

PRINCIPLES AND PRACTICE OF SESSION TYPES

Rumyana Neykova & Nobuko Yoshida

Imperial College
London

Communications are Ubiquitous

- The way to organise software and systems is increasingly based on **communications**.

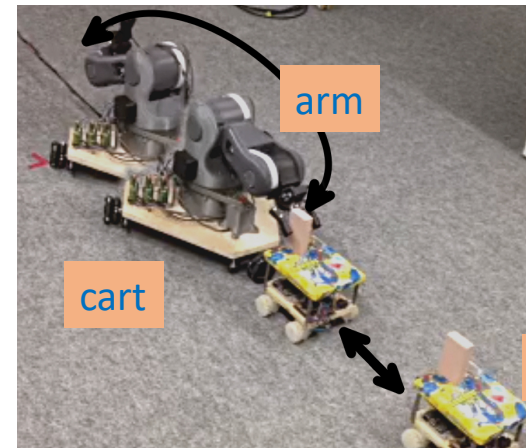
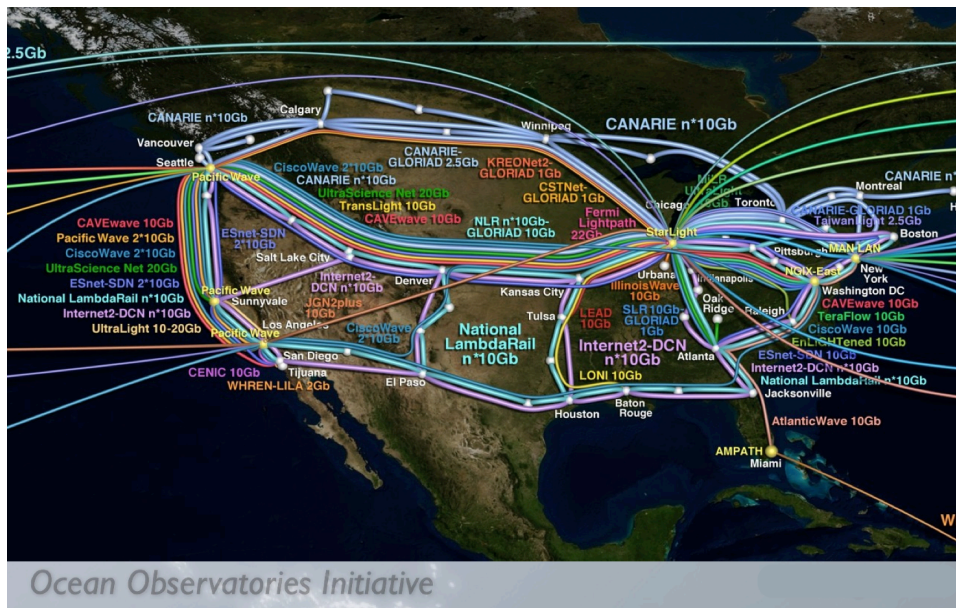


Computing is Communications

- The way to organise software and systems is increasingly based on **communications**.



micro services



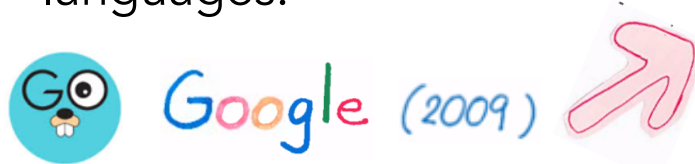
Communications are Ubiquitous

- The way to organise software and systems is increasingly based on **communications**.
- Industry trend – **explicit communication features** in the programming languages.

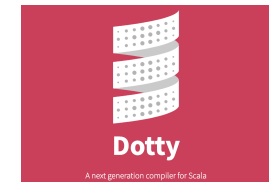


Computing is Communications

- The way to organise software and systems is increasingly based on **communications**.
- Industry trend – **explicit communication features** in the main stream programming languages.



*Do not communicate by sharing memory;
share memory by communicating*
– Go Philosophy



Problems: Concurrency Bugs

- Communications increase **concurrency bugs**
 - Survey of 4k users
 - Analysis of 6 large software (2019)

more than a half of concurrency bugs in Go are caused by communications

deadlock

channel errors





Problems: Concurrency Bugs and Ambiguity

- Communications increase **concurrency bugs**
 - Survey of 4k Go users – the main challenge is to avoid concurrency bugs
 - Analysis of 6 large software – more than a half of concurrency bugs in Go are communication-oriented
- Protocol descriptions are **ambiguous**
 - They are written in English, often very long



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



Problems: Concurrency Bugs and Ambiguity

- Communications increase **concurrency bugs**
 - Survey of 4k Go users – the main challenge is to avoid concurrency bugs
 - Analysis of 6 large software – more than a half of concurrency bugs in Go are communication-oriented
- Protocols are **ambiguous**
 - Written in English, often large

3.1. MAIL

There are three steps to SMTP mail transactions. The transaction is started with a MAIL command which gives the sender identification. A series of one or more RCPT commands follows giving the receiver information. Then a DATA command gives the mail data. And finally, the end of mail data indicator confirms the transaction.

The first step in the procedure is the MAIL command. The <reverse-path> contains the source mailbox.

```
MAIL <SP> FROM:<reverse-path> <CRLF>
```

This command tells the SMTP-receiver that a new mail transaction is starting and to reset all its state tables and buffers, including any recipients or mail data. It gives the reverse-path which can be used to report errors. If accepted, the receiver-SMTP returns a 250 OK reply.

The <reverse-path> can contain more than just a mailbox. The <reverse-path> is a reverse source routing list of hosts and source mailbox. The first host in the <reverse-path> should be the host sending this command.

The second step in the procedure is the RCPT command.

```
RCPT <SP> TO:<forward-path> <CRLF>
```

This command gives a forward-path identifying one recipient. If accepted, the receiver-SMTP returns a 250 OK reply, and stores the forward-path. If the recipient is unknown the receiver-SMTP returns a 550 Failure reply. This second step of the procedure can be repeated any number of times.

Problems: Concurrency Bugs

- Communications increase **concurrency bugs**
 - Survey of 4k users
 - Analysis of 6 large software (2019)

more than a half of concurrency bugs in Go are caused by communications



Session Types

- Can abstract, implement and manage communications as **Protocols**
- Based on programming language techniques, **Clean**, **Cheap** and **Retrofittable**

Why Session Types?

- Robin Milner: *Types are the leaven of computer programming; they make it **digestible***
 - Can describe **communication protocols** as **types**
- **Scalable automatic verifications** for **deadlock-freedom, safety** and **liveness** properties without state-space explosion problems
- **Programming language techniques** with a **correctness-by-construction**
- Can be **retrofitted to exiting systems and tools** such as microservices, and IDEs (integrated development environment).

Why Session Types, and Why Now?

- Robin Milner: *Types are the leaven of computer programming; they make it **digestible***
 - ➔ Can describe **communication protocols** as **types**
- **Scalable automatic verifications** for **deadlock-freedom**, **safety** and **liveness** properties without state-space explosion problems
- **Programming language techniques** with a **correctness-by-construction**
- Can be **retrofitted to exiting systems and tools** such as microservices, and IDEs (integrated development environment).
- **Extendable** by transferring accumulated techniques in type theories



since 1930



since 1990

Why Session Types?

Significant academic and industry interests via fundamental breakthroughs

Binary Session Types

ESOP'98



Why Session Types?

Significant academic and industry interests via fundamental breakthroughs

Binary Session Types ESOP'98



Joined W3C 2002

Why Session Types?

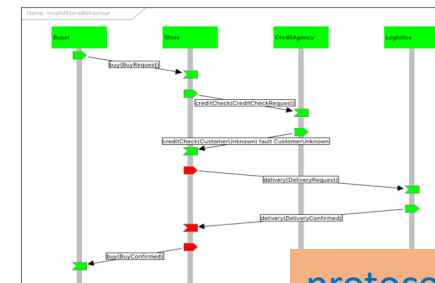
Significant academic and industry interests via fundamental breakthroughs

