Typed Connector Calculus (& more…)

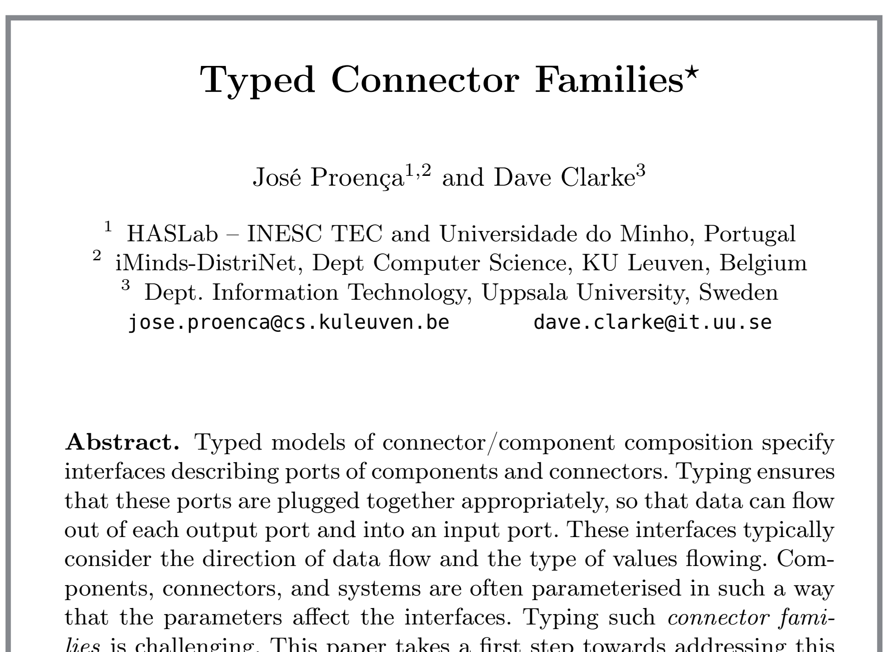
José Proença

InfoBlender
14 Jan 2016

For today

My research history… (briefly)

(some of) my ongoing interests



Typed Connector Families*

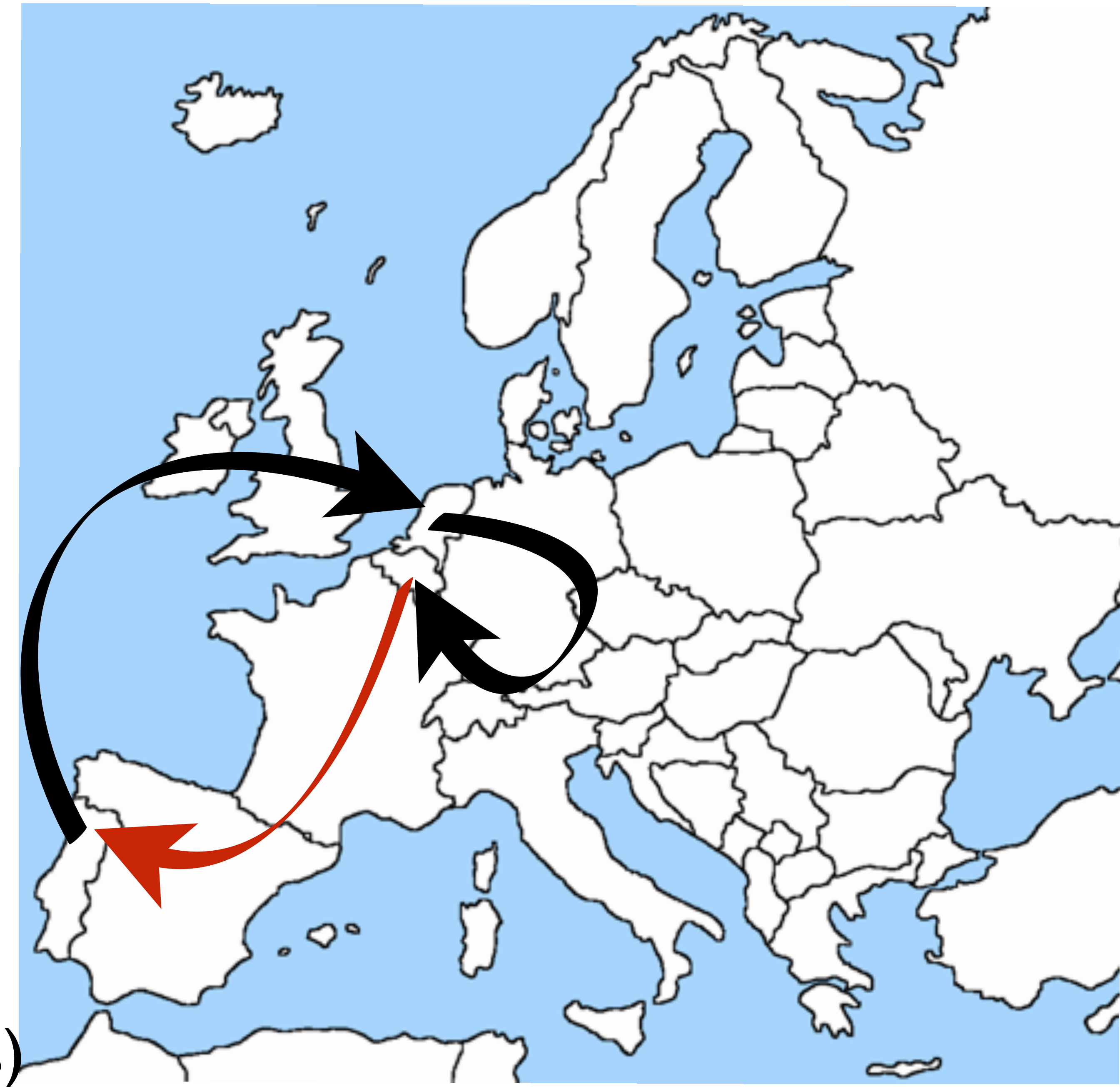
José Proença[1,2] and Dave Clarke[3]

