Honors Data Structures

Lecture 3: Immutable Lists in Scala.

02/31/2022

Daniel Bauer

Purely Functional Data Structures

- In the functional programming paradigm, functions must not have any "side effects". They must not change
  - the argument they are called on or instance variables of a class instance.
  - the instance variables of any class instance (if a method)
  - any other state of the program!
- Therefore, in functional programming, all objects are *immutable.*

Mutable and Immutable Data Structures

- The content of a *mutable* data structure can be modified at any time (example: ArrayList).
- Objects of an *immutable* data type cannot be changed once they have been created.
- How do we construct immutable data structures?
- How to efficiently implement operations like insert(x,k) if the underlying data structure can’t be modified?

Scala REPL (read/evaluate/print loop)

- Scala programs can be compiled like .java files
- There is also an interactive mode that interprets code line-by-line.

```
$ scala
Welcome to Scala 2.12.1 (Java HotSpot(TM) 64-
Bit Server VM, Java 1.8.0_121).
Type in expressions for evaluation. Or
try :help.

scala> print("hello world")
hello world
```

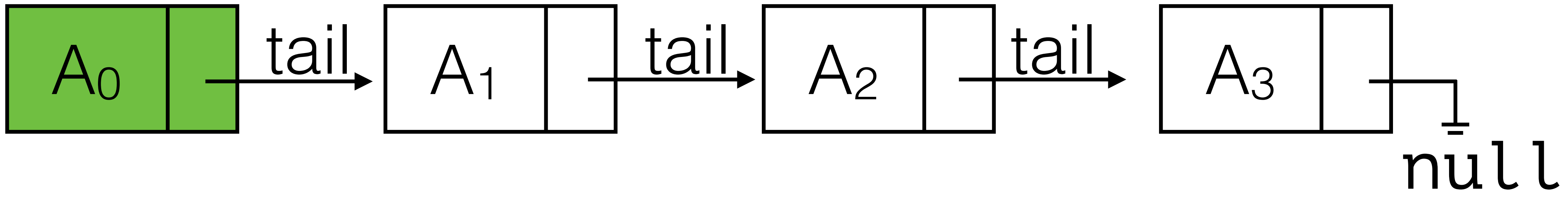
var and val

- There are two types of names in Scala:
  - **var** creates a new variable. A different object can be assigned to a var.
  - **val** creates a name for a value. Once assigned, no new object can be assigned to the val.

```
scala> var x = 27;
x: Int = 27

scala> x = 42;
x: Int = 42

scala> val y = 27;
y: Int = 27

scala> y = 23;
<console>:12: error: reassignment to val
       y = 23;
```

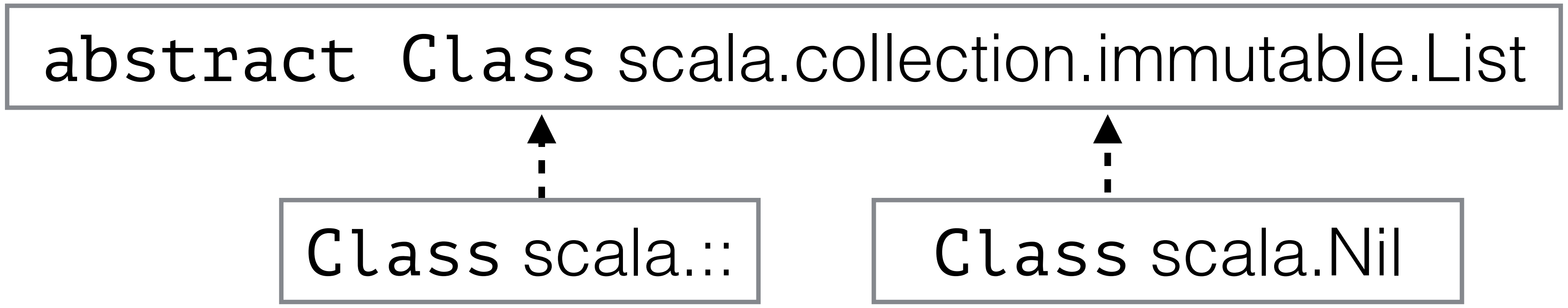
Immutable Lists

- Any insertion and deletion operation must leave the original List unchanged, including when the list is built.
- All operations that “change” the list must return a new list.
- The recursive definition of an immutable list is similar to the standard linked list. A list consists of:
  - a data item (the “head”).
  - a (possibly empty) List (the “tail”)



