

# Systems Programming

Nobuko Yoshida, 28th November, Lorentz Seminar



DEPARTMENT OF  
**COMPUTER  
SCIENCE**

# Deadlock-Free Asynchronous Message Reordering in Rust with Multiparty Session Types

27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming **[PPoPP 2022]**



Zak Cutner, NY and Martin Vassor

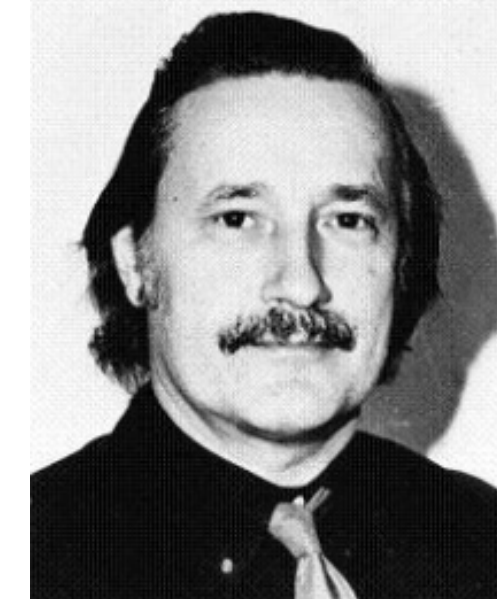
Imperial College  
London



DEPARTMENT OF  
**COMPUTER  
SCIENCE**

# Multiparty Session Types

## Specification-Guided Concurrent and Distributed Programming



1916-1975

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

# Communications are Ubiquitous

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



microservices



# Problems: Ambiguity

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



RFC 821

August 1982  
Simple Mail Transfer Protocol

## TABLE OF CONTENTS

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

# Problems: Ambiguity

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



## 3.1. MAIL

There are three steps to SMTP mail transactions. The transaction is started with a MAIL command which gives the sender identification. A series of one or more RCPT commands follows giving the receiver information. Then a DATA command gives the mail data. And finally, the end of mail data indicator confirms the transaction.

The first step in the procedure is the MAIL command. The <reverse-path> contains the source mailbox.

```
MAIL <SP> FROM:<reverse-path> <CRLF>
```

This command tells the SMTP-receiver that a new mail transaction is starting and to reset all its state tables and buffers, including any recipients or mail data. It gives the reverse-path which can be used to report errors. If accepted, the receiver-SMTP returns a 250 OK reply.

The <reverse-path> can contain more than just a mailbox. The <reverse-path> is a reverse source routing list of hosts and source mailbox. The first host in the <reverse-path> should be the host sending this command.

The second step in the procedure is the RCPT command.

```
RCPT <SP> TO:<forward-path> <CRLF>
```

This command gives a forward-path identifying one recipient. If accepted, the receiver-SMTP returns a 250 OK reply, and stores the forward-path. If the recipient is unknown the receiver-SMTP returns a 550 Failure reply. This second step of the procedure can be repeated any number of times.

# Problems: Concurrency Bugs

- Communications increase **concurrency bugs**

- Survey of 4k users [golang.org]
- Analysis of 6 large software systems [ASPLOS 19]  
[PLDI 22]



Uber

GO

Google (2009)



The Go Gopher

CSP<sub>80'</sub>

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

– *Go Philosophy*

# Problems: Concurrency Bugs

---

- Communications increase **concurrency bugs**

- Survey of 4K users [golang.org]
- Analysis of 6 large software systems [ASPLOS 19]

Uber's 14 million lines of Go hosting 2100 microservices [PLDI 22]

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

deadlock

channel errors



The Go Gopher

# Problems: Concurrency Bugs

- Communications increase **concurrency bugs**
  - Survey of 4k users [golang.org]
  - Analysis of 6 large software systems [ASPLOS 19]  
[PLDI 22]

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



## Session Types

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

# Why Session Types, Why Now?

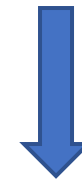
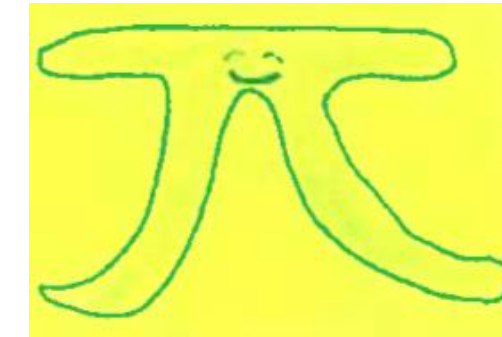
Significant academic and industry interests via fundamental breakthroughs

Milner,  
Honda, NY



Binary Session Types

ESOP'98

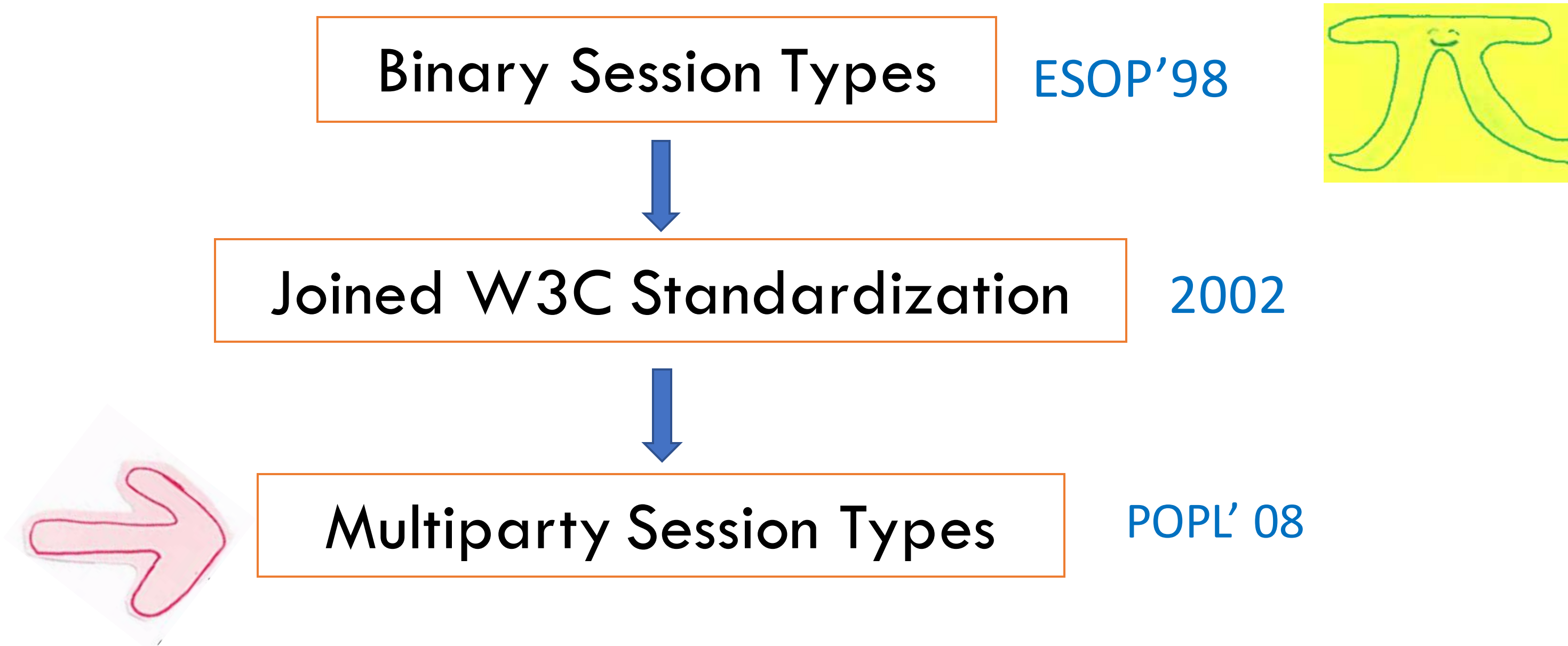


Joined W3C Standardization

2002

# Why Session Types, Why Now?

Significant academic and industry interests via fundamental breakthroughs

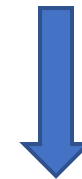
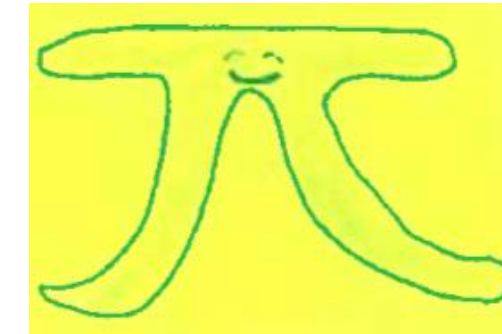


# Why Session Types, Why Now?

Significant academic and industry interests via fundamental breakthroughs

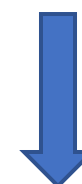
Binary Session Types

ESOP'98



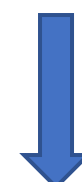
Joined W3C Standardization

2002



Multiparty Session Types

POPL' 08

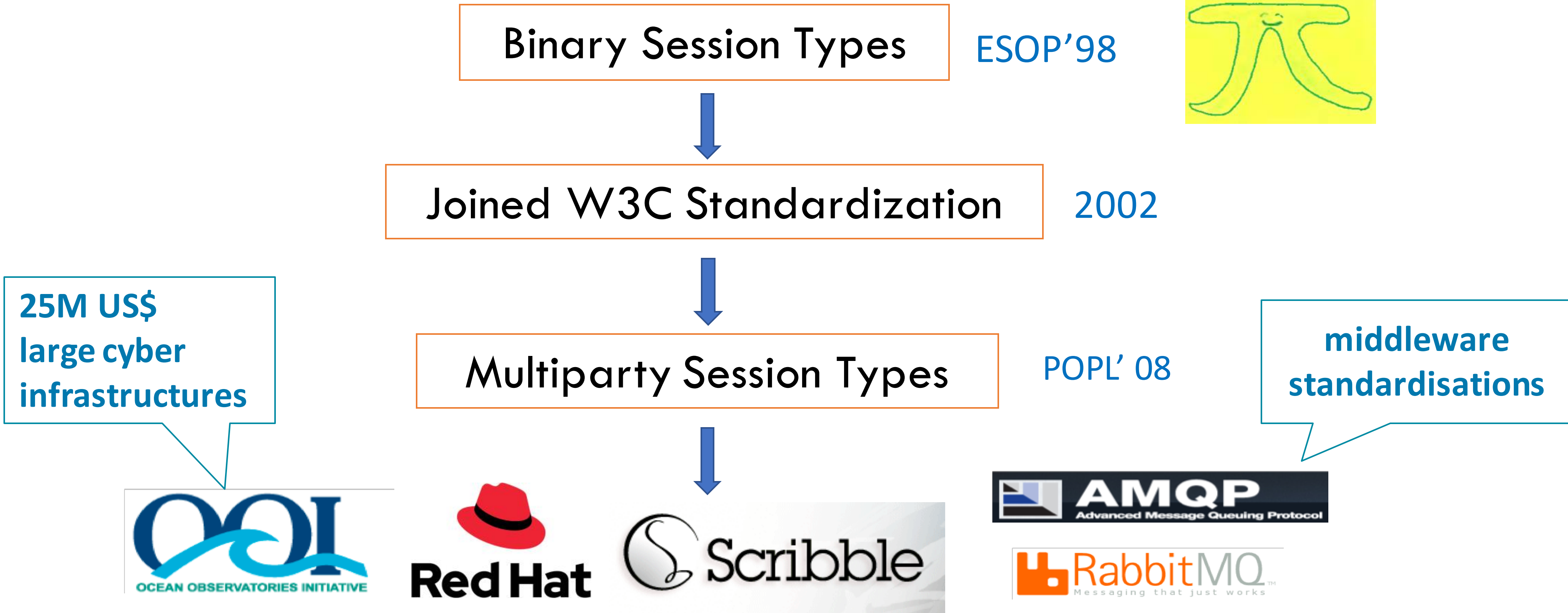


largest open source  
company in the world



# Why Session Types, Why Now?

Significant academic and industry interests via fundamental breakthroughs

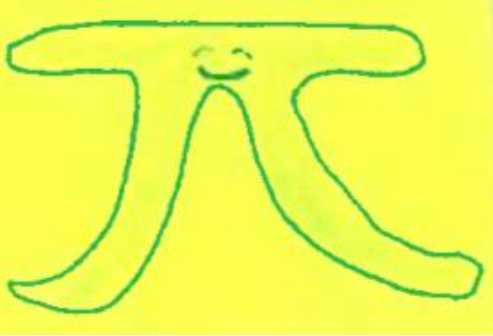


# Why Session Types, Why Now?

Significant academic and industry interests via fundamental breakthroughs

Binary Session Types

ESOP'98



Joined W3C Standardization

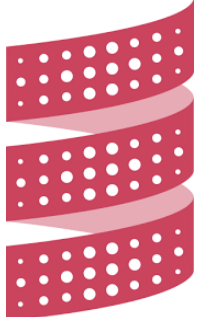
2002

Multiparty Session Types

POPL'08



TypeScript



Scala

akka



ERLANG

MPI



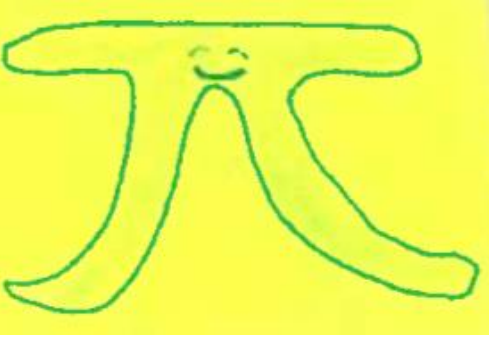
# Why Session Types, Why Now?

Significant academic and industry interests via fundamental breakthroughs

ETAPS Test Time Award 2019

Binary Session Types

ESOP'98



Joined W3C Standardization

2002



Multiparty Session Types

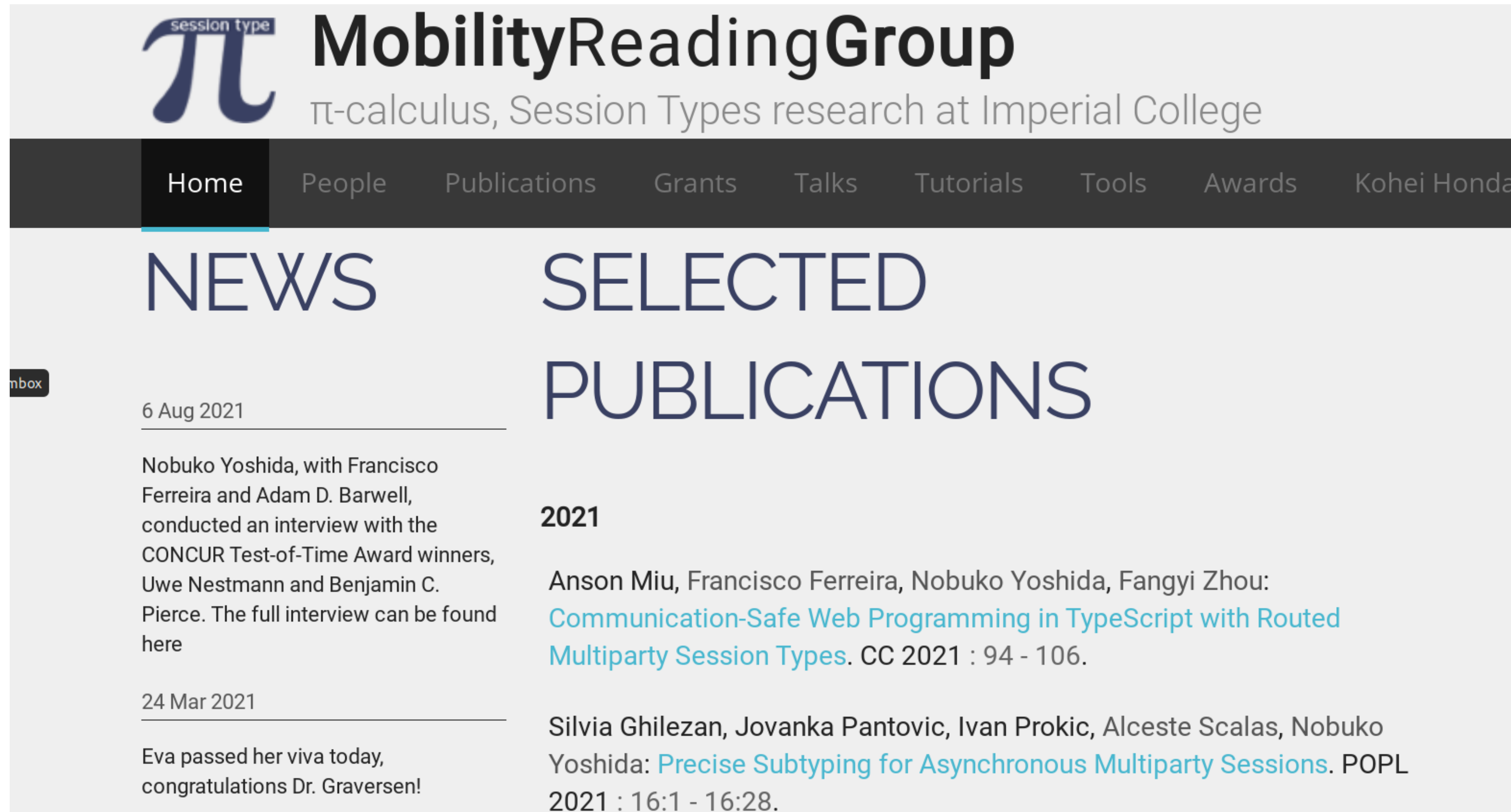
POPL' 08

POPL Influential Paper Award 2018



# Mobility Reading Group

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



The screenshot shows the website for the Mobility Reading Group. At the top left is a logo consisting of a blue pi symbol with the text "session type" above it. To the right of the logo is the text "MobilityReadingGroup" in a large, bold, black font, followed by "π-calculus, Session Types research at Imperial College" in a smaller, grey font. Below this is a dark grey navigation bar with white text for "Home", "People", "Publications", "Grants", "Talks", "Tutorials", "Tools", "Awards", and "Kohei Honda". The "Home" link is highlighted with a blue underline. The main content area is split into two columns. The left column is titled "NEWS" in large blue letters. It contains two news items, each with a date and a short paragraph. The right column is titled "SELECTED PUBLICATIONS" in large blue letters. It contains a section for the year "2021" with two publication entries, each listing authors and a link to the paper.

**session type** **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.

# Introduction

## Rust Language

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

# Introduction

## Rust Language

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

# Introduction

## Rust Language

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

# Introduction

## Rust Language

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

# Ring Protocol

## Example

### Global Type

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

# Ring Protocol

## Example

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

# Ring Protocol

## Example

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

# Ring Protocol

## Example

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

# Ring Protocol

## Example

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

# Ring Protocol

## Example

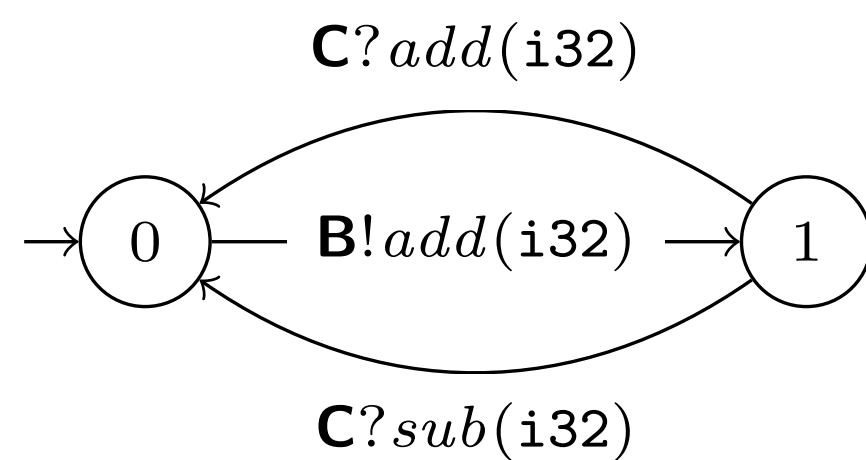
$$G = \mu t. \mathbf{A} \rightarrow \mathbf{B} : \left\{ \begin{array}{l} \mathit{add}(\mathit{i32}).\mathbf{B} \rightarrow \mathbf{C} : \left\{ \begin{array}{l} \mathit{add}(\mathit{i32}).\mathbf{C} \rightarrow \mathbf{A} : \{ \mathit{add}(\mathit{i32}).\mathbf{t} \} \\ \mathit{sub}(\mathit{i32}).\mathbf{C}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

# Ring Protocol

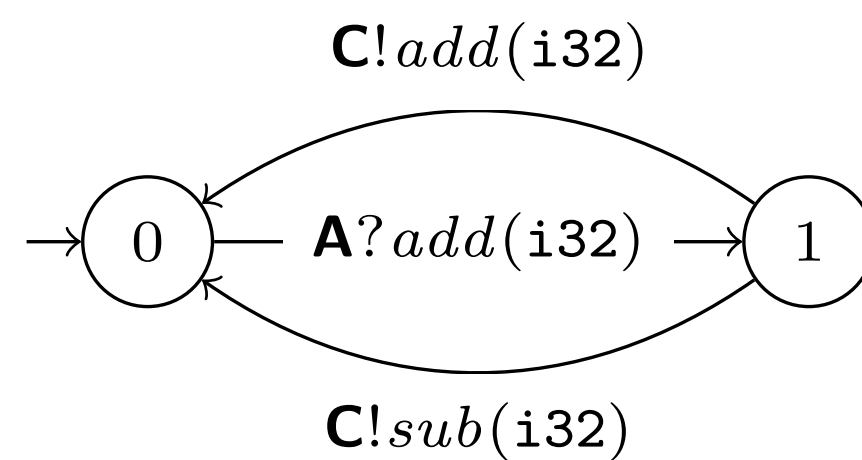
## Example

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

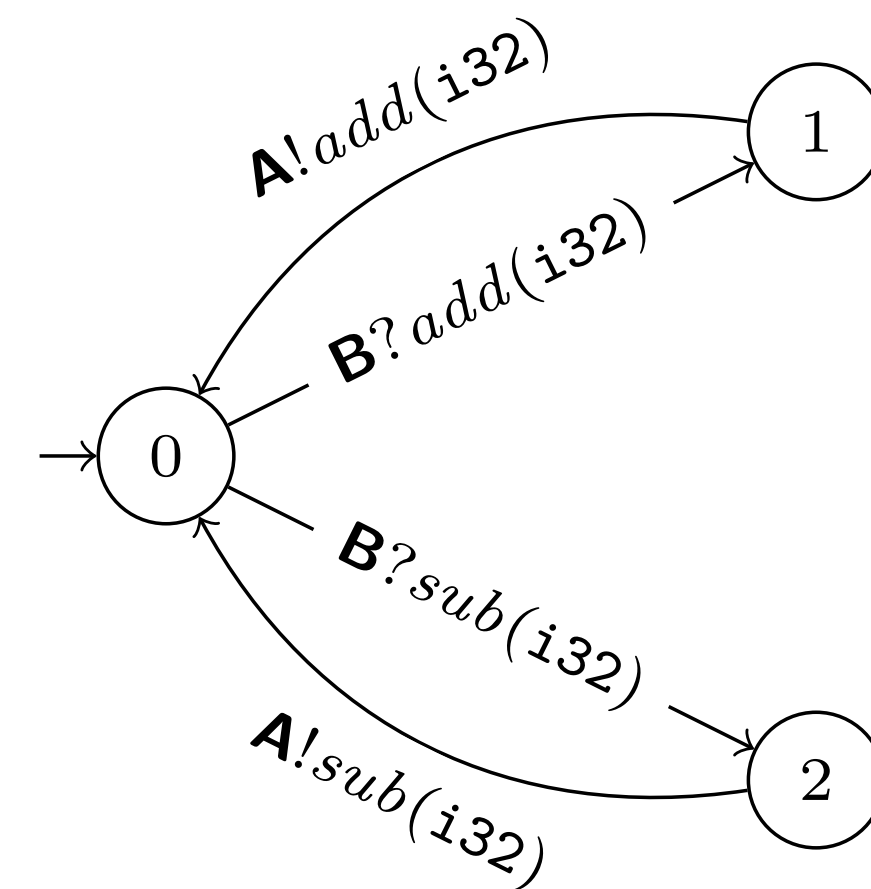
PROJECTION



PROJECTION



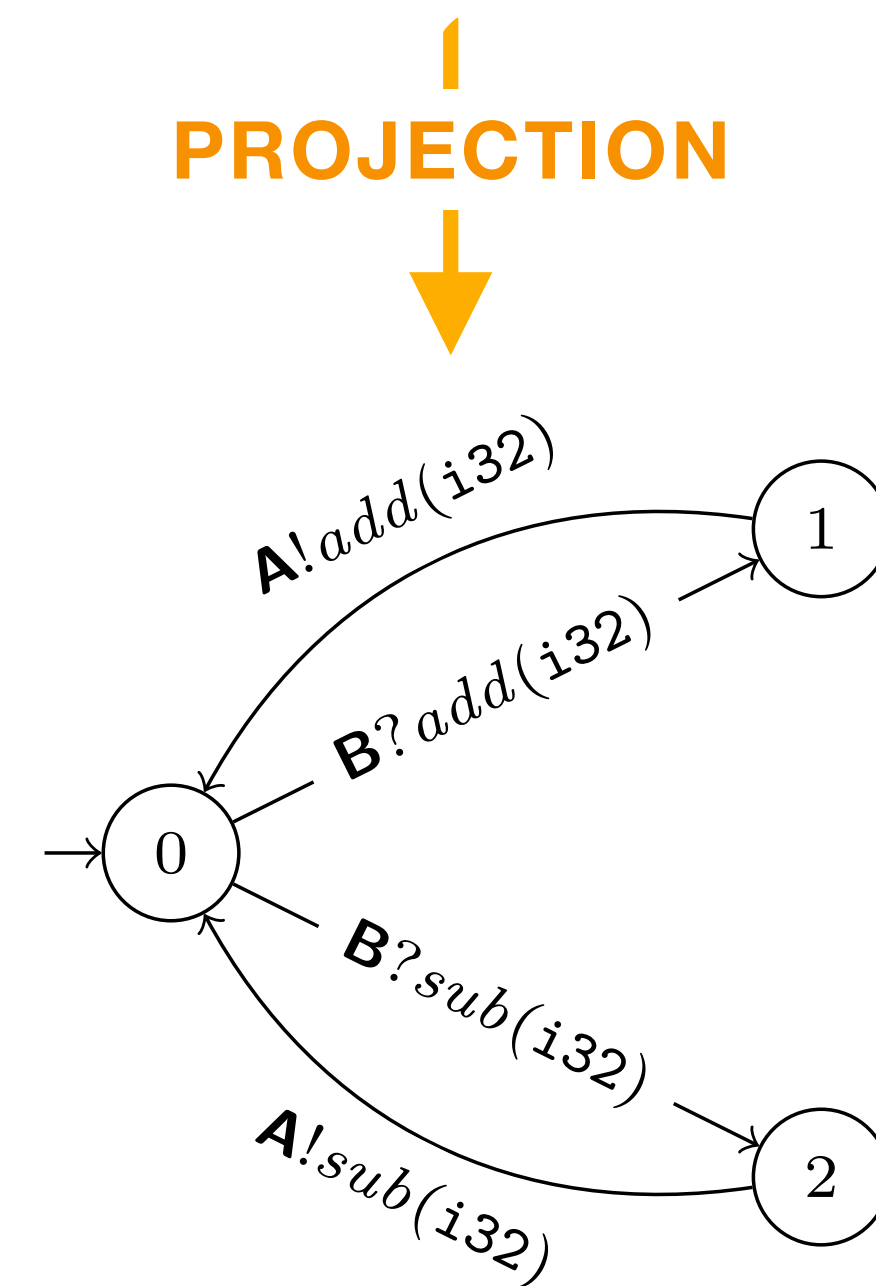
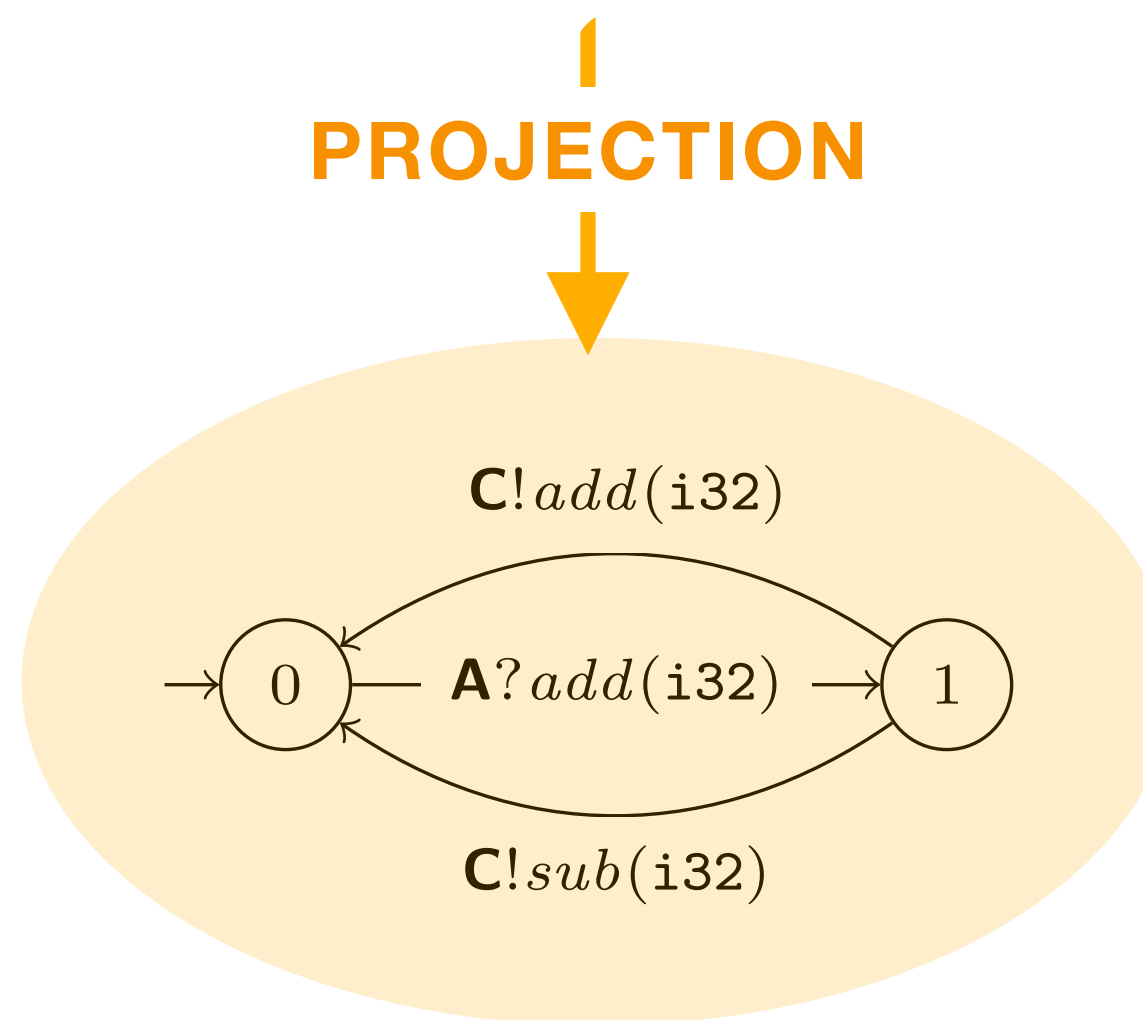
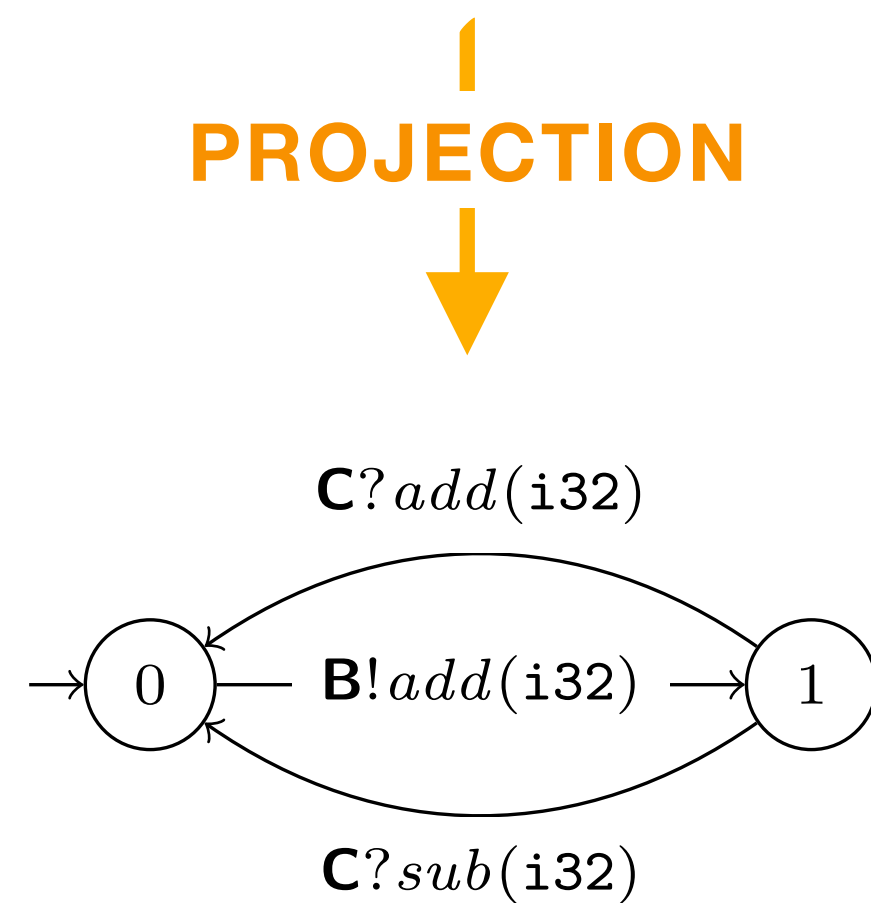
PROJECTION



