“A Survey on Reactive Programming”

Engineer Bainomugisha, Andoni Lombide Carreton, Tom Van Cutsem, Stijn Mostinckx, Wolfgang De Meuter

ACM Computing Surveys (CSUR) – 2013

PLaNES reading club
21 Jan 2015

Languages

6 Dimensions

Language	Basic abstractions	Evaluation model	Lifting	Multidirectionality	Glitch avoidance	Support for distribution
FRP Siblings						
Fran	behaviours and events	Pull	Explicit	N	Y	N
Yampa	signal functions and events	Pull	Explicit	N	Y	N
FrTime	behaviours and events	Push	Implicit	N	Y	N
NewFran	behaviours and events	Push and Pull	Explicit	N	Y	N
Frappé	behaviours and events	Push	Explicit	N	N	N
Scala.React	signals and events	Push	Manual	N	Y	N
Flapjax	behaviours and events	Push	Explicit and implicit	N	Y (local)	Y
AmbientTalk/R	behaviours and events	Push	Implicit	N	Y (local)	Y
Cousins of Reactive Programming						
Cells	rules, cells and observers	Push	Manual	N	Y	N
Lamport Cells	reactors and reporters	Push and Pull	Manual	N	N	Y
SuperGlue	signals, components, and rules	Push	Manual	N	Y	N
Trellis	cells and rules	Push	Manual	N	Y*	N
Radul/Sussman Propagators	propagators and cells	Push	Manual	Y	N	N
Coherence	reactions and actions	Pull	N/A	Y	Y	N
.NET Rx	events	Push	Manual	N	N?	N

Languages

6 Dimensions

Language	Basic abstractions	Evaluation model	Lifting	Multidirectionality	Glitch avoidance	Support for distribution
FRP Siblings						
Fran	behaviours and events	Pull	Explicit	N	Y	N
Yampa	signal functions and events	Pull	Explicit	N	Y	N
FrTime	behaviours and events	Push	Implicit	N	Y	N
NewFran	behaviours and events	Push and Pull	Explicit	N	Y	N
Frappé	behaviours and events	Push	Explicit	N	N	N
Scala.React	signals and events	Push	Manual	N	Y	N
Flapjax	behaviours and events	Push	Explicit and implicit	N	Y (local)	Y
AmbientTalk/R	behaviours and events	Push	Implicit	N	Y (local)	Y
Cousins of Reactive Programming						
Cells	rules, cells and observers	Push	Manual	N	Y	N
Lamport Cells	reactors and reporters	Push and Pull	Manual	N	N	Y
SuperGlue	signals, components, and rules	Push	Manual	N	Y	N
Trellis	cells and rules	Push	Manual	N	Y*	N
Radul/Sussman Propagators	propagators and cells	Push	Manual	Y	N	N
Coherence	reactions and actions	Pull	N/A	Y	Y	N
.NET Rx	events	Push	Manual	N	N?	N

Christophe's paper next time

VUB's work

Reactive programming

- for event-driven and interactive applications

- express time-varying values

e.g., GUIs, web-apps

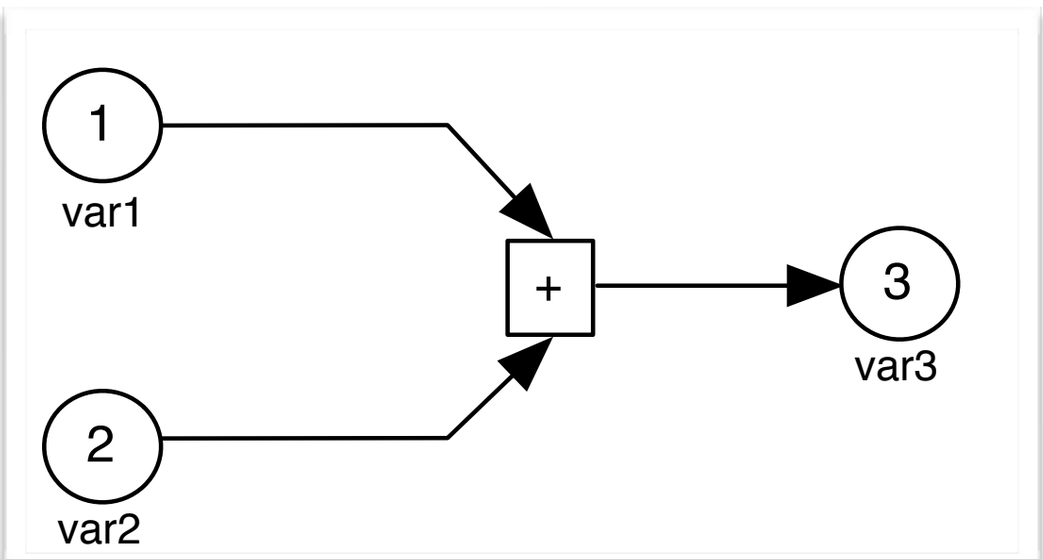
- automatically manage dependencies between such values

- abstract over time management

- *like spreadsheets*:
  change 1 cell => others are recalculated

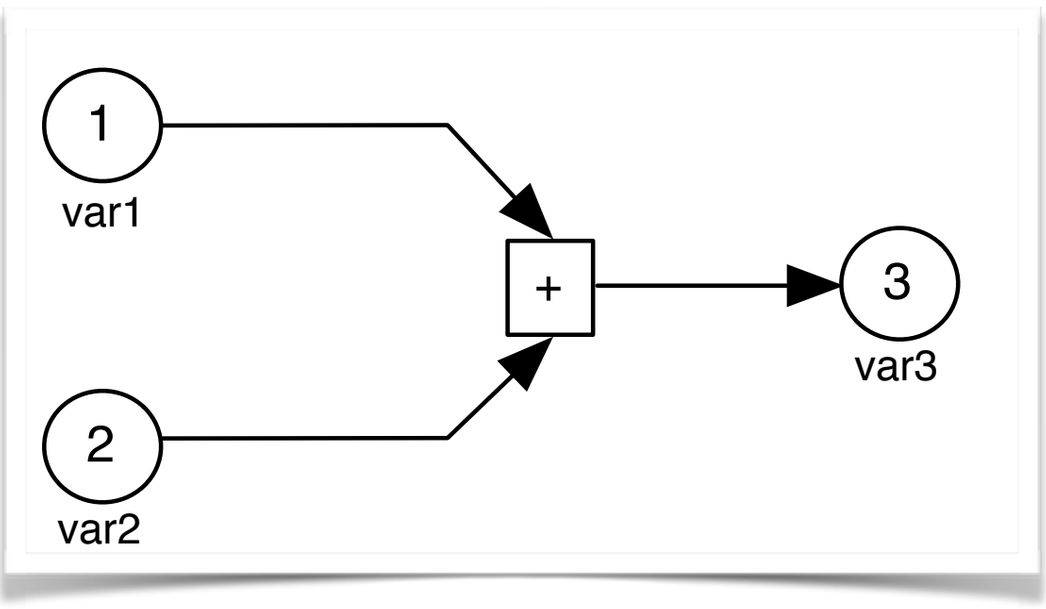
Example

```
var1 = 1
var2 = 2
var3 = var1 + var2
```



