SCALA

WHAT IS IT

AND WHAT IS IT USED FOR

INTRODUCTION

SCALA

- GENERAL-PURPOSE PROGRAMMING LANGUGAGE
- MULTI-PARADIGM (OBJECT ORIENTED + FUNCTIONAL)
- WORKS (ORIGINALLY) ON JAVA VIRTUAL MACHINE (JVM)

SCALABLE LANGUAGE

- THE NAME *SCALA* COMES FROM *SCALABLE*
- SCALA COMBINES *CONCISE SYNTAX* WITH *SCALABILITY*

USAGE

- Data engineering
- Distributed systems
- Concurrency, Parallel processing
- Web development

BRIEF HISTORY

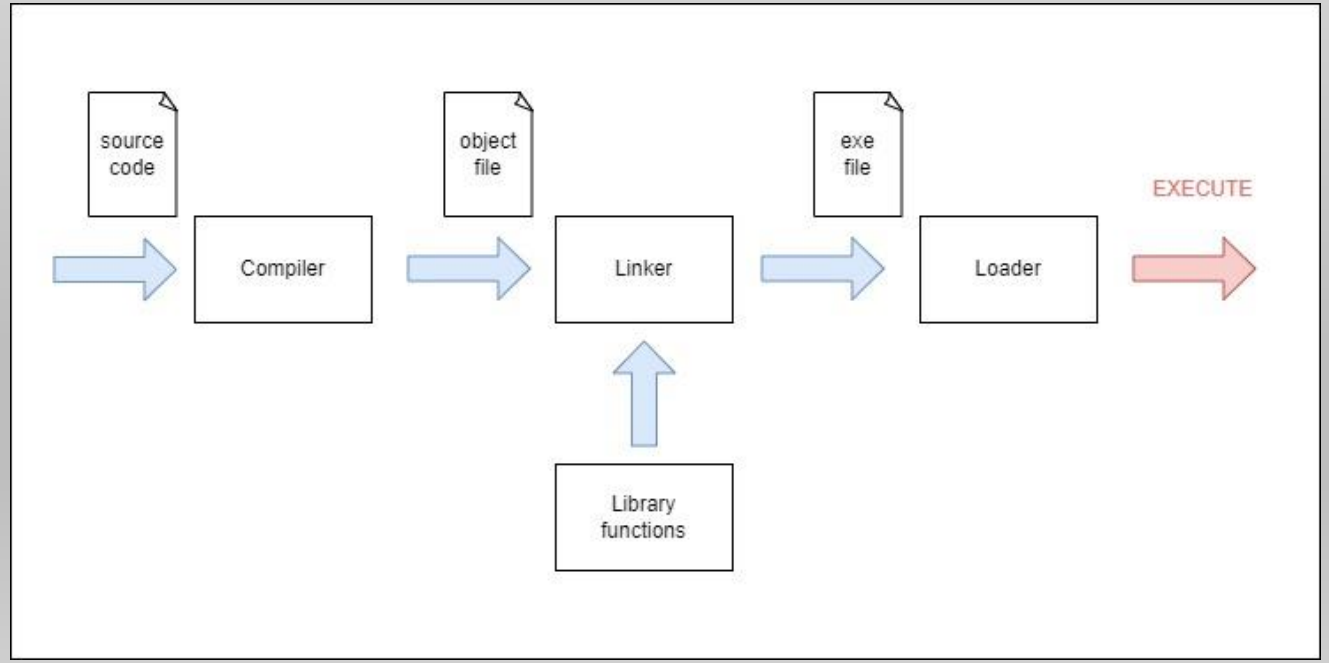
- Created and developed by Martin Odersky
- 2001 – Martin started working on Scala at the Ecole Polytechnique Federale de Lausanne (EPFL)
- 2004 – Scala officially released
- 2006 – Scala 2.0
- 2012 – started working on Scala 3
- 2021 – Scala 3

HOW IS PROGRAM EXECUTED?

