

# 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

|                        |  |                    |
|------------------------|--|--------------------|
| <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]

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?

---

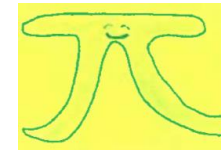
Significant academic and industry interests via fundamental breakthroughs

Milner,  
Honda, NY



Binary Session Types

ESOP'98



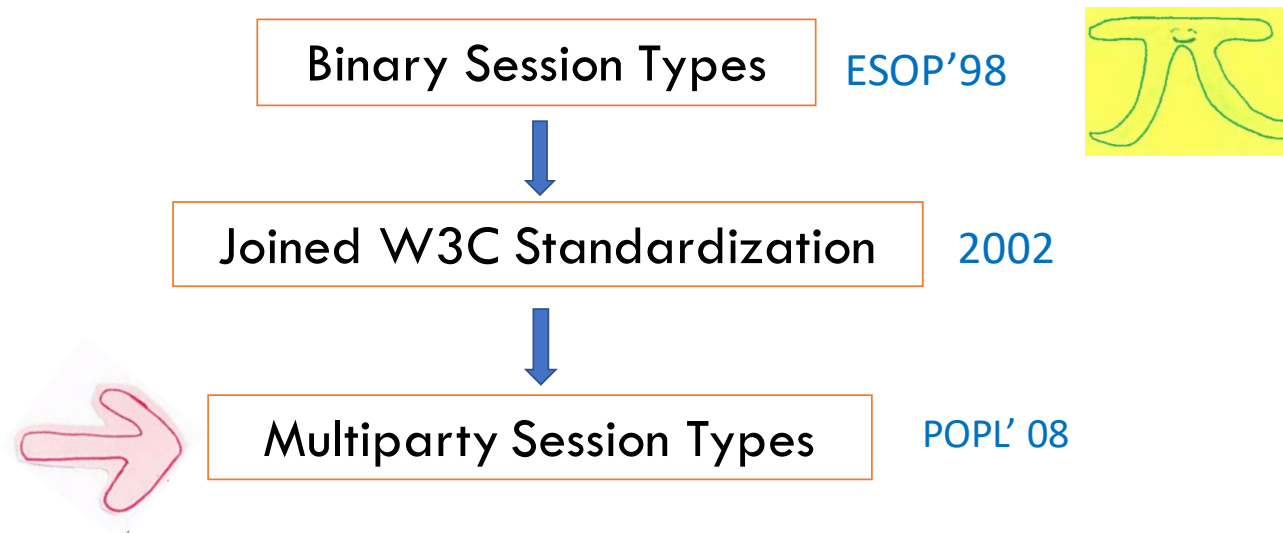
Joined W3C Standardization

2002

# Why Session Types, Why Now?

---

Significant academic and industry interests via fundamental breakthroughs

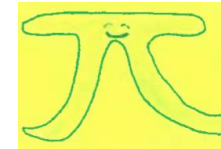


# Why Session Types, Why Now?

Significant academic and industry interests via fundamental breakthroughs

Binary Session Types

ESOP'98



Joined W3C Standardization

2002



Multiparty Session Types

POPL'08

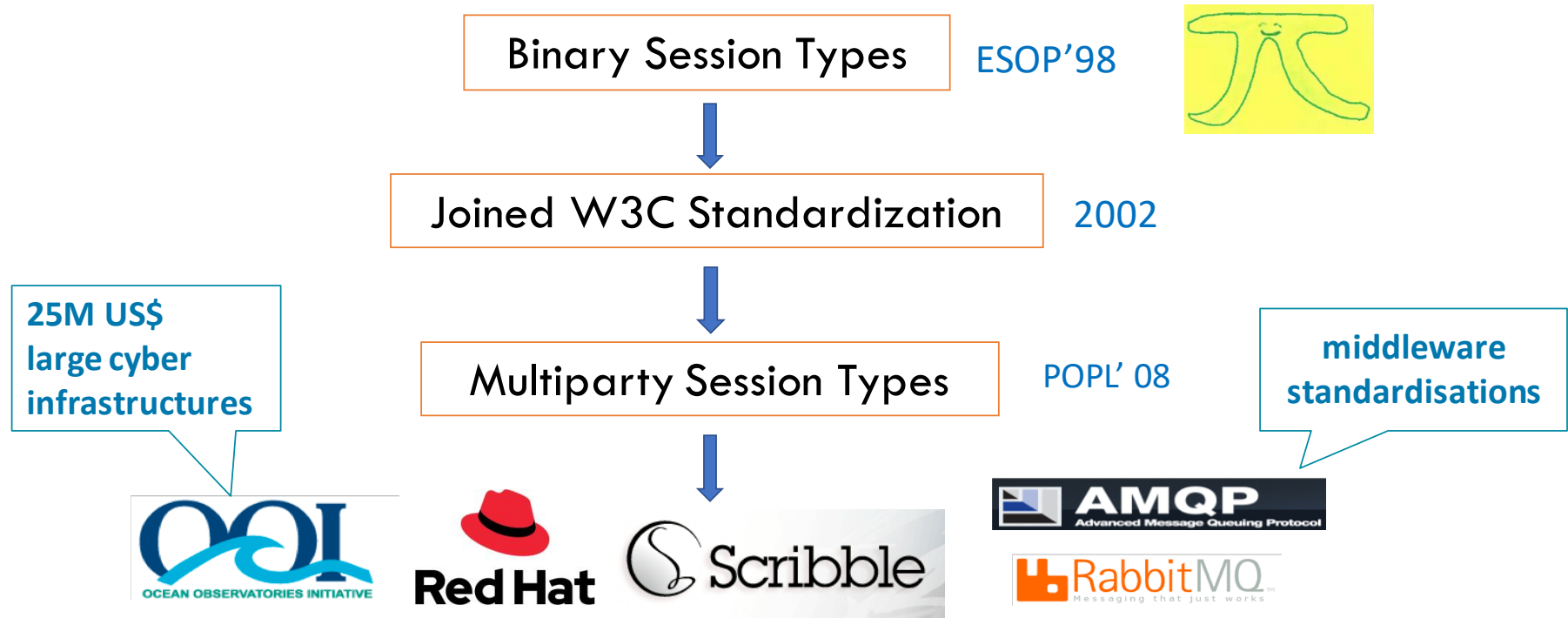


largest open source  
company in the world



# Why Session Types, Why Now?

Significant academic and industry interests via fundamental breakthroughs



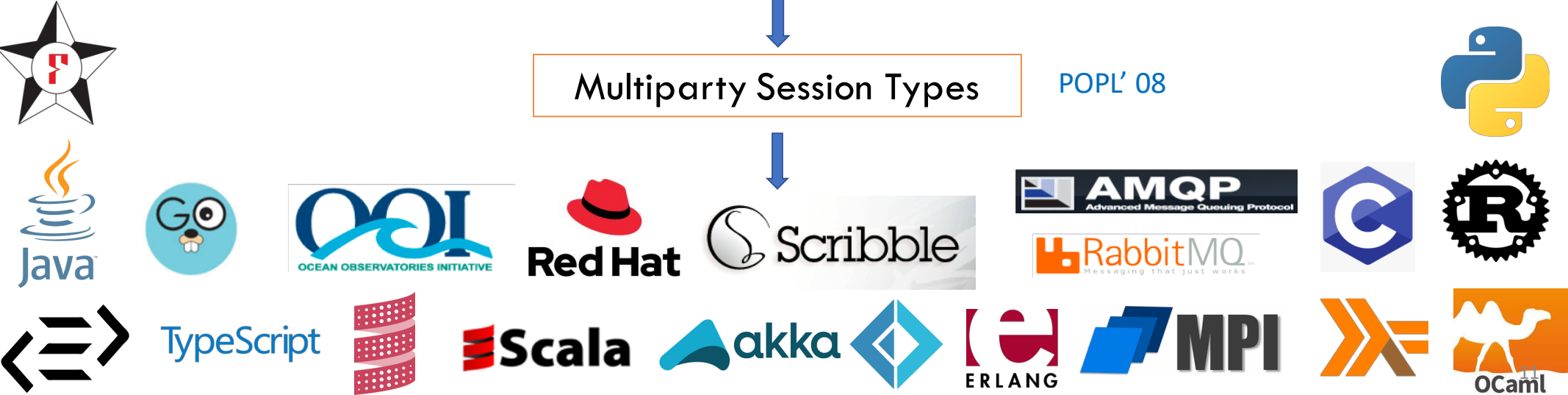
# Why Session Types, Why Now?

Significant academic and industry interests via fundamental breakthroughs

Binary Session Types ESOP'98 

Joined W3C Standardization 2002

Multiparty Session Types POPL'08



A collection of logos for various programming languages, frameworks, and organizations associated with session types. The logos include:

- Star with a red question mark
- Java logo
- Go logo
- Ocean Observatories Initiative (OOI) logo
- Red Hat logo
- Scribble logo
- AMQP (Advanced Message Queuing Protocol) logo
- RabbitMQ logo
- Python logo
- R logo
- TypeScript logo
- Scala logo
- akka logo
- ERLANG logo
- MPI logo
- OCaml logo

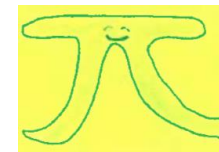
# Why Session Types, Why Now?

