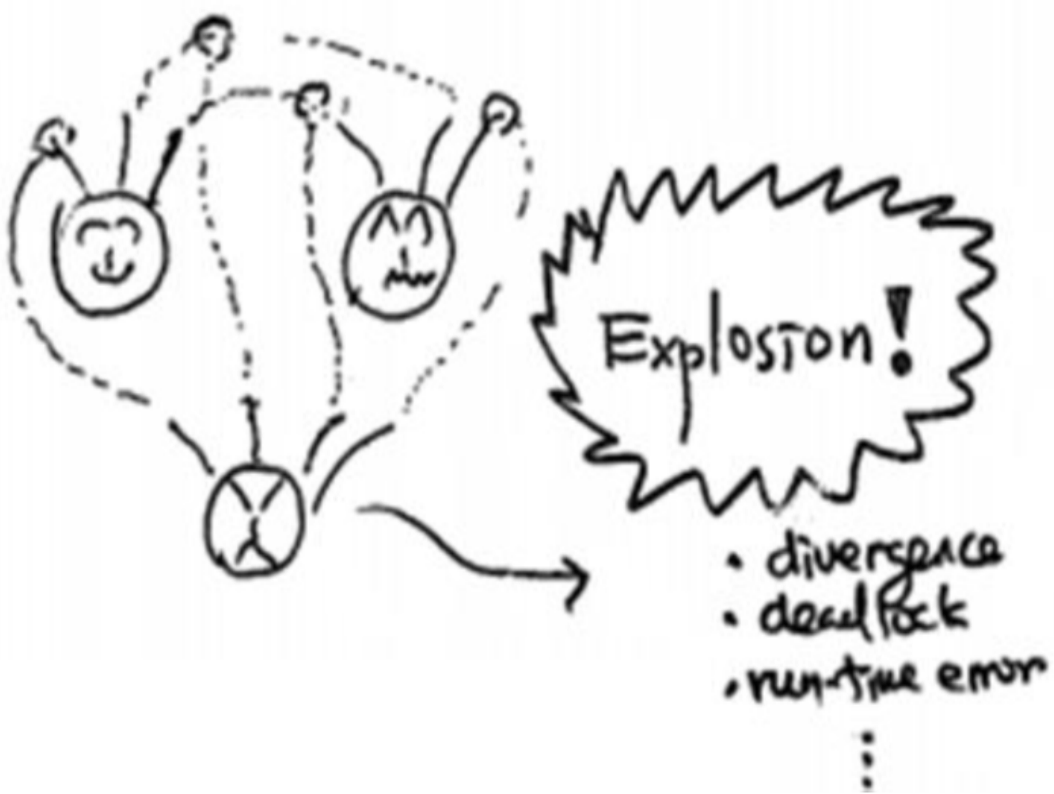
Session Types and Open Problems



Nobuko Yoshida

Betty Meeting 6th October 2016

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

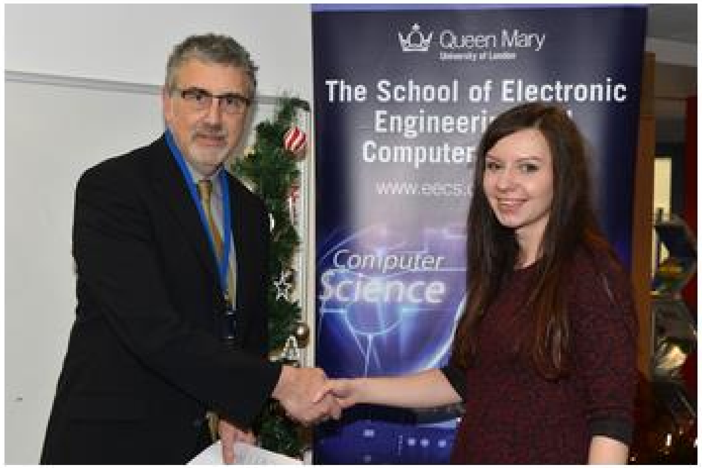
Posted with permission from QMUL on 17th Dec 2013. Original article written by Edmund Robinson.

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

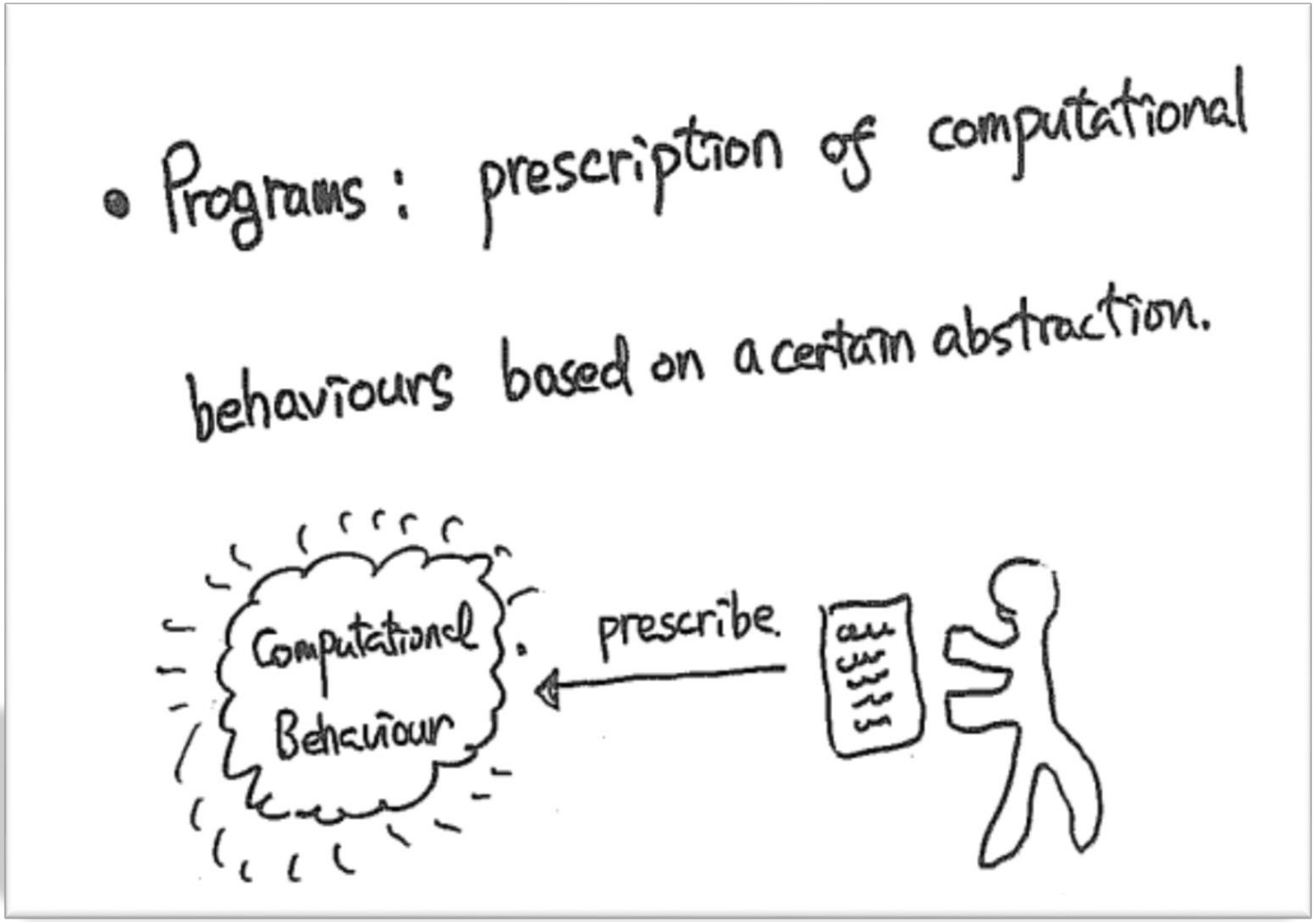


Ms Anna Pawlicka
2013 winner (Undergraduate) source: QMUL

Mr. Valdmir Negacevshi
2013 winner (Postgraduate) source: QMUL

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

- *Imperative*
- *Functional*
- *Logical*

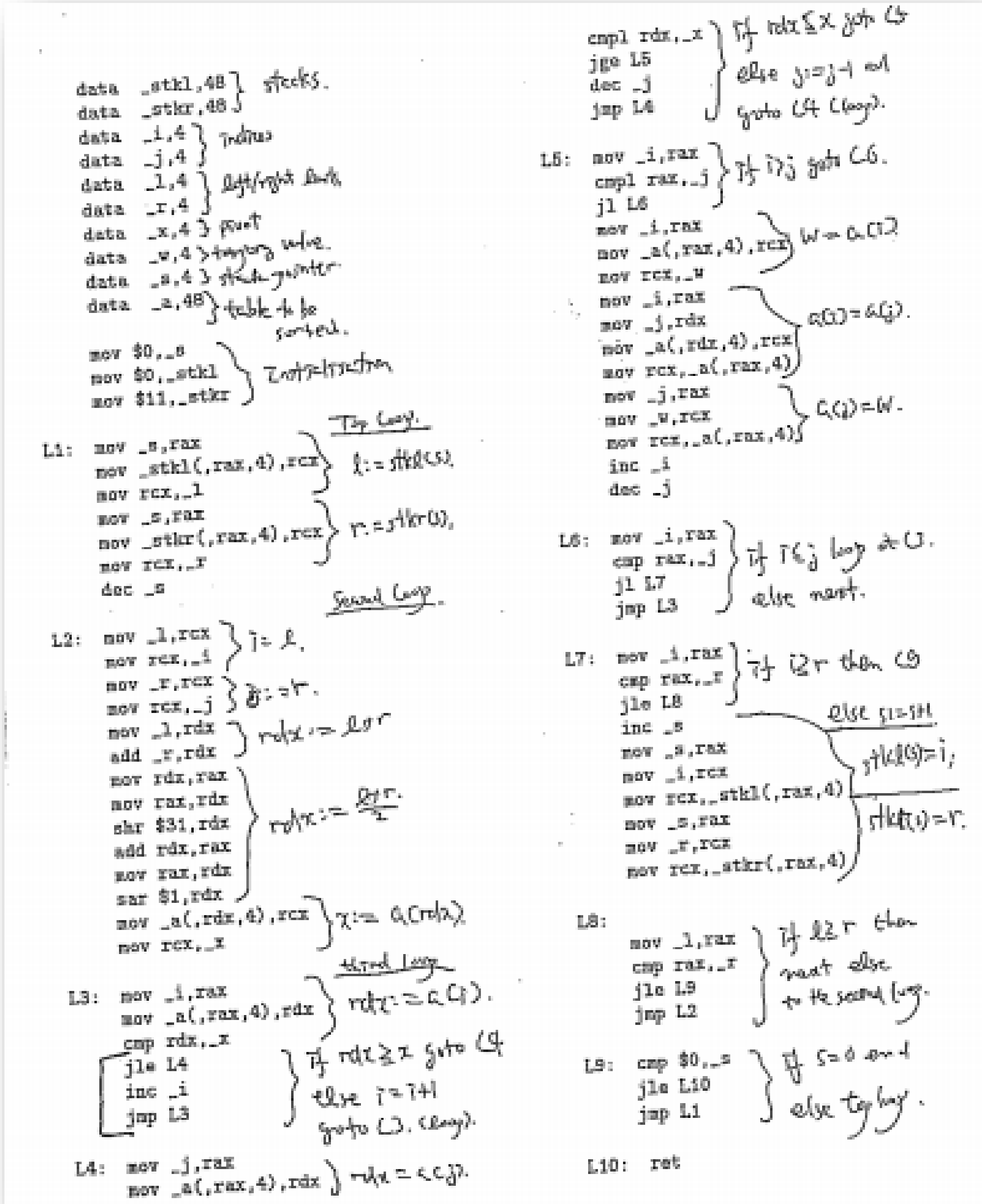


On Programs and Programming

- The most fundamental element of a PL in this context is a set of operations it is based on:
  - Imperative: assignment, jump.
  - Functional: β-reduction.
  - Logical: unification.

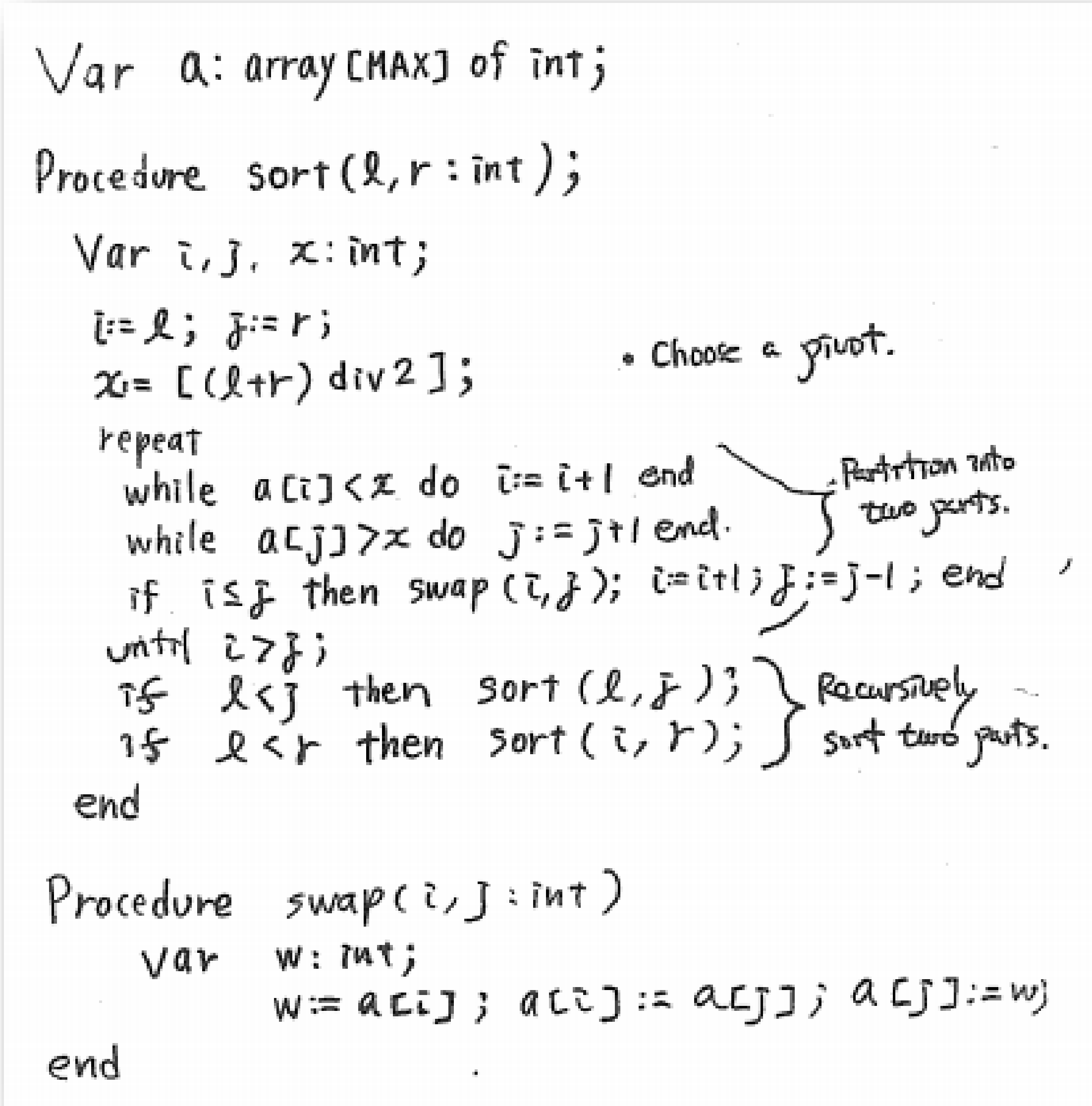
- Another element is how we can combine, or structure, these operations:
  - Imperative: sequential composition, if-then-else, while, procedures, module, ...
  - Functional: application, product, union, recursion, modules, ...

