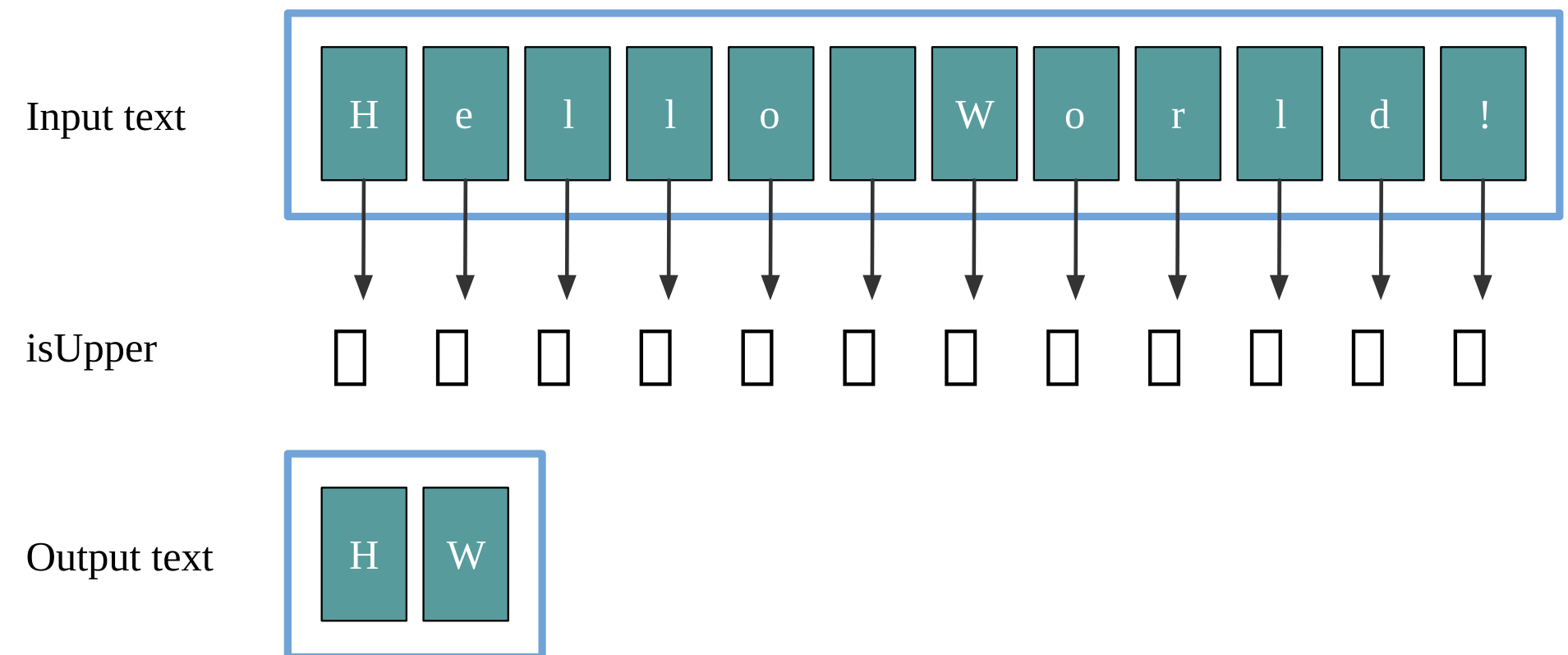# FUNCTION AS INPUTS

Function as inputs

```scala
def filter(
  text     : String,
  predicate: Char => Boolean
): String = ...
```

