Session TYPES

AS A

Descriptive π Tool

for Distributed PROTOCOL

Nobuko Yoshida & Raymond Hu

25th Oct 2017

Awards

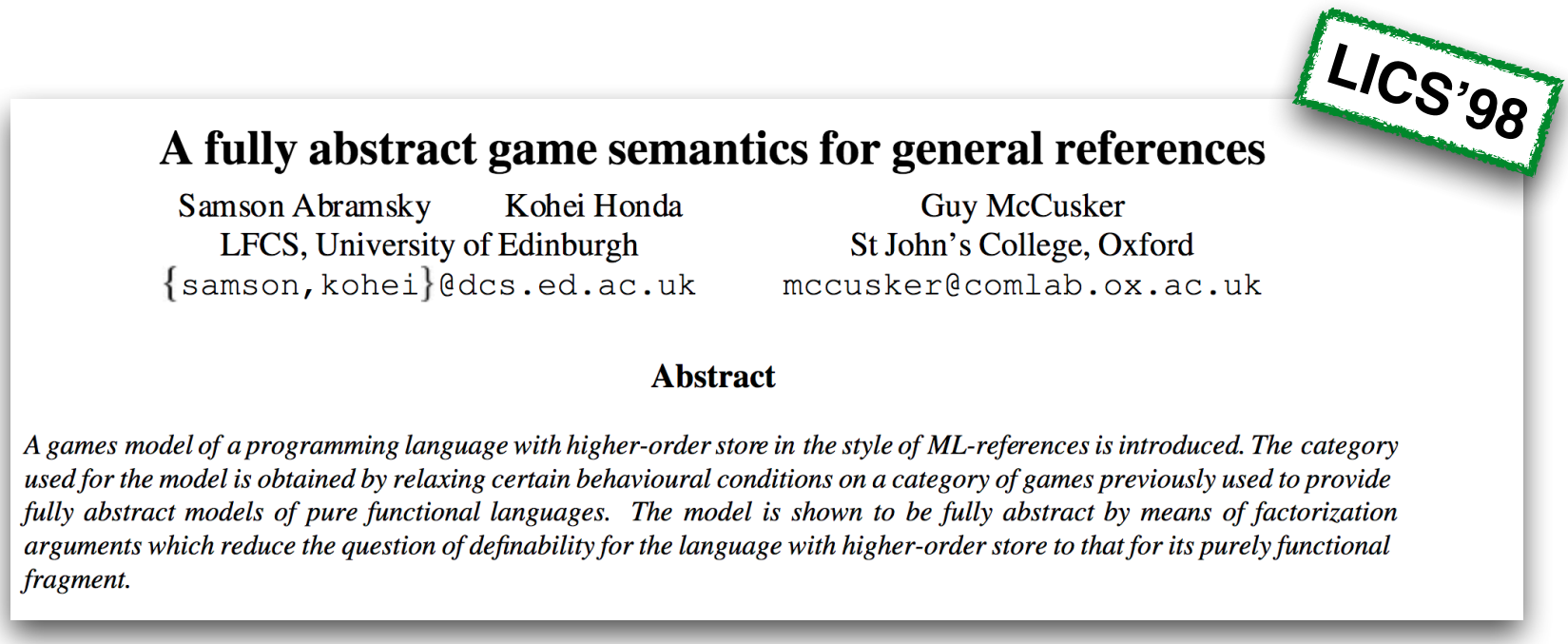

ETAPS 2019
TEST-OF-TIME
AWARD
Language primitives and type discipline for structured communication-based programming' by Kohei Honda, Vasco T. Vasconcelos and Makoto Kubo

Awards

POPL 2008 MOST INFLUENTIAL PAPER AWARD

Awards


LICS 2018 TEST OF TIME AWARD

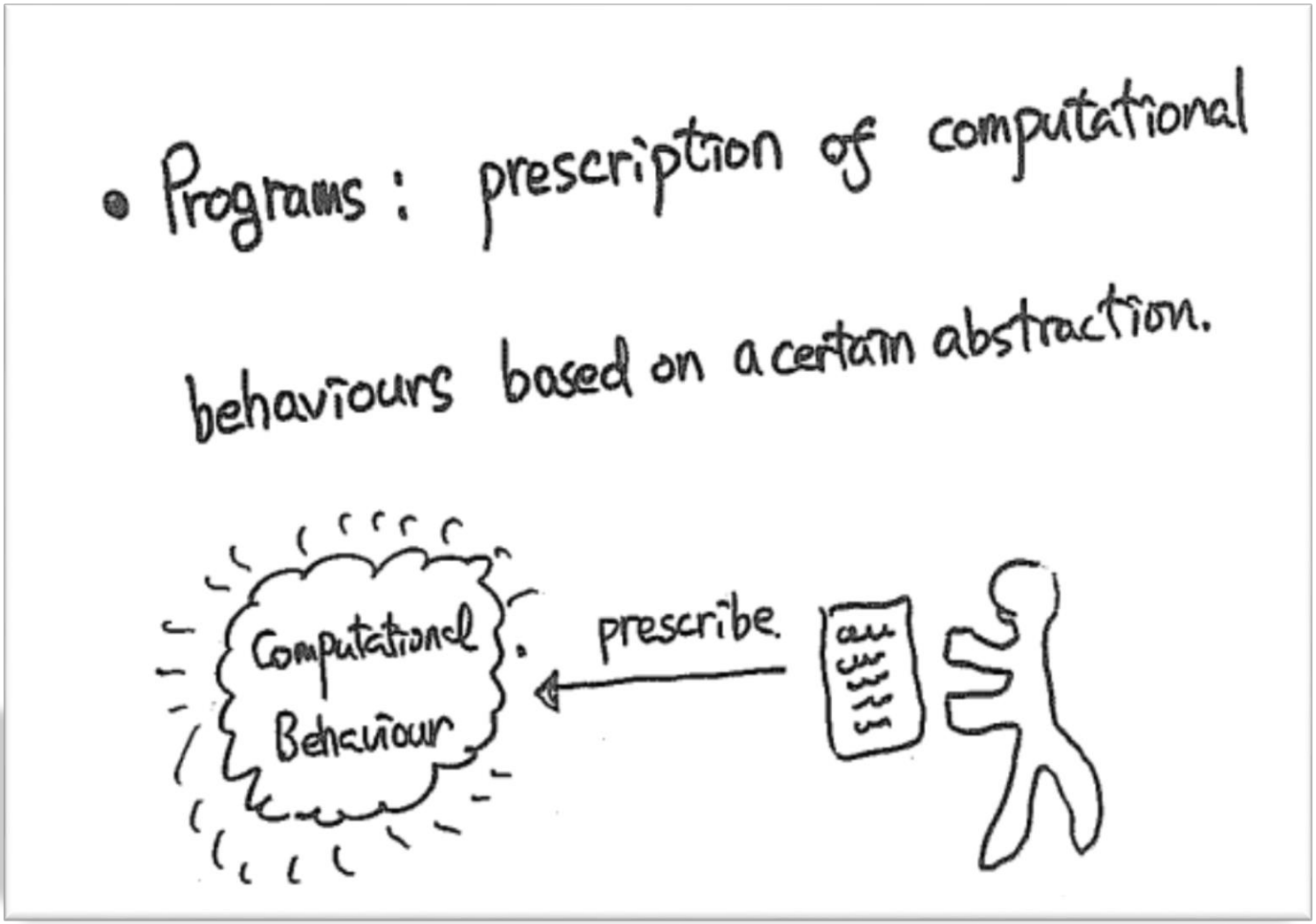

LICS'98
A fully abstract game semantics for general references
Samson Abramsky
Kohei Honda
LFCS, University of Edinburgh
{samson,kohei}@dcs.ed.ac.uk
Guy McCusker
St John's College, Oxford
mccusker@comlab.ox.ac.uk
Abstract
A games model of a programming language with higher-order store in the style of ML-references is introduced. The category used for the model is obtained by relaxing certain behavioural conditions on a category of games previously used to provide fully abstract models of pure functional languages. The model is shown to be fully abstract by means of factorization arguments which reduce the question of definability for the language with higher-order store to that for its purely functional fragment.

Idioms for Interaction

—Invitation to Hacking on π-calculus—

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:

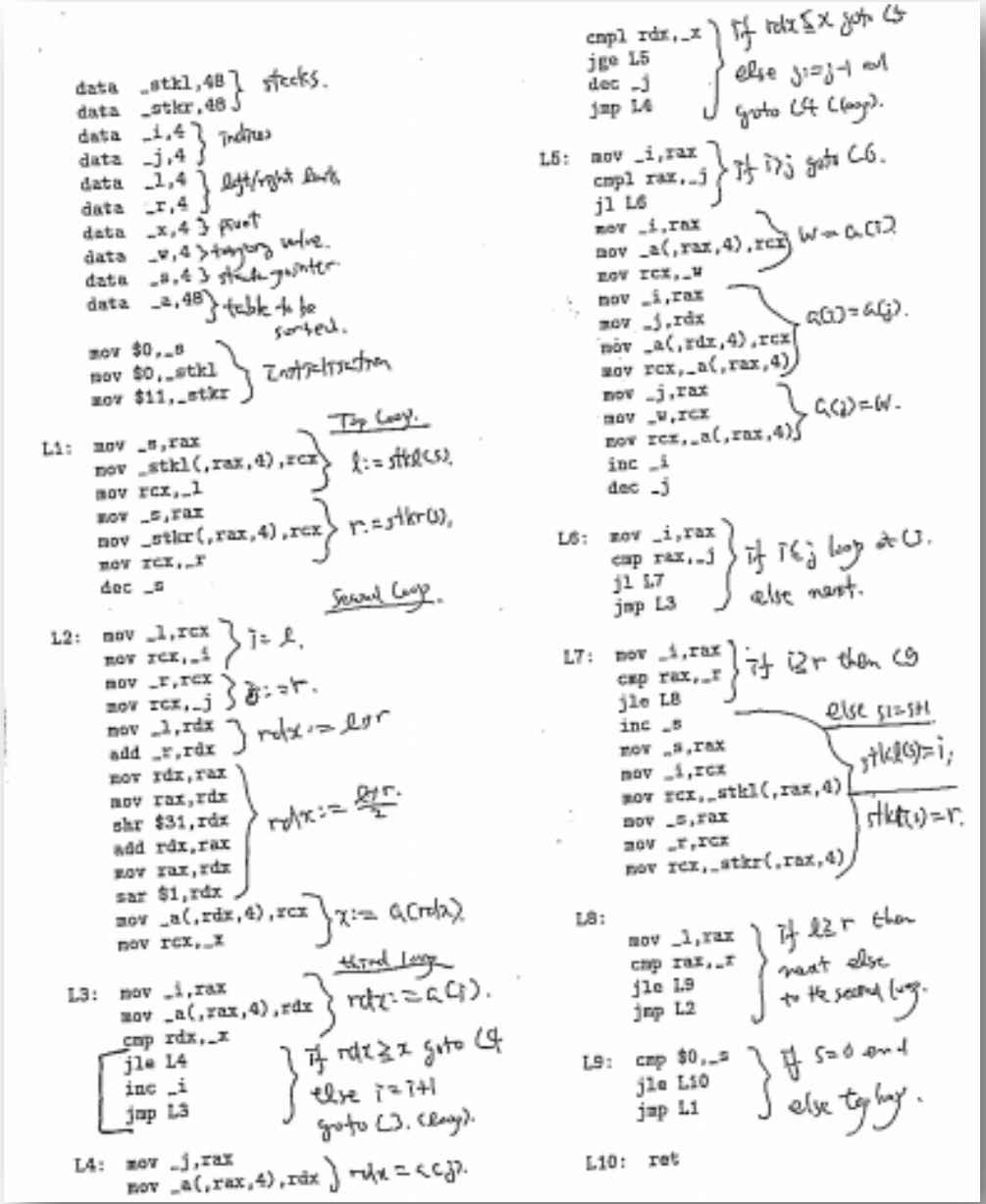
  Imperative: sequential composition, if-then-else,

  while, procedures, modules, ...

  Functional: application, product, union,

  recursion, modules, ...

UNSTRUCTURED:



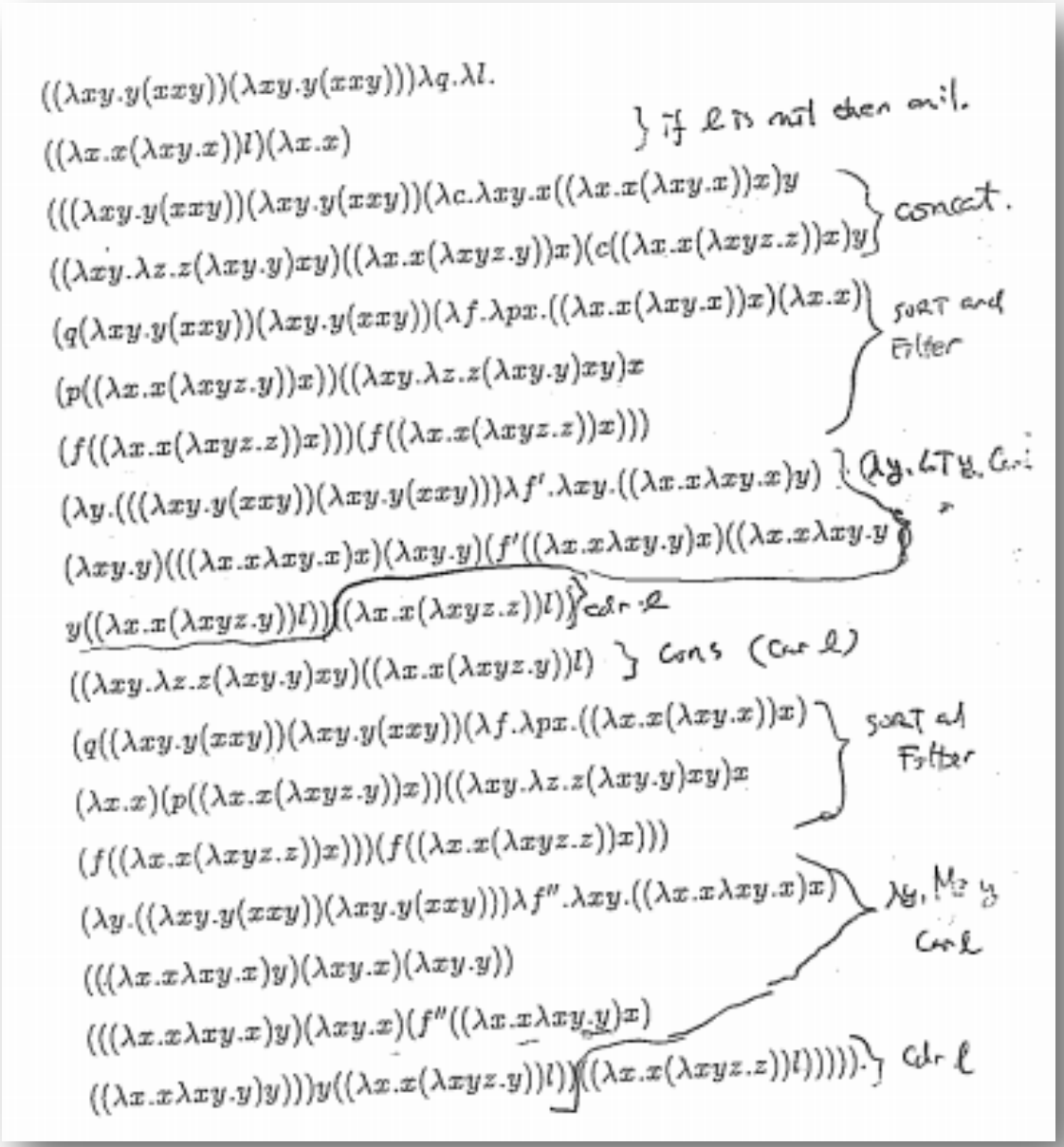
STRUCTURED:

```
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.     } Partition into 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:



Quicksort with combinators:

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

$I = \lambda x.x \quad T = \lambda xy.x \quad F = \lambda xy.y \quad Y = (\lambda xy.y(xxy))(\lambda xy.y(xxy))$

$\text{Cons} = \lambda xy.\lambda z.zFxy \quad \text{Isnil} = \lambda x.xT$

$\text{Car} = \lambda x.x(\lambda xyz.y) \quad \text{Cdr} = \lambda x.x(\lambda xyz.z)$

$\text{Concat} = Y(\lambda c.\lambda xy.x\ (\text{Isnil } x)\ y\ (\text{Cons } (\text{Car } x)\ (c\ (\text{Cdr } x)\ y)))$

$\text{Filter} = Y(\lambda f.\lambda px.(\text{Isnil } x)\ I\ (p\ (\text{Car } x))\ (\text{Cons } x\ (\text{Filter Cdr } x))\ (\text{Filter } (\text{Cdr } x)))$

$\text{Iszero} = \lambda x.xT \quad \text{Pred} = \lambda x.xF$

$LT = Y(\lambda f.\lambda xy.(\text{Iszero } y)\ F\ ((\text{Iszero } x)\ T\ (f\ (\text{Pred } x)\ (\text{Pred } y))))$

$ME = Y(\lambda f.\lambda xy.(\text{Iszero } x)\ ((\text{Iszero } y)\ I\ F)\ ((\text{Iszero } y)\ (T)\ (f\ (\text{Pred } x)\ (\text{Pred } y))))$

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

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

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

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

λ in π

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

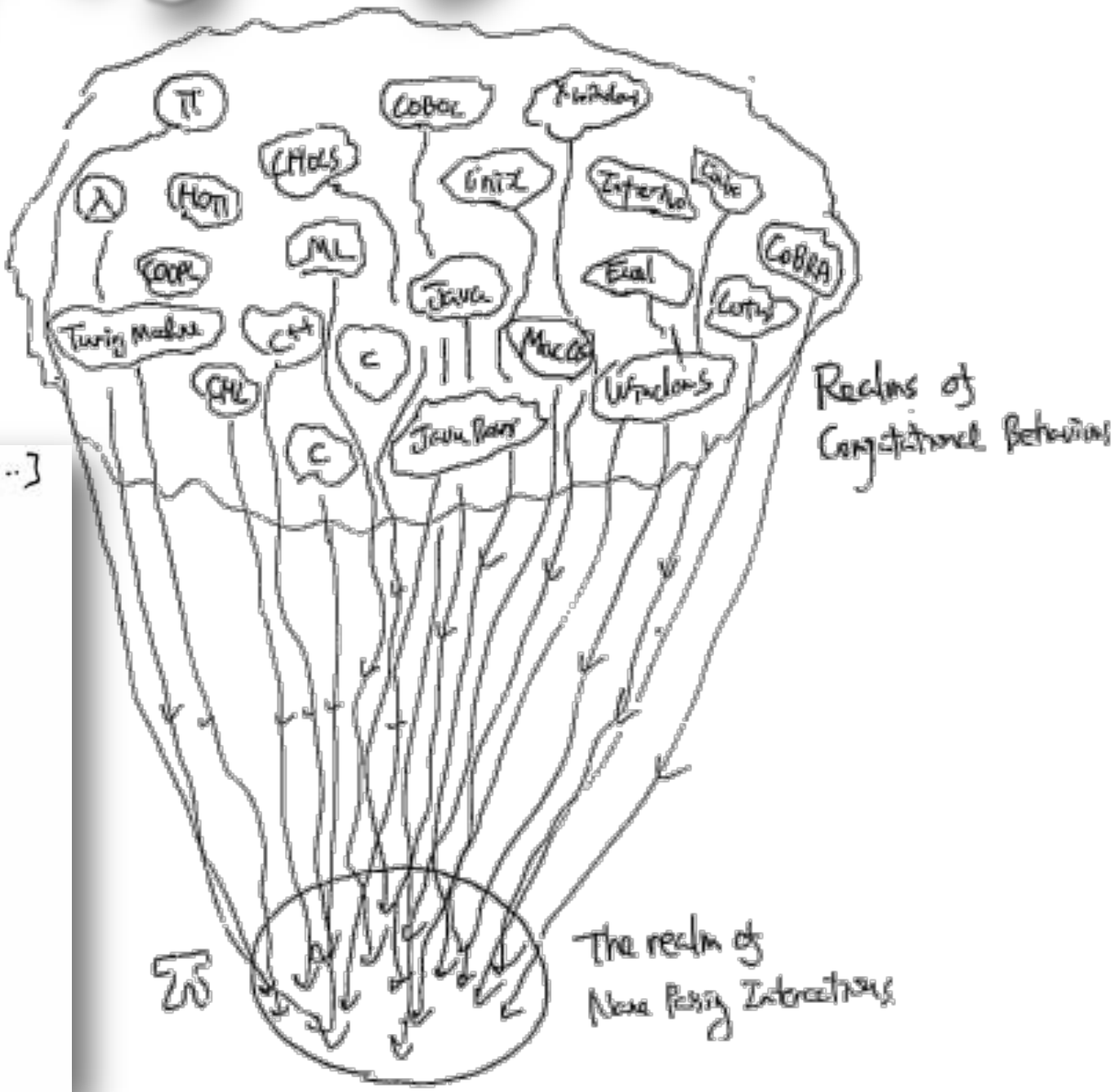
$[\![\lambda x.M]\!]_u \stackrel{\text{def}}{=} u(x u').[\![M]\!]_{u'}.$

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

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 [HYS4]
- strategies on Games [HO95]

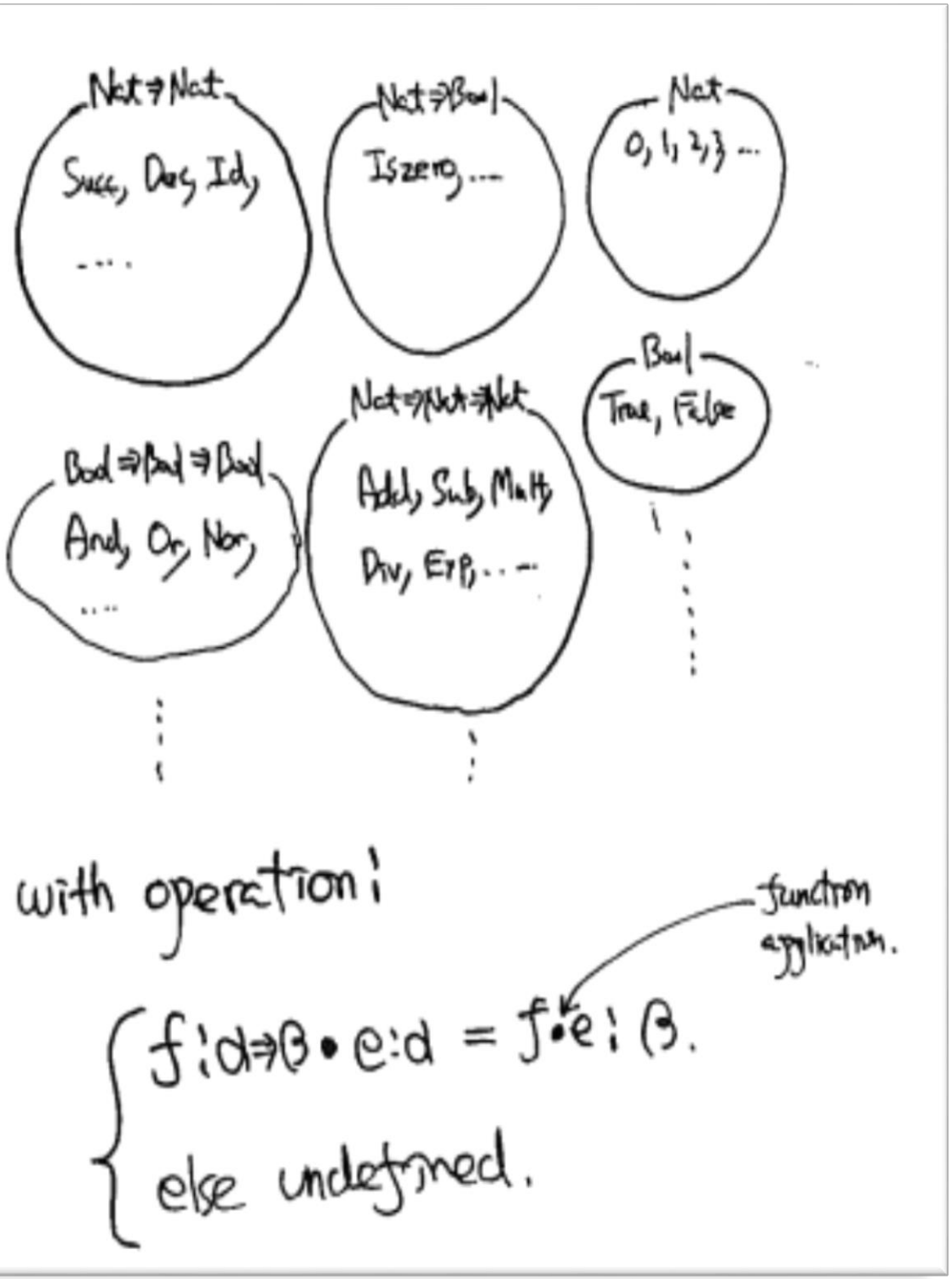


The Role of Types in π-calculus

- (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?

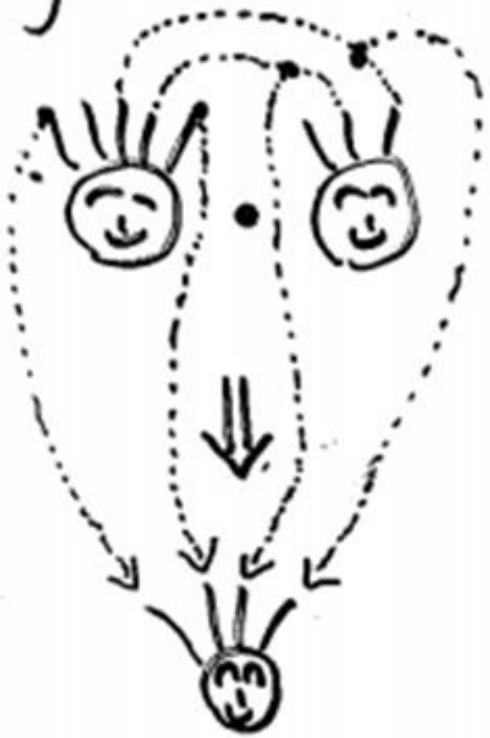
- (safety) Is this program/system in the safe (or correct, relevant, ...) classes of behaviours? Can the safety be preserved compositionally?

Functional Types



Process Types

- When it comes to processes, composition becomes:



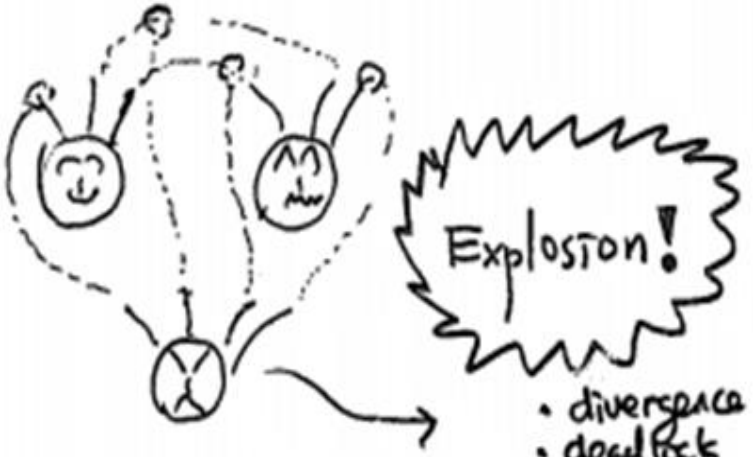
cf.

3 + 5

8

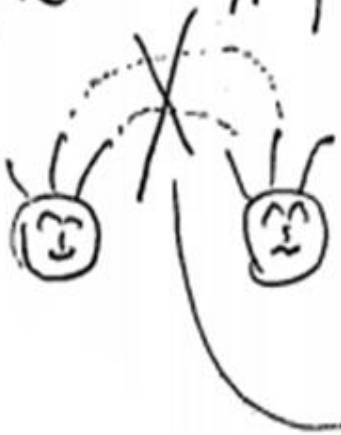
$P \bullet_\delta \delta \longmapsto_\tau \delta$

- But some composition is dangerous!



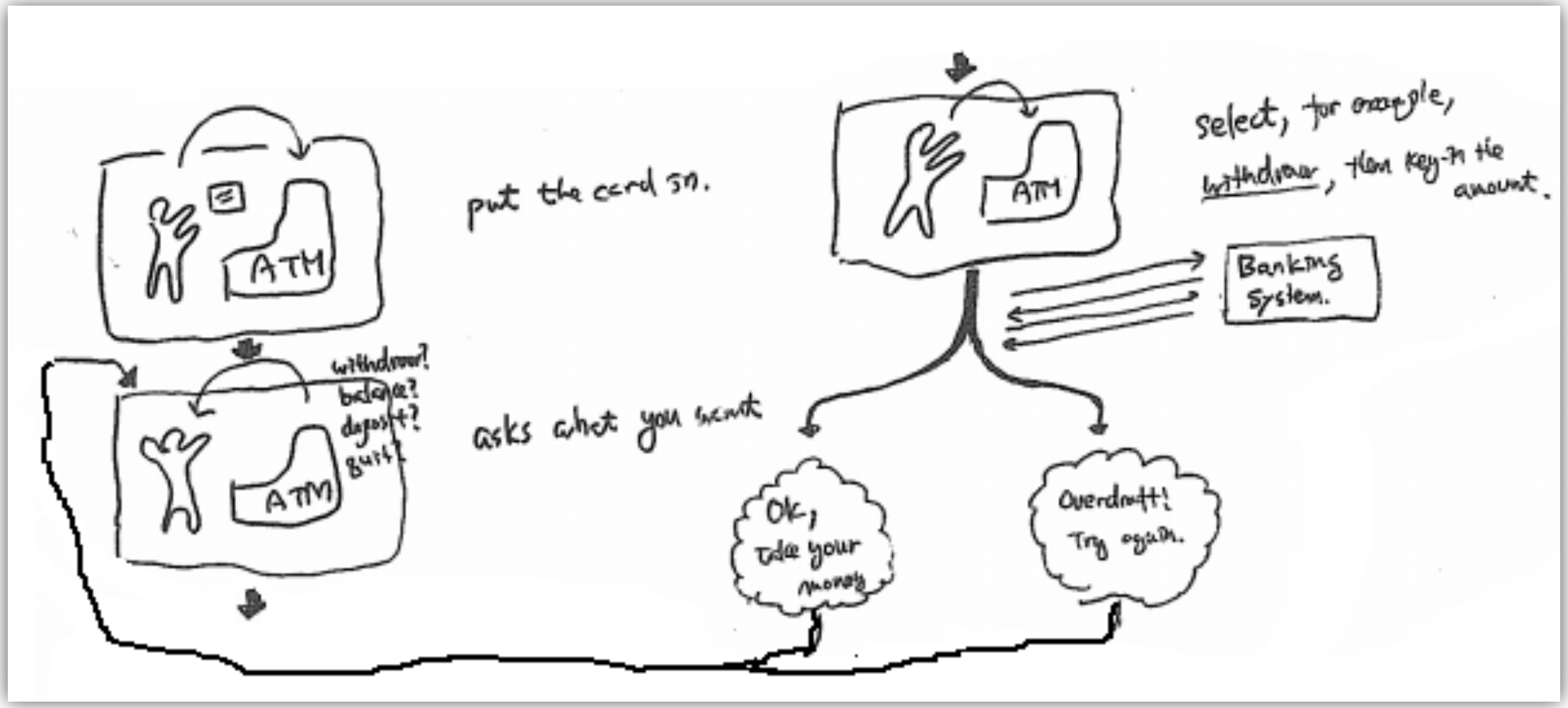
- divergence
- deadlock
- run-time error
- ⋮

- Therefore we type processes,



The connection is prohibited

Implementing ATM

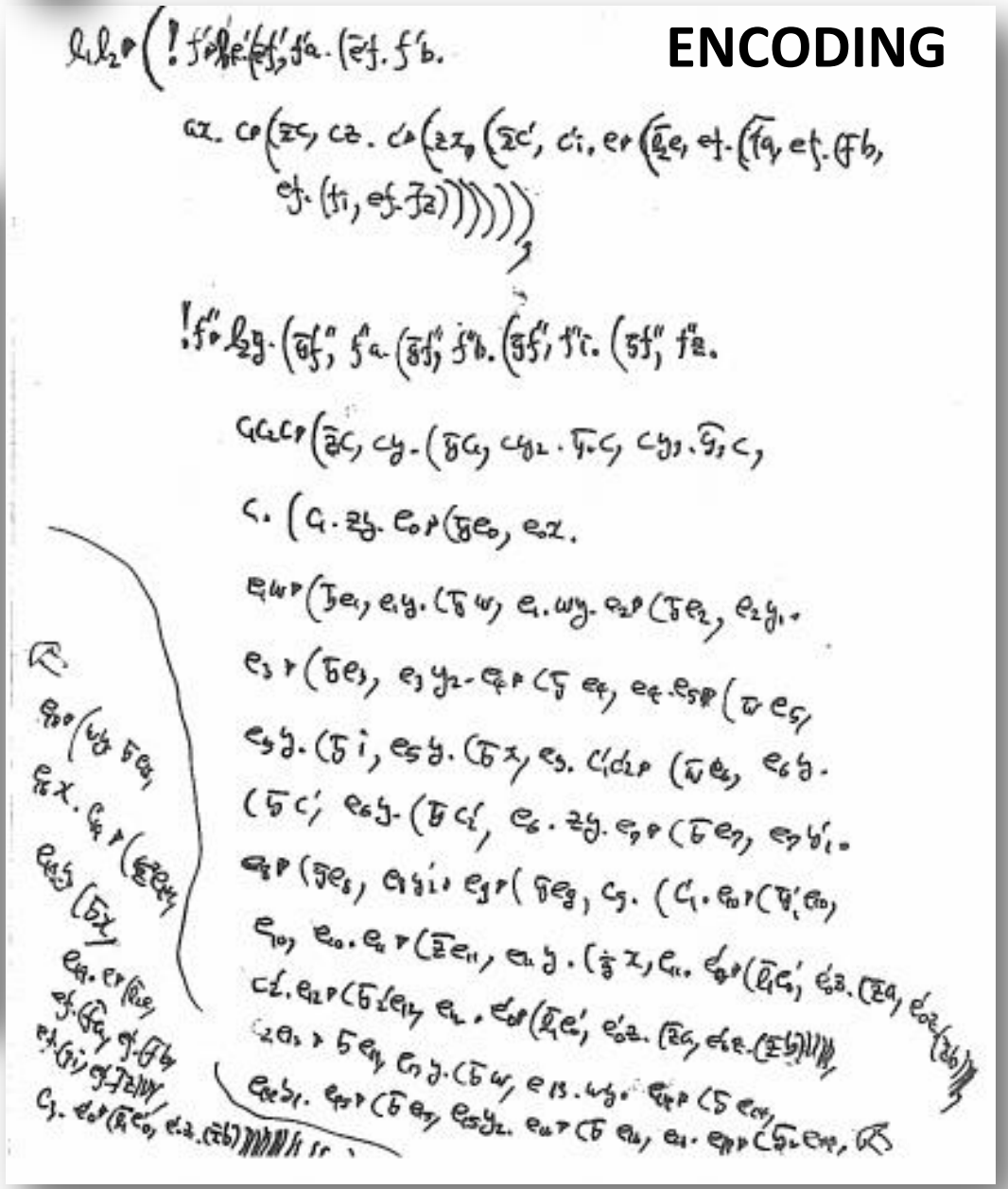


Implementing ATM

$ATM(cb) \stackrel{def}{=}$

```
ATM<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.
        ?bd;                                      * ↔user
          b<w>:: w!bd; ?X;                        * ↔bank
            z!X; ATM<cb>]
```

ENCODING



Dialogue between Industry and Academia

Binary Session Types [PARL'94, ESOP'98]

$\Downarrow$

Milner, Honda and Yoshida joined W3C WS-CDL (2002)

$\Downarrow$

Formalisation of W3C WS-CDL [ESOP'07]

$\Downarrow$

Scribble at $\pi 4$ Technology

CDL Equivalent

- Basic example:

```
package HelloWorld {
    roleType YouRole, WorldRole;
    participantType You{YouRole}, World{WorldRole};
    relationshipType YouWorldRel between YouRole and WorldRole;
    channelType WorldChannelType with roleType WorldRole;

    choreography Main {
        WorldChannelType worldChannel;

        interaction operation=hello from=YouRole to=WorldRole
                relationship=YouWorldRel channel=worldChannel {
            request messageType=Hello;
        }
    }
}
```

Scribble Protocol

- *"Scribbling is necessary for architects, either physical or computing, since all great ideas of architectural construction come from that unconscious moment, when you do not realise what it is, when there is no concrete shape, only a whisper which is not a whisper, an image which is not an image, somehow it starts to urge you in your mind, in so small a voice but how persistent it is, at that point you start scribbling" - Kohei Honda 2007*

- Basic example:

```
protocol HelloWorld {
    role You, World;
    Hello from You to World;
}
```

Dialogue between Industry and Academia

Binary Session Types [PARL'94, ESOP'98]

⇓

Milner, Honda and Yoshida joined W3C WS-CDL (2002)

⇓

Formalisation of W3C WS-CDL [ESOP'07]

⇓

Scribble at $\pi 4$ Technology

⇓

Multiparty Session Types [POPL'08]

⇓

Dialogue between Industry and Academia

Binary Session Types [PARL'94, ESOP'98]

⇓

Milner, Honda and Yoshida joined W3C WS-CDL (2002)

⇓

Formalisation of W3C WS-CDL [ESOP'07]

⇓

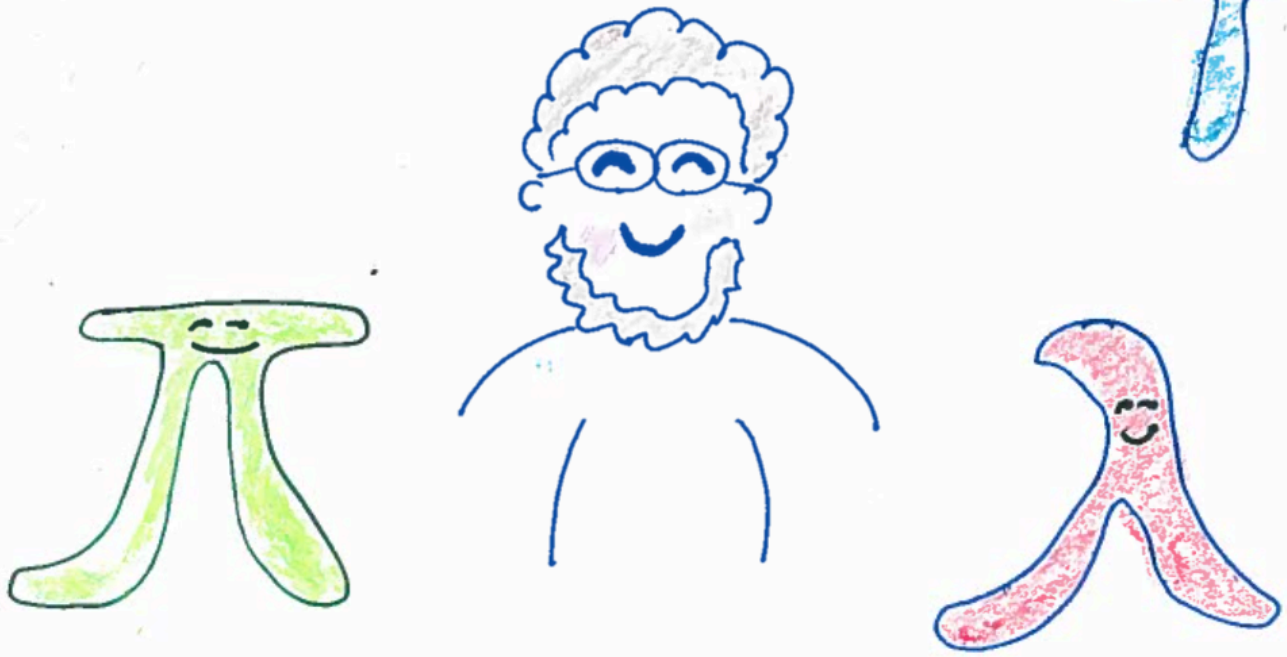
Scribble at $\pi 4$ Technology

⇓

Multiparty Session Types [POPL'08]

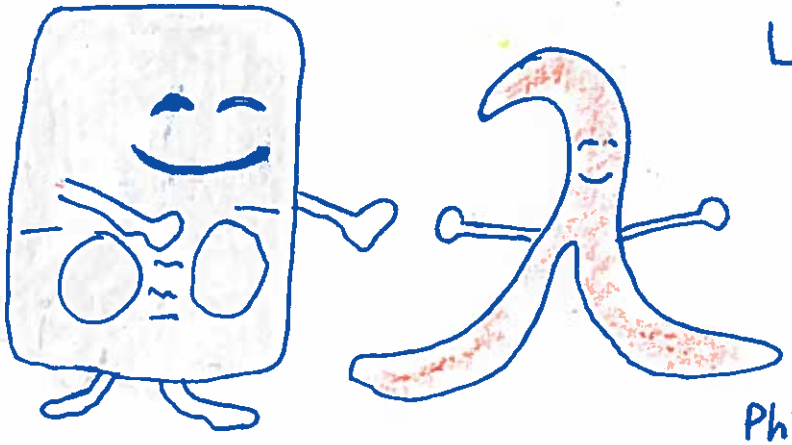
⇓

PHIL & SESSION TYPES



Imperial College London

199?

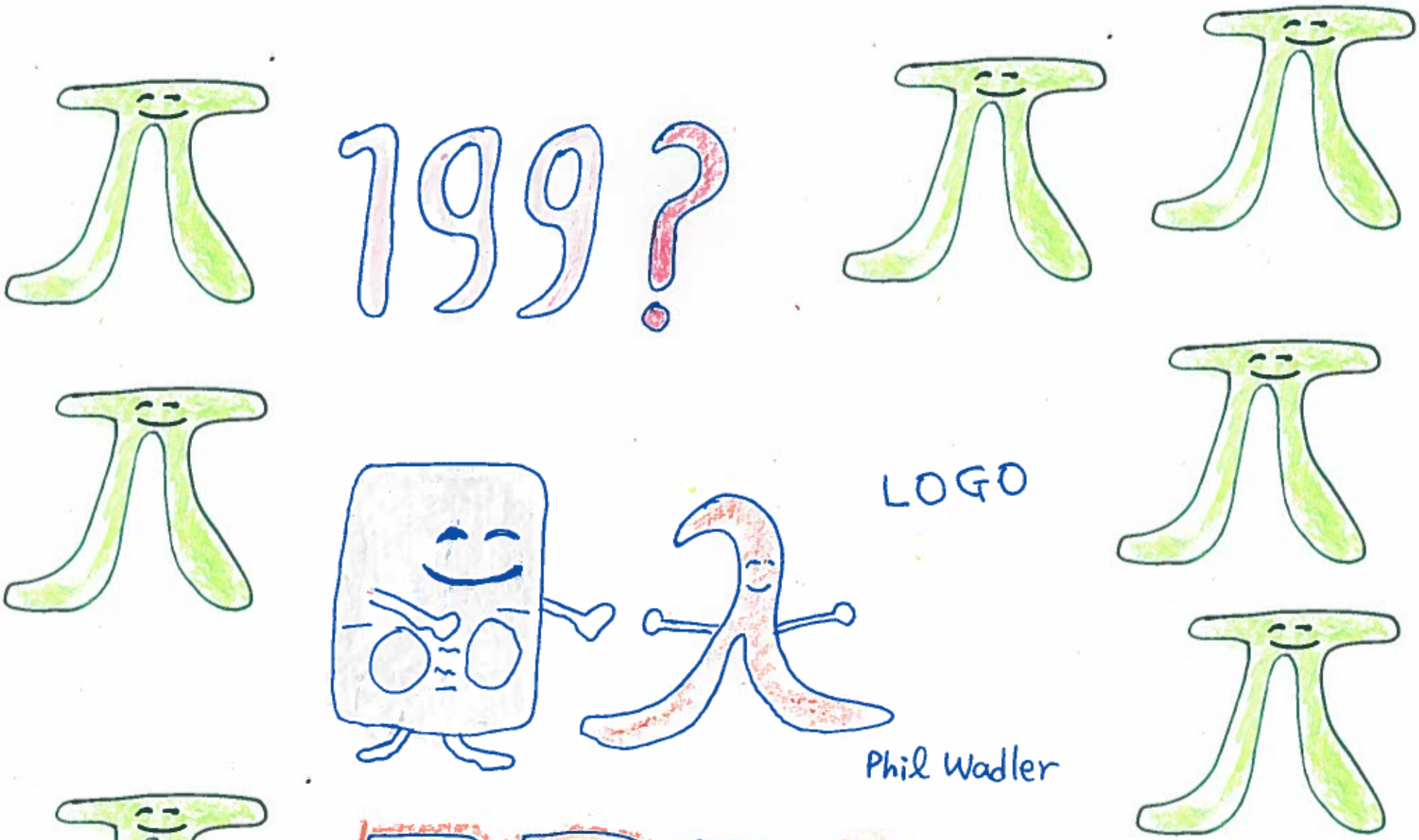


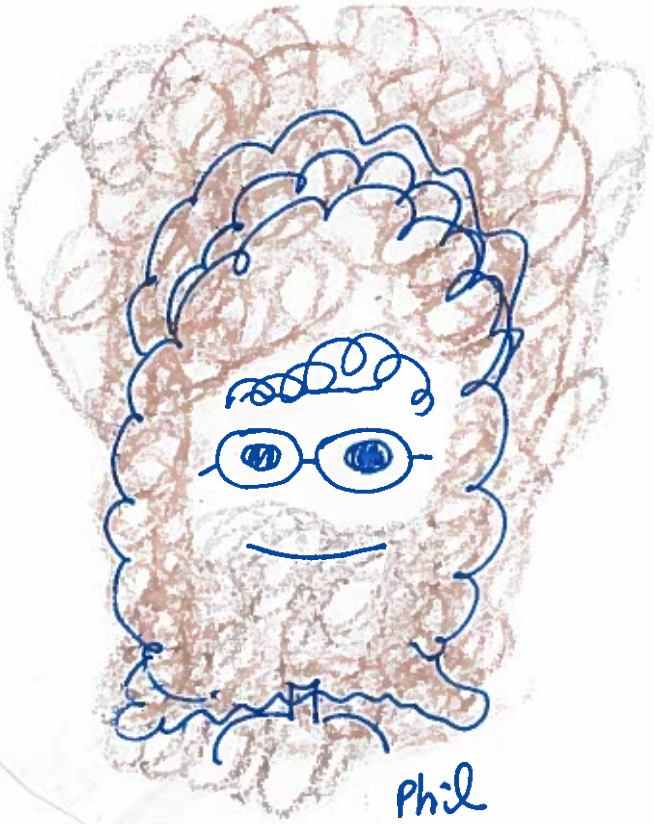
LOGO

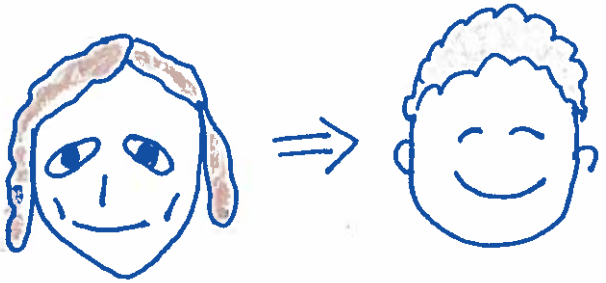
Phil Wadler

FPCA

Functional Programming Languages and Computer Architectures


199?
LOGO
Phil Wadler
FPCA
Functional
Programming
Languages and
computer
Architectures

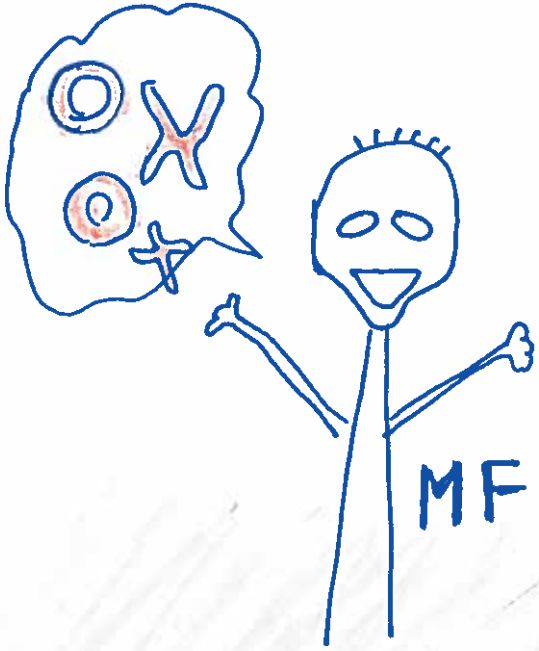


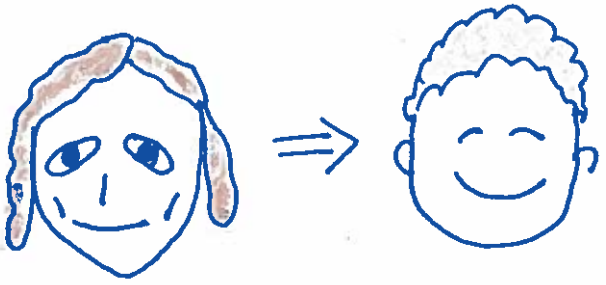


Jean-Jacque Levy

Andy Gordon

Luc Maranget

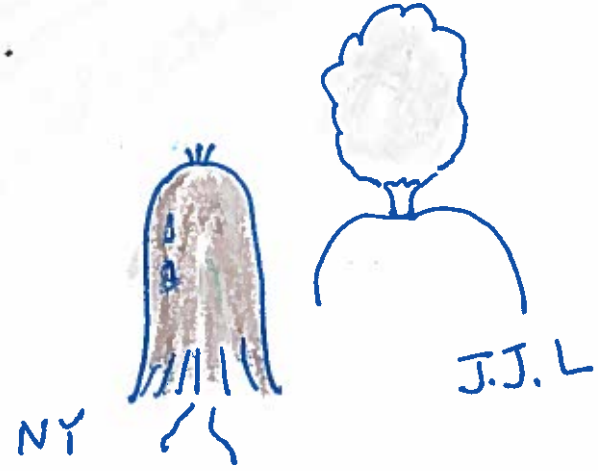




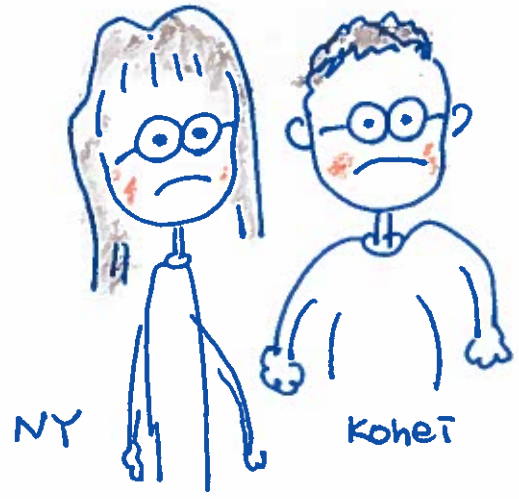
Jean-Jacque Levy

Andy Gordon

Luc Maranget



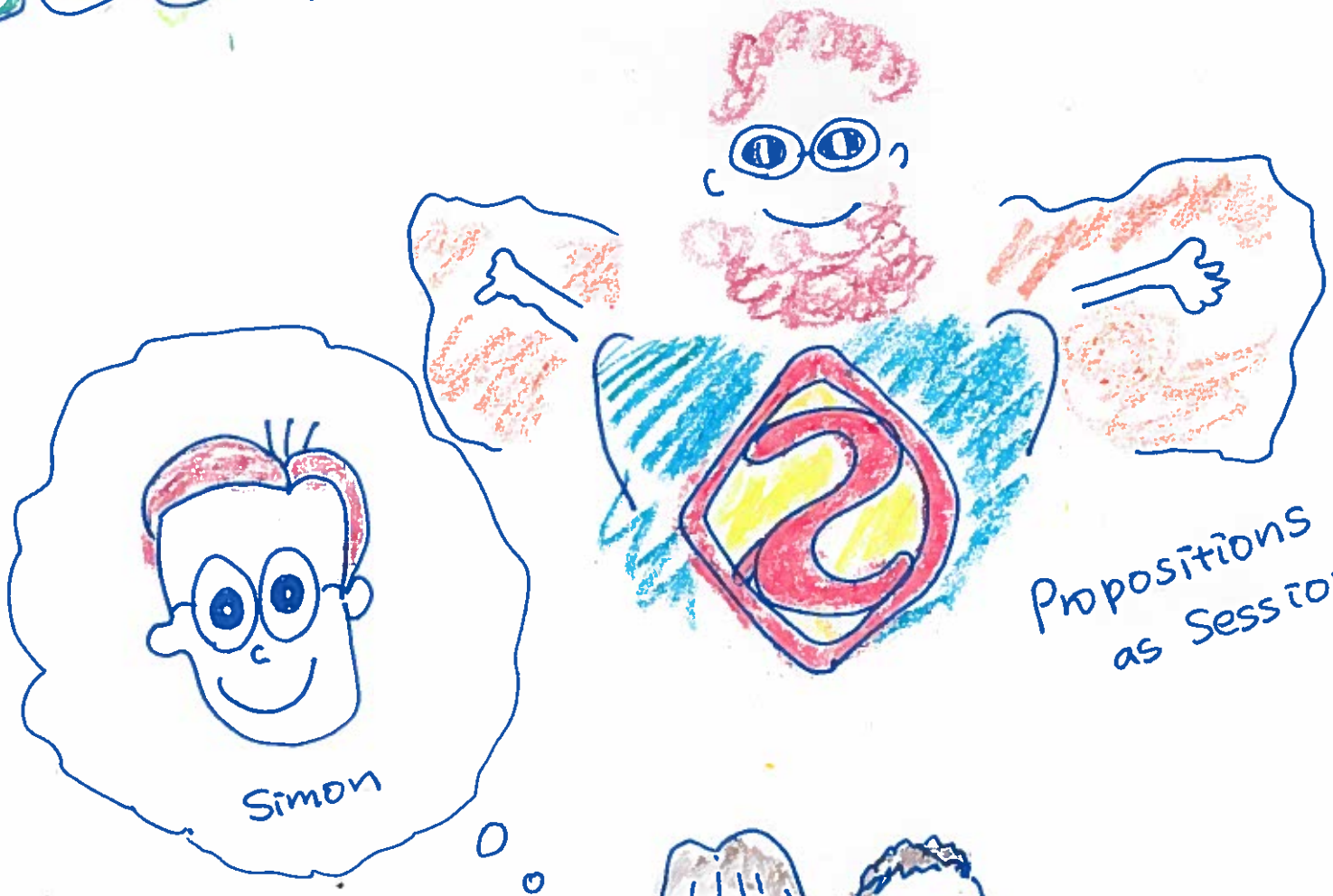
ABCD



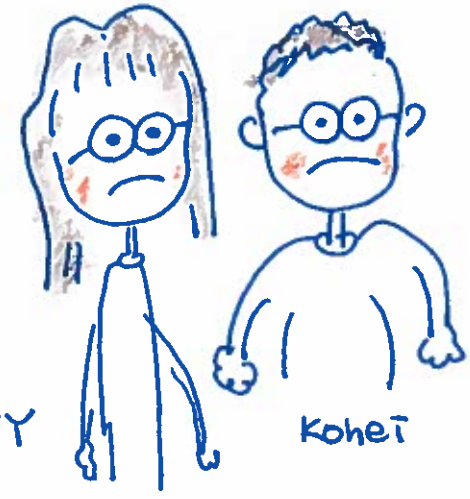
HOD

Imperial College

ABCD program grant 2013–2018 EPSRC



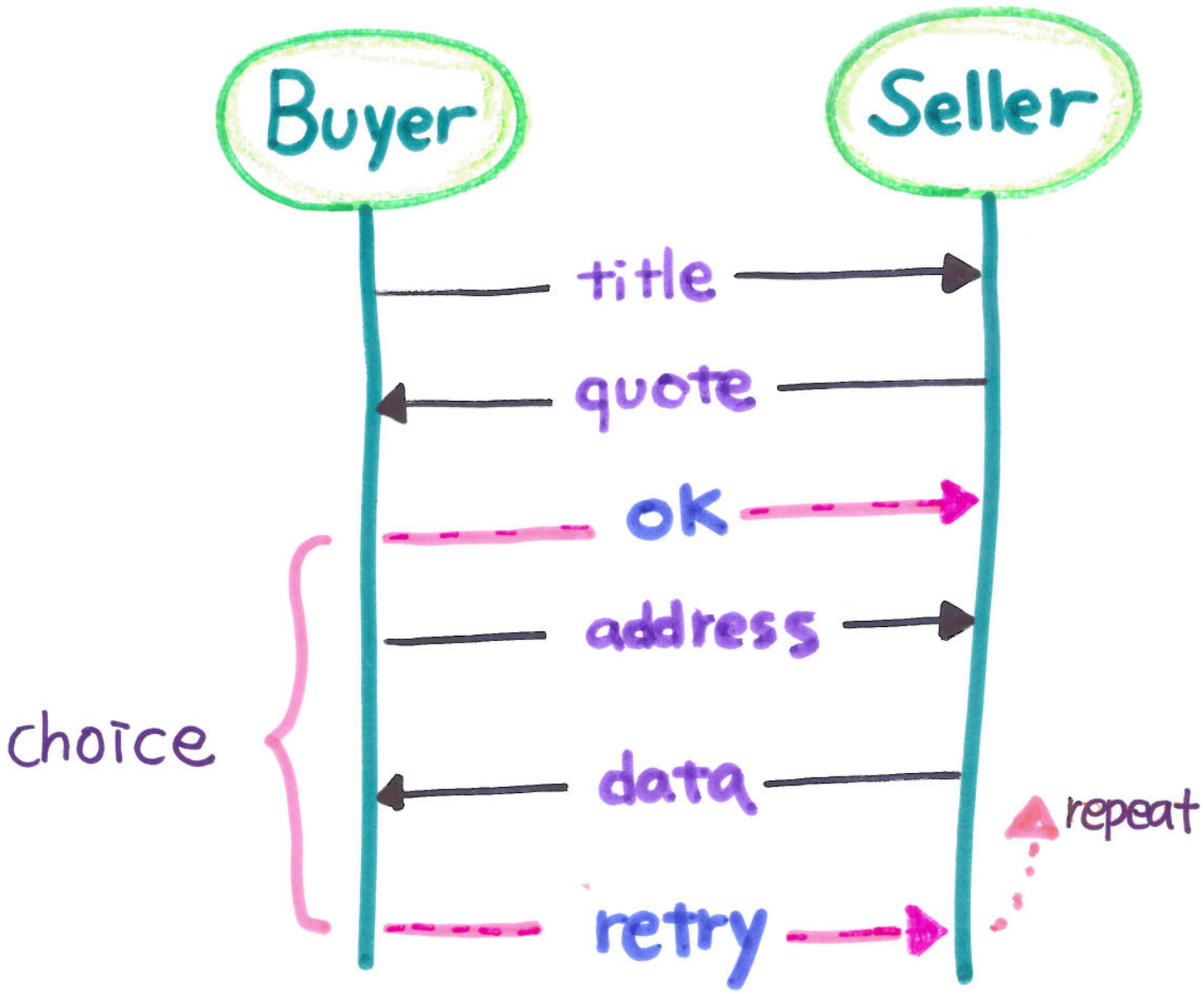
Propositions as Sessions @ ICFP'12



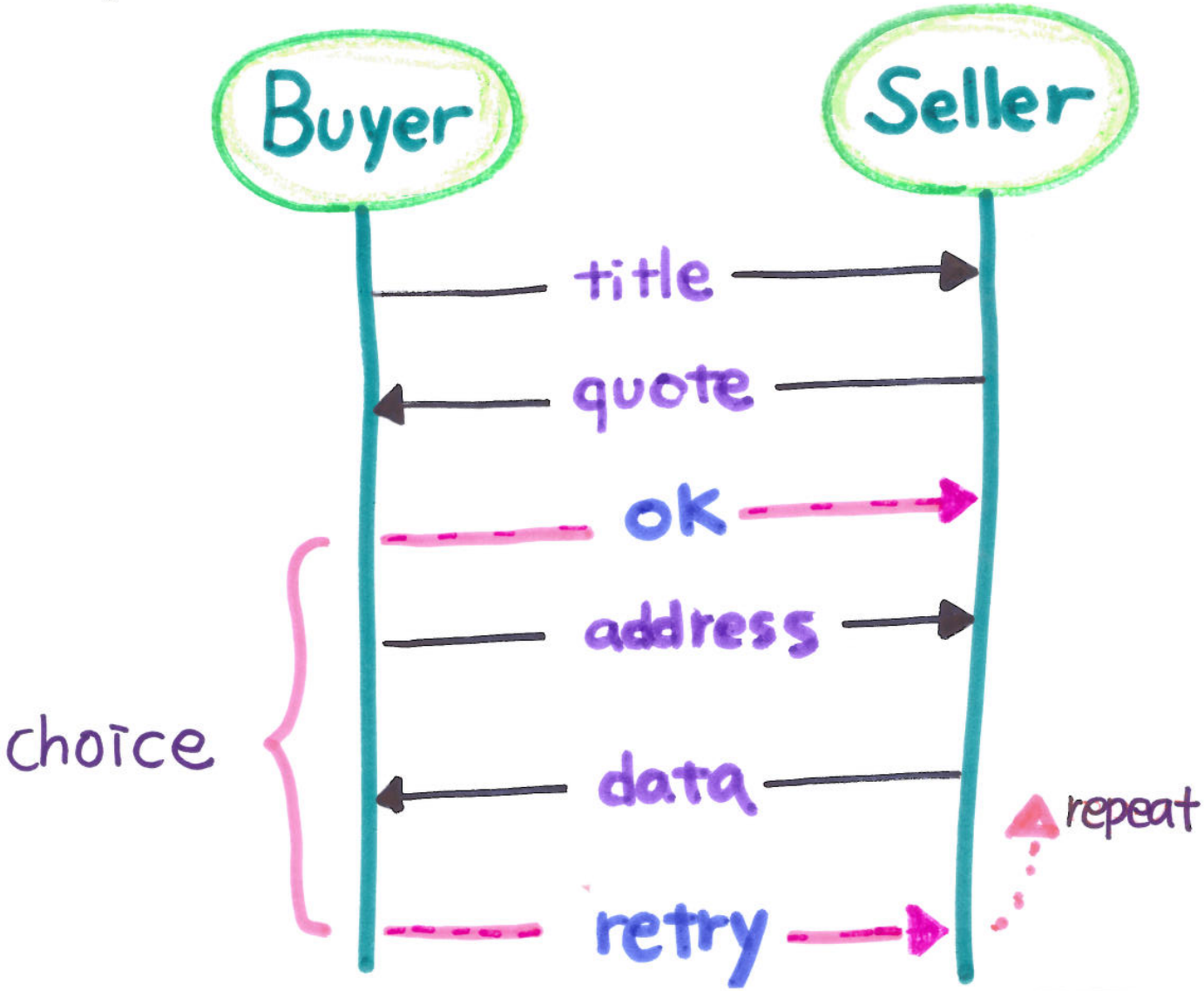
HOD

Imperial College

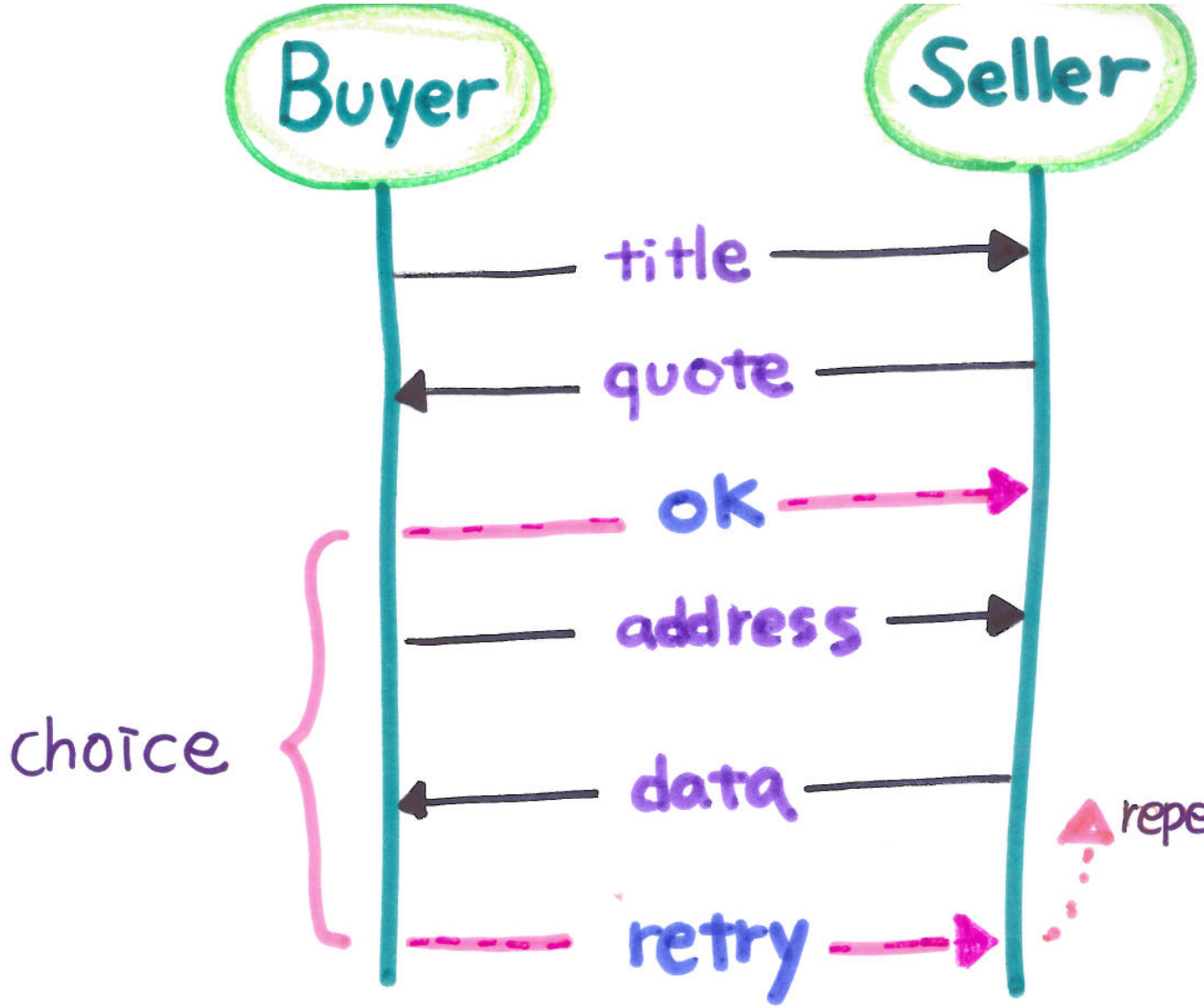
Binary Session Types: Buyer - Seller Protocol



Binary Session Types: Buyer - Seller Protocol



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



P has T

Q has $\overline{T}$

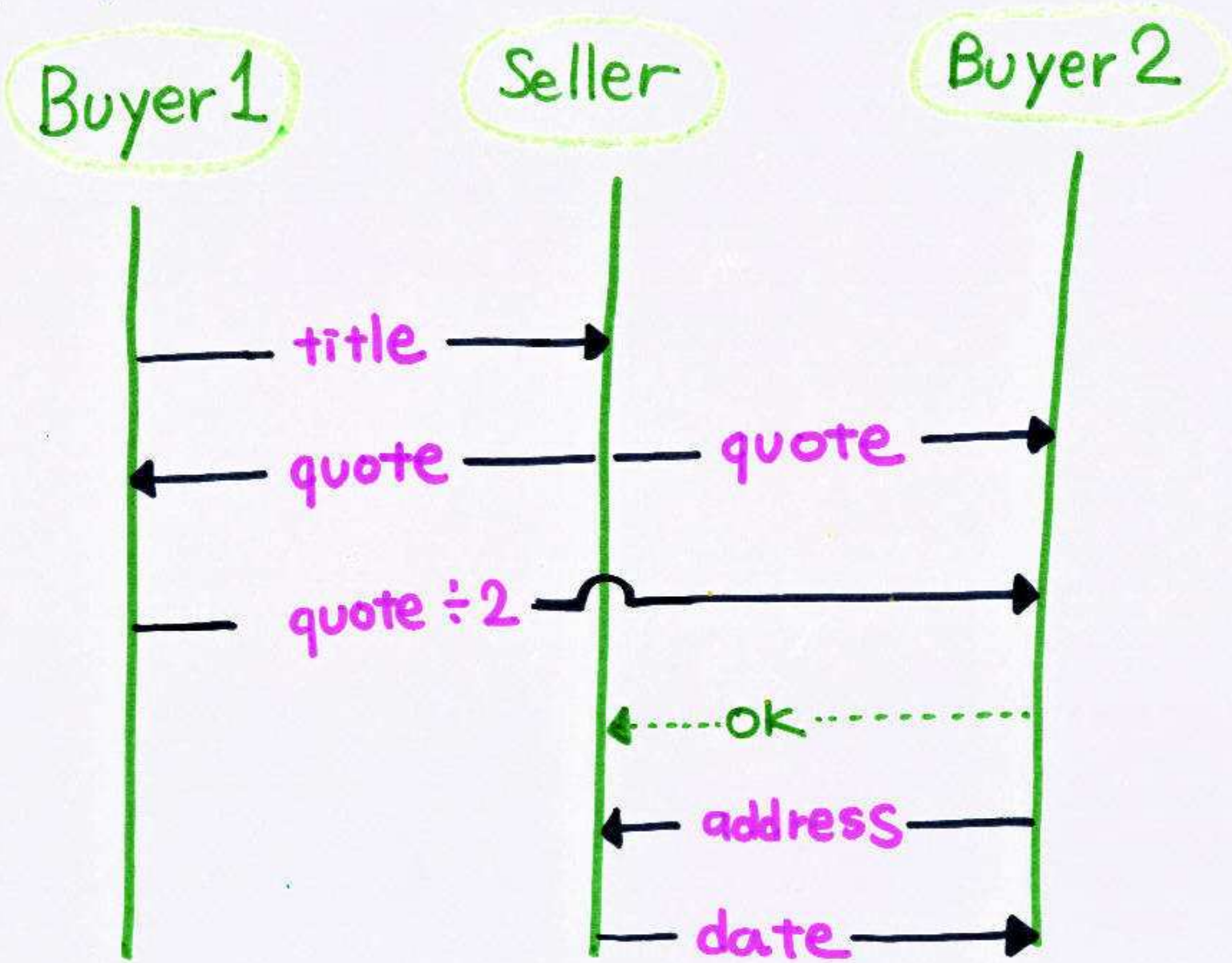
dual

