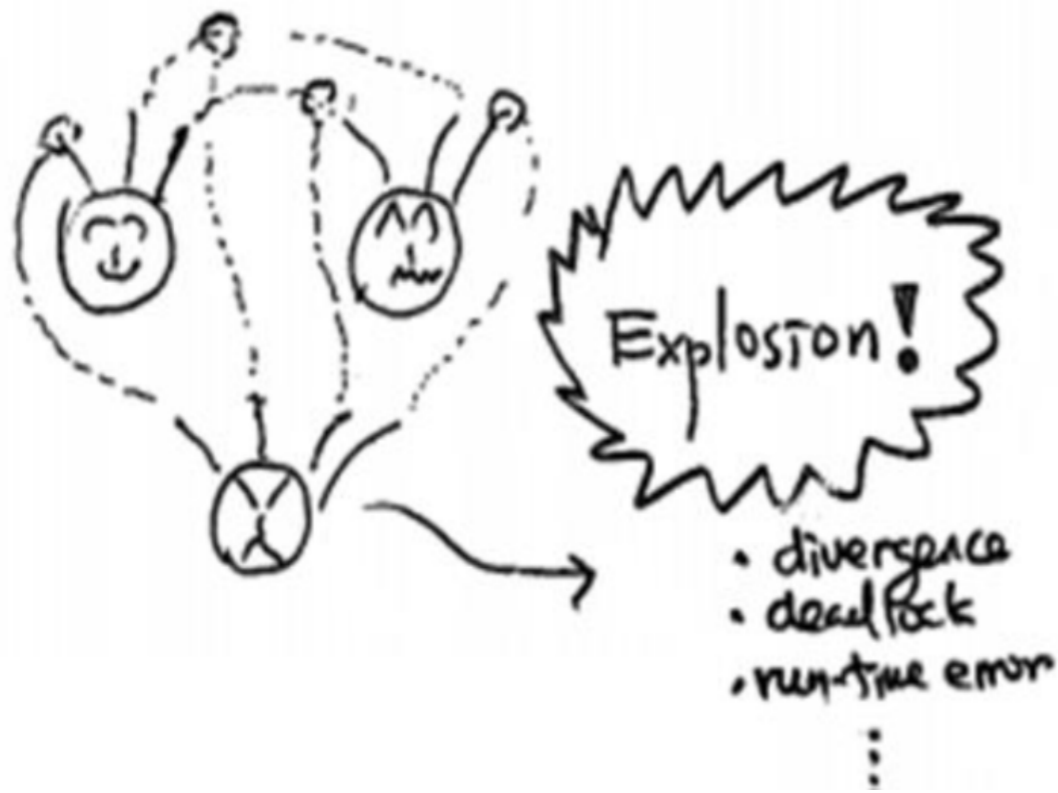


# Session Types and Open Problems



Nobuko Yoshida  
Betty Meeting 6th October 2016

# The Kohei Honda Prize for Distributed Systems Queen Mary, University of London

Posted with permission from QMUL on 17<sup>th</sup> Dec 2013. [Original article](#) written by Edmund Robinson.

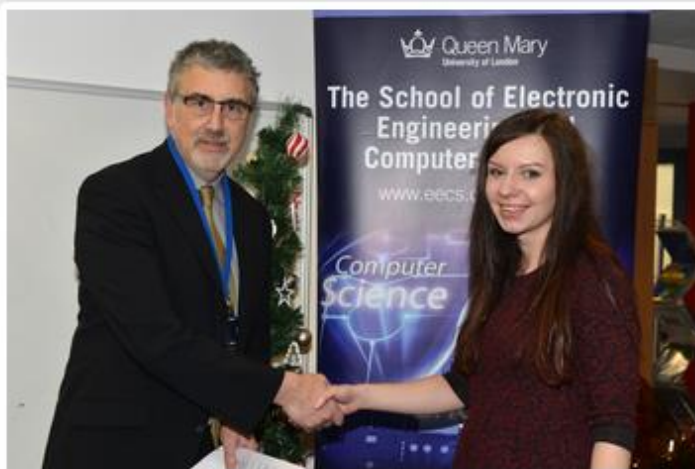
This prize was instituted in 2013 and is awarded annually to one undergraduate student and one postgraduate student in recognition of their achievement in applying the highest quality scientific and engineering principles in the broad area of Distributed Systems. This is the area in which Dr Honda concentrated most of his teaching, and it is also the area in which he conducted his research. Its primary funding comes from a donation from his family, who wished to commemorate Dr Honda in this way. Additional funding has come from Dr Honda's own ETAPS Award. This prize is sponsored by Springer Verlag, and awarded annually by the ETAPS committee in recognition of an individual's research contribution. Dr Honda received the first such award posthumously, and the awarding panel expressed a wish that the funding be used to supplement this prize fund. The laudation for this award, written by Dr Honda's colleague, Prof Vladimiro Sassone is included later.

## About Dr Honda

Kohei Honda was born and lived the first part of his life in Japan. Like many scientists he was fascinated by the idea of finding basic explanatory theories, like the physicists looking for grand unified theories of the universe. Kohei, though, was passionately interested in finding the right basic explanatory theory for the process of computation. Most academics agree that the basic theory



## Winners 2013



**Ms Anna Pawlicka**

2013 winner (Undergraduate) source: QMUL



**Mr. Valdmir Negacevshi**

2013 winner (Postgraduate) source: QMUL

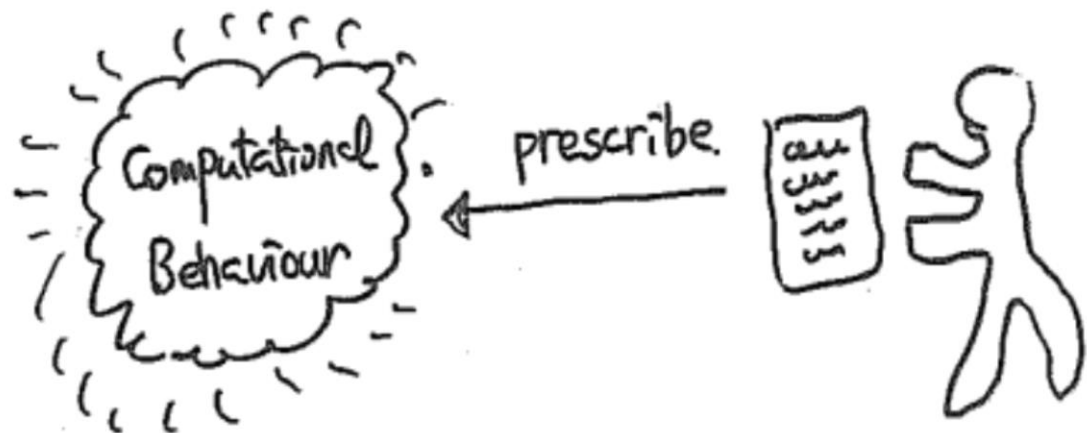
# Idioms for Interaction

— Invitation to Hacking in  $\pi$ -calculus —

Programming languages are tools which offer frameworks of abstraction for such activities – promoting or limiting them

- Imperative
- Functional
- Logical

- Programs : prescription of computational behaviours based on a certain abstraction.



# On Programs and Programming

- The most fundamental element of a PL in this context is a set of operations it is based on:

Imperative: assignment, jump.

Functional:  $\beta$ -reduction.

Logical: unification.

- Another element is how we can combine, on structure, these operations:

Imperative: sequential composition, if-then-else, while, procedures, module, ....

Functional: application, product, union, recursion, modules, .....

# UNSTRUCTURED:

```

data _stkl, 48 } stacks.
data _stkr, 48 }
data _i, 4 } index
data _j, 4 }
data _l, 4 } left/right limit
data _r, 4 }
data _x, 4 } pivot
data _w, 4 } temporary value.
data _s, 4 } stack pointer
data _a, 48 } table to be sorted.

mov $0, _s
mov $0, _stkl } 2nd instruction
mov $11, _stkr

L1: mov _s, rax } Top loop.
    mov _stkl(, rax, 4), rcx } l := stkl(s)
    mov rcx, _l
    mov _s, rax
    mov _stkr(, rax, 4), rcx } r := stkr(s)
    mov rcx, _r
    dec _s

L2: mov _l, rcx } j := l.
    mov rcx, _i
    mov _r, rcx } j := r.
    mov rcx, _j
    mov _l, rcx } rdx := l+r
    add _r, rcx
    mov rcx, rax } rdx := (l+r)/2
    mov rax, rdx
    shr $31, rdx
    add rdx, rax
    mov rax, rdx
    sar $1, rdx
    mov _a(, rdx, 4), rcx } x := a(rdx)
    mov rcx, _x

L3: mov _i, rax } third loop
    mov _a(, rax, 4), rdx } rdx := a(i)
    cmp rdx, _x
    jle L4
    inc _i
    jmp L3 } if rdx >= x goto (4)
    } else i := i+1
    } goto (3) (loop)

L4: mov _j, rax
    mov _a(, rax, 4), rdx } rdx := a(j)

```

# STRUCTURED:

```

Var a: array[MAX] of int;

Procedure sort(l, r: int);
  Var i, j, x: int;
  i := l; j := r;
  x := [(l+r) div 2];
  • Choose a pivot.
  repeat
    while a[i] < x do i := i+1 end
    while a[j] > x do j := j-1 end
    if i <= j then swap(i, j); i := i+1; j := j-1; end
  until i > j;
  if l < j then sort(l, j);
  if l < i then sort(i, r);
  end
  } Partition into two parts.
  } Recursively sort two parts.

Procedure swap(i, j: int)
  Var w: int;
  w := a[i]; a[i] := a[j]; a[j] := w;
end

```

# Quicksort in pure lambda:

$((\lambda xy. y(xy))(\lambda xy. y(xy)))\lambda q. \lambda l.$   
 $((\lambda x. x(\lambda xy. x))l)(\lambda x. x)$  } if l is not then nil.  
 $((\lambda xy. y(xy))(\lambda xy. y(xy)))(\lambda c. \lambda xy. x((\lambda x. x(\lambda py. x))x)y$  } concat.  
 $((\lambda xy. \lambda z. z(\lambda xy. y)xy)((\lambda x. x(\lambda xyz. y))x)(c((\lambda x. x(\lambda xyz. z))x)y$   
 $(q(\lambda xy. y(xy))(\lambda xy. y(xy)))(\lambda f. \lambda px. ((\lambda x. x(\lambda py. x))x)(\lambda x. x))$  } sort and  
 $(p((\lambda x. x(\lambda xyz. y))x))((\lambda xy. \lambda z. z(\lambda xy. y)xy)x$  Filter  
 $(f((\lambda x. x(\lambda xyz. z))x)))(f((\lambda x. x(\lambda xyz. z))x)))$   
 $(\lambda y. (((\lambda xy. y(xy))(\lambda xy. y(xy)))\lambda f'. \lambda xy. ((\lambda x. x(\lambda py. x))y$  } (λy, LTy, Car  
 $(\lambda xy. y)(((\lambda x. x(\lambda py. x))x)(\lambda xy. y)(f'((\lambda x. x(\lambda py. y))x)((\lambda x. x(\lambda py. y)$   
 $y((\lambda x. x(\lambda xyz. y))l))((\lambda x. x(\lambda xyz. z))l)))$  } cdr l  
 $((\lambda xy. \lambda z. z(\lambda xy. y)xy)((\lambda x. x(\lambda xyz. y))l)$  } cons (car l)  
 $(q((\lambda xy. y(xy))(\lambda xy. y(xy)))(\lambda f. \lambda px. ((\lambda x. x(\lambda py. x))x)$  } sort and  
 $(\lambda x. x)(p((\lambda x. x(\lambda xyz. y))x))((\lambda xy. \lambda z. z(\lambda xy. y)xy)x$  Filter  
 $(f((\lambda x. x(\lambda xyz. z))x)))(f((\lambda x. x(\lambda xyz. z))x)))$   
 $(\lambda y. (((\lambda xy. y(xy))(\lambda xy. y(xy)))\lambda f''. \lambda xy. ((\lambda x. x(\lambda py. x))x$  } λy, Mxy  
 $((\lambda x. x(\lambda py. x))y)(\lambda xy. x)(\lambda xy. y))$  Car l  
 $((\lambda x. x(\lambda py. x))y)(\lambda xy. x)(f''((\lambda x. x(\lambda py. y))x$   
 $((\lambda x. x(\lambda py. y))y))y((\lambda x. x(\lambda xyz. y))l))((\lambda x. x(\lambda xyz. z))l))))$  } cdr l

# Quicksort with combinators:

$Y(\lambda f. \lambda l.$   
 $(\text{Isnil } l)$   
 $(\text{Concat } (f \text{ Filter } (\lambda y. \text{LTy } (\text{Car } l))$   
 $(\text{Cdr } l))$   
 $(\text{Cons } (\text{Car } l)$   
 $(f \text{ Filter } (\lambda y. \text{MEy } (\text{Car } l)$   
 $(\text{Cdr } l))))))$

$I = \lambda x. x \quad T = \lambda xy. x \quad F = \lambda xy. y \quad Y = (\lambda xy. y(xy))(\lambda xy. y(xy))$   
 $\text{cons} = \lambda xy. \lambda z. z F xy \quad \text{isnil} = \lambda x. x T$   
 $\text{car} = \lambda x. x (\lambda yz. y) \quad \text{cdr} = \lambda x. x (\lambda yz. z)$   
 $\text{concat} = Y(\lambda c. \lambda xy. x (\text{isnil } x) y (\text{cons } (\text{car } x) (c (\text{cdr } x)) y)$   
 $\text{filter} = Y(\lambda f. \lambda px. (\text{isnil } x) I (p (\text{car } x)) (\text{cons } x (\text{filter } (\text{cdr } x))))$   
 $\text{iszero} = \lambda x. x T \quad \text{pred} = \lambda x. x F \quad (\text{filter } (\text{cdr } x))$   
 $\text{LT} = Y(\lambda f. \lambda xy. (\text{iszero } y) F ((\text{iszero } x) F (f (\text{pred } x) (\text{pred } y)))$   
 $\text{ME} = Y(\lambda f. \lambda xy. (\text{iszero } x) ((\text{iszero } y) I F) ((\text{iszero } y) (T) (f (f (pred x) (pred y)))))$

## Quicksort in ML:

```
fun qs nil: int list = nil
```

```
| qs (x::r) = let val small =  
                filter (fn y => y < x) r  
                and large =  
                filter (fn y => y >= x) r
```

```
    in qs small @ [x] @ qs large
```

```
    end
```

```
fun filter p nil = nil
```

```
| filter p (x::r) =
```

```
    if p x then x := filter p r
```

```
    else filter p r
```

