CSE 250
Data Structures

Dr. Eric Mikida
epmikida@buffalo.edu
208 Capen Hall

Lec 09: Sequences and Lists

Announcements

- PA1 Implementation due Sunday at midnight
  - Be aware that course staff is not guaranteed to be available after 5PM or on weekends
  - Be thoughtful in your submissions to Autolab

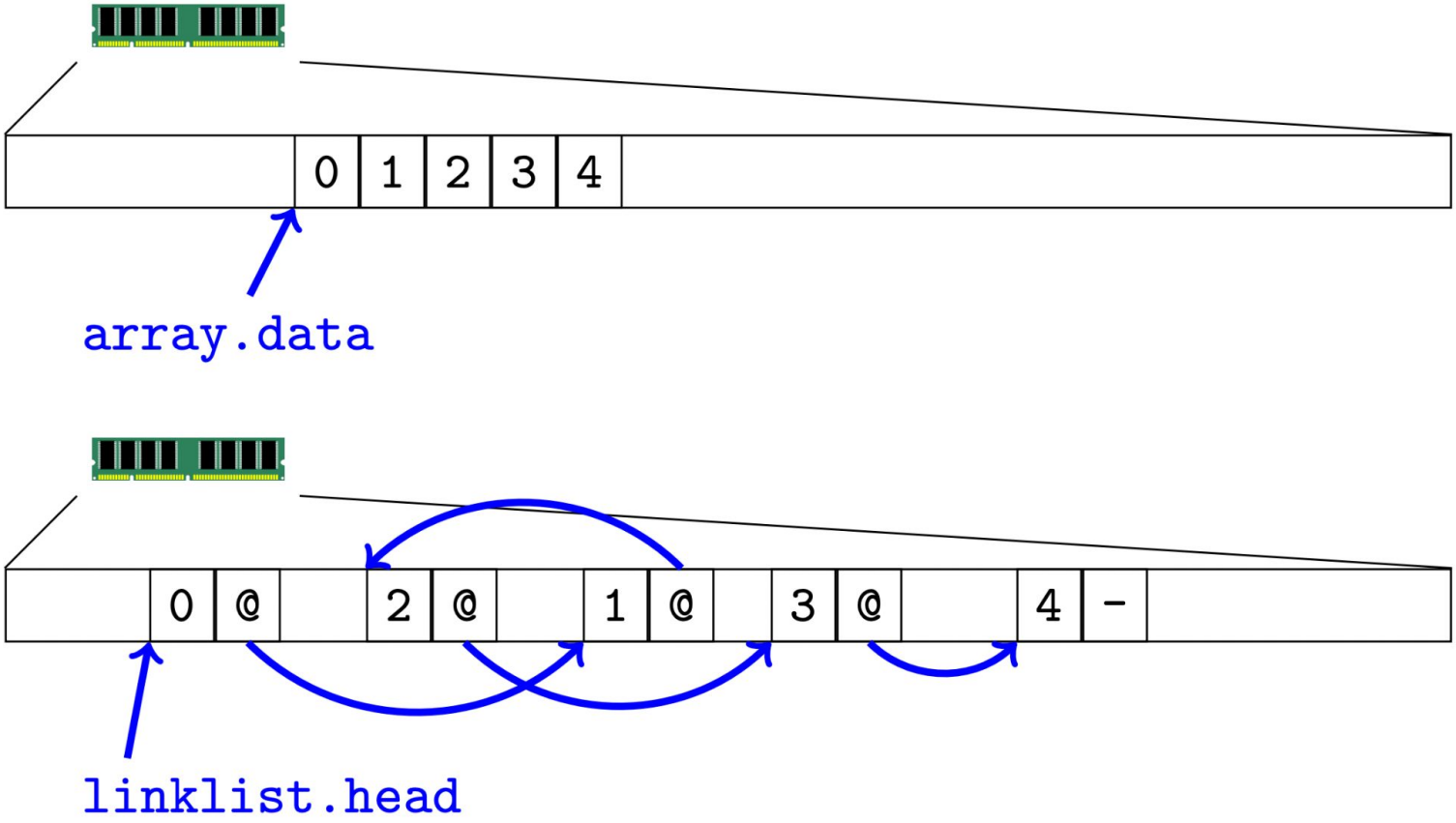
The Sequence ADT

`T get(int idx)`
Get the element (of type **`T`**) at position **`idx`**

`T set(int idx, T value)`
Set the element (of type **`T`**) at position **`idx`** to a new value

`int length`
Get the number of elements in the seq

`Iterator<T> iterator()`
Get access to view all elements in the sequence, in order, once



Arrays and Linked Lists in Memory

Arrays in Detail

How would we implement the methods of the Sequence ADT for an Array:

`T get(int idx)`

Compute the address of the element in constant time

`T set(int idx, T value)`

Compute the address of the element in constant time

`int length`

Access the **`length`** field in constant time

Arrays in Detail

How would we implement the methods of the Sequence ADT for an Array:

`T get(int idx)`

Compute the address of the element in constant time

`T set(int idx, T value)`

Compute the address of the element in constant time

`int length`

Access the **`length`** field in constant time

What is the complexity of these operations?

Arrays in Detail

How would we implement the methods of the Sequence ADT for an Array:

`T get(int idx)`

Compute the address of the element in constant time $\Theta(1)$

`T set(int idx, T value)`

Compute the address of the element in constant time $\Theta(1)$

`int length`

Access the **`length`** field in constant time $\Theta(1)$

What is the complexity of these operations?

Linked Lists in Detail

How do we implement the methods of the Sequence ADT for a Linked List:

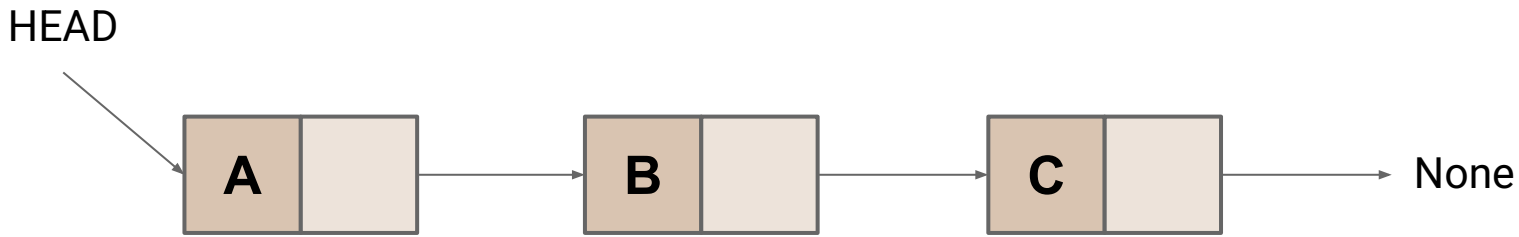
```
T get(int idx)

T set(int idx, T value)

int length
```