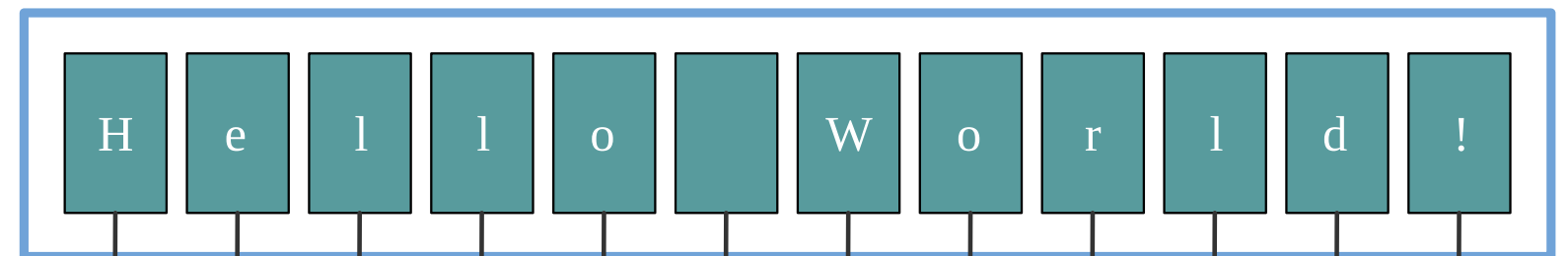# Function as inputs

```
filter(
  "Hello World!",
  (c: Char) => c.isUpper
)
// res0: String = "HW"
```
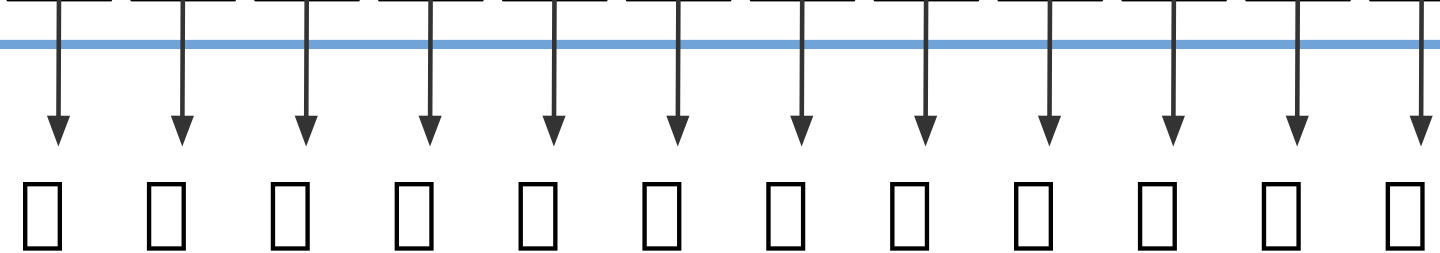
Input text

H	e	l	l	o		W	o	r	l	d	!

isUpper

| ⬜ | ⬜ | ⬜ | ⬜ | ⬜ | ⬜ | ⬜ | ⬜ | ⬜ | ⬜ | ⬜ | ⬜ |

Output text

H	W

Function as inputs

```
filter(
  "Hello World!",
  (c: Char) => c.isLetter
)
// res1: String = "HelloWorld"
```

Input text

| H | e | l | l | o | | W | o | r | l | d | ! |

isLetter

Output text

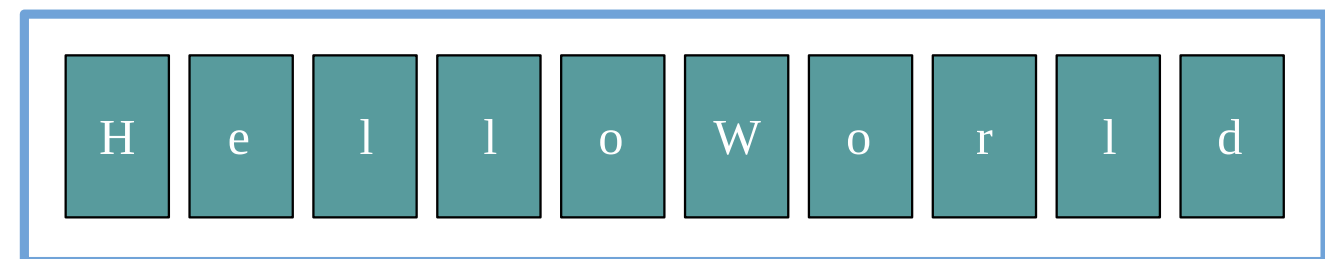| H | e | l | l | o | W | o | r | l | d |

