scalafmt: opinionated code formatter for Scala

Ólafur Páll Geirsson

École Polytechnique Fédérale de Lausanne
School of Computer and Communication Sciences

July 6, 2016

Today's agenda

1. Introduction
2. Background
3. scalafmt
4. Results
5. Conclusion

Overview

What is code formatting?

Unformatted

```
object     MyApp
  extends App {
  Initialize  ( context, config(port(
    "port.http"),
   settings + custom))
  Loop(  )
}
```

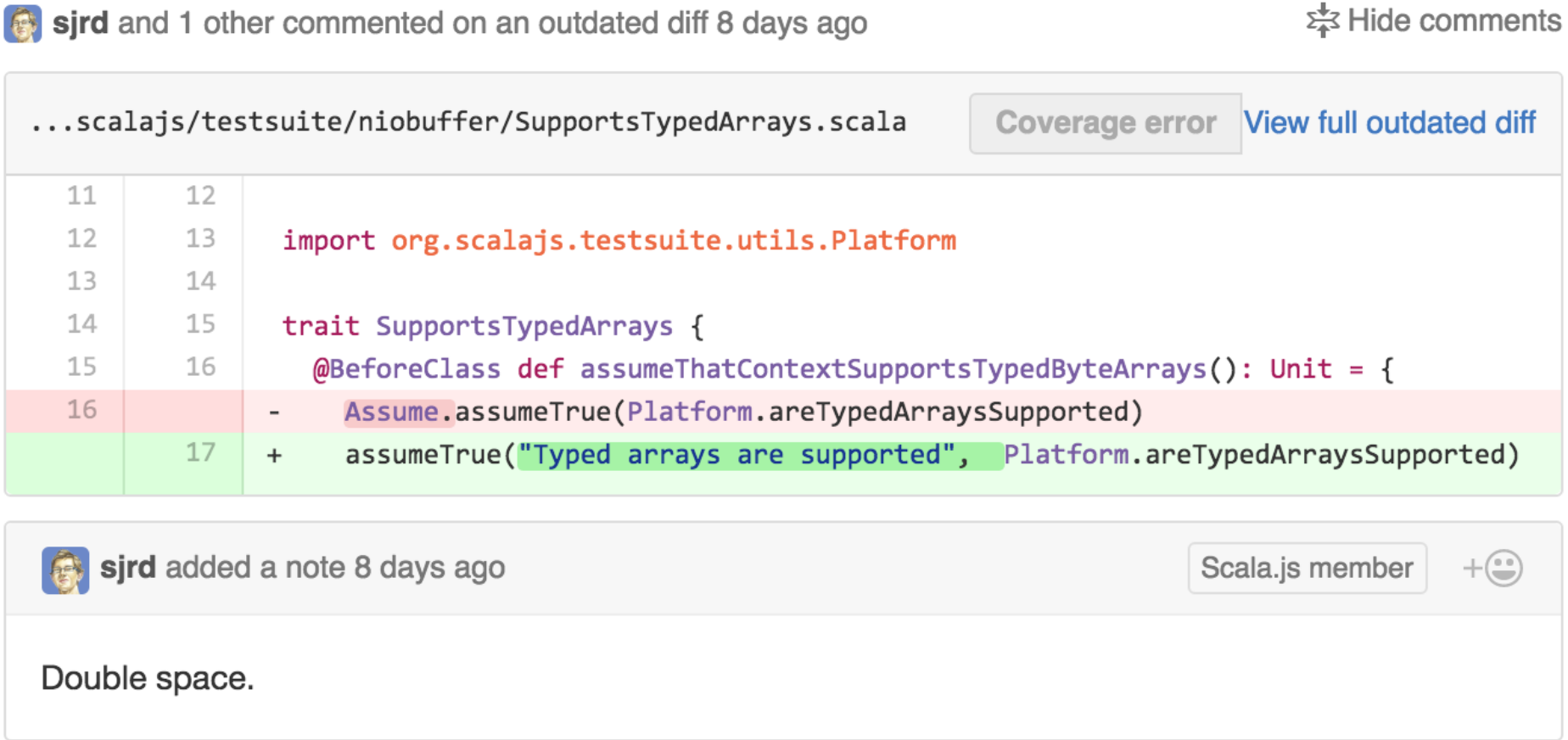
What is code formatting?

