COMPLEX EVENT PROCESSING: LANGUAGE-LEVEL INTEGRATION INTO SCALA

Bachelor thesis by Mark Goldenstein

Software Technology Group
Fachbereich Informatik
TU Darmstadt

DEFINITIONS

Event Processing

is a method of tracking and analyzing (*processing*)
streams of information (*data*)
about things that happen (*events*).

Complex Event Processing (CEP)

is event processing that
combines multiple sources of simple events to
higher-level and more abstract *complex events*.

EXAMPLE OF CEP AT THE STOCK MARKET

Event Sources	Event Query	Complex Event	Reaction
Exchange Data (Nasdaq, …)	• the share price of a company is going up for 10 consecutive seconds AND • an analyst has upgraded the company to a *buy* within the last minute	initiating a trade could be profitable	place a buy order
Analyst Reports			

CHARACTERISTICS OF CEP

- Events should be processed as fast as possible.
- Traditional databases are not applicable because
  - data has to be stored and indexed
  - processing only occurs when explicitly asked, i.e. asynchronously with respect to its arrival.
- CEP requires systems that were specifically designed to process information as a flow.

STATE OF THE ART

Which tools do we have to work with events?

Programming Languages

- OOP
  - Observer Pattern (Java)
  - Events as Object Attributes (C#)
- AOP (AspectJ)
  - Implicit Events
  - Declarative Join Points
- Mix of the above (EScala, Ptolemy)

STATE OF THE ART

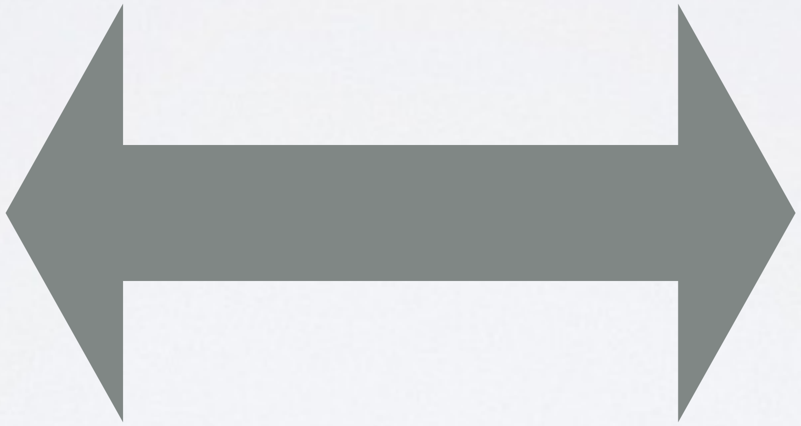
Which tools do we have to work with events?

CEP Libraries

- Stream Processing systems (Aurora, Borealis)
- Full CEP systems (Esper, Cayuga)
- Usually an SQL-like API (with custom extensions)

STATE OF THE ART

Which tools do we have to work with events?