# Ring Protocol

## Example

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



# Challenge

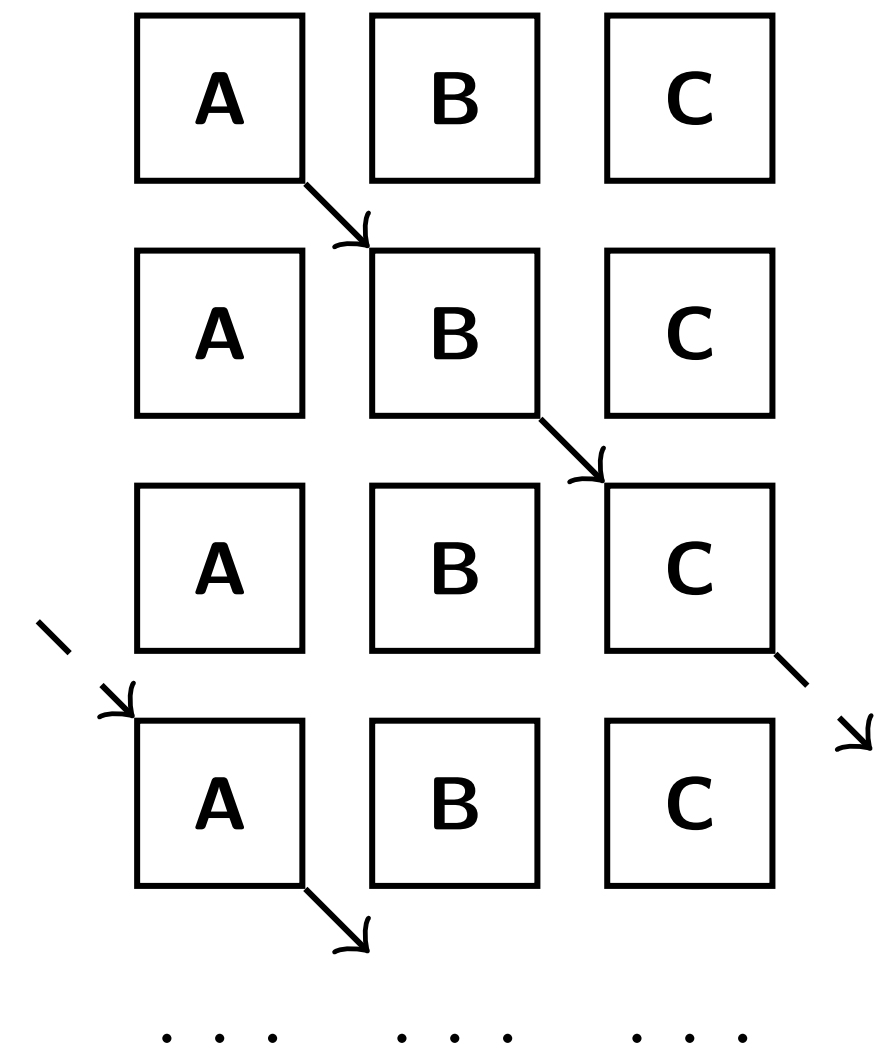
## Asynchronous Orderings

- Global types are inherently **synchronous**

# Challenge

## Asynchronous Orderings

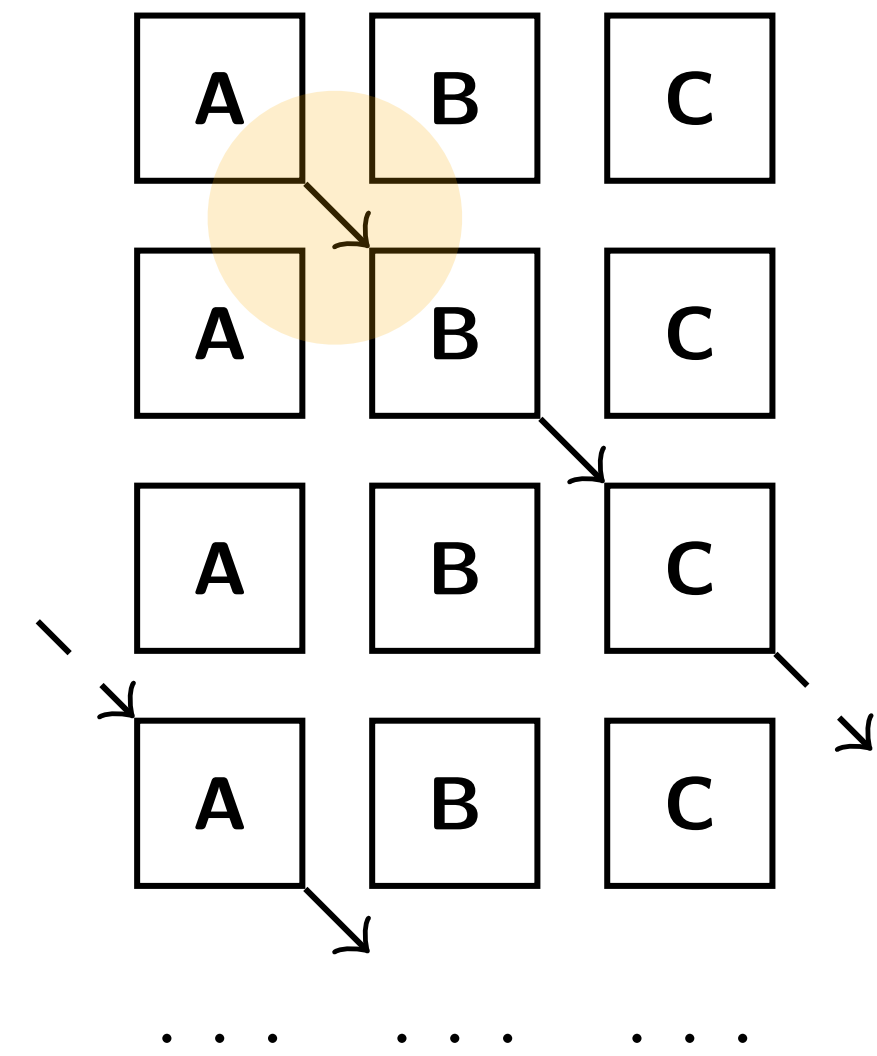
- Global types are inherently **synchronous**
  - Projection provides only one possible ordering



# Challenge

## Asynchronous Orderings

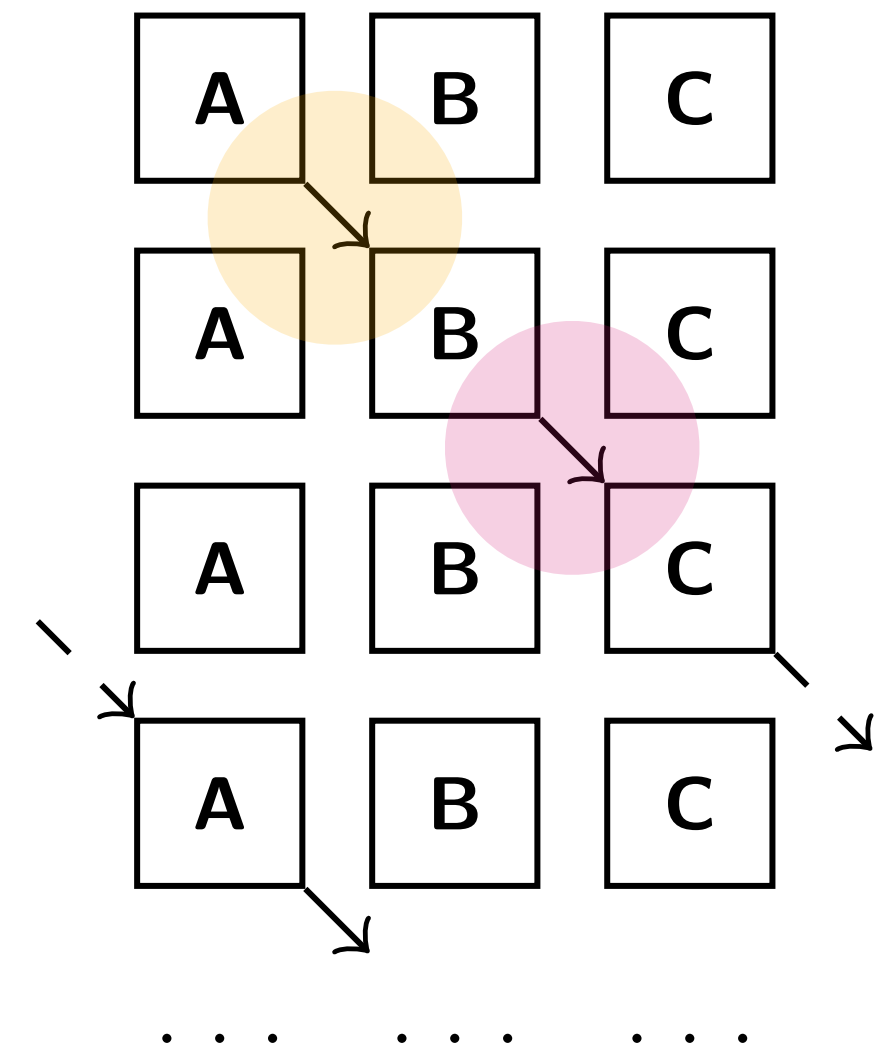
- Global types are inherently **synchronous**
  - Projection provides only one possible ordering



# Challenge

## Asynchronous Orderings

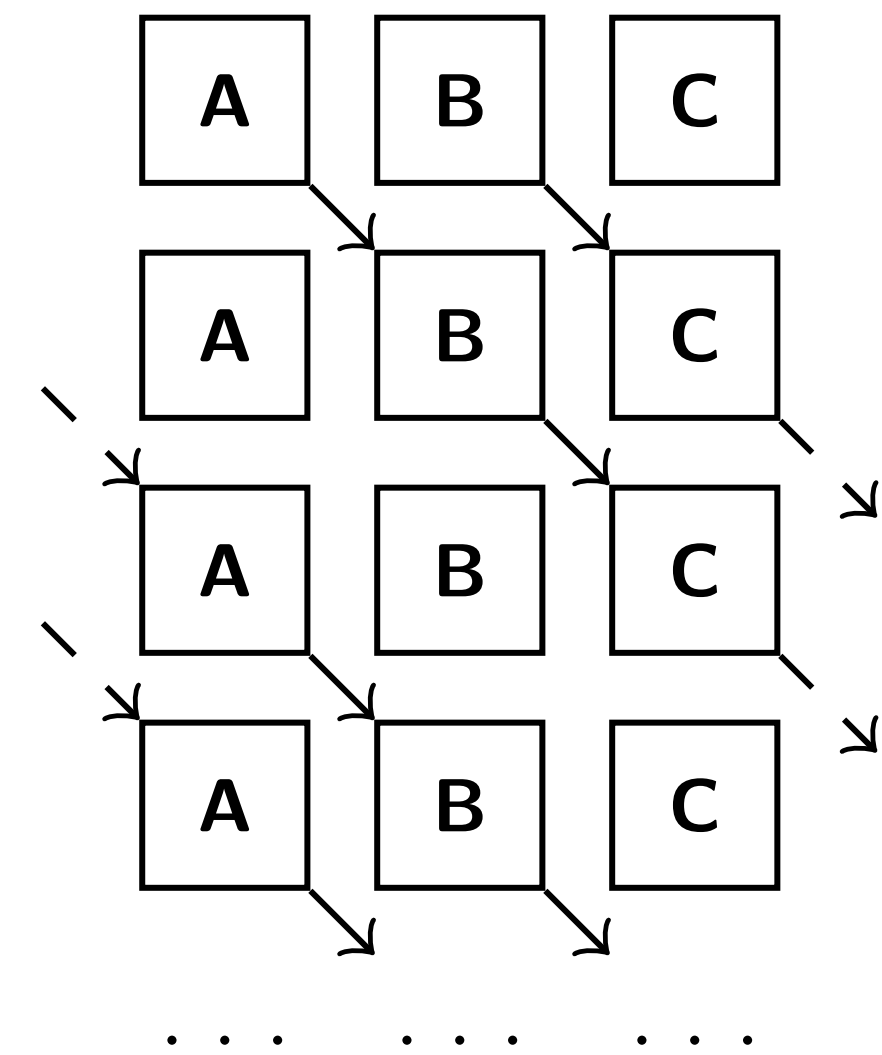
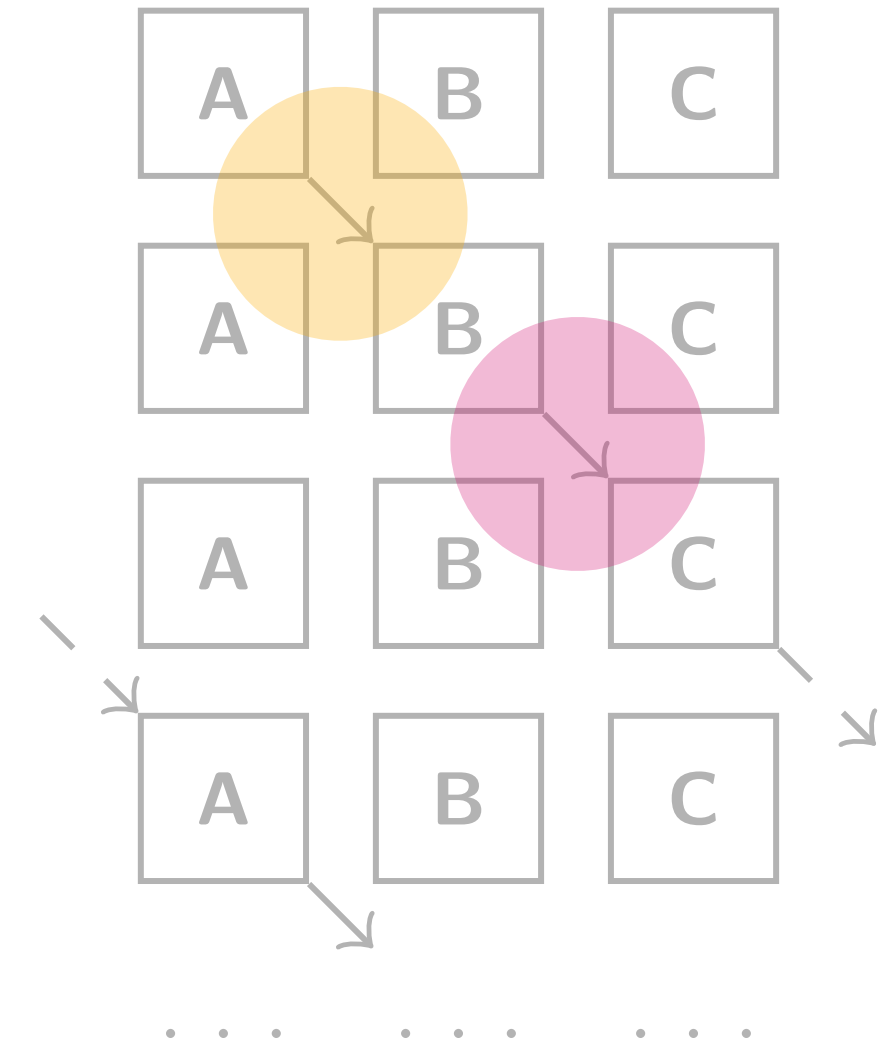
- Global types are inherently *synchronous*
  - Projection provides only one possible ordering



# Challenge

## Asynchronous Orderings

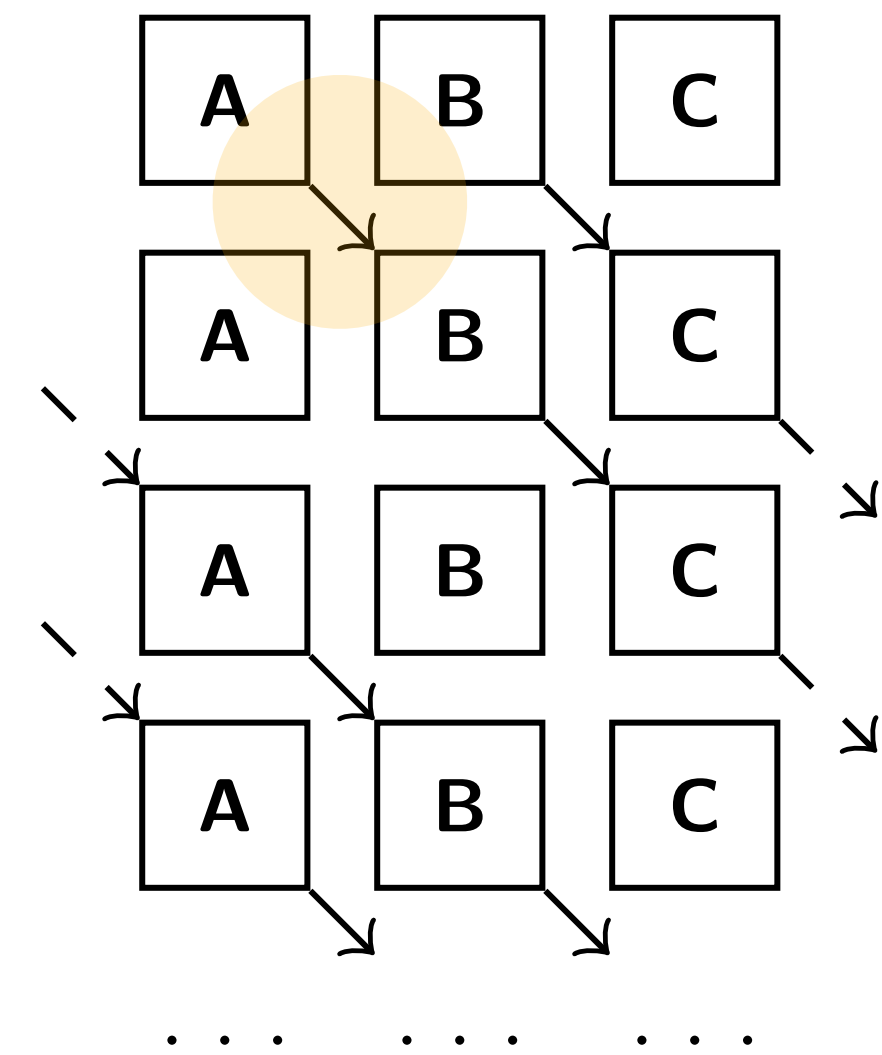
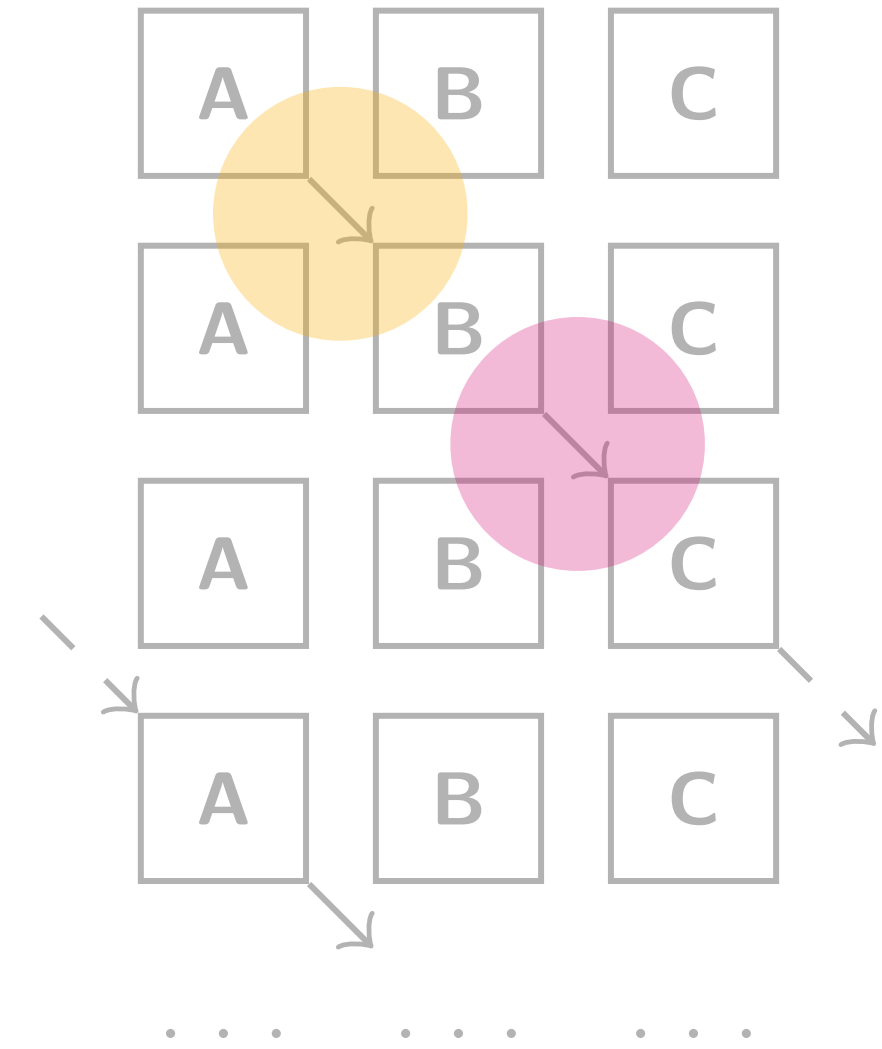
- Global types are inherently **synchronous**
  - Projection provides only one possible ordering
- Interactions can be **reordered** for efficiency while preserving safety



# Challenge

## Asynchronous Orderings

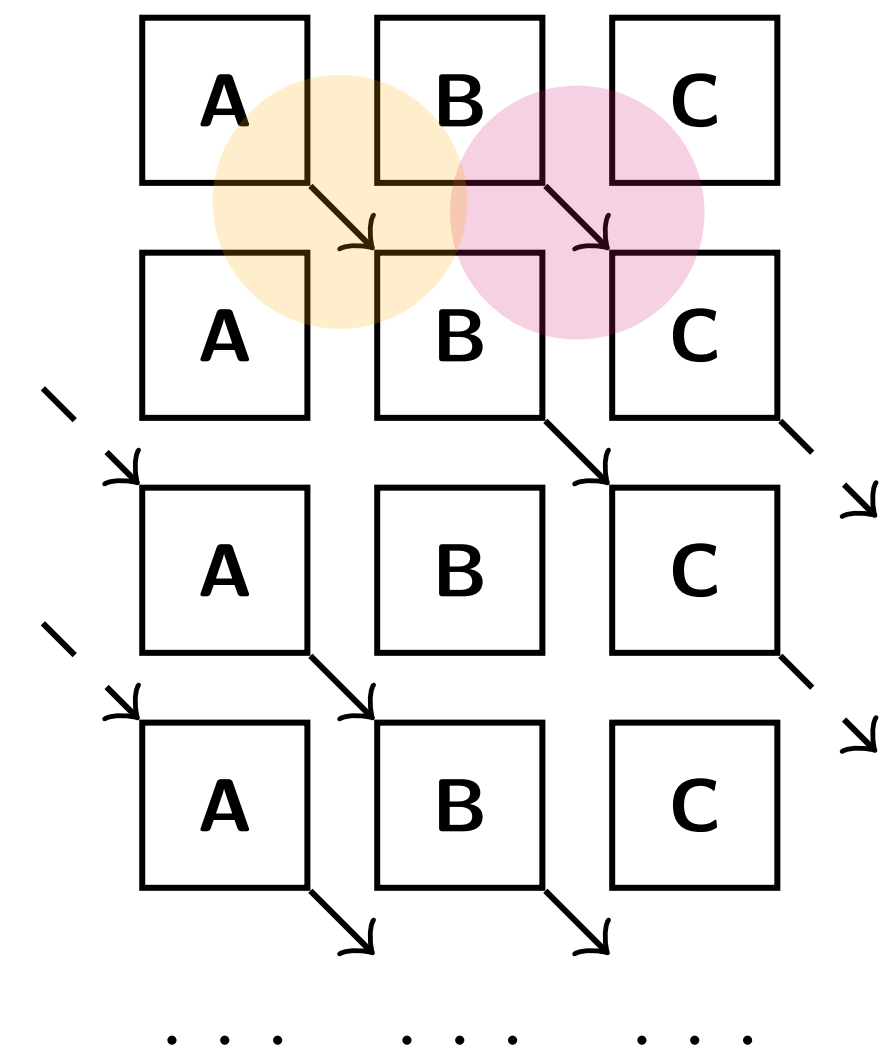
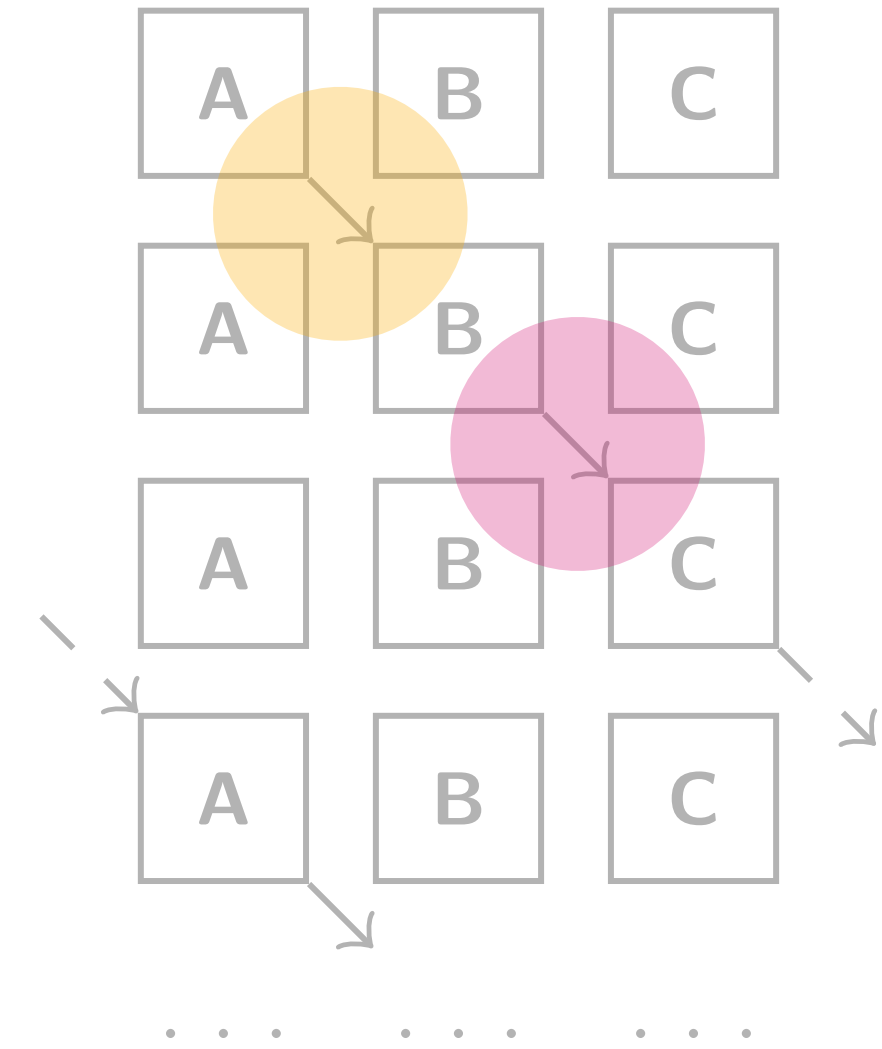
- Global types are inherently **synchronous**
  - Projection provides only one possible ordering
- Interactions can be **reordered** for efficiency while preserving safety



