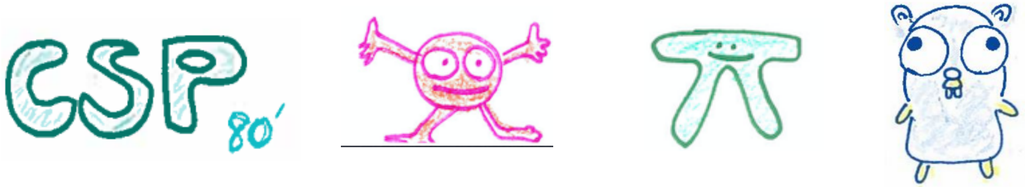
AIMS: Mini-Project Presentation

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



Nobuko Yoshida (nobuko.yoshida@cs.ox.ac.uk)

Actor Models

1944-2020

Actor Models

- **Carl Hewitt (MIT)**

1944-2020

Actor Models

1944-2020

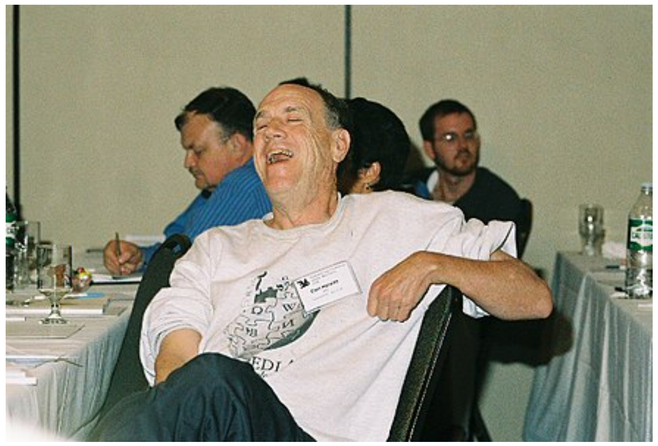
- Carl Hewitt (MIT)
  - **Mathematical Models** for Concurrent Computations

Actor Models

1944-2020

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

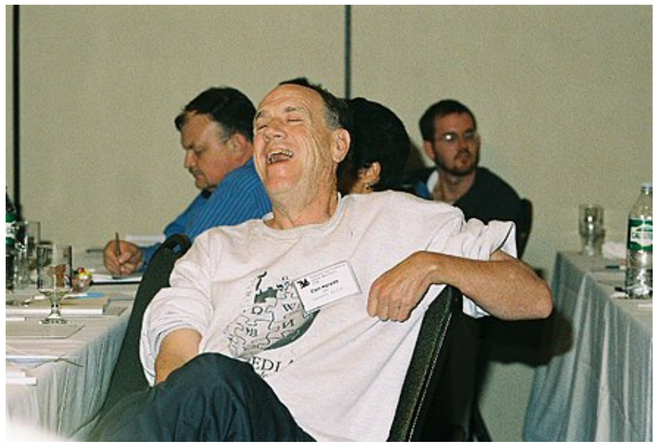
Actor Models



1944-2020

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

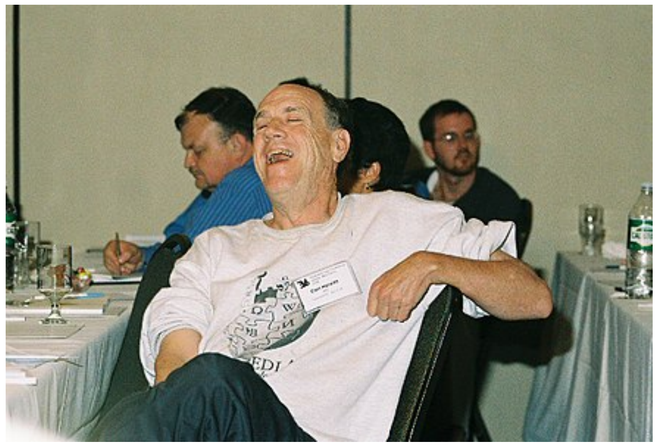
Actor Models



1944-2020

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

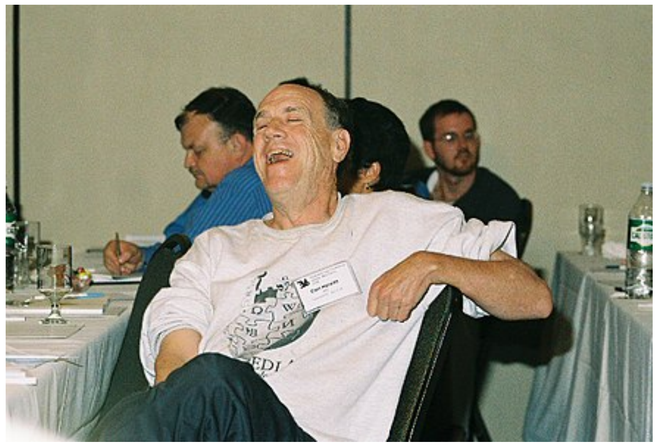
Actor Models



1944-2020

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

Actor Models



1944-2020

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

Communications are Ubiquitous

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

microservices






Problems: Ambiguity

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

RFC 821 August 1982
Simple Mail Transfer Protocol

TABLE OF CONTENTS

Problems: Ambiguity

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

3.1. MAIL

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

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

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

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

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

The second step in the procedure is the RCPT command.