Significant academic and industry interests via fundamental breakthroughs

ETAPS Test Time Award 2019

Binary Session Types

ESOP'98



Joined W3C Standardization

2002



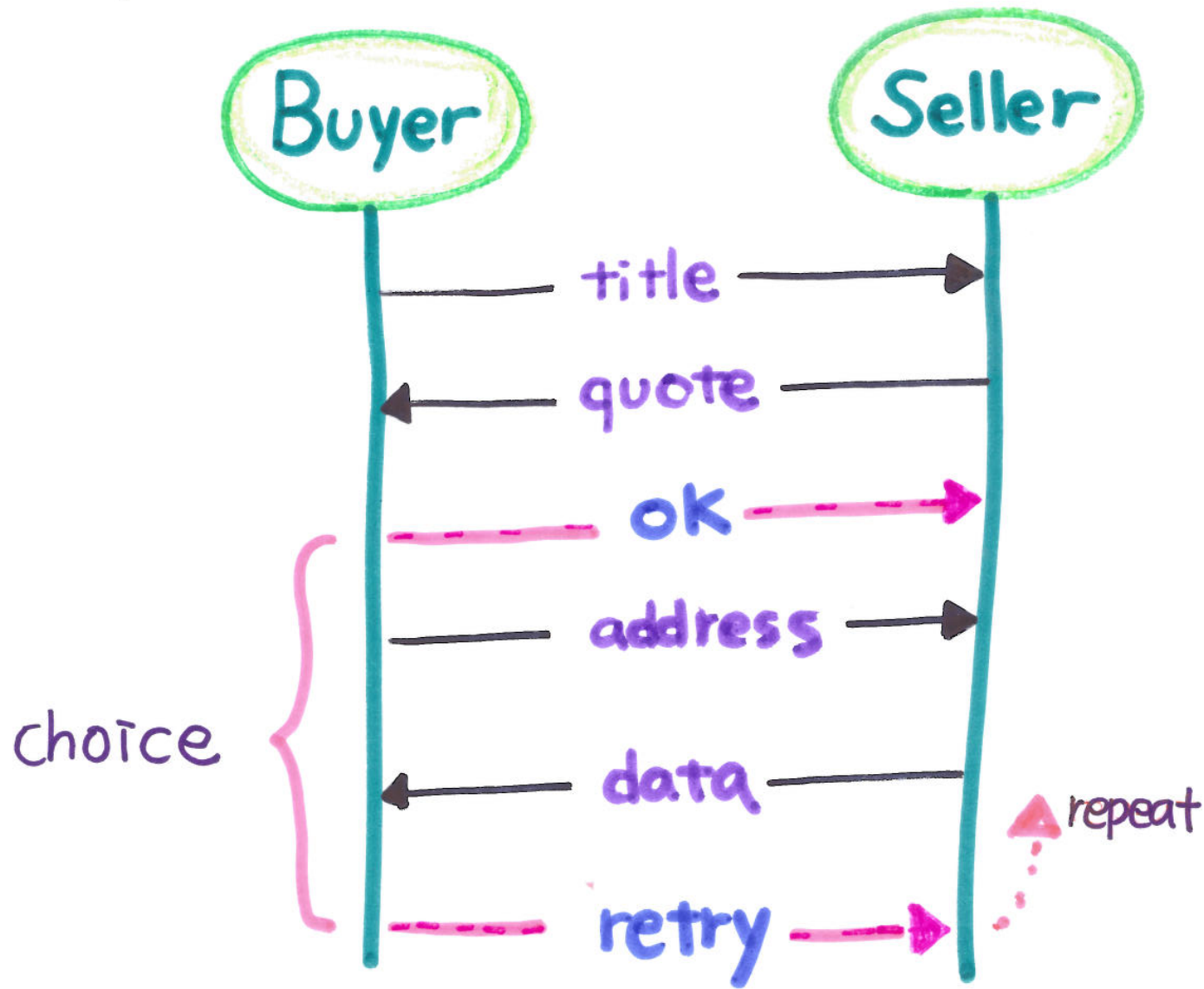
Multiparty Session Types

POPL' 08

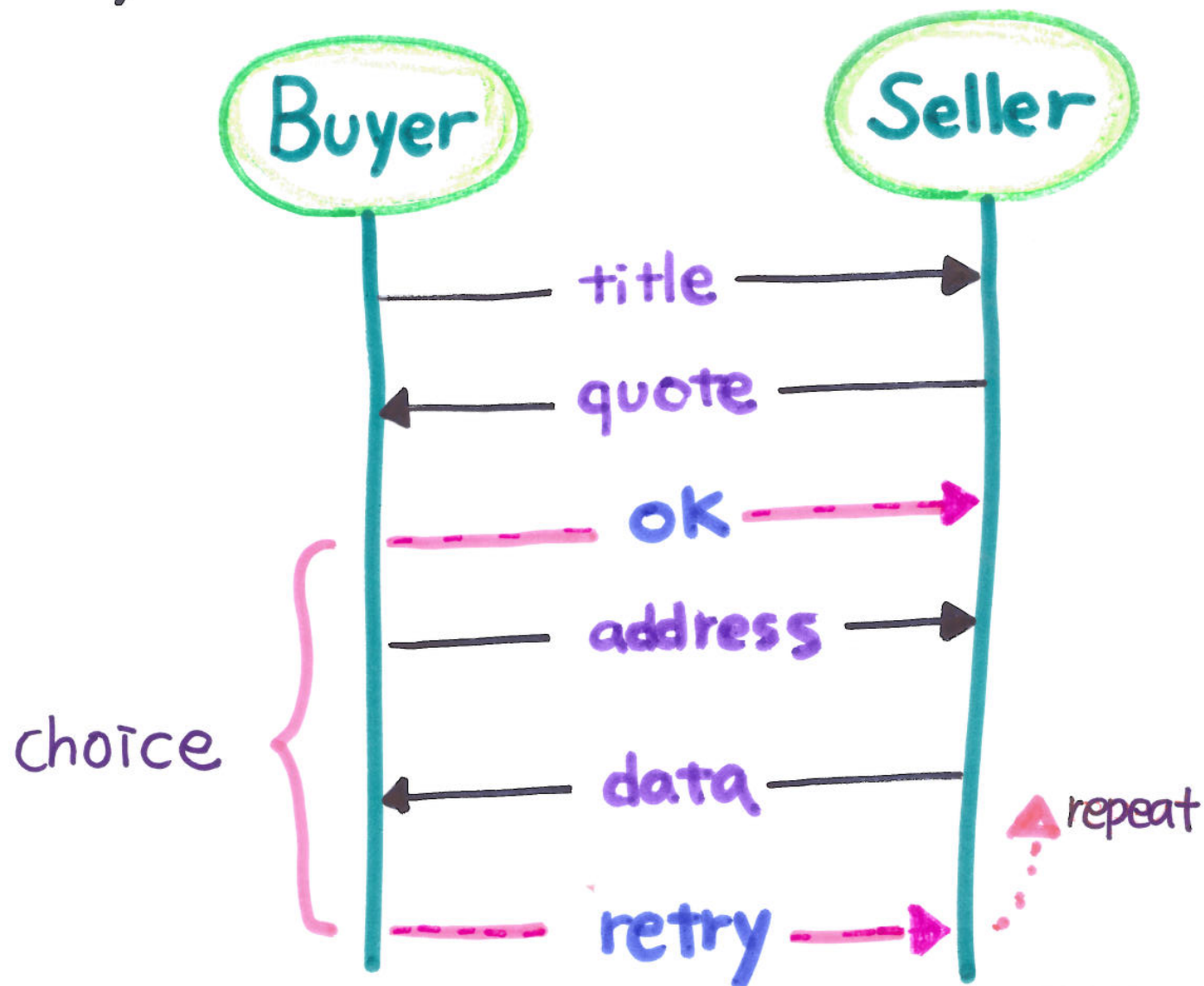
POPL Influential Paper Award 2018



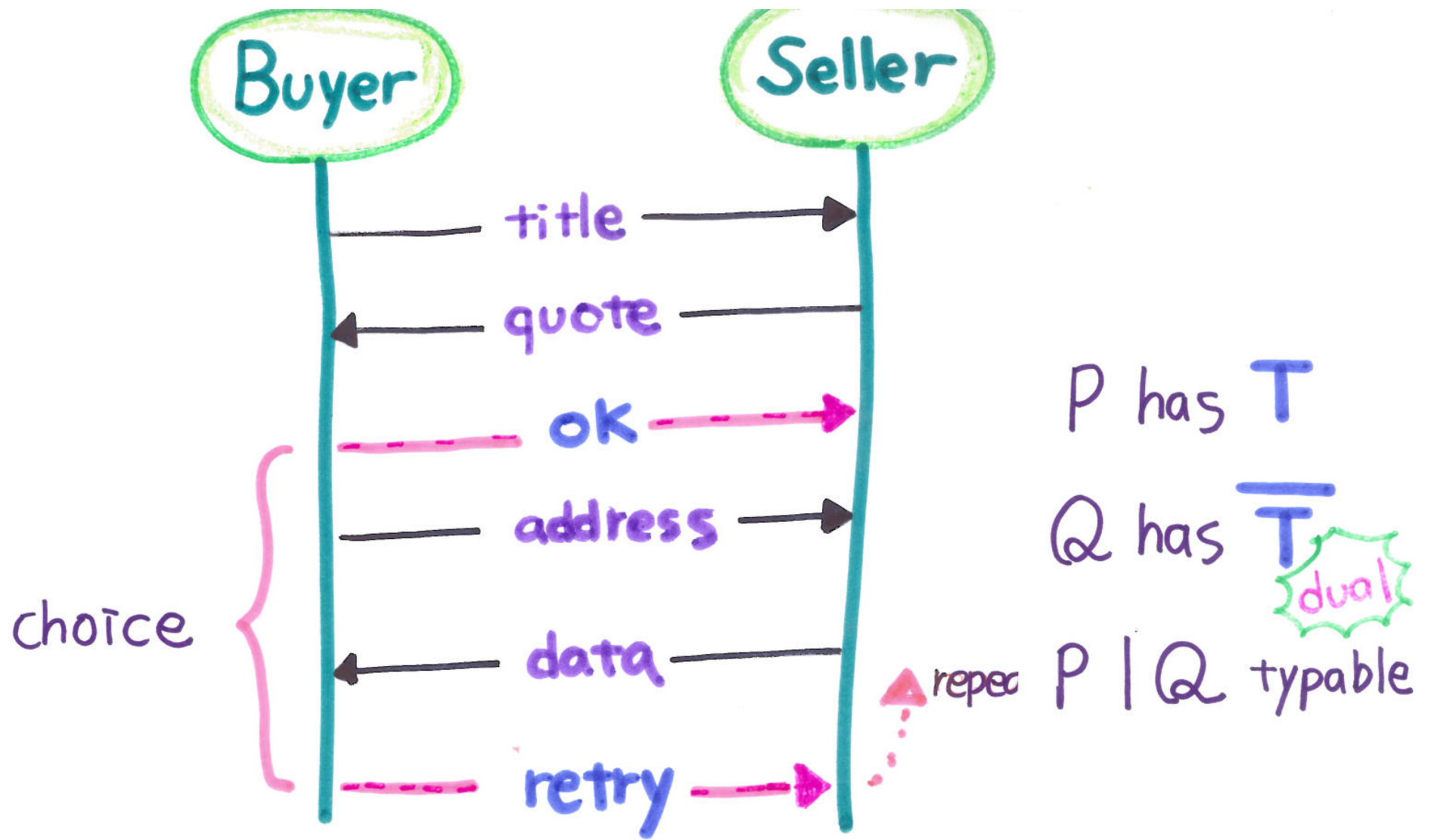
# Binary Session Types: Buyer - Seller Protocol



