Multiparty Session Types and their Applications

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

Nobuko Yoshida

Imperial College London

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

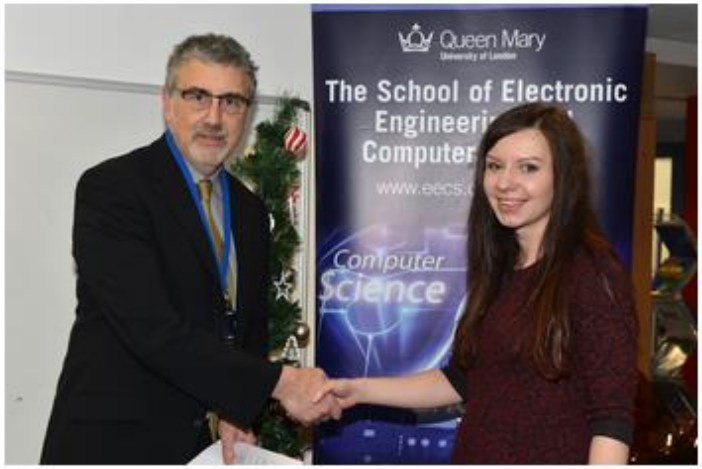
Posted with permission from QMUL on 17th Dec 2013. Original article written by Edmund Robinson.

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

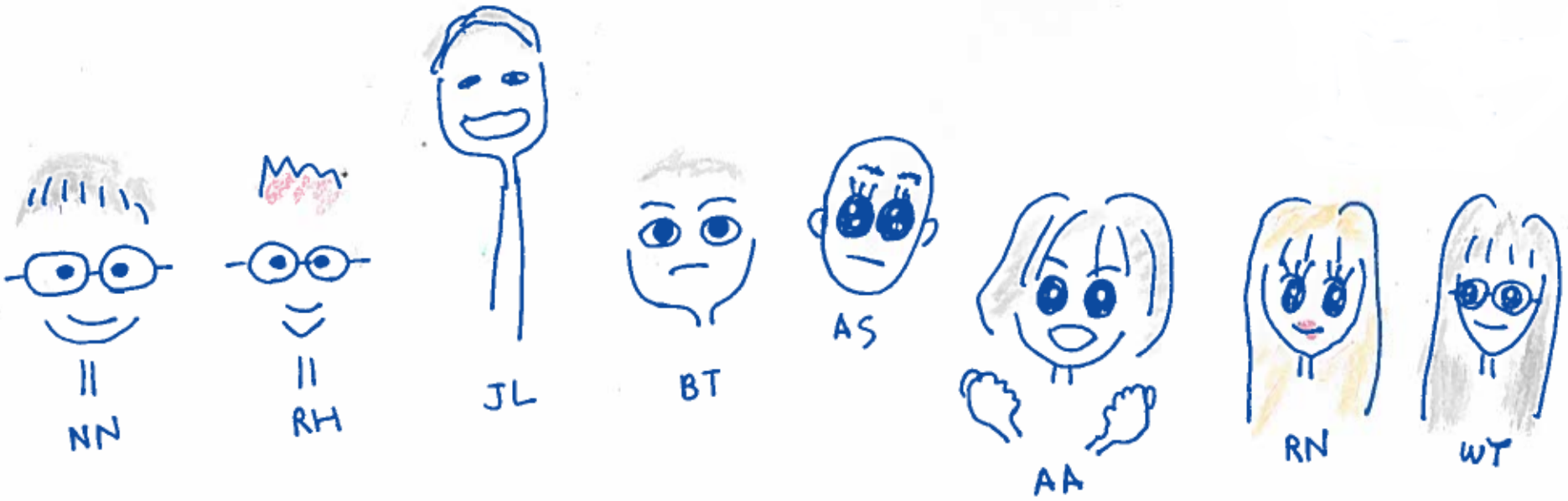
Winners 2013



Ms Anna Pawlicka
2013 winner (Undergraduate) source: QMUL

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

Session Type Mobility Group



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.

Current: Communication is Ubiquitous

- The way to organise software is increasingly based on communications (Cloud Computing, many cores, message-passing parallel computation, ...)
- Question
  - How to formally abstract/specify/implement/control communications?
  - How to apply mobile processes and their type theories to real distributed applications and programming languages?

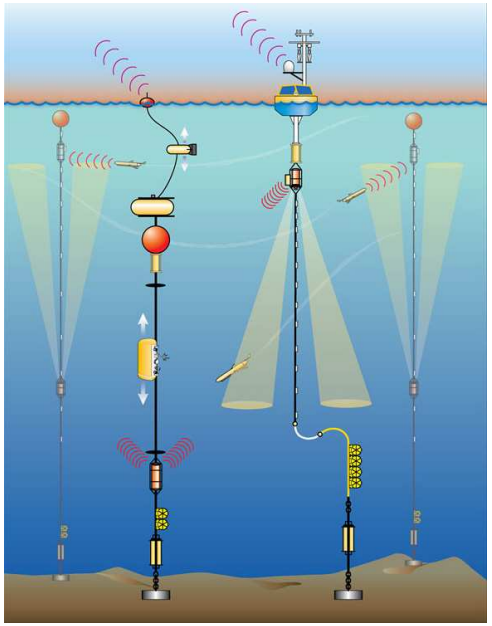
Current: Communication is Ubiquitous

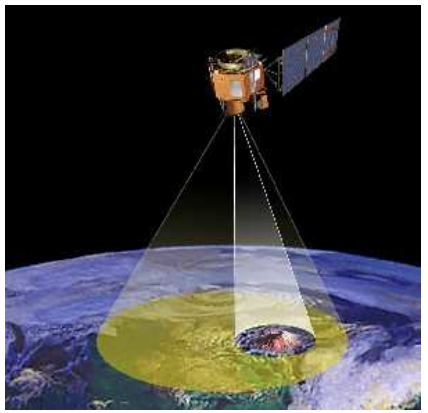
- The way to organise software is increasingly based on communications (Cloud Computing, many cores, message-passing parallel computation, ...)
- Question $\Longrightarrow$ Multiparty session type theory
  - How to formally abstract/specify/implement/control communications?
  - How to apply mobile processes and their type theories to real distributed applications and programming languages?

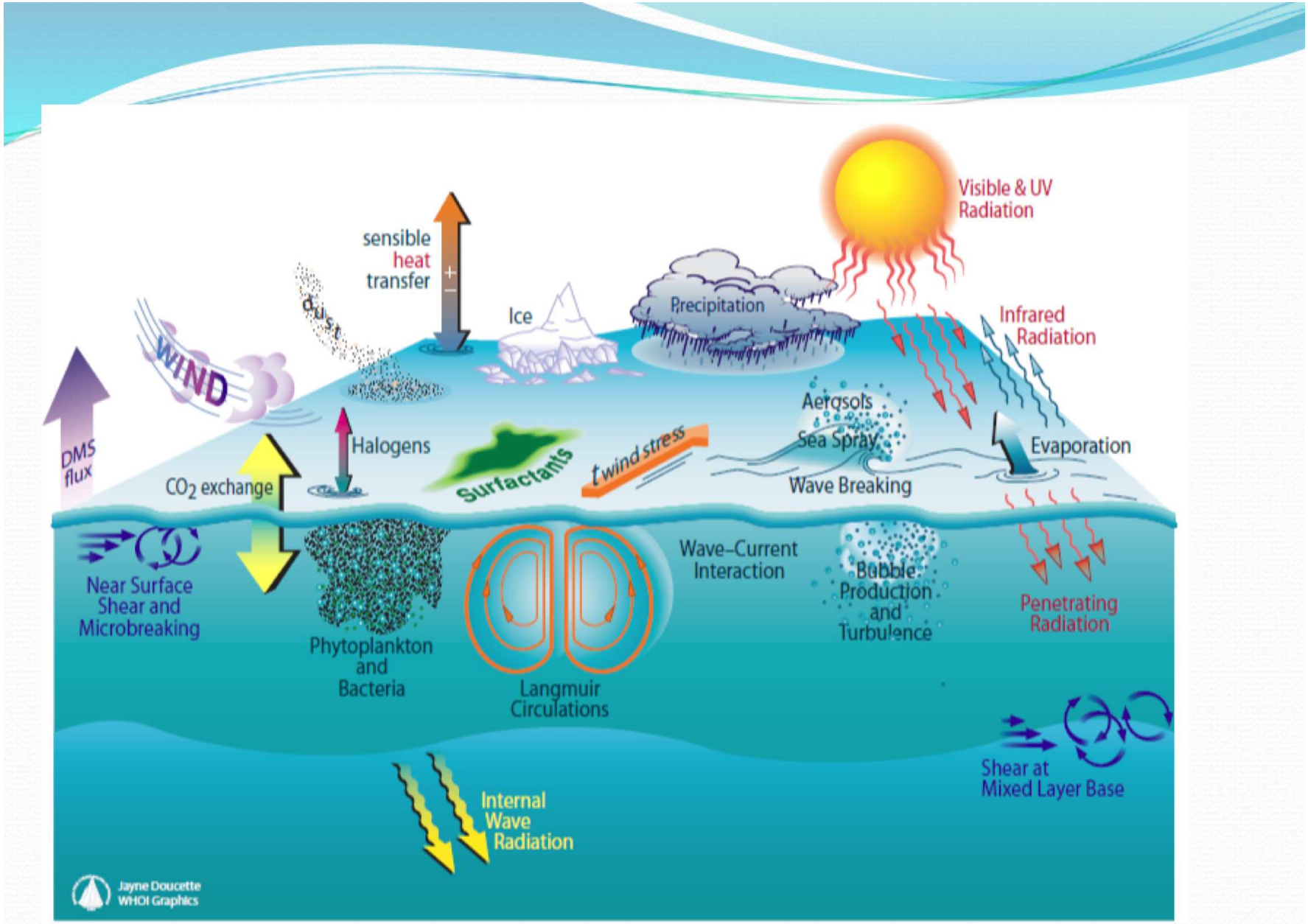
    $\Longrightarrow$ large-scale cyberinfrastructure for e-Science

Ocean Observatories Initiative

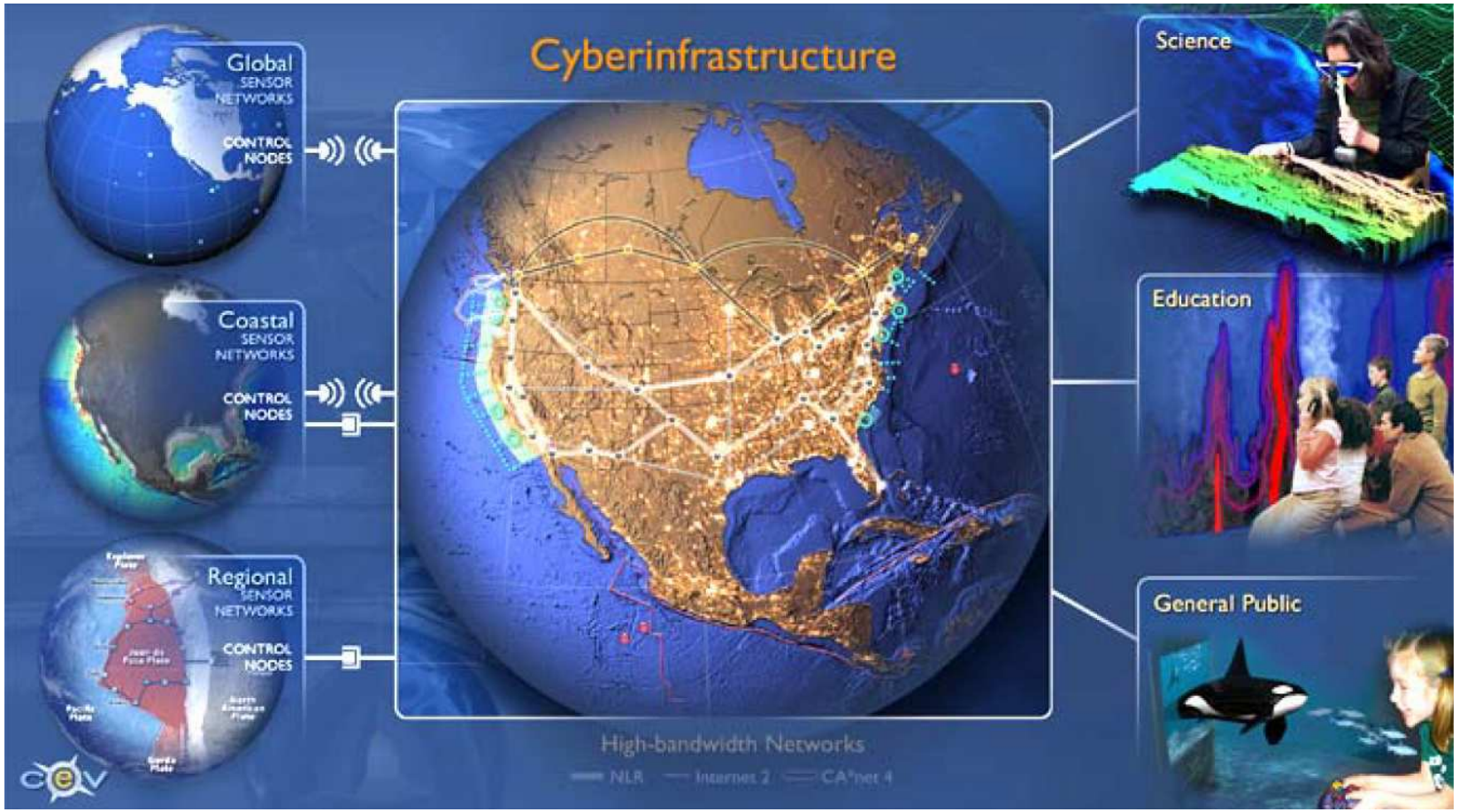
- A NSF project (400M$, 5 Years) to build a cyberinfrastructure for observing oceans around US and beyond.
- Real-time sensor data constantly coming from both off-shore and on-shore (e.g. buoys, submarines, under-water cameras, satellites), transmitted via high-speed networks.

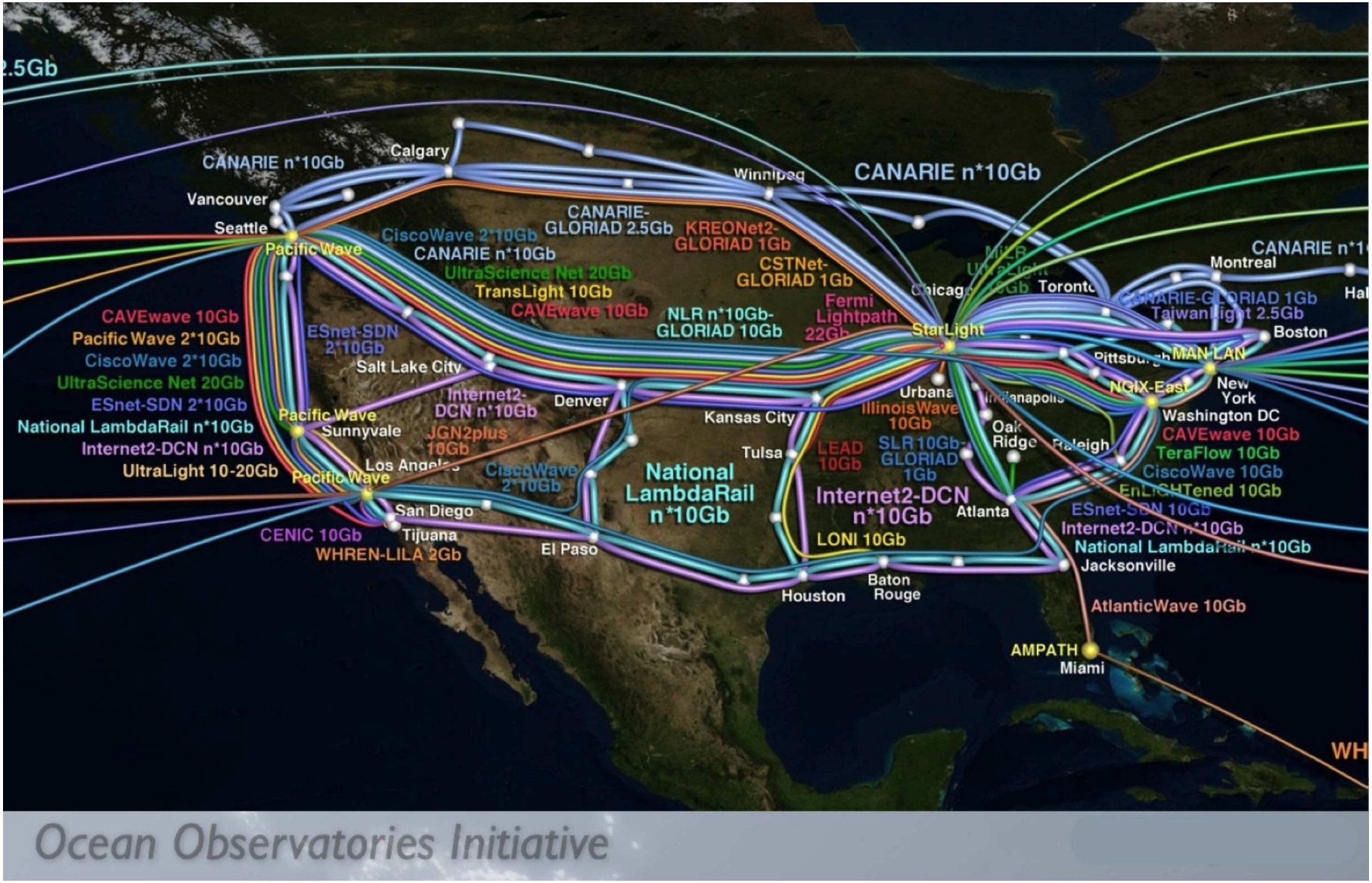





Visible & UV Radiation
sensible heat transfer
+
−
Ice
Precipitation
Infrared Radiation
WIND
dust
DMS flux
Halogens
Aerosols
Sea Spray
Evaporation
CO2 exchange
Surfactants
t wind stress
Wave Breaking
Near Surface Shear and Microbreaking
Wave–Current Interaction
Bubble Production and Turbulence
Penetrating Radiation
Phytoplankton and Bacteria
Langmuir Circulations
Shear at Mixed Layer Base
Internal Wave Radiation
Jayne Doucette
WHOI Graphics

Ocean Observatories Initiative





Ocean Observatories Initiative

Challenges

- The need to specify, catalogue, program, implement and manage *multiparty message passing protocols*.
- Communication assurance
  - Correct message ordering and synchronisation
  - Deadlock-freedom, progress and liveness
  - Dynamic message monitoring and recovery
  - Logical constraints on message values
- Shared and used over a long-term period (e.g. 30 years in OOI).