Example

```
var1 = 1
var2 = 2
var3 = var1 + var2
```




```
Stream s1 = new Stream(“1”);
Stream s2 = new Stream(“2”);
Stream s3 = Stream.add(s1,s2);
```

“Callback Hell” [Edw09]

- Lots of event handlers - asynchronous callbacks
- Manipulating the same data - unpredictable order
- No return value => update state via side-effects

The 6 dimensions

1. representation of time-varying values
2. evaluation model
3. lifting operations
4. multi-directionality
5. glitch avoidance
6. support for distribution

Conflicting

The 6 dimensions

1. representation of time-varying values
2. evaluation model
3. lifting operations
4. multi-directionality
5. glitch avoidance
6. support for distribution

Host languages:

- ✦ Haskell
- ✦ Scala
- ✦ Scheme/Racket
- ✦ JavaScript
- ✦ Java
- ✦ Python
- ✦ C#.Net
- ✦ …

1. Basic abstractions

What is manipulated?

Behaviour

- ✦ time-varying values
- ✦ continuously changing over time
- ✦ e.g.: `“seconds”` `“seconds*10”`

Events

- ✦ (maybe infinite) streams of values
- ✦ discrete point in time
- ✦ e.g.: `“key-press”` `“merge”` `“filter”`

Language	Basic abstractions	Evaluation model	Lifting	Multidirectionality
FRP Siblings				
Fran	behaviours and events	Pull	Explicit	N
Yampa	signal functions and events	Pull	Explicit	N
FrTime	behaviours and events	Push	Implicit	N
NewFran	behaviours and events	Push and Pull	Explicit	N
Frappé	behaviours and events	Push	Explicit	N
Scala.React	signals and events	Push	Manual	N
Flapjax	behaviours and events	Push	Explicit and implicit	N
AmbientTalk/R	behaviours and events	Push	Implicit	N
Cousins of Reactive Programming				
Cells	rules, cells and observers	Push	Manual	N
Lamport Cells	reactors and reporters	Push and Pull	Manual	N
SuperGlue	signals, components, and rules	Push	Manual	N
Trellis	cells and rules	Push	Manual	N
Radul/Sussman Propagators	propagators and cells	Push	Manual	Y
Coherence	reactions and actions	Pull	N/A	Y
.NET Rx	events	Push	Manual	N

2. Evaluation model

Who triggers sending of messages?

Pull-based - good for continuous streams

- ✦ consumer asks for value
- ✦ like a method call
- ✦ demand-driven propagation
- ✦ result of lazy evaluation (e.g., in Haskell)

Push-based - good for discrete events

- ✦ producer pushes data based on availability
- ✦ data-driven propagation
- ✦ followed by most recent implementations

Language	Basic abstractions	Evaluation model	Lifting	Multidirectionality	Glitch avoidan
FRP Siblings					
Fran	behaviours and events	Pull	Explicit	N	Y
Yampa	signal functions and events	Pull	Explicit	N	Y
FrTime	behaviours and events	Push	Implicit	N	Y
NewFran	behaviours and events	Push and Pull	Explicit	N	Y
Frappé	behaviours and events	Push	Explicit	N	N
Scala.React	signals and events	Push	Manual	N	Y
Flapjax	behaviours and events	Push	Explicit and implicit	N	Y (local)
AmbientTalk/R	behaviours and events	Push	Implicit	N	Y (local)
Cousins of Reactive Programming					
Cells	rules, cells and observers	Push	Manual	N	Y
Lamport Cells	reactors and reporters	Push and Pull	Manual	N	N
SuperGlue	signals, components, and rules	Push	Manual	N	Y
Trellis	cells and rules	Push	Manual	N	Y*
Radul/Sussman Propagators	propagators and cells	Push	Manual	Y	N
Coherence	reactions and actions	Pull	N/A	Y	Y
.NET Rx	events	Push	Manual	N	N?

3. Lifting

```
add : (Int,Int) -> Int
```

```
lift(add) :
  (Stream<Int>,Stream<Int>)
  -> Stream<Int>
```

- ✦ Map operations to all elements of the streams
- ✦ Registers a dependency graph

(In the paper: Stream <-> Behaviour/Events)

3. Lifting

Explicit

```
lift(add) :
  (Stream<Int>,Stream<Int>)
  -> Stream<Int>
```

Implicit

```
add : (Int,Int) -> Int
// this works automatically!
s3 = add(stream1,stream2)
```

Manual

```
x = add(get(stream1),get(stream2))
```

3. Lifting

method overloading (define **add** for Int and Stream) still counts as explicit

Explicit

in dynamically typed languages

```
lift(add) :
  (Stream<Int>,Stream<Int>)
  -> Stream<Int>
```

Implicit

```
add : (Int,Int) -> Int
// this works automatically!
s3 = add(stream1,stream2)
```

Manual

```
x = add(get(stream1),get(stream2))
```

Language	Basic abstractions	Evaluation model	Lifting	Multidirectionality	Glitch avoidan
FRP Siblings					
Fran	behaviours and events	Pull	Explicit	N	Y
Yampa	signal functions and events	Pull	Explicit	N	Y
FrTime	behaviours and events	Push	Implicit	N	Y
NewFran	behaviours and events	Push and Pull	Explicit	N	Y
Frappé	behaviours and events	Push	Explicit	N	N
Scala.React	signals and events	Push	Manual	N	Y
Flapjax	behaviours and events	Push	Explicit and implicit	N	Y (local)
AmbientTalk/R	behaviours and events	Push	Implicit	N	Y (local)
Cousins of Reactive Programming					
Cells	rules, cells and observers	Push	Manual	N	Y
Lamport Cells	reactors and reporters	Push and Pull	Manual	N	N
SuperGlue	signals, components, and rules	Push	Manual	N	Y
Trellis	cells and rules	Push	Manual	N	Y*
Radul/Sussman Propagators	propagators and cells	Push	Manual	Y	N
Coherence	reactions and actions	Pull	N/A	Y	Y
.NET Rx	events	Push	Manual	N	N?

