Introducing Node-based Data Structures: Linked Lists

Giovanni Dicanio
AUTHOR, SOFTWARE ENGINEER

https://blogs.msmvps.com/gdicanio

Overview

What is a linked list?

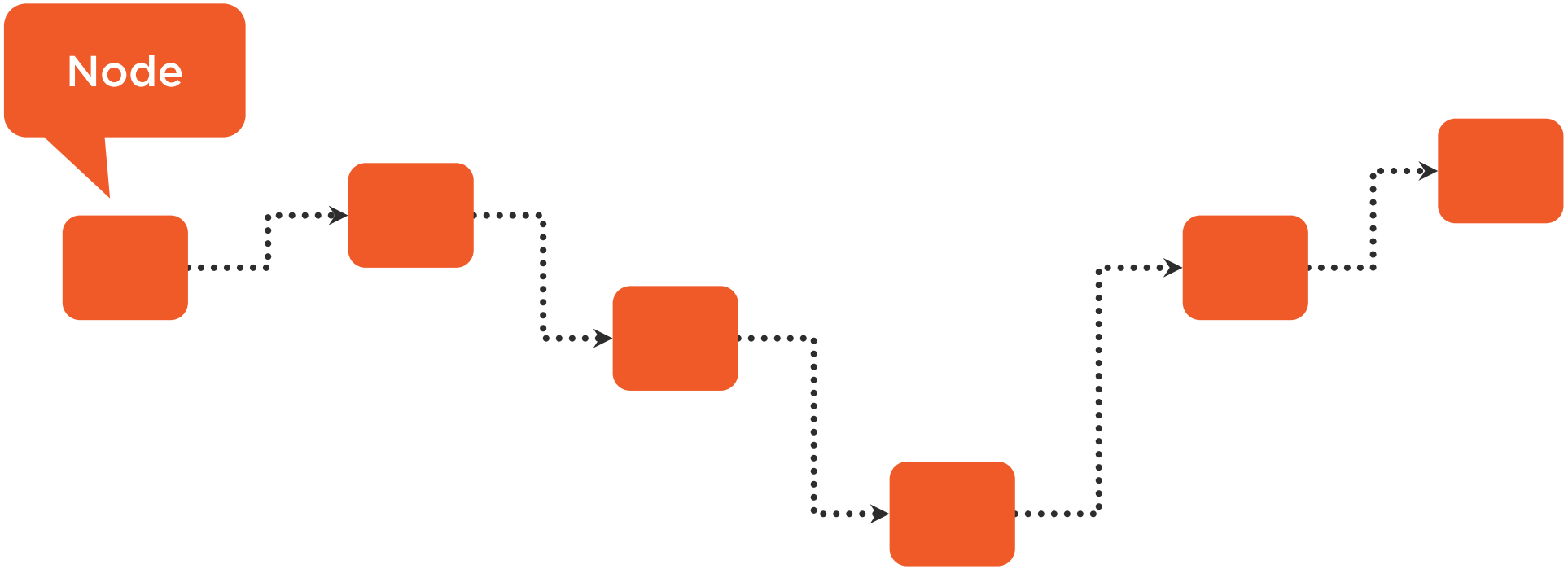
Basic linked list operations

- Insert, remove, traverse

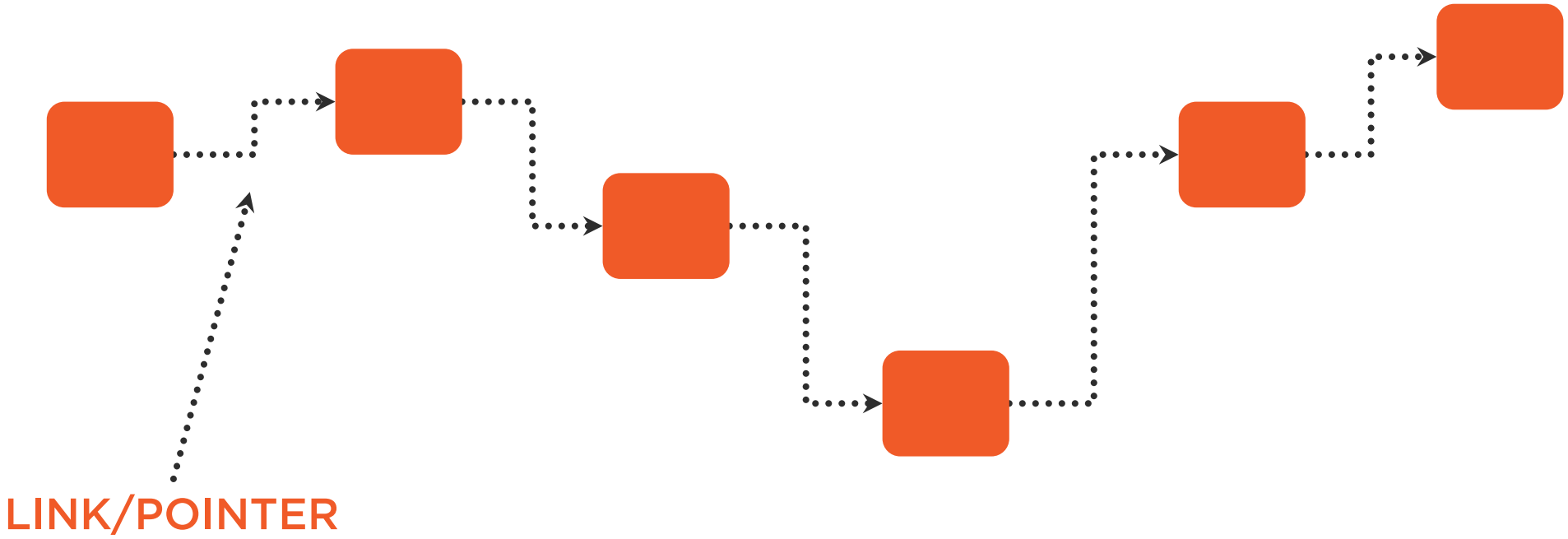
Concrete C++ implementation

Linked List

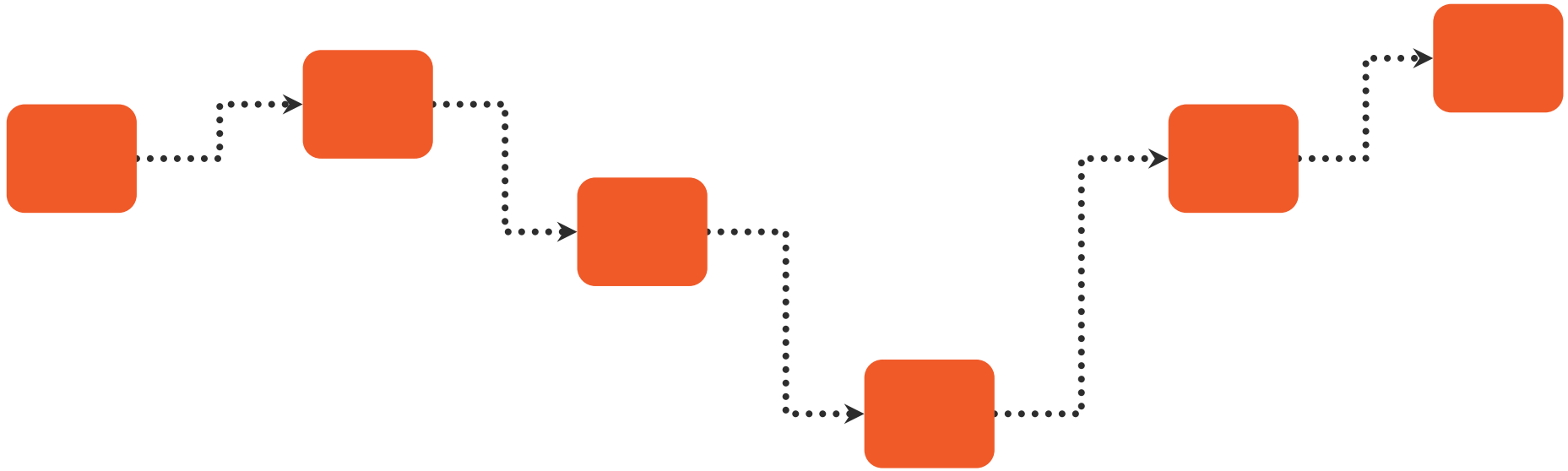
Linked List



Linked List



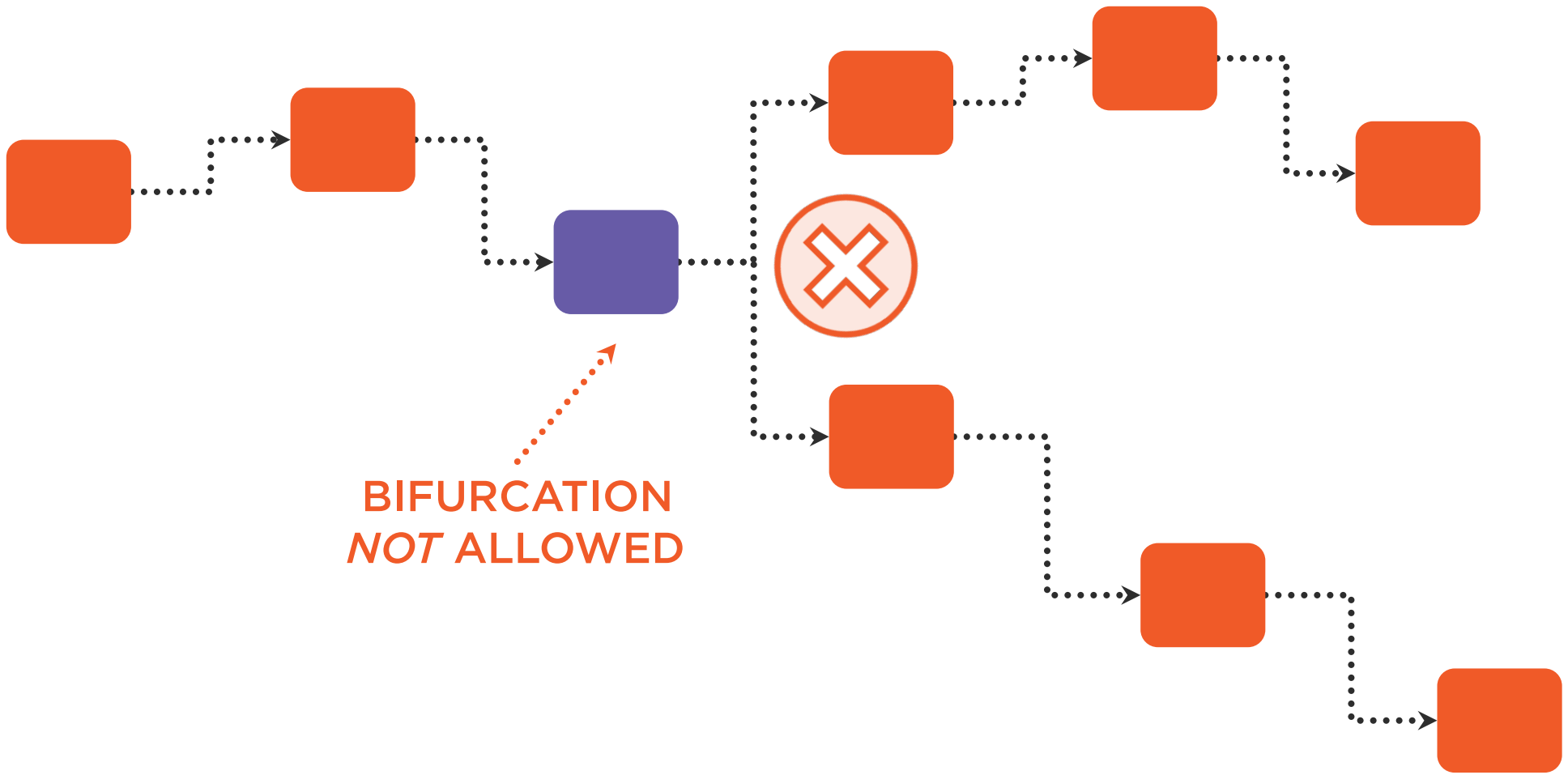
Linked List



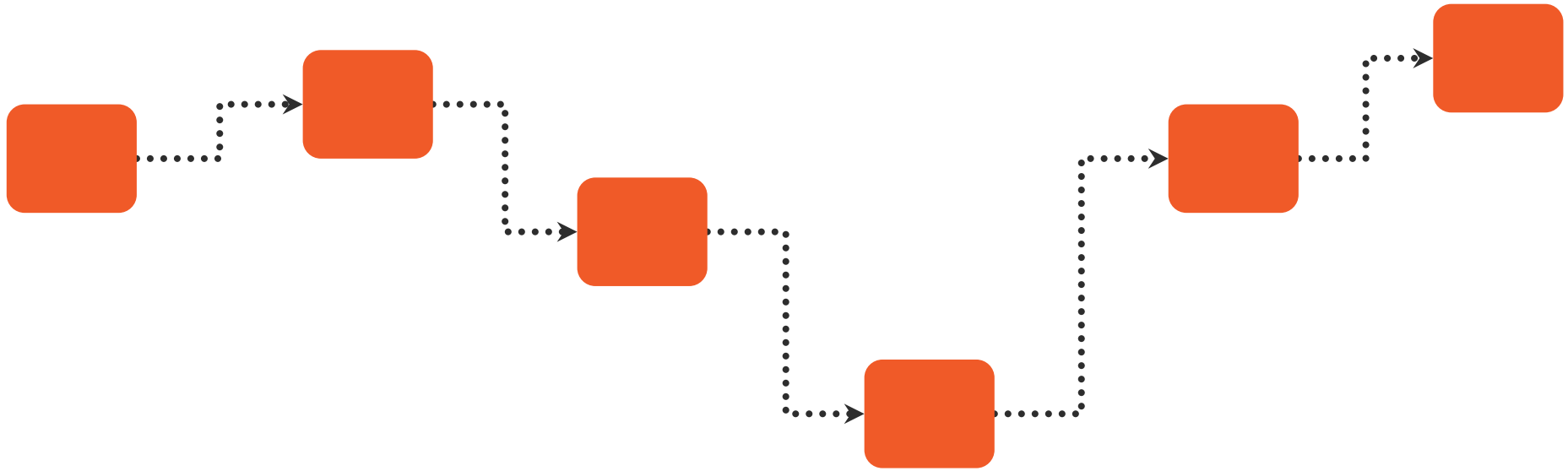
Linear sequence of nodes

No bifurcations/branches

Not a Linked List



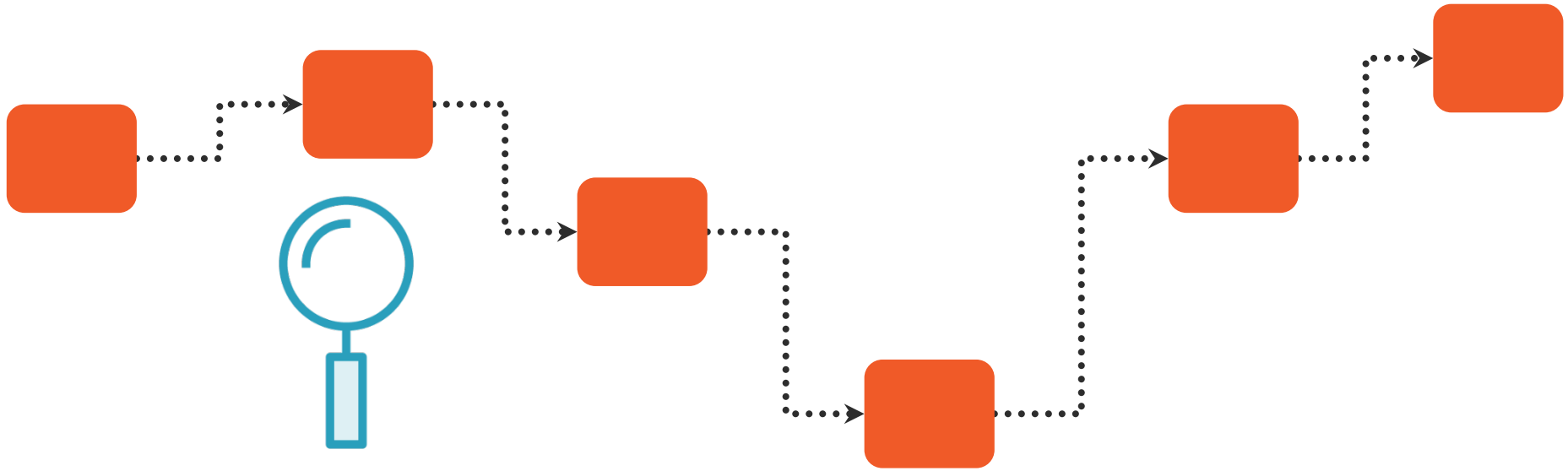
Linked List

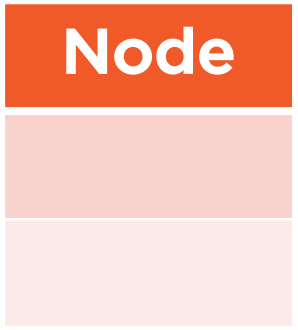


Linear sequence of nodes

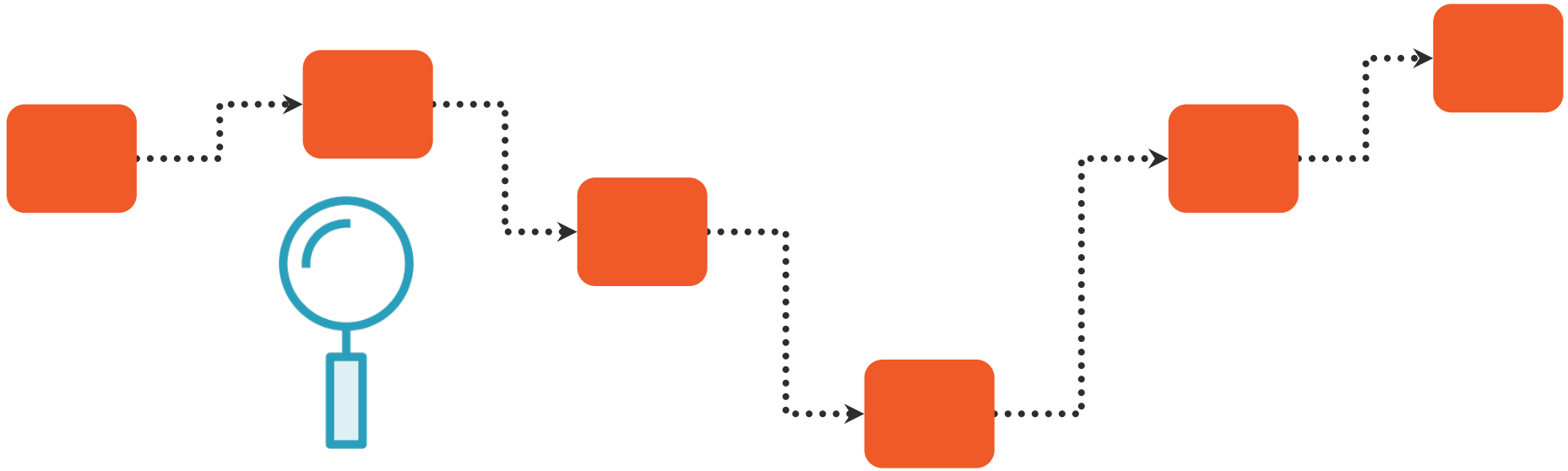
No bifurcations/branches

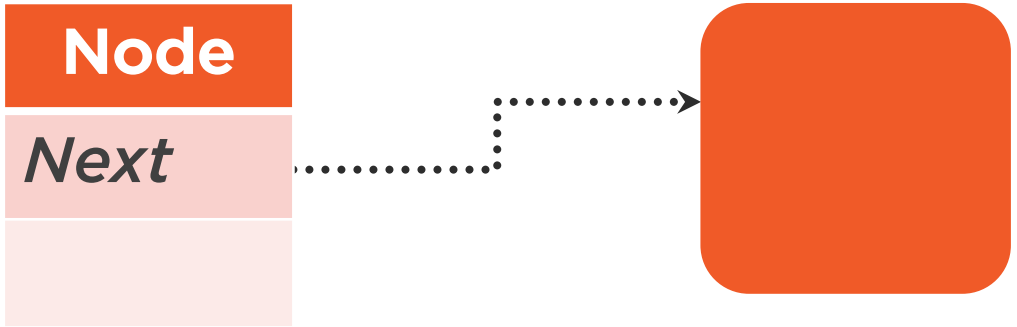
What's Inside a Linked List Node?



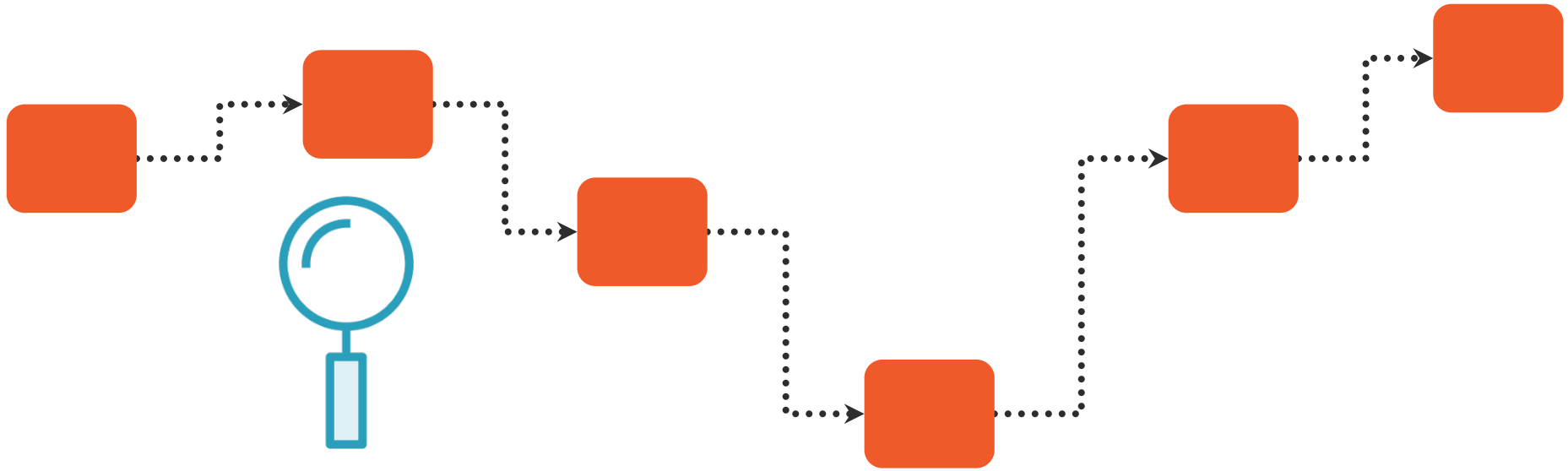


What's Inside a Linked List Node?