# Binary Session Types: Buyer - Seller Protocol



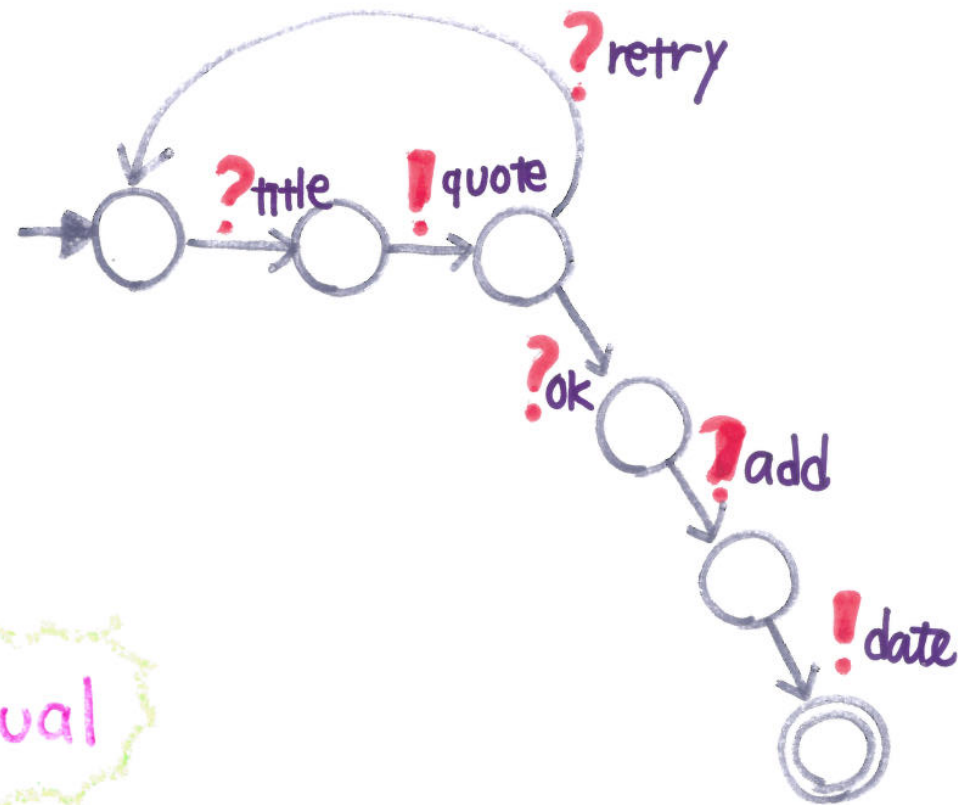
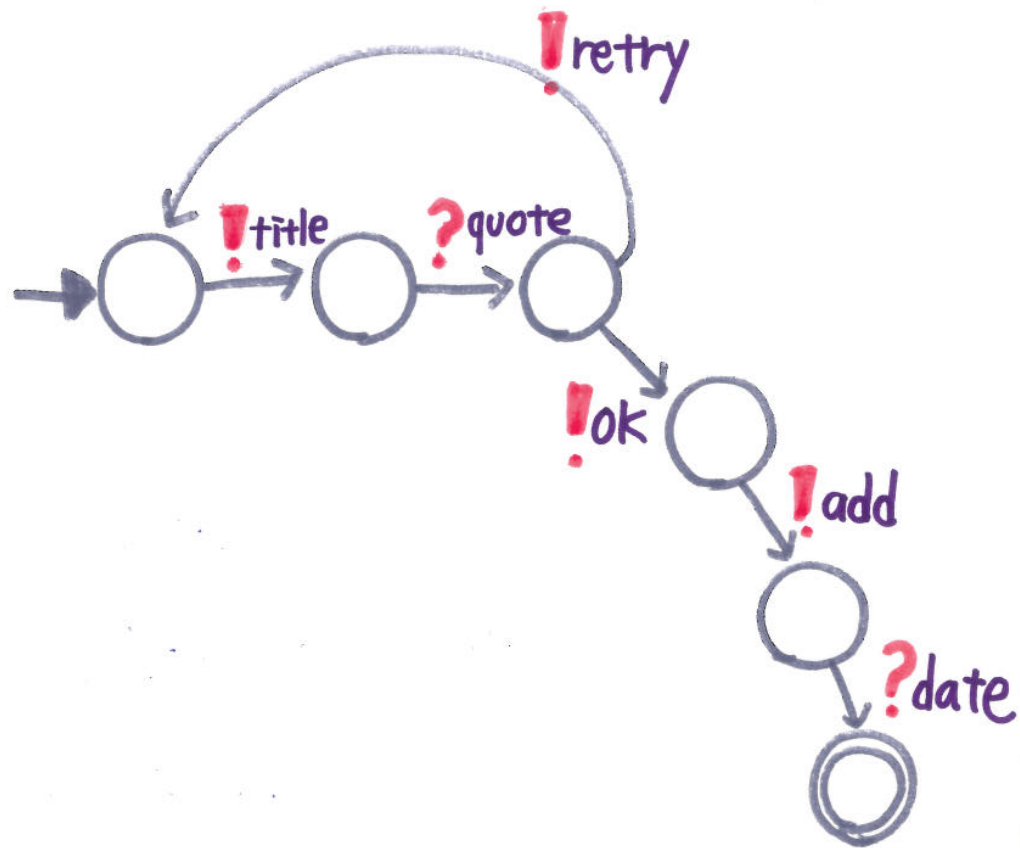
nt! Title ; ? Quote ; ! { ok: ! Add ; ? Date, retry: t }



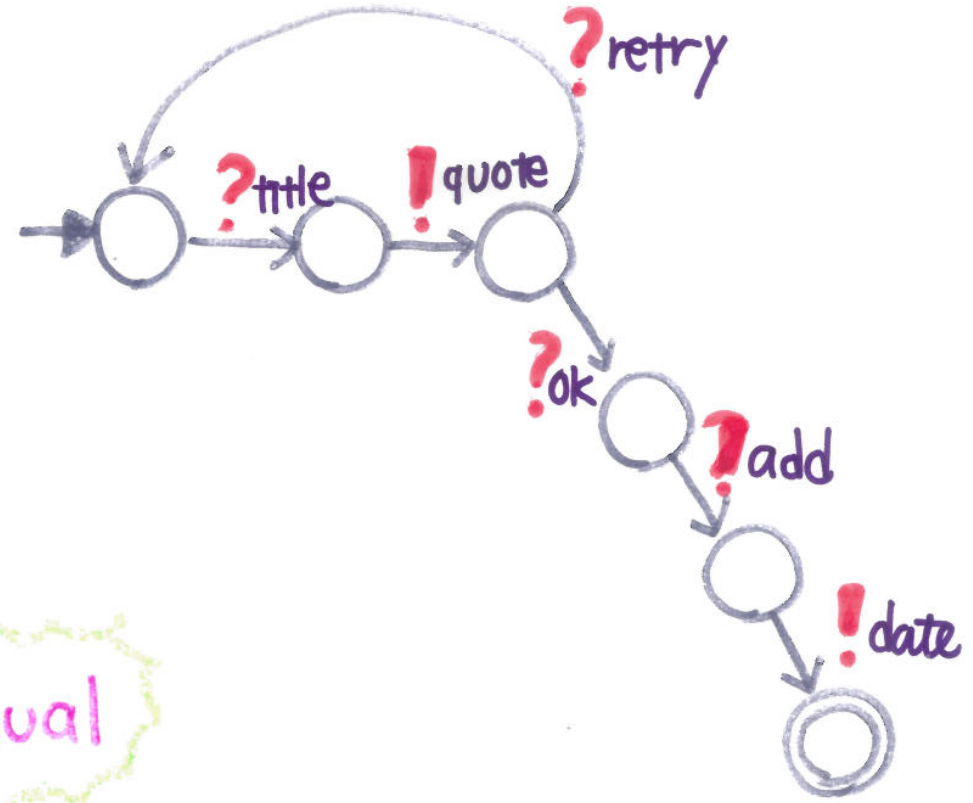
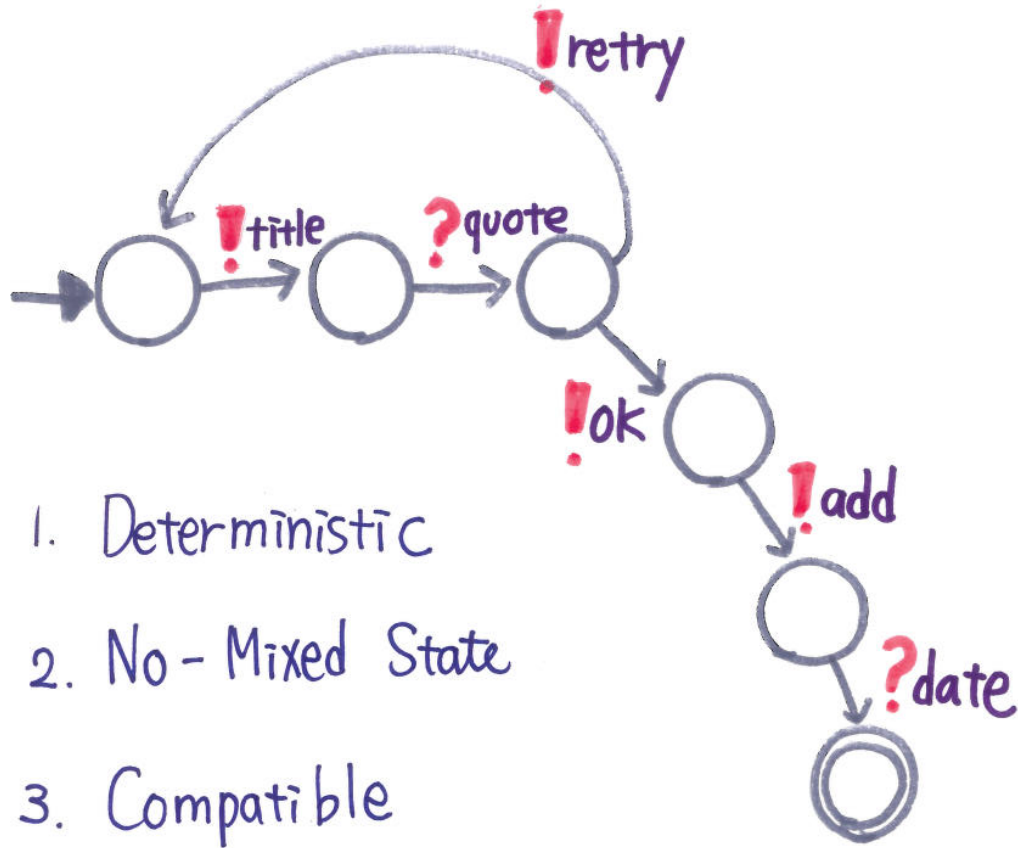
nt! Title ; ? Quote ; ! { ok: ! Add ; ? Date, retry: t }

