

What's an Iterator?

In a nutshell it's another way to traverse lists.

So far, we've mainly used for loops to traverse through elements in an array or list.

There is the traditional for loop and an index to index into a list.

And also, the enhanced for loop and a collection to step through the elements one at a time.

We've used both quite a few times in the course.

Let's look at how the Iterator works.

How does an Iterator work?

If you're familiar with databases, you might be familiar with a database cursor, which is a mechanism that enables traversal over records in a database.

An iterator can be thought of as something similar to a database cursor.

Specifically, an iterator is an object that allows traversal over records in a collection.

How does an Iterator work?

The Iterator is pretty simple.

When you get an instance of an iterator, you can call the **next** method to get the next element in the list.

You can use the **hasNext** method to check if any elements remain to be processed.

In the code, you can see a while loop which uses the iterator's **hasNext** method to determine if it should continue looping

In the loop, the **next** method is called and its value assigned to a local variable and the local variable printed out.

This prints each element in a list but does it through or via the iterator object.

How does an Iterator work?

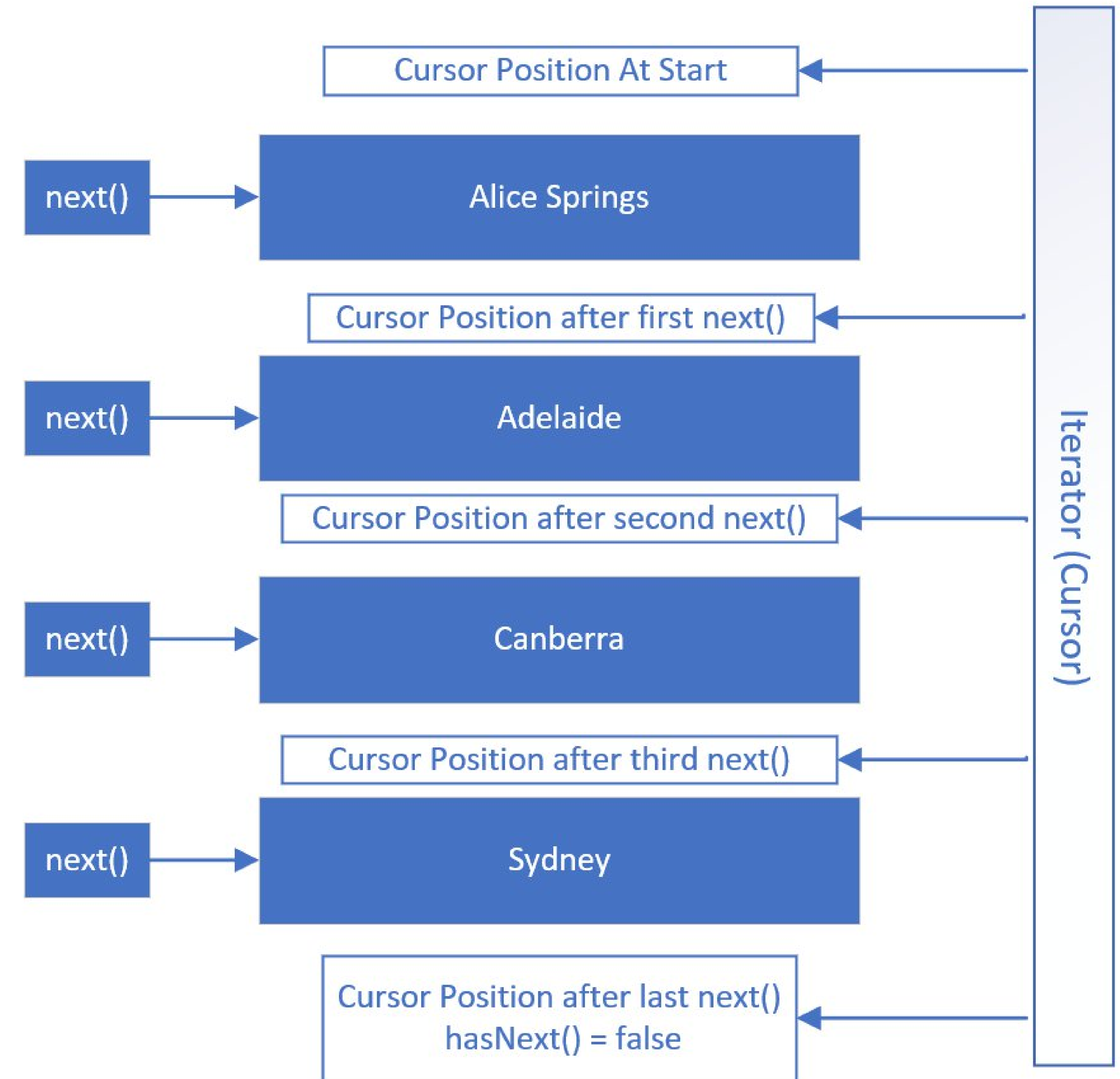
This slide shows visually how an Iterator works using the PlacesToVisit List.

