QUARP:
Quality Aware Reactive Programming for the IoT

José Proença & Carlos Baquero

FSEN 2017

Universidade do Minho

3. Solution:
use thresholds

Quality Aware Reactive Programming for the IoT

2. Motivation:
RP challenges

1. context:
WSN applications

slides: http://quarp.proenca.org

IoT application
(a WSN perspective)

DistriNet

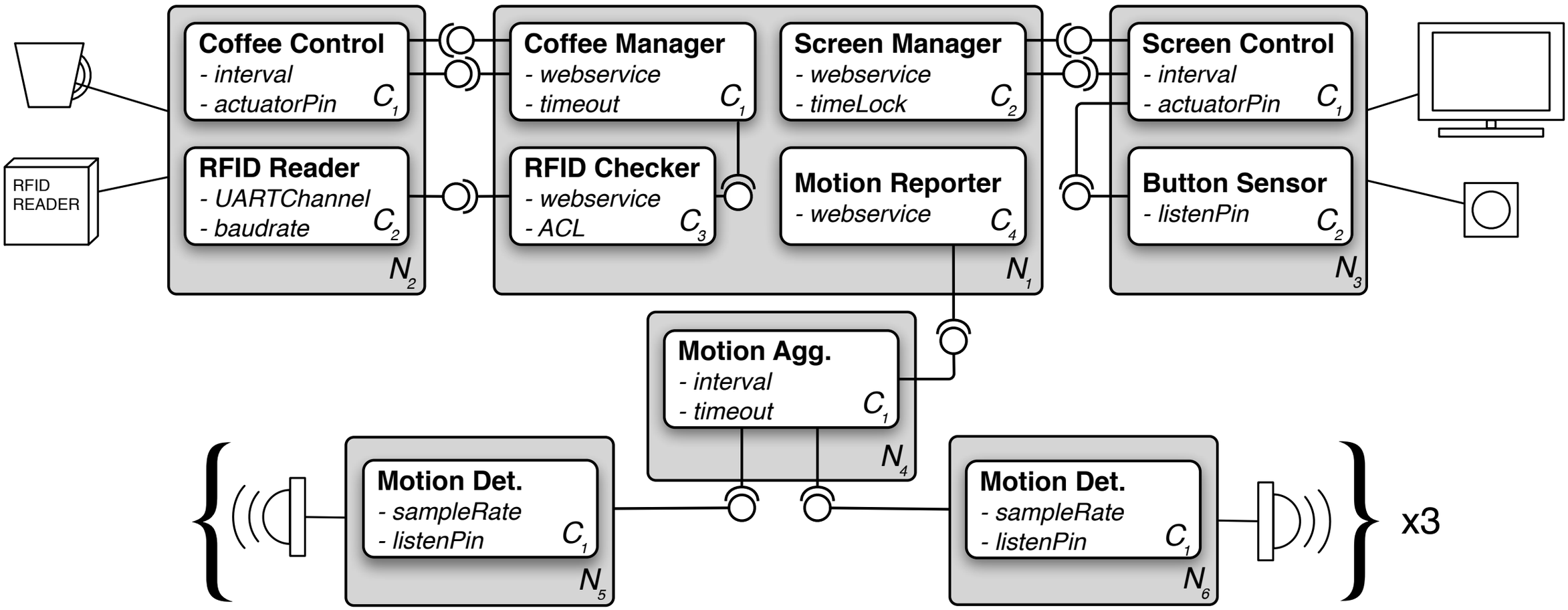
iMinds

KU LEUVEN

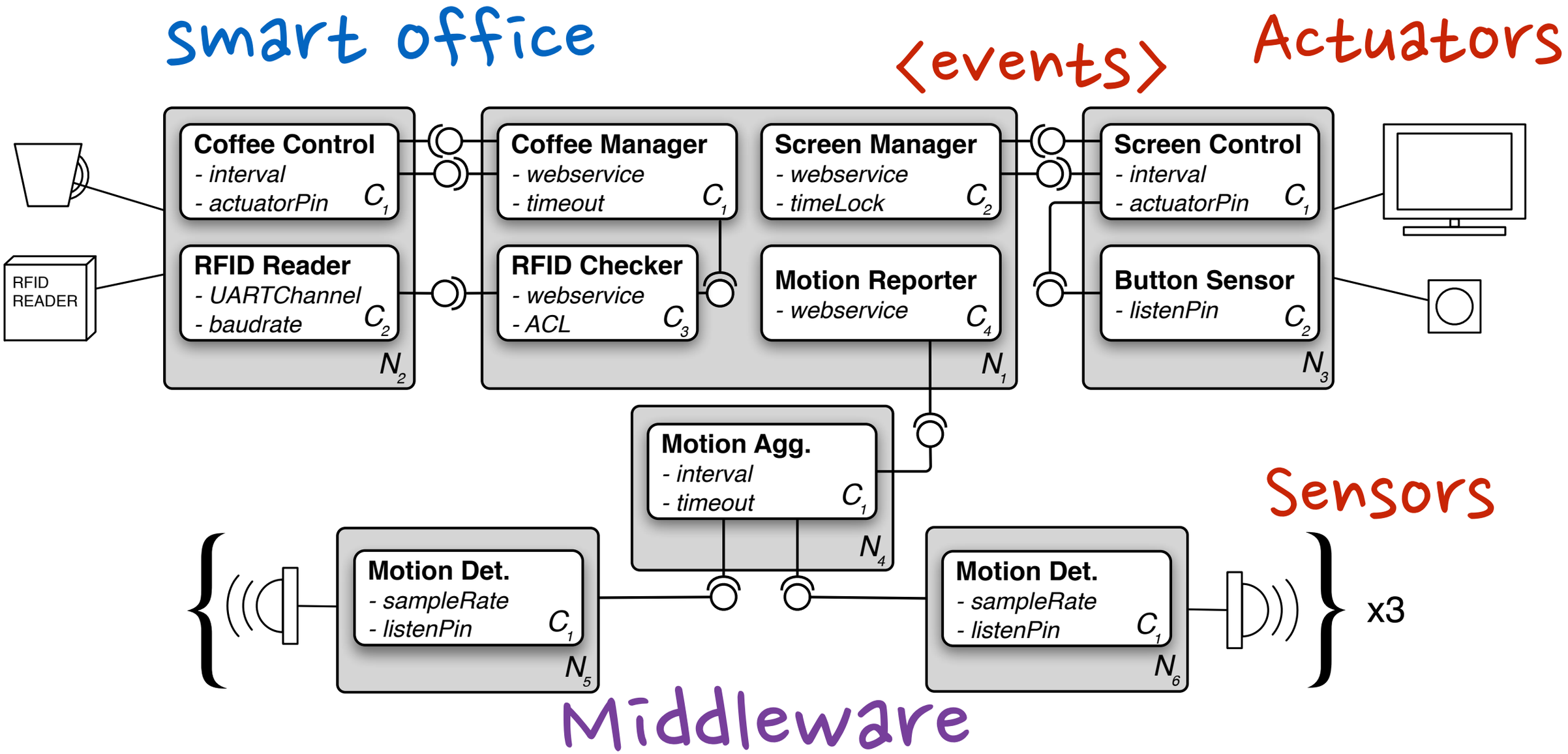
slides: http://quarp.proenca.org

IoT application
(a WSN perspective)

smart office

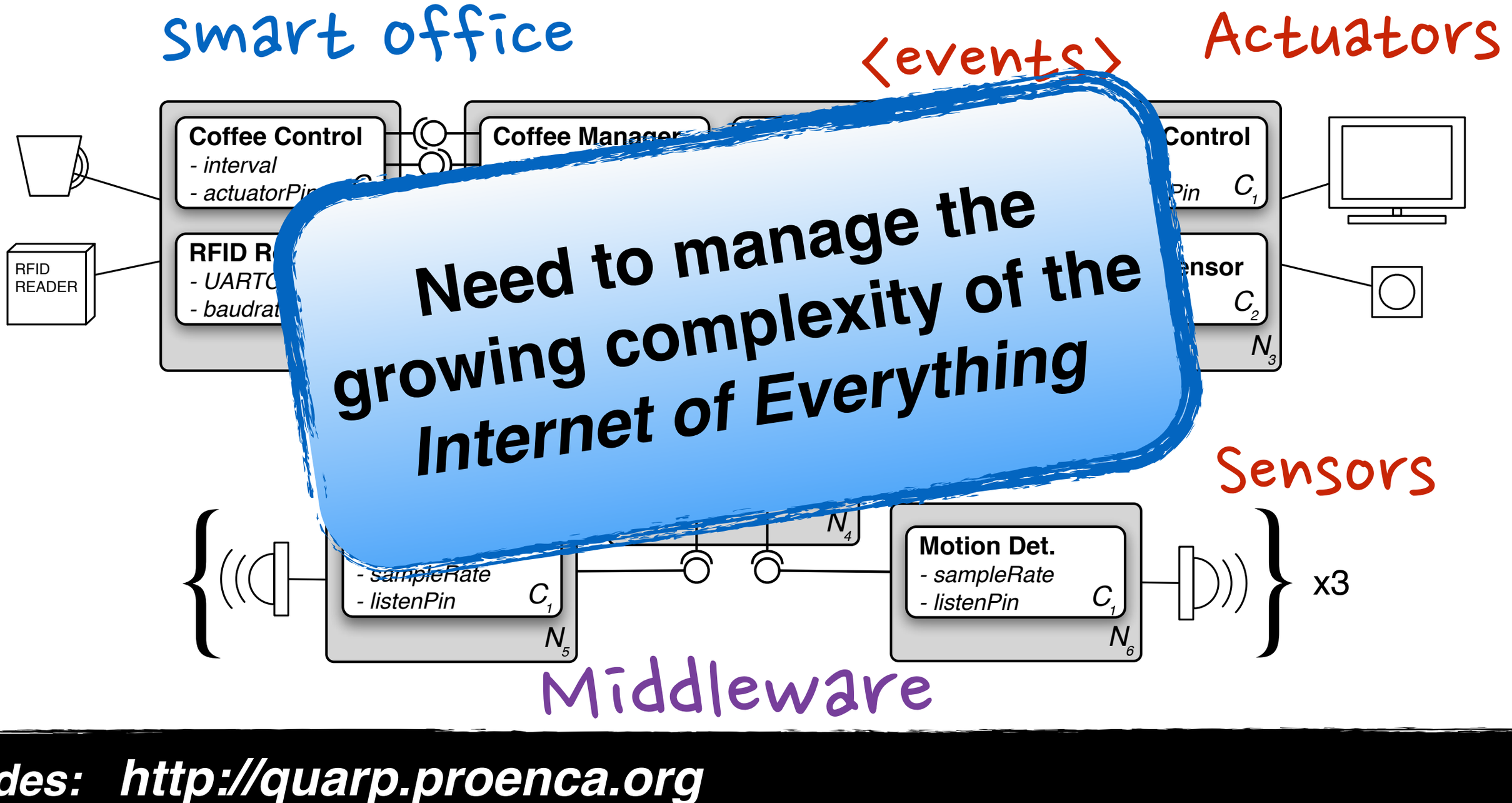


IoT application
(a WSN perspective)



slides: http://quarp.proenca.org

IoT application
(a WSN perspective)



slides: http://quarp.proenca.org

(Distributed)