What's Inside a Linked List Node?

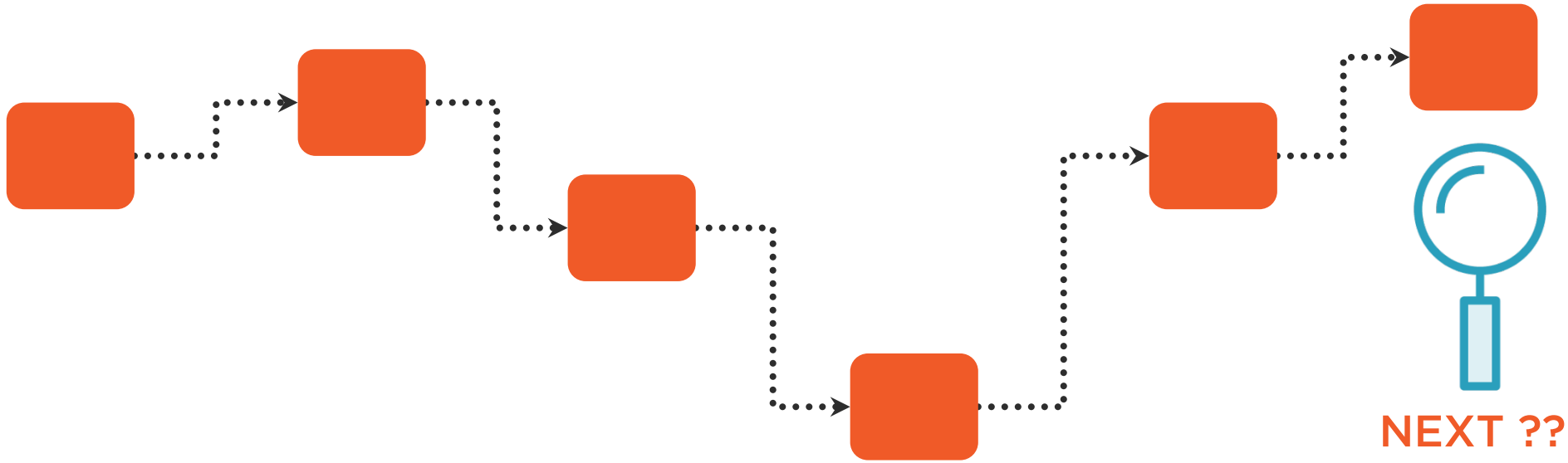


Node
Next
Data

64

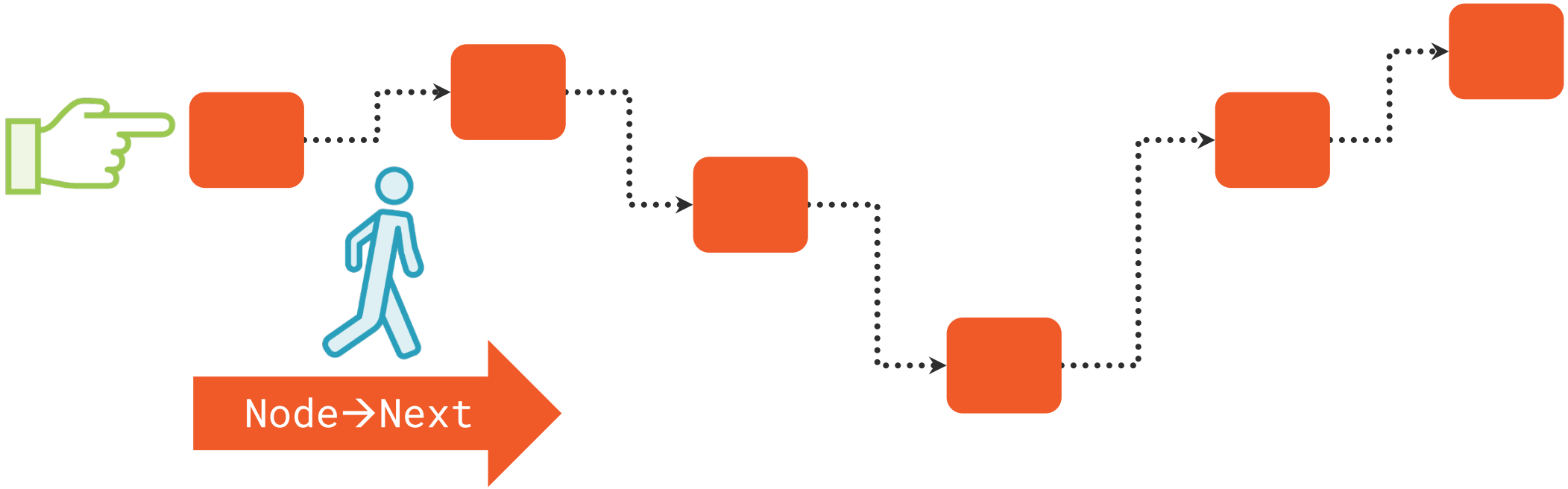
List<*T*>

Last Node's Next?

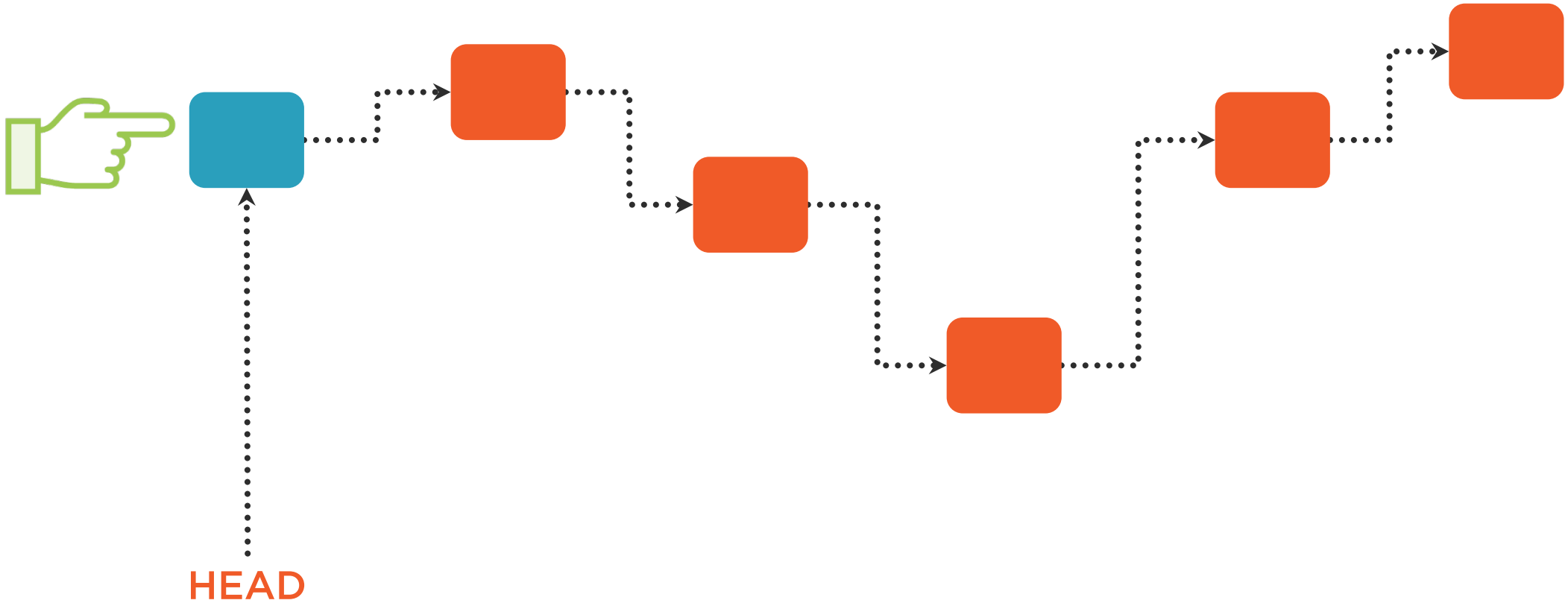


NEXT ??

