

# Specification-Guided Hybrid Verification for Parallel Programming in Rust

**Nobuko Yoshida, Martin Vassor, and Adam D. Barwell**  
University of Oxford

Programming Languages and Systems Group Seminar, University of Kent  
30 January 2023



# Communications are Ubiquitous

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



microservices



# 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?

---

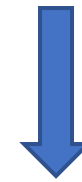
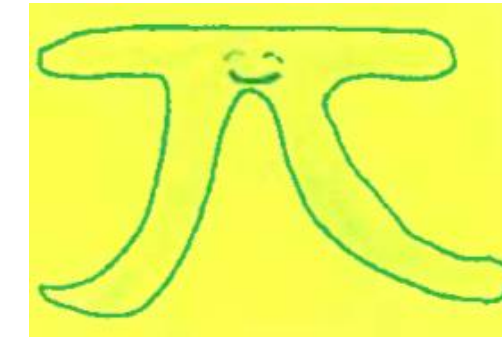
Significant academic and industry interests via fundamental breakthroughs

Milner,  
Honda, NY



Binary Session Types

ESOP'98

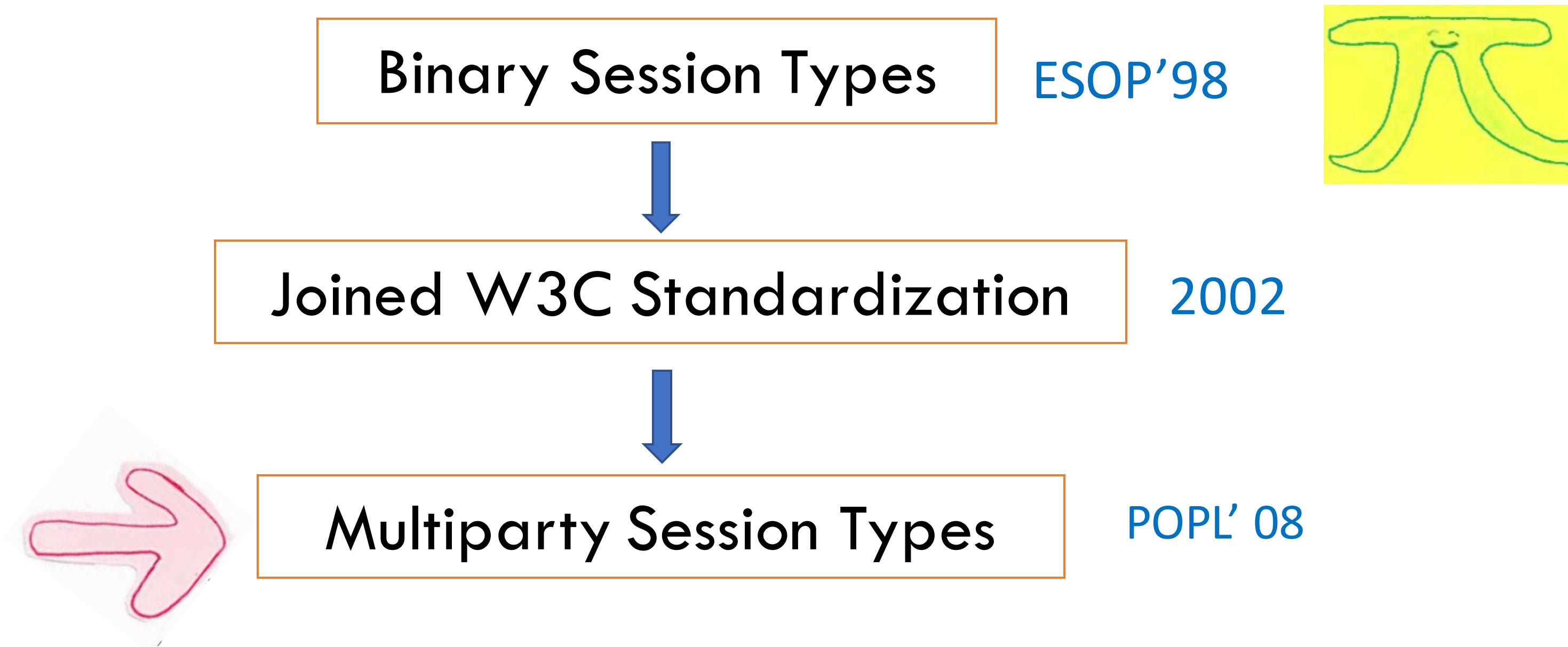


Joined W3C Standardization

2002

# Why Session Types, Why Now?

Significant academic and industry interests via fundamental breakthroughs

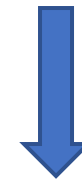
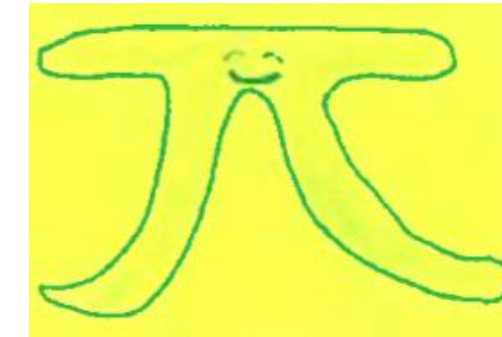


# Why Session Types, Why Now?

Significant academic and industry interests via fundamental breakthroughs

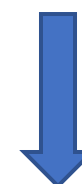
Binary Session Types

ESOP'98



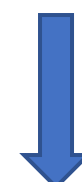
Joined W3C Standardization

2002



Multiparty Session Types

POPL' 08

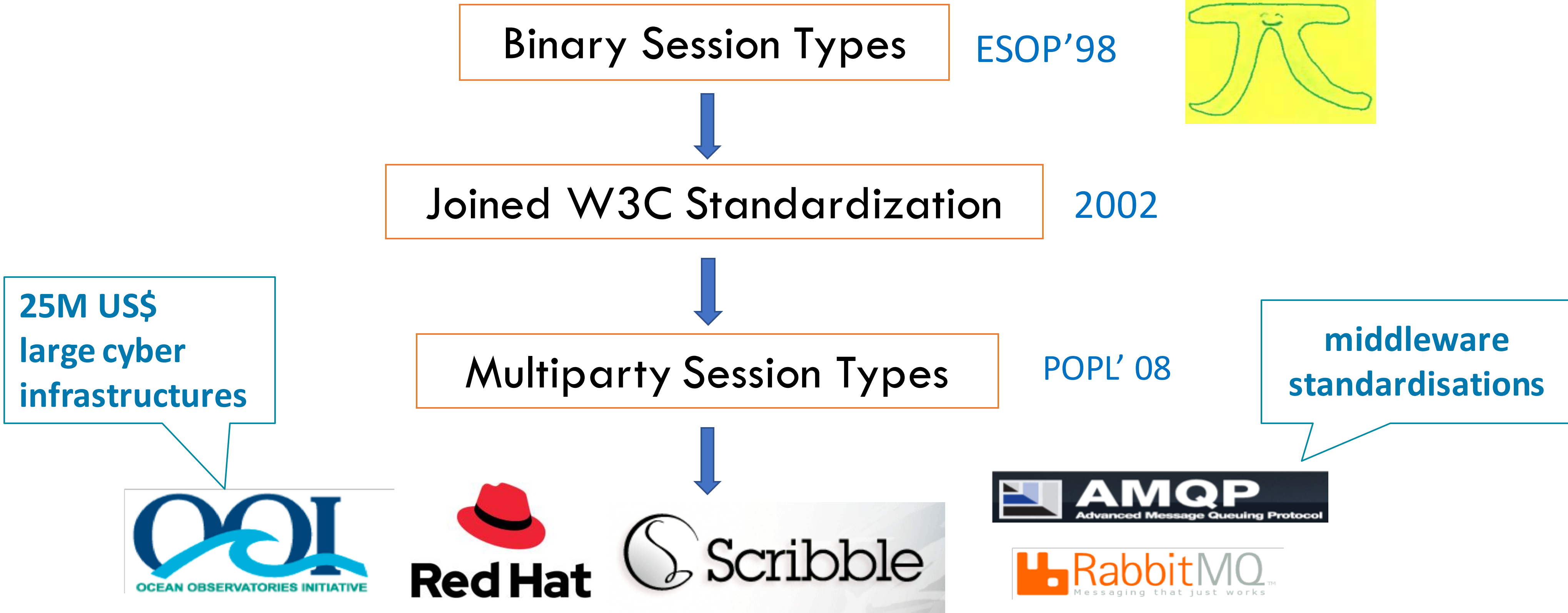


largest open source  
company in the world



# Why Session Types, Why Now?

Significant academic and industry interests via fundamental breakthroughs

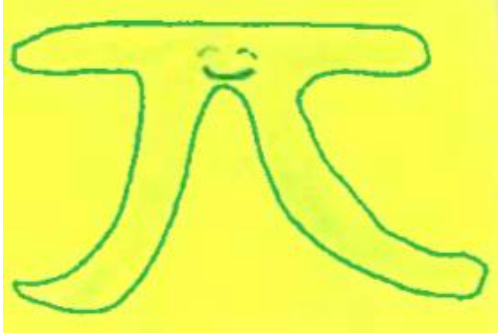


# Why Session Types, Why Now?

Significant academic and industry interests via fundamental breakthroughs

Binary Session Types

ESOP'98



Joined W3C Standardization

2002

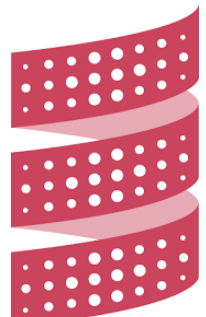


Multiparty Session Types

POPL'08



TypeScript



Scala

akka



ERLANG

MPI



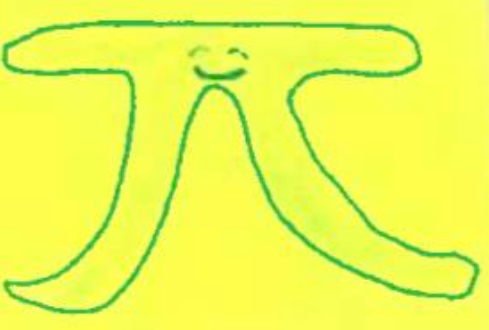
# Why Session Types, Why Now?

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



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

# Mobility Reading Group

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



## MobilityReadingGroup

$\pi$ -calculus, Session Types research at the University of Oxford

Home

People

Publications

Grants

Talks

Tutorials

Tools

Awards

Kohei Honda

CONCUR Interview

## NEWS

22 Mar 2022

MEng student, Zak Cutner, awarded Microsoft Prize and Distinguished Project award.

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!

9 Nov 2020

## SELECTED PUBLICATIONS

2023

Romain Demangeon, Nobuko Yoshida: [Causal Computational Complexity of Distributed Processes](#). IC 2023 : 104998.

2022

Zak Cutner, Nobuko Yoshida, Martin Vassor: [Deadlock-Free Asynchronous Message Reordering in Rust with Multiparty Session Types](#). PPOPP '22 : 261 - 246.

Lorenzo Gheri, Ivan Lanese, Neil Sayers, Emilio Tuosto, Nobuko Yoshida: [Design-by-Contract for Flexible Multiparty Session Protocols](#). ECOOP 2022 : 8:1 - 8:28.

Nicolas Lagailardie, Rumyana Neykova, Nobuko Yoshida: [Stay Safe under Panic: Affine Rust Programming with Multiparty Session Types](#). ECOOP 2022 : 4:1 - 4:29.

# 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

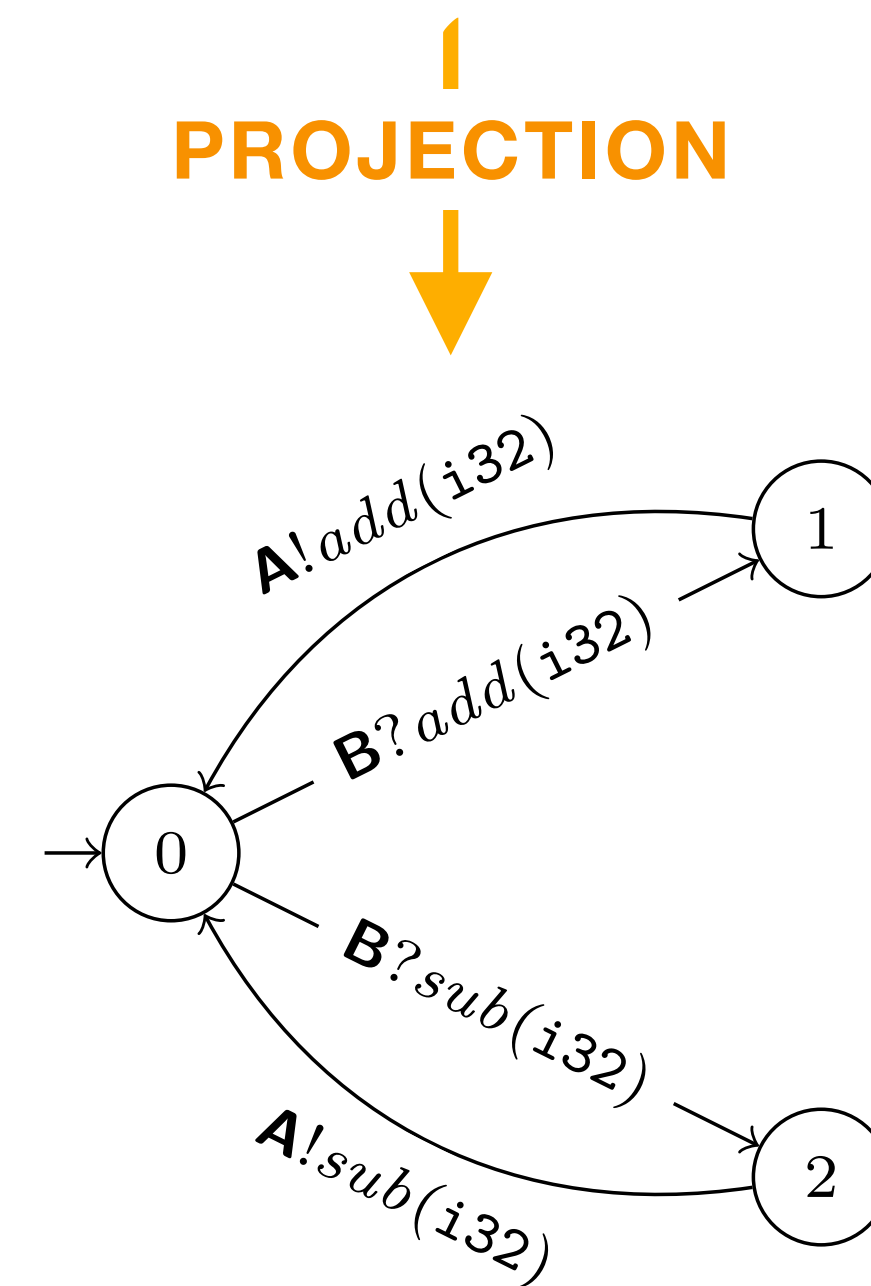
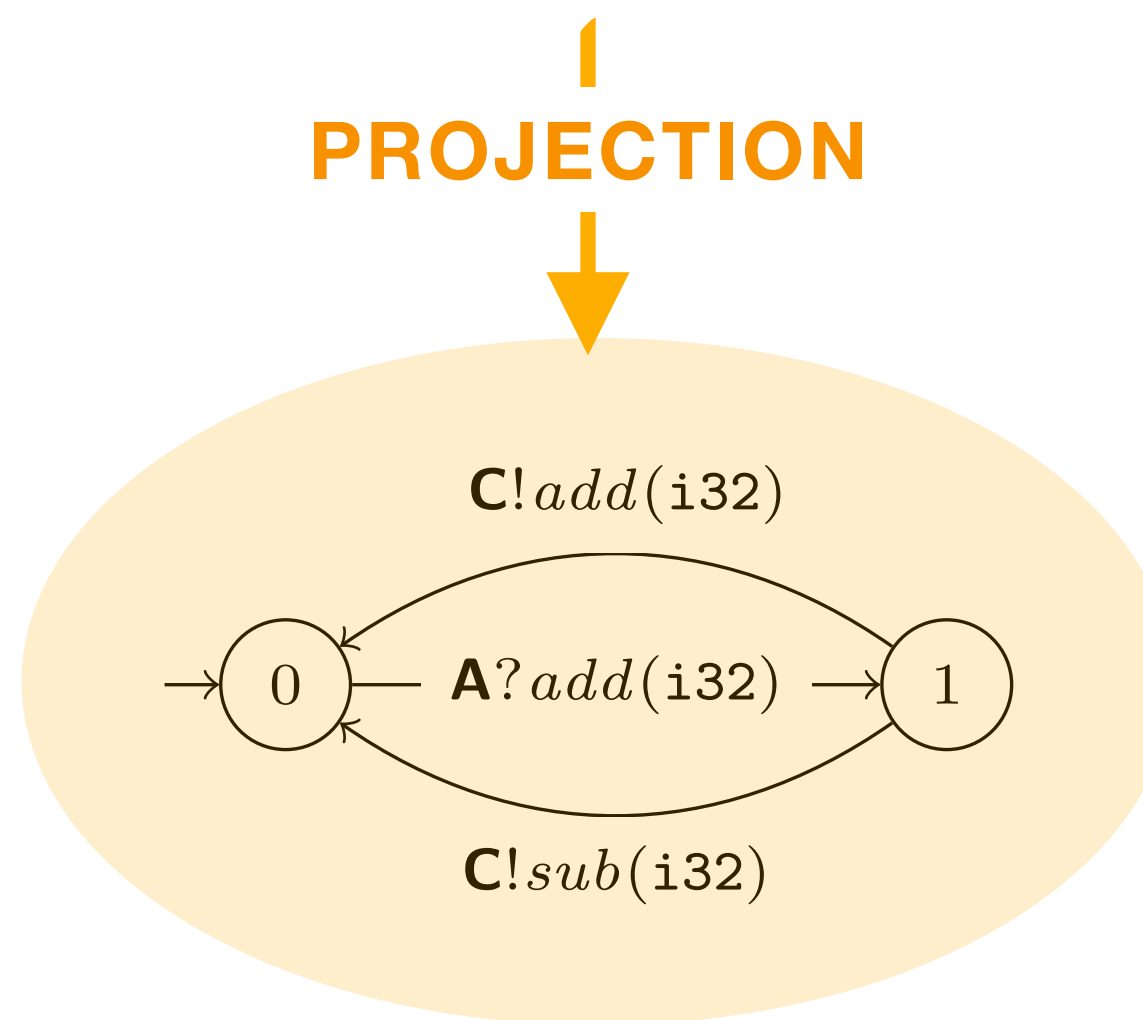
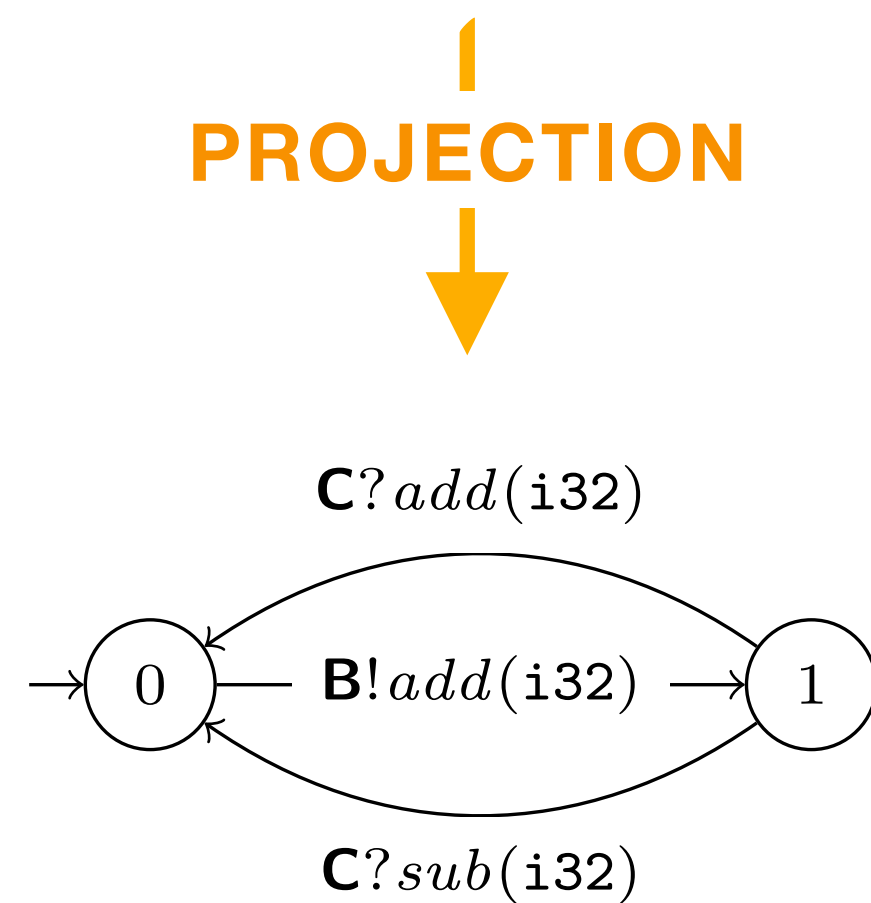
### Global Type