3. Solution:
use thresholds

Quality Aware Reactive Programming for the IoT

2. Motivation:
RP challenges

1. context:
WSN applications

Reactive programming

- for event-driven and interactive applications

- express time-varying values

e.g., GUIs, web-apps

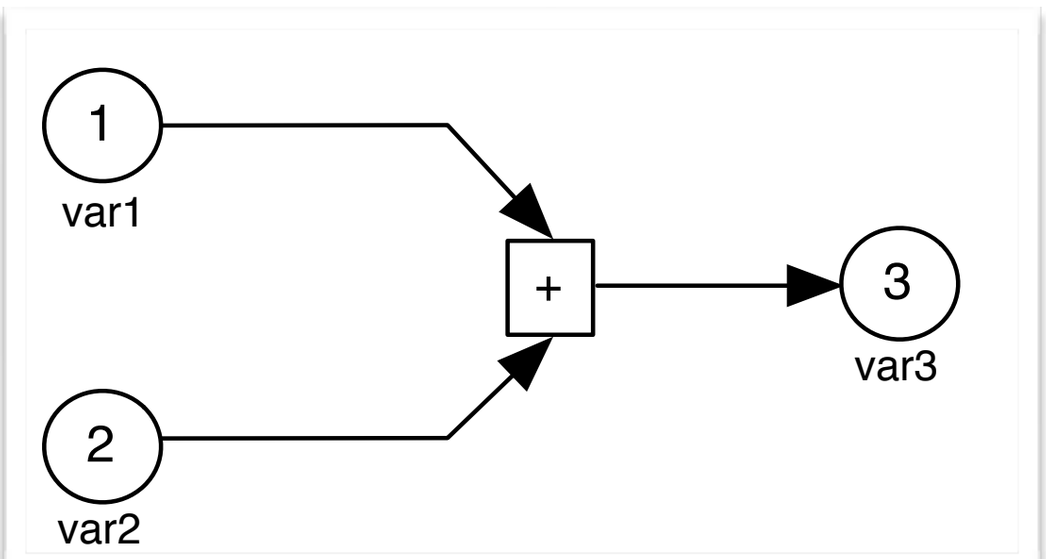
- automatically manage dependencies between such values

- abstract over time management

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

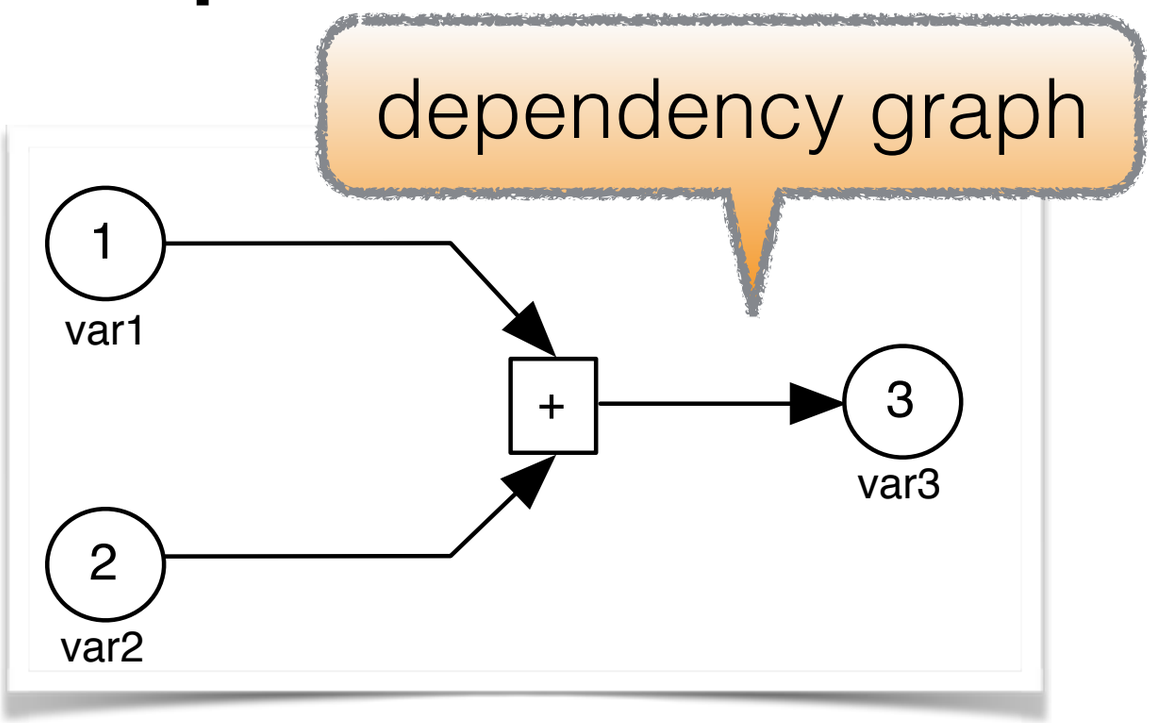
Example

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



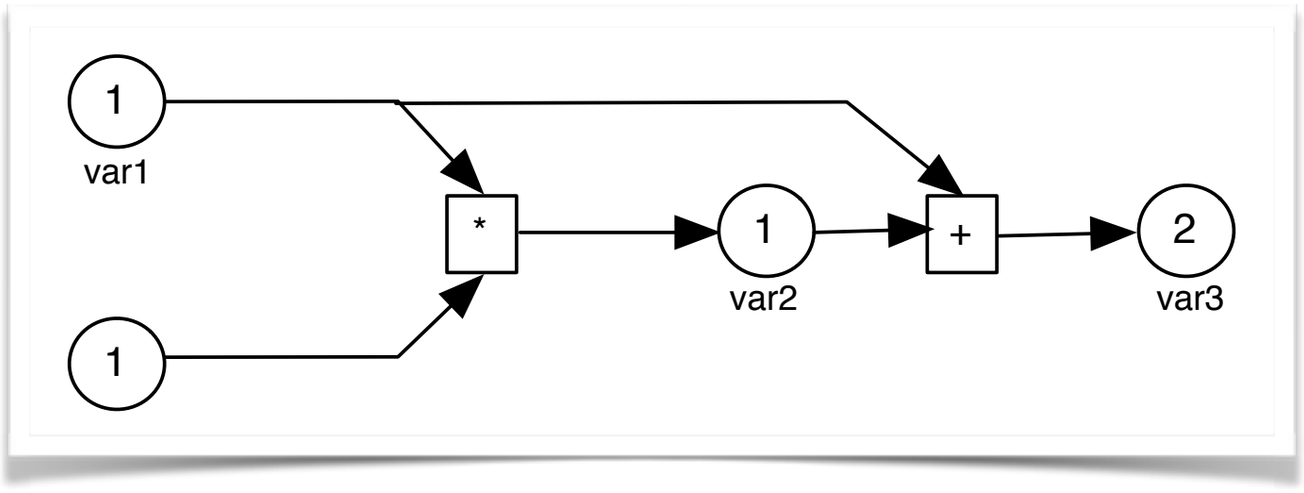
Example

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




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

Challenges



Push vs. Pull behaviour

“Lifting” operations

Order of evaluation

Distribution

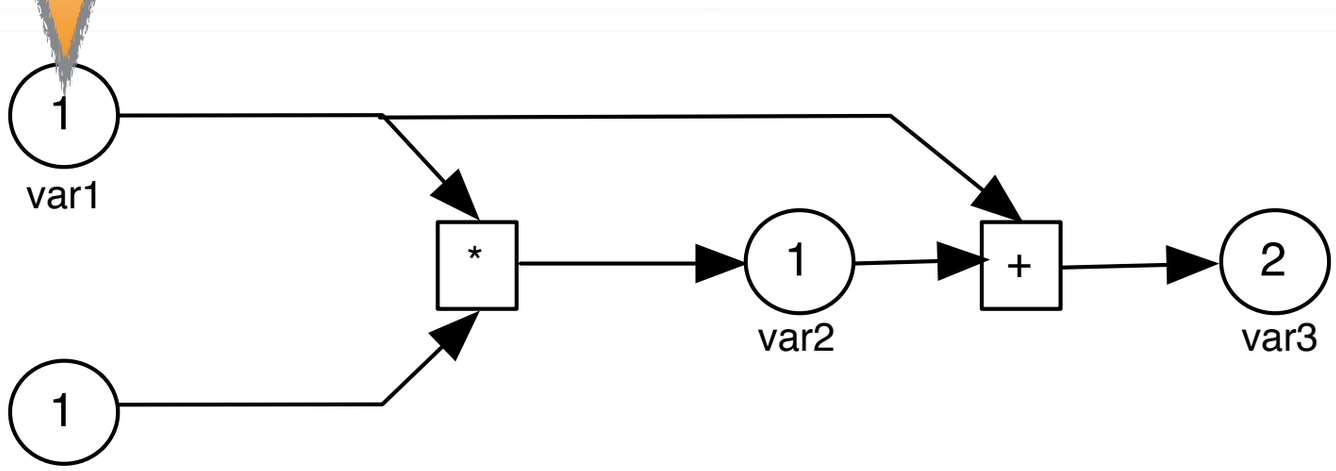
avoid “glitches”

Glitches

"Momentary view of inconsistent data"