# The $\pi$ -calculus as a Descriptive Tool

$$\lambda \quad M ::= x \mid \lambda x.M \mid MN.$$

$$\pi \quad P ::= \sum \pi_i.P_i \mid P|Q \mid \nu x.P \mid !P \mid \emptyset.$$

$$\text{with } \pi ::= x(\bar{y}) \mid \bar{x}(y).$$

$\lambda$  in  $\pi$

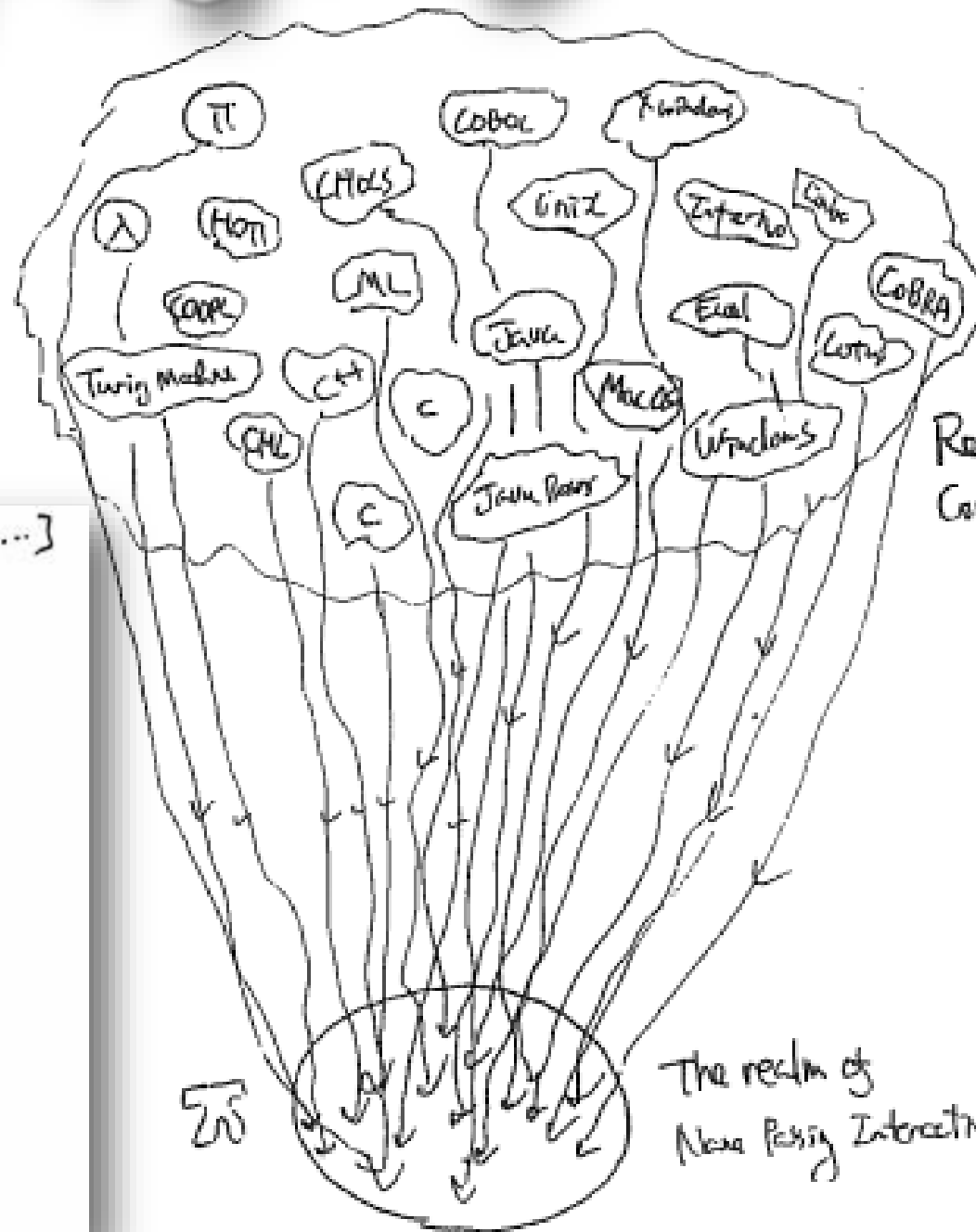
$$[x]_u \stackrel{\text{def}}{=} \bar{x}(u).$$

$$[\lambda x.M]_u \stackrel{\text{def}}{=} u(x). [M]_u.$$

$$[MN]_u \stackrel{\text{def}}{=} (\nu f_x) ([M]_f \mid \bar{f}(x)) \mid [x=N]_u$$

$$\text{with } [x=N]_u \stackrel{\text{def}}{=} !x(u). [N]_u.$$

# \* Examples of Representable Computation.



- $\lambda$ -calculus [MPW89, Milner90, Milner92, ...]
- Concurrent Object [Walker91]
- $\omega$ -order term passing [Sangiorgi 92]
- Various data structures [Milner 92, ...]
- Proof Nets [Bellon and Scott 93]
- Abstract "constant" interaction [HRS4]
- Strategies on Games [HO95]

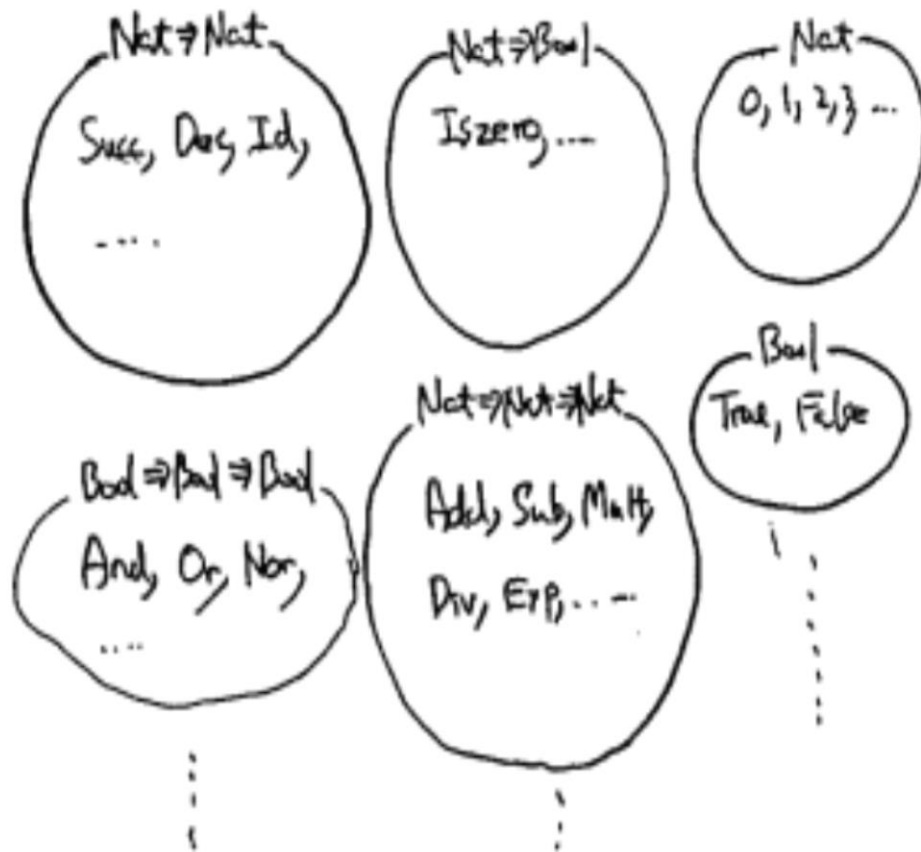
⋮

# The Role of Types in $\pi$ -calculus.

- (classification) How can we classify name-passing interactive behaviours, i.e. behaviours representable in  $\pi$ -calculus? What classes ("types") of behaviours can we find in the calculus?

- (safety) Is this program/system in the safe (or correct, relevant,...) classes of behaviours? Can the safety be preserved compositionally?

# Functional Types



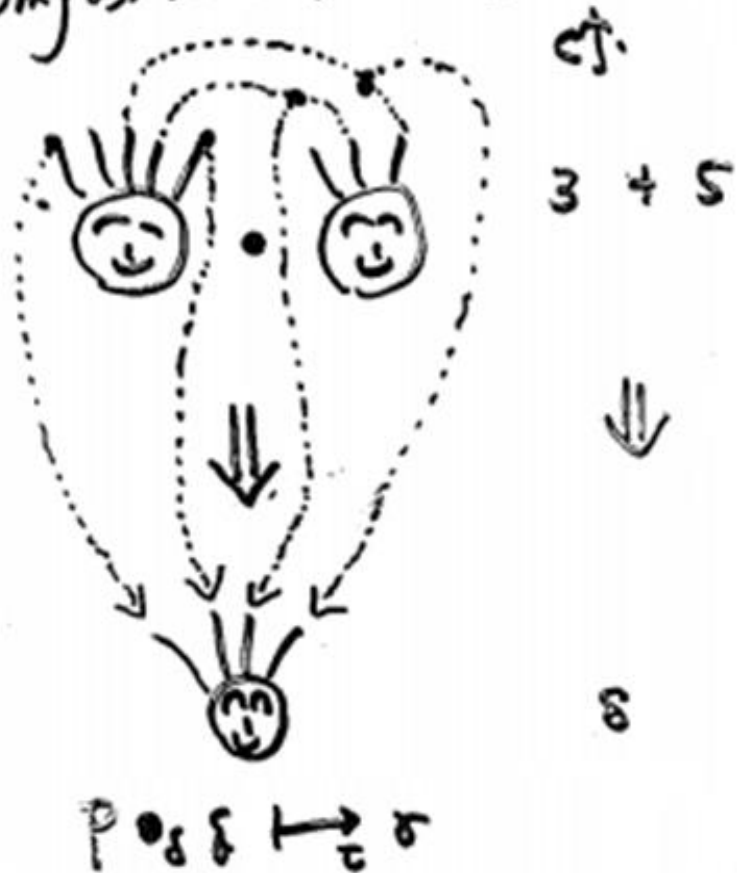
with operation!

$$\left\{ \begin{array}{l} f : \alpha \Rightarrow \beta \bullet e : \alpha = f \bullet e : \beta. \\ \text{else undefined.} \end{array} \right.$$

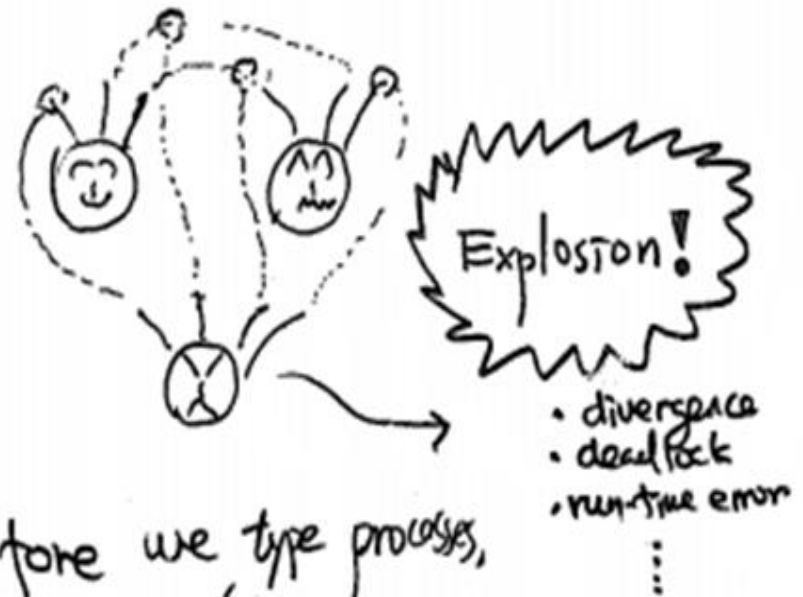
function application.

# Process Types

- When it comes to processes, composition becomes:



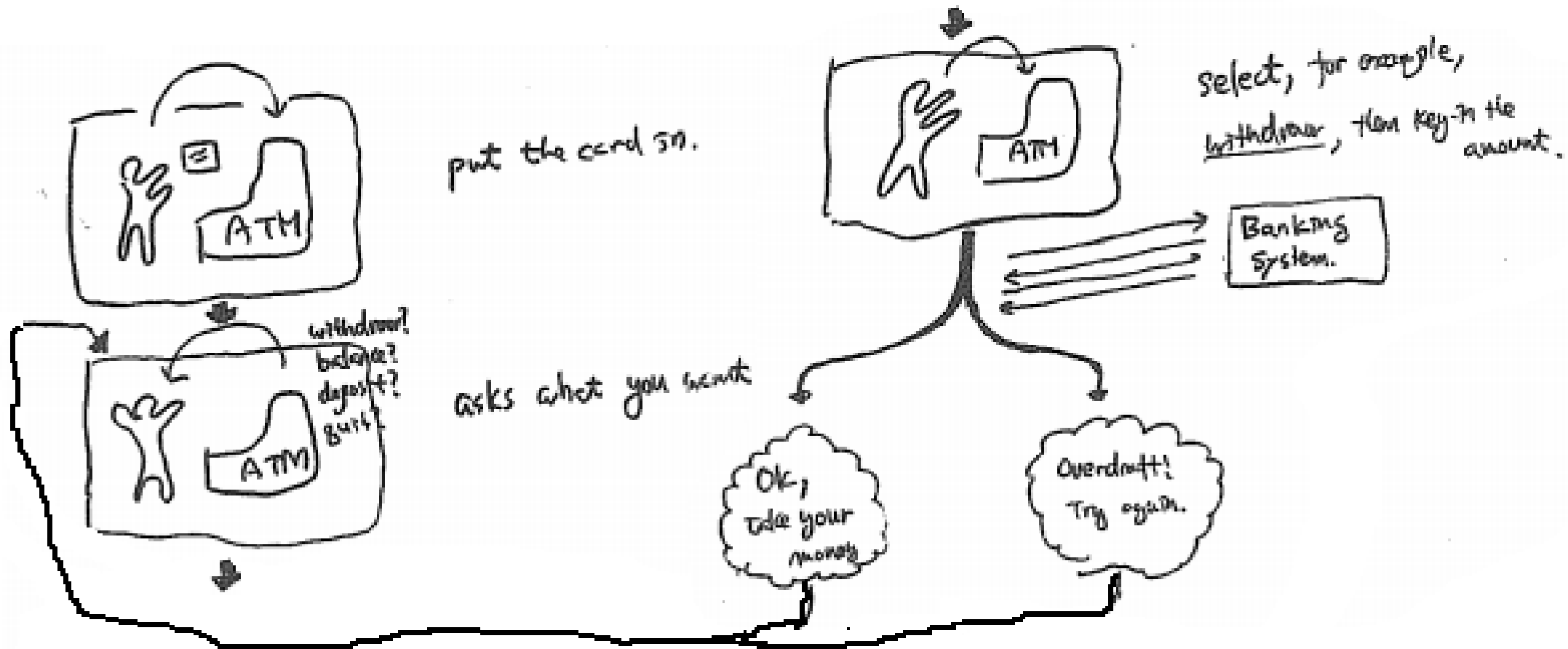
- But some composition is dangerous!



