Intro

(who we are)

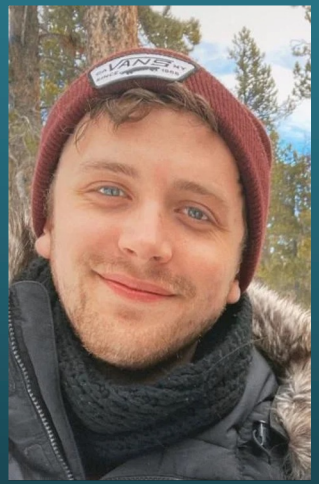
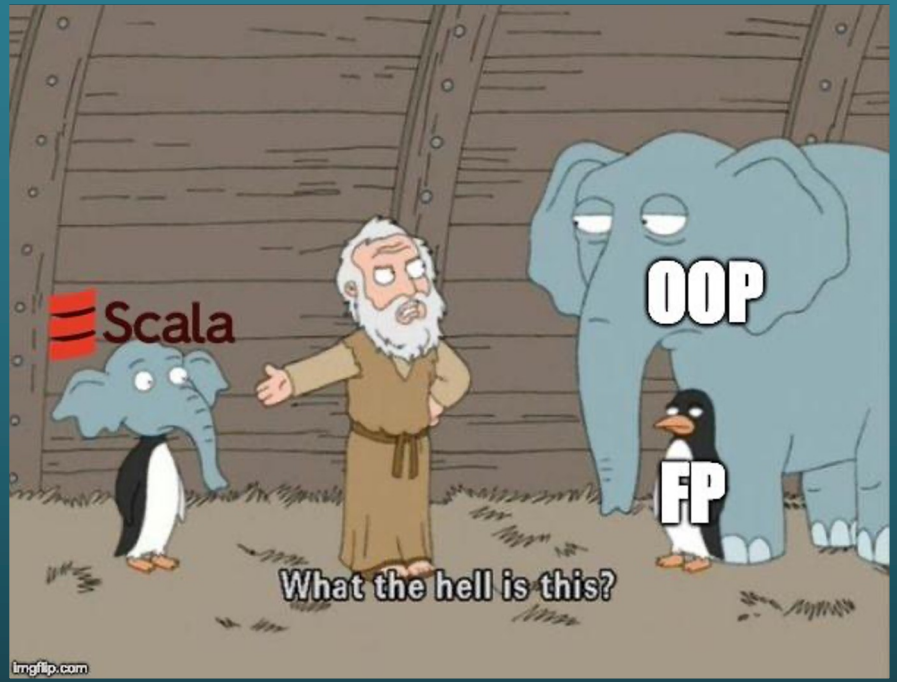


Table of Contents



History & Motivation

EPFL

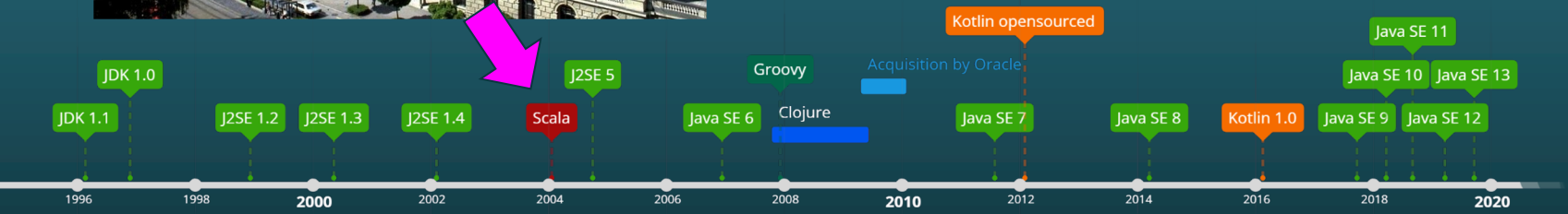


Figure 1: Java version release dates 2006-2021 in context to Scala release.
Source: https://time.graphics/event/3439612