Binary Session Types ESOP'98



Joined W3C 2002

Scribble at π 4 technology



protocol

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

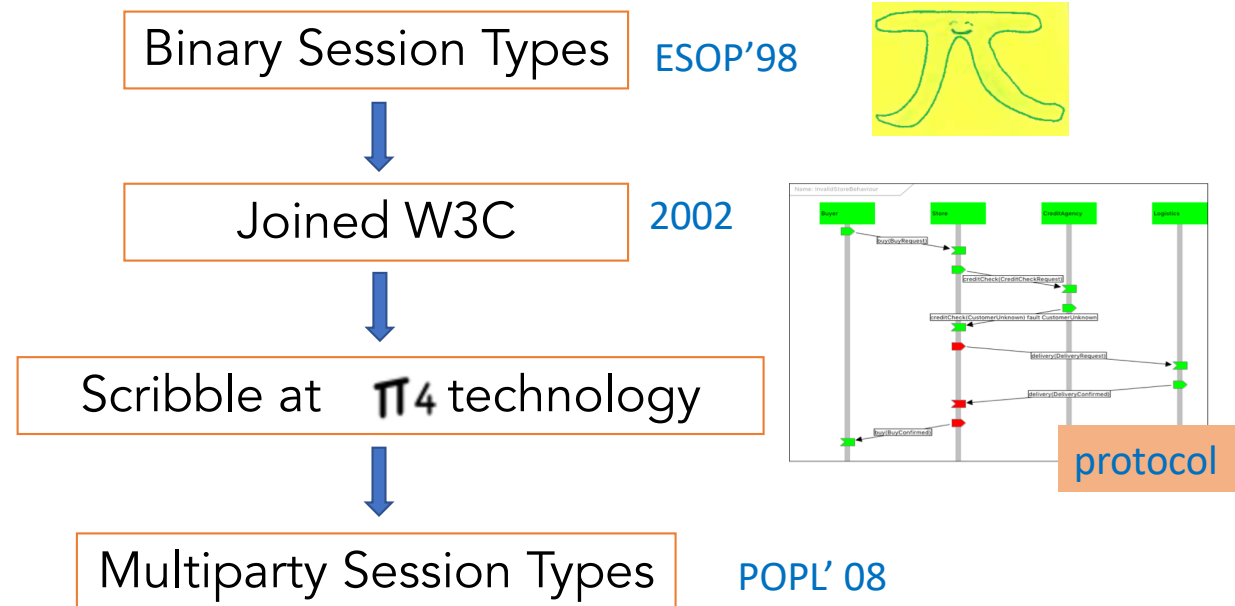
- *"Scribbling is necessary for architects, either physical or computing, since all great ideas of architectural construction come from that unconscious moment, when you do not realise what it is, when there is no concrete shape, only a whisper which is not a whisper, an image which is not an image, 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;  
}
```

Why Session Types?

Significant academic and industry interests via a fundamental breakthrough



Why Session Types?

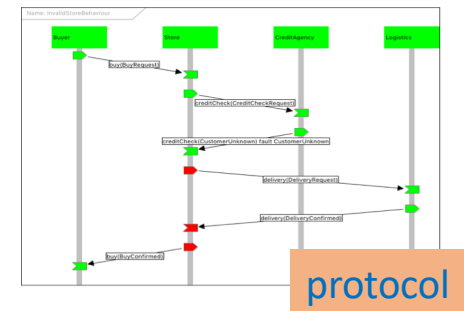
Significant academic and industry interests via a fundamental breakthrough

Binary Session Types ESOP'98



Joined W3C 2002

Scribble at π 4 technology

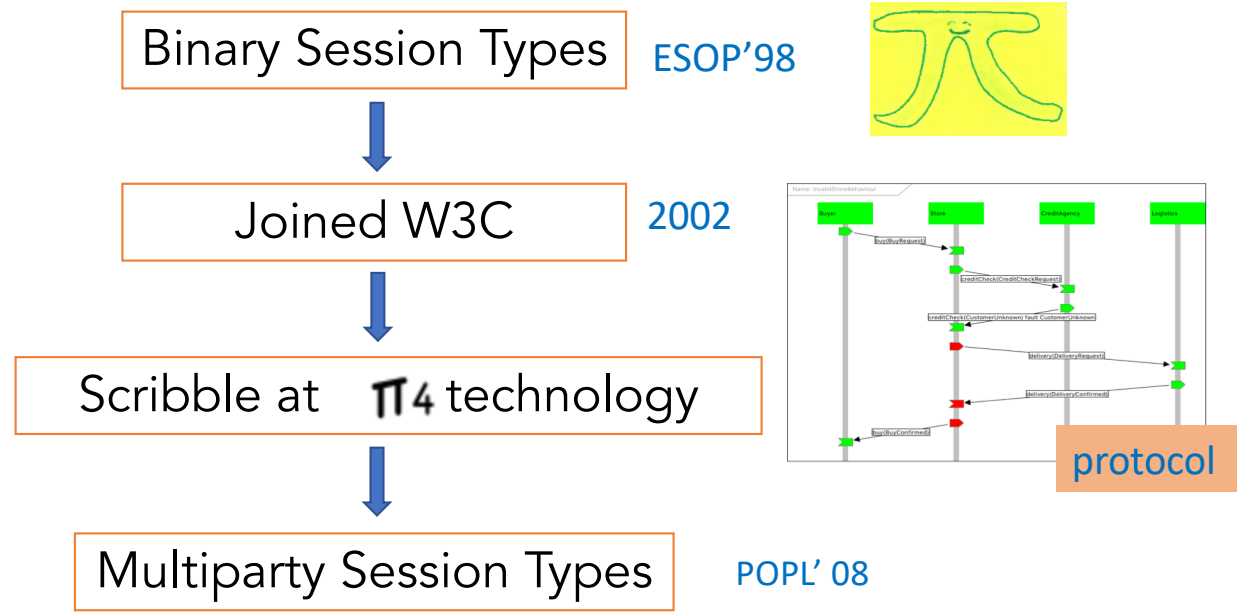


Multiparty Session Types POPL'08



Why Session Types?

Significant academic and industry interests via a fundamental breakthrough





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

POPL 2008 MOST INFLUENTIAL PAPER AWARD



POPL 2008 Most Influential Paper Award

Kohei Honda, Nobuko Yoshida and Marco Carbone

Multiparty asynchronous session types





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.

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

Mobility Research Group



π -calculus, Session Types research at Imperial College

Home

People

Publications

Grants

Talks

Tutorials

Tools

Awards

Kohei Honda

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 FERREIRA

Raymond HU

Rumyana NEYKOVA

Nicholas NG

Alceste SCALIA

PhD Students:

Assel ALTAYEV

Juliana FRANCO

Eva GRAVERSEN

Interactions with Industries

Strange Loop

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



Nobuko Yoshida
Imperial College, London



Adam Bowen @adamnbowen · Sep 15

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

DoC researcher to speak at Golang UK conference

by Vicky Kapogianni
20 July 2016



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

[Click here to add content](#)



.@nicholascwng rocking on @GolangUKconf about static deadlock detection in [#golang](#) [#gouk16](#)





GOLANG UK CONFERENCE

16th*, 17th & 18th AUGUST 2017

The Brewery, London



Imperial College
London

Home

College and Campus

Science

Engineering

Health

Business

Search here...

Go

Go concurrency verification research at DoC grabs headline

POPL'17

the morning paper

ICSE'18

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

Home About Info QR Editions Subscribe

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

currency
rates a

featured in the
which
interesting
easily
le of the
L (Principles

CC'18

A Session Type Provider

Compile-Time API Generation of Distributed Protocols with Refinements in F#

Rumyana Neykova
Imperial College London
United Kingdom

Raymond Hu
Imperial College London
United Kingdom

Nobuko Yoshida
Imperial College London
United Kingdom

Fahd Abdeljallal
Imperial College London
United Kingdom

Abstract