UNSTRUCTURED:



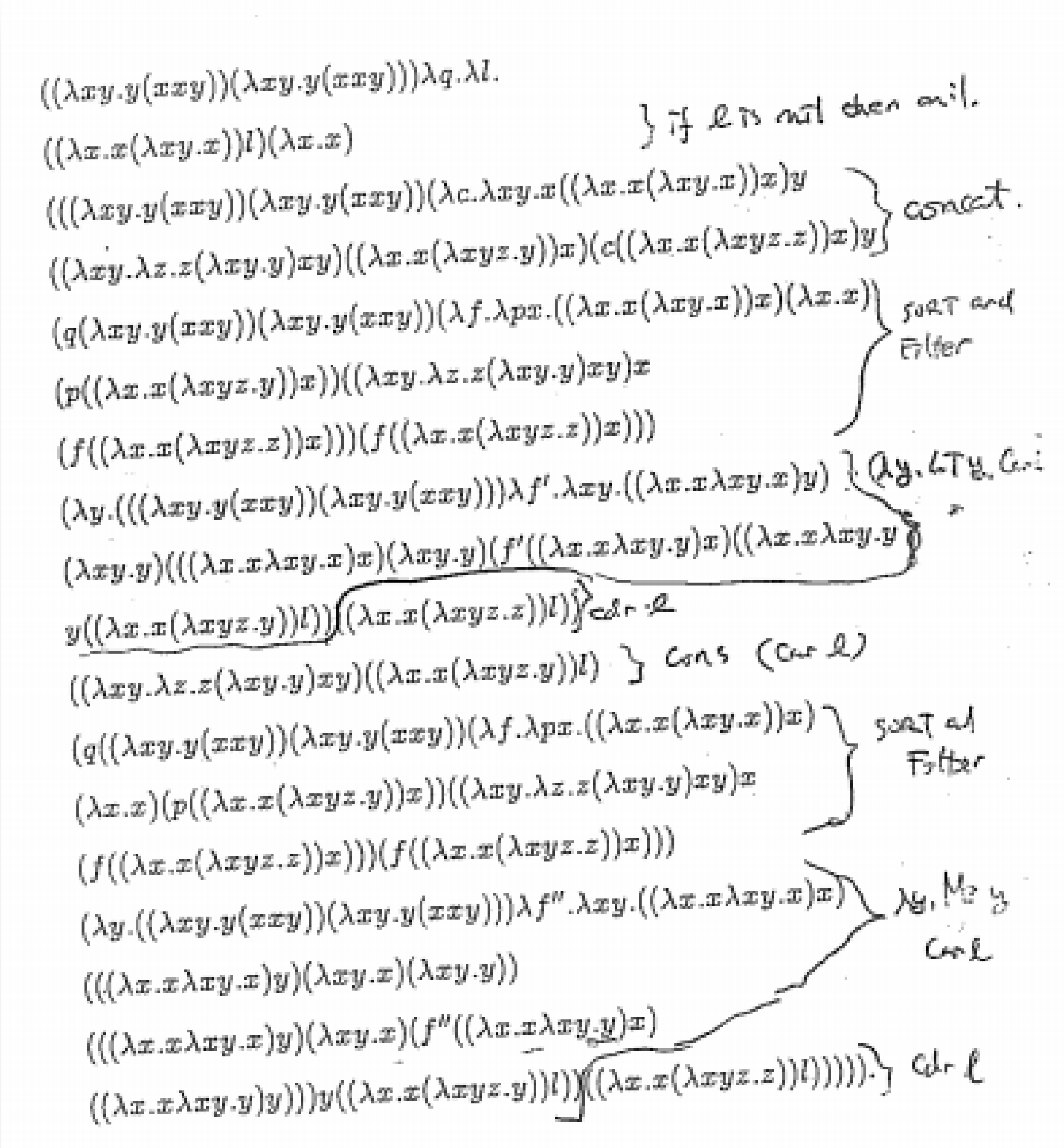
```
        data _stkl,48
        data _stkr,48
        data _i,4
        data _j,4
        data _l,4
        data _r,4
        data _x,4
        data _w,4
        data _s,4
        data _a,48

        mov $0,_s
        mov $0,_stkl
        mov $11,_stkr

L1:     mov _s,rax
        mov _stkl(,rax,4),rcx
        mov rcx,_l
        mov _s,rax
        mov _stkr(,rax,4),rcx
        mov rcx,_r
        dec _s

L2:     mov _l,rcx
        mov rcx,_i
        mov _r,rcx
        mov rcx,_j
        mov _l,rdx
        add _r,rdx
        mov rdx,rax
        mov rax,rdx
        shr $31,rdx
        add rdx,rax
        mov rax,rdx
        sar $1,rdx
        mov _a(,rdx,4),rcx
        mov rcx,_x

L3:     mov _i,rax
        mov _a(,rax,4),rdx
        cmp rdx,_x
        jle L4
        inc _i
        jmp L3

L4:     mov _j,rax
        mov _a(,rax,4),rdx
        cmpl rdx,_x
        jge L5
        dec _j
        jmp L4

L5:     mov _i,rax
        cmpl rax,_j
        jl L6
        mov _i,rax
        mov _a(,rax,4),rcx
        mov rcx,_w
        mov _i,rax
        mov _j,rdx
        mov _a(,rdx,4),rcx
        mov rcx,_a(,rax,4)
        mov _j,rax
        mov _w,rcx
        mov rcx,_a(,rax,4)
        inc _i
        dec _j

L6:     mov _i,rax
        cmp rax,_j
        jl L7
        jmp L3

L7:     mov _i,rax
        cmp rax,_r
        jle L8
        inc _s
        mov _s,rax
        mov _i,rcx
        mov rcx,_stkl(,rax,4)
        mov _s,rax
        mov _r,rcx
        mov rcx,_stkr(,rax,4)

L8:     mov _l,rax
        cmp rax,_r
        jle L9
        jmp L2

L9:     cmp $0,_s
        jle L10
        jmp L1

L10:    ret
```

STRUCTURED:



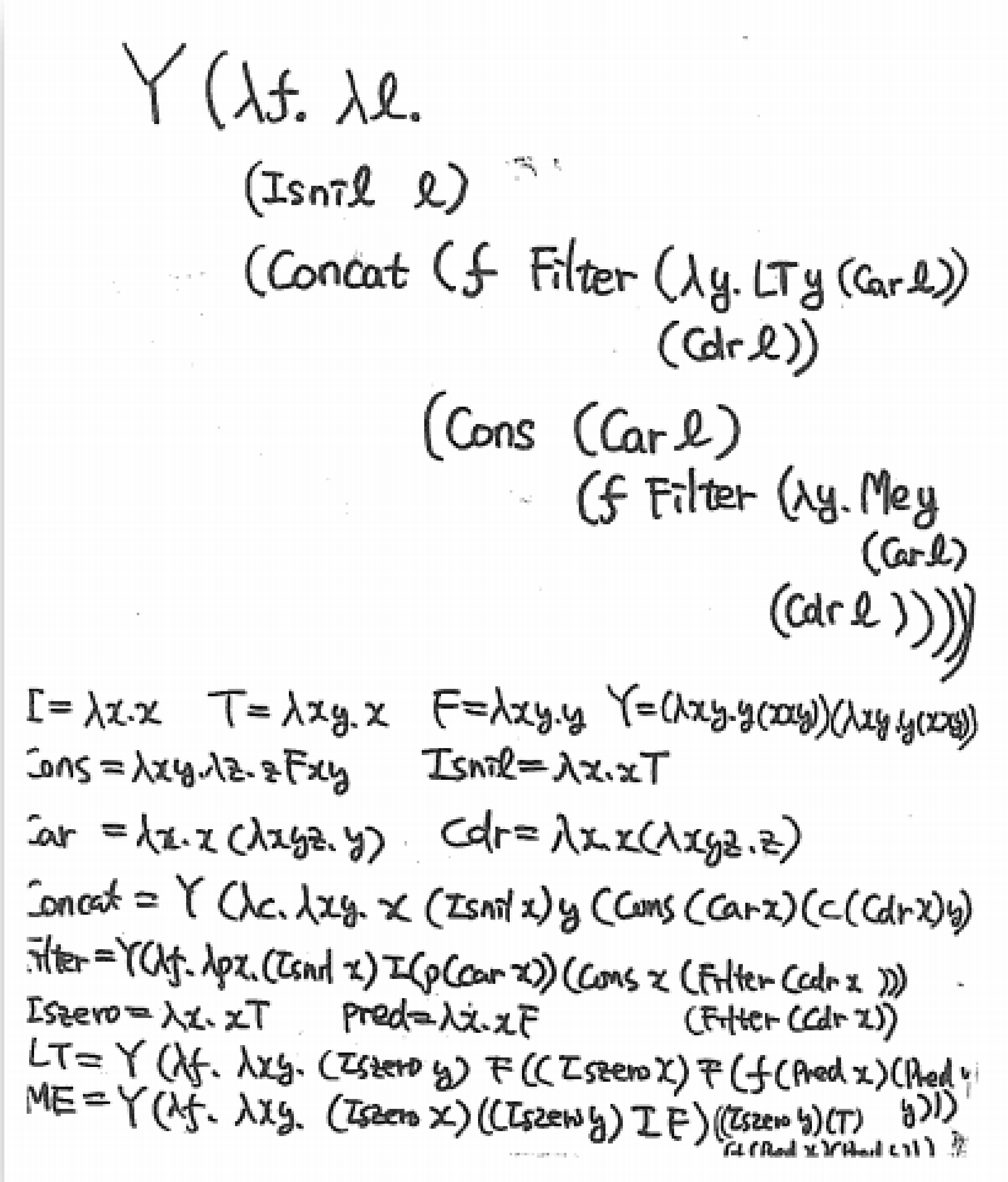
```
Var a: array[MAX] of int;

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        } Partition into
    while a[j]>x do j:=j+1 end.       } two parts.
    if i≤j then swap(i,j); i:=i+1; j:=j-1; end
   until i>j;
   if l<j then sort(l,j);   } Recursively
   if l<r then sort(i,r);   } sort two parts.
  end

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

Quicksort in pure lambda:



((λxy.y(xxy))(λxy.y(xxy)))λq.λl.
((λx.x(λxy.x))l)(λx.x) } if l is nil then nil.
(((λxy.y(xxy))(λxy.y(xxy))(λc.λxy.x((λx.x(λxy.x))x)y
((λxy.λz.z(λxy.y)xy)((λx.x(λxyz.y))x)(c((λx.x(λxyz.z))x)y } concat.
(q(λxy.y(xxy))(λxy.y(xxy))(λf.λpx.((λx.x(λxy.x))x)(λx.x)
(p((λx.x(λxyz.y))x))((λxy.λz.z(λxy.y)xy)x
(f((λx.x(λxyz.z))x)))(f((λx.x(λxyz.z))x))) } sort and Filter
(λy.(((λxy.y(xxy))(λxy.y(xxy)))λf′.λxy.((λx.xλxy.x)y) } (λy. LT y. Car
(λxy.y)(((λx.xλxy.x)x)(λxy.y)(f′((λx.xλxy.y)x)((λx.xλxy.y
y((λx.x(λxyz.y))l))((λx.x(λxyz.z))l)} cdr l
((λxy.λz.z(λxy.y)xy)((λx.x(λxyz.y))l) } Cons (Car l)
(q((λxy.y(xxy))(λxy.y(xxy))(λf.λpx.((λx.x(λxy.x))x)
(λx.x)(p((λx.x(λxyz.y))x))((λxy.λz.z(λxy.y)xy)x
(f((λx.x(λxyz.z))x)))(f((λx.x(λxyz.z))x))) } sort and Filter
(λy.((λxy.y(xxy))(λxy.y(xxy)))λf″.λxy.((λx.xλxy.x)x)
(((λx.xλxy.x)y)(λxy.x)(λxy.y)) } λy. Me y Car l
(((λx.xλxy.x)y)(λxy.x)(f″((λx.xλxy.y)x)
((λx.xλxy.y)y)))y((λx.x(λxyz.y))l))((λx.x(λxyz.z))l))))). } Cdr l

Quicksort with combinators:



Y (λf. λl.
(Isnil l)
(Concat (f Filter (λy. LT y (Car l))
(Cdr l))
(Cons (Car l)
(f Filter (λy. Me y
(Car l)
(Cdr l)))))

I = λx.x T = λxy.x F = λxy.y Y = (λxy.y(xxy))(λxy.y(xxy))
Cons = λxy.λz.zFxy Isnil = λx.xT
Car = λx.x(λxyz.y) Cdr = λx.x(λxyz.z)
Concat = Y (λc.λxy. x (Isnil x) y (Cons (Car x)(c (Cdr x) y)
Filter = Y(λf. λpx.(Isnil x) I (p(Car x))(Cons x (Filter (Cdr x)))
(Filter (Cdr x))
Iszero = λx. xT Pred = λx.xF
LT = Y (λf. λxy. (Iszero y) F ((Iszero x) F (f (Pred x)(Pred y)))
ME = Y (λf. λxy. (Iszero x) ((Iszero y) I F) ((Iszero y)(T)
(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 π-calculus as a Descriptive Tool

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

$\pi \qquad P ::= \Sigma \pi_i . P_i \mid P|Q \mid (\nu x)P \mid !P \mid 0.$

with $\pi ::= x(\tilde{y}) \mid \overline{x}\langle \tilde{y} \rangle.$

λ in π

$$[\![x]\!]_u \stackrel{\text{def}}{=} \overline{x}\langle u \rangle.$$

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

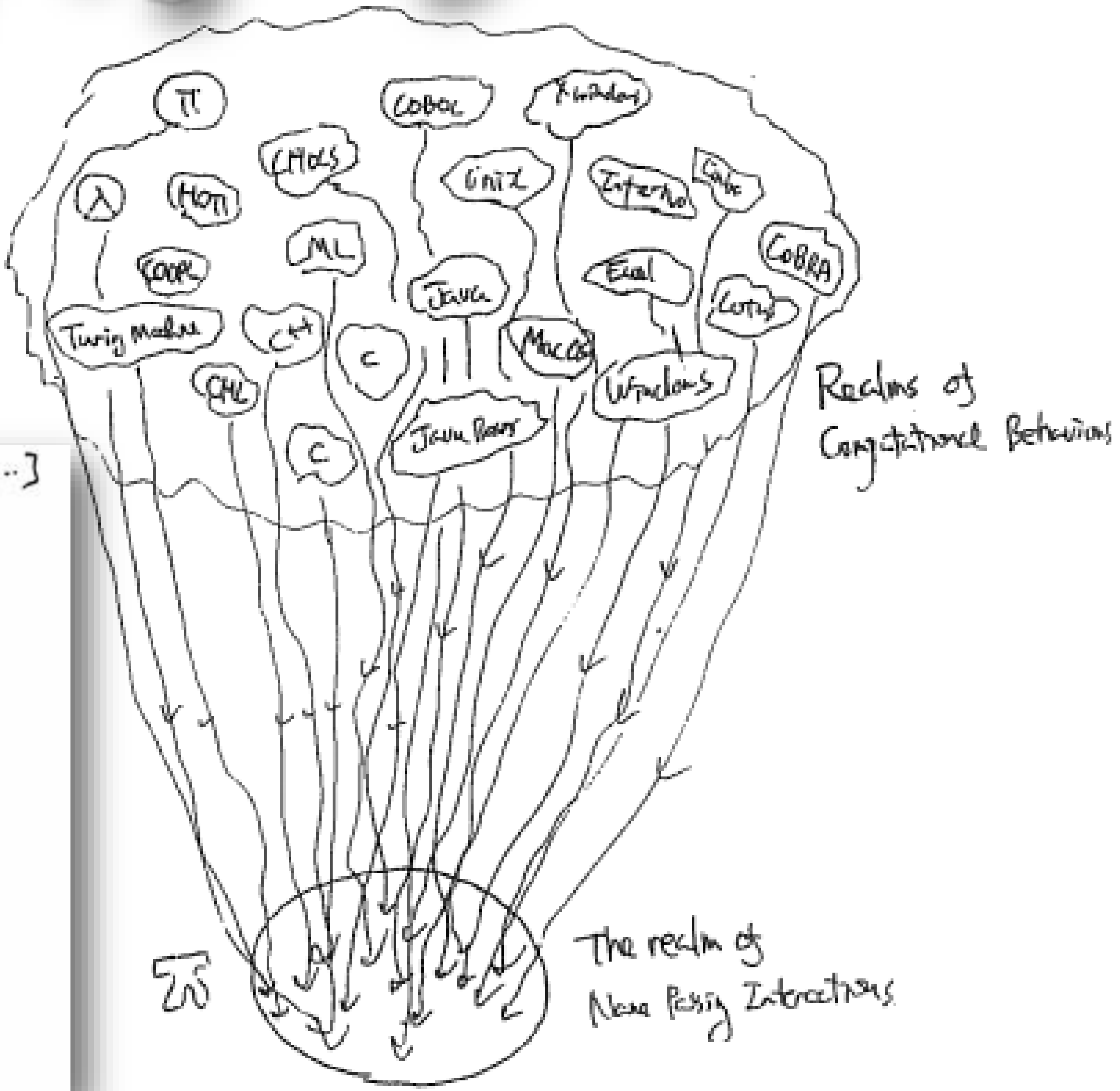
$$[\![MN]\!]_u \stackrel{\text{def}}{=} (\nu f x)\left([\![M]\!]_f \mid \overline{f}\langle x u \rangle \mid [\![x = N]\!]\right)$$

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

* Examples of Representable Computation.

- λ-calculus [MPW89, Milner90, Milner92, …]
- Concurrent Object [Walker91]
- ω-order term passing [Sangiorgi 92]
- Various data structures [Milner 92, …]
- Proof Nets [Bellin and Scott 93]
- Arbitrary 'constant' interaction [HY94]
- strategies on Games [HO95]

⋮

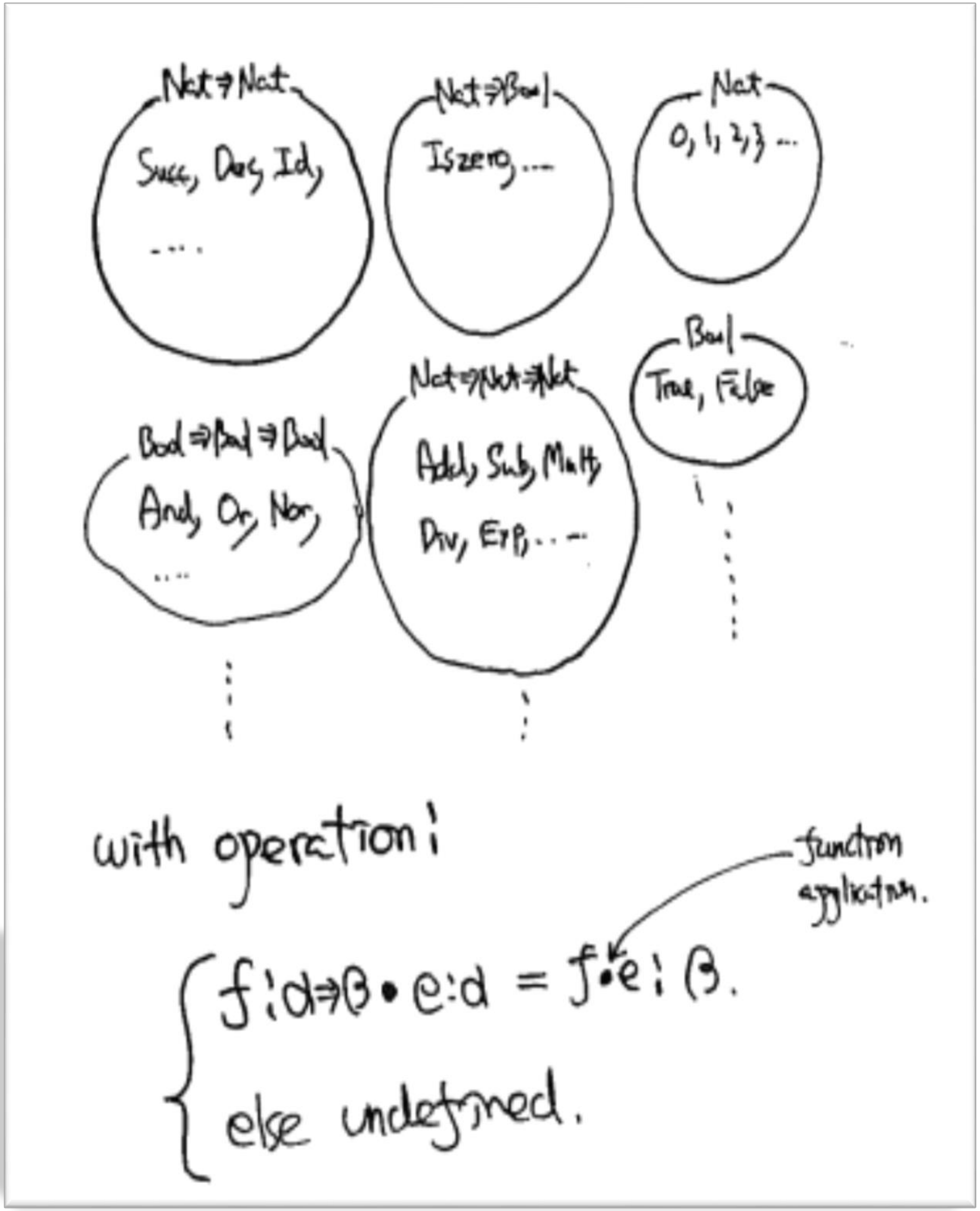


The Role of Types in π-calculus

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

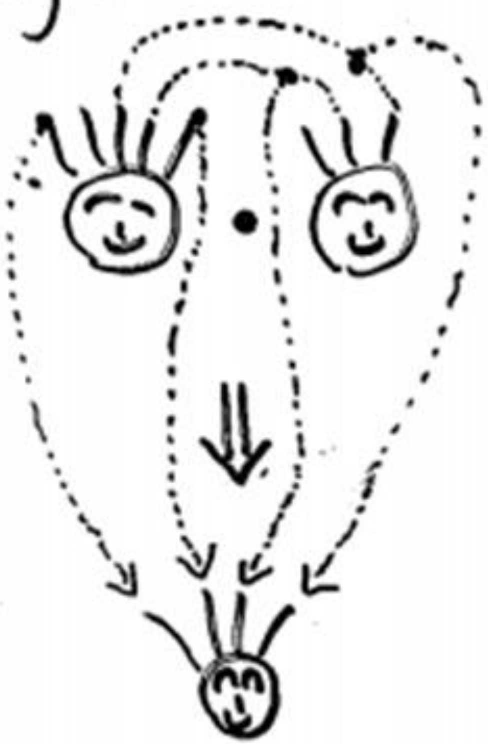
- (safety) Is this program/system in the safe (or correct, relevant, ...) classes of behaviours? Can the safety be preserved compositionally?

Functional Types



Process Types

- When it comes to processes, composition becomes:



cf.

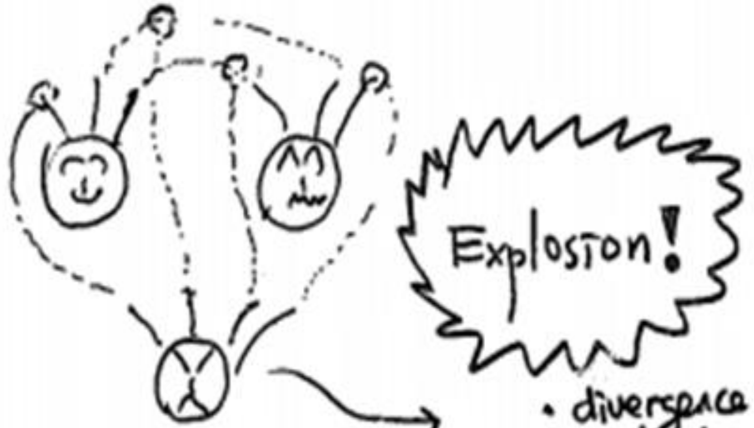
3 + 5

⇓

8

$P \bullet_{\delta} \delta \longmapsto_{\tau} \sigma$

- But some composition is dangerous!



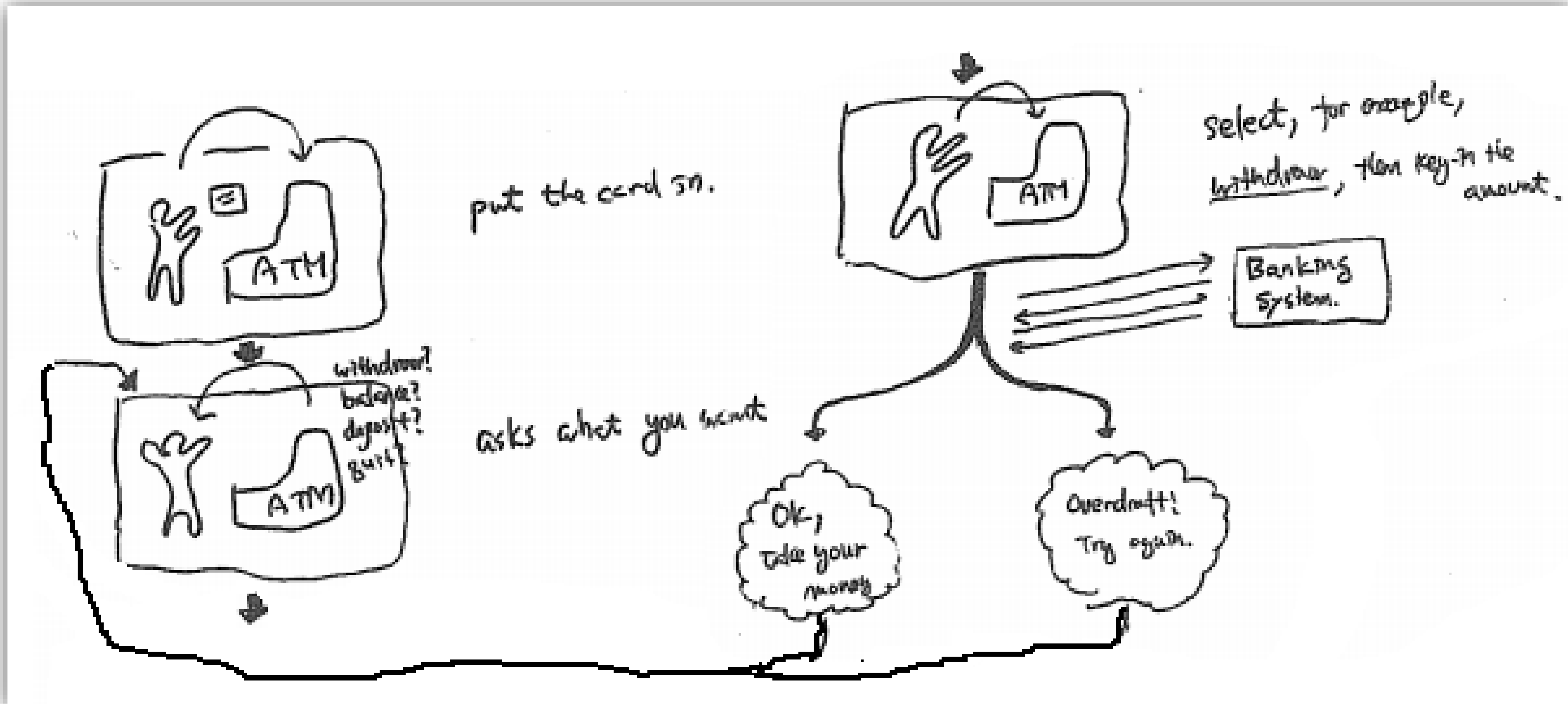
- divergence
- deadlock
- run-time error

⋮

- Therefore we type processes,

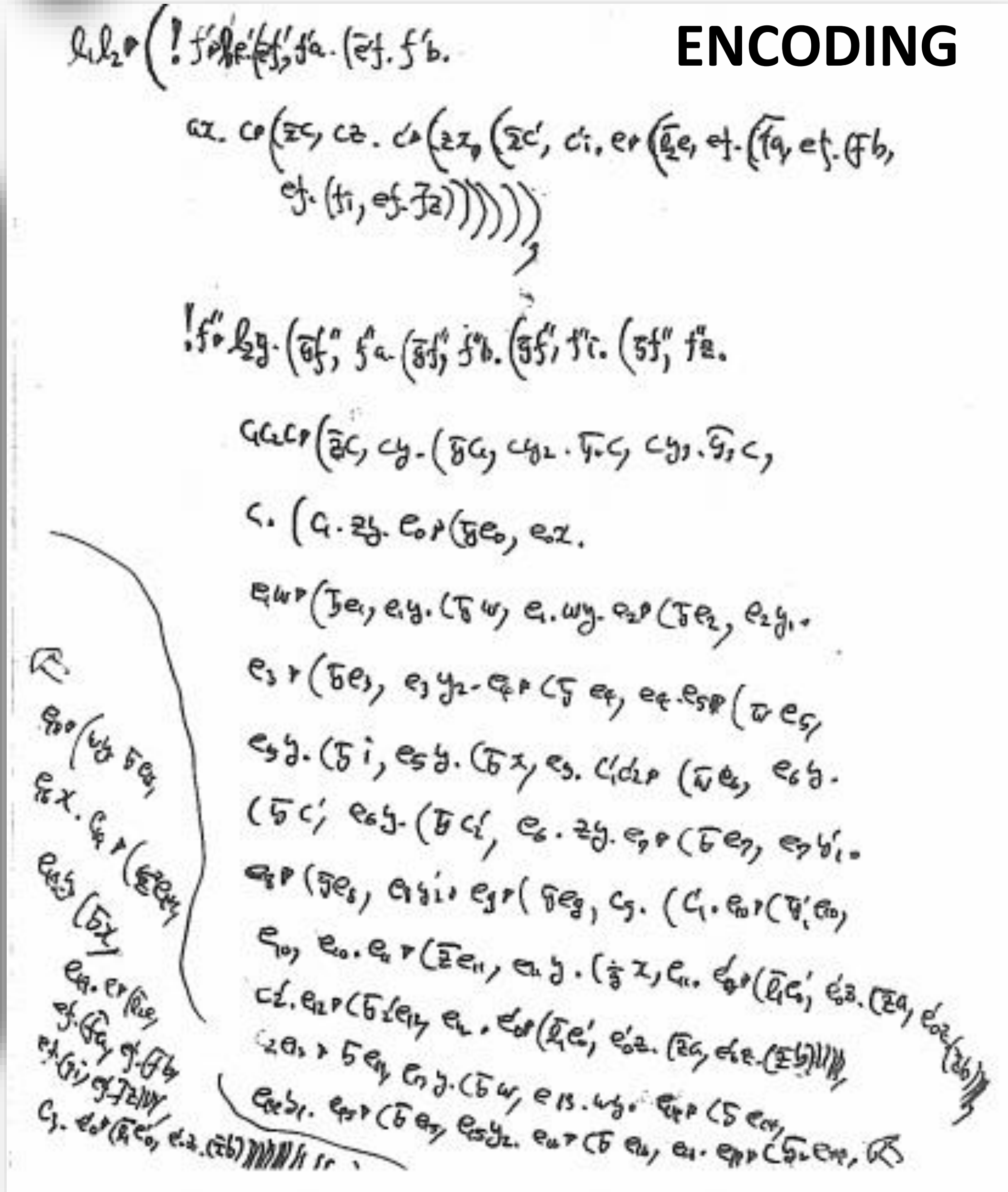
The connection is prohibited

Implementing ATM

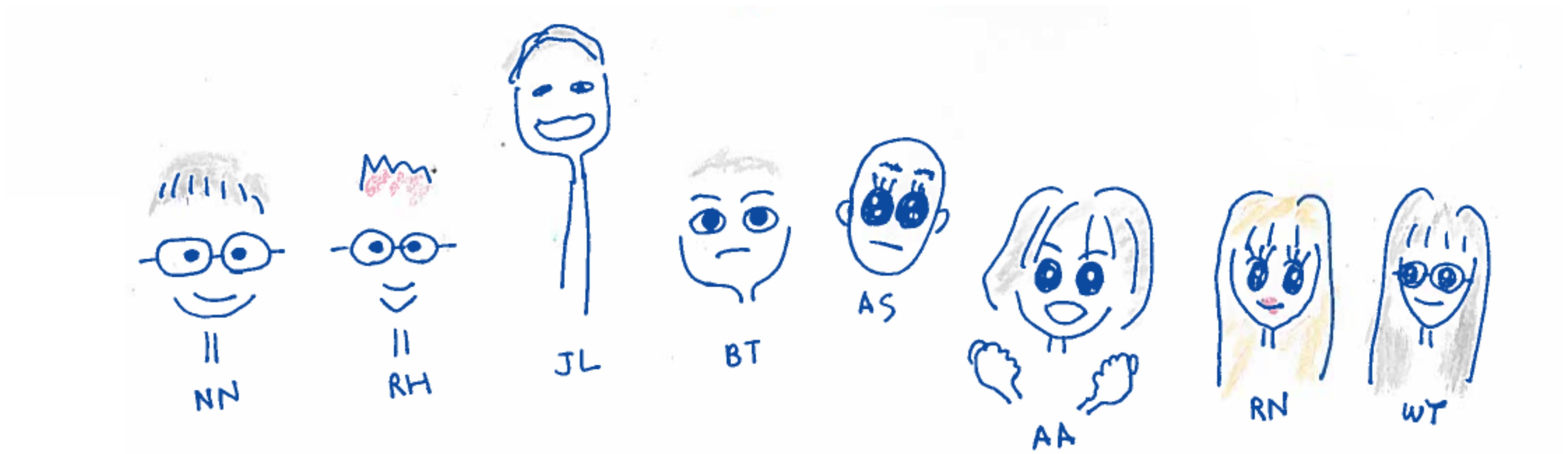


Implementing ATM