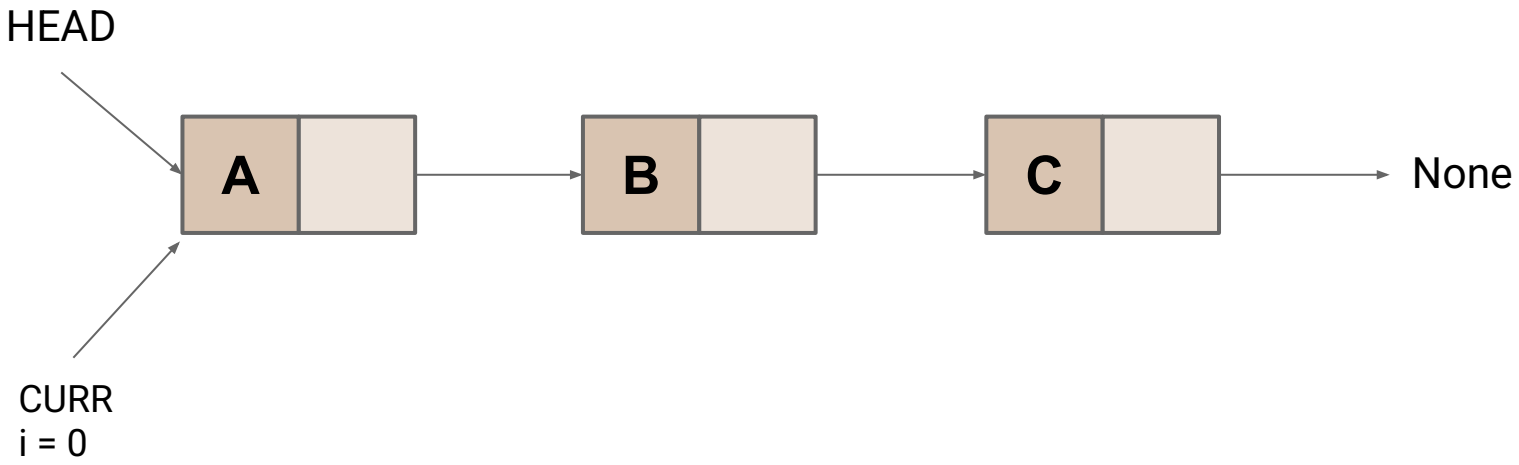
Implementing `get/set`

`get(2)`



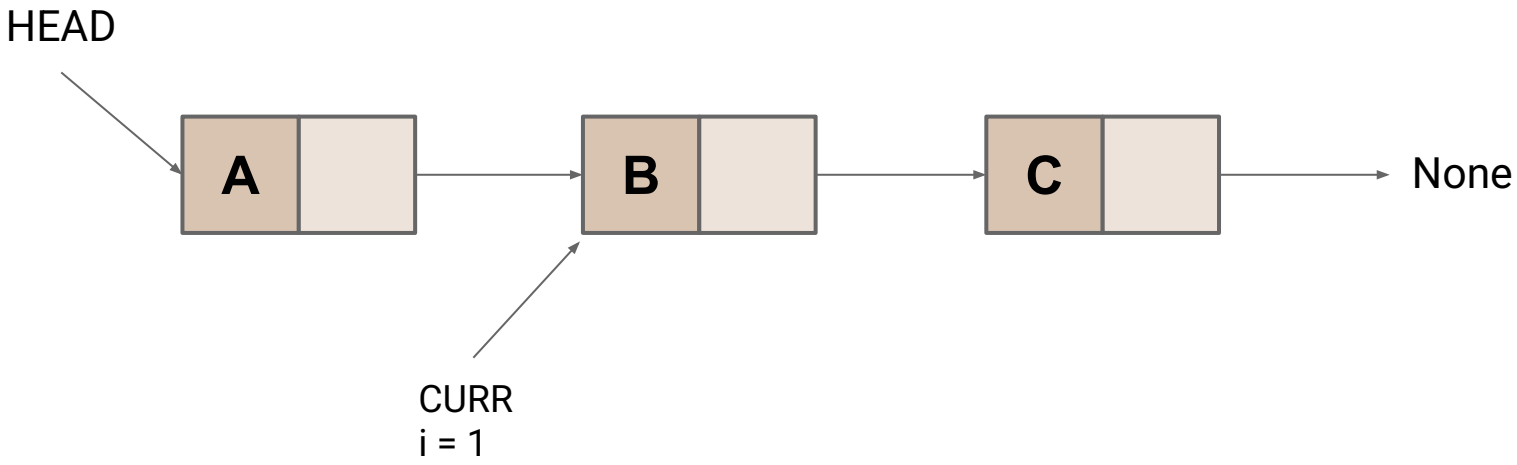
Implementing get/set

`get(2)`

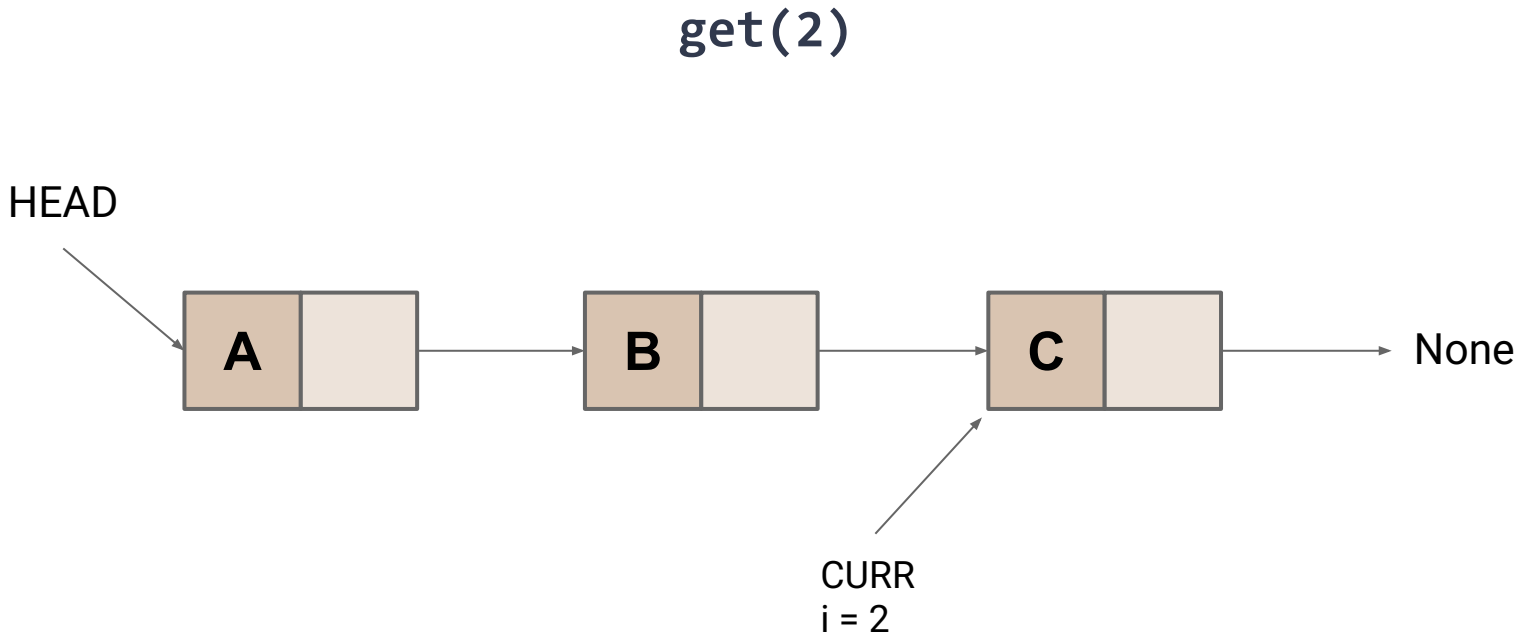


Implementing get/set

`get(2)`



Implementing get/set



Implementing get/set

```
1  public T get(int idx) {
2    int i = 0;
3    Optional<LinkedListNode<T>> curr = head;
4    while(i < idx) {
5      if (!curr.isPresent()) { throw new IndexOutOfBoundsException(); }
6      i++;
7      curr = curr.get().next;
8    }
9    if(!curr.isPresent()) { throw new IndexOutOfBoundsException(); }
10   return curr.get().value;
11 }
```

Implementing get/set

```
1  public T get(int idx) {
2    int i = 0;
3    Optional<LinkedListNode<T>> curr = head;
4    while(i < idx) {
5      if (!curr.isPresent()) { throw new IndexOutOfBoundsException(); }
6      i++;
7      curr = curr.get().next;
8    }
9    if(!curr.isPresent()) { throw new IndexOutOfBoundsException(); }
10   return curr.get().value;
11 }
```

All of this is $\Theta(1)$

Implementing get/set

```
1  public T get(int idx) {
2    Θ(1)
3    while(i < idx) {
4      Θ(1)
5    }
6    Θ(1)
7  }
```

Implementing get/set

```
1  public T get(int idx) {
2    Θ(1)
3    Θ(idx)
4    Θ(1)
5  }
```

Complexity: $\Theta(idx) \subset O(n)$

Linked Lists in Detail

How do we implement the methods of the Sequence ADT for a Linked List:

`T get(int idx)`

Go node-by-node until you reach **`idx`** $\Theta(idx) \subset O(n)$

`T set(int idx, T value)`

Go node-by-node until you reach **`idx`** $\Theta(idx) \subset O(n)$

`int length`

Implementing `length`

```
1  public int length() {
2      int i = 0;
3      Optional<LinkedListNode<T>> curr = head;
4      while(curr.isPresent()) { i++; curr = curr.get().next; }
5      return i;
6  }
```

Implementing `length`

```
1  public int length() {
2      Θ(1)
3      while(curr.isPresent()) { Θ(1) }
4      Θ(1)
5  }
```

Implementing `length`

```
1  public int length() {
2      Θ(1)
3      Θ(n)
4      Θ(1)
5  }
```

Complexity: $\Theta(n)$
Can we do better?

Implementing `length`

Idea: Have the Linked List class store the length

```
1  class LinkedList<T> {
2    Optional<LinkedListNode<T>> head = Optional.empty();
3    int length;
4    /* ... */
5  }
```

Now complexity of getting **`length`** is $\Theta(1)$

Implementing `length`

Idea: Have the Linked List class store the length

```
1  class LinkedList<T> {
2    Optional<LinkedListNode<T>> head = Optional.empty();
3    int length;
4    /* ... */
5  }
```

Now complexity of getting **`length`** is $\Theta(1)$

How much extra space is required? $\Theta(1)$

How much extra work is required to insert/remove? $\Theta(1)$

Implementing `length`

Idea: Have the Linked List class store the length

```
1  class LinkedList<T> {
2    0
3    i
4    /
5  }
```

Common trade-off: Sometimes storing extra information can decrease complexity!

Now complexity of getting `length` is $\Theta(1)$

How much extra space is required? $\Theta(1)$

How much extra work is required to insert/remove? $\Theta(1)$

Access by-Reference vs by-Index

Complexity of getting the value of the ***n***th node in a Linked List?

Complexity of getting the value of the ***n***th node if we have a reference to that node?

Complexity of getting the value of (***n*+1**)th node if we have a reference to the ***n***th node?

Complexity of getting the value of (***n*-1**)th node if we have a reference to the ***n***th node?

Access by-Reference vs by-Index

Complexity of getting the value of the ***n***th node in a Linked List? $\Theta(n)$

Complexity of getting the value of the ***n***th node if we have a reference to that node? $\Theta(1)$

Complexity of getting the value of (***n***+1)th node if we have a reference to the ***n***th node? $\Theta(1)$

Complexity of getting the value of (***n***-1)th node if we have a reference to the ***n***th node? $\Theta(n)$

Sequence Runtimes (so far…)

	Array	Linked List (by index)	Linked List (by reference)
`get(...)`	$\Theta(1)$	$\Theta(idx)$ or $O(n)$	$\Theta(1)$
`set(...)`	$\Theta(1)$	$\Theta(idx)$ or $O(n)$	$\Theta(1)$
`size()`	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$

The Sequence ADT

```
1 public interface Sequence<E> {
2   public E get(int idx);
3   public void set(int idx, E value);
4   public int size();
5   public Iterator<E> iterator();
6 }
```

What about adding/removing elements?

The List ADT

```
1  public interface List<E>
2      extends Sequence<E> { // Everything a sequence has, and...
3    /** Extend the sequence with a new element at the end */
4    public void add(E value);
5
6    /** Extend the sequence by inserting a new element */
7    public void add(int idx, E value);
8
9    /** Remove the element at a given index */
10   public void remove(int idx);
11 }
```