We present a library for the specification and implementation of distributed protocols in native F# (and other .NET languages) based on multiparty session types (MPST). There are two main contributions. Our library is the first practical development of MPST to support what we refer to as *interaction refinements*: a collection of features related to the refinement of *protocols*, such as message-type refinements (value constraints) and message-value dependent control flow. A well-typed endpoint program using our library is guaranteed to perform only compliant session I/O actions on the refined protocol, up to premature termination. Our library is developed as a session *type provider*,

1 Introduction

Type providers [20, 27] are a .NET feature for a form of compile-time meta programming, designed to bridge between programming in statically typed languages such as F# and C#, and working with so-called *information spaces*—structured data sources such as SQL databases or XML data.

A type provider works as a compiler plugin that performs on-demand generation of *types*: it takes a schema for an external information space, and generates types that allow the data to be manipulated via a strongly-typed interface, with benefits such as static error detection and IDE auto-completion. For example, an instantiation of the in-built type provider for WSDL Web services [6] may look like



Graydon Hoare

@graydon_pub

(This stuff is _fantastic_)

11:31 PM - 11 Mar 2018

32 Retweets 83 Likes



shots fired @zeeshanlakhani · Mar 12

Replying to @graydon_pub @dsyme

Awesome!

Brendan Zabarauskas @brendanzab ·

Replying to @graydon_pub

This stuff fills me with hope!

Ryan Riley @panesofglass · Mar 12

Replying to @graydon_pub

This is amazing! I guess I need to switch



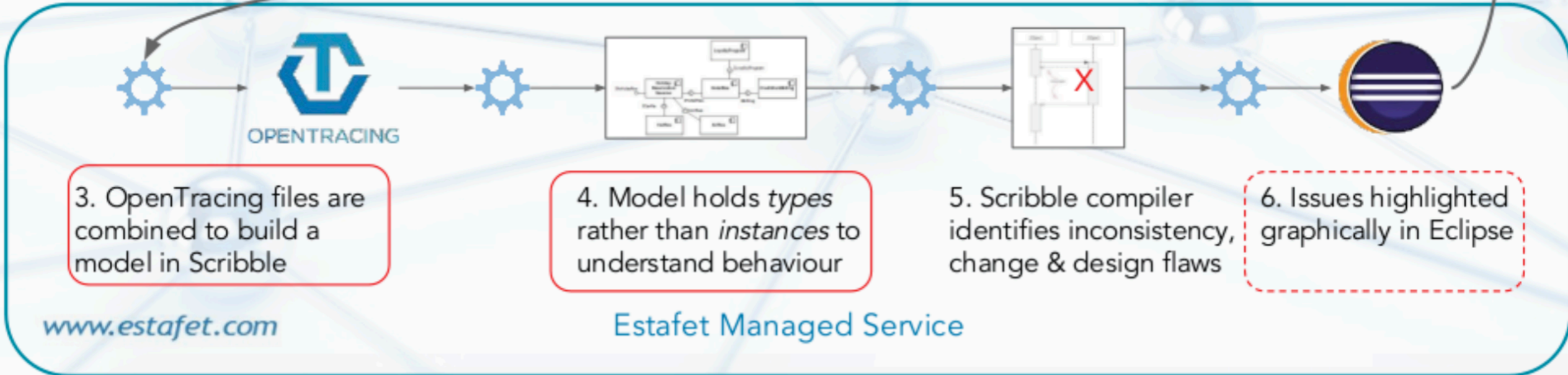
Scribble – Proving a distributed design



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




POPL 2019 Research Papers

[Write a Blog >>](#)

15:21 - 16:27: [Research Papers - Capabilities and Session Types I at POPL Track 2](#)


15:21 - 15:43 *Talk* ☆ [StkTokens: Enforcing Well-Bracketed Control Flow and Stack Encapsulation Using Linear Capabilities](#)
Lau Skorstengaard , Dominique Devriese Vrije Universiteit Brussel, Belgium, Lars Birkedal Aarhus University

15:43 - 16:05 *Talk*  ☆ [Two sides of the same coin: Session Types and Game Semantics](#)
Simon Castelan Imperial College London, UK, Nobuko Yoshida Imperial College London
[DOI](#) [Pre-print](#)

16:05 - 16:27 *Talk* ☆ [Exceptional Asynchronous Session Types: Session Types without Tiers](#)
Simon Fowler The University of Edinburgh, Sam Lindley University of Edinburgh, UK, J. Garrett Morris University of Kansas, USA, Sara Décova
[Pre-print](#)

16:37 - 17:43: [Research Papers - Session Types II at POPL Track 2](#)

16:37 - 16:59 *Talk*  ☆ [Interconnectability of Session-Based Logical Processes](#) **TOPLAS**
Bernardo Toninho NOVA-LINCS, FCT/UNL, Nobuko Yoshida Imperial College London
[DOI](#) [Pre-print](#)

16:59 - 17:21 *Talk*  ☆ [Distributed Programming using Role-Parametric Session Types in Go](#)
David Castro Imperial College London, Raymond Hu Imperial College London, Sung-Shik Jongmans Open University of the Netherlands, Nicholas Ng Imperial College London, Nobuko Yoshida Imperial College London
[DOI](#) [Pre-print](#)

17:21 - 17:43 *Talk*  ☆ [Less is More: Multiparty Session Types Revisited](#)