History & Motivation

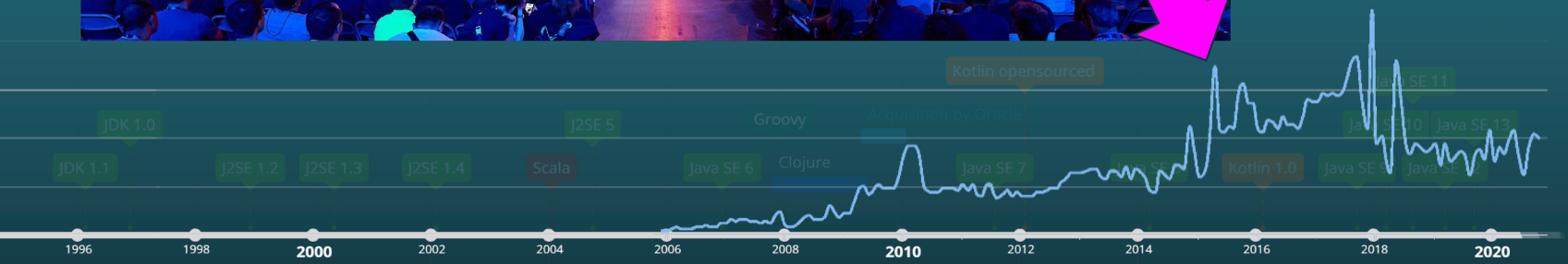


Figure 2: TIOBE ratings of Scala from 2006-2021
Source: https://www.tiobe.com/tiobe-index/scala/

History & Motivation

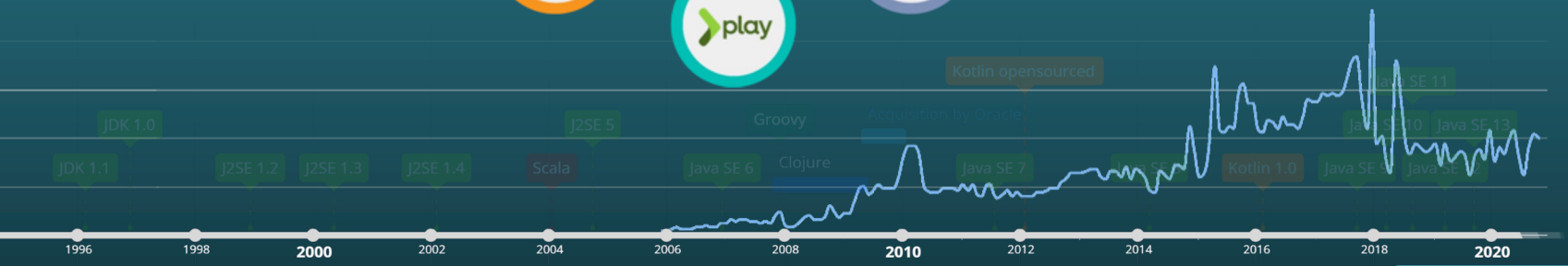
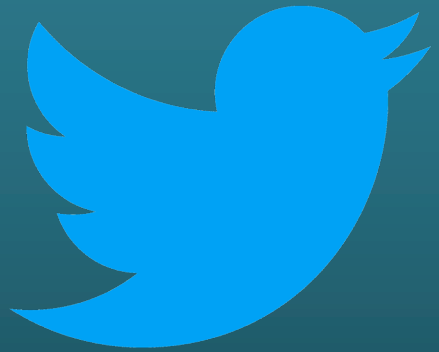


Figure 2: TIOBE ratings of Scala from 2006-2021
Source: https://www.tiobe.com/tiobe-index/scala/

