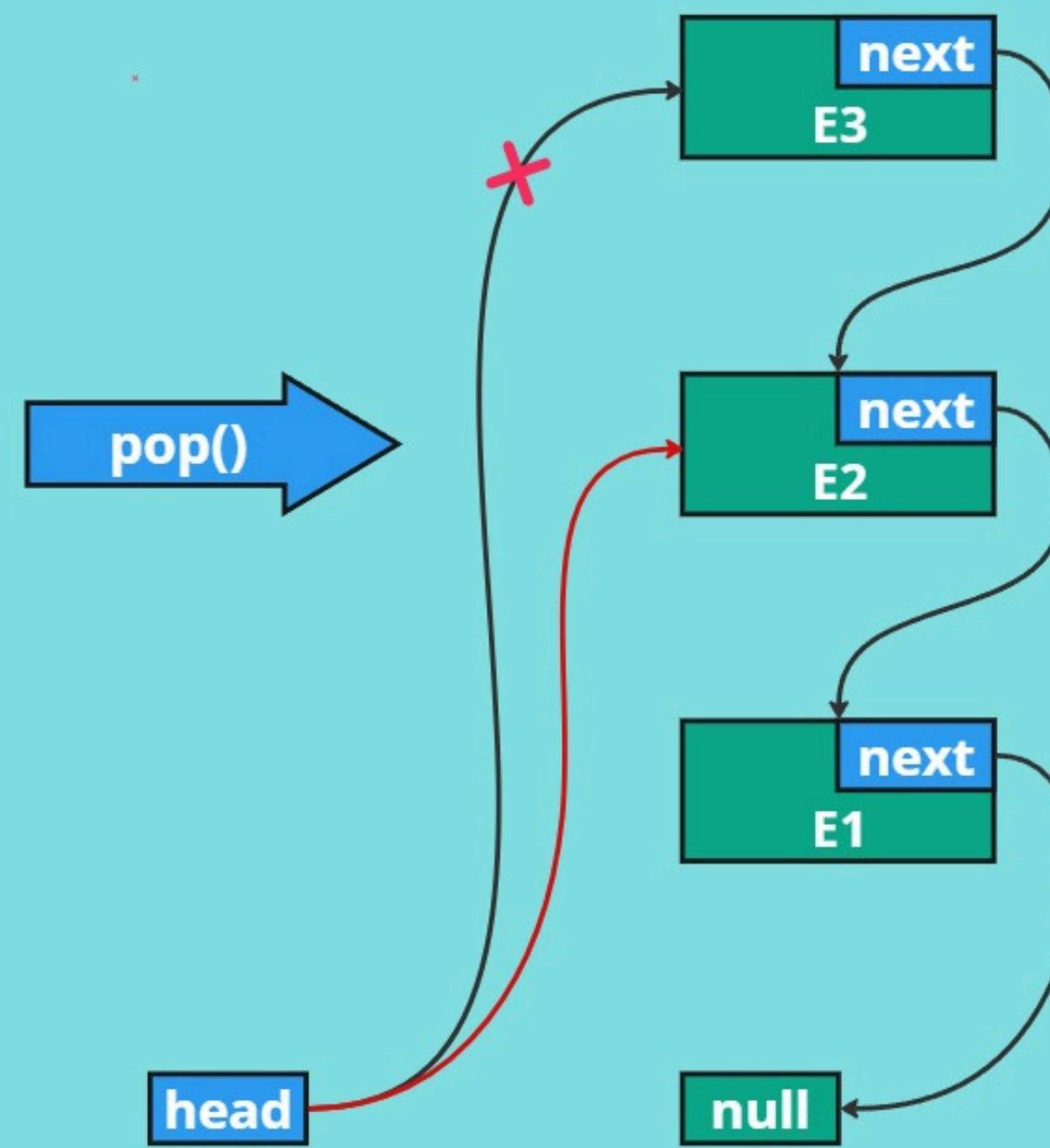
```
public T pop() {  
    //memorize previous head  
    //head should be assigned previous head's next node  
    //extract previous head's value  
    //return extracted value  
}
```

ConcurrentStack



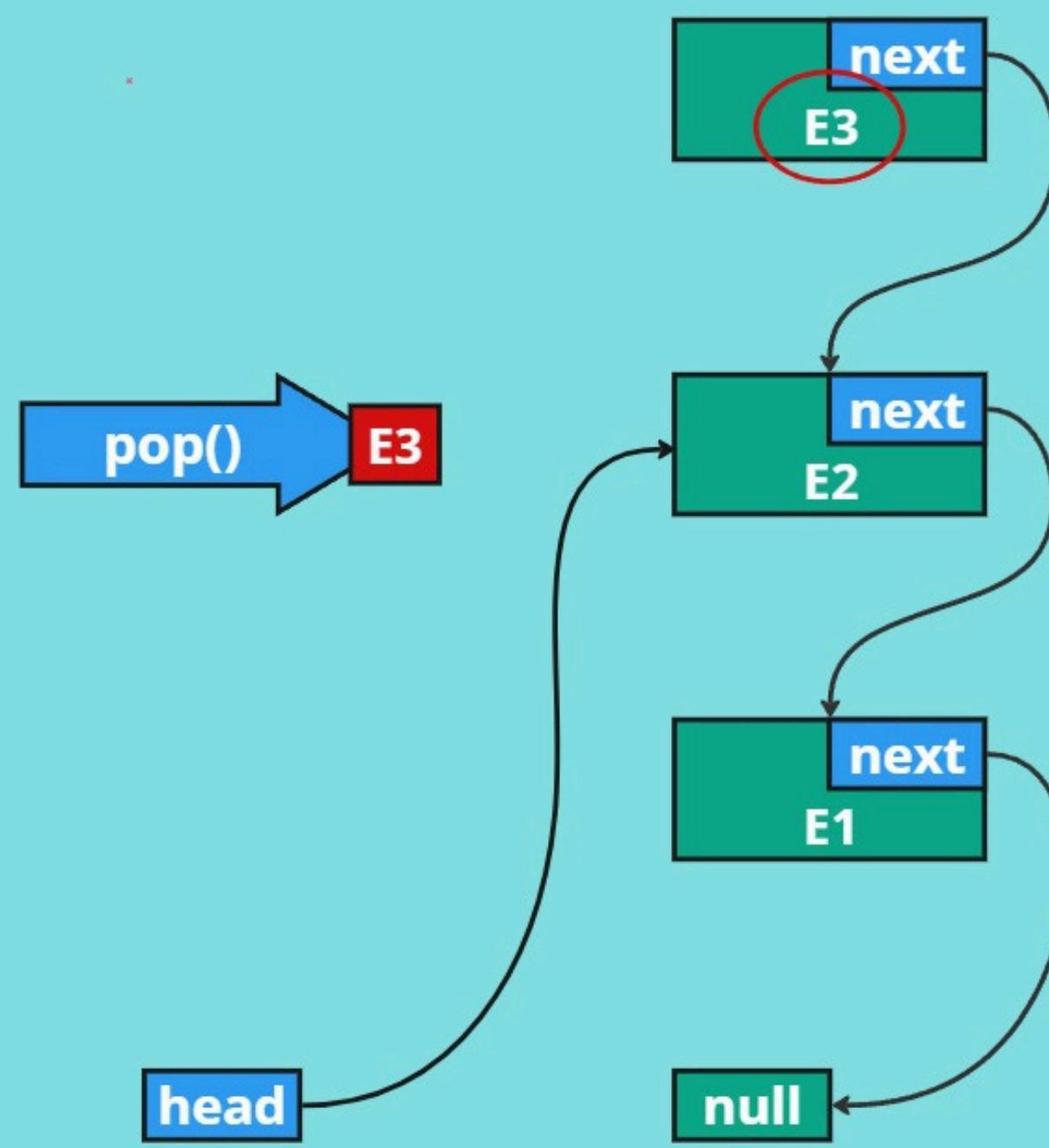
```
public T pop() {  
    //memorize previous head  
    //head should be assigned previous head's next node  
    //extract previous head's value  
    //return extracted value  
}
```

ConcurrentStack



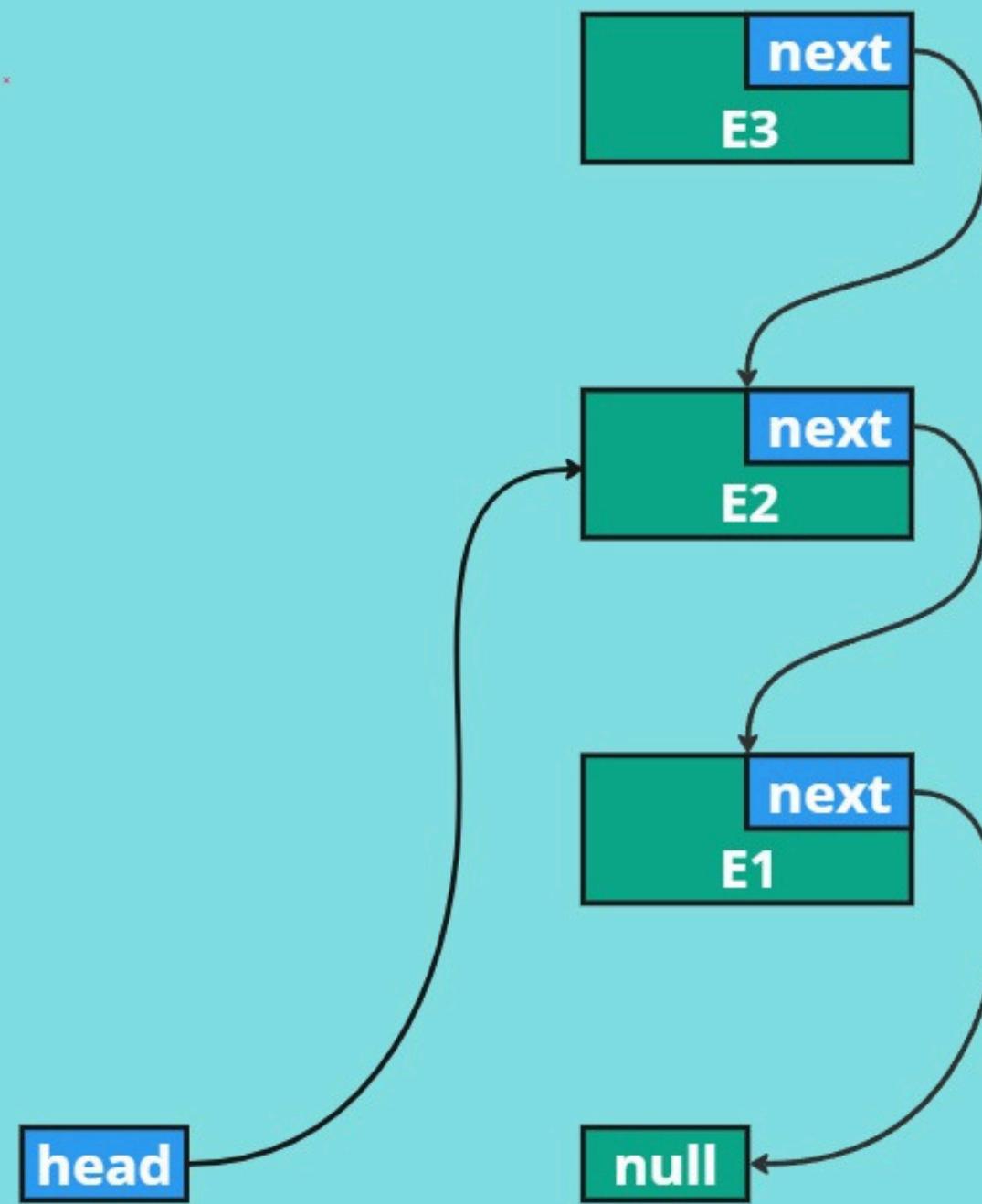
```
public T pop() {  
    //memorize previous head  
    //head should be assigned previous head's next node  
    //extract previous head's value  
    //return extracted value  
}
```

ConcurrentStack



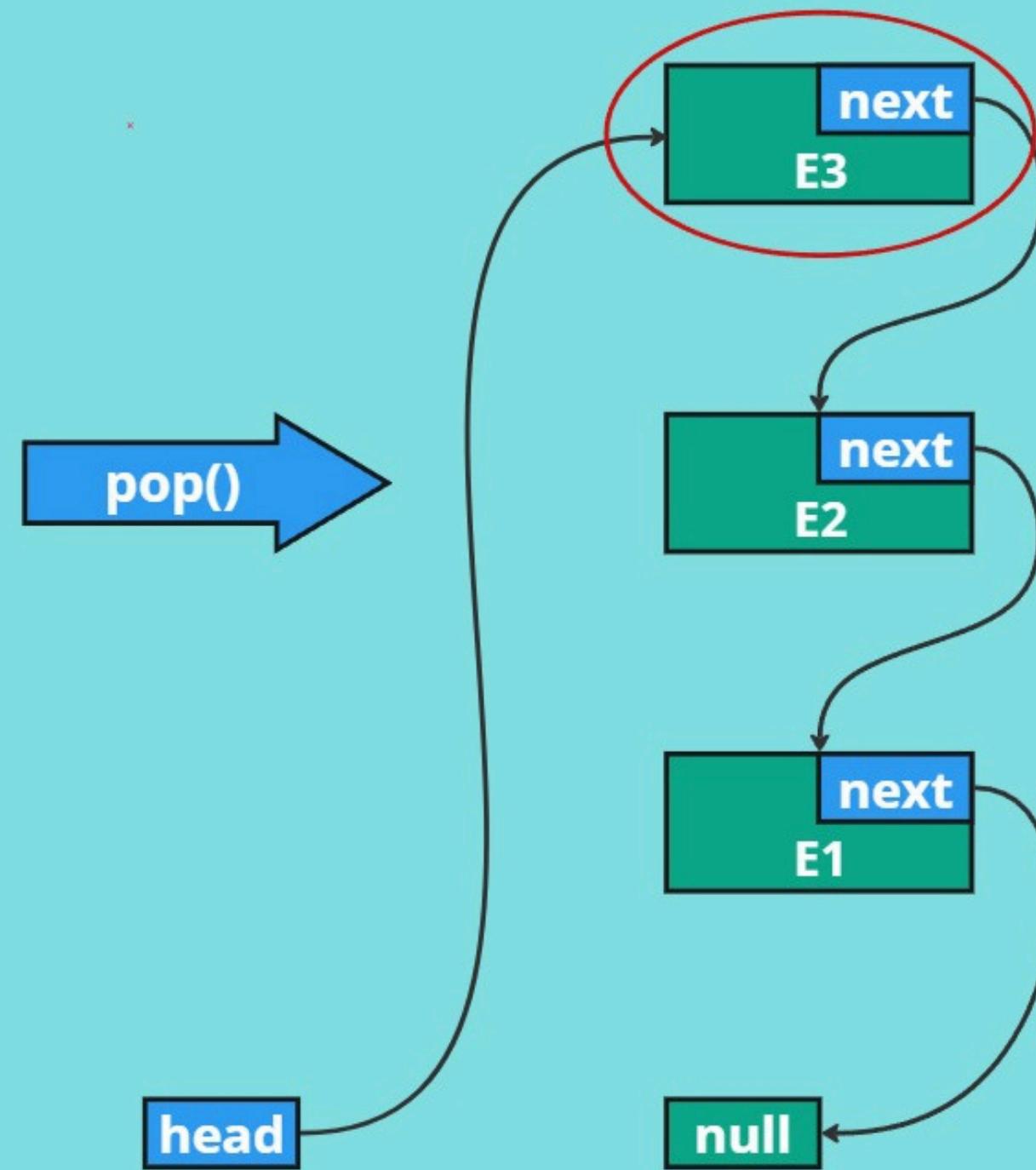
```
public T pop() {  
    //memorize previous head  
    //head should be assigned previous head's next node  
    //extract previous head's value  
    //return extracted value  
}
```

ConcurrentStack



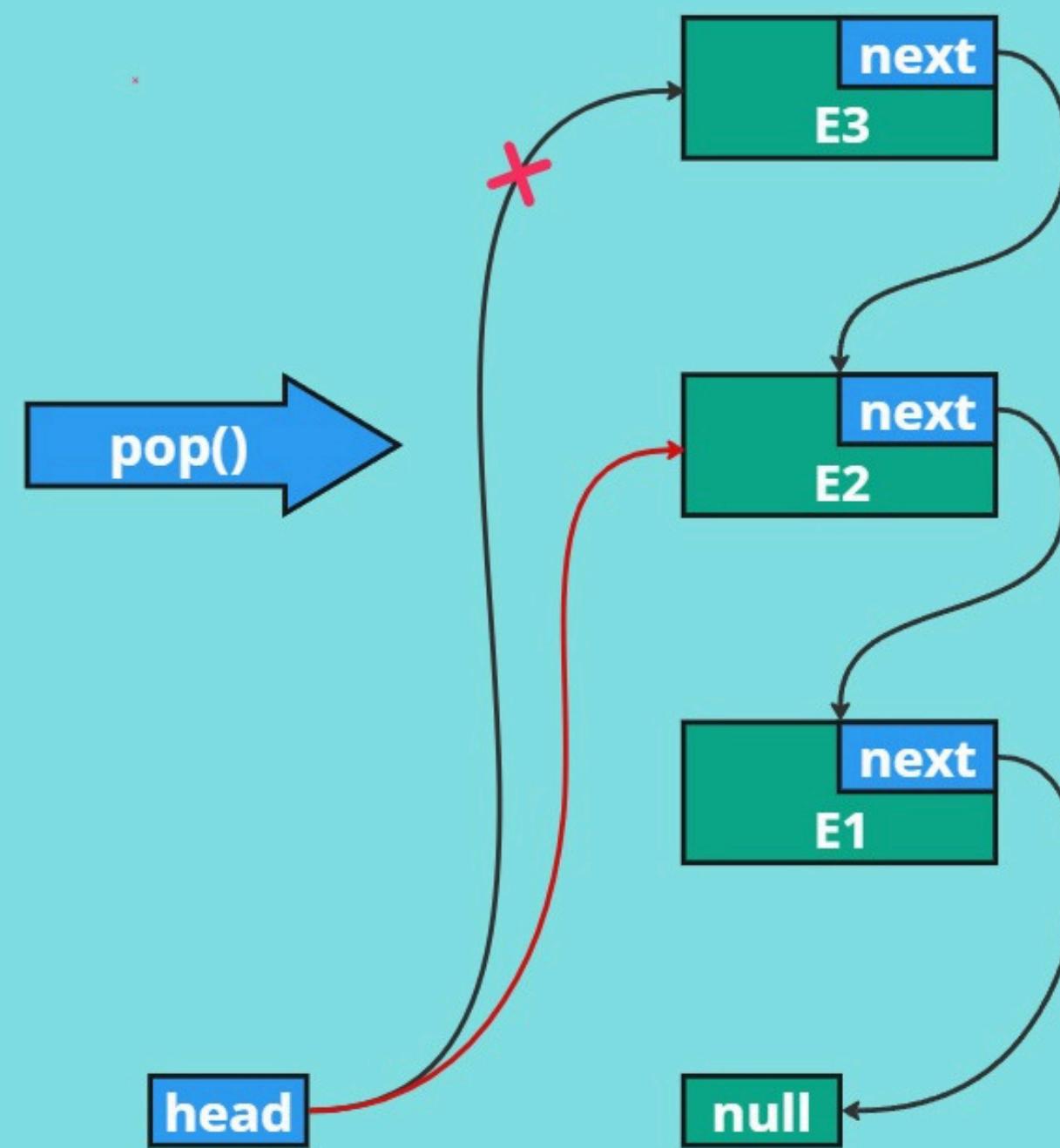
```
public T pop() {  
    //memorize previous head  
    //head should be assigned previous head's next node  
    //extract previous head's value  
    //return extracted value  
}
```