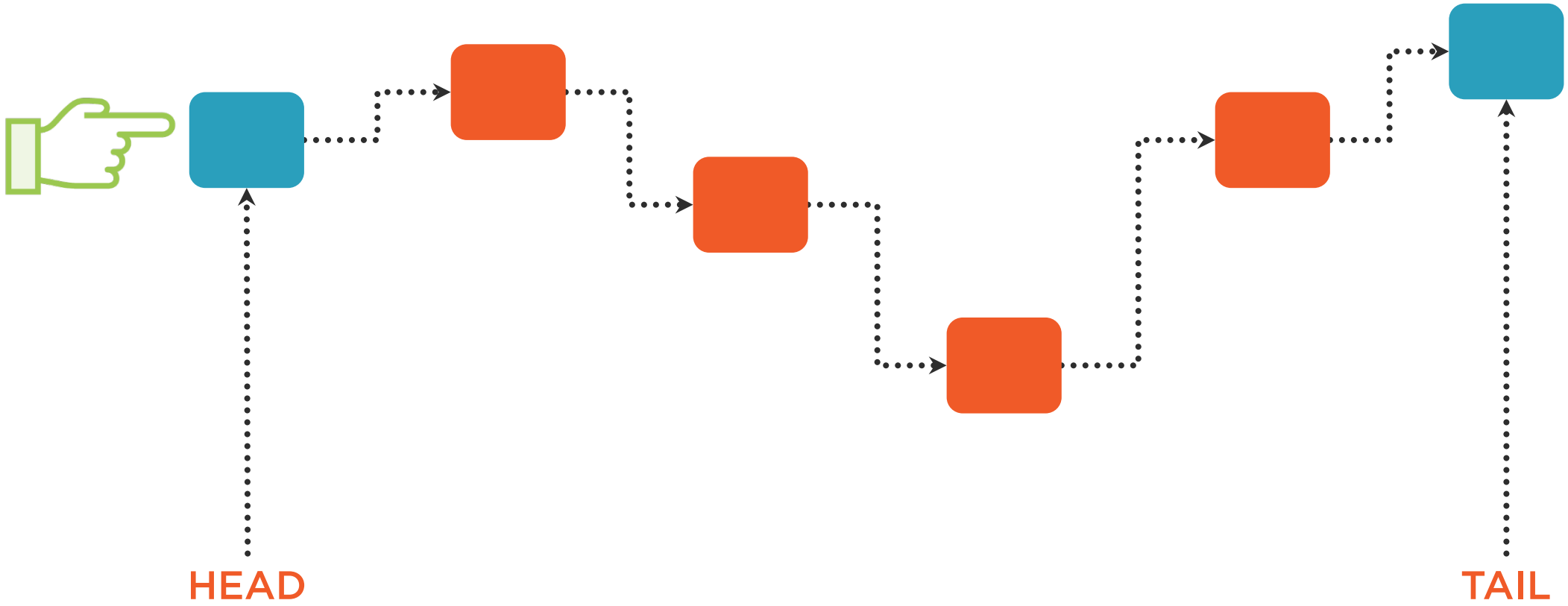
Linked List's Head and Tail



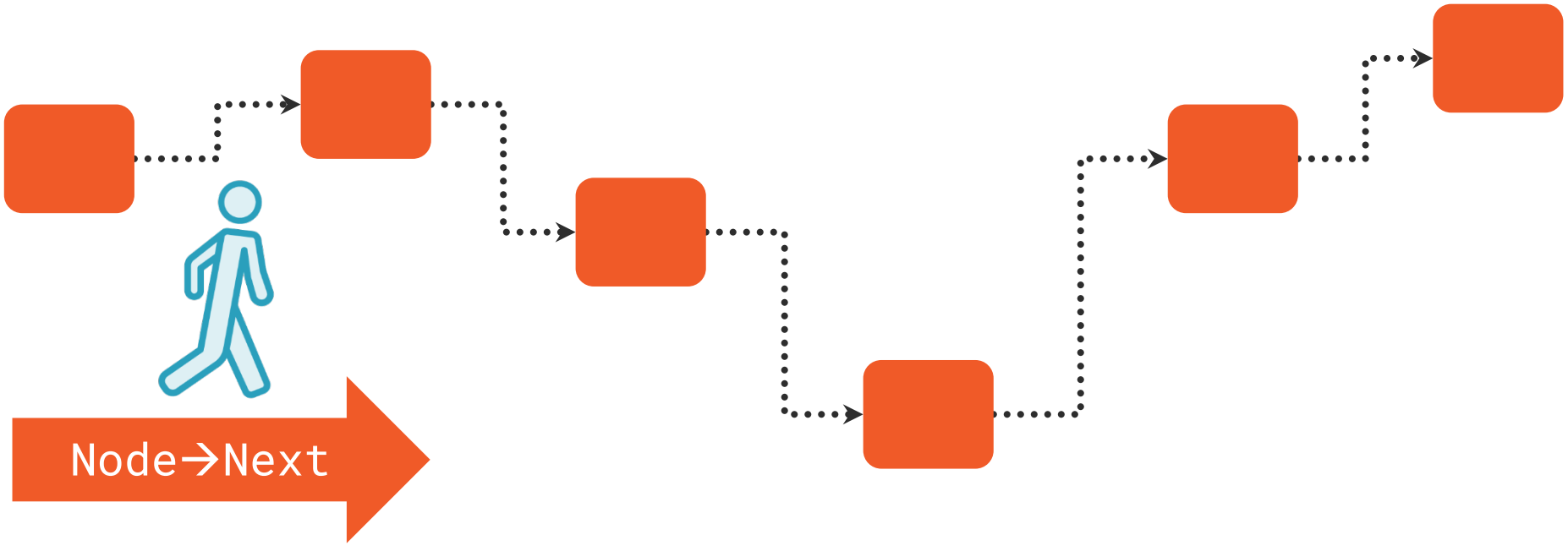
Linked List's Head and Tail



Linked List's Head and Tail

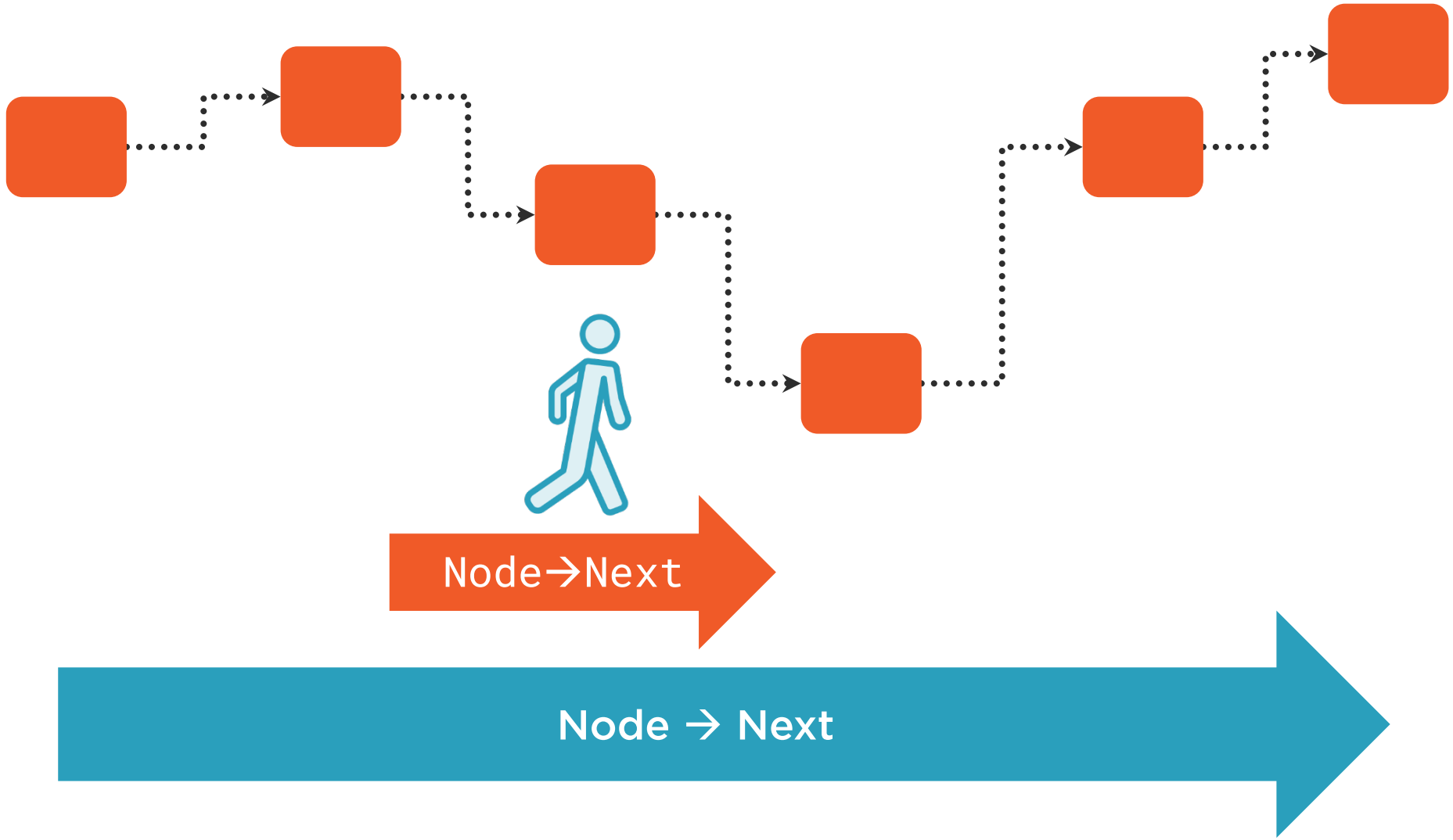


Singly Linked List

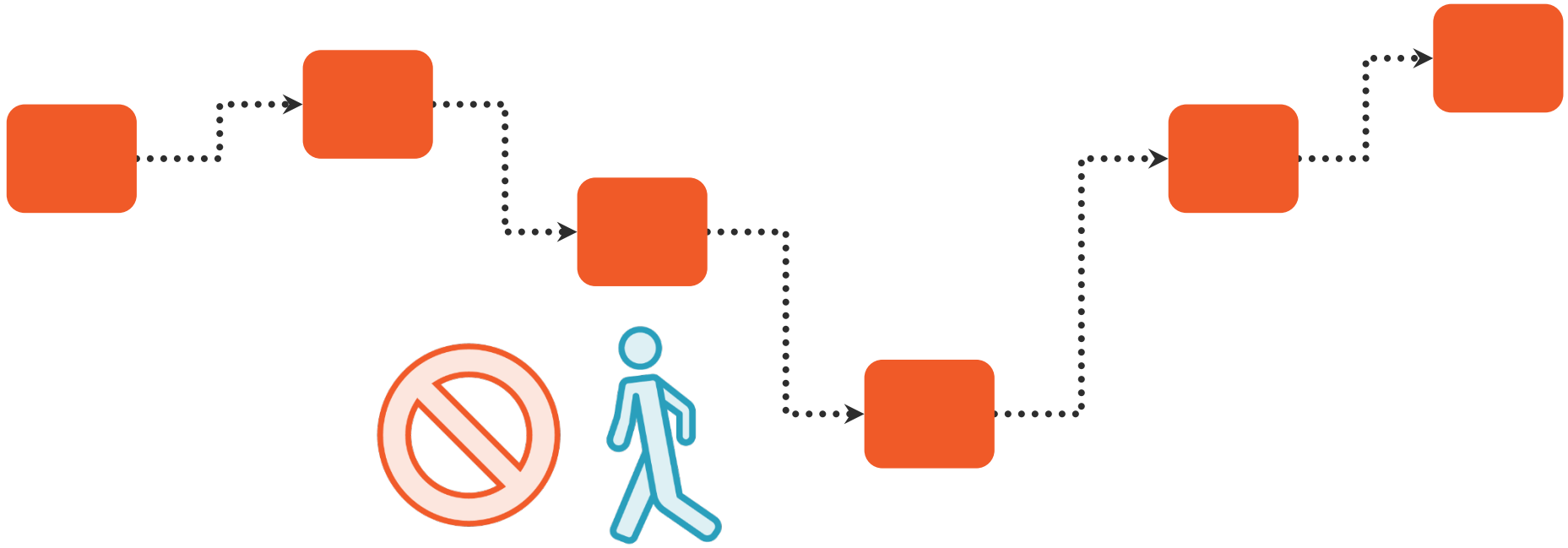


Node → Next

Singly Linked List

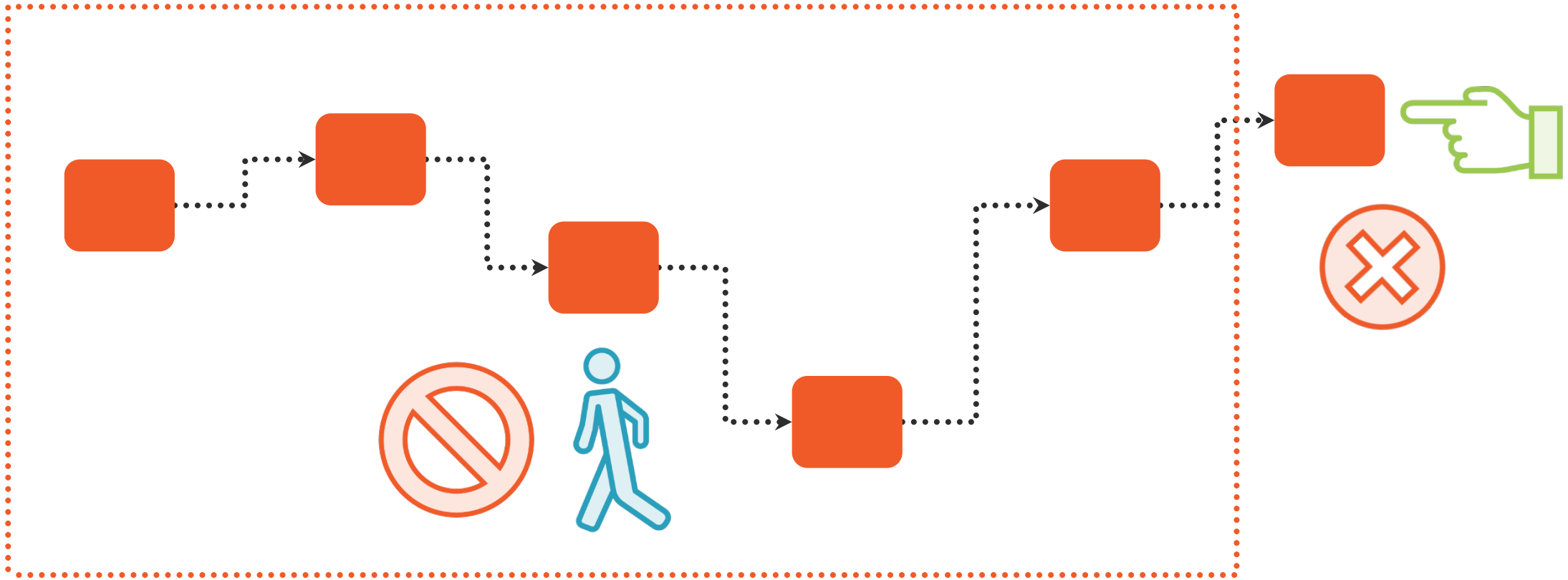


Singly Linked List

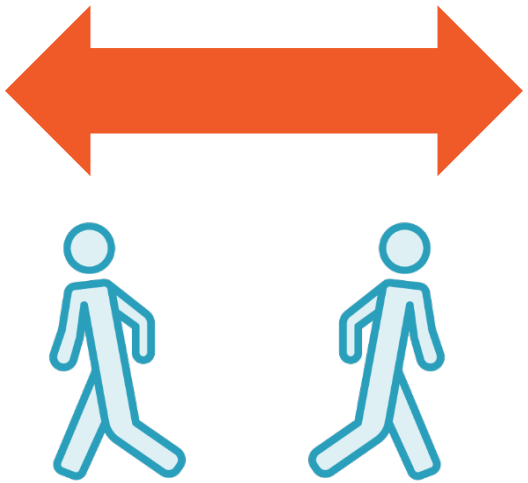


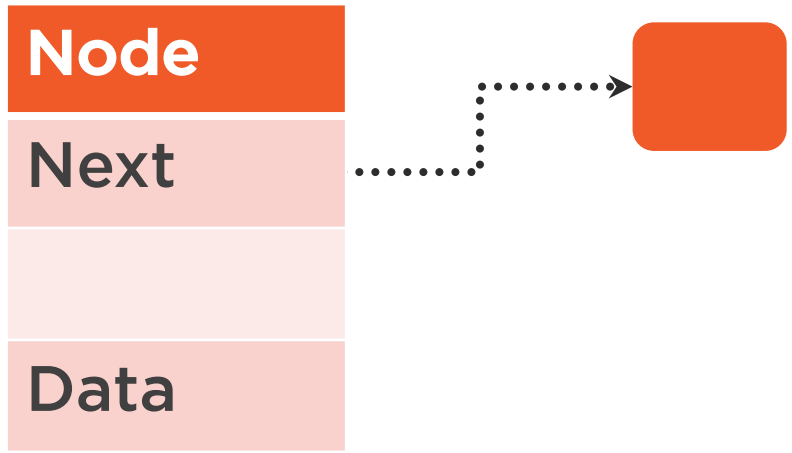
There's *no* Node→Previous

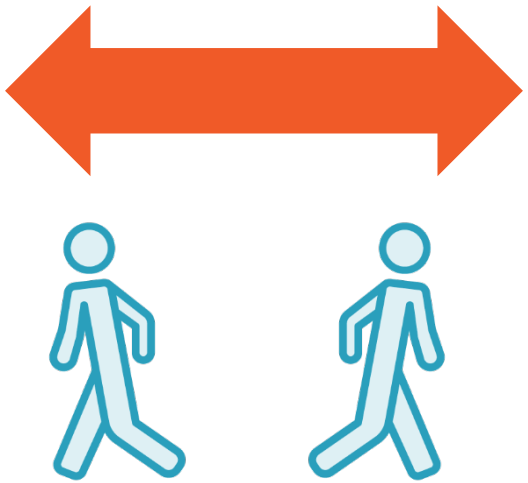
Singly Linked List

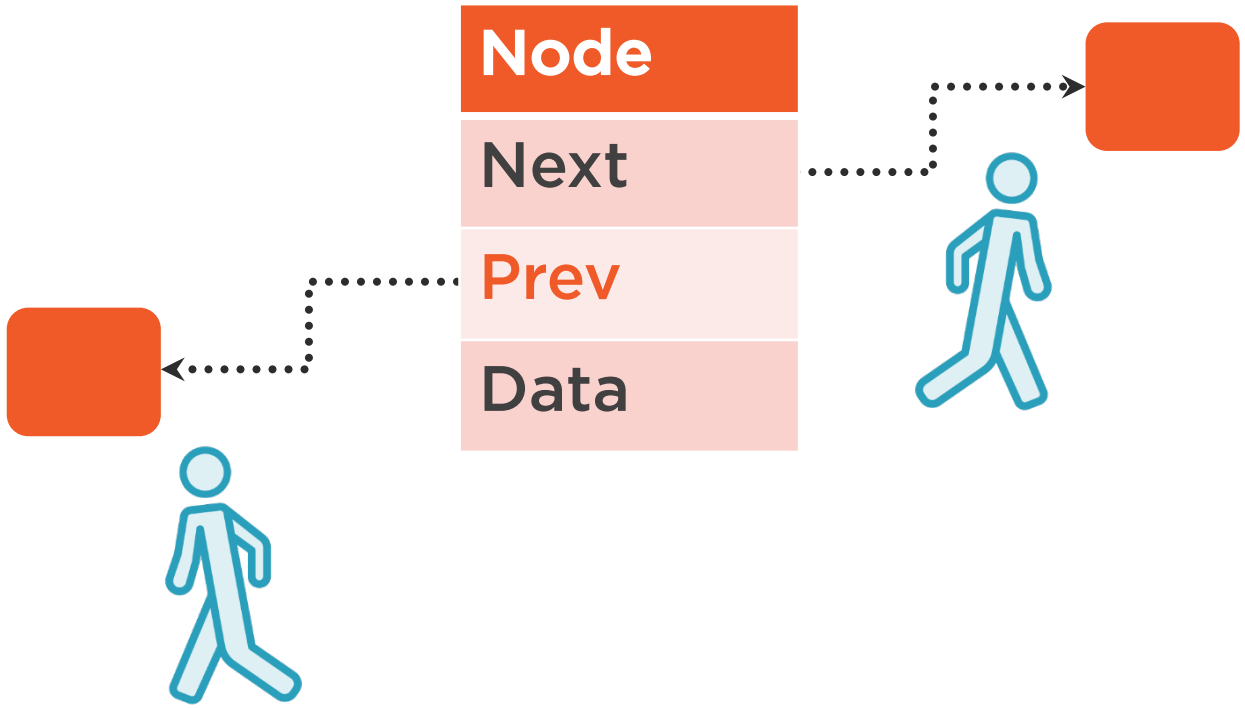


There's *no* Node→Previous

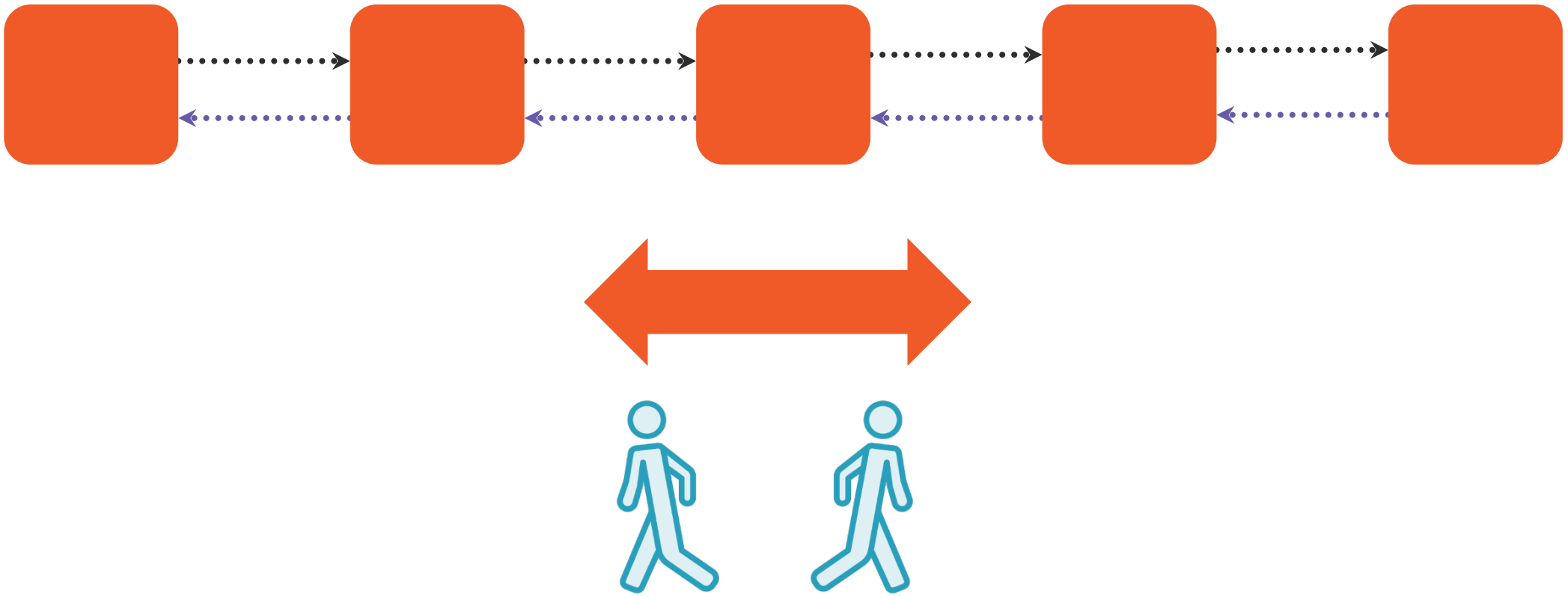



Node
Next
Data

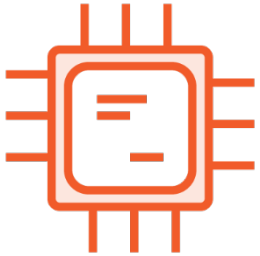



Node
Next
Prev
Data

Doubly Linked List



Doubly Linked List Overhead

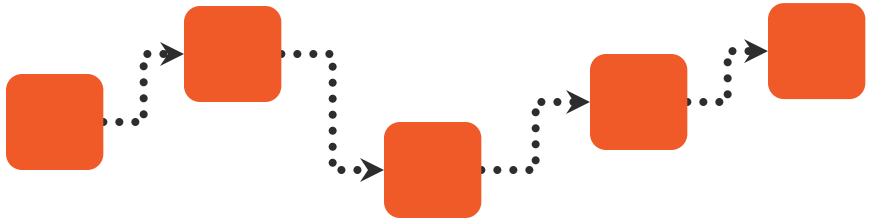


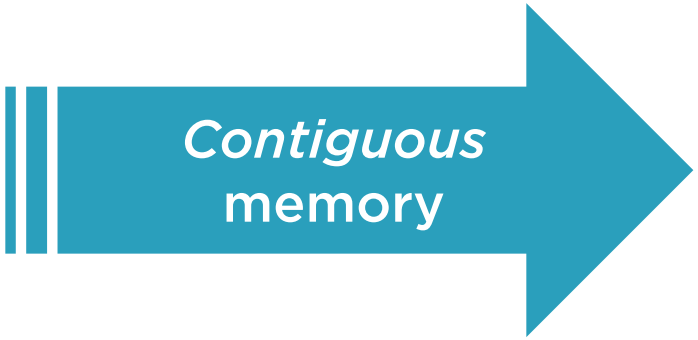
64 bit

+ 64 bit (8 byte) pointer *per node*

10M nodes → 76 MB *overhead*

Arrays vs. Linked Lists: Memory Layout

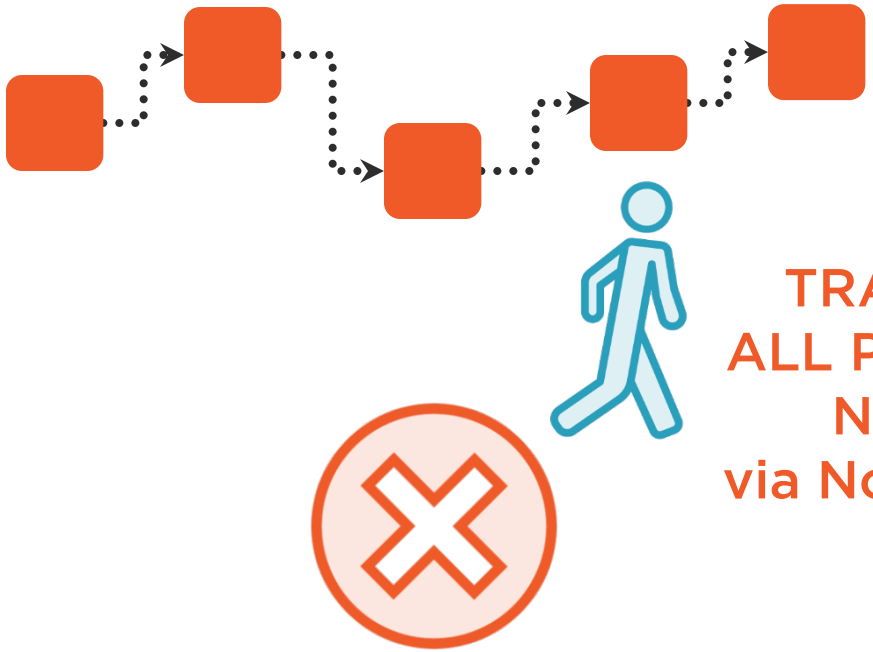




Arrays vs. Linked Lists: Element Access

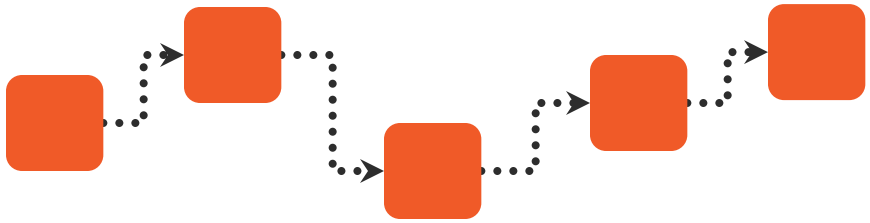
a[i]

