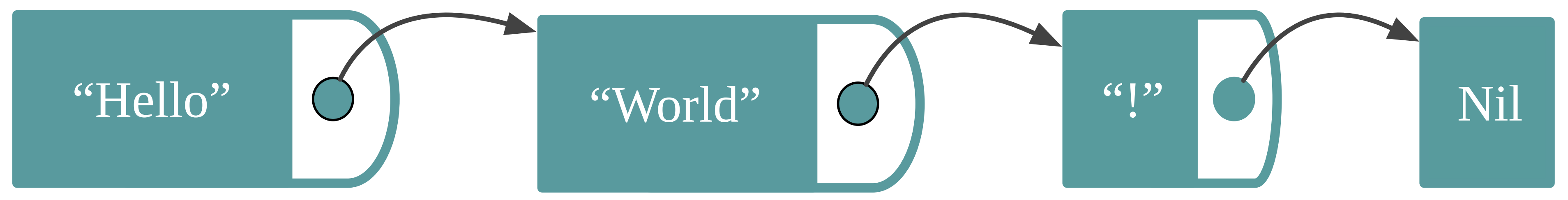
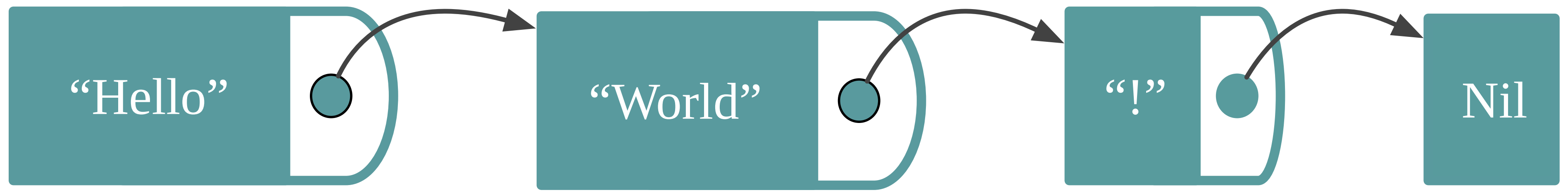
FOUNDATION