# Challenge

## Asynchronous Orderings

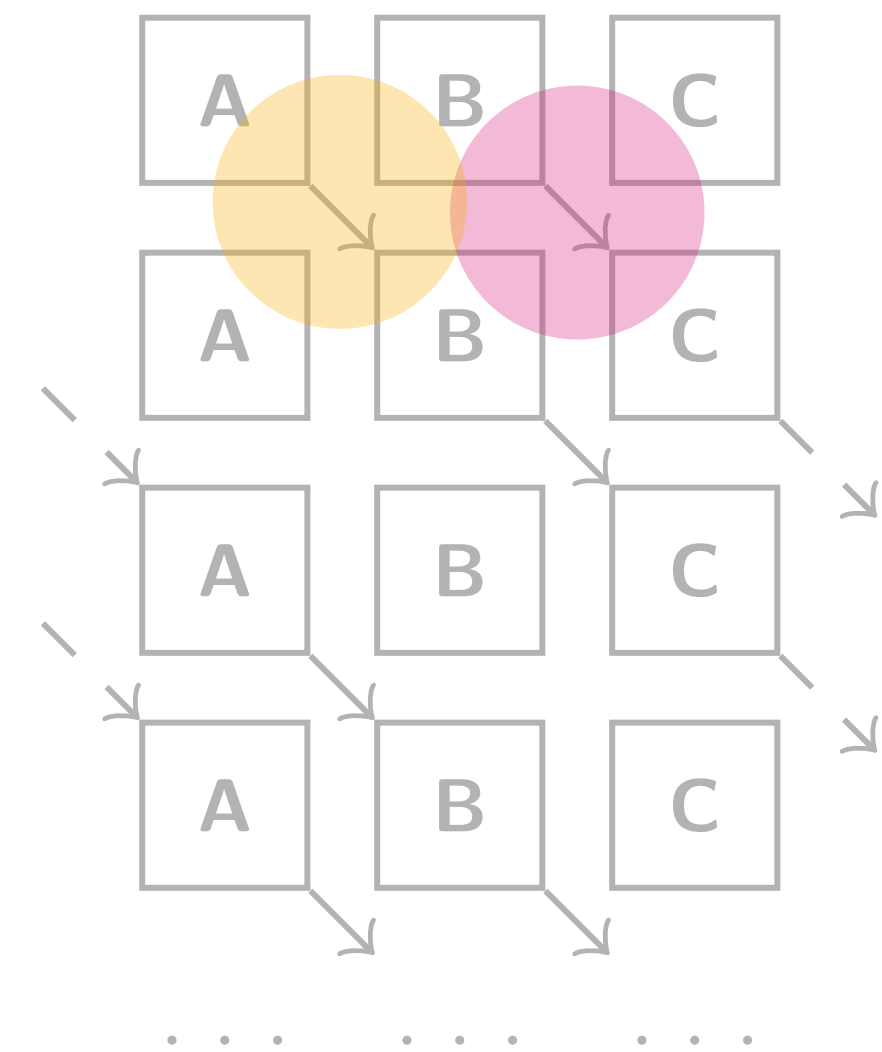
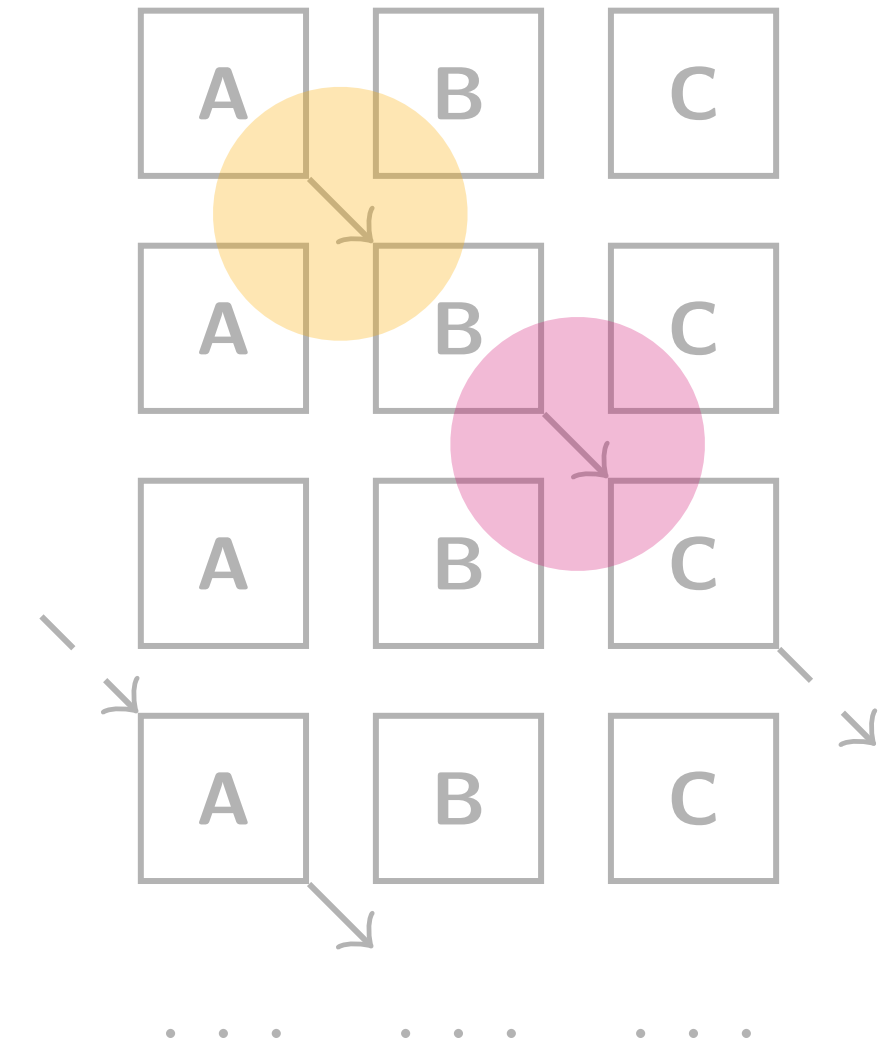
- Global types are inherently **synchronous**
  - Projection provides only one possible ordering
- Interactions can be **reordered** for efficiency while preserving safety



# Challenge

## Asynchronous Orderings

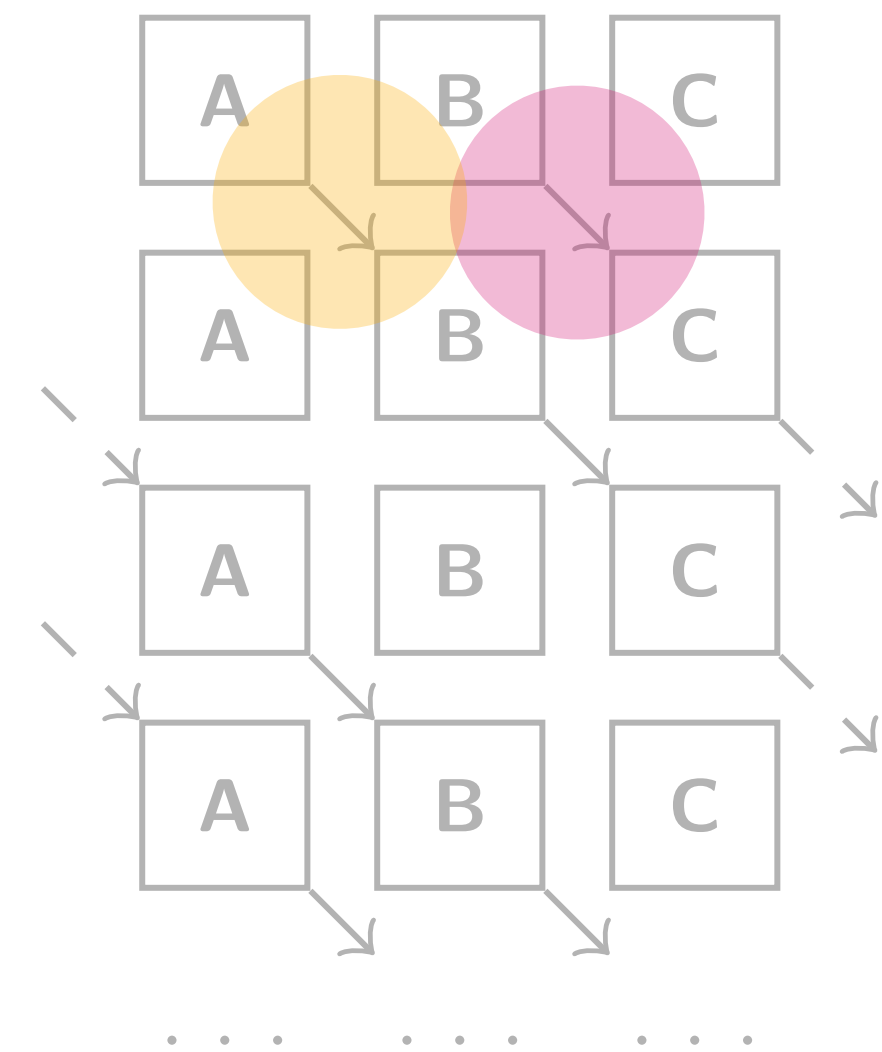
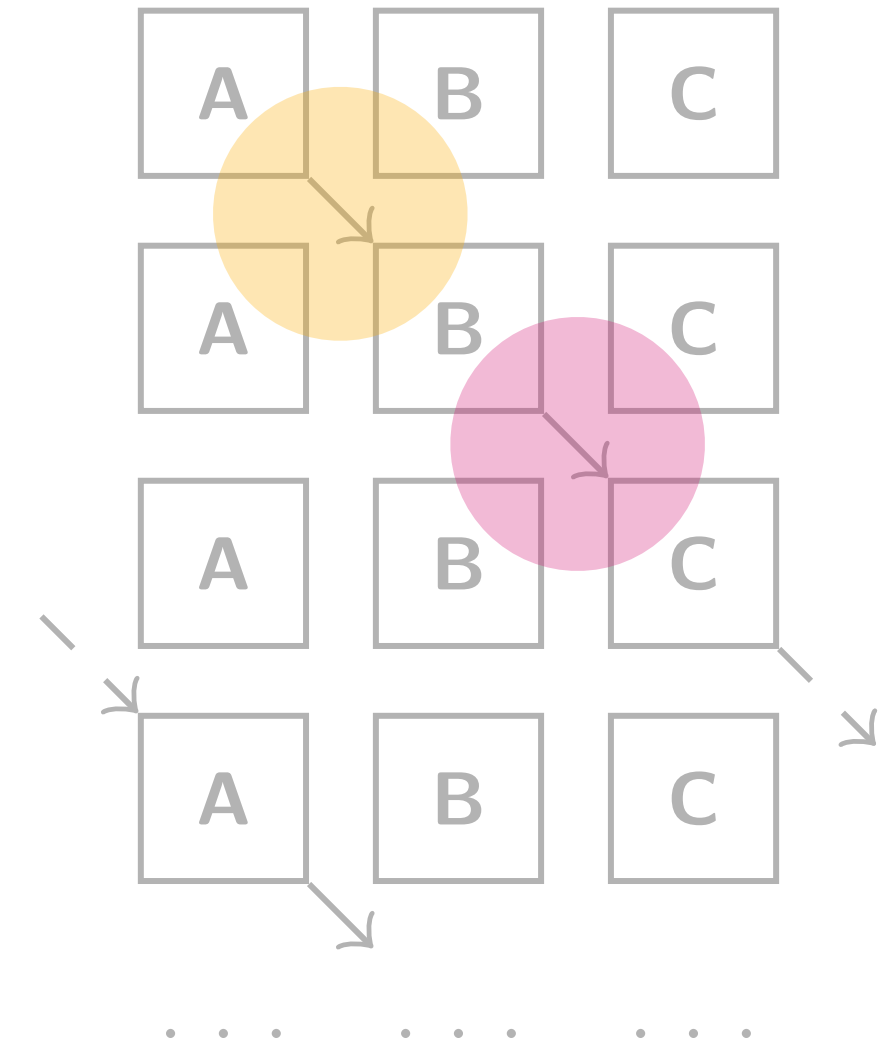
- Global types are inherently **synchronous**
  - Projection provides only one possible ordering
- Interactions can be **reordered** for efficiency while preserving safety
  1. Data **dependencies** must be preserved



# Challenge

## Asynchronous Orderings

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



# Rumpsteak Framework

## Three Approaches

**G** Global Type

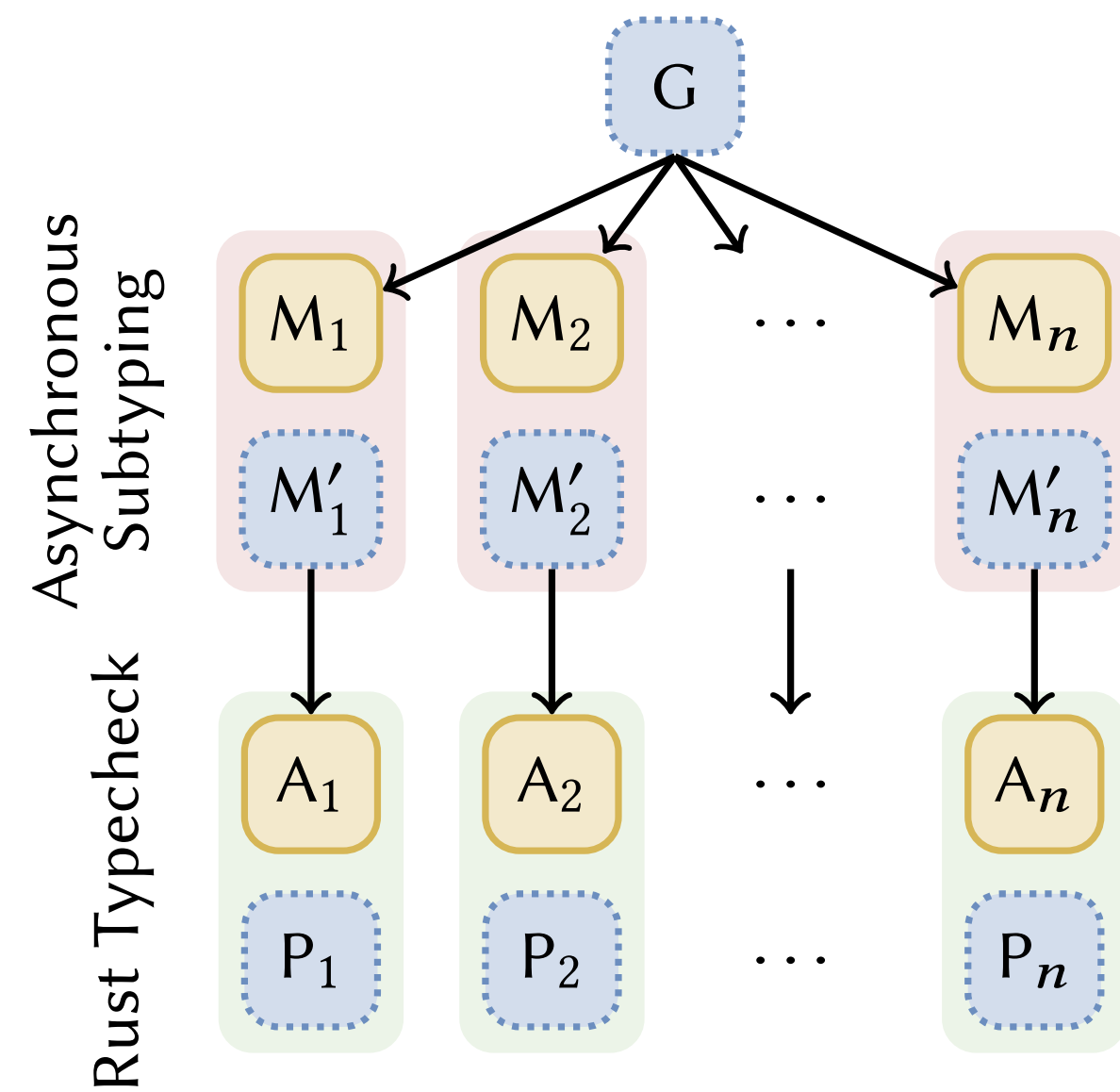
**M** Finite State Machine (FSM)

**M'** Optimised FSM

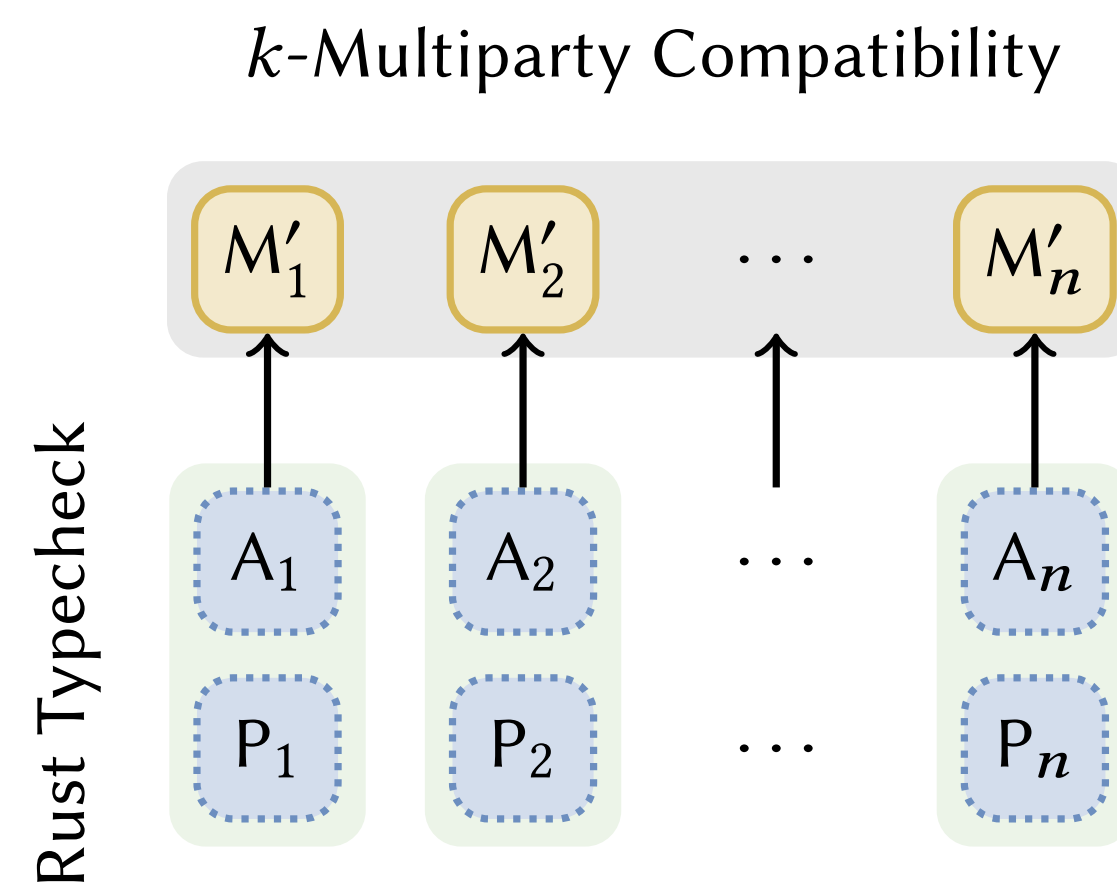
**A** Rust API

**P** Rust Process

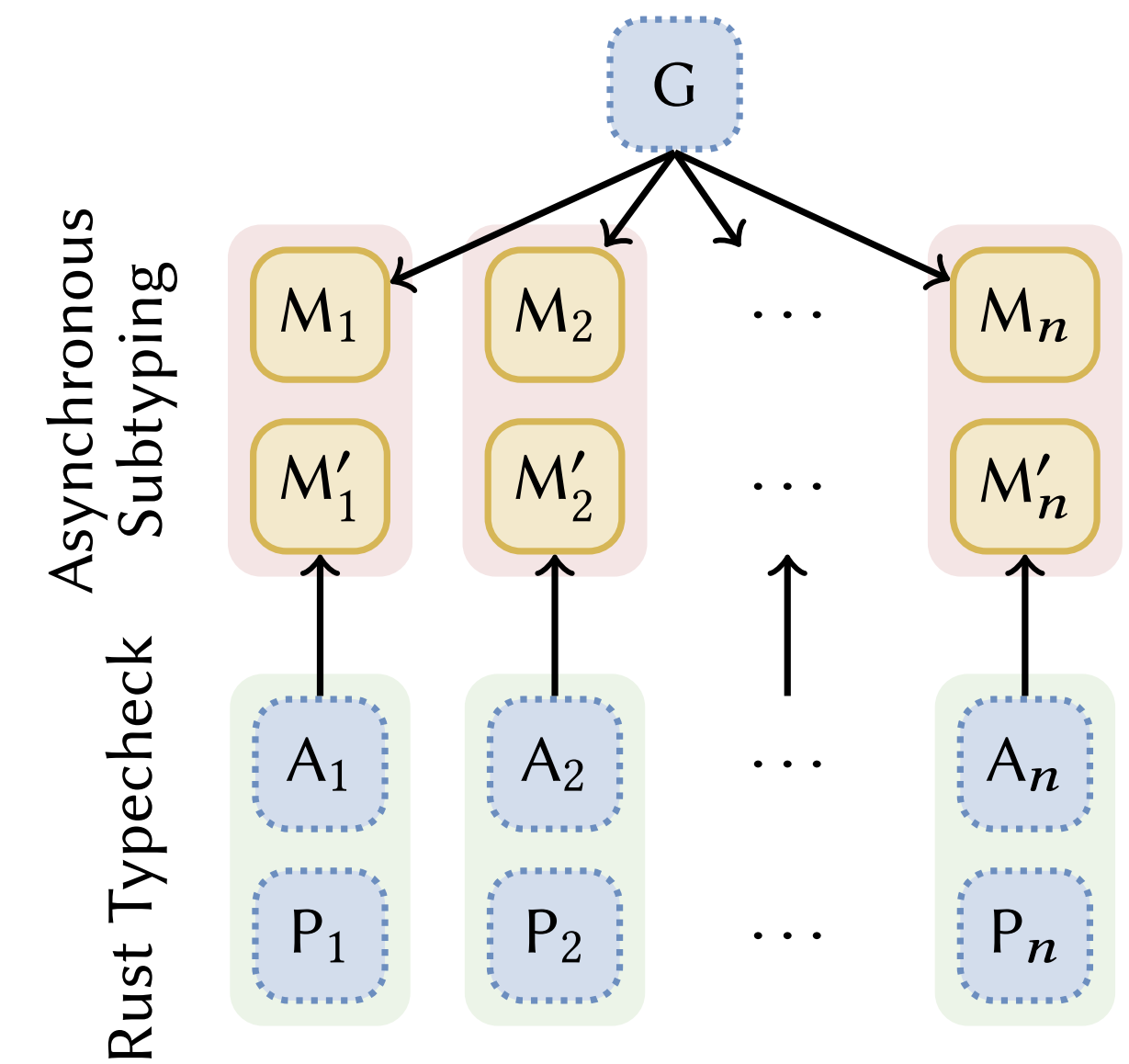
 User-Written     Generated



(a) Top-down



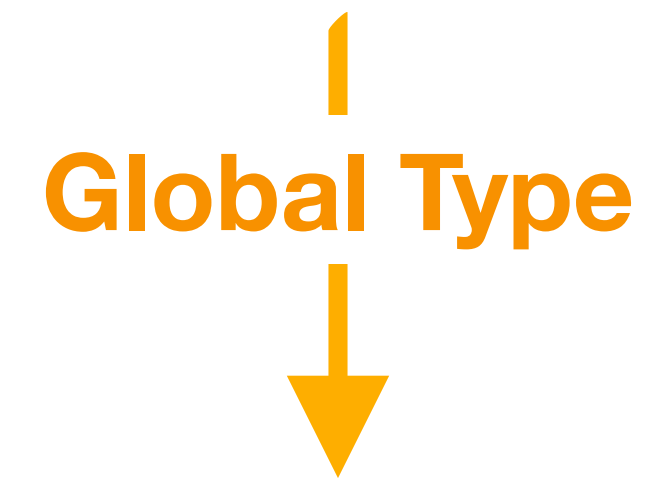
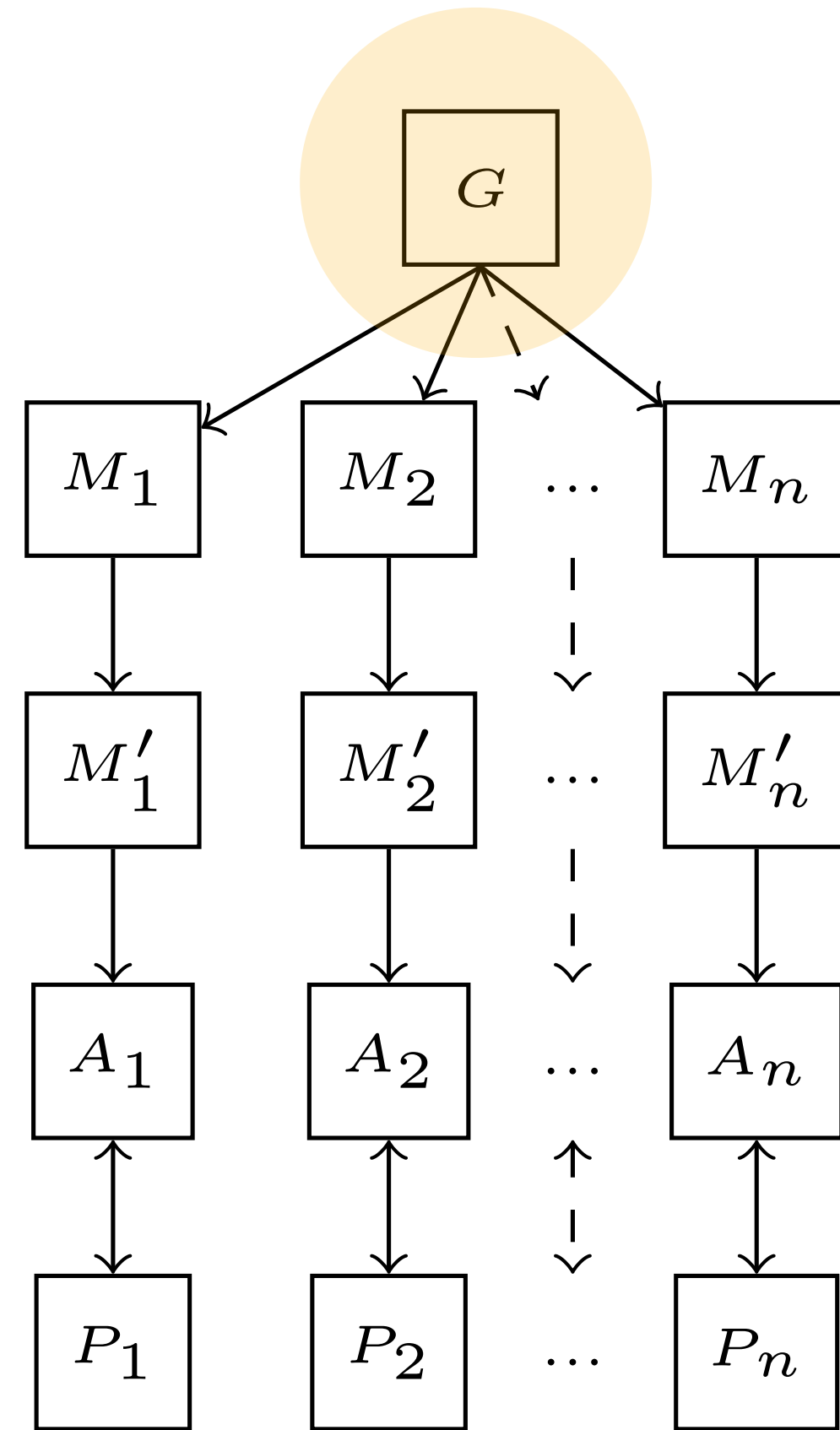
(b) Bottom-up