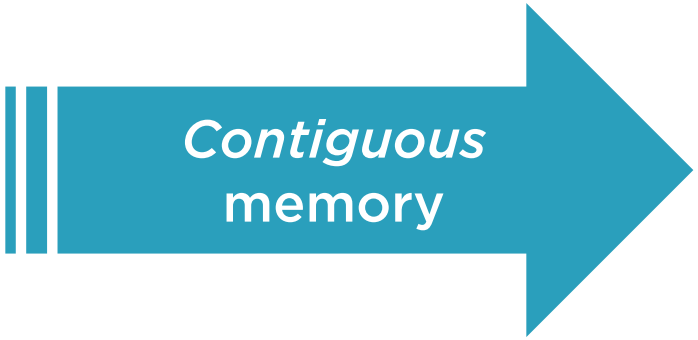
Direct fast
element access
by index

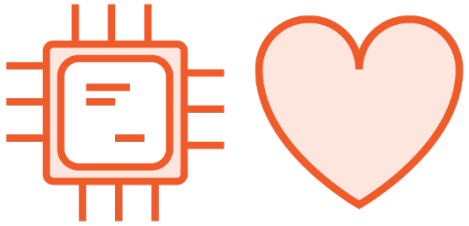


TRAVERSE
ALL PREVIOUS
NODES
via Node→*Next*

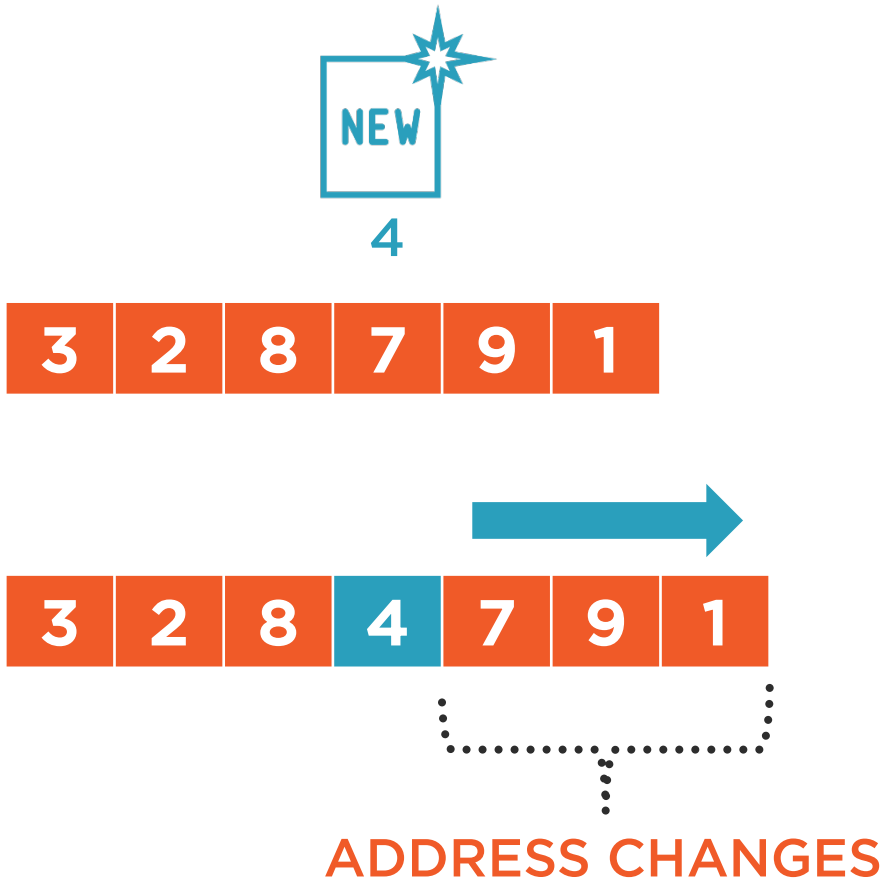
Arrays vs. Linked Lists: Cache Behavior





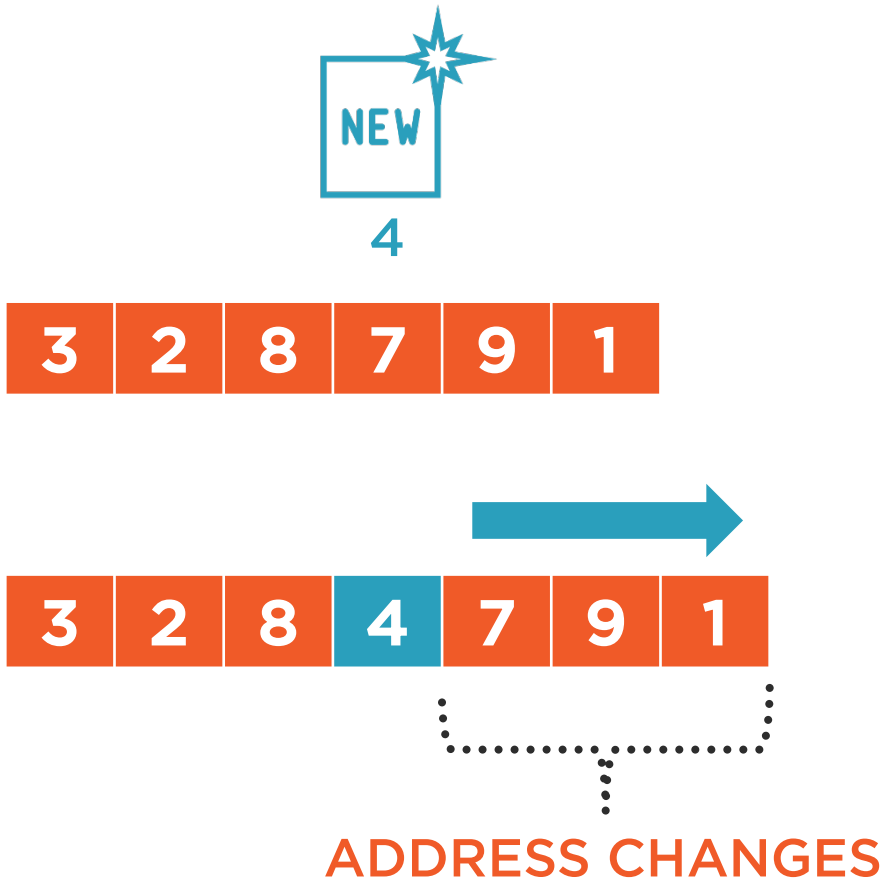


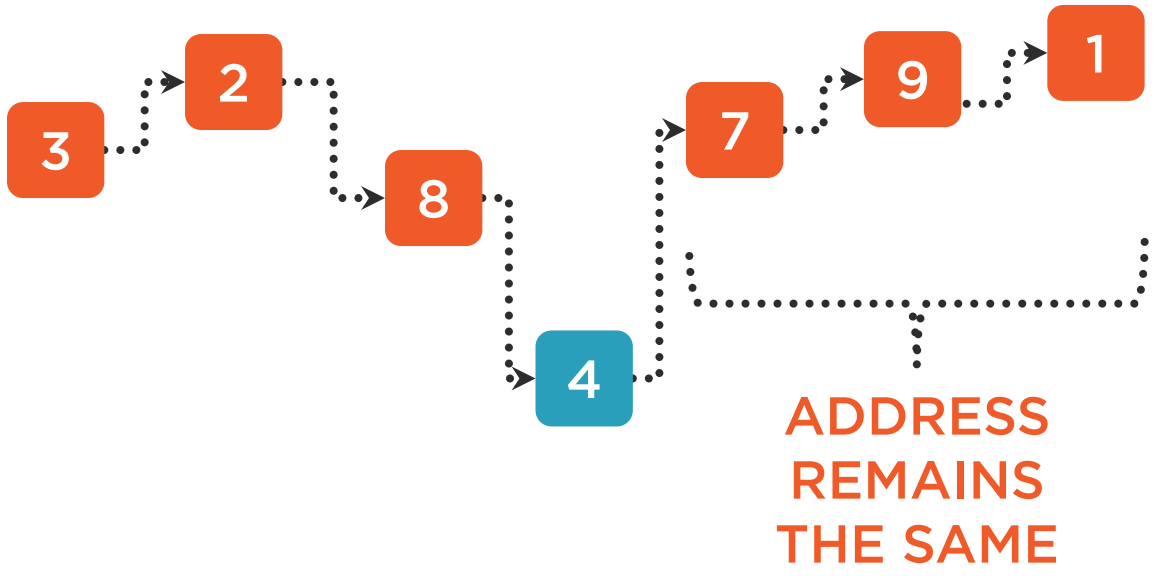
Arrays vs. Linked Lists: Item Insertion



Memory (re)allocation/copy overhead

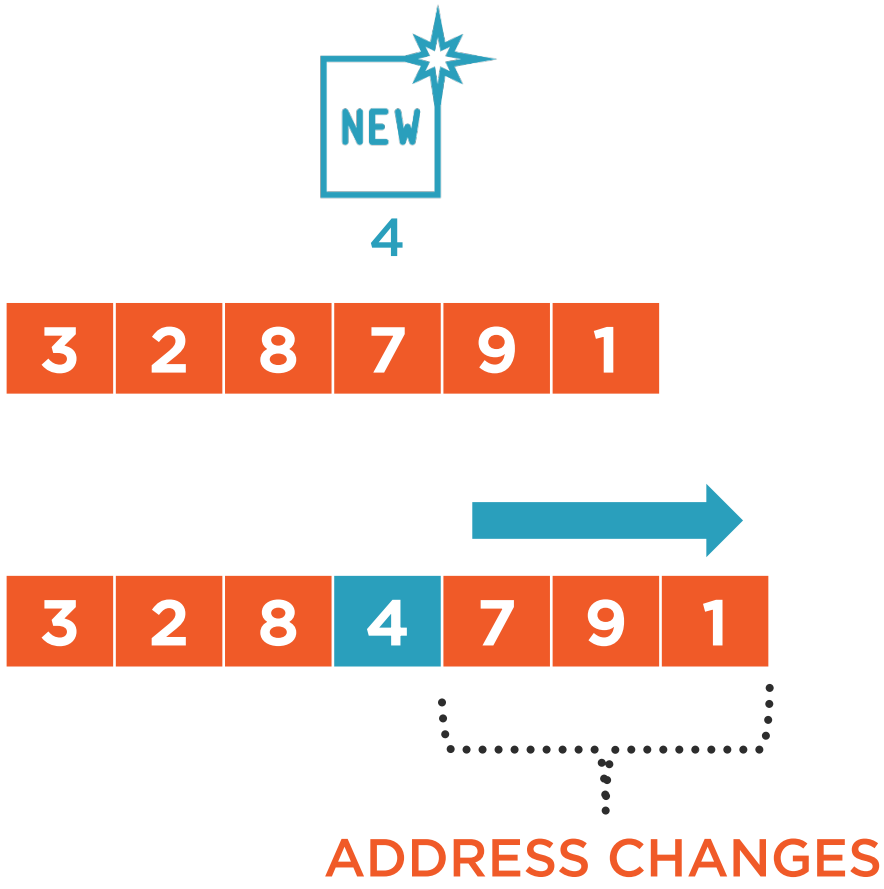
Arrays vs. Linked Lists: Item Insertion

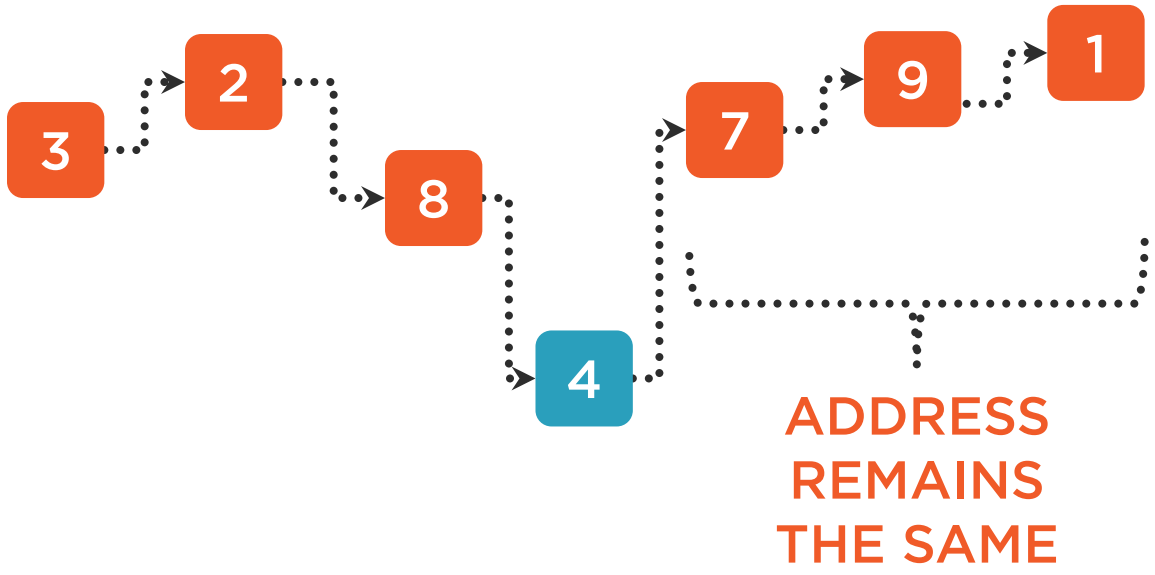




NO
(re)allocation/copy
overhead

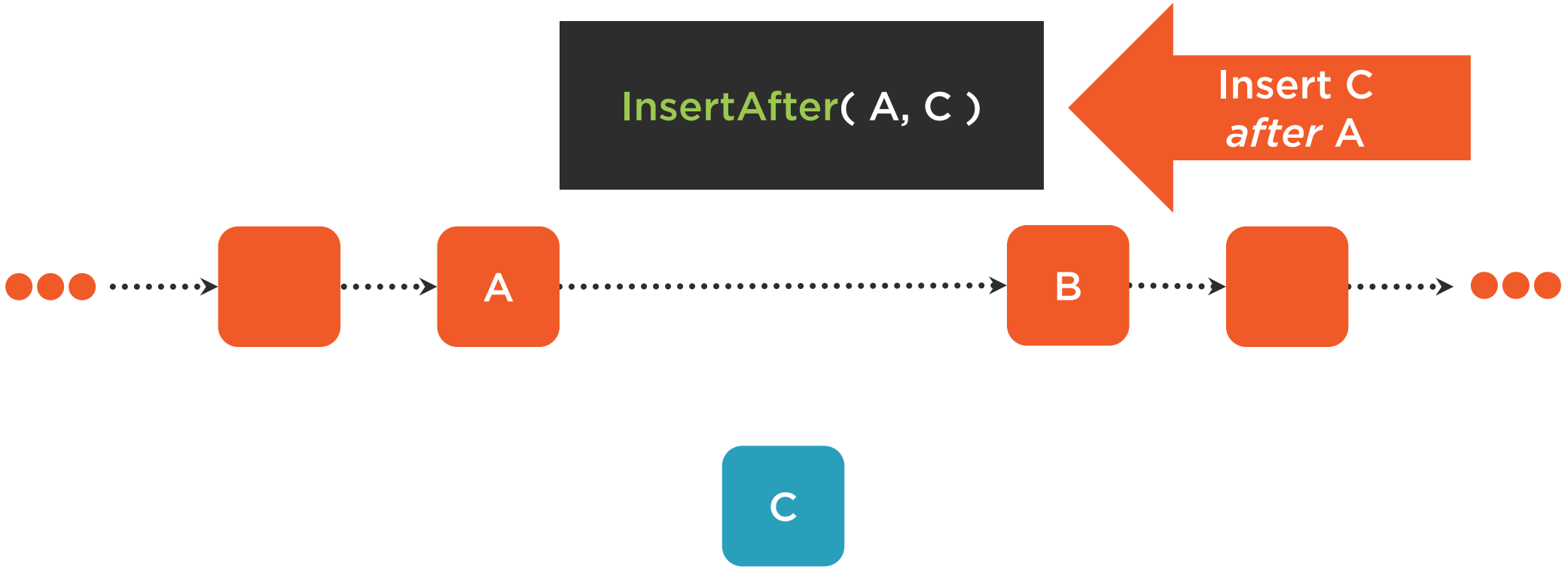
Linked List Invariant: Address of Existing Nodes