Lists in Other Languages

Java, Python: `List`, `list`

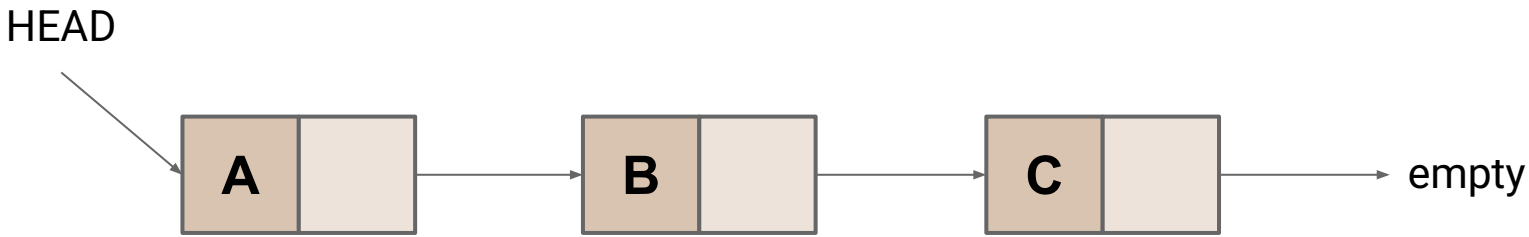
C++, Rust: `vector`, `Vec`

Scala: `Buffer`

Go: `Slice`

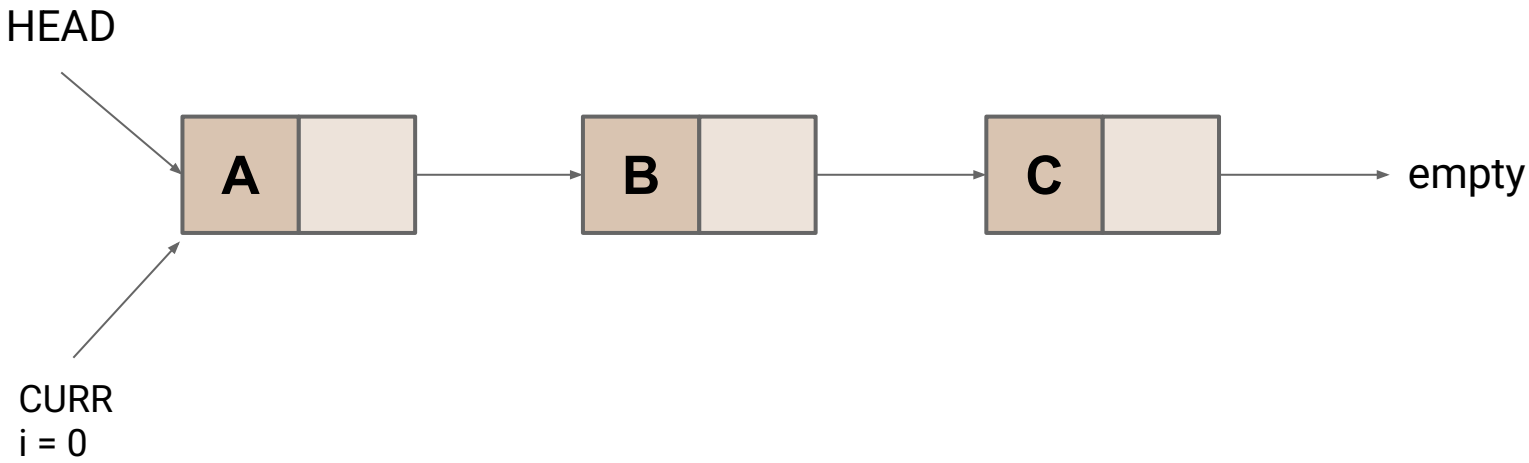
Linked Lists - `add(idx, e)`

`add(2,"D")`



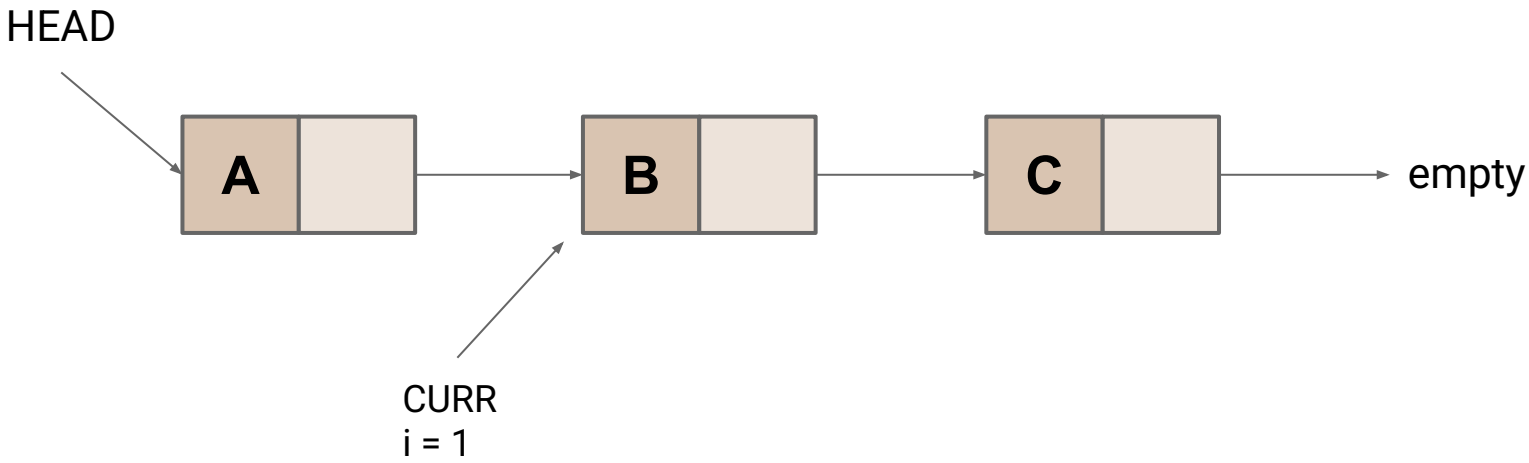
Linked Lists - add(idx, e)

add(2,"D")



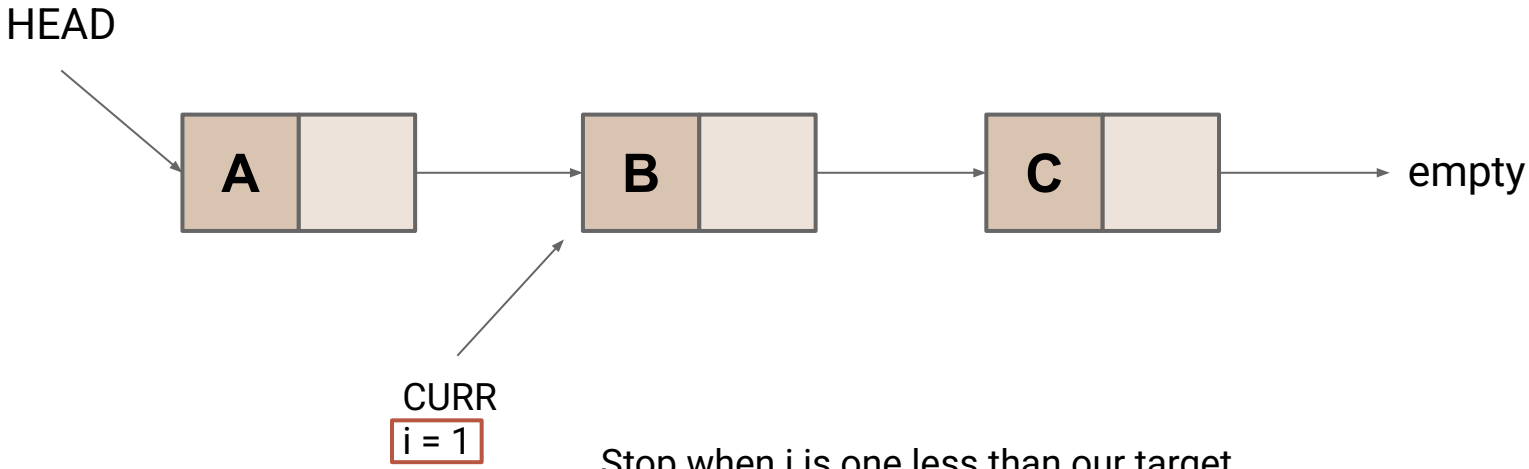
Linked Lists - add(idx, e)

add(2,"D")



Linked Lists - add(idx, e)

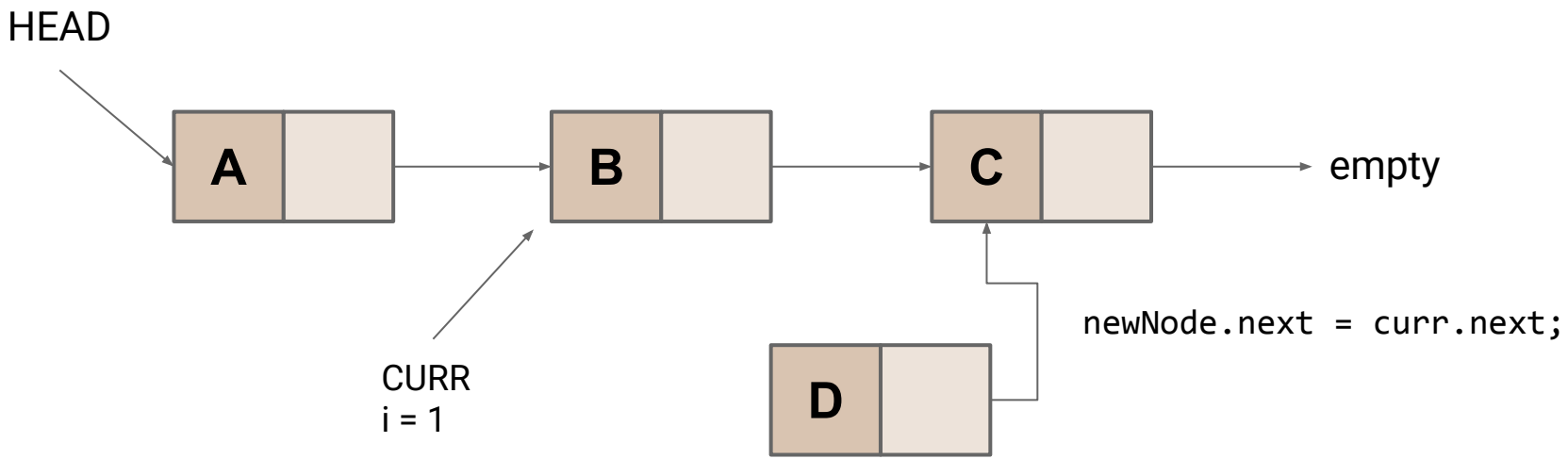
add(2, "D")



Stop when i is one less than our target

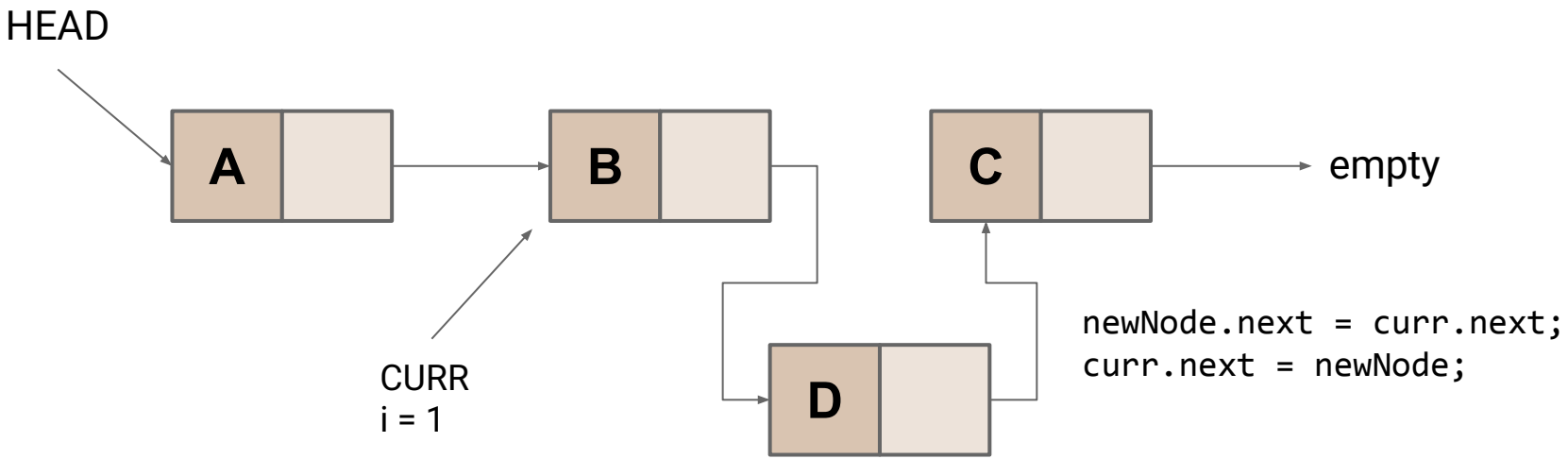
Linked Lists - add(idx, e)

```
add(2,"D")
```



