Verification

Next Generation Protocols for Heterogeneous Systems

Nobuko Yoshida, 30 January 2024, Dagstuhl Seminar 24051

UNIVERSITY OF OXFORD

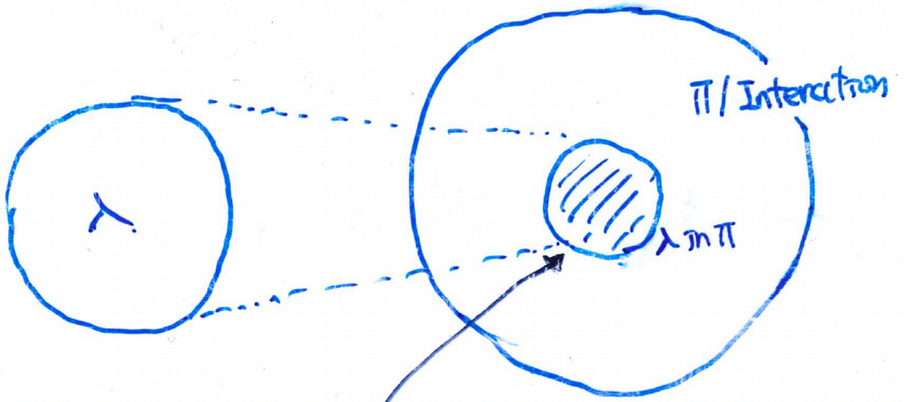
Prologue (1)

Kohei Honda [1995]

- "Functions as Processes" : [Milner 90]
  cf. [Girard 87].

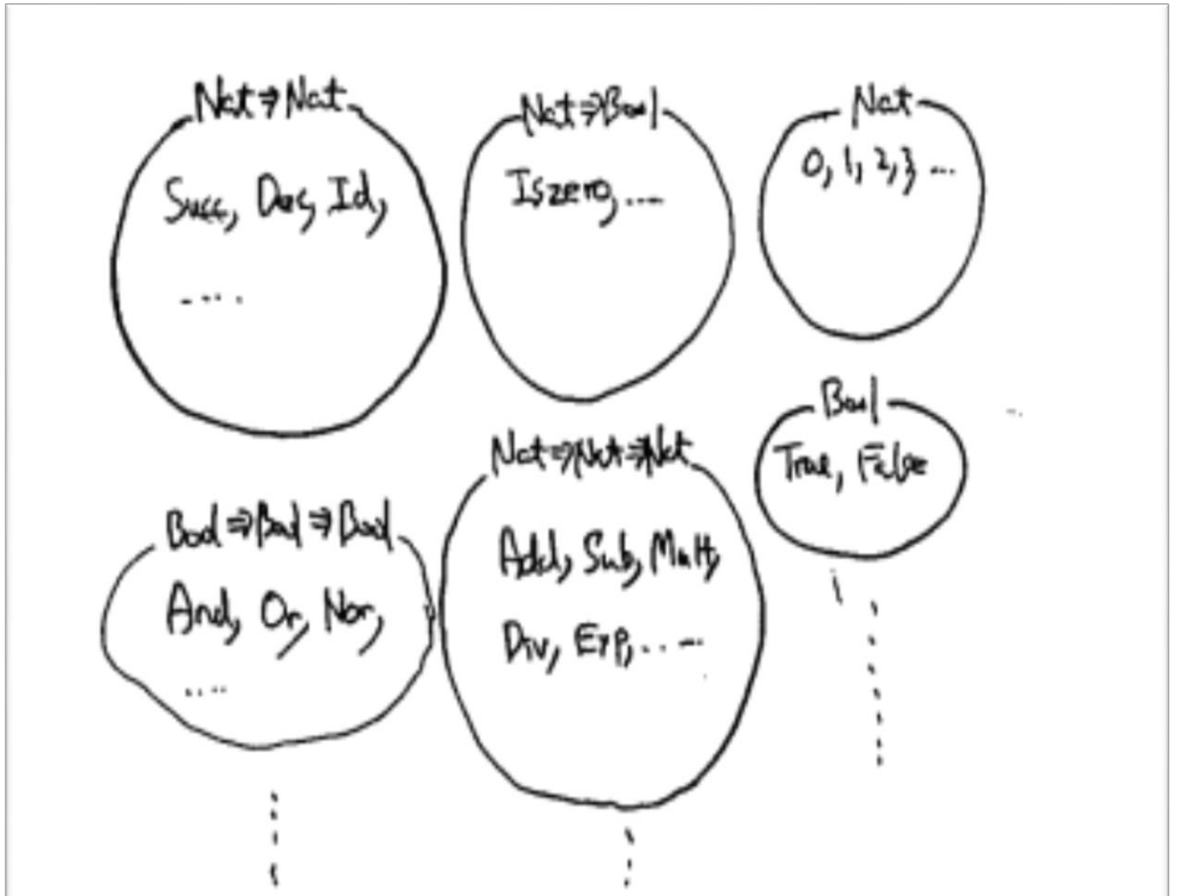
Functions

Interactions



☆ What are the class of interactive behaviours, or Rules of Interaction, of λ-agents?