Formatted

```
object MyApp extends App {
  Initialize(
      context,
      config(port("port.http"),
             settings + custom))
  Loop()
}
```

Why?

Reason 1: Collaborative environments



Reason 2: Refactoring

Large-Scale Automated Refactoring Using ClangMR[1]

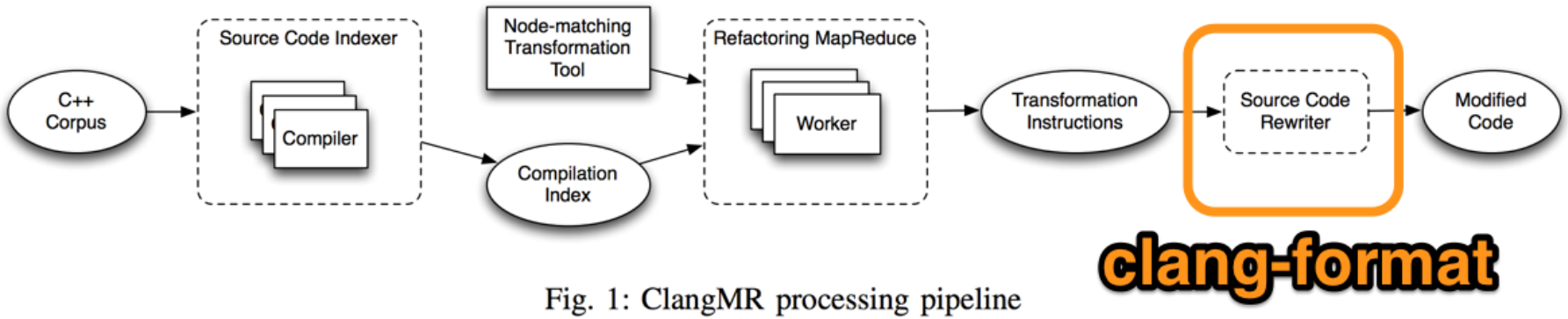


Fig. 1: ClangMR processing pipeline

[1] Wright et al. 2013.

Problem statement

What *algorithms* and *data structures* allow us to develop a Scala code formatter with the following features?

- *Maximum line length setting*
- *Opinionated setting*
- *Vertical alignment*
- *Good performance*

Maximum line length setting

```
// 40 character max line length          |
object MyApp extends App {
  // BAD
  Initialize(context, config(port("port.http"),
    settings + custom))

  // OK
  Initialize(
      context,
      config(port("port.http"),
             settings + custom))
}
```

Opinionated setting

My definition

Disregard line breaking decisions in the original source to ensure that formatted sources follow a uniform coding style.

```
// Bin-pack
class Point(val x: Int, val y: Int,
    val z: Int)

// No bin-pack
class Point(val x: Int,
            val y: Int,
            val z: Int)
```

Vertical alignment

```
object VerticalAlignment {
  x match {
    case 1  => 1  -> 2  // first
    case 11 => 11 -> 22 // second
  }

  def name   = column[String]("name")
  def status = column[Int]("status")

  libraryDependencies ++= Seq(
    "org.scala-lang" % "scala-compiler" % "2.11.7",
    "com.lihaoyi"    %% "sourcecode"    % "0.1.1"
  )
}
```

Performance

- IDEs: reformat file on save
- Build tools: reformat file on compile
- Continuous integration: reformat diff before code review