List

Linked list

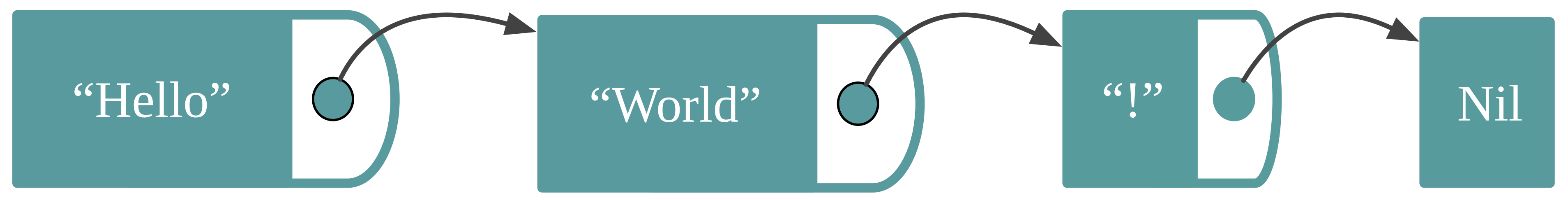


Linked list



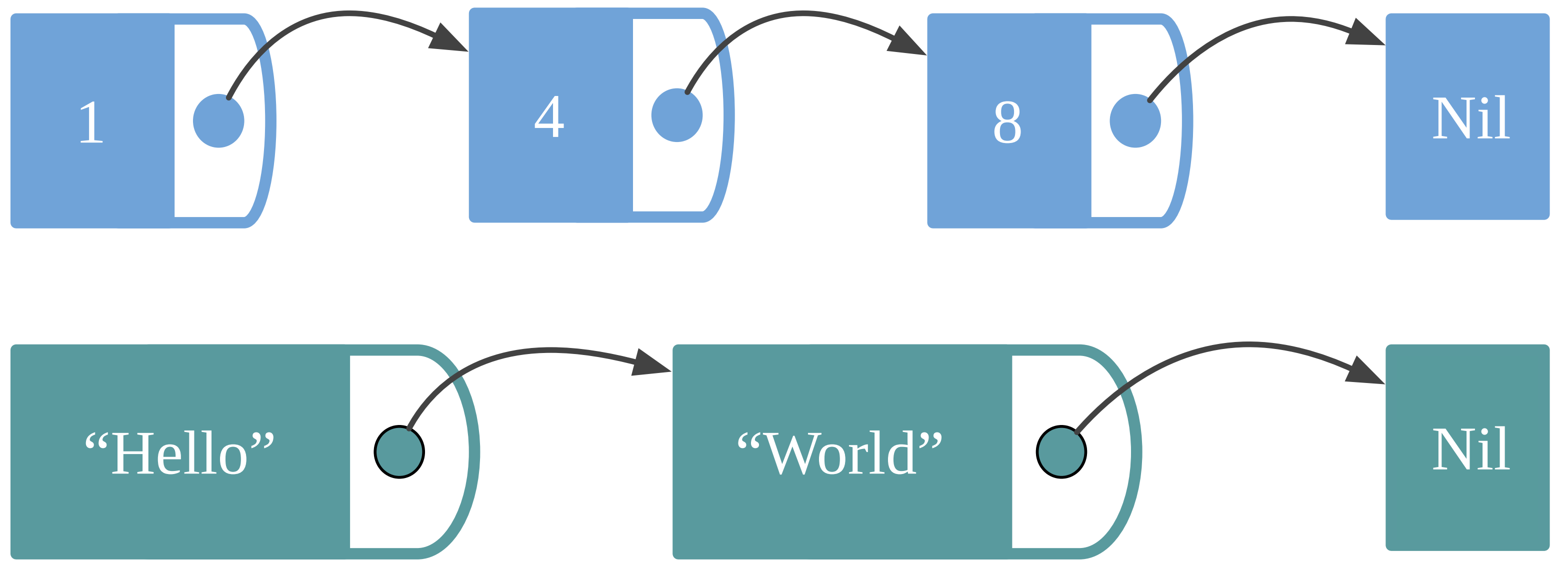
```
val words = List("Hello", "World", "!")
```

Linked list



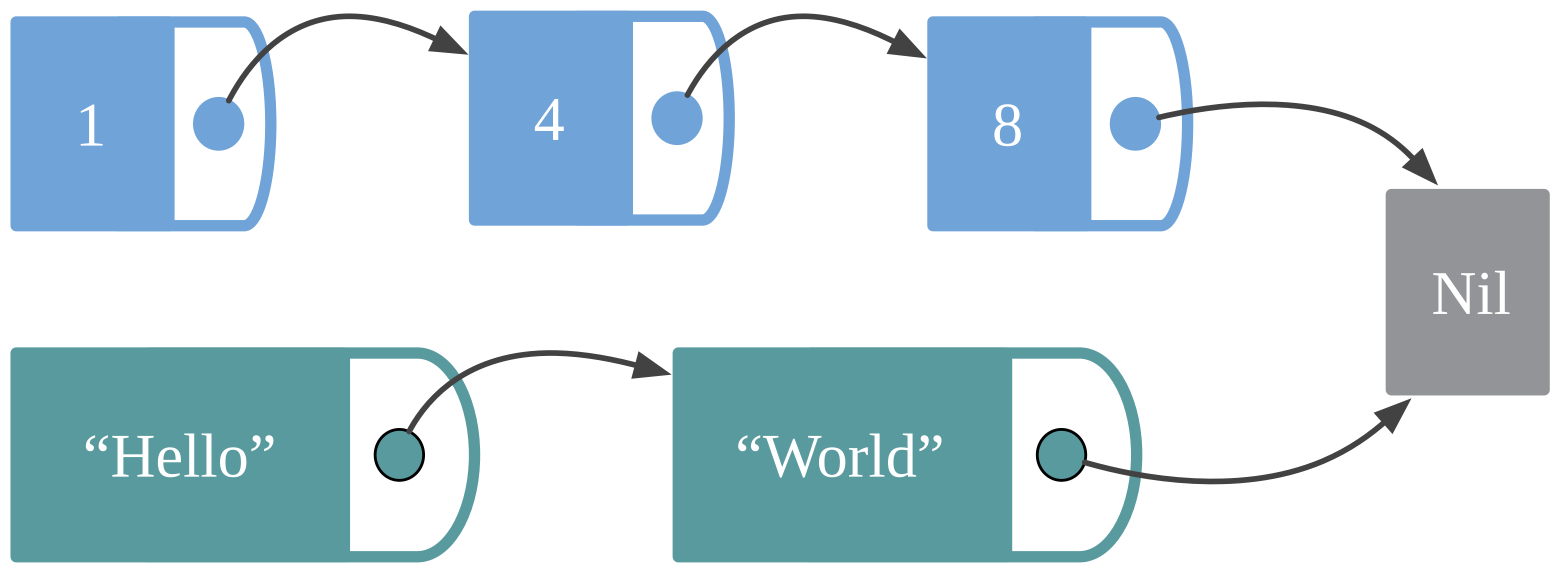
```
val words = "Hello" :: "World" :: "!" :: Nil
```