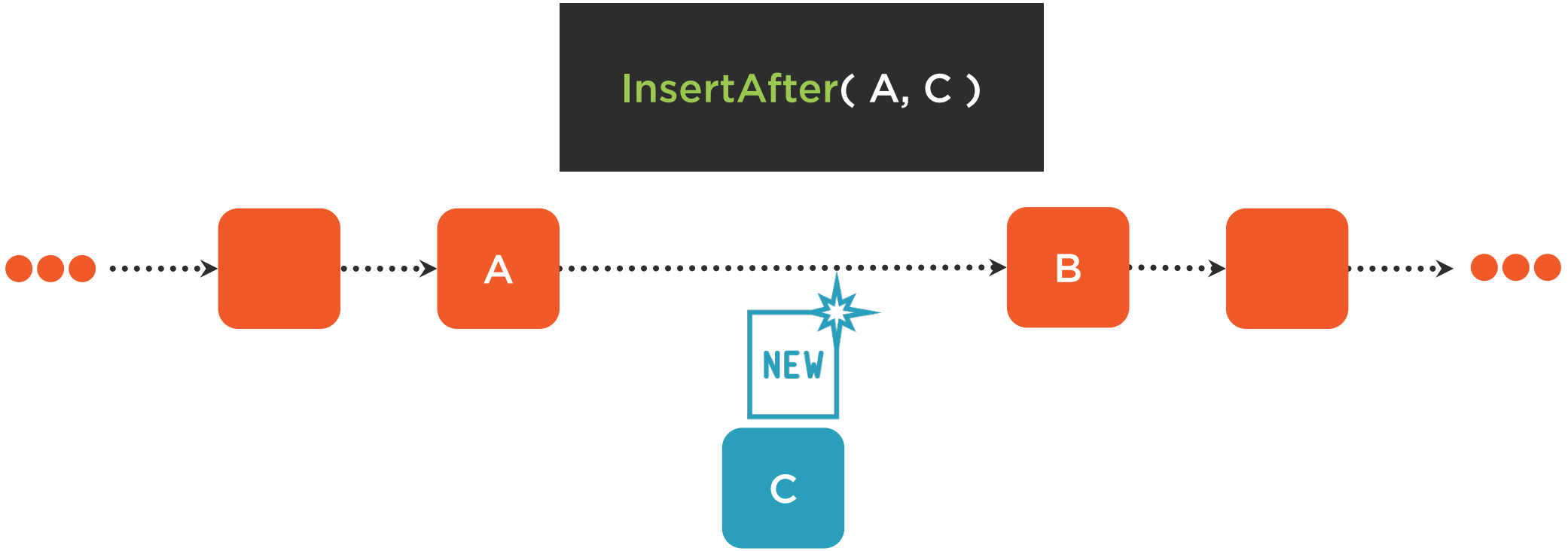


When in Doubt, Measure

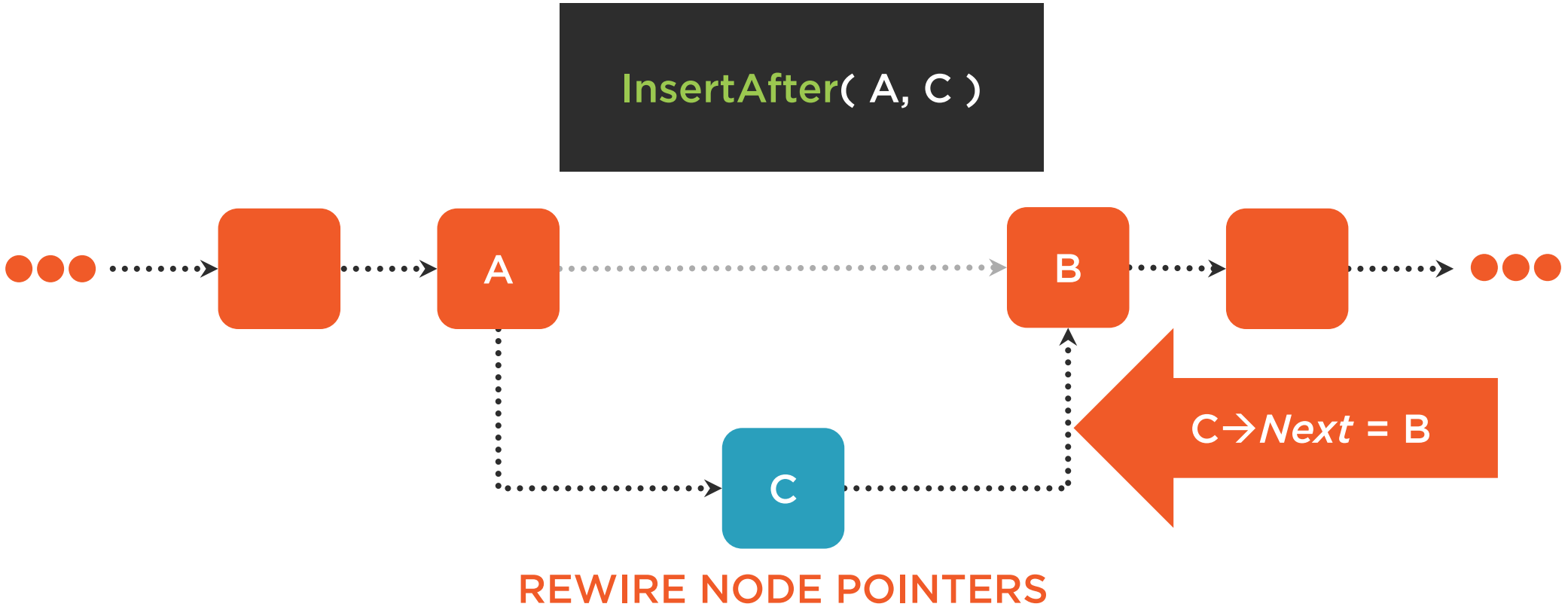
Inserting a New Node



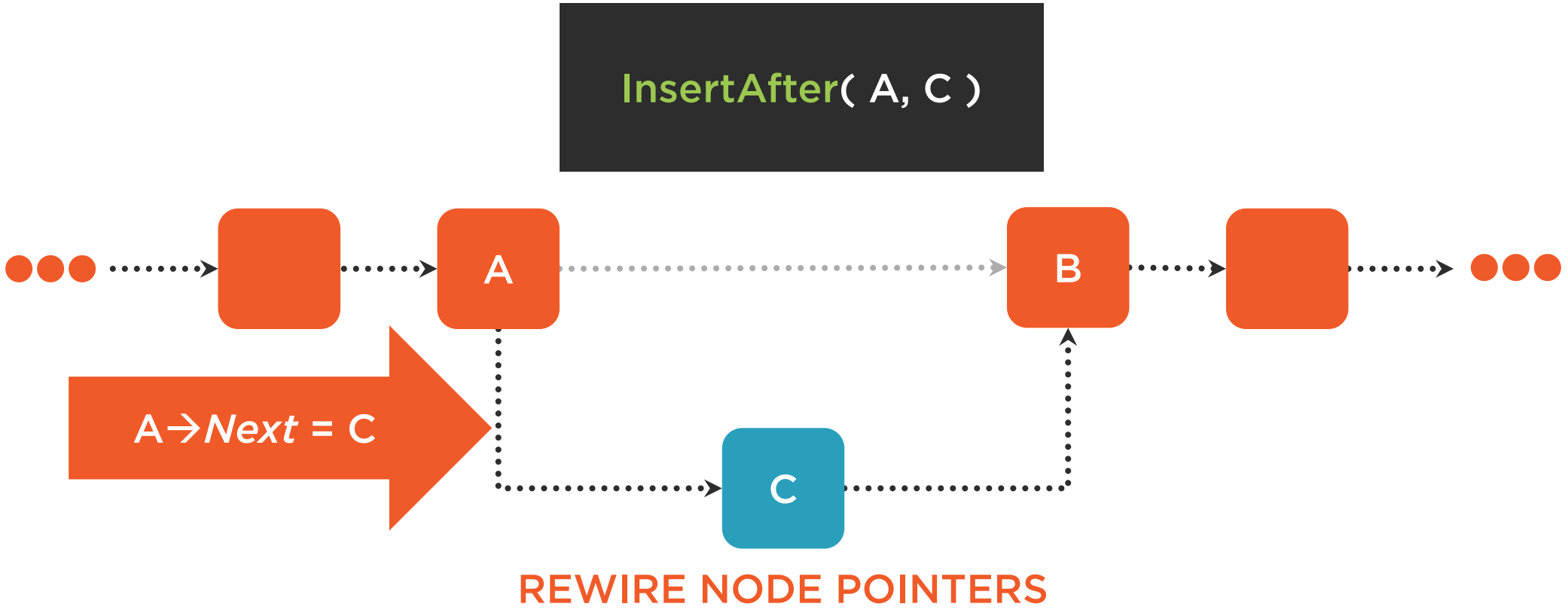
Inserting a New Node



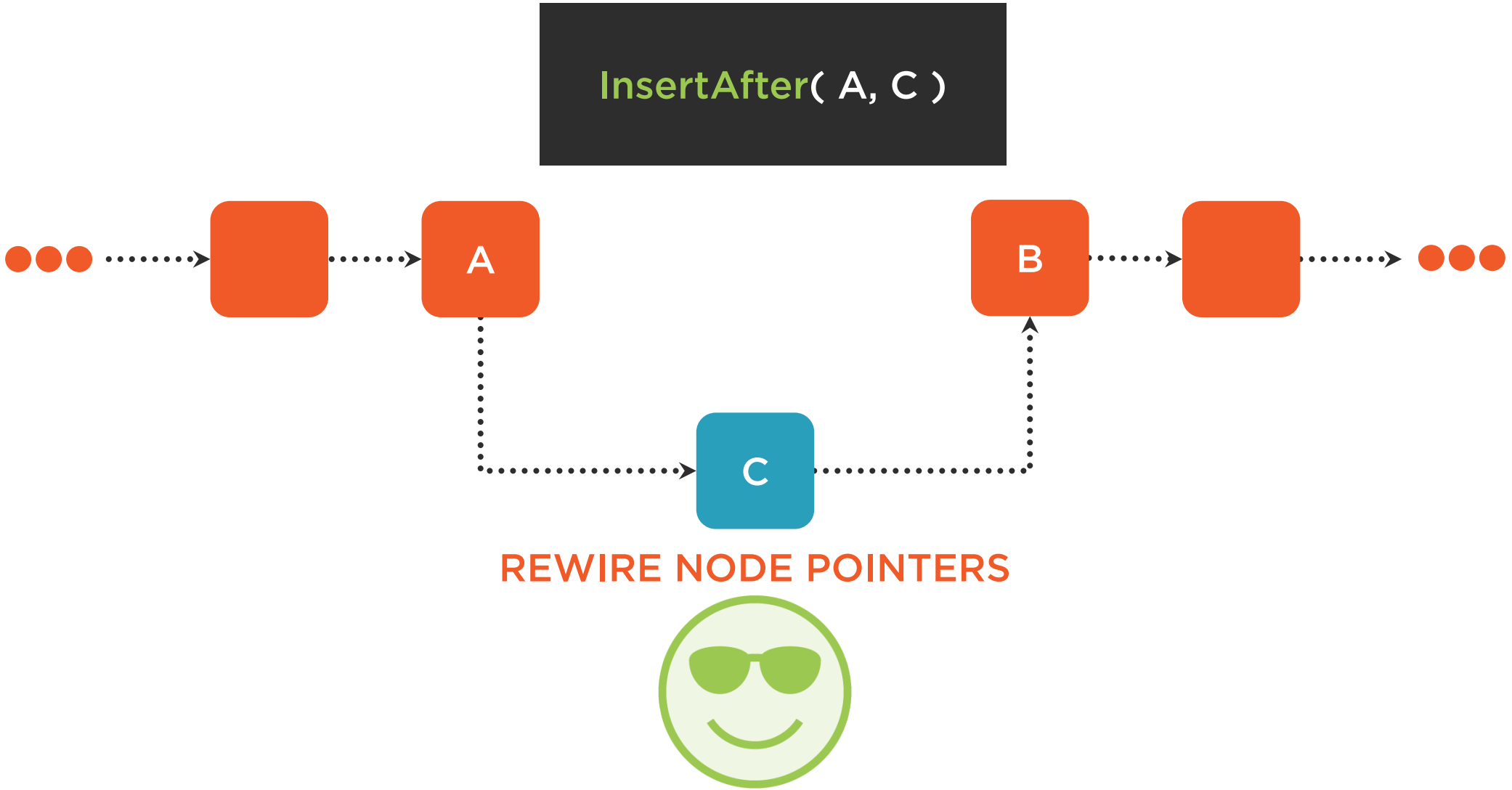
Inserting a New Node



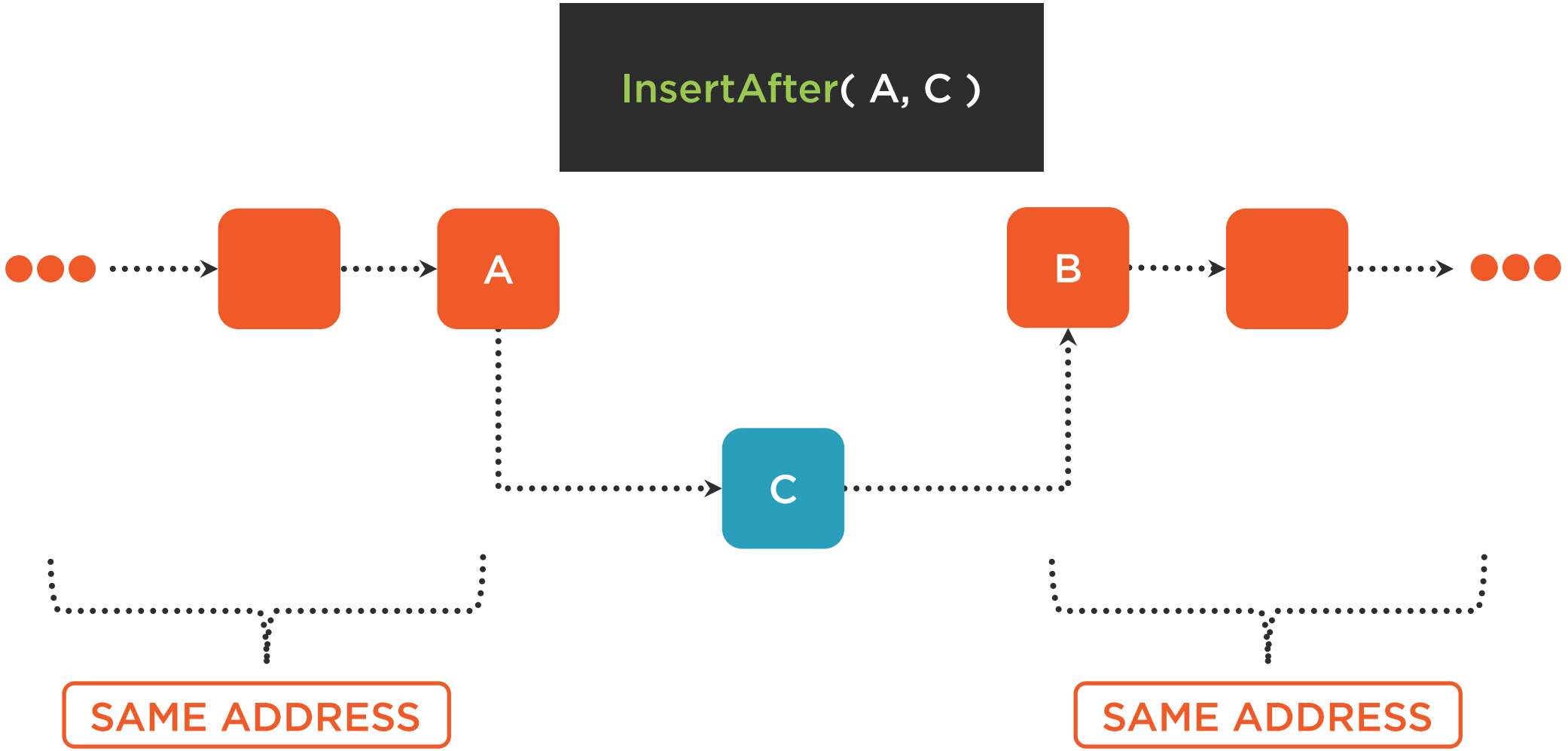
Inserting a New Node



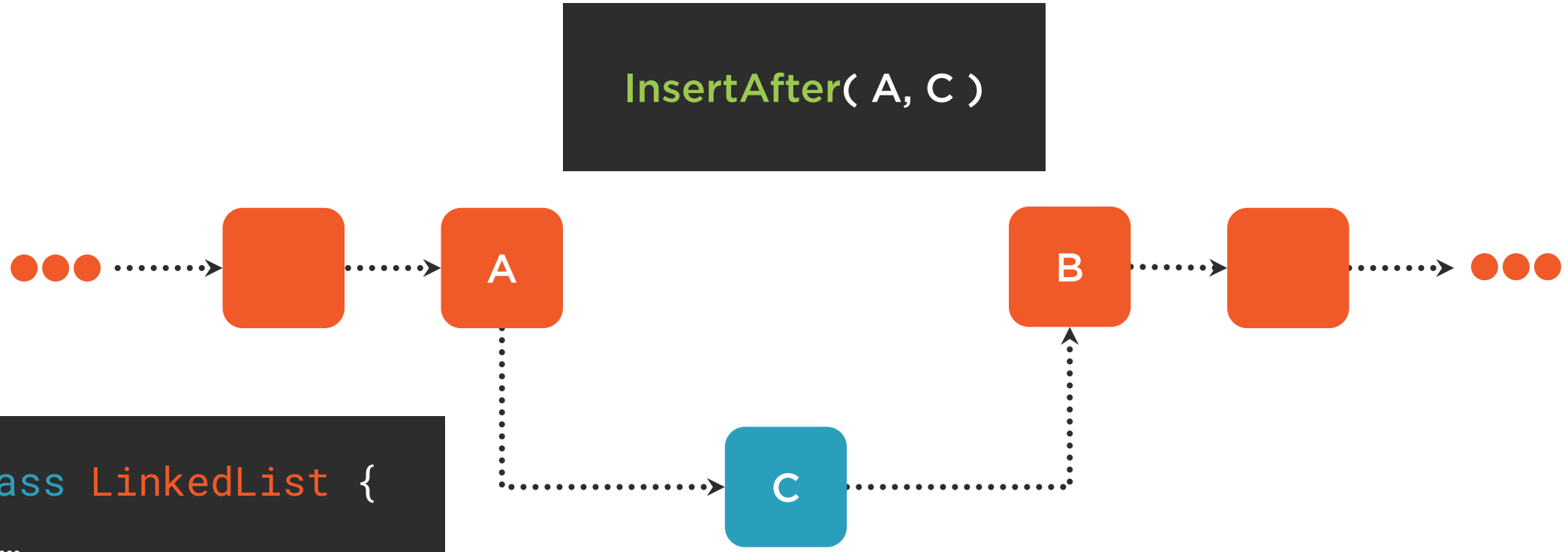
Inserting a New Node



Existing Node Address Invariant

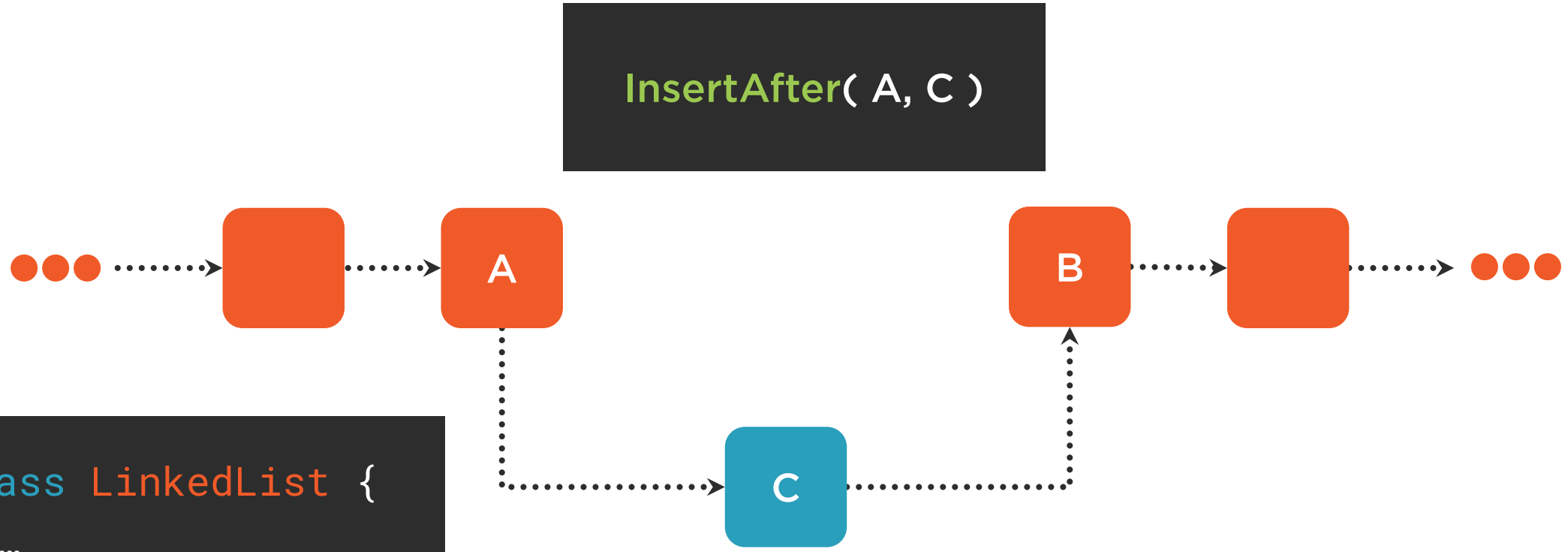


Inserting a New Node




```
class LinkedList {
   …
private:
   int m_nodeCount;
};
```

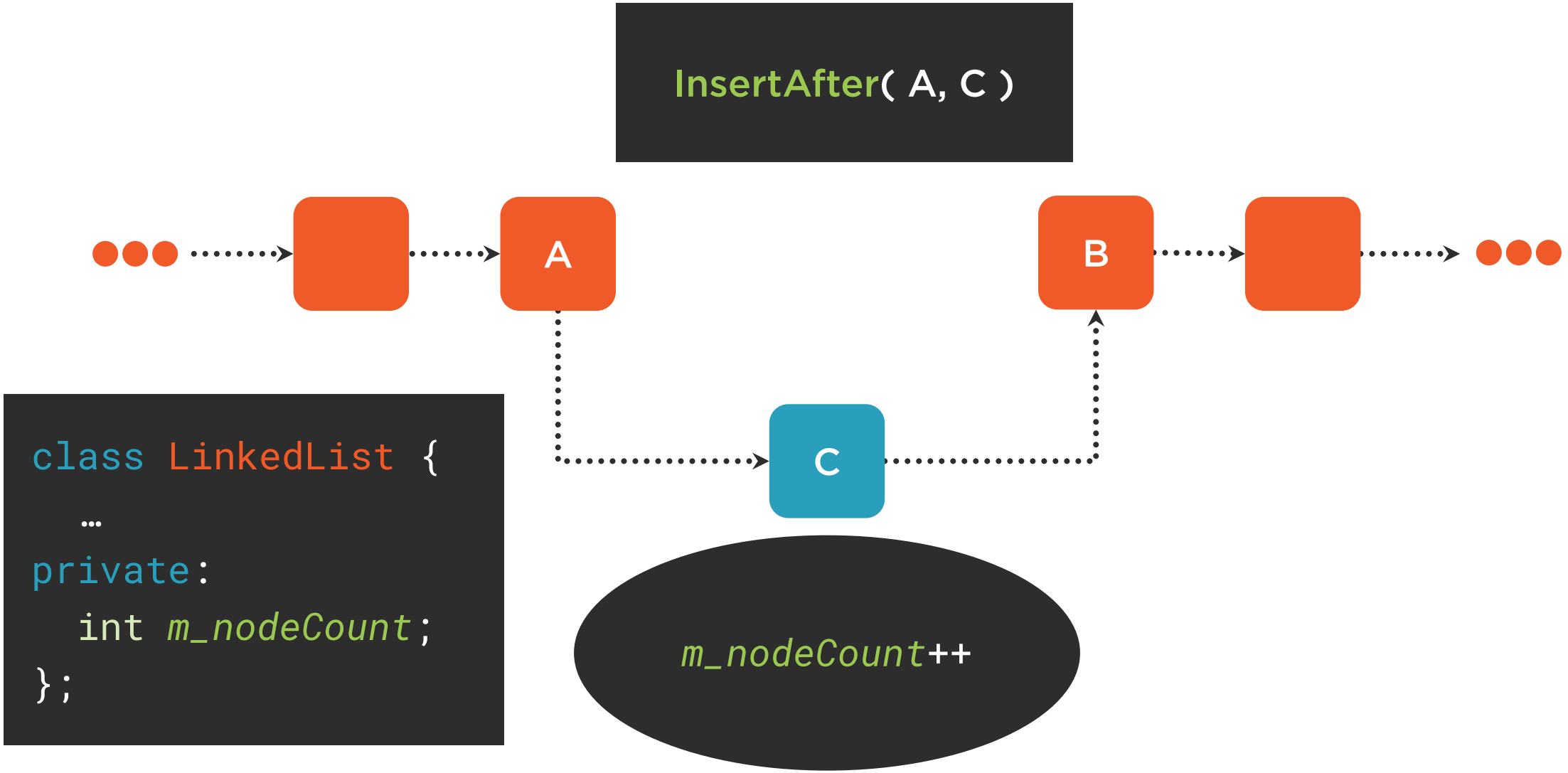
Inserting a New Node




```
class LinkedList {
   …
private:
   int m_nodeCount;
};
```

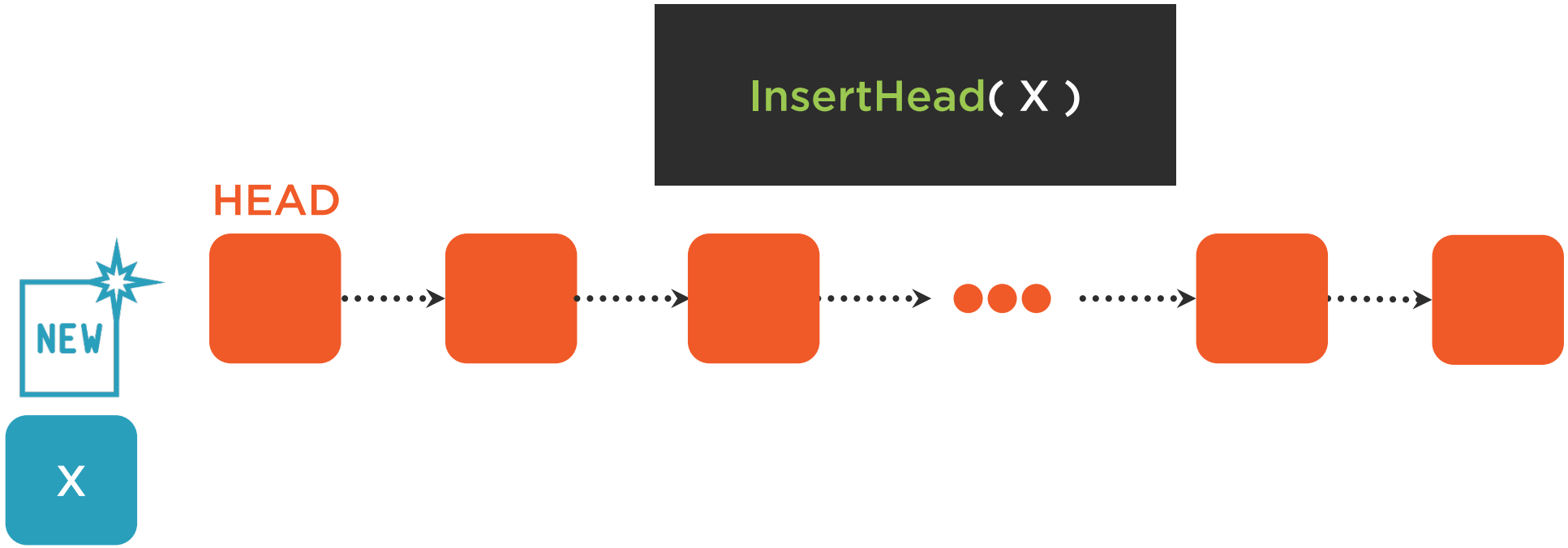
Just return it

Inserting a New Node

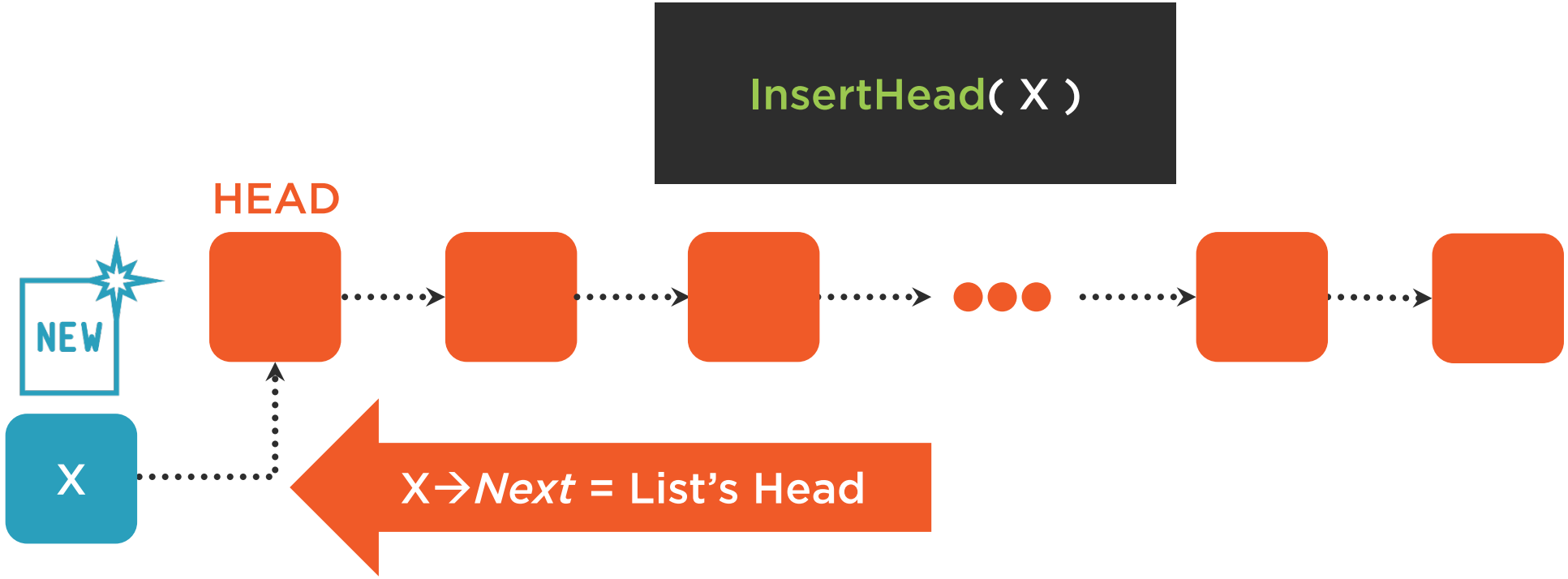



```
class LinkedList {
  …
private:
  int m_nodeCount;
};
```

Special Case: Head Insertion

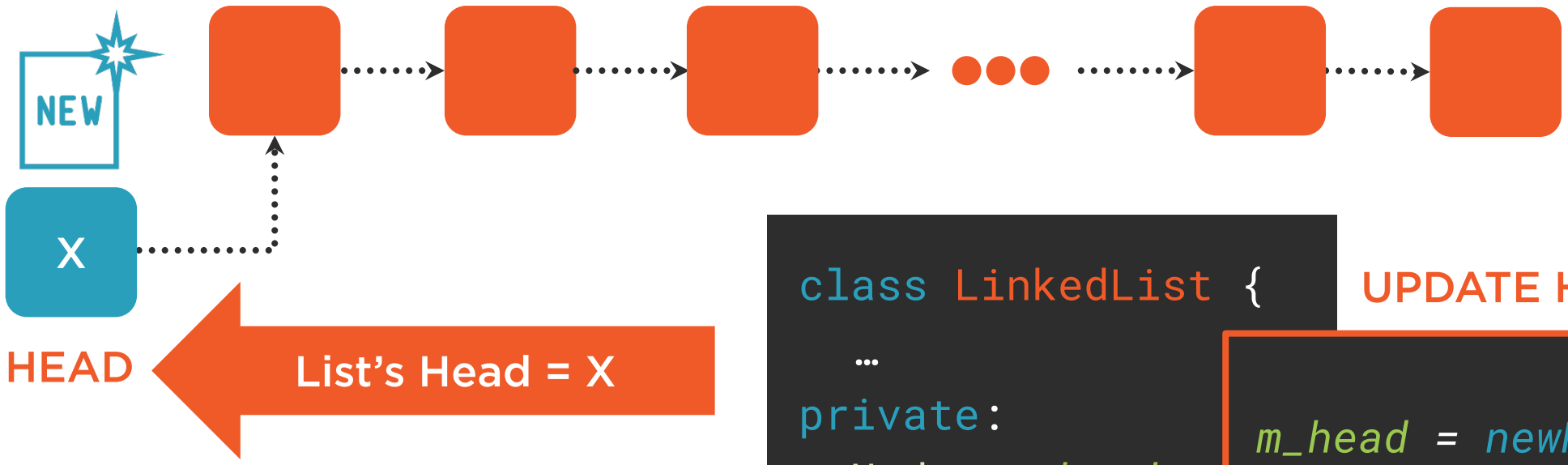


Special Case: Head Insertion



Special Case: Head Insertion

InsertHead(X)