Programming Languages

compiler checked

CEP Libraries

SQL-like API

GOAL: Language-level integration for CEP

GOAL: Language-level integration for CEP

INTRODUCING CESCALA

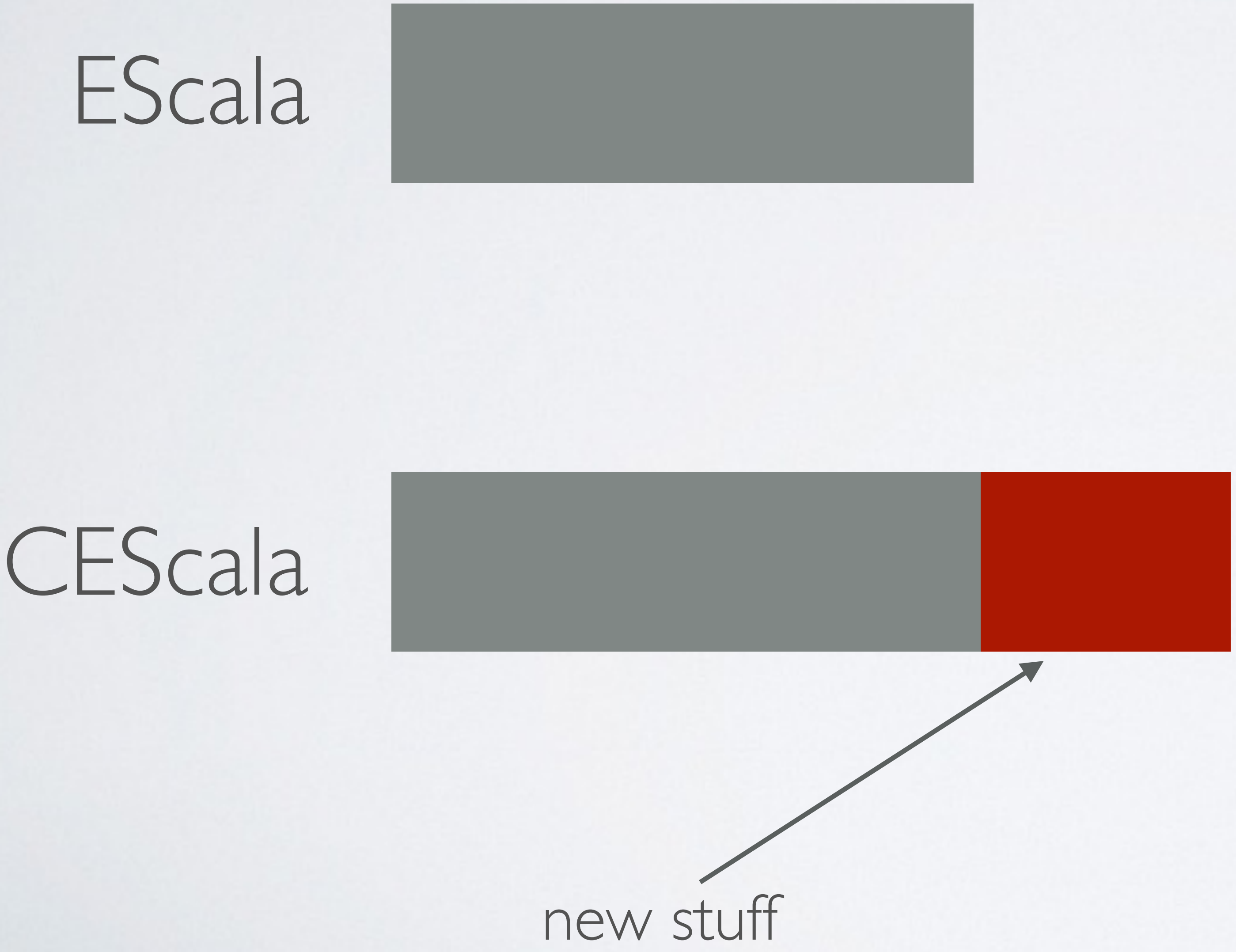
Design Characteristics

CEScala should ...

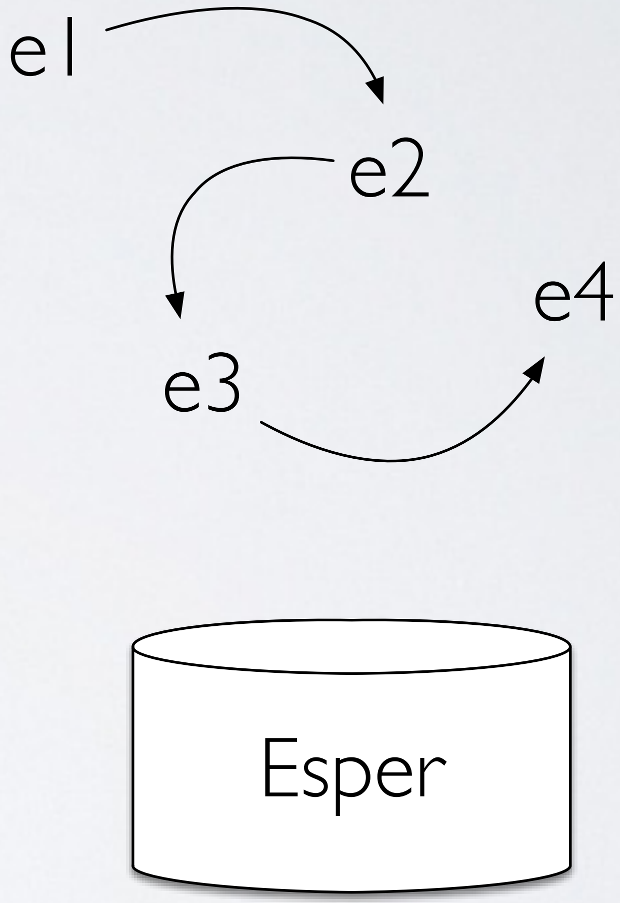
- provide the expressivity of CEP libraries
- provide the level of language integration for event processing offered by event-based languages

