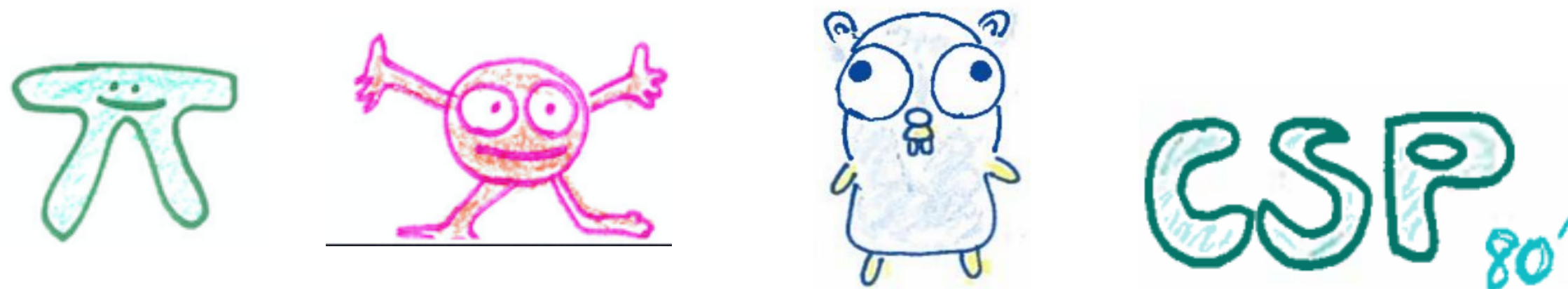


# Actor Programming and Verifications

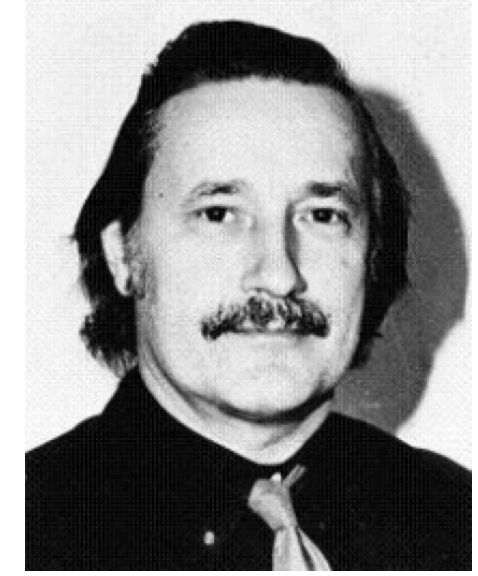
Specification-Guided Concurrent and Distributed Programming



Nobuko Yoshida, University of Oxford

AIMS Seminar, University of Oxford, 10th March 2023

# AI and Strachey



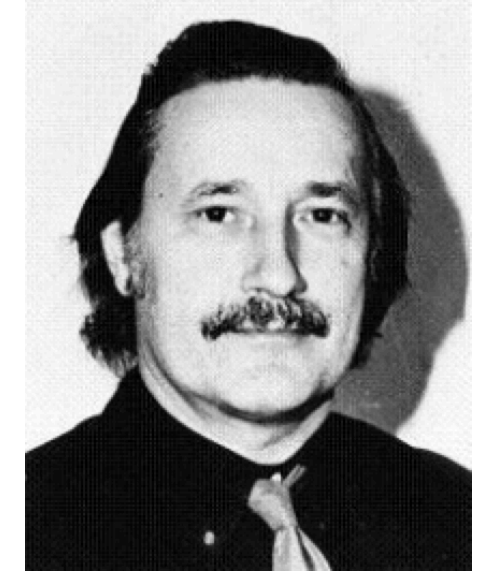
1916-1975

**Computer Games, Literature and Music**      A schoolmaster at Harrow

**Specification-Guided Concurrent and Distributed Programming**

**Fundamental Concepts of  
Programming Languages**

# AI and Strachey



1916-1975

**Computer Games, Literature and Music**

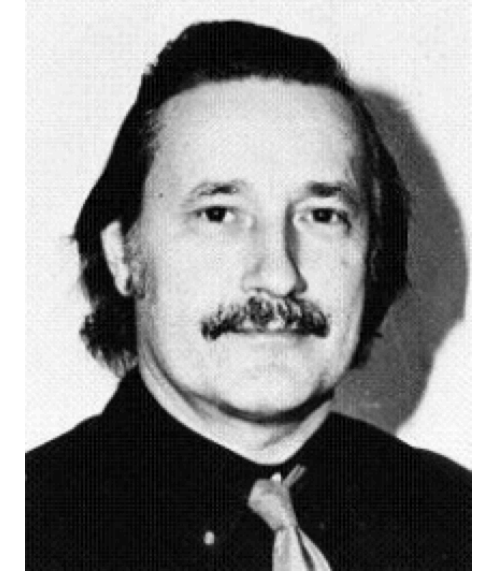
A schoolmaster at Harrow

**Specification-Guided Concurrent and Distributed Programming**

- **Christopher Strachey** (sequential computation)

**Fundamental Concepts of  
Programming Languages**

# AI and Strachey



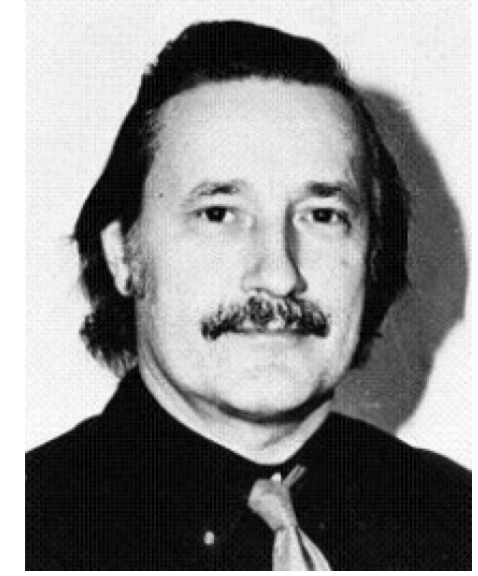
1916-1975

**Computer Games, Literature and Music**      A schoolmaster at Harrow

**Specification-Guided Concurrent and Distributed Programming**

- **Christopher Strachey** (sequential computation)      **Fundamental Concepts of Programming Languages**
  - ▶ **Types** = abstract and digest computation (data types, polymorphism)

# AI and Strachey



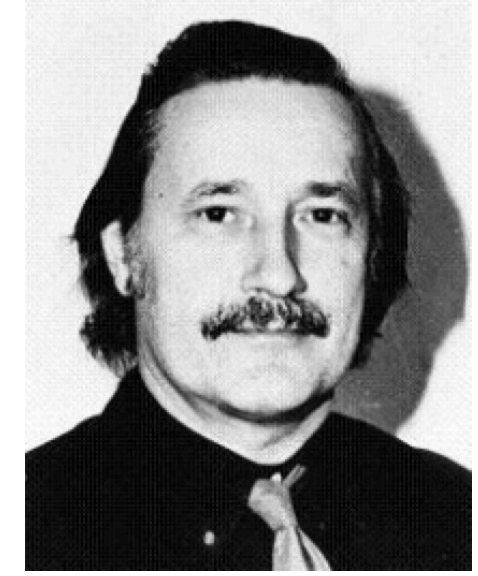
1916-1975

**Computer Games, Literature and Music**      A schoolmaster at Harrow

## Specification-Guided Concurrent and Distributed Programming

- **Christopher Strachey** (sequential computation)      **Fundamental Concepts of Programming Languages**
  - *Types* = abstract and digest computation (data types, polymorphism)
  - **Structured programming = High-level programming**

# AI and Strachey



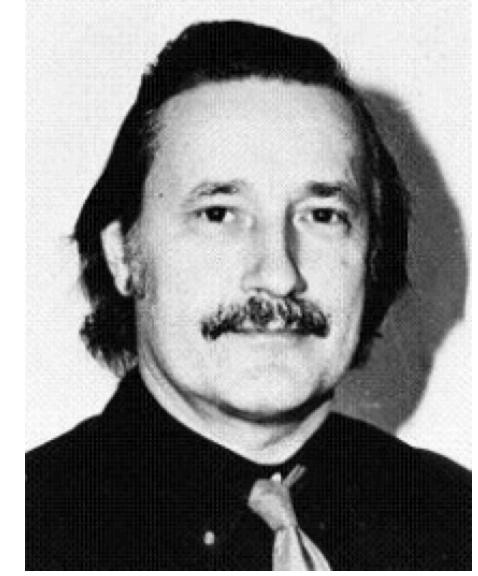
1916-1975

**Computer Games, Literature and Music**      A schoolmaster at Harrow

## Specification-Guided Concurrent and Distributed Programming

- **Christopher Strachey** (sequential computation)      **Fundamental Concepts of Programming Languages**
  - *Types* = abstract and digest computation (data types, polymorphism)
  - Structured programming = High-level programming
- **Session types** (concurrency & communication)

# AI and Strachey



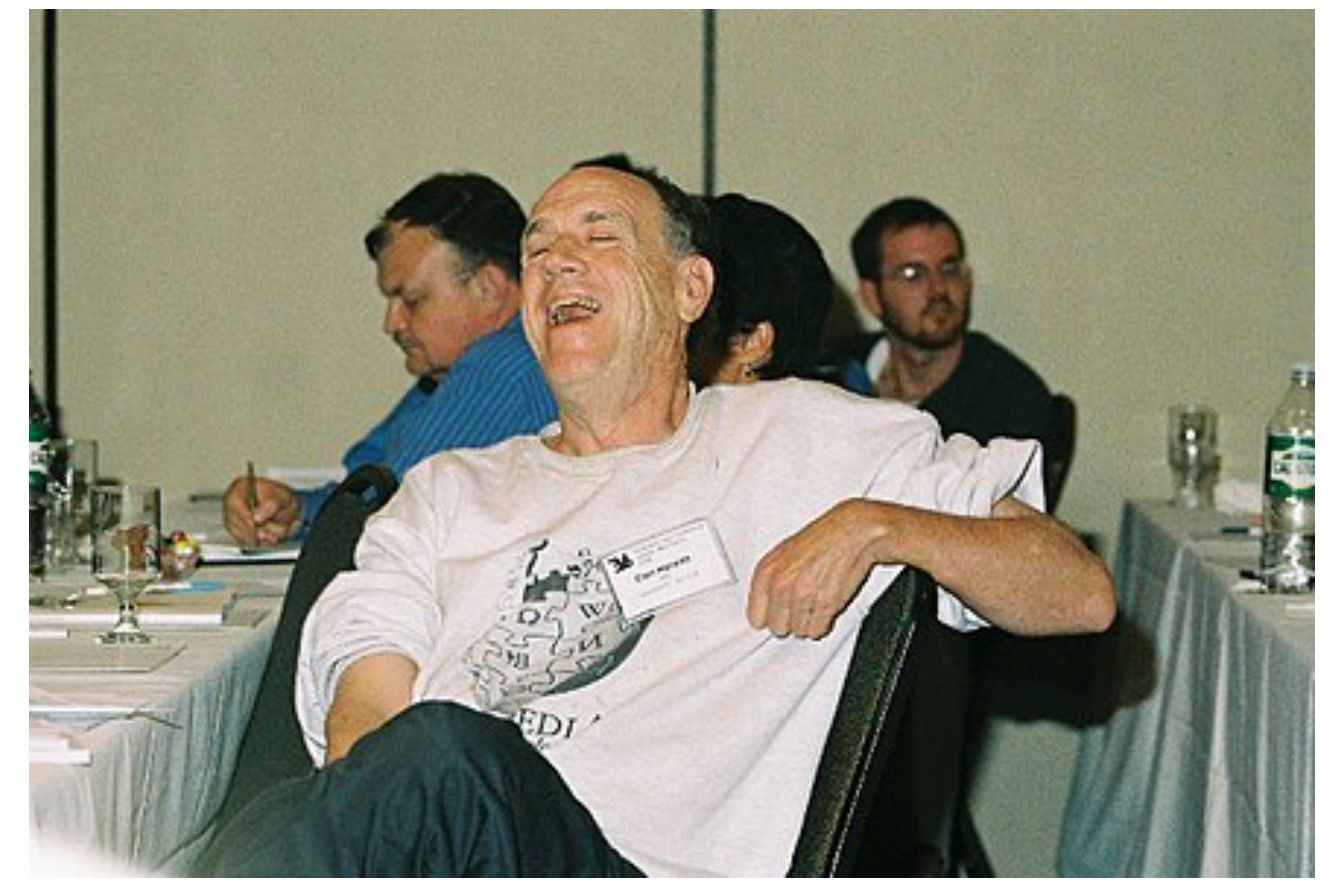
1916-1975

**Computer Games, Literature and Music**      A schoolmaster at Harrow

## Specification-Guided Concurrent and Distributed Programming

- **Christopher Strachey** (sequential computation)      **Fundamental Concepts of Programming Languages**
  - ▶ **Types** = abstract and digest computation (data types, polymorphism)
  - ▶ Structured programming = High-level programming
- **Session types** (concurrency & communication)
  - ▶ Structured programming = **protocols**

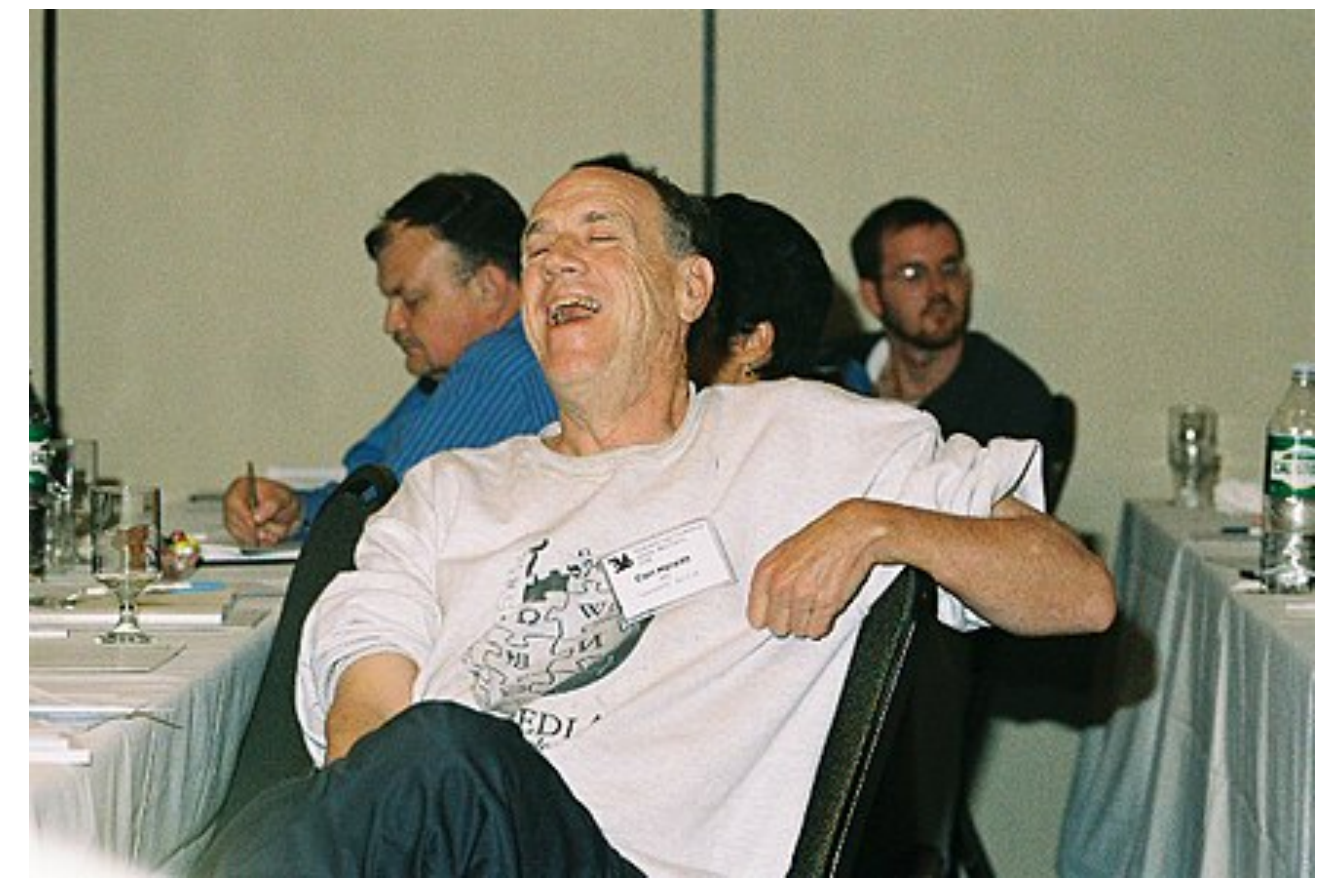
# Actor Models



**1944-2020**

# Actor Models

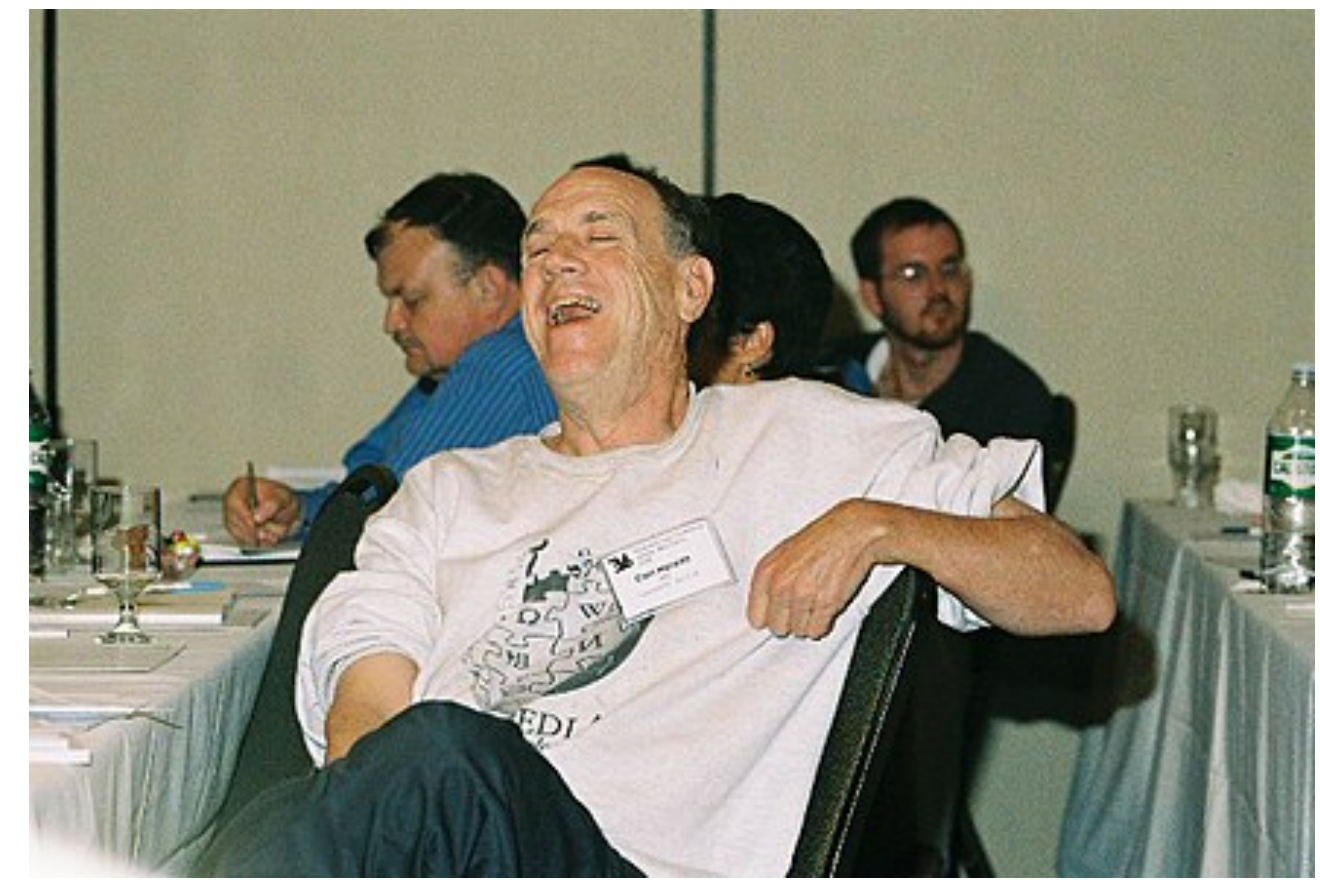
- **Carl Hewitt (MIT)**



**1944-2020**

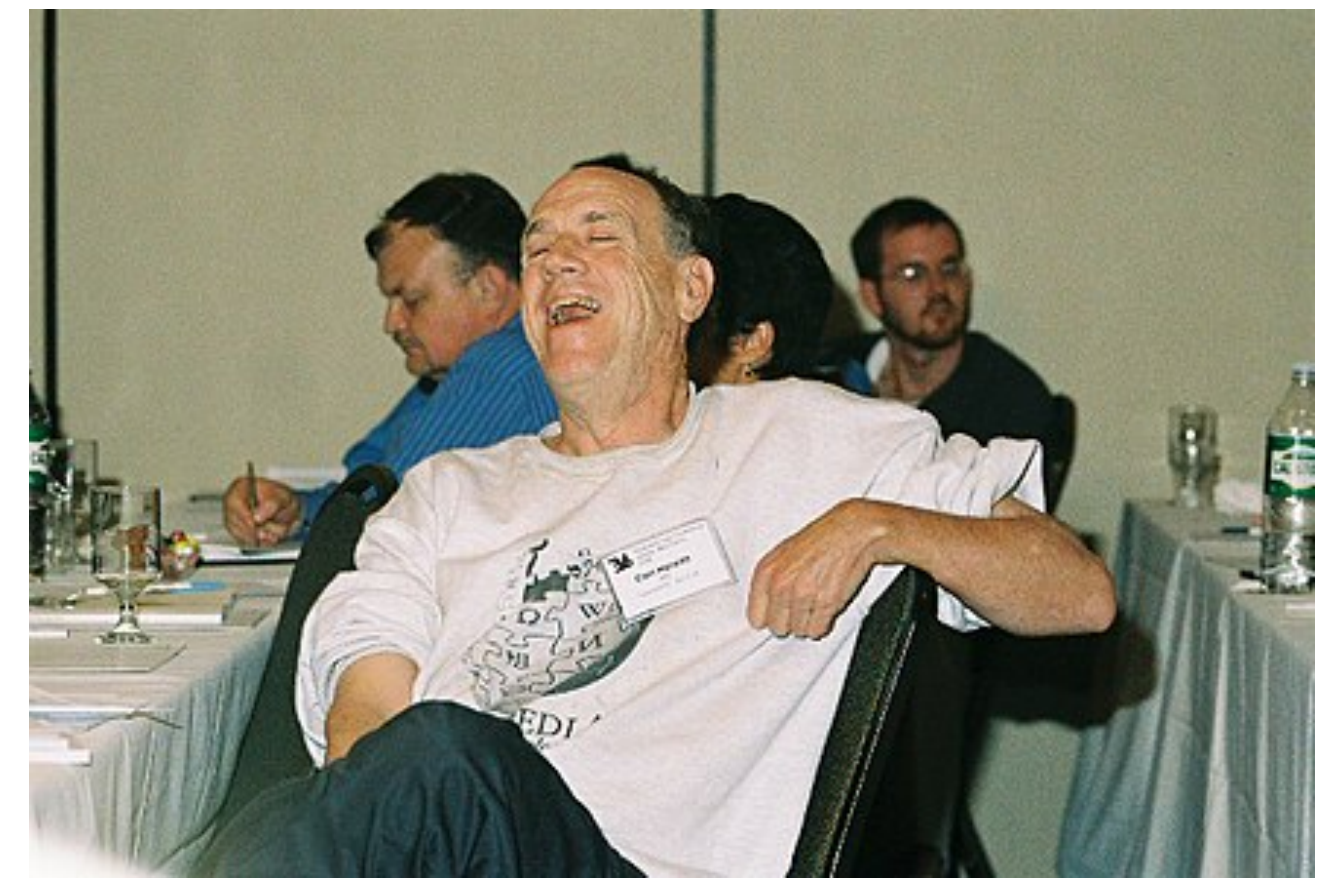
# Actor Models

- Carl Hewitt (MIT)
- **Mathematical Models** for Concurrent Computations



1944-2020

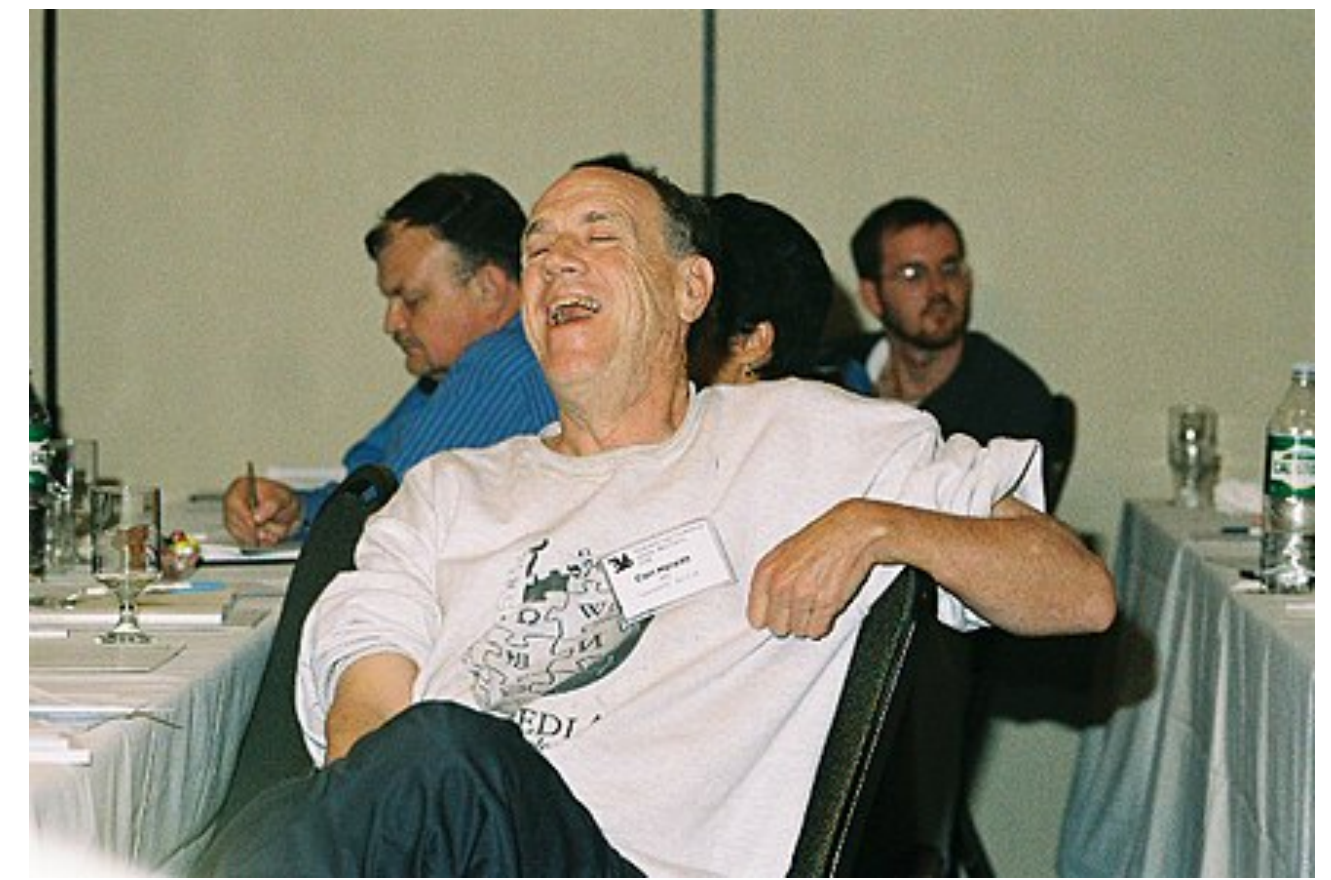
# Actor Models



1944-2020

- Carl Hewitt (MIT)
- Mathematical Models for Concurrent Computations
- **Asynchronous Message Passings among Concurrent and Distributed Independent Agents**

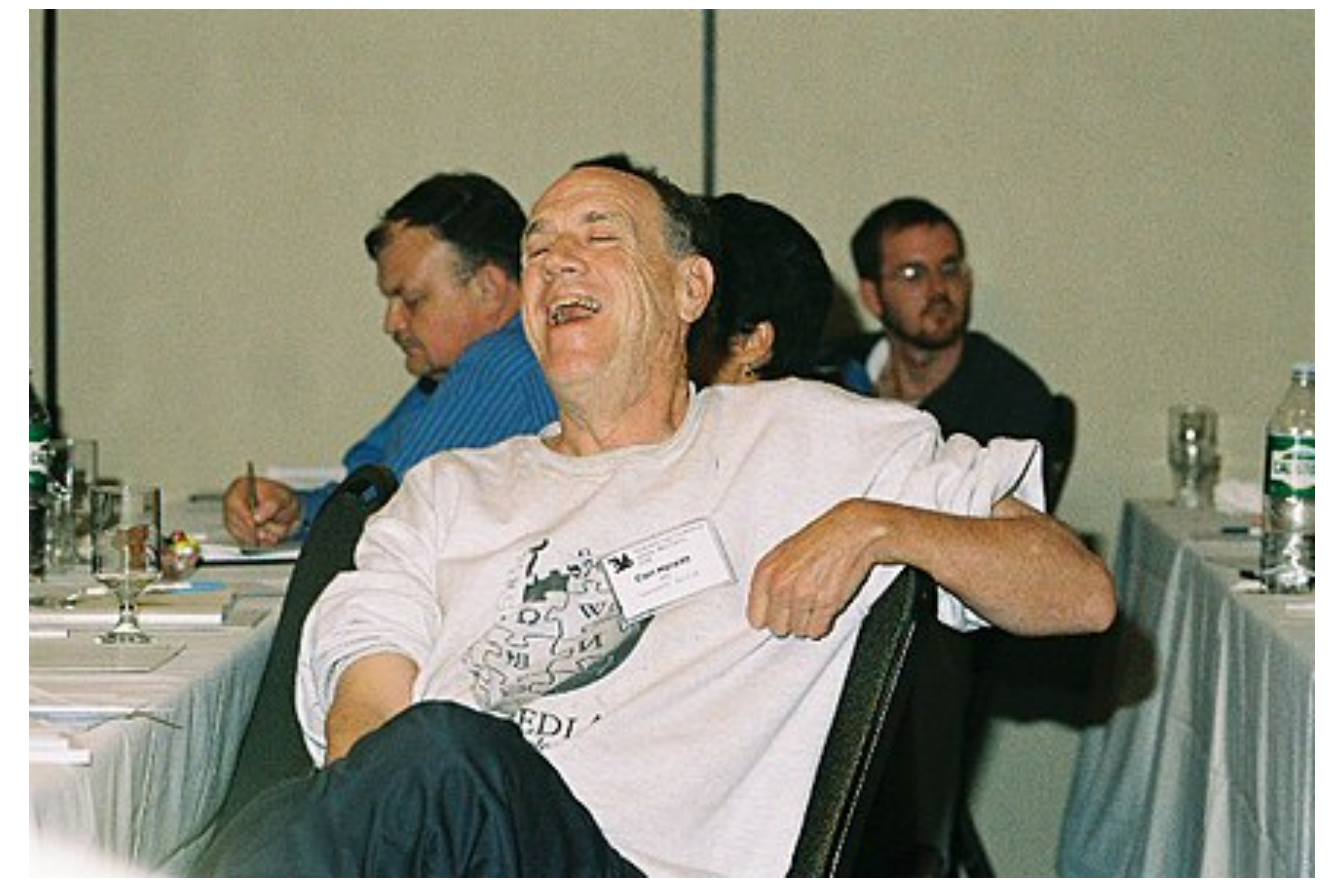
# Actor Models



1944-2020

- Carl Hewitt (MIT)
- Mathematical Models for Concurrent Computations
- Asynchronous Message Passings among Concurrent and Distributed Independent Agents
- Concurrent Object Oriented Languages

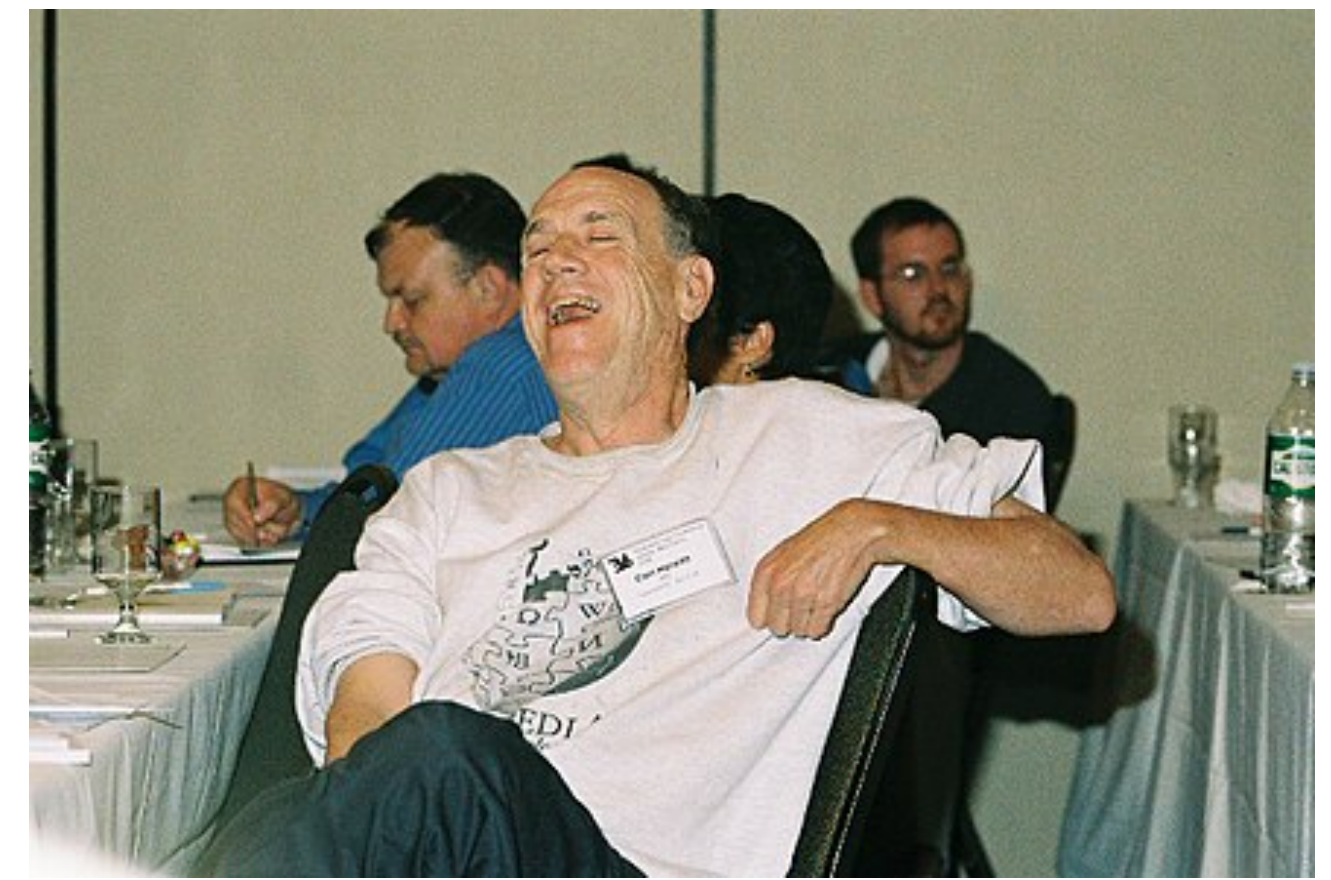
# Actor Models



1944-2020

- Carl Hewitt (MIT)
- Mathematical Models for Concurrent Computations
- Asynchronous Message Passings among Concurrent and Distributed Independent Agents
  - Concurrent Object Oriented Languages
  - Multi Agent Systems

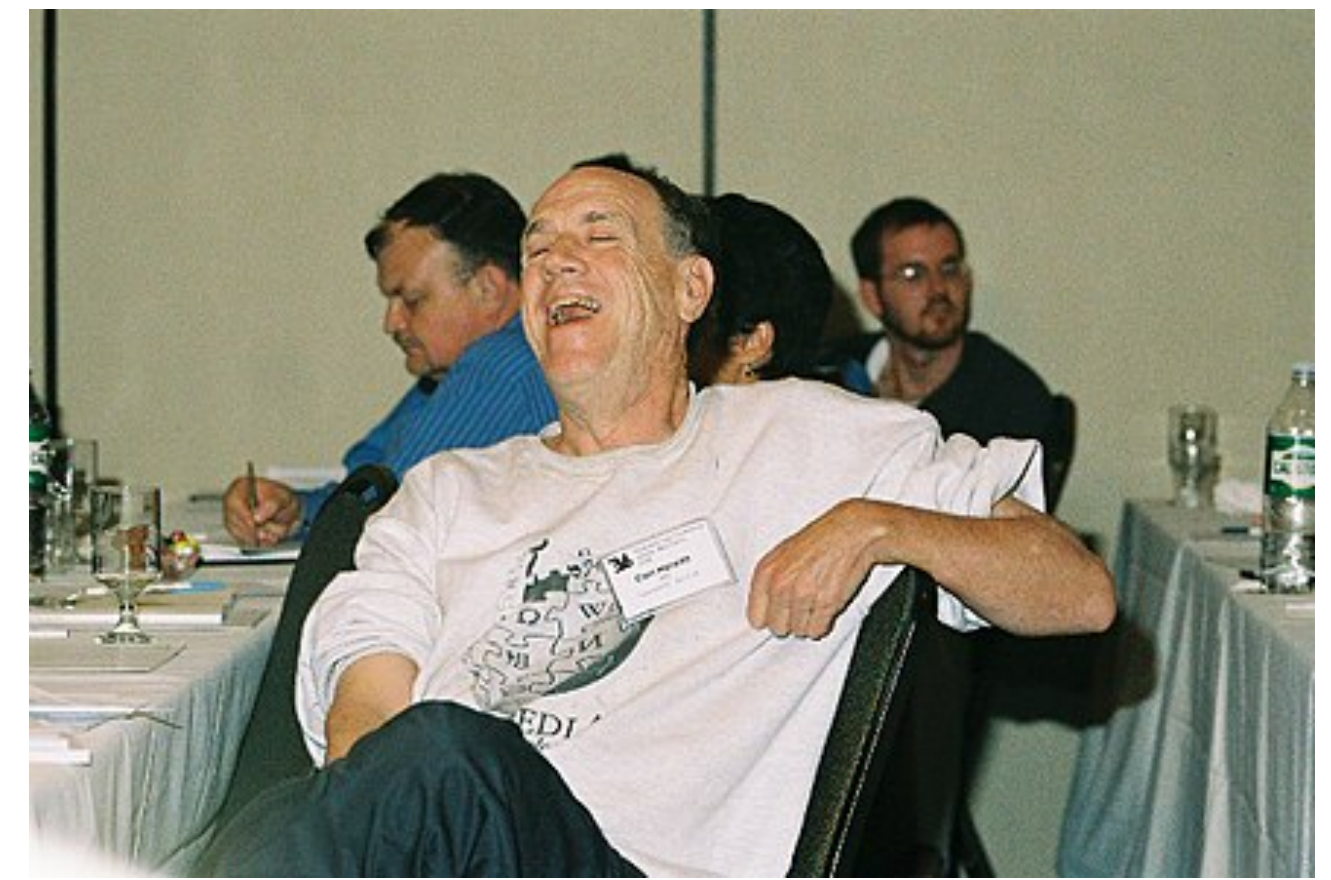
# Actor Models



1944-2020

- Carl Hewitt (MIT)
- Mathematical Models for Concurrent Computations
- Asynchronous Message Passings among Concurrent and Distributed Independent Agents
  - Concurrent Object Oriented Languages
  - Multi Agent Systems
  - Web Services and Simple Object Access **Protocols** (SOAP)

# Actor Models



1944-2020

- Carl Hewitt (MIT)
- Mathematical Models for Concurrent Computations
- Asynchronous Message Passings among Concurrent and Distributed Independent Agents
  - Concurrent Object Oriented Languages
  - Multi Agent Systems
  - Web Services and Simple Object Access **Protocols** (SOAP)
  - Mathematical Models (Logics and Process Algebra)

# Communications are Ubiquitous

- Increasingly, **communications** are the way to organise software and systems.
- Industry trend – programming languages with **explicit message-passing primitives**.



microservices



# Problems: Ambiguity

- Protocol descriptions are **ambiguous**
- **SMTP: simple mail transfer protocol**
  - They are written in English, often very long



RFC 821

August 1982  
Simple Mail Transfer Protocol

## TABLE OF CONTENTS

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

# Problems: Ambiguity

- Protocol descriptions are **ambiguous**
- **SMTP: simple mail transfer protocol**
  - They are written in English, often very long



## 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 [golang.org]
  - Analysis of 6 large software systems [ASPLOS 19]



GO

Google (2009)



The Go Gopher

CSP<sub>80'</sub>

*Do not communicate by sharing memory;  
share memory by communicating*

*– Go Philosophy*

# Problems: Concurrency Bugs

---

- Communications increase **concurrency bugs**
  - Survey of 4K users [golang.org]
  - Analysis of 6 large software systems [ASPLOS 19]

deadlock

channel errors

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



The Go Gopher

# Problems: Concurrency Bugs

- Communications increase **concurrency bugs**
  - Survey of 4k users [golang.org]
  - Analysis of 6 large software systems [ASPLOS 19]

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

## Session Types

- Prevent concurrency bugs.
- Can abstract, implement and manage communications as **Protocols**.
- **Clean, Cheap** and **Retrofittable**.