[1] HASLab – INESC TEC and Universidade do Minho, Portugal
[2] iMinds-DistriNet, Dept Computer Science, KU Leuven, Belgium
[3] Dept. Information Technology, Uppsala University, Sweden
jose.proenca@cs.kuleuven.be dave.clarke@it.uu.se

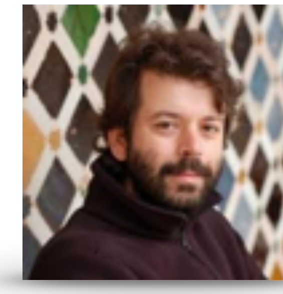
Abstract. Typed models of connector/component composition specify interfaces describing ports of components and connectors. Typing ensures that these ports are plugged together appropriately, so that data can flow out of each output port and into an input port. These interfaces typically consider the 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

My most recent work

- 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

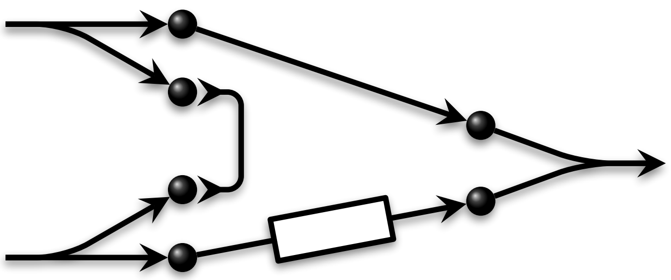

Dave clarke

- 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

HATS
Highly Adaptable and Trustworthy Software using Formal Models

SEVENTH FRAMEWORK PROGRAMME

delta programming

Feature Nets

petri nets with annotations

mTVL

language for describing variability

Wireless Sensor Networks

coordinating lightweight components

LooCI

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=JJjIeFs8f_4

Typed Connector Families

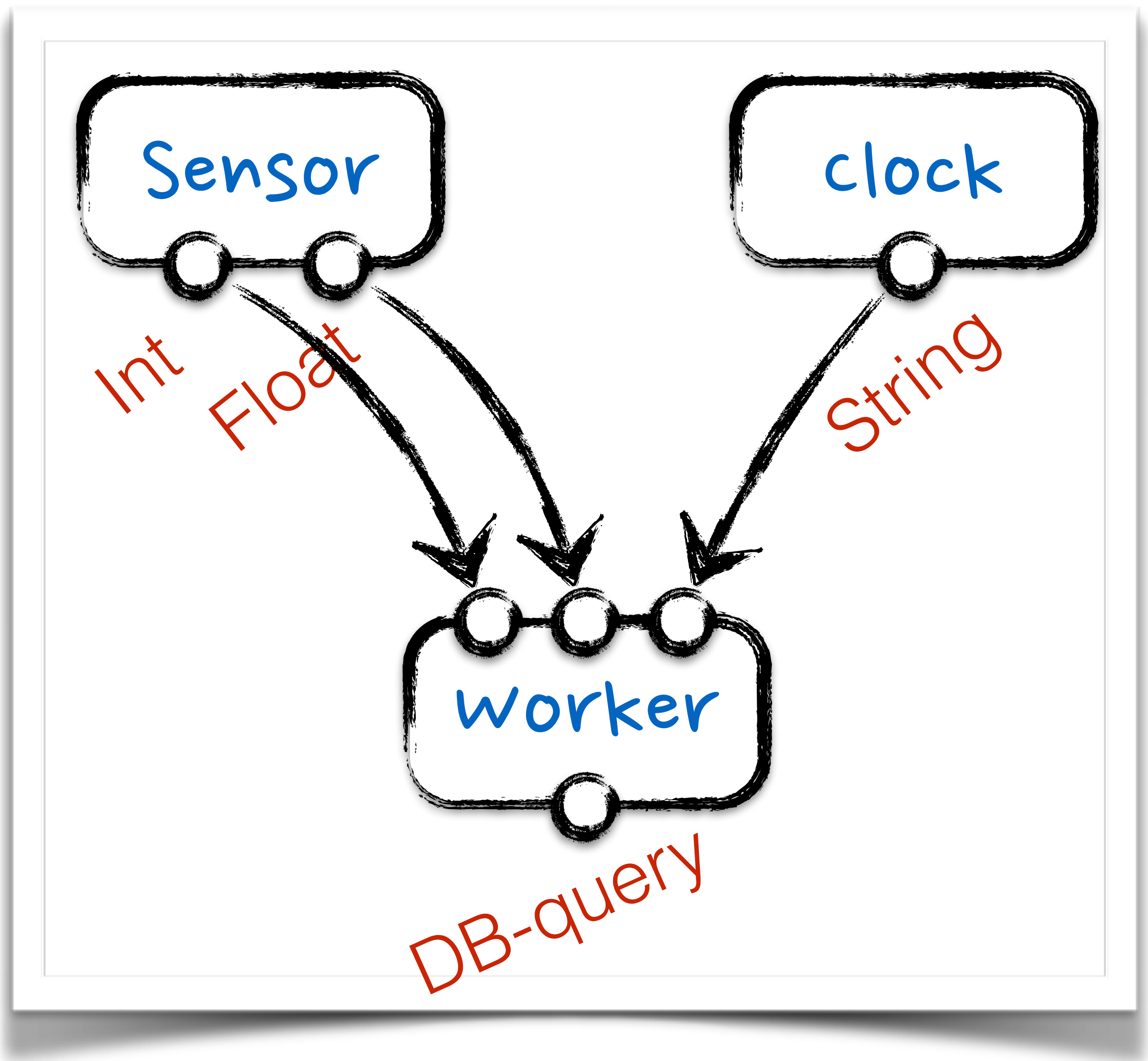
<u>José Proença</u> & Dave Clarke
(KU Leuven, Belgium) (UPPSALA University, Sweden)

