TECHNISCHE UNIVERSITÄT DARMSTADT

erc

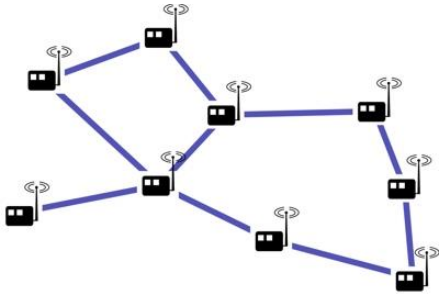
Language Support for Reactive Applications

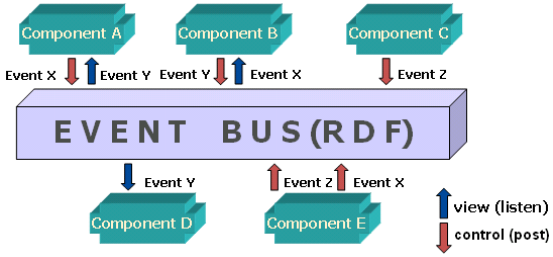
Guido Salvaneschi
Software Technology Group (prof. Mira Mezini)
University of Darmstadt

Reactive Applications

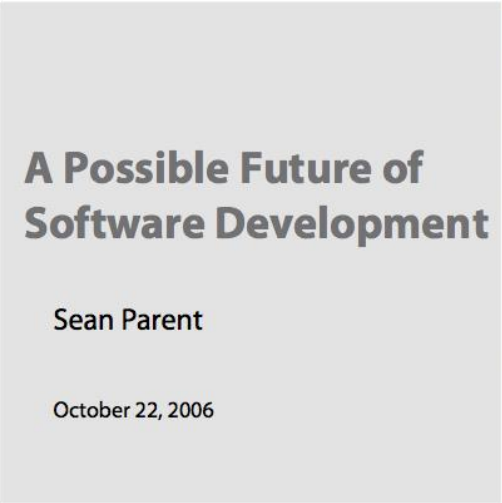
- External/internal **events** trigger a reaction
  - User input, network, interrupt, Data from sensors, ..

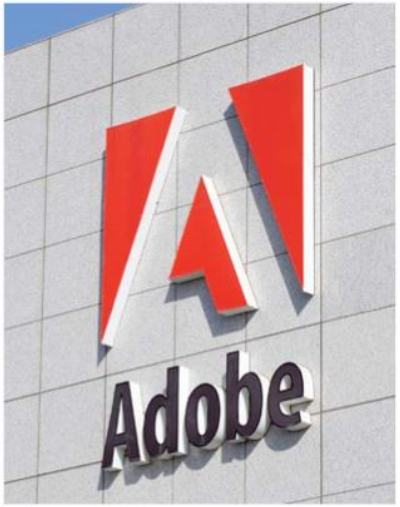
- Classic examples:
  - Web applications
  - Data change in MVC
  - …





Common but Hard to Implement!



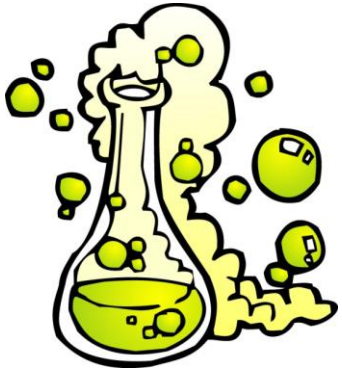


http://stlab.adobe.com/wiki/images/0/0c/Possible_future.pdf

- 1/3 of code in desktop applications devoted to event handling
- 1/2 of bugs reported during a product cycle exist in this code

Reactive Applications

- Implementation in OO languages
  - Observer design pattern
  - No composition
  - Boilerplate code
  - Hard to understand/analyze
  - …



Most languages just ignore reactivity in their design…

REScala: Events and OO Programming

- Objects fields model state
- Event handlers (imperatively) update state

```
abstract class Figure { ...
  protected evt moved[Unit] = after(moveBy)
  evt resized[Unit]
  evt changed[Unit] = resized || moved || after(setColor)
  evt invalidated[Rectangle] = changed.map(() => getBounds())
    ...
  def moveBy(dx: Int, dy: Int) { position.move(dx, dy) }
  def setColor(col: Color) { color = col }
  def getBounds(): Rectangle   ...
}
```

```
class RectangleFigure extends Figure f
  evt resized[Unit] = after(resize) || after(setBounds)
  override evt moved[Unit] = super.moved || after(setBounds)
    ...
  def resize(size: Size) { this.size = size }
  def setBounds(x1: Int, y1: Int, x2: Int, y2: Int) {
    position.set(x1, y1); size.set(x2 - x1, y2 - y1)
  } ...
}
```

