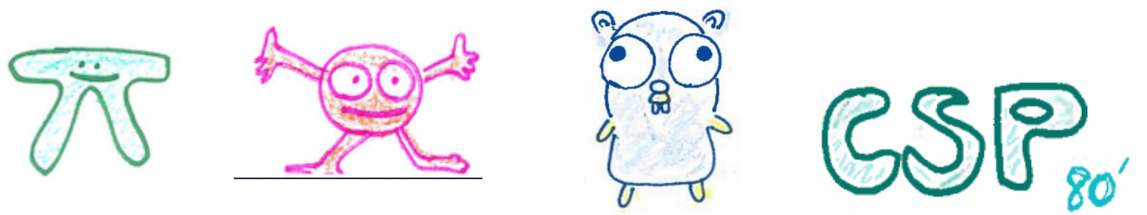
Actor Programming and Verifications

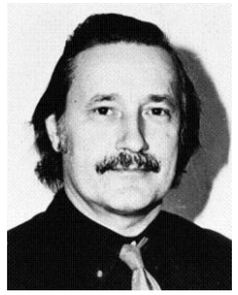
Specification-Guided Concurrent and Distributed Programming



Nobuko Yoshida, University of Oxford

AIMS Seminar, University of Oxford, 10th March 2023

AI and Strachey



1916-1975

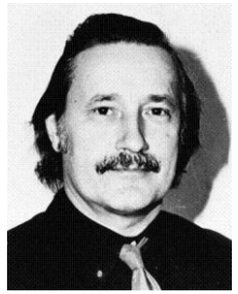
Computer Games, Literature and Music

A schoolmaster at Harrow

Specification-Guided Concurrent and Distributed Programming

Fundamental Concepts of
Programming Languages

AI and Strachey



1916-1975

Computer Games, Literature and Music

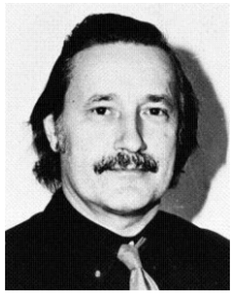
A schoolmaster at Harrow

Specification-Guided Concurrent and Distributed Programming

- **Christopher Strachey** (sequential computation)

Fundamental Concepts of Programming Languages

AI and Strachey



1916-1975

Computer Games, Literature and Music

A schoolmaster at Harrow

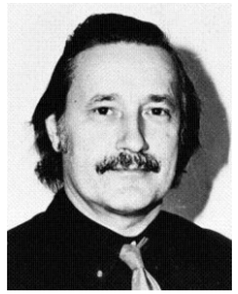
Specification-Guided Concurrent and Distributed Programming

- **Christopher Strachey** (sequential computation)

Fundamental Concepts of Programming Languages

  - ***Types*** = **abstract** and **digest computation (data types, polymorphism)**

AI and Strachey



1916-1975

Computer Games, Literature and Music

A schoolmaster at Harrow

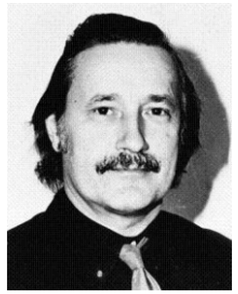
Specification-Guided Concurrent and Distributed Programming

- **Christopher Strachey** (sequential computation)

  Fundamental Concepts of Programming Languages

  - ***Types*** = **abstract** and **digest computation (data types, polymorphism)**
  - **Structured programming = High-level programming**

AI and Strachey



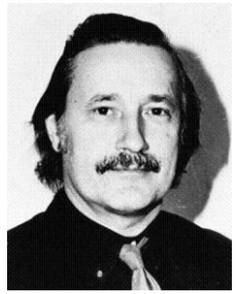
1916-1975

Computer Games, Literature and Music A schoolmaster at Harrow

Specification-Guided Concurrent and Distributed Programming

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

AI and Strachey



1916-1975

Computer Games, Literature and Music

A schoolmaster at Harrow

Specification-Guided Concurrent and Distributed Programming

- **Christopher Strachey** (sequential computation)

  Fundamental Concepts of Programming Languages

  - ***Types*** = **abstract** and **digest computation (data types, polymorphism)**
  - **Structured programming = High-level programming**
- **Session types** (concurrency & communication)
  - Structured programming = ***protocols***

Actor Models

1944-2020

Actor Models

- **Carl Hewitt (MIT)**

1944-2020

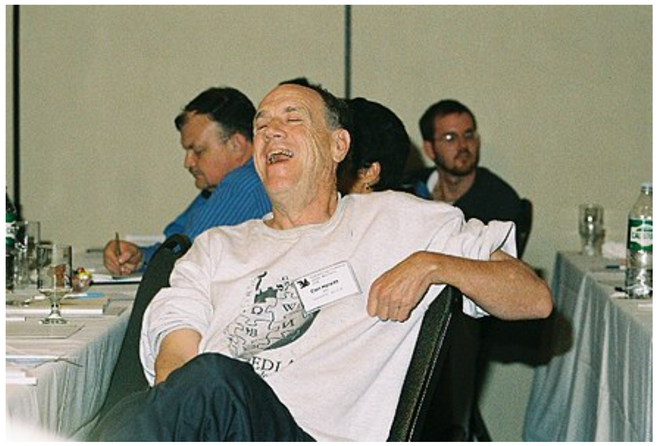
Actor Models

- Carl Hewitt (MIT)

1944-2020

  - **Mathematical Models** for Concurrent Computations

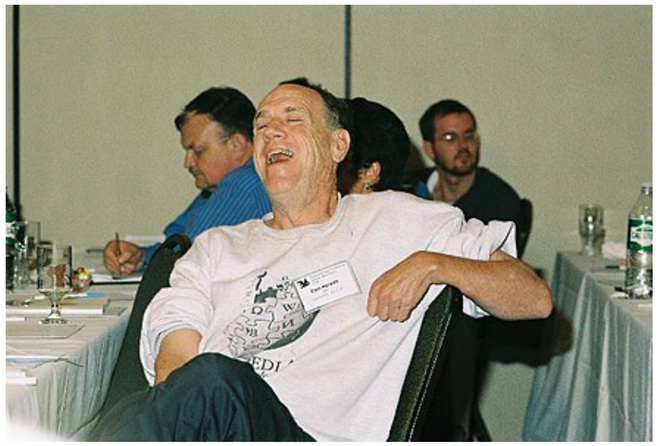
Actor Models



1944-2020

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

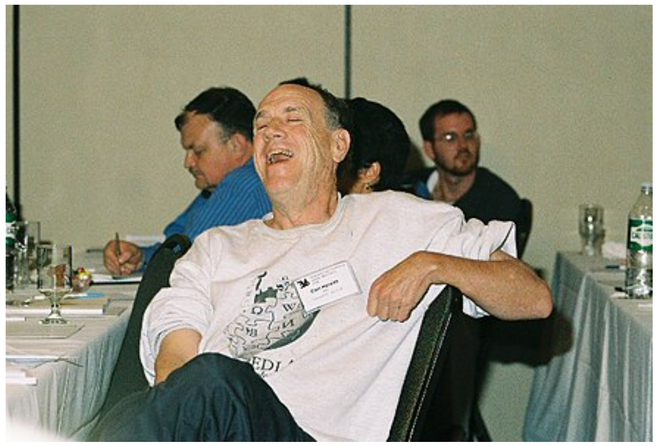
Actor Models



1944-2020

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

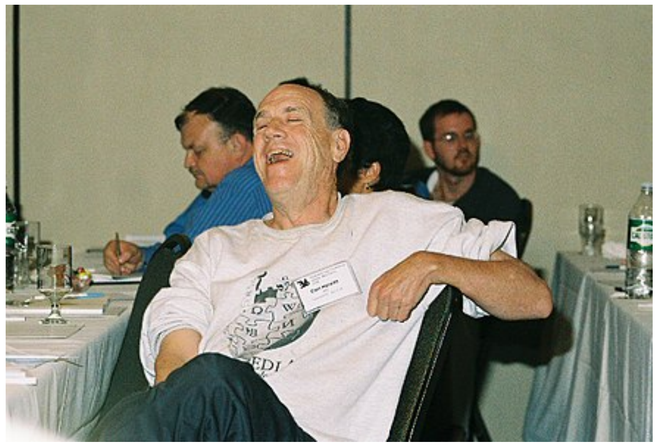
Actor Models



1944-2020

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

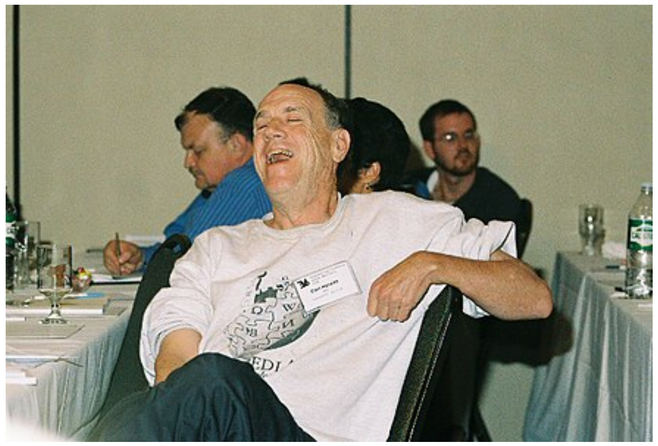
Actor Models



1944-2020

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

Actor Models



1944-2020

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

Communications are Ubiquitous

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

microservices

Problems: Ambiguity

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

RFC 821 August 1982
Simple Mail Transfer Protocol

TABLE OF CONTENTS

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]

The Go Gopher

Do not communicate by sharing memory; share memory by communicating

– Go Philosophy

Problems: Concurrency Bugs

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

deadlock

channel errors

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

The Go Gopher

Problems: Concurrency Bugs

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

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

Session Types

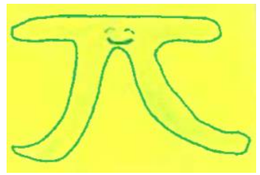
- Prevent concurrency bugs.
- Can abstract, implement and manage communications as **Protocols.**
- **Clean, Cheap** and **Retrofittable.**

Why Session Types, Why Now?

Significant academic and industry interests via fundamental breakthroughs

Milner, Honda, NY

Binary Session Types — ESOP'98



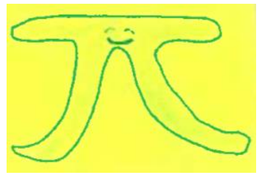
↓

Joined W3C Standardization — 2002

Why Session Types, Why Now?

Significant academic and industry interests via fundamental breakthroughs

Binary Session Types — ESOP'98



↓

Joined W3C Standardization — 2002

↓

Multiparty Session Types — POPL' 08

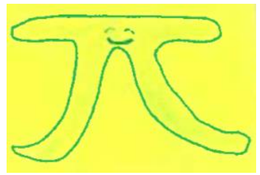
Choreography Description Language (W3C)

```
package HelloWorld {
    roleType YouRole, WorldRole;
    participantType You{YouRole}, World{WorldRole};
    relationshipType YouWorldRel between YouRole and WorldRole;
    channelType WorldChannelType with roleType WorldRole;

    choreography Main {
        WorldChannelType worldChannel;

        interaction operation=hello from=YouRole to=WorldRole
                relationship=YouWorldRel channel=worldChannel {
            request messageType=Hello;
        }
    }
}
```

Why Session Types, Why Now?

Significant academic and industry interests via fundamental breakthroughs

Binary Session Types — ESOP'98



↓

Joined W3C Standardization — 2002

↓

Multiparty Session Types — POPL' 08

↓

largest open source company in the world

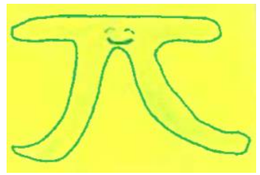
Red Hat

Scribble

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

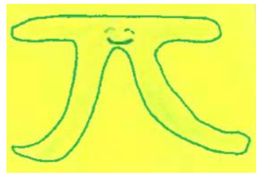
25M US$ large cyber infrastructures

middleware standardisations

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







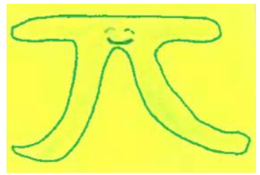



Why Session Types, Why Now?

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

What is a concept of *Types* in Programming?

- Types can avoid runtime error

Boolean, Natural Number

100 x 200

100 x true

What is a concept of *Types* in Programming?

- Types can avoid runtime error

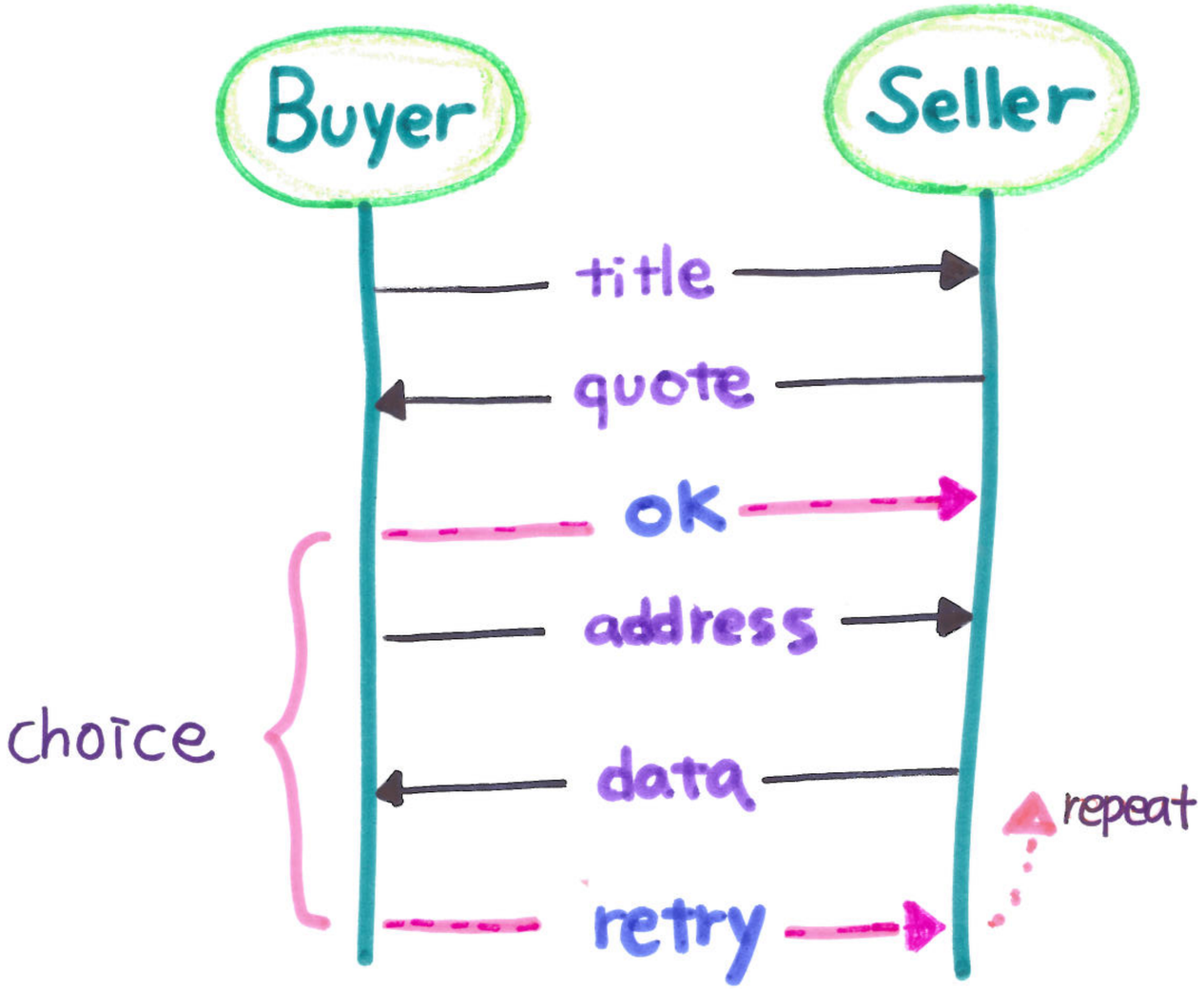
Boolean, Natural Number (Data Types)

100 x 200 **Correct**

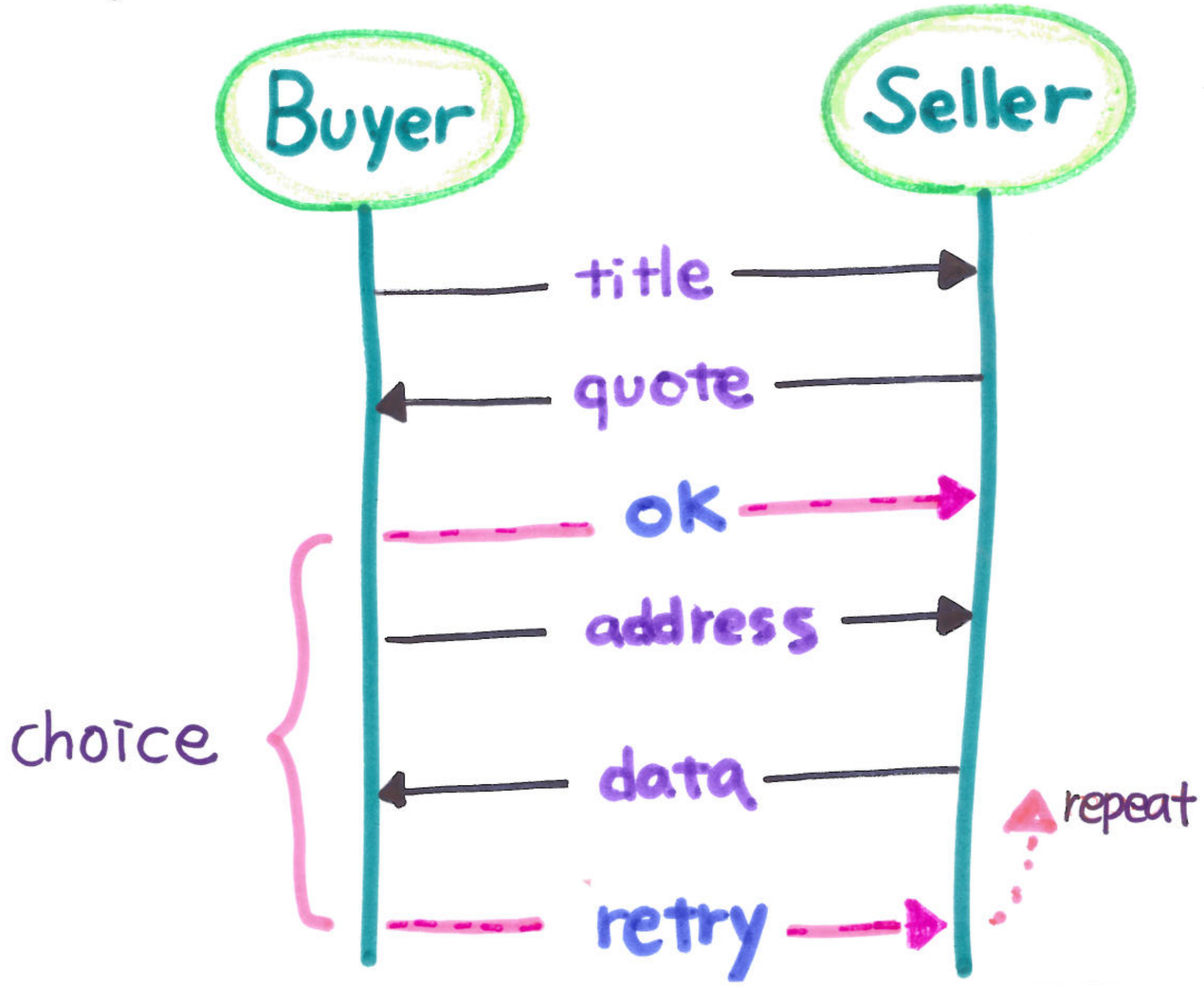
100 x true **Wrong**

Types for *Protocol*?