When an iterator is first created, its cursor position is pointed at a position **before** the first element.

The first call to the **next** method retrieves the first element and moves the cursor position to be between the first and second elements.

Subsequent calls to the **next** method moves the iterator's position through the list, as shown, until there are **no elements left**, meaning **hasNext = false**.

At this point, the iterator or cursor position is below the last element.



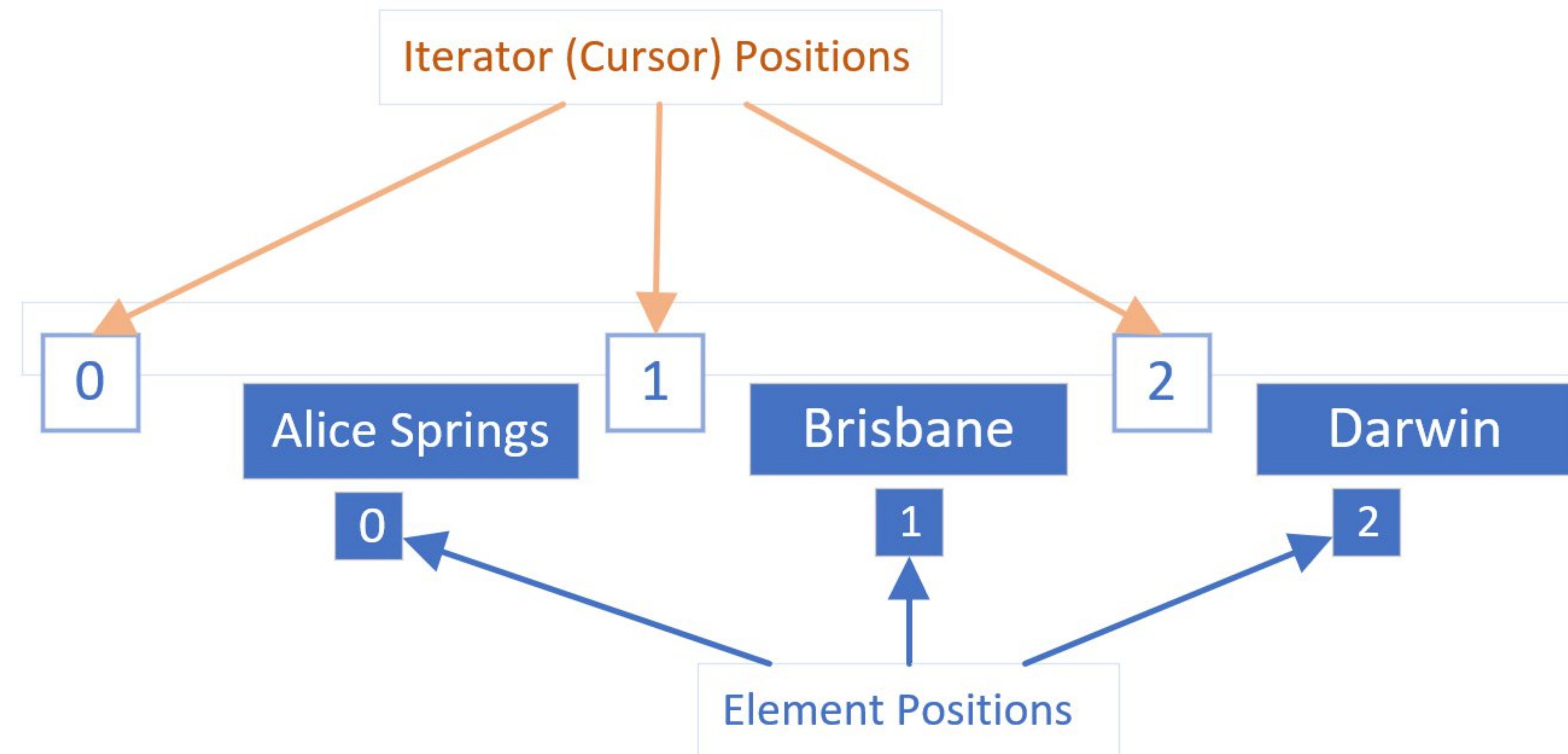
Iterator vs. ListIterator

An Iterator is forwards only and only supports the **remove** method.

A ListIterator allows you to navigate both forwards and backwards. Besides the **remove** method, it also supports the **add** and **set** methods, which function as you probably expect.

Iterator positions vs. Element positions

It's really important to understand that the iterator's cursor positions are **between** the elements.



```
var iterator = list.listIterator();
```

```
String first = iterator.next(); // Alice Springs returned, cursor moved to  
// cursor position 1
```

```
String second = iterator.next(); // Brisbane returned, cursor moved to cursor  
// position 2
```

```
// Reversing Directions
```

```
String reversed = iterator.previous(); // Brisbane returned, cursor moved to  
// cursor position 1
```