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

# Ring Protocol

## Example

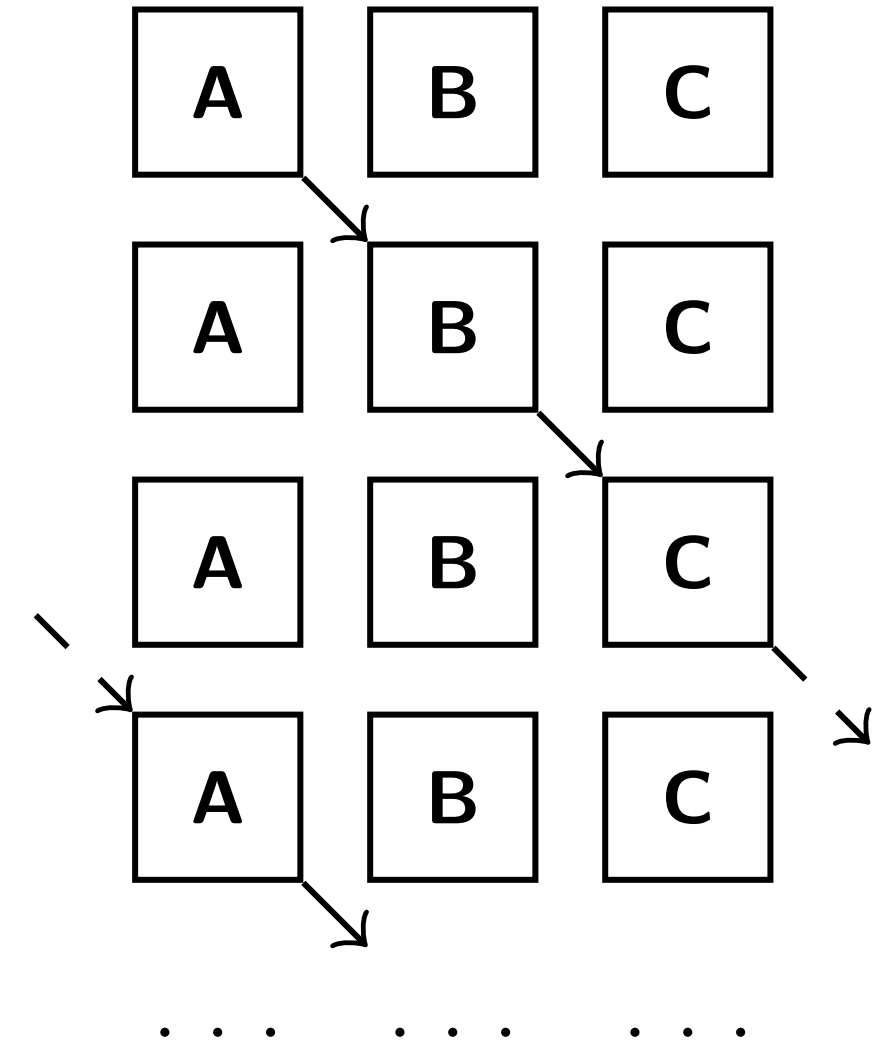
$$G = \mu t. \mathbf{A} \rightarrow \mathbf{B} : \left\{ \begin{array}{l} \mathit{add}(i32). \mathbf{B} \rightarrow \mathbf{C} : \left\{ \begin{array}{l} \mathit{add}(i32). \mathbf{C} \rightarrow \mathbf{A} : \{ \mathit{add}(i32). t \} \\ \mathit{sub}(i32). \mathbf{C} \rightarrow \mathbf{A} : \{ \mathit{sub}(i32).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



# Challenge

## Asynchronous Orderings

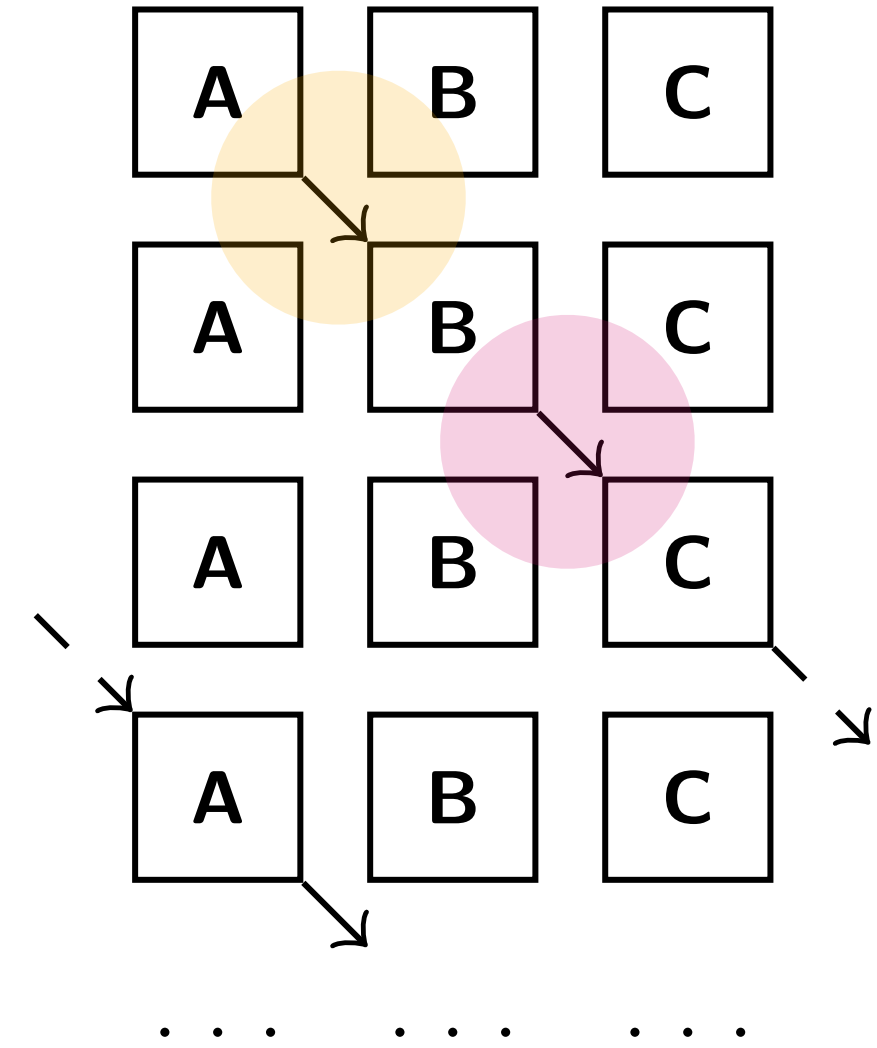
- Global types are inherently **synchronous**
  - Projection provides only one possible ordering



# Challenge

## Asynchronous Orderings

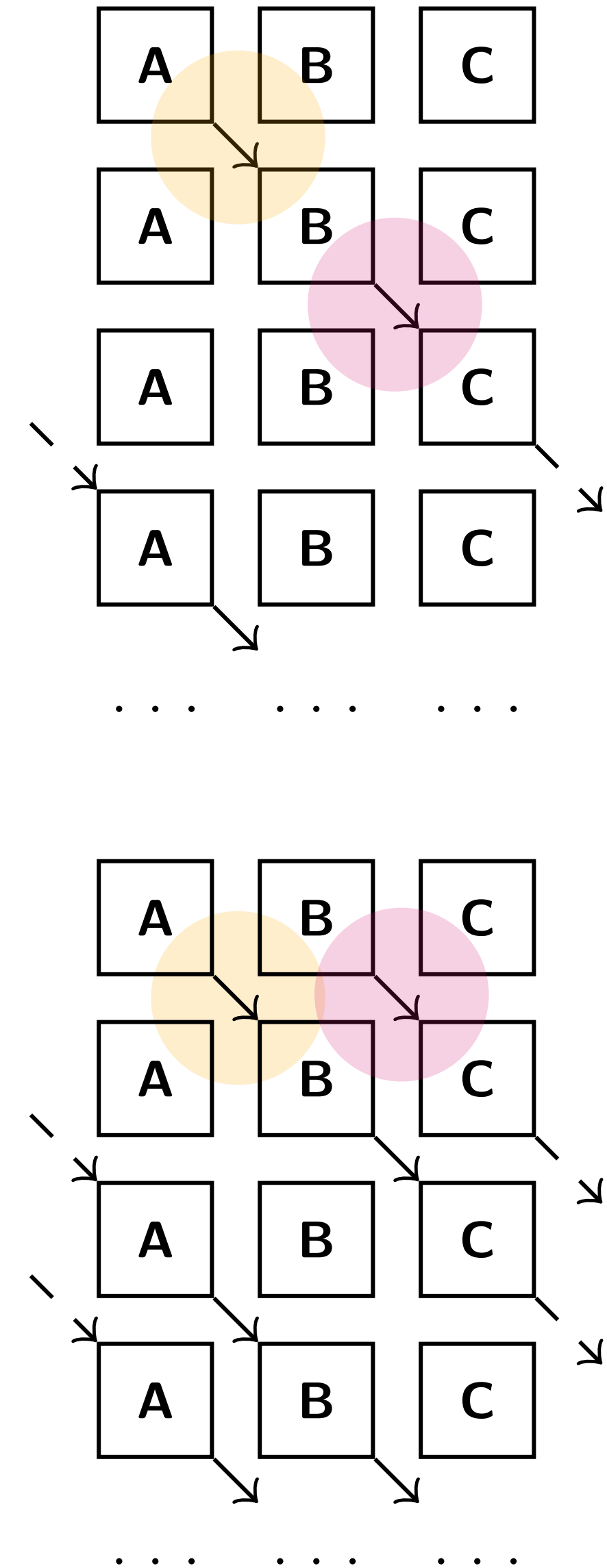
- Global types are inherently **synchronous**
  - Projection provides only one possible ordering



