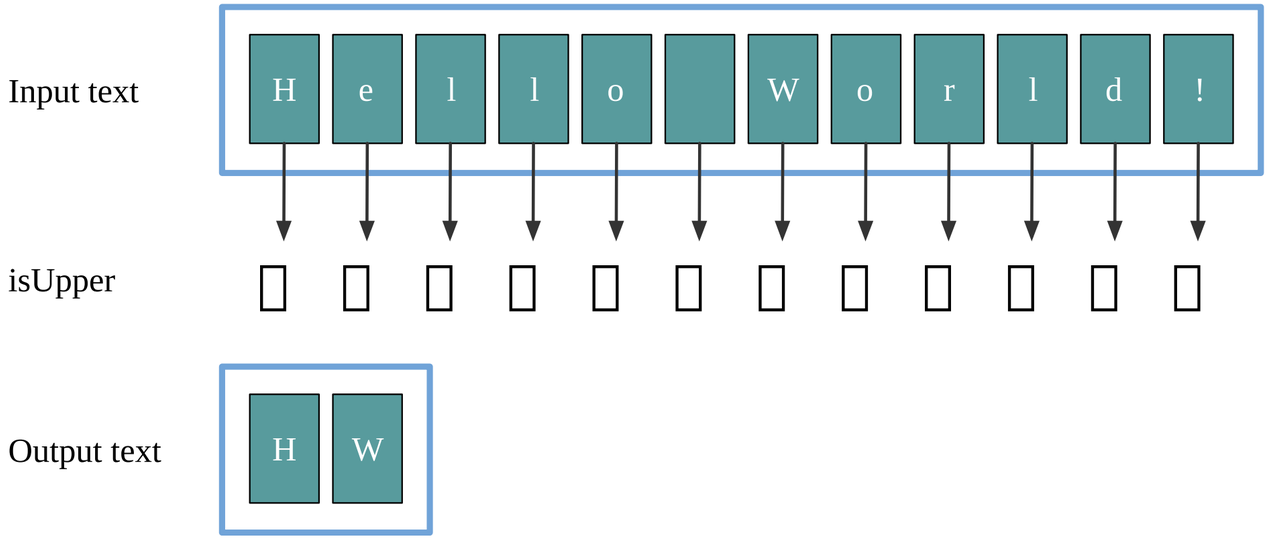
FUNCTION AS INPUTS

Function as inputs

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

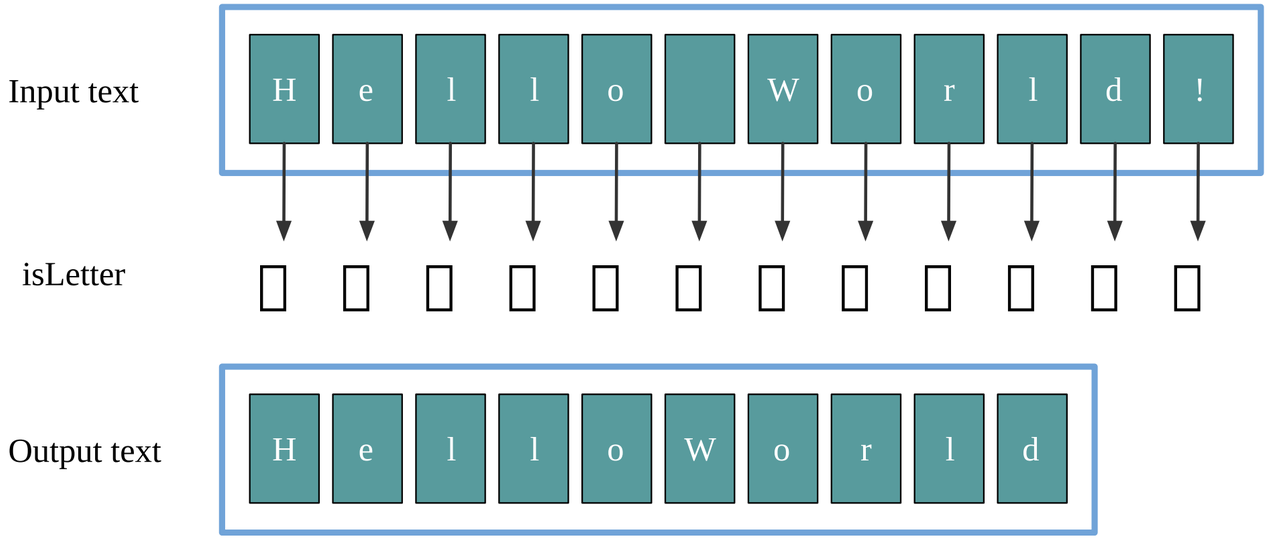
Function as inputs

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



Function as inputs

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



Reduce code duplication

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

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