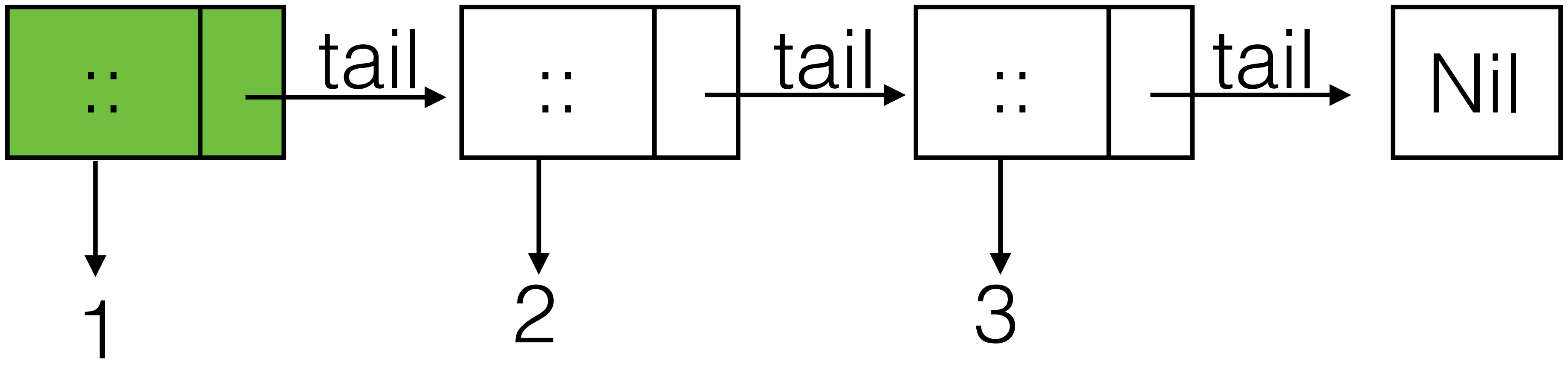
* careful with the terms “head” and “tail”! These are used differently in this definition.

Immutable Lists in Scala



- **Nil** represents the empty list.
- :: represents a non empty list with a head and tail. :: is pronounced "cons"

```
scala> val x = ::(1,::(2,::(3,Nil)))
x: scala.collection.immutable.::[Int] = List(1, 2, 3)
```