ConcurrentStack



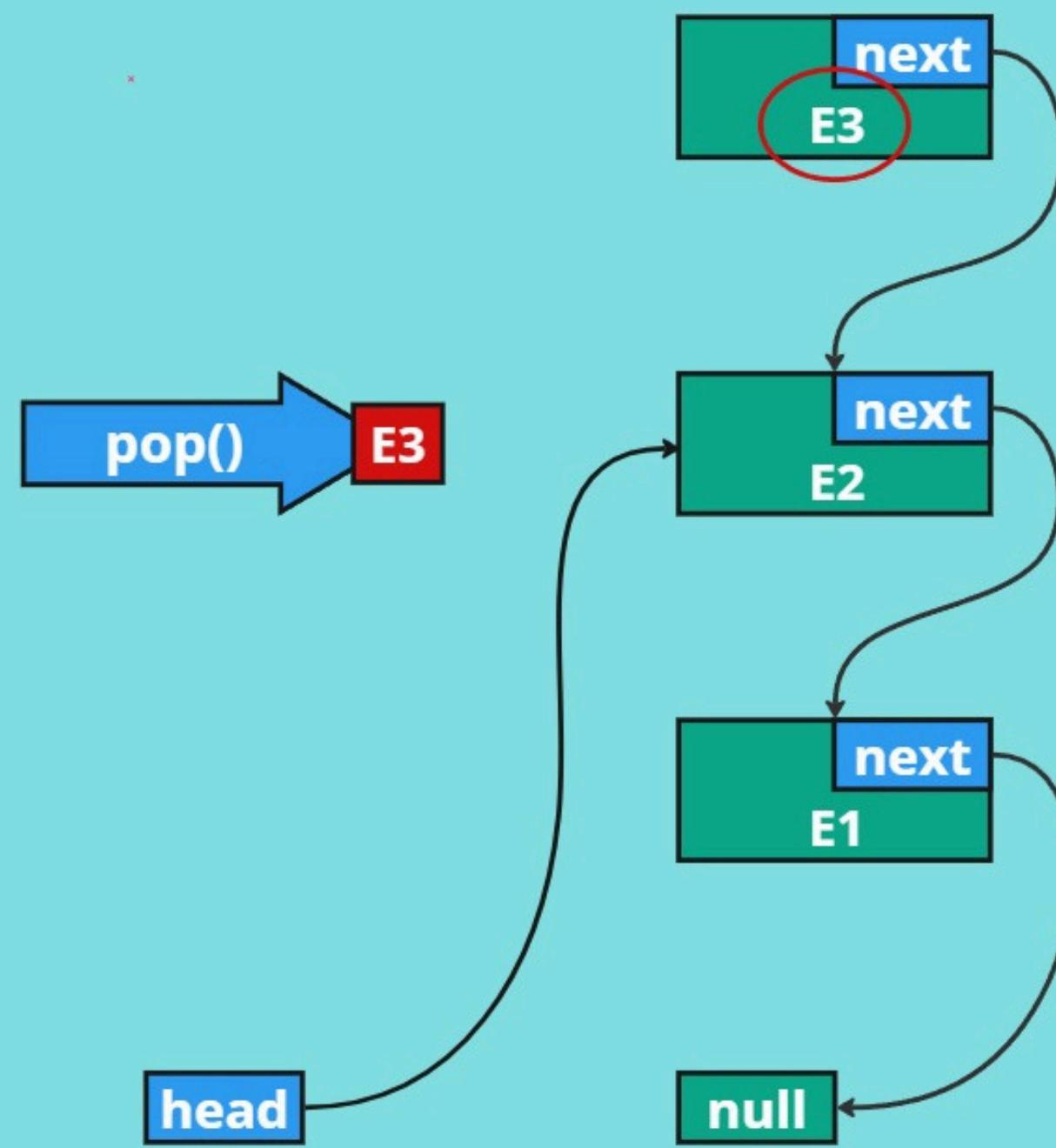
```
public T pop() {  
    final Node<T> previousHead = head.get();  
    //head should be assigned previous head's next node  
    //extract previous head's value  
    //return extracted value  
}
```

ConcurrentStack



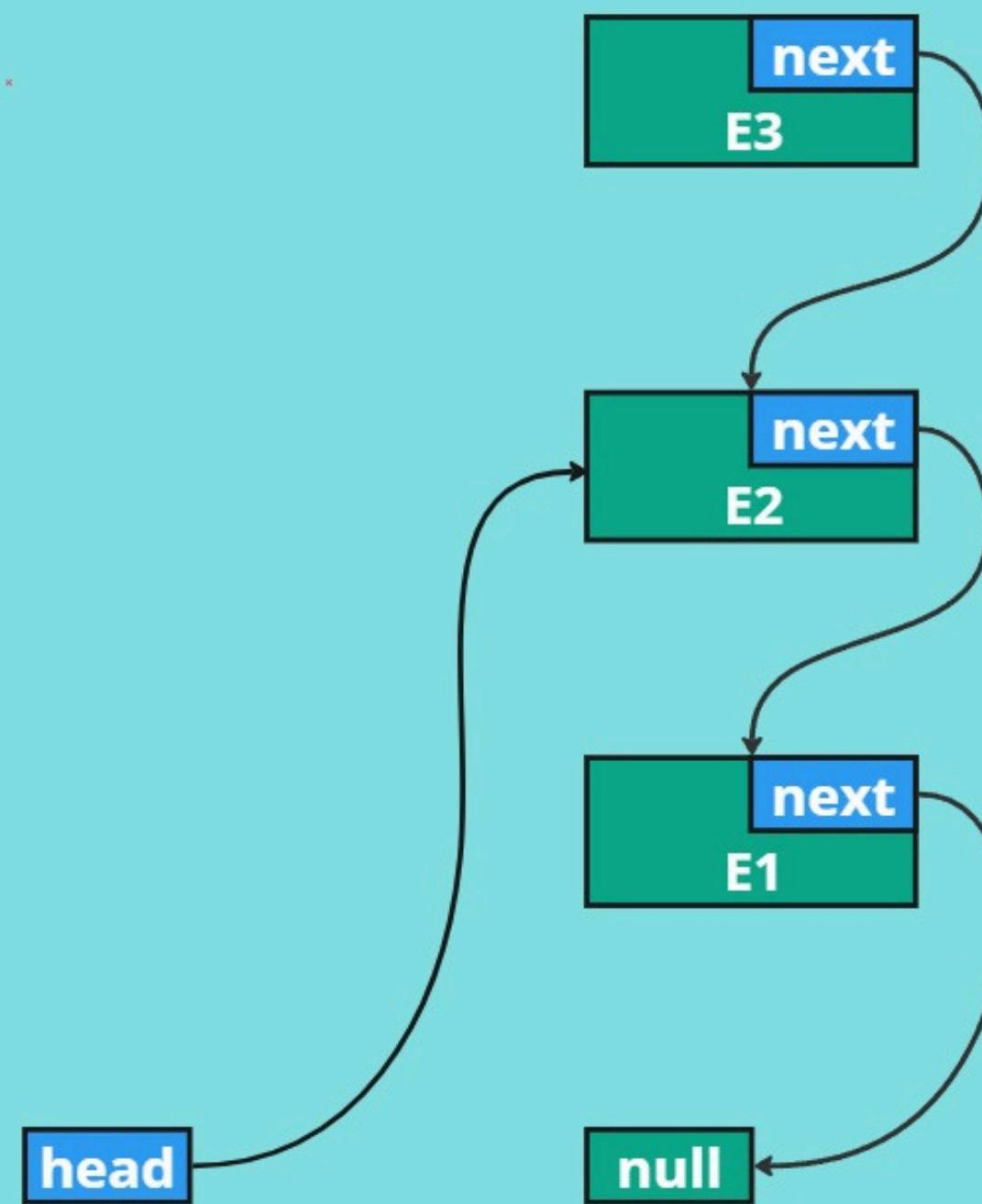
```
public T pop() {  
    final Node<T> previousHead = head.get();  
    head.set(previousHead.next);  
    //extract previous head's value  
    //return extracted value  
}
```

ConcurrentStack



```
public T pop() {  
    final Node<T> previousHead = head.get();  
    head.set(previousHead.next);  
    return previousHead.value;  
}
```

ConcurrentStack



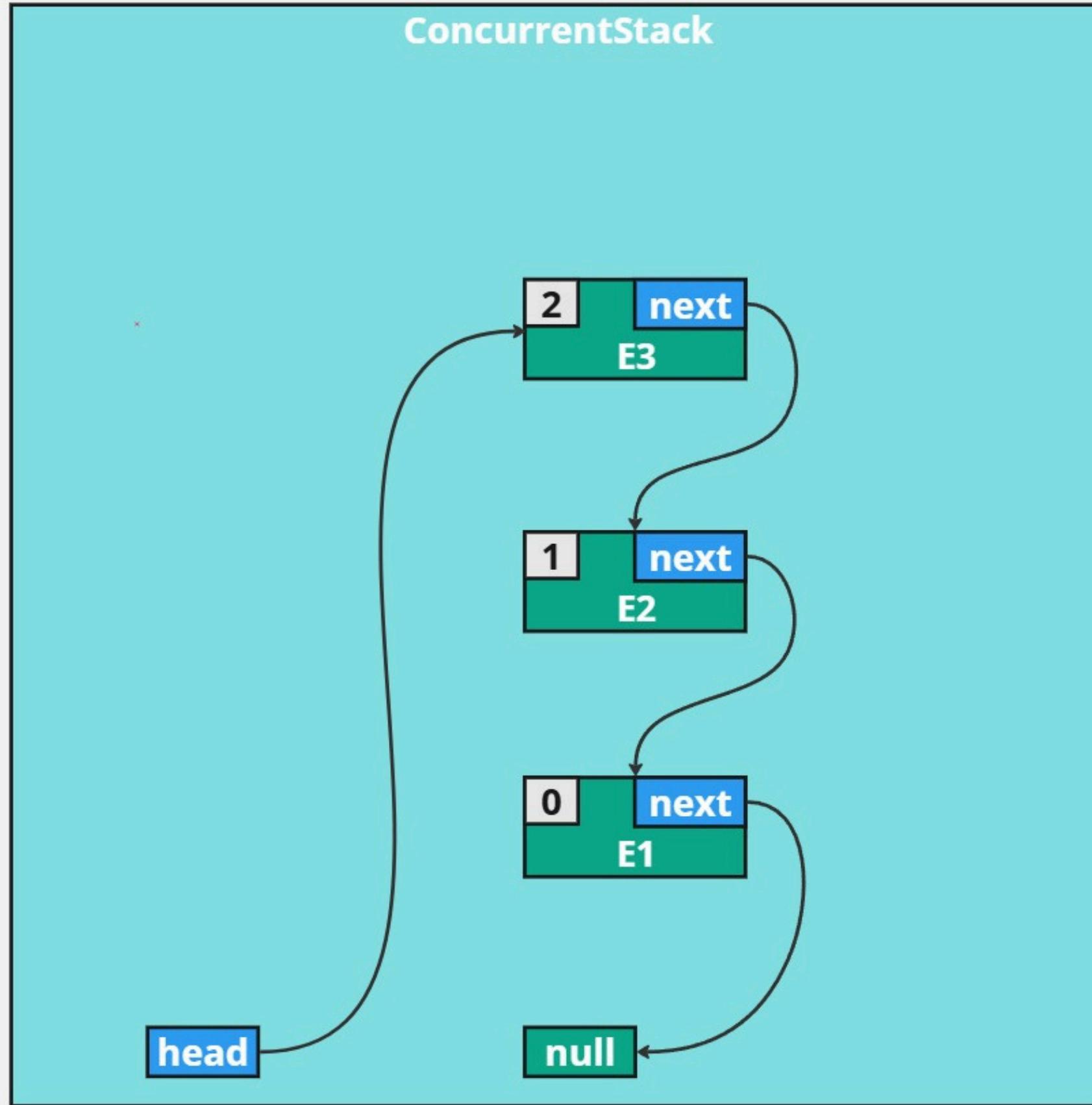
pop() → E3

```
public T pop() {  
    final Node<T> previousHead = head.get();  
    head.set(previousHead.next);  
    return previousHead.value;  
}
```

thread-1 →

thread-2 →

thread-3 →



thread-1

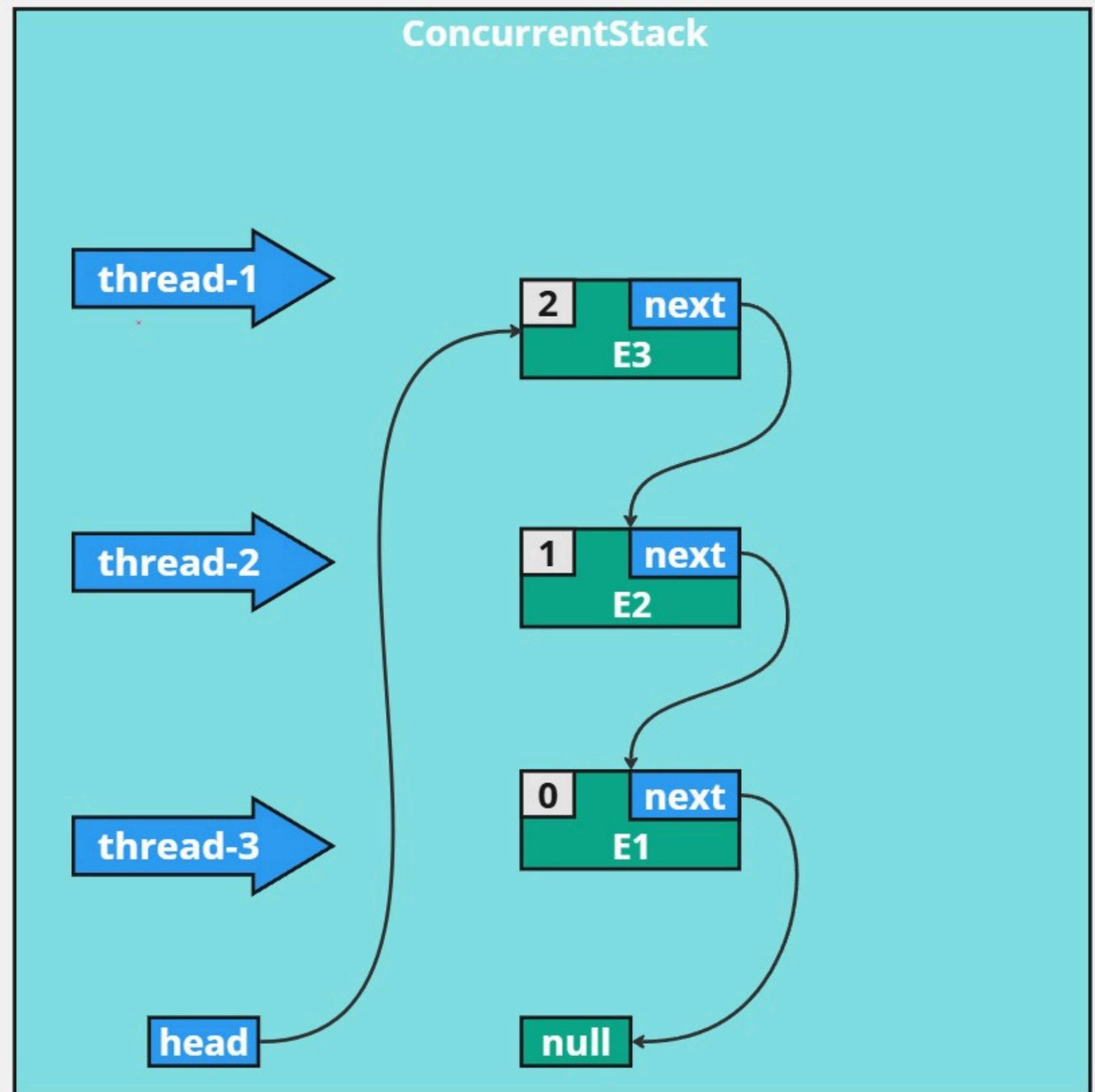
```
public T pop() {  
    final Node<T> previousHead = head.get();  
    head.set(previousHead.next);  
    return previousHead.value;  
}
```

thread-2

```
public T pop() {  
    final Node<T> previousHead = head.get();  
    head.set(previousHead.next);  
    return previousHead.value;  
}
```

thread-3

```
public T pop() {  
    final Node<T> previousHead = head.get();  
    head.set(previousHead.next);  
    return previousHead.value;  
}
```



thread-1

```

public T pop() {
    final Node<T> previousHead = head.get();
    head.set(previousHead.next);
    return previousHead.value;
}

