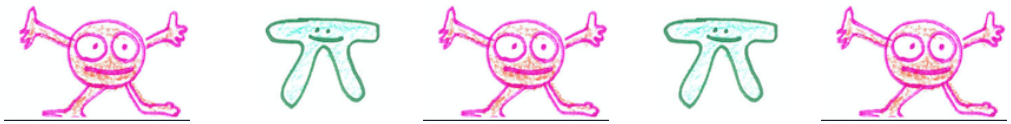
Multiparty Session Types and Communicating Automata

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

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

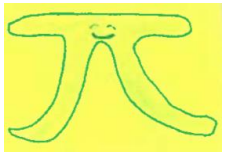
- Prevent concurrency bugs.
- Can abstract, implement and manage communications as **Protocols.**
- **Clean, Cheap** and **Retrofittable.**

Why Session Types, Why Now?

Significant academic and industry interests via fundamental breakthroughs

Milner, Honda, NY

Binary Session Types — ESOP'98

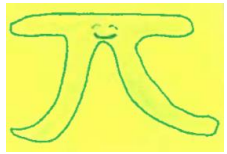


Joined W3C Standardization — 2002

Why Session Types, Why Now?

Significant academic and industry interests via fundamental breakthroughs

Binary Session Types — ESOP'98



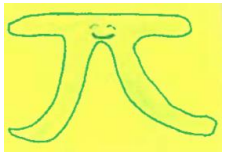
Joined W3C Standardization — 2002

Multiparty Session Types — POPL' 08

Why Session Types, Why Now?

Significant academic and industry interests via fundamental breakthroughs

Binary Session Types — ESOP'98



↓

Joined W3C Standardization — 2002

↓

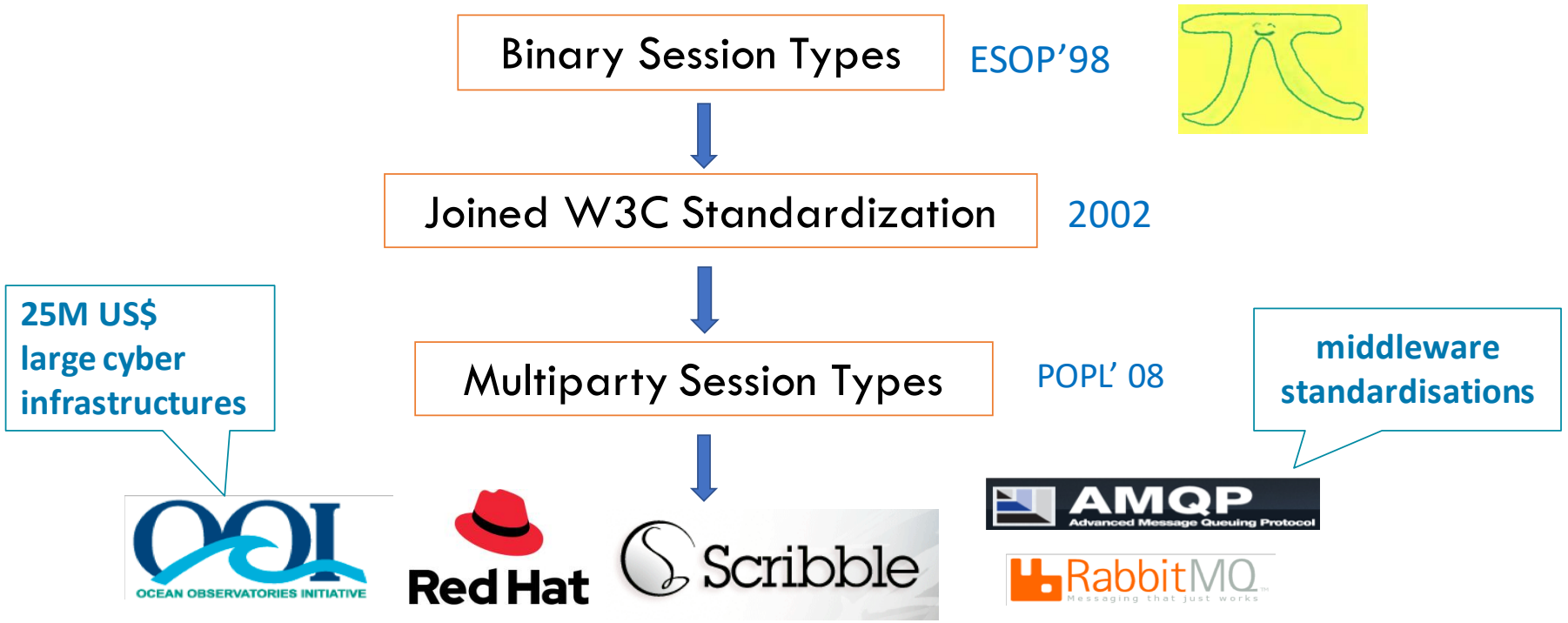
Multiparty Session Types — POPL' 08

↓

largest open source company in the world

Why Session Types, Why Now?

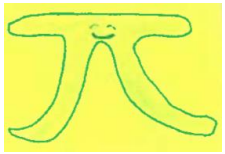
Significant academic and industry interests via fundamental breakthroughs



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

















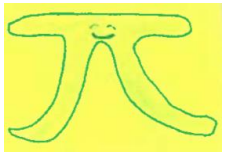



Why Session Types, Why Now?