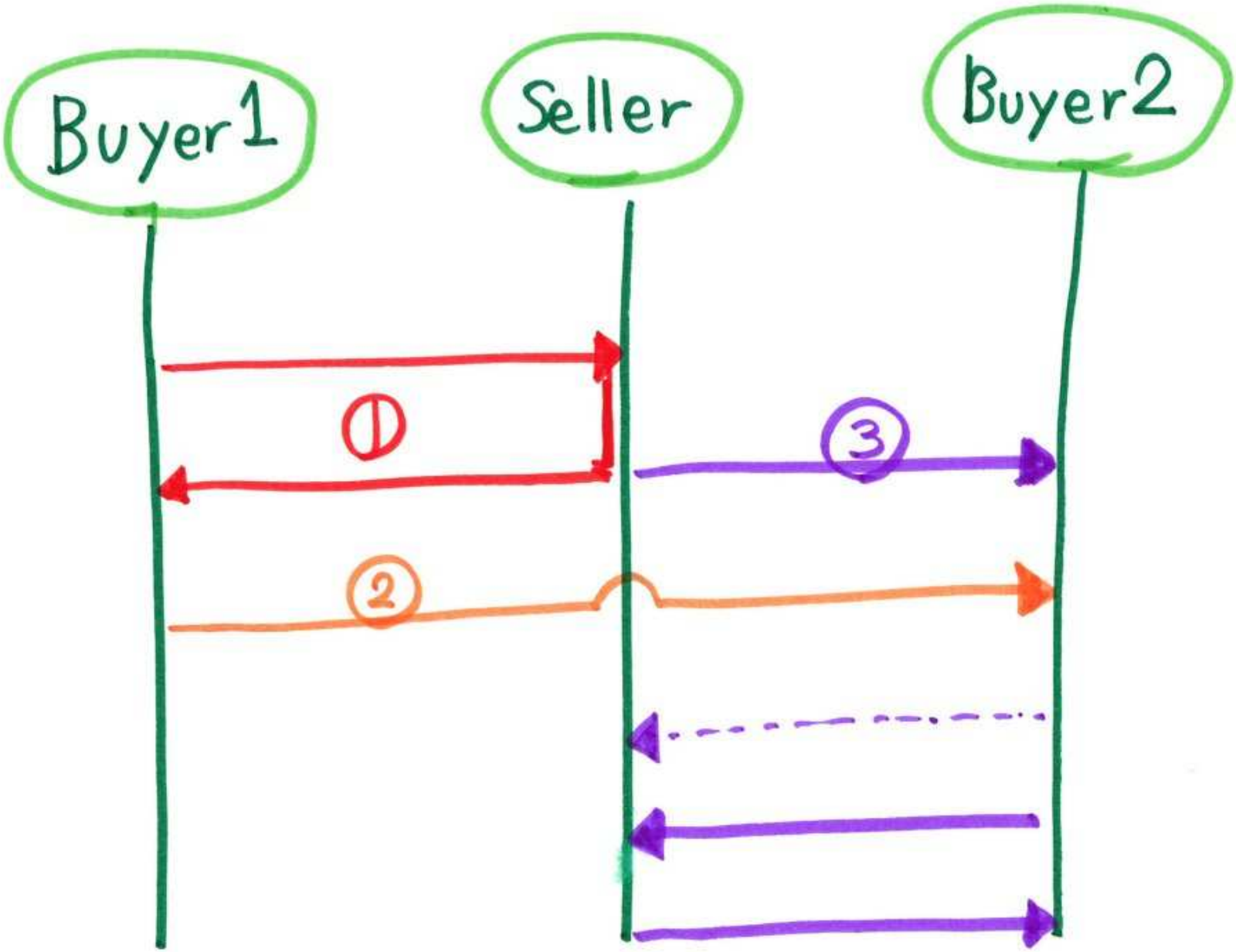
P | Q typable

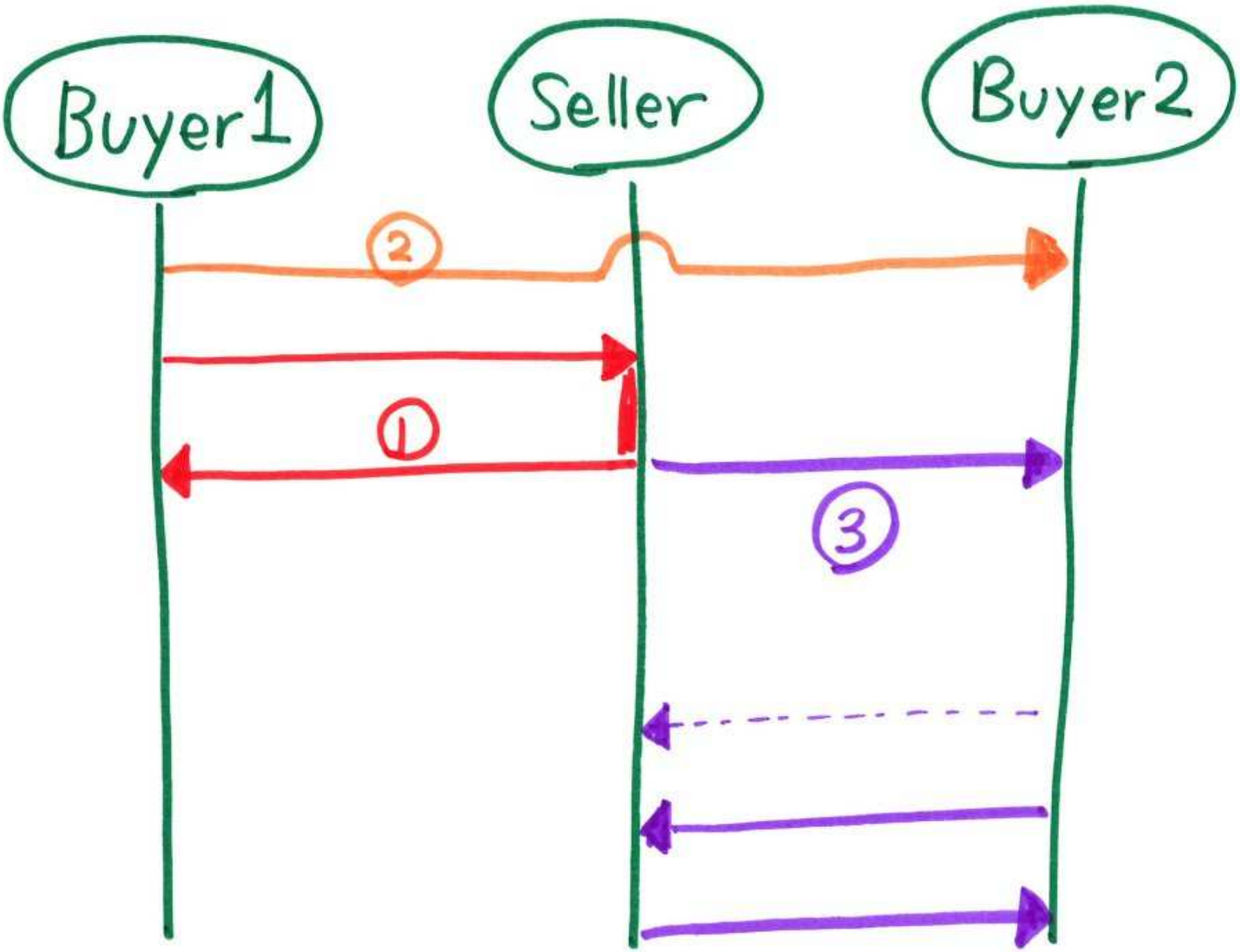
$\mu t.!\ \text{Title}\ ;?\ \text{Quote}\ ;!\ \{\text{ok}:!\ \text{Add}\ ;?\ \text{Date}\ ,\ \text{retry}:t\}$

$\mu t.?\ \text{Title}\ ;!\ \text{Quote}\ ;?\ \{\text{ok}:?\ \text{Add}\ ;!\ \text{Date}\ ,\ \text{retry}:t\}$

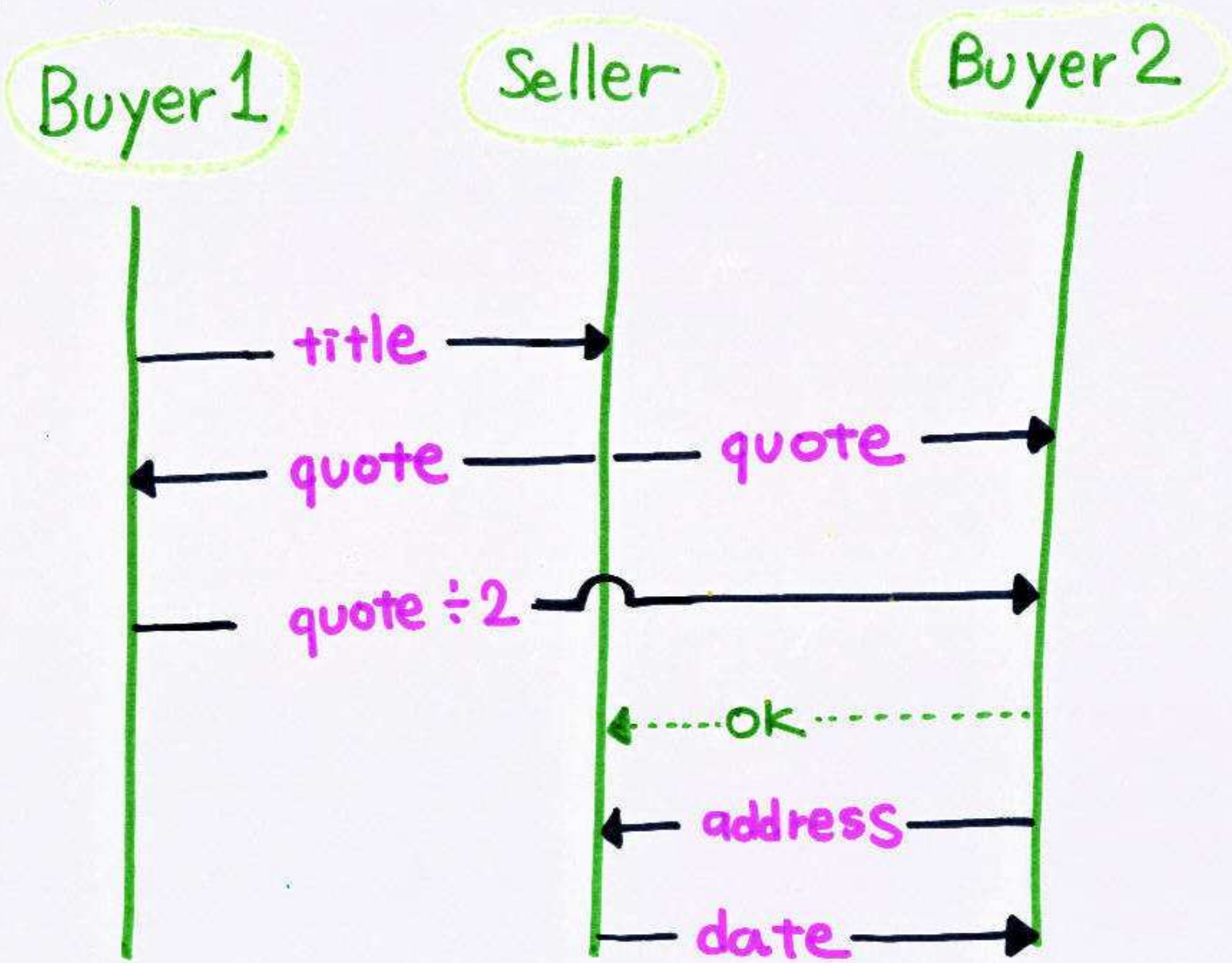
Multiparty Session Types

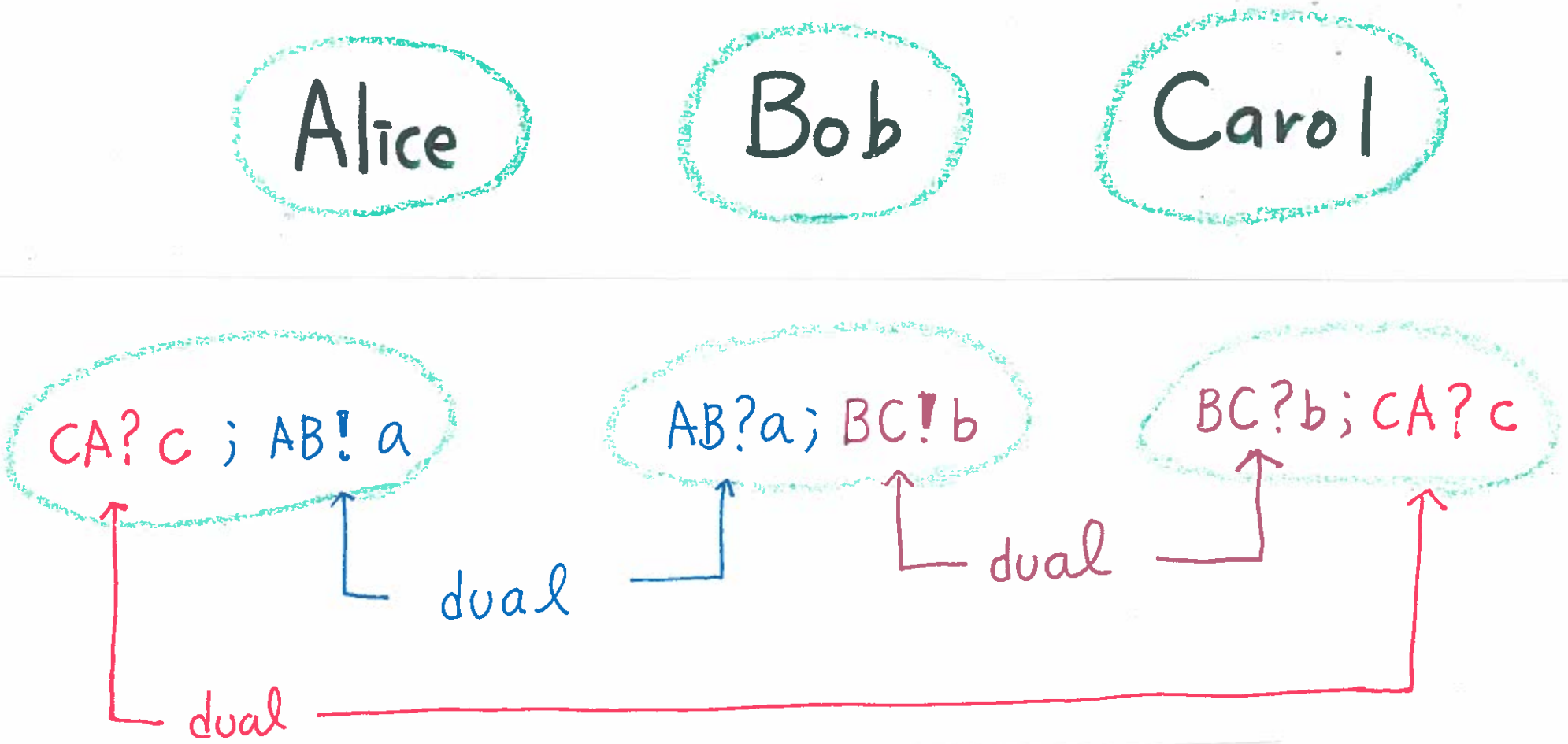



Buyer1
Seller
Buyer2
1
3
2


Buyer1
Seller
Buyer2
2
1
3

Multiparty Session Types



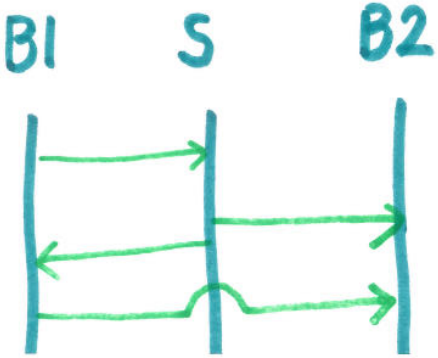


3 dual pairs

If you use binary Session Types ....

Deadlock!

Multiparty Session Types [Honda, Yoshida, Carbone 2008]



G

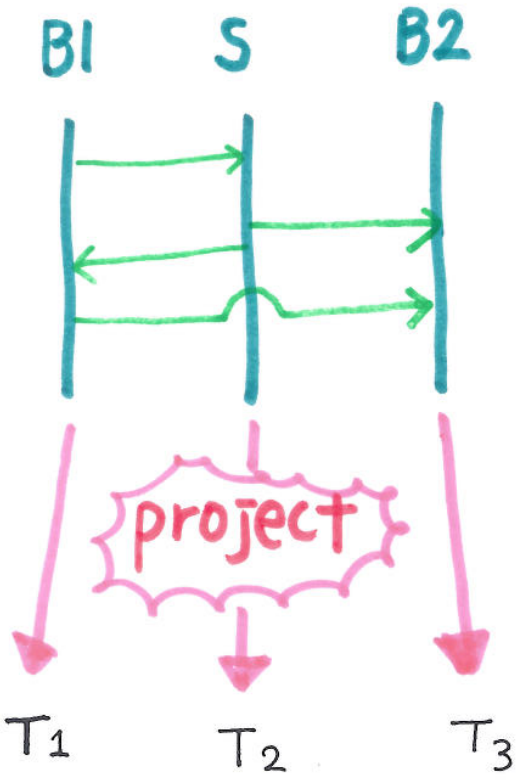
B1 → S Int.

S → B2 Char

STEP 1

Write Global Type

Multiparty Session Types [Honda, Yoshida, Carbone 2008]



G

$B1 \rightarrow S$ Int.

$S \rightarrow B2$ Char

T

$B_1?$ Int. $B_2!$ Char

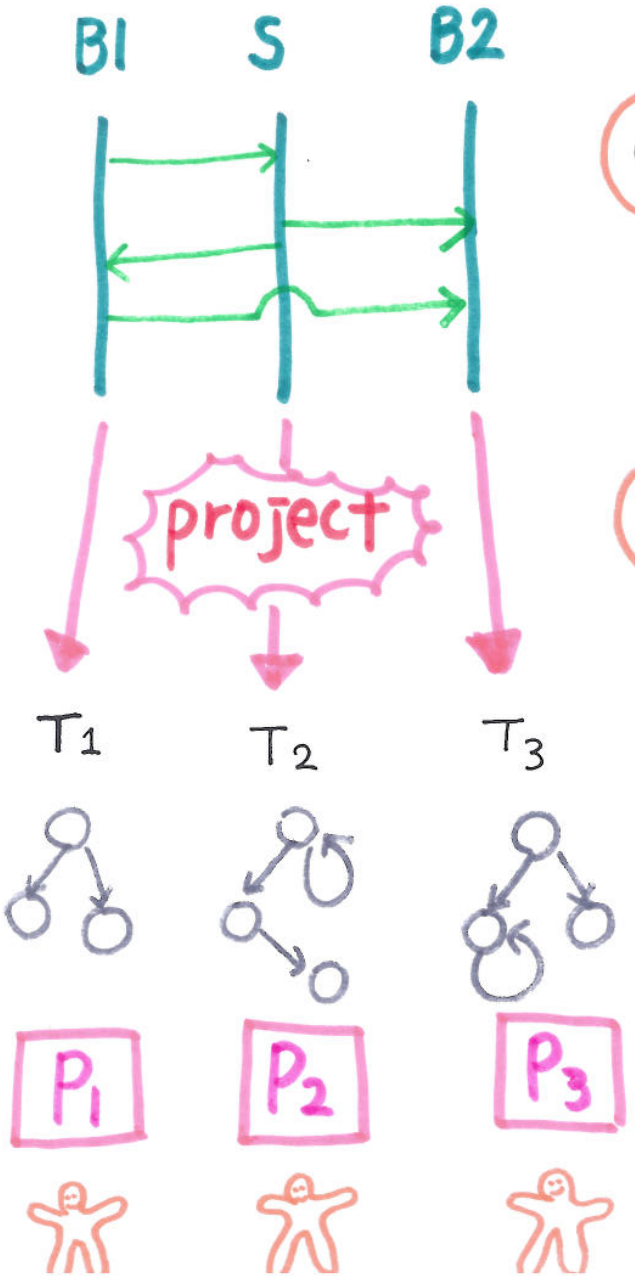
STEP 1

Write Global Type

STEP 2

Project to Local Types

Multiparty Session Types [Honda, Yoshida, Carbone 2008]



G $B1 \rightarrow S$ Int.
$S \rightarrow B2$ Char

T $B_1?\text{Int}.\ B_2!\text{Char}$

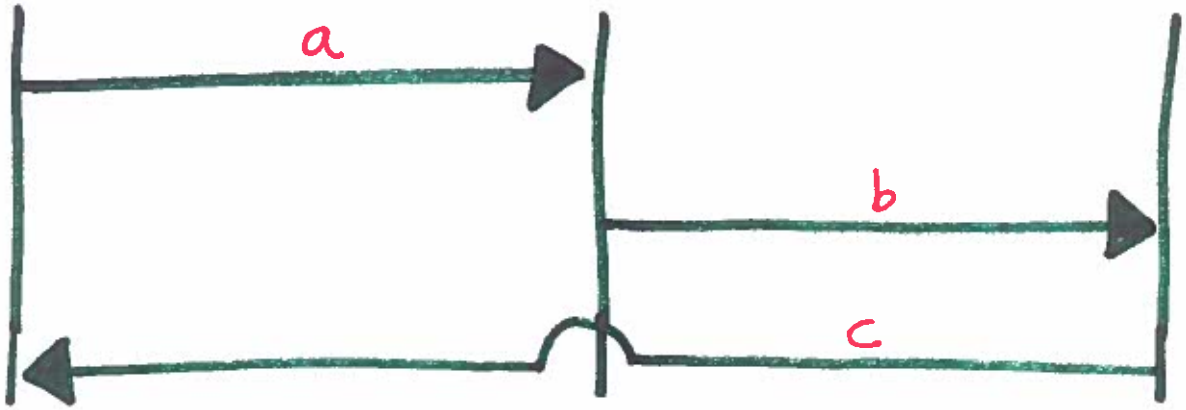
P $B_1?(x).\ B_2!\langle\text{"apple"}\rangle$

STEP 1
Write Global Type

STEP 2
Project to Local Type

STEP 3
- Static Check
- Generate Code
- Run-time check

Alice Bob Carol



Global Type

Projection

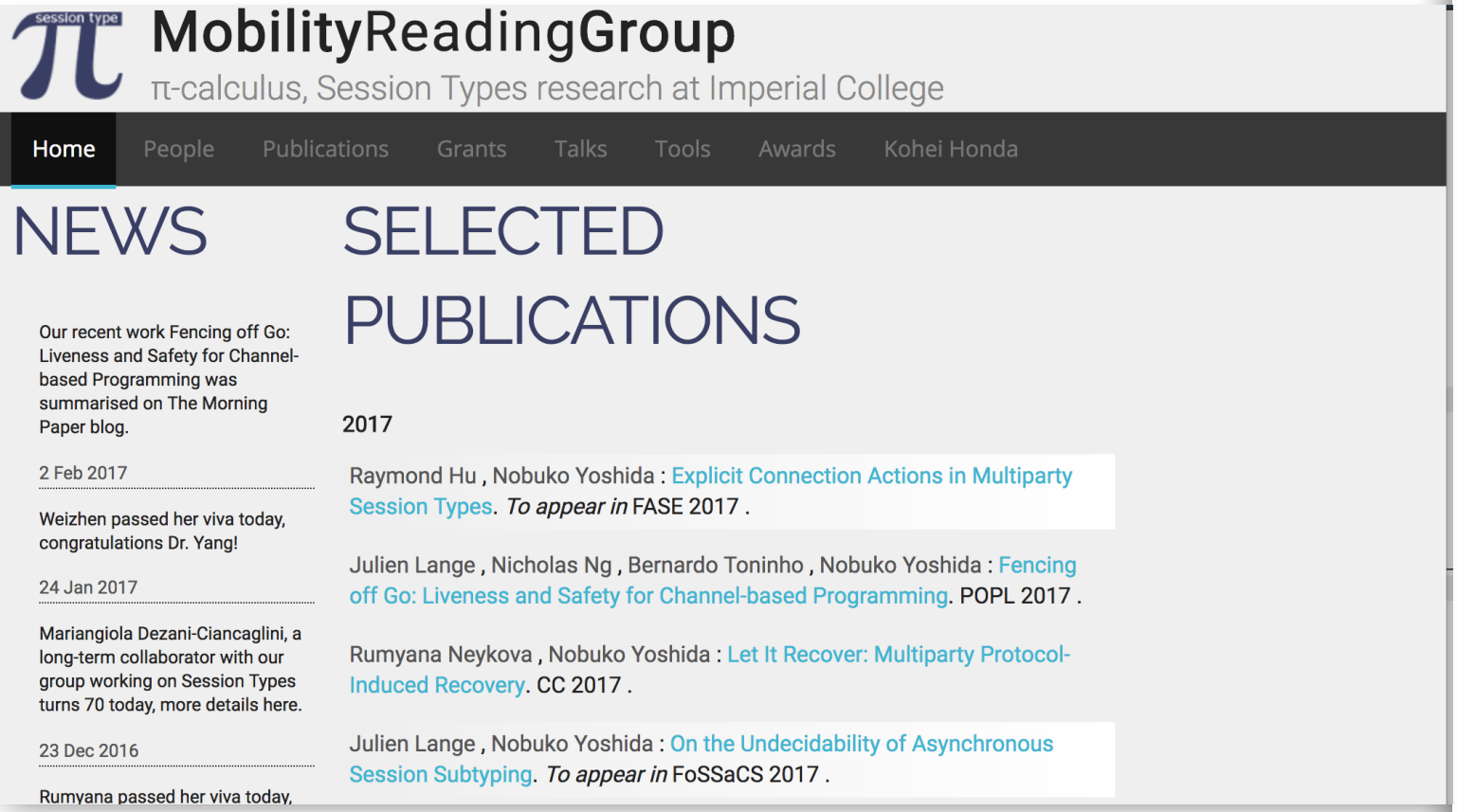
Alice AB!a; CA?c

Bob AB?a; BC!b

Carol BC?b; CA!c;

LOCAL TYPES

Us ∈ Mobility Research Group



http://mrg.doc.ic.ac.uk/

Academic Staff

Nobuko Yoshida

Research Associate

Raymond Hu

Julien Lange

Nicholas Ng

Xinyu Niu

Alceste Scalas

Bernardo Toninho

PhD Student

Assel Altayeva

Juliana Franco

Rumyana Neykova

Weizhen Yang

Selected Publications 2016/2017