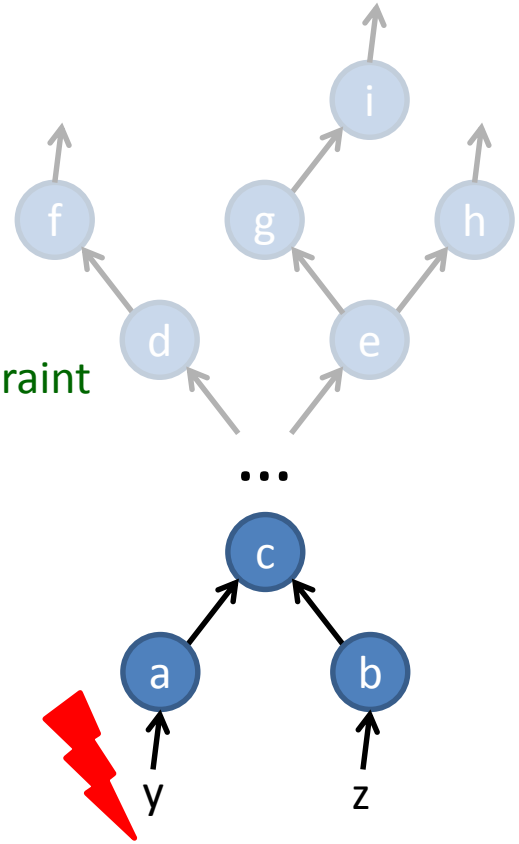
EVENTS

(Functional) Reactive Languages

- Signals: What about expressing functional dependencies as constraints ?

```
val a = 3
val b = 7
val c = a + b // Statement
 …
println(c)
> 10
a= 4
println(c)
> 10
```

```
val a = Var(3)
val b = Var(7)
val c = Signal{ a + b } // Constraint
 …
println(c)
> 10
a= 4
println(c)
> 11
```



REScala Signals

```
val tick = new Var(0)
val hour = Signal{ tick() % 24 }
val day = Signal{ (tick()/24)%7 + 1 }
val week = Signal{ ... }
```

SIGNALS

01:12:04
ww:dd:hh

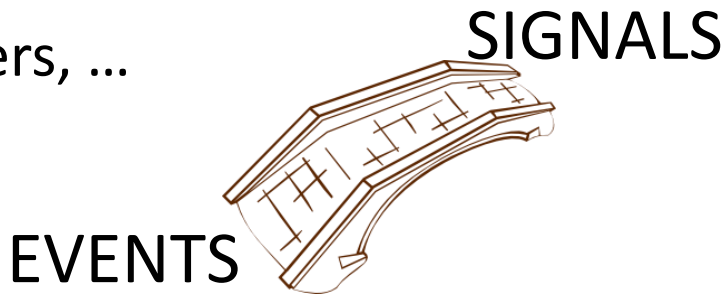
```
imperative evt tick[Unit]
var hour: Int = 0
var day: Int = 0
var week: Int = 0

tick += nextHour
def nextHour() {
  hour = (hour + 1) % 24
}
evt newDay [Unit] = tick && (() => hour == 0)
newDay += nextDay
def nextDay () {
  day = (day + 1) % 7
}
evt newWeek [Unit] = ...
newWeek += nextWeek
def nextWeek() {
  ...
}
```

EVENTS

REScala

Combine Signals + Events in OO Design

- Signals (and events) are objects fields
  - Inheritance, late binding, modifiers, …
- Conversions bridge signals and events



REScala

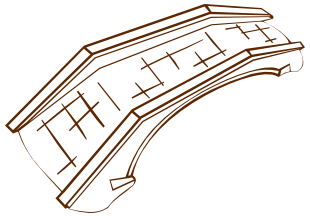
```
val position: Signal[(Int, Int)] = mouse.position
evt mouseMoved: Event[Unit] = position.change()
```

Example: snapshot

- Abstract over state
- Remove handlers
- Express the programmer intention.

E/S

```
evt clicked: Event[Unit] = mouse.clicked
val position: Signal[(Int,Int)] = mouse.position
var lastClickPos = Var(0,0)
val lastClick: Signal[(Int,Int)] = Signal{ lastClickPos() }
clicked += { =>
  lastClickPos()= position()
}
```