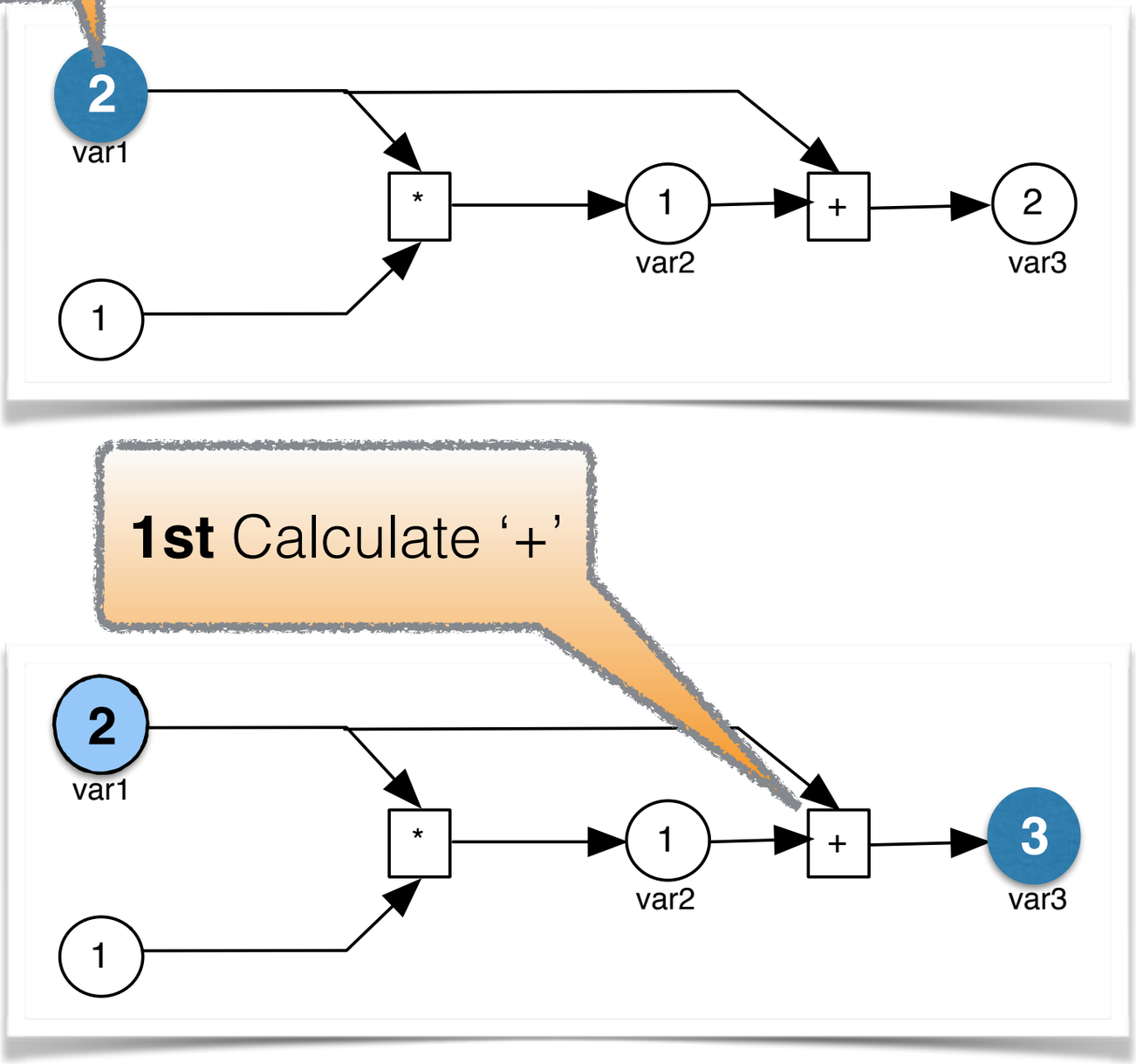
Change to **"2"**

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



Glitches

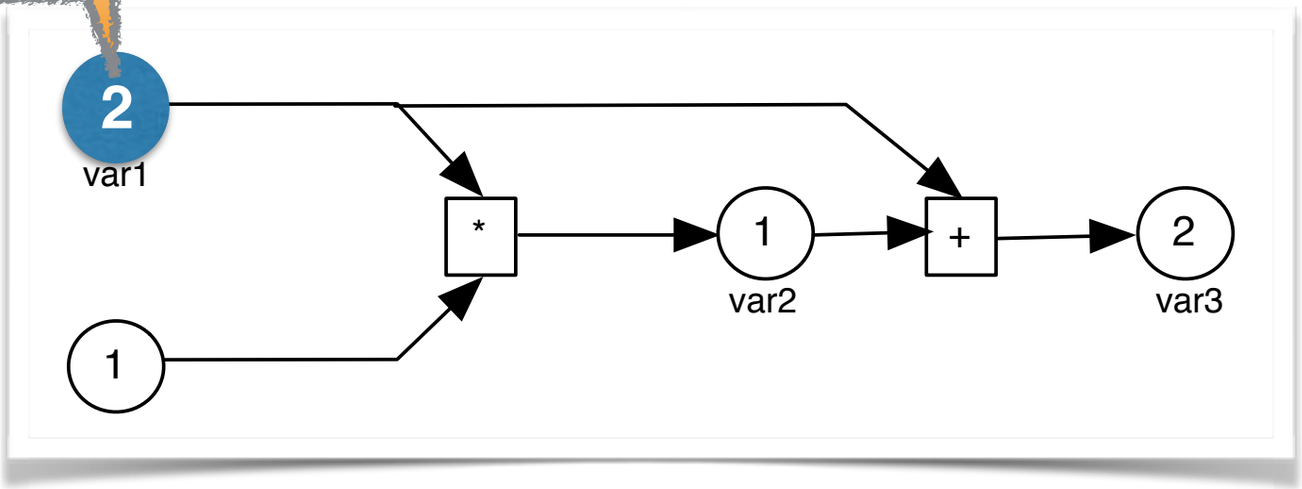
Change to **“2”**



1st Calculate ‘+’

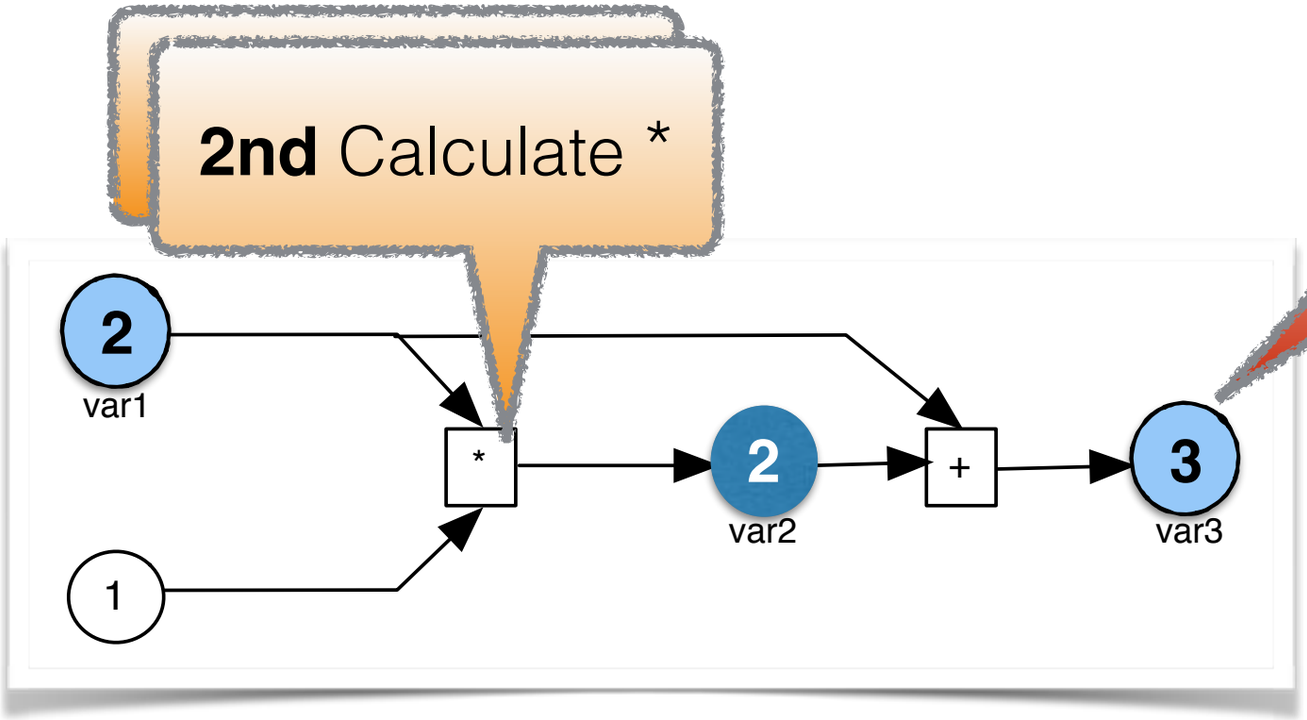
Glitches

Change to **“2”**



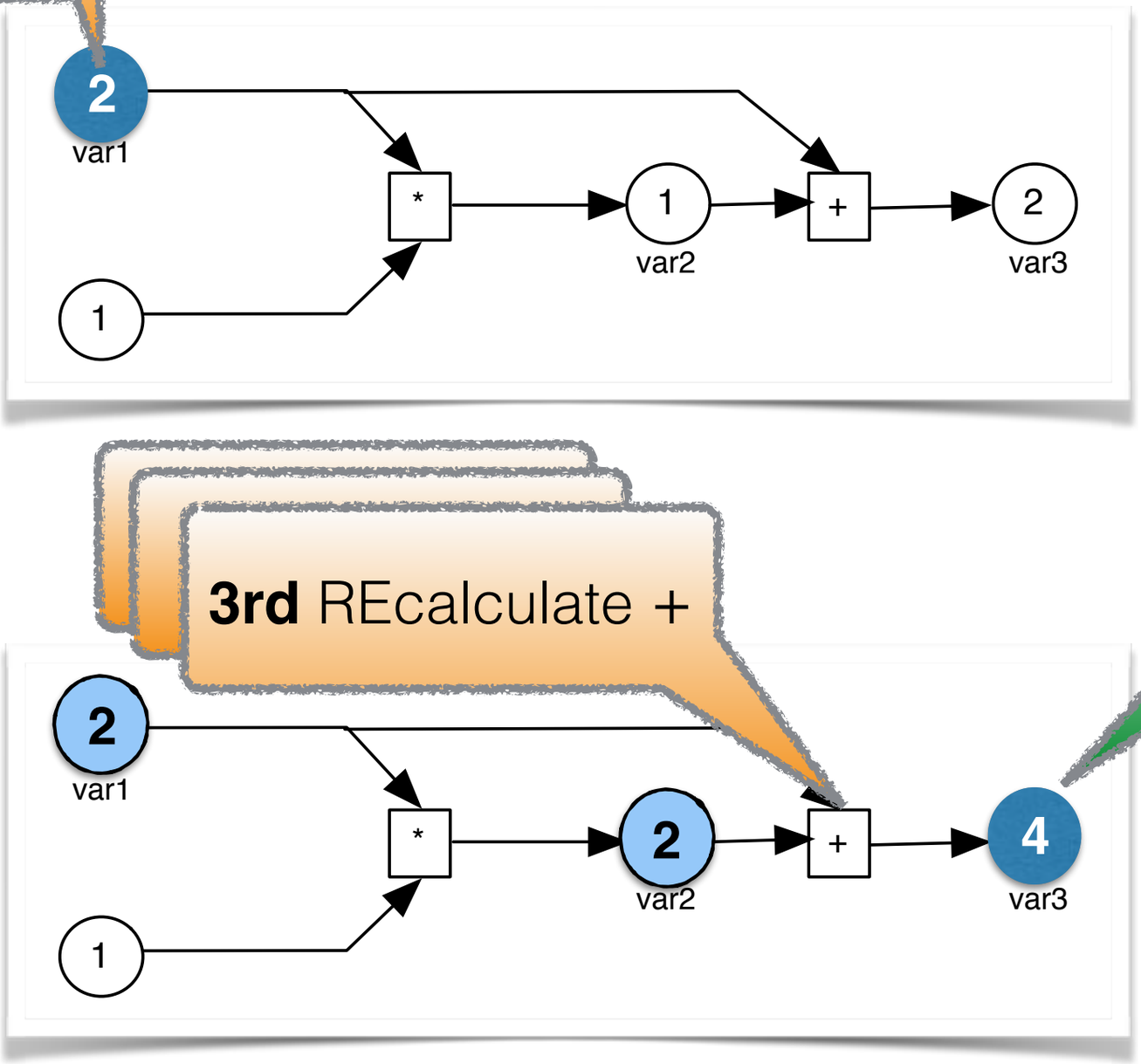
2nd Calculate *

WRONG!



