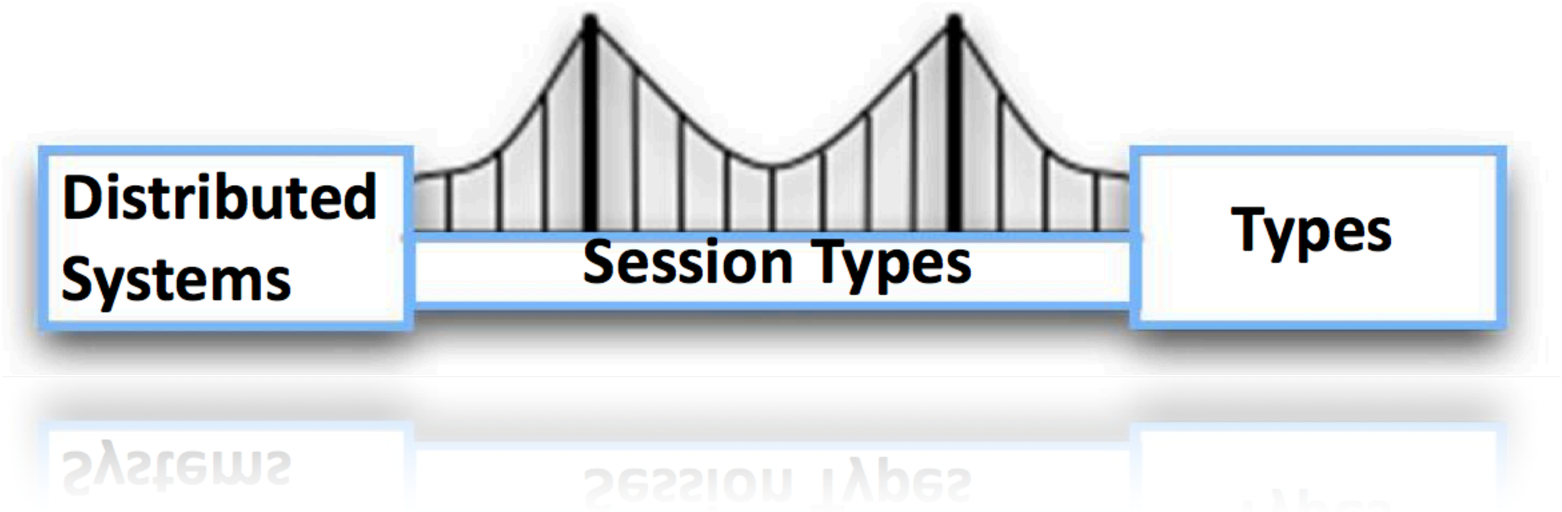


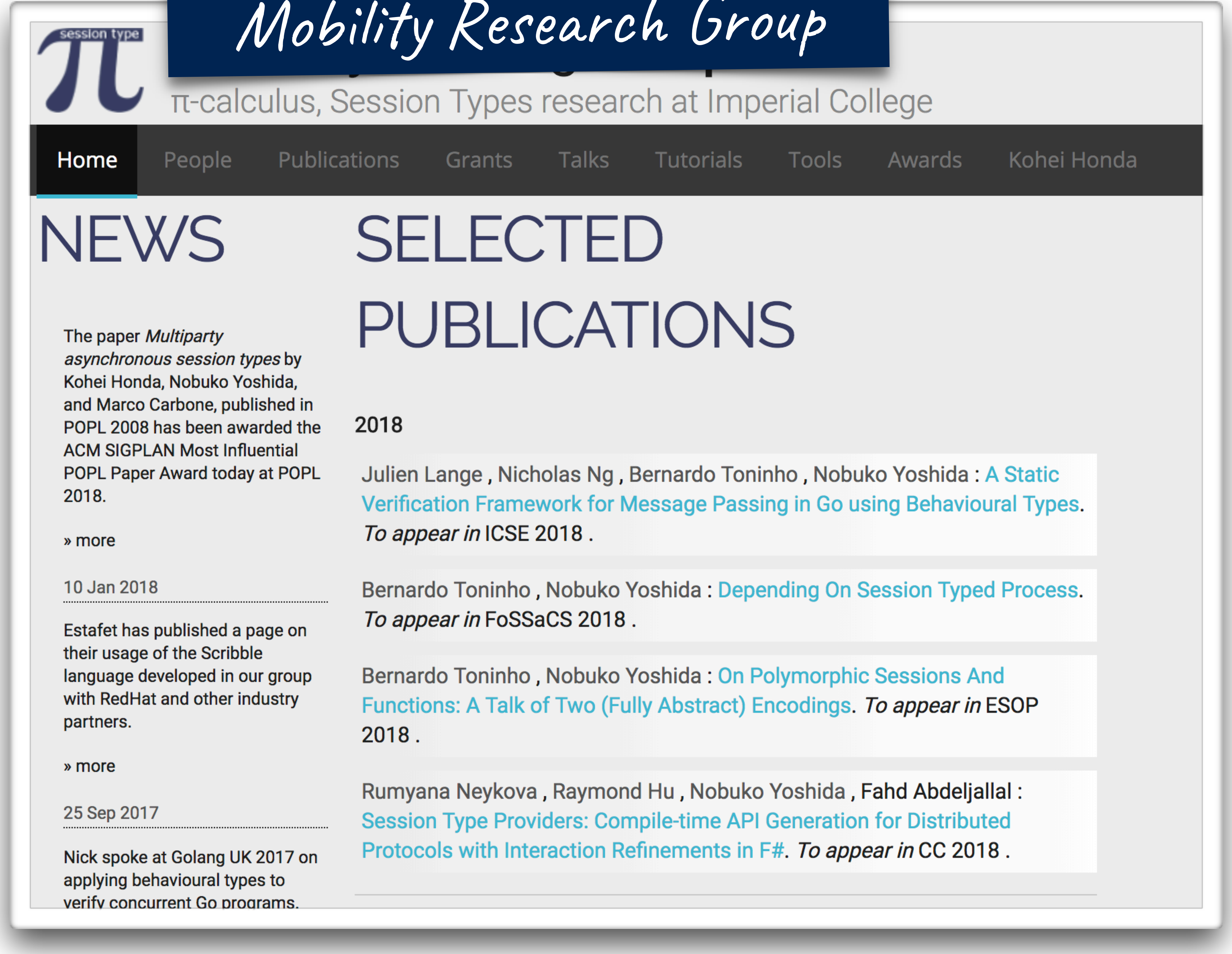
PRINCIPLES AND PRACTICE OF SESSION TYPES

Rumyana Neykova & Nobuko Yoshida

Imperial College London

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

Mobility Research Group



π-calculus, Session Types research at Imperial College

NEWS

The paper *Multiparty asynchronous session types* by Kohei Honda, Nobuko Yoshida, and Marco Carbone, published in POPL 2008 has been awarded the ACM SIGPLAN Most Influential POPL Paper Award today at POPL 2018.

» more

10 Jan 2018

Estafet has published a page on their usage of the Scribble language developed in our group with RedHat and other industry partners.

» more

25 Sep 2017

Nick spoke at Golang UK 2017 on applying behavioural types to verify concurrent Go programs.

SELECTED PUBLICATIONS

2018

Julien Lange , Nicholas Ng , Bernardo Toninho , Nobuko Yoshida : A Static Verification Framework for Message Passing in Go using Behavioural Types. *To appear in* ICSE 2018 .

Bernardo Toninho , Nobuko Yoshida : Depending On Session Typed Process. *To appear in* FoSSaCS 2018 .

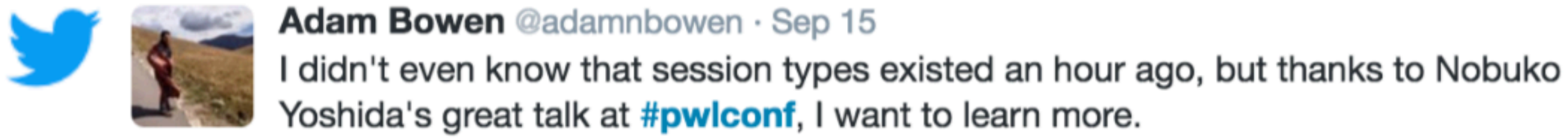
Bernardo Toninho , Nobuko Yoshida : On Polymorphic Sessions And Functions: A Talk of Two (Fully Abstract) Encodings. *To appear in* ESOP 2018 .

Rumyana Neykova , Raymond Hu , Nobuko Yoshida , Fahd Abdeljallal : Session Type Providers: Compile-time API Generation for Distributed Protocols with Interaction Refinements in F#. *To appear in* CC 2018 .

Post-docs:
Simon CASTEL
David CASTRO
Francisco FER
Raymond HU
Rumyana NEY
Nicholas NG
Alceste SCALA

PhD Students:
Assel ALTAYEV
Juliana FRANC
Eva GRAVERSE

Interactions with Industries

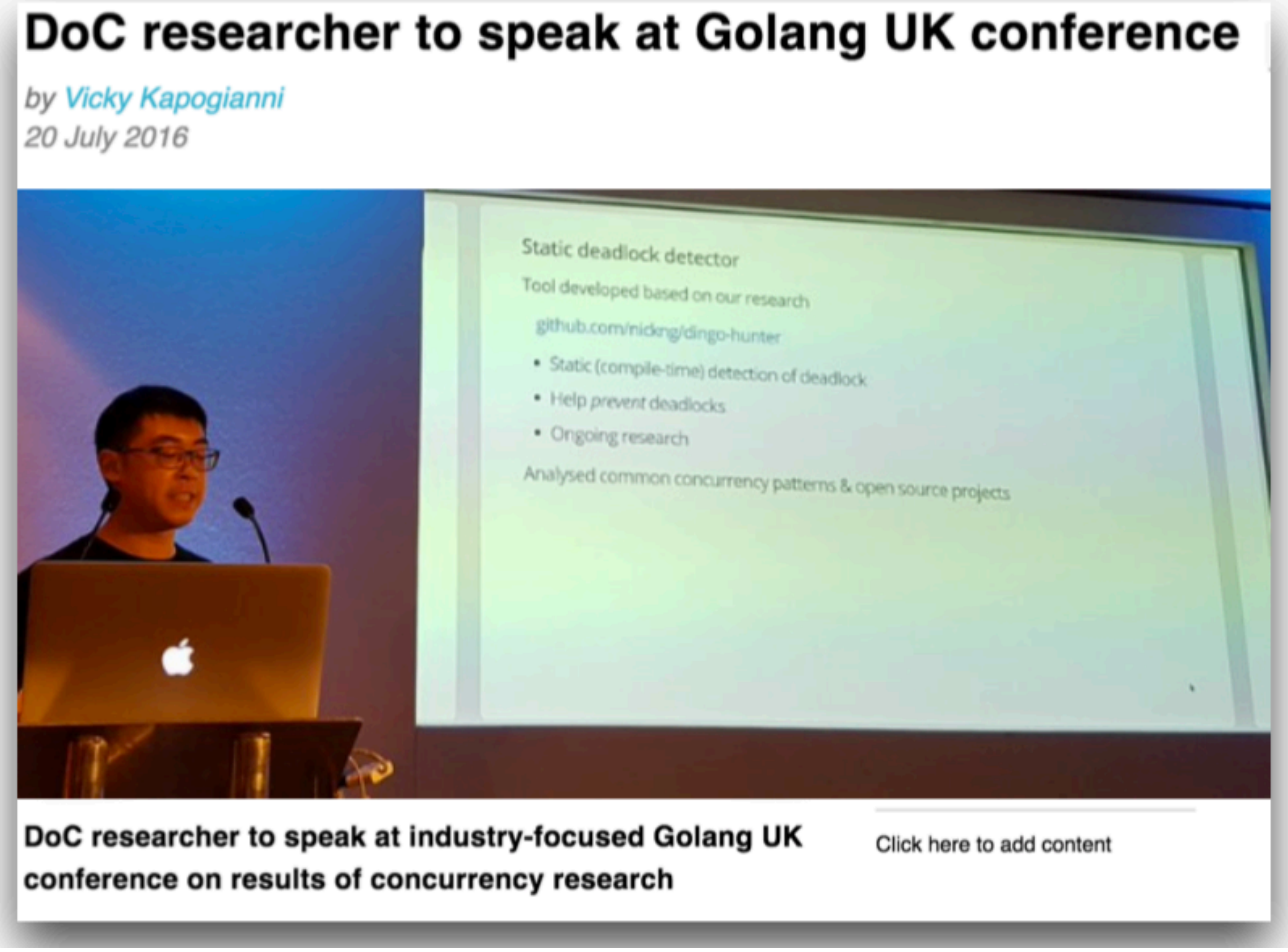


Adam Bowen @adamnbowen · Sep 15

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

Static deadlock detector

Tool developed based on our research

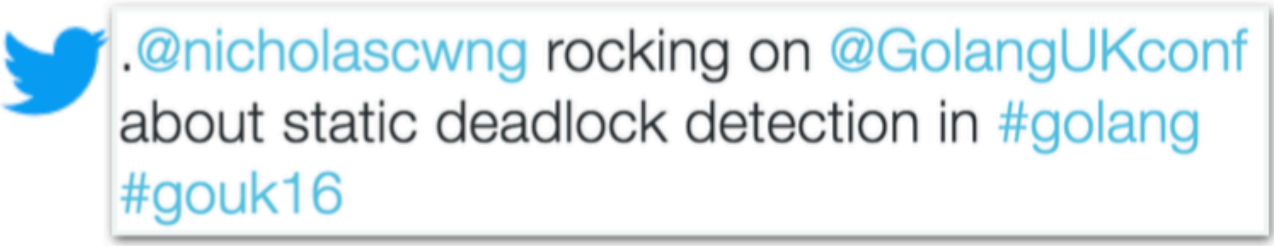
github.com/nickng/dingo-hunter

- Static (compile-time) detection of deadlock
- Help *prevent* deadlocks
- Ongoing research

Analysed common concurrency patterns & open source projects

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

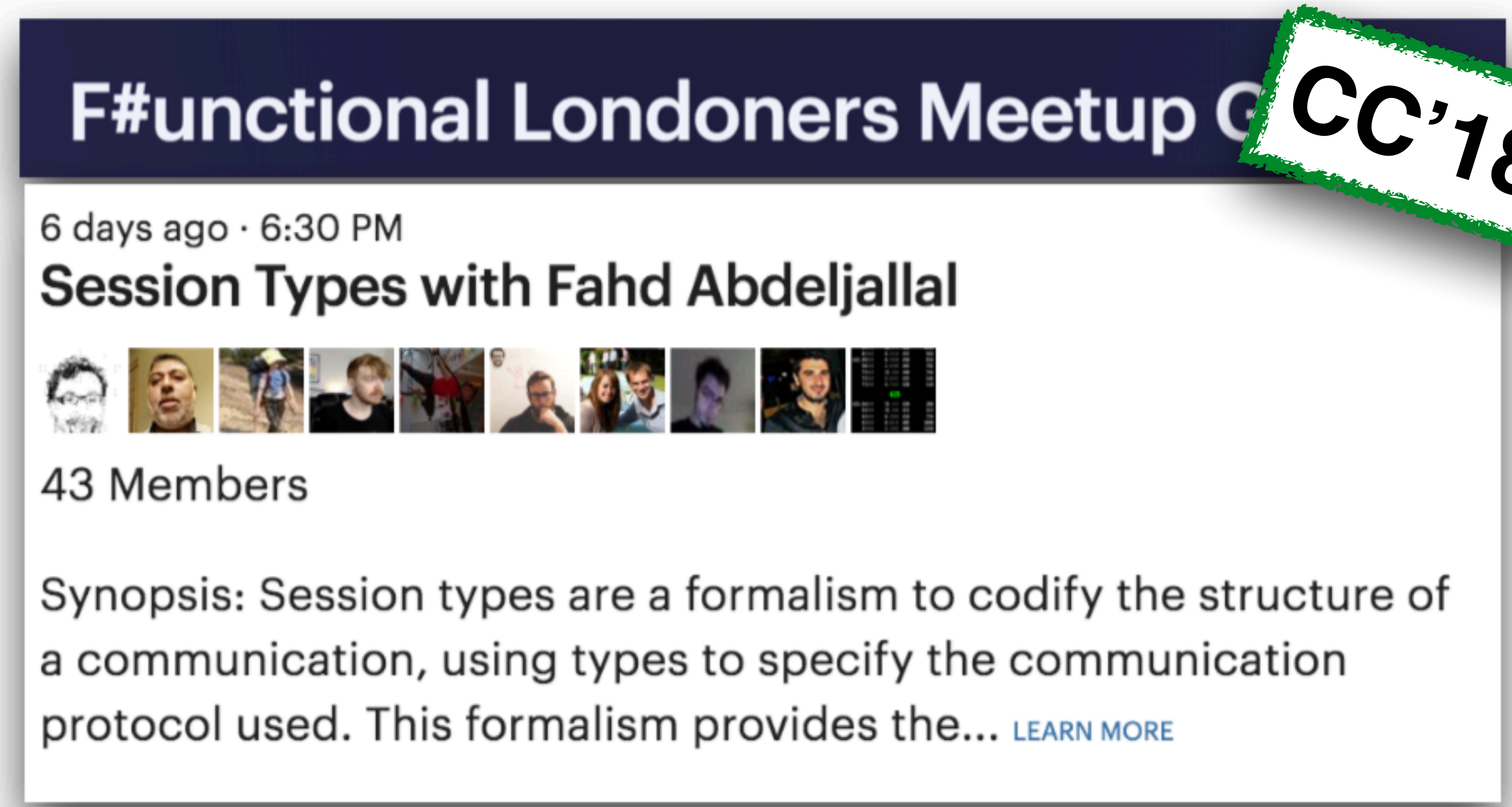
Click here to add content



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

Interactions with Industries

CC'18



F#unctional Londoners Meetup G

6 days ago · 6:30 PM

Session Types with Fahd Abdeljallal

43 Members

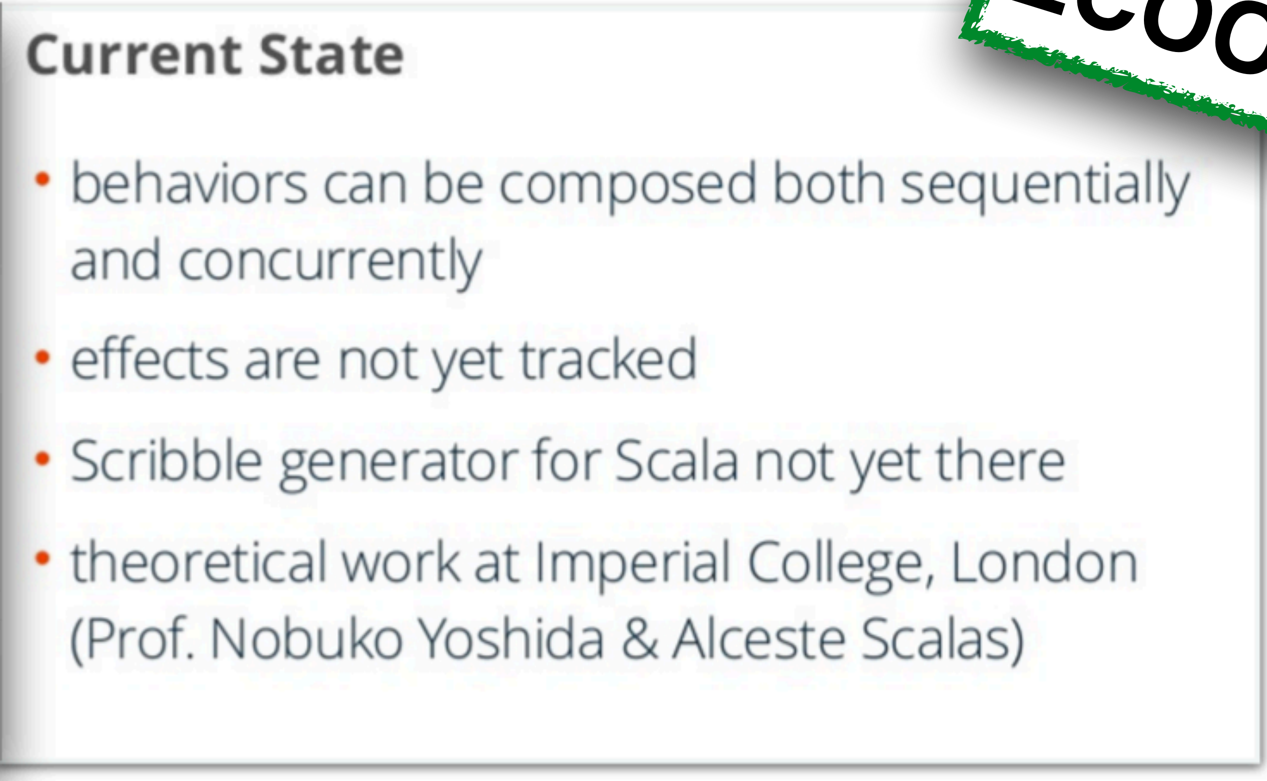
Synopsis: Session types are a formalism to codify the structure of a communication, using types to specify the communication protocol used. This formalism provides the... LEARN MORE

ECOOP'17

ibuted Systems
vs.
Compositionality

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

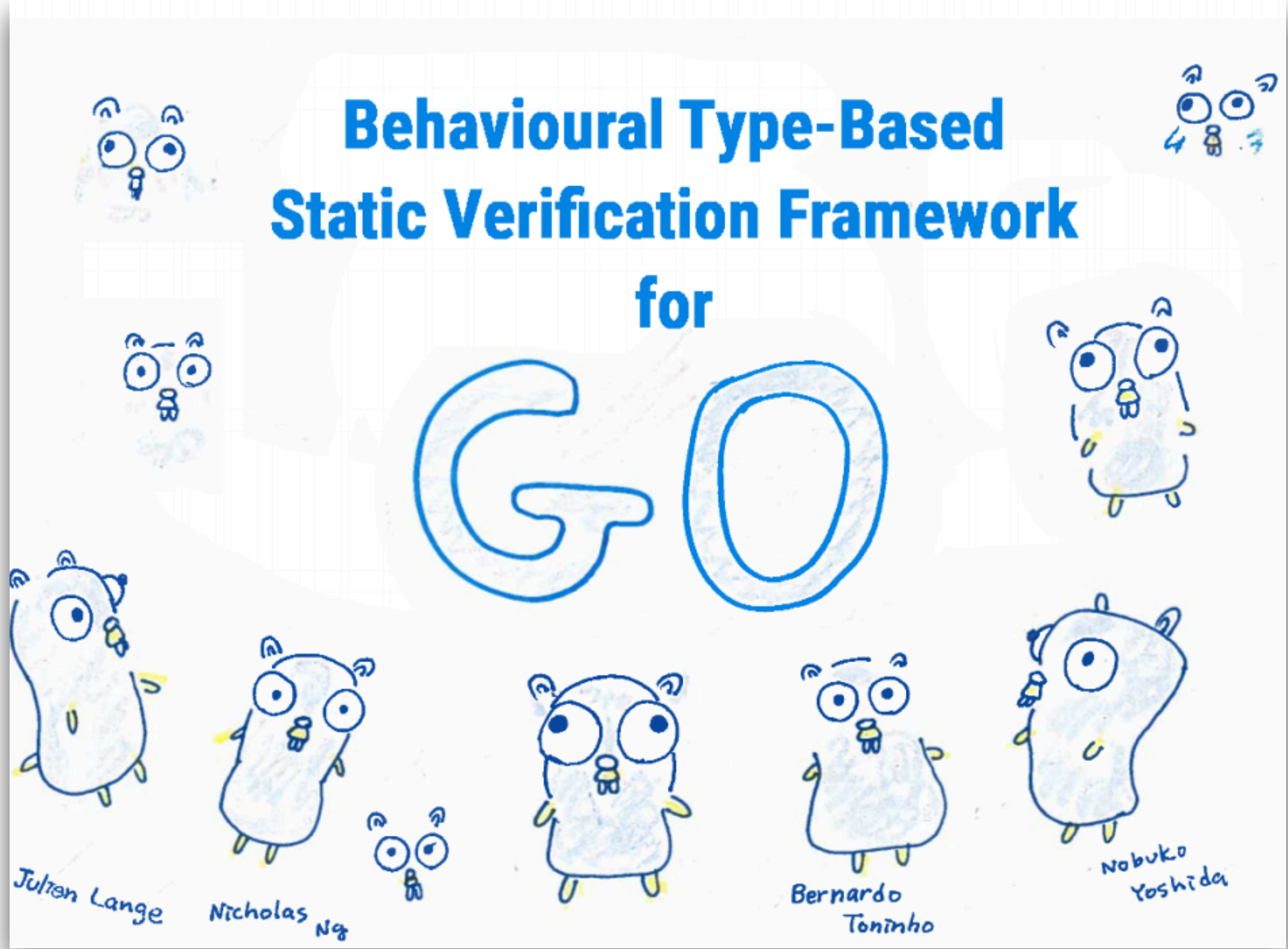
ECOOP'16



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)