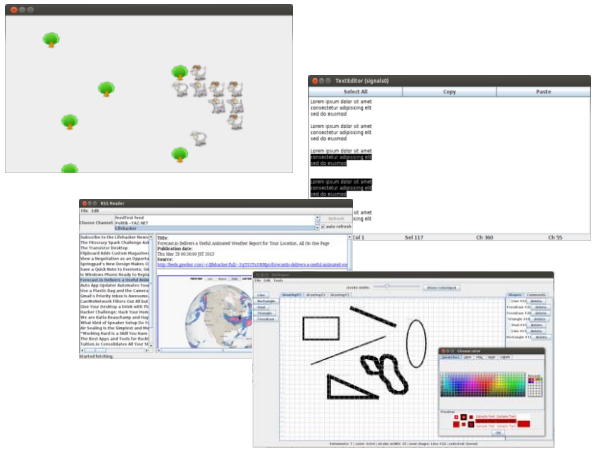
REScala

```
evt clicked: Event[Unit] = mouse.clicked
val position: Signal[(Int,Int)] = mouse.position
val lastClick: Signal[(Int,Int)] = position snapshot clicked
```

Preliminary Validation: REScala Design

- **Q1**: Is the design based REScala better ?
- **Q2**: What is the role of conversion functions ?

- **Refactorings** introduce signals:
  - Candidates: Concerns modelled by functionally dependent values

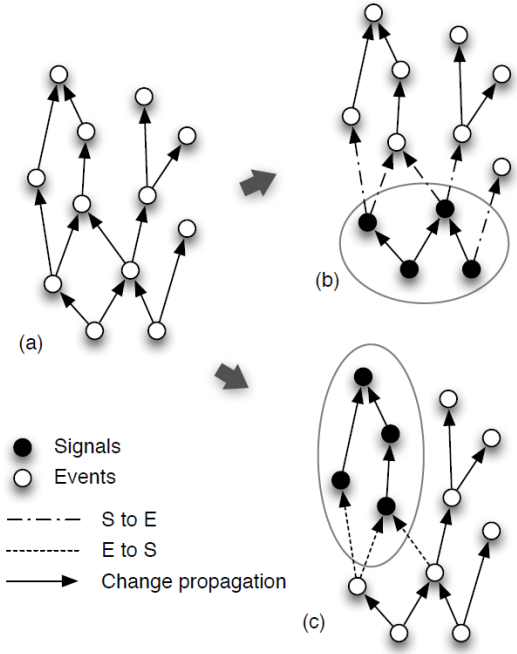


REScala Design: Results

- Removed callbacks - 44%
- More composable abstractions +275% (29 E, 91 S)

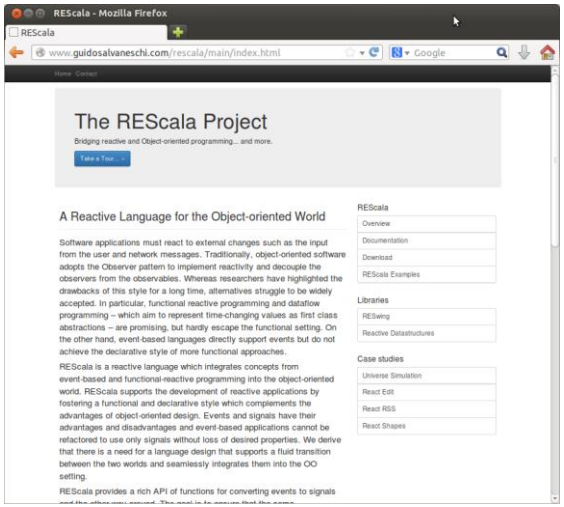
- Use of conversions
  - 13/15 refactorings require conversions

```
val charCountLabel = ReLabel(
    Signal { "Ch " + textArea.charCount() }
)
```



www.rescala-lang.com

- Documentation
- Examples / Case studies
- RESwing library
- Reactive Datastructures
- …



[G. Salvaneschi, G.Hintz, M. Mezini, **REScala: Bridging Between Object-oriented and Functional Style in Reactive Applications**, MODULARITY'14]

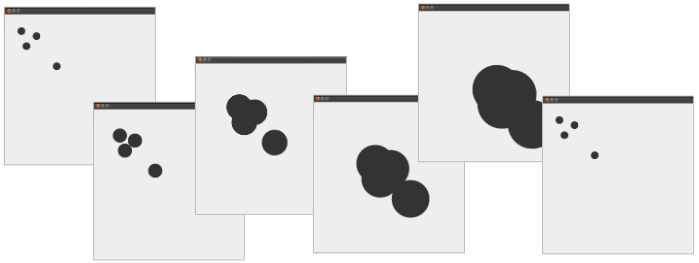
[Guido Salvaneschi, Mira Mezini, **Reactive Behavior in Object-oriented Applications: An Analysis and a Research Roadmap**, MODULARITY'13]

Validation, continued

- Why **RP?**
  - Abstractions are more composable
  - Management of state is not explicit
  - **Declarative style enhances program comprehension**
  - Automatic memory management
  - Consistency guarantees
  - …