Alceste Scalas Imperial College London, Nobuko Yoshida Imperial College London
[DOI](#) [Pre-print](#)



Selected Publications [2018-2019]

- [PLDI19] [Alceste Scalas](#), NY, Elias Benussi: [Verifying message-passing programs with dependent behavioural types](#).
- [CAV19] [Julien Lange](#), NY: [Verifying Asynchronous Interactions via Communicating Session Automata](#).
- [CONCUR19] Mario Bravetti, Marco Carbone, [Julien Lange](#), NY, Gianluigi Zavattaro: [A Sound Algorithm for Asynchronous Session Subtyping](#)
- [FSE19] Nicola Atzei, Massimo Bartoletti, Stefano Lande, NY, Roberto Zunino: [Developing secure Bitcoin contracts with BitML](#)
- [ECOOP19] Rupak Majumdar, Marcus Pirron, NY, and Damien Zufferey, Motion Session Types for Robotic Interactions
- [FoSSaCs19] [Simon Castellan](#), NY: [Causality in Linear Logic](#)
- [ESOP19] [Laura Bocchi](#), Maurizio Murgia, Vasco T. Vasconcelos, NY: [Asynchronous Timed Session Types](#)
- [POPL19] [Simon Castellan](#), NY: [Two Sides of the Same Coin: Session Types and Game Semantics](#)
- [POPL19] [David Castro](#), [Raymond Hu](#), Sung-Shik Jongmans, [Nicholas Ng](#), NY: [Distributed Programming Using Role Parametric Session Types in Go](#)
- [POPL19] [Alceste Scalas](#), [Nobuko Yoshida](#): [Less Is More: Multiparty Session Types Revisited](#)
- [POPL19] [Bernardo Toninho](#), NY: [Interconnectability of Session Based Logical Processes](#)
- [ICSE18] [Julien Lange](#), [Nicholas Ng](#), [Bernardo Toninho](#), NY: [A Static Verification Framework for Message Passing in Go using Behavioural Types](#)
- [LICS18] [Romain Demangeon](#), NY: [Causal Computational Complexity of Distributed Processes](#)
- [FoSSaCs18] [Bernardo Toninho](#), [Nobuko Yoshida](#): [Depending On Session Typed Process](#)
- [ESOP18] [Bernardo Toninho](#), NY: [On Polymorphic Sessions And Functions: A Tale of Two \(Fully Abstract\) Encodings](#)
- [CC18] [Rumyana Neykova](#), [Raymond Hu](#), NY, Fahd Abdeljallal: [A Session Type Provider: Compile-time API Generation for Distributed Protocols with Interaction Refinements in F#](#)

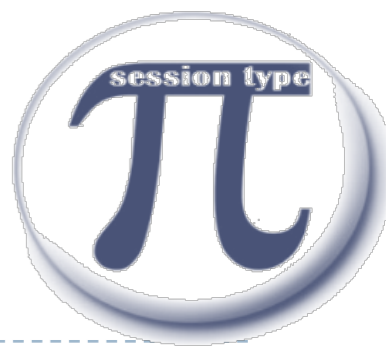
Selected Publications [2018-2019]

- [PLDI19] [Alceste Scalas](#), NY, [Elias Benussi](#): [Verifying message-passing programs with dependent behavioural types](#).
- [CAV19] [Julien Lange](#), NY: [Verifying Asynchronous Interactions via Communicating Session Automata](#).
- [CONCUR19] [Mario Bravetti](#), [Marco Carbone](#), [Julien Lange](#), NY, [Gianluigi Zavattaro](#): [A Sound Algorithm for Asynchronous Session Subtyping](#)
- [FSE19] [Nicola Atzei](#), [Massimo Bartoletti](#), [Stefano Lande](#), NY, [Roberto Zunino](#): [Developing secure Bitcoin contracts with BitML](#)
- [ECOOP19] [Rupak Majumdar](#), [Marcus Pirron](#), NY, and [Damien Zufferey](#): [Motion Session Types for Robotic Interactions](#)
- [FoSSaCs19] [Simon Castellan](#), NY: [Causality in Linear Logic](#)
- [ESOP19] [Laura Bocchi](#), [Maurizio Murgia](#), [Vasco T. Vasconcelos](#), NY: [Asynchronous Timed Session Types](#)
- [POPL19] [Simon Castellan](#), NY: [Two Sides of the Same Coin: Session Types and Game Semantics](#)
- [POPL19] [David Castro](#), [Raymond Hu](#), [Sung-Shik Jongmans](#), [Nicholas Ng](#), NY: [Distributed Programming Using Role Parametric Session Types in Go](#)