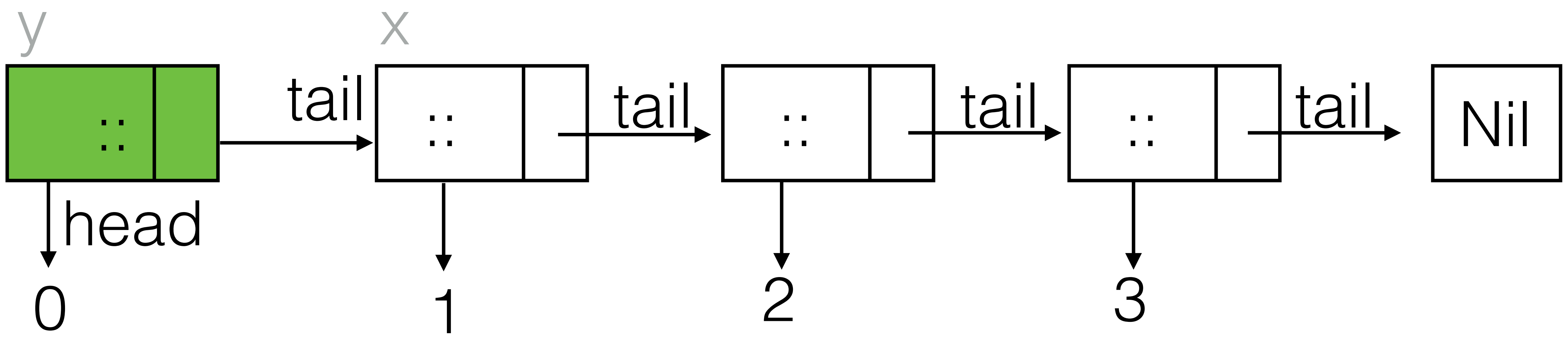
Immutable Lists in Scala

- Shortcut: We can also use :: as a binary operator (we will discuss why this works later).
- So instead of ::(3,Nil) we can write 3 :: Nil.

```
scala> val x = 1 :: 2 :: 3 :: Nil;
x: List[Int] = List(1, 2, 3)
```

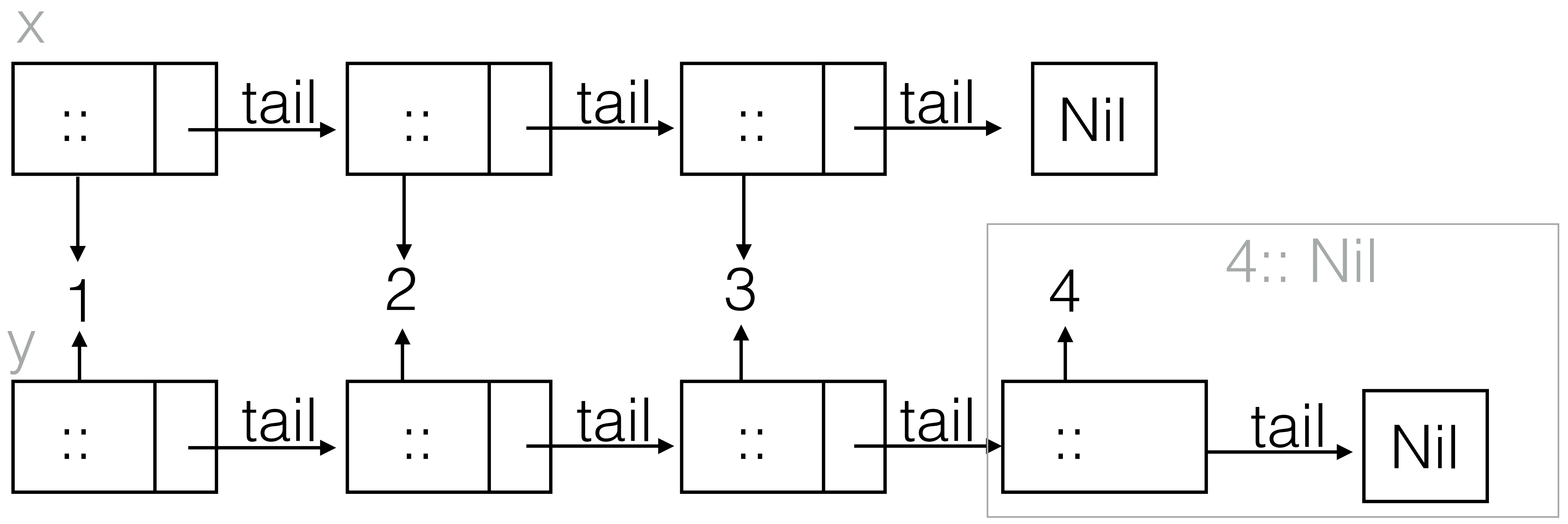
Immutable Lists in Scala: prepend

```
scala> val x = ::(1,::(2,::(3,Nil)))
x: scala.collection.immutable.::[Int] = List(1, 2, 3)
scala> val y = 0 :: x;
y: List[Int] = List(0, 1, 2, 3)
```