Significant academic and industry interests via fundamental breakthroughs

ETAPS Test Time Award 2019

Binary Session Types — ESOP'98

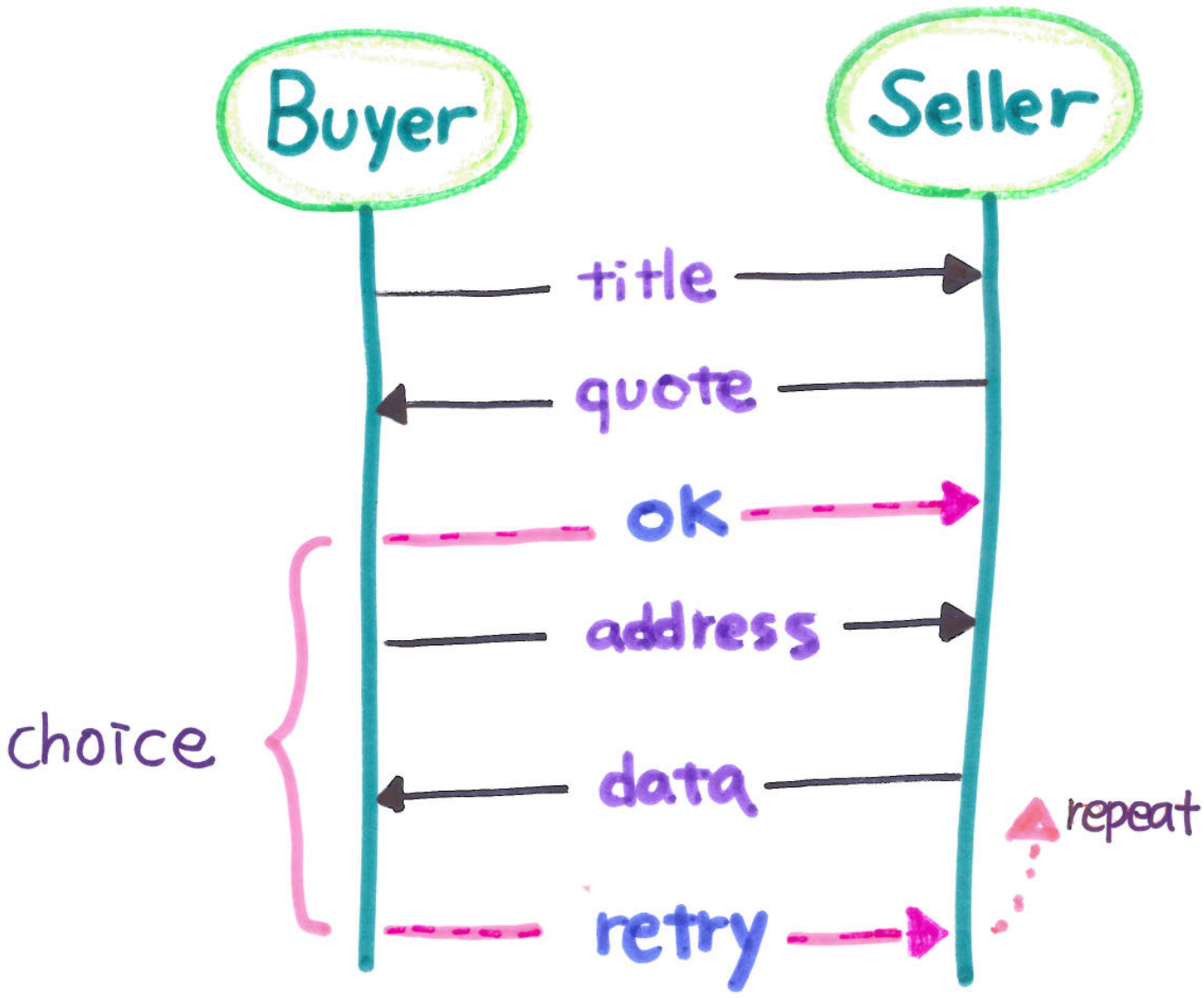


Joined W3C Standardization — 2002

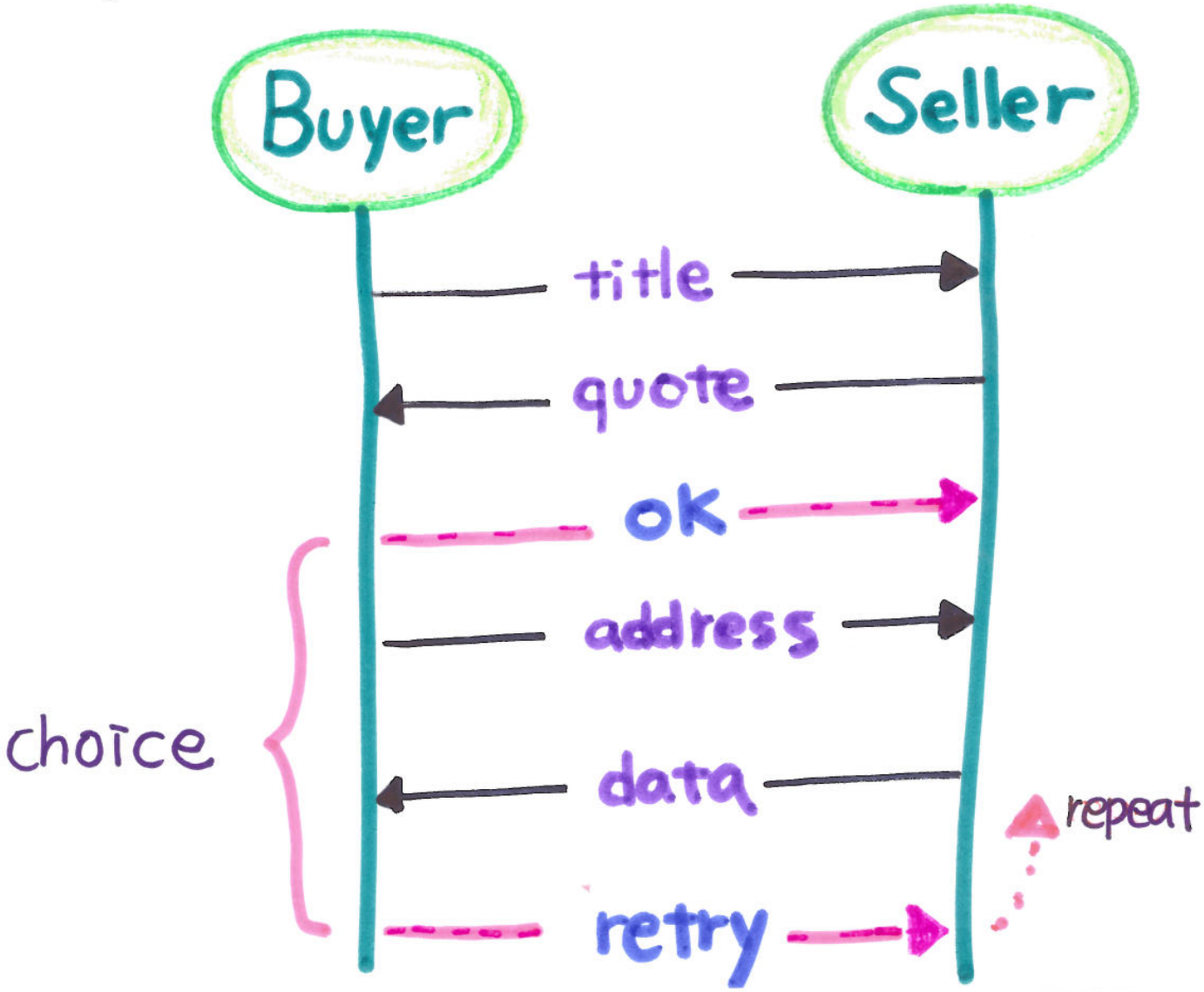
POPL Influential Paper Award 2018

Multiparty Session Types — POPL' 08

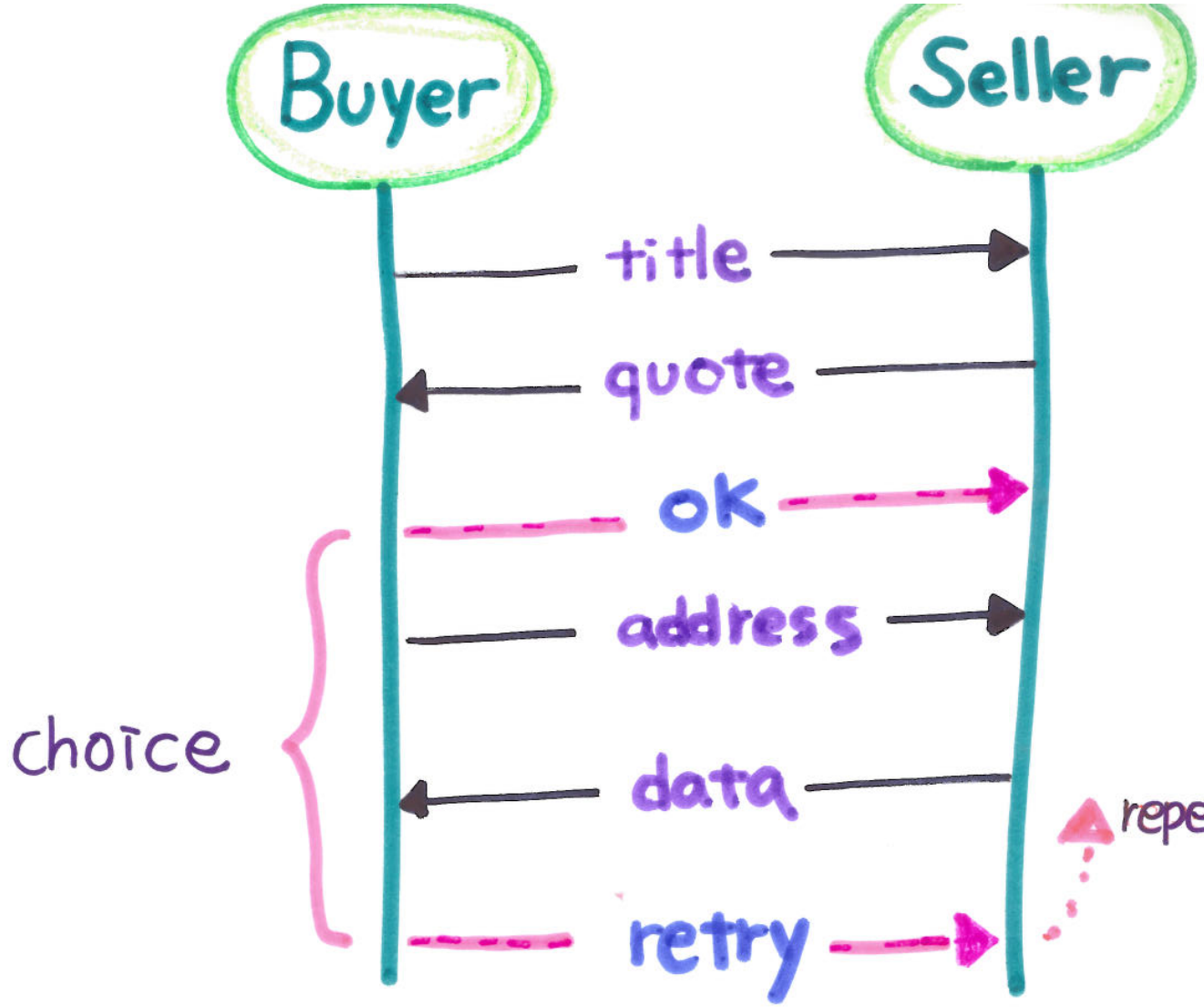
Binary Session Types: Buyer - Seller Protocol