- [POPL19] [Alceste Scalas](#), [Nobuko Yoshida](#): [Less Is More: Multiparty Session Types Revisited](#)
- [POPL19] [Bernardo Toninho](#), NY: [Interconnectability of Session Based Logical Processes](#)
- [ICSE18] [Julien Lange](#), [Nicholas Ng](#), [Bernardo Toninho](#), NY: [A Static Verification Framework for Message Passing in Go using Behavioural Types](#)
- [LICS18] [Romain Demangeon](#), NY: [Causal Computational Complexity of Distributed Processes](#)
- [FoSSaCs18] [Bernardo Toninho](#), [Nobuko Yoshida](#): [Depending On Session Typed Process](#)
- [ESOP18] [Bernardo Toninho](#), NY: [On Polymorphic Sessions And Functions: A Tale of Two \(Fully Abstract\) Encodings](#)
- [CC18] [Rumyana Neykova](#), [Raymond Hu](#), NY, [Fahd Abdeljallal](#): [A Session Type Provider: Compile-time API Generation for Distributed Protocols with Interaction Refinements in F#](#)

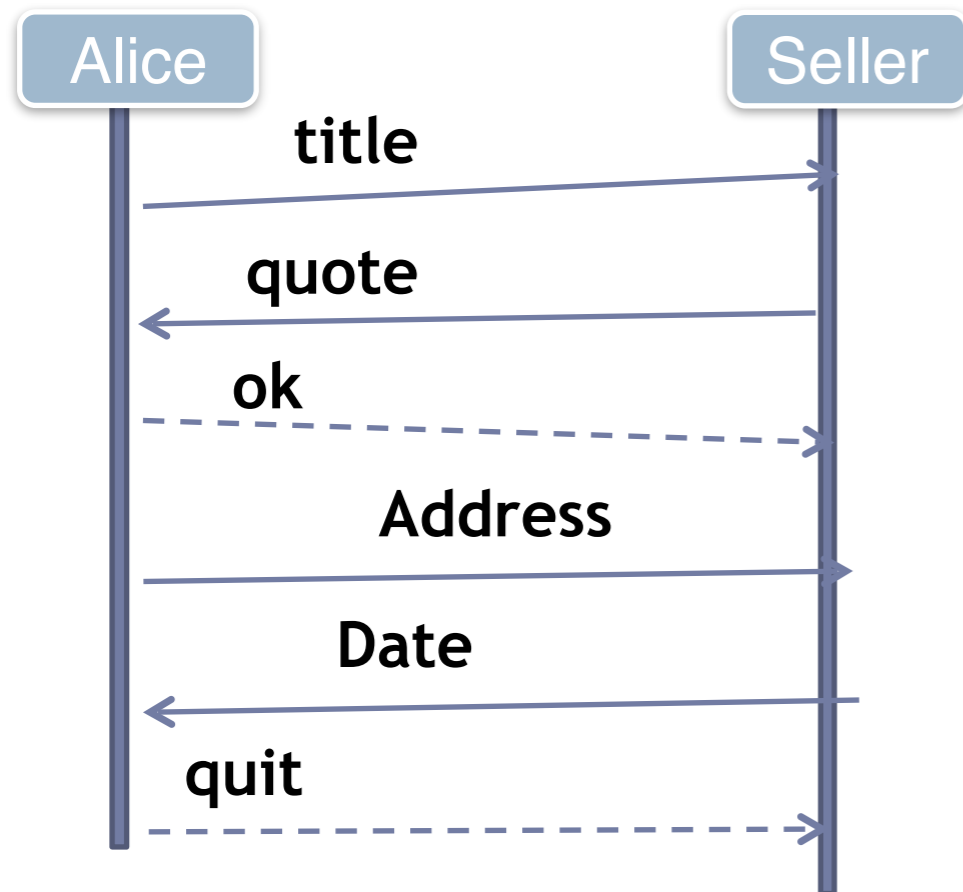


Type Me If You Can: Introduction to Session Types and Scribble

Rumyana Neykova, Nobuko Yoshida



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?

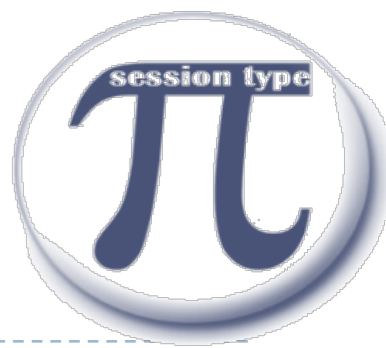
send(int).send(int).receive(bool)



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

It is all about duality!





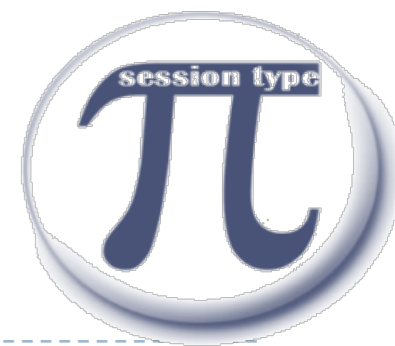
Are we compatible?

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

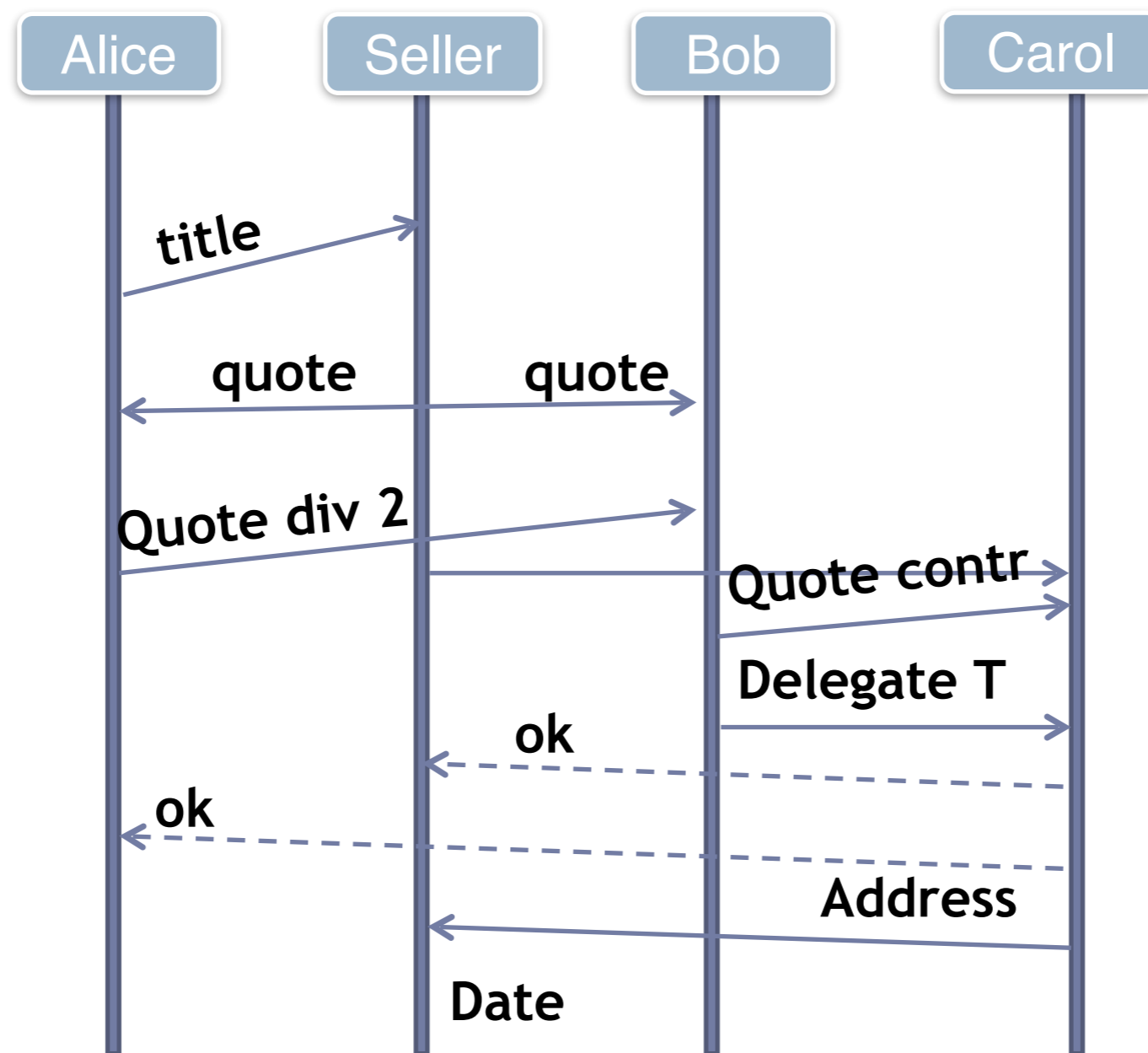


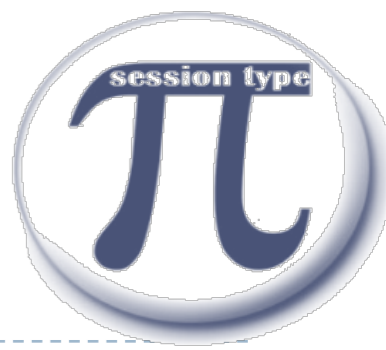
receive(int).receive(int).send(bool)



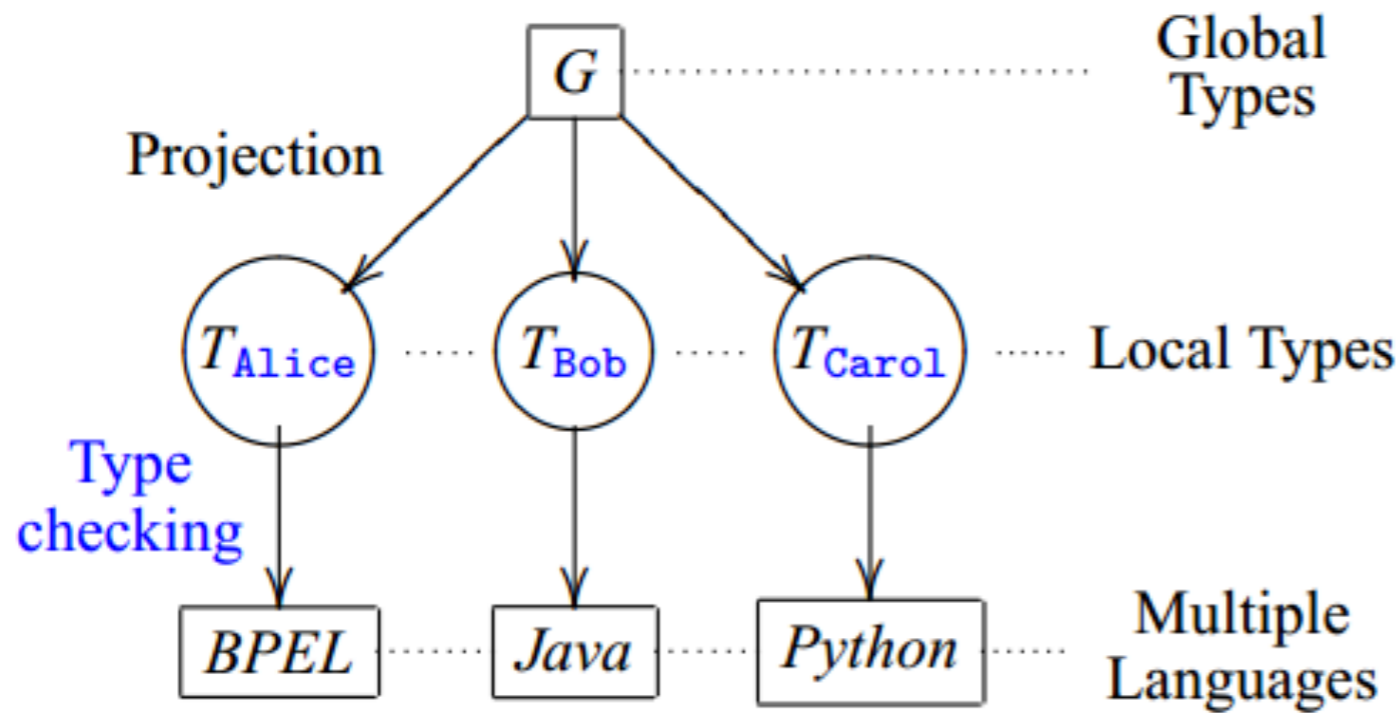


Wait a minute! What if it is more than 2?





How does it work?

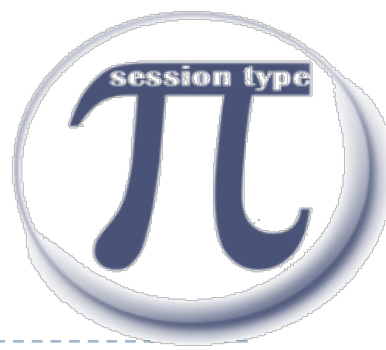


$Alice \rightarrow Bob: \langle Nat \rangle.$
 $Bob \rightarrow Carol: \langle Nat \rangle.end$

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

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

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

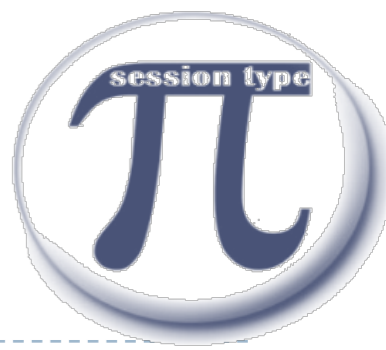


Session Types in a Nutshell

SESSION = STRUCTURED SEQUENCE OF COMMUNICATION

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





What is type safe communication?

Communication Safety

- No communication mismatch

Session Fidelity

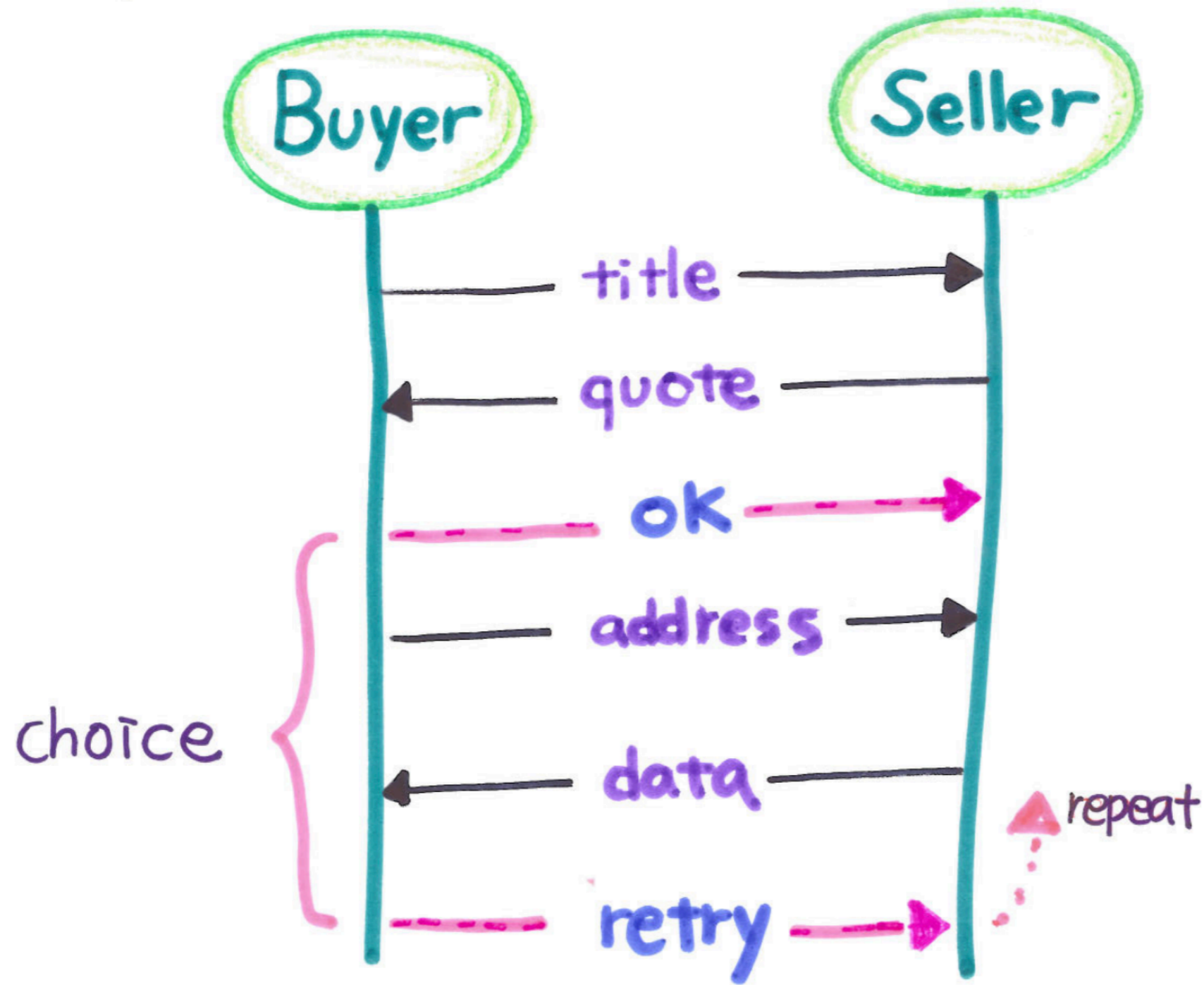
- Communication follow the described protocol

Progress

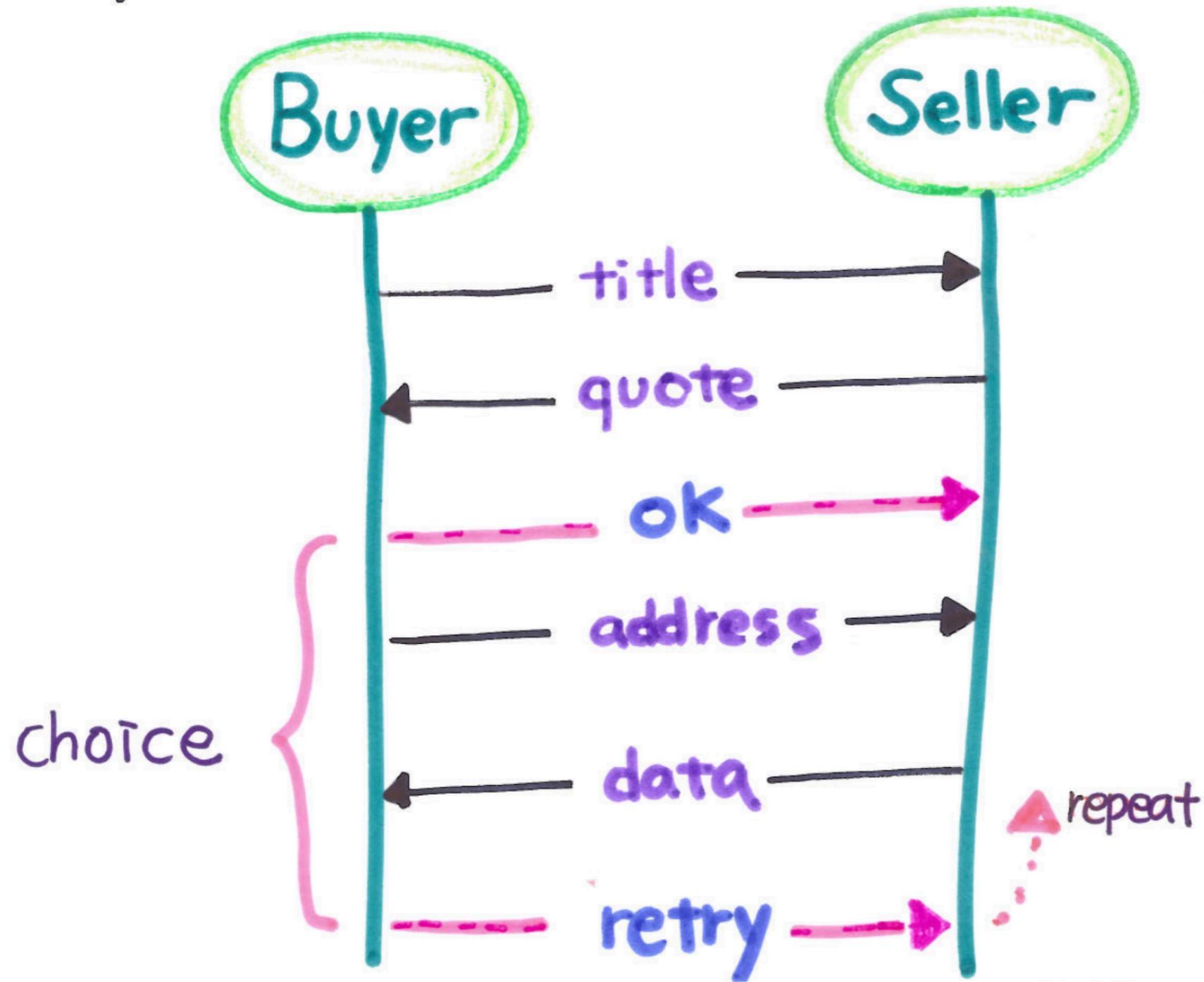
- No deadlock/ stuck in a session



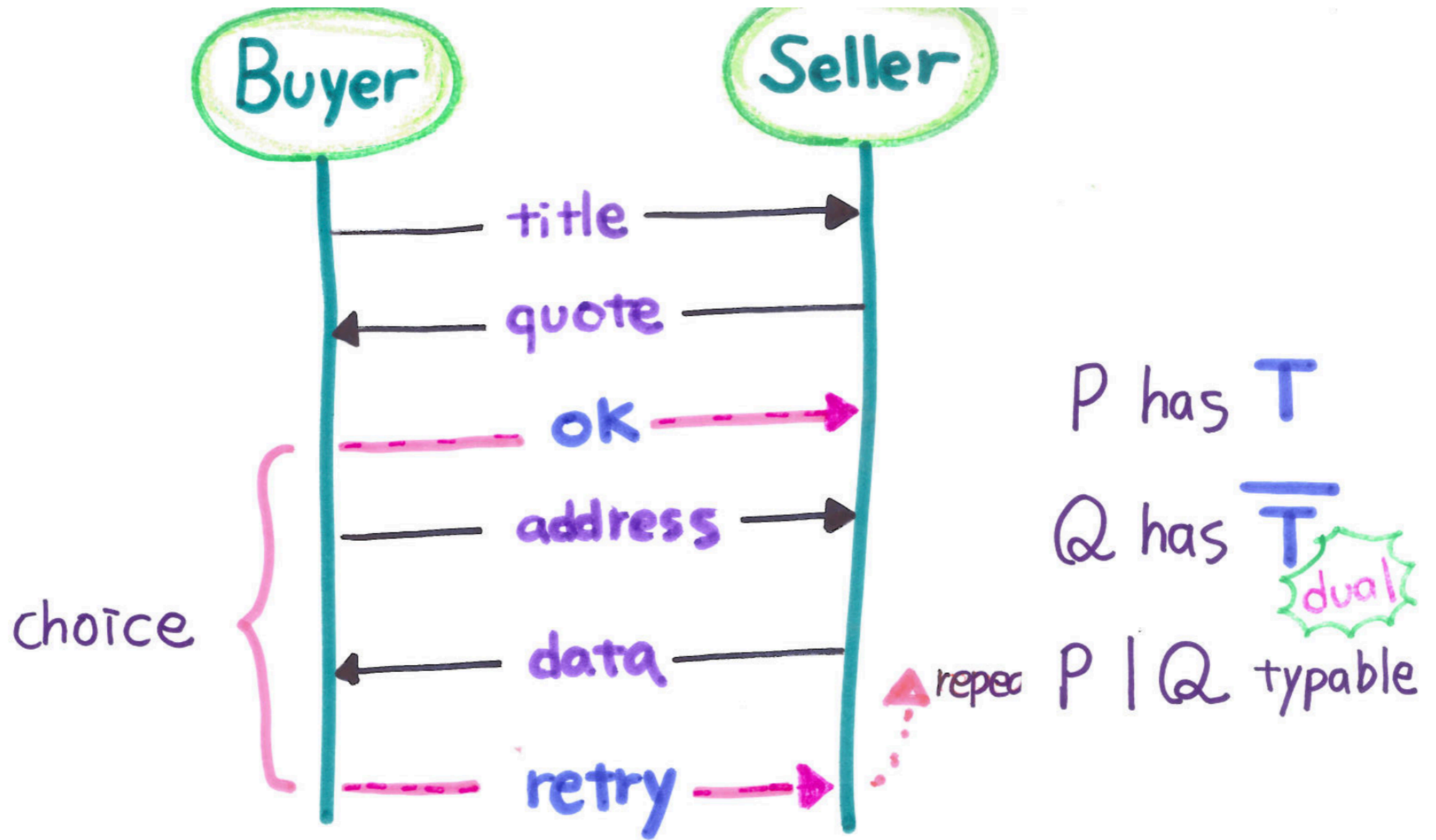
Binary Session Types: Buyer - Seller Protocol



Binary Session Types: Buyer - Seller Protocol



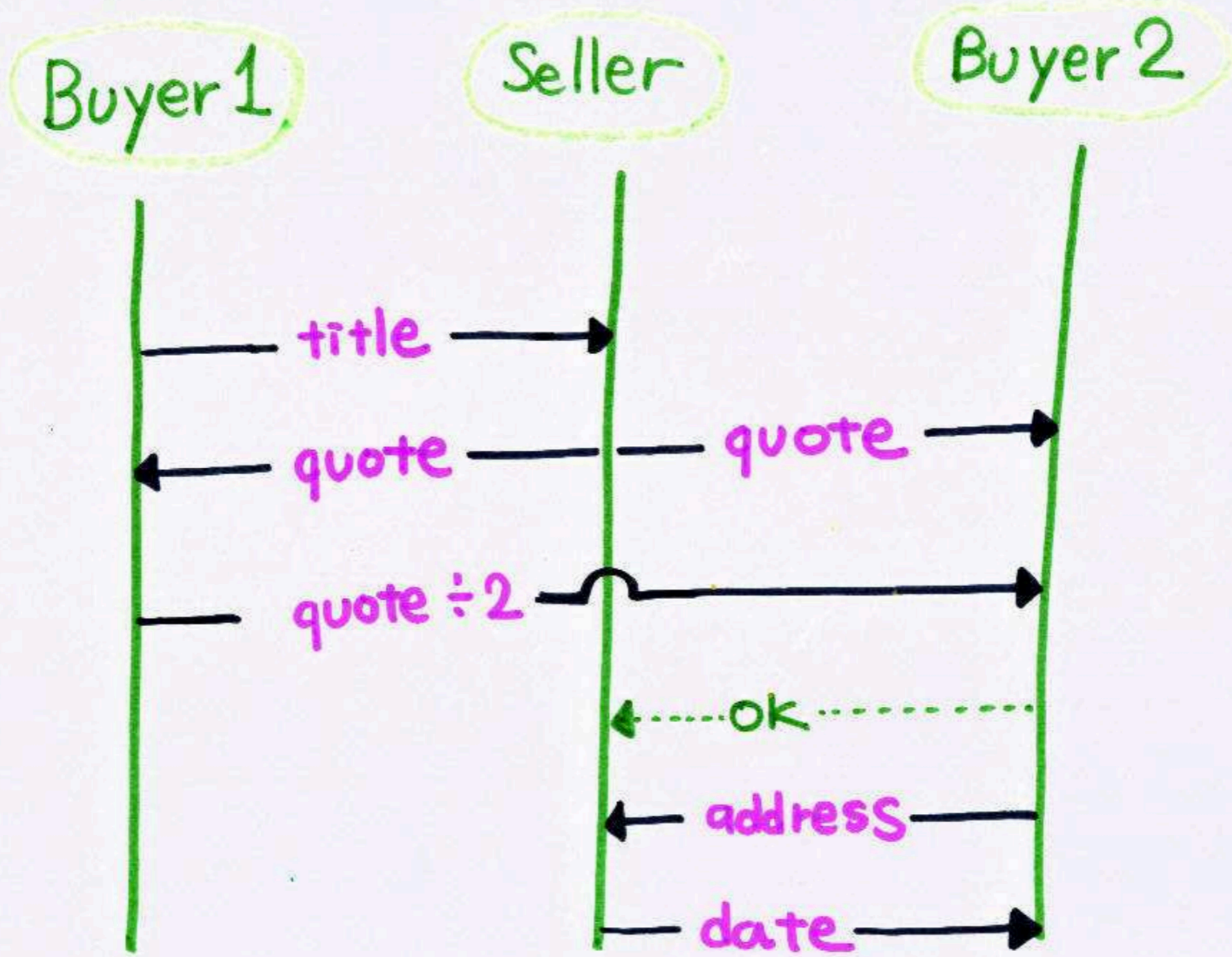
nt! Title ; ? Quote ; ! { ok: ! Add ; ? Date, retry: t }

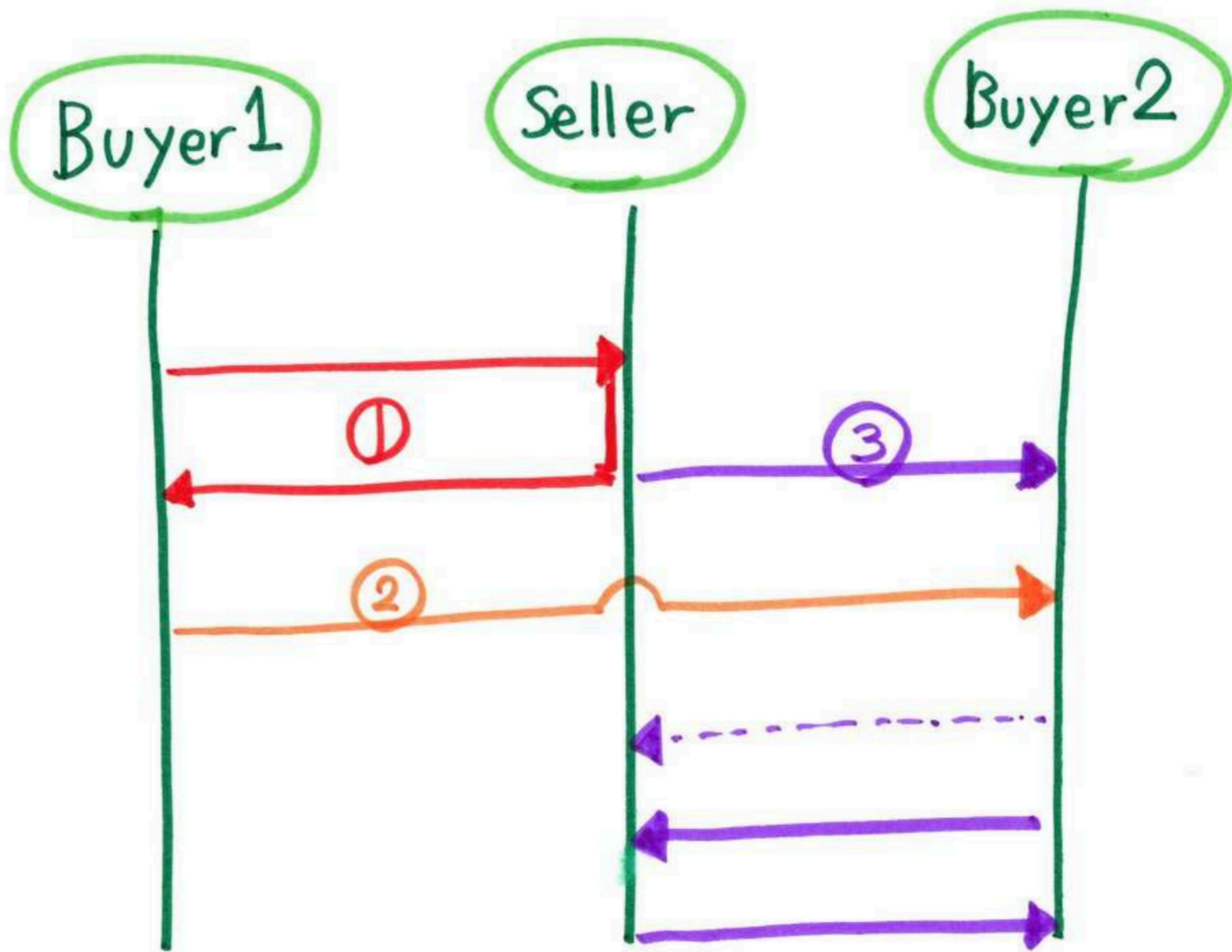


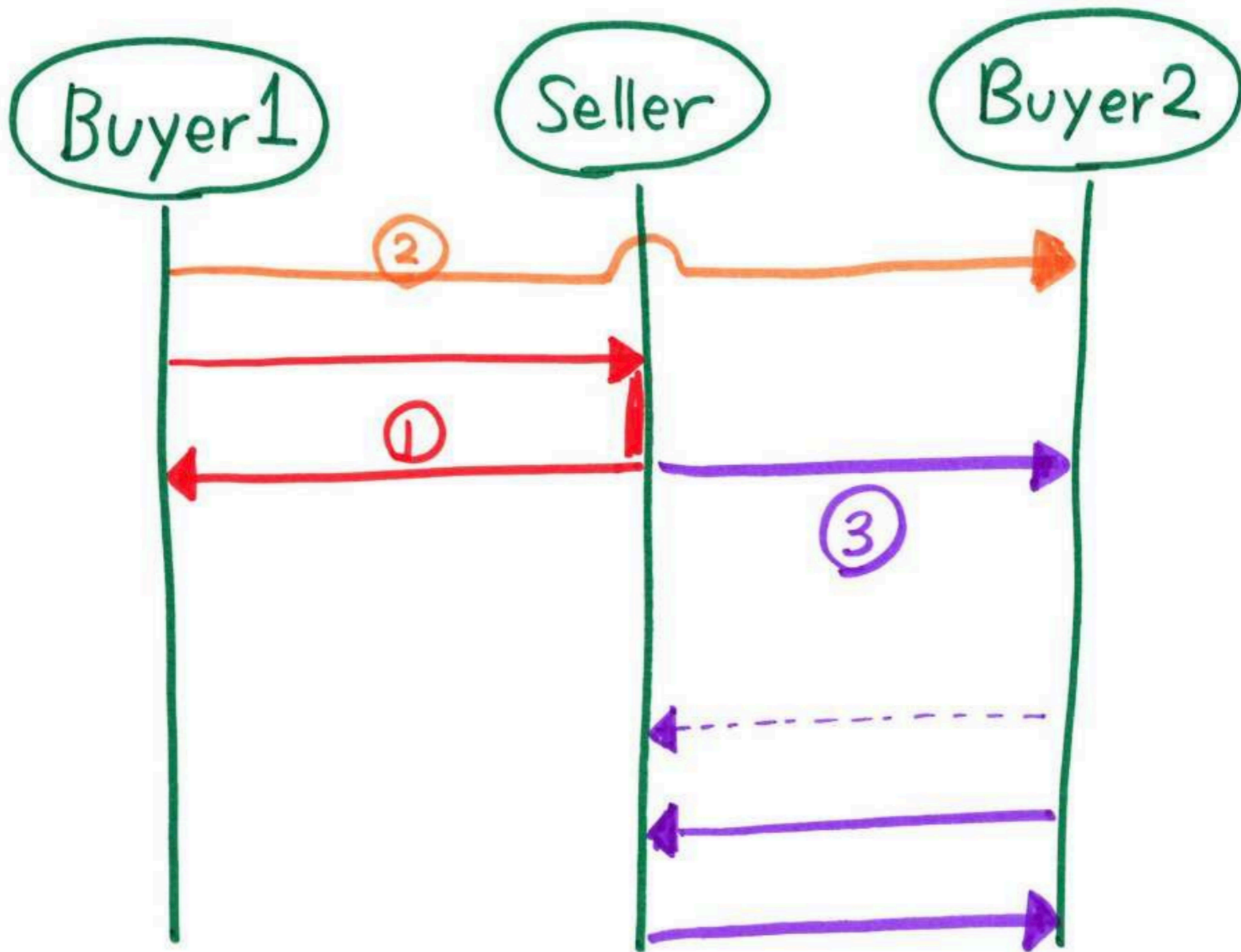
nt! Title ; ? Quote ; ! { ok: ! Add ; ? Date, retry: t }

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

Multiparty Session Types







Alice

Bob

Carol

CA? c ; AB! a

AB? a ; BC! b

BC? b ; CA? c

dual

dual

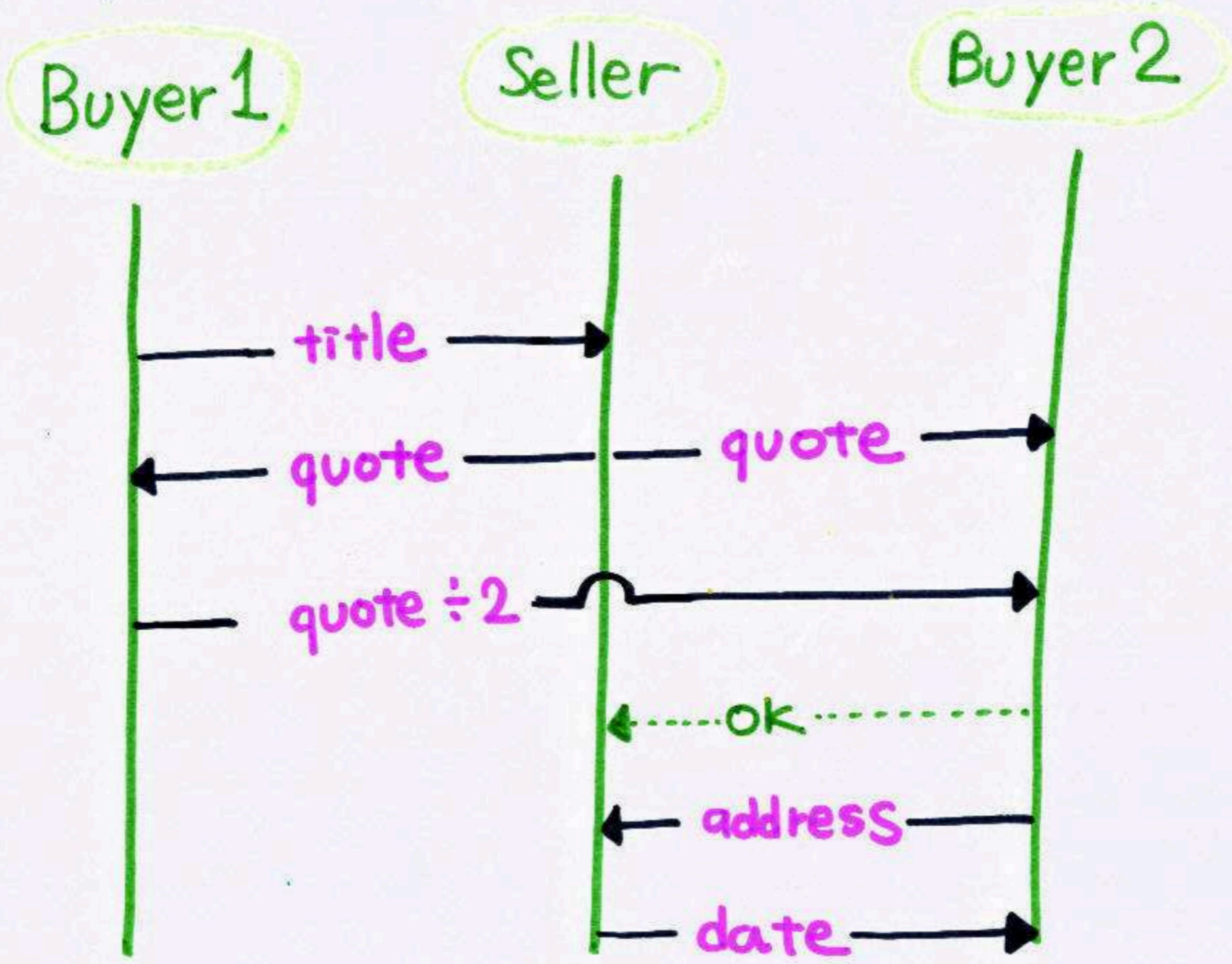
dual

3 dual pairs

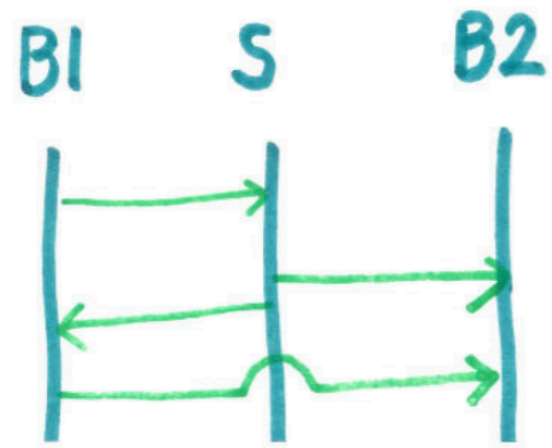
If you use
binary Session
Types

Deadlock!

Multiparty Session Types



Multi party Session Types [Honda, Yoshida, Carbone 2008]



ⓐ

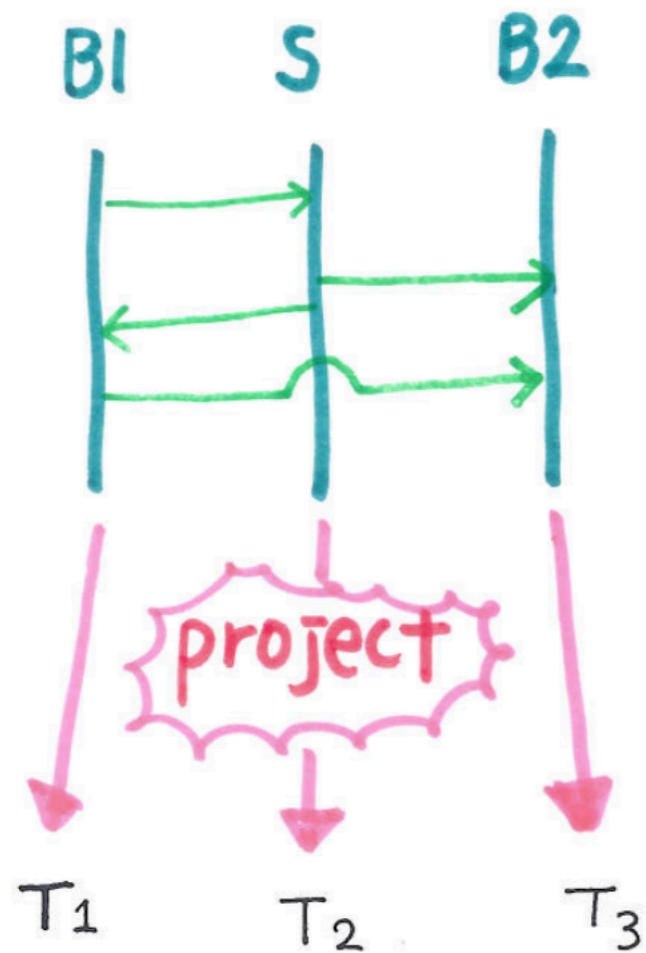
BI \rightarrow S Int.