- Therefore we type processes,

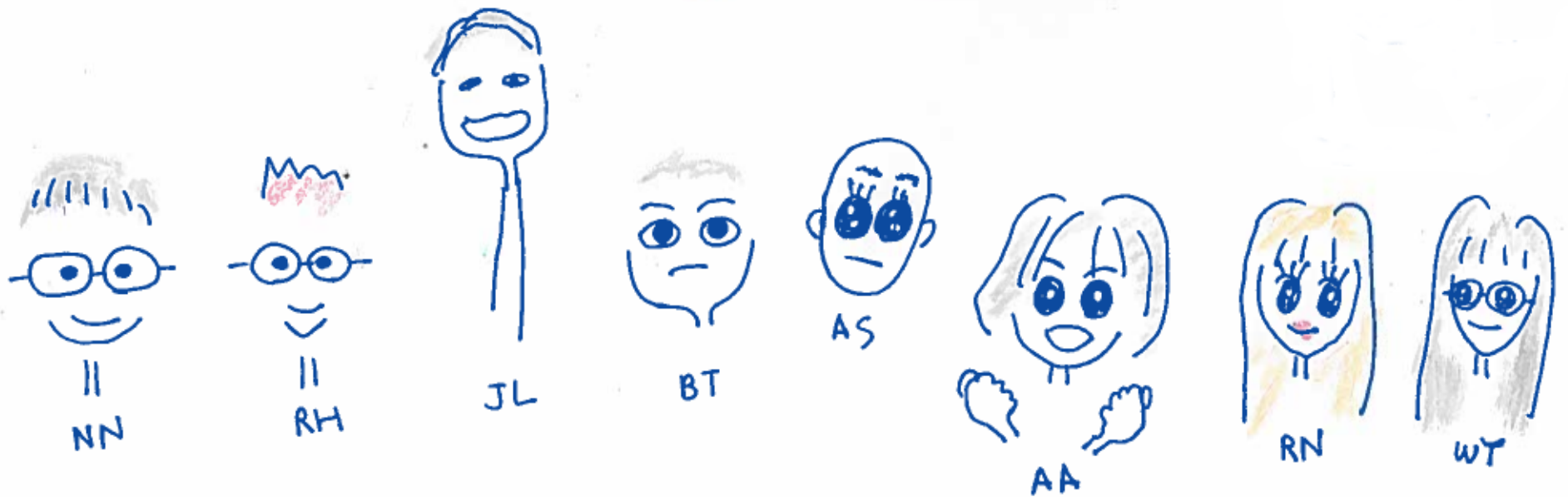


# Implementing ATM





# Session Type Mobility Group



[www.mrg.doc.ic.ac.uk](http://www.mrg.doc.ic.ac.uk)

# Selected Publications 2015/2016



- **[POPL'17]** Julien Lange, Nicholas Ng, Bernardo Toninho, NY: Fencing off Go: Liveness and Safety for Channel-based Programming.
- **[FPL'16]** Xinyu Niu , Nicholas Ng , Tomofumi Yuki , Shaojun Wang , NY, Wayne Luk : EURECA Compilation: Automatic Optimisation of Cycle-Reconfigurable Circuits.
- **[ECOOP'16]** Alceste Scala, NY: Lightweight Session Programming in Scala
- **[CC'16]** Nicholas Ng, NY: Static Deadlock Detection for Concurrent Go by Global Session Graph Synthesis.
- **[FASE'16]** Raymond Hu, NY: Hybrid Session Verification through Endpoint API Generation.
- **[TACAS'16]** Julien Lange, NY: Characteristic Formulae for Session Types.
- **[ESOP'16]** Dimitrios Kouzapas, Jorge A. Pérez, NY: On the Relative Expressiveness of Higher-Order Session Processes.
- **[POPL'16]** Dominic Orchard, NY: Effects as sessions, sessions as effects .
- **[FSTTCS'15]** Romain Demangeon, NY: On the Expressiveness of Multiparty Session Types.
- **[OOPSLA'15]** Hugo A. López, Eduardo R. B. Marques, Francisco Martins, Nicholas Ng, César Santos, Vasco Thudichum Vasconcelos, NY: Protocol-Based Verification of Message-Passing Parallel Programs .
- **[CONCUR'15]** Dimitrios Kouzapas, Jorge A. Pérez, NY: Characteristic Bisimulations for Higher-Order Session Processes .
- **[CONCUR'15]** Laura Bocchi, Julien Lange, NY: Meeting Deadlines Together.
- **[CONCUR'15]** Marco Carbone, Fabrizio Montesi, Carsten Schürmann, NY: Multiparty Session Types as Coherence Proofs.
- **[CC'15]** Nicholas Ng, Jose G.F. Coutinho, NY: Protocols by Default: Safe MPI Code Generation based on Session Types.
- **[PPoPP'15]** Tiago Cogumbreiro, Raymond Hu, Francisco Martins, NY: Dynamic deadlock verification for general barrier synchronisation.
- **[POPL'15]** Julien Lange, Emilio Tuosto, NY: From communicating machines to graphical choreographies.

# Selected Publications 2015/2016



- **[POPL'17]** Julien Lange, Nicholas Ng, Bernardo Toninho, NY: Fencing off Go: Liveness and Safety for Channel-based Programming.
- **[FPL'16]** Xinyu Niu , Nicholas Ng , Tomofumi Yuki , Shaojun Wang , NY, Wayne Luk : EURECA Compilation: Automatic Optimisation of Cycle-Reconfigurable Circuits.
- **[ECOOP'16]** Alceste Scala, NY: Lightweight Session Programming in Scala
- **[CC'16]** Nicholas Ng, NY: Static Deadlock Detection for Concurrent Go by Global Session Graph Synthesis.
- **[FASE'16]** Raymond Hu, NY: Hybrid Session Verification through Endpoint API Generation.
- **[TACAS'16]** Julien Lange, NY: Characteristic Formulae for Session Types.
- **[ESOP'16]** Dimitrios Kouzapas, Jorge A. Pérez, NY: On the Relative Expressiveness of Higher-Order Session Processes.
- **[POPL'16]** Dominic Orchard, NY: Effects as sessions, sessions as effects.
- **[FSTTCS'15]** Romain Demangeon, NY: On the Expressiveness of Multiparty Session Types.
- **[OOPSLA'15]** Hugo A. López, Eduardo R. B. Marques, Francisco Martins, Nicholas Ng, César Santos, Vasco Thudichum Vasconcelos, NY: Protocol-Based Verification of Message-Passing Parallel Programs .
- **[CONCUR'15]** Dimitrios Kouzapas, Jorge A. Pérez, NY: Characteristic Bisimulations for Higher-Order Session Processes.
- **[CONCUR'15]** Laura Bocchi, Julien Lange, Nobuko Yoshida: Meeting Deadlines Together.
- **[CONCUR'15]** Marco Carbone, Fabrizio Montesi, Carsten Schürmann, NY: Multiparty Session Types as Coherence Proofs.
- **[CC'15]** Nicholas Ng, Jose G.F. Coutinho, NY: Protocols by Default: Safe MPI Code Generation based on Session Types.
- **[PPoPP'15]** Tiago Cogumbreiro, Raymond Hu, Francisco Martins, NY: Dynamic deadlock verification for general barrier synchronisation.
- **[POPL'15]** Julien Lange, Emilio Tuosto, NY: From communicating machines to graphical choreographies.

# OOI Collaboration

**OOI OCEAN OBSERVATORY INITIATIVE** CREATE ACCOUNT SIGN I

**Location** CURRENT LOCATION FILTER

**RECENT IMAGES**

- Glider**  
Last Modified: 2011-06-15  
Last Viewed: 2011-12-15  
Last Updated: 2011-12-30, 13.24
- Gorgonian Coral**  
Last Modified: 2011-06-15  
Last Viewed: 2011-12-15  
Last Updated: 2011-12-30, 13.24
- Acoustic Release**  
Last Modified: 2011-06-15  
Last Viewed: 2011-12-15  
Last Updated: 2011-12-30, 13.24

**POPULAR RESOURCES**

- SeaBird CDT**  
Last Modified: 2011-06-15  
Last Viewed: 2011-12-15  
Last Updated: 2011-12-30, 13.24
- Marine caption**  
Last Modified: 2011-06-15  
Last Viewed: 2011-12-15  
Last Updated: 2011-12-30, 13.24
- Surface Buoy**  
Last Modified: 2011-06-15  
Last Viewed: 2011-12-15  
Last Updated: 2011-12-30, 13.24

**UNUSUAL EVENTS**

- Oregon Coast Wave Height**  
Last Modified: 2011-06-15  
Last Viewed: 2011-12-15  
Last Updated: 2011-12-30, 13.24
- Water Surface Elevation**  
Last Modified: 2011-06-15  
Last Viewed: 2011-12-15  
Last Updated: 2011-12-30, 13.24

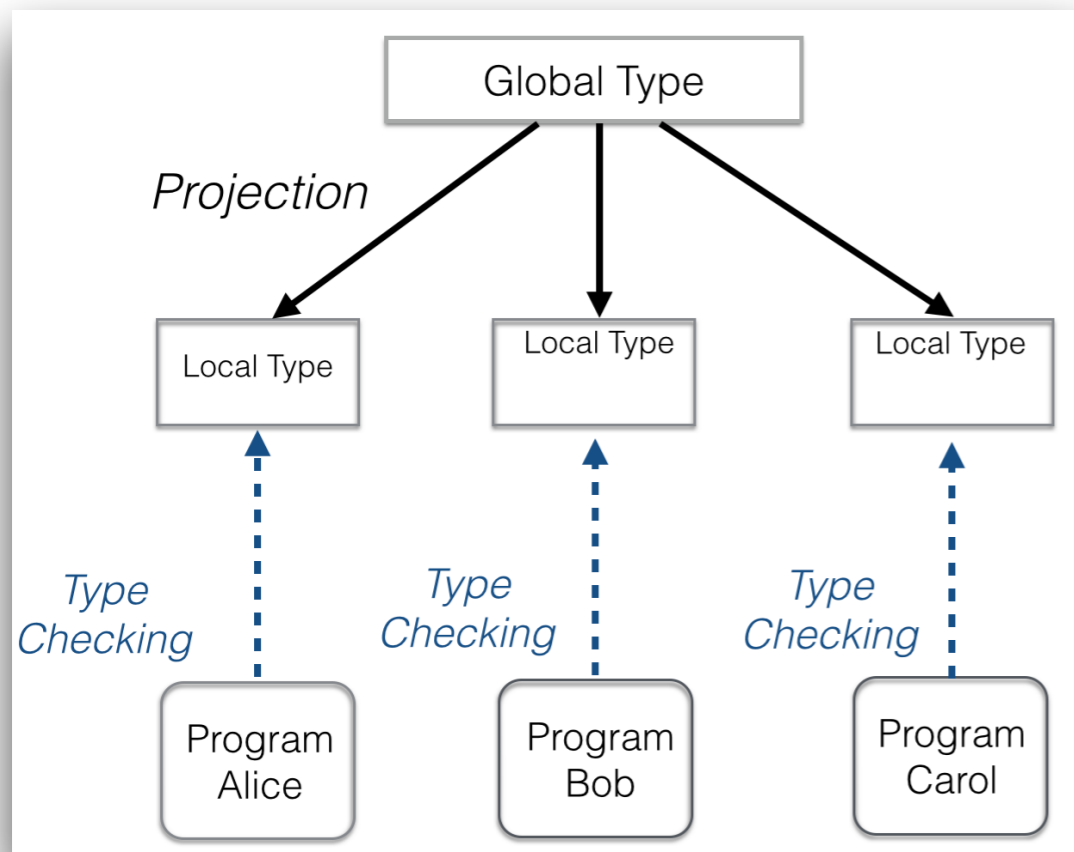
**RECENT UPDATES**

NAME	DATE	TYPE	EVENT	DESCRIPTION	NOTE
01 m Oregon Coast North Salinity	2012-01-10 23:55:55	Type	Event	Description goes here	Note goes here
01 m California South 100m pH	2012-01-10 23:55:55	Type	Event	Description goes here	Note goes here
01 m California South salinity	2012-01-10 23:55:55	Type	Event	Description goes here	Note goes here
03 m Oregon North Turbidity	2012-01-10 23:55:55	Type	Event	Description goes here	Note goes here
05 m Oregon South Temperature	2012-01-10 23:55:55	Type	Event	Description goes here	Note goes here
20 m Oregon Coast Currents	2012-01-10 23:55:55	Type	Event	Description goes here	Note goes here
01 h California South Seismology	2012-01-10 23:55:55	Type	Event	Description goes here	Note goes here
01 h Oregon Coast South 1000m O <sub>2</sub>	2012-01-10 23:55:55	Type	Event	Description goes here	Note goes here
02 h California Coast Seismology	2012-01-10 23:55:55	Type	Event	Description goes here	Note goes here
04 h California North Seismology	2012-01-10 23:55:55	Type	Event	Description goes here	Note goes here

FACEPAGE RELATED COMPOSITE STATUS

- **TCS'16:** Monitoring Networks through Multiparty Session Types. Laura Bocchi , Tzu-Chun Chen , Romain Demangeon , Kohei Honda , Nobuko Yoshida
- **LMCS'16:** Multiparty Session Actors. Romyana Neykova, Nobuko Yoshida
- **FMSD'15:** Practical interruptible conversations: Distributed dynamic verification with multiparty session types and Python. Romain Demangeon , Kohei Honda , Raymond Hu , Romyana Neykova , Nobuko Yoshida
- **TGC'13:** The Scribble Protocol Language. Nobuko Yoshida , Raymond Hu , Romyana Neykova , Nicholas Ng

# Session Types Overview



- Global session type

$$G = A \rightarrow B : \langle U_1 \rangle . B \rightarrow C : \langle U_2 \rangle . C \rightarrow A : \langle U_3 \rangle$$

- Local session type

- Slice of global protocol relevant to one role
- Mechanically derived from a global protocol

$$T_A = !\langle B, U_1 \rangle . ?\langle C, U_3 \rangle$$

- Process language

- Execution model of I/O actions by session participants
- Mechanically derived from a global protocol

$$P_A = a[A](x) . x ! \langle B, u_1 \rangle . x ? (C, y)$$

- (Static) type checking for communication safety and progress

# The Scribble Protocol Language

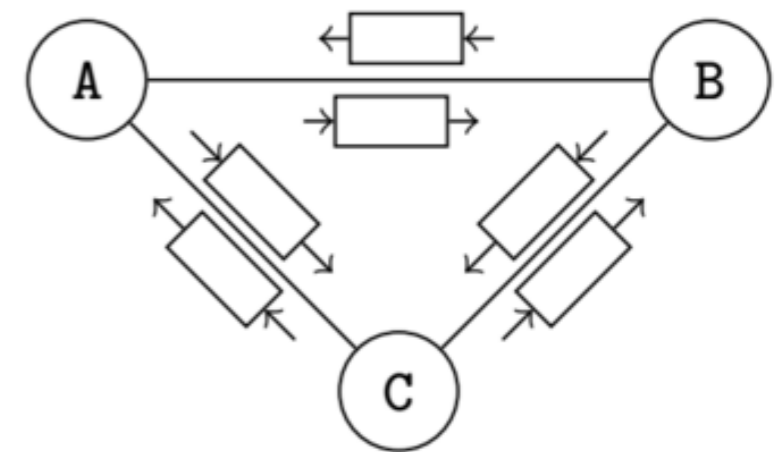


Scribble: adapts and extends MPST as an engineering language for describing multiparty message passing protocols

```
global protocol MyProtocol(role A, role B, role C) {  
  m1(int) from A to B;  
  rec X {  
    choice at B {  
      m2(String) from B to C;  
      continue X;  
    } or {  
      m3() from B to C;  
    }  
  } } }
```

## Communication Model

- ▶ asynchronous, reliable, role-to-role ordering
- ▶ Scribble sessions can be conducted over any transport that supports this model



## Scribble: Describing Multi Party Protocols

Scribble is a language to describe application-level protocols among communicating systems. A protocol represents an agreement on how participating systems interact with each other. Without a protocol, it is hard to do meaningful interaction: participants simply cannot communicate effectively, since they do not know when to expect the other parties to send data, or whether the other party is ready to receive data. However, having a description of a protocol has further benefits. It enables verification to ensure that the protocol can be implemented without resulting in unintended consequences, such as deadlocks.

### Describe

Scribble is a language for describing multiparty protocols from a global, or endpoint neutral, perspective.

### Verify

Scribble has a theoretical foundation, based on the Pi Calculus and Session Types, to ensure that protocols described using the language are sound, and do not suffer from deadlocks or livelocks.

### Project

Endpoint projection is the term used for identifying the responsibility of a particular role (or endpoint) within a protocol.

### Implement

Various options exist, including (a) using the endpoint projection for a role to generate a skeleton code, (b) using session type APIs to clearly describe the behaviour, and (c) statically verify the code against the projection.

### Monitor

Use the endpoint projection for roles defined within a Scribble protocol, to monitor the activity of a particular endpoint, to ensure it correctly implements the expected behaviour.