4. Multidirectionality

- updates in both directions

```
F = (C * 1.8) + 32
```

	Multidirectionality	
	N	
	N	
	N	
	N	
	N	
	N	
	N	
	N	
	N	
	N	
	N	
	N	
	N	
	Y	←
	Y	←
	N	

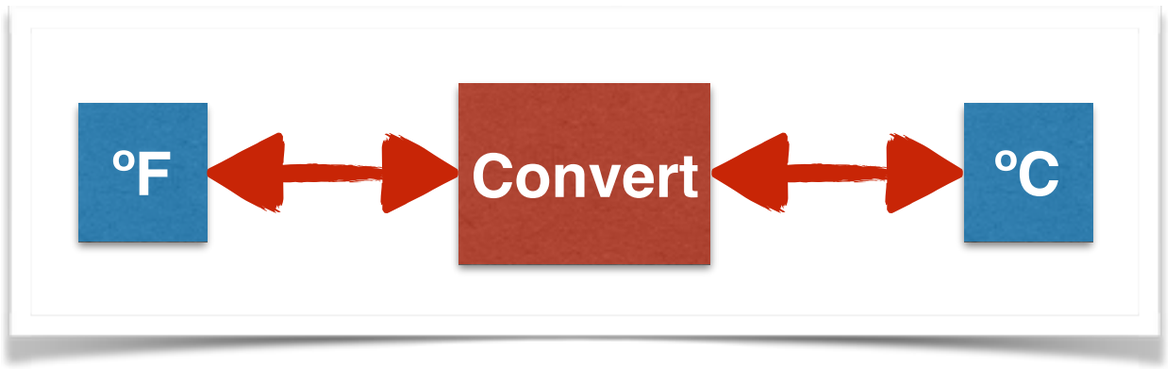
(scheme)

Radul/Sussman Propagators
Coherence

4. Multidirectionality

- updates in both directions

```
F = (C * 1.8) + 32
```



(scheme)

Radul/Sussman Propagators
Coherence

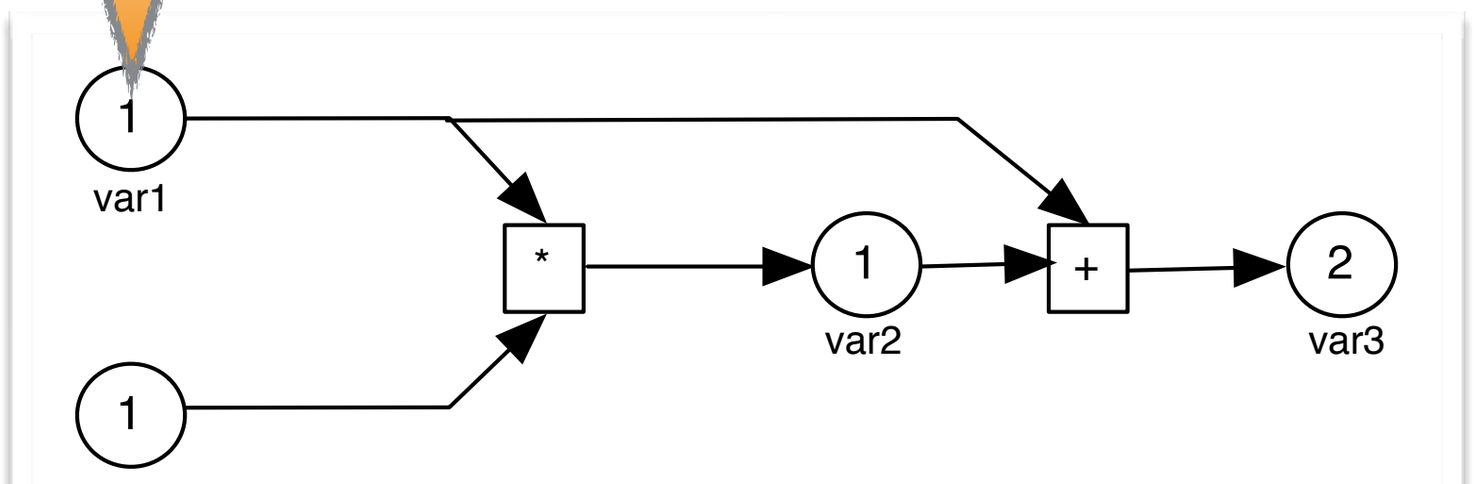
Multidirectionality
N
N
N
N
N
N
N
N
N
N
N
N
Y
Y
N

5. Glitches

"Momentary view of inconsistent data"

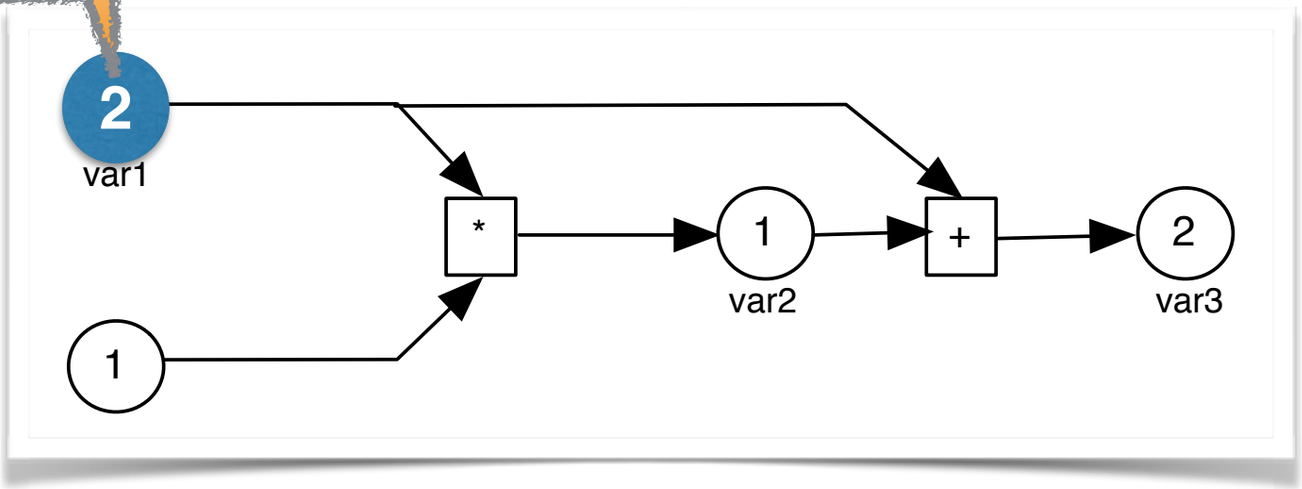
Change to **"2"**

```
var1 = 1
var2 = var1 * 1
var3 = var1 + var2
```