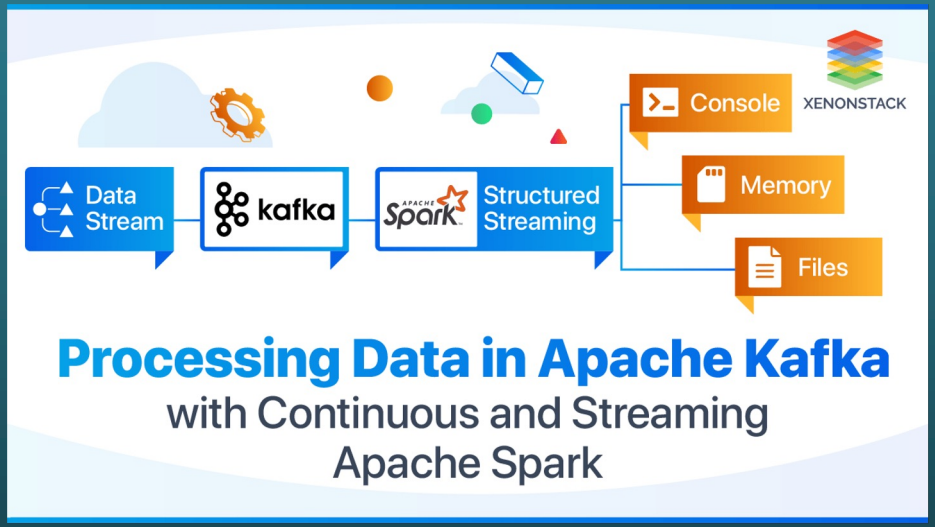
Big Data Energy: Java vs Scala

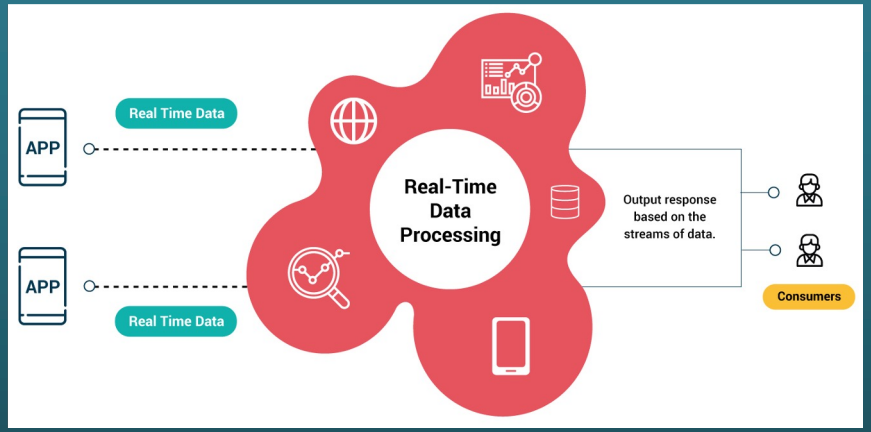
Scala in the Wild

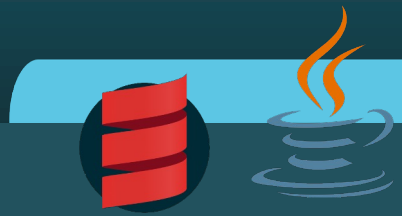
Scala in the Wild



Scala in the Wild







Java & Scala similarities

Multiple Inheritances via Mix-In Traits

```
// JAVA
interface Alpha
interface Bravo
interface Charlie
// order matters
// Alpha searched first, then Bravo, then Charlie
class Delta implements Alpha, Bravo, Charlie
```


```
// SCALA
trait Alpha
trait Bravo
trait Charlie
// order matters
// Alpha searched first, then Bravo, then Charlie
class Delta extends Alpha with Bravo with Charlie
```

Java & Scala similarities

The Strategy Pattern

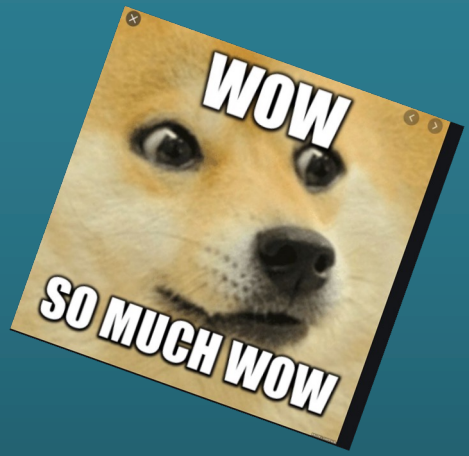
```
// JAVA
interface AbstractStrategy {
        public void foo();
}
class ConcreteStrategyA implements AbstractStrategy {
        public void foo() { System.out.println("HI"); }
}
class ConcreteStrategyB implements AbstractStrategy {
        public void foo() { System.out.println("BYE"); }
}
class Alpha {
        private AbstractStrategy strategy;
        public Alpha(AbstractStrategy strategy) {
                this.strategy = strategy;
        }
}
```

```
// SCALA
trait AbstractStrategy {
        def foo
}
class ConcreteStrategyA extends AbstractStrategy {
        def foo = println("HI")
}
class ConcreteStrategyB extends AbstractStrategy {
        def foo = println("BYE")
}
class Alpha(strategy: AbstractStrategy)
```