```

2

thread-2

```

public T pop() {
    final Node<T> previousHead = head.get();
    head.set(previousHead.next);
    return previousHead.value;
}

```

2

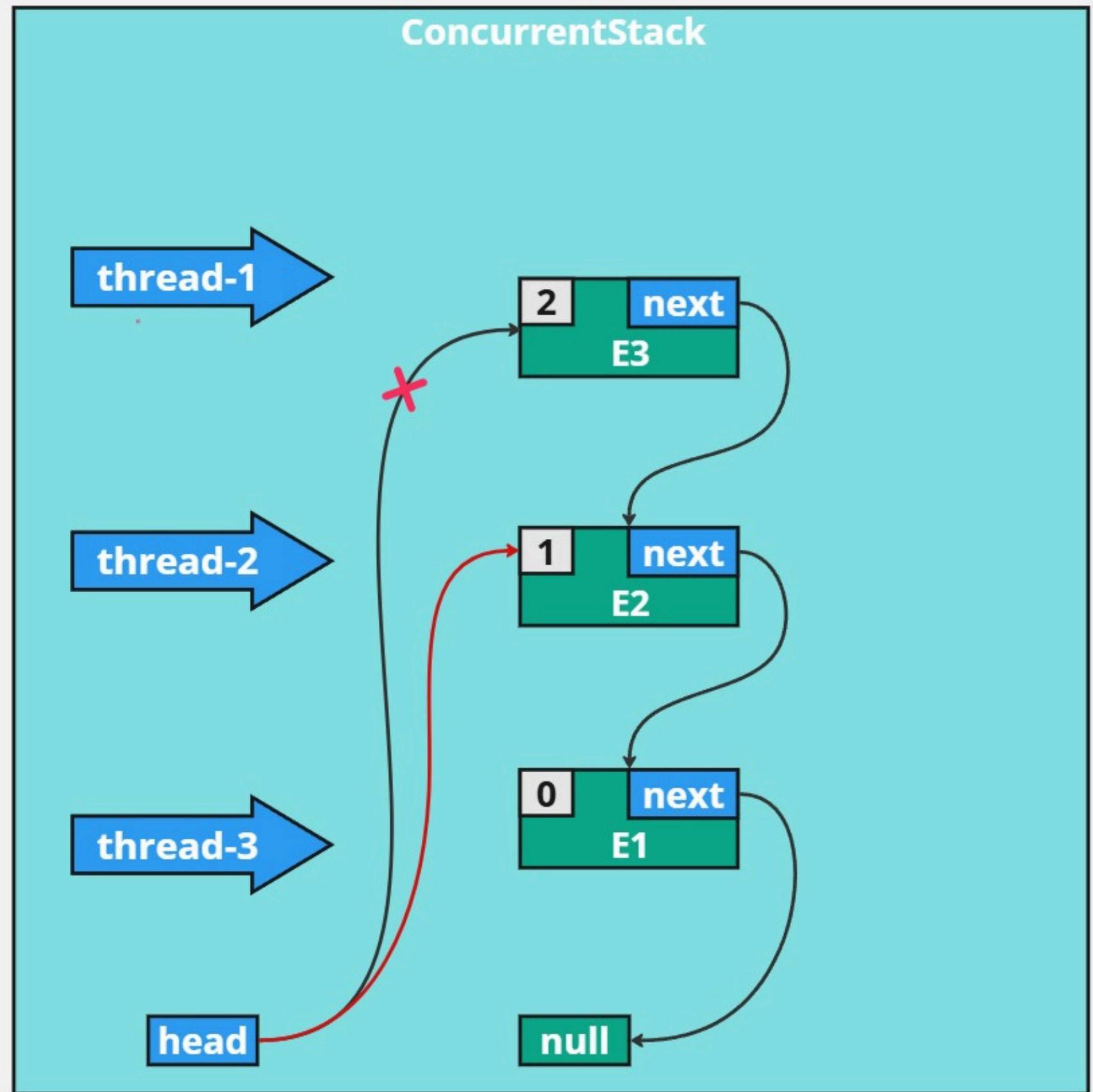
thread-3

```

public T pop() {
    final Node<T> previousHead = head.get();
    head.set(previousHead.next);
    return previousHead.value;
}

```

2



thread-1

```
public T pop() {
    final Node<T> previousHead = head.get();
    head.set(previousHead.next);
    return previousHead.value;
}
```

2

thread-2

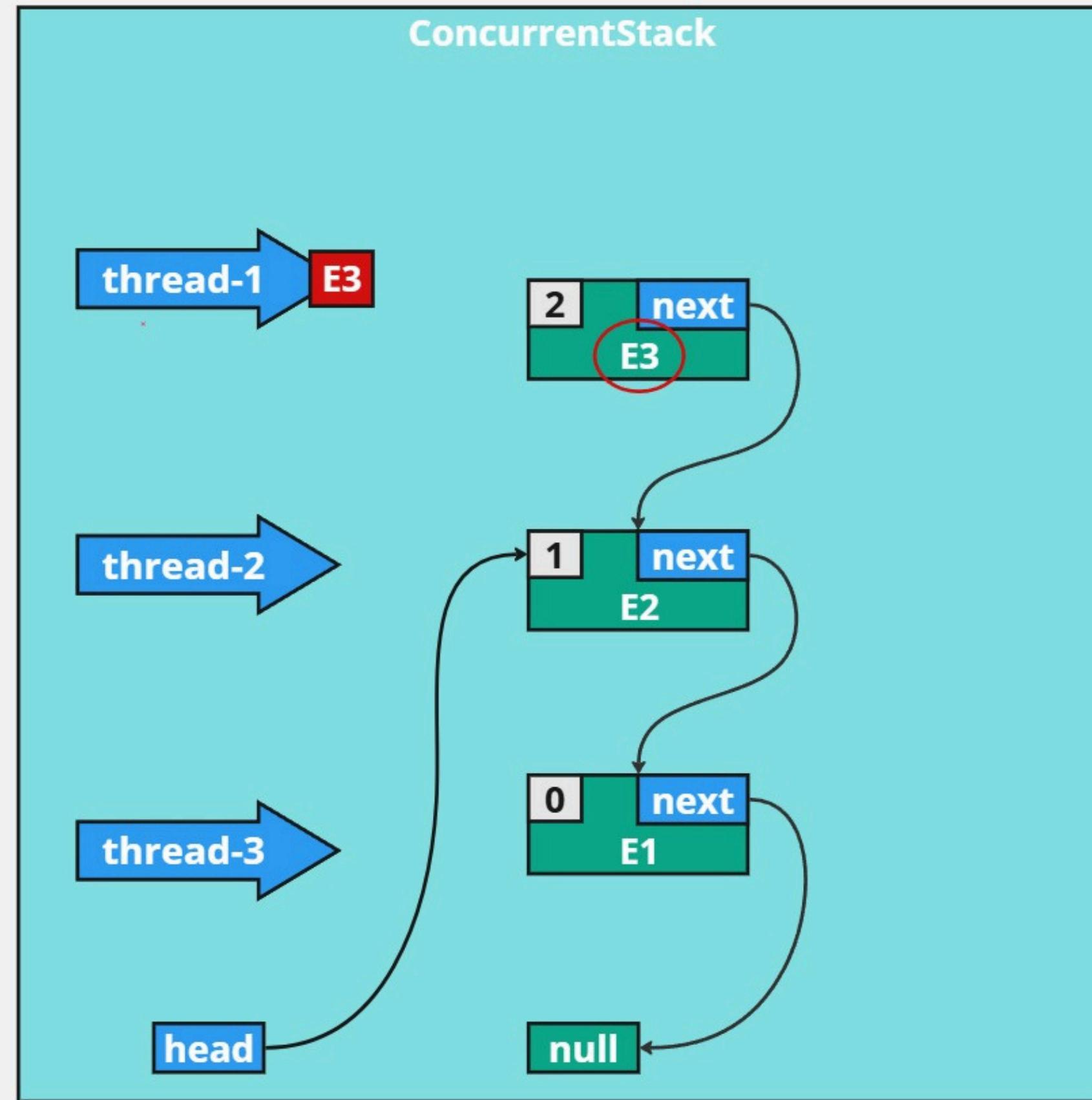
```
public T pop() {
    final Node<T> previousHead = head.get();
    head.set(previousHead.next);
    return previousHead.value;
}
```

2

thread-3

```
public T pop() {
    final Node<T> previousHead = head.get();
    head.set(previousHead.next);
    return previousHead.value;
}
```

2



thread-1

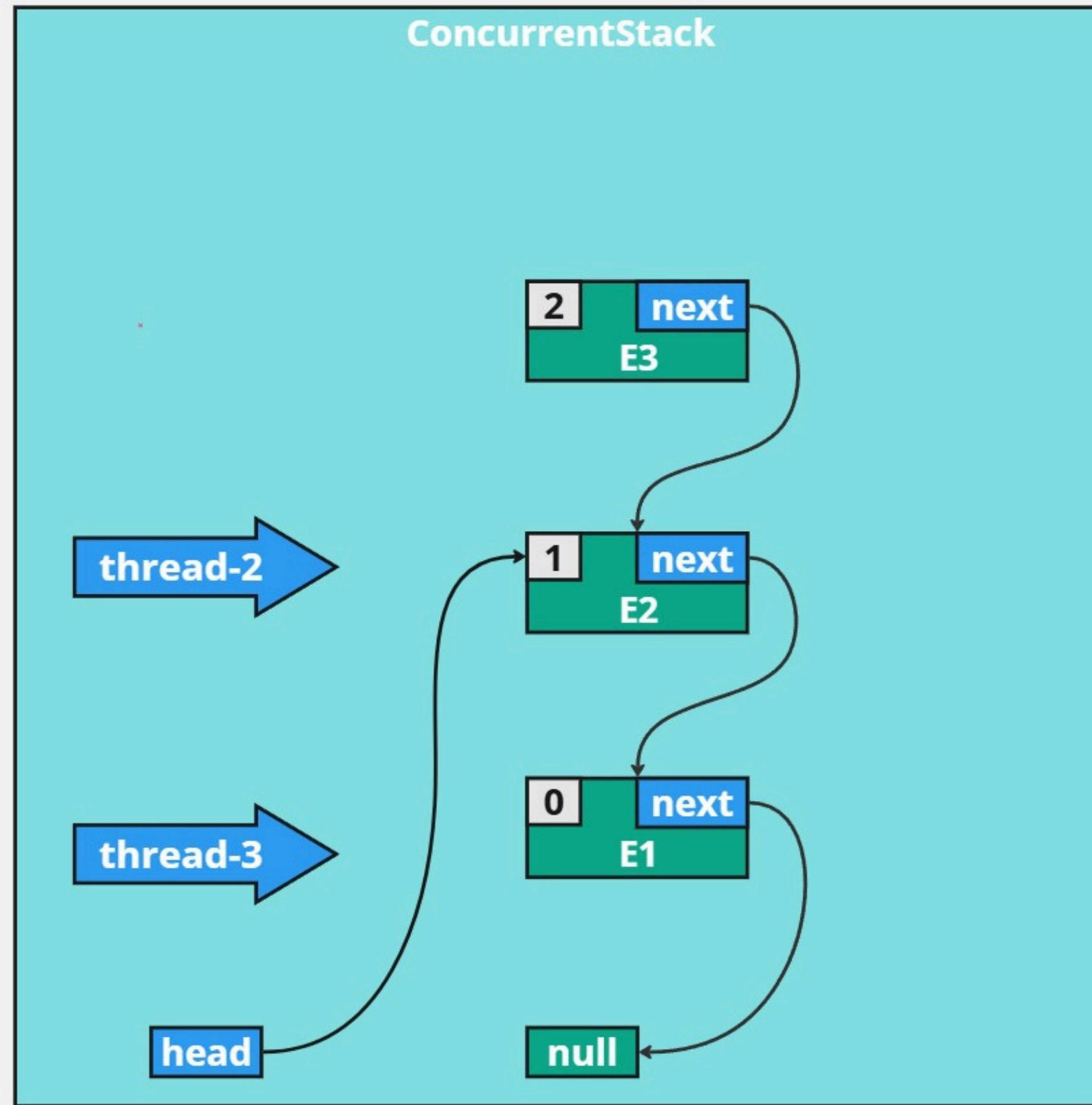
```
public T pop() {
    final Node<T> previousHead = head.get();
    head.set(previousHead.next);
    return previousHead.value;
}
```

thread-2

```
public T pop() {
    final Node<T> previousHead = head.get();
    head.set(previousHead.next);
    return previousHead.value;
}
```

thread-3

```
public T pop() {
    final Node<T> previousHead = head.get();
    head.set(previousHead.next);
    return previousHead.value;
}
```



thread-1

```

public T pop() {
    final Node<T> previousHead = head.get();
    head.set(previousHead.next);
    return previousHead.value;
}
  
```

thread-2

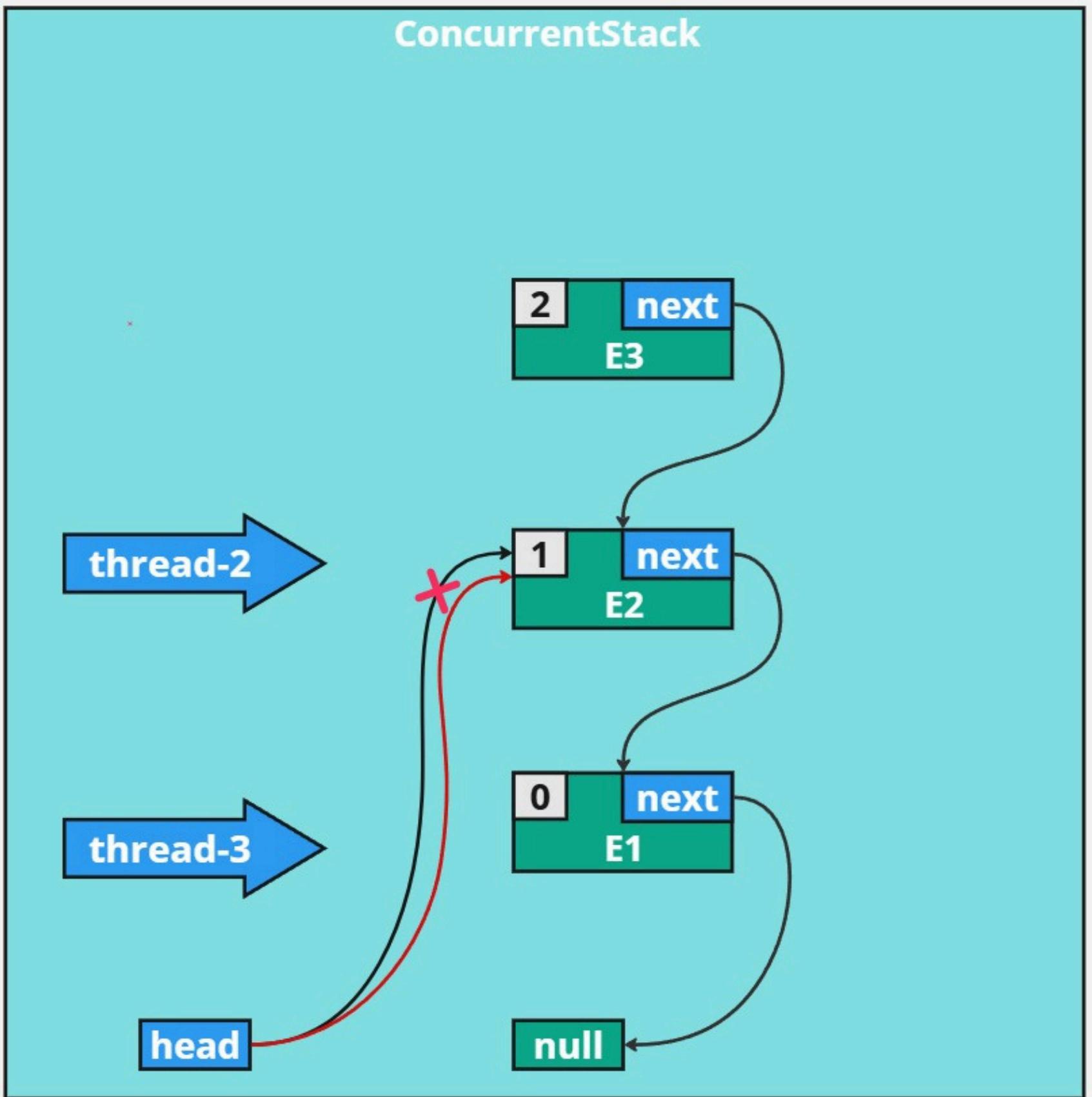
```

public T pop() {
    final Node<T> previousHead = head.get();
    head.set(previousHead.next);
    return previousHead.value;
}
  
```

thread-3

```

public T pop() {
    final Node<T> previousHead = head.get();
    head.set(previousHead.next);
    return previousHead.value;
}
  