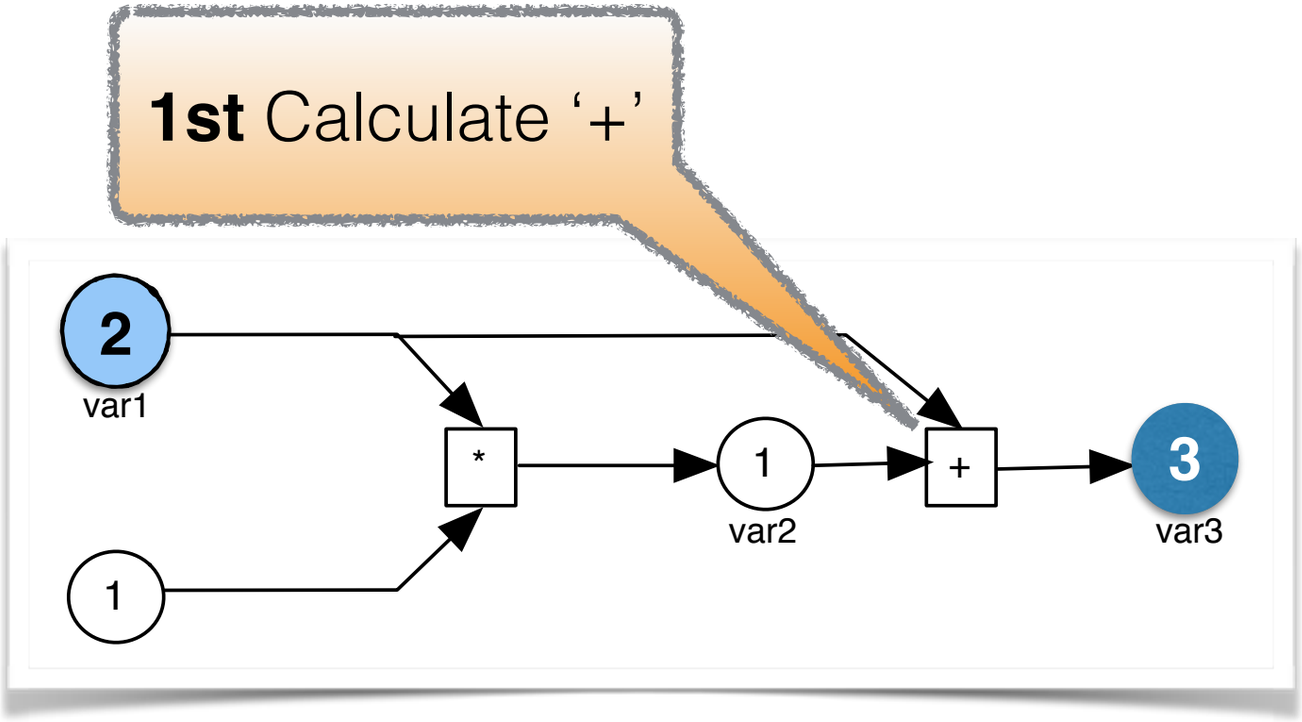


5. Glitches

Change to **“2”**

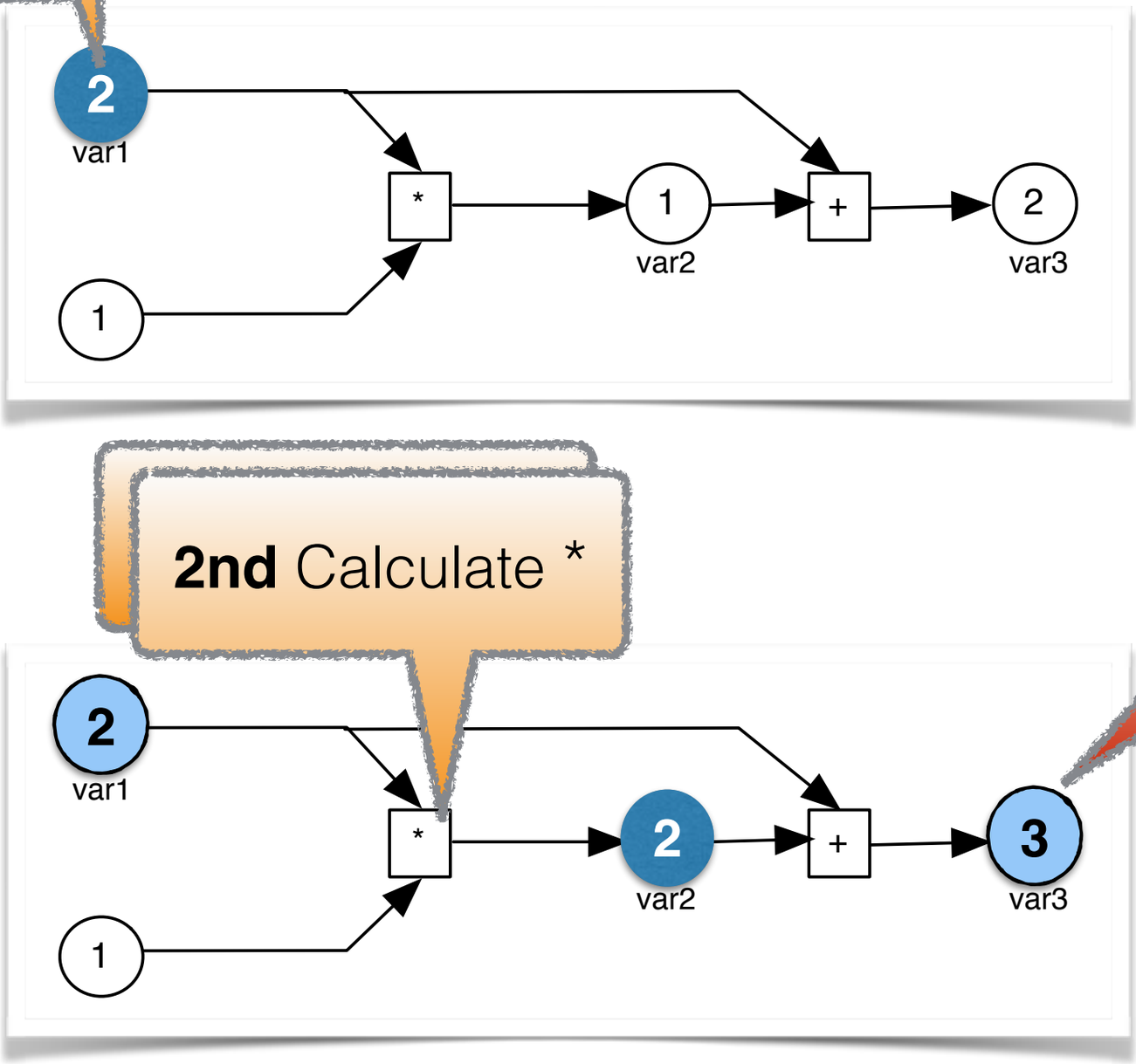


1st Calculate ‘+’