Java & Scala differences

The Singleton Pattern

```
// JAVA
public class Alpha {
        private instance = null;
        private Alpha() { return; }
        public getInstance() {
                if (instance == null) {
                        instance = new Alpha();
                }
                return instance;
        }
}
```




```
// SCALA
object Alpha
```

Java & Scala differences

The Decorator Pattern

```
// JAVA
public class Alpha {
        private int x;
        public Alpha(int x) {
                this.x = x;
        }
}

public class Bravo extends Alpha {
        private Alpha alpha;
         private int y;
        public Bravo(Alpha alpha, int y) {
                this.alpha = new Alpha(y + 1);
                this.y = y;
        }
}
```


```
// Scala
class Alpha(x: Int)
class Bravo(a: Alpha, y: Int) extends Alpha(y + 1)
```

Java & Scala complex pattern

The Observer Pattern

```
// JAVA
// https://www.baeldung.com/java-observer-pattern
import java.beans.PropertyChangeListener;
import java.beans.PropertyChangeSupport;

public class Publisher {
        private PropertyChangeSupport support;

        public Publisher() {
                support = new PropertyChangeSupport(this);
        }
        public void publishMsg(String msg) {
                this.updateEvent(msg);
        }
}
```

```
// JAVA
// https://www.baeldung.com/java-observer-pattern
import java.beans.PropertyChangeListener;
import java.beans.PropertyChangeEvent;
import java.util.ArrayList;

public class Subscriber implements PropertyChangeListener {
        protected List<String> events;
        public Subscriber(String name) {
                this.events = new ArrayList<>();
                this.connect();
        }
}
```

Java & Scala complex pattern

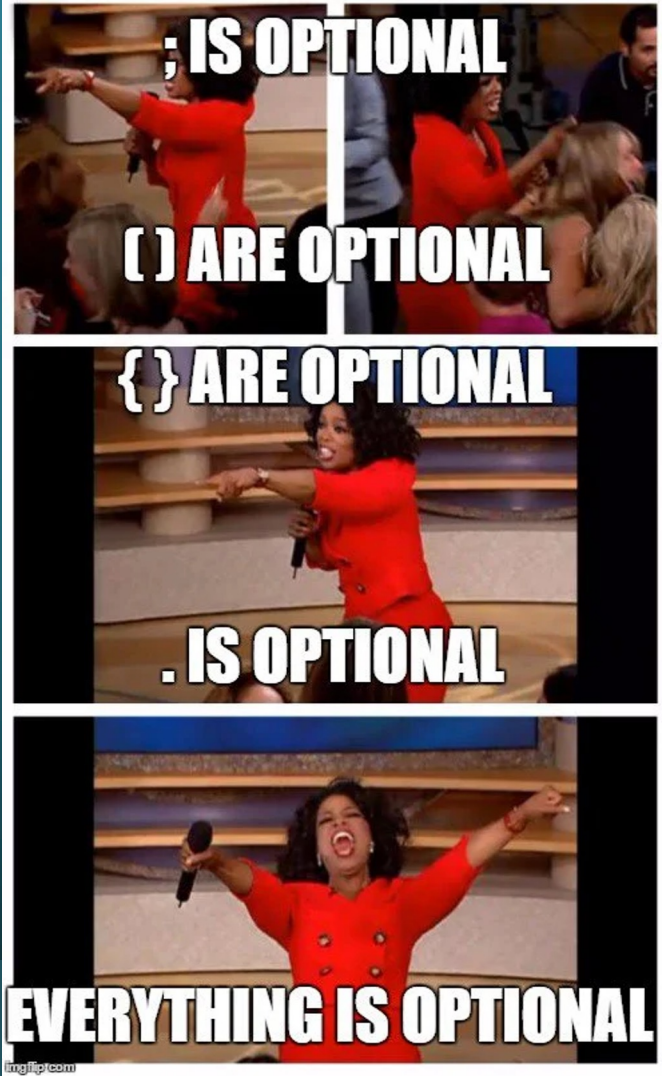
The Observer Pattern and Type Class Pattern

```
// SCALA:: DEPENDS ON: TypeClassPattern https://stackoverflow.com/questions/13435151/implementing-observer-pattern
import scala.collection.mutable.ArrayBuffer

trait Publisher[M] {
        this: M =>
        private var subscribers: List[Subscriber[M]] = Nil
        def addSubscriber(subscriber: Subscriber[M]) = subscribers = subscriber :: subscribers
        def notifySubscribers() = subscribers.foreach(_.receiveUpdate(this))
}

class Subscriber[M]() {
        val events = new ArrayBuffer[M]()
        def receiveUpdate(event: M) = {
                this.events += event
        }
}
```