# Why Session Types, Why Now?

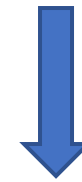
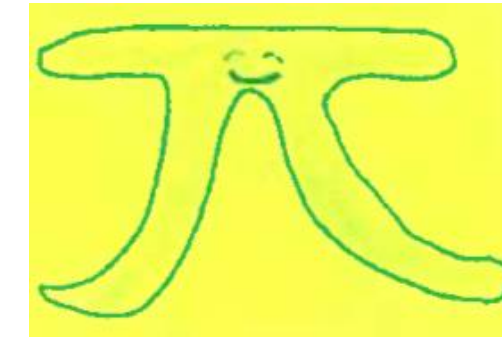
Significant academic and industry interests via fundamental breakthroughs

Milner,  
Honda, NY



Binary Session Types

ESOP'98

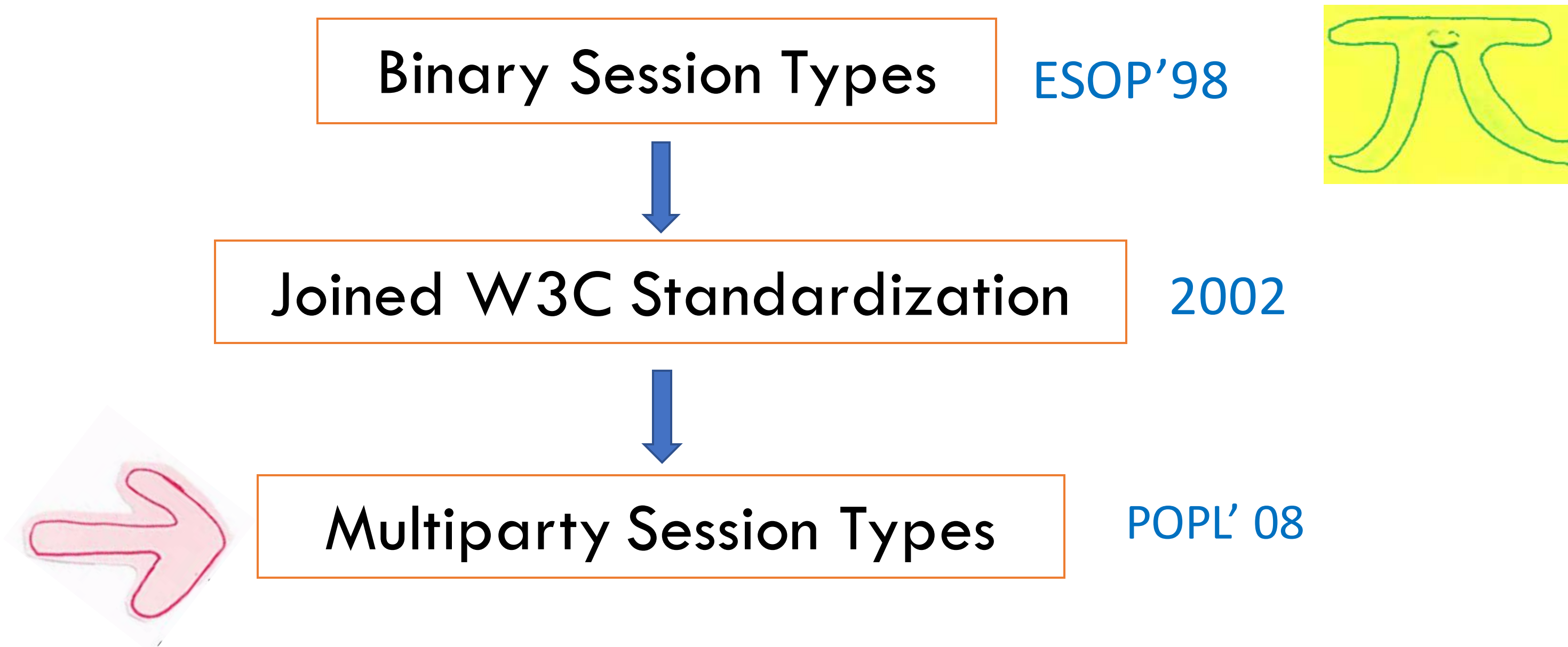


Joined W3C Standardization

2002

# Why Session Types, Why Now?

Significant academic and industry interests via fundamental breakthroughs



# Choreography Description Language (W3C)

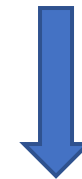
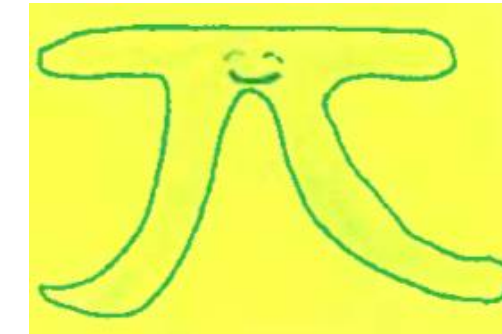
```
package HelloWorld {  
    roleType YouRole, WorldRole;  
    participantType You{YouRole}, World{WorldRole};  
    relationshipType YouWorldRel between YouRole and WorldRole;  
    channelType WorldChannelType with roleType WorldRole;  
  
    choreography Main {  
        WorldChannelType worldChannel;  
  
        interaction operation=hello from=YouRole to=WorldRole  
            relationship=YouWorldRel channel=worldChannel {  
            request messageType=Hello;  
        }  
    }  
}
```

# Why Session Types, Why Now?

Significant academic and industry interests via fundamental breakthroughs

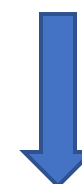
Binary Session Types

ESOP'98



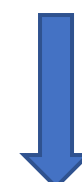
Joined W3C Standardization

2002



Multiparty Session Types

POPL' 08

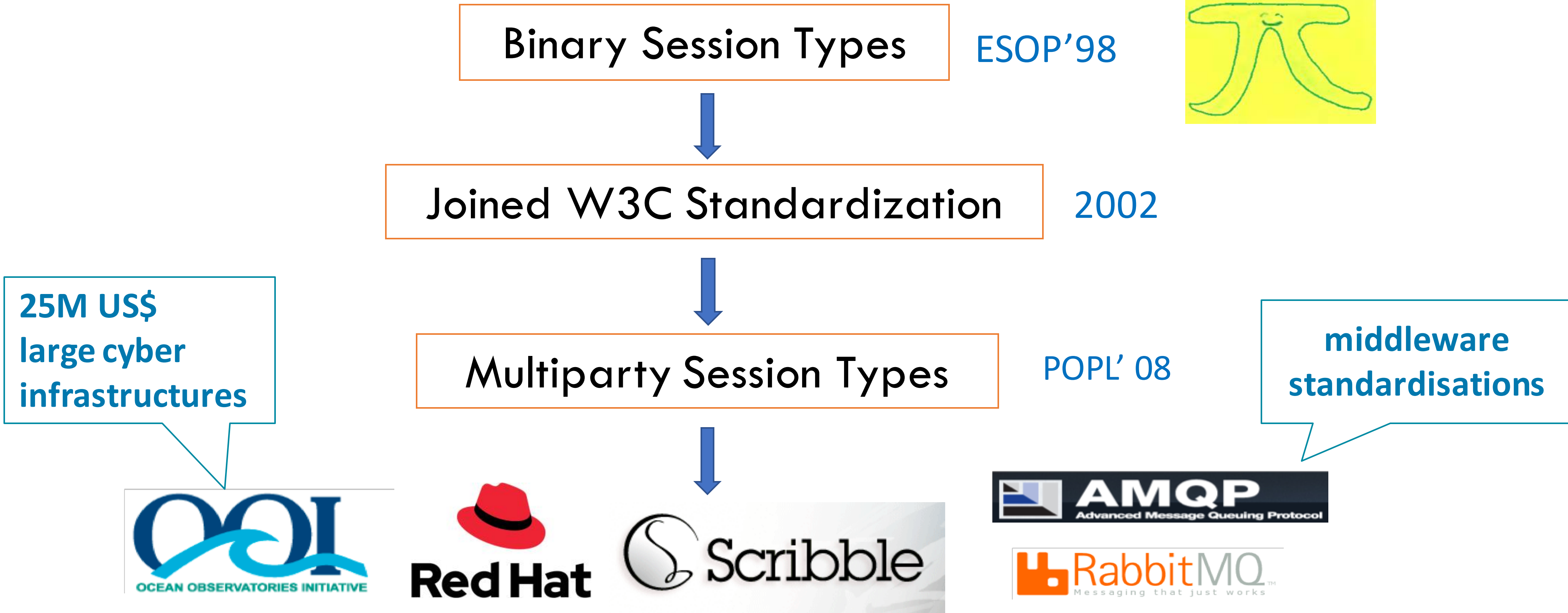


largest open source  
company in the world



# Why Session Types, Why Now?

Significant academic and industry interests via fundamental breakthroughs

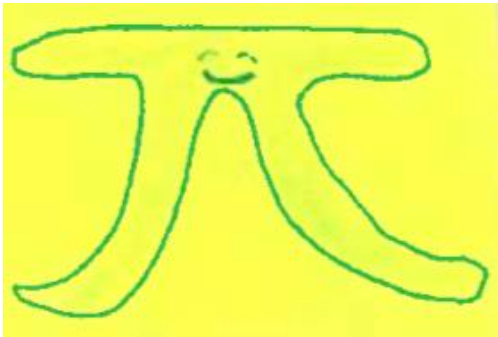


# Why Session Types, Why Now?

Significant academic and industry interests via fundamental breakthroughs

Binary Session Types

ESOP'98



Joined W3C Standardization

2002

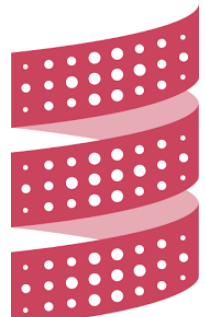


Multiparty Session Types

POPL'08



TypeScript



Scala

akka



ERLANG

MPI



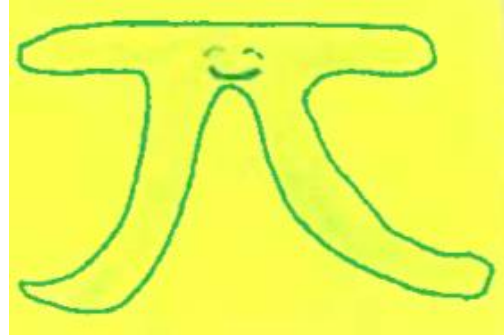
# Why Session Types, Why Now?

Significant academic and industry interests via fundamental breakthroughs

ETAPS Test Time Award 2019

Binary Session Types

ESOP'98



Joined W3C Standardization

2002



Multiparty Session Types

POPL' 08

POPL Influential Paper Award 2018



A collection of logos for various programming languages and frameworks, including Java, Go, OOI (Ocean Observatories Initiative), Red Hat, Scribble, AMQP (Advanced Message Queuing Protocol), RabbitMQ, Python, Scala, akka, Erlang, MPI, and OCaml.

# What is a concept of *Types* in Programming?

- Types can avoid runtime error

**Boolean, Natural Number**

**100 x 200**

**100 x true**

# What is a concept of *Types* in Programming?

- Types can avoid runtime error

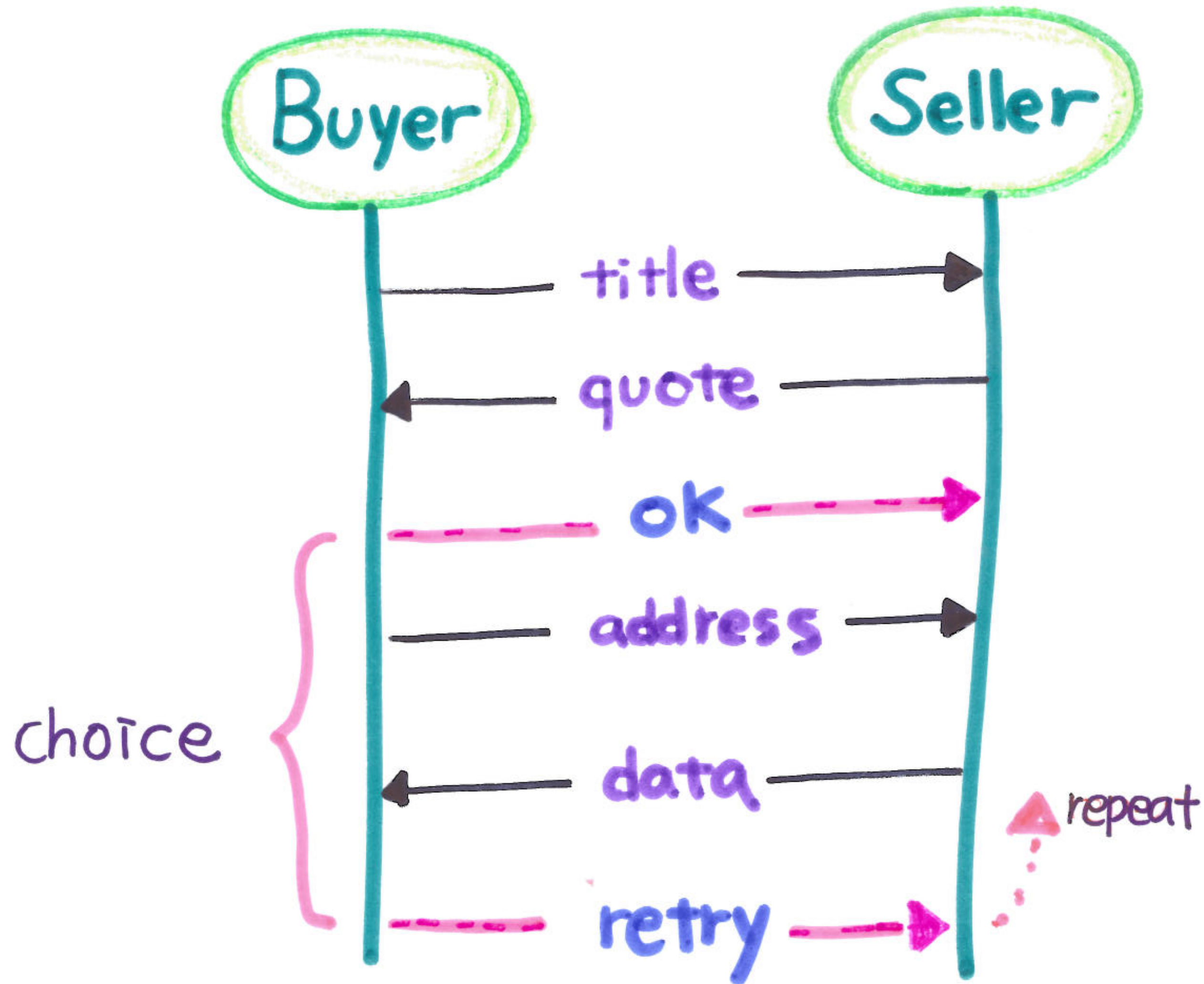
**Boolean, Natural Number (Data Types)**

**100 x 200**      **Correct**

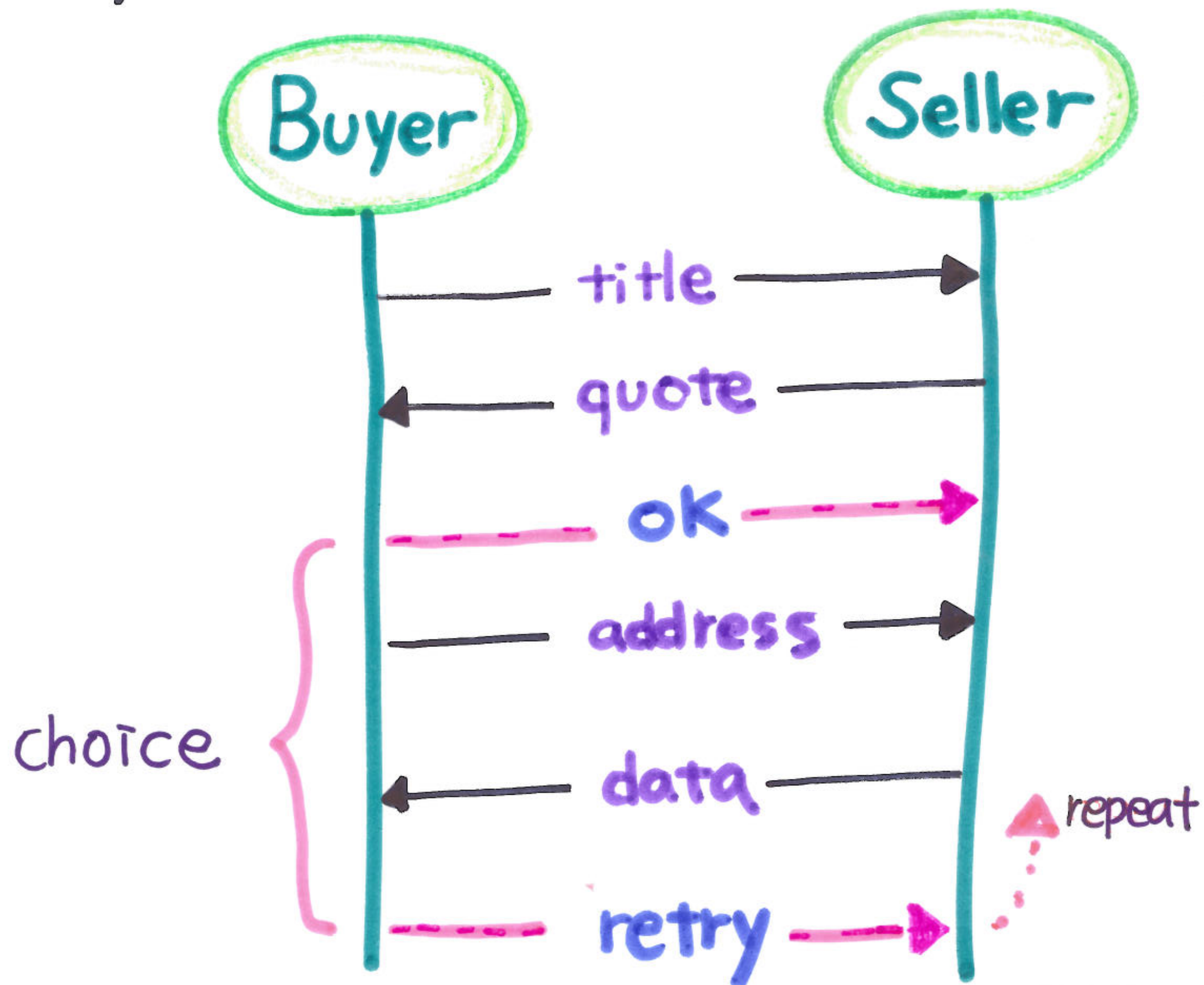
**100 x true**      **Wrong**

**Types for *Protocol*?**

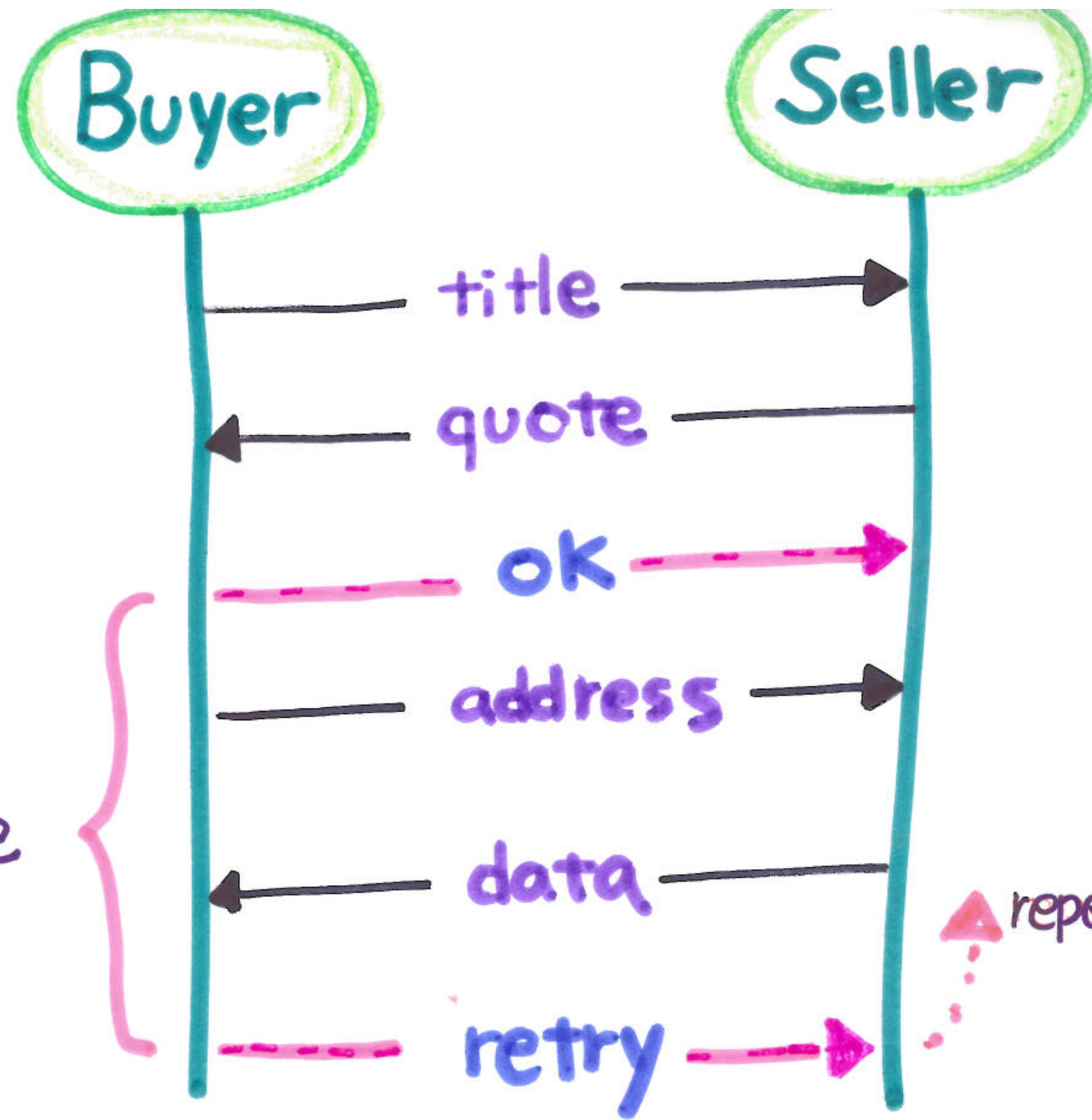
# Binary Session Types: Buyer - Seller Protocol



# Binary Session Types: Buyer - Seller Protocol



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

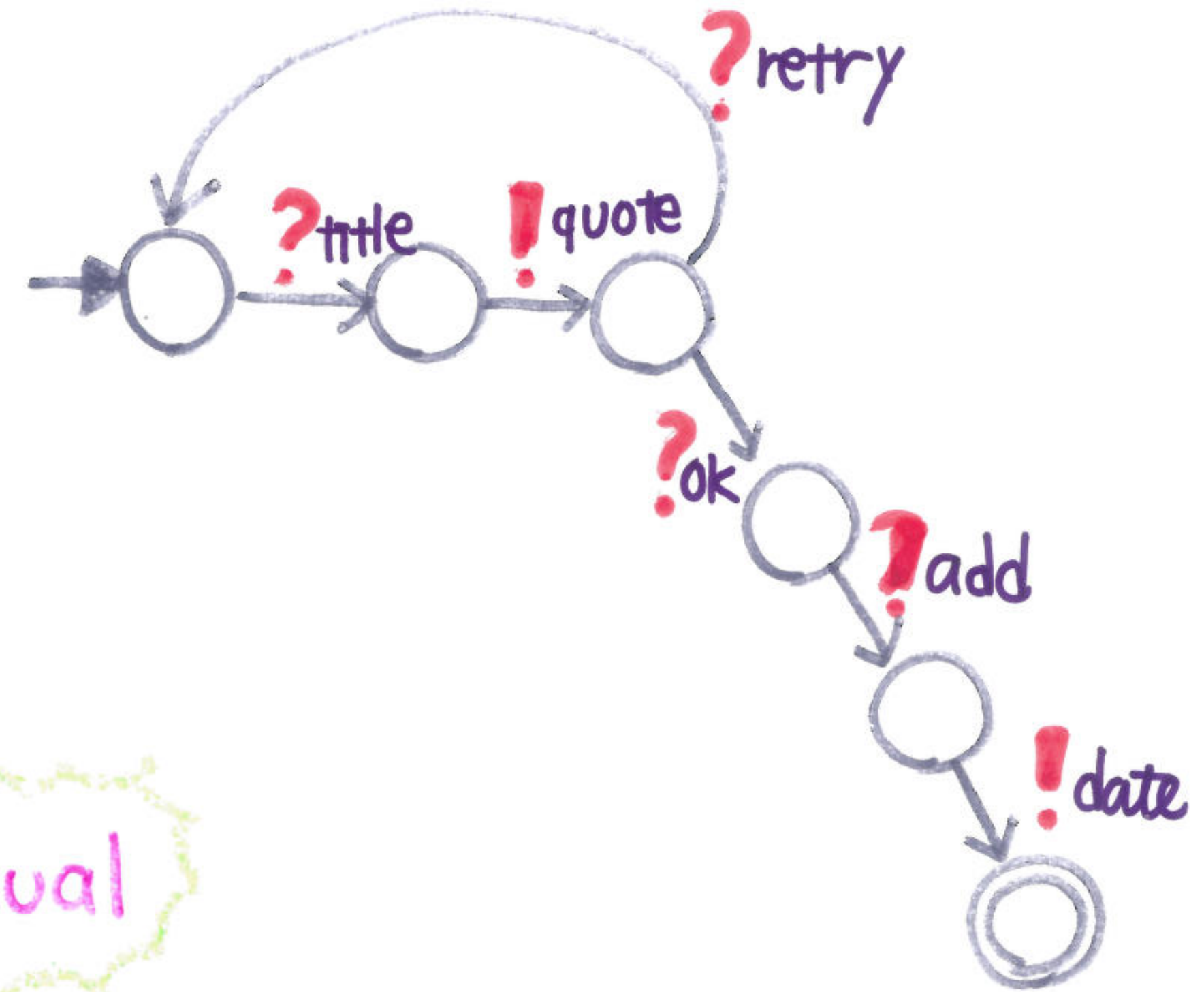
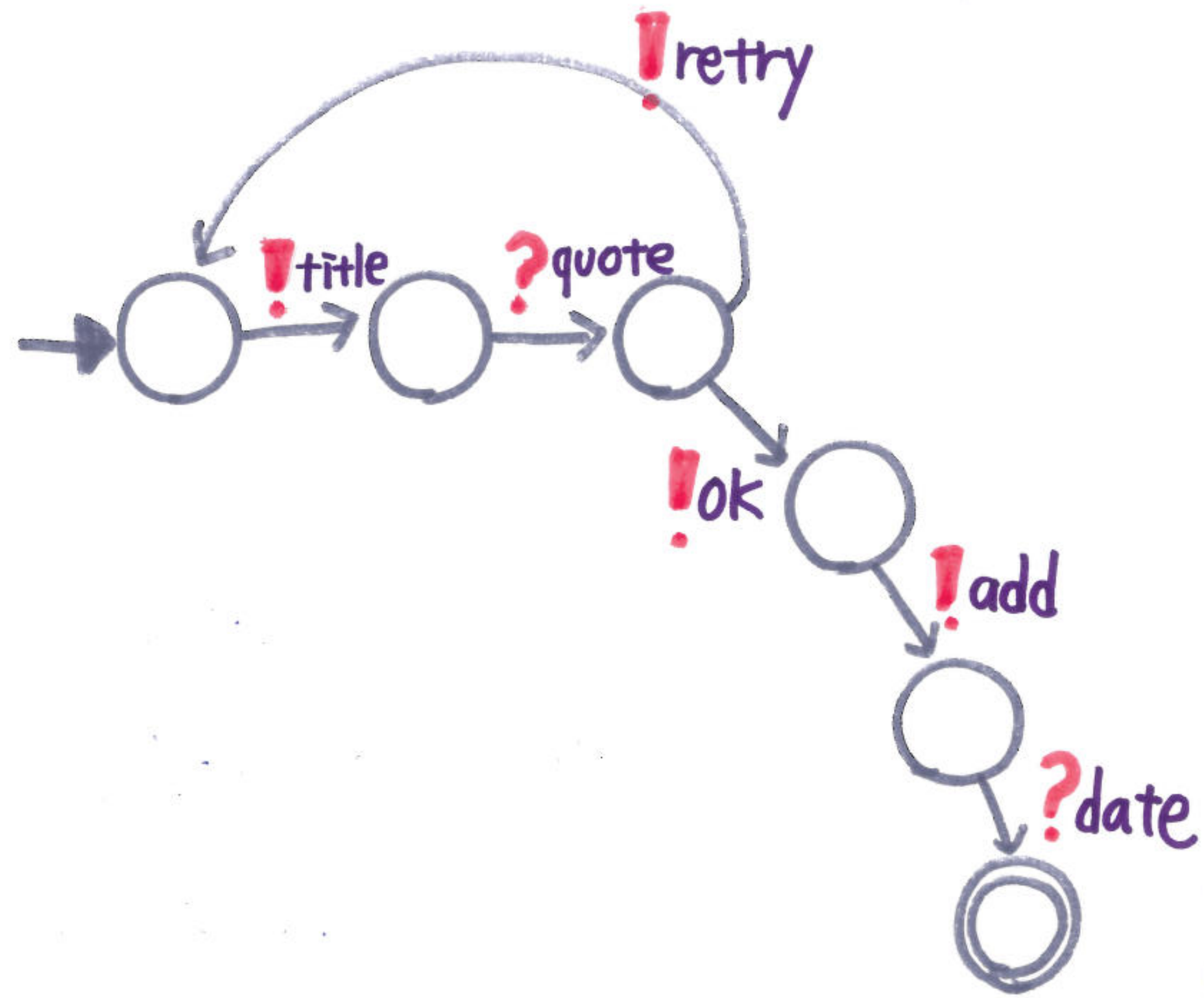


P has  $T$   
 Q has  $\overline{T}$  *dual*  
 P | Q typable

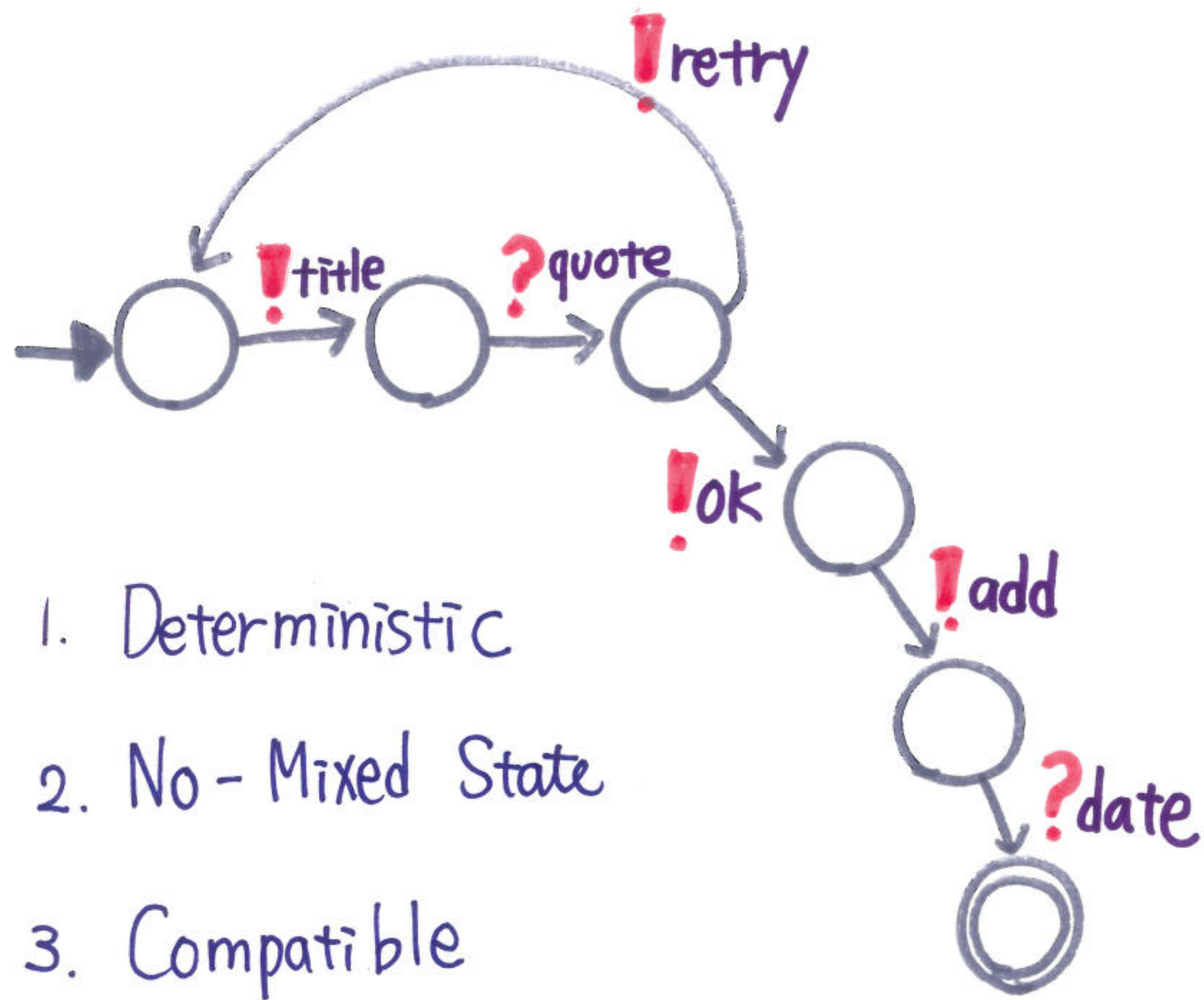
nt! Title ; ? Quote ; ! { ok: ! Add ; ? Date, **retry**: t }

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

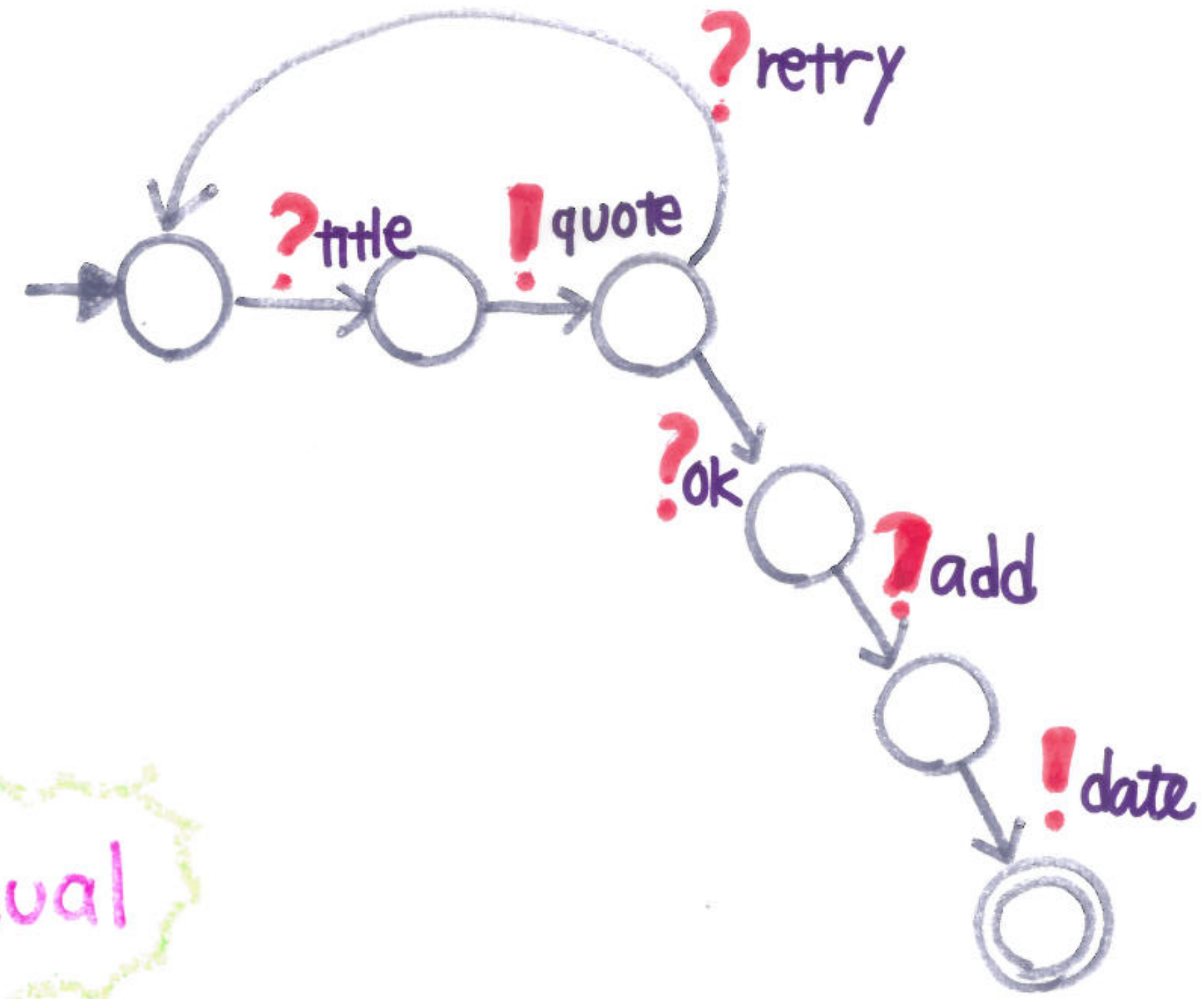
# Communicating Automata [1980s]



dual



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



dual

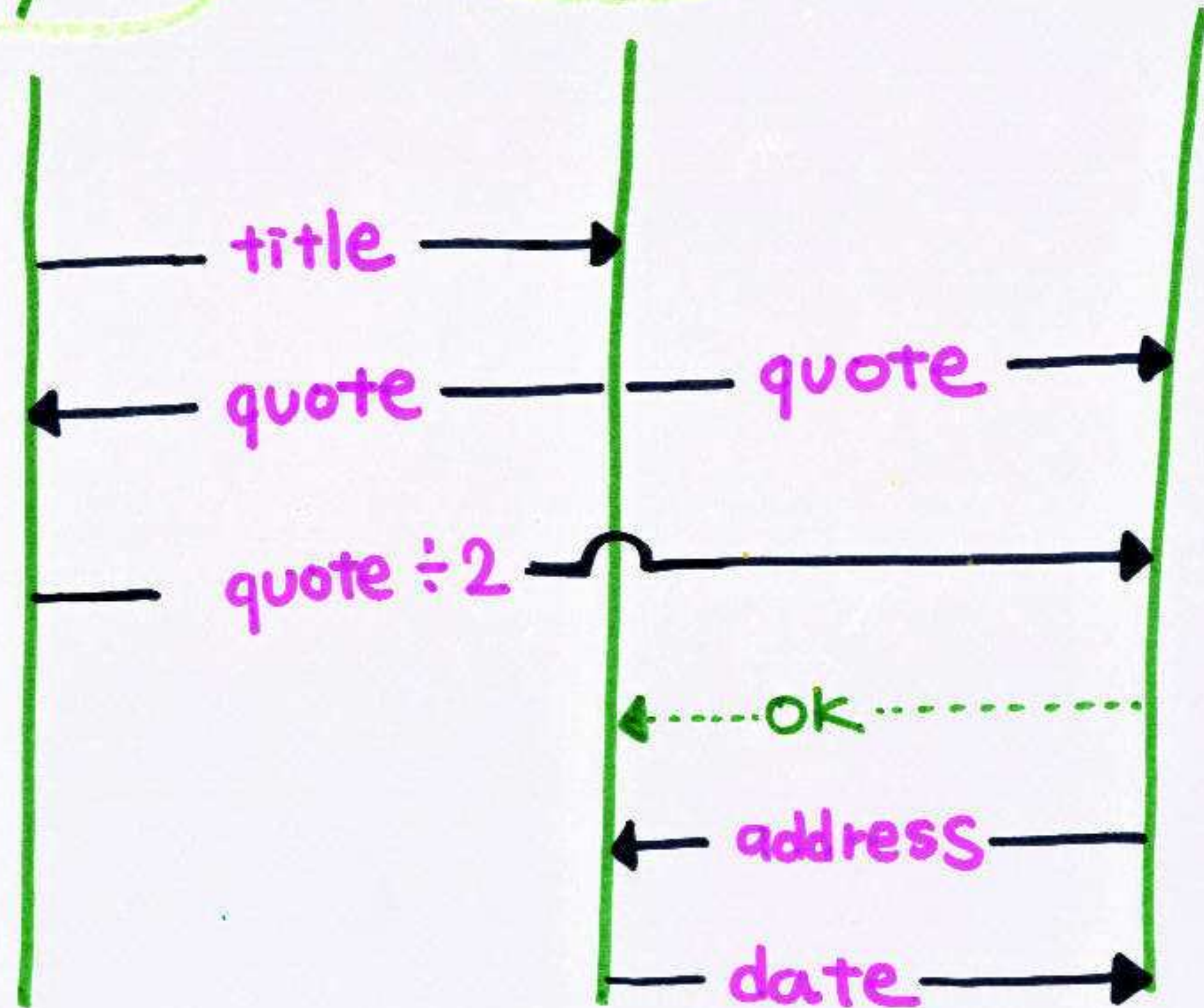
[Gouda et al 1986] Two compatible machines without mixed states which are deterministic satisfy deadlock-freedom.

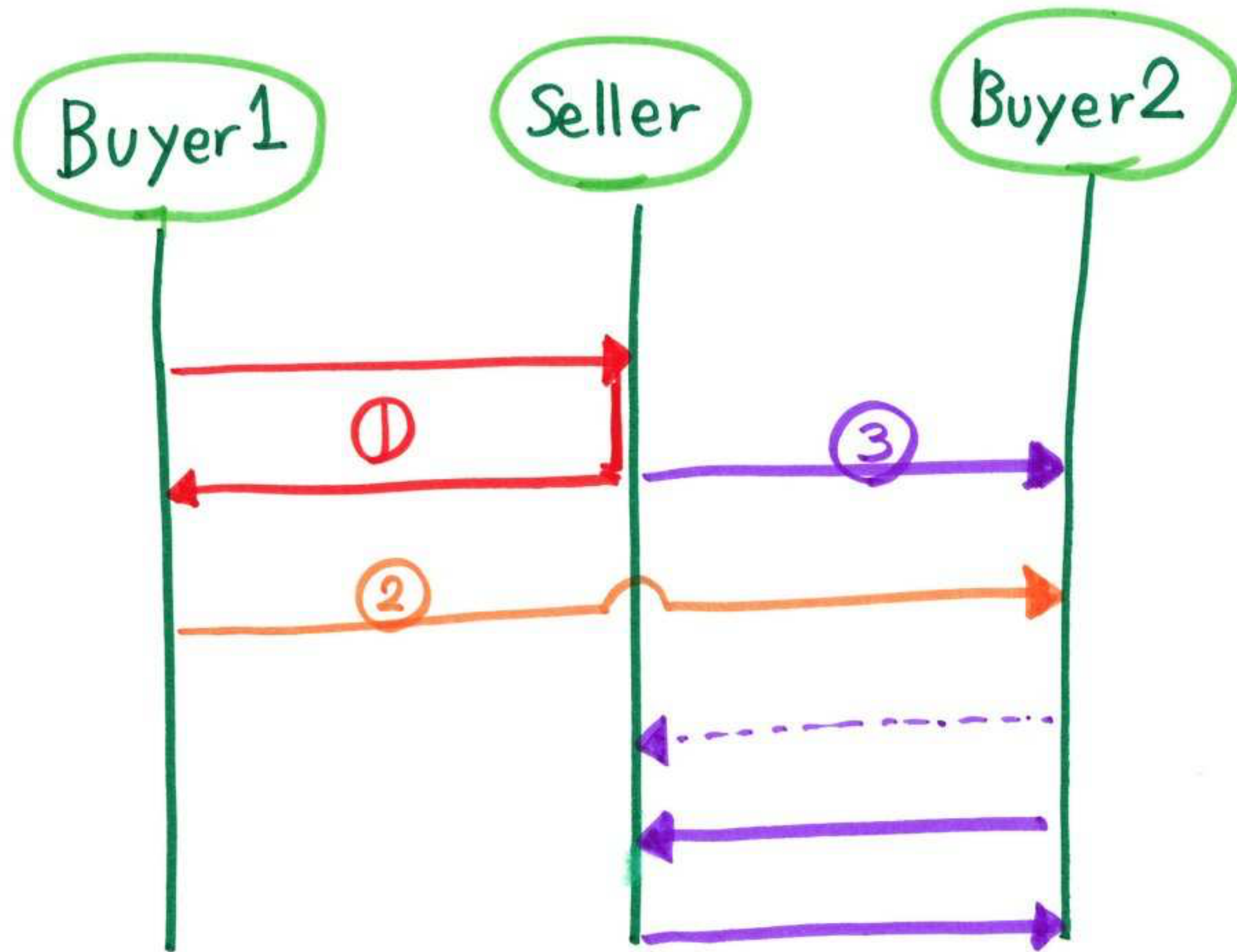
# Multiparty Session Types

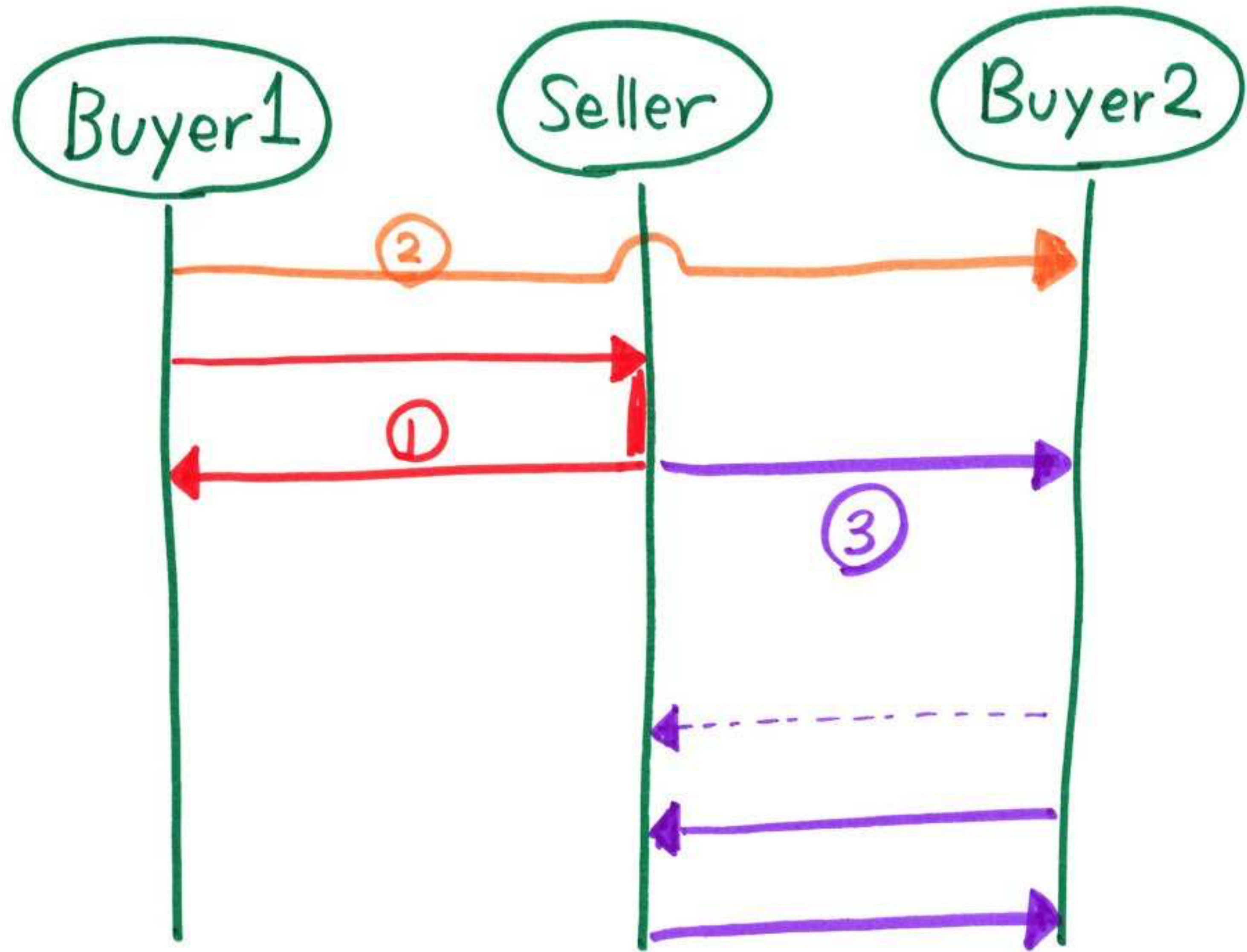
Buyer1

Seller

Buyer2





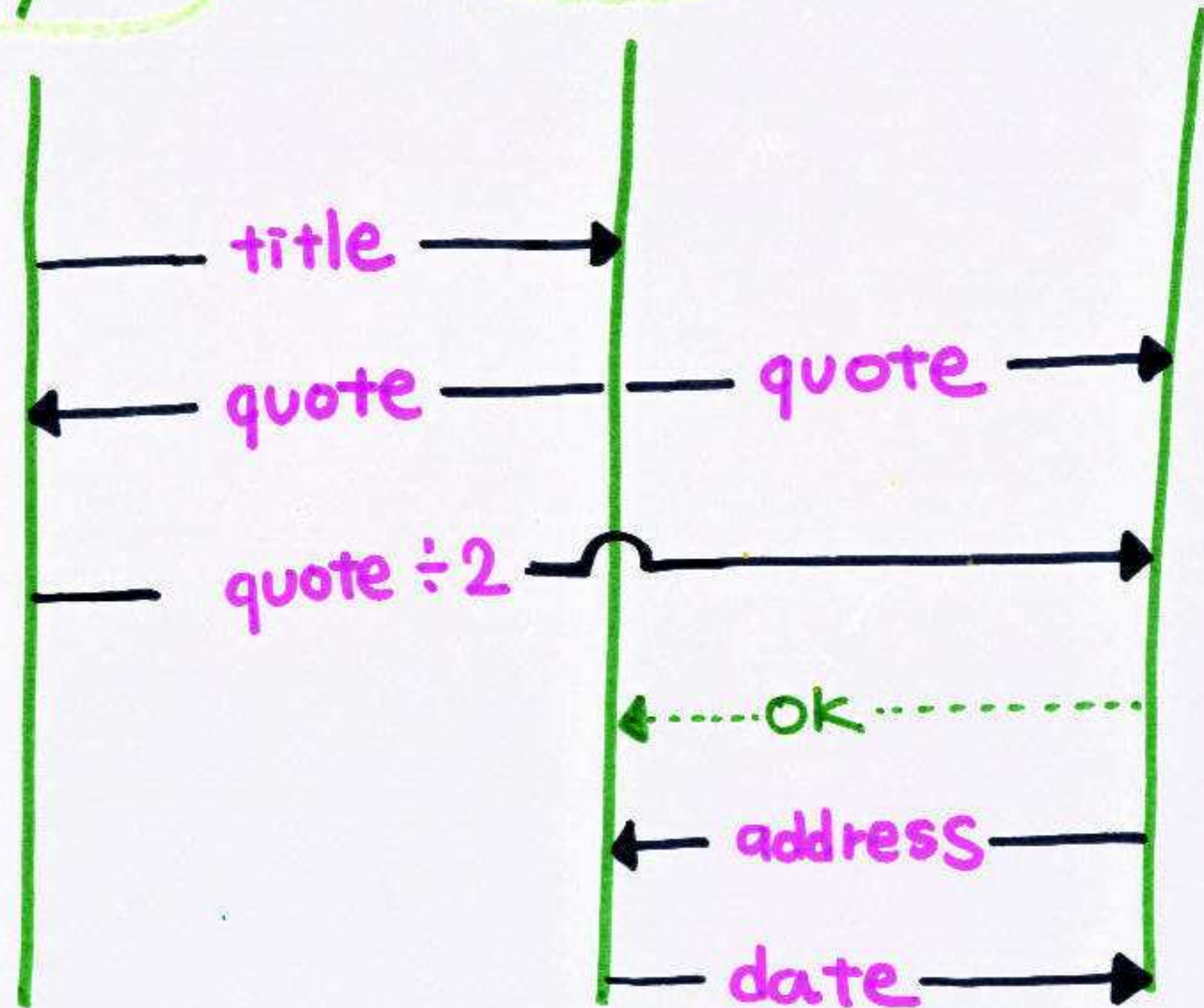


# Multiparty Session Types

Buyer1

Seller

Buyer2



Alice

Bob

Carol

CA?c ; AB!a

AB?a ; BC!b

BC?b ; CA!c

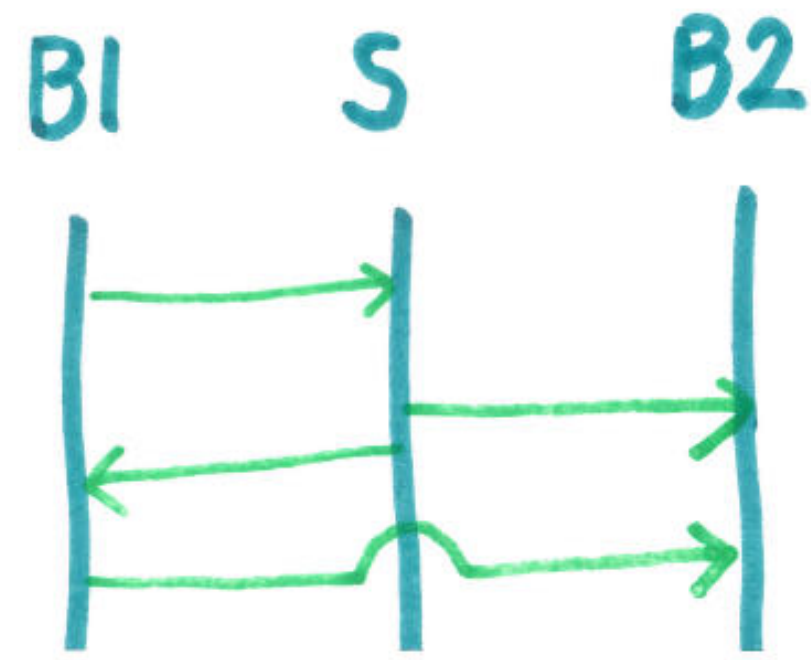


3 dual pairs

If you use  
binary Session  
Types ...