5. Glitches

Change to **“2”**

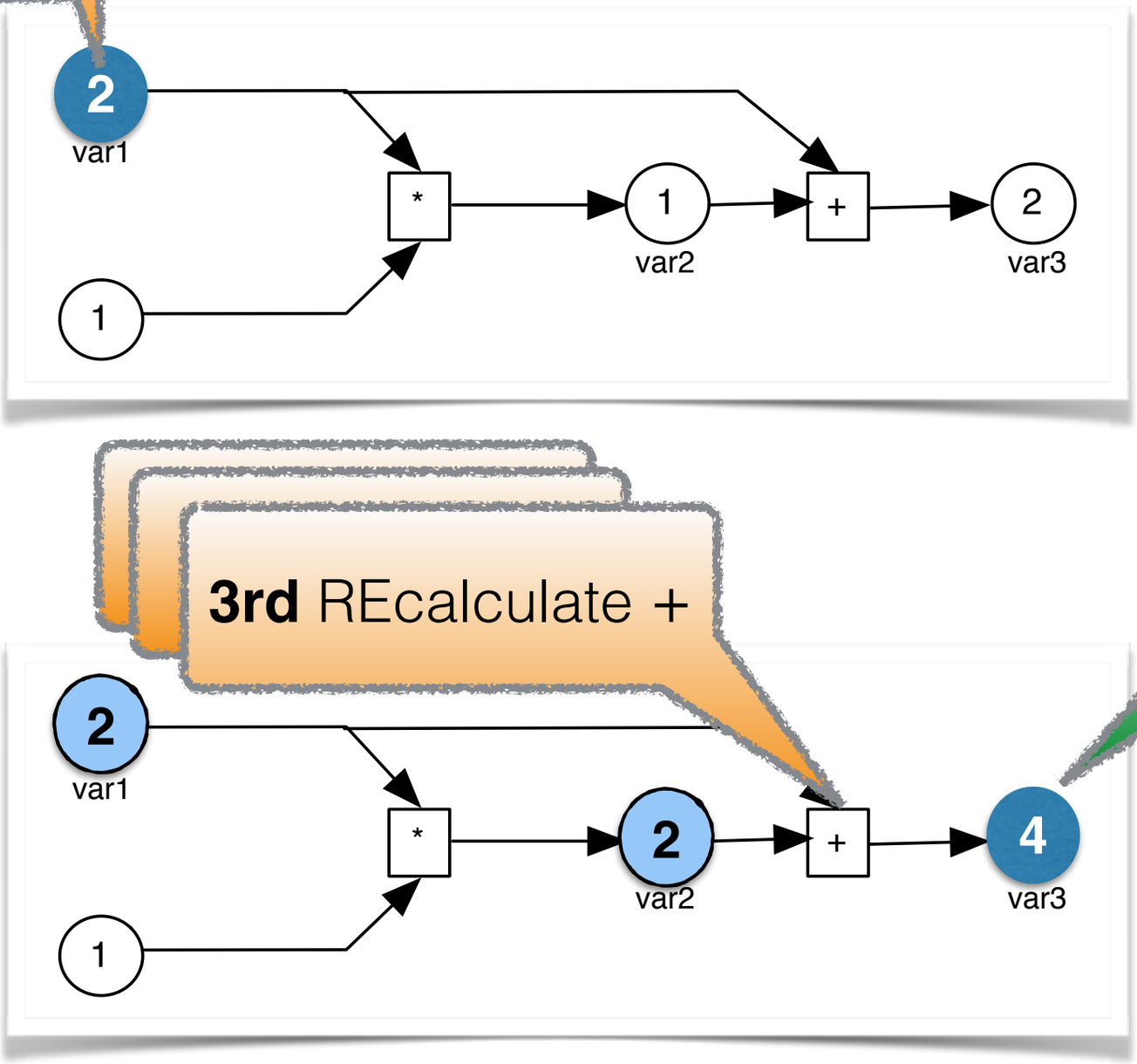


2nd Calculate *

WRONG!

5. Glitches

1. Change to **“2”**

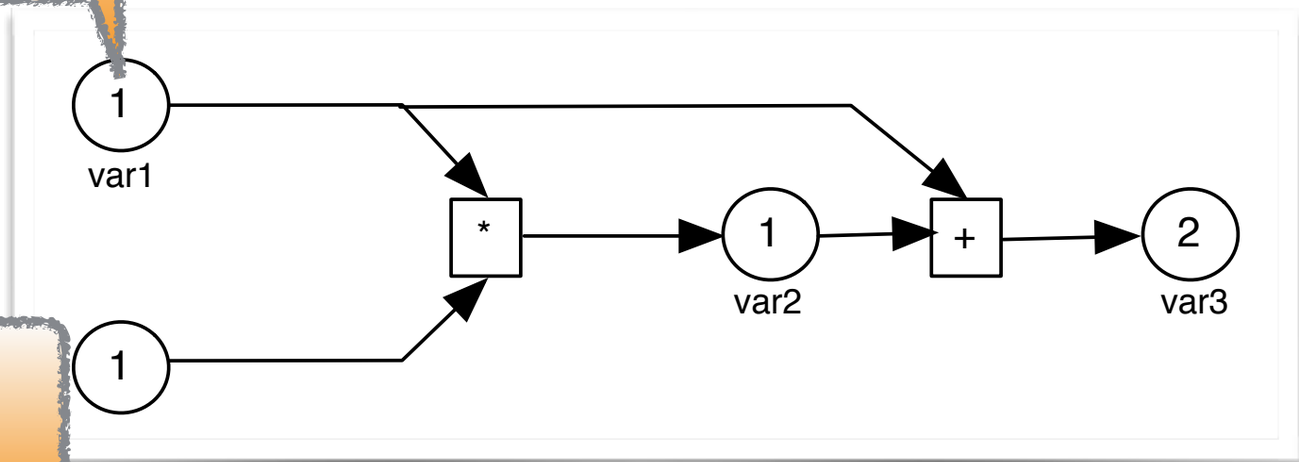


3rd REcalculate +

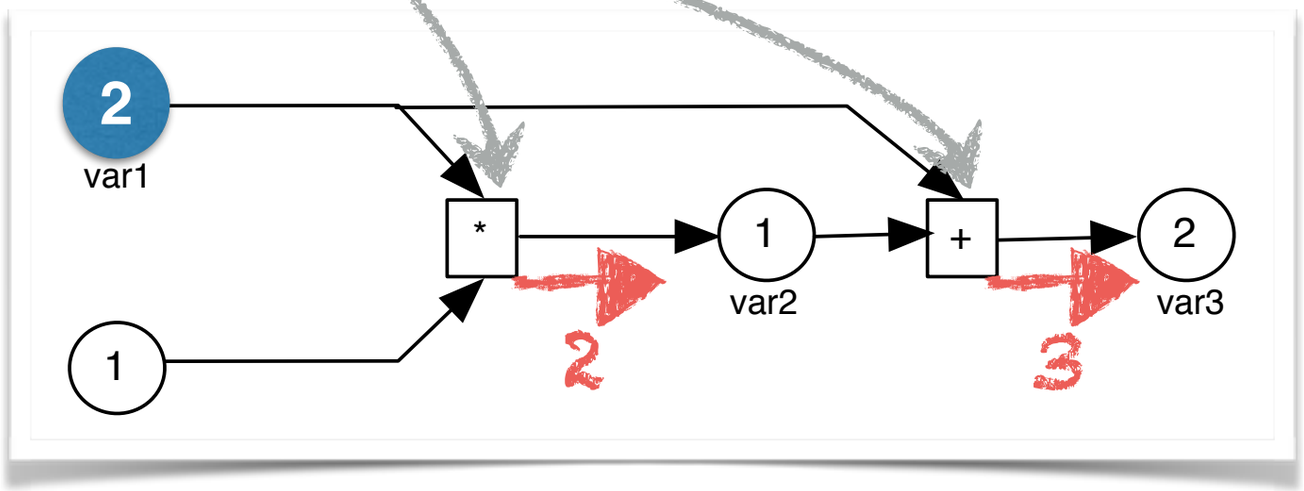
OK (now!)