INTRODUCING CESCALA

API



Implementation



THE CESCALA DSL

based on EScala

Declaring an Event

one property

```
val e1 = new ImperativeEvent[Int]
```

multiple properties

```
val e2 = new ImperativeEvent[(Int, String)]
```

THE CESCALA DSL

based on EScala

```
val e2 = new ImperativeEvent[(Int, String)]
```

Declaring a Reaction

```
val r2 = (e: (Int, String)) => println(e._2)
```

Binding a Reaction to an Event

```
e2 += r2
```

THE CESCALA DSL

based on EScala

```
val e2 = new ImperativeEvent[(Int, String)]
```

Triggering an Event

```
e2(42, “Hello World!”)
```

THE CESCALA DSL

based on EScala

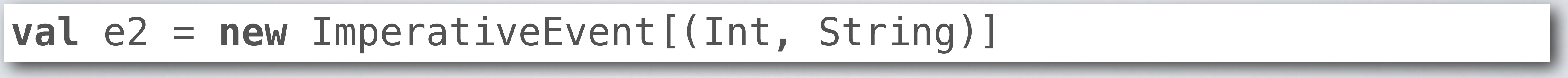
```
val e2 = new ImperativeEvent[(Int, String)]
```

Transforming an Event

```
val e3 = e2.map((e: (Int, String)) => (e._1, e._2.length))
```

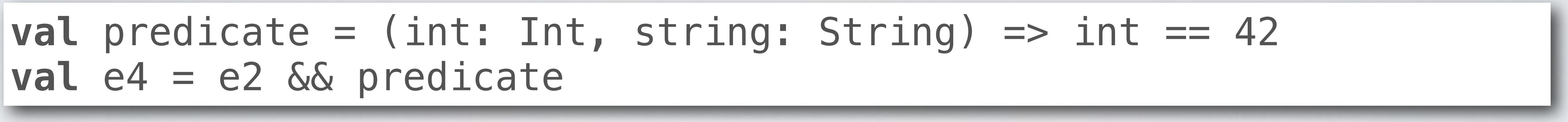
THE CESCALA DSL

based on EScala



```
val e2 = new ImperativeEvent[(Int, String)]
```

Filtering an Event



```
val predicate = (int: Int, string: String) => int == 42
val e4 = e2 && predicate
```

THE CESCALA DSL

based on EScala

```
val e1 = new ImperativeEvent[(Int, String)]
val e2 = new ImperativeEvent[(Int, String)]
```

Composing Events

```
val e3 = e1 || e2
```

THE CESCALA DSL

new features

```
val e1 = new ImperativeEvent[(Int, String)]
val e2 = new ImperativeEvent[(Int, String)]
```

Joining Events

```
val e3 = e1 join e2 window time(30 sec) on ((a,b) => a._1===b._1)
```

Creating a Reaction to a Joined Event

```
val r3 = (e: (Int, String, Int, String)) =>
  println(e._2 + “ “ + e._4)
e3 += r3
```

THE CESCALA DSL

new features

```
val e1 = new ImperativeEvent[(Int, String)]
```

Creating a Repeat Event Pattern

```
val e4 = e1 repeat 3
```

Creating a Reaction to a Repeat Event Pattern

```
val r4 = (e: Seq[[(Int, String)]) =>
  println(e(0)._2 + “ “ + e(1)._2 + “ “ + e(2)._2)
e4 += r4
```

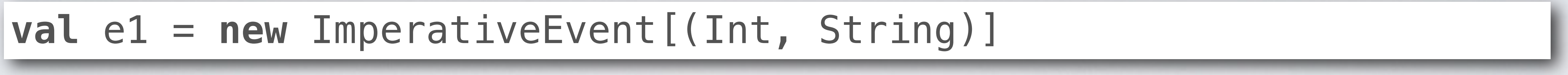
THE CESCALA DSL

Custom Types are supported.

```
class CustomEventType(val int: Int, val string: String) {

  override def equals(o: Any) = o match {
    case o: IntString => int == o.int && string == o.string
    case _ => false
  }

  override def hashCode = string.hashCode + int

}
```