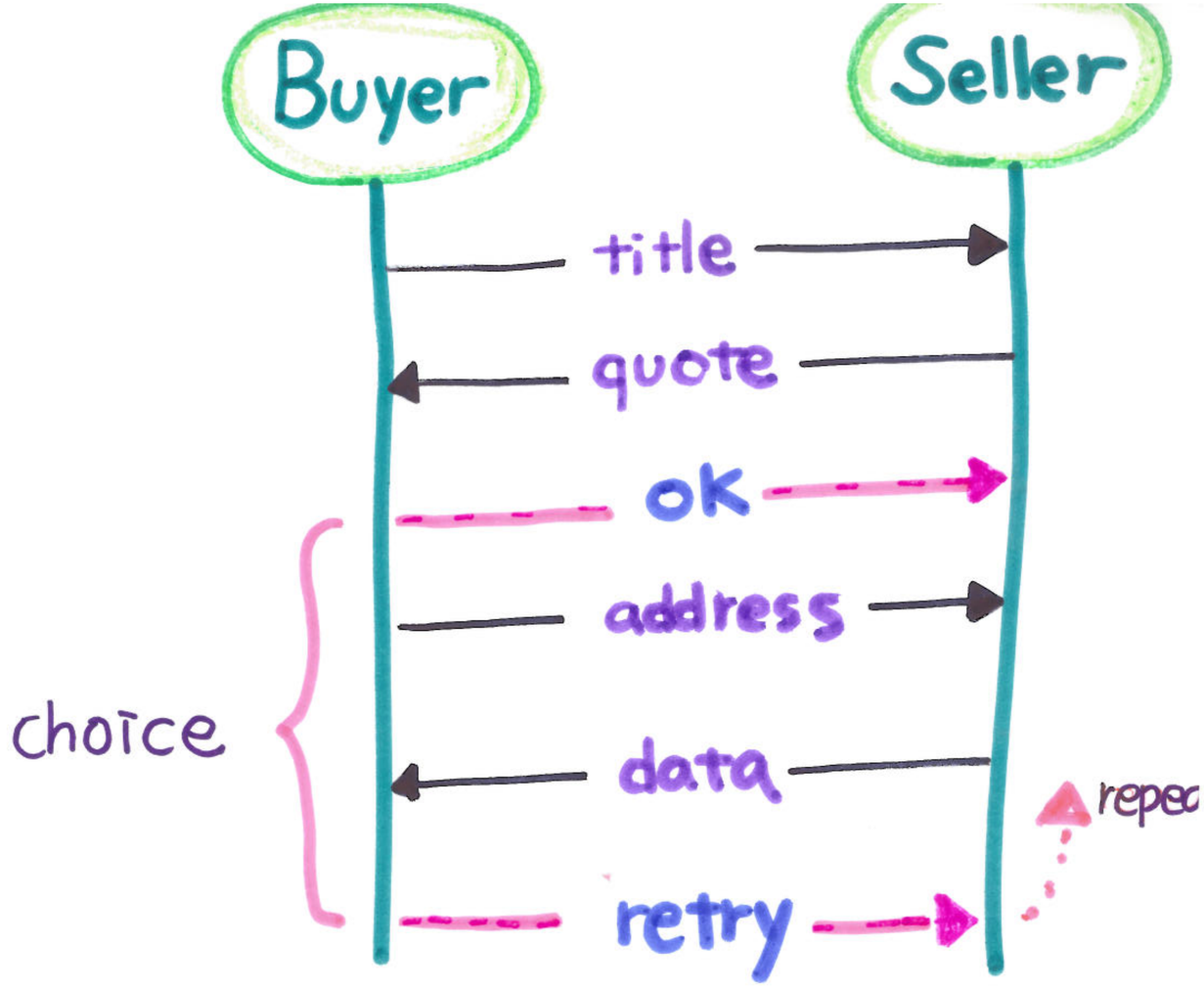
Binary Session Types: Buyer - Seller Protocol



Binary Session Types: Buyer - Seller Protocol



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



P has T

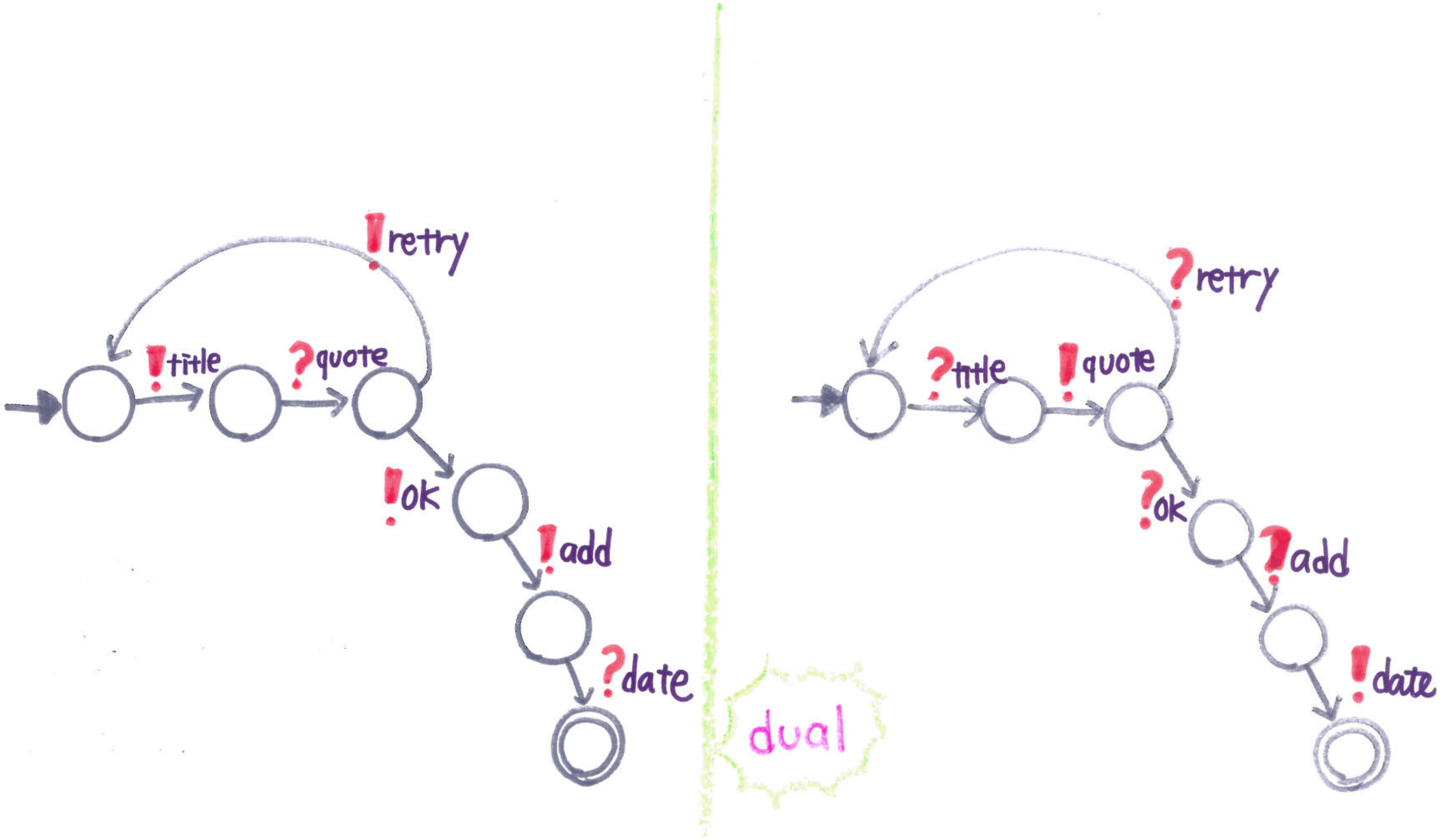
Q has $\overline{T}$ dual

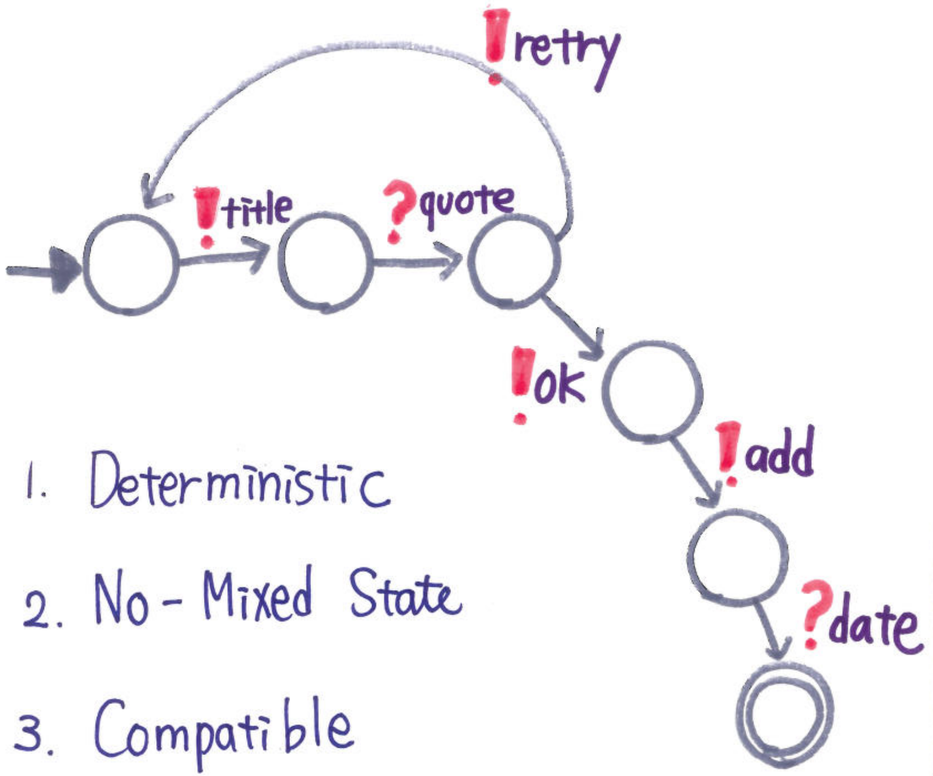
P | Q typable

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

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

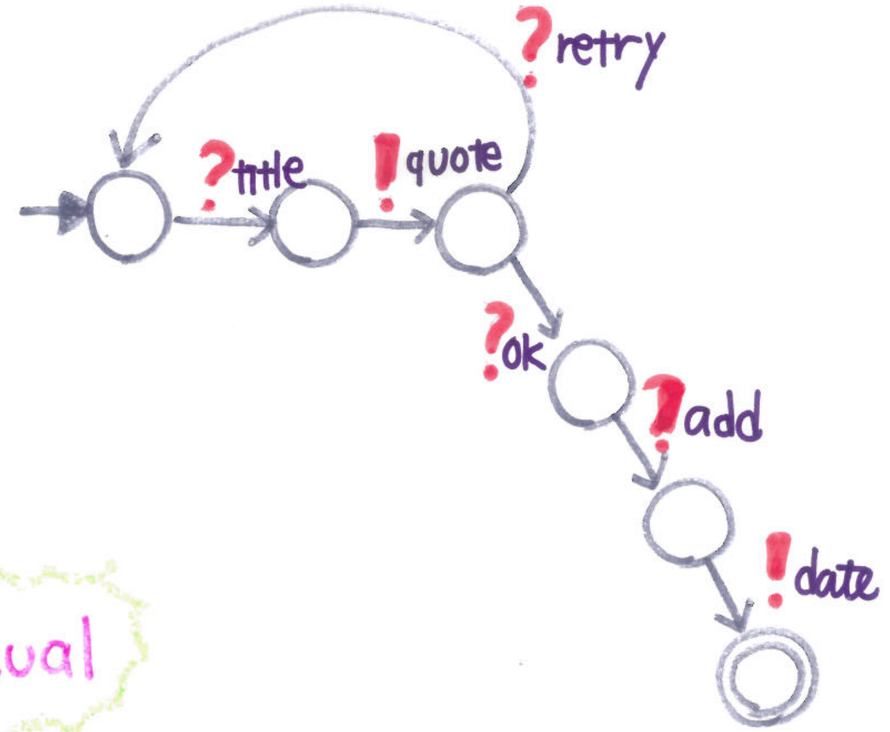
Communicating Automata [1980s]





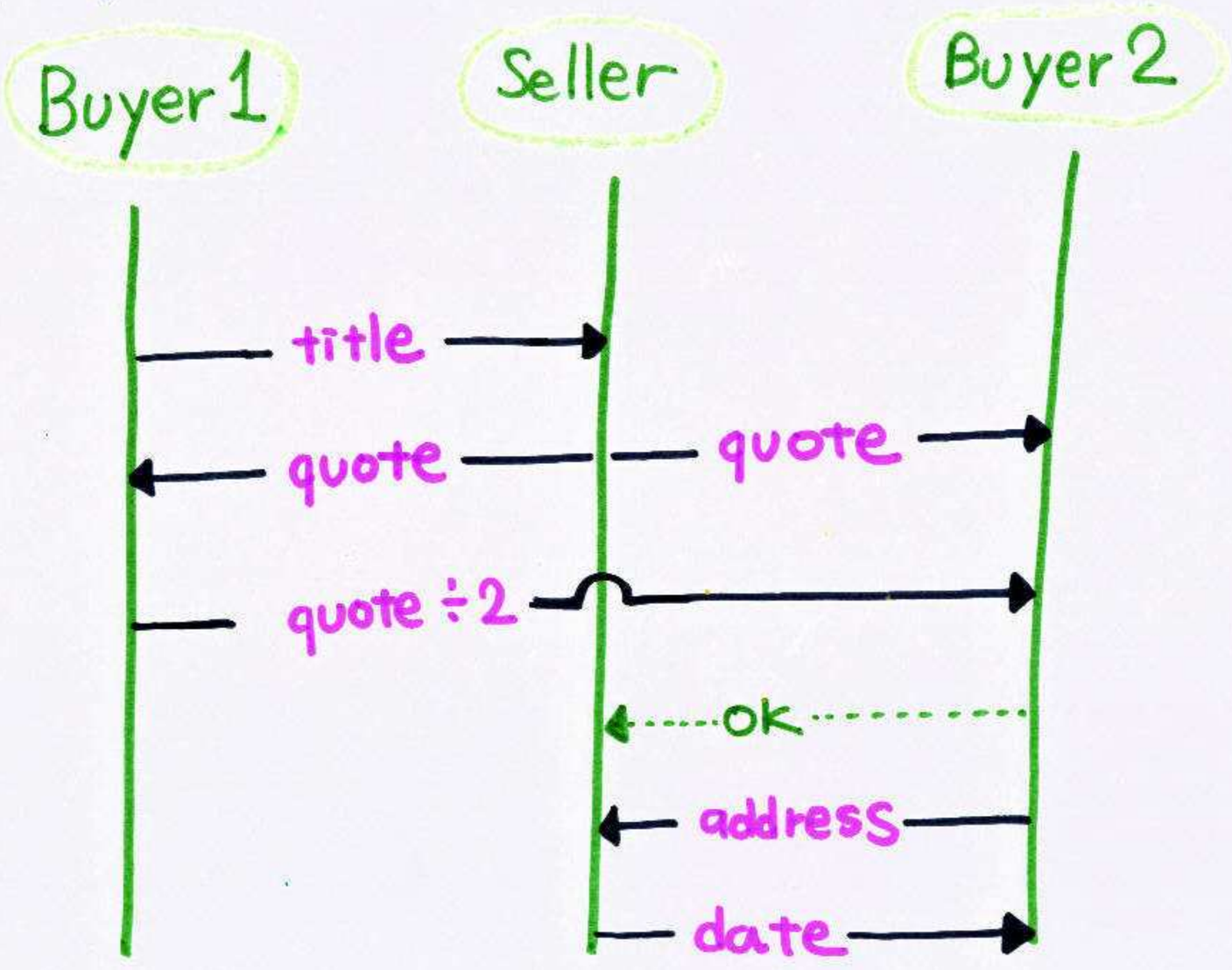
1. Deterministic
2. No-Mixed State
3. Compatible

