CSE 250
Data Structures

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

Lec 09: List ADT and Linked Lists

Warm Up Question

If $f(n) \in O(n^3)$, then:

    (a) $f(n)$ must be in $\Theta(n^2)$

    (b) $f(n)$ could be in $\Theta(n^2)$

    (c) $f(n)$ can't be in $\Theta(n^2)$

If $f(n) \in \Theta(n^2)$, then:

    (a) $f(n)$ must be in $O(n^3)$

    (b) $f(n)$ could be in $O(n^3)$

    (c) $f(n)$ can't be in $O(n^3)$

Warm Up Question

If $f(n) \in O(n^3)$, then:

    (a) $f(n)$ must be in $\Theta(n^2)$

    (b) $f(n)$ could be in $\Theta(n^2)$

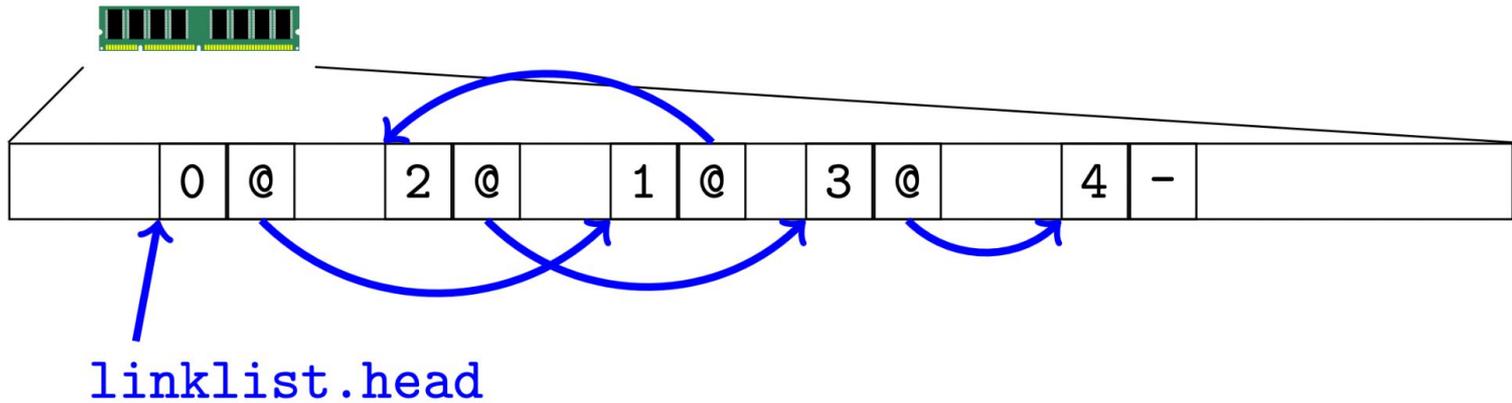   (c) $f(n)$ can't be in $\Theta(n^2)$

If $f(n) \in \Theta(n^2)$, then:
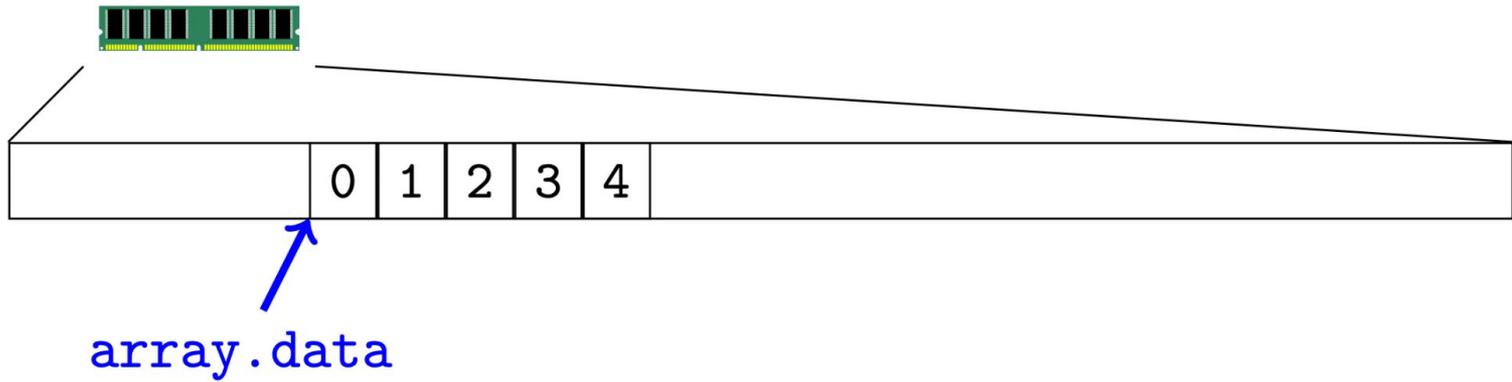
    (a) $f(n)$ must be in $O(n^3)$

    (b) $f(n)$ could be in $O(n^3)$

    (c) $f(n)$ can't be in $O(n^3)$

Announcements

- PA1 Implementation due Sunday, 2/15 @ 11:59PM
    - Continue with the same repo you've been using
- WA2 will be released after the PA1 deadline, due 2/22 @ 11:59PM

The Sequence ADT

```java
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  }
```

array.data

linklist.head

Arrays and Linked Lists in Memory

Sequence Runtimes

	Array	**Linked List (by index)**	**Linked List (by reference)**
`get(...)`	$\Theta(1)$	$\Theta(\text{idx})$ or $\boldsymbol{O}(n)$	$\Theta(1)$
`set(...)`	$\Theta(1)$	$\Theta(\text{idx})$ or $\boldsymbol{O}(n)$	$\Theta(1)$
`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

```java
public interface List<E>
    extends Sequence<E> { // Everything a sequence has, and...
  /** Extend the sequence with a new element at the end */
  public void add(E value);

  /** Extend the sequence by inserting a new element */
  public void add(int idx, E value);

  /** Remove the element at a given index */
  public void remove(int idx);
}
```

Lists in Other Languages

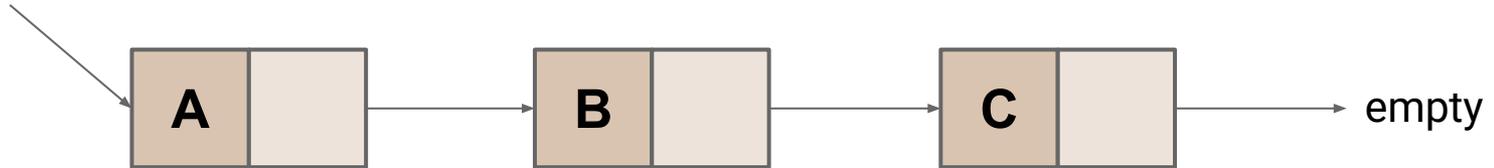**Java, Python:** `List`, `list`

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

Scala: `Buffer`

Go: `Slice`

Linked Lists - add(idx, e)

add(2,"D")

