

# Typed Connector Calculus & more...

José Proença

1<sup>st</sup> ARCA-seminar  
19 Nov 2015



software **AR**chitecture  
& design **CA**culi

Arca, SADC, ARC, ...

More ideas?

cálculo

de programas, processos, arquitecturas, algoritmos

desenvolvimento “in-the-large”

componentes, arquitectura, coordenação, SPL, etc.

# ARCA

software **AR**chitecture & design **CA**lculi

meet every 2 weeks

3 Dec, 17 Dec, 14 Jan, 28 Jan, ...

ongoing work

volunteers!

practice presentations

tutorials

round table updates

<http://arca.di.uminho.pt>

# For today



My research history... (briefly)

(some of) my ongoing interests

## Typed Connector Families\*

José Proença<sup>1,2</sup> and Dave Clarke<sup>3</sup>

<sup>1</sup> HASLab – INESC TEC and Universidade do Minho, Portugal

<sup>2</sup> iMinds-DistriNet, Dept Computer Science, KU Leuven, Belgium

<sup>3</sup> Dept. Information Technology, Uppsala University, Sweden

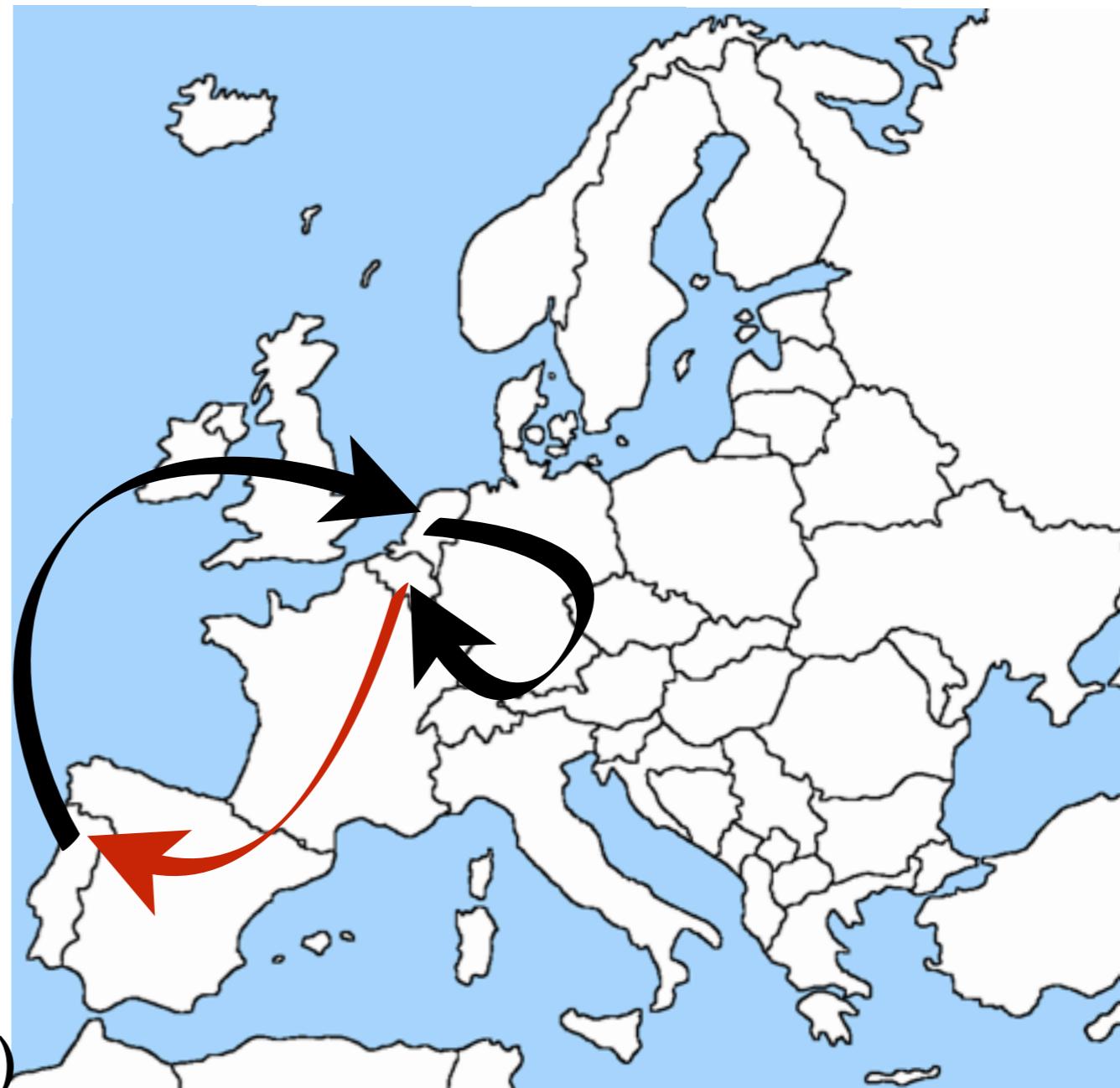
[jose.proenca@cs.kuleuven.be](mailto:jose.proenca@cs.kuleuven.be)

[dave.clarke@it.uu.se](mailto:dave.clarke@it.uu.se)

My most recent work

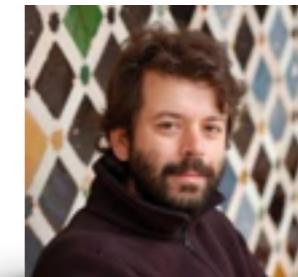
**Abstract.** Typed models of connector/component composition specify interfaces describing ports of components and connectors. Typing ensures that these ports are plugged together appropriately, so that data can flow out of each output port and into an input port. These interfaces typically consider the direction of data flow and the type of values flowing. Components, connectors, and systems are often parameterised in such a way that the parameters affect the interfaces. Typing such *connector families* is challenging. This paper takes a first step towards addressing this

- PT 2000-05
  - ▶ University of Minho  
Functional programming
- Netherlands 06-10
  - ▶ CWI (Amsterdam)  
Concurrency, Coordination
- Belgium 11-15
  - ▶ KU Leuven (Next to Brussels)  
(Coordination), Software product lines, logic, programming languages, wireless sensor networks...