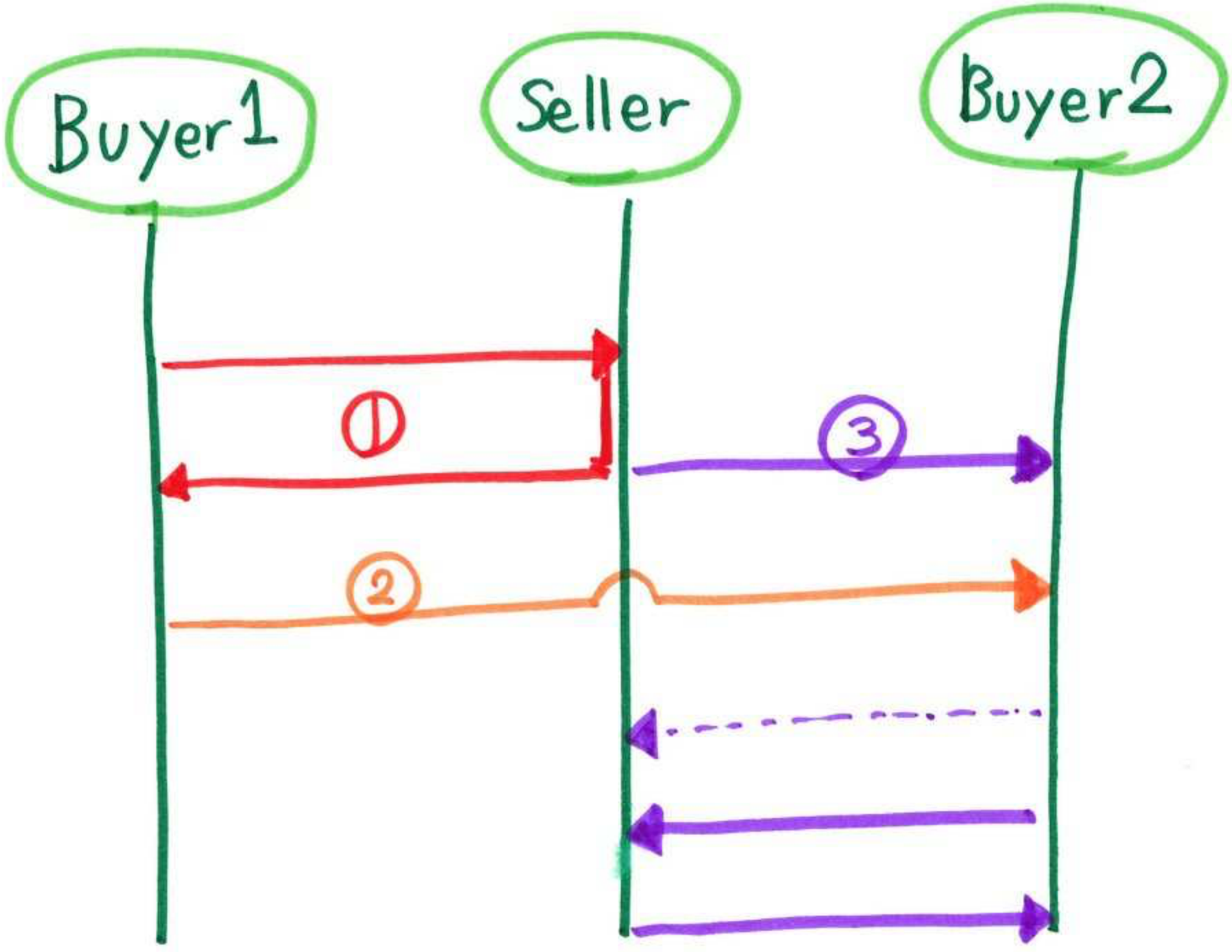
dual

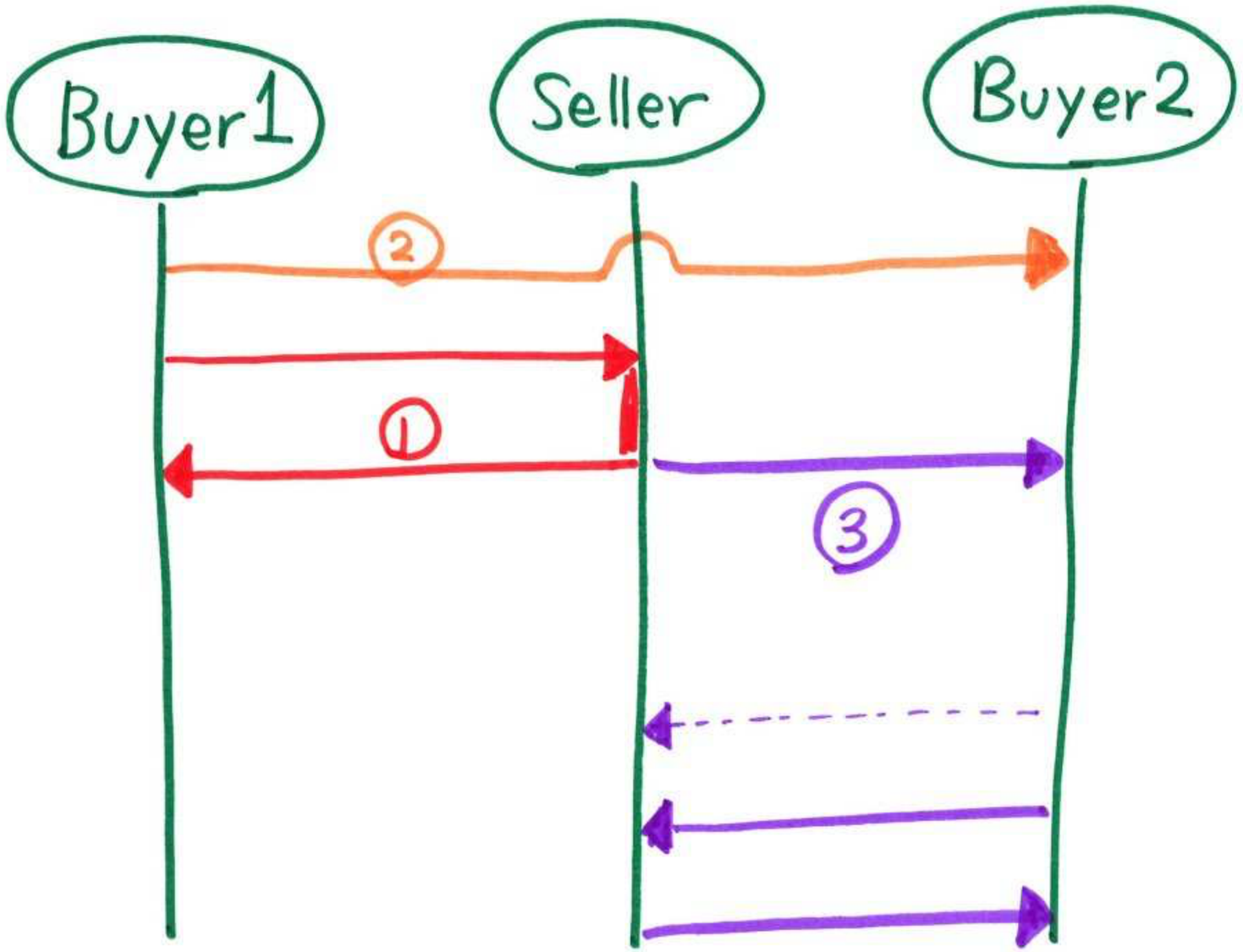


[Gouda et al 1986] Two compatible machines without mixed states which are deterministic satisfy deadlock-freedom.

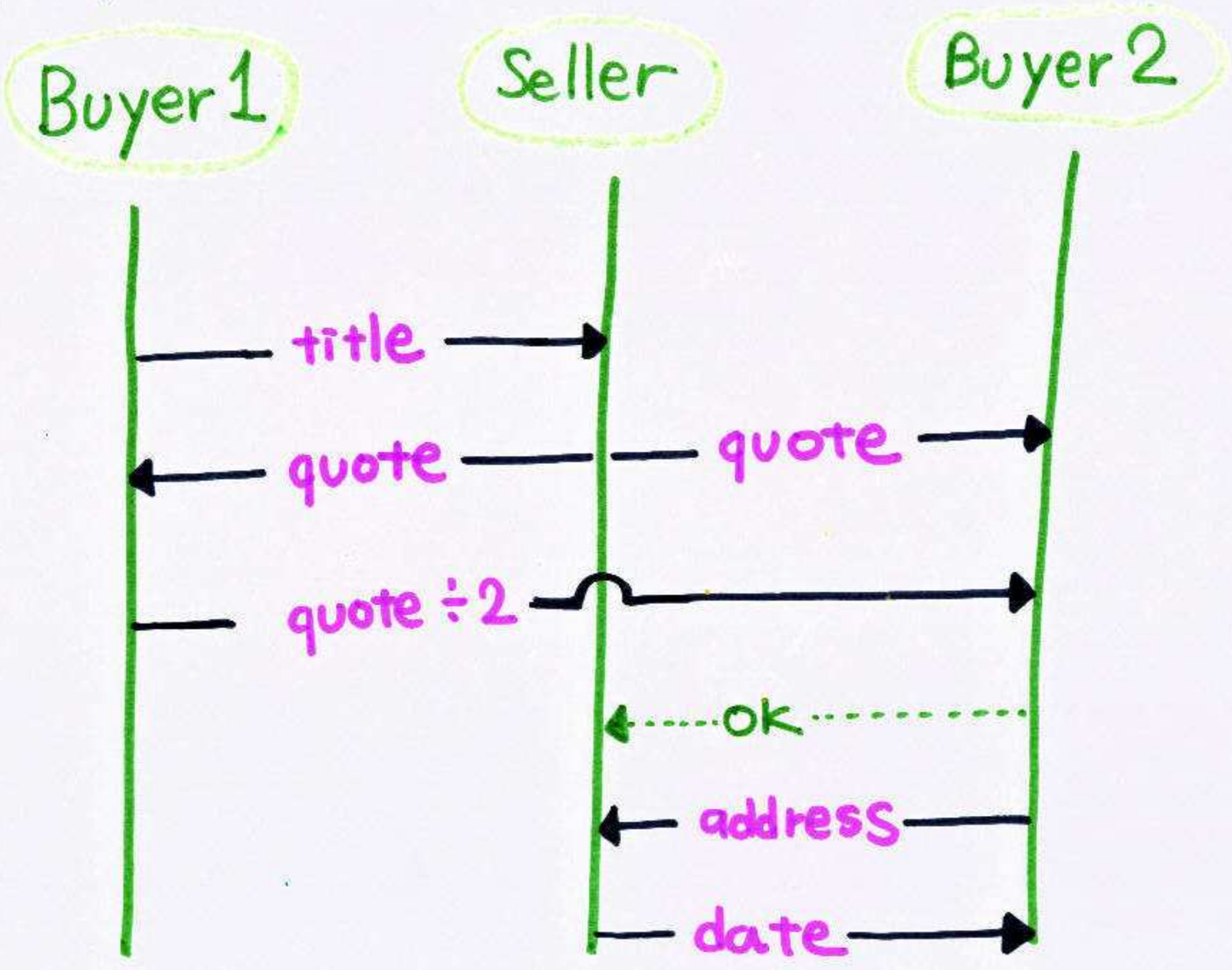
Multiparty Session Types

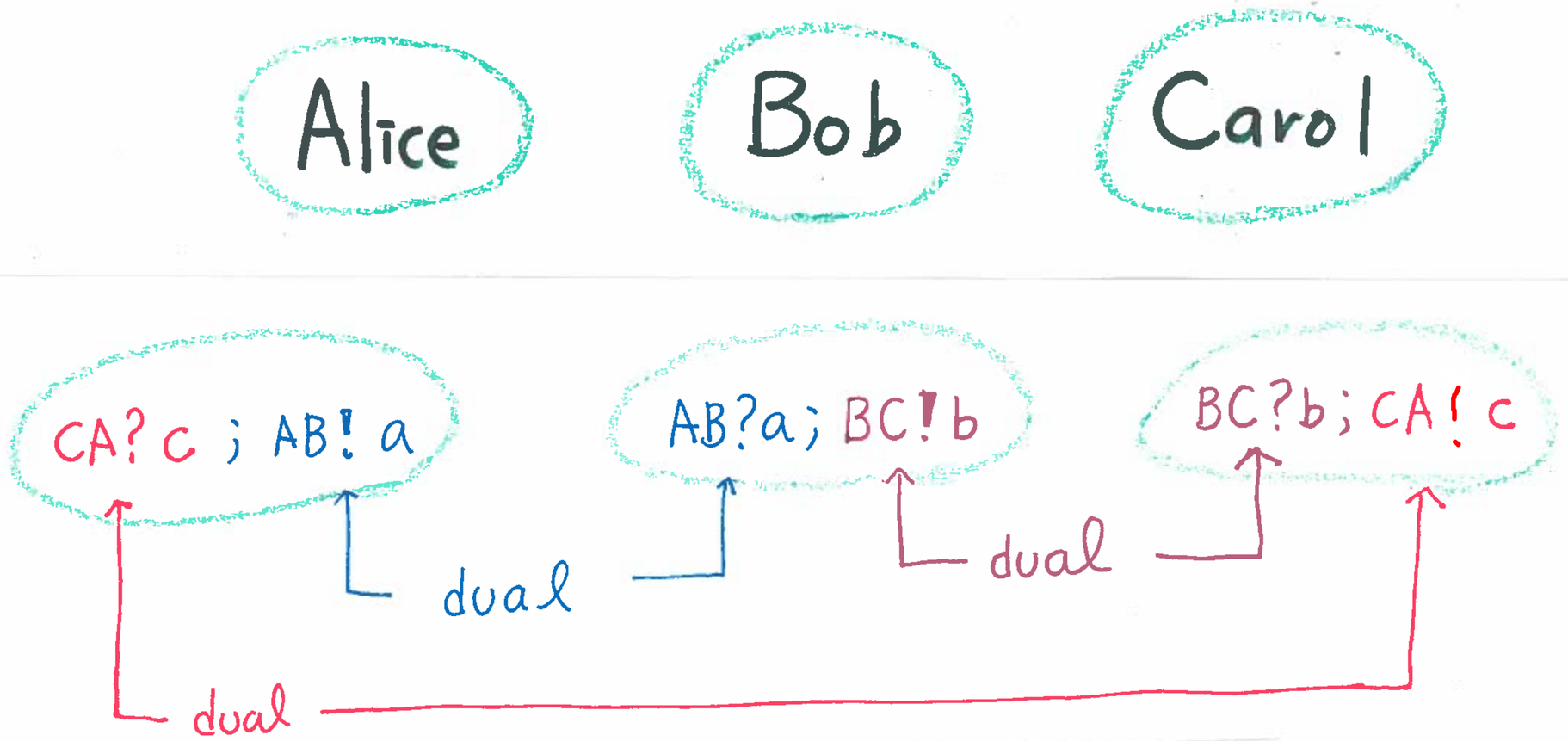



Buyer1
Seller
Buyer2
1
3
2


Buyer1
Seller
Buyer2
2
1
3

Multiparty Session Types



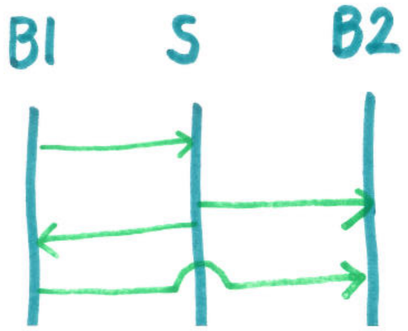


3 dual pairs

If you use binary Session Types ....

Deadlock!

Multiparty Session Types [Honda, Yoshida, Carbone 2008]



G

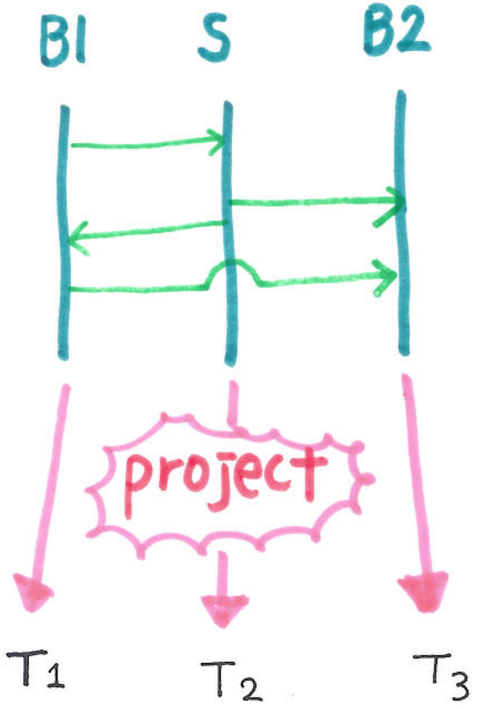
B1 → S Int.