HEAD

```
A | → B | → C | → empty
```

Linked Lists - add(idx, e)
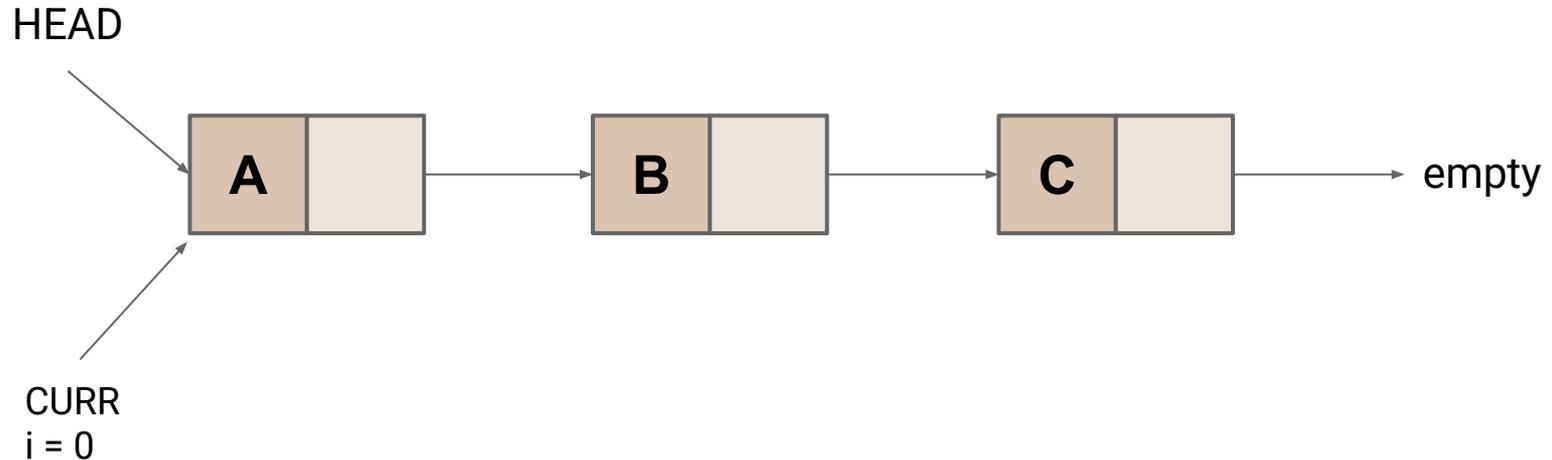
add(2,"D")

Linked Lists - add(`idx, e`)
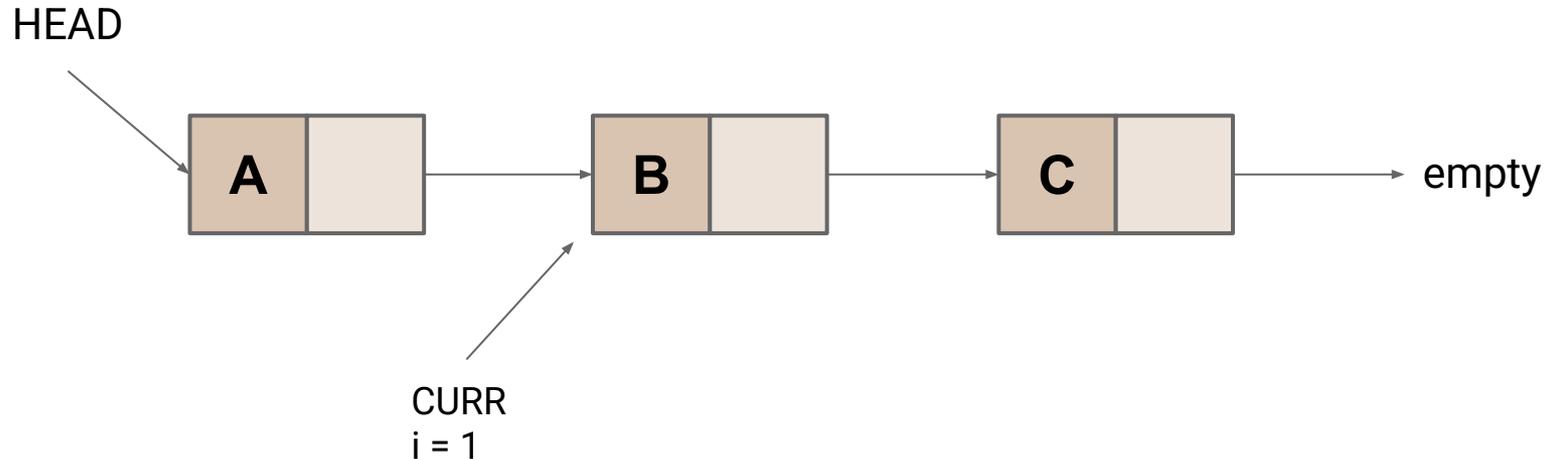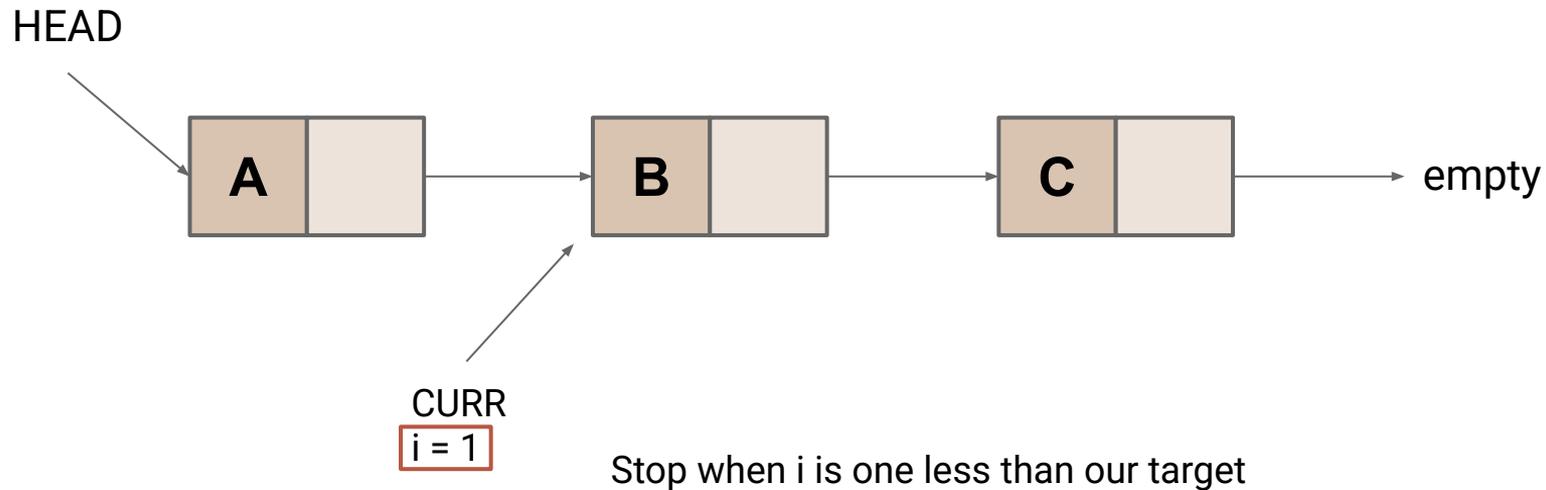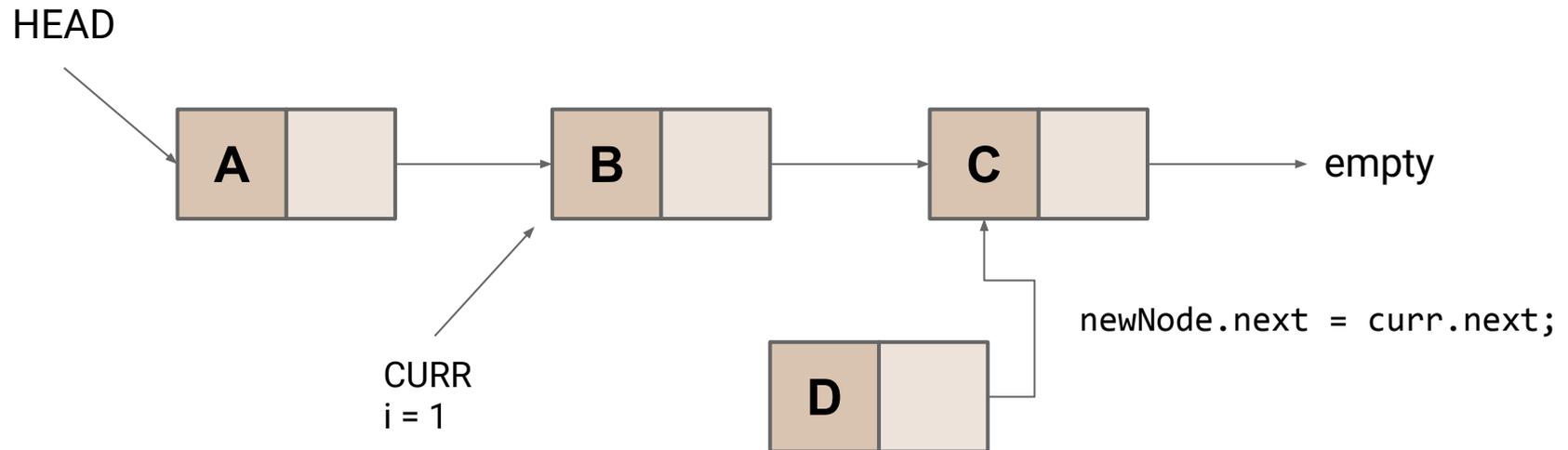
add(2,"D")
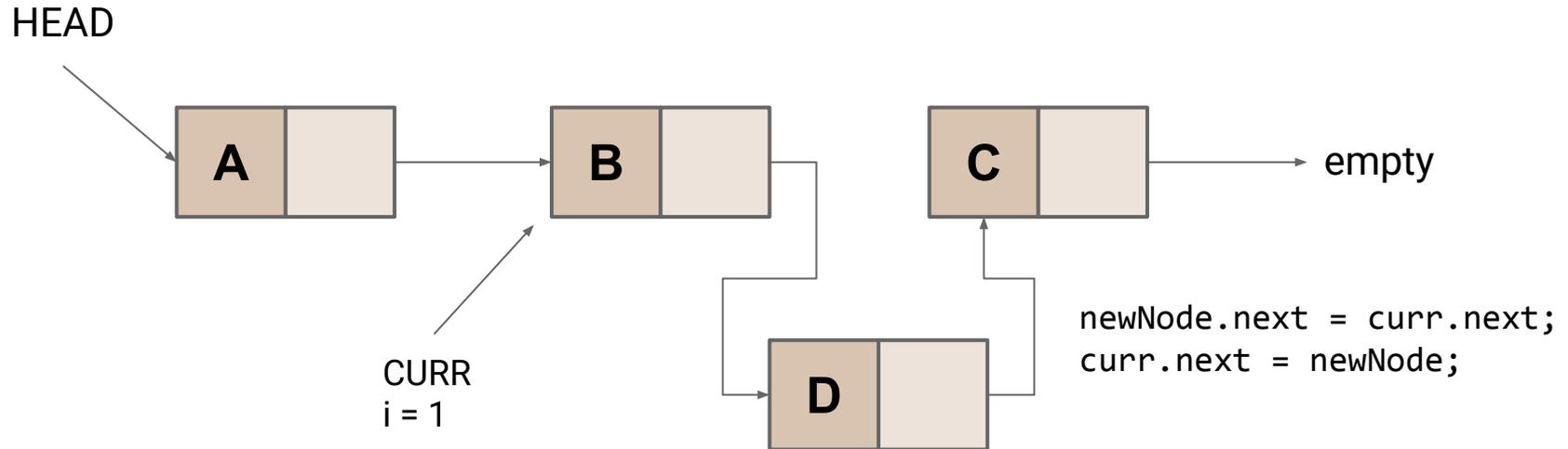
Linked Lists - add(idx, e)