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

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

Problems: Concurrency Bugs

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

deadlock

channel errors

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

The Go Gopher

Problems: Concurrency Bugs

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

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

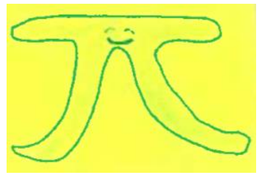
Session Types

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

Why Session Types, Why Now?

Significant academic and industry interests via fundamental breakthroughs

Binary Session Types — ESOP'98



↓

Joined W3C Standardization — 2002

↓

Multiparty Session Types — POPL' 08

↓




















OCaml

What is a concept of *Types* in Programming?

- Types can avoid runtime error

Boolean, Natural Number

100 x 200

100 x true

What is a concept of *Types* in Programming?

- Types can avoid runtime error

Boolean, Natural Number (Data Types)

100 x 200 **Correct**

100 x true **Wrong**

Types for *Protocol*?

Introduction

Rust Language

- Modern systems language focussed on safety and performance

Introduction

Rust Language

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

Introduction

Rust Language

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

Introduction

Rust Language

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

Ring Protocol

Example

Global Type

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

Ring Protocol

Example

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

Ring Protocol

Example

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

Ring Protocol

Example

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

Ring Protocol

Example

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

Ring Protocol

Example

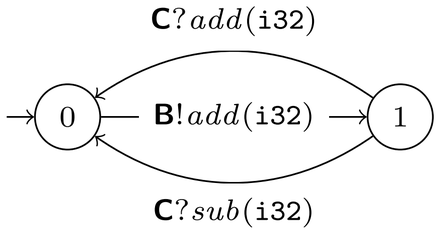
$$\mathsf{G} = \mu \mathbf{t}.\mathbf{A} \to \mathbf{B} : \left\{ add(\texttt{i32}).\mathbf{B} \to \mathbf{C} : \left\{ \begin{array}{l} add(\texttt{i32}).\mathbf{C} \to \mathbf{A} : \{add(\texttt{i32}).\mathbf{t}\} \\ sub(\texttt{i32}).\mathbf{C} \to \mathbf{A} : \{sub(\texttt{i32}).\mathbf{t}\} \end{array}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