HOW C CODE RUNS

In general:

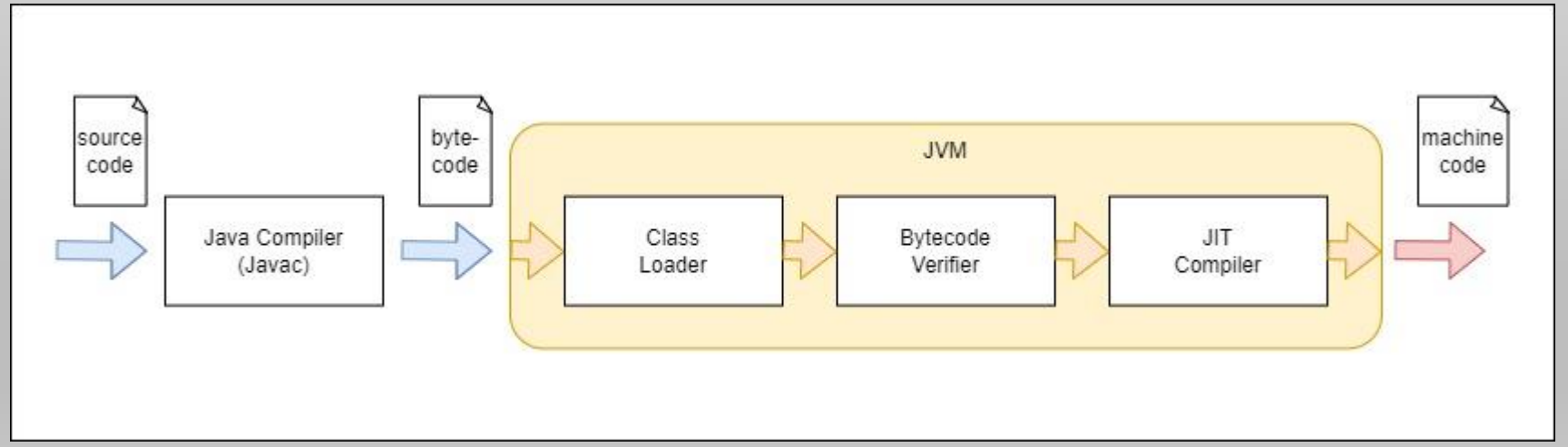
Source code → Compilation → Machine code → run



HOW JAVA CODE RUNS

In general:

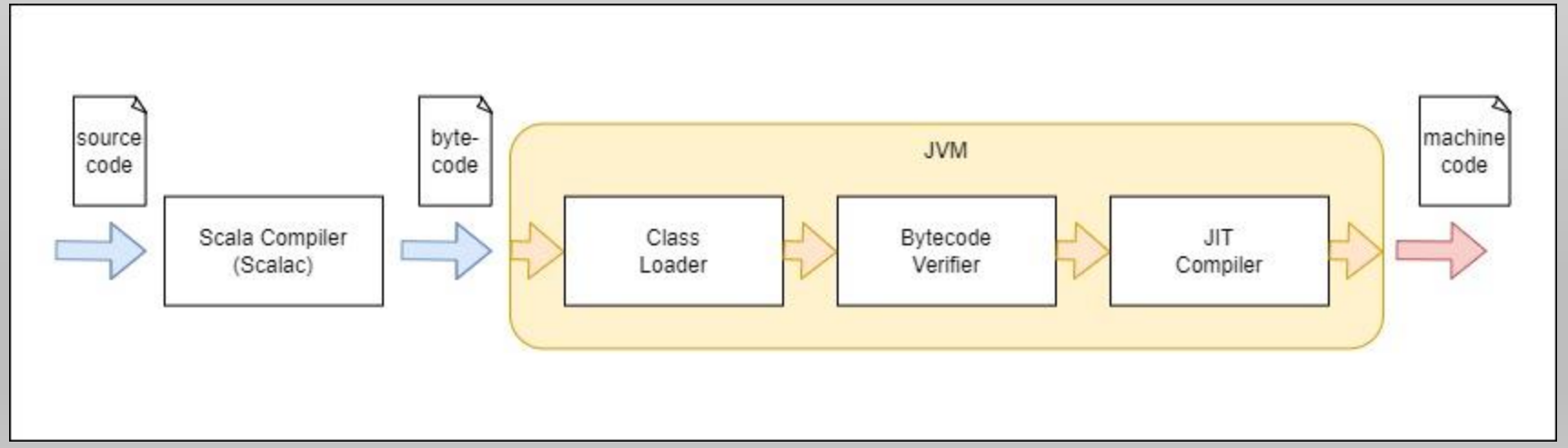
Source code → Compilation → Bytecode → JVM → run



HOW SCALA CODE RUNS

In general:

Source code → Compilation → Bytecode → JVM → run



SO LET'S RUN IT!