Behavioural Type-Based Static Verification Framework for GO

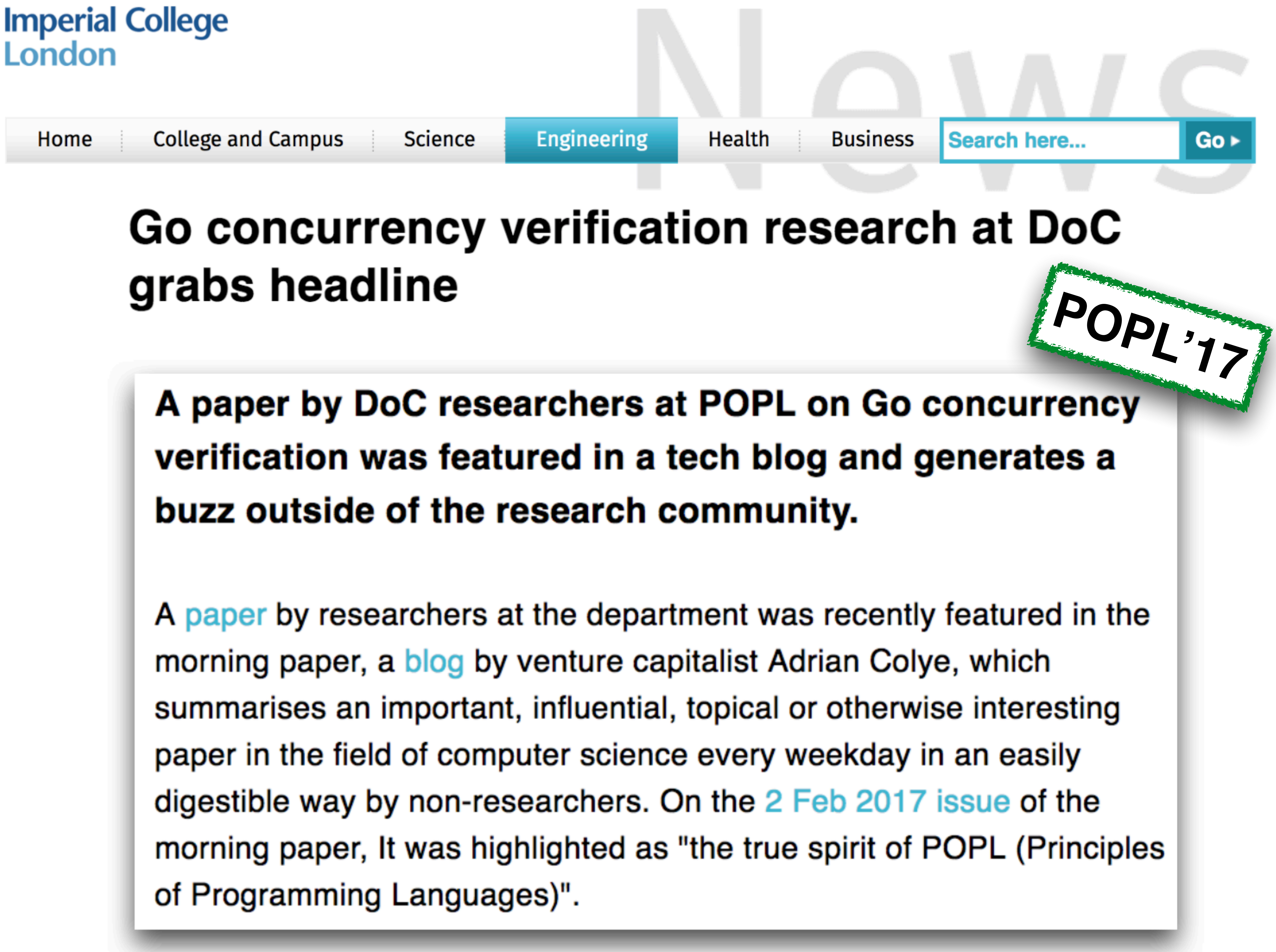


Julien Lange

Nicholas Ng

Bernardo Toninho

Nobuko Yoshida



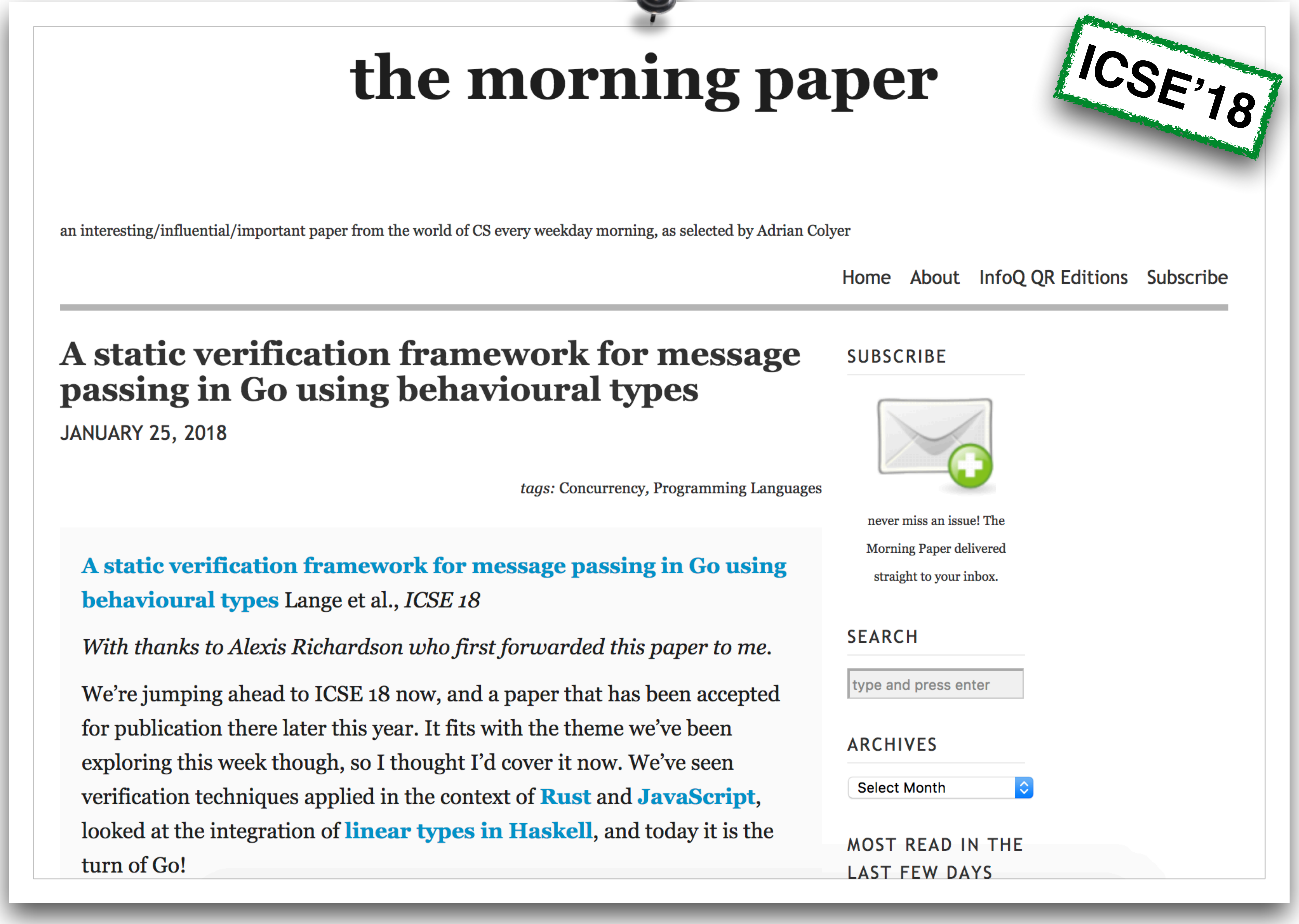
Go concurrency verification research at DoC grabs headline

POPL'17

A paper by DoC researchers at POPL on Go concurrency verification was featured in a tech blog and generates a buzz outside of the research community.

A paper by researchers at the department was recently featured in the morning paper, a blog by venture capitalist Adrian Colye, which summarises an important, influential, topical or otherwise interesting paper in the field of computer science every weekday in an easily digestible way by non-researchers. On the 2 Feb 2017 issue of the morning paper, It was highlighted as "the true spirit of POPL (Principles of Programming Languages)".

ICSE’18



the morning paper

an interesting/influential/important paper from the world of CS every weekday morning, as selected by Adrian Colyer

A static verification framework for message passing in Go using behavioural types

JANUARY 25, 2018

tags: Concurrency, Programming Languages

A static verification framework for message passing in Go using behavioural types Lange et al., *ICSE 18*

With thanks to Alexis Richardson who first forwarded this paper to me.

We’re jumping ahead to ICSE 18 now, and a paper that has been accepted for publication there later this year. It fits with the theme we’ve been exploring this week though, so I thought I’d cover it now. We’ve seen verification techniques applied in the context of **Rust** and **JavaScript**, looked at the integration of **linear types in Haskell**, and today it is the turn of Go!

SUBSCRIBE
never miss an issue! The Morning Paper delivered straight to your inbox.
SEARCH
type and press enter
ARCHIVES
Select Month
MOST READ IN THE LAST FEW DAYS

Selected Publications 2017/2018

- **[CC’18]** Rumyana Neykova , Raymond Hu, NY, Fahd Abdeljallal: Session Type Providers: Compile-time API Generation for Distributed Protocols with Interaction Refinements in F#.
- **[FoSSaCS’18]** Bernardo Toninho, NY: Depending On Session Typed Process.
- **[ESOP’18]** Bernardo Toninho, NY: On Polymorphic Sessions And Functions: A Talk of Two (Fully Abstract) Encodings.
- **[ESOP’18]** Malte Viering, Tzu-Chun Chen, Patrick Eugster, Raymond Hu , Lukasz Ziarek: A Typing Discipline for Statically Verified Crash Failure Handling in Distributed Systems.
- **[ICSE’18]** Julien Lange, Nicholas Ng, Bernardo Toninho, NY : A Static Verification Framework for Message Passing in Go using Behavioural Types
- **[ECOOP’17]** Alceste Scala, Raymond Hu, Ornela Darda, NY: A Linear Decomposition of Multiparty Sessions for Safe Distributed Programming..
- **[COORDINATION’17]** Keigo Imai, NY, 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.

Selected Publications 2017/2018

- **[CC’18]** Rumyana Neykova , Raymond Hu, NY, Fahd Abdeljallal: Session Type Providers: Compile-time API Generation for Distributed Protocols with Interaction Refinements in F#.
- **[FoSSaCS’18]** Bernardo Toninho, NY: Depending On Session Typed Process.
- **[ESOP’18]** Bernardo Toninho, NY: On Polymorphic Sessions And Functions: A Talk of Two (Fully Abstract) Encodings.
- **[ESOP’18]** Malte Viering, Tzu-Chun Chen, Patrick Eugster, Raymond Hu , Lukasz Ziarek: A Typing Discipline for Statically Verified Crash Failure Handling in Distributed Systems.
- **[ICSE’18]** Julien Lange, Nicholas Ng, Bernardo Toninho, NY : A Static Verification Framework for Message Passing in Go using Behavioural Types.
- **[ECOOP’17]** Alceste Scala, Raymond Hu, Ornela Darda, NY: A Linear Decomposition of Multiparty Sessions for Safe Distributed Programming.
- **[COORDINATION’17]** Keigo Imai, NY, 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.

Dialogue between Industry and Academia

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

⇓

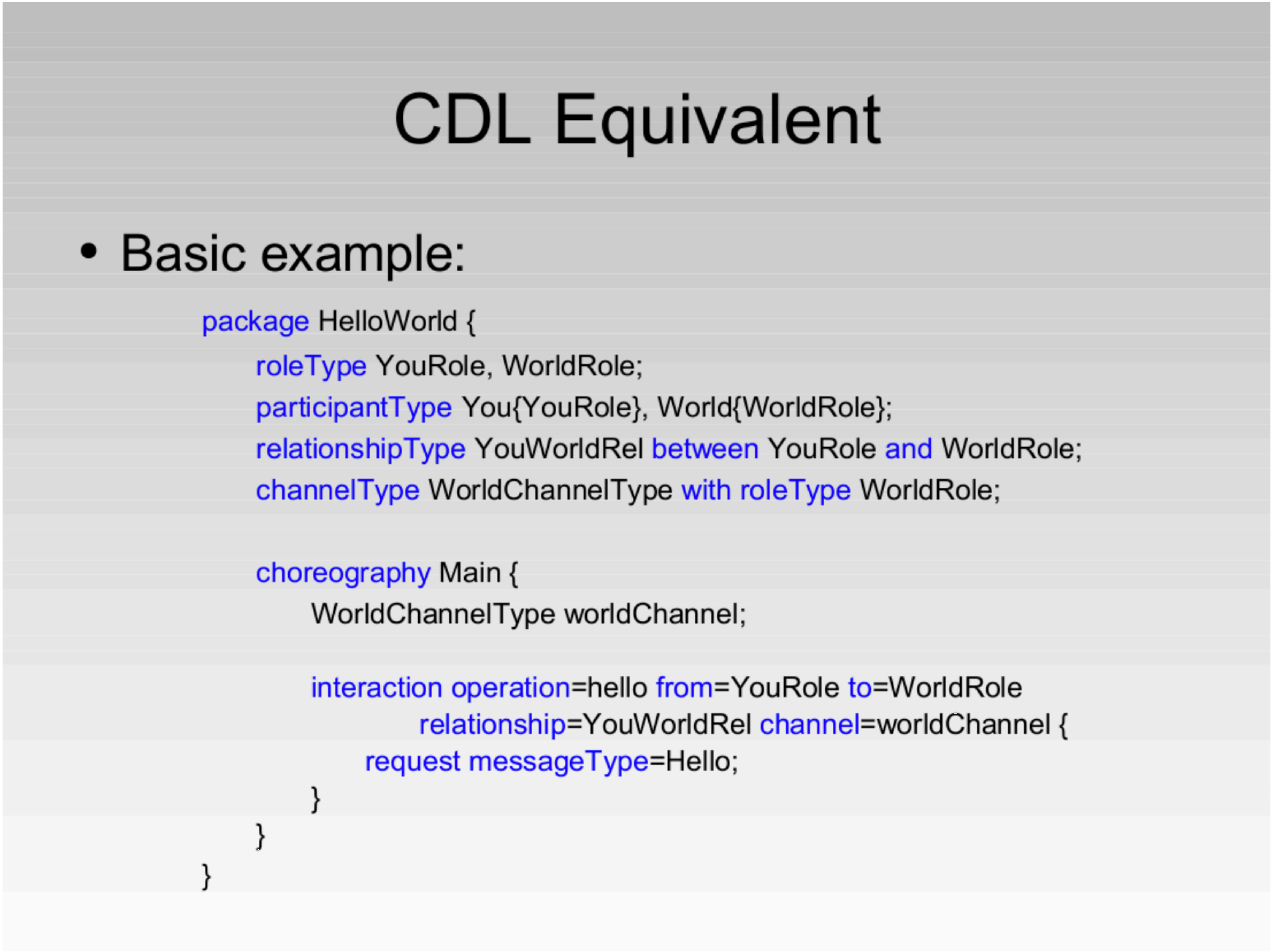
Milner, Honda and Yoshida joined W3C WS-CDL (2002)

⇓

Formalisation of W3C WS-CDL [ESOP'07]