Ring Protocol

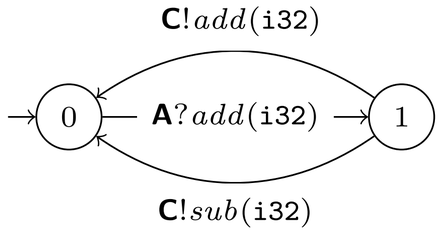
Example

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

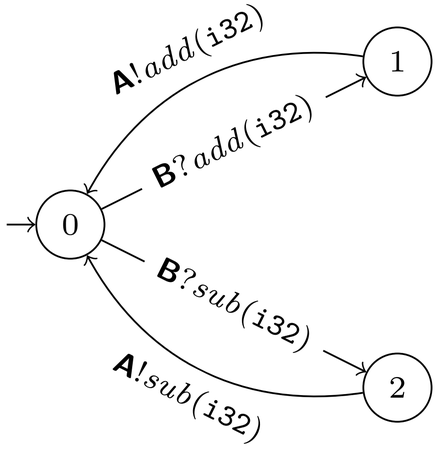
PROJECTION



PROJECTION



PROJECTION

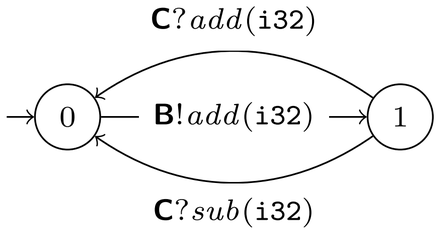


Ring Protocol

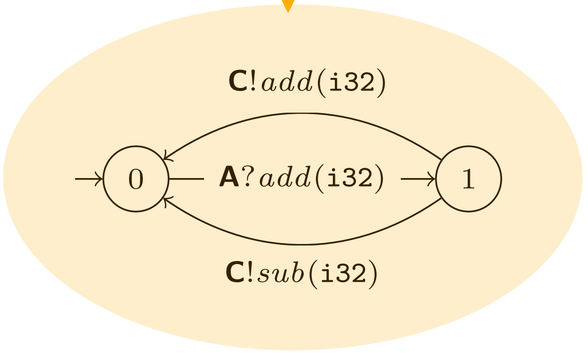
Example

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

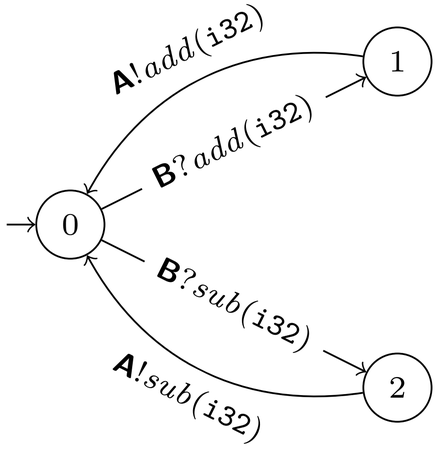
PROJECTION



PROJECTION

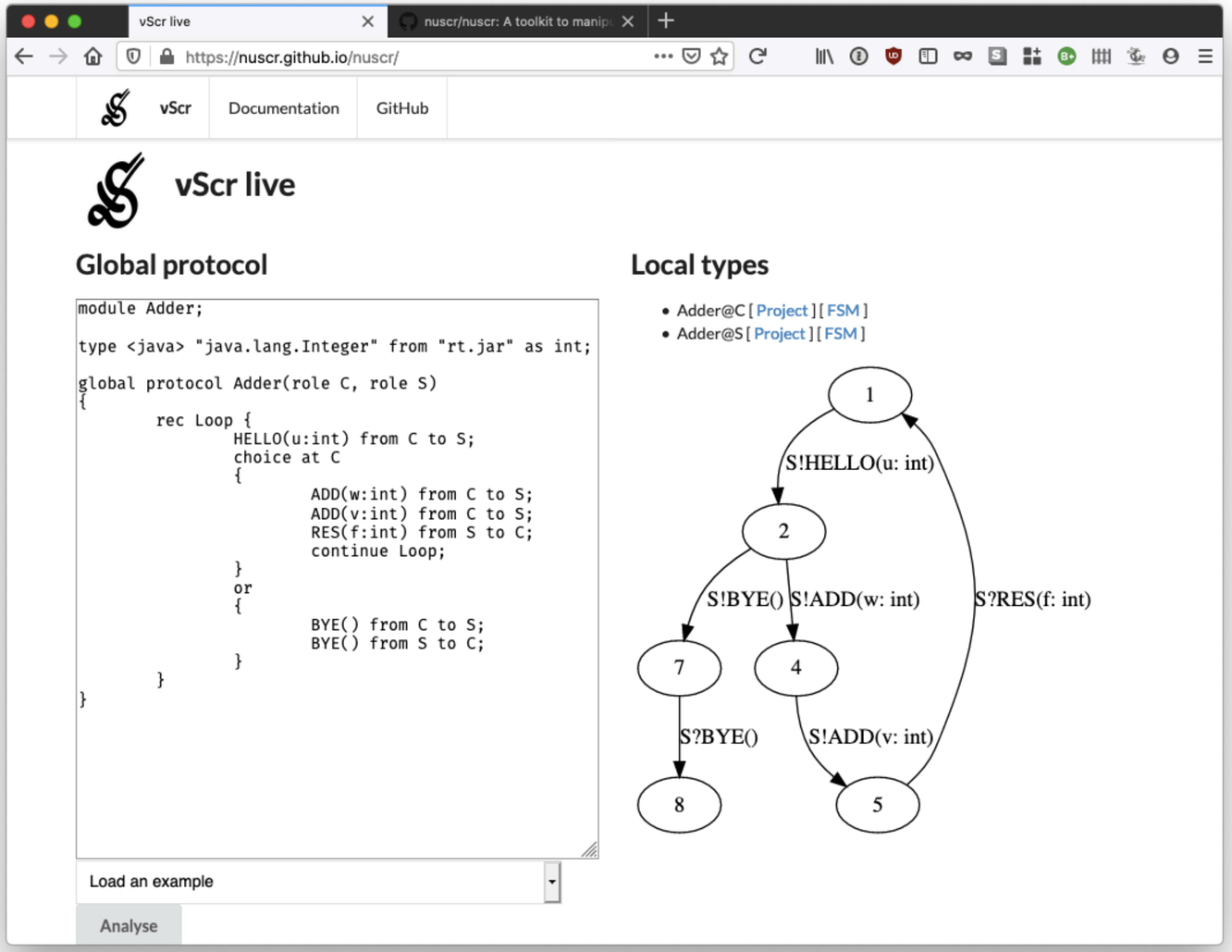


PROJECTION



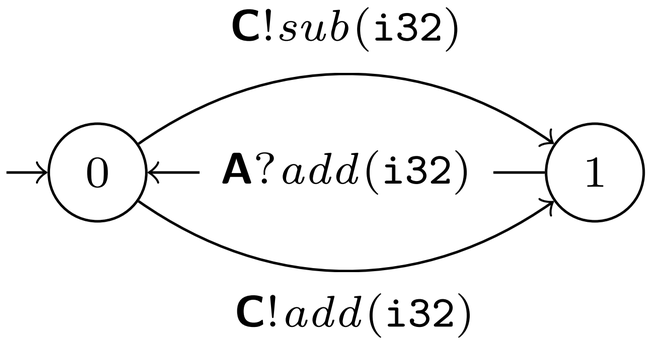
vScr An Extensible Toolchain for Multiparty Session Types