Deadlock!

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



ⓐ

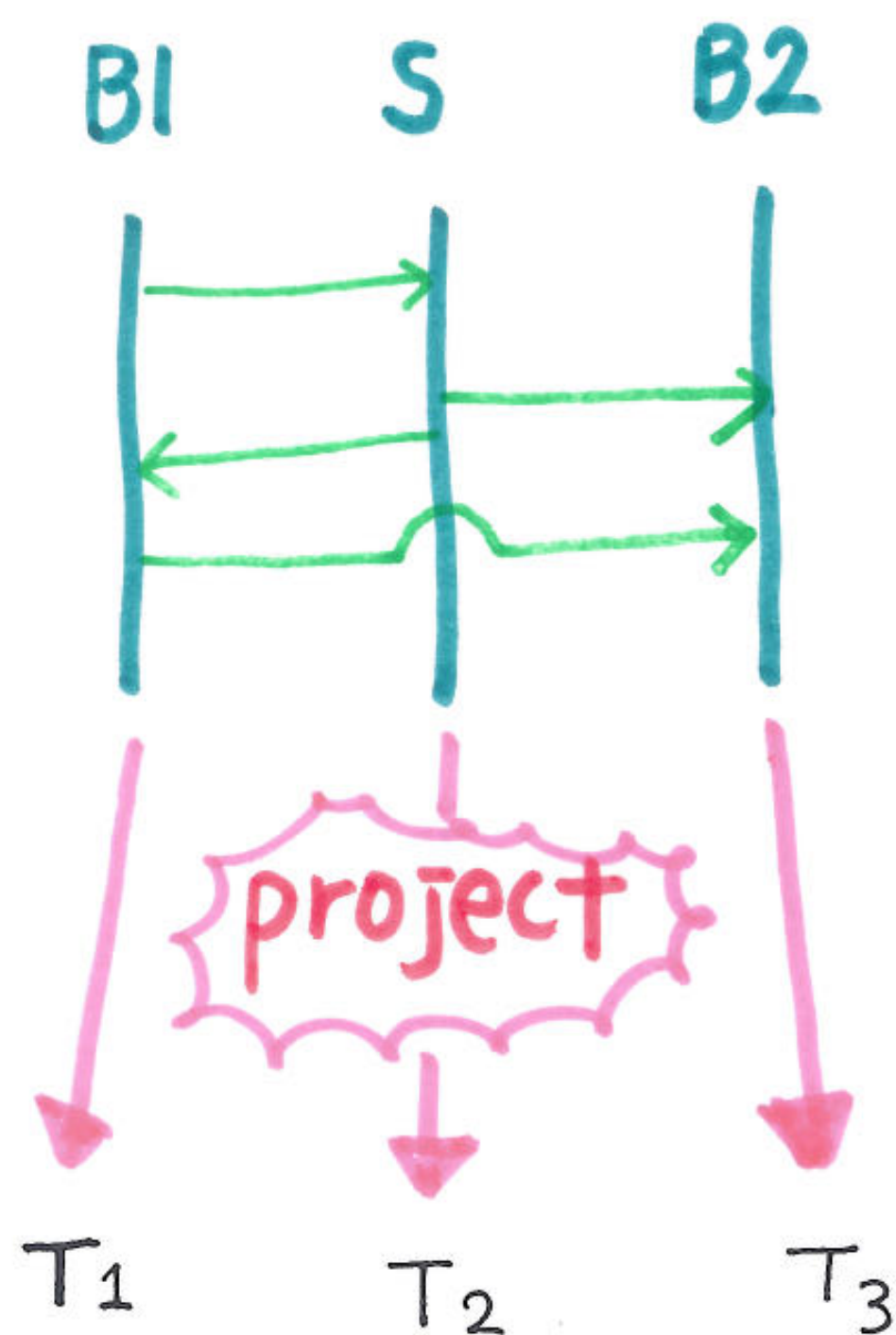
BI  $\rightarrow$  S Int.

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

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

STEP 1

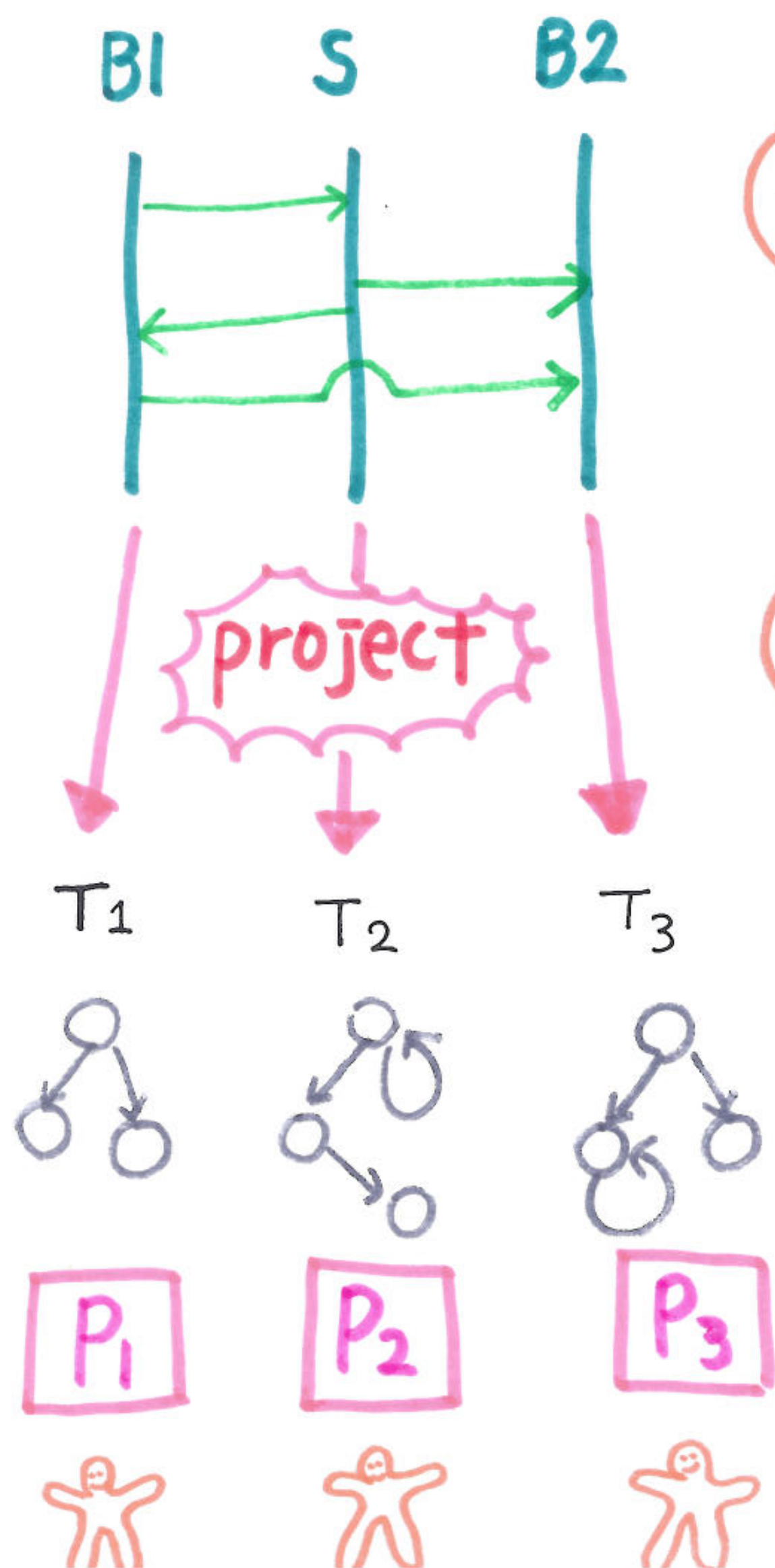
Write Global Type

STEP 2

Project to Local Types

# Multi party Session Types

[Honda, Yoshida, Carbone 2008]



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

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

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

## STEP 1

Write Global Type

## STEP 2

Project to Local Type

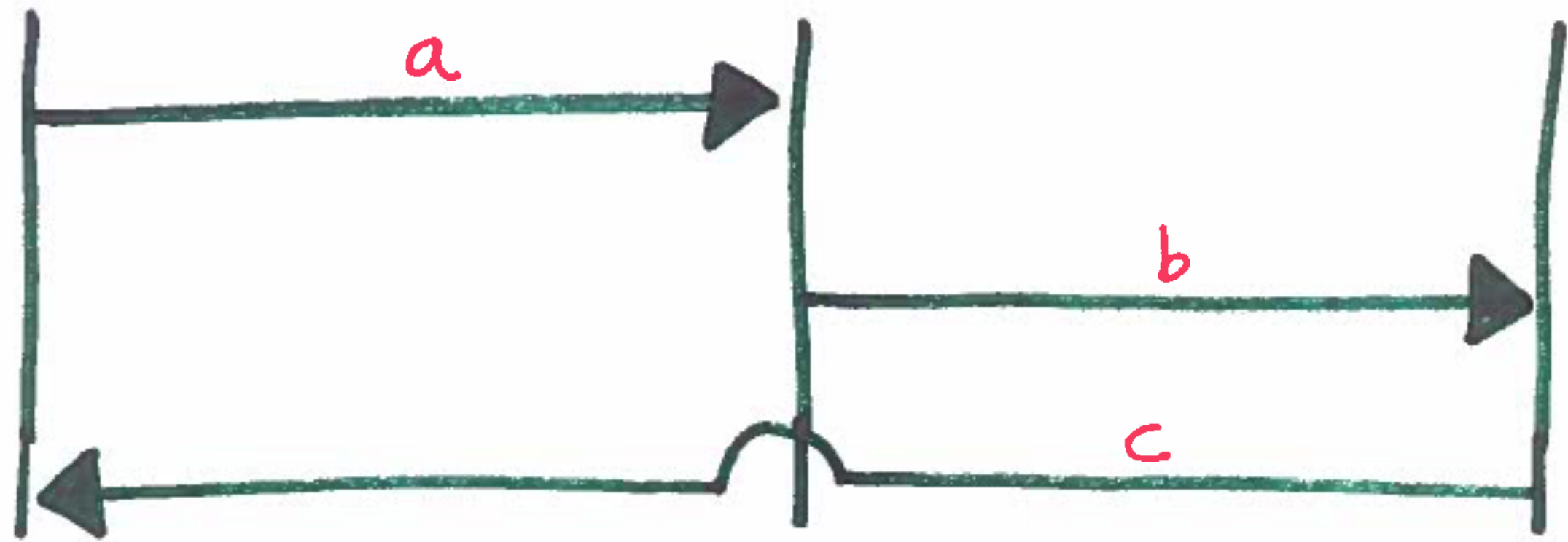
## STEP 3

- Static Check
- Generate Code
- Run-time check

Alice

Bob

Carol



Global Type

Alice  $AB!a; CA?c$

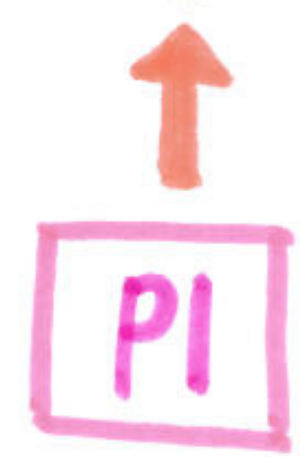
Bob  $AB?a; BC!b$

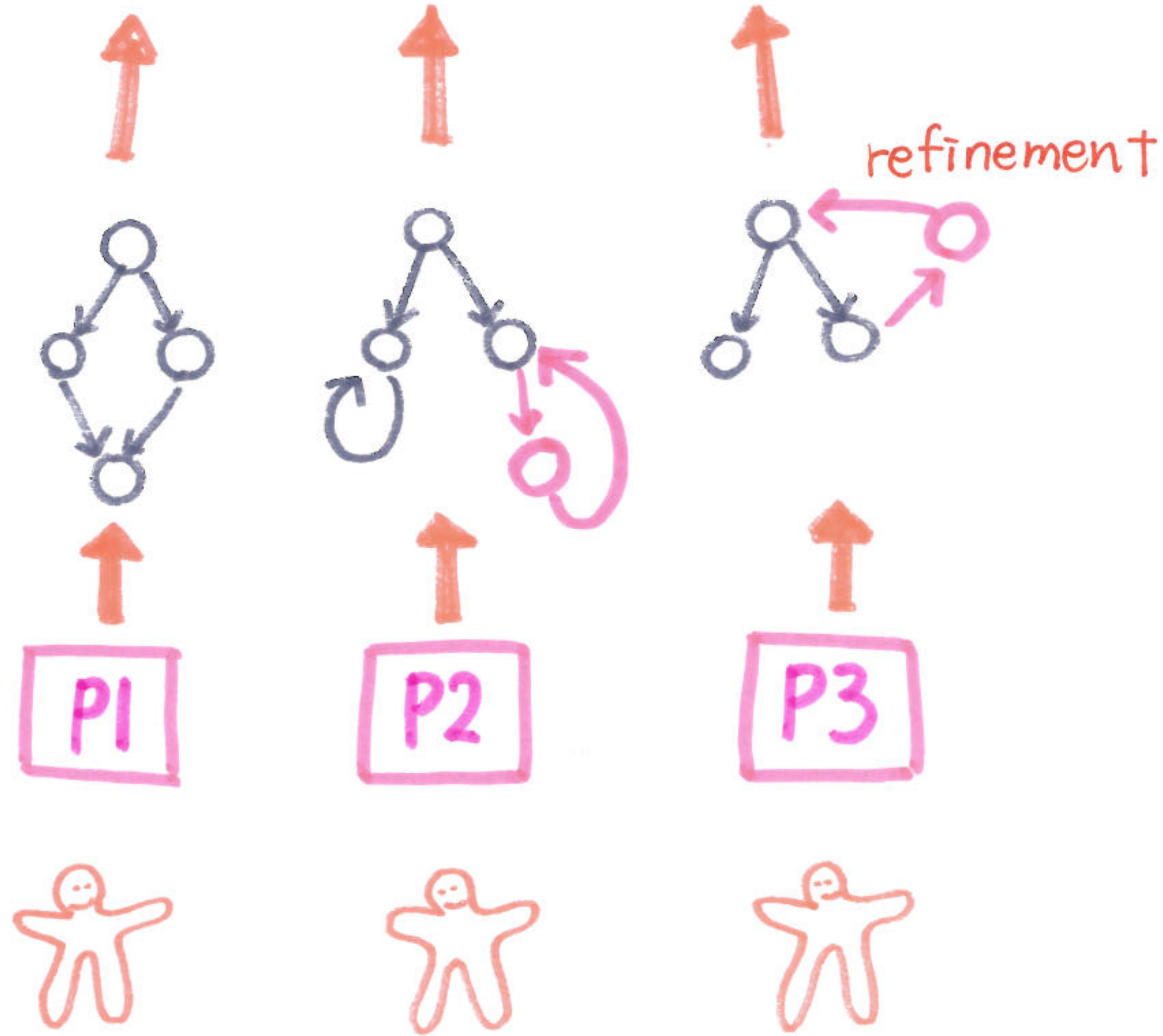
Carol  $BC?b; CA!c$

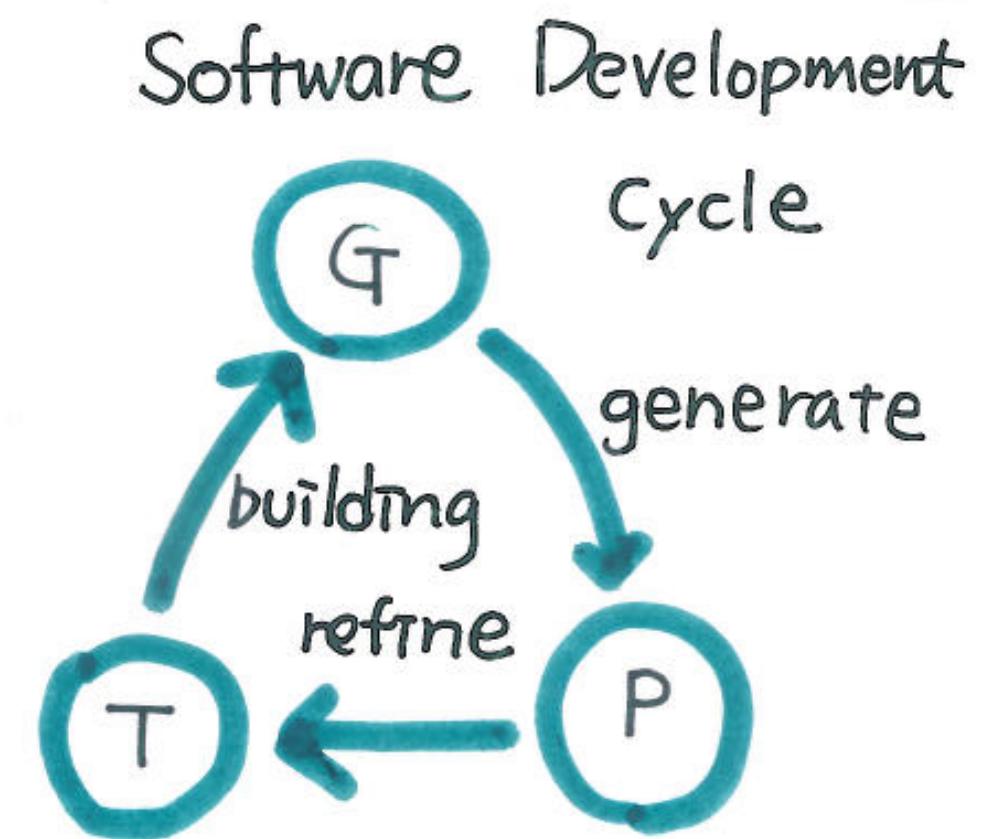
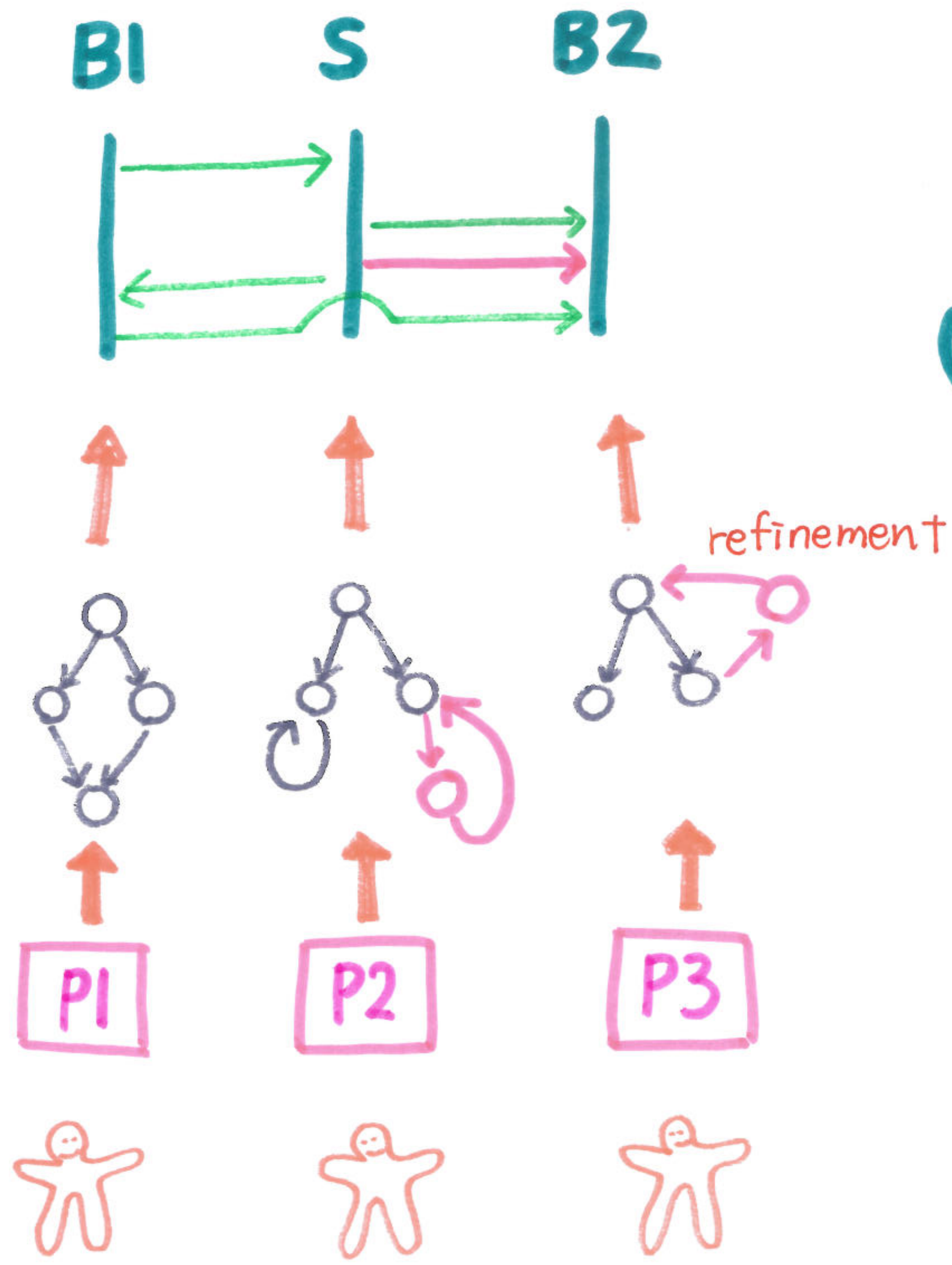


LOCAL TYPES

NO Deadlock







- Optimisation
- refinement
- inference
- Testing

# Mobility Reading Group

<http://mrg.cs.ox.ac.uk/>



The screenshot shows the website for the Mobility Reading Group. At the top left is a logo consisting of a blue Greek letter pi ( $\pi$ ) with the words "session type" in a small box above it. To the right of the logo is the text "MobilityReadingGroup" in a large, bold, black font, followed by "π-calculus, Session Types research at the University of Oxford" in a smaller, grey font. Below this is a dark grey navigation bar with white text for "Home", "People", "Publications", "Grants", "Talks", "Tutorials", "Tools", "Awards", and "Kohei Honda". The "Home" link is highlighted with a blue underline. Below the navigation bar are two main sections: "NEWS" on the left and "SELECTED PUBLICATIONS" on the right. The "NEWS" section contains two entries: one dated "22 Mar 2022" about an MEng student receiving awards, and another dated "6 Aug 2021" about an interview with award winners. The "SELECTED PUBLICATIONS" section contains three entries: one for "2023" about a paper on causal computational complexity, one for "2022" about a paper on deadlock-free asynchronous message reordering, and one for "2022" about a paper on design-by-contract for flexible multiparty session protocols.

**session type**  $\pi$  **MobilityReadingGroup**  
π-calculus, Session Types research at the University of Oxford

Home People Publications Grants Talks Tutorials Tools Awards Kohei Honda

## NEWS

22 Mar 2022

MEng student, Zak Cutner, awarded Microsoft Prize and Distinguished Project award.

6 Aug 2021

Nobuko Yoshida, with Francisco Ferreira and Adam D. Barwell, conducted an interview with the CONCUR Test-of-Time Award winners, Uwe Nestmann and Benjamin C. Pierce. The full interview can be found here

24 Mar 2021

## SELECTED PUBLICATIONS

### 2023

Romain Demangeon, Nobuko Yoshida: [Causal Computational Complexity of Distributed Processes](#). IC 2023 : 104998.

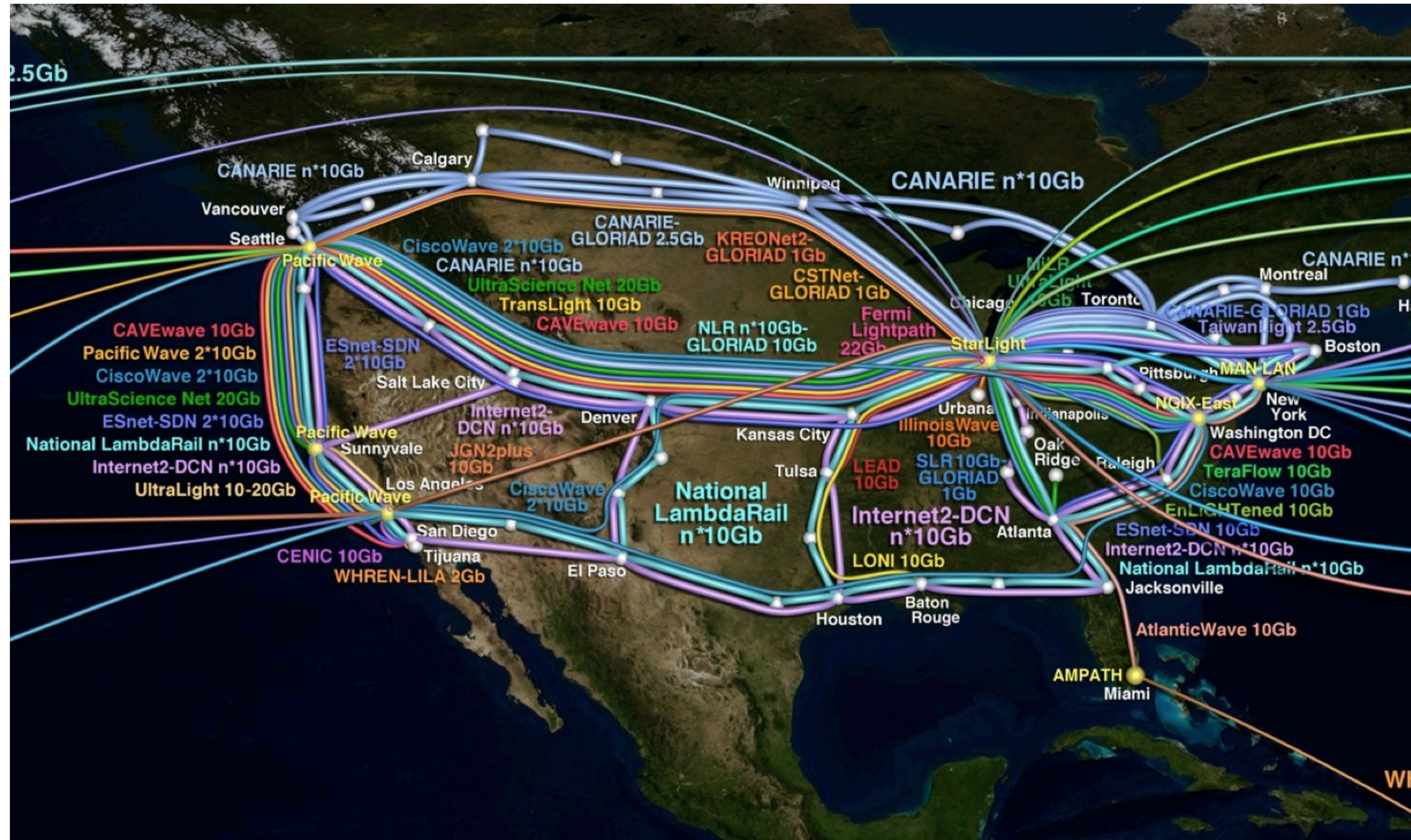
---

### 2022

Zak Cutner, Nobuko Yoshida, Martin Vassor: [Deadlock-Free Asynchronous Message Reordering in Rust with Multiparty Session Types](#). PPOPP '22 : 261 - 246.

Lorenzo Gheri, Ivan Lanese, Neil Sayers, Emilio Tuosto, Nobuko Yoshida: [Design-by-Contract for Flexible Multiparty Session Protocols](#). ECOOP 2022 : 8:1 - 8:28.

# Some Applications on Multiparty Session Types



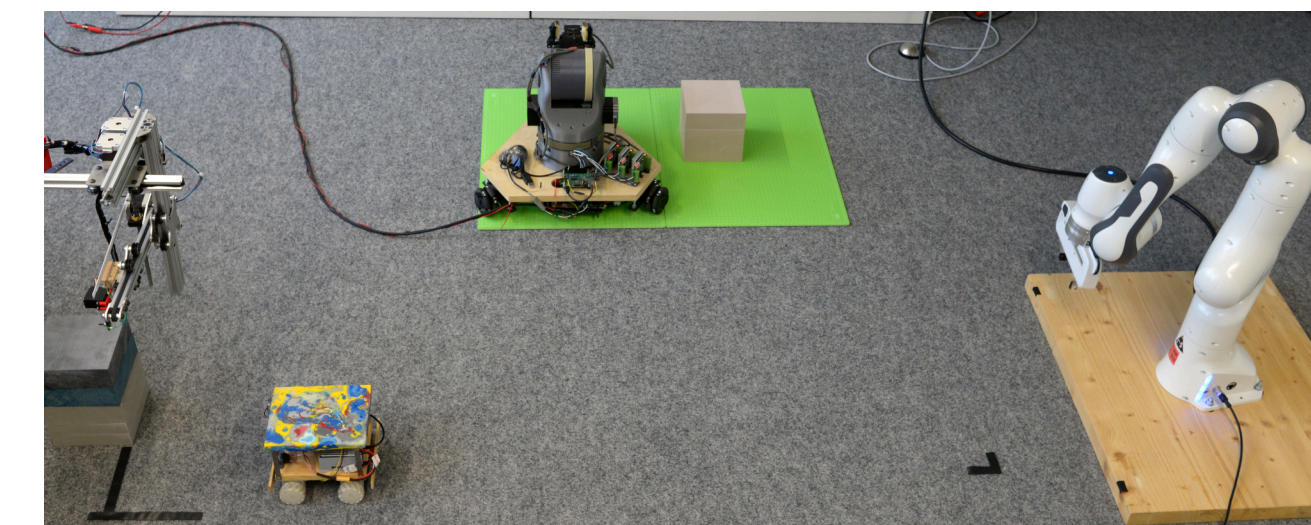
Ocean Observatories Initiative

Distributed Tracing



OpenTelemetry

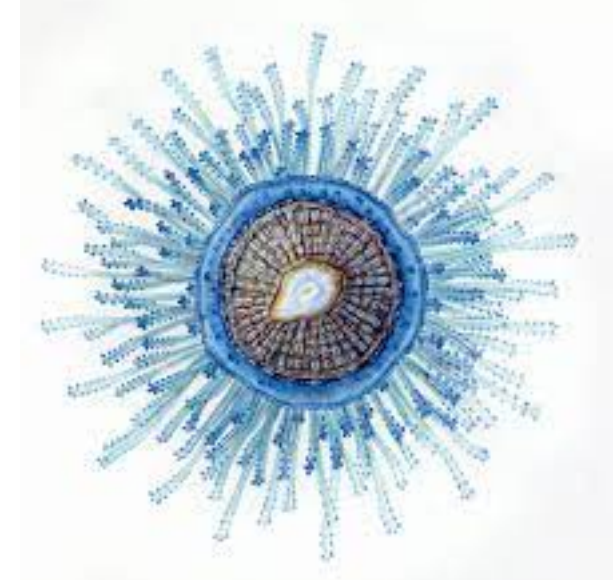
Robotics



Mechanisation

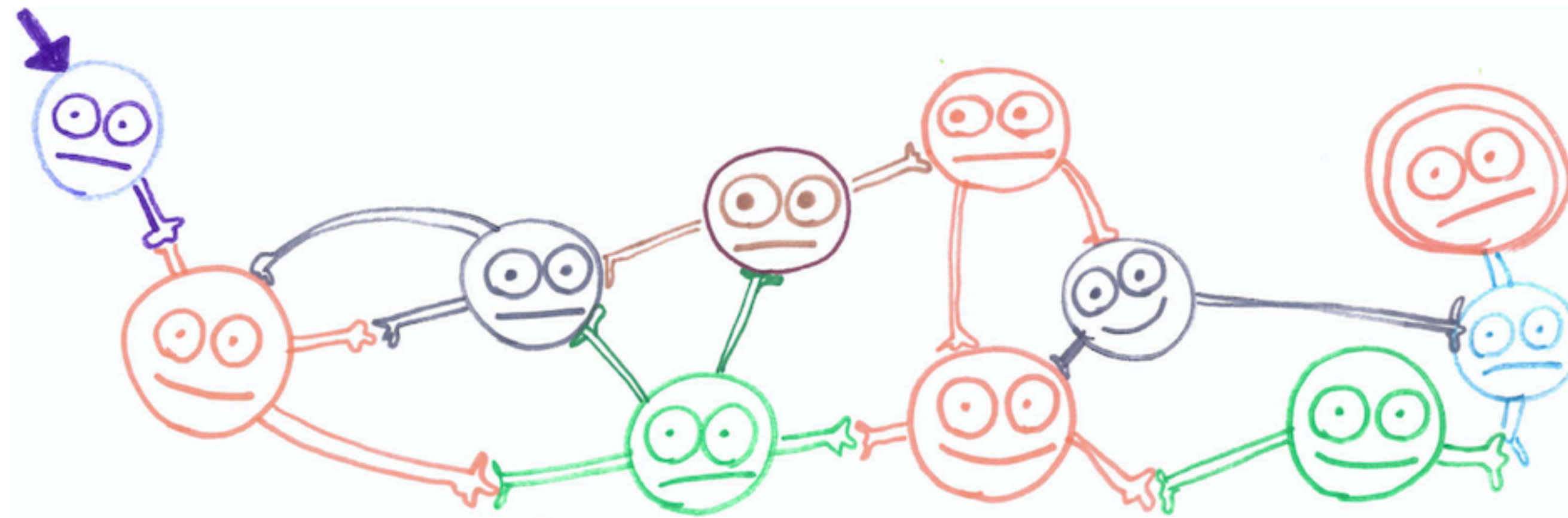


Zooid



# Optimising Asynchronous Communication in Rust

Deadlock-Free Message Reordering  
with Multiparty Session Types **[PPoPP 2022]**



Zak Cutner, NY and Martin Vassor

# Introduction

## Rust Language

- Modern systems language focussed on **safety** and **performance**

# Introduction

## Rust Language

- Modern systems language focussed on **safety** and **performance**
- “Most loved language” for past five years on StackOverflow

# Introduction

## Rust Language

- Modern systems language focussed on **safety** and **performance**
- “Most loved language” for past five years on StackOverflow
- Particular emphasis on safe concurrency using **message passing**

# Introduction

## Rust Language

- Modern systems language focussed on **safety** and **performance**
- “Most loved language” for past five years on StackOverflow
- Particular emphasis on safe concurrency using **message passing**
- **Affine** type system is well-suited to session types

# Ring Protocol

## Example

### Global Type

$$G = \mu t. \mathbf{A} \rightarrow \mathbf{B} : \left\{ \begin{array}{l} \mathit{add}(\mathit{i32}).\mathbf{B} \rightarrow \mathbf{C} : \left\{ \begin{array}{l} \mathit{add}(\mathit{i32}).\mathbf{C} \rightarrow \mathbf{A} : \{\mathit{add}(\mathit{i32}).\mathbf{t}\} \\ \mathit{sub}(\mathit{i32}).\mathbf{C} \rightarrow \mathbf{A} : \{\mathit{sub}(\mathit{i32}).\mathbf{t}\} \end{array} \right\} \end{array} \right\}$$

# Ring Protocol

## Example

$$G = \mu t. \mathbf{A} \rightarrow \mathbf{B} : \left\{ \begin{array}{l} \mathit{add}(\mathit{i32}).\mathbf{B} \rightarrow \mathbf{C} : \left\{ \begin{array}{l} \mathit{add}(\mathit{i32}).\mathbf{C} \rightarrow \mathbf{A} : \{ \mathit{add}(\mathit{i32}).\mathbf{t} \} \\ \mathit{sub}(\mathit{i32}).\mathbf{C} \rightarrow \mathbf{A} : \{ \mathit{sub}(\mathit{i32}).\mathbf{t} \} \end{array} \right\} \end{array} \right\}$$

# Ring Protocol

## Example

$$G = \mu t. \mathbf{A} \rightarrow \mathbf{B} : \left\{ \begin{array}{l} \mathit{add}(\mathit{i32}). \mathbf{B} \rightarrow \mathbf{C} : \left\{ \begin{array}{l} \mathit{add}(\mathit{i32}). \mathbf{C} \rightarrow \mathbf{A} : \{ \mathit{add}(\mathit{i32}). \mathbf{t} \} \\ \mathit{sub}(\mathit{i32}). \mathbf{C} \rightarrow \mathbf{A} : \{ \mathit{sub}(\mathit{i32}). \mathbf{t} \} \end{array} \right\} \end{array} \right\}$$

# Ring Protocol

## Example

$$G = \mu t. \mathbf{A} \rightarrow \mathbf{B} : \left\{ \begin{array}{l} \mathit{add}(\mathit{i32}). \mathbf{B} \rightarrow \mathbf{C} : \left\{ \begin{array}{l} \mathit{add}(\mathit{i32}). \mathbf{C} \rightarrow \mathbf{A} : \{ \mathit{add}(\mathit{i32}). \mathbf{t} \} \\ \mathit{sub}(\mathit{i32}). \mathbf{C} \rightarrow \mathbf{A} : \{ \mathit{sub}(\mathit{i32}). \mathbf{t} \} \end{array} \right\} \end{array} \right\}$$

# Ring Protocol

## Example

$$G = \mu t. \mathbf{A} \rightarrow \mathbf{B} : \left\{ \begin{array}{l} \mathit{add}(\mathit{i32}).\mathbf{B} \rightarrow \mathbf{C} : \left\{ \begin{array}{l} \mathit{add}(\mathit{i32}).\mathbf{C} \rightarrow \mathbf{A} : \{ \mathit{add}(\mathit{i32}).\mathbf{t} \} \\ \mathit{sub}(\mathit{i32}).\mathbf{C} \rightarrow \mathbf{A} : \{ \mathit{sub}(\mathit{i32}).\mathbf{t} \} \end{array} \right\} \end{array} \right\}$$

# Ring Protocol

## Example

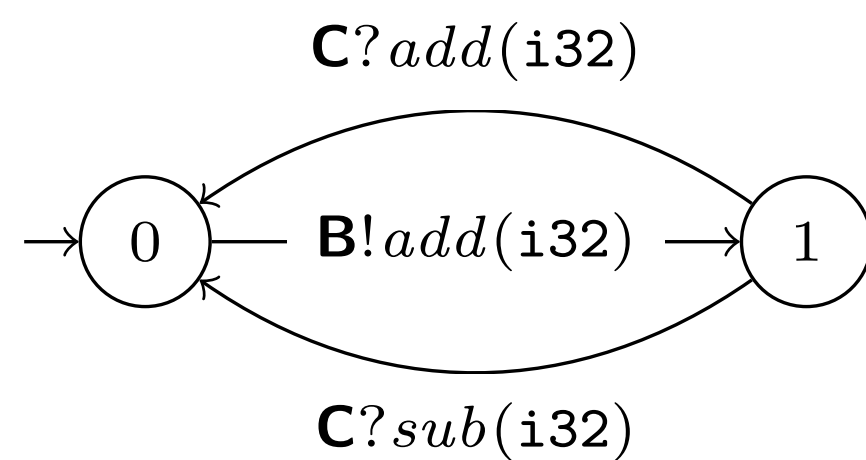
$$G = \mu t. \mathbf{A} \rightarrow \mathbf{B} : \left\{ \begin{array}{l} \mathit{add}(\mathit{i32}).\mathbf{B} \rightarrow \mathbf{C} : \left\{ \begin{array}{l} \mathit{add}(\mathit{i32}).\mathbf{C} \rightarrow \mathbf{A} : \{ \mathit{add}(\mathit{i32}).\mathbf{t} \} \\ \mathit{sub}(\mathit{i32}).\mathbf{C} \rightarrow \mathbf{A} : \{ \mathit{sub}(\mathit{i32}).\mathbf{t} \} \end{array} \right\} \end{array} \right\}$$

# Ring Protocol

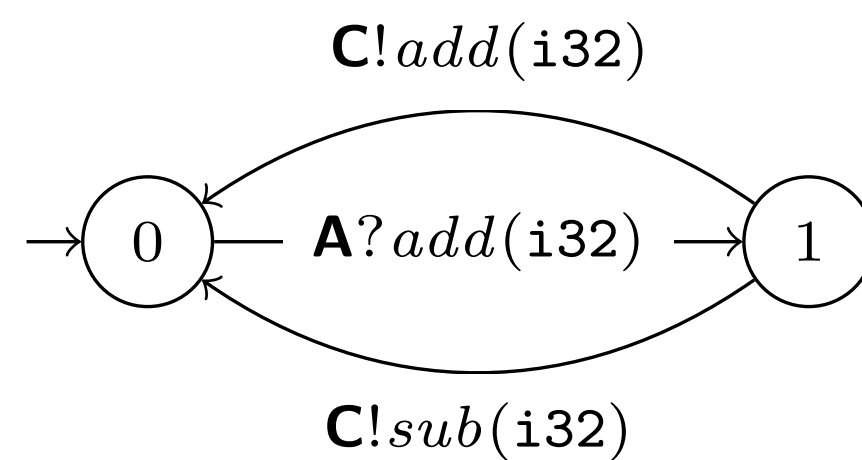
## Example

$$G = \mu t. \mathbf{A} \rightarrow \mathbf{B} : \left\{ \begin{array}{l} \mathit{add}(i32). \mathbf{B} \rightarrow \mathbf{C} : \left\{ \begin{array}{l} \mathit{add}(i32). \mathbf{C} \rightarrow \mathbf{A} : \{ \mathit{add}(i32). t \} \\ \mathit{sub}(i32). \mathbf{C} \rightarrow \mathbf{A} : \{ \mathit{sub}(i32). t \} \end{array} \right\} \end{array} \right\}$$

