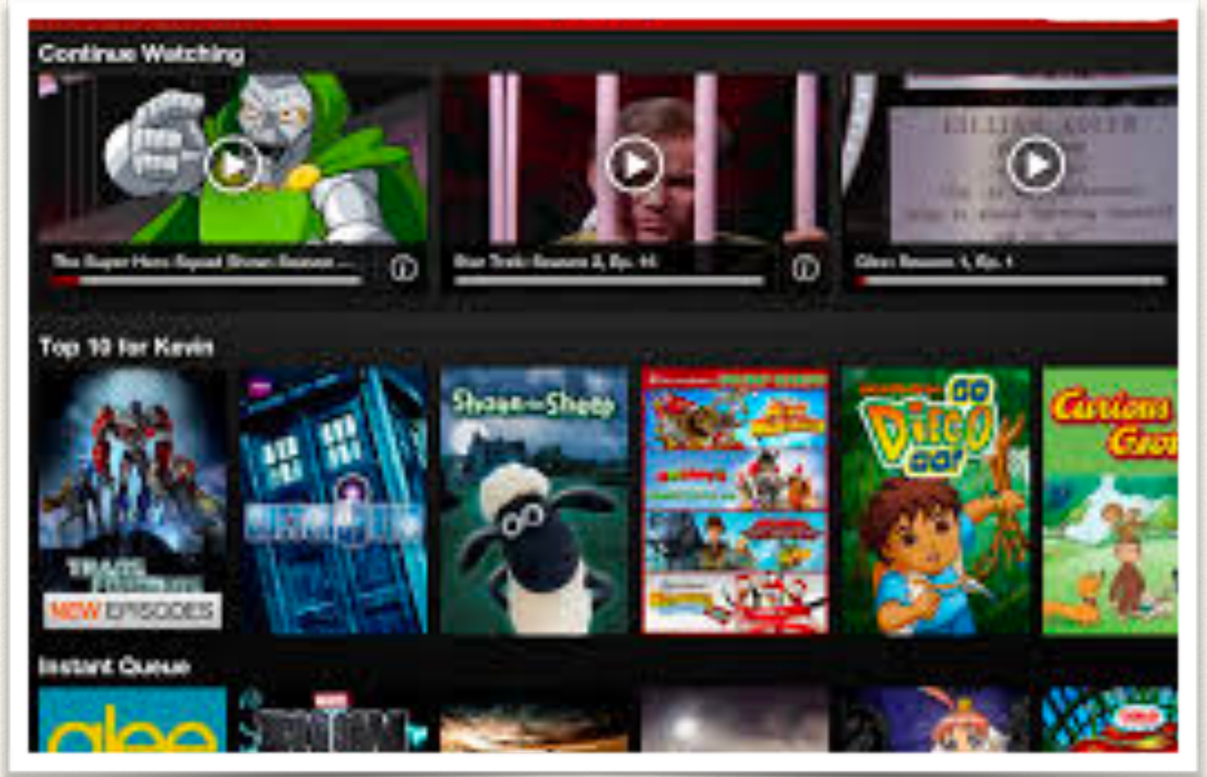
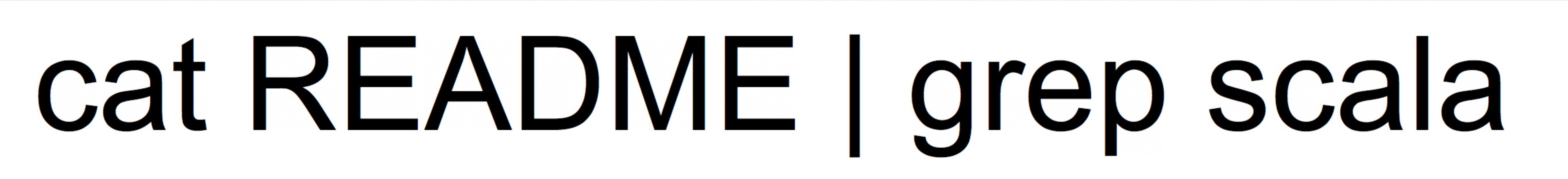
Streams

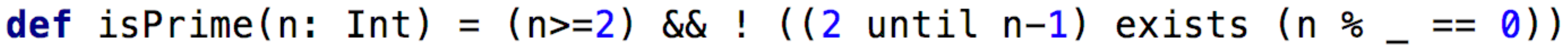


Continue Watching
The Super Hero Squad Show: Season ...
Star Trek: Season 3, Ep. 14
Glee: Season 1, Ep. 1
Top 10 for Kevin
NEW EPISODES
Shaun the Sheep
Go Diego Go!
Curious George
Instant Queue
glee

amazon
instant video

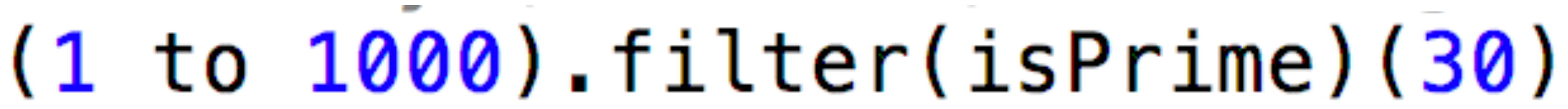


```
cat README | grep scala
```

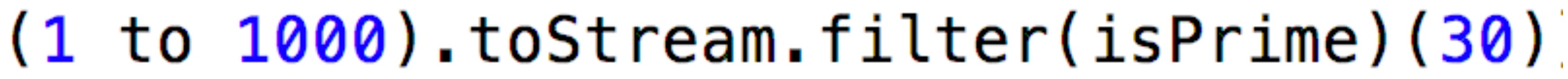


```
def isPrime(n: Int) = (n>=2) && ! ((2 until n-1) exists (n % _ == 0))
```

A textbook example:
Find the 31-st prime number



```
(1 to 1000).filter(isPrime)(30)
```



```
(1 to 1000).toStream.filter(isPrime)(30)
```

DEMO

A scala example

- Example: Find the 31-st prime number
- Use your laptop to find the answer, then we share the numbers we got, and time elapsed for the laptop to do the computing

DEMO

So, why Streams ?

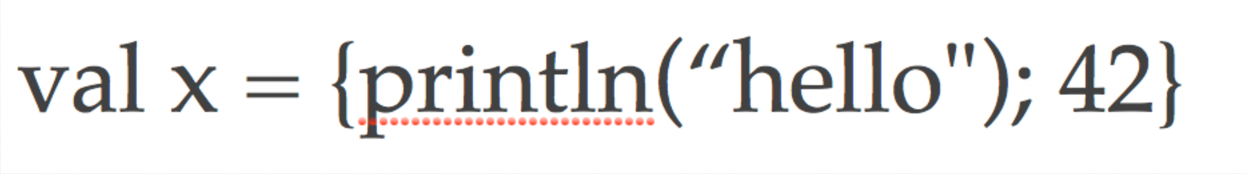
- Performance

- Large data

Concepts for implementing streams

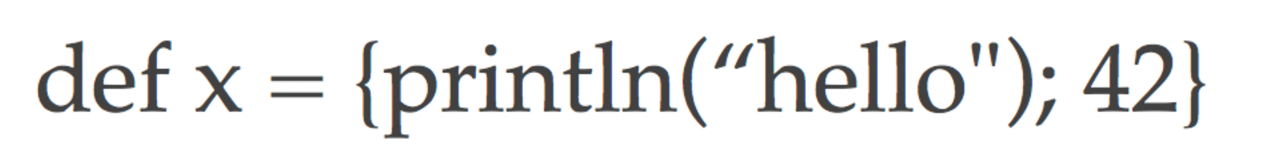
- Evaluation strategies
- Strictness / laziness

Three Kinds of Evaluation Strategies



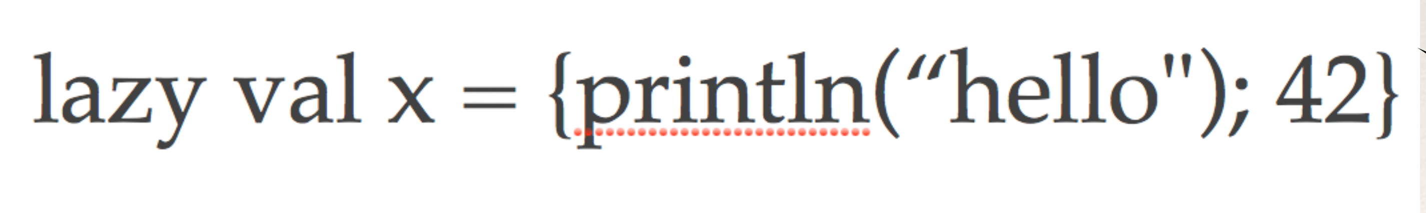
```
val x = {println(“hello"); 42}
```

By-value



```
def x = {println(“hello"); 42}
```

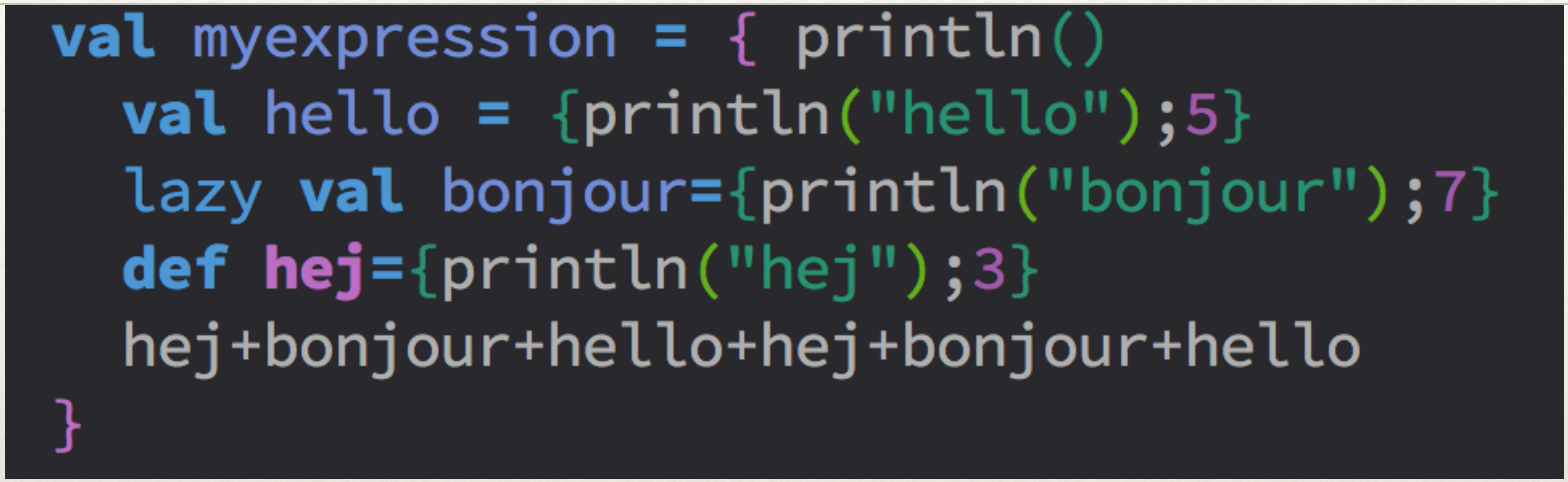
By-name



```
lazy val x = {println(“hello"); 42}
```

By-need

Discuss in groups of 2-3 people



```
val myexpression = { println()
  val hello = {println("hello");5}
  lazy val bonjour={println("bonjour");7}
  def hej={println("hej");3}
  hej+bonjour+hello+hej+bonjour+hello
}
```

- What will be the output?
- Remind:
  - val: immediately
  - lazy val: first access
  - def: each access

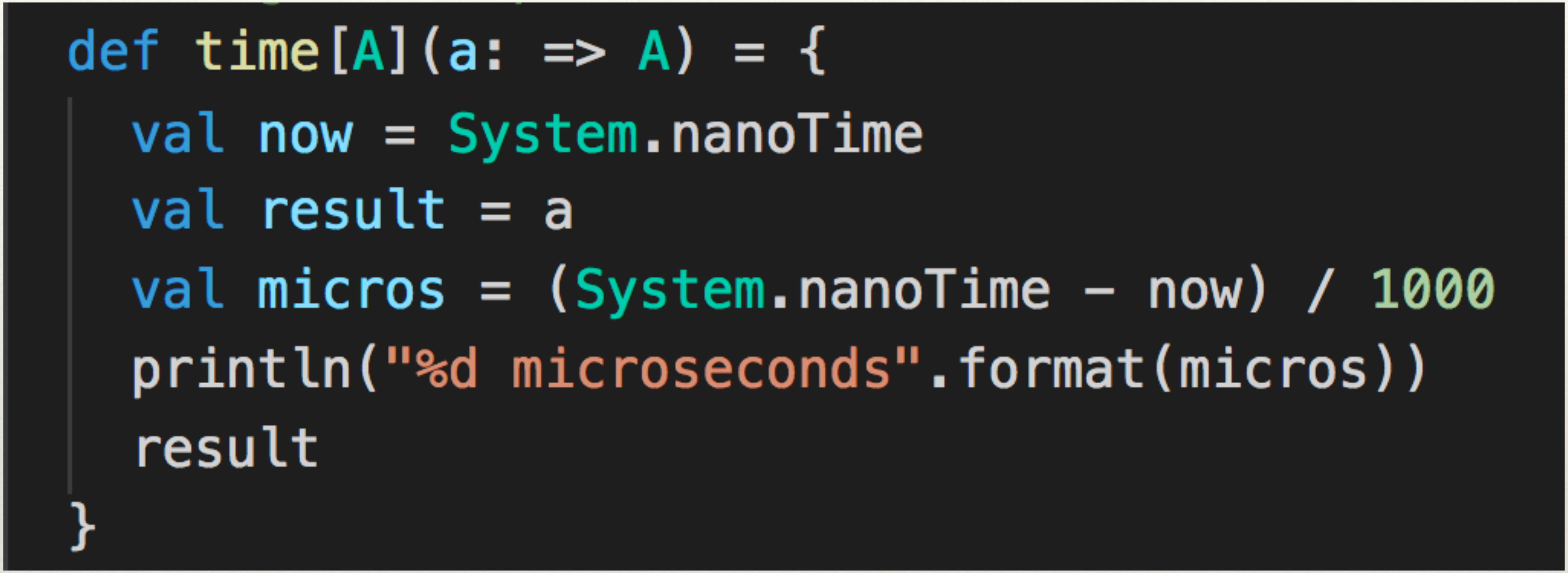
DEMO

Strictness/Laziness

- We use the terms strictness/laziness on evaluation strategies of function calls
- A strict function evaluates all of its arguments
    - Scala functions are strict by default
- A lazy function evaluates arguments by need
    - &&, | |

DEMO

An application of lazy function

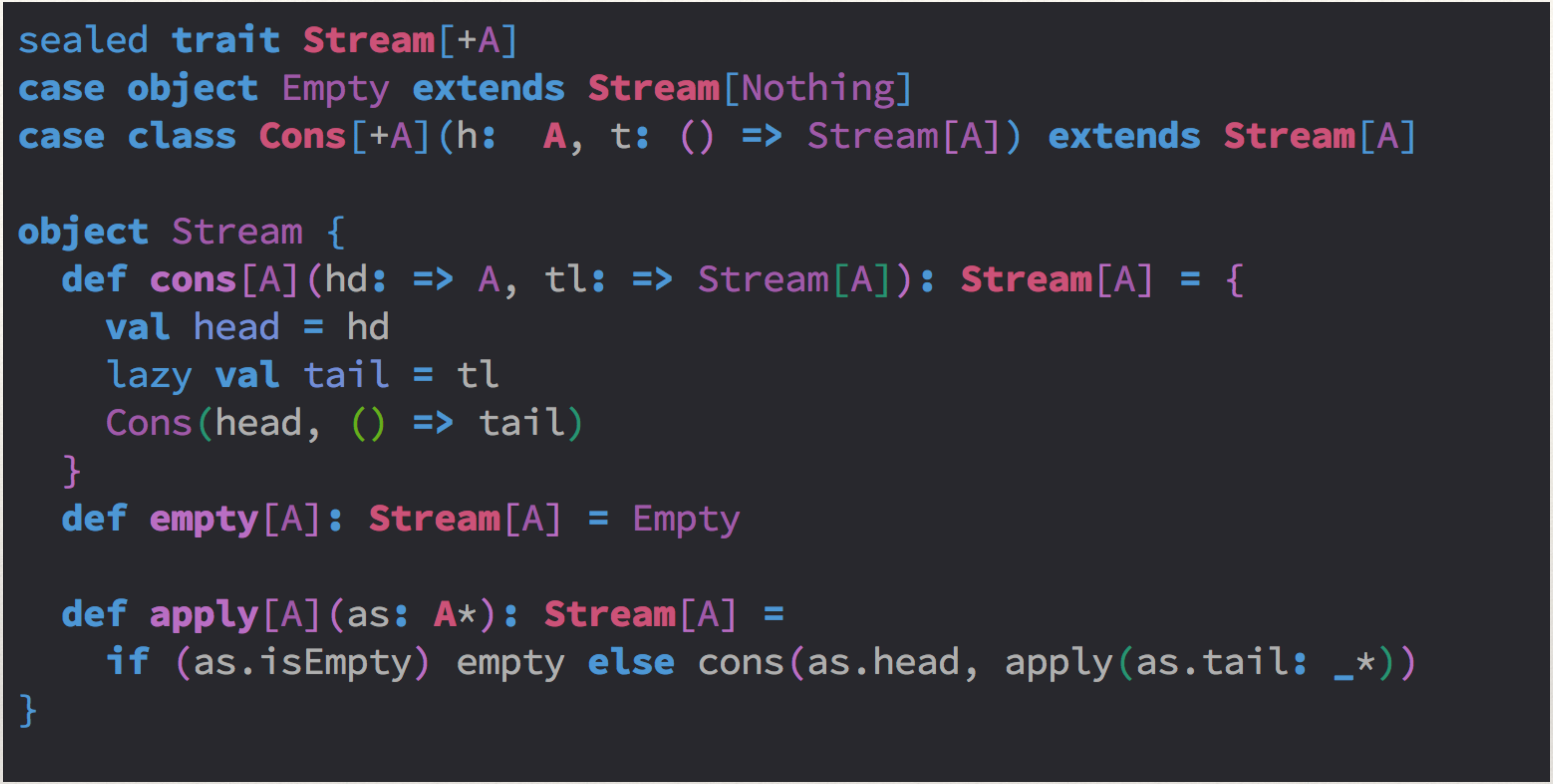


```scala
def time[A](a: => A) = {
  val now = System.nanoTime
  val result = a
  val micros = (System.nanoTime - now) / 1000
  println("%d microseconds".format(micros))
  result
}
```

DEMO

Implementation:

Stream = Lazy list



```scala
sealed trait Stream[+A]
case object Empty extends Stream[Nothing]
case class Cons[+A](h:  A, t: () => Stream[A]) extends Stream[A]

object Stream {
  def cons[A](hd: => A, tl: => Stream[A]): Stream[A] = {
    val head = hd
    lazy val tail = tl
    Cons(head, () => tail)
  }
  def empty[A]: Stream[A] = Empty

  def apply[A](as: A*): Stream[A] =
    if (as.isEmpty) empty else cons(as.head, apply(as.tail: _*))
}
```

DEMO

Quiz

- Implement get[A](n:Int, s:Stream[A]): A that retrieves the nth item of stream s
- Implement filter[A](p: A => Boolean, s:Stream[A]): Stream[A]
- Implement streamRange[A](l:Int,h:Int):Stream[A] that gets the stream from l to h
- Test your implementation with this line: "get(30, filter (isPrime, streamRange(1,1000)))"

Performance comparison

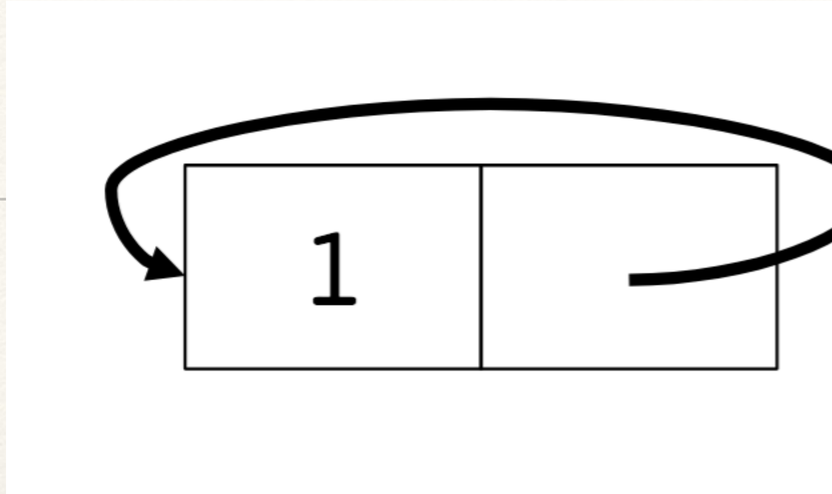
- Compare the time of running
  - get(30, filter (isPrime, streamRange(1,1000))), and
  - (1 to 1000).filter(isPrime)(30)
- Think (again) why the former is faster

DEMO

Streams in the real world:

```
import scala.collection.immutable.Stream
```

Infinite List



- import scala.collection.immutable.Stream
- val ones: Stream[Int] = Stream.cons(1, ones)
- ones(1000)
- Stream(1,2,3,4).map(_ + 10).filter(_ % 2 == 0).toList
-

DEMO

Conclusions

- By-value, by-name and by-need evaluations
- Strictness and laziness
- Stream is a useful for handling large data efficiently