"*Obviously, the Flapjax code may not appear any 'easier' to a first-time reader*"
Flapjax: a programming language for Ajax applications. OOPSLA'09

Research Questions

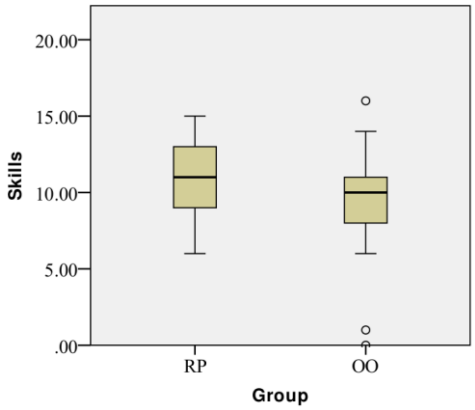


RQ1: *Does REScala increase* ***correctness*** *of comprehension?*

RQ2: *Is comprehending reactive applications in REScala* ***no more time-consuming*** *?*

RQ3: *Does comprehending REScala programs require more advanced programming* ***skills*** *?*

- 10 apps, REScala Vs. OO, 40 subjects
  - Between subjects: no learning effects
  - H0: No difference
  - Balanced skills



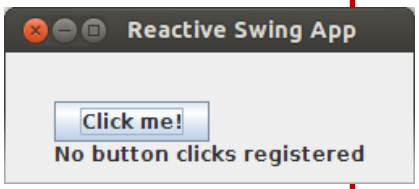
OO

```
/* Create the graphics */
title = "Reactive Swing App“
val button = new Button {
  text = "Click me!"
}
val label = new Label {
  text = "No button clicks registered"
}
contents = new BoxPanel(Orientation.Vertical) {
  contents += button
  contents += label
}
```

```
/* The logic */
listenTo(button)
var nClicks = 0
reactions += {
  case ButtonClicked(b) =>
    nClicks += 1
    label.text = "Number of button clicks: " + nClicks
    if (nClicks > 0)
      button.text = "Click me again"
}
```

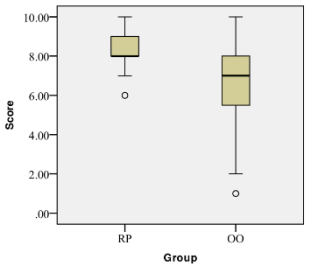
REScala

```
title = "Reactive Swing App"
val label = new ReactiveLabel
val button = new ReactiveButton

 /* The logic */
val nClicks = button.clicked.count

label.text = Signal { ( if (nClicks() == 0) "No" else nClicks() ) + " button clicks registered" }

button.text = Signal { "Click me" + (if (nClicks() == 0) "!" else " again " )}

contents = new BoxPanel(Orientation.Vertical) {
   contents += button
   contents += label
}
```

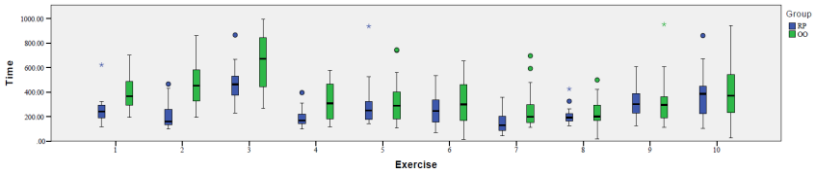


Results

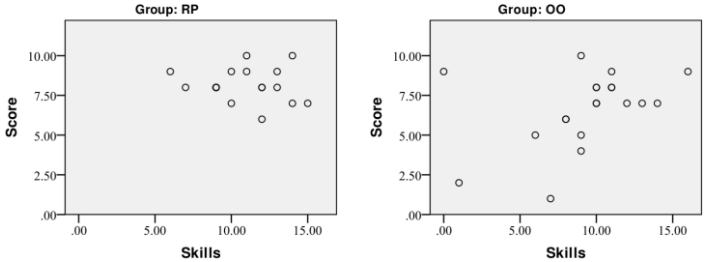
- REScala **increases correctness** of program comprehension



- In REScala, comprehension is **no more time-consuming**



- REScala does not require more advanced **skills** than the OO style



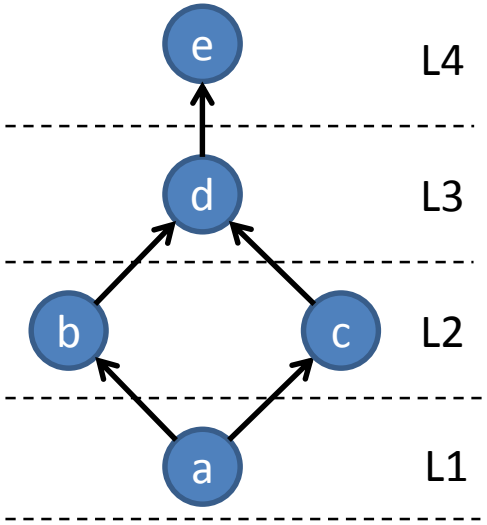
[Guido Salvaneschi, Sven Amann, Sebastian Proksch, Mira Mezini, **An Empirical Study on Program Comprehension with Reactive Programming,** FSE'14.]