(c) Hybrid

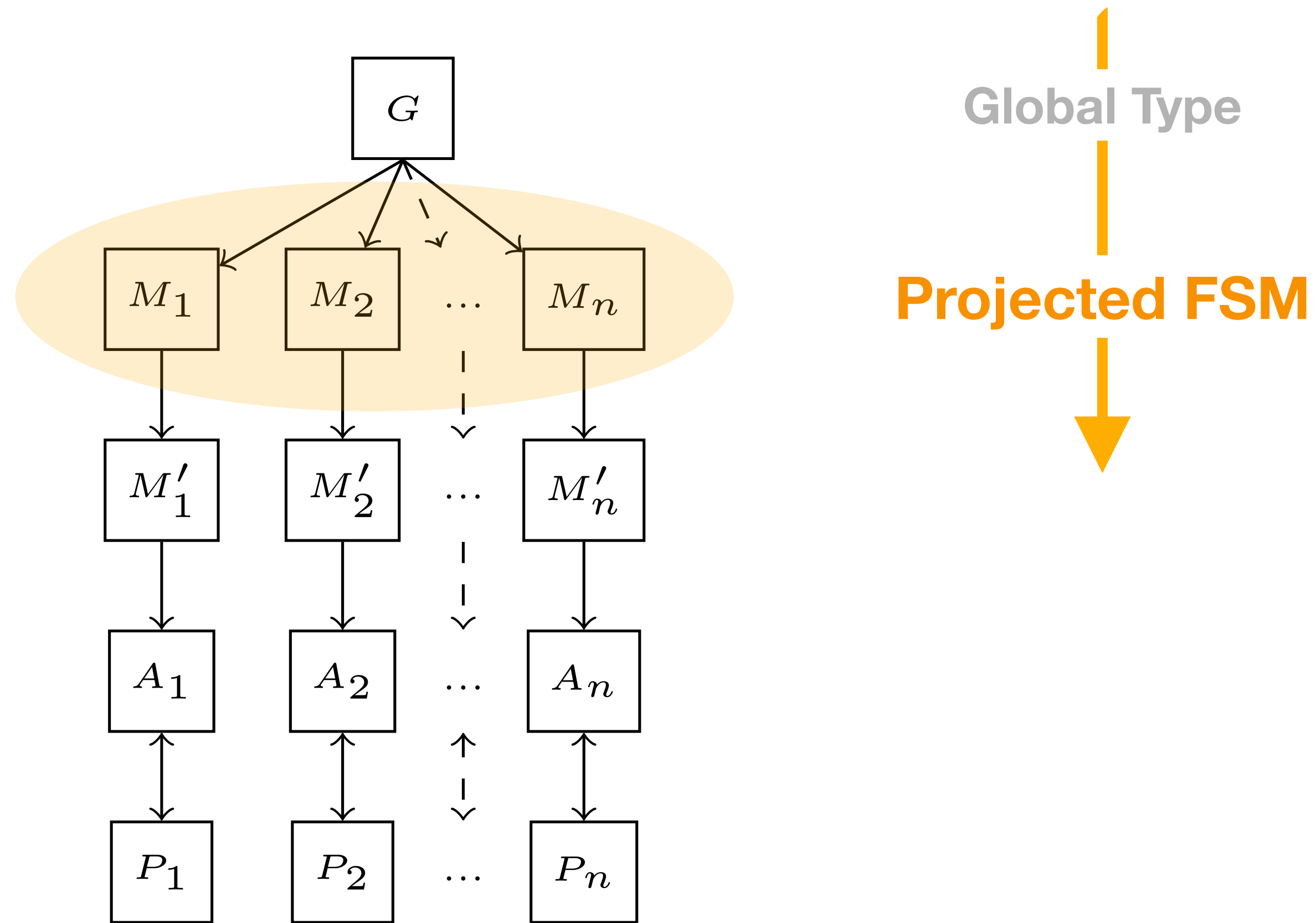
# Workflow

## Top-Down Approach



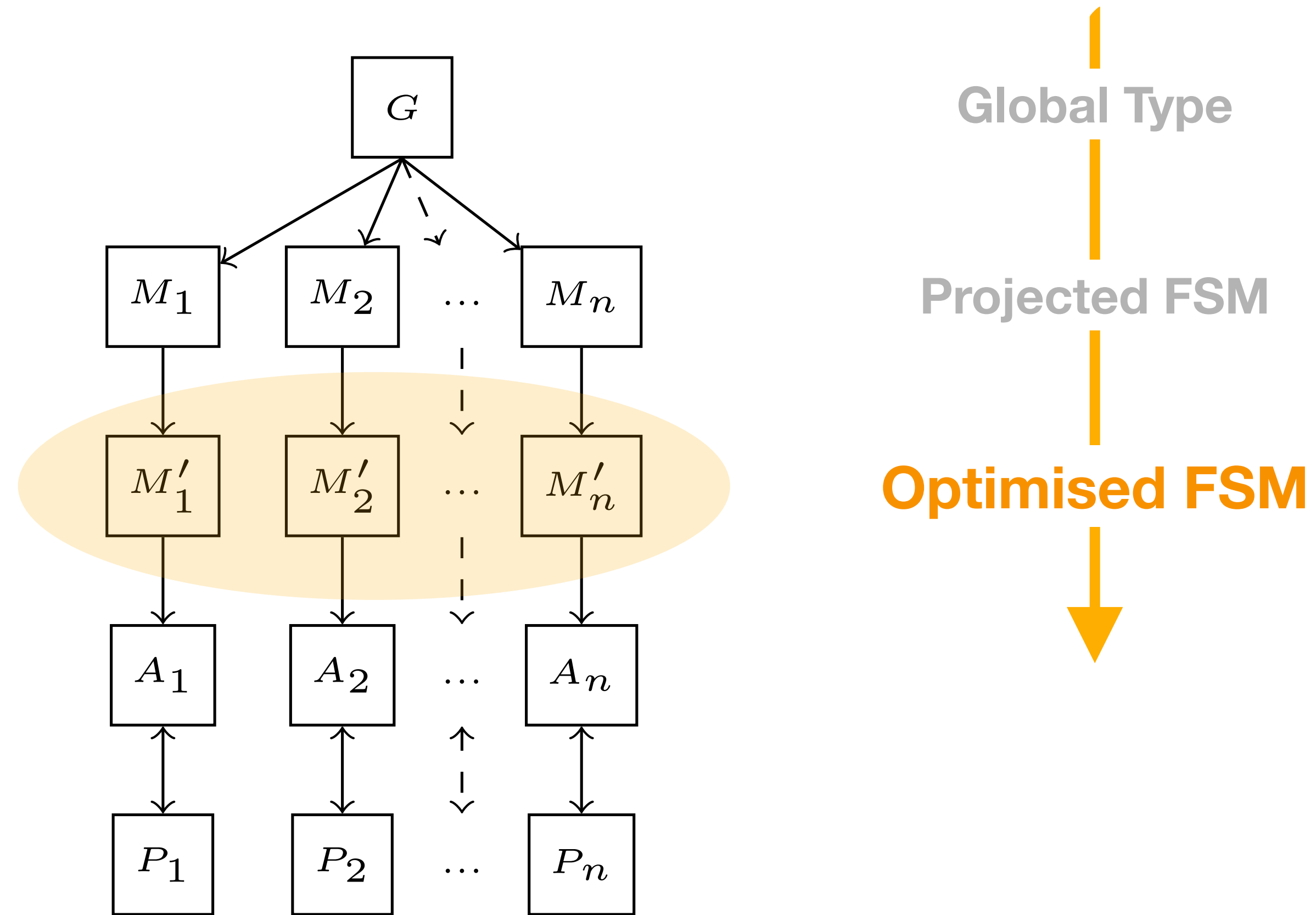
# Workflow

## Top-Down Approach



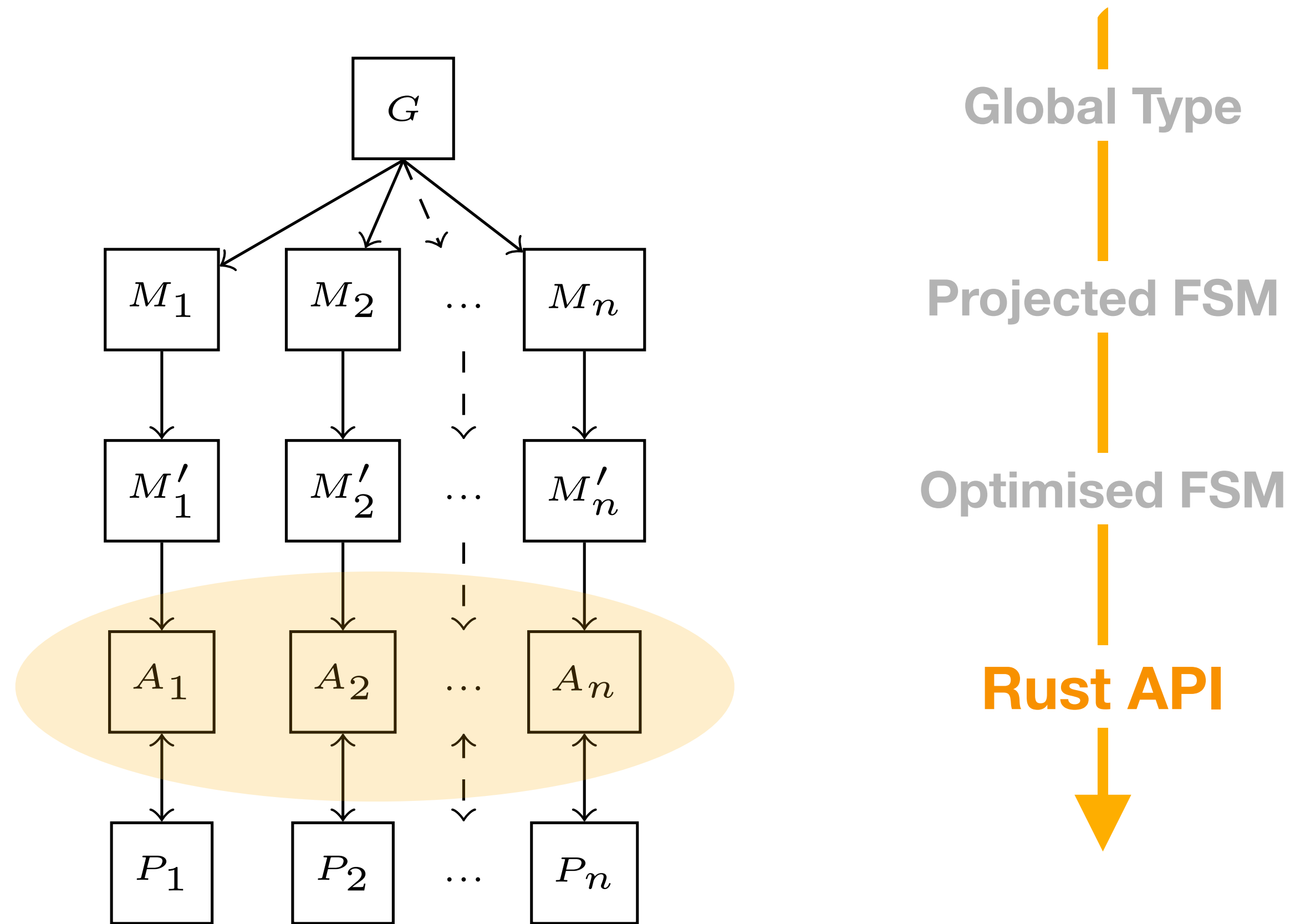
# Workflow

## Top-Down Approach



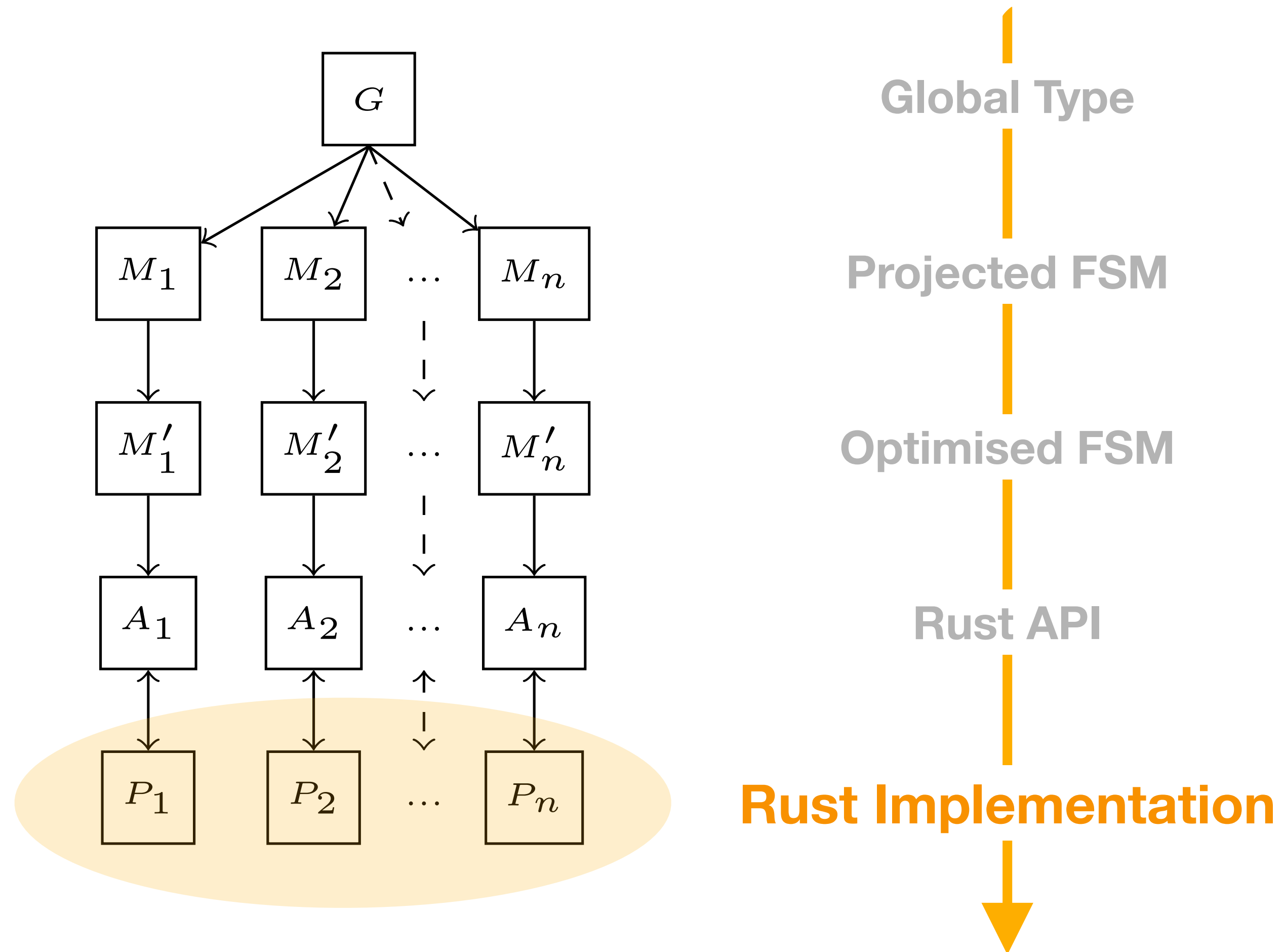
# Workflow

## Top-Down Approach



# Workflow

## Top-Down Approach



# Ring Protocol

## Example

### Global Type

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

# Ring Protocol

## Example

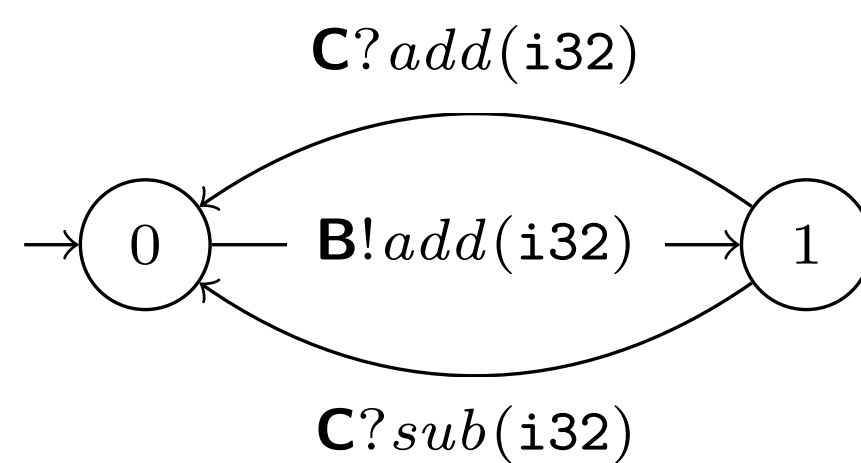
$$G = \mu t. \mathbf{A} \rightarrow \mathbf{B} : \left\{ \begin{array}{l} \mathit{add}(\mathit{i32}).\mathbf{B} \rightarrow \mathbf{C} : \left\{ \begin{array}{l} \mathit{add}(\mathit{i32}).\mathbf{C} \rightarrow \mathbf{A} : \{ \mathit{add}(\mathit{i32}).\mathbf{t} \} \\ \mathit{sub}(\mathit{i32}).\mathbf{C}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

# Ring Protocol

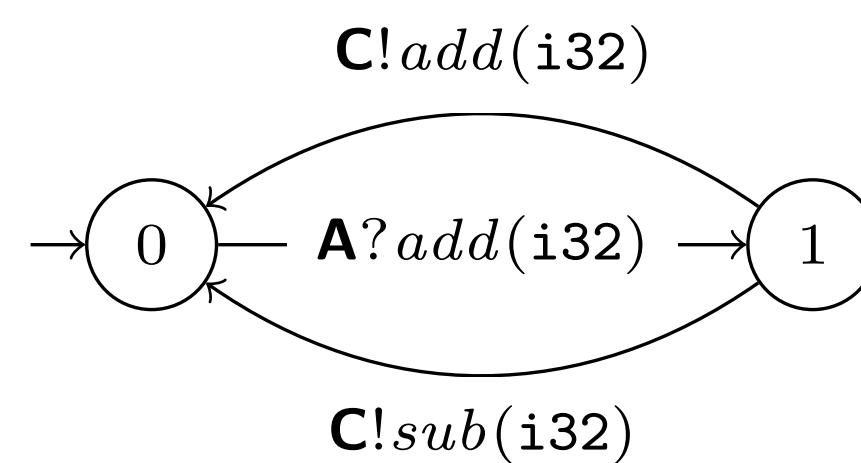
## Example

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

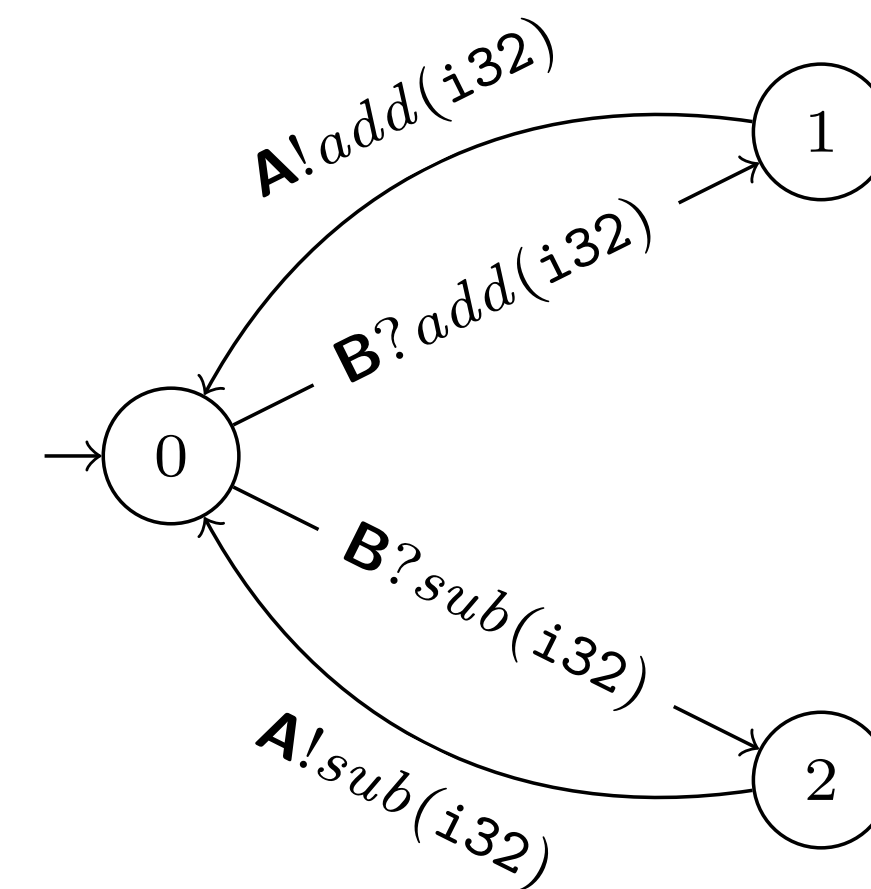
PROJECTION



PROJECTION

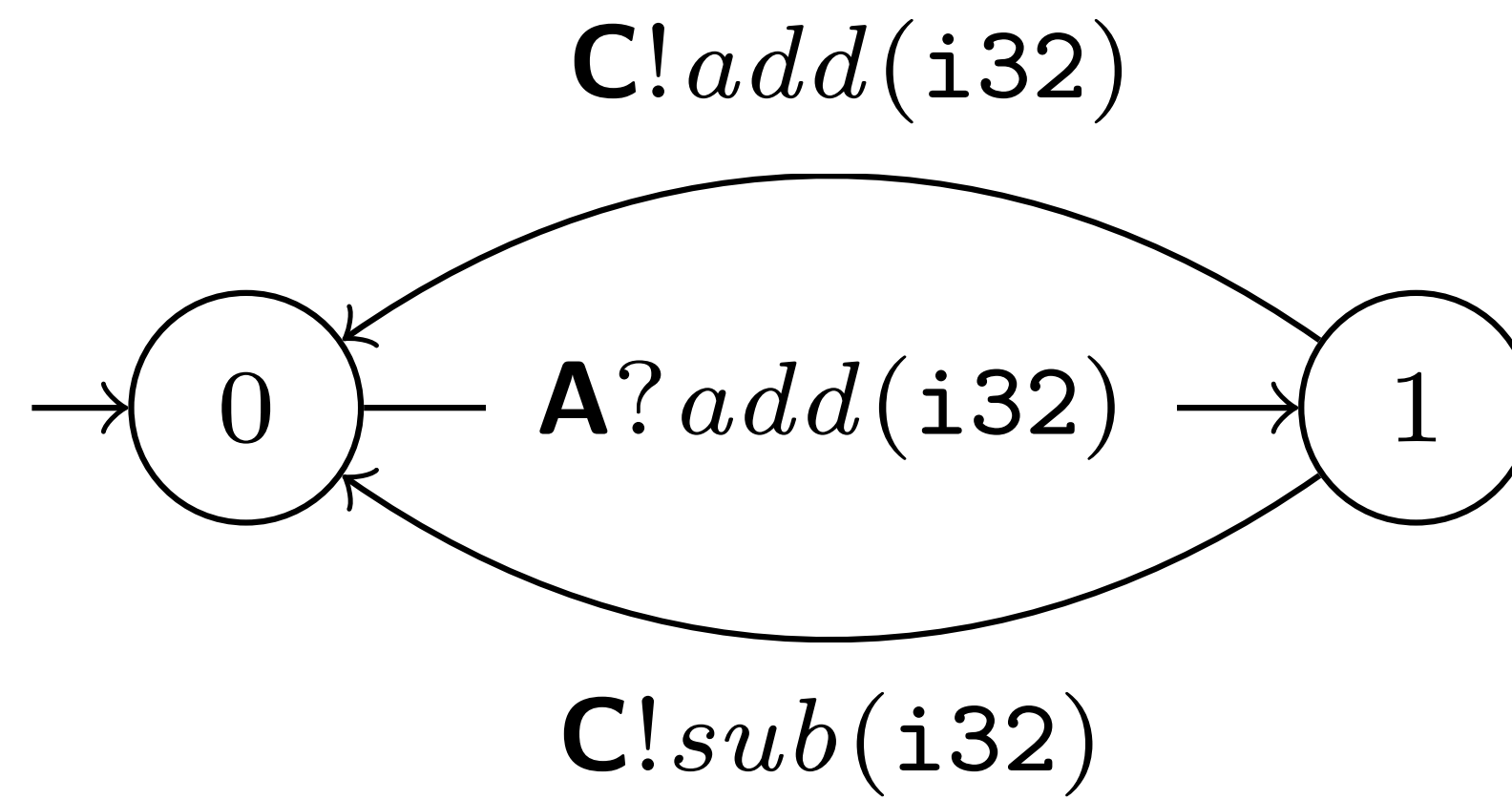


PROJECTION



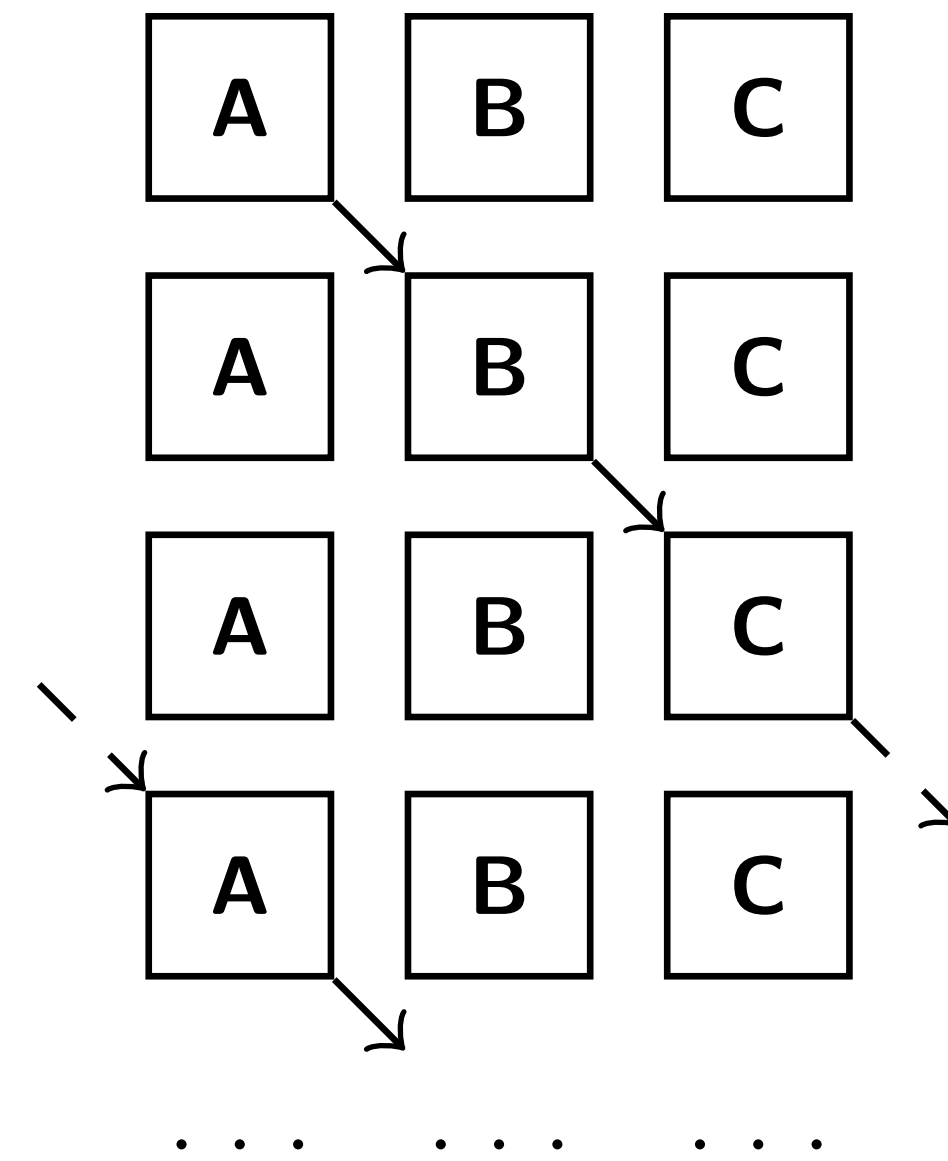
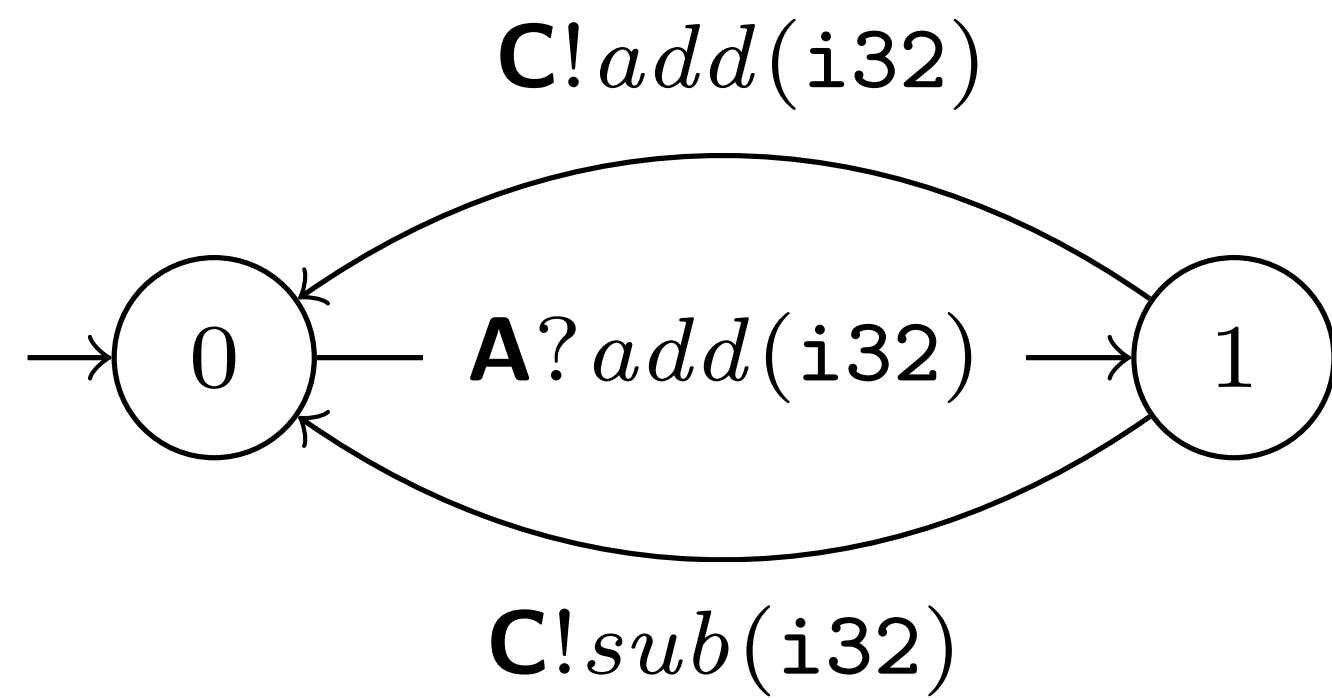
# Ring Protocol