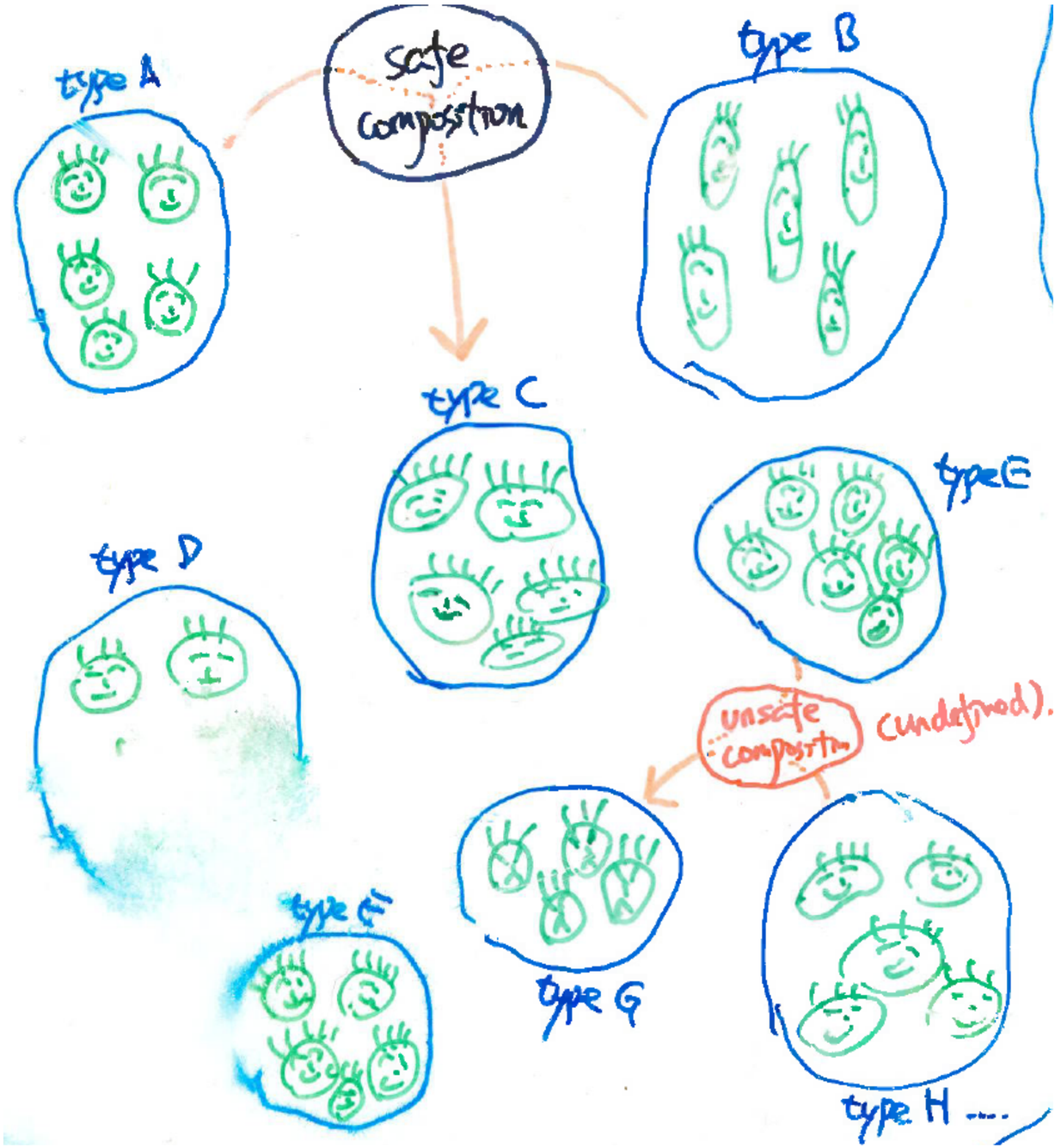
Functional Types



with operation:

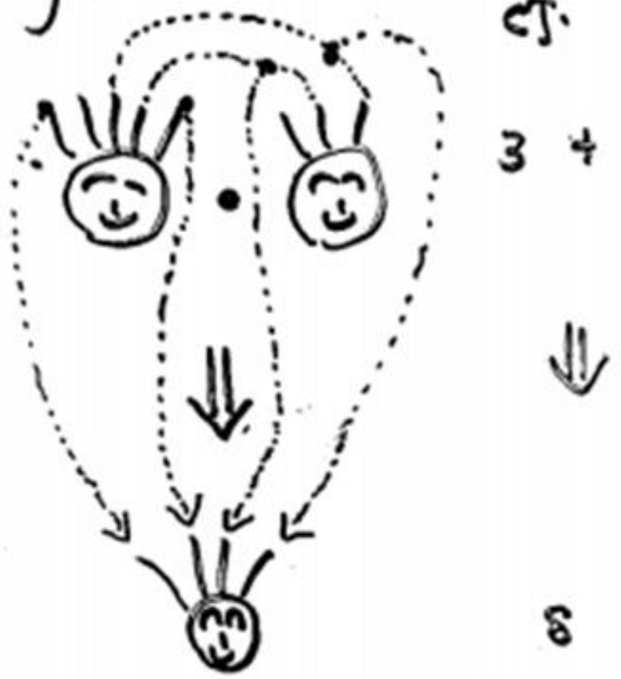
function application.

$$\begin{cases} f:\alpha\Rightarrow\beta \bullet e:\alpha = f \bullet e : \beta. \\ \text{else undefined.} \end{cases}$$

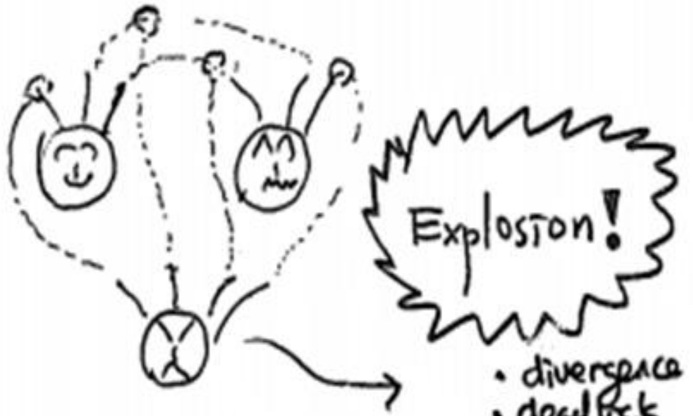

type A
safe composition
type B
type C
type D
type E
unsafe composition
(undefined).
type F
type G
type H ....

Process Types

- When it comes to processes, composition becomes:



- But some composition is dangerous!



- Therefore we type processes,

The π-calculus as a Descriptive Tool

by Kohei Honda
1995

$\lambda \quad M ::= x \mid \lambda x.M \mid MN$

$\pi \quad P ::= \Sigma \pi_i.P_i \mid P|Q \mid (\nu x)P \mid !P \mid \emptyset.$

with $\pi ::= x(\tilde{y}) \mid \bar{x}\langle \tilde{y} \rangle.$

Milner's Encoding
1991

λ in π

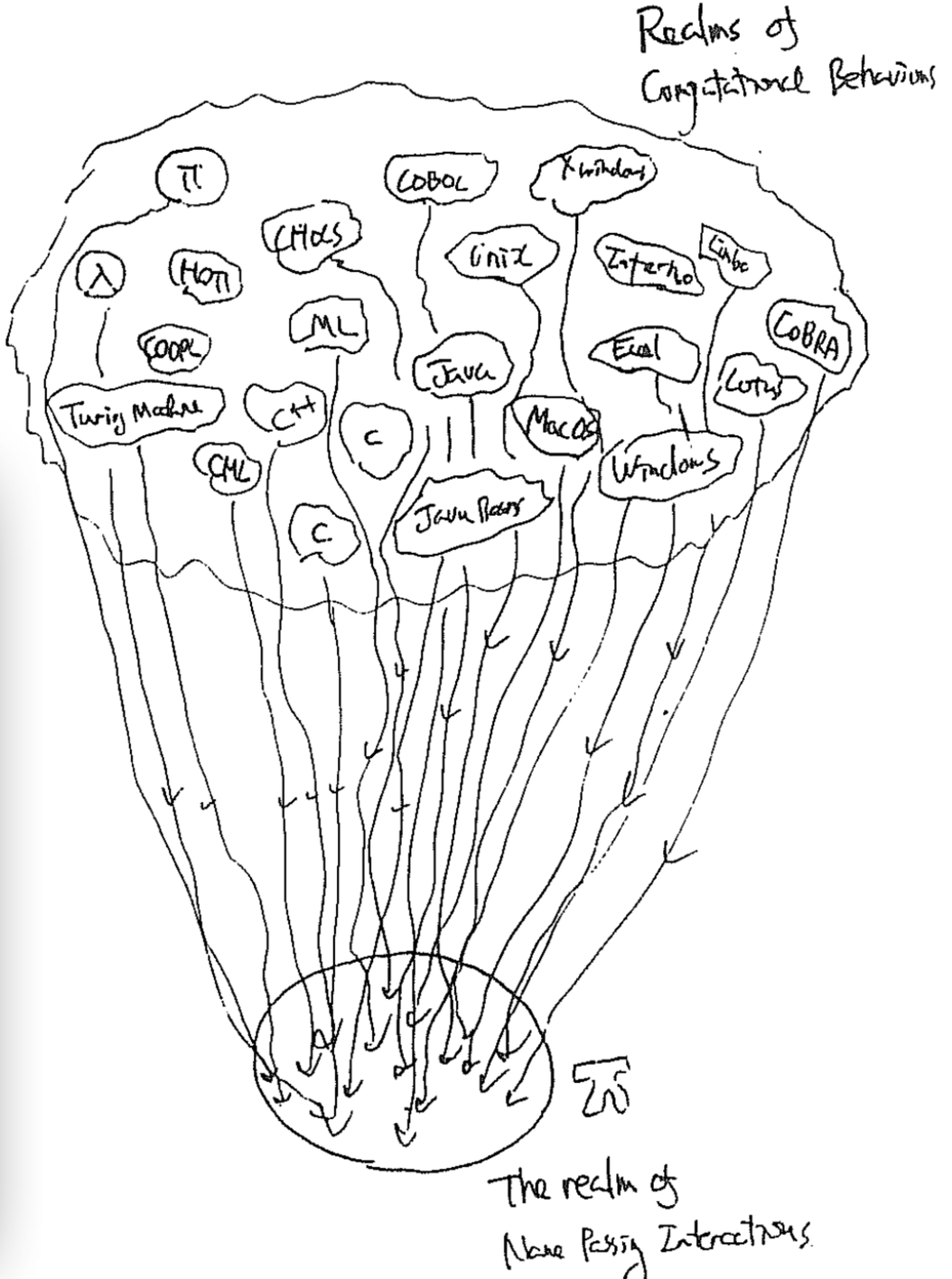
$[\![x]\!]_u \stackrel{def}{=} \bar{x}(u).$