add(2,"D")

Linked Lists - add(idx, e)

add(2,"D")

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(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: $O(n)$

What about Ω?

Linked Lists - `add(idx, e)`

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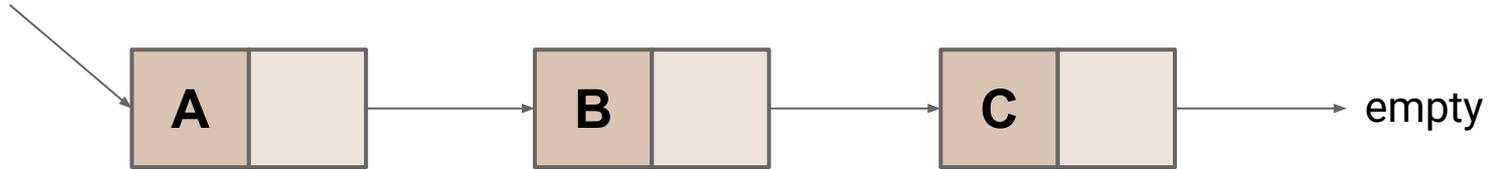4. Update the node before `idx`'s `next` reference: $O(1)$

Total: $O(n)$, $\Omega(1)$

What about Ω?

Linked Lists - add(e)
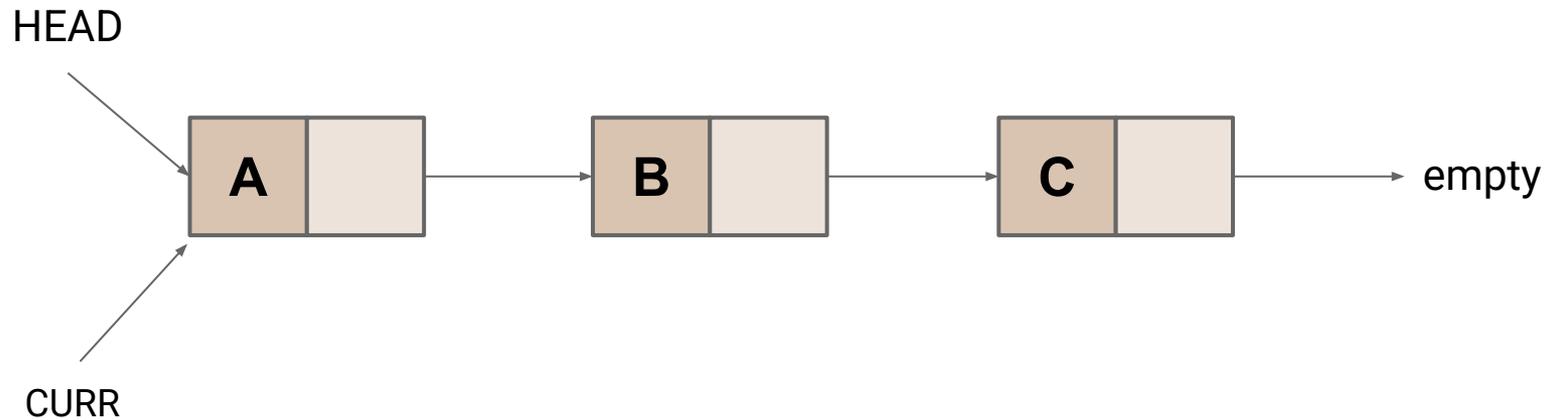
add("Z")
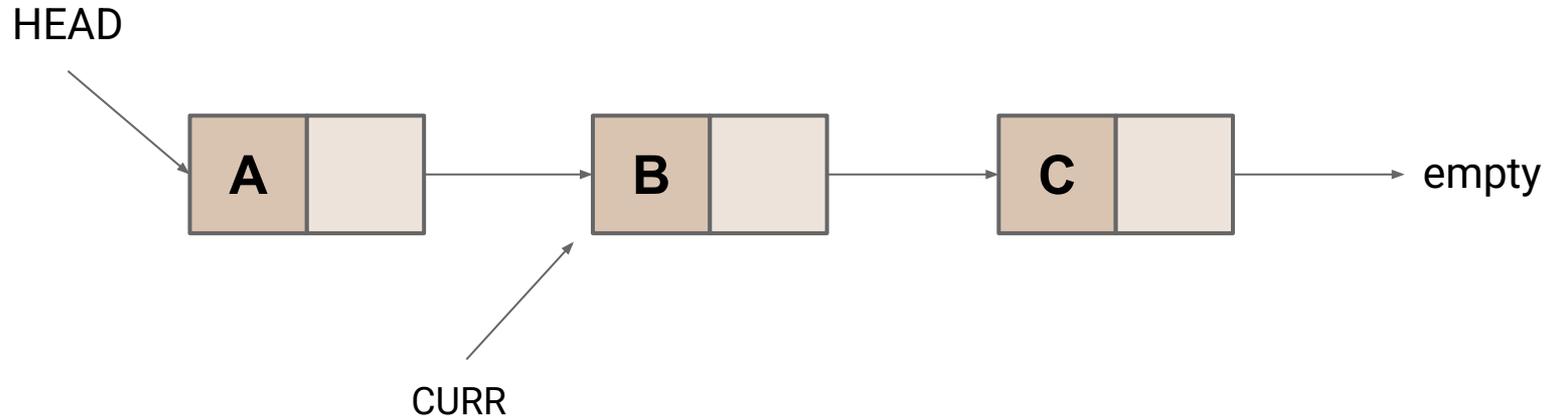
HEAD