# Online tool : <http://scribble.doc.ic.ac.uk/>

---

```
1 module examples;
2
3 global protocol HelloWorld(role Me, role World) {
4     hello() from Me to World;
5     choice at World {
6         goodMorning1() from World to Me;
7     } or {
8         goodMorning1() from World to Me;
9     }
10 }
```

Load a sample 

Check

Protocol:

Role:

Project

Generate Graph

# Open Problems

---

1. Behavioural Theories and Session Types
2. Relationship with Other Frameworks
  - ▶ Linear Logic
  - ▶ Communicating Automata
  - ▶ Petri Nets
3. Outreach
  - ▶ Industry
  - ▶ Developers
  - ▶ Education

# Interactions with Industries

## Strange Loop

SEPTEMBER 15-17 2016 / PEABODY OPERA HOUSE / ST. LOUIS, MO



Nobuko Yoshida  
Imperial College, London

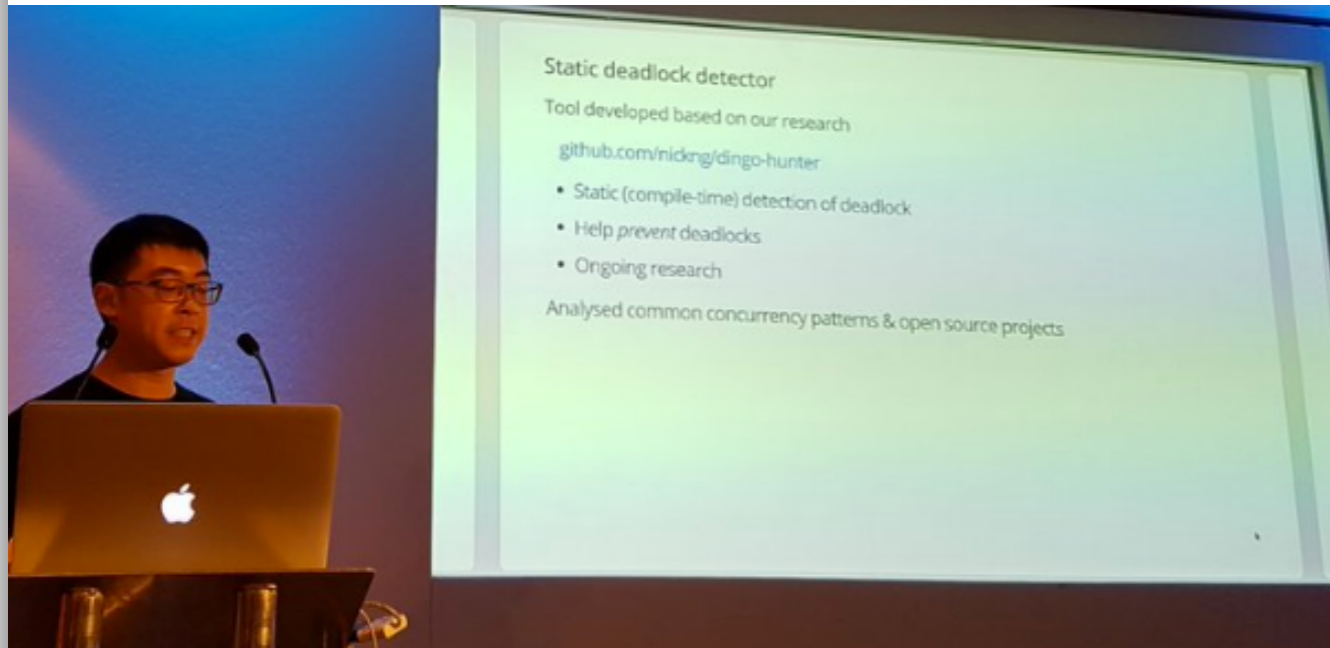


Adam Bowen @adamnbowen · Sep 15

I didn't even know that session types existed an hour ago, but thanks to Nobuko Yoshida's great talk at [#pwlconf](#), I want to learn more.

## DoC researcher to speak at Golang UK conference

by Vicky Kapogianni  
20 July 2016



DoC researcher to speak at industry-focused Golang UK conference on results of concurrency research

[Click here to add content](#)



.@nicholascwng rocking on @GolangUKconf about static deadlock detection in #golang #gouk16



# Interactions with Industries

## F#unctional Londoners Meetup Group

6 days ago · 6:30 PM

### Session Types with Fahd Abdeljallal



43 Members

Synopsis: Session types are a formalism to codify the structure of a communication, using types to specify the communication protocol used. This formalism provides the... [LEARN MORE](#)

## Distributed Systems vs. Compositionality

Dr. Roland Kuhn  
@rolandkuhn — CTO of Actyx

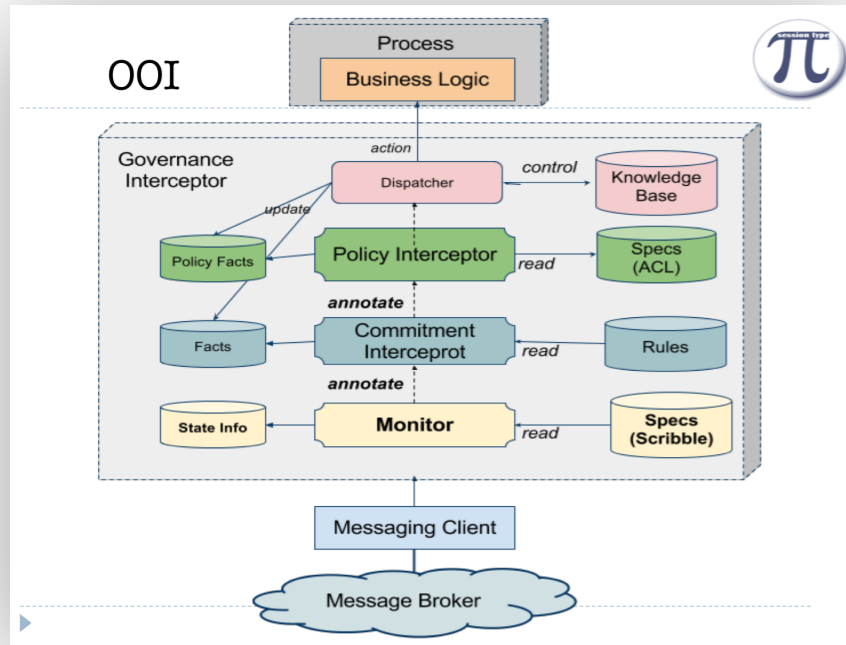
actyx

### Current State

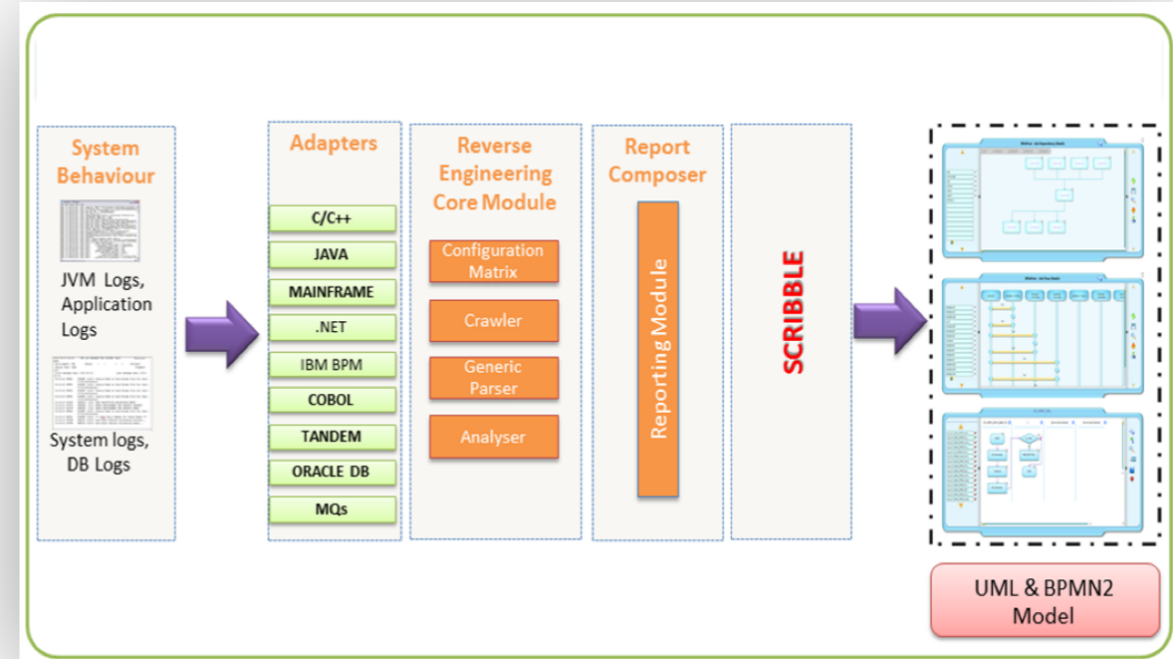
- behaviors can be composed both sequentially and concurrently
- effects are not yet tracked
- Scribble generator for Scala not yet there
- theoretical work at Imperial College, London (Prof. Nobuko Yoshida & Alceste Scalas)