$[\![\lambda x.M]\!]_u \stackrel{def}{=} u(xu').[\![M]\!]_{u'}$

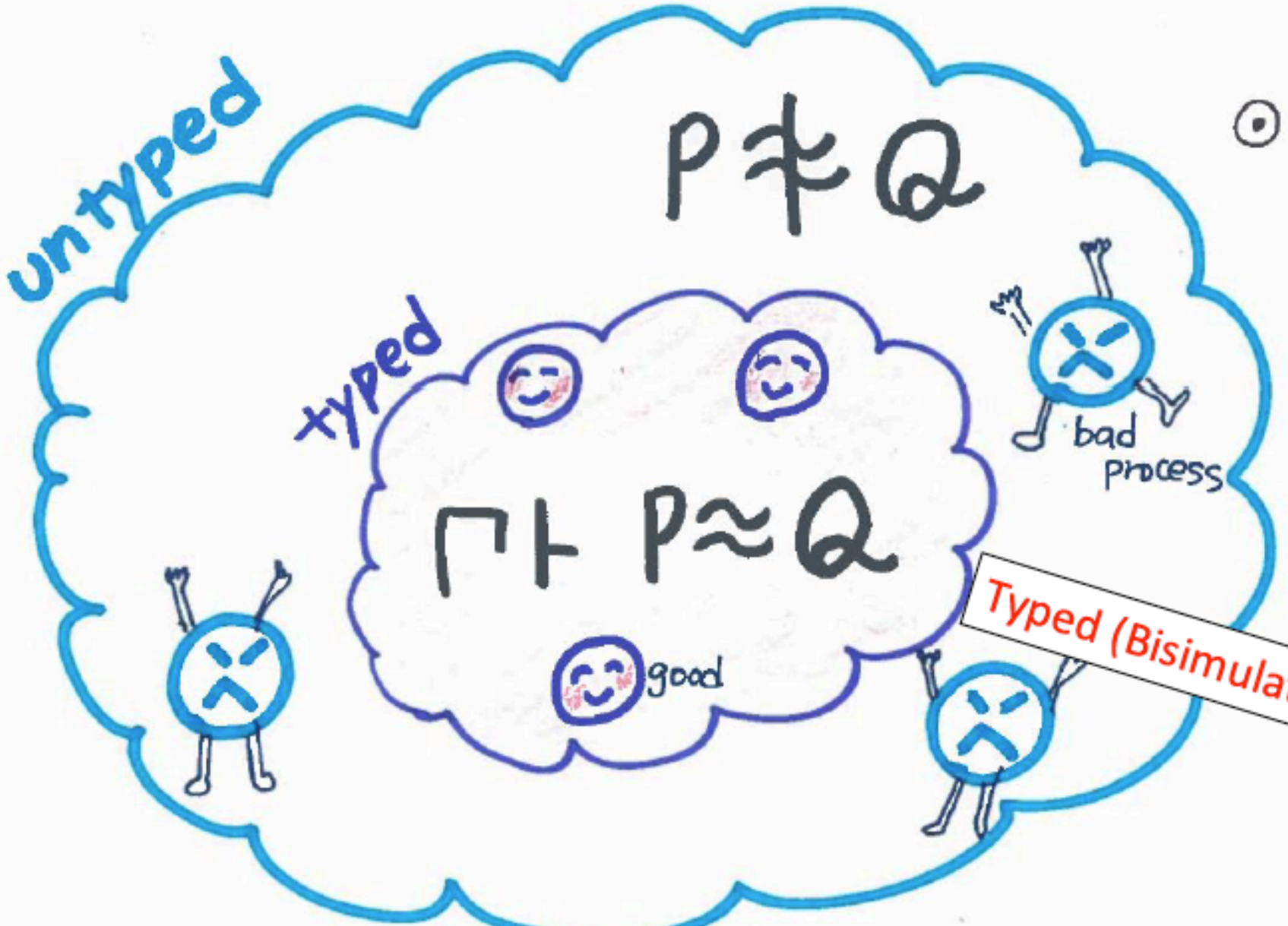
$[\![MN]\!]_u \stackrel{def}{=} (\nu f x)\left([\![M]\!]_f \mid \bar{f}\langle xu \rangle \mid [\![x=N]\!]\right)$

with $[\![x=N]\!] \stackrel{def}{=} !x(u').[\![N]\!]_{u'}.$

- λ-calculus [MPW89, Milner90, Milner92, ...]
- Concurrent Object [Walker91]
- ω-order term passing [Sangiorgi92]
- Various data structures [Milner92, ...]
- Proof Nets [Bellin and Scott93]
- Arbitrary 'constant' interaction [HYS4]
- Strategies on Games [HO95]



IO-subtyping, Linear types, Secure Information Flow, ...