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

# Communicating Automata [1980s]

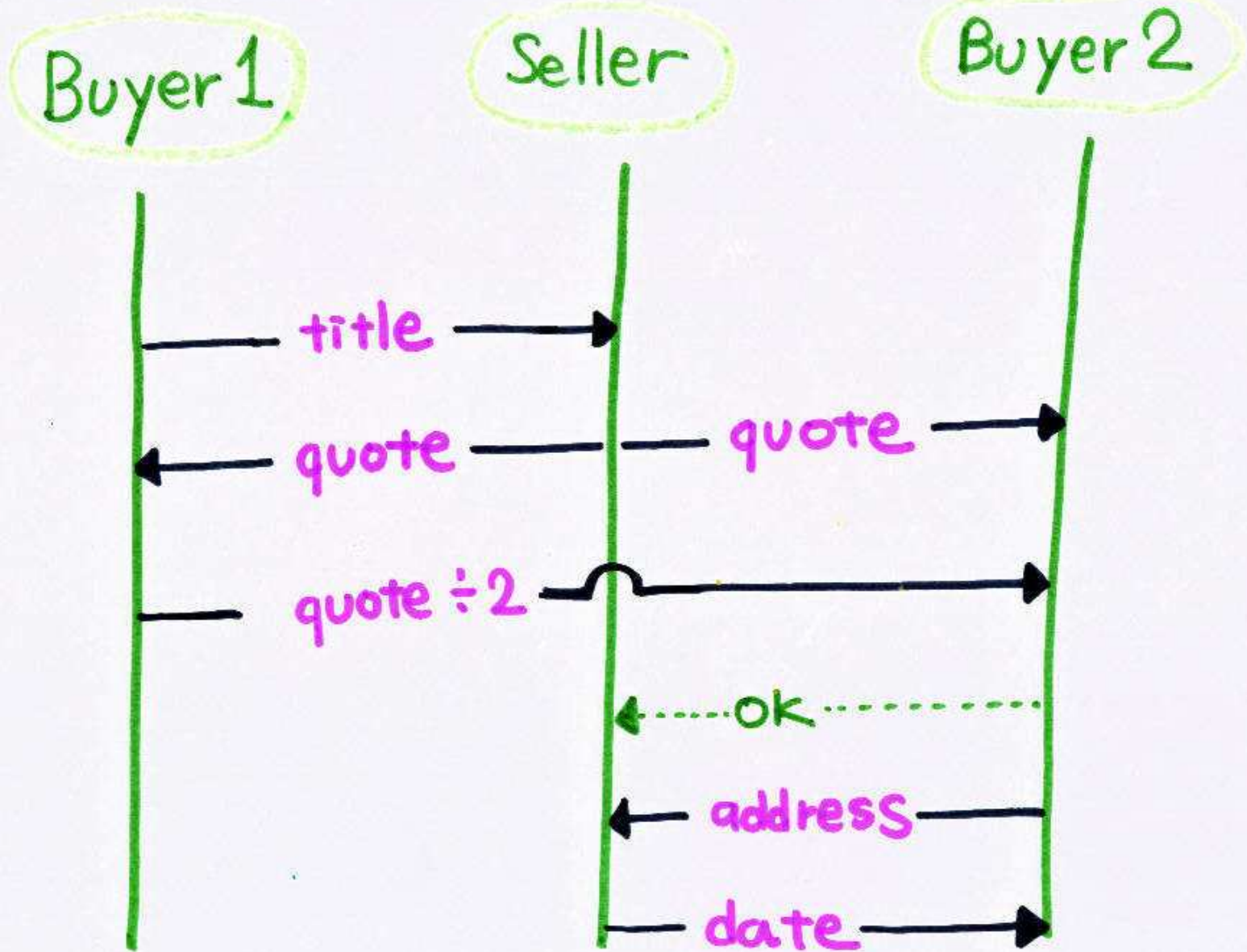


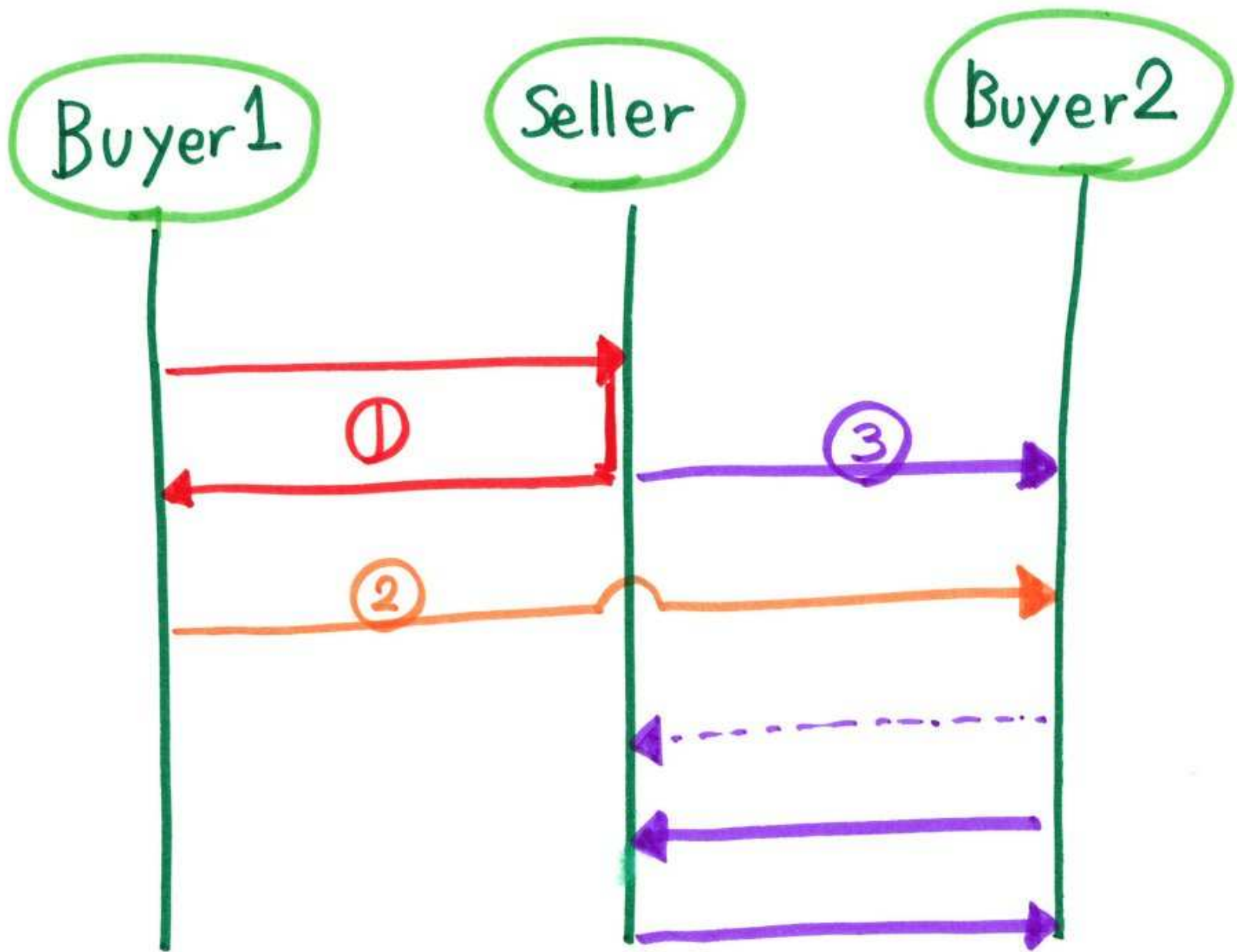
dual

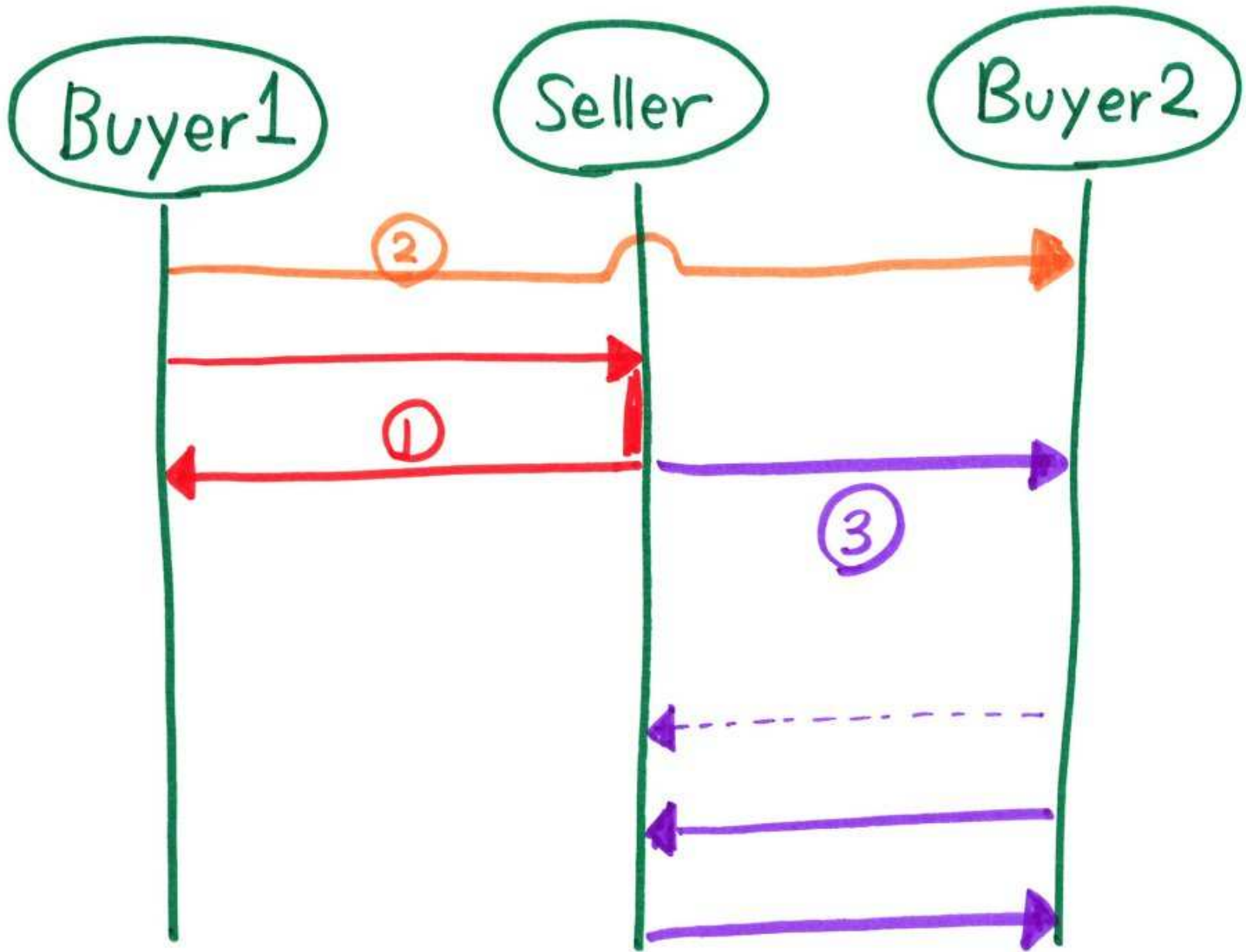


**[Gouda et al 1986]** Two compatible machines without mixed states which are deterministic satisfy deadlock-freedom.

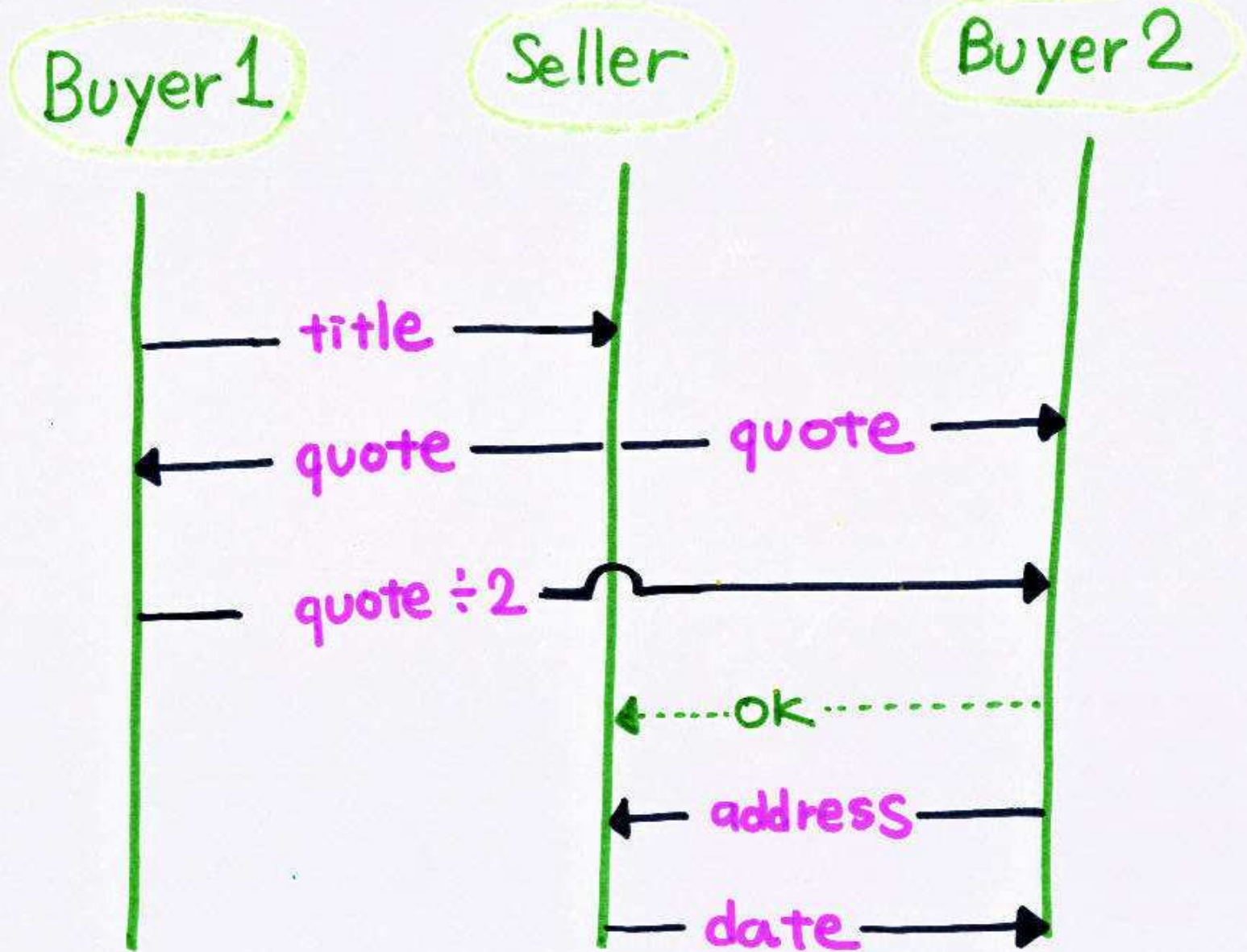