## Example



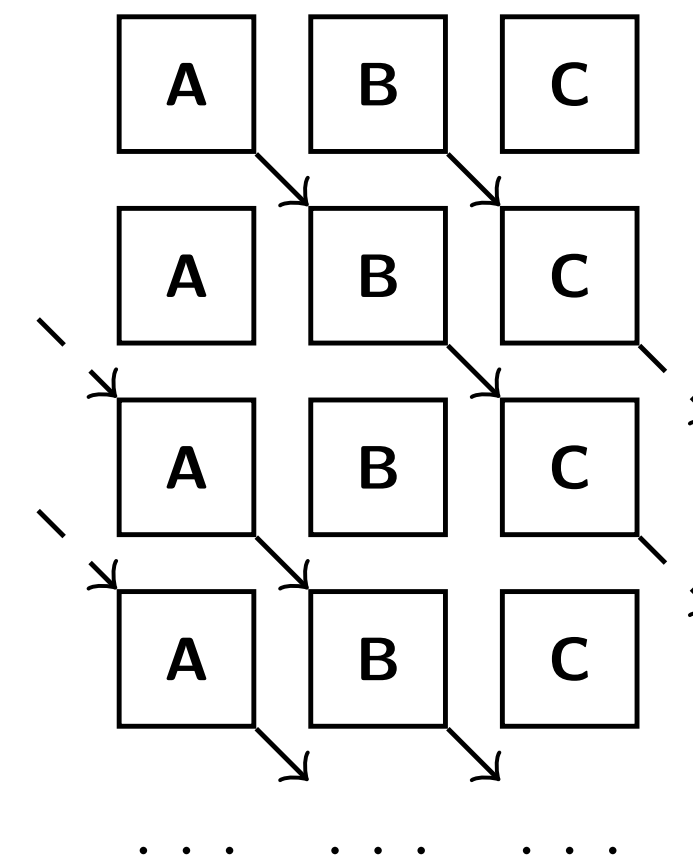
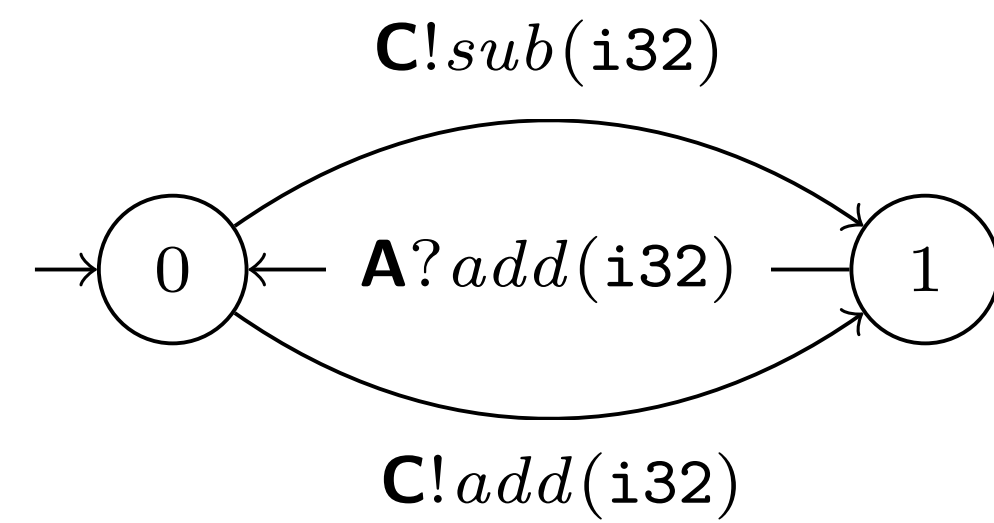
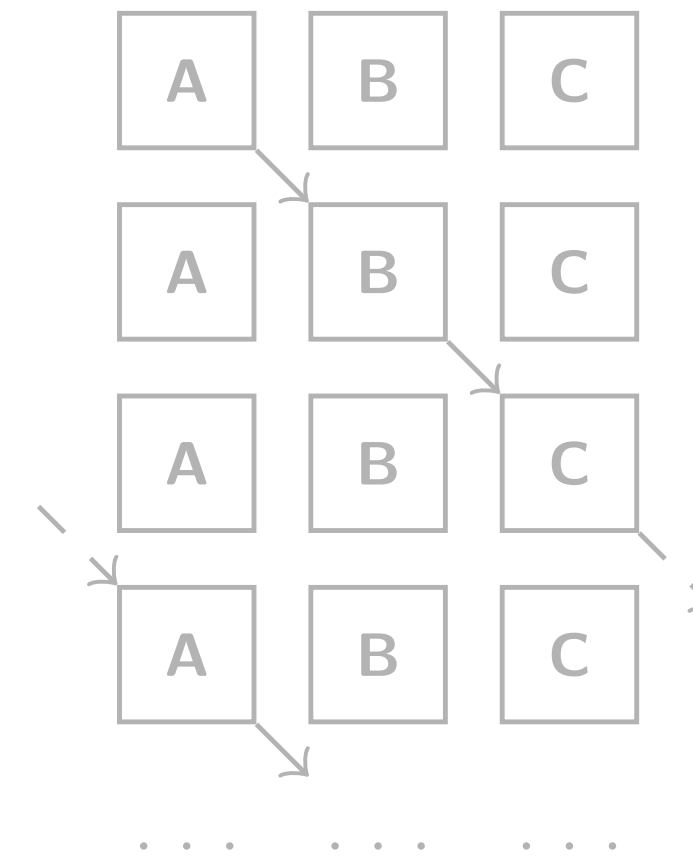
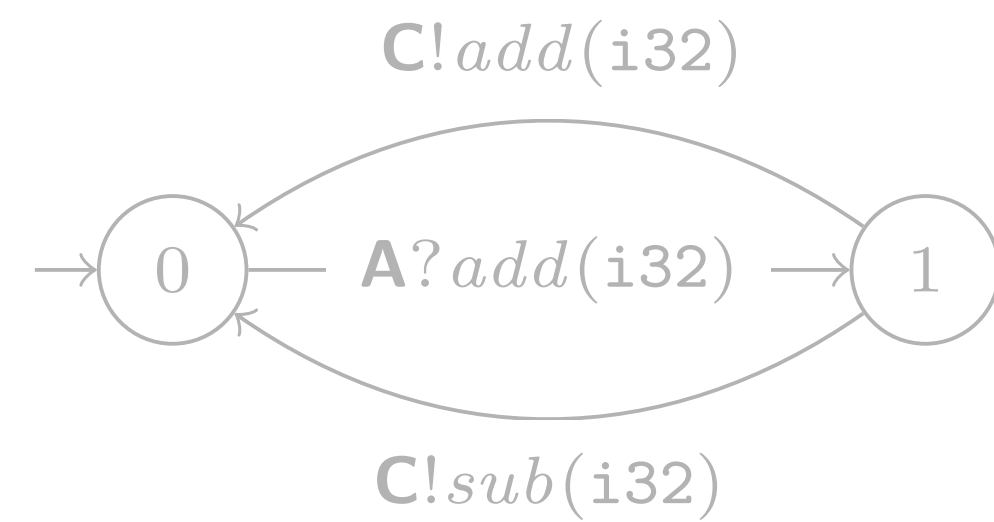
# Ring Protocol

## Example



# Ring Protocol

## Example



# vScr An Extensible Toolchain for Multiparty Session Types

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

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

**Global protocol**

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

**Local types**

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

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

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

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

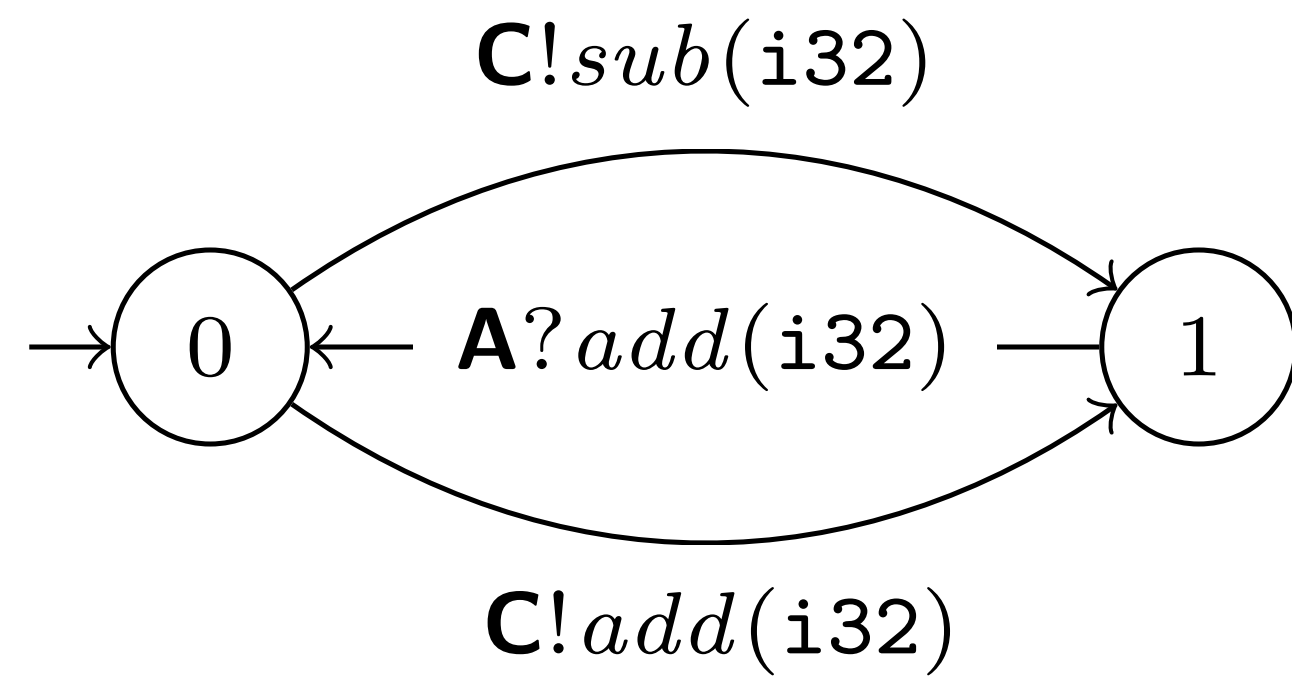
# Scribble

## Protocol Description Language

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

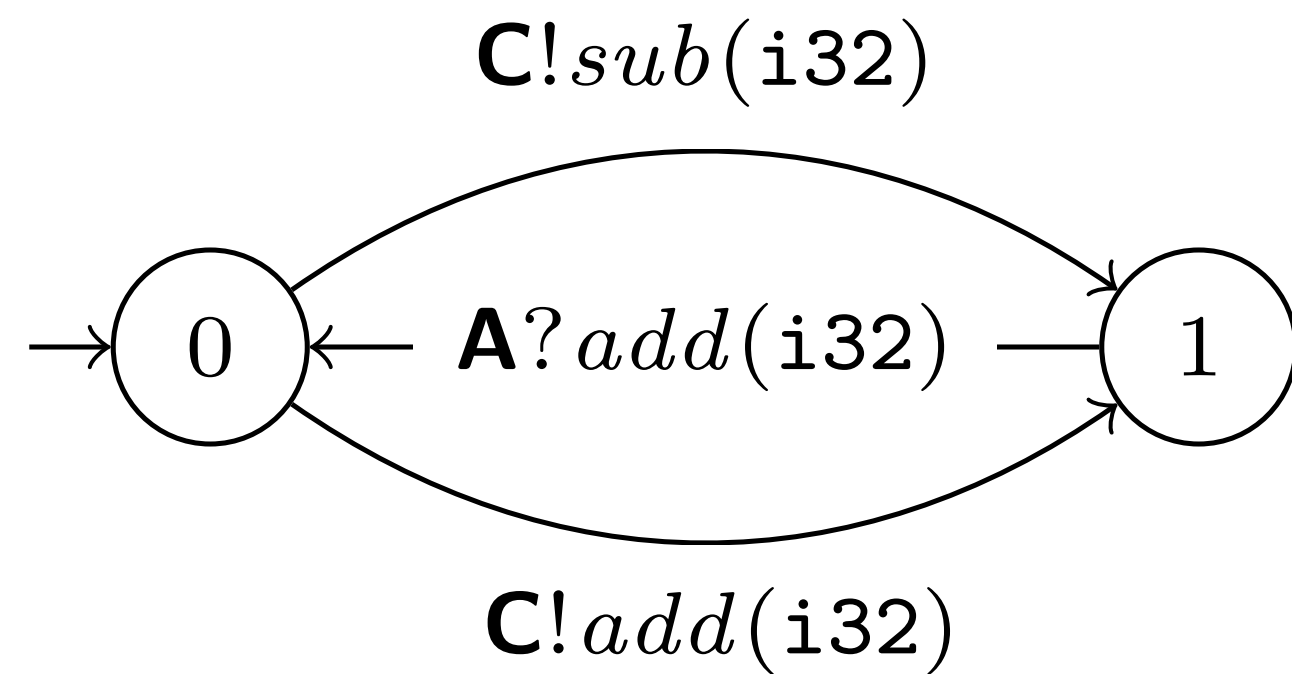
# Ring Protocol

## Rust API



# Ring Protocol

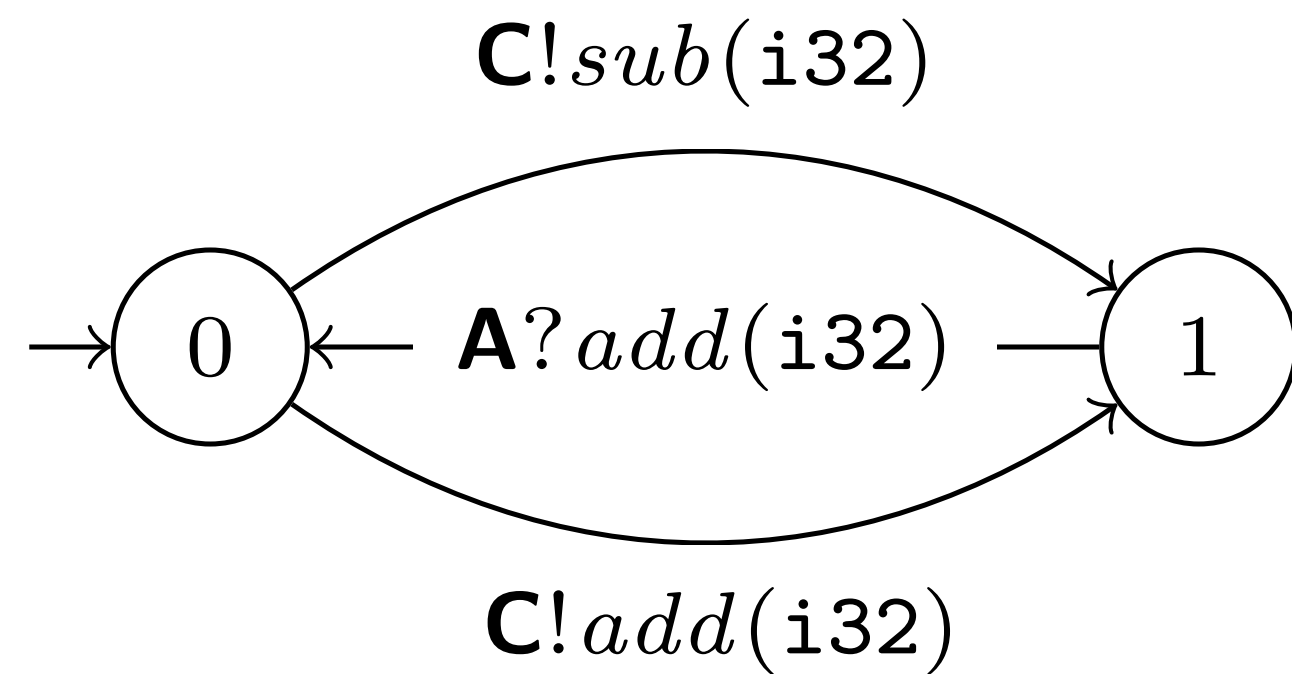
## Rust API



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

# Ring Protocol

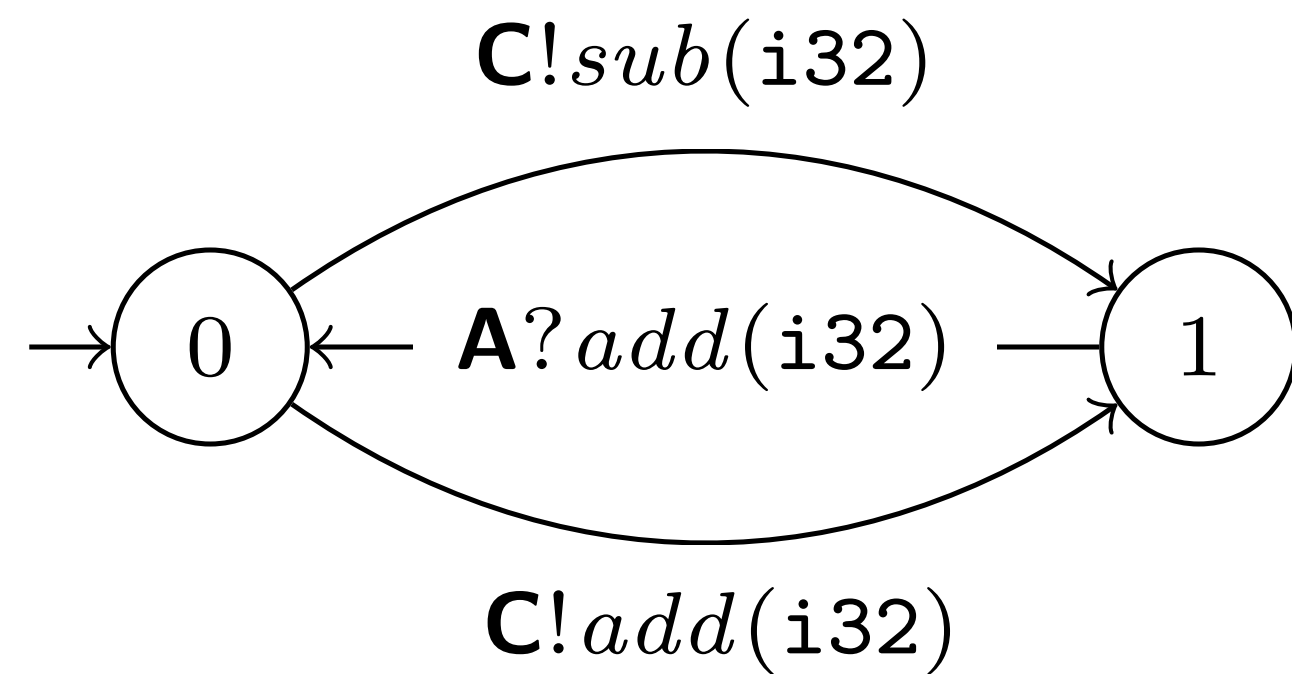
## Rust API



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

# Ring Protocol

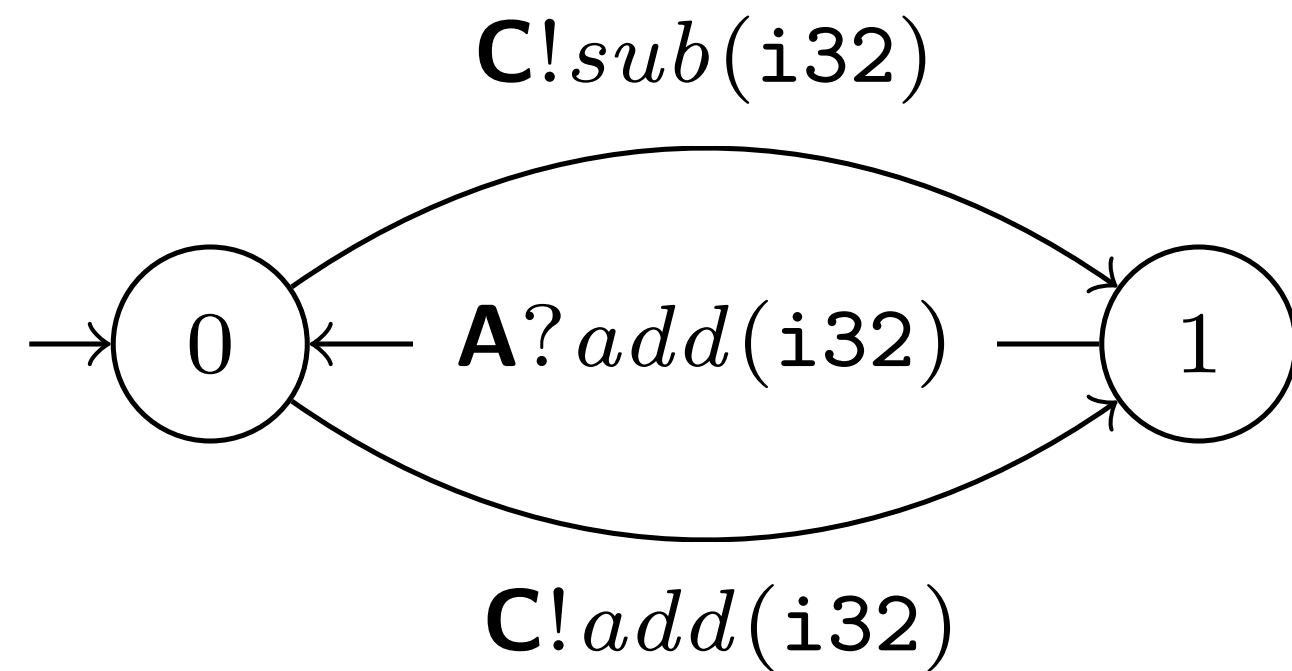
## Rust API



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

# Ring Protocol

## Rust API



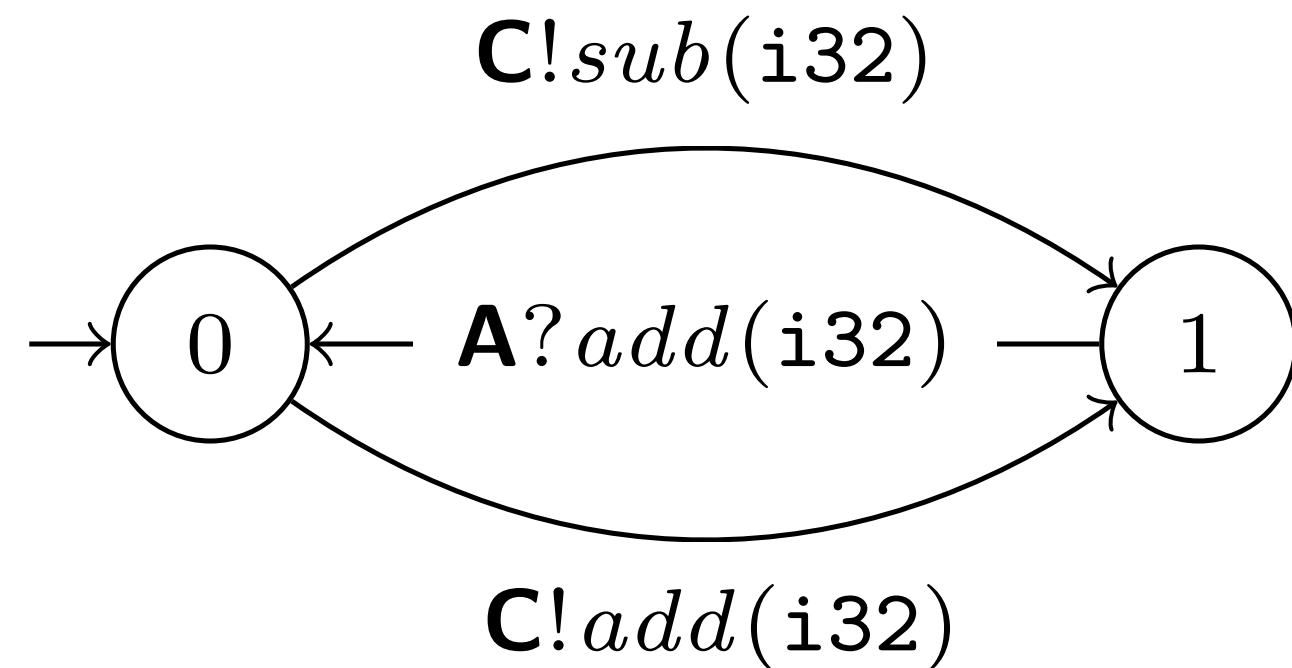
```
#[derive(Role)]  
#[message(Label)]  
struct B(#[route(A)] Receiver, #[route(C)] Sender);
```

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

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

# Ring Protocol

## Rust API



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

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

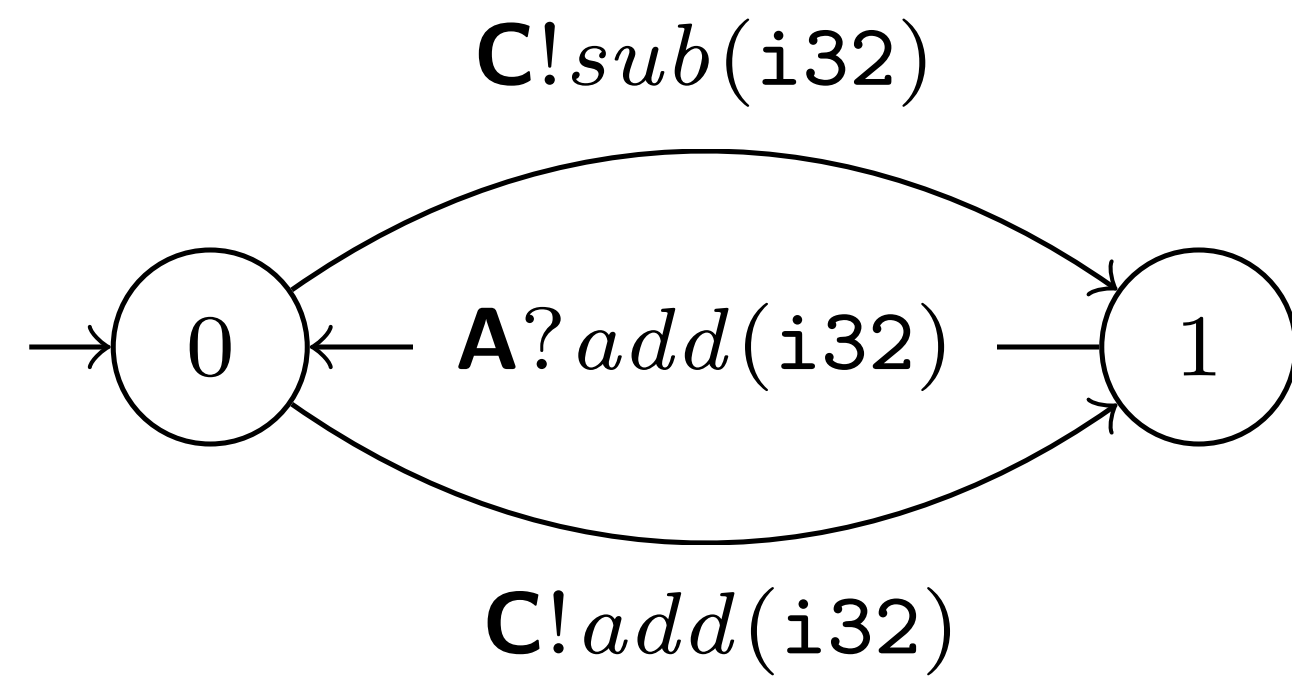
struct Add(i32);
struct Sub(i32);