# Challenge

## Asynchronous Orderings

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



# Rumpsteak Framework

## Three Approaches

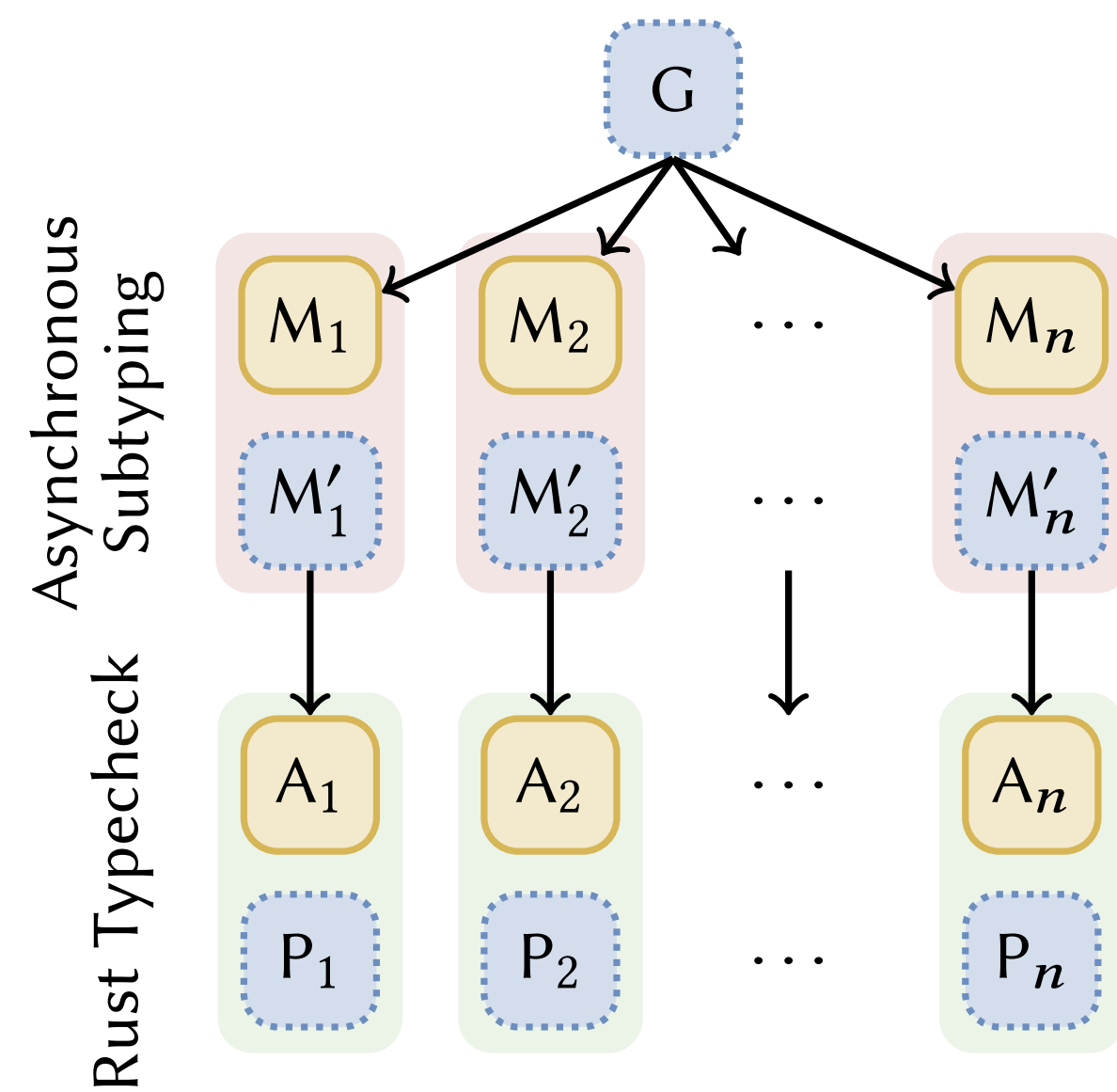
**G** Global Type

**M** Finite State Machine (FSM)

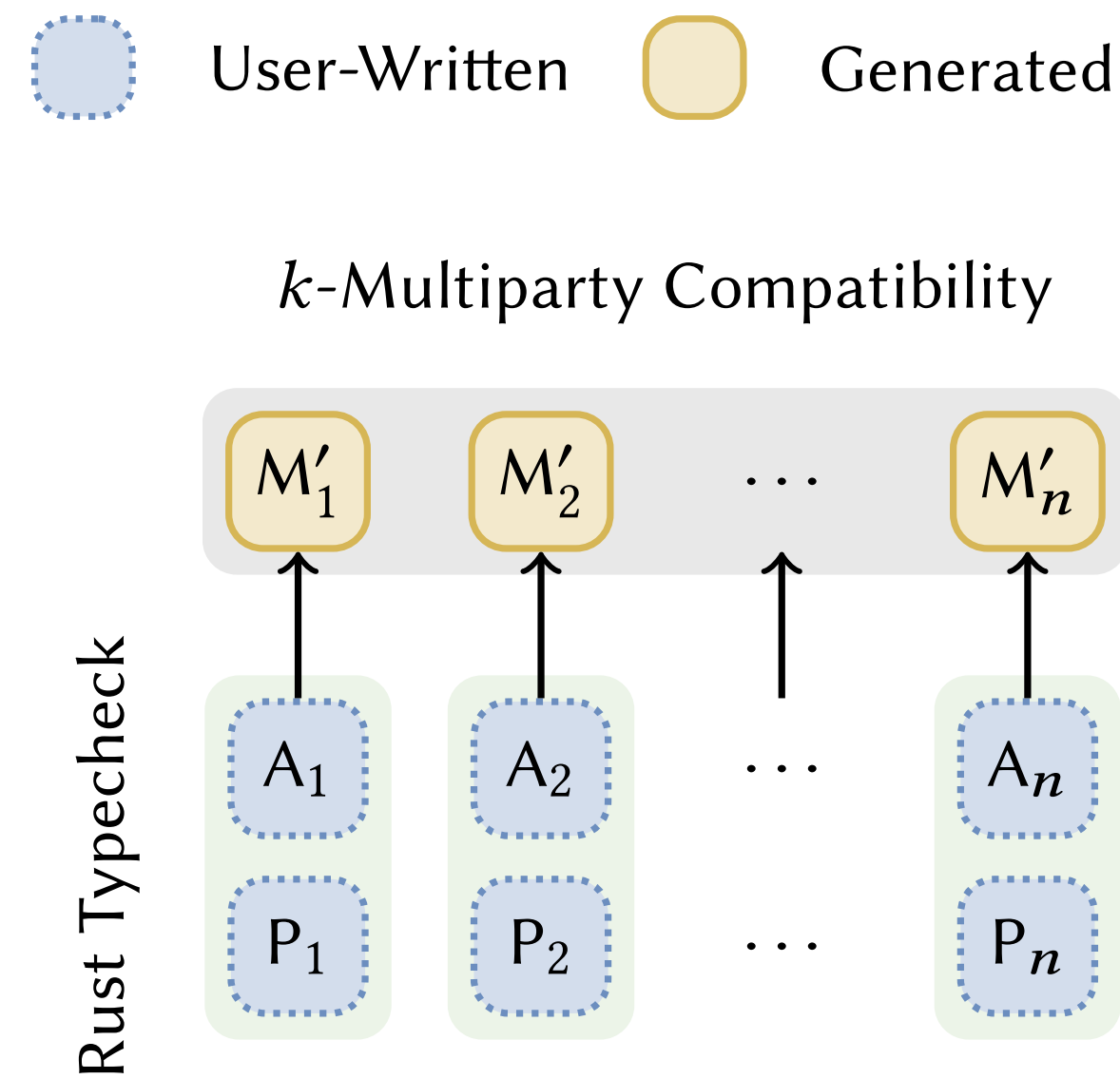
**M'** Optimised FSM

**A** Rust API

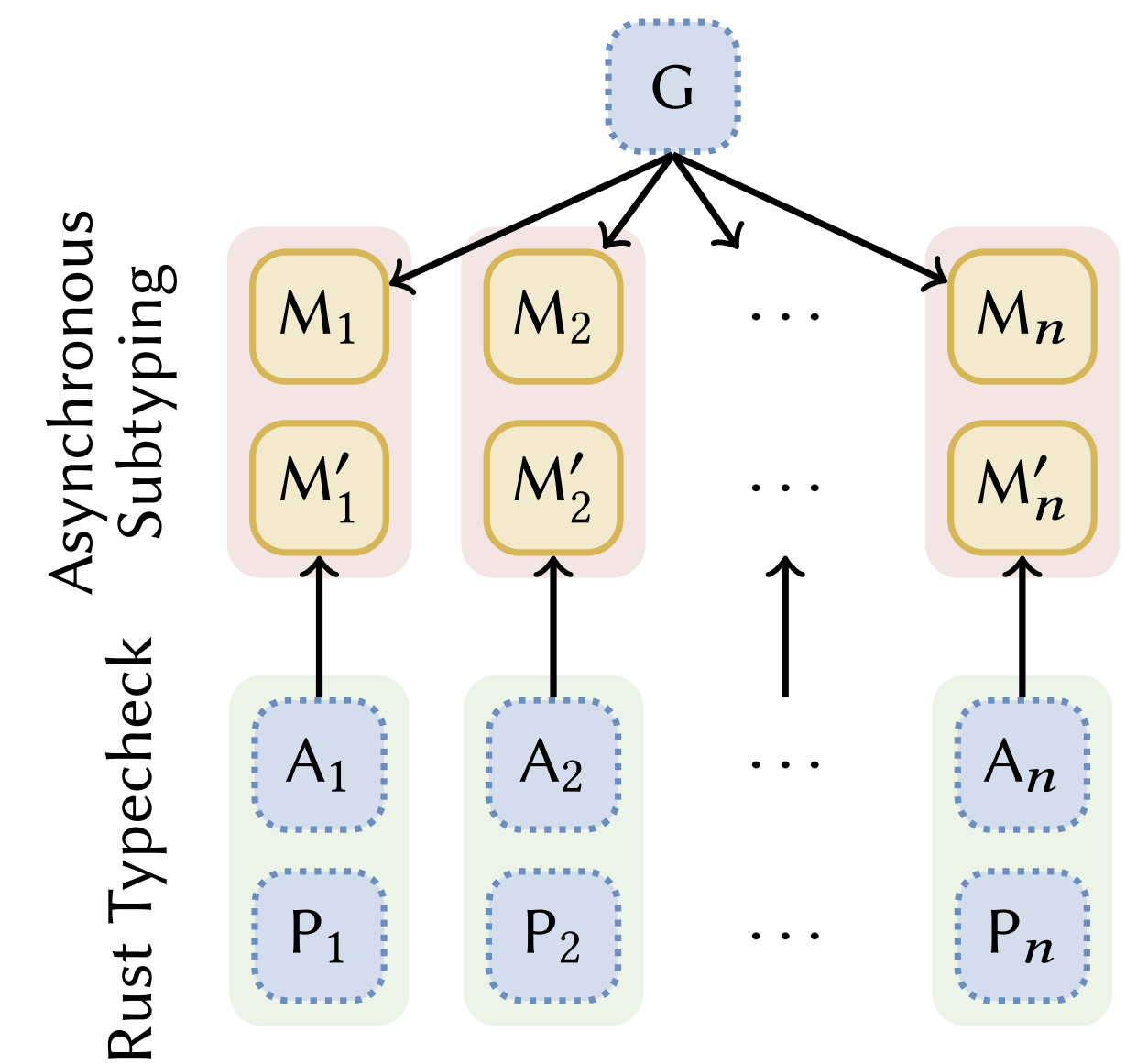
**P** Rust Process



(a) Top-down



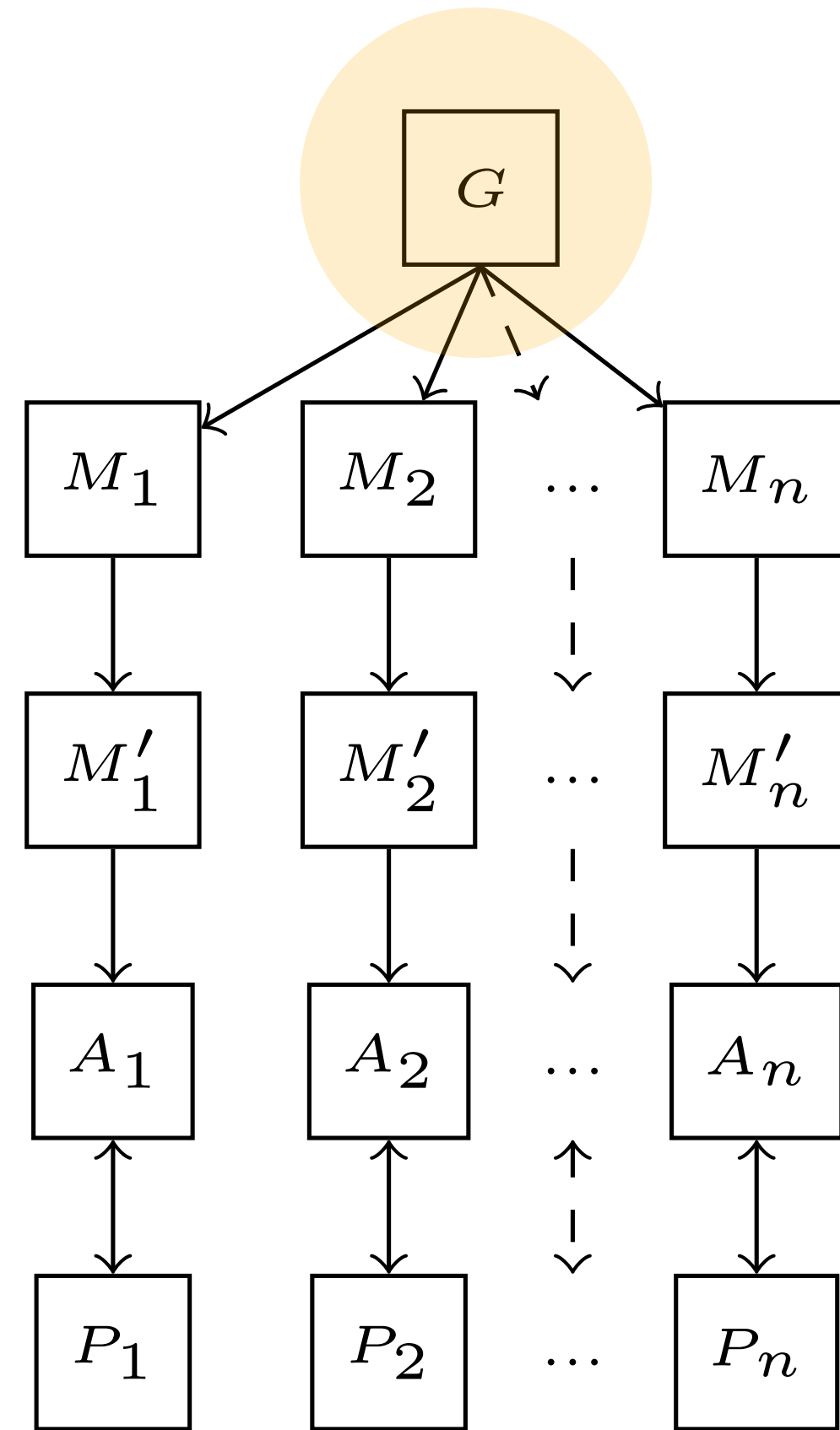
(b) Bottom-up



(c) Hybrid

# Workflow

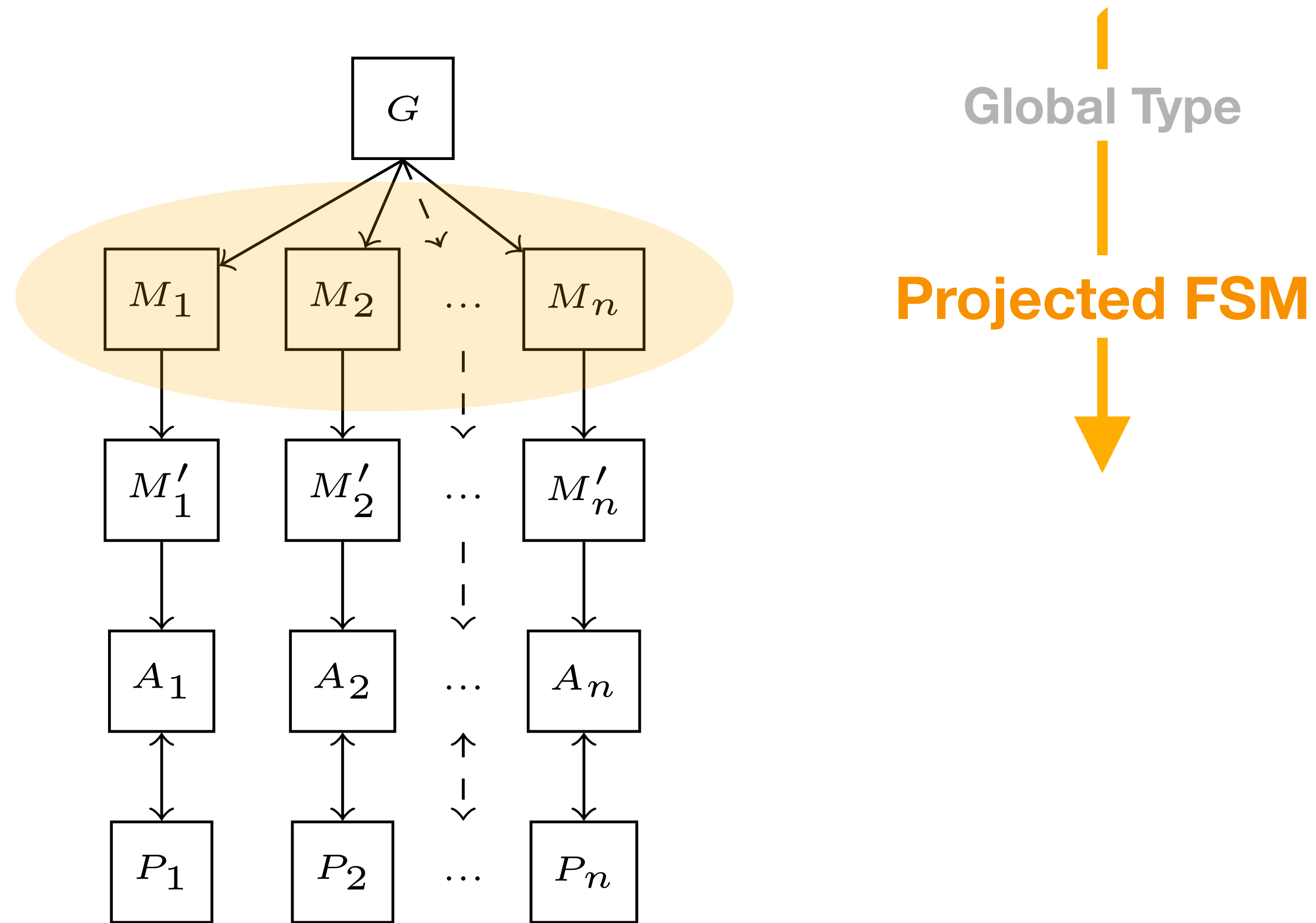
## Top-Down Approach



↓  
Global Type  
↓

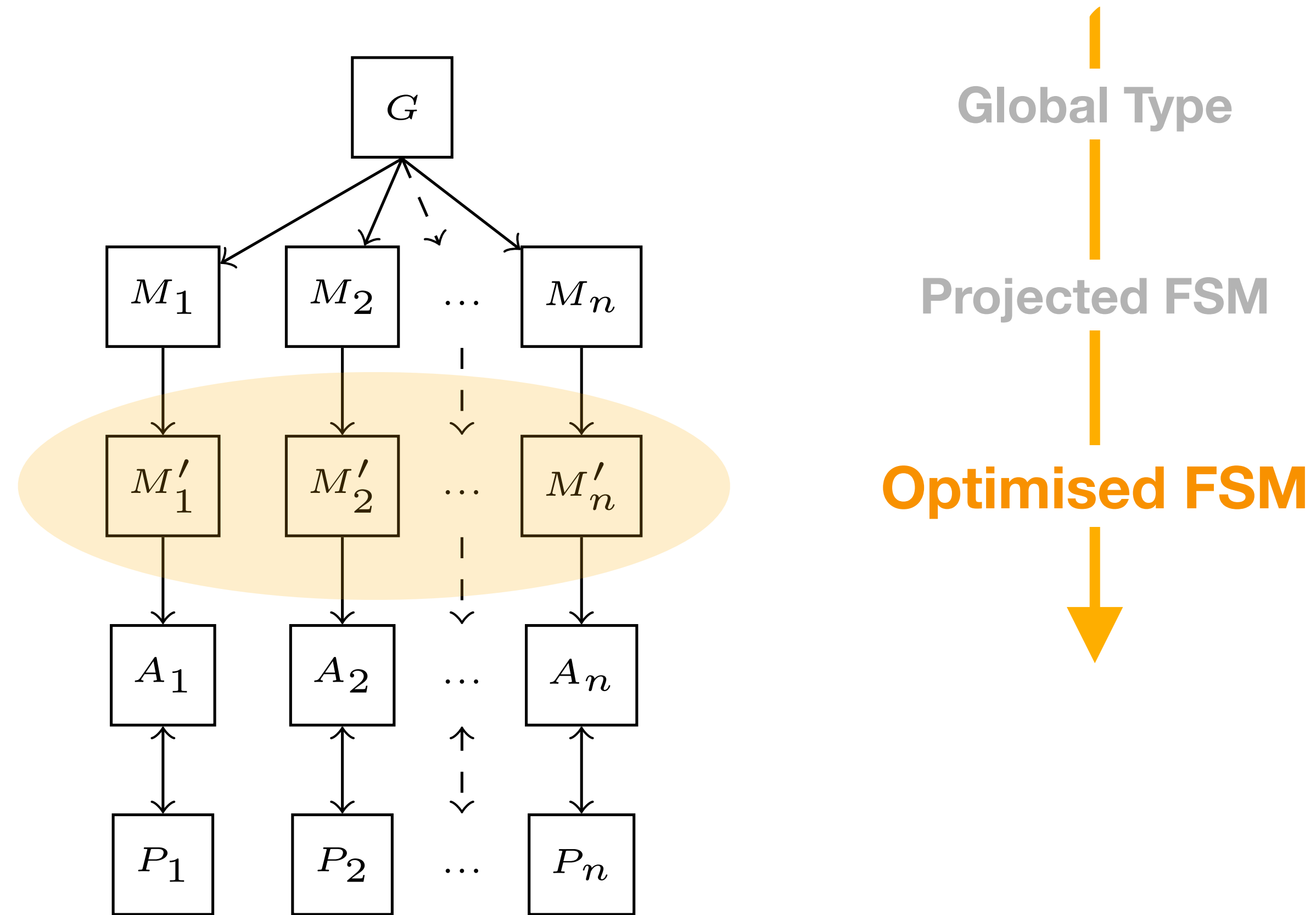
# Workflow

## Top-Down Approach



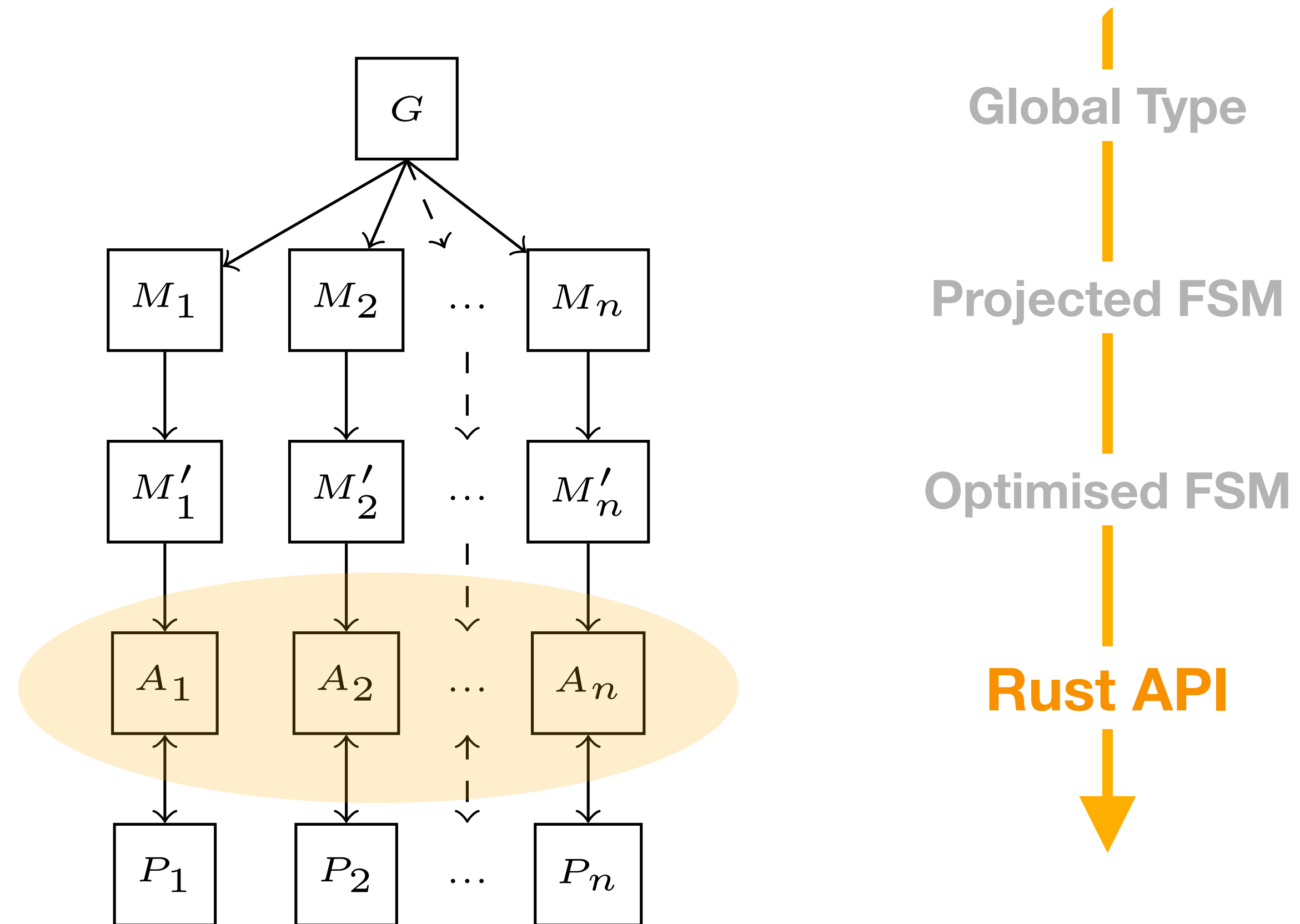
# Workflow

## Top-Down Approach



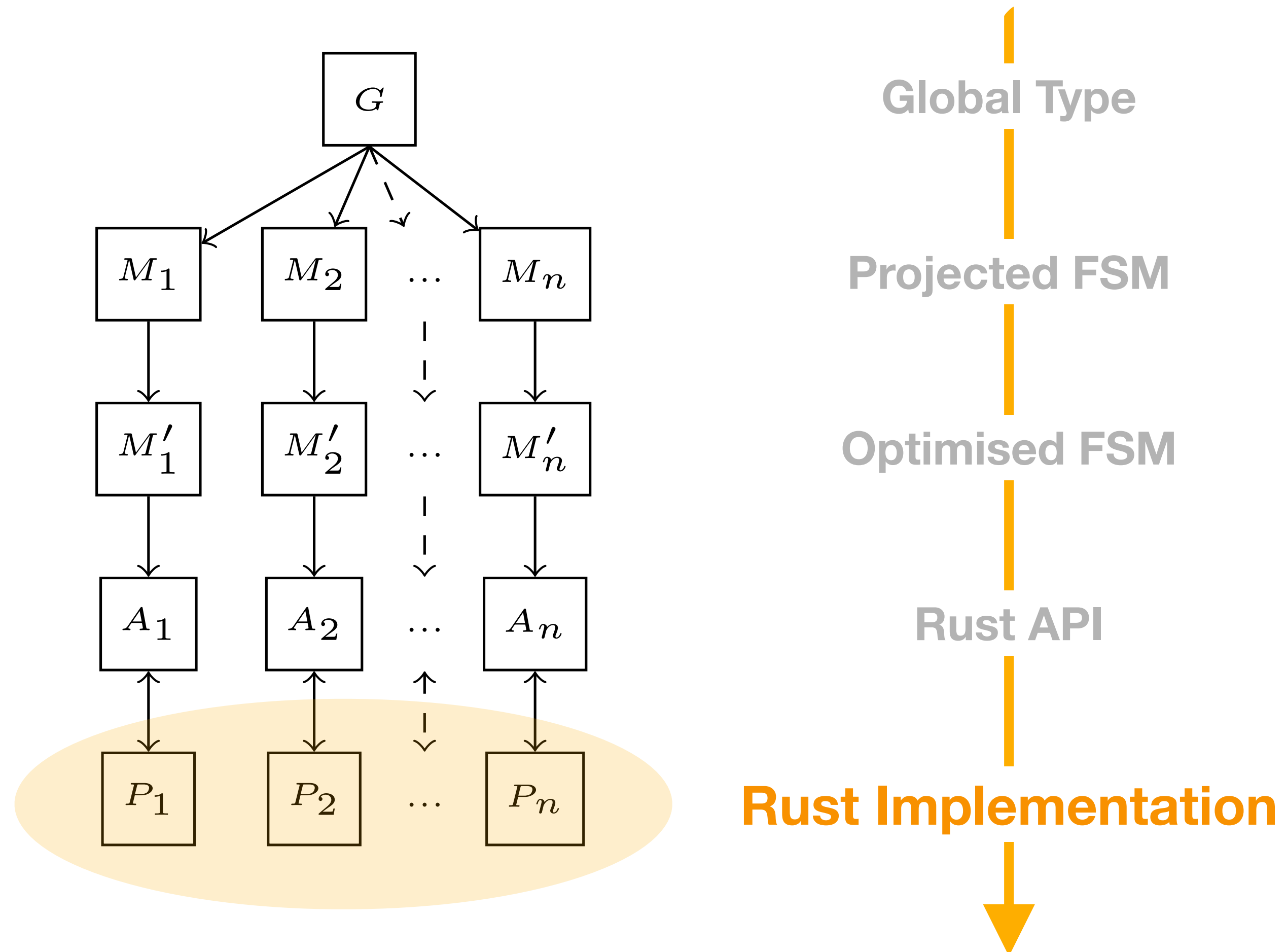
# Workflow

## Top-Down Approach



# Workflow

## Top-Down Approach



# Ring Protocol

## Example

### Global Type

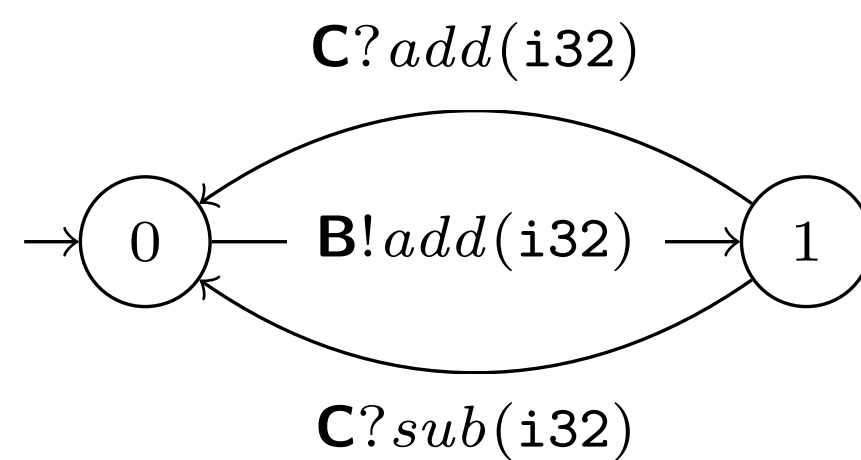
$$G = \mu t. \mathbf{A} \rightarrow \mathbf{B} : \left\{ \begin{array}{l} \mathit{add}(\mathit{i32}).\mathbf{B} \rightarrow \mathbf{C} : \left\{ \begin{array}{l} \mathit{add}(\mathit{i32}).\mathbf{C} \rightarrow \mathbf{A} : \{\mathit{add}(\mathit{i32}).\mathbf{t}\} \\ \mathit{sub}(\mathit{i32}).\mathbf{C}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