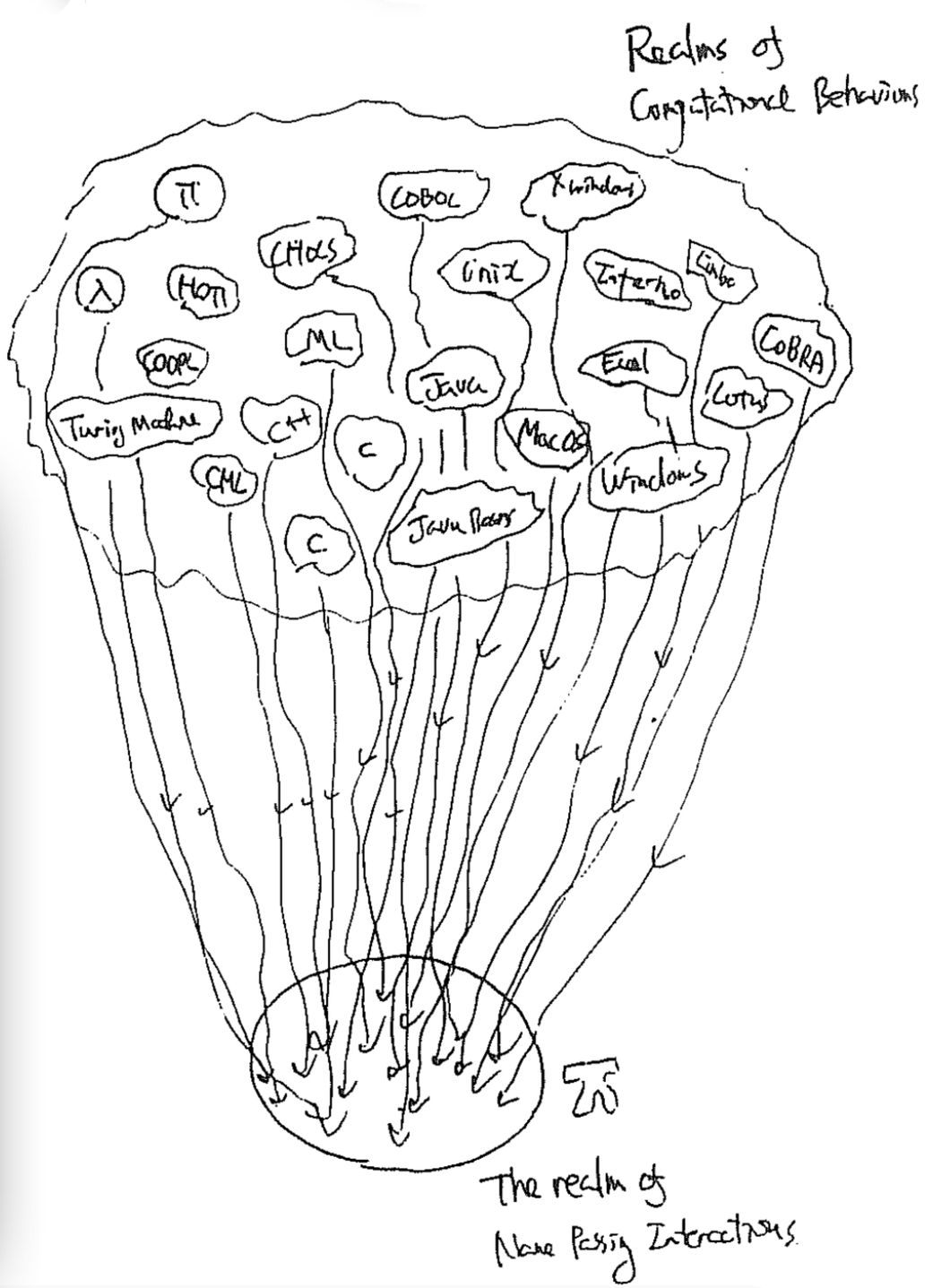


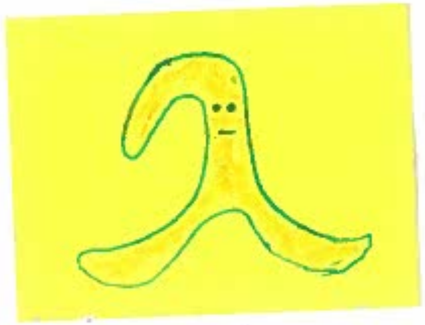
- Correctness of Encoding
- Limit environment Γ
  ⇒ Equate more processes
- Compositional

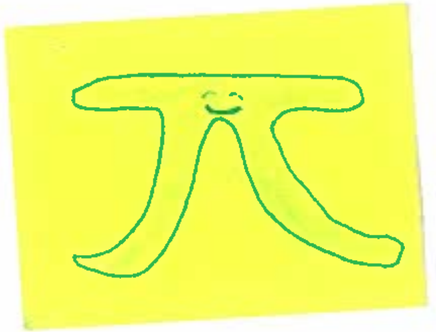
Typed (Bisimulation) Semantics

Operationally Sound but NOT Fully Abstract

- λ-calculus [MPW89, Milner90, Milner92, ...]
- Concurrent Object [Walker91]
- ω-order term passing [Sangiorgi92]
- Various data structures [Milner92, ...]
- Proof Nets [Bellin and Scott93]
- Arbitrary 'constant' interaction [HYS4]
- strategies on Games [HO95]
- ⋮





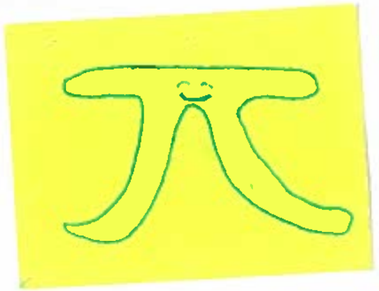


$M \simeq N$

$[\![M]\!] \simeq [\![N]\!]$

Conextual Congruence

Contextual Congruence



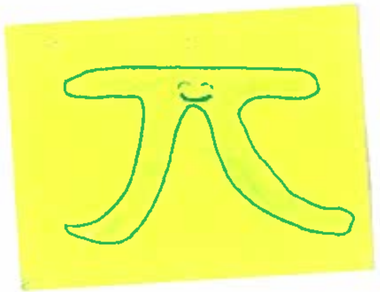
$M \simeq N$  $[\![M]\!] \simeq [\![N]\!]$

Conextual Congruence

Contextual Congruence

$C[P]\Downarrow_a$ iff $C[Q]\Downarrow_a$

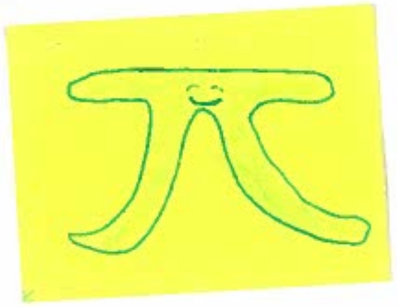
$C[[\![M]\!]]\Downarrow_a$ iff $C[[\![N]\!]]\Downarrow_a$



$M \simeq N$ NOT $[\![M]\!] \simeq [\![N]\!]$

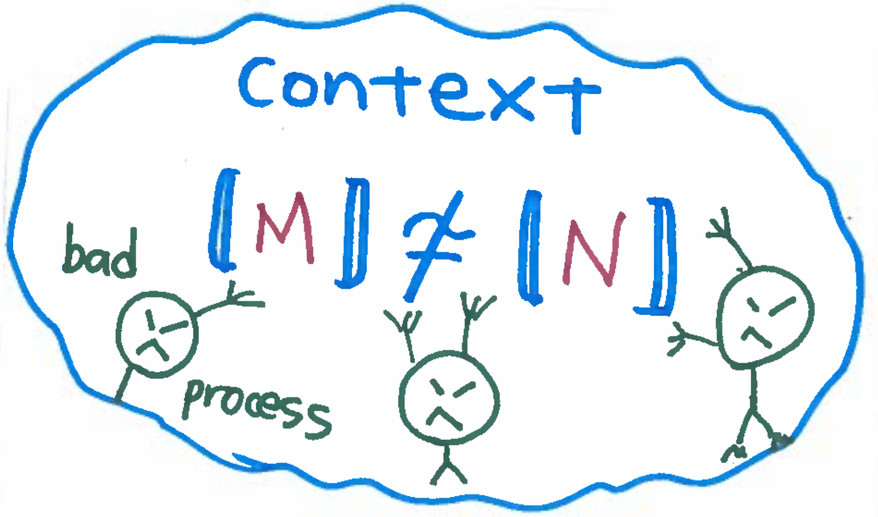
$C[P]\Downarrow_a$ iff $C[Q]\Downarrow_a$