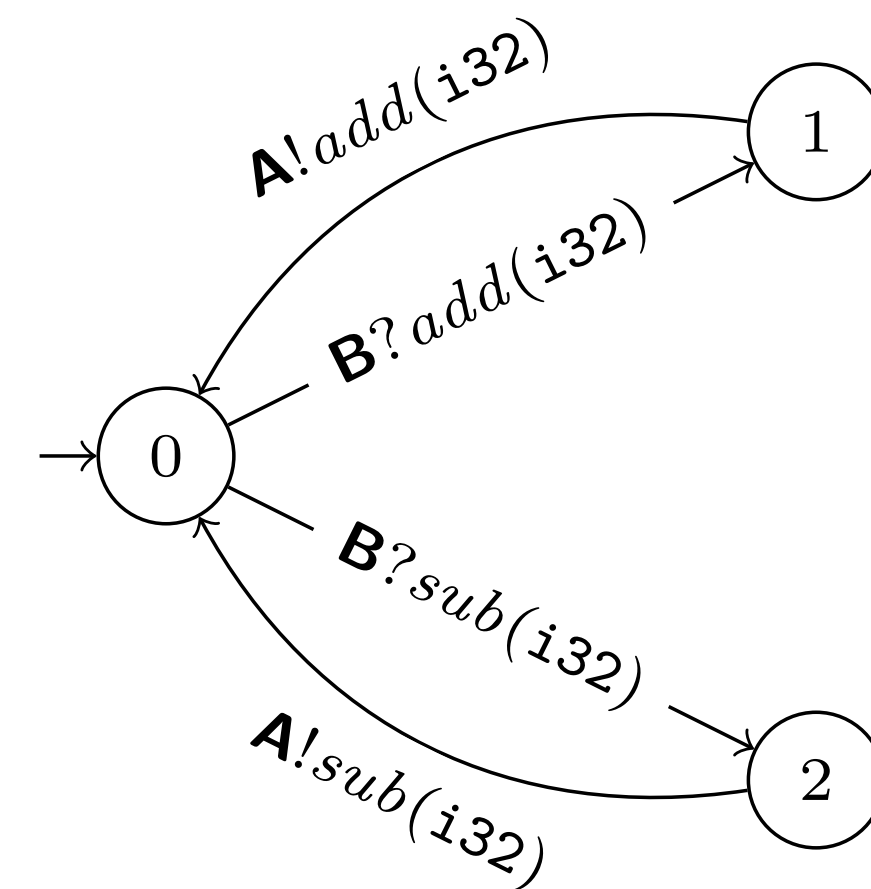
PROJECTION



PROJECTION



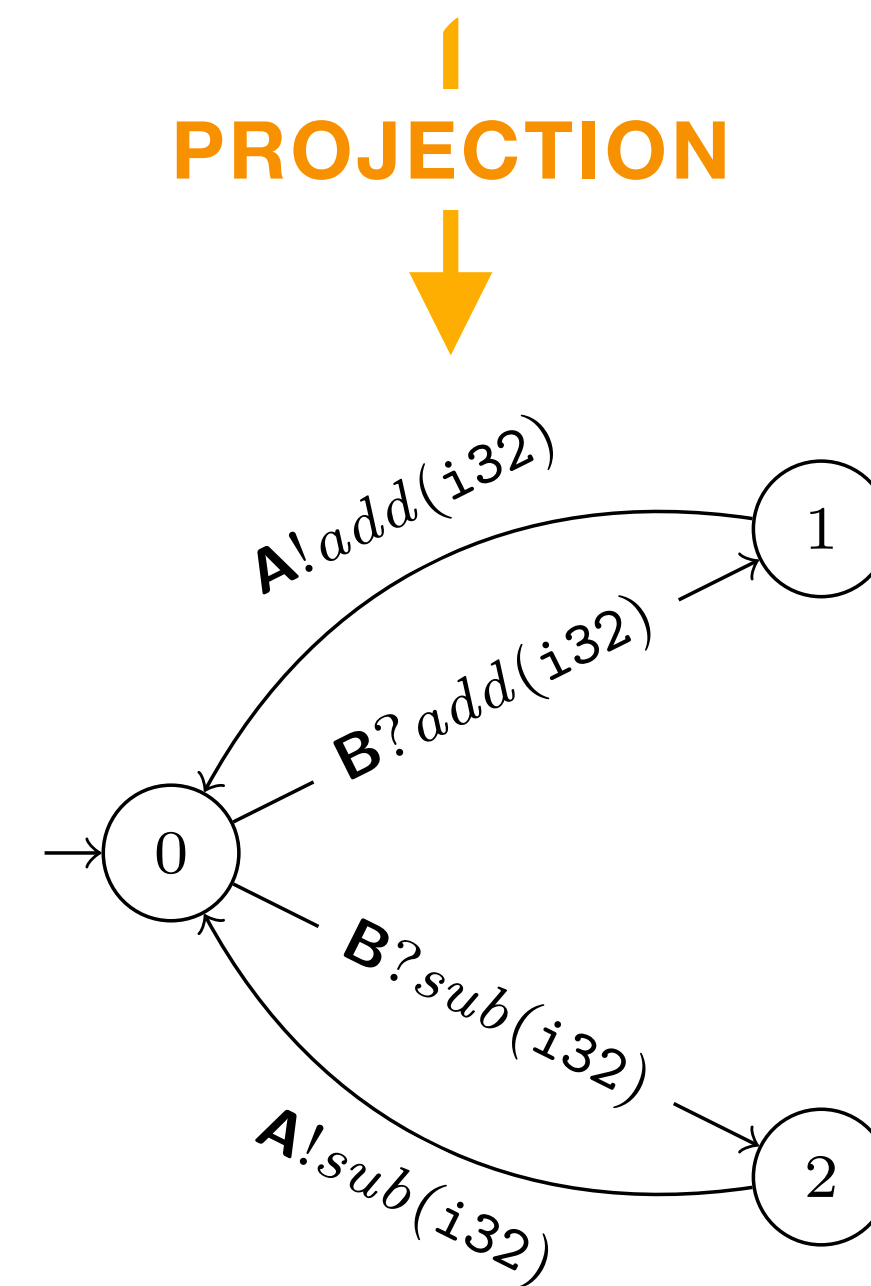
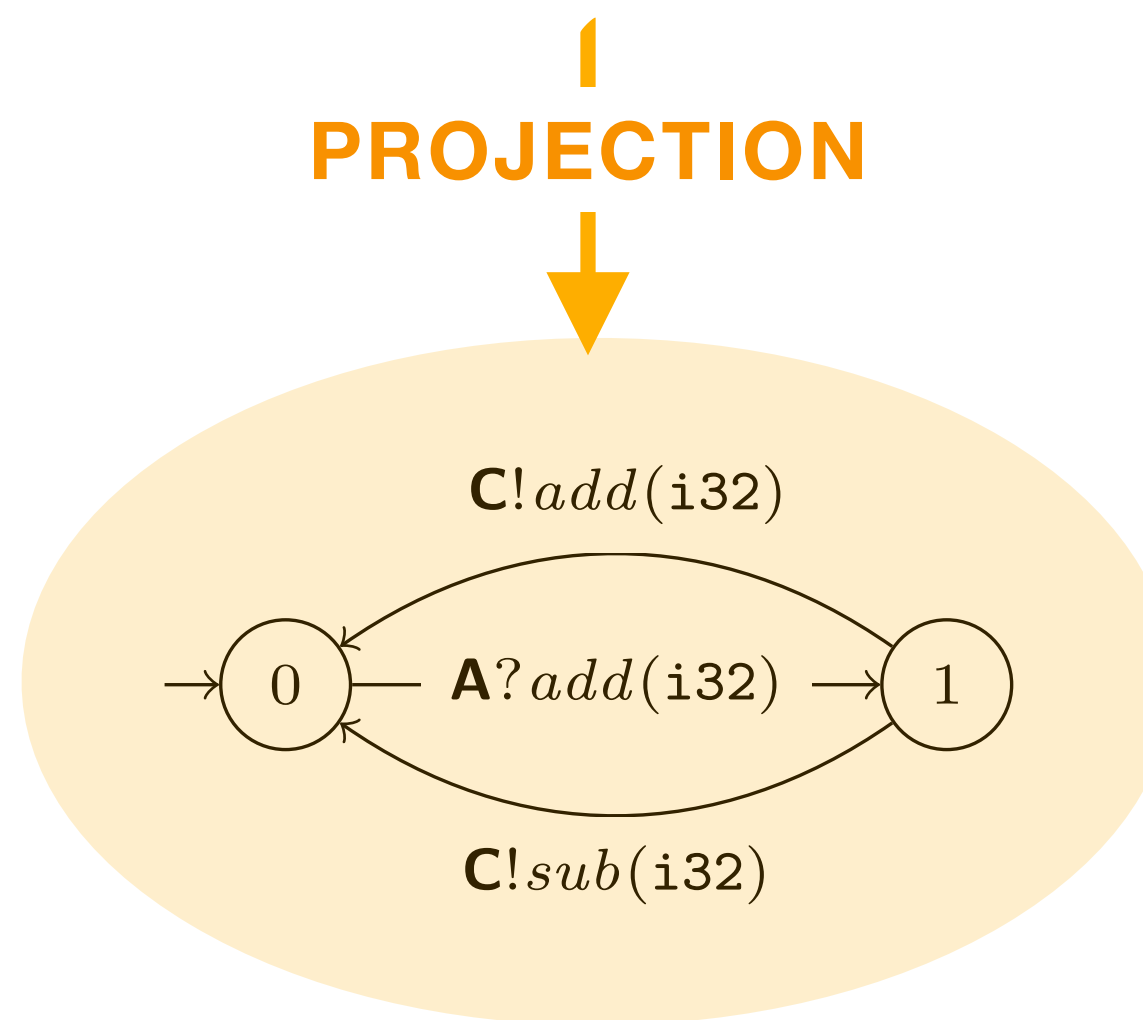
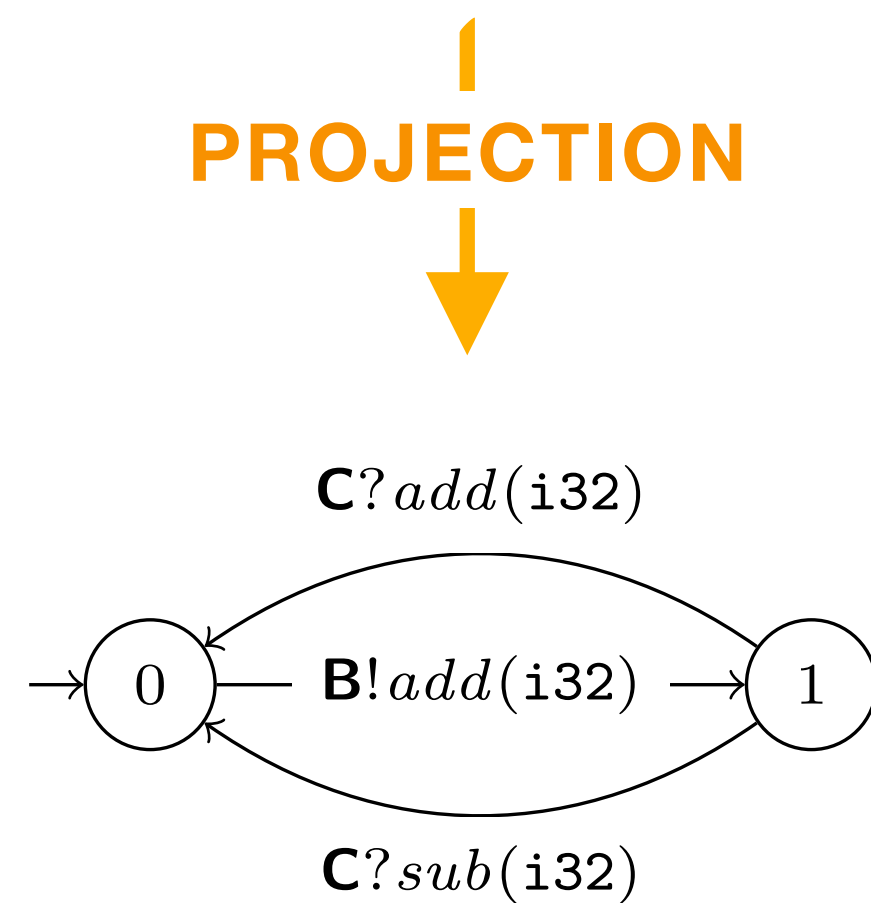
PROJECTION



# Ring Protocol

## Example

$$G = \mu t. \mathbf{A} \rightarrow \mathbf{B} : \left\{ \begin{array}{l} \mathit{add}(i32). \mathbf{B} \rightarrow \mathbf{C} : \left\{ \begin{array}{l} \mathit{add}(i32). \mathbf{C} \rightarrow \mathbf{A} : \{ \mathit{add}(i32). t \} \\ \mathit{sub}(i32). \mathbf{C} \rightarrow \mathbf{A} : \{ \mathit{sub}(i32). t \} \end{array} \right\} \end{array} \right\}$$



# Challenge

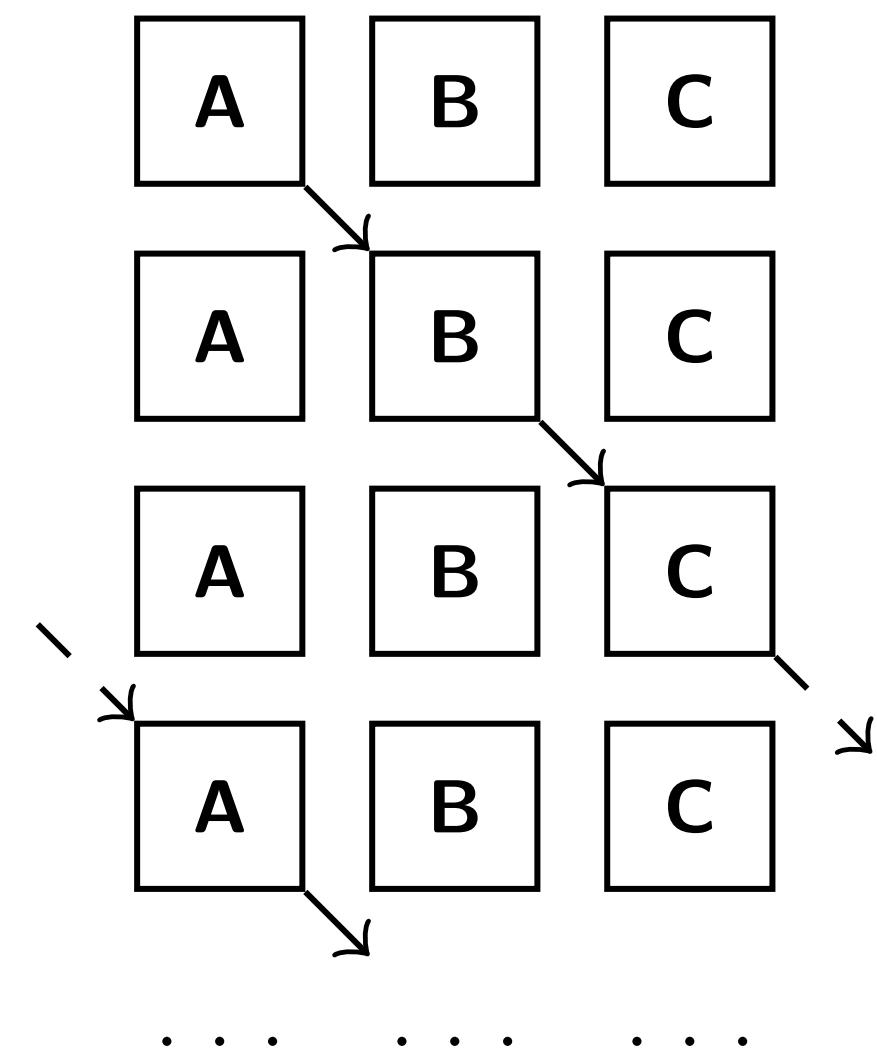
## Asynchronous Orderings

- Global types are inherently **synchronous**

# Challenge

## Asynchronous Orderings

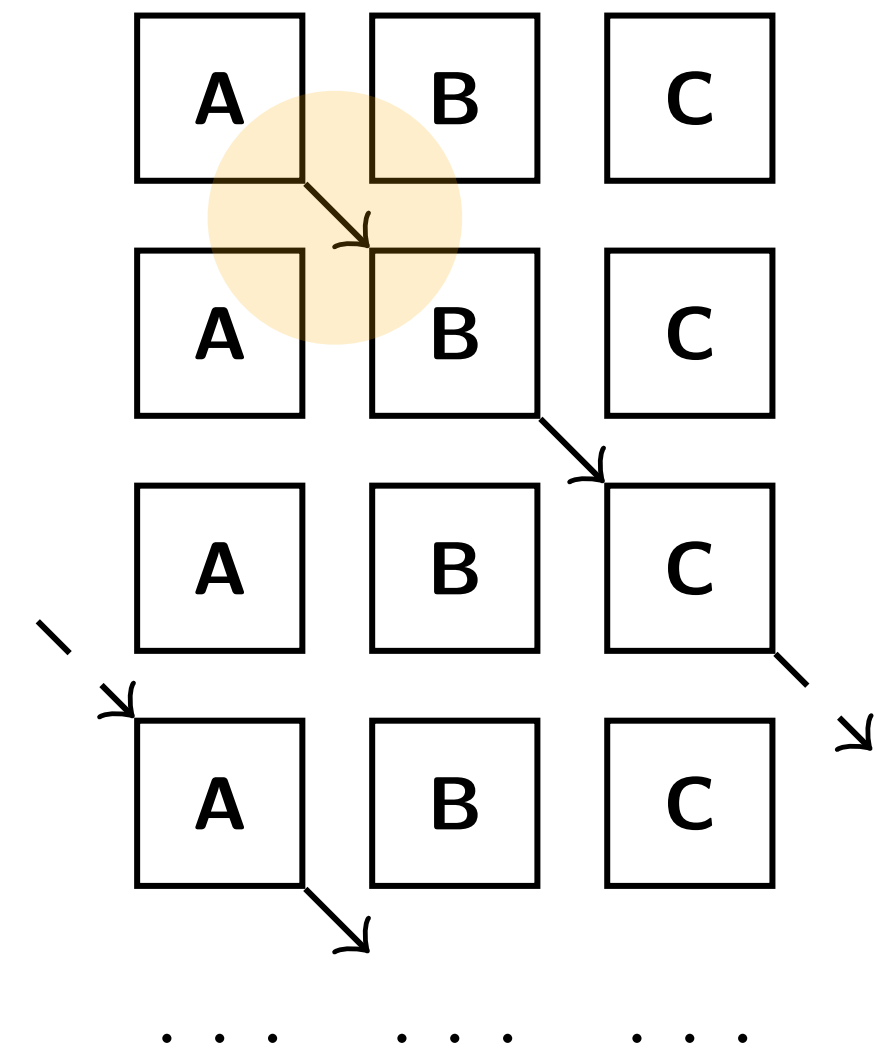
- Global types are inherently **synchronous**
  - Projection provides only one possible ordering



# Challenge

## Asynchronous Orderings

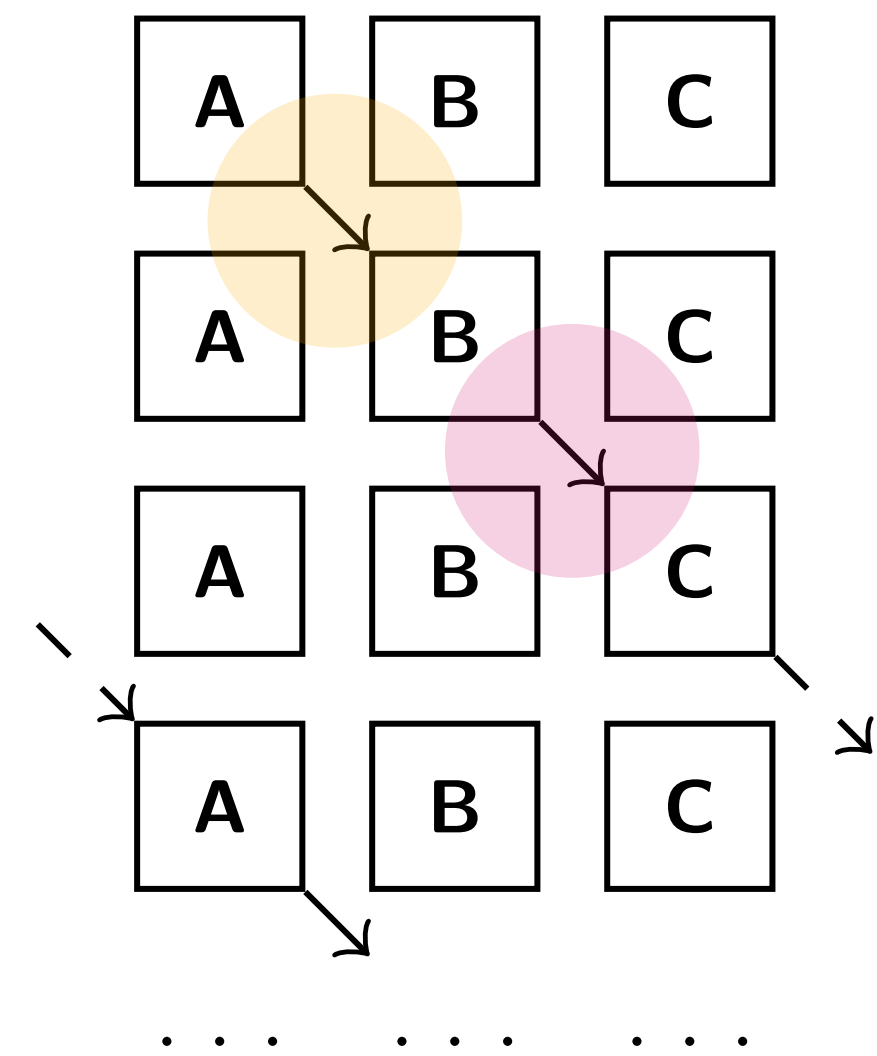
- Global types are inherently *synchronous*
  - Projection provides only one possible ordering



# Challenge

## Asynchronous Orderings

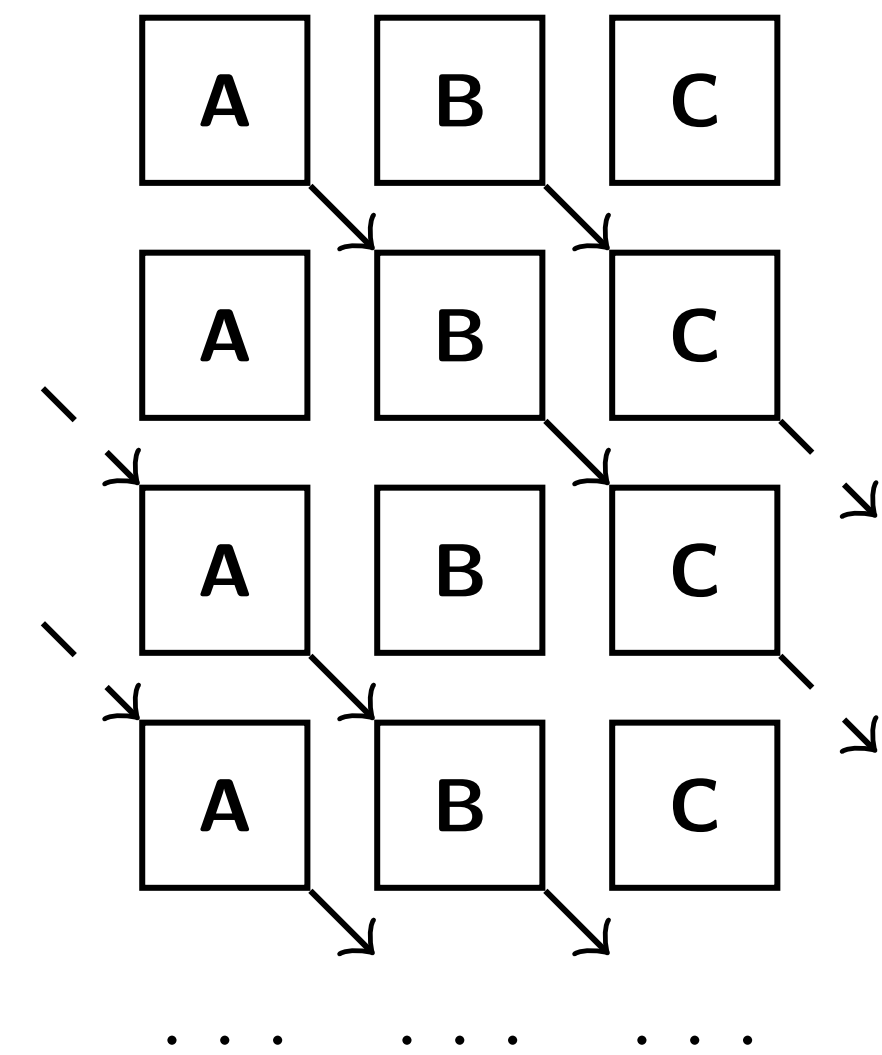
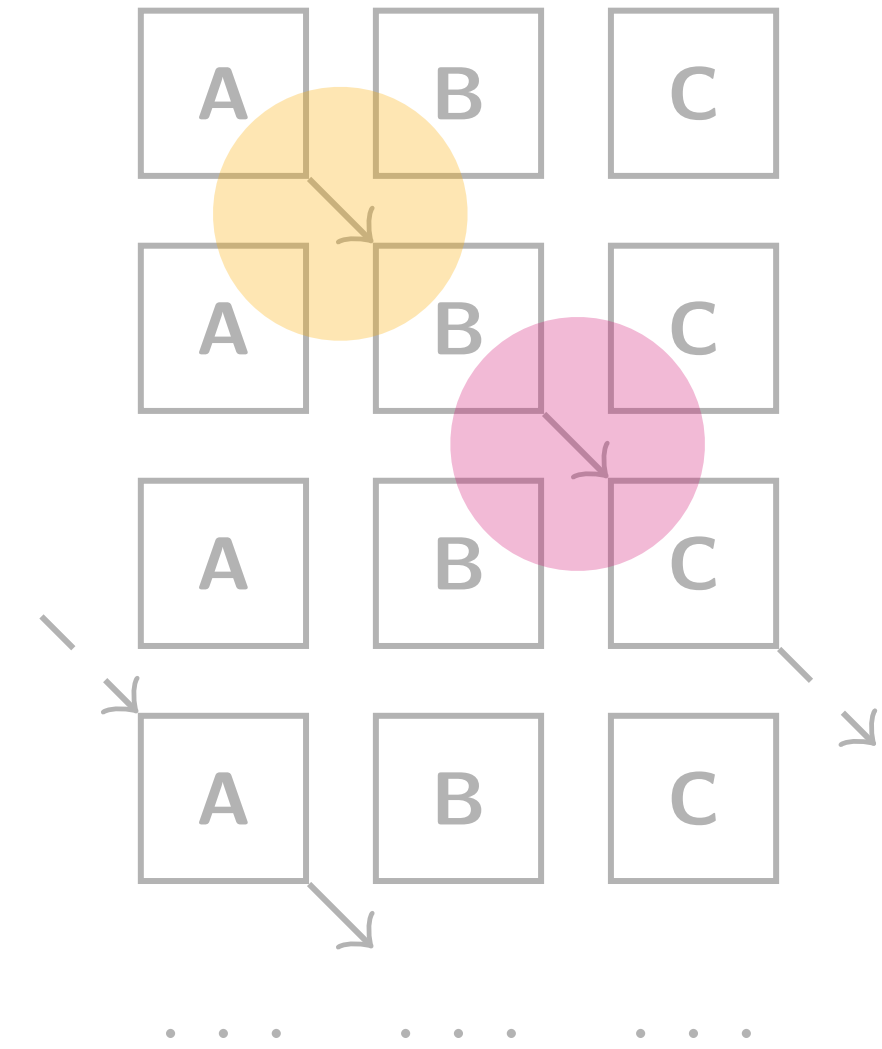
- Global types are inherently *synchronous*
  - Projection provides only one possible ordering



# Challenge

## Asynchronous Orderings

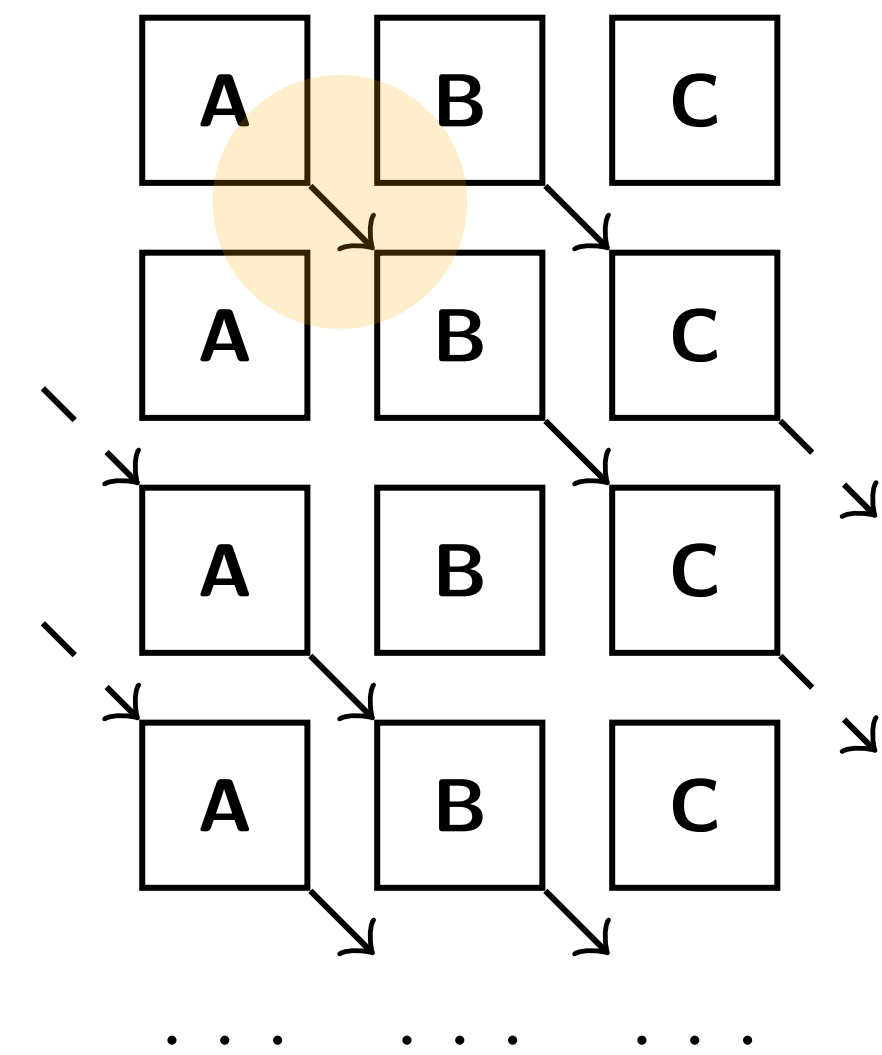
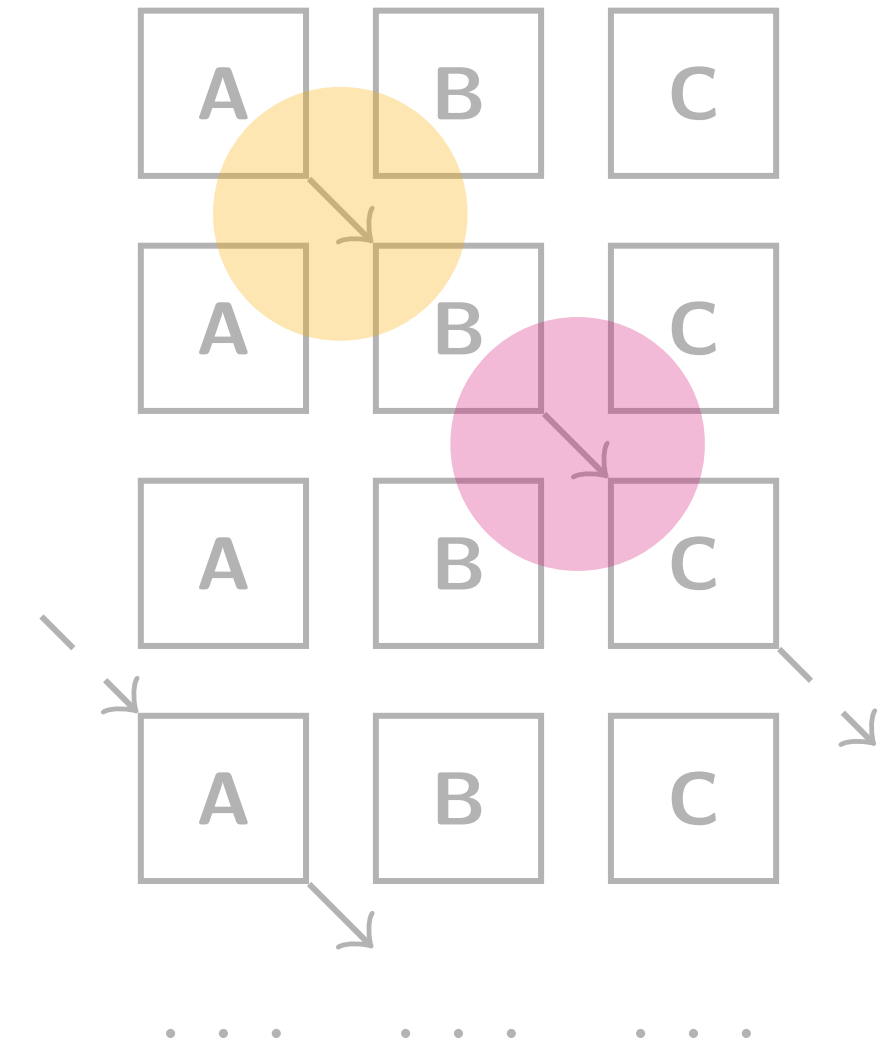
- Global types are inherently **synchronous**
  - Projection provides only one possible ordering
- Interactions can be **reordered** for efficiency while preserving safety



# Challenge

## Asynchronous Orderings

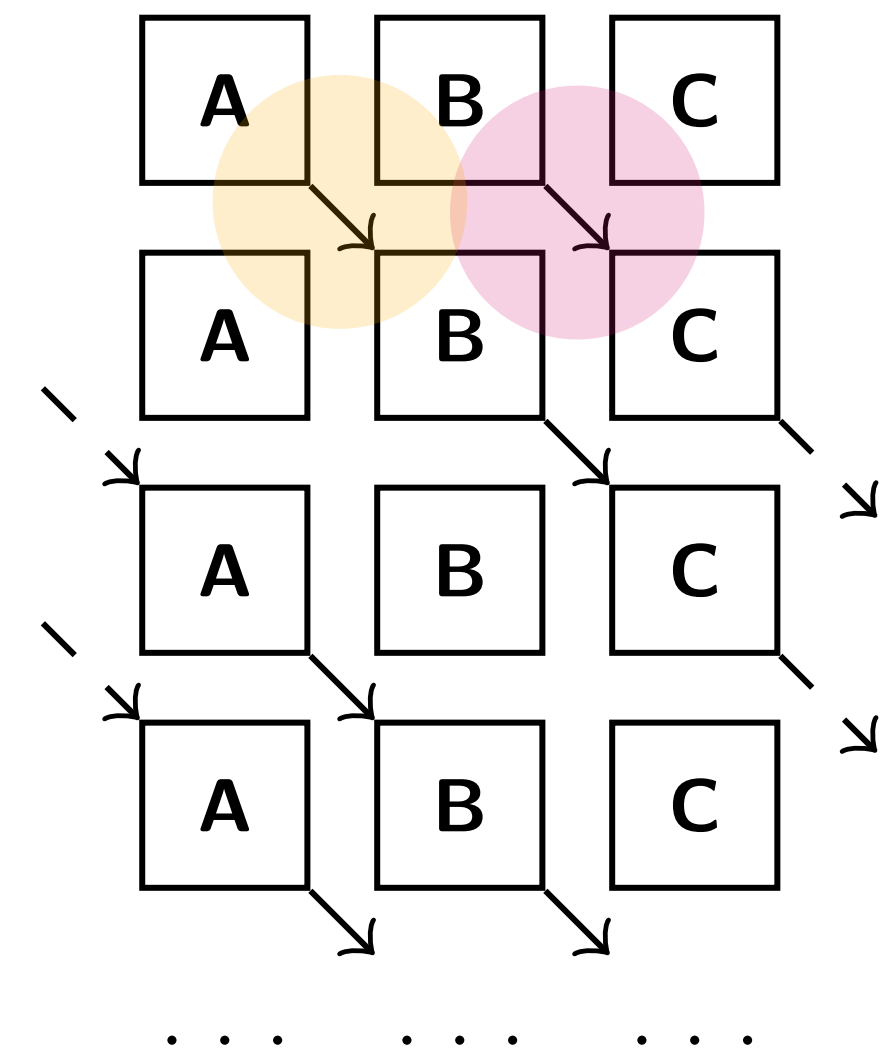
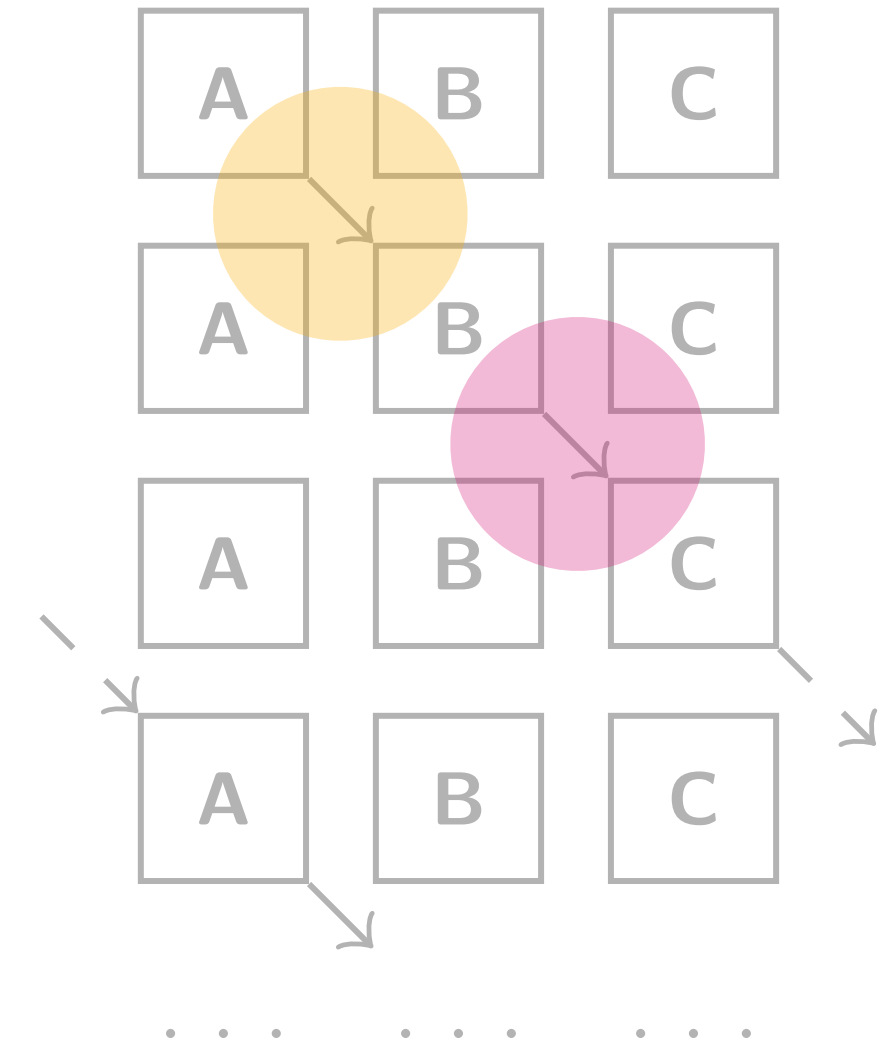
- Global types are inherently **synchronous**
  - Projection provides only one possible ordering
- Interactions can be **reordered** for efficiency while preserving safety



# Challenge

## Asynchronous Orderings

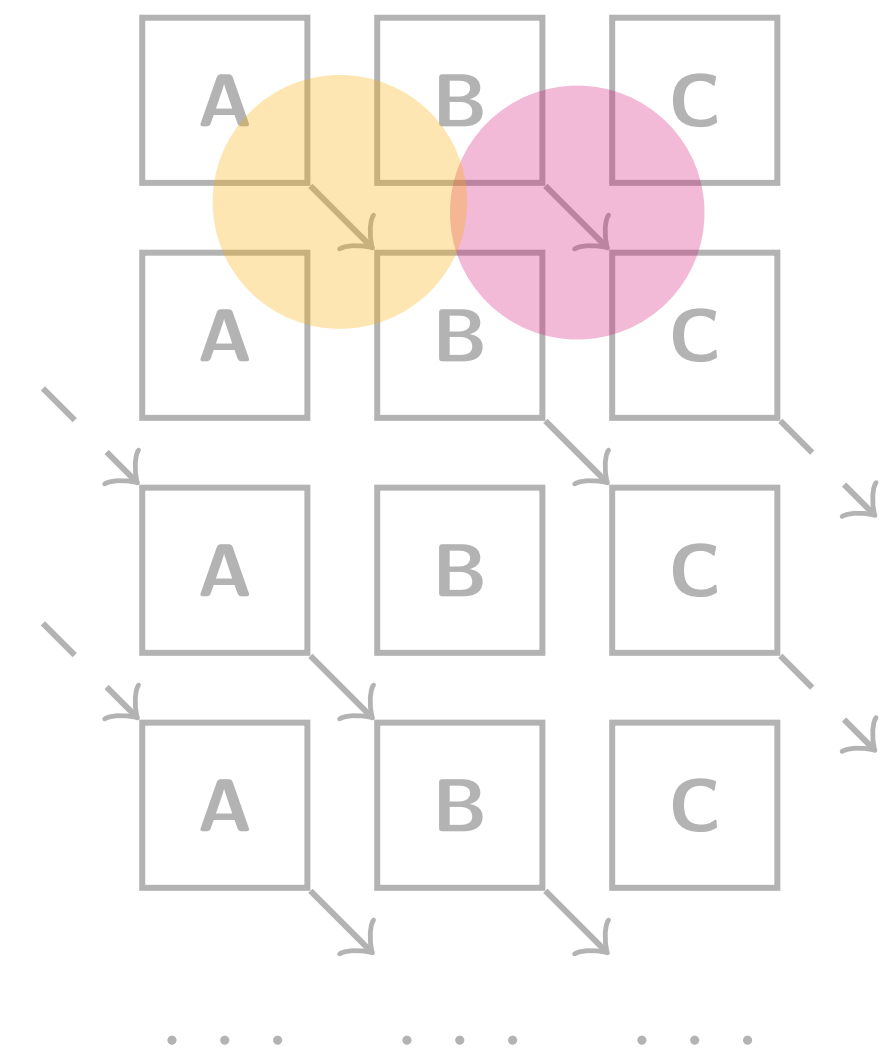
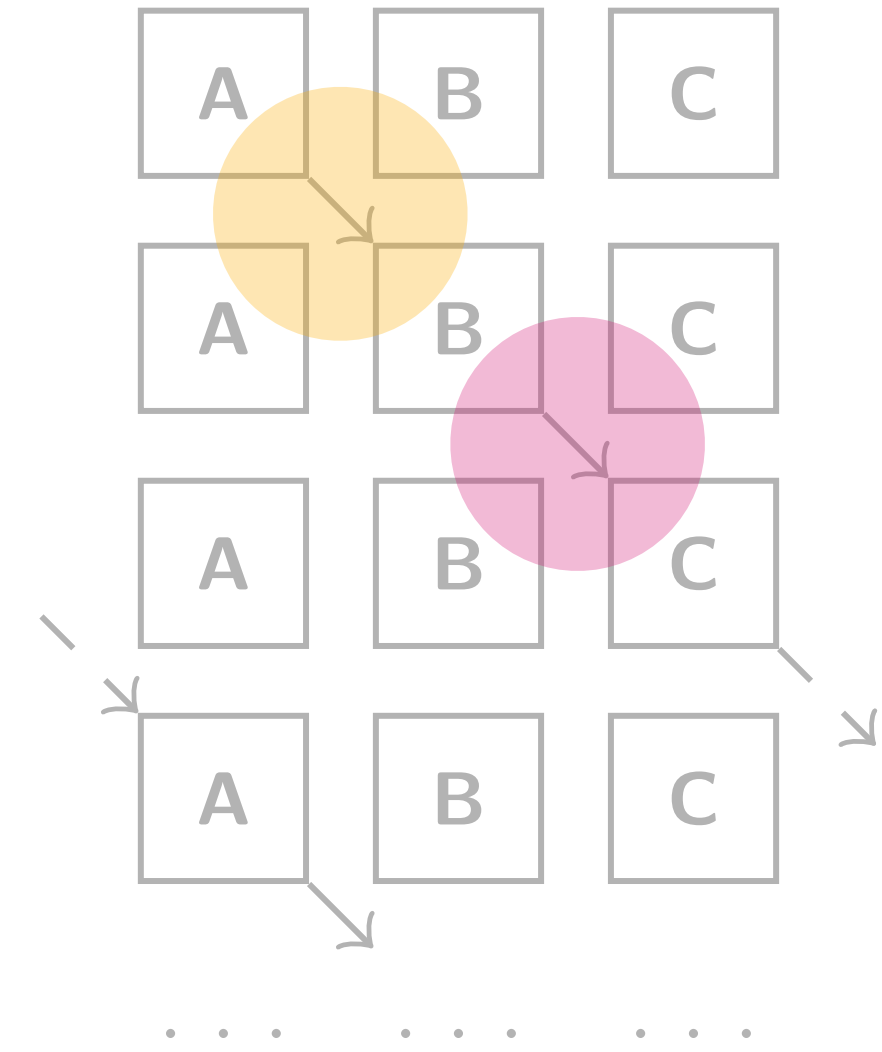
- Global types are inherently **synchronous**
  - Projection provides only one possible ordering
- Interactions can be **reordered** for efficiency while preserving safety