- 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



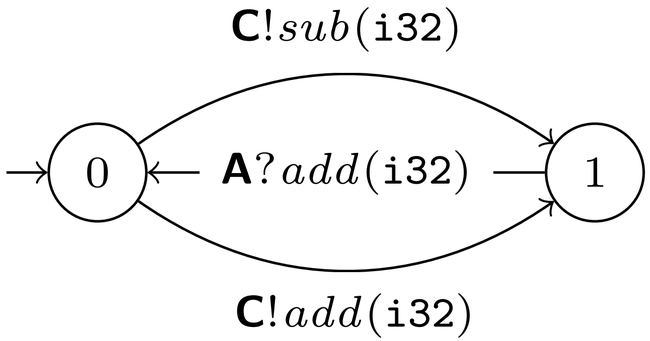
Ring Protocol

Rust API



Ring Protocol

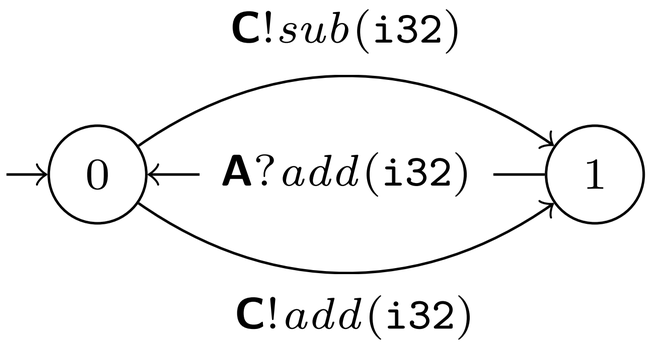
Rust API




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

Ring Protocol

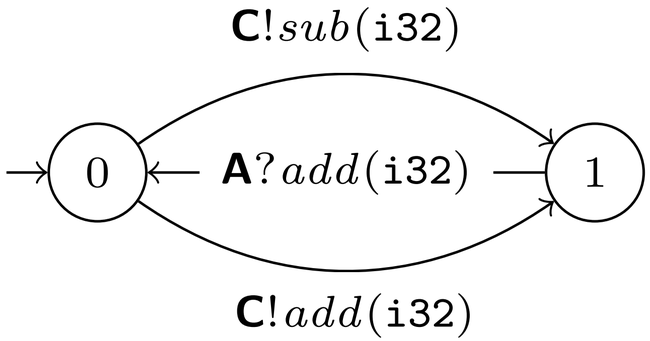
Rust API




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

Ring Protocol

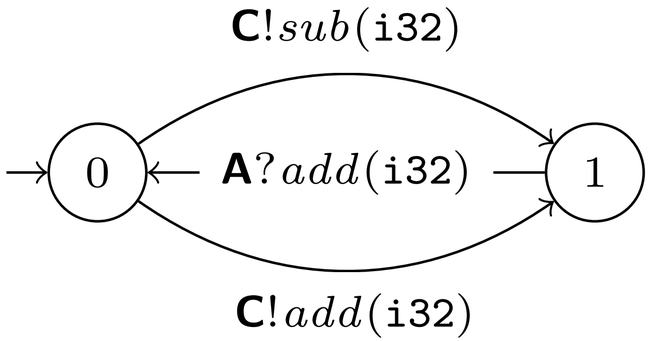
Rust API




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

Ring Protocol

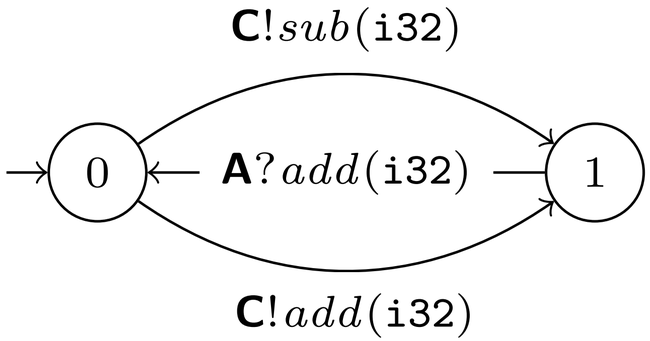
Rust API




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

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

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

Ring Protocol

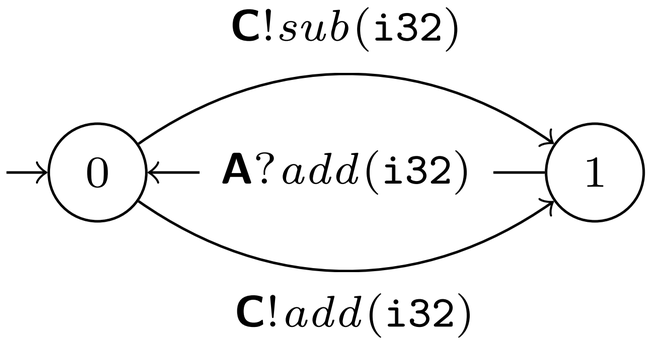
Rust API




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