```
ATM(cb) ≝

a<z> :: z?ID; [?wd; ?X;                     % ↔ user.
        b<w> :: w!wt; !ID; !X;              % ↔ bank.
          [?ok;                             %
             z!ok; !X; ATM(cb),             % ↔ user
           ?overdraft:                      % ↔ bank
             z!over; ATM(cb)],              % ↔ user.
      ?bal;                                 % ↔ user
        b<w> :: w!bal; ?X;                  % ↔ bank
             z!X; ATM(cb)]
```



Session Type Mobility Group



www.mrg.doc.ic.ac.uk

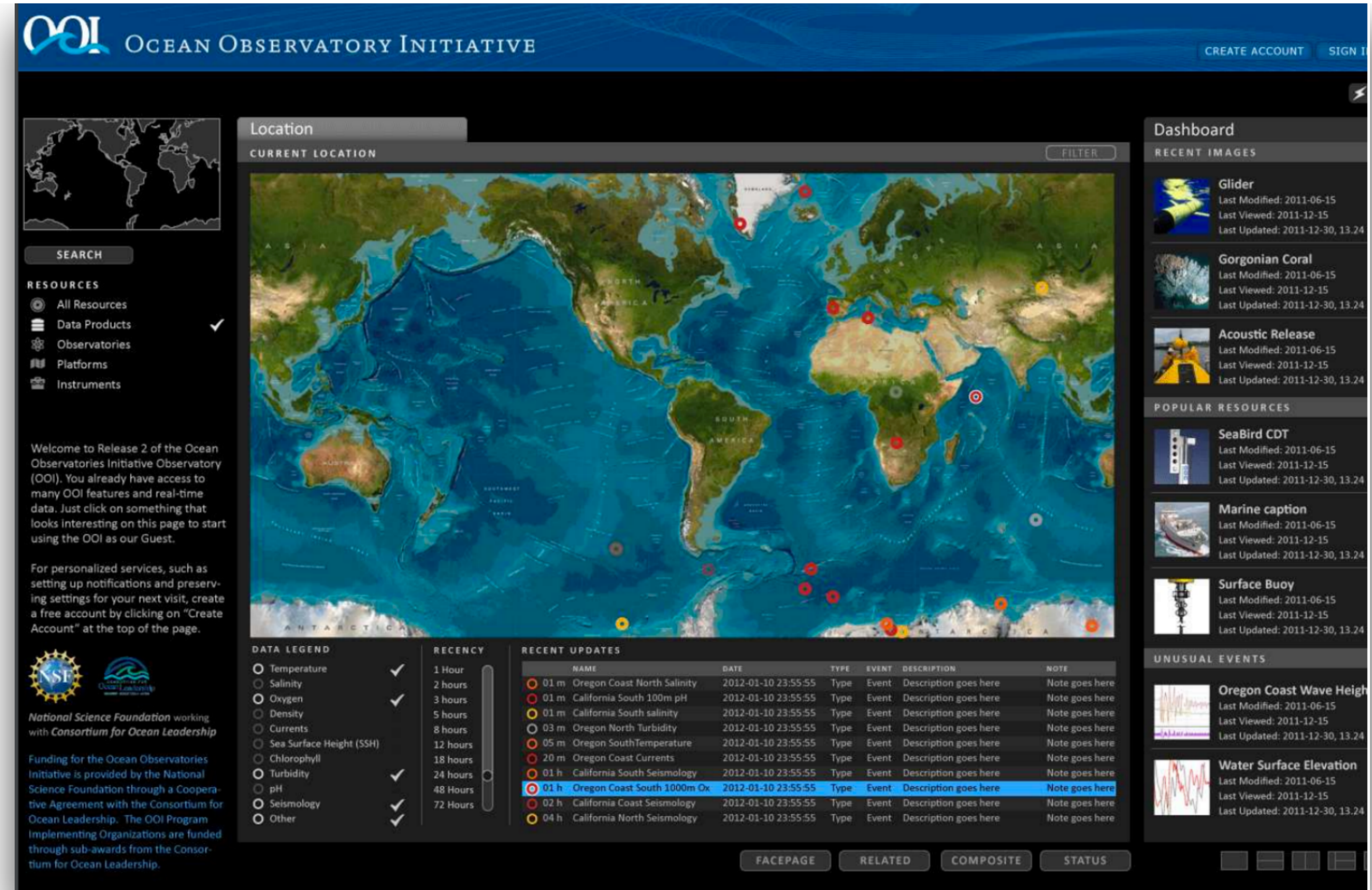
Selected Publications 2015/2016

- **[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

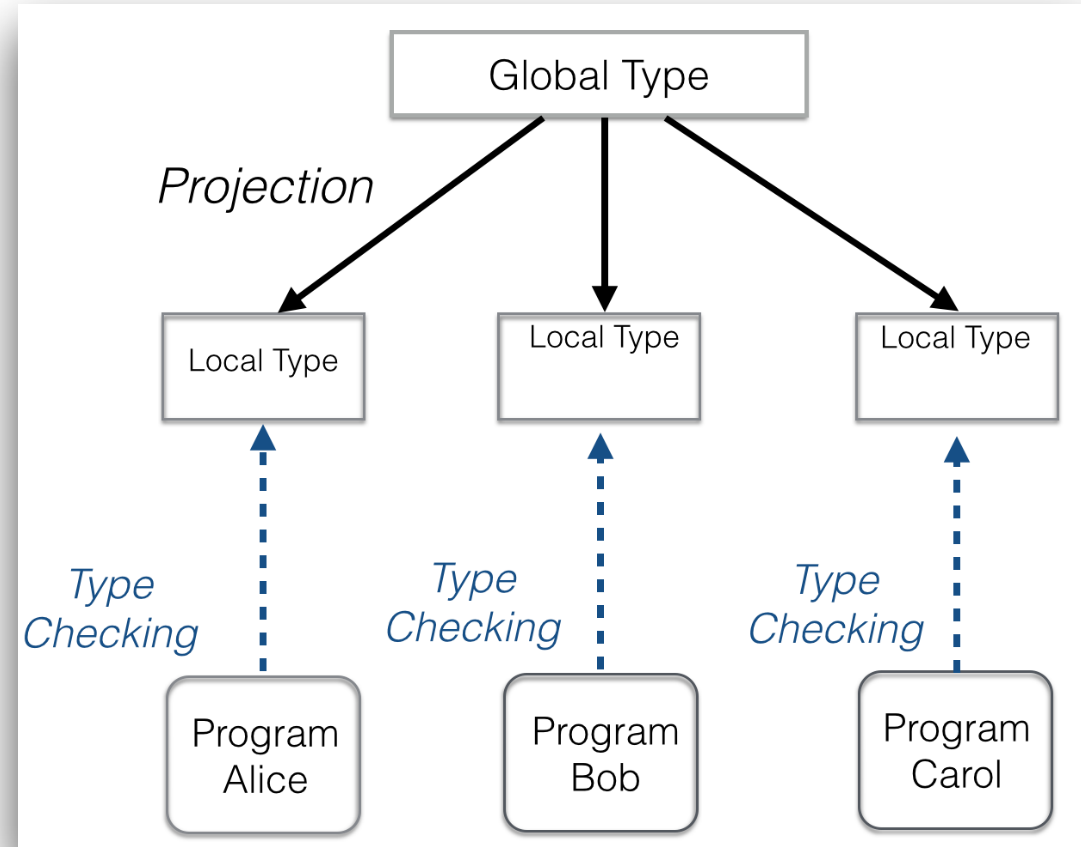
- **[FPL'16]** Xinyu Niu , Nicholas Ng , Tomofumi Yuki , Shaojun Wang , NY, Wayne Luk : EURECA Compilation: Automatic Optimisation of Cycle-Reconfigurable Circuits.
- **[ECOOP'16]** Alceste Scala, NY: Lightweight Session Programming in Scala
- **[CC'16]** Nicholas Ng, NY: Static Deadlock Detection for 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



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

Session Types Overview



- Global session type

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

- 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

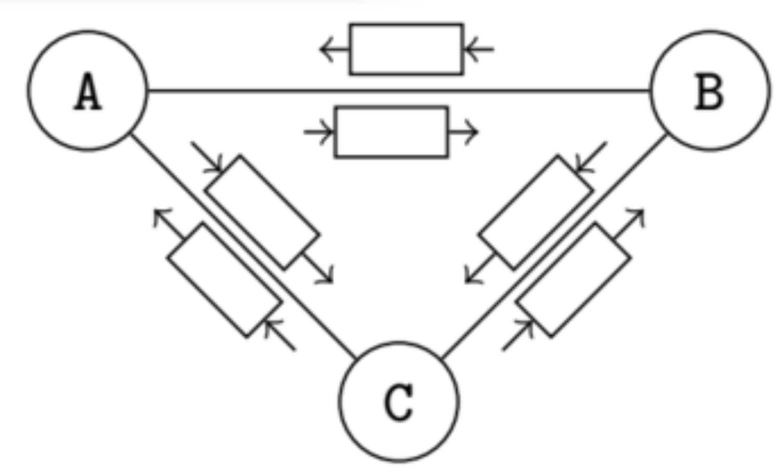
The Scribble Protocol Language

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



www.scribble.org

Home Getting Started Downloads Documentation ▾ Community ▾

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/

