

Multiparty Session Types and Communicating Automata

FCT 2021: 23rd International Symposium on Fundamentals of Computation Theory



Nobuko Yoshida

Imperial College
London

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

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

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]

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?

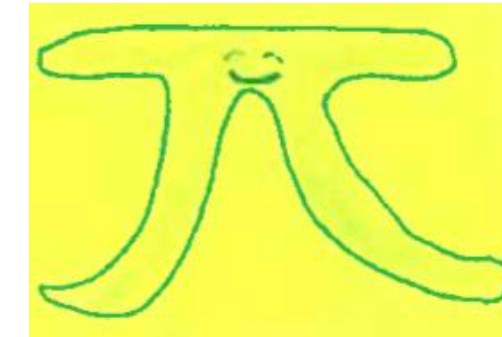
Significant academic and industry interests via fundamental breakthroughs

Milner,
Honda, NY



Binary Session Types

ESOP'98

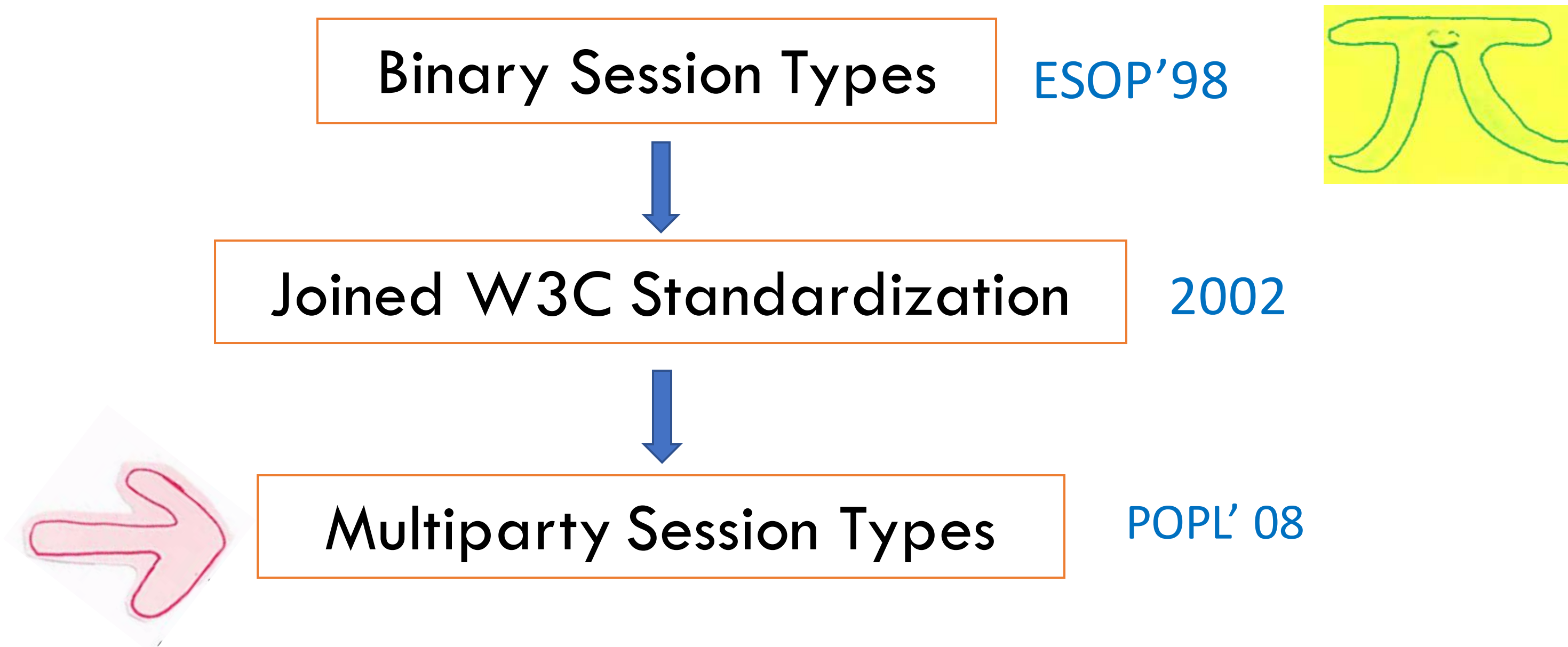


Joined W3C Standardization

2002

Why Session Types, Why Now?

Significant academic and industry interests via fundamental breakthroughs

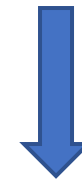
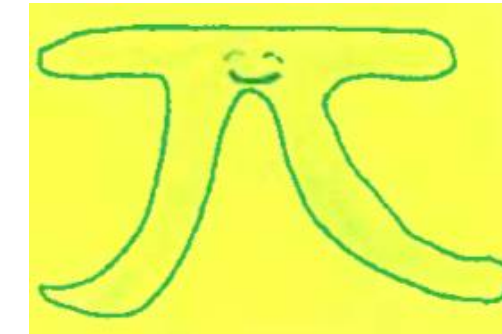


Why Session Types, Why Now?

Significant academic and industry interests via fundamental breakthroughs

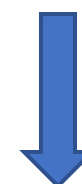
Binary Session Types

ESOP'98



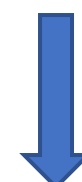
Joined W3C Standardization

2002



Multiparty Session Types

POPL' 08

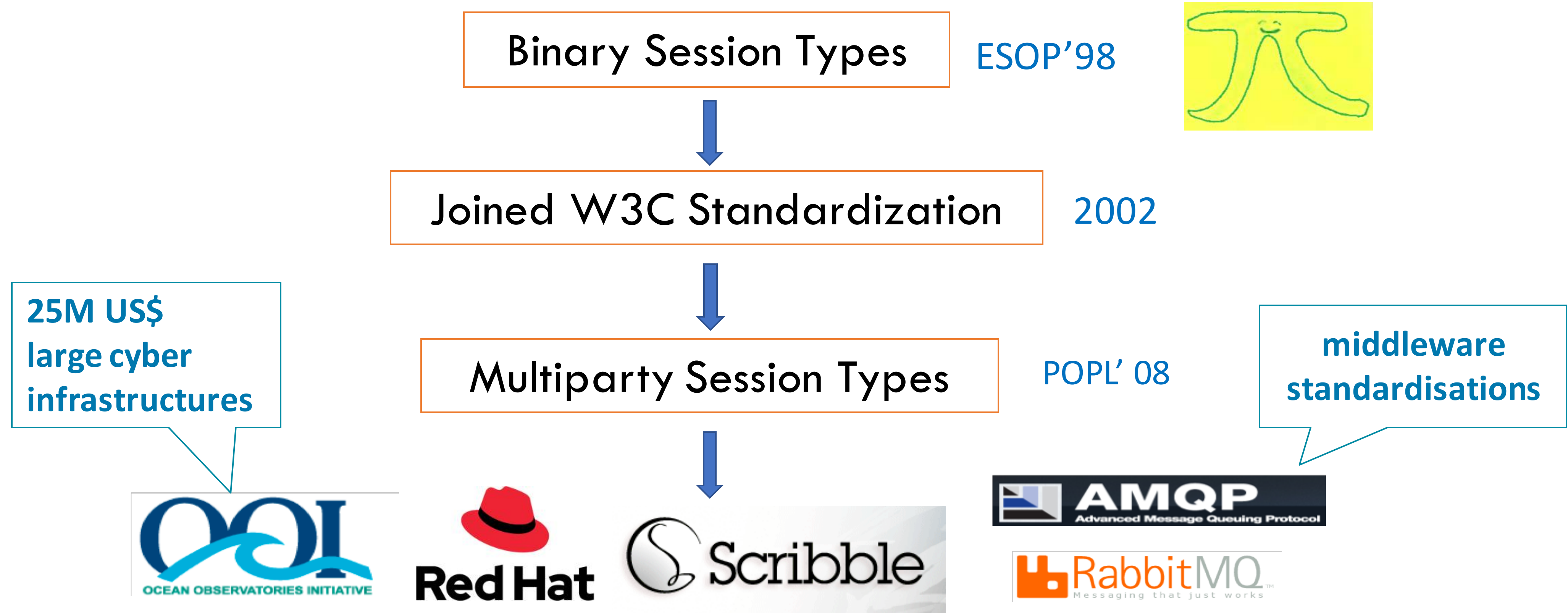


largest open source
company in the world



Why Session Types, Why Now?

Significant academic and industry interests via fundamental breakthroughs

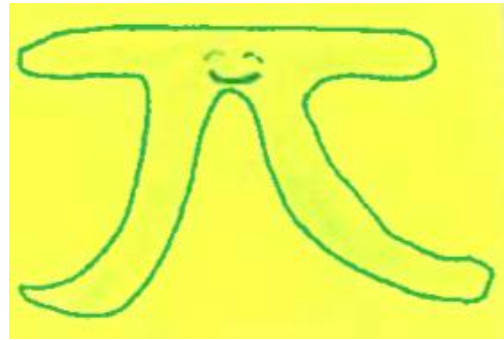


Why Session Types, Why Now?

Significant academic and industry interests via fundamental breakthroughs

Binary Session Types

ESOP'98



Joined W3C Standardization

2002

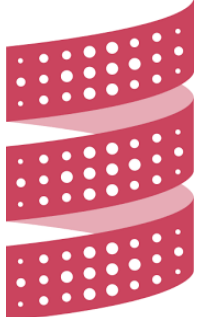


Multiparty Session Types

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

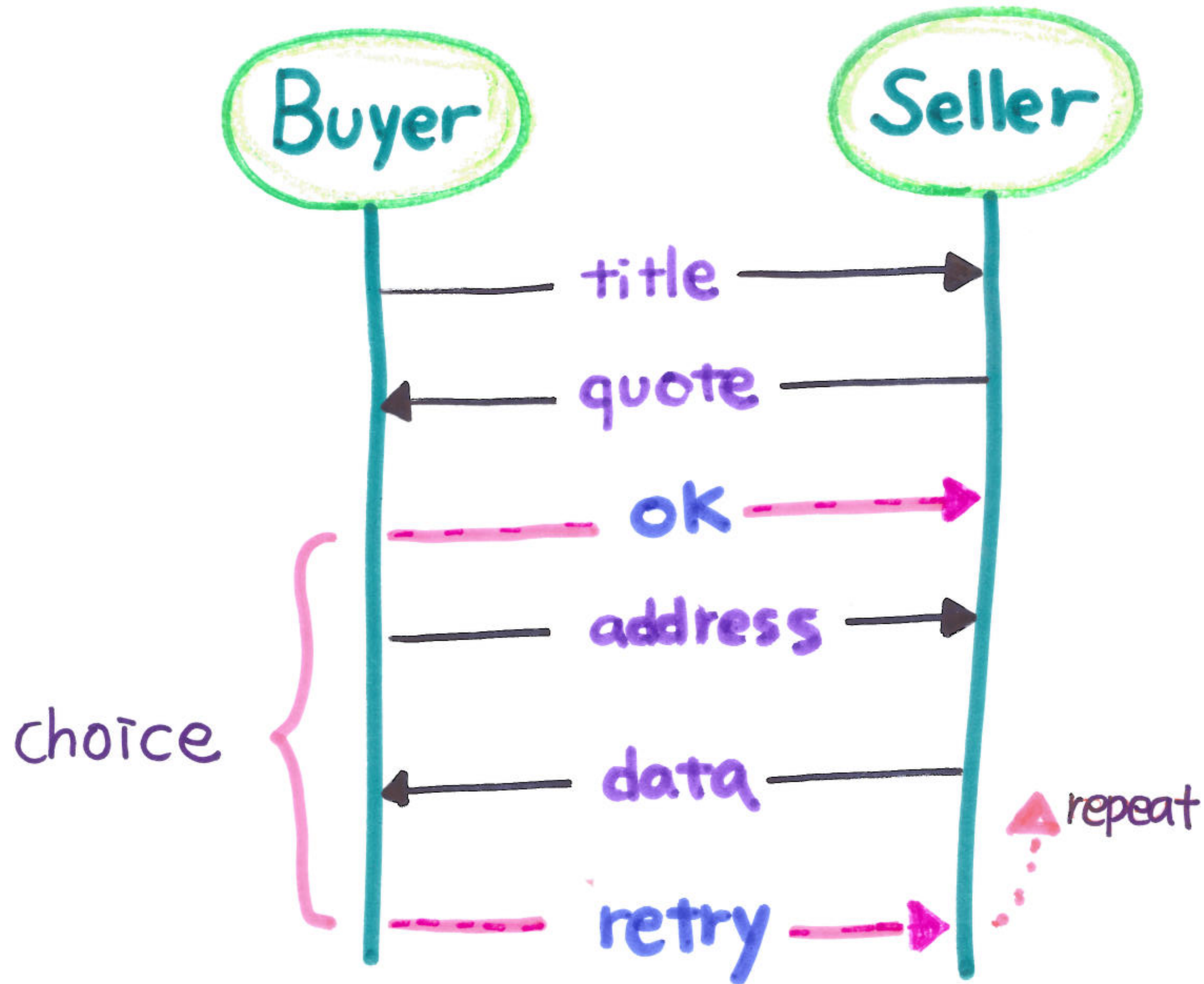
POPL' 08

POPL Influential Paper Award 2018

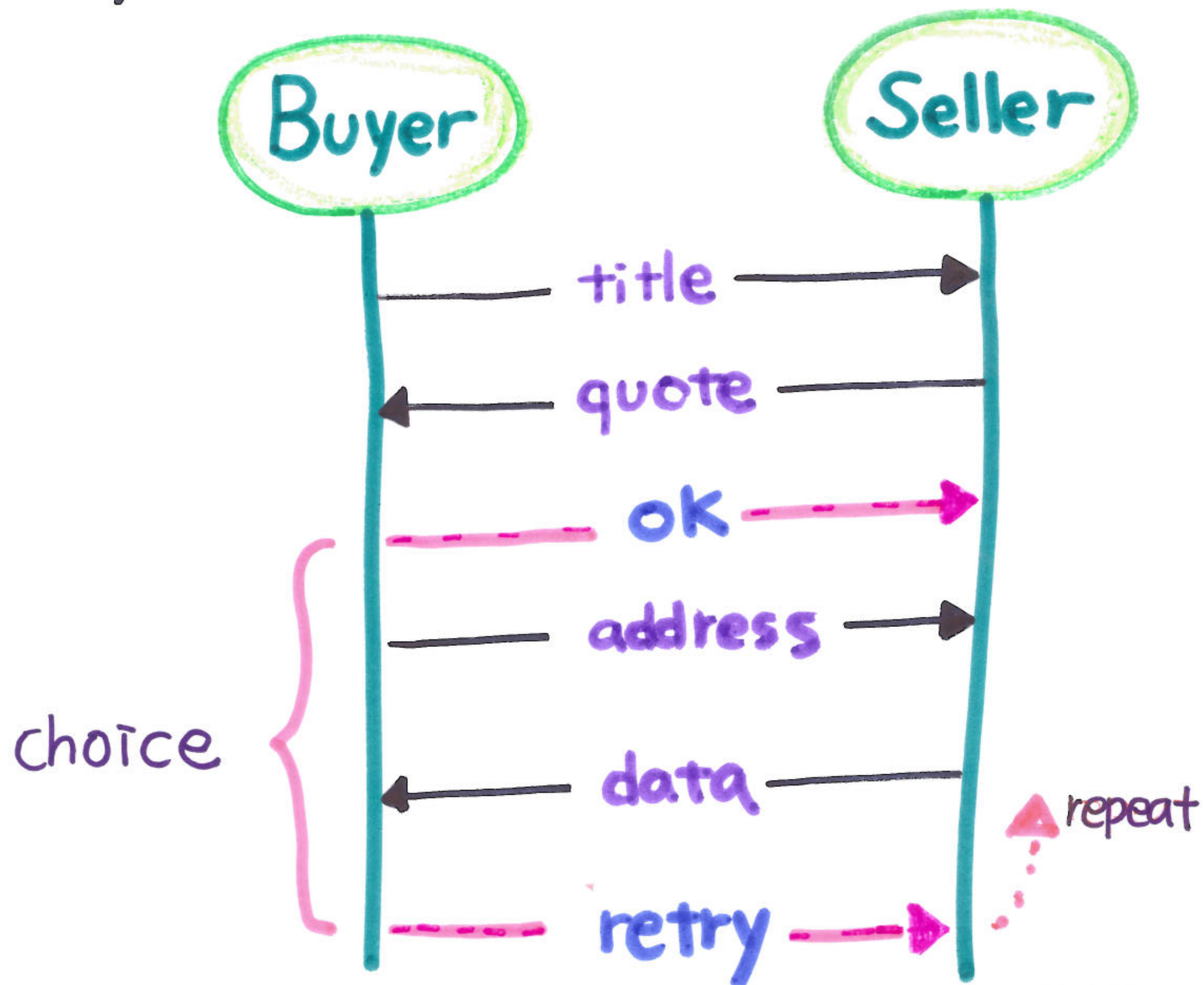


A collection of logos for various programming languages and frameworks, including Java, Go, OOI (Ocean Observatories Initiative), Red Hat, Scribble, AMQP (Advanced Message Queuing Protocol), RabbitMQ, Python, Scala, akka, Erlang, MPI, and OCaml.

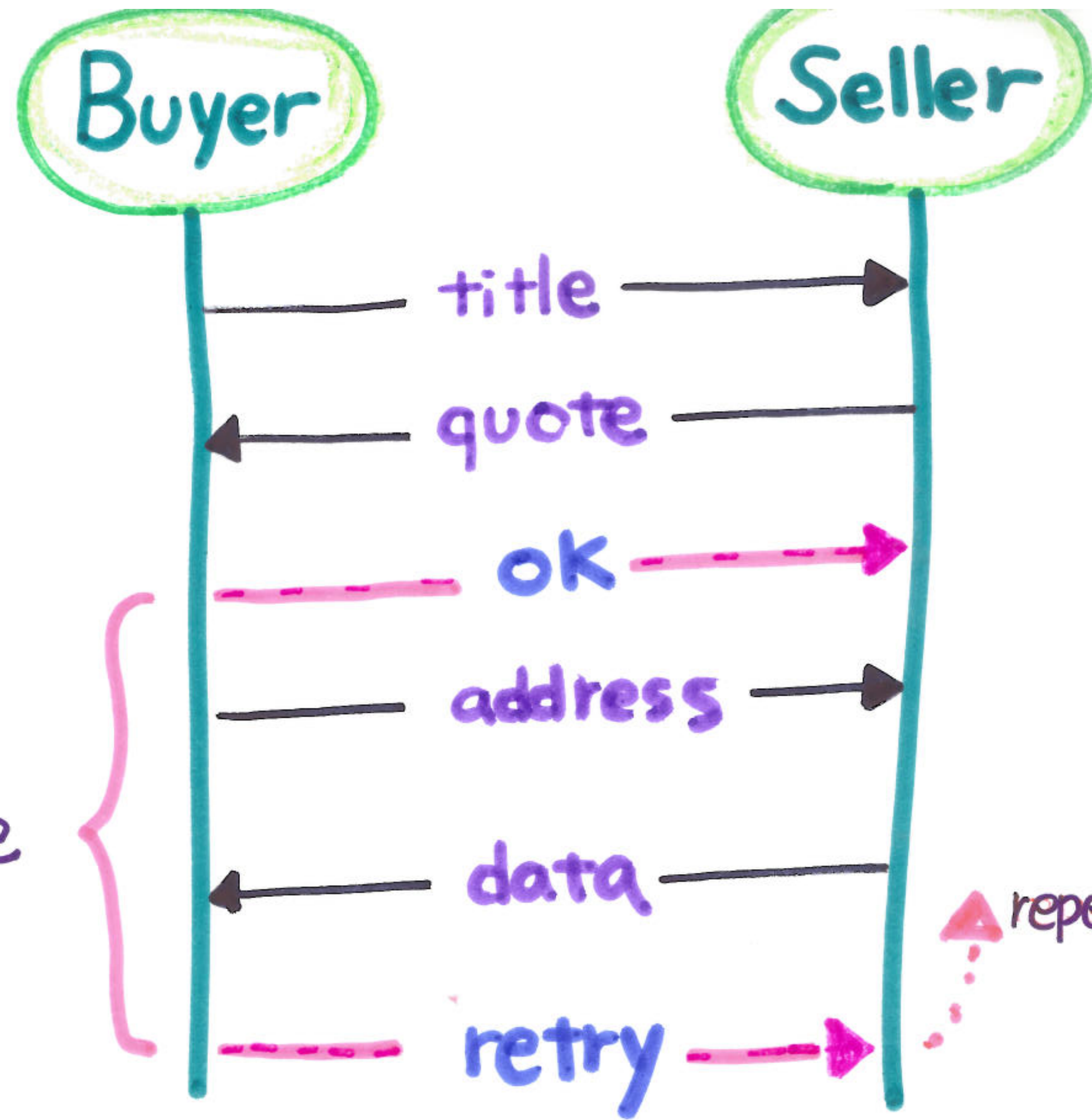
Binary Session Types: Buyer - Seller Protocol



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 }

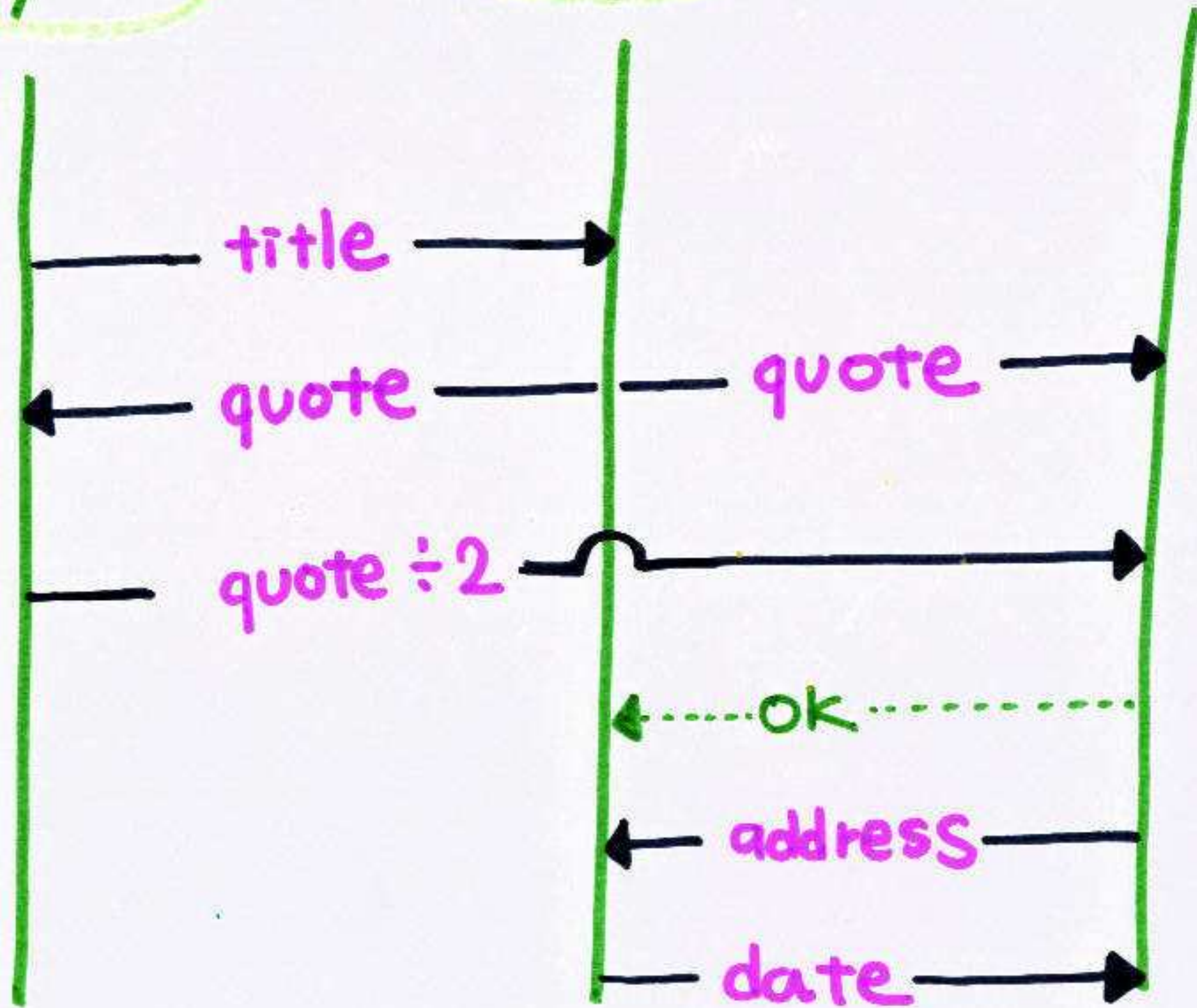
nt? Title ; ! Quote ; ? { ok: ? Add ; ! Date, **retry**: t }

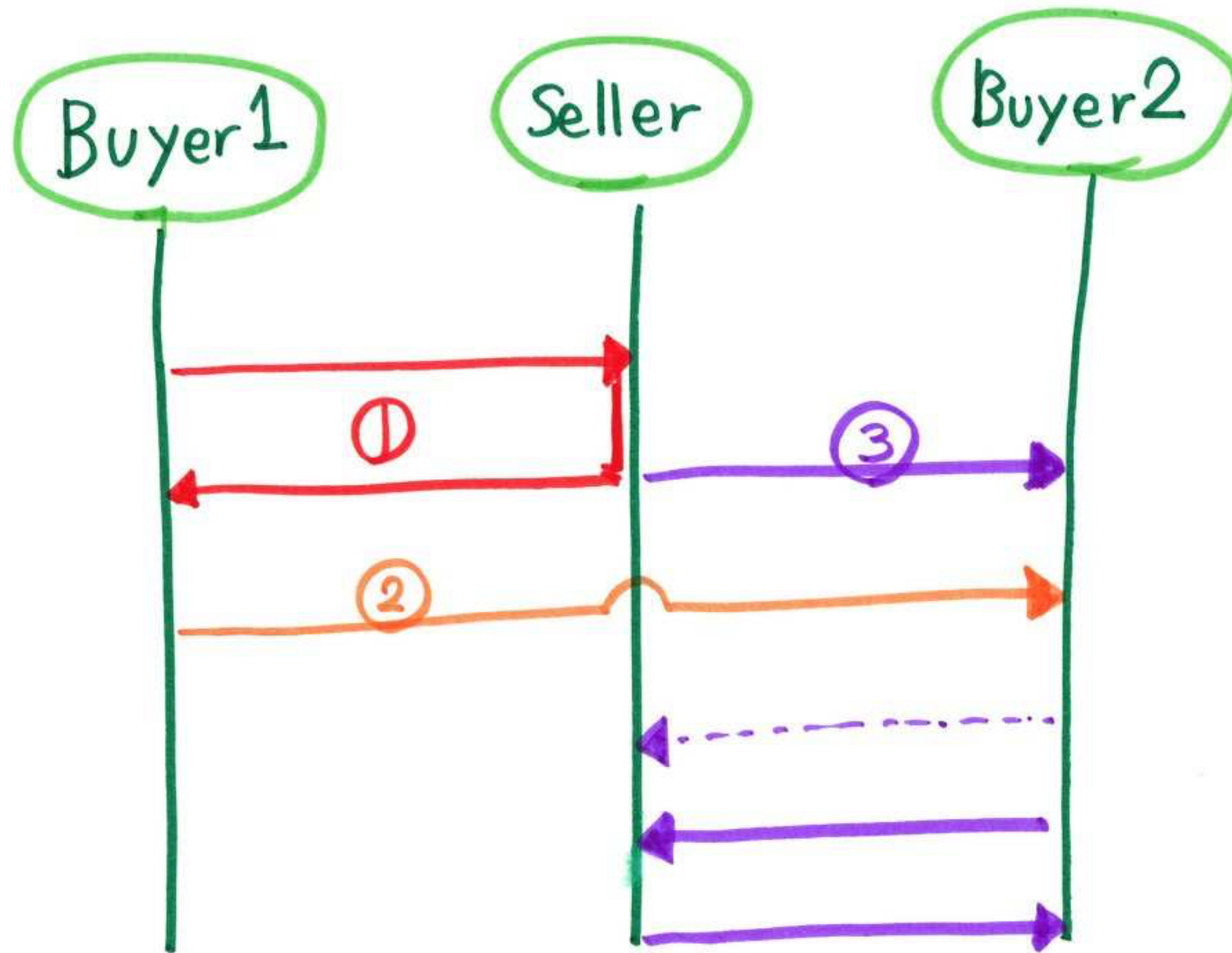
Multiparty Session Types

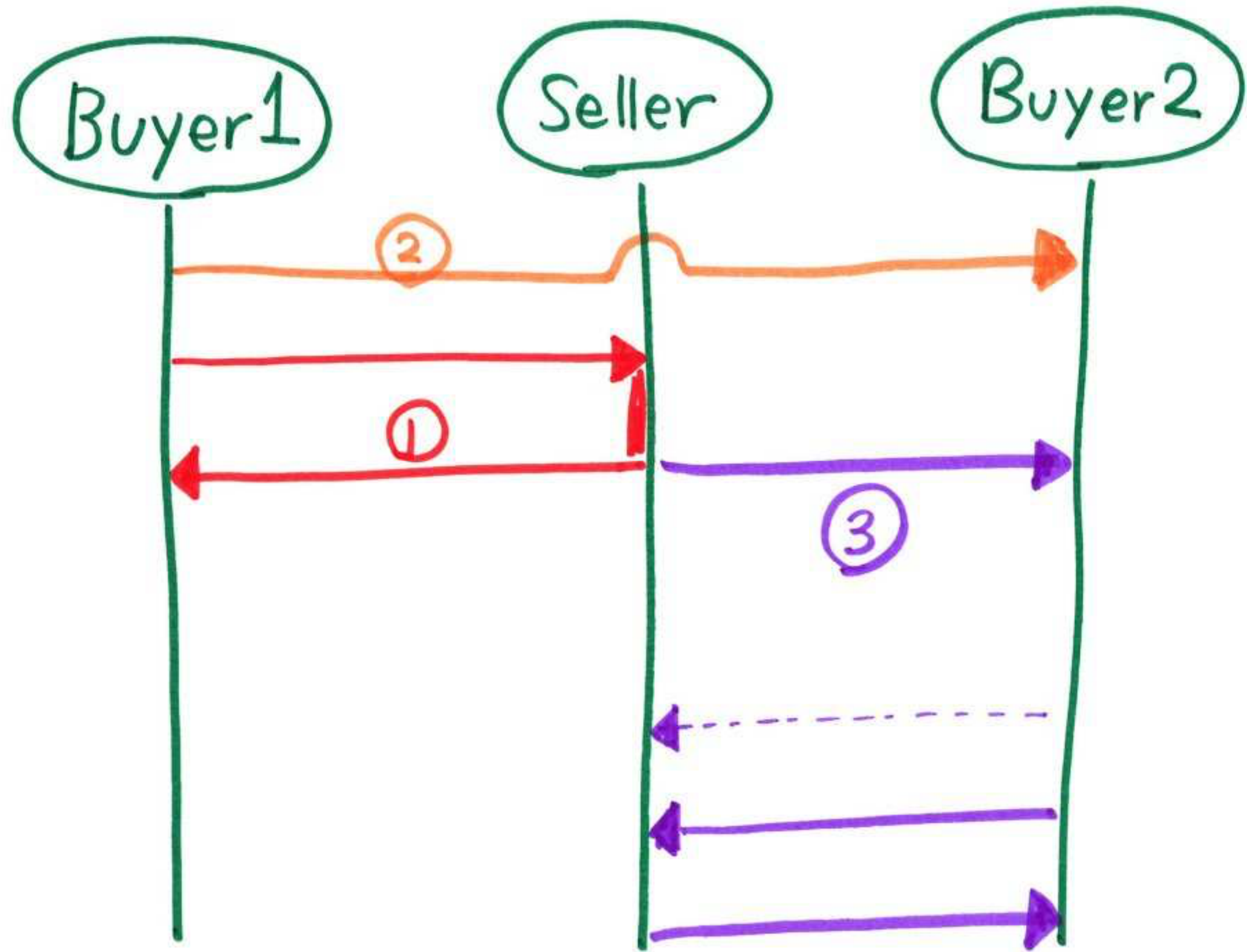
Buyer1

Seller

Buyer2





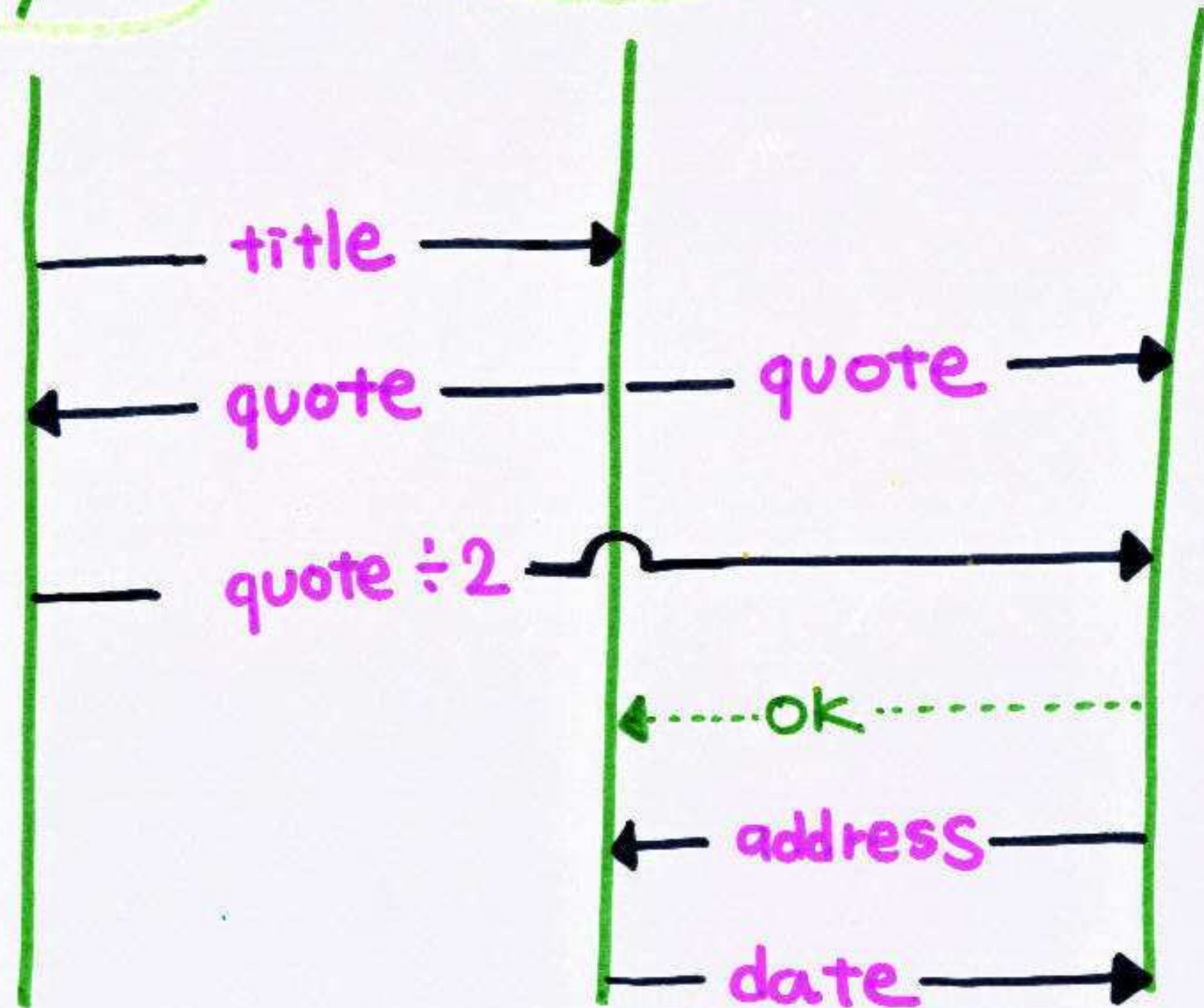


Multiparty Session Types

Buyer1

Seller

Buyer2



Alice

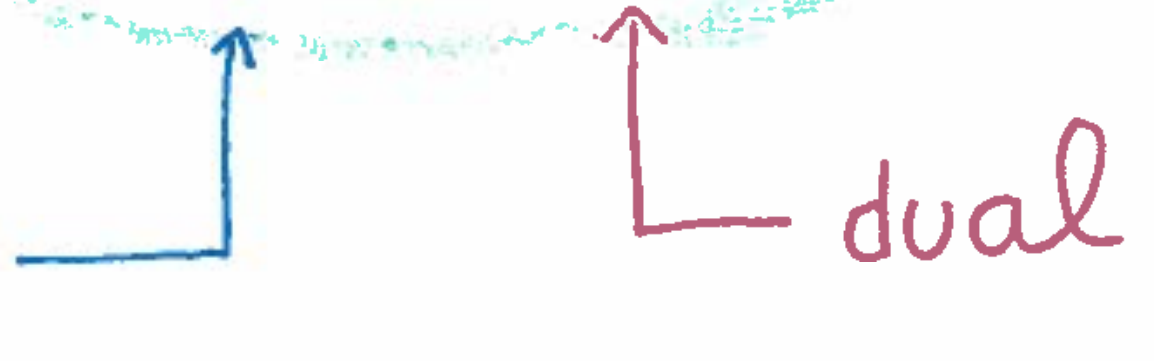
Bob

Carol

CA?c ; AB!a

AB?a ; BC!b

BC?b ; CA!c

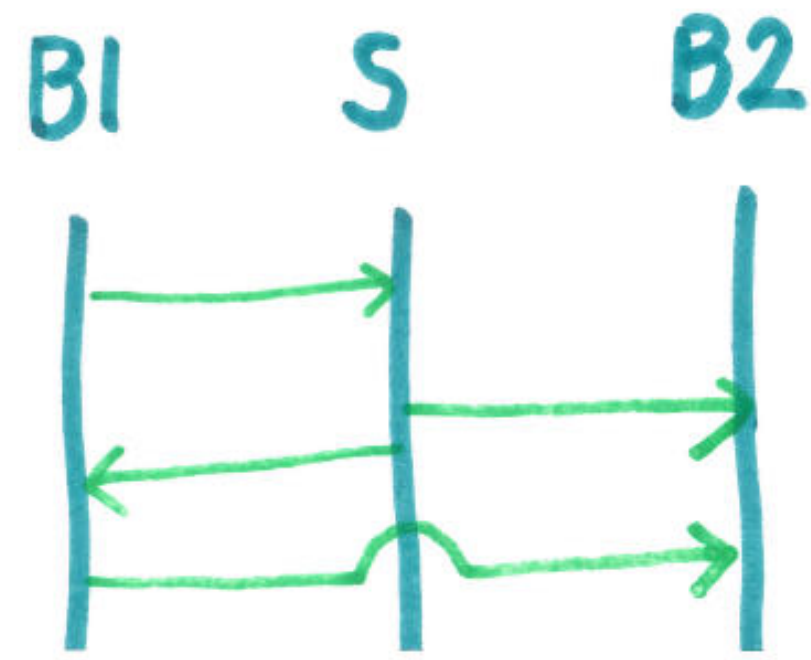


3 dual pairs

If you use binary Session Types ...

Deadlock!

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



ⓐ

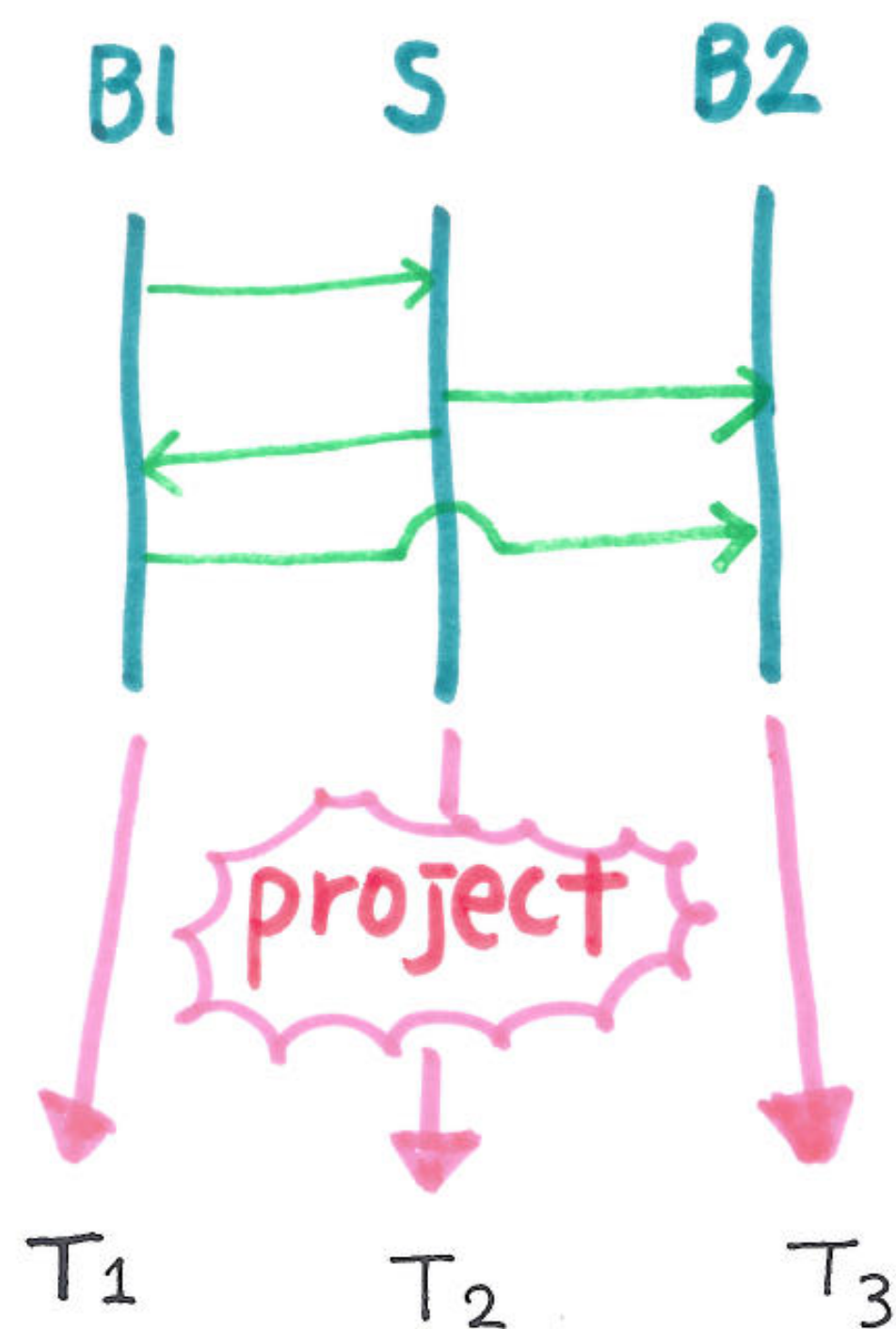
BI \rightarrow S Int.