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

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

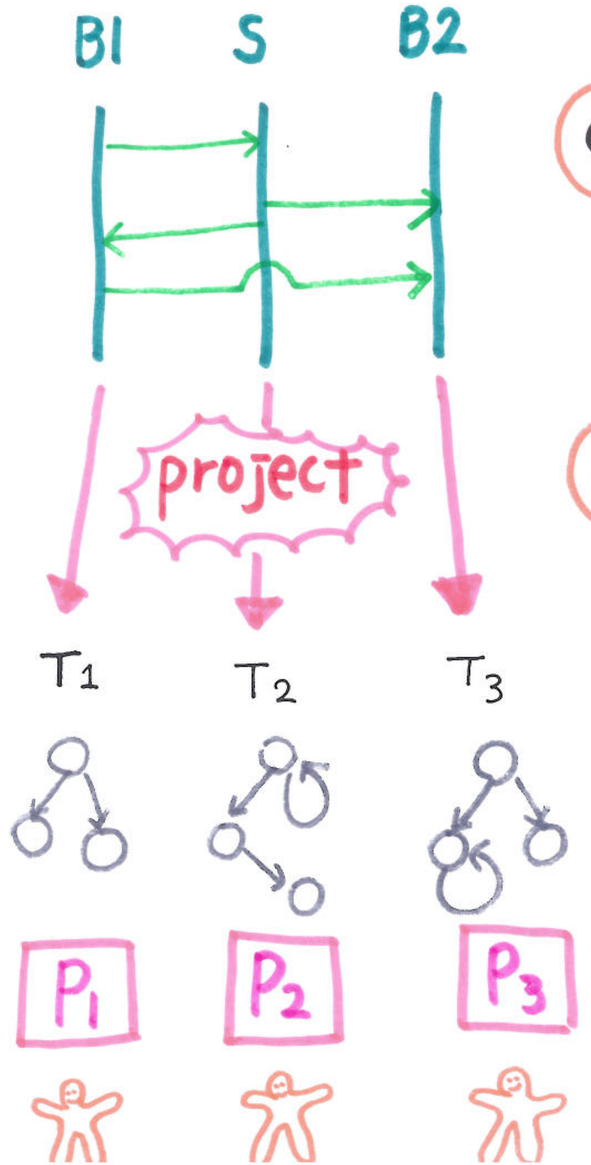
STEP 1

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

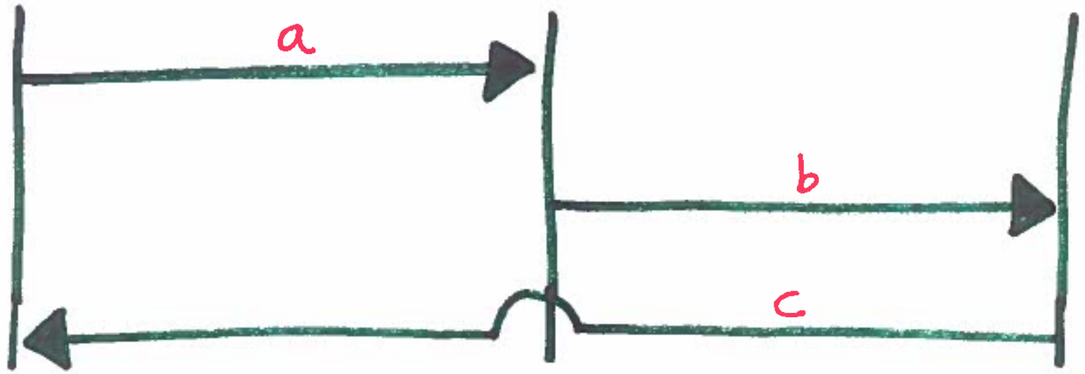
STEP 2
Project to Local Type

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

Alice

Bob

Carol



Global Type

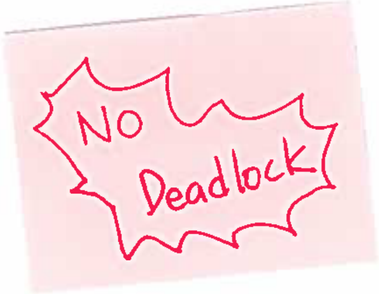
Projection

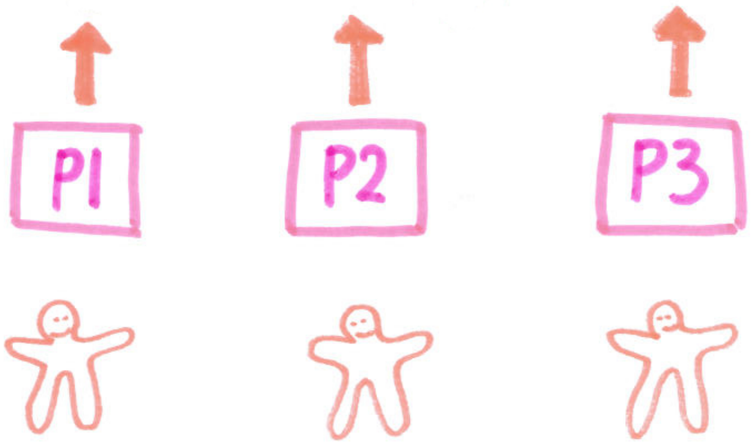
Alice AB!a;CA?c

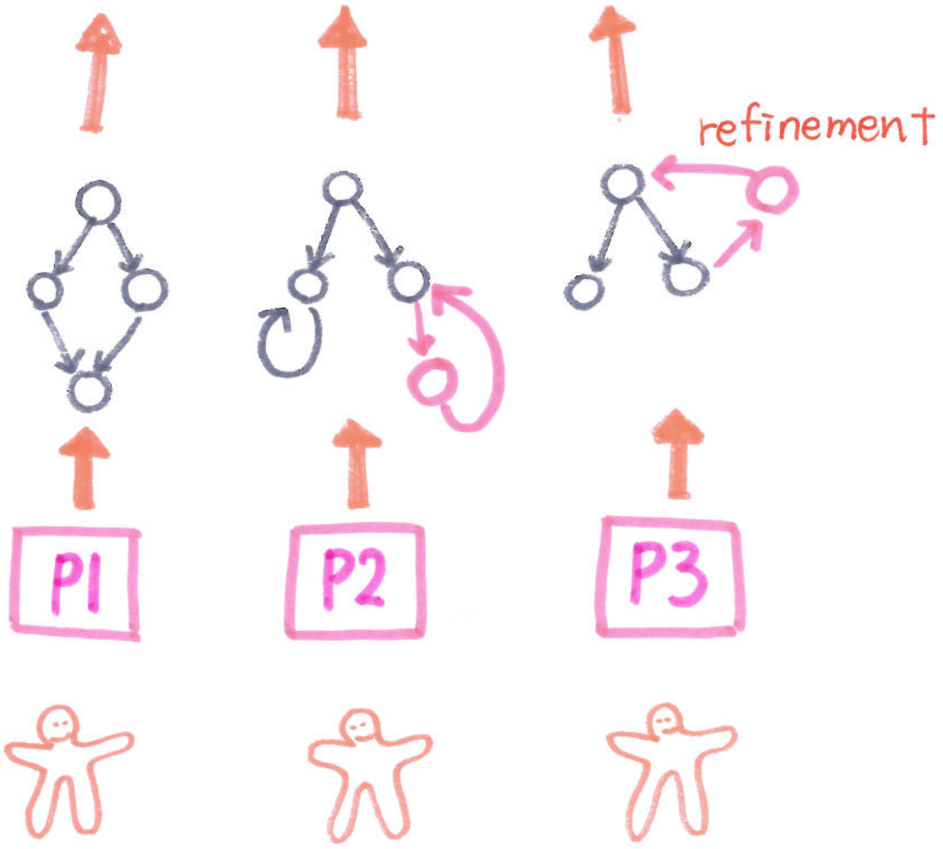
Bob AB?a;BC!b

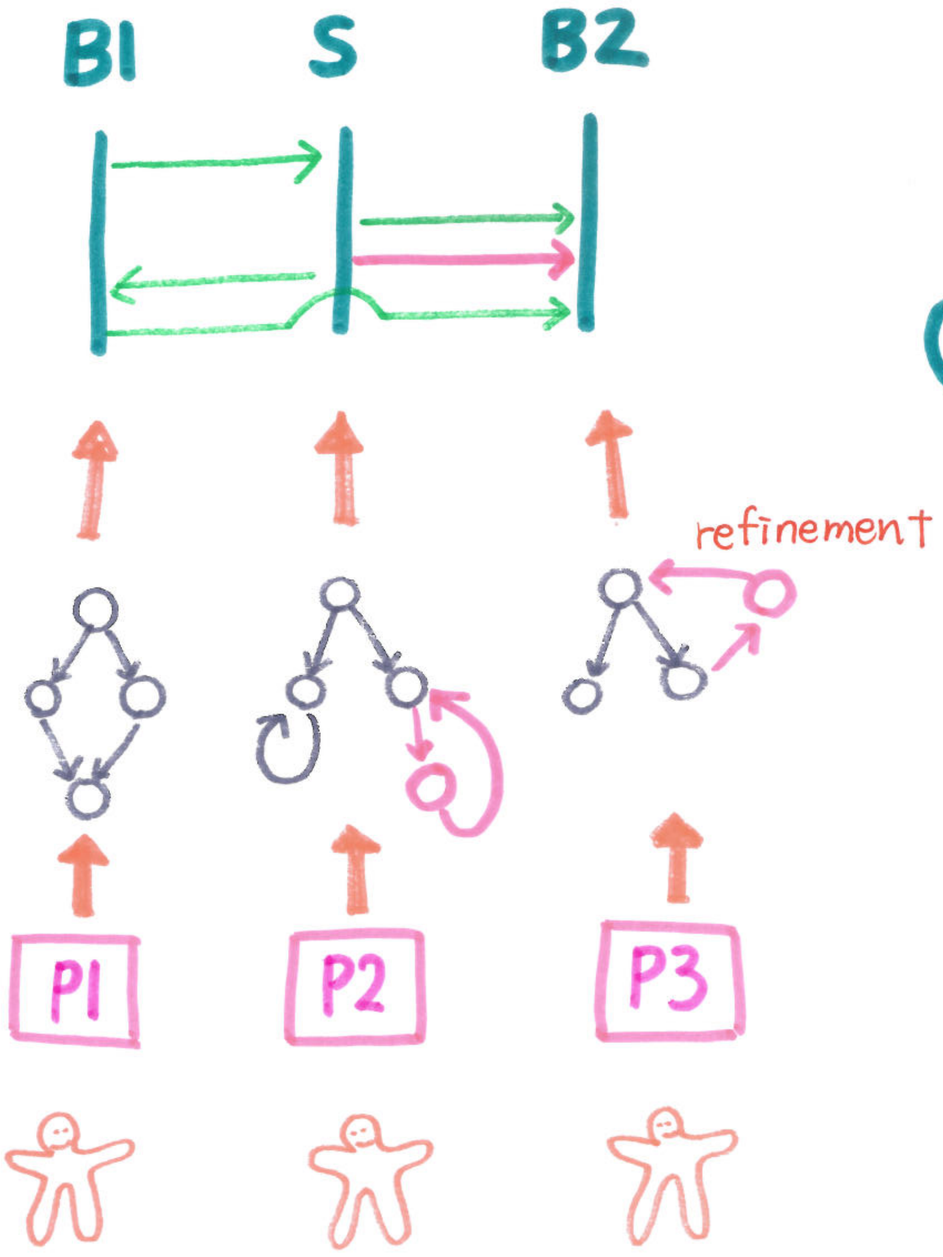
Carol BC?b;CA!c;

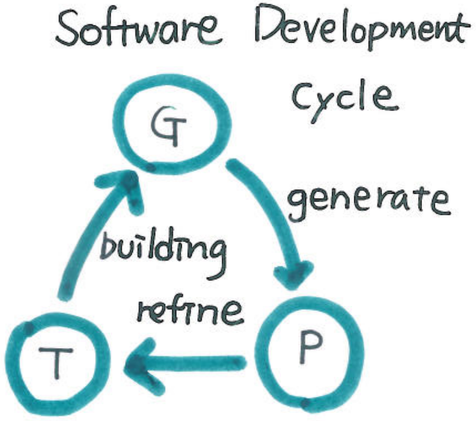
LOCAL TYPES




P1
P2
P3


refinement
P1
P2
P3



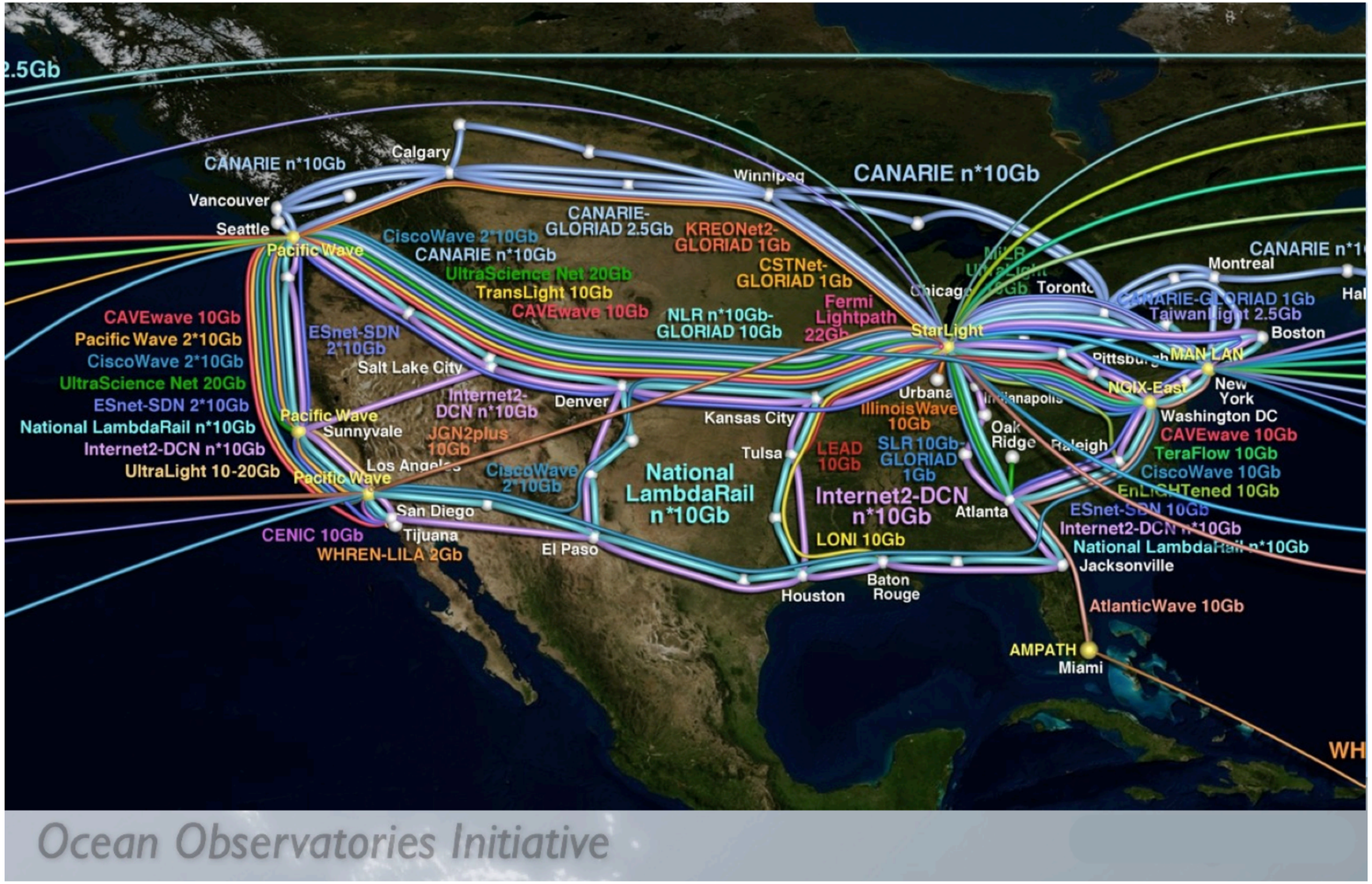


- Optimisation
- refinement
- Inference
- Testing

Mobility Reading Group

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

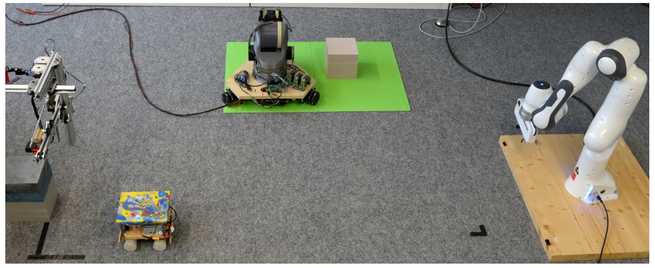
Some Applications on Multiparty Session Types



Distributed Tracing

OpenTelemetry

Robotics

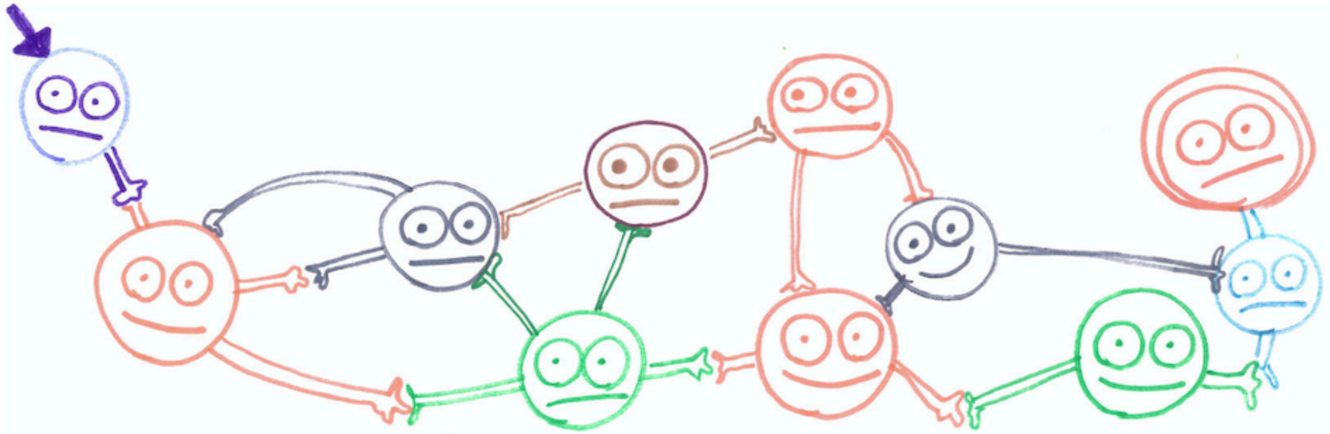


Mechanisation

Zooid

Optimising Asynchronous Communication in Rust

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



Zak Cutner, NY and Martin Vassor

Introduction

Rust Language

- Modern systems language focussed on safety and performance

Introduction

Rust Language

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

Introduction

Rust Language

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

Introduction

Rust Language

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

Ring Protocol

Example

Global Type

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

Ring Protocol

Example

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

Ring Protocol

Example

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

Ring Protocol

Example

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

Ring Protocol

Example

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

Ring Protocol

Example

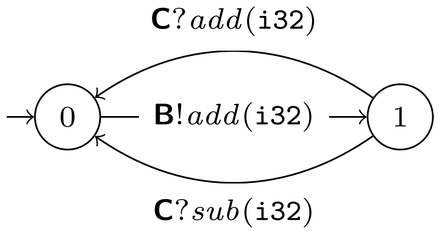
$$\mathsf{G} = \mu \mathbf{t}.\mathbf{A} \to \mathbf{B} : \left\{ add(\texttt{i32}).\mathbf{B} \to \mathbf{C} : \left\{ \begin{array}{l} add(\texttt{i32}).\mathbf{C} \to \mathbf{A} : \{add(\texttt{i32}).\mathbf{t}\} \\ sub(\texttt{i32}).\mathbf{C} \to \mathbf{A} : \{sub(\texttt{i32}).\mathbf{t}\} \end{array}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

Ring Protocol

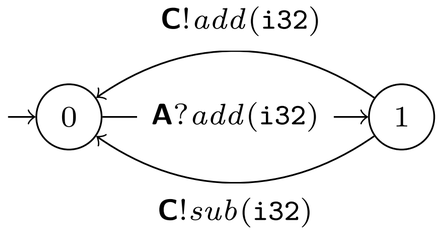
Example

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

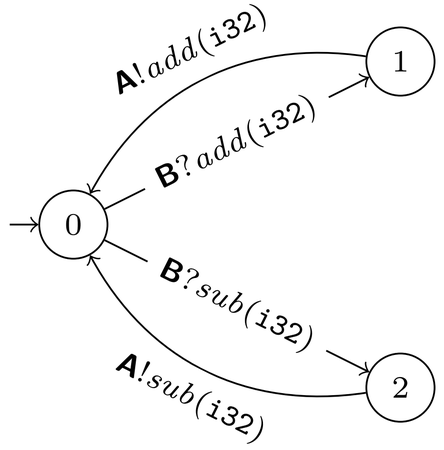
PROJECTION



PROJECTION



PROJECTION

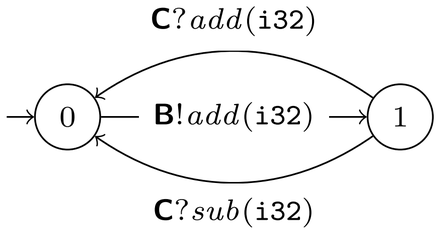


Ring Protocol

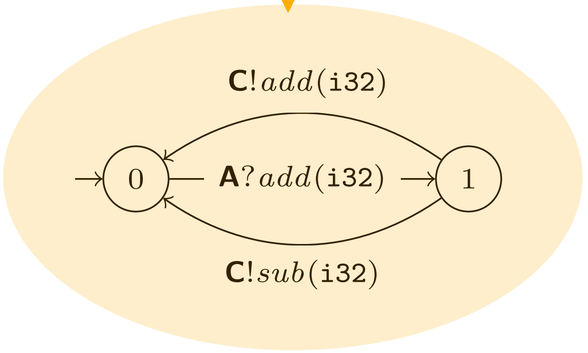
Example

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

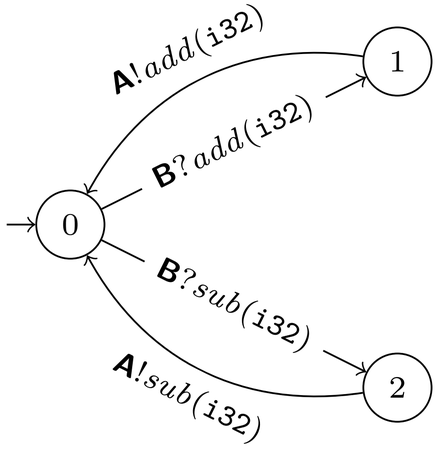
PROJECTION



PROJECTION



PROJECTION



Challenge

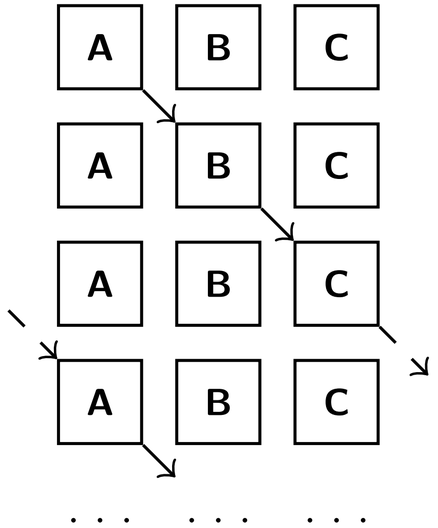
Asynchronous Orderings

- Global types are inherently synchronous

Challenge

Asynchronous Orderings

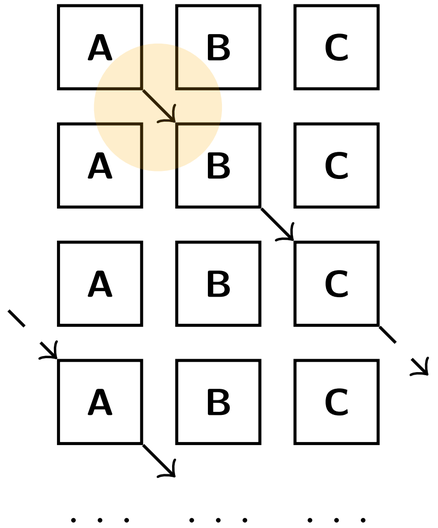
- Global types are inherently synchronous
  - Projection provides only one possible ordering