MORE ON RP

Glitches

- Glitch: a temporary *spurious* value due to the propagation order.

```
val a = Var(1)
val b = Signal{ a()*2 }
val c = Signal{ a()*3 }
val d = Signal{ b() + c() }
val e =  … d …
```

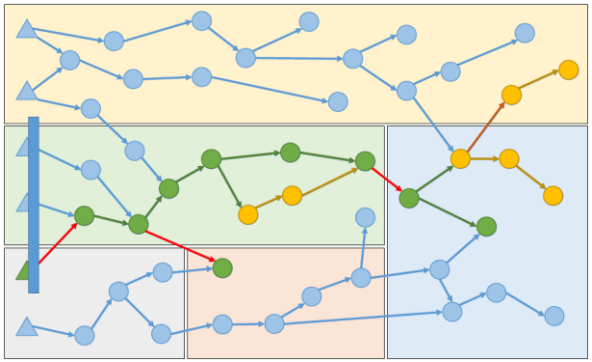


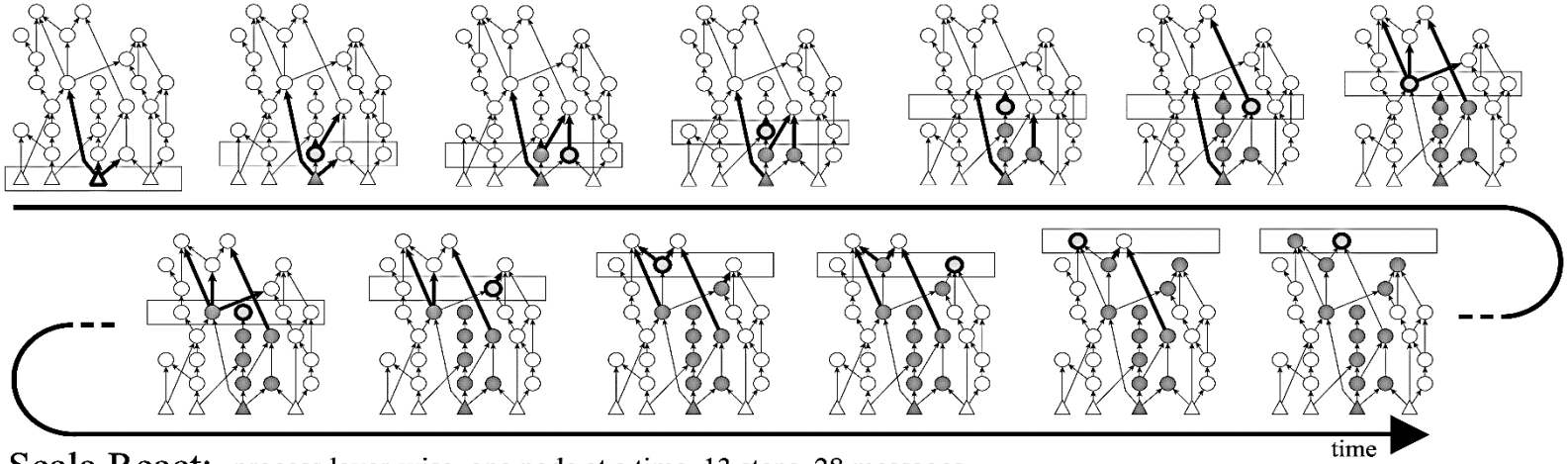
  - Consider the update order abdc
  - a()=2 b<-4, d<-7, c<-6, d<-10