Reduce code duplication

```scala
def upperCase(text: String): String = {
  val characters = text.toArray
  for (i <- 0 until text.length) {
    characters(i) = characters(i).toUpper
  }
  new String(characters)
}
```

```scala
def lowerCase(text: String): String = {
  val characters = text.toArray
  for (i <- 0 until text.length) {
    characters(i) = characters(i).toLower
  }
  new String(characters)
}
```

```scala
upperCase("Hello")
// res2: String = "HELLO"
```

```scala
lowerCase("Hello")
// res3: String = "hello"
```
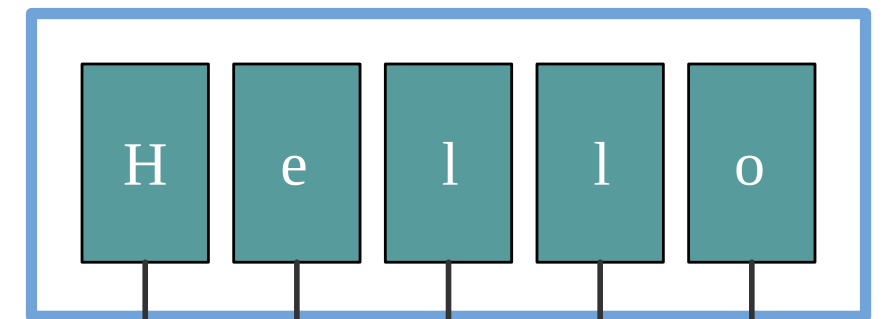
Capture pattern

```scala
def map(text: String, update: Char => Char): String =
  val characters = text.toArray
  for (i <- 0 until text.length) {
    characters(i) = update(characters(i))
  }
  new String(characters)
}
```
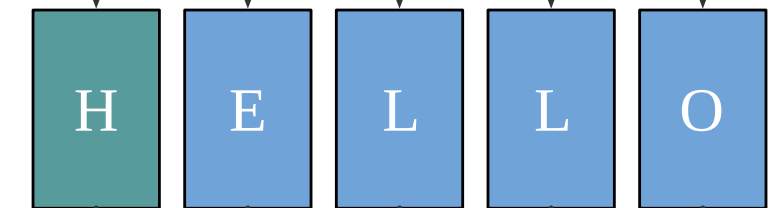
```scala
def upperCase(text: String): String =
  map(text, c => c.toUpper)

def lowerCase(text: String): String =
  map(text, c => c.toLower)
```