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

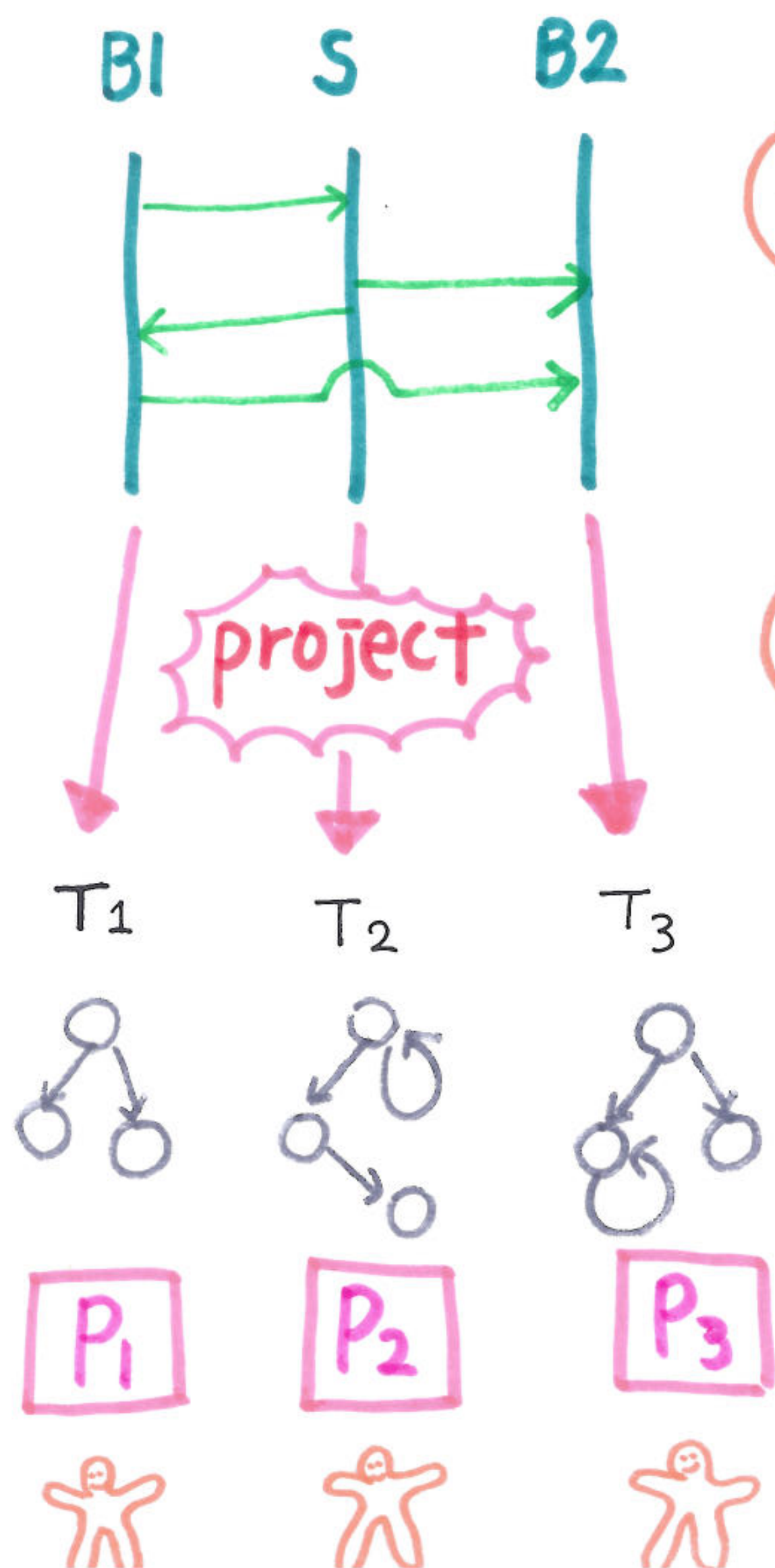
STEP 1
Write Global Type

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

STEP 2
Project to Local Types

Multi party Session Types

[Honda, Yoshida, Carbone 2008]



(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

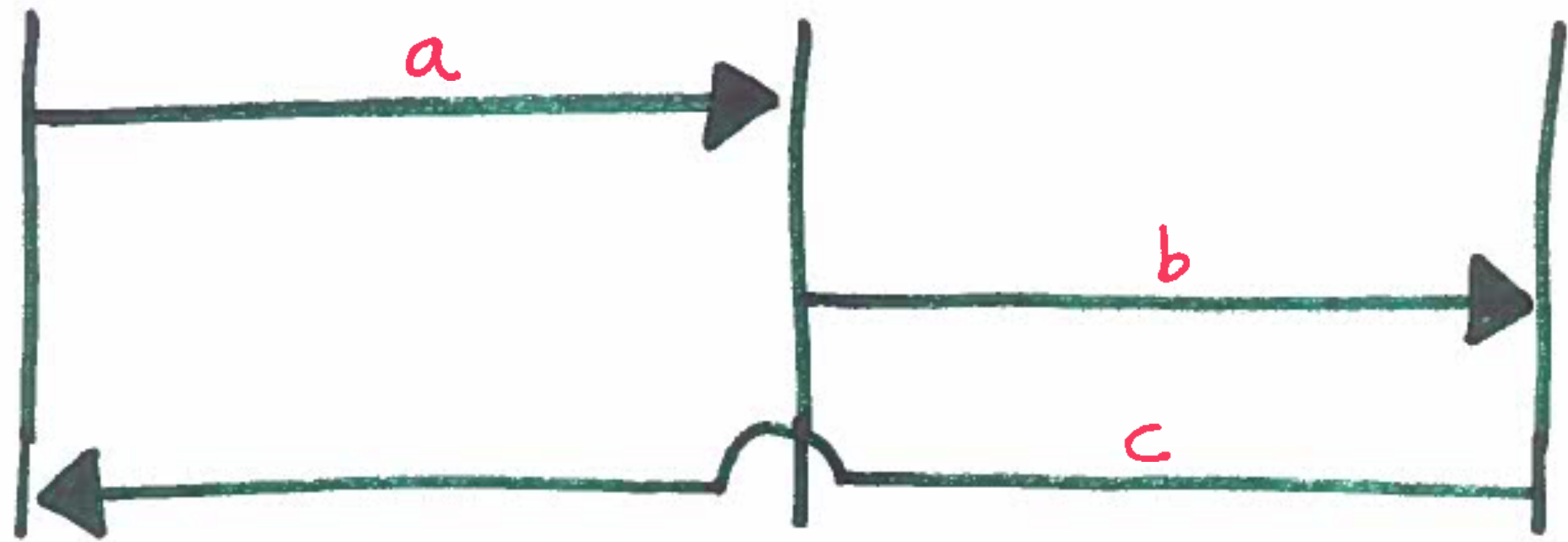
STEP 3

- Static Check
- Generate Code
- Run-time check

Alice

Bob

Carol



Global Type

Alice $AB!a; CA?c$

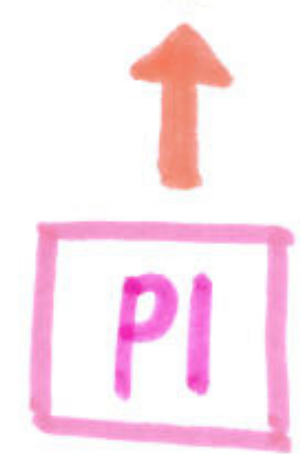
Bob $AB?a; BC!b$

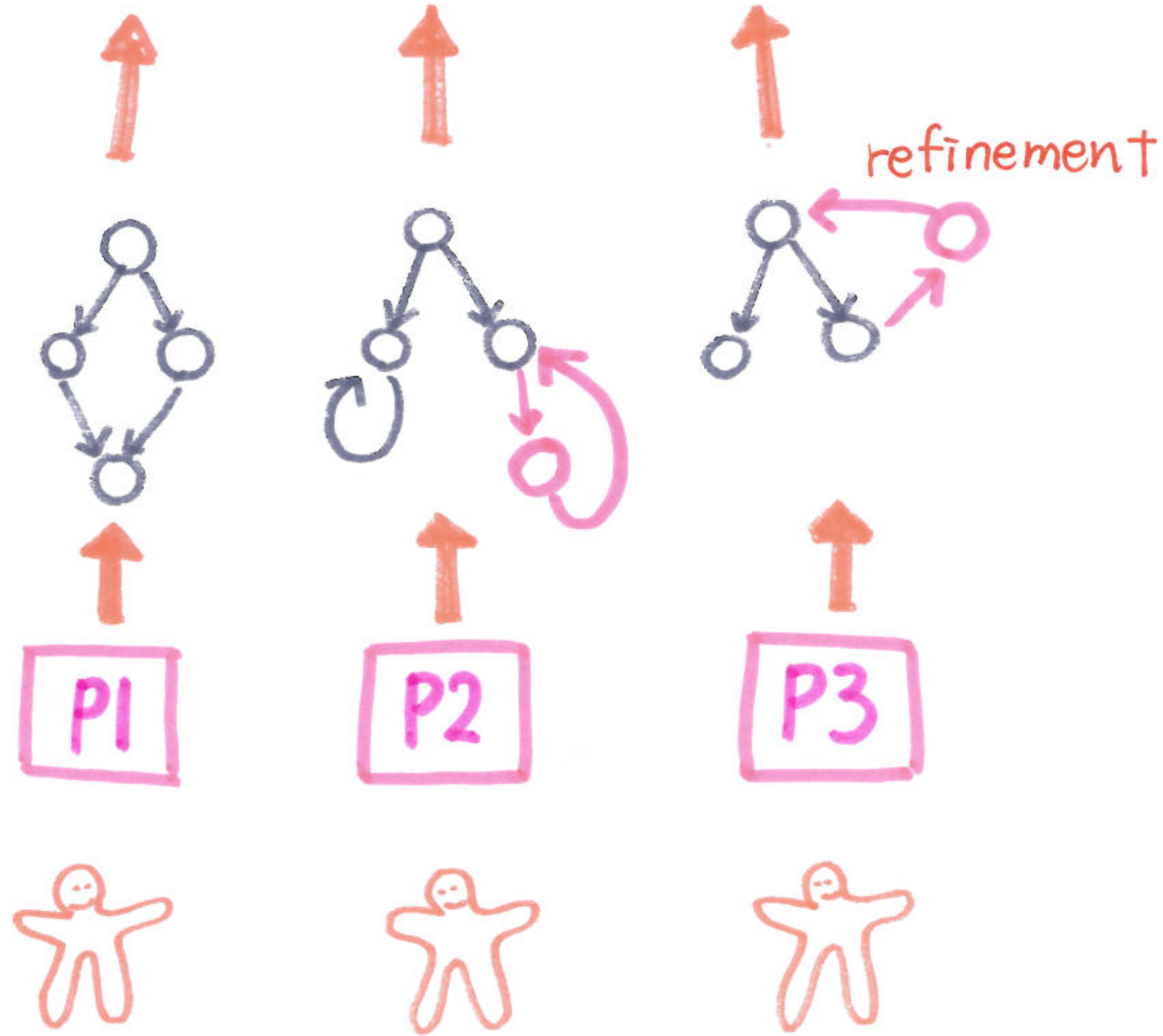
Carol $BC?b; CA!c;$

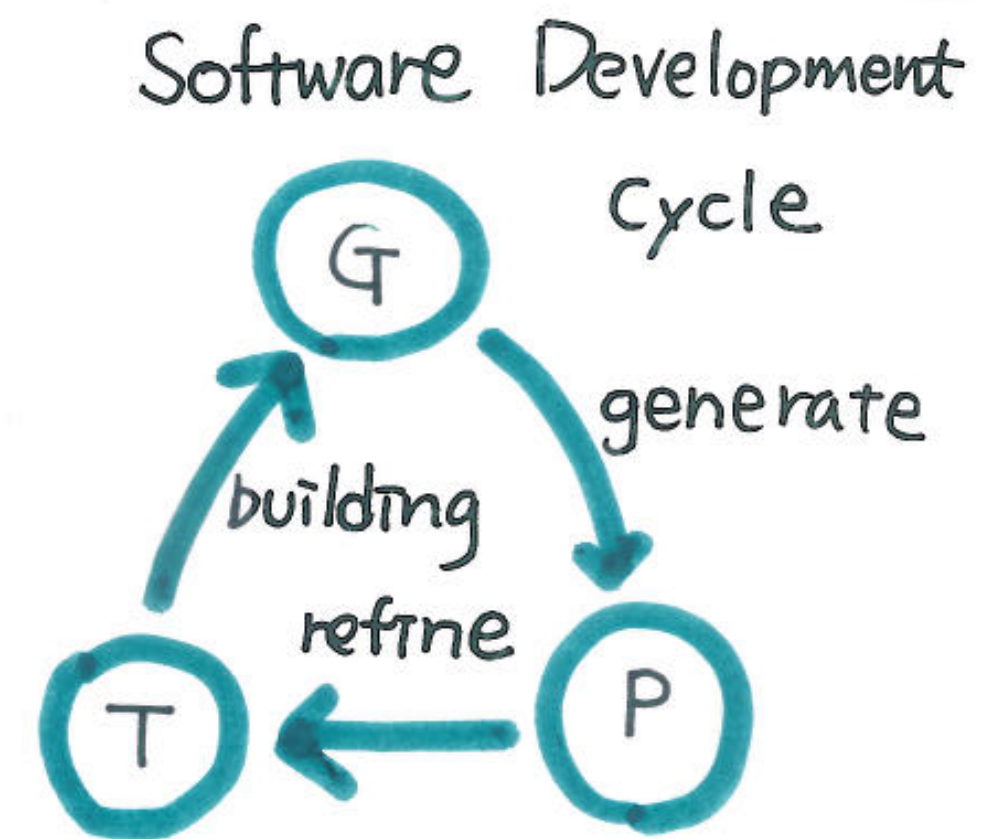
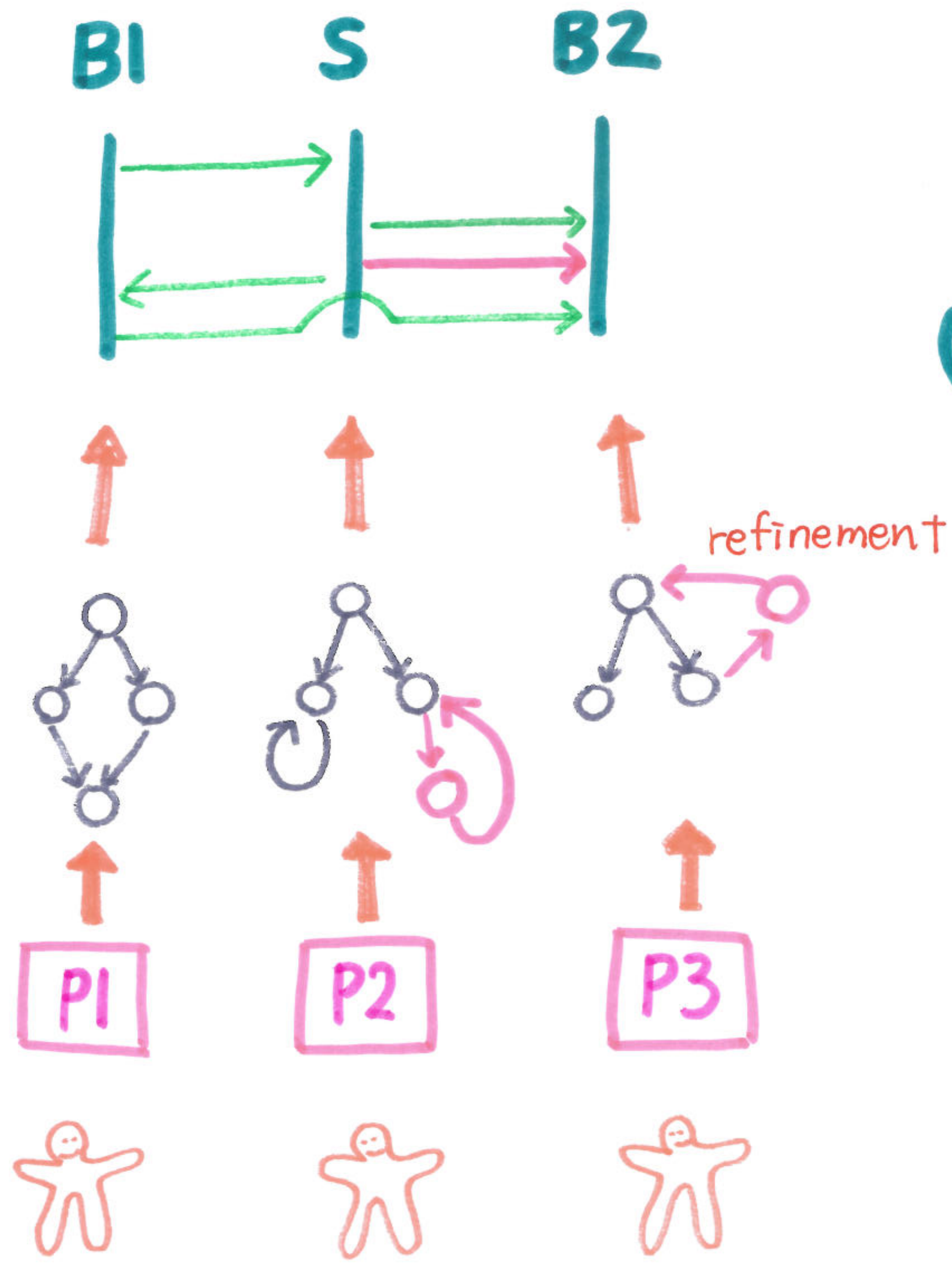


NO Deadlock

LOCAL TYPES







- Optimisation
- refinement
- inference
- Testing

Global Types

$G ::= P \rightarrow P' : \{ a_j . G_j \}_{j \in J}$

| mt. G

| t

| end

P, P', \dots Participant

a, b, c Σ Alphabet

Local Types

$T ::= P ! \{ a_j . T_j \}_{j \in J}$

| $P ? \{ a_j . T_j \}_{j \in J}$

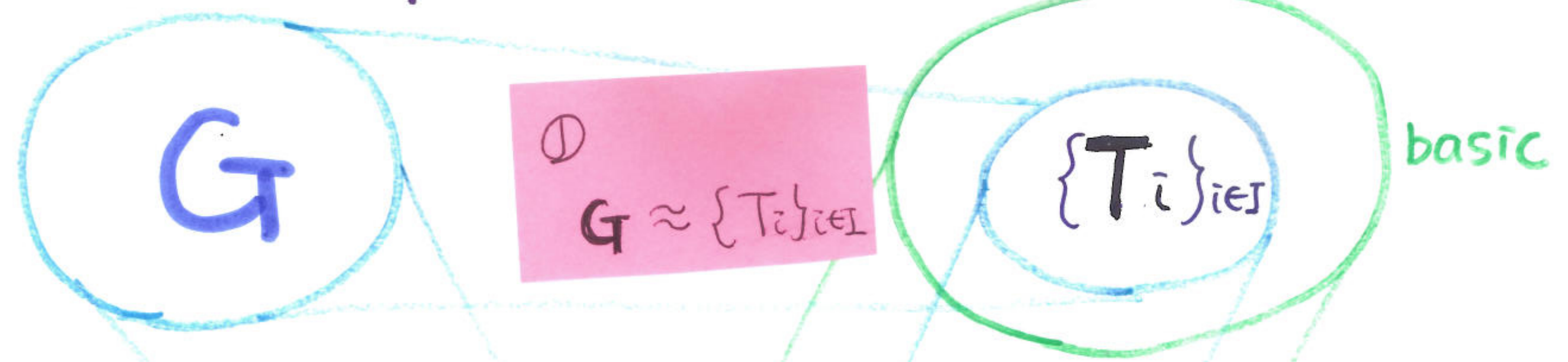
| mt. T

| t

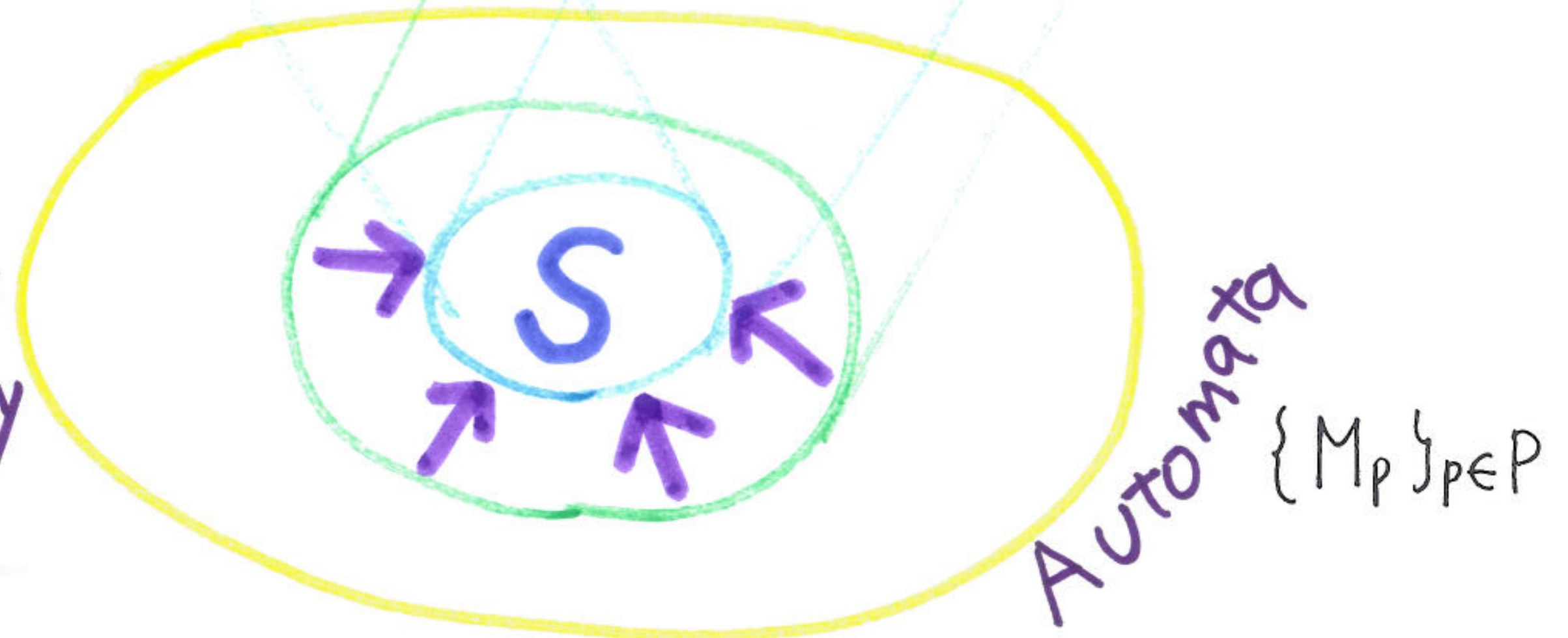
| end

Global Type

Local Types



← multiparty compatibility



Projection from Γ onto q $\Gamma \upharpoonright q$

$$(P \rightarrow P' : \{a_j, \Gamma_j\}_{j \in J}) \upharpoonright q$$

$$= \begin{cases} P' \upharpoonright \{a_j, \Gamma_j \upharpoonright q\}_{j \in J} & q = P \\ P \upharpoonright \{a_j, \Gamma_j \upharpoonright q\}_{j \in J} & q = P' \\ \Gamma \upharpoonright q = \Gamma_j \upharpoonright q & \forall j \in J \text{ otherwise} \end{cases}$$

Projection from Γ onto ϱ $\Gamma \upharpoonright \varrho$

$$(P \rightarrow P' : \{a_j. \Gamma_j\}_{j \in J}) \upharpoonright \varrho$$

$$= \begin{cases} P' ! \{a_j. \Gamma_j \upharpoonright \varrho\}_{j \in J} & \varrho = P \\ P ? \{a_j. \Gamma_j \upharpoonright \varrho\}_{j \in J} & \varrho = P' \\ \Gamma_1 \upharpoonright \varrho = \Gamma_j \upharpoonright \varrho & \forall j \in J \text{ otherwise} \end{cases}$$

Bad $A \rightarrow B : \{ \underline{a}. C \rightarrow D : \underline{c}, \underline{b}. C \rightarrow D : \underline{d} \}$

Good $A \rightarrow B : \{ \underline{a}. C \rightarrow D : \underline{c}, \underline{b}. C \rightarrow D : \underline{c} \}$

LTS Local Types $l ::= pq!a \mid pq?a$

$$q! \{a_i. T_i\}_{i \in I} \xrightarrow{pq!a_i} T_i \quad q? \{a_i. T_i\}_{i \in I} \xrightarrow{qp?a_i} T_i$$

$$\frac{T[mt.T/t] \xrightarrow{l} T'}{mt.T \xrightarrow{l} T'}$$

$(\vec{T}; \vec{w})$

W_{pq}
queue

Send $T_p \xrightarrow{pq!l} T'_p \Rightarrow$

$$(\dots T_p \dots ; \dots W_{pq} \dots) \xrightarrow{pq!l} (\dots T'_p \dots ; \dots W_{pq} \cdot l \dots)$$

Receive $T_q \xrightarrow{pq?l} T'_q \Rightarrow$

$$(\dots T_q \dots ; \dots l \cdot W_{pq} \dots) \xrightarrow{pq?l} (\dots T'_q \dots ; \dots W_{pq} \dots)$$

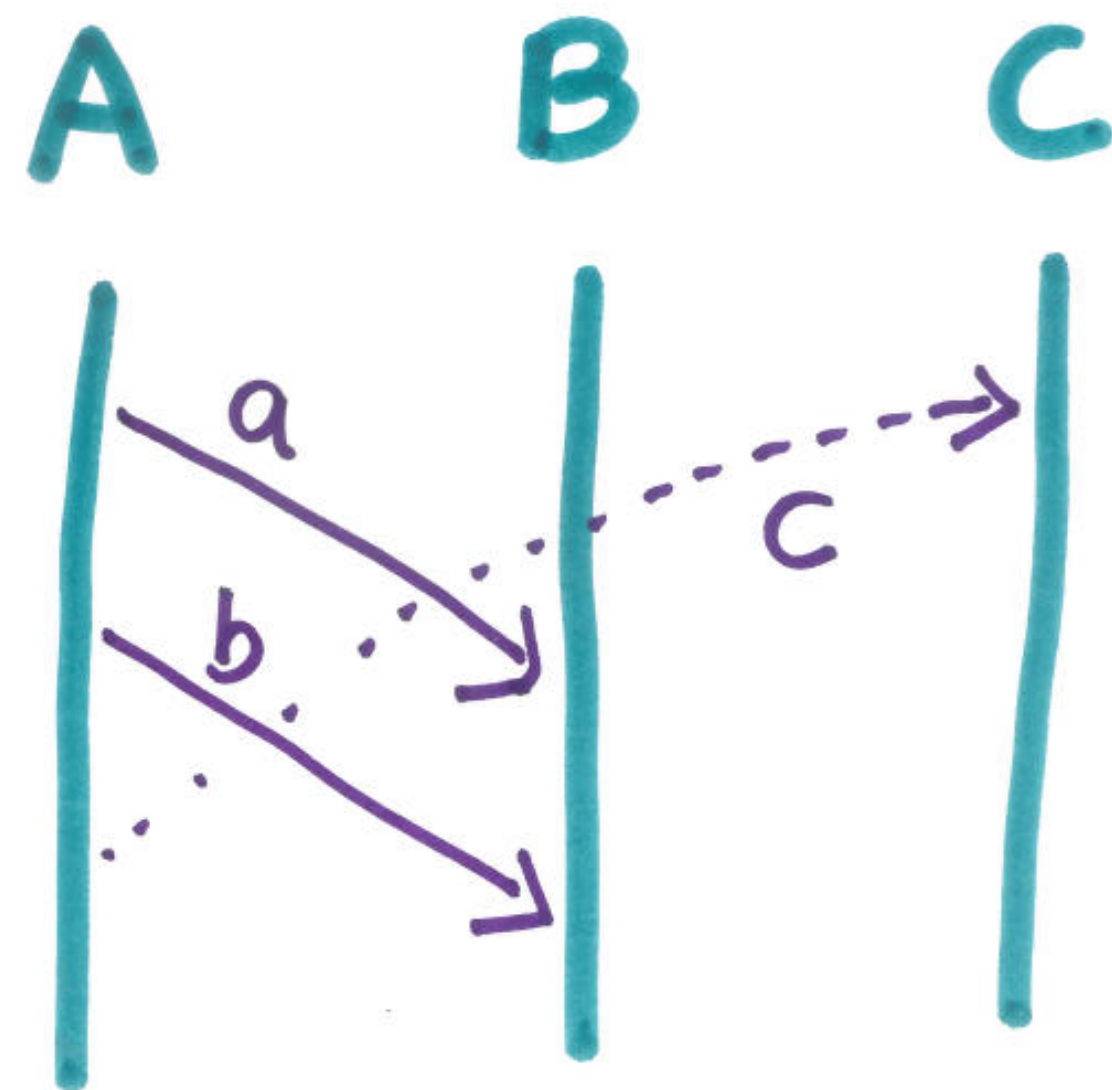
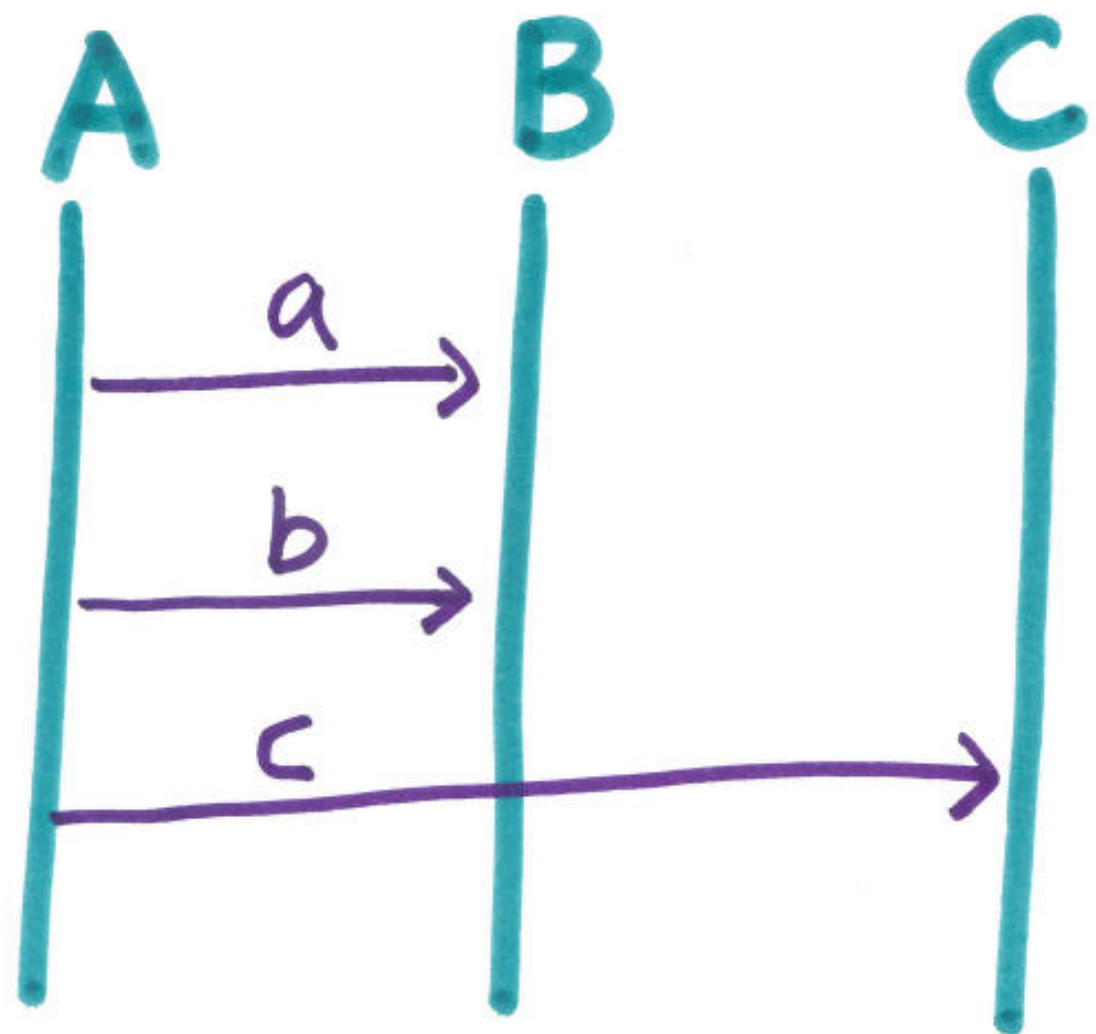
LTS Global Types

$$p \rightsquigarrow p' : \bar{j} \{a_i. G_i\}_{i \in I}$$

put $p \rightarrow p' : \{a_i. G_i\}_{i \in I} \xrightarrow{pp'!a_j} p \rightsquigarrow p' : \bar{j} \{a_i. G_i\}_{i \in I}$

get $p \rightsquigarrow p' : \bar{j} \{a_i. G_i\}_{i \in I} \xrightarrow{pp'?a_j} G_j$

$$\frac{\forall j \in I \quad \underline{G_j} \xrightarrow{\ell} \underline{G'_j} \quad p, q \notin \text{sub}(\ell) \quad q \notin \text{sub}(\ell) \quad \forall i \in I \setminus \{j\} \quad G'_i = G_i}{p \rightarrow p' : \{a_i. \underline{G_i}\}_{i \in I} \xrightarrow{\ell} p \rightarrow p' : \{a_i. \underline{G'_i}\}_{i \in I} \quad p \rightsquigarrow q : \bar{j} \{a_i. \underline{G_i}\}_{i \in I} \xrightarrow{\ell} p \rightsquigarrow q : \bar{j} \{a_i. \underline{G'_i}\}_{i \in I}}$$



$A \rightarrow B:a . A \rightarrow B:b . A \rightarrow C:c$

↓ $AB!a$ OUT

$A \rightsquigarrow B:a . A \rightarrow B:b . A \rightarrow C:c$

↓ $AB?a$ IN

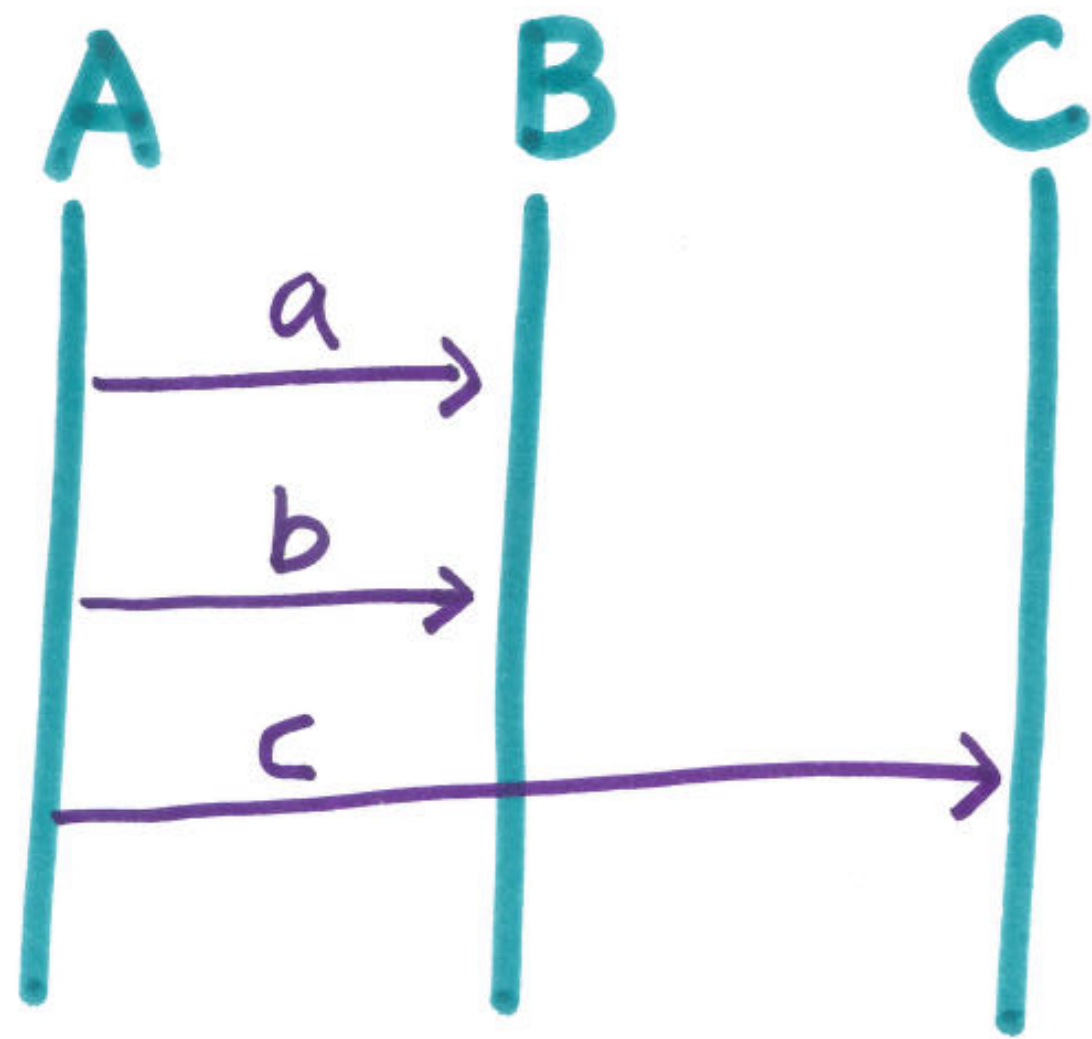
$A \rightarrow B:b . A \rightarrow C:c$

↓ $AB!b$ OUT

$A \rightsquigarrow B:b . A \rightarrow C:c$

↓ $AB?b$ IN

$A \rightarrow C:c$



$A \rightarrow B:a. A \rightarrow B:b. A \rightarrow C:c$

↓ $AB!a$ OUT

$A \rightsquigarrow B:a. A \rightarrow B:b. A \rightarrow C:c$

↓ $AB?a$ IN

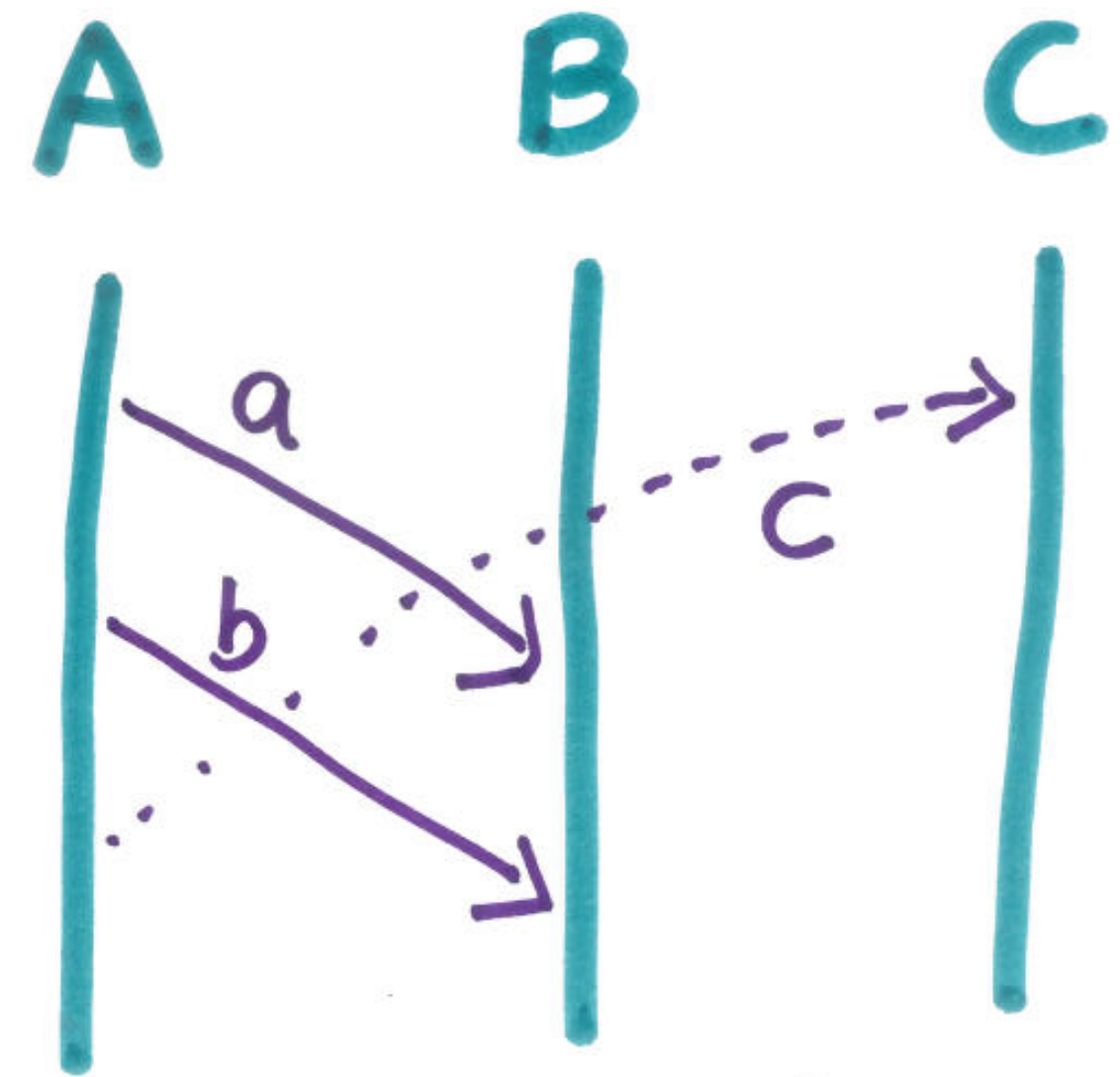
$A \rightarrow B:b. A \rightarrow C:c$

↓ $AB!b$ OUT

$A \rightsquigarrow B:b. A \rightarrow C:c$

↓ $AB?b$ IN

$A \rightarrow C:c$



$A \rightarrow B:a. A \rightarrow B:b. A \rightarrow C:c$

↓ $AB!a$ OUT

↓ $AB!b$ OUT

↓ $AC!c$ OUT

$A \rightsquigarrow B:a. A \rightsquigarrow B:b. A \rightsquigarrow C:c$

↓ $AC?c$ IN

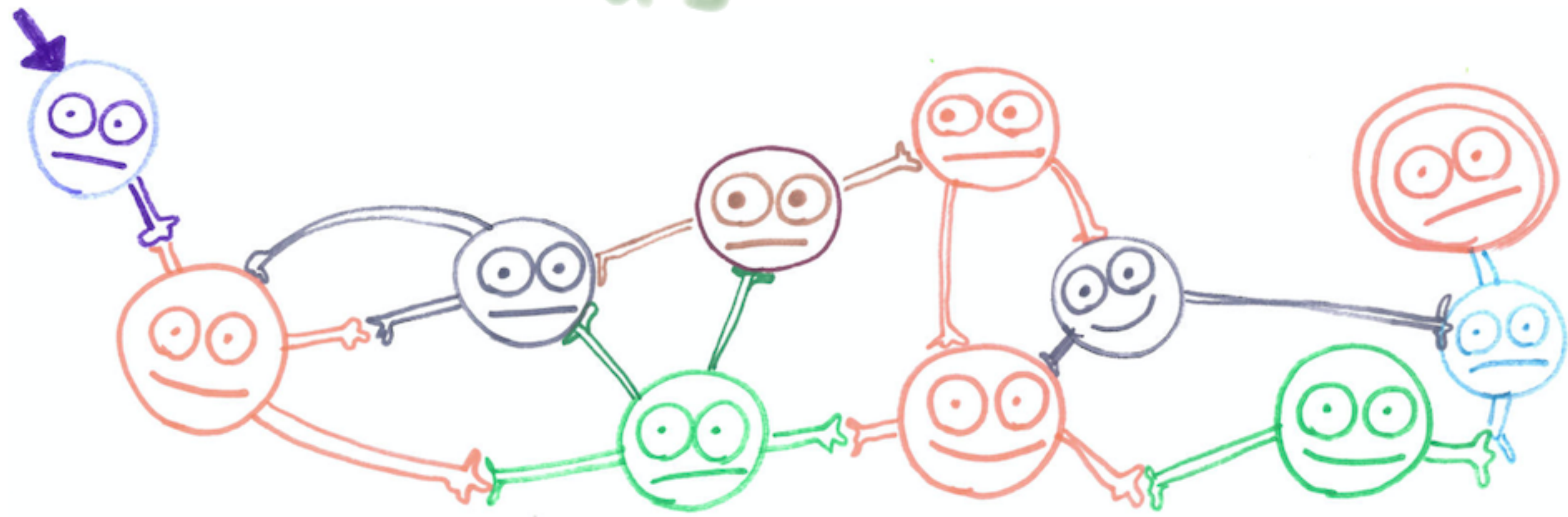
$A \rightsquigarrow B:a. A \rightsquigarrow B:b$

↓ $AB?a$ IN

$A \rightsquigarrow B:b$

Multiparty Session Types

as



?? Communicating Automata ??