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

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

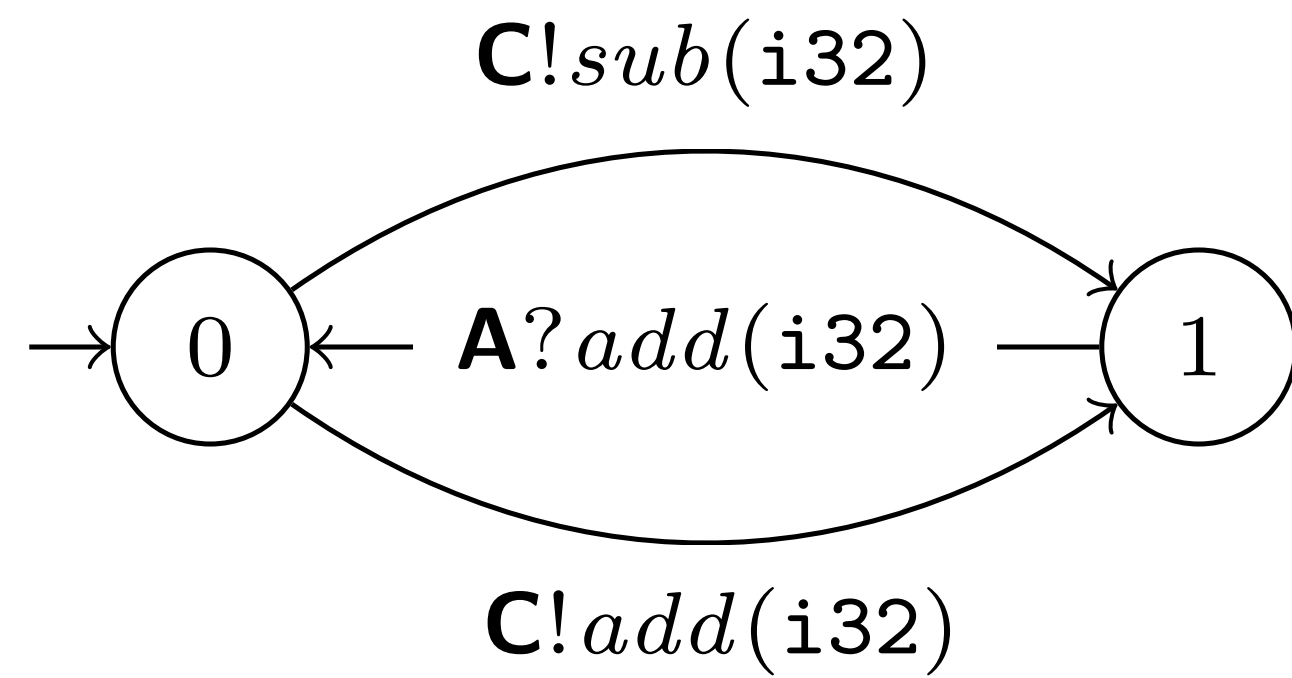
# Ring Protocol

## Rust API



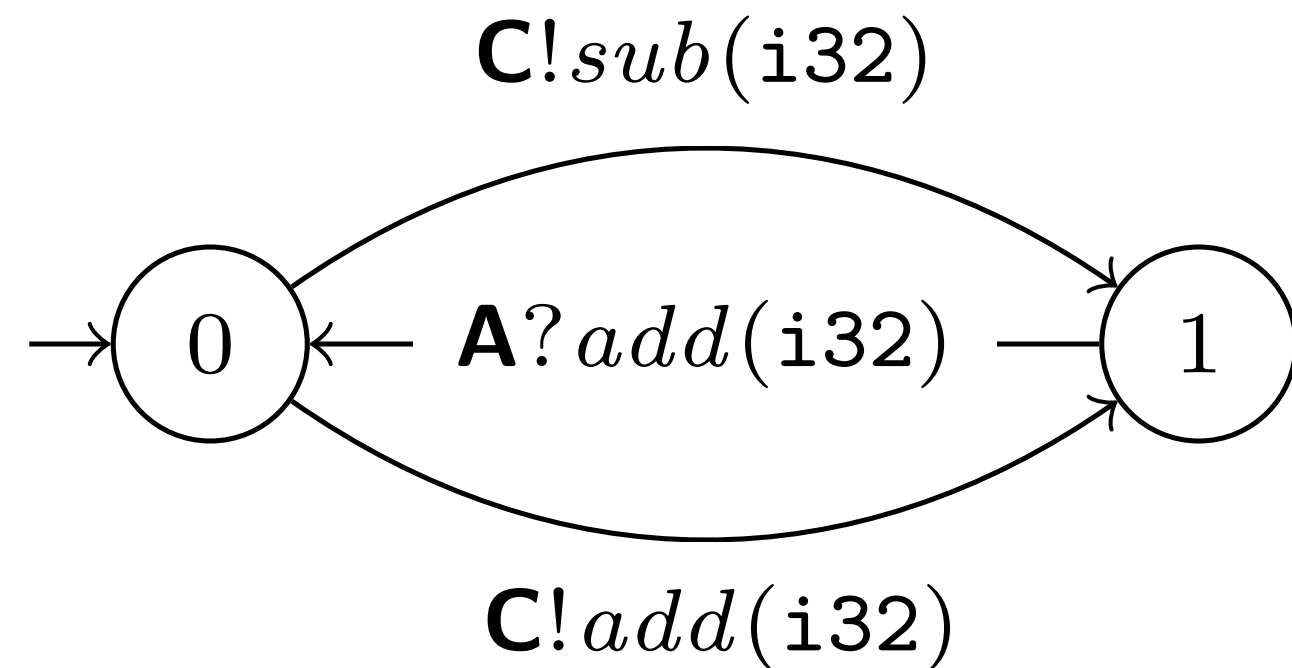
# Ring Protocol

## Implementation



# Ring Protocol

## Implementation

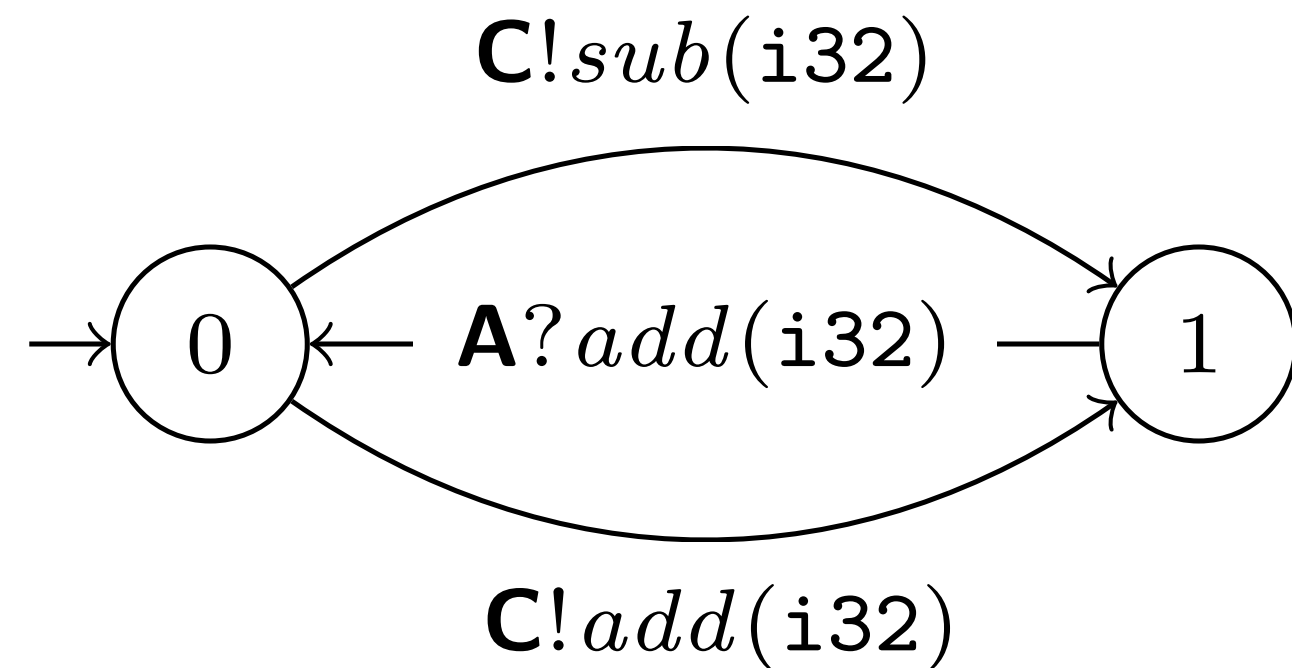


```
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

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

# Ring Protocol

## Implementation

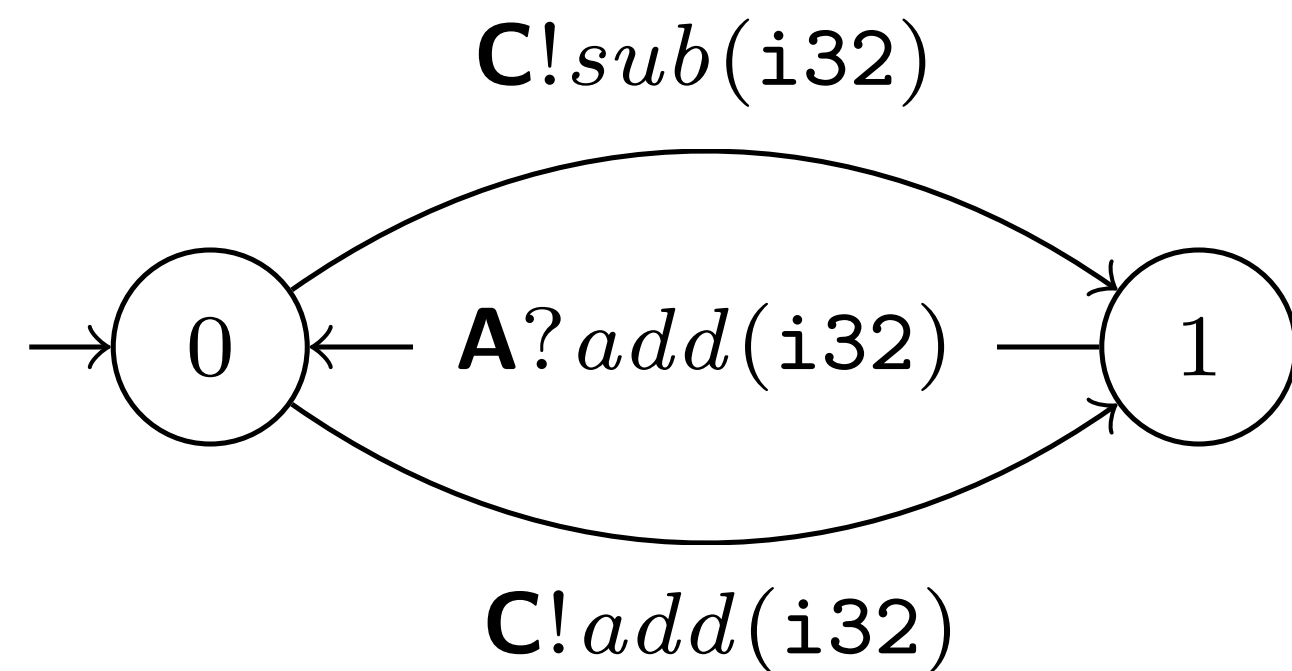


```
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

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

# Ring Protocol

## Implementation

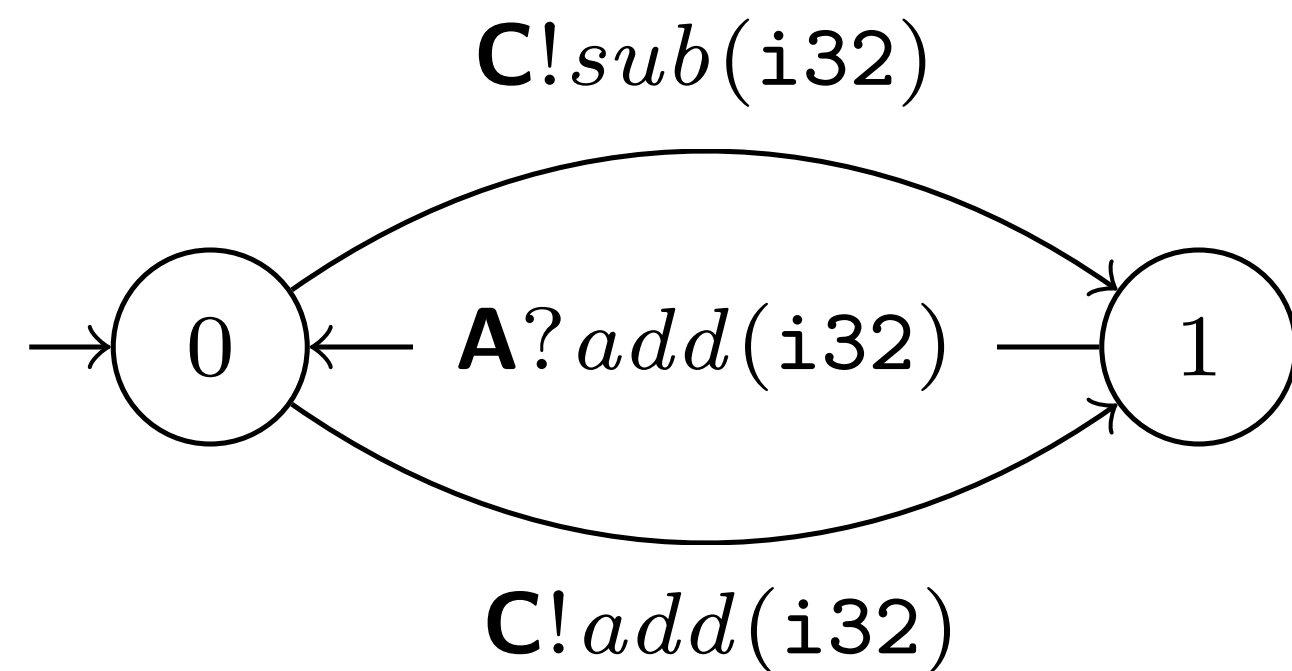


```
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

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

# Ring Protocol

## Implementation

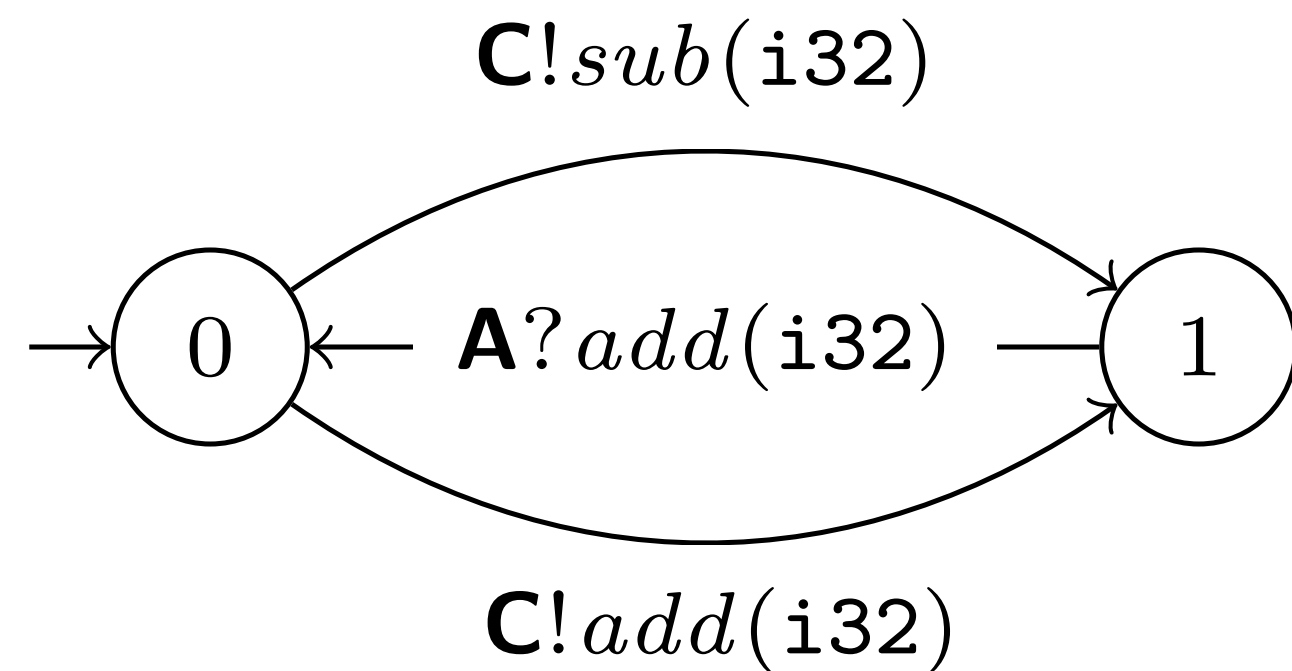


```
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

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

# Ring Protocol

## Implementation

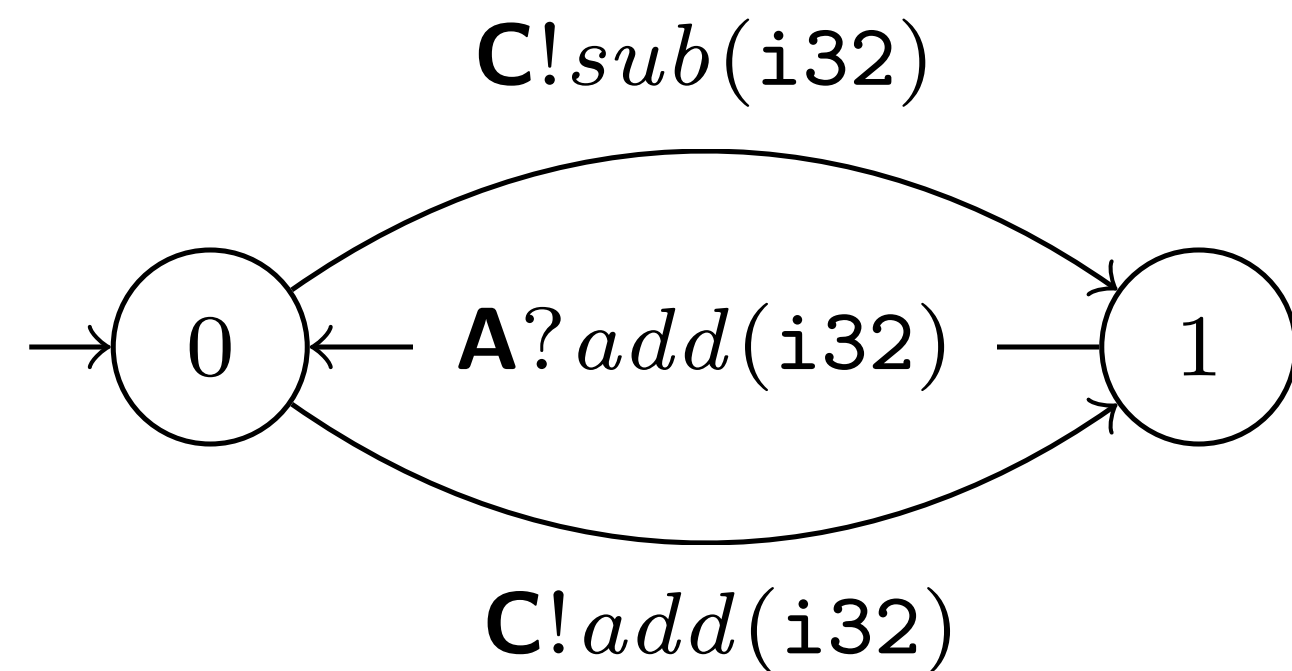


```
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

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

# Ring Protocol

## Implementation

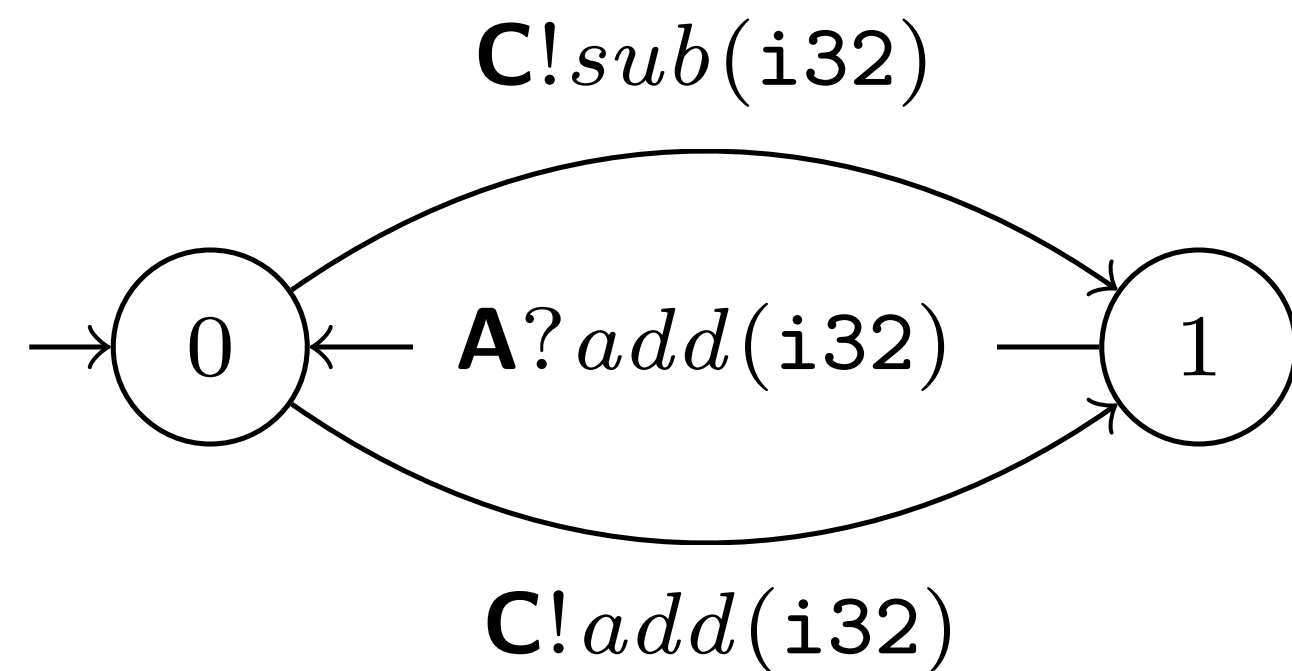


```
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

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

# Ring Protocol

## Implementation

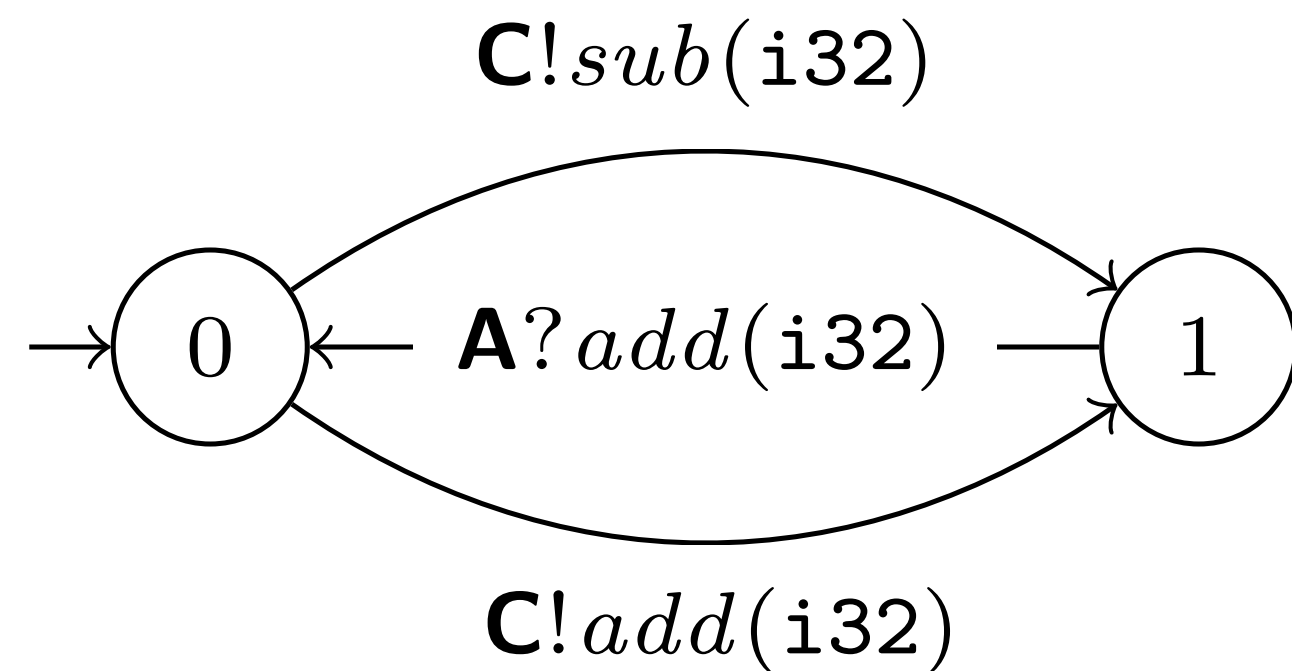


```
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

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

# Ring Protocol

## Implementation

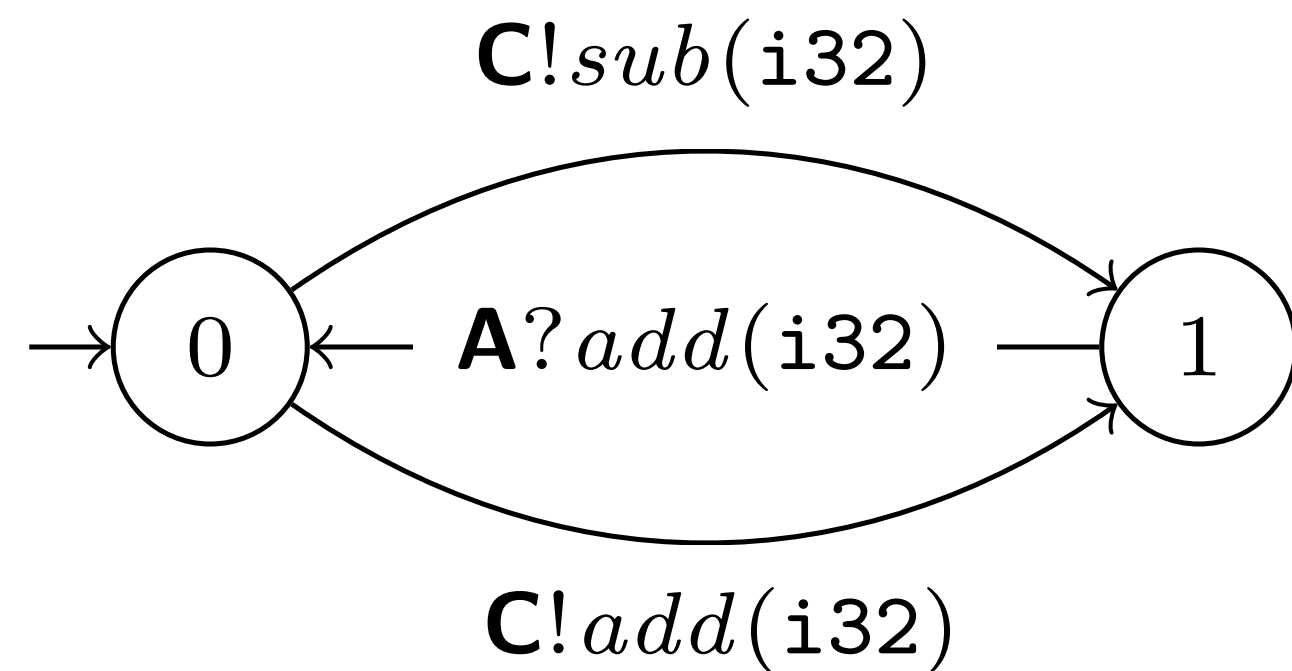