⇓

Scribble at π4 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;
        }
    }
}
```

Dr Gary Brown (Pi4 Tech) in 2007

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 π4 Technology

⇓

Multiparty Session Types [POPL'08]

⇓

Scribble redhat

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 π4 Technology

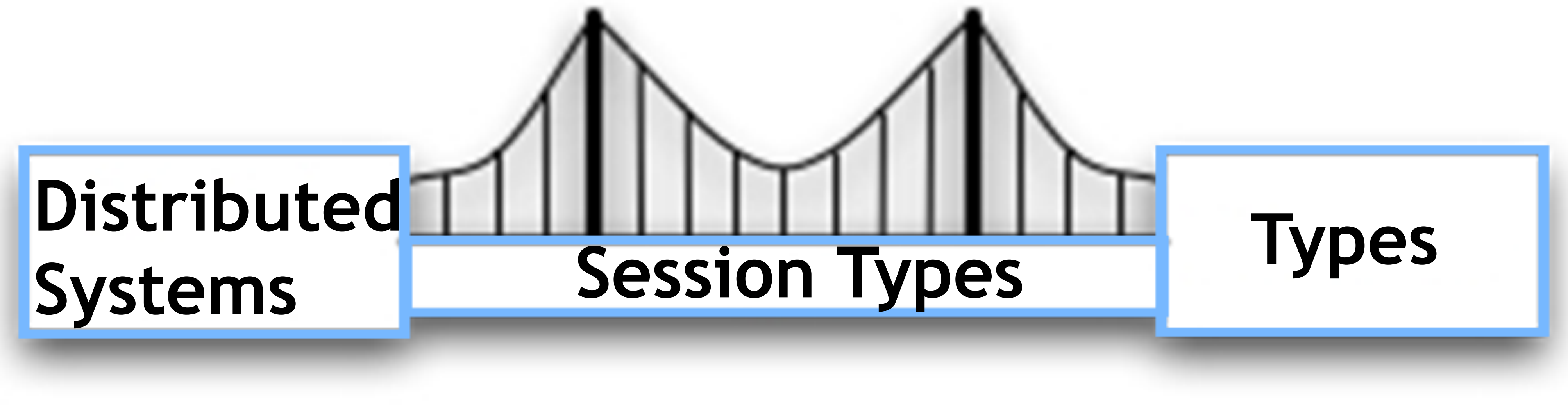
⇓

Multiparty Session Types [POPL'08]

⇓

OOI OCEAN OBSERVATORIES INITIATIVE   Scribble   redhat   SAVARA   Cognizant

Part One

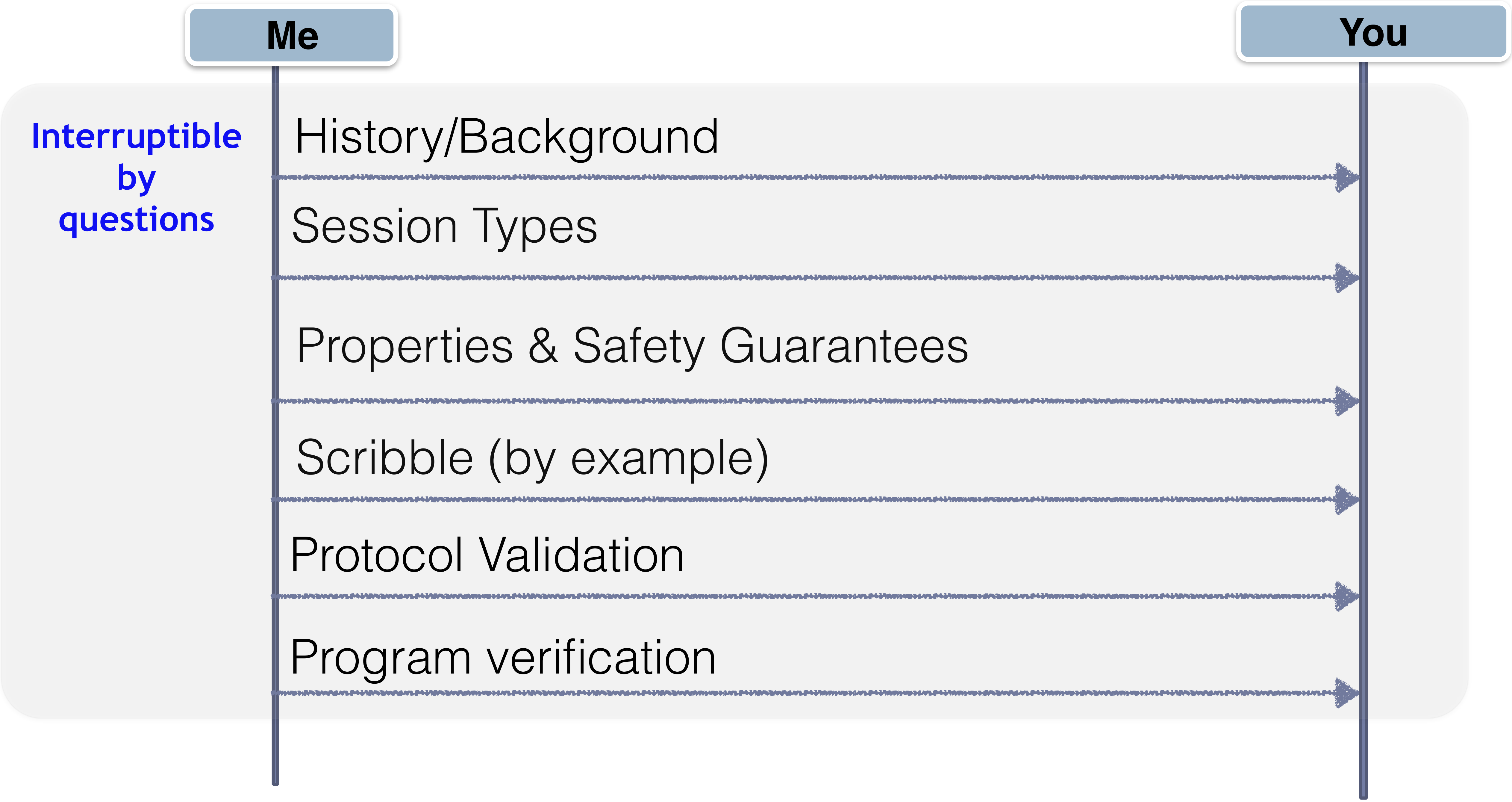


Type Me If You Can:
Introduction to Session Types and Scribble

Rumyana Neykova, Nobuko Yoshida

Content

Specification and Verification of Distributed Protocols



Me

You

Interruptible by questions

- History/Background
- Session Types
- Properties & Safety Guarantees
- Scribble (by example)
- Protocol Validation
- Program verification

Session Types

Motivation

Observation 1: Types

- One of the computing most successful concepts
- Codify the structure of the data
- Serve as a fundamental unit of compositionality
- Allow easy error prevention
- Appears from the oldest to the newest programming languages

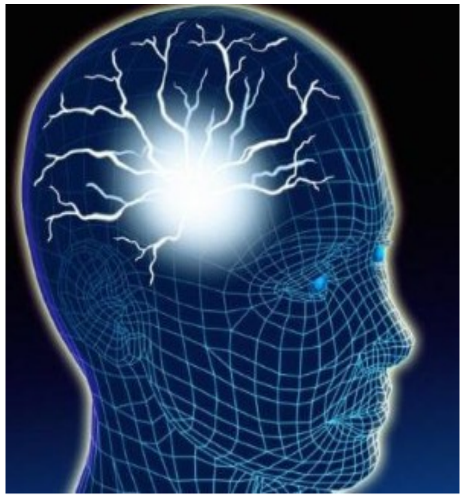
Observation 2: But distributed systems …

focus on the

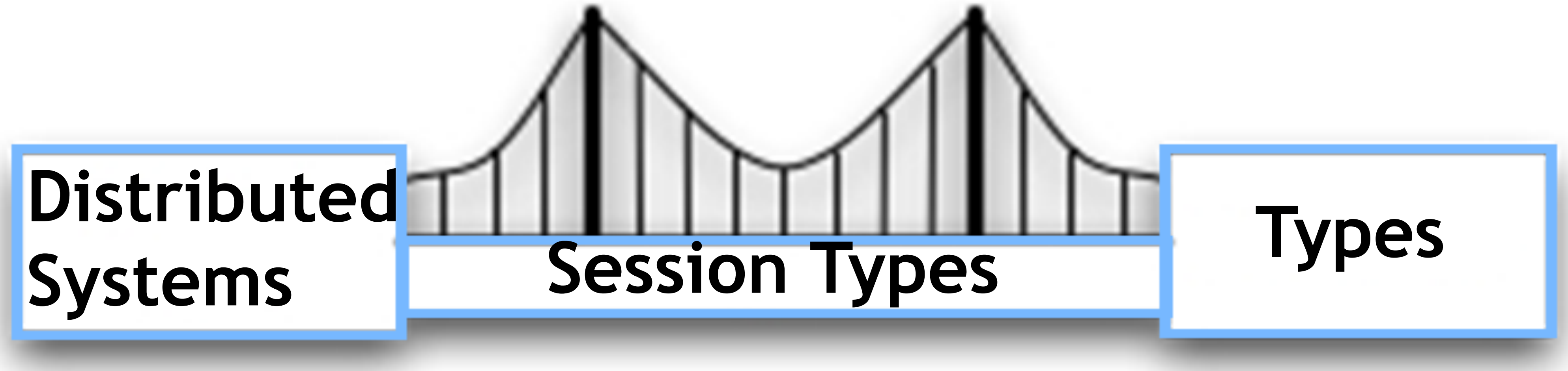
communication

not on

computation



Then…



Building blocks

- Primitives – to build the types
  - send, receive (well , there are few more, but it boils down to these two ☺️)

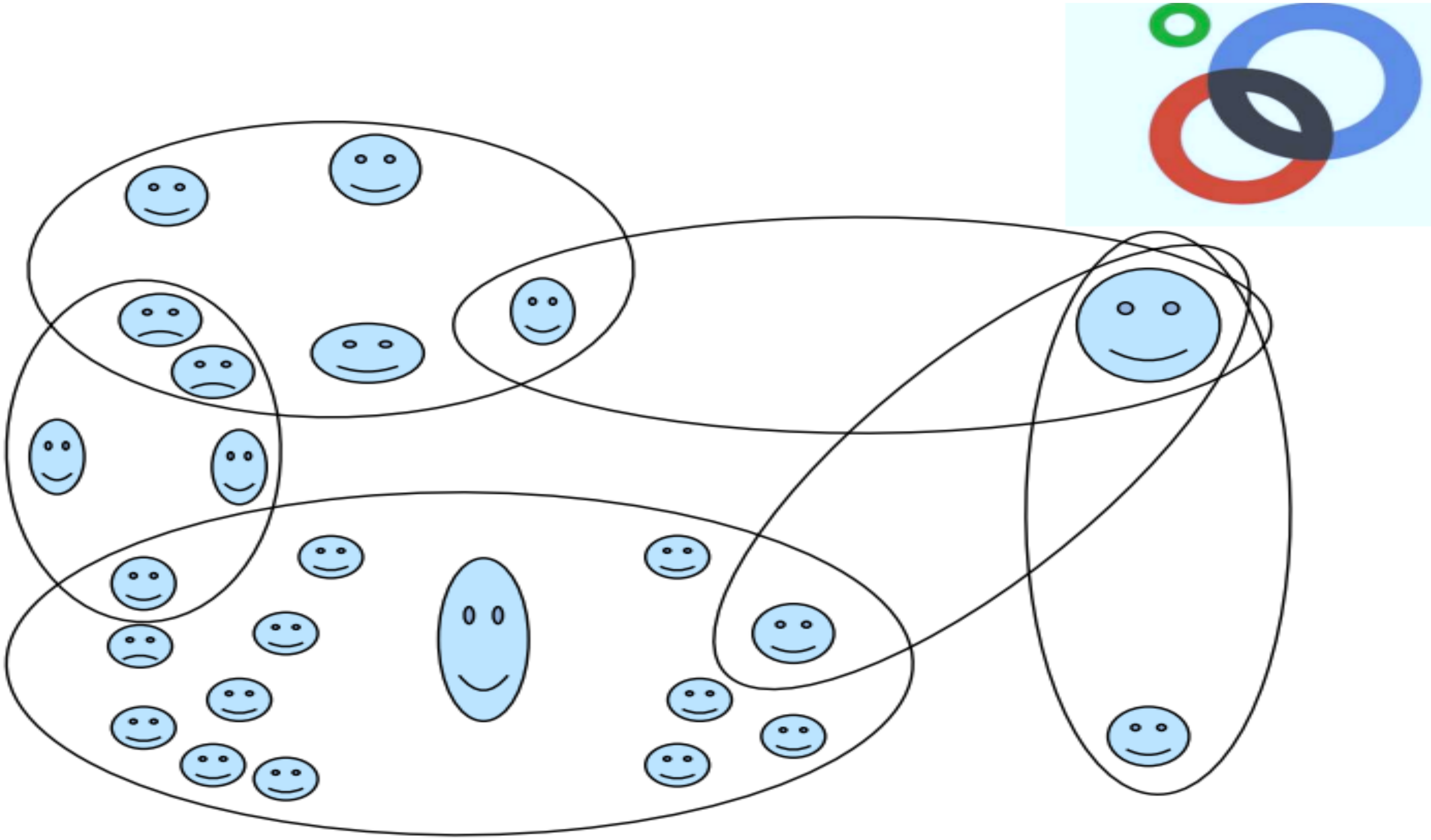
send(int).send(int).receive(bool)

- Context – to be checked by the type system
  - protocols – describe the communication between processes

SESSION= STRUCTURED SEQUENCE OF INTERACTIONS

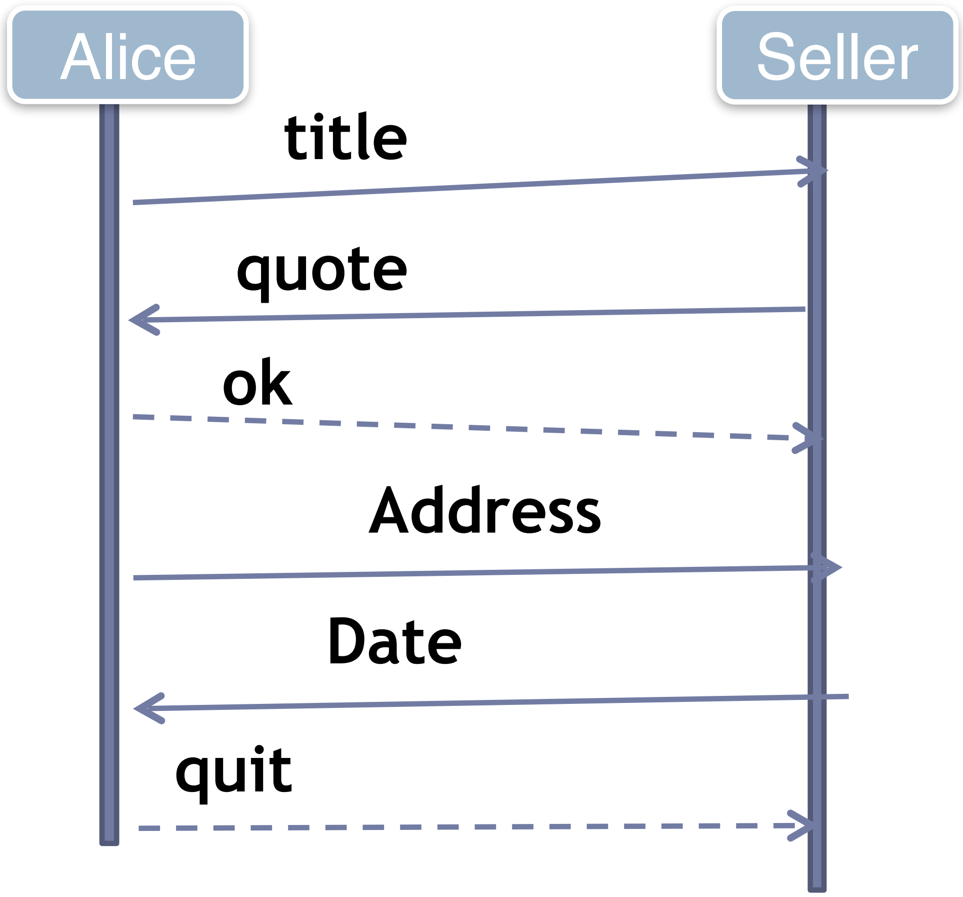
Defining the type

- Separate the communication into sessions



- Each process has a type in a session, defined by the interactions on the session channel

A Protocol



- Protocol: Buyer-Seller
- Description: Alice buying a book

send(int).receive(int).⊕{ok: send(string).receive(date), quit:end}

receive(int).send(int).&{ok: receive(string).send(date), quit: end}

Are we compatible?

send(int).send(int).receive(bool)

receive(int).receive(int).send(bool)

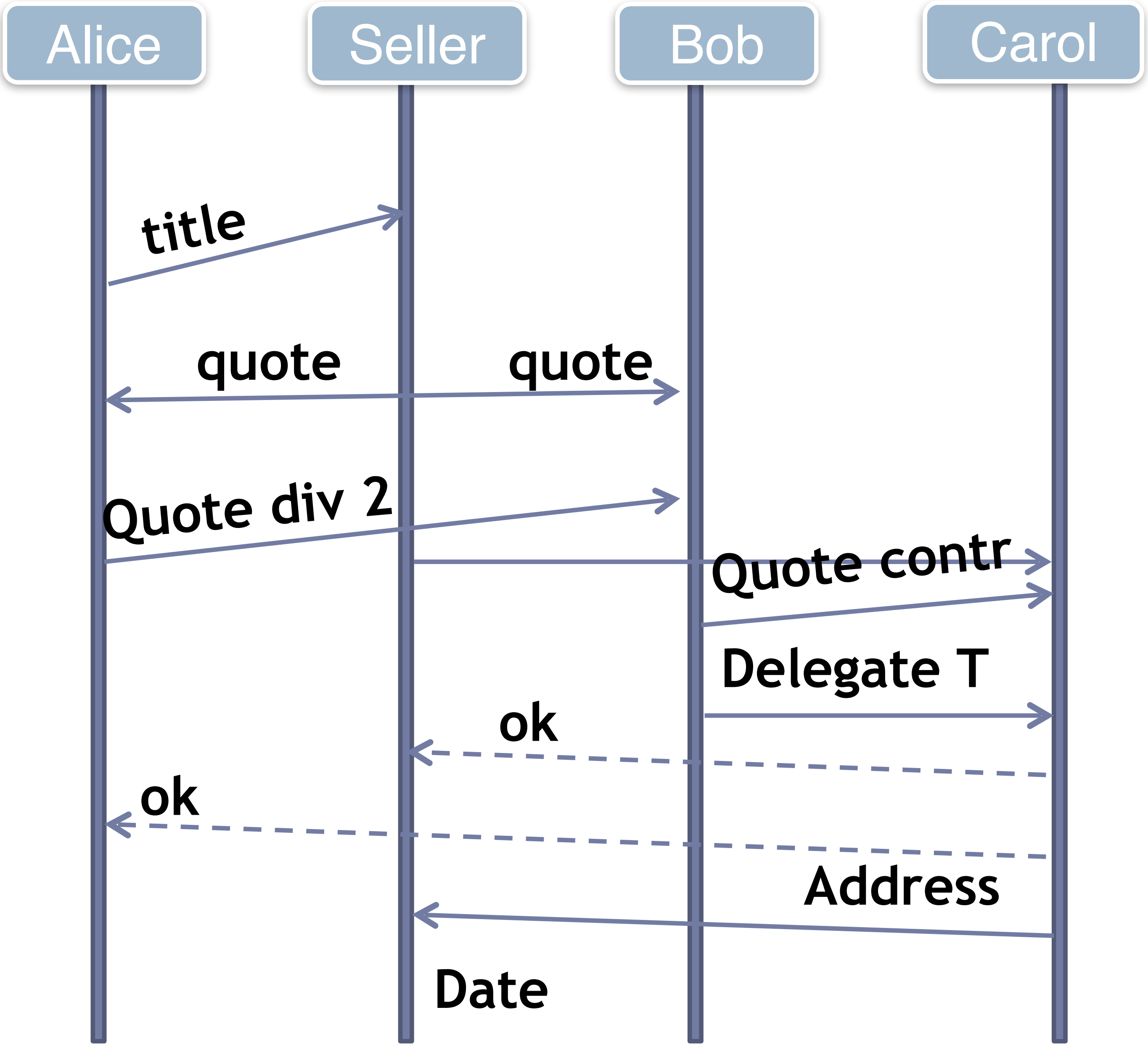
It is all about duality!

Are we compatible?

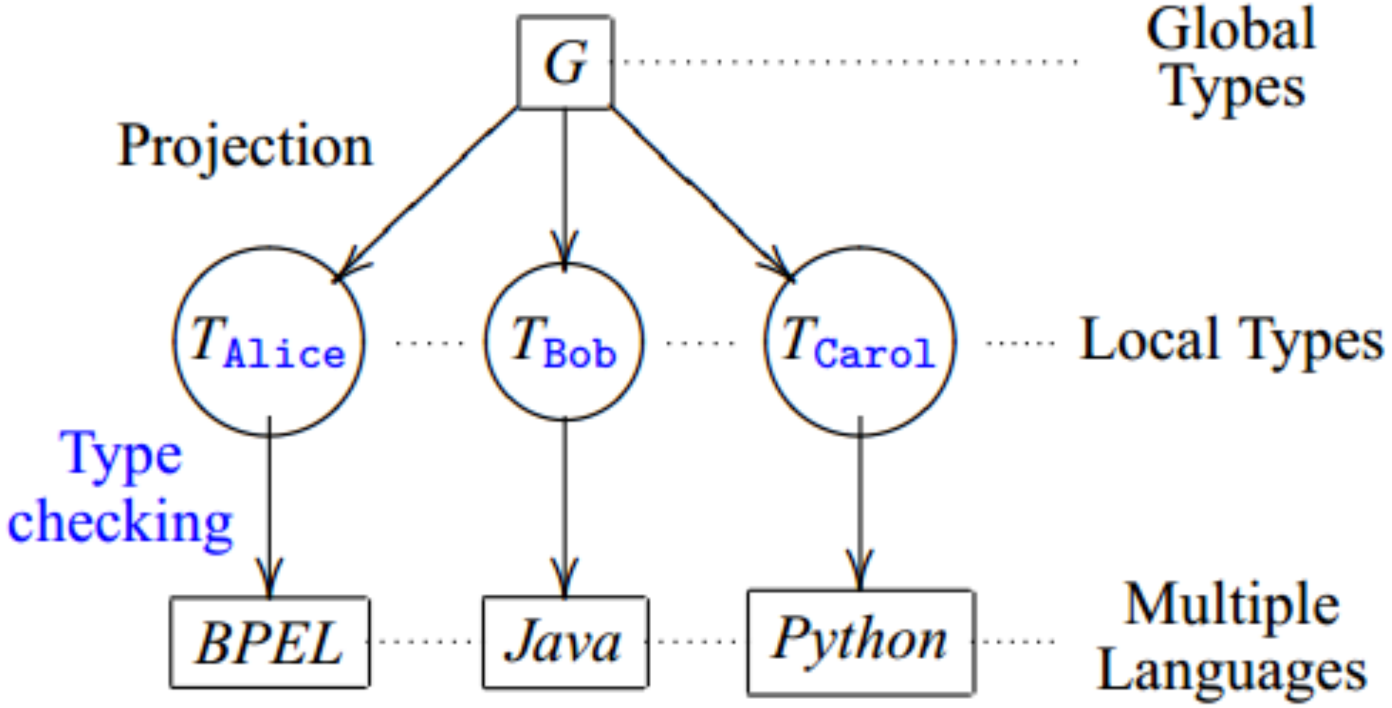
receive(int).send(int).receive(bool)

receive(int).receive(int).send(bool)

Wait a minute! What if it is more than 2?



How does it work?



$$\mathtt{Alice} \to \mathtt{Bob} : \langle \mathsf{Nat} \rangle . \; \mathtt{Bob} \to \mathtt{Carol} : \langle \mathsf{Nat} \rangle . \mathsf{end}$$

$$T_{\mathtt{Bob}} = ?\langle \mathtt{Alice}, \mathsf{Nat} \rangle ; \; !\langle \mathtt{Carol}, \mathsf{Nat} \rangle ; \mathsf{end}$$

$$P_{\mathtt{Bob}} = s?(\mathtt{Alice}, x) ; \; s!\langle \mathtt{Carol}, x \rangle ; 0$$

- Step 1: Write a Global Type
- Step 2: Write Local Programs
- Step 3: Project and Type Check Locally

Session Types in a Nutshell

SESSION = STRUCTURED SEQUENCE OF COMMUNICATION

send(int).send(int).receive(bool)