# Ring Protocol

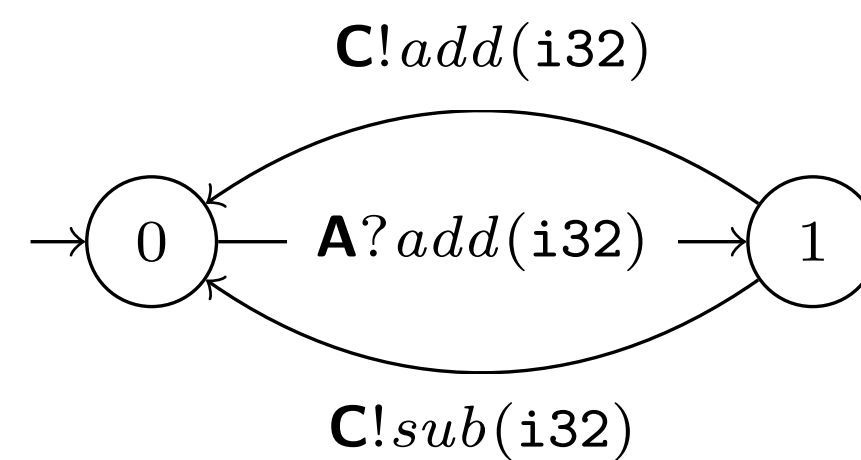
## Example

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

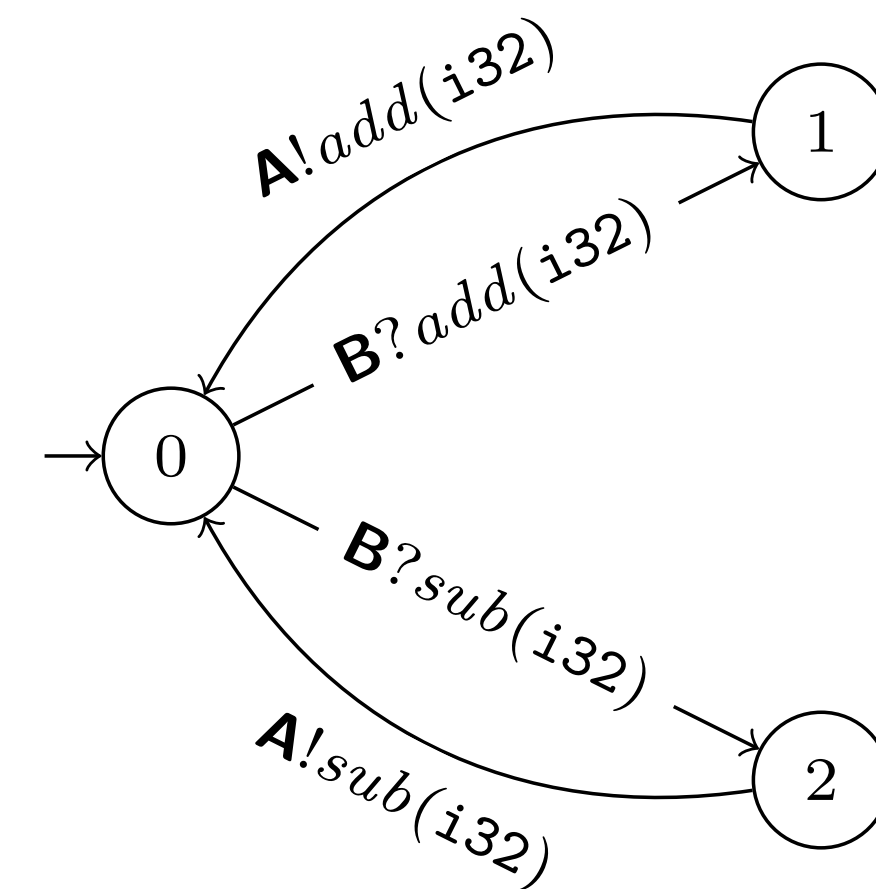
PROJECTION



PROJECTION

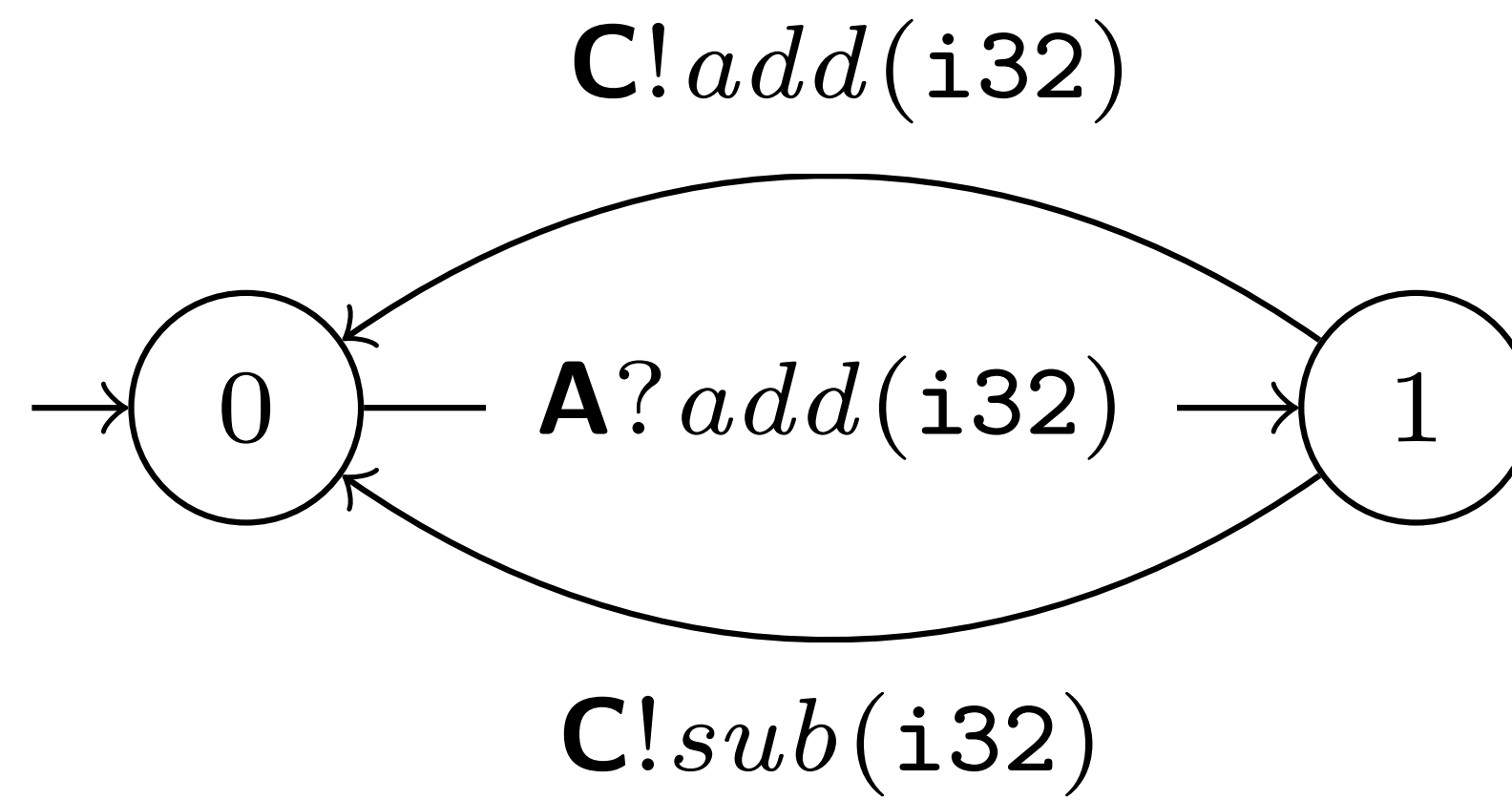


PROJECTION



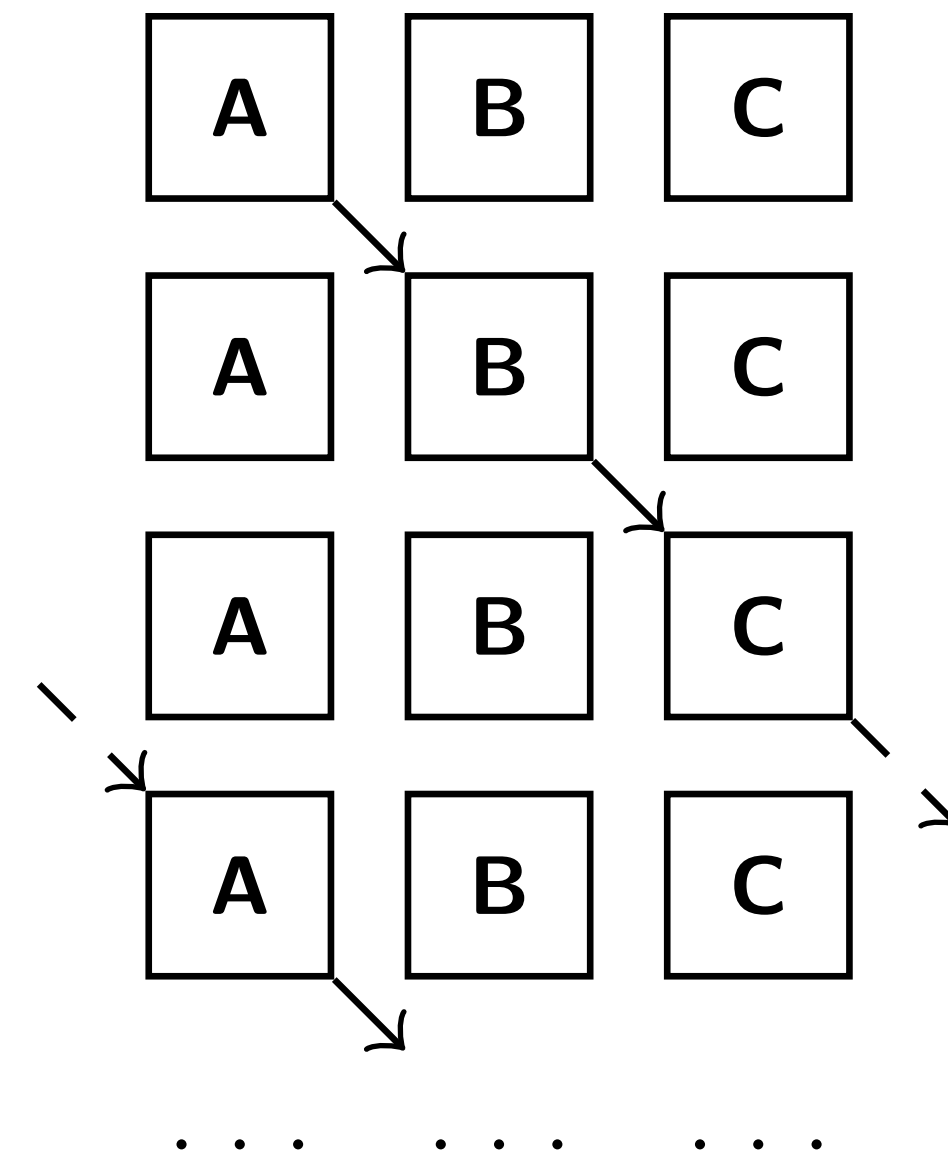
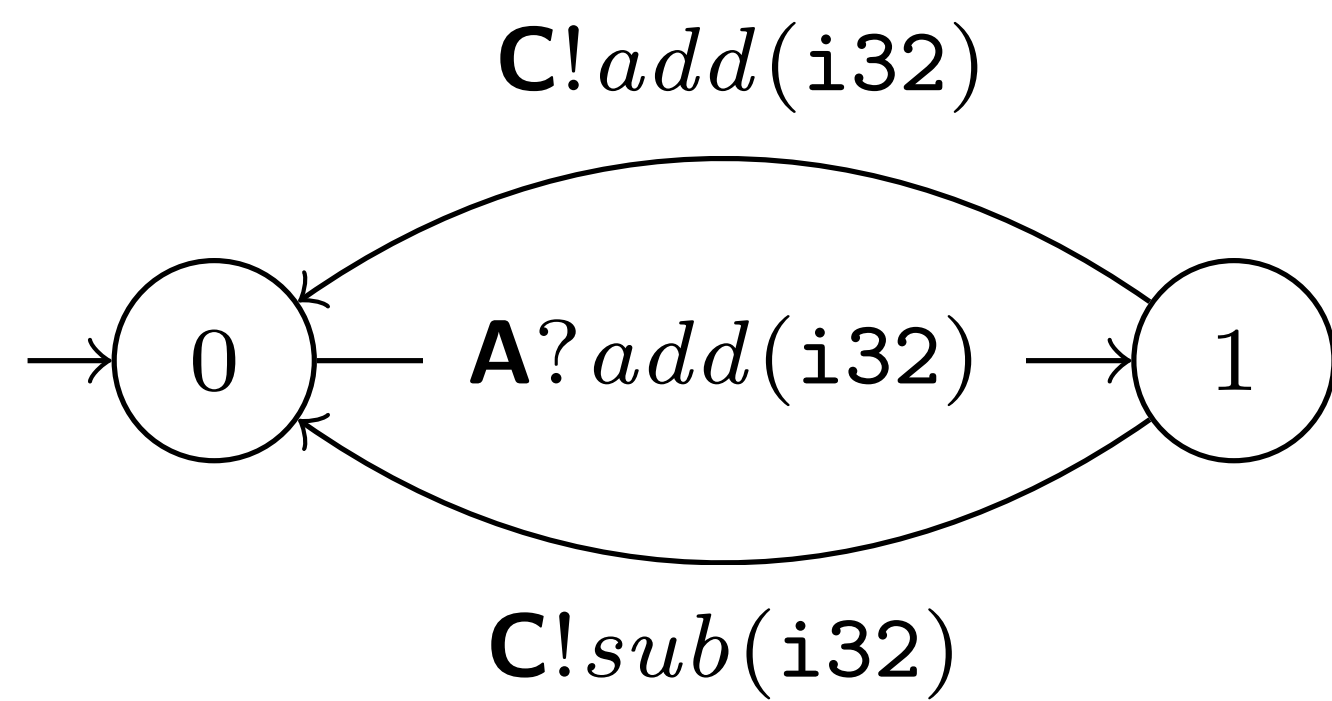
# Ring Protocol