Why Multiparty Session Types?

- Robin Milner (2002): *Types are the leaven of computer programming; they make it digestible*.

  $\Longrightarrow$ Can describe communication protocols as *types*

  $\Longrightarrow$ Can be materialised as *new communications programming languages* and *tool chains*.

- *Scalable* automatic verifications (deadlock-freedom, safety and liveness) without *state-space explosion problems* (*polynomial time complexity*).

- Extendable to *logical verifications* and flexible *dynamic monitoring*.

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

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]

⇓

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/

```
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

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

Distributed Systems vs. Compositionality

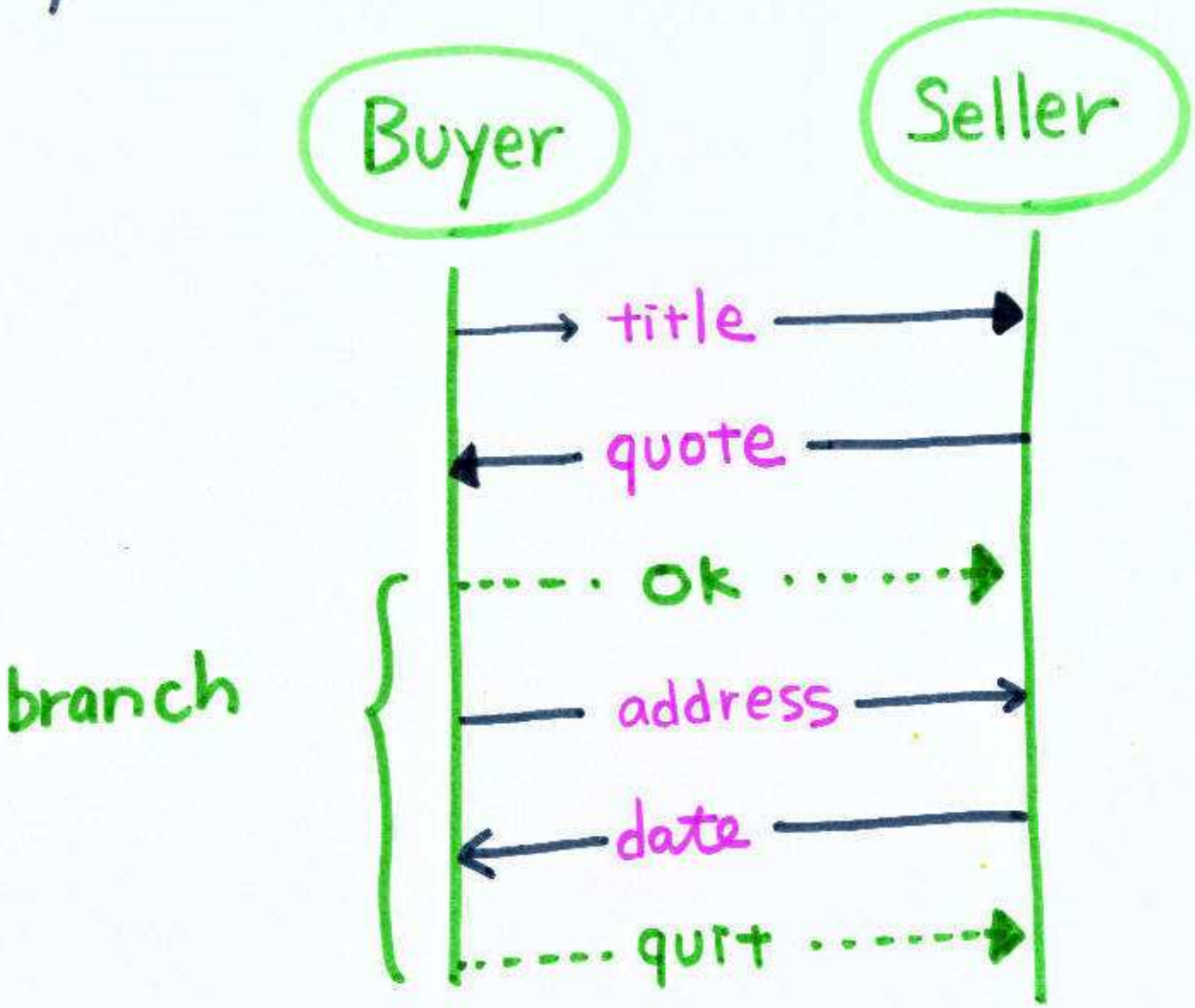
Dr. Roland Kuhn
@rolandkuhn — *CTO of Actyx*

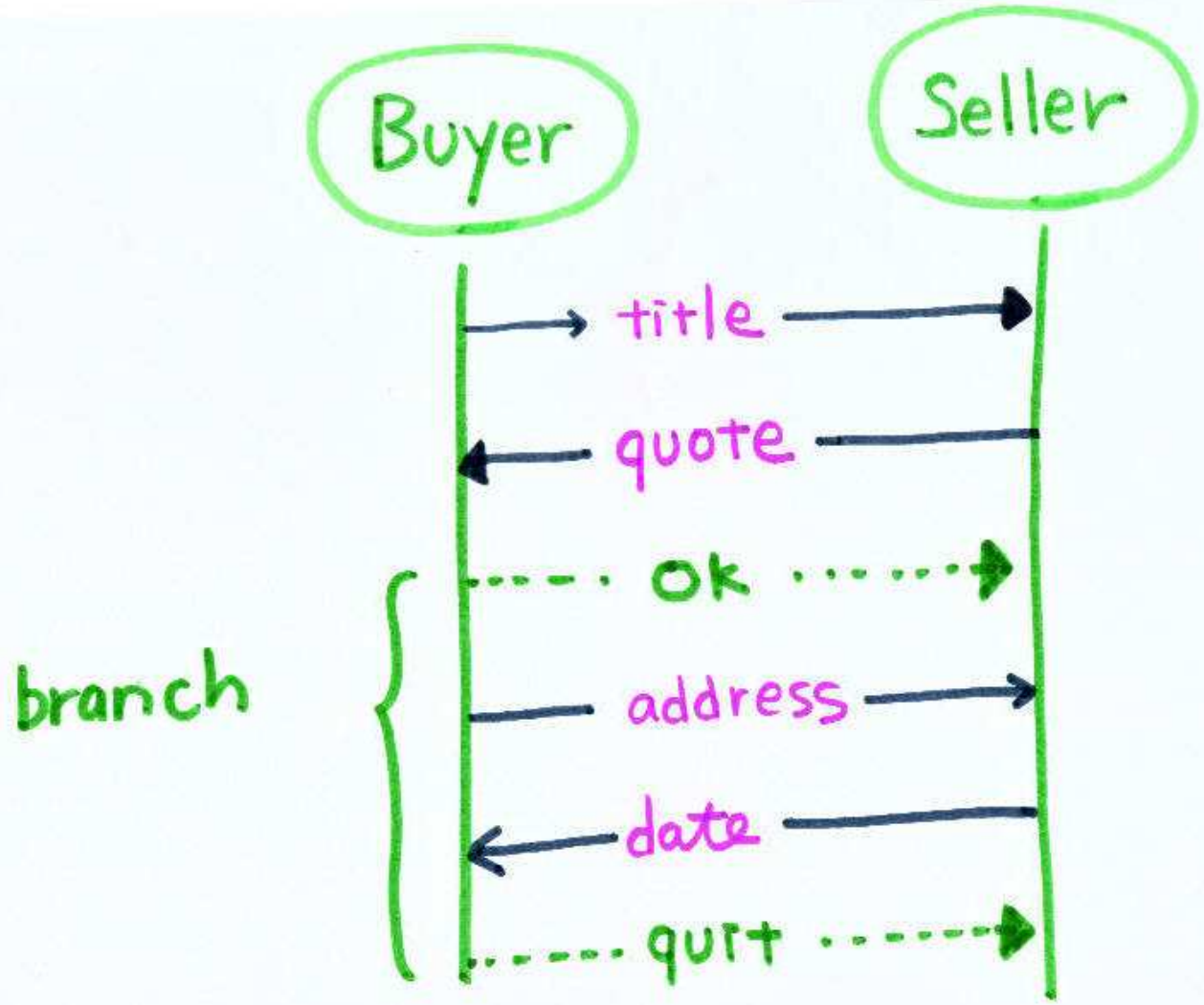
actyx

Current State

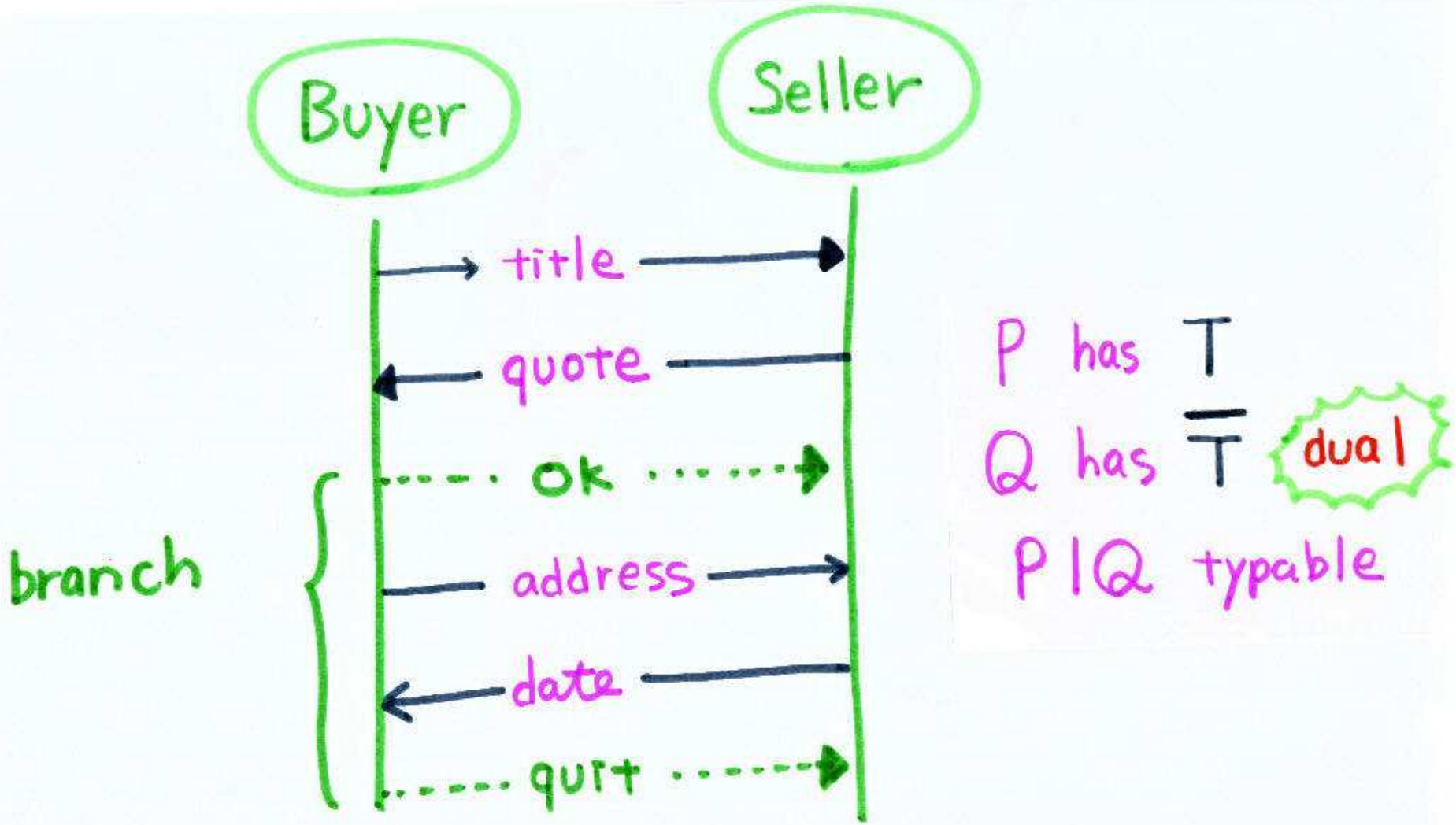
- behaviors can be composed both sequentially and concurrently
- effects are not yet tracked
- Scribble generator for Scala not yet there
- theoretical work at Imperial College, London (Prof. Nobuko Yoshida & Alceste Scalas)

Binary Session Types : Buyer - Seller Protocol





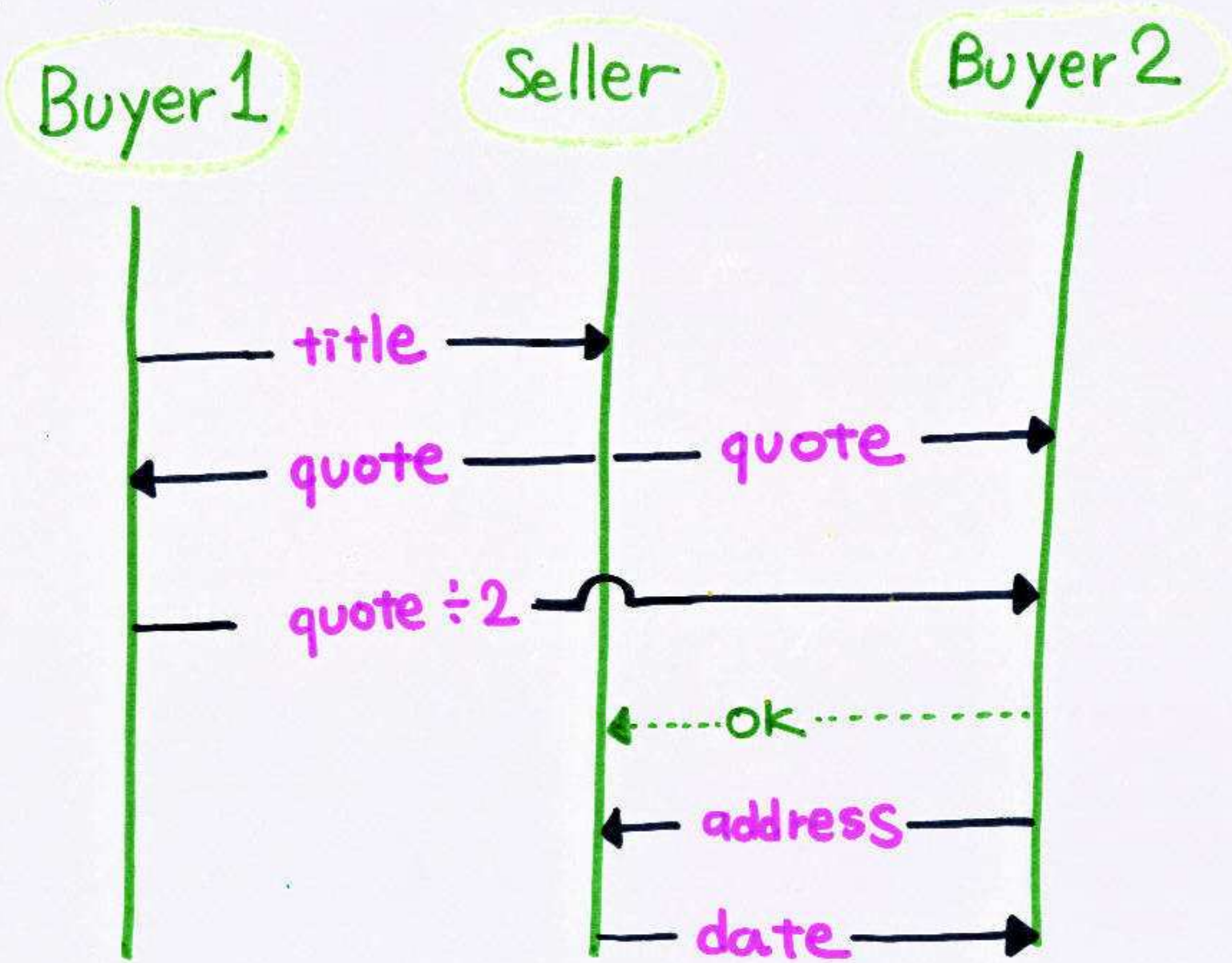
! String ; ? Int ; ⊕ { ok : ! String ; ? Date ; end , quit : end }

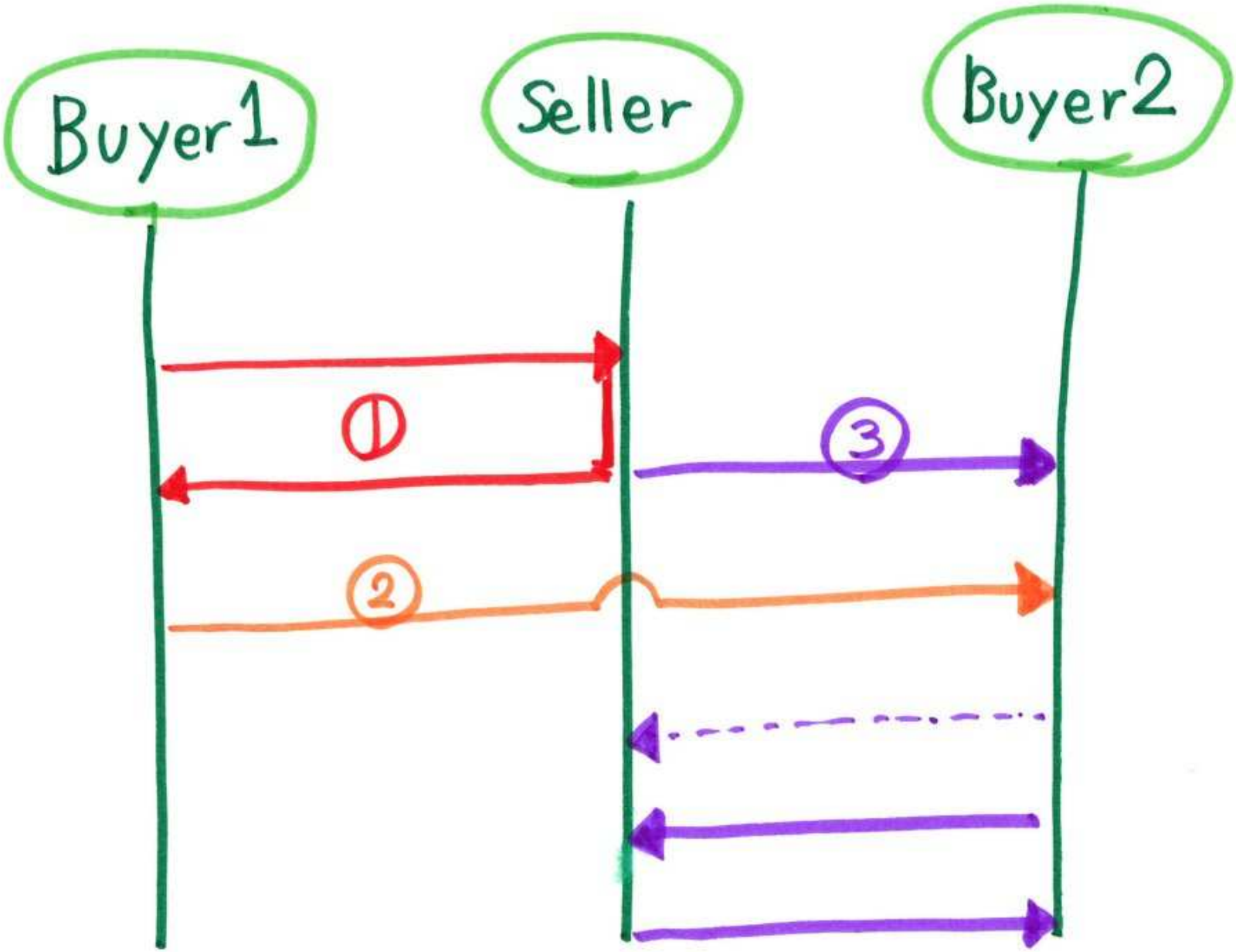


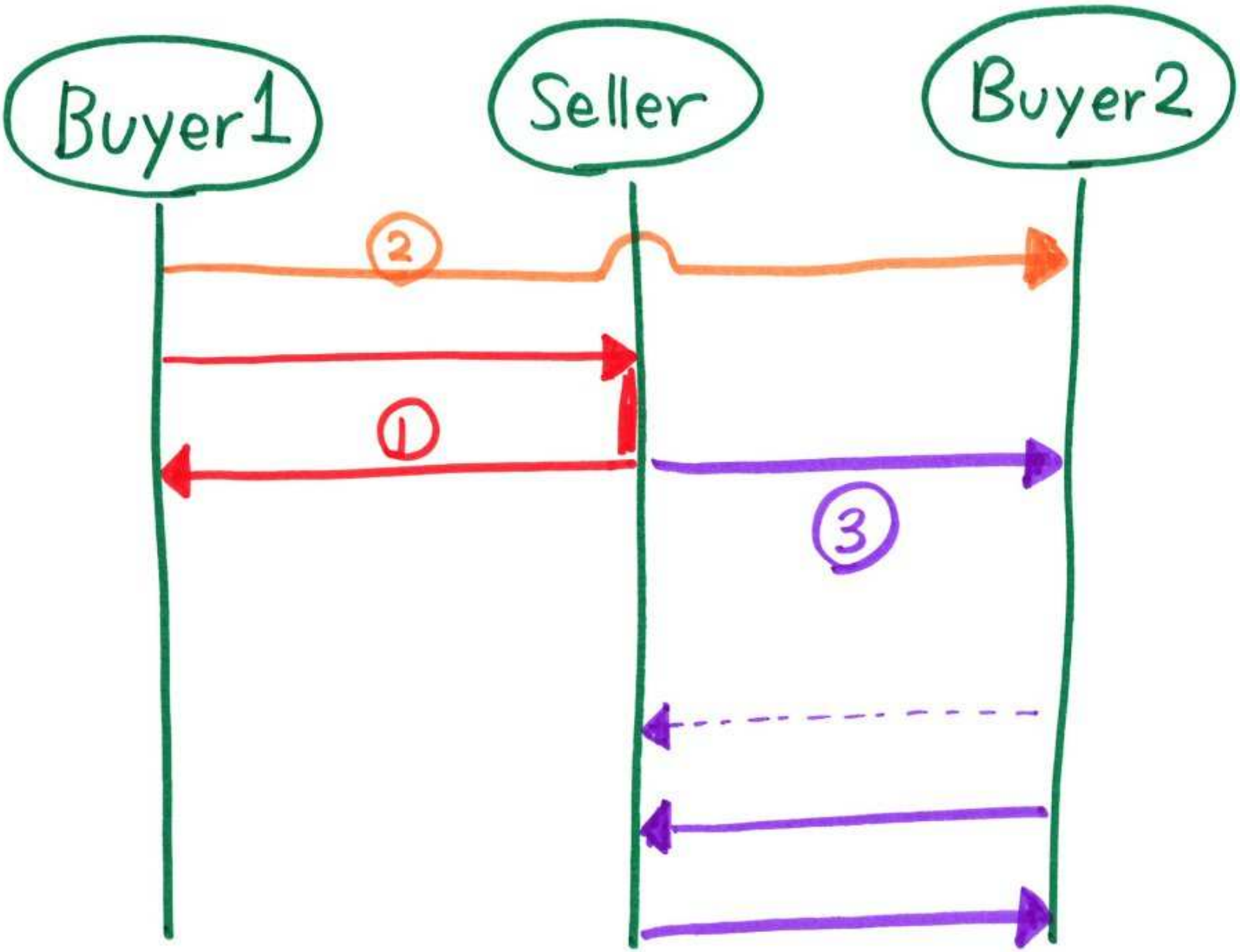
! String ; ? Int ; ⊕ { ok : ! String ; ? Date ; end , quit : end }

dual ? String ; ! Int ; & { ok : ? String ; ! Date ; end , quit : end }

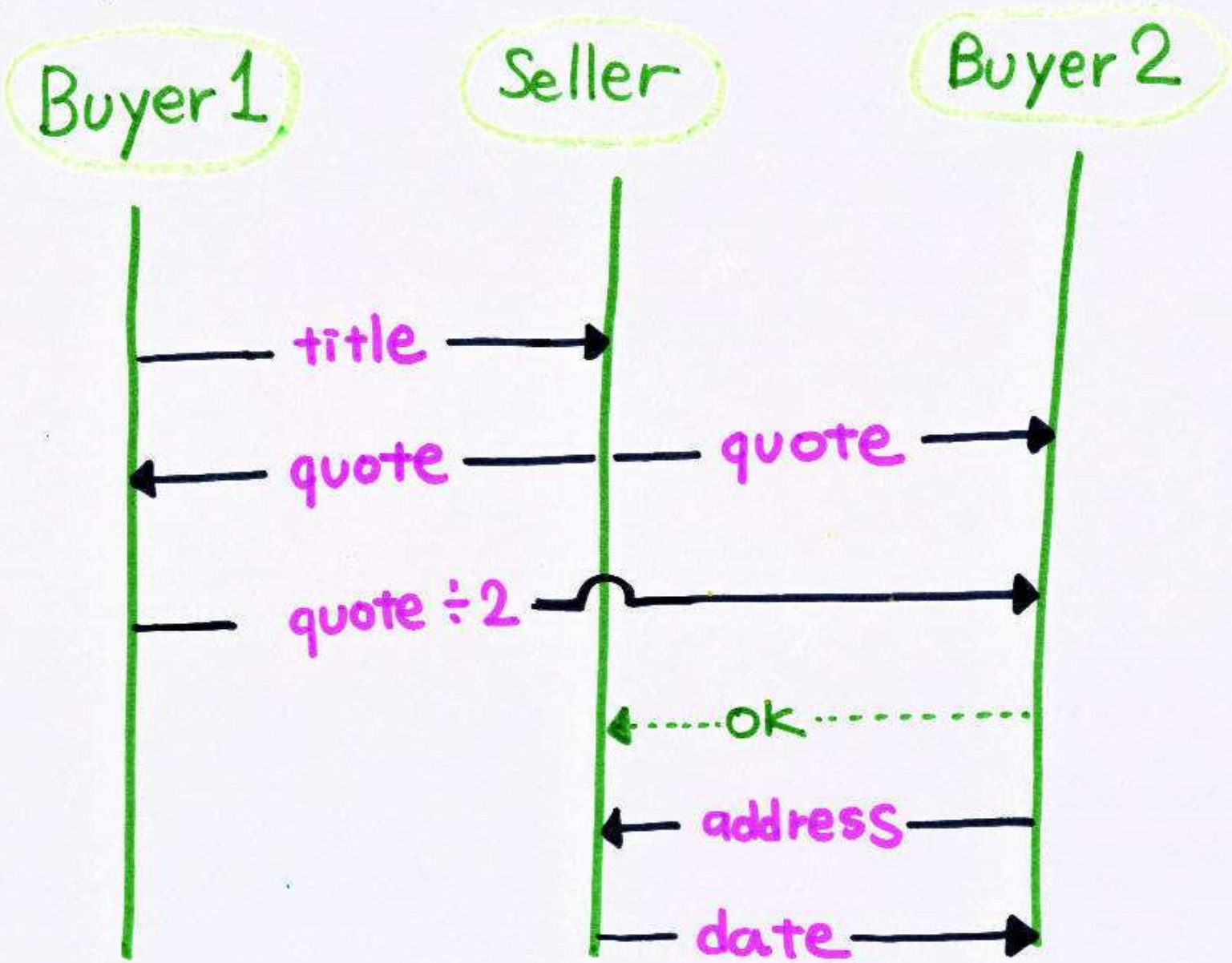
Multiparty Session Types




Buyer1
Seller
Buyer2
1
3
2

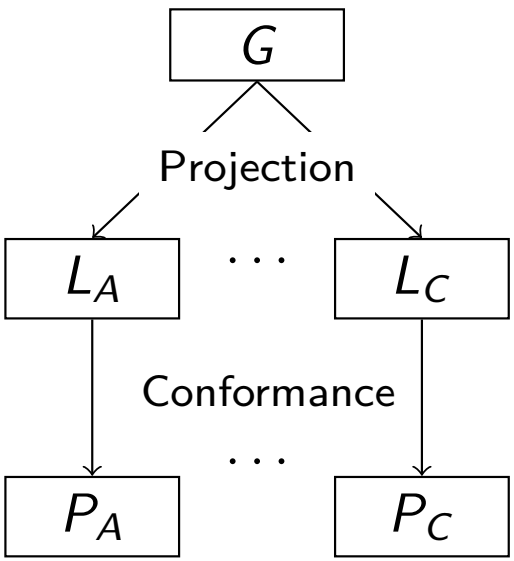

Buyer1
Seller
Buyer2
2
1
3

Multiparty Session Types



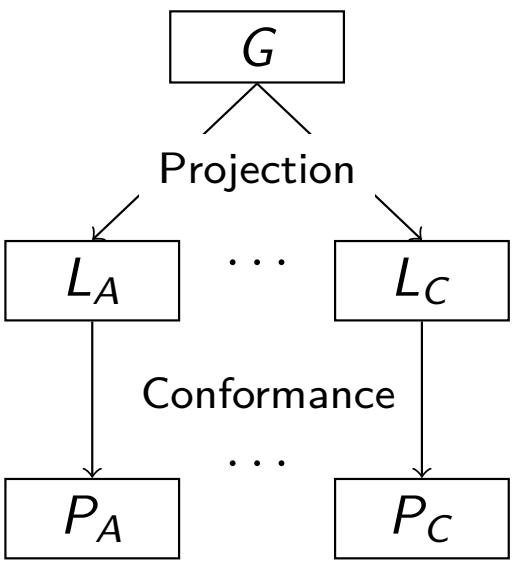
Multiparty session types

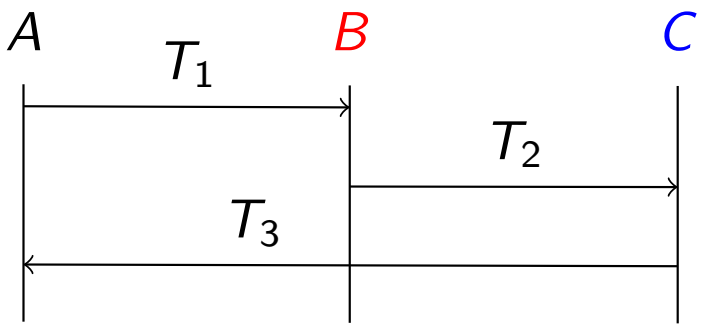
- Global Types and End Point Projection (Abstract Choreography)
  - Potential errors:
    - × Communication mismatch: e.g. receiver is sent an unexpected message
    - × Protocol violation: executed interaction does not follow the protocol
    - × Deadlock: e.g. all endpoints blocked on input



Multiparty session types