Linked Lists - add(idx, e)

```
add(2,"D")
```



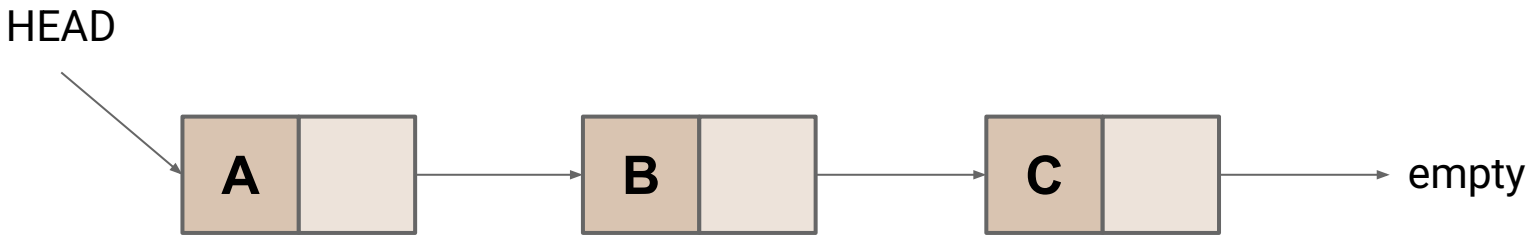
Linked Lists - `add(idx, e)`

1. Find node before **`idx`**: ***O*(*n*)**
2. Allocate a new node and assign its value: ***O*(1)**
3. Set the new nodes **`next`** reference: ***O*(1)**
4. Update the node before **`idx`**'s **`next`** reference: ***O*(1)**

Total: *O*(*n*)

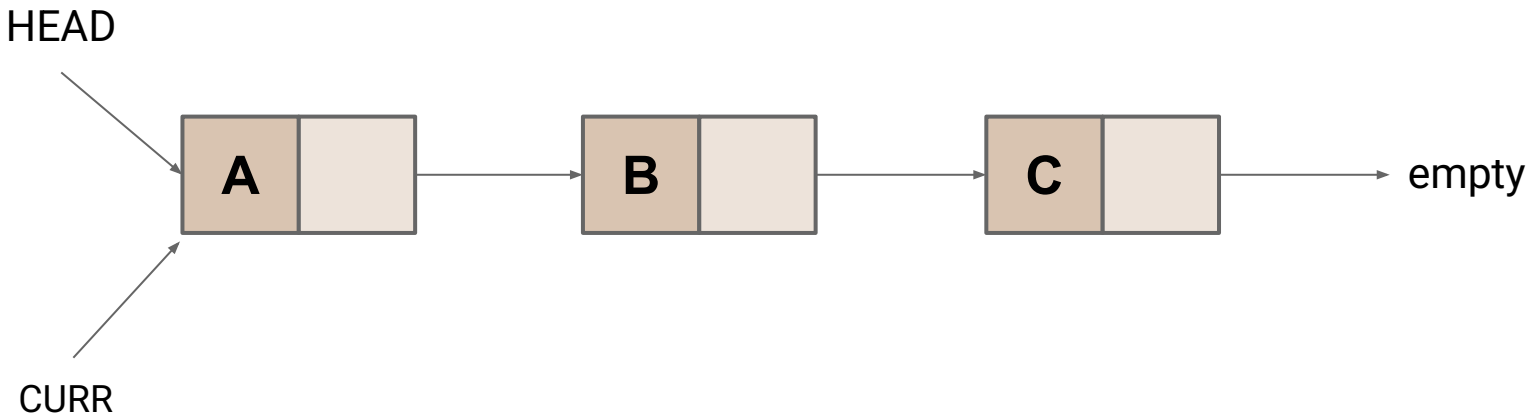
Linked Lists - add(e)

add("Z")

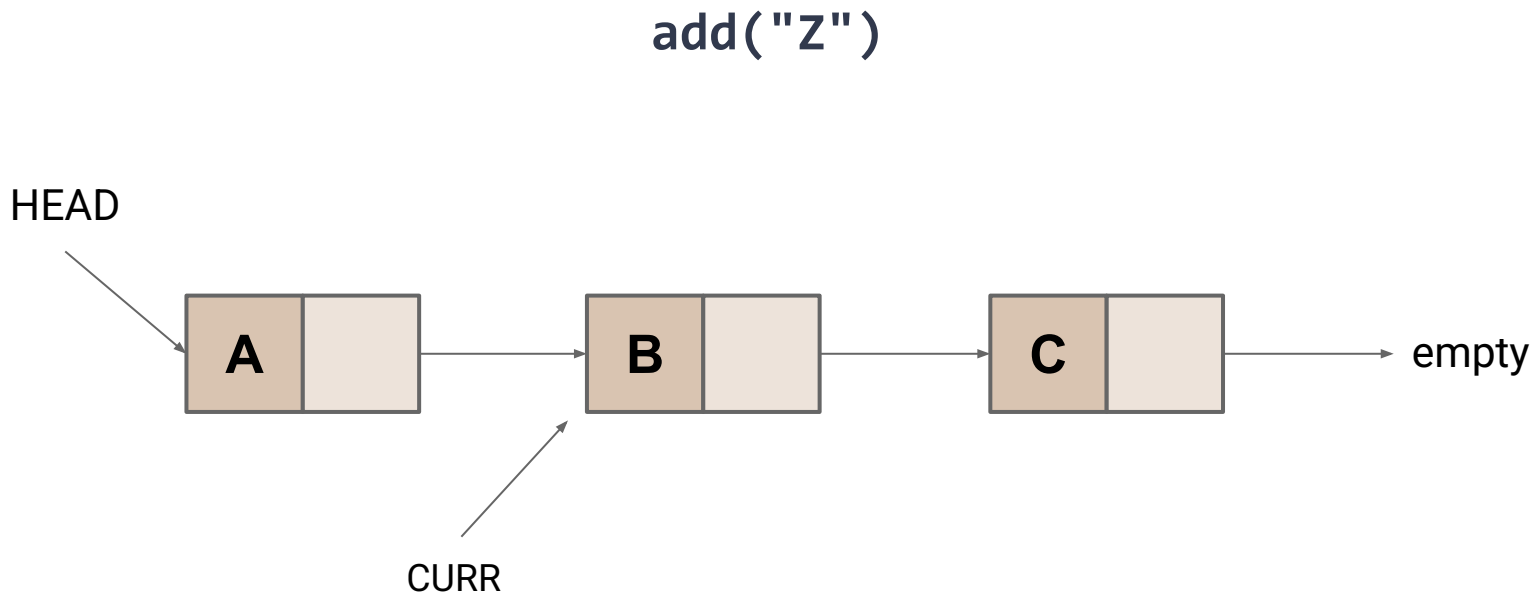


Linked Lists - add(e)

add("Z")

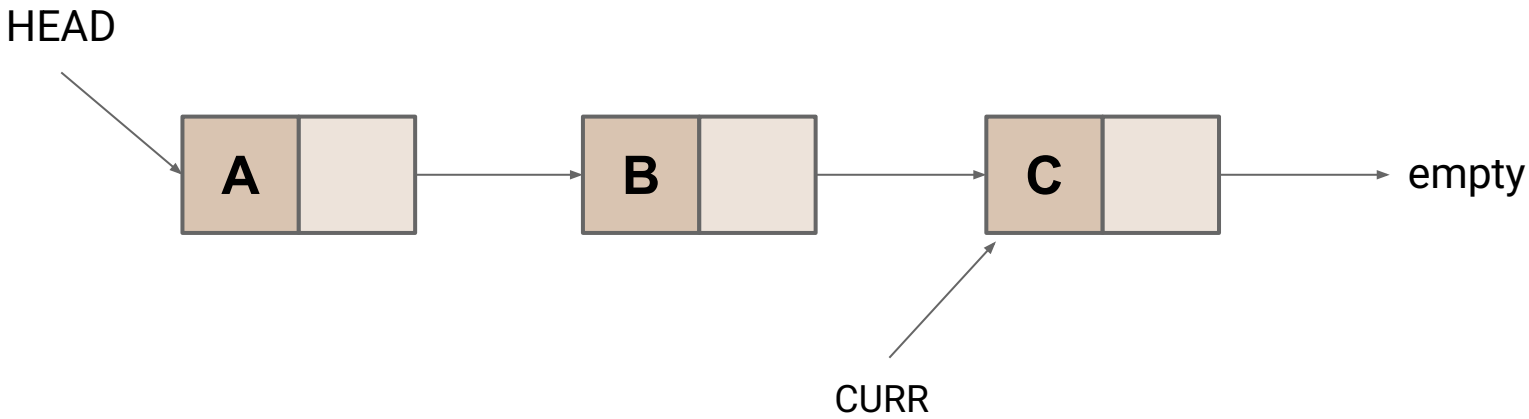


Linked Lists - add(e)

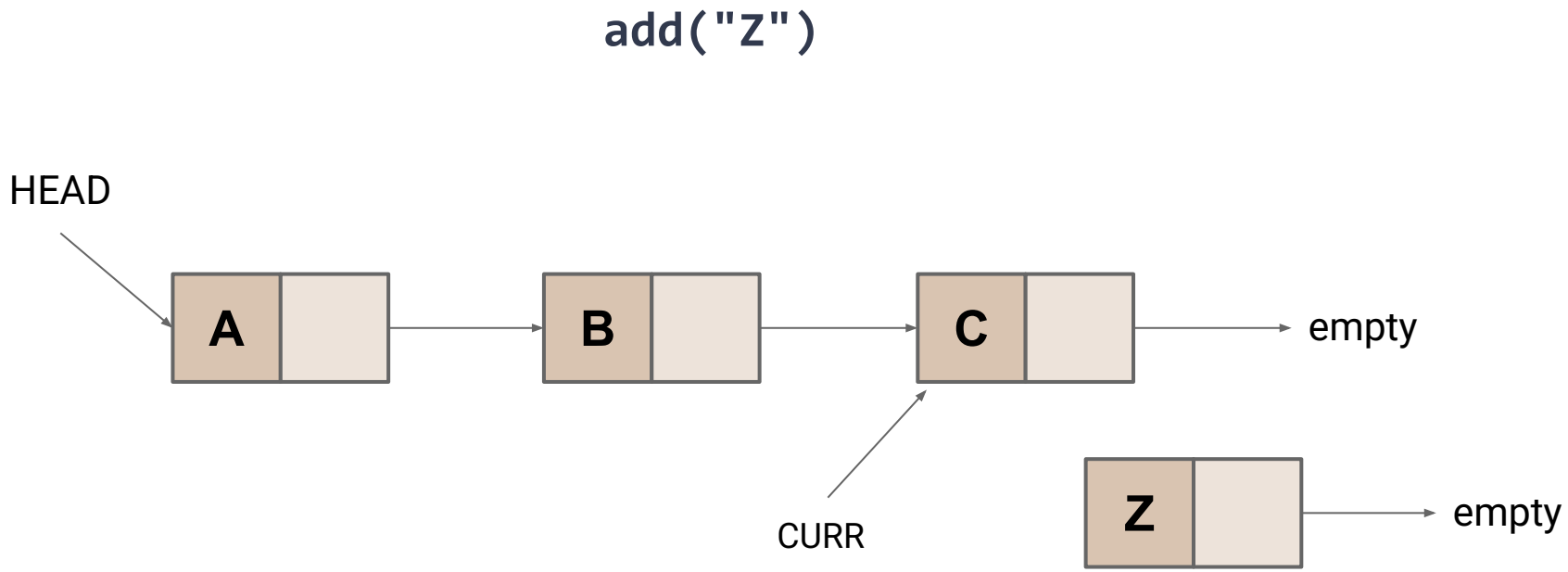


Linked Lists - add(e)