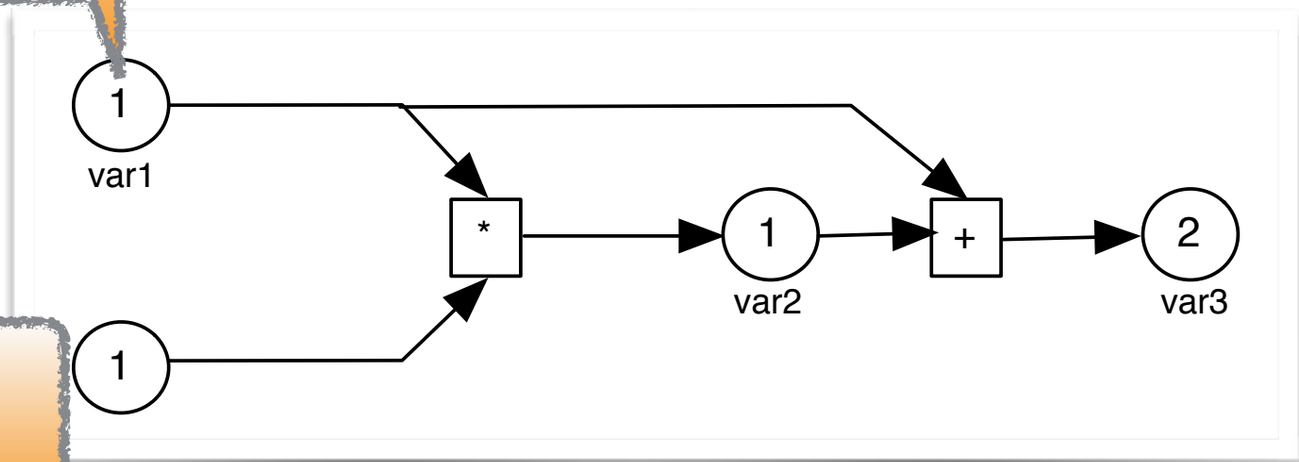
Glitches

1. Change to **“2”**

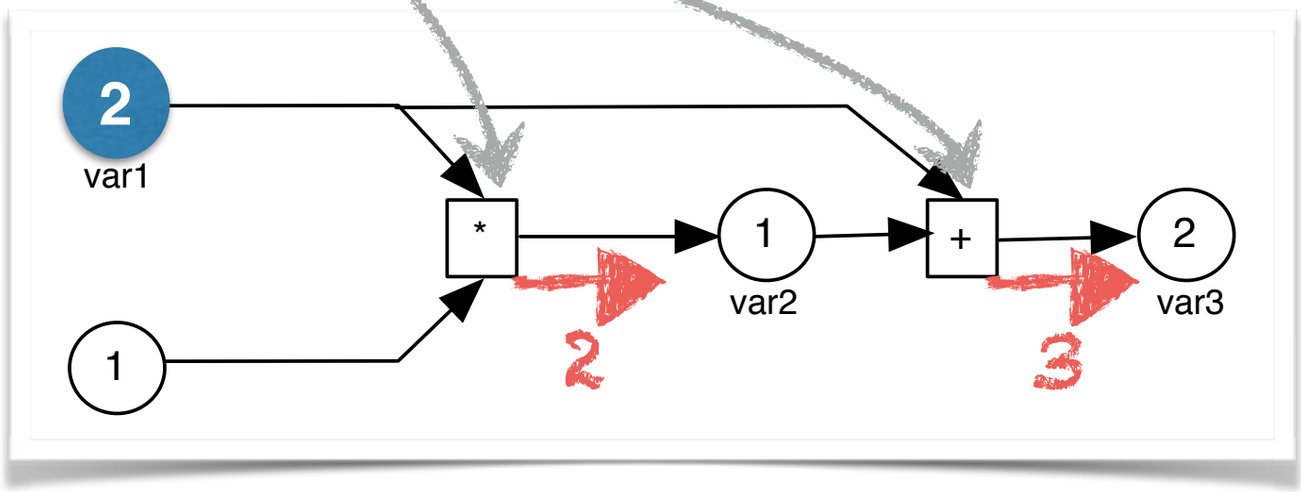


Glitches (distributed view)

1. Change to **“2”**

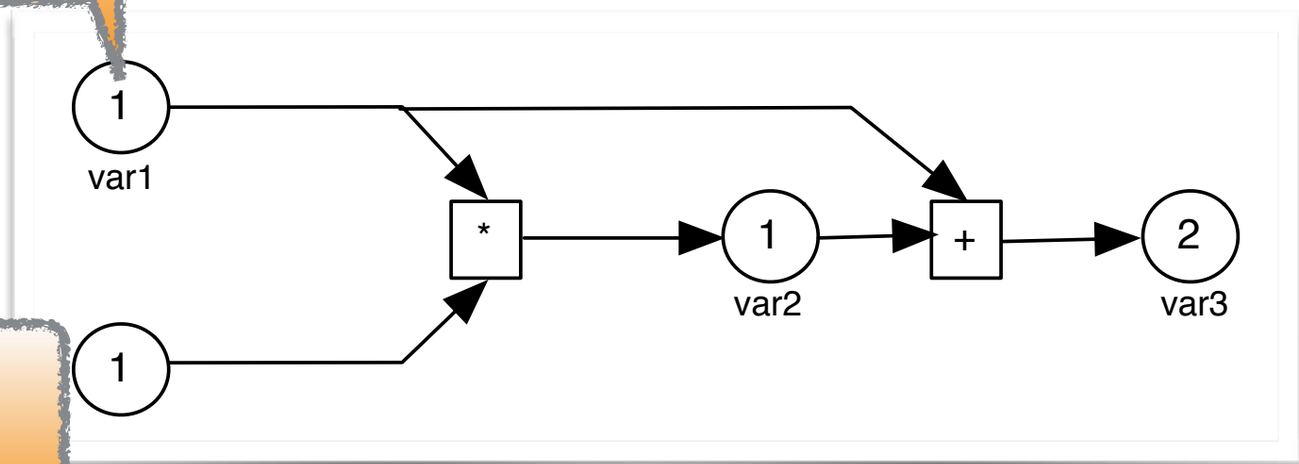


2. Calculate

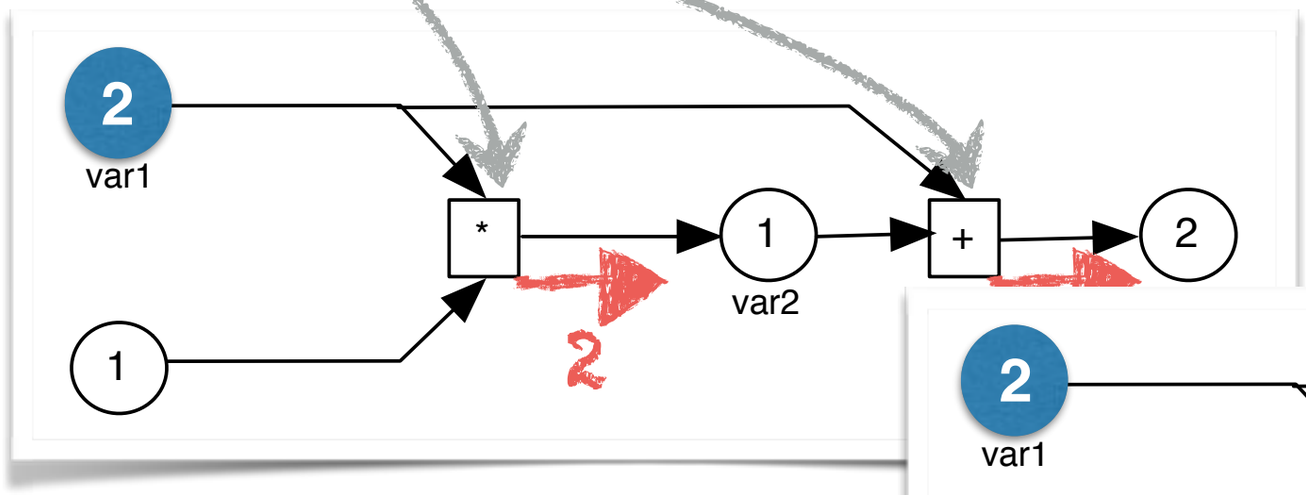


Glitches (distributed view)

1. Change to **“2”**

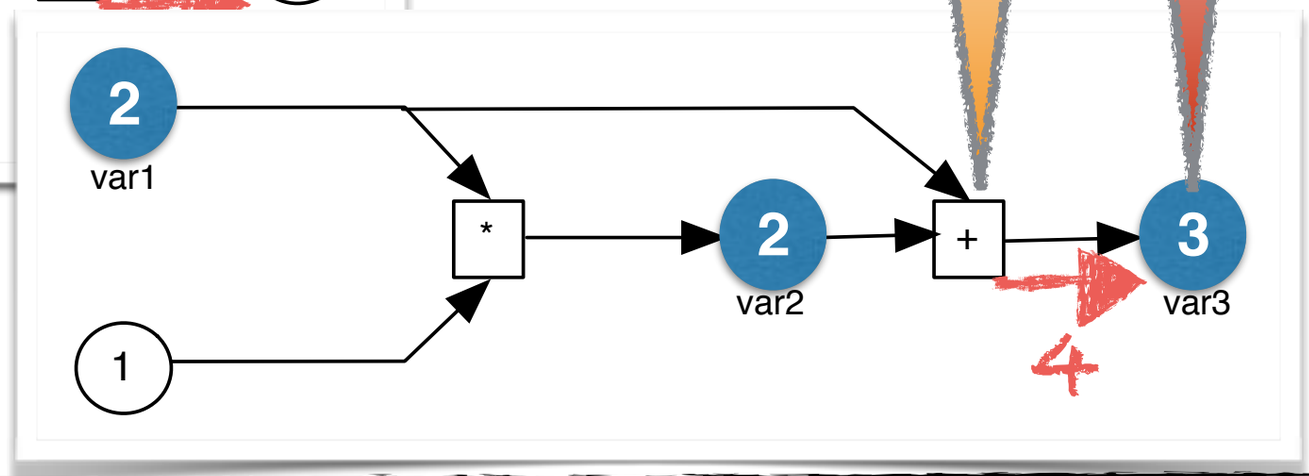


2. Calculate



3. Recalculate

WRONG!



slides: http://quarp.proenca.org

Distributed Reactive Programming

OOPSLA'14

Distributed REScala: An Update Algorithm for Distributed Reactive Programming

Joscha Drechsler,
Guido Salvaneschi
Technische Universität Darmstadt, Germany
< *lastname* >@cs.tu-darmstadt.de

Ragnar Mogk
Technische Universität Darmstadt, Germany
ragnar.mogk@stud.tu-darmstadt.de

Mira Mezini
Technische Universität Darmstadt, Germany; Lancaster University, UK
mezini@cs.tu-darmstadt.de

