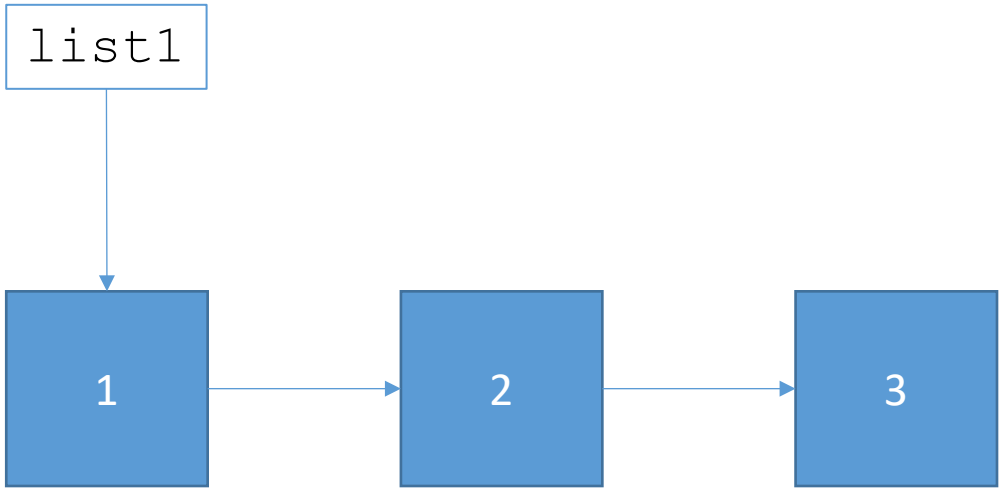
Scala Lists

Peter Chapin

Vermont State University

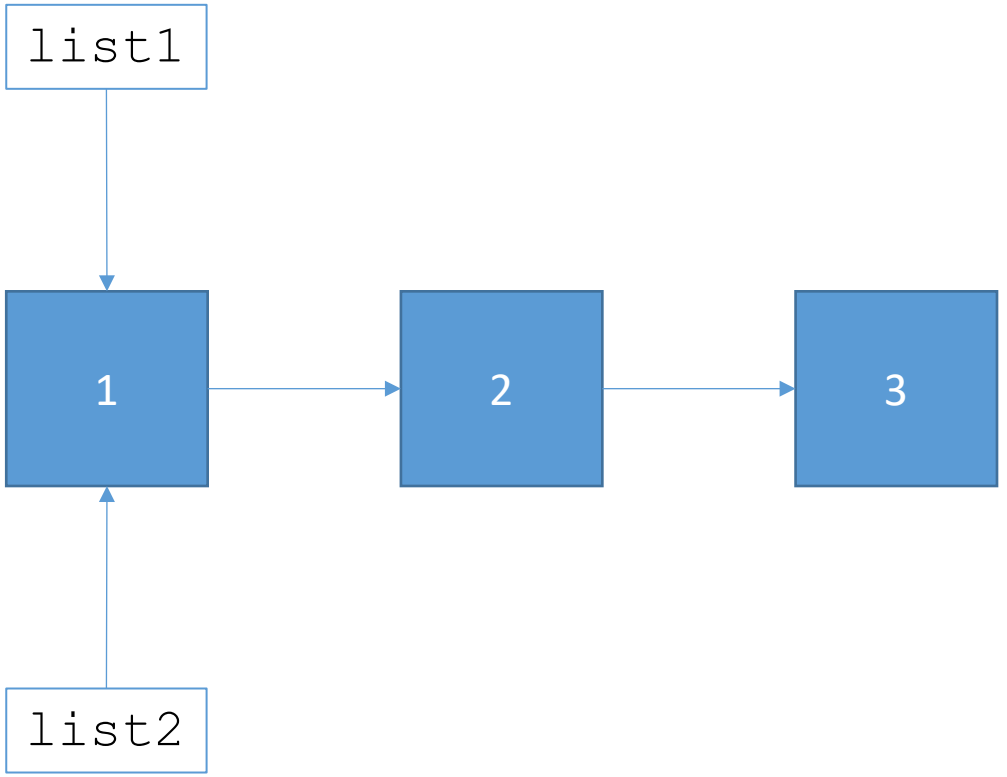
Create a List

```
val list1 = List(1, 2, 3)
```