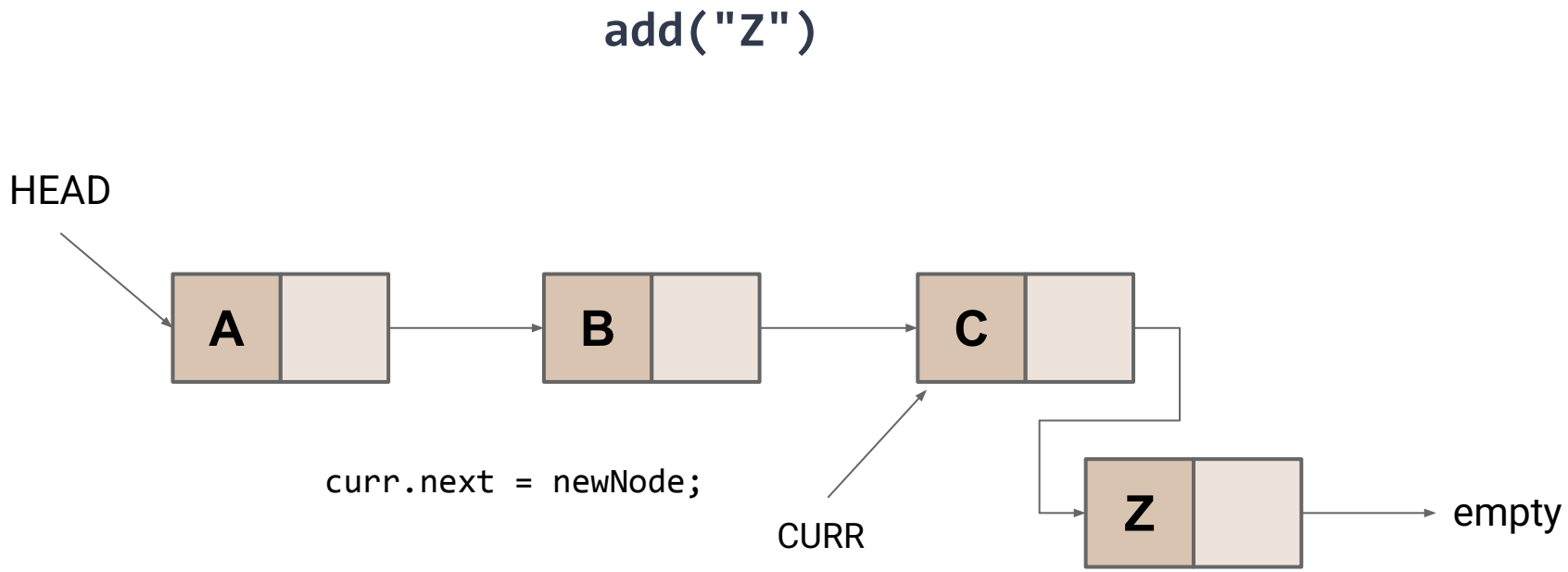
add("Z")



Linked Lists - add(e)



Linked Lists - add(e)

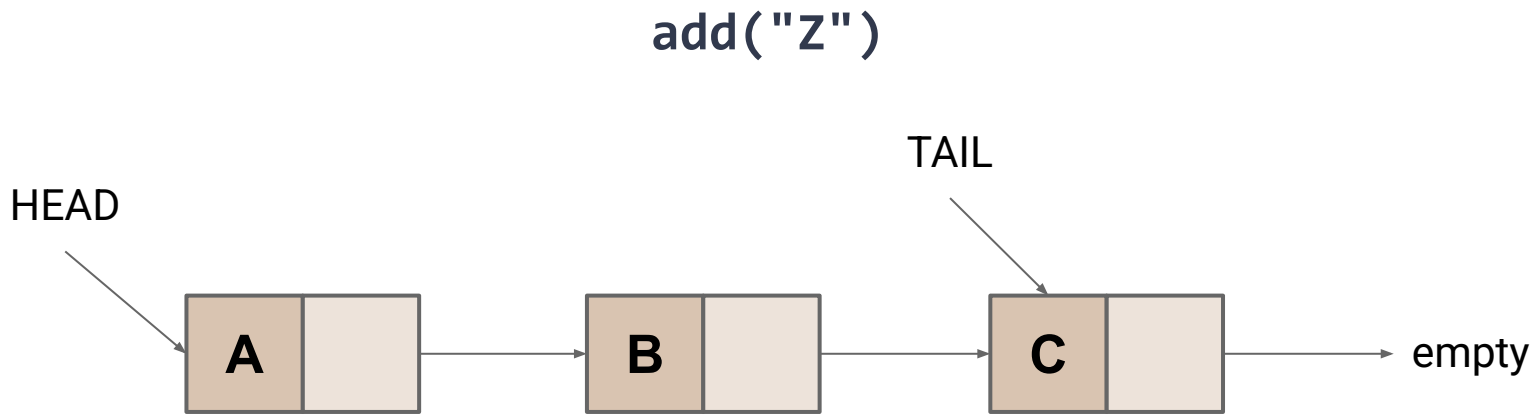


Linked Lists - `add(e)`

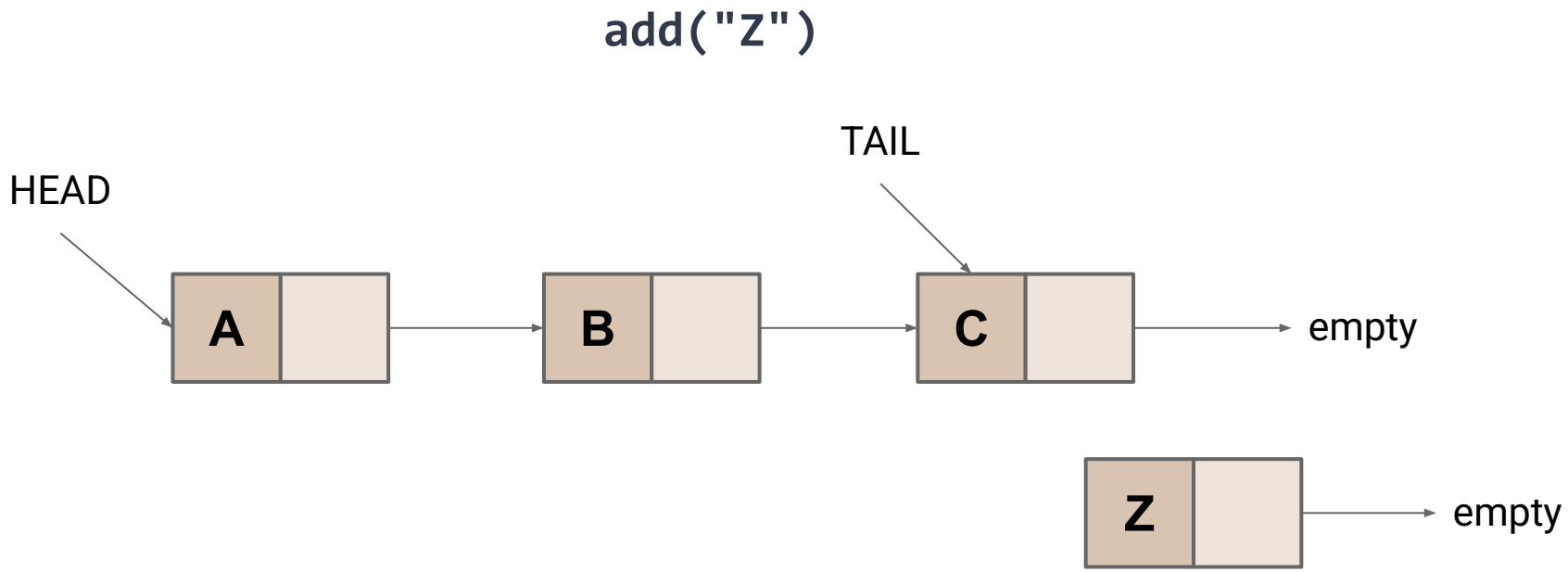
1. Find last node: $O(n)$
2. Allocate a new node and assign its value: $O(1)$
3. Update the last nodes **`next`** reference: $O(1)$

Total: $O(n)$

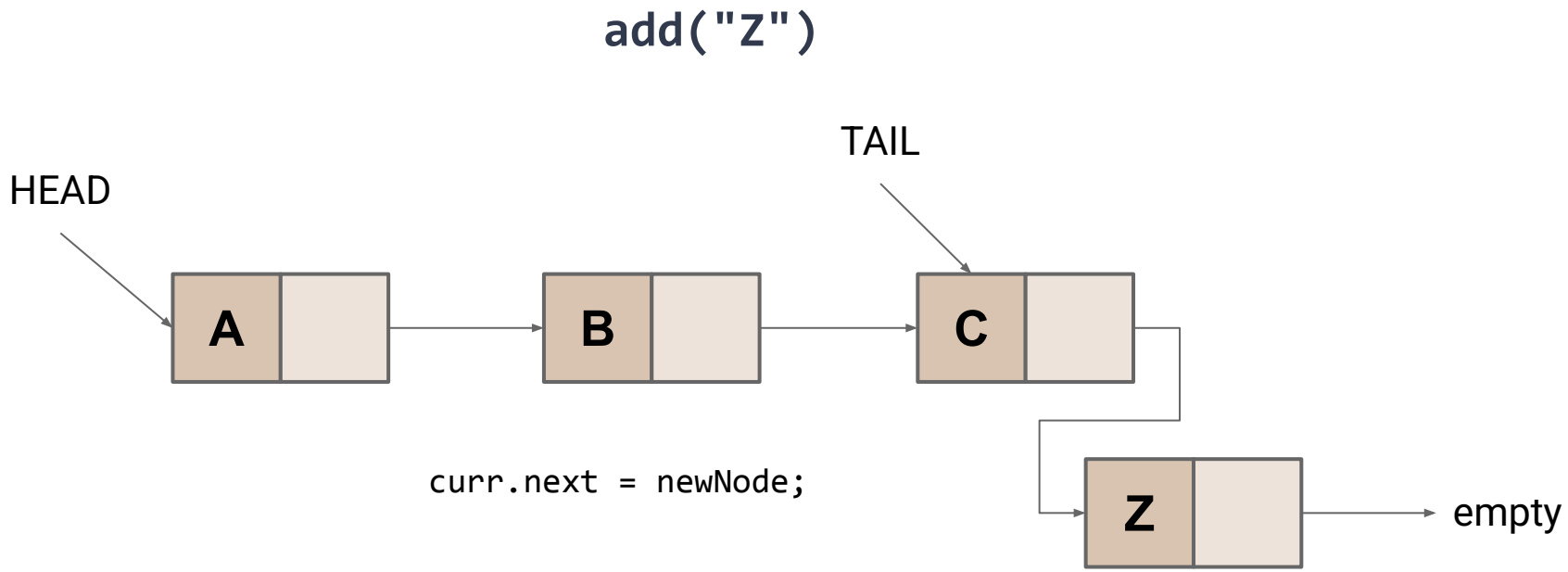
Linked Lists (w/ref to tail) - add(e)