## Example



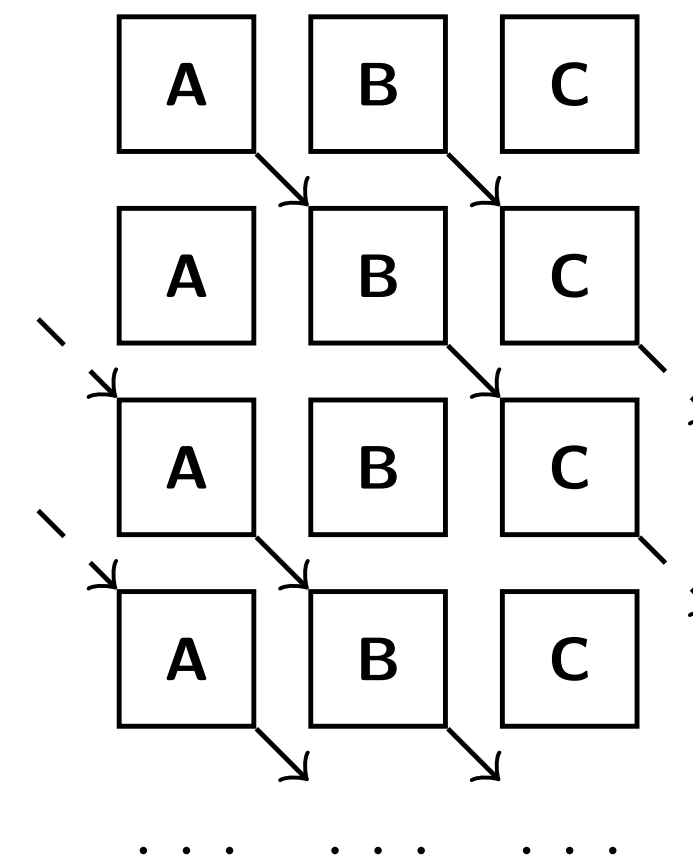
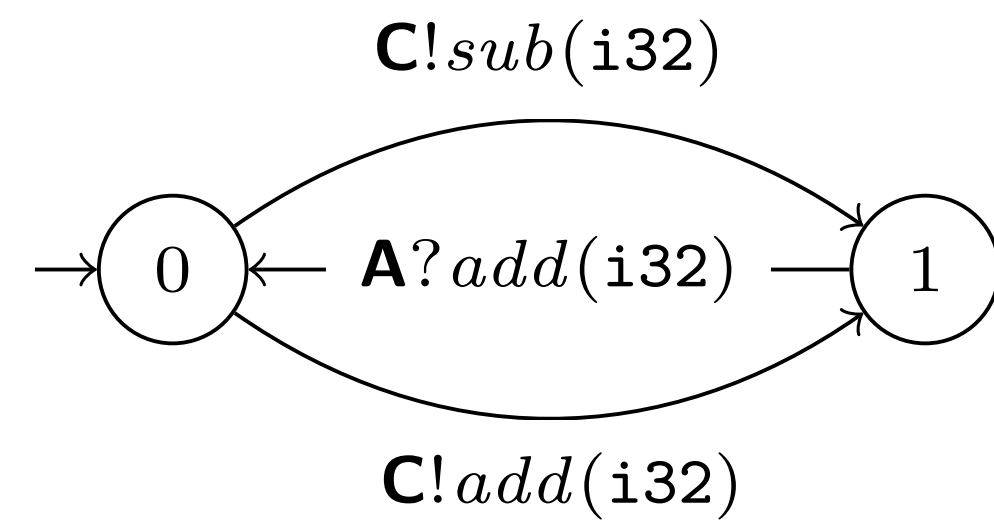
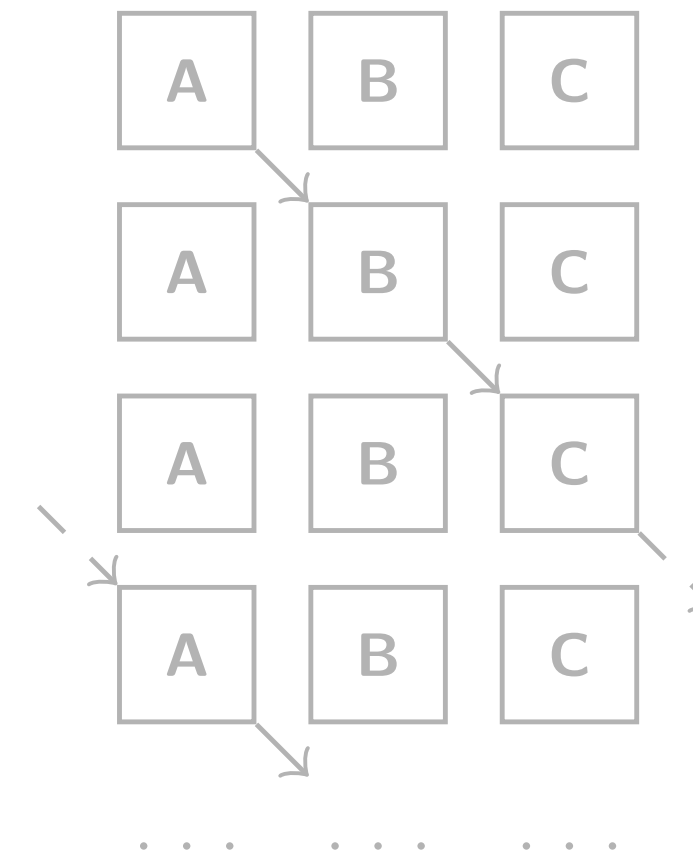
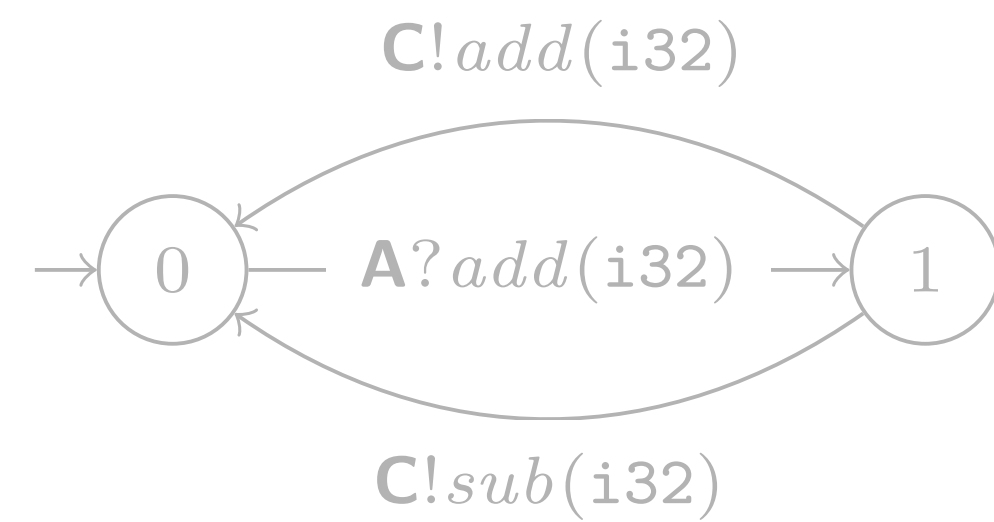
# Ring Protocol

## Example



# Ring Protocol

## Example



# vScr An Extensible Toolchain for Multiparty Session Types

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

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

**Global protocol**

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

**Local types**

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

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

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

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

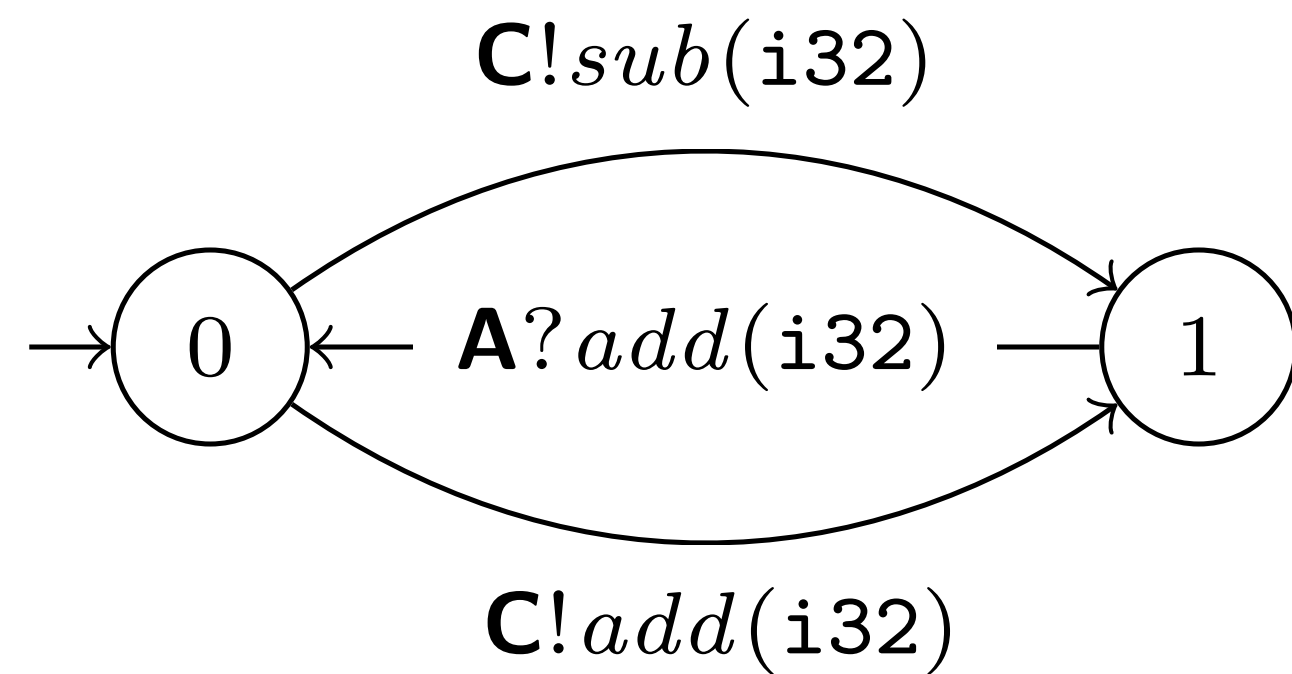
# Scribble

## Protocol Description Language

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

# Ring Protocol

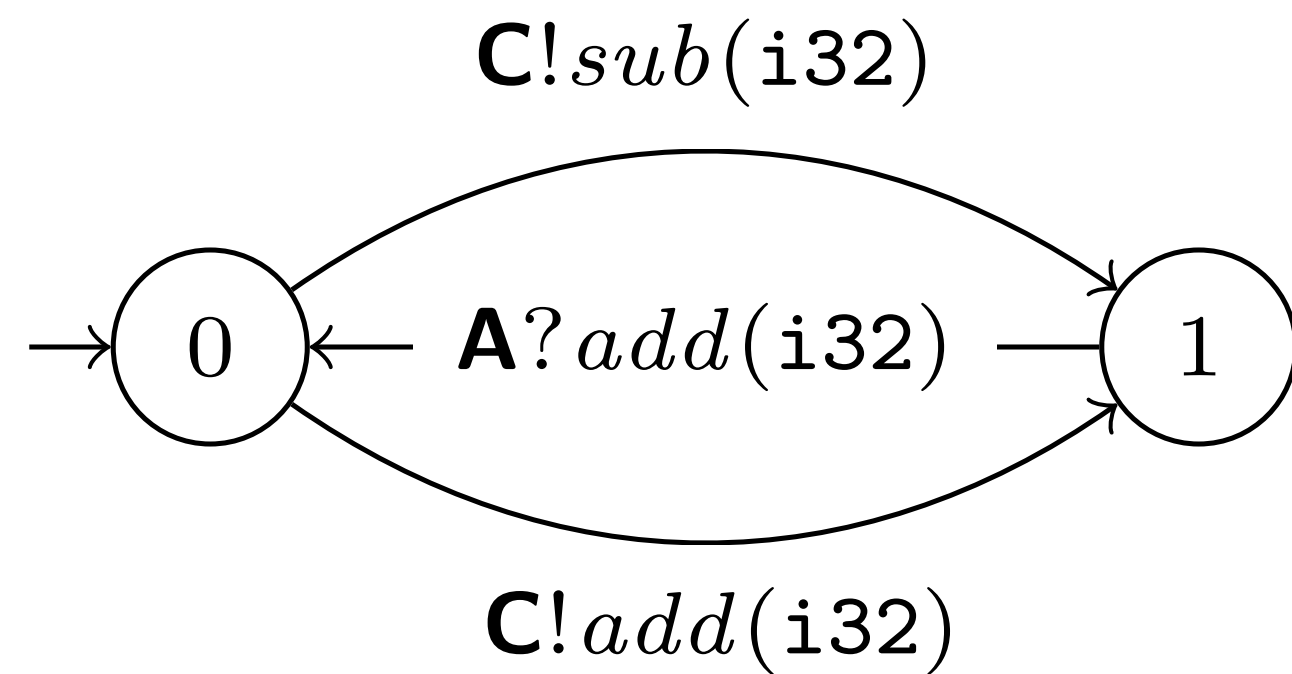
## Rust API



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

# Ring Protocol

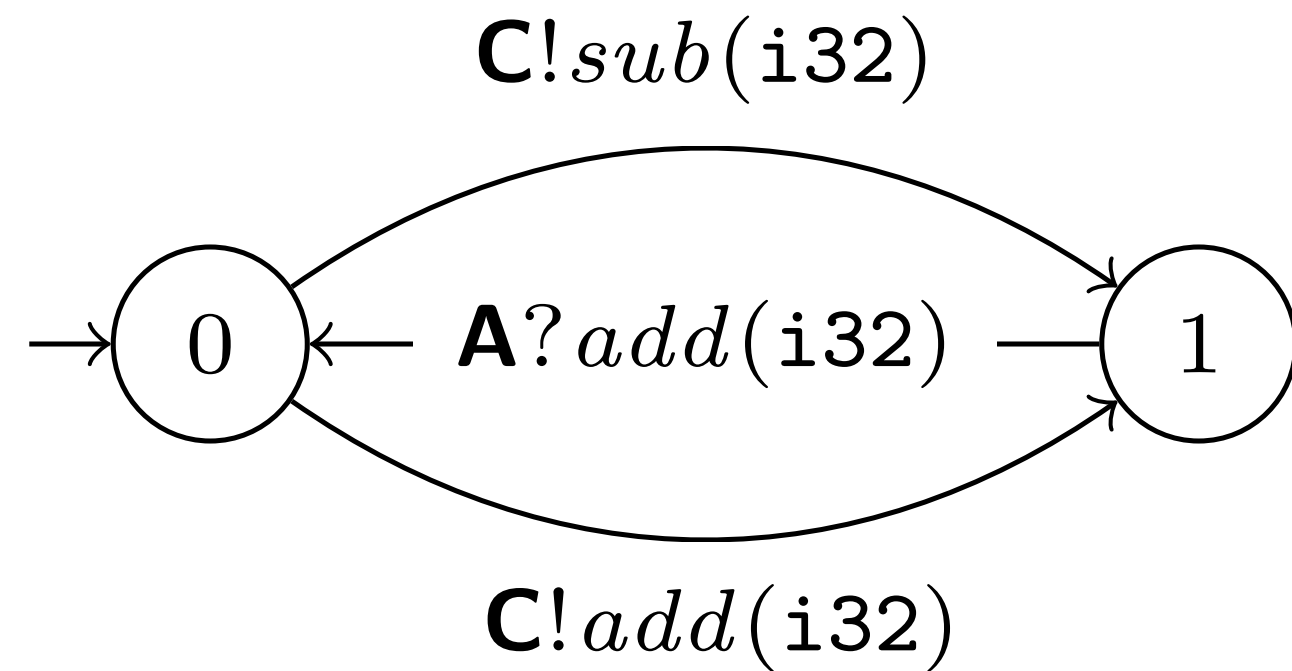
## Rust API



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

# Ring Protocol

## Rust API



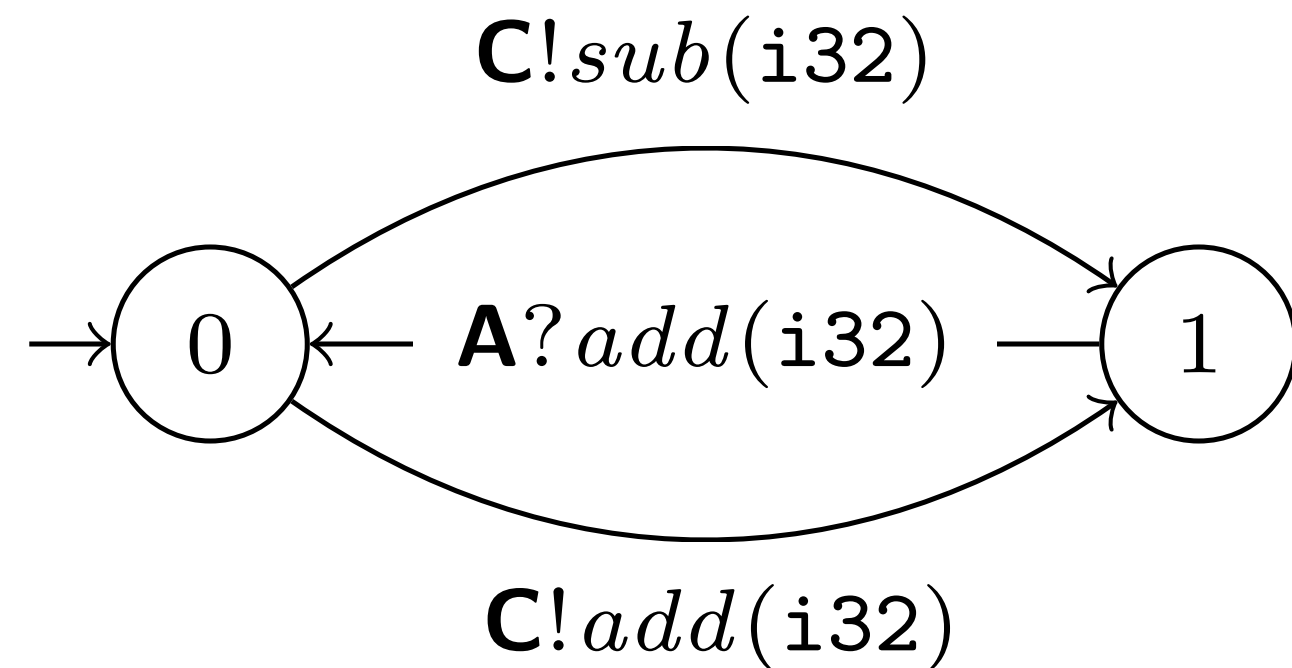
```
#[derive(Role)]  
#[message(Label)]  
struct B(#[route(A)] Receiver, #[route(C)] Sender);
```

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

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

# Ring Protocol

## Rust API



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

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

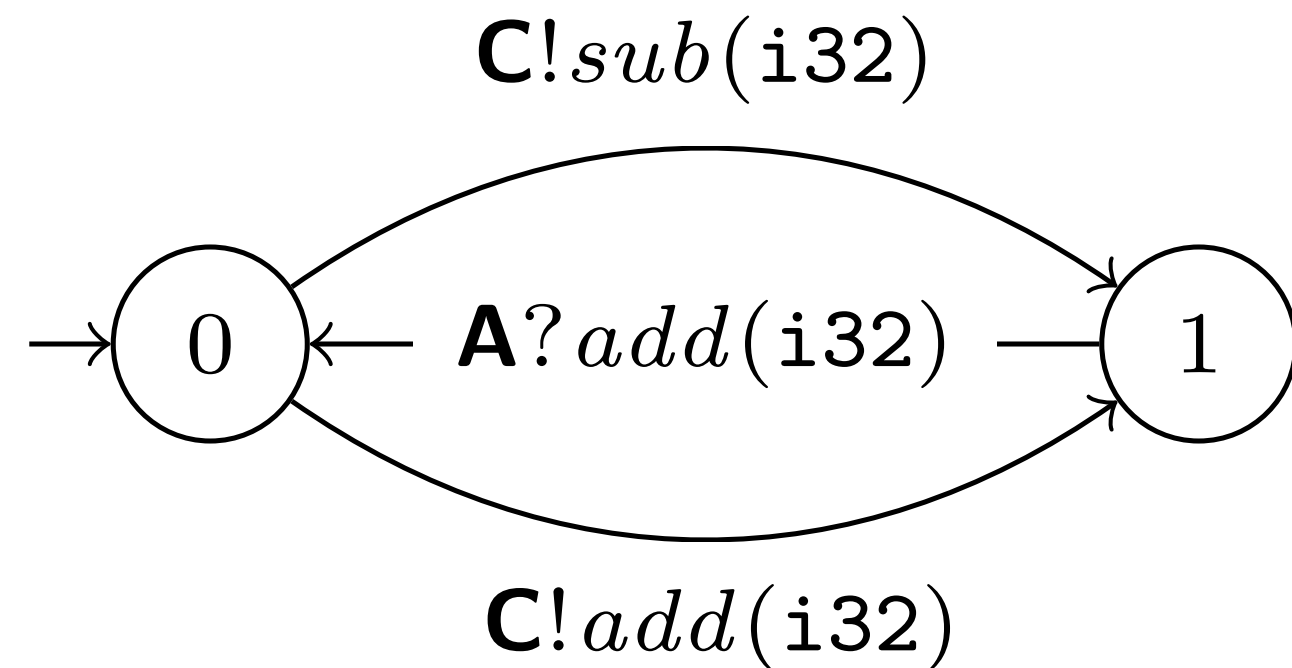
struct Add(i32);
struct Sub(i32);