# Multiparty Session Types







# Multiparty Session Types



Alice

Bob

Carol

CA?c ; AB!a

AB?a ; BC!b

BC?b ; CA!c

dual

dual

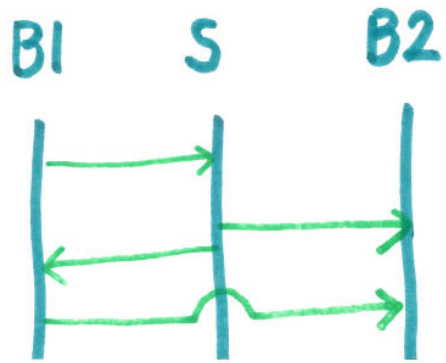
dual

3 dual pairs

If you use  
binary Session  
Types ...

Deadlock!

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



Ⓞ G

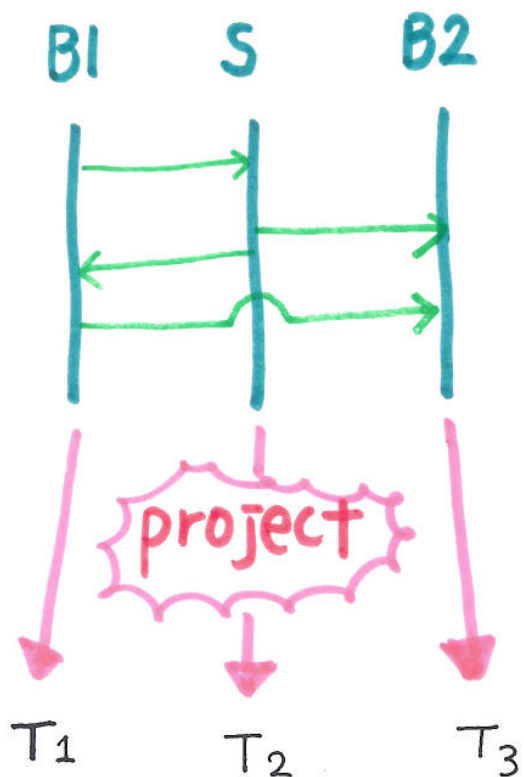
$BI \rightarrow S$  Int.