Binary Session Types: Buyer - Seller Protocol



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



P has T

Q has $\overline{T}$

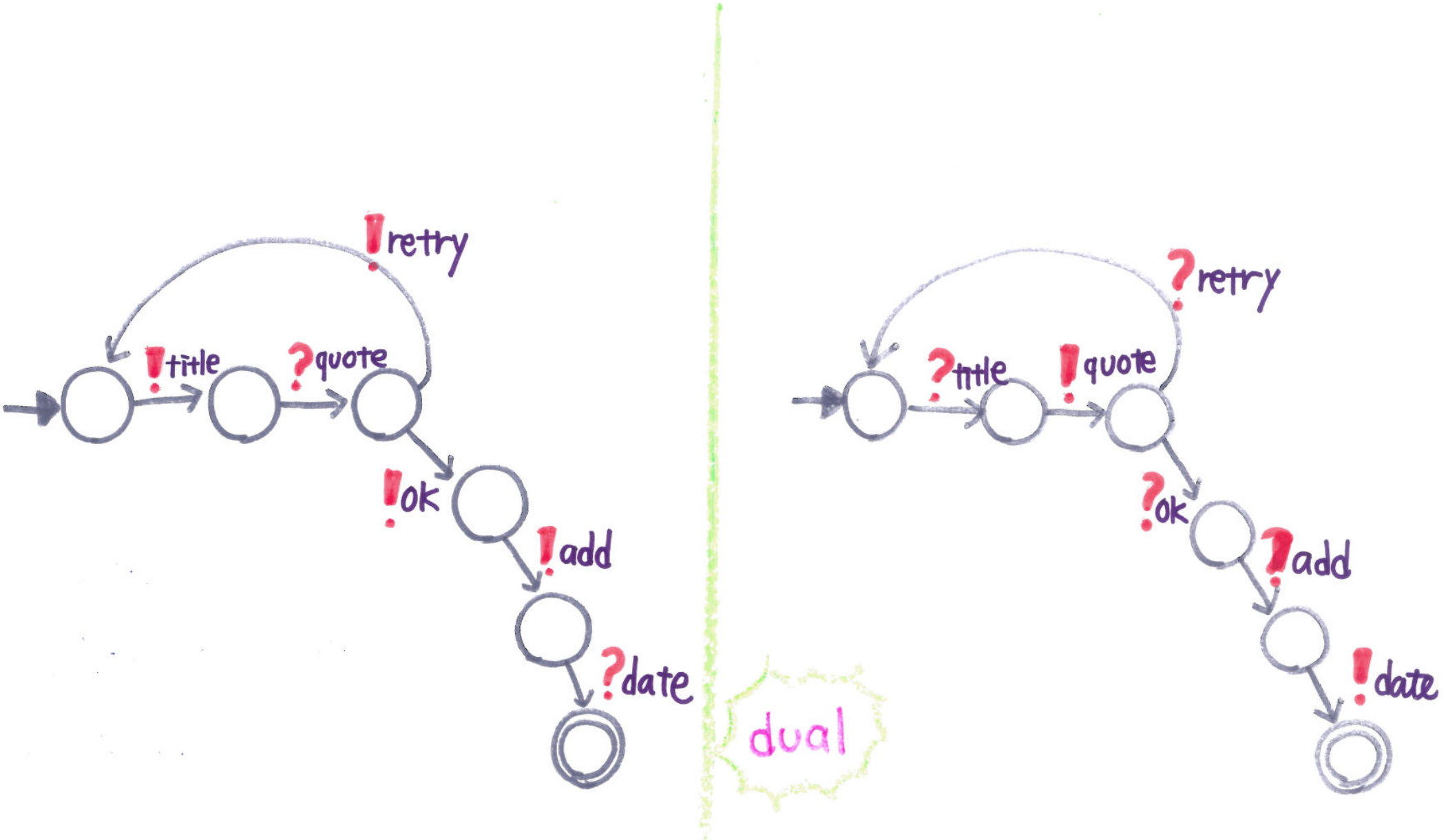
dual

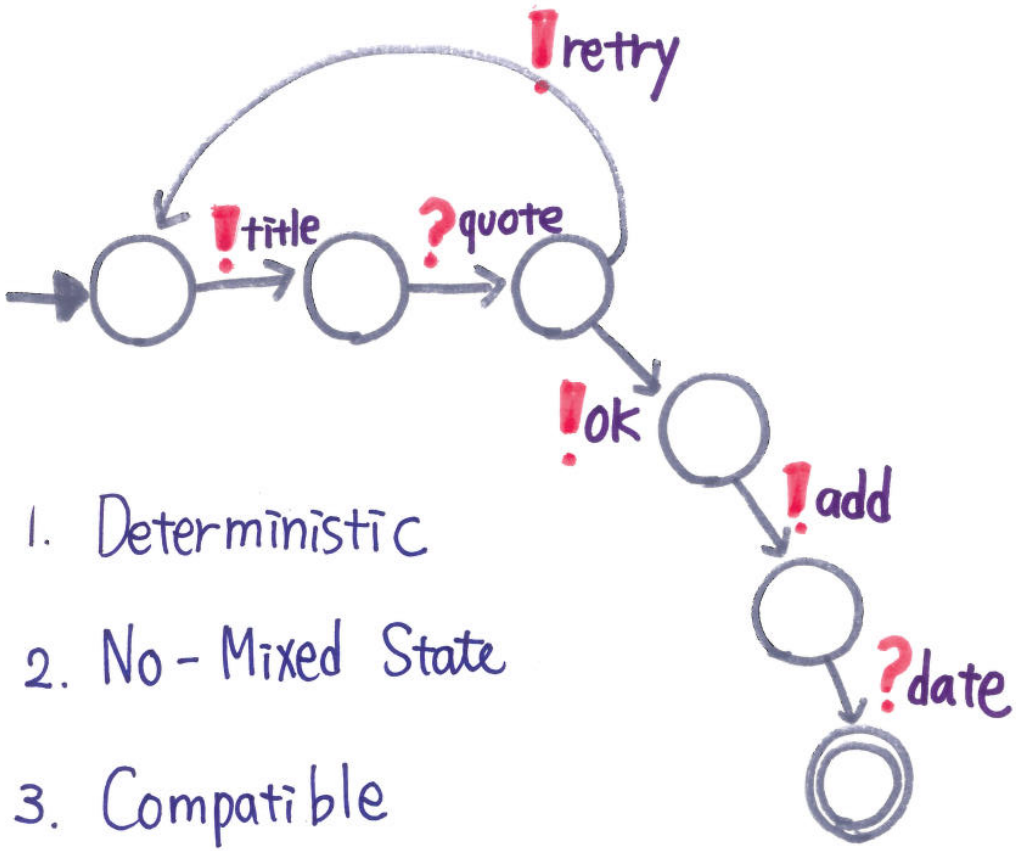
P | Q typable

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

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

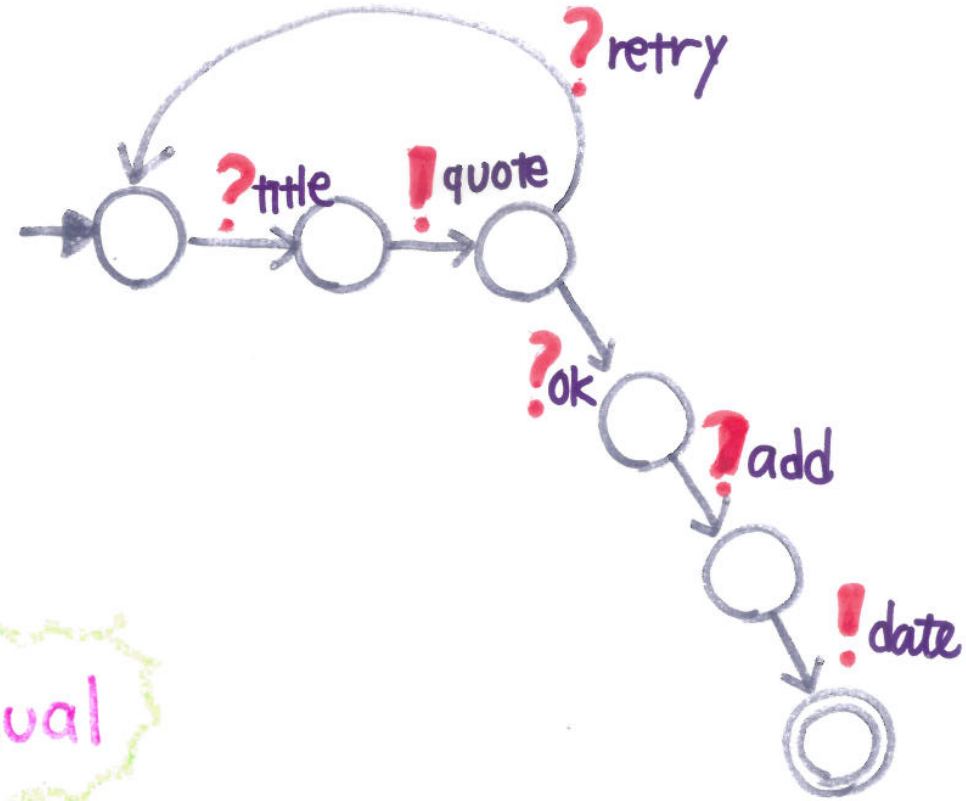
Communicating Automata [1980s]





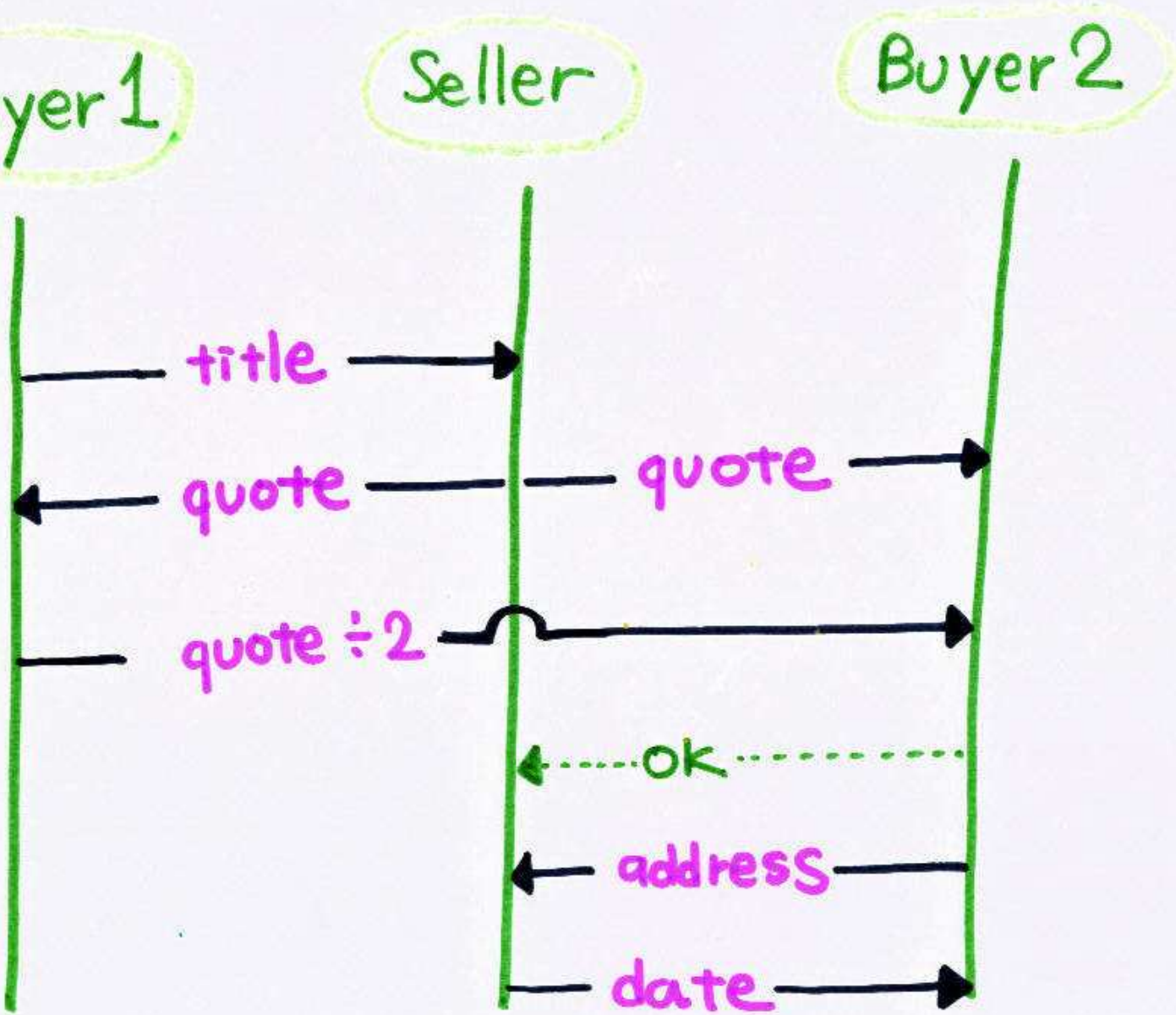
1. Deterministic
2. No-Mixed State
3. Compatible