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

Σ a finite alphabet of messages

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

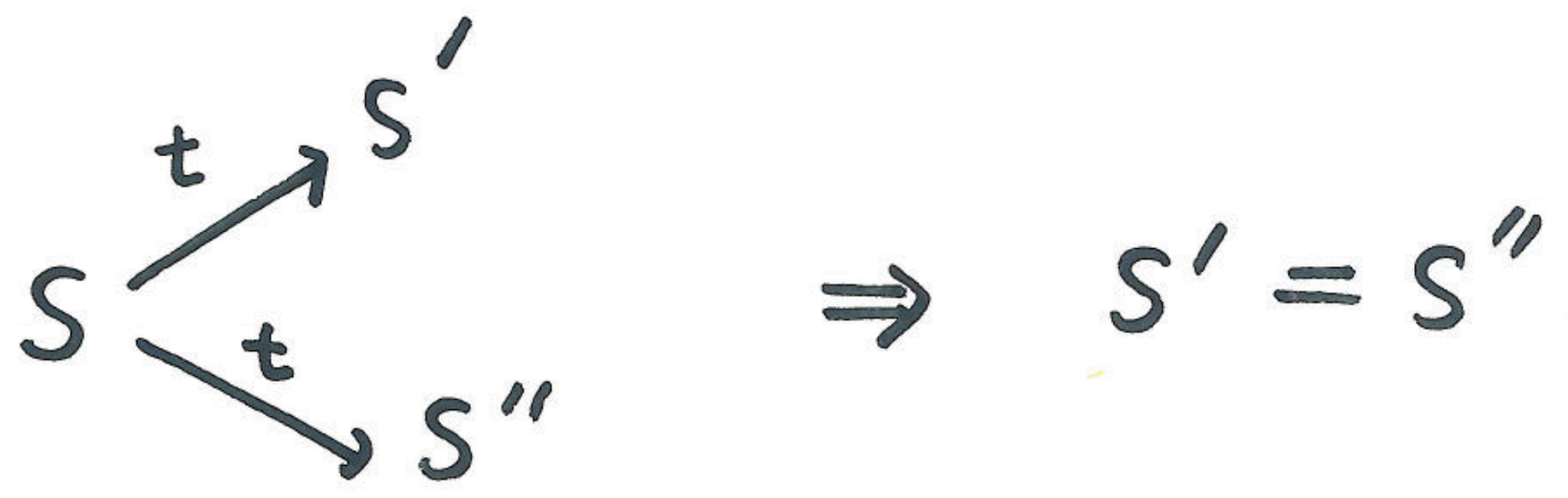
Def CS $S = (M_p)_{p \in \text{Participant}}$

$S \xrightarrow{t} S'$ 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}

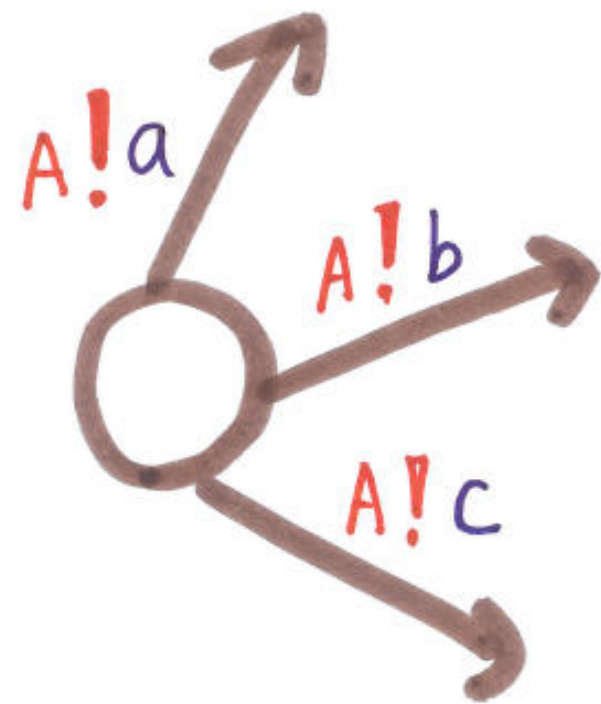
Receive $(\dots q_q \dots; \dots l \cdot W_{pq} \dots) \xrightarrow{pq?l} (\dots q'_q \dots; \dots W_{pq} \dots)$

Deterministic CFM



Basic CFSMs

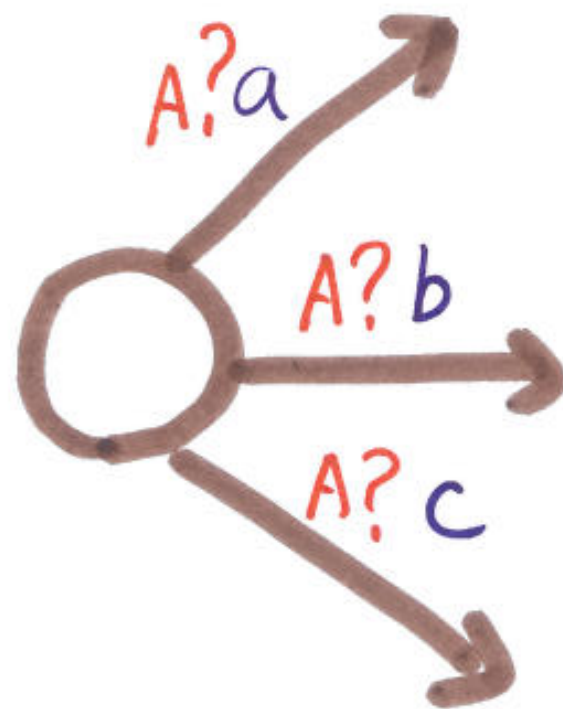
A CFSM is **Basic** if **deterministic**
directed, has **no mixed states**



sending



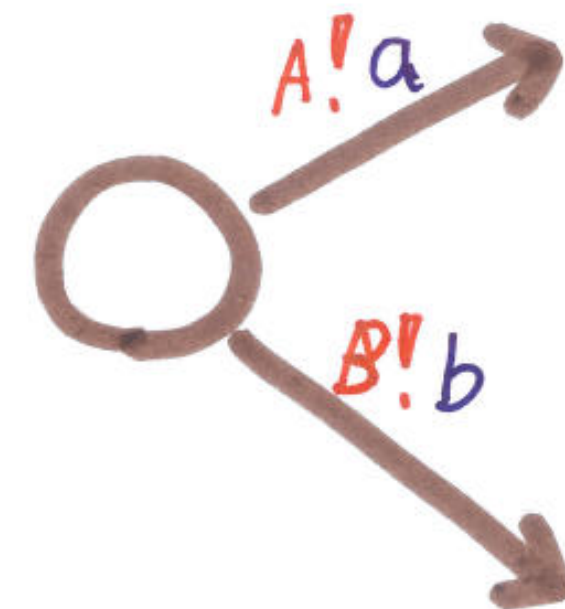
$T = A!\{a, b, c\}$



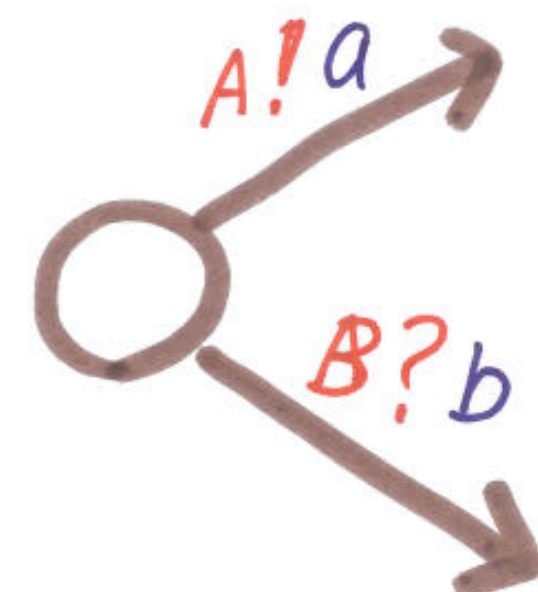
receiving



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

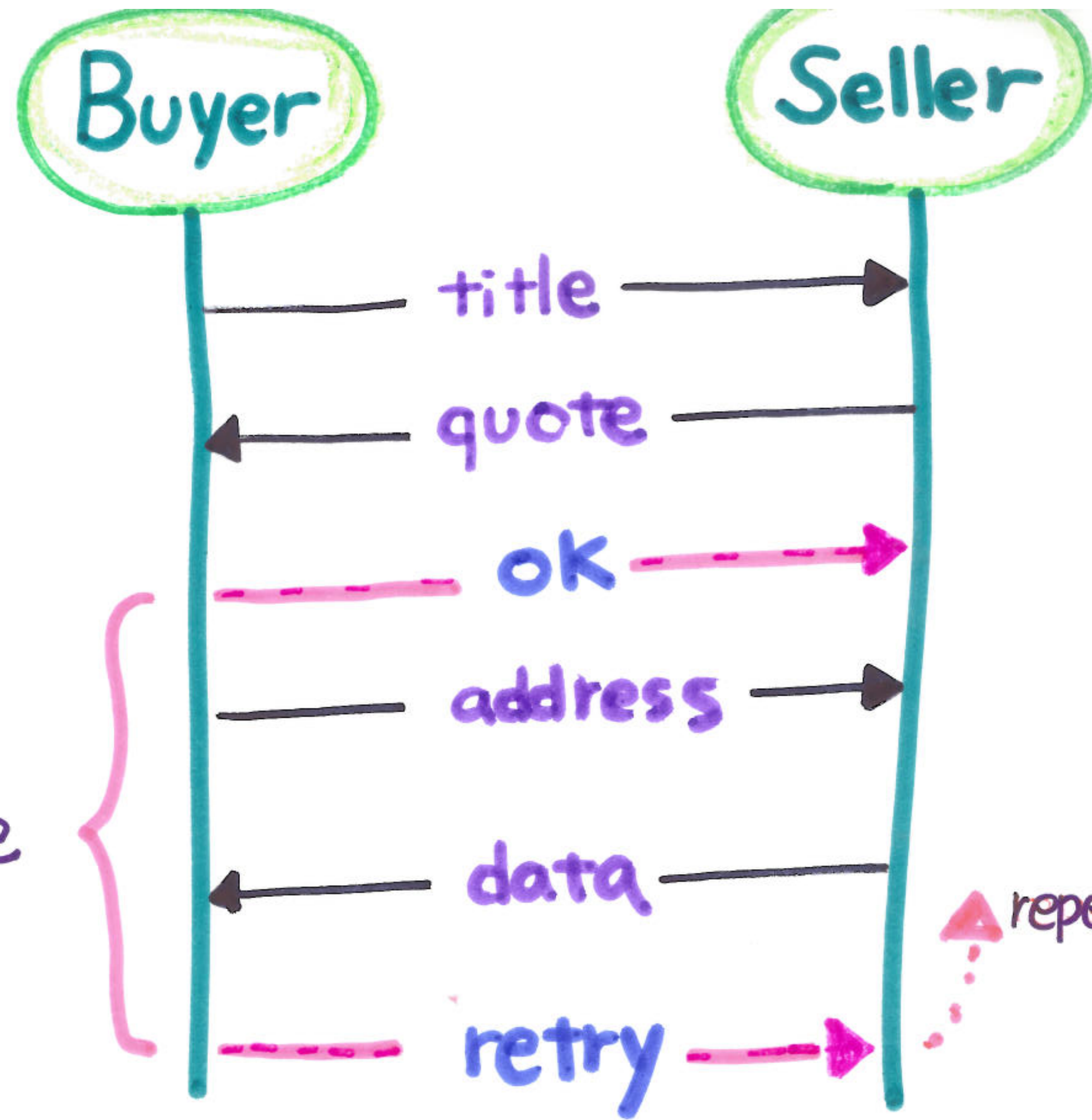


non
directed



mixed



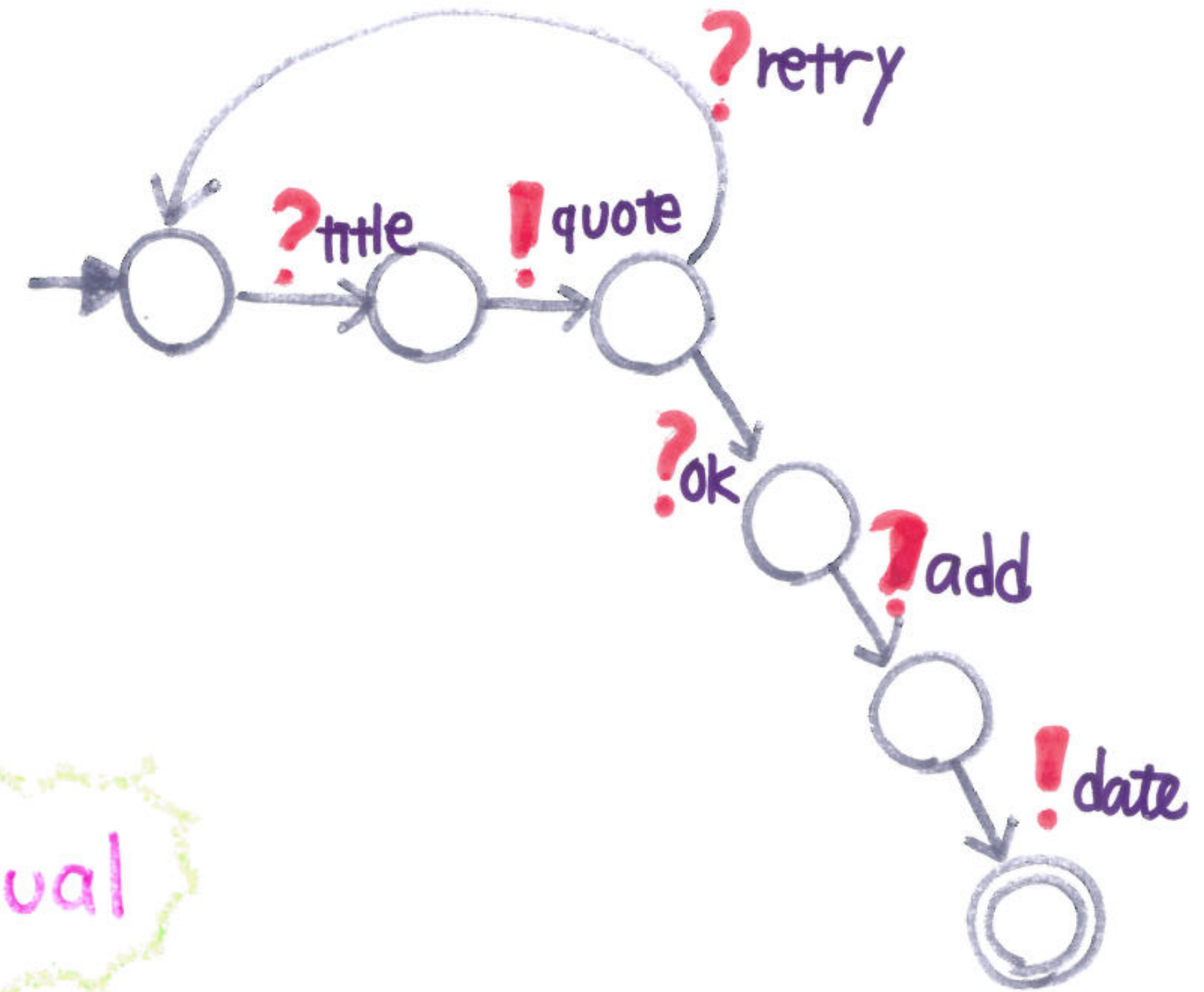
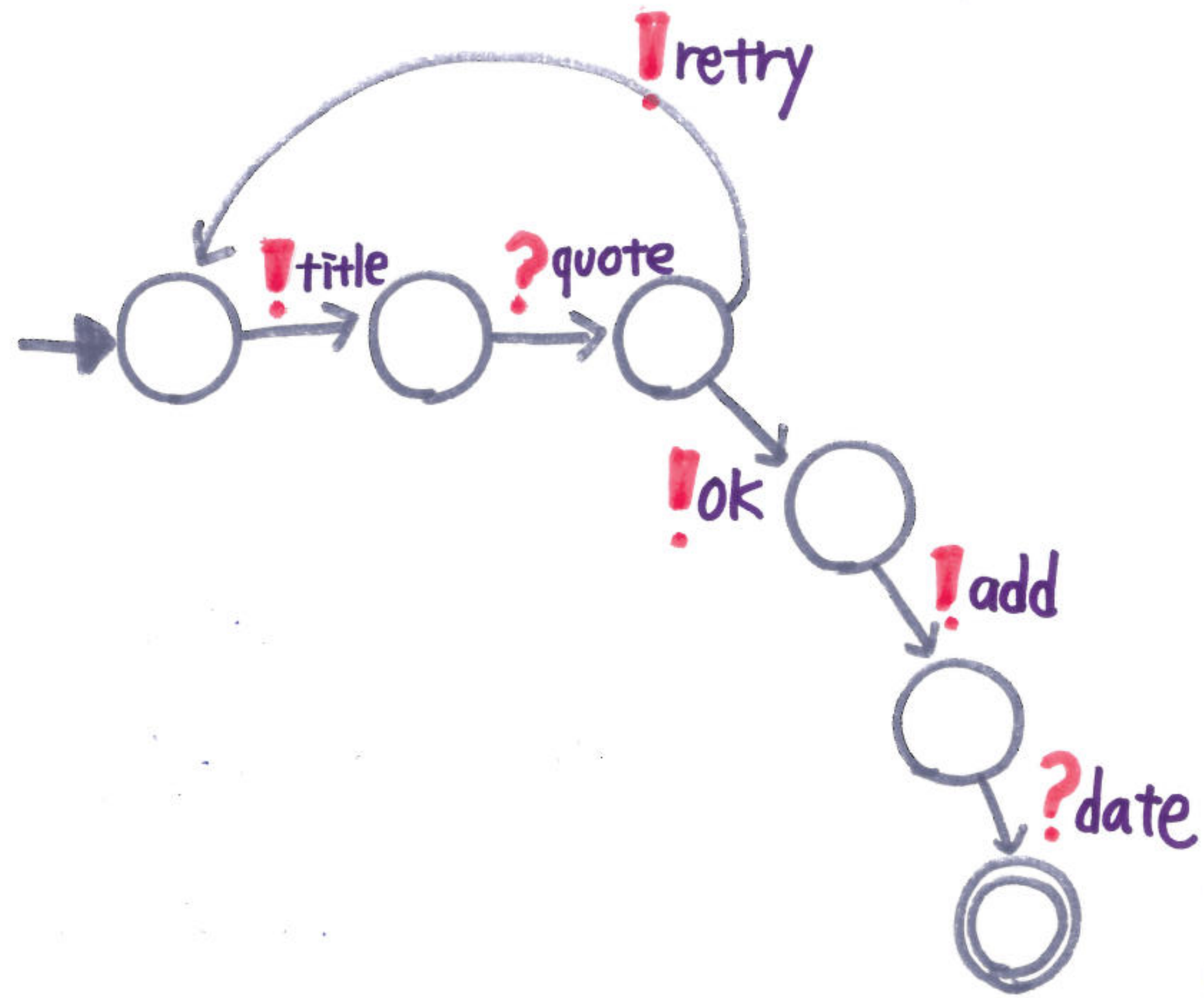


P has T
 Q has \overline{T} *dual*
 P | Q typable

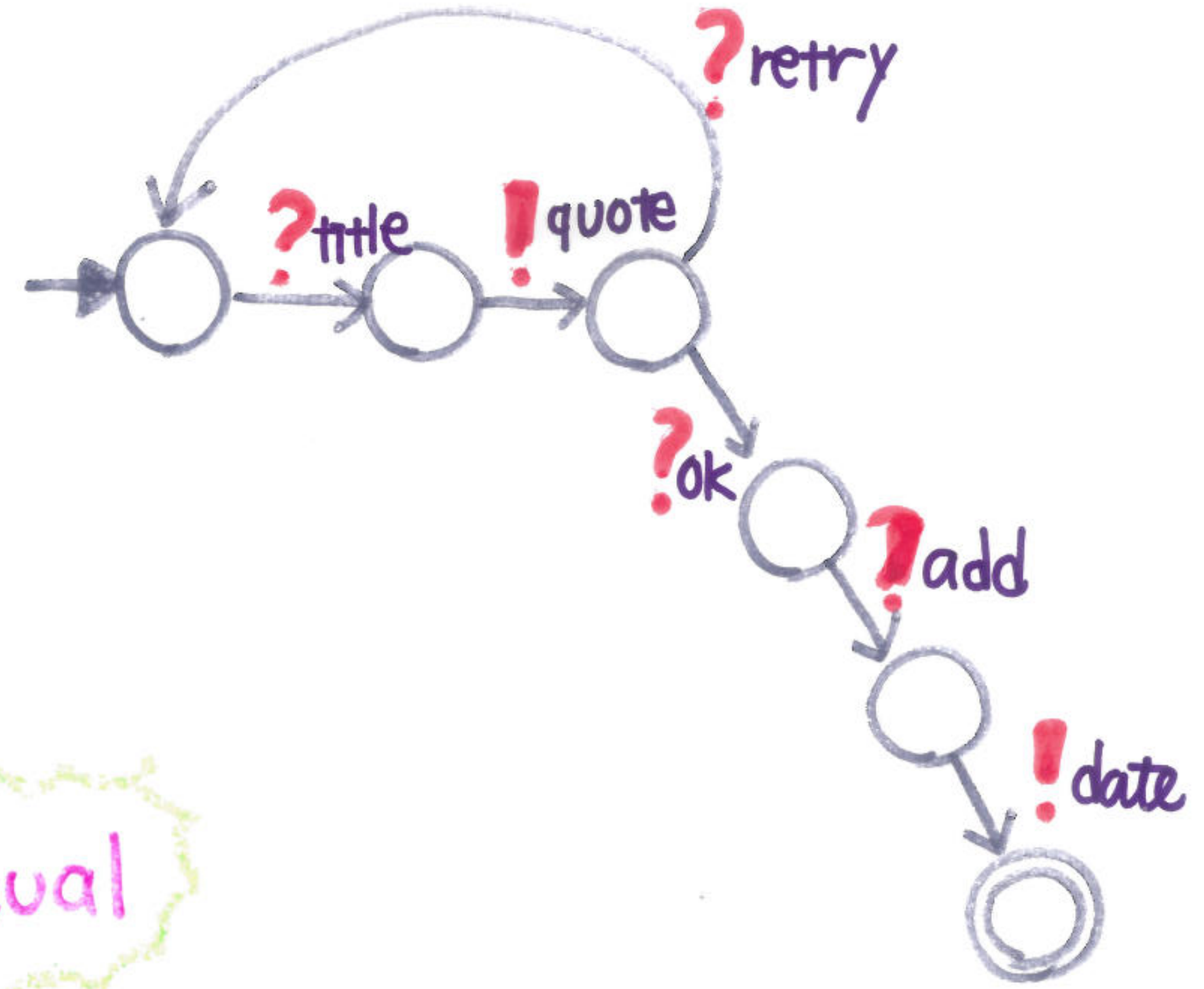
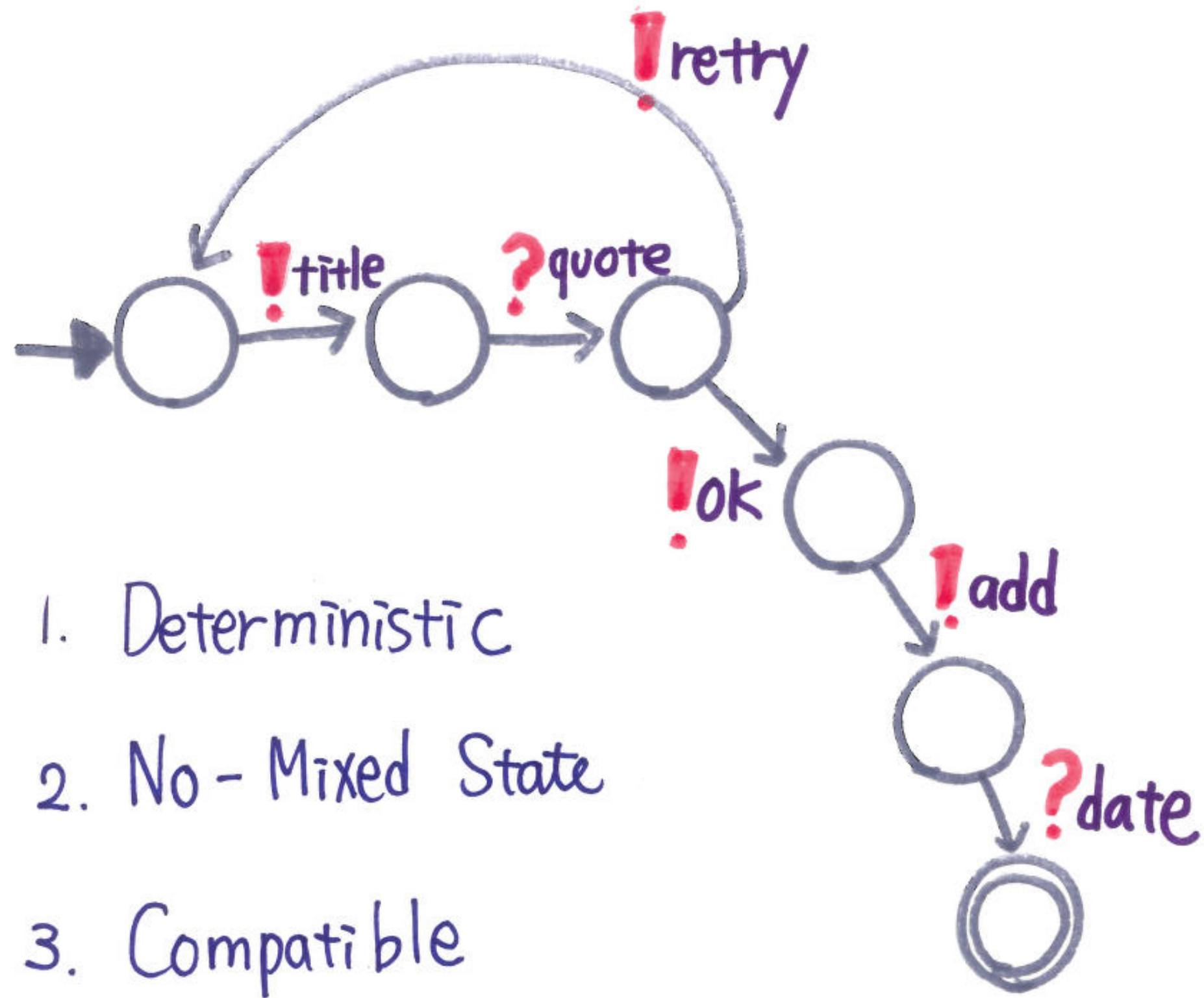
nt! Title ; ? Quote ; ! { ok: ! Add ; ? Date , **retry** : t }

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

Communicating Automata [1980s]



dual



dual

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

Theorem 1 $\text{Tr}(G) \approx \text{Tr}(\{G \upharpoonright P\}_{P \in \text{Participant}})$

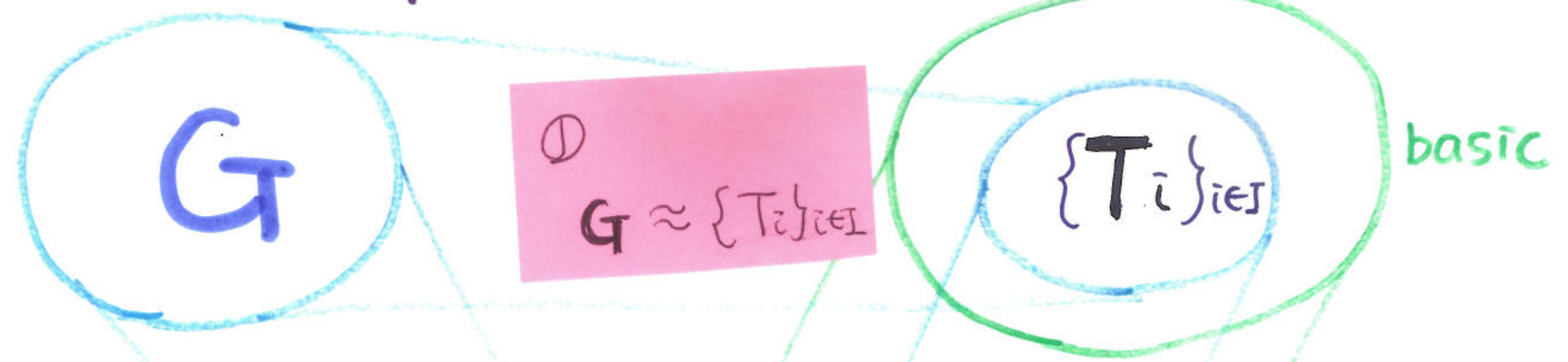
Soundness and Completeness between
a global type and **projected** local types

Theorem 2 $\text{Tr}(T) \approx \text{Tr}(A(T))$ T basic

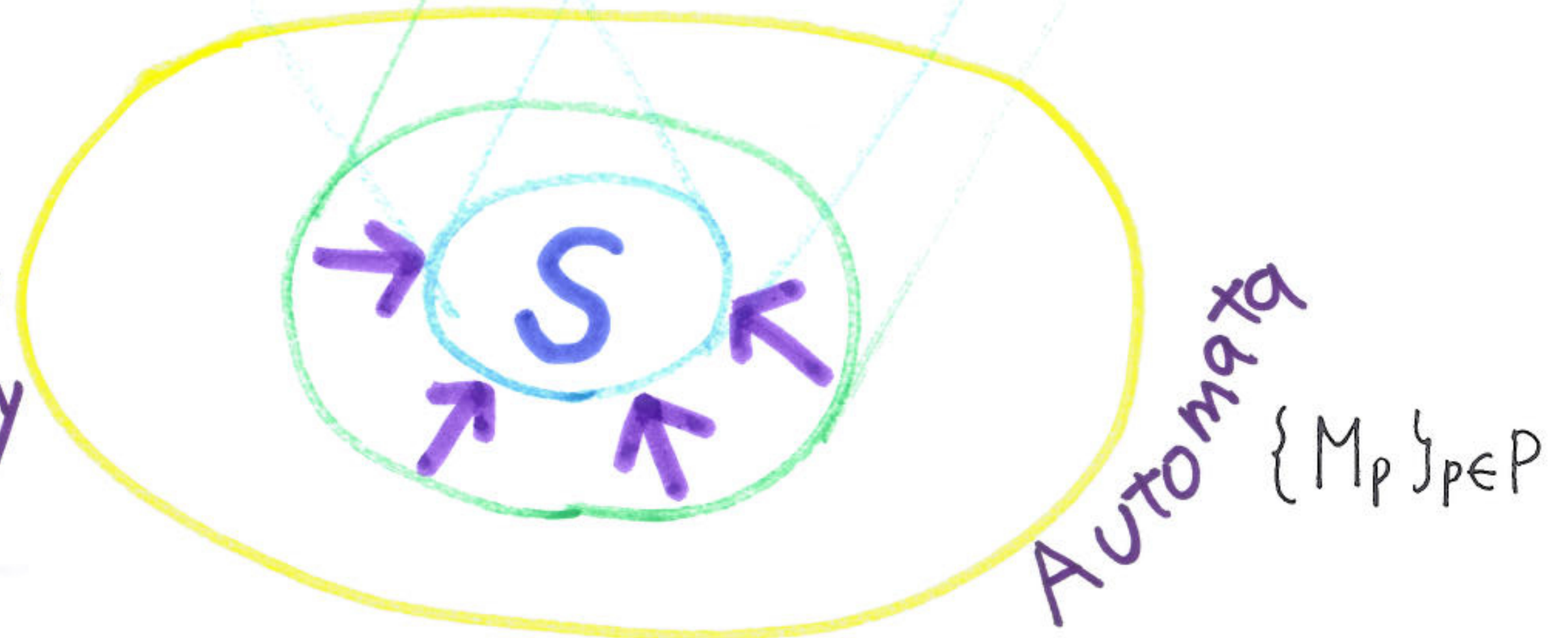
Soundness and Completeness between
a local type T and a basic CFMSM of T

Global Type

Local Types

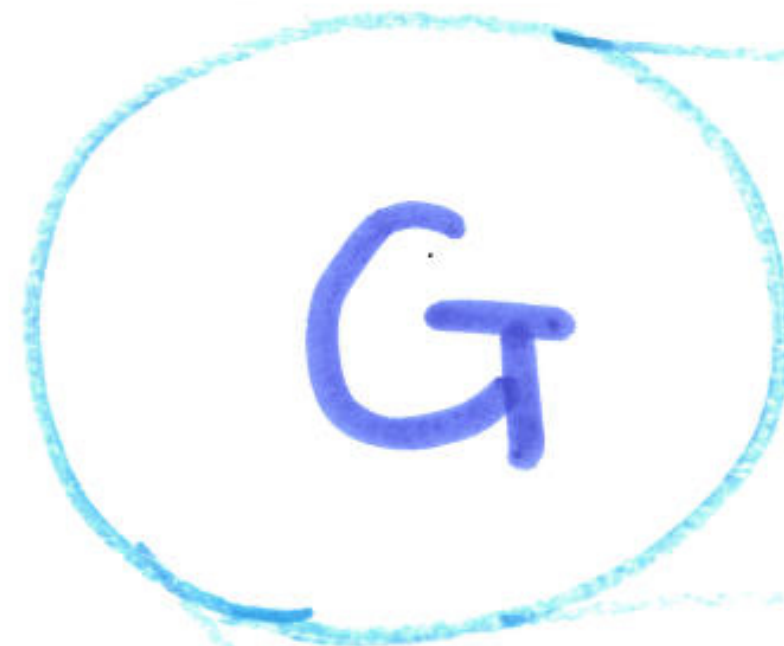


← multiparty compatibility

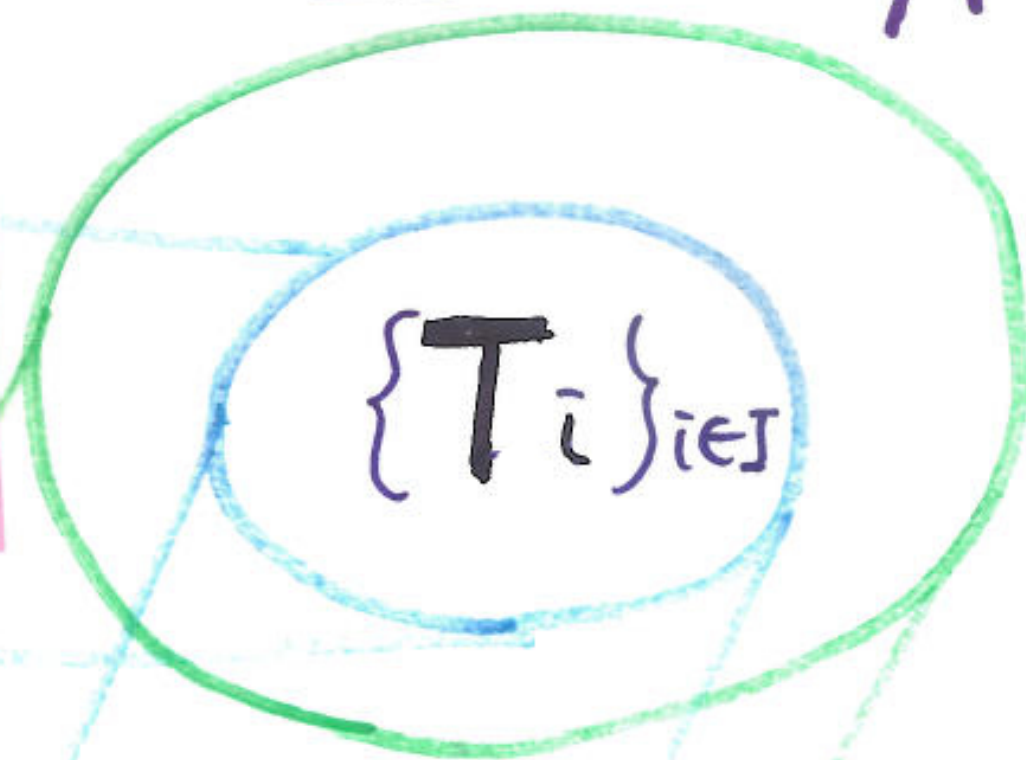


Global Type

Local Types



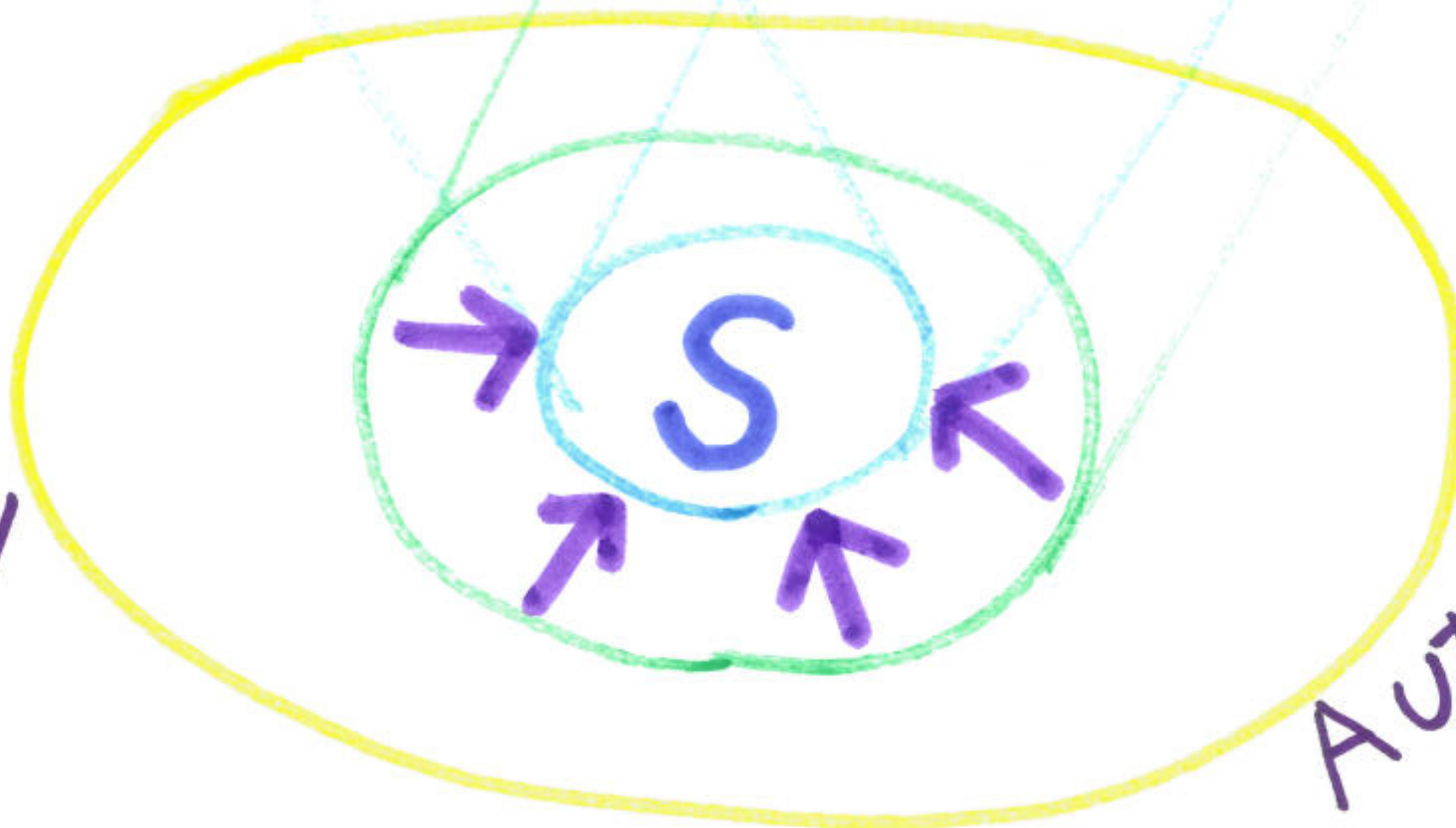
①
 $G \approx \{T_i\}_{i \in I}$



basic

②
 $T_i \approx M_i$

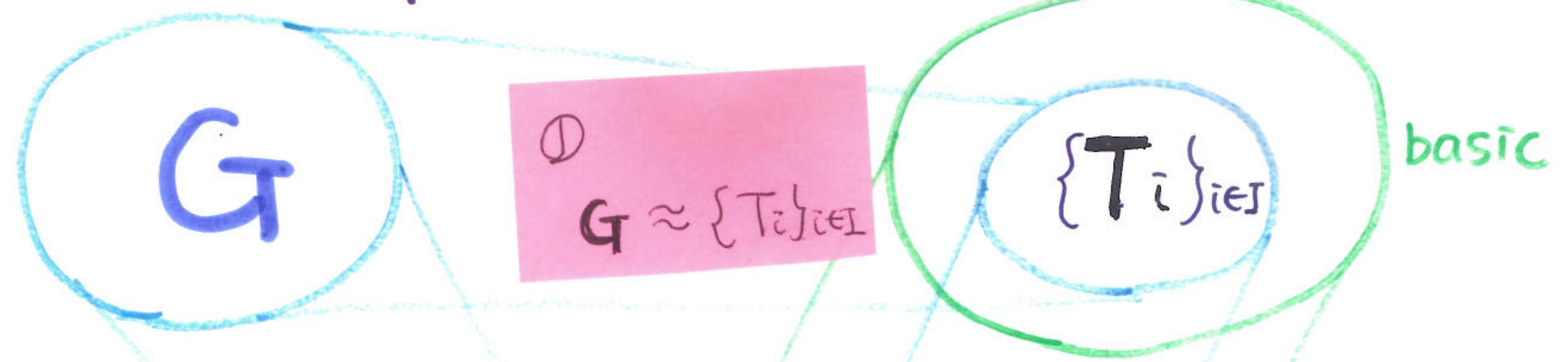
← multiparty compatibility



Automata $\{M_p\}_{p \in P}$

Global Type

Local Types



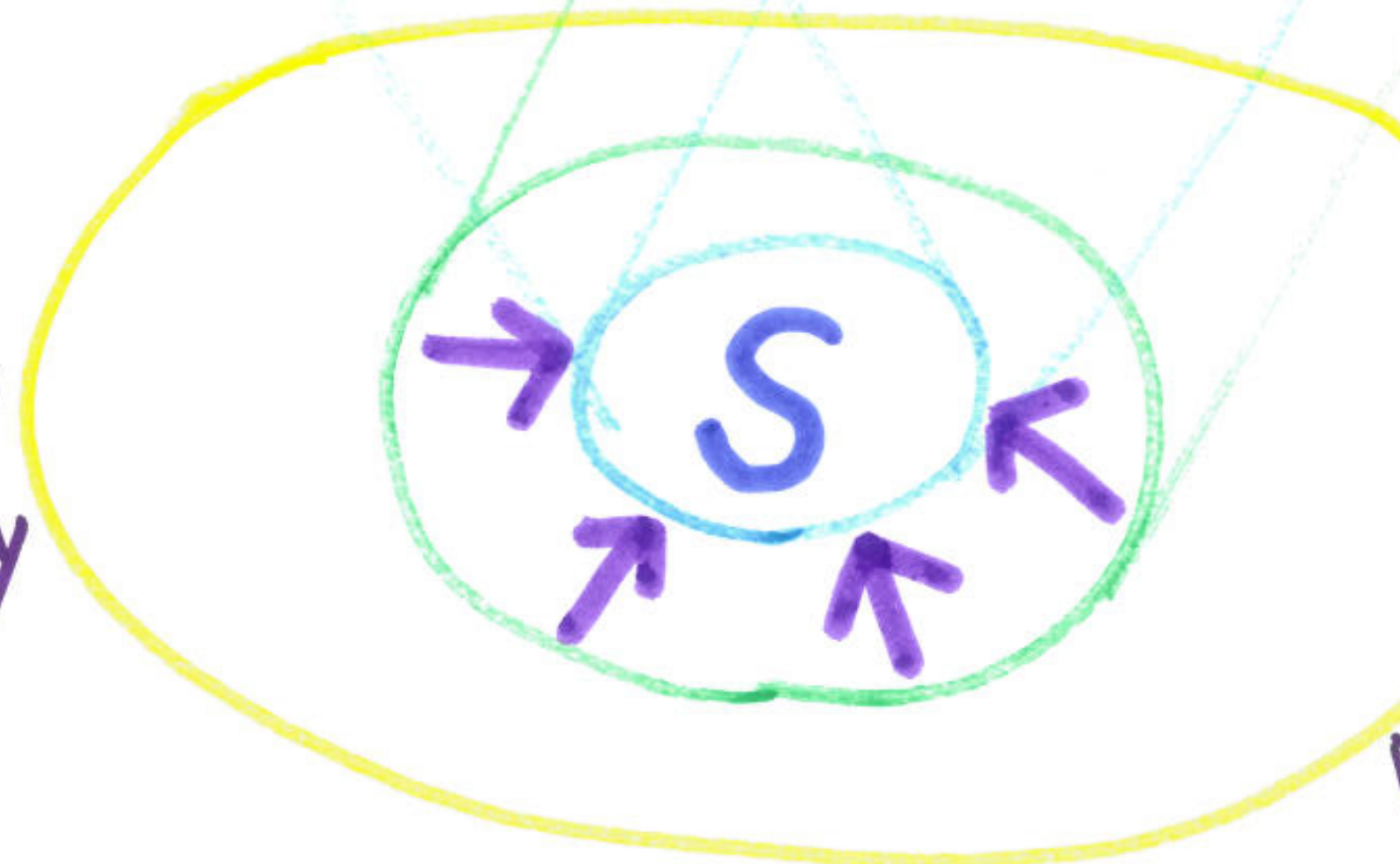
① $G \approx \{T_i\}_{i \in I}$

② $T_i \approx M_i$

③ $\{T_i\}_{i \in I}$ Multiparty Compatible

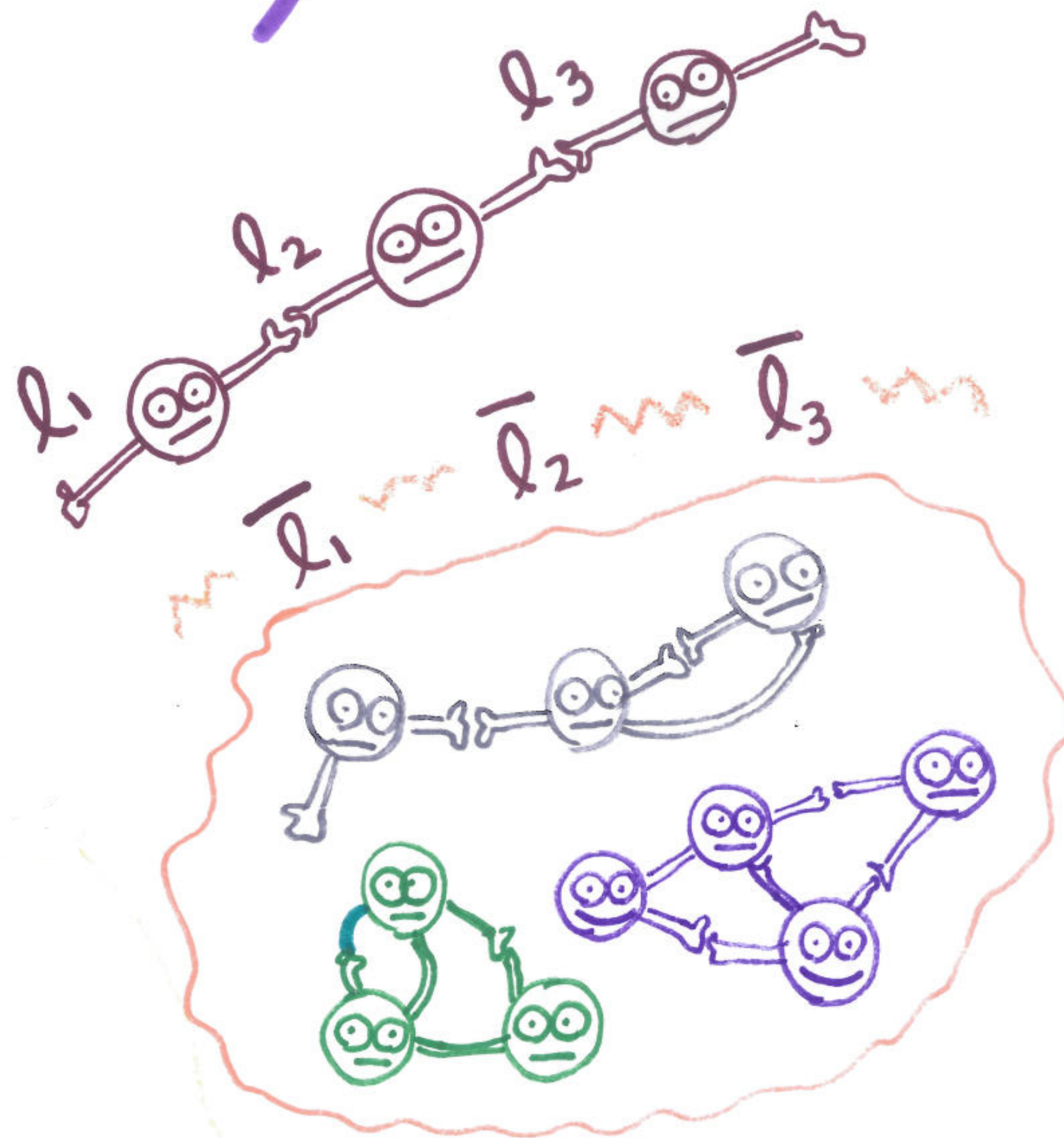
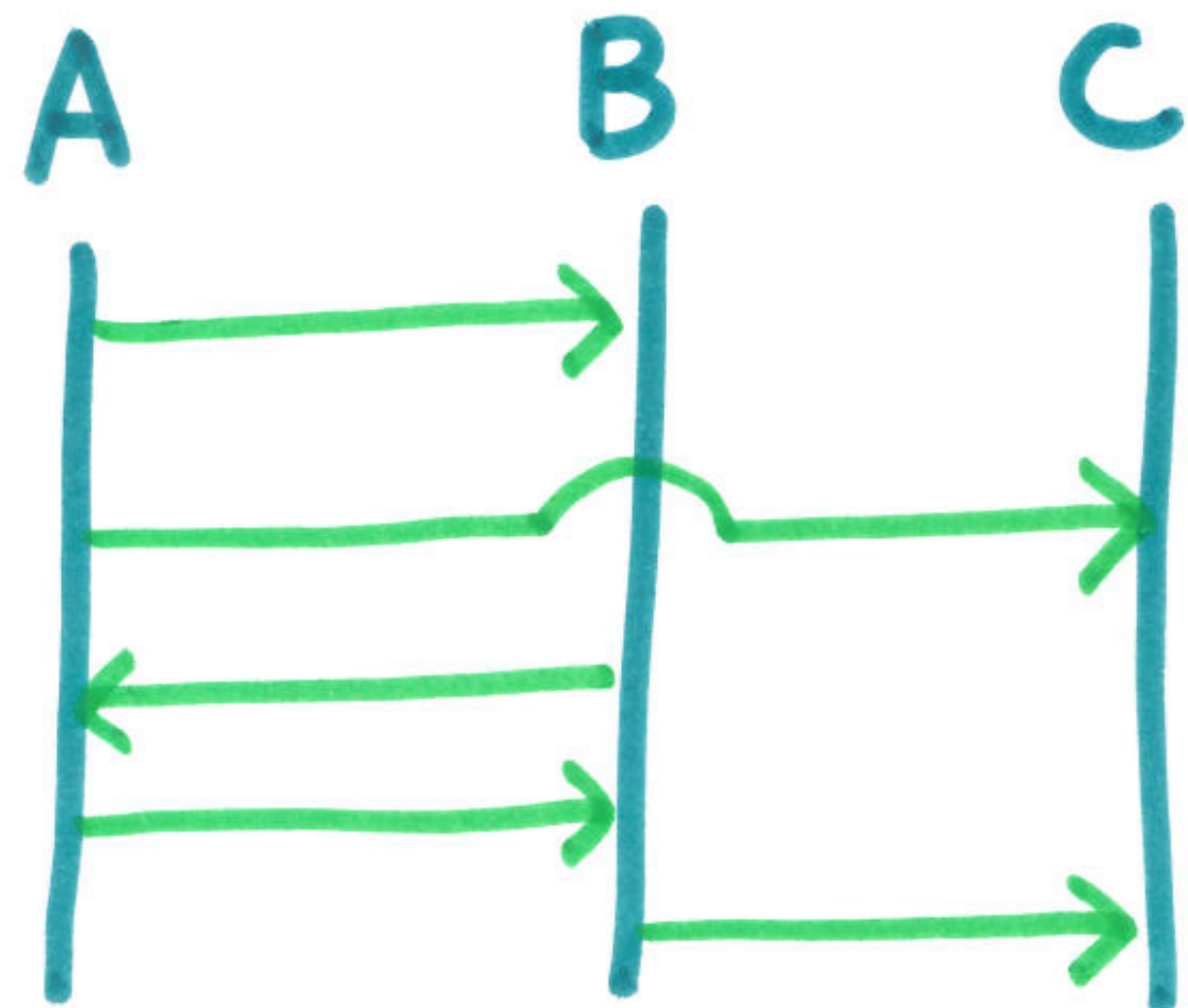
Multiparty Compatible
 $\{M_i\}_{i \in I}$ is safe