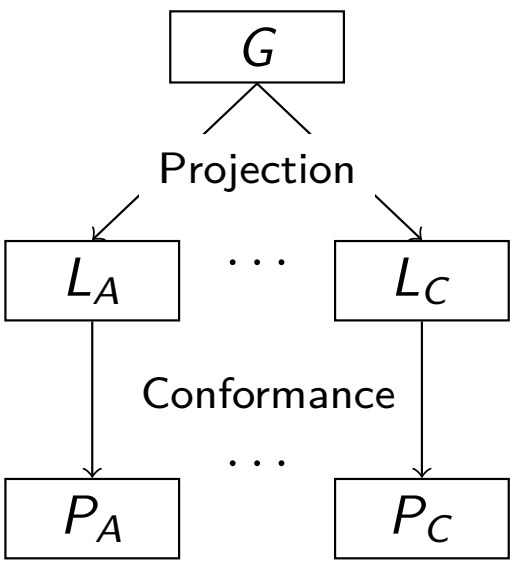
- Global Types and End Point Projection (Abstract Choreography)
  - Potential errors:
    - × Communication mismatch: e.g. receiver is sent an unexpected message
    - × Protocol violation: executed interaction does not follow the protocol
    - × Deadlock: e.g. all endpoints blocked on input

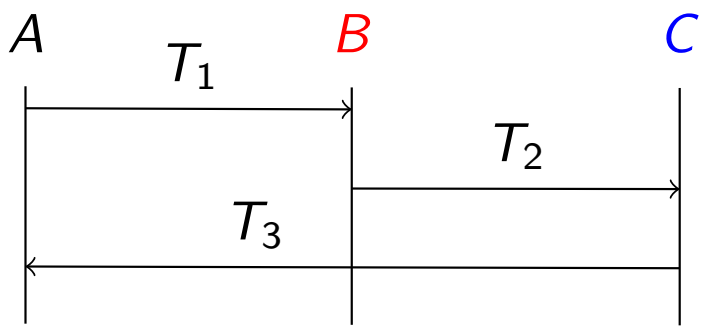




Multiparty session types

- Global Types and End Point Projection (Abstract Choreography)
  - Potential errors:
    - × Communication mismatch: e.g. receiver is sent an unexpected message
    - × Protocol violation: executed interaction does not follow the protocol
    - × Deadlock: e.g. all endpoints blocked on input

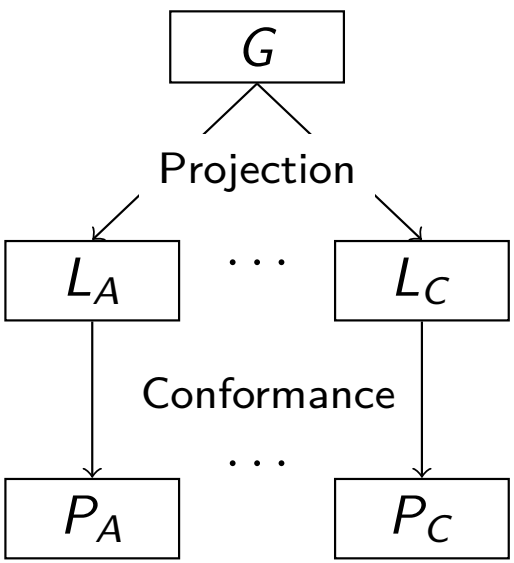


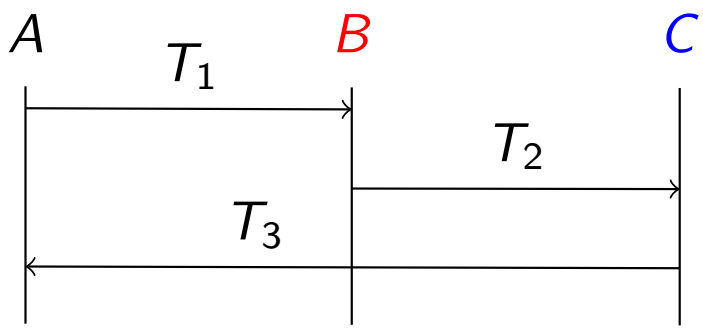


$$G = A \to B : T_1.$$
$$B \to C : T_2.$$
$$C \to A : T_3$$

Multiparty session types

- Global Types and End Point Projection (Abstract Choreography)
  - Potential errors:
    - × Communication mismatch: e.g. receiver is sent an unexpected message
    - × Protocol violation: executed interaction does not follow the protocol
    - × Deadlock: e.g. all endpoints blocked on input





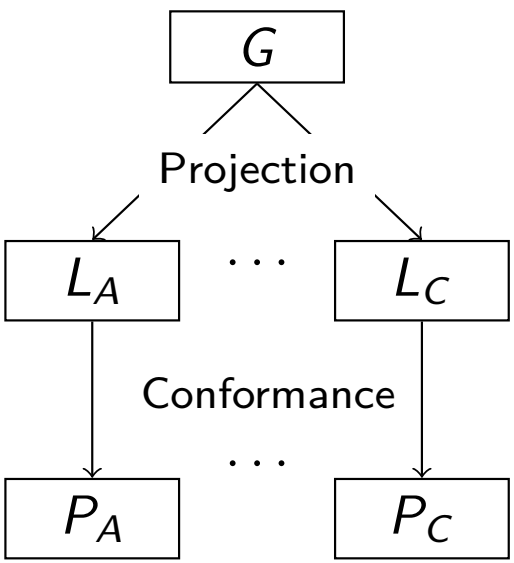
$$G = A \to B : T_1.\ B \to C : T_2.\ C \to A : T_3$$

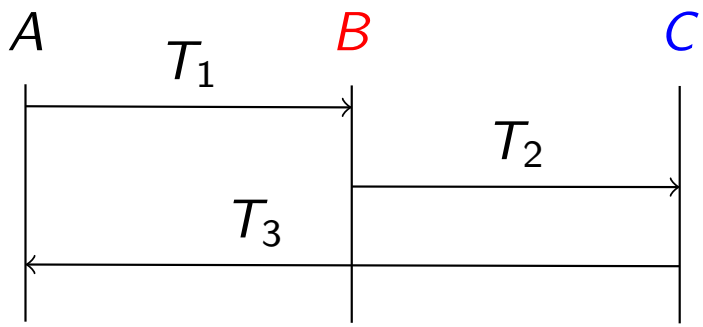
Projection (Good)

$A :!\langle B, T_1\rangle.?(C, T_3)$
$B :?(A, T_1).!\langle C, T_2\rangle$
$C :?(B, T_2).!\langle A, T_3\rangle$

Multiparty session types

- Global Types and End Point Projection (Abstract Choreography)
  - Potential errors:
    - × Communication mismatch: e.g. receiver is sent an unexpected message
    - × Protocol violation: executed interaction does not follow the protocol
    - × Deadlock: e.g. all endpoints blocked on input





$$G = A \to B : T_1.\ B \to C : T_2.\ C \to A : T_3$$

Projection (Good)

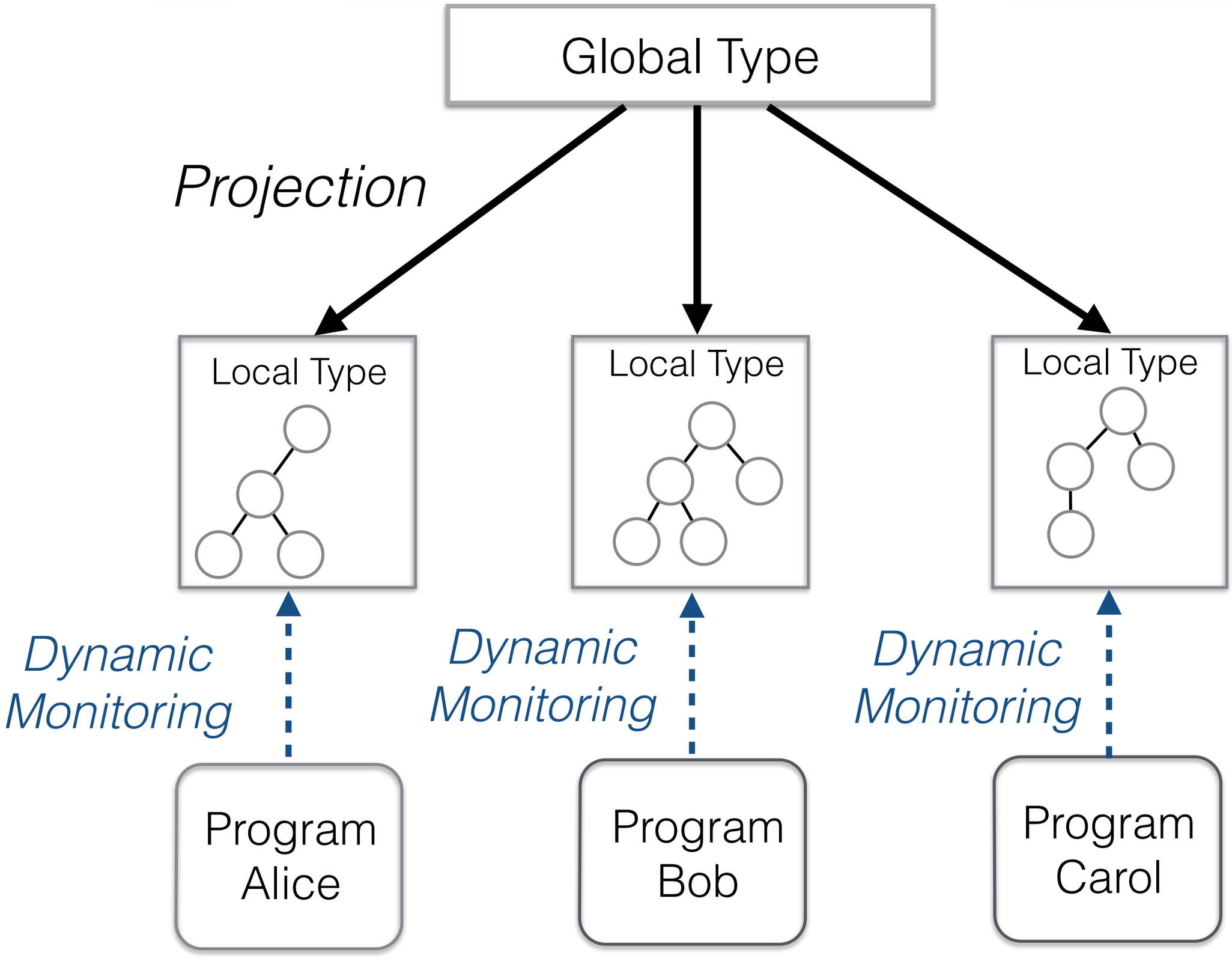
$A :!\langle B, T_1\rangle.?(C, T_3)$
$B :?(A, T_1).!\langle C, T_2\rangle$
$C :?(B, T_2).!\langle A, T_3\rangle$

Bad (Deadlock)

$A :?(C, T_3).!\langle B, T_1\rangle$
$B :?(A, T_1).!\langle C, T_2\rangle$
$C :?(B, T_2).!\langle A, T_3\rangle$

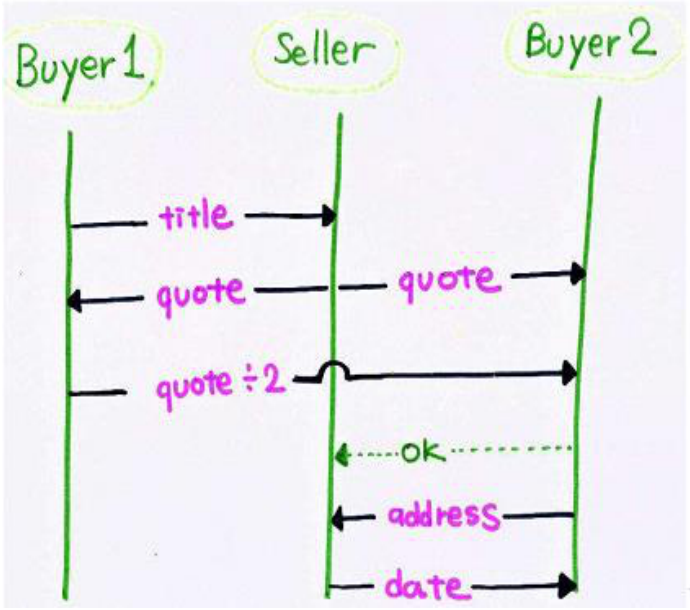
Dynamic Monitoring

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

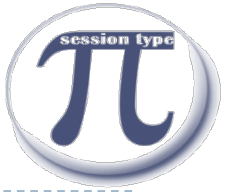


Two Buyer Protocol in Scribble