- PT 2000-05

- ▶ University of Minho  
Functional programming



- Netherlands 06-10

- ▶ CWI (Amsterdam)  
Concurrency, Coordination



Farhad Arbab



Dave Clarke

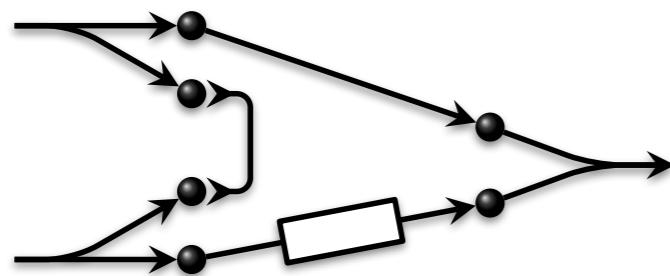
- Belgium 11-15

- ▶ KU Leuven (Next to Brussels)  
(Coordination), Software product lines, logic, programming languages, wireless sensor networks, ...



Danny  
Hughes

# Coordination



Reo coordination language

different  
semantics

distributed implementation

Synchrony vs. asynchrony

constraint solving  
as coordination

# Variability



delta programming

mTVL

language for  
describing variability

Feature Nets

petri nets with  
annotations

# Wireless Sensor Networks



coordinating lightweight components



network overlay based on piggybacking

formalisation

cheap inspection/update of meta-data  
(properties of components & bindings)

IOT

Smartmesh IP

reactive programming

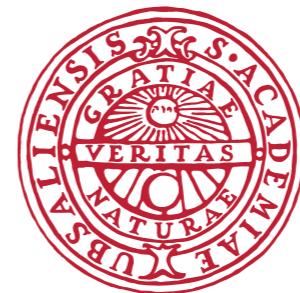
# Watch video



[https://www.youtube.com/watch?v=JJjleFs8f\\_4](https://www.youtube.com/watch?v=JJjleFs8f_4)