5. Glitches (distributed view)

1. Change to **“2”**

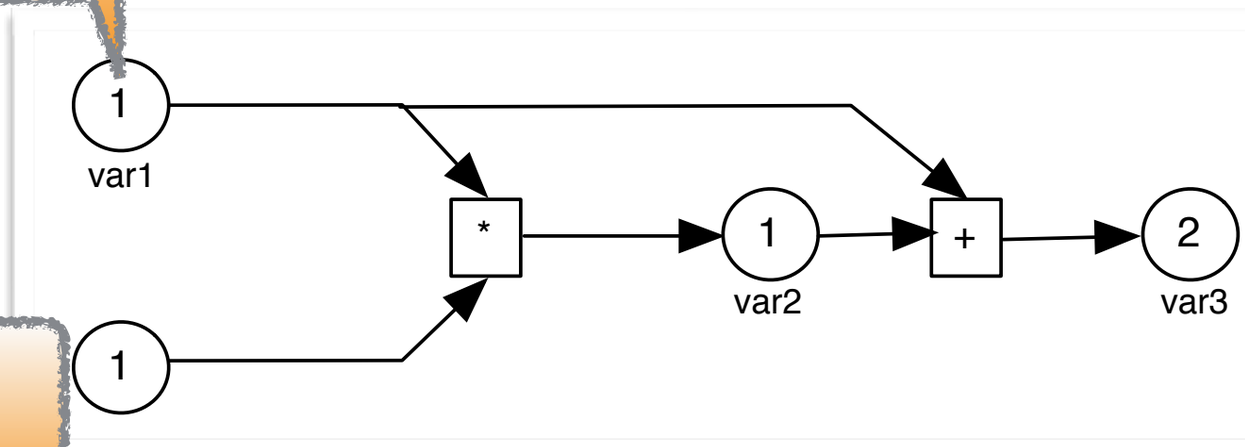


2. Calculate

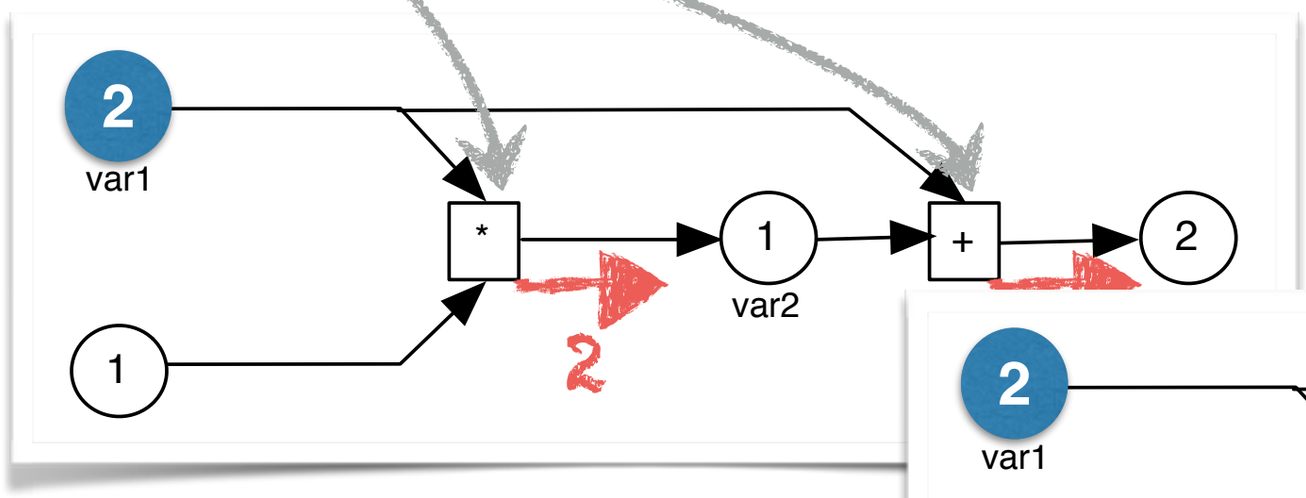


5. Glitches (distributed view)

1. Change to **“2”**

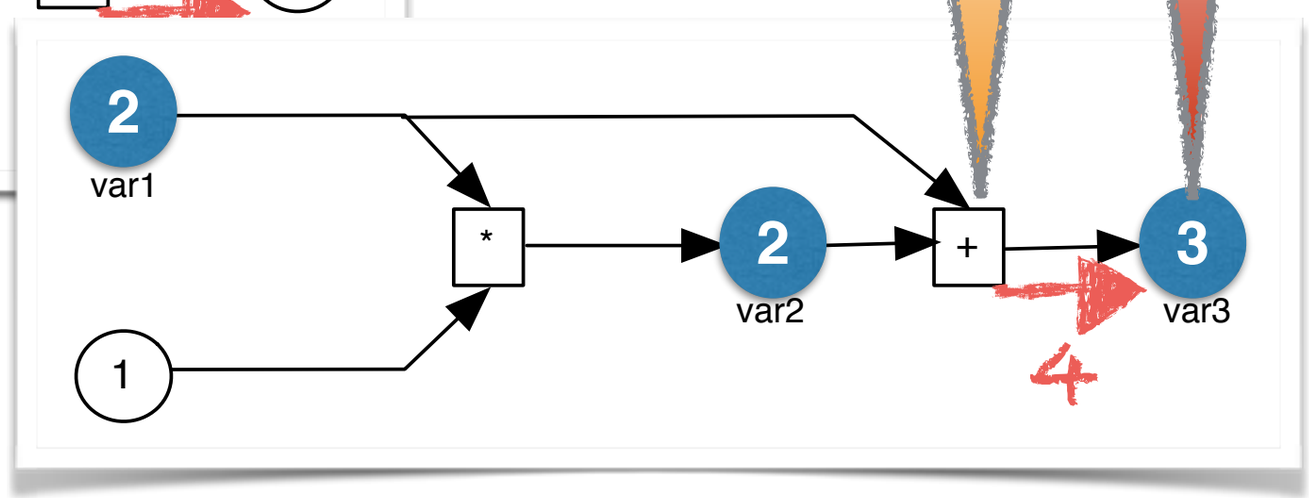


2. Calculate

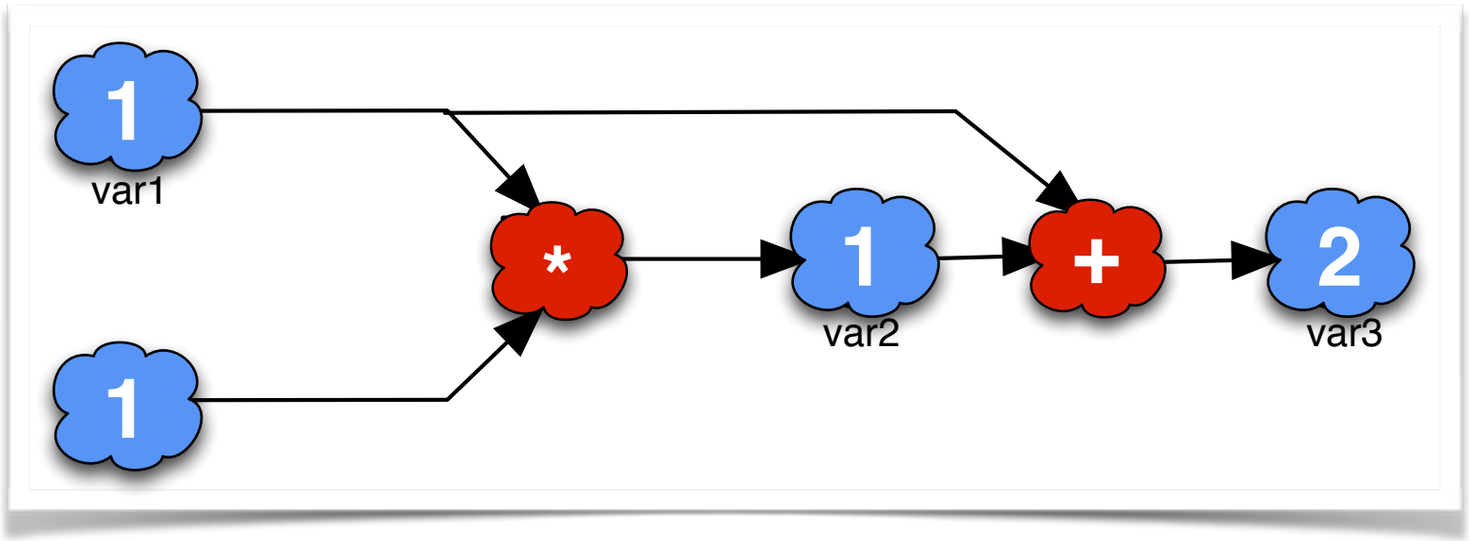


3. Recalculate

WRONG!



6. Distribution



- Operations in different network nodes
- hard to ensure consistency
- (latency, network failures, etc.)

Language	Lifting	Multidirectionality	Glitch avoidance	Support for distribution
FRP Siblings				
Fran	Explicit	N	Y	N
Yampa	Explicit	N	Y	N
FrTime	Implicit		Y	N
NewFran	Explicit		Y	N
Frappé	Explicit		N	N
Scala.React	Manual		Y	N
Flapjax	Explicit a implicit		Y (local)	Y
AmbientTalk/R	Implicit	N	Y (local)	Y
Cousins of Reac				
Cells	Manual	N	Y	N
Lamport Cells	Manu		N	Y
SuperGlue	Manu		Y	N
Trellis	Manu		Y*	N
Radul/Sussman Propagators	Manual	Y	N	N
Coherence	N/A		Y	N
.NET Rx	Ma		N?	N

within each node

extra care by developers

authors are not sure…

Going back to abstractions…

What is manipulated:
Behaviour (continuous) vs. Events (discrete)

Siblings of RP - based on Fran

- ✦ about time-varying values (behaviour) and *lifting*

Cousins of RP - less "pure"

- ✦ about "containers" with dedicated code to manage dependencies.

Code examples

Language	Host language
Fran [Elliott and Hudak 1997]	Haskell
Yampa [Hudak et al. 2003]	Haskell
Frappé [Courtney 2001]	Java
FrTime [Cooper and Krishnamurthi 2006]	PLT Scheme (now known as Racket)
NewFran [Elliott 2009]	Haskell
Flapjax [Meyerovich et al. 2009]	JavaScript
Scala.React [Maier et al. 2010]	Scala
AmbientTalk/R [Carreton et al. 2010]	AmbientTalk

Siblings
(lifting)

Language	Host language
Cells [Tilton 2008]	CLOS
Lamport Cells [Miller 2003]	E
SuperGlue [McDirmid and Hsieh 2006]	Java
Trellis [Eby 2008]	Python
Radul/Sussman Propagators [Radul and Sussman 2009]	MIT/GNU Scheme
Coherence [Edwards 2009]	Coherence
.NET Rx [Hamilton and Dyer 2010]	C#.NET

Cousins
(dependencies)

Fran & Yampa (Haskell)

```
tempConverter :: Behavior Double
tempConverter = tempF
    where
      tempC = temp
      tempF = (tempC*1.8)+32
```

```
tempConverter = proc -> do
    tempC <- tempSF
    tempF <- (tempC*1.8)+32
returnA -< tempF
```

```
drawcircle :: ImageB
drawcircle = withColour colour circle
    where
    colour = stepper red (lbp -=> green .|. rbp -=> red)
```

```
drawCircle = proc input -> do
    lbpE      <- lbp -< input
    rbpE      <- rbp -< input
    redB      <- constantB red
    thecolour <- selectcolour (lbpE 'lmerge' rbpE)