```
module examples;

global protocol HelloWorld(role Me, role World) {
  hello() from Me to World;
  choice at World {
    goodMorning1() from World to Me;
  } or {
    goodMorning1() from World to Me;
  }
}

```

Load a sample | Check | Protocol: examples.HelloWorld | Role: Me | 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

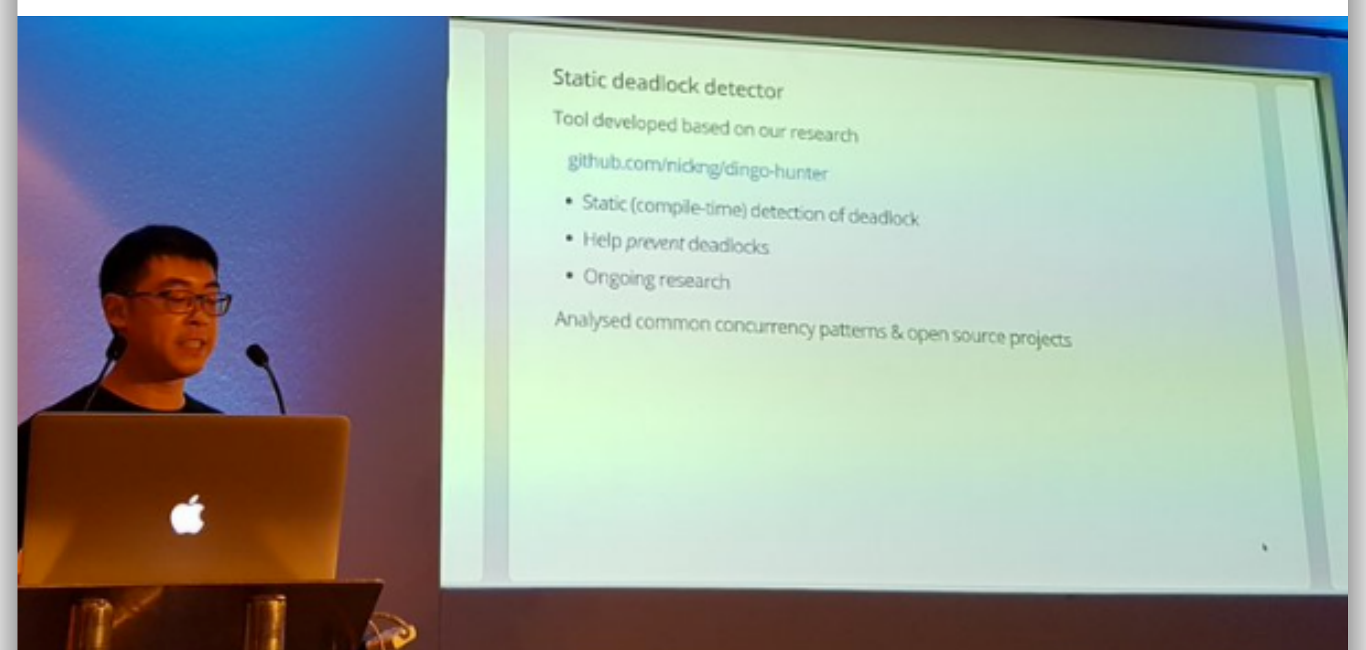
Interactions with Industries

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.

Nobuko Yoshida
Imperial College, London

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

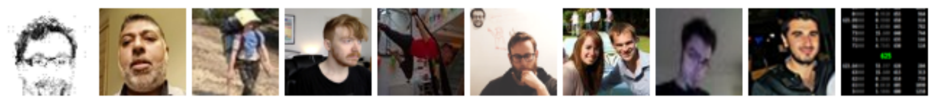
.@nicholascwng rocking on @GolangUKconf about static deadlock detection in #golang #gouk16

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

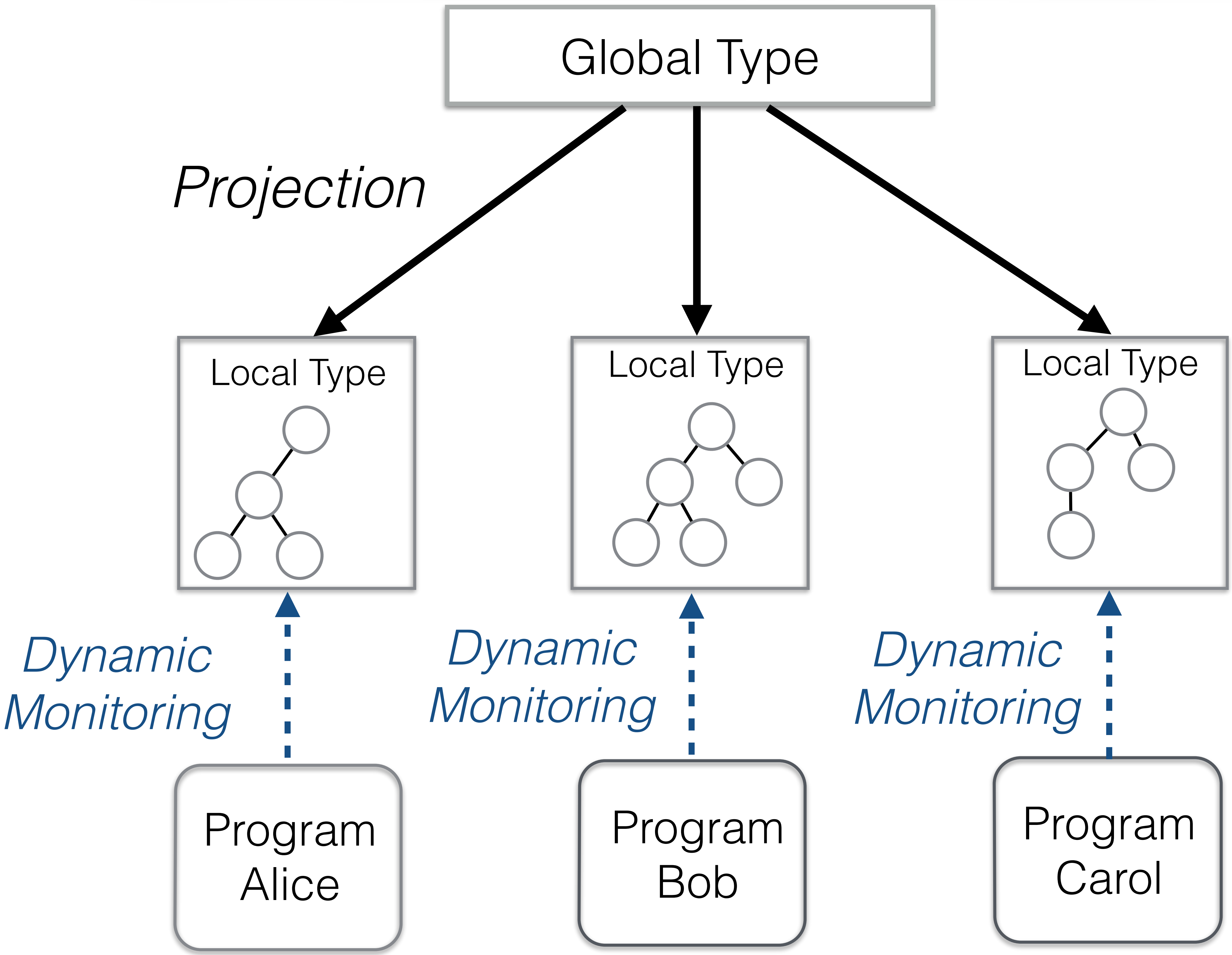
Dr. Roland Kuhn
@rolandkuhn — *CTO of Actyx*

actyx

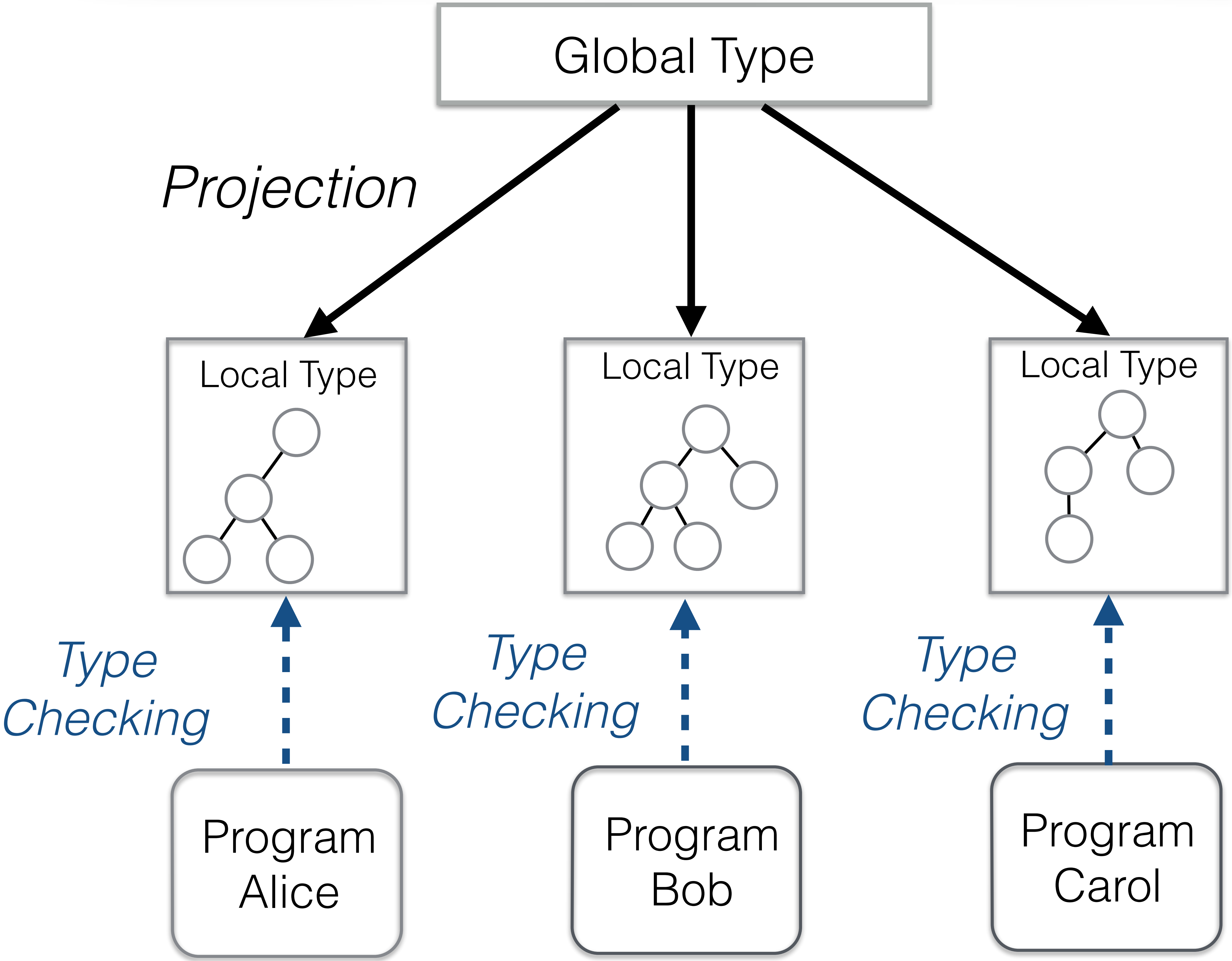
Current State

- 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)