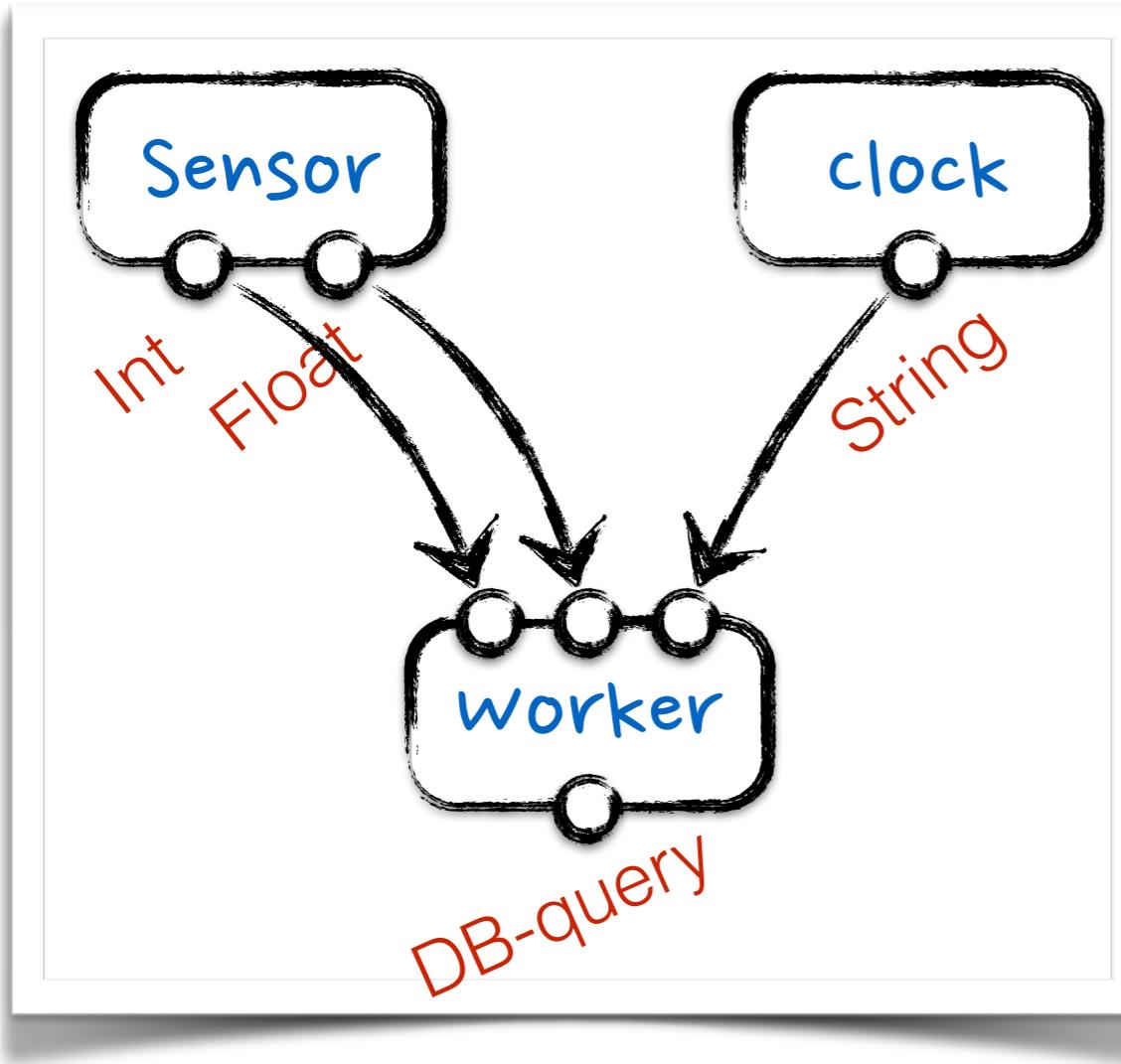
# Typed Connector Families

**José Proença** & Dave Clarke  
(KU Leuven, Belgium) (UPPSALA University, Sweden)

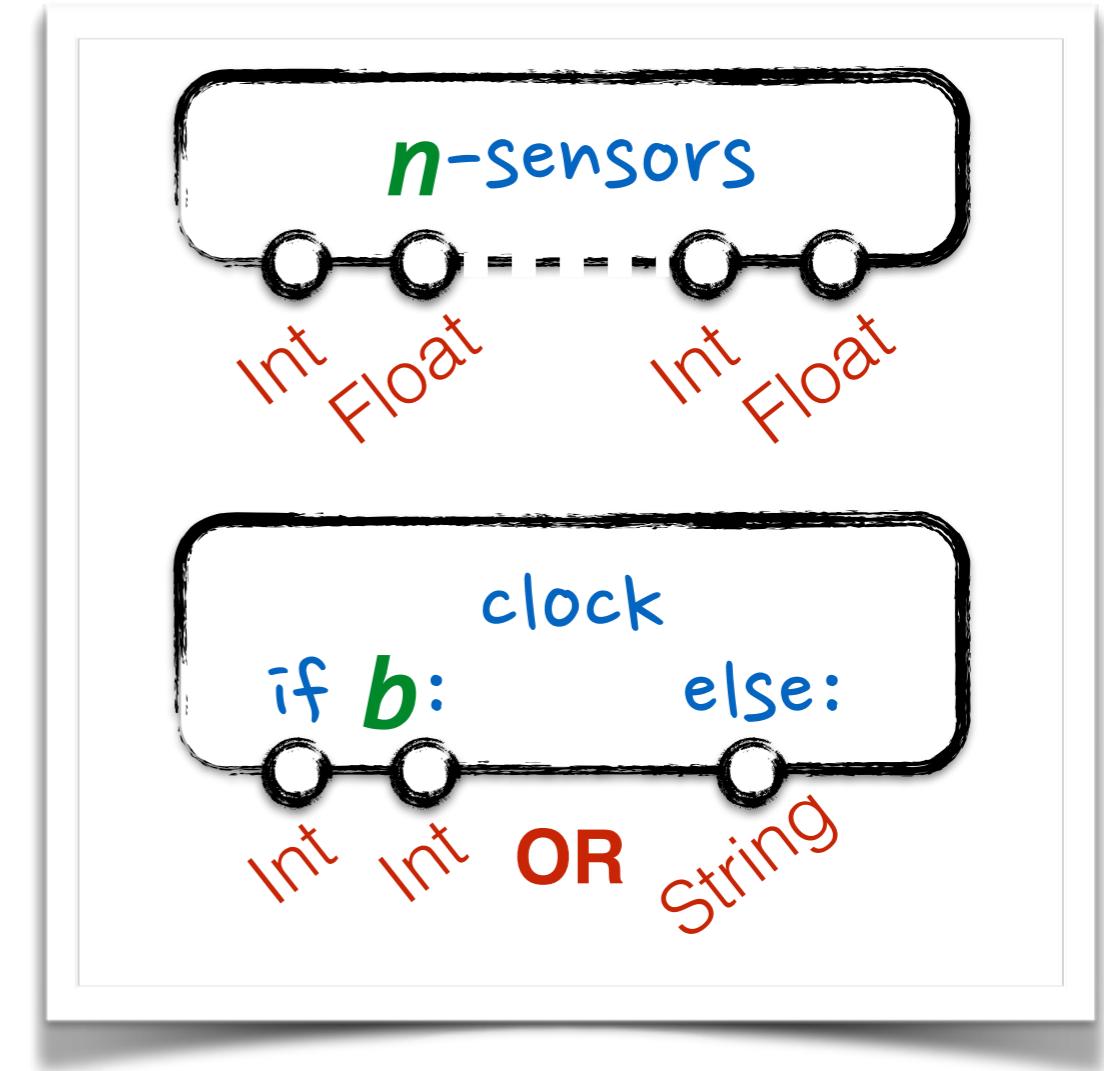


UPPSALA  
UNIVERSITET

# Motivation

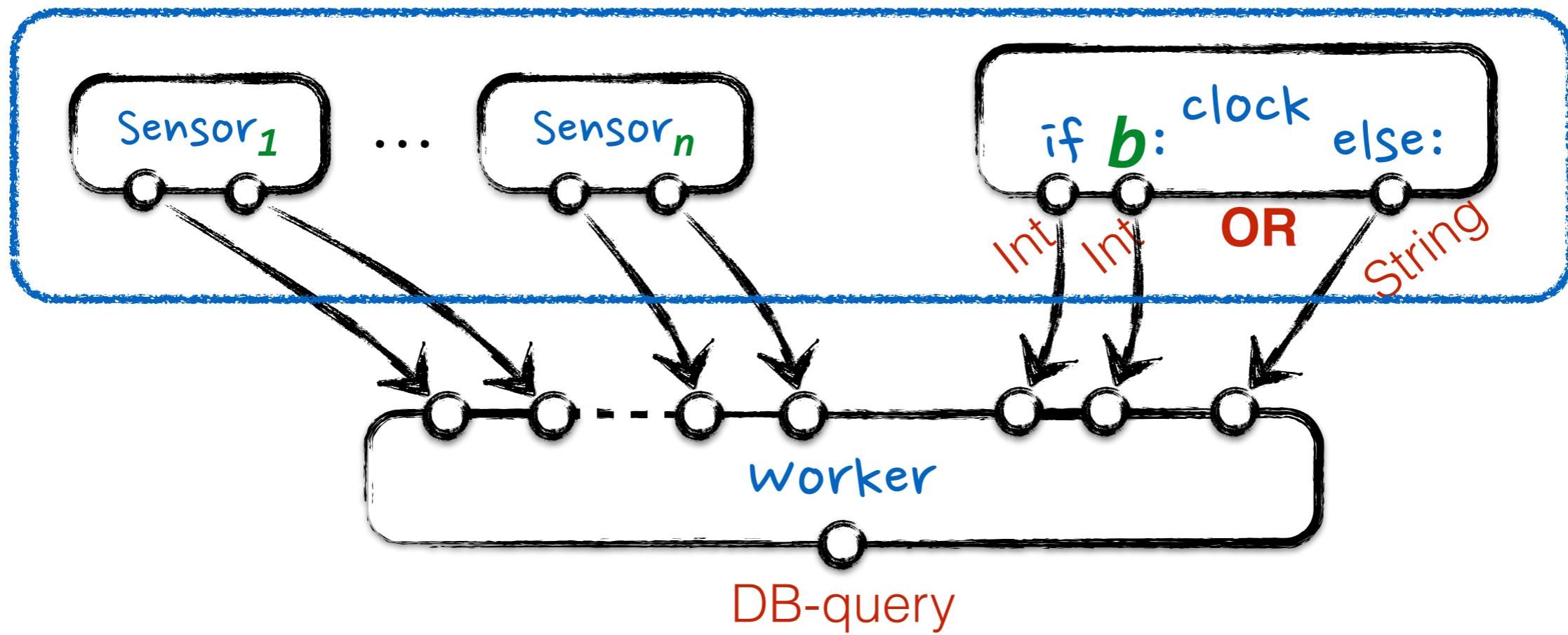


Static interfaces



Software product lines

# Motivation



$$\lambda n:\text{Int}, b:\text{Bool} \cdot \text{Sensor}^n \otimes \text{clock}(b)$$

$$: \forall n:\text{Int}, b:\text{bool} \cdot o \rightarrow (\text{Int} \otimes \text{Float})^n \otimes (\text{Int} \otimes \text{Int} \oplus^b \text{String})$$