← multiparty compatibility

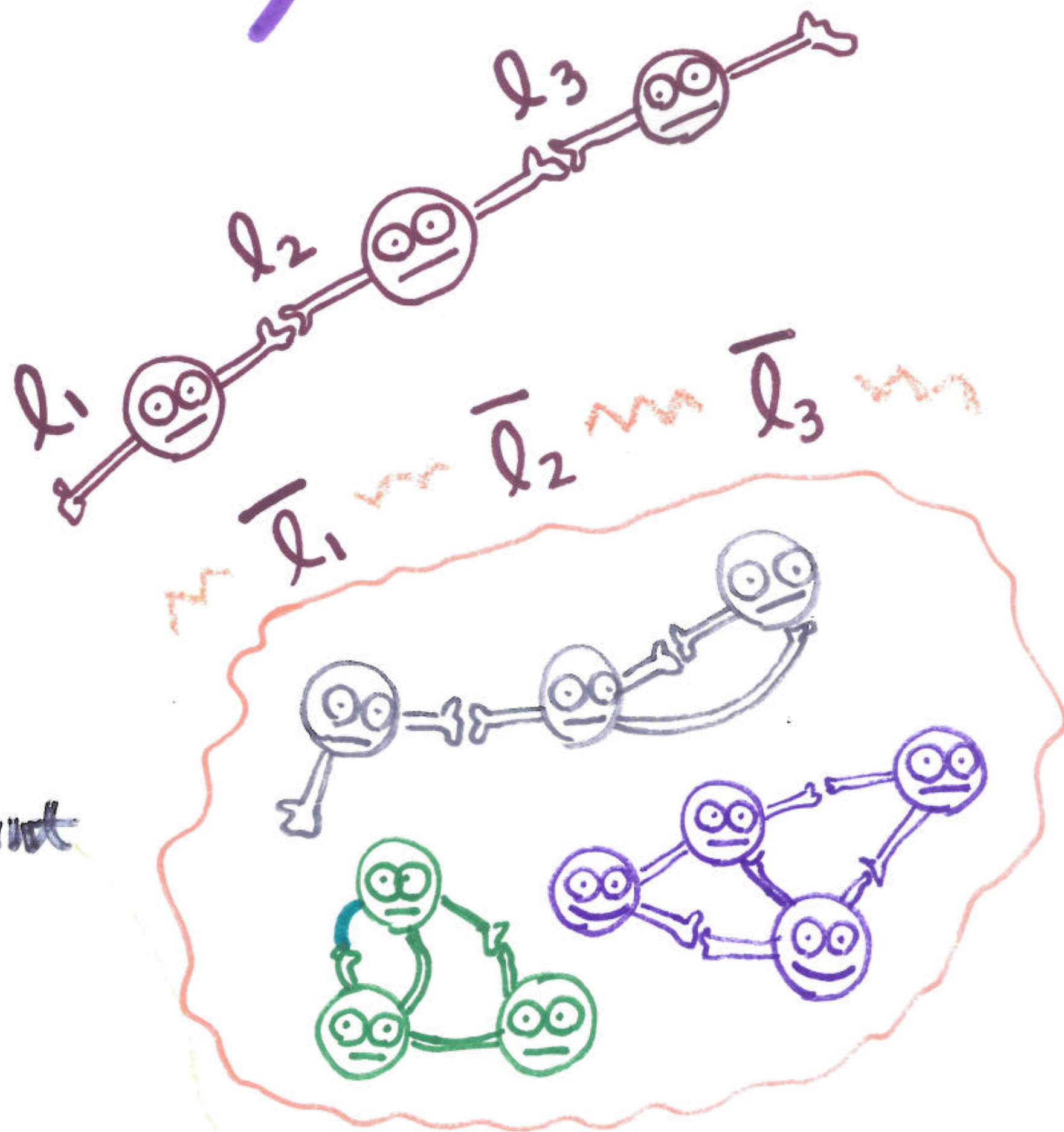
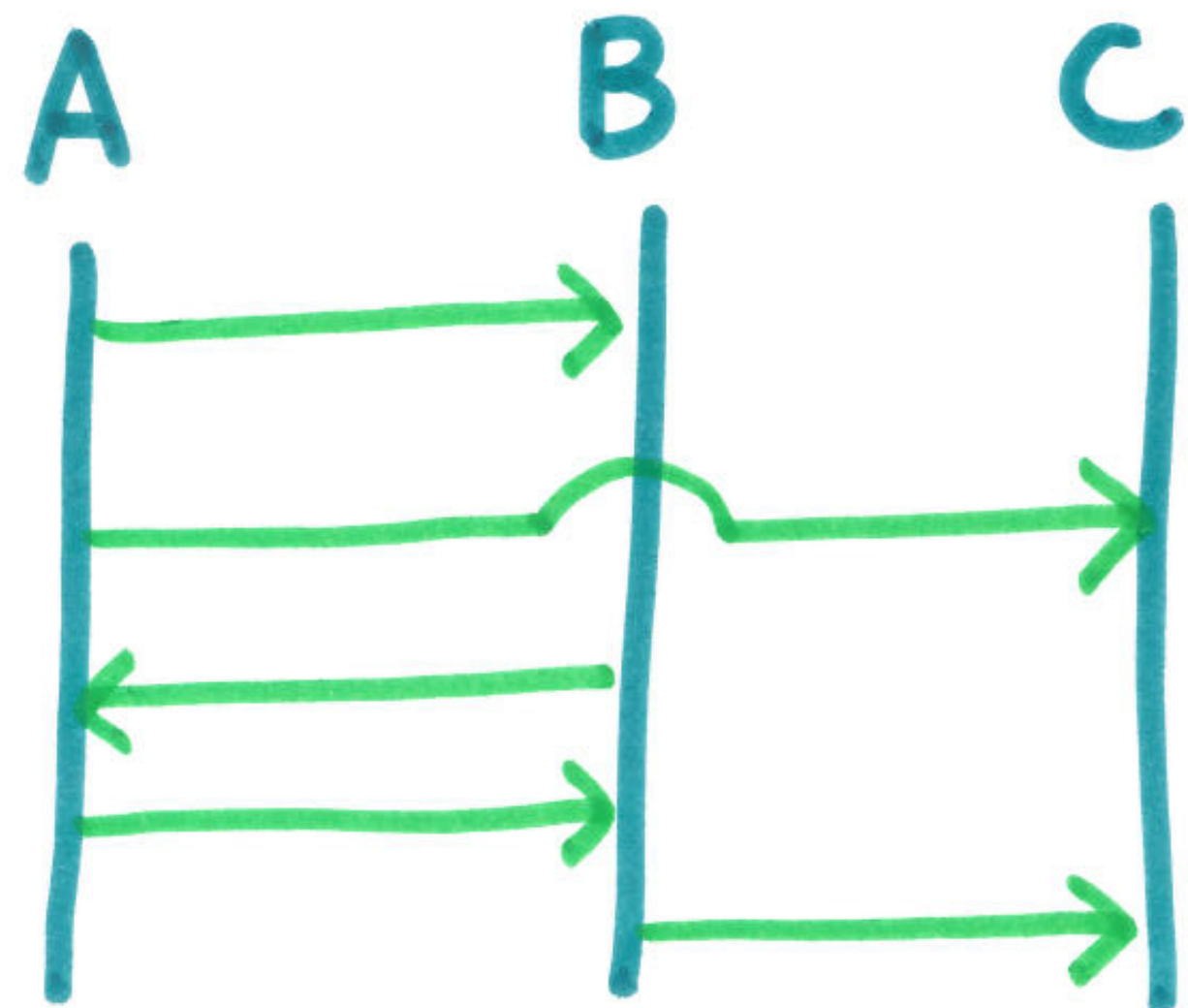


Automata $\{M_p\}_{p \in P}$

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)

Theorem (Safety) Suppose S is basic and multiparty compatible.

Then S is free from the following errors. Let $S = (\vec{q}; \vec{w})$

Deadlock if $\vec{w} = \vec{E}$ and not final, each q_p is receiving

Ophan Message all q are final, but $\vec{w} \neq \vec{E}$

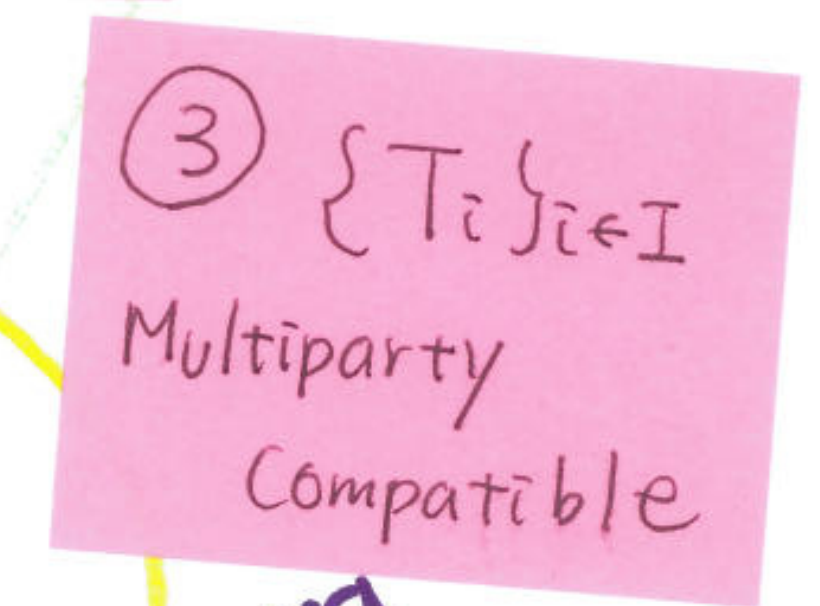
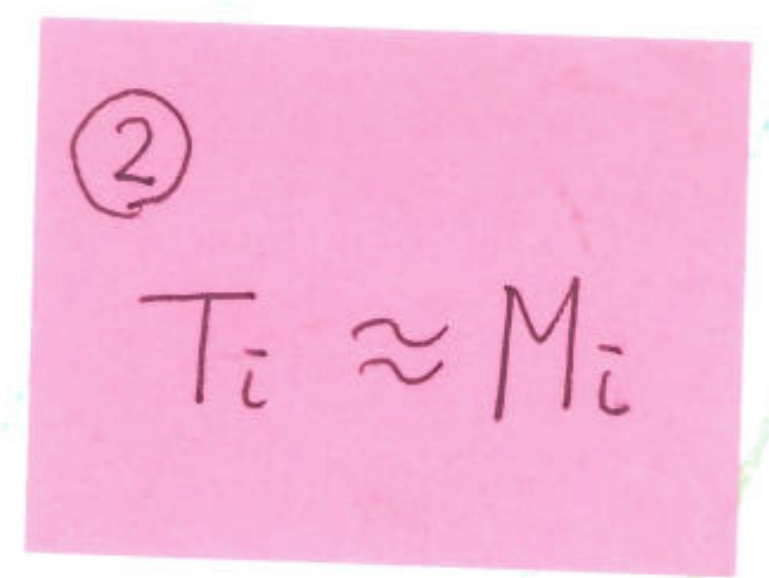
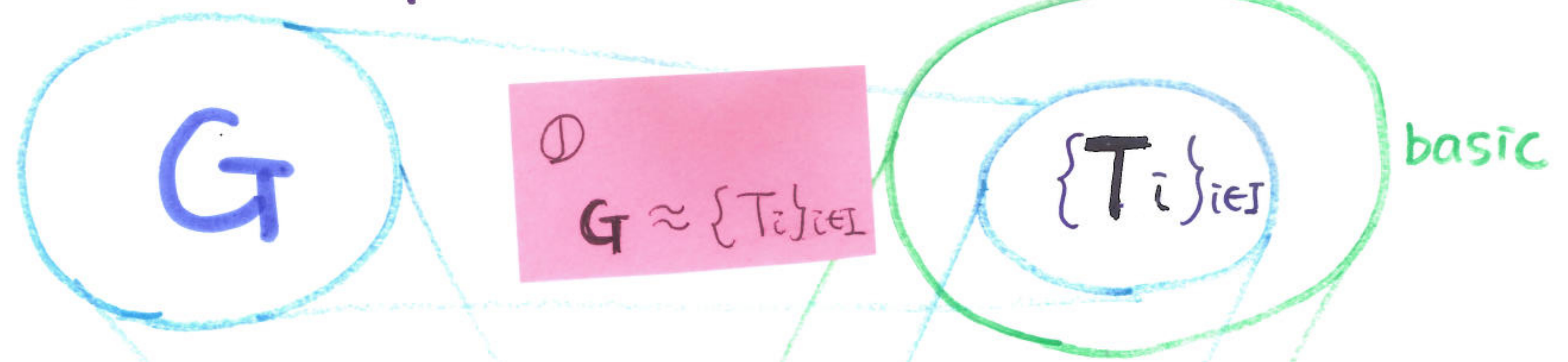
Unspecified Reception $(q_p, p \stackrel{?}{\leftarrow} a, q'_p)$ implies $|w_{pq}| > 1$ and $w_{pq} \notin \underline{a} \cdot \Sigma^*$

Proposition Soundness $A(\{G \upharpoonright_p\}_{p \in \text{Participant}})$ is multiparty compatible. **projection**

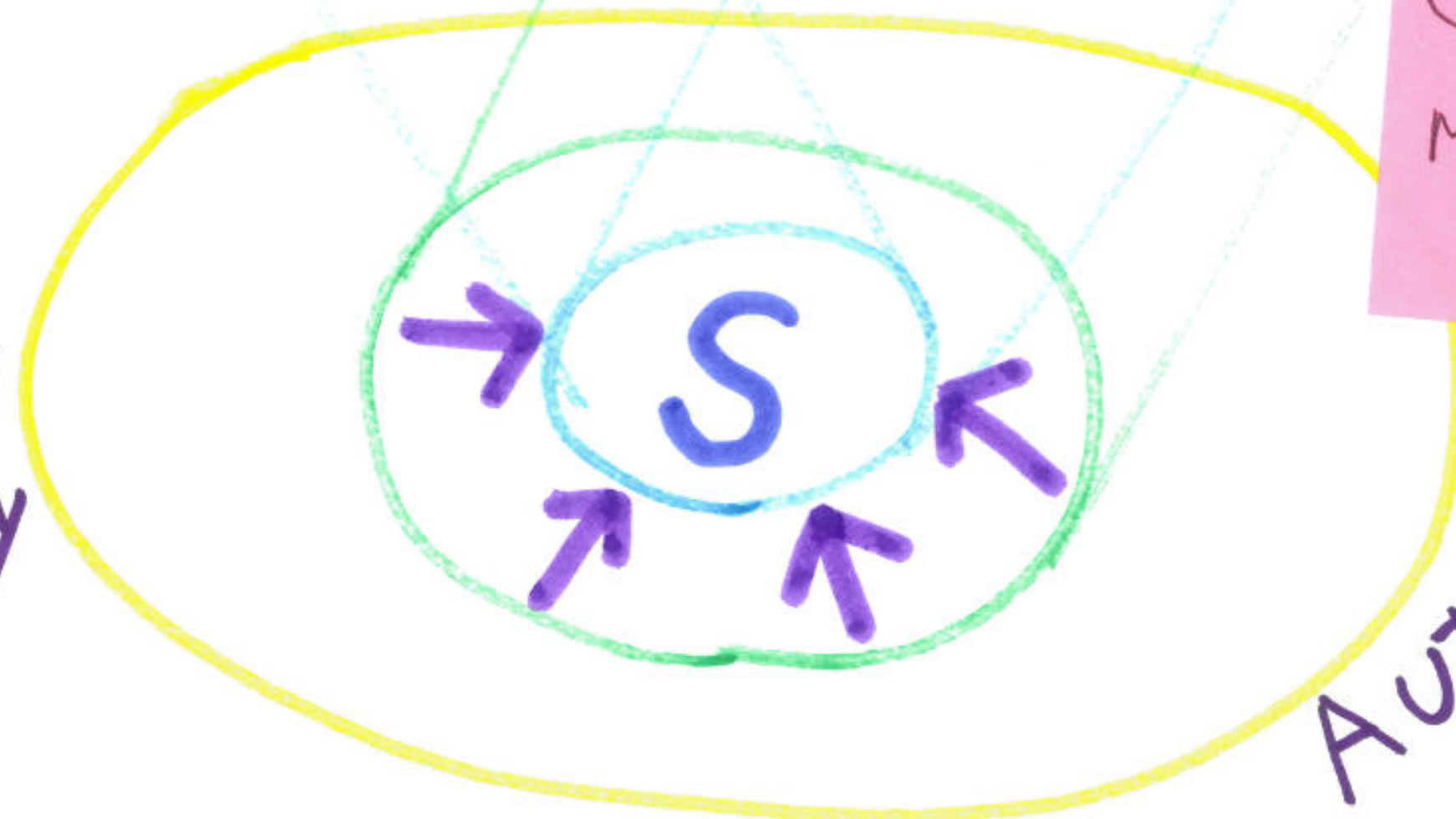
Proposition Multiparty Compatibility is decidable.

Global Type

Local Types



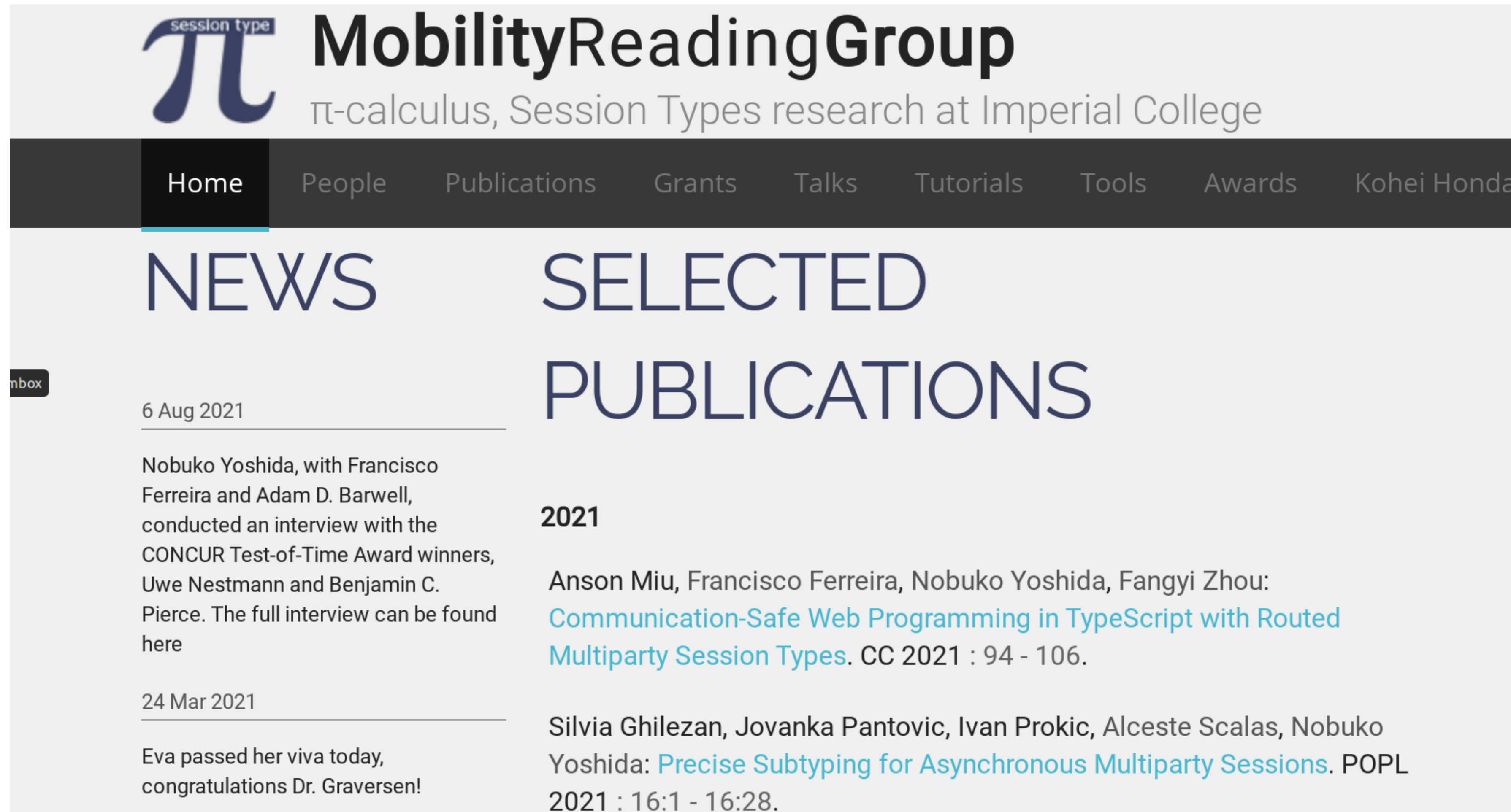
← multiparty compatibility



Automata {Mp}p in P

Mobility Reading Group

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



The screenshot shows the website for the Mobility Reading Group. At the top, there is a logo consisting of a blue pi symbol with the text "session type" 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 Imperial College" in a smaller, grey font. Below this is a dark grey navigation bar with the following items: "Home" (highlighted with a blue underline), "People", "Publications", "Grants", "Talks", "Tutorials", "Tools", "Awards", and "Kohei Honda".

The main content area is divided into two columns. The left column is titled "NEWS" in large, blue, uppercase letters. It contains two news items, each with a date and a short paragraph of text. The right column is titled "SELECTED PUBLICATIONS" in large, blue, uppercase letters. It contains a section for the year "2021" with two publication entries, each with the authors' names and a link to the publication title.

NEWS

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

Eva passed her viva today, congratulations Dr. Graversen!

SELECTED PUBLICATIONS

2021

Anson Miu, Francisco Ferreira, Nobuko Yoshida, Fangyi Zhou: [Communication-Safe Web Programming in TypeScript with Routed Multiparty Session Types](#). CC 2021 : 94 - 106.

Silvia Ghilezan, Jovanka Pantovic, Ivan Prokic, Alceste Scalas, Nobuko Yoshida: [Precise Subtyping for Asynchronous Multiparty Sessions](#). POPL 2021 : 16:1 - 16:28.

Optimising Asynchronous Communication in Rust

Deadlock-Free Message Reordering
with Multiparty Session Types

Zak Cutner, NY and Martin Vassor

Imperial College
London

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

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

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

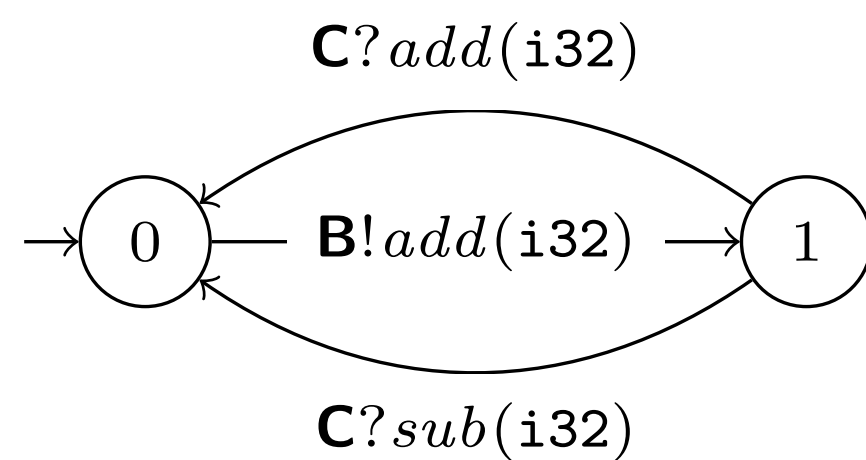
$$G = \mu t. \mathbf{A} \rightarrow \mathbf{B} : \left\{ \begin{array}{l} \mathit{add}(\mathit{i32}).\mathbf{B} \rightarrow \mathbf{C} : \left\{ \begin{array}{l} \mathit{add}(\mathit{i32}).\mathbf{C} \rightarrow \mathbf{A} : \{ \mathit{add}(\mathit{i32}).\mathbf{t} \} \\ \mathit{sub}(\mathit{i32}).\mathbf{C}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

Ring Protocol

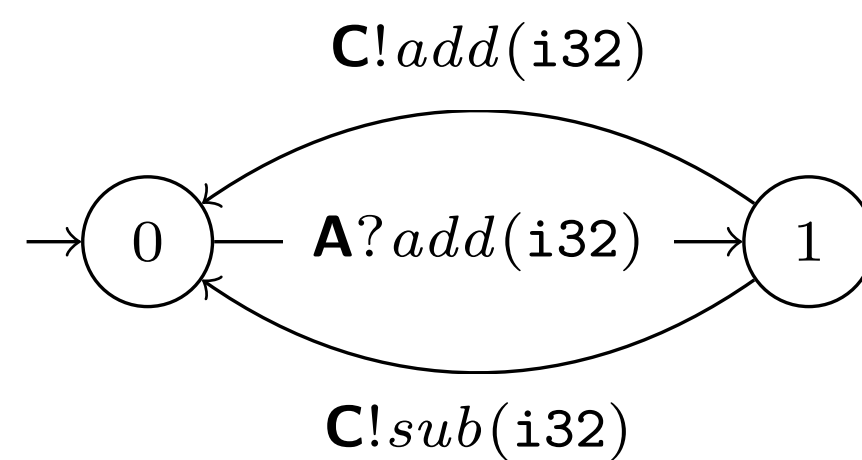
Example

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

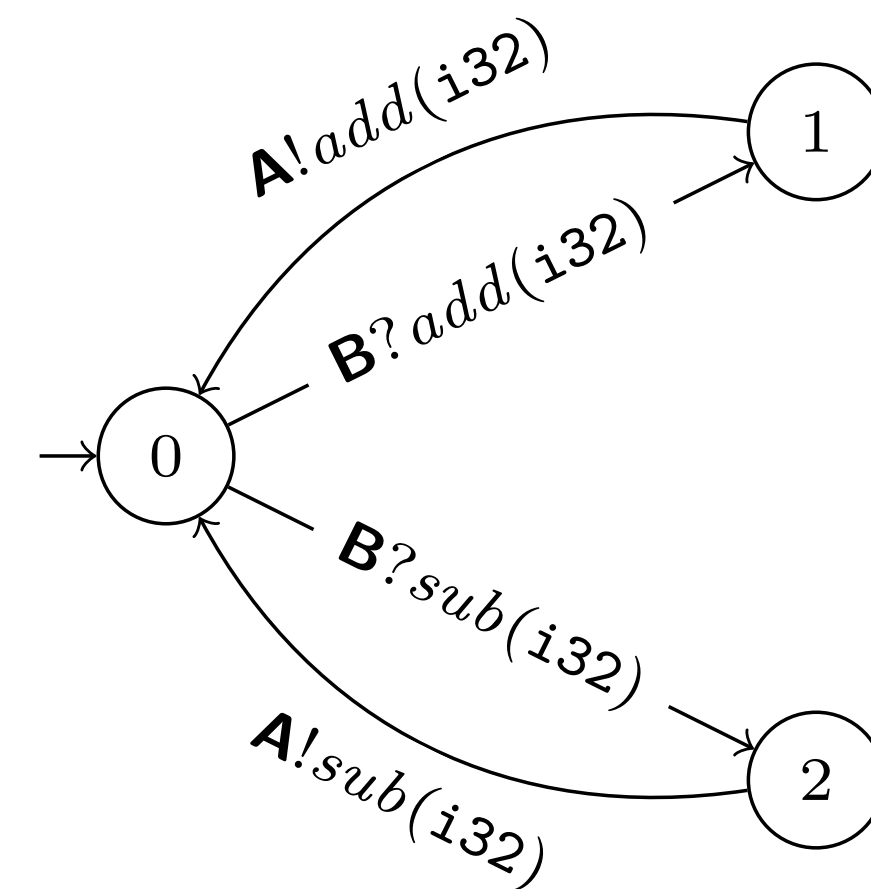
PROJECTION



PROJECTION



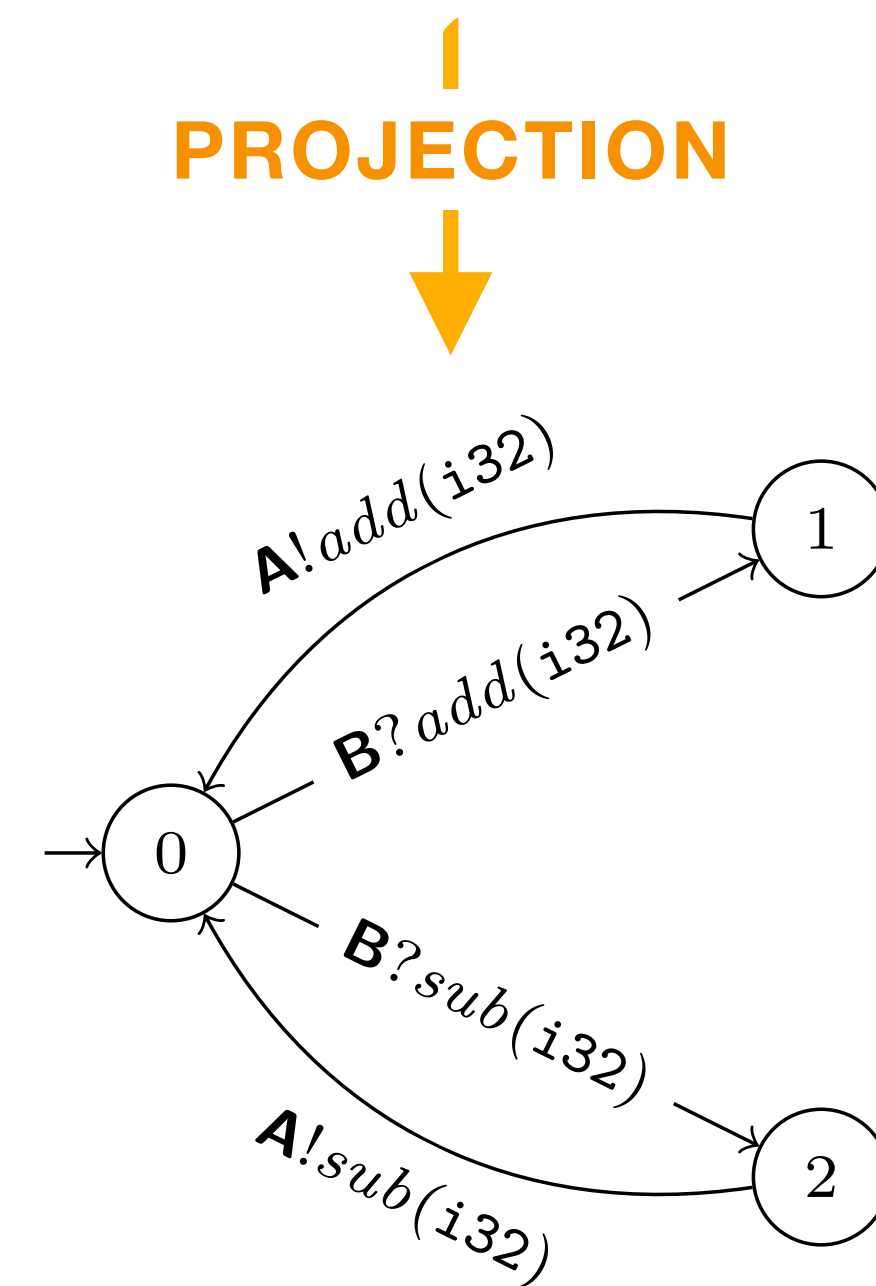
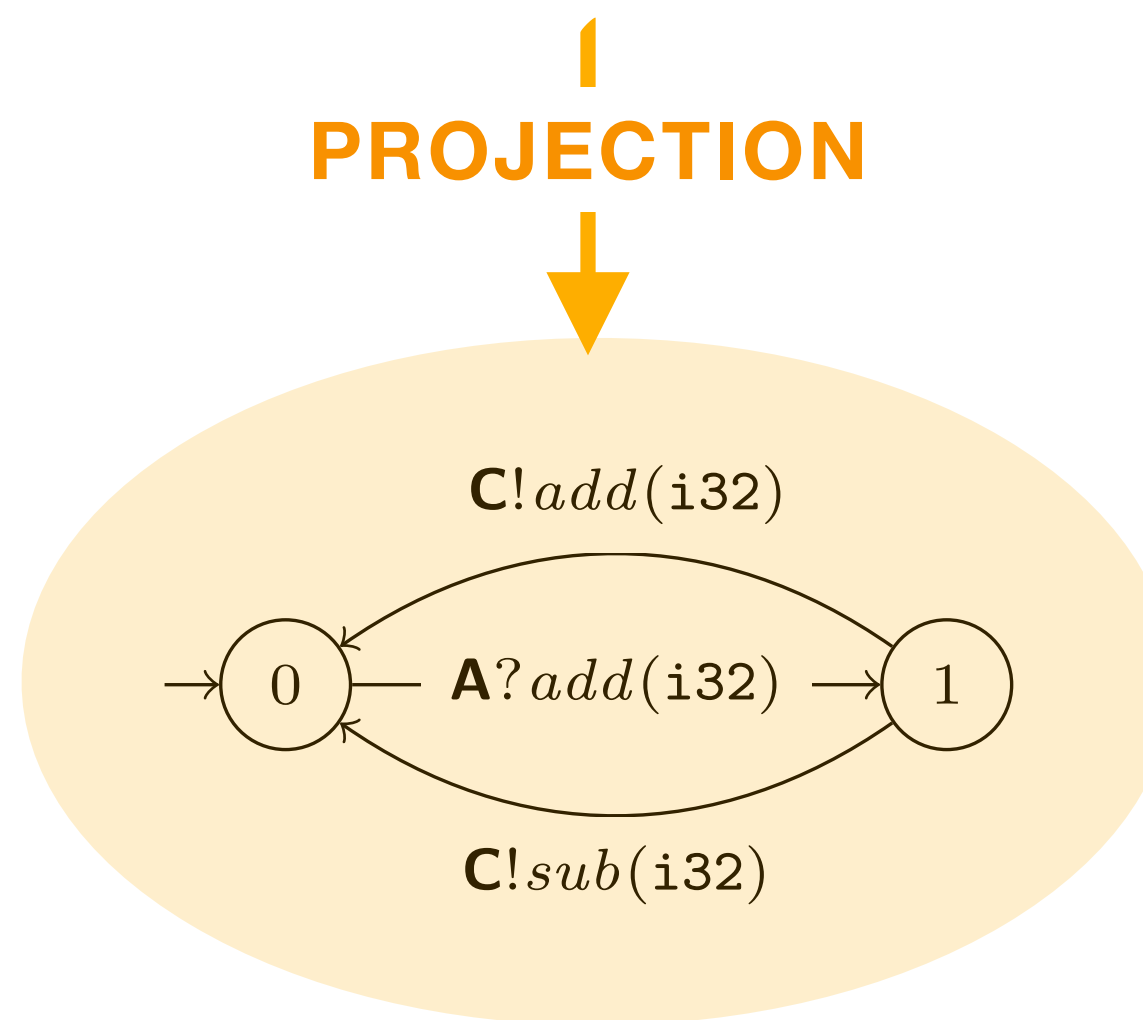
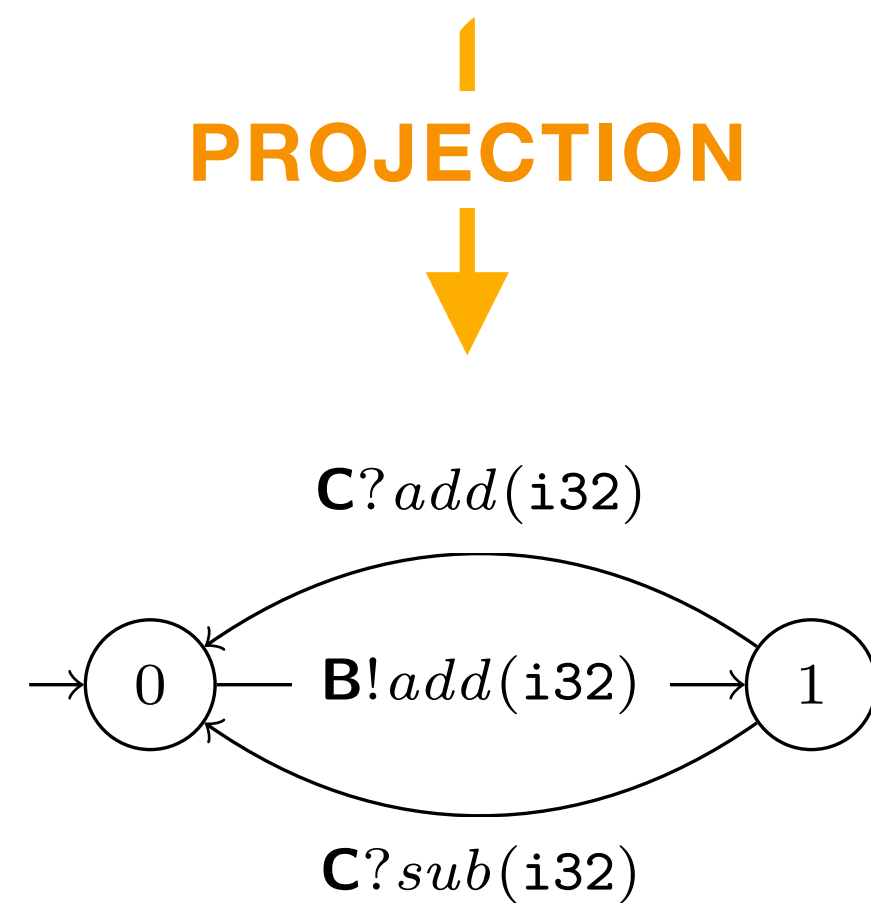
PROJECTION



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

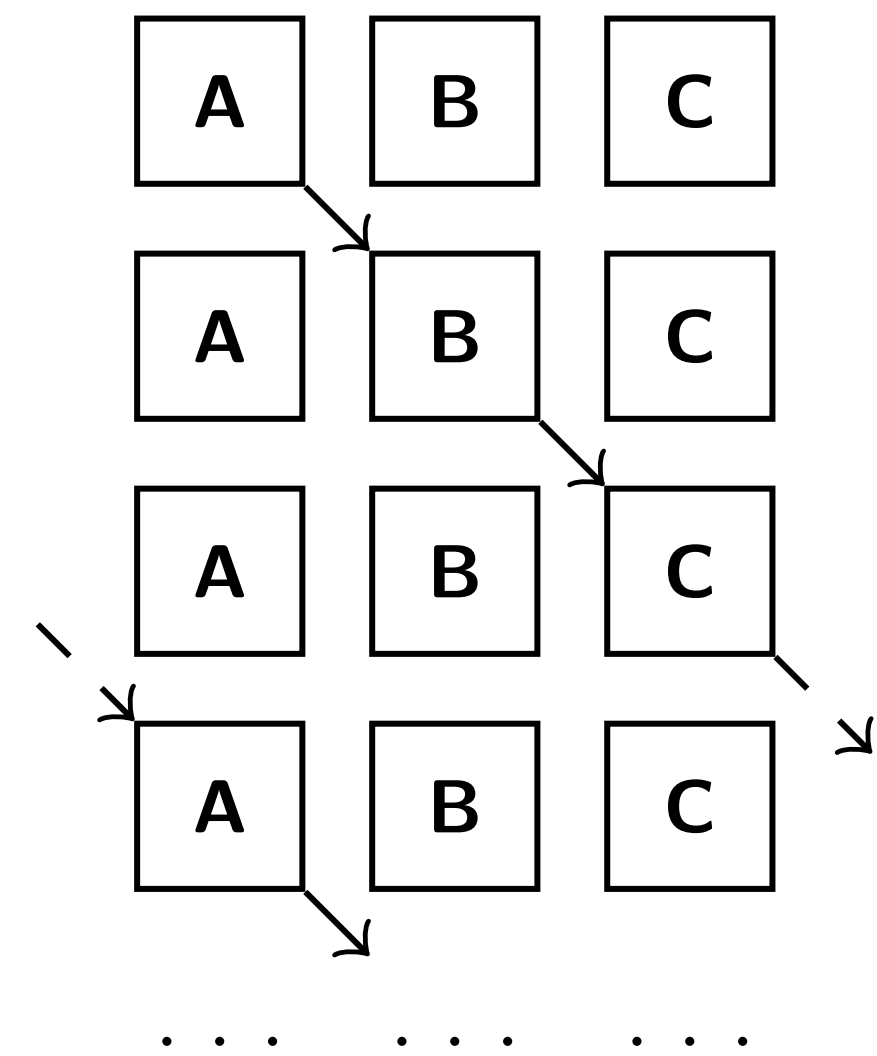
Asynchronous Orderings

- Global types are inherently **synchronous**

Challenge

Asynchronous Orderings

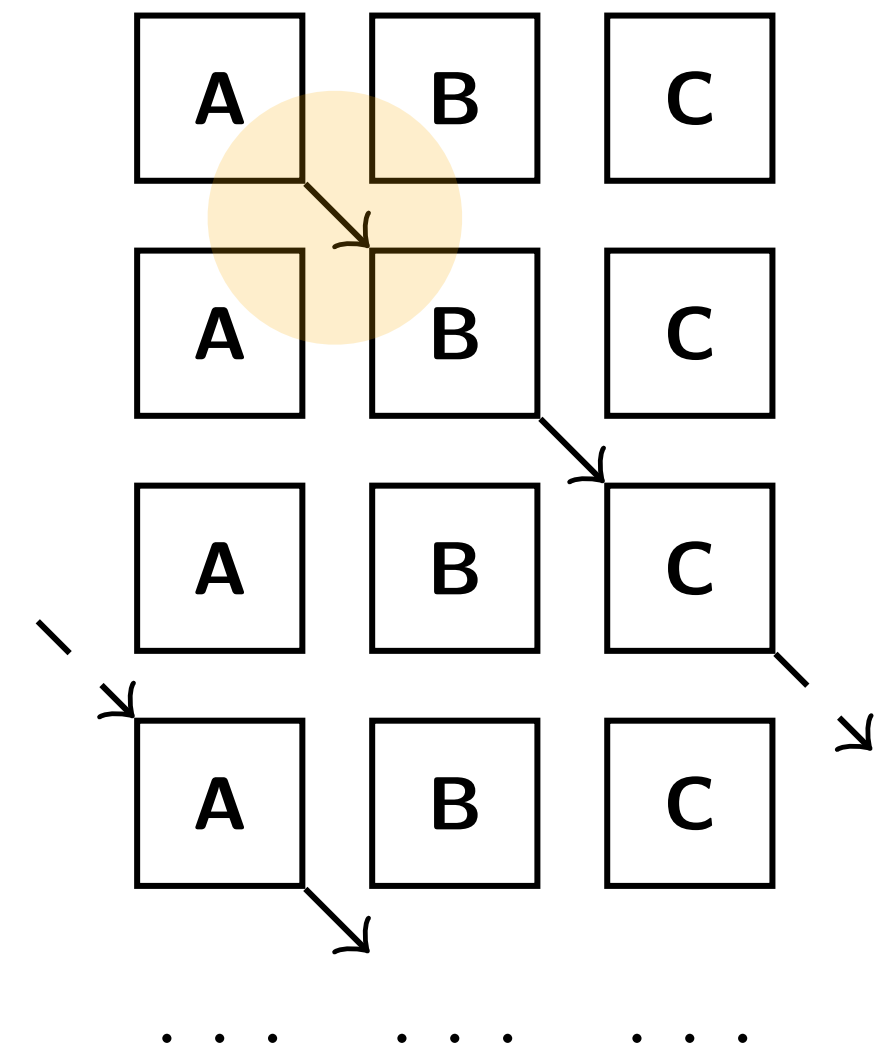
- Global types are inherently **synchronous**
 - Projection provides only one possible ordering