$S \rightarrow B2$  Char

**STEP 1**

Write Global Type

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



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

(T)  $B1?Int. B2!Char$

STEP 1

Write Global Type

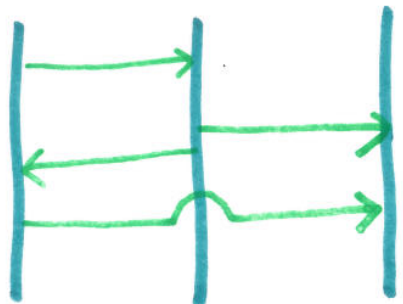
STEP 2

Project to Local Types

# Multiparty Session Types

[Honda, Yoshida, Carbone 2008]

B1    S    B2



(G)

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

$S \rightarrow B2 \text{ Char}$

STEP 1

Write Global Type

(T)

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

STEP 2

Project to Local Type

T<sub>1</sub>    T<sub>2</sub>    T<sub>3</sub>



P<sub>1</sub>

P<sub>2</sub>

P<sub>3</sub>



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

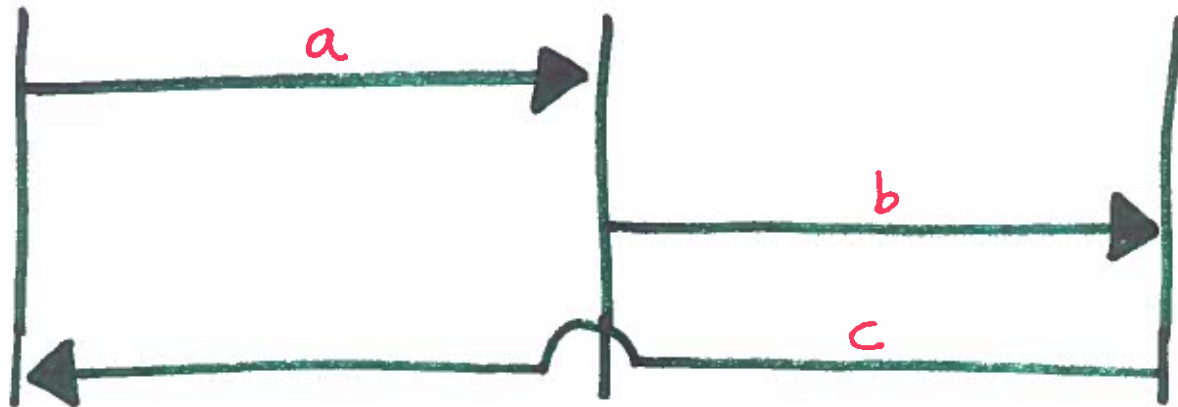
STEP 3

- Static Check
- Generate Code
- Run-time check

Alice

Bob

Carol



Global Type



Alice  $AB!a; CA?c$

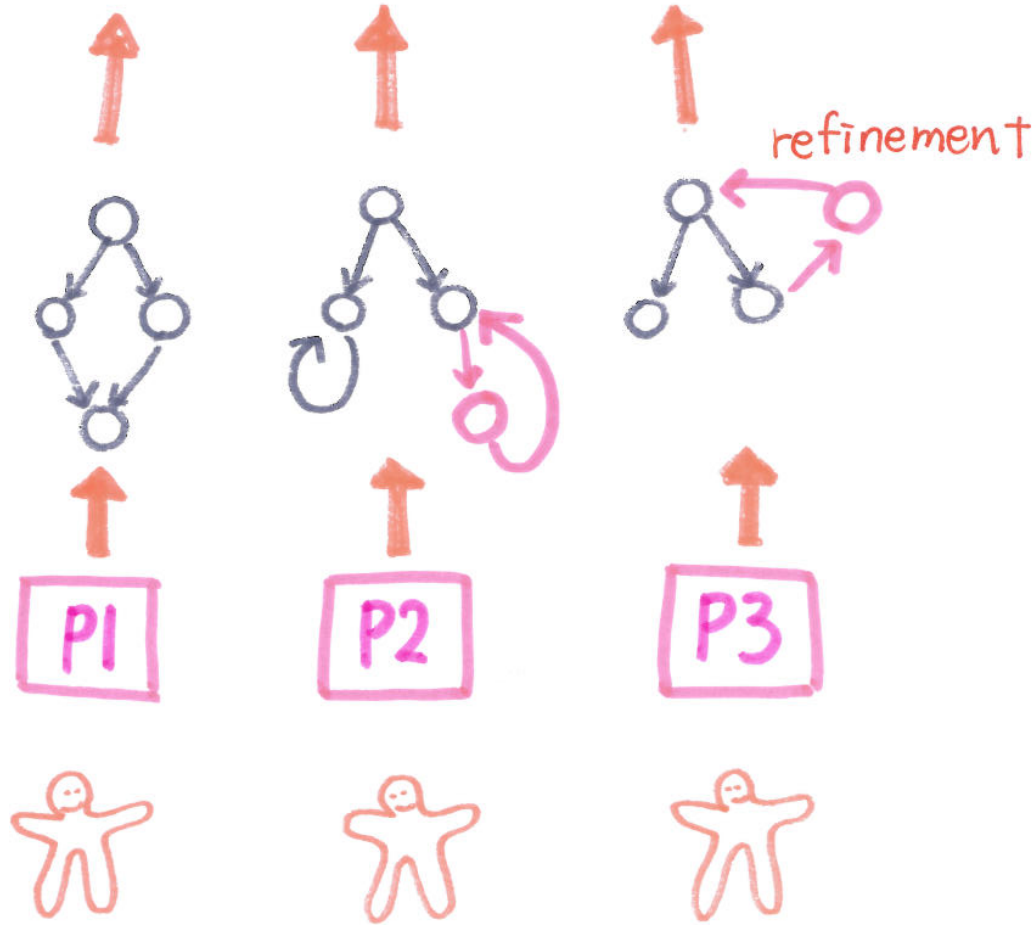
Bob  $AB?a; BC!b$

Carol  $BC?b; CA!c;$

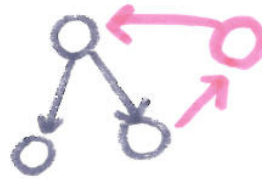
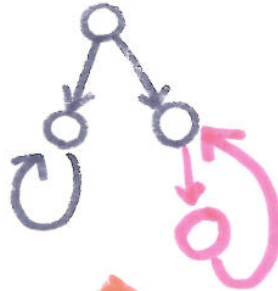
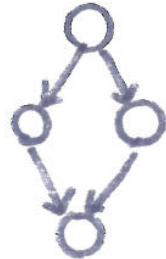
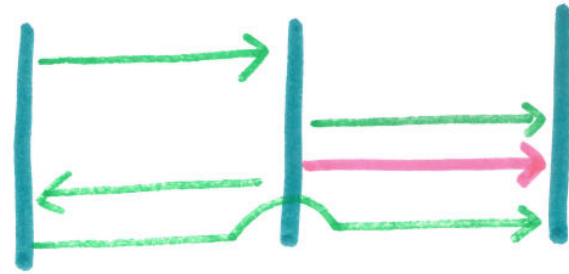
No Deadlock

LOCAL TYPES

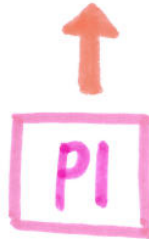




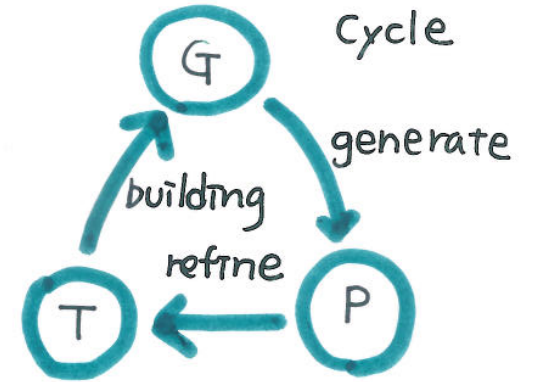
BI S B2



refinement



Software Development Cycle



- Optimisation
- refinement
- inference
- Testing

# Global Types

# Local Types

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

| mt.  $G$

| t

| end

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

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

| mt.  $T$

| t

| end

$P, P', \dots$  Participant

$a, b, c$   $\Sigma$  Alphabet

Projection from  $\Gamma$  onto  $\mathfrak{q}$   $\Gamma \upharpoonright \mathfrak{q}$

$$\left( p \rightarrow p' : \{ a_j, \Gamma_j \}_{j \in J} \right) \upharpoonright \mathfrak{q}$$

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

**Projection** from  $\Gamma$  onto  $\varrho$       $\Gamma \upharpoonright \varrho$

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

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

**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

$$l ::= pq!a \mid pq?a$$

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

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

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

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

$W_{pq}$   
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  $T_q \xrightarrow{pq?l} T_q' \Rightarrow$

$$(\dots T_q \dots ; \dots l \cdot W_{pq} \dots) \xrightarrow{pq?l} (\dots T_q' \dots ; \dots W_{pq} \dots)$$

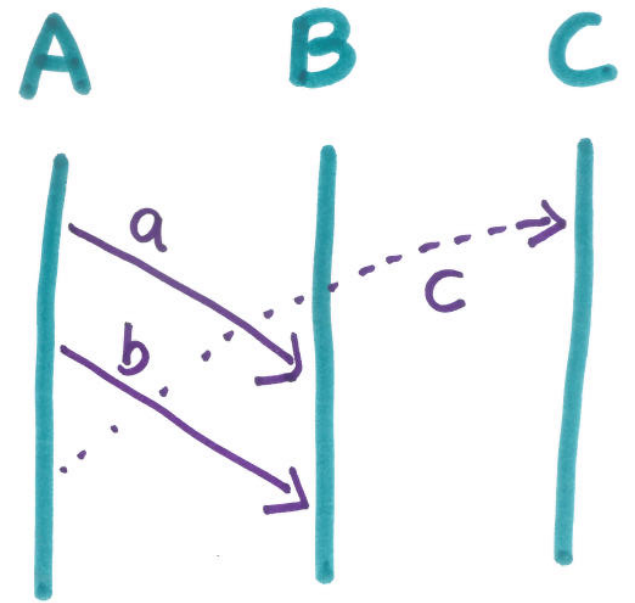
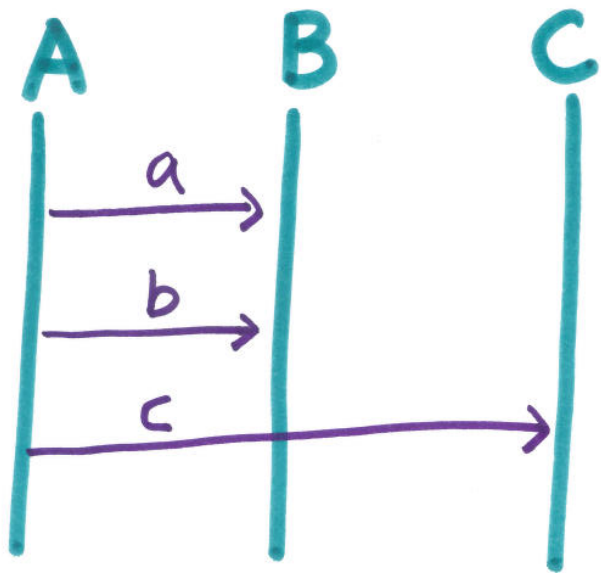
# LTS Global Types