# Outline

basic connector calculus

parameterised connector calculus

**connector families**

Type-checking approach

Sensor  $\otimes$  clock ; worker  
:  $o \rightarrow$  DB-query

$\lambda n:\text{Int} \cdot \text{Sensor}^n$   
:  $\forall n:\text{Int} \cdot o \rightarrow \dots$

composing  
parameterised cc

for untyped ports

# Basic connector calculus

$c ::= c_1 ; c_2$	sequential composition
$c_1 \otimes c_2$	parallel composition
$\text{id}_I$	identity connectors
$\gamma_{I,J}$	symmetries
$\text{Tr}_I(c)$	traces
$p \in \mathcal{P}$	primitive connectors

category with  
a tensor (monoid)  
symmetries  
traces

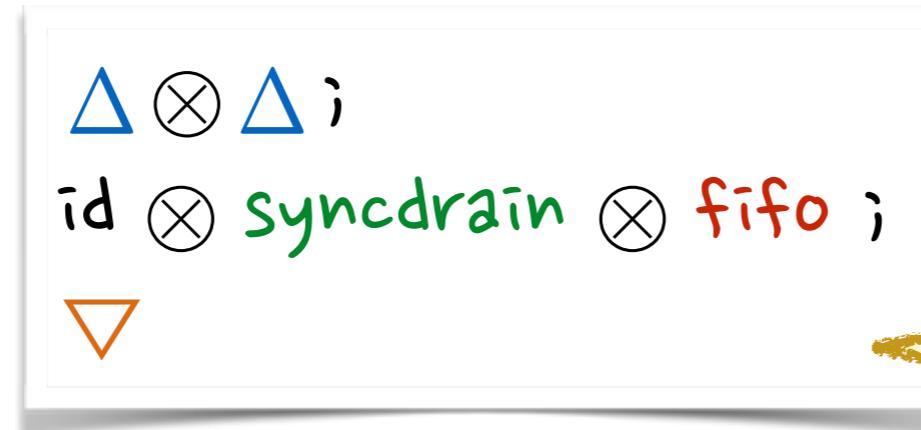
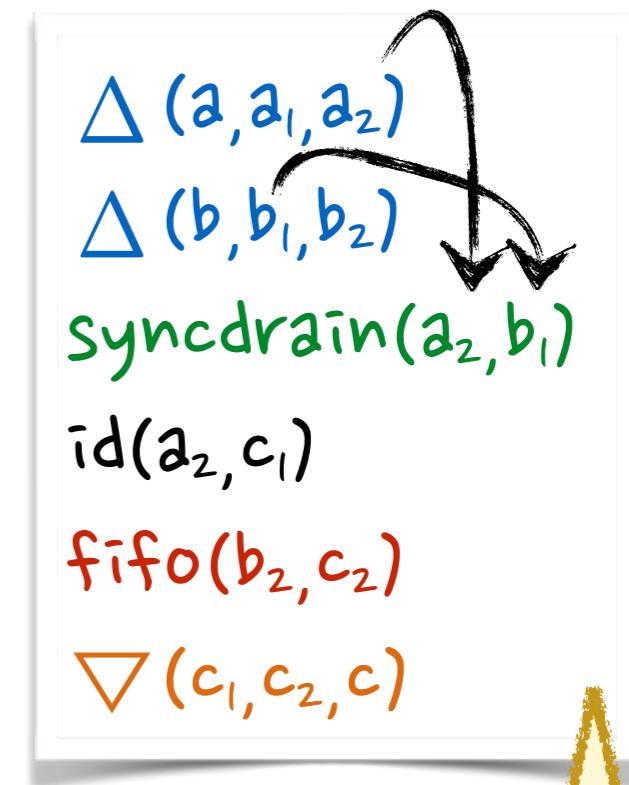
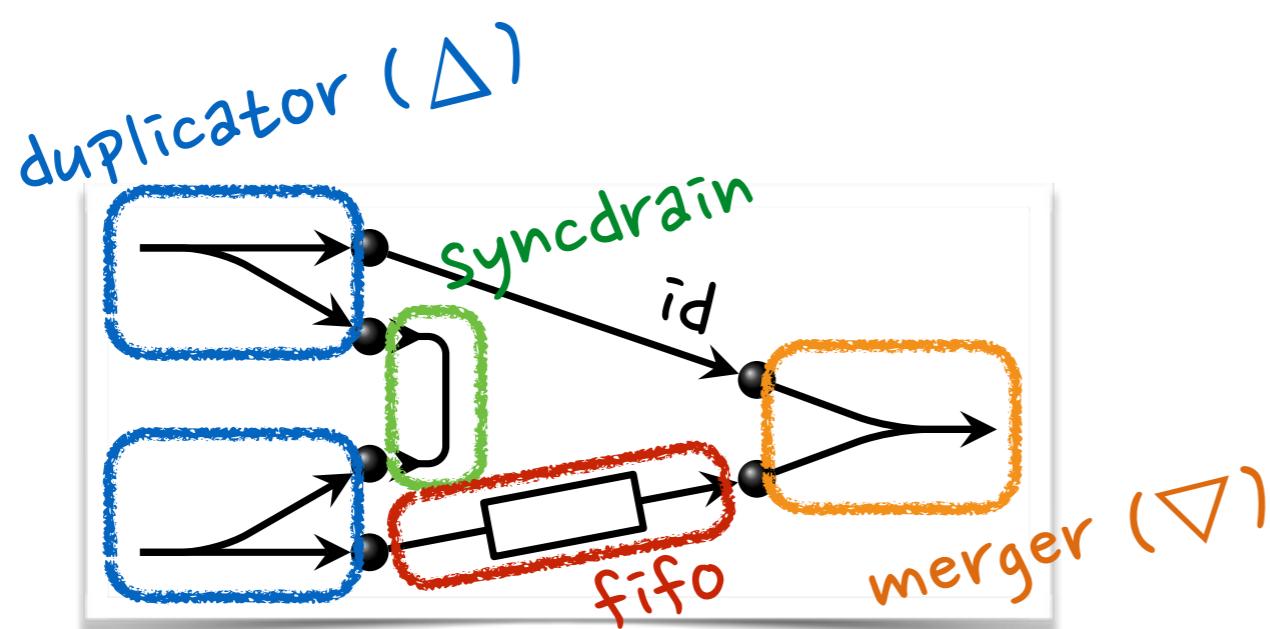
$I, J ::= I \otimes J$	tensor
0	empty interface
A	port type

connectors:  
morphisms

interfaces:  
objects

Based on connector algebra from Bruni et. al. (TCS'06)

# Reo example



**traditional:**  
with port names

**connector calculus**

# Visualisation of connectors

recall:

$\text{id}_I$

$\gamma_{I,J}$

$\text{Tr}_I(c)$

$I, J ::= I \otimes J$  tensor

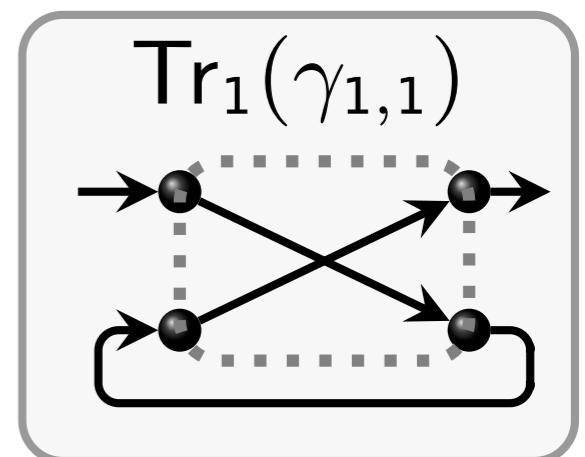
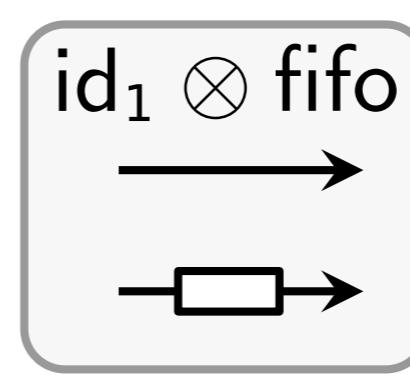
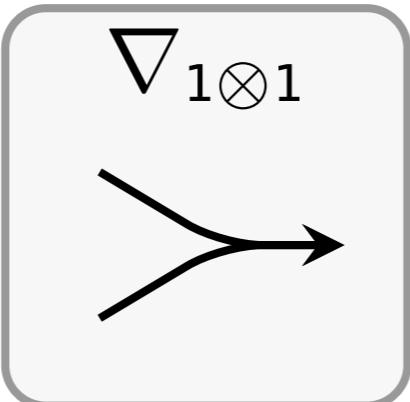
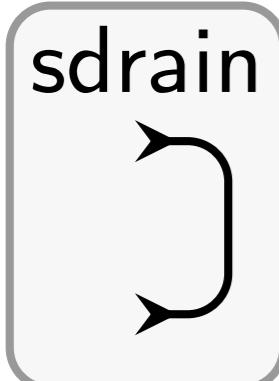
| 0

empty interface

| A

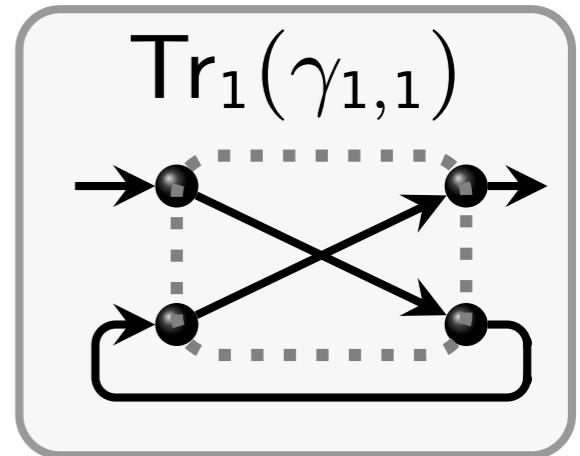
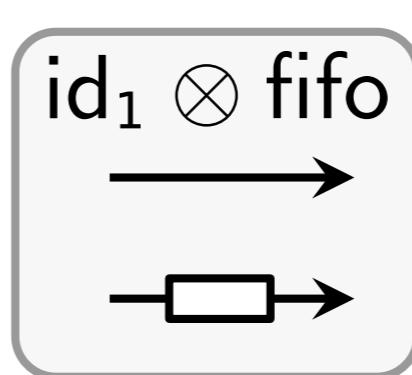
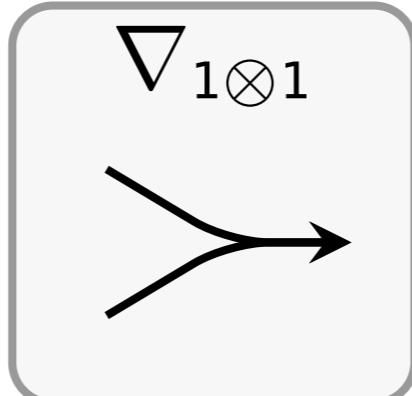
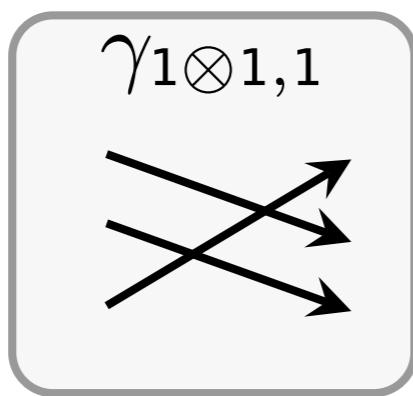
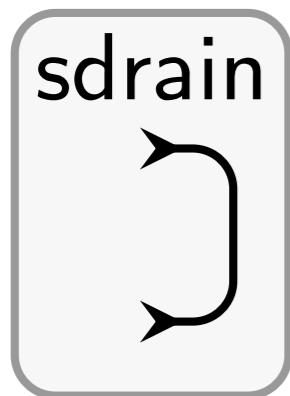
port type

1 is some port type



Data goes always from left to right

# Typing connectors



$1 \otimes 1$   
→ 0

$1 \otimes 1 \otimes 1$   
→  $1 \otimes 1 \otimes 1$

$1 \otimes 1$   
→ 1

$1 \otimes 1$   
→  $1 \otimes 1$

$1 \rightarrow 1$

**conn** :  $I \rightarrow J$

IF     $c1 : I_1 \rightarrow J$    &    $c2 : J \rightarrow J_2$   
 THEN     $c1 ; c2 : I_1 \rightarrow J_2$

# Constraint-based type rules

(sequence)

$$\frac{\Gamma \mid \phi \vdash c_1 : I_1 \rightarrow J_1 \quad \Gamma \mid \phi \vdash c_2 : I_2 \rightarrow J_2}{\Gamma \mid \phi, J_1 = I_2 \vdash c_1 ; c_2 : I_1 \rightarrow J_2}$$

(trace)

$$\frac{\Gamma \mid \phi \vdash c : J_1 \rightarrow J_2}{\Gamma \mid \phi, J_1 = X \otimes I, J_2 = Y \otimes I \vdash \text{Tr}_I(c) : X \rightarrow Y}$$