A → B → C → empty

Linked Lists - add(e)

add("Z")

Linked Lists - add(e)

add("Z")

Linked Lists - add(e)

add("Z")

HEAD

A → B → C → empty

CURR

Linked Lists - add(e)

add("Z")

HEAD

A → B → C → empty

CURR

Z → empty

Linked Lists - add(e)

add("Z")

HEAD

A | **B** | **C**

curr.next = newNode;

CURR

Z → empty
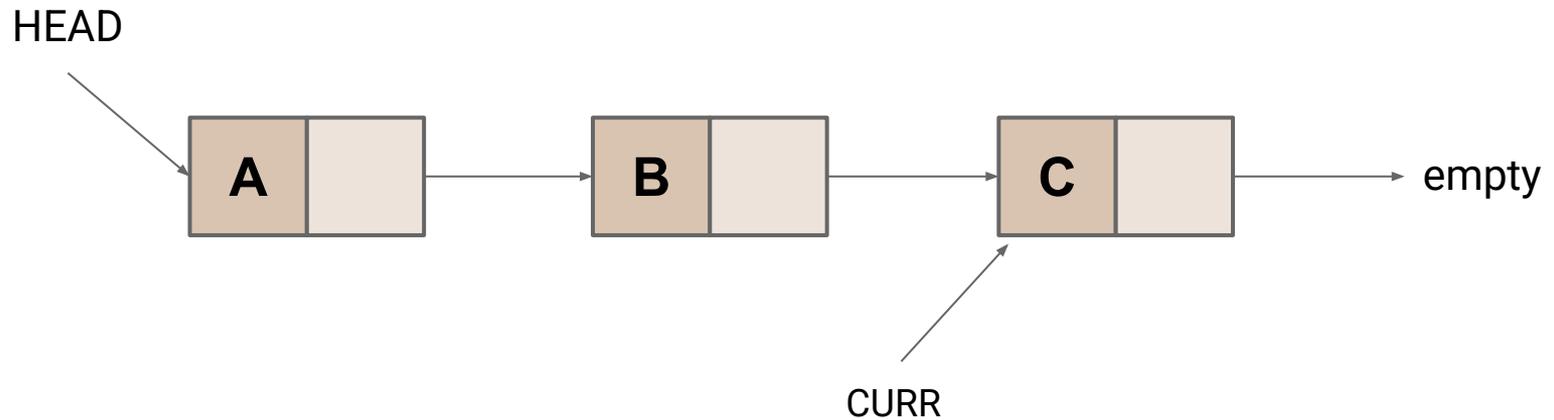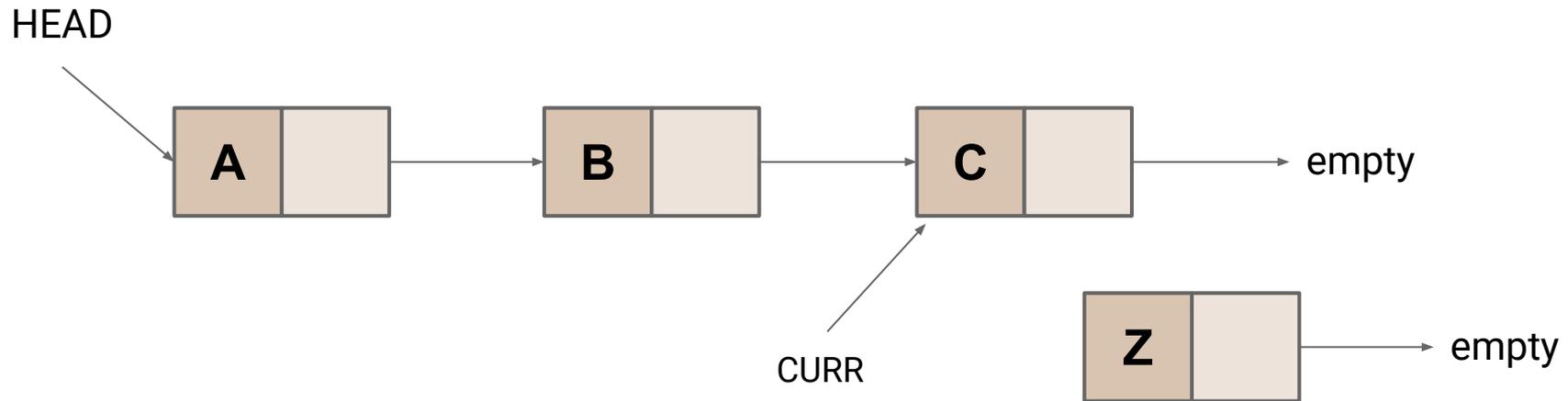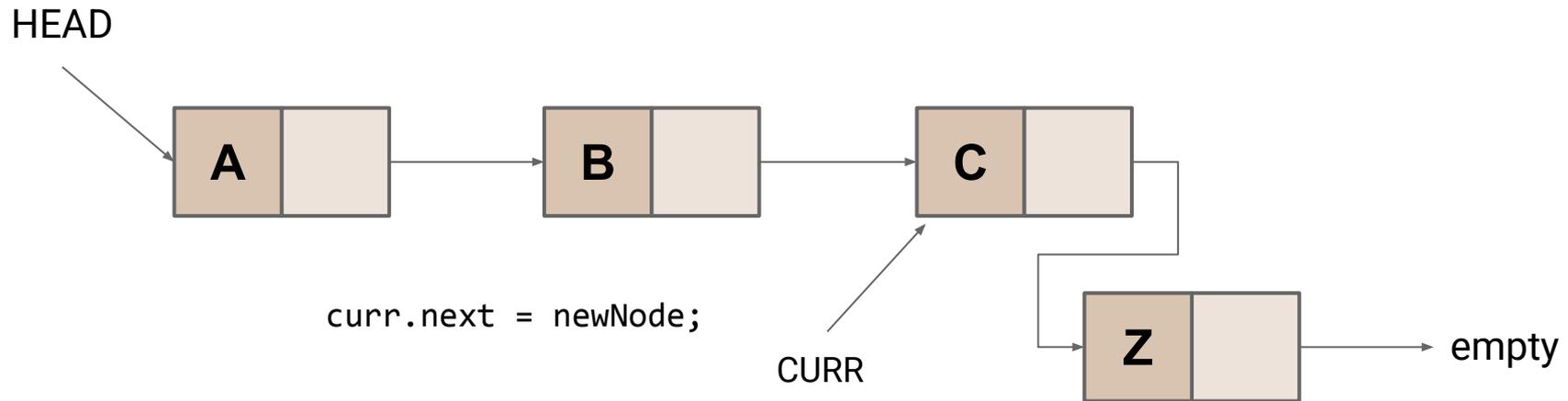
Linked Lists - add(e)

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 - 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: $O(n)$

What about Ω?