# Applications

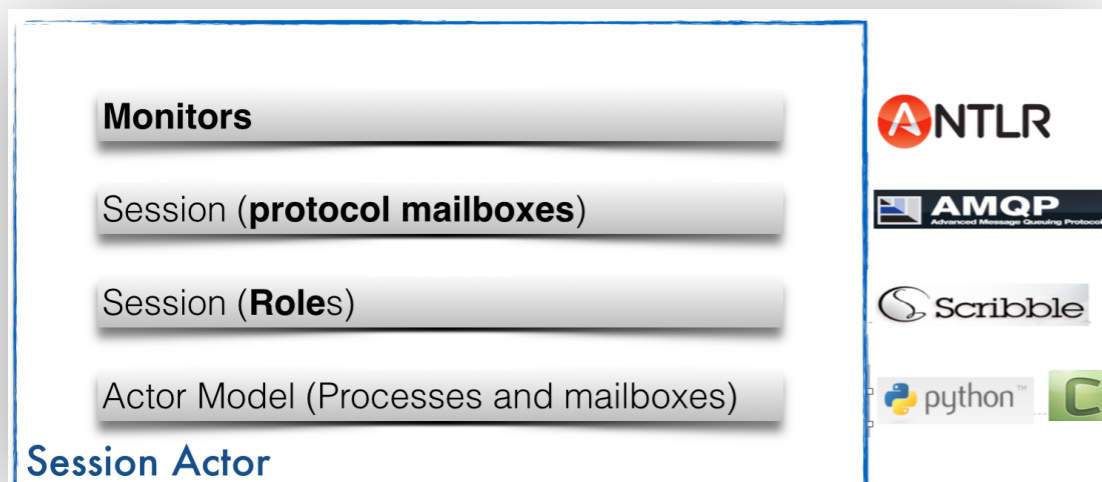
## OOI Governance



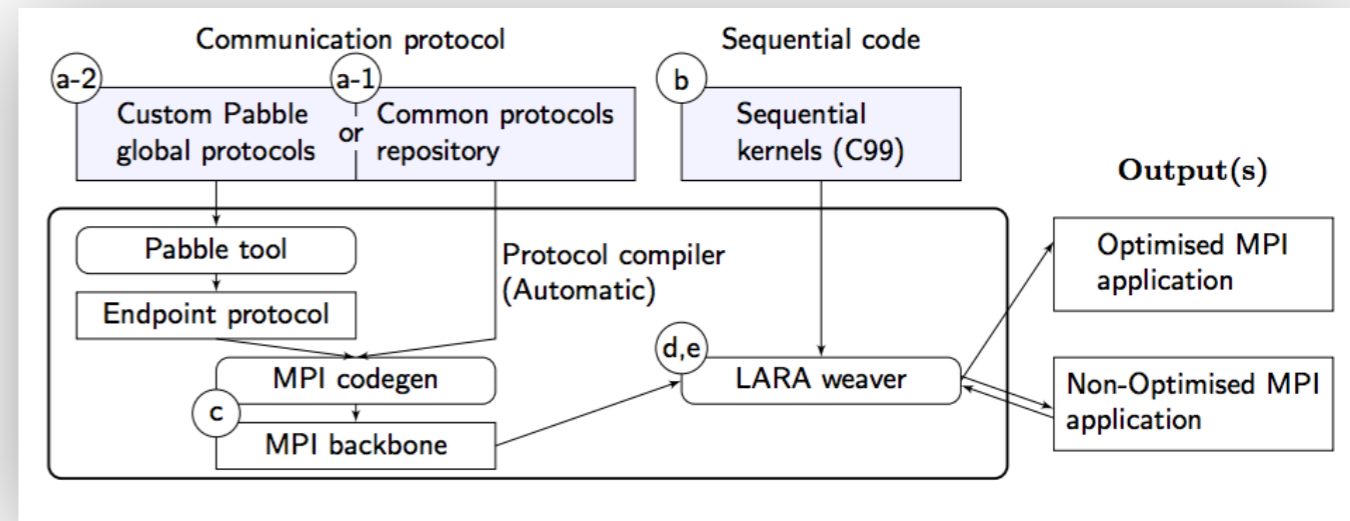
## ZDLC: Process Modeling



## Protocol Verification

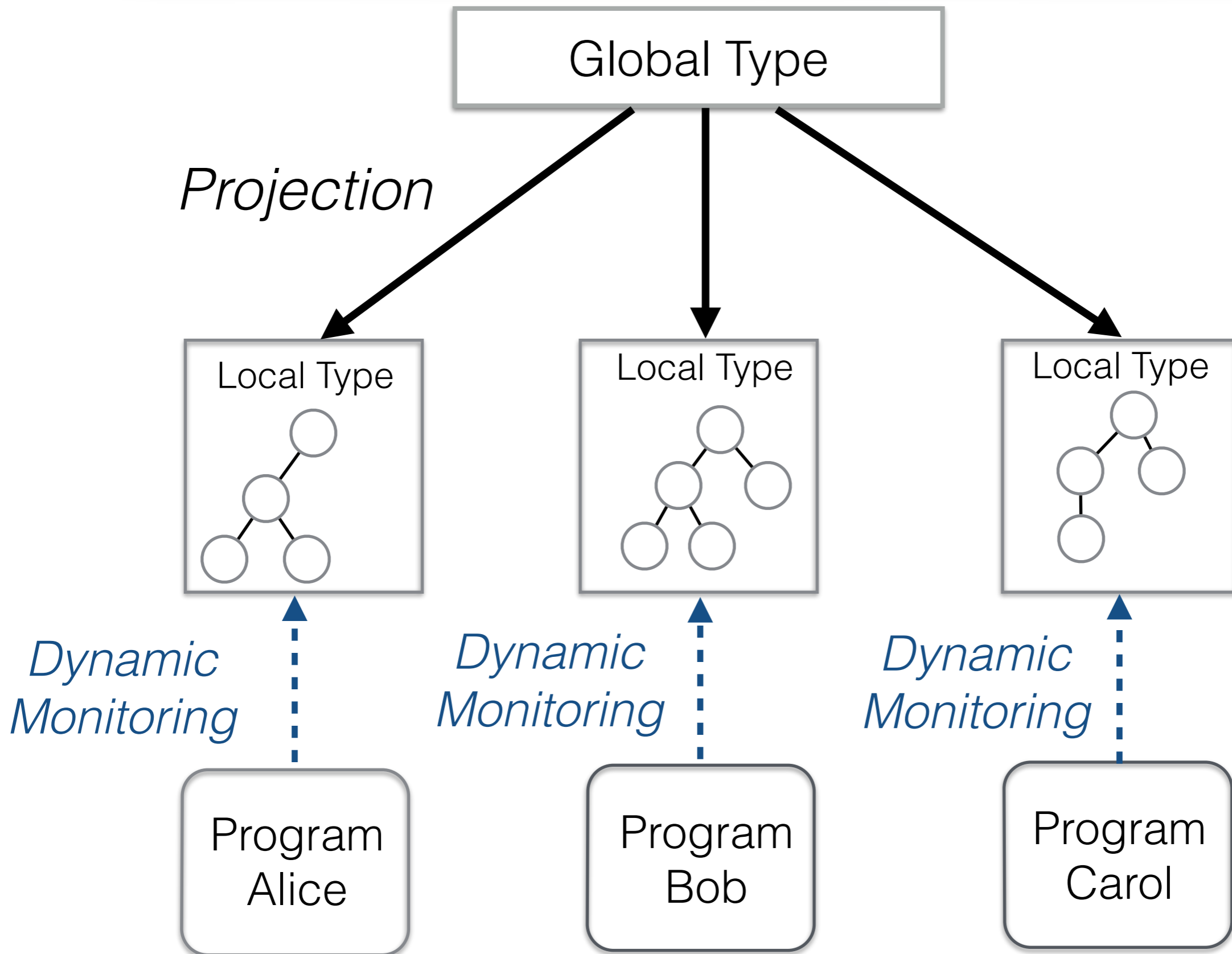


## MPI code generations

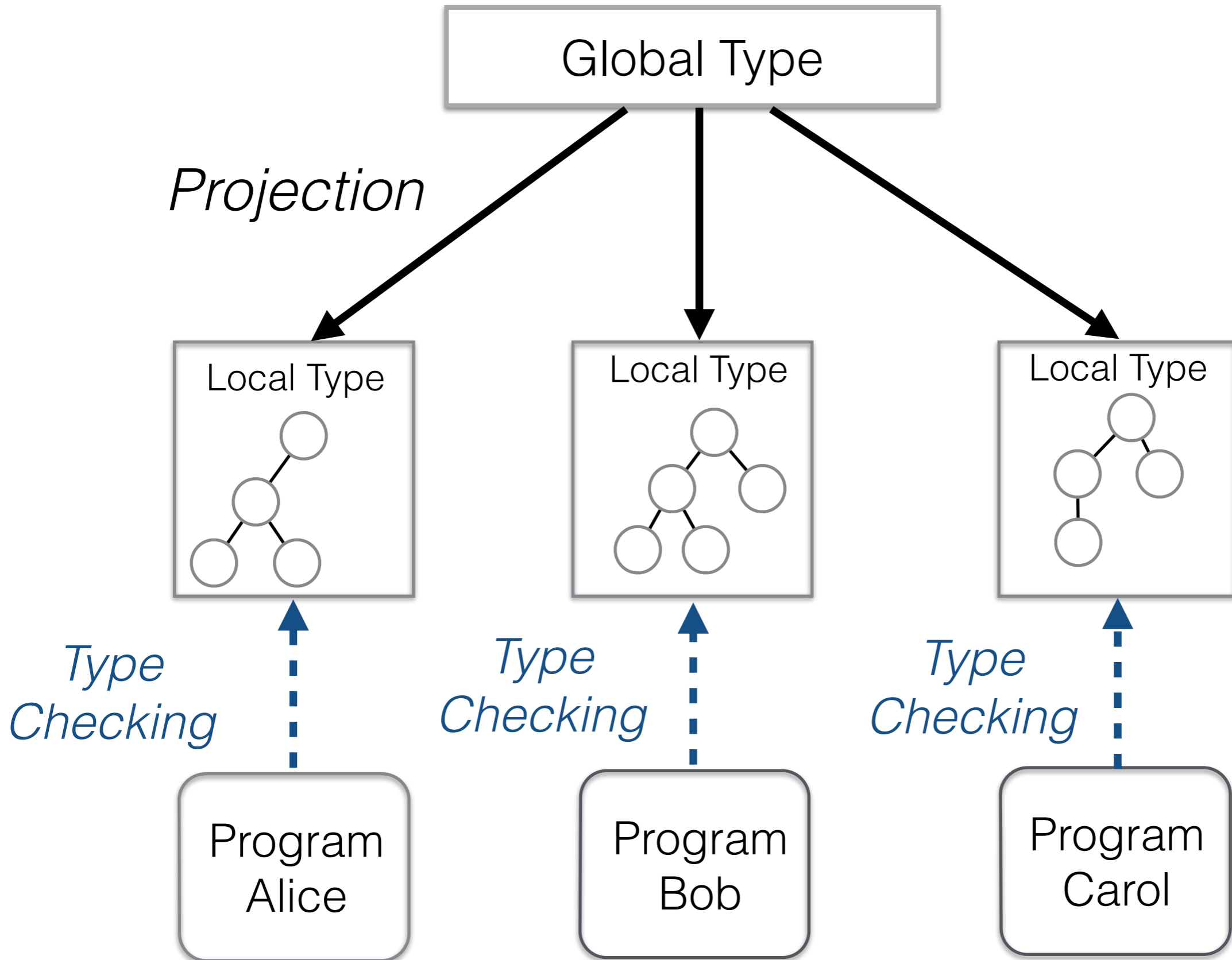


# Dynamic Monitoring

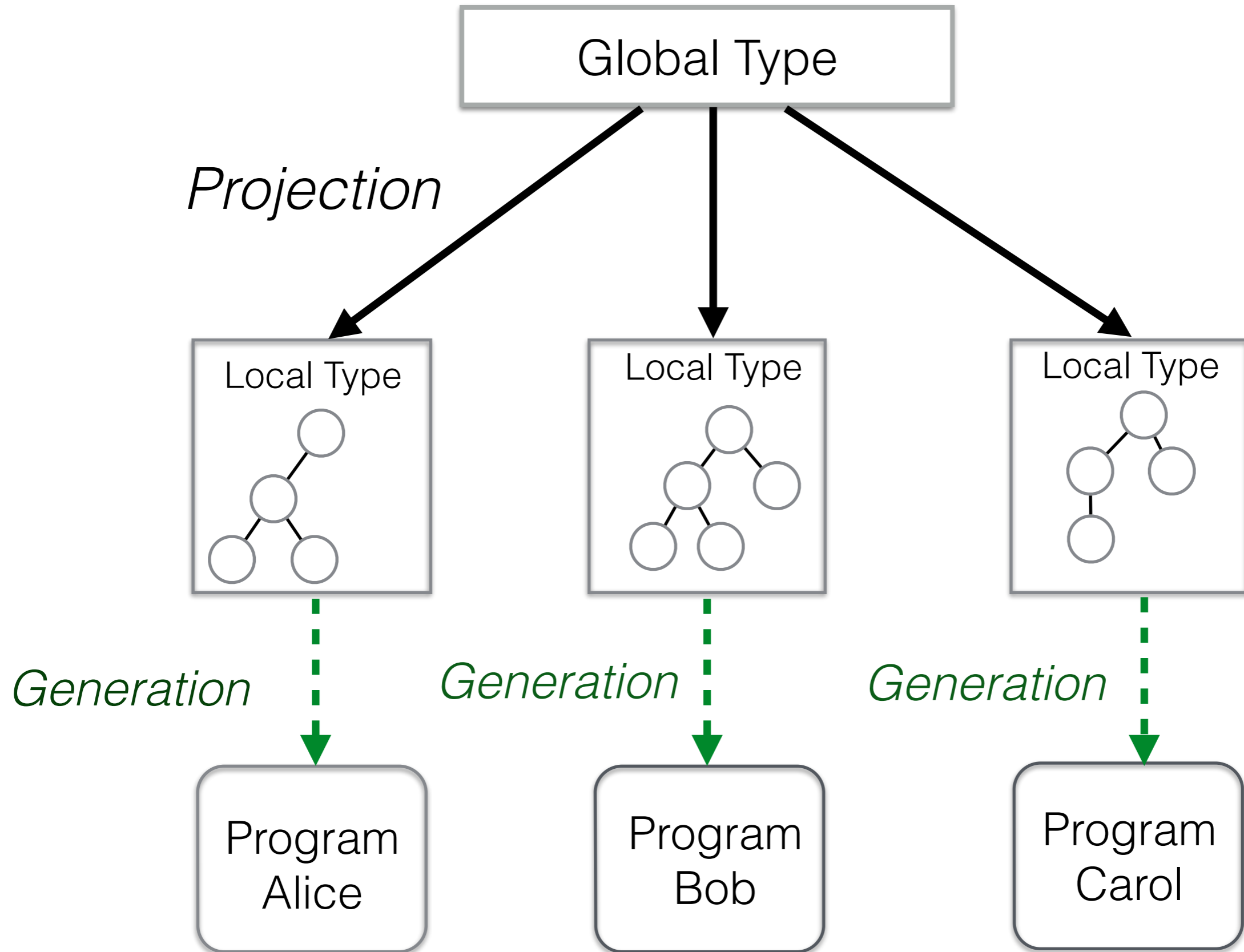
[RV'13, COORDINATION'14, FMSSD'15]



# Type Checking [OOPSLA'15, POPL'16]

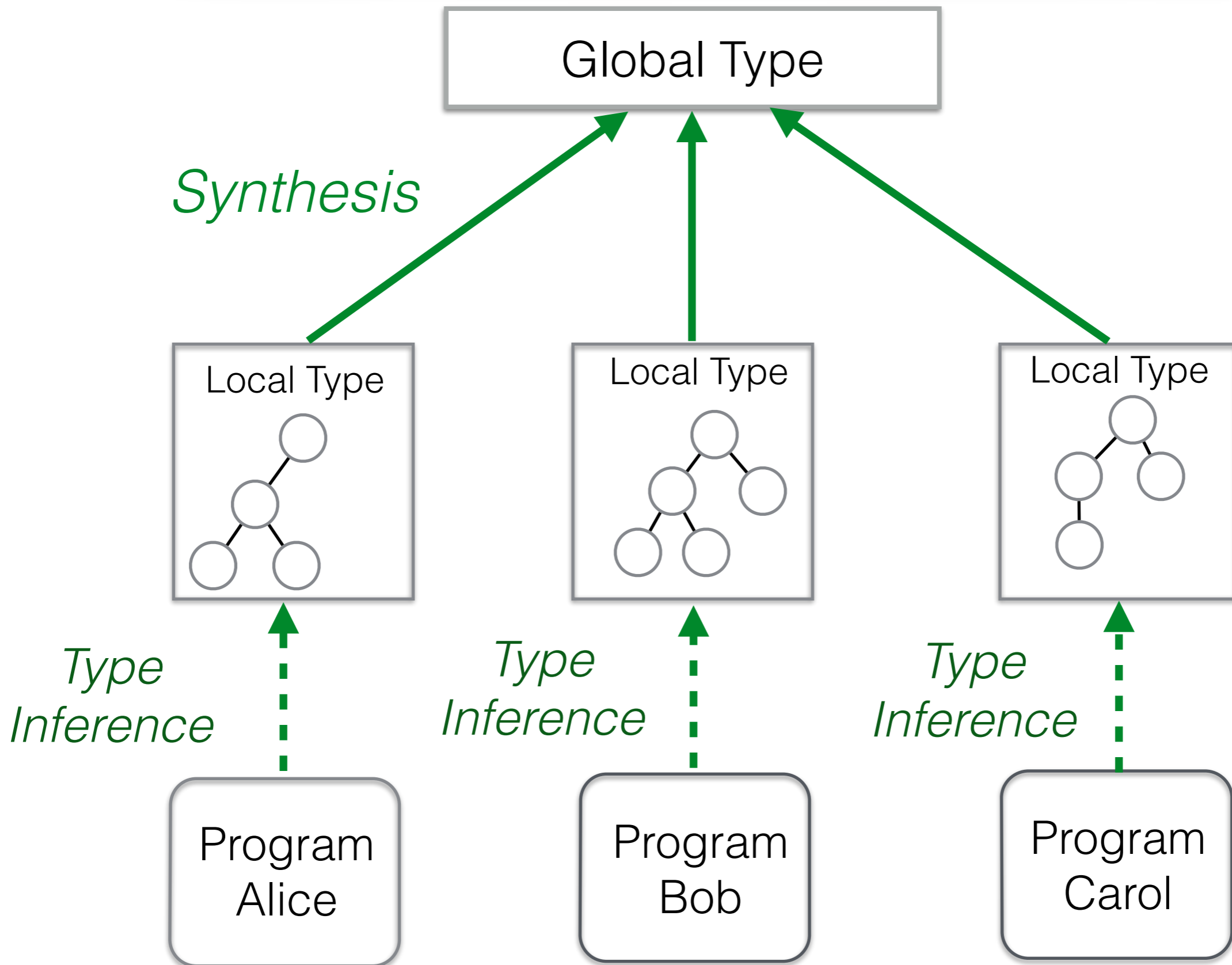


# Code Generation [CC'15]



# Synthesis

[ICALP'13, POPL'15, CONCUR'15]