S \rightarrow B2 Char

STEP 1

Write Global Type

Multi party Session Types [Honda, Yoshida, Carbone 2008]



(G) $B_1 \rightarrow S$ Int.
 $S \rightarrow B_2$ Char

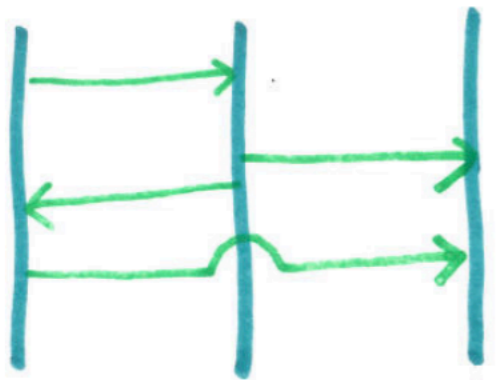
STEP 1
Write Global Type

(T) $B_1 ?$ Int. $B_2 !$ Char

STEP 2
Project to Local Types

Multi party Session Types [Honda, Yoshida, Carbone 2008]

B1 S B2



(G)

$B1 \rightarrow S \text{ Int.}$
 $S \rightarrow B2 \text{ Char}$

STEP 1
 Write Global Type

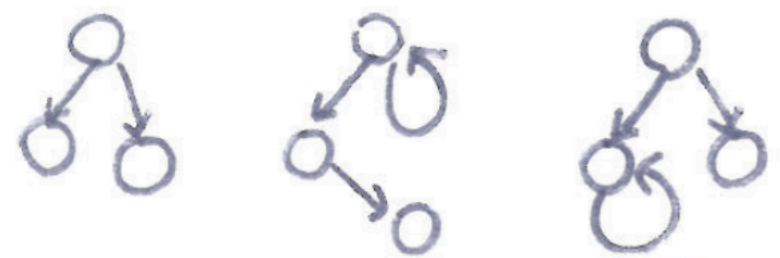
project

(T)

$B1? \text{Int. } B2! \text{Char}$

STEP 2
 Project to Local Type

T₁ T₂ T₃



P₁ P₂ P₃

(P) $B1?(x). B2! \langle \text{"apple"} \rangle$

STEP 3

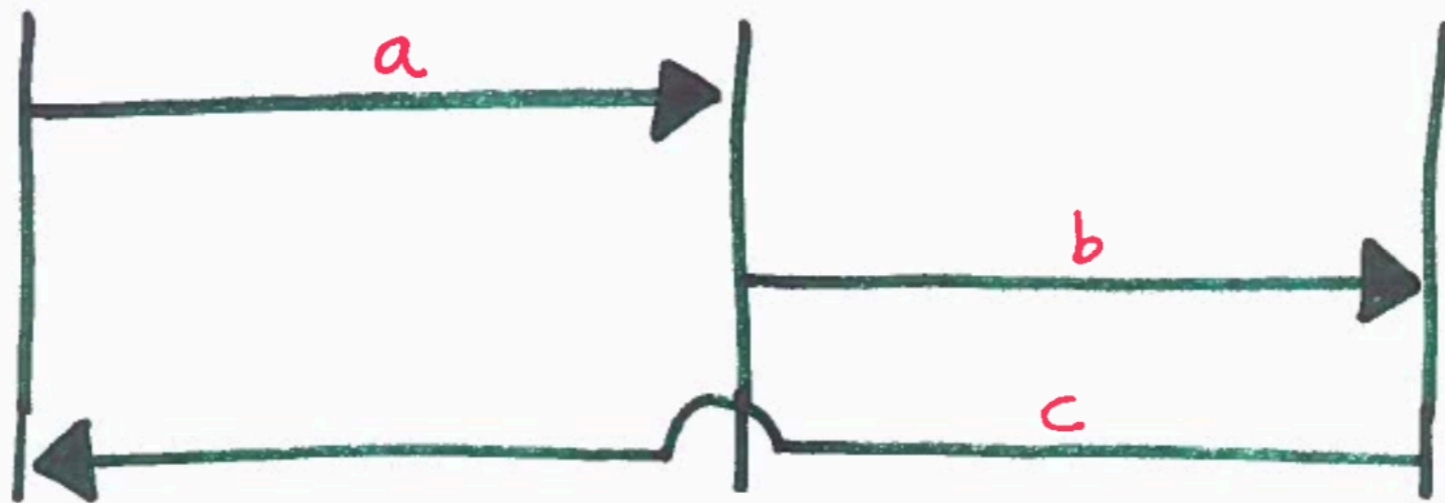
- Static Check
- Generate Code
- Run-time check



Alice

Bob

Carol



Global Type



LOCAL TYPES

Alice $AB!a; CA?c$

Bob $AB?a; BC!b$

Carol $BC?b; CA!c;$

NO Deadlock

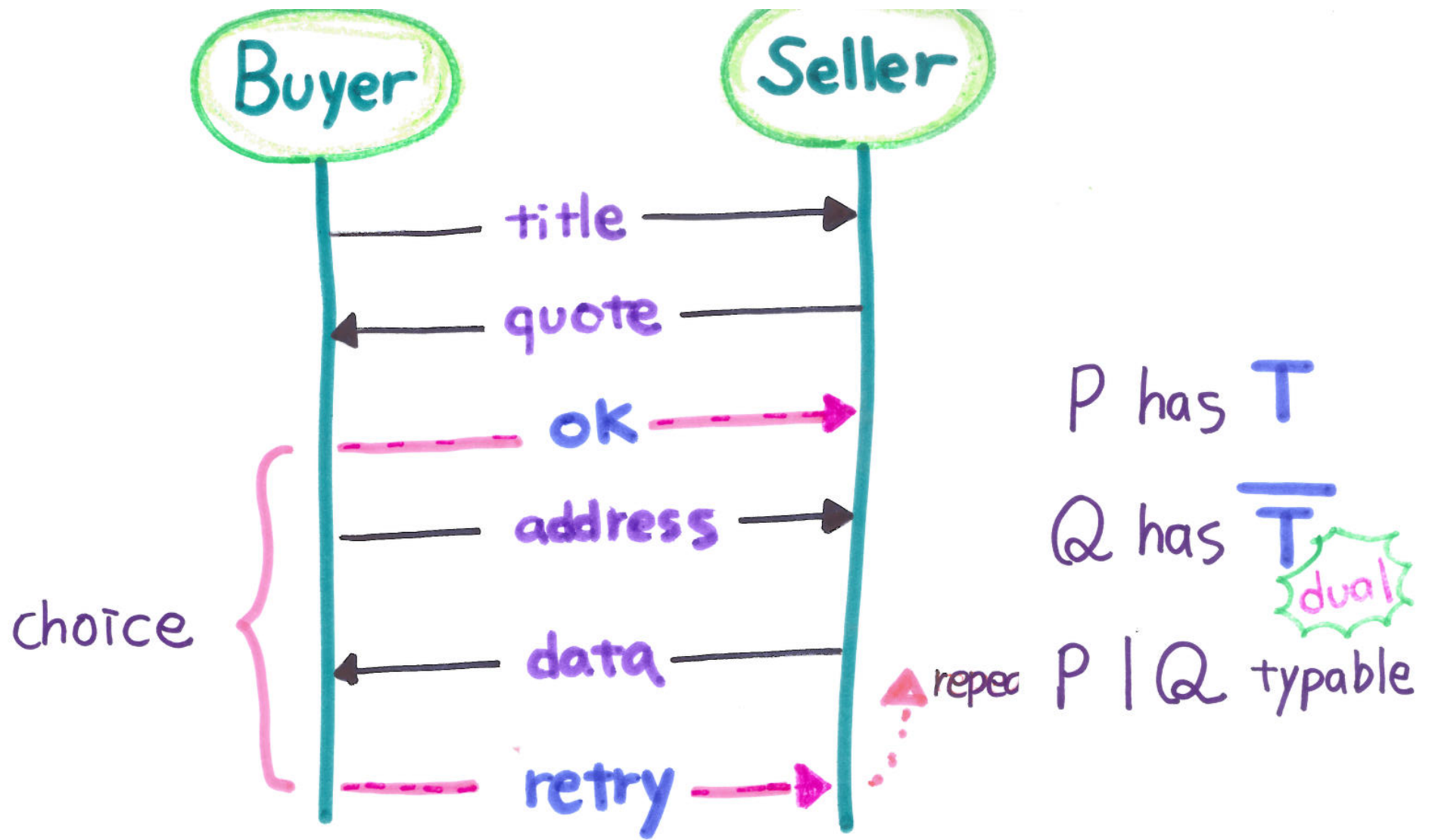
Properties of Session Types

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”



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

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

Relationship with Communicating AUTOMATA

ESOP 12

Characterisation

ICALP 13

Synthesis

CONCUR 15

Timed

Automata

POPL 15

Graphical

Synthesis

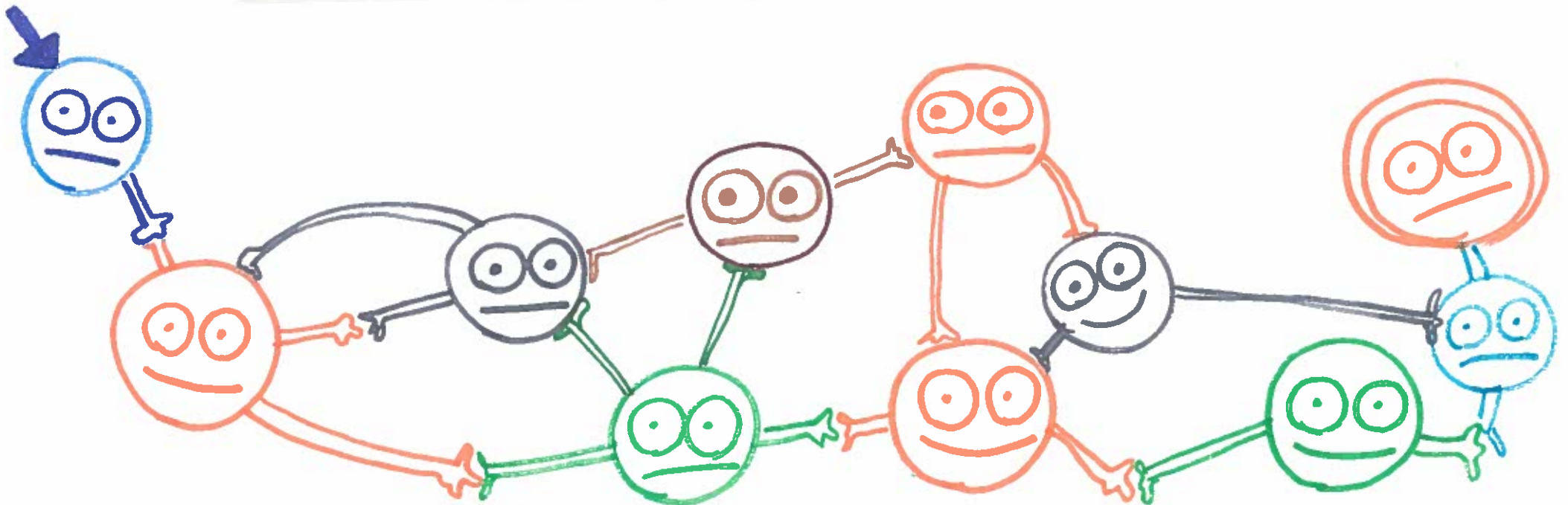
TACAS 16

Subtyping

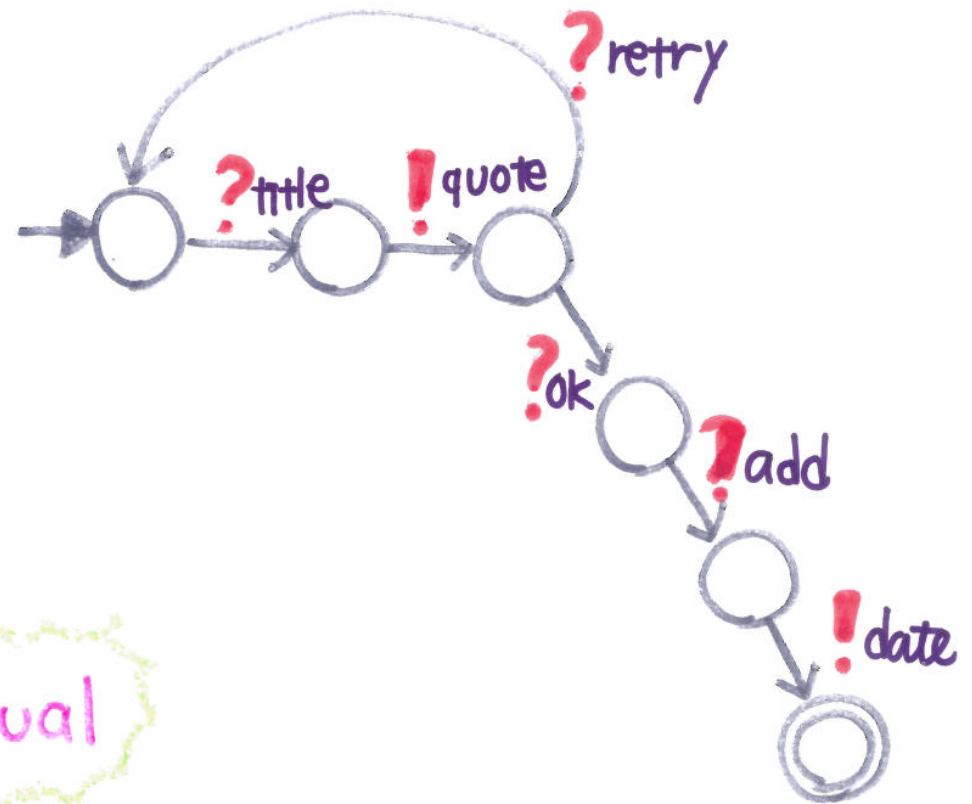
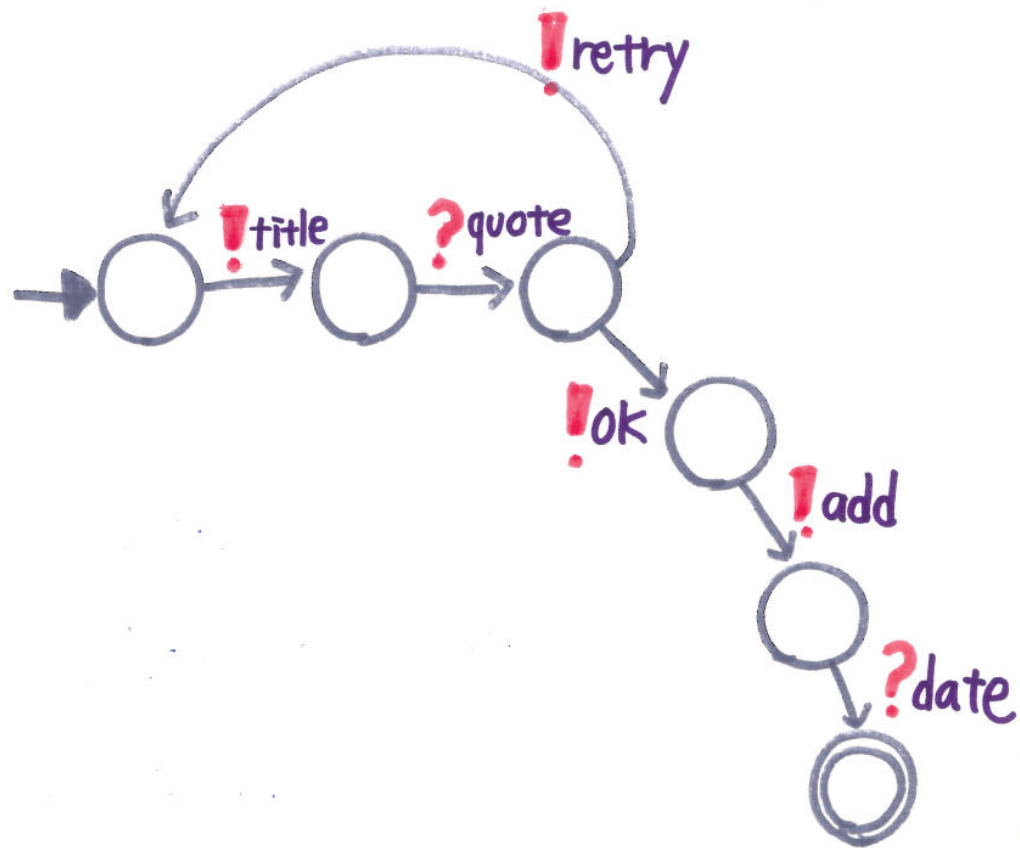
+ Model-checking

FOSSACS 17

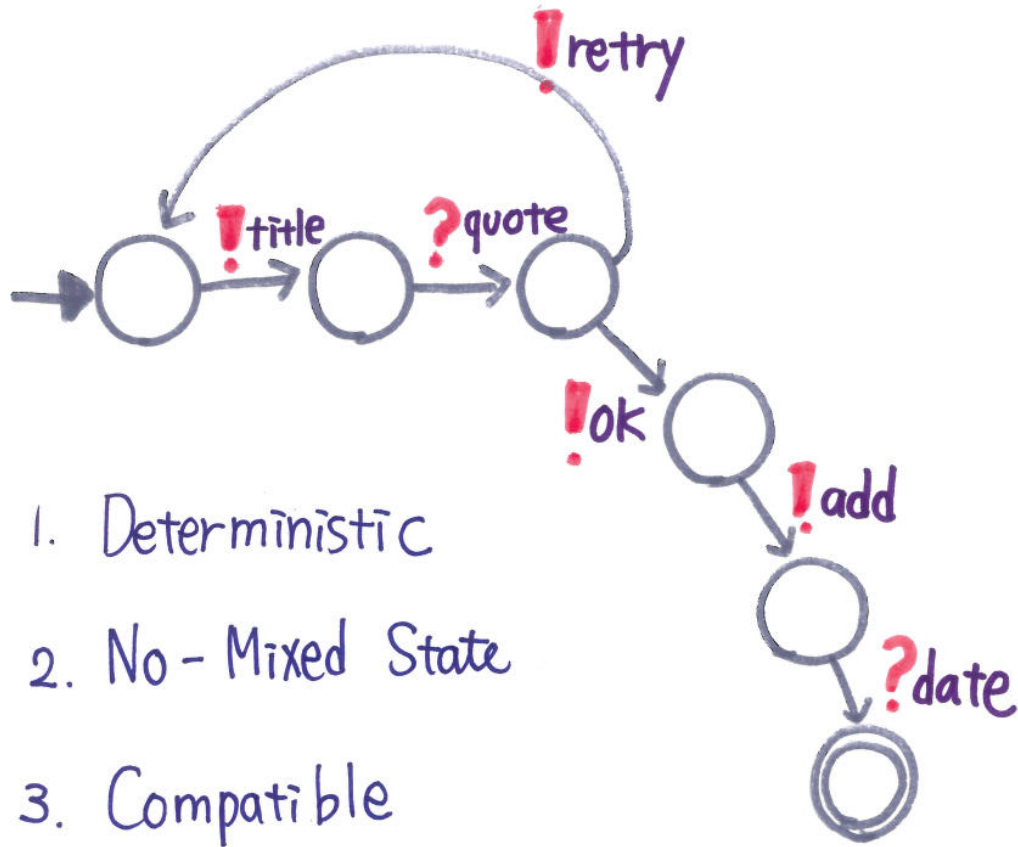
Undecidability



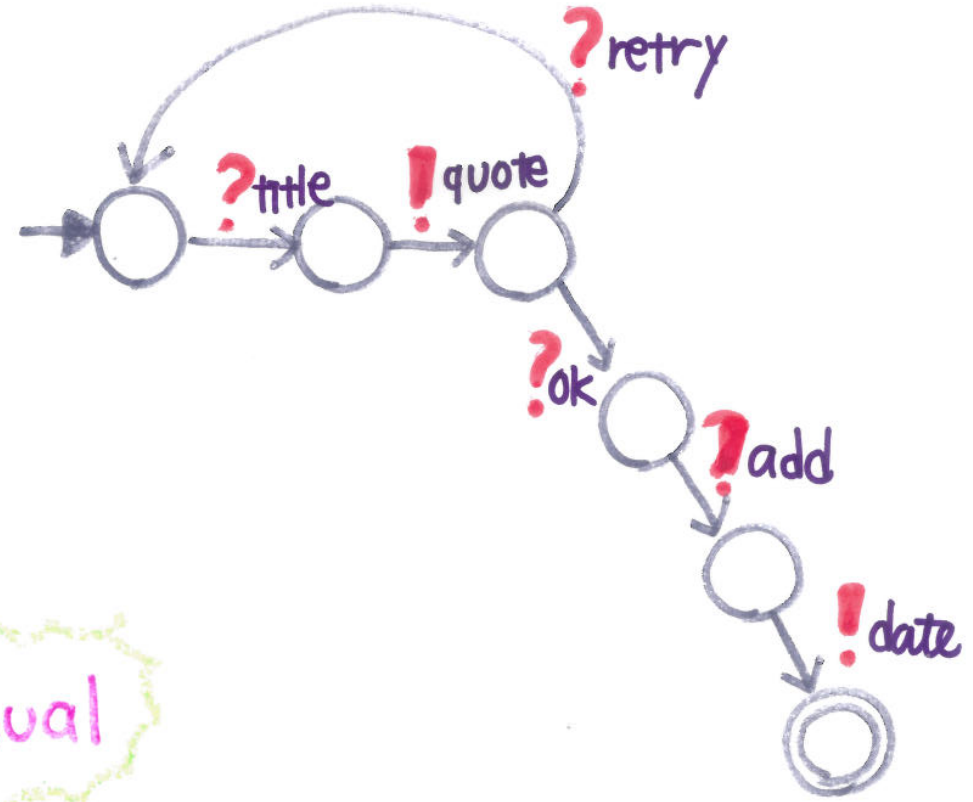
Communicating Automata [1980s]



dual



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



dual

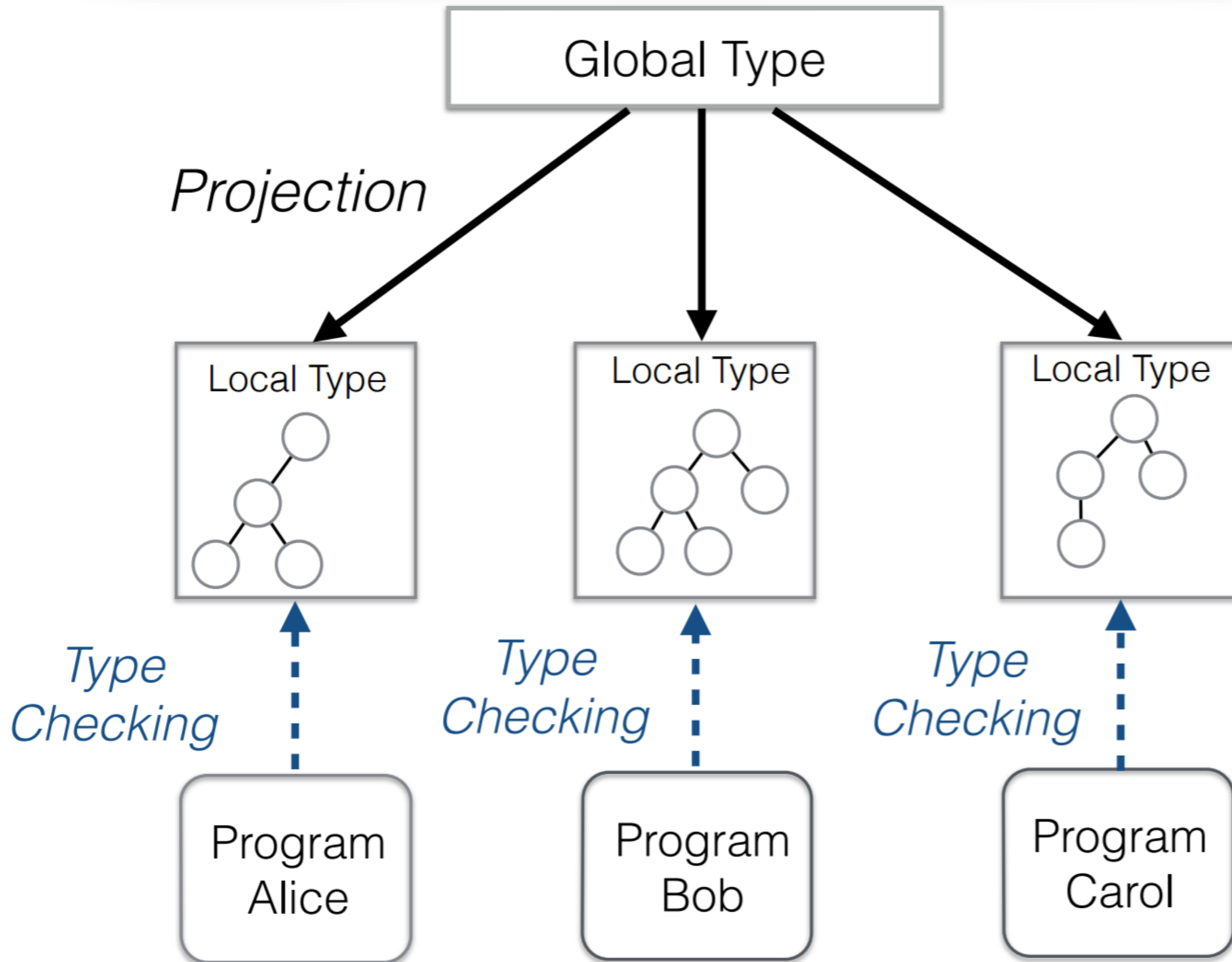
[Gouda et al 1986] Two compatible machines without mixed states which are deterministic satisfy deadlock-freedom.

Session Types

Applications

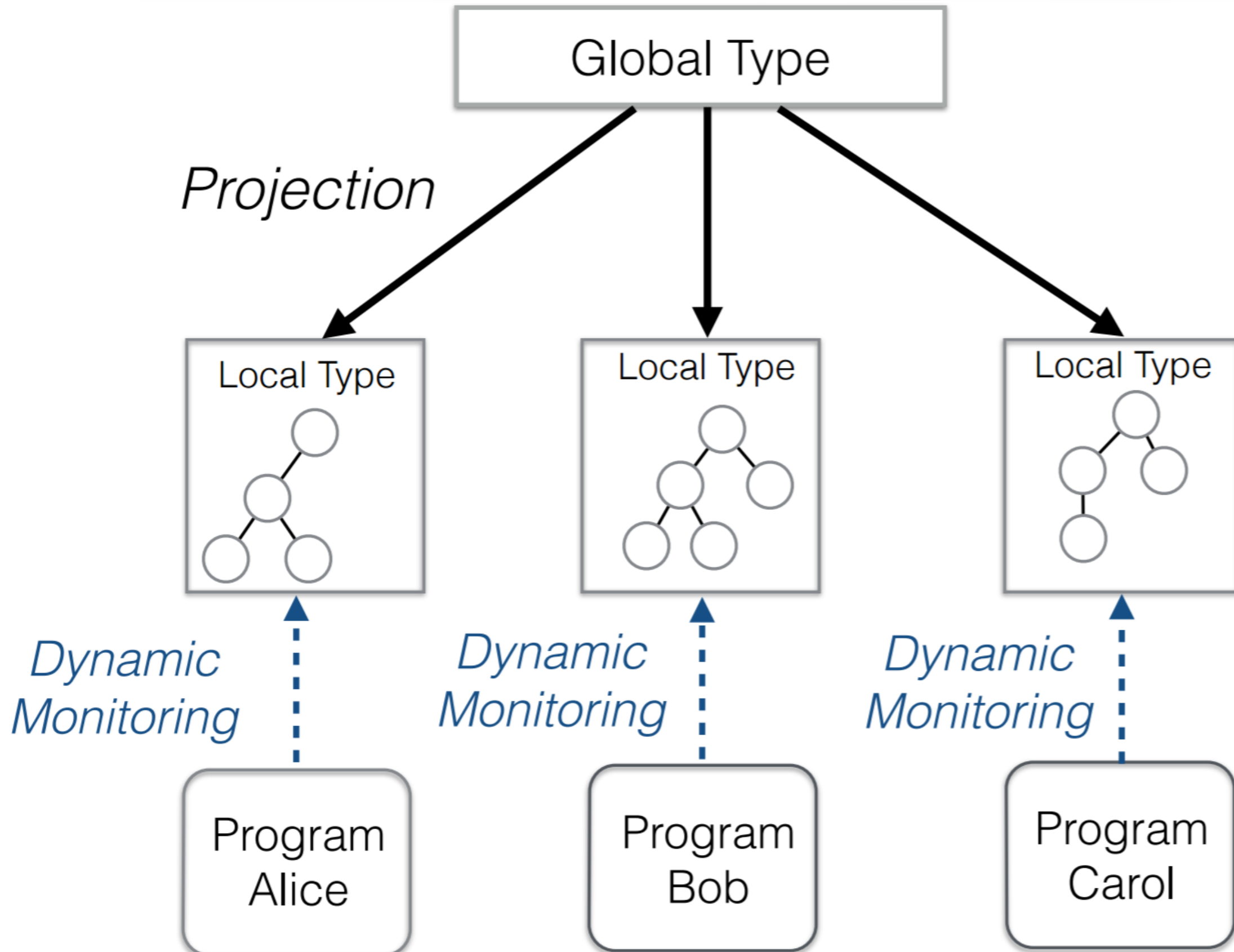
Type Checking

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

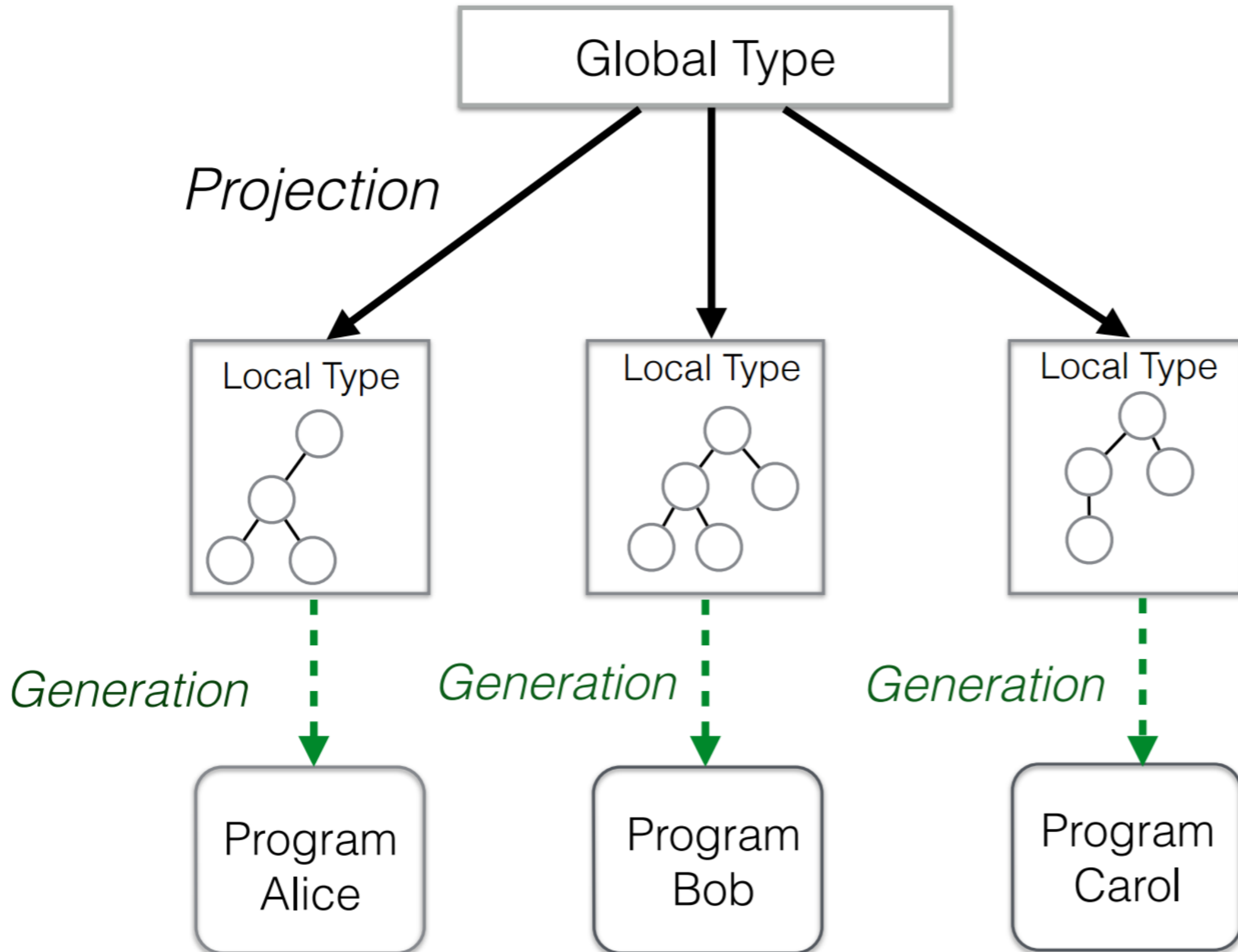


Dynamic Monitoring

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

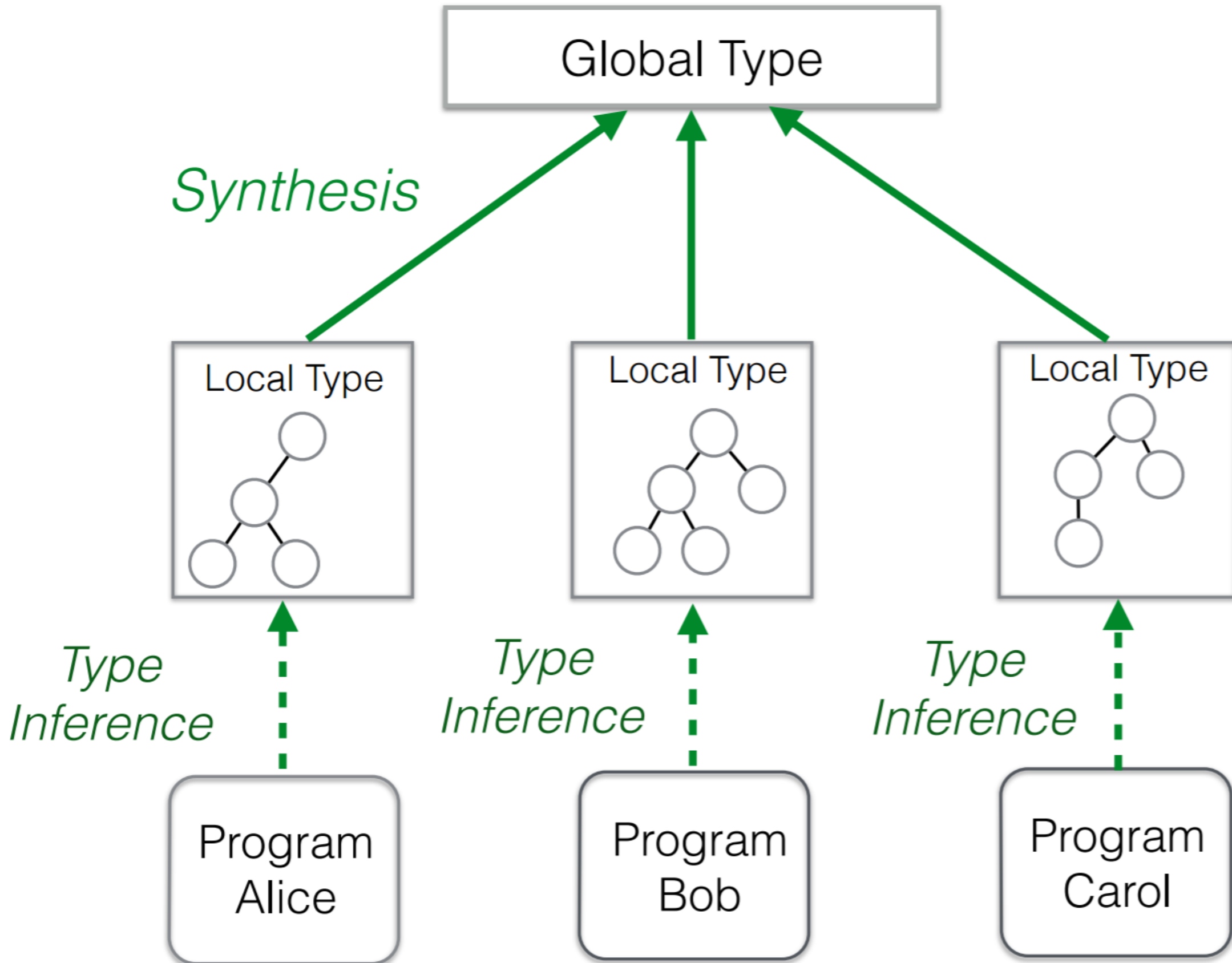


Code Generation [CC'15, FASE'16]



Synthesis

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



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





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](#)

Applications

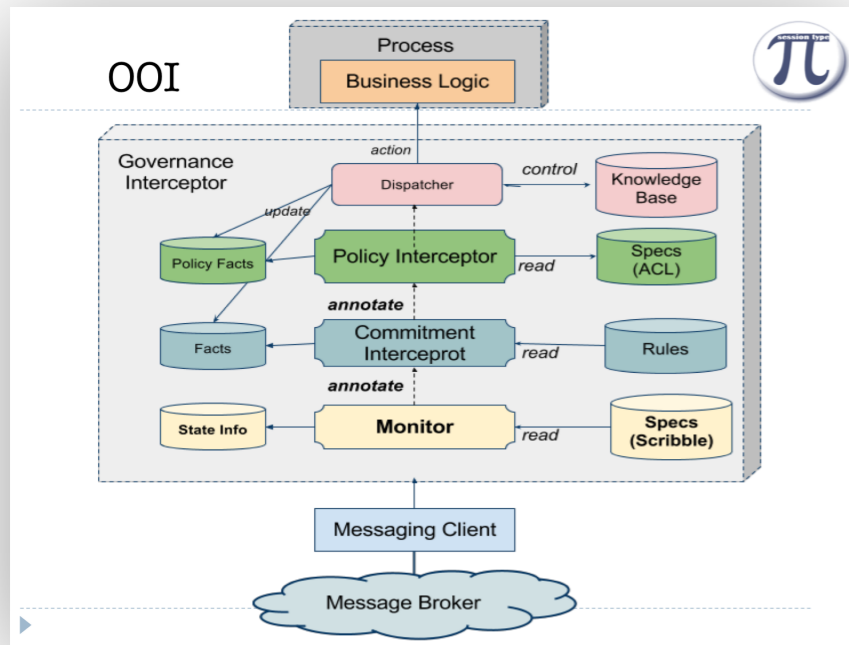


Session C

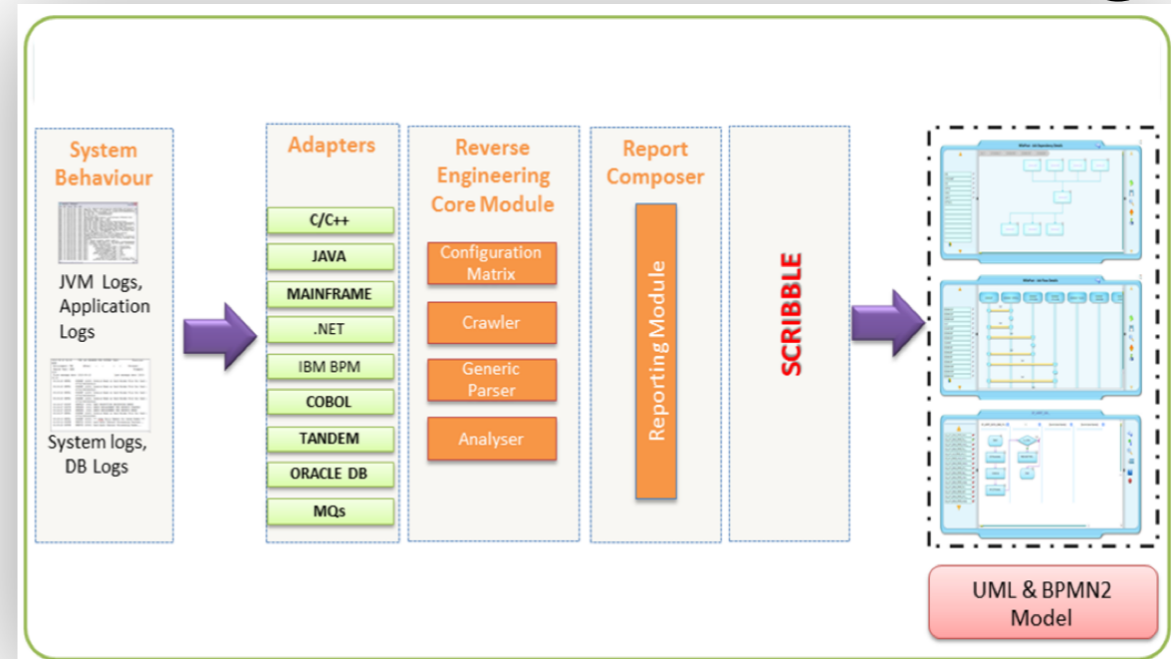


Session Type Based Tools

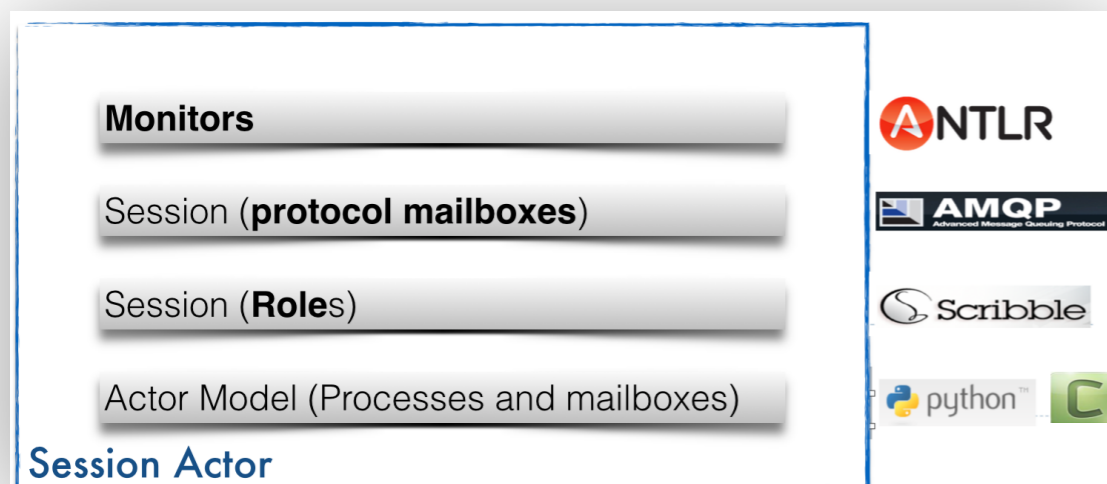
OOI Governance



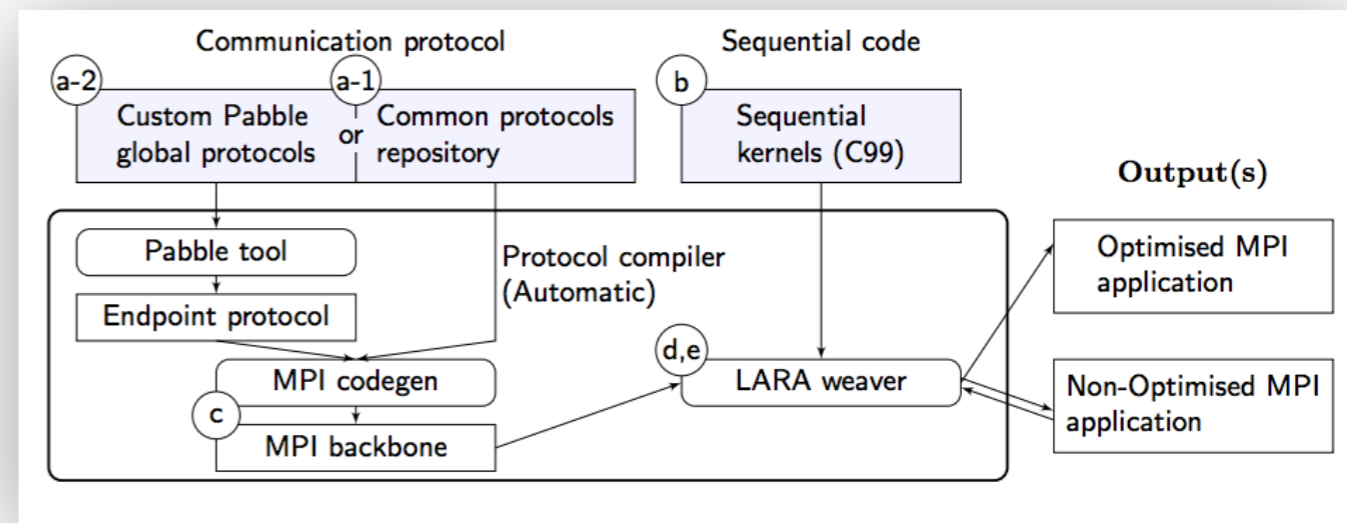
ZDLC: Process Modeling



Actor Verification

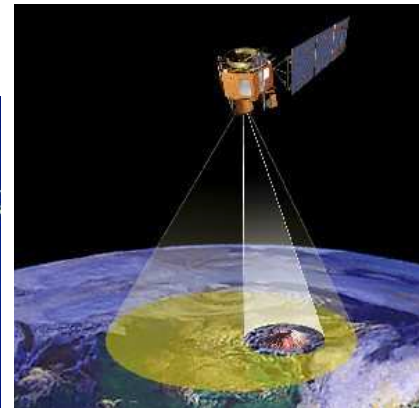
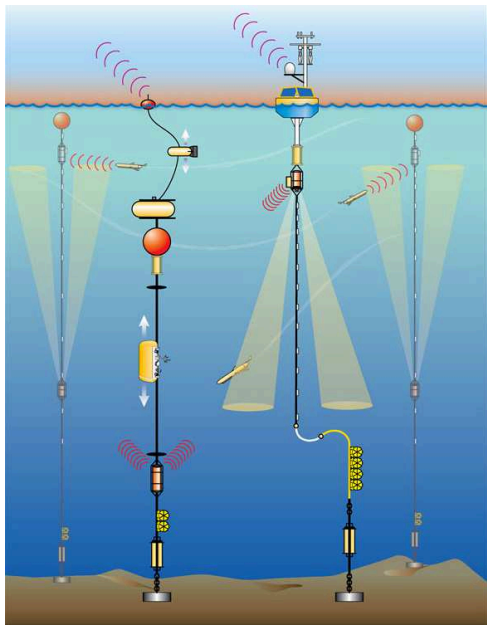


MPI code generations

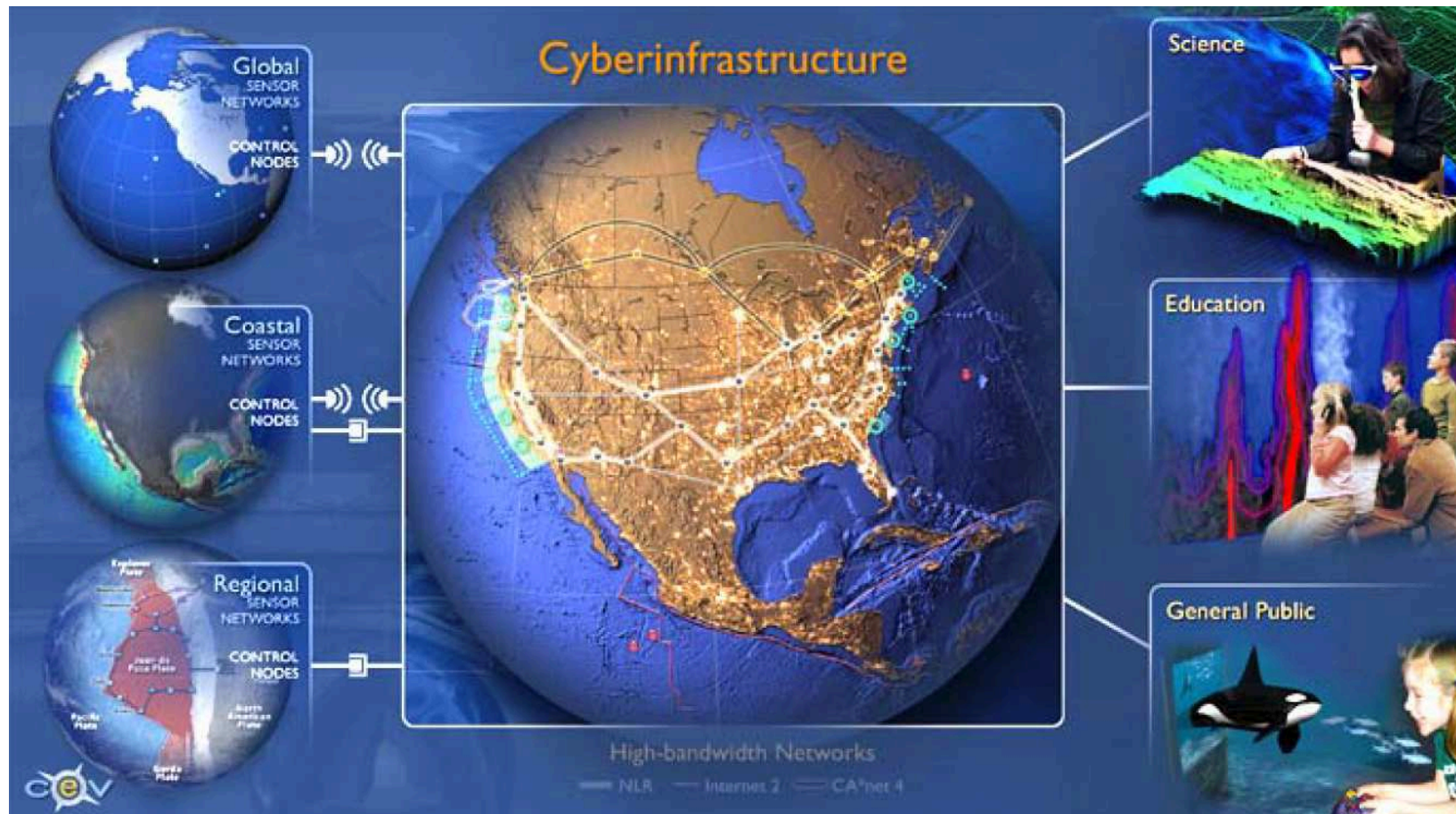


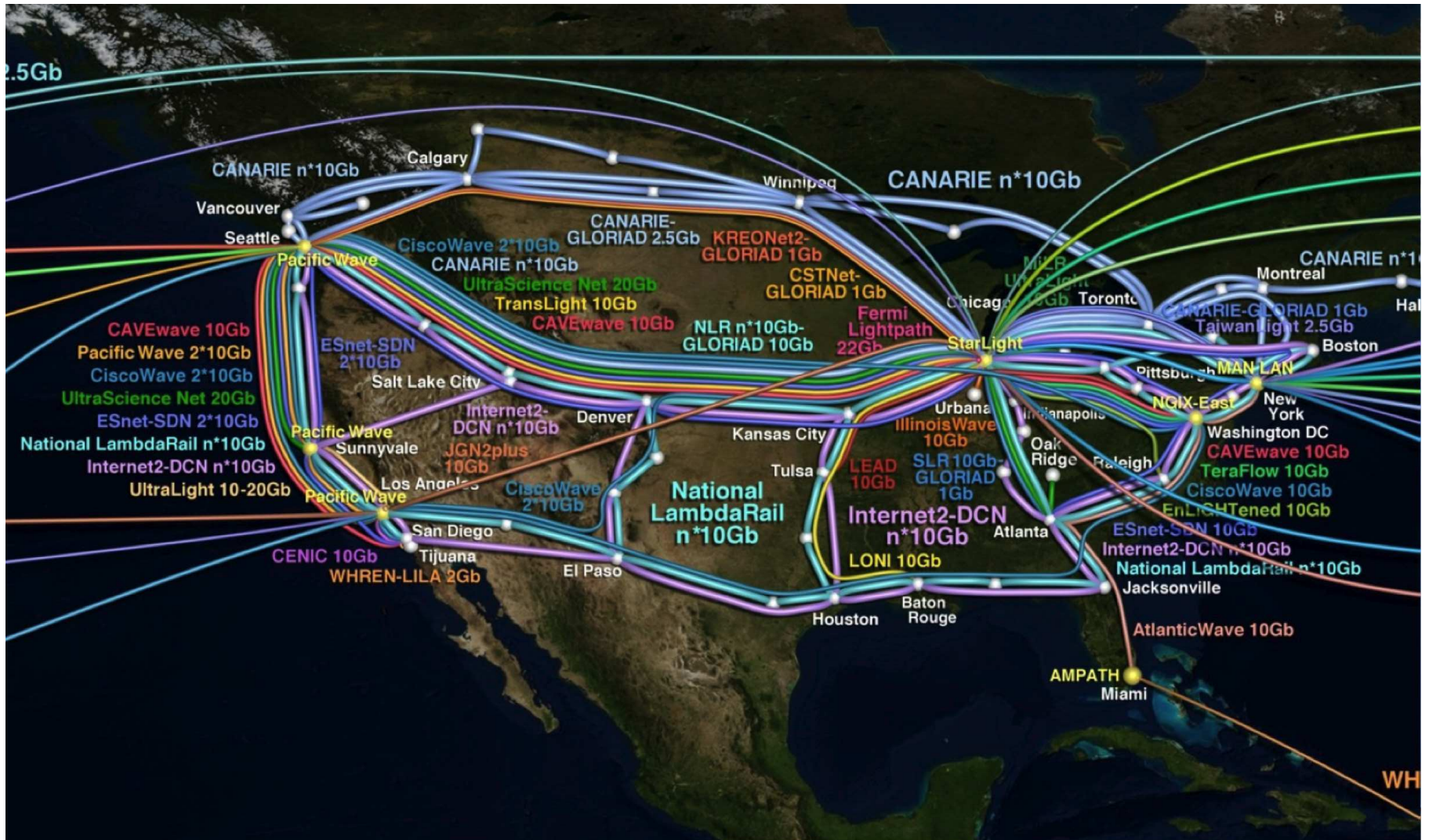
Ocean Observatories Initiative

- 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.



Ocean Observatories Initiative





Ocean Observatories Initiative

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

channels

- C
 - ioifaces
 - EndSocket.java
 - Smtplib_C_1_Future.java
 - Smtplib_C_1.java
 - Smtplib_C_10.java
 - Smtplib_C_11_Cases.java
 - Smtplib_C_11_Handler.java
 - Smtplib_C_11.java
 - Smtplib_C_12.java

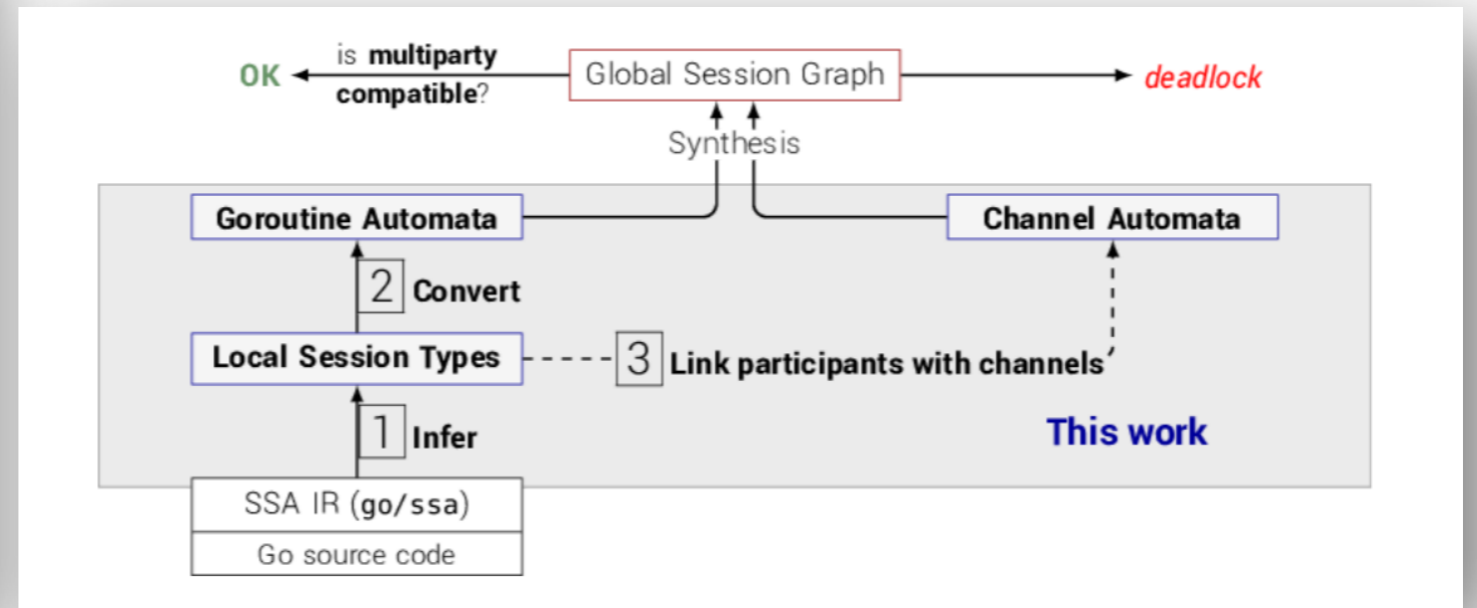
```