```



thread-1

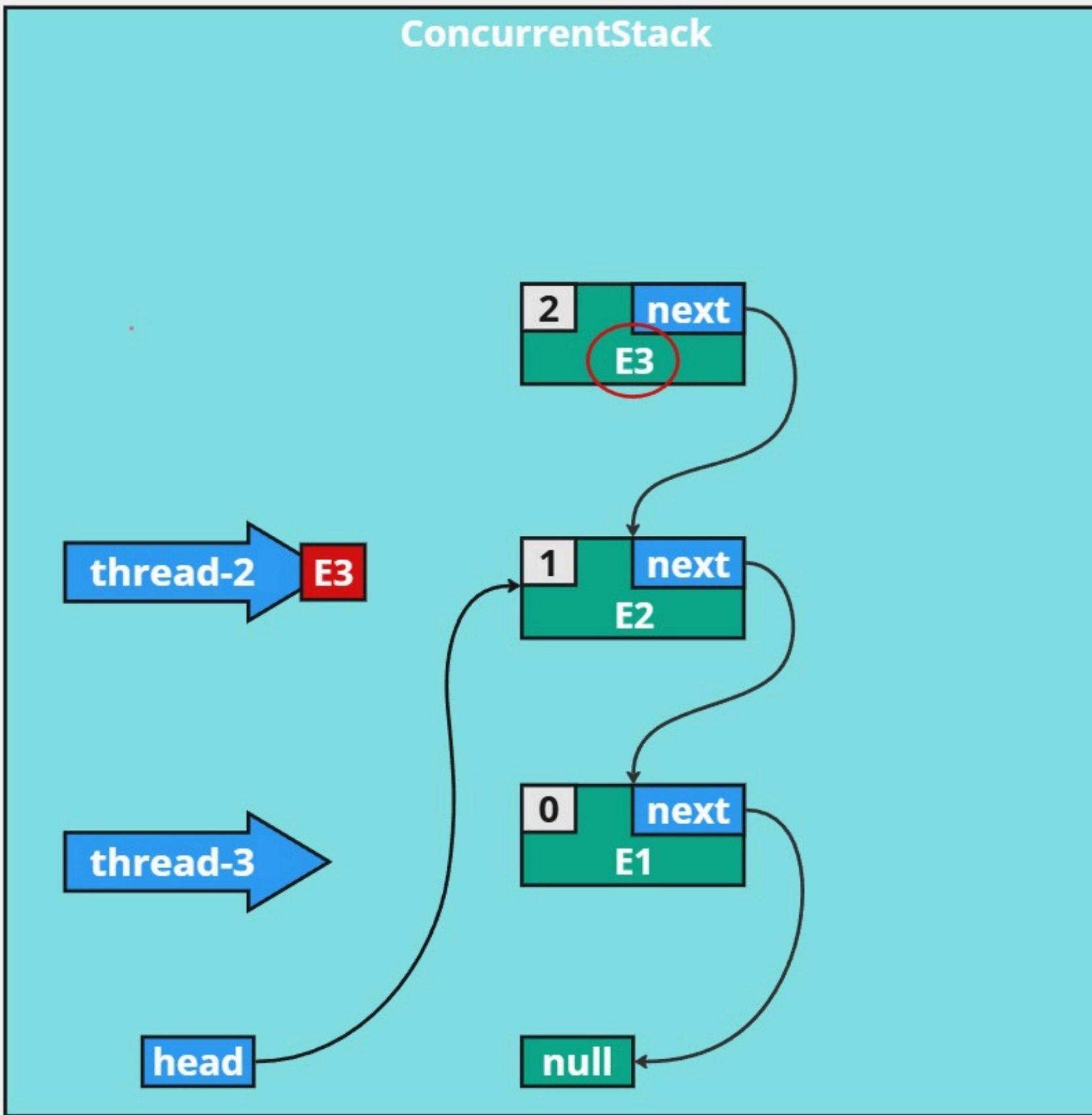
```
public T pop() {
    final Node<T> previousHead = head.get();
    head.set(previousHead.next);
    return previousHead.value;
}
```

thread-2

```
public T pop() {
    final Node<T> previousHead = head.get();
    head.set(previousHead.next); -----
    return previousHead.value;
}
```

thread-3

```
public T pop() {
    final Node<T> previousHead = head.get();
    head.set(previousHead.next); -----
    return previousHead.value;
}
```



thread-1

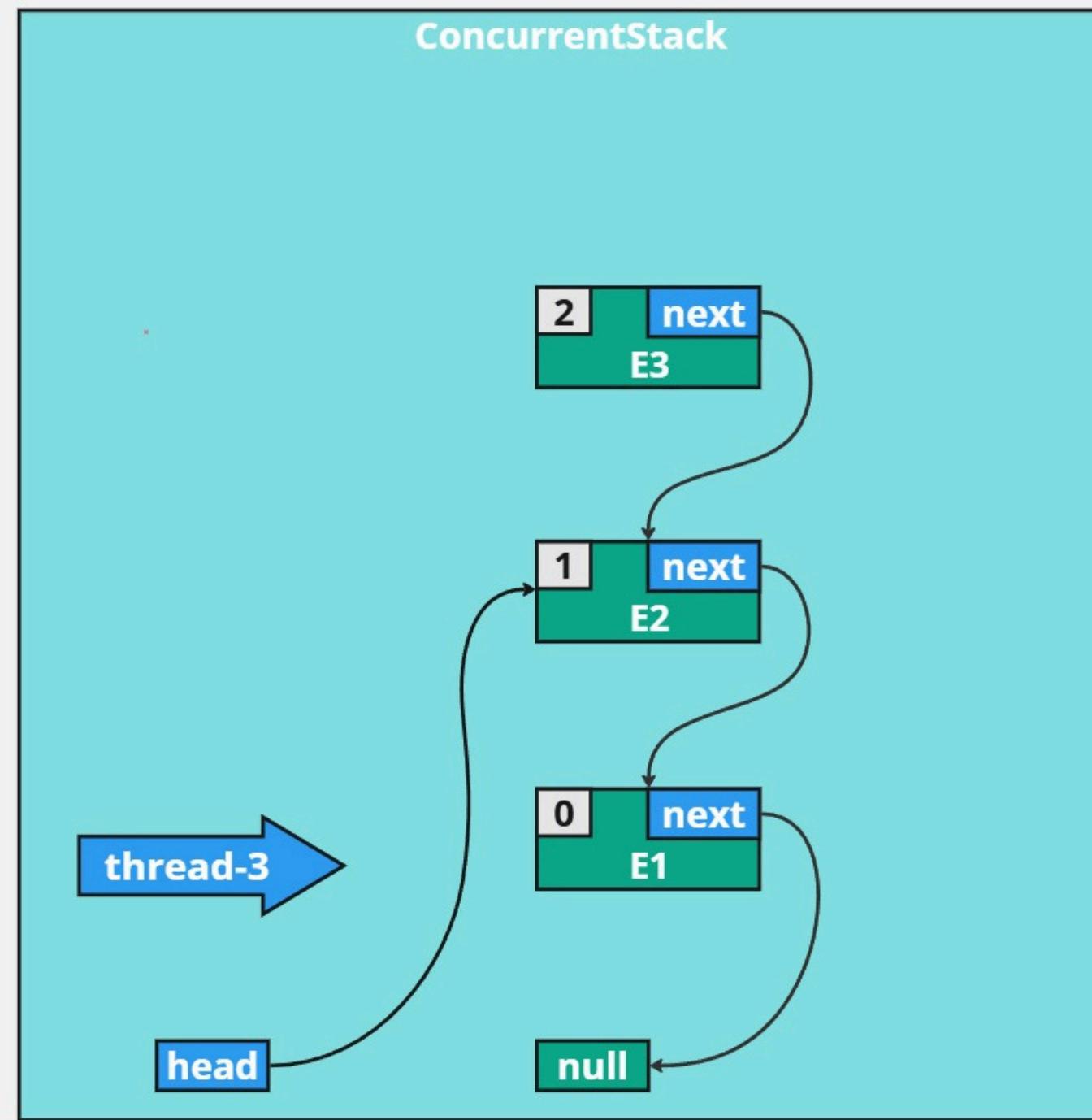
```
public T pop() {
    final Node<T> previousHead = head.get();
    head.set(previousHead.next);
    return previousHead.value;
}
```

thread-2

```
public T pop() {
    final Node<T> previousHead = head.get();
    head.set(previousHead.next);
    return previousHead.value;
}
```

thread-3

```
public T pop() {
    final Node<T> previousHead = head.get();
    head.set(previousHead.next);
    return previousHead.value;
}
```



thread-1 → E3

thread-1

```

public T pop() {
    final Node<T> previousHead = head.get();
    head.set(previousHead.next);
    return previousHead.value;
}
  
```

thread-2 → E3

thread-2

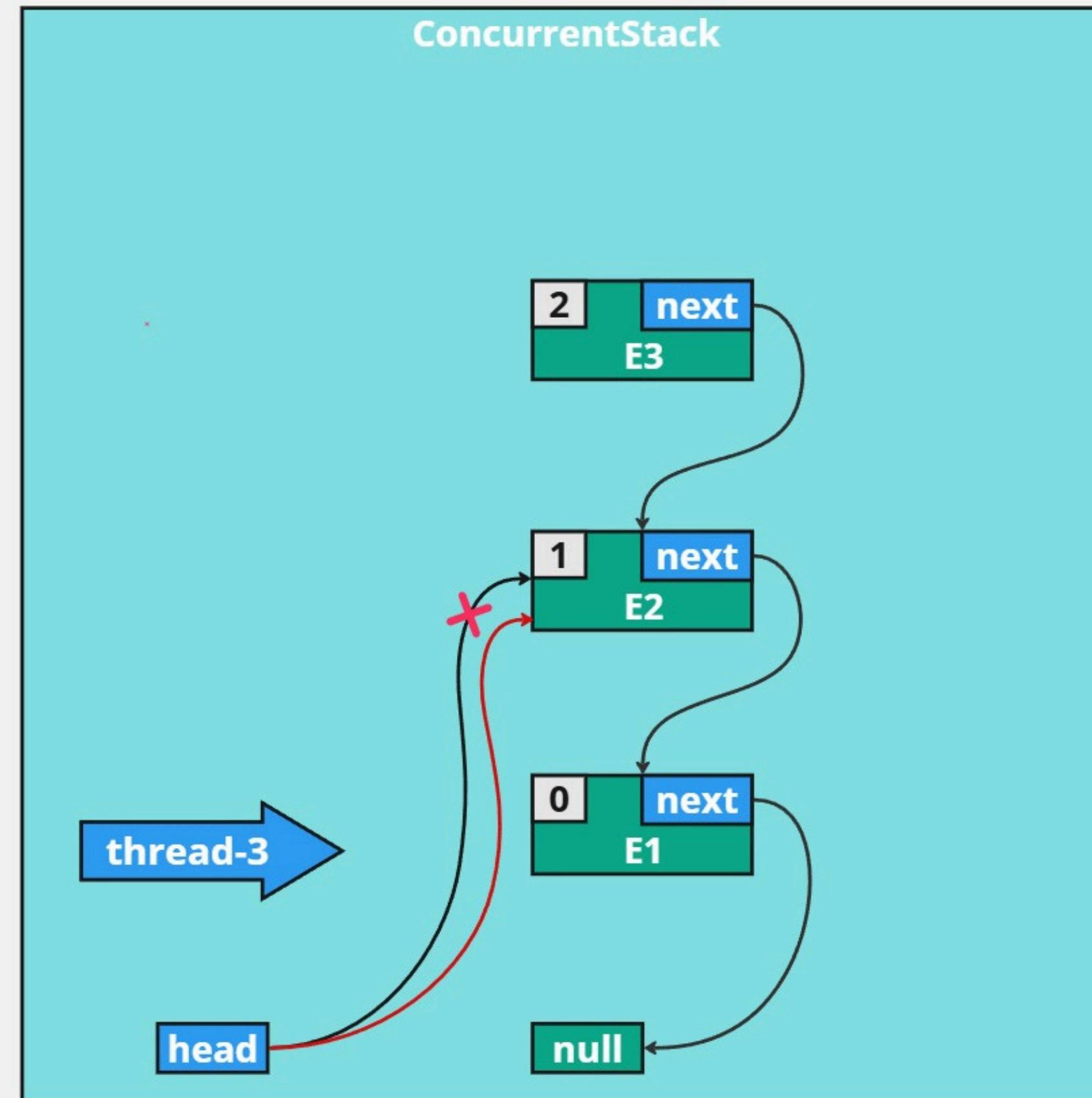
```

public T pop() {
    final Node<T> previousHead = head.get();
    head.set(previousHead.next);
    return previousHead.value;
}
  
```

thread-3

```

public T pop() {
    final Node<T> previousHead = head.get();
    head.set(previousHead.next);
    return previousHead.value;
}
  
```



thread-1

```
public T pop() {
    final Node<T> previousHead = head.get();
    head.set(previousHead.next);
    return previousHead.value;
}
```



thread-2

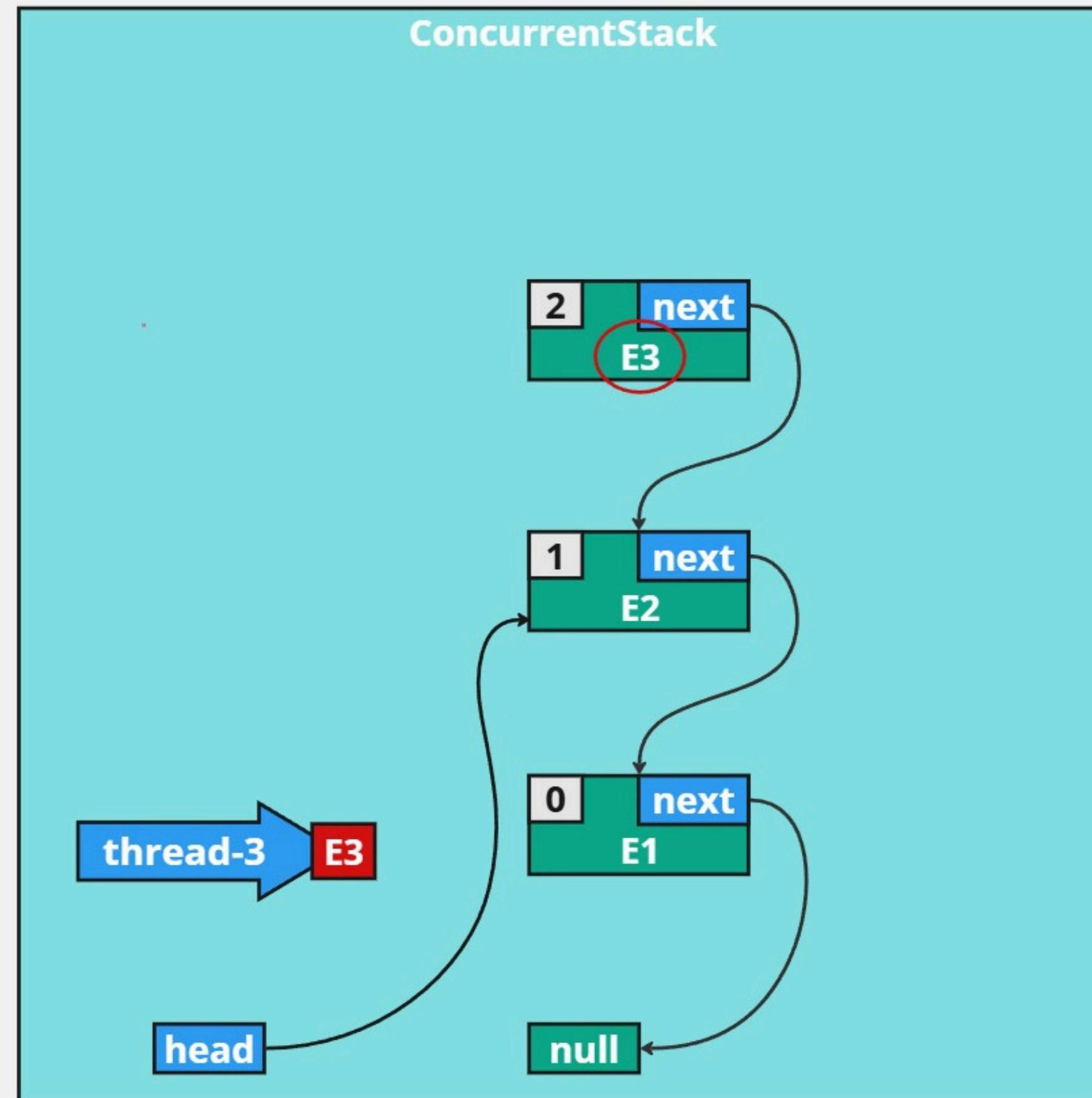
```
public T pop() {
    final Node<T> previousHead = head.get();
    head.set(previousHead.next);
    return previousHead.value;
}
```



thread-3

```
public T pop() {
    final Node<T> previousHead = head.get();
    head.set(previousHead.next);
    return previousHead.value;
}
```

2



thread-1

```
public T pop() {
    final Node<T> previousHead = head.get();
    head.set(previousHead.next);
    return previousHead.value;
}
```

thread-1 → E3

thread-2

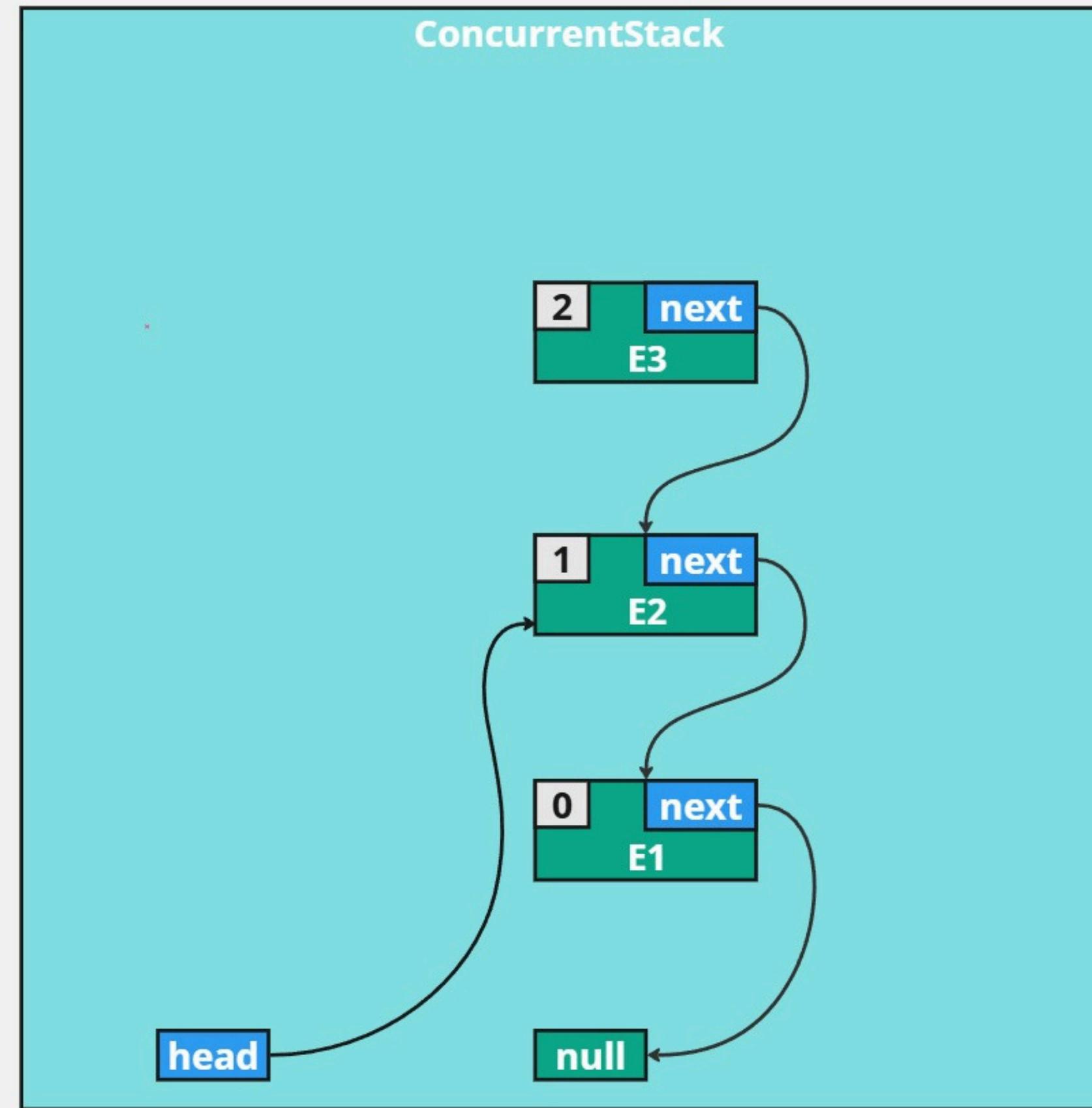
```
public T pop() {
    final Node<T> previousHead = head.get();
    head.set(previousHead.next);
    return previousHead.value;
}
```

thread-2 → E3

thread-3

```
public T pop() {
    final Node<T> previousHead = head.get();
    head.set(previousHead.next);
    return previousHead.value;
}
```