Input text
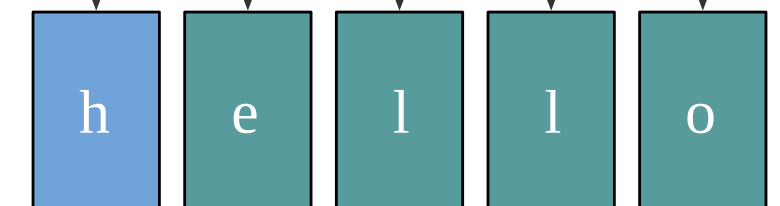
| H | e | l | l | o |

toUpper

| H | E | L | L | O |

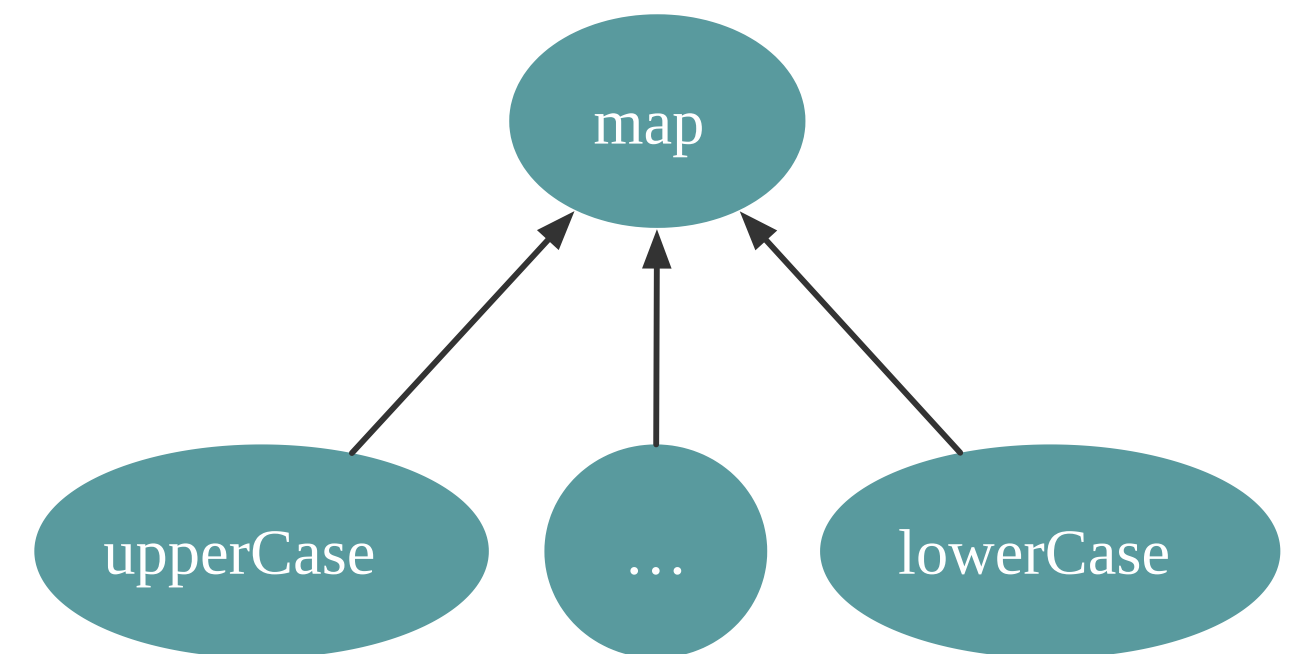toLower

| h | e | l | l | o |

Capture pattern

```scala
def map(text: String, update: Char => Char): String =
  val characters = text.toArray
  for (i <- 0 until text.length) {
    characters(i) = update(characters(i))
  }
  new String(characters)
}
```
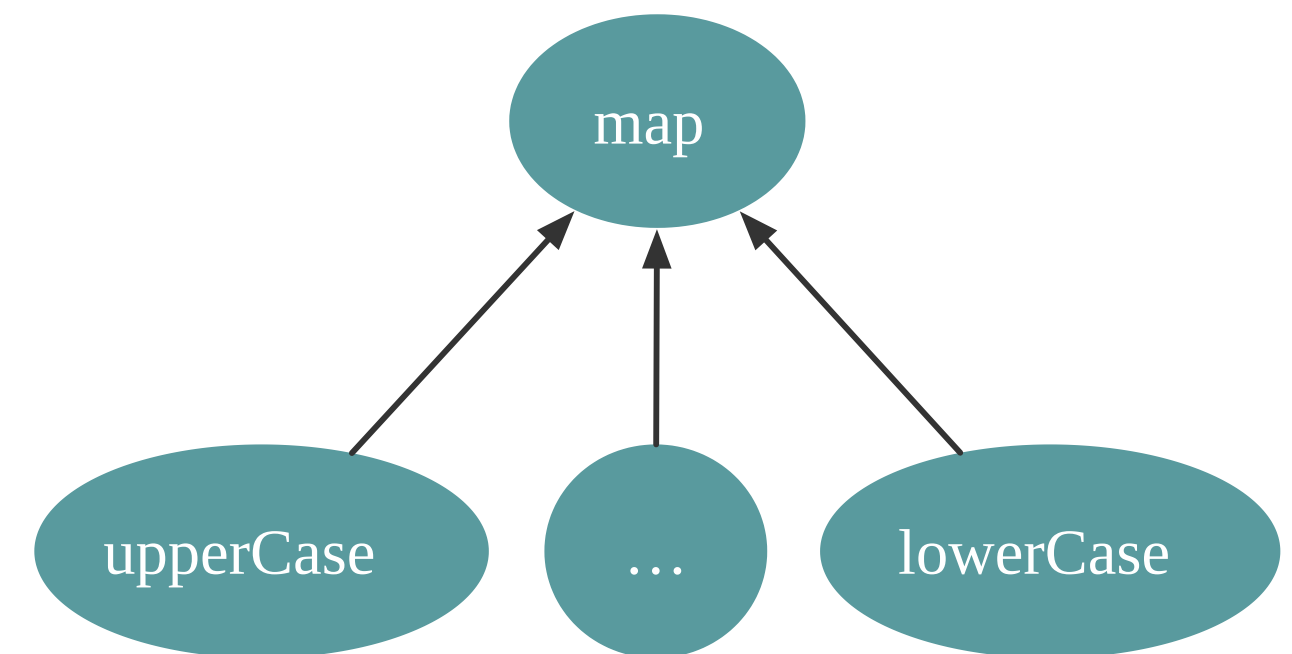