Distributed REScala

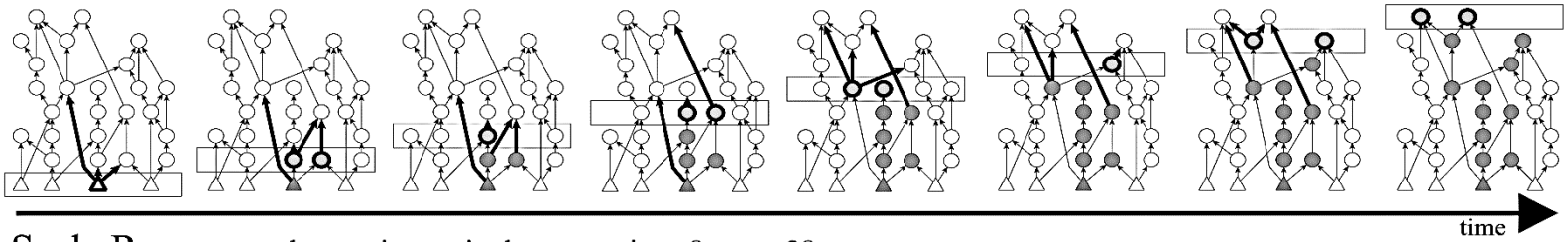
- Pub/sub systems heavily use the Observer pattern
  - Let's use RP!

- Research issues
  - Communication-efficient algorithm
  - Glitch freedom
  - Higher-order signals

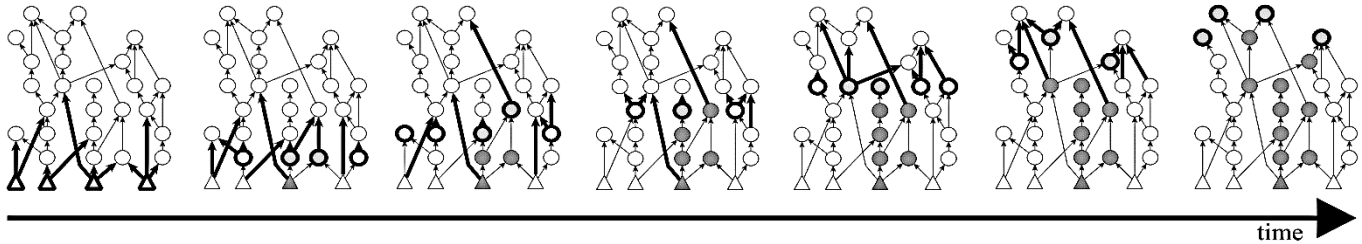




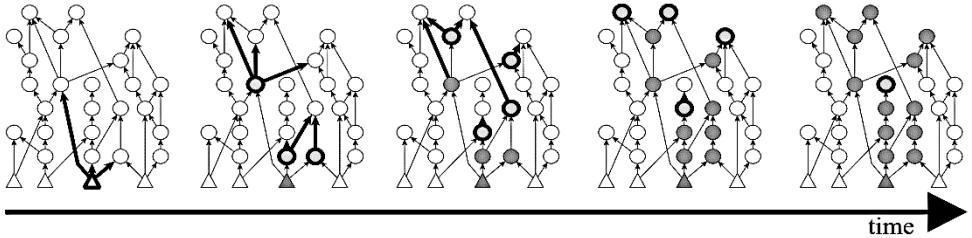
Scala.React: process layer-wise, one node at a time. 13 steps. 28 messages.



Scala.Rx: process layer-wise, entire layer at a time. 8 steps. 28 messages.



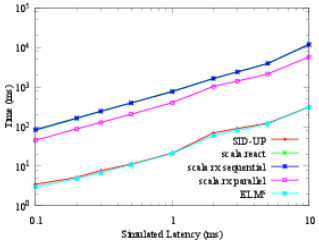
ELM: wait for message from every incoming edge. 7 steps. 37 messages.

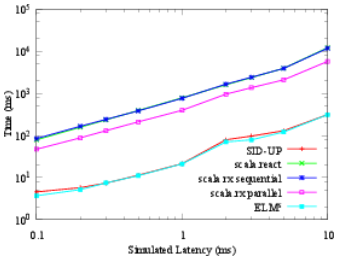


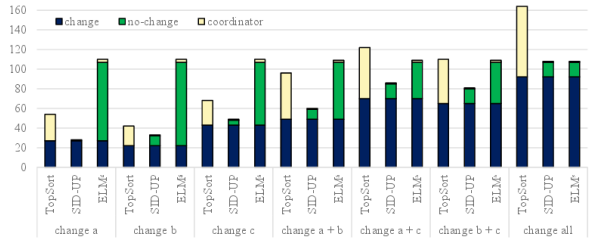
SID-UP: wait for message from every node transitively reachable from a changed source. 5 steps. 15 messages.