2



thread-1

```
public T pop() {
    final Node<T> previousHead = head.get();
    head.set(previousHead.next);
    return previousHead.value;
}
```

thread-2

```
public T pop() {
    final Node<T> previousHead = head.get();
    head.set(previousHead.next);
    return previousHead.value;
}
```

thread-3

```
public T pop() {
    final Node<T> previousHead = head.get();
    head.set(previousHead.next);
    return previousHead.value;
}
```

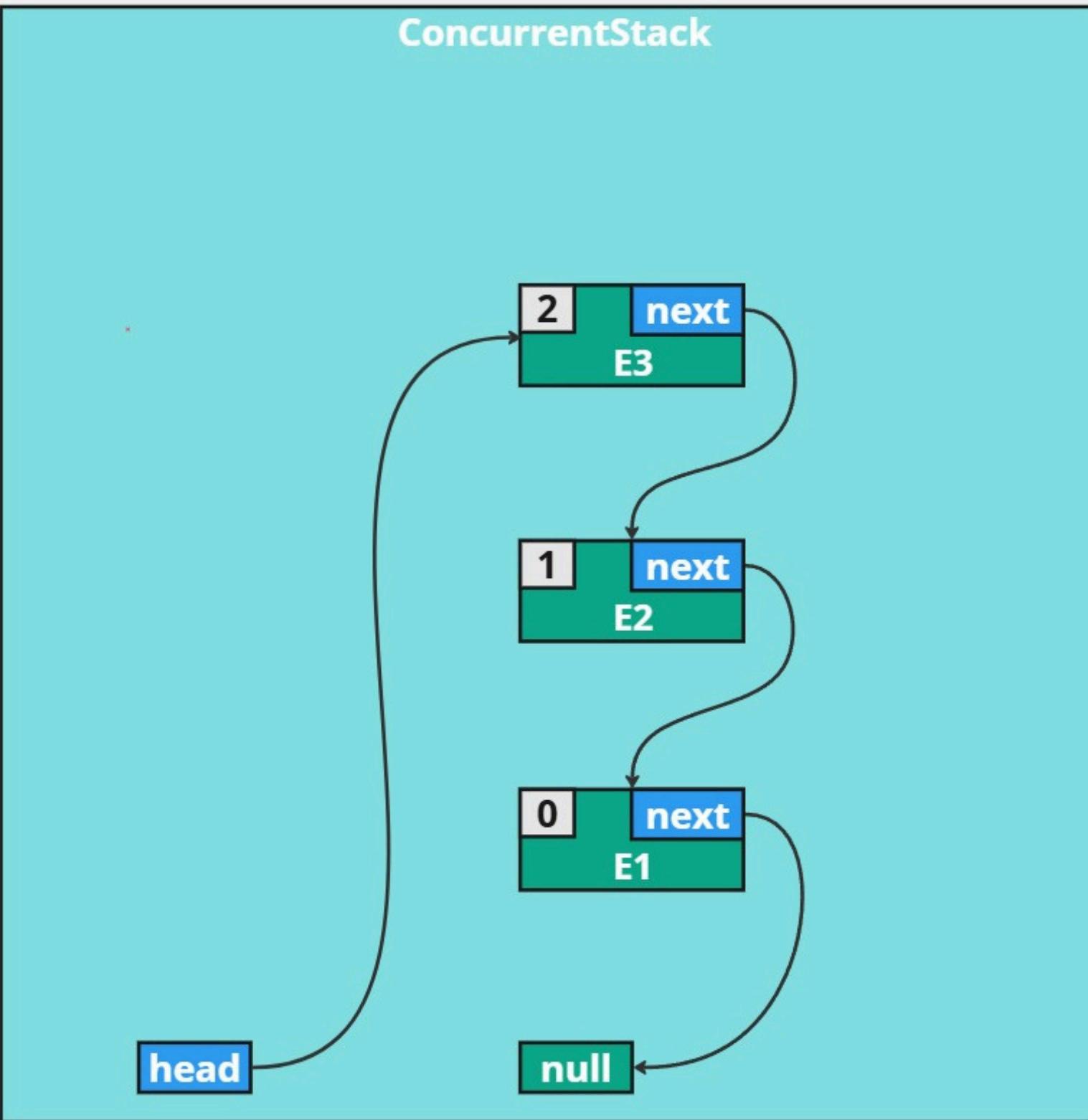
A blue arrow labeled 'thread-1' points to the 'E3' node. Another blue arrow labeled 'thread-2' points to the 'E3' node. A third blue arrow labeled 'thread-3' points to the 'E3' node. A dashed line with a box labeled '2' connects the 'thread-3' arrow to the 'return previousHead.value;' line in the code.

```
public T pop() {  
    final Node<T> previousHead = head.get();  
    head.set(previousHead.next);  
    return previousHead.value;  
}
```



```
public T pop() {  
    while (true) {  
        final Node<T> previousHead = head.get();  
        if (head.compareAndSet(previousHead, previousHead.next)) {  
            return previousHead.value;  
        }  
    }  
}
```

thread-1 →
thread-2 →
thread-3 →



thread-1

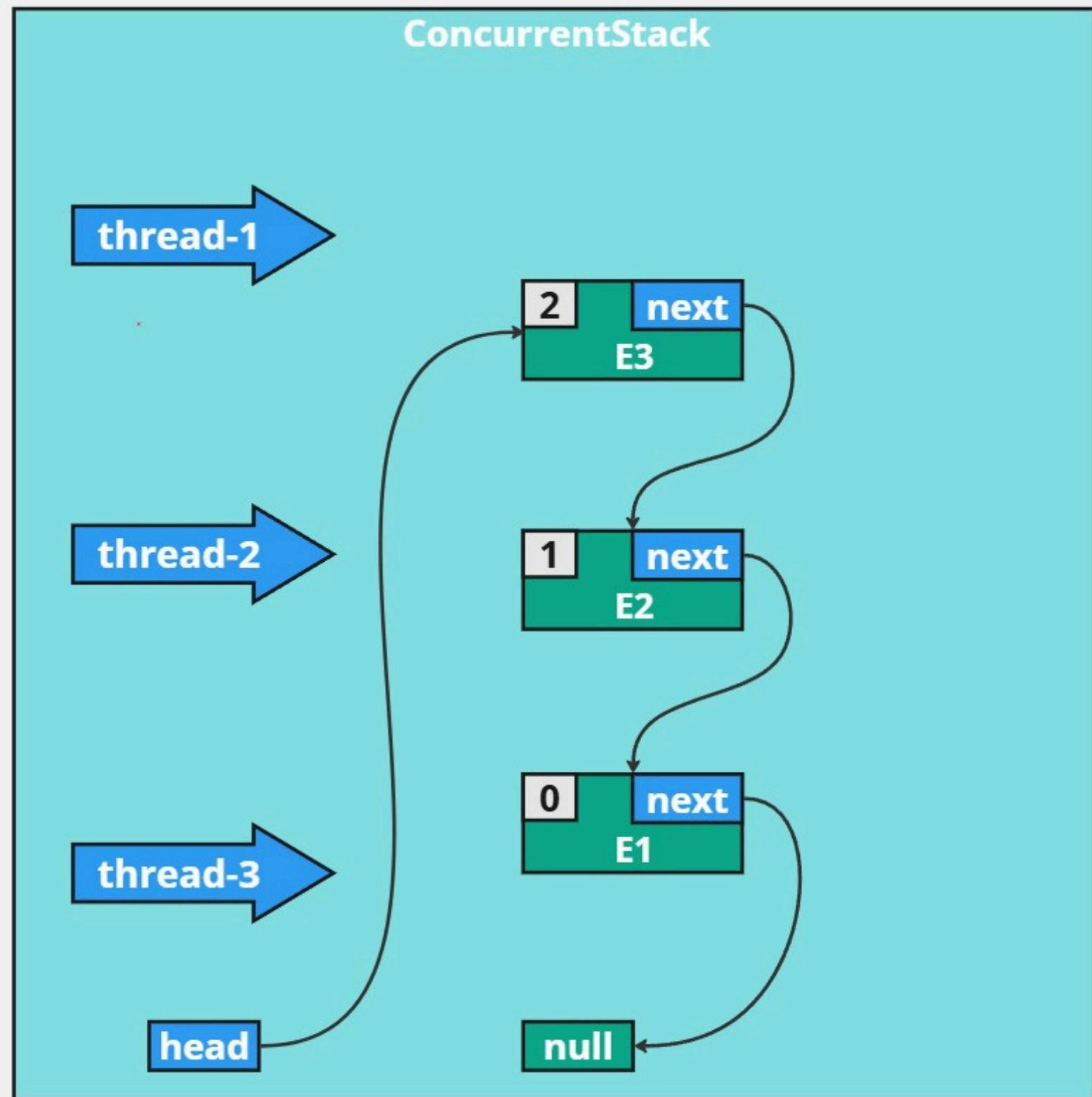
```
public T pop() {  
    while (true) {  
        final Node<T> previousHead = head.get();  
        if (head.compareAndSet(previousHead, previousHead.next)) {  
            return previousHead.value;  
        }  
    }  
}
```

thread-2

```
public T pop() {  
    while (true) {  
        final Node<T> previousHead = head.get();  
        if (head.compareAndSet(previousHead, previousHead.next)) {  
            return previousHead.value;  
        }  
    }  
}
```

thread-3

```
public T pop() {  
    while (true) {  
        final Node<T> previousHead = head.get();  
        if (head.compareAndSet(previousHead, previousHead.next)) {  
            return previousHead.value;  
        }  
    }  
}
```



thread-1

```
public T pop() {
    while (true) {
        final Node<T> previousHead = head.get();
        if (head.compareAndSet(previousHead, previousHead.next)) {
            return previousHead.value;
        }
    }
}
```

2

thread-2

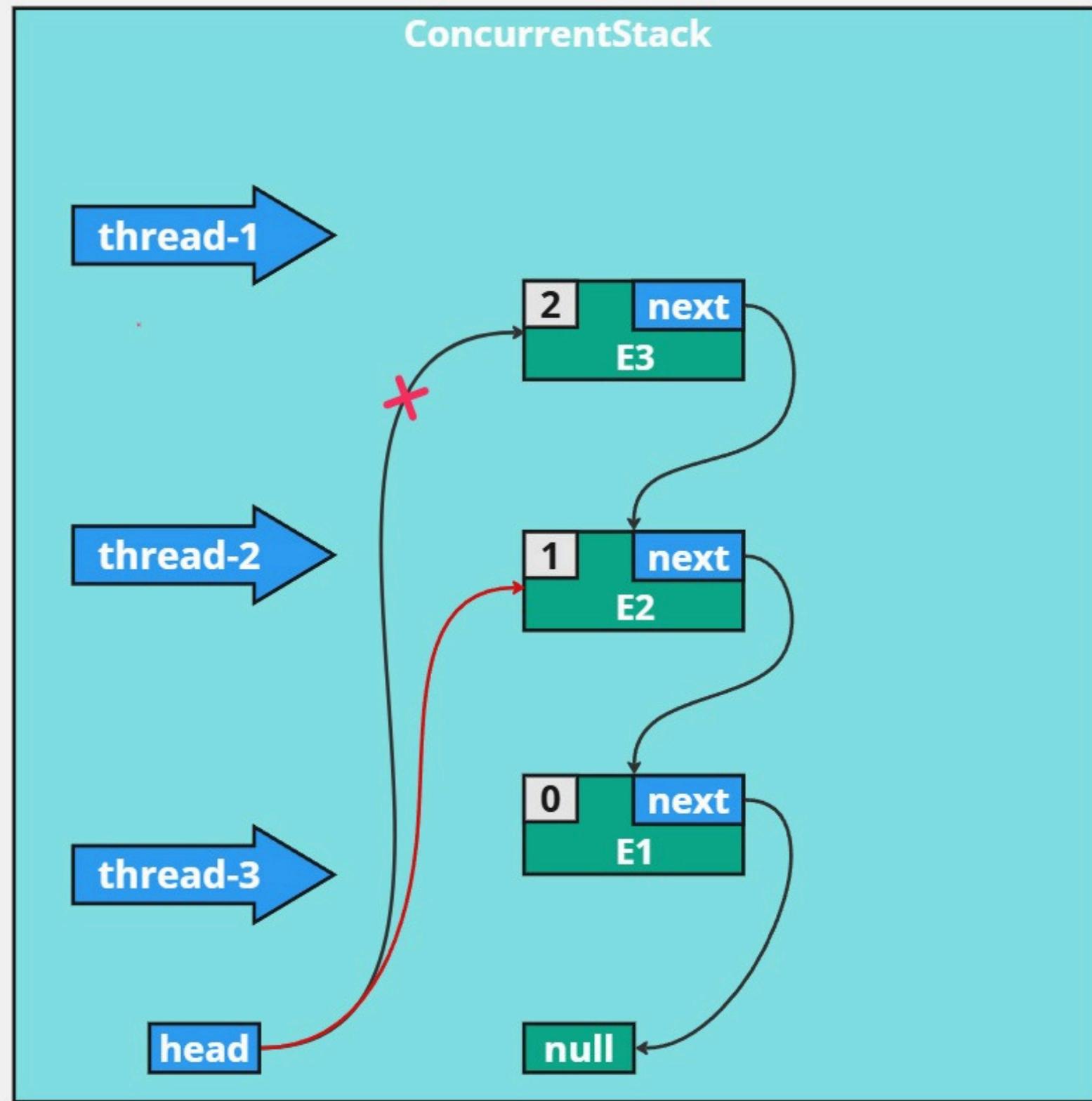
```
public T pop() {
    while (true) {
        final Node<T> previousHead = head.get();
        if (head.compareAndSet(previousHead, previousHead.next)) {
            return previousHead.value;
        }
    }
}
```

2

thread-3

```
public T pop() {
    while (true) {
        final Node<T> previousHead = head.get();
        if (head.compareAndSet(previousHead, previousHead.next)) {
            return previousHead.value;
        }
    }
}
```