Linked Lists - add(e)

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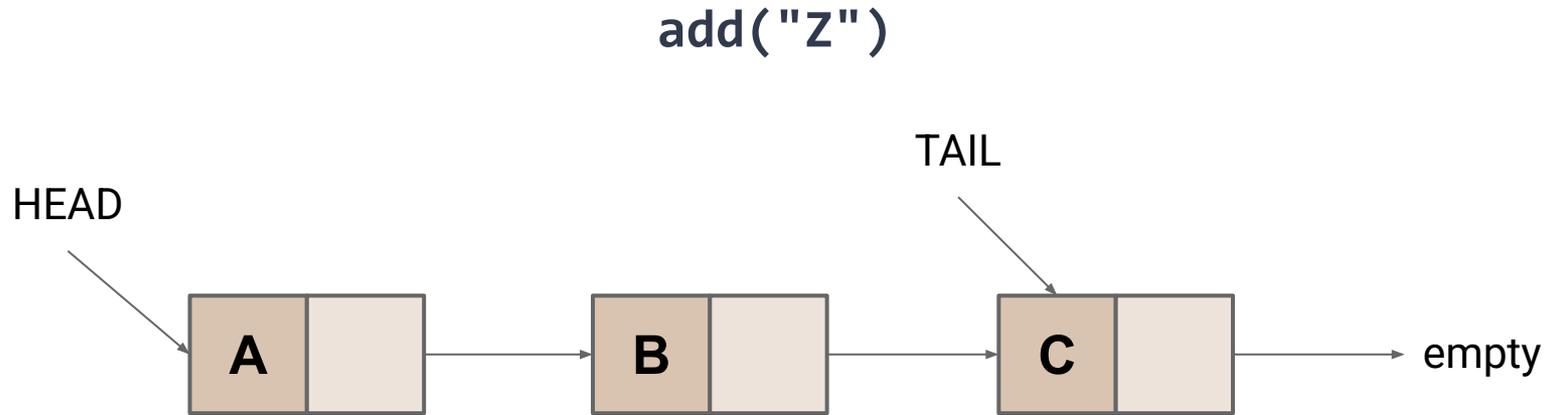
Total: $O(n)$, $\Omega(n)$

What about Ω?

Linked Lists - add(e)

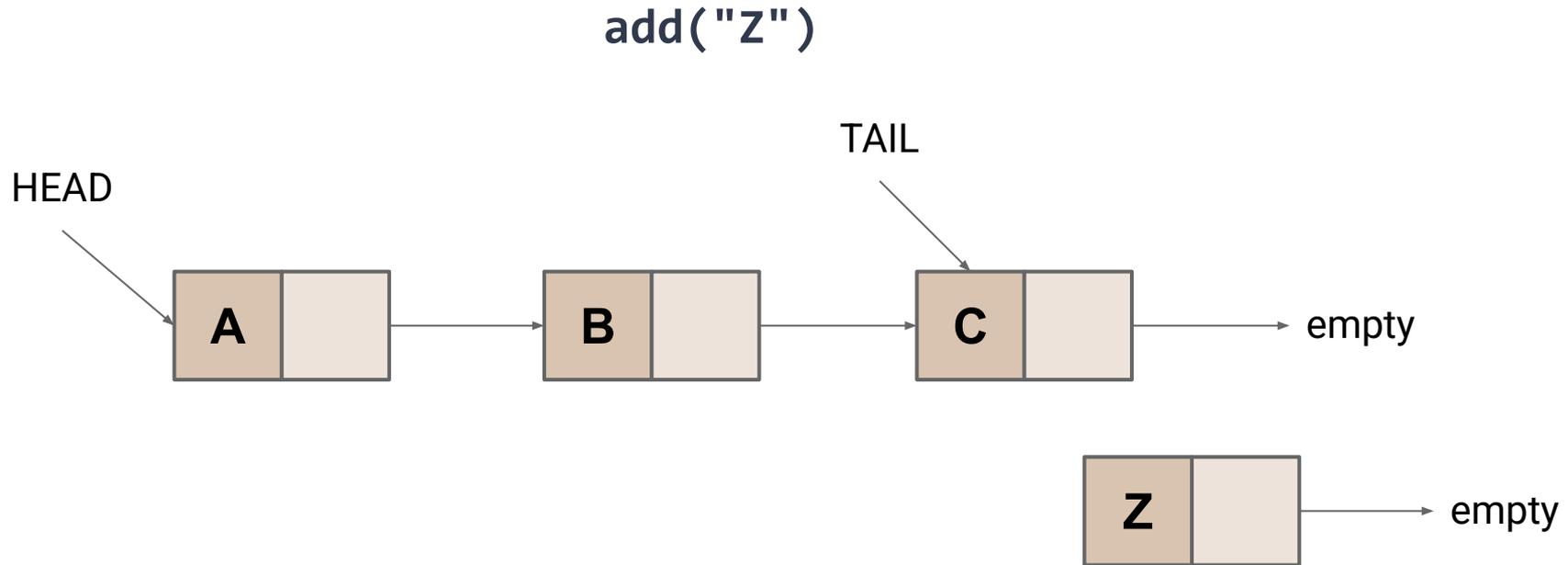1. Find last node: $\Theta(n)$
2. Allocate a new node and assign its value: $O(1)$
3. Update the last nodes **next** reference: $O(1)$

Total: $\Theta(n)$

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

add("Z")

TAIL

HEAD

| A | | B | | C | | empty |
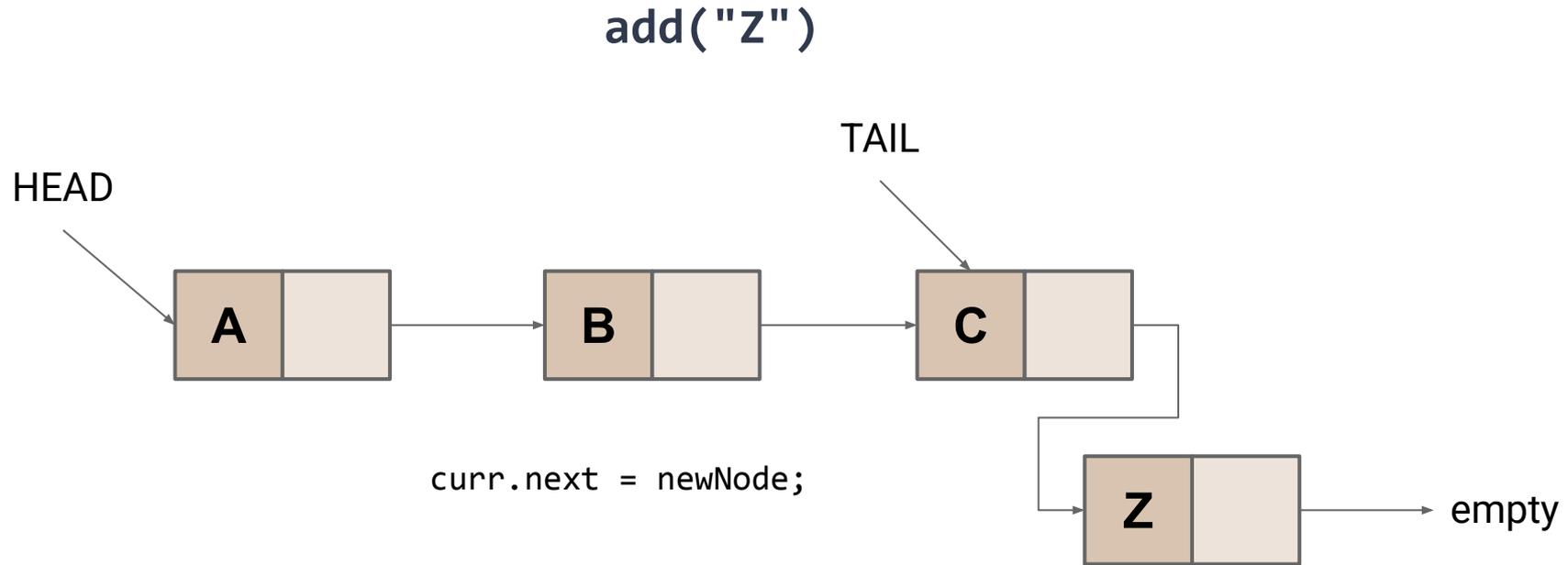
Linked Lists (w/ref to tail) - add(e)

add("Z")