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

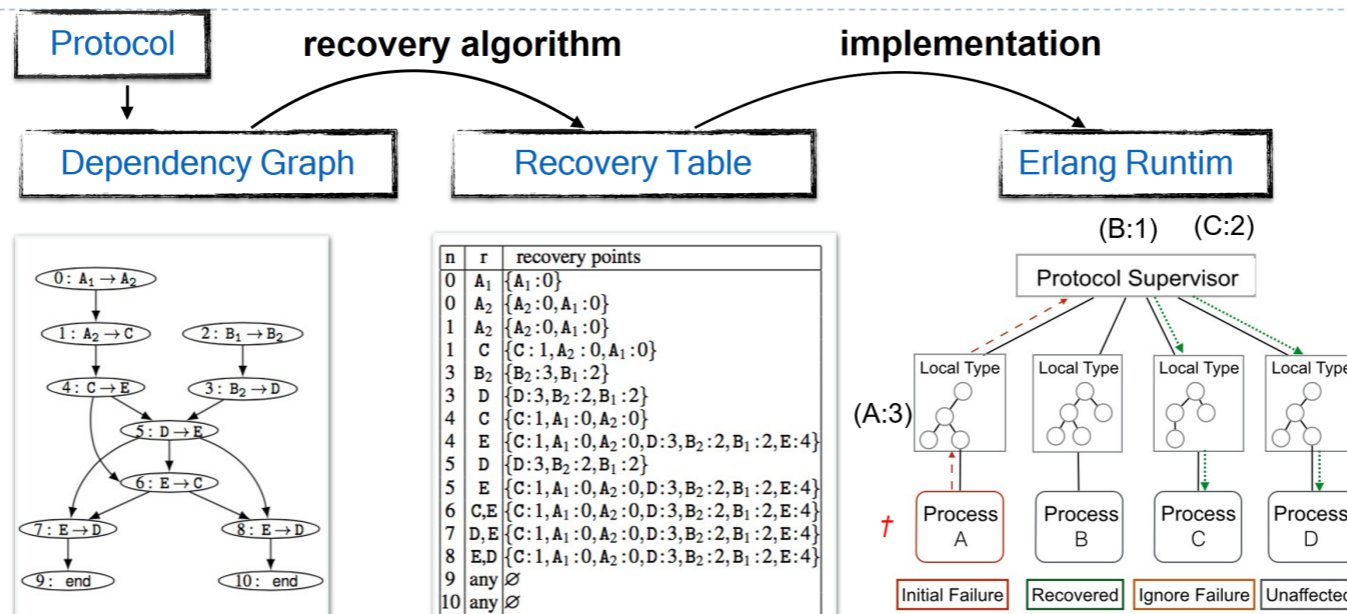
```

- send(S role, Mail m) : Smtplib_C_11 - Smtplib_C_10
- send(S role, Quit m) : EndSocket - Smtplib_C_10

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



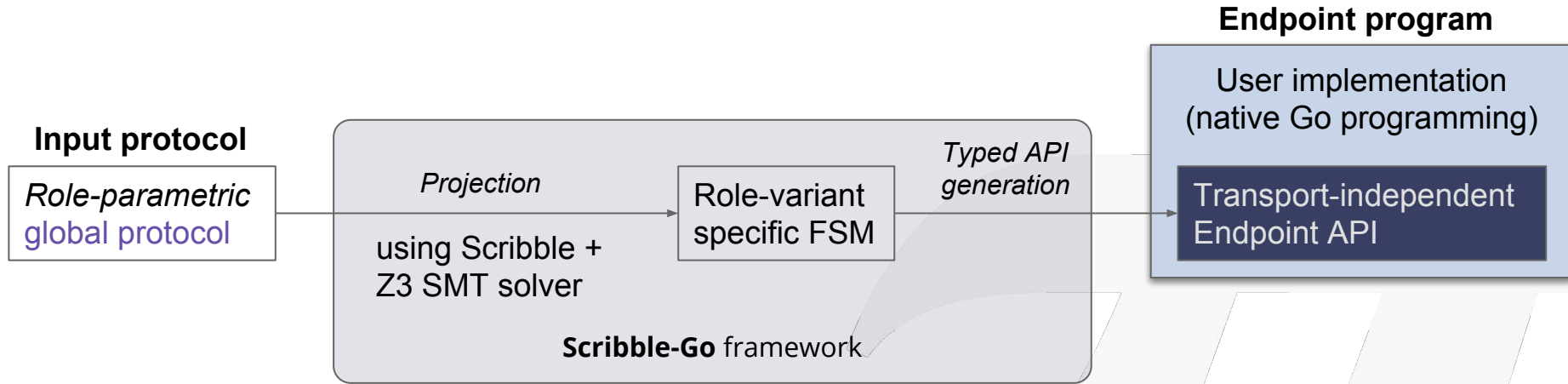
Safe Recovery for Erlang [CC'15]



Distributed Programming using Role-Parametric Session Types in Go

David Castro, Raymond Hu, Sung-Shik Jongmans,
Nicholas Ng, Nobuko Yoshida

Scribble-Go workflow



1. Write a *role-parametric global* protocol
2. Select endpoint *role variant* to implement (e.g. Fetcher)
3. Use **Scribble-Go** to project and generate **Endpoint API**
4. Implement endpoint (e.g. Fetcher [3]) using the **Endpoint API**

Role variant

Role variant are *unique kinds* of endpoints

$\{M, F[1..n], \text{Server}\}$

If $F[1]$ sends an extra request

HTTP HEAD to Server to get total size

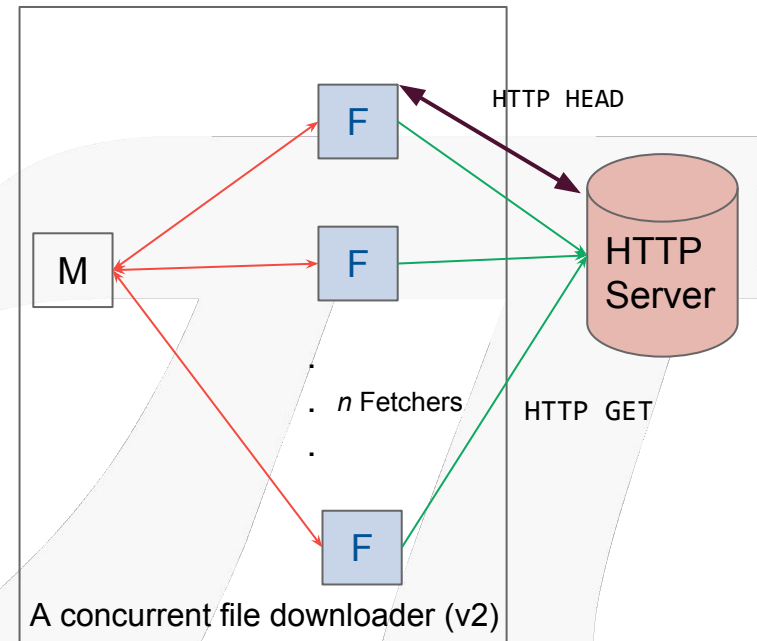
Then acts as a normal F

The role variants are:

$\{M, F[1], F[2..n], \text{Server}\}$

→ $F[1]$ and $F[2..n]$ are different endpoints

Inference of role variants (indices): formulated as SMT constraints for Z3



Endpoint API generation and usage

FSMs from local protocols → Message passing API

- Fluent-style
 - Every state is a unique type (struct)
 - Method calls (communication) returns next state
- Type information can be leveraged by IDEs
 - “dot-driven” content assist & auto complete

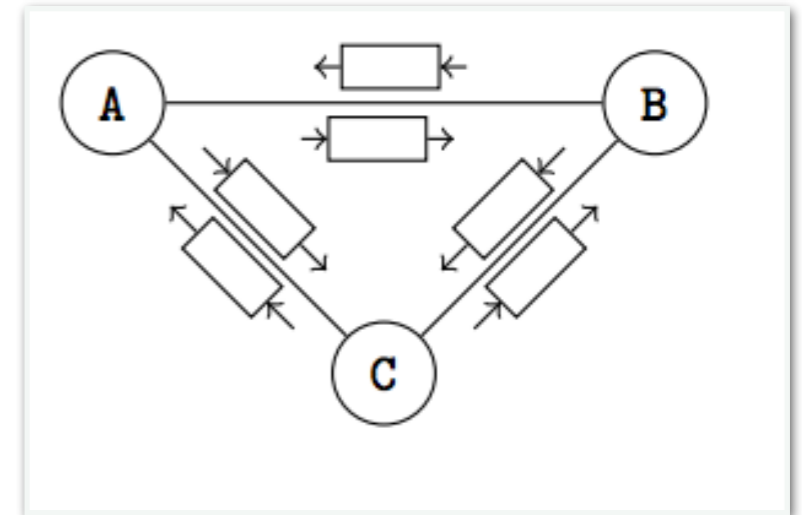
```
func doM2(m2 *M_2) M_End {  
    if m3 := m2.  
        □ Err: error  
        □ F_1_to_k:t1  
  
    func doM2(m2 *M_2) M_End {  
        if m3 := m2.F_1_to_k.|  
            □ Scatter:t2  
  
        func doM2(m2 *M_2) M_End {  
            if m3 := m2.F_1_to_k.Scatter.  
                ● Job(a []Job) *M_3  
  
                :  
  
        func doM2(m2 *M_2, meta *Meta) M_End {  
            if m3 := m2.F_1_to_k.Scatter.Job(split(meta)); m3.Err != nil {  
            } else {  
                return m3.F_1_to_k.Gather.  
            }  
            ● Data(a []Data) M_End
```

Session Types



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)
send(A, Div, int) | recv(A, Add, string)
send(B, Div, int) | recv(A, Div, int)
```

❌ Wrong **label**
❌ Wrong **payload**
❌ Wrong **role**

☑ Orphan messages

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

☑ Deadlock

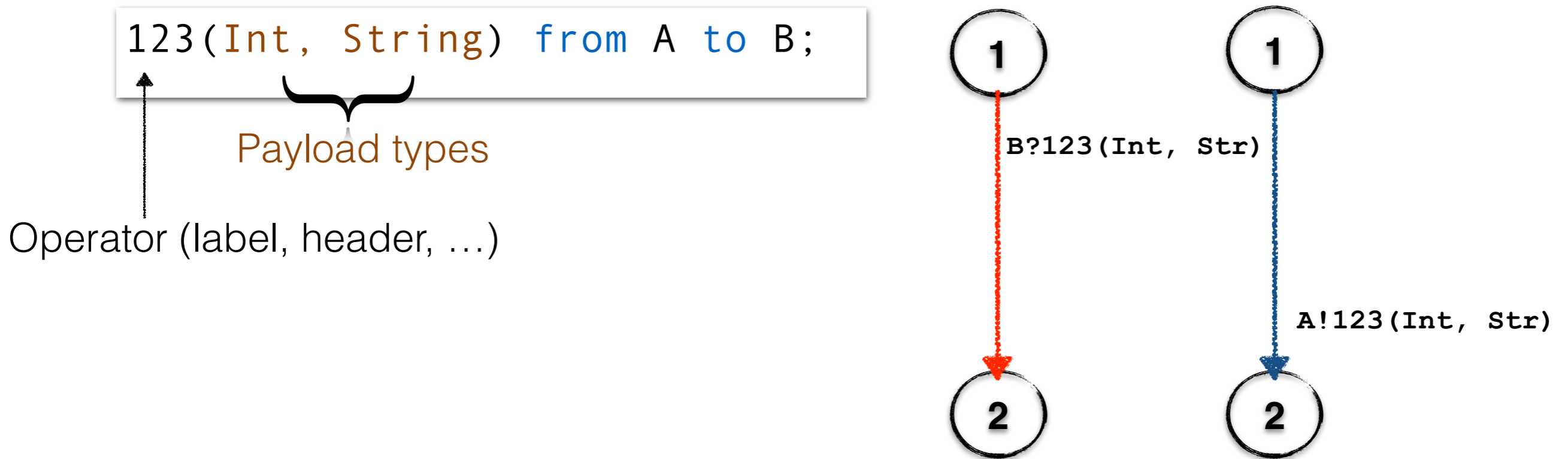
```
recv(A) | recv(B)
```

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



Scribble constructs:

Role-to-role Message passing



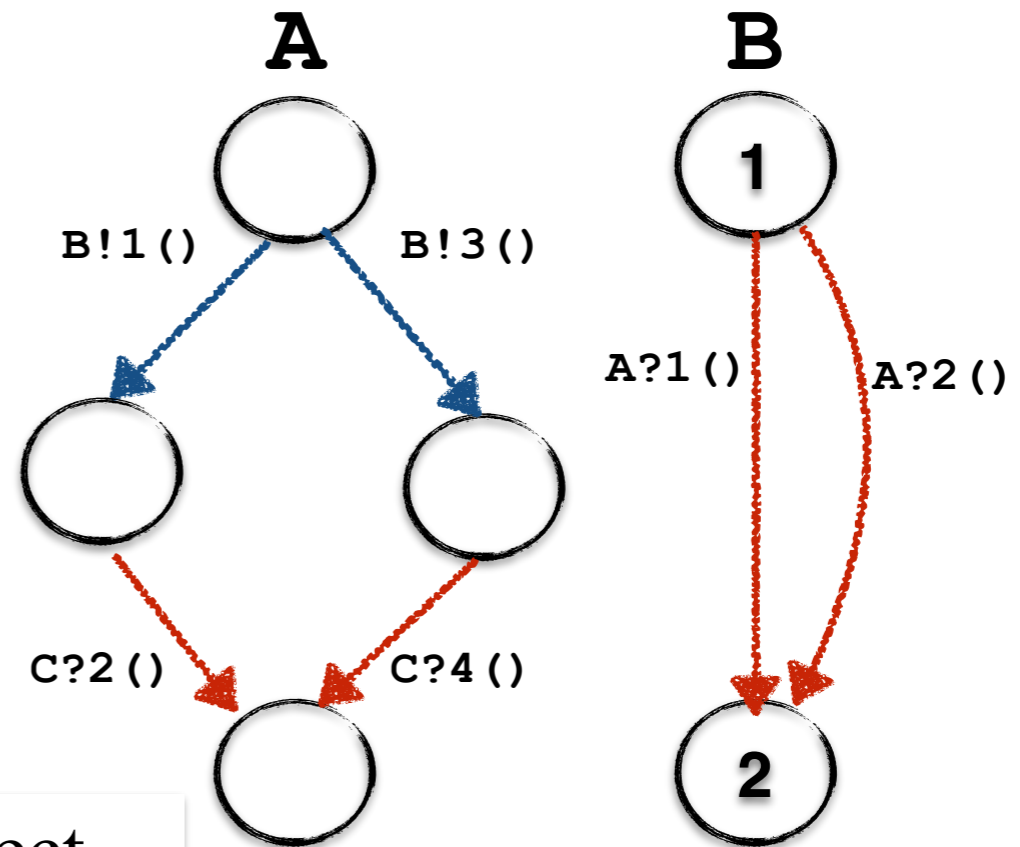
- Empty operator and/or payload is allowed

```
( ) 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

$p \rightarrow q : \{l_i : G_i\}_{i \in I}$ 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 ?

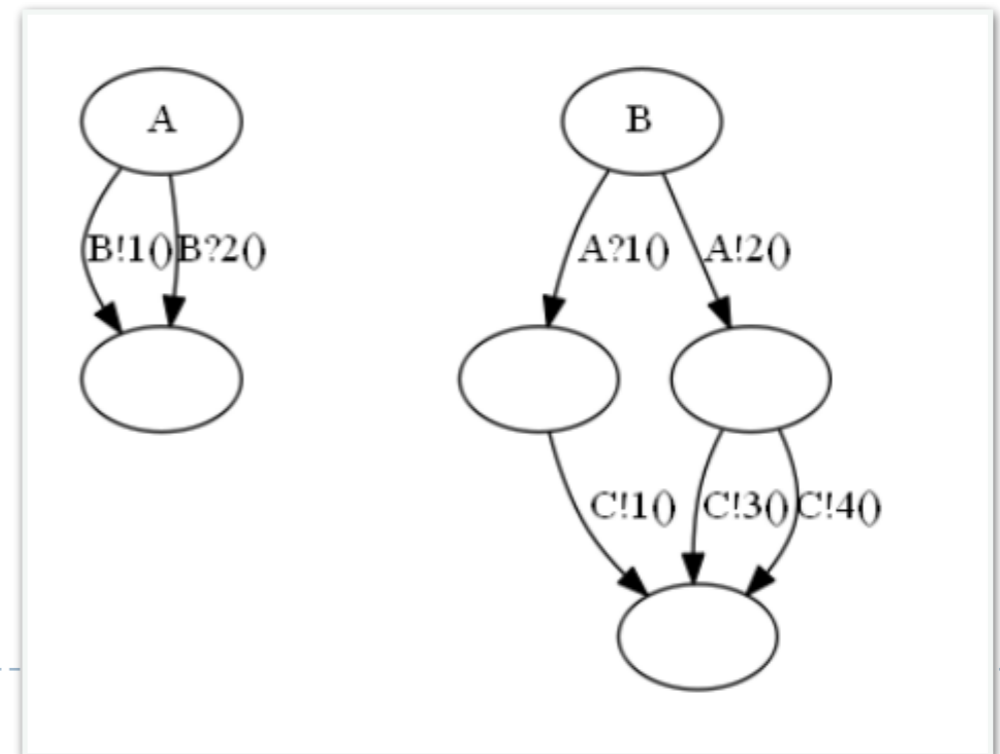
- MPST Safety errors:
 - reception error, orphan message, deadlock

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;  
}
```