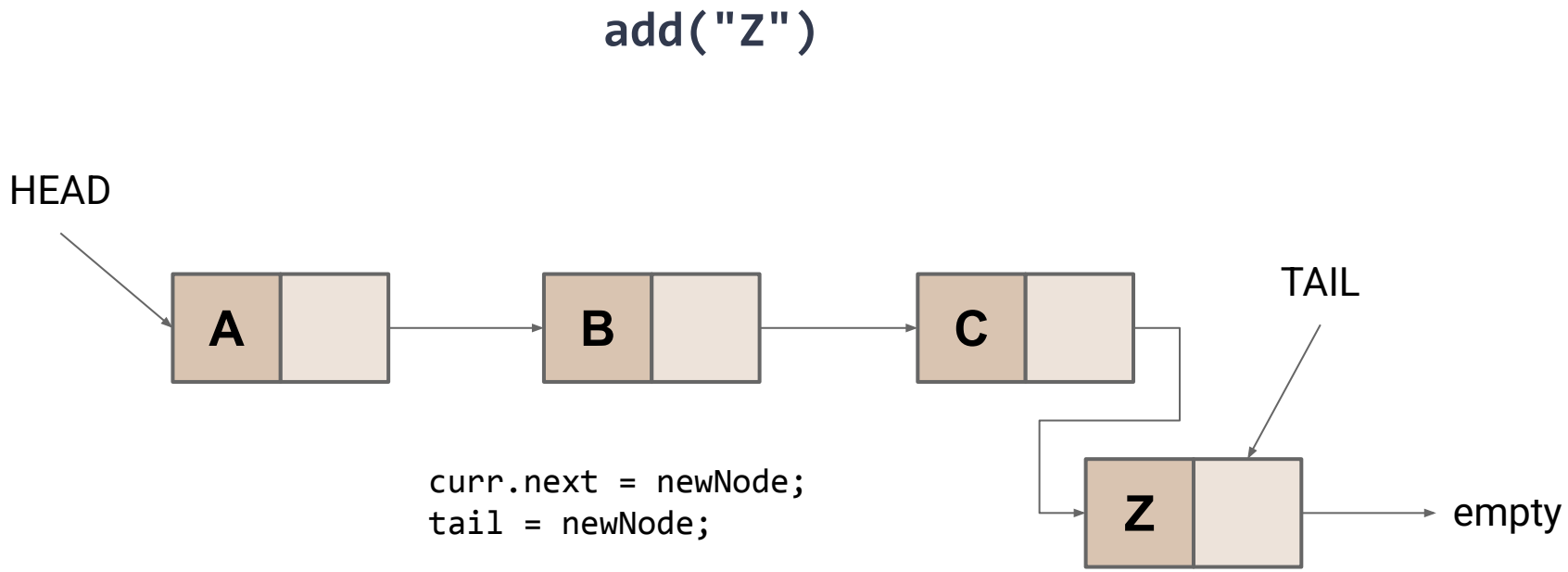
Linked Lists (w/ref to tail) - add(e)



Linked Lists (w/ref to tail) - add(e)



Linked Lists (w/ref to tail) - add(e)



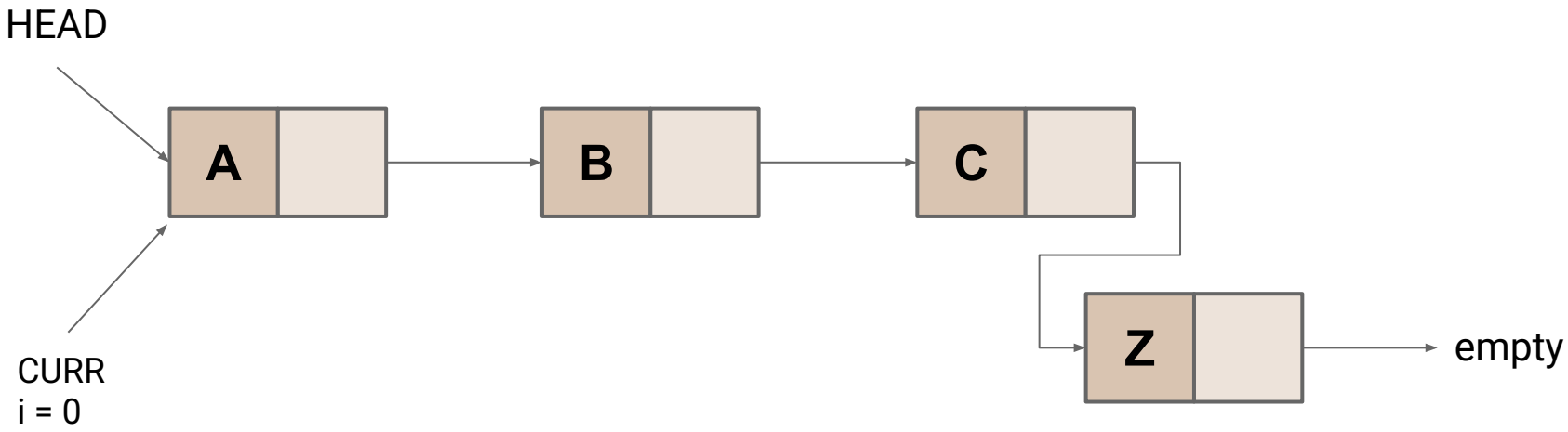
Linked Lists (w/ref to tail) - add(e)

1. Find last node: ***O*(1)**
2. Allocate a new node and assign its value: ***O*(1)**
3. Update the last nodes **`next`** reference: ***O*(1)**
4. Update the TAIL reference: ***O*(1)**

Total: *O*(1)

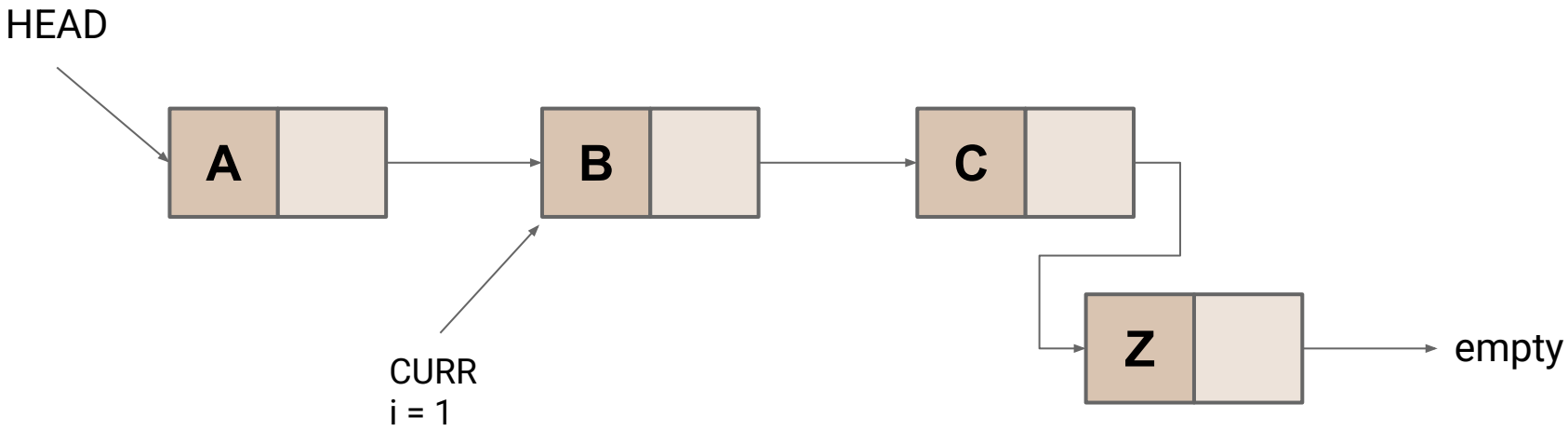
Linked Lists - `remove(idx)`

`remove(2)`

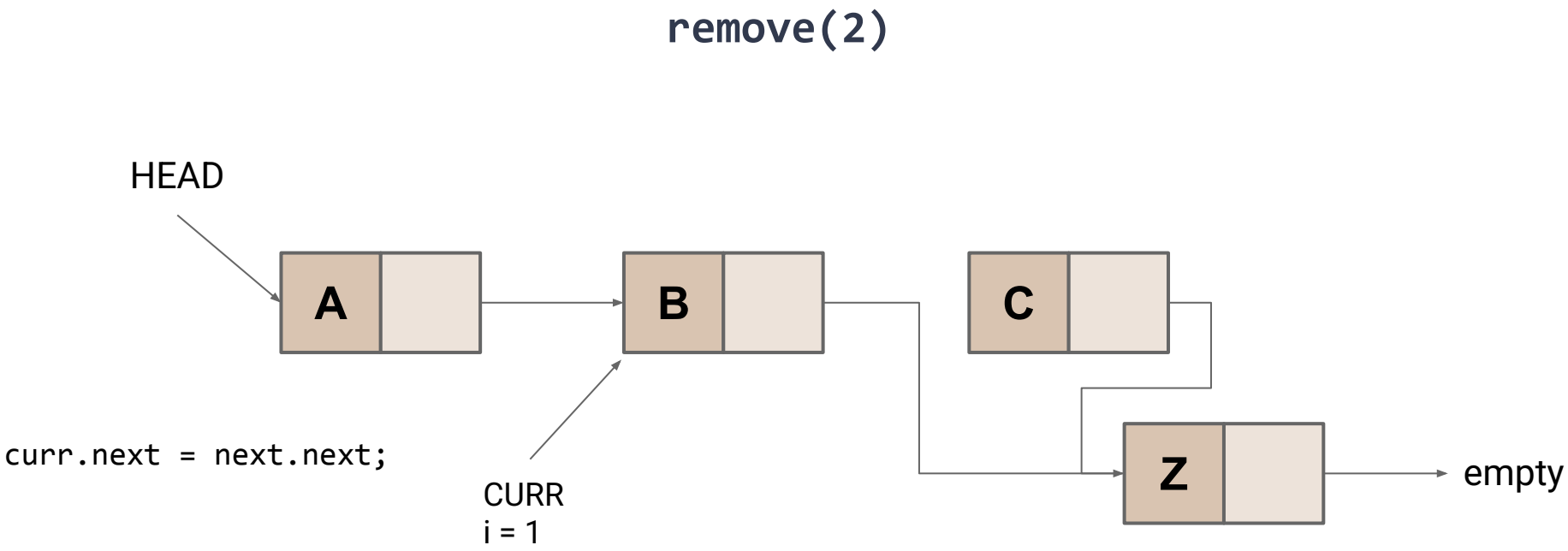


Linked Lists - `remove(idx)`

`remove(2)`



Linked Lists - `remove(idx)`



Linked Lists - `remove(idx)`

1. Find node before **`idx`**: ***O*(*n*)**
2. Update the node before **`idx`**'s **`next`** reference: ***O*(1)**
3. Allow the old node to be reclaimed: ***O*(1)**

Total: *O*(*n*)

Linked Lists

What is the expensive part of all of these operations?