# Challenge

## Asynchronous Orderings

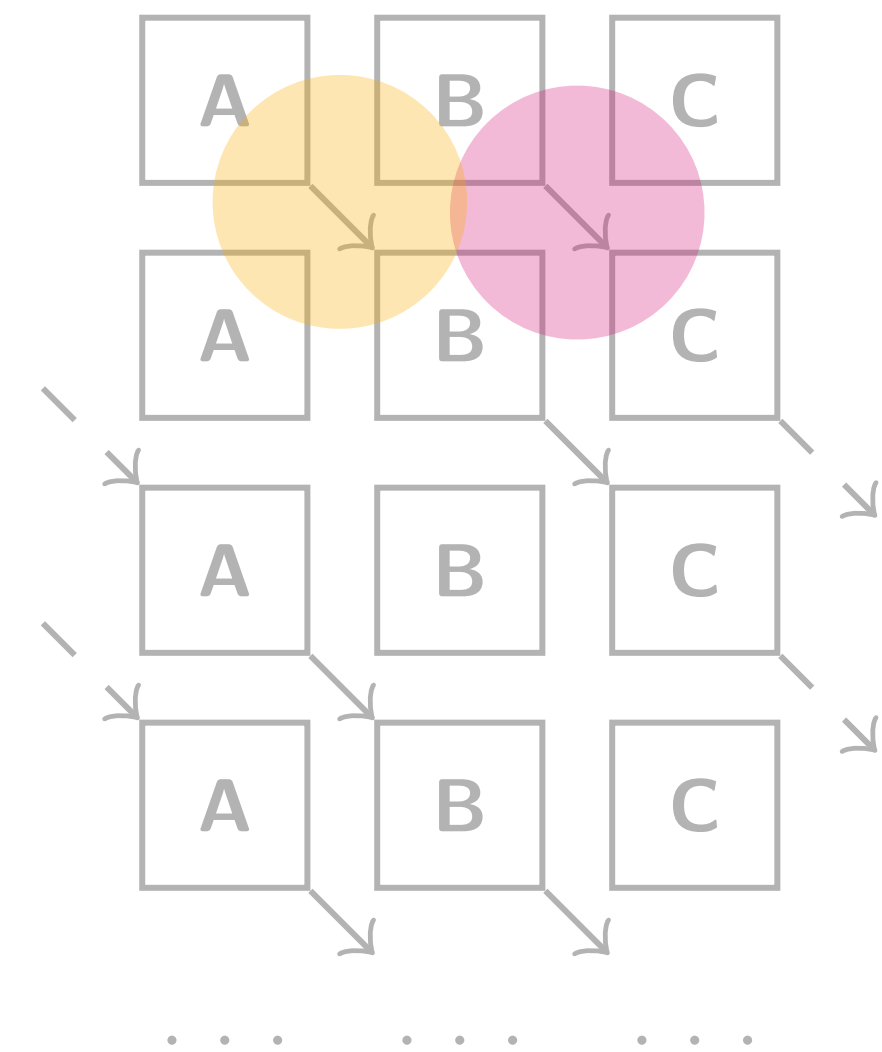
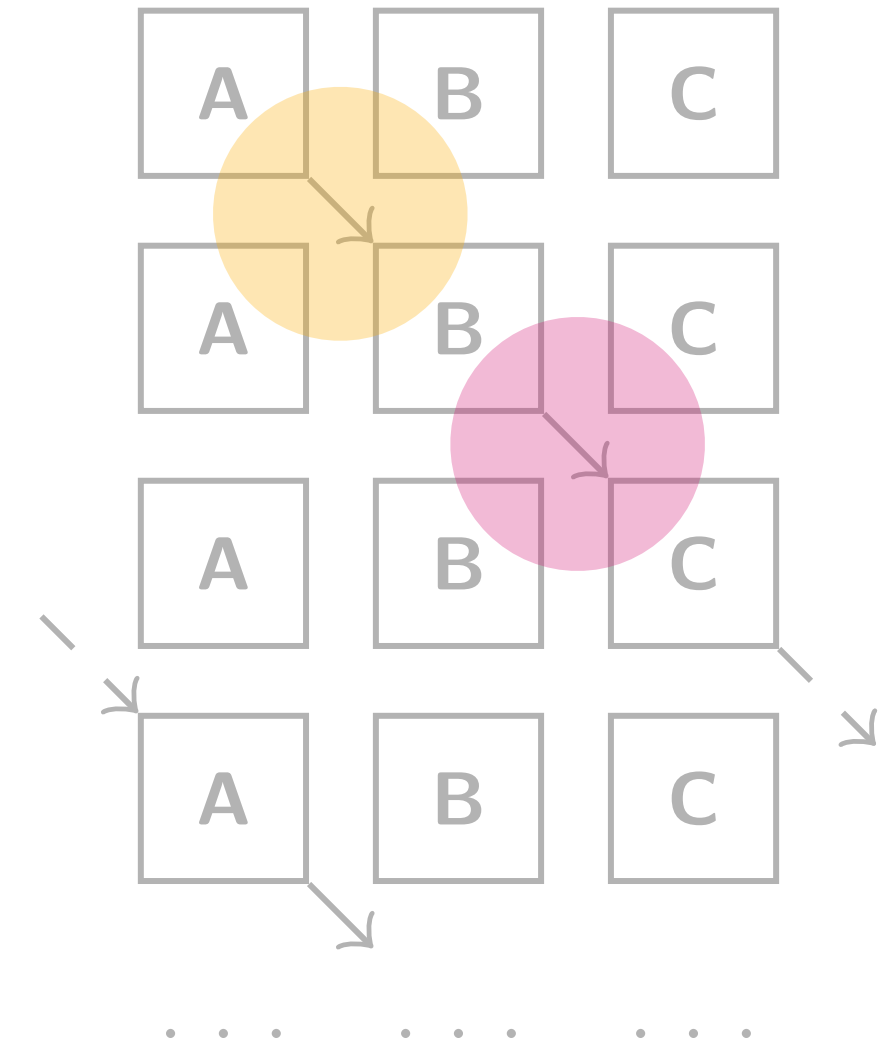
- Global types are inherently **synchronous**
  - Projection provides only one possible ordering
- Interactions can be **reordered** for efficiency while preserving safety
  1. Data **dependencies** must be preserved



# Challenge

## Asynchronous Orderings

- Global types are inherently **synchronous**
  - Projection provides only one possible ordering
- Interactions can be **reordered** for efficiency while preserving safety
  1. Data **dependencies** must be preserved
  2. **Sound** and **practical** asynchronous reordering rules must be found



# Rumpsteak Framework

## Three Approaches

**G** Global Type

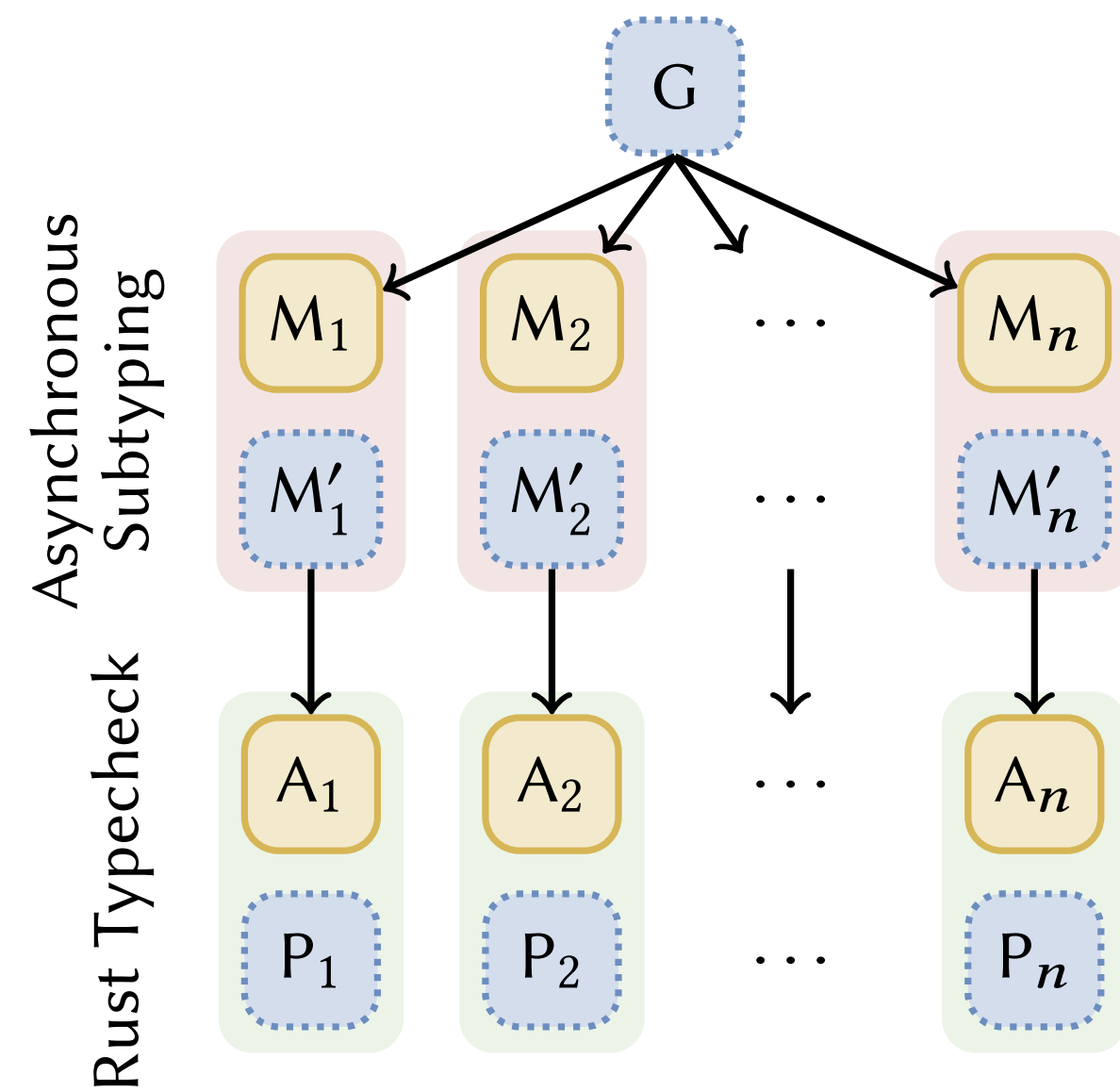
**M** Finite State Machine (FSM)

**M'** Optimised FSM

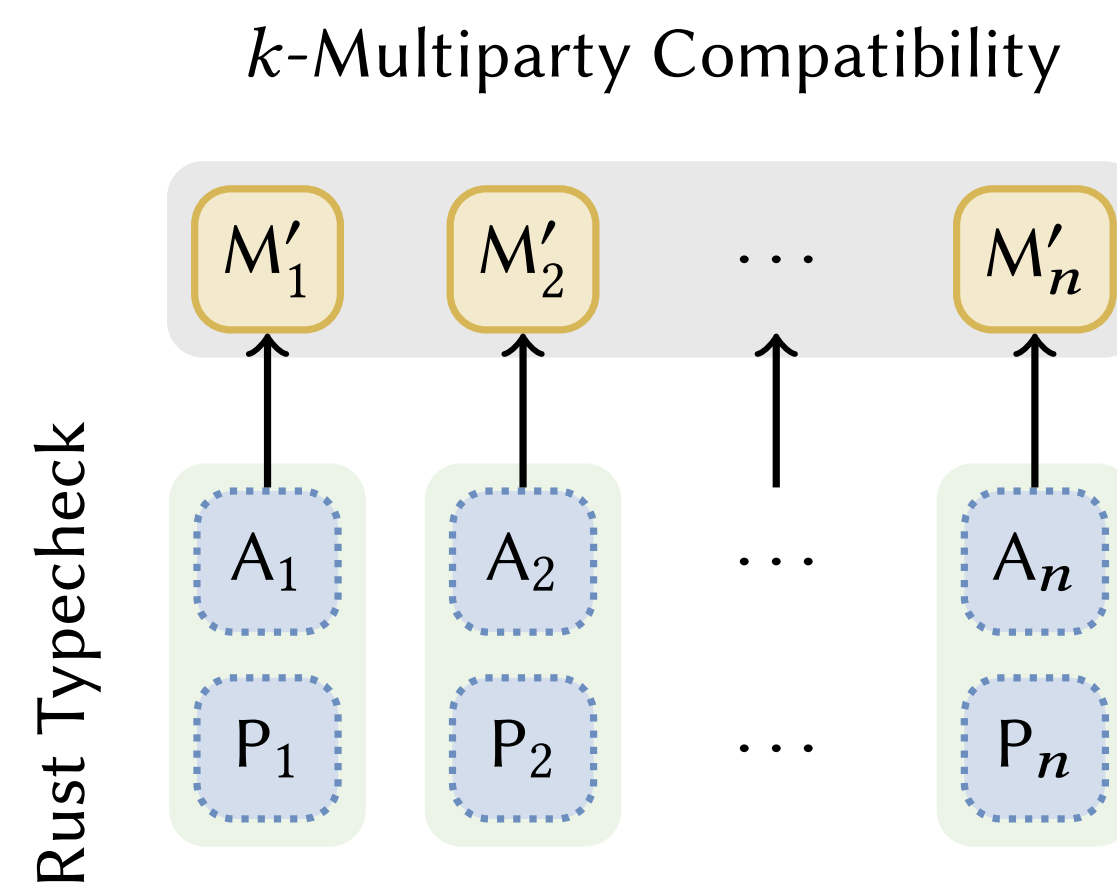
**A** Rust API

**P** Rust Process

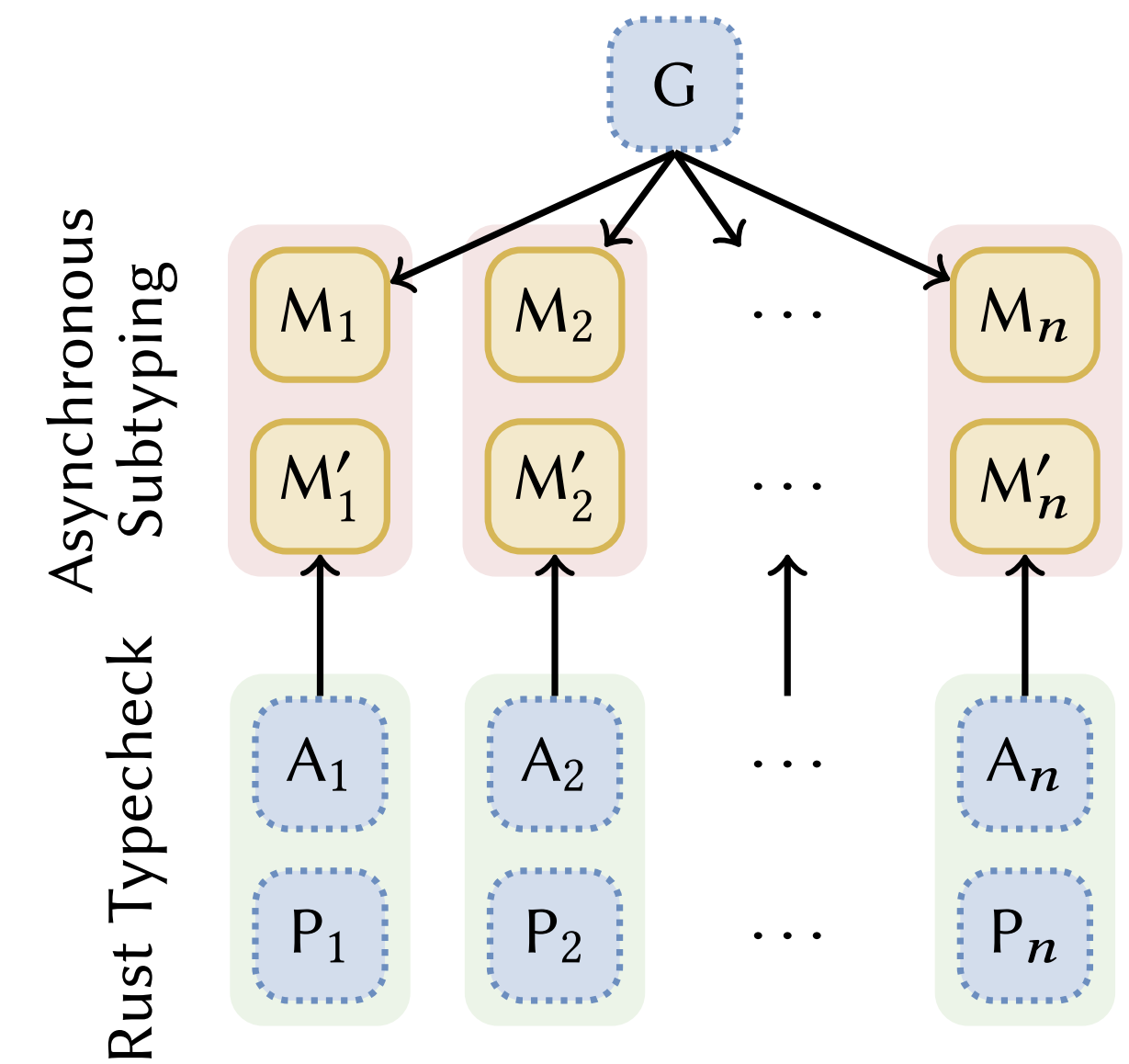
 User-Written     Generated



(a) Top-down



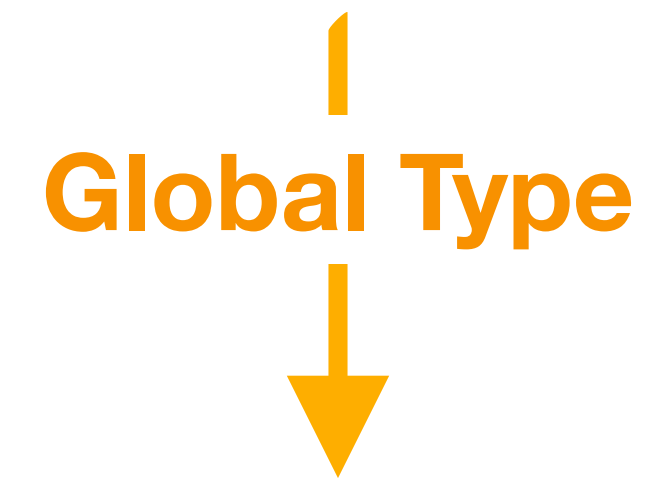
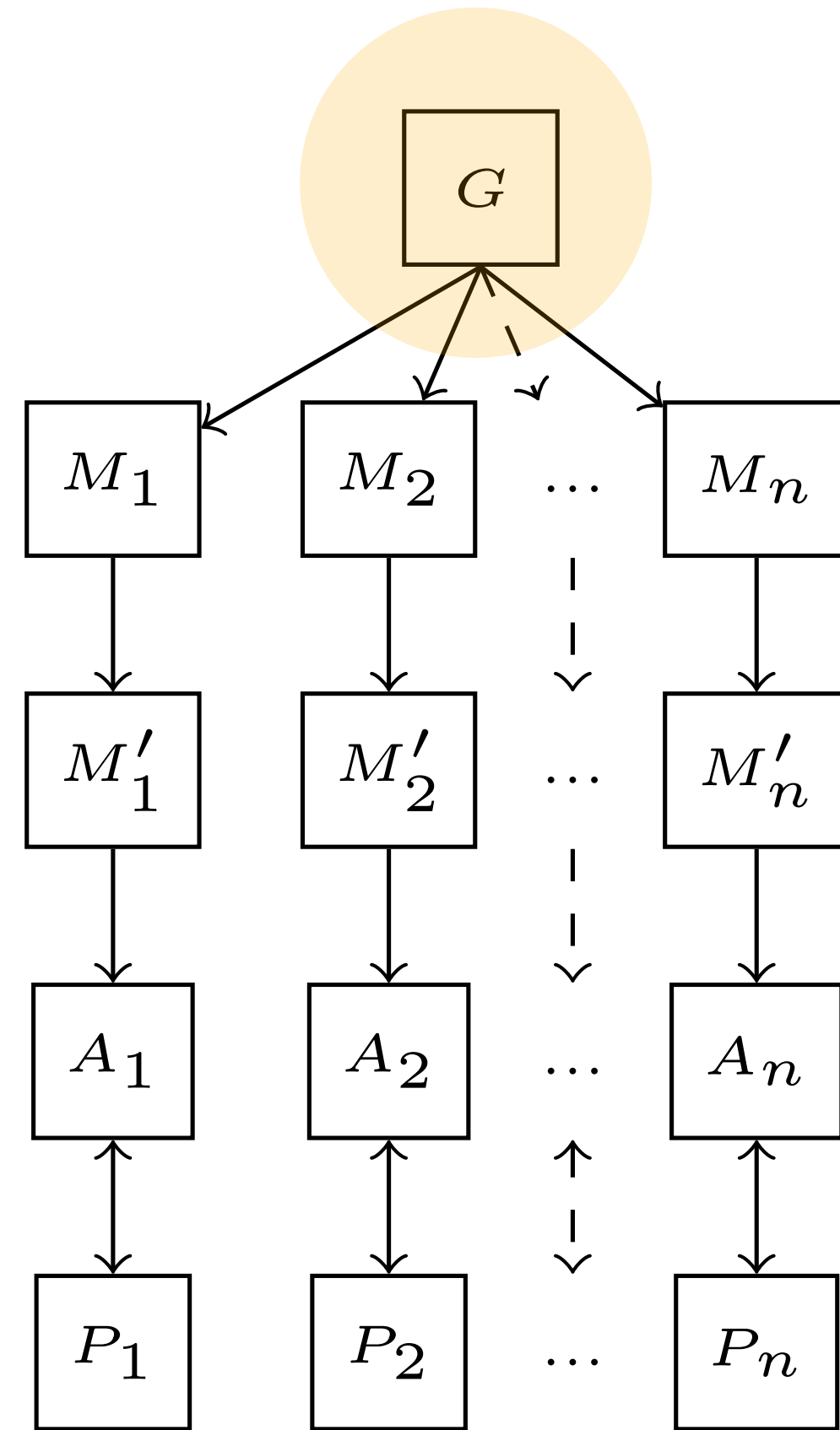
(b) Bottom-up



(c) Hybrid

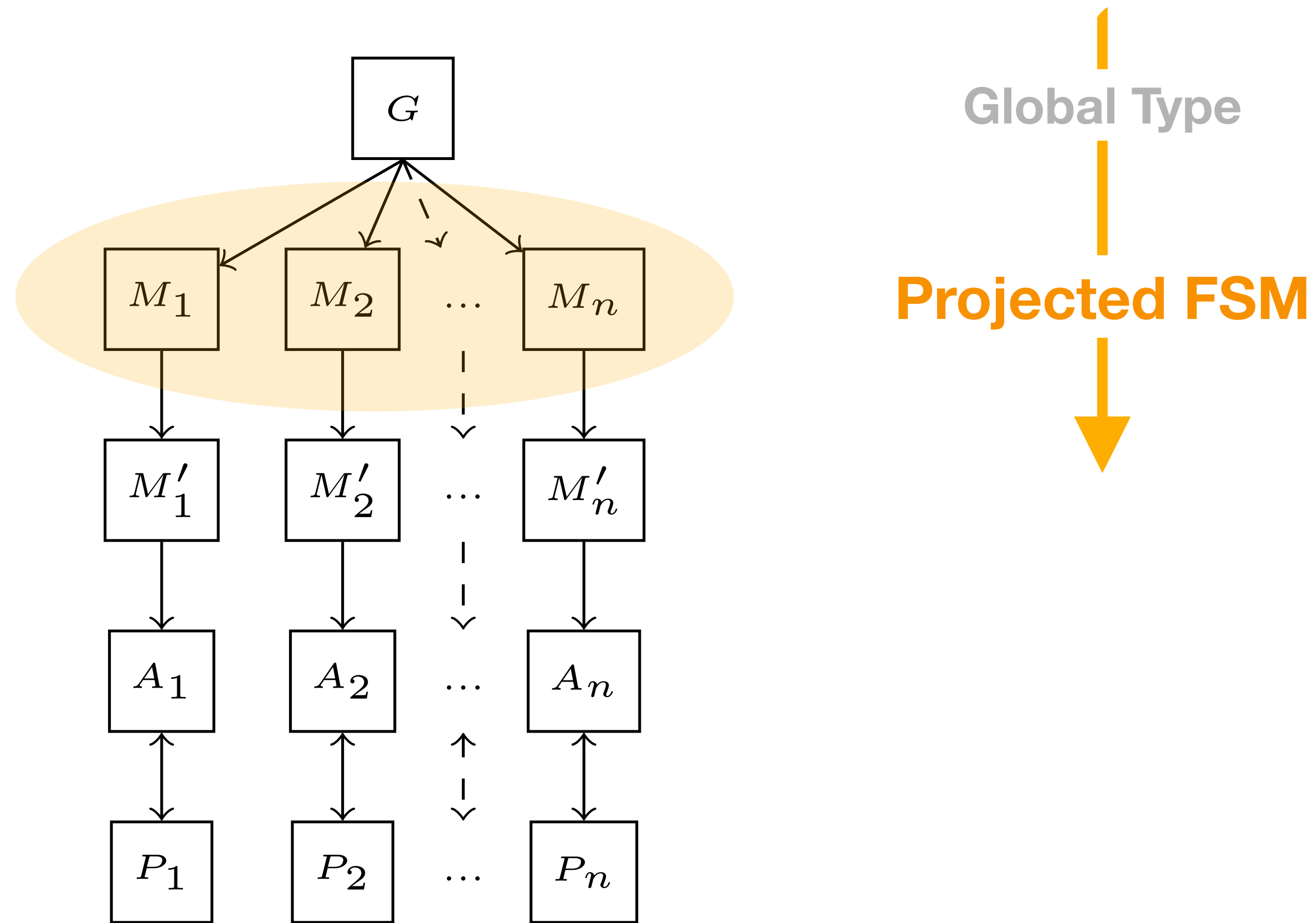
# Workflow

## Top-Down Approach



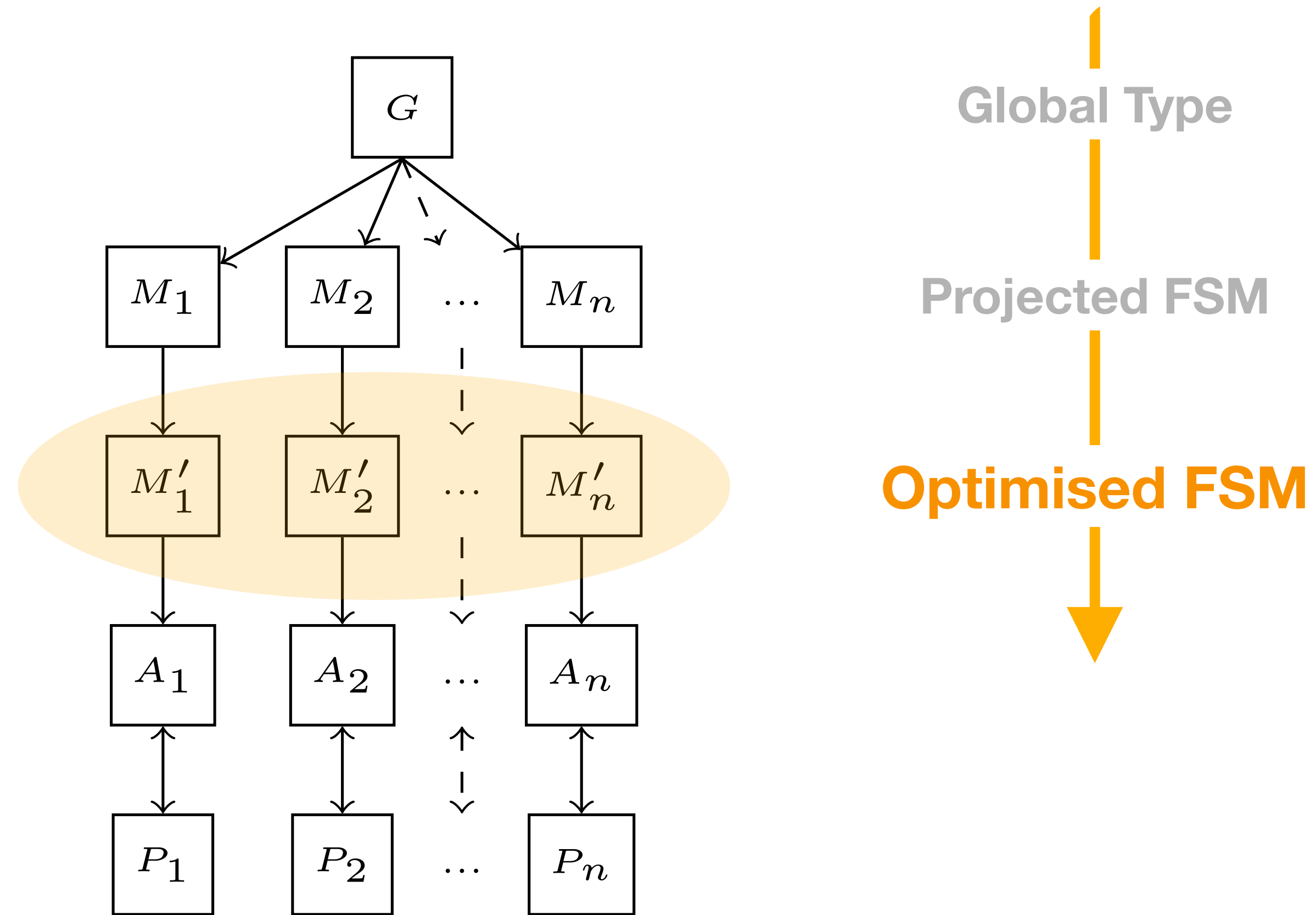
# Workflow

## Top-Down Approach



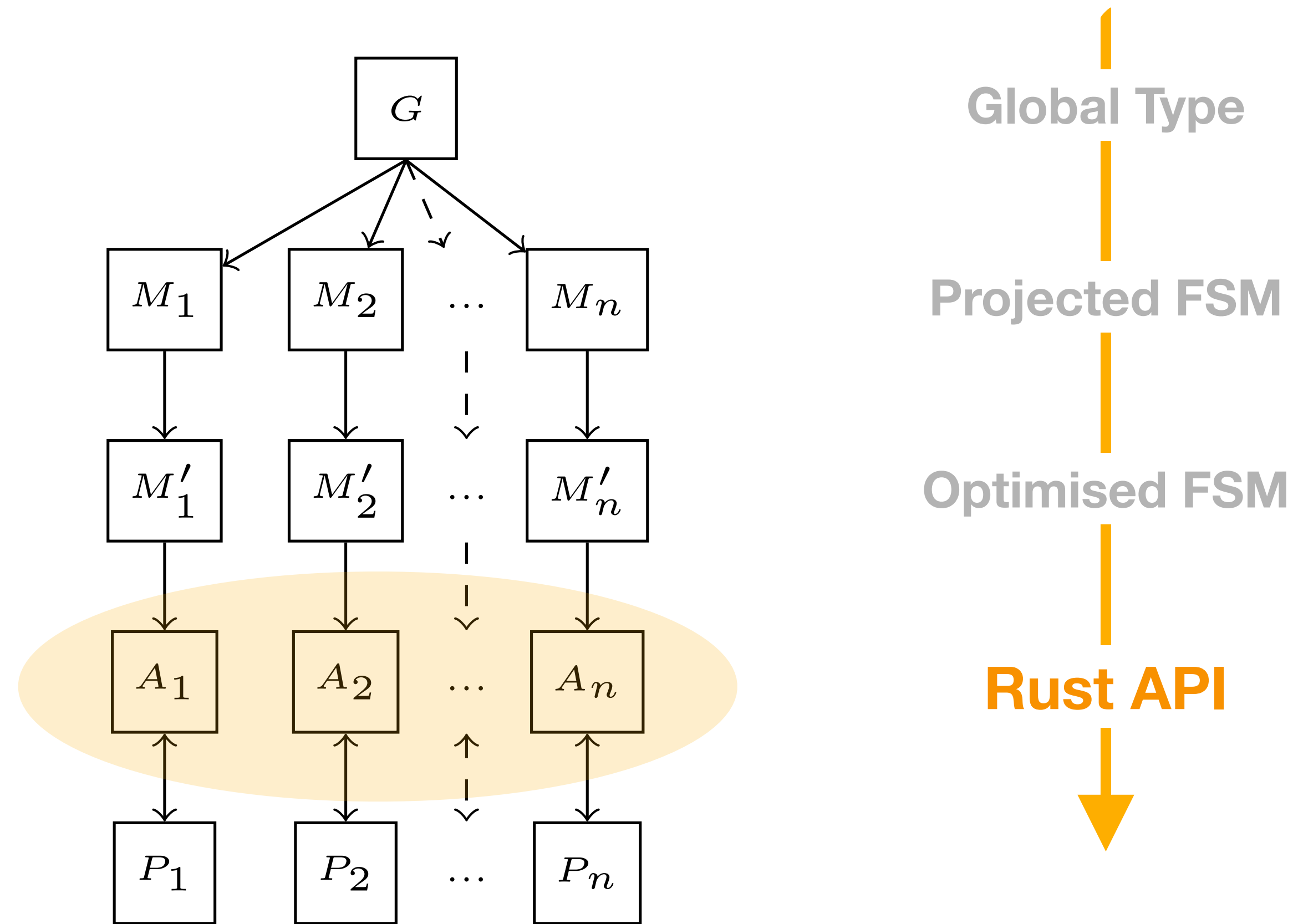
# Workflow

## Top-Down Approach



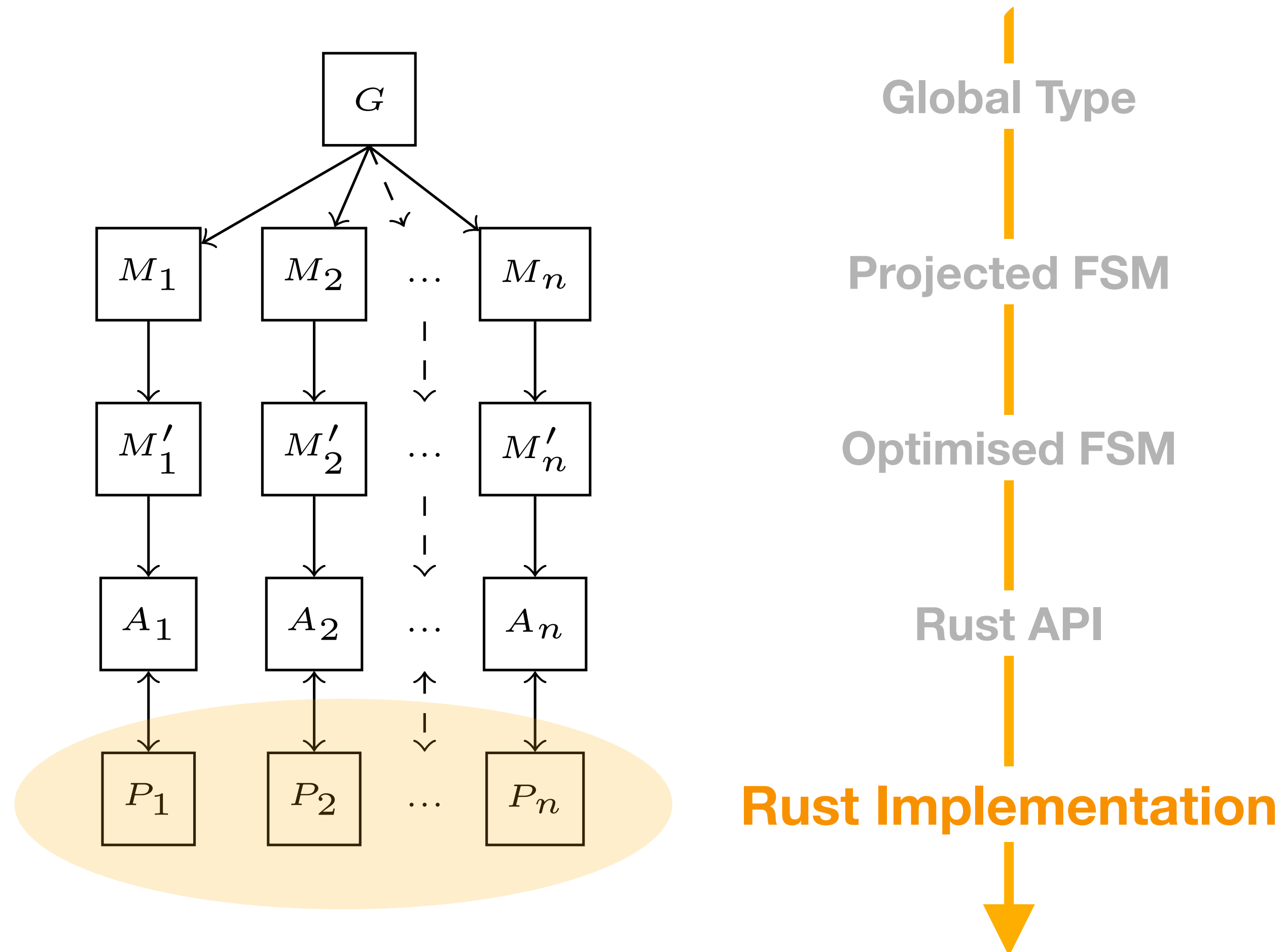
# Workflow

## Top-Down Approach



# Workflow

## Top-Down Approach



# Ring Protocol

## Example

### Global Type

$$G = \mu t. \mathbf{A} \rightarrow \mathbf{B} : \left\{ \begin{array}{l} \mathit{add}(\mathit{i32}).\mathbf{B} \rightarrow \mathbf{C} : \left\{ \begin{array}{l} \mathit{add}(\mathit{i32}).\mathbf{C} \rightarrow \mathbf{A} : \{\mathit{add}(\mathit{i32}).\mathbf{t}\} \\ \mathit{sub}(\mathit{i32}).\mathbf{C} \rightarrow \mathbf{A} : \{\mathit{sub}(\mathit{i32}).\mathbf{t}\} \end{array} \right\} \end{array} \right\}$$

# Ring Protocol

## Example

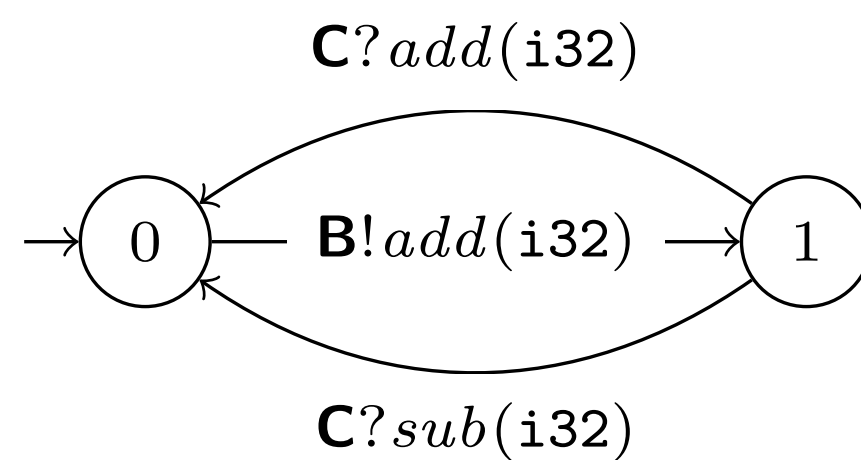
$$G = \mu t. \mathbf{A} \rightarrow \mathbf{B} : \left\{ \begin{array}{l} \mathit{add}(\mathit{i32}).\mathbf{B} \rightarrow \mathbf{C} : \left\{ \begin{array}{l} \mathit{add}(\mathit{i32}).\mathbf{C} \rightarrow \mathbf{A} : \{ \mathit{add}(\mathit{i32}).\mathbf{t} \} \\ \mathit{sub}(\mathit{i32}).\mathbf{C} \rightarrow \mathbf{A} : \{ \mathit{sub}(\mathit{i32}).\mathbf{t} \} \end{array} \right\} \end{array} \right\}$$

# Ring Protocol

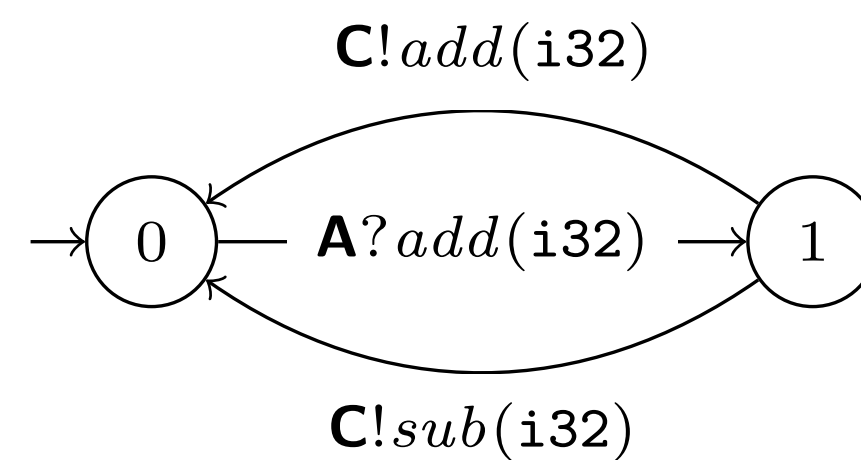
## Example

$$G = \mu t. \mathbf{A} \rightarrow \mathbf{B} : \left\{ \begin{array}{l} \mathit{add}(i32). \mathbf{B} \rightarrow \mathbf{C} : \left\{ \begin{array}{l} \mathit{add}(i32). \mathbf{C} \rightarrow \mathbf{A} : \{ \mathit{add}(i32). t \} \\ \mathit{sub}(i32). \mathbf{C} \rightarrow \mathbf{A} : \{ \mathit{sub}(i32). t \} \end{array} \right\} \end{array} \right\}$$