Distributed REScala

- Minimize network messages
- Maximize parallelism
- Avoid central coordinator







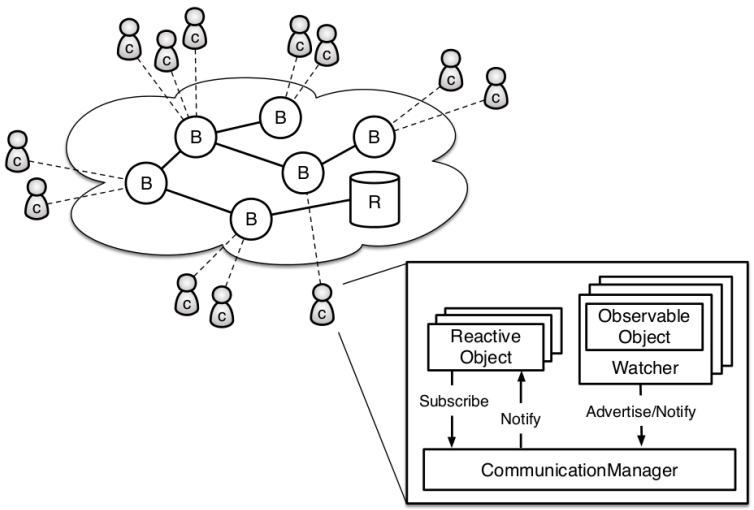
[Guido Salvaneschi, Joscha Drechsler, Mira Mezini. **Towards Distributed Reactive Programming**, COORDINATION'13.]

[Joscha Drechsler, Guido Salvaneschi, Mira Mezini. **Distributed REScala: An Update Algorithm for Distributed Reactive Programming**, OOPSLA'14.]

Distributed RP: Dream

- Applications require different consistency guarantees

- Dream
  - Causal consistency
  - Glitch freedom
  - Transactional consistency



[A. Margara and G. Salvaneschi, **We have a DREAM: Distributed Reactive Programming with Consistency Guarantees**, DEBS ’14.]