Overview

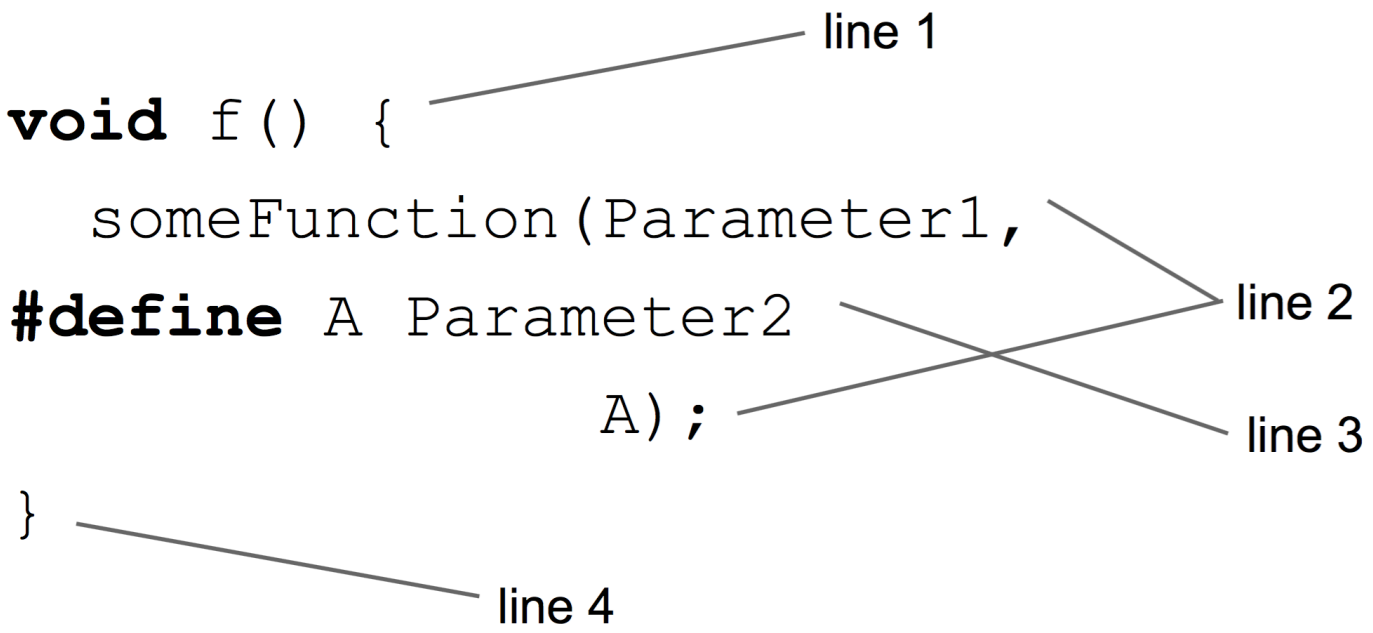
Scalariform

- No maximum line length setting
- No opinionated setting

ClangFormat

Parser

- Custom *UnwrappedLine* parser for C, C++, Objective-C, Java, JavaScript and Protobuf[2]
  - handles invalid code code
  - ∼4.000 LOC



[2] **jasper_clang-format_2014**.

Line breaking: shortest path search

- Dijkstra's shortest path for optimal line breaking.[3]
  - Non-whitespace tokens are nodes
  - Whitespace tokens are edges

```
aaaaaaaa(aaaaaaaaaaaaa, aaaaaaaaaaaaaaa(aaaaaaaaaaaaaaaaaaaaaaaaaaaaa(   Penalty: 100
                              aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa)),   Penalty:  41
         aaaaaaaa(aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa(                        Penalty: 100
             aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa)));   Total:   241
```

[3] **jasper_clang-format_2014**.

rfmt

Formatting algebra

- Three layout operators

Lorem ipsum dolor	$'txt'$
Lorem ipsum dolor consectetur adipiscing elit	$l_1 \updownarrow l_2$
Lorem ipsum dolor consectetur adipiscing elit Aliquam erat volutpat condimentum vitae leo sit	$l_1 \leftrightarrow l_2$

- one *choice* operator "?"