List is a parametric type



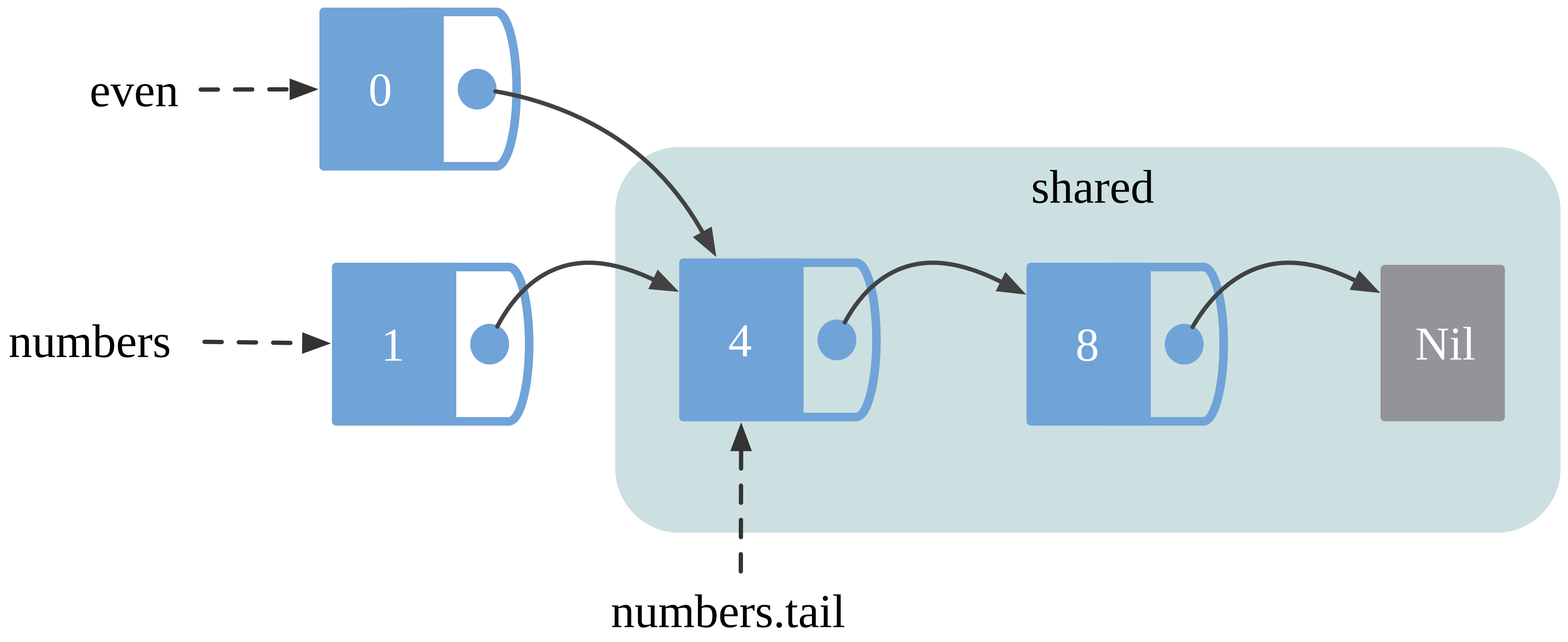
```scala
val numbers: List[Int]    = List(1, 4, 8)

val words  : List[String] = List("Hello", "World")
```