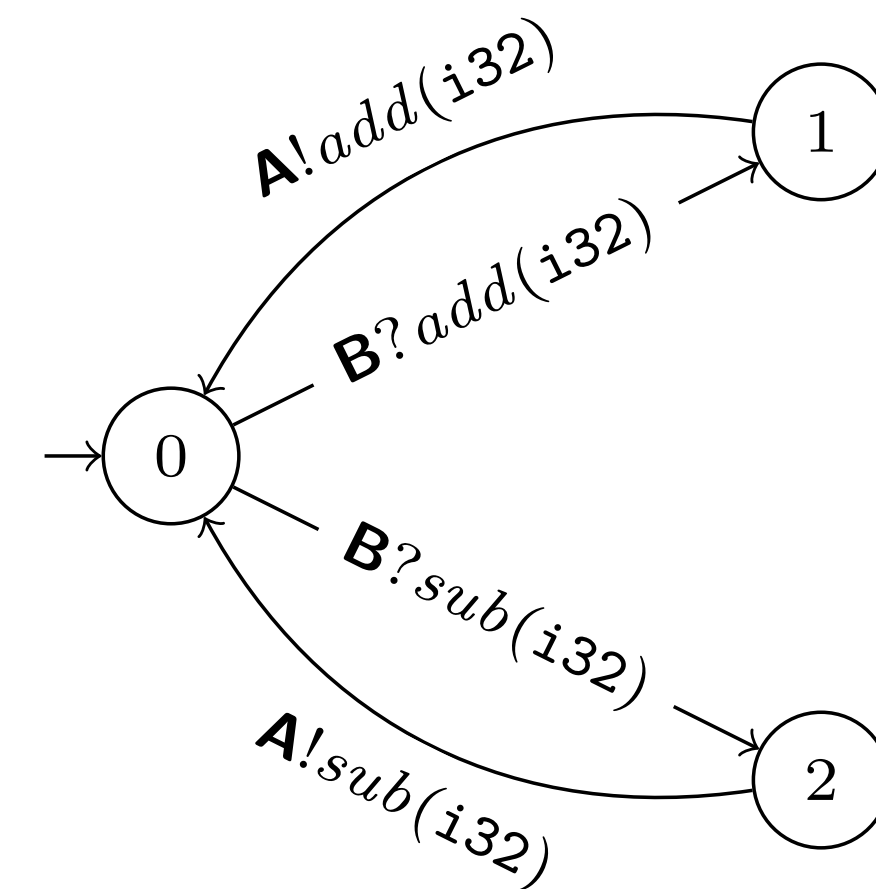
PROJECTION



PROJECTION

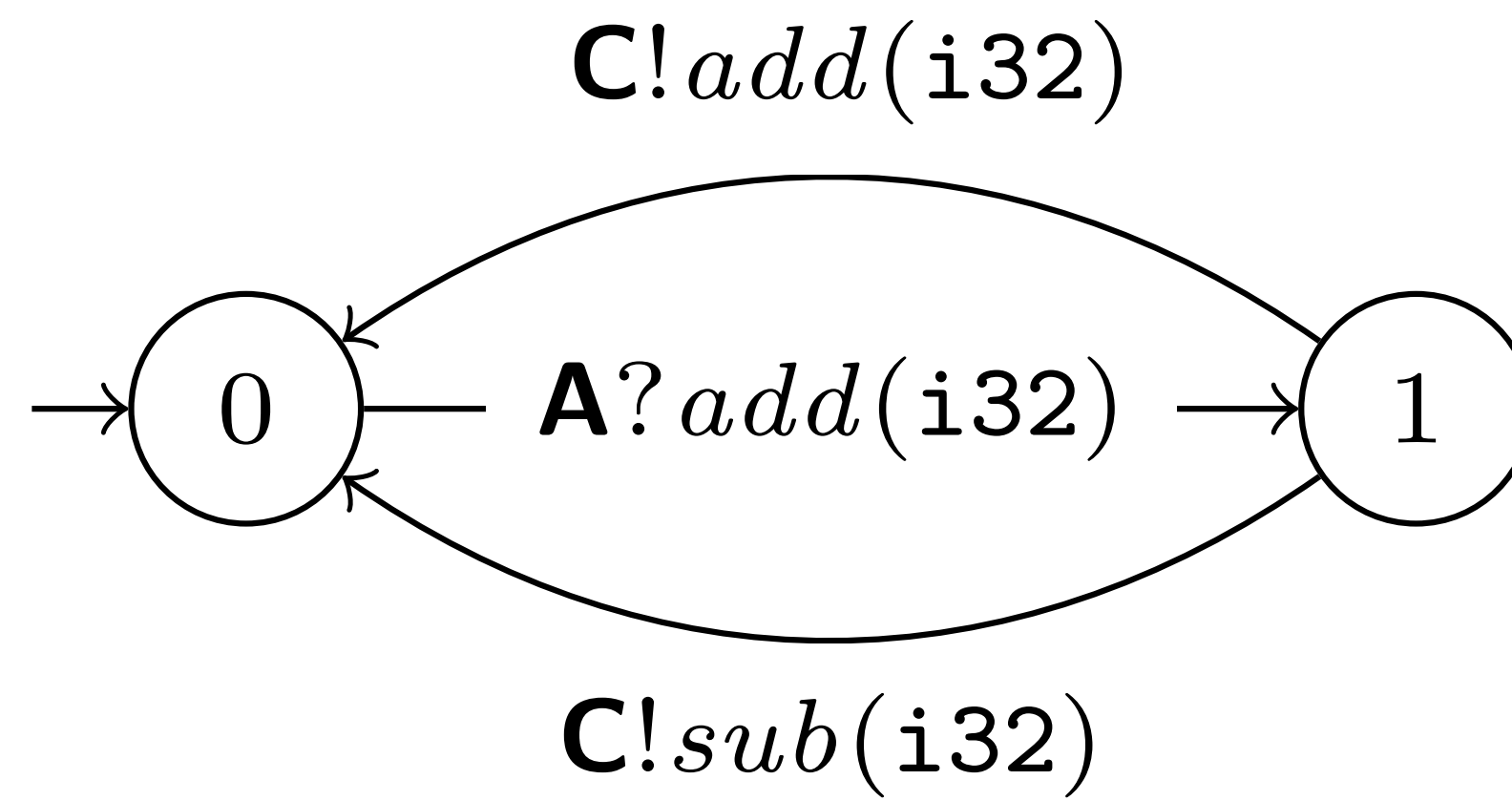


PROJECTION



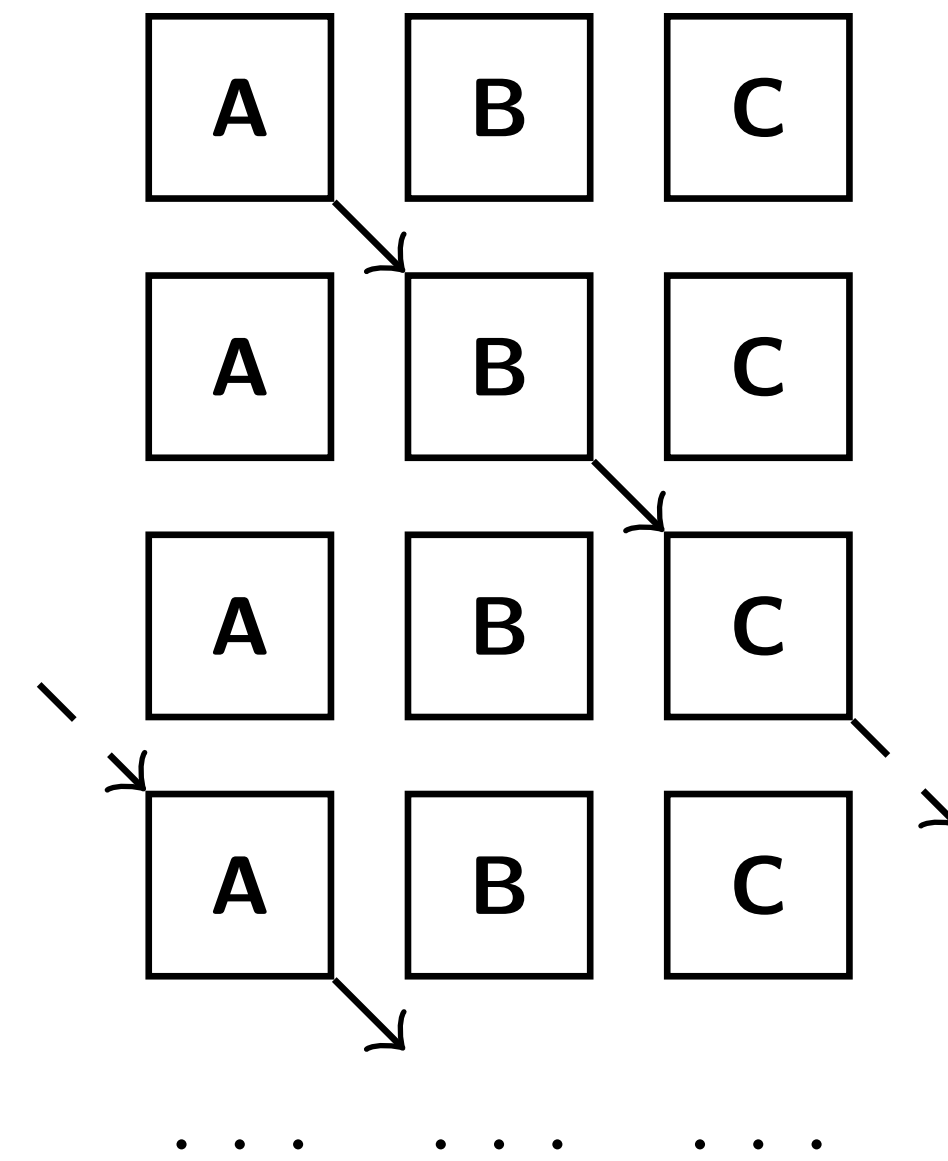
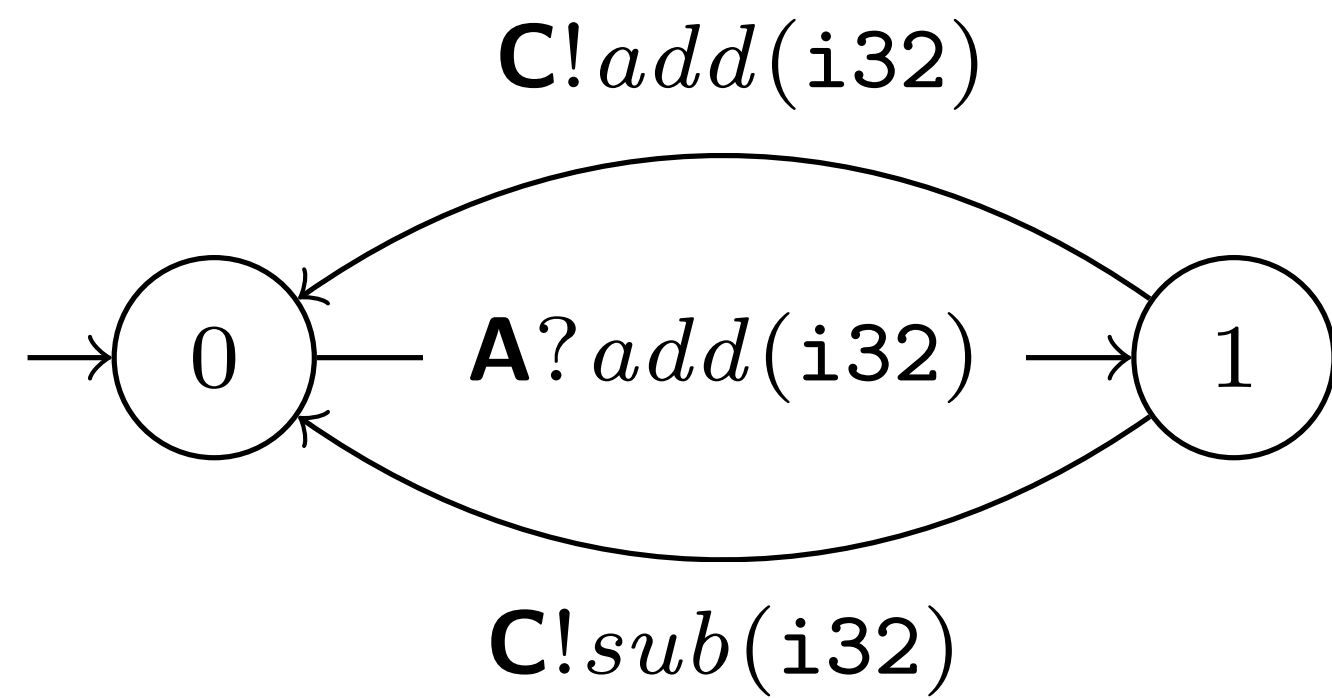
# Ring Protocol

## Example



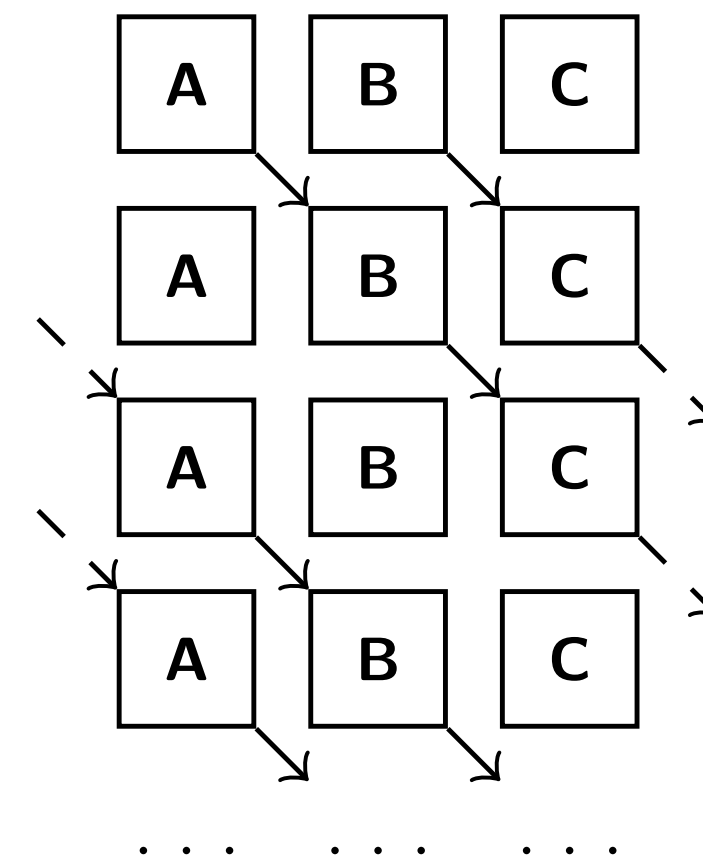
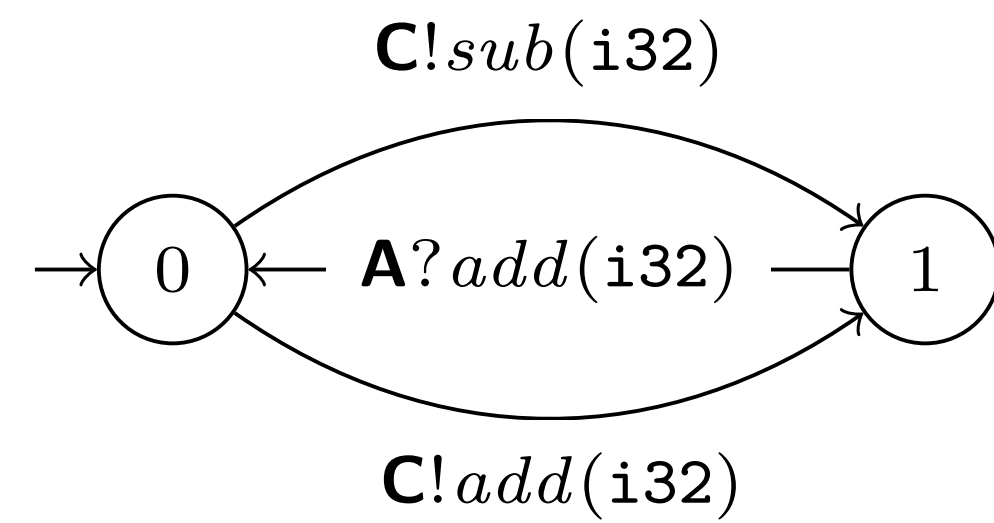
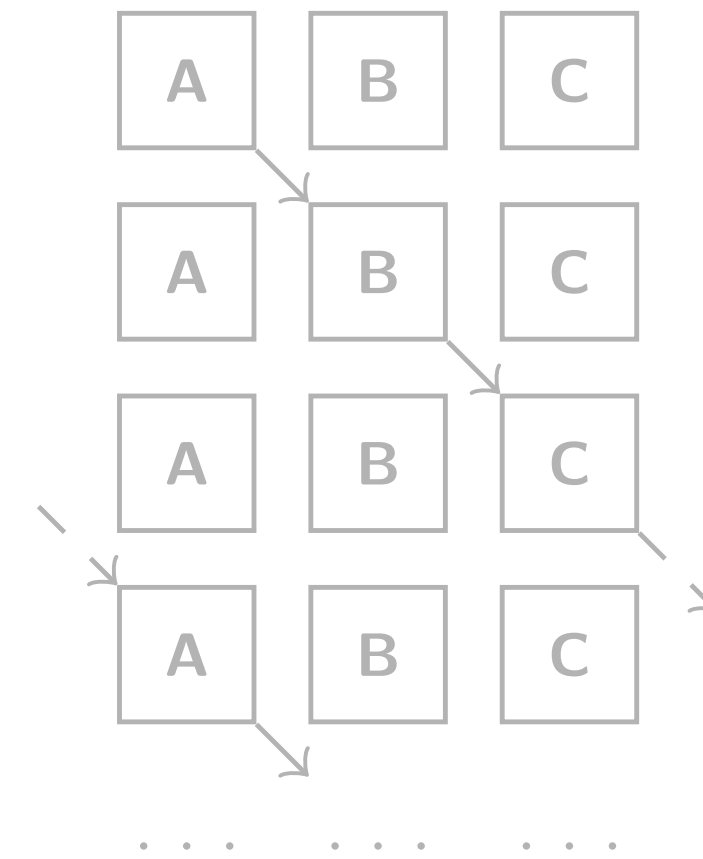
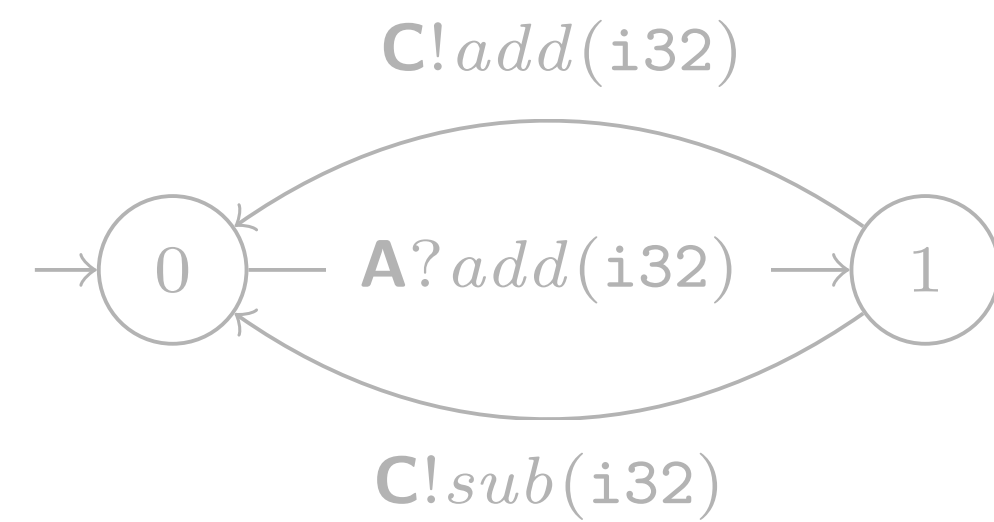
# Ring Protocol

## Example



# Ring Protocol

## Example



# vScr An Extensible Toolchain for Multiparty Session Types

- It's small and easy to modify
- Available on opam
  - [opam install nuscr](#)
- Available on GitHub
  - <https://github.com/nuscr>
- Available on the web
  - <https://nuscr.dev>

The screenshot shows the vScr live web interface. The browser address bar displays <https://nuscr.github.io/nuscr/>. The page features a navigation bar with 'vScr', 'Documentation', and 'GitHub' links. The main content is divided into two sections: 'Global protocol' and 'Local types'.

**Global protocol**

```
module Adder;  
type <java> "java.lang.Integer" from "rt.jar" as int;  
global protocol Adder(role C, role S)  
{  
  rec Loop {  
    HELLO(u:int) from C to S;  
    choice at C  
    {  
      ADD(w:int) from C to S;  
      ADD(v:int) from C to S;  
      RES(f:int) from S to C;  
      continue Loop;  
    }  
    or  
    {  
      BYE() from C to S;  
      BYE() from S to C;  
    }  
  }  
}
```

**Local types**

- Adder@C[Project][FSM]
- Adder@S[Project][FSM]

The local types section displays a state transition diagram with 8 states (1-8) and transitions labeled with session types. The transitions are:

- 1 to 2: S!HELLO(u: int)
- 2 to 7: S!BYE()
- 2 to 4: S!ADD(w: int)
- 4 to 5: S!ADD(v: int)
- 5 to 1: S?RES(f: int)
- 7 to 8: S?BYE()

At the bottom of the interface, there is a 'Load an example' dropdown menu and an 'Analyse' button.

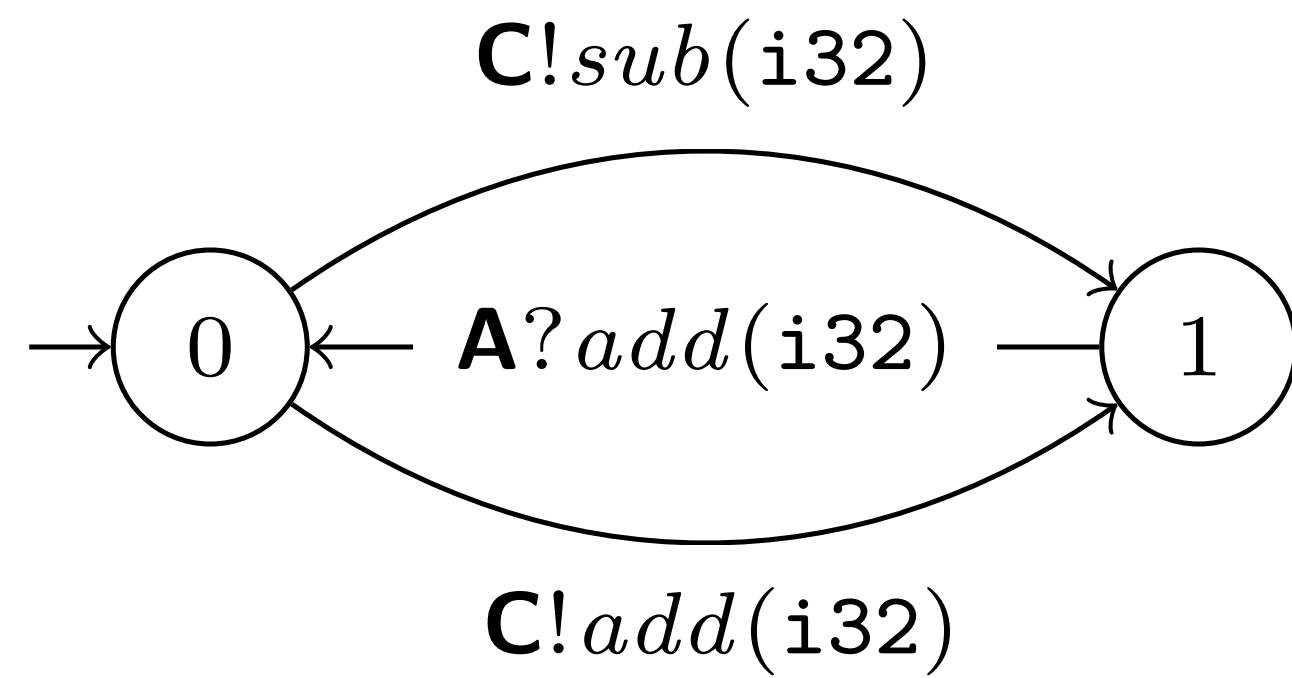
# Scribble

## Protocol Description Language

```
global protocol Ring(role A, role B, role C) {  
  Add(i32) from A to B;  
  choice at B {  
    Add(i32) from B to C;  
    Add(i32) from C to A;  
    do Ring(A, B, C);  
  } or {  
    Sub(i32) from B to C;  
    Sub(i32) from C to A;  
    do Ring(A, B, C);  
  }  
}
```

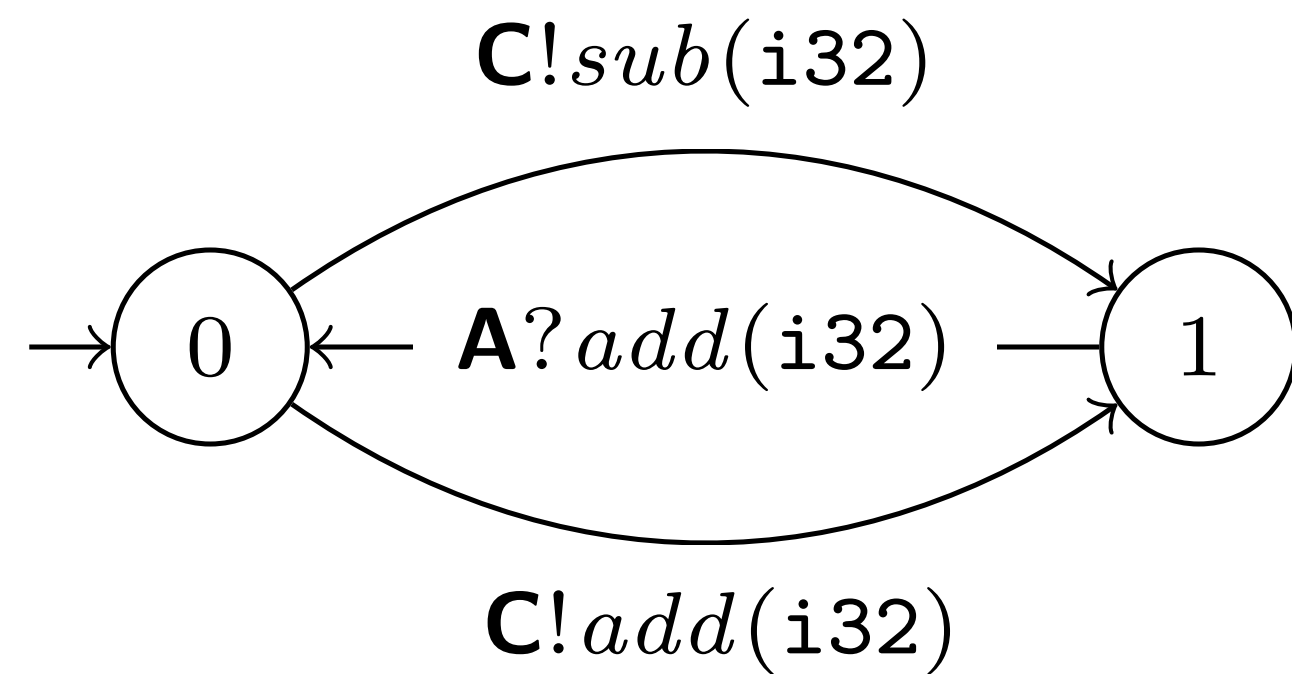
# Ring Protocol

## Rust API



# Ring Protocol

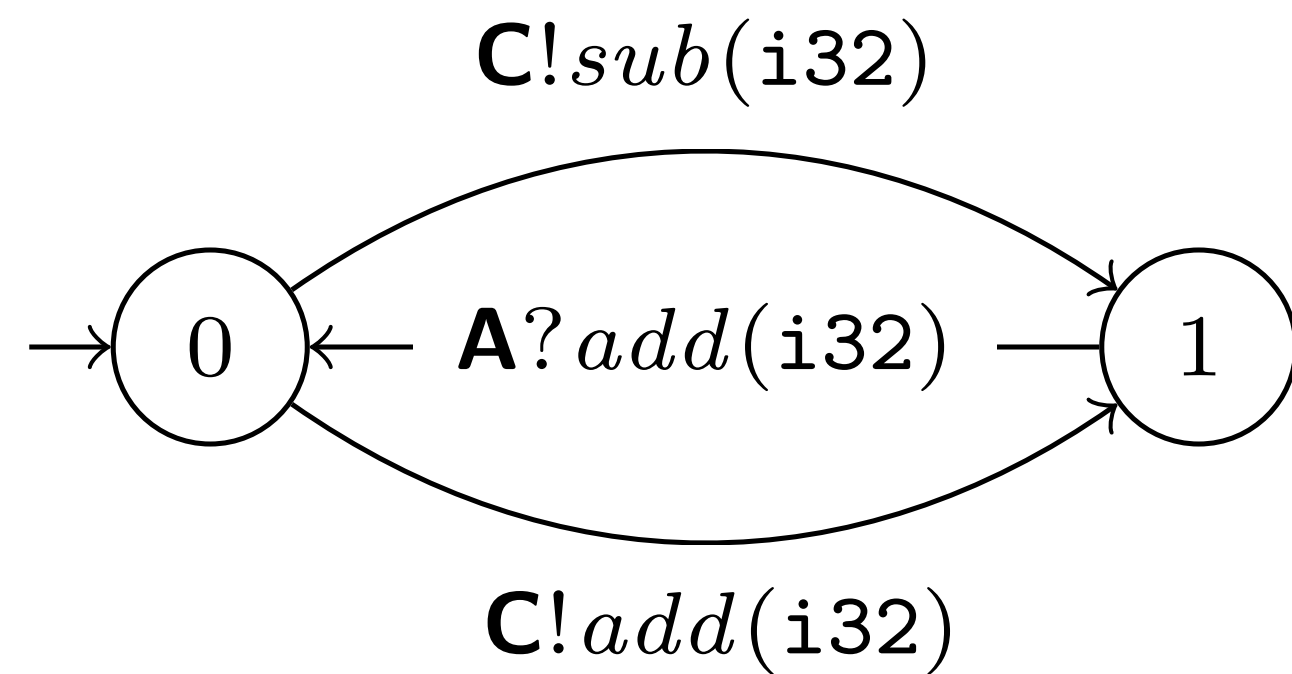
## Rust API



```
#[derive(Role)]  
#[message(Label)]  
struct B(#[route(A)] Receiver, #[route(C)] Sender);
```

# Ring Protocol

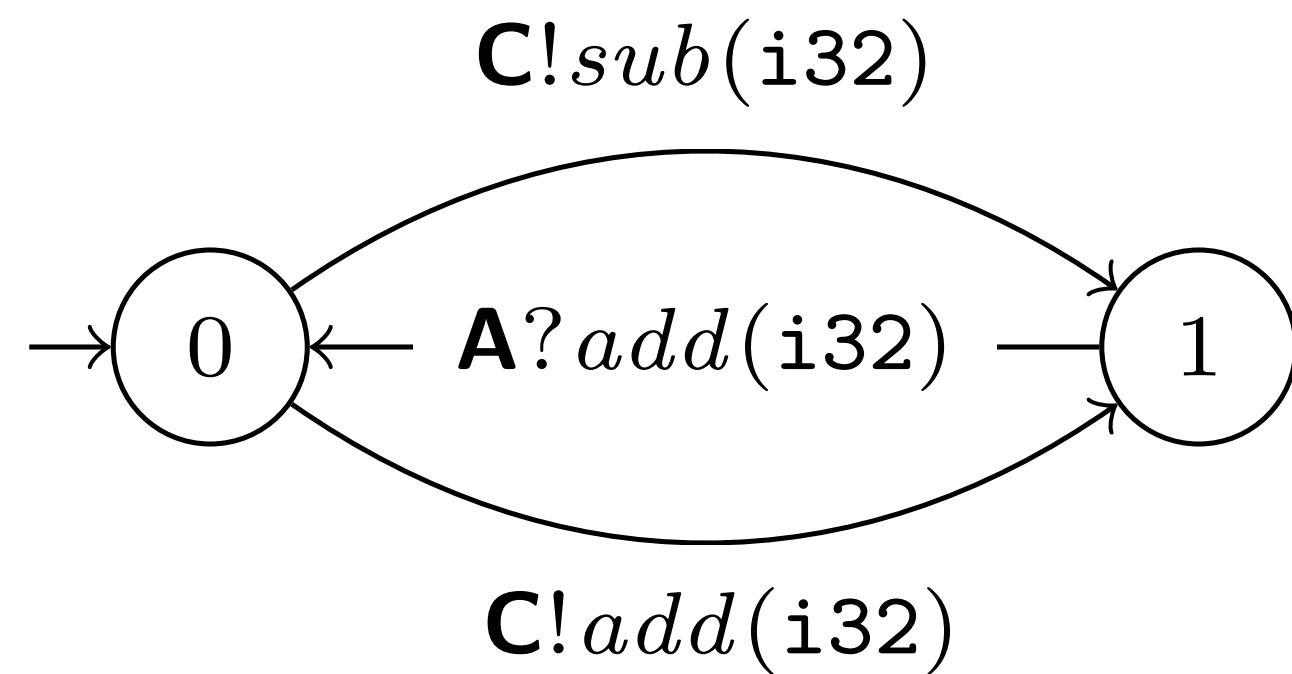
## Rust API



```
#[derive(Role)]  
#[message(Label)]  
struct B(#[route(A)] Receiver, #[route(C)] Sender);
```

# Ring Protocol

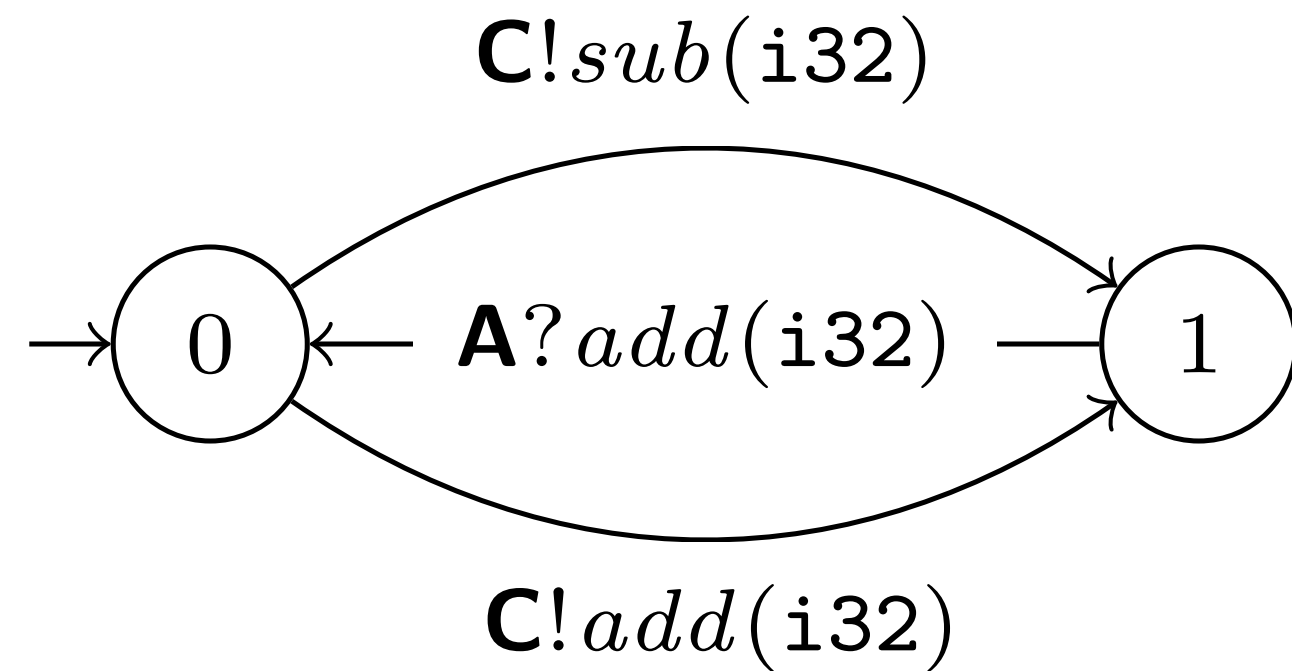
## Rust API



```
#[derive(Role)]  
#[message(Label)]  
struct B(#[route(A)] Receiver, #[route(C)] Sender);
```

# Ring Protocol

## Rust API



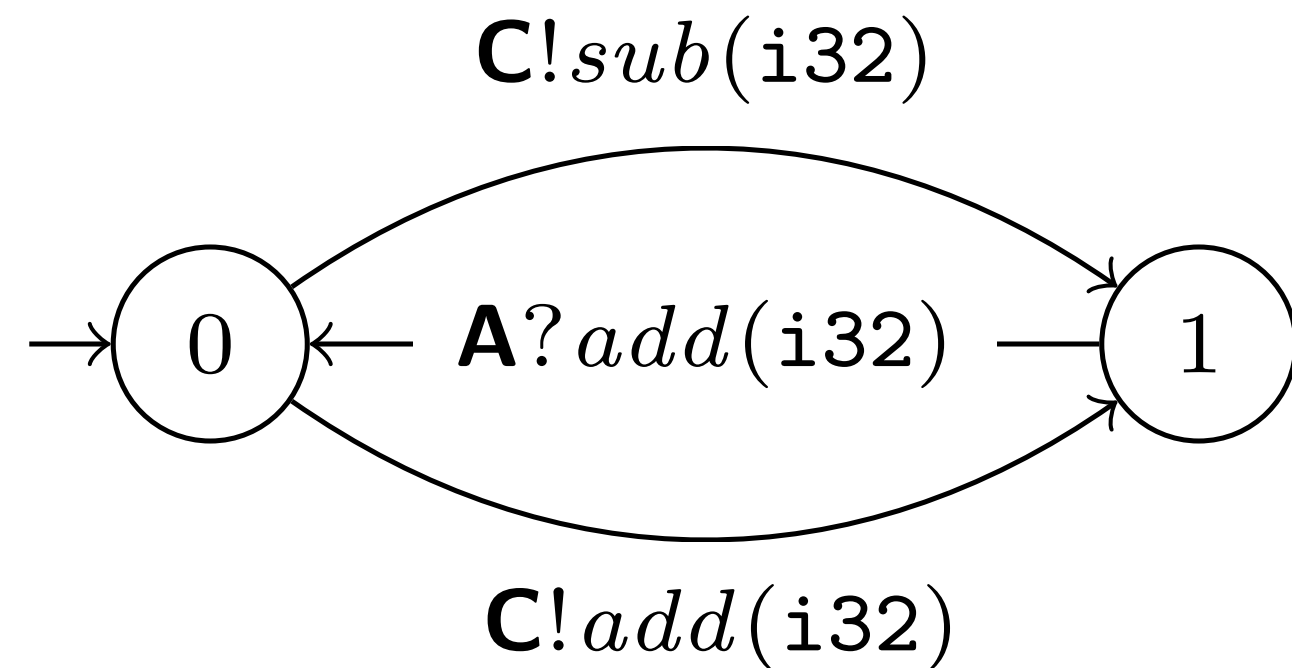
```
#[derive(Role)]  
#[message(Label)]  
struct B(#[route(A)] Receiver, #[route(C)] Sender);
```

```
#[derive(Message)]  
enum Label {  
    Add(Add),  
    Sub(Sub),  
}
```

```
struct Add(i32);  
struct Sub(i32);
```

# Ring Protocol

## Rust API



```
#[derive(Role)]
#[message(Label)]
struct B(#[route(A)] Receiver, #[route(C)] Sender);

#[derive(Message)]
enum Label {
    Add(Add),
    Sub(Sub),
}

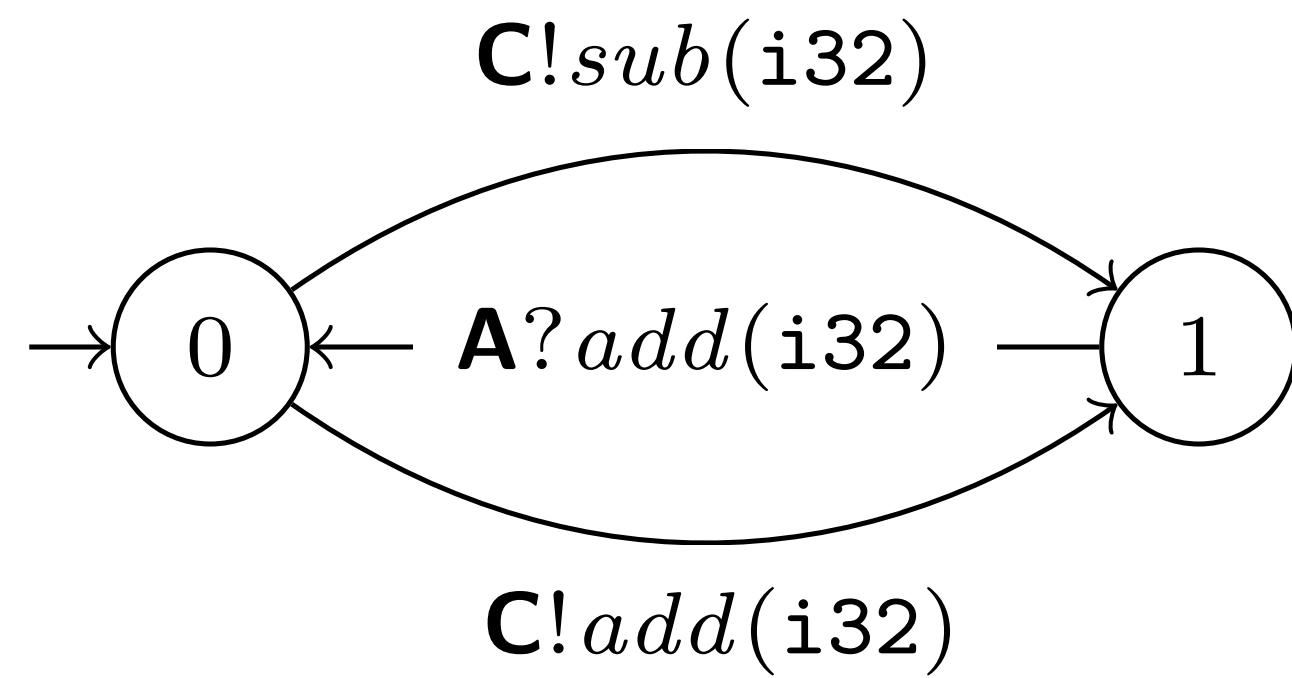
struct Add(i32);
struct Sub(i32);

#[session]
type RingB = Select<C, RingBChoice>;

#[session]
enum RingBChoice {
    Add(Add, Receive<A, Add, RingB>),
    Sub(Sub, Receive<A, Add, RingB>),
}
```

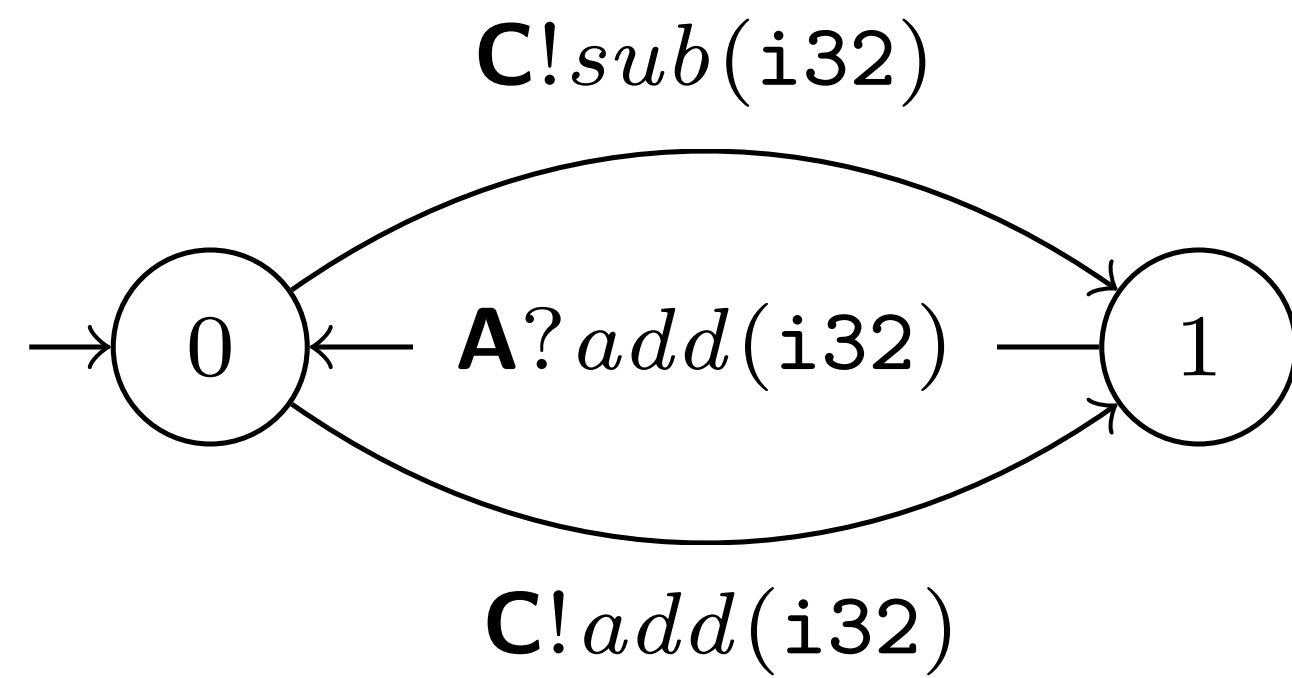
# Ring Protocol

## Rust API



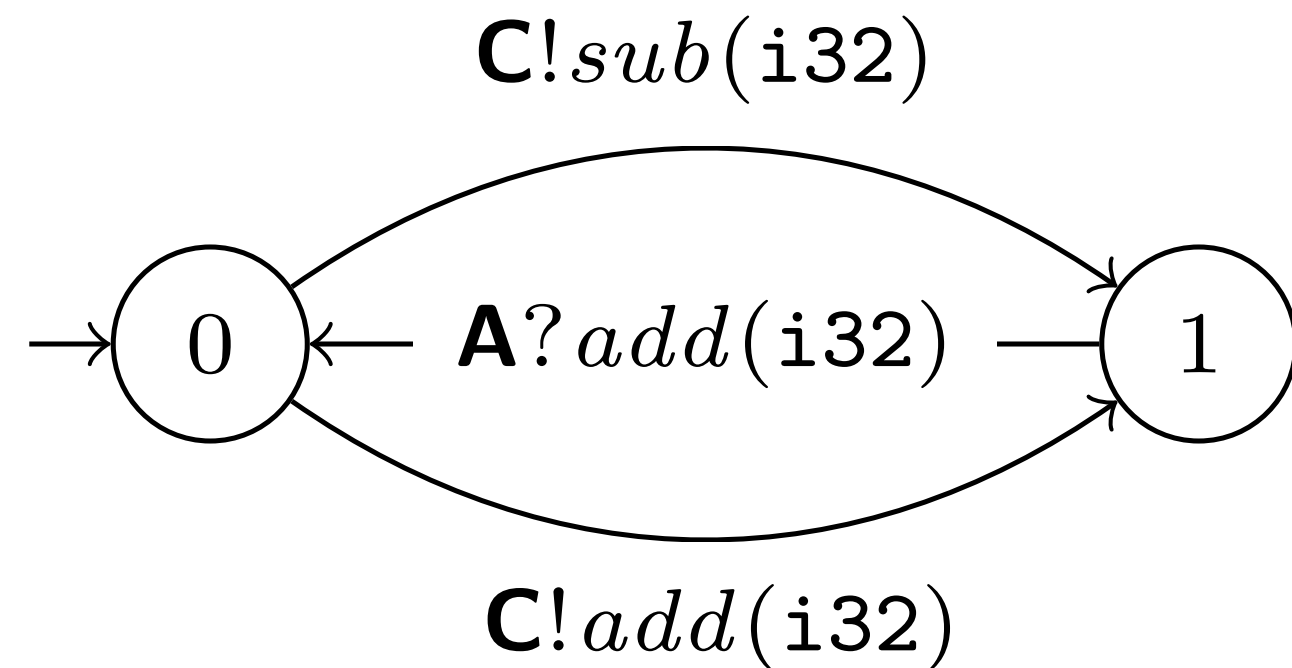
# Ring Protocol

## Implementation



# Ring Protocol

## Implementation

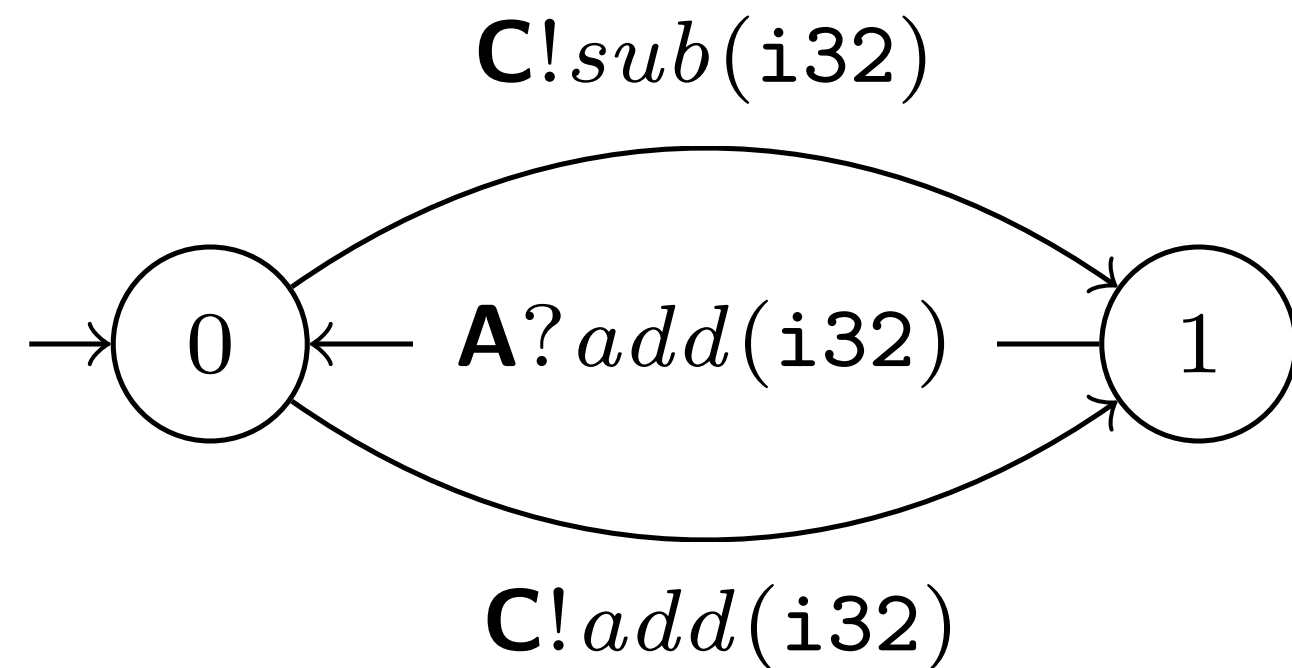


```
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

            s = if x > 0 {
                let s = s.select(Add(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y + x;
                s
            } else {
                let s = s.select(Sub(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y - x;
                s
            };
        }
    })
    .await
}
```