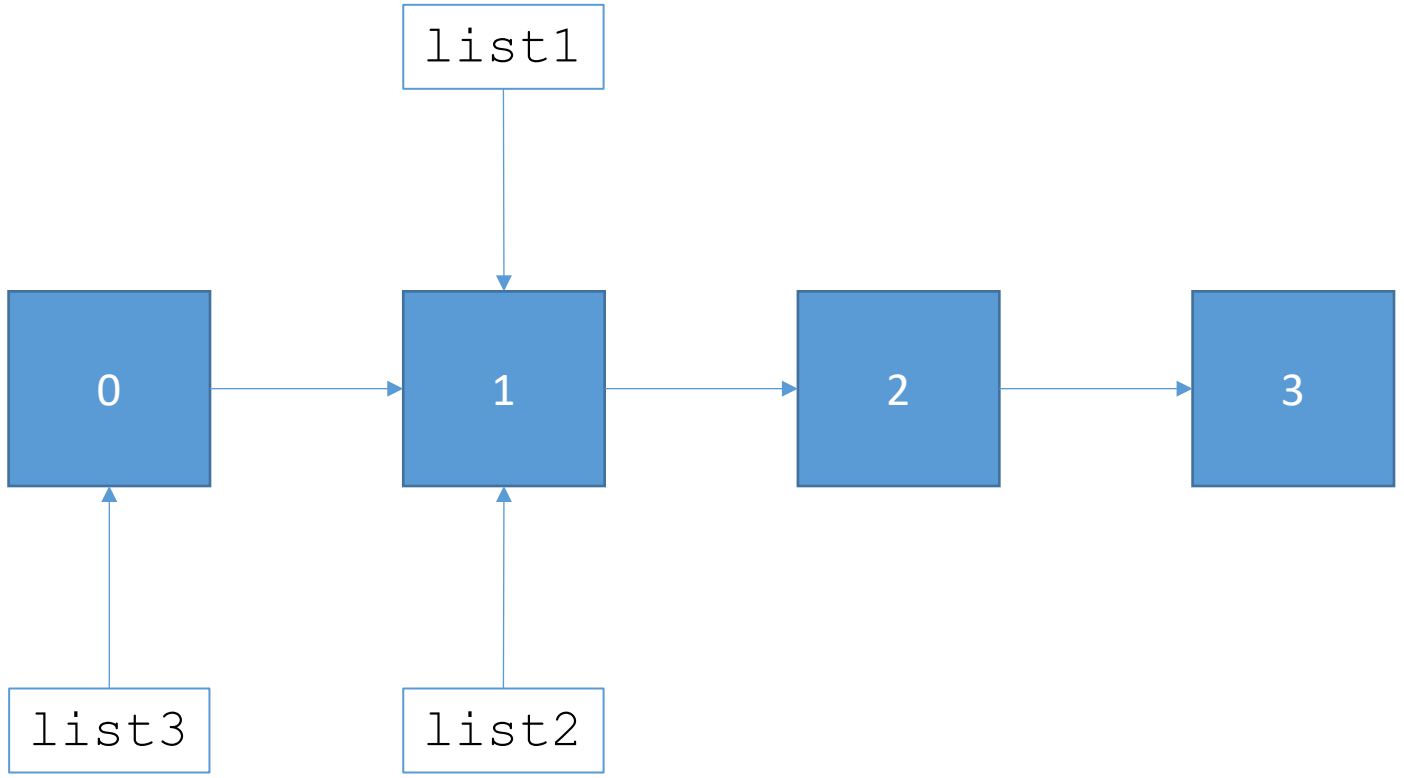
Create(?) a Second List

```
val list2 = list1
```