Challenge

Asynchronous Orderings

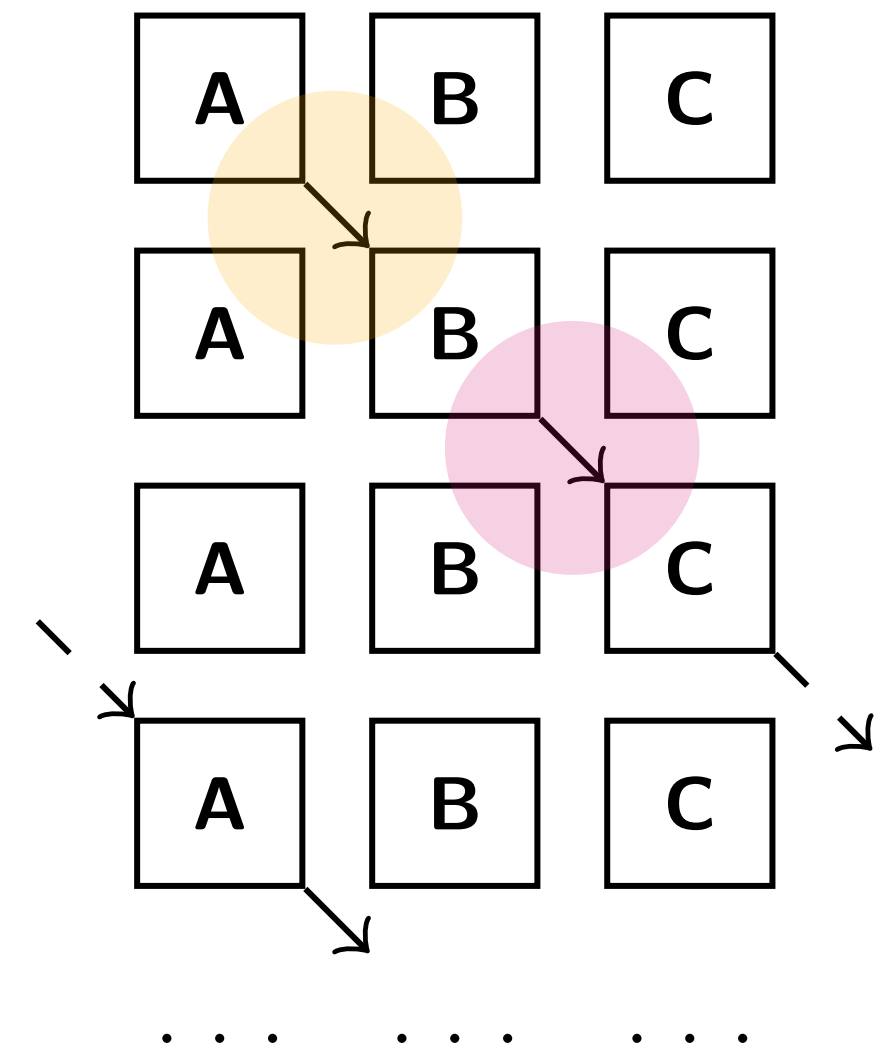
- Global types are inherently **synchronous**
 - Projection provides only one possible ordering



Challenge

Asynchronous Orderings

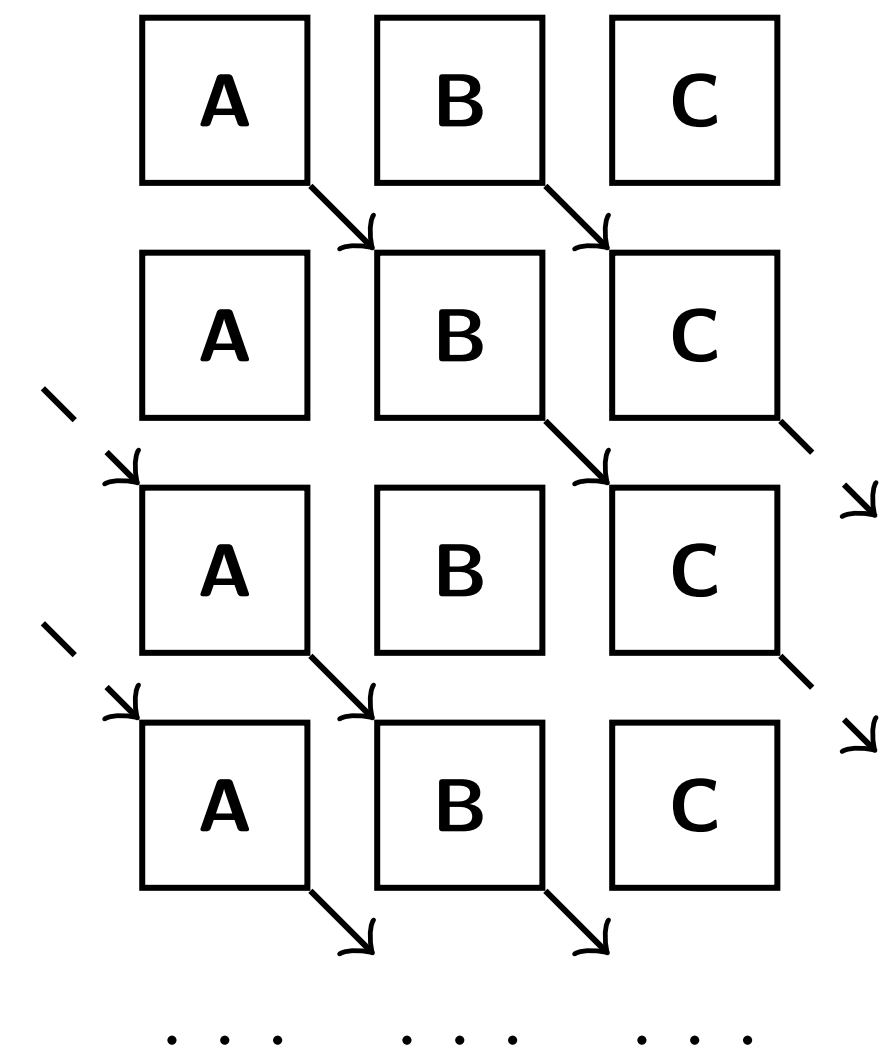
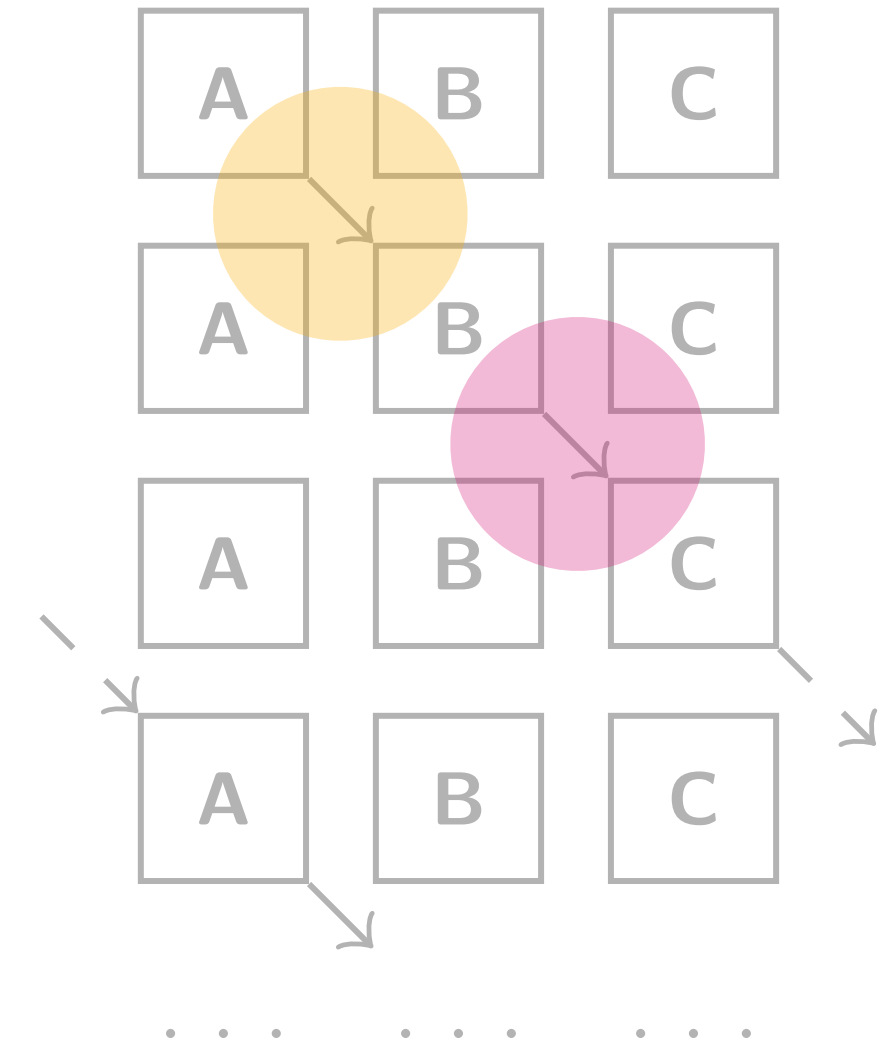
- Global types are inherently *synchronous*
 - Projection provides only one possible ordering



Challenge

Asynchronous Orderings

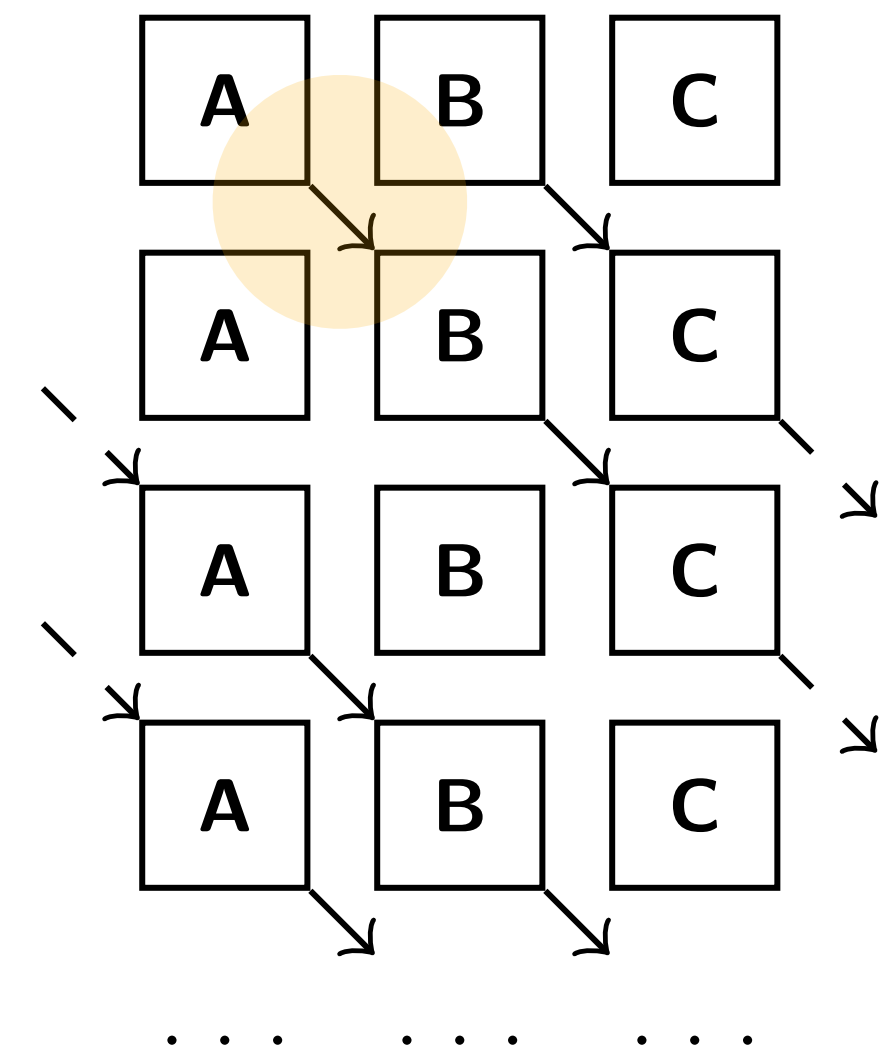
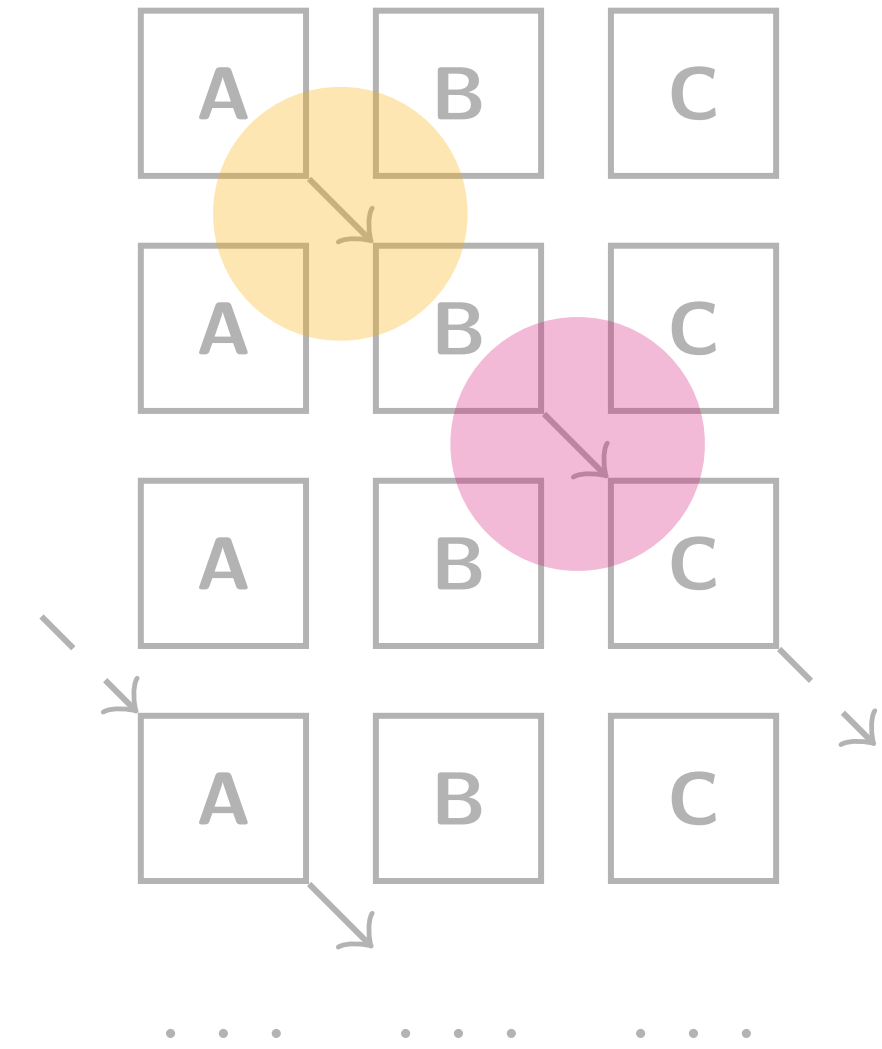
- Global types are inherently **synchronous**
 - Projection provides only one possible ordering
- Interactions can be **reordered** for efficiency while preserving safety



Challenge

Asynchronous Orderings

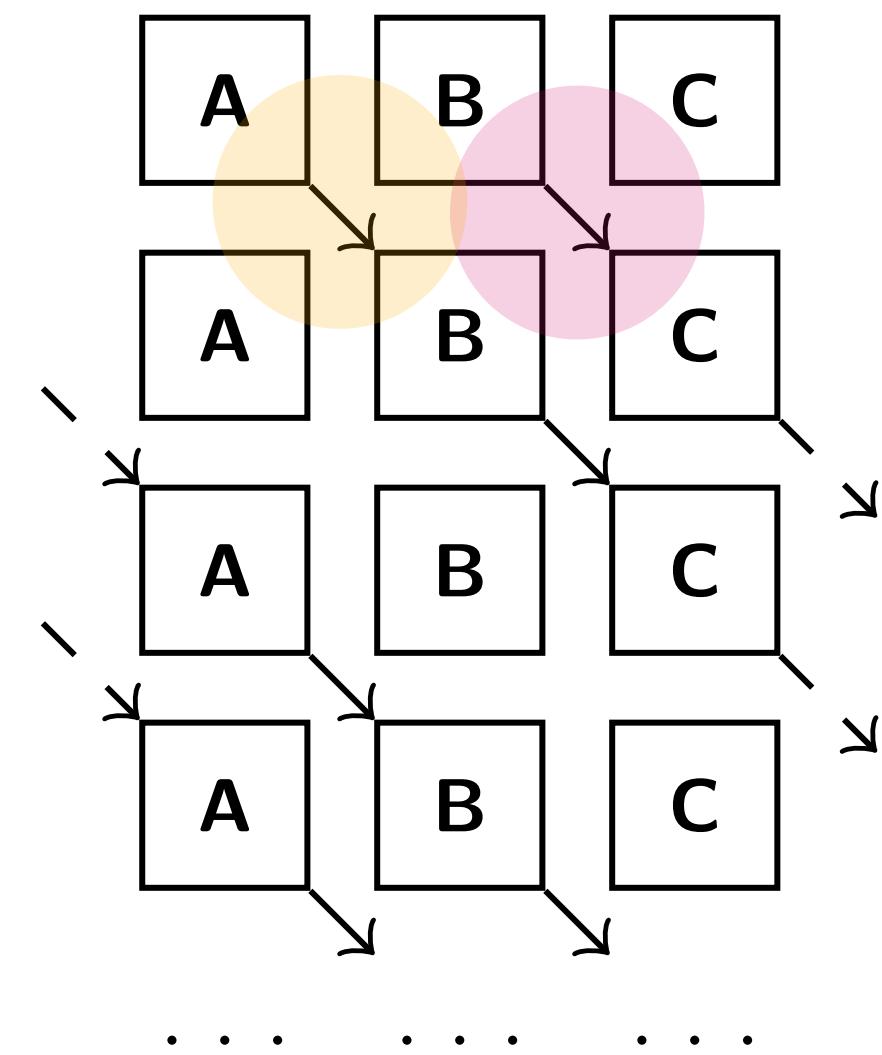
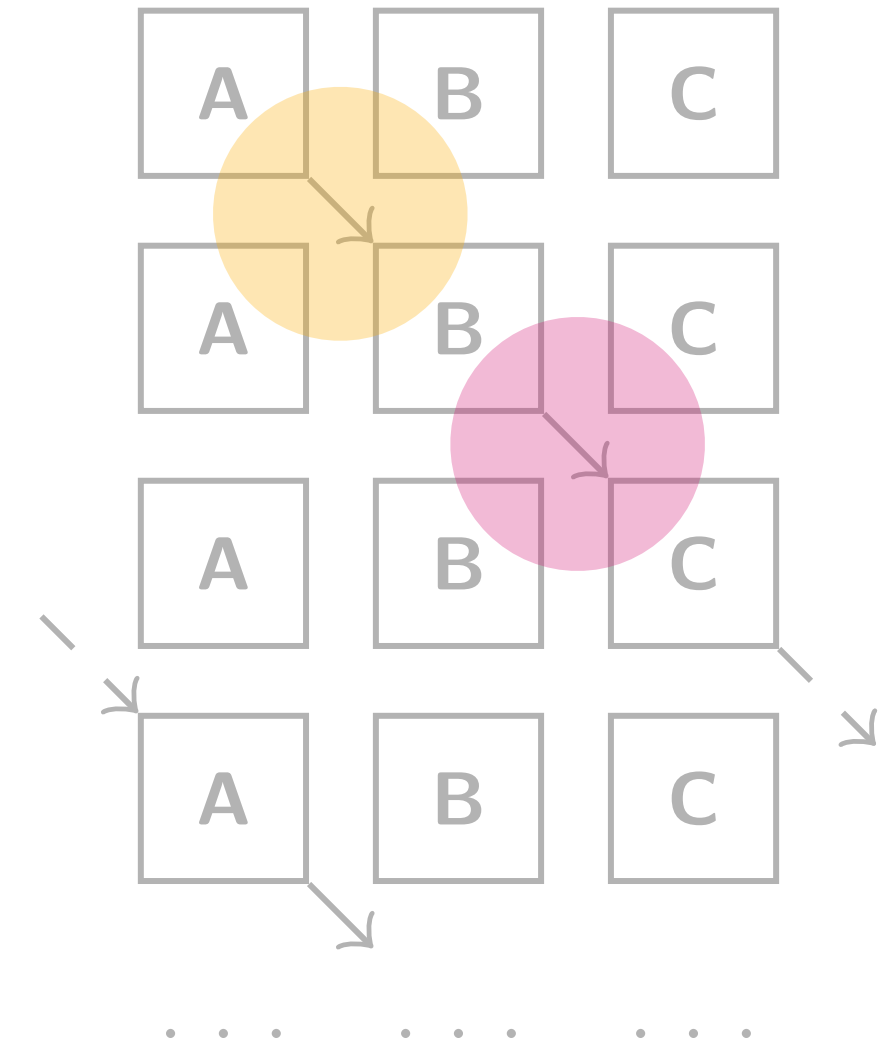
- Global types are inherently **synchronous**
 - Projection provides only one possible ordering
- Interactions can be **reordered** for efficiency while preserving safety



Challenge

Asynchronous Orderings

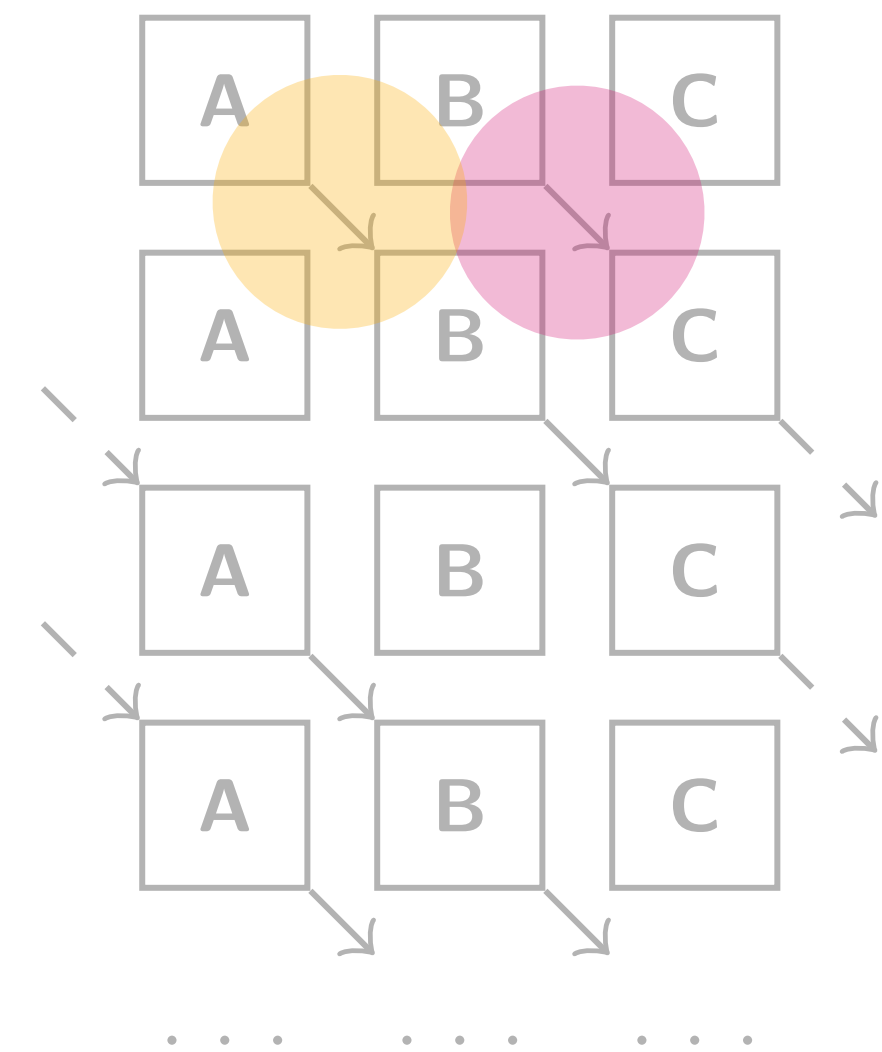
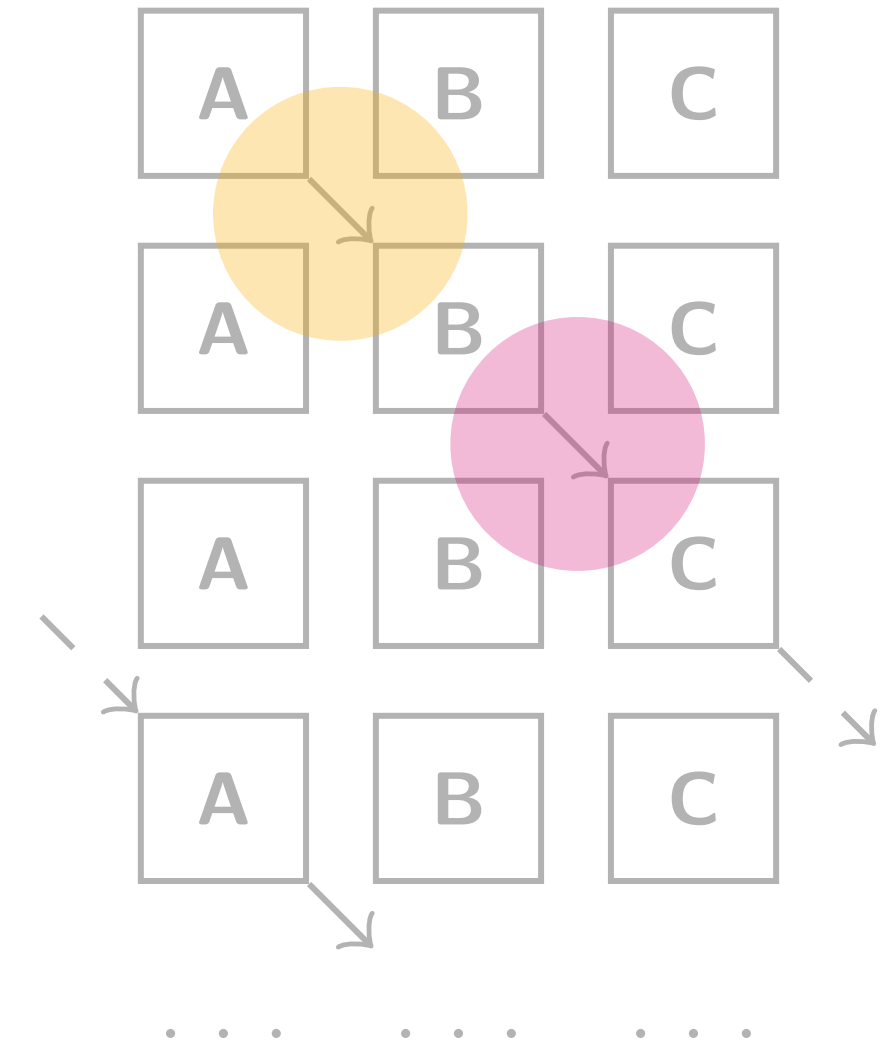
- Global types are inherently **synchronous**
 - Projection provides only one possible ordering
- Interactions can be **reordered** for efficiency while preserving safety



Challenge

Asynchronous Orderings

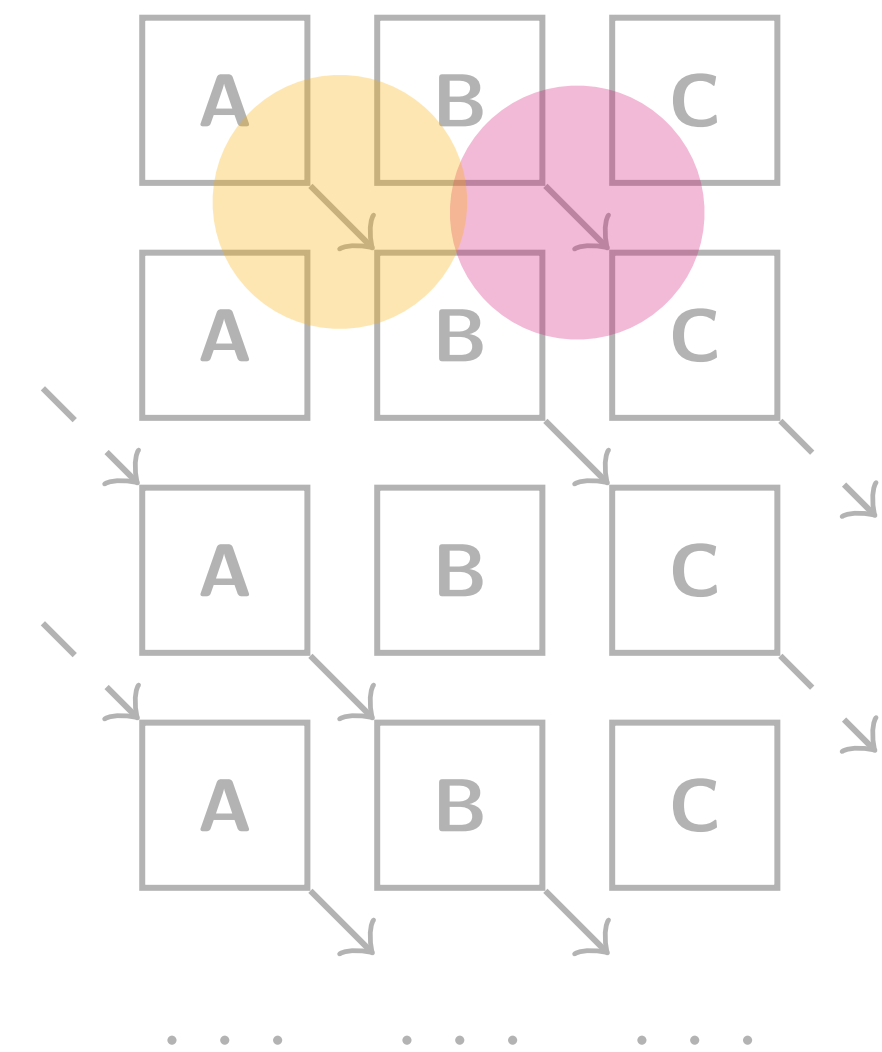
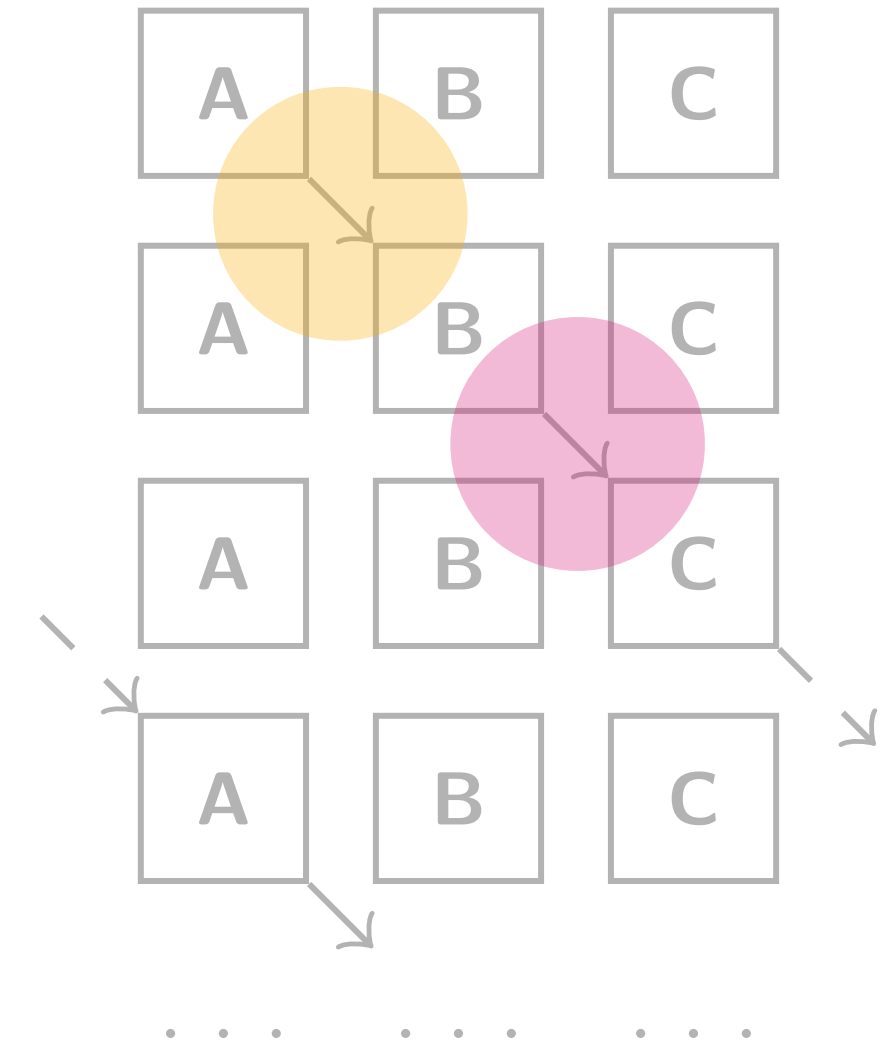
- Global types are inherently **synchronous**
 - Projection provides only one possible ordering
- Interactions can be **reordered** for efficiency while preserving safety
 1. Data **dependencies** must be preserved



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

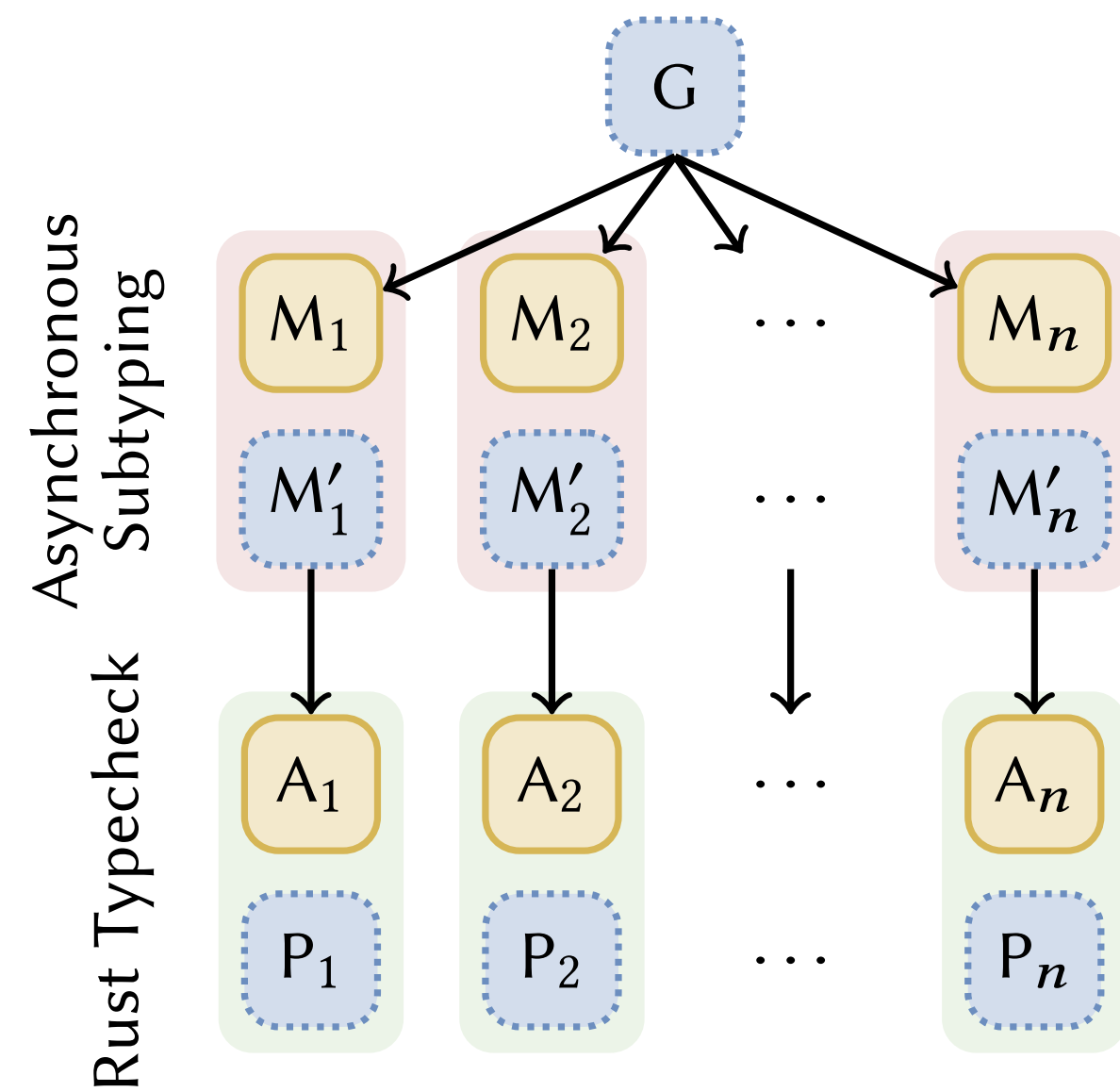
M Finite State Machine (FSM)

M' Optimised FSM

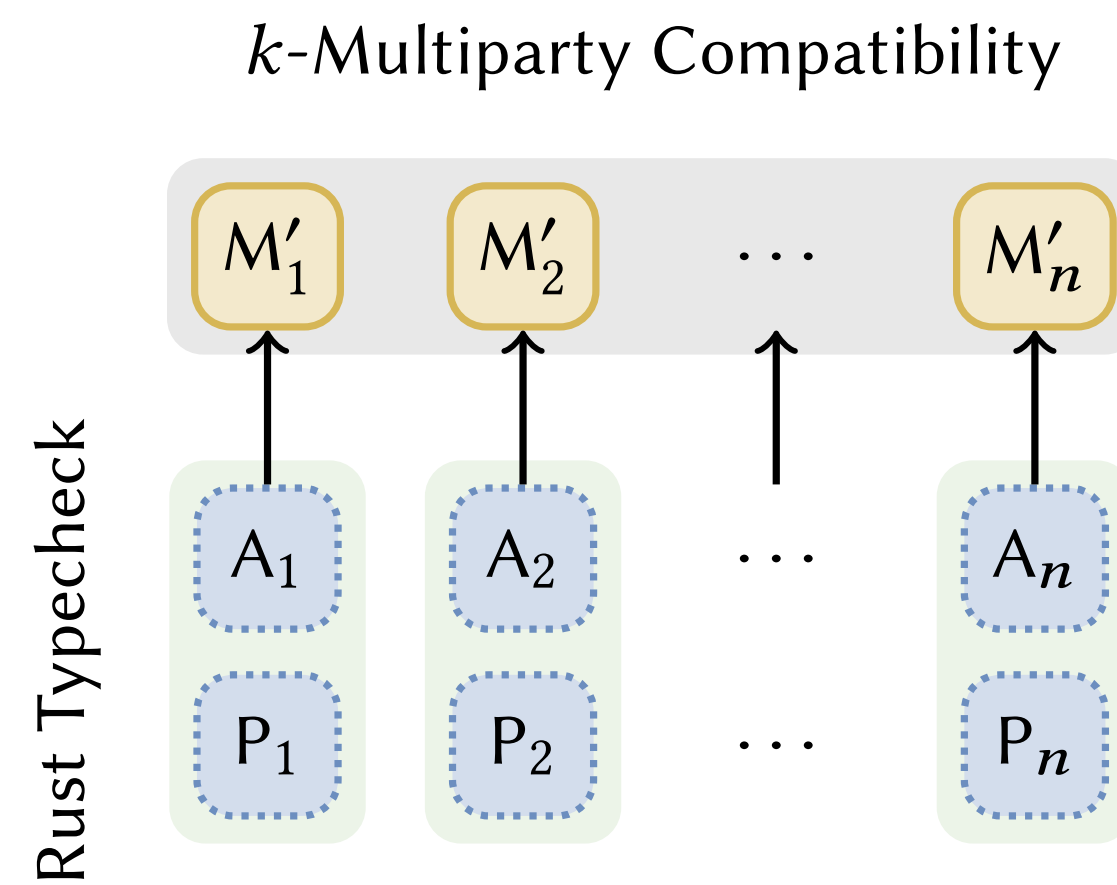
A Rust API

P Rust Process

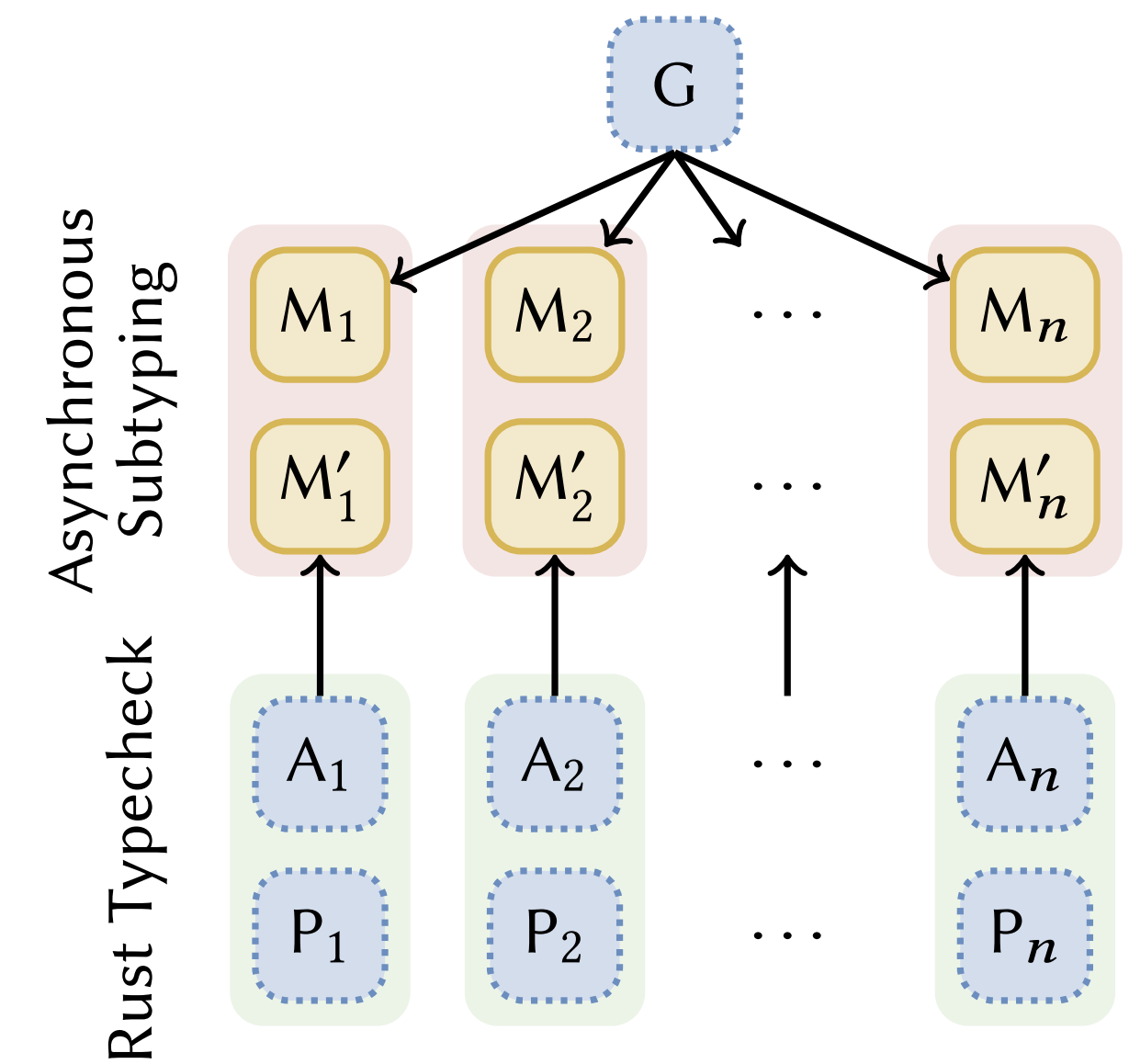
● User-Written ● Generated



(a) Top-down



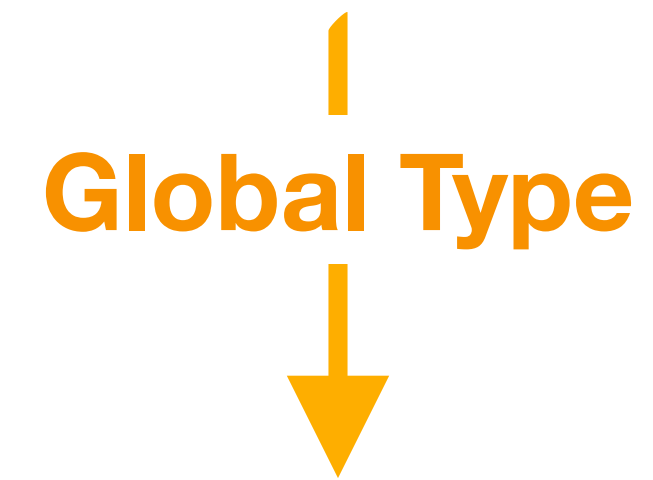
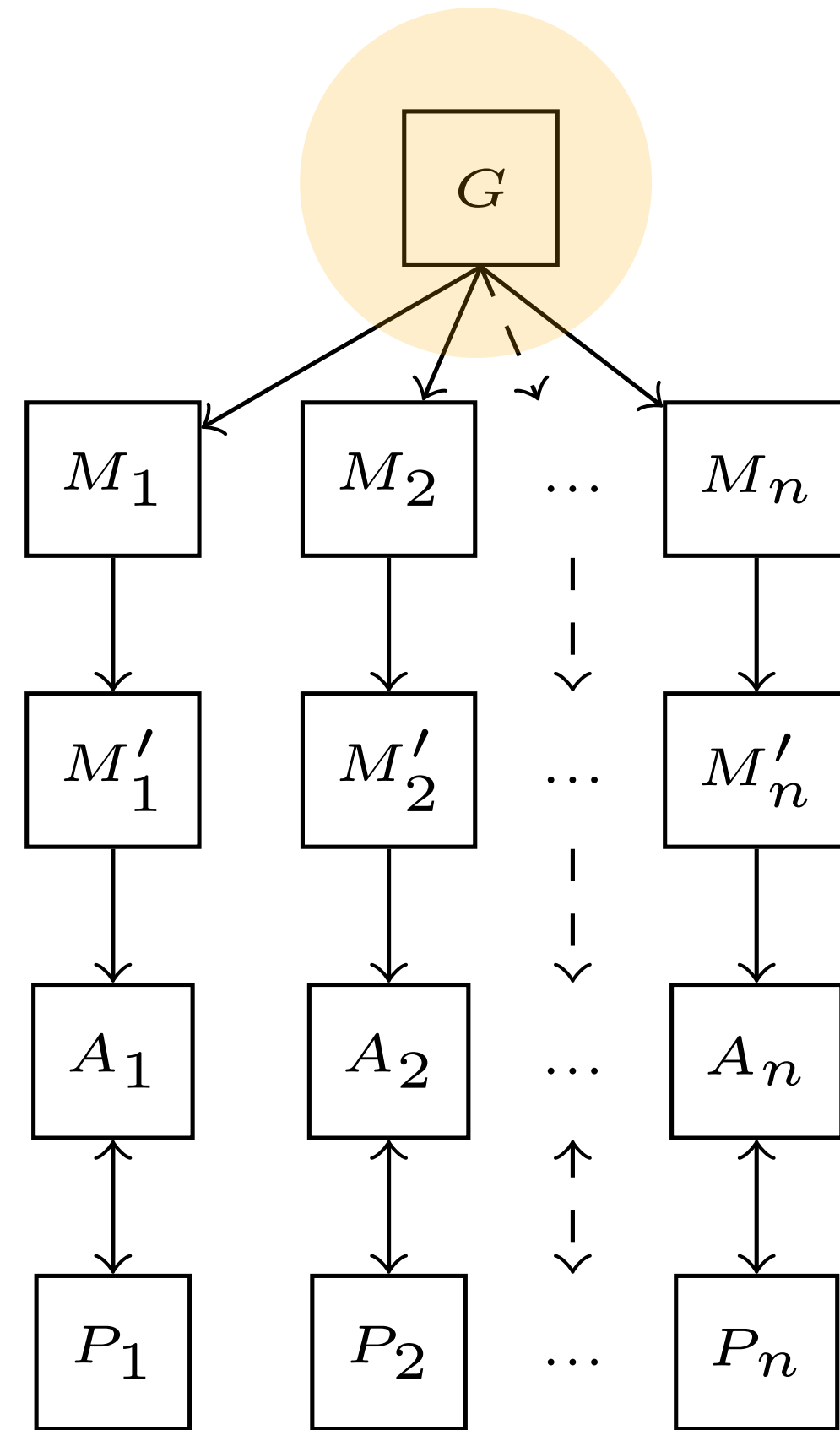
(b) Bottom-up