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

add("Z")

TAIL

HEAD

A → B → C

curr.next = newNode;

Z → empty

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

add("Z")

HEAD

A	
B	
C	

TAIL

| Z | | → empty

```
curr.next = newNode;
tail = newNode;
```
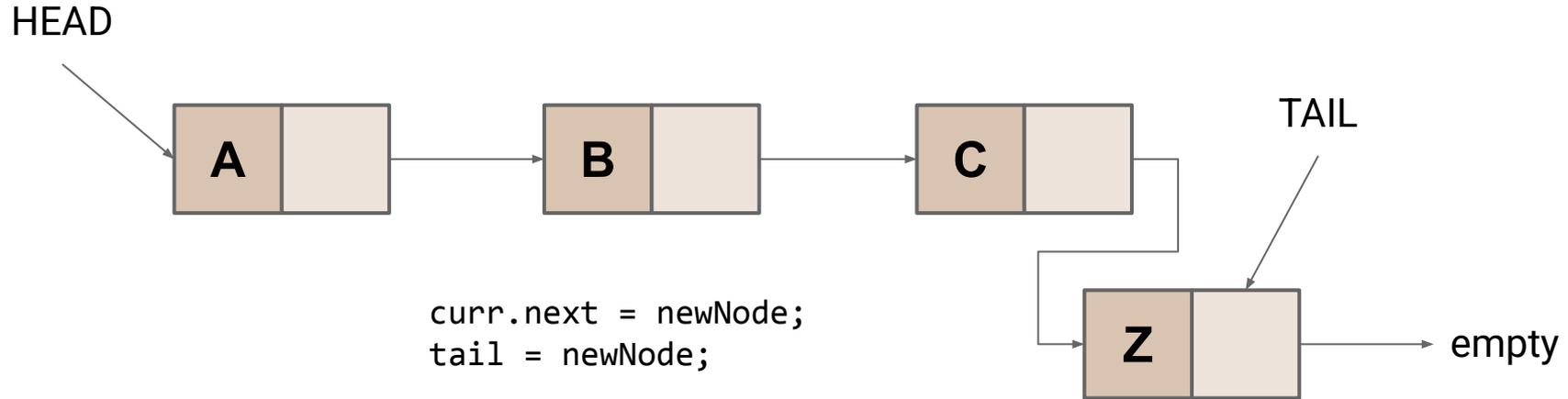
Linked Lists (w/ref to tail) - add(e)

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)`

HEAD

A → B → C

Z → empty

CURR
i = 0

Linked Lists - `remove(idx)`

`remove(2)`

HEAD

A

B

C

Z

empty

CURR
i = 1

Linked Lists - `remove(idx)`

`remove(2)`

HEAD

A → B   C

Z → empty

`curr.next = next.next;`

CURR
i = 1
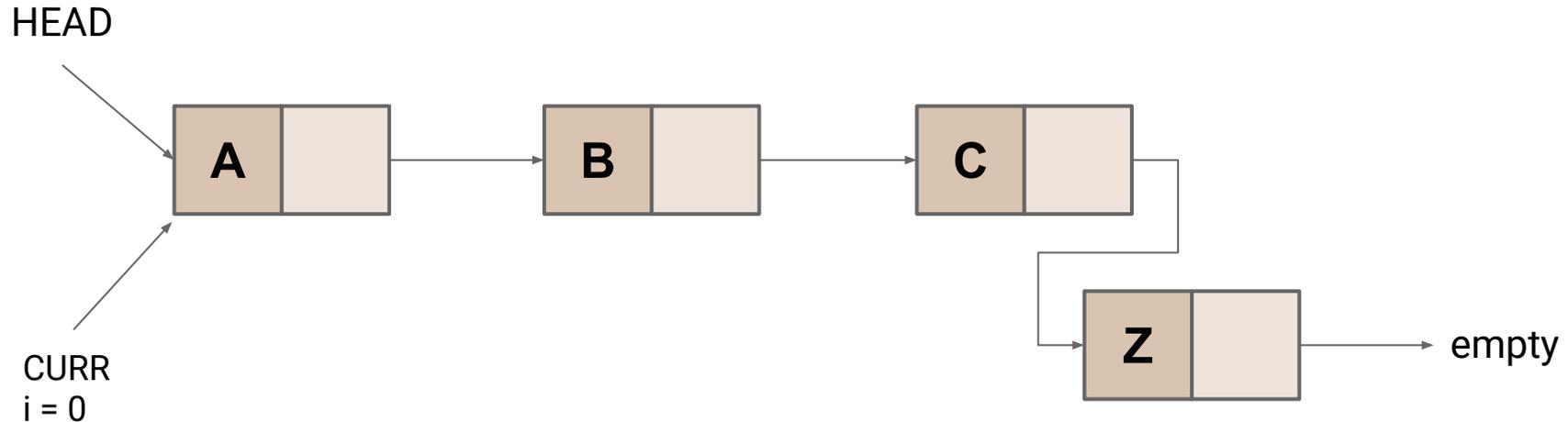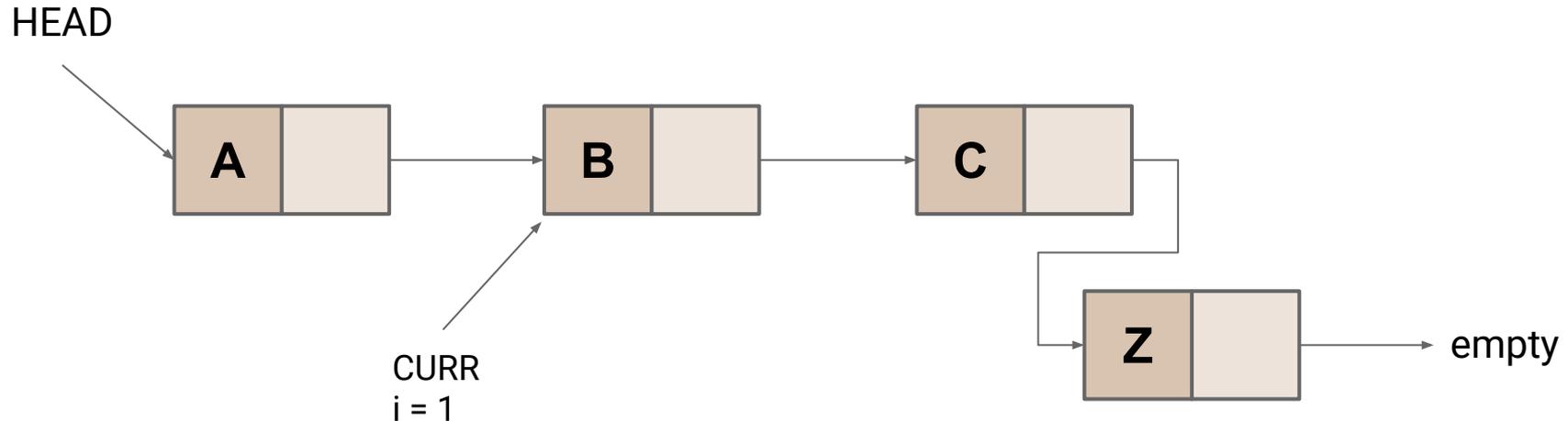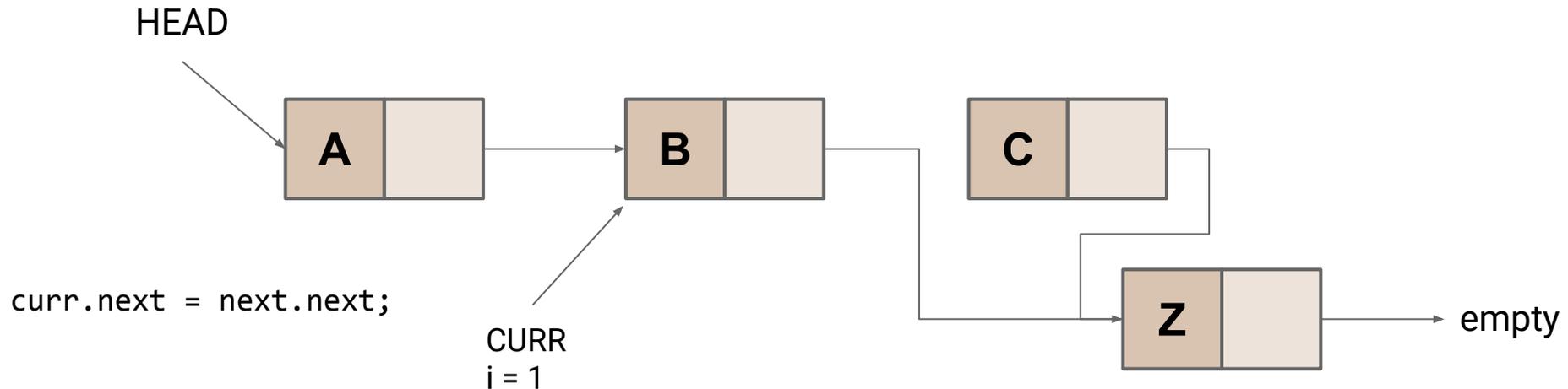
Linked Lists - `remove(idx)`