$C[[\![M]\!]]\Downarrow_a$ iff $C[[\![N]\!]]\Downarrow_a$



$M \simeq N$ 

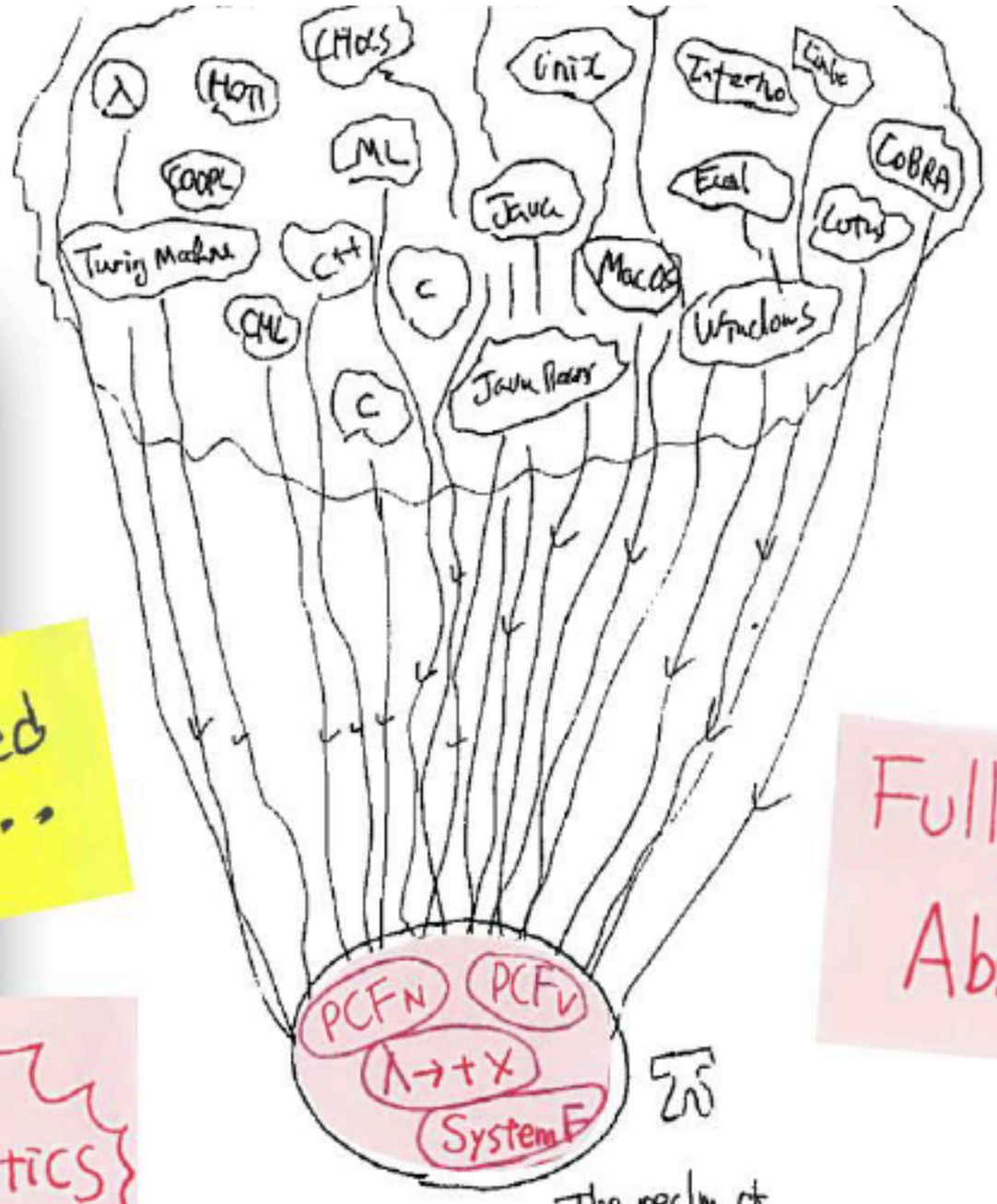
$[\![M]\!] \simeq [\![N]\!]$



- λ-calculus [MPW89, Milner90, Milner92, ...]
- Concurrent Object [Walker81]
- ω-order term passing [Sangiorgi92]
- Various data structures [Milner
- Proof Nets [Bellin and Scott 93]
- Arbitrary 'constant' interaction [
- strategies on Games [HO95]

Complicated ...

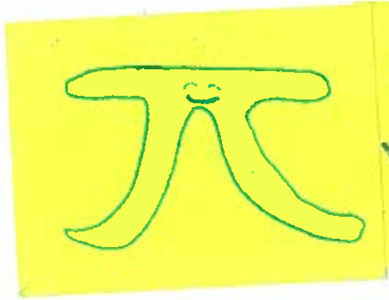
Game Semantics



The realm of Name Passing Interactions

Fully Abstract

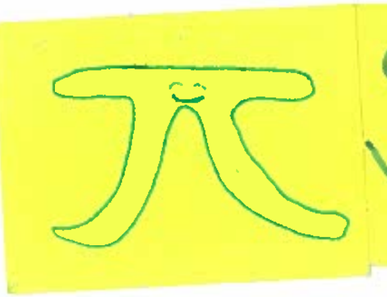
System F



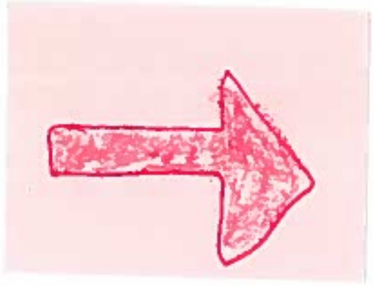
Session Π

$M \simeq N$  $[\![M]\!] \simeq [\![N]\!]$

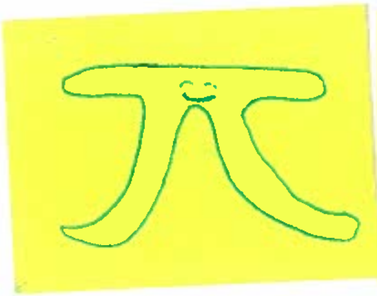
System F



Session

$M \simeq N$  $[\![M]\!] \simeq [\![N]\!]$

System F

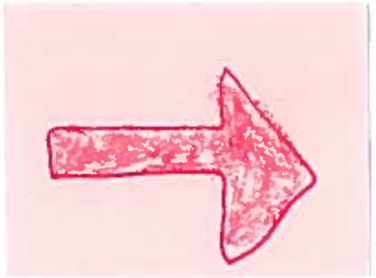


Session

$M \simeq N$

$[\![M]\!] \simeq [\![N]\!]$

$[\![P]\!] \simeq [\![Q]\!]$

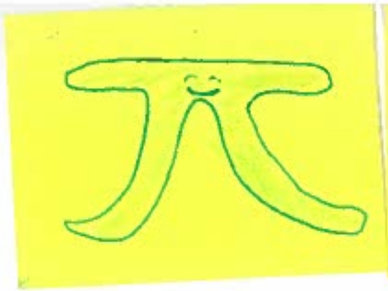


$P \simeq Q$

Reverse

New

System F



Session

$M \simeq N$

$[\![M]\!] \simeq [\![N]\!]$

$[\![P]\!] \simeq [\![Q]\!]$

$P \simeq Q$

Reverse

New

Session π

Caires, Pérez, Pfenning, Toninho 13

Types

$$A ::= A \otimes B \mid A \multimap B \mid \mathbf{1} \mid !A \mid \forall X.A \mid \exists X.A \mid X$$

Processes

$$P ::= x\langle y\rangle.P \mid x(y).P \mid \mathbf{0} \mid !x(y).P \mid x(Y).P \mid x\langle B\rangle.P \mid [x \leftrightarrow y] \mid (\nu x)P \mid (P \mid Q)$$