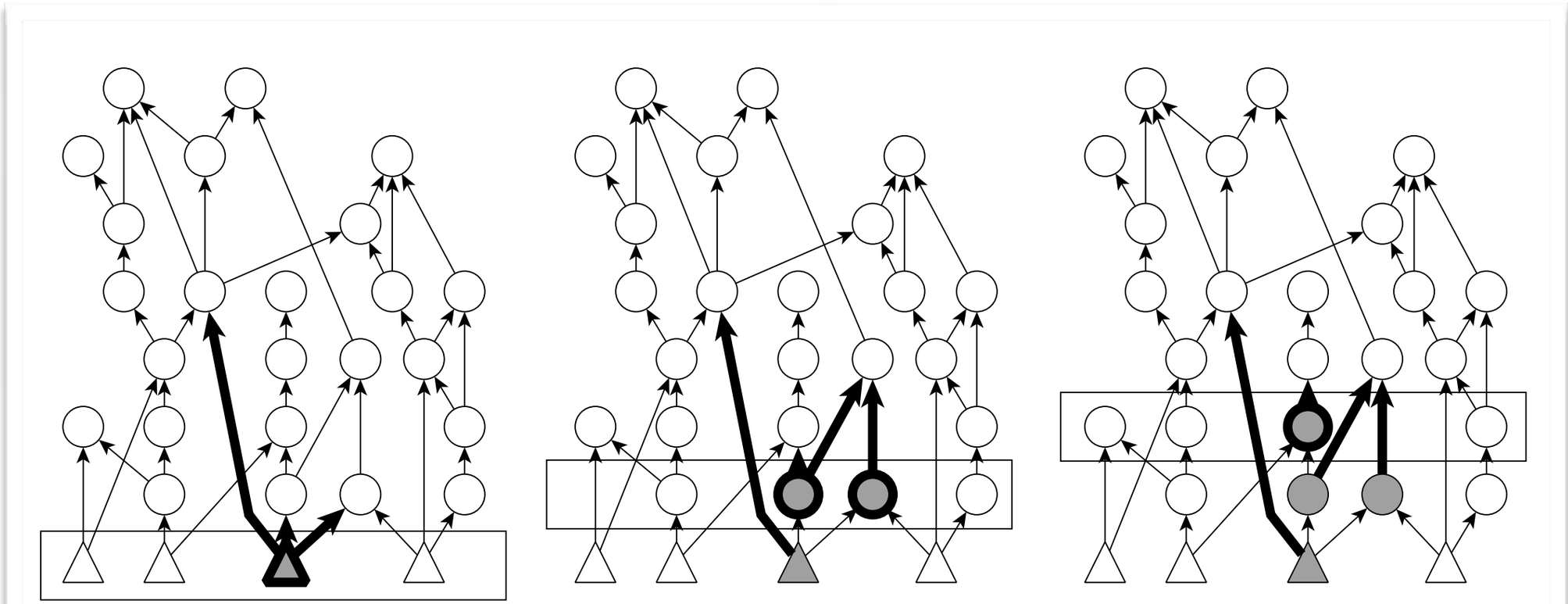
Abstract

Reactive programming improves the design of reactive applications by relocating the logic for managing dependencies

continuously process incoming network packets fall into this category. Historically, reactivity has been achieved via callbacks and inversion of control [14], commonly implemented

slides: http://quarp.proenca.org

DRP: minimise overall time



count steps

count messages

Scala.React

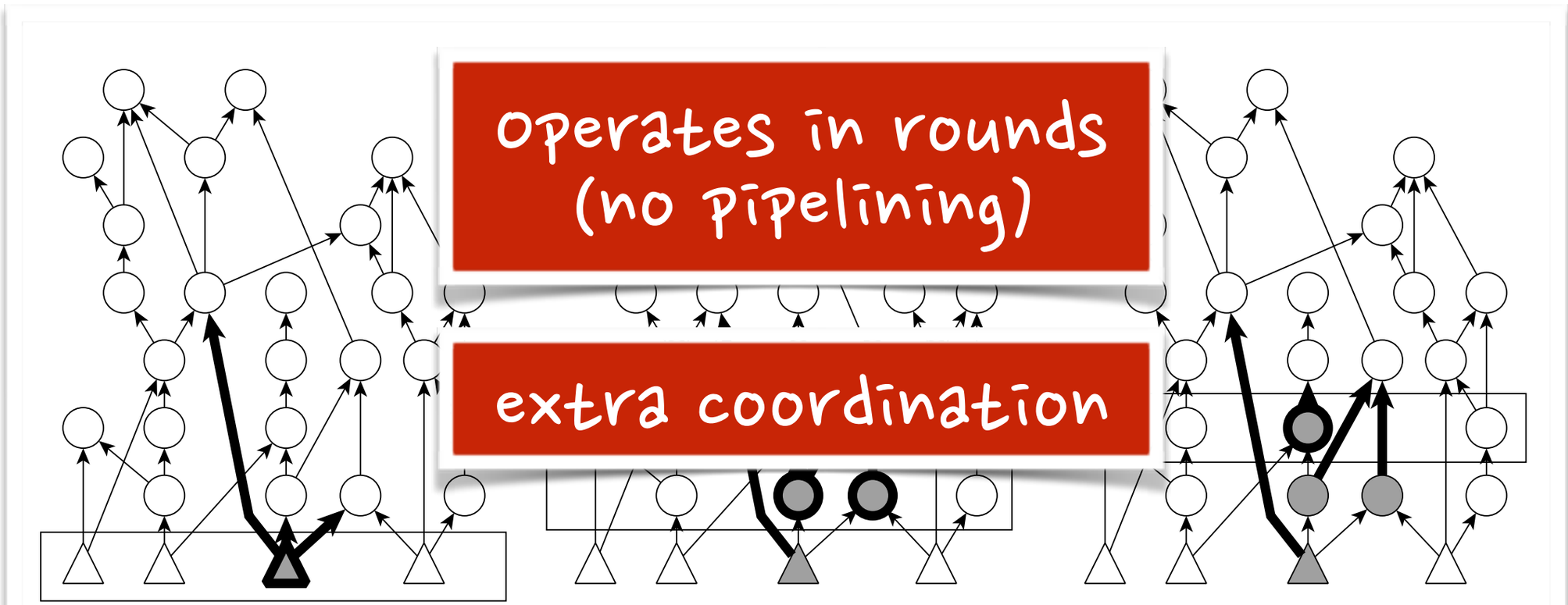
Scala.Rx

ELM

SID-UP

slides: http://quarp.proenca.org

DRP: minimise overall time



count steps

count messages

Scala.React

Scala.Rx

ELM

SID-UP

slides: http://quarp.proenca.org

DRP for the IoT

~~Operates in rounds (no pipelining)~~

~~extra coordination~~

Minimum coordination

maximum concurrency

Data *can* be lost

Avoid glitches (and similar probs.)