DistriNet
iMinds KU LEUVEN

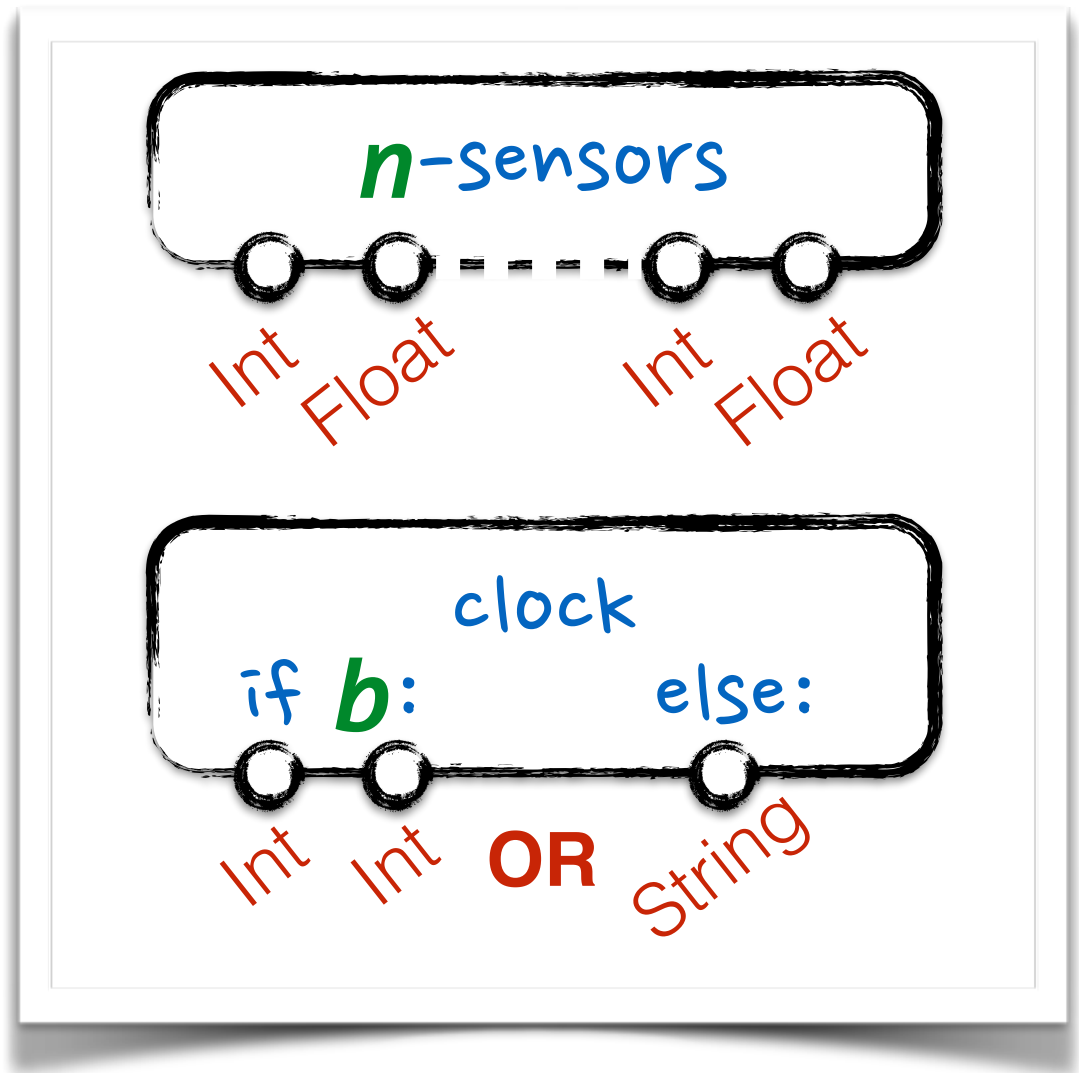
UPPSALA
UNIVERSITET

Paper+slides: http://facs15.proenca.org

Motivation

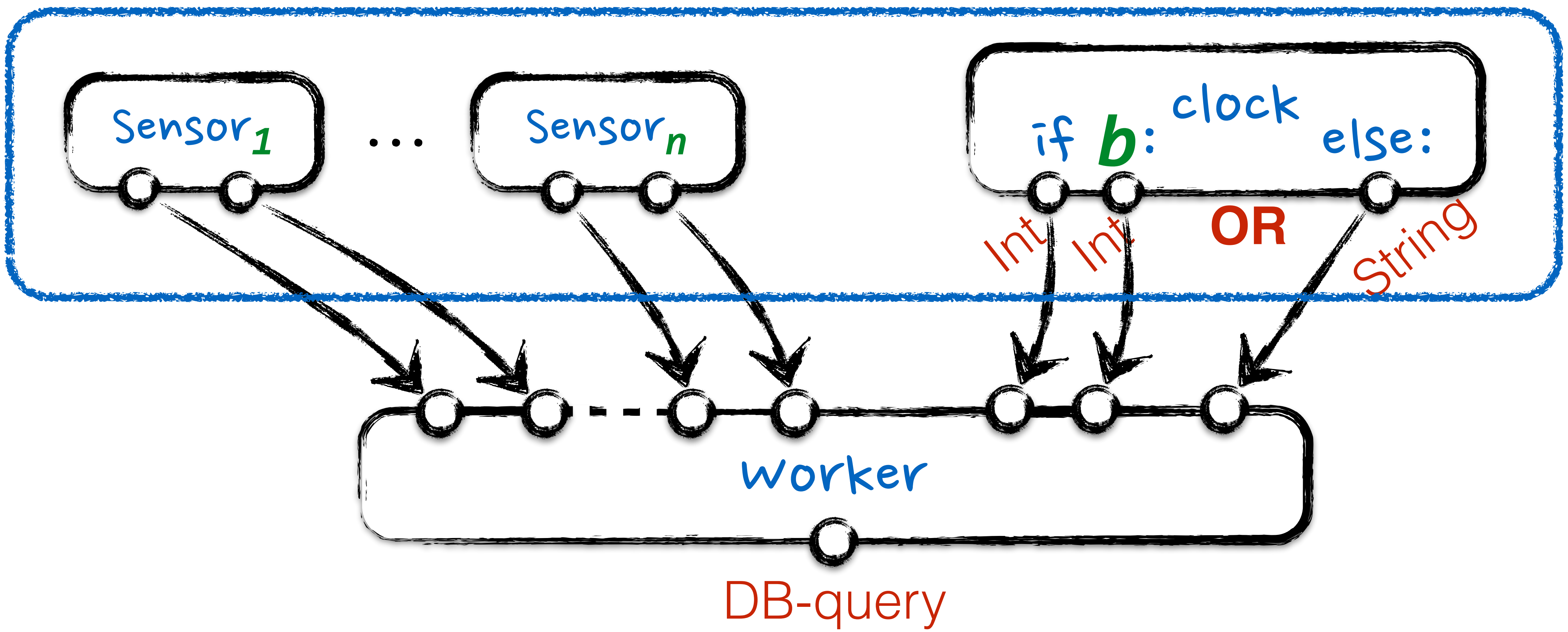


Static interfaces



Software product lines

Motivation



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

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

Outline

basic connector calculus

Sensor ⊗ clock ; worker

: 0 ⟶ DB-query

parameterised connector calculus

λn:Int · Sensor^n

: ∀ n:Int · 0 ⟶ ···

connector families

composing

parameterised cc

Type-checking approach