HELLO WORLD

Scala 2

```scala
object hello {
  def main(args: Array[String]) = {
    println("Hello, World!")
  }
}
```

Scala 3

```scala
@main def hello() = println("Hello, World!")
```

We can put this code in the *hello.scala* file

HOW TO RUN SCALA CODE?

- FROM COMMAND LINE

SCALAC – COMPILE SCALA SOURCE CODE

SCALA – RUN COMPILED CODE

```
D:\Projekty>scalac hello.scala

D:\Projekty>scala hello
Hello, World!
```

HOW TO RUN SCALA CODE?

- FROM REPL

JUST TYPE *SCALA* IN COMMAND LINE

```
D:\Projekty>scala
Welcome to Scala 3.2.0 (1.8.0_292, Java OpenJDK 64-Bit Server VM).
Type in expressions for evaluation. Or try :help.

scala> val a: Int = 5
val a: Int = 5

scala> a*2 + 3
val res0: Int = 13
```

How to run Scala code?

- From ide

Popular ide's for scala:

- Scala ide for eclipse
- Intellij idea (with scala plugin)
- VSCode (with metals extension)

HOW TO RUN SCALA CODE?

- USING SBT …

SBT

- SBT (SCALA BUILD TOOL) IS A POPULAR BUILD TOOL FOR SCALA PROJECTS
- SBT ENABLES YOU TO CONVENIENTLY:
  - COMPILE
  - RUN
  - TEST
  - PUBLISH

  PROJECT OF ANY SIZE

SCALA
AND PROGRAMMING PARADIGMS

SCALA'S PARADIGM

- SCALA IS A **FUSION OF OBJECT-ORIENTED AND FUNCTIONAL PROGRAMMING**
- OBJECTS IN SCALA ARE USED FOR MODULARITY AND FUNCTIONS ARE USED FOR LOGIC
- SCALA IS DESIGNED SO THAT TO SHOW THAT FUSION OF OBJECT-ORIENTED AND FUNCTIONAL PROGRAMMING IS POSSIBLE AND PRACTICAL

WORTH READING – PROGRAMMING PARADIGMS

HTTPS://WWW.GEEKSFORGEEKS.ORG/INTRODUCTION-OF-PROGRAMMING-PARADIGMS/

BIRD'S EYE VIEW

STATICALLY TYPED BUT …

- SCALA IS A **STATICALLY-TYPED LANGUAGE** – VARIABLE'S TYPE HAS TO BE KNOWN AT COMPILE TIME
- THAT SAID, SCALA'S CODE OFTEN LOOKS LIKE DYNAMICALLY TYPED DUE TO **TYPE INFERENCE**

```
scala> val s = "some string"
val s: String = some string

scala> val n = 123
val n: Int = 123

scala> val fp = 123.5
val fp: Double = 123.5

scala> n / 4
val res0: Int = 30
```

WRITE IT LIKE FUNCTIONAL!

IMPERATIVE

```
import scala.collection.mutable.ListBuffer

def double(ints: List[Int]): List[Int] =
  val buffer = new ListBuffer[Int]()
  for i <- ints do
    buffer += i * 2
  buffer.toList

val oldNumbers = List(1, 2, 3)
val newNumbers = double(oldNumbers)
```

FUNCTIONAL

```
val newNumbers = oldNumbers.map(_ * 2)
```

(Example from Scala 3 – Book from docs.scala-lang.org)

IMMUTABILTY

PYTHON

```
>>> some_list = [1, 2, 3]
>>> some_list.append(4)
>>> some_list
[1, 2, 3, 4]
```

SCALA

```
scala> val someList = List(3, 2, 1)
val someList: List[Int] = List(3, 2, 1)

scala> 4 :: someList
val res0: List[Int] = List(4, 3, 2, 1)

scala> someList
val res1: List[Int] = List(3, 2, 1)
```