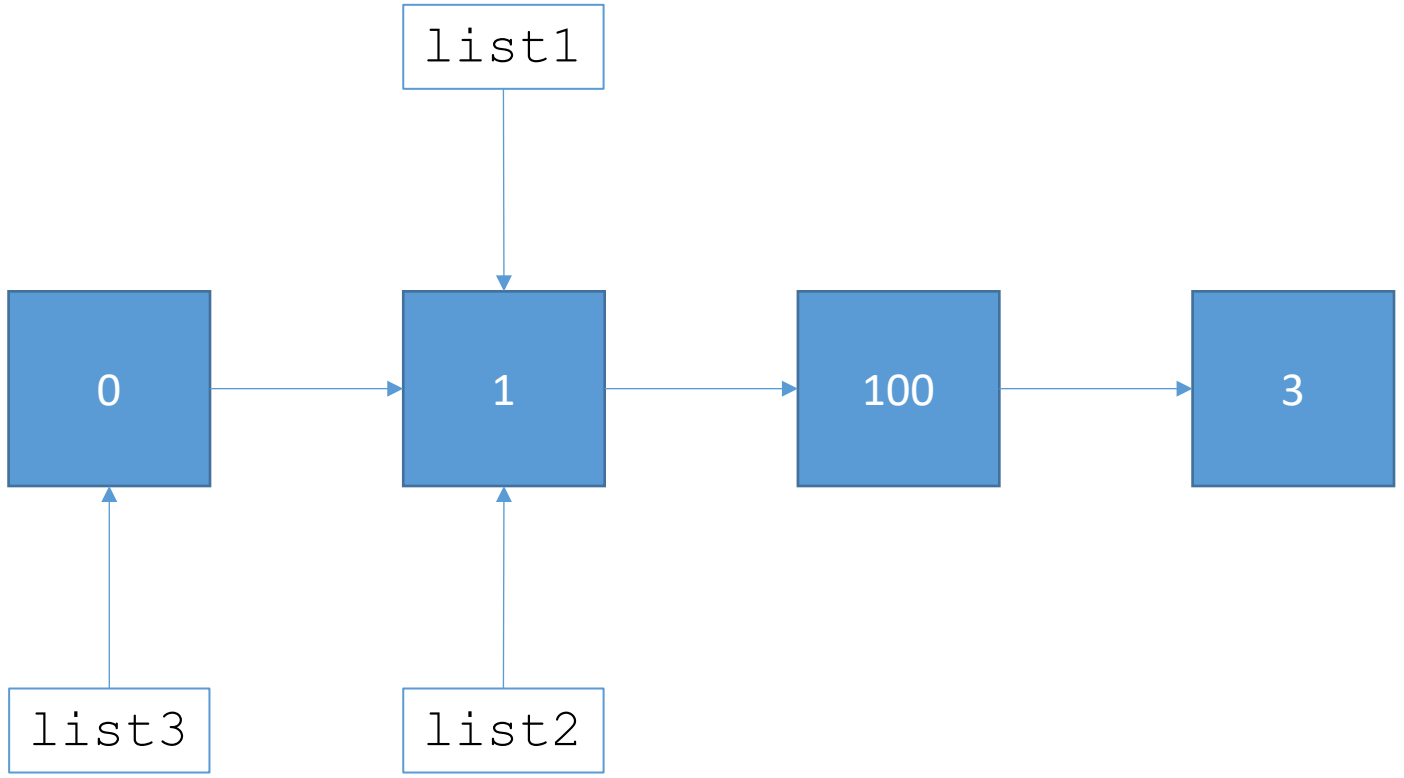
Cons (Prepend) an Element

```
val list3 = 0 :: list2
```