# Ring Protocol

## Implementation

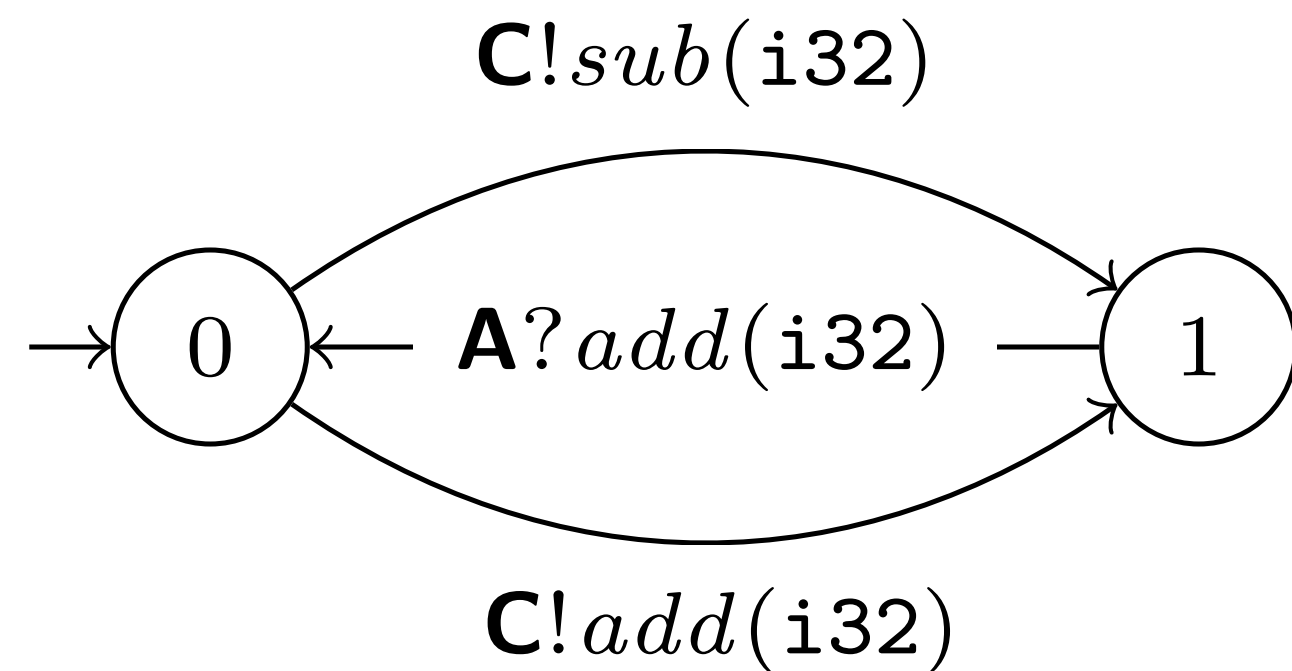


```
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

            s = if x > 0 {
                let s = s.select(Add(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y + x;
                s
            } else {
                let s = s.select(Sub(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y - x;
                s
            };
        }
    })
    .await
}
```

# Ring Protocol

## Implementation

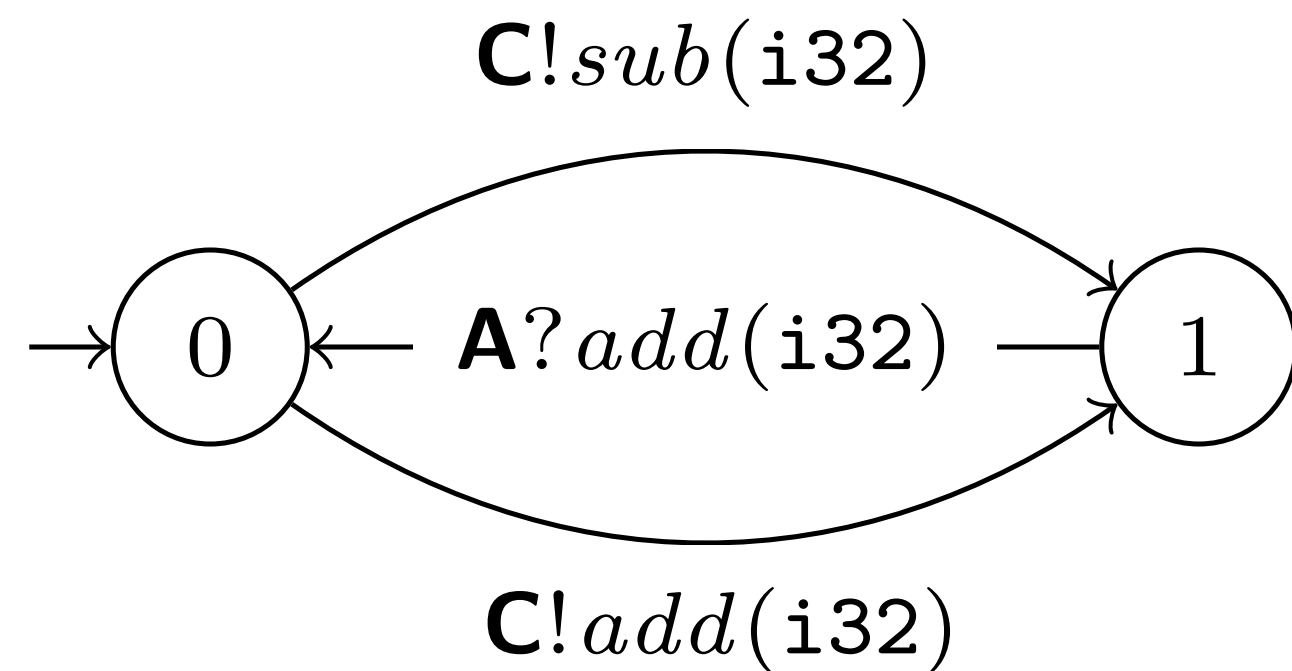


```
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

            s = if x > 0 {
                let s = s.select(Add(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y + x;
                s
            } else {
                let s = s.select(Sub(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y - x;
                s
            };
        }
    })
    .await
}
```

# Ring Protocol

## Implementation

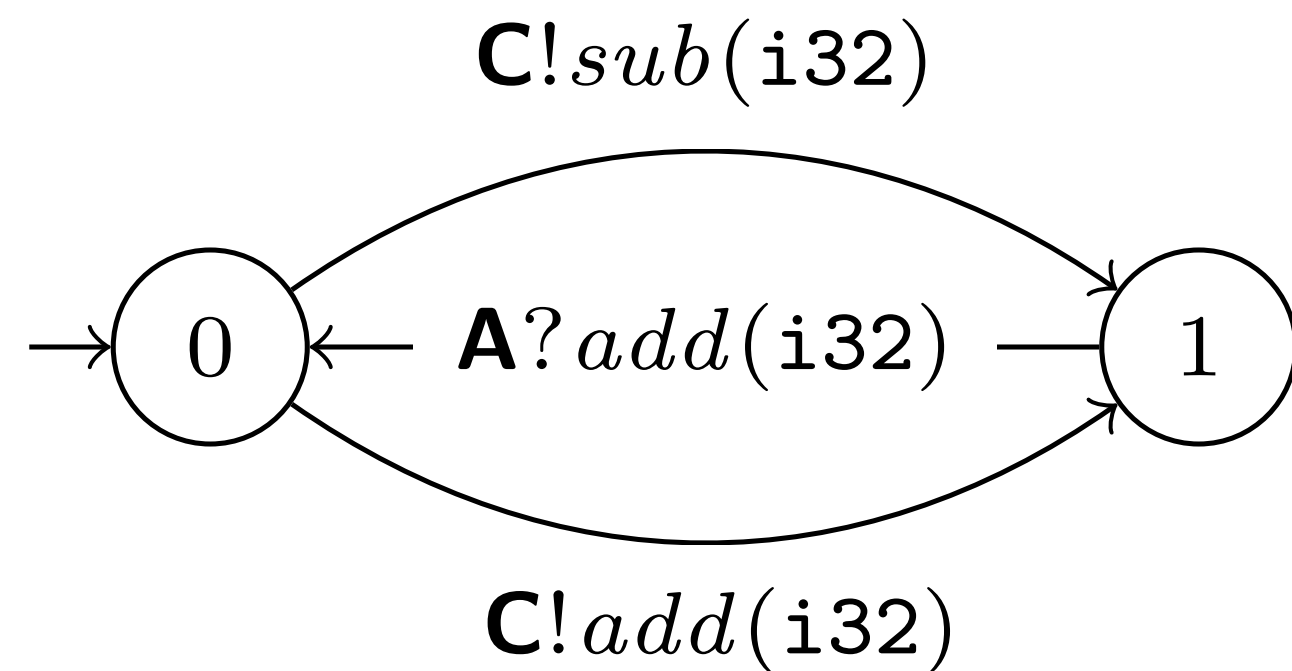


```
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

            s = if x > 0 {
                let s = s.select(Add(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y + x;
                s
            } else {
                let s = s.select(Sub(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y - x;
                s
            };
        }
    })
    .await
}
```

# Ring Protocol

## Implementation

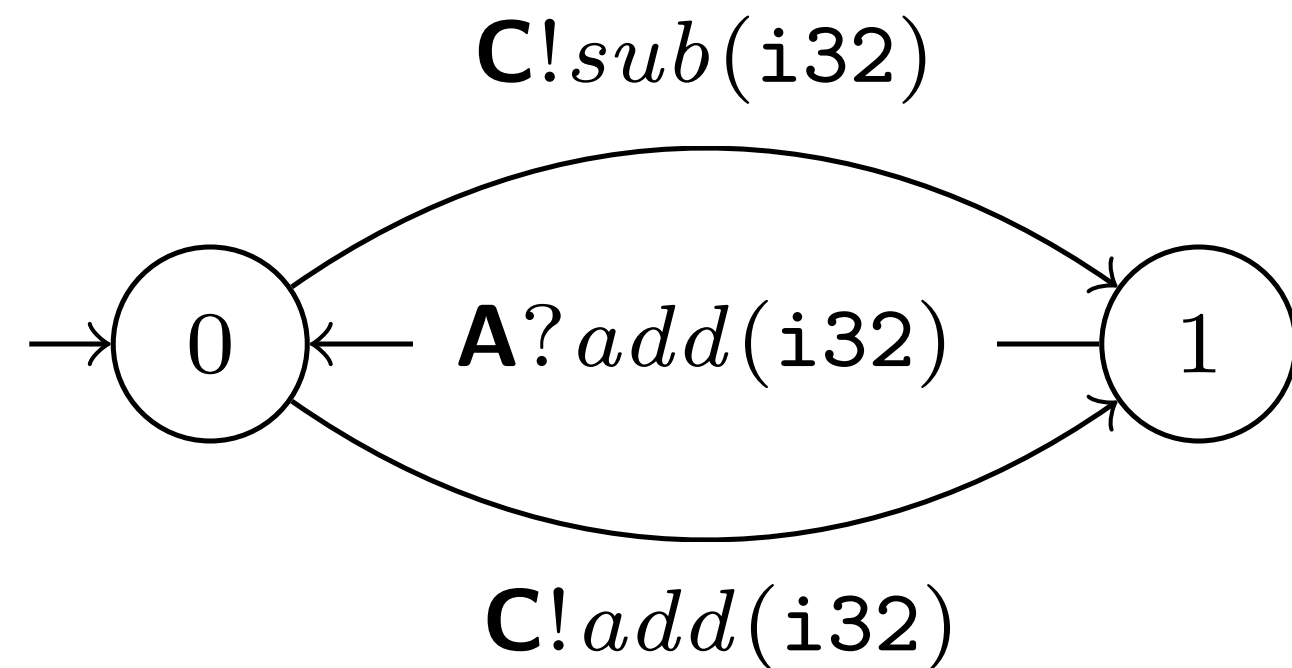


```
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

            s = if x > 0 {
                let s = s.select(Add(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y + x;
                s
            } else {
                let s = s.select(Sub(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y - x;
                s
            };
        }
    })
    .await
}
```

# Ring Protocol

## Implementation

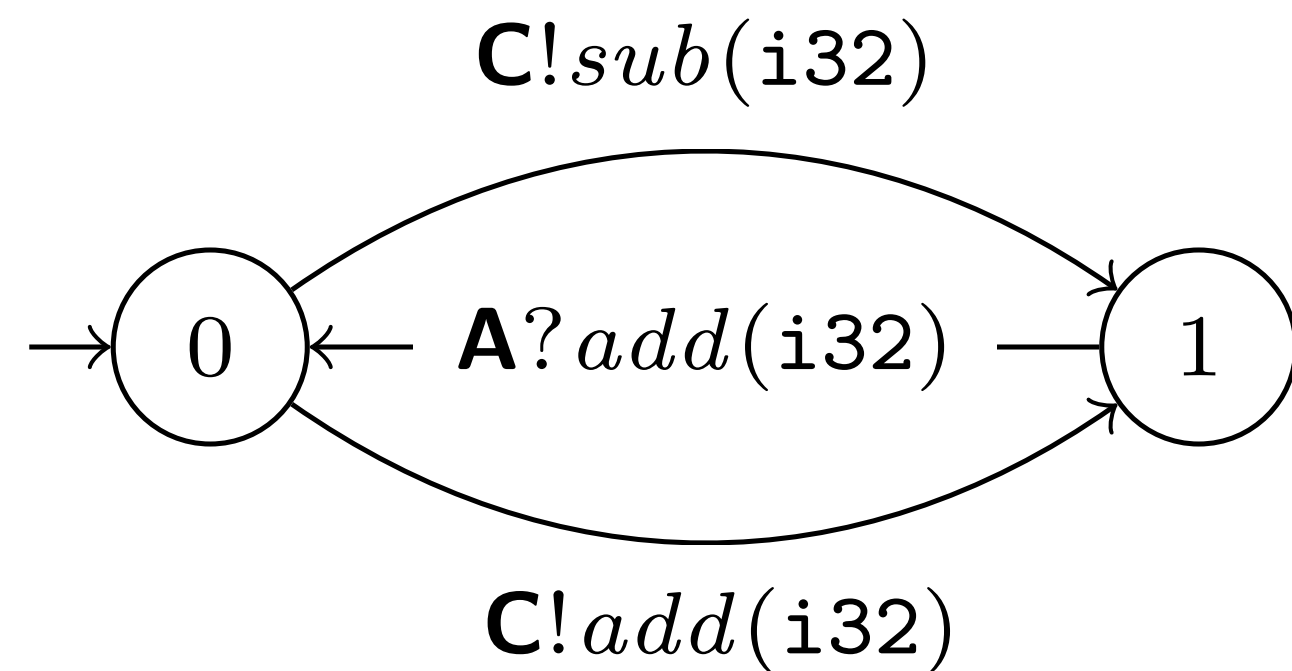


```
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

            s = if x > 0 {
                let s = s.select(Add(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y + x;
                s
            } else {
                let s = s.select(Sub(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y - x;
                s
            };
        }
    })
    .await
}
```

# Ring Protocol

## Implementation

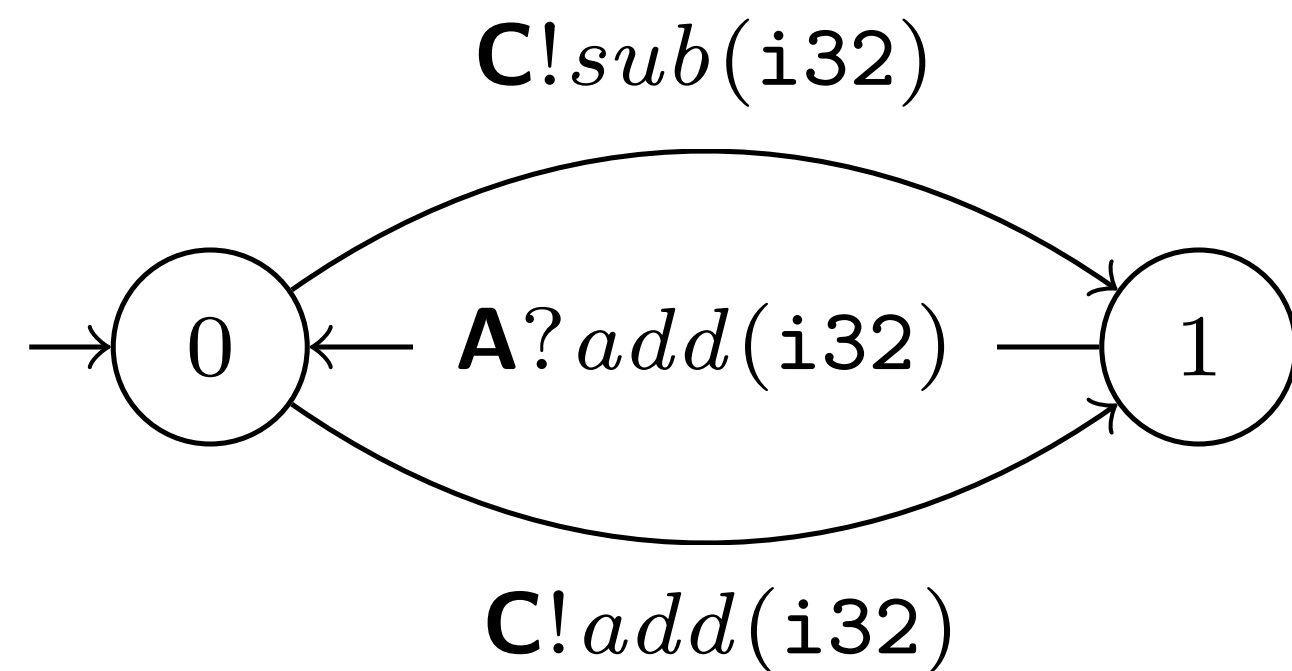


```
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

            s = if x > 0 {
                let s = s.select(Add(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y + x;
                s
            } else {
                let s = s.select(Sub(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y - x;
                s
            };
        }
    })
    .await
}
```

# Ring Protocol

## Implementation

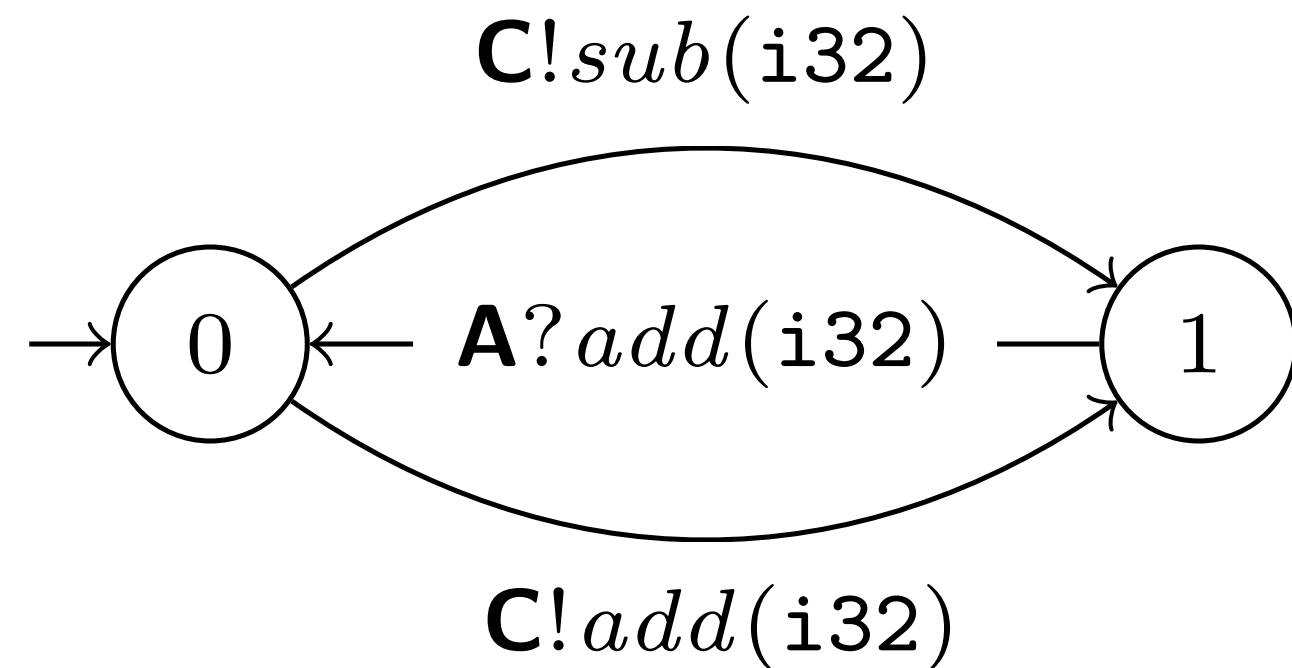


```
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

            s = if x > 0 {
                let s = s.select(Add(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y + x;
                s
            } else {
                let s = s.select(Sub(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y - x;
                s
            };
        }
    })
    .await
}
```

# Ring Protocol

## Implementation

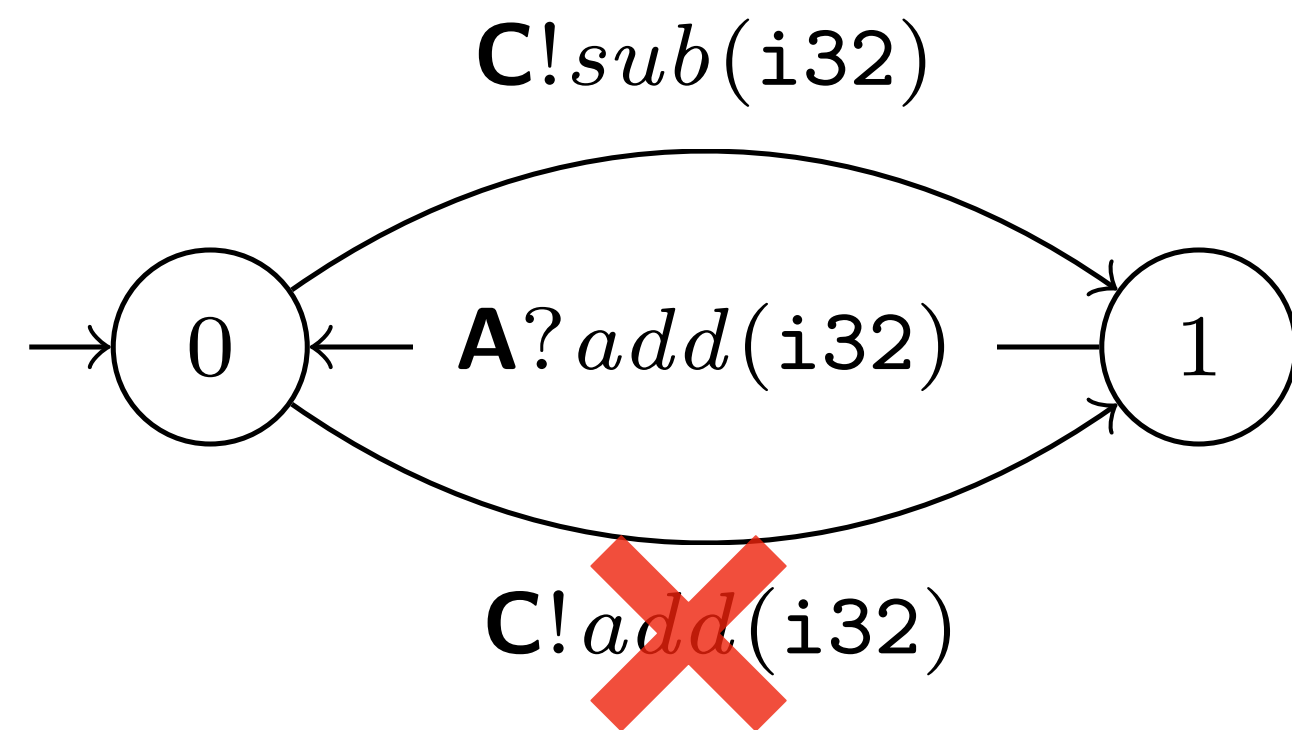


```
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

            s = if x > 0 {
                let s = s.select(Add(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y + x;
                s
            } else {
                let s = s.select(Sub(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y - x;
                s
            };
        }
    })
    .await
}
```

# Ring Protocol

## Implementation

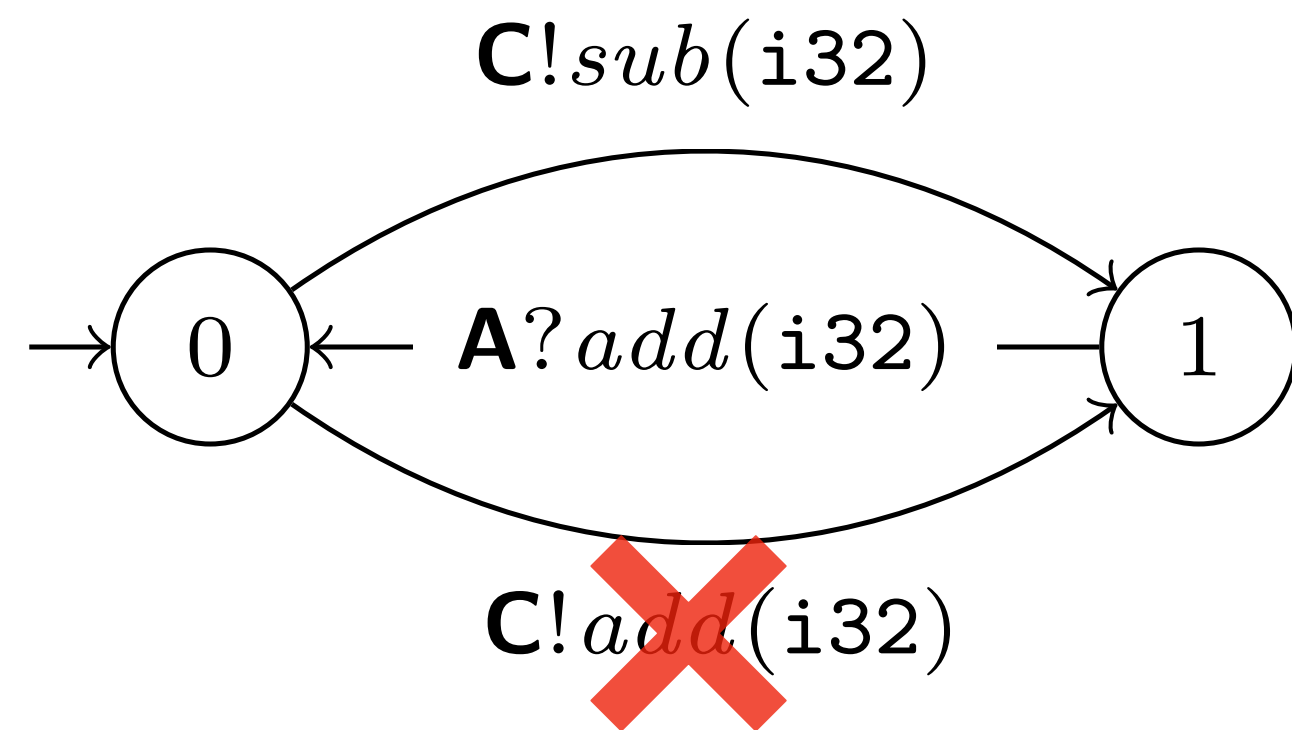


```
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

            s = if x > 0 {
let s = s.select(Add(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y + x;
                s
            } else {
                let s = s.select(Sub(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y - x;
                s
            };
        }
    })
    .await
}
```

# Ring Protocol

## Implementation



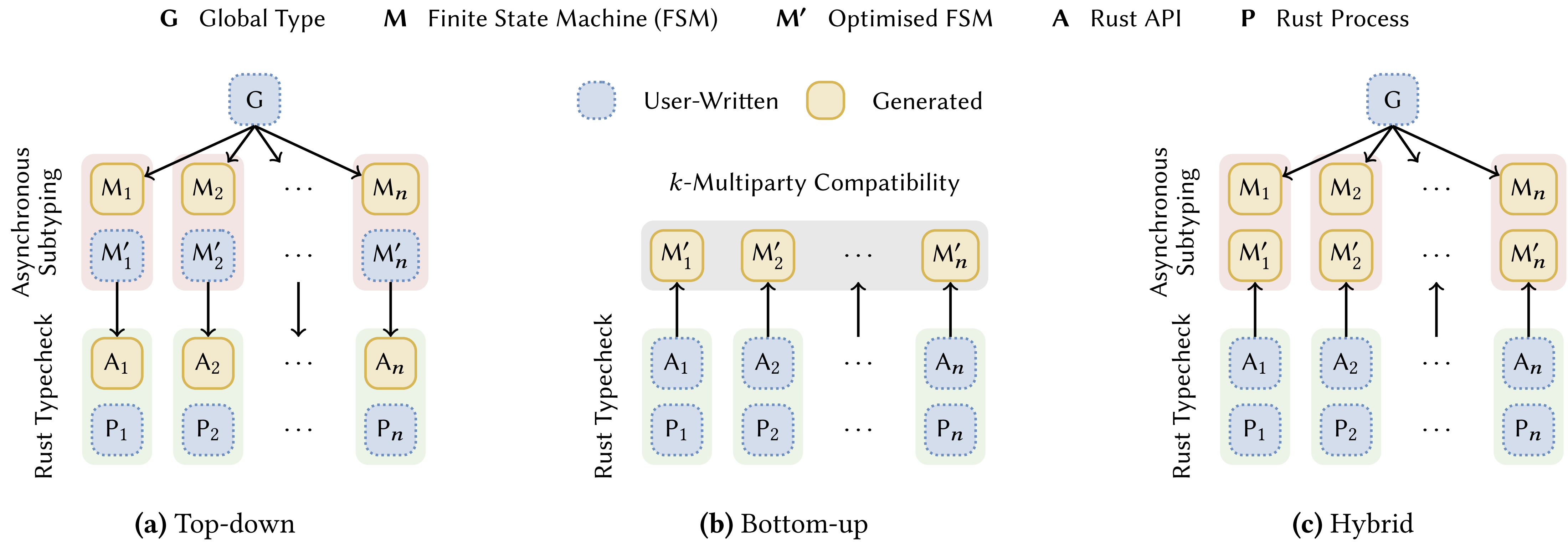
```
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

            s = if x > 0 {
let s = s.select(Add(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y + x;
            } else {
                let s = s.select(Sub(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y - x;
            }
        }
    })
    .await
}
```

method not found in `rumpsteak::Select<'\_, B, C, RingBChoice<'\_, B>>`

# Rumpsteak Framework

## Three Approaches



# Theories for Communication Optimisation

## Asynchronous Reordering Revisited

How do we check that asynchronous reorderings are **safe**?

# Theories for Communication Optimisation

## Asynchronous Reordering Revisited

How do we check that asynchronous reorderings are *safe*?

1. Asynchronous subtyping relation [Ghilezan et al., POPL'2021]

# Theories for Communication Optimisation

## Asynchronous Reordering Revisited

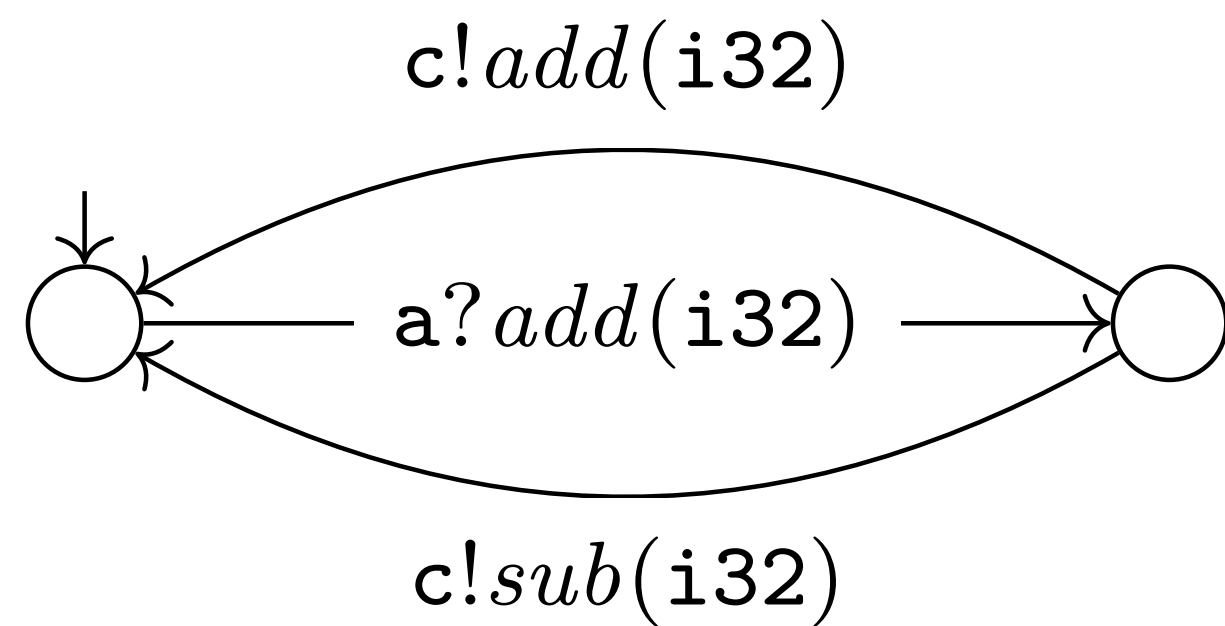
How do we check that asynchronous reorderings are *safe*?

1. Asynchronous subtyping relation [Ghilezan et al., POPL'2021]
2.  $k$ -multiparty compatibility [Lange and Yoshida, CAV'2019]

# Safety

## Asynchronous Subtyping

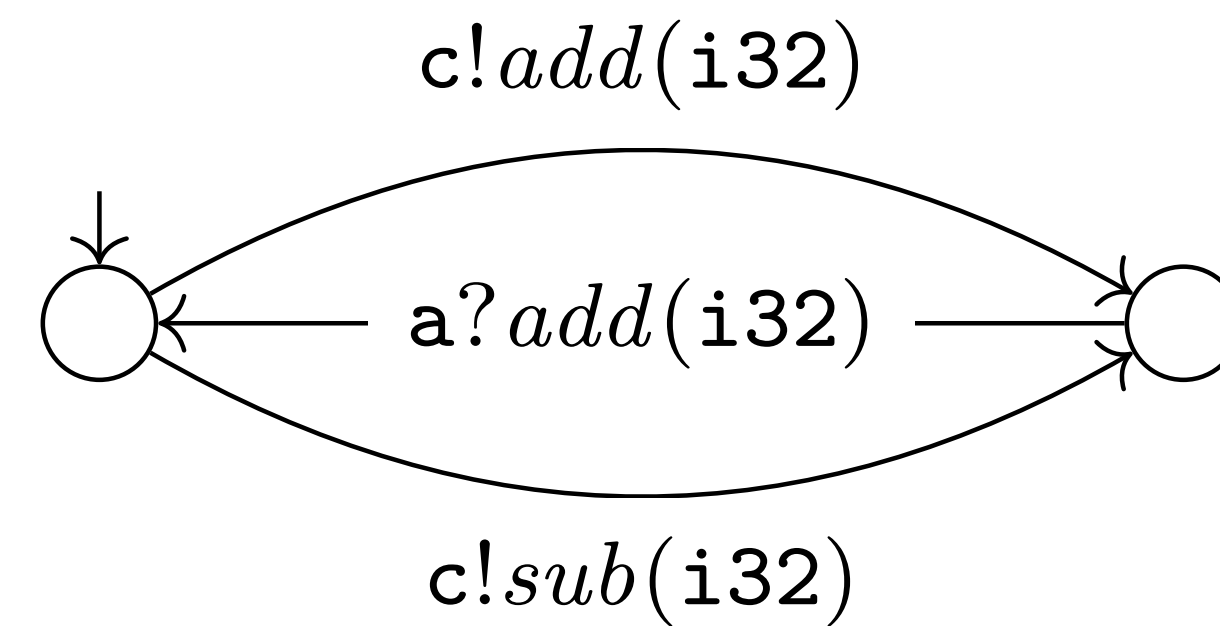
PROJECTED B



Safe?



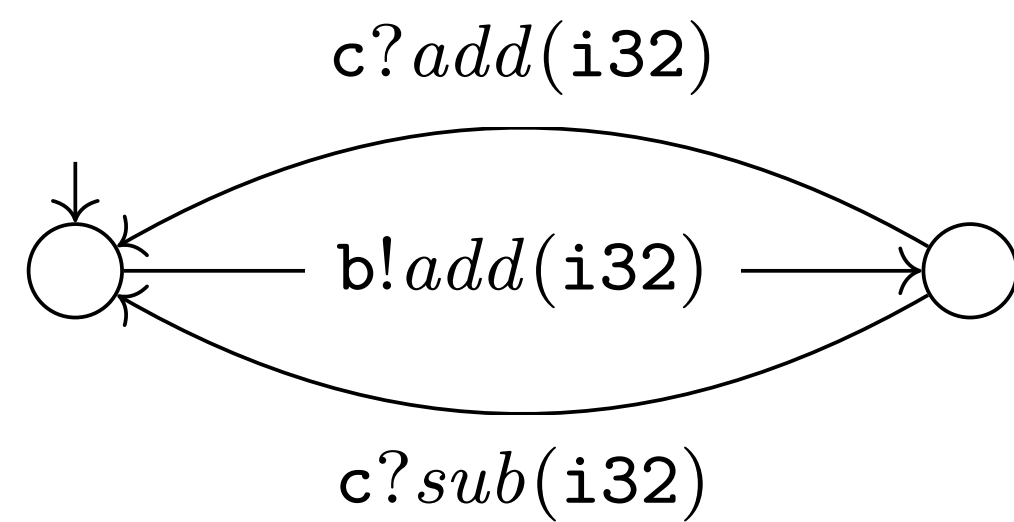
OPTIMISED B



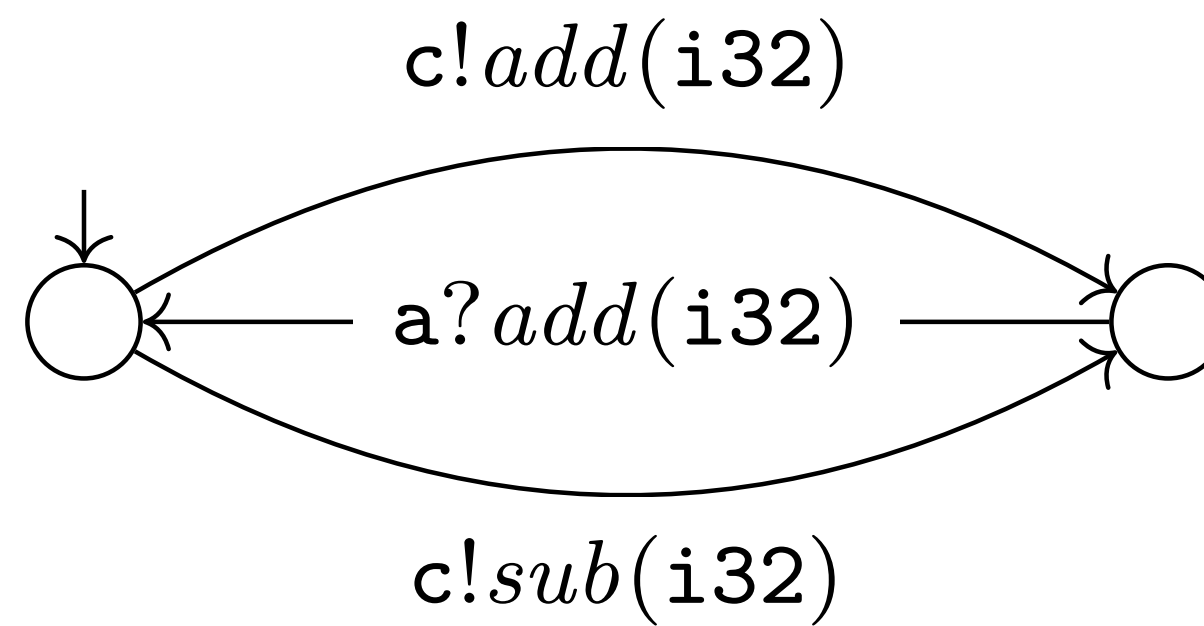
# Safety

## *k*-Multiparty Compatibility

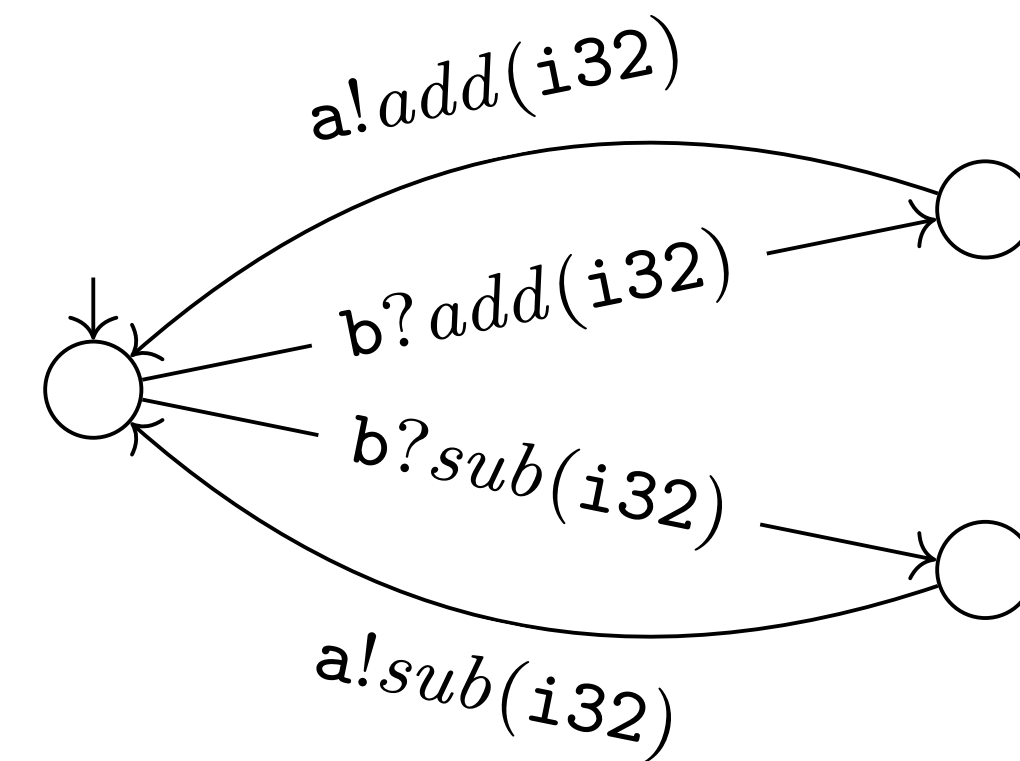
OPTIMISED A



OPTIMISED B



OPTIMISED C



Safe?

# Asynchronous Subtyping

## Existing work

- Relation given by [Ghilezan et al., POPL 2021]

# Asynchronous Subtyping

## Existing work

- Relation given by [Ghilezan et al., POPL 2021]
  - Sound 

# Asynchronous Subtyping

## Existing work

- Relation given by [Ghilezan et al., POPL 2021]
  - Sound 
  - Complete 

# Asynchronous Subtyping

## Existing work

- Relation given by [Ghilezan et al., POPL 2021]
  - Sound 
  - Complete 
  - Decidable [Lange and Yoshida, FoSSaCs 2017] 

# Asynchronous Subtyping

## Existing work

- Relation given by [Ghilezan et al., POPL 2021]
  - Sound ✓
  - Complete ✓
  - Decidable [Lange and Yoshida, FoSSaCs 2017] ✗
- Our aim is a sound and decidable algorithm!