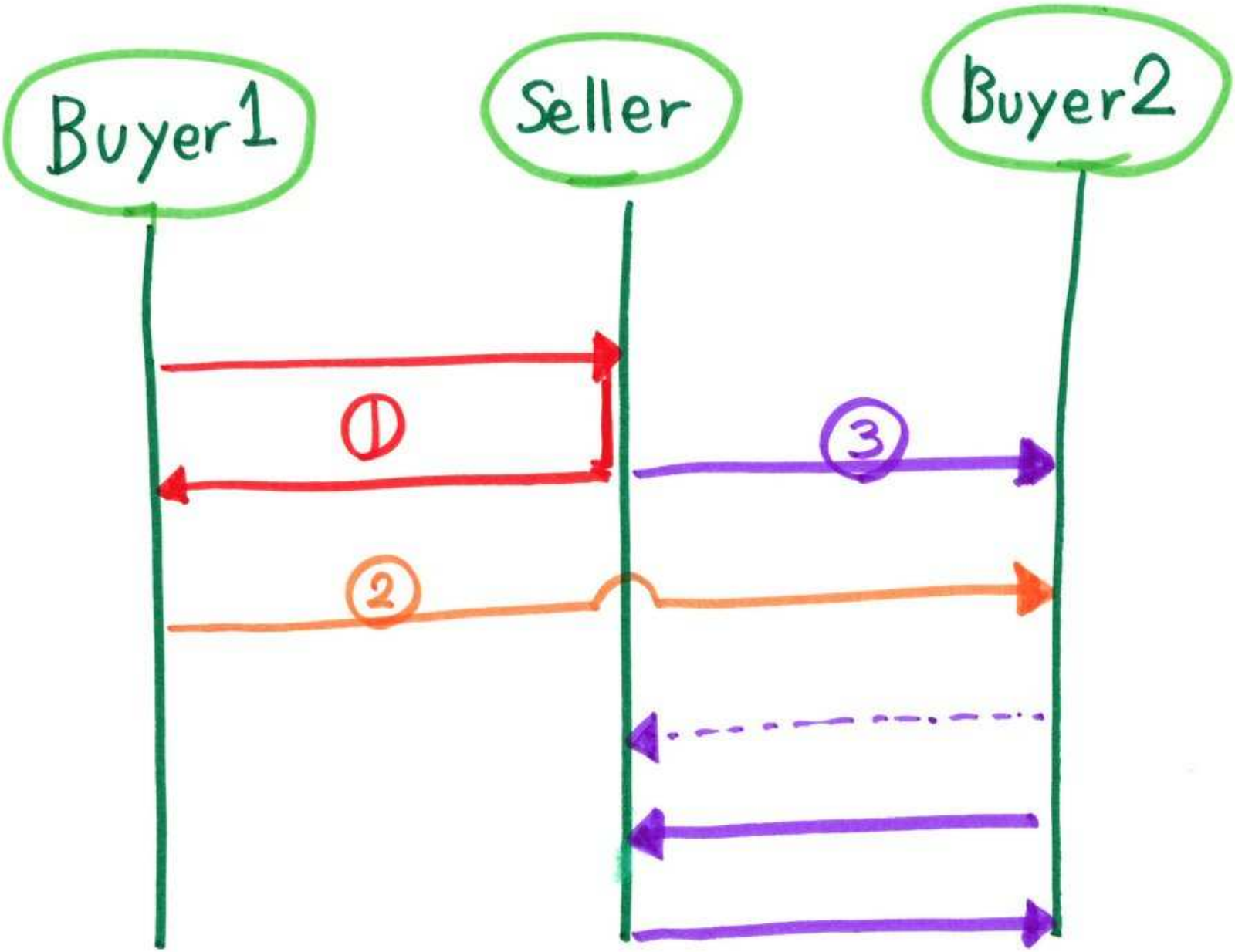
dual

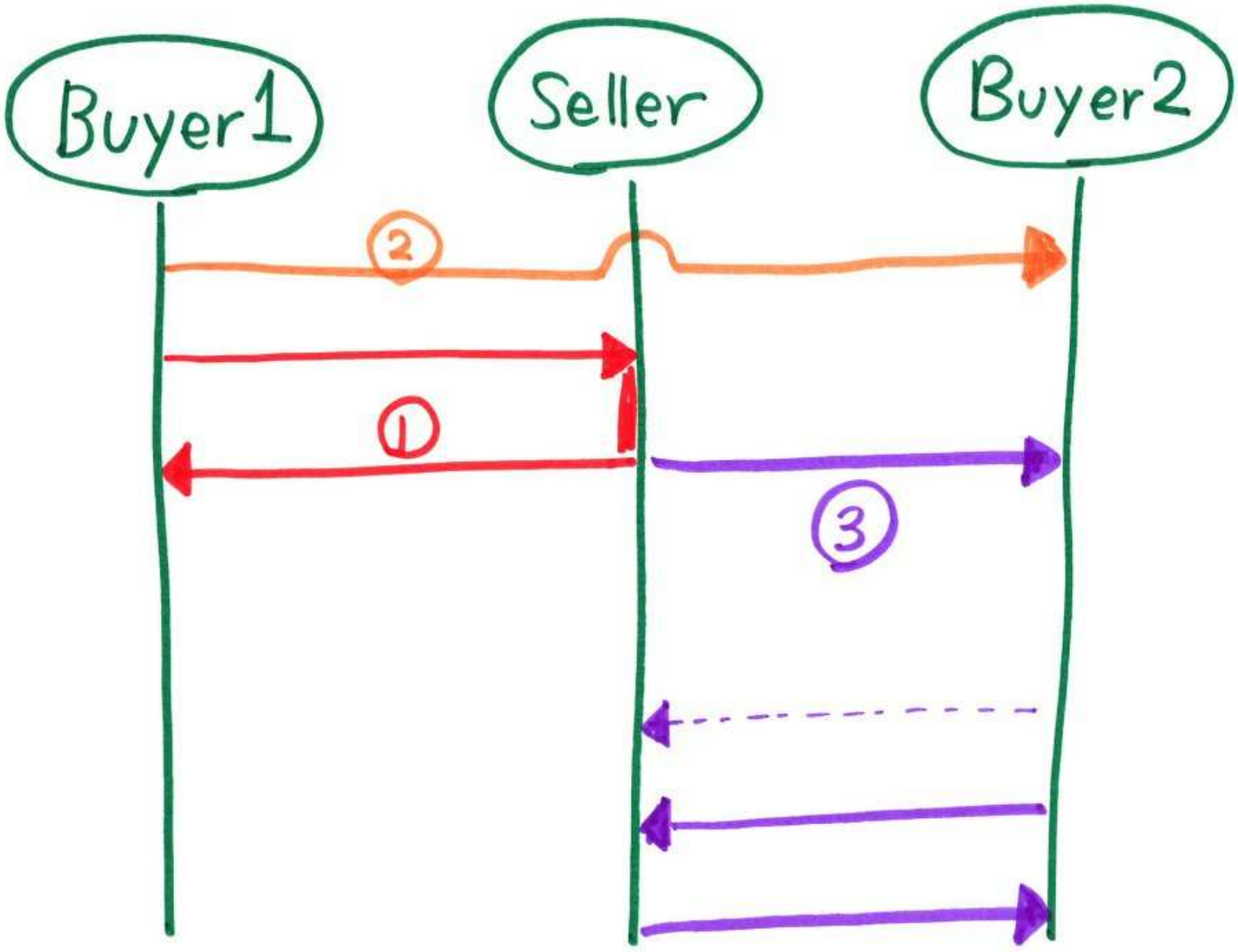


[Gouda et al 1986] Two compatible machines without mixed states which are deterministic satisfy deadlock-freedom.