Linked Lists

What is the expensive part of all of these operations?

Iterating to the correct index

Enumeration Example

```
1  int sumList(List<Integer> list){
2    int rslt = 0;
3    for(int i = 0; i < list.length; i++){
4      int temp = list.get(i);
5      rslt += temp;
6    }
7    return rslt;
8  }
```

What is the complexity of this code?

Enumeration Example

```
1  int sumList(List<Integer> list){
2    Θ(1)
3    for(int i = 0; i < list.length; i++){
4      int temp = list.get(i);
5      Θ(1)
6    }
7    Θ(1)
8  }
```

What is the complexity of this code?

Enumeration Example

```
1  int sumList(List<Integer> list){
2    Θ(1)
3    for(int i = 0; i < list.length; i++){
4      int temp = list.get(i);
5      Θ(1)
6    }
7    Θ(1)
8  }
```

What is the complexity of this code?

Enumeration Example

```
1  int sumList(List<Integer> list){
2    Θ(1)
3    for(int i = 0; i < list.length; i++){
4      Θ(i)
5      Θ(1)
6    }
7    Θ(1)
8  }
```

What is the complexity of this code?

Enumeration Example

```
1  int sumList(List<Integer> list){
2    Θ(1)
3    for(int i = 0; i < list.length; i++){
4      Θ(i)
5    }
6    Θ(1)
7  }
8
```

What is the complexity of this code?

Enumeration Example

```
1  int sumList(List<Integer> list){
2    Θ(1)
3    for(int i = 0; i < list.length; i++){
4      Θ(i)
5    }
6    Θ(1)
7  }
8
```

Note: This is **NOT** constant w.r.t. our loop variable!

What is the complexity of this code?

Enumeration Example

```
1  int sumList(List<Integer> list){
2    Θ(1)
3    for(int i = 0; i < list.length; i++){
4      Θ(i)
5    }
6    Θ(1)
7  }
8
```

What is the complexity

$$\sum_{i=0}^{n-1} \Theta(i)$$

Enumeration Example

```
1  int sumList(List<Integer> list){
2    Θ(1)
3    for(int i = 0; i < list.length; i++){
4      Θ(i)
5    }
6    Θ(1)
7  }
8
```

What is the complexity

$$\sum_{i=0}^{n-1} \Theta(i) = \Theta\left(\sum_{i=0}^{n-1} i\right)$$

Enumeration Example

```
1  int sumList(List<Integer> list){
2    Θ(1)
3    for(int i = 0; i < list.length; i++){
4      Θ(i)
5    }
6    Θ(1)
7  }
8
```

What is the complexity

$$\sum_{i=0}^{n-1} \Theta(i) = \Theta\left(\sum_{i=0}^{n-1} i\right) = \frac{(n-1)(n)}{2}$$

Enumeration Example

```
1  int sumList(List<Integer> list){
2    Θ(1)
3    for(int i = 0; i < list.length; i++){
4      Θ(i)
5    }
6    Θ(1)
7  }
8
```

What is the complexity

$$\sum_{i=0}^{n-1} \Theta(i) = \Theta\left(\sum_{i=0}^{n-1} i\right) = \frac{(n-1)(n)}{2} = \Theta(n^2)$$

Enumeration Example

```
1  int sumList(List<Integer> list){
2    Θ(1)
3    Θ(n^2)
4    Θ(1)
5  }
```

What is the complexity of this code? $\Theta(n^2)$

Enumeration Example

```
1  int sumList(List<Integer> list){
2    Θ(1)
3    Θ(n^2)
4    Θ(1)
5  }
```

What is the complexity of this code? $\Theta(n^2)$

Why is it so expensive?

Enumeration Example

```
1  int sumList(List<Integer> list){
2    Θ(1)
3    Θ(n^2)
4    Θ(1)
5  }
```

What is the complexity of this code? $\Theta(n^2)$

Why is it so expensive? **We start from index 0 every time!**

Enumeration Example #2

```
1  int sumLinkedList(LinkedList<Integer> list){
2    int rslt = 0;
3    Optional<LinkedListNode> n = list.headNode;
4    while (n.isPresent()){
5      int temp = n.get().value;
6      rslt += temp;
7      n = n.get().next;
8    }
9    return rslt;
10 }
```

Enumeration Example #2

```
1  int sumLinkedList(LinkedList<Integer> list){
2    int rslt = 0;
3    Optional<LinkedListNode> n = list.headNode;
4    while (n.isPresent()){
5      int temp = n.get().value;
6      rslt += temp;
7      n = n.get().next;
8    }
9    return rslt;
10 }
```

Now this is all constant time

Enumeration Example #2

```
1  int sumLinkedList(LinkedList<Integer> list){
2    Θ(1)
3    while (n.isPresent()){
4      Θ(1)
5    }
6    Θ(1)
7  }
```

Enumeration Example #2

```
1  int sumLinkedList(LinkedList<Integer> list){
2    Θ(1)
3    Θ(n)
4    Θ(1)
5  }
```

Total complexity: $\Theta(n)$

Enumeration

Problem:

- This code is specialized for `LinkedList`
- It will not work for other types of `List` (ie `ArrayList`)

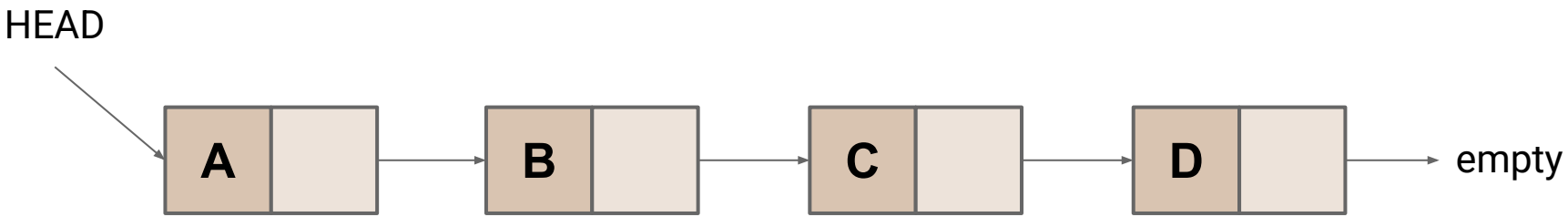
How can we get code that is both fast and general?

- Must be able to represent a reference to the `idx`'th element of a `List`

ListIterator

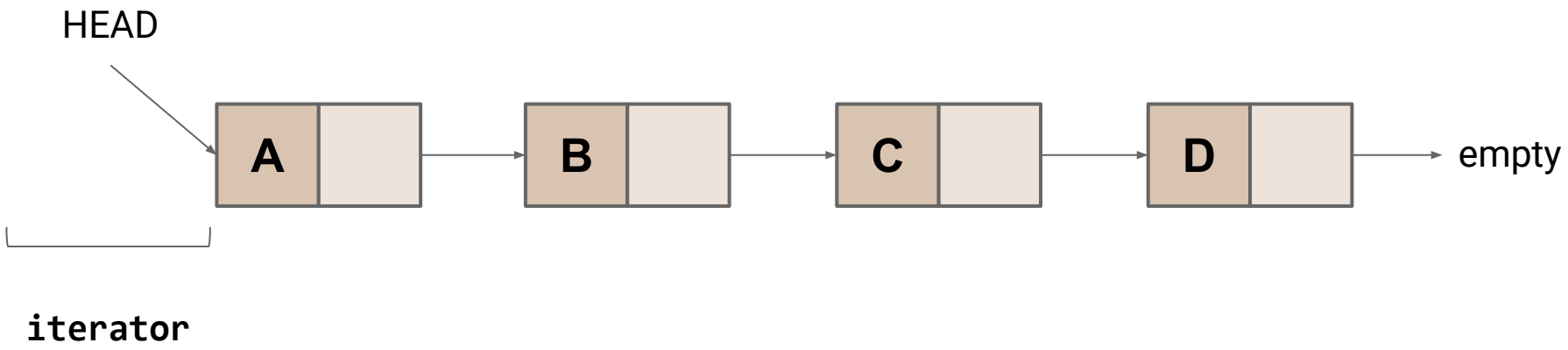
```
1 public interface ListIterator<E> {
2   public boolean hasNext();
3   public E next();
4   public void add(E value);
5   public void set(E value);
6   public void remove();
7 }
```

Linked List Iterator



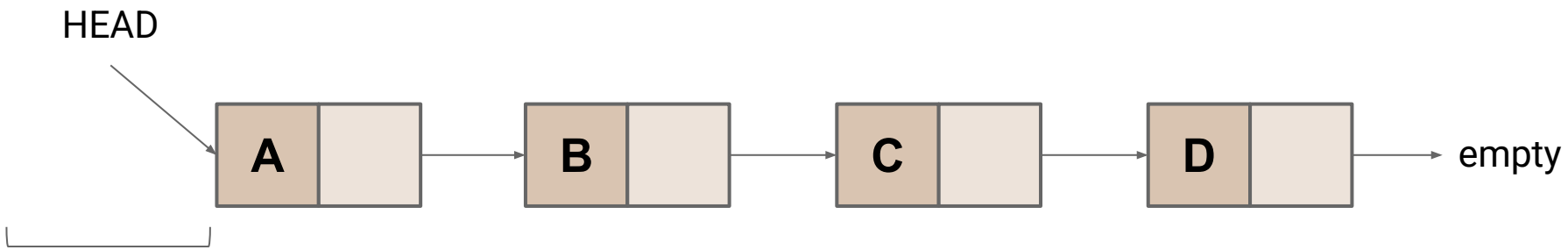
Linked List Iterator

```
iterator = list.iterator()
```