for untyped ports

Basic connector calculus

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

$$
\begin{array}{lll}
I, J ::= I \otimes J & & \text{tensor} \\
\qquad \mid\ \mathsf{0} & & \text{empty interface} \\
\qquad \mid\ A & & \text{port type}
\end{array}
$$

category with

a tensor (monoid)

symmetries

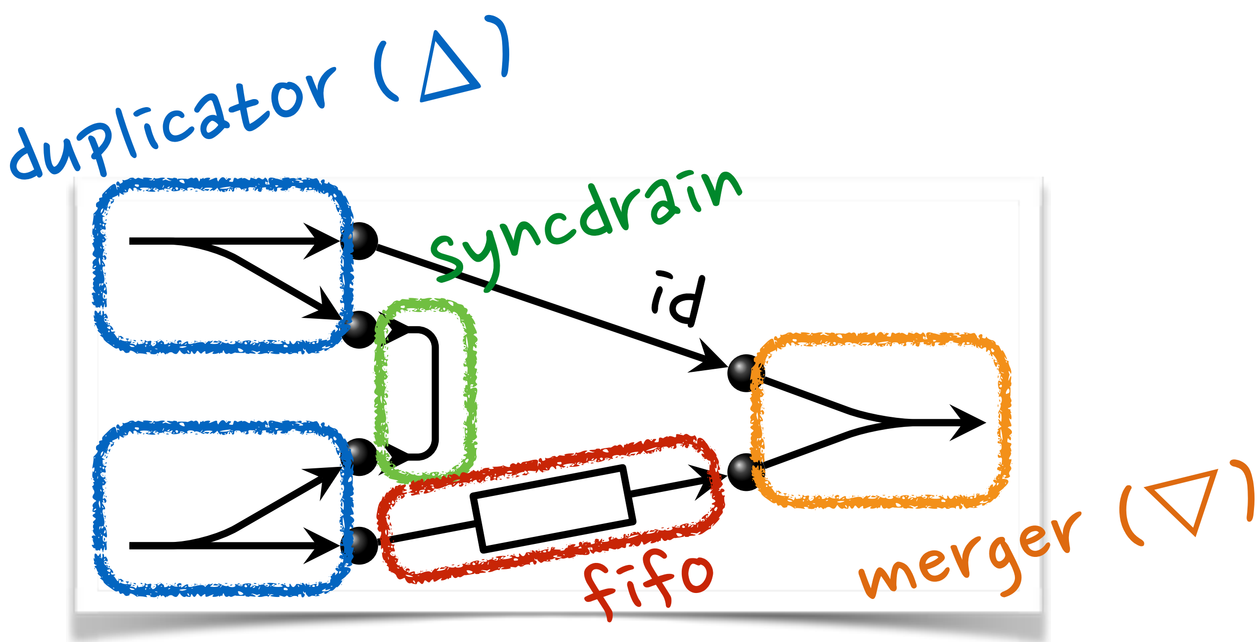
traces

connectors: morphisms

interfaces: objects

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

Reo example



$\Delta(a, a_1, a_2)$

$\Delta(b, b_1, b_2)$

syncdrain(a_2, b_1)

id(a_2, c_1)

fifo(b_2, c_2)