```
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

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

# Ring Protocol

## Implementation

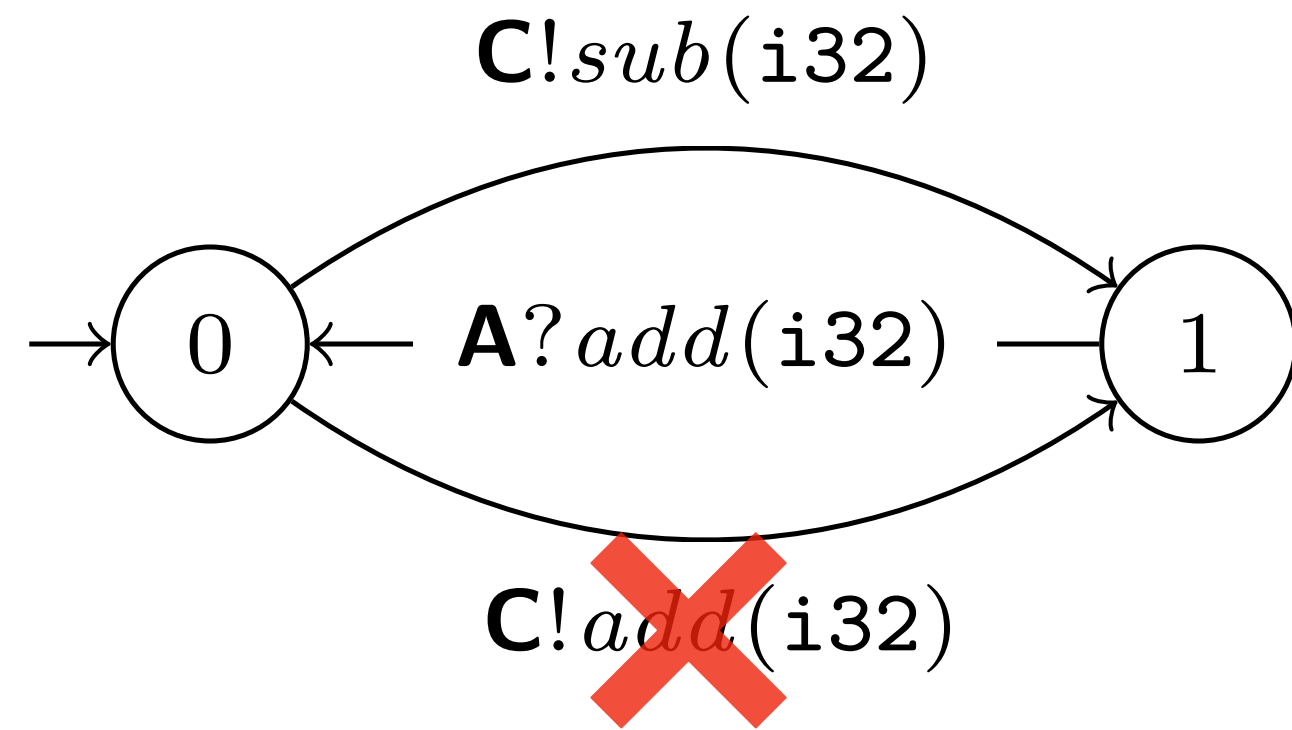


```
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

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

# Ring Protocol

## Implementation

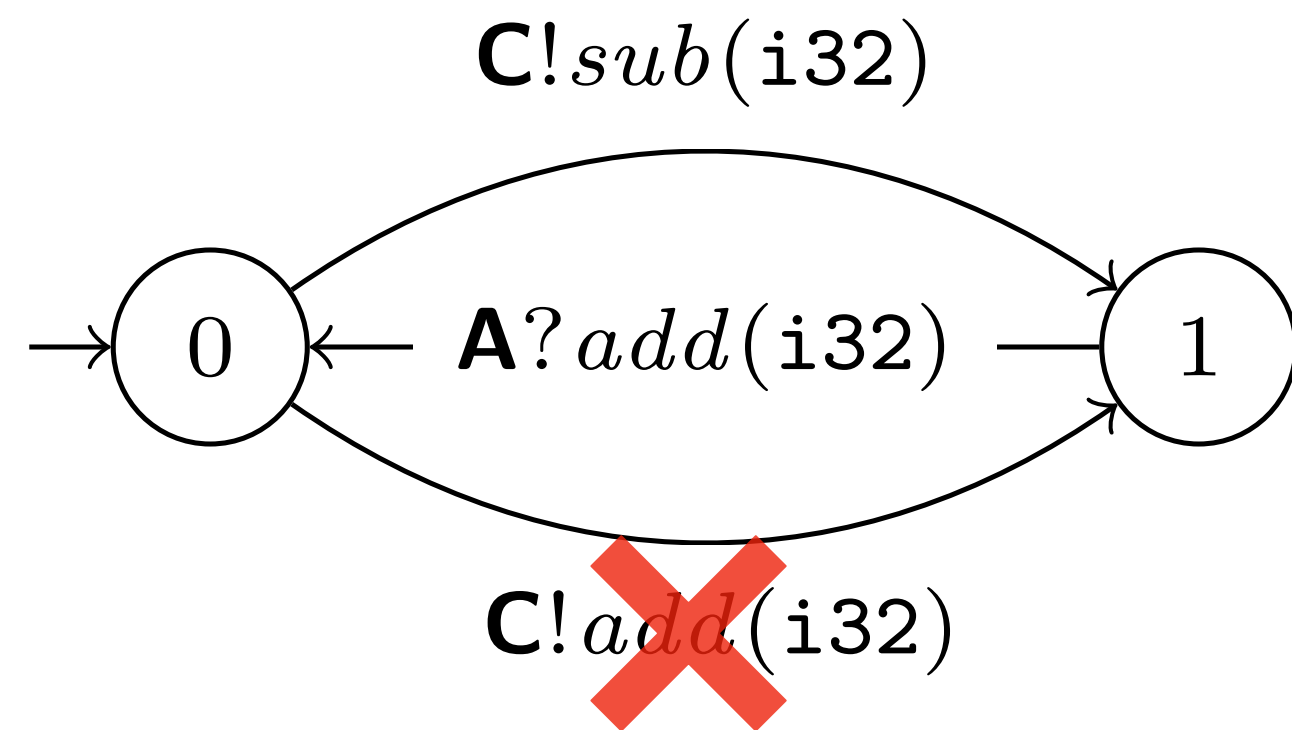