- **[ECOOP'17]** Alceste Scala, Raymond Hu, Ornela Darda, NY: A Linear Decomposition of Multiparty Sessions for Safe Distributed Programming..
- **[COORDINATION'17]** Keigo Imai, NY and Shoji Yuen: Session-ocaml: a session-based library with polarities and lenses.
- **[FoSSaCS'17]** Julien Lange , NY: On the Undecidability of Asynchronous Session Subtyping.
- **[FASE'17]** Raymond Hu , NY: Explicit Connection Actions in Multiparty Session Types.
- **[CC'17]** Rumyana Neykova , NY: Let It Recover: Multiparty Protocol-Induced Recovery.
- **[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 .

Selected Publications 2016/2017

- [ECOOP'17] Alceste Scala, Raymond Hu, Ornela Darda, NY :A Linear Decomposition of Multiparty Sessions for Safe Distributed Programming.
- [COORDINATION'17] Keigo Imai, NY and Shoji Yuen: Session-ocaml: a session-based library with polarities and lenses.
- [FoSSaCS'17] Julien Lange , NY : On the Undecidability of Asynchronous Session Subtyping.
- [FASE'17] Raymond Hu , NY : Explicit Connection Actions in Multiparty Session Types.
- [CC'17] Rumyana Neykova , NY: Let It Recover: Multiparty Protocol-Induced Recovery.
- [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.

Relationship with Communicating AUTOMATA

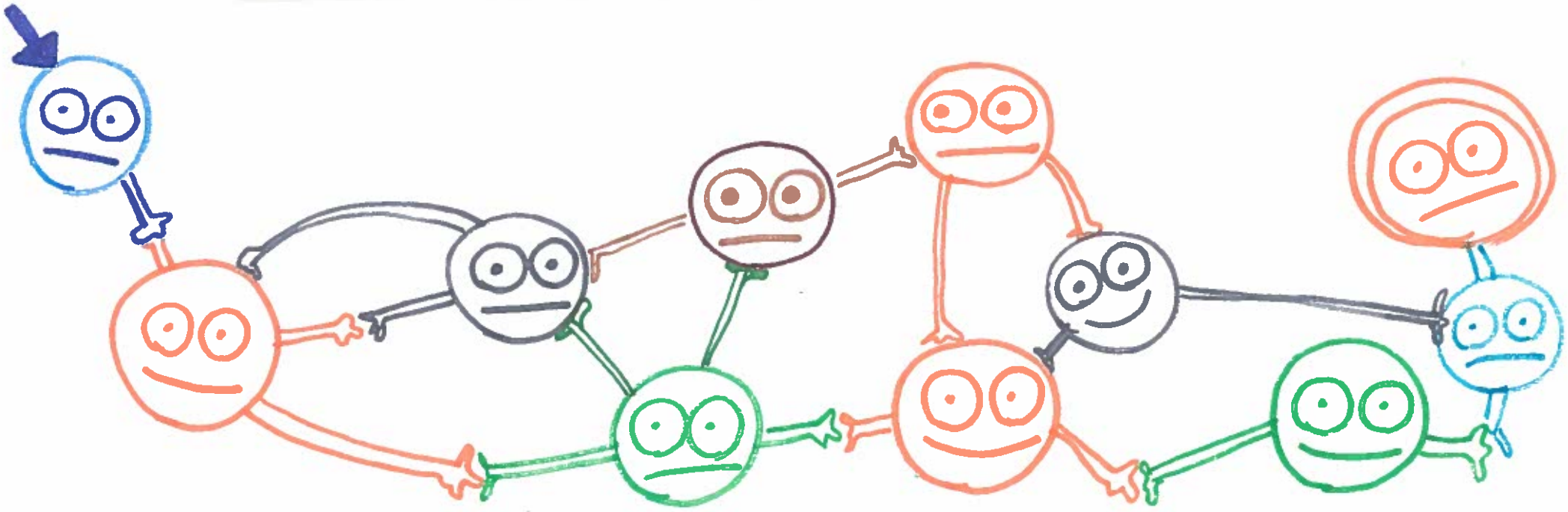
ESOP 12
ICALP 13
Characterisation
Synthesis

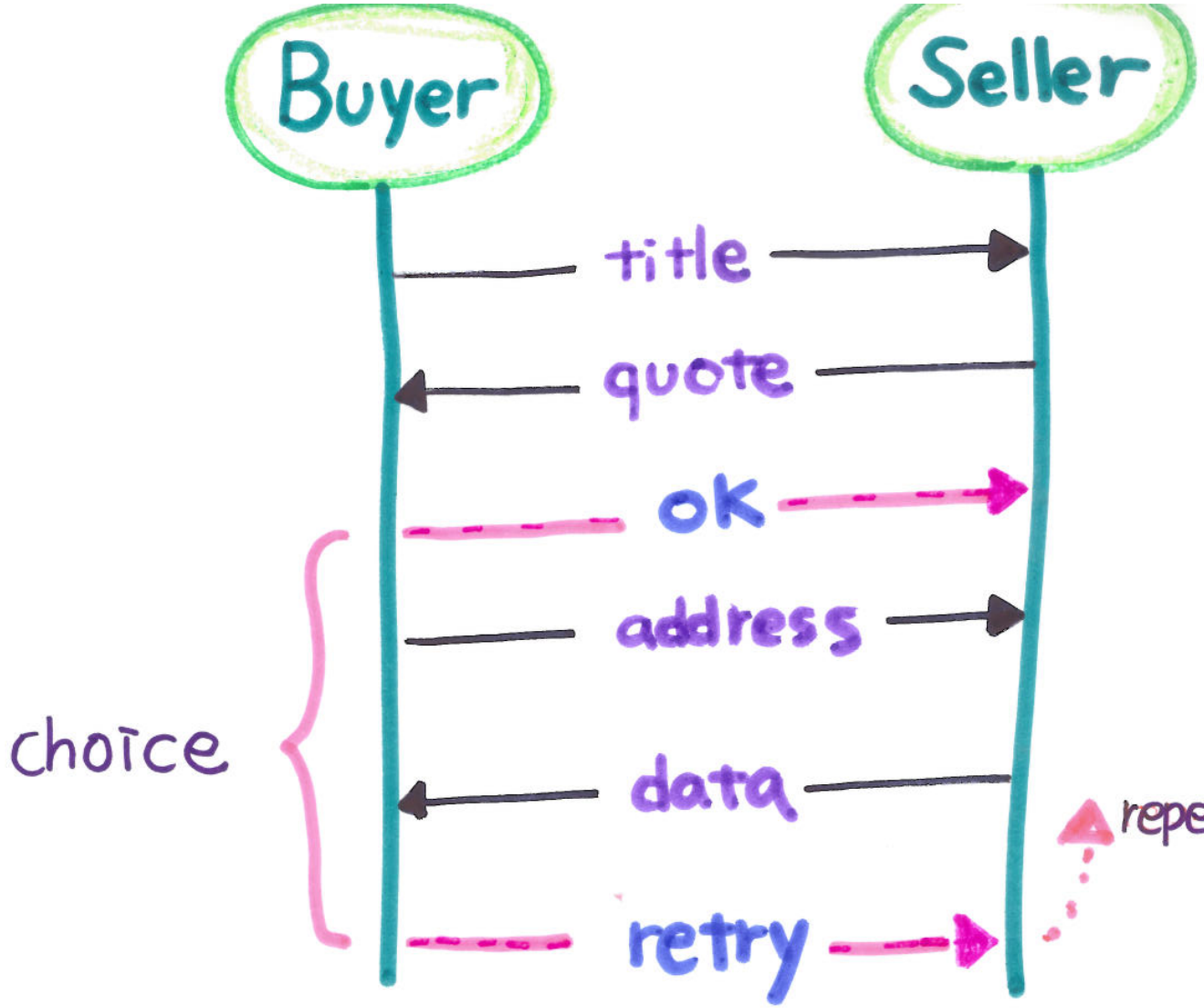
CONCUR 15
Timed Automata

POPL 15
Graphical Synthesis

TACAS 16
Subtyping + Model-checking

FOSSACS17
Undecidability





P has T

Q has $\overline{T}$

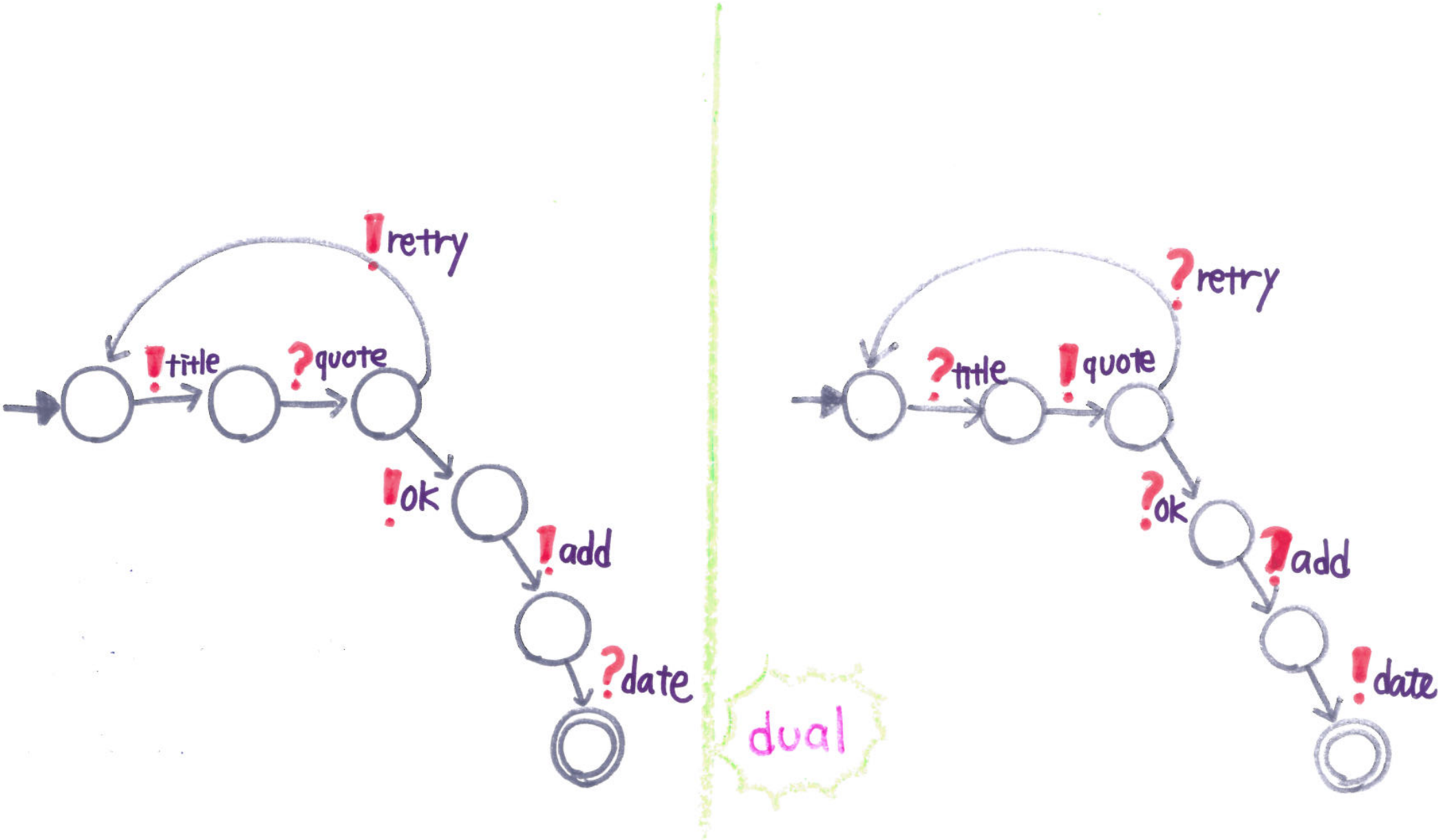
dual

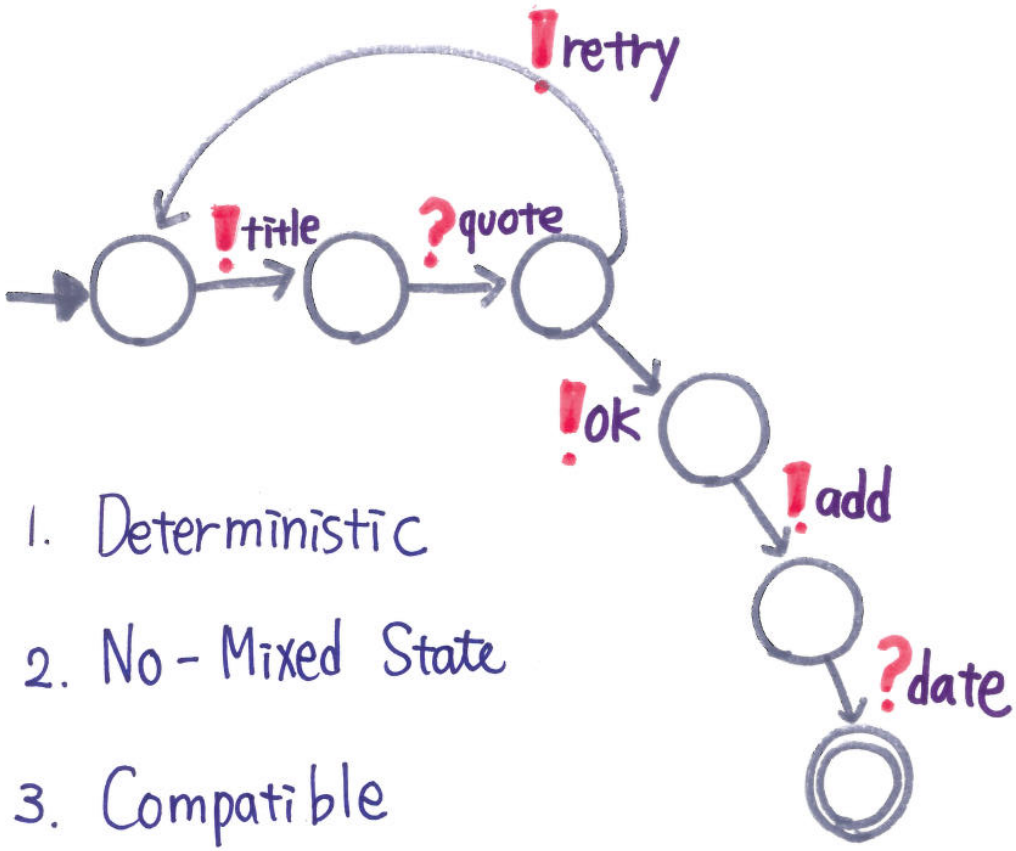
P | Q typable

$\mu t.!\ \text{Title}\ ;?\ \text{Quote}\ ;!\ \{\text{ok}:!\ \text{Add}\ ;?\ \text{Date}\ ,\ \text{retry}:t\}$

$\mu t.?\ \text{Title}\ ;!\ \text{Quote}\ ;?\ \{\text{ok}:?\ \text{Add}\ ;!\ \text{Date}\ ,\ \text{retry}:t\}$

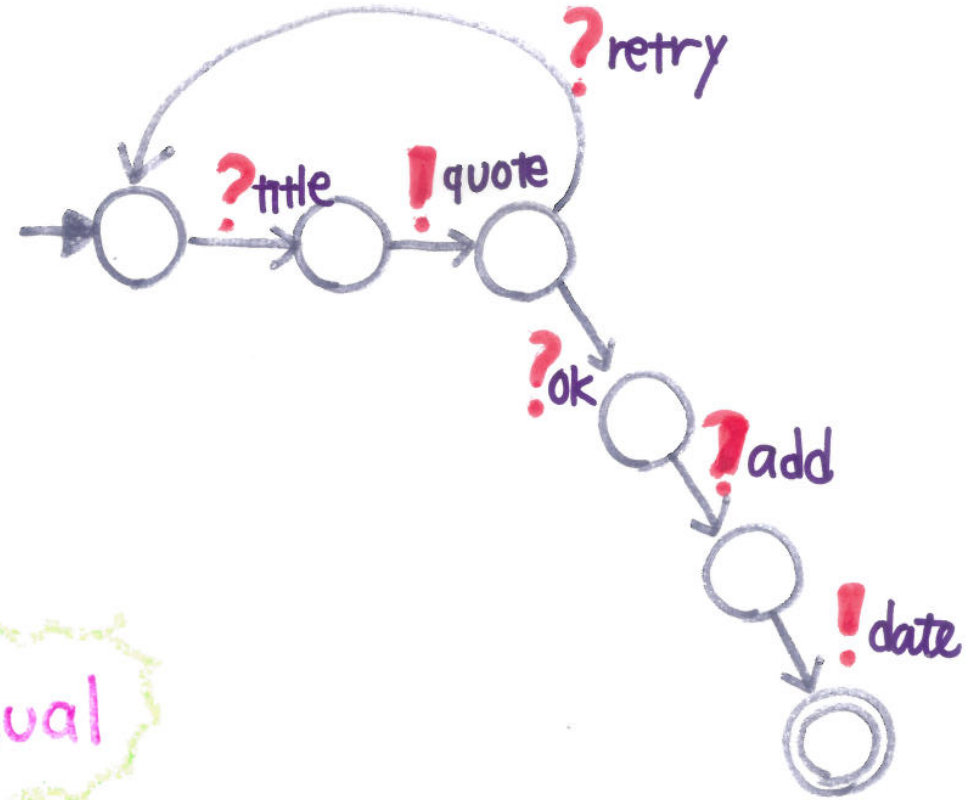