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

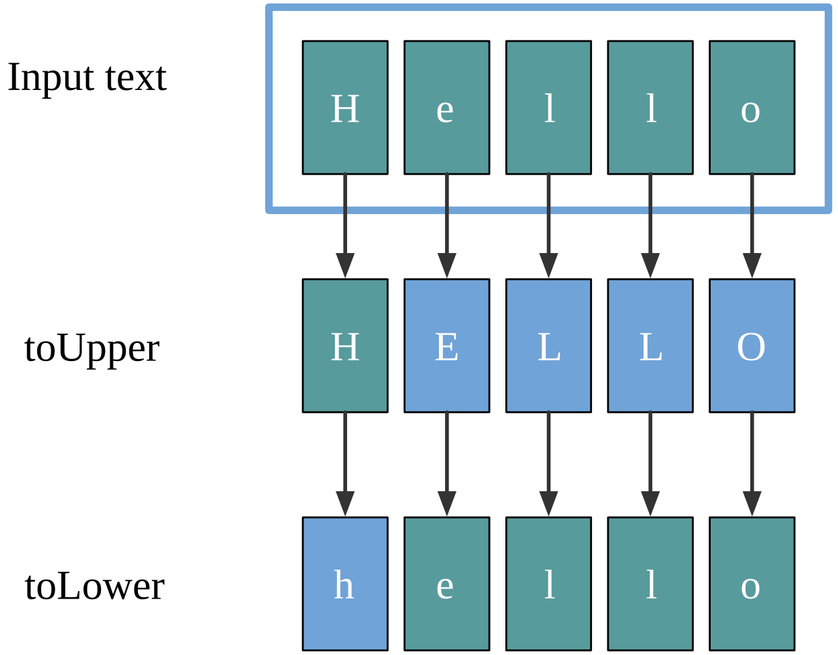
```
lowerCase("Hello")
// res3: String = "hello"
```

Capture pattern

```
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)
}
```

```
def upperCase(text: String): String =
  map(text, c => c.toUpper)

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

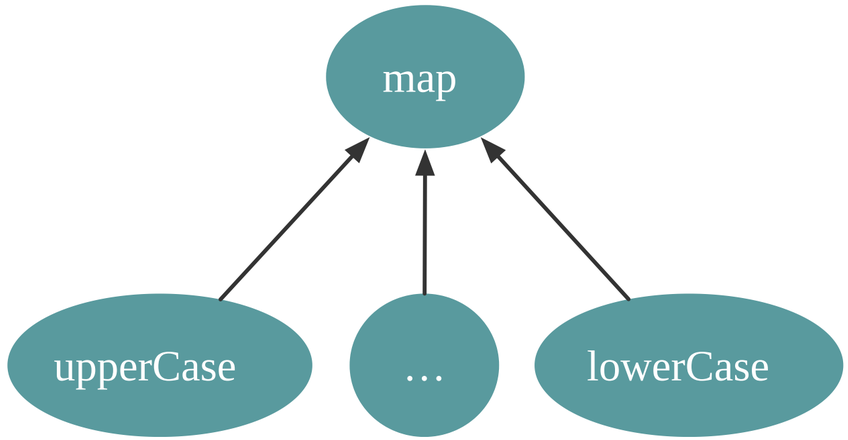


Capture pattern