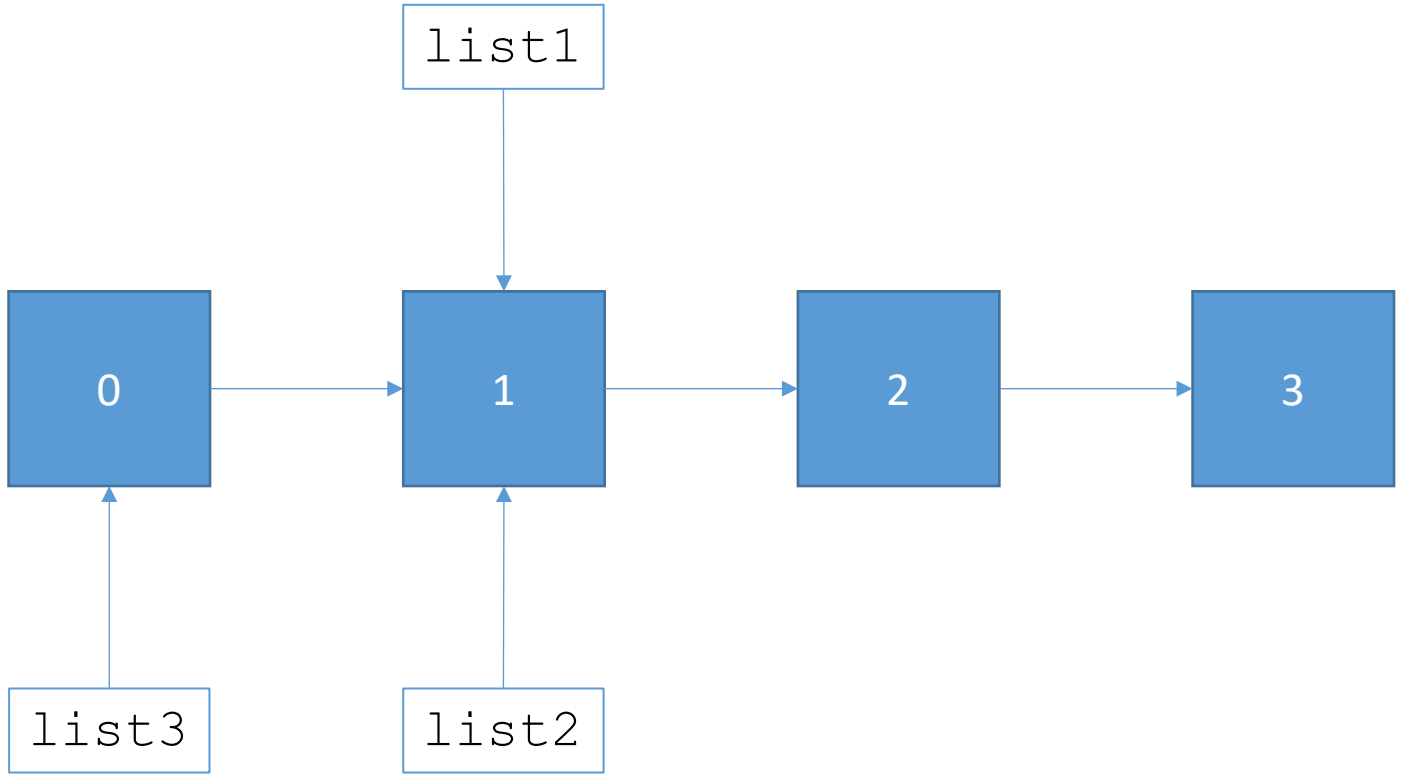
Modifying List3 Changes List1 and List2?

```
list3(2) = 100
```



No! Lists are Immutable

```
list3(2) = 100    error: value update is not a member of List[Int]
```



Summary

- Because of List immutability, different List objects can share nodes
  - Nobody can tell because nodes can never be modified†
  - Thus, creating a "new" list after each list operation is not always expensive.
  - `1 :: superLongList` does *not* entail making a copy of `superLongList`.
- *Functional data structures* try to share representation
  - You can reason as if all objects are distinct
  - … without paying the price of actually making zillions of copies.
  - We will see how this helps later with more complex examples.
- *Immutability is the key!*

† This is not entirely true; there are ways to tell

Arrays

- In Scala arrays are mutable; their elements can be modified

```
val array1 = Array(1, 2, 3)
val array2 = array1
array2(1) = 100
```

  - Now `array1` is `Array(1, 100, 3)`
  - There is only one Array object; both vals reference it
  - … and the object can be modified “out from underneath” one of the vals.
  - This behavior is compatible with Java (Scala arrays are the same as Java arrays).
- Mutability makes reasoning about program behavior harder.

Aren’t Vals Immutable?

- Yes! Once a val has been bound it can never refer to a different object
  - … but the mutability of that object is a separate matter!

```
val myArray = Array(1, 2, 3)
myArray = Array(4, 5, 6)  // Error! Can’t reassign a val
myArray(1) = 100          // Fine. The Arrays are mutable
```

- Vars can be bound to a different object
  - … even if that object is immutable

```
var myList = List(1, 2, 3)
myList = List(4, 5, 6)    // Fine. Vars can be reassigned
myList(1) = 100           // Error! Lists are immutable
```

- *Use vals by default; vars only when necessary!*