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

Total: *O*(*n*), Ω(1)

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    Θ(n²)
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(n²)
4      Θ(1)
5  }
```

What is the complexity of this code? **Θ(*n²*)**

Why is it so expensive?

Enumeration Example

```
1  int sumList(List<Integer> list){
2    Θ(1)
3    Θ(n²)
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;        Now this is all constant time
8    }
9    return rslt;
10 }
```

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: Θ(*n*)

Enumeration

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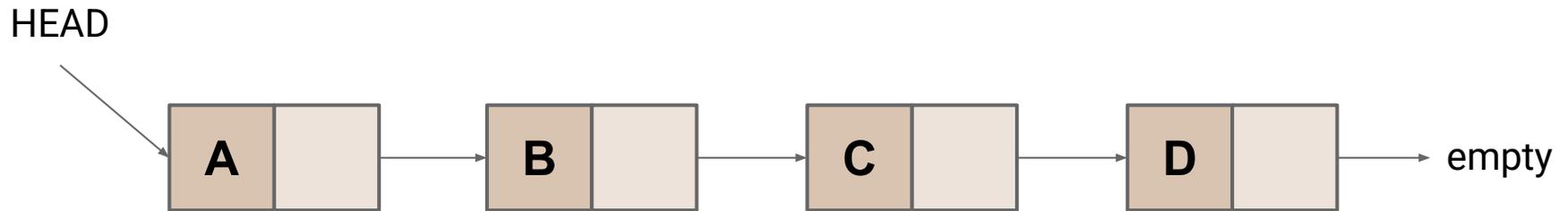- It will not work for other types of `List` (ie `ArrayList`)

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`
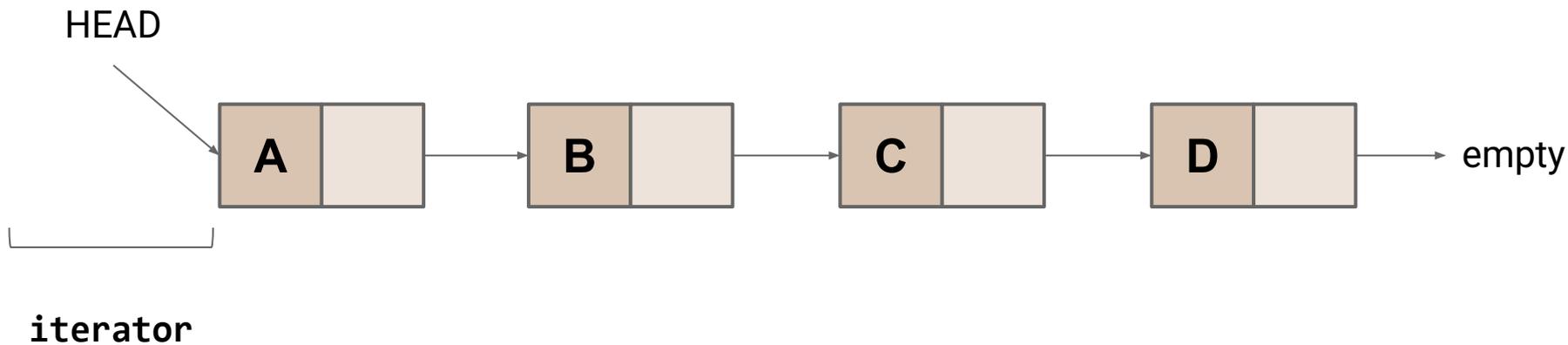
ListIterator