Judgement

$X_1, .., X_k ; a_1 : A_1, \ldots, a_n : A_n \vdash P :: b : A$

Process

Poly Vars

name

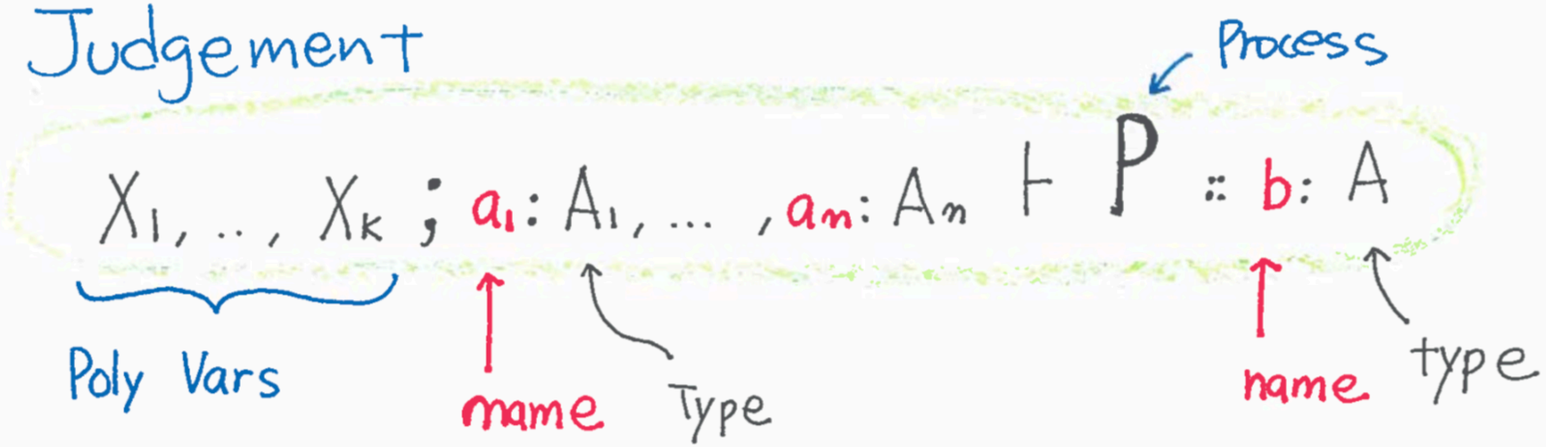
Type

name

type

process P provides A along b if composed with sessions $\vec{a} : \vec{A}$

Judgement



Cut Elimination

$$\frac{\Delta_1 \vdash P_1 :: a : A \qquad \Delta_2, a : A \vdash P_2 :: b : B}{\Delta_1, \Delta_2 \vdash (\nu a)(P_1 \mid P_2) :: b : B}$$

Identity

$$\frac{}{a : A \vdash [a \leftrightarrow b] :: b : A}$$

Linear F

Zhao, Zhang, Zdancewic 2010

Types

$$A ::= A \otimes B \mid A \multimap B \mid !A \mid \mathbf{1} \mid \mathbf{2}$$
$$\mid \forall X.A \mid \exists X.A \mid X$$

Terms

$$M, N ::= \lambda x.M \mid MN \mid \langle M \otimes N \rangle \mid \text{let } x \otimes y = M \text{ in } N$$
$$\mid !M \mid \text{let } !u = M \text{ in } N$$
$$\mid \Lambda X.M \mid M[A] \mid \text{pack } A \text{ with } M \mid \text{let } (X, y) = M \text{ in } N$$
$$\mid \text{let } \mathbf{1} = M \text{ in } N \mid \langle\rangle \mid \mathbf{T} \mid \mathbf{F}$$

λ in π

Milner's Encoding 1991

$$[\![x]\!]_u \stackrel{\text{def}}{=} \overline{x}\langle u \rangle.$$

$$[\![\lambda x.M]\!]_u \stackrel{\text{def}}{=} u(x u'). [\![M]\!]_{u'}$$

$$[\![MN]\!]_u \stackrel{\text{def}}{=} (\nu f x)\left([\![M]\!]_f \mid \overline{f}\langle x u \rangle \mid [\![x=N]\!] \right)$$

with $[\![x=N]\!] \stackrel{\text{def}}{=} !x(u'). [\![N]\!]_{u'}.$

$[\![M]\!]_a$

↑ name

Session Π

λ in π

$[\![x]\!]_u \stackrel{def}{=} \overline{x}(u).$

$[\![\lambda x.M]\!]_u \stackrel{def}{=} u(x u'). [\![M]\!]_{u'}.$

$[\![MN]\!]_u \stackrel{def}{=} (\nu f x)\left([\![M]\!]_f \mid !f(x u) \mid [\![x{=}N]\!]\right)$

with $[\![x{=}N]\!] \stackrel{def}{=} !x(u'). [\![N]\!]_{u'}.$

$[\![x]\!]_a = [x \leftrightarrow a]$

$[\![\langle\rangle]\!]_a = 0$

$[\![\lambda x.M]\!]_a = a(x). [\![M]\!]_a$

$[\![MN]\!]_a = [\![M]\!]x \mid \overline{x}(y). [\![N]\!]_y \mid [x \leftrightarrow a]$

From  to 

From Sequent Calculus to Natural Deduction

$$[\![P]\!]_{\Delta \vdash a : A} \quad \text{with} \quad \Delta \vdash P :: a : A$$

$$[\![0]\!] = \langle\rangle$$

$$[\![[x \leftrightarrow a]]\!] = x$$

$$[\![a(x).P]\!] = \lambda x.[\![P]\!]$$

$$[\![a(X).P]\!] = \Lambda X.[\![P]\!]$$

Parallel

$$\left[\frac{\Delta_1 \vdash P :: a:A \qquad \Delta_2, a:A \vdash Q :: b:C}{\Delta_1, \Delta_2 \vdash (\nu a)(P \mid Q) :: b:C}\right]$$

Substitute P into a in Q

$$= \frac{\Delta_2, a:A \vdash [\![Q]\!]_b :: C \qquad \Delta_1 \vdash [\![P]\!]_a : A}{\Delta_1, \Delta_2 \vdash [\![Q]\!]\{[\![P]\!]/a\} : C}$$