```
type <java> "java.lang.String" from "rt.jar" as String

global protocol TwoBuyers(role A, role B, role S) {
    title(String) from A to S;
    quote(Integer) from S to A, B;
    rec LOOP {
        share(Integer) from A to B;
        choice at B {
            accept(address:String) from B to A,
            date(String) from S to B;
        } or {
            retry() from B to A, S;
            continue LOOP;
        } or {
            quit() from B to A, S;
}   }   }
```

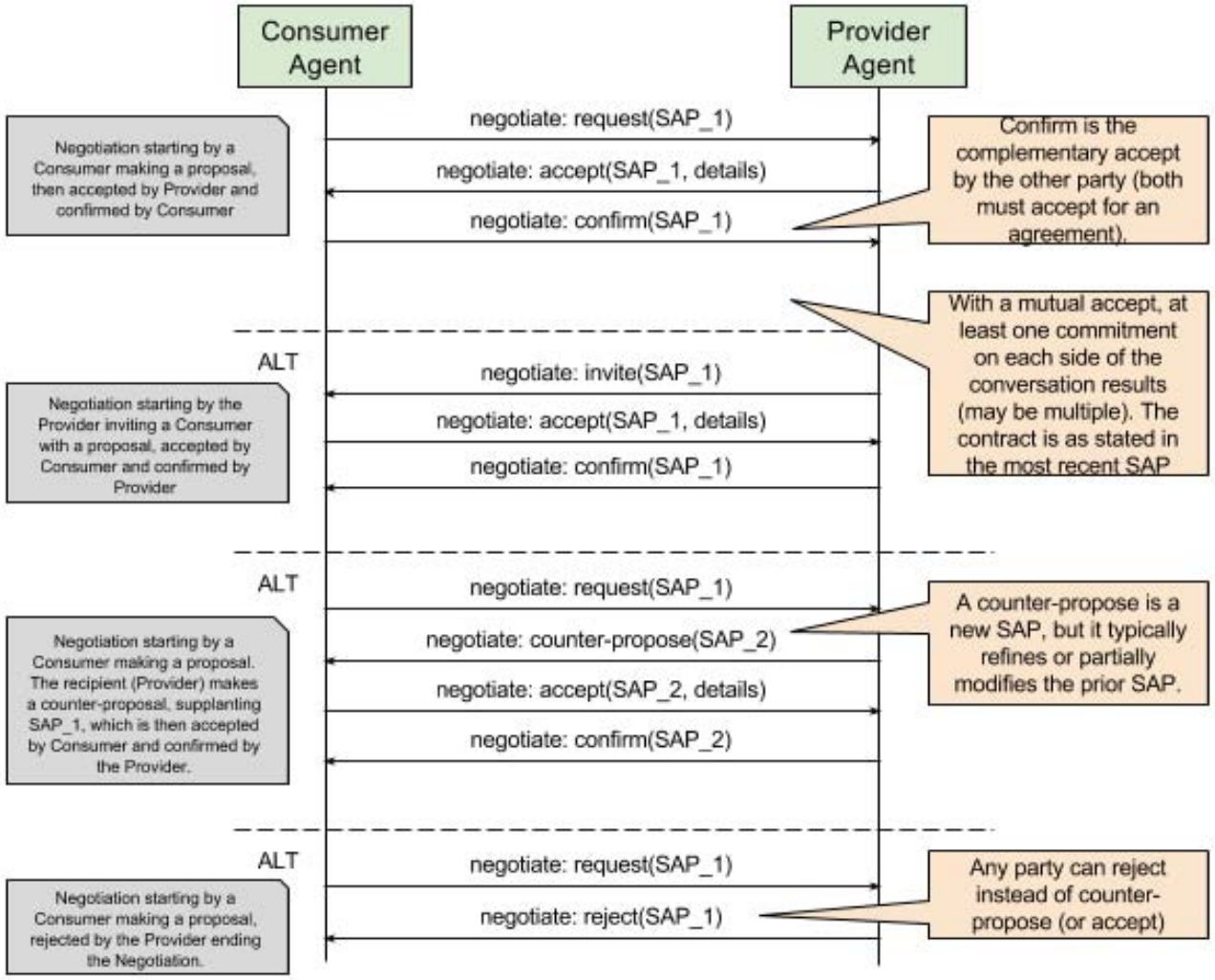


Buyer: A local projection