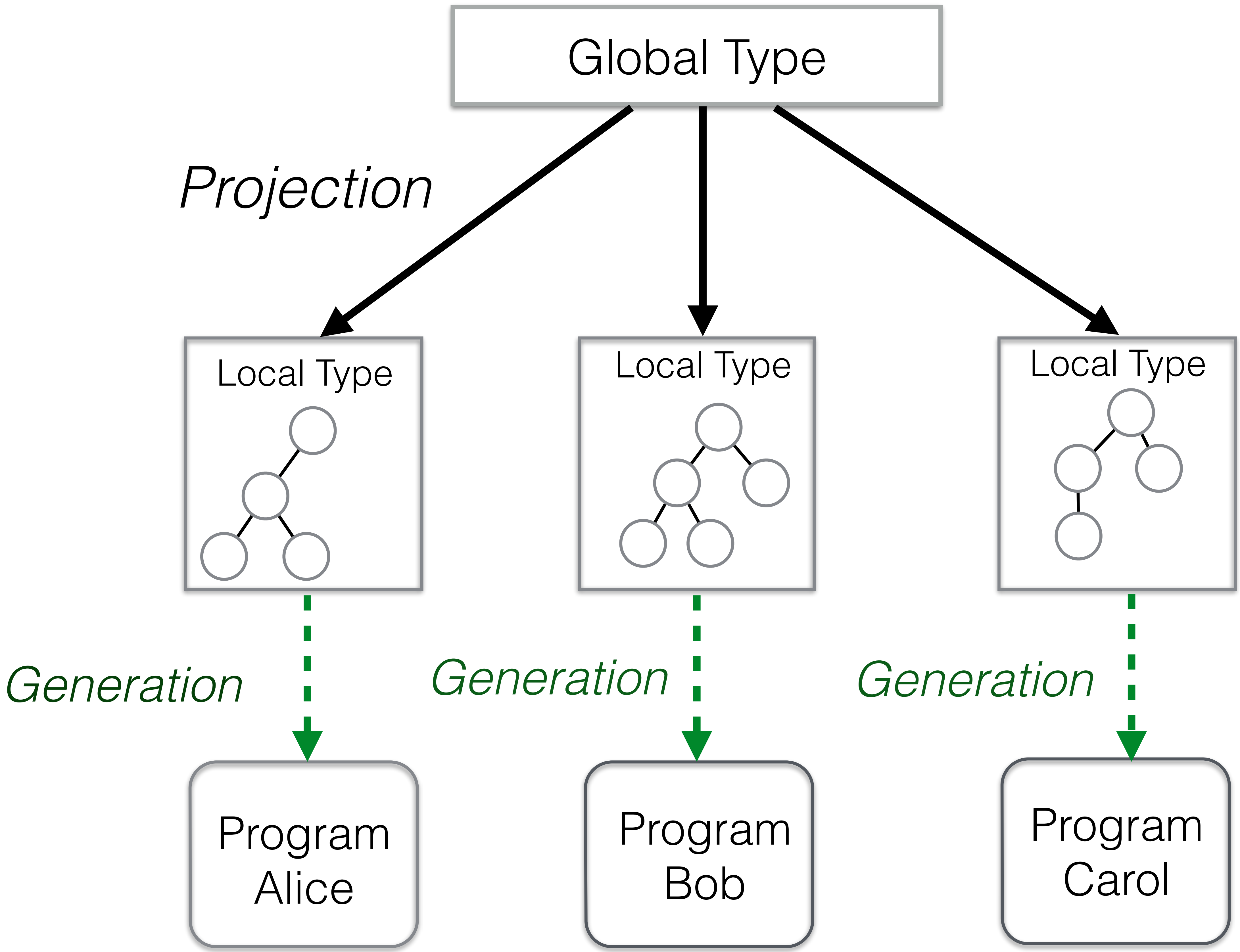
Dynamic Monitoring
[RV'13, COORDINATION'14, FMSD'15]



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

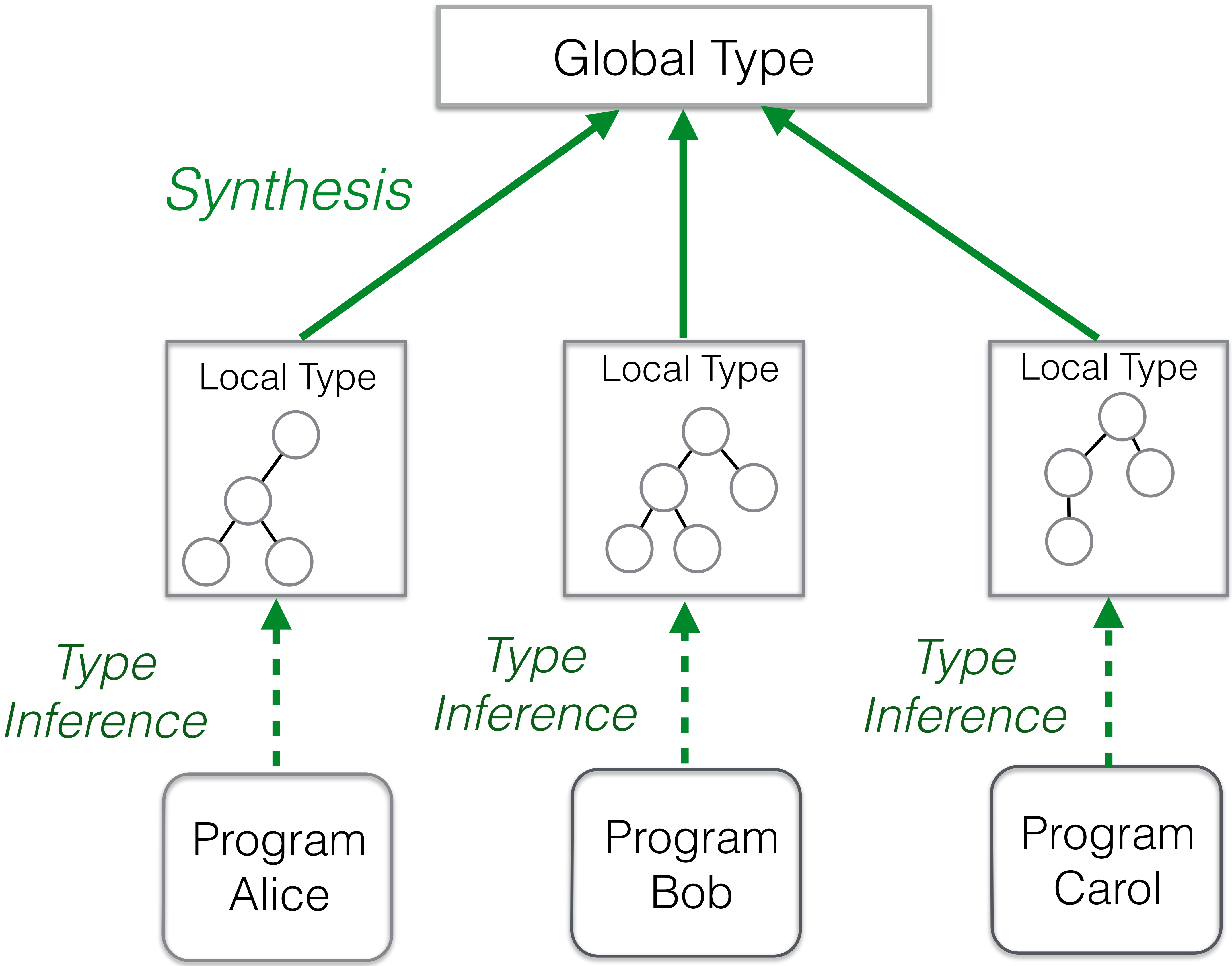


Code Generation [CC'15, FASE'16]



Synthesis

[ICALP'13, POPL'15, CONCUR'15, TACAS'16, CC'16]



GLOBALLY GOVERNED SESSION SEMANTICS

Dmitrios Kouzapas

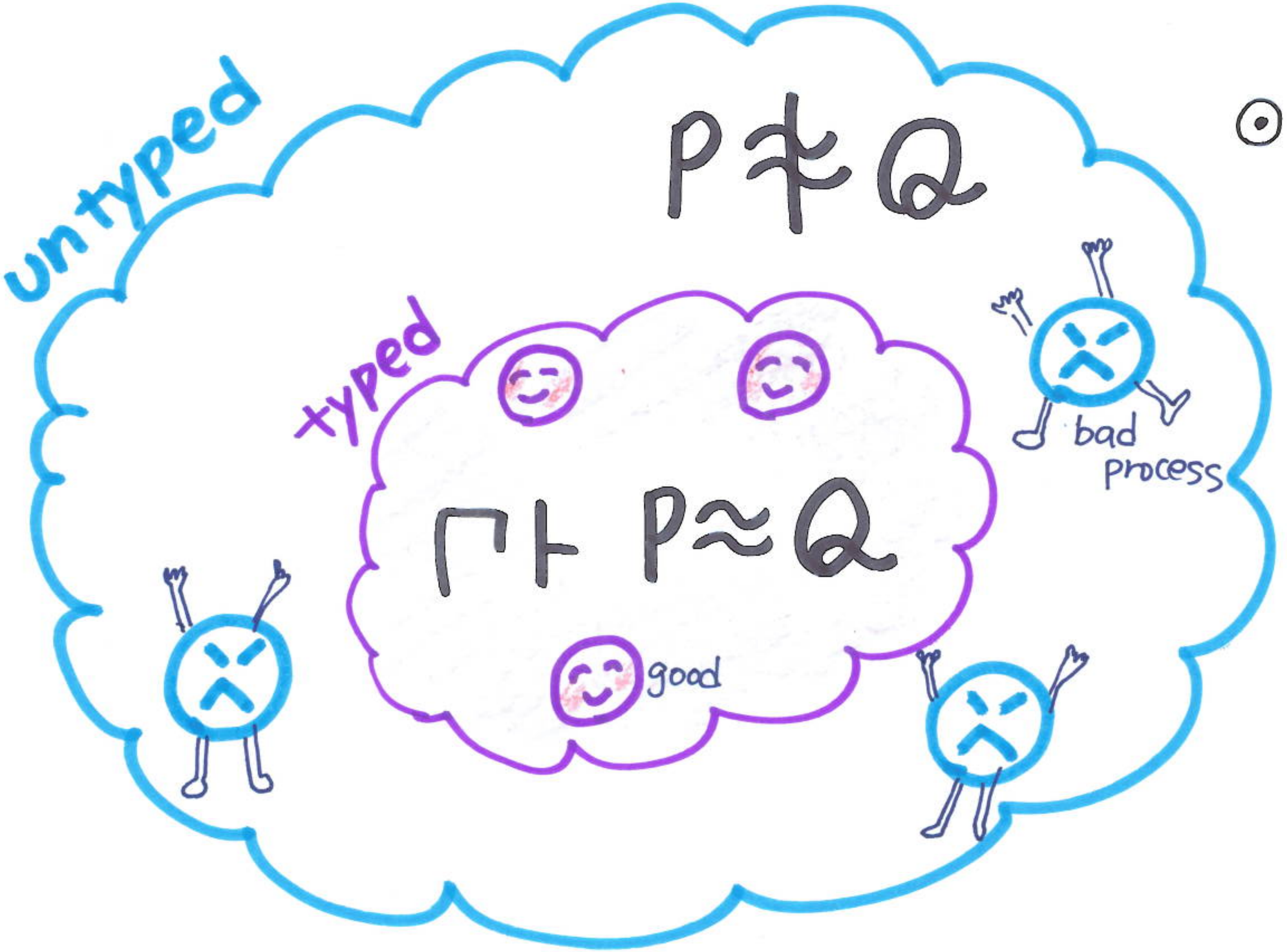
Glasgow

Nobuko Yoshida

Imperial College London

Typed Semantics in π 1991 →

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



- ⊙ Correctness of Encoding
- ⊙ Limit environments (Γ)
  ⇒ Equate <u>more</u> 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

$s_1 : G_1,\ s_2 : G_2,\ \ldots,\ s_n : G_n$

Governed Bisimulations

? Compositional? Coincides with Contextual Equiv?

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

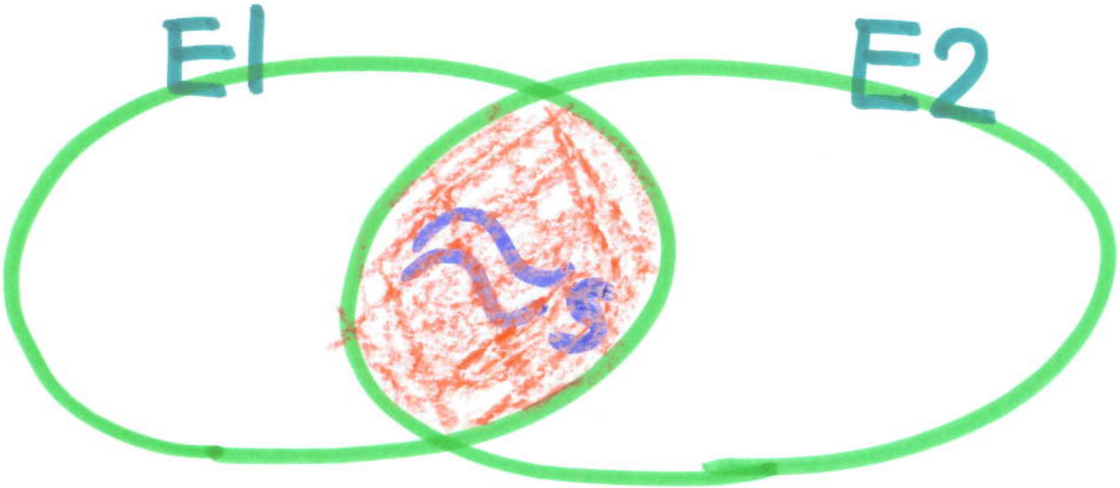
? Under what condition $\approx_s$ and $\approx_g$ can coincide?