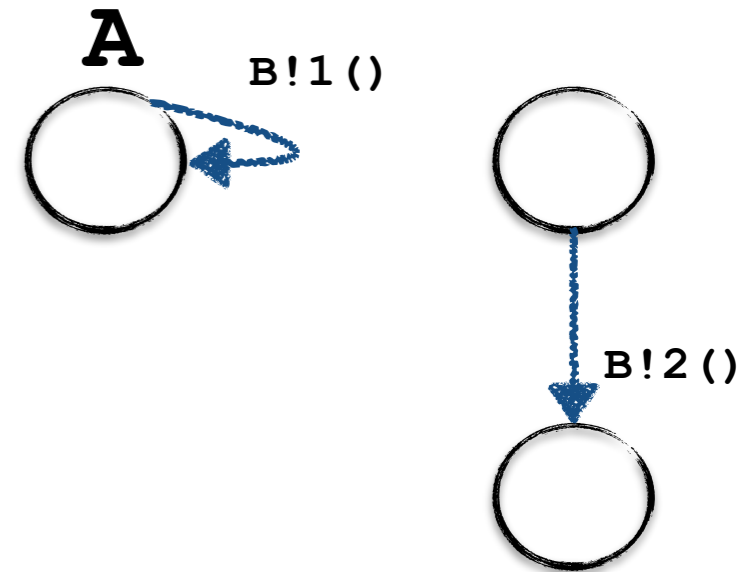
```
choice at A {  
  1() from A to C;  
  2() from B to C;  
} or {  
  3() from A to B;  
  4() from B to C;  
}
```



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

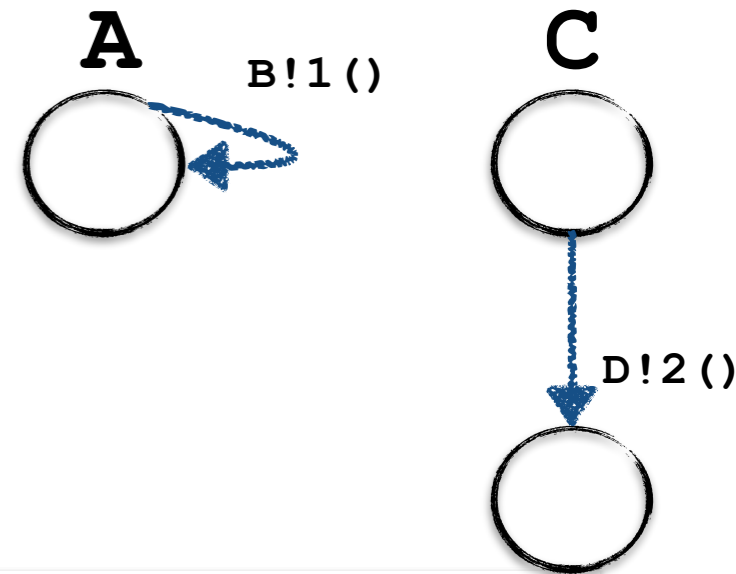
- Reachability of protocol states (no “dead code”)
 - Checked via projection (reachability w.r.t per-role protocol flow)
- Regular interaction structure at endpoints (CFSM)

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

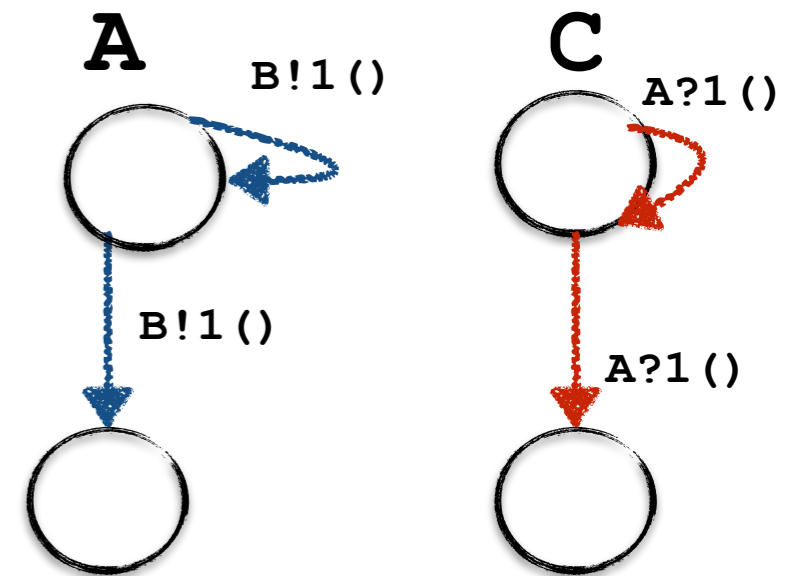
```
rec X {  
  choice at A {  
    1() from A to B;  
    2() from B to C;  
    3() from C to B;  
  } or {  
    4() from A to C;  
    5() from C to B;  
  }  
  continue X;  
}
```

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

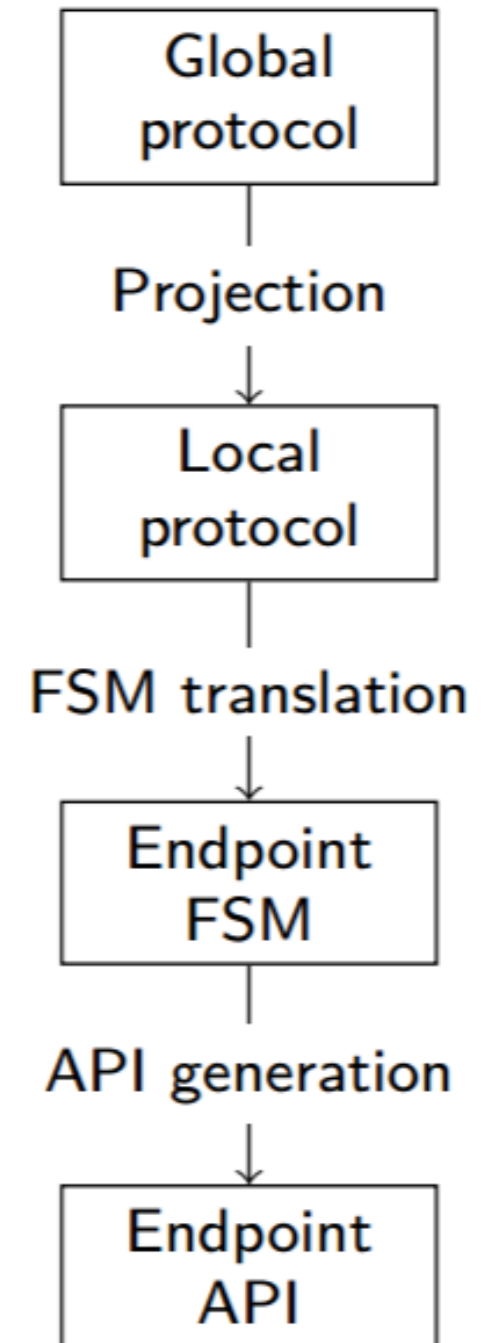




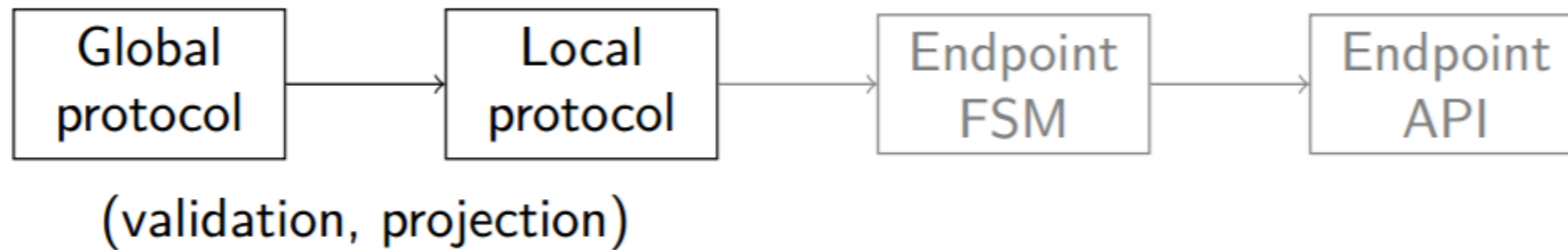
Program Verification

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



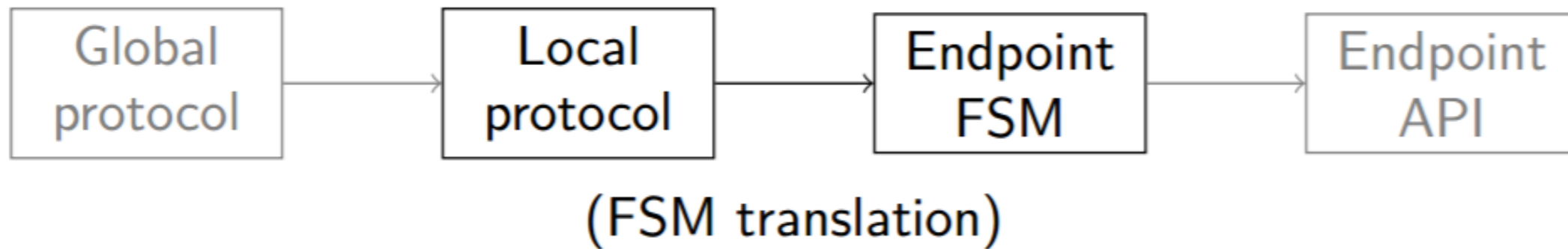
Example: Adder



```
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;
  }
}
```

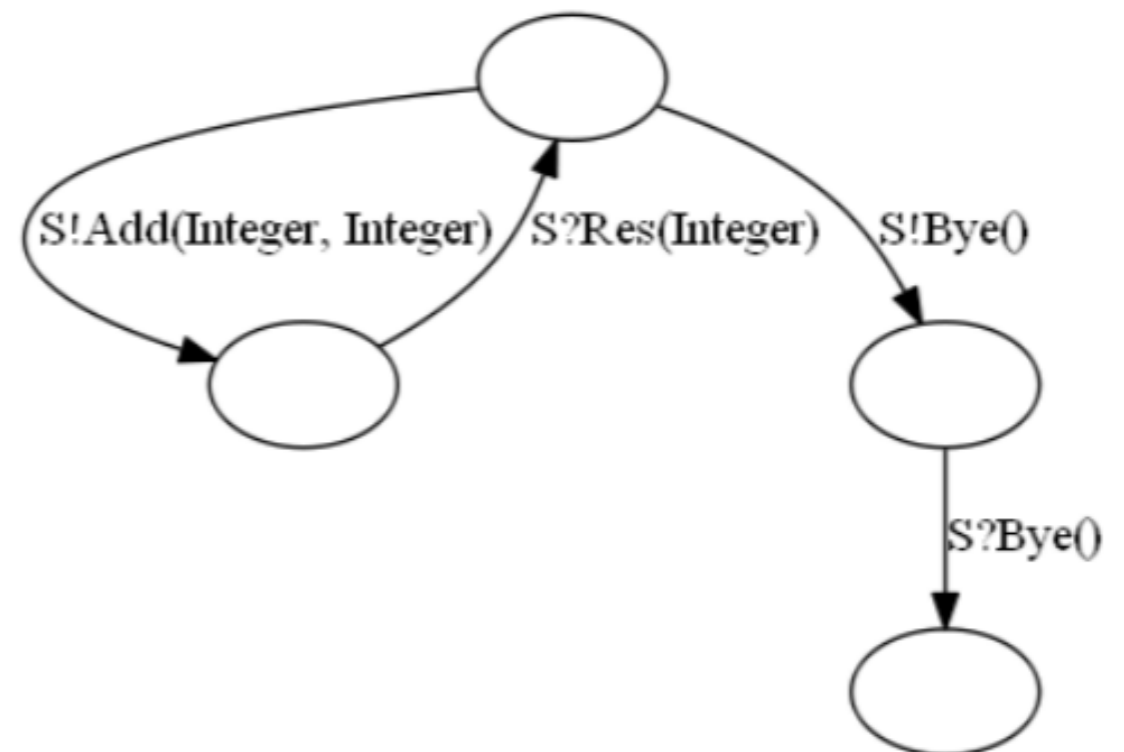


Example: Adder

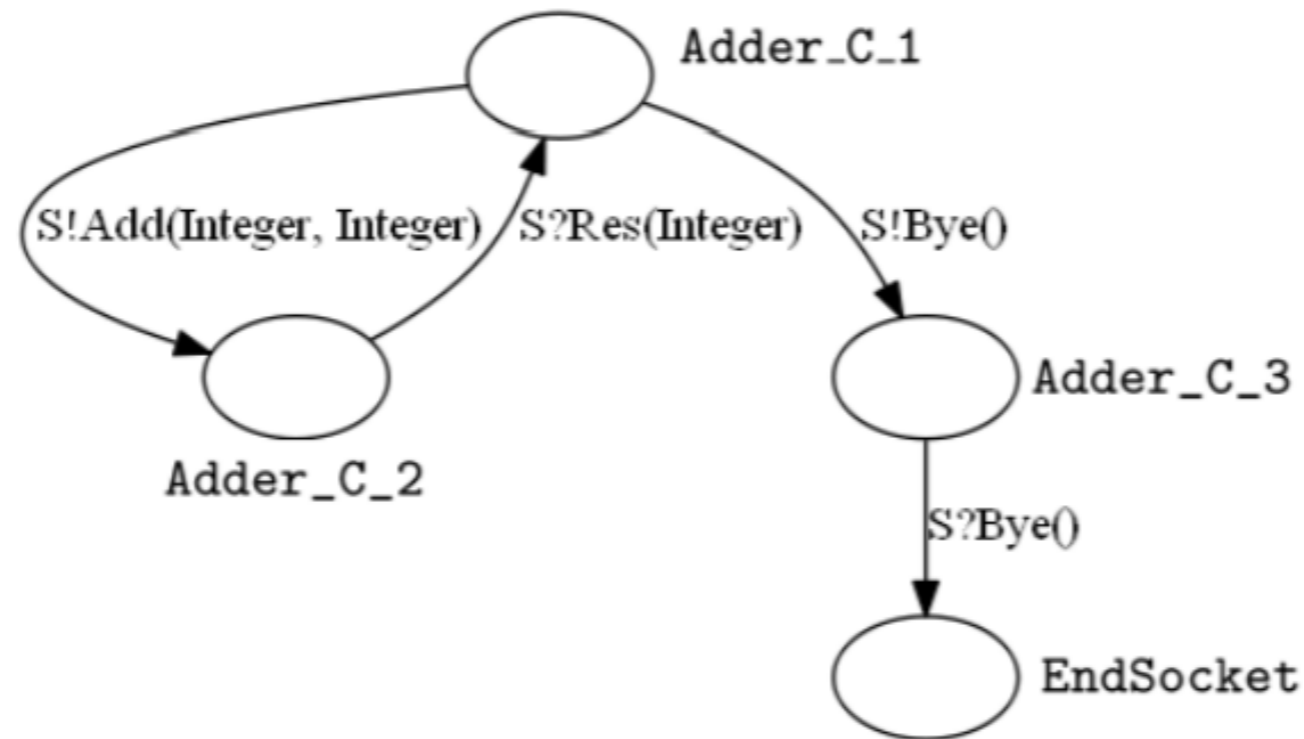


► Projected Endpoint FSM (EFSM) for C

```
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;  
  }  
}
```



Adder: State Channel API for C



- ▶ Adder_C_1

- ▶ Output state channel: (overloaded) send methods

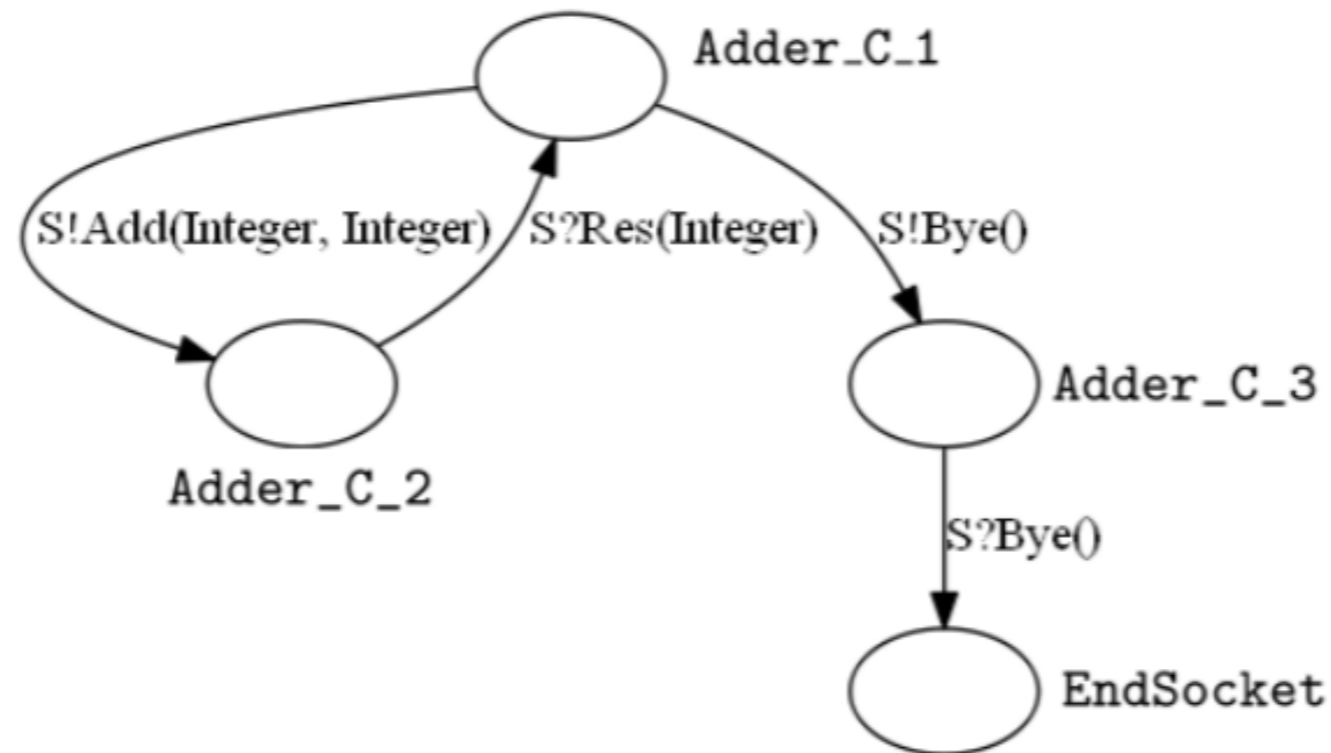
Adder_C_2 **send**(*S* role, *Add* op, Integer arg0, Integer arg1) **throws** ...

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