```
module Bookstore_TwoBuyers_A;

type <java> "java.lang.Integer" from "rt.jar" as Integer;
type <java> "java.lang.String" from "rt.jar" as String;

local protocol TwoBuyers_A at A(role A, role B, role S) {
 title(String) to S;
 quote(Integer) from S;
 rec LOOP {
  share(Integer) to B;
  choice at B {
   accept(address:String) from B;
  } or {
   retry() from B;
   continue LOOP;
  } or {
   quit() from B;
} } }
```

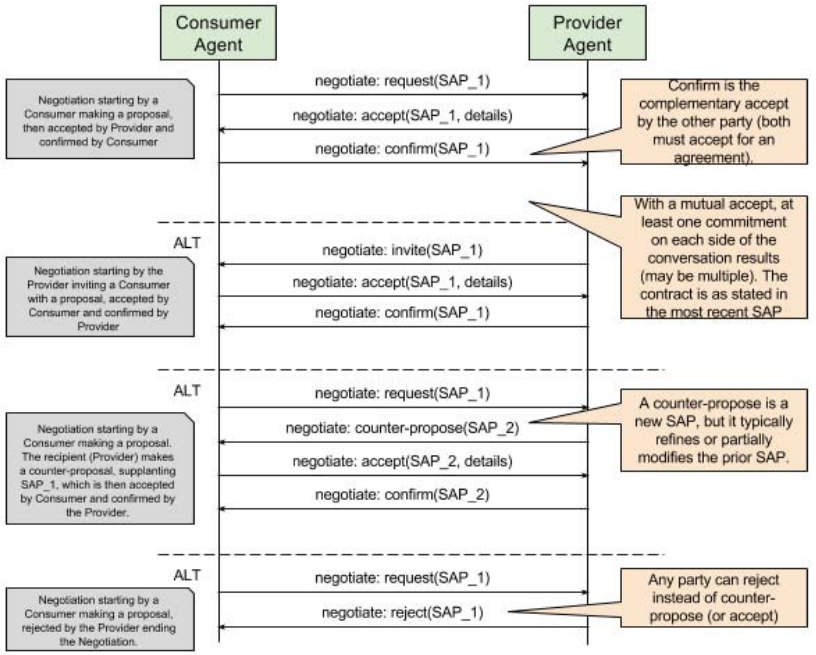
OOI agent negotiation 1/5



- `https://confluence.oceanobservatories.org/display/syseng/CIAD+COI+OV+Negotiate+Protocol`

OOI agent negotiation 2/5

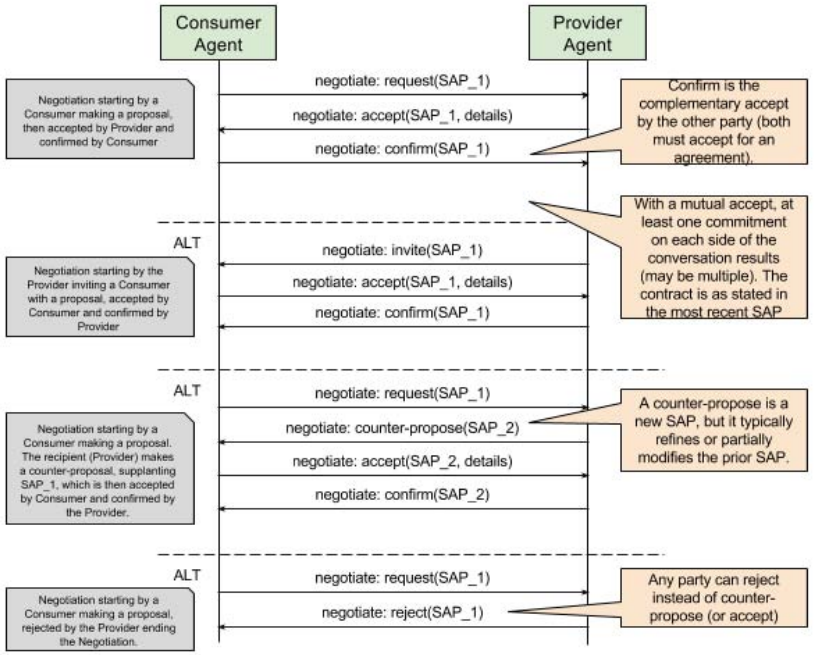
```
type <yml> "SAPDoc1" from "SAPDoc1.yml" as SAP;

global protocol Negotiate(role Consumer as C, role Producer as P) {
```




```
}
```

OOI agent negotiation 3/5 (choice)

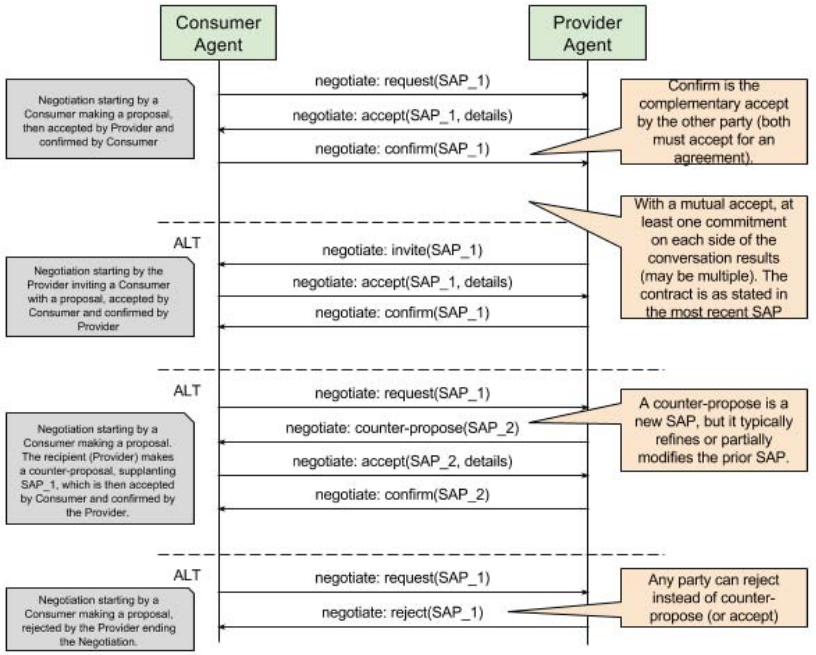
```
type <yml> "SAPDoc1" from "SAPDoc1.yml" as SAP;

global protocol Negotiate(role Consumer as C, role Producer as P) {
  propose(SAP) from C to P;

    choice at P {
      accept() from P to C;
      confirm() from C to P;
    } or {
      reject() from P to C;
    } or {
      propose(SAP) from P to C;
```




```
} }
```

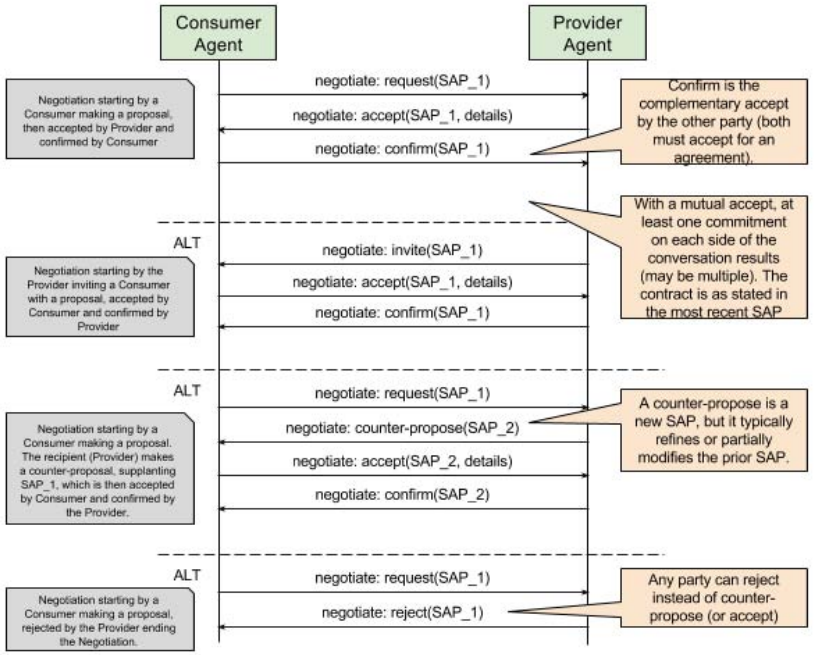
OOI agent negotiation 4/5

```
type <yml> "SAPDoc1" from "SAPDoc1.yml" as SAP;

global protocol Negotiate(role Consumer as C, role Producer as P) {
  propose(SAP) from C to P;

    choice at P {
      accept() from P to C;
      confirm() from C to P;
    } or {
      reject() from P to C;
    } or {
      propose(SAP) from P to C;
      choice at C {
        accept() from C to P;
        confirm() from P to C;
      } or {
        reject() from C to P;
      } or {
        propose(SAP) from C to P;

} } }
```



OOI agent negotiation 5/5 (recursion)

```
type <yml> "SAPDoc1" from "SAPDoc1.yml" as SAP;

global protocol Negotiate(role Consumer as C, role Producer as P) {
  propose(SAP) from C to P;
  rec X {
    choice at P {
      accept() from P to C;
      confirm() from C to P;
    } or {
      reject() from P to C;
    } or {
      propose(SAP) from P to C;
      choice at C {
        accept() from C to P;
        confirm() from P to C;
      } or {
        reject() from C to P;
      } or {
        propose(SAP) from C to P;
        continue X;
      }
    }
```

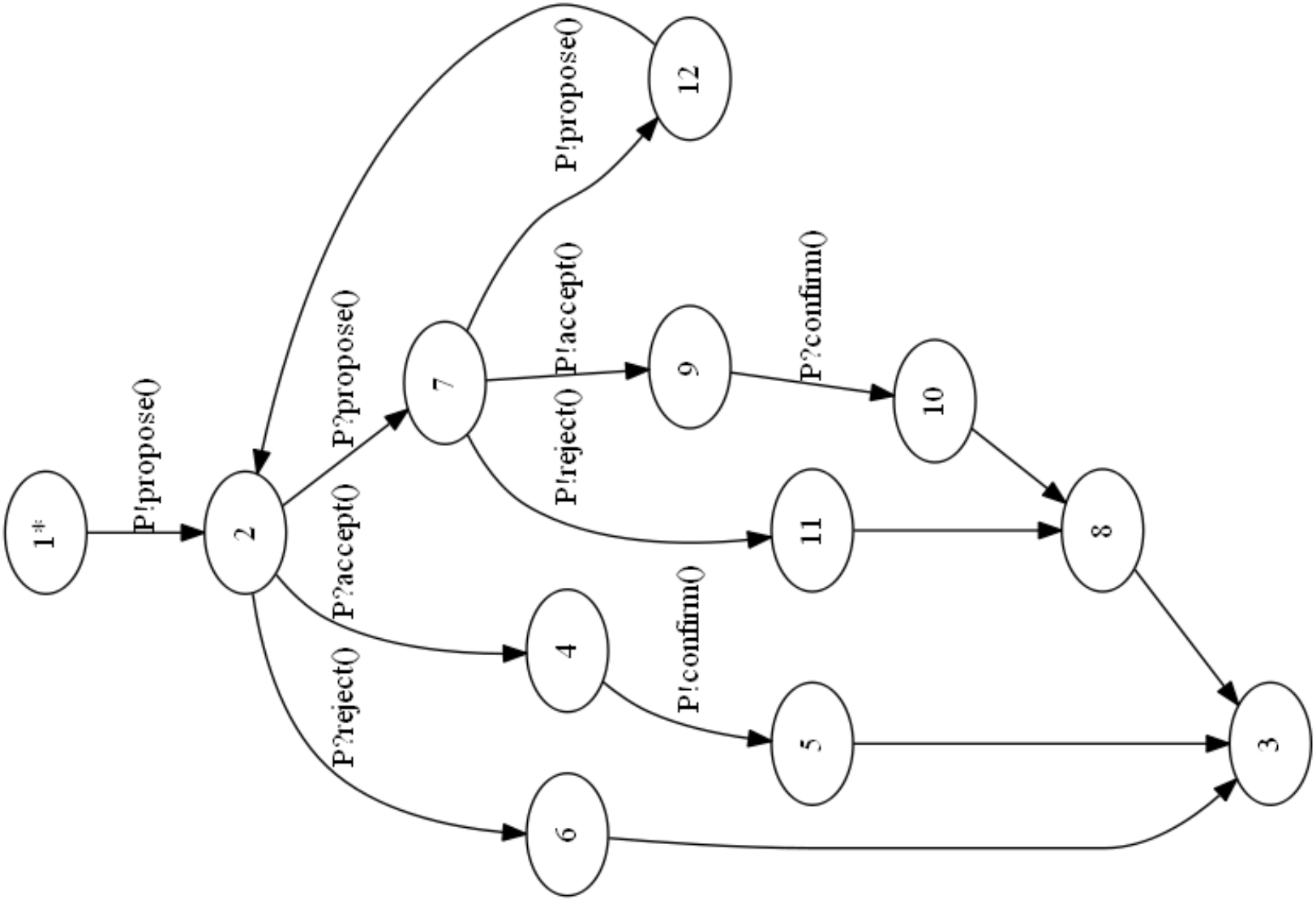


Local protocol projection (Negotiation Consumer)

```
// Global
propose(SAP) from C to P;
rec START {
  choice at P {
    accept() from P to C;
    confirm() from C to P;
  } or {
    reject() from P to C;
  } or {
    propose(SAP) from P to C;
    choice at C {
      accept() from C to P;
      confirm() from P to C;
    } or {
      reject() from C to P;
    } or {
      propose(SAP) from C to P;
      continue START;
} } }
```