(c) Hybrid

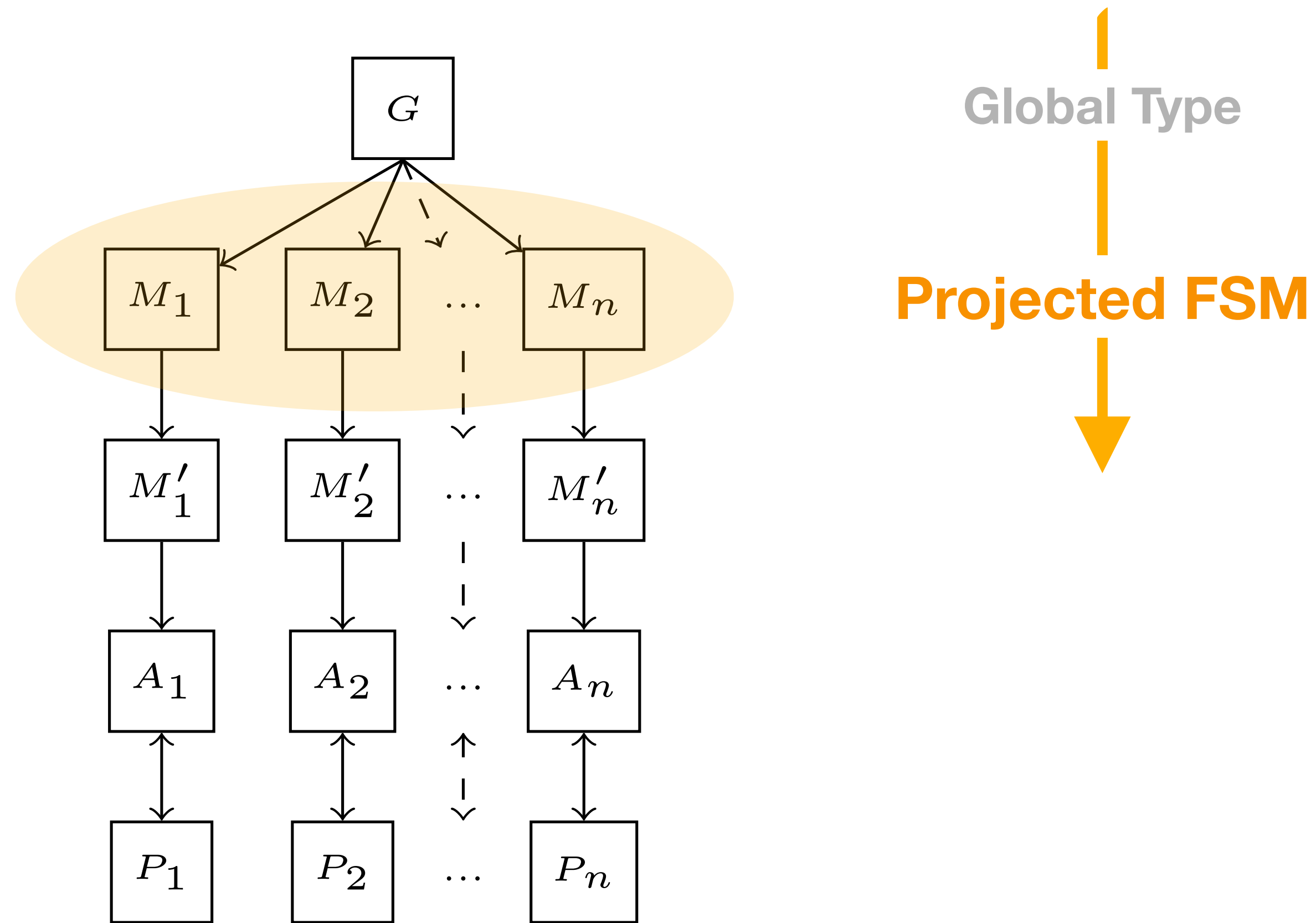
Workflow

Top-Down Approach



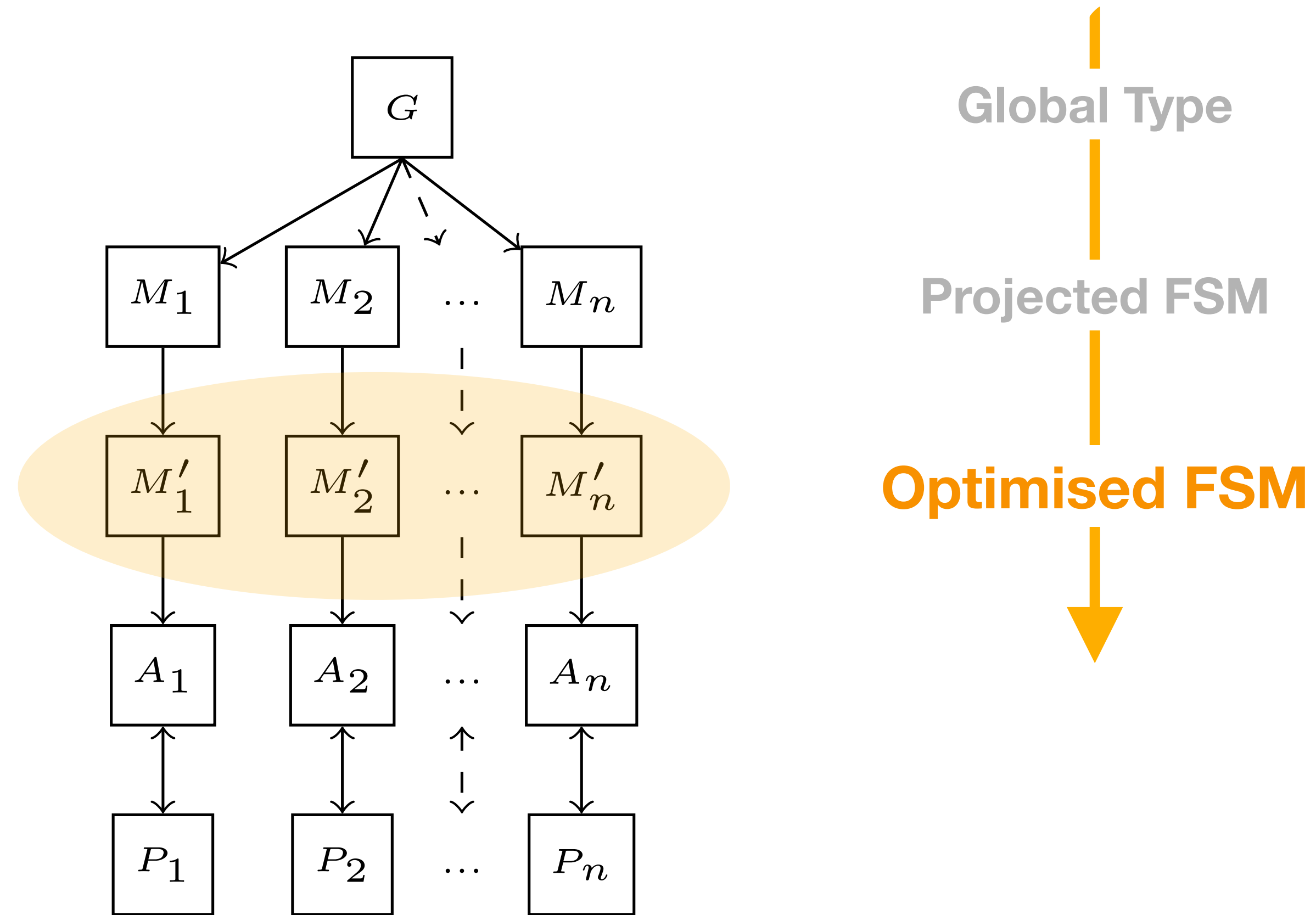
Workflow

Top-Down Approach



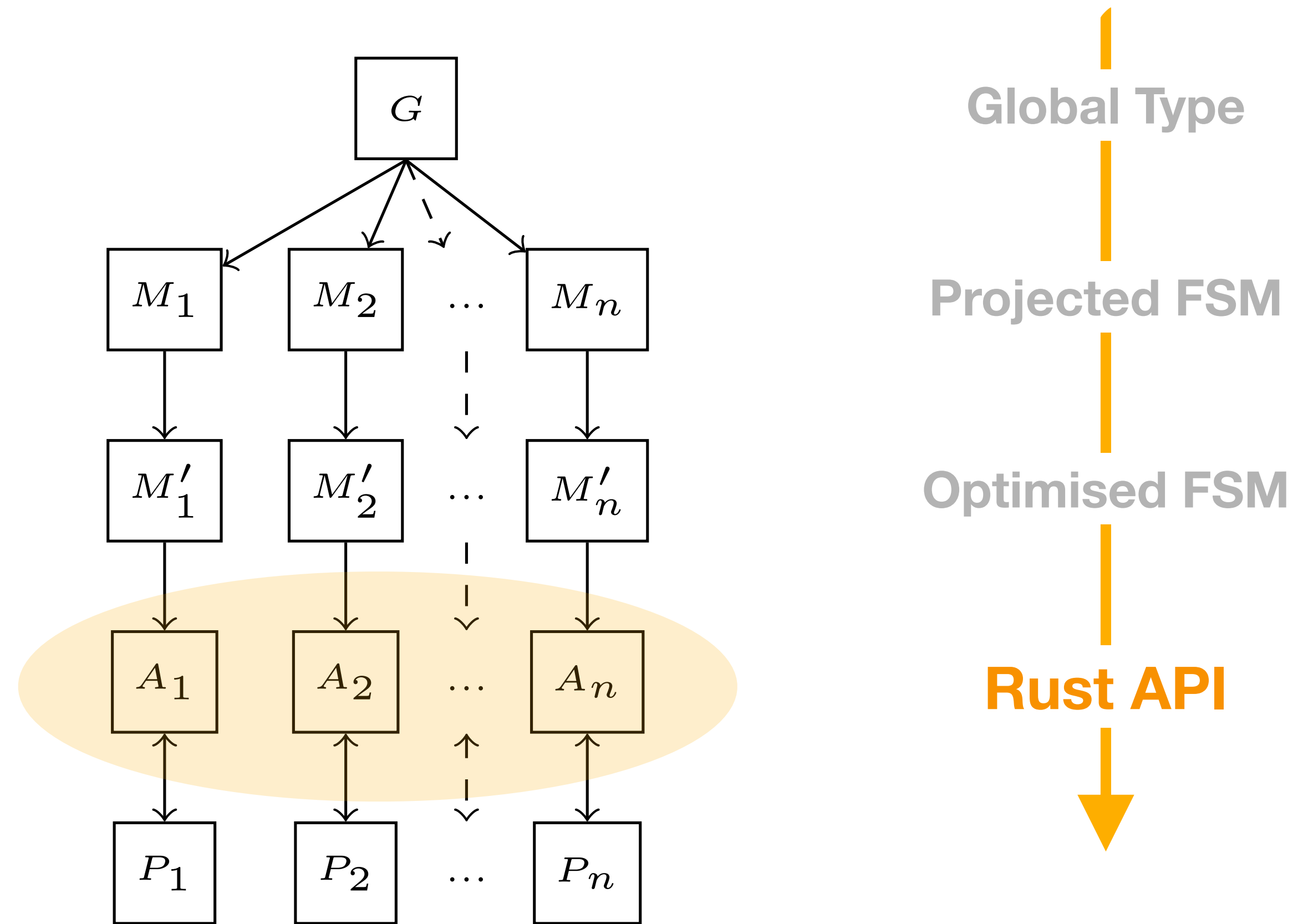
Workflow

Top-Down Approach



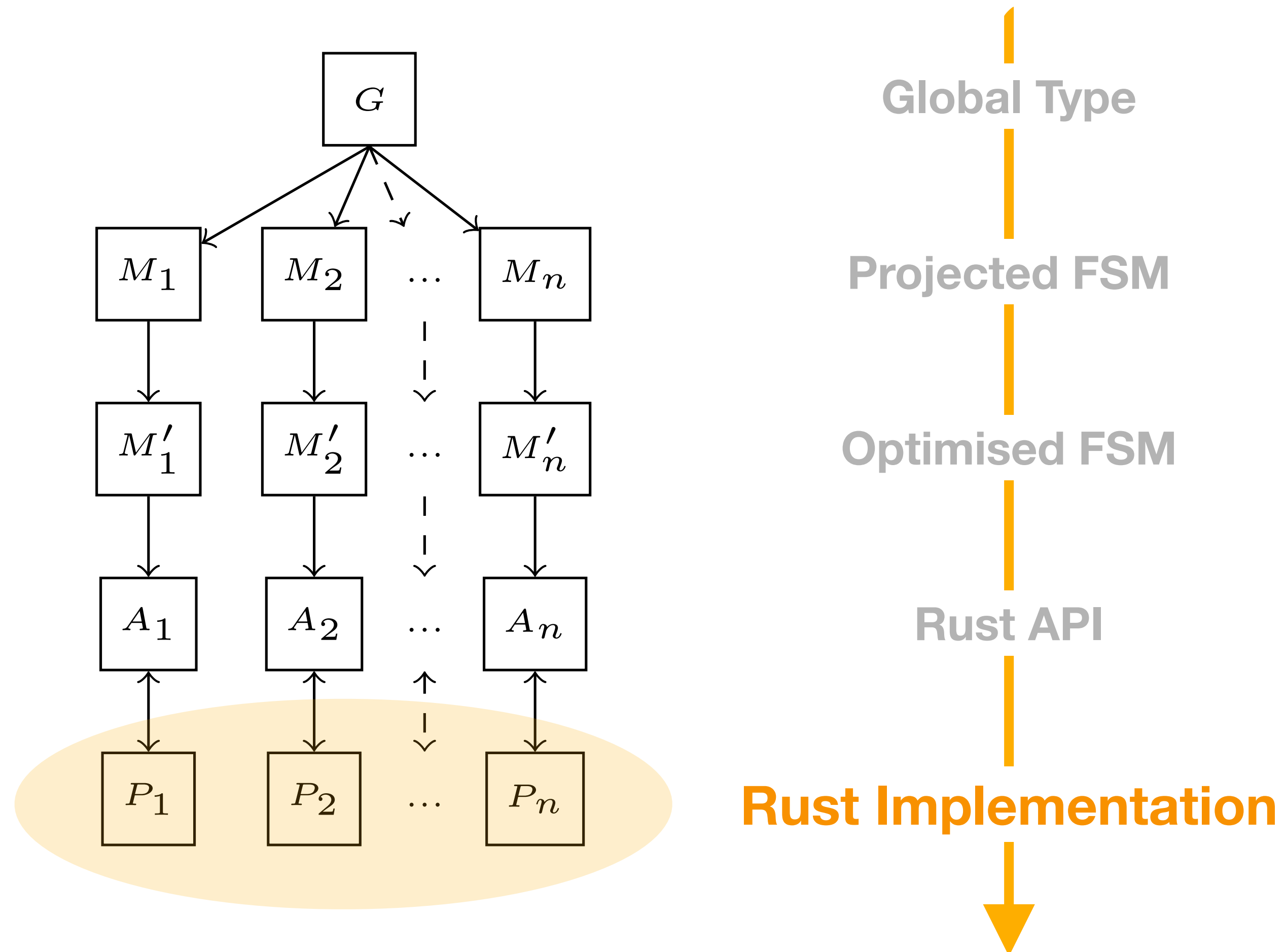
Workflow

Top-Down Approach



Workflow

Top-Down Approach



Ring Protocol

Example

Global Type

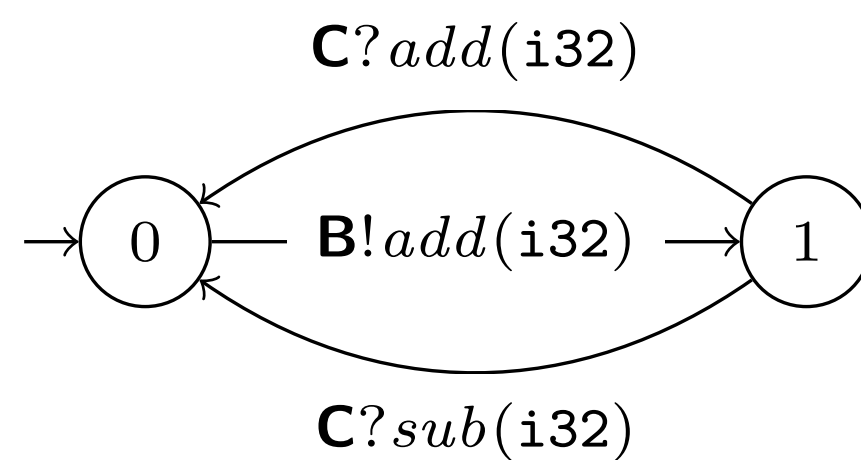
$$G = \mu t. \mathbf{A} \rightarrow \mathbf{B} : \left\{ \begin{array}{l} \mathit{add}(\mathit{i32}).\mathbf{B} \rightarrow \mathbf{C} : \left\{ \begin{array}{l} \mathit{add}(\mathit{i32}).\mathbf{C} \rightarrow \mathbf{A} : \{\mathit{add}(\mathit{i32}).\mathbf{t}\} \\ \mathit{sub}(\mathit{i32}).\mathbf{C}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

Ring Protocol

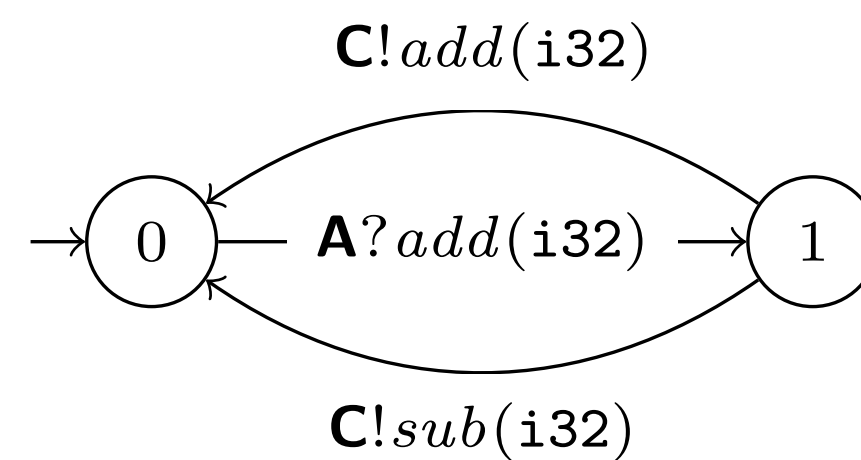
Example

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

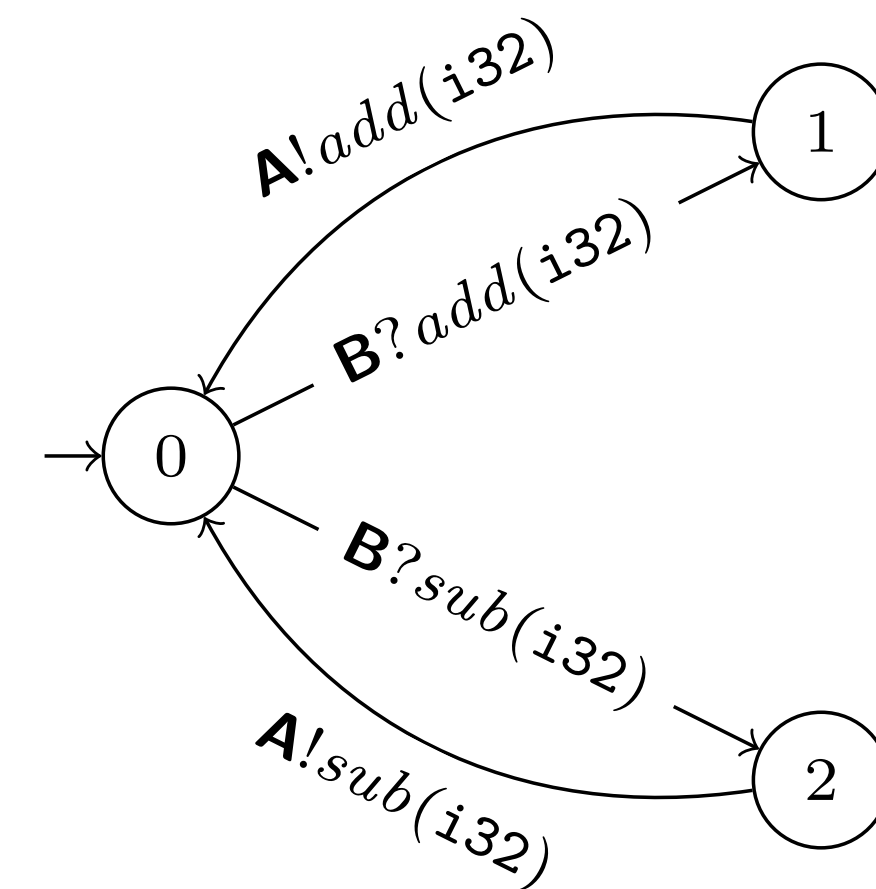
PROJECTION



PROJECTION

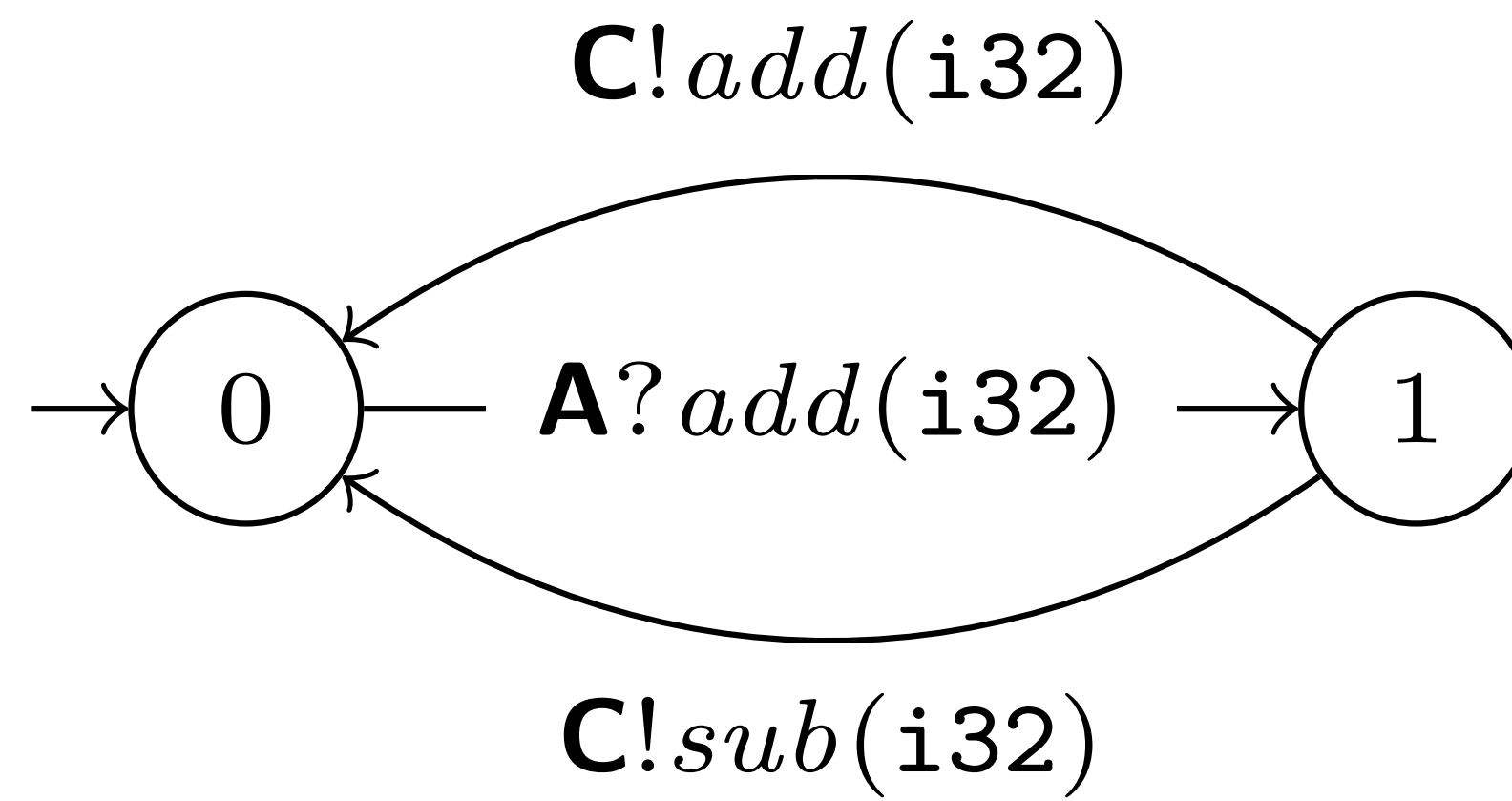


PROJECTION



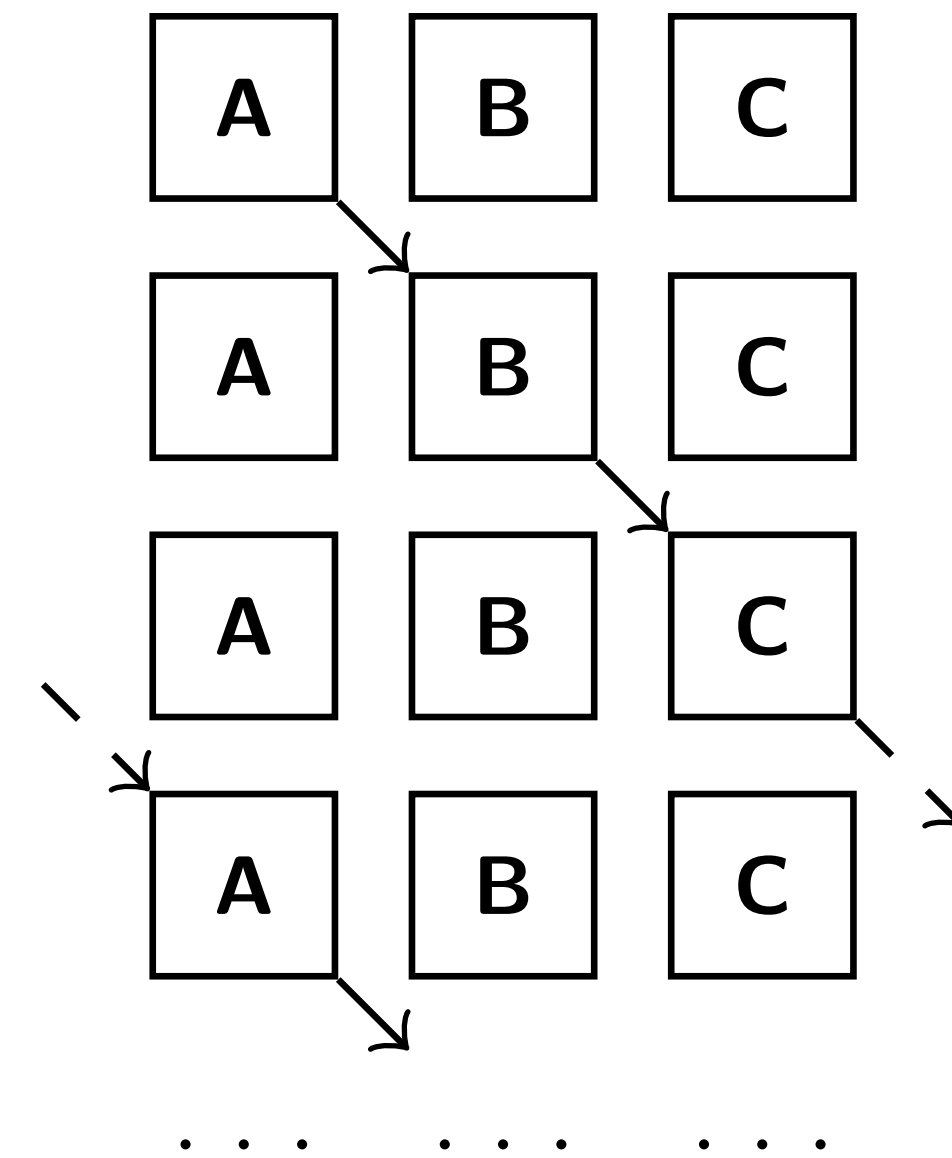
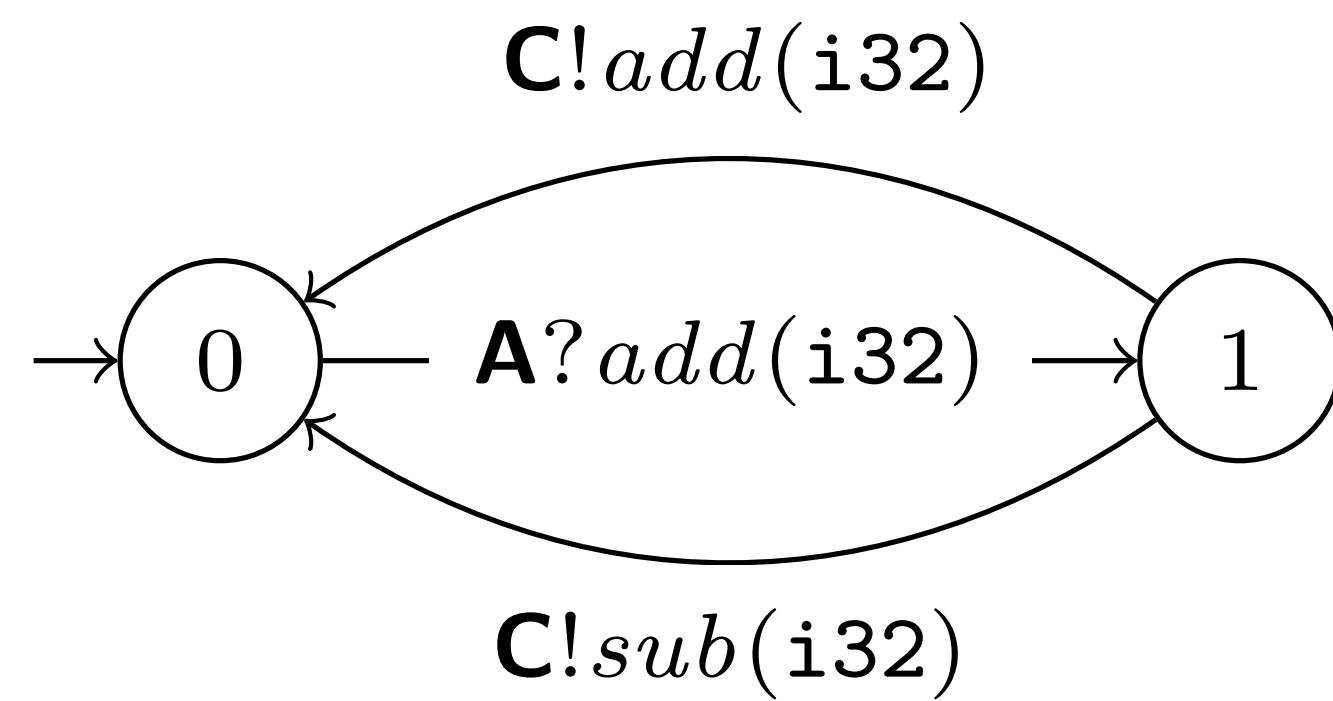
Ring Protocol

Example



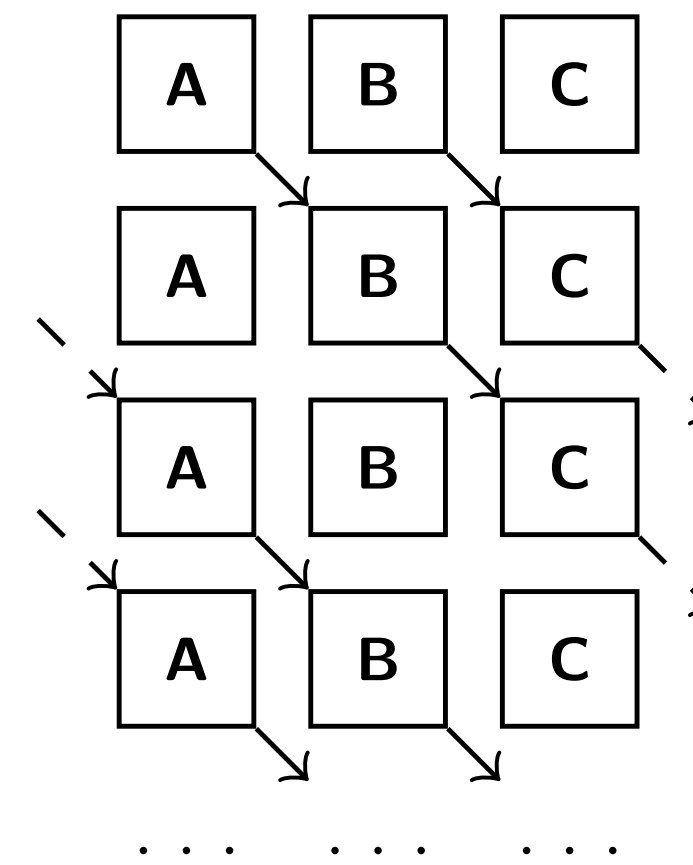
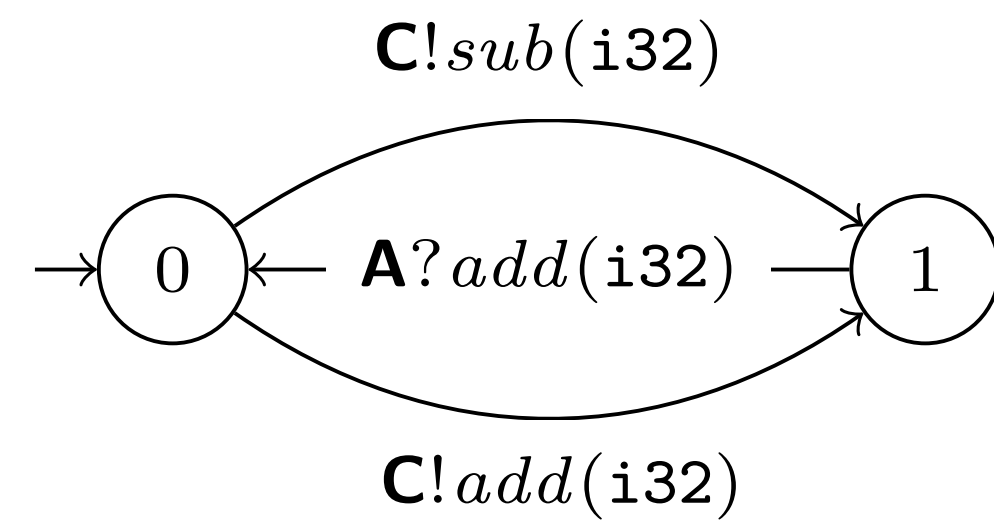
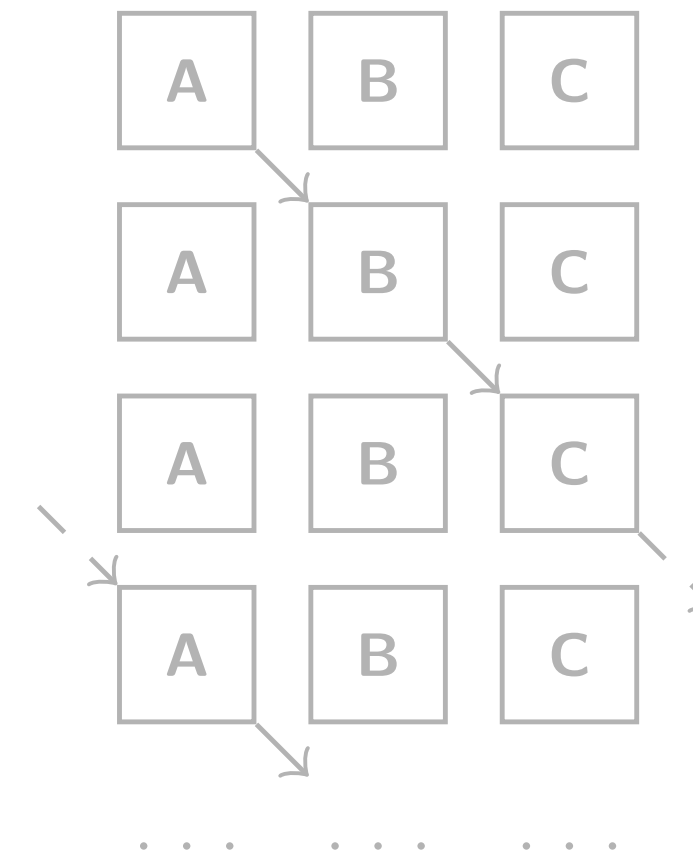
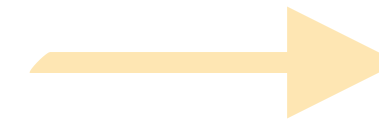
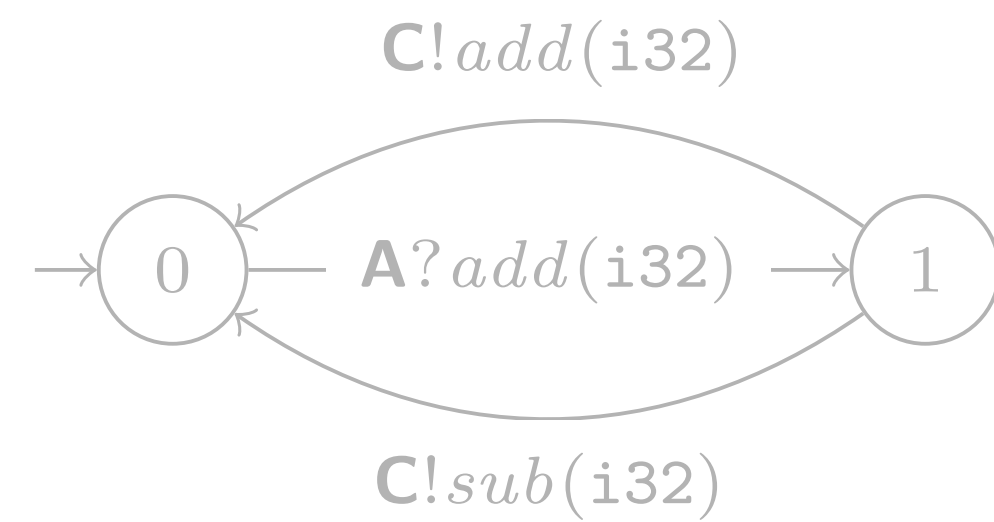
Ring Protocol

Example



Ring Protocol

Example



vScr

An Extensible Toolchain for Multiparty Session Types

- Accepts **Scribble** protocols
- Extracts **global types**
- Produces **local types** and **CFSMs**
- Generates **APIs** for implementation (currently supports Go, F★ and OCaml)
- Easily extendable (currently supports refinement types and nested protocols as major extensions)

vScr

A Playground for Ideas

- 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 messages and their directions. The transitions are:

- 1 to 2: S!HELLO(u: int)
- 2 to 1: S?RES(f: 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.