Communicating Automata [1980s]





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

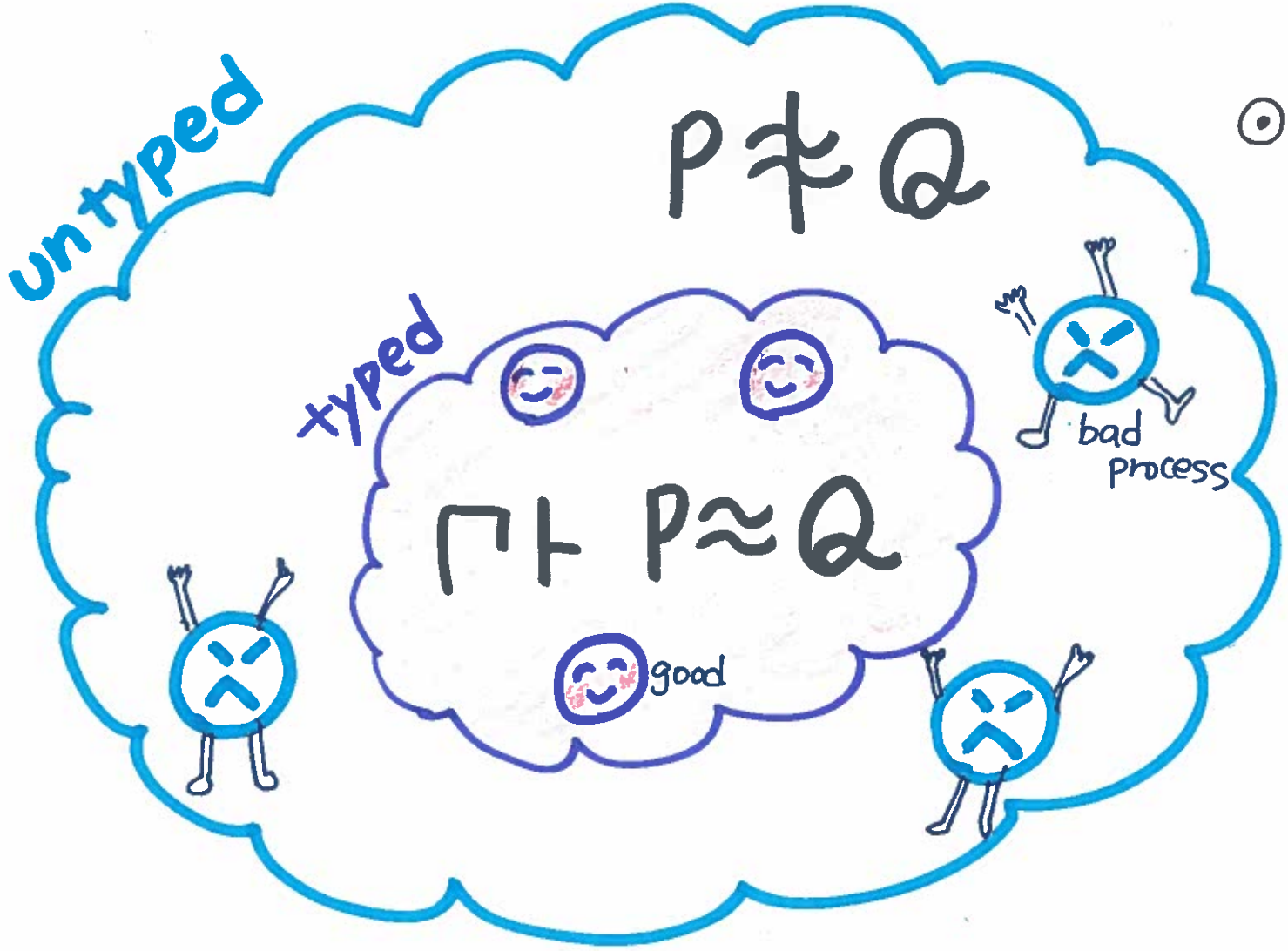
dual



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

Typed Semantics in π 1991→

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



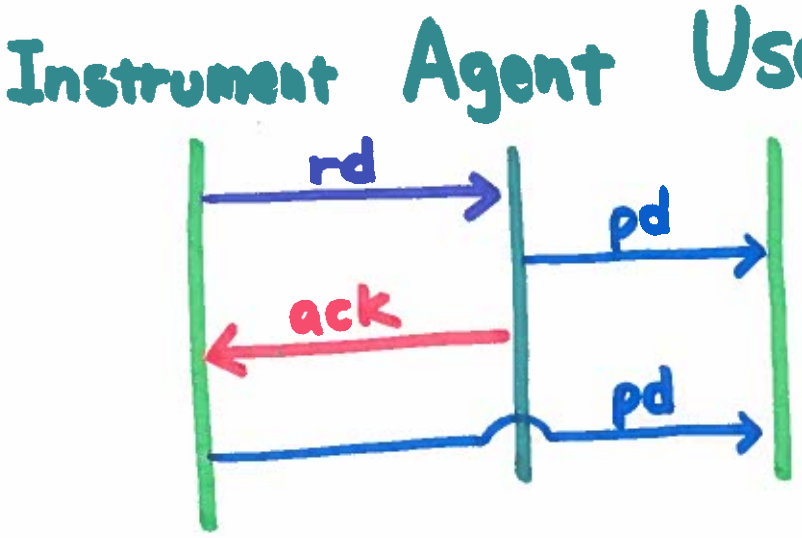
- ⊙ Correctness of Encoding
- ⊙ Limit environment Γ
  ⇒ Equate more processes
- ⊙ Compositional

GLOBALLY GOVERNED SESSION SEMANTICS

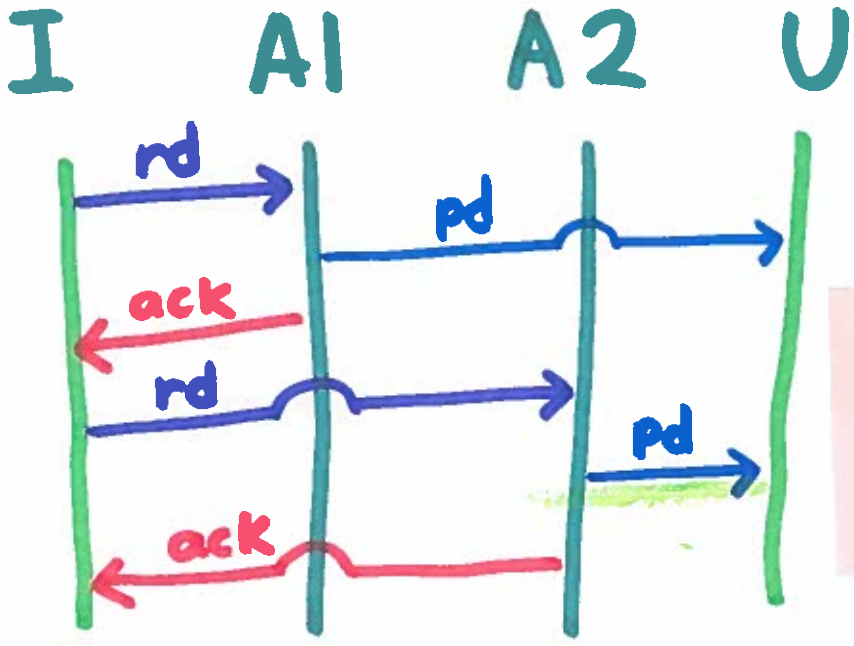
Dmitrios Kouzapas
Glasgow

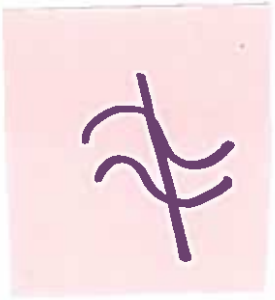
Nobuko Yoshida
Imperial college London

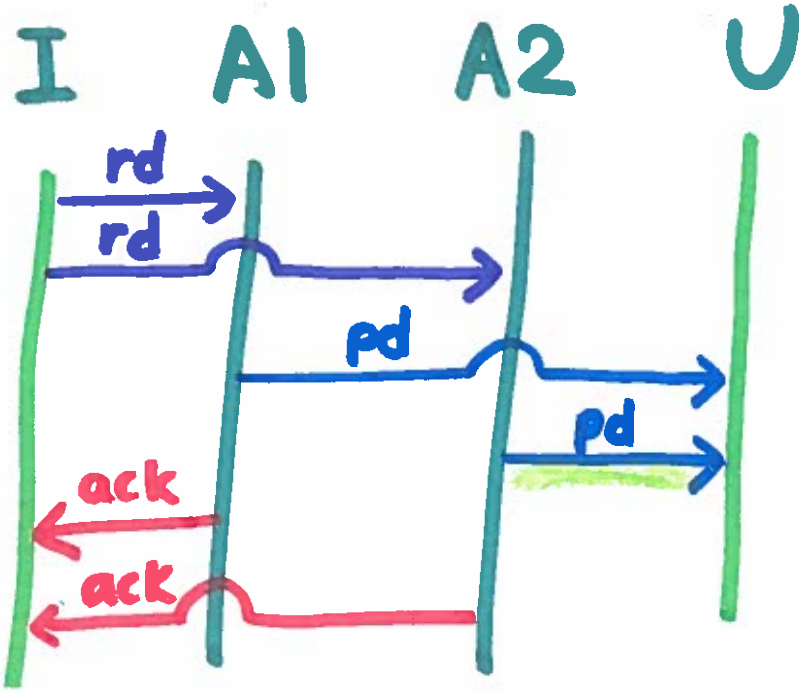
Usecase : UC.R2.13 "Acquire Data from Instrument" from Ocean Observatories Initiative (OOI)



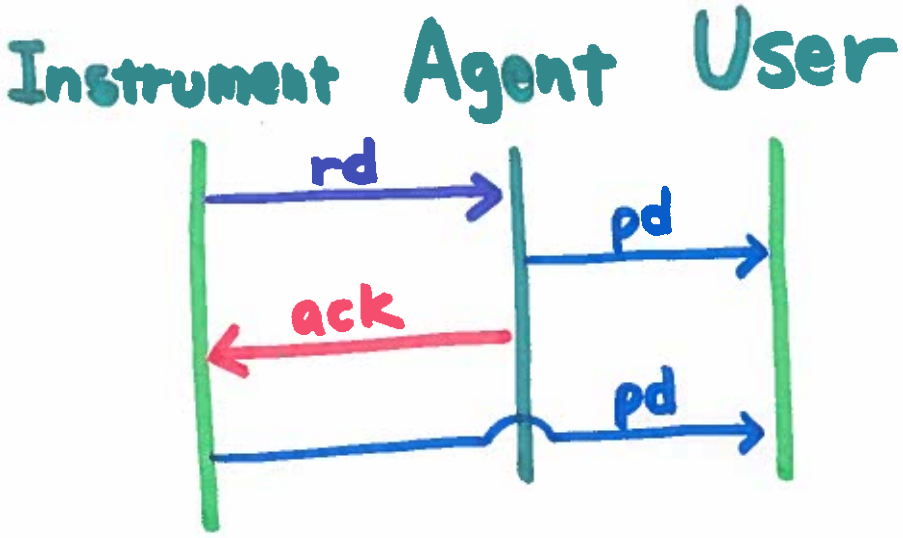
A1 → U: <pd>.
A2 → U: <pd>.
end



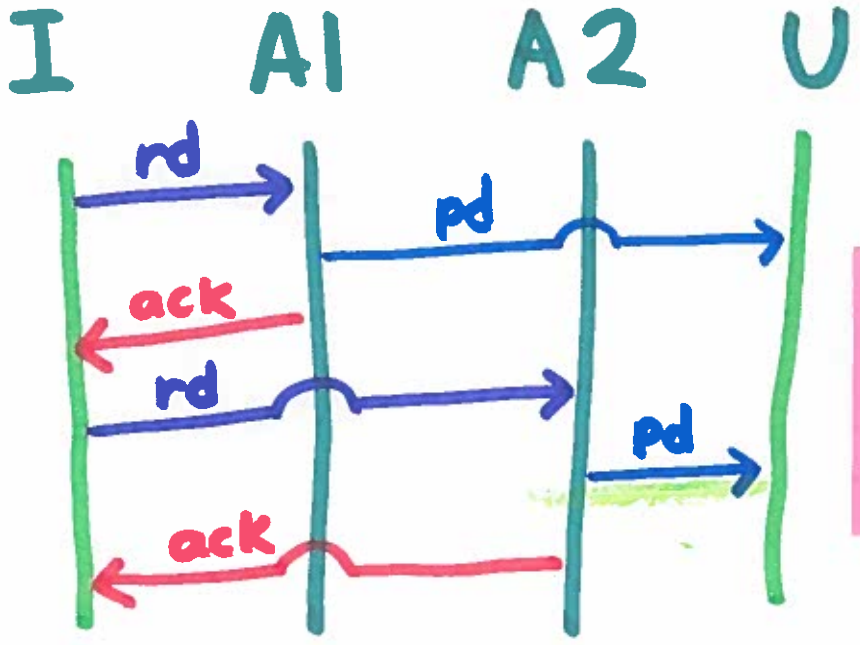




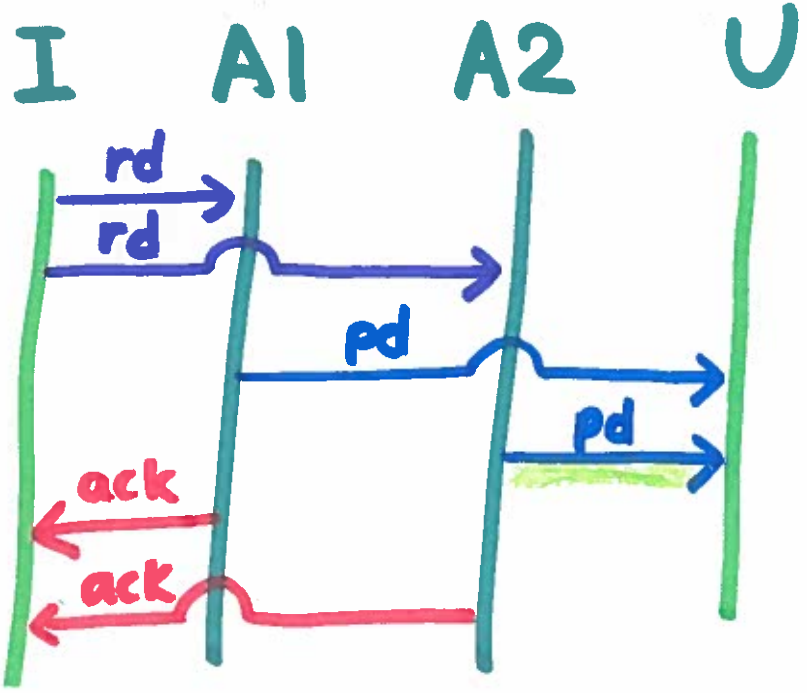
Usecase : UC.R2.13 "Acquire Data from Instrument" from Ocean Observatories Initiative (OOI)



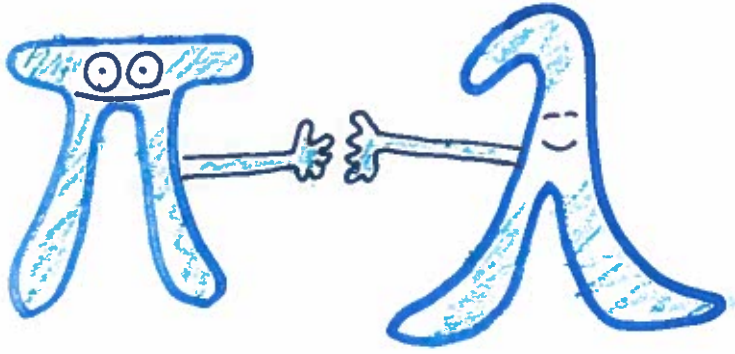
A1 → U : ⟨pd⟩.
A2 → U : ⟨pd⟩.
end



≈g



HOπ and TYPES

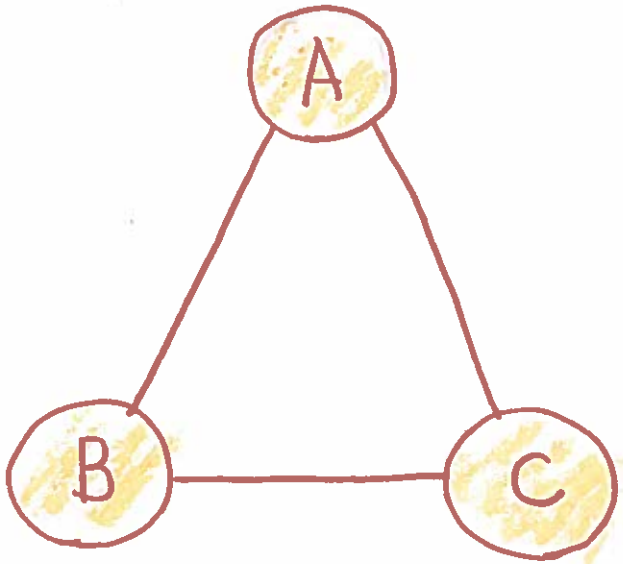


CONCUR 16 Characteristic Bisimulations

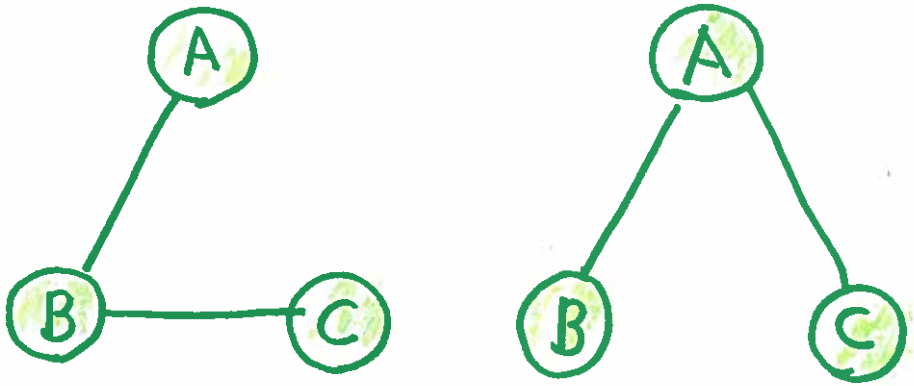
ESOP 16 Expressiveness

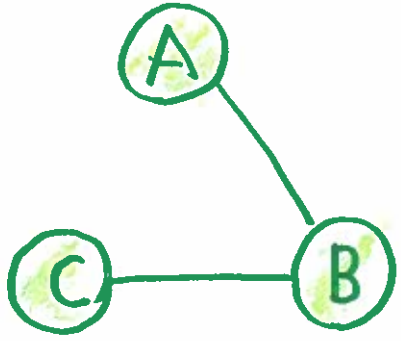
ACTA INFORMATICA

Interconnectability of Session Logical Processes



Multi Party

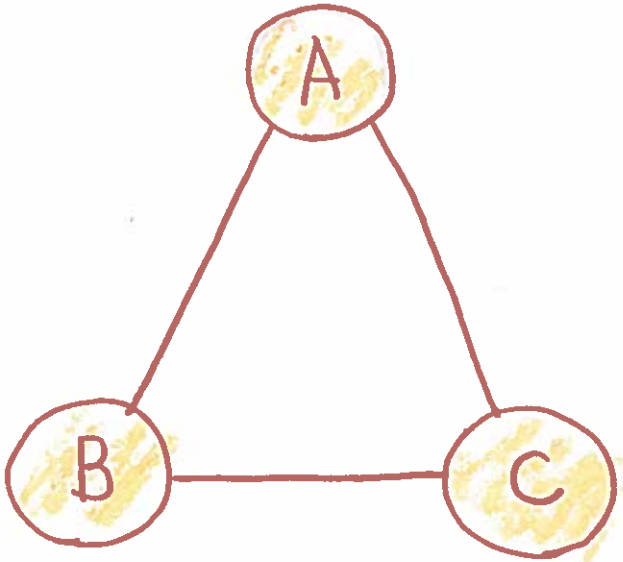




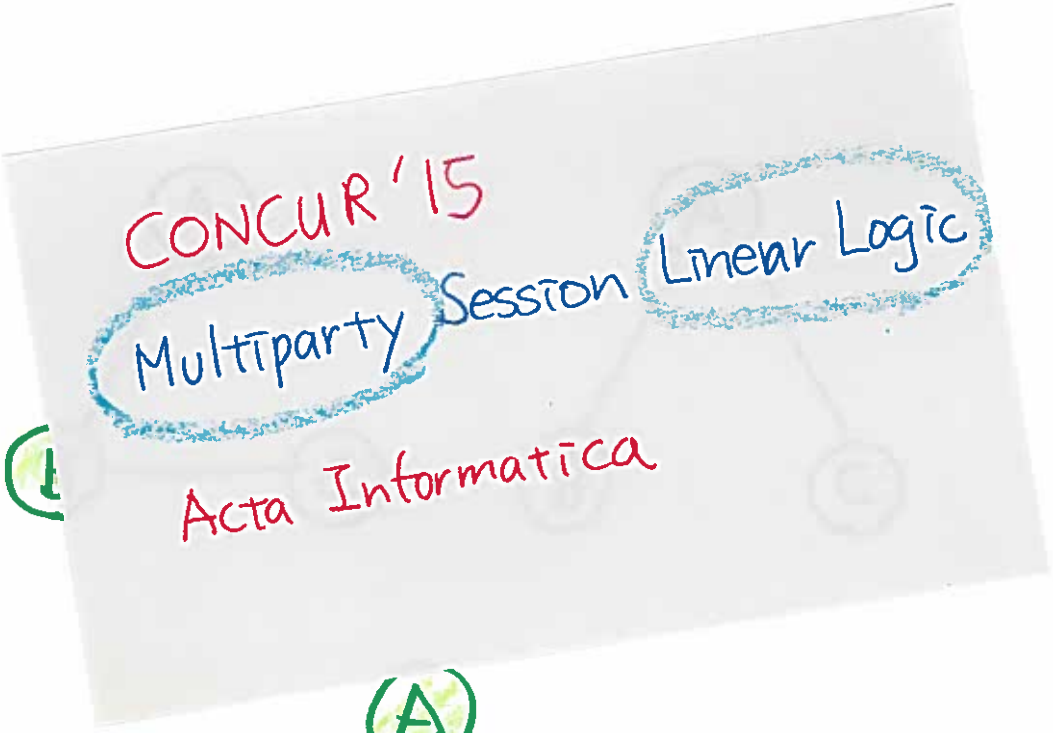
Linear Logical Session

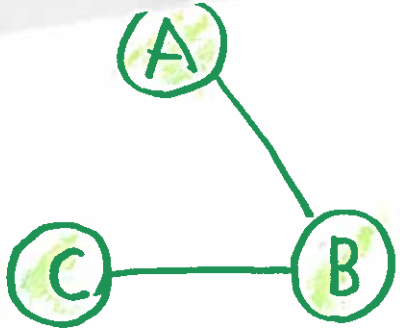
cf. [1995] Specification Structures and Proposition as Types

Interconnectability of Session Logical Processes



Multi Party





Linear Logical Session

cf. [1995] Specification Structures and Proposition as Types

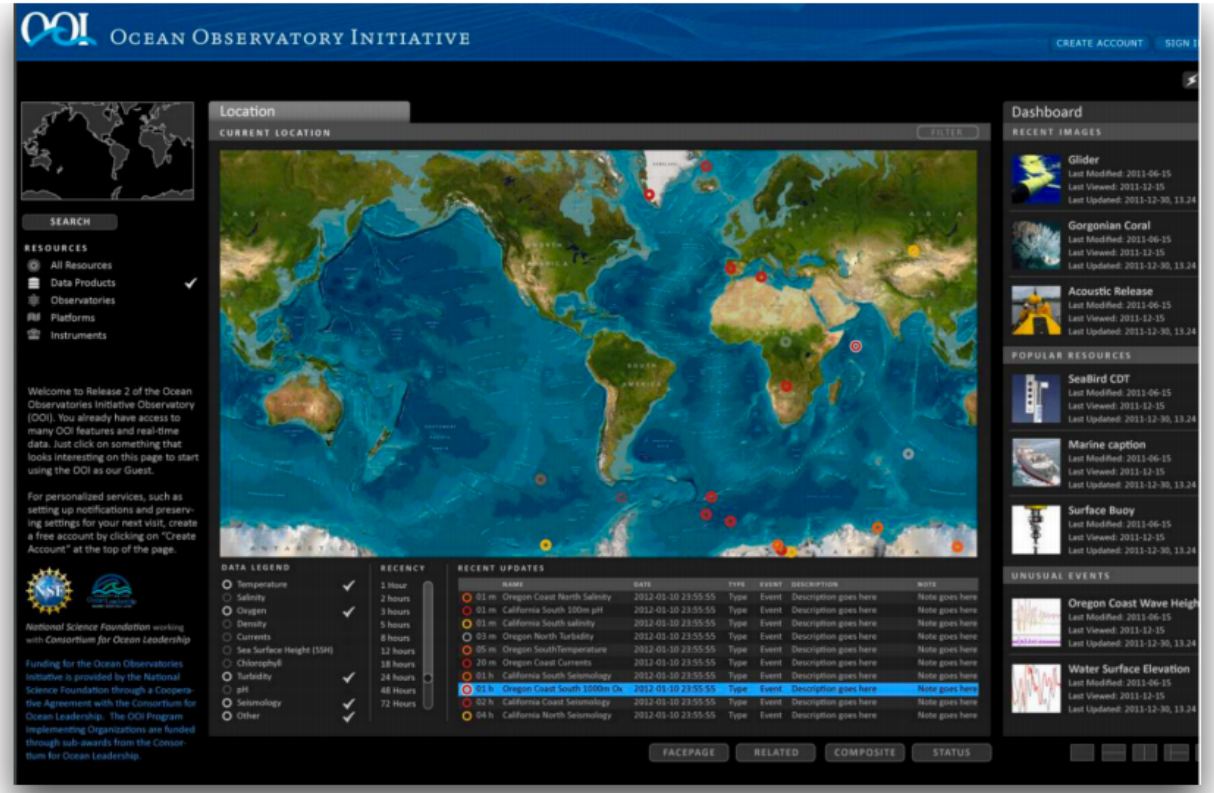
www.scribble.org

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 }
11
```