```
// Projection for Consumer
propose(SAP) to P;
rec START {
  choice at P {
    accept() from P;
    confirm() to P;
  } or {
    reject() from P;
  } or {
    propose(SAP) from P;
    choice at C {
      accept() to P;
      confirm() from P;
    } or {
      reject() to P;
    } or {
      propose(SAP) to P;
      continue START;
} } }
```

FSM generation (Negotiation Consumer)



Scribble Community

- Webpage:
  - www.scribble.org
- GitHub:
  - https://github.com/scribble
- Tutorial:
  - www.doc.ic.ac.uk/~rhu/scribble/tutorial.html
- Specification (0.3)
  - www.doc.ic.ac.uk/~rhu/scribble/langref.html

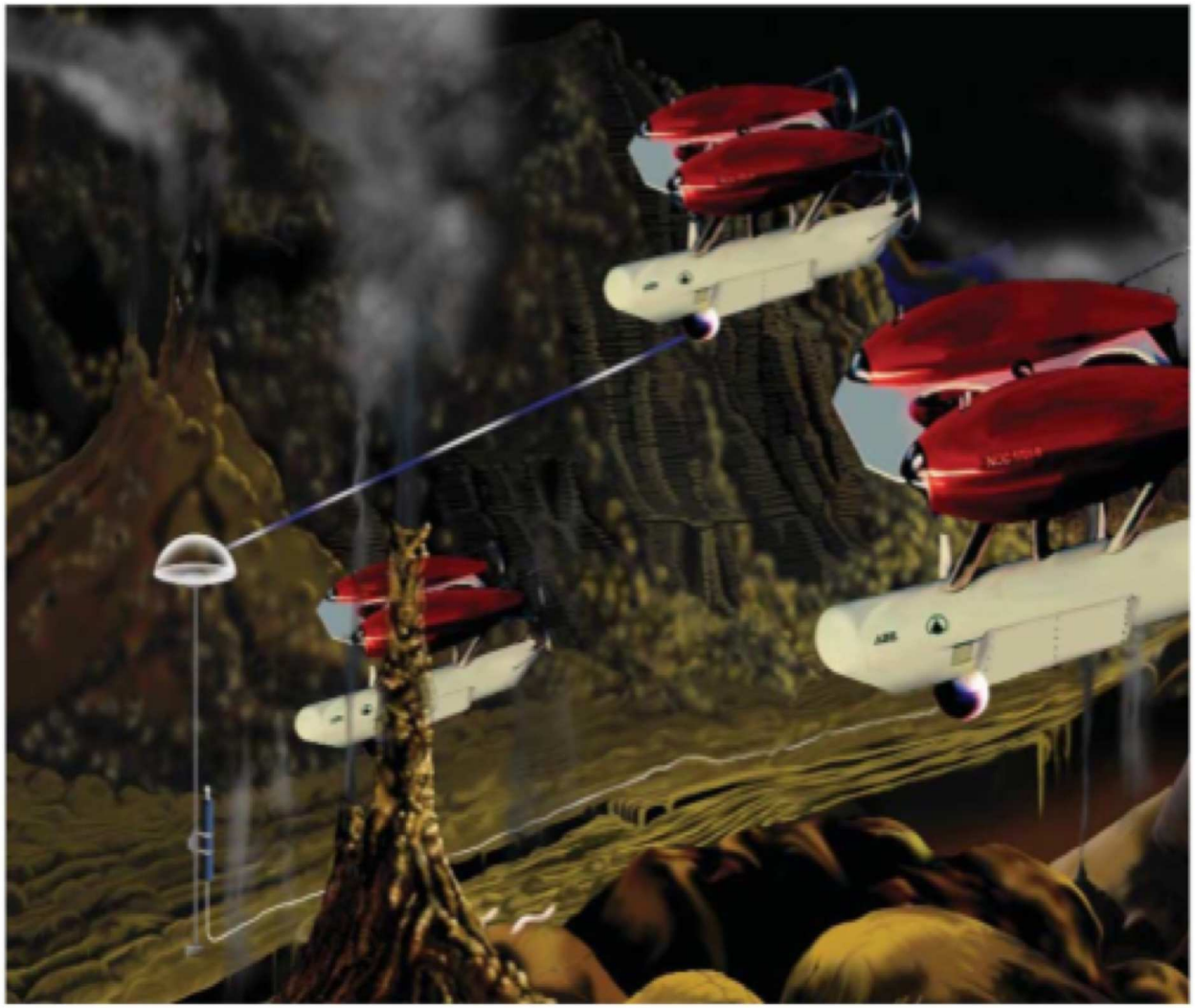


Figure 5: A coordinated set of autonomous underwater vehicles

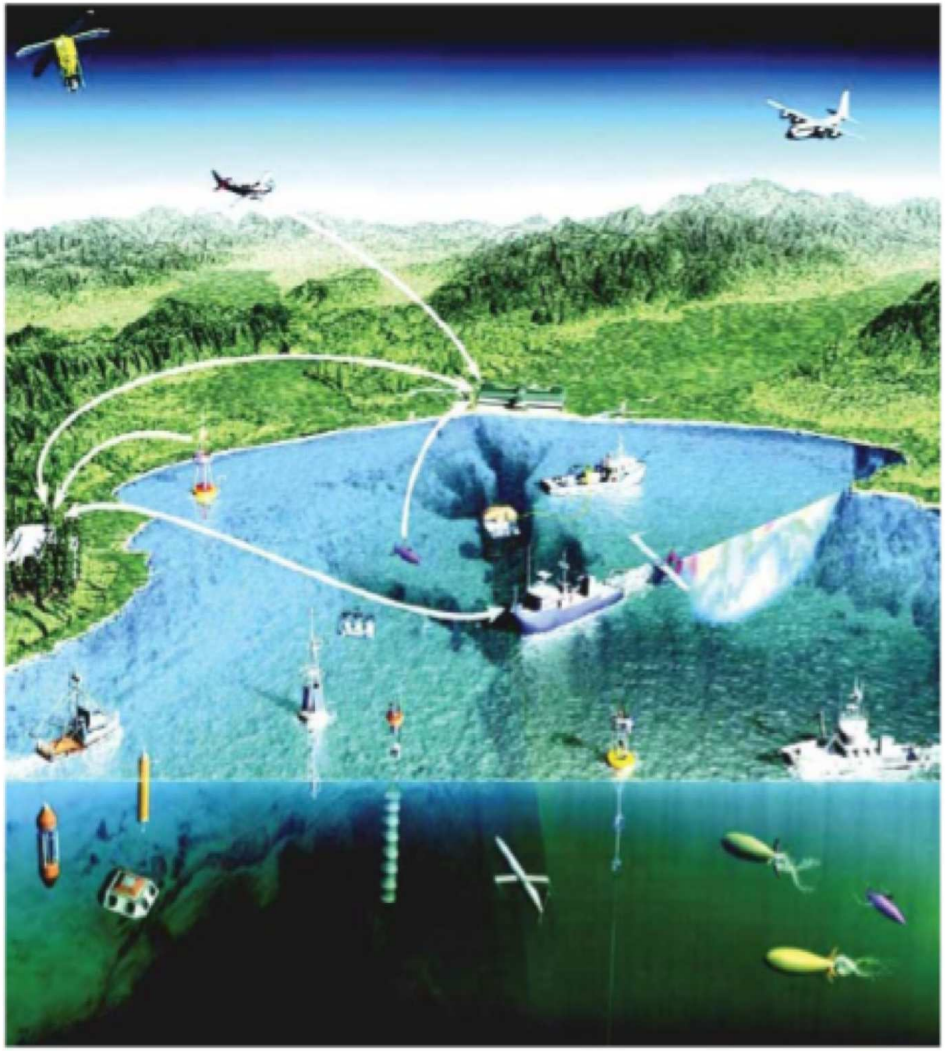


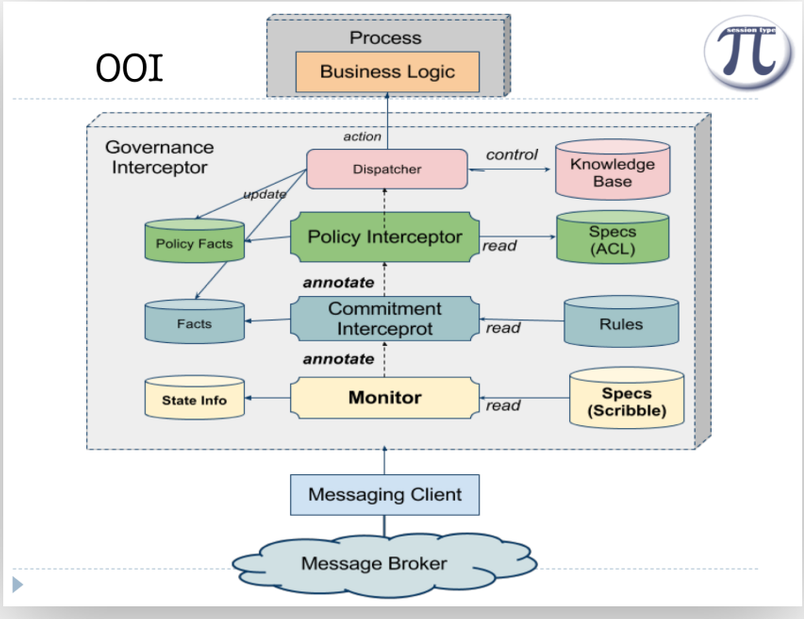
Figure 3: Observatory comprised of ships, aircraft and autonomous vehicles linked to assimilation modeling capabilities on shore