# Parameterised connector calculus

$\lambda x:\text{Int} \cdot c$

fifo<sup>exp</sup>

means: fifo  $\otimes \dots \otimes$  fifo ("exp" times)

$(\Delta_I^x)^x \leftarrow \text{exp}$

means:  $\Delta_I^0 \otimes \dots \otimes \Delta_I^{\text{exp}-1}$

fifo  $\oplus^{\text{exp}}$  drain

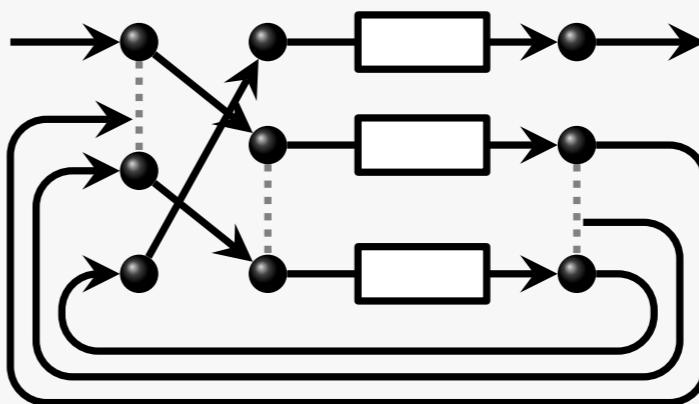
means: if (exp) then (fifo)  
else (drain)

$c ::= \dots$	connectors
$  \quad c^{x \leftarrow \alpha}$	$n$ -ary parallel replication
$  \quad c_1 \oplus^\phi c_2$	conditional choice
$  \quad \lambda x : P \cdot c$	parameterised connector
$  \quad c(\phi)$	bool-instantiation
$  \quad c(\alpha)$	int-instantiation

$I ::= \dots$	interfaces
$  \quad I^\alpha$	$n$ -ary parallel replication
$  \quad I \oplus^\phi J$	conditional choice
$\alpha, \beta$	integer expressions
$\phi, \psi$	boolean expressions

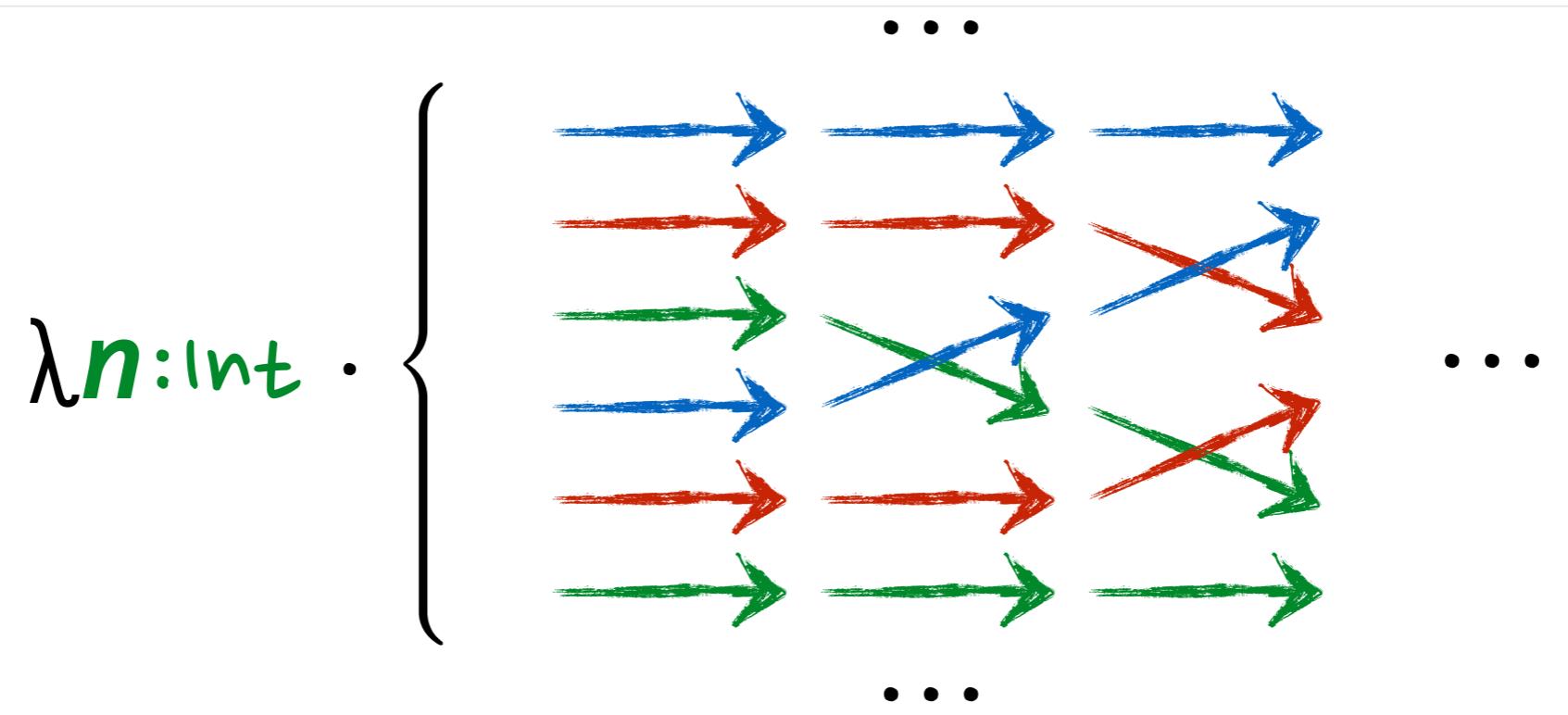
# Example: seq-fifo

*seq-fifo* =  
 $\lambda n : \mathbb{N} .$   
 $\text{Tr}_{n-1}$   
 $(\gamma_{n-1,1} ; \text{fifo}^n)$