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

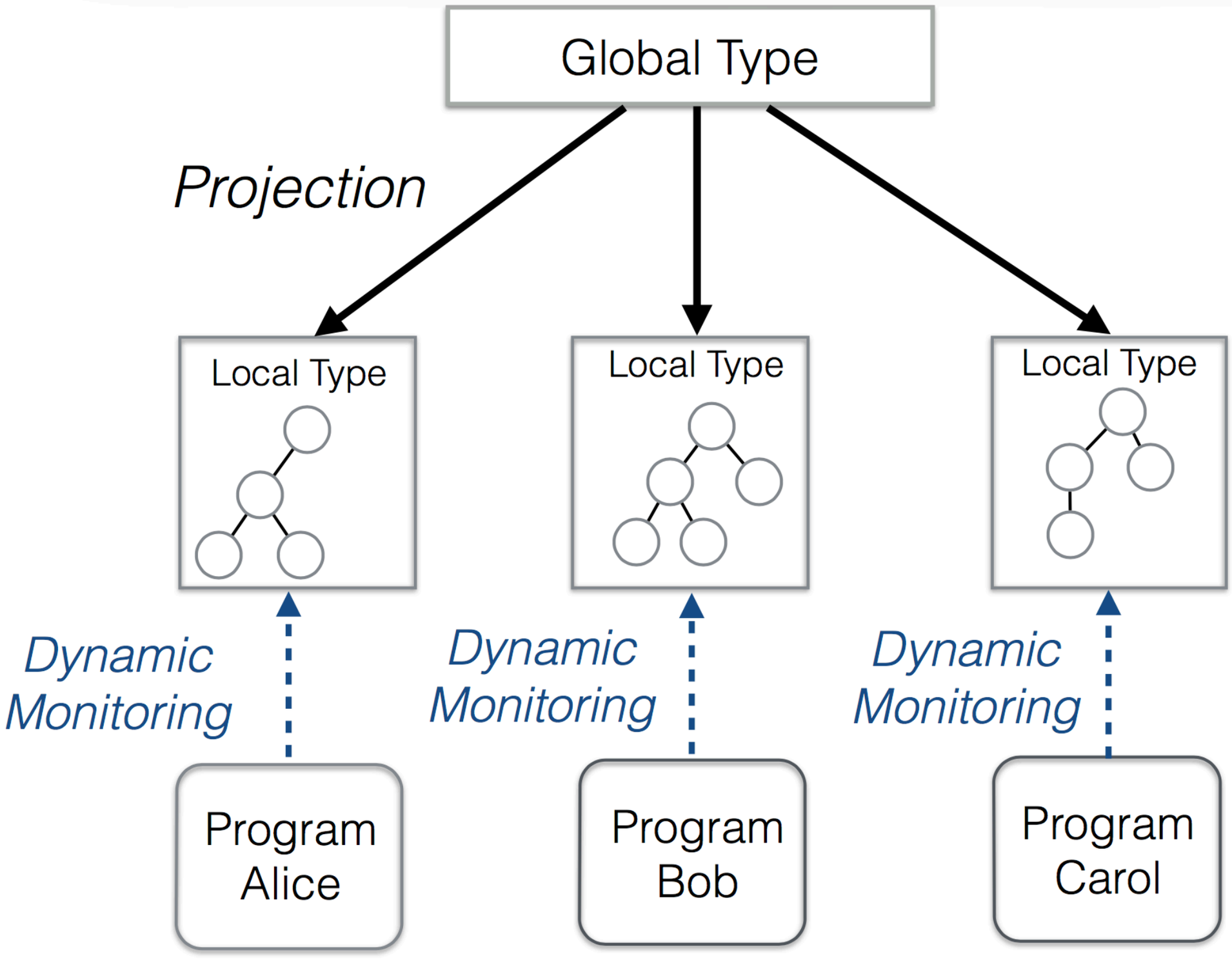
OOI Collaboration



- **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

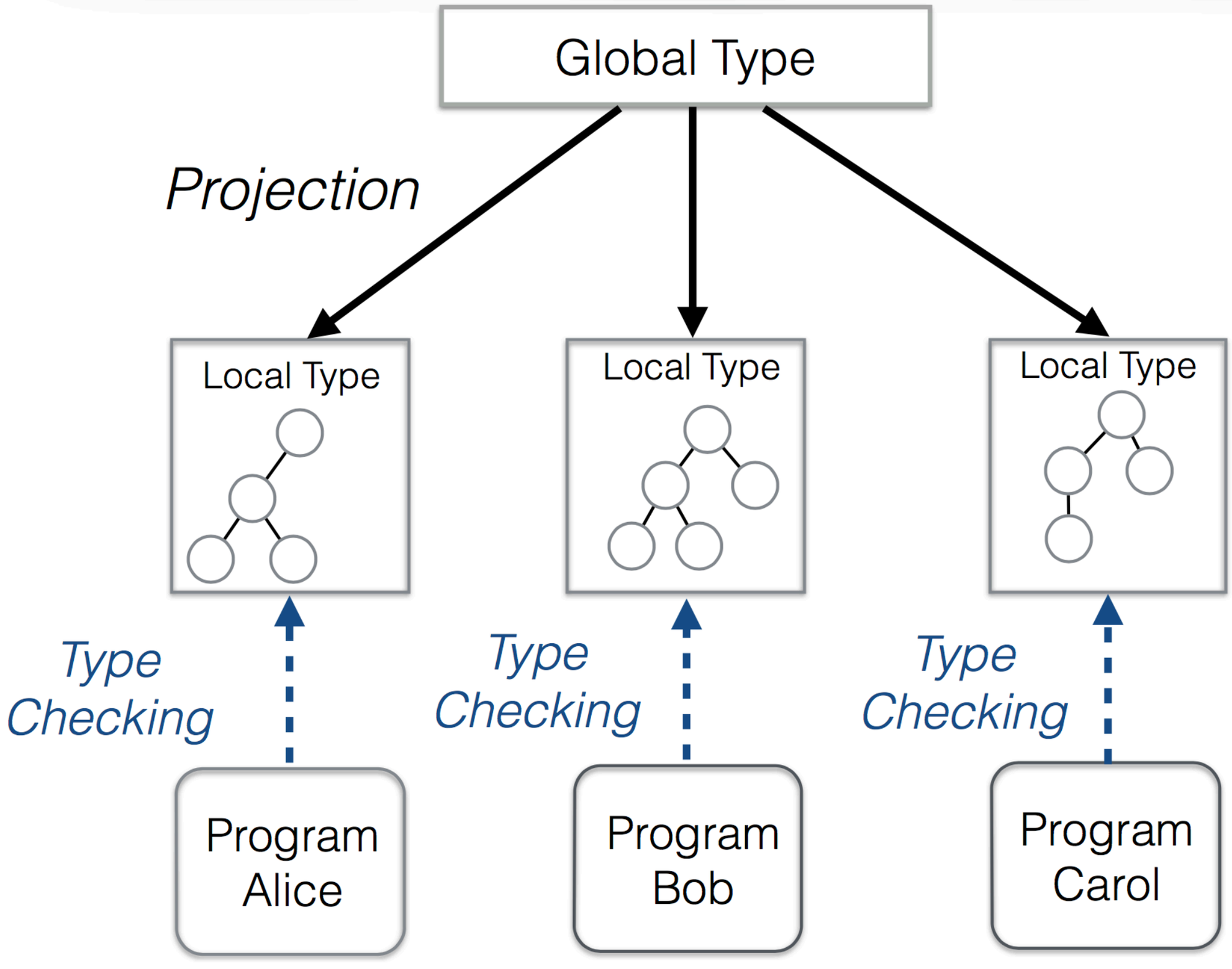
Dynamic Monitoring

[RV'13, COORDINATION'14, FMSD'15, LMCS'17, CC'17]