Nil is a list of all types



```
val nil = Nil

val numbers: List[Int]    = 1 :: 4 :: 8 :: nil

val words  : List[String] = "Hello" :: "World" :: nil
```

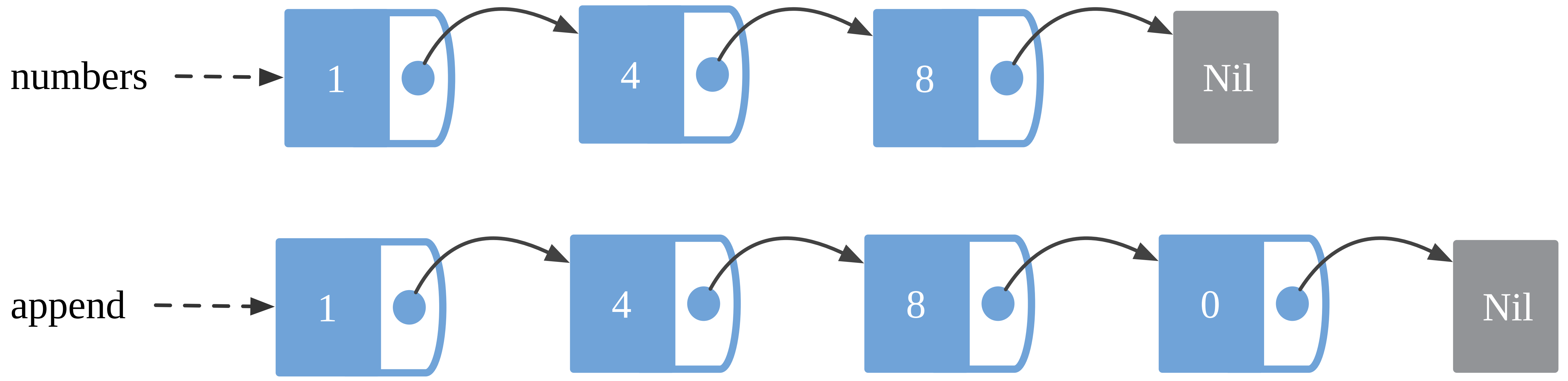
List is immutable



```
val numbers = 1 :: 4 :: 8 :: Nil
// numbers: List[Int] = List(1, 4, 8)
```

```
val even = 0 :: numbers.tail
// even: List[Int] = List(0, 4, 8)
```

List is immutable



```scala
val numbers = 1 :: 4 :: 8 :: Nil
// numbers: List[Int] = List(1, 4, 8)
```

```scala
val append = numbers :+ 0
// append: List[Int] = List(1, 4, 8, 0)
```

Persistent data structure

```scala
case class User(name: String, age: Int)

val users  = List(User("John", 17), User("Alice", 54), User("Bob", 23))

val adults = users.filter(_.age >= 18)
```

Persistent data structure

```
case class User(name: String, age: Int)

val users  = List(User("John", 17), User("Alice", 54), User("Bob", 23))

val adults = users.filter(_.age >= 18)
```

```
users.length
// res5: Int = 3

adults.length
// res6: Int = 2
```

```
users.length != adults.length
// res7: Boolean = true
```