seq-fifo :  $\forall n:\text{int} . 1 \rightarrow 1$

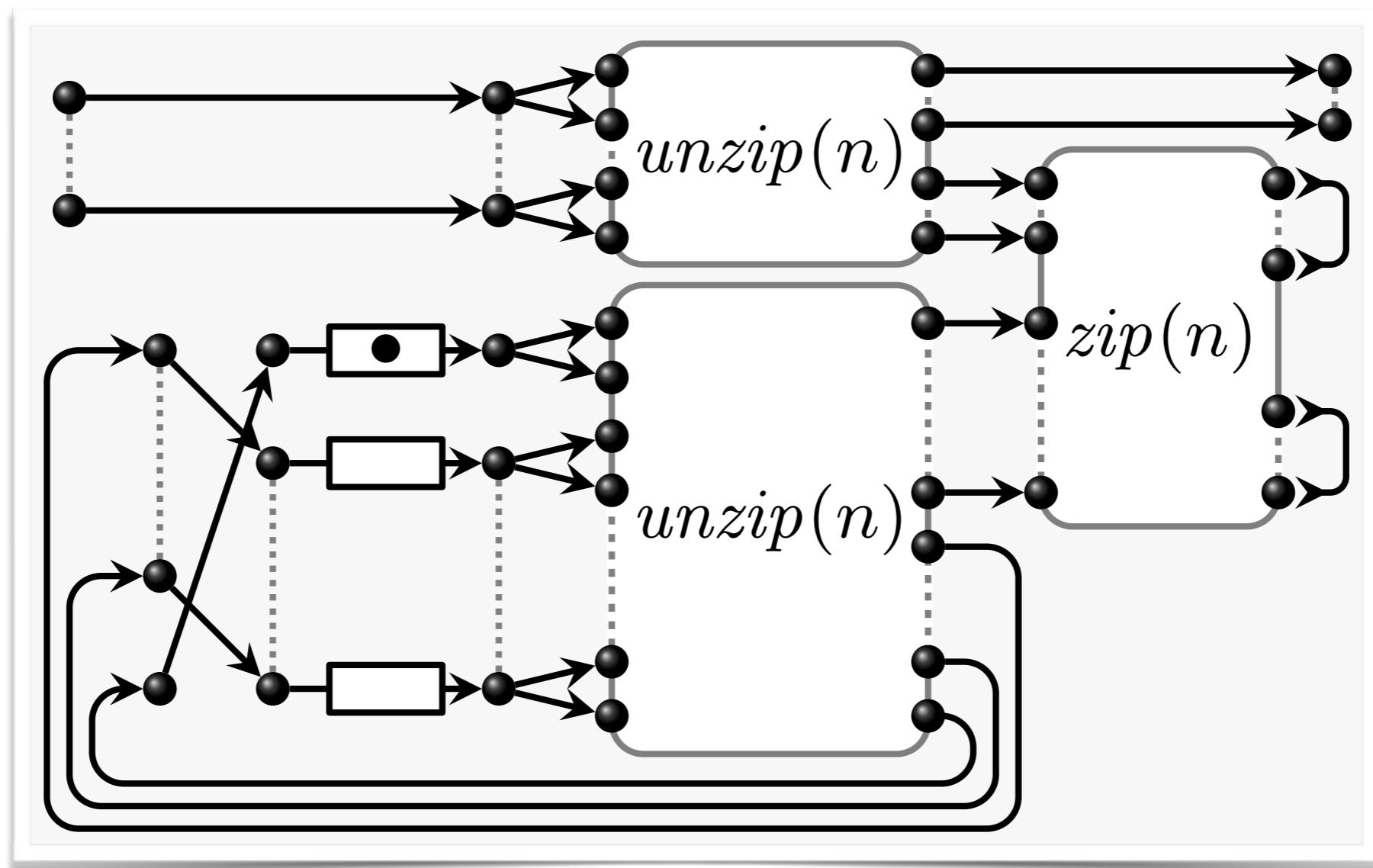
# Example - zip



$$\text{zip} = \lambda n : \mathbb{N} \cdot \text{Tr}_{2n^2-2n} \left( \gamma_{2n^2-2n, 2n} ; (\text{id}_{n-x} \otimes \gamma_{1,1}^x \otimes \text{id}_{n-x})^{x \leftarrow n} \right)$$

ZIP :  $\forall n:\text{Int} \cdot (1^n)^2 \rightarrow (1^2)^n$

# Example - sequencer



sequencer :  $\forall n:\text{Int} \cdot 1^n \rightarrow 1^n$

# Connector Families

$$\frac{(\text{restriction})}{\Gamma | \phi \vdash \psi \quad \Gamma | \phi, \psi \vdash c : T}
 \frac{\Gamma | \phi \vdash \psi \quad \Gamma | \phi, \psi \vdash c : T}{\Gamma | \phi \vdash c |_{\psi} : T |_{\psi}}$$

$$\lambda n:\text{Int} \cdot \text{Tr}_{n-1}(\gamma_{n-1,1} ; \text{fifo}^n) \mid_{n < 5}$$

$$\frac{(\text{fam-sequence})}{\Gamma | \phi \vdash c_1 : \forall \overline{x_1 : T_1} \cdot I_1 \rightarrow J_1 |_{\psi_1} \quad \Gamma | \phi \vdash c_2 : \forall \overline{x_2 : T_2} \cdot I_2 \rightarrow J_2 |_{\psi_2} \quad \overline{x_1} \cap \overline{x_2} = \emptyset}
 \frac{\Gamma | \phi, J_1 = I_2 \vdash c_1 ; c_2 : \forall \overline{x_1 : T_1}, \overline{x_2 : T_2} \cdot I_1 \rightarrow J_2 |_{\psi_1, \psi_2}}$$

$$\underline{(\lambda x:\text{Int} \cdot c1) ; (\lambda y:\text{Int} \cdot c2)} : \underline{\forall x:\text{Int}, y:\text{Int} \cdot I_1 \rightarrow J_2}$$

# Solving Type Constraints

$$\Gamma \mid \phi \vdash \mathbf{c} : T \mid \psi$$

c is well-typed if:

given an empty context  $\underline{\Gamma}$

the type rules yield  $T, \varphi, \psi$

such that  $\varphi \wedge \psi$  have some solution

# Solving Type Constraints

c is well-typed

given

the type

such that

untyped ports:

interfaces as integers

$$([0]) = 0$$

$$([1]) = 1$$

$$([I \otimes J]) = ([I]) + ([J])$$

$$([I^\alpha]) = ([I]) * \alpha$$

# Solving Type Constraints

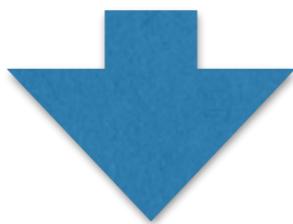
untyped ports:  
interfaces as integers

$$\begin{aligned} ([0]) &= 0 \\ ([1]) &= 1 \\ ([I \otimes J]) &= ([I]) + ([J]) \\ ([I^\alpha]) &= ([I]) * \alpha \end{aligned}$$

```
scala> import paramConnectors.DSL._  
import paramConnectors.DSL._  
  
scala> fifo  
res1: paramConnectors.Prim =  
fifo  
: 1 -> 1  
  
scala> lam(n, fifo | n > 5)  
res2: paramConnectors.IAbs =  
\n.(fifo | (n > 5))  
: ∀n:I . 1 -> 1 | n > 5  
  
scala> val sequencer = ...  
sequencer: paramConnectors.IAbs =  
\n(...)  
: ∀n:I . n -> n  
  
scala> lam(b, b? fifo + drain) &  
      lam(c, c? fifo + id*fifo)  
res3: paramConnectors.Seq = ...  
: ∀b:B,c:B . 1 -> 1 | c & b
```