# GLOBALLY GOVERNED SESSION SEMANTICS



Dimitrios  
Kouzapas  
Glasgow



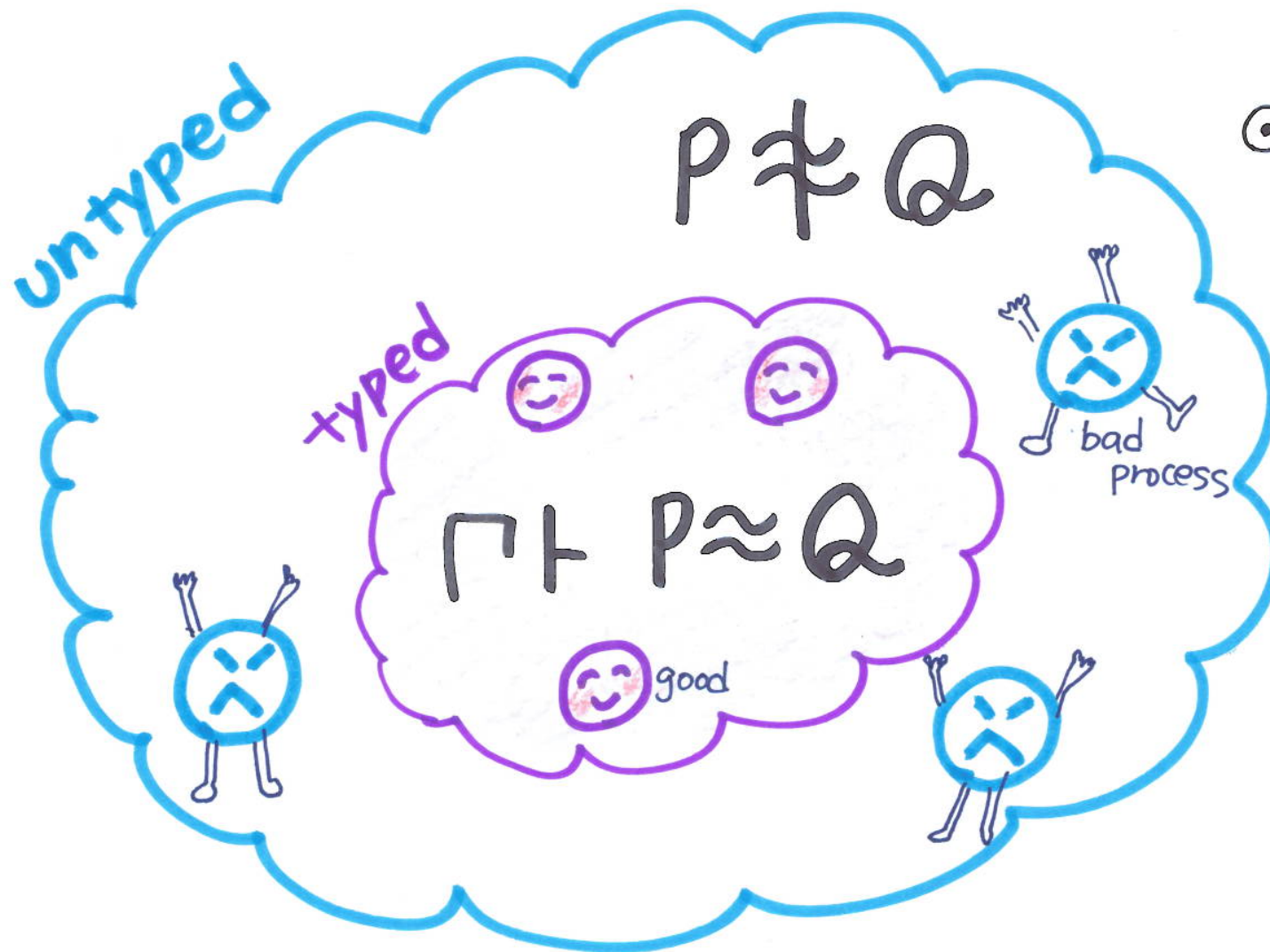
Nobuko  
Yoshida

Imperial  
College London




# Typed Semantics in $\pi$ 1991 $\rightarrow$

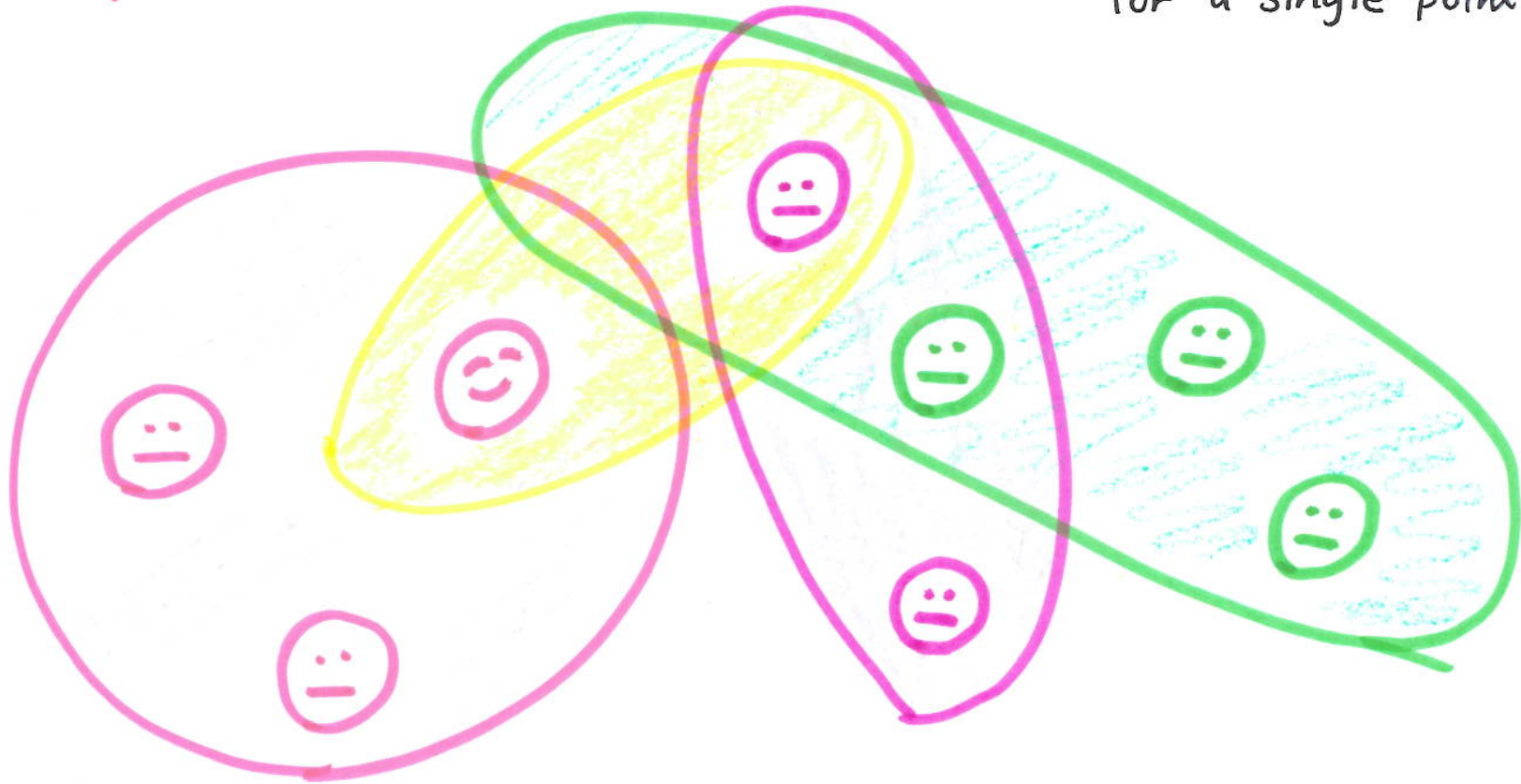
IO-subtyping, Linear types, Secure Information Flow, ...



- ⊙ Correctness of Encoding
- ⊙ Limit environments  $\vdash$   
 $\Rightarrow$  Equate more processes
- ⊙ Compositional

# Multiparty Session Types

- Participants agreed with global protocols 
- **Many** Multiparty Sessions can **interleave**  
for a single point application



with each message clearly identifiable as belonging to a specific session

# Multiparty Session Bisimulations

Standard Multiparty Session Bisimulations  $\approx_s$

$\Gamma \vdash P \triangleright \Delta$

Shared  
Env

Session  
channels  
Env

Governed Multiparty Session Bisimulations  $\approx_g$

$E, \Gamma \vdash P \triangleright \Delta$

Global Type  
Env

a mapping from session to global types

$S_1: G_1, S_2: G_2, \dots, S_m: G_m$

# Governed BSimulations

? Compositional? Coincides with Contextual Equiv?

? What is a difference between  $\approx_s$  and  $\approx_g$ ?

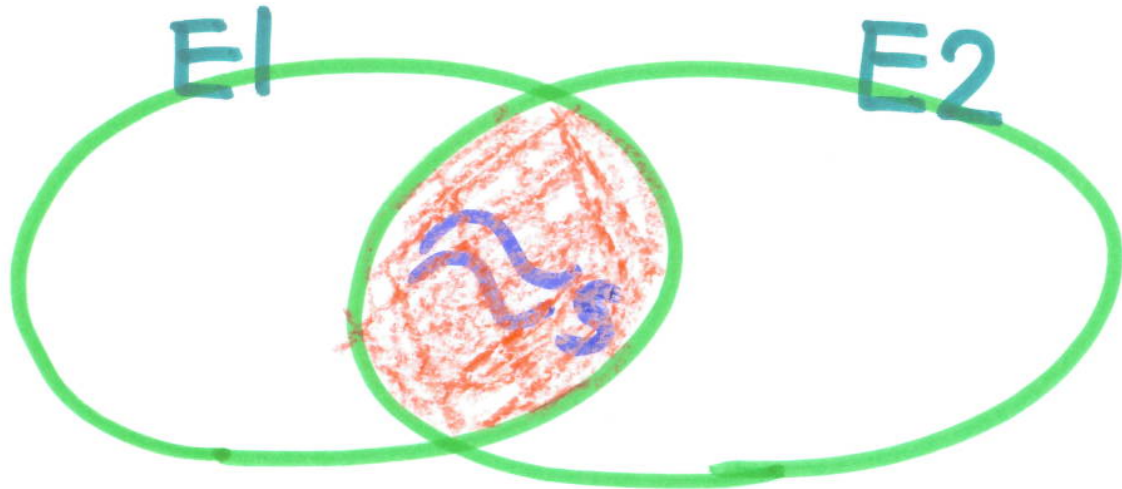
? Under what condition  $\approx_s$  and  $\approx_g$  can coincide?

? Applications?



# Theorem 3

- If for all  $E$   $E, \Gamma \vdash P_1 \triangleright \Delta_1 \approx_g P_2 \triangleright \Delta_2$   
then  $\Gamma \vdash P_1 \triangleright \Delta_1 \approx_s P_2 \triangleright \Delta_2$
- If  $\Gamma \vdash P_1 \triangleright \Delta_1 \approx_s P_2 \triangleright \Delta_2$   
then for all  $E$   $E, \Gamma \vdash P_1 \triangleright \Delta_1 \approx_g P_2 \triangleright \Delta_2$

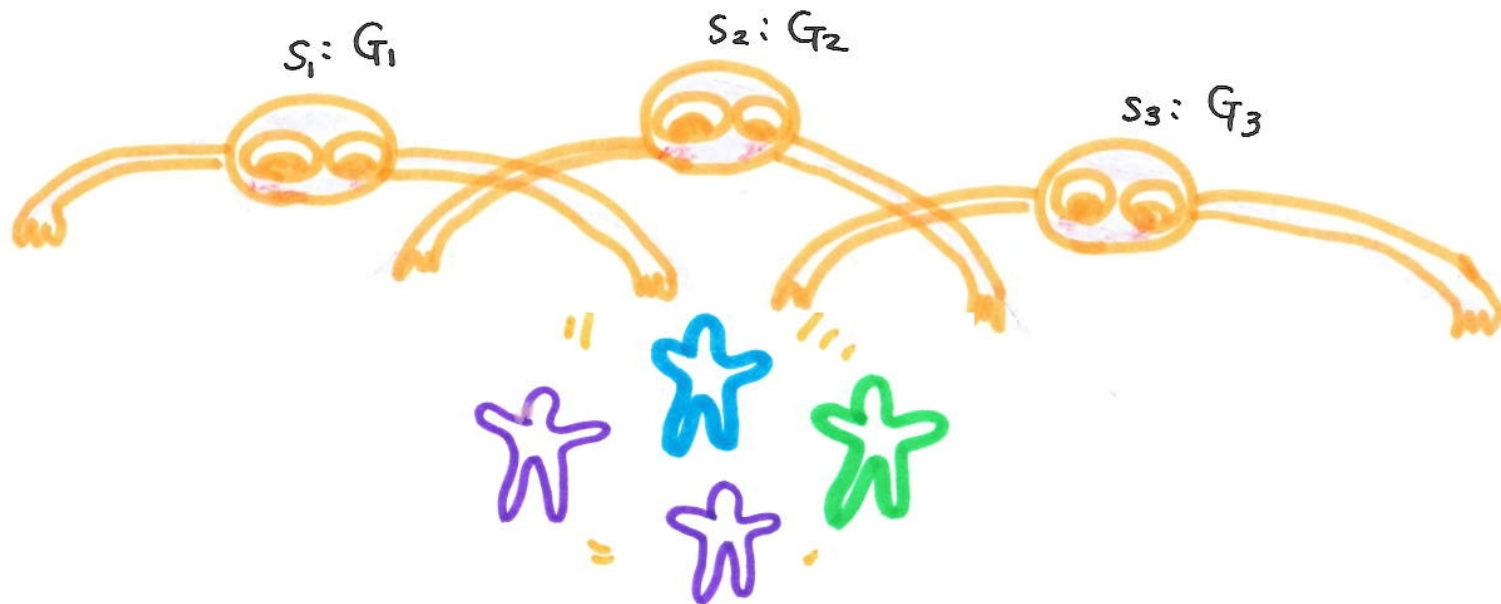


# Theorem 4 (coincidence) no interleaved sessions

Assume  $P_1$  and  $P_2$  are simple. If there exists

$E$  s.t.  $E, \Gamma \vdash P_1 \triangleright \Delta_1 \approx_g P_2 \triangleright \Delta_2$ , then

$\Gamma \vdash P_1 \triangleright \Delta_1 \approx_s P_2 \triangleright \Delta_2$



# Theorem 4 (coincidence) no interleaved sessions

Assume  $P_1$  and  $P_2$  are simple. If there exists

$E$  s.t.  $E, \Gamma \vdash P_1 \triangleright \Delta_1 \approx_g P_2 \triangleright \Delta_2$ , then

$\Gamma \vdash P_1 \triangleright \Delta_1 \approx_s P_2 \triangleright \Delta_2$

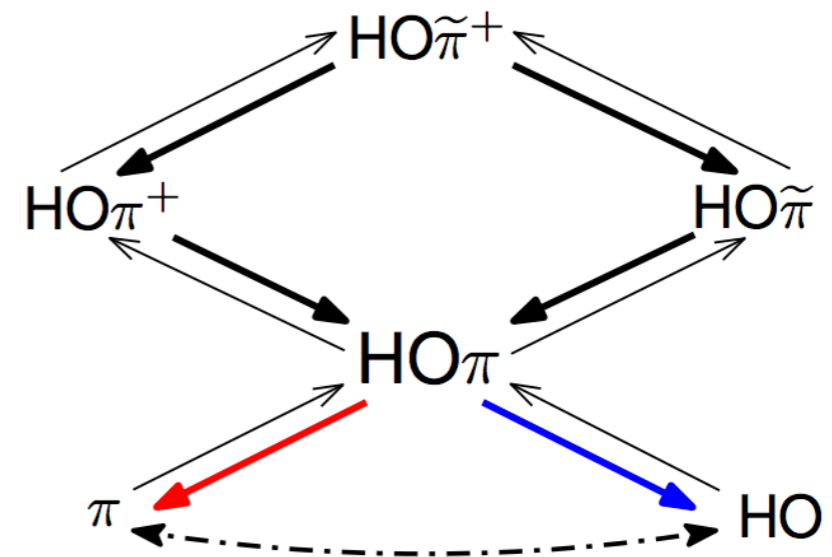


# Behaviour Theory of Higher-Order Pi Calculus

## Bridging functions and concurrency



- First expressivity results for HO process calculi with **session types**
- Different calculi with functional and concurrent features, tightly connected
- Session types guide encodings, and induce **strong forms of correctness**
- Several other results in the paper



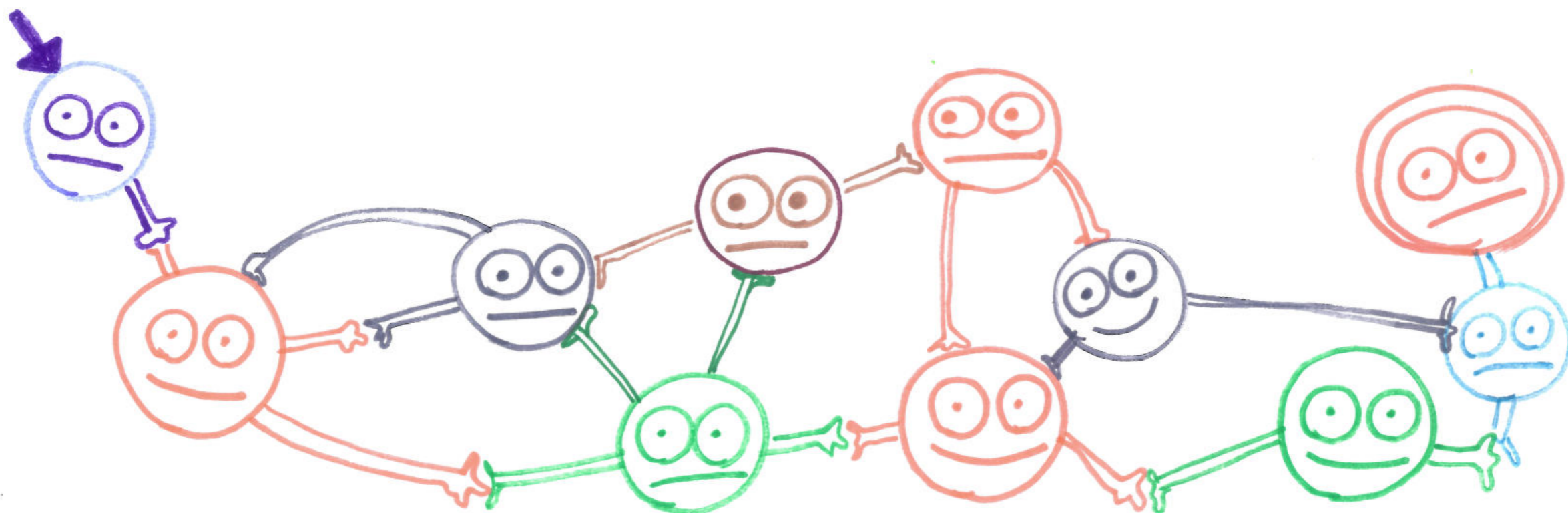
- **ESOP'16:** On the Relative Expressiveness of Higher-Order Session Processes. Dimitrios Kouzapas , Jorge A. Pérez , Nobuko Yoshida
- **CONCUR'15:** Characteristic Bisimulations for Higher-Order Session Processes. Dimitrios Kouzapas , Jorge A. Pérez , Nobuko Yoshida