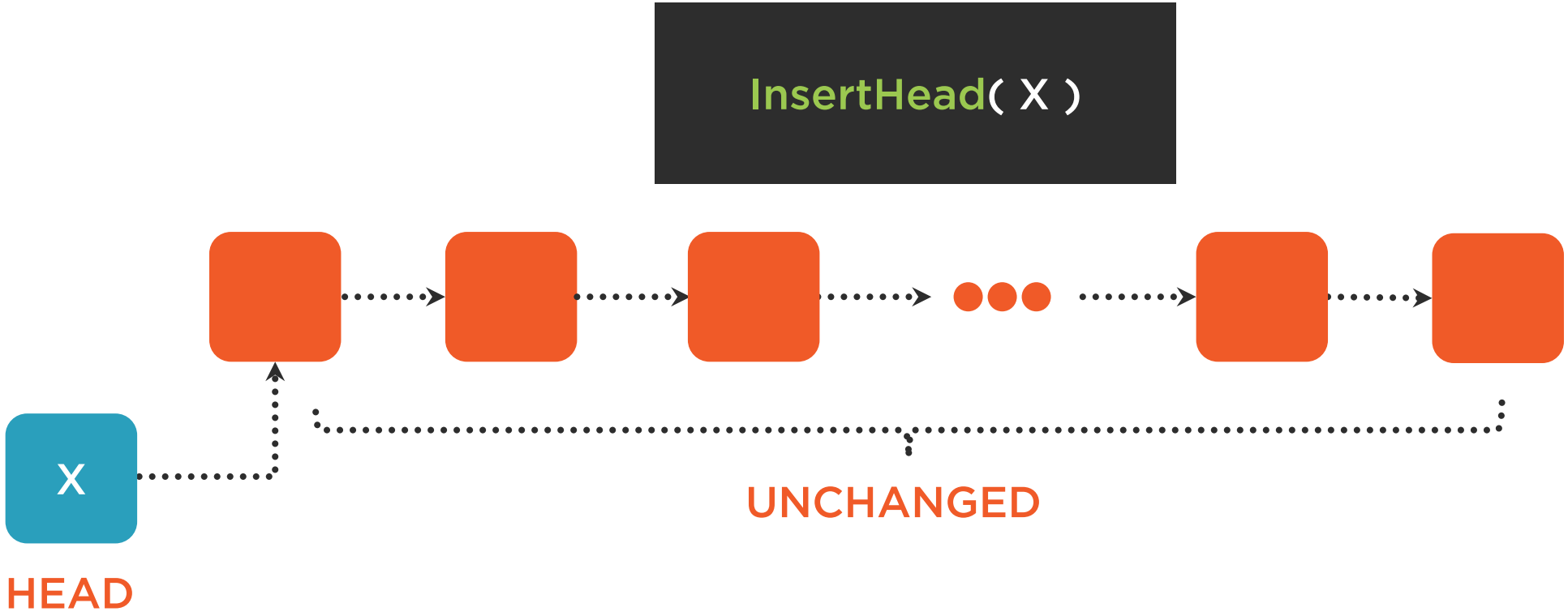



```
class LinkedList {
  …
private:
  Node* m_head;
};
```

UPDATE HEAD MEMBER

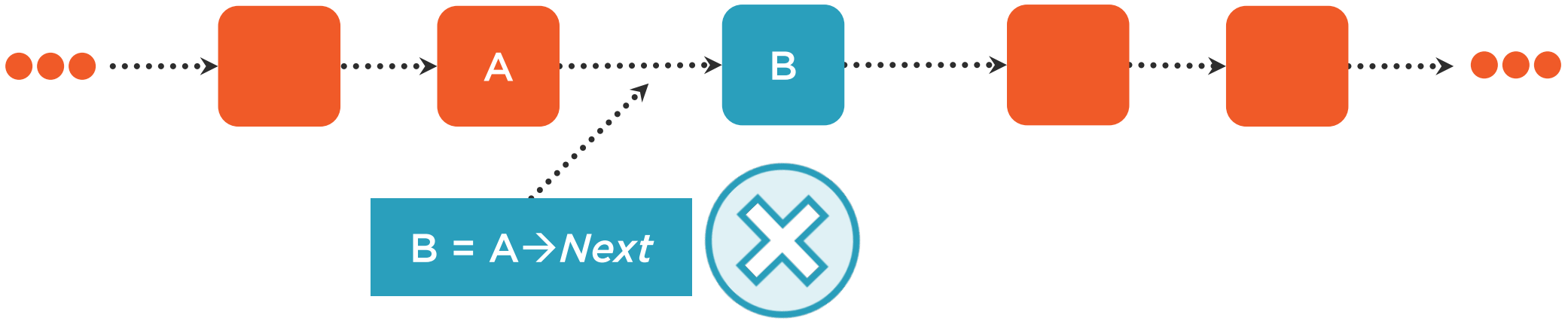
```
m_head = newNode;
```

Special Case: Head Insertion



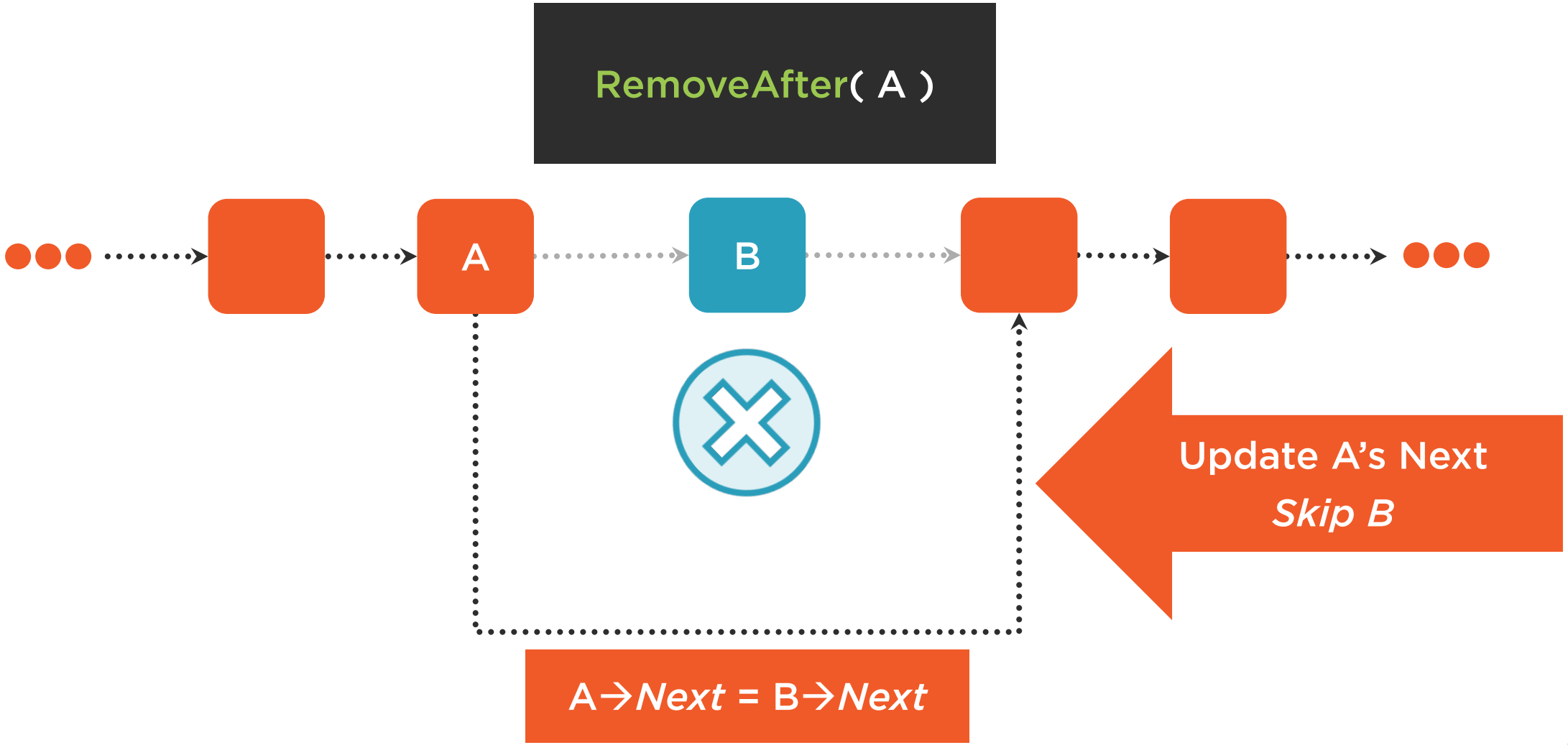
Removing a Node

RemoveAfter(A)

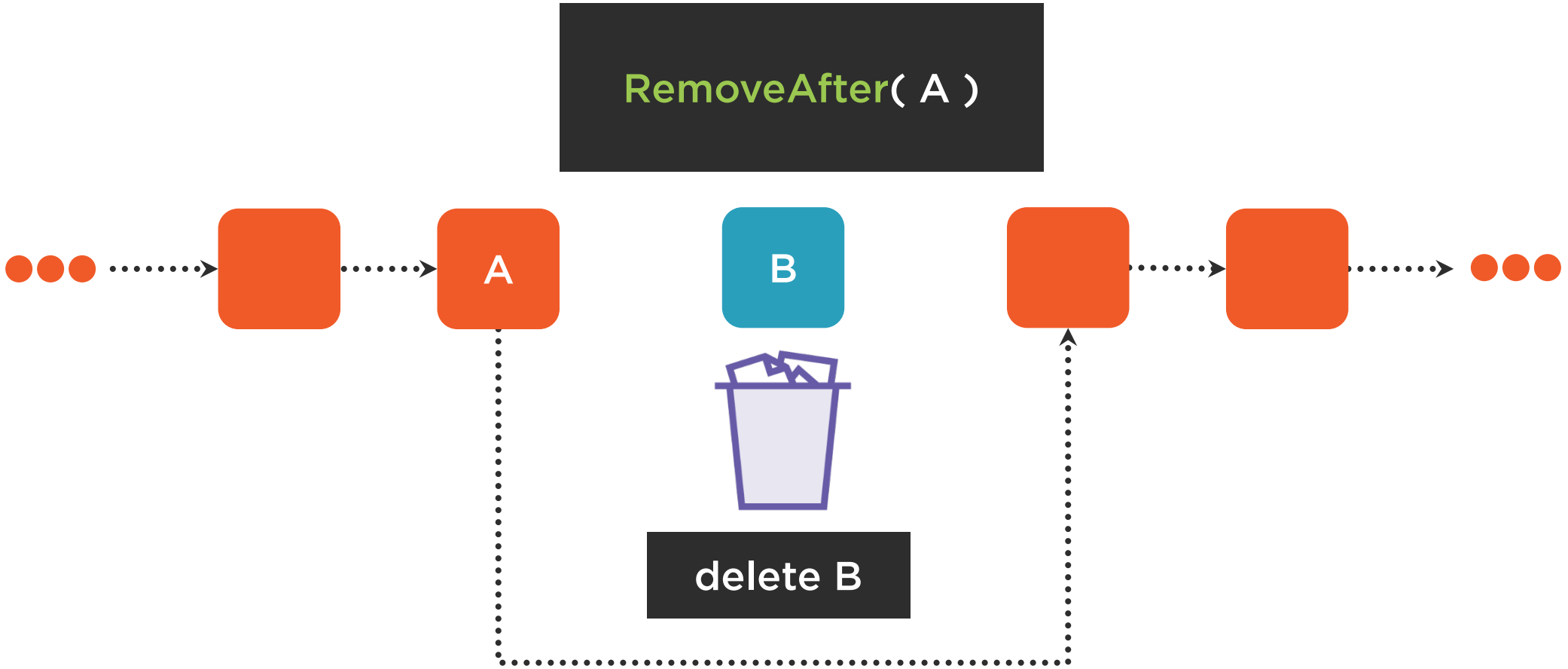


REWIRE NODE POINTERS

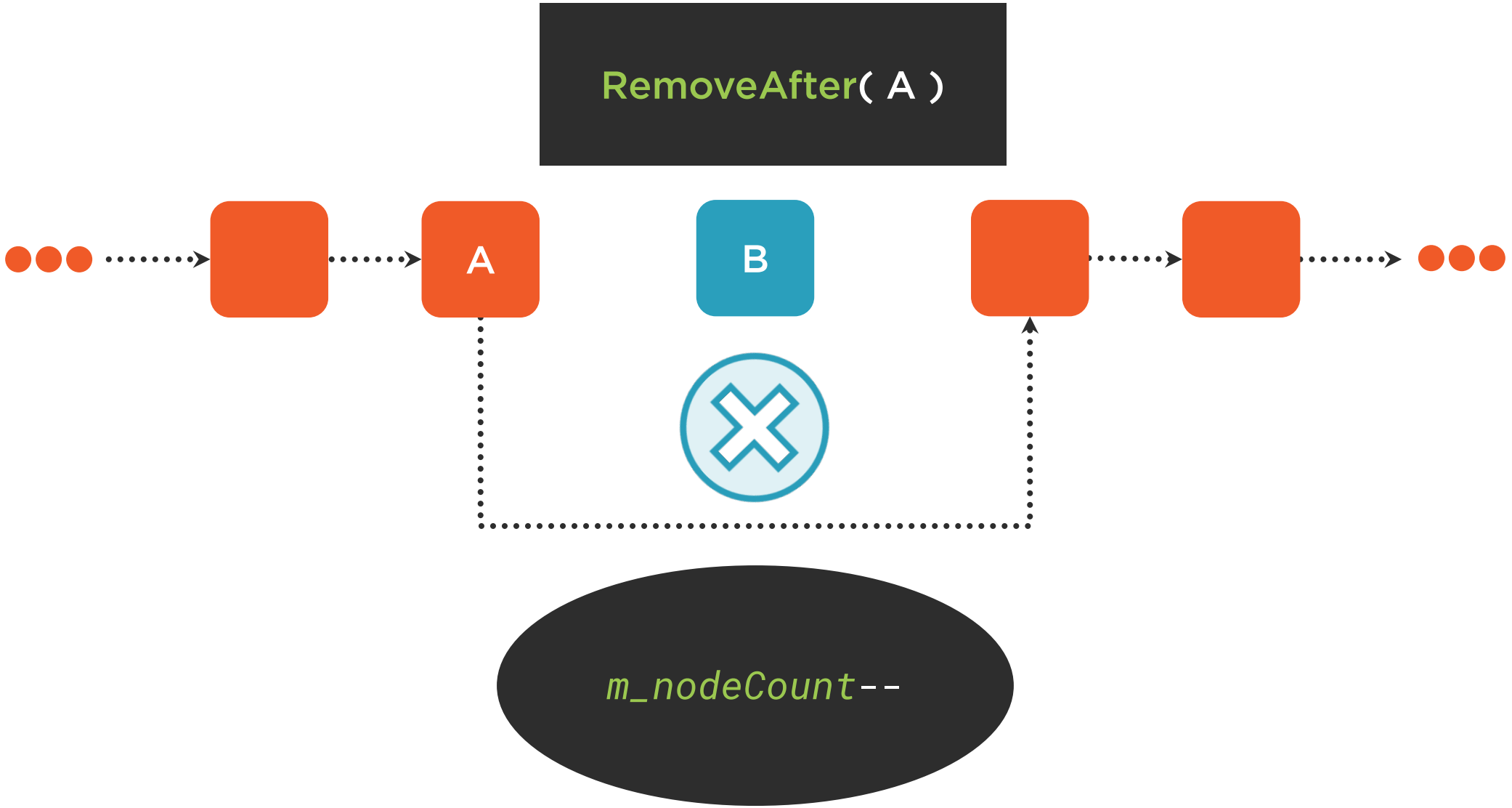
Removing a Node



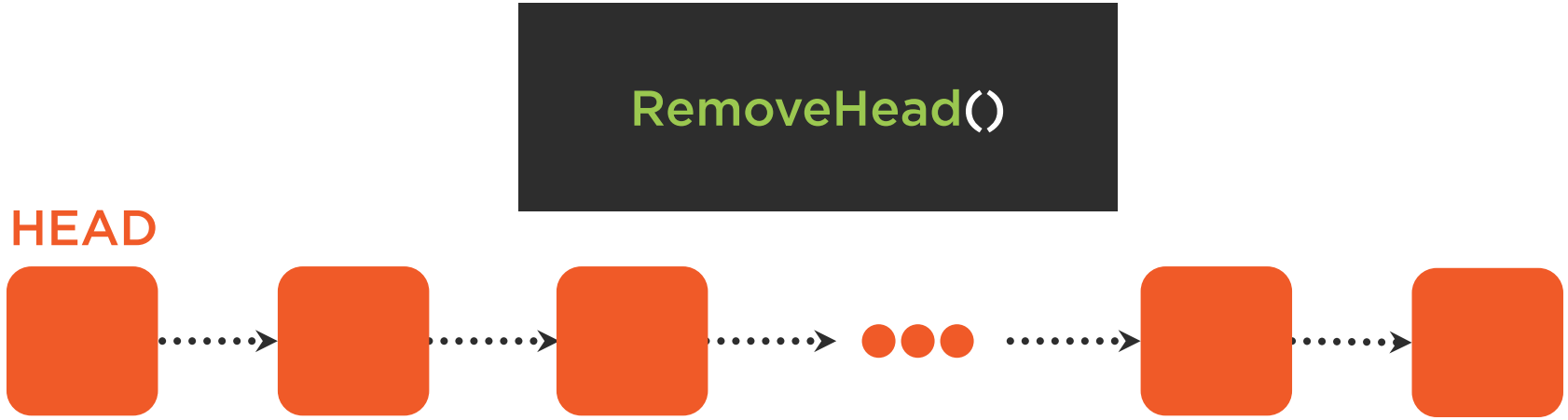
Removing a Node



Removing a Node

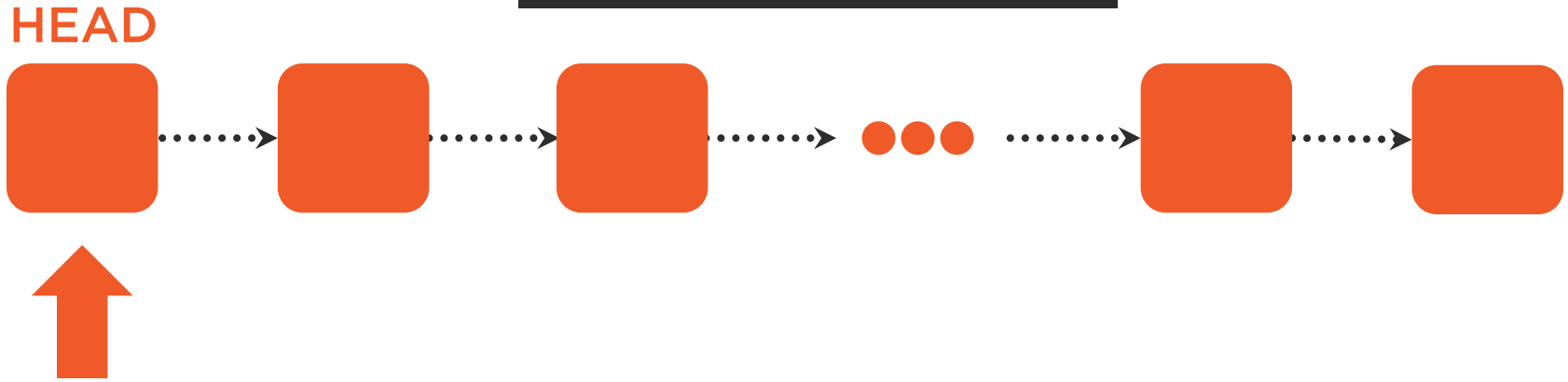


Special Case: Head Removal



Special Case: Head Removal

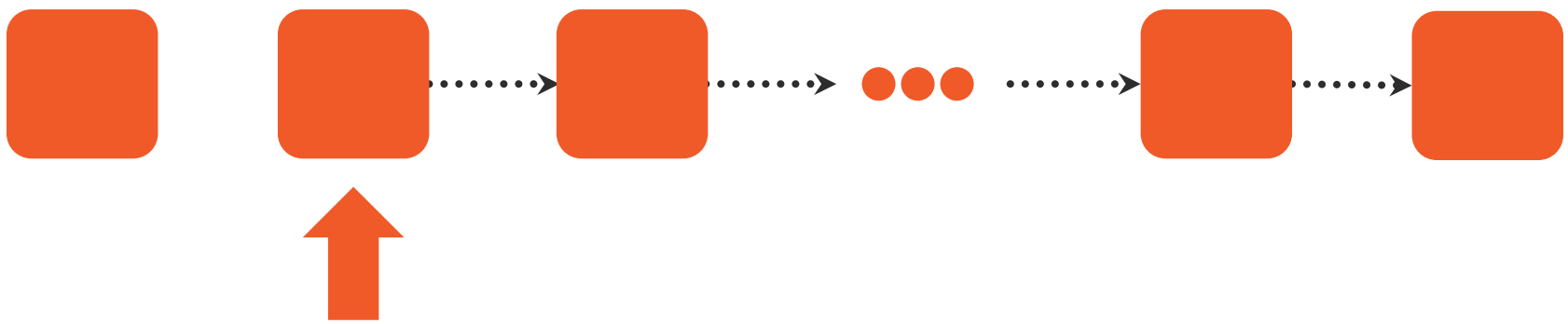
RemoveHead()




```
Node* oldHead = m_head;
```