Challenge

Asynchronous Orderings

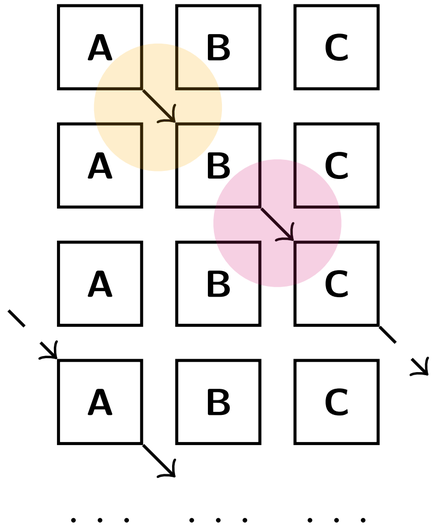
- Global types are inherently synchronous
  - Projection provides only one possible ordering



Challenge

Asynchronous Orderings

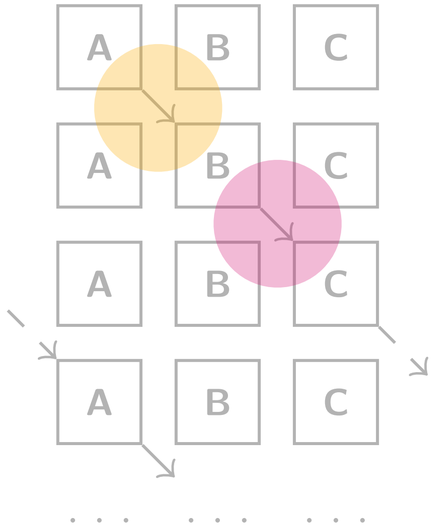
- Global types are inherently synchronous
  - Projection provides only one possible ordering

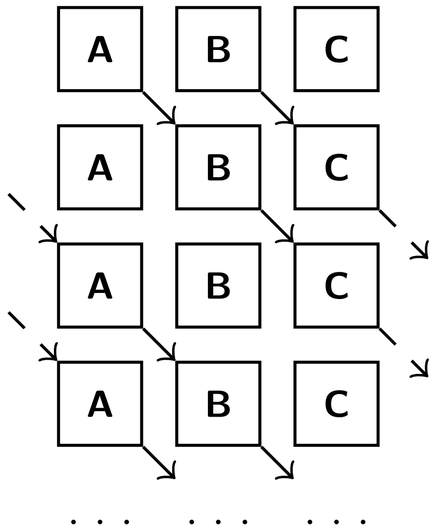


Challenge

Asynchronous Orderings

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

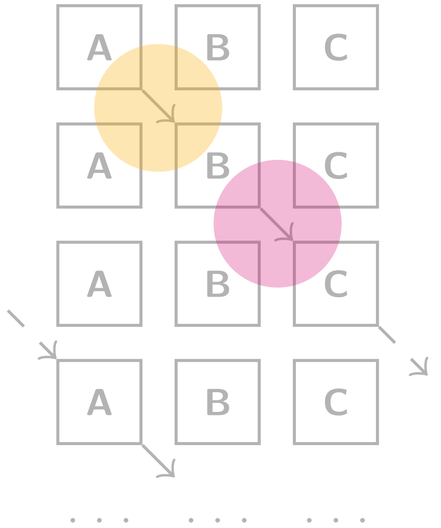


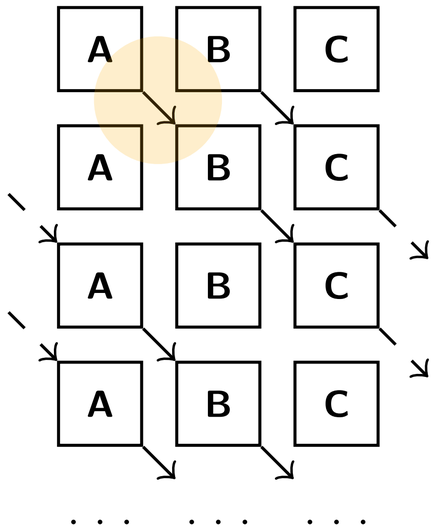


Challenge

Asynchronous Orderings

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

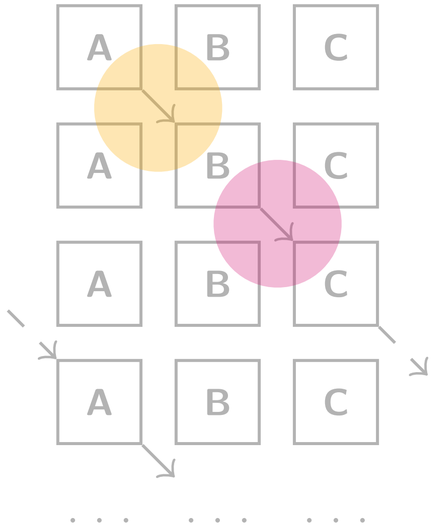


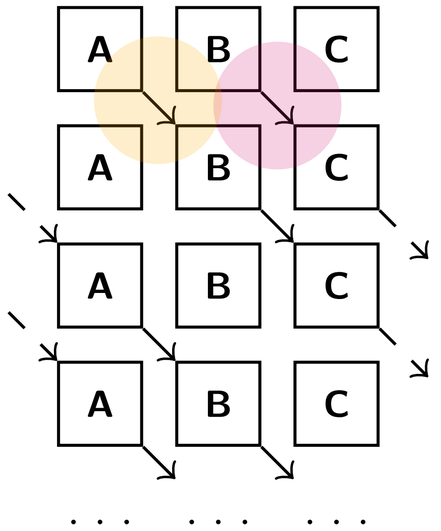


Challenge

Asynchronous Orderings

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

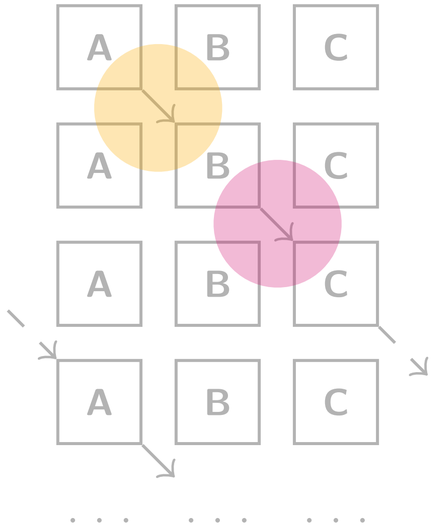


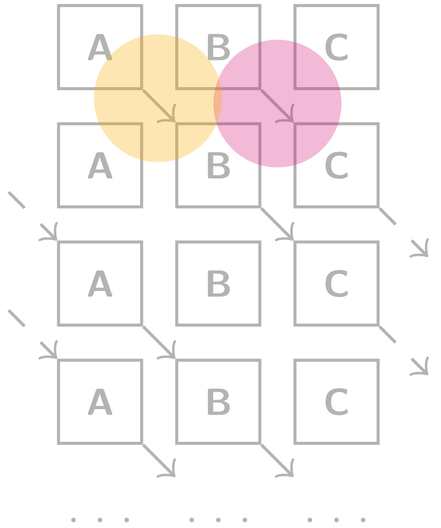


Challenge

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

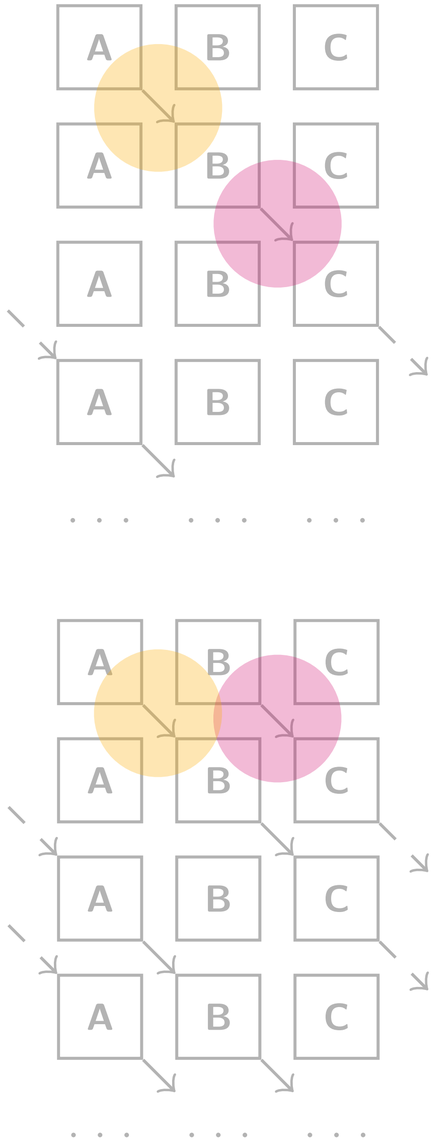




Challenge

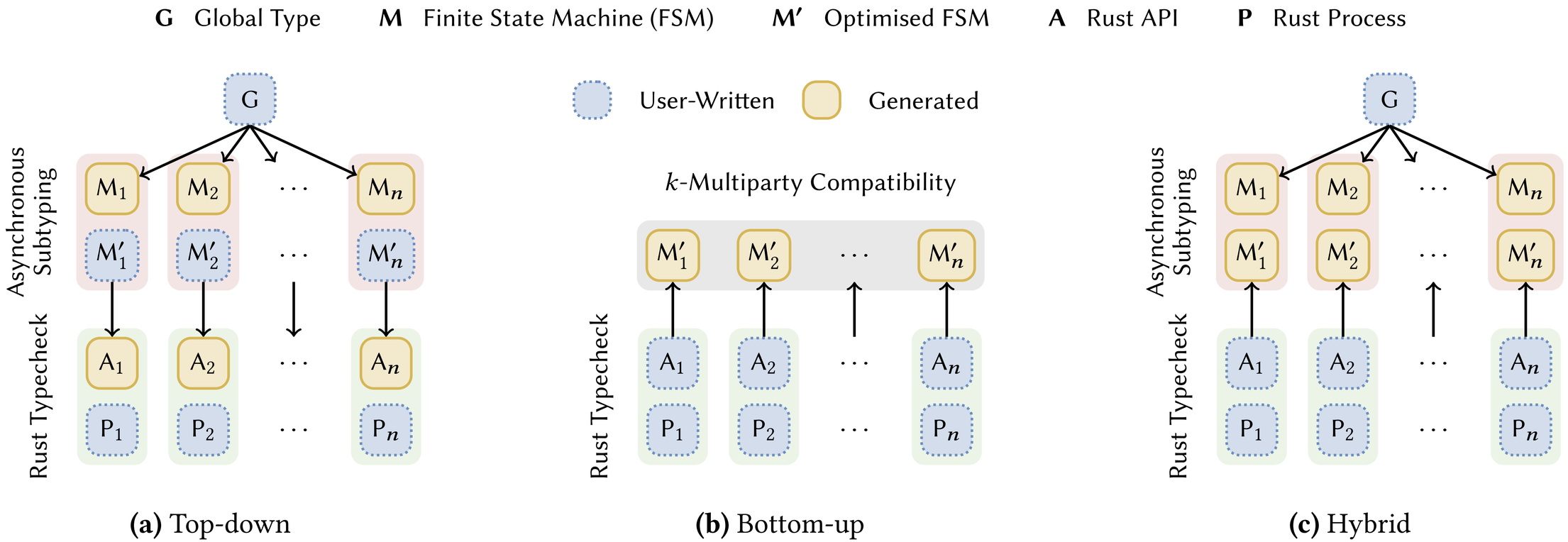
Asynchronous Orderings

- 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



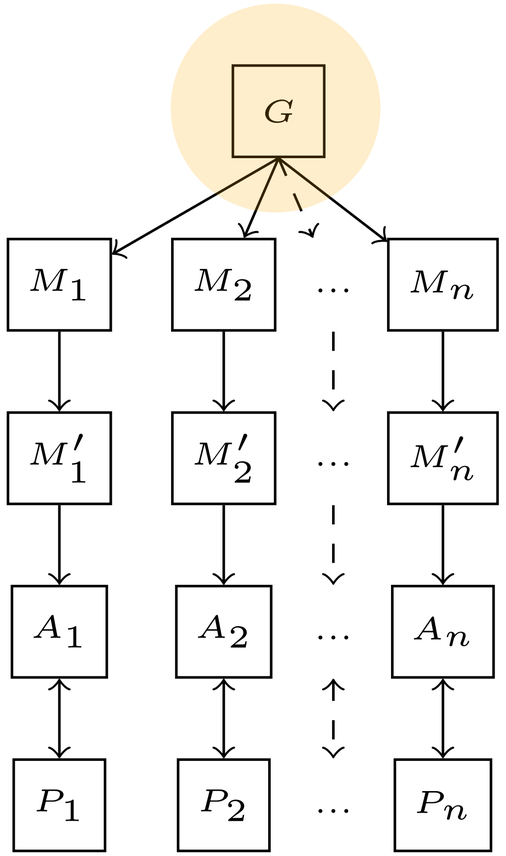
(a) Top-down

(b) Bottom-up

(c) Hybrid

Workflow

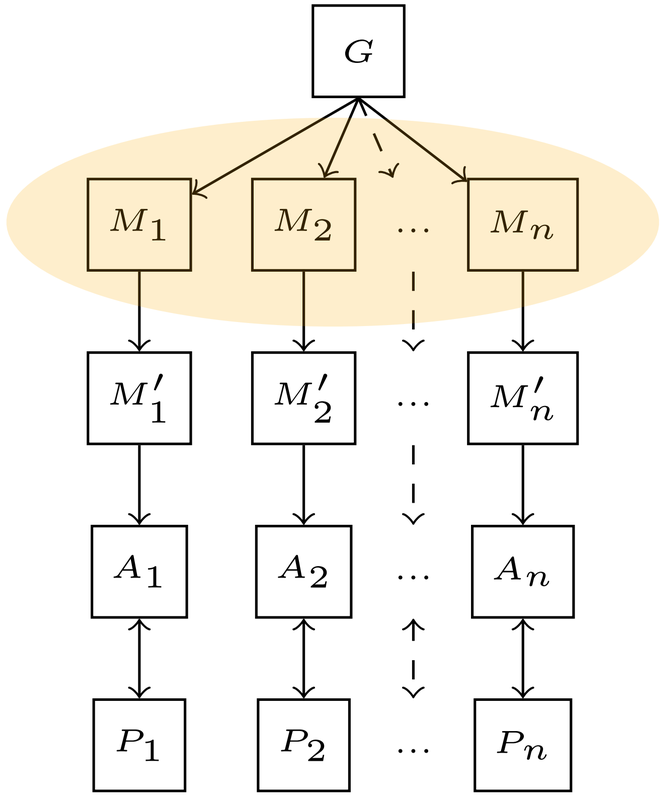
Top-Down Approach



Global Type

Workflow

Top-Down Approach

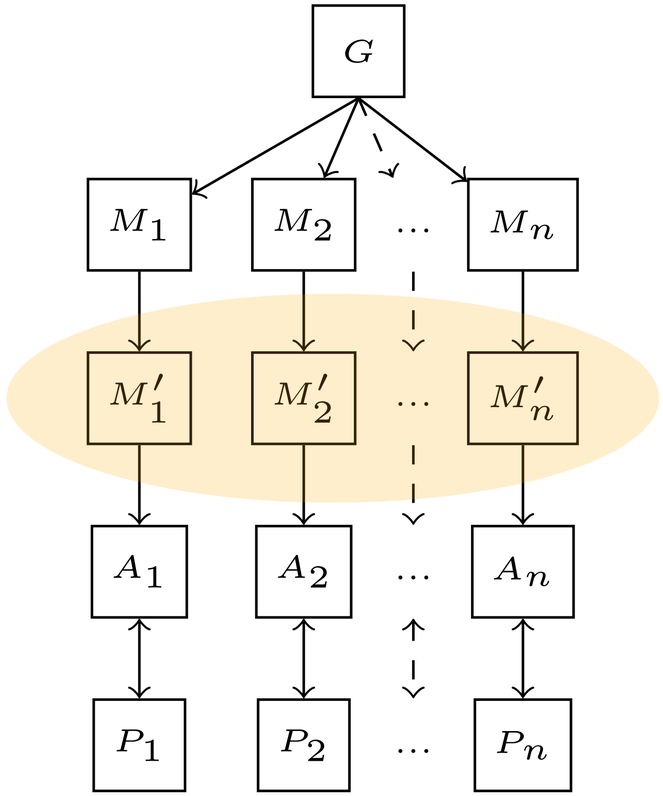


Global Type

Projected FSM

Workflow

Top-Down Approach



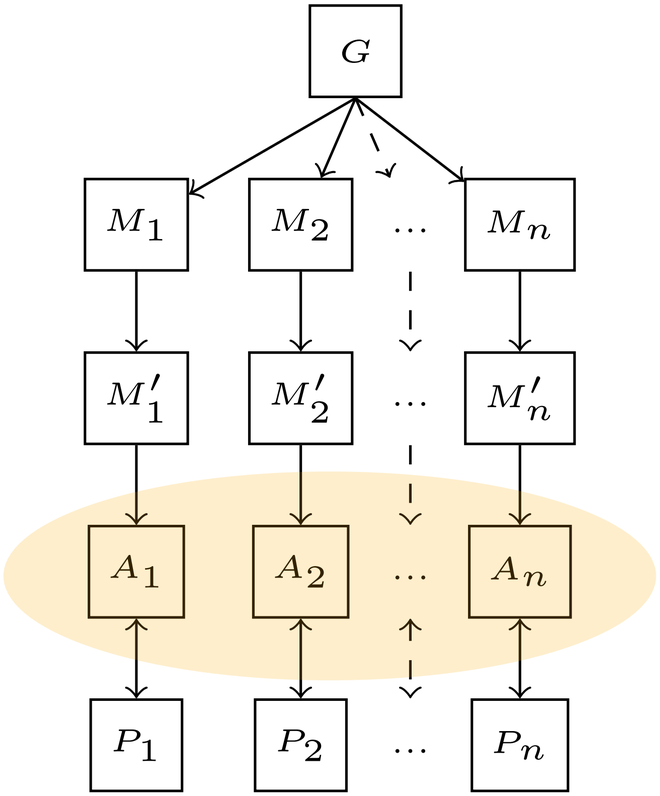
Global Type

Projected FSM

Optimised FSM

Workflow

Top-Down Approach



Global Type

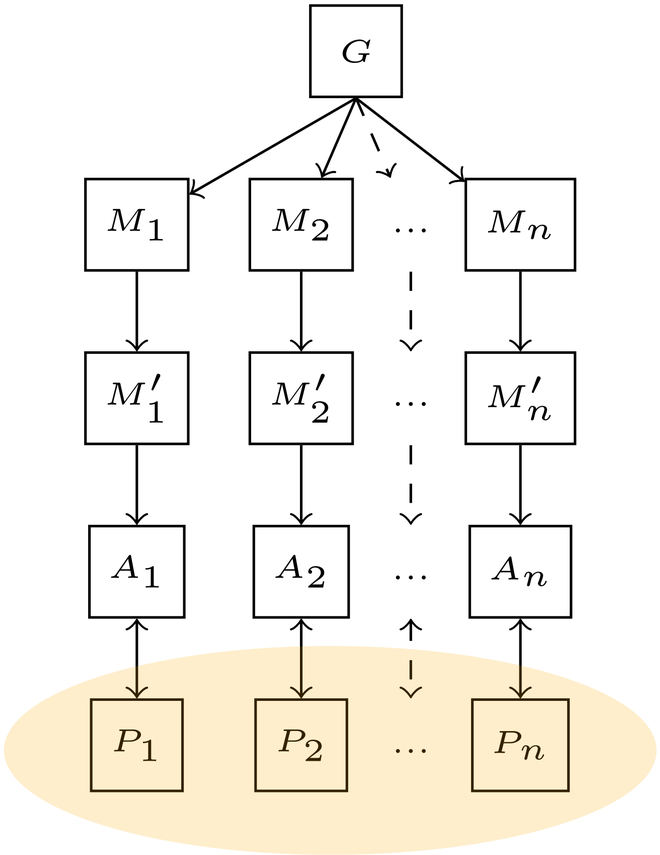
Projected FSM

Optimised FSM

Rust API

Workflow

Top-Down Approach



Global Type

Projected FSM

Optimised FSM

Rust API

Rust Implementation

Ring Protocol

Example

Global Type

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

Ring Protocol

Example

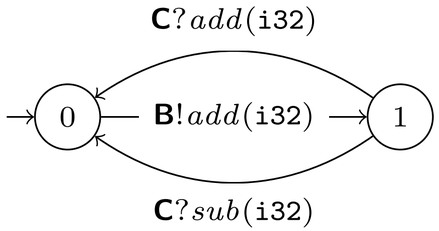
$$\mathsf{G} = \mu \mathbf{t}.\mathbf{A} \to \mathbf{B} : \left\{ add(\texttt{i32}).\mathbf{B} \to \mathbf{C} : \left\{ \begin{array}{l} add(\texttt{i32}).\mathbf{C} \to \mathbf{A} : \{add(\texttt{i32}).\mathbf{t}\} \\ sub(\texttt{i32}).\mathbf{C} \to \mathbf{A} : \{sub(\texttt{i32}).\mathbf{t}\} \end{array}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