Scala functional features

```
1.+(2)
```

```
1 + 2
```



Scala functional features

```
def f(i: Int): Int = i + 1
val l:List[Int] = List(6, 7, 8)
```

`l.map(f)` vs `l map f`

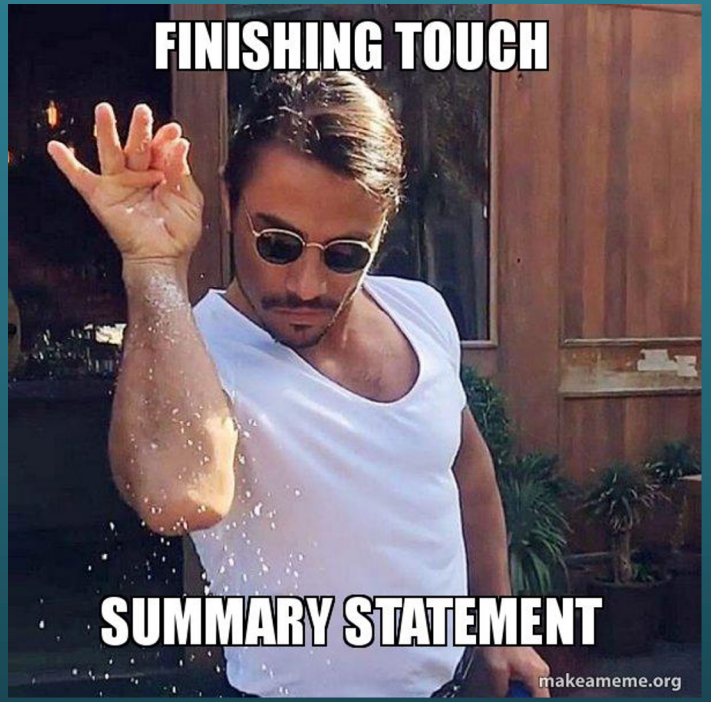
Results in List(7, 8, 9)

Scala complications

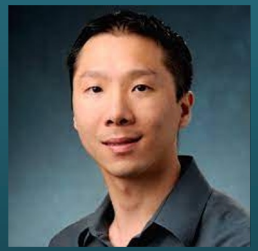
```
sealed trait Tree {
        def +(i: Int): Tree = this match {
                case EmptyTree => Node(EmptyTree, i, EmptyTree)
                case Node(l, d, r) if i <= d => Node(l + i, d, r)
                case Node(l, d, r) => Node(l, d, r + i)
        }
        def foldLeft[A](z: A)(sc: (A, Int) => A):A = this match {
                case EmptyTree => z
                case Node(l, d, r) => r.foldLeft(sc(l.foldLeft(z)(sc), d))(sc)
        }
        def map(k: Int => Int): Tree = this match {
                case EmptyTree => this
                case Node(l, d, r) => Node(l map k, k(d), r map k)
        }
}
case object EmptyTree extends Tree
case class Node(l: Tree, d: Int, r: Tree) extends Tree
```

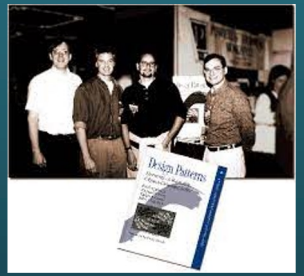
(EmptyTree + 22 + 4448 + -1 map { _ + 5 } foldLeft (0)){ _ + _ }

Table of Contents



Thank you





LettuceWrapInc.
And many many more