What is type safe communication?

Communication Safety

- No communication mismatch

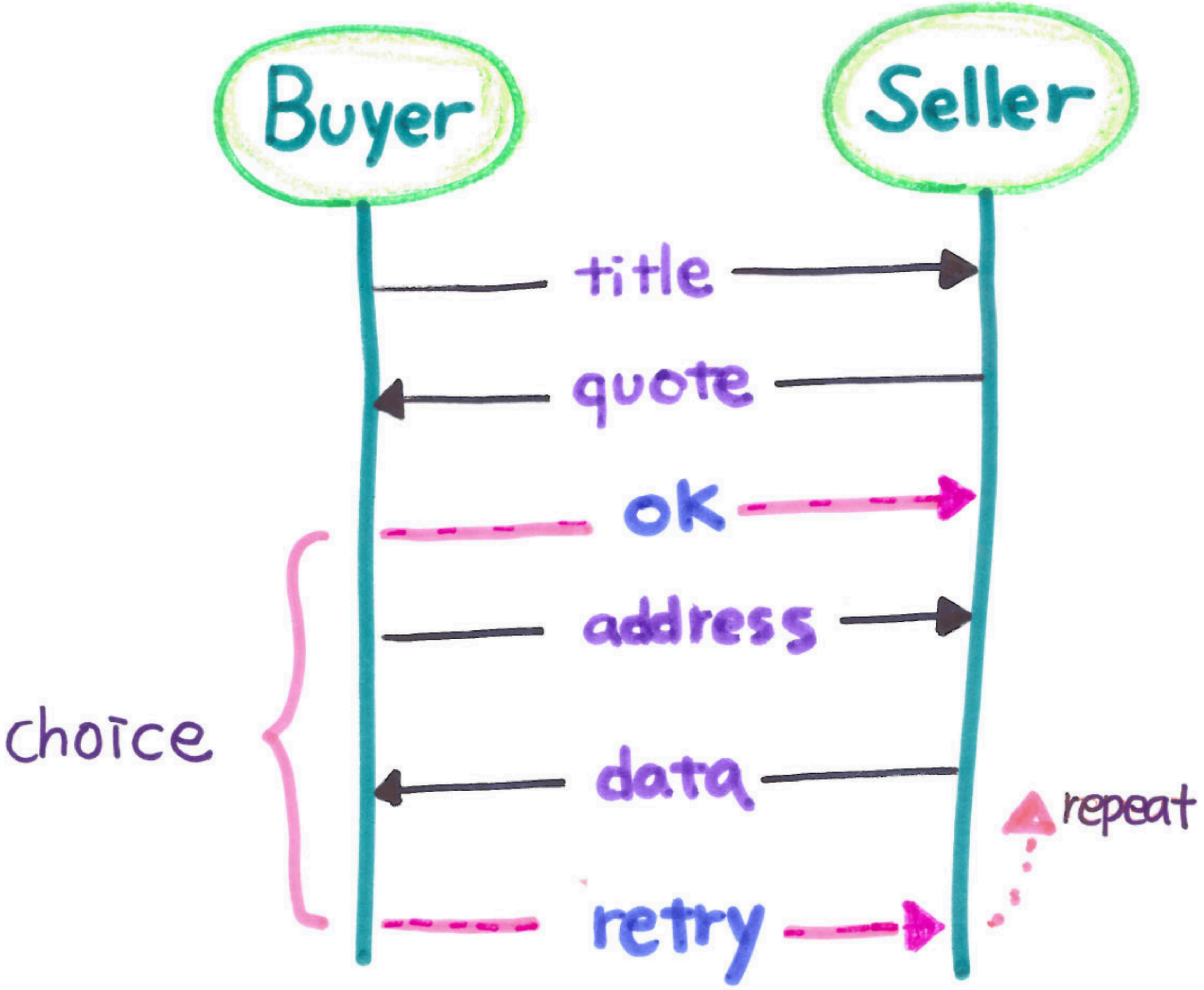
Session Fidelity

- Communication follow the described protocol

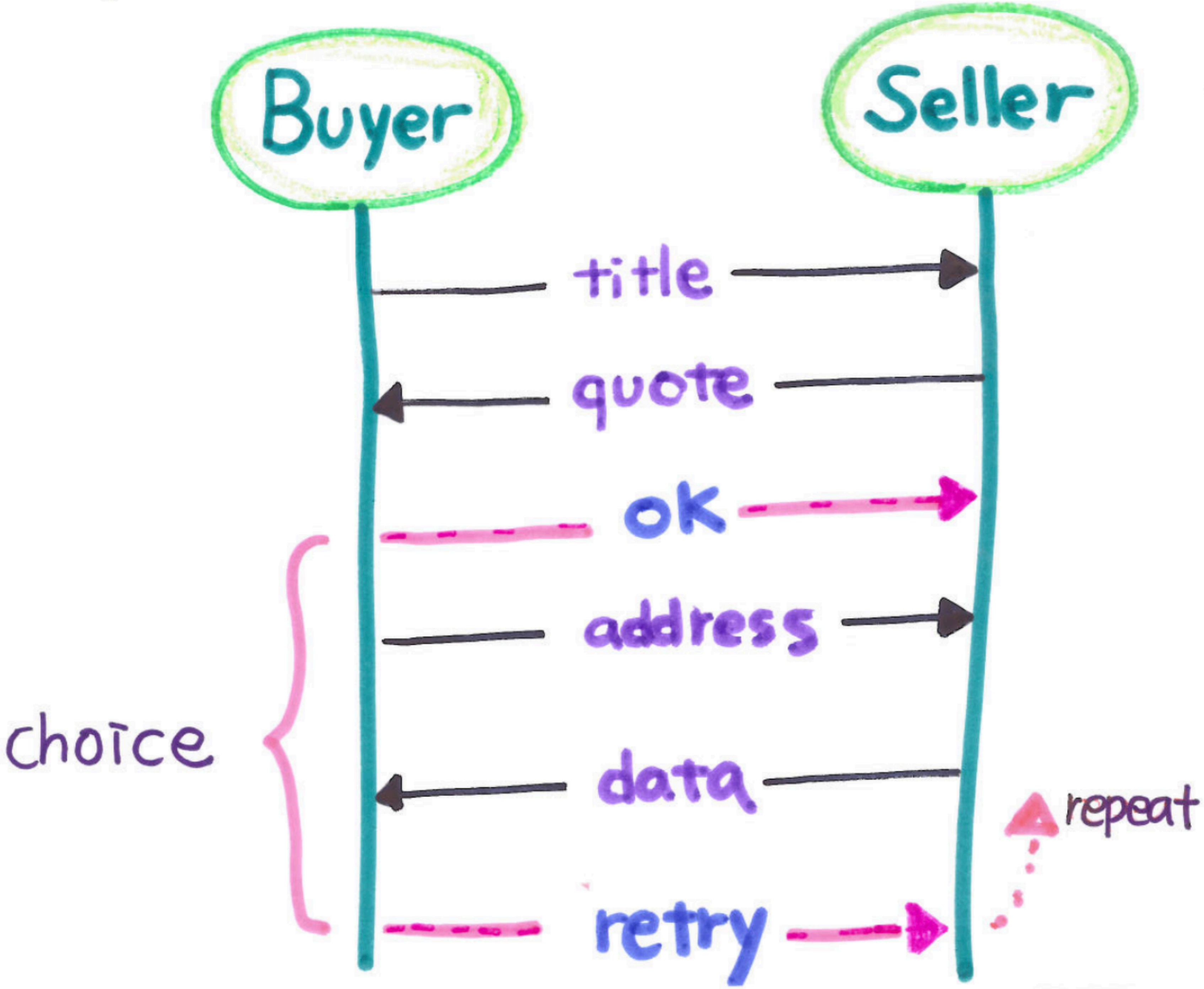
Progress

- No deadlock/ stuck in a session

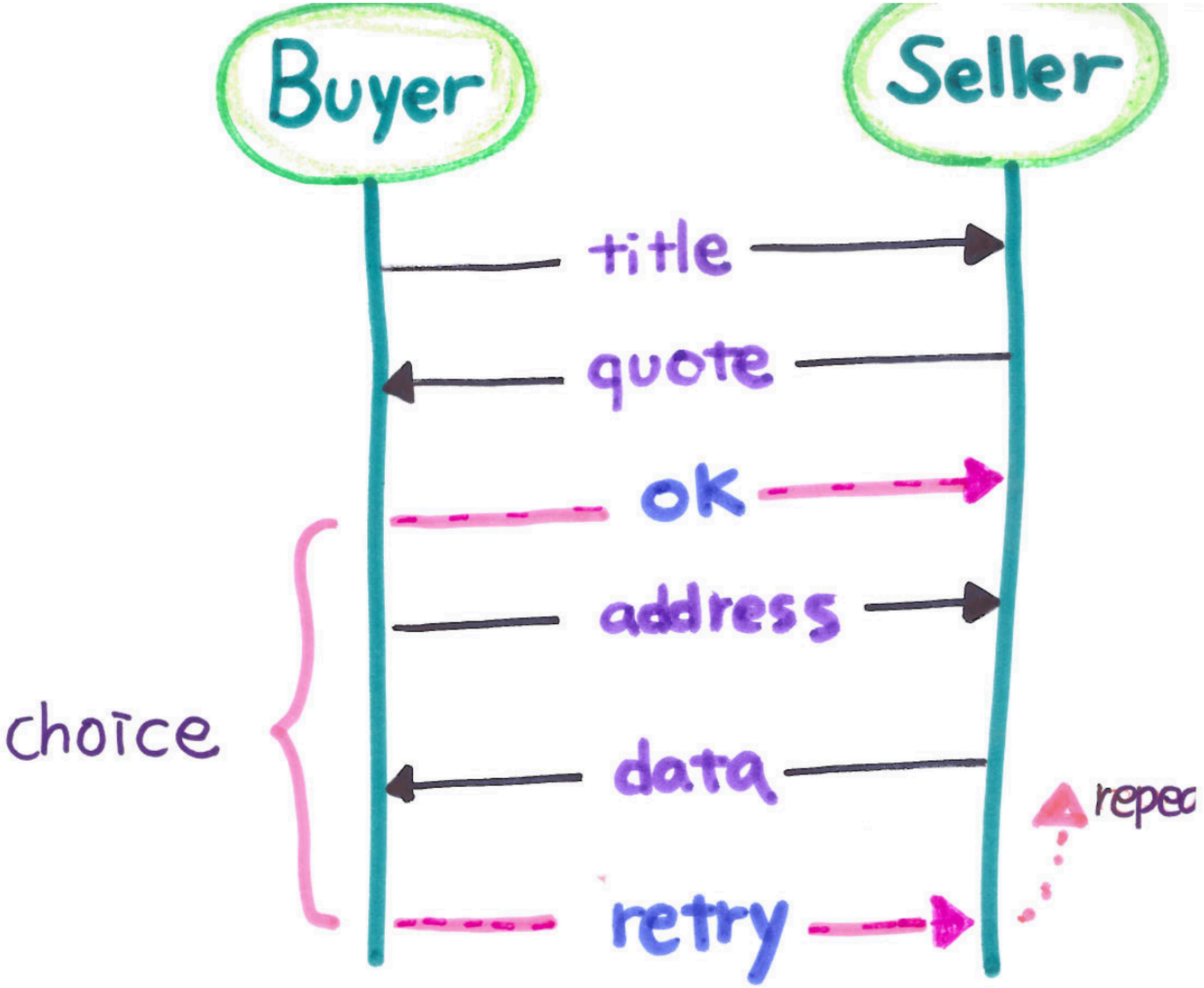
Binary Session Types: Buyer-Seller Protocol



Binary Session Types: Buyer-Seller Protocol



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



P has T

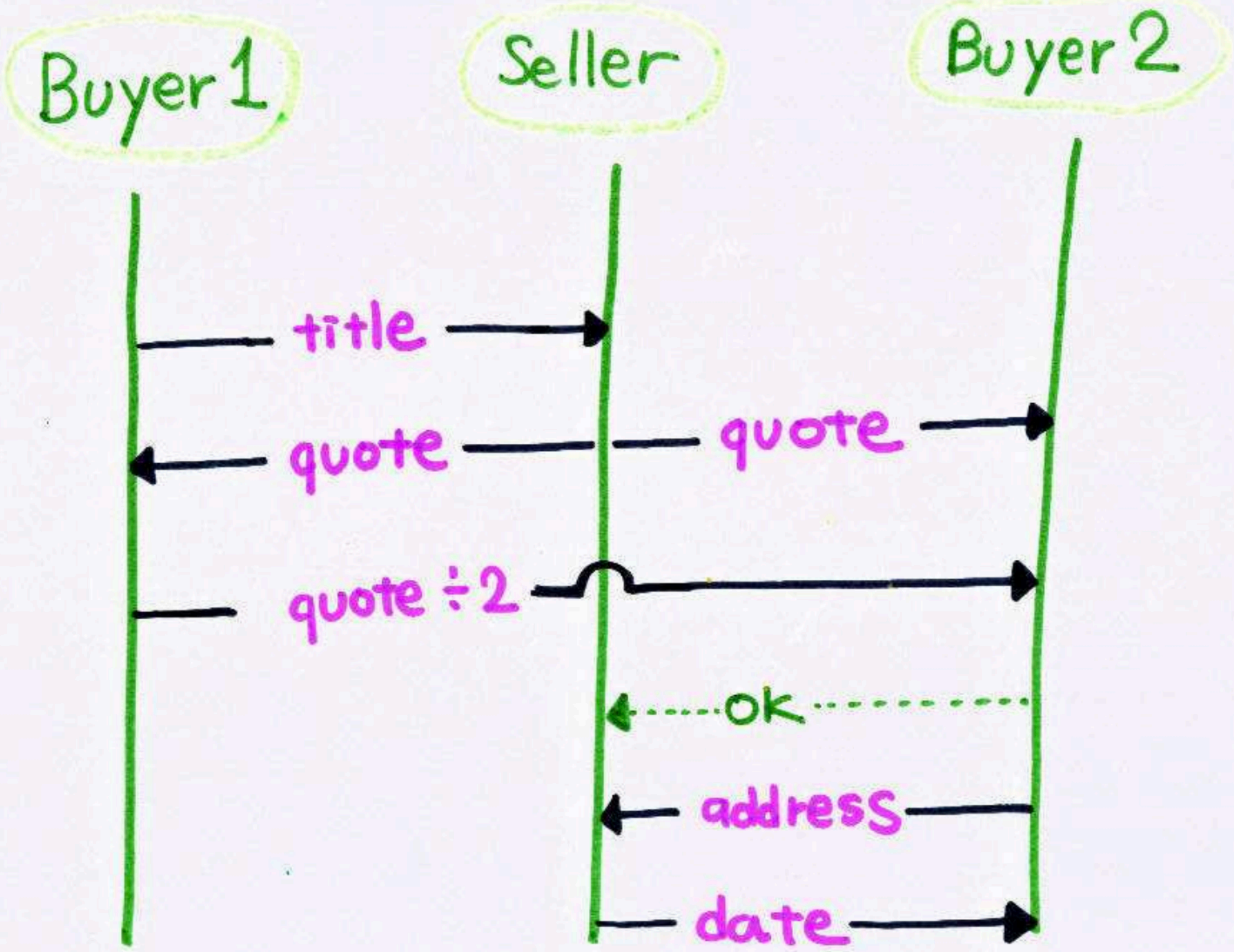
Q has $\overline{T}$ dual

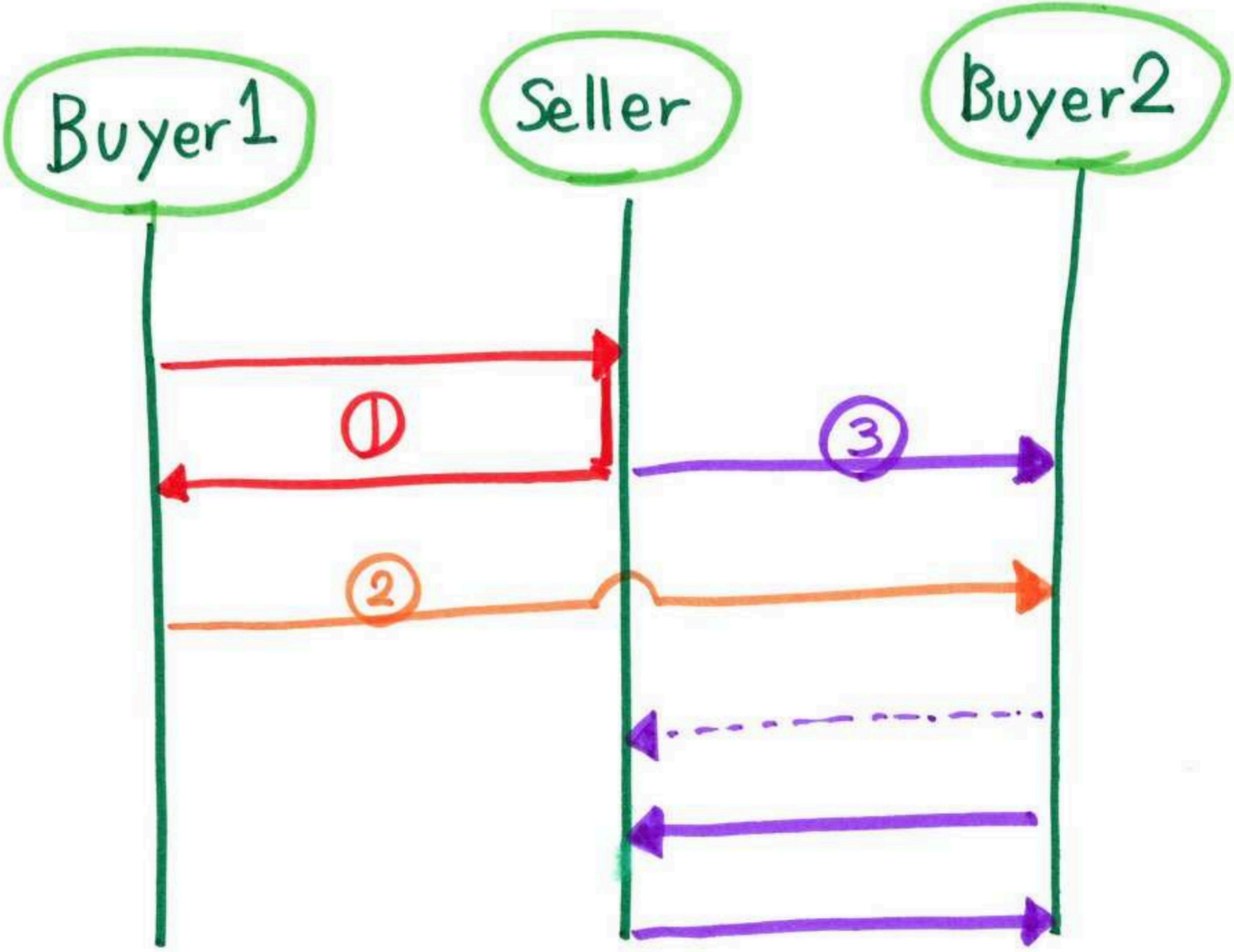
P | Q typable

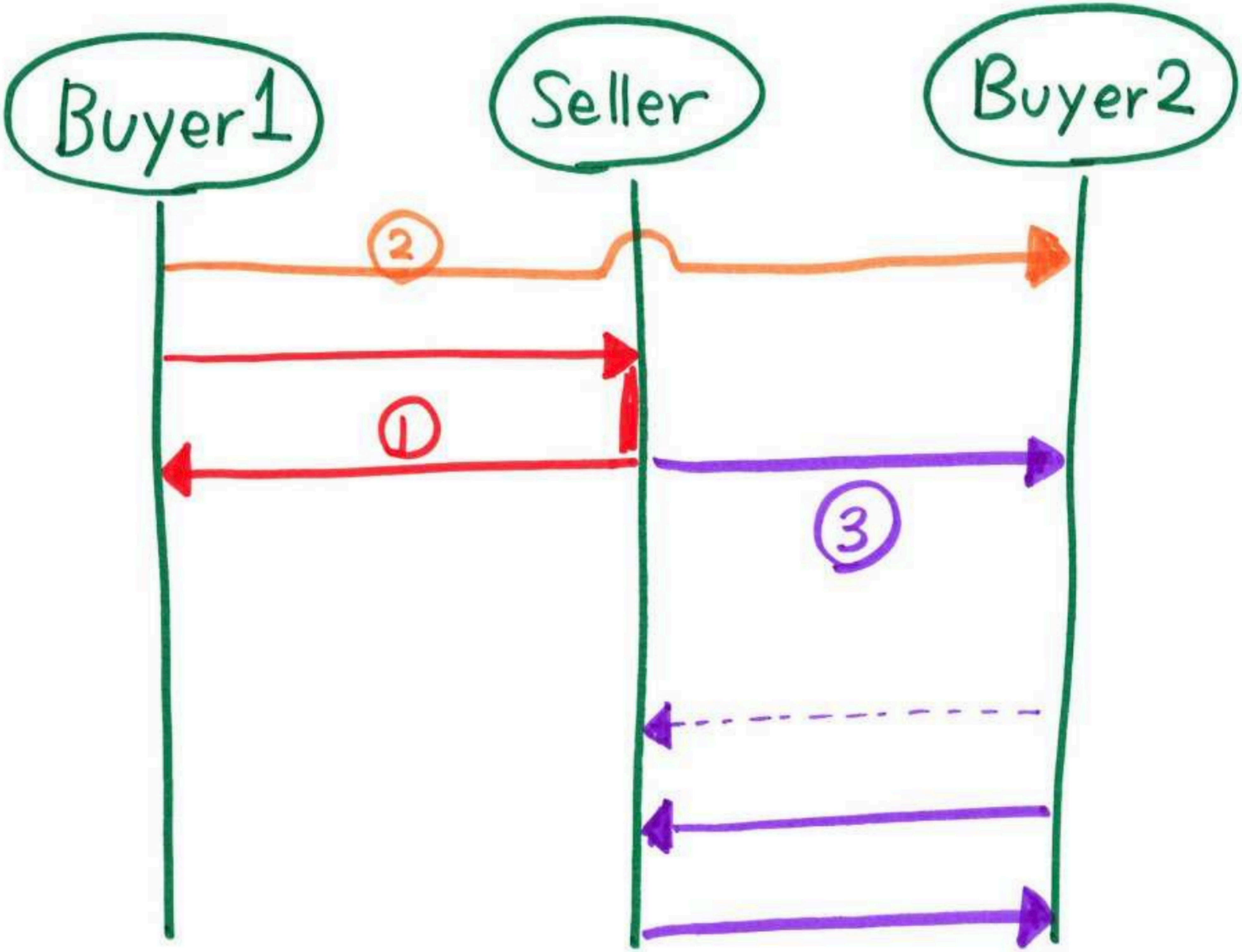
μt. ! Title ; ? Quote ; ! {ok: ! Add ; ? Date , retry : t }

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

Multiparty Session Types



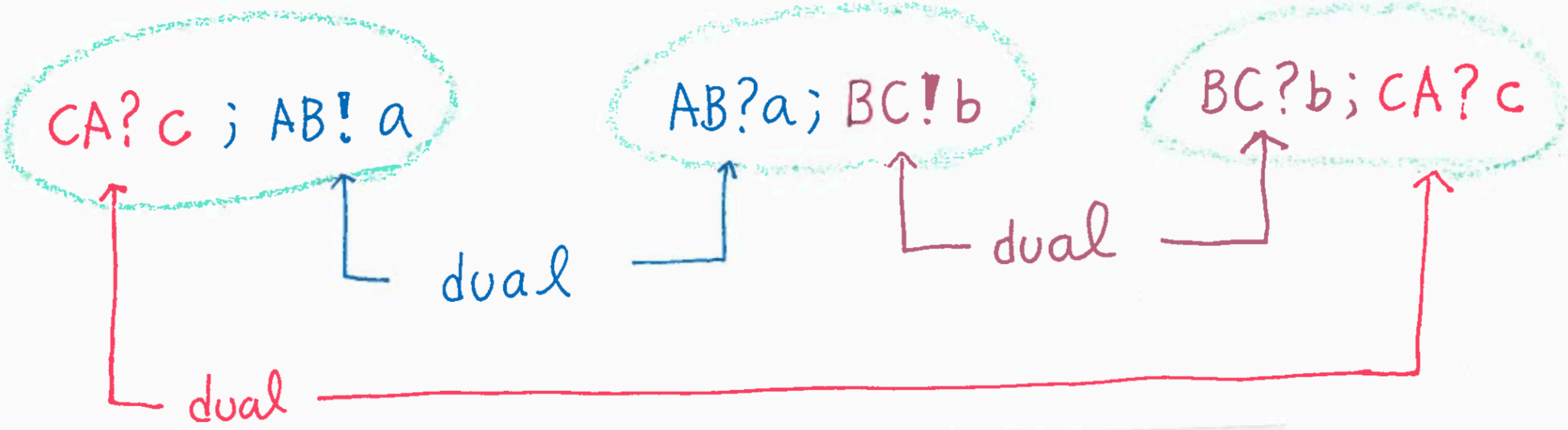

Buyer1
Seller
Buyer2
①
③
②


Buyer1
Seller
Buyer2
2
1
3

Alice

Bob

Carol



CA?c ; AB!a

AB?a ; BC!b

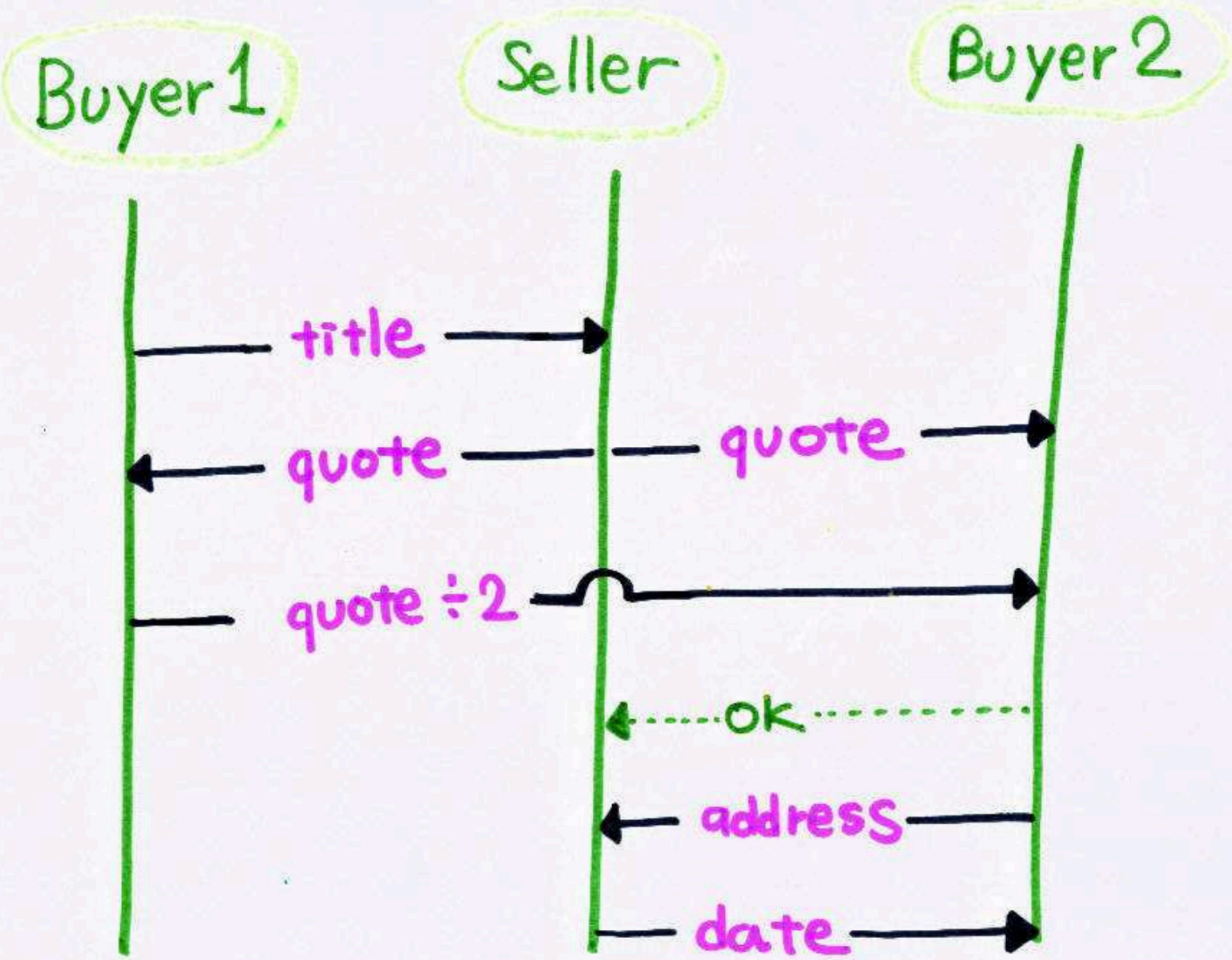
BC?b ; CA?c

3 dual pairs

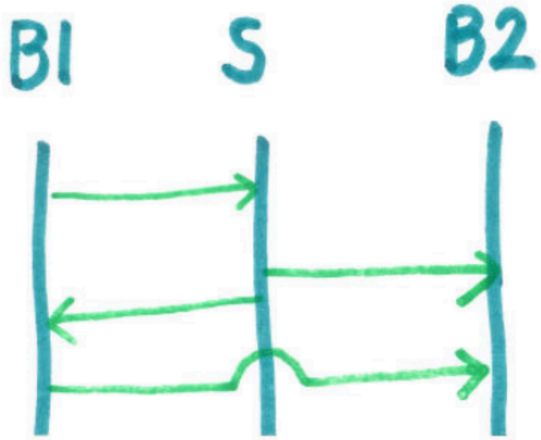
If you use binary Session Types....

Deadlock!

Multiparty Session Types



Multiparty Session Types [Honda, Yoshida, Carbone 2008]



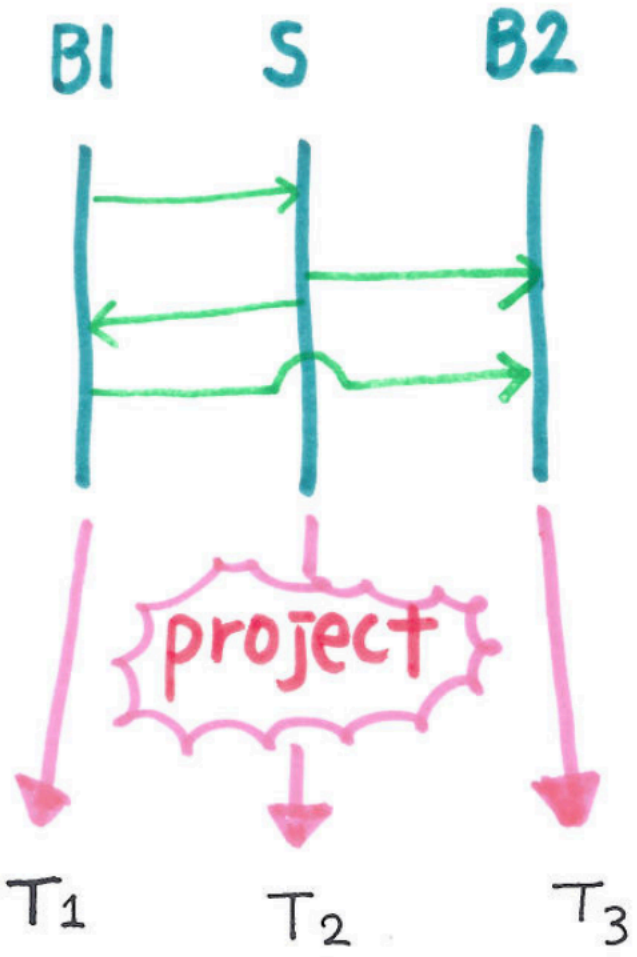
Ⓖ B1 → S Int.

S → B2 Char

STEP 1

Write Global Type

Multiparty Session Types [Honda, Yoshida, Carbone 2008]



G $B1 \to S$ Int.

$S \to B2$ Char

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

STEP 1

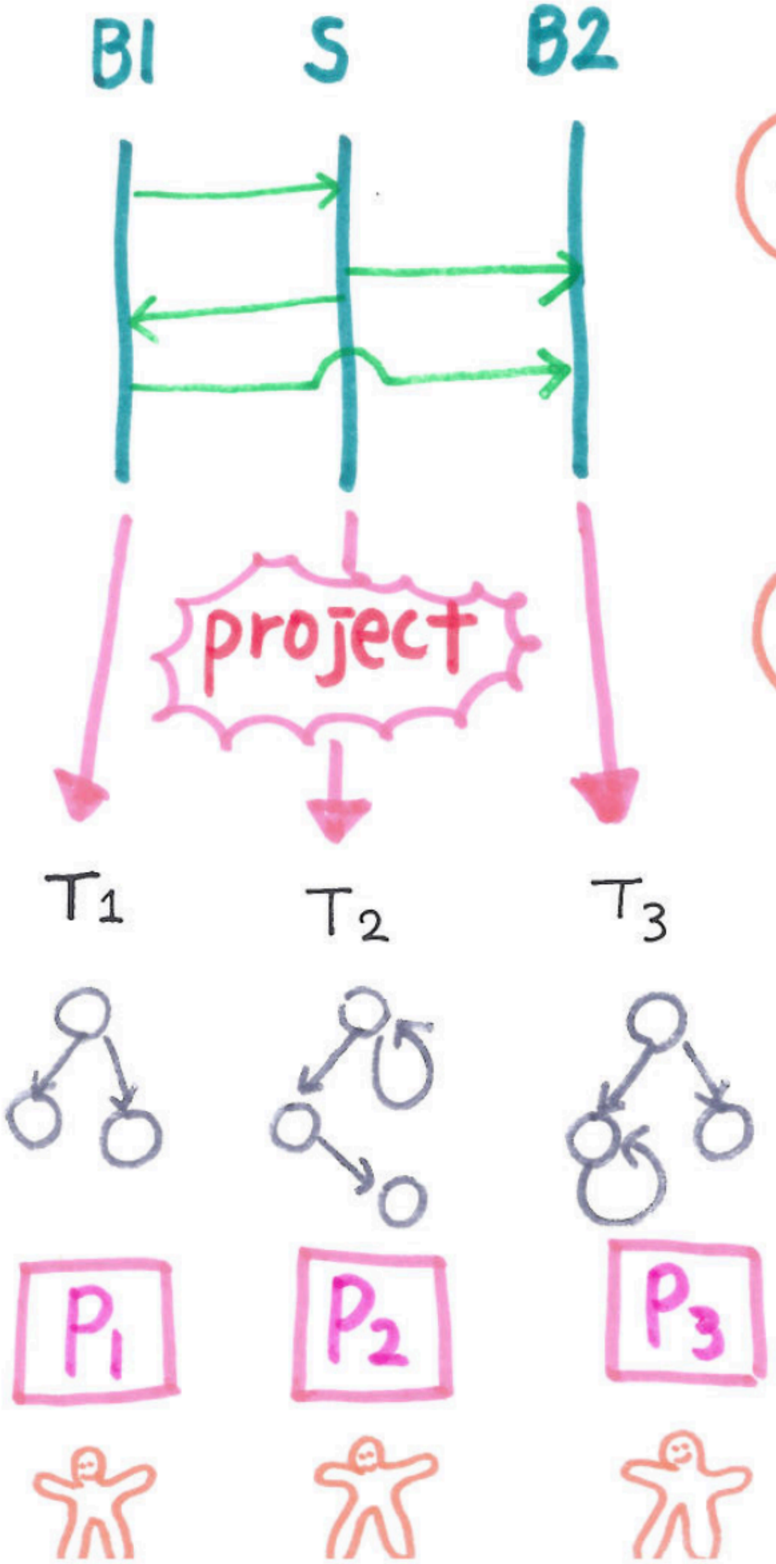
Write Global Type

STEP 2

Project to Local Types

Multiparty Session Types

[Honda, Yoshida, Carbone 2008]



Ⓖ $B1 \to S$ Int.

$S \to B2$ Char

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

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

STEP 1

Write Global Type

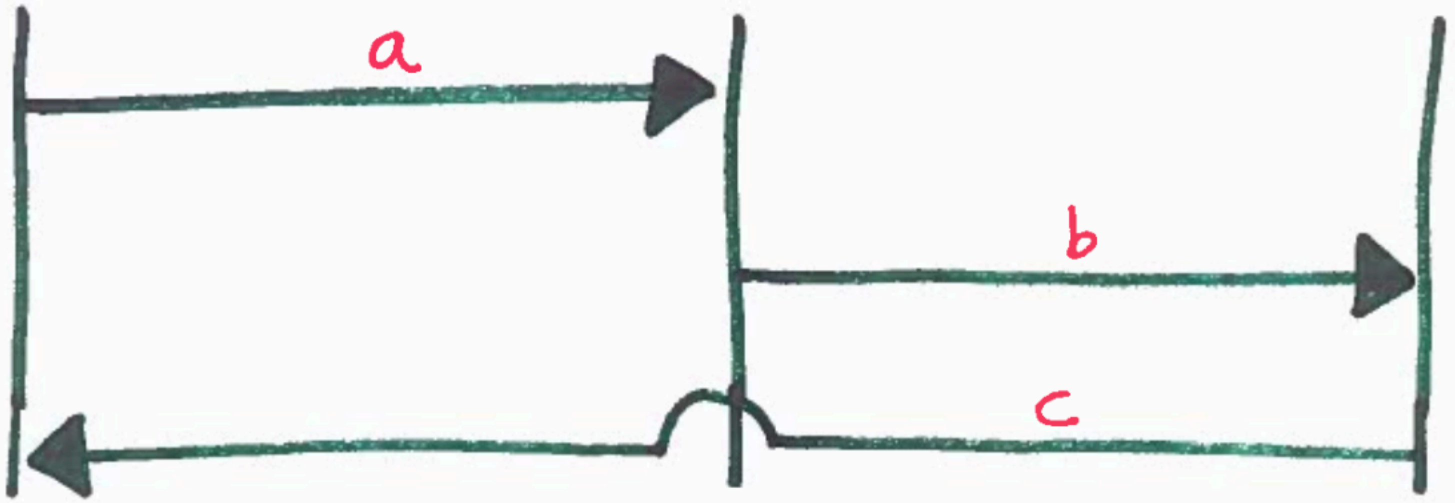
STEP 2

Project to Local Type

STEP 3

- Static Check
- Generate Code
- Run-time check

Alice Bob Carol



Global Type

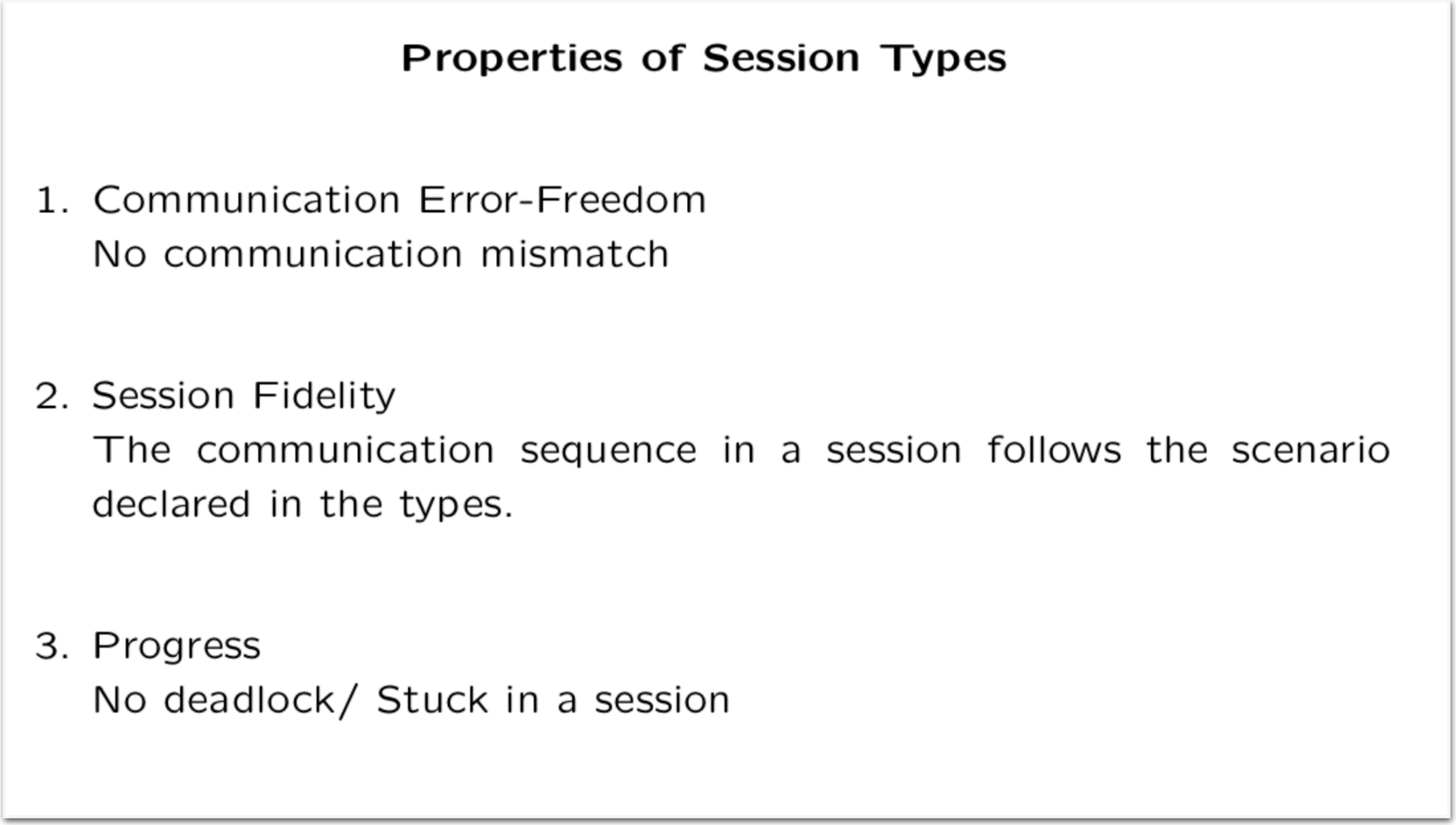
Alice AB!a ; CA?c

Bob AB?a; BC!b

Carol BC?b; CA!c;

LOCAL TYPES

No Deadlock



Properties of Session Types

1. Communication Error-Freedom
   No communication mismatch

2. Session Fidelity
   The communication sequence in a session follows the scenario declared in the types.

3. Progress
   No deadlock/ Stuck in a session

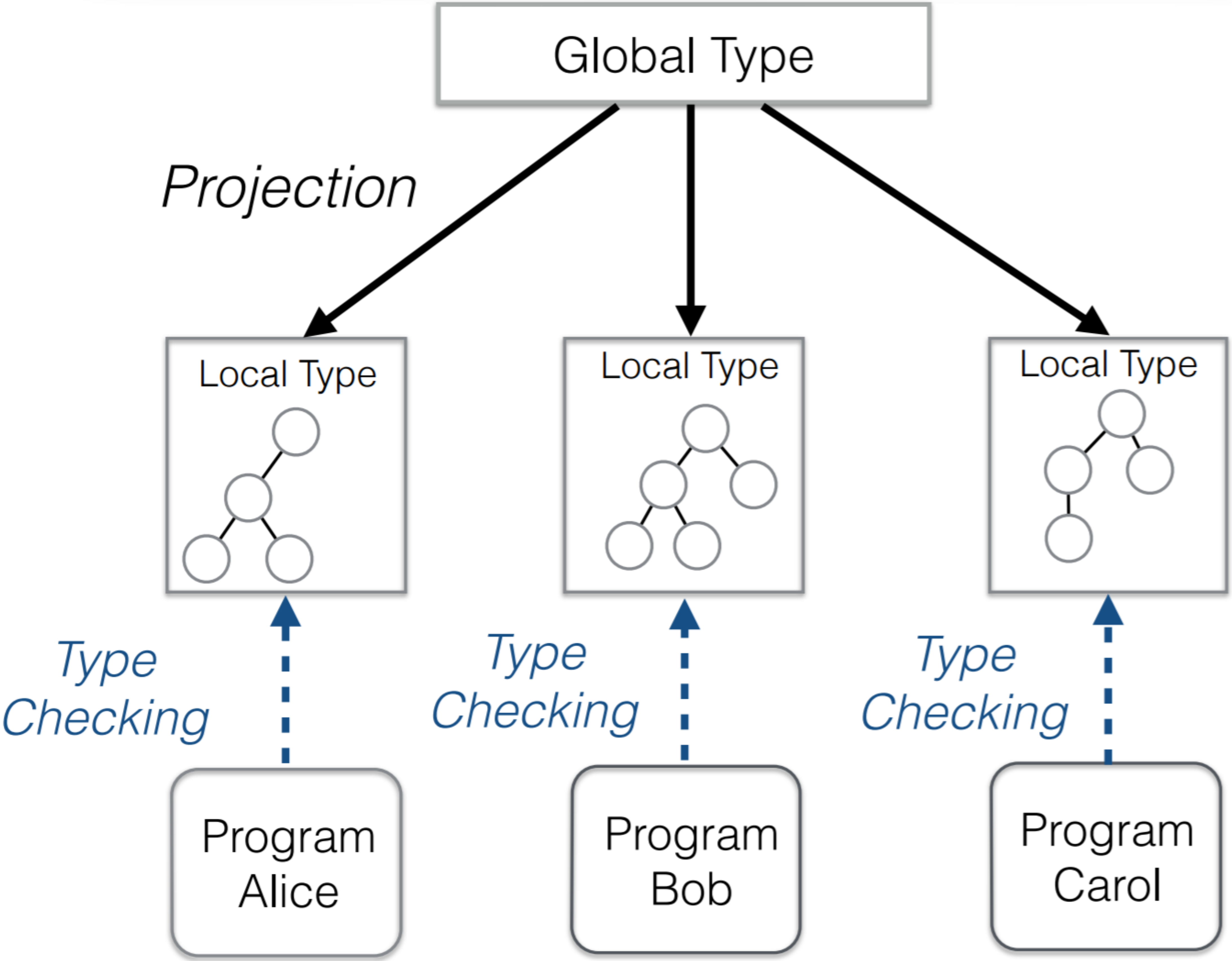
“well-typed **channels** cannot go wrong”

Session Types

Applications

Type Checking

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



Global Type

Projection

Local Type

Local Type

Local Type

Type Checking

Type Checking

Type Checking

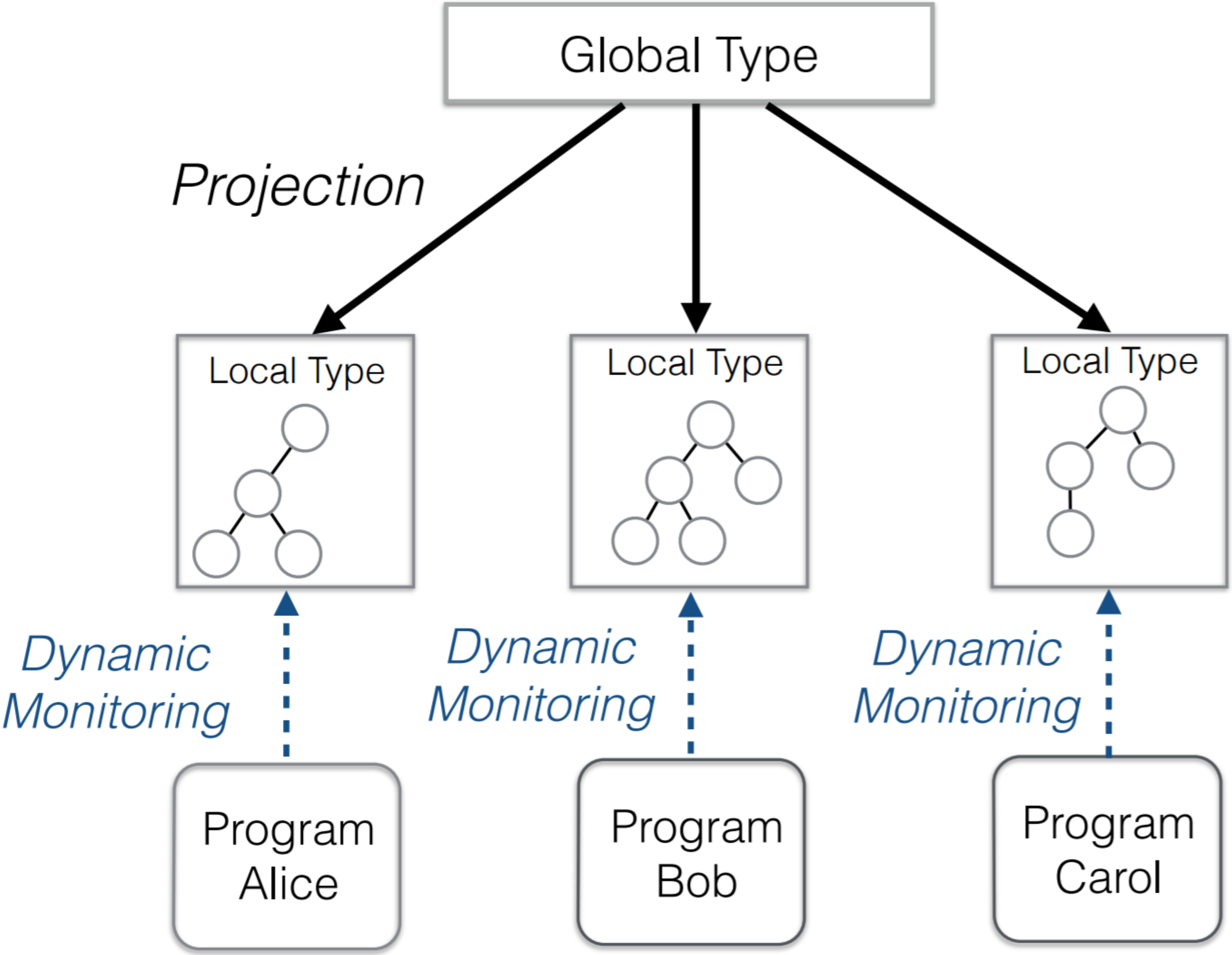
Program Alice

Program Bob

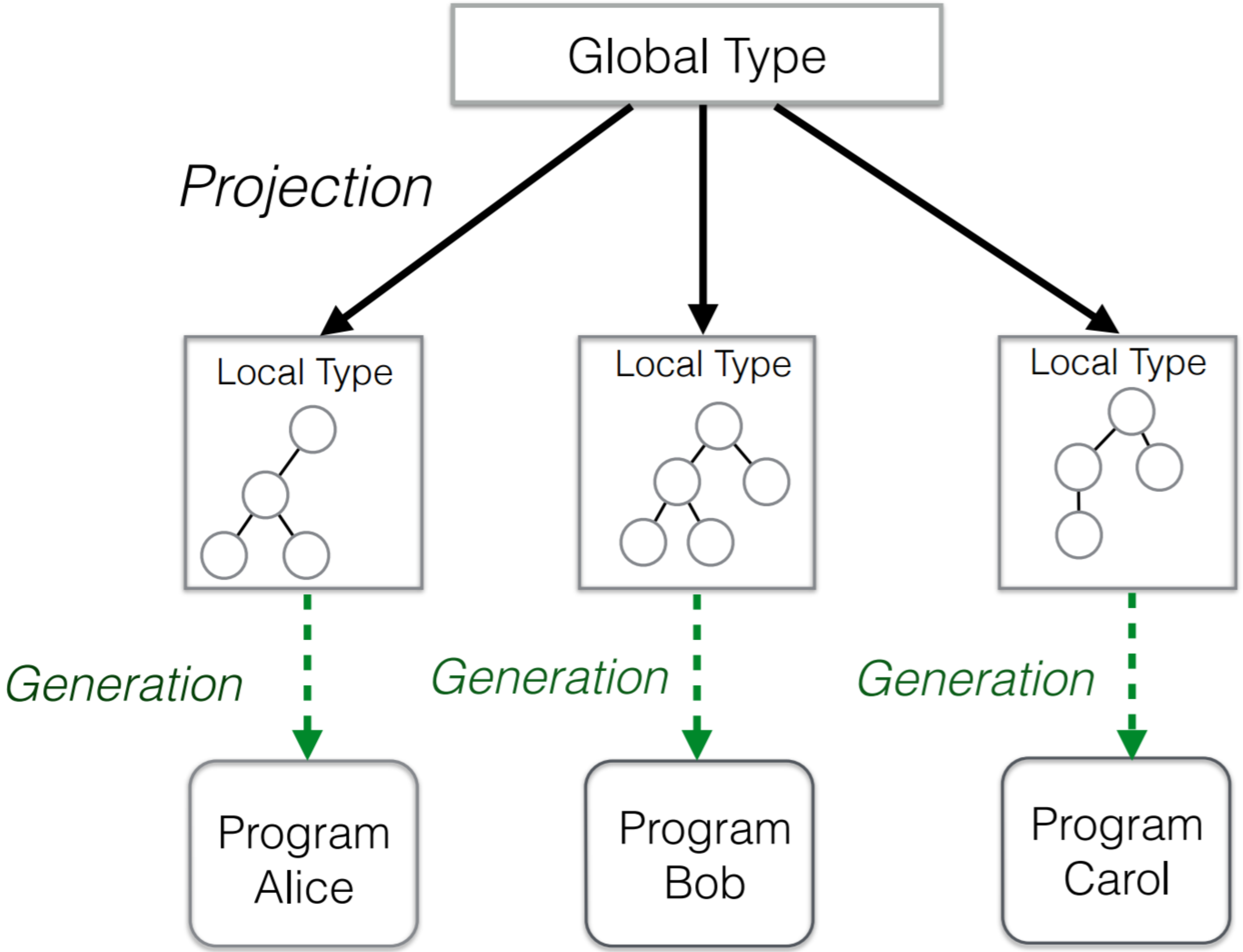
Program Carol

Dynamic Monitoring

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

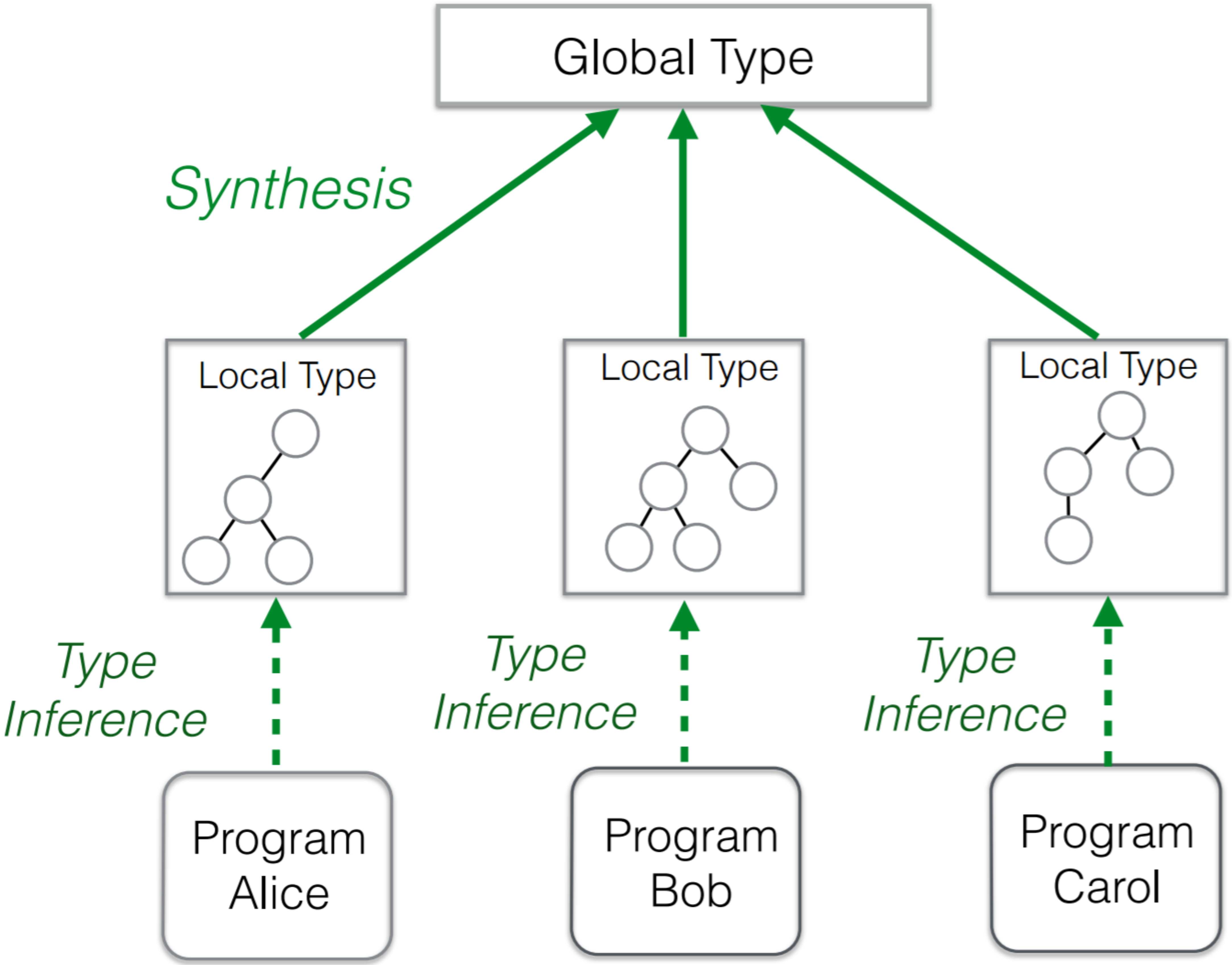


Code Generation [CC'15, FASE'16]



Synthesis

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



MPST

Scribble

- Applications
  - Deadlock Detection (Go)
  - Recovery strategies(Erlang)
  - Type-driven programming (Java, Scala, F#)