$\nabla(c_1, c_2, c)$

traditional:
with port names

$\Delta \otimes \Delta$;

id $\otimes$ syncdrain $\otimes$ fifo ;

∇

connector calculus

Visualisation of connectors

recall:

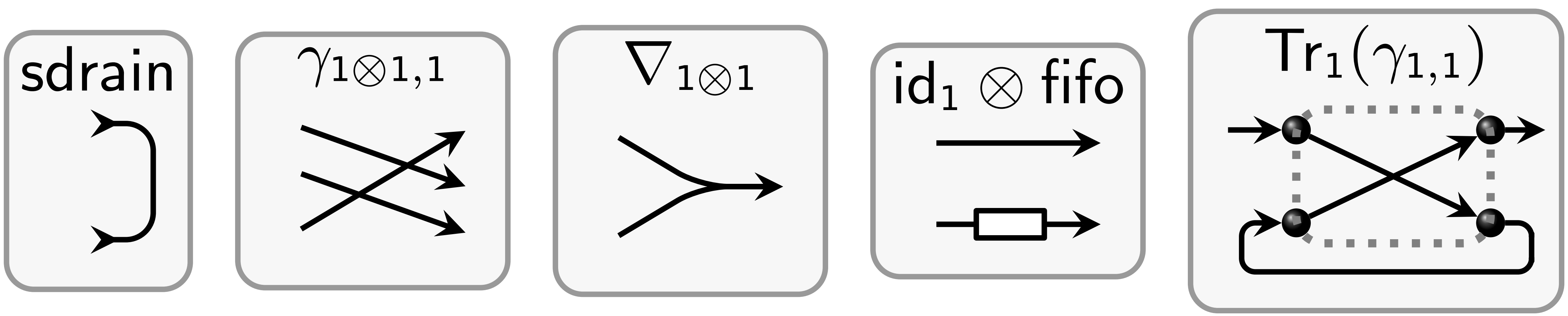
id_I

$\gamma_{I,J}$

$\mathsf{Tr}_I(c)$

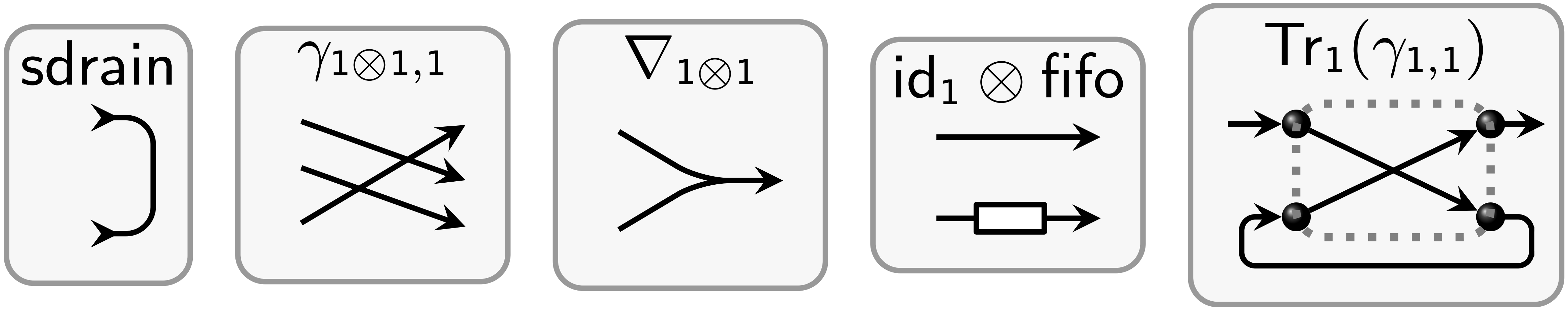
$$
\begin{array}{rcll}
I, J & ::= & I \otimes J & \text{tensor} \\
 & \mid & \mathsf{0} & \text{empty interface} \\
 & \mid & A & \text{port type}
\end{array}
$$

1 is some port type



Data goes always from left to right

Typing connectors



$1\otimes1 \longrightarrow 0$	$1\otimes1\otimes1 \longrightarrow 1\otimes1\otimes1$	$1\otimes1 \longrightarrow 1$	$1\otimes1 \longrightarrow 1\otimes1$	$1 \longrightarrow 1$

conn : $I \longrightarrow J$

IF c1 : $I_1 \longrightarrow J$ & c2 : $J \longrightarrow J_2$

THEN c1 ; c2 : $I_1 \longrightarrow J_2$

Constraint-based type rules

(sequence)

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

(trace)

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

Parameterised connector calculus

$\lambda x\!:\!\mathrm{Int} \cdot c$

$\mathrm{fifo}^{\mathrm{exp}}$ means: $\mathrm{fifo} \otimes \cdots \otimes \mathrm{fifo}$ ("exp" times)