Quality Aware

Reactive Programming

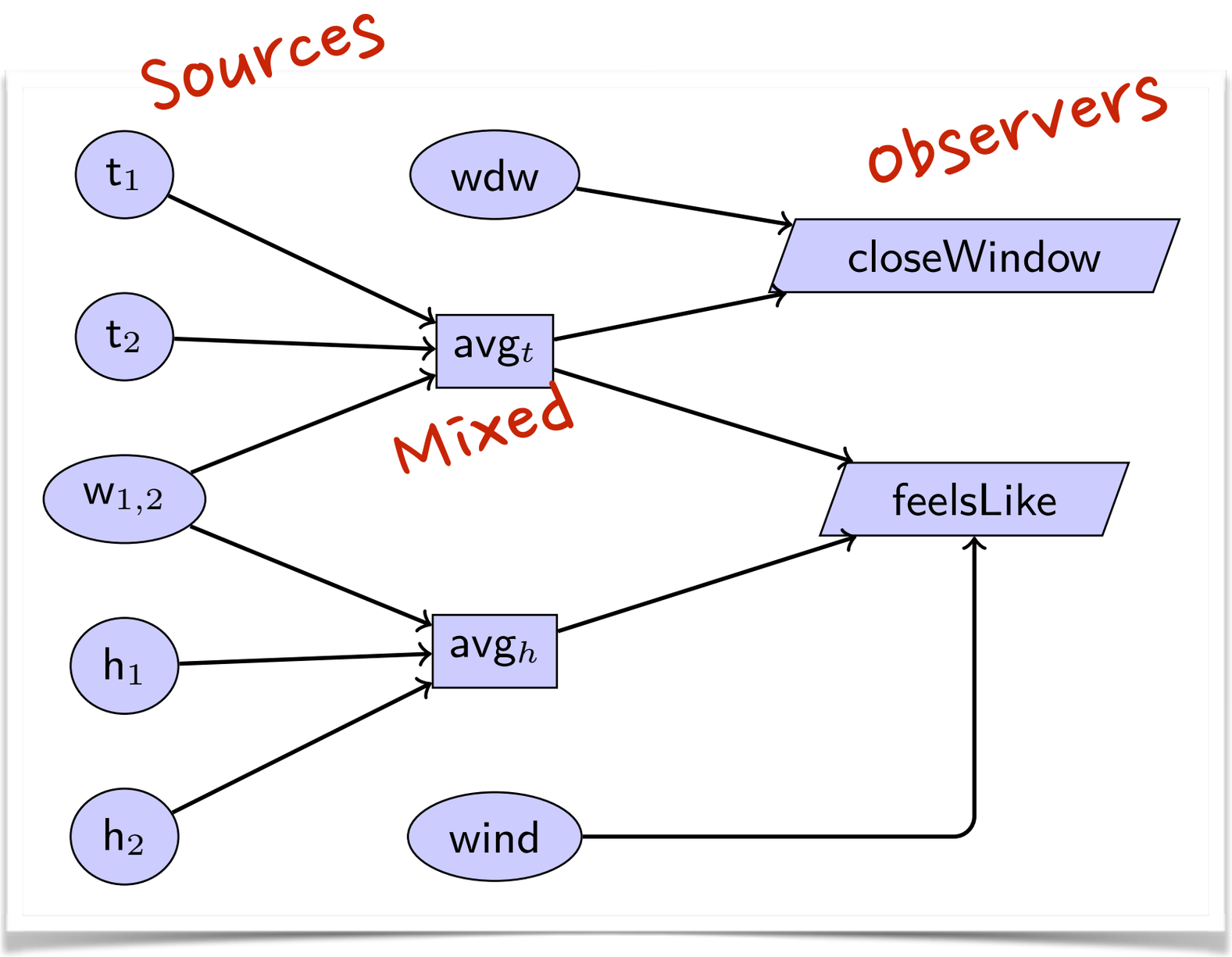
for the **IoT**

Reactive Programming with Failure

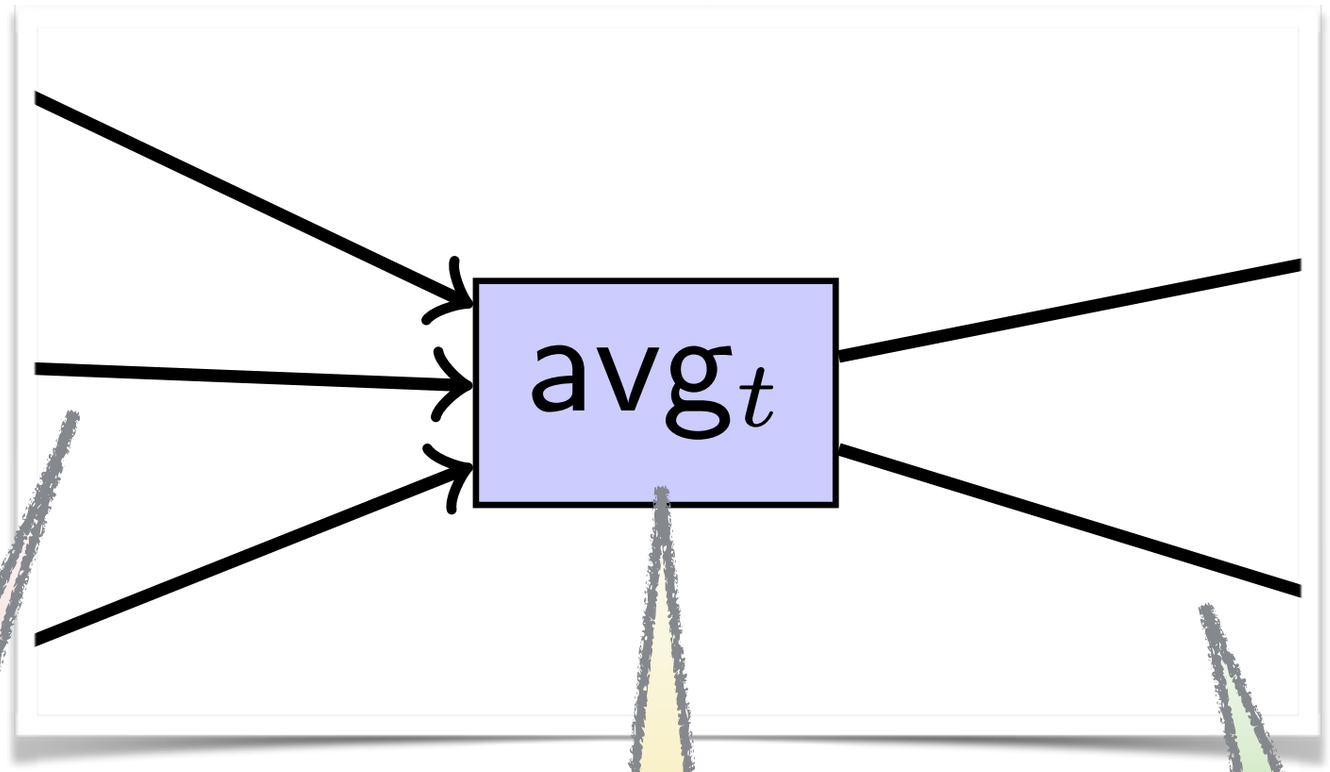
Sources keep publishing

Data lost: wait for next round

components can "chose" to ignore data



“Good enough” inputs

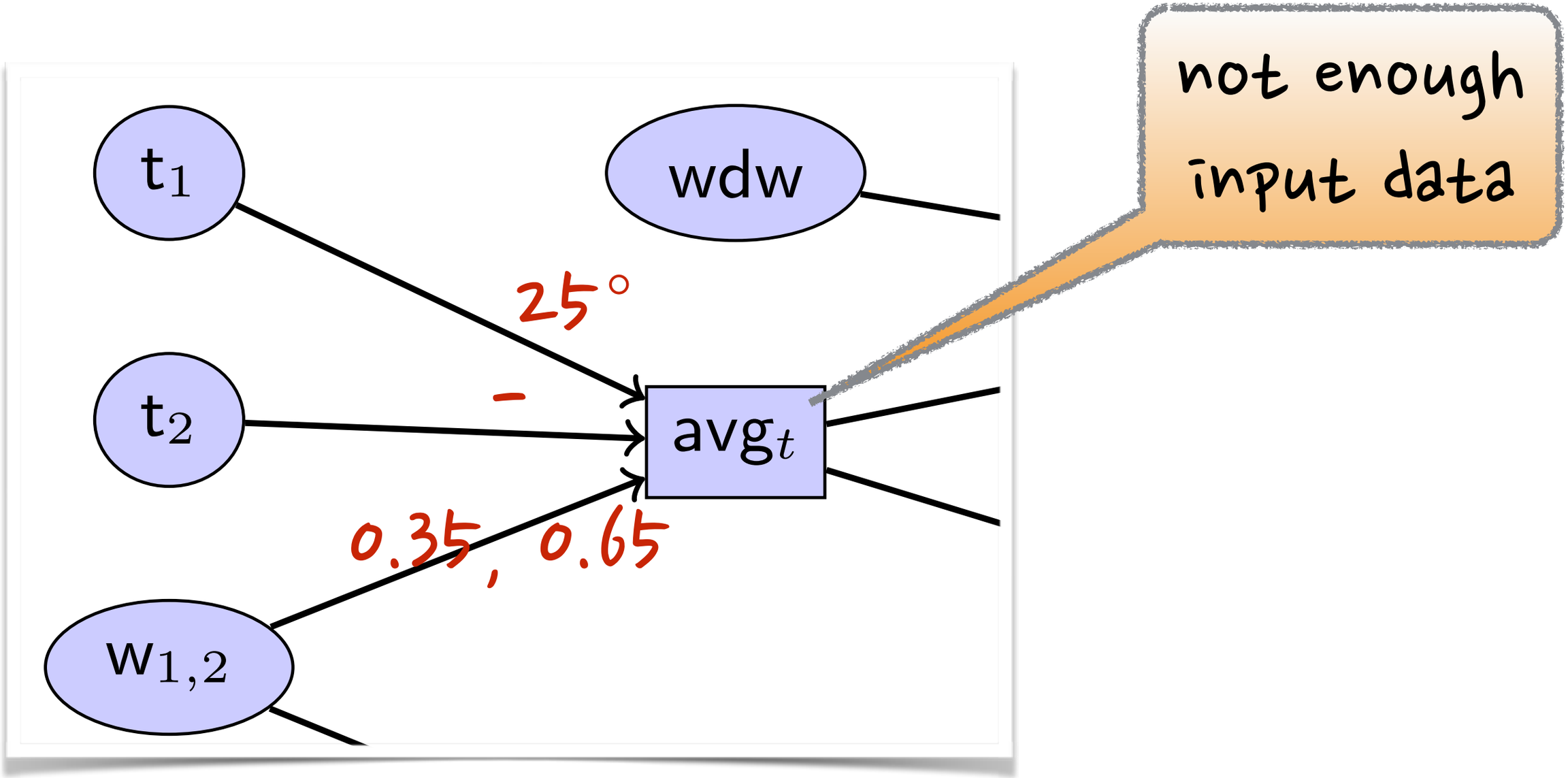


1. Data arrives

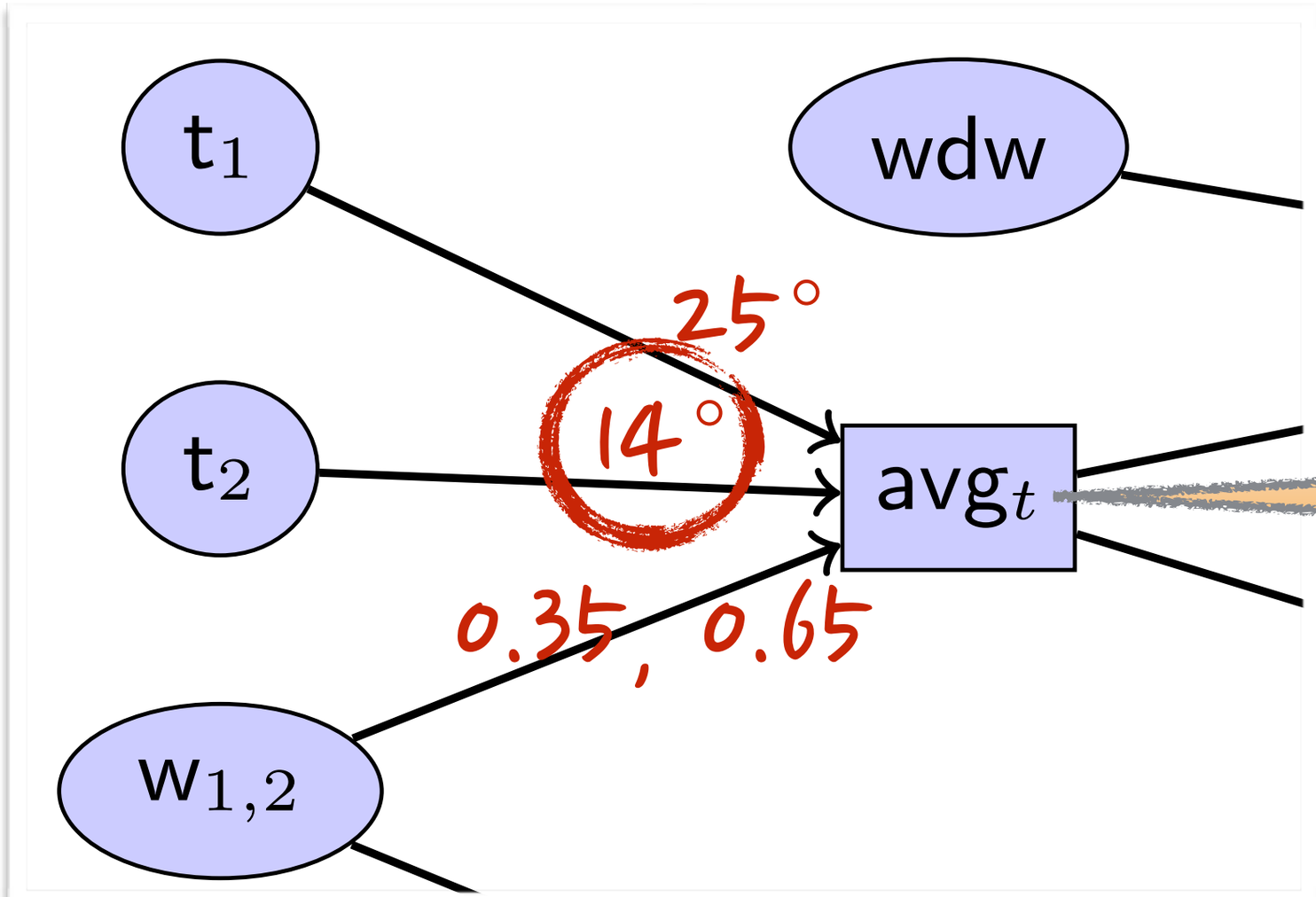
2. Input data is “good enough”?