```
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)
}
```

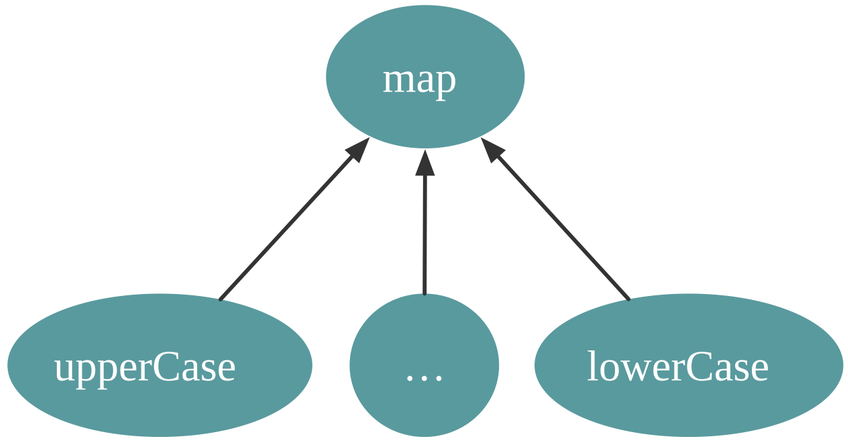
```
def upperCase(text: String): String =
  map(text, c => c.toUpper)