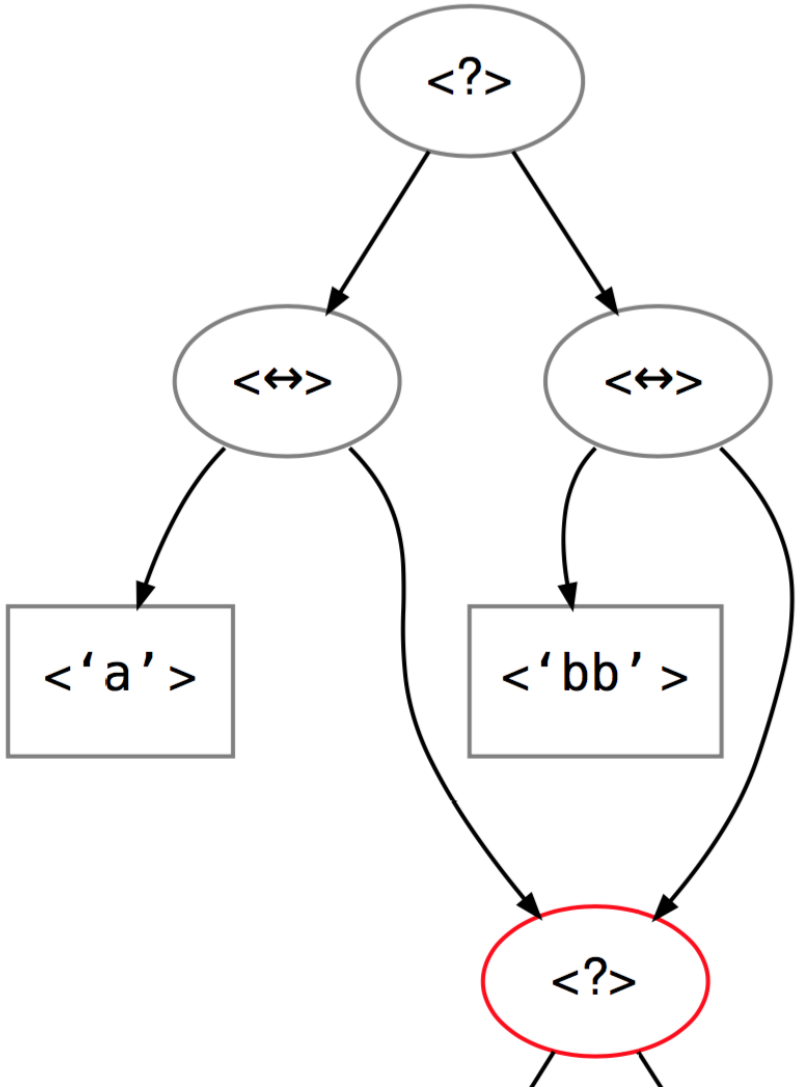
Translating R to formatting algebra

- Custom R parser
  - ∼1.000 LOC
  - Comments are AST nodes
- "Block language" implemented in terms of primitive combinators

ChoiceBlock(
  LineBlock(LineBlock(TextBlock(f), TextBlock('('))),
    WrapBlock($a_1, \ldots, a_m$),
    TextBlock(')'),
  StackBlock(LineBlock(TextBlock(f), TextBlock('('))),
    IndentBlock(4, WrapBlock($a_1, \ldots, a_m$)),
    TextBlock(')')).