? 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)

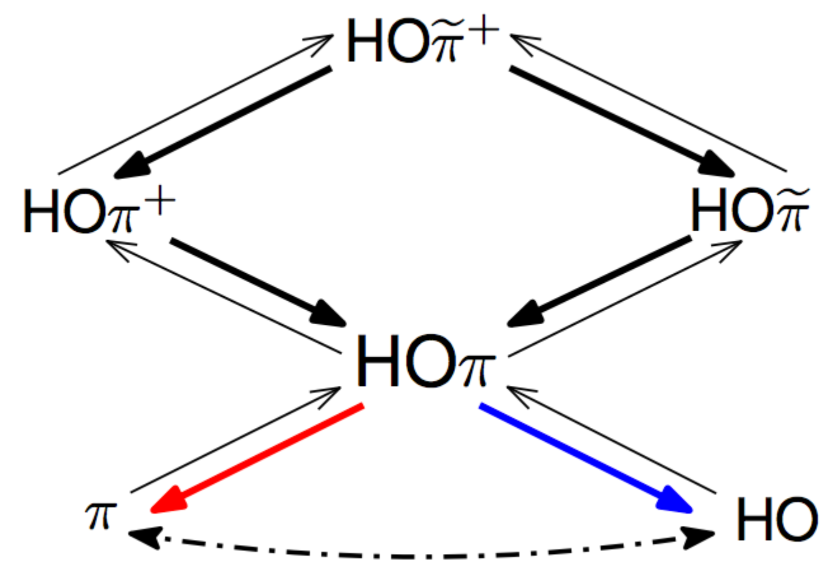
no interleaved sessions

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



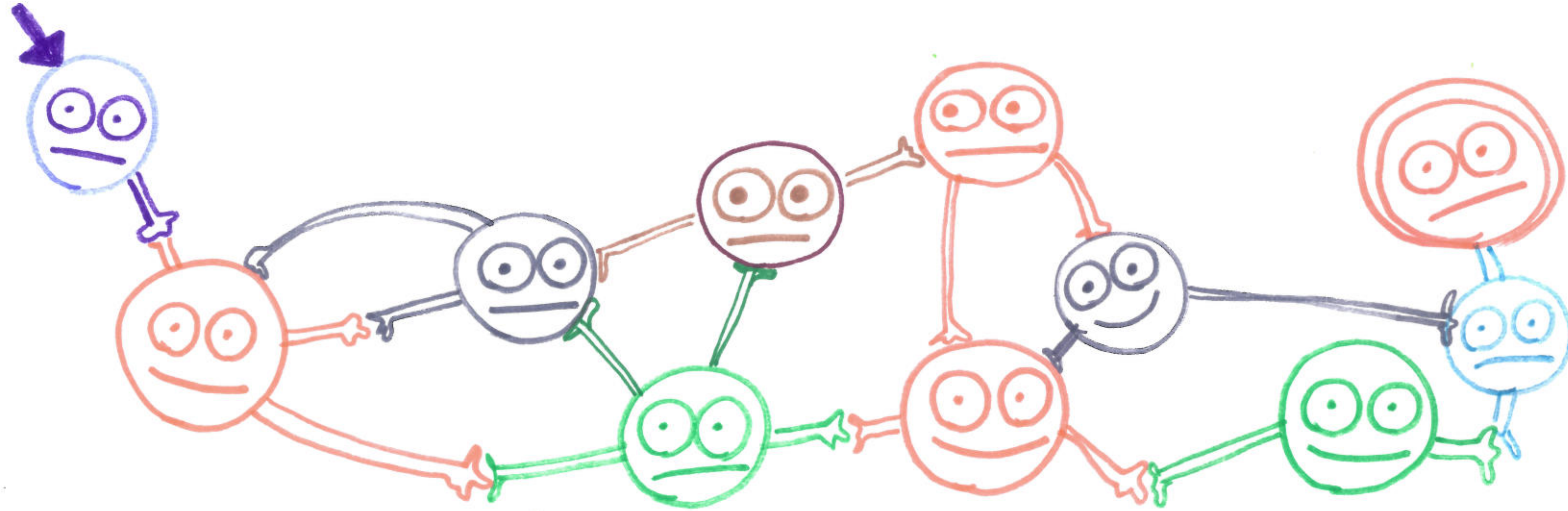
- **ESOP'16:** On the Relative Expressiveness of Higher-Order Session Processes. Dimitrios Kouzapas , Jorge A. Pérez , Nobuko Yoshida
- **CONCUR'15**: Characteristic Bisimulations for Higher-Order Session Processes. Dimitrios Kouzapas , Jorge A. Pérez , Nobuko Yoshida

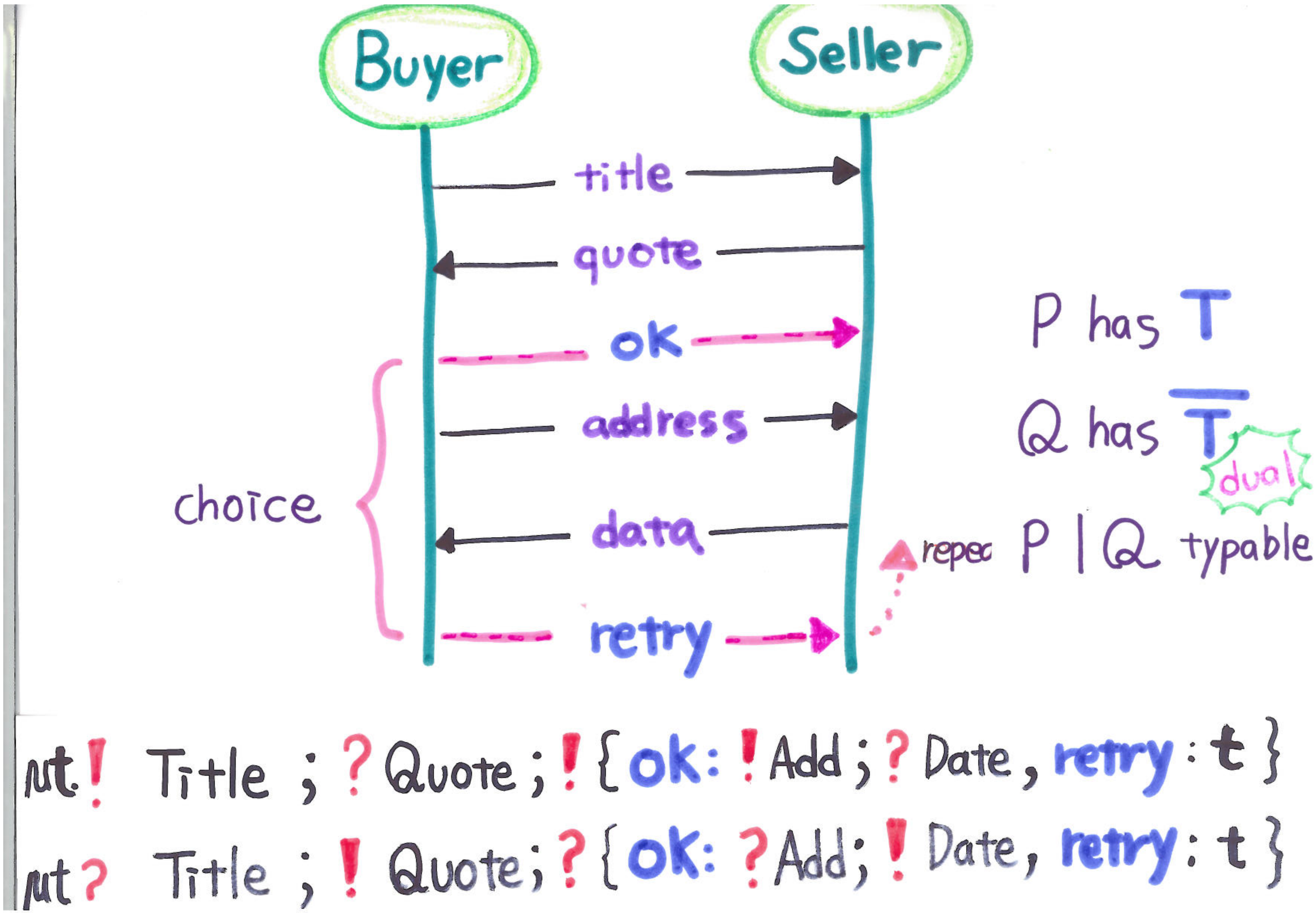
Multiparty Compatibility in Communicating Automata

Synthesis and Characterisation of Multiparty Session Types

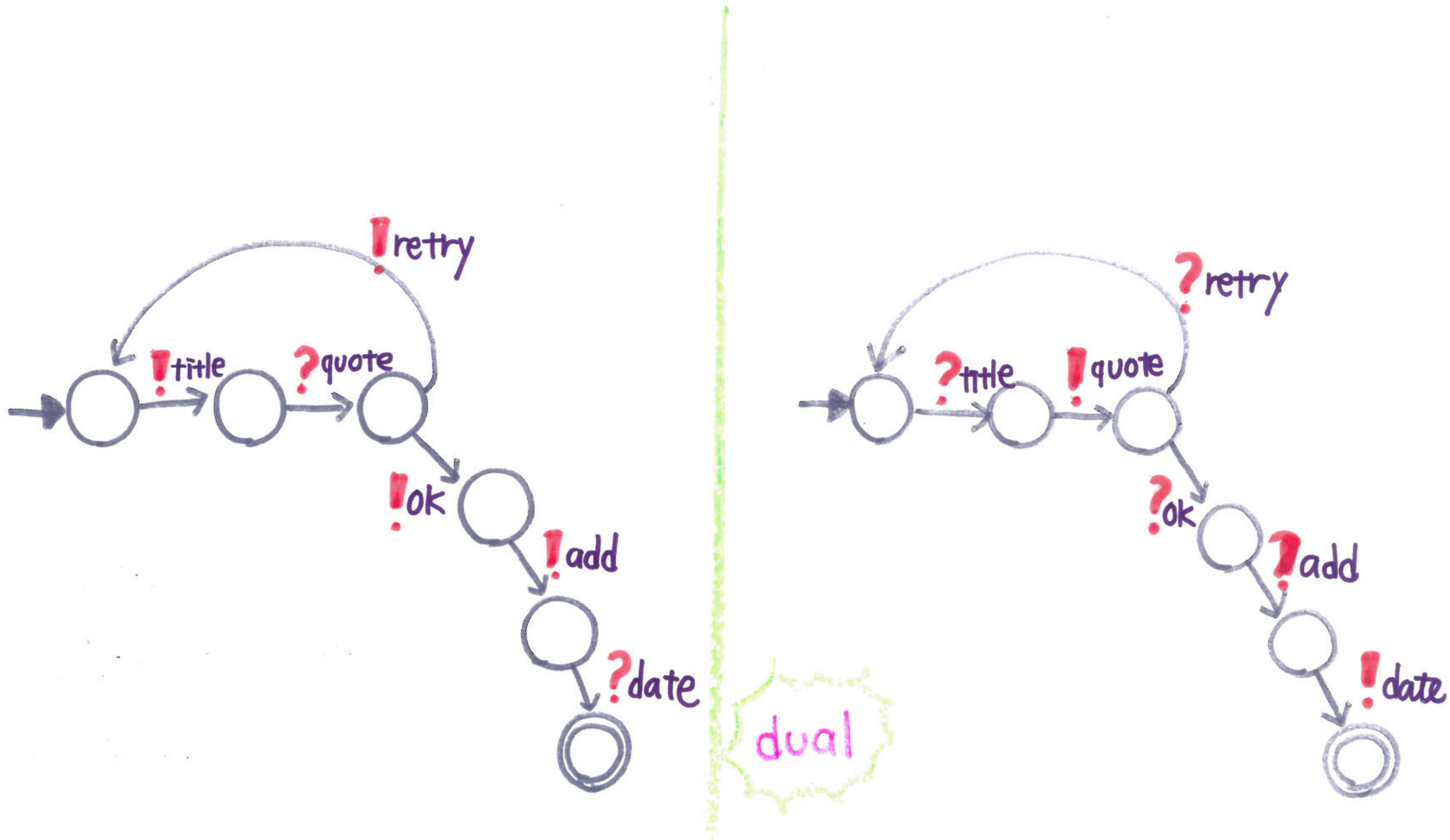
ICALP'13

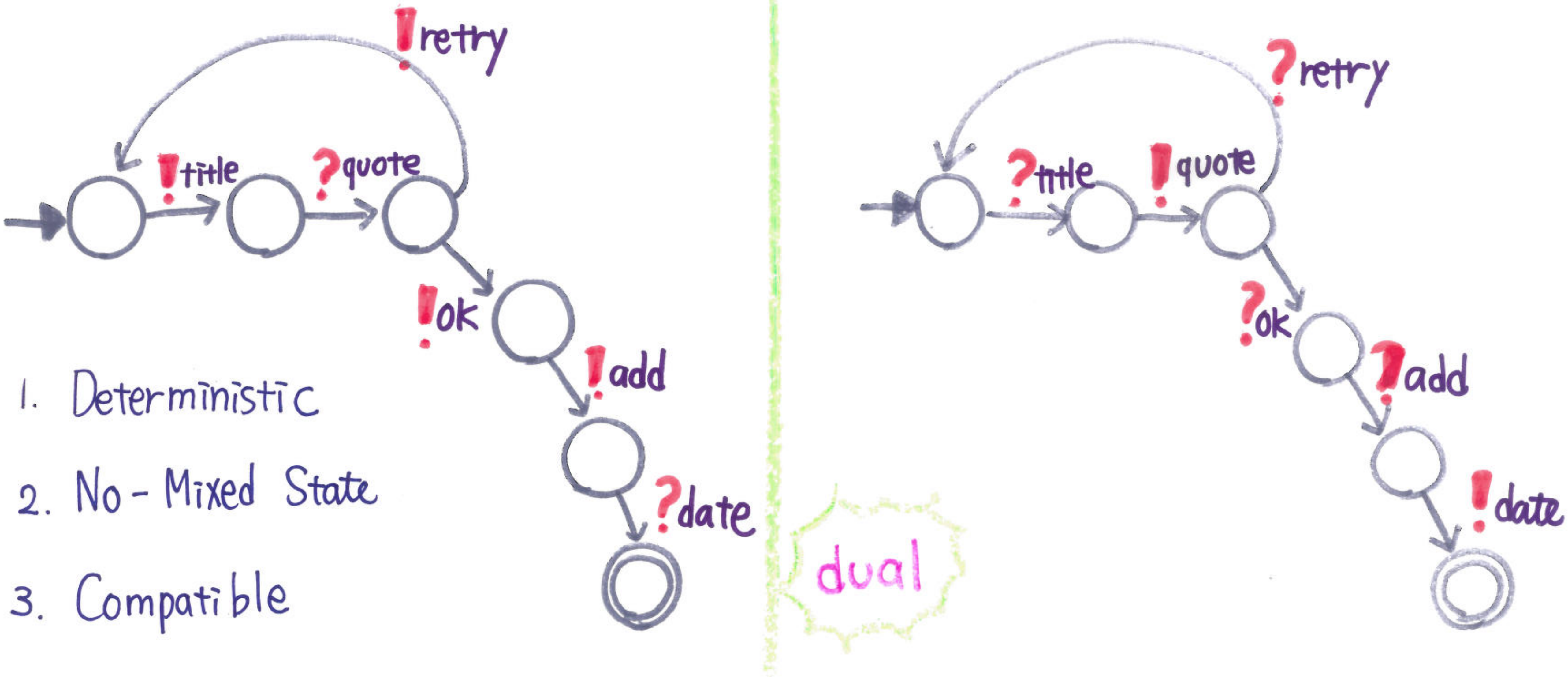
Nobuko Yoshida   Pierre-Malo Deniélou




Buyer
Seller
title
quote
ok
address
data
retry
choice
P has T
Q has T̄
dual
P | Q typable
μt.! Title ; ? Quote ; ! {ok: ! Add ; ? Date , retry : t }