  - Static Verification (C, OCaml, Rust)
  - Runtime monitoring (Python)

F#
GO
Scala
Session C
Java
ERLANG
RabbitMQ
Messaging that just works
python
AMQP
Advanced Message Queuing Protocol
OOI
OCEAN OBSERVATORIES INITIATIVE

Applications

Scribble

F#

Scala

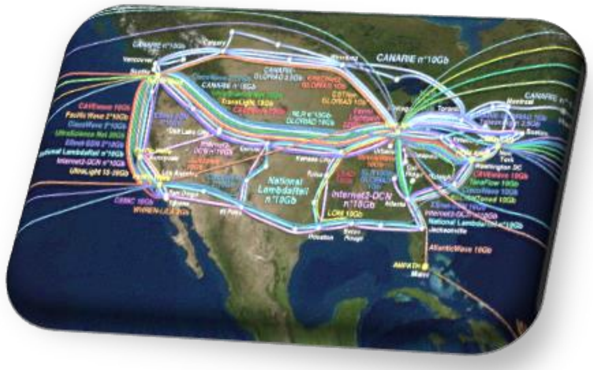
Session C

Java

ERLANG

RabbitMQ™

Messaging that just works



python™

AMQP

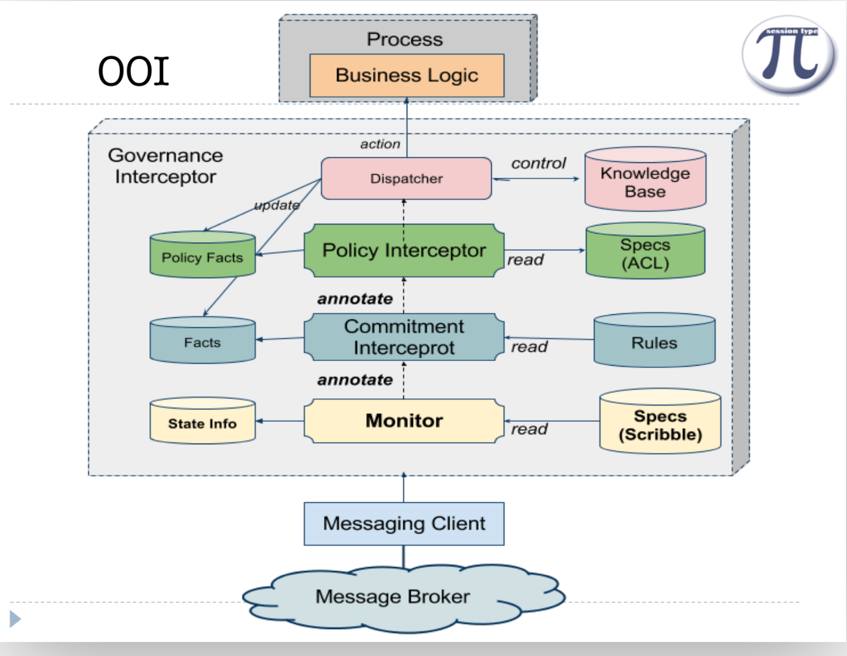
Advanced Message Queuing Protocol

OOI

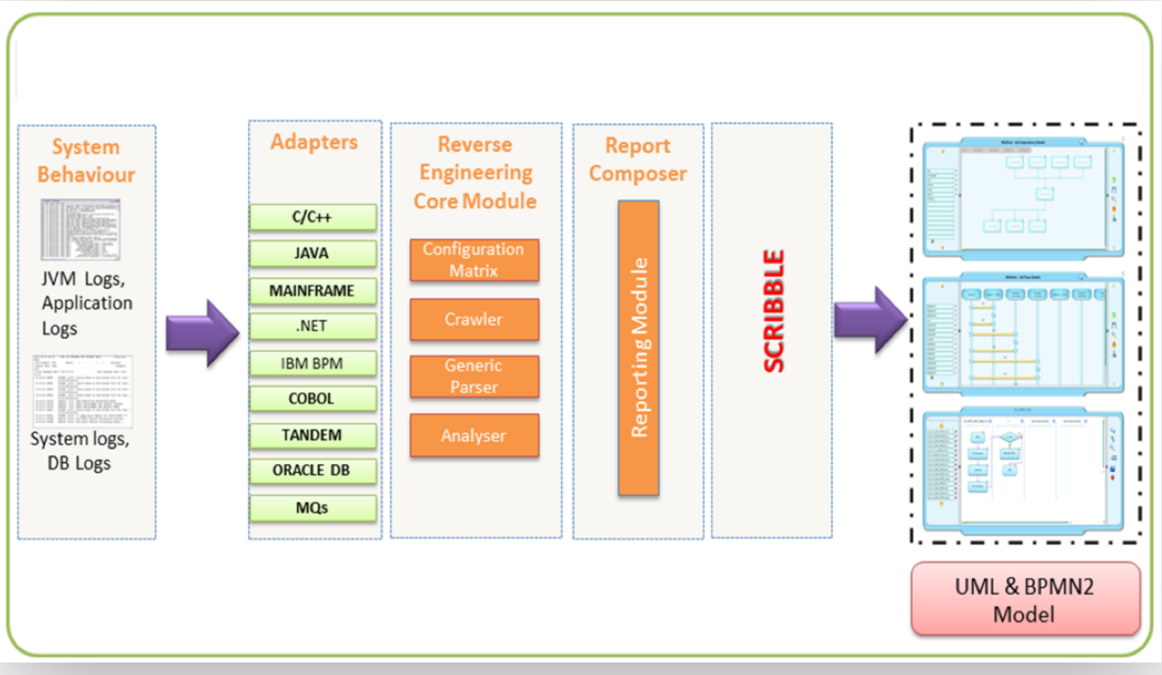
OCEAN OBSERVATORIES INITIATIVE

Session Type Based Tools

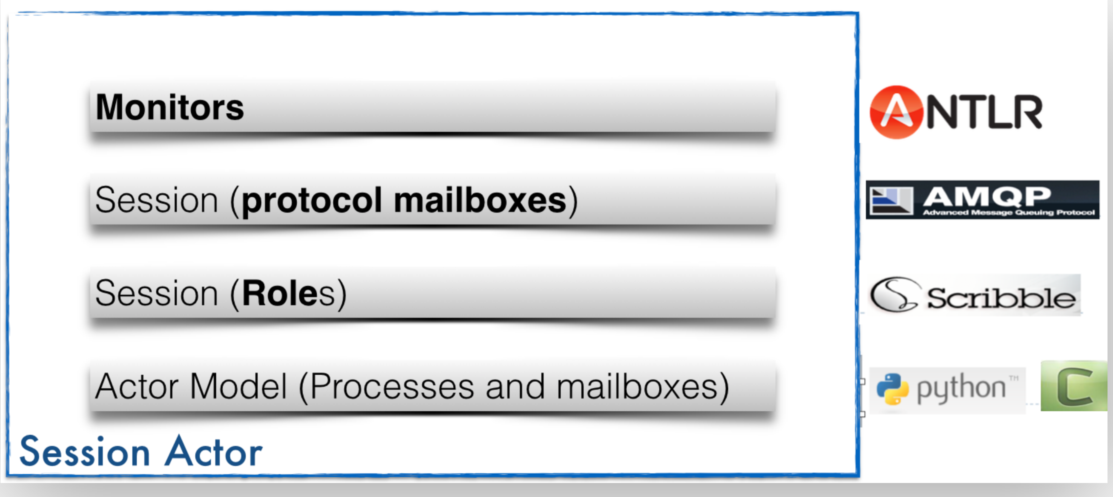
OOI Governance



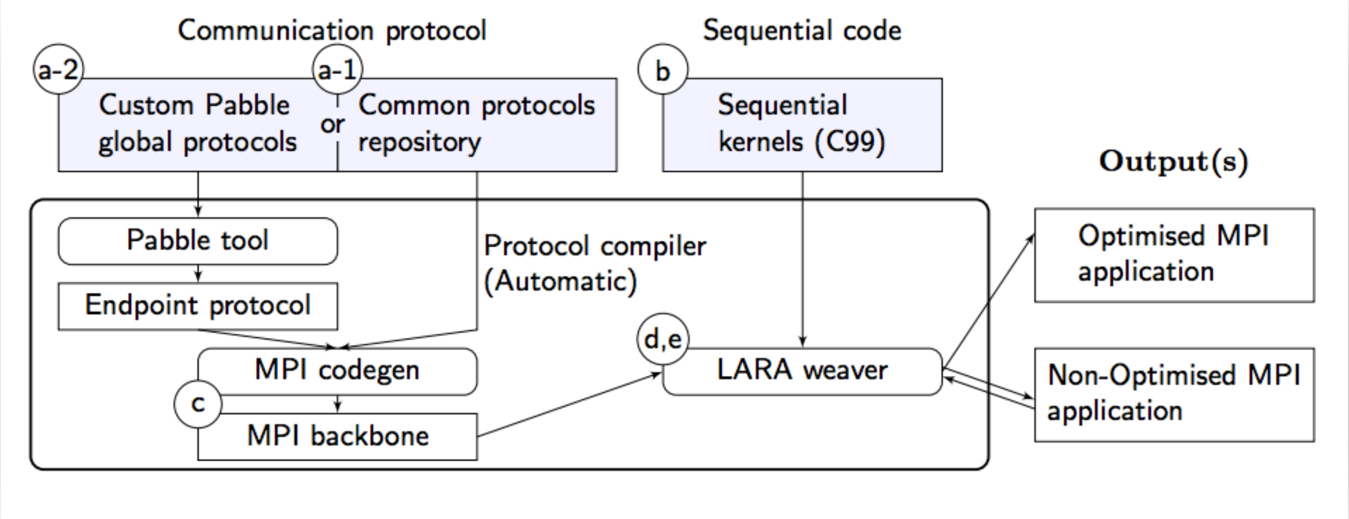
ZDLC: Process Modeling



Actor Verification

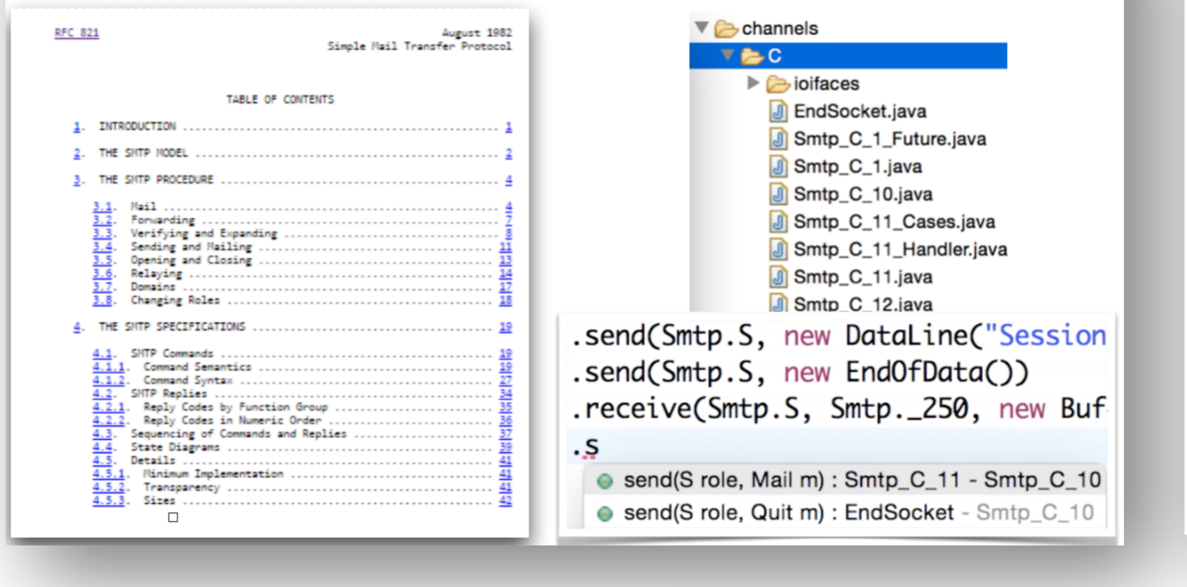

Monitors
Session (protocol mailboxes)
Session (Roles)
Actor Model (Processes and mailboxes)
Session Actor
ANTLR
AMQP
Advanced Message Queuing Protocol
Scribble
python
C

MPI code generations



Session Type based Tools

Java API Generation [FASE'16]



RFC 821 August 1982
Simple Mail Transfer Protocol

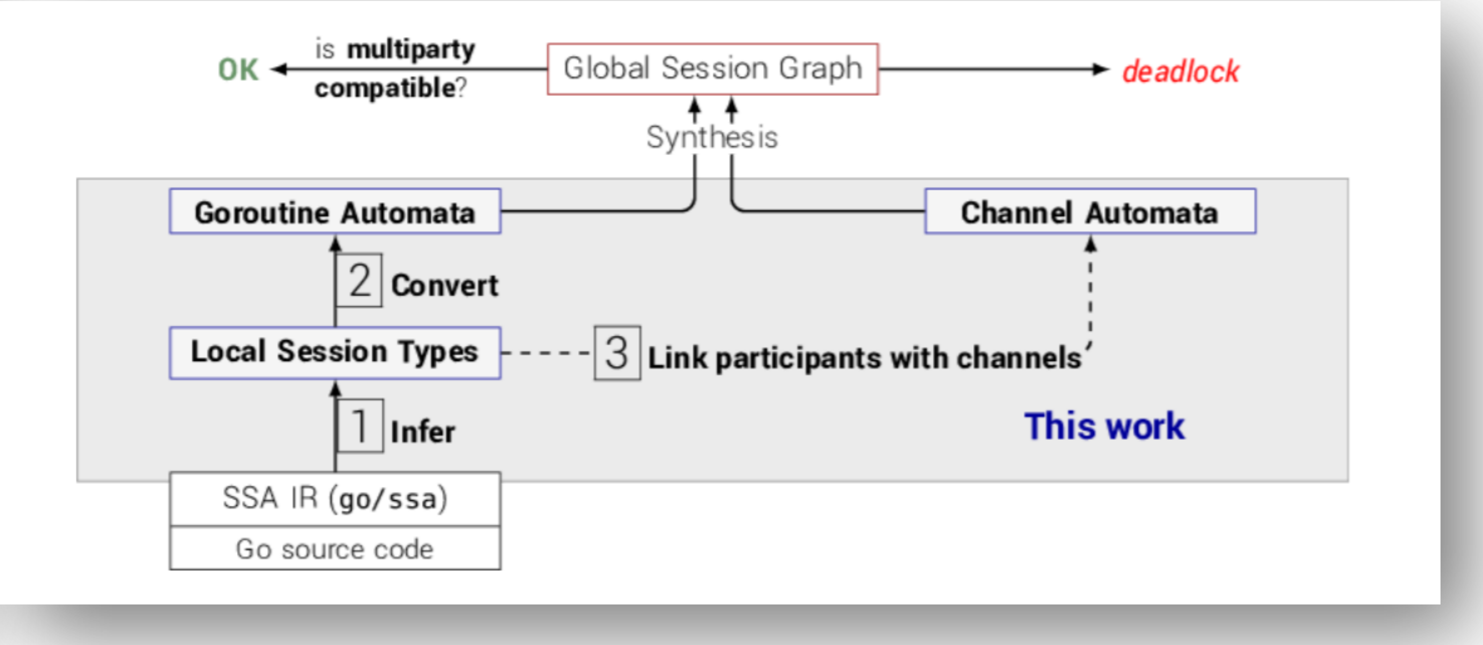
TABLE OF CONTENTS

1. INTRODUCTION 1
2. THE SMTP MODEL 2
3. THE SMTP PROCEDURE 4
   3.1. Mail 4
   3.2. Forwarding 7
   3.3. Verifying and Expanding 8
   3.4. Sending and Mailing 11
   3.5. Opening and Closing 13
   3.6. Relaying 14
   3.7. Domains 17
   3.8. Changing Roles 18
4. THE SMTP SPECIFICATIONS 19
   4.1. SMTP Commands 19
   4.1.1. Command Semantics 19
   4.1.2. Command Syntax 27
   4.2. SMTP Replies 34
   4.2.1. Reply Codes by Function Group 35
   4.2.2. Reply Codes in Numeric Order 36
   4.3. Sequencing of Commands and Replies 37
   4.4. State Diagrams 39
   4.5. Details 41
   4.5.1. Minimum Implementation 41
   4.5.2. Transparency 41
   4.5.3. Sizes 42

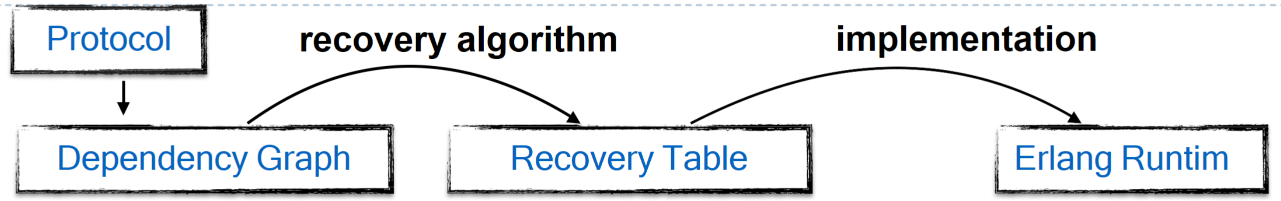
```
.send(Smtp.S, new DataLine("Session
.send(Smtp.S, new EndOfData())
.receive(Smtp.S, Smtp._250, new Buf
.S
```

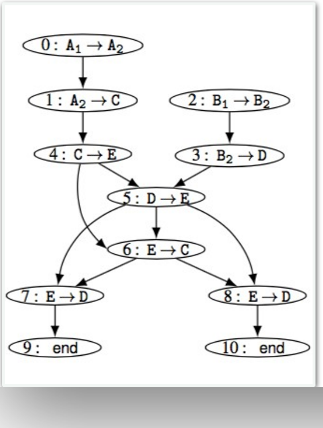
- send(S role, Mail m) : Smtp_C_11 - Smtp_C_10
- send(S role, Quit m) : EndSocket - Smtp_C_10

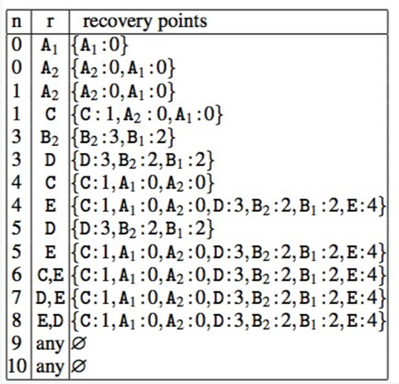
Deadlock Detection for Go [CC'16, POPL'17, ICSE'18]



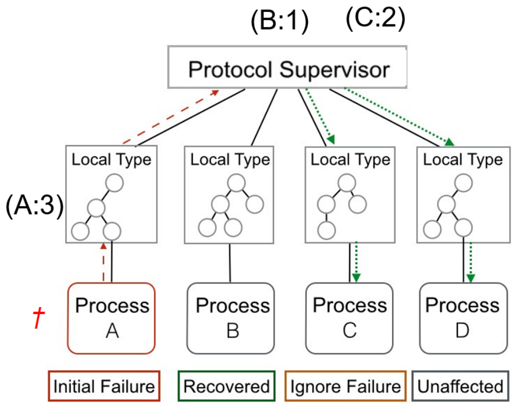
Safe Recovery for Erlang [CC'15]





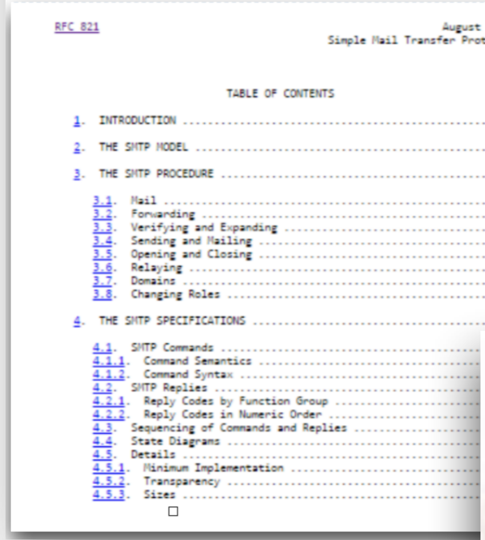


n	r	recovery points
0	A_1	$\{A_1:0\}$
0	A_2	$\{A_2:0, A_1:0\}$
1	A_2	$\{A_2:0, A_1:0\}$
1	C	$\{C:1, A_2:0, A_1:0\}$
3	B_2	$\{B_2:3, B_1:2\}$
3	D	$\{D:3, B_2:2, B_1:2\}$
4	C	$\{C:1, A_1:0, A_2:0\}$
4	E	$\{C:1, A_1:0, A_2:0, D:3, B_2:2, B_1:2, E:4\}$
5	D	$\{D:3, B_2:2, B_1:2\}$
5	E	$\{C:1, A_1:0, A_2:0, D:3, B_2:2, B_1:2, E:4\}$
6	C,E	$\{C:1, A_1:0, A_2:0, D:3, B_2:2, B_1:2, E:4\}$
7	D,E	$\{C:1, A_1:0, A_2:0, D:3, B_2:2, B_1:2, E:4\}$
8	E,D	$\{C:1, A_1:0, A_2:0, D:3, B_2:2, B_1:2, E:4\}$
9	any	∅
10	any	∅



Applications

Java API Generation [FASE'16]

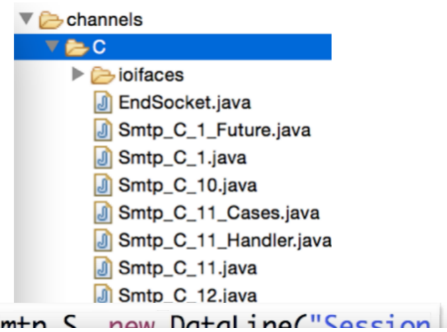


RFC 821

August 1982
Simple Mail Transfer Protocol

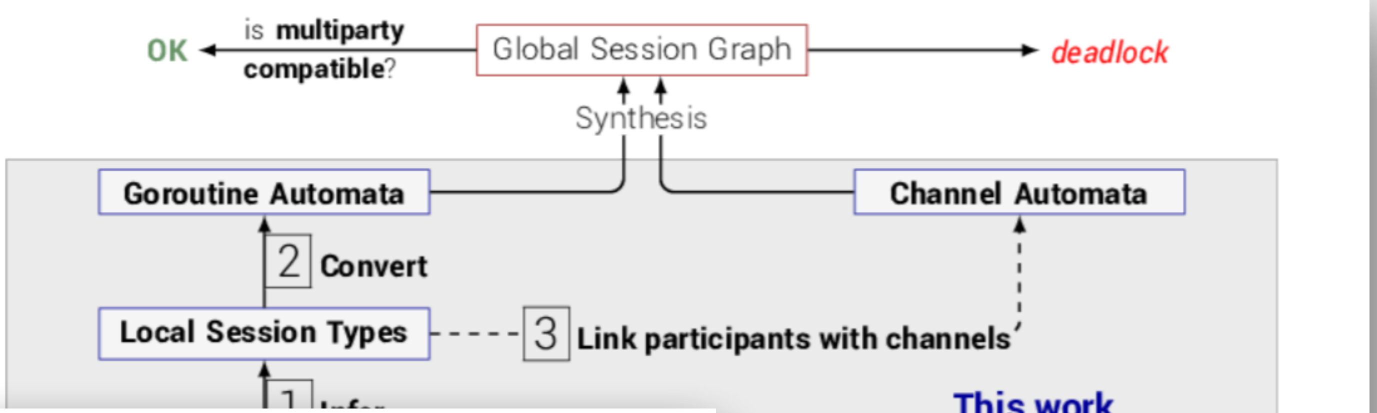
TABLE OF CONTENTS

□

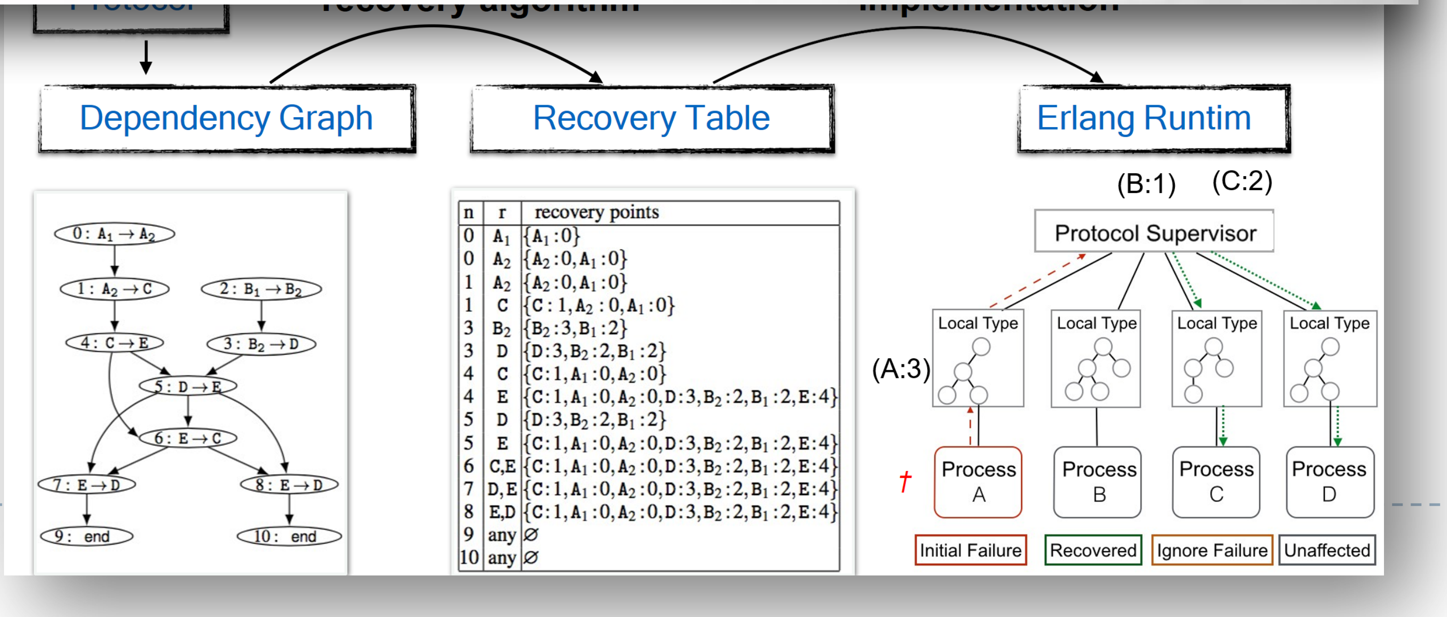