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

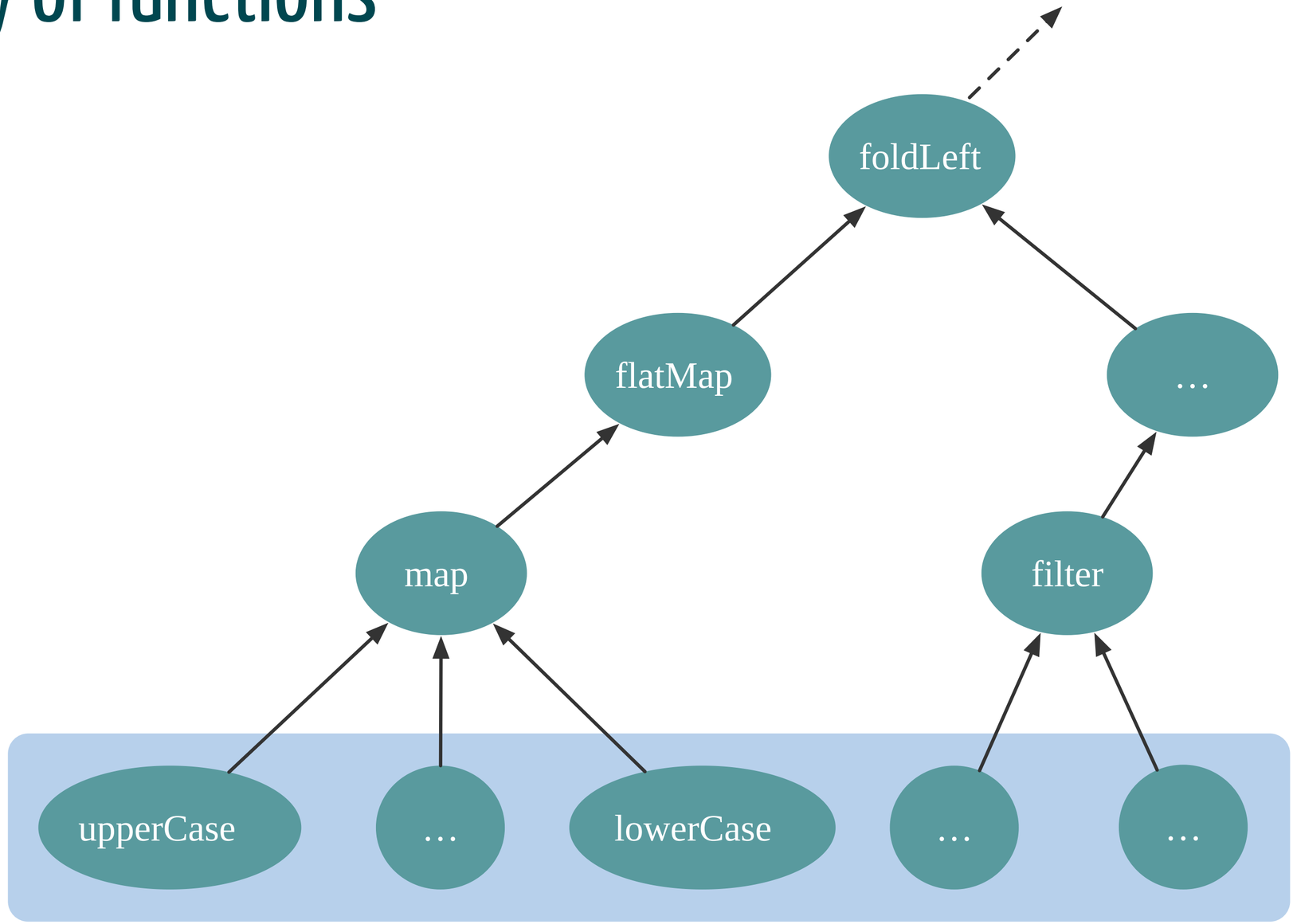


Property based testing

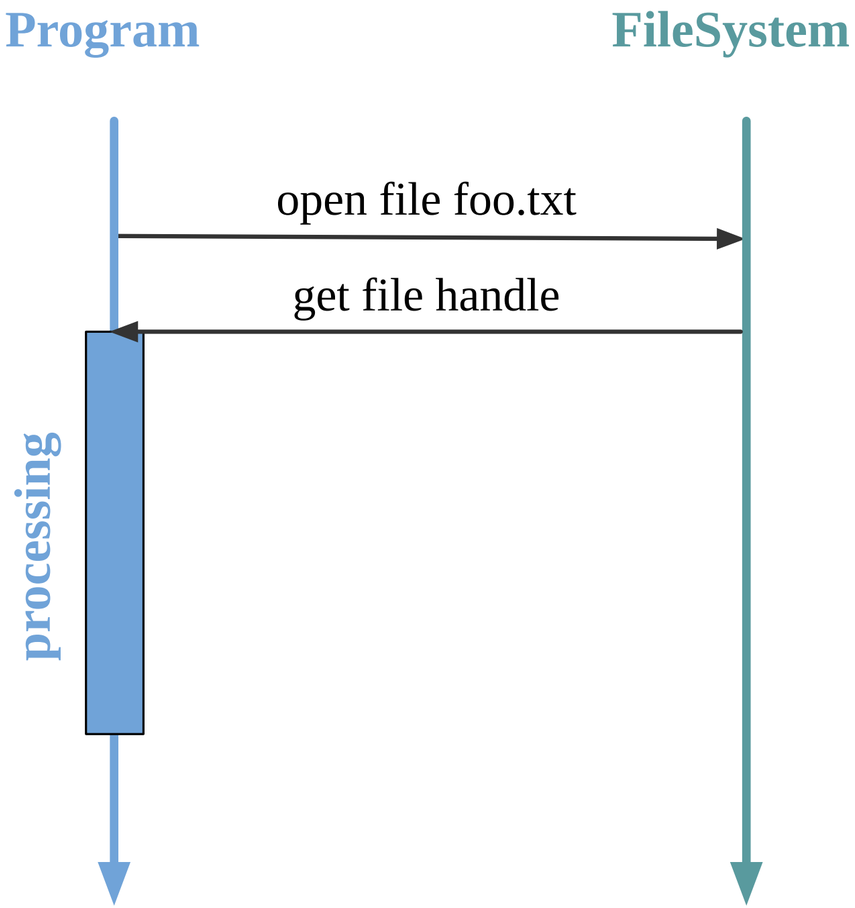
```
test("map does not modify the size of a text") {
  forAll((
    text  : String,
    update: Char => Char
  ) =>
    val outputText = map(text, update)
    outputText.length == text.length
  )
}
```