3. Publish if “yes”

“Good enough” inputs



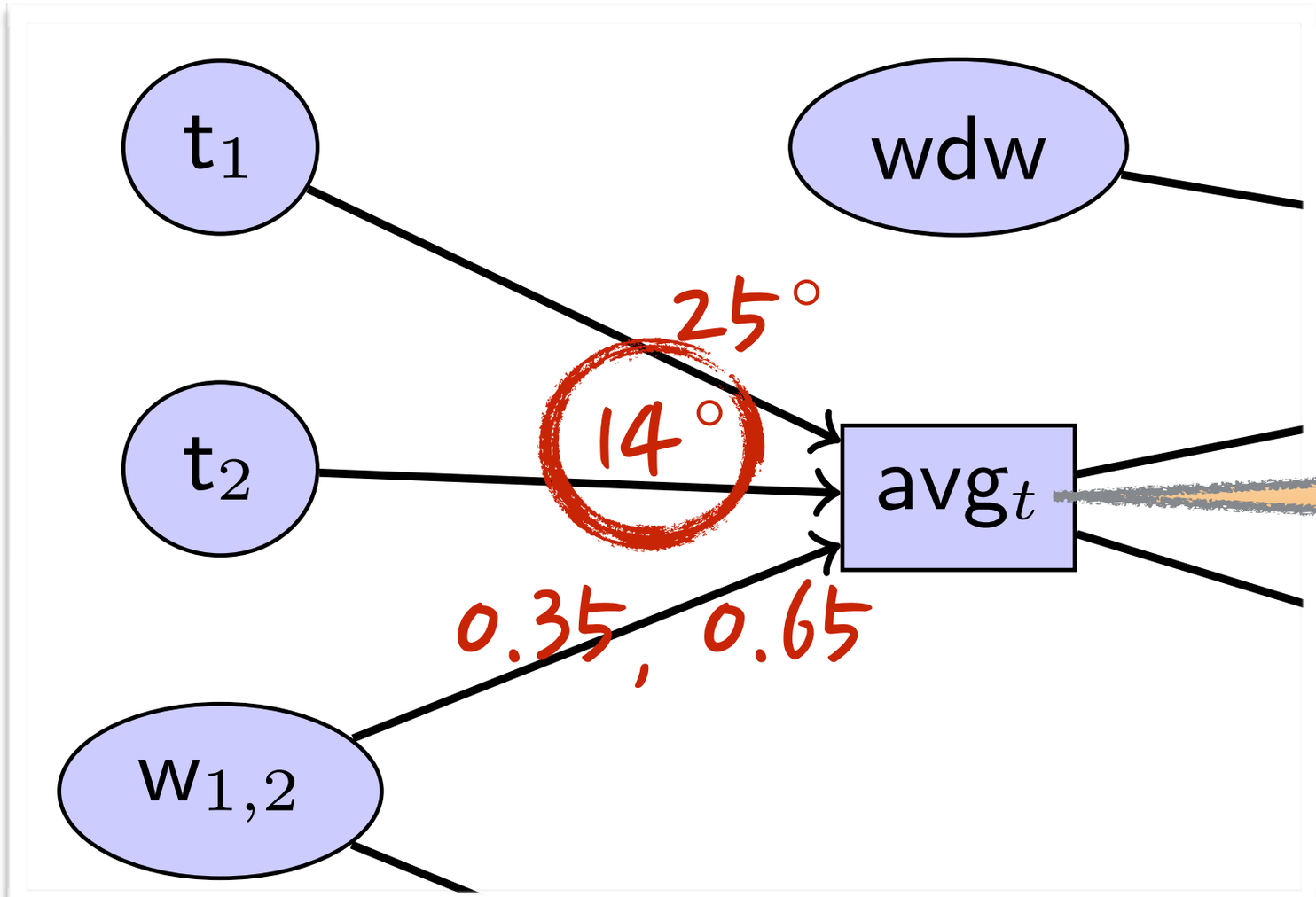
“Good enough” inputs



not enough input data

seems ok...?

“Good enough” inputs



not enough input data

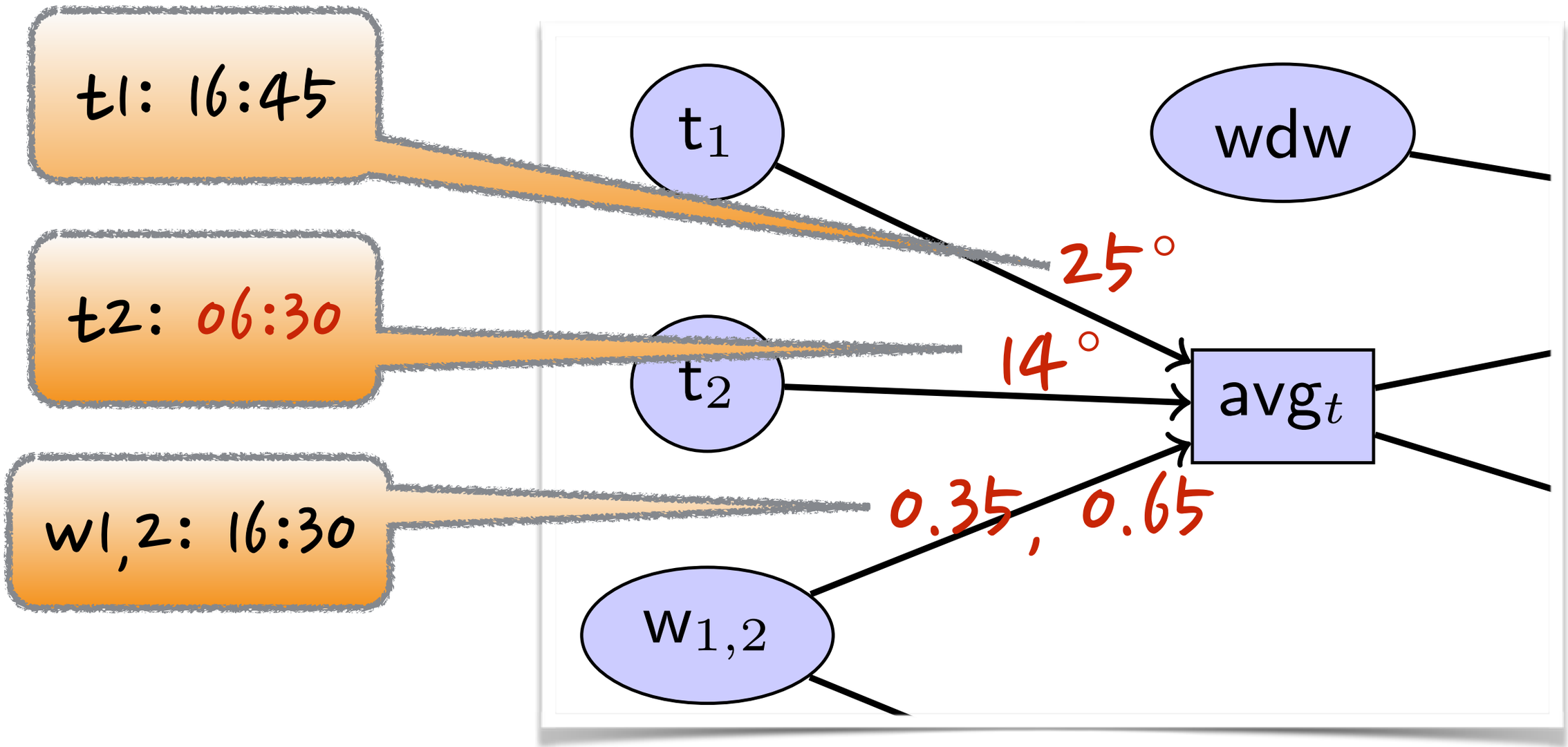
seems ok...?

no glitches?

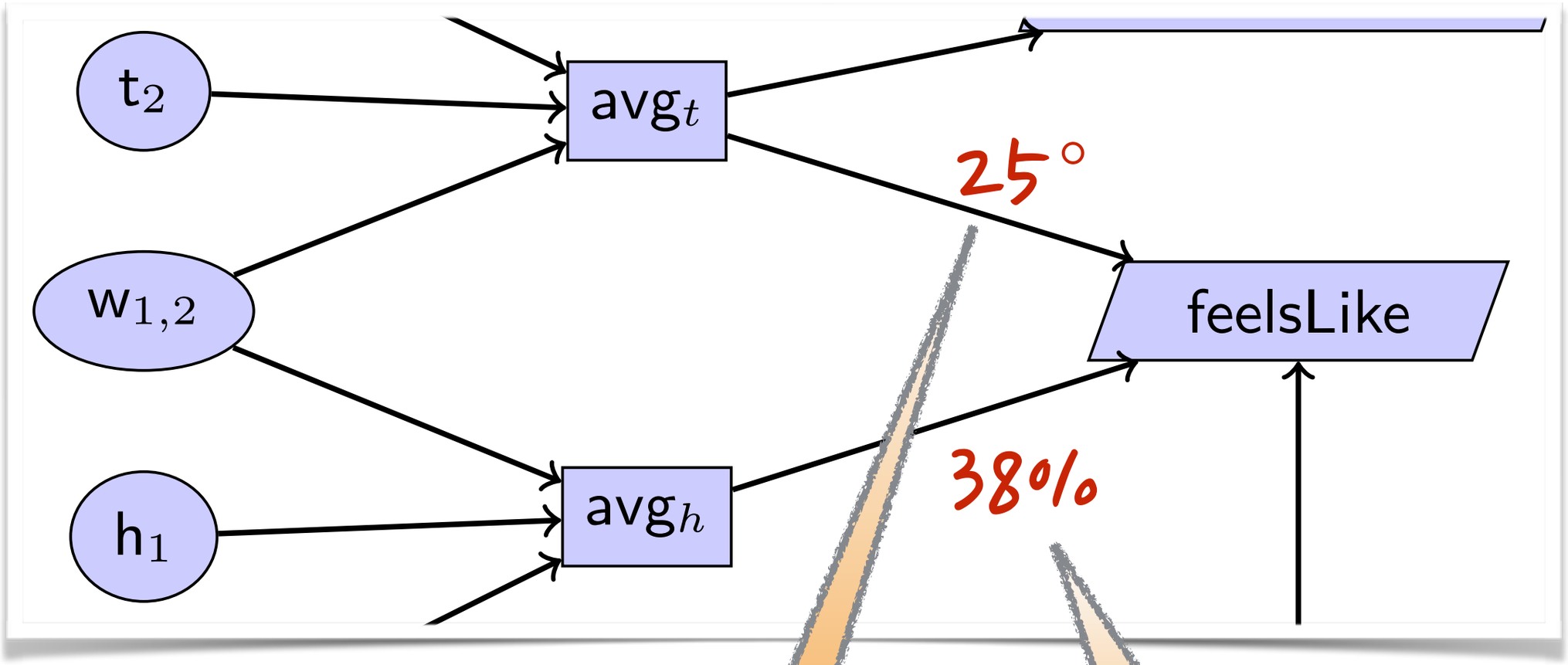
values up-to-date?

...