```
send(Smtp_S, new DataLine("Session
```

Deadlock Detection for Go [CC'16, POPL'17]



Scribble



n	r	recovery points
0	A_1	$\{A_1:0\}$
0	A_2	$\{A_2:0, A_1:0\}$
1	A_2	$\{A_2:0, A_1:0\}$
1	C	$\{C:1, A_2:0, A_1:0\}$
3	B_2	$\{B_2:3, B_1:2\}$
3	D	$\{D:3, B_2:2, B_1:2\}$
4	C	$\{C:1, A_1:0, A_2:0\}$
4	E	$\{C:1, A_1:0, A_2:0, D:3, B_2:2, B_1:2, E:4\}$
5	D	$\{D:3, B_2:2, B_1:2\}$
5	E	$\{C:1, A_1:0, A_2:0, D:3, B_2:2, B_1:2, E:4\}$
6	C,E	$\{C:1, A_1:0, A_2:0, D:3, B_2:2, B_1:2, E:4\}$
7	D,E	$\{C:1, A_1:0, A_2:0, D:3, B_2:2, B_1:2, E:4\}$
8	E,D	$\{C:1, A_1:0, A_2:0, D:3, B_2:2, B_1:2, E:4\}$
9	any	$\varnothing$
10	any	$\varnothing$

Session Types

Scribble

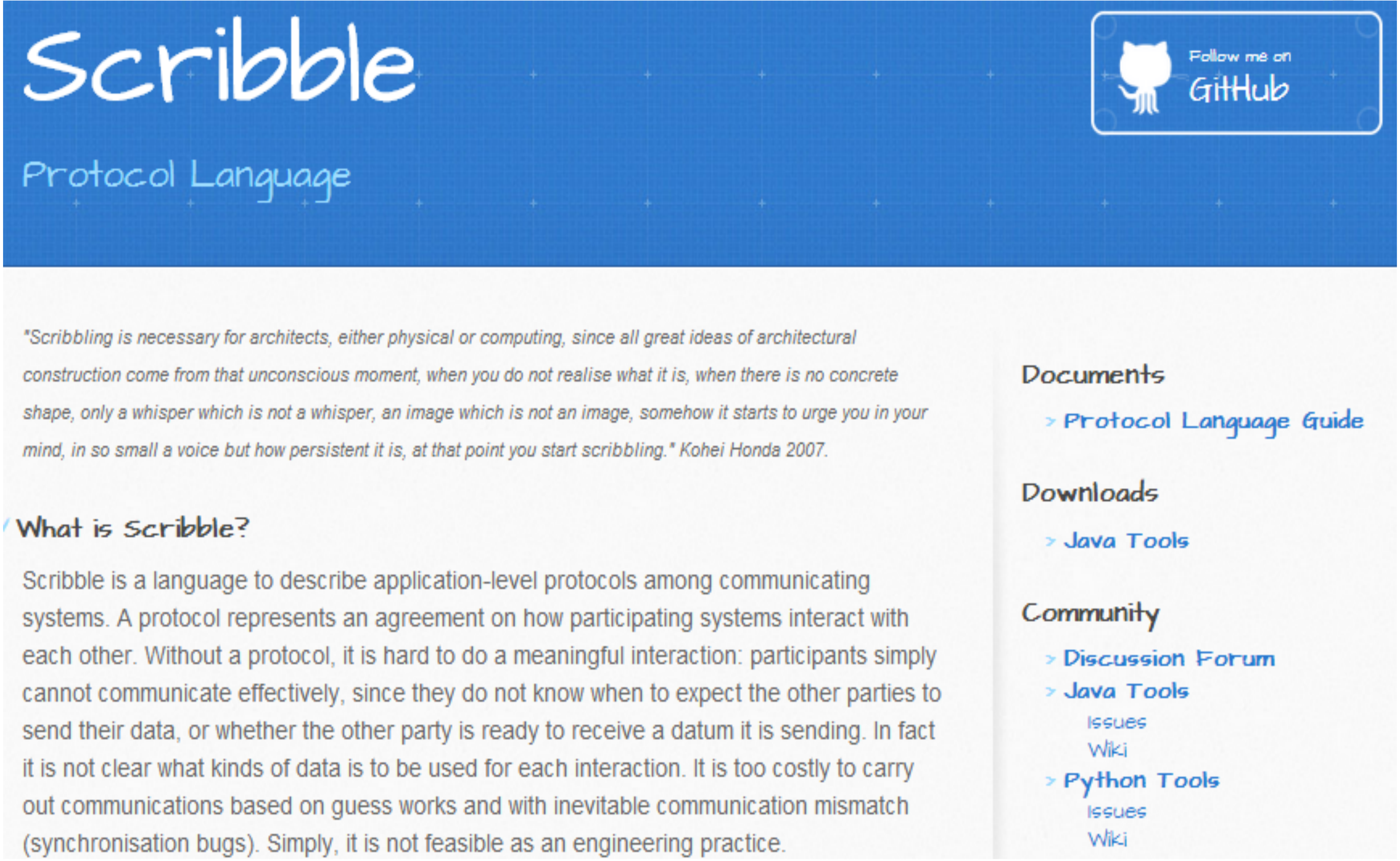
Scribble Protocol

- *"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;
}
```