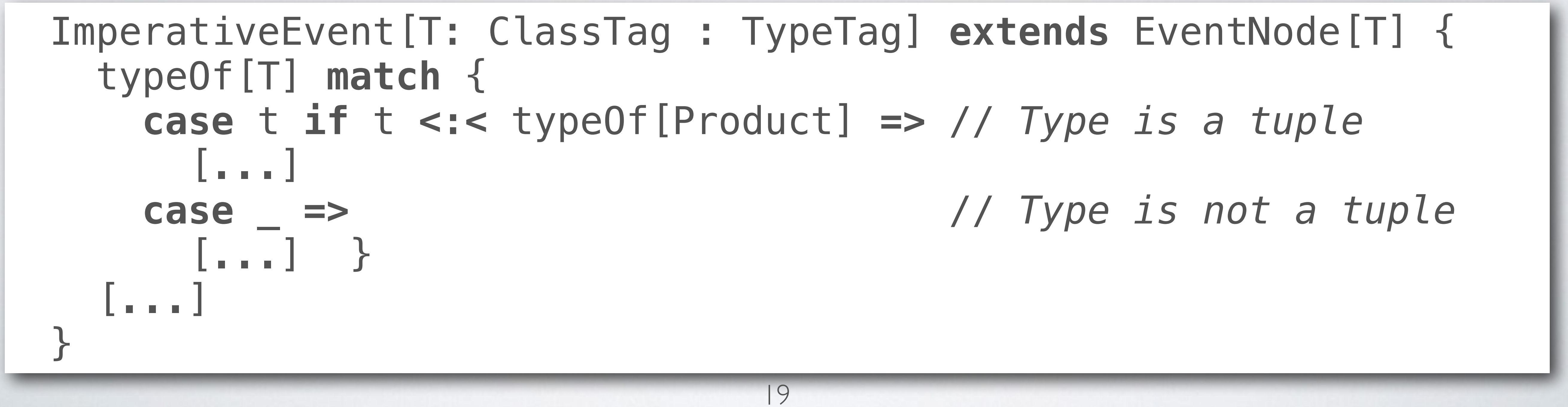
IMPLEMENTATION DETAILS OF CESCALA

makes use of Scala Reflection and Scala shapeless
to map from CEScala's API to Esper



```
val e1 = new ImperativeEvent[(Int, String)]
```

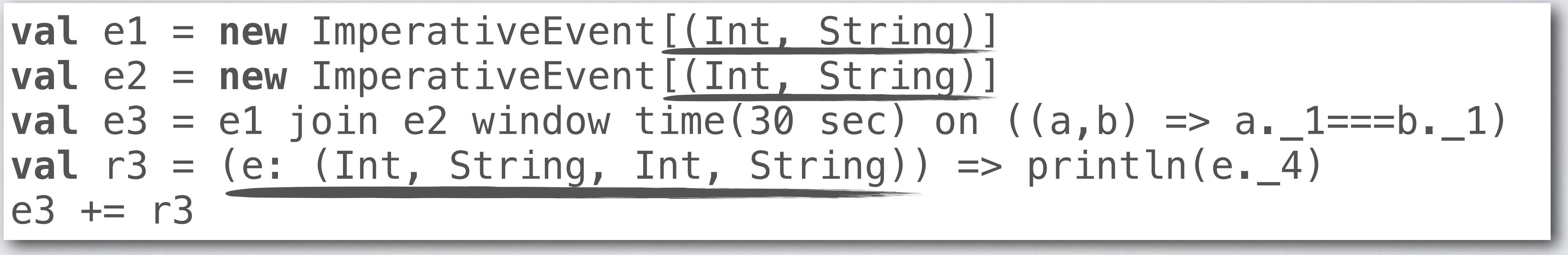
We need to extract `Int` and `String` and pass them to Esper.



```
ImperativeEvent[T: ClassTag : TypeTag] extends EventNode[T] {
  typeOf[T] match {
    case t if t <:< typeOf[Product] => // Type is a tuple
      [...]
    case _ =>                          // Type is not a tuple
      [...]  }
  [...]
}
```