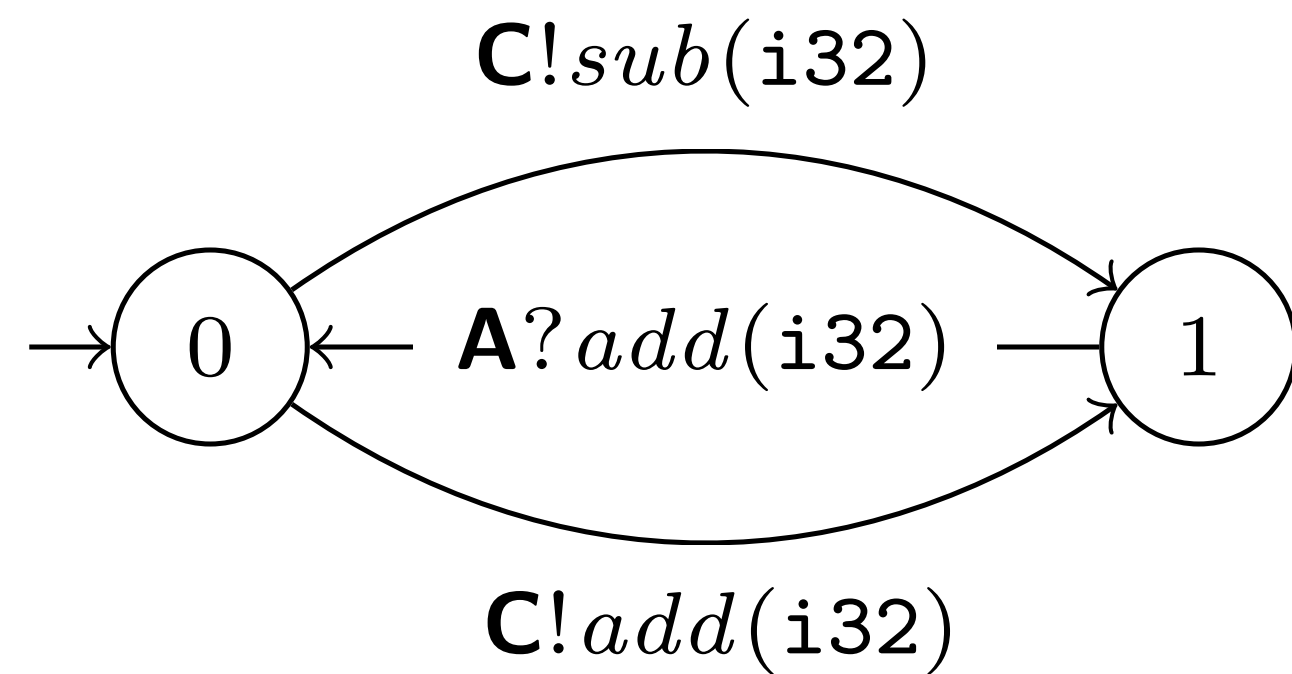
Scribble

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

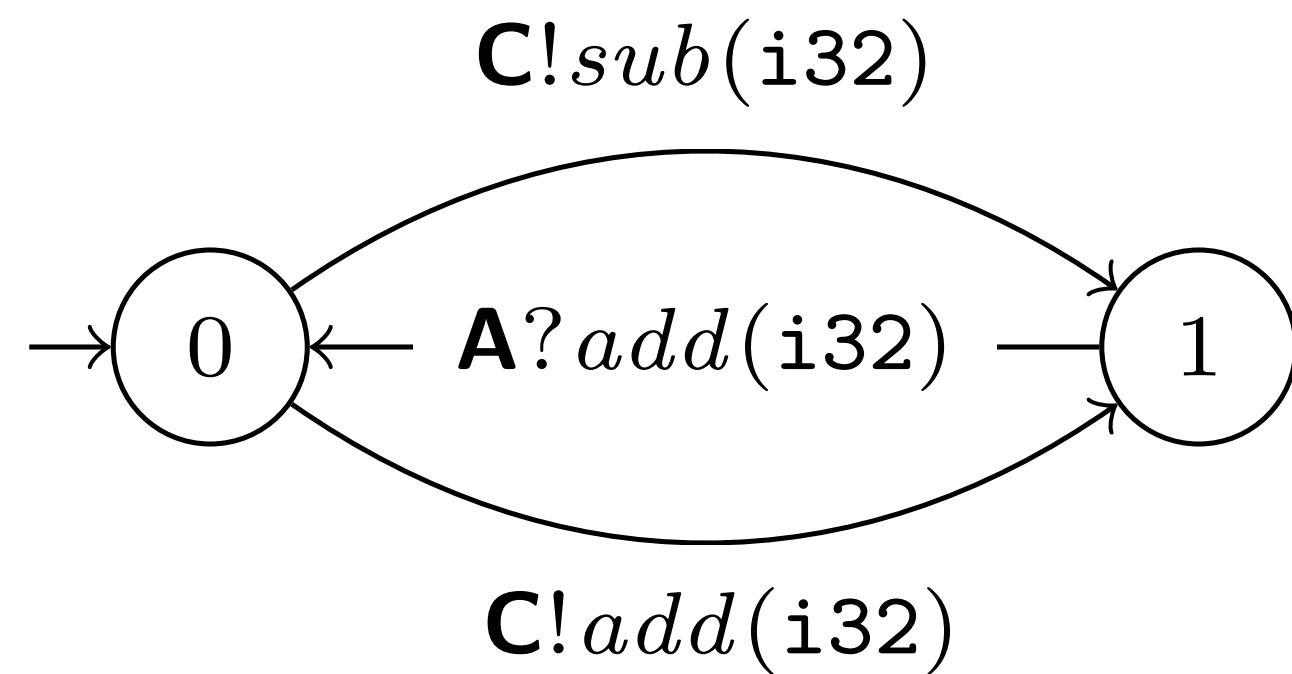
Rust API



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

Ring Protocol

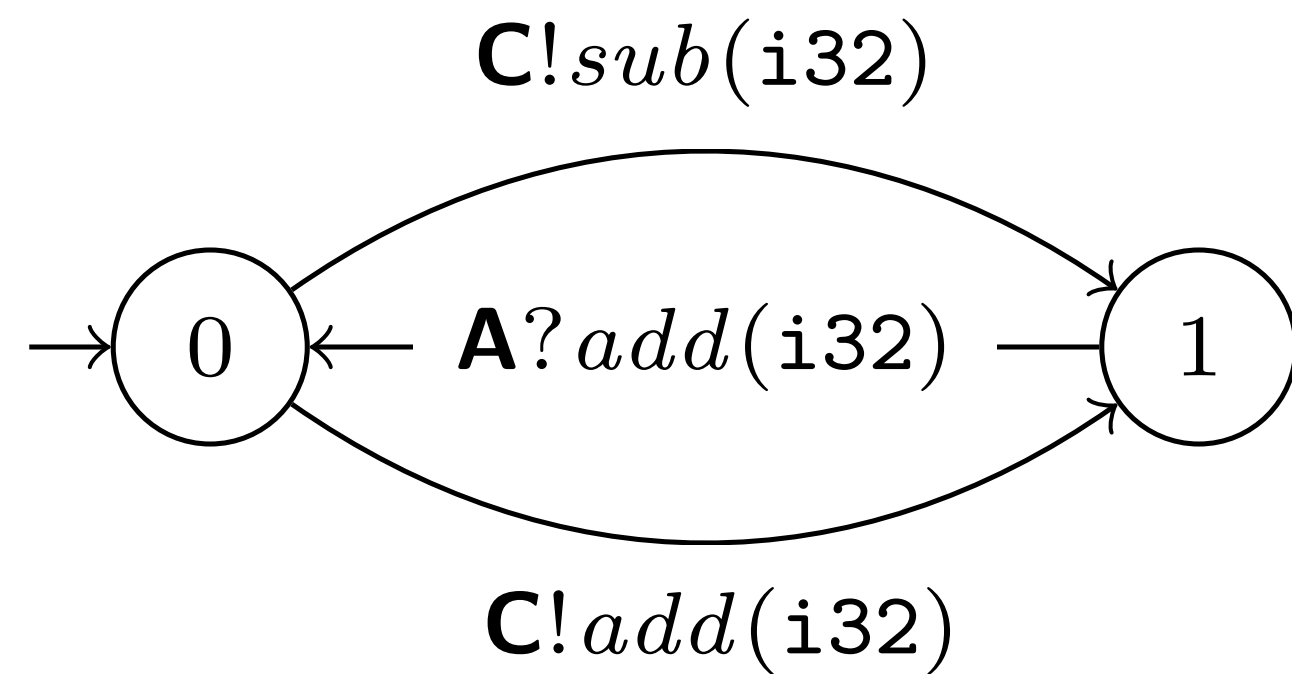
Rust API



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

Ring Protocol

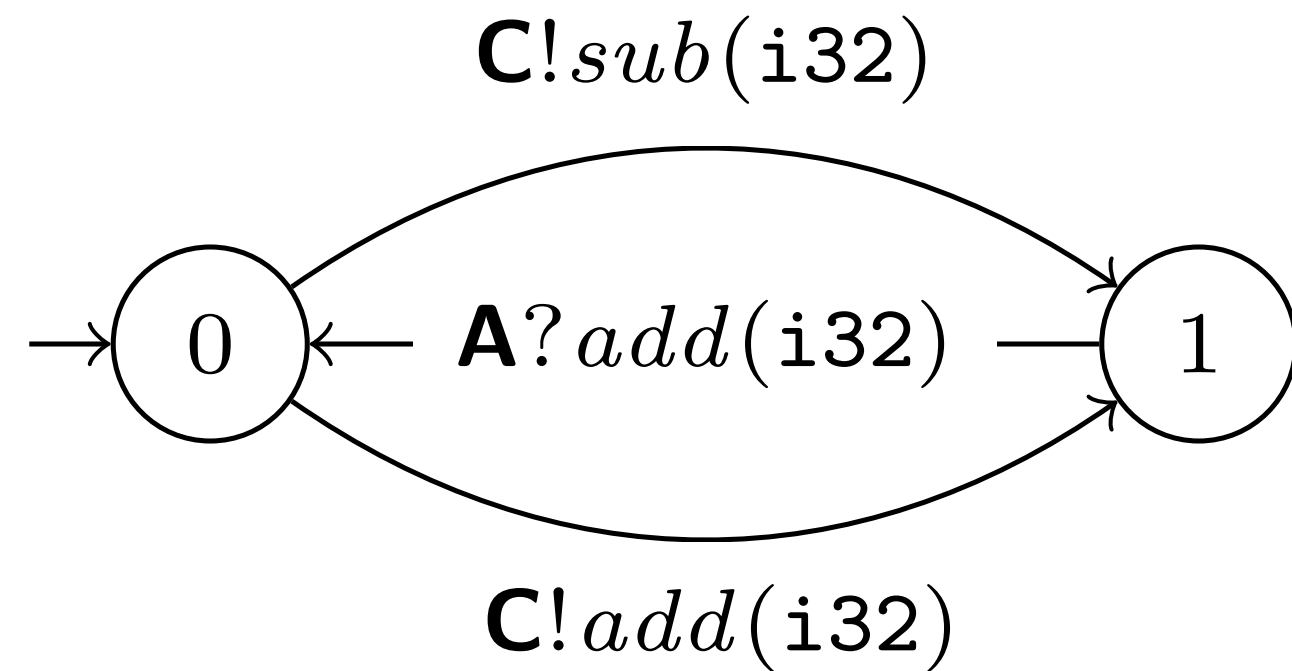
Rust API



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

Ring Protocol

Rust API



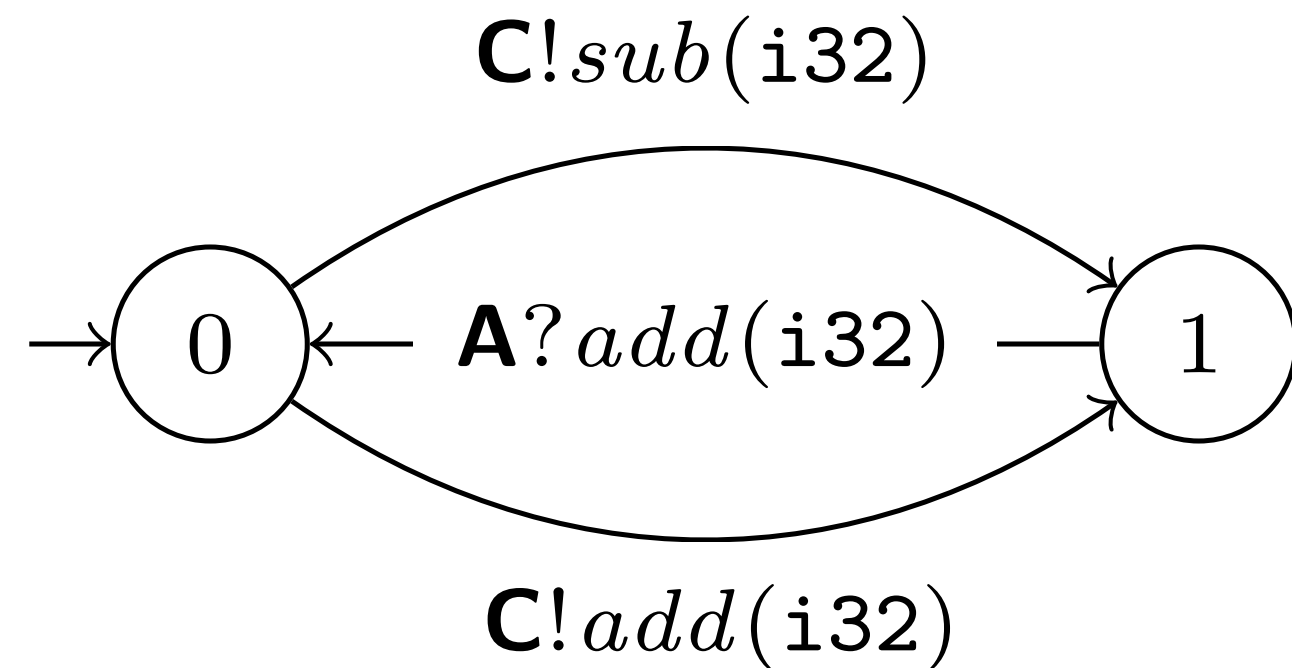
```
#[derive(Role)]  
#[message(Label)]  
struct B(#[route(A)] Receiver, #[route(C)] Sender);
```

```
#[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),
}

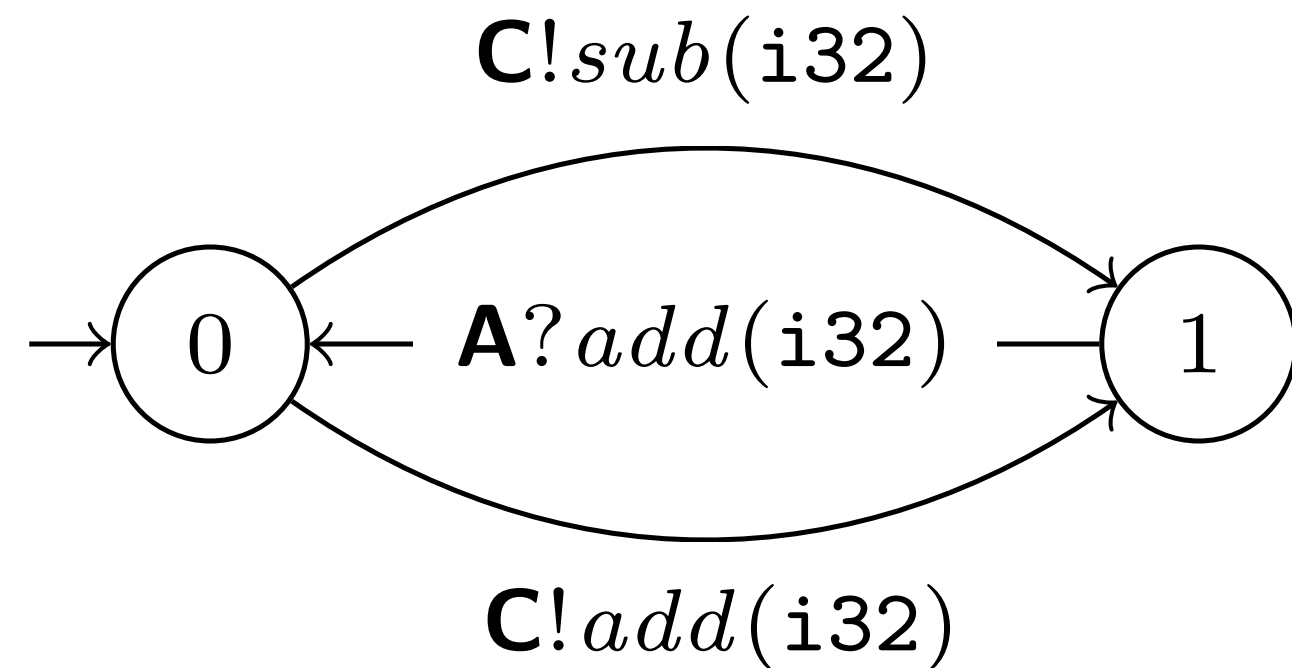
struct Add(i32);
struct Sub(i32);

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

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

Ring Protocol

Implementation

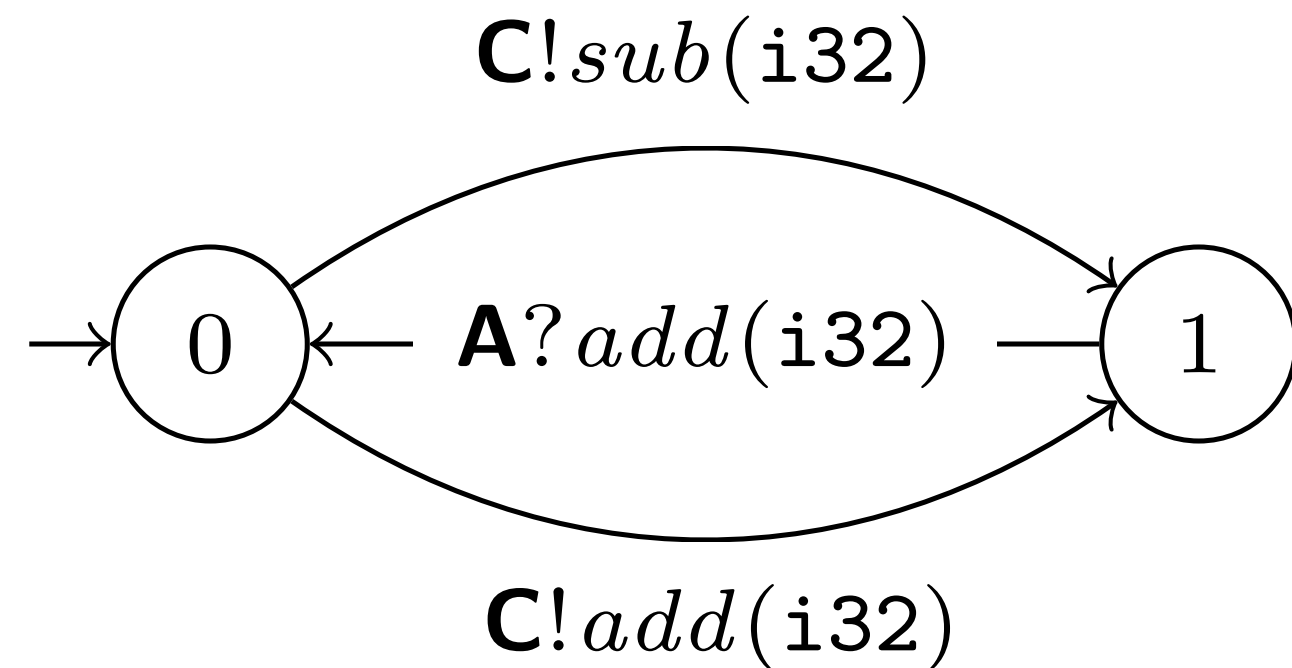


```
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

            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

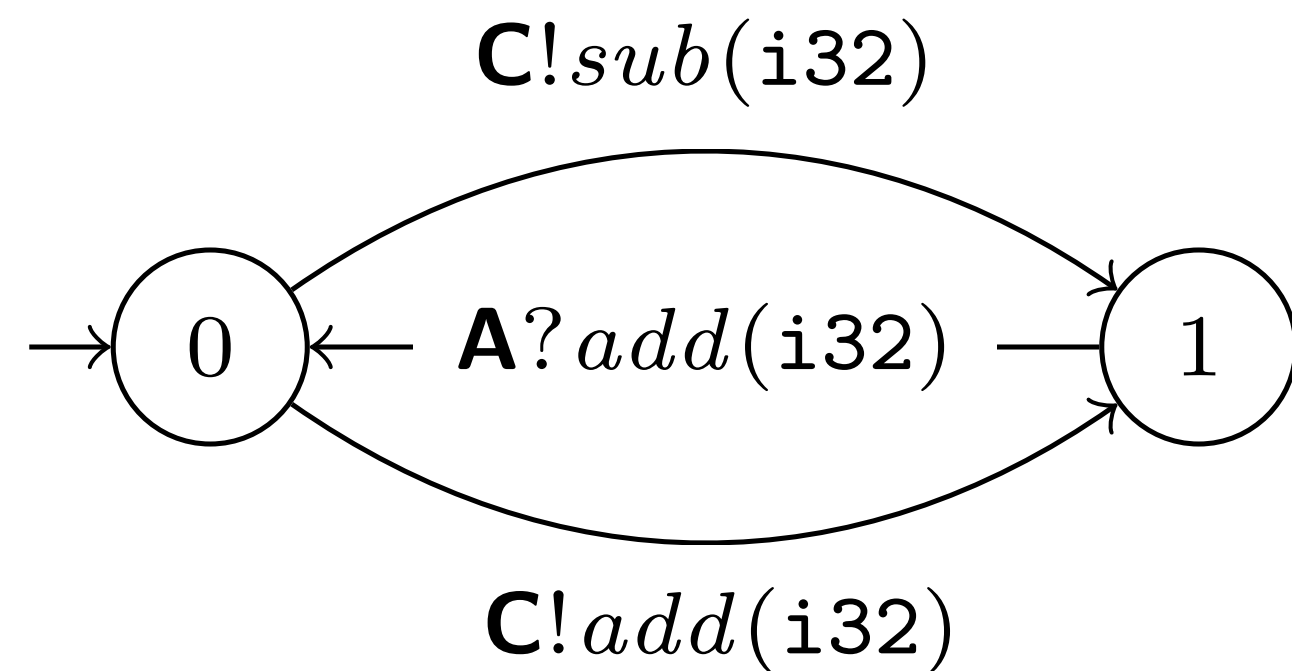


```
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

            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

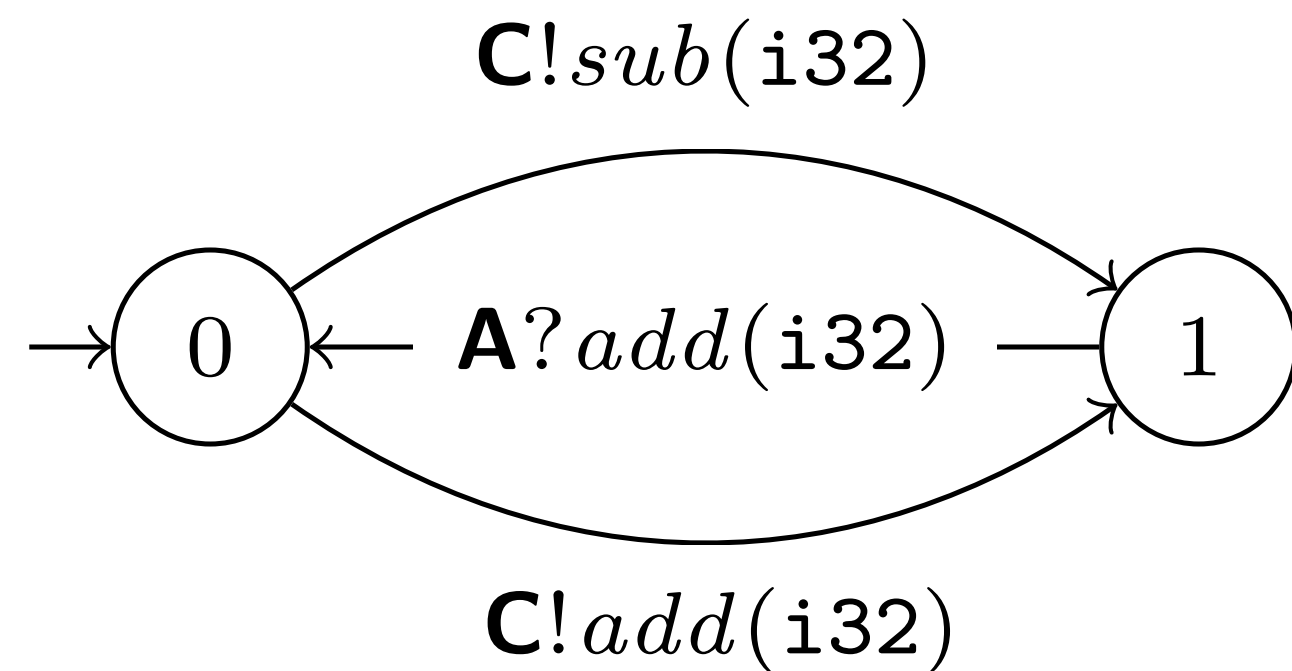


```
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

            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

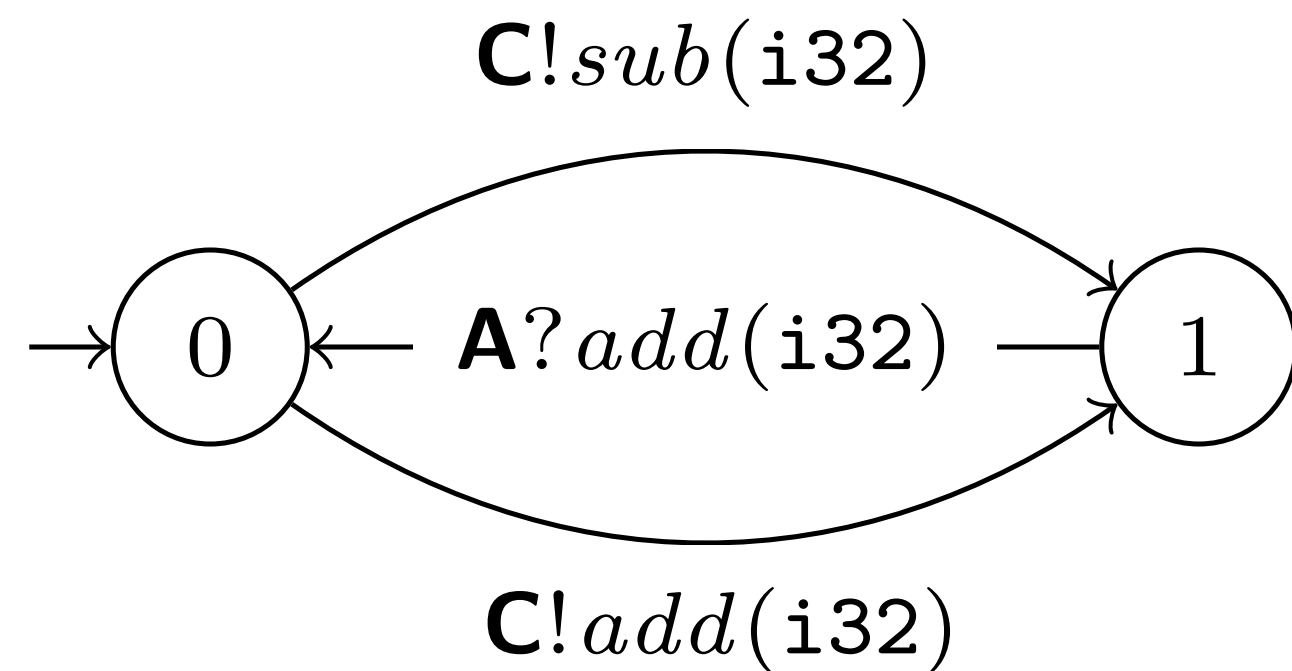


```
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

            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

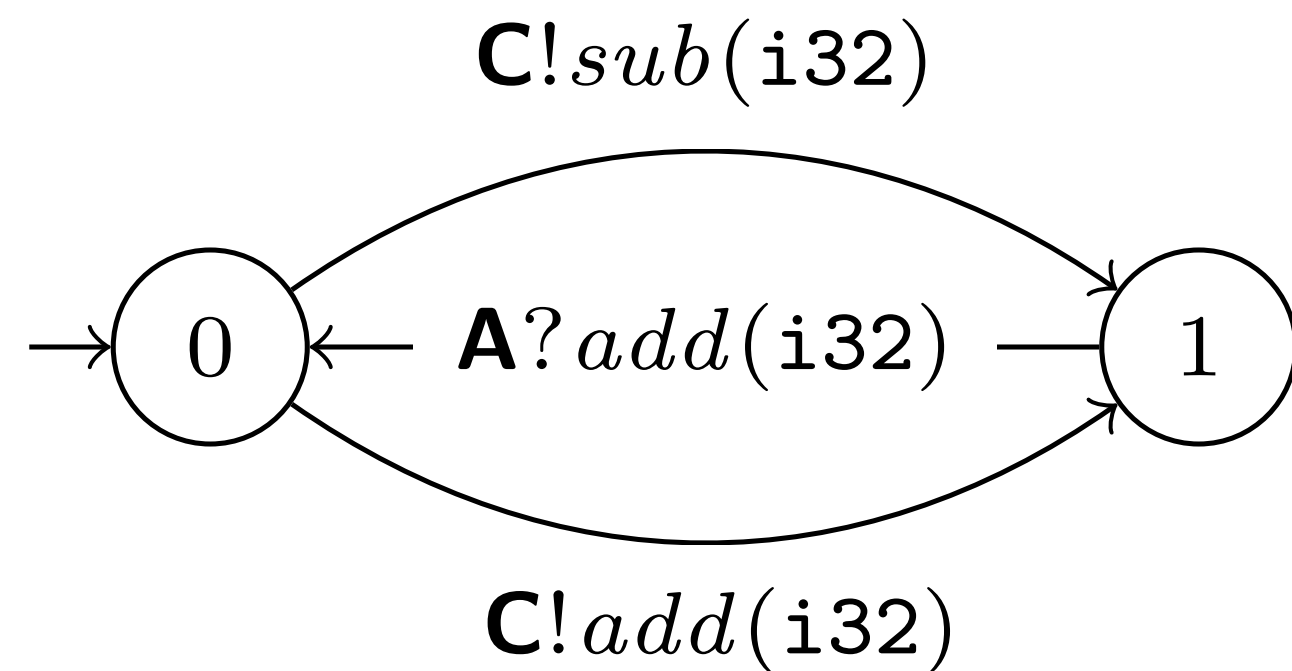


```
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

            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

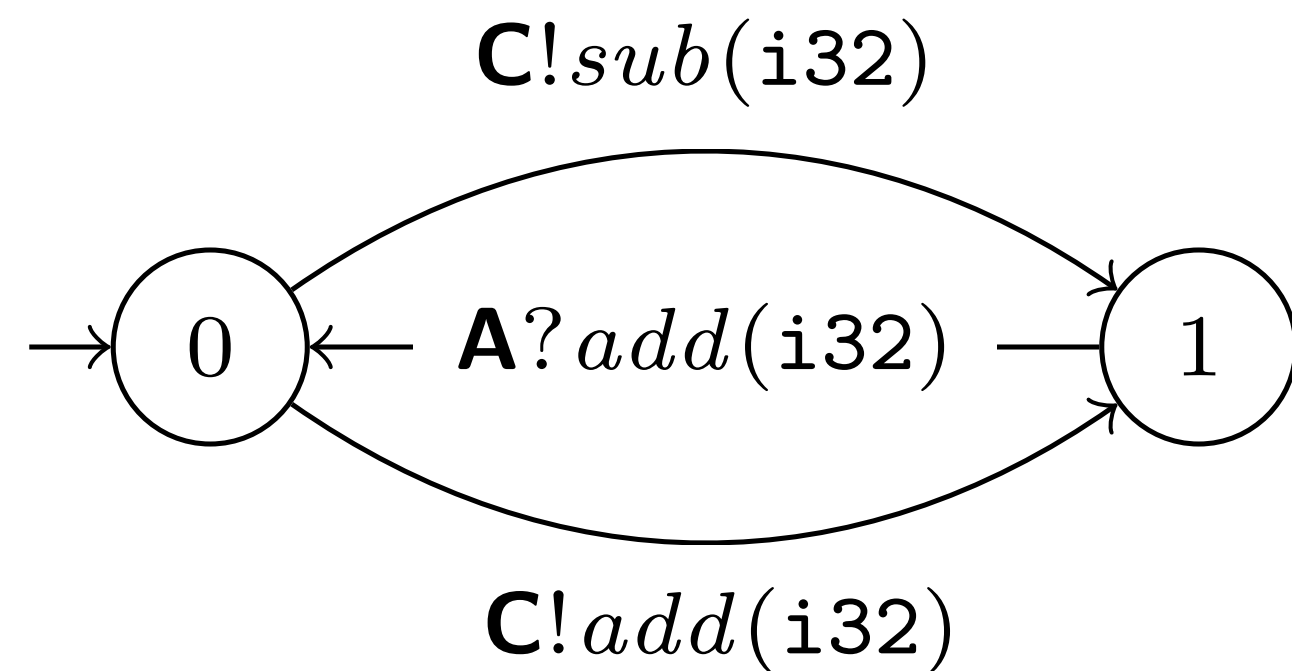


```
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

            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

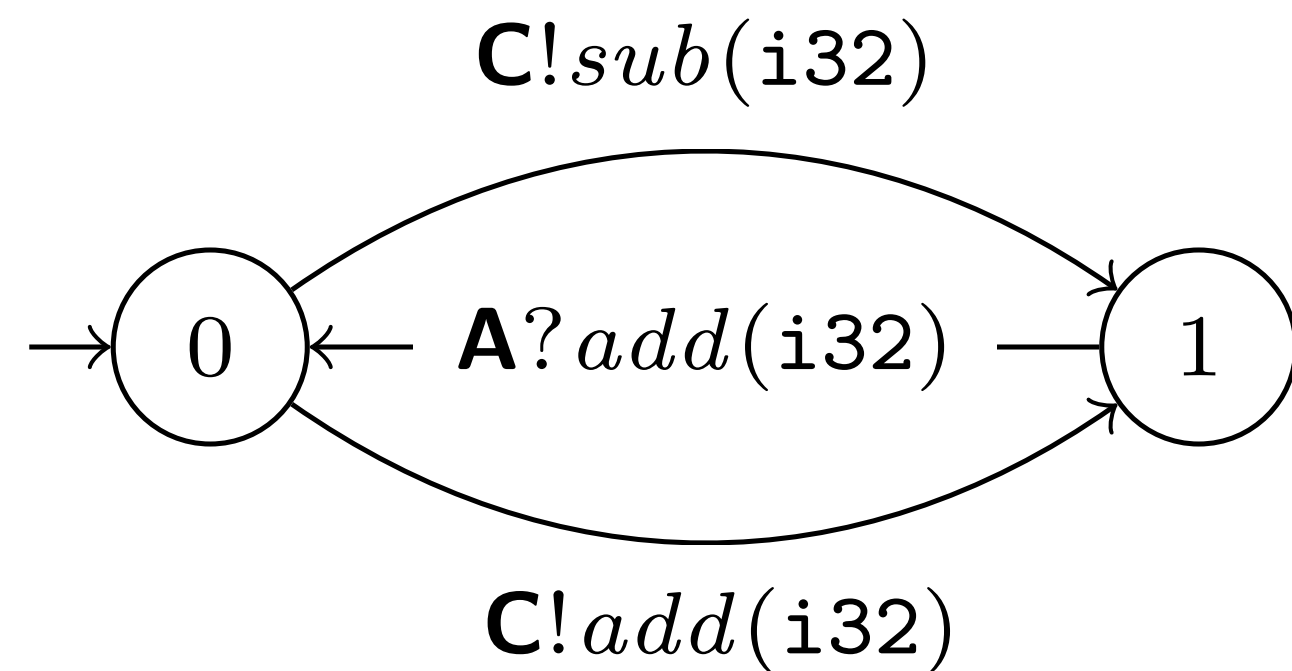


```
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

            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

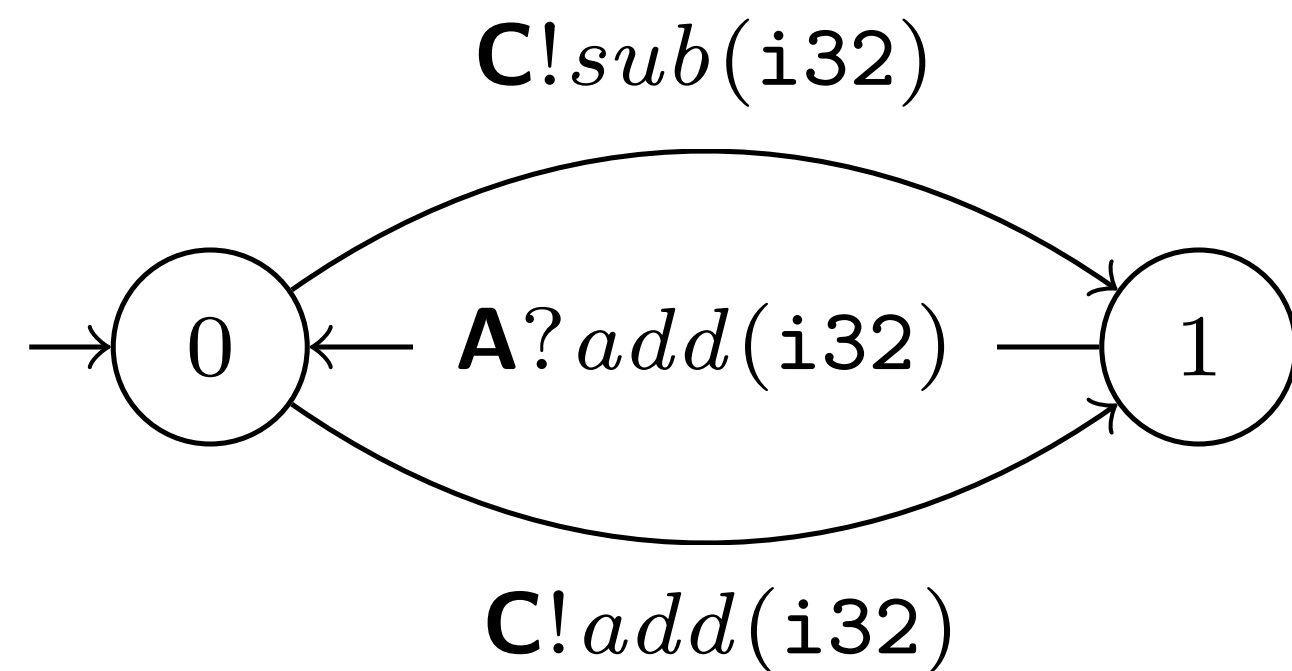


```
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

            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

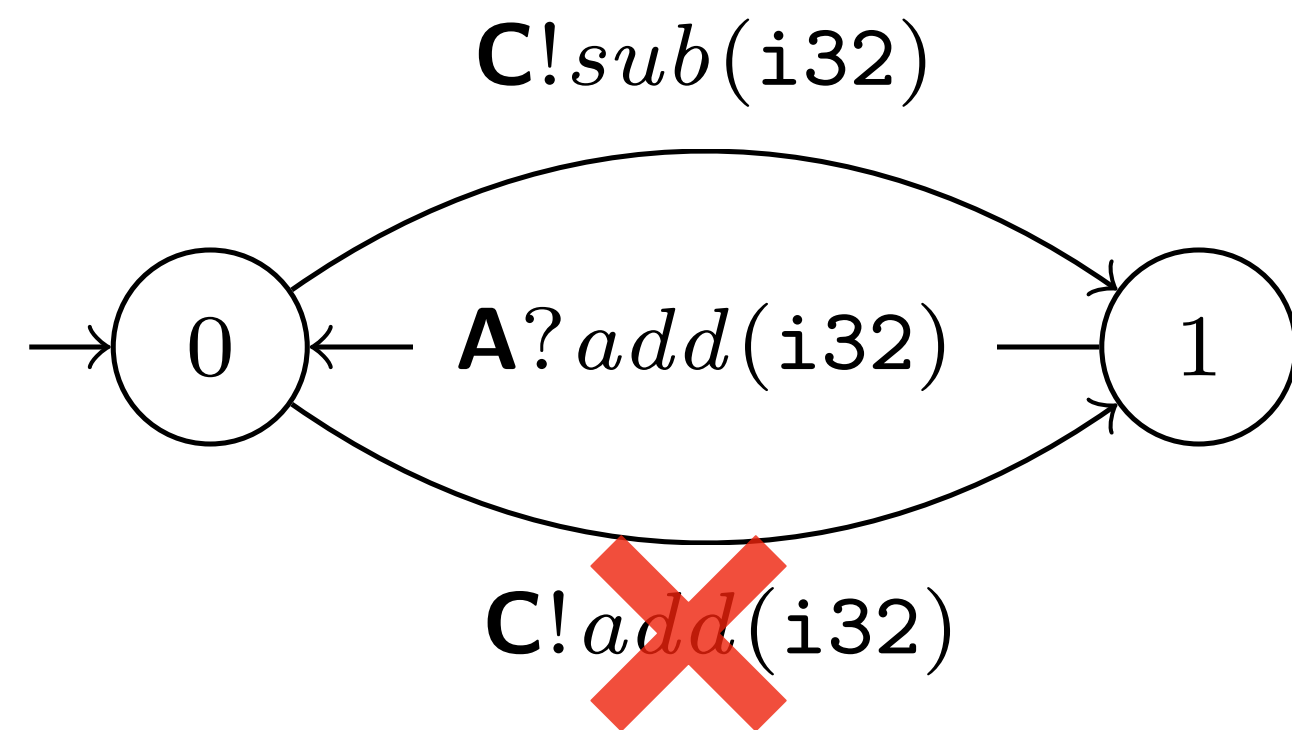