```scala
def upperCase(text: String): String =
  map(text, c => c.toUpper)

def lowerCase(text: String): String =
  map(text, c => c.toLower)
```
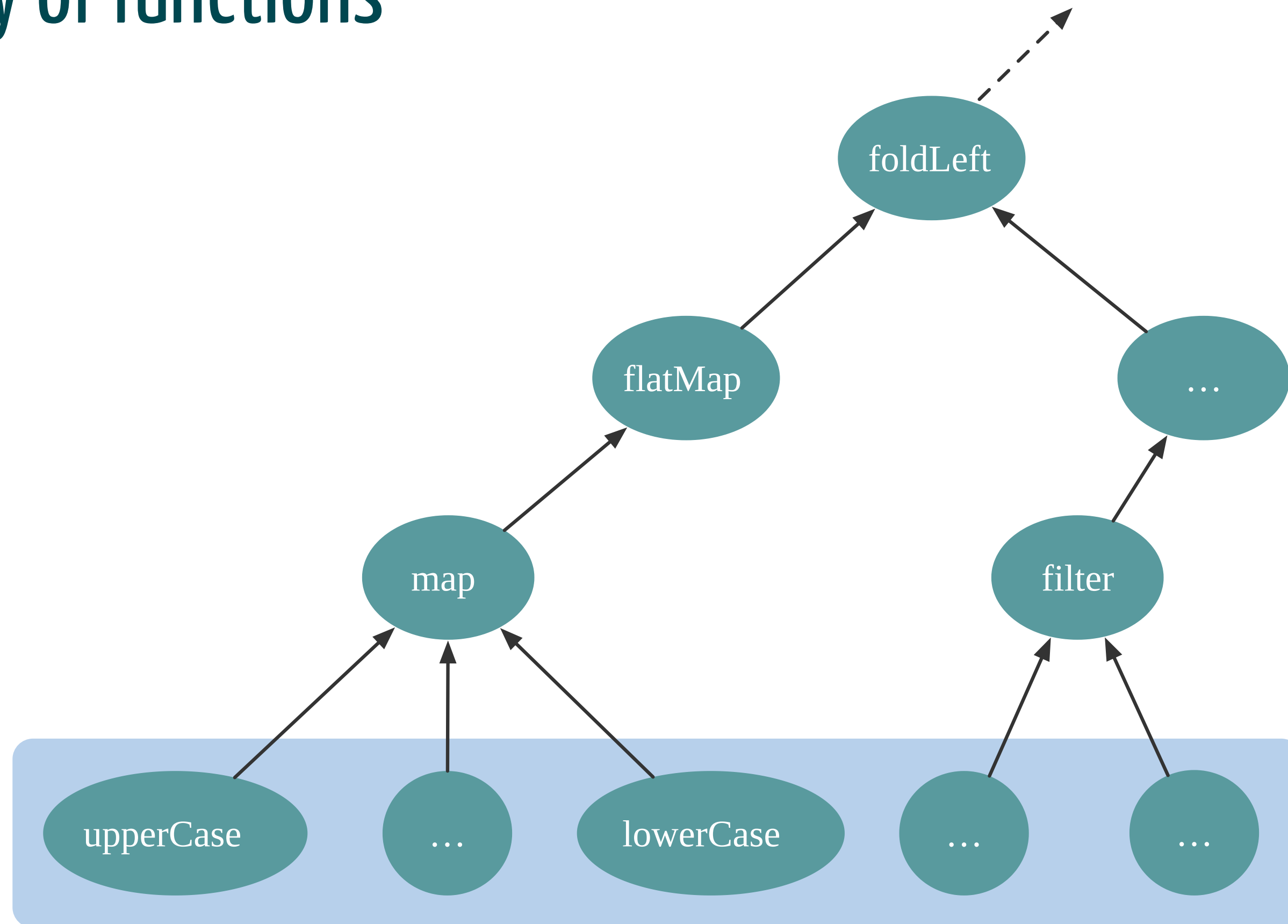
Property based testing

```
test("map does not modify the size of a text") {
  forAll((
    text  : String,
    update: Char => Char
  ) =>
    val outputText = map(text, update)
    outputText.length == text.length
  )
}
```