2



thread-1

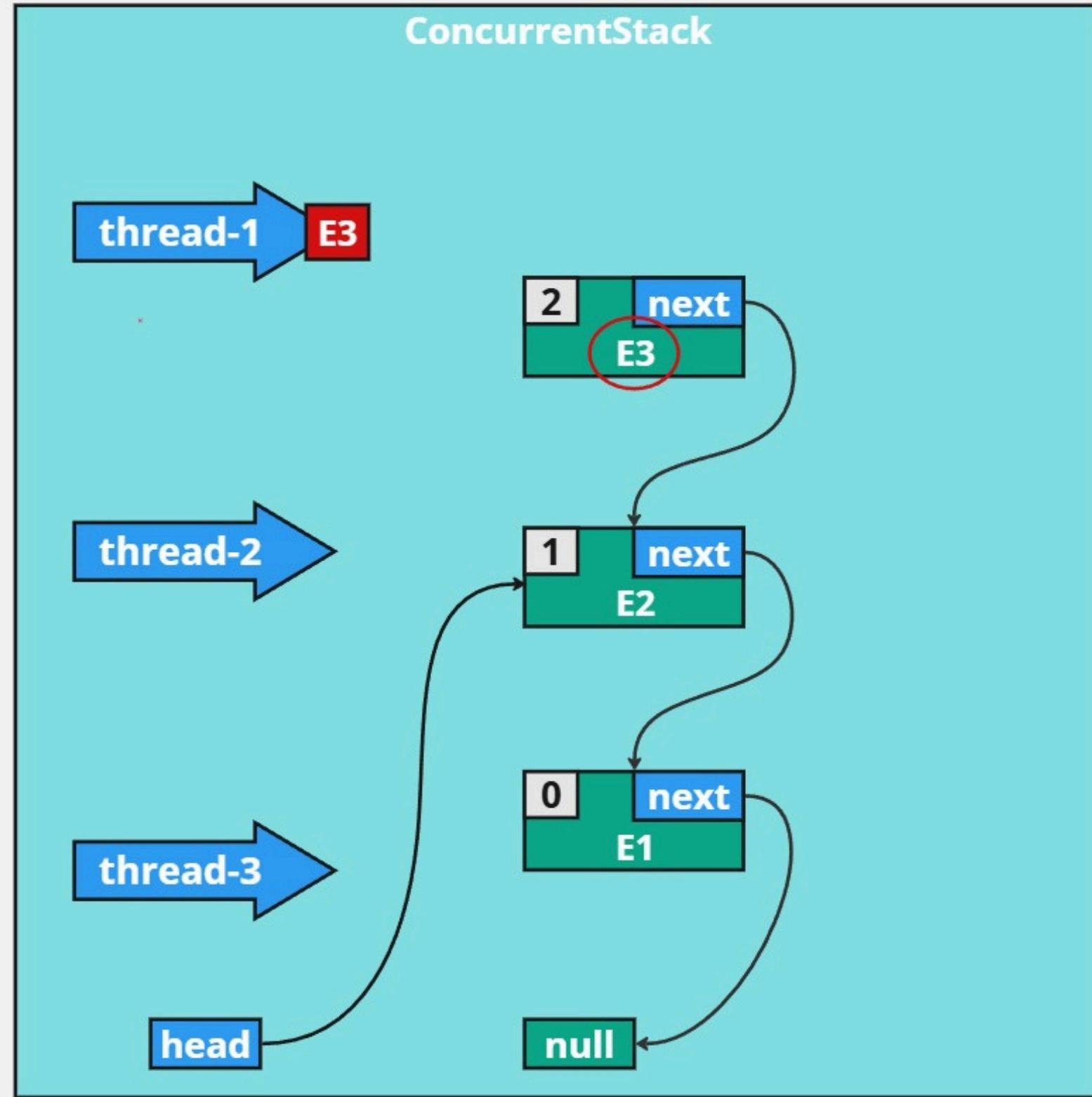
```
public T pop() {
    while (true) {
        final Node<T> previousHead = head.get();
        if (head.compareAndSet(previousHead, previousHead.next)) {
            return previousHead.value;
        }
    }
}
```

thread-2

```
public T pop() {
    while (true) {
        final Node<T> previousHead = head.get();
        if (head.compareAndSet(previousHead, previousHead.next)) {
            return previousHead.value;
        }
    }
}
```

thread-3

```
public T pop() {
    while (true) {
        final Node<T> previousHead = head.get();
        if (head.compareAndSet(previousHead, previousHead.next)) {
            return previousHead.value;
        }
    }
}
```



thread-1

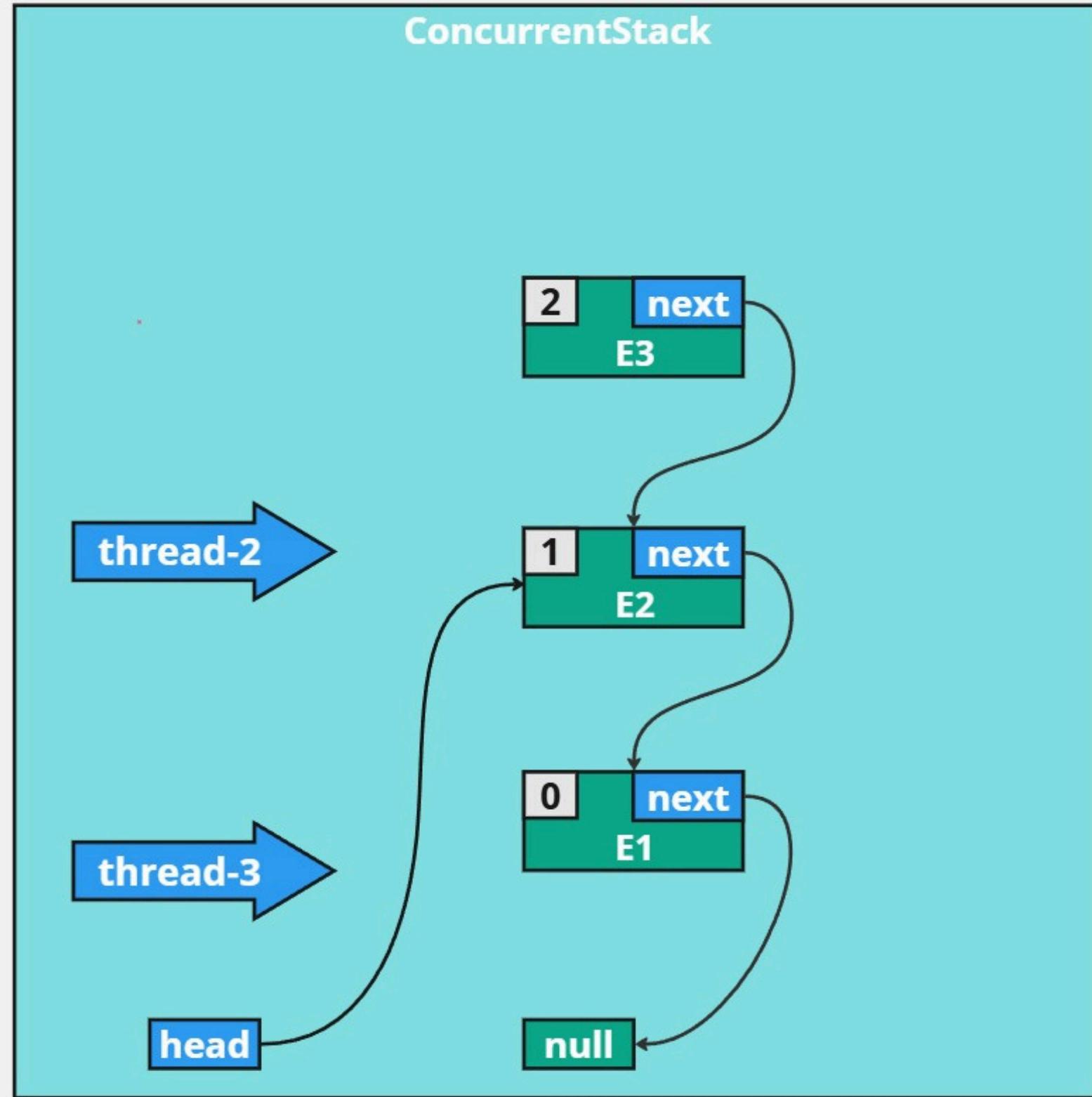
```
public T pop() {
    while (true) {
        final Node<T> previousHead = head.get();
        if (head.compareAndSet(previousHead, previousHead.next)) {
            return previousHead.value;
        }
    }
}
```

thread-2

```
public T pop() {
    while (true) {
        final Node<T> previousHead = head.get();
        if (head.compareAndSet(previousHead, previousHead.next)) {
            return previousHead.value;
        }
    }
}
```

thread-3

```
public T pop() {
    while (true) {
        final Node<T> previousHead = head.get();
        if (head.compareAndSet(previousHead, previousHead.next)) {
            return previousHead.value;
        }
    }
}
```



thread-1 → E3

thread-1

```
public T pop() {
    while (true) {
        final Node<T> previousHead = head.get();
        if (head.compareAndSet(previousHead, previousHead.next)) {
            return previousHead.value;
        }
    }
}
```

2

thread-2 → E2

thread-2

```
public T pop() {
    while (true) {
        final Node<T> previousHead = head.get();
        if (head.compareAndSet(previousHead, previousHead.next)) {
            return previousHead.value;
        }
    }
}
```

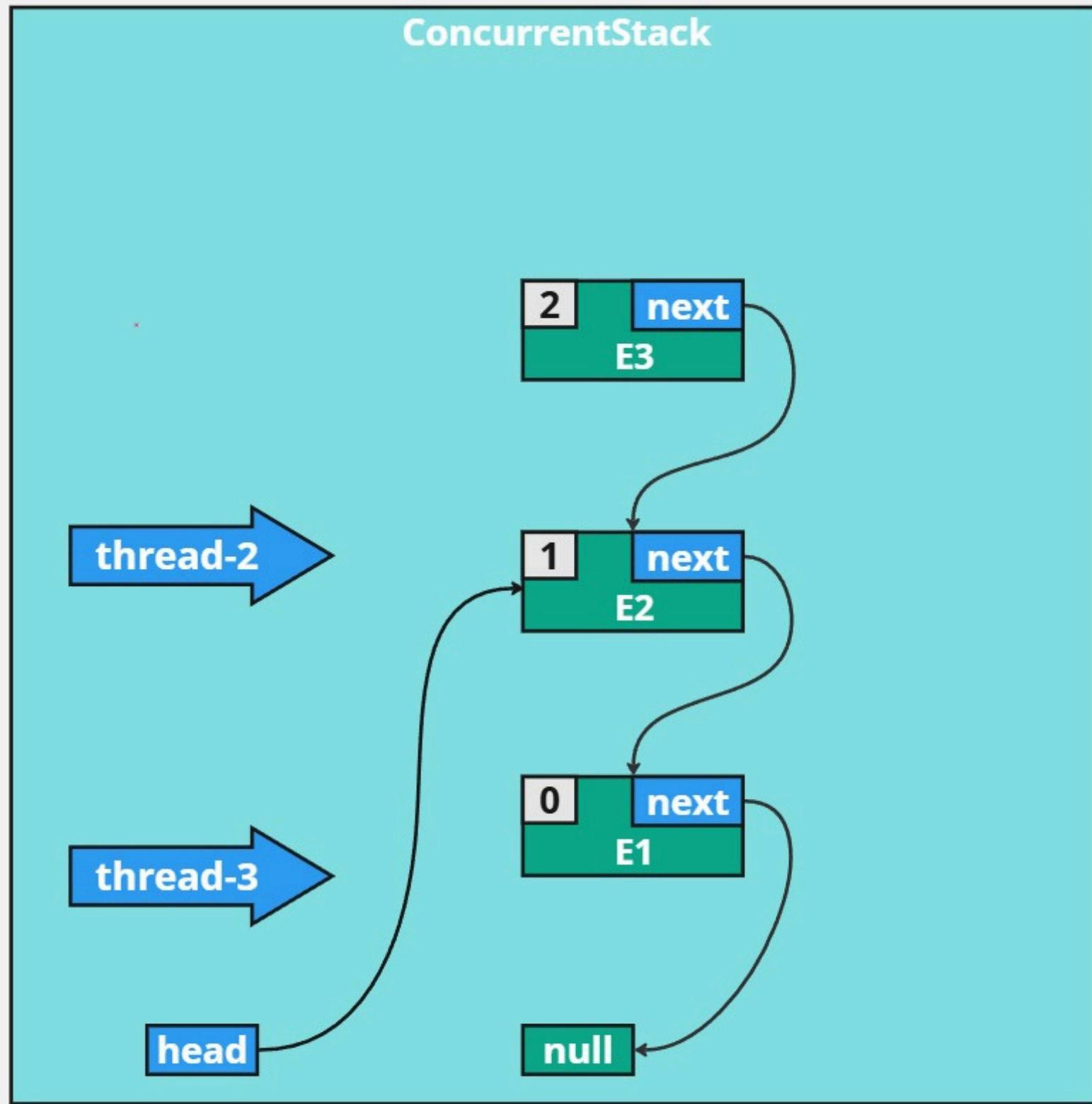
2

thread-3 → E1

thread-3

```
public T pop() {
    while (true) {
        final Node<T> previousHead = head.get();
        if (head.compareAndSet(previousHead, previousHead.next)) {
            return previousHead.value;
        }
    }
}
```

2



```

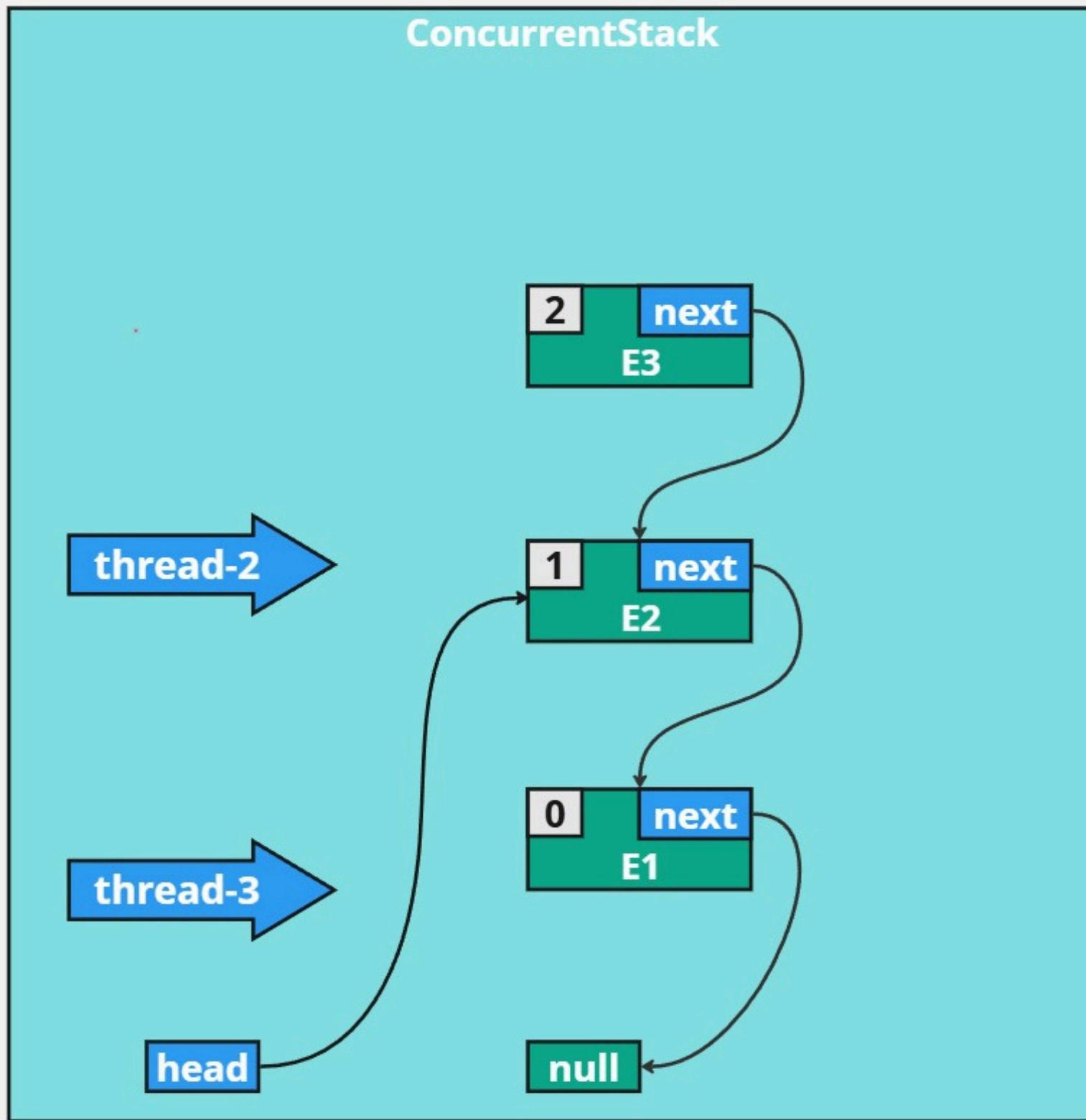
thread-1
public T pop() {
    while (true) {
        final Node<T> previousHead = head.get();
        if (head.compareAndSet(previousHead, previousHead.next)) {
            return previousHead.value;
        }
    }
}

thread-2
public T pop() {
    while (true) {
        final Node<T> previousHead = head.get();
if (head.compareAndSet(previousHead, previousHead.next)) {
        return previousHead.value;
    }
}

thread-3
public T pop() {
    while (true) {
        final Node<T> previousHead = head.get();
if (head.compareAndSet(previousHead, previousHead.next)) {
        return previousHead.value;
    }
}

```

Dashed arrows from the code snippets point to the "next" field assignments in the stack diagram, indicating the target of the compare-and-set operation.



thread-1

```
public T pop() {
    while (true) {
        final Node<T> previousHead = head.get();
        if (head.compareAndSet(previousHead, previousHead.next)) {
            return previousHead.value;
        }
    }
}
```

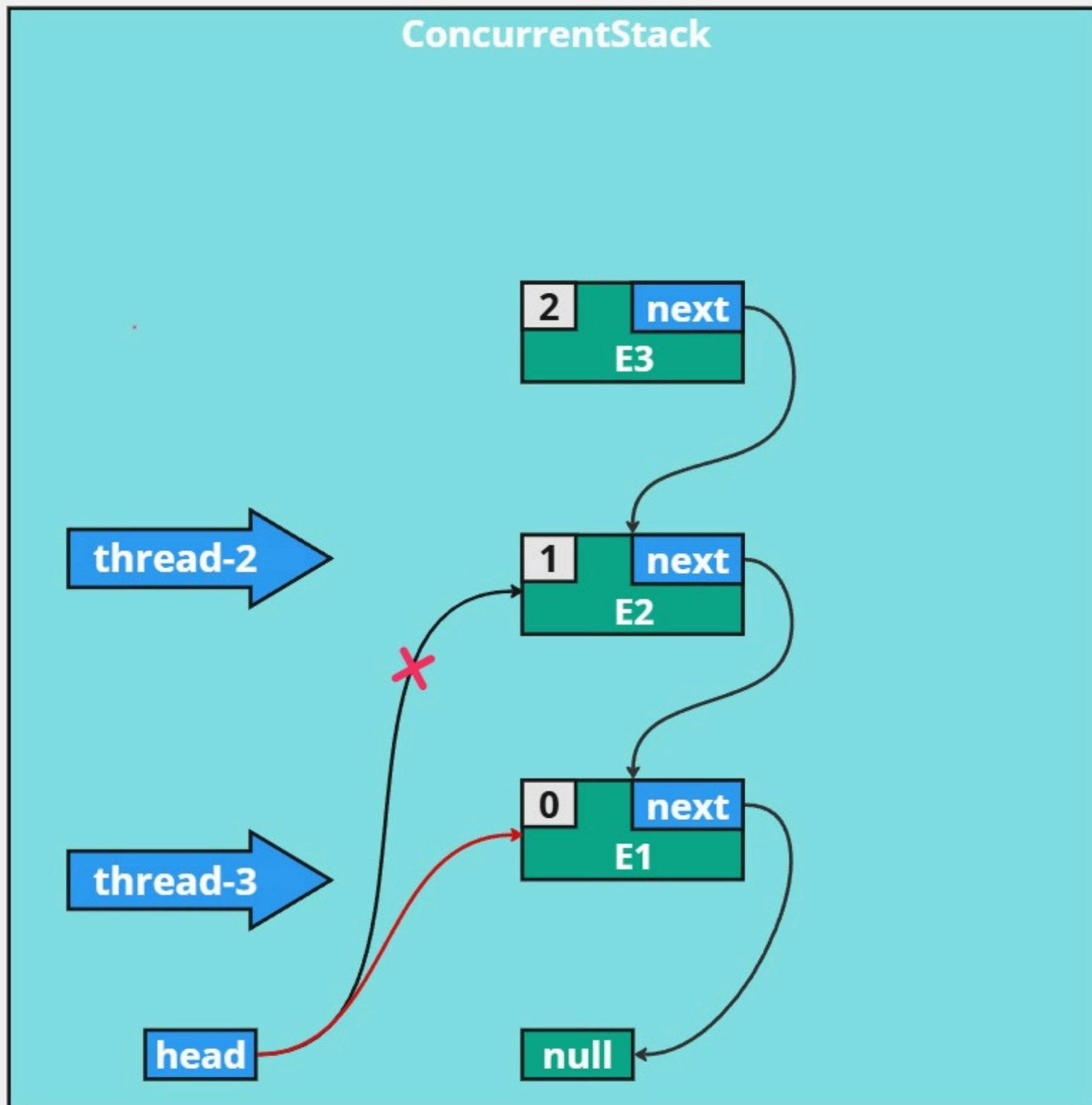
thread-2

```
public T pop() {
    while (true) {
        final Node<T> previousHead = head.get();
        if (head.compareAndSet(previousHead, previousHead.next)) {
            return previousHead.value;
        }
    }
}
```

thread-3

```
public T pop() {
    while (true) {
        final Node<T> previousHead = head.get();
        if (head.compareAndSet(previousHead, previousHead.next)) {
            return previousHead.value;
        }
    }
}
```