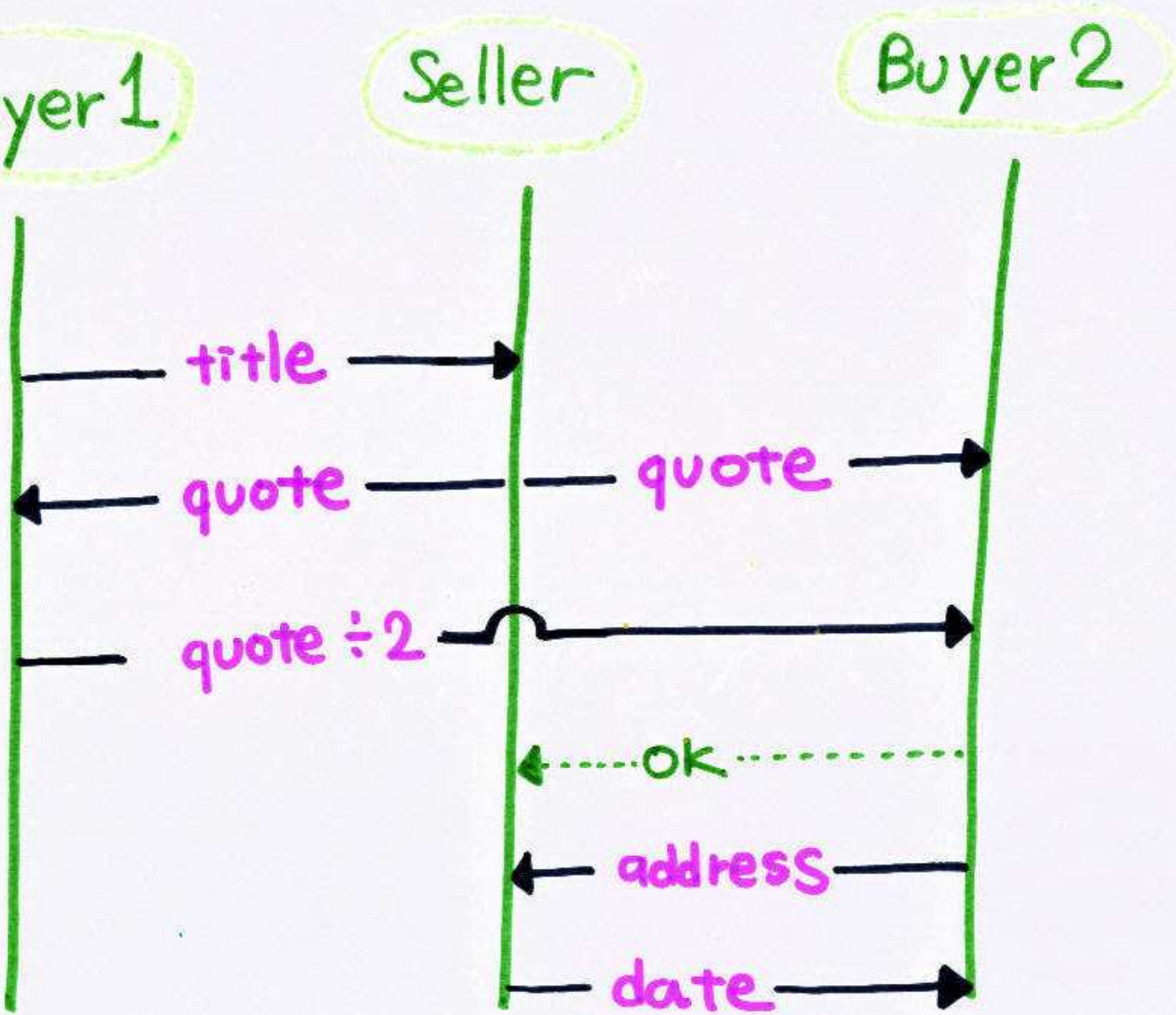
Multiparty Session Types

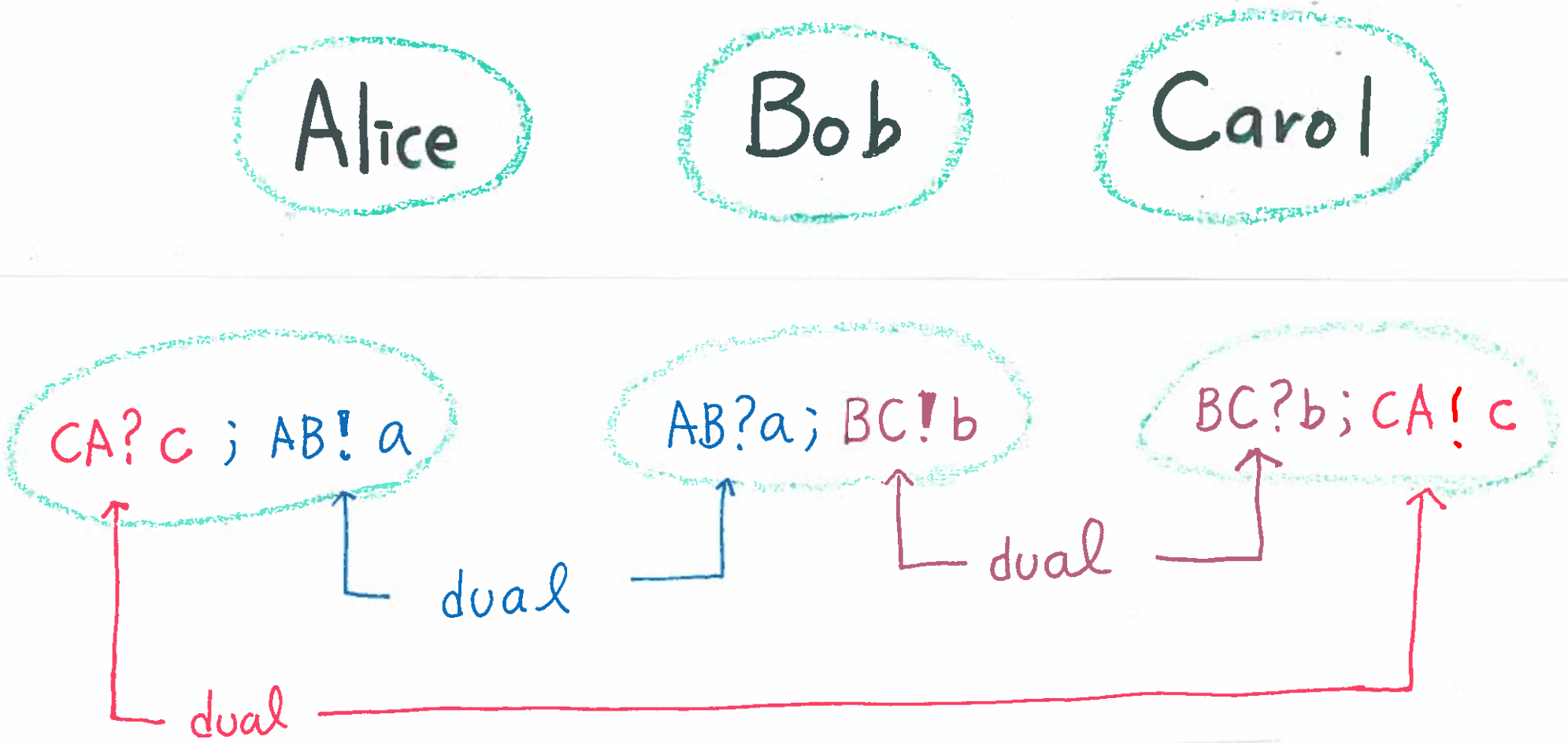



Buyer1
Seller
Buyer2
1
3
2


Buyer1
Seller
Buyer2
2
1
3

Multiparty Session Types



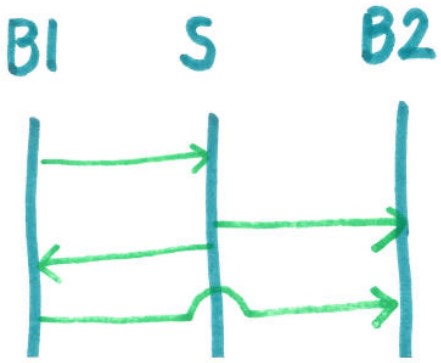


3 dual pairs

If you use binary Session Types ....

Deadlock!

Multiparty Session Types [Honda, Yoshida, Carbone 2008]



G

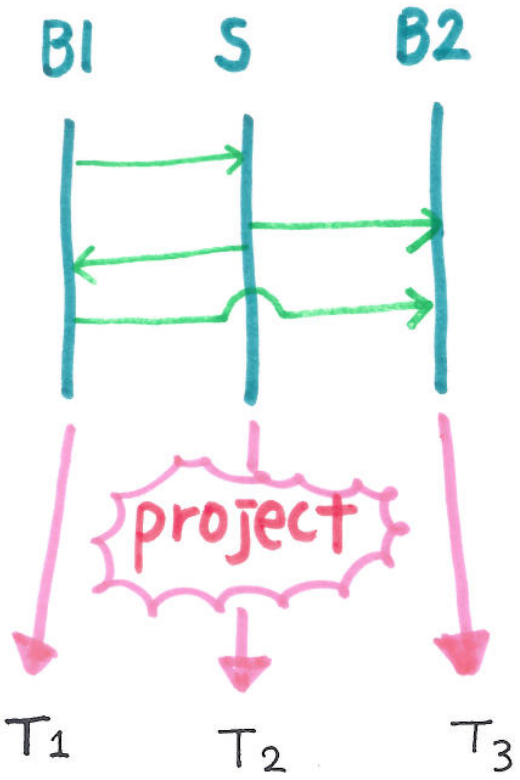
B1 → S Int.

S → B2 Char

STEP 1

Write Global Type

Multiparty Session Types [Honda, Yoshida, Carbone 2008]



G

$B1 \rightarrow S$ Int.

$S \rightarrow B2$ Char

T

$B_1?$ Int. $B_2!$ Char

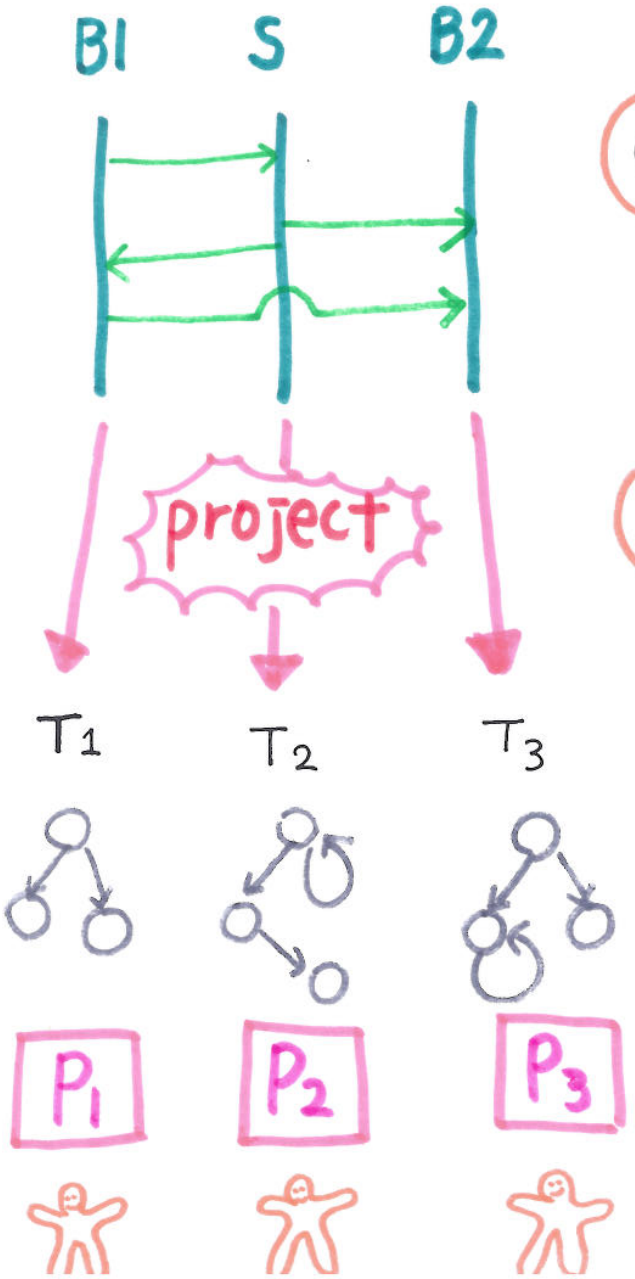
STEP 1

Write Global Type

STEP 2

Project to Local Types

Multiparty Session Types [Honda, Yoshida, Carbone 2008]



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

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

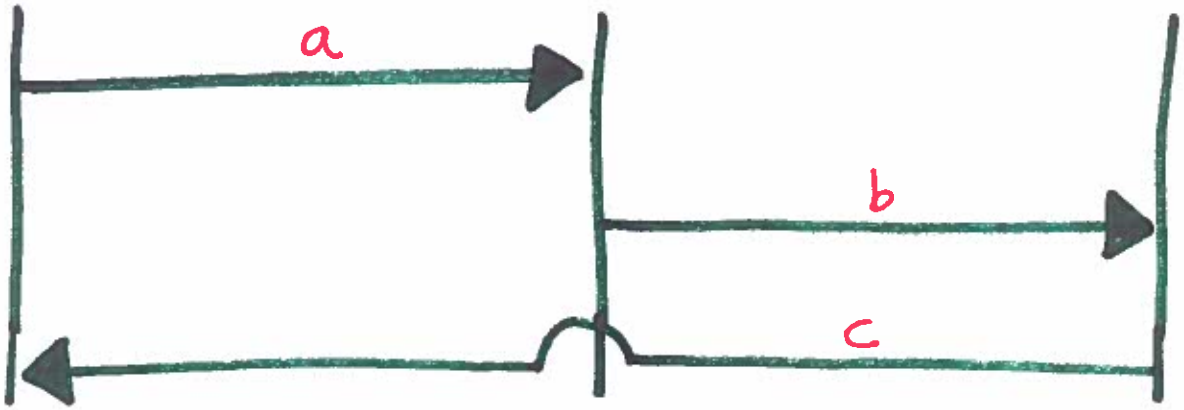
P $B_1?(x).\, B_2!\langle\text{"apple"}\rangle$