Ring Protocol

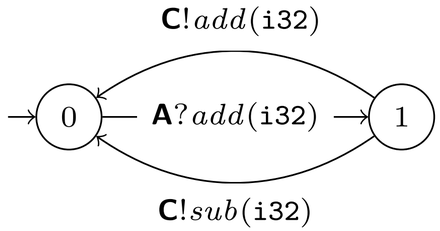
Example

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

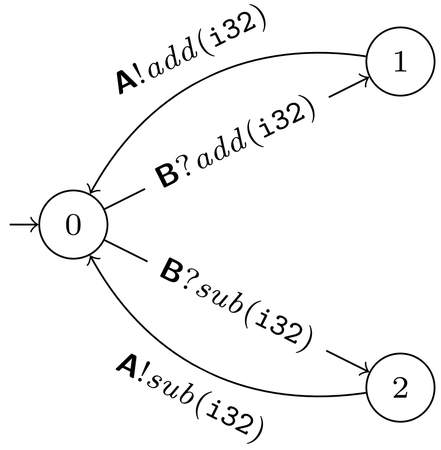
PROJECTION



PROJECTION

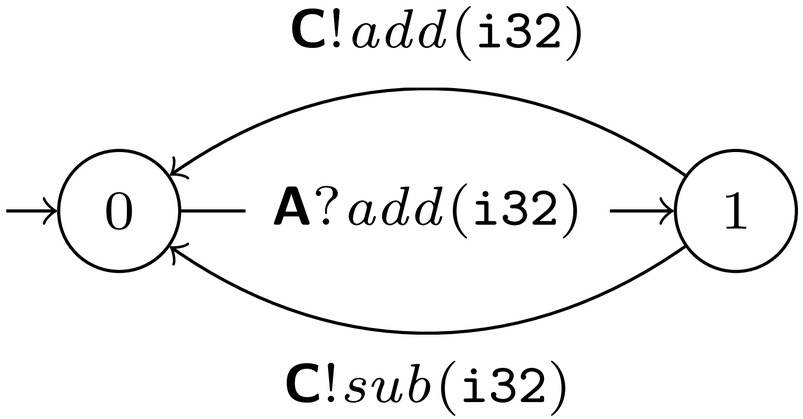


PROJECTION



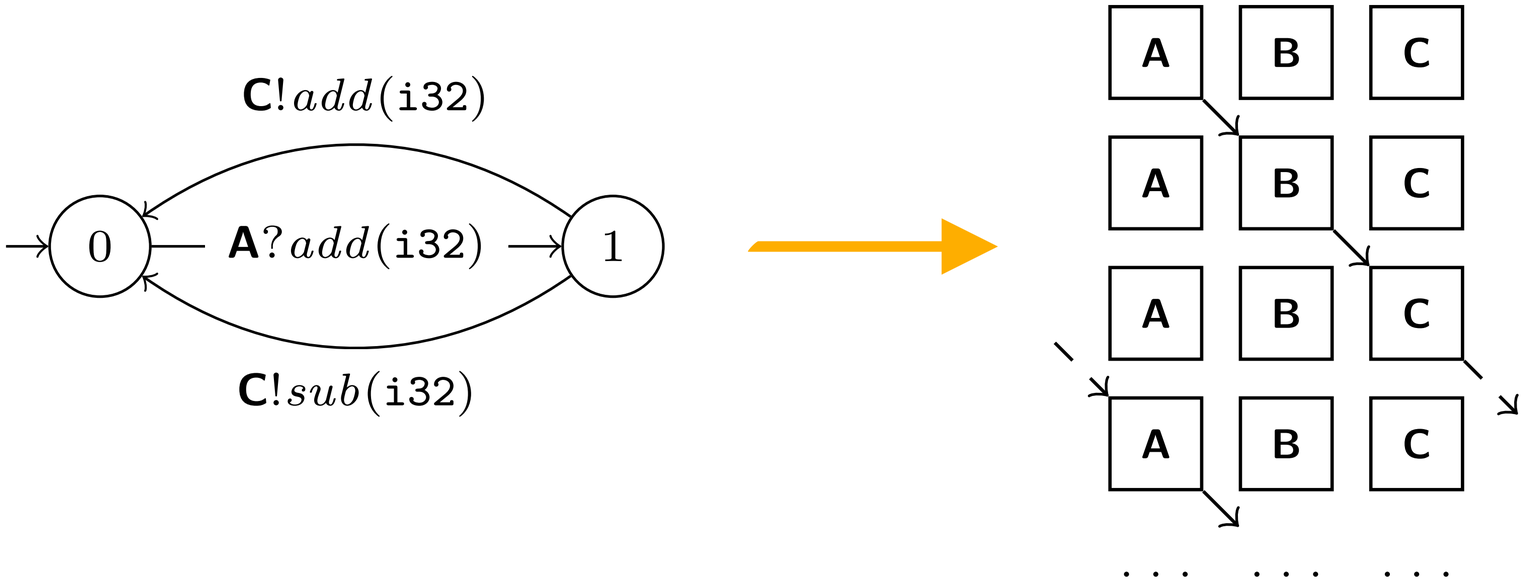
Ring Protocol

Example



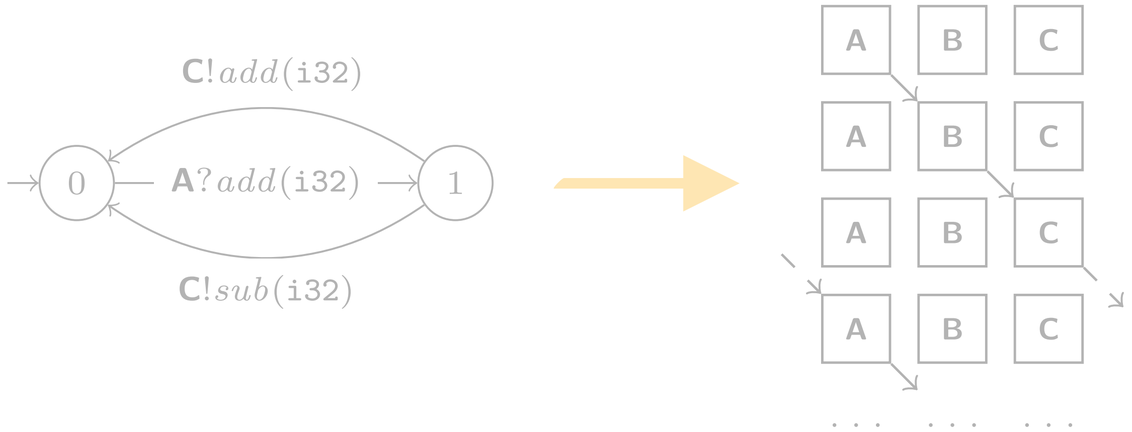
Ring Protocol

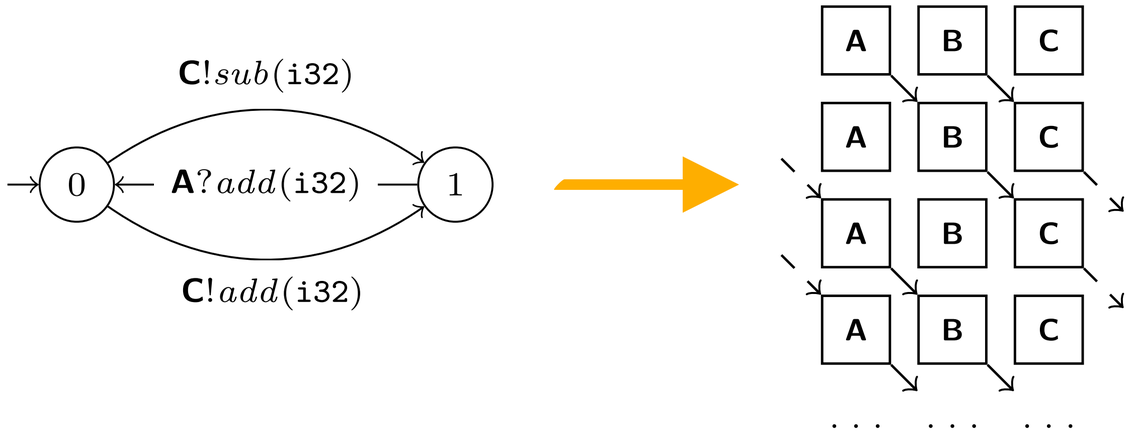
Example



Ring Protocol

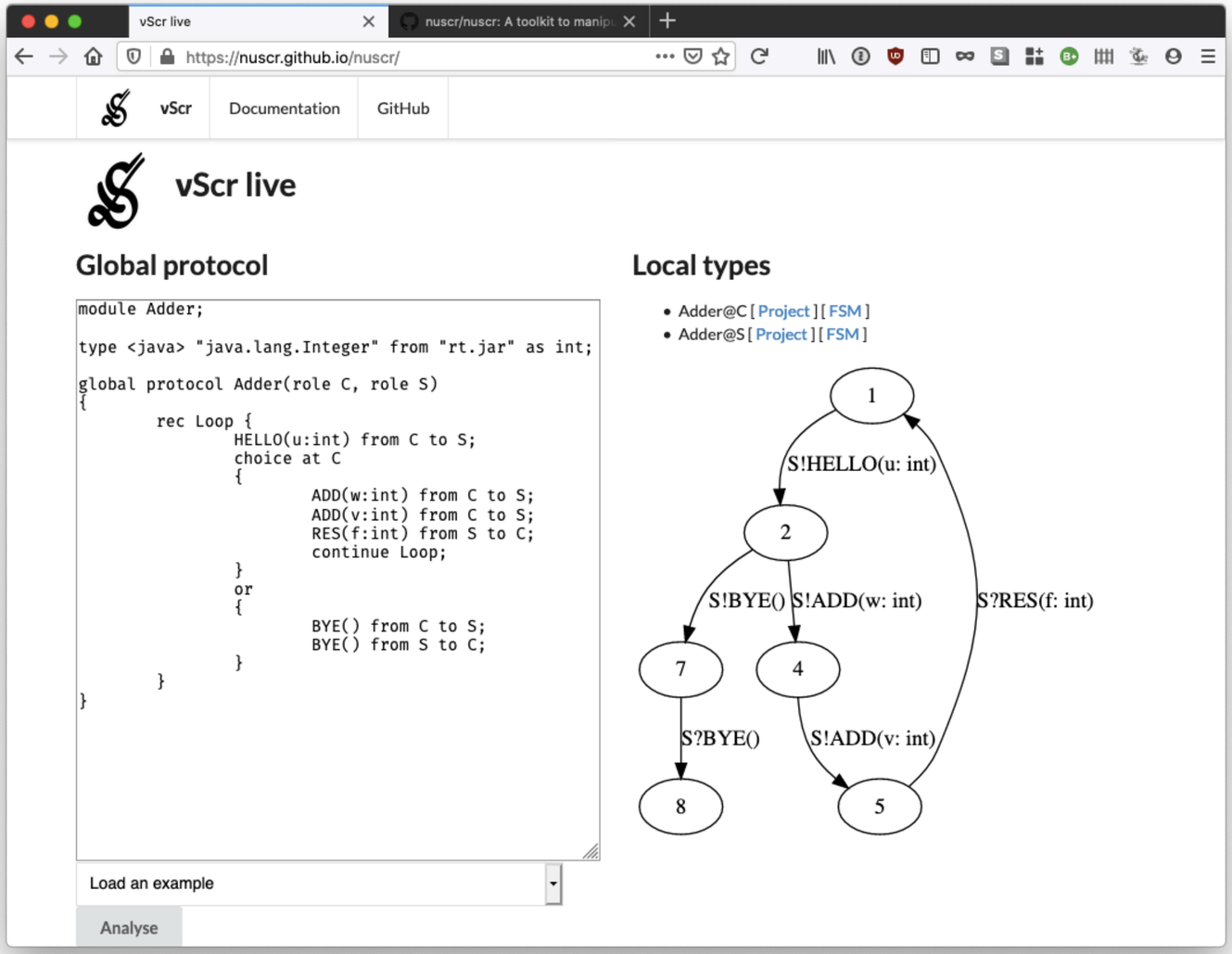
Example





vScr An Extensible Toolchain for Multiparty Session Types

- It’s small and easy to modify
- Available on opam
  - opam install nuscr
- Available on GitHub
  - https://github.com/nuscr
- Available on the web
  - https://nuscr.dev



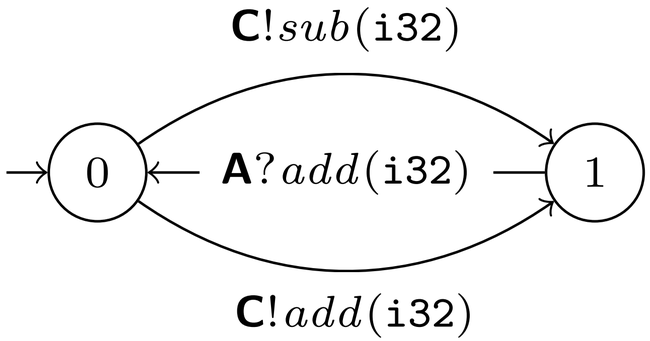
Scribble

Protocol Description Language

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

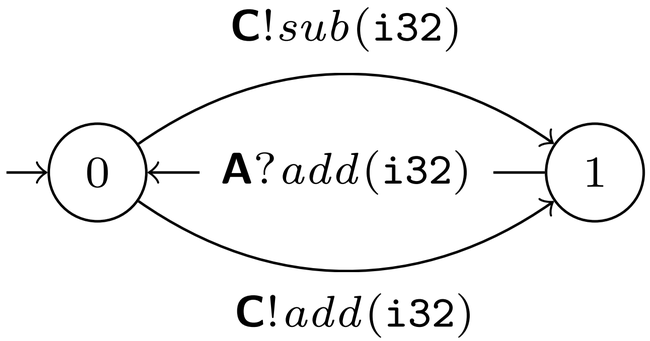
Ring Protocol

Rust API



Ring Protocol

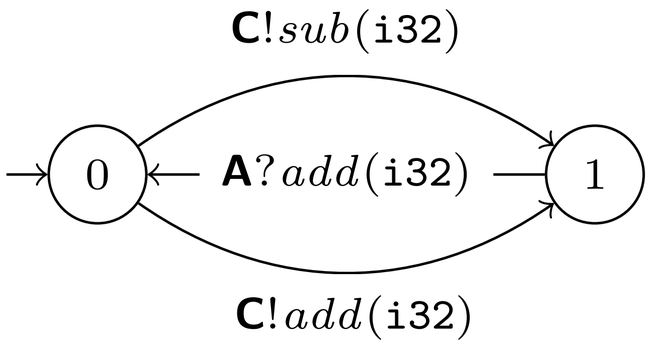
Rust API




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

Ring Protocol

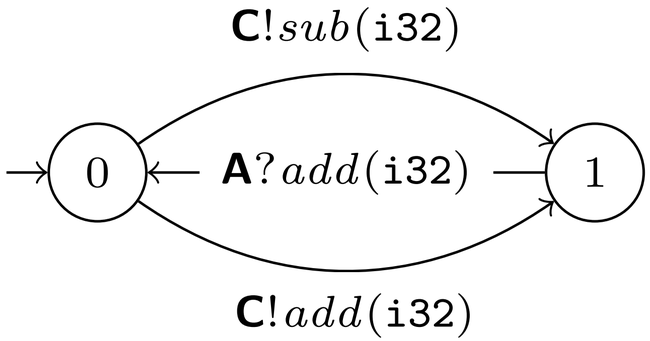
Rust API




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

Ring Protocol

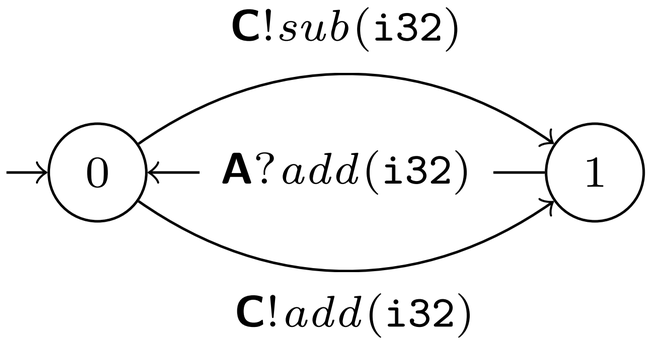
Rust API




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

Ring Protocol

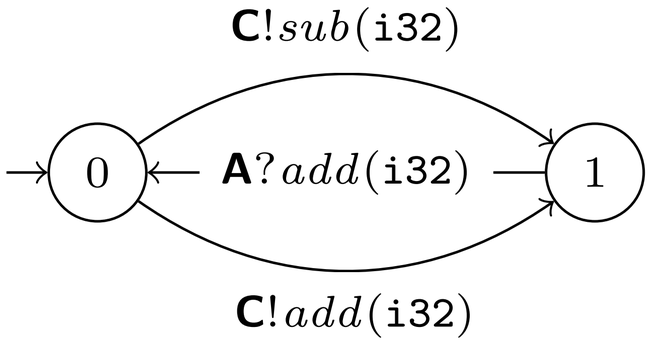
Rust API




```rust
#[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

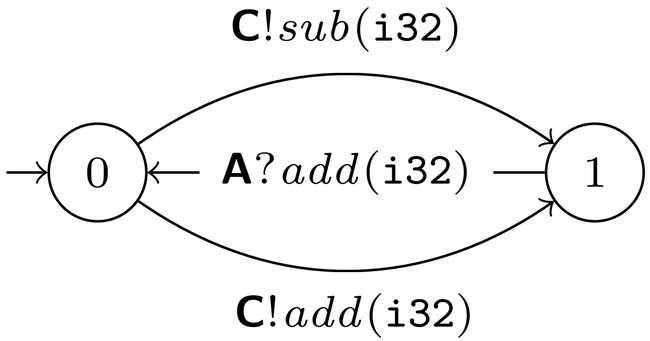
Rust API




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

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

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

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

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

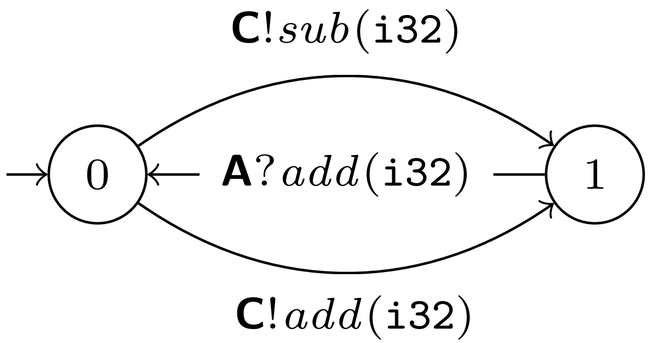
Ring Protocol

Rust API



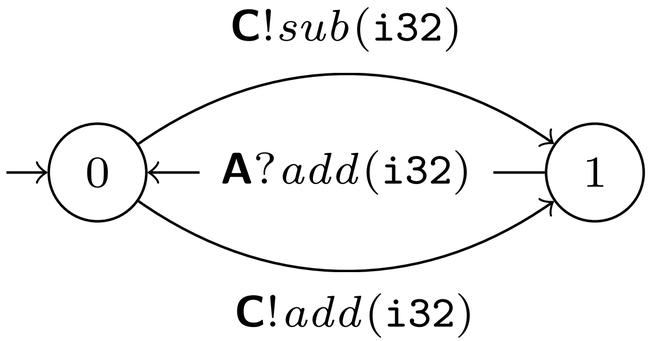
Ring Protocol

Implementation



Ring Protocol

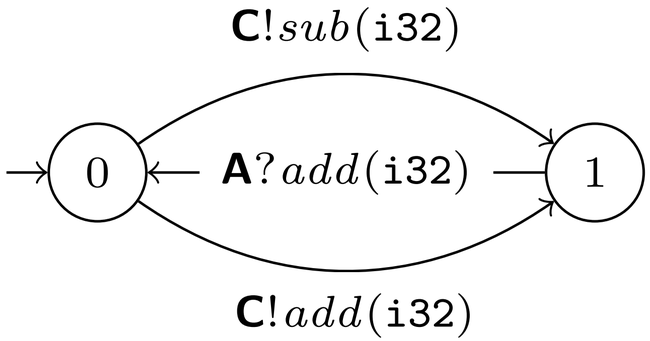
Implementation