Dashed arrows point from the highlighted code in thread-2 and thread-3 to the corresponding code in thread-1, indicating that both threads are attempting to update the same previousHead variable.



thread-1

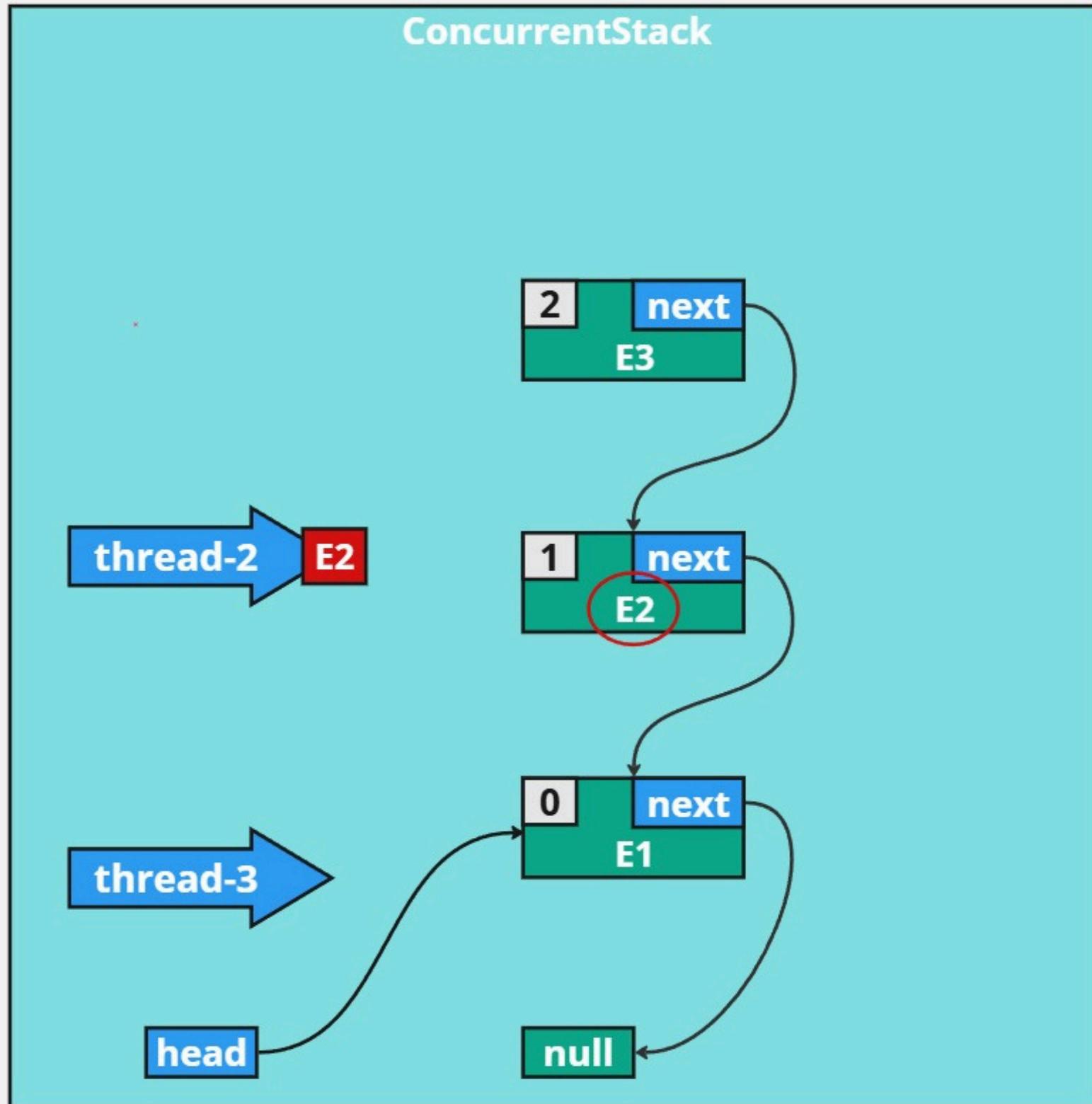
```
public T pop() {
    while (true) {
        final Node<T> previousHead = head.get();
        if (head.compareAndSet(previousHead, previousHead.next)) {
            return previousHead.value;
        }
    }
}
```

thread-2

```
public T pop() {
    while (true) {
        final Node<T> previousHead = head.get();
        if (head.compareAndSet(previousHead, previousHead.next)) {
            return previousHead.value;
        }
    }
}
```

thread-3

```
public T pop() {
    while (true) {
        final Node<T> previousHead = head.get();
        if (head.compareAndSet(previousHead, previousHead.next)) {
            return previousHead.value;
        }
    }
}
```



```

thread-1
public T pop() {
    while (true) {
        final Node<T> previousHead = head.get();
        if (head.compareAndSet(previousHead, previousHead.next)) {
            return previousHead.value;
        }
    }
}

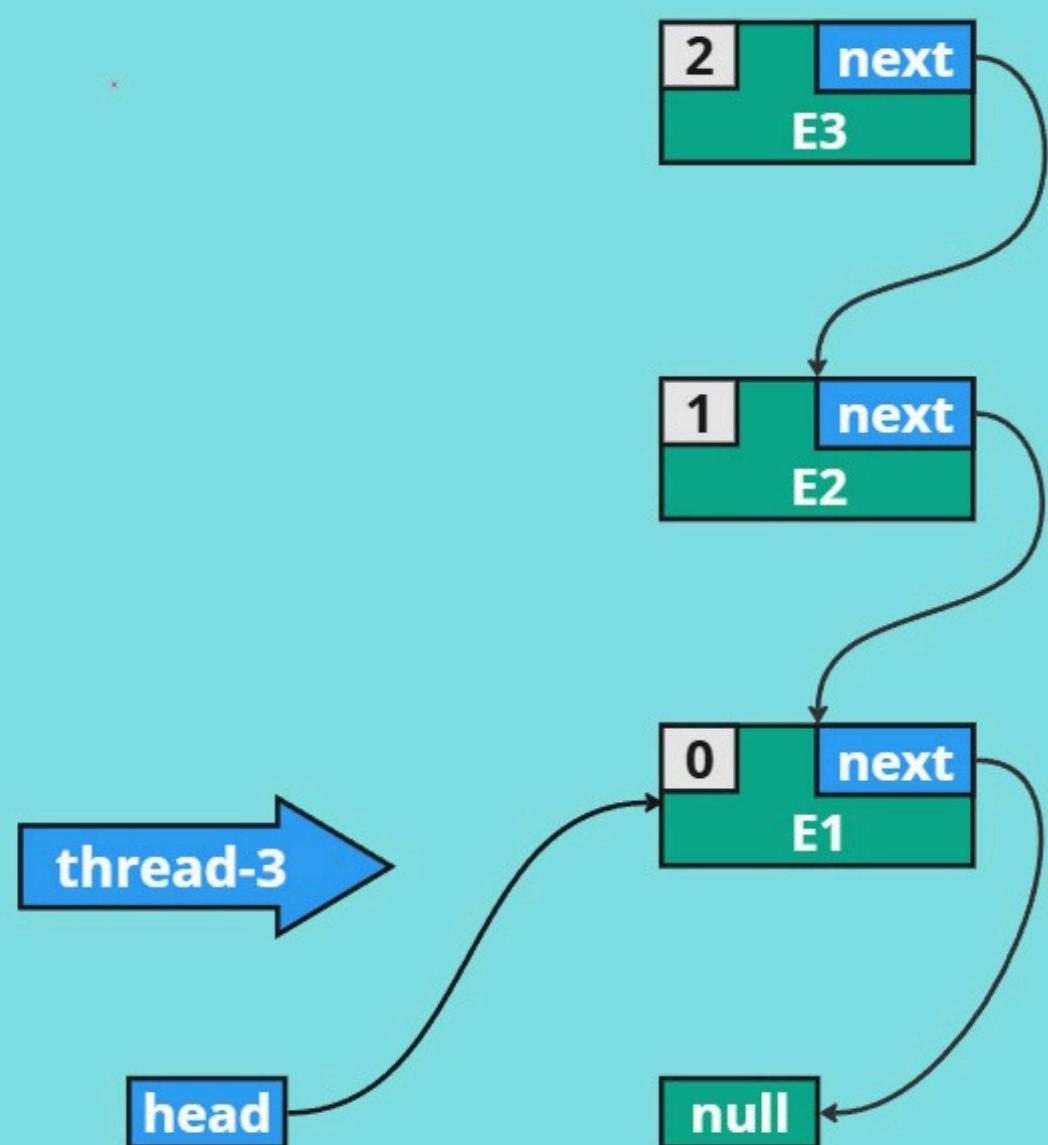
thread-2
public T pop() {
    while (true) {
        final Node<T> previousHead = head.get();
        if (head.compareAndSet(previousHead, previousHead.next)) {
            return previousHead.value;
        }
    }
}

thread-3
public T pop() {
    while (true) {
        final Node<T> previousHead = head.get();
        if (head.compareAndSet(previousHead, previousHead.next)) {
            return previousHead.value;
        }
    }
}

```

Dashed arrows from the code snippets point to the corresponding lines in the stack diagram. In the first snippet, the line `return previousHead.value;` is highlighted with a red box. In the second snippet, the line `return previousHead.value;` is also highlighted with a red box. In the third snippet, the line `if (head.compareAndSet(previousHead, previousHead.next)) {` is highlighted with a red box.

ConcurrentStack



thread-1

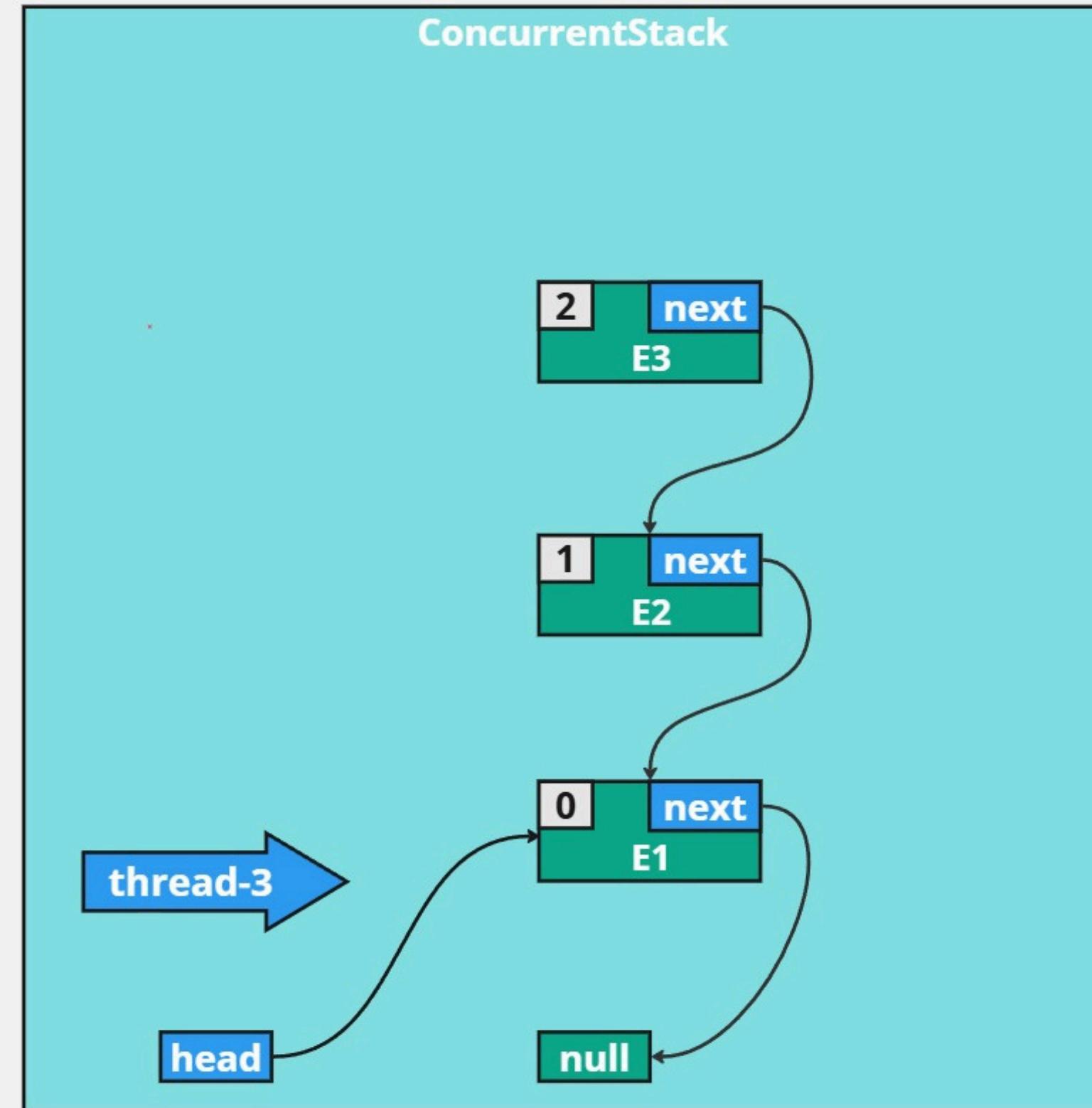
```
public T pop() {  
    while (true) {  
        final Node<T> previousHead = head.get();  
        if (head.compareAndSet(previousHead, previousHead.next)) {  
            return previousHead.value;  
        }  
    }  
}
```

thread-2

```
public T pop() {  
    while (true) {  
        final Node<T> previousHead = head.get();  
        if (head.compareAndSet(previousHead, previousHead.next)) {  
            return previousHead.value;  
        }  
    }  
}
```