STEP 1
Write Global Type

STEP 2
Project to Local Type

STEP 3
- Static Check
- Generate Code
- Run-time Check

Alice Bob Carol



Global Type

Projection

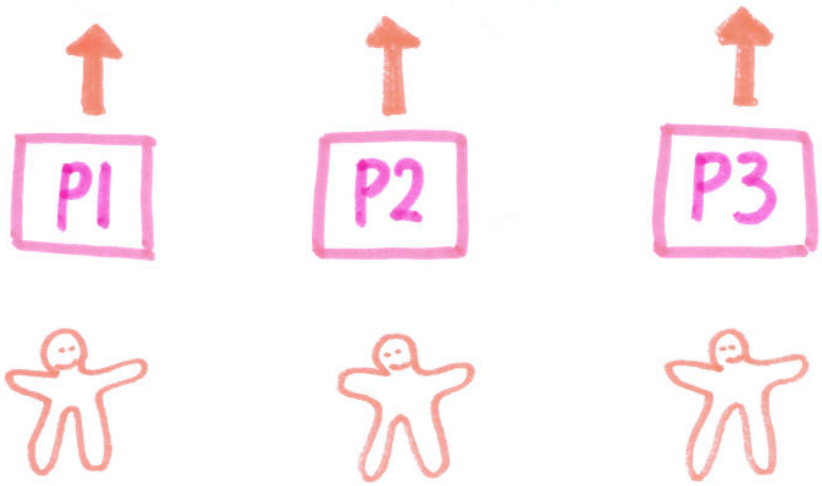
Alice AB!a;CA?c

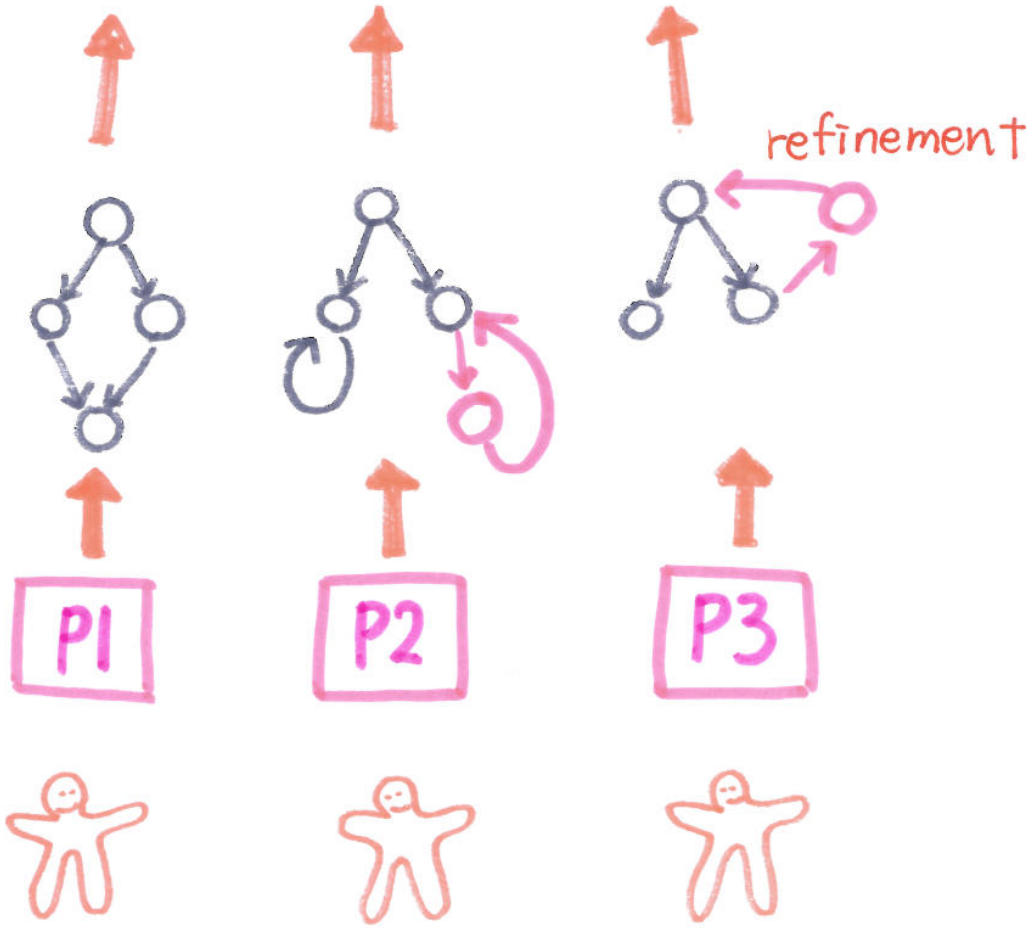
Bob AB?a;BC!b

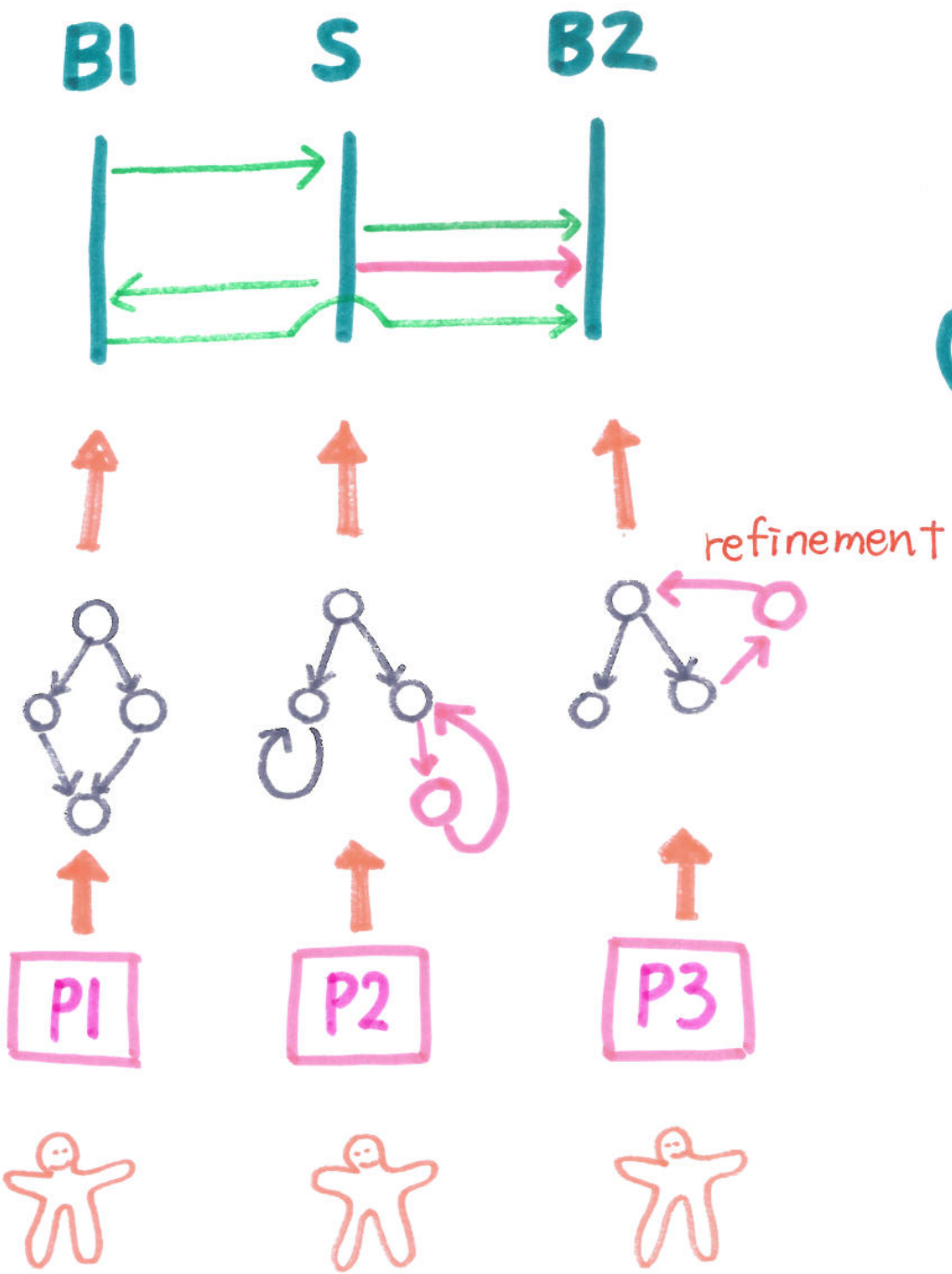
Carol BC?b;CA!c;

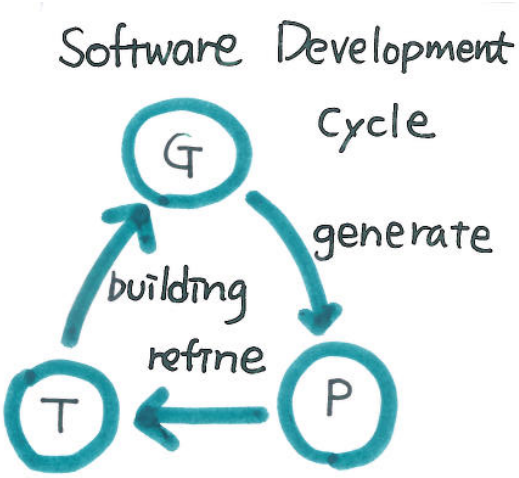
LOCAL TYPES

NO Deadlock


P1
P2
P3


refinement
P1
P2
P3





- Optimisation
- refinement
- Inference
- Testing

Global Types

$$G ::= p \to p' : \{ a_j . G_j \}_{j \in J}$$

$$\mid \quad \mu t . G$$

$$\mid \quad t$$

$$\mid \quad \text{end}$$

Local Types

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

$$\mid \; p ? \{ a_j . T_j \}_{j \in J}$$

$$\mid \; \mu t . T$$

$$\mid \; t$$

$$\mid \; \text{end}$$

$p, p', \ldots$ Participant

a, b, c Σ Alphabet

Projection from G onto q $\quad G \upharpoonright q$

$$\left(p \to p' : \{a_j . G_j\}_{j \in J}\right) \upharpoonright q$$

$$= \begin{cases} p' ! \{a_j . G_j \upharpoonright q\}_{j \in J} & q = p \\ p ? \{a_j . G_j \upharpoonright q\}_{j \in J} & q = p' \\ G_1 \upharpoonright q = G_j \upharpoonright q & \forall j \in J \quad \text{otherwise} \end{cases}$$

Projection from G onto q $\quad G\upharpoonright q$

$$(p \to p' : \{a_j . G_j\}_{j \in J})\upharpoonright q$$

$$= \begin{cases} p'!\{a_j . G_j\upharpoonright q\}_{j \in J} & q = p \\ p?\{a_j . G_j\upharpoonright q\}_{j \in J} & q = p' \\ G_1\upharpoonright q = G_j\upharpoonright q & \forall j \in J \quad \text{otherwise} \end{cases}$$

Bad $\quad A \to B : \{a . C \to D : c \;,\; b . C \to D : d\}$

Good $\quad A \to B : \{a . C \to D : c \;,\; b . C \to D : c\}$

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

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

$$\frac{T[\mu t.T/t] \xrightarrow{\ell} T'}{\mu t.T \xrightarrow{\ell} T'}$$

$(\vec{T};\vec{w})$ w_{pq} queue

Send $T_p \xrightarrow{pq!\ell} T_p' \Rightarrow$

$(\ldots T_p \ldots ; \ldots w_{pq} \ldots) \xrightarrow{pq!\ell} (\ldots T_p' \ldots ; \ldots w_{pq} \cdot \ell \ldots)$

Receive $T_q \xrightarrow{pq?\ell} T_q' \Rightarrow$

$(\ldots T_q \ldots ; \ldots \ell \cdot w_{pq} \ldots) \xrightarrow{pq?\ell} (\ldots T_q' \ldots ; \ldots w_{pq} \ldots)$