Easier to test

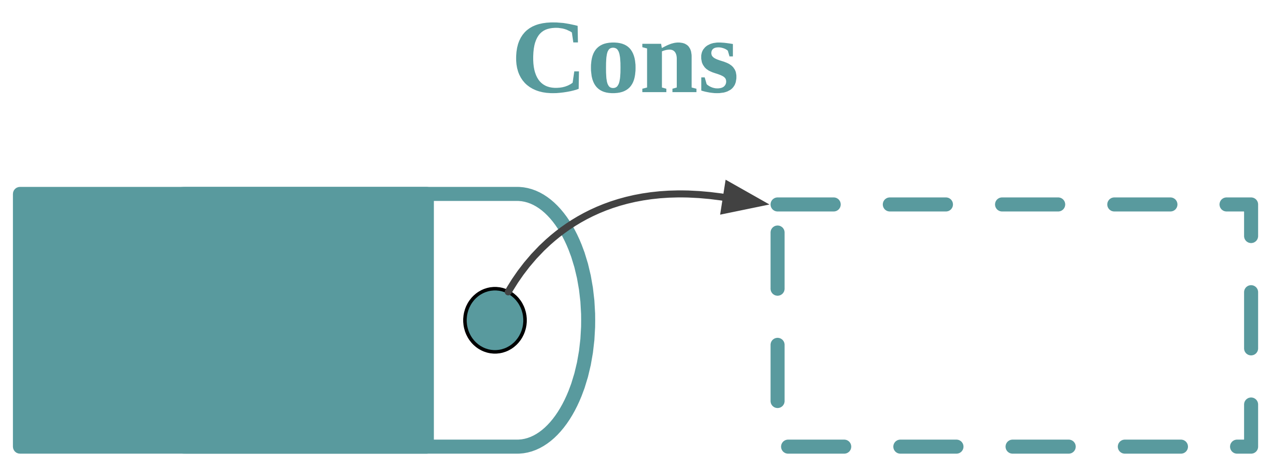
Concurrency

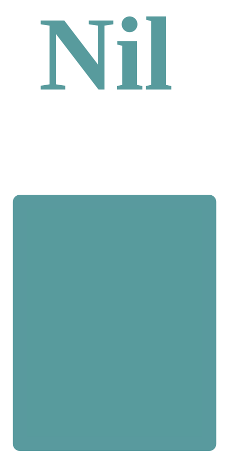
Corecursive podcast

Rust and bitter C++ developers with Jim Blandy

Why List is so popular?