SYNTAX

CREATING VARIABLES – *VAR* AND *VAL*

val / var *variable_name*[: *variable_type*] = *variable_value*

- *VAL* CREATES *IMMUTABLE* VARIABLE (MEANING YOU CAN NOT REASSIGN IT)
- *VAR* CREATES *MUTABLE* VARIABLE (MEANING YOU CAN REASSIGN IT)
- IT IS RECOMMENDED TO CREATE ALWAYS *VAL* VARIABLES UNLESS YOU KNOW IT WILL HAVE TO CHANGE OVER TIME

```
scala> val fp: Float = 3.0
val fp: Float = 3.0

scala> var fpReassign: Float = 4.0
var fpReassign: Float = 4.0

scala> fp = 10.5
-- [E052] Type Error: ----------------------------------------------
1 |fp = 10.5
  |^^^^^^^^^
  |Reassignment to val fp
  |
  | longer explanation available when compiling with `-explain`
1 error found

scala> fpReassign = 10.5
fpReassign: Float = 10.5
```

VAL = *IMMUTABLE* – REALLY??

```
scala> import scala.collection.mutable.ListBuffer

scala> val someBuffer = new ListBuffer[Int]()
val someBuffer: scala.collection.mutable.ListBuffer[Int] = ListBuffer()

scala> someBuffer += 12
val res0: scala.collection.mutable.ListBuffer[Int] = ListBuffer(12)

scala> someBuffer
val res1: scala.collection.mutable.ListBuffer[Int] = ListBuffer(12)

scala> someBuffer = new ListBuffer[Int]()
-- [E052] Type Error: ----------------------------------------------------
1 |someBuffer = new ListBuffer[Int]()
  |^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  |Reassignment to val someBuffer
  |
  | longer explanation available when compiling with `-explain`
1 error found
```

- *VAL* CREATES *IMMUTABLE VARIABLE* IN THE SENSE THAT YOU CANNOT ***REASSIGN*** IT
- BUT IF *VAL* VARIABLE CONTAINS OBJECT WHICH IS MUTABLE ITSELF AND YOU MODIFY THE OBJECT (WITHOUT VARIABLE REASSINGMENT)
  THEN IT WILL WORK

DECLARING VARIABLE TYPES

- As it's been already mentioned, you can omit type declaring when creating a new variable – the compiler can *infer* the variable type
- On the other hand, if you explicitly declare the variable type, you have more control over it

```
scala> val floatingNumber = 5.5
val floatingNumber: Double = 5.5

scala> val anotherFloatingNumber: Float = 5.5
val anotherFloatingNumber: Float = 5.5
```

CHARS AND STRINGS

- ***CHAR*** VARIABLE CONTAINS A SINGLE CHARACTER; WE CREATE *CHARS* USING SINGLE QUOTES
- ***STRING*** CAN CONTAIN MORE CHARACTERS; WE CREATE *STRINGS* USING DOUBLE QUOTES
- WE CAN CONCATENATE STRINGS USING + AND INTERPOLATE THEM USING *S AND* $

```
scala> val singleCharacter = 'C'
val singleCharacter: Char = C

scala> val multipleCharacters = "bla bla blabla"
val multipleCharacters: String = bla bla blabla

scala> multipleCharacters + " something"
val res2: String = bla bla blabla something

scala> val firstName = "Jan"
val firstName: String = Jan

scala> println(s"His name is $firstName")
His name is Jan
```

CONDITIONALS – *IF/ELSE*

Scala 2

```
if (x < 0) {
  println("negative")
} else if (x == 0) {
  println("zero")
} else {
  println("positive")
}
```

Scala 3

```
if x < 0 then
  println("negative")
else if x == 0 then
  println("zero")
else
  println("positive")
```

FOR LOOP

Scala 2

```
val ints = List(1, 2, 3, 4, 5)
for (i <- ints) println(i)
```

Scala 3

```
val ints = List(1, 2, 3, 4, 5)
for i <- ints do println(i)
```

and many additional possibilities …

WHILE LOOP

Scala 2

```
while (x >= 0) { x = f(x) }
```

Scala 3

```
while x >= 0 do x = f(x)
```

SOURCES

- SCALA-LANG.ORG
- WIKIMEDIA COMMONS
- GEEKSFORGEEKS.ORG
- INTELLIPAAT.COM
- JAVATPOINT.COM
- LIVEBOOK.MANNING.COM
- TOWARDSDATASCIENCE.COM