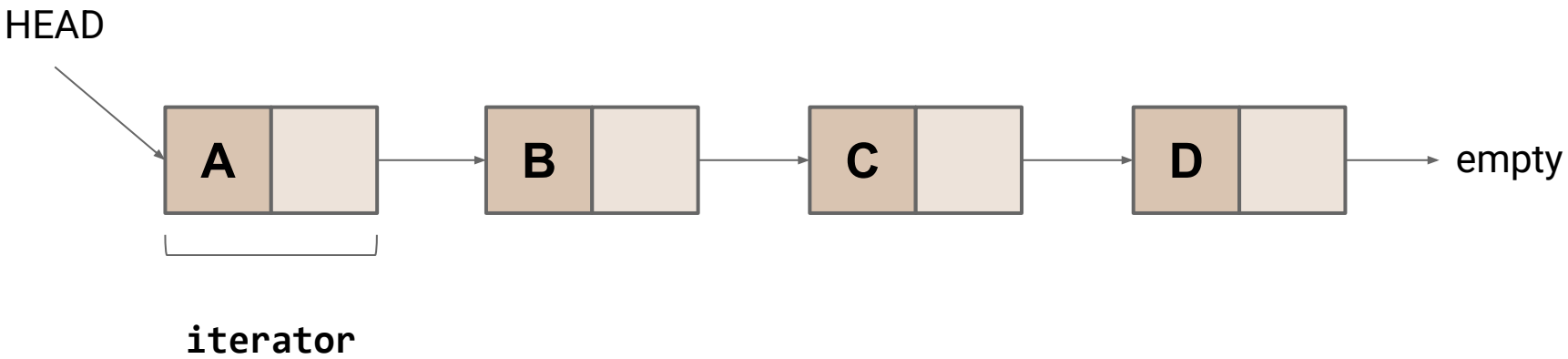
Linked List Iterator

`iterator.hasNext() → true`



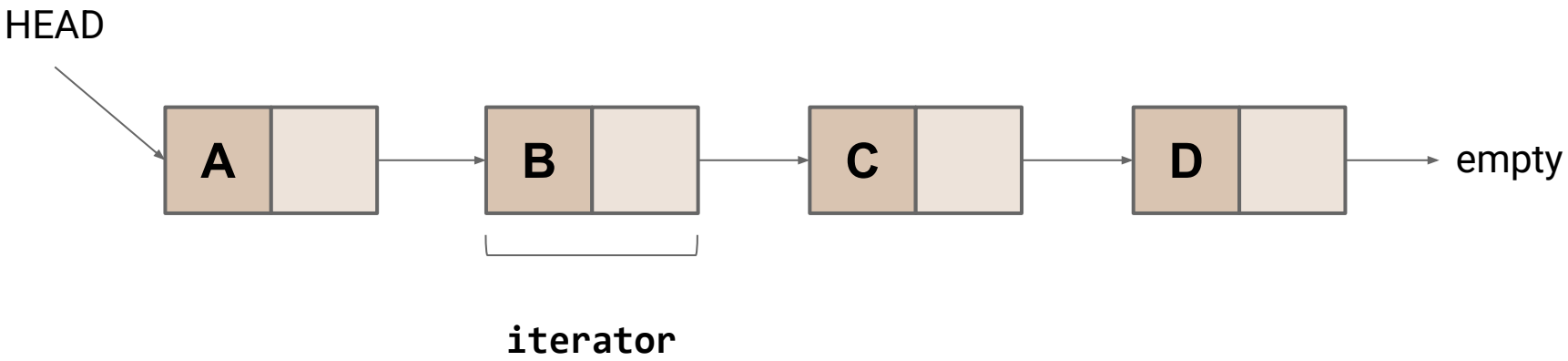
Linked List Iterator

`iterator.next() → A`



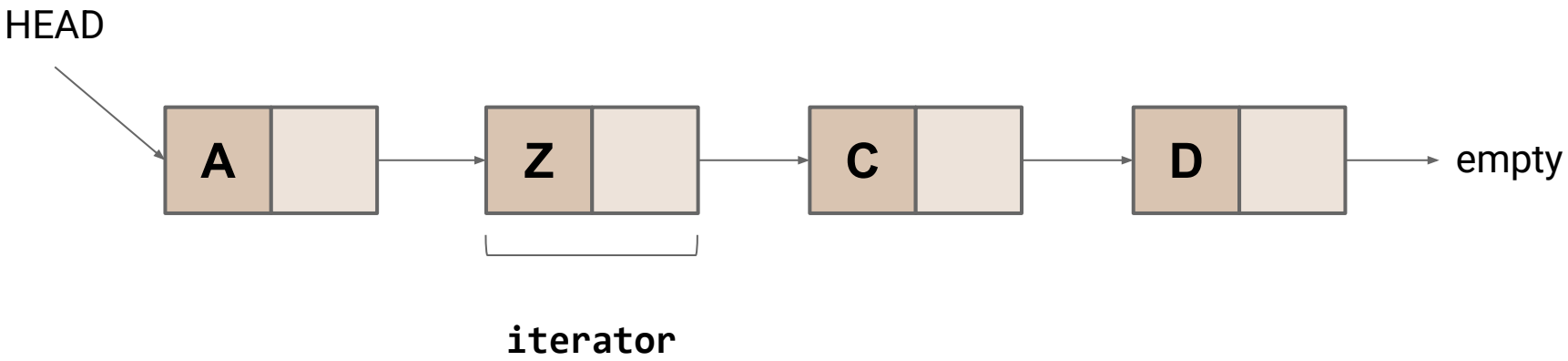
Linked List Iterator

```
iterator.next() → B
```



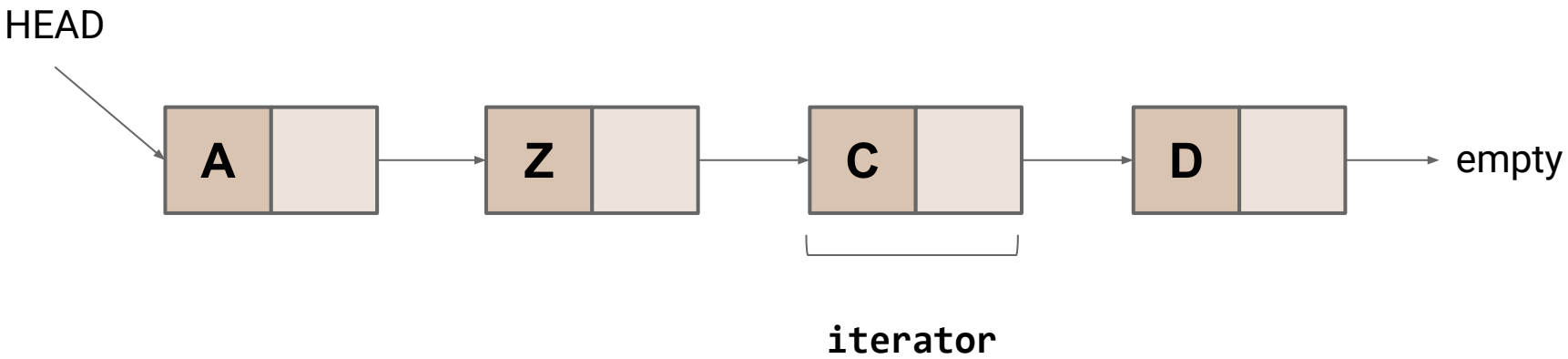
Linked List Iterator

```
iterator.set(Z)
```



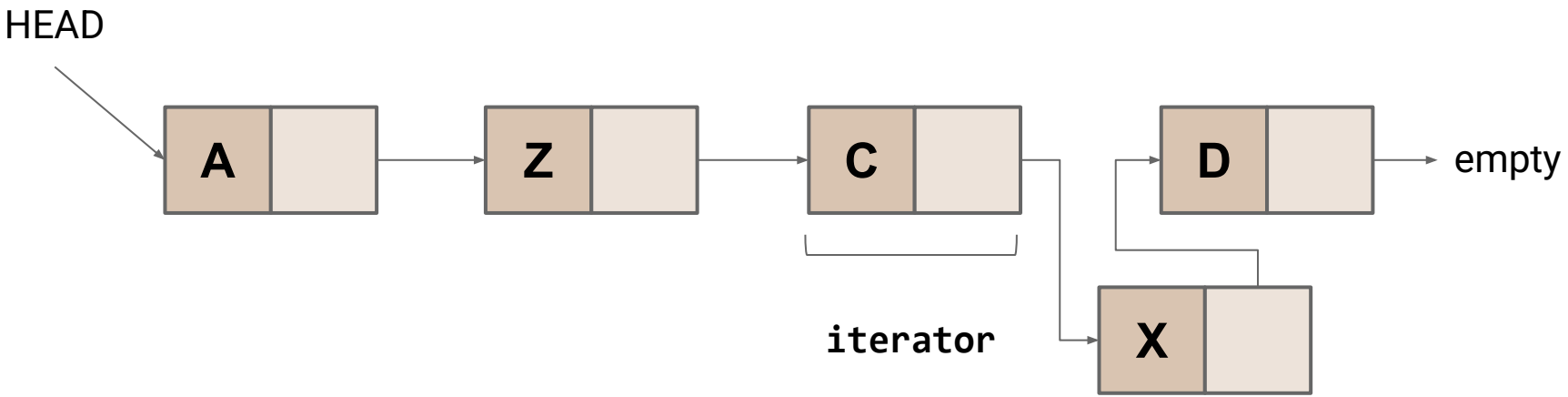
Linked List Iterator

`iterator.next() → C`



Linked List Iterator

`iterator.add(X)`



Linked List Iterator

```
1 public class LinkedListIterator<E>
2   extends ListIterator<E> {
3   LinkedList<E> list;
4   Optional<LinkedListNode<E>> before = Optional.empty();
5   Optional<LinkedListNode<E>> after = Optional.of(list.head);
6   /* ... */
7 }
```

Linked List Iterator

`boolean hasNext()`

If **`after.isPresent()`**, return true — $O(1)$

`T next()`

If **`after.isPresent()`**, advance **`before`** and **`after`**, and return the value of **`before`** — $O(1)$

`void set(T value)`

Update **`before.value`** — $O(1)$

Linked List Iterator

`void add(T value)`

Create a new node, update it's **`next`** and **`before.next`** $O(1)$

`void remove(T value)`

Set **`before.next`** to **`after.next`** and update **`after`** $O(1)$

Enumeration using ListIterator

```
1 public void int sumUpList(List<Integer> list) {
2   int total = 0;
3   ListIterator<Integer> iterator = list.iterator();
4   while(iterator.hasNext()) {
5     int value = iterator.next();
6     total += value;
7   }
8   return total;
9 }
```

Enumeration using ListIterator

```
1  public void int sumUpList(List<Integer> list) {
2    int total = 0;
3    ListIterator<Integer> iterator = list.iterator();
4    while(iterator.hasNext()) {
5      int value = iterator.next();
6      total += value;
7    }
8    return total;
9  }
```

Generalized to work with any kind of list!

Enumeration using ListIterator

```
1  public void int sumUpList(List<Integer> list) {
2    int total = 0;
3    ListIterator<Integer> iterator = list.iterator();
4    while(iterator.hasNext()) {
5      int value = iterator.next();
6      total += value;
7    }
8    return total;
9  }
```

Loop body only contains $\Theta(1)$ operations

Enumeration using ListIterator

```
1  public void int sumUpList(List<Integer> list) {
2    int total = 0;
3    ListIterator<Integer> iterator = list.iterator();
4    while(iterator.hasNext()) {
5      int value = iterator.next();
6      total += value;
7    }
8    return total;
9  }
```

Total Complexity: $\Theta(n)$

ArrayLists

Question: How can we implement **`add(e)`** on an **`ArrayList`**?

ArrayLists

Question: How can we implement **`add(e)`** on an **`ArrayList`**?

Problem: Arrays have a fixed size!

ArrayLists - Attempt #1

1. Allocate a new array of size ***n* + 1** — ***O*(1)**
2. Copy all ***n*** elements to the new array — ***O*(*n*)**
3. Insert the new item at position ***n*** — ***O*(1)**

Total: *O*(*n*)

ArrayLists - Attempt #1

1. Allocate a new array of size ***n* + 1** — ***O*(1)**
2. Copy all ***n*** elements to the new array — ***O*(*n*)**
3. Insert the new item at position ***n*** — ***O*(1)**

Total: *O*(*n*)

Can we do better? next class…