```
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

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

# Ring Protocol

## Implementation



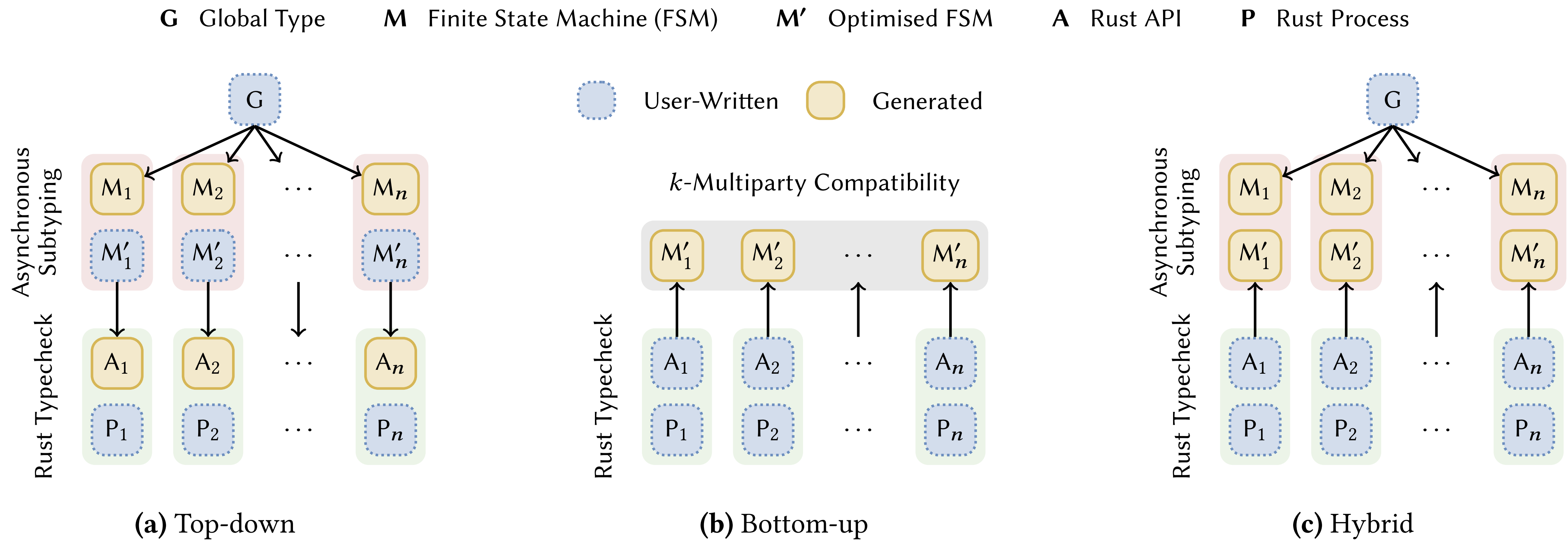
```
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

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

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

# Rumpsteak Framework

## Three Approaches



# Theories for Communication Optimisation

## Asynchronous Reordering Revisited

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

# Theories for Communication Optimisation

## Asynchronous Reordering Revisited

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

1. Asynchronous subtyping [Ghilezan, Pantvic, Prokic, Scalas and NY **POPL'2021**]

# Theories for Communication Optimisation

## Asynchronous Reordering Revisited

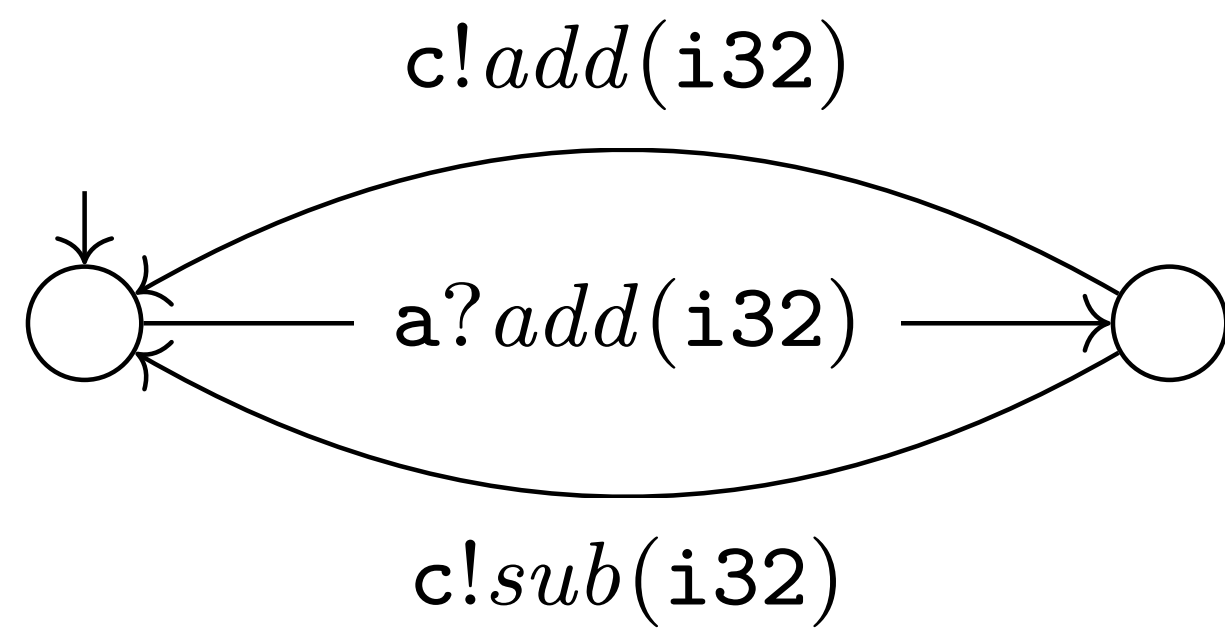
How do we check that asynchronous reorderings are *safe*?

1. Asynchronous subtyping [Ghilezan, Pantvic, Prokic, Scalas and NY **POPL'2021**]
2.  $k$ -multiparty compatibility [Lange and NY, **CAV'2019**]

# Safety

## Asynchronous Subtyping

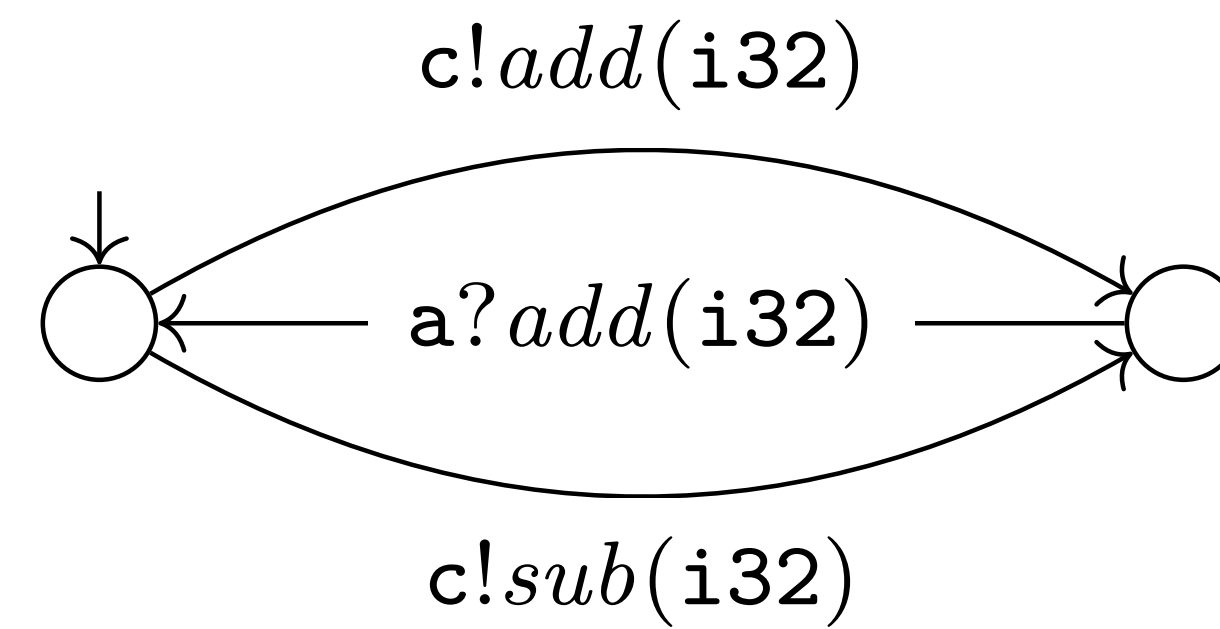
PROJECTED B



Safe?



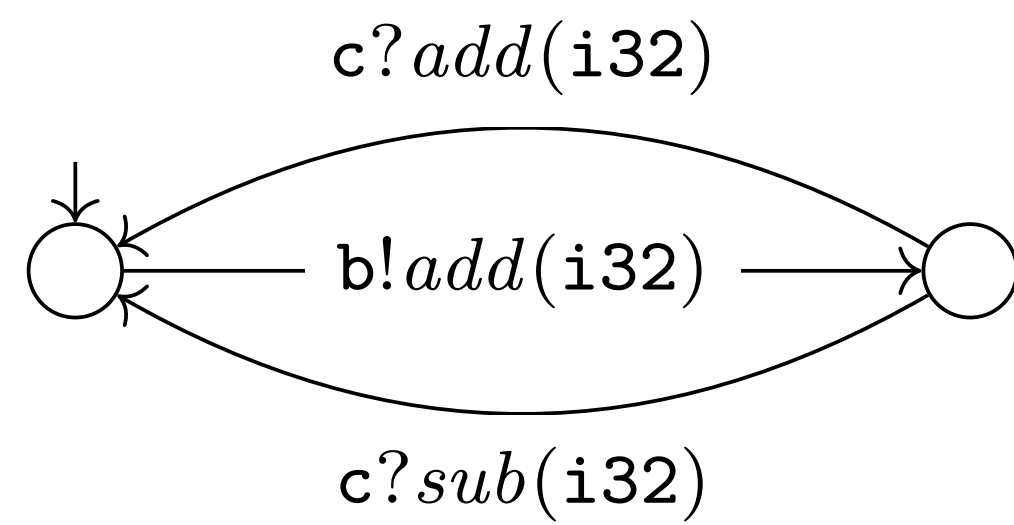
OPTIMISED B



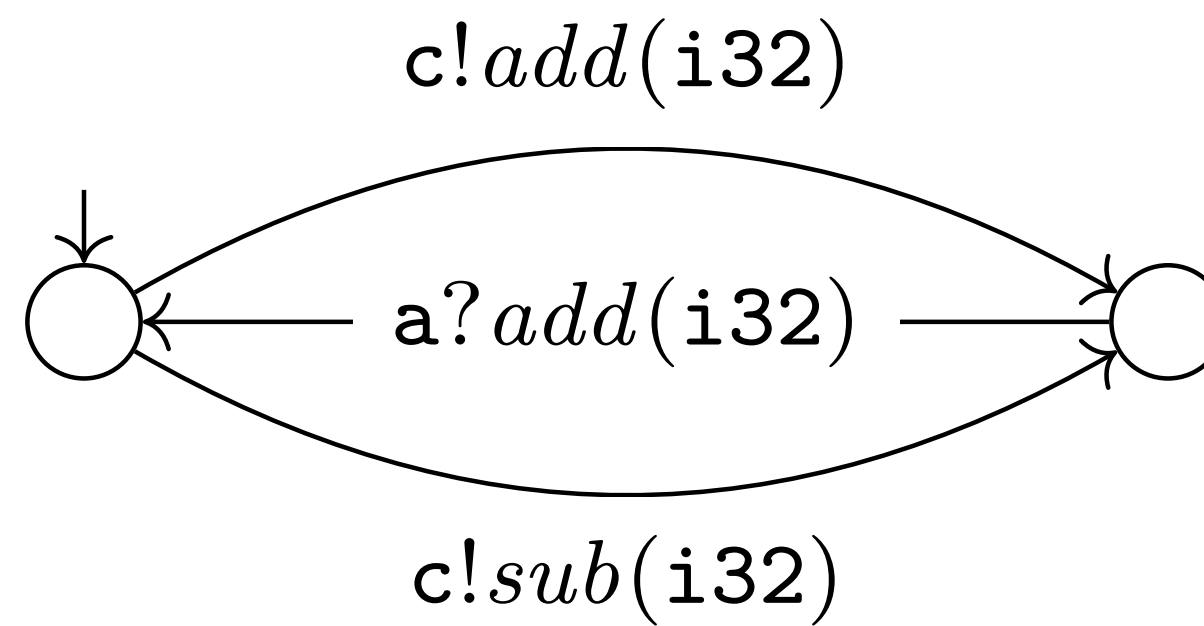
# Safety

## *k*-Multiparty Compatibility

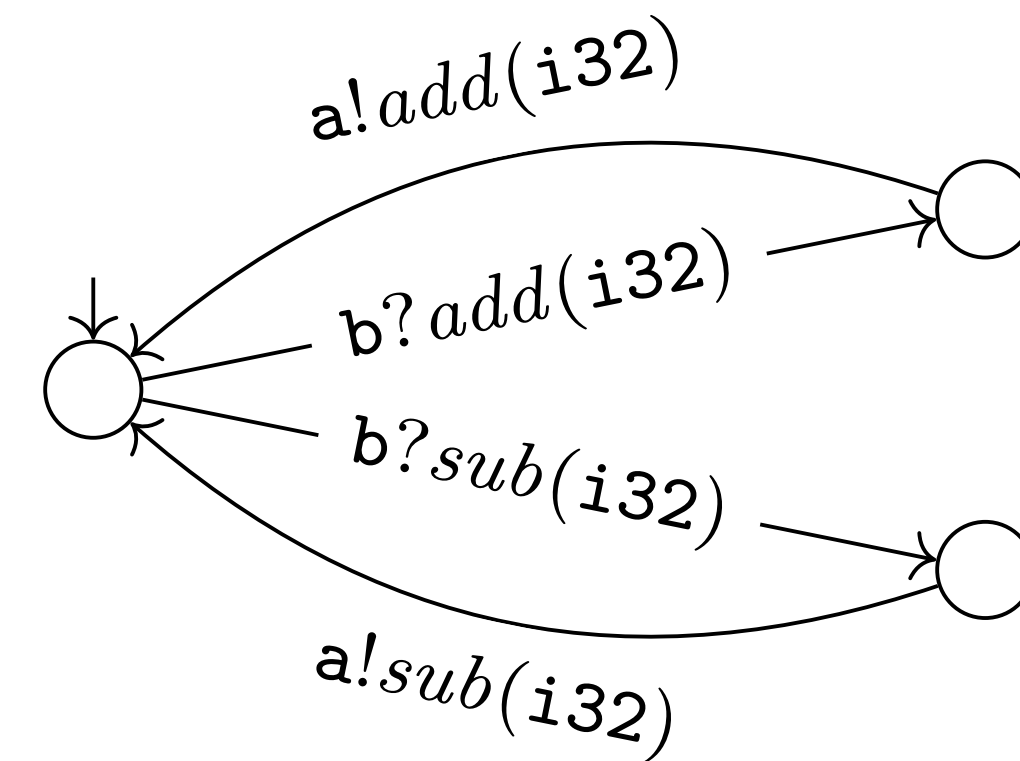
OPTIMISED A



OPTIMISED B

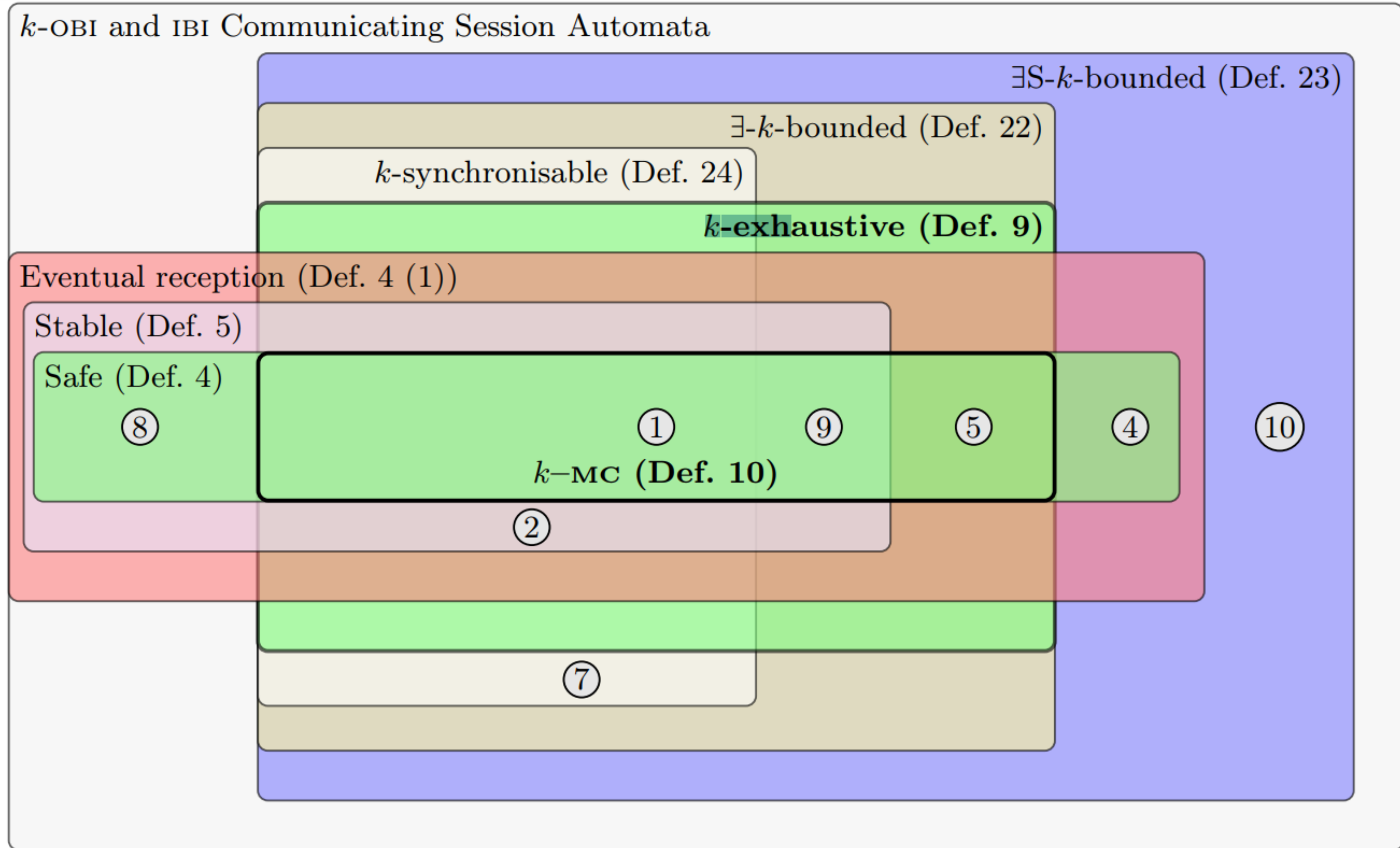


OPTIMISED C



Safe?

# k-Multiparty Compatibility [CAV'19]



# Asynchronous Subtyping

## Existing work

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

## Theorem [POPL 2021]

Internal and external choices can be decomposed into single input and single output trees

# Asynchronous Subtyping

## Existing work

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

## Theorem [POPL 2021]

Internal and external choices can be decomposed into single input and single output trees

# Asynchronous Subtyping

## Existing work

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

## Theorem [POPL 2021]

Internal and external choices can be decomposed into single input and single output trees

# Asynchronous Subtyping

## Existing work

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

## Theorem [POPL 2021]

Internal and external choices can be decomposed into single input and single output trees

# Asynchronous Subtyping

## Existing work

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

## Theorem [POPL 2021]

Internal and external choices can be decomposed into single input and single output trees

# Asynchronous Subtyping

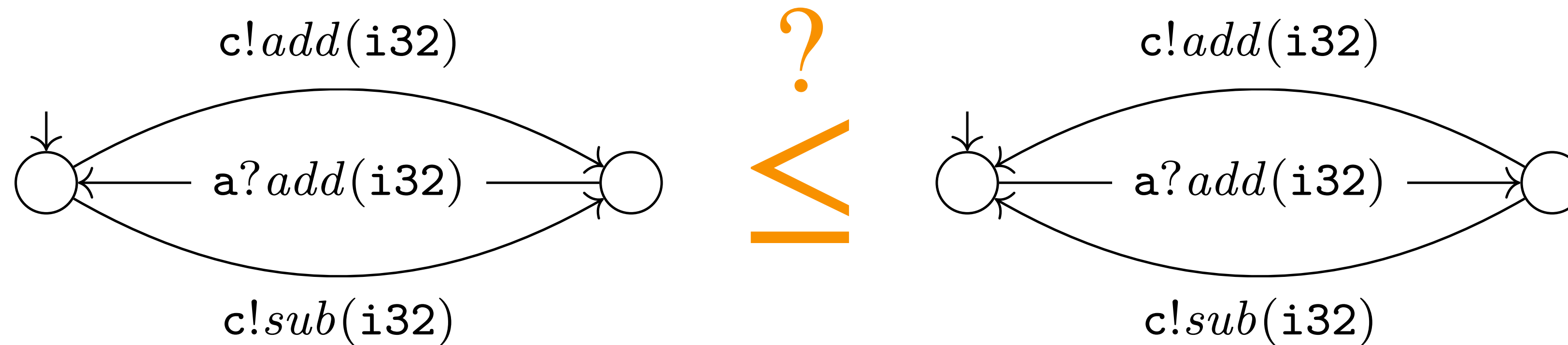
## Existing work

- Relation given by [Ghilezan et al., POPL 2021]
  - Sound ✓
  - Complete ✓
  - Decidable [Lange and NY, FoSSaCs 2017] ✗
- Our aim is a sound and decidable algorithm
- **Theorem [POPL 2021]**  
Internal and external choices can be decomposed into single input and single output trees

# Asynchronous Subtyping

## The Problem

- Choice and recursion make subtyping hard



# Asynchronous Subtyping

## SISO Refinement [POPL'21]

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

$$\begin{array}{c}
 \overline{\overline{\text{end} \lesssim \text{end}}} \\
 \\
 \frac{S' \leqslant S \quad W \lesssim W'}{\overline{\overline{\mathbf{p}?\ell(S).W \lesssim \mathbf{p}?\ell(S').W'}}} \qquad \frac{S' \leqslant S \quad W \lesssim \mathcal{A}^{(\mathbf{p})}.W' \quad \text{act}(W) = \text{act}(\mathcal{A}^{(\mathbf{p})}.W')}{\overline{\overline{\mathbf{p}?\ell(S).W \lesssim \mathcal{A}^{(\mathbf{p})}.\mathbf{p}?\ell(S').W'}}} \\
 \\
 \frac{S \leqslant S' \quad W \lesssim W'}{\overline{\overline{\mathbf{p}!\ell(S).W \lesssim \mathbf{p}!\ell(S').W'}}} \qquad \frac{S \leqslant S' \quad W \lesssim \mathcal{B}^{(\mathbf{p})}.W' \quad \text{act}(W) = \text{act}(\mathcal{B}^{(\mathbf{p})}.W')}{\overline{\overline{\mathbf{p}!\ell(S).W \lesssim \mathcal{B}^{(\mathbf{p})}.\mathbf{p}!\ell(S').W'}}}
 \end{array}$$

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

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

# Algorithm for Asynchronous Subtyping

## Practical, Sound and Terminating

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

# Asynchronous Subtyping

## Session Type Prefix

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

# Asynchronous Subtyping

## Reduction Rules

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

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

# Asynchronous Subtyping

## Reduction Rules

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

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

# Asynchronous Subtyping

## Reduction Rules

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

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

# Asynchronous Subtyping

## Reduction Rules

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

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

# Asynchronous Subtyping

## Reduction Rules

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

# Asynchronous Subtyping

## Reduction Rules

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

# Asynchronous Subtyping

## Reduction Rules

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

# Asynchronous Subtyping

## Reduction Rules

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



# Asynchronous Subtyping

## Reduction Rules

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

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

# Asynchronous Subtyping

## Reduction Rules

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



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

# Asynchronous Subtyping

## Reduction Rules

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



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

# Asynchronous Subtyping

## Reduction Rules

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



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

# Asynchronous Subtyping

## Reduction Rules

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

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

# Asynchronous Subtyping

## Reduction Rules

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

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

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

# Asynchronous Subtyping

## Reduction Rules

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

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

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

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

# Asynchronous Subtyping

## Reduction Rules

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

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

# Theorems

## Termination, Soundness & Complexity

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

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

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

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

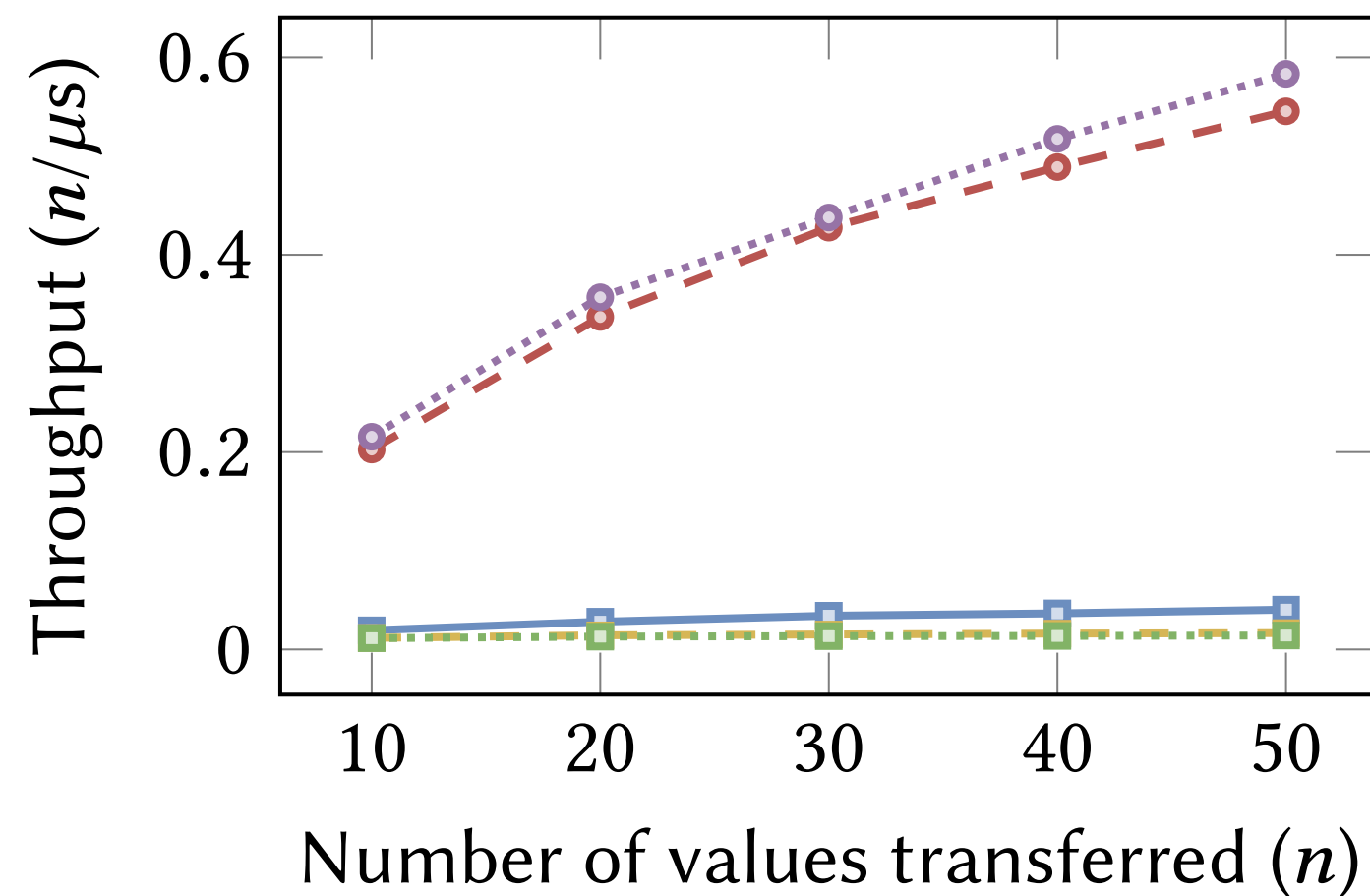
**Theorem 7 (Complexity).** *Consider  $T$  and  $T'$  as (possibly infinite) trees  $\mathcal{T}(T)$  and  $\mathcal{T}(T')$  with asymptotic branching factors  $b$  and  $b'$  respectively. Our algorithm has time complexity  $O(n \min(b, b')^n)$  and space complexity  $O(n \min(b, b'))$  in the worst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

# Evaluation

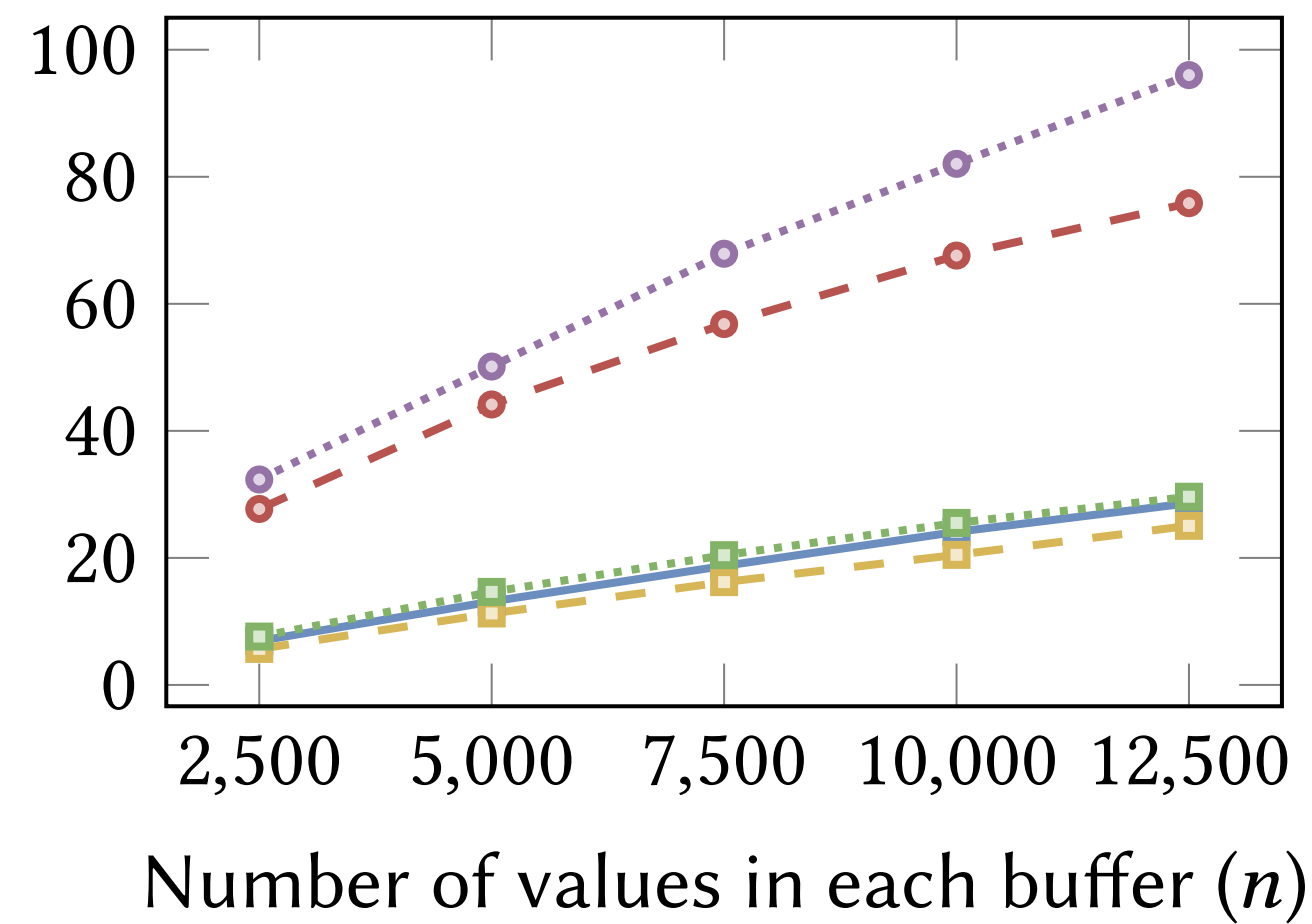
## Rust Framework Benchmarks

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

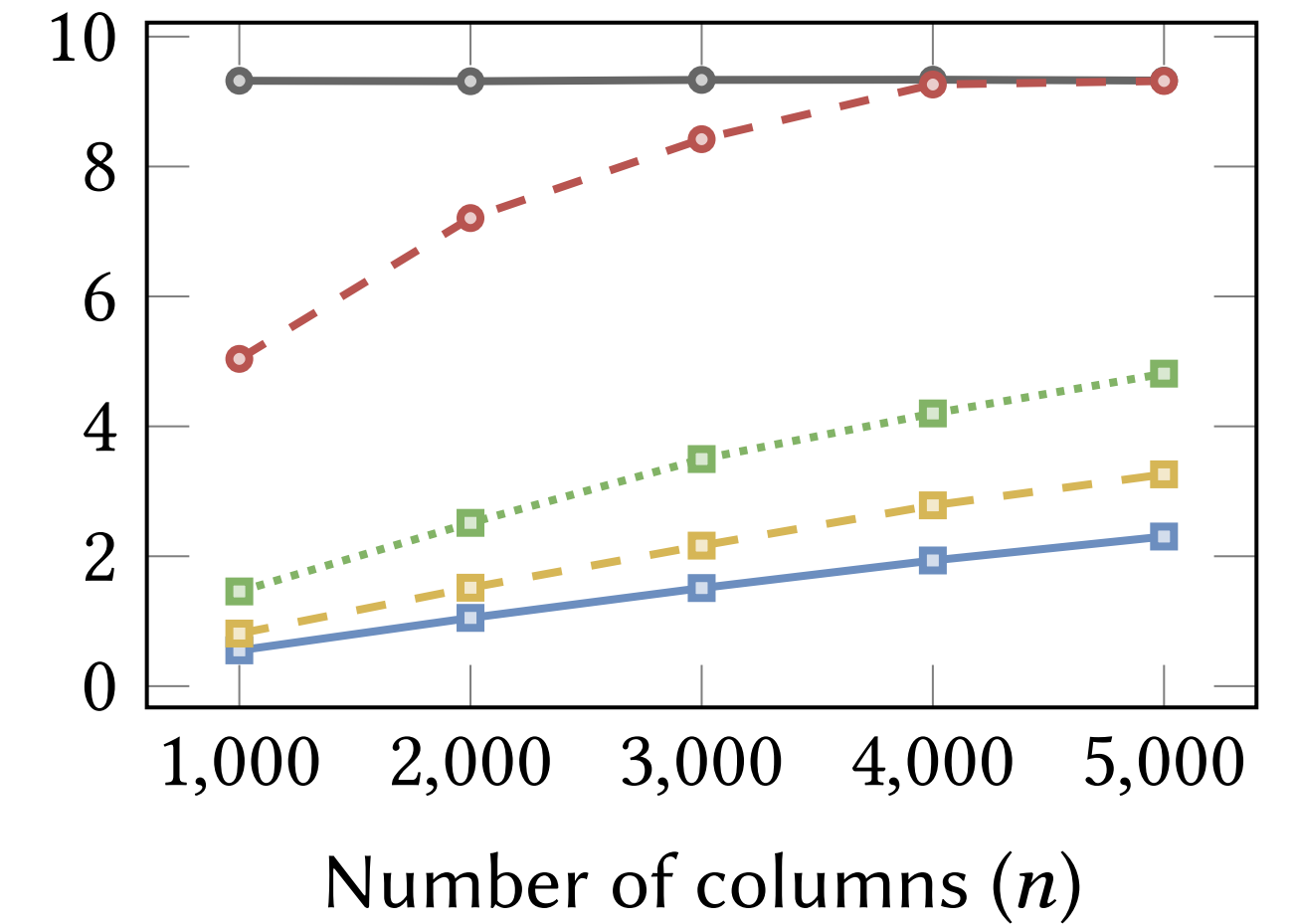
*Stream*



*Double Buffering*

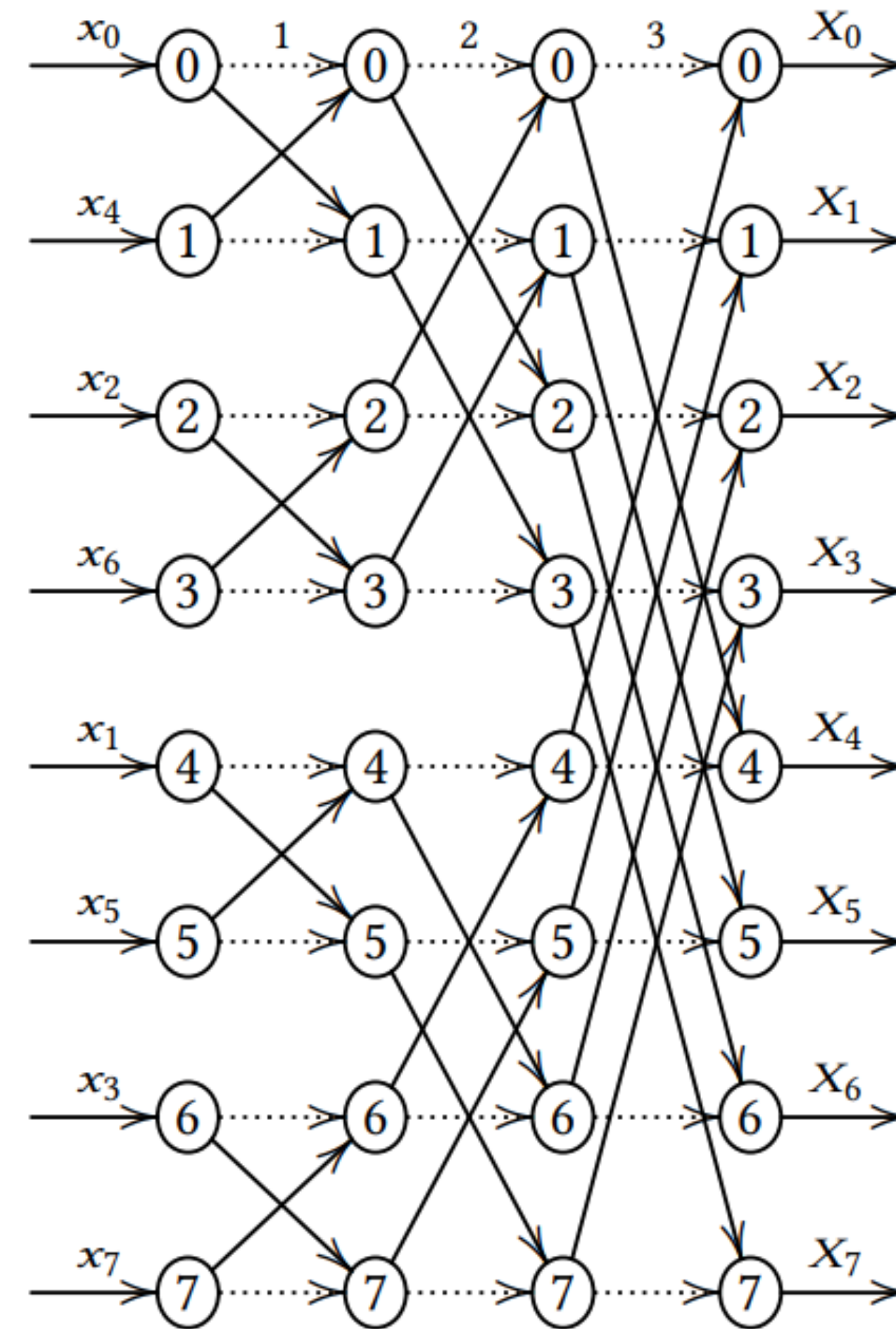
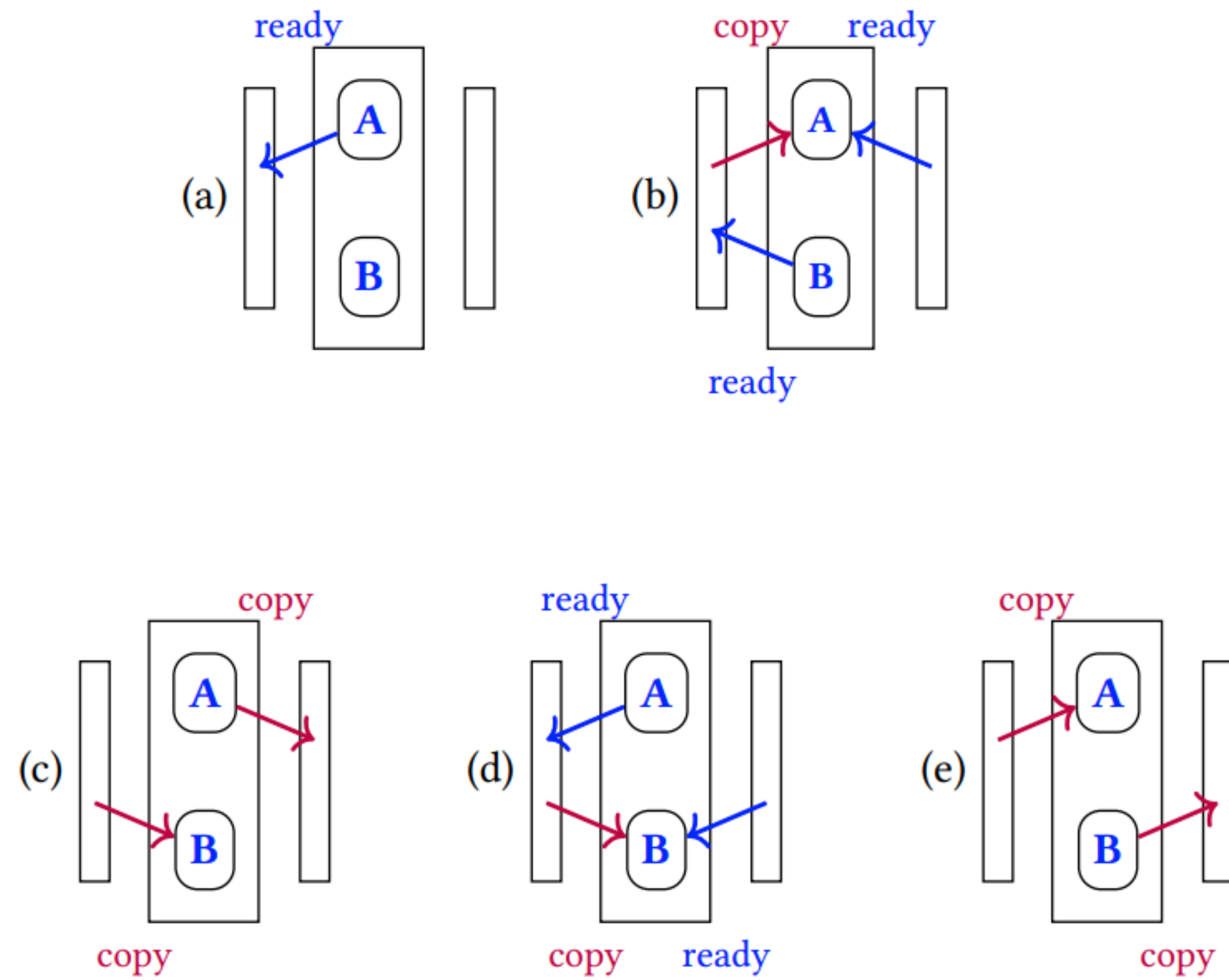


*FFT*



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

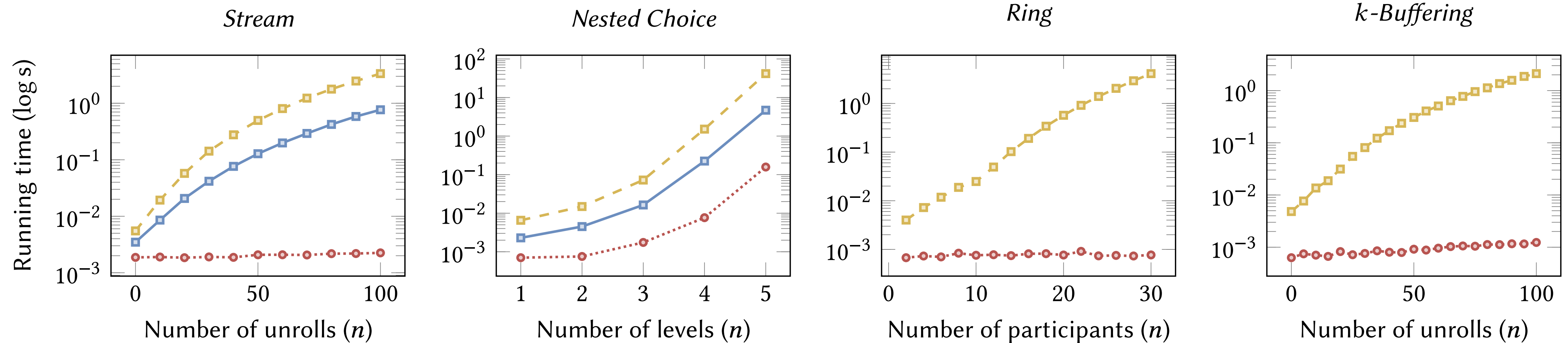
# Double DB & Butterfly Topologies for FFT



# Evaluation

## Asynchronous Reordering Benchmarks

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



# Nested Session Asynchronous Subtyping

## Precise Subtyping by Chen, Dezani et al

ON THE PRECISENESS OF SUBTYPING IN SESSION TYPES

23

$$\frac{S_m^r \leq S_m \quad S_m^s \leq S_m \quad S_p^r \leq S_p \quad S_p^s \leq S_p \quad T_m \leq ?r(S_r).T_r \ \& \ ?s(S_s).T_s \quad T_p \leq ?r(S_r).T'_r \ \& \ ?s(S_s).T'_s}{!m\langle S_m \rangle.T_m \oplus !p\langle S_p \rangle.T_p \leq ?r(S_r).( !m\langle S_m^r \rangle.T_r \oplus !p\langle S_p^r \rangle.T'_r \oplus !q\langle S_q \rangle.T_q) \ \& \ ?s(S_s).( !m\langle S_m^s \rangle.T_s \oplus !p\langle S_p^s \rangle.T'_s)}$$

Figure 3: Application of [SUB-PERM-ASYNC], where  $T_m = ?r(S_r).T_r \ \& \ ?s(S_s).T_s \ \& \ ?u(S_u).T_u$  and  $T_p = ?r(S'_r).T'_r \ \& \ ?s(S_s).T'_s$  and we assume  $S'_r \leq S_r$ .

$$\begin{aligned} T_0 &= T'_0 = \text{end} \\ T_{n+1} &= !m.(?r.T_n \ \& \ ?s.T_n \ \& \ ?u.T_n) \oplus !p.(?r.T_n \ \& \ ?s.T_n) \\ T'_{n+1} &= ?r.( !m.T'_n \oplus !p.T'_n \oplus !q.T'_n) \ \& \ ?s.( !m.T'_n \oplus !p.T'_n) \end{aligned}$$

# Evaluation

## Expressiveness



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

$n$  Number of participants    AMR Asynchronous message reordering

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

# References

## Multiparty Session Types and Rust

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

# History of Session Types with mCRL2

- [TACAS'17] Using mCRL2 to benchmark **session subtyping** checking
- [POPL'17] Analysing properties (e.g., deadlock-freedom) of **Go** programs
- [ICSE'18] Analysing properties of **message-passing Go** programs
- [POPL'19] Analysing properties of **synchronous multiparty session pi-calculus**
- [PLDI'19] Analysing properties of **distributed Scala** programs
- [ECOOP'20] Analysing properties of **shared memory Go** programs
- [CONCUR'22] Extension of [POPL'19] with **Stop-Failure**

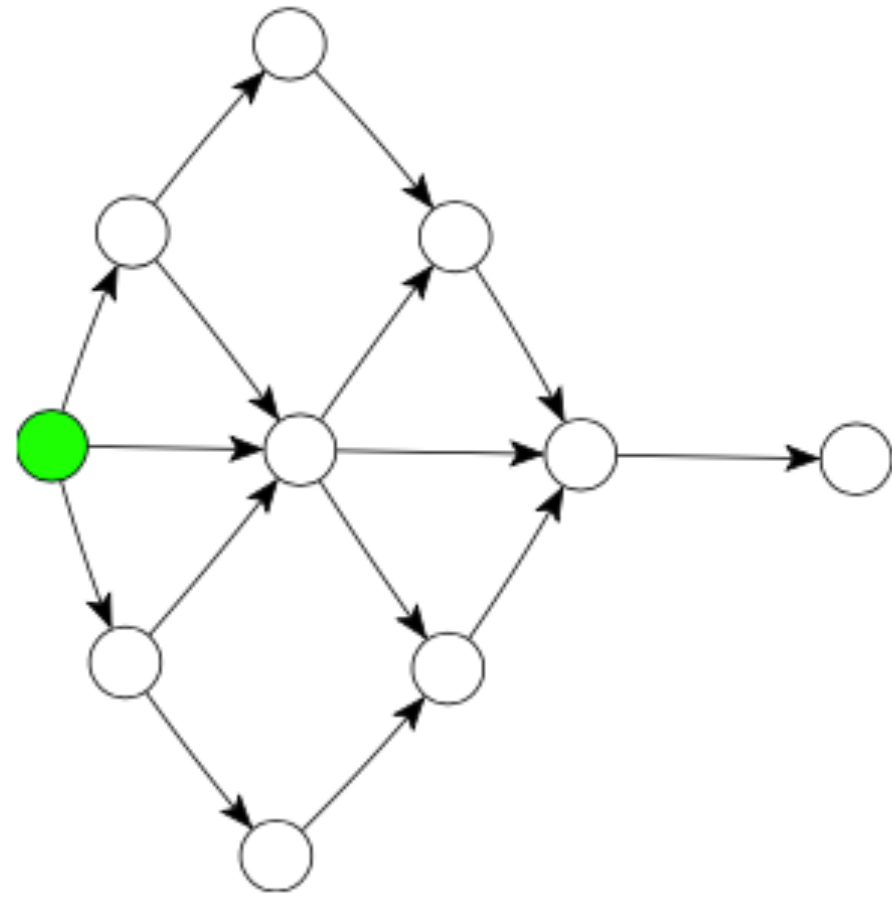
# History of Session Types with mCRL2

- [TACAS'17] Using mCRL2 to benchmark **session subtyping** checking
- [POPL'17] Analysing properties (e.g., deadlock-freedom) of **Go** programs
- [ICSE'18] Analysing properties of **message-passing Go** programs
- [POPL'19] Analysing properties of **synchronous multiparty session pi-calculus**
- [PLDI'19] Analysing properties of **distributed Scala** programs
- [ECOOP'20] Analysing properties of **shared memory Go** programs
- [CONCUR'22] Extension of [POPL'19] with **Stop-Failure**

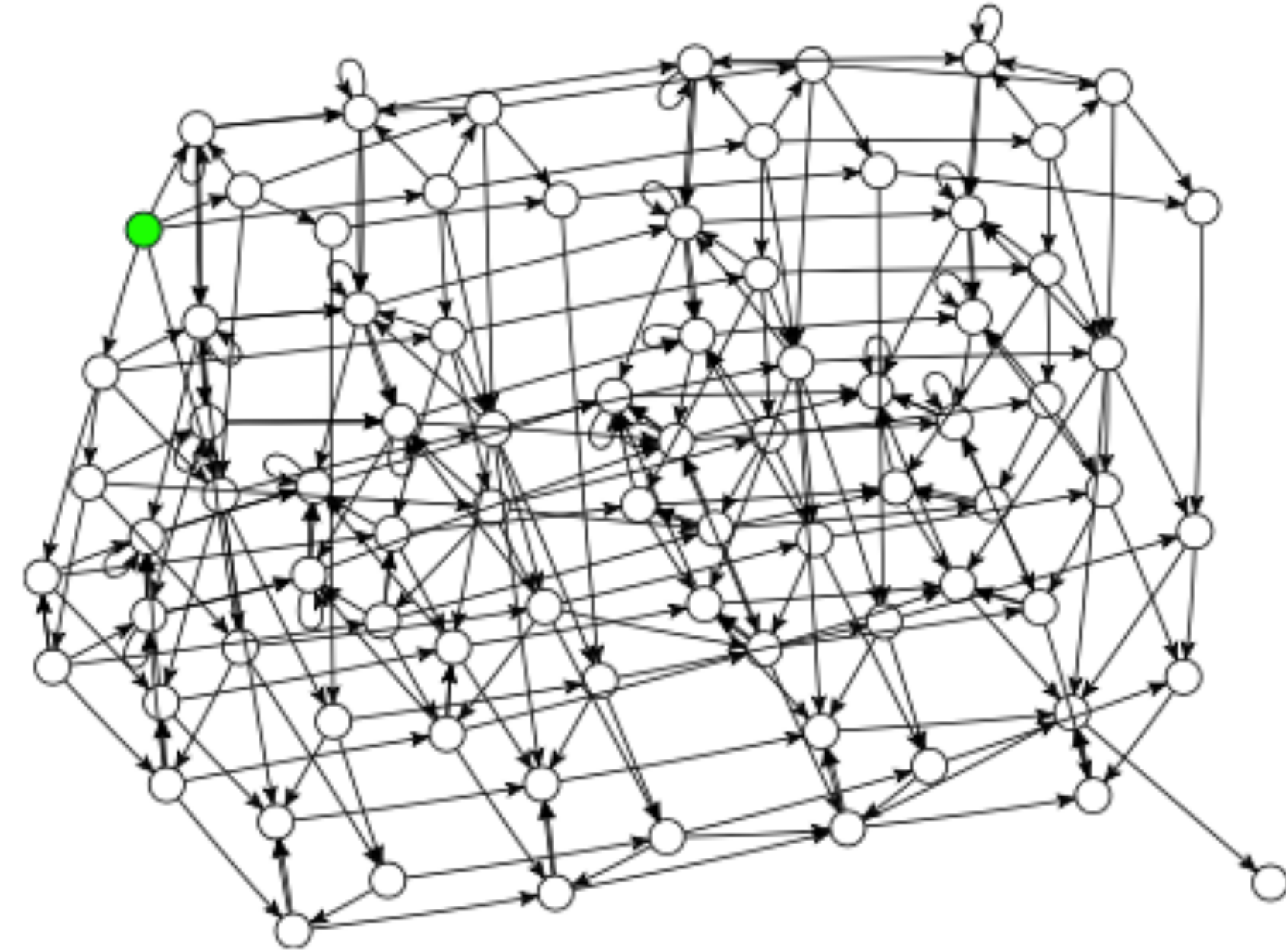
**No gap between type and process properties**

# Why mCRL2?

- Can analyse properties than the top-down approach
- Can scale



No Failure



with Failure

# Current Projects

- **TaRDIS: Horizon EU** (Nova, DTU, Novi Sad, Oxford+)
- **POST:** Protocols, Observabilities and Session Types ([EPSRC Established Career Fellowship](#))
- Parallel Programming (Hardware)
  - **Morello-HAT:** Morello High-Level API and Tooling ([ISCF Digital Security by Design](#)) (GL, Essex)
  - **AppControl:** Enforcing Application Behaviour through Type-Based Constraints ([ISCF Digital Security by Design](#)) (GL, Essex)
  - **Border Patrol:** Improving Smart Device Security through Type-Aware Systems Design (GL, Heriot-Watt)
- Distributed Programming
  - **Stardust:** Session Types for Reliable Distributed Systems (GL, Kent)
  - **Turtles:** Protocol-Based Foundations for Distributed Multiagent Systems (Lancaster)
- Security and Safety (2 **VeTSS** projects on Rust, Go)



# Discussions