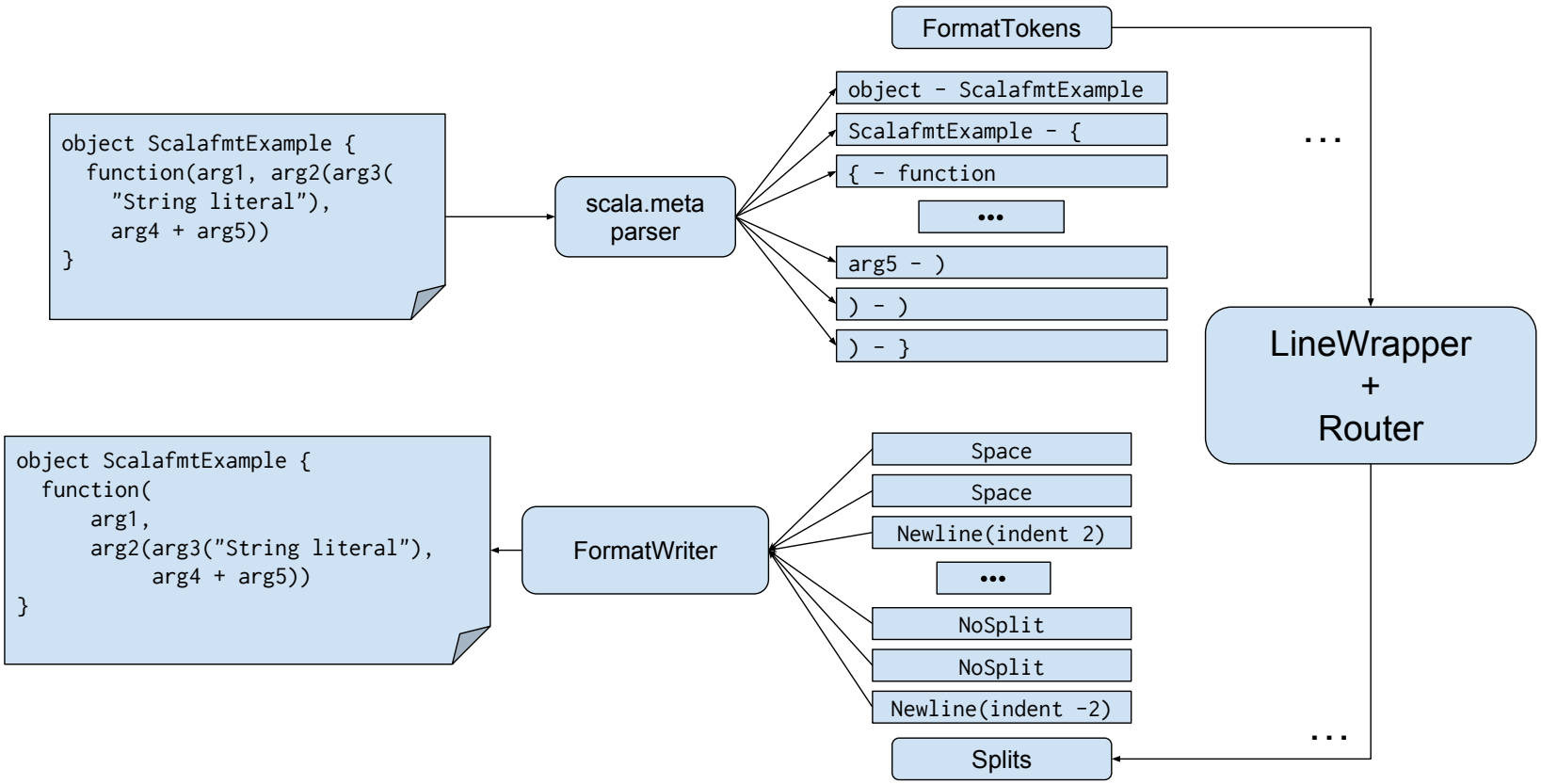
Line breaking: dynamic programming

- Dynamic programming to find optimal line breaking
  - (AST node, column) pairs are keys
  - can extrapolate missing columns