www.scribble.org



Scribble

Protocol Language

Follow me on GitHub

"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.

What is Scribble?

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 a meaningful interaction: participants simply cannot communicate effectively, since they do not know when to expect the other parties to send their data, or whether the other party is ready to receive a datum it is sending. In fact it is not clear what kinds of data is to be used for each interaction. It is too costly to carry out communications based on guess works and with inevitable communication mismatch (synchronisation bugs). Simply, it is not feasible as an engineering practice.

Documents

- Protocol Language Guide

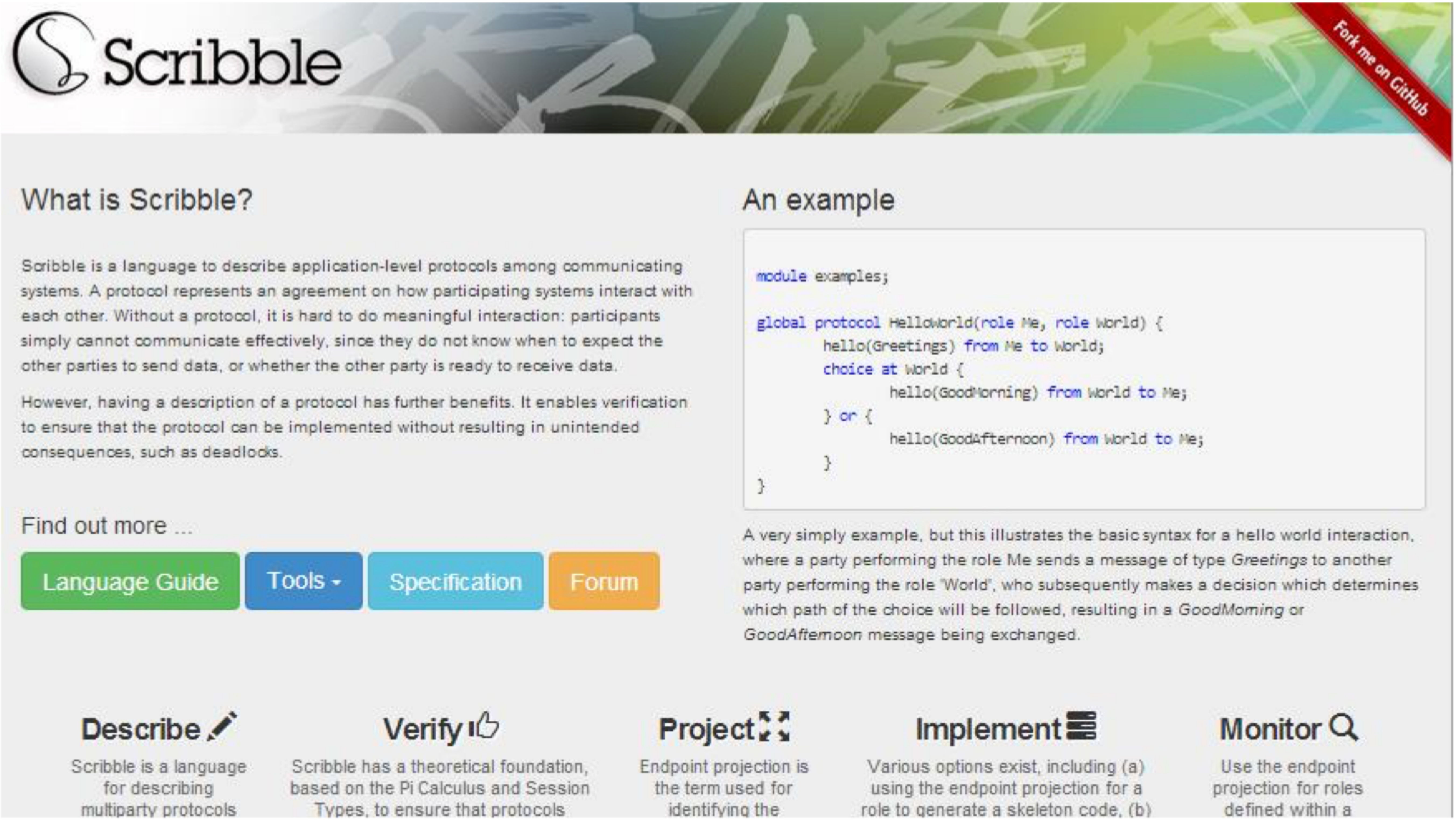
Downloads

- Java Tools

Community

- Discussion Forum
- Java Tools
  - Issues
  - Wiki
- Python Tools
  - Issues
  - Wiki

Meet Scribble www.scribble.org



What is Scribble?

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.

Find out more ...

An example

```
module examples;

global protocol HelloWorld(role Me, role World) {
        hello(Greetings) from Me to World;
        choice at World {
                hello(GoodMorning) from World to Me;
        } or {
                hello(GoodAfternoon) from World to Me;
        }
}
```

A very simply example, but this illustrates the basic syntax for a hello world interaction, where a party performing the role Me sends a message of type *Greetings* to another party performing the role 'World', who subsequently makes a decision which determines which path of the choice will be followed, resulting in a *GoodMorning* or *GoodAfternoon* message being exchanged.

Describe

Scribble is a language for describing multiparty protocols

Verify

Scribble has a theoretical foundation, based on the Pi Calculus and Session Types, to ensure that protocols

Project

Endpoint projection is the term used for identifying the

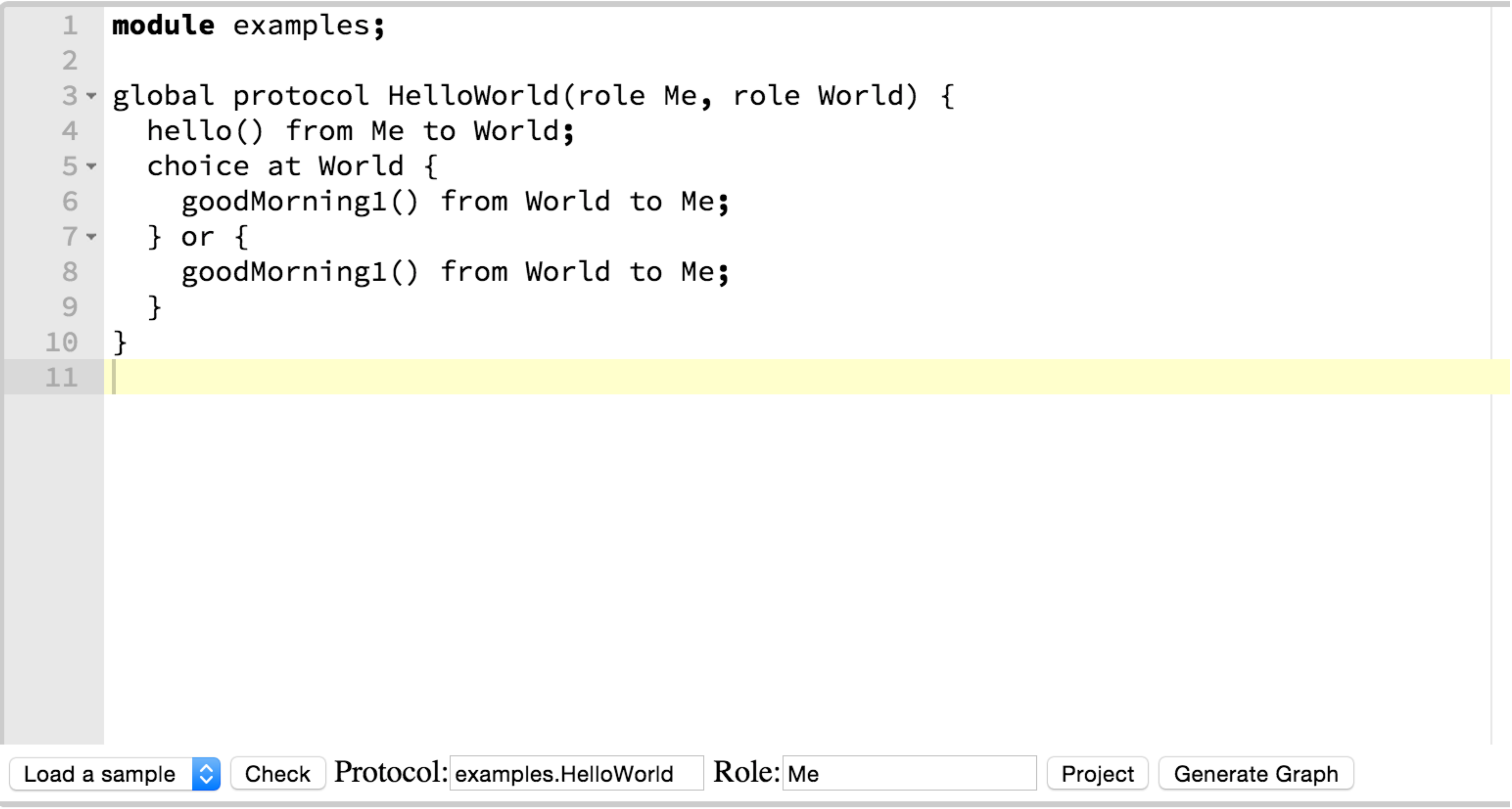
Implement

Various options exist, including (a) using the endpoint projection for a role to generate a skeleton code, (b)

Monitor

Use the endpoint projection for roles defined within a

Let’s try some protocols: 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;
  }
}

```

Protocol: examples.HelloWorld

Role: Me

Example

protocol def

recursion

send-receive

choice

```
global protocol Q&A(role me, role you){
rec loop {
      ask(string) from you to me;
      choice at me
           { response (string) from me to you;
              continue loop; }
       or { enough() from me to you; }}
```

Scribble

Protocol Validation

Are we compatible?

send(int).send(int).receive(bool)

receive(int).receive(int).send(bool)

It is all about duality!

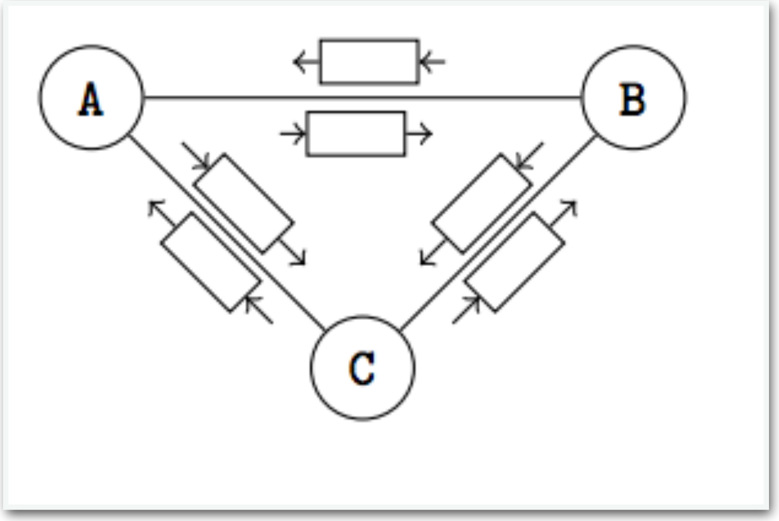
Are we compatible?

receive(int).send(int).receive(bool)

receive(int).receive(int).send(bool)

Good/Bad MPST by example

- Communication model:
  - asynchronous, reliable, role-to-role ordering
  - MPST applies to transports that fit this model
    - TCP, HTTP, …, AMQP, …shared memory
- MPST protocols should be fully specified
  - no implicit messages needed to conduct a session



Next….

- Core Scribble constructs
- What can go wrong ?
- MPST safety and liveness errors (informally)
- How are they ruled out (syntactically)

Properties (by example)

☑ Communication mismatch

```
send(A, Div, int) | recv(A, Add, int)      ✖ Wrong label
send(A, Div, int) | recv(A, Add, string)   ✖ Wrong payload
send(B, Div, int) | recv(A, Div, int)      ✖ Wrong role
```

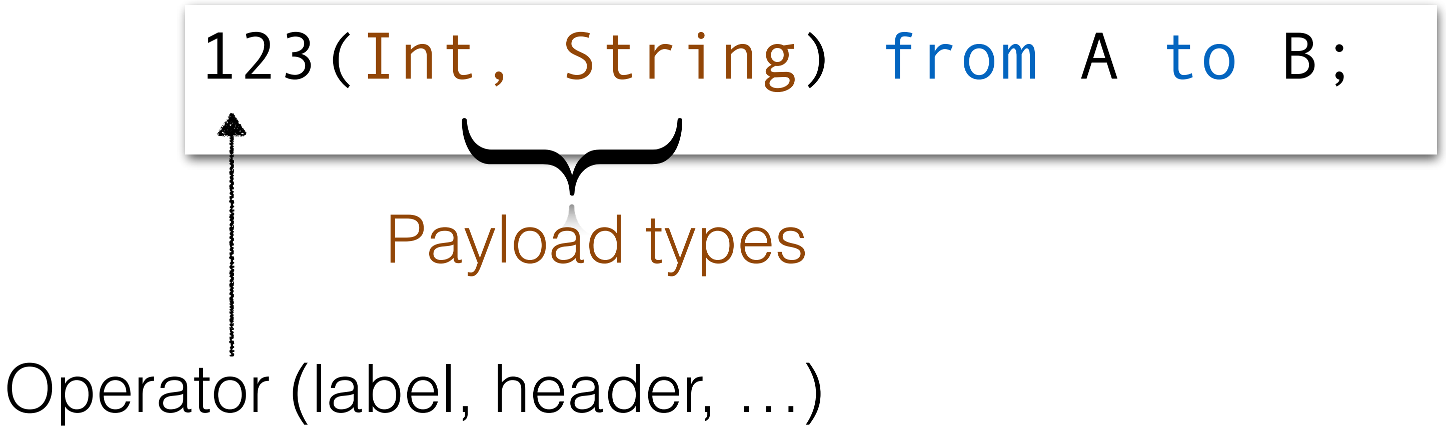
☑ Orphan messages

```
send(A)|send(A)
```

☑ Deadlock

```
recv(A)|recv(B)
recv(C)|recv(C)|if (n=0) then send(A) else send(B)
```

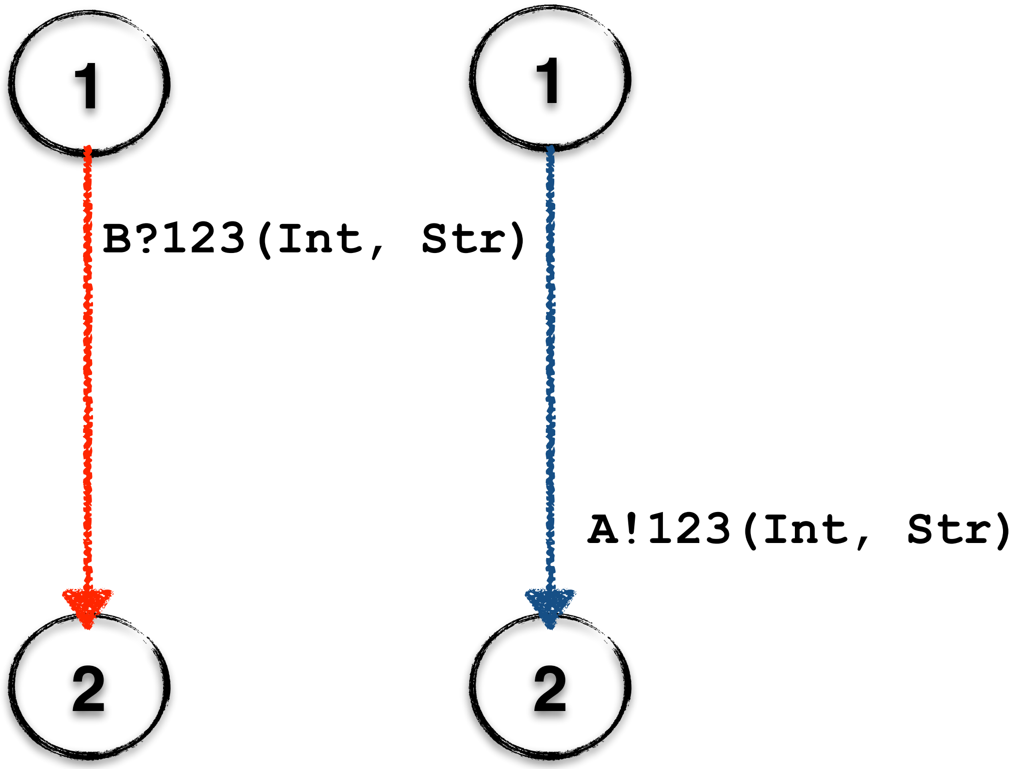
Scribble constructs:
Role-to-role Message passing



```
123(Int, String) from A to B;
```

Payload types

Operator (label, header, …)



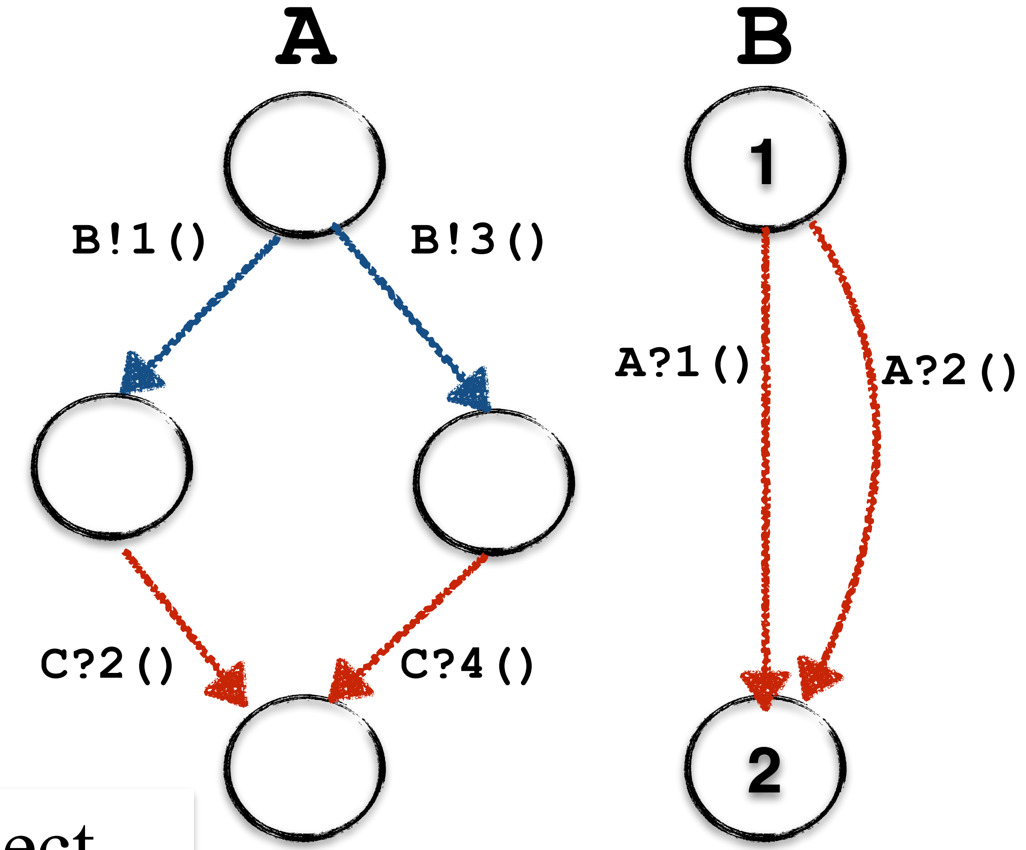
- Empty operator and/or payload is allowed

```
() from A to B;
```

✅

Scribble constructs:
"Located" choice

```
choice at A {
  1() from A to B;
  2() from A to C;
  } or {
  3() from A to B;
  4() from A to C;
  }
}
```



- Internal choice by global choice subject
- External choice for all other roles

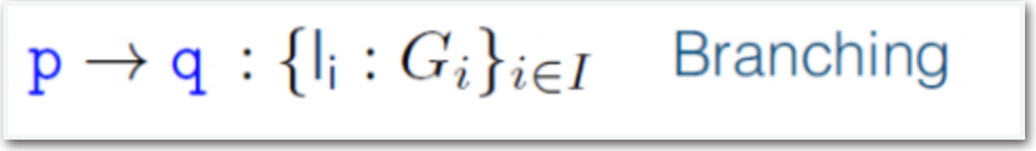
Condition

- Only enabled roles can send messages in choice paths
  - Start role enabled, other disabled
  - a role is enabled by receiving a message from an enabled role

Scribble constructs:
“Located” choice

```
 choice at A {
   buyer1(int) from A to B; // Total to pay
   (int) from B to A;// B will pay that much
   buyer1(int) from A to C; // C pays the remainder
 } or {
   buyer1(x:int, y:int) from A to C; // Total to pay
   (Int) from C to A; // C pays that much
   buyer2(x:int, y:int) from A to  B;// B pays the remainder
 }
}
```

- More flexible than directed choice



$\mathsf{p} \to \mathsf{q} : \{\mathsf{l}_\mathsf{i} : G_i\}_{i \in I}$ Branching

- Branching via different payloads not allowed

```
choice at A {1() from A to B;} or {1(int) from A to B;}
```

✗

Exercise:
“Located” choice

Condition

- Only enabled roles can send messages in choice paths
  - Start role enabled, other disabled
  - a role is enabled by receiving a message from an enabled role

```
 choice at A {
   1() from A to B;
   1() from B to C;
   1() from C to A;
 } or {
   2() from B to A;   ❌ Role B not enabled
   choice at B {
     2() from B to C;
   } or {
     3() from B to C;
   }
   4() from C to A;
}
```

What actually goes wrong ?

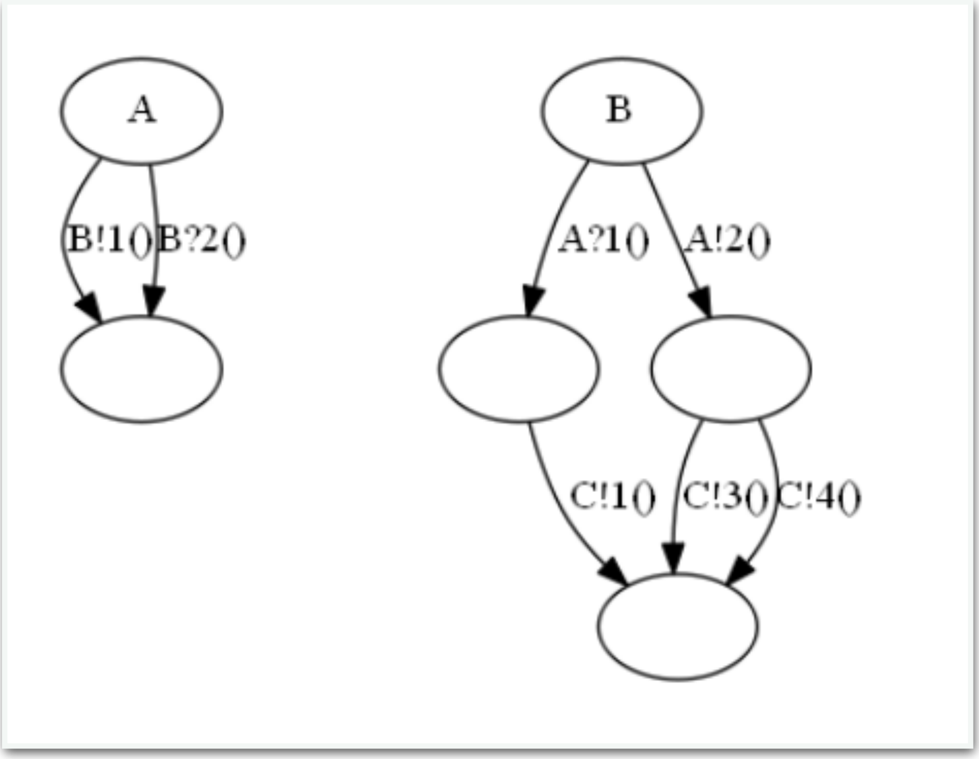
- MPST Safety errors:
  - reception error, orphan message, deadlock

Exercise:
“Located” choice

What actually goes wrong ?

- MPST Safety errors:
  - reception error, orphan message, deadlock