    colour    <- rSwitch (redB thecolour)
returnA -< circle 0 0 1 1 colour
```

FrTime & FlapJax

(Racket & JavaScript)

language or library

```
(define (temp-converter)
  (let* ((tempC temperature)
         (tempF (+ (* tempC 1.8) 32)))
    tempF))
```

```
function tempConverter() {
  var temp = Temperature();
  var tempC = temp;
  var tempF = tempC * 1.8 + 32;
  insertValueB(tempC, "tempCtext", "innerHTML");
  insertValueB(tempF, "tempFtext", "innerHTML");
}

<body onLoad = "tempConverter()">
<div id= "tempCtext"> </div>
<div id= "tempFtext"> </div>
</body>
```

```
(define (drawcircle)
  (let ((radius 60)
        (colour (new-cell "red")))
    (map-e (lambda (e) (set-cell! colour "green")) left-clicks)
    (map-e (lambda (e) (set-cell! colour "red")) right-clicks)
    (display-shapes
     (list
      (make-circle mouse-pos radius colour)))))
```

```
//draw circle at (x,y) and paint it colour
function drawcircle(x, y, colour) {...};

//map button press to colour
function handleMouseEvent(evt) {...};

var buttonE = extractEventE(document,"mousedown");
var colourE = buttonE.mapE(handleMouseEvent);
var colourB = startsWith(colourE, "red");
var canvas = document.getElementById('draw');
drawcircle(mouseLeftB(canvas), mouseTopB(canvas), colourB);
```

extends JavaBeans

Frappé (Java)

```
Temperature temp = new Temperature();
Behavior tempC = FRPUtilities.makeBehavior(sched, temp,
                                           "currentTemp");
Behavior tempF = FRPUtilities.liftMethod(sched, temp,
          "temperatureConverter", new Behavior[]{tempC});
```

```
Drawable circle    = new ShapeDrawable(
                                  new Ellipse2D.Double(-1,-1,2,2));
FRPEventSource lbp = FRPUtilities.makeFRPEvent(sched,
                                  frame, "franMouse","lbp");
FRPEventSource rbp = FRPUtilities.makeFRPEvent(sched,
                                  frame, "franMouse","rbp");

FRPEventSource lbpgreen = new EventBind(sched, lbp,
             FRPUtilities.makeComputation(new ConstB(Colour.green)));
FRPEventSource rbpred = new EventBind(sched, rbp,
             FRPUtilities.makeComputation(new ConstB(Colour.red)));
FRPEventSource colourE = new EventMerge(sched,
                                        lbpgreen, rbpred);

Behavior colourB = new Switcher(sched,
                           new ConstB(Colour.red), colourE);
Behavior anim = FRPUtilities.liftMethod(sched,
                new ConstB(circle), "withColour",
                new Behavior[] {colourB});
```