```java
public interface ListIterator<E> {
  public boolean hasNext();
  public E next();
  public void add(E value);
  public void set(E value);
  public void remove();
}
```

Linked List Iterator

HEAD

A → B → C → D → empty
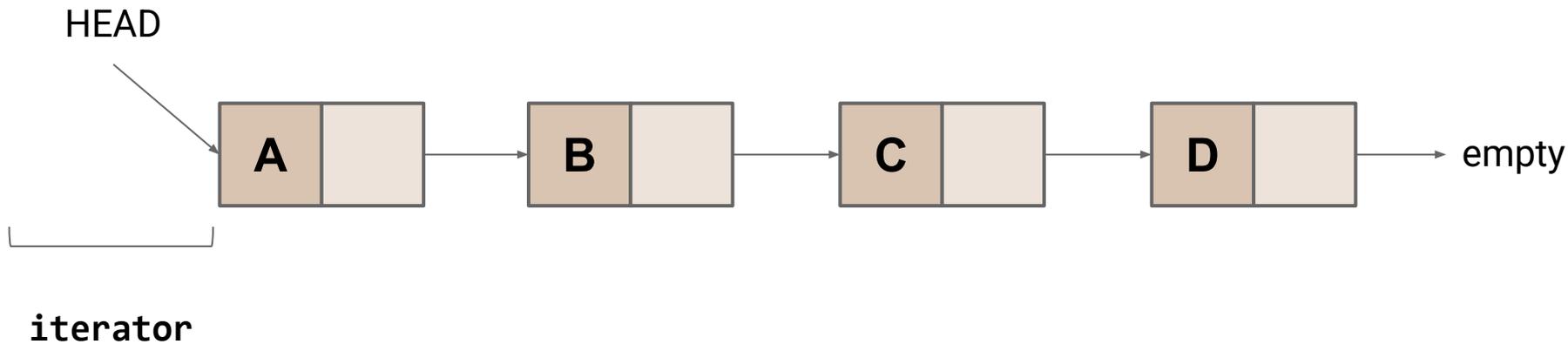
Linked List Iterator

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

HEAD

A → B → C → D → empty

iterator

Linked List Iterator

`iterator.hasNext() → true`

Linked List Iterator

`iterator.next()` → `A`

HEAD

A | B | C | D | empty
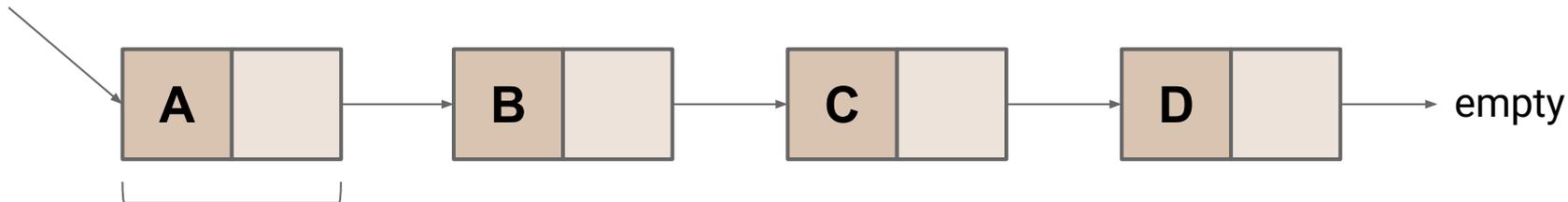
iterator

Linked List Iterator

$$iterator.next() \rightarrow B$$

HEAD

A → B → C → D → empty
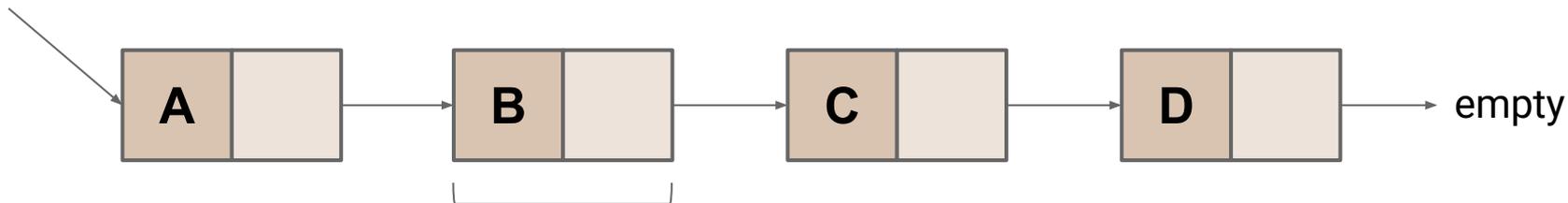
iterator

Linked List Iterator

`iterator.set(Z)`

HEAD

A → Z → C → D → empty
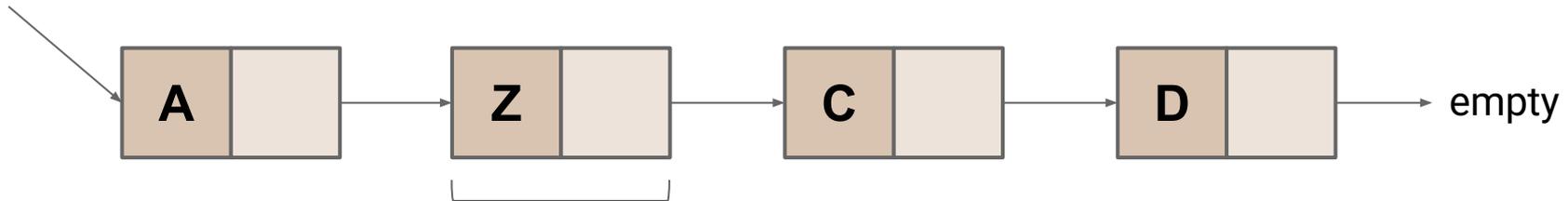
iterator

Linked List Iterator

`iterator.next() → C`

HEAD

A → Z → C → D → empty

iterator

Linked List Iterator

iterator.add(X)

HEAD

| A | | Z | | C | | D | | empty |

iterator

X

61
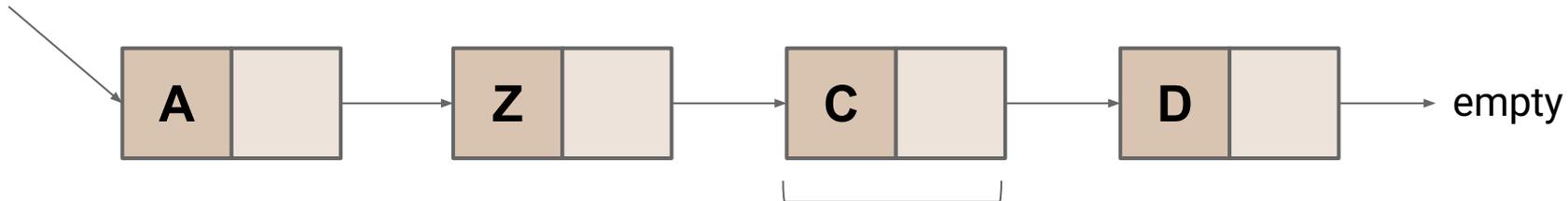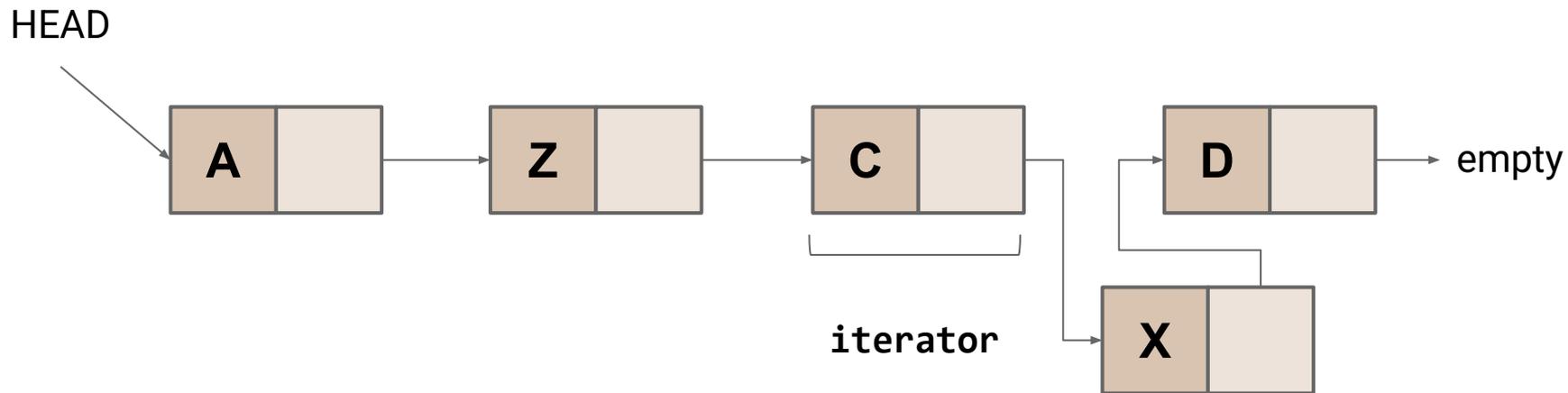
Linked List Iterator

```java
public class LinkedListIterator<E>
  extends ListIterator<E> {
  LinkedList<E> list;
  Optional<LinkedListNode<E>> before = Optional.empty();
  Optional<LinkedListNode<E>> after = Optional.of(list.head);
  /* ... */
}
```

Linked List Iterator

`boolean hasNext()`
    If **`after.isPresent()`**, return true   *O*(**1**)

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

`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

```java
public void int sumUpList(List<Integer> list) {
  int total = 0;
  ListIterator<Integer> iterator = list.iterator();
  while(iterator.hasNext()) {
    int value = iterator.next();
    total += value;
  }
  return total;
}
```

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();          Generalized to work with any kind of list!
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    }                          Loop body only contains Θ(1) operations
8    return total;
9  }
```

Enumeration using ListIterator

```java
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…