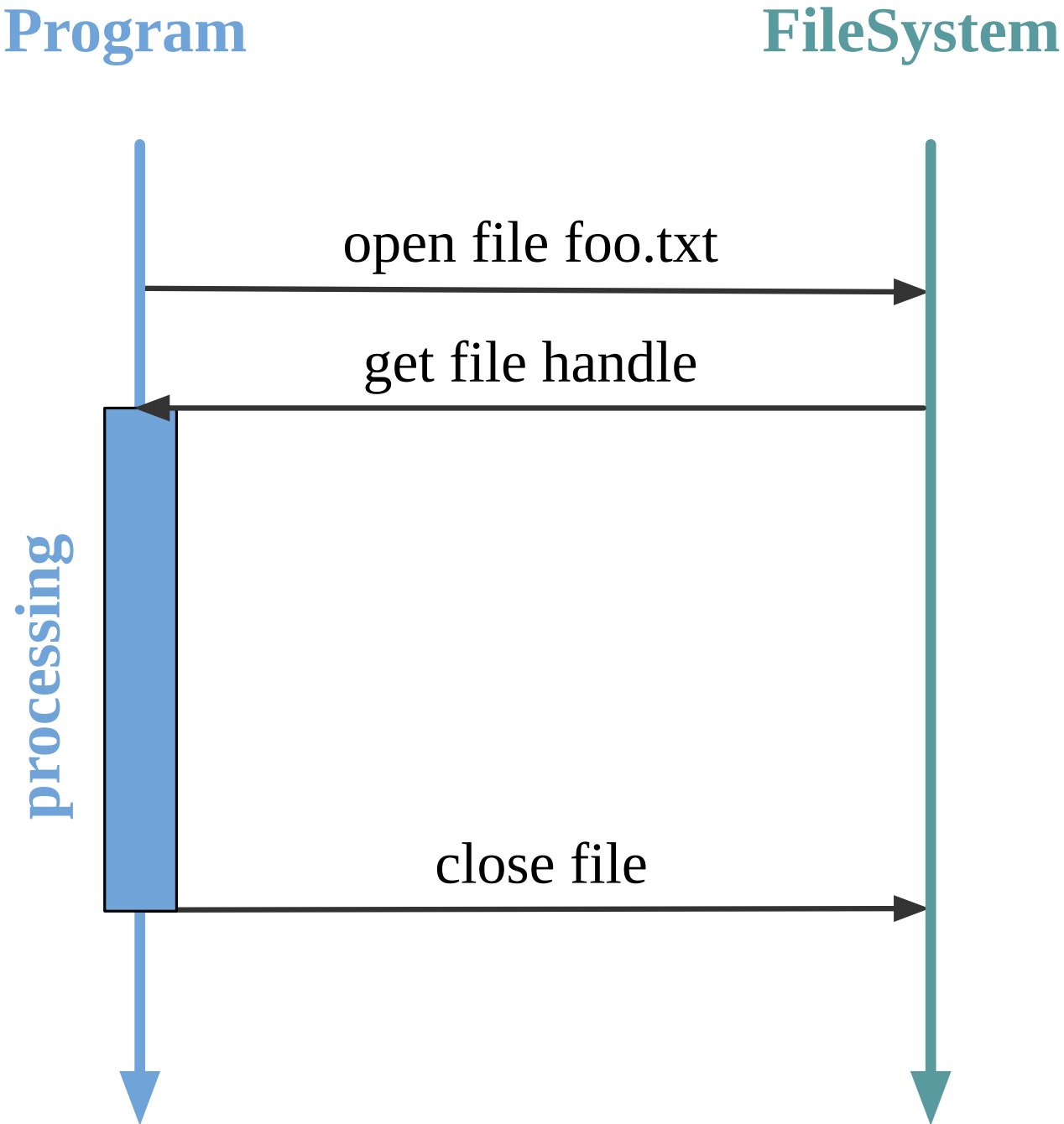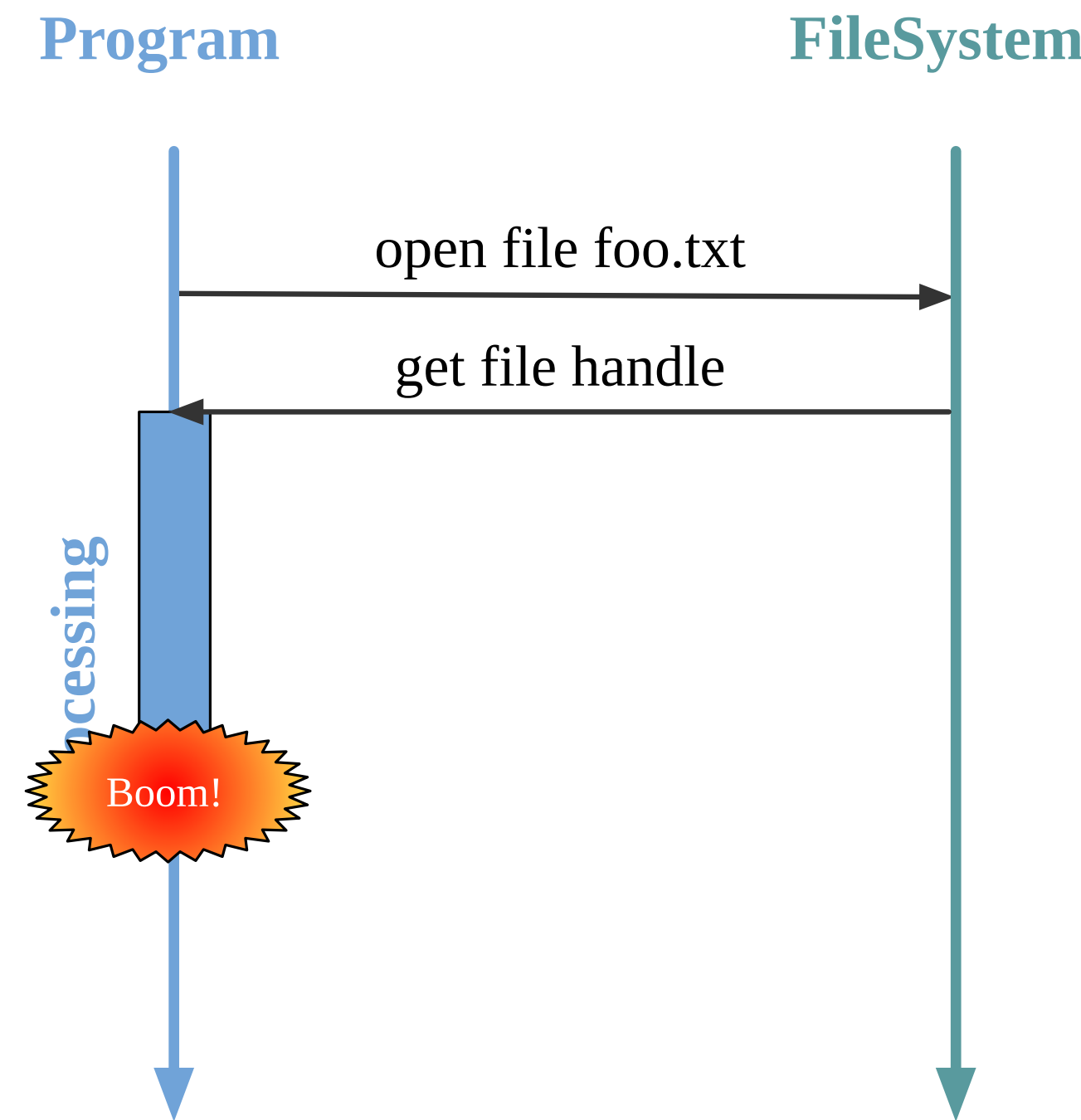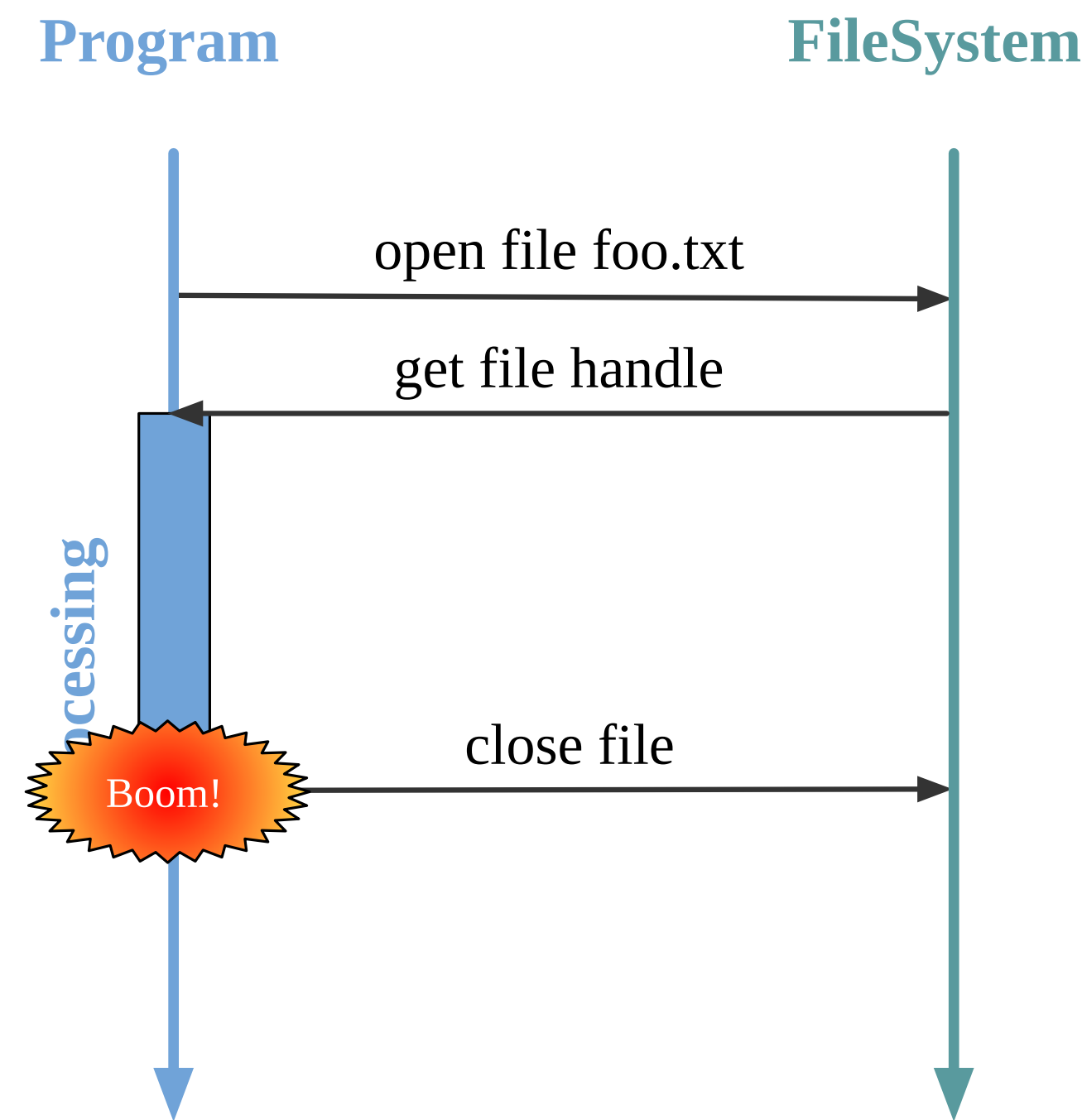
Hierarchy of functions

File processing

File processing

File processing

Write tricky code once

```scala
import scala.io.Source

def usingFile(fileName: String, processing: Iterator[String] => Int): Int = {
  val source = Source.fromResource(fileName)
  try {
    processing(source.getLines())
  } finally {
    source.close()
  }
}
```

Write tricky code once

```scala
import scala.io.Source

def usingFile(fileName: String, processing: Iterator[String] => Int): Int = {
  val source = Source.fromResource(fileName)
  try {
    processing(source.getLines())
  } finally {
    source.close()
  }
}
```

```scala
val countLines: Iterator[String] => Int =
  lines => lines.size
```

```scala
val countWords: Iterator[String] => Int =
  lines => ...
```

Write tricky code once

```scala
import scala.io.Source

def usingFile(fileName: String, processing: Iterator[String] => Int): Int = {
  val source = Source.fromResource(fileName)
  try {
    processing(source.getLines())
  } finally {
    source.close()
  }
}
```

```scala
usingFile("50-word-count.txt", countLines)
// res7: Int = 2
```

```scala
usingFile("50-word-count.txt", countWords)
// res8: Int = 50
```

Summary

- Higher order function

- Reduce code duplication

- Improve code quality