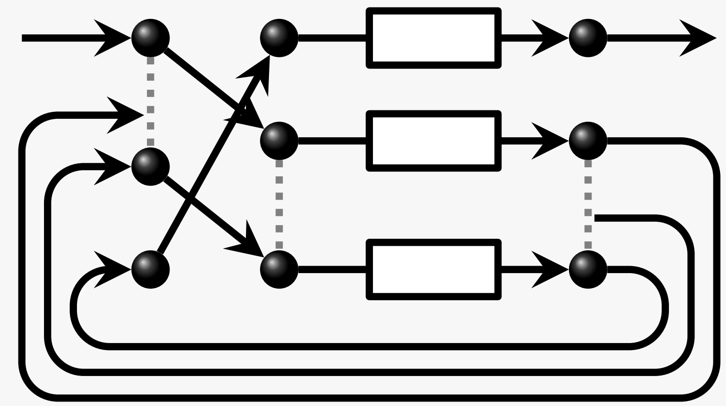
$(\Delta_{I^x})^{x \leftarrow \mathrm{exp}}$ means: $\Delta_{I^0} \otimes \cdots \otimes \Delta_{I^{\mathrm{exp}-1}}$

$\mathrm{fifo} \oplus^{\mathrm{exp}} \mathrm{drain}$ means: if (exp) then (fifo) else (drain)

$c ::= \ldots$	connectors	$I ::= \ldots$	interfaces
$\mid\ c^{x \leftarrow \alpha}$	n-ary parallel replication	$\mid\ I^{\alpha}$	n-ary parallel replication
$\mid\ c_1 \oplus^{\phi} c_2$	conditional choice	$\mid\ I \oplus^{\phi} J$	conditional choice
$\mid\ \lambda x : P \cdot c$	parameterised connector		
$\mid\ c(\phi)$	bool-instantiation	α, β	integer expressions
$\mid\ c(\alpha)$	int-instantiation	ϕ, ψ	boolean expressions

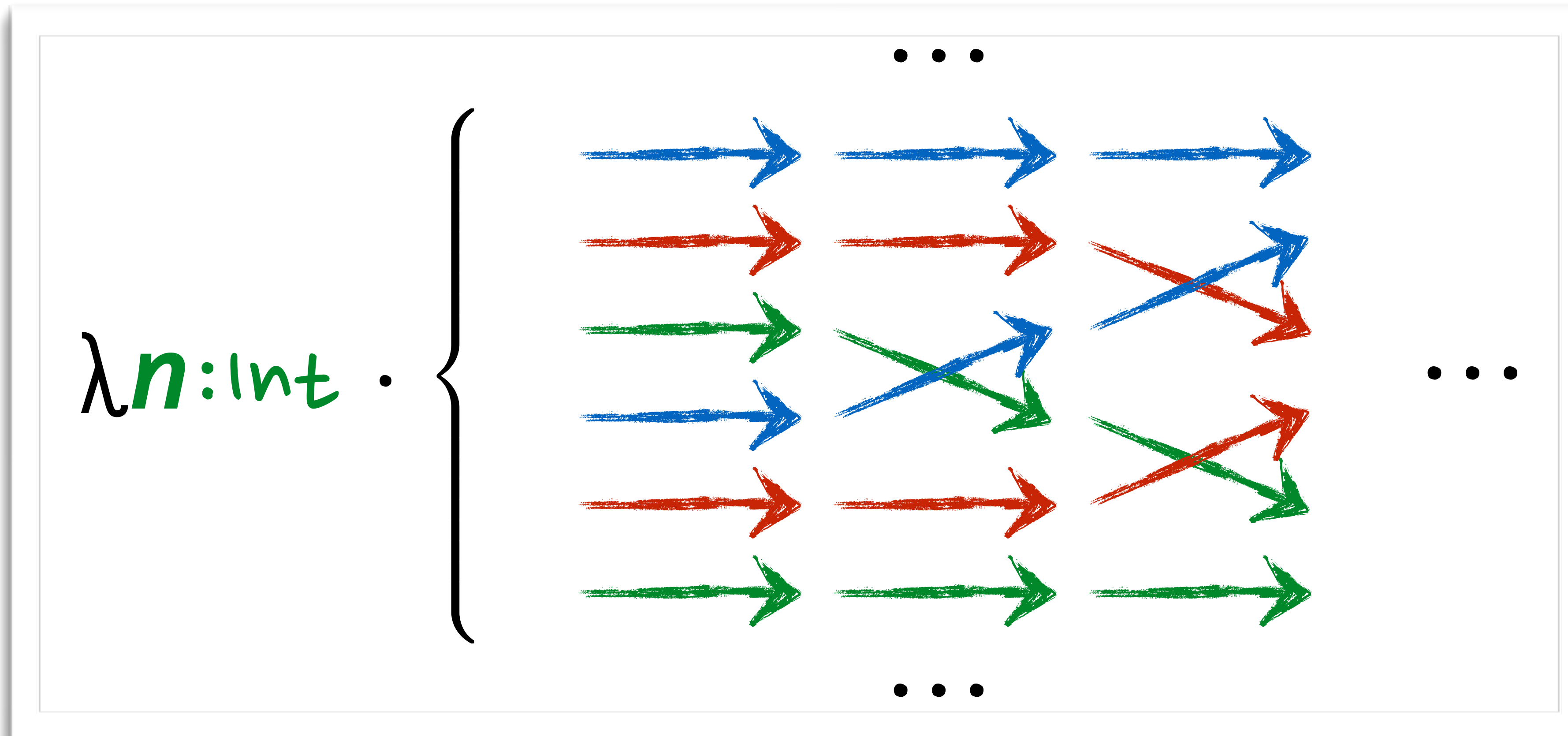
Example: *seq-fifo*

$$\textbf{\textit{seq-fifo}} = \lambda n : \mathbb{N} \cdot \mathsf{Tr}_{n-1} \left(\gamma_{n-1,1} \; ; \; \mathsf{fifo}^{n}\right)$$