EPSRC
Engineering and Physical Sciences Research Council
EP/N027833/1

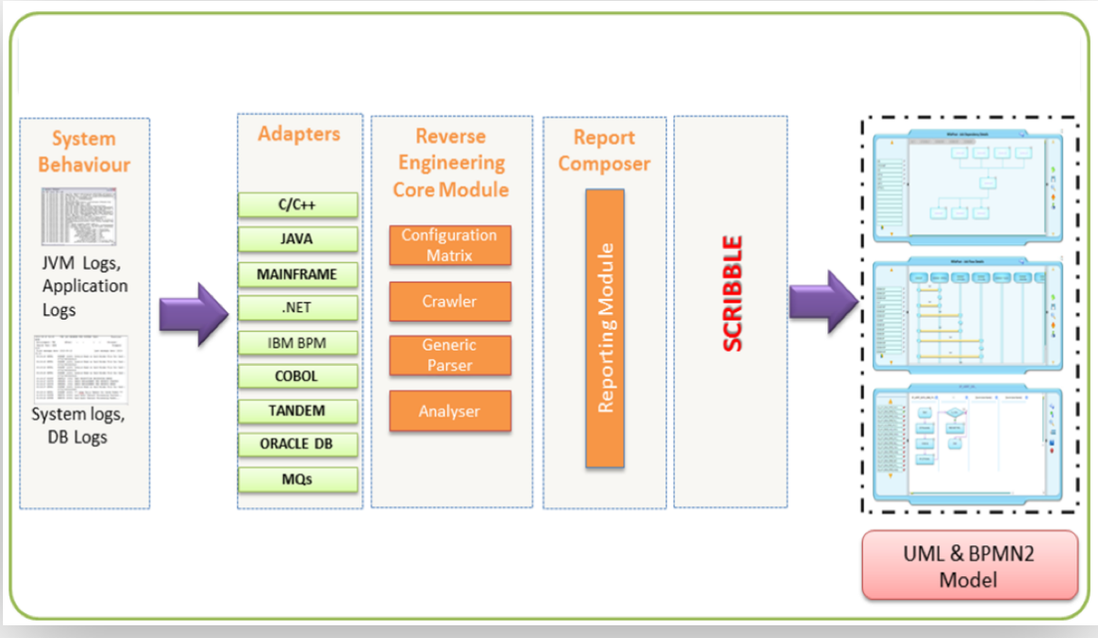
Turtles: Protocol-Based Foundations for Distributed Multiagent Systems

Applications

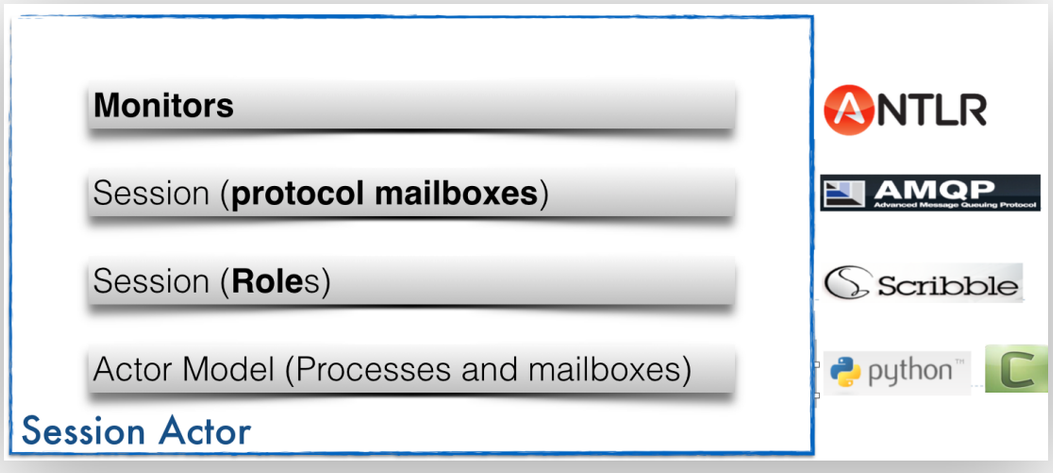
OOI Governance



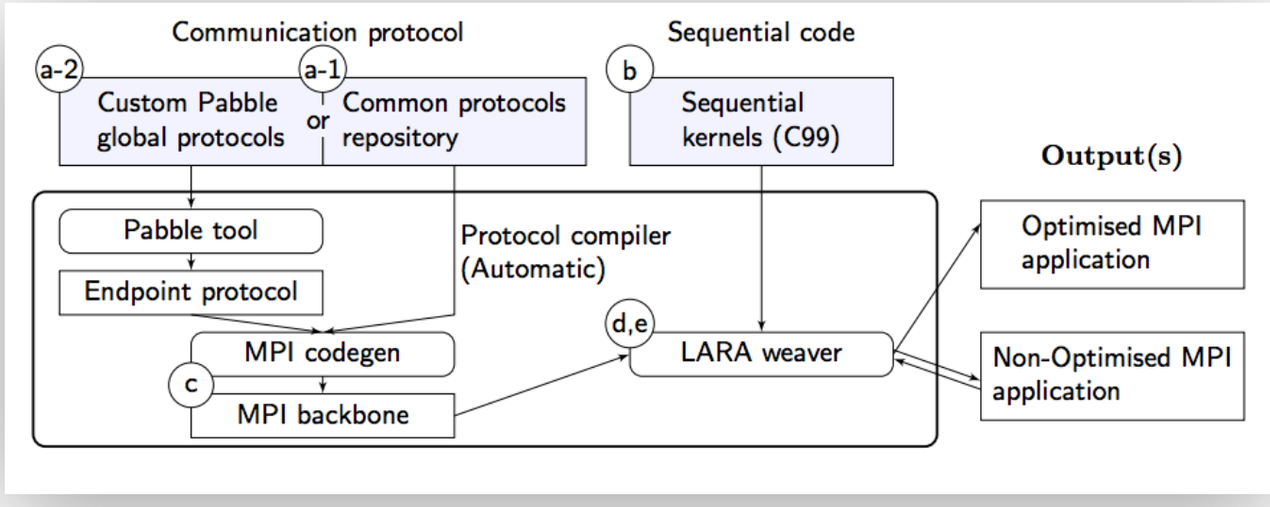
ZDLC: Process Modeling



Protocol Verification

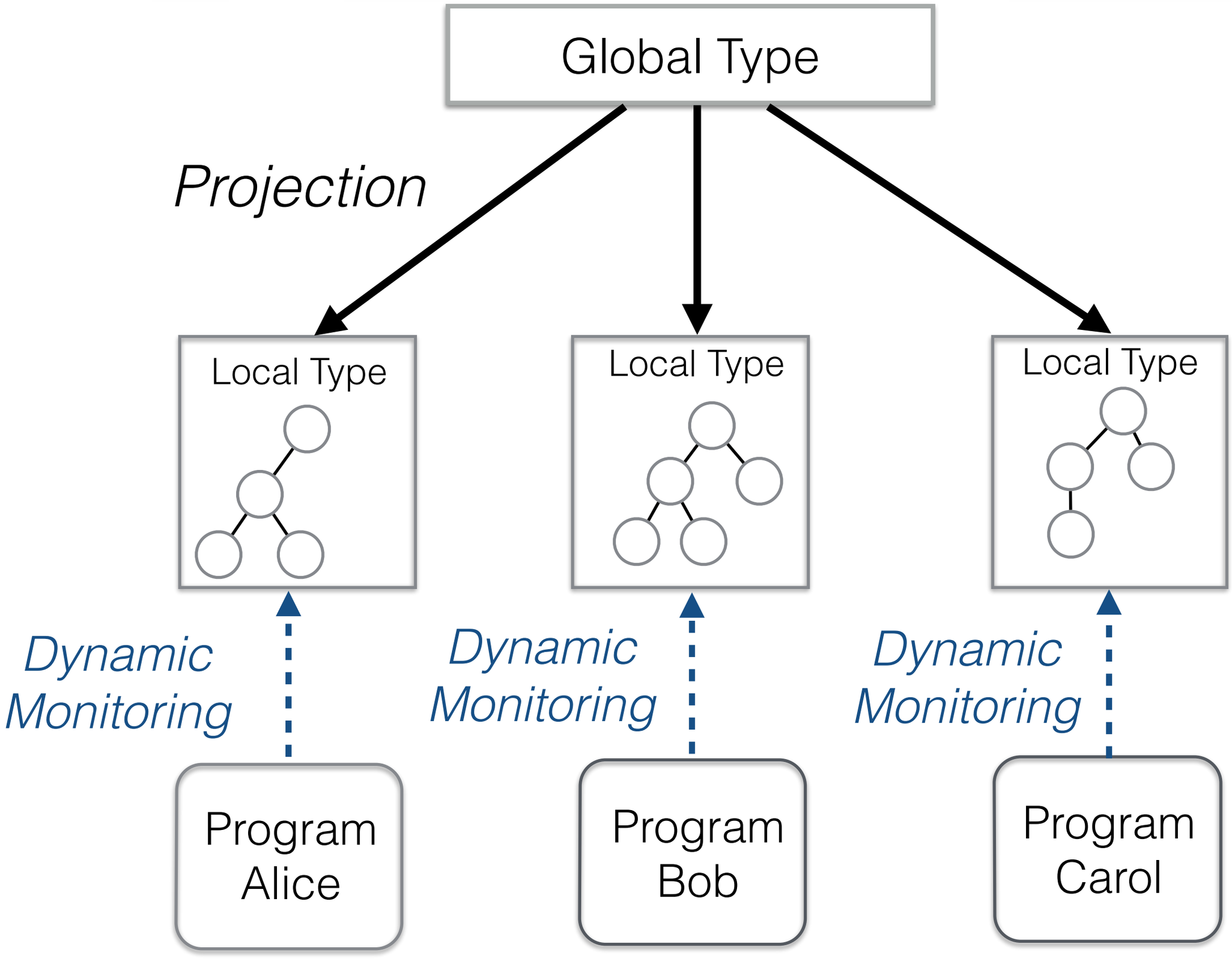


MPI code generations



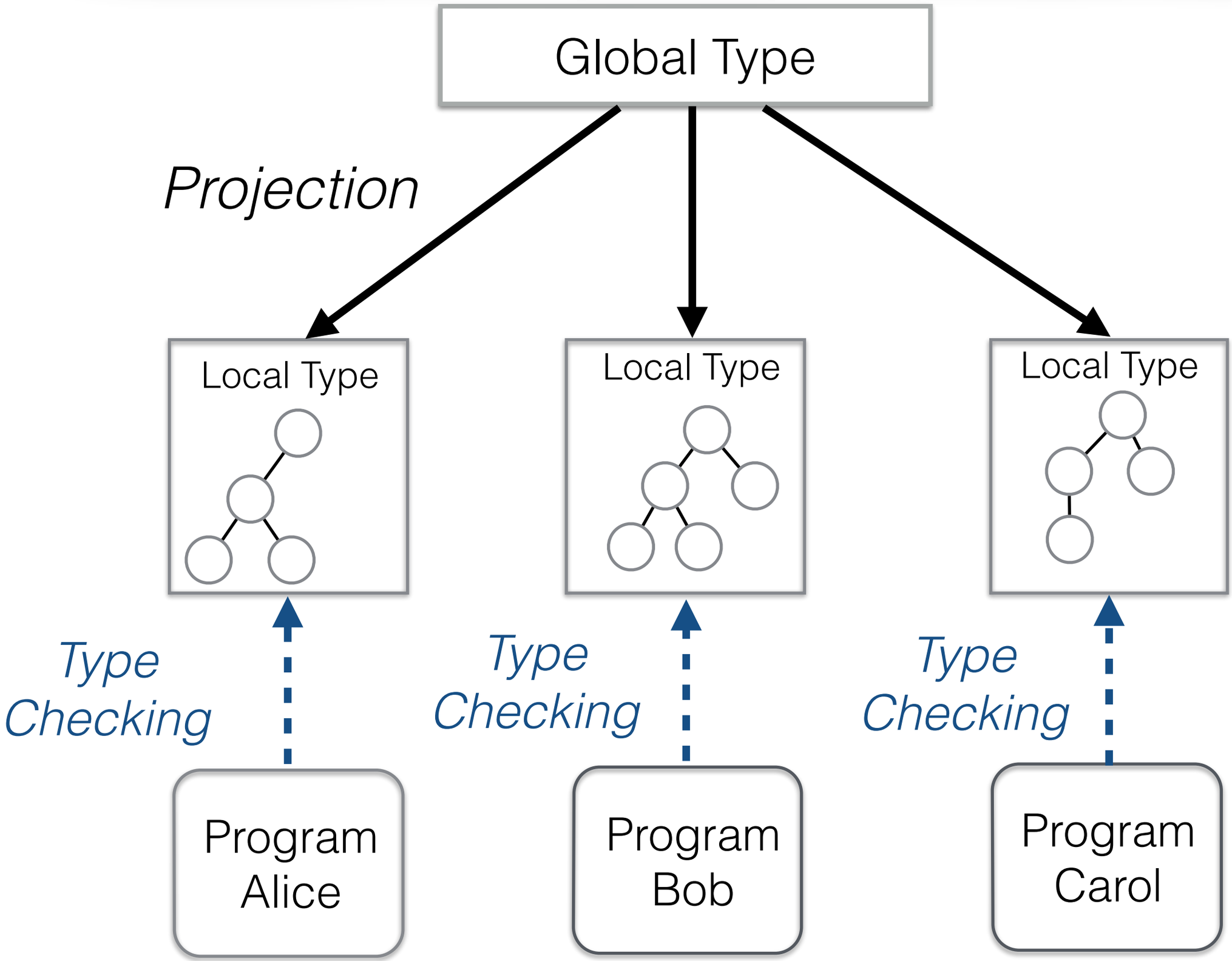
Dynamic Monitoring

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

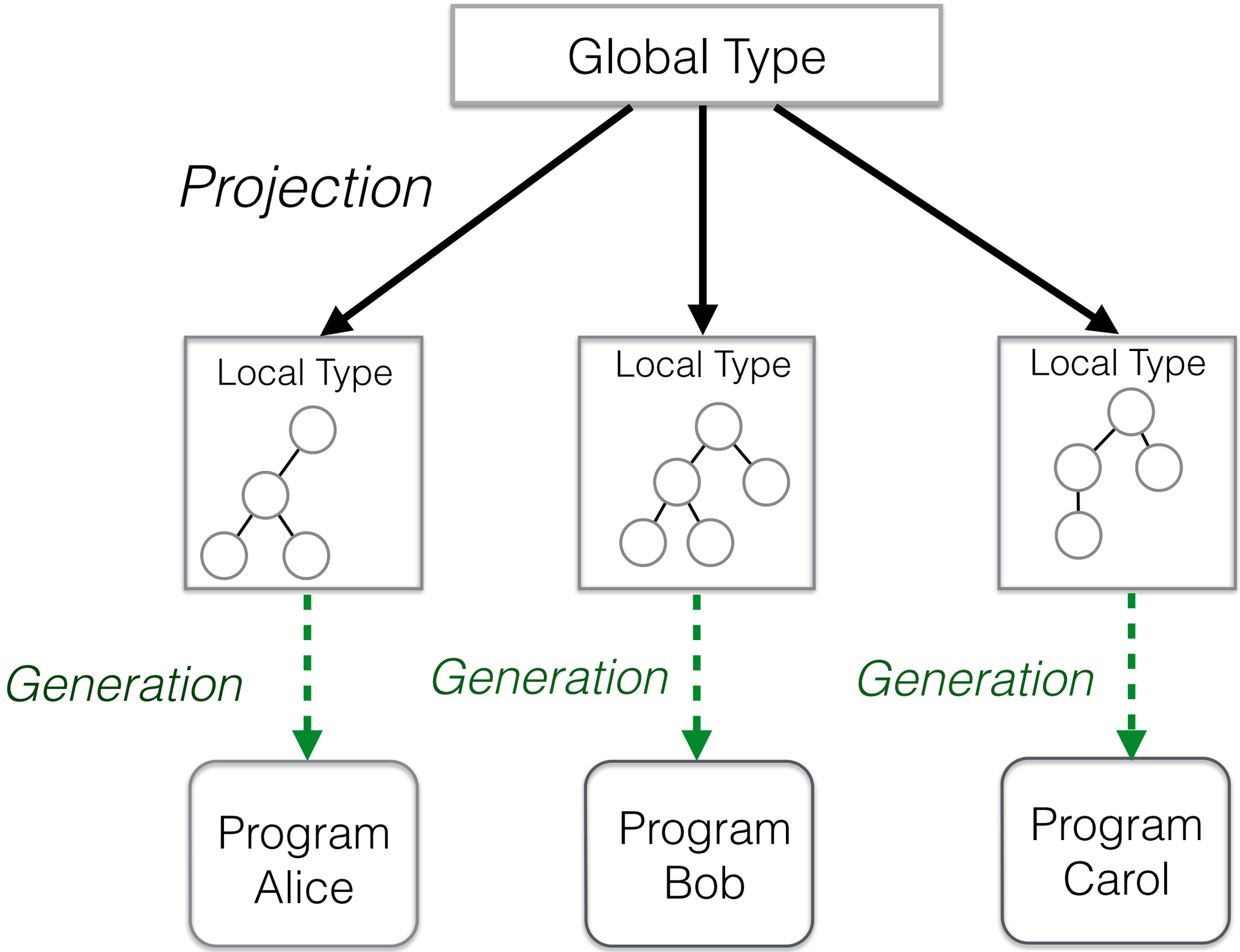


Type Checking

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

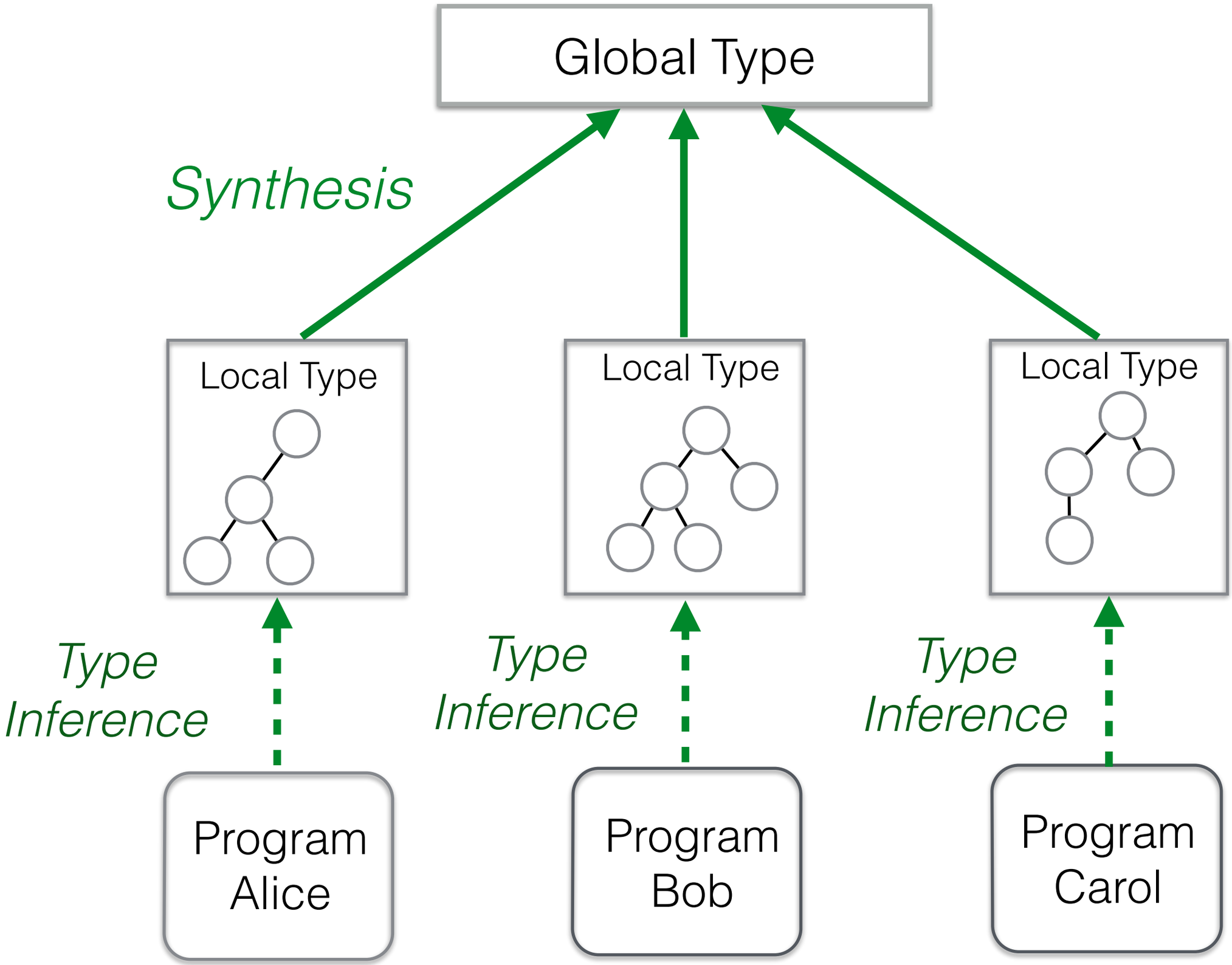


Code Generation [CC'15, FASE'16]



Synthesis

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



Scribble SMTP

TABLE OF CONTENTS

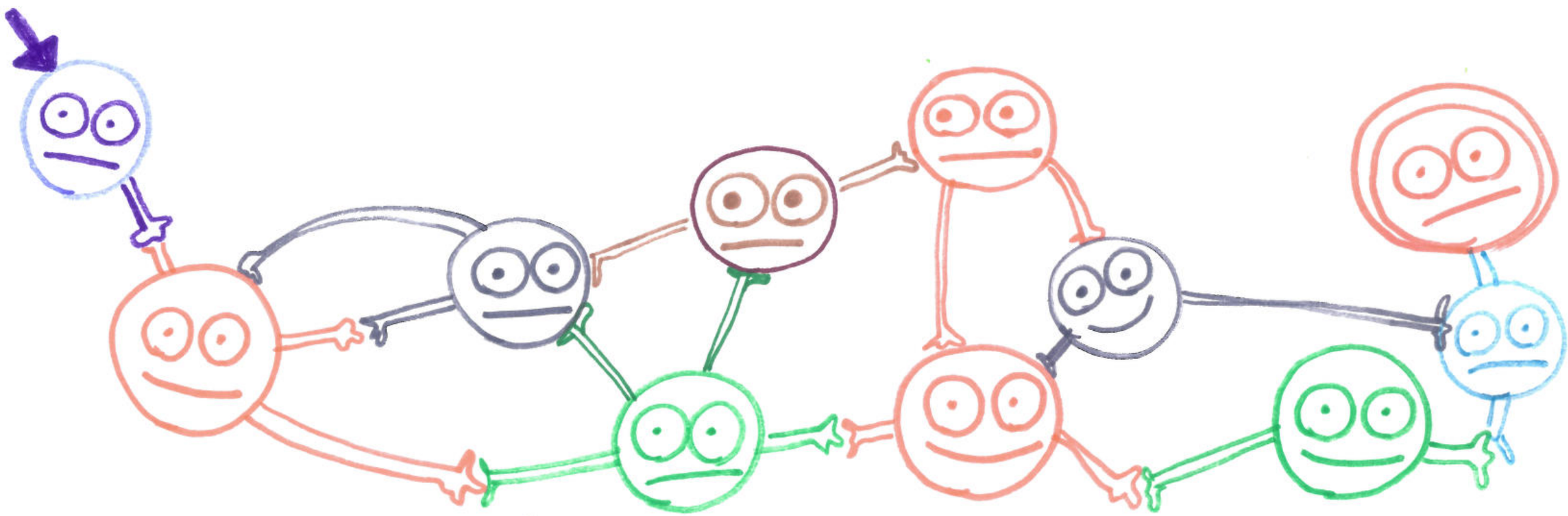
Multiparty Compatibility in Communicating Automata