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

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

# Ring Protocol

## Implementation

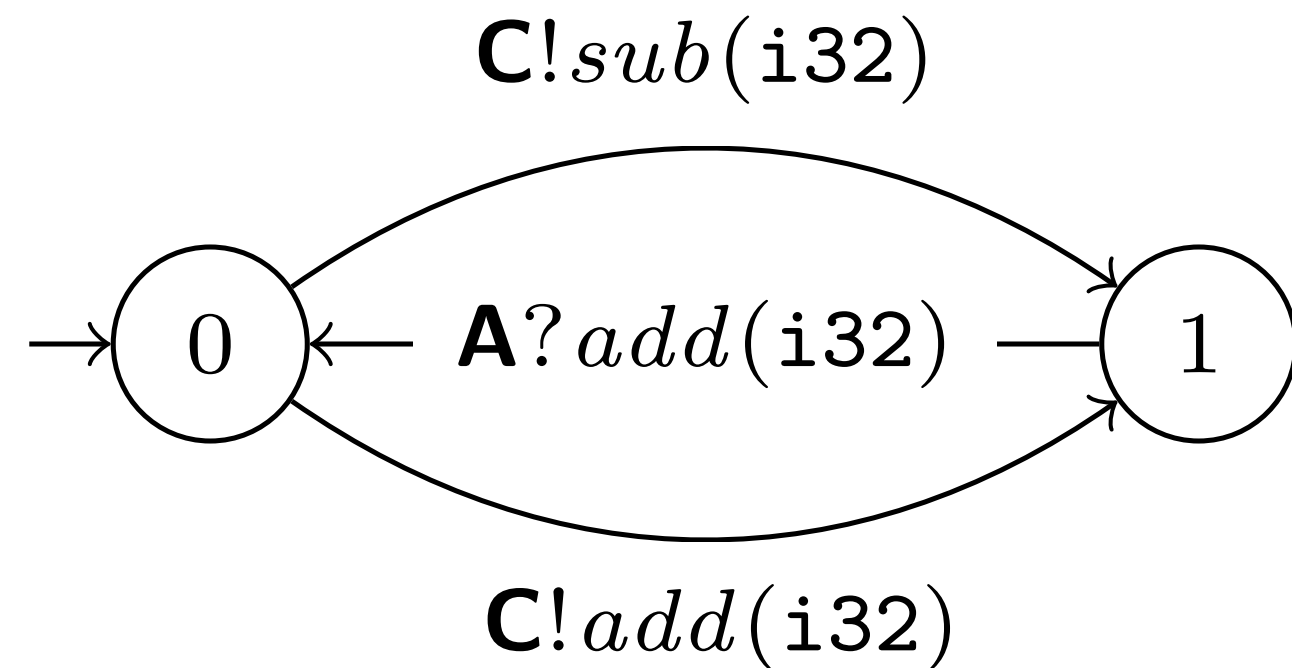


```
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

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

# Ring Protocol

## Implementation

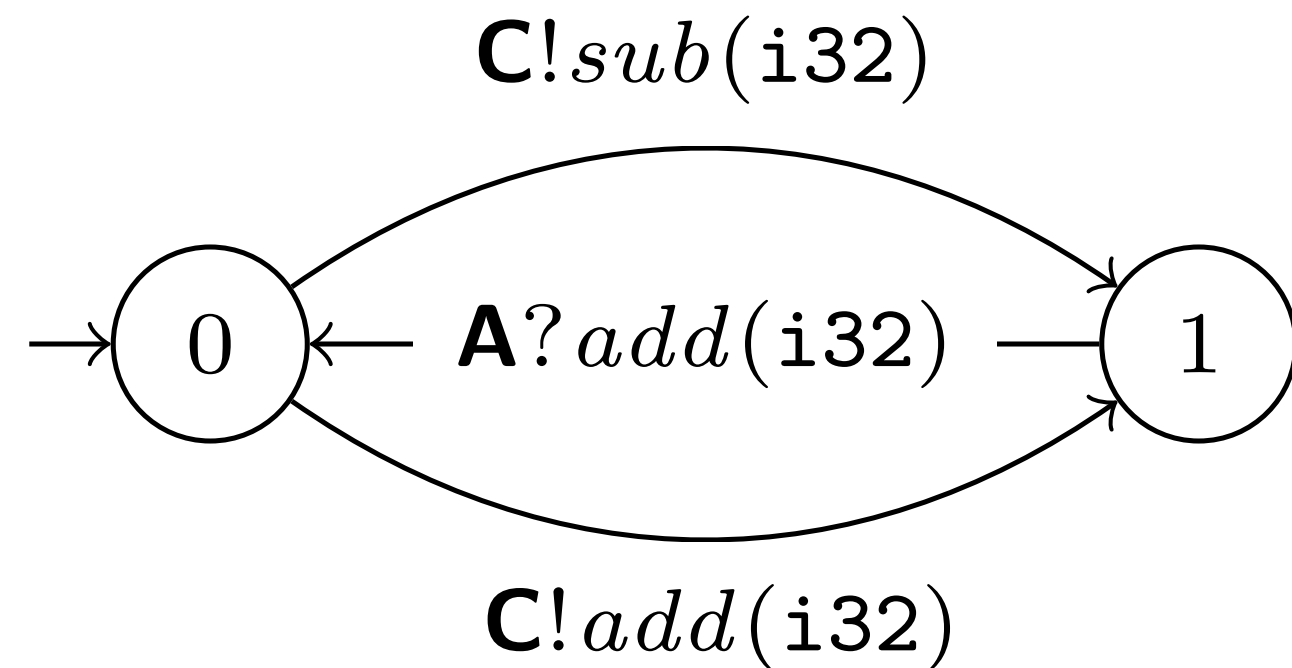


```
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

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

# Ring Protocol

## Implementation

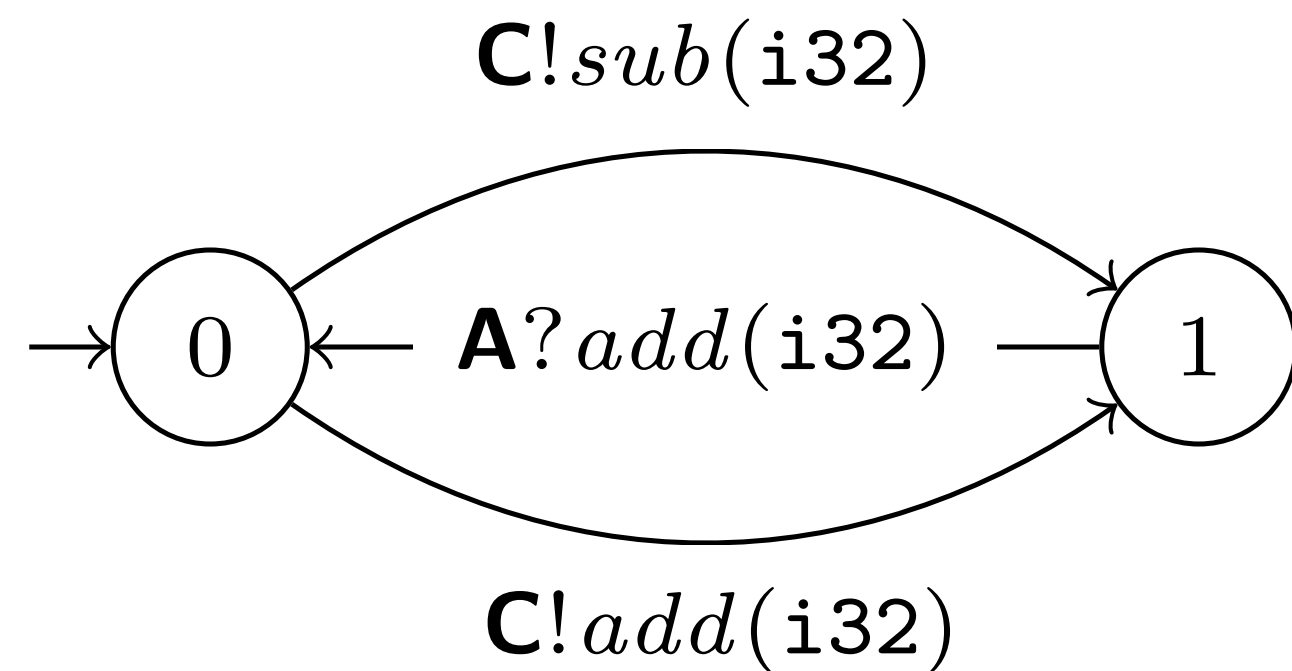


```
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

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

# Ring Protocol

## Implementation

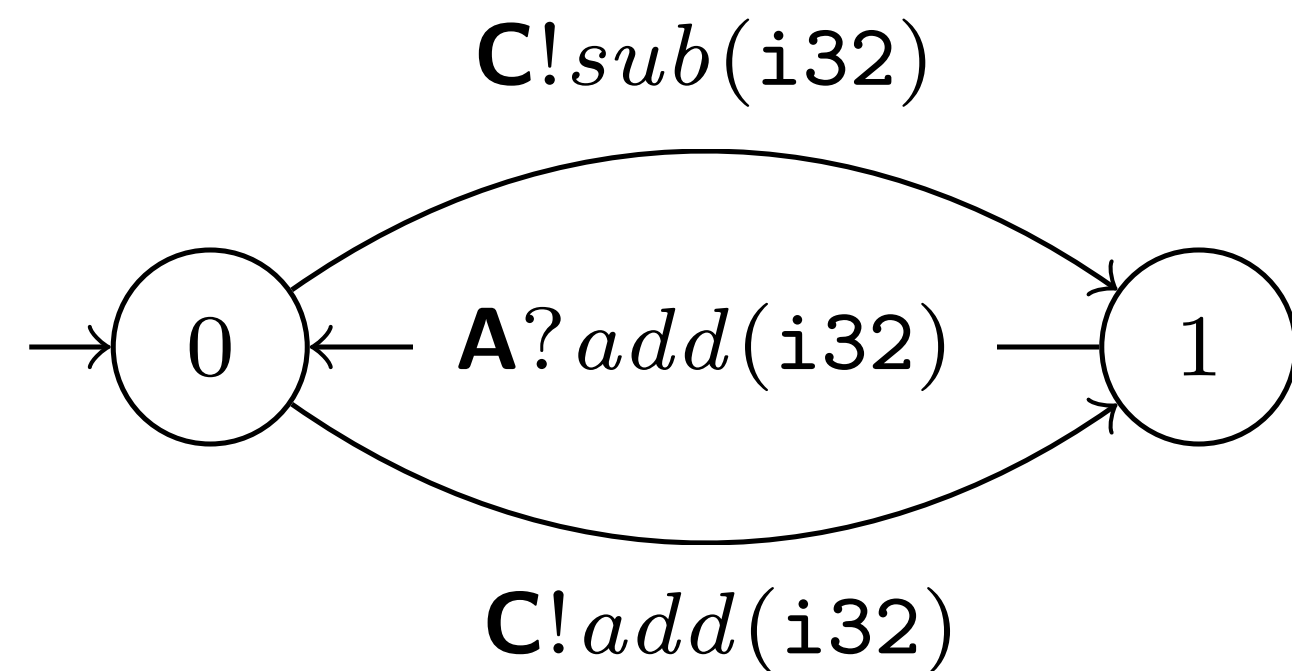


```
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

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

# Ring Protocol

## Implementation

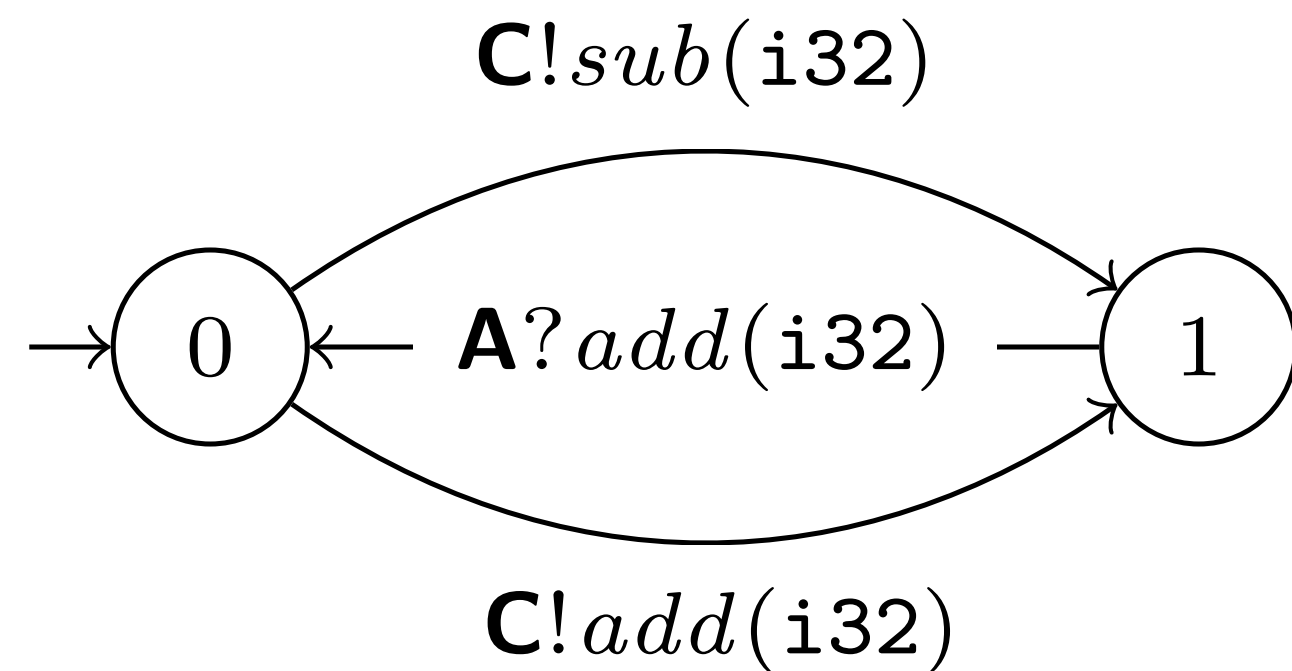


```
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

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

# Ring Protocol

## Implementation

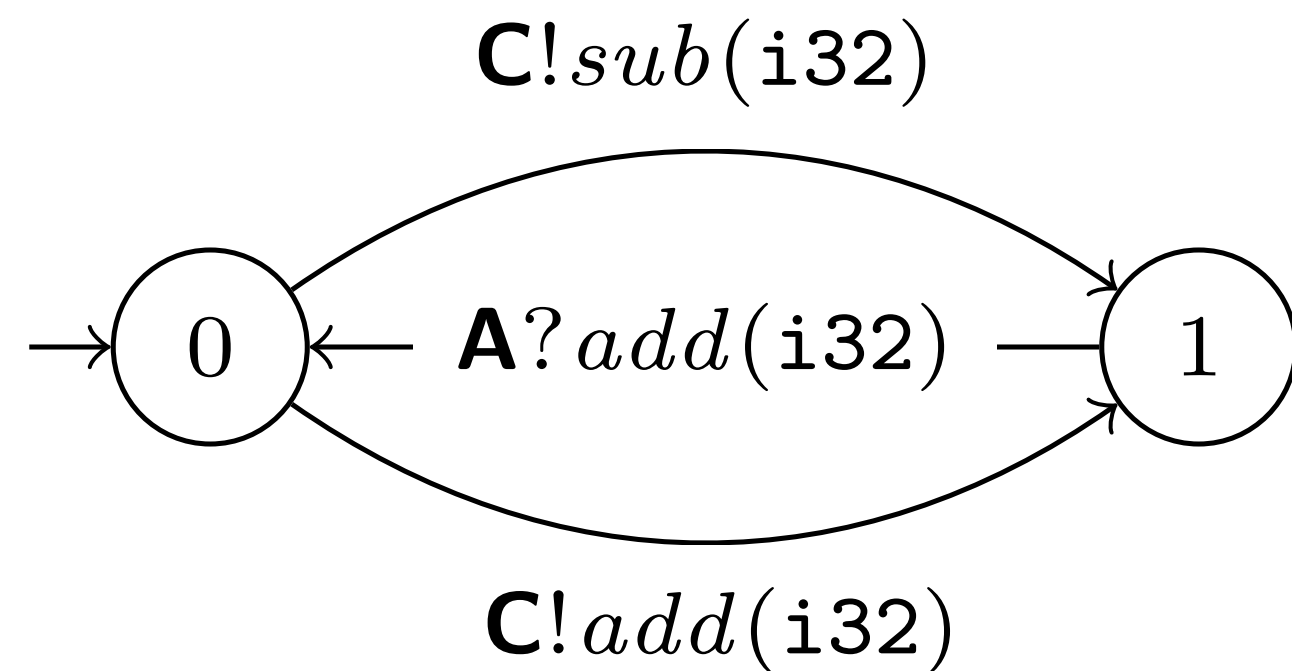


```
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

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

# Ring Protocol

## Implementation

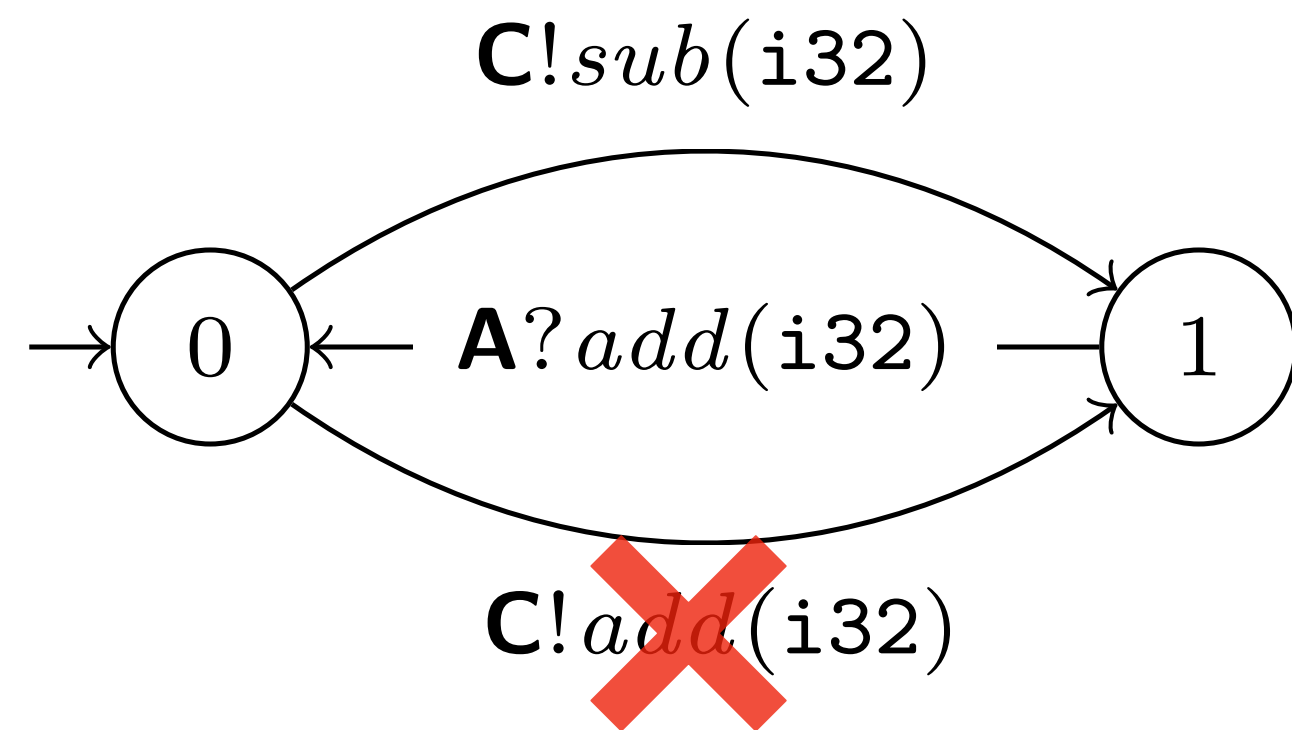


```
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

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

# Ring Protocol

## Implementation



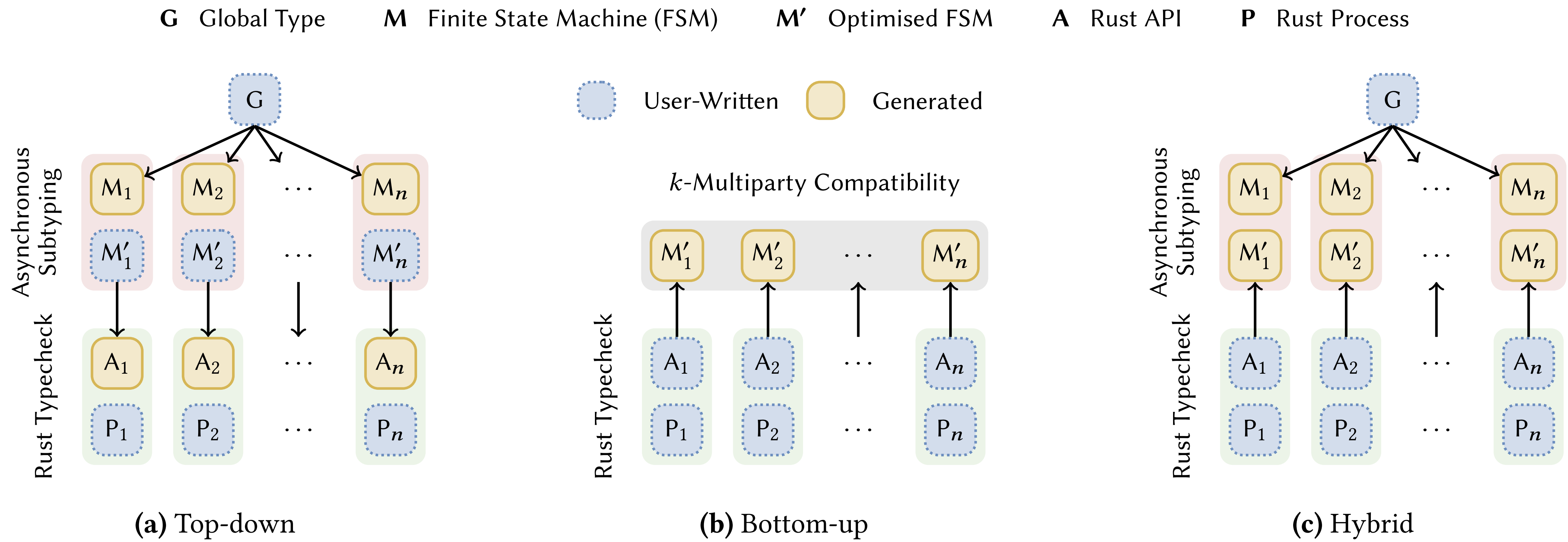
```
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

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

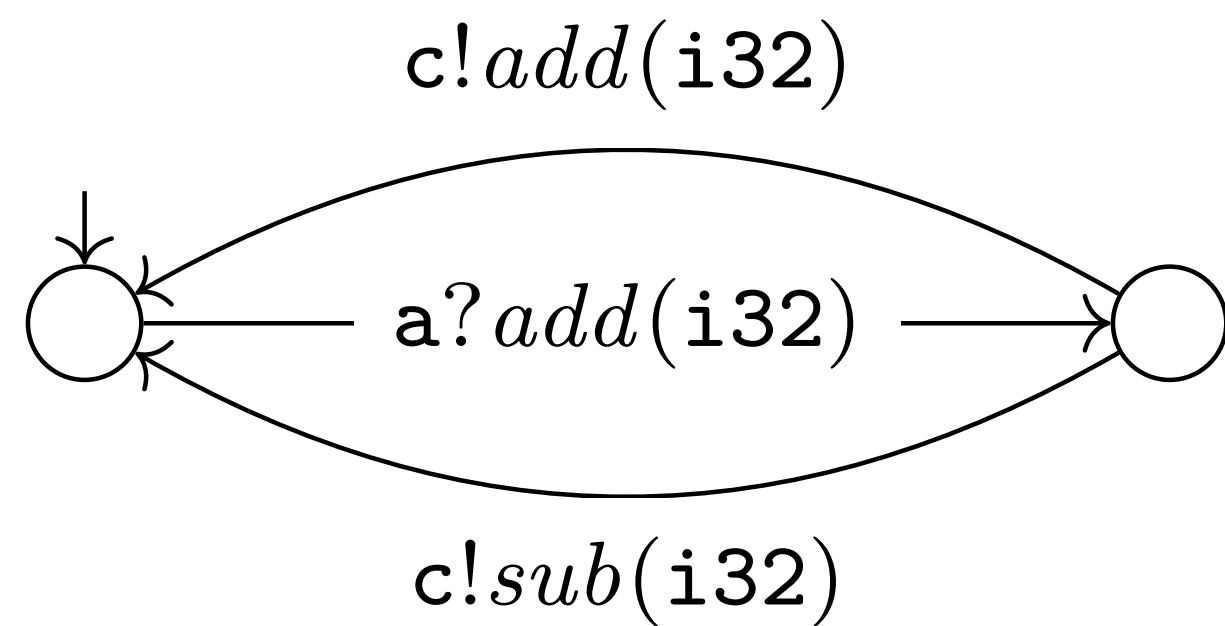
How do we check that asynchronous reorderings are **safe**?