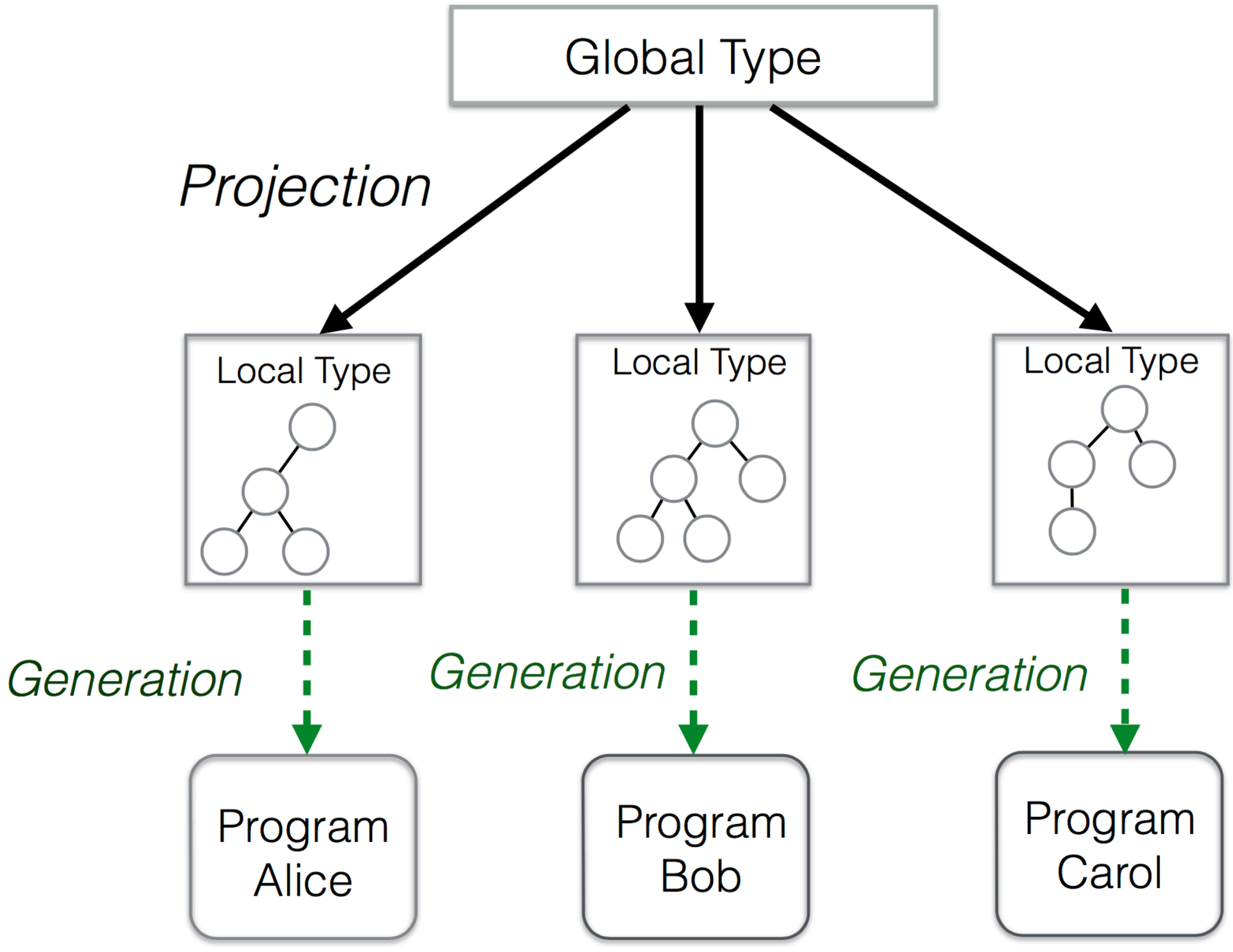
Type Checking

[OOPSLA'15, ECOOP'16, ECOOP'17, COORDINATION'17]



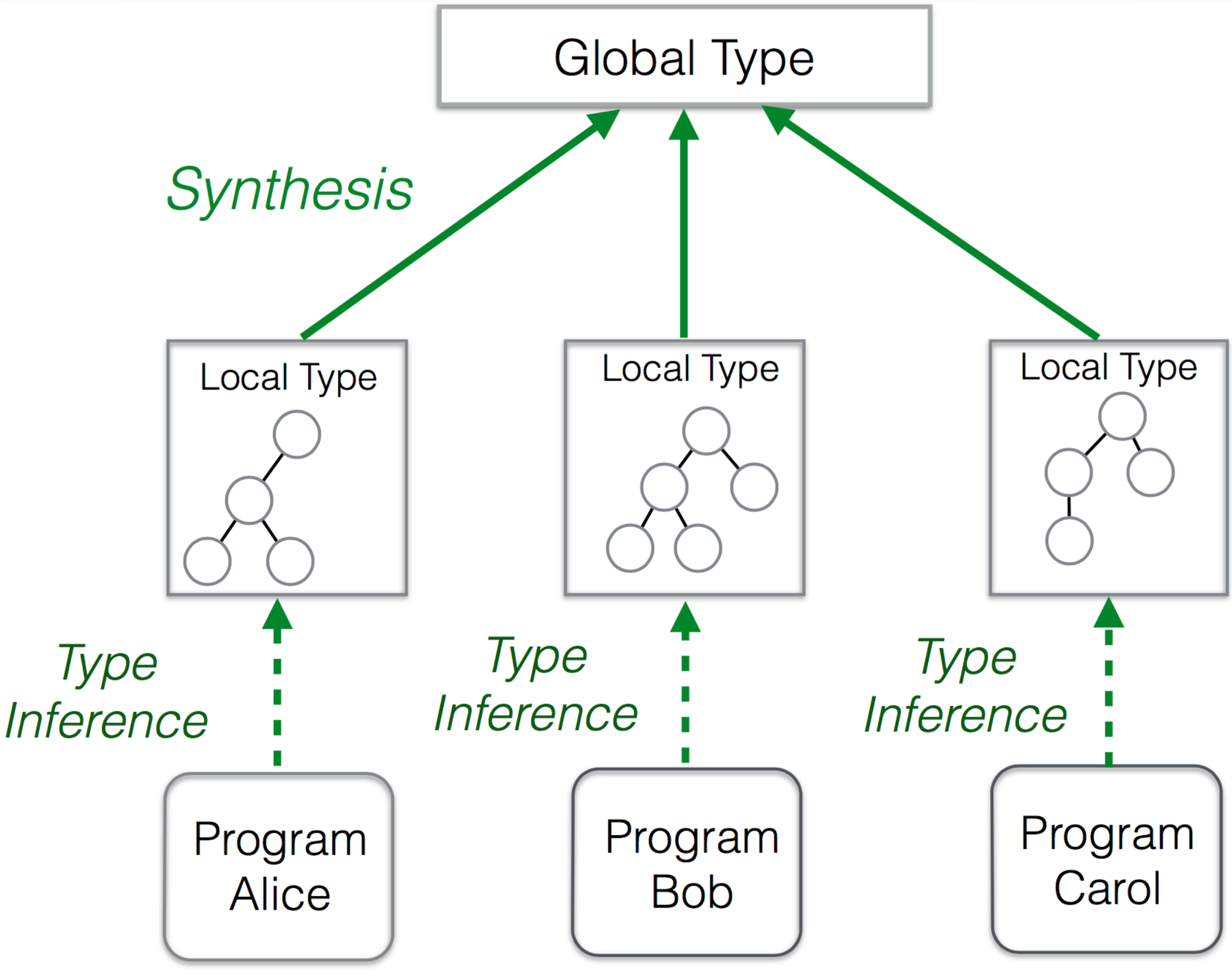
Code Generation

[CC'15, FASE'16, FASE'17]



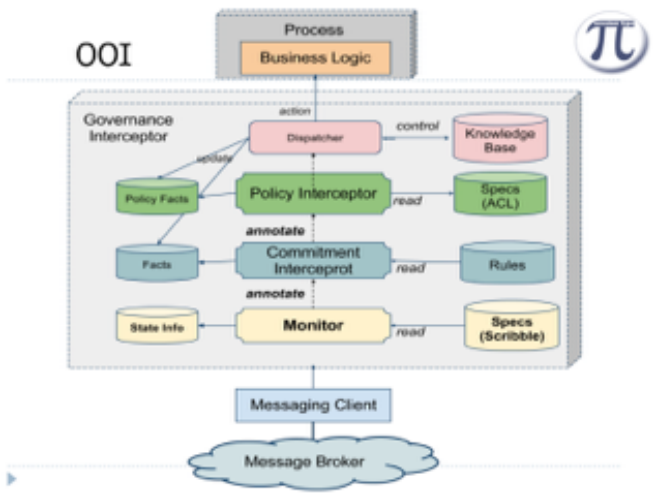
Synthesis

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

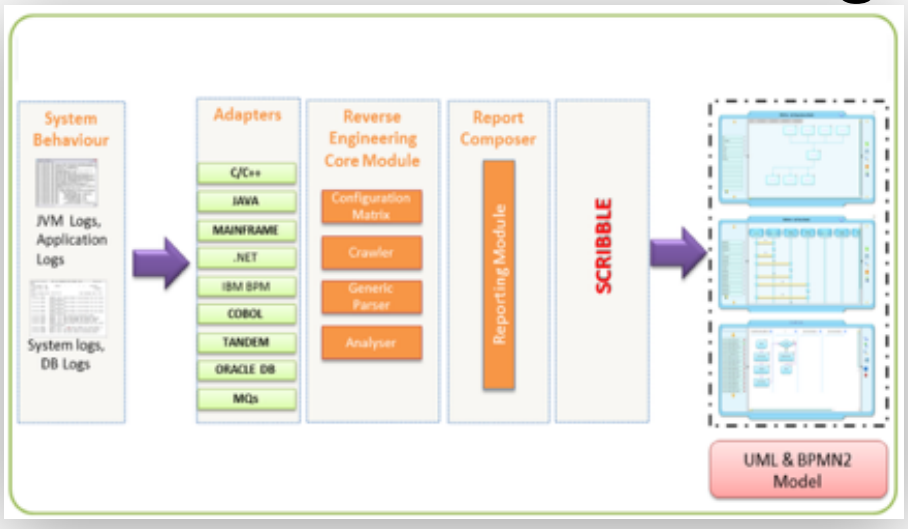


Session Type based Tools

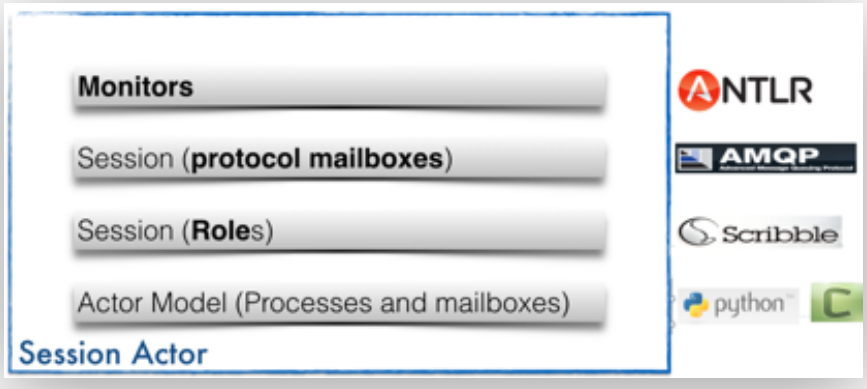
OOI Governance



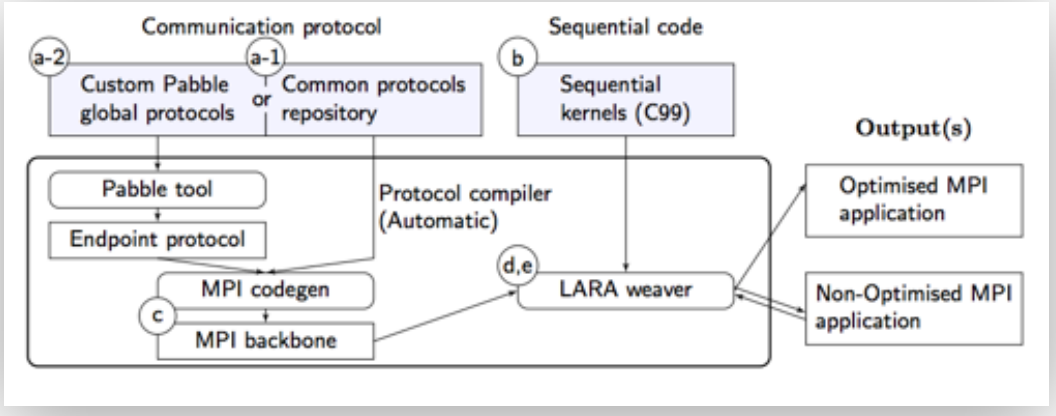
ZDLC: Process Modeling



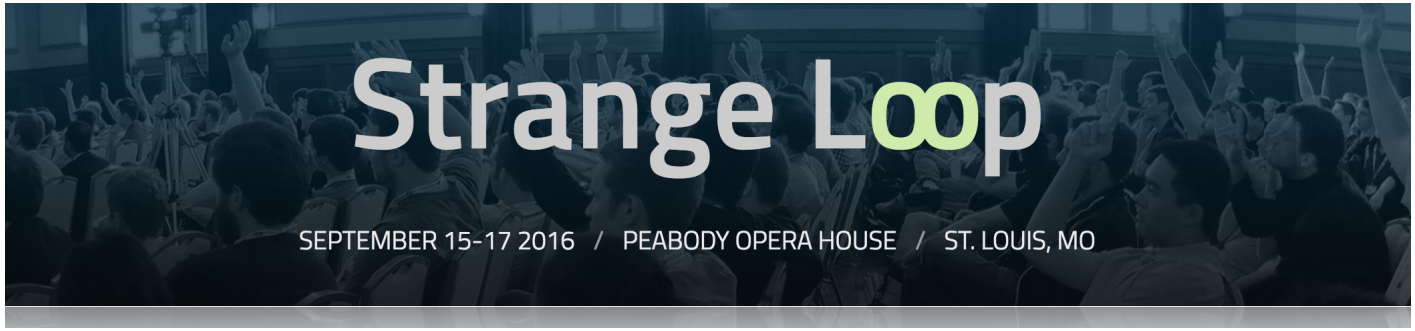
Actor Verification



MPI code generations



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

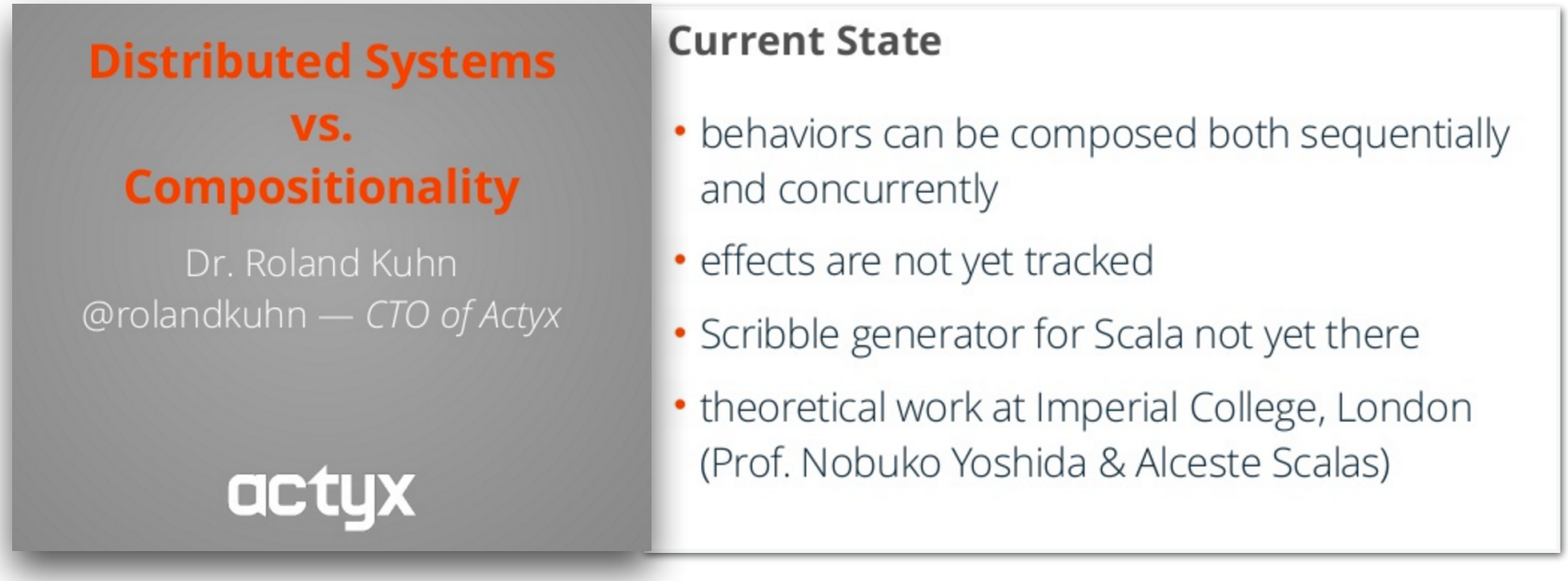
by Vicky Kapogianni
20 July 2016

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

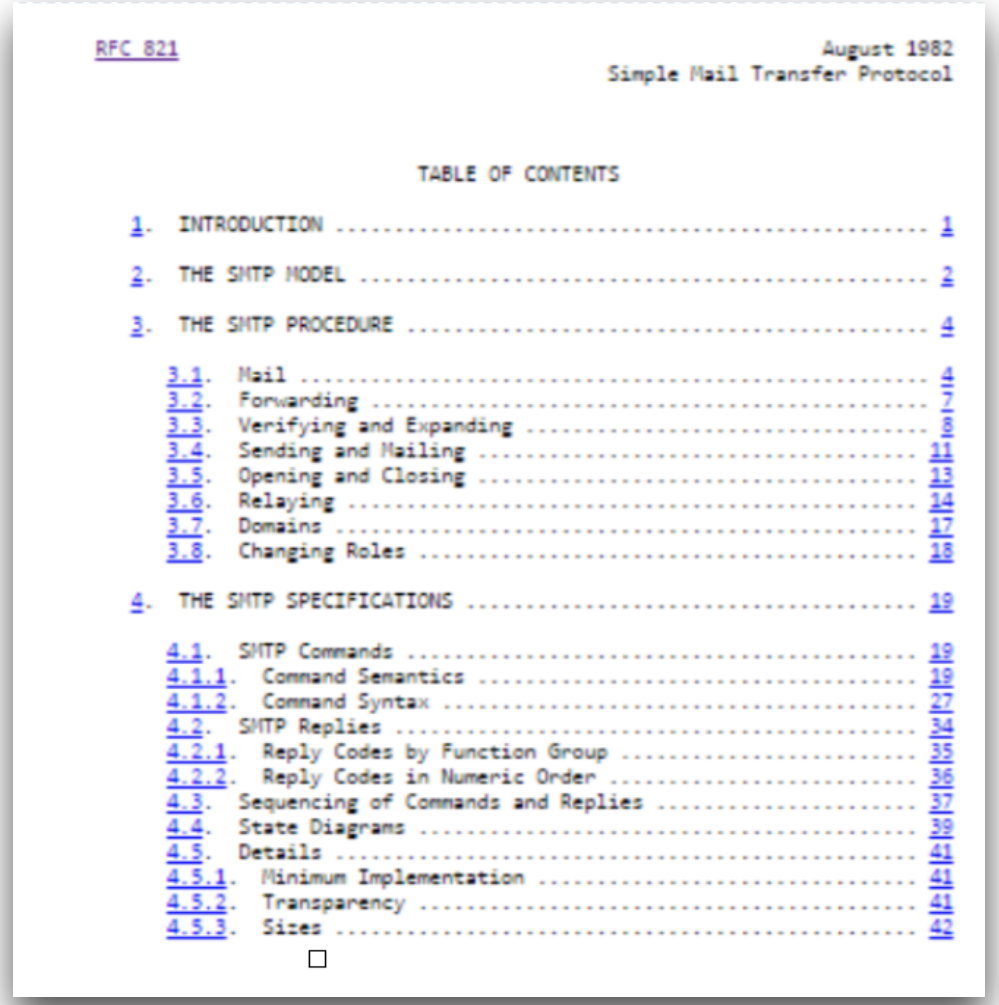
Click here to add content

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

Interactions with Industries

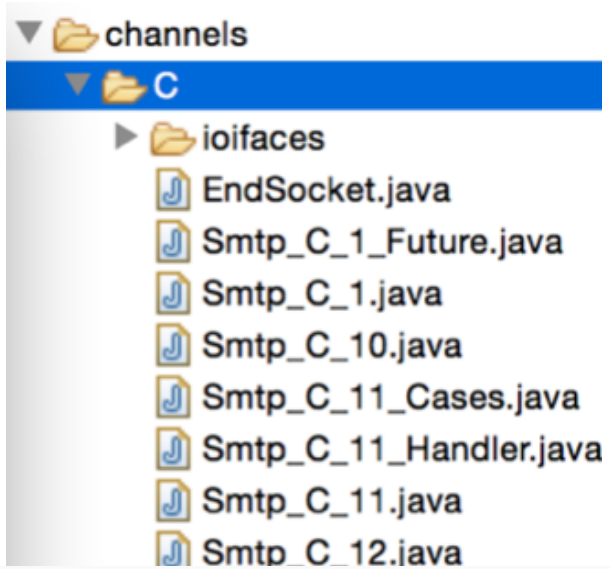


Java API Generation [FASE'16]



RFC 821 August 1982
Simple Mail Transfer Protocol

TABLE OF CONTENTS




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

send(S role, Mail m) : Smtp_C_11 - Smtp_C_10
send(S role, Quit m) : EndSocket - Smtp_C_10

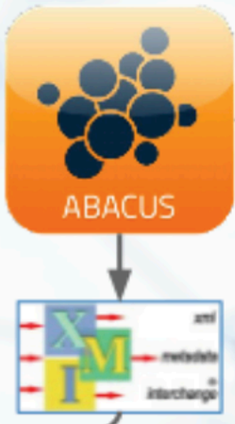
Scribble – Proving a distributed design

Estafet

Innovate | Deliver | Transform

1. All design work takes place in ABACUS, DCC's enterprise architecture tool. This can export standard XMI files (an open standard for UML5)

2. XMI is converted into OpenTracing format for consumption by managed service



7. Generate exception report and send back to DCC

3. OpenTracing files are combined to build a model in Scribble

4. Model holds *types* rather than *instances* to understand behaviour

5. Scribble compiler identifies inconsistency, change & design flaws

6. Issues highlighted graphically in Eclipse

www.estafet.com

Estafet Managed Service