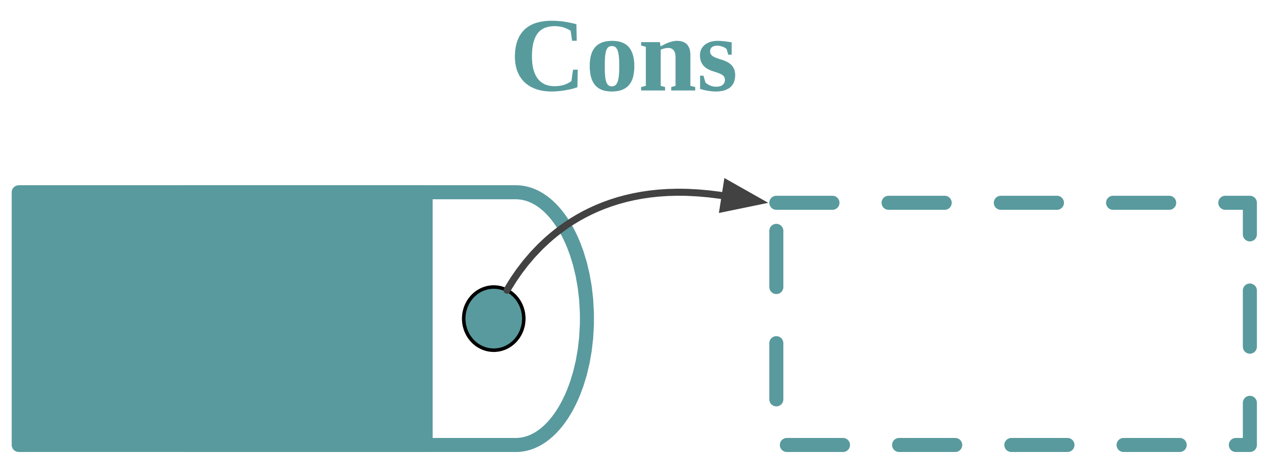
List is an enumeration





List is an enumeration

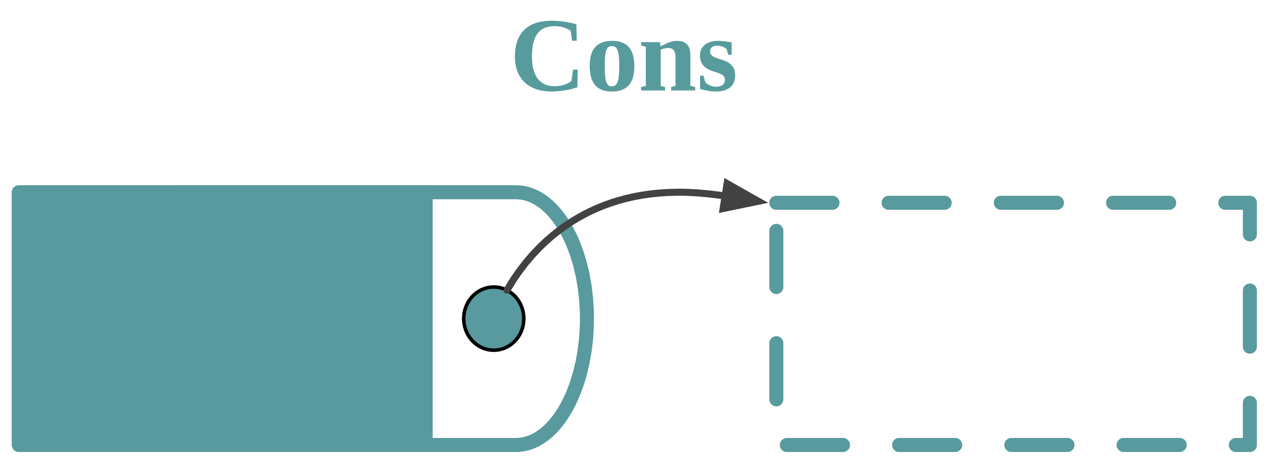
```scala
sealed trait List[+A]


case class Cons[+A](head: A, tail: List[A])
  extends List[A]


case object Nil extends List[Nothing]
```



List is an enumeration

```
enum List[+A] {
  case class Cons(head: A, tail: List[A])
  case object Nil
}
```

In Dotty/Scala 3



Pattern matching

```
list match {
  case Cons(head, tail) => println("list is not empty")
  case Nil              => println("list is empty")
}
```

Pattern matching

```
list match {
  case Cons(head, tail) =>
    tail match {
      case Cons(second, rest) => println("list is has at least 2 elements")
      case Nil                => println("list is has 1 element")
    }
  case Nil                => println("list is empty")
}
```

Pattern matching

```
list match {
  case Cons(head, Cons(second, tail)) => println("list is has at least 2 elements")
  case Cons(head, Nil)                => println("list is has 1 element")
  case Nil                            => println("list is empty")
}
```

Pattern matching with infix Cons

```
list match {
  case head :: second :: tail => println("list is has at least 2 elements")
  case head :: tail           => println("list is has 1 element")
  case Nil                    => println("list is empty")
}
```

```
"Hello" :: "World" :: Nil
```

Summary

- Extremely simple data structure
- Easy to test and safe to share
- Good for pattern matching, prepending elements, iterating
- Bad for appending elements, random access, size