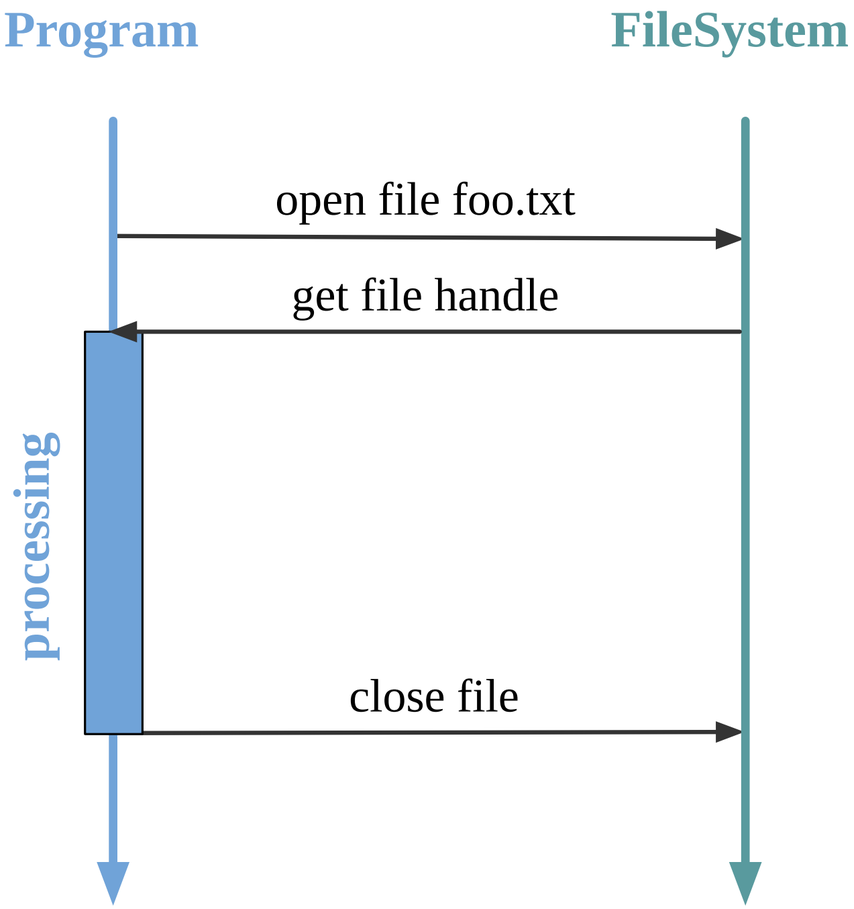
Hierarchy of functions



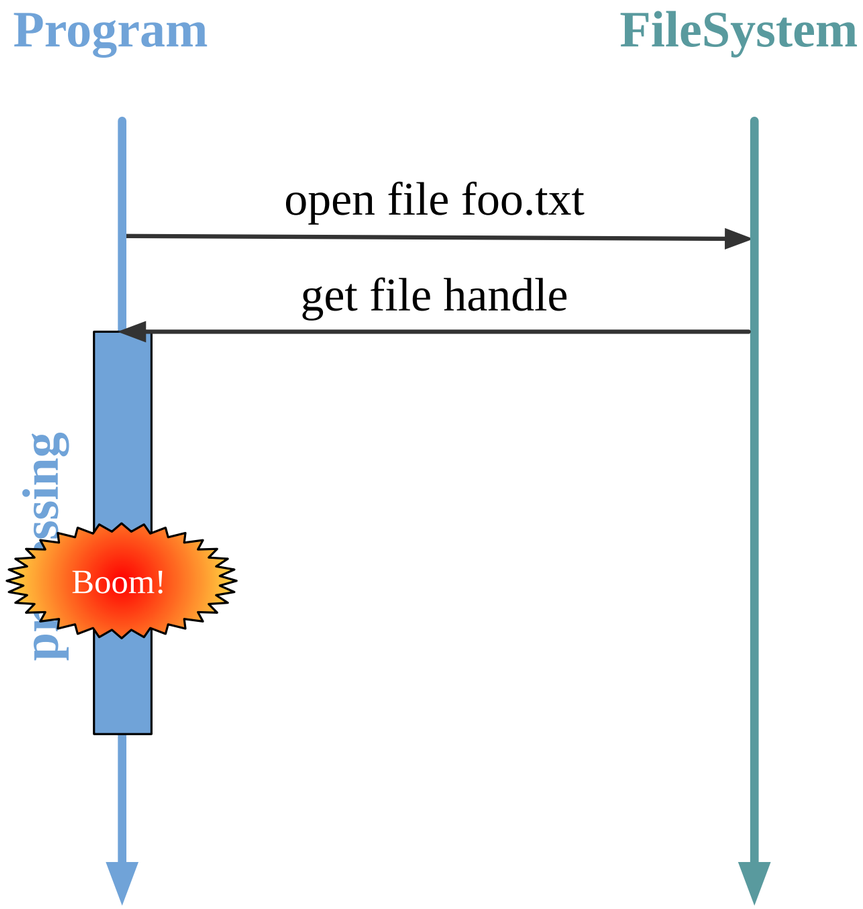
File processing



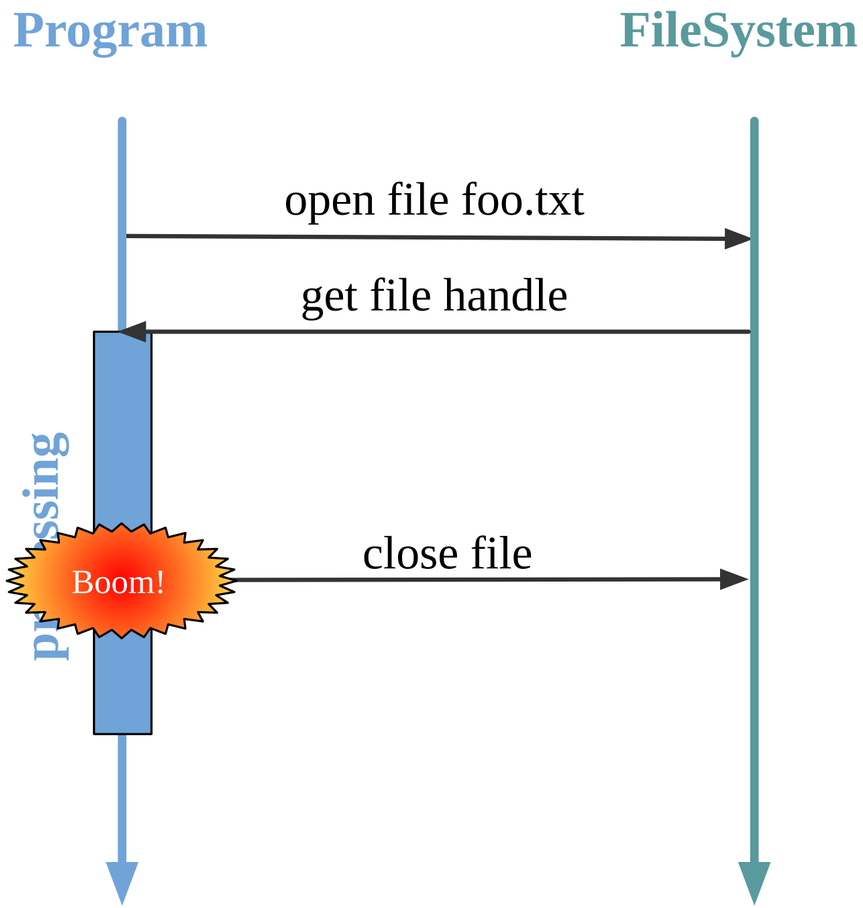
File processing



File processing



File processing



Write tricky code once

```
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

```
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()
  }
}
```

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

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

Write tricky code once

```
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()
  }
}
```

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

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

Summary

- Higher order function
- Reduce code duplication
- Improve code quality