# Example

seq-fifo =  $\lambda n:\text{Int} \cdot Tr_{n-1}(g_{n-1,1}; \text{fifo}^n) \mid_{n < 5}$



$\emptyset \mid \mathbf{1} \otimes (n - 1) = \mathbf{1}^n , \quad (n - 1) \otimes \mathbf{1} = X \otimes (n - 1) , \quad \mathbf{1}^n = Y \otimes (n - 1)$

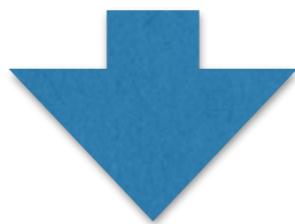
$\vdash \text{seq-fifo} : \forall n:\mathbb{N} \cdot X \rightarrow Y \mid_{n < 5}$

Solution exists: well-typed.

Enough?

# Example

seq-fifo =  $\lambda n:\text{Int} \cdot Tr_{n-1}(g_{n-1,1}; \text{fifo}^n) \mid_{n < 5}$



$\emptyset \mid \mathbf{1} \otimes (n - 1) = \mathbf{1}^n , \quad (n - 1) \otimes \mathbf{1} = X \otimes (n - 1) , \quad \mathbf{1}^n = Y \otimes (n - 1)$

$\vdash \text{seq-fifo} : \forall n:\mathbb{N} \cdot X \rightarrow Y \mid_{n < 5}$

seq-fifo :  $\forall n:\text{Int} \cdot 1 \rightarrow 1 \mid_{n < 5}$

# 3-Phase Solver

1. Simplify

arithmetic rewrites

2. Unify

most general unification (partial)

3. constraint  
solving

off-the-shelf constraint solver  
+ check uniqueness

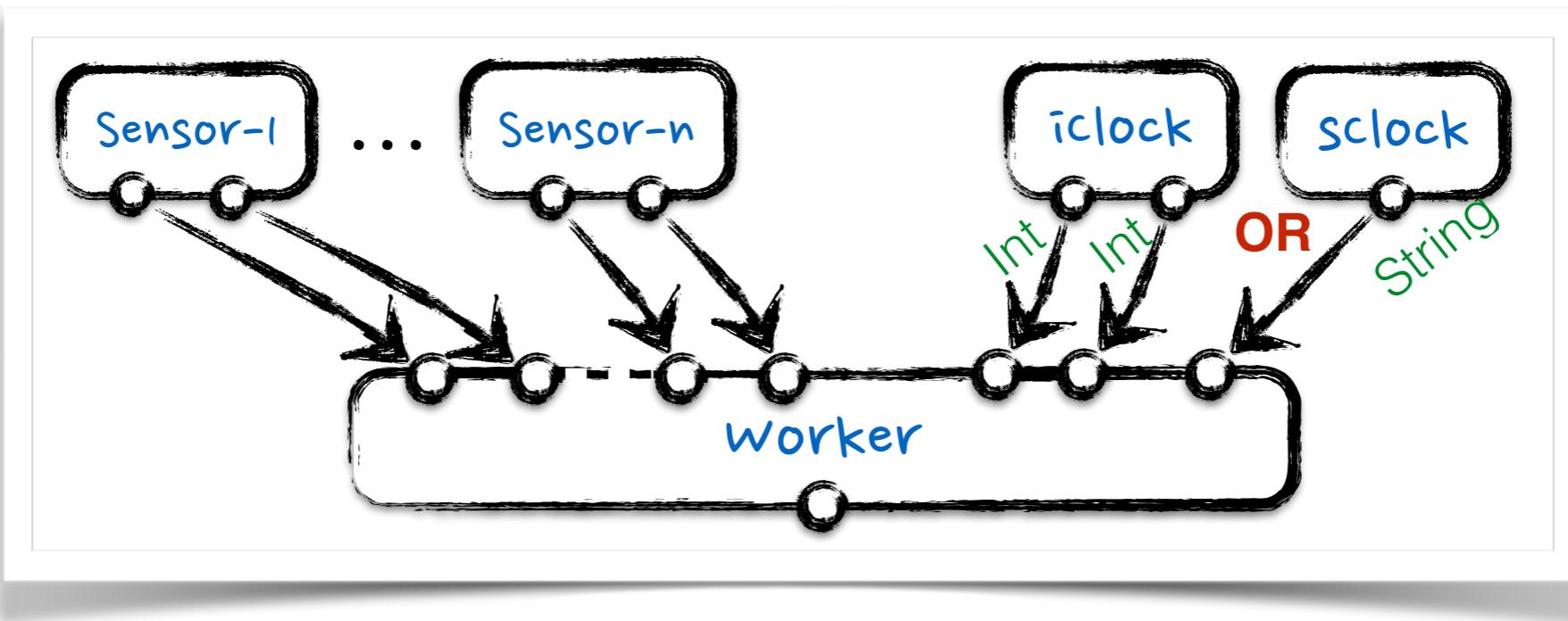
# 3-Phase Solver

```
1. Scala> debug(seqfifo)
\h. Tr_(n - 1){sym(n - 1,1) ; (fifo^n)}
  : ∀n:I . 1 -> 1
  - type-rules:  ∀n:I . x1 -> x2 | ((x1 + (n - 1)) == ...
  - [ unification: [x1:I -> 1, x2:I -> 1] ]
  - [ missing: true ]
  - substituted: ∀n:I . 1 -> 1 | ((1 + (n - 1)) == ...
  - simplified: ∀n:I . 1 -> 1
  - [ solution: Some([]) ]
  - post-solver: ∀n:I . 1 -> 1
  - instantiation: 1 -> 1

2. Scala>

3. Scala>
```

# Wrapping up


$$(\lambda^{n:\text{Int}} \cdot \text{Sensor}^n) \otimes (\lambda^{b:\text{Bool}} \cdot (\text{iclock} \oplus^b \text{sclock})) ; \text{worker}$$

**parameterised  
calculus**

**restriction  
+ composition**

**solver for type  
constraints**



Better name?

different protocol?

Mailing list

calendar

twitter feed

repositories

<http://arca.di.uminho.pt>