$$\texttt{seq-fifo} : \forall\, n{:}\mathrm{Int} \cdot 1 \longrightarrow 1$$

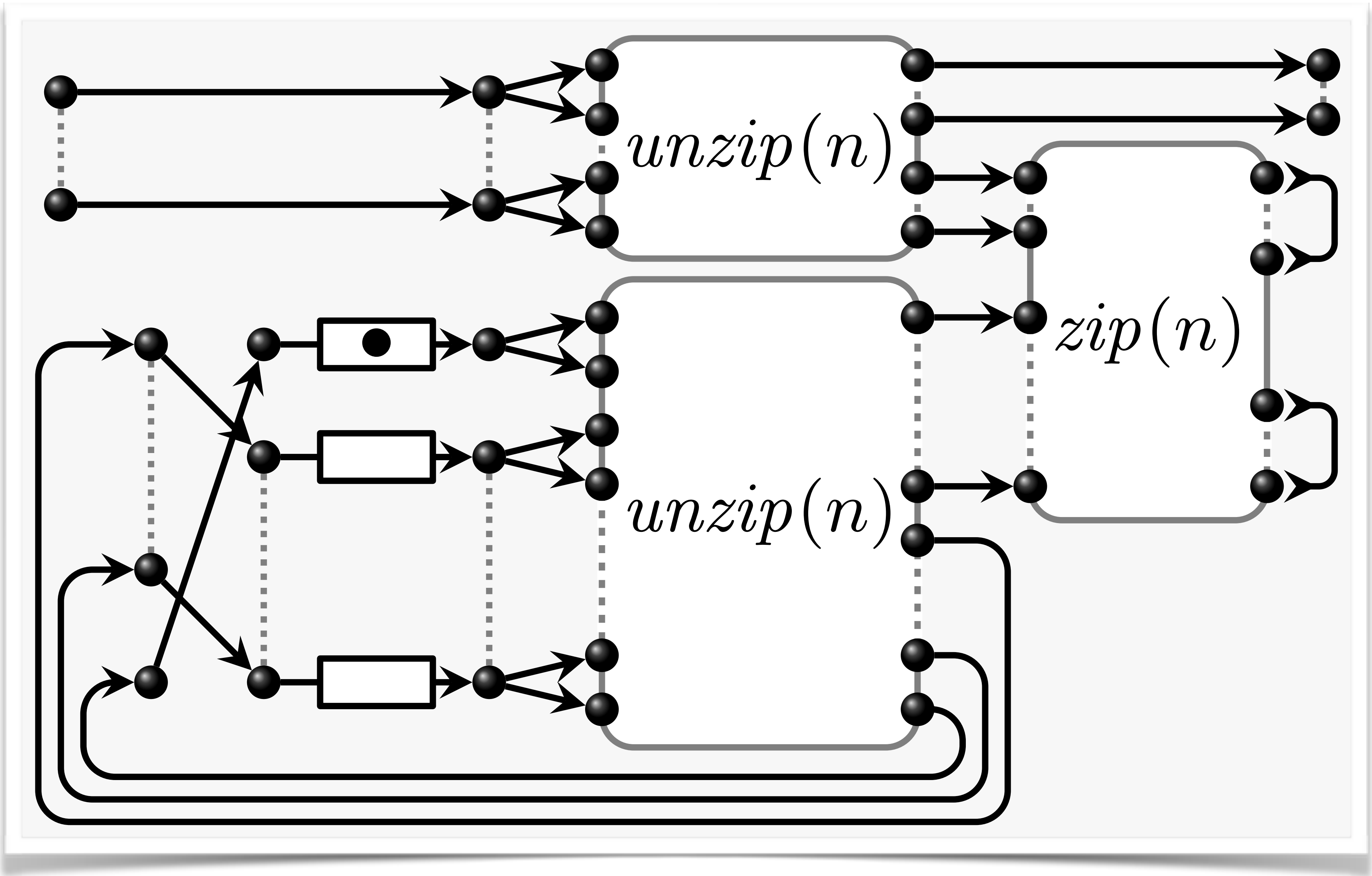
Example - *zip*



$$\boldsymbol{zip} = \lambda n : \mathbb{N} \cdot \mathsf{Tr}_{2n^2-2n}\big(\gamma_{2n^2-2n,2n} ; (\mathsf{id}_{n-x} \otimes \gamma^{x}_{1,1} \otimes \mathsf{id}_{n-x})^{x \leftarrow n} \big)$$

zip : $\forall\, n{:}\mathrm{Int} \cdot (1^n)^2 \longrightarrow (1^2)^n$

Example - *sequencer*



sequencer : $\forall\, n{:}\mathrm{Int} \cdot 1^n \longrightarrow 1^n$

Connector Families

(restriction)

$$\frac{\Gamma \mid \phi \vdash \psi \qquad \Gamma \mid \phi, \psi \vdash c : T}{\Gamma \mid \phi \vdash c|_{\psi} : T|_{\psi}}$$

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

(fam-sequence)

$$\frac{\Gamma \mid \phi \vdash c_1 : \forall \overline{x_1 : T_1} \cdot I_1 \to J_1 |_{\psi_1} \qquad \Gamma \mid \phi \vdash c_2 : \forall \overline{x_2 : T_2} \cdot I_2 \to J_2 |_{\psi_2} \qquad \overline{x_1} \cap \overline{x_2} = \varnothing}{\Gamma \mid \phi, J_1 = I_2 \vdash c_1 \;;\; c_2 : \forall \overline{x_1 : T_1}, \overline{x_2 : T_2} \cdot I_1 \to J_2 |_{\psi_1, \psi_2}}$$