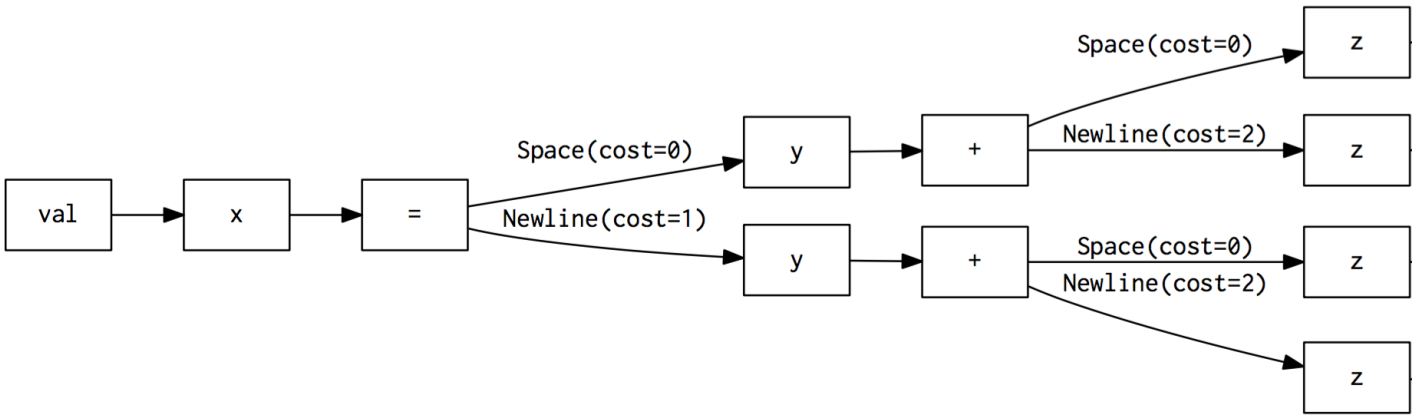


Overview

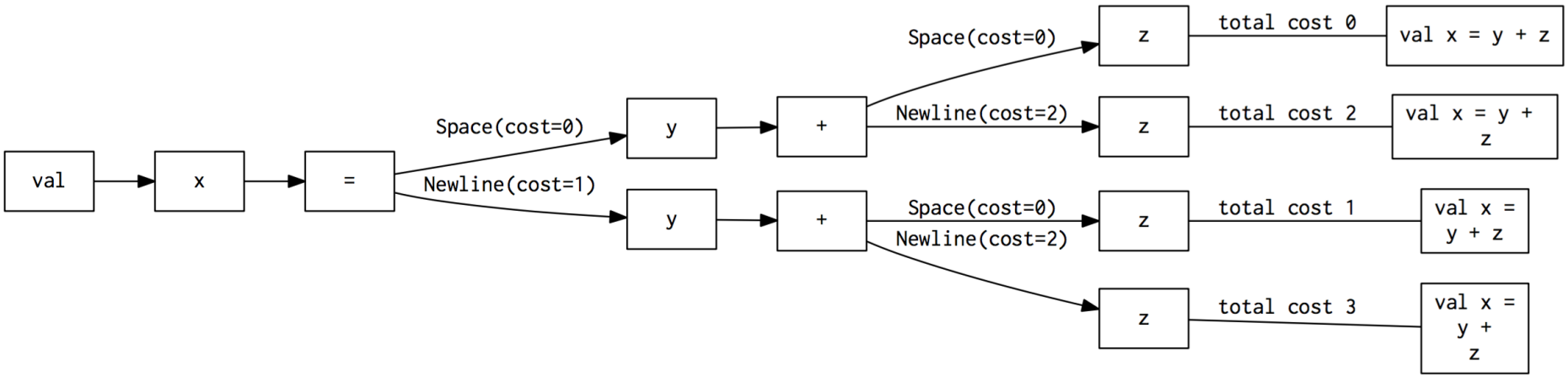
Architecture



LineWrapper + Router



LineWrapper + Router



Plain best-first search

- Small: $\sim$ 20 LOC
- Exponential running time for basic programs

Optimization 1: dequeueOnNewStatements

```
def x = {
  function1(argument1, argument2, argument3)
  function2(argument1, argument2, argument3)
}
```

Optimization 2: OptimalToken

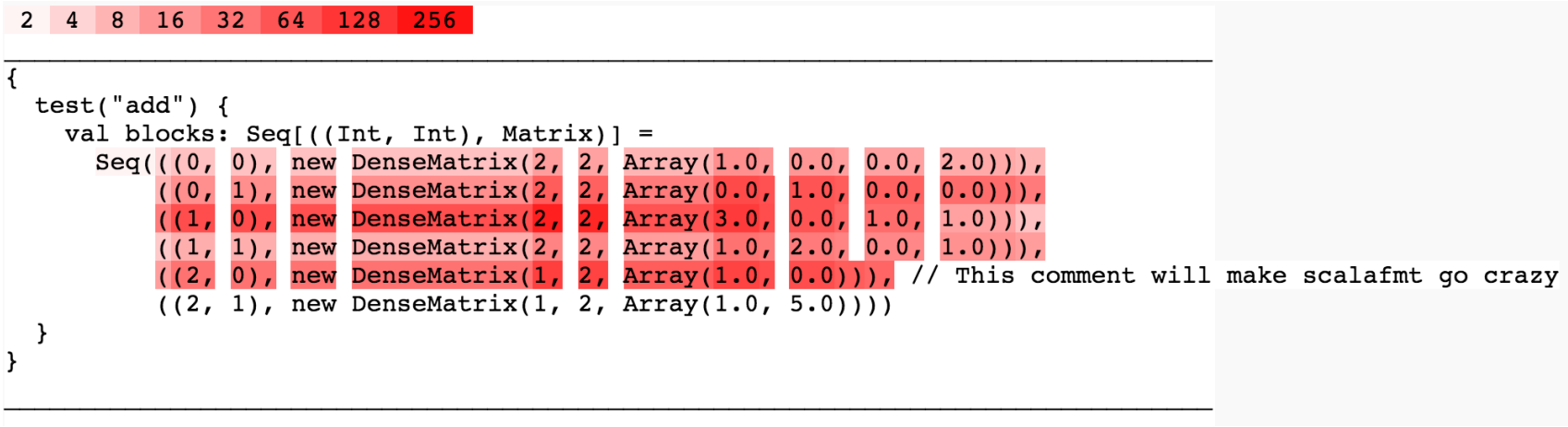
```
Database(UserObject(name1, age1),
         UserObject(name2, age2),
         // ...
         UserObject(nameN, ageN))
```