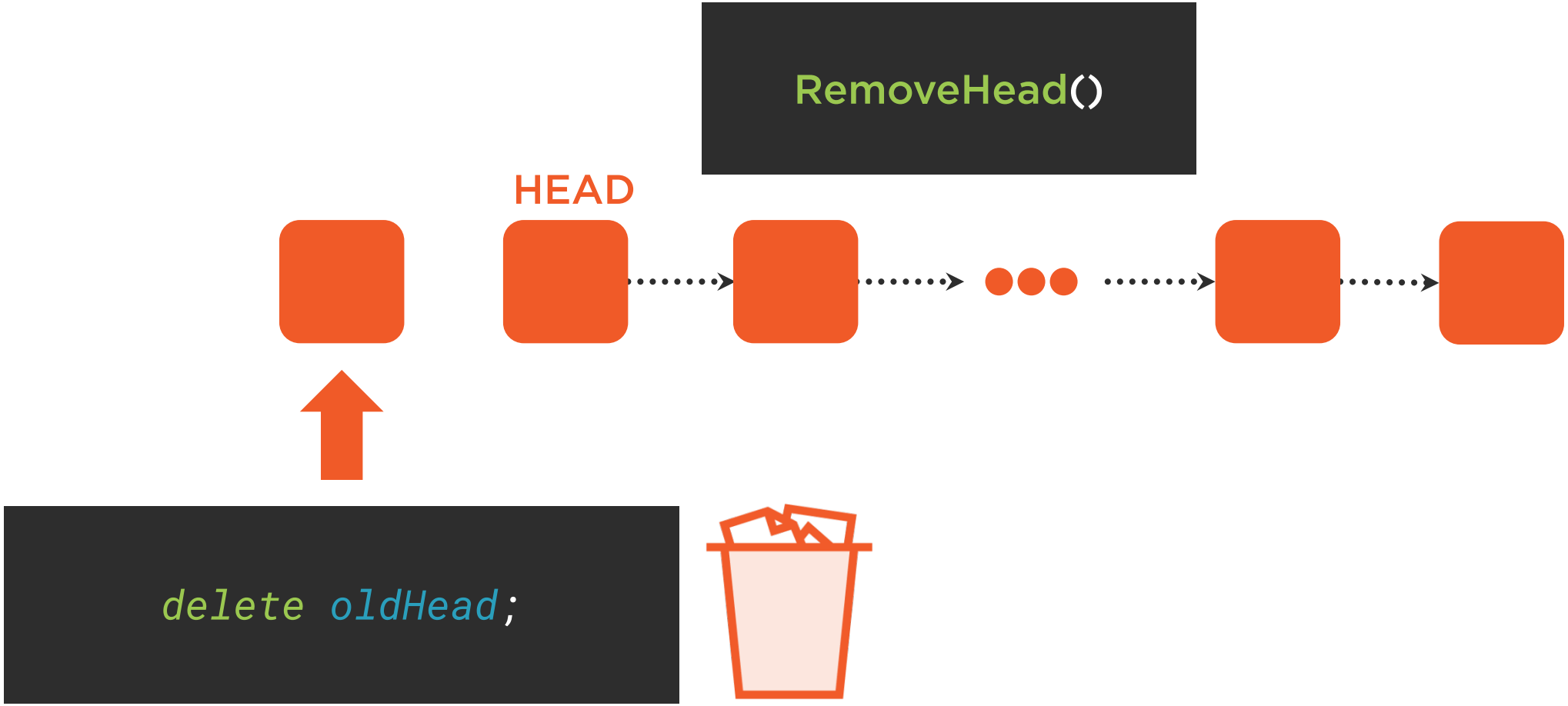
Special Case: Head Removal

RemoveHead()



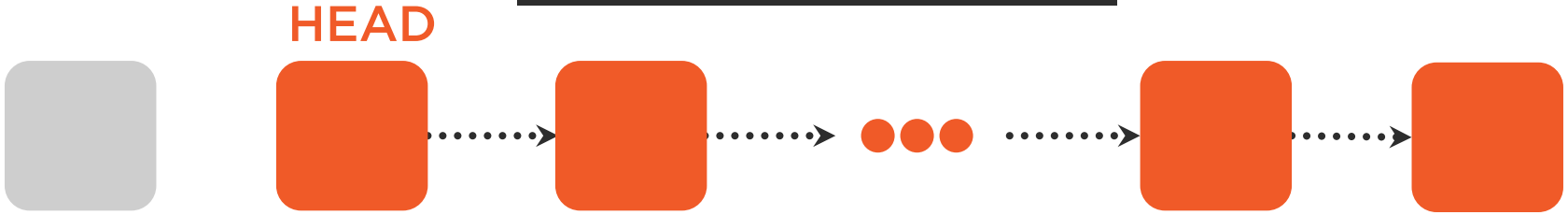

```
m_head = oldHead->Next;
```

Special Case: Head Removal



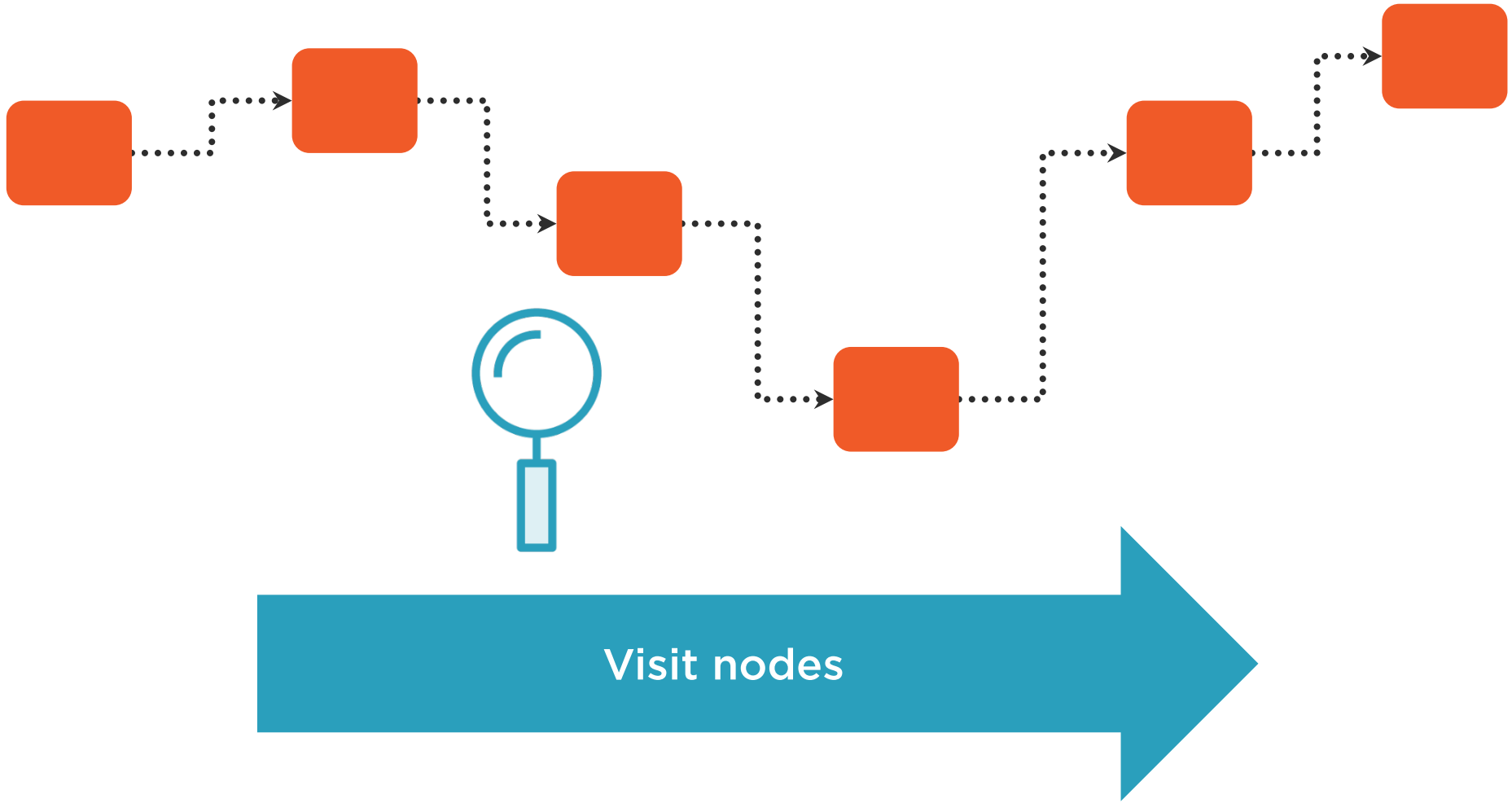
Special Case: Head Removal

RemoveHead()