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

# Safety

## Asynchronous Subtyping

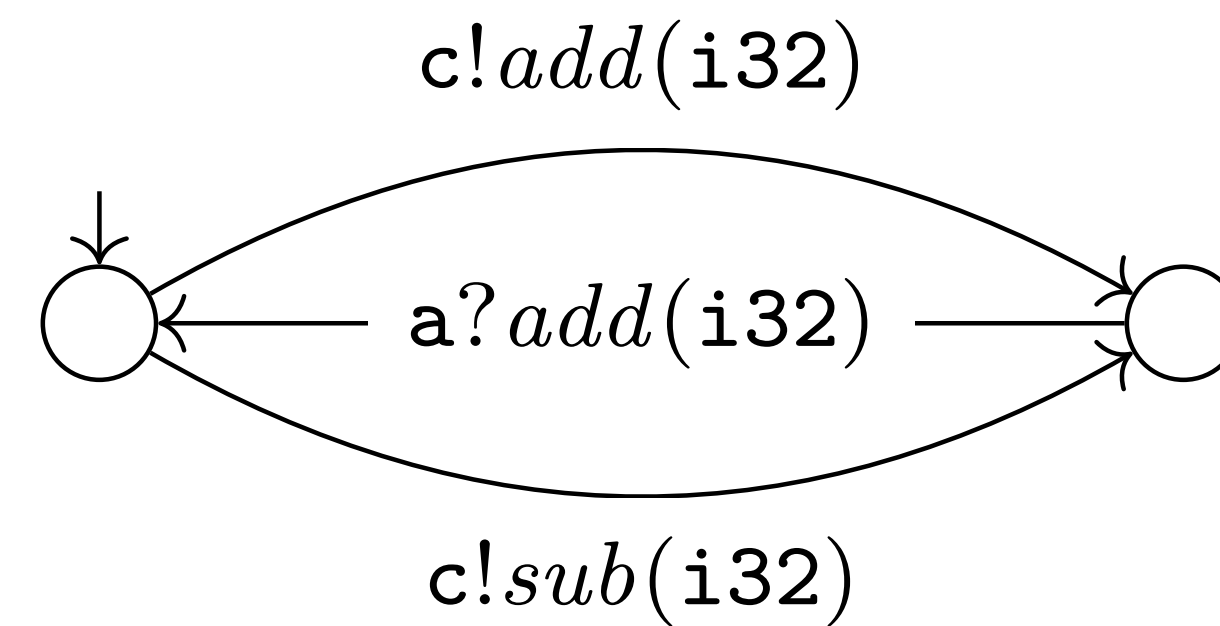
PROJECTED B



Safe?



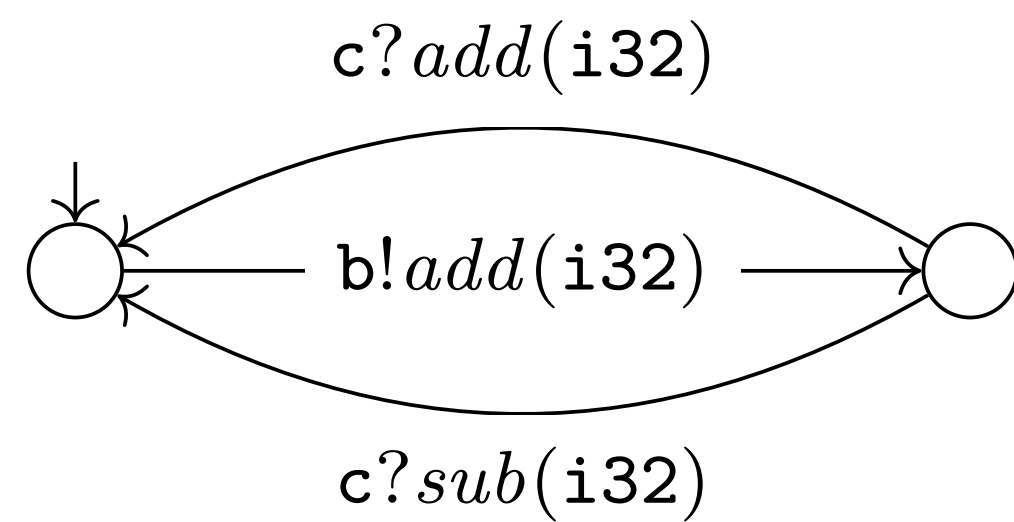
OPTIMISED B



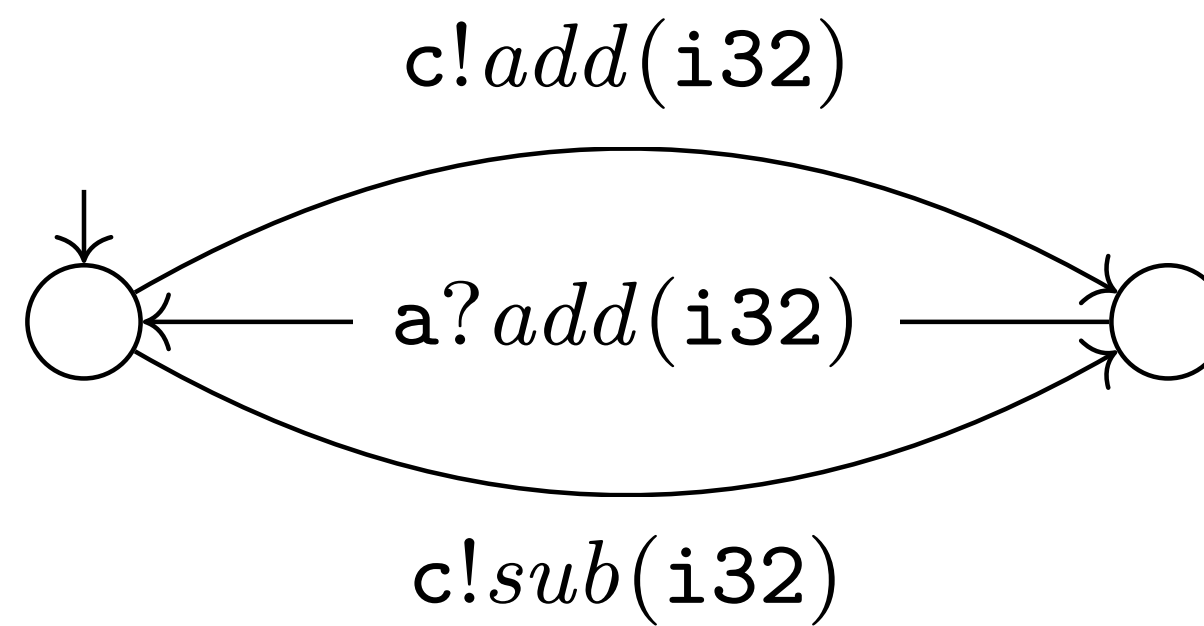
# Safety

## *k*-Multiparty Compatibility

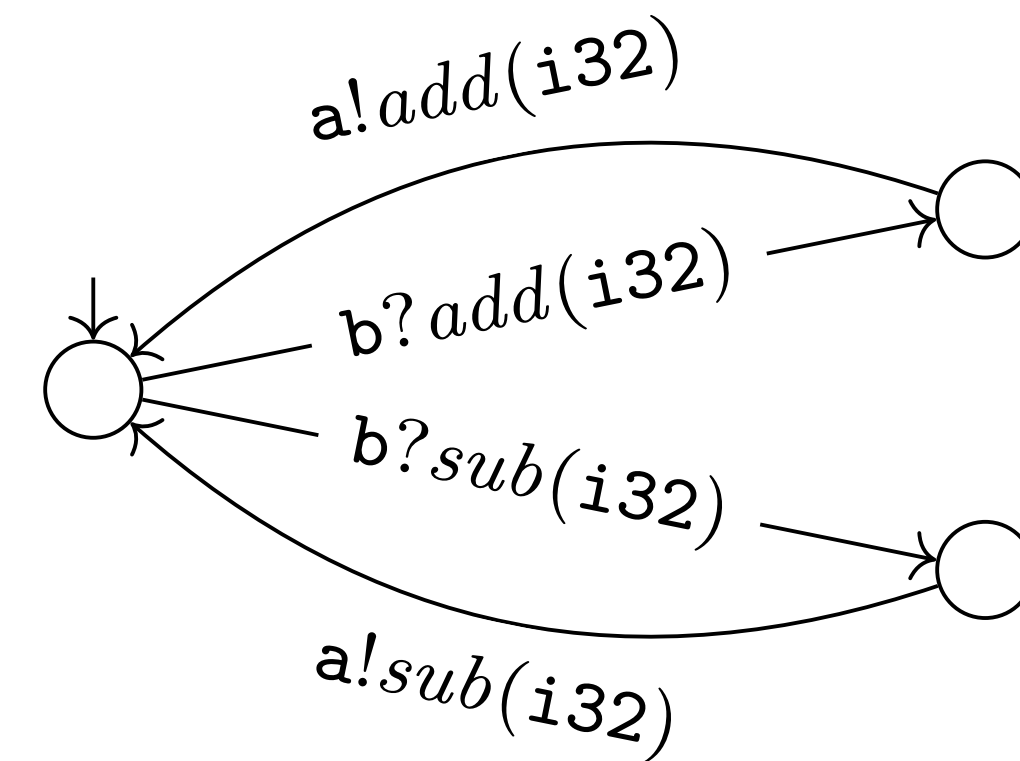
OPTIMISED A



OPTIMISED B

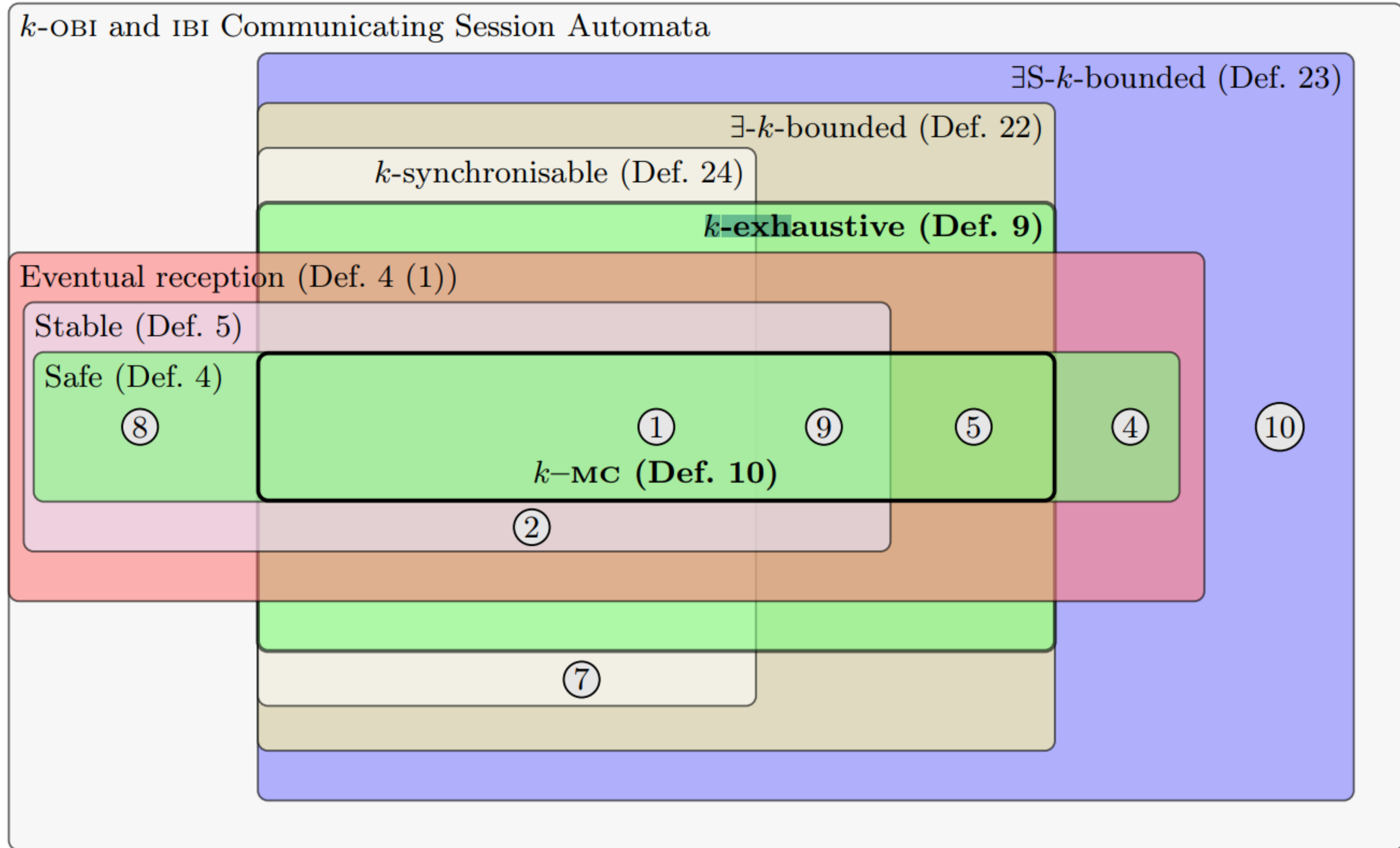


OPTIMISED C






Safe?

# k-Multiparty Compatibility [CAV'19]



# Asynchronous Subtyping

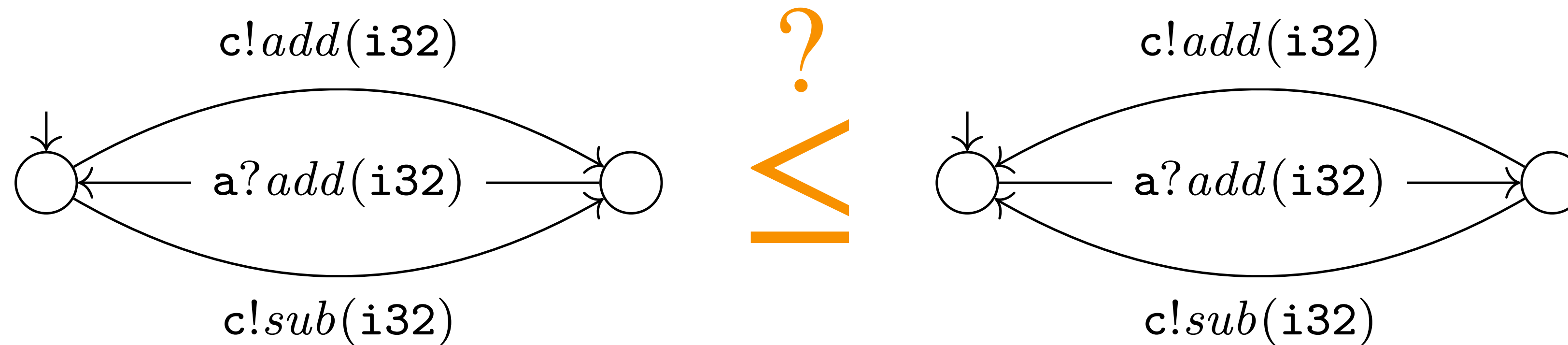
## Existing work

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



# 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  $S'_r \leq S_r$ .

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

# 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

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

$\mathcal{A}^{(p)}$  ↑

# Asynchronous Subtyping

## Reduction Rules

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

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

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

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

# Asynchronous Subtyping

## Reduction Rules

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

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

# Asynchronous Subtyping

## Reduction Rules

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

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

# Asynchronous Subtyping

## Reduction Rules

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

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

# Asynchronous Subtyping

## Reduction Rules

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

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

# Asynchronous Subtyping

## Reduction Rules

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

# Asynchronous Subtyping

## Reduction Rules

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

# Asynchronous Subtyping

## Reduction Rules

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

# Asynchronous Subtyping

## Reduction Rules

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

$\uparrow$   
 $\mathcal{B}(p)$

# Asynchronous Subtyping

## Reduction Rules

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

$\uparrow$   
 $\mathcal{B}(p)$

# 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  $O(\min(|\pi|, |\pi'|))$ .*

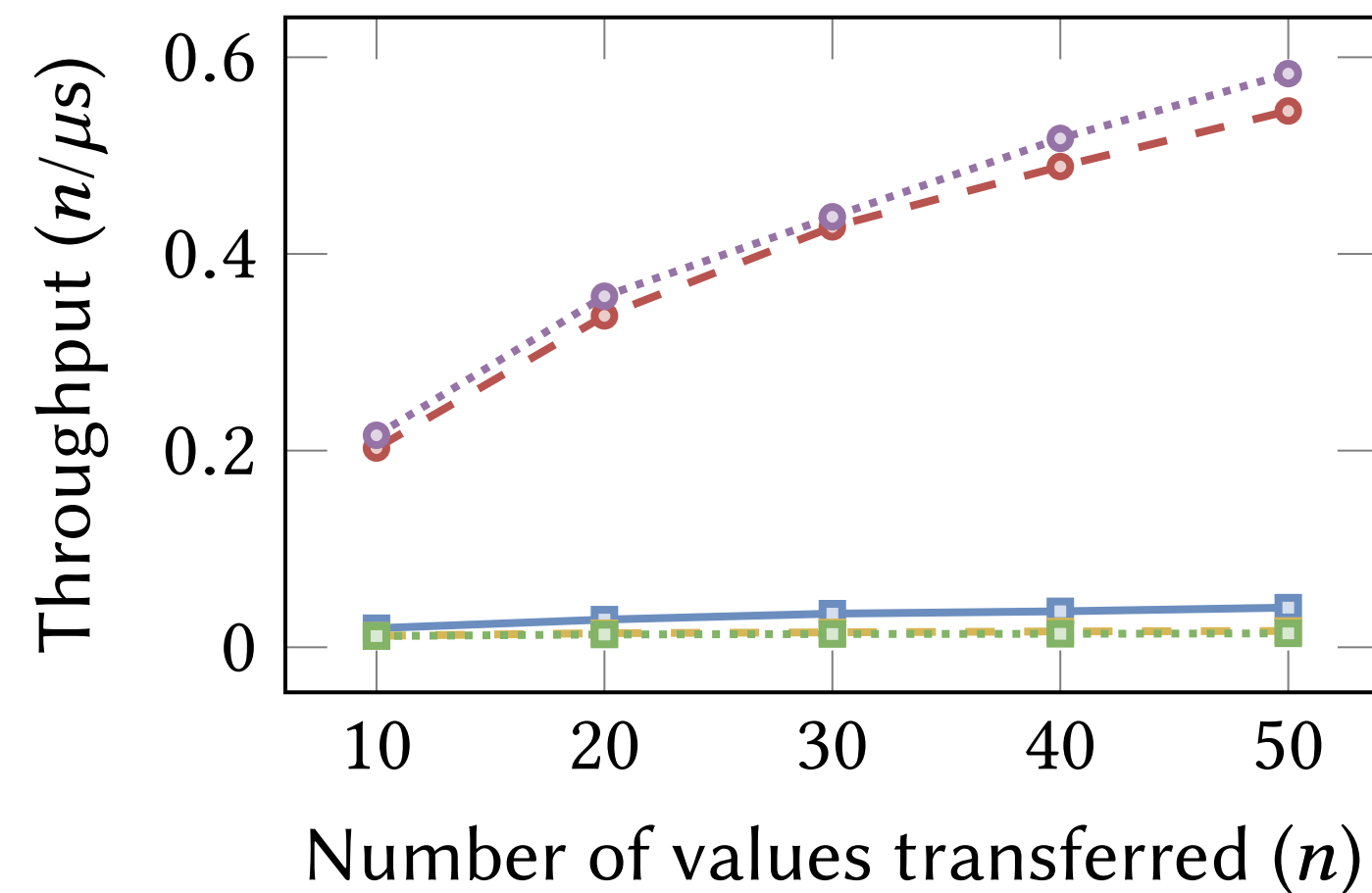
**Theorem 7 (Complexity).** *Consider  $T$  and  $T'$  as (possibly infinite) trees  $\mathcal{T}(T)$  and  $\mathcal{T}(T')$  with asymptotic branching factors  $b$  and  $b'$  respectively. Our algorithm has time complexity  $O(n \min(b, b')^n)$  and space complexity  $O(n \min(b, b'))$  in the worst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