μt ? Title ; ! Quote ; ? {ok: ? Add ; ! Date , retry : t }

Communicating Automata [1980s]

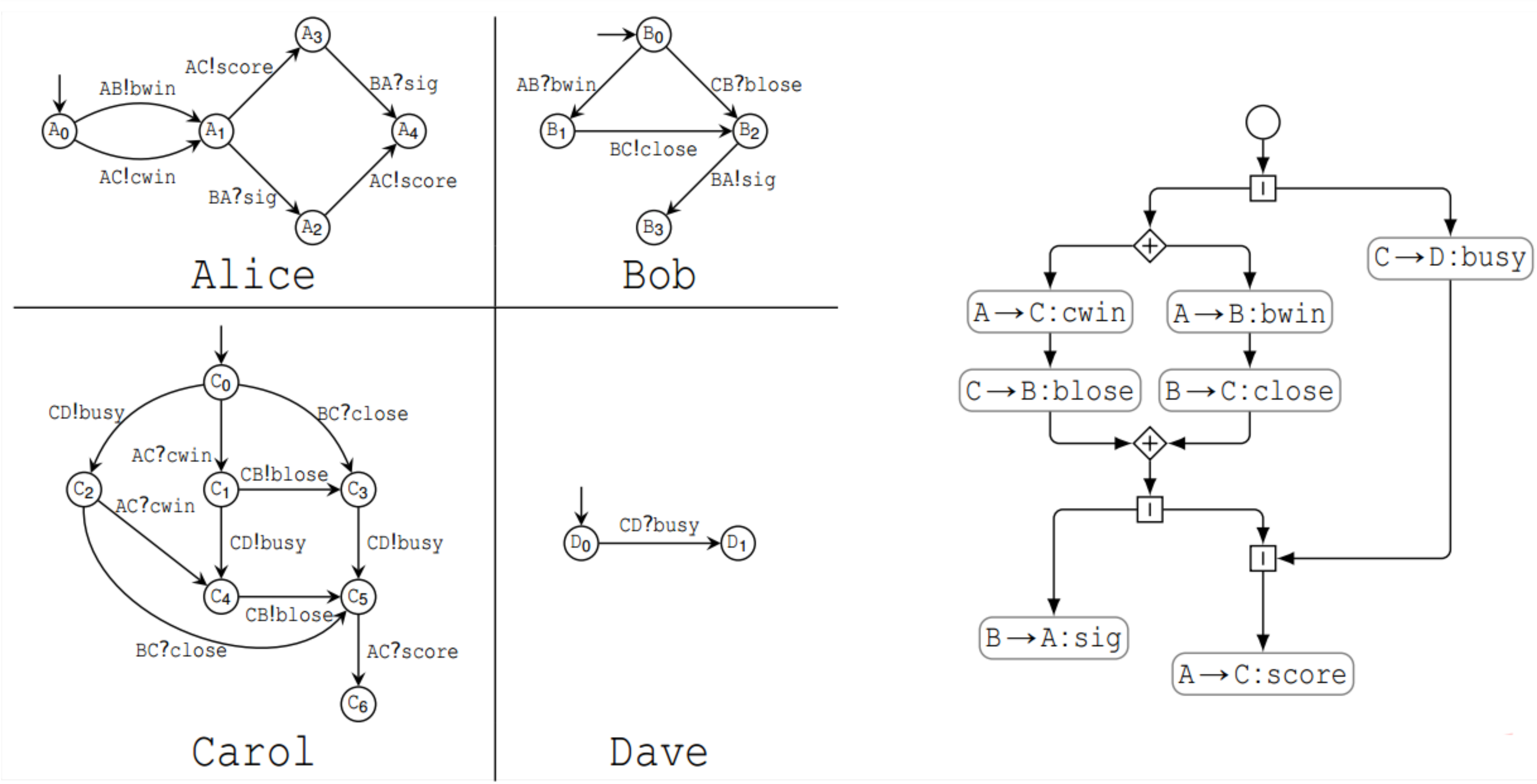




1. Deterministic
2. No-Mixed State
3. Compatible

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

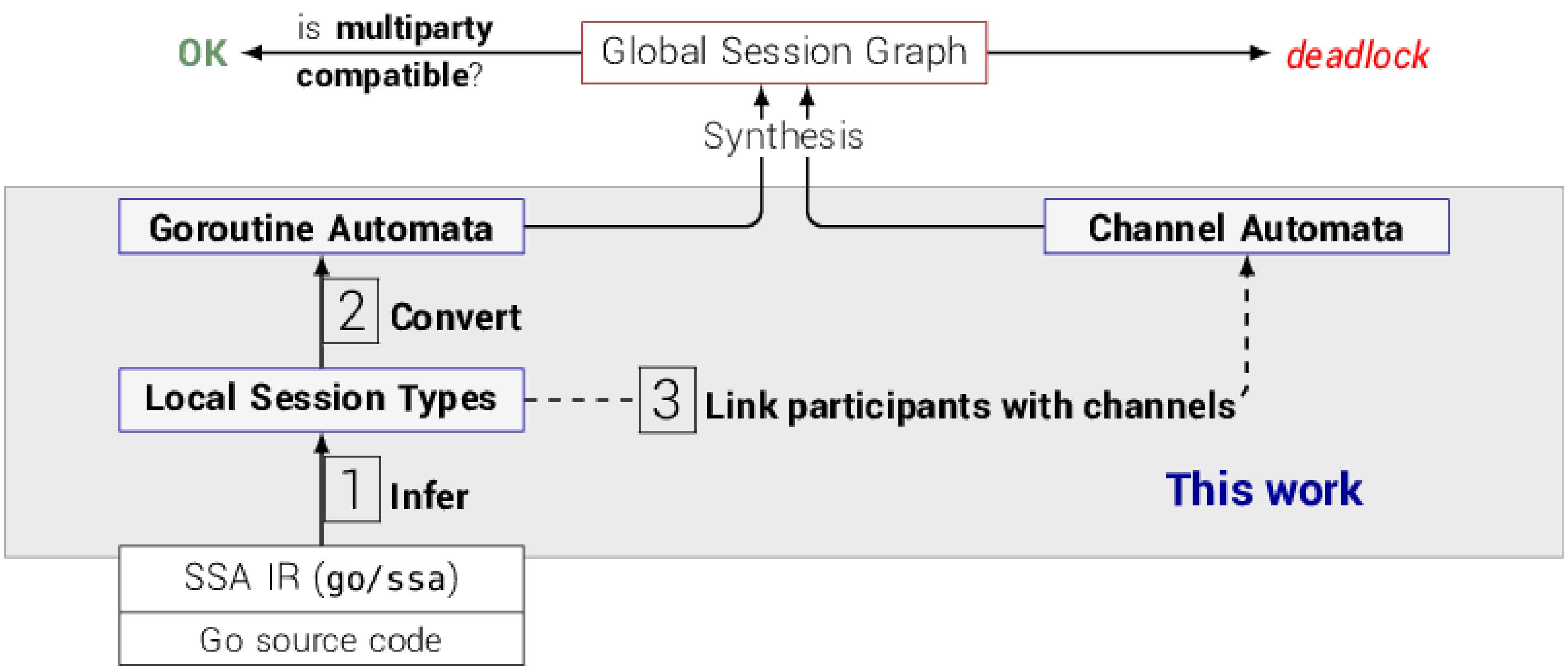
From Communicating Machines to Graphical Choreographies [POPL'15, CONCUR'15]



[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

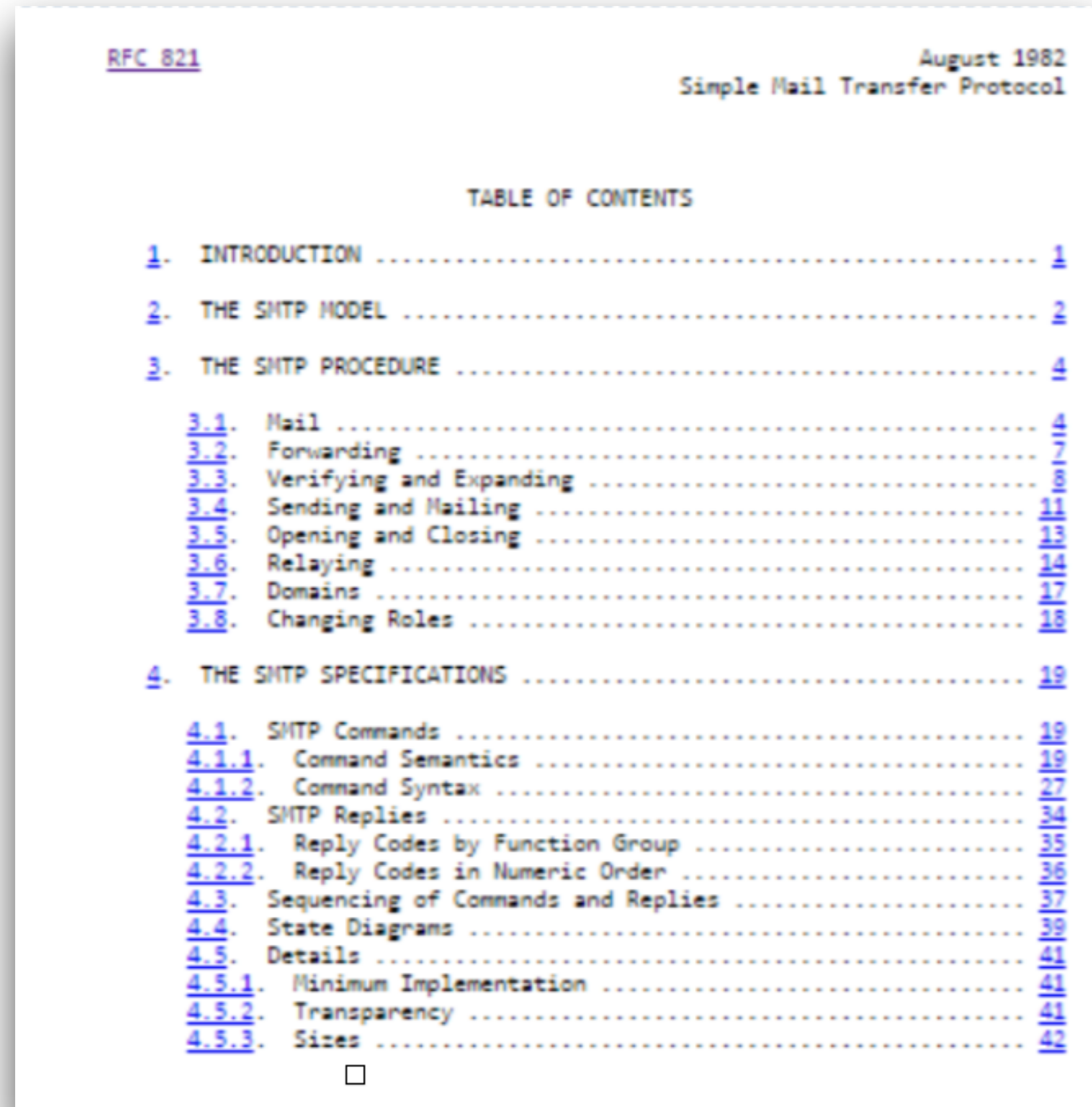


Go and Concurrency

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

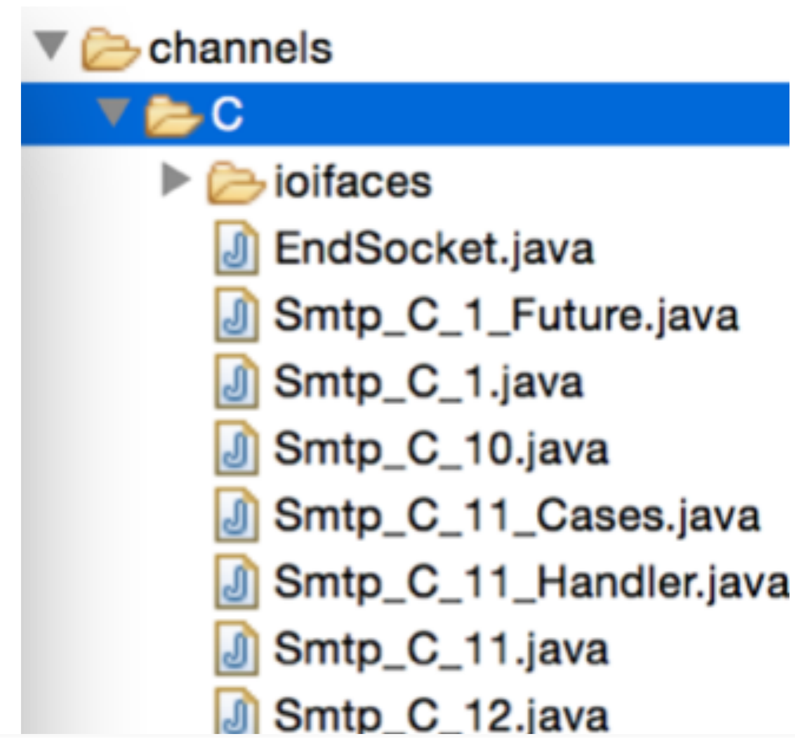
" *Do not communicate by sharing memory*; *instead*, *share memory by communicating*. "
-- Effective Go (developer guide)

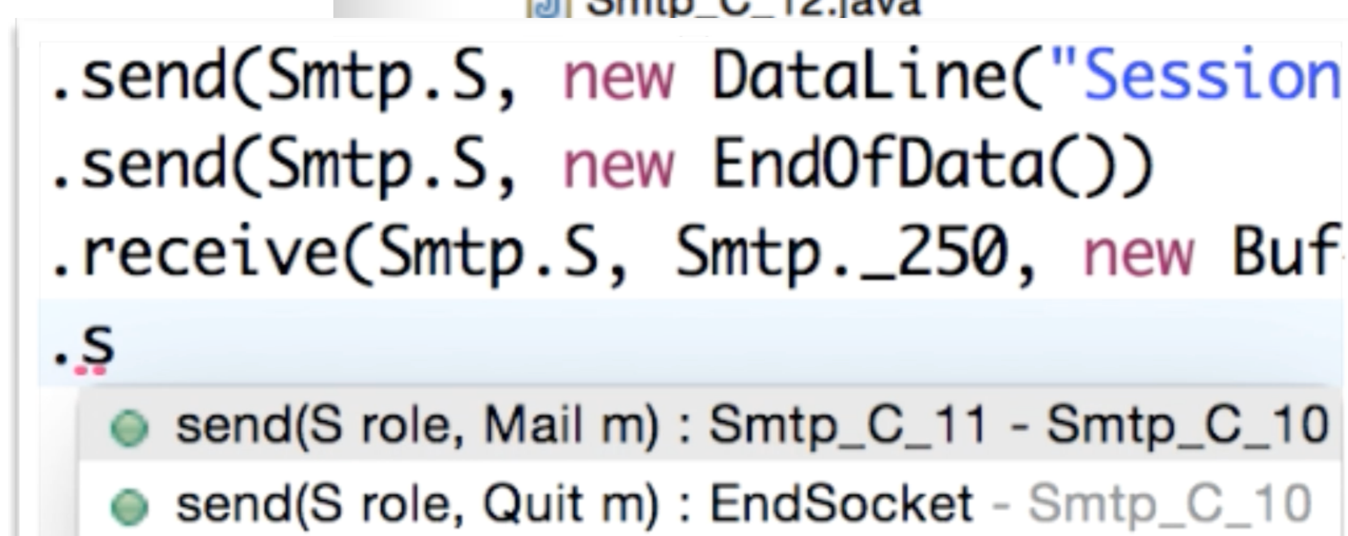
Java API Generation [FASE'16]



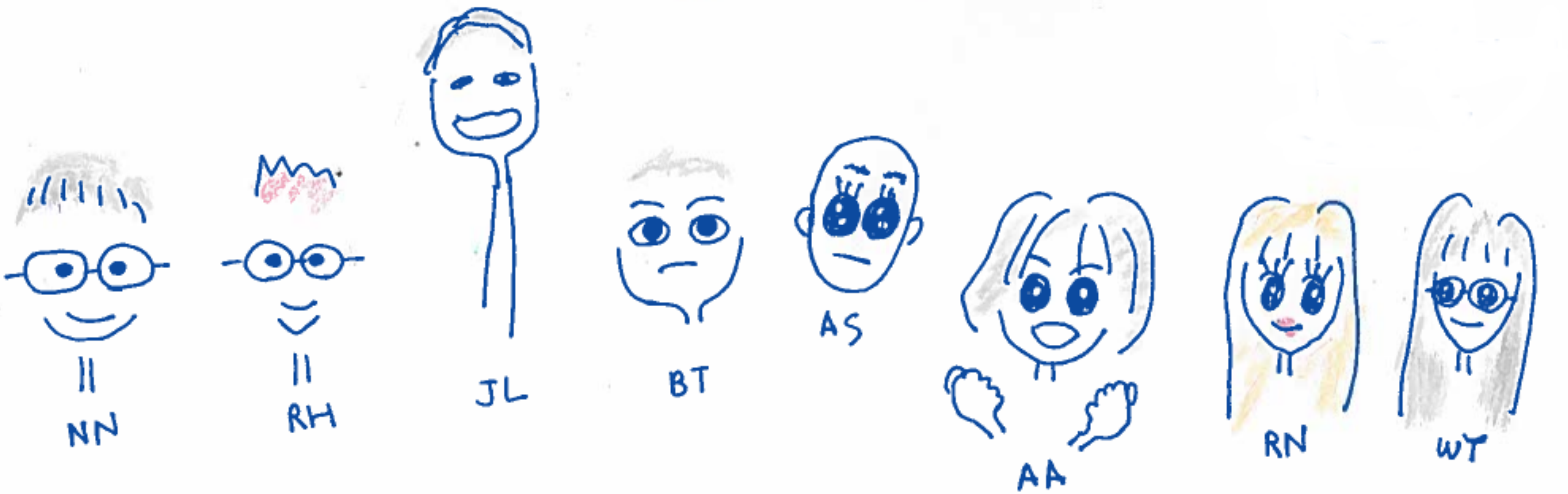
RFC 821 August 1982
Simple Mail Transfer Protocol

TABLE OF CONTENTS





Session Type Mobility Group



www.mrg.doc.ic.ac.uk