IMPLEMENTATION DETAILS OF CESCALA

makes use of Scala Reflection and Scala shapeless to map from CEScala's API to Esper



```
val e1 = new ImperativeEvent[(Int, String)]
val e2 = new ImperativeEvent[(Int, String)]
val e3 = e1 join e2 window time(30 sec) on ((a,b) => a._1===b._1)
val r3 = (e: (Int, String, Int, String)) => println(e._4)
e3 += r3
```

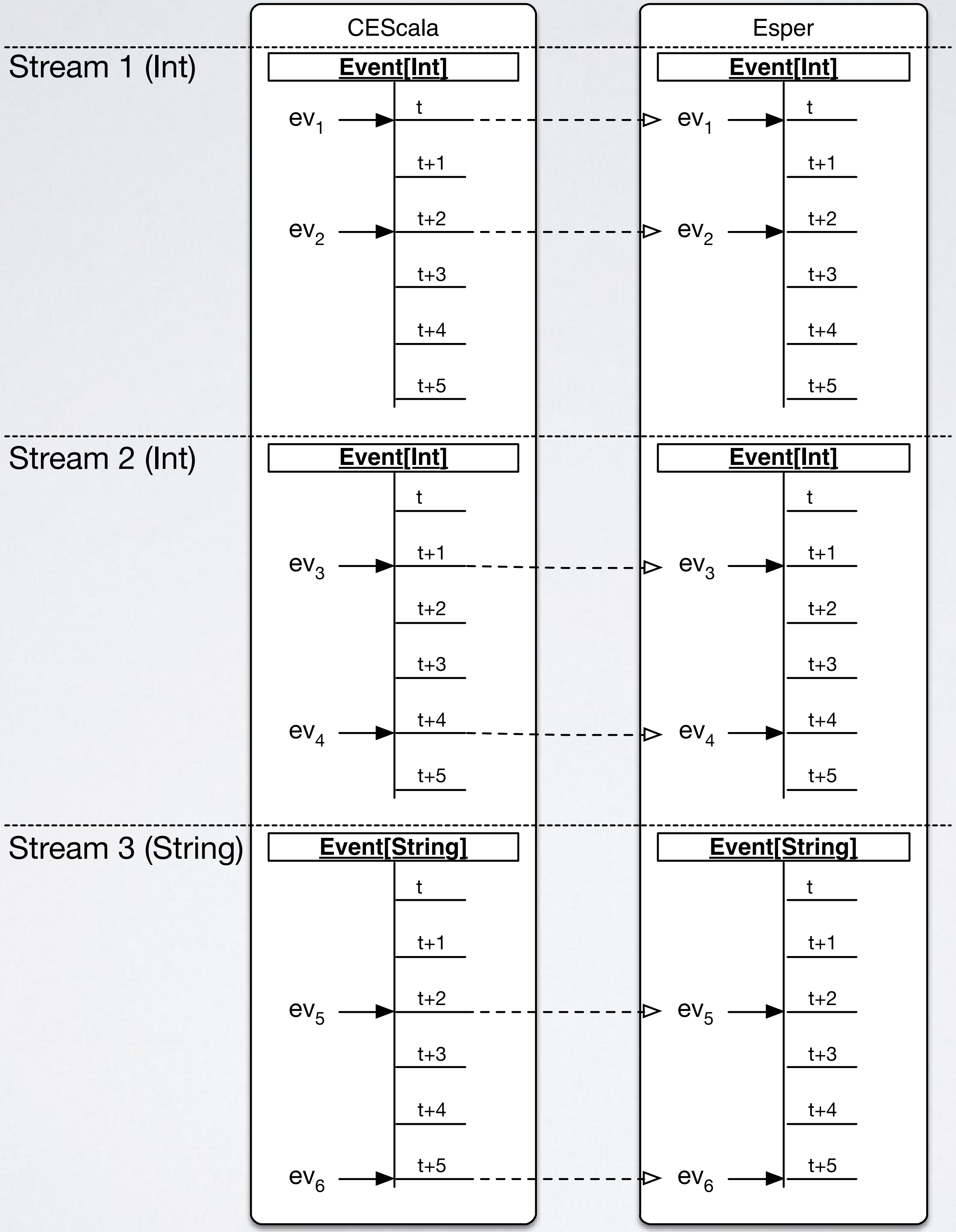
IMPLEMENTATION DETAILS OF CESCALA

Design Parameters

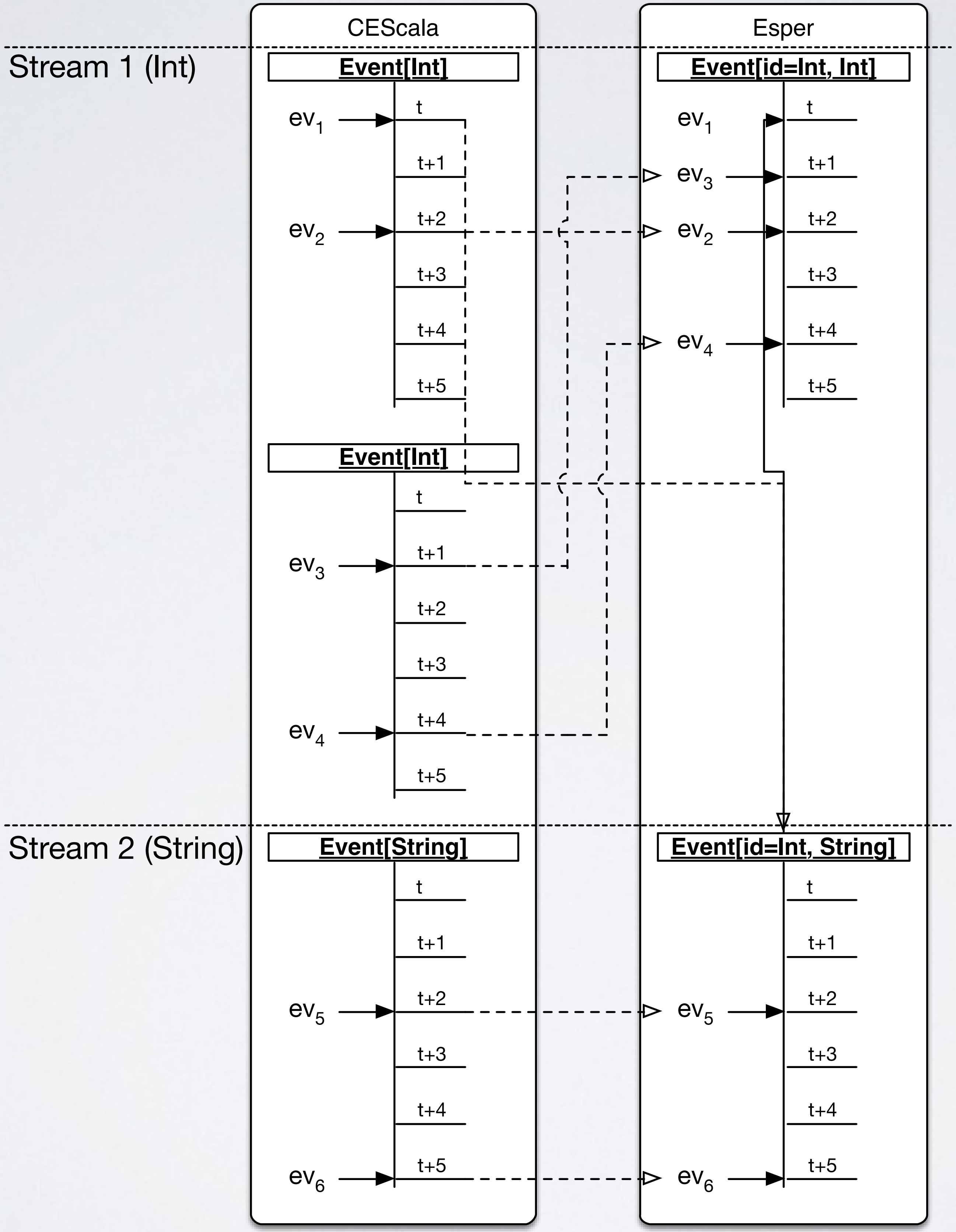
- Should we map CEScala events to separate event streams in Esper or merge events of the same type to a single stream?

IMPLEMENTATION DETAILS OF CESCALA

Separate event streams



Merged event streams



IMPLEMENTATION DETAILS OF CESCALA

Design Parameters

- Should we map CEScala events to distinct event streams in Esper or events of the same type to a single stream?
- How do we attach reactions to events? - Esper provides us with two different options (UpdateListeners and Subscribers).