```
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

            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

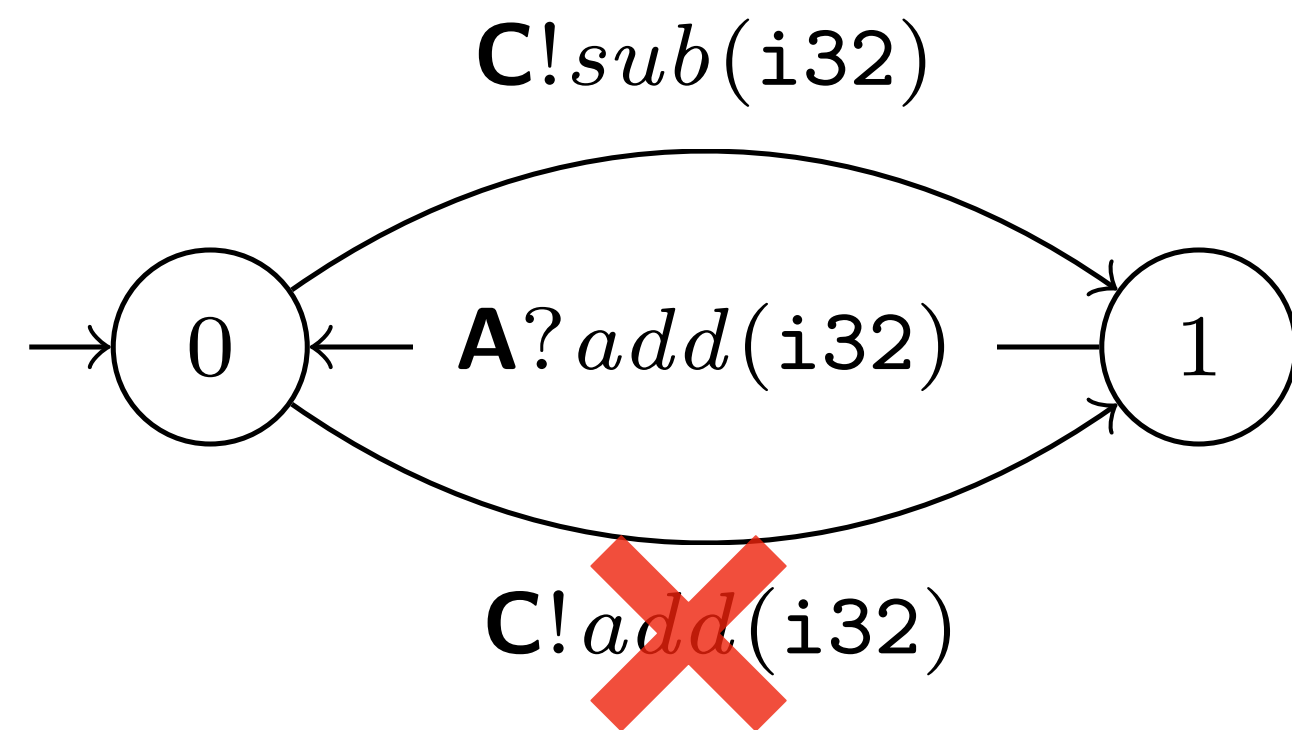


```
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

            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

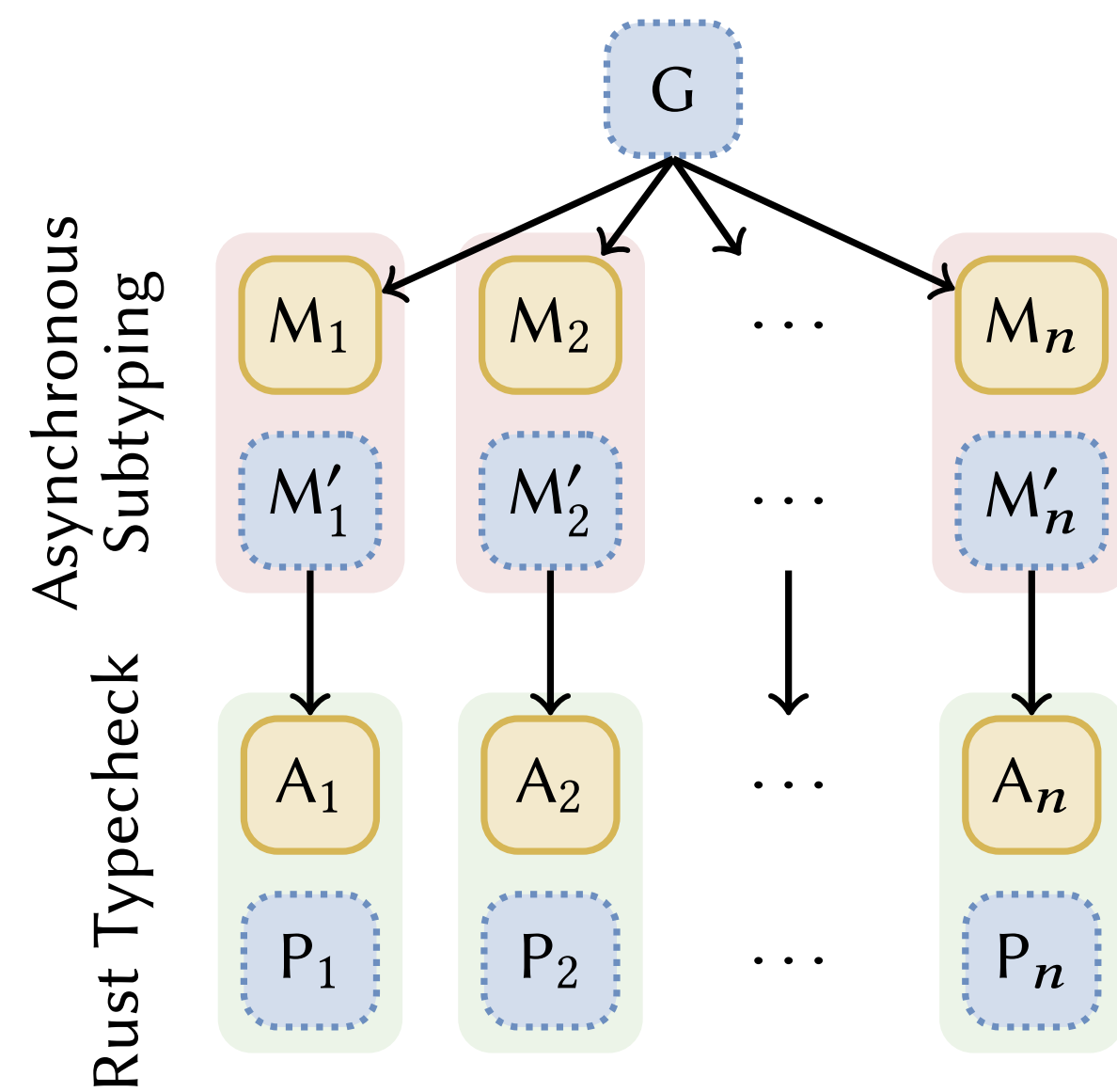
G Global Type

M Finite State Machine (FSM)

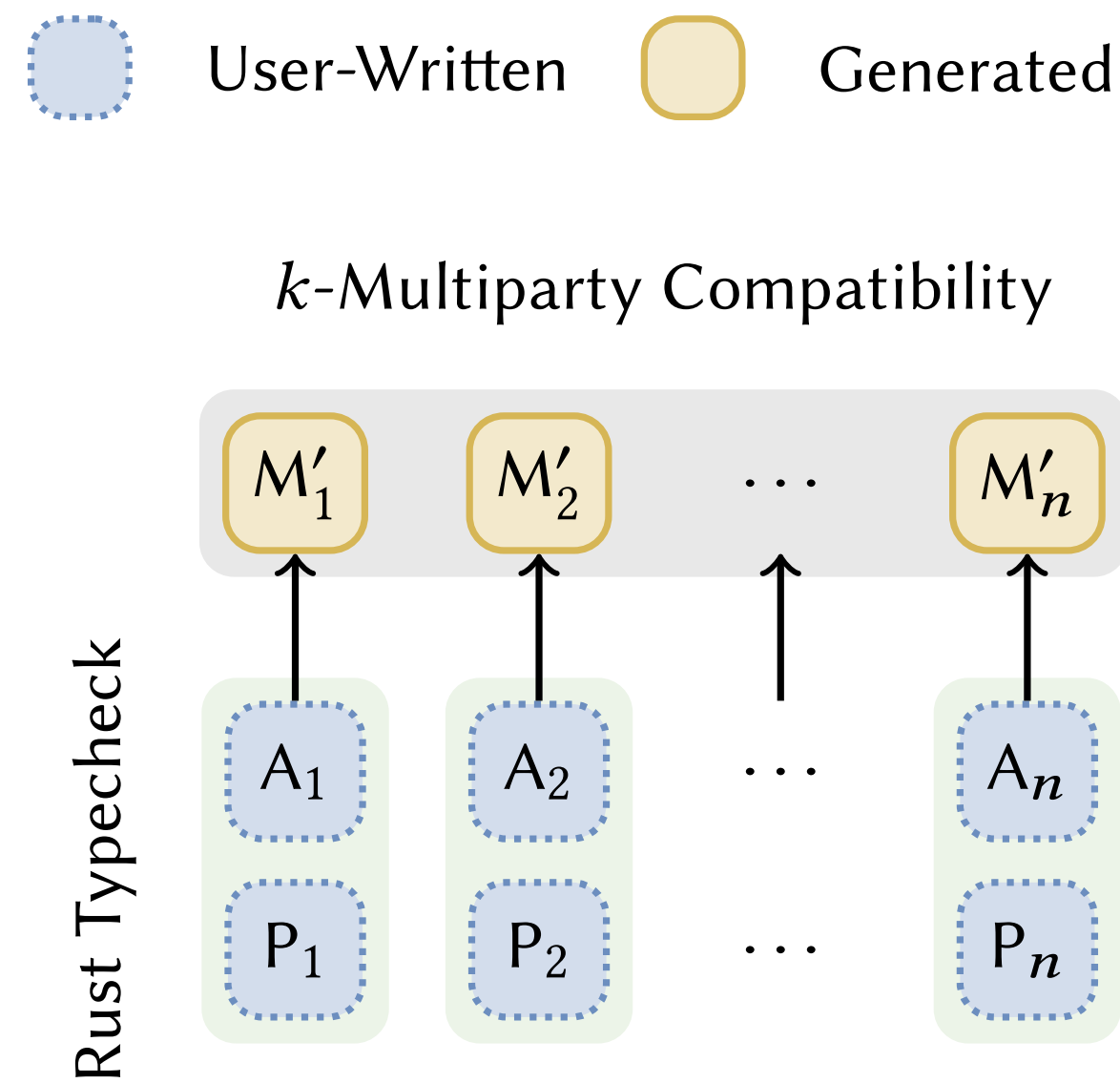
M' Optimised FSM

A Rust API

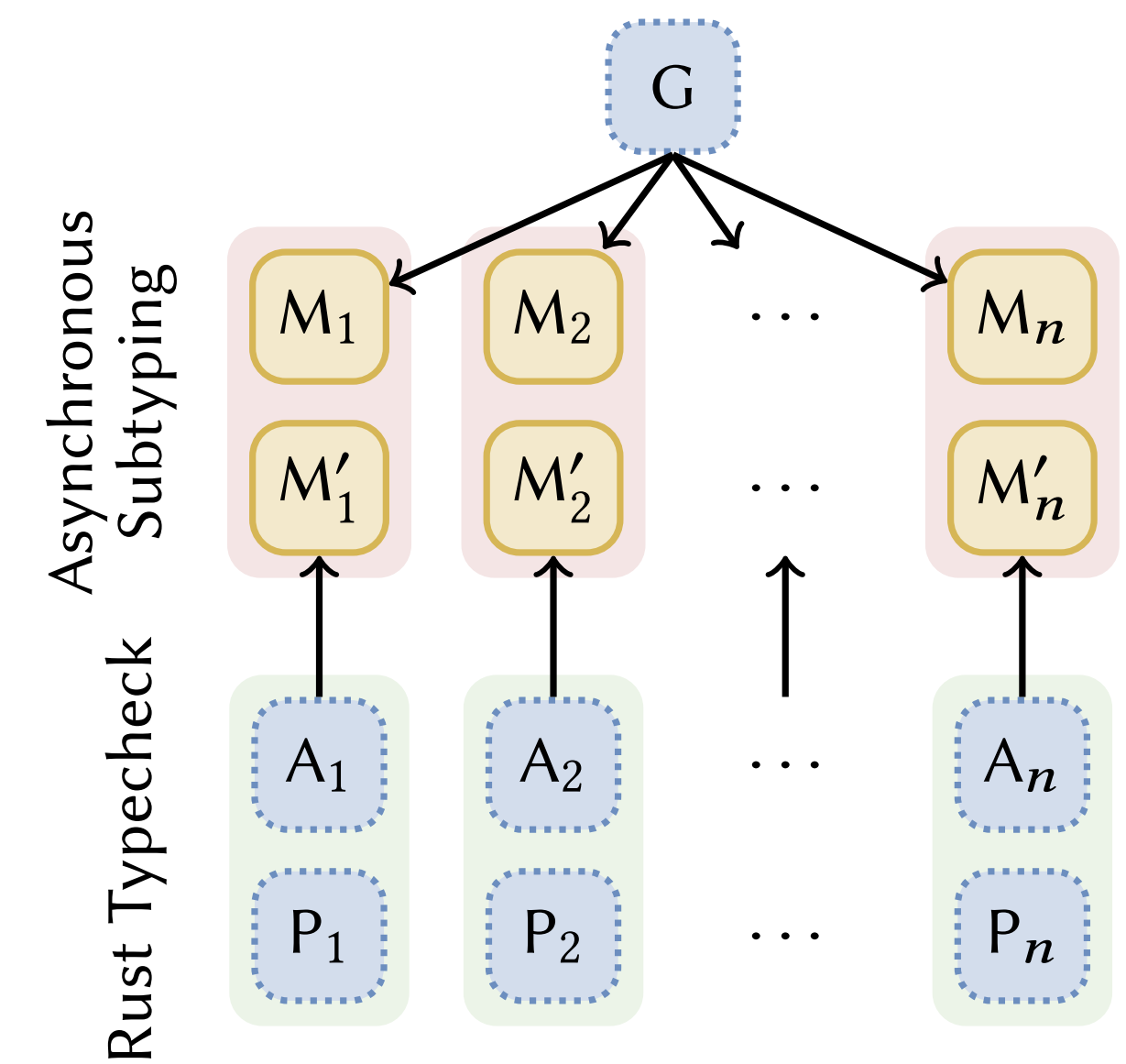
P Rust Process



(a) Top-down



(b) Bottom-up



(c) Hybrid

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

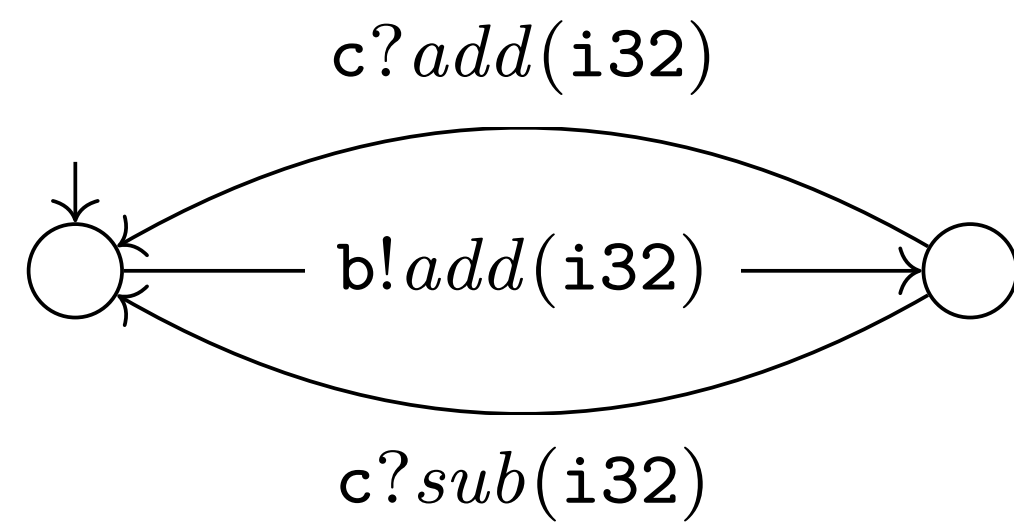
How do we check that asynchronous reorderings are *safe*?

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

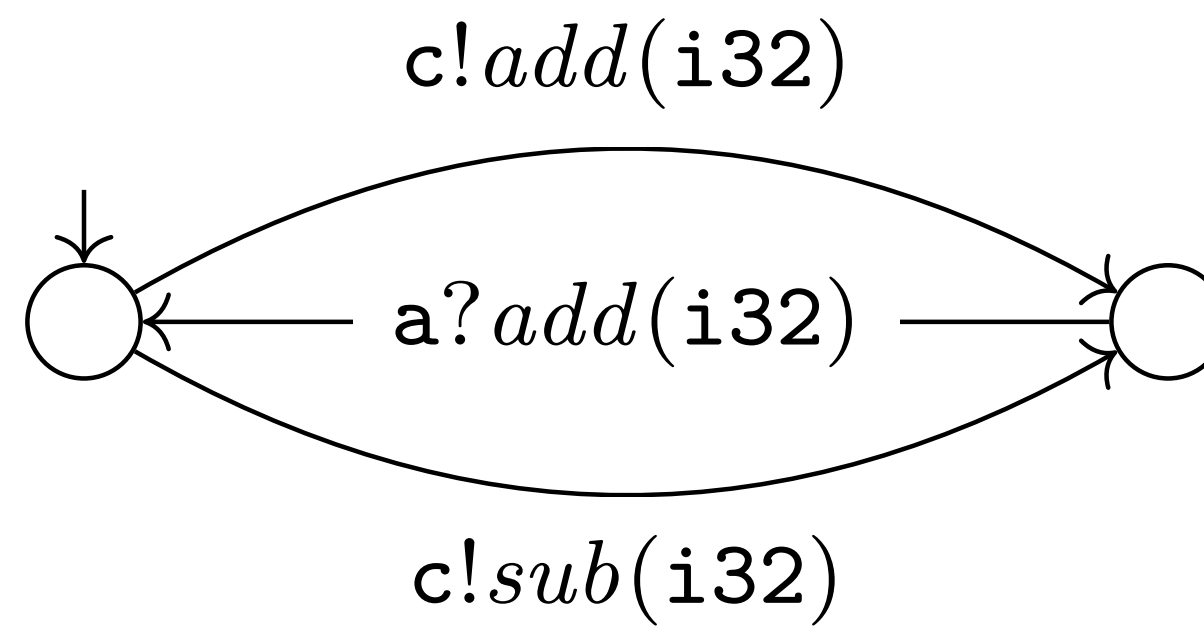
Safety

k-Multiparty Compatibility

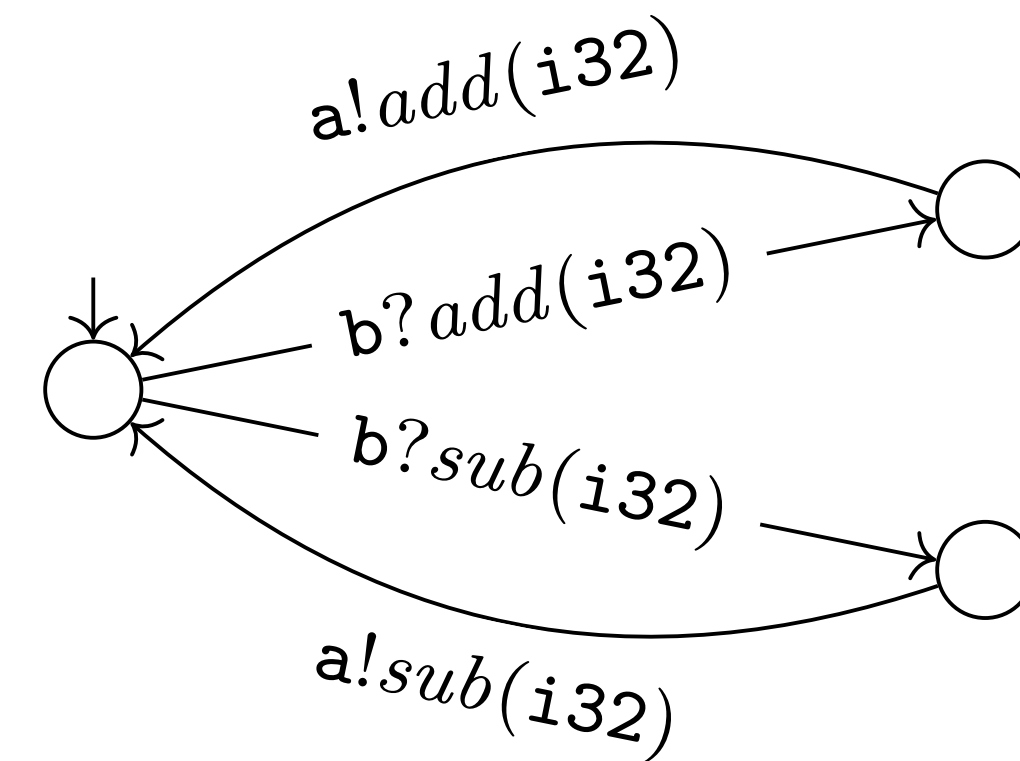
OPTIMISED A



OPTIMISED B



OPTIMISED C

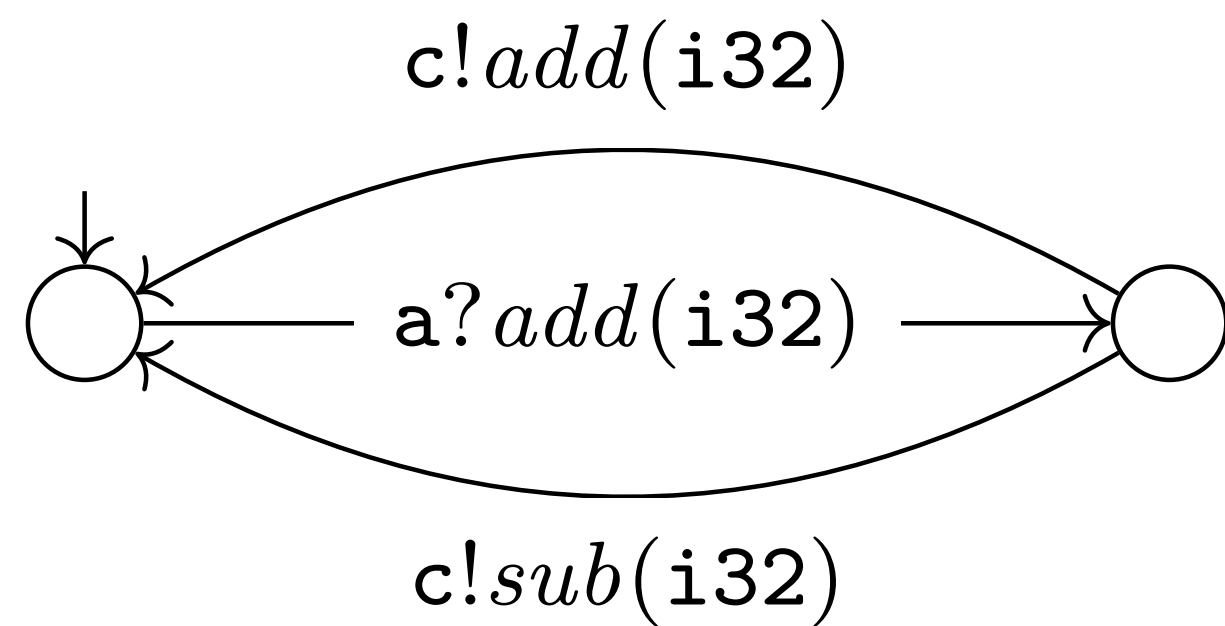


Safe?

Safety

Asynchronous Subtyping

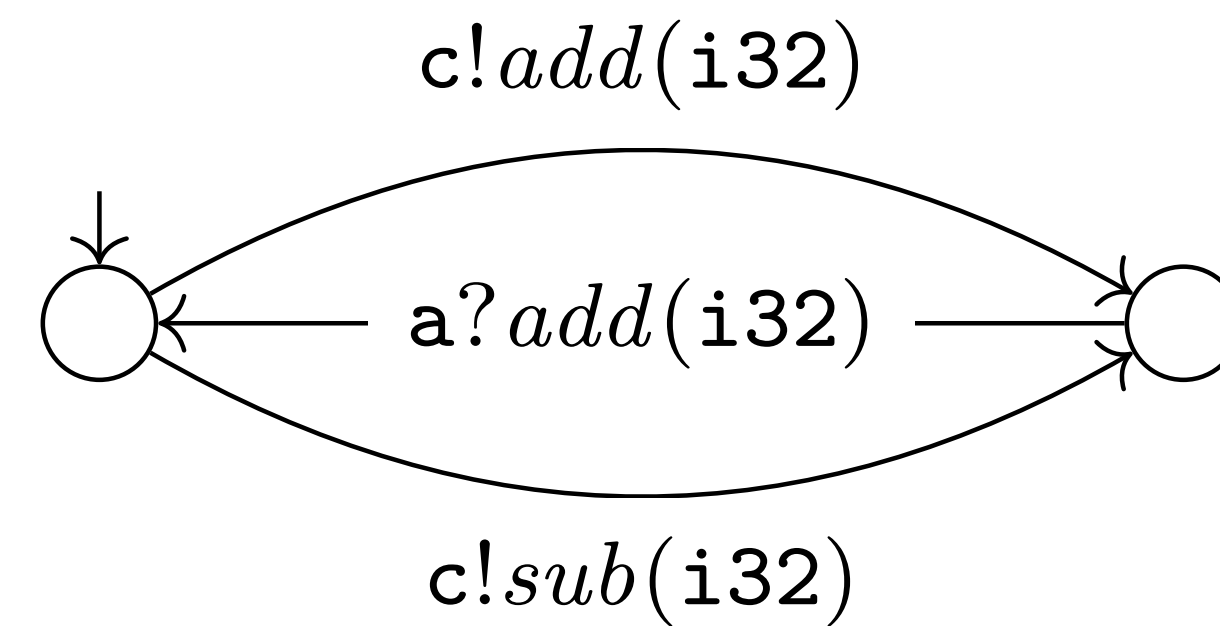
PROJECTED B



Safe?



OPTIMISED B



Session Decomposition

Challenge

Our asynchronous subtyping must subsume **synchronous** subtyping

$$\mathbb{T} = \& \mathbf{q} \begin{cases} ?\text{done}(\text{int}) \\ ?\text{fail}(\text{bool}) \end{cases} \leq \mathbf{q} ?\text{done}(\text{int}) \quad \mathbf{r} !\text{cont}(\text{int}) \leq \oplus \mathbf{r} \begin{cases} !\text{cont}(\text{int}) \\ !\text{stop}(\text{unit}) \end{cases} = \mathbb{T}'$$

...plus **multiparty asynchronous send/receive reorderings** are **hard to formalise!**

$$\mathbf{r} !\text{cont}(\text{int}).\mathbf{q} ?\text{done}(\text{nat}) \leq \mathbf{q} ?\text{done}(\text{nat}).\mathbf{r} !\text{cont}(\text{int})$$

$$\mathbf{r} ?\text{cont}(\text{int}).\mathbf{q} ?\text{done}(\text{int}) \leq \mathbf{q} ?\text{done}(\text{int}).\mathbf{r} ?\text{cont}(\text{int})$$

$$\mathbf{r} !\text{cont}(\text{int}).\mathbf{q} !\text{done}(\text{int}) \leq \mathbf{q} !\text{done}(\text{int}).\mathbf{r} !\text{cont}(\text{int})$$

Session Decomposition

Key Insight and Novelty

Session decomposition — decompose a session into subtrees

- single-input (SI), single-output (SO), single-input-single-output (SISO)

Then, we formalise message reorderings as a refinement for SISO trees only!



$$\llbracket T \rrbracket_{\text{SI}} = \{ \mathbf{q}?\text{done}(\text{int}), \mathbf{q}?\text{fail}(\text{bool}) \}$$

$$\llbracket T' \rrbracket_{\text{SO}} = \{ \mathbf{r}!\text{cont}(\text{int}), \mathbf{r}!\text{stop}(\text{unit}) \}$$

Asynchronous Subtyping

SISO Refinement

SISO trees are just **paths** — i.e. **sequences of inputs and outputs!**

$$\frac{}{\text{end} \lesssim \text{end}}$$

$$\frac{S' \leqslant S \quad W \lesssim W'}{\text{p?}\ell(S).W \lesssim \text{p?}\ell(S').W'}$$

$$\frac{S \leqslant S' \quad W \lesssim W'}{\text{p!}\ell(S).W \lesssim \text{p!}\ell(S').W'}$$

$$\frac{S' \leqslant S \quad W \lesssim \mathcal{A}^{(\mathbf{p})}.W' \quad \text{act}(W) = \text{act}(\mathcal{A}^{(\mathbf{p})}.W')}{\text{p?}\ell(S).W \lesssim \mathcal{A}^{(\mathbf{p})}.\text{p?}\ell(S').W'}$$

$$\frac{S \leqslant S' \quad W \lesssim \mathcal{B}^{(\mathbf{p})}.W' \quad \text{act}(W) = \text{act}(\mathcal{B}^{(\mathbf{p})}.W')}{\text{p!}\ell(S).W \lesssim \mathcal{B}^{(\mathbf{p})}.\text{p!}\ell(S').W'}$$

$$\mathcal{A}^{(\mathbf{p})} ::= \mathbf{q?}\ell(S) \parallel \mathbf{q?}\ell(S).\mathcal{A}^{(\mathbf{p})} \quad \mathcal{B}^{(\mathbf{p})} ::= \mathbf{r?}\ell(S) \parallel \mathbf{q!}\ell(S) \parallel \mathbf{r?}\ell(S).\mathcal{B}^{(\mathbf{p})} \parallel \mathbf{q!}\ell(S).\mathcal{B}^{(\mathbf{p})} \quad (\mathbf{q} \neq \mathbf{p})$$

$$\frac{\forall U' \in [\mathbf{T}']_{\text{so}} \quad \forall V \in [\mathbf{T}]_{\text{si}} \quad \exists W' \in [\mathbf{U}']_{\text{si}} \quad \exists W \in [\mathbf{V}]_{\text{so}} \quad W' \lesssim W}{\mathbf{T}' \leqslant \mathbf{T}}$$

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

| | | | |
|-------------|-------|---------------|-----------------|
| π, ρ | $::=$ | ϵ | 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 π and π' , $\langle \pi \sqcup \pi' \rangle$ 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 π and π' , the time complexity of reducing $\langle \pi \sqcup \pi' \rangle$ is $O(\min(|\pi|, |\pi'|))$.*

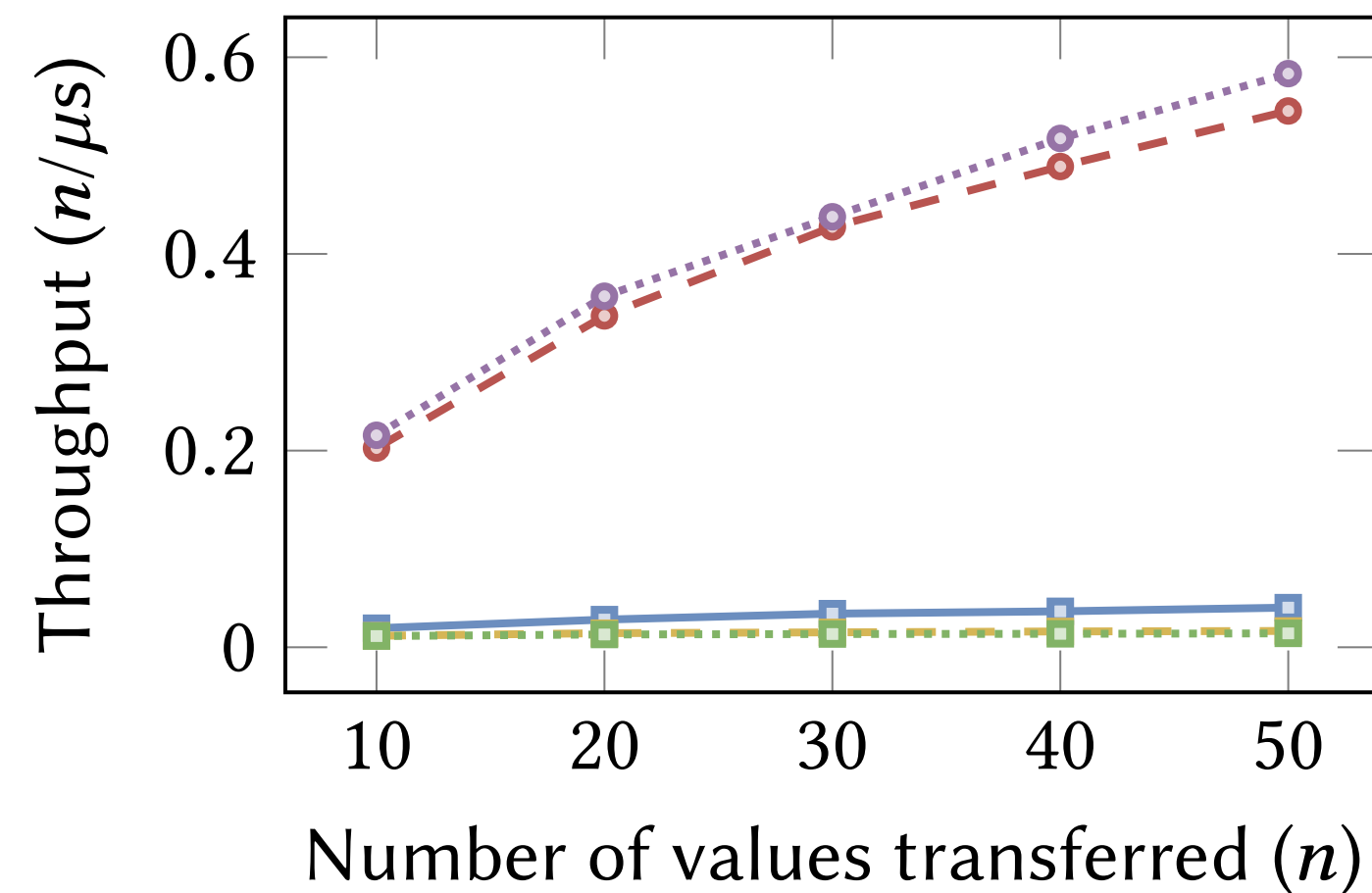
Theorem 7 (Complexity). *Consider T and T' as (possibly infinite) trees $\mathcal{T}(T)$ and $\mathcal{T}(T')$ with asymptotic branching factors b and b' respectively. Our algorithm has time complexity $O(n \min(b, b')^n)$ and space complexity $O(n \min(b, b'))$ in the worst case to determine if $T \leq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

Evaluation

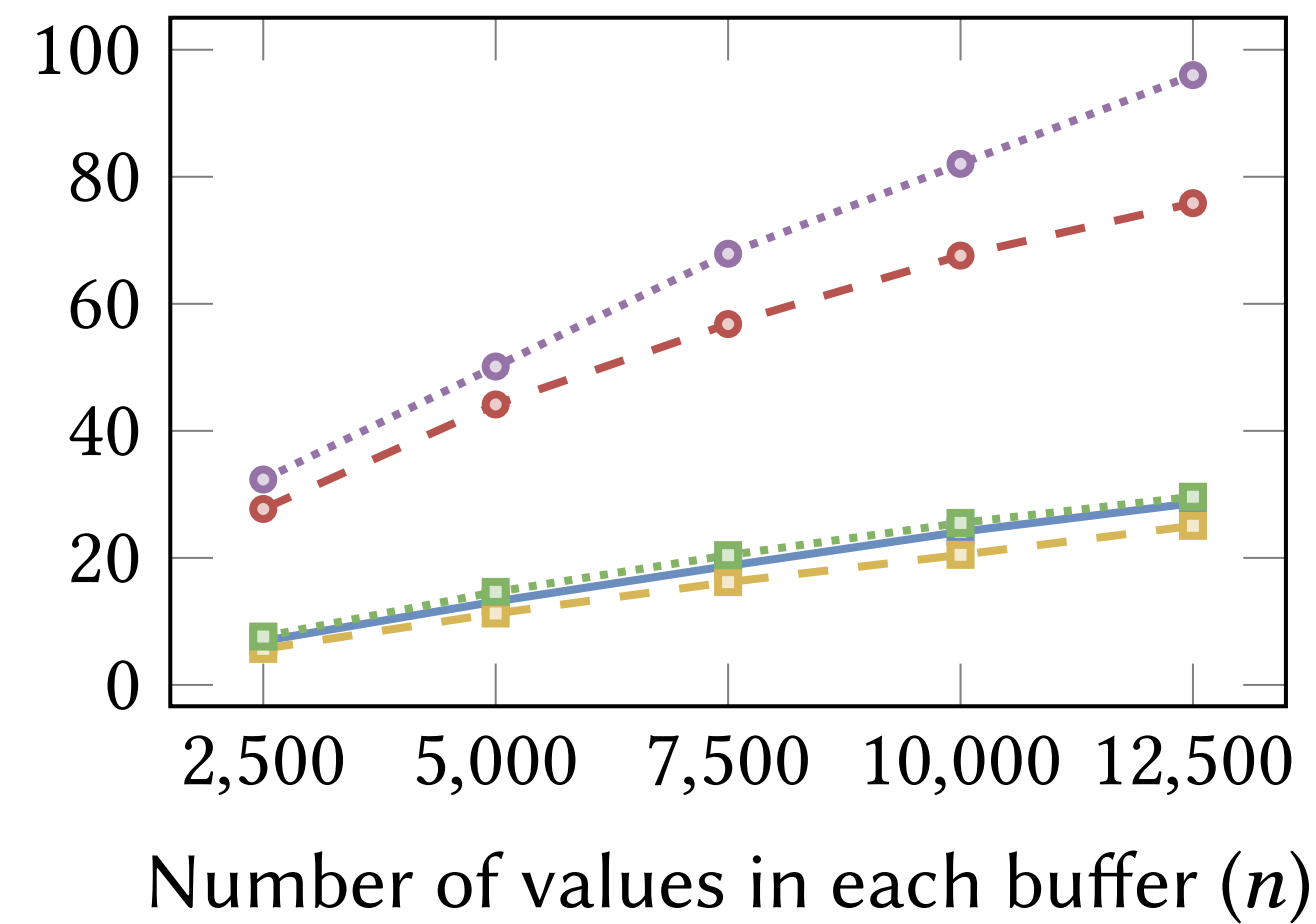
Rust Framework Benchmarks

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

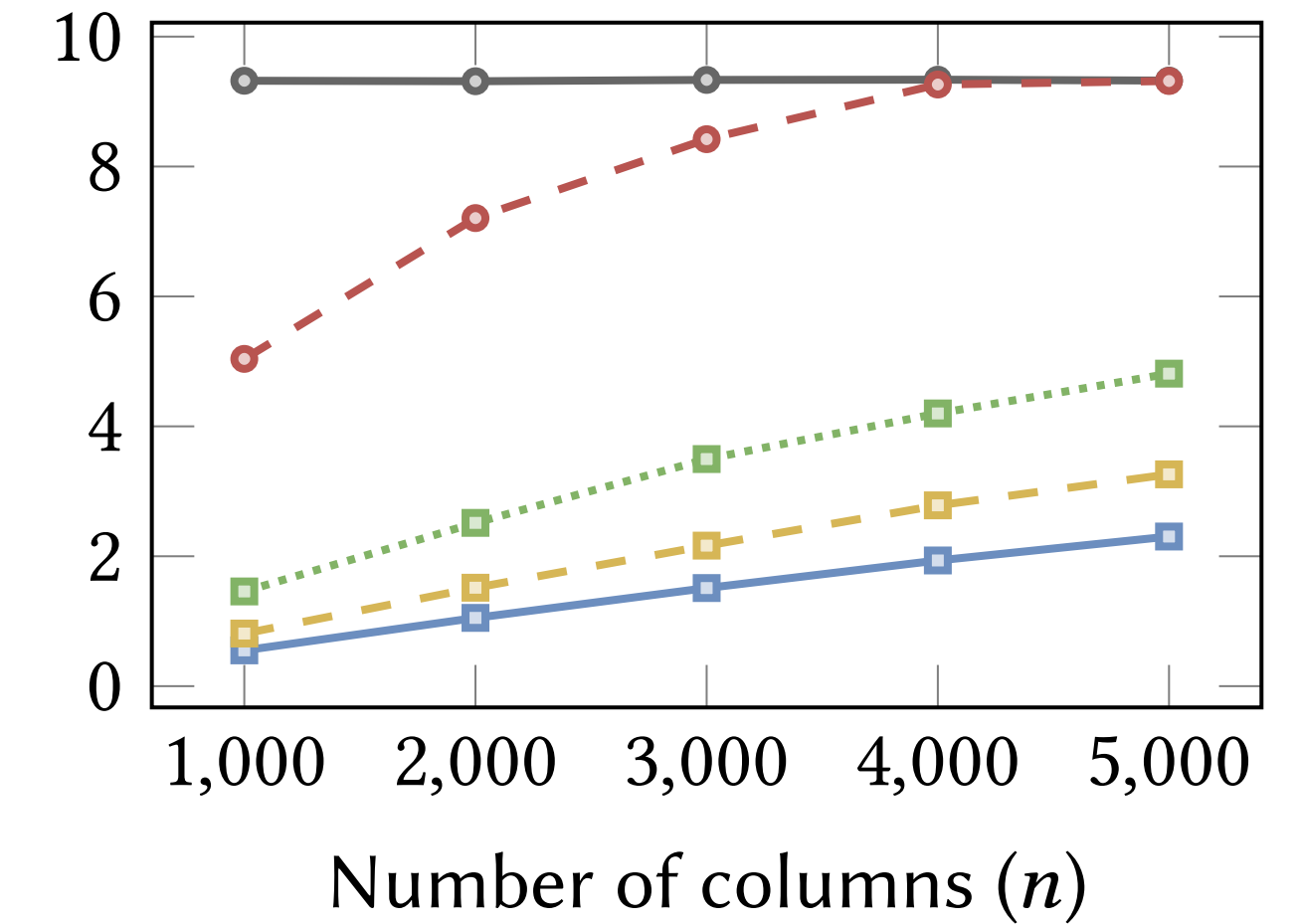
Stream



Double Buffering



FFT

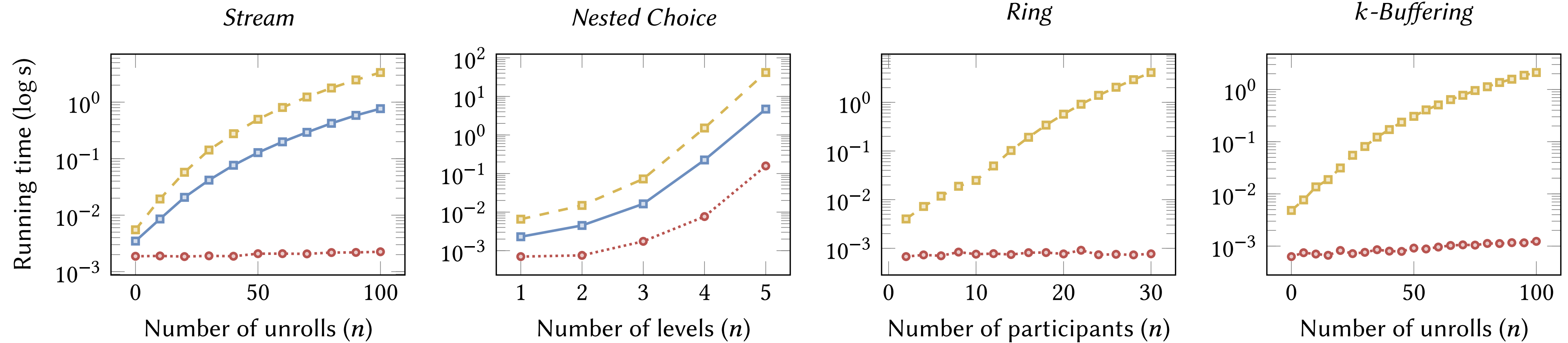


16-core AMD Opteron™ 6200 Series CPU @ 2.6GHz with hyperthreading, 128GB of RAM, Ubuntu 18.04.5 LTS and Rust Nightly 2021-07-06. We use version 0.3.5 of the Criterion.rs library and a multi-threaded asynchronous runtime from version 1.11.0 of the Tokio library.

Evaluation

Asynchronous Reordering Benchmarks

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



Evaluation

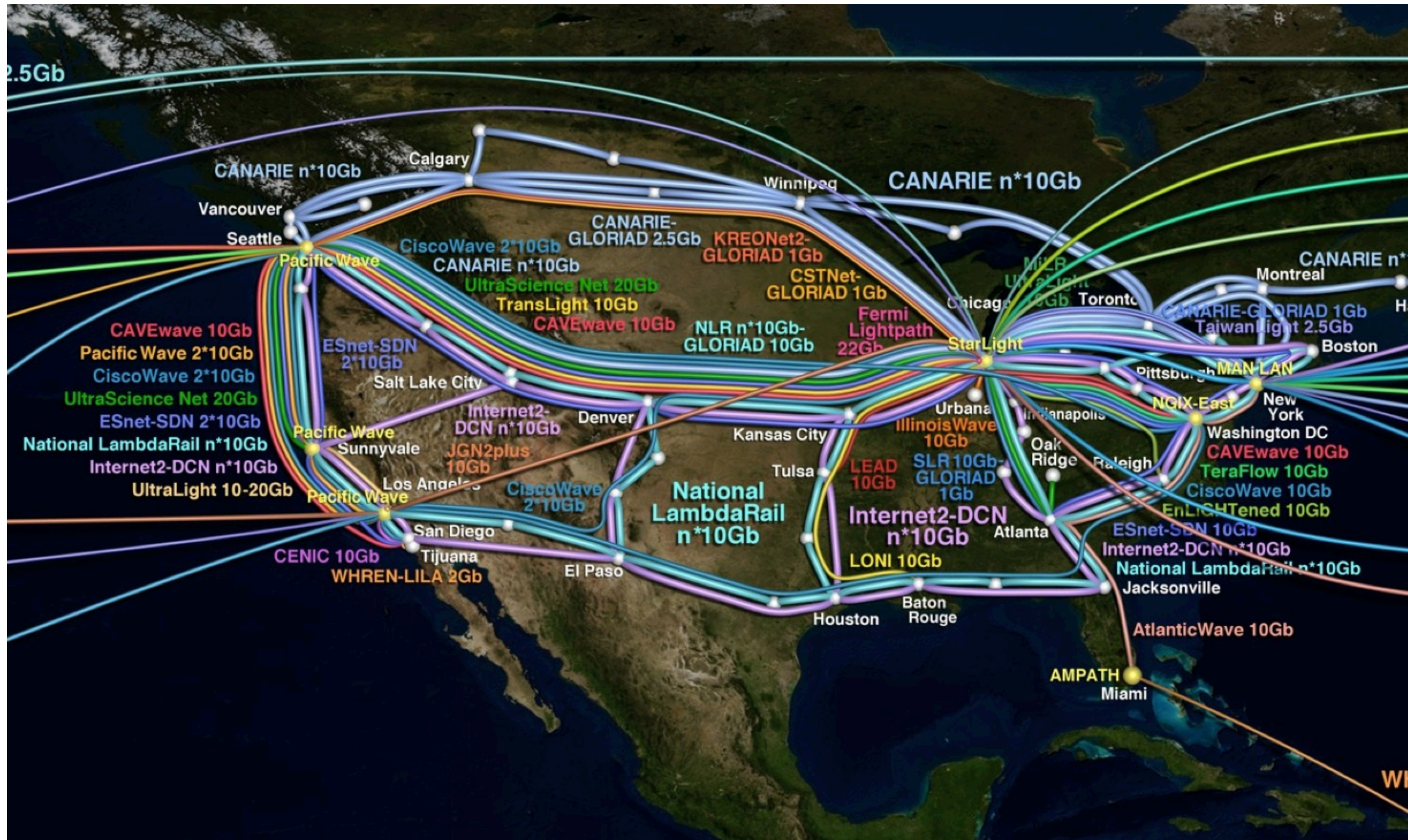
Expressiveness

| 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

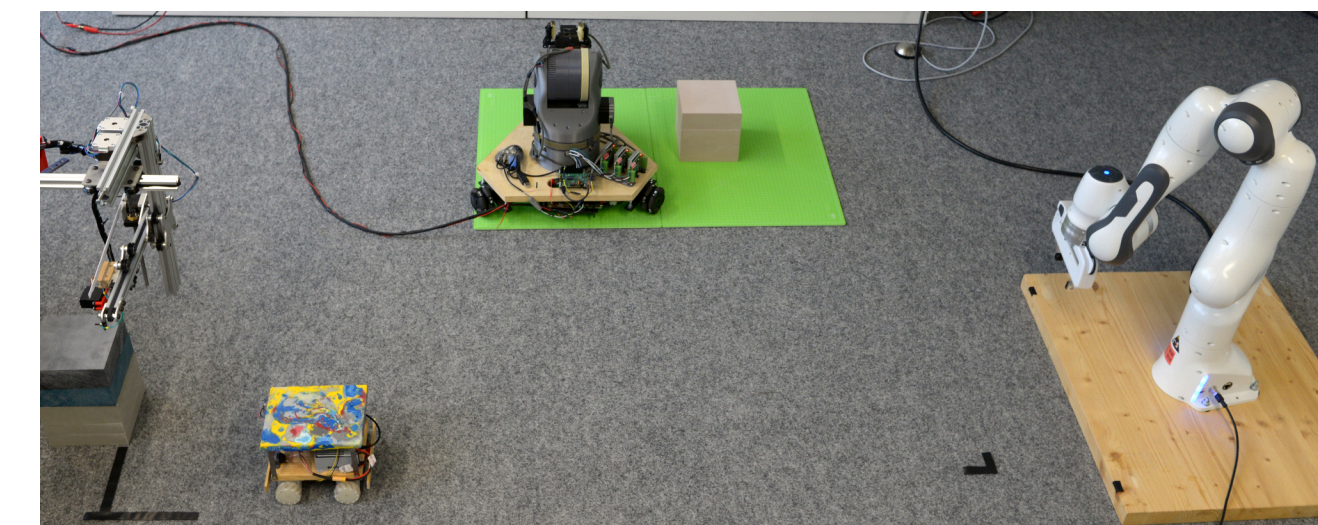
More Applications on Multiparty Session Types



Ocean Observatories Initiative



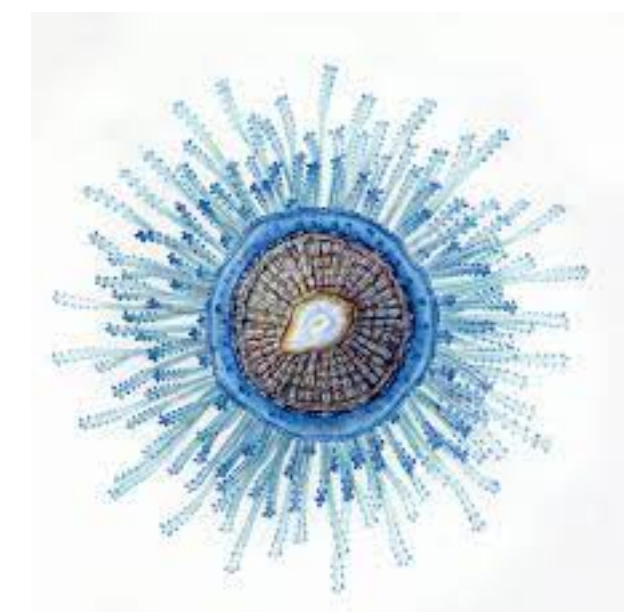
ECOOP'19, OOPSLA'20



PLDI'21






Zoid



Conclusion

Multiparty Session Types and Communicating Automata

- Multiparty session types and communicating automata
 - Invited paper in the FCT '21 proceedings
 -  Scribble <https://github.com/scribble>
 -  <https://github.com/nuscr>
- Applications of multiparty session types using communicating automata
 -  <https://github.com/zakcutner/rumpsteak>