# Evaluation

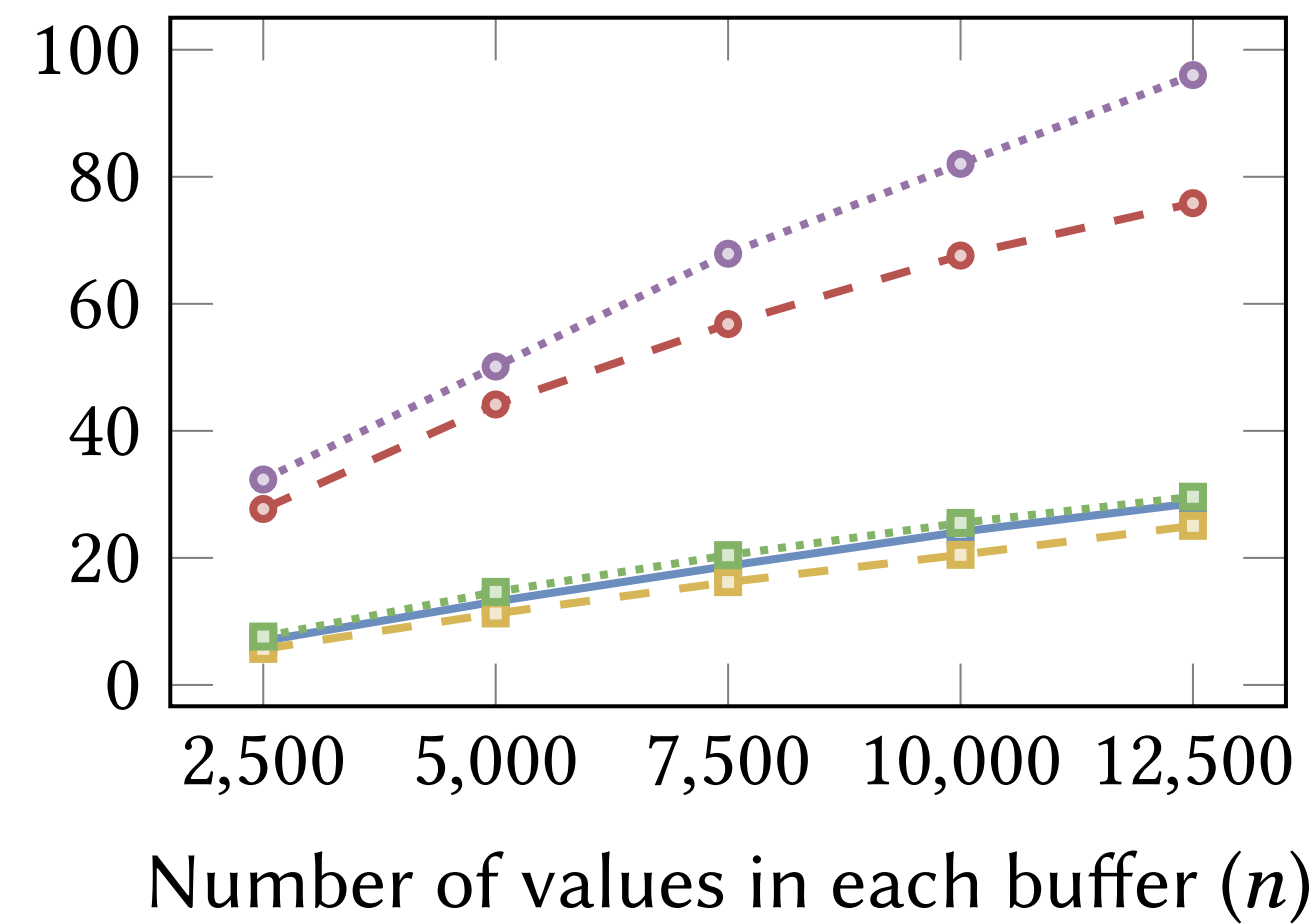
## Rust Framework Benchmarks

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

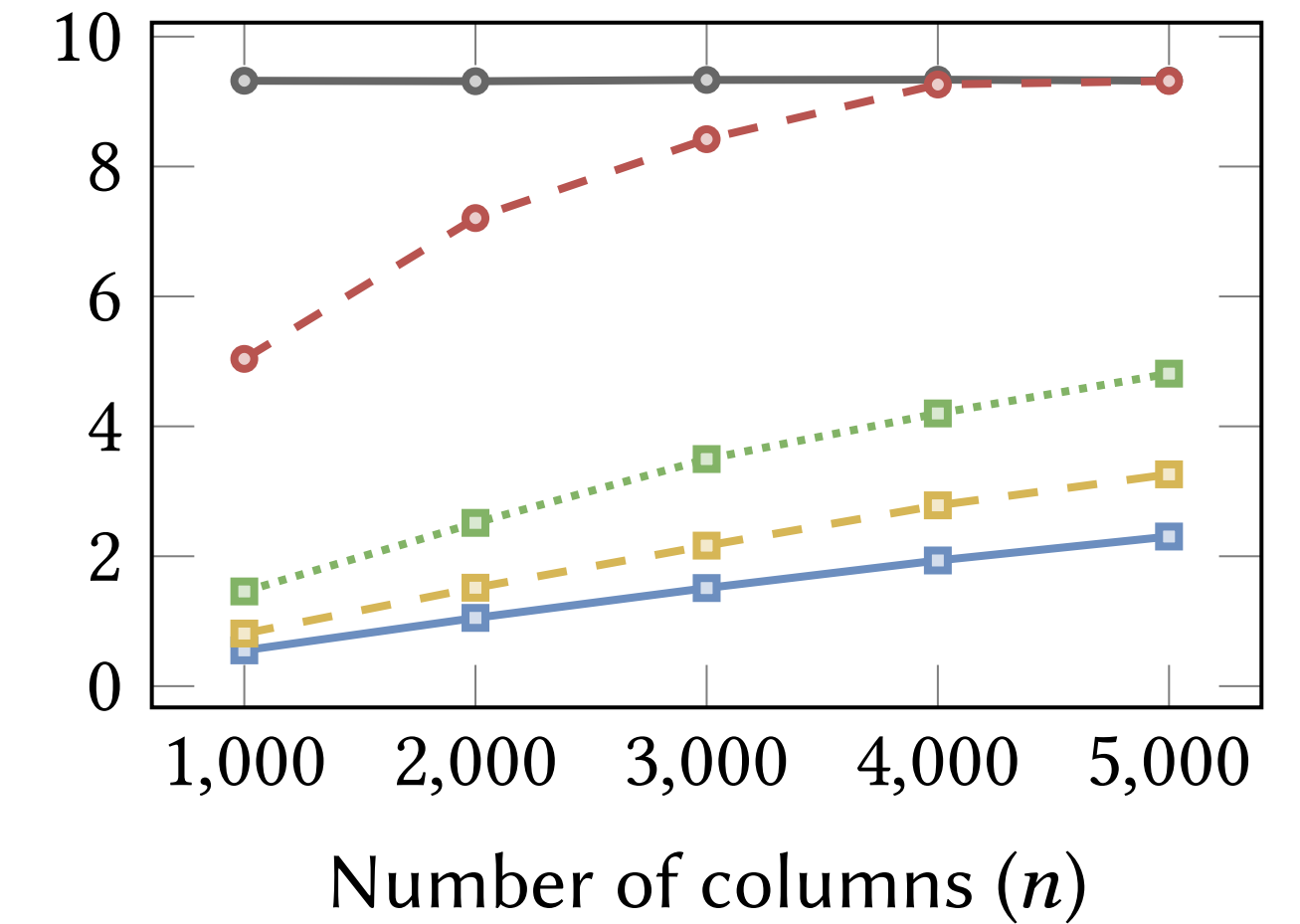
*Stream*



*Double Buffering*

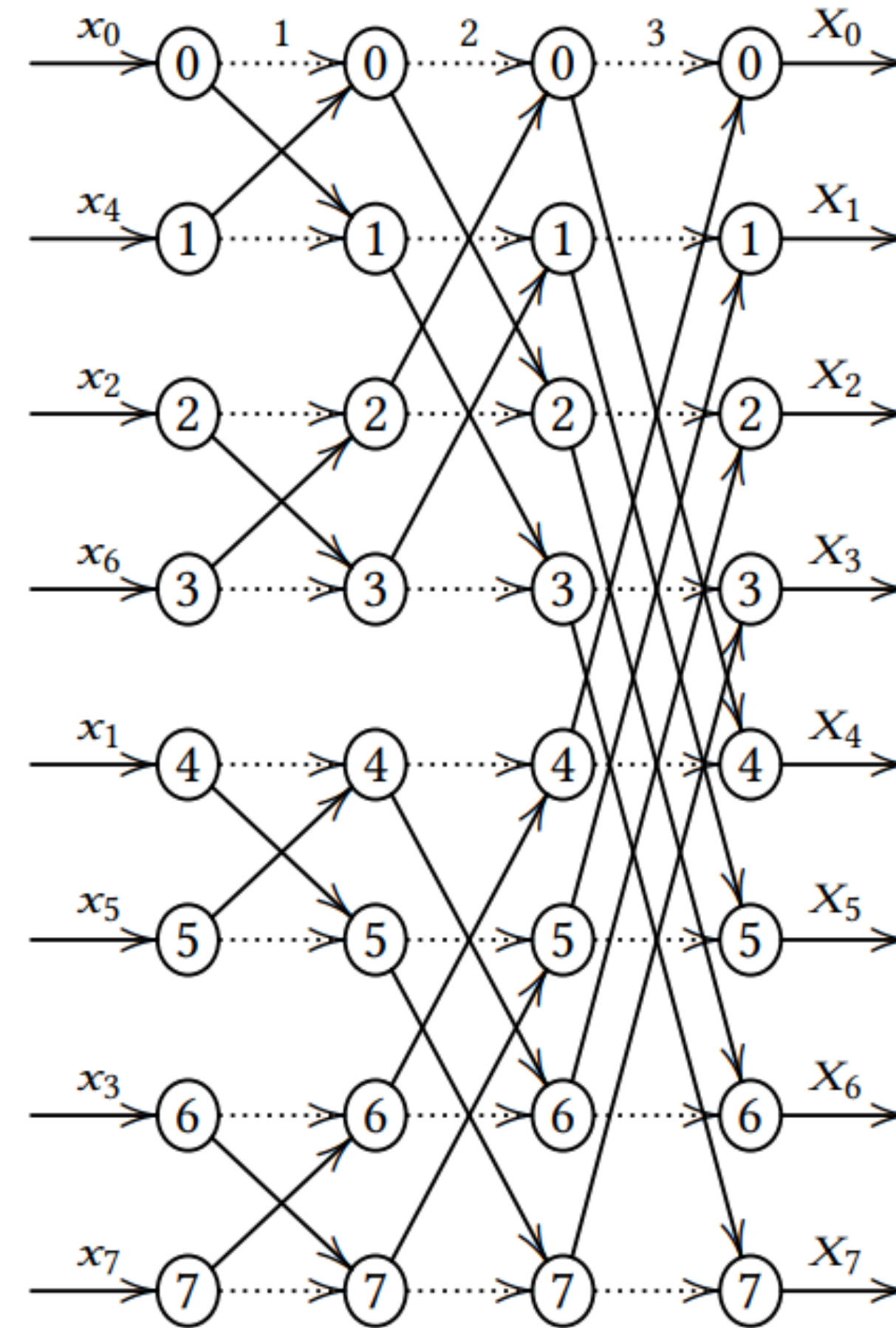
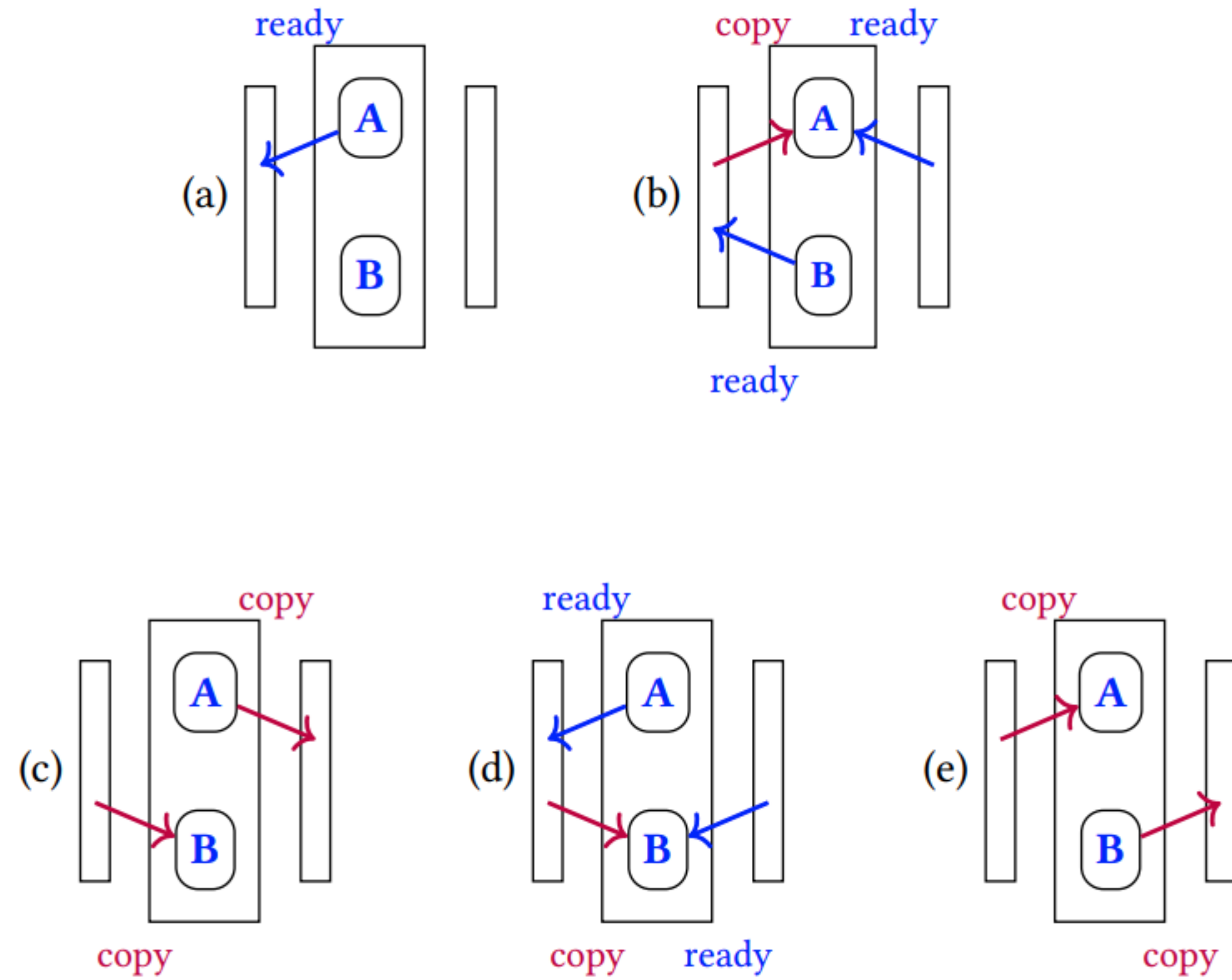


*FFT*



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.

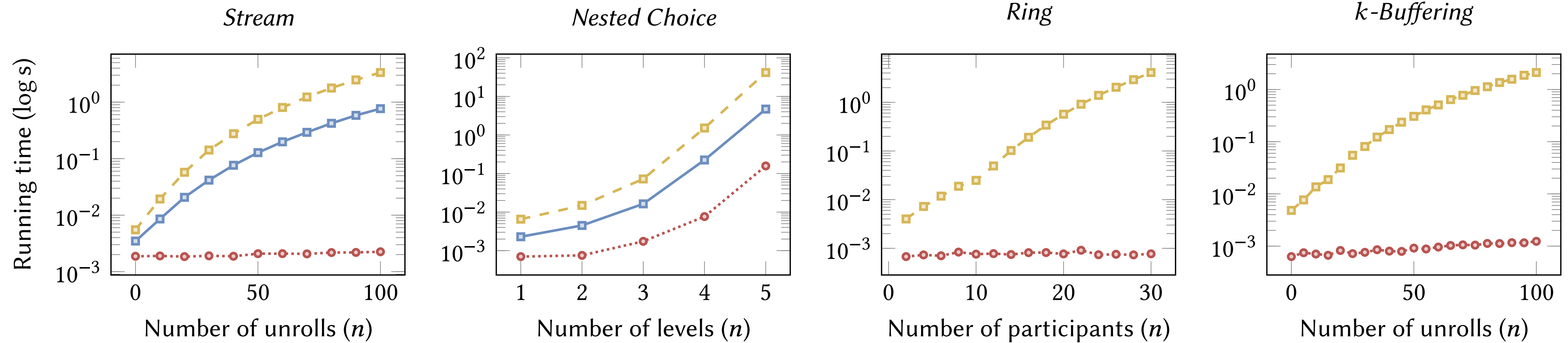
# Double DB & Butterfly Topologies for FFT



# 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 (IC), R. Neykova (Brunel), NY

# Ongoing Related Projects

<https://dsbd-appcontrol.github.io>

- **AppControl** (EP/V000462/1)
  - Ensuring specification compliance of systems-on-chip *via* CHERI capabilities and behavioural types
- **Morello-HAT** (EP/X015955/1)
  - Creation of a common API for compiler developers to leverage CHERI hardware features, improving memory security and type safety
- Other DSbD Projects...
  - **CapC**: Capability C semantics, tools and reasoning
    - Collaboration with Kent: using Kent's Rust compiler





Thank you! Questions?



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