$(\lambda x{:}\mathrm{Int} \cdot c1) \;;\; (\lambda y{:}\mathrm{Int} \cdot c2) \;:\; \forall x{:}\mathrm{Int}, y{:}\mathrm{Int} \cdot I_1 \longrightarrow J_2$

Solving Type Constraints

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

<u>**c**</u> is well-typed if:

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

the type rules yield T, φ, ψ

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

Solving Type Constraints

$\underline{c}$ is well-ty

given

the ty

such th

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

$$(\![0]\!) = 0$$
$$(\![1]\!) = 1$$
$$(\![I \otimes J]\!) = (\![I]\!) + (\![J]\!)$$
$$(\![I^\alpha]\!) = (\![I]\!) * \alpha$$

```
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 = $\lambda n{:}Int \cdot Tr_{n-1}(\gamma_{n-1,1} \,;\, fifo^n)\ |_{n<5}$

$$\varnothing \mid \mathbf{1} \otimes (n-1) = \mathbf{1}^n \ ,\ (n-1) \otimes \mathbf{1} = X \otimes (n-1) \ ,\ \mathbf{1}^n = Y \otimes (n-1)$$
$$\vdash \text{seq-fifo} \ : \ \forall n : \mathbb{N} \cdot X \to Y \,|_{n<5}$$

Solution exists: well-typed.

Enough?

Example

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

$$\varnothing \mid \mathbf{1} \otimes (n-1) = \mathbf{1}^n \;\;,\;\; (n-1) \otimes \mathbf{1} = X \otimes (n-1) \;\;,\;\; \mathbf{1}^n = Y \otimes (n-1)$$
$$\vdash \text{seq-fifo} \;:\; \forall n : \mathbb{N} \cdot X \to Y \,|_{n<5}$$

$$\text{seq-fifo} \;:\; \forall\, n{:}\mathrm{Int} \cdot 1 \longrightarrow 1 \;|_{n<5}$$

3-Phase Solver

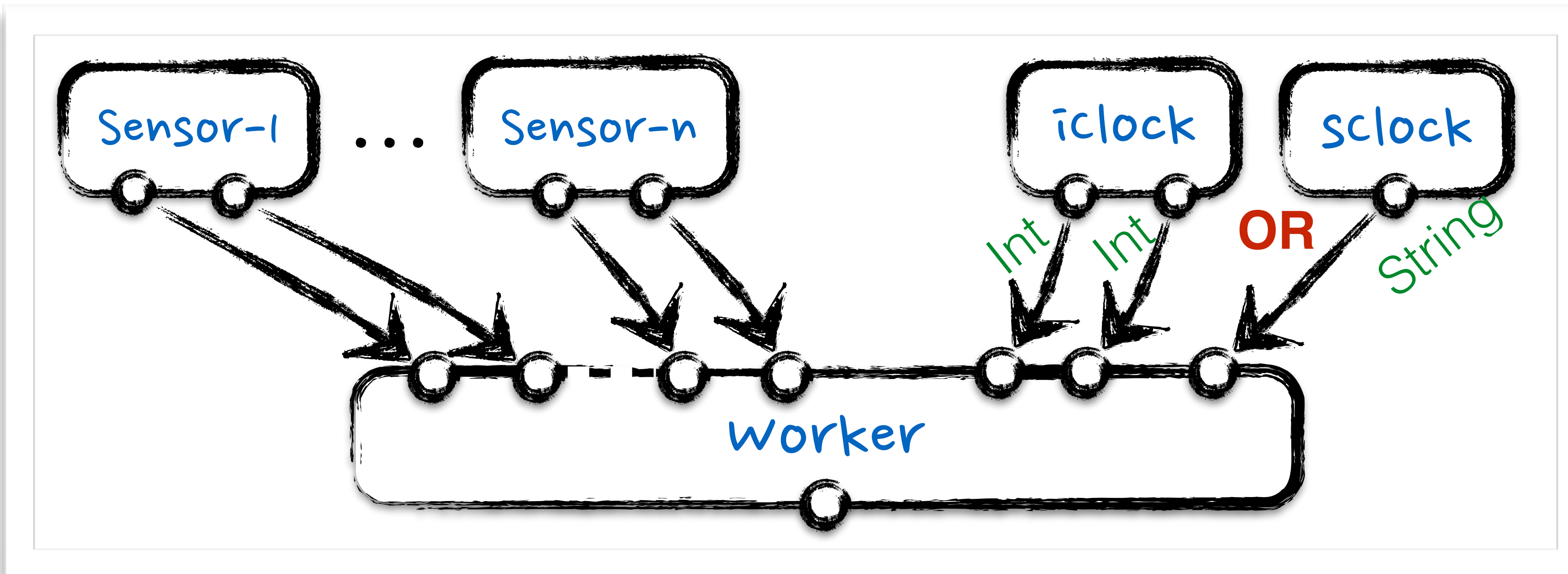
1. Simplify	arithmetic rewrites
2. unify	most general unification (partial)
3. constraint solving	off-the-shelf constraint solver + check uniqueness

3-Phase Solver

```
scala> debug(seqfifo)
\n.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

scala>
```

Wrapping up



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

parameterised calculus

restriction + composition

solver for type constraints