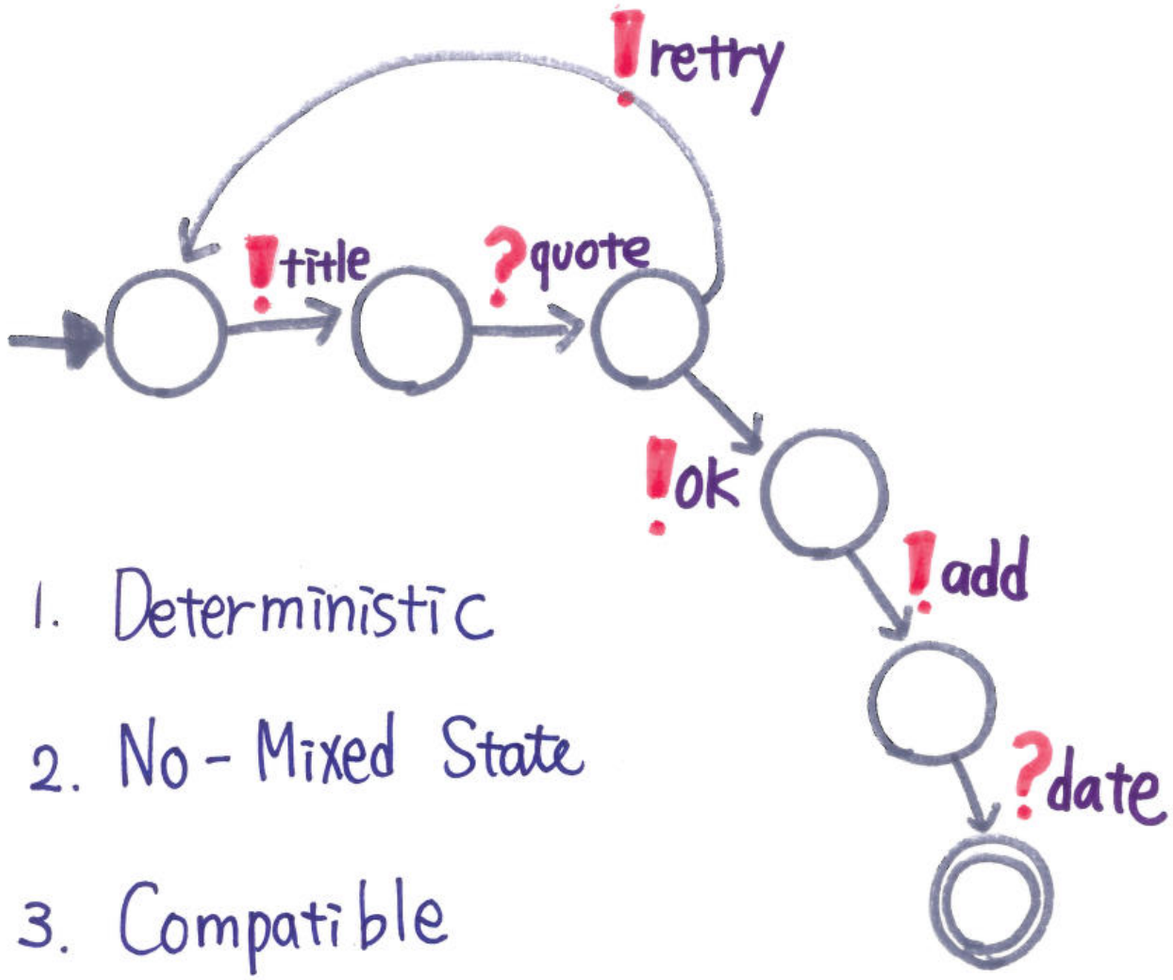
Synthesis and Characterisation of Multiparty Session Types

Nobuko Yoshida

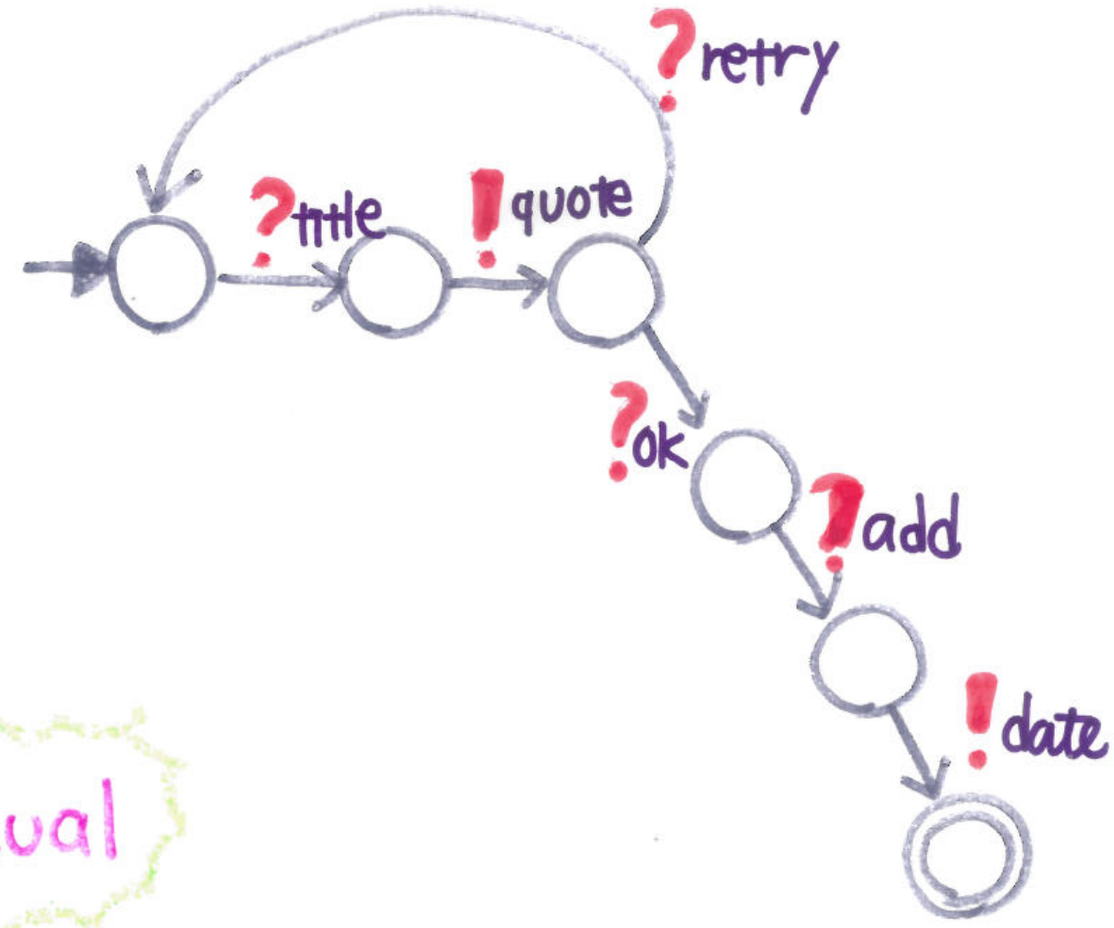
Pierre-Malo Deniélou

ICALP'13





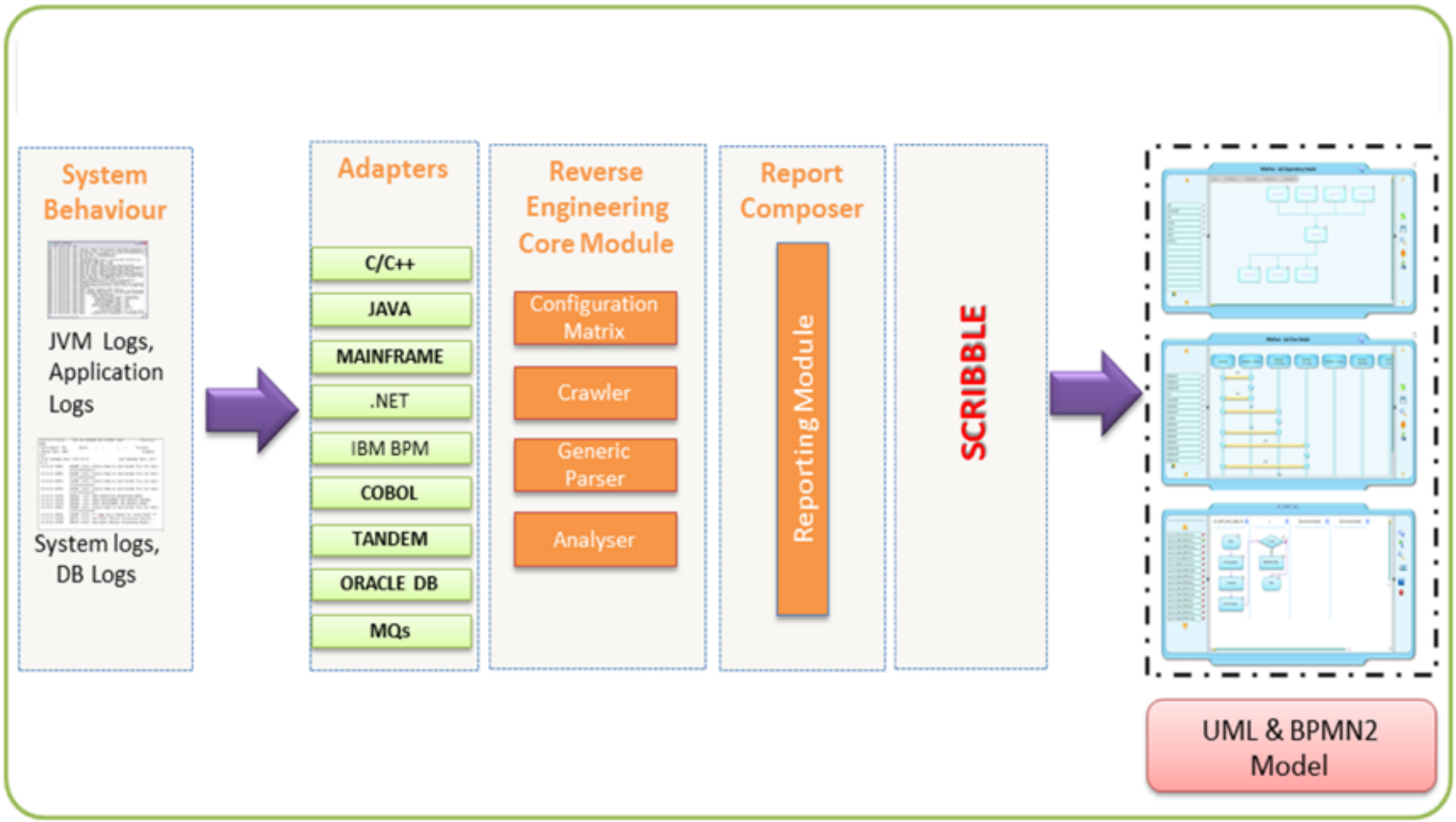
1. Deterministic
2. No-Mixed State
3. Compatible



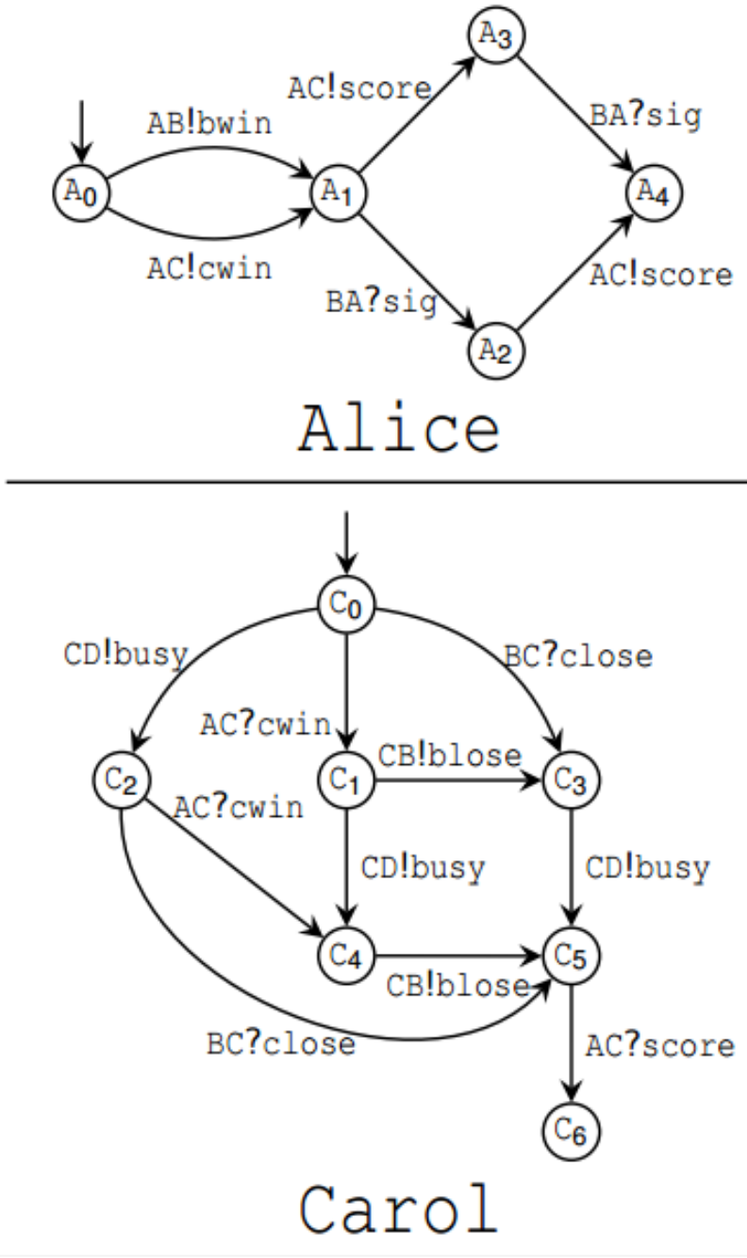
[Gouda et al 1986] Two compatible machines without mixed states which are deterministic satisfy deadlock-freedom.

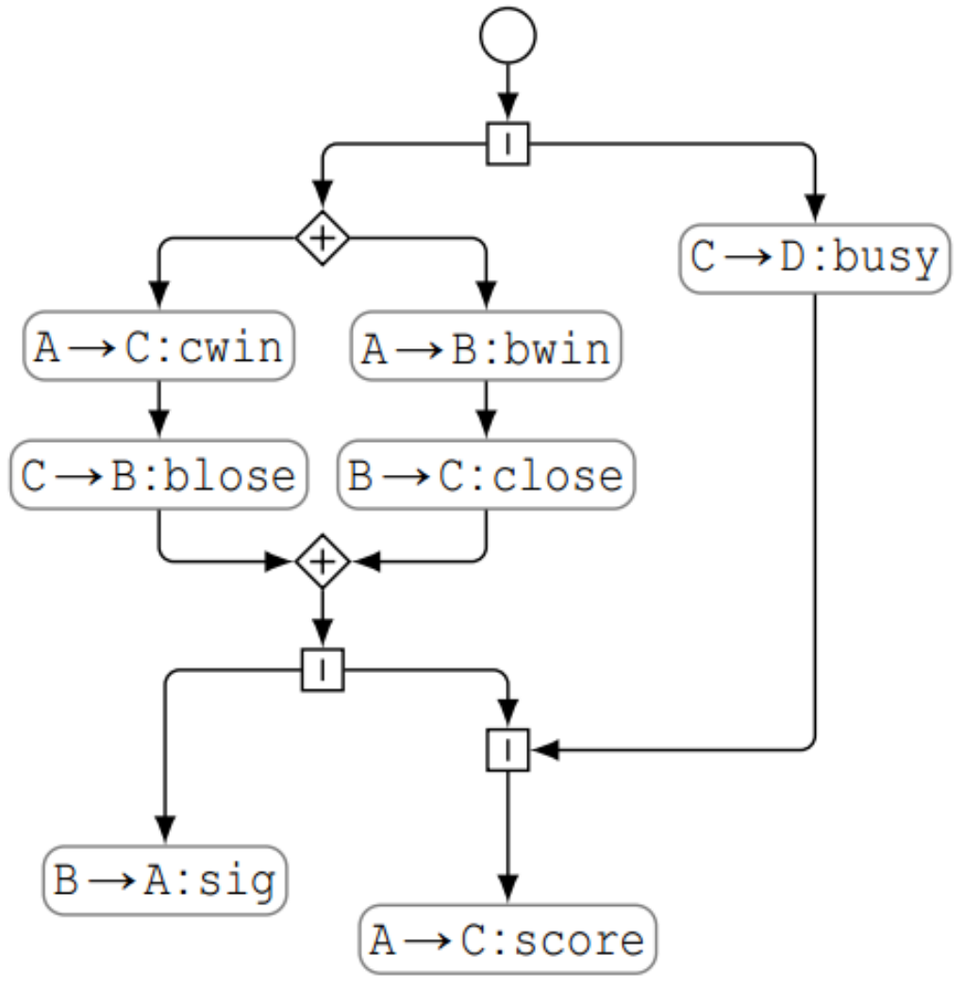
http://www.zdlc.co/faq/

Zero Deviation Life Cycle Platform



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

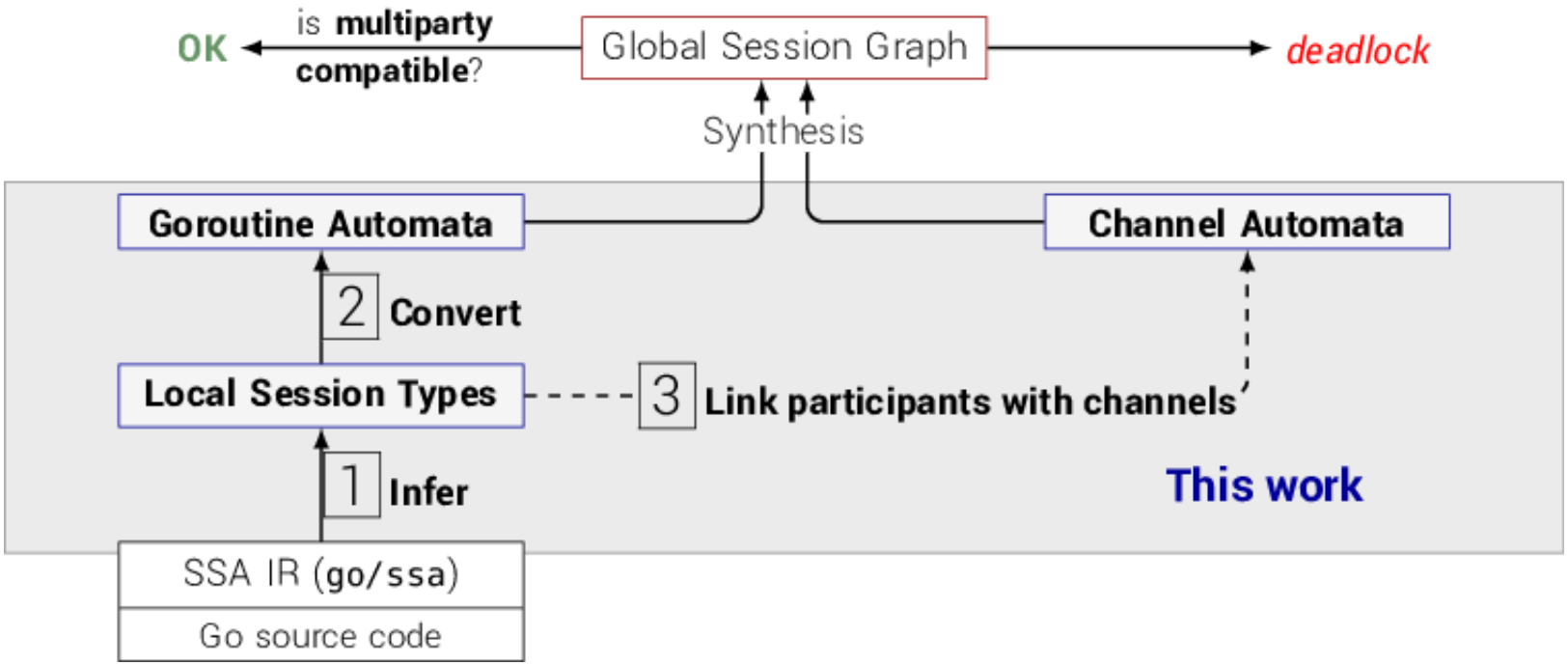




[ESOP'10,ESOP'12,CONCUR'12,CONCUR'14]

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

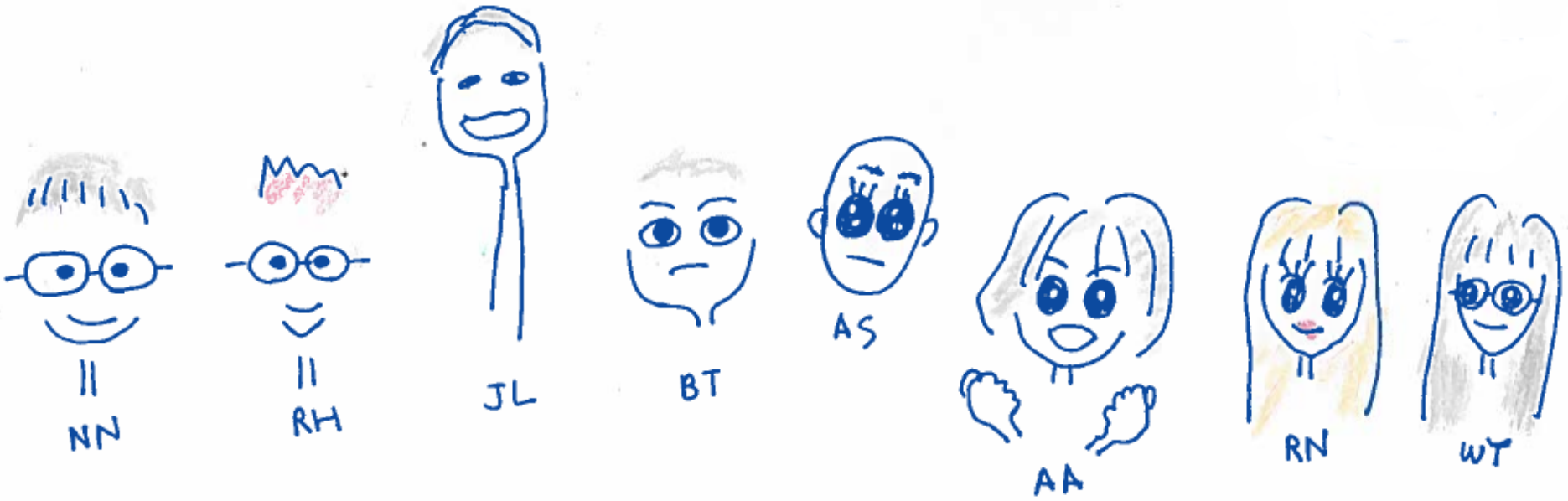


Go and Concurrency

- 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)

Session Type Mobility Group



www.mrg.doc.ic.ac.uk