```rust
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

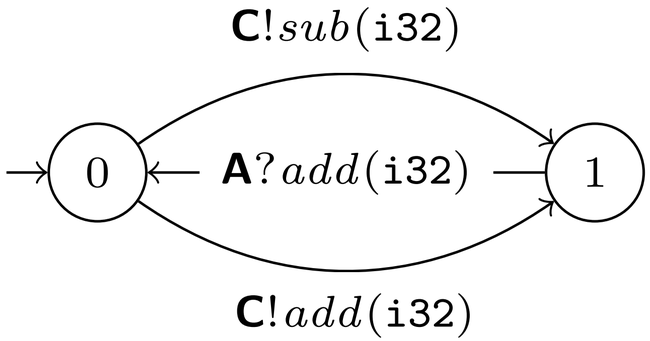
Implementation




```rust
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

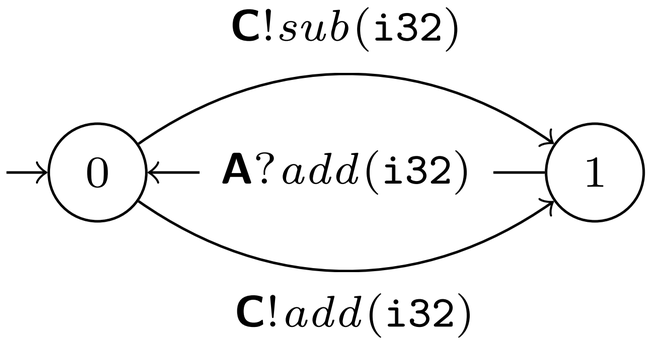
Implementation




```rust
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

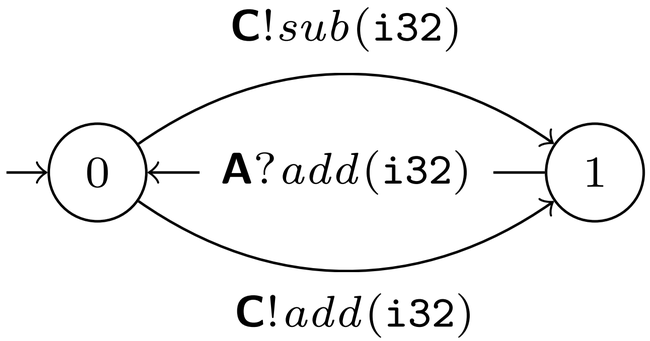
Implementation




```rust
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

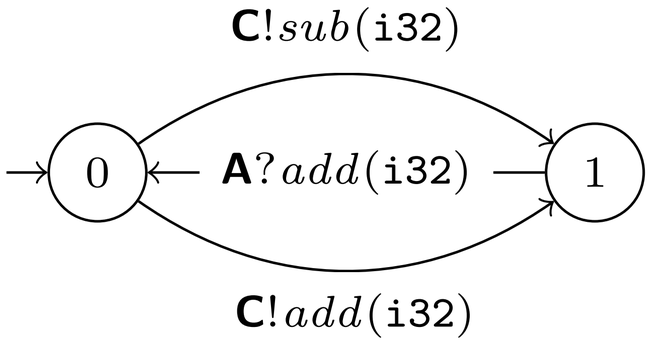
Implementation




```rust
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

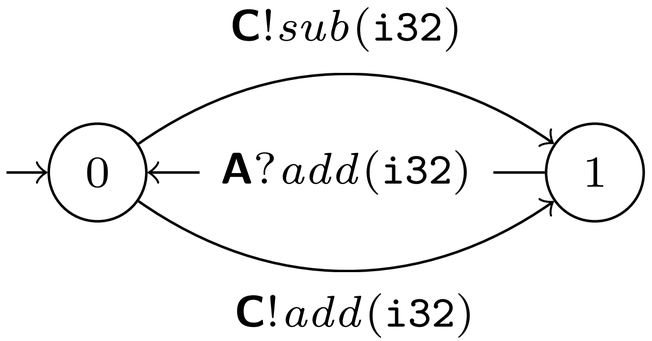
Implementation




```rust
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

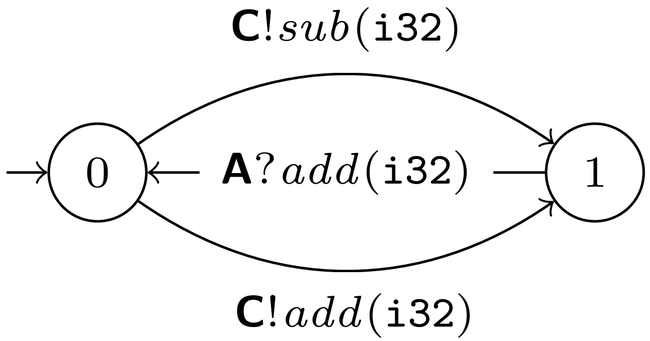
Implementation




```rust
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

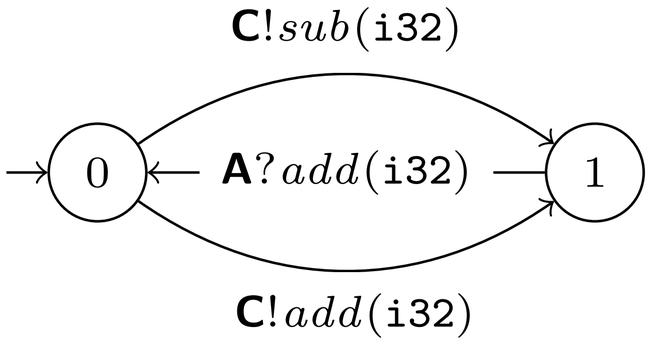
Implementation




```rust
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

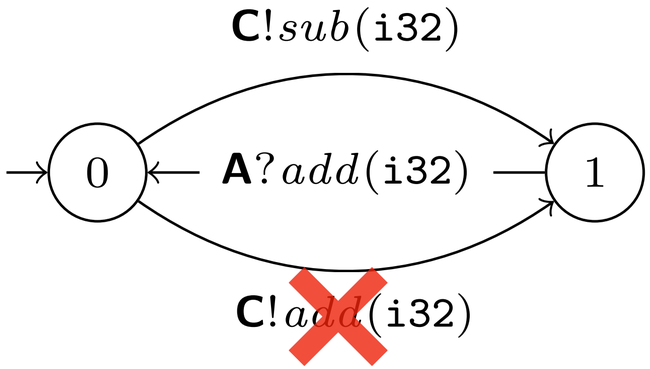
Implementation




```rust
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

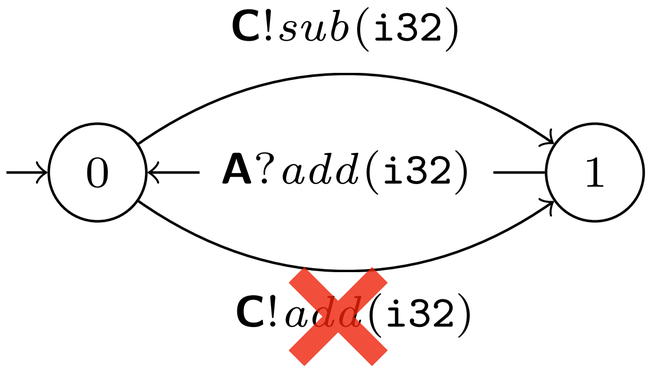
Implementation




```rust
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

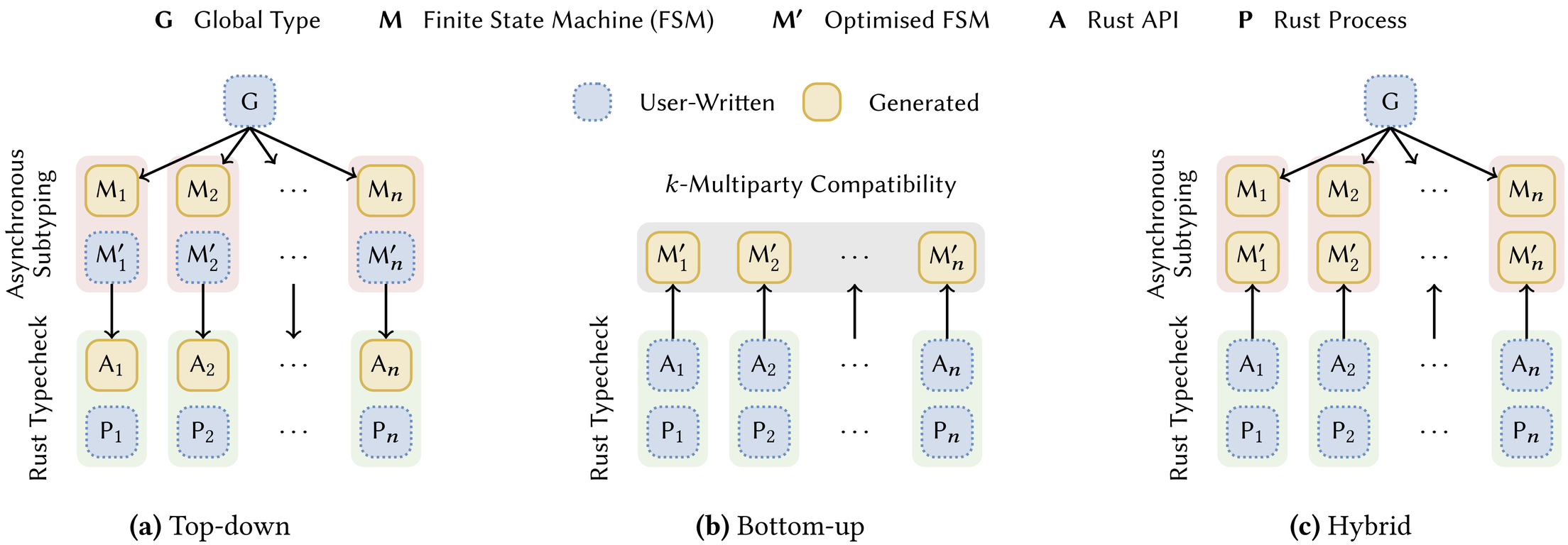