Need **Context**

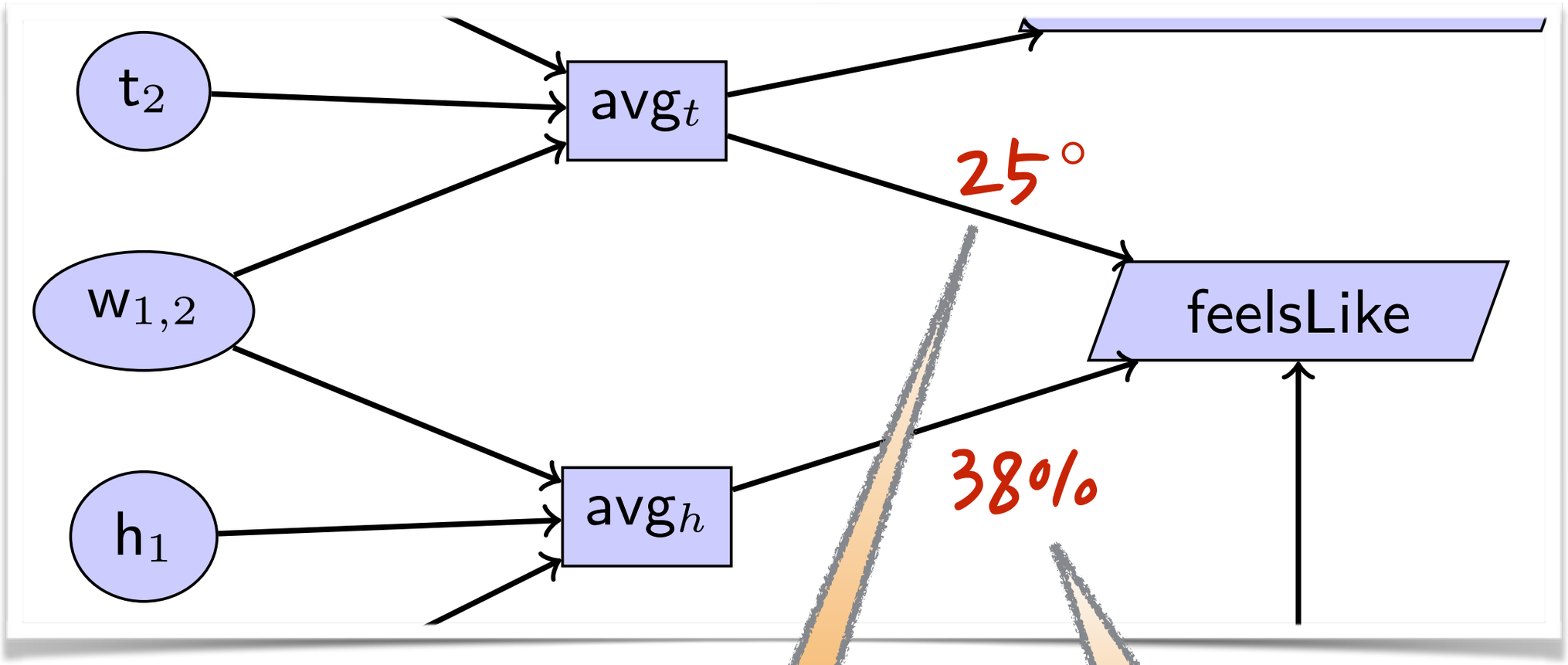


Avoiding glitches

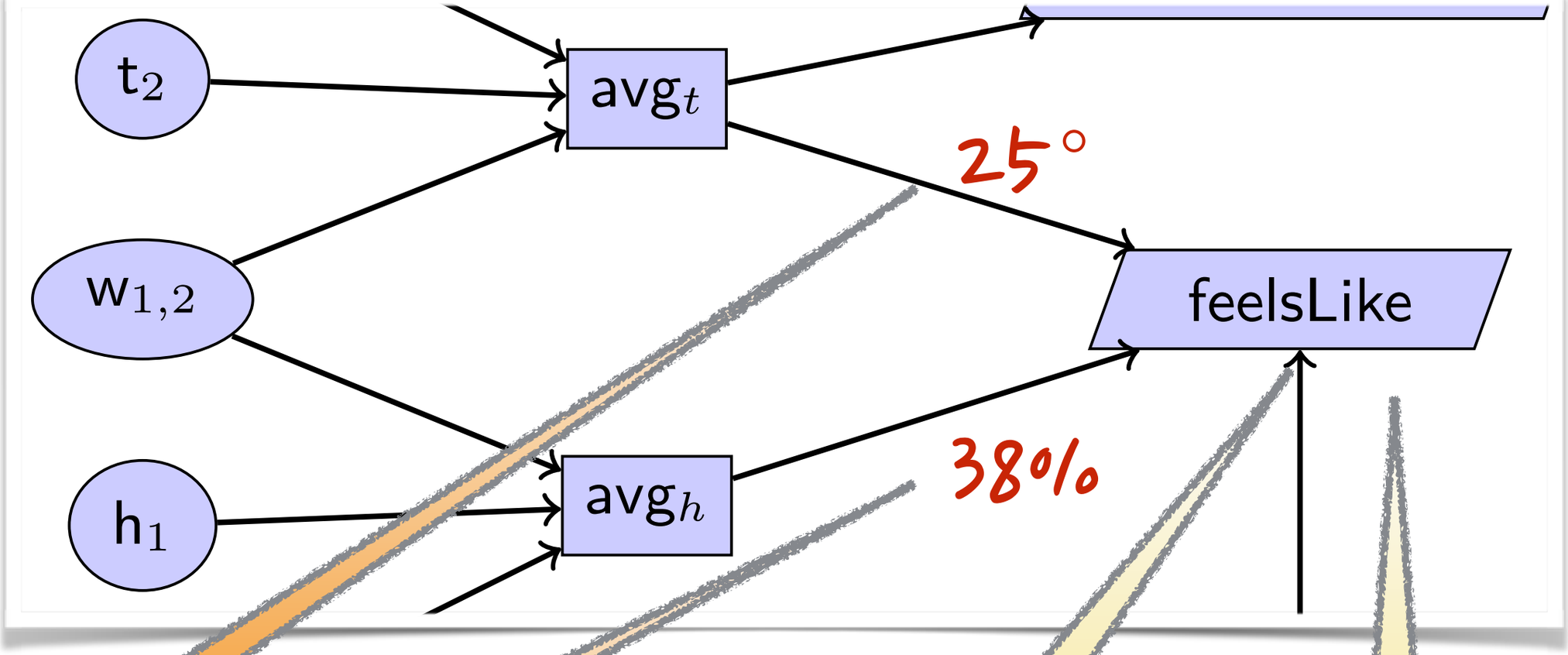


source-ID + counter

Avoiding glitches



Generalising contexts



ctx-t

ctx-h

ctx-FL =
ctx-t + ctx-h

Qual(ctx-FL)
$\geq$ threshold

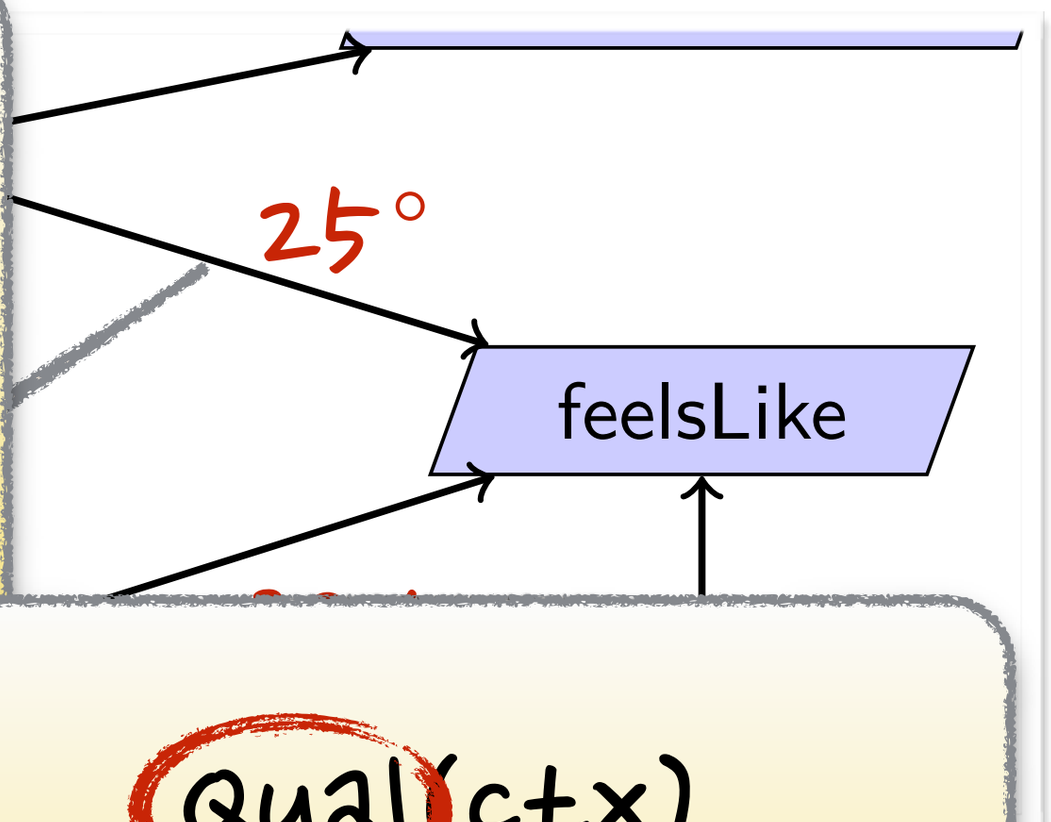
Generalising contexts

ctx-FL =

ctx-1 + ctx-2

ctx-t

ctx-h



Qual(ctx) ≥ threshold

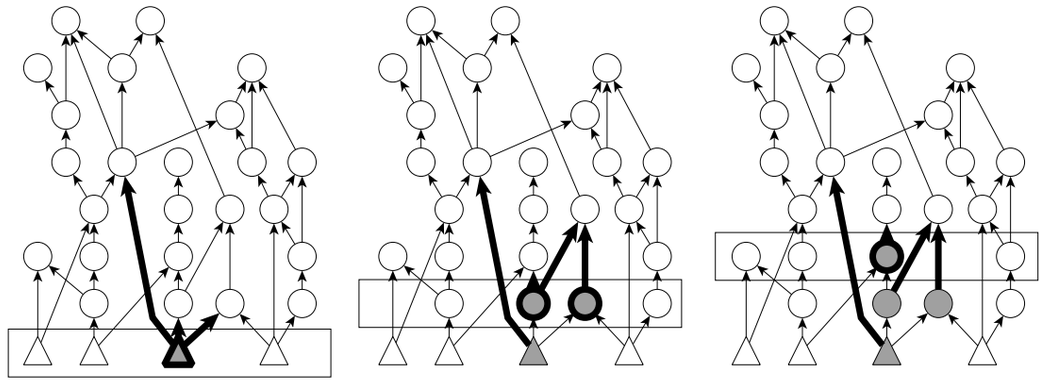
Beyond glitches

ctx-1 + ctx-2

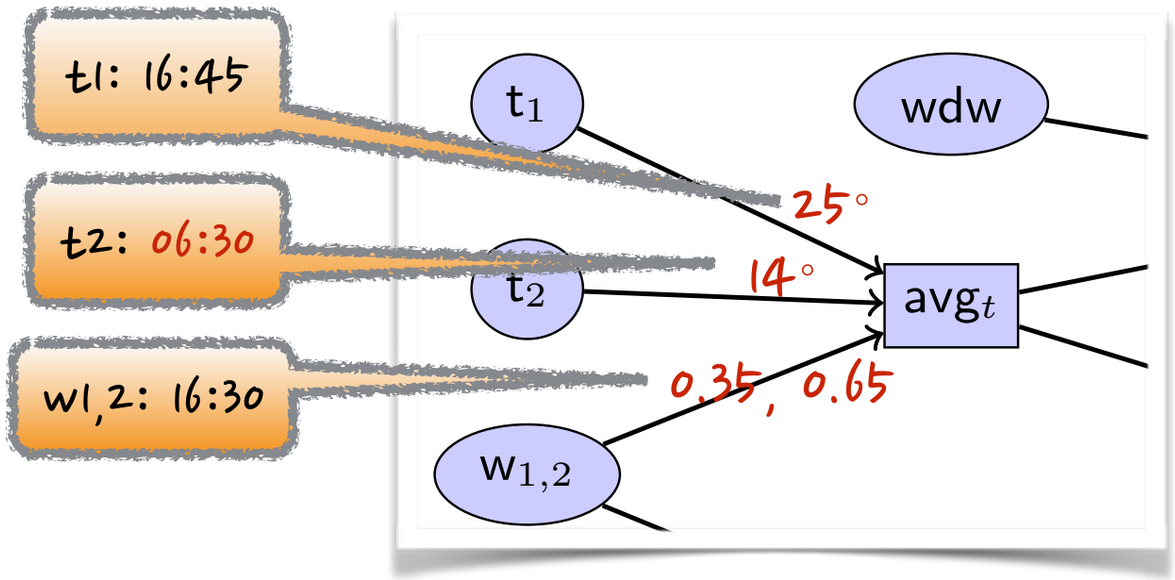
Qual(ctx) ≥ threshold

Geographic location	Ok = close-by enough
Relaxed glitch-freedom	Ok = small counter difference
Wall-clock Difference	Ok = small time-stamp difference

Wrapping up



Distributed Reactive Programming: not optimal for the IoT



Add context to messages

combine and measure contexts

"discard" instead of "wait"

Thank you!

slides: http://quarp.proenca.org