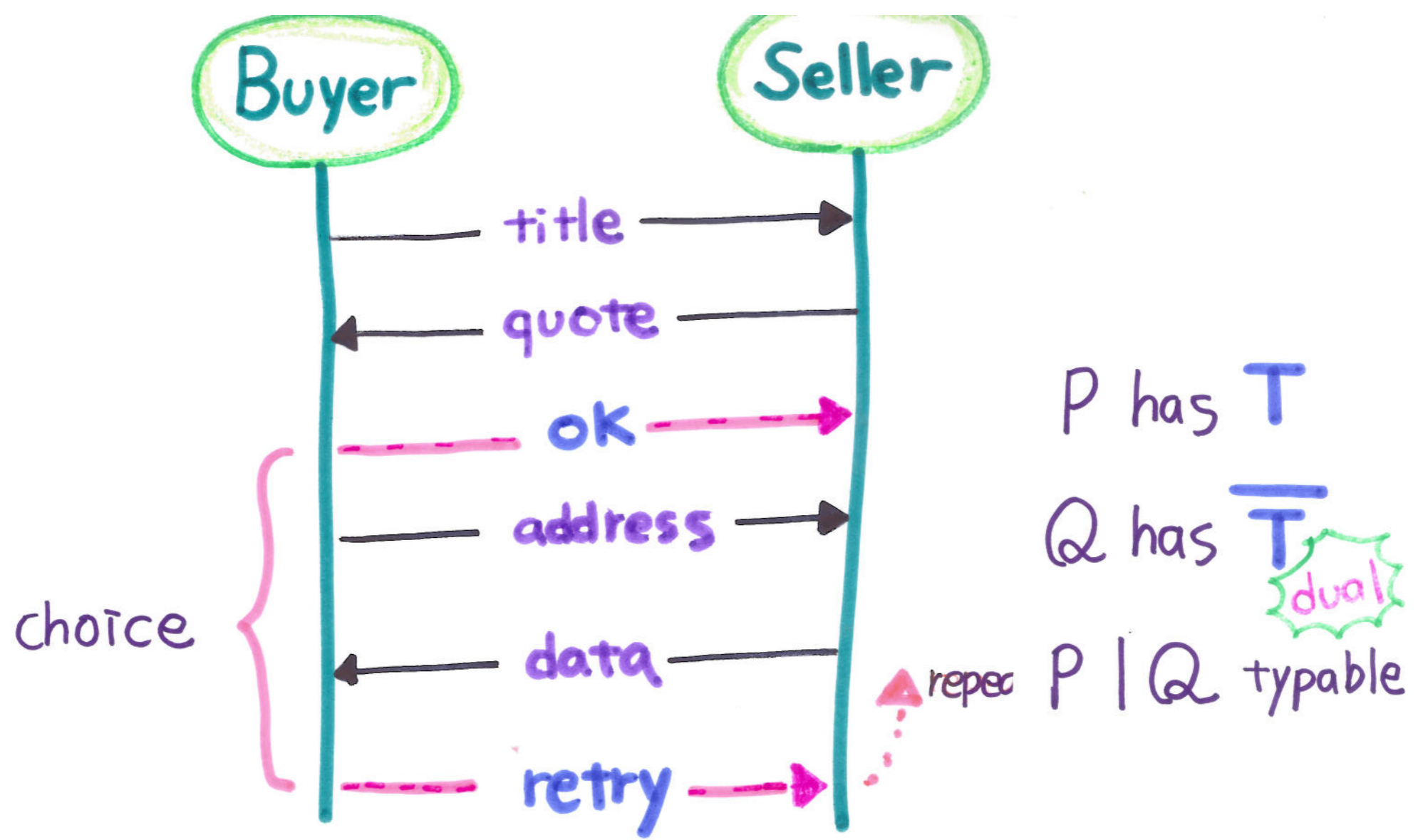
# Multiparty Compatibility in Communicating Automata

## Synthesis and Characterisation of Multiparty Session Types

Nobuko Yoshida

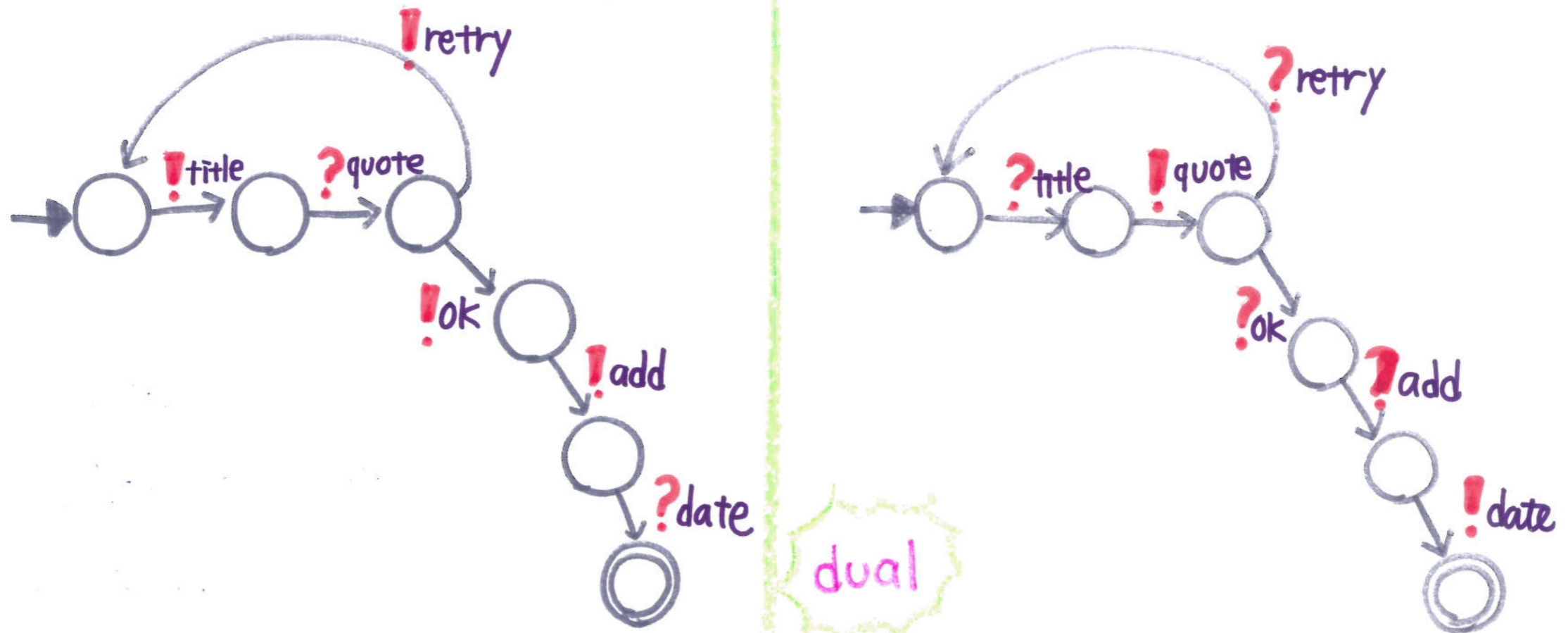
Pierre-Malo Denielou **ICALP'13**

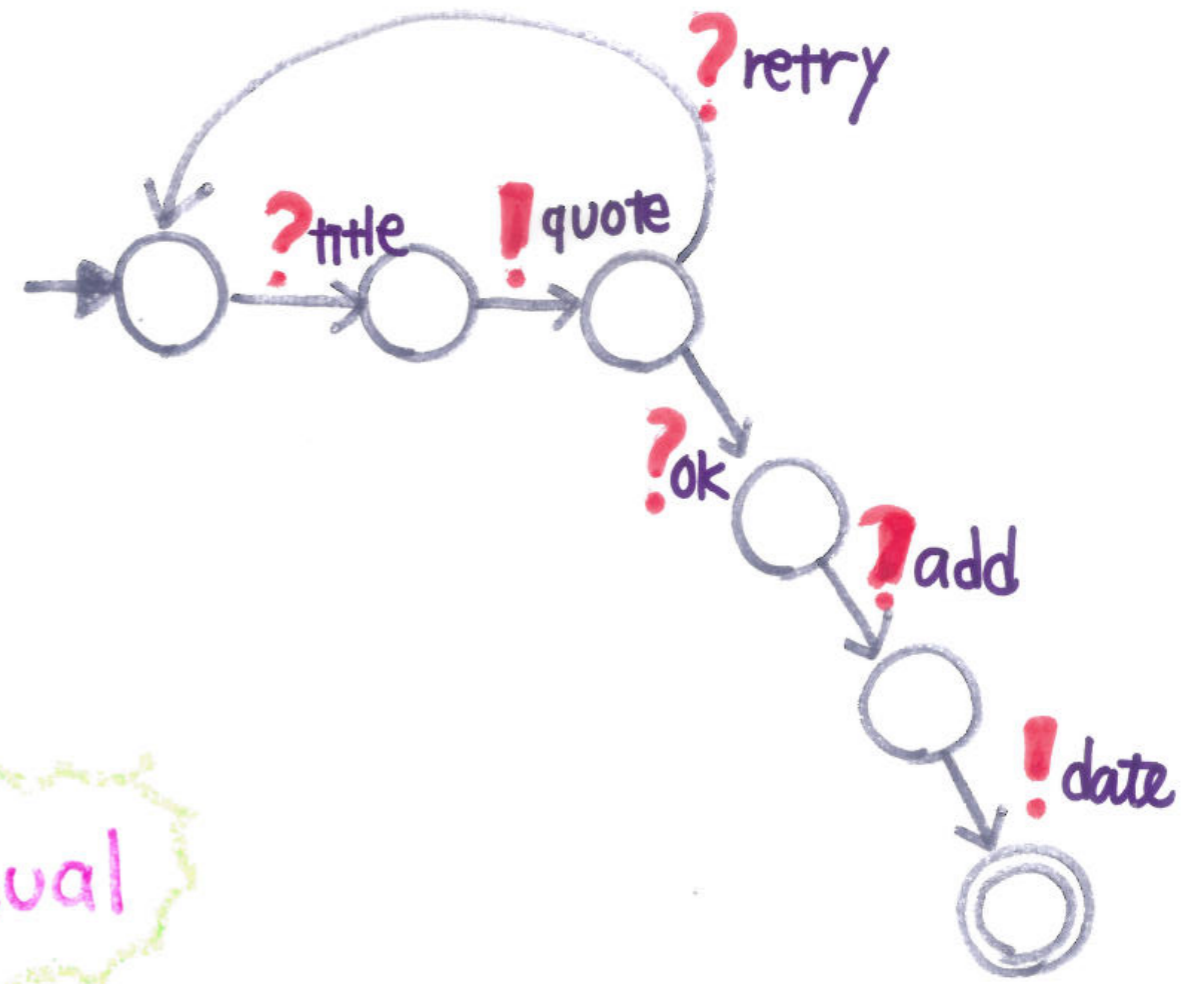
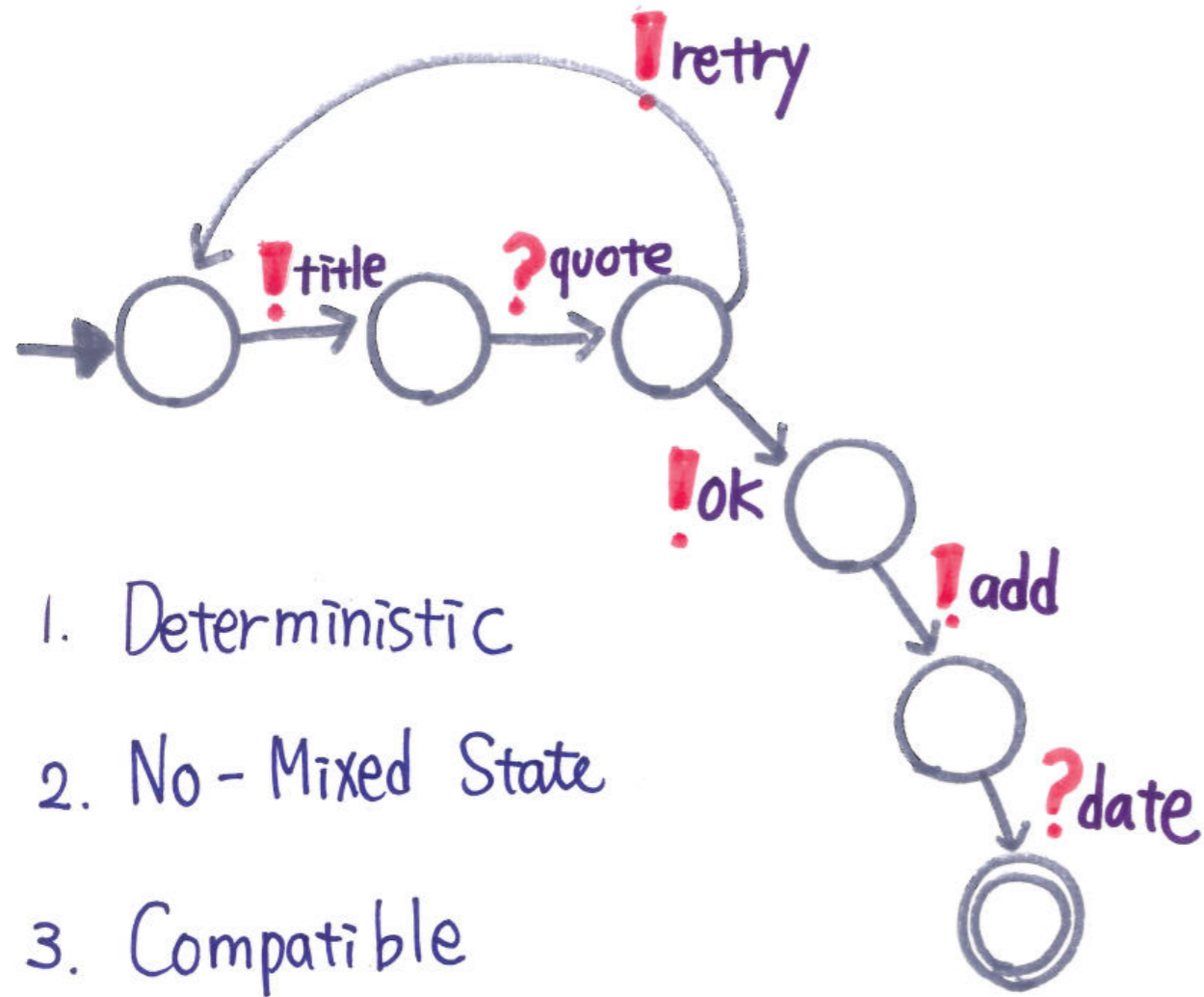




ut! Title ; ? Quote ; ! { ok: ! Add ; ? Date , retry: t }  
 ut? Title ; ! Quote ; ? { ok: ? Add ; ! Date , retry: t }

# Communicating Automata [1980s]

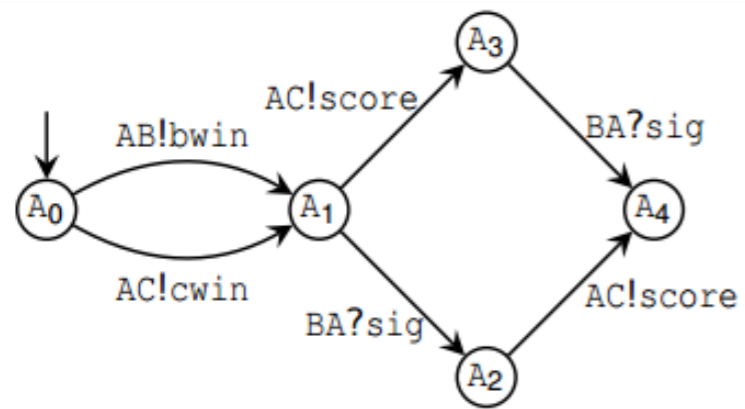




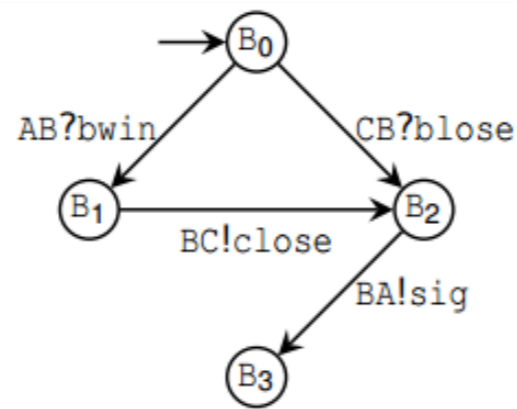
dual

[Gouda et al 1986] Two compatible machines without mixed states which are deterministic satisfy deadlock-freedom.