Solution

Implement all variants and compare the performances.

PERFORMANCE EVALUATION

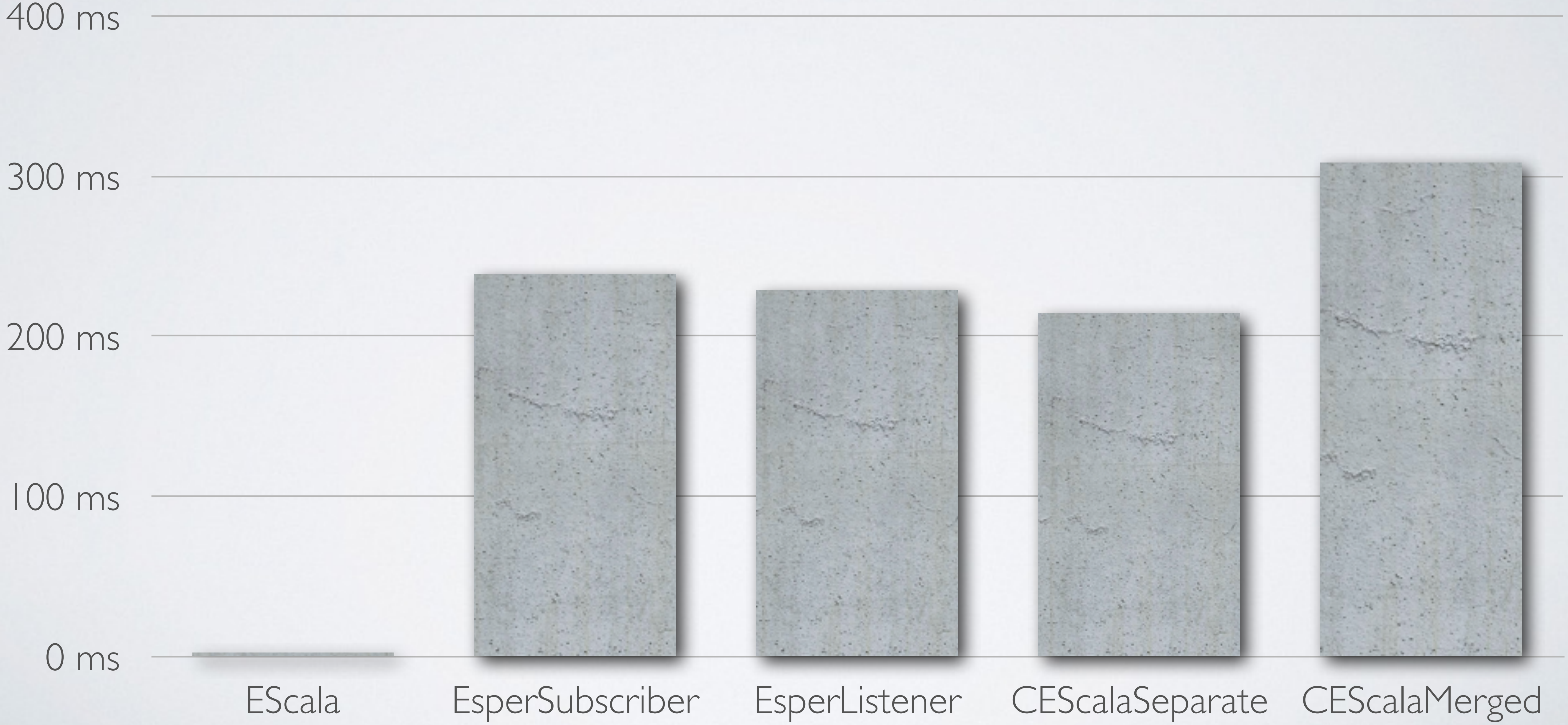
How much time does each library need
to trigger 800 events 100 times?

Libraries included in the comparison:

- EScala
- CEScalaSeparate
- CEScalaMerged
- EsperSubscriber
- EsperListener

PERFORMANCE EVALUATION

How much time does each library need to trigger 800 events 100 times?



SUMMARY

CEScala is a DSL which combines

- the level of language-integration of EScala with
- the expressivity of the CEP engine Esper
- at a negligible performance cost (compared to Esper).