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

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

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

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

Ring Protocol

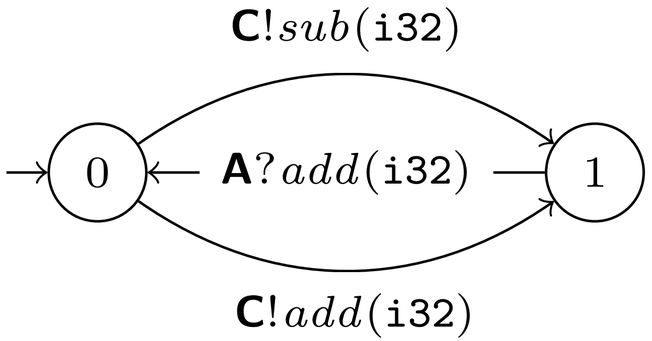
Implementation




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

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

Ring Protocol

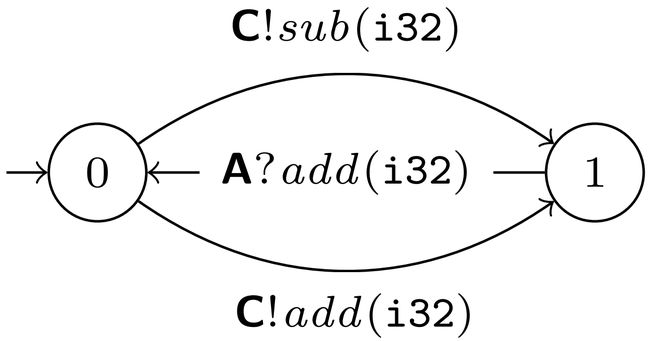
Implementation




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

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

Ring Protocol

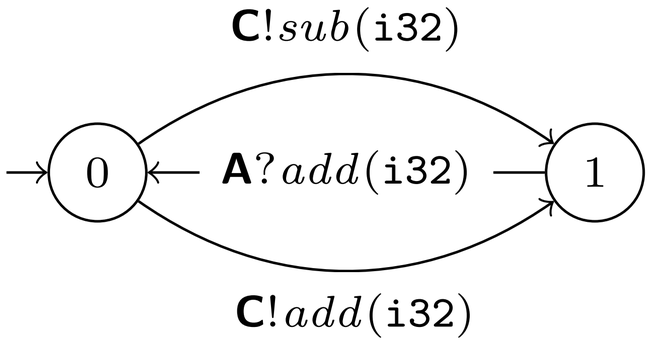
Implementation




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

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

Ring Protocol

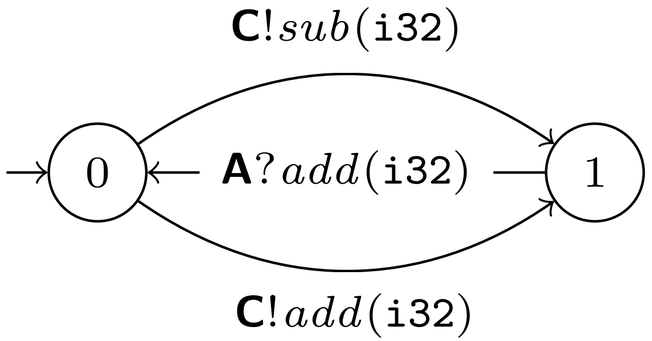
Implementation