Theorems

Operational Correspondence / Type Preserving

Definability

Inverse

$$(\!|\, [\![M]\!]_z \,|\!) \cong M$$

$$[\![(\!|\, P \,|\!)]\!]_z \cong P$$

$$\forall P \; \exists M \quad [\![M]\!]_z \cong P$$

$$\forall M \; \exists P \quad (\!|\, P \,|\!) \cong M$$

Full Abstraction

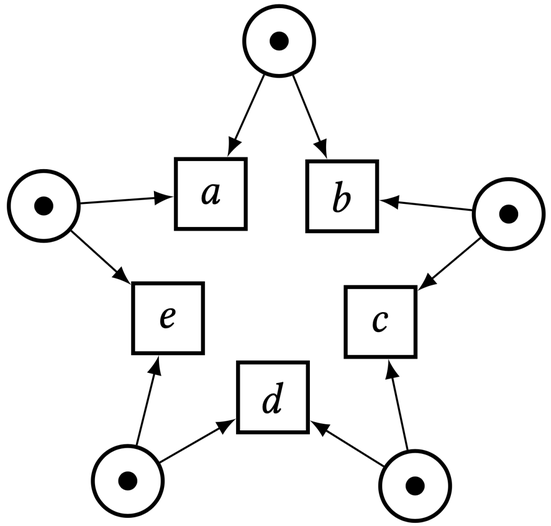
$$[\![M]\!]_z \cong [\![N]\!]_z \quad \text{iff} \quad M \cong N$$

$$(\!|\, P \,|\!) \cong (\!|\, Q \,|\!) \quad \text{iff} \quad P \cong Q$$

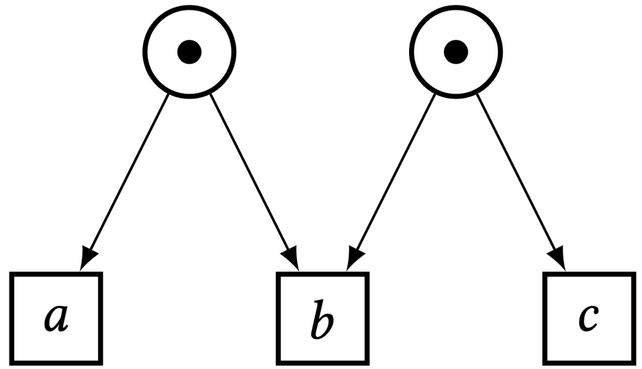
Petri nets and interconnectability

M-structures and Star

- Petri net structures are used to identify ***synchronously fully distributed*** and ***asynchronously fully distributed*** process calculi



Star

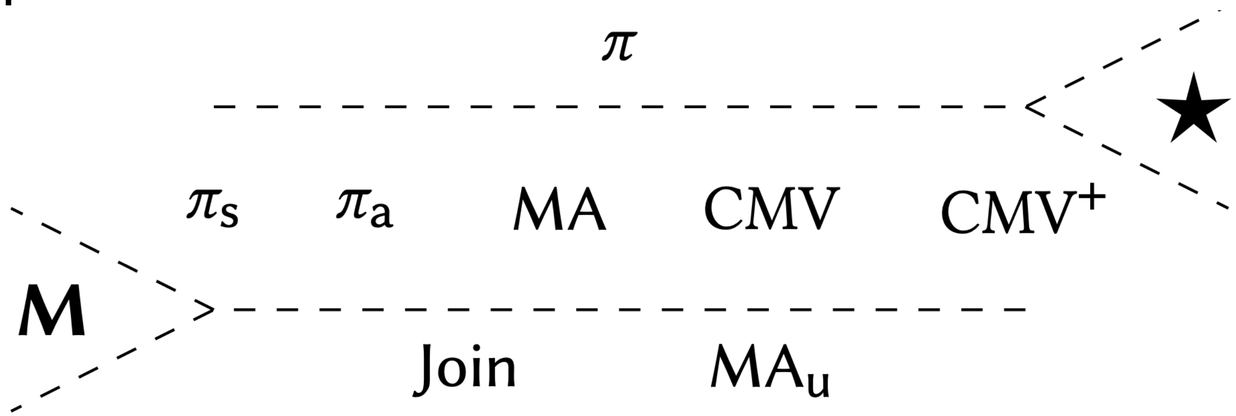


M

Petri nets and interconnectability

M-structures and Star

- ***synchronously fully distributed*** and ***asynchronously fully distributed*** process calculi