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

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

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

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



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

↓  $AB!a$  OUT

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

↓  $AB?a$  IN

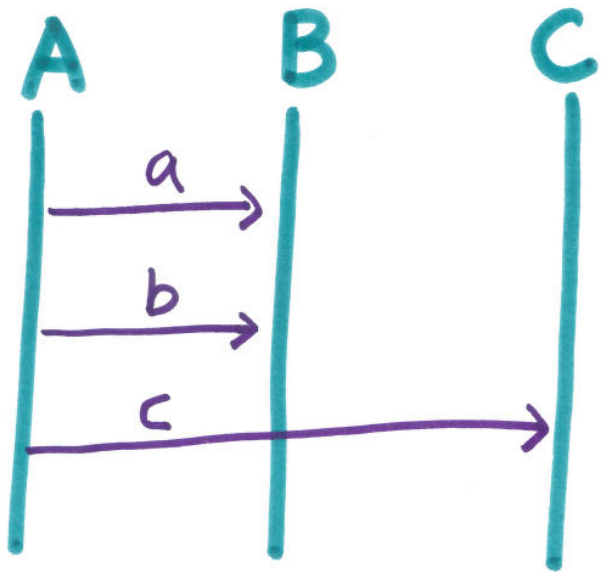
$A \rightarrow B:b \quad A \rightarrow C:c$

↓  $AB!b$  OUT

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

↓  $AB?b$  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

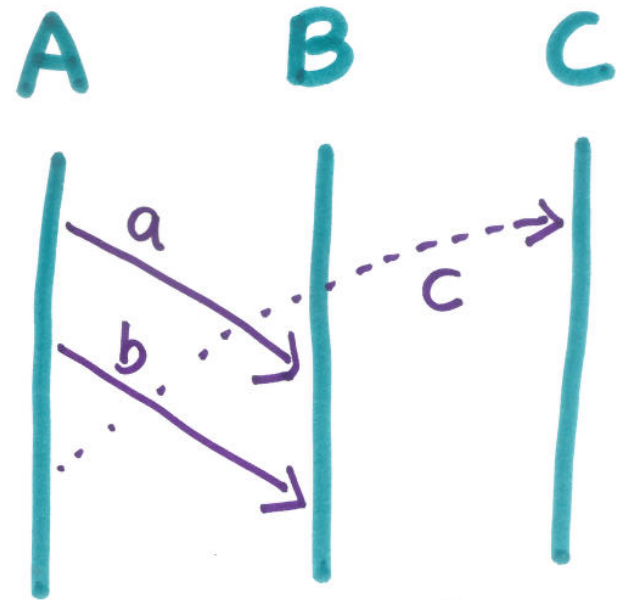
$A \rightarrow B:b. A \rightarrow C:c$

↓  $AB!b$  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

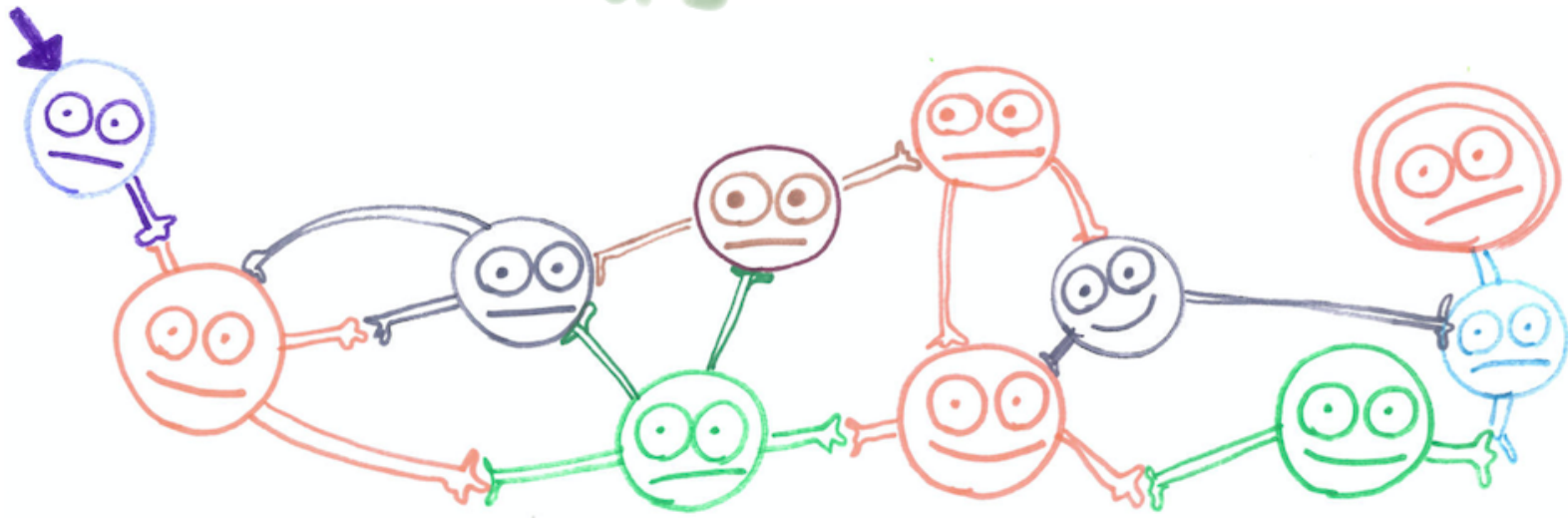
$A \rightsquigarrow B:a. A \rightsquigarrow B:b$

↓  $AB?a$  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

$\Sigma$  a finite alphabet of messages

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

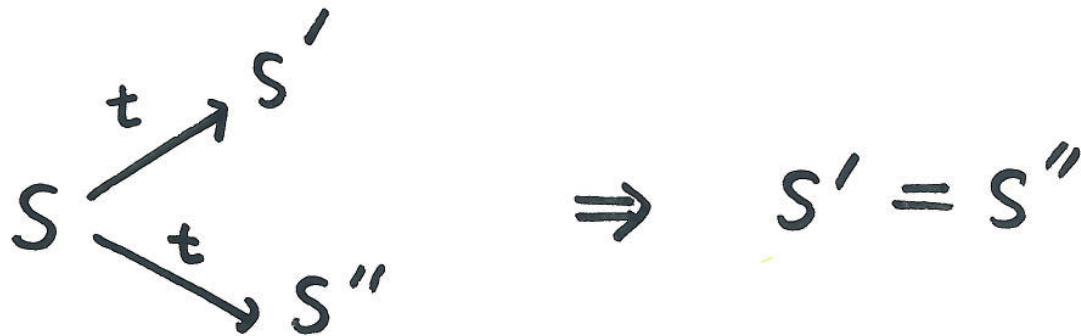
**Def** CS  $S = (M_p)_{p \in \text{Participant}}$

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

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

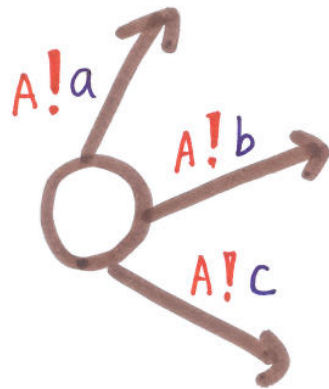
Receive  
 $(\dots q_q \dots; \dots l \cdot w_{pq} \dots) \xrightarrow{pq?l} (\dots q'_q \dots; \dots w_{pq} \dots)$