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

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

Ring Protocol

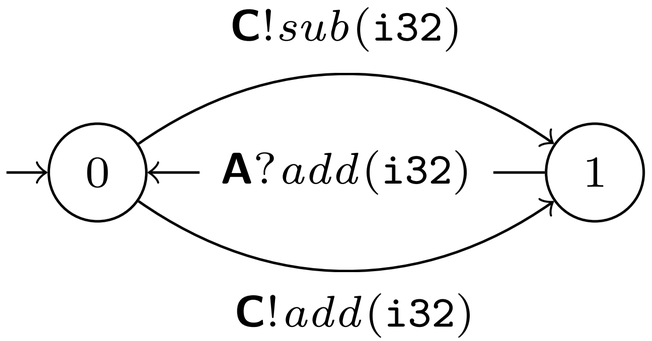
Implementation




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

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

Ring Protocol

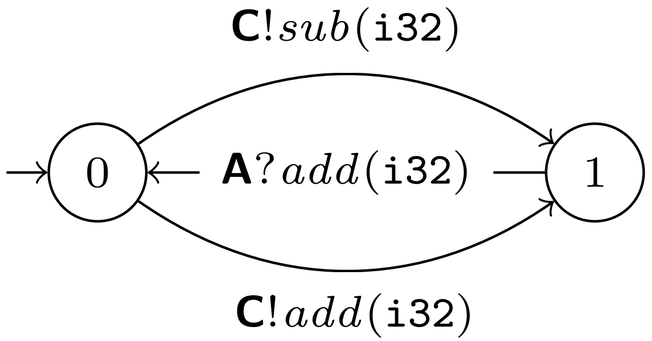
Implementation




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

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

Ring Protocol

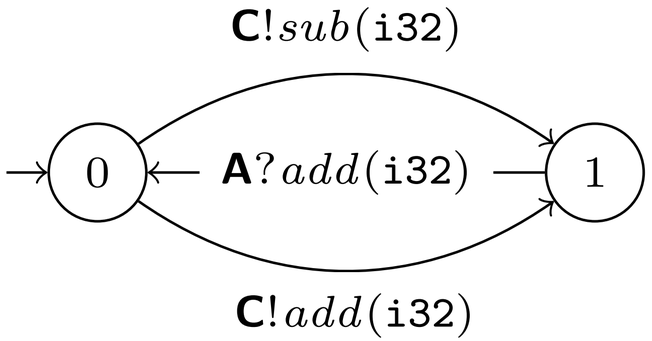
Implementation




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

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

Ring Protocol

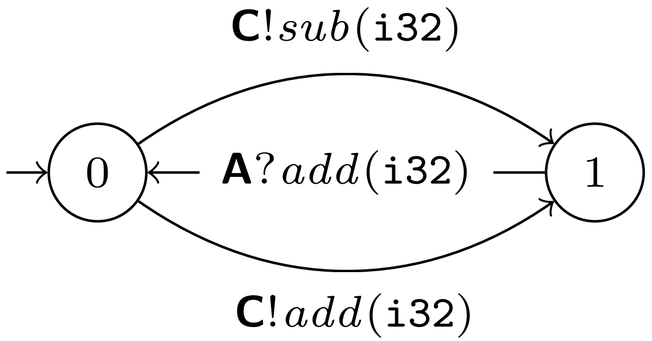
Implementation




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

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

Ring Protocol

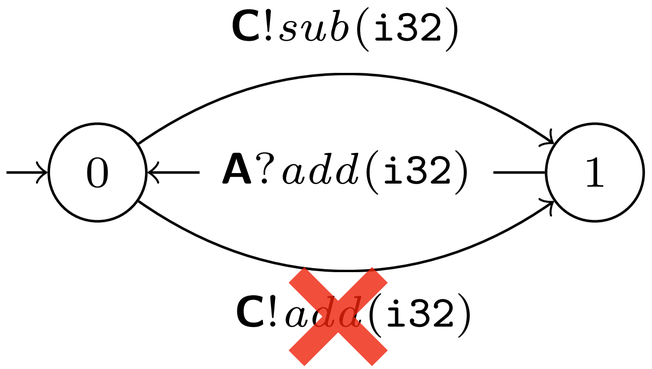