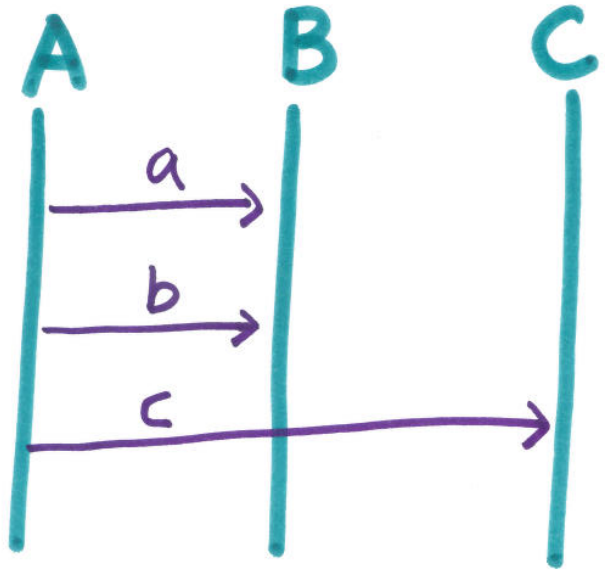
LTS Global Types

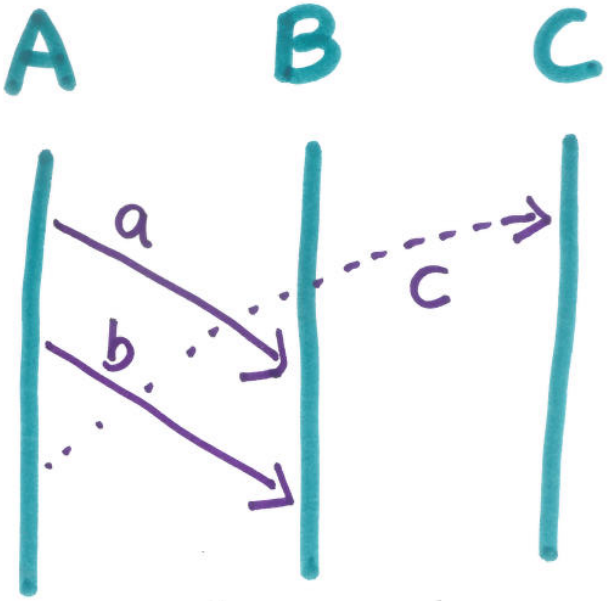
$p \rightsquigarrow p' : \tilde{\jmath}\, \{a_i . G_i\}_{i \in I}$

put $\quad p \to p' : \{a_i . G_i\}_{i \in I} \xrightarrow{pp'!a_j} p \rightsquigarrow p' : \tilde{\jmath}\, \{a_i . G_i\}_{i \in I}$

get $\quad p \rightsquigarrow p' : \tilde{\jmath}\, \{a_i . G_i\}_{i \in I} \xrightarrow{pp'?a_j} G_j$

$$\frac{\forall j \in I \quad G_j \xrightarrow{\ell} G_j' \qquad p, q \notin \mathrm{sub}(\ell)}{p \to p' : \{a_i . G_i\}_{i \in I} \xrightarrow{\ell} p \to p' : \{a_i . G_i'\}_{i \in I}} \qquad \frac{G_j \xrightarrow{\ell} G_j' \qquad q \notin \mathrm{sub}(\ell) \qquad \forall i \in I \setminus j \;\; G_i' = G_i}{p \rightsquigarrow q : \tilde{\jmath}\, \{a_i . G_i\}_{i \in I} \xrightarrow{\ell} p \rightsquigarrow q : \tilde{\jmath}\, \{a_i . G_i'\}_{i \in I}}$$





$A \to B:a \,.\, A \to B:b \,.\, A \to C:c$

↓ AB!a OUT

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

↓ AB?a IN

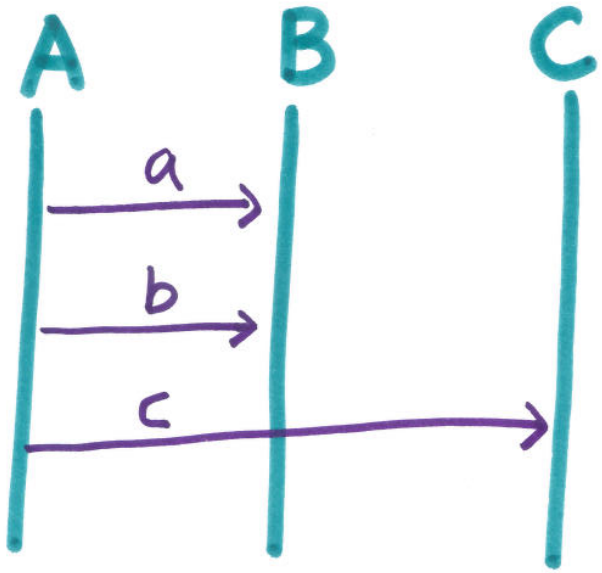
$A \to B:b \;\; A \to C:c$

↓ AB!b OUT

$A \rightsquigarrow B:b \;\; A \to C:c$

↓ AB?b IN

$A \to C:c$



$A \to B:a.\ A \to B:b,\ A \to C:c$

↓ AB!a OUT

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

↓ AB?a IN

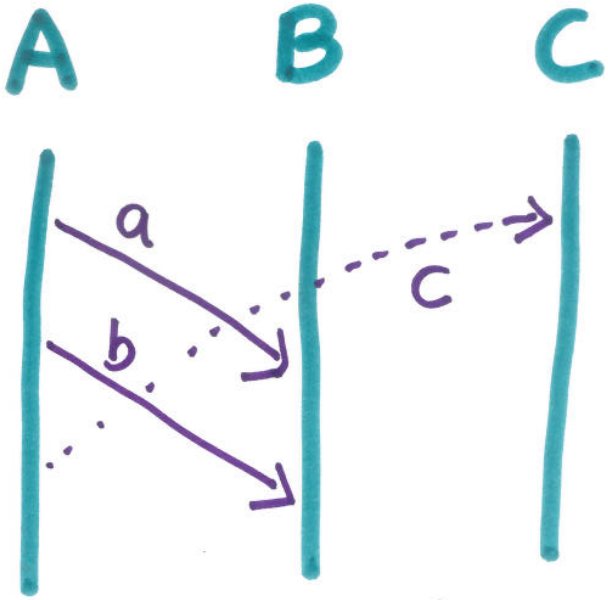
$A \to B:b\ \ A \to C:c$

↓ AB!b OUT

$A \rightsquigarrow B:b\ \ A \to C:c$

↓ AB?b IN

$A \to C:c$



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

↓ AB!a OUT

↓ AB!b OUT

↓ AC!c OUT

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

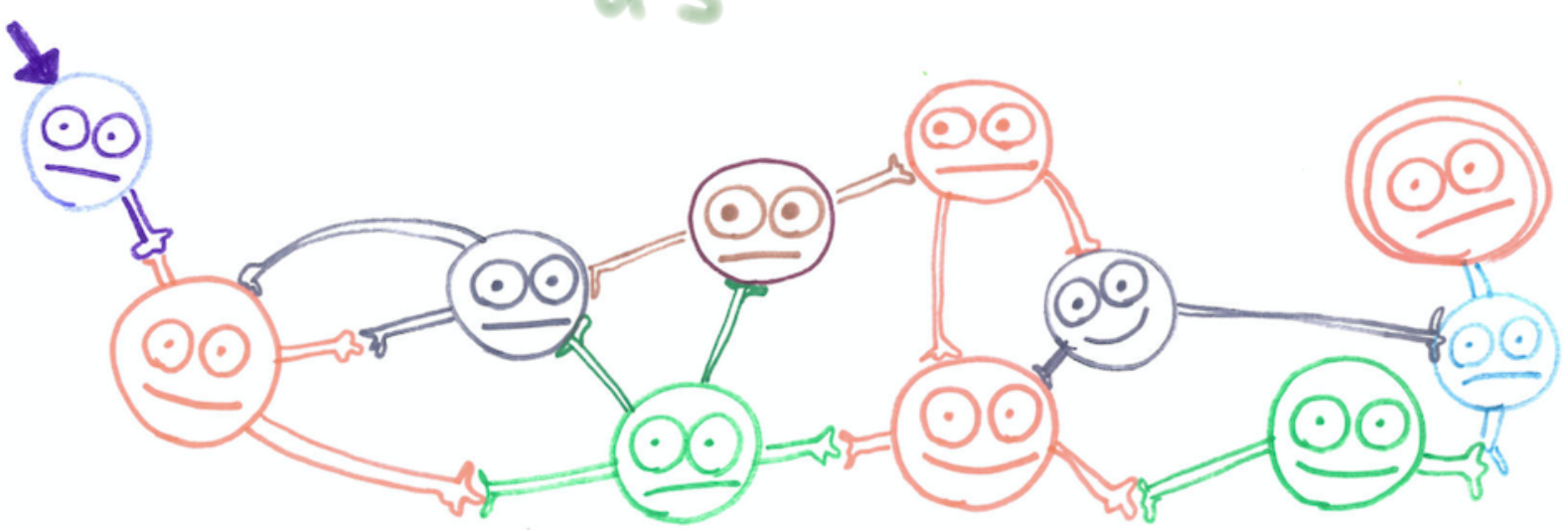
↓ AC?c IN

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

↓ AB?a IN

$A \rightsquigarrow B:b$

Multiparty Session Types as



?? Communicating Automata ??

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

Def A CFSM $M = (Q, C, q0, \Sigma, \delta)$

Q a finite set of states

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

$q0$ 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 $S = (M_p)_{p \in Participant}$

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

states queues

q_p w_{pq}

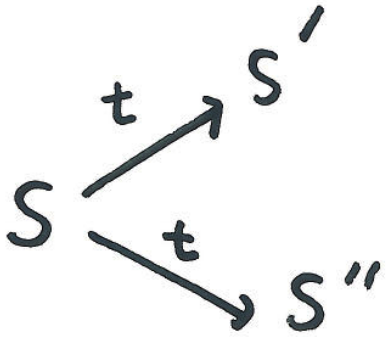
Send

$(\ldots q_p \ldots\,;\ldots w_{pq} \ldots) \xrightarrow{pq!\ell} (\ldots q_p' \ldots\,;\ldots w_{pq} \cdot \ell \ldots)$

Receive

$(\ldots q_q \ldots\,;\ldots \ell \cdot w_{pq} \ldots) \xrightarrow{pq?\ell} (\ldots q_q' \ldots\,;\ldots w_{pq} \ldots)$