AmbientTalk/R & Scala.React

```
def temperatureConverter := object: {
    def @Reactive temp := Temperature.new();
    def tempC := temp;
    def tempF := tempC * 1.8 + 32;
}
```

```
val tempC = Signal{ Temperature() }
val tempF = Signal{ tempC() * 1.8 + 32}

observe(tempC) { C =>
  // print on label
}
observe(tempF) { F =>
  // print on label
}
```

manual lifting

```
def drawCircle(circle) { ... };

def @Reactive circle := object: {
  def posx := 0;
  def posy := 0;
  def colour := Colour.red;
};

def circleEventSource := changes: circle;
circleEventSource.foreach: { |circle| drawCircle(circle);

def handleMouseClickEvent(e) {
  // Update the circle object's coordinates and colour given e.
};
```

```
val selectedcolour = mouseDown map {md =>
//transform button press events to colour signal
}
val colour = selectedcolour switchTo Signal{Colour.red}
observe(colour) { c =>
 // redraw circle
}
```

Cousins
(dependencies)

Language	Host language
Cells [Tilton 2008]	CLOS
Lamport Cells [Miller 2003]	E
SuperGlue [McDirmid and Hsieh 2006]	Java
Trellis [Eby 2008]	Python
Radul/Sussman Propagators [Radul and Sussman 2009]	MIT/GNU Scheme
Coherence [Edwards 2009]	Coherence
.NET Rx [Hamilton and Dyer 2010]	C#.NET

- Manual lifting
- no combinators (e.g., `merge`, `map-e`, `switch`)
- only `TempConverter` example

Cells

```
(defmodel TempConverter ()
  ((tempC :cell t
          :initform (c-in Temperature)
          :accessor tempC)
   (tempF :cell t
          :initform (c? (+ (* (^tempC) 1.8) 32))
          :accessor tempF)))
```

(Python)

Trellis

```
class TempConverter(trellis.Component):
    tempC = trellis.attr(Temperature)
    tempF = trellis.maintain(
        lambda self: self.tempC * 1.8 + 32,
        initially = 32
    )

    @trellis.perform
    def viewGUI(self):
         display "Celsius: ", self.tempC
         display "Fahrenheit: ", self.tempF
```

SuperGlue

```
atom Thermometer {
    export temp : Float;
}

atom Label {
     import tempCText : String;
     import tempFText : String;
}

let model = new Thermometer;
let view = new Label;
let tempF = (model.temp * 1.8) + 32;
view.tempCtext = "Celsius: " + model.temp;
view.tempFtext = "Fahrenheit: "+tempF;
```

(wraps Java)

Propagators

```
(define (temp-converter C F)
  (let ((nine/five  (make-cell))
        (C*9/5  (make-cell))
        (thirty-two (make-cell)))
    ((constant 1.8) nine/five)
    ((constant 32) thirty-two)
    (multiplier C nine/five C*9/5)
    (adder C*9/5 thirty-two F)))

(define tempC (make-cell Temperature))
(define tempF (make-cell))
(temp-converter tempC tempF)
```

(multidirectional)

Coherence

```
temperatureConverter: {
  tempC = Temperature,
  tempF = Sum(Product(tempC, 1.8), 32)}
```

(multidirectional)

Lamport Cells

```
def tempConverter(){
  def tempC := makeLamportSlot(Temperature);
  def tempF := whenever([tempC], fn{tempC * 1.8 + 32}, fn{true});
  return tempF;
}
```

(distributed)

.Net RX

```
var temperature = new Temperature();
temperature.Subscribe( temp =>
                       {
                         var tempC = temp.currentTemperature;
                         var tempF = (tempC*1.8)+32;
                       })
```

Other languages

Synchronous

- Reactions take no time – atomic
- Compilation = finite state machine
- E.g. Esterel, StateCharts, Politi, FairThreads

Dataflow

- program as a graph
- nodes: operations
- arcs: data dependencies
- E.g. LabVIEW, Simulink

Synchronous dataflow

- combination of the 2
- graph known at compile time (static scheduling)
- time plays a key role
- E.g. Lustre, Signal, [RT-FRP, E-FRP]

Open questions

Can multidirectionality be embedded in “sibling” RP?

- use of constraints to relate streams **vs.**
- explicitly define operations (first-class values)

Avoiding glitches in a distributed setting?

- Need for *time-stamping*
- Extra centralised clock? (not great)
- Use “ticks” – Guarantee all clocks do not deviate more than 1 tick-time

Handling network failure?

- suggestion: integrating publish/subscribe-style
- self-reference to extension of AmbientTalk/R

Next meetings

Christophe, 28 Feb:

Ingo Maier, Tiark Rompf, and Martin Odersky, **Deprecating the Observer Pattern with Scala.React**, Technical report, École Polytechnique Fédérale de Lausanne, 2010

Candidate papers

- Margara, A., & Salvaneschi, G., **We have a DREAM: distributed reactive programming with consistency guarantees**. In: ACM DEBS, 2014.
- Evan Czaplicki and Stephen Chong. **Asynchronous Functional Reactive Programming for GUIs**. In: ACM SIGPLAN PLDI, 2013.
- Meijer, E., **Reactive extensions (Rx): curing your asynchronous programming blues**, In: ACM SIGPLAN CUFP, 2010.
- Andoni L. Carreton, Stijn Mostinckx, Tom Van Cutsem, and Wolfgang De Meuter, **Loosely-coupled distributed reactive programming in mobile ad hoc networks**. In: TOOLS, 2010.
- Ankush Desai, Vivek Gupta, Ethan Jackson, Shaz Qadeer, Sriram Rajamani, and Damien Zufferey - **P: Safe Asynchronous Event-Driven Programming**, 2012 (referred to by Prof. Piessens with respect to state of the art in event-driven programming)
- Geoffrey Mainland Greg Morrisett Matt Welsh - **Flask: Staged Functional Programming for Sensor Networks**, 2008
- Frédéroc Boussinot - **Reactive C: An Extension of C to Program Reactive Systems** - 1991
- *Bob Reynders - FRP overview + previous and ongoing work*