```
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

            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
}
```

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

Rumpsteak Framework

Three Approaches



(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

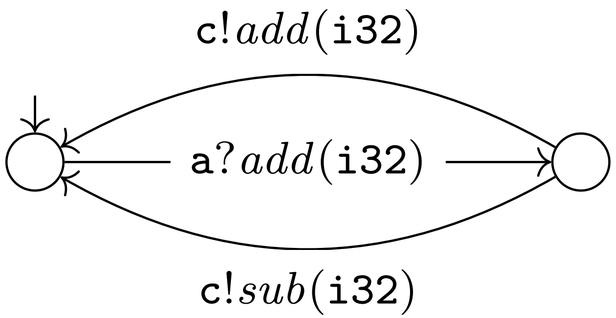
How do we check that asynchronous reorderings are safe?

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

Safety

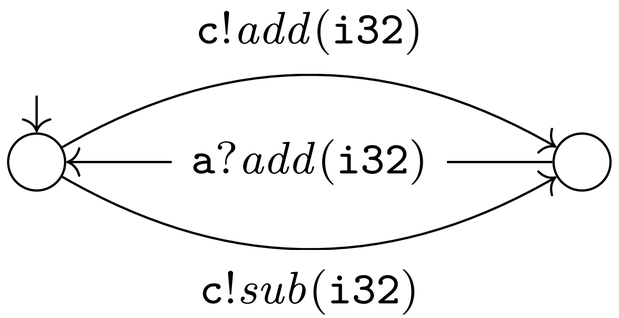
Asynchronous Subtyping

PROJECTED B



Safe?

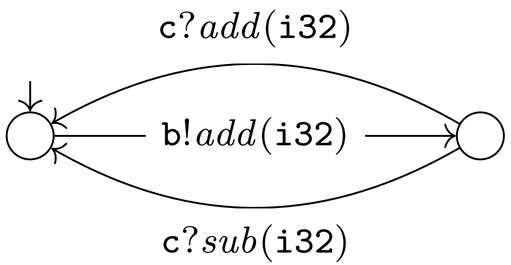
OPTIMISED B



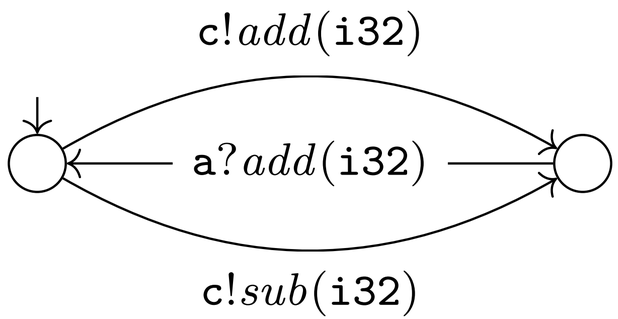
Safety

k-Multiparty Compatibility

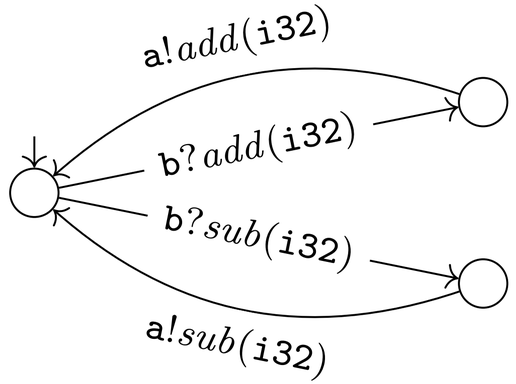
OPTIMISED A



OPTIMISED B



OPTIMISED C



Safe?

Asynchronous Subtyping

Existing work

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

Asynchronous Subtyping

Existing work

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

Asynchronous Subtyping

Existing work

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

Asynchronous Subtyping

Existing work

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

Asynchronous Subtyping

Existing work

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

Algorithm for Asynchronous Subtyping

Practical, Sound and Terminating

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

Asynchronous Subtyping

Session Type Prefix

$$
\begin{array}{rcll}
\pi, \rho & ::= & \epsilon & \text{empty prefix} \\
& | & \mathtt{p}!\ell(S) & \text{message send} \\
& | & \mathtt{p}?\ell(S) & \text{message receive} \\
& | & \pi_1.\pi_2 & \text{concatenation}
\end{array}
$$

Asynchronous Subtyping

Reduction Rules

$$\mathcal{A}^{(\mathtt{p})} \;::=\; \mathtt{q}?\ell(S) \mid \mathtt{q}?\ell(S).\mathcal{A}^{(\mathtt{p})} \qquad (\mathtt{p} \neq \mathtt{q})$$

$$\frac{S' \leq: S}{\left\langle \mathtt{p}?\ell(S).\pi, \mathcal{A}^{(\mathtt{p})}.\mathtt{p}?\ell(S').\pi' \right\rangle \to \left\langle \pi, \mathcal{A}^{(\mathtt{p})}.\pi' \right\rangle}\ [\textsc{red-}\mathcal{A}]$$

Asynchronous Subtyping

Reduction Rules

$$\mathcal{A}^{(\mathtt{p})} \;::=\; \mathtt{q}?\ell(S) \mid \mathtt{q}?\ell(S).\mathcal{A}^{(\mathtt{p})} \qquad (\mathtt{p} \neq \mathtt{q})$$

$$\frac{S' \leq: S}{\left\langle \mathtt{p}?\ell(S).\pi, \mathcal{A}^{(\mathtt{p})}.\mathtt{p}?\ell(S').\pi' \right\rangle \to \left\langle \pi, \mathcal{A}^{(\mathtt{p})}.\pi' \right\rangle} \; [\textsc{red-}\mathcal{A}]$$

Asynchronous Subtyping

Reduction Rules

$$\mathcal{A}^{(\mathtt{p})} \;::=\; \mathtt{q}?\ell(S) \mid \mathtt{q}?\ell(S).\mathcal{A}^{(\mathtt{p})} \qquad (\mathtt{p} \neq \mathtt{q})$$

$$\frac{S' \leq: S}{\left\langle \mathtt{p}?\ell(S).\pi, \mathcal{A}^{(\mathtt{p})}.\mathtt{p}?\ell(S').\pi' \right\rangle \to \left\langle \pi, \mathcal{A}^{(\mathtt{p})}.\pi' \right\rangle}\ [\textsc{red-}\mathcal{A}]$$

Asynchronous Subtyping

Reduction Rules

$$\mathcal{A}^{(\mathtt{p})} \;::=\; \mathtt{q}?\ell(S) \mid \mathtt{q}?\ell(S).\mathcal{A}^{(\mathtt{p})} \qquad (\mathtt{p} \neq \mathtt{q})$$

$$\frac{S' \leq: S}{\left\langle \mathtt{p}?\ell(S).\pi, \mathcal{A}^{(\mathtt{p})}.\mathtt{p}?\ell(S').\pi' \right\rangle \to \left\langle \pi, \mathcal{A}^{(\mathtt{p})}.\pi' \right\rangle}\ [\textsc{red-}\mathcal{A}]$$

Asynchronous Subtyping

Reduction Rules

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

Asynchronous Subtyping

Reduction Rules

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

Asynchronous Subtyping

Reduction Rules

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

Asynchronous Subtyping

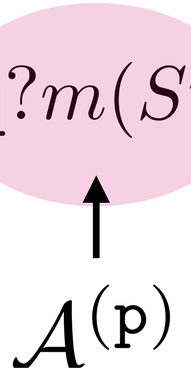
Reduction Rules

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

Asynchronous Subtyping

Reduction Rules

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



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

✅

Asynchronous Subtyping

Reduction Rules

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

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

Asynchronous Subtyping

Reduction Rules

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

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

Asynchronous Subtyping

Reduction Rules

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

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

Asynchronous Subtyping

Reduction Rules

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

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

Asynchronous Subtyping

Reduction Rules

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

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

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

Asynchronous Subtyping

Reduction Rules

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

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

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

$$\mathcal{A}^{(\mathtt{p})} ::= \mathtt{q}?\ell(S) \mid \mathtt{q}?\ell(S).\mathcal{A}^{(\mathtt{p})} \qquad (\mathtt{p} \neq \mathtt{q})$$

Theorems

Termination, Soundness & Complexity

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

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

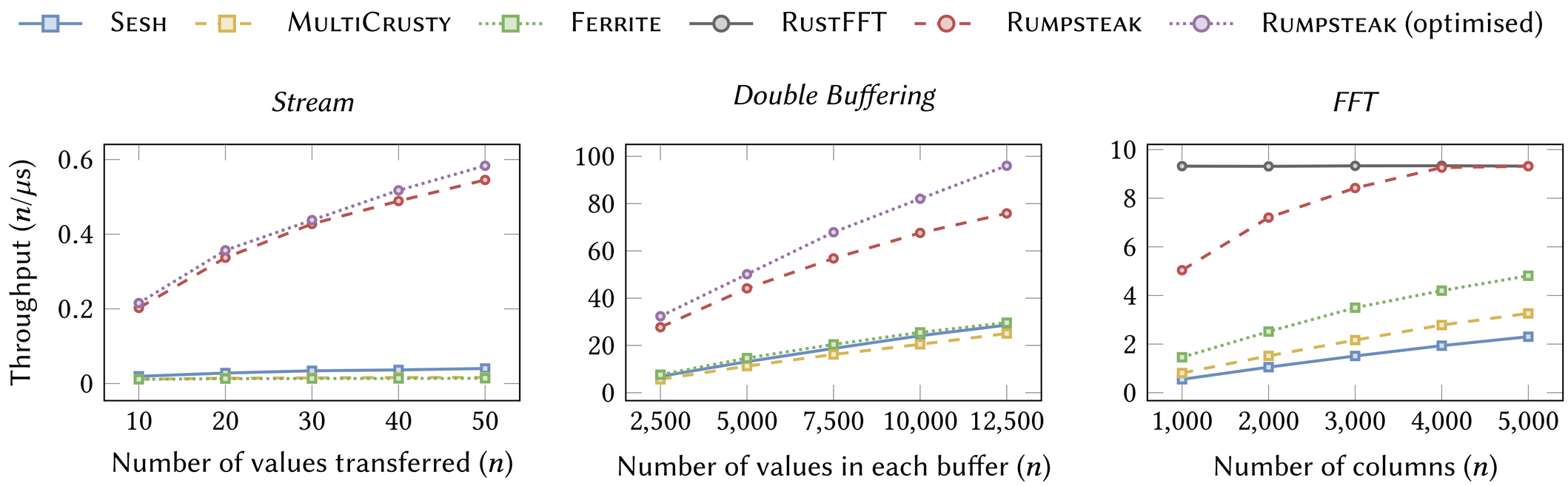
Theorem 5 (Soundness)**.** *Our subtyping algorithm is sound.*

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

Theorem 7 (Complexity)**.** *Consider* T *and* T' *as (possibly infinite) trees* $\mathcal{T}(\mathsf{T})$ *and* $\mathcal{T}(\mathsf{T}')$ *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* $\mathsf{T} \leq \mathsf{T}'$ *with bound* n.

Evaluation

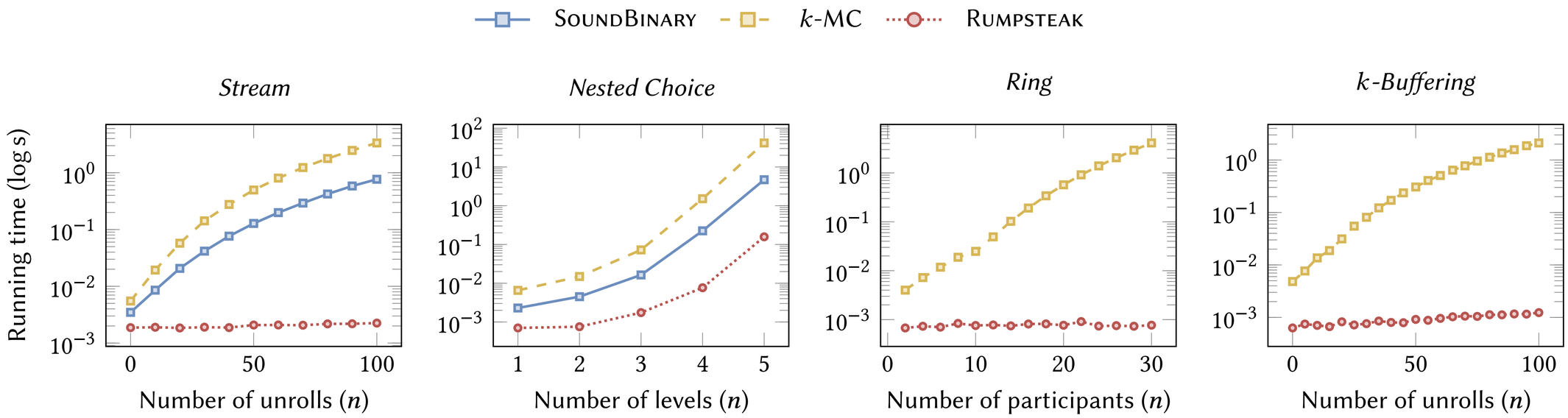
Rust Framework Benchmarks



16-core AMD OpteronTM 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



Evaluation

Expressiveness

Protocol	n	AMR	Sesh	Ferrite	MultiCrusty	Rumpsteak	k-MC	SoundBinary
Two Adder	2		✔	✔	✔	✔	✔	✔
Three Adder	3		✗	✗	✔	✔	✔	✗
Stream	2		✔	✔	✔	✔	✔	✔
Optimised Stream	2	✔	✗	✗	✗	✔	✔	✔
Ring	3		✗	✗	✔	✔	✔	✗
Optimised Ring	3	✔	✗	✗	✗	✔	✔	✗
Ring With Choice	3		✗	✗	✔	✔	✔	✗
Optimised Ring With Choice	3	✔	✗	✗	✗	✔	✔	✗
Double Buffering	3		✗	✗	✔	✔	✔	✗
Optimised Double Buffering	3	✔	✗	✗	✗	✔	✔	✗
Alternating Bit	2		✗	✗	✗	✔	✔	✔
Elevator	3	✔	✗	✗	✗	✔	✔	✗
FFT	8		✗	✗	✔	✔	✔	✗
Optimised FFT	8	✔	✗	✗	✗	✔	✔	✗
Authentication	3		✗	✗	✔	✔	✔	✗
Client-Server Log	3		✗	✗	✔	✔	✔	✗
Hospital	2	✔	✗	✗	✗	✗	✗	✔

n Number of participants **AMR** Asynchronous message reordering

✔ Expressible ✗ Expressible using endpoint types (but without deadlock-freedom guarantee) ✗ Not expressible

References

Multiparty Session Types and Rust

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

Mini-Project Proposals

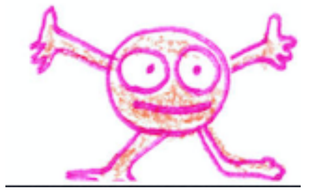
Amazon Prime Video Team and Actor Languages

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

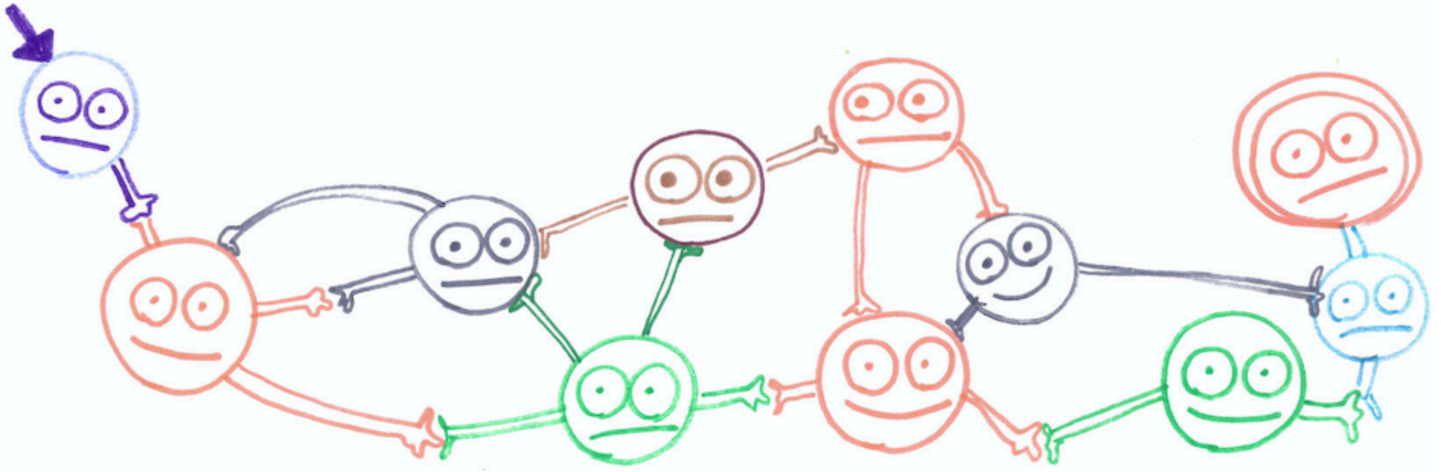




Thank you! Questions?



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



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

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

Q a finite set of states

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

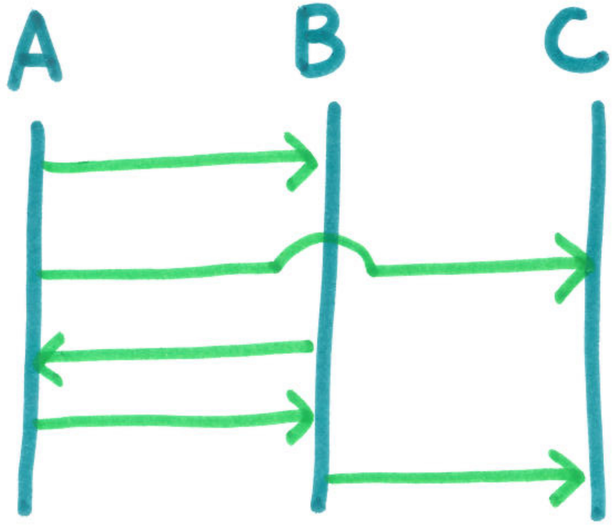
$q0$ initial state

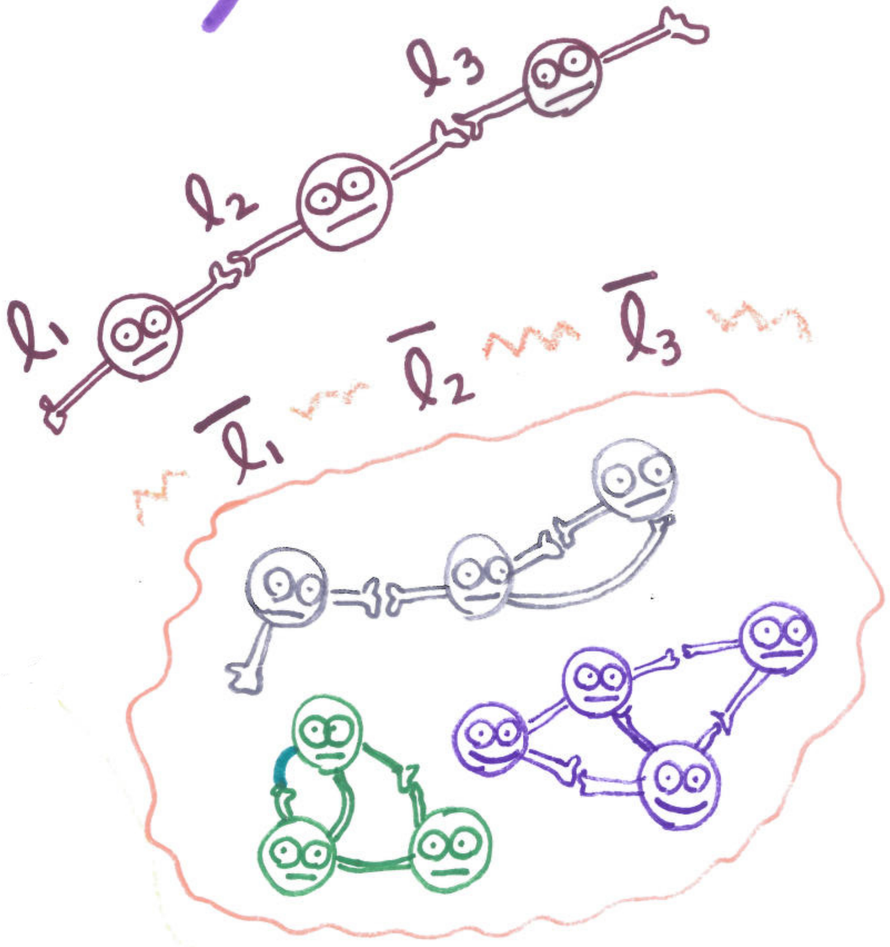
Σ a finite alphabet of messages

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

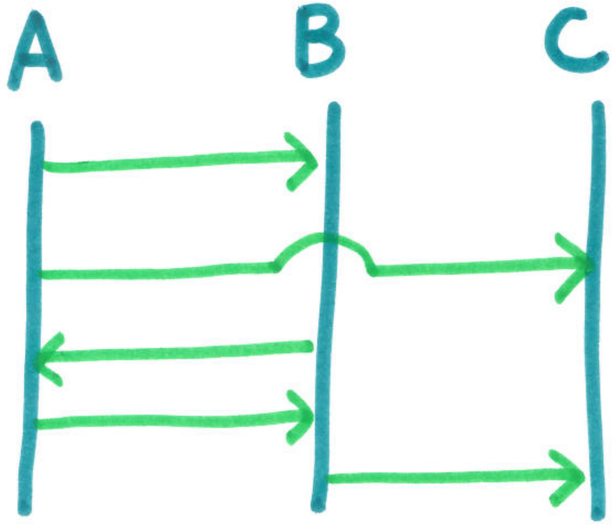
Def CS $S = (M_p)_{p \in Participant}$

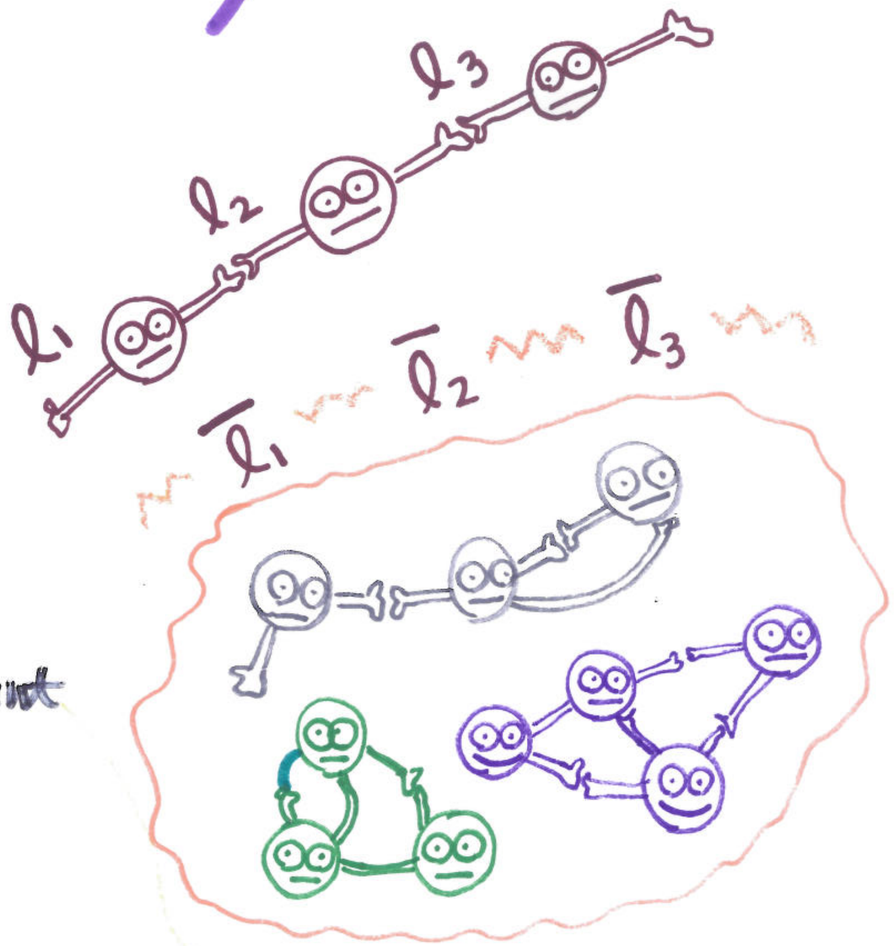
Multiparty Compatibility





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 $\rightsquigarrow$

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 $act(t) = pq?a$ and $p \notin act(\varphi)$

(Dual for Input)

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

states queues

q_p w_{pq}

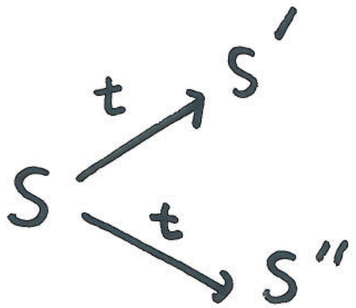
Send

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

Receive

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

Deteministic CFSM

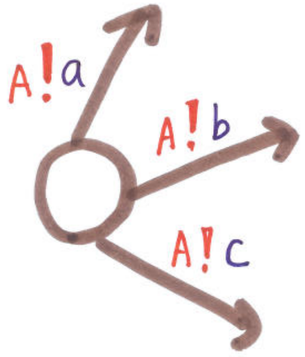


$\Rightarrow \quad s' = s''$

Basic CFSMs

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

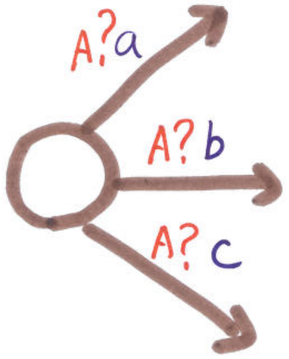
directed, has **no mixed states**



sending

✓

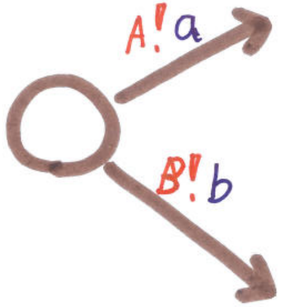
$T = A!\{a, b, c\}$



receiving

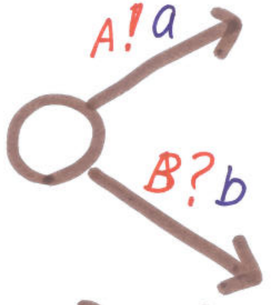
✓

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



non directed

✗



mixed

✗

k-Multiparty Compatibility [CAV'19]