# Algorithm for Asynchronous Subtyping

## Practical, Sound and Terminating

1. **Bound** the number of times we unroll recursions
2. Only unwrap choice **on demand**

# Asynchronous Subtyping

## Session Type Prefix

$\pi, \rho$	$::=$	$\epsilon$	empty prefix
		$p!l(S)$	message send
		$p?l(S)$	message receive
		$\pi_1.\pi_2$	concatenation

# Asynchronous Subtyping

## Reduction Rules

$$\mathcal{A}^{(p)} ::= q?l(S) \mid q?l(S).\mathcal{A}^{(p)} \quad (p \neq q)$$

$$\frac{S' \leq: S}{\langle p?l(S).\pi, \mathcal{A}^{(p)}.p?l(S').\pi' \rangle \rightarrow \langle \pi, \mathcal{A}^{(p)}.\pi' \rangle} [\text{RED-}\mathcal{A}]$$

# Asynchronous Subtyping

## Reduction Rules

$$\mathcal{A}^{(p)} ::= q?l(S) \mid q?l(S).\mathcal{A}^{(p)} \quad (p \neq q)$$

$$\frac{S' \leq: S}{\langle p?l(S).\pi, \mathcal{A}^{(p)}.p?l(S').\pi' \rangle \rightarrow \langle \pi, \mathcal{A}^{(p)}.\pi' \rangle} [\text{RED-}\mathcal{A}]$$

# Asynchronous Subtyping

## Reduction Rules

$$\mathcal{A}^{(p)} ::= q?l(S) \mid q?l(S).\mathcal{A}^{(p)} \quad (p \neq q)$$

$$\frac{S' \leq: S}{\langle p?l(S).\pi, \mathcal{A}^{(p)}.p?l(S').\pi' \rangle \rightarrow \langle \pi, \mathcal{A}^{(p)}.\pi' \rangle} [\text{RED-}\mathcal{A}]$$

# Asynchronous Subtyping

## Reduction Rules

$$\mathcal{A}^{(p)} ::= q?l(S) \mid q?l(S).\mathcal{A}^{(p)} \quad (p \neq q)$$

$$\frac{S' \leq: S}{\langle p?l(S).\pi, \mathcal{A}^{(p)}.p?l(S').\pi' \rangle \rightarrow \langle \pi, \mathcal{A}^{(p)}.\pi' \rangle} [\text{RED-}\mathcal{A}]$$

# Asynchronous Subtyping

## Reduction Rules

$$\langle p?\ell(S).q?m(S'), q?m(S').p?\ell(S) \rangle \xrightarrow{?} \langle q?m(S'), q?m(S') \rangle$$

# Asynchronous Subtyping

## Reduction Rules

$$\langle p?\ell(S).q?m(S'), q?m(S').p?\ell(S) \rangle \xrightarrow{?} \langle q?m(S'), q?m(S') \rangle$$

# Asynchronous Subtyping

## Reduction Rules

$$\langle p?\ell(S).q?m(S'), q?m(S').p?\ell(S) \rangle \xrightarrow{?} \langle q?m(S'), q?m(S') \rangle$$

# Asynchronous Subtyping

## Reduction Rules

$$\langle p?\ell(S).q?m(S'), q?m(S').p?\ell(S) \rangle \xrightarrow{?} \langle q?m(S'), q?m(S') \rangle$$



# Asynchronous Subtyping

## Reduction Rules

$$\langle p?\ell(S).q?m(S'), q?m(S').p?\ell(S) \rangle \xrightarrow{?} \langle q?m(S'), q?m(S') \rangle \quad \checkmark$$

$\uparrow$   
 $\mathcal{A}^{(p)}$

# Asynchronous Subtyping

## Reduction Rules

$$\langle p?l(S).q?m(S'), q?m(S').p?l(S) \rangle \xrightarrow{?} \langle q?m(S'), q?m(S') \rangle$$



$$\langle p?l(S).p?m(S'), p?m(S').p?l(S) \rangle \xrightarrow{?} \langle p?m(S'), p?m(S') \rangle$$

# Asynchronous Subtyping

## Reduction Rules

$$\langle p?l(S).q?m(S'), q?m(S').p?l(S) \rangle \xrightarrow{?} \langle q?m(S'), q?m(S') \rangle$$



$$\langle p?l(S).p?m(S'), p?m(S').p?l(S) \rangle \xrightarrow{?} \langle p?m(S'), p?m(S') \rangle$$

# Asynchronous Subtyping

## Reduction Rules

$$\langle p?l(S).q?m(S'), q?m(S').p?l(S) \rangle \xrightarrow{?} \langle q?m(S'), q?m(S') \rangle$$



$$\langle p?l(S).p?m(S'), p?m(S').p?l(S) \rangle \xrightarrow{?} \langle p?m(S'), p?m(S') \rangle$$

# Asynchronous Subtyping

## Reduction Rules

$$\langle p?\ell(S).q?m(S'), q?m(S').p?\ell(S) \rangle \xrightarrow{?} \langle q?m(S'), q?m(S') \rangle \quad \checkmark$$

$$\langle p?\ell(S).p?m(S'), p?m(S').p?\ell(S) \rangle \xrightarrow{?} \langle p?m(S'), p?m(S') \rangle \quad \times$$

# Asynchronous Subtyping

## Reduction Rules

$$\langle p?\ell(S).q?m(S'), q?m(S').p?\ell(S) \rangle \xrightarrow{?} \langle q?m(S'), q?m(S') \rangle \quad \checkmark$$

$$\langle p?\ell(S).p?m(S'), p?m(S').p?\ell(S) \rangle \xrightarrow{?} \langle p?m(S'), p?m(S') \rangle \quad \times$$

$\uparrow$   
 $\mathcal{A}^{(p)}$

# Asynchronous Subtyping

## Reduction Rules

$$\langle p?l(S).q?m(S'), q?m(S').p?l(S) \rangle \xrightarrow{?} \langle q?m(S'), q?m(S') \rangle \quad \checkmark$$

$$\langle p?l(S).p?m(S'), p?m(S').p?l(S) \rangle \xrightarrow{?} \langle p?m(S'), p?m(S') \rangle \quad \times$$

$\uparrow$   
 $\mathcal{A}^{(p)}$

$$\mathcal{A}^{(p)} ::= q?l(S) \mid q?l(S).\mathcal{A}^{(p)} \quad (p \neq q)$$

# Theorems

## Termination, Soundness & Complexity

**Lemma 3.** *Given finite prefixes  $\pi$  and  $\pi'$ ,  $\langle \pi \sqcup \pi' \rangle$  can be reduced only a finite number of times.*

**Theorem 4 (Termination).** *Our subtyping algorithm always eventually terminates.*

**Theorem 5 (Soundness).** *Our subtyping algorithm is sound.*

**Lemma 6.** *Given finite prefixes  $\pi$  and  $\pi'$ , the time complexity of reducing  $\langle \pi \sqcup \pi' \rangle$  is  $O(\min(|\pi|, |\pi'|))$ .*

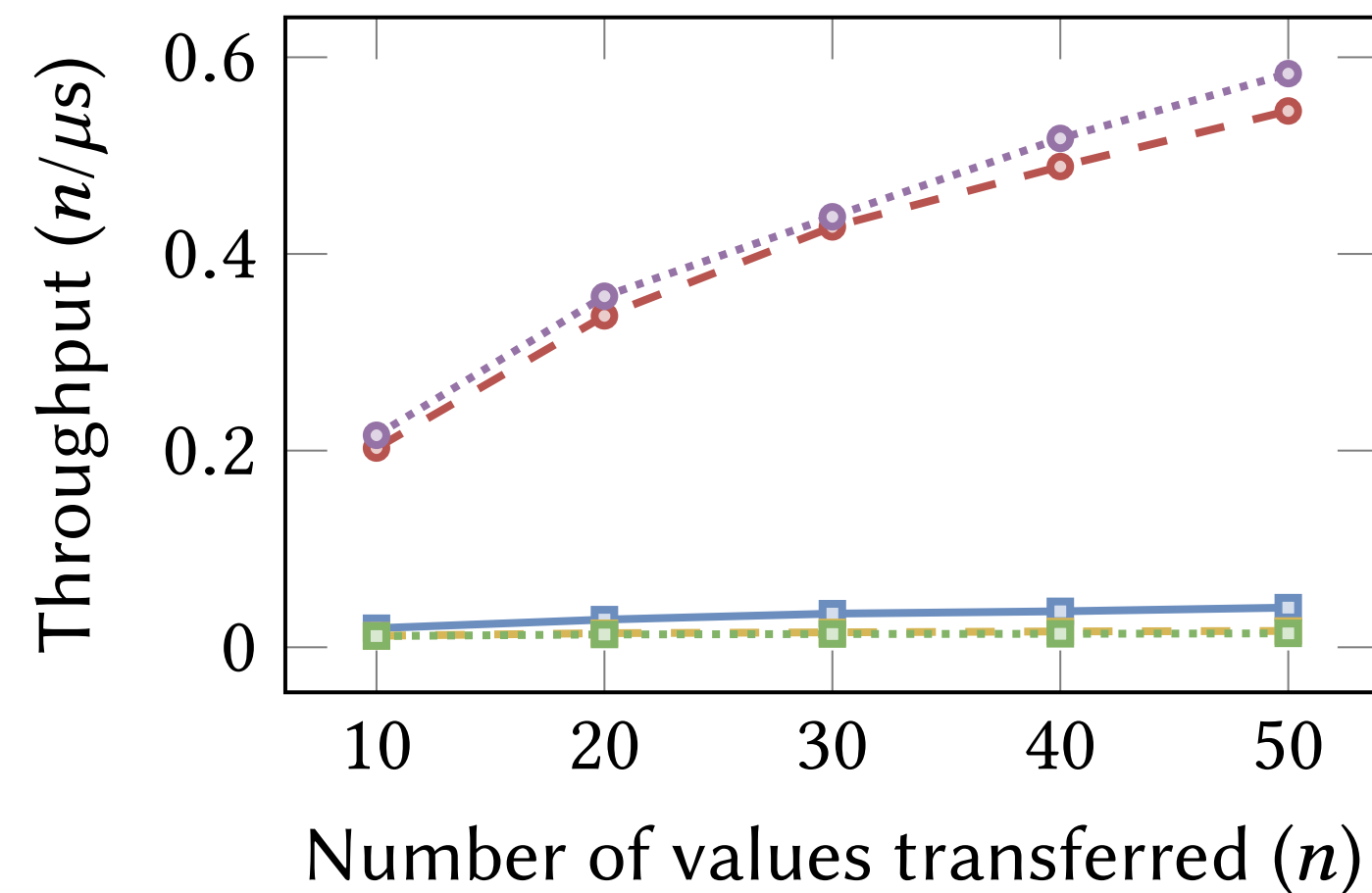
**Theorem 7 (Complexity).** *Consider  $T$  and  $T'$  as (possibly infinite) trees  $\mathcal{T}(T)$  and  $\mathcal{T}(T')$  with asymptotic branching factors  $b$  and  $b'$  respectively. Our algorithm has time complexity  $O(n \min(b, b')^n)$  and space complexity  $O(n \min(b, b'))$  in the worst case to determine if  $T \leq T'$  with bound  $n$ .*

# Evaluation

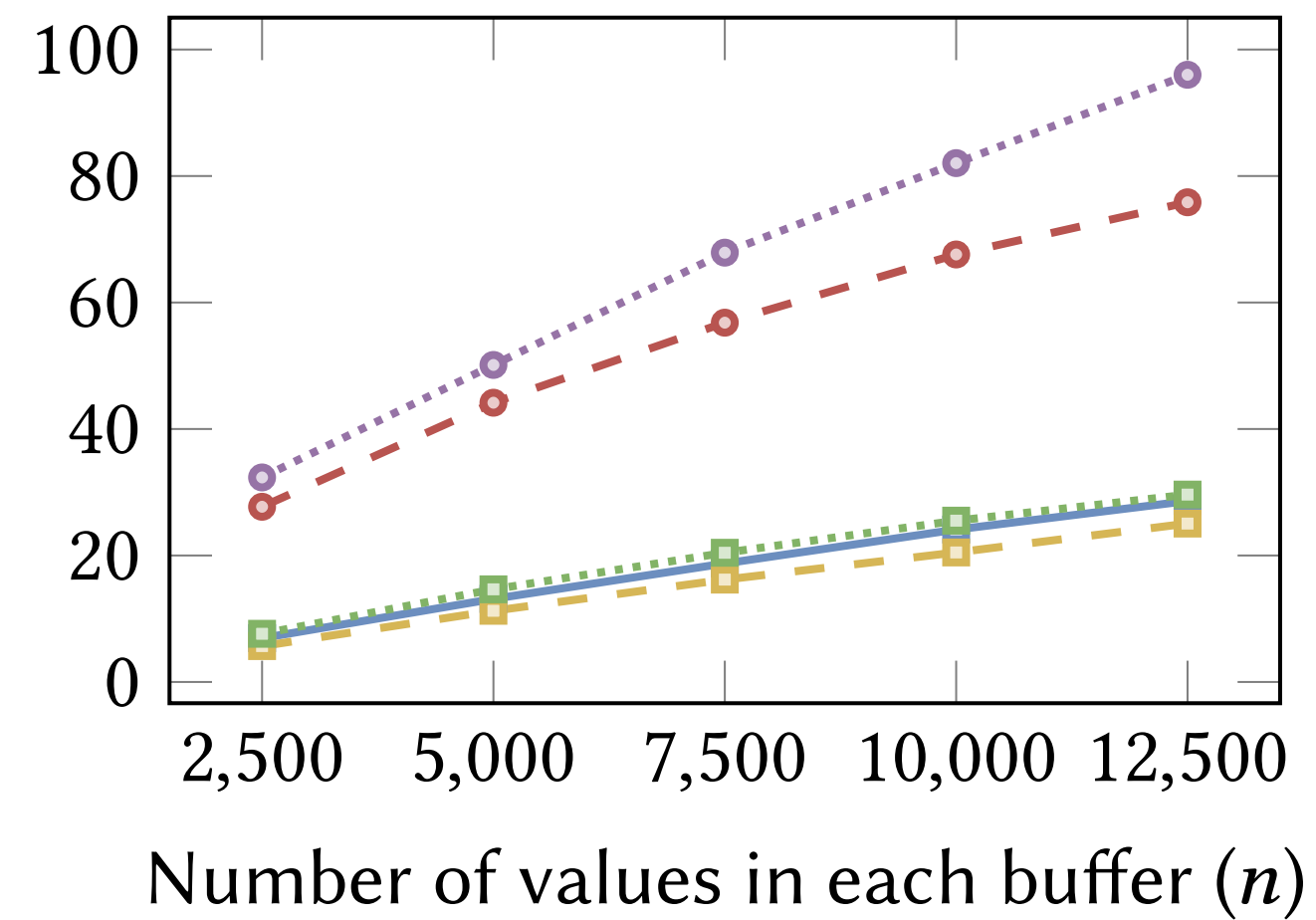
## Rust Framework Benchmarks

—■— SESH    -■- MULTICRUSTY    ...■... FERRITE    —○— RUSTFFT    -○- RUMPSTEAK    ...○... RUMPSTEAK (optimised)

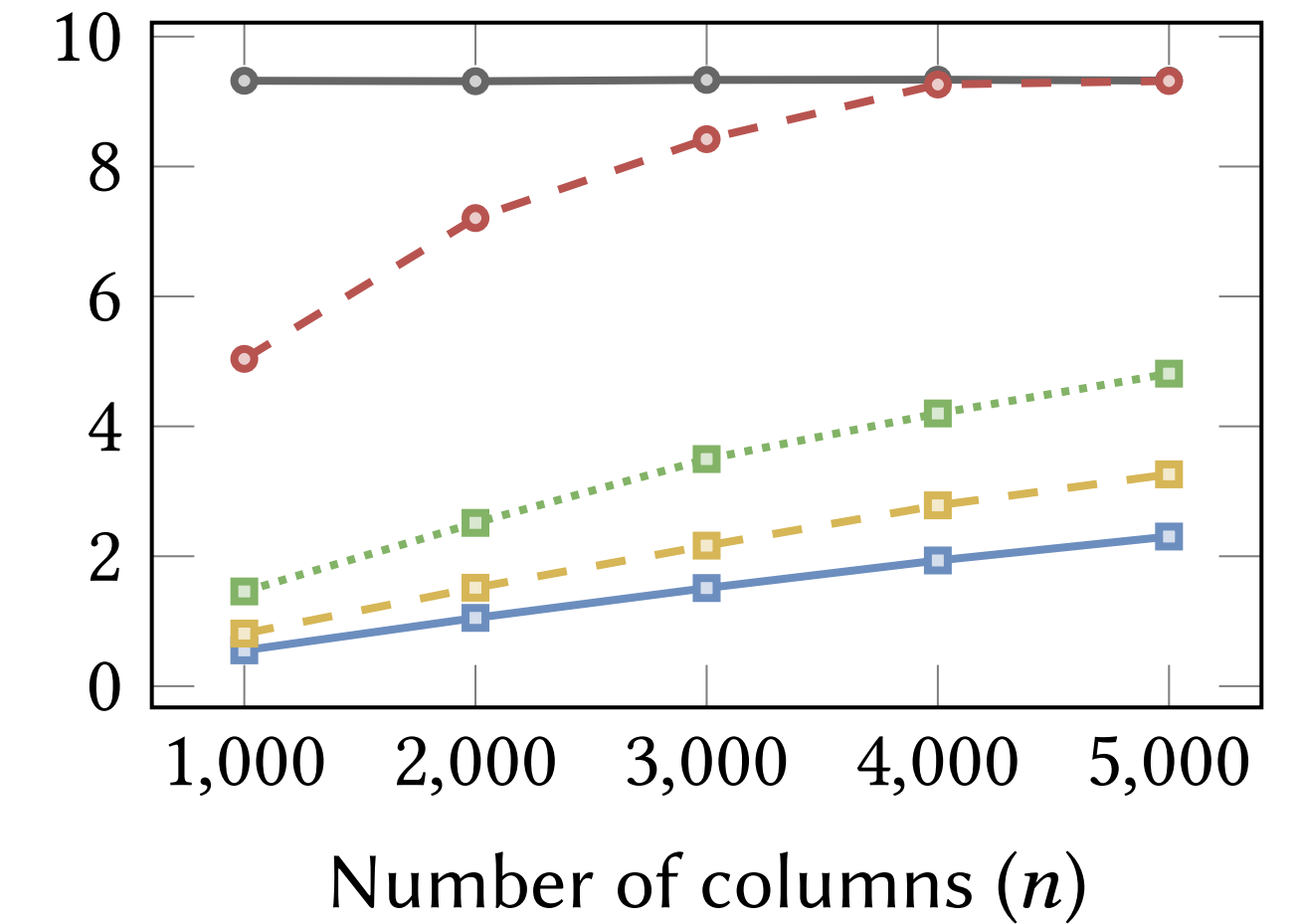
*Stream*



*Double Buffering*



*FFT*

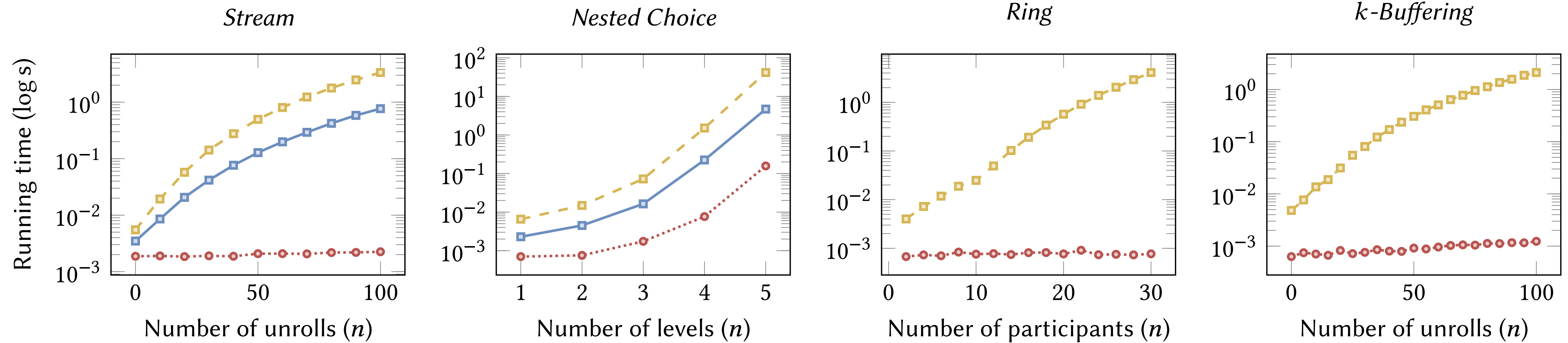


16-core AMD Opteron™ 6200 Series CPU @ 2.6GHz with hyperthreading, 128GB of RAM, Ubuntu 18.04.5 LTS and Rust Nightly 2021-07-06. We use version 0.3.5 of the Criterion.rs library and a multi-threaded asynchronous runtime from version 1.11.0 of the Tokio library.

# Evaluation

## Asynchronous Reordering Benchmarks

—■— SOUNDBINARY    -■-  $k$ -MC    ···○··· RUMPSTEAK



# Evaluation

## Expressiveness



Protocol	$n$	AMR	SESH	FERRITE	MULTICRUSTY	RUMPSTEAK	$k$ -MC	SOUNDBINARY
Two Adder	2		✓	✓	✓	✓	✓	✓
Three Adder	3		x	x	✓	✓	✓	x
Stream	2		✓	✓	✓	✓	✓	✓
Optimised Stream	2	✓	x	x	x	✓	✓	✓
Ring	3		x	x	✓	✓	✓	x
Optimised Ring	3	✓	x	x	x	✓	✓	x
Ring With Choice	3		x	x	✓	✓	✓	x
Optimised Ring With Choice	3	✓	x	x	x	✓	✓	x
Double Buffering	3		x	x	✓	✓	✓	x
Optimised Double Buffering	3	✓	x	x	x	✓	✓	x
Alternating Bit	2		x	x	x	✓	✓	✓
Elevator	3	✓	x	x	x	✓	✓	x
FFT	8		x	x	✓	✓	✓	x
Optimised FFT	8	✓	x	x	x	✓	✓	x
Authentication	3		x	x	✓	✓	✓	x
Client-Server Log	3		x	x	✓	✓	✓	x
Hospital	2	✓	x	x	x	x	x	✓

$n$  Number of participants    AMR Asynchronous message reordering

✓ Expressible    x Expressible using endpoint types (but without deadlock-freedom guarantee)    x Not expressible

# References

## Multiparty Session Types and Rust

- Multiparty session types and communicating automata
  - Invited paper in the FCT '21 proceedings
  -  Scribble <https://github.com/scribble>
  -  <https://github.com/nuscr>
  - **rumpsteak** <https://github.com/zakcutner/rumpsteak>
- **multi-crusty** <http://mrg.doc.ic.ac.uk/tools/multicrusty/>
  - **[ECOOP'22]** N. Laguardie (IC), R. Neykova (Brunel), NY

# Mini-Project Proposals

## Amazon Prime Video Team and Actor Languages

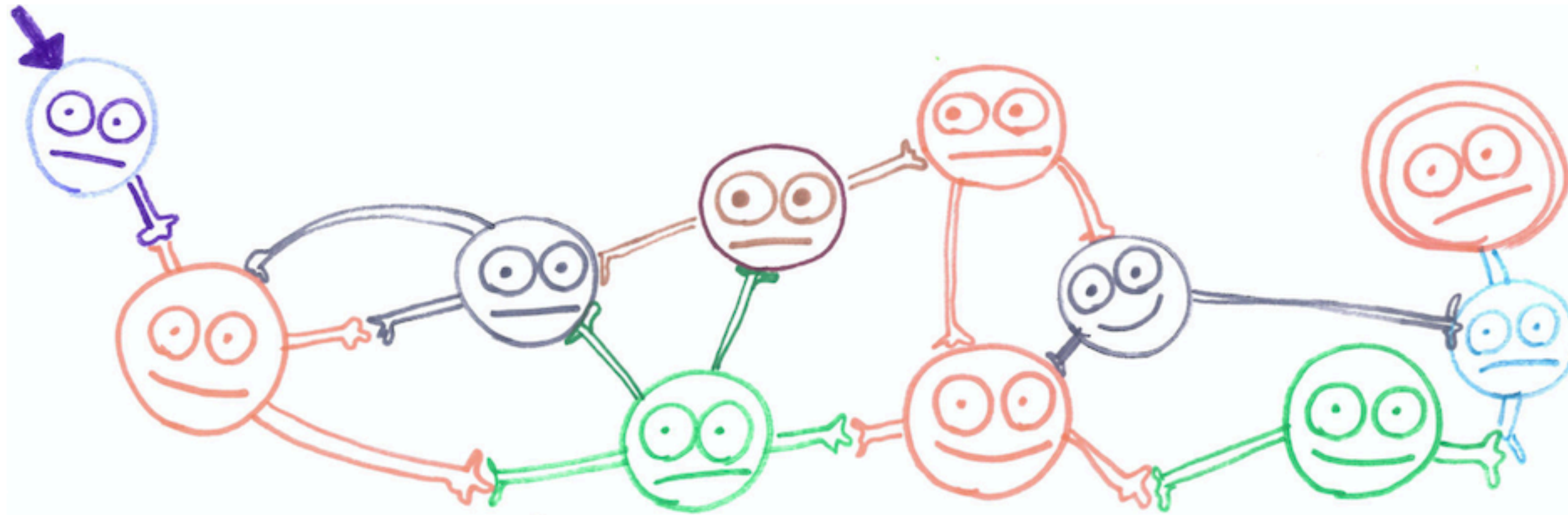
- **Automatic App Navigation** (with Amazon Web Services) conducted by Amazon Prime Video Team at London
- **Rust Programming** for Actor Languages (with AcTyx AG (<https://www.actyx.com/>))
- **Verified Multi-Agent Programming with Actor Models** (Model Checking of Go)



# Thank you! Questions?



<http://mrg.cs.ox.ac.uk/>



# CFSMs [1980-] ITU notation SDL · MSCS ...

**Def** A CFSM  $M = (Q, C, q_0, \Sigma, \delta)$

$Q$  a finite set of states

$C = \{ pq \in \text{Participant}^2 \mid p \neq q \}$

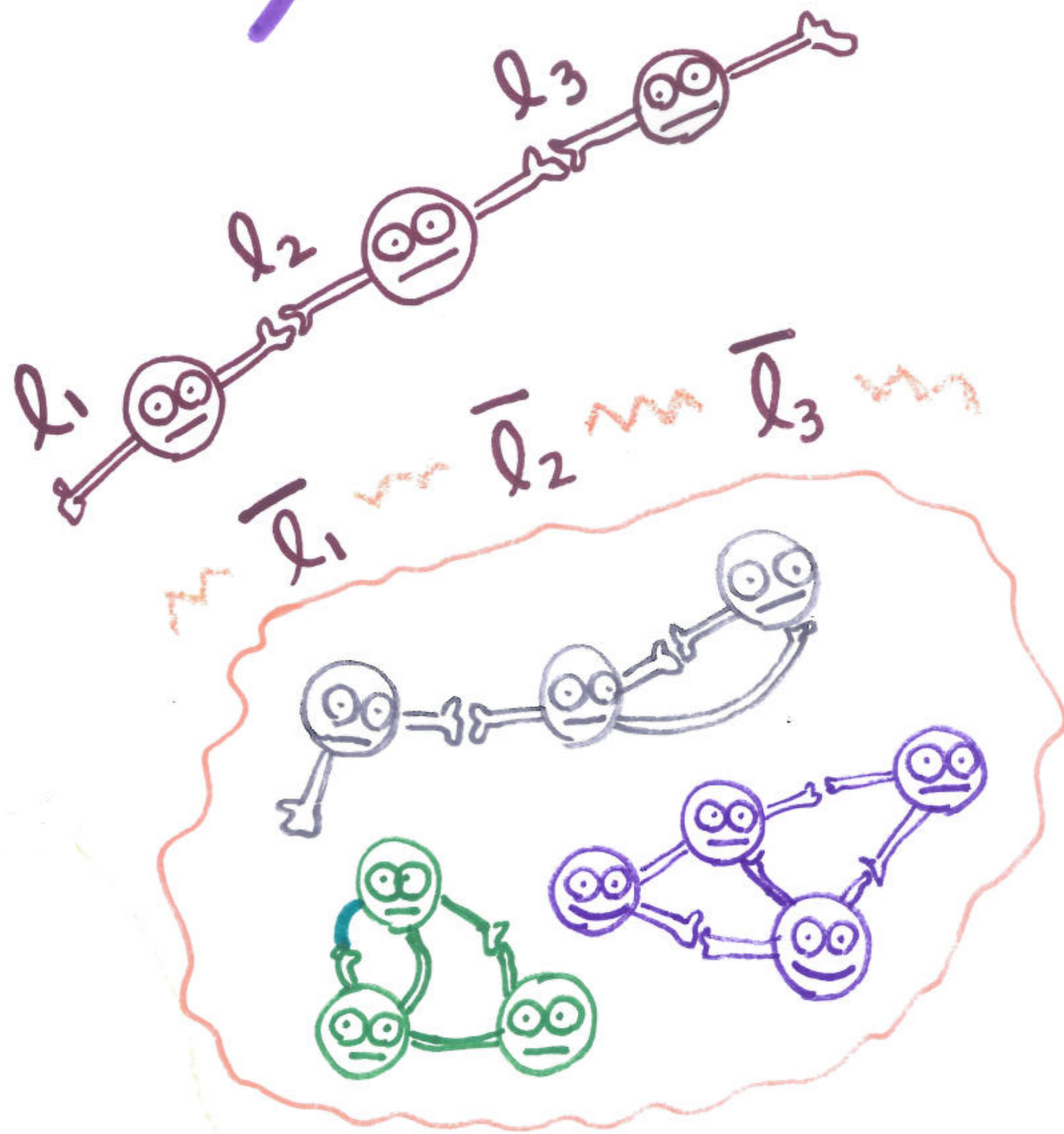
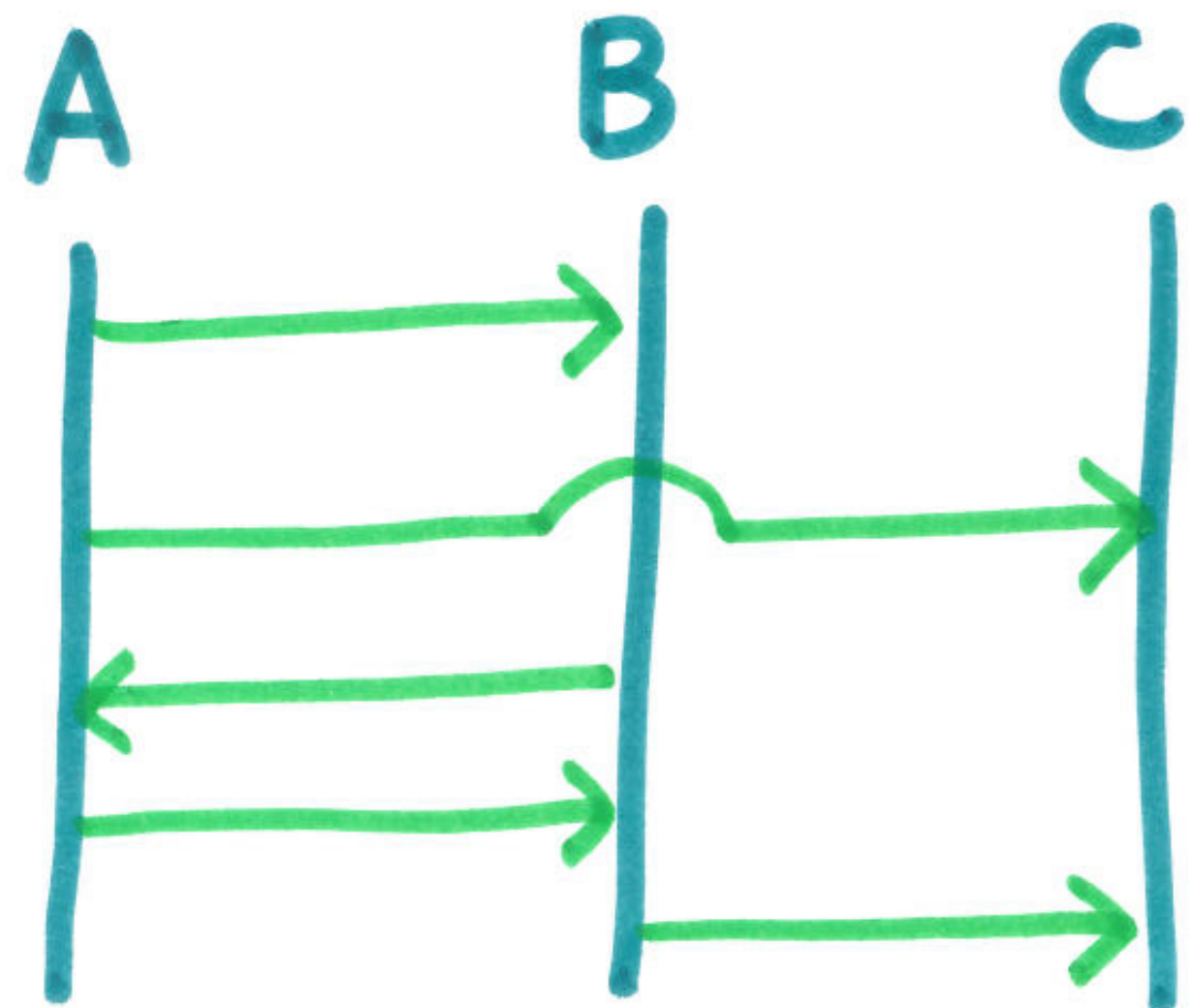
$q_0$  initial state

$\Sigma$  a finite alphabet of messages

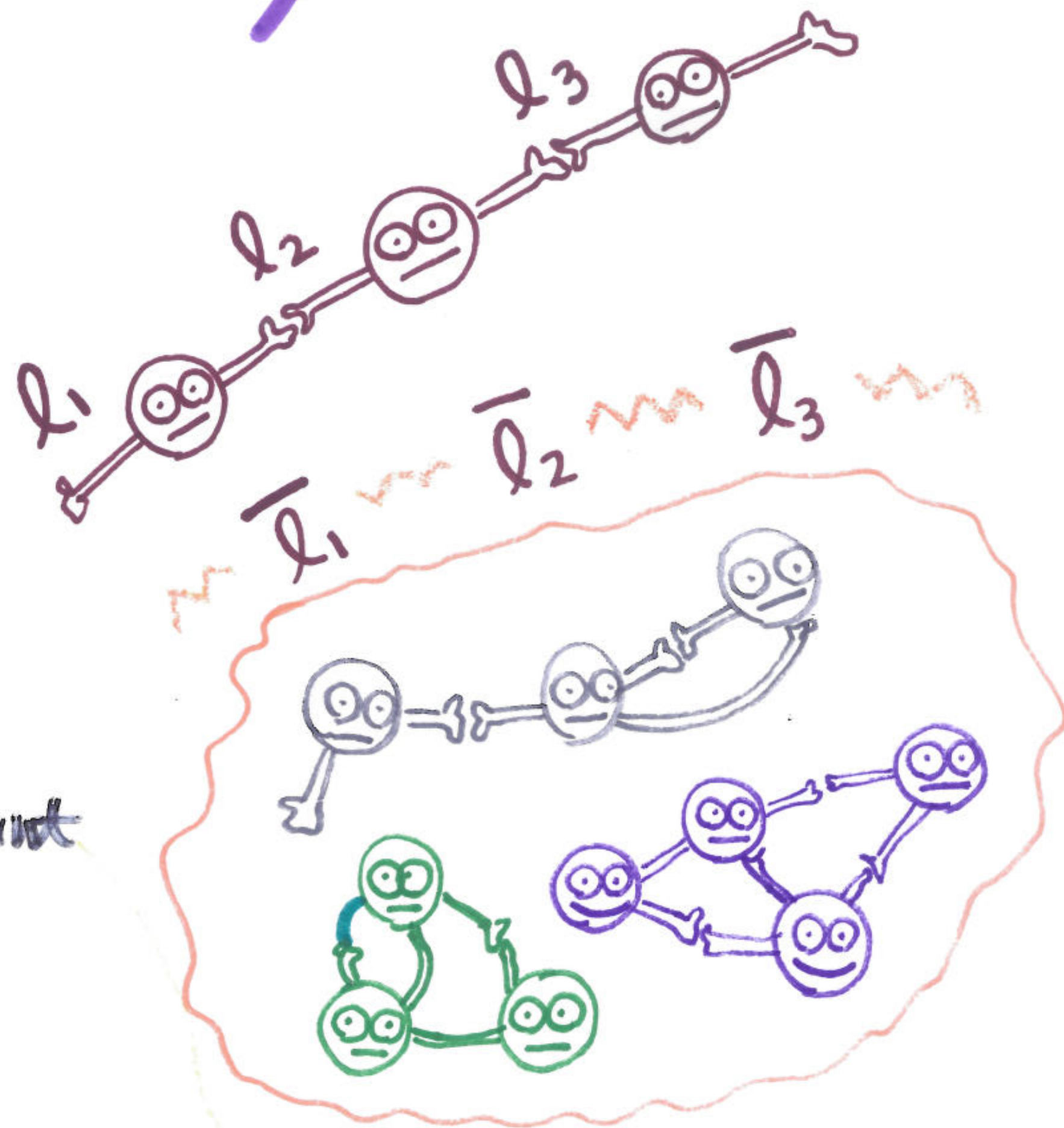
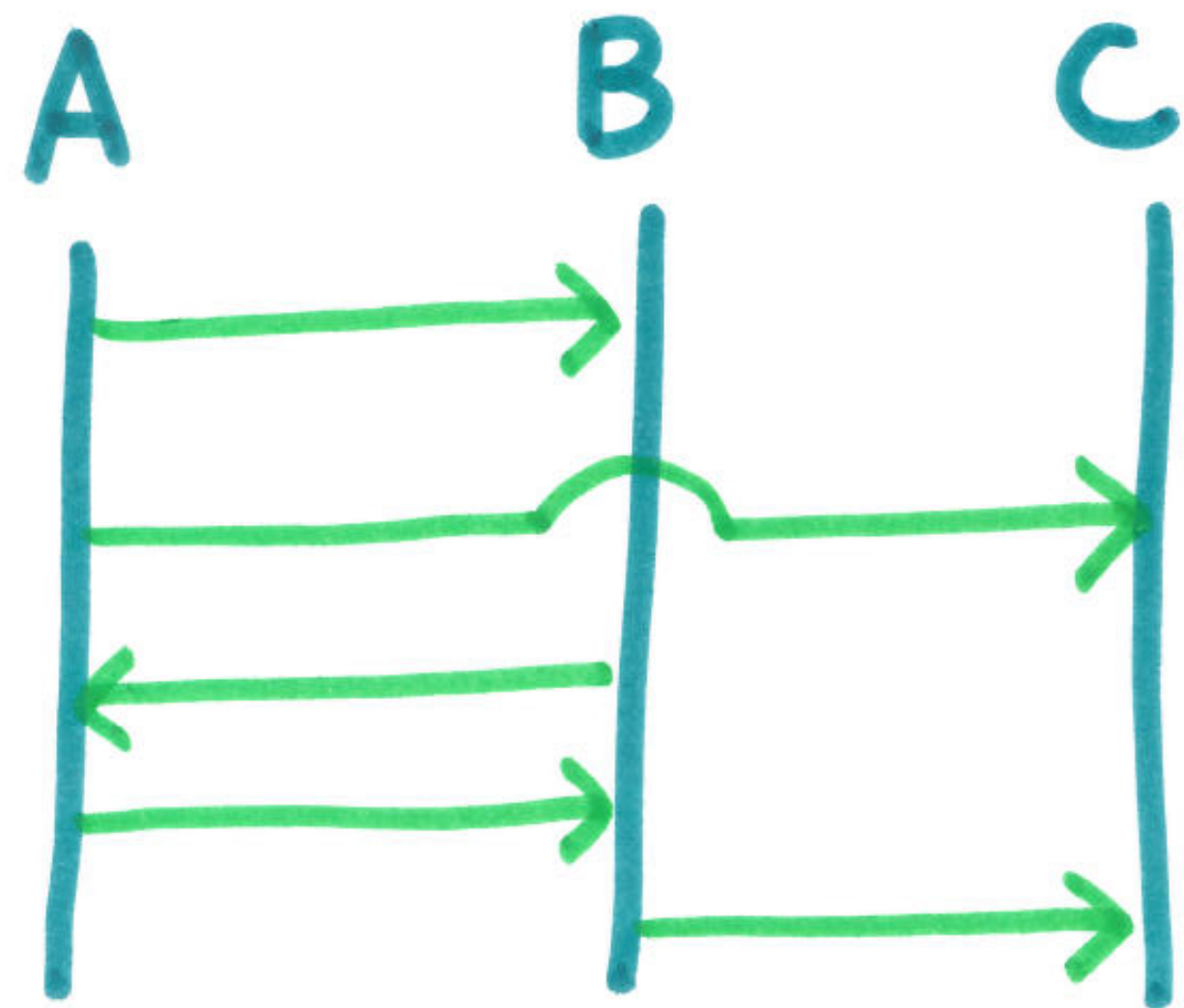
$\delta \subseteq Q \times (C \times \{!, ?\} \times \Sigma) \times Q$  a finite set of transitions

**Def** CS  $S = (M_p)_{p \in \text{Participant}}$

# Multiparty Compatibility



# Multiparty Compatibility



Def  $S = (M_p)_{p \in \text{Participant}}$

$\forall s . s_0 \rightsquigarrow^* s$   
1-buffer execution

if  $M_i$  does action  $l$

then  $(M_{\bar{j}})_{\bar{j} \in P \setminus i}$  do action  $\bar{l}$   
 after some  $\rightsquigarrow^*$

# Multiparty Compatibility

**Definition** System  $S = (M_p)_{p \in \mathcal{P}}$  is **MC** if for any 1-bound reachable state  $s \in RS_1(S)$ , and any output action  $pq!a$  from  $s$  in  $M_p$ , there exists an alternation  $\varphi.t$  from  $s$  in a system where  $\text{act}(t) = pq!a$  and  $p \notin \text{act}(\varphi)$

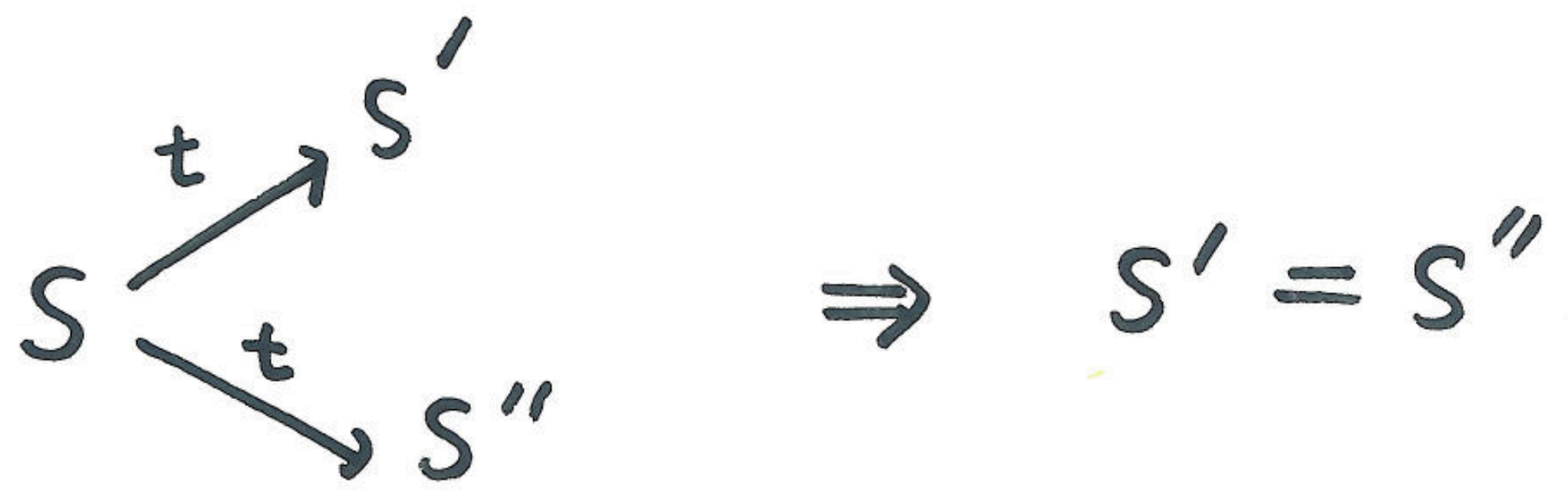
(Dual for input)

$S \xrightarrow{t} S'$  configuration  $S = (\vec{q}; \vec{W})$   
states queues

Send  $(\dots q_p \dots; \dots W_{pq} \dots) \xrightarrow{pq!l} (\dots q'_p; \dots W_{pq} \cdot l \dots)$   
 $q_p$   $W_{pq}$

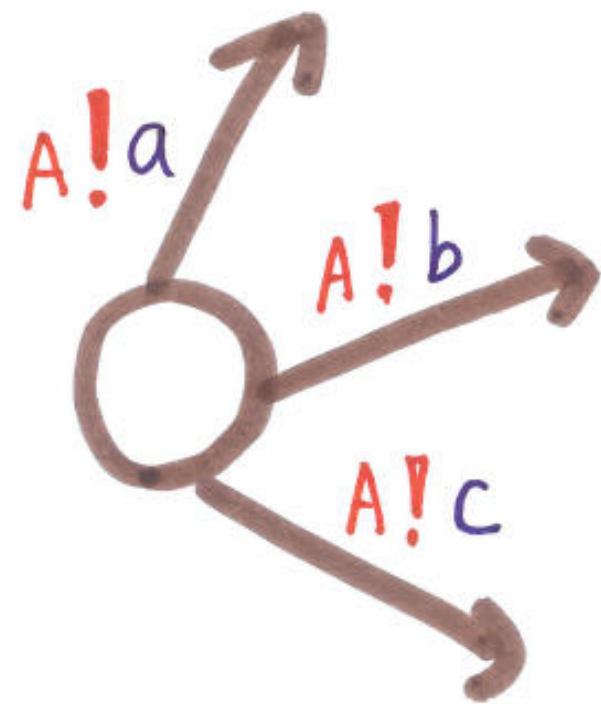
Receive  $(\dots q_q \dots; \dots l \cdot W_{pq} \dots) \xrightarrow{pq?l} (\dots q'_q \dots; \dots W_{pq} \dots)$

Deterministic CFM



# Basic CFSMs

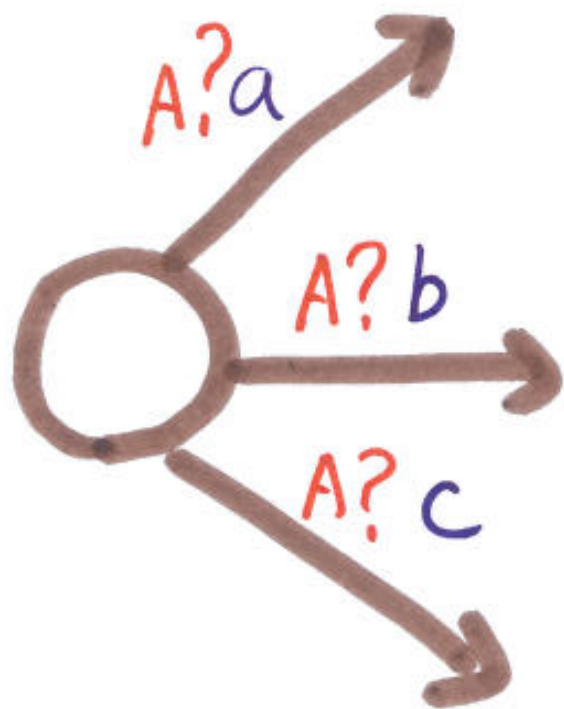
A CFSM is **Basic** if **deterministic**  
**directed**, has **no mixed states**



sending



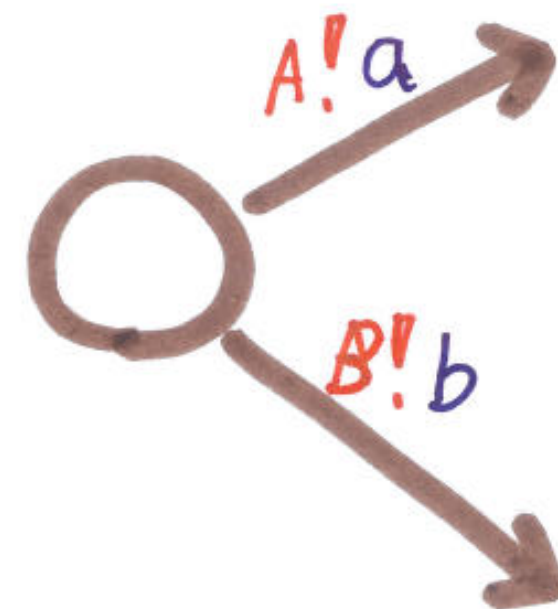
$T = A!\{a, b, c\}$



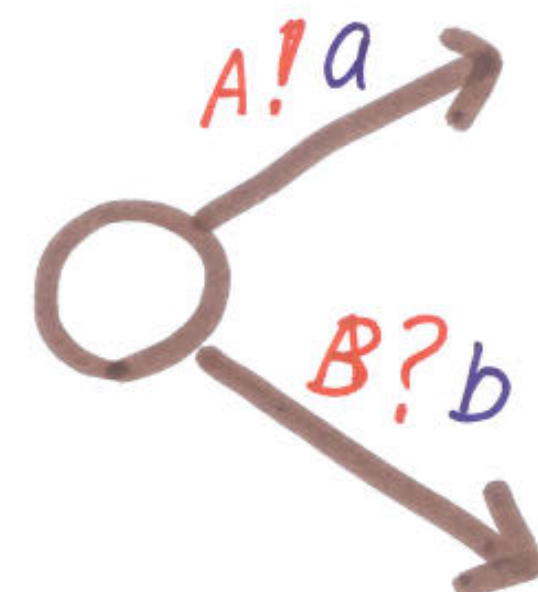
receiving



$A?\{a, b, c\}$



non  
directed



mixed



# k-Multiparty Compatibility [CAV'19]