Deterministic CFSM



# Basic CFSMs

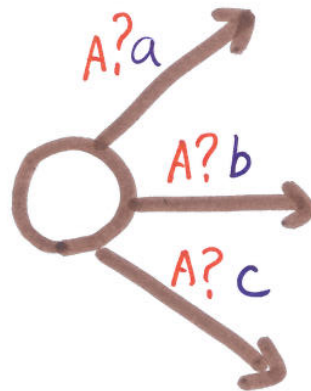
A CFSM is **Basic** if **deterministic**  
**directed**, has **no mixed states**



sending



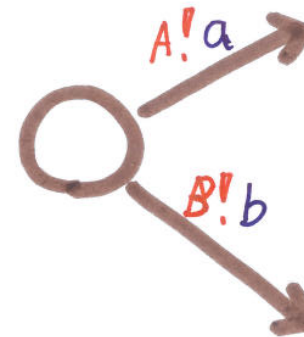
$$T = A!\{a, b, c\}$$



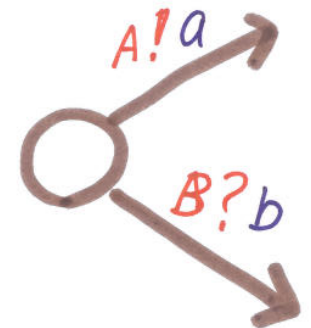
receiving



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



non  
directed



mixed



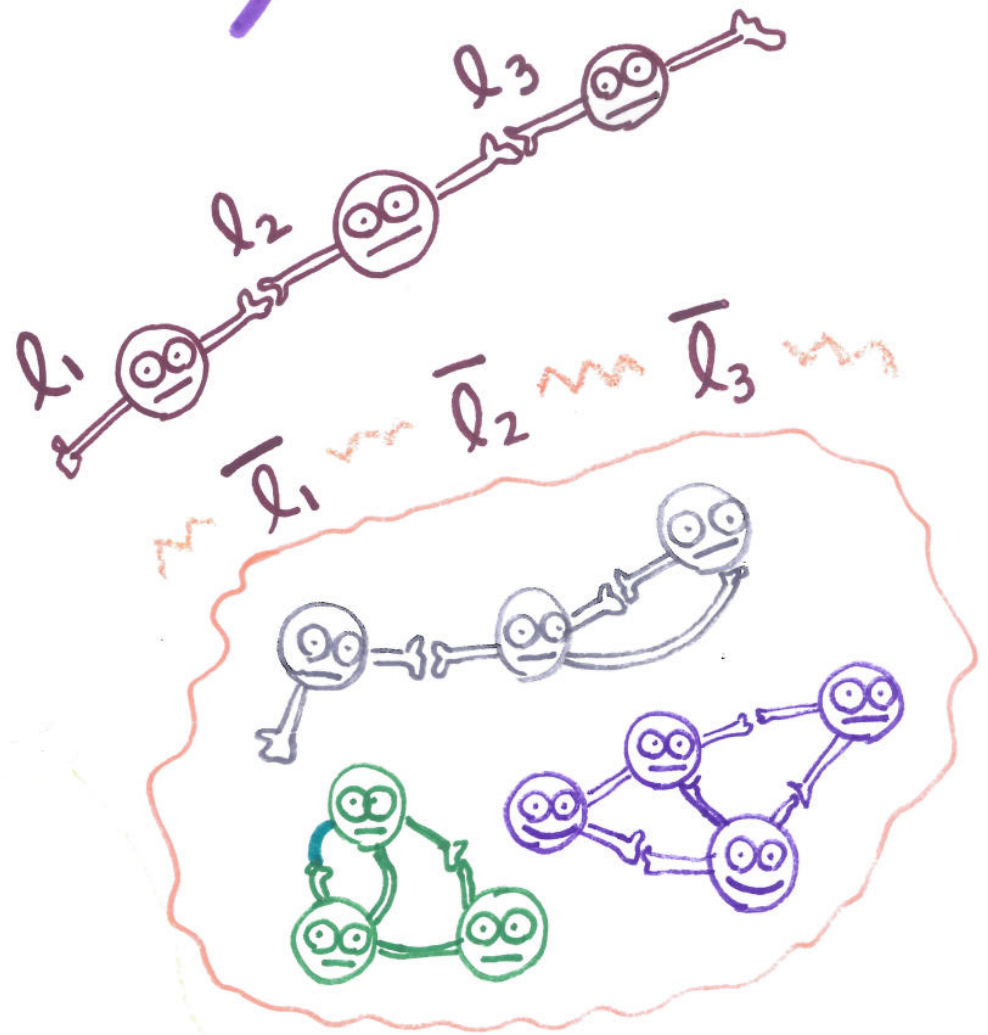
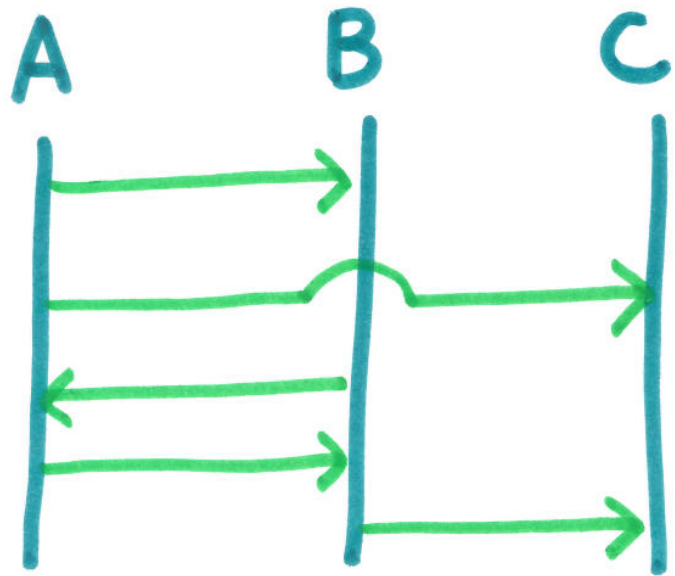
**Theorem 1**  $\text{Tr}(G) \approx \text{Tr}(\{G \upharpoonright P\}_{P \in \text{Participant}})$

Soundness and Completeness between  
a global type and <sup>projected</sup> local types

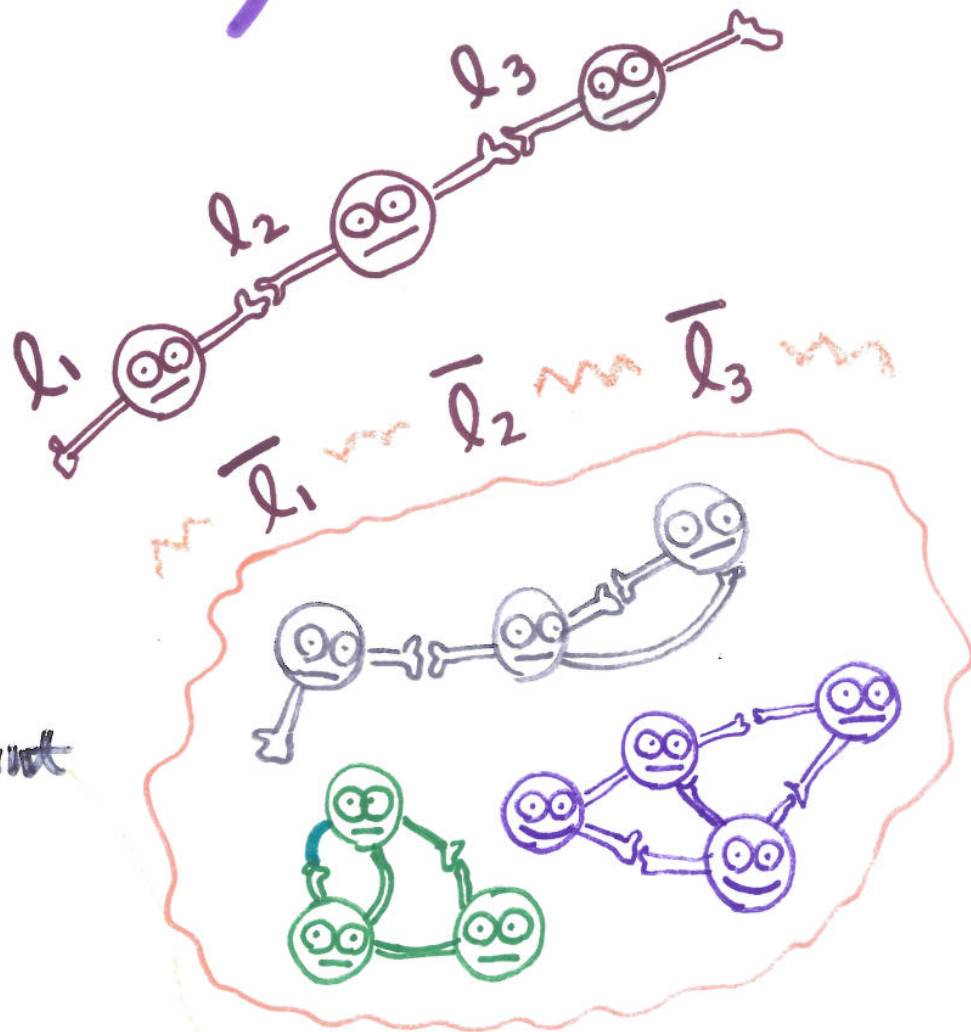
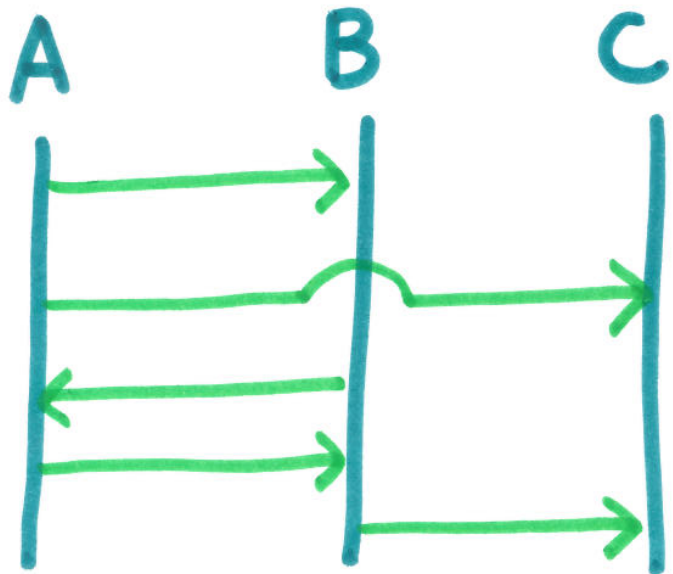
**Theorem 2**  $\text{Tr}(T) \approx \text{Tr}(A(T))_{T \text{ basic}}$

Soundness and Completeness between  
a local type and a basic CFSM of  $T$

# 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  $\rightsquigarrow$

**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  $\vec{w} = \vec{\epsilon}$  and not final, each  $q_p$  is receiving

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


**Unspecified Reception**  $(q_p, p \stackrel{?}{\underline{a}}, q'_p)$  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.

# Mobility Reading Group

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



## MobilityReadingGroup

π-calculus, Session Types research at Imperial College

- Home
- People
- Publications
- Grants
- Talks
- Tutorials
- Tools
- Awards
- Kohei Honda

### 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.