```
choice at A {
  1() from A to B;
  1() from B to C;
  1() from C to A;
} or {
  2() from B to A;  ✖ Role B not enabled
  choice at B {
    2() from B to C;
  } or {
    3() from B to C;
  }
  4() from C to A;
}
```



Is this protocol OK? 1/4

```
choice at A {
   1() from A to B;
   3() from B to C; ✖
   4() from C to A;
} or {
   2() from A to B;
   3() from A to C; ✖
   5() from A to C;
}
```

Errors explained ?

- Ambitious choice for C
  - Should C send a 4 or 5 to A?
  - potential reception errors (4, 5) if interpreted non-deterministically
- *Non-deterministic choice at C* inconsistent with the choice by A
  - Not mergeable in syntactic projections
  - has to merge continuations (undefined for distinct outputs)

Is this protocol OK? 1/4

```
choice at A {
  1() from A to B;
  3() from B to C;
  4() from C to A;
} or {
  2() from A to B;
  3() from A to C;
  5() from A to C;
}
```

How to fix t?

Is this protocol OK? 1/4

```
choice at A {
  1() from A to B;
  3a() from B to C;
  4() from C to A;
} or {
  2() from A to B;
  3b() from A to C;
  5() from A to C;
}
```

Distinguish label 3!

Is this protocol OK? 2/4 ✅

```
choice at A {
  1() from A to B;
  3() from B to C;
  do Merge(A, C);
} or {
  2() from A to B;
  3() from B to C;
  do Merge(A, C);
}
```

```
global protocol Merge(role A, role C){
  choice at A {
   5() from A to C;
 } or {
   5() from A to C;
}}
```

- Duplicate cases inherently mergeable, e.g [POPL’11]

Is this protocol OK? 3/4

```
choice at A {
  1a() from A to B;
  2() from A to C;
  3() from B to C;  ❌
  4() from C to A;
} or {
  1b() from A to B;
  3() from B to C;  ❌
  4() from C to A;
}
```

Errors explained ?

- "Race condition" on choice on C due to asynchrony
  - What should C do after receiving a 3?
  - Potential orphan message (2) if interpreted as multi-queue FIFO
- Inconsistent external choice subject
  - (trivially non-mergeable in standard MPST)
  - A role must be enabled by the same role in choice paths

Is this protocol OK? 4/4

```
choice at A {
 1() from A to B;
 2() from A to C;  ✖
 } or {
 3() from B to B;
}
```

Errors explained ?

- Unrealisable choice at C
  - No implicit message can be assumed, e.g end of session
  - How can C determine if a message is coming?
  - Potential deadlock (C waiting for A), or potential orphan (2), depending on the interpretation
- Empty action option to terminal state
  - can’t merge end type with anything else

Quiz: Mergeability

```
choice at A {
   1() from A to B;
   2() from C to B;
} or {
   3() from A to D;
   4() from D to B;
}
```

❌

```
choice at A {
   1() from A to B;
   2() from C to D;
} or {
   3() from A to B;
   4() from C to D;
}
```

❌

```
choice at A {
   1() from A to C;
   2() from C to D;
} or {
   3() from A to B;
   2() from C to D;
}
```

✅

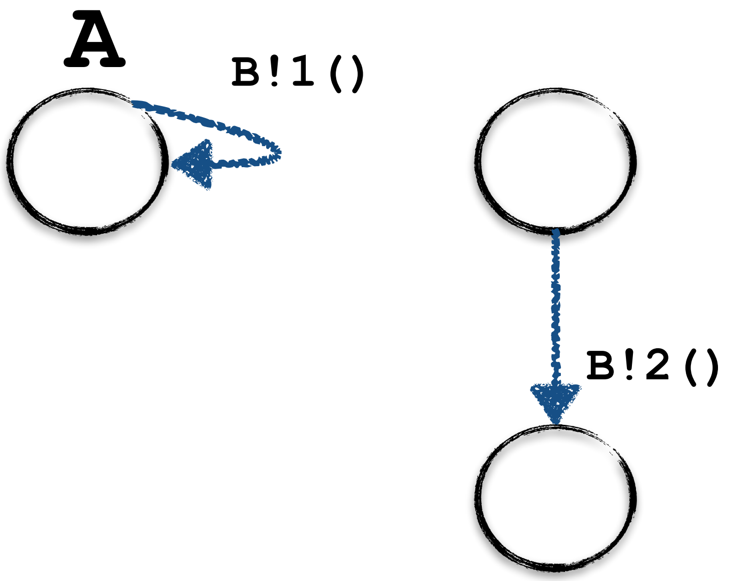
```
choice at A {
   1() from A to C;
   2() from B to C;
} or {
   3() from A to B;
   4() from B to C;
}
```

✅

Scribble construct: **Recursion**

- Tail recursion with recursive scopes

```
rec X {
  1() from A to B;
  continue X;
}
2() from A to B; ✖ Dead code
```



Condition

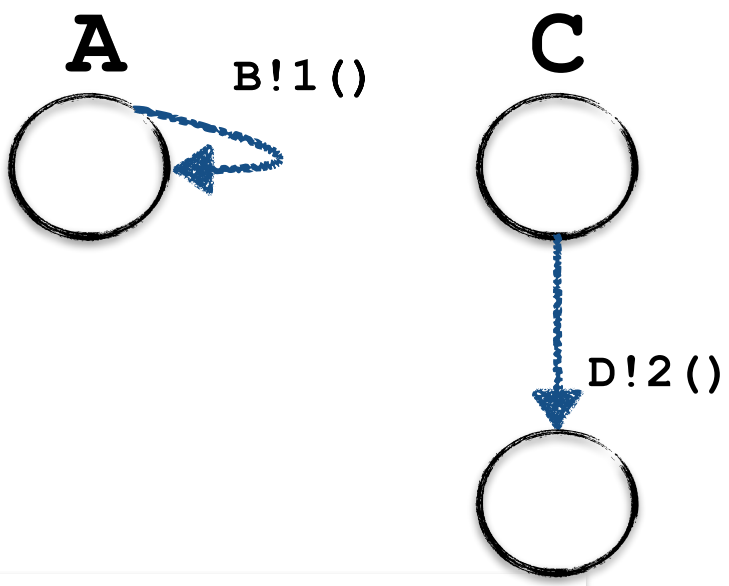
- Reachability of protocol states (no “dead code”)
  - Checked via projection (reachability w.r.t per-role protocol flow)
- Regular interaction structure at endpoints (CFSM)

Scribble construct: **Recursion**

- Tail recursion with recursive scopes

```
rec X {
   1() from A to B;
   continue X;
}
2() from A to B; ✖ Dead code
```

```
rec X {
   1() from A to B;
   continue X;
}
2() from C to D; ✅
```



Condition

- Reachability of protocol states (no "dead code")
  - Checked via projection (reachability w.r.t per-role protocol flow)
- Regular interaction structure at endpoints (CFSM)

Is this protocol ok? 1/4

```
rec X {
  choice at A
    1() from A to B;
    continue X;
    2() from A to B; ✖ Dead code
  } or {
    3() from A to B;
}
4() from A to B; ✖
}
5() from A to B;
```

Condition

- Reachability of protocol states (no "dead code")
  - Checked via projection (reachability w.r.t per-role protocol flow)
- Regular interaction structure at endpoints (CFSM)

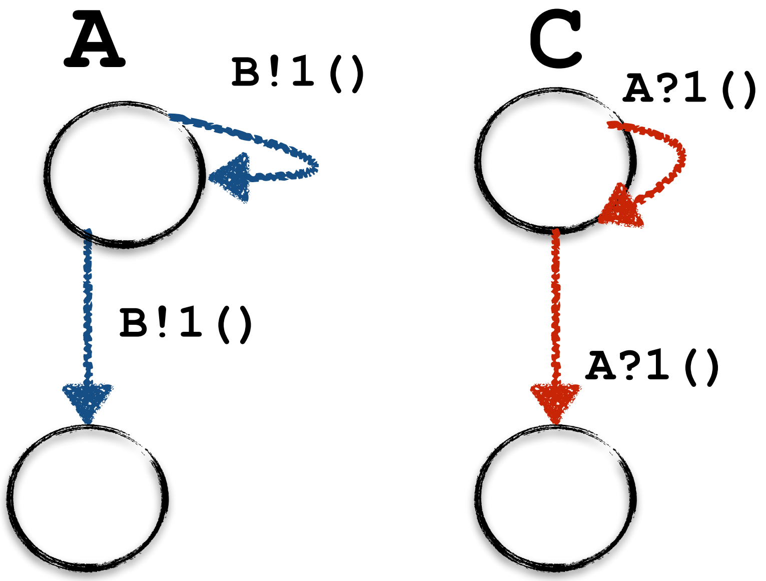
Is this protocol OK? 2/4

```
rec X {
  choice at A {
    1() from A to B;
    2() from B to C;
    3() from C to B;
} or {
    4() from A to C;
    5() from C to B;
}
continue X;
```

Why does Scribble not allow this protocol?

Is this protocol OK? 3/4

```
rec X {
  choice at A {
    1() from A to B;
    continue X;
} or {
    1() from A to B;
}
```



Potential ***deadlocks*** or ***orphans***

Is this protocol ok? 4/4

```
rec X {
  choice at A {
    1() from A to B;
    1() from B to C;
    continue X;
  } or {
    2() from A to B;
    2() from B to C; ❌
}
```

- Safety errors?
  - hint: Consider the FSM at A?

Is this protocol ok? 4/4

```
rec X {
  choice at A {
    1() from A to B;
    1() from B to C;
    continue X;
  } or {
    2() from A to B;
    2() from B to C; ❌
}
```

- Safety errors?
  - hint: Consider the FSM at A?
  - How about now?

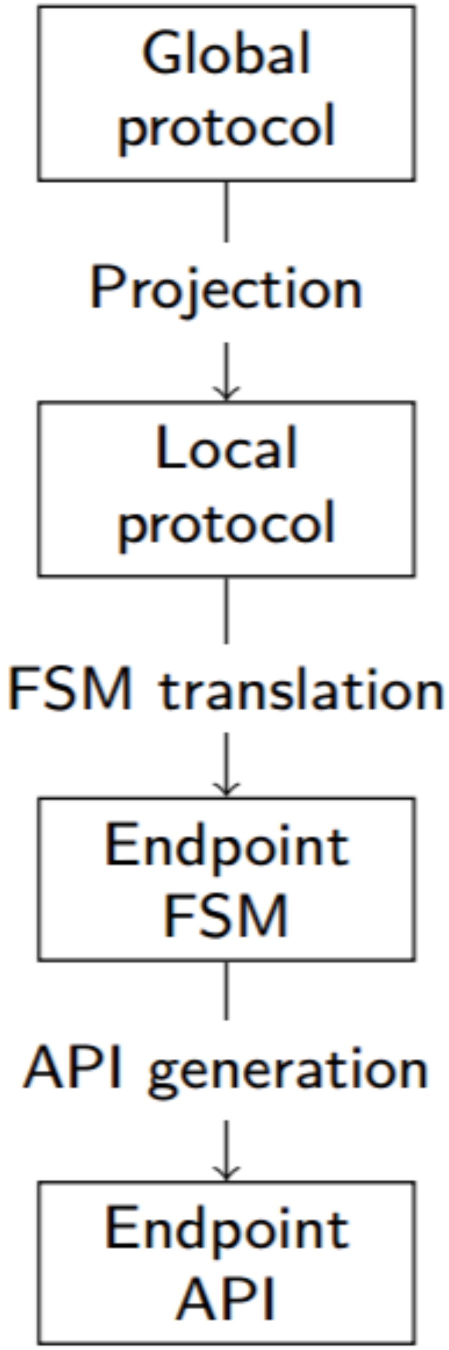
- Liveness errors?
  - Role progress
  - Message liveness

Scribble

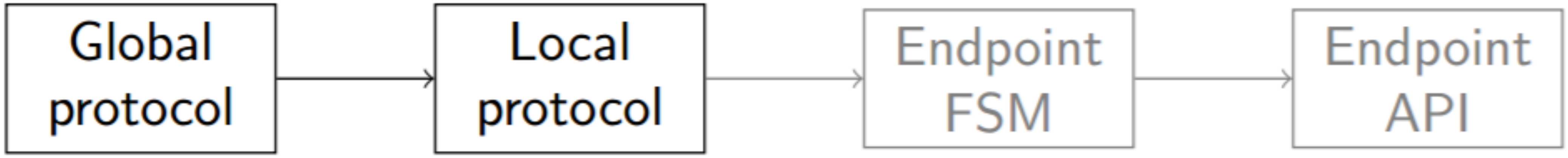
Program Verification

Scribble Endpoint API generation toolchain

- Protocol spec. as Scribble protocol (asynchronous MPST)
  - Global protocol validation
    (safely distributable asynchronous protocol)
  - Syntactic projection to local protocols
    (static session typing if supported)
  - Endpoint FSM (EFSM) translation
    (dynamic session typing by monitors)
    - Protocol states as state-specific channel *types*
    - Call chaining API to link successor states
- Java APIs for implementing the endpoints



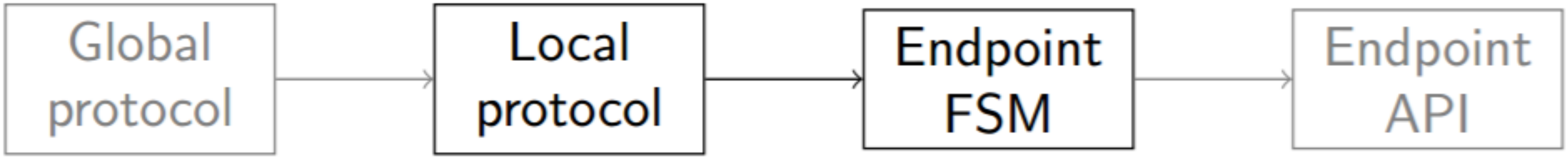
Example: Adder



(validation, projection)

```
global protocol Adder(role C, role S) {
  choice at C {
    Add(Integer, Integer) from C to S;
    Res(Integer) from S to C;
    do Adder(C, S);
  } or {
    Bye() from C to S;
    Bye() from S to C;
  }
}
```

Example: Adder



Global protocol

Local protocol

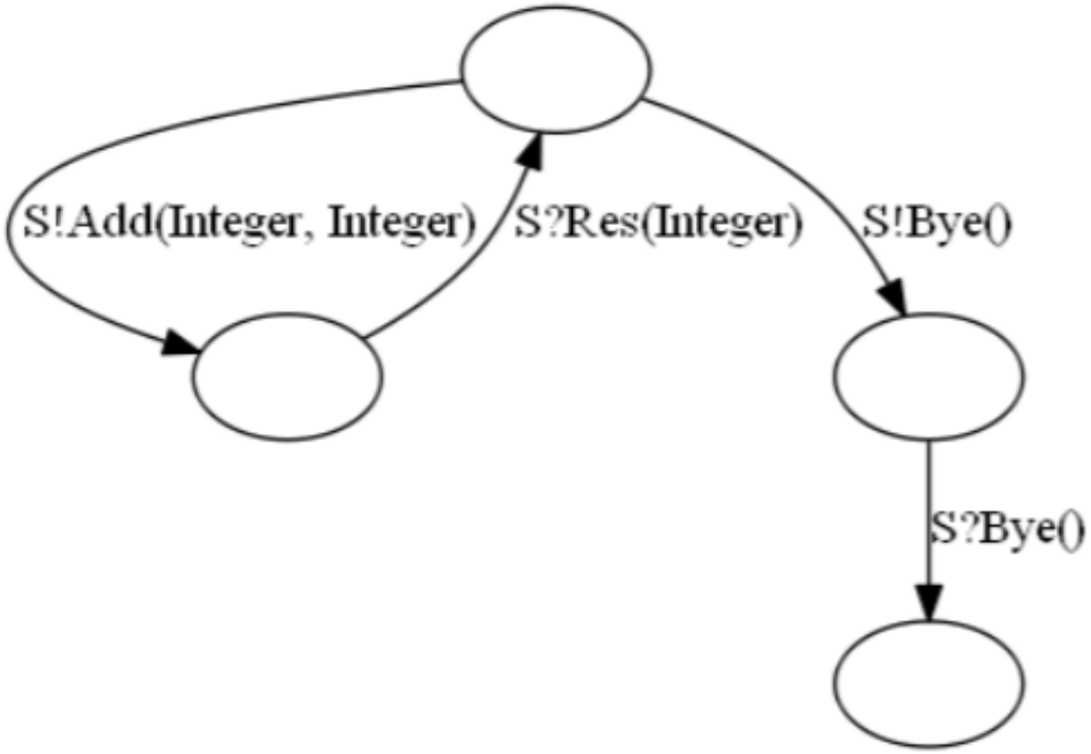
Endpoint FSM

Endpoint API

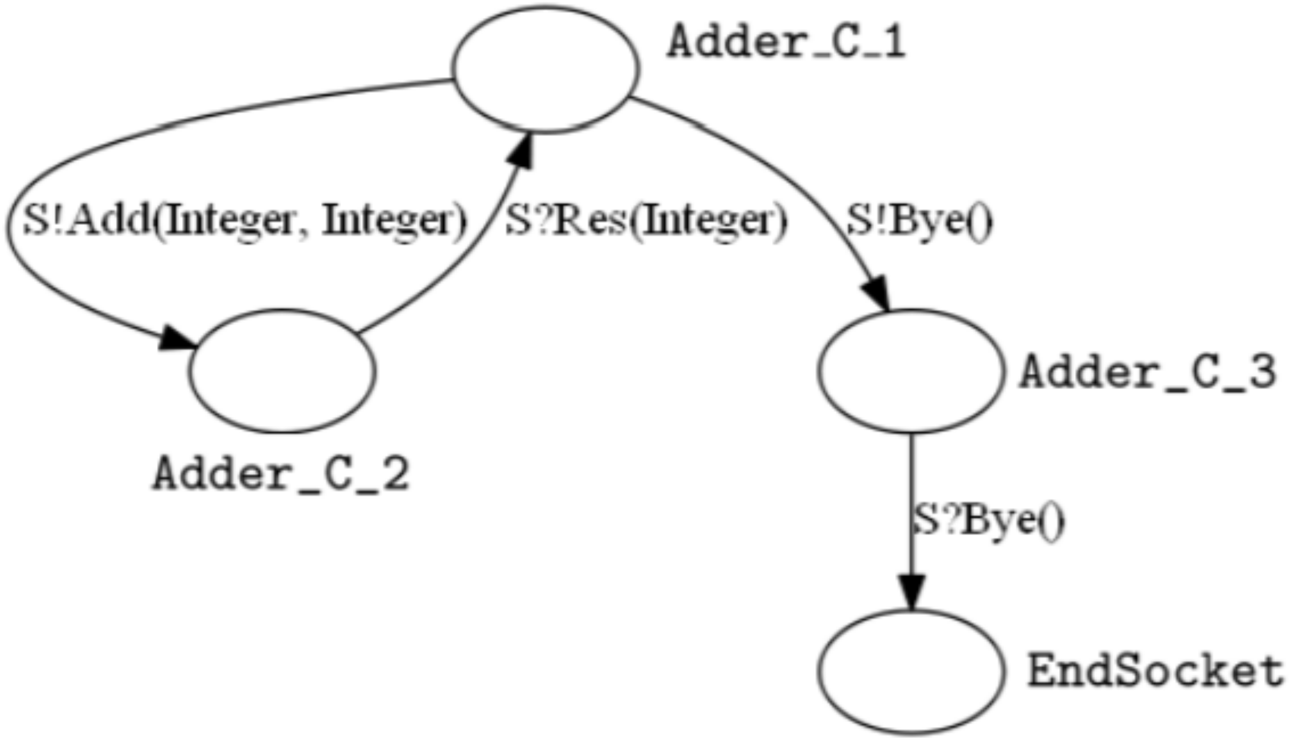
(FSM translation)

```
global protocol Adder(role C, role S) {
  choice at C {
    Add(Integer, Integer) from C to S;
    Res(Integer) from S to C;
    do Adder(C, S);
  } or {
    Bye() from C to S;
    Bye() from S to C;
  }
}
```

- Projected Endpoint FSM (EFSM) for C



Adder: State Channel API for C

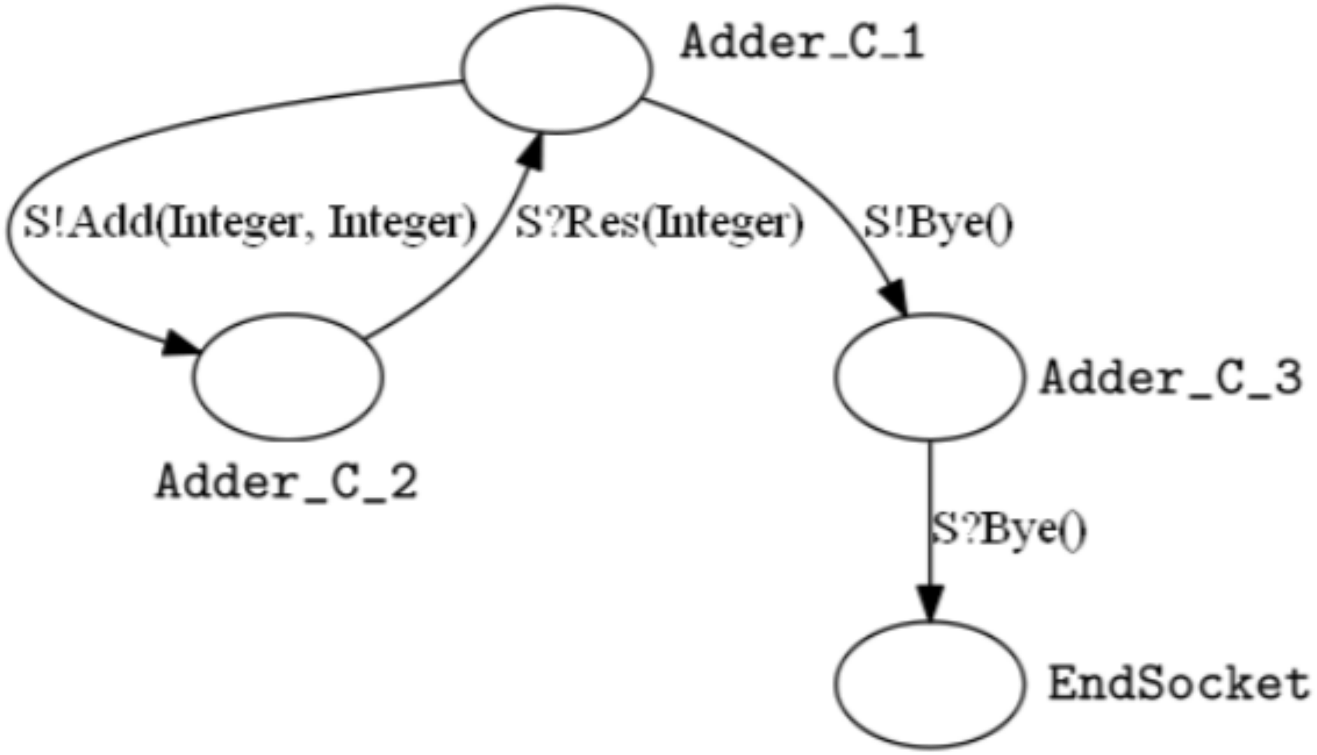


- Adder_C_1
  - Output state channel: (overloaded) send methods

```
Adder_C_2 send(S role, Add op, Integer arg0, Integer arg1) throws ...
Adder_C_3 send(S role, Bye op) throws ...
```

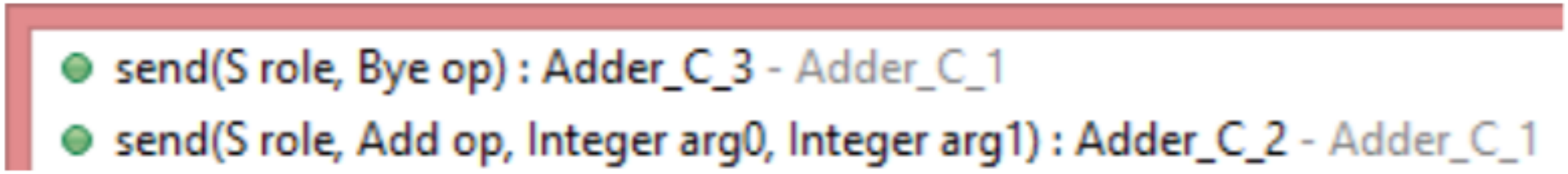
    - Parameter types: message recipient, operator and payload
    - Return type: successor state

Adder: endpoint implementation for C



```
Adder_C_1 c1 = new Adder_C_1(...);

c1.
```



- send(S role, Bye op) : Adder_C_3 - Adder_C_1
- send(S role, Add op, Integer arg0, Integer arg1) : Adder_C_2 - Adder_C_1

A demo is worth a thousand slides

MPST beyond verification

Let it Recover:

Multiparty Protocol-Induced Recovery

Rumyana Neykova, Nobuko Yoshida
Imperial College London