Optimization 3: escapeInPathologicalCases

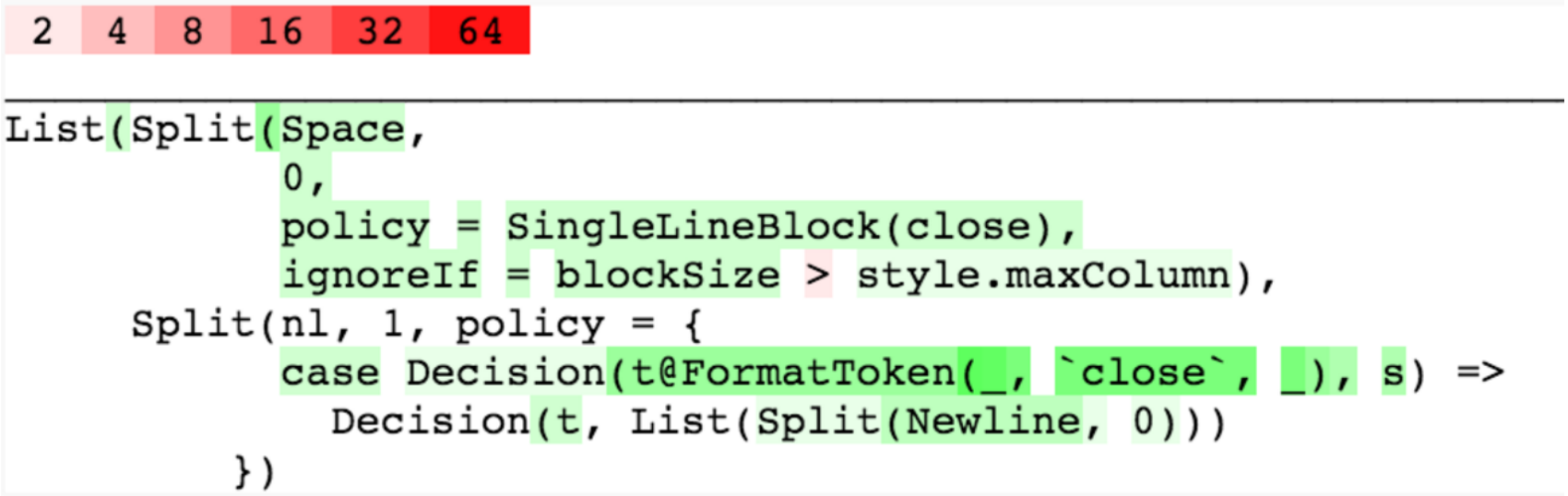
```
Defn.Object(Nil, Term.Name("State"), Template(Nil,
    Seq(Ctor.Ref.Name("Logger")), Term.Param(Nil,
    Name.Anonymous(), None, None),
    Some(Seq(Defn.Val(Nil,
    Seq(Pat.Var.Term(Term.Name("start")))), None,
    Term.Apply(Term.Name("State"), Seq()))),
    Defn.Def( /* ... */))))
```

Tooling

Heatmap


2 4 8 16 32 64 128 256
{
test("add") {
val blocks: Seq[((Int, Int), Matrix)] =
Seq(((0, 0), new DenseMatrix(2, 2, Array(1.0, 0.0, 0.0, 2.0))),
((0, 1), new DenseMatrix(2, 2, Array(0.0, 1.0, 0.0, 0.0))),
((1, 0), new DenseMatrix(2, 2, Array(3.0, 0.0, 1.0, 1.0))),
((1, 1), new DenseMatrix(2, 2, Array(1.0, 2.0, 0.0, 1.0))),
((2, 0), new DenseMatrix(1, 2, Array(1.0, 0.0))), // This comment will make scalafmt go crazy
((2, 1), new DenseMatrix(1, 2, Array(1.0, 5.0))))
}
}

Diff heatmap



Property 1: can format

```
forAll { code =>
  whenever(scalaCompilerCanParse(code)) {
    format(code).isInstanceOf[Success]
  }
}
```

Property 2: can format

```
forAll { code =>
  ast(code) == ast(format(code))
}
```

Property 3: idempotent

```
forAll { code =>
  format(code) == format(format(code))
}
```

Overview

Verizon

“Verizon is now including scalafmt (with reformat on compile settings) in the default template for all new projects (which, in a sizable microservices shop, is a lot of projects)”

— Daniel Spiewak

Overview

Conclusion

Scalafmt

- ???

The End

Thank you!

References

Wright, Hyrum et al. (2013). “Large-Scale Automated Refactoring Using ClangMR”. In: URL: https://research.google.com/pubs/pub41342.html (visited on 04/21/2016).