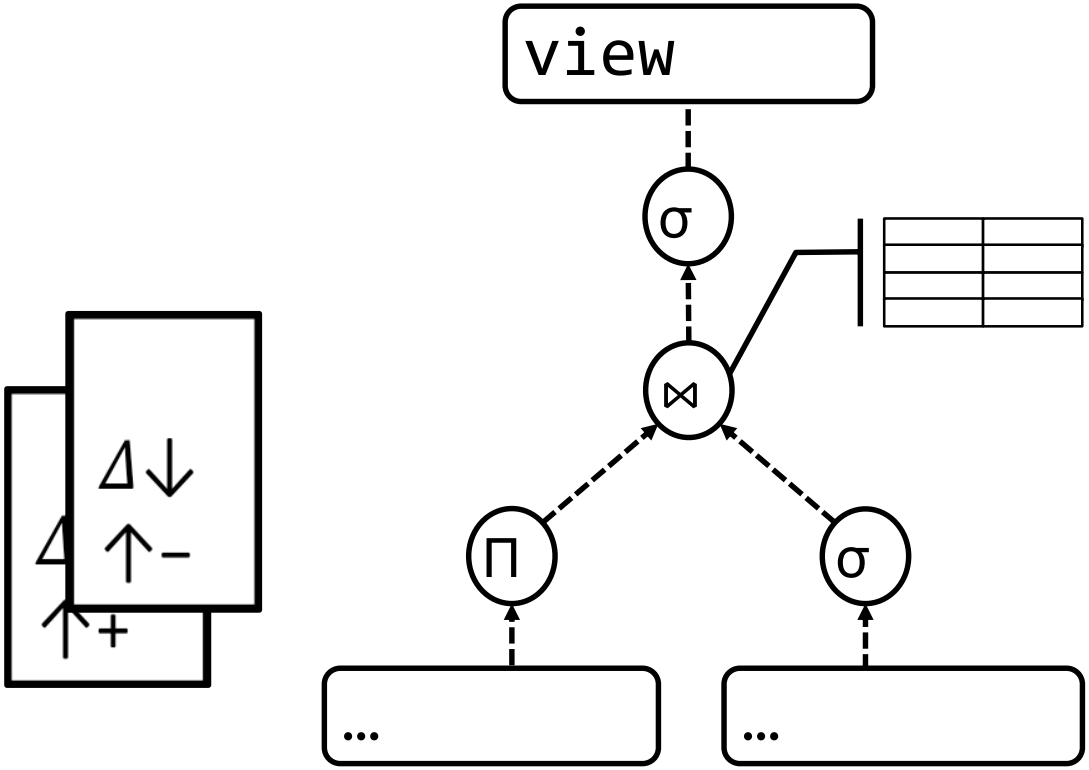
Incrementality: i3QL

- RP requires from-scratch recomputation of dependent values

View

Incremental Computation

Extents



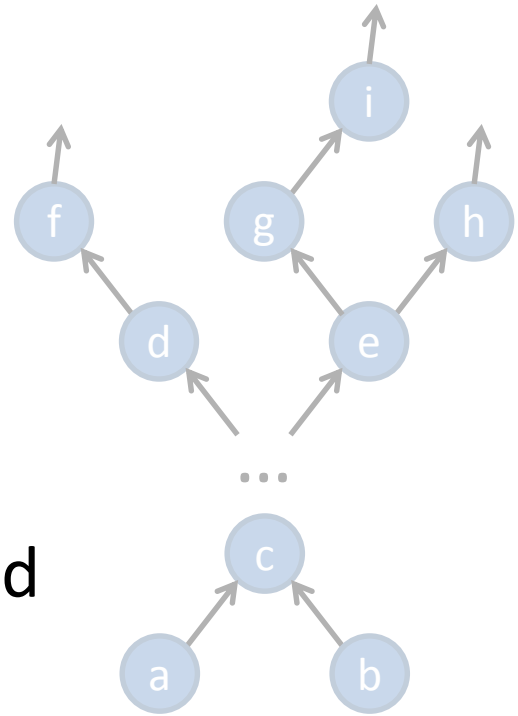
Incrementality: i3QL

- i3QL Optimizations
  - Relational algebra
  - LMS

```
val students: Table[Student] = new Table[Student]()
val sallies: View[Student] =
   (SELECT (∗) FROM students
    WHERE (s => s.firstName == "Sally")).asMaterialized

students.add(new Student("Sally", "Fields"))
students.add(new Student("George", "Tailor"))

sallies.foreach(s => println(s.lastName))
// prints: ”Fields”
students.add(new Student("Sally", "Joel"))
// incremental update of sallies
sallies.foreach(s => println(s.lastName))
// prints: ”Fields” and ”Joel”
```

[Ralf Mitschke, Sebastian Erdweg, Mira Mezini, Mirko Kohler, Guido Salvaneschi, **i3QL: Language-Integrated Live Data Views,** *conditionally accepted* OOPSLA’14.]

Outlook: RP Optimization



- RP is notoriously slow
- Optimizing RP is an open problem

- Previous approaches are quite limited
  - E.g. compile-time only optimization

- Truffle ?

Take Away

- REScala: abstractions for reactive sw (events, signals, …)
  - Improves software design (***REScala design***)
  - Improves program comprehension (***Empirical Study***)

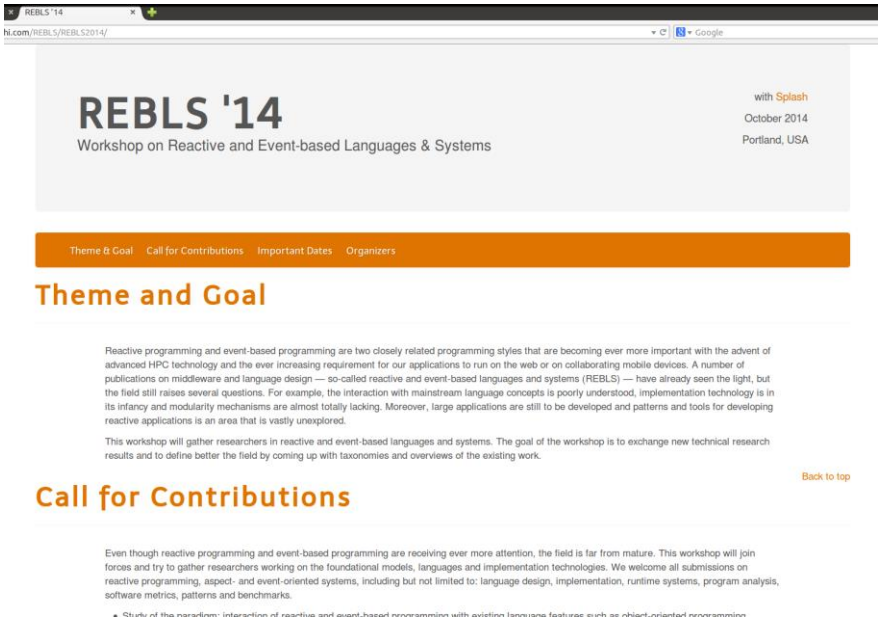
- Going further
  - Distribution (***Distributed REScala/Dream***)
  - Incrementality (***i3QL***)

- Outlook
  - VM/Compiler optimizations

Interested in RP? Be **REBLS**!

- **REBLS**: Reactive Event-Based Languages and System
  - OOPSLA Workshop
  - 21 October - Portland

- **www.rebls-ws.com**



THANKS!

Take Away

- REScala: abstractions for reactive sw (events, signals, …)
  - Improves software design (***REScala design***)
  - Improves program comprehension (***Empirical Study***)

- Going further
  - Distribution (***Distributed REScala/Dream***)
  - Incrementality (***i3QL***)

- Outlook
  - VM/Compiler optimizations