Implementation




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

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

Ring Protocol

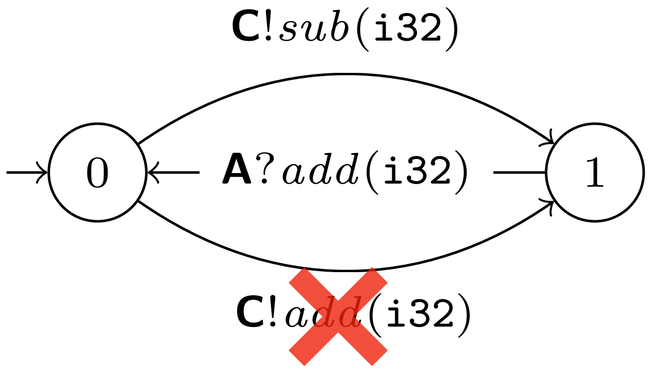
Implementation




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

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

Ring Protocol

Implementation

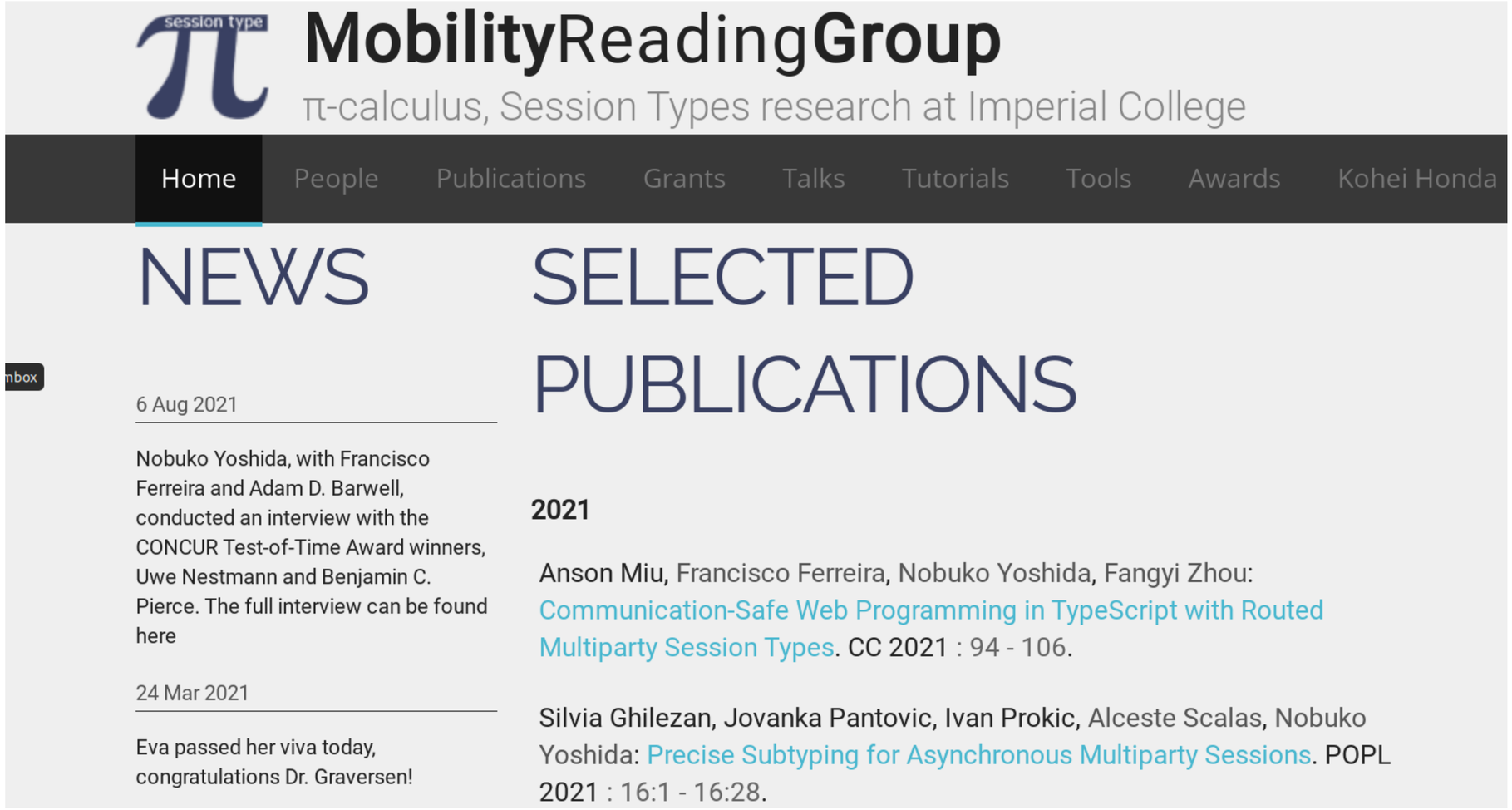



```
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

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

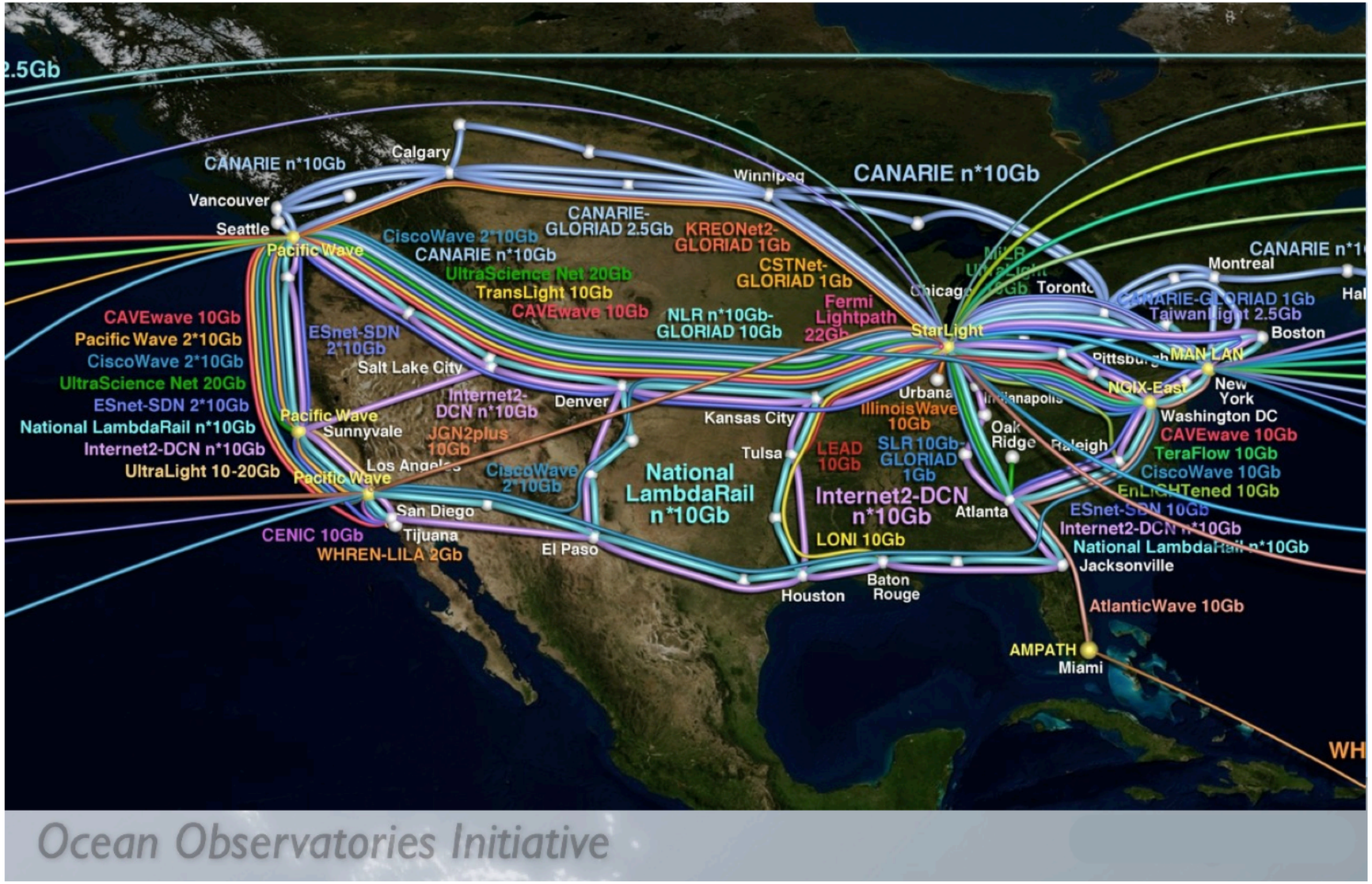
```
method not found in `rumpsteak::Select<'_, B, C, RingBChoice<'_, B>>`
```

Mobility Reading Group

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

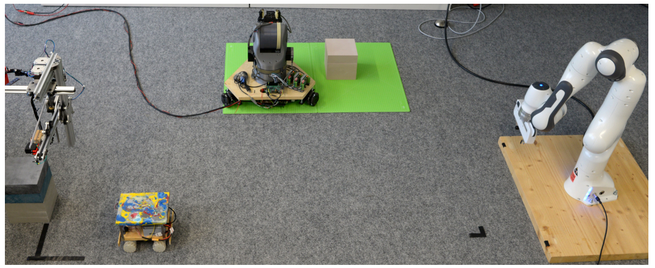


More Applications on Multiparty Session Types



OpenTelemetry

Distributed Traces



Robotics

Zooid

Mechanisation

Rust and Multiparty Session Types

- Multiparty session types and communicating automata
  - Invited paper in the FCT ’21 proceedings
  -  **https://github.com/scribble**
  -  **https://github.com/nuscr**
- Applications of multiparty session types using communicating automata
  -  **https://github.com/zakcutner/rumpsteak**

Mini-Project Proposals

- **Rust** Programming for **Autonomous Actors** or **Robotics Coordination** with AcTyx AG (https://www.actyx.com/)
- Verified Multi-Agent Programming with Actor Models (**Model Checking of Go**)
- **Theorem Provers (Coq)** to Verify Actor Models
- Efficient Machine Learning Algorithms with **Rust** or **Go**
- **Probabilistic Process Calculi**/Programming Language Theory
- Please contact with nobuko.yoshida@cs.ox.ac.uk if you wish to learn or do **programming language** or **theorem prover**'s projects