thread-3

```
public T pop() {  
    while (true) {  
        final Node<T> previousHead = head.get();  
        if (head.compareAndSet(previousHead, previousHead.next)) {  
            return previousHead.value;  
        }  
    }  
}
```



```

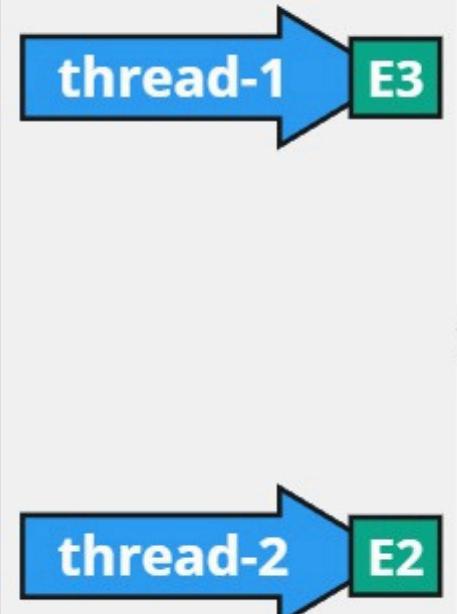
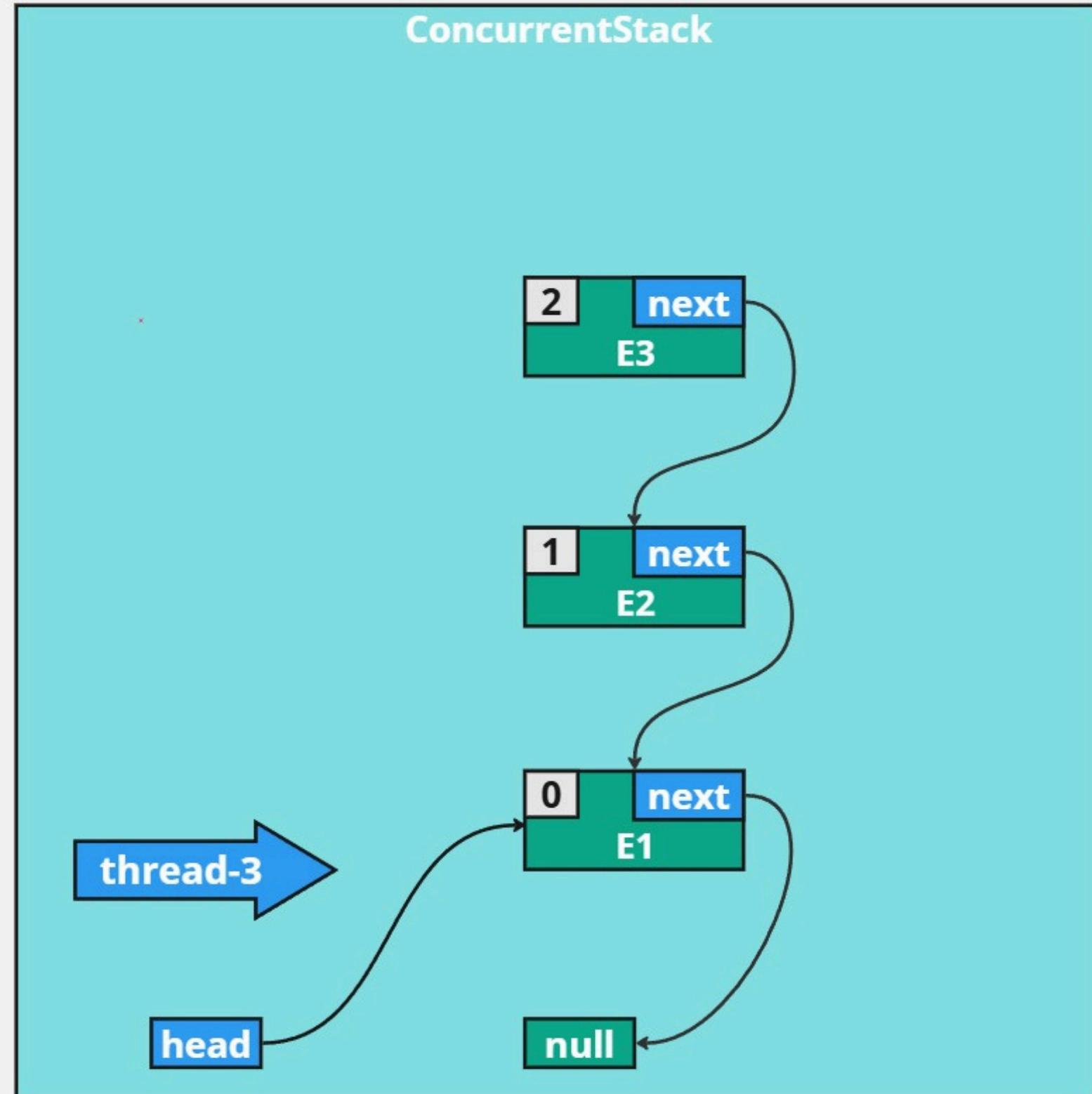
thread-1
public T pop() {
    while (true) {
        final Node<T> previousHead = head.get();
        if (head.compareAndSet(previousHead, previousHead.next)) {
            return previousHead.value;
        }
    }
}

thread-2
public T pop() {
    while (true) {
        final Node<T> previousHead = head.get();
        if (head.compareAndSet(previousHead, previousHead.next)) {
            return previousHead.value;
        }
    }
}

thread-3
public T pop() {
    while (true) {
        final Node<T> previousHead = head.get();
if (head.compareAndSet(previousHead, previousHead.next)) {
            return previousHead.value;
}
    }
}

```

The code snippet shows three threads (thread-1, thread-2, thread-3) performing a 'pop' operation on a concurrent stack. Each thread uses a compare-and-set (CAS) operation to update the 'head' pointer. In thread-3, the condition of the if-statement is highlighted in red, indicating that the code is being executed. A dashed arrow points from the end of the if-statement back to the value '1' at the top right, suggesting that the value returned by the 'pop' operation is 1.



thread-1

```
public T pop() {
    while (true) {
        final Node<T> previousHead = head.get();
        if (head.compareAndSet(previousHead, previousHead.next)) {
            return previousHead.value;
        }
    }
}
```

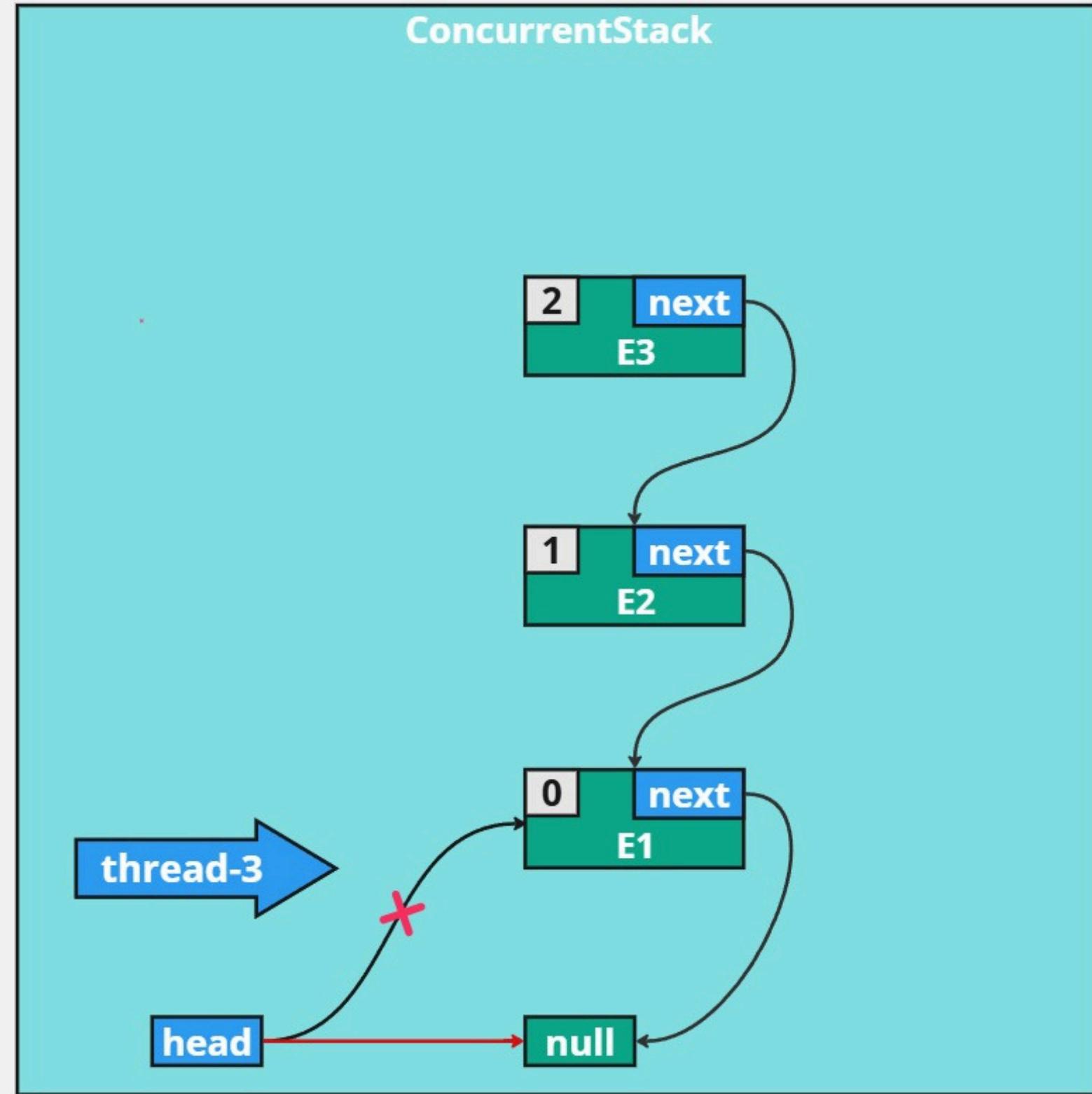
thread-2

```
public T pop() {
    while (true) {
        final Node<T> previousHead = head.get();
        if (head.compareAndSet(previousHead, previousHead.next)) {
            return previousHead.value;
        }
    }
}
```

thread-3

```
public T pop() {
    while (true) {
        final Node<T> previousHead = head.get(); // Line 1
        if (head.compareAndSet(previousHead, previousHead.next)) {
            return previousHead.value;
        }
    }
}
```

0



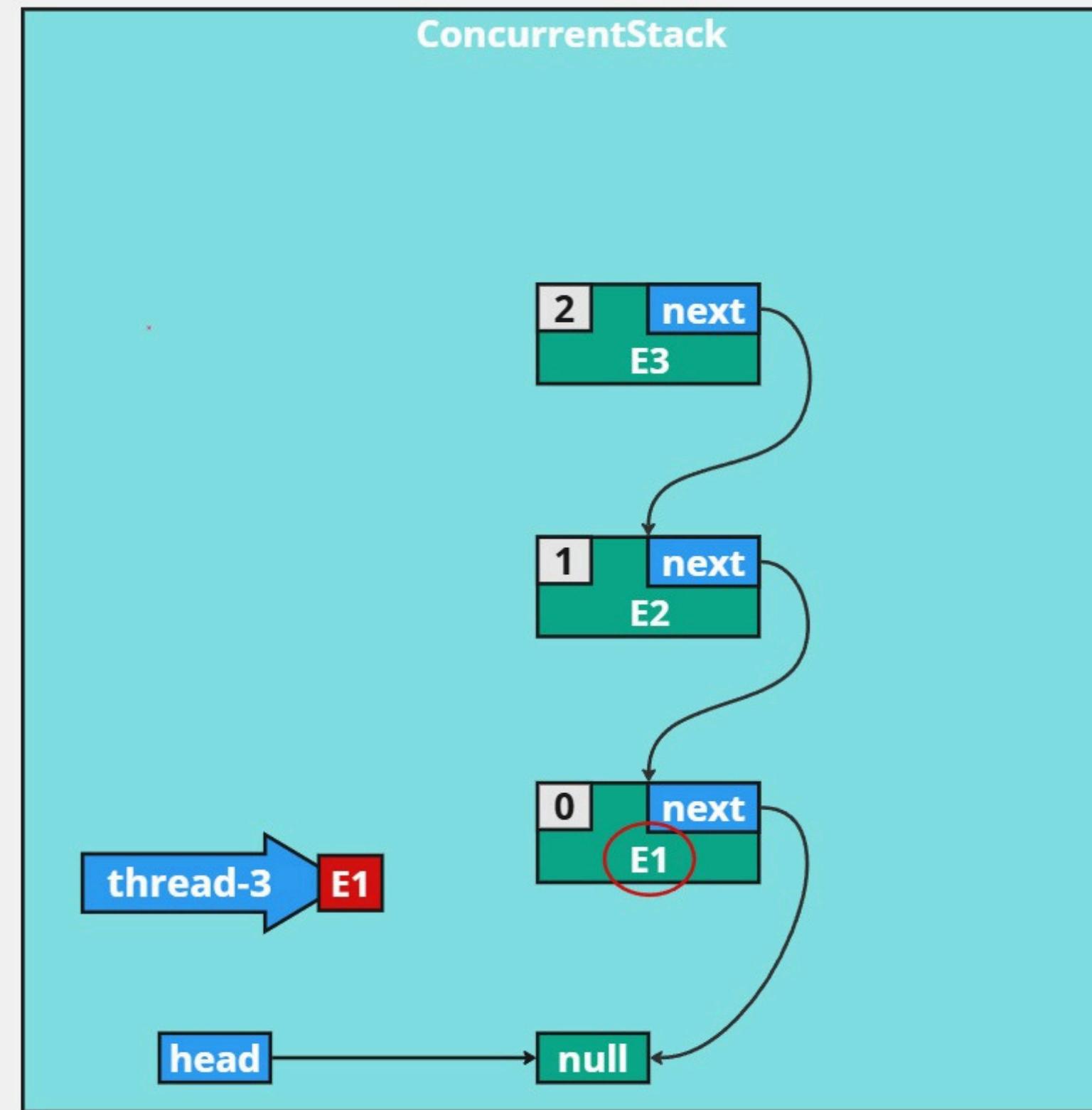
```

thread-1
public T pop() {
    while (true) {
        final Node<T> previousHead = head.get();
        if (head.compareAndSet(previousHead, previousHead.next)) {
            return previousHead.value;
        }
    }
}

thread-2
public T pop() {
    while (true) {
        final Node<T> previousHead = head.get();
        if (head.compareAndSet(previousHead, previousHead.next)) {
            return previousHead.value;
        }
    }
}

thread-3
public T pop() {
    while (true) {
        final Node<T> previousHead = head.get();
if (head.compareAndSet(previousHead, previousHead.next)) {
            return previousHead.value;
        }
    }
}

```



thread-1 → E3

thread-2 → E2

thread-3 → E1

thread-1

```
public T pop() {
    while (true) {
        final Node<T> previousHead = head.get();
        if (head.compareAndSet(previousHead, previousHead.next)) {
            return previousHead.value;
        }
    }
}
```

thread-2

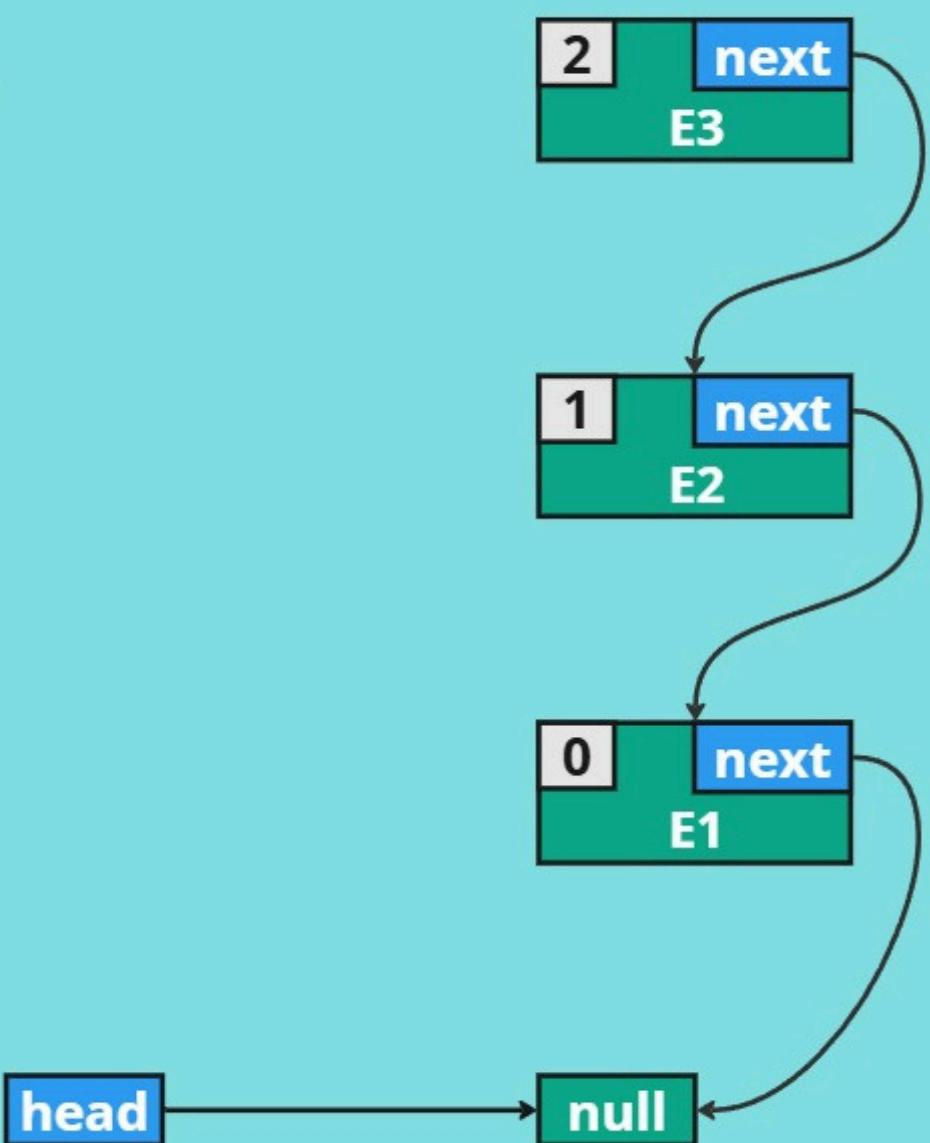
```
public T pop() {
    while (true) {
        final Node<T> previousHead = head.get();
        if (head.compareAndSet(previousHead, previousHead.next)) {
            return previousHead.value;
        }
    }
}
```

thread-3

```
public T pop() {
    while (true) {
        final Node<T> previousHead = head.get();
        if (head.compareAndSet(previousHead, previousHead.next)) {
            return previousHead.value;
        }
    }
}
```

A dashed arrow points from the 'value' return statement in the thread-3 code to a box labeled '0'.

ConcurrentStack



thread-1 → E3

thread-2 → E2

thread-3 → E1

thread-1

```
public T pop() {  
    while (true) {  
        final Node<T> previousHead = head.get();  
        if (head.compareAndSet(previousHead, previousHead.next)) {  
            return previousHead.value;  
        }  
    }  
}
```

thread-2

```
public T pop() {  
    while (true) {  
        final Node<T> previousHead = head.get();  
        if (head.compareAndSet(previousHead, previousHead.next)) {  
            return previousHead.value;  
        }  
    }  
}
```

thread-3

```
public T pop() {  
    while (true) {  
        final Node<T> previousHead = head.get();  
        if (head.compareAndSet(previousHead, previousHead.next)) {  
            return previousHead.value;  
        }  
    }  
}
```

A dashed arrow points from the highlighted line "return previousHead.value;" to a small box containing the value "0".

ConcurrentStack

thread →



```
public T pop() {
    while (true) {
        final Node<T> previousHead = head.get();
        if (head.compareAndSet(previousHead, previousHead.next)) {
            return previousHead.value;
        }
    }
}
```

A red callout box with a black border and a red 'X' icon contains the text "NullPointerException". A curved black arrow points from the word "next" in the highlighted line of code to the "NullPointerException" box.

ConcurrentStack

thread →



```
public Optional<T> pop() {  
    while (true) {  
        final Node<T> previousHead = head.get();  
        if (previousHead == null) {  
            return empty();  
        }  
        if (head.compareAndSet(previousHead, previousHead.next)) {  
            return Optional.of(previousHead.value);  
        }  
    }  
}
```