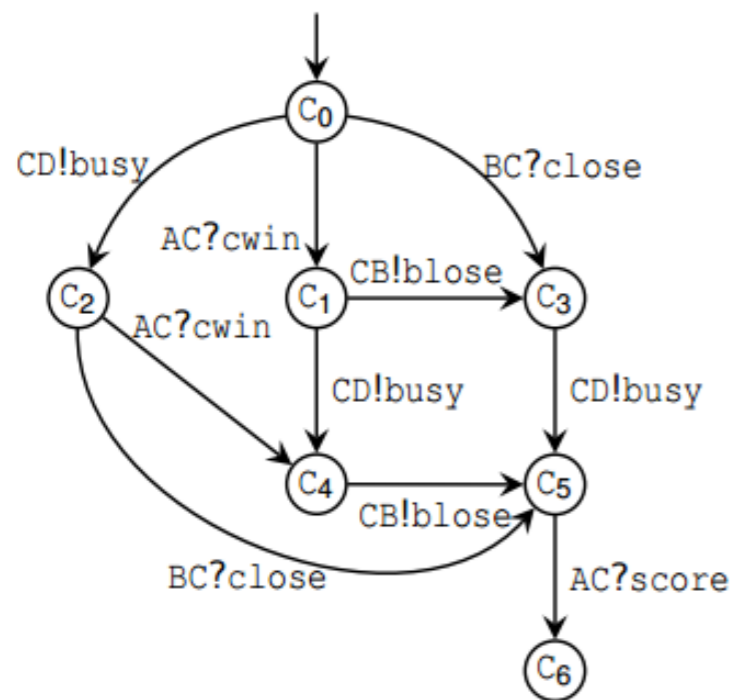
# From Communicating Machines to Graphical Choreographies [POPL'15, CONCUR'15]



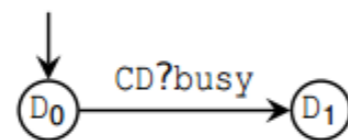
Alice



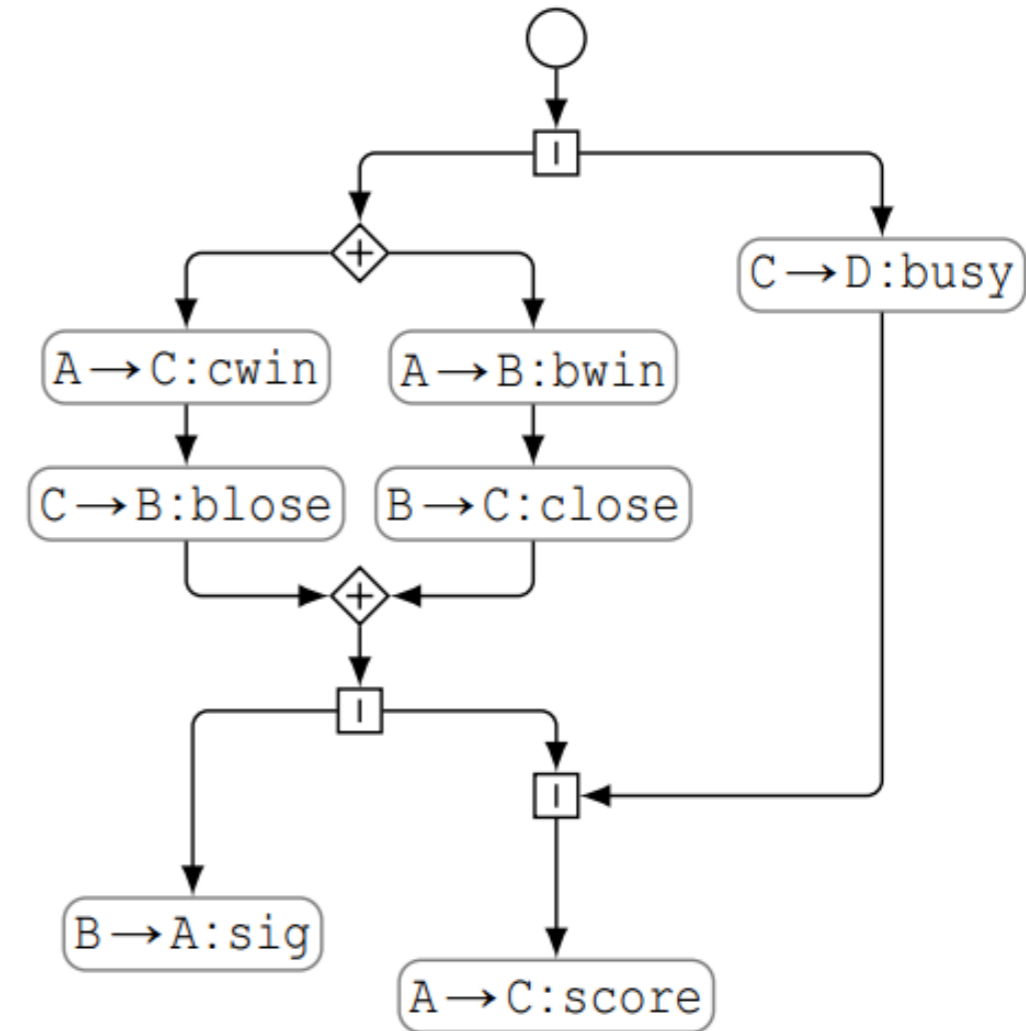
Bob



Carol



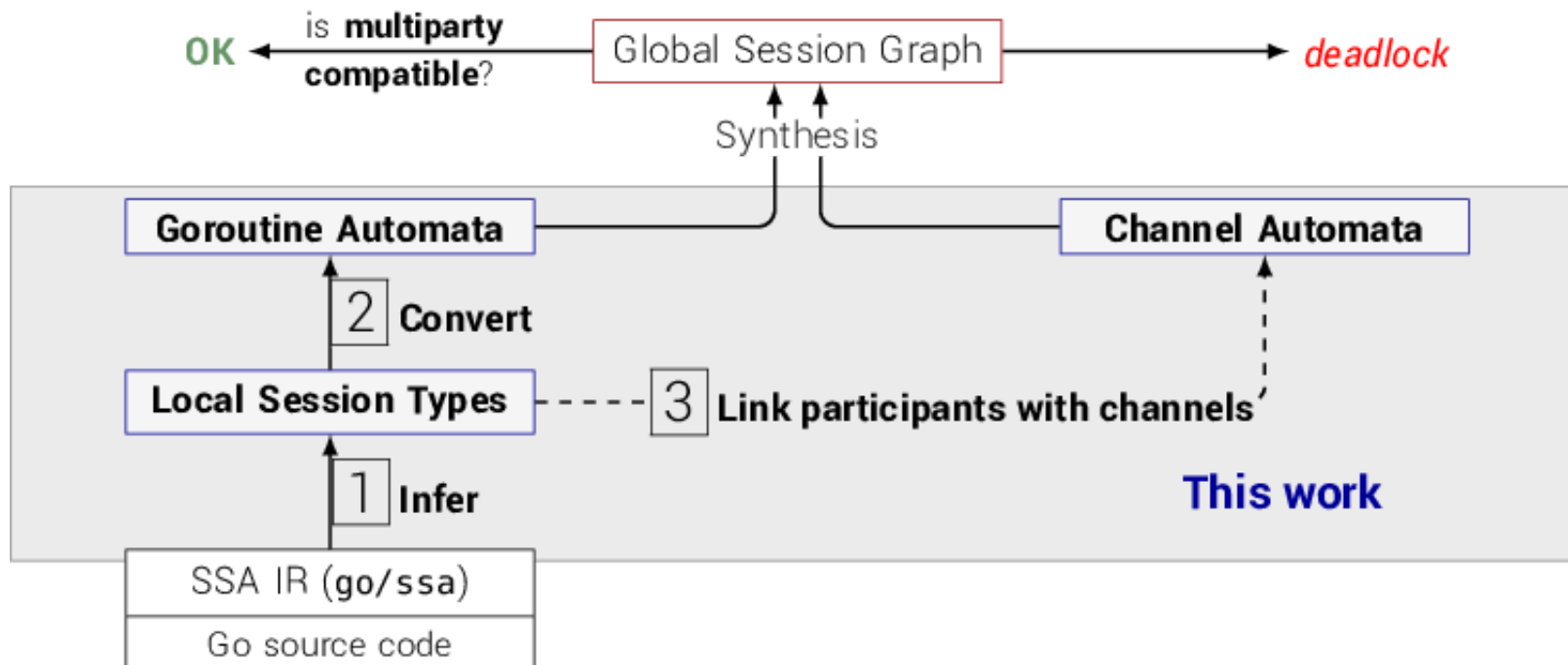
Dave



[ESOP'10, ESOP'12, CONCUR'12, CONCUR'14]

# Contributions

- Static deadlock detection tool *dingo-hunter*
- Deadlock detection based on session types
- **Infer** session types as Communicating Automata
- **Synthesise** global session graphs from CA



# Go and Concurrency

- Developed by Google for multi-core programming
- Concurrency model built on CSP (process calculi)
- Message-passing **communication** over channels

*" Do not communicate by sharing memory; instead, share memory by communicating. "*

-- Effective Go (developer guide)

# Java API Generation [FASE'16]



RFC 821

August 1982  
Simple Mail Transfer Protocol

## TABLE OF CONTENTS

<a href="#">1.</a>	<a href="#">INTRODUCTION</a>	<a href="#">1</a>
<a href="#">2.</a>	<a href="#">THE SMTP MODEL</a>	<a href="#">2</a>
<a href="#">3.</a>	<a href="#">THE SMTP PROCEDURE</a>	<a href="#">4</a>
<a href="#">3.1.</a>	<a href="#">Mail</a>	<a href="#">4</a>
<a href="#">3.2.</a>	<a href="#">Forwarding</a>	<a href="#">4</a>
<a href="#">3.3.</a>	<a href="#">Verifying and Expanding</a>	<a href="#">4</a>
<a href="#">3.4.</a>	<a href="#">Sending and Mailing</a>	<a href="#">4</a>
<a href="#">3.5.</a>	<a href="#">Opening and Closing</a>	<a href="#">4</a>
<a href="#">3.6.</a>	<a href="#">Relaying</a>	<a href="#">4</a>
<a href="#">3.7.</a>	<a href="#">Domains</a>	<a href="#">4</a>
<a href="#">3.8.</a>	<a href="#">Changing Roles</a>	<a href="#">4</a>
<a href="#">4.</a>	<a href="#">THE SMTP SPECIFICATIONS</a>	<a href="#">10</a>
<a href="#">4.1.</a>	<a href="#">SMTP Commands</a>	<a href="#">10</a>
<a href="#">4.1.1.</a>	<a href="#">Command Semantics</a>	<a href="#">10</a>
<a href="#">4.1.2.</a>	<a href="#">Command Syntax</a>	<a href="#">10</a>
<a href="#">4.2.</a>	<a href="#">SMTP Replies</a>	<a href="#">10</a>
<a href="#">4.2.1.</a>	<a href="#">Reply Codes by Function Group</a>	<a href="#">10</a>
<a href="#">4.2.2.</a>	<a href="#">Reply Codes in Numeric Order</a>	<a href="#">10</a>
<a href="#">4.3.</a>	<a href="#">Sequencing of Commands and Replies</a>	<a href="#">10</a>
<a href="#">4.4.</a>	<a href="#">State Diagrams</a>	<a href="#">10</a>
<a href="#">4.5.</a>	<a href="#">Details</a>	<a href="#">10</a>
<a href="#">4.5.1.</a>	<a href="#">Minimum Implementation</a>	<a href="#">10</a>
<a href="#">4.5.2.</a>	<a href="#">Transparency</a>	<a href="#">10</a>
<a href="#">4.5.3.</a>	<a href="#">Sizes</a>	<a href="#">10</a>

□

channels

C

ioifaces

EndSocket.java

Smtplib\_C\_1\_Future.java

Smtplib\_C\_1.java

Smtplib\_C\_10.java

Smtplib\_C\_11\_Cases.java

Smtplib\_C\_11\_Handler.java

Smtplib\_C\_11.java

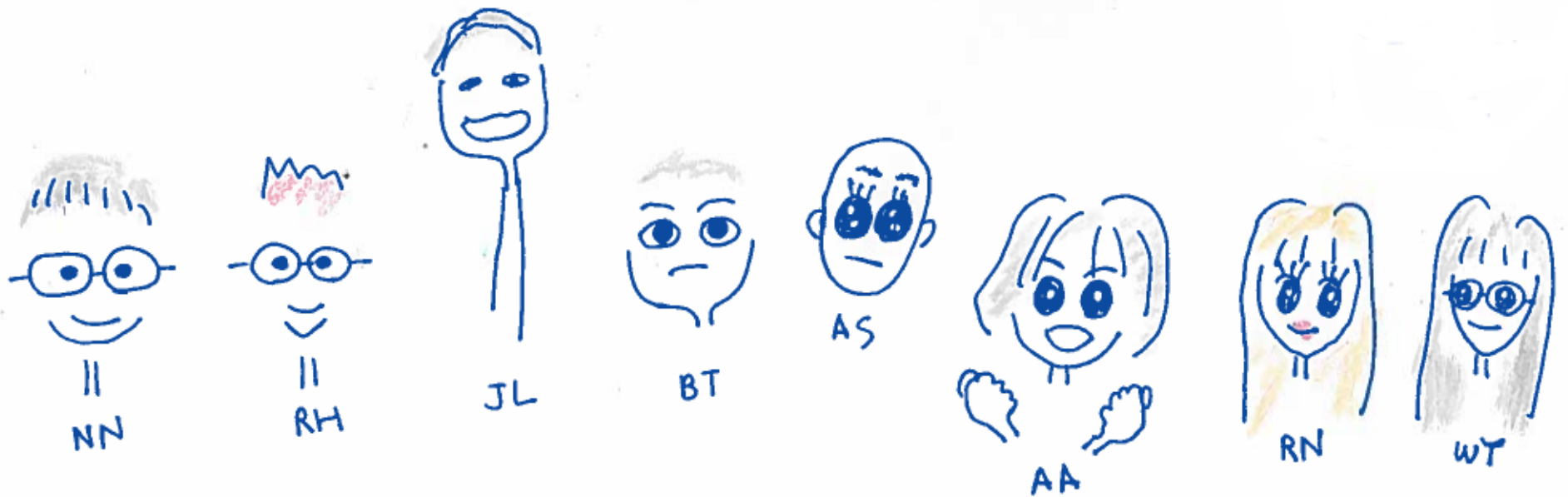
Smtplib\_C\_12.java

```
.send(Smtplib.S, new DataLine("Session  
.send(Smtplib.S, new EndOfData())  
.receive(Smtplib.S, Smtplib._250, new Buf  
.S
```

● send(S role, Mail m) : Smtplib\_C\_11 - Smtplib\_C\_10

● send(S role, Quit m) : EndSocket - Smtplib\_C\_10

# Session Type Mobility Group



[www.mrg.doc.ic.ac.uk](http://www.mrg.doc.ic.ac.uk)