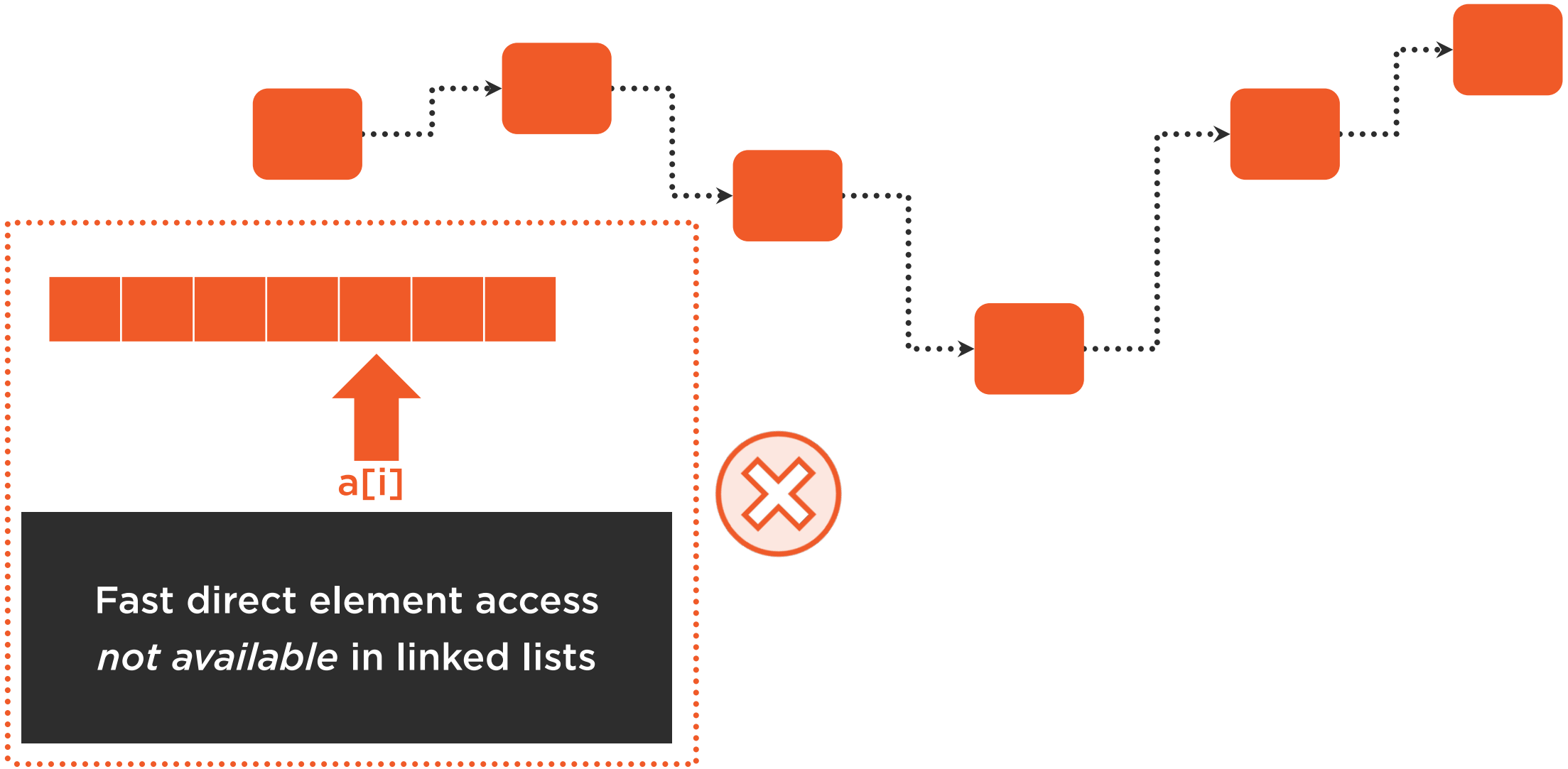



```
m_nodeCount--;
```

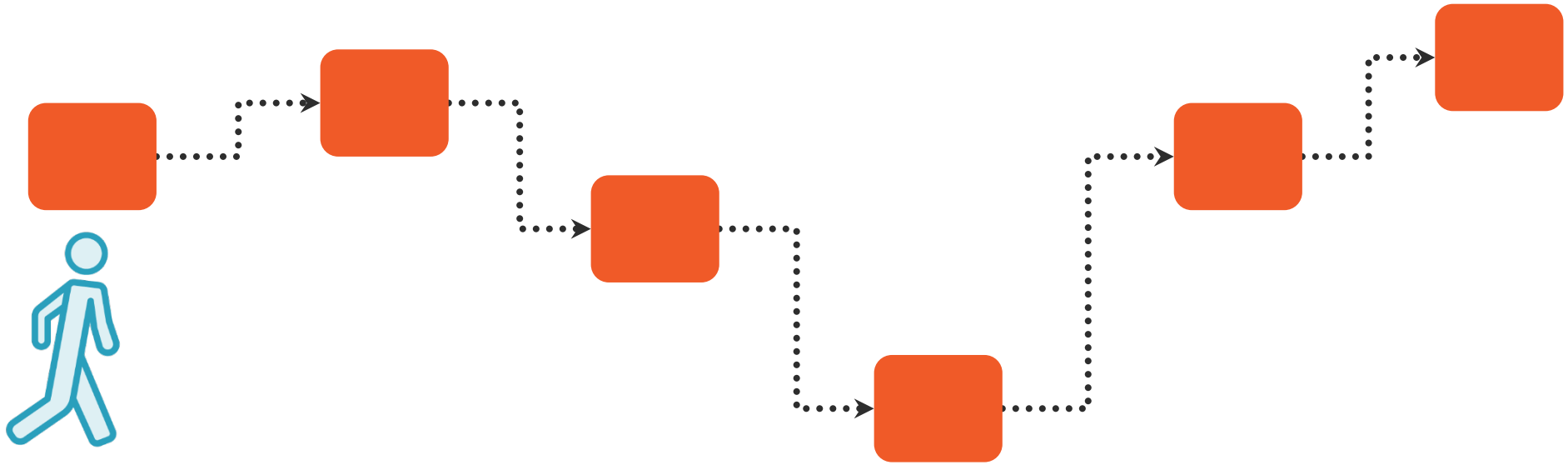
Traversing a Linked List



Traversing a Linked List

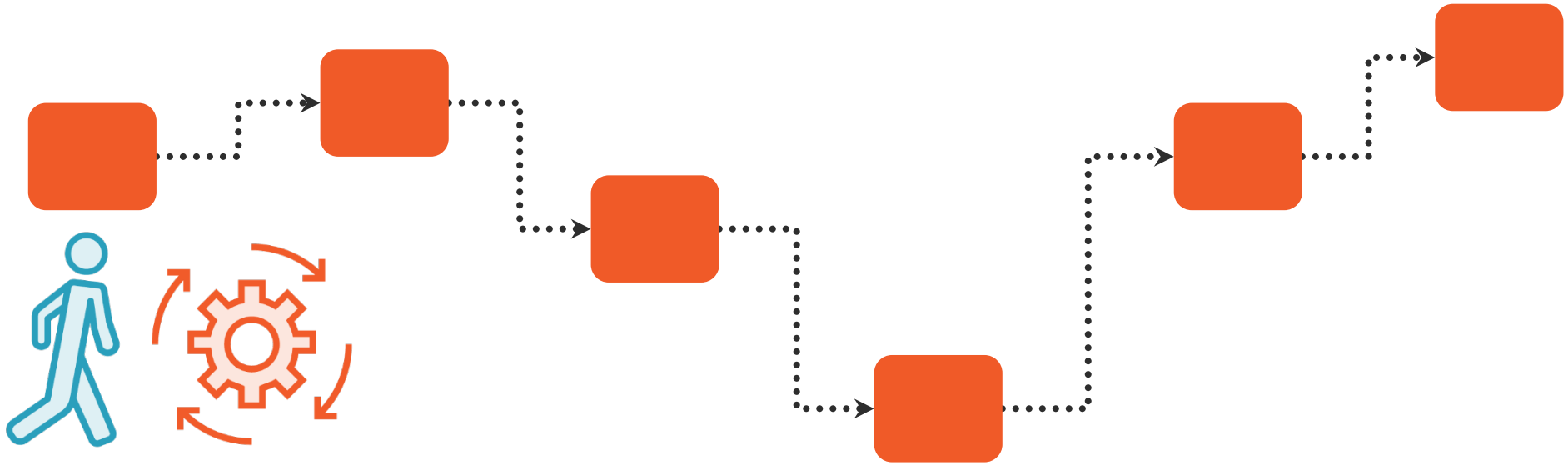


Traversing a Linked List



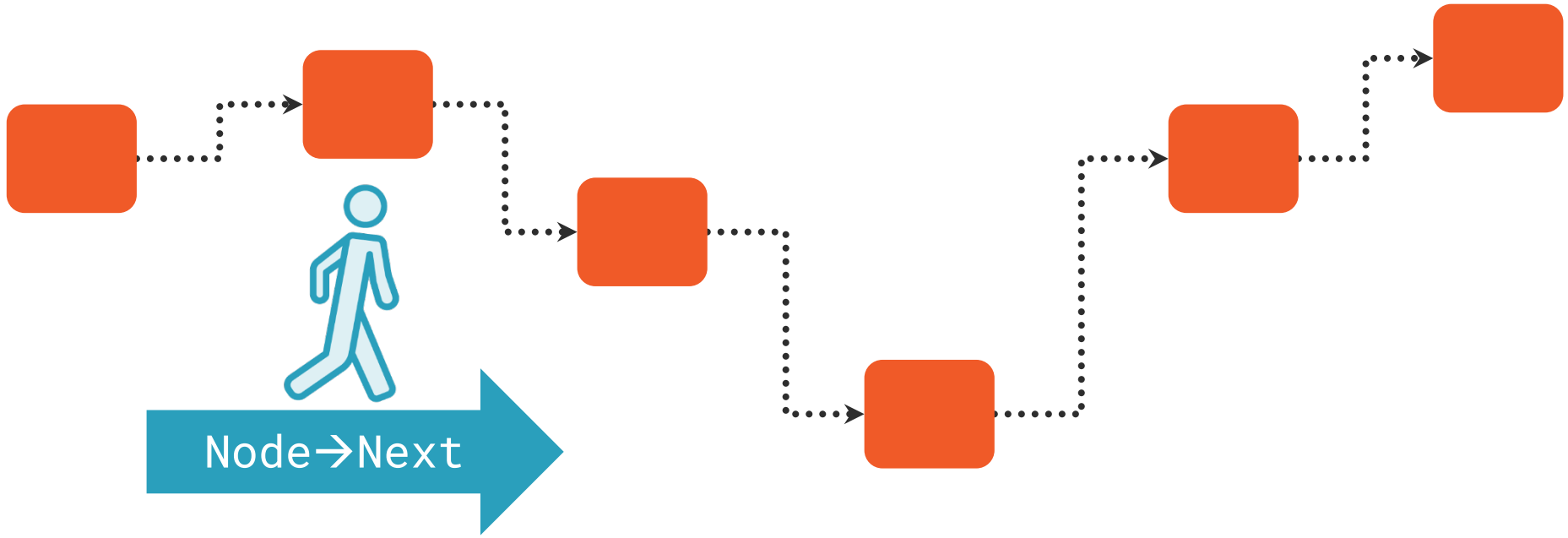
```
Node* current = head;
```

Traversing a Linked List



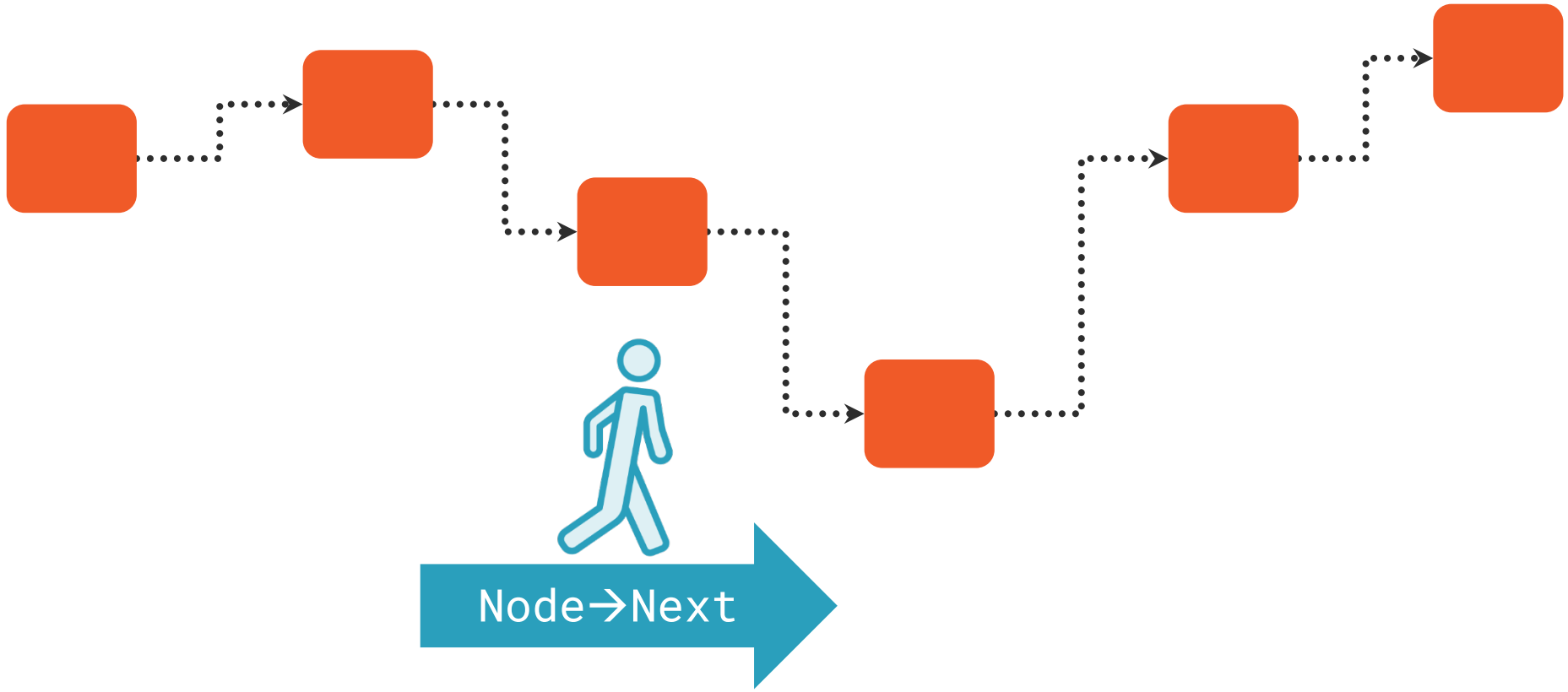
```
// Process current node
```

Traversing a Linked List

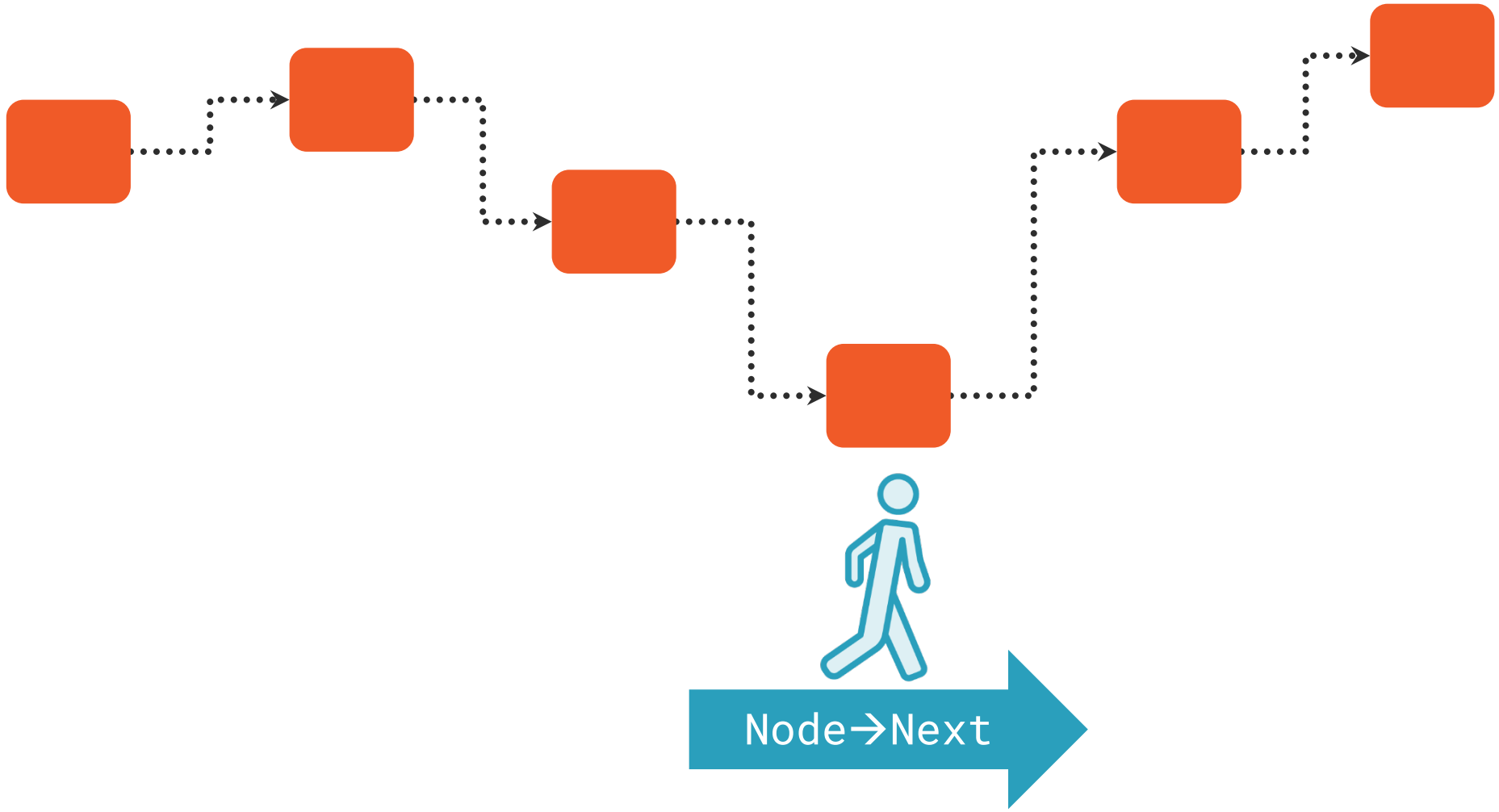



```
current = current->Next;
```

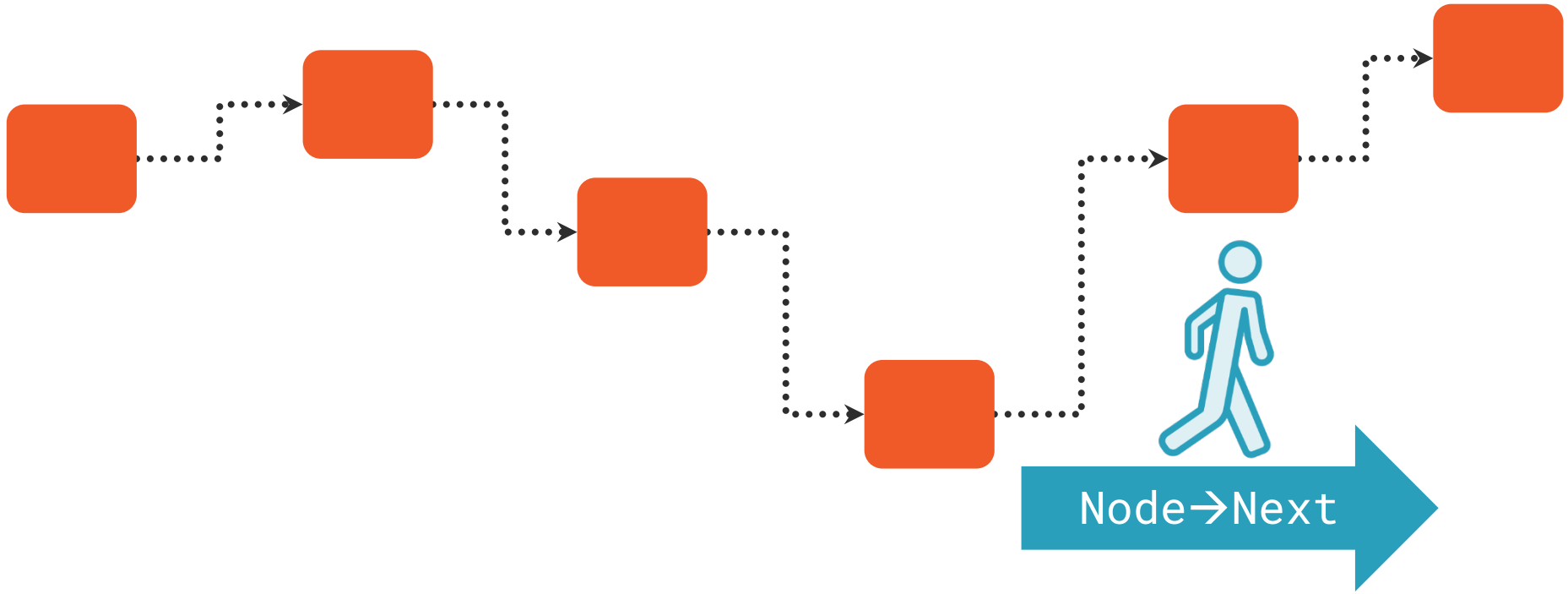
Traversing a Linked List



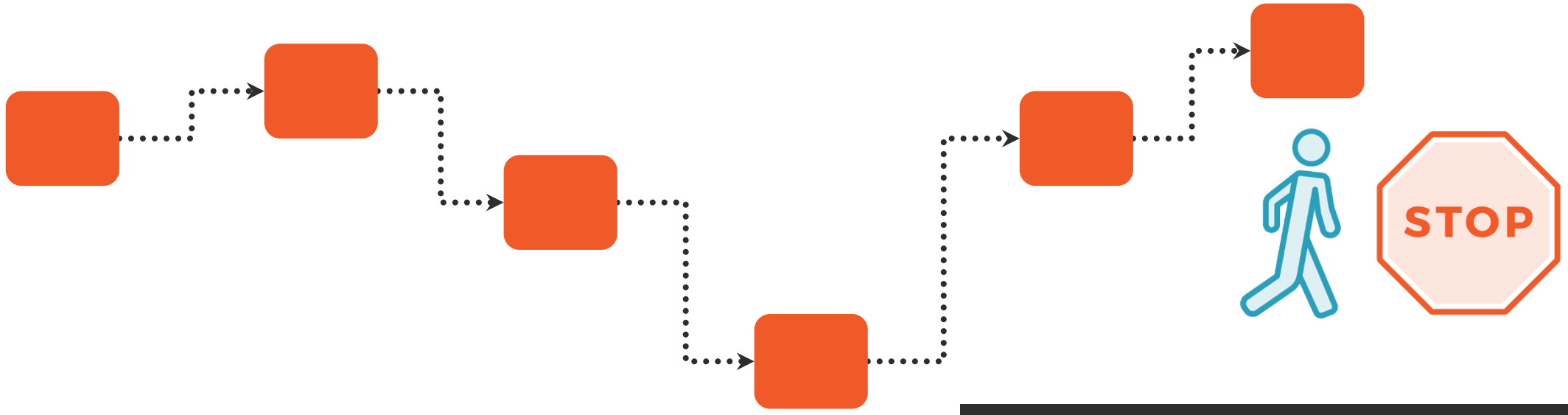
Traversing a Linked List



Traversing a Linked List

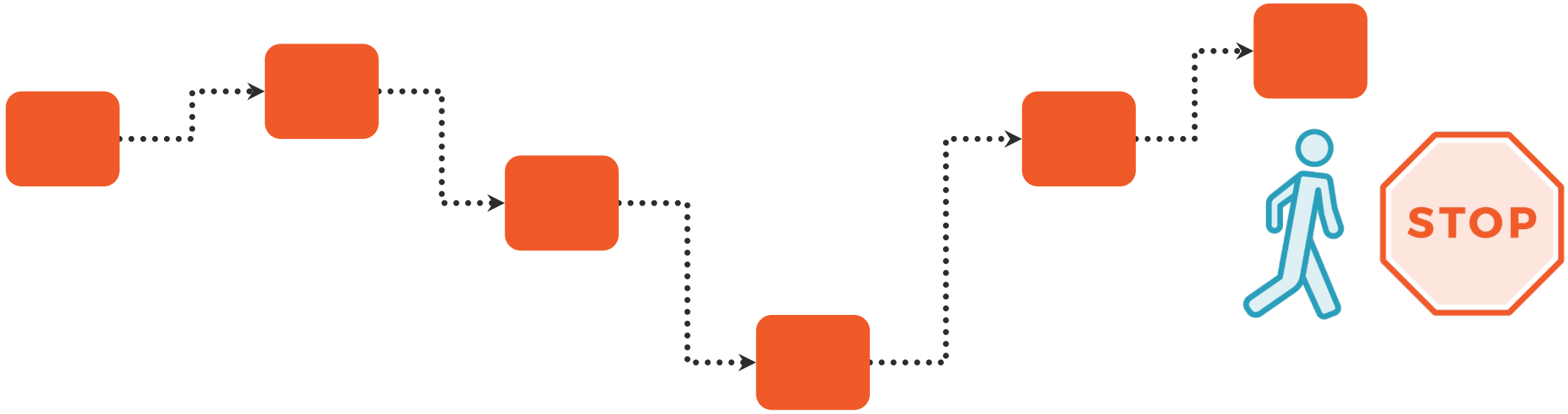


Traversing a Linked List



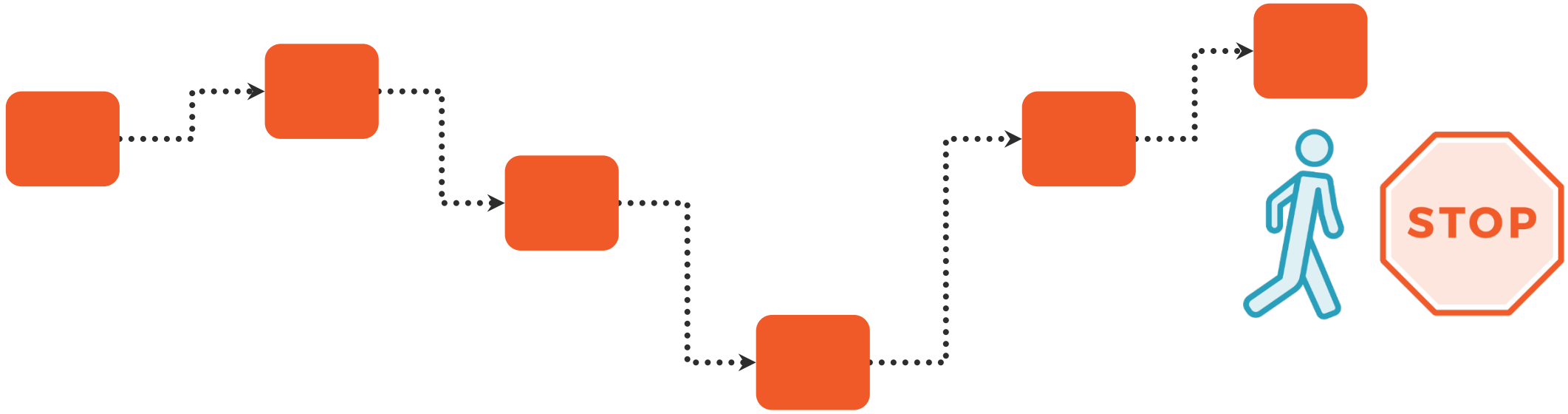

```
current->Next == nullptr
```

Traversing a Linked List



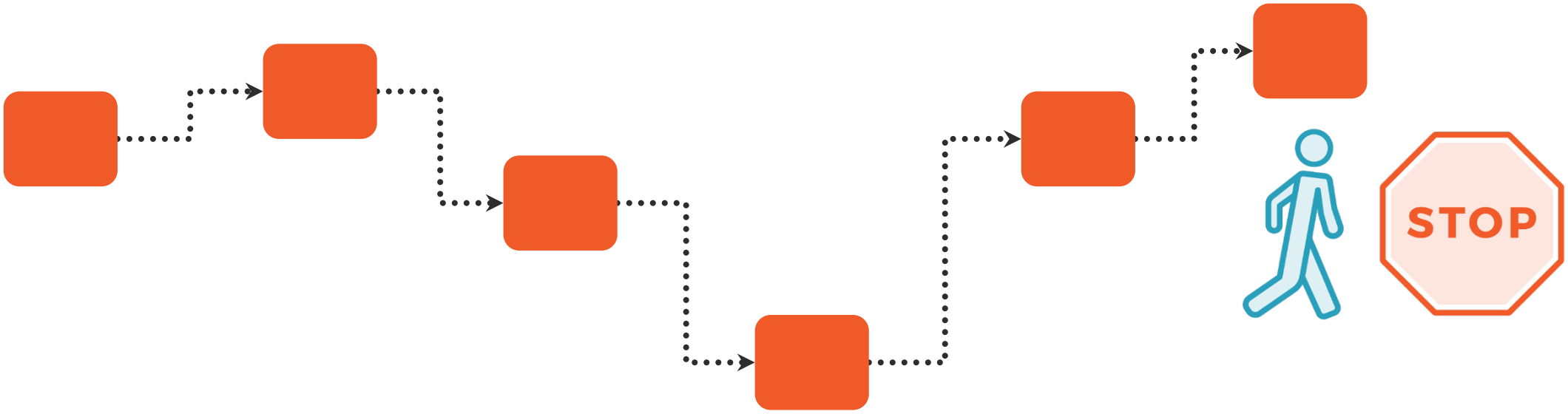

```
while (node != nullptr) {
  // Process current node
  …
  node = node->Next;
}
```

Traversing a Linked List



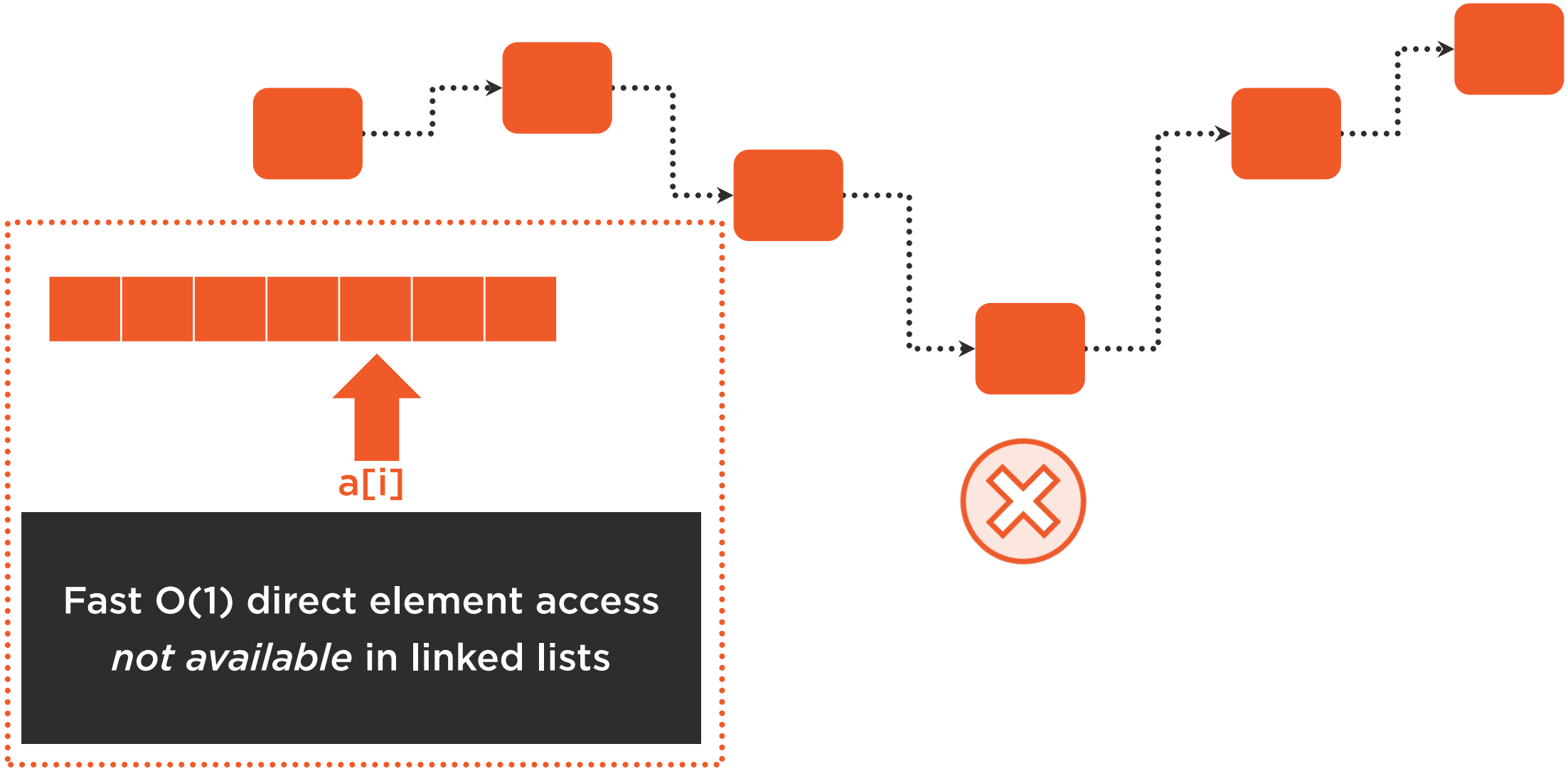

```
while (node != nullptr) {
  // Process current node
  …
  node = node->Next;
}
```

Traversing a Linked List



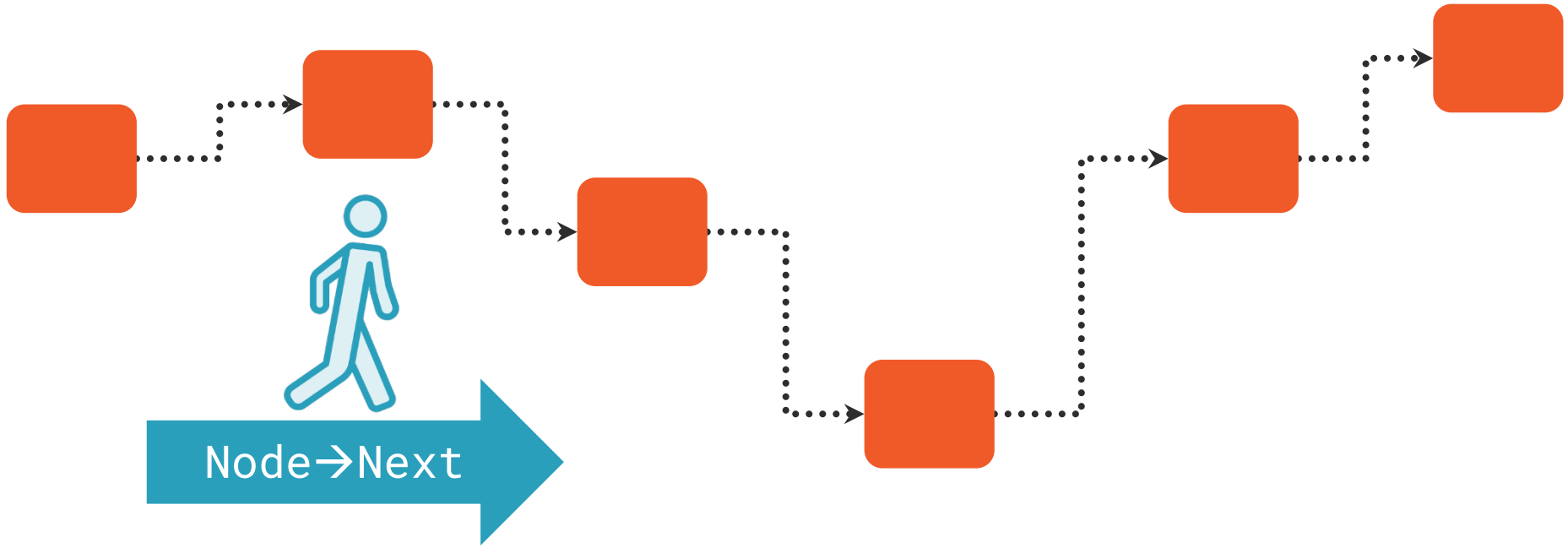

```
while (node != nullptr) {
  // Process current node
  …
  node = node->Next;
}
```

Accessing i-th Node



Fast O(1) direct element access
not available in linked lists

Accessing i-th Node



O(*N*)

Summary

Introduction to linked lists

Fundametal operations

- Node insertion and removal
- Traversing a linked list

C++ implementation code

Thank You!