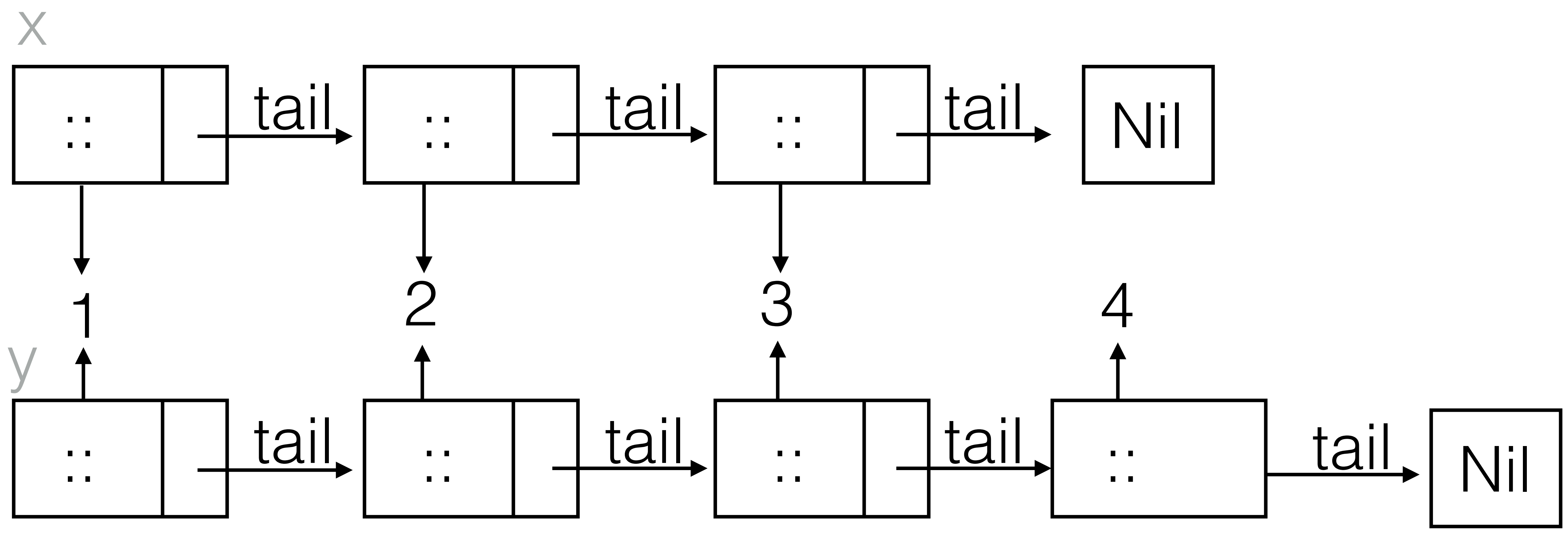
Immutable Lists in Scala: concatenating two lists

```
scala> val x = ::(1,::(2,::(3,Nil)))
x: scala.collection.immutable.::[Int] = List(1, 2, 3)
scala> val y = x ::: 4 :: Nil
res4: List[Int] = List(1, 2, 3, 4)
```



Immutable Lists in Scala: Appending to a list

```
scala> val x = ::(1,::(2,::(3,Nil)))
x: scala.collection.immutable.::[Int] = List(1, 2, 3)
scala> val y = x :+ 4
res4: List[Int] = List(1, 2, 3, 4)
```



Other ways to run Scala

- Run a "script" (identical to running multiple REPL lines in a batch).

```
$ scala Hello.scala
hello world
```

- Compiling to JVM bytecode, then running the bytecode (like Java).

```
$ scalac HelloProgram.scala
$ ls
Hello.scala HelloProgram.scala
HelloProgram.class
$ scala HelloProgram
hello world
```