Deteministic CFSM

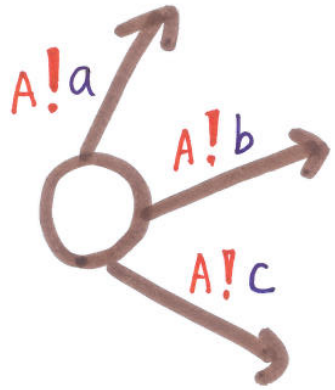


$\Rightarrow \quad S' = S''$

Basic CFSMs

A CFSM is **Basic** if **deterministic**

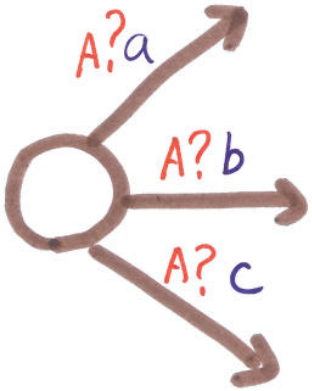
directed, has **no mixed states**



sending

✓

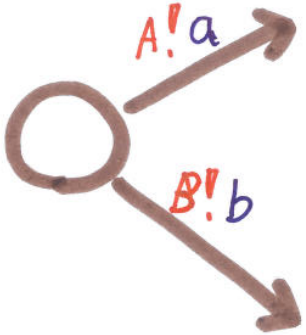
$T = A!\{a, b, c\}$



receiving

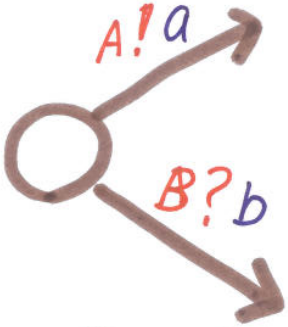
✓

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



non directed

✗



mixed

✗

Theorem 1 $Tr(G) \approx Tr(\{G\lceil p\}_{p \in Participant})$

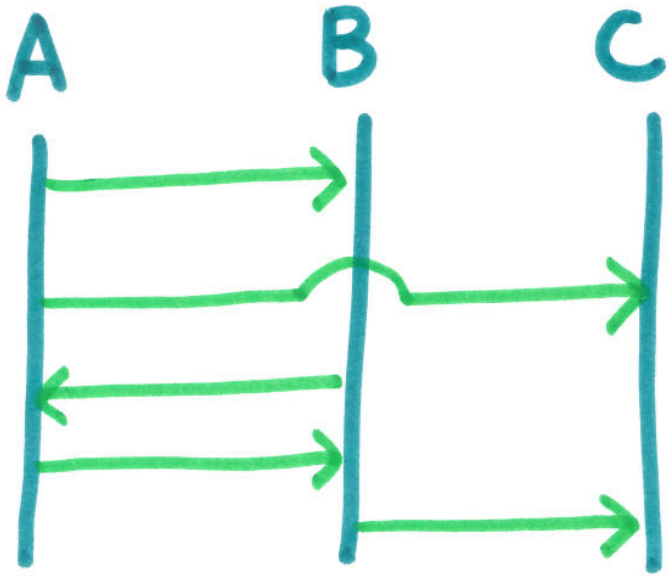
Soundness and Completeness between a global type and projected local types

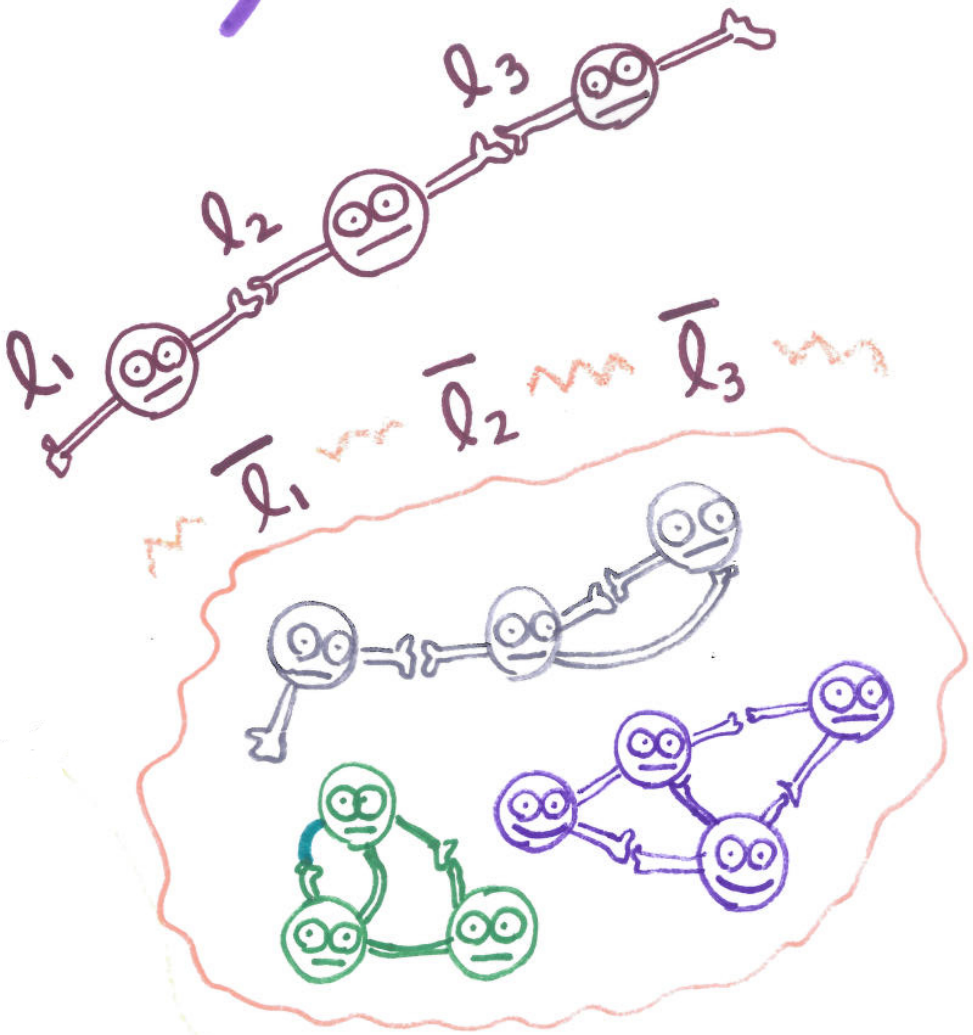
Theorem 2 $Tr(T) \approx Tr(A(T))$ T basic

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

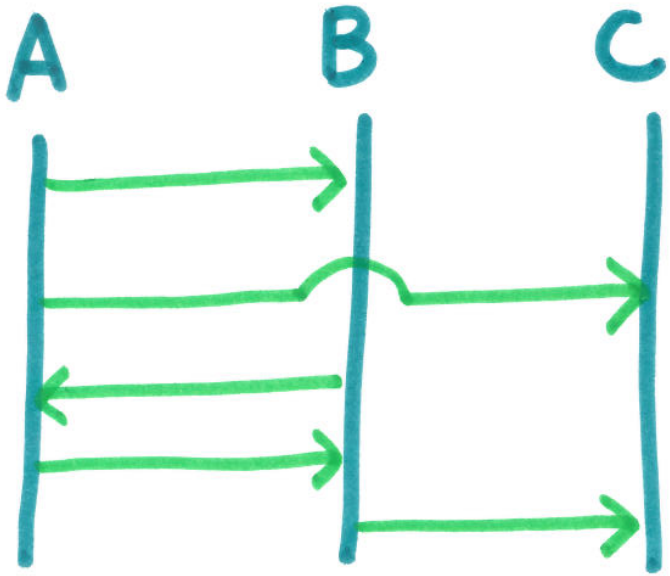
T

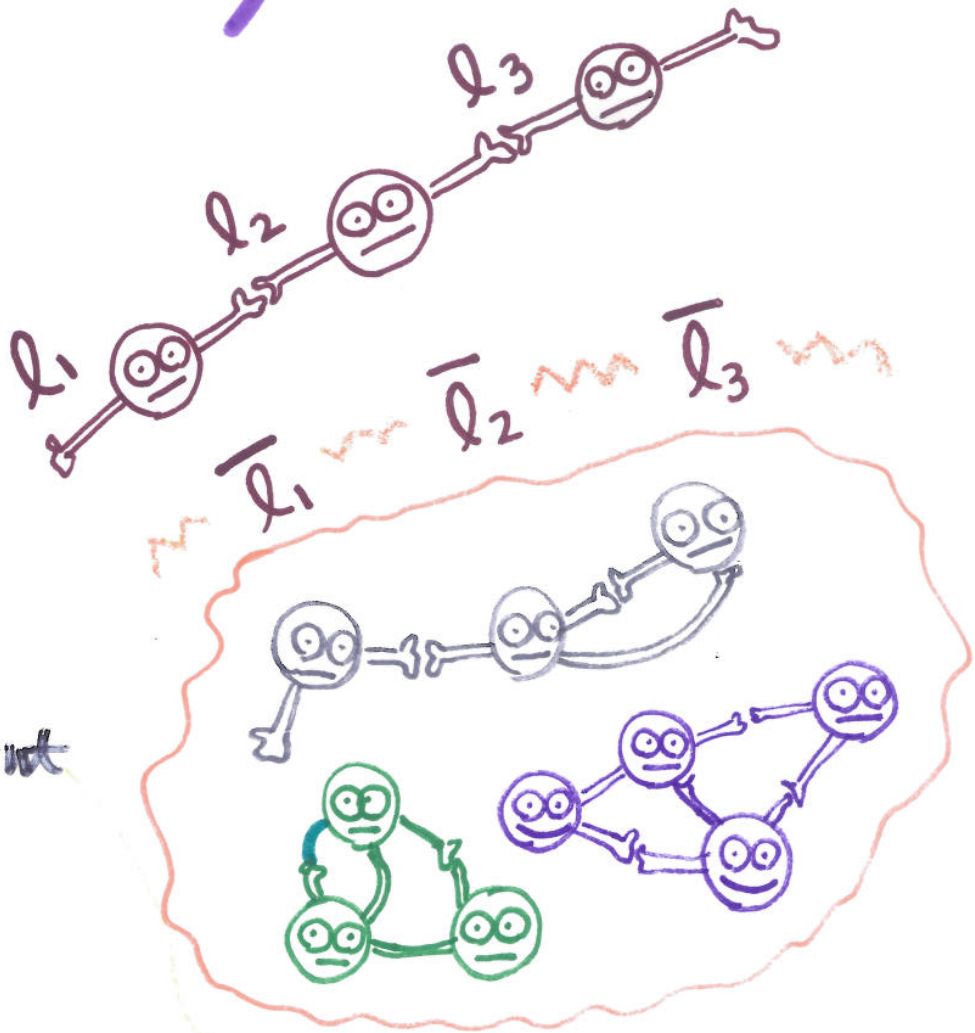
Multiparty Compatibility





Multiparty Compatibility





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

$\forall s \,.\, s_0 \rightsquigarrow s$ (1-buffer execution)

if M_i does action ℓ

then $(M_j)_{j \in P \setminus i}$ do action $\bar{\ell}$

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

Orphan Message all q are final, but $\vec{w} \neq \vec{\varepsilon}$

Unspecified Reception $(q_p, pq?\underline{a}, q'_p)$ 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/