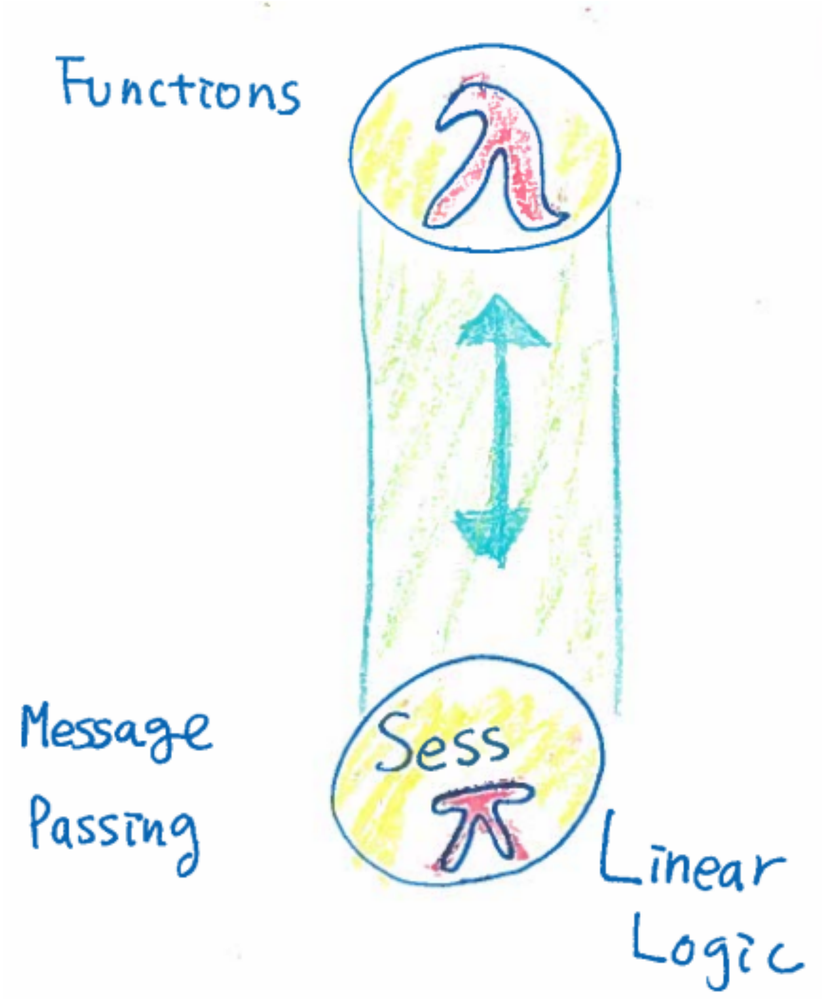


Linear Logic-based Pi

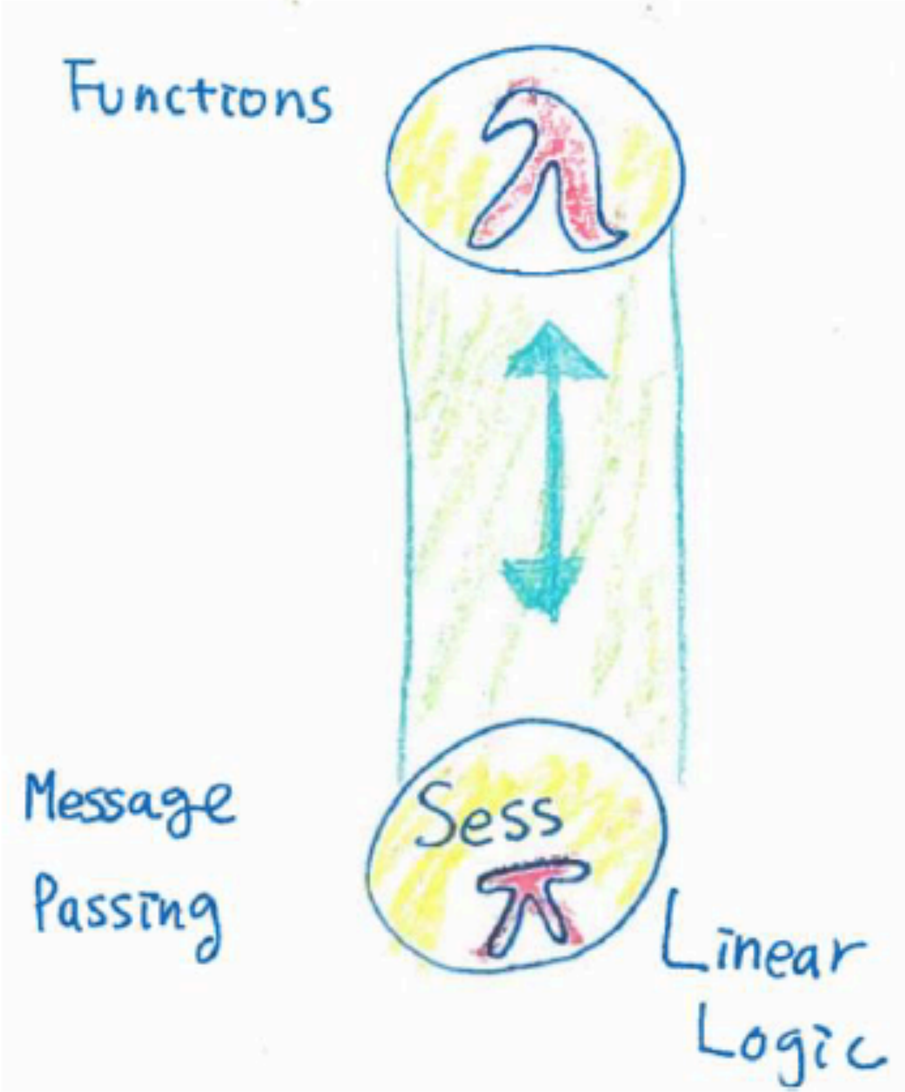
CMV : Fundamental Session Pi-Calculus (Branching and Selection) [Vasconcelos 2012]

CMV+ : Mixed Sessions [Casal, Mordito and Vasconcelos 2022]

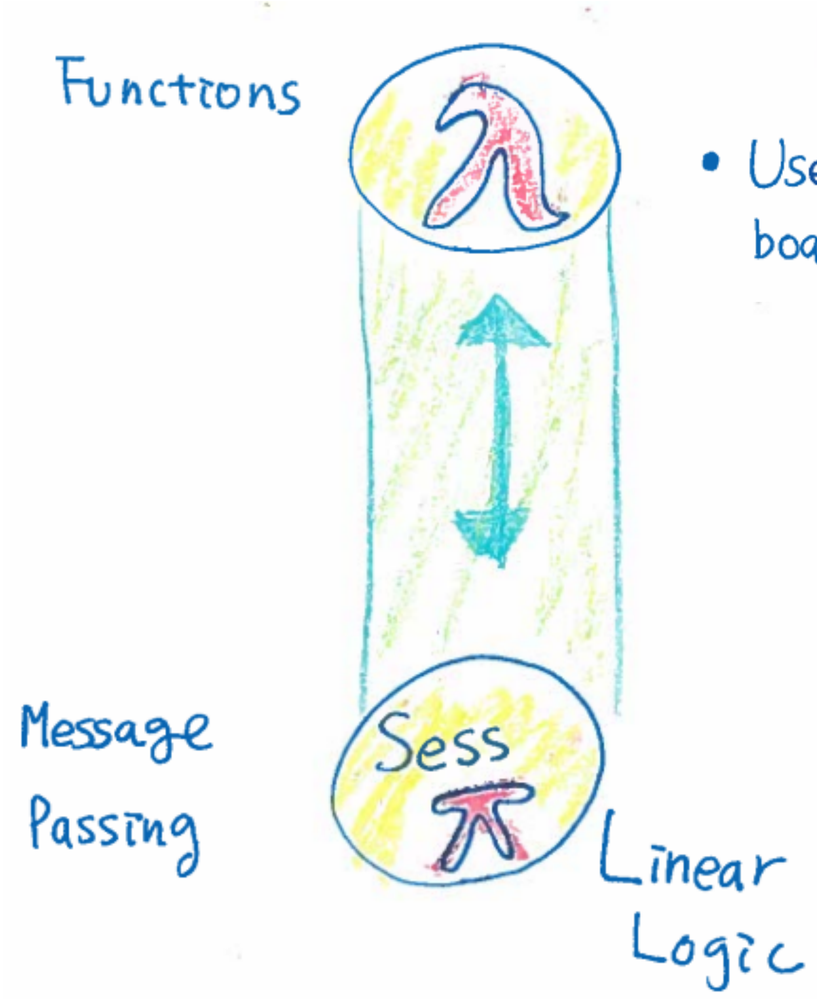
Summary



Summary

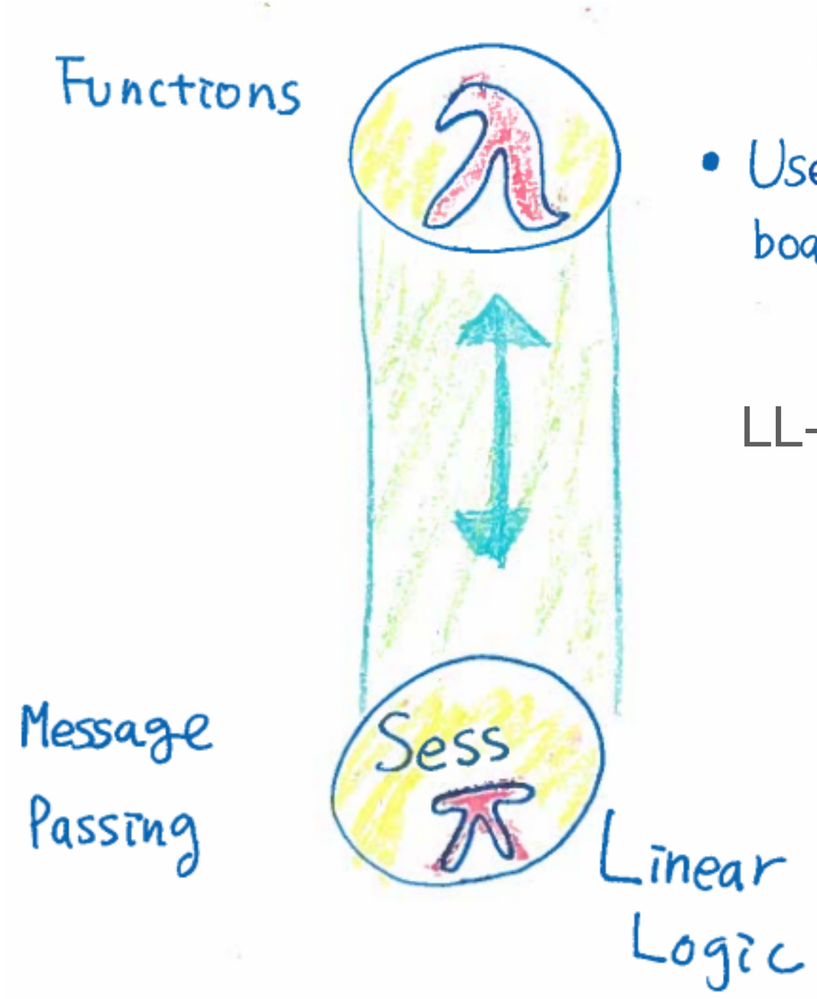


Summary



- LL-based Use Sess π to articulate boarder computations

Summary



LL-based

