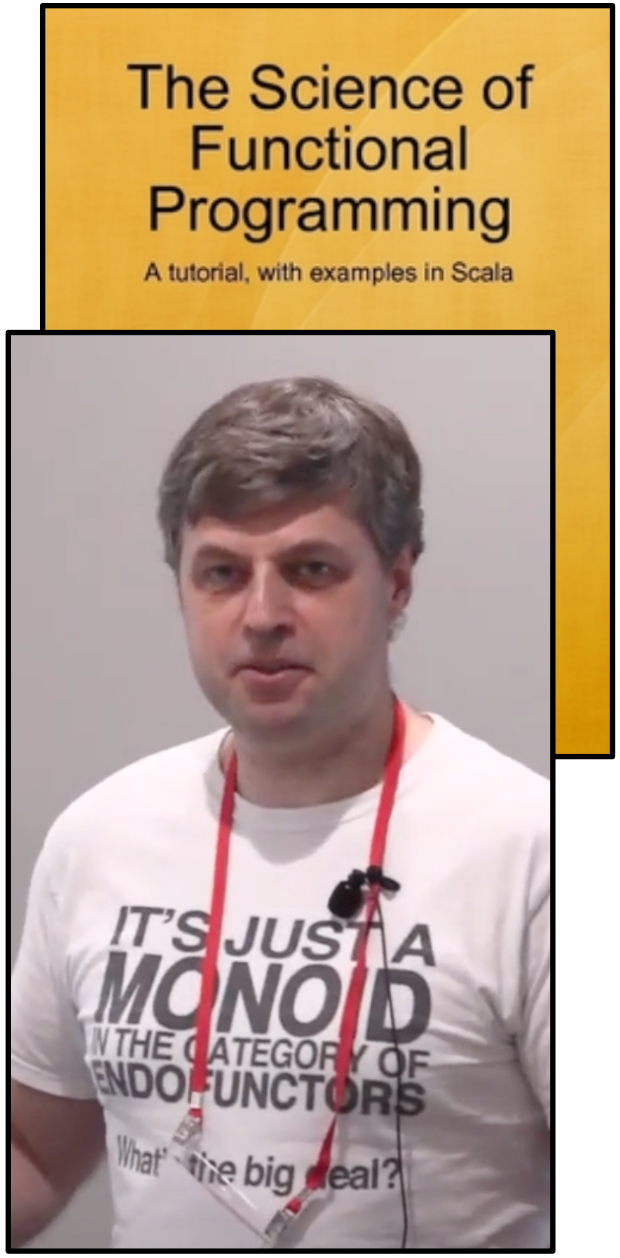
Iterative calculations are implemented by translating **mathematical induction** directly into code. In the **functional programming paradigm**, the programmer does not need to write any **loops** or use **array indices**. Instead, the programmer reasons about **sequences** as **mathematical values**: "Starting from this value, we get that **sequence**, then **transform** it into this other **sequence**," etc.

This is a powerful way of working with **sequences**, dictionaries, and sets. Many kinds of **programming errors** (such as an incorrect **array index**) are avoided from the outset, and the code is **shorter** and **easier to read** than conventional code written using **loops**.

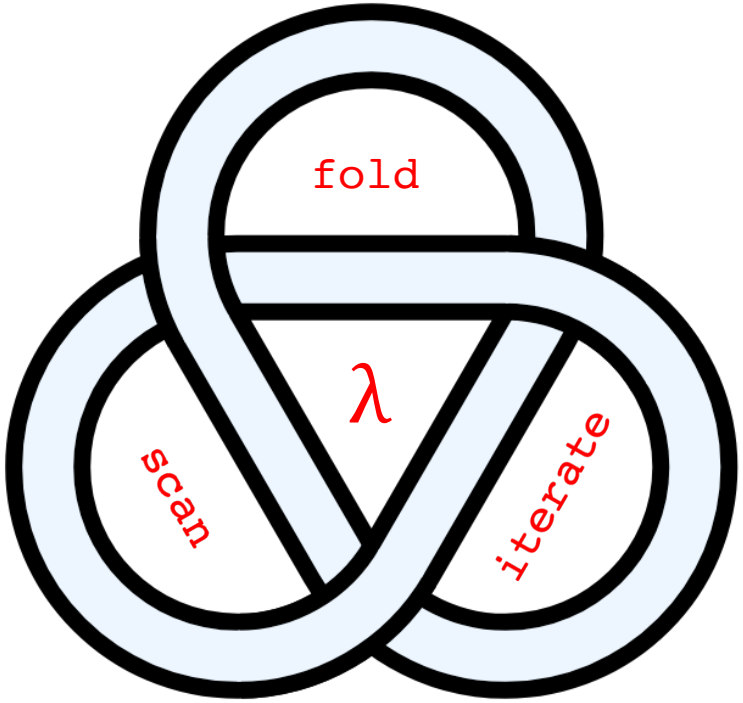
Definition by induction	Scala code example
$f([]) = b\,;\;\; f(s{+\!\!+}[x]) = g(f(s), x)$	`f(xs) = xs.foldLeft(b)(g)`
$x_0 = b\,;\;\; x_{k+1} = g(x_k)$	`xs = Stream.iterate(b)(g)`
$y_0 = b\,;\;\; y_{k+1} = g(y_k, x_k)$	`ys = xs.scanLeft(b)(g)`

Implementing **mathematical induction**



Sergei Winitzki
sergei-winitzki-11a6431

Definition by induction	Scala code example
$f([]) = b\,;\;\; f(s{+\!\!+}[x]) = g(f(s), x)$	`f(xs) = xs.foldLeft(b)(g)`
$x_0 = b\,;\;\; x_{k+1} = g(x_k)$	`xs = Stream.iterate(b)(g)`
$y_0 = b\,;\;\; y_{k+1} = g(y_k, x_k)$	`ys = xs.scanLeft(b)(g)`



Implementing mathematical induction

Haskell

```
Haskell> foldl (+) 0 [1,2,3,4]
10

Haskell> take 4 (iterate (+ 1) 1)
[1,2,3,4]

Haskell> scanl (+) 0 [1,2,3,4]
[0,1,3,6,10]

Haskell>
```

Scala

```
scala> List(1,2,3,4).foldLeft(0)(_+_)
val res0: Int = 10

scala> Stream.iterate(1)(_ + 1).take(4).toList
val res1: List[Int] = List(1, 2, 3, 4)

scala> List(1,2,3,4).scanLeft(0)(_+_)
val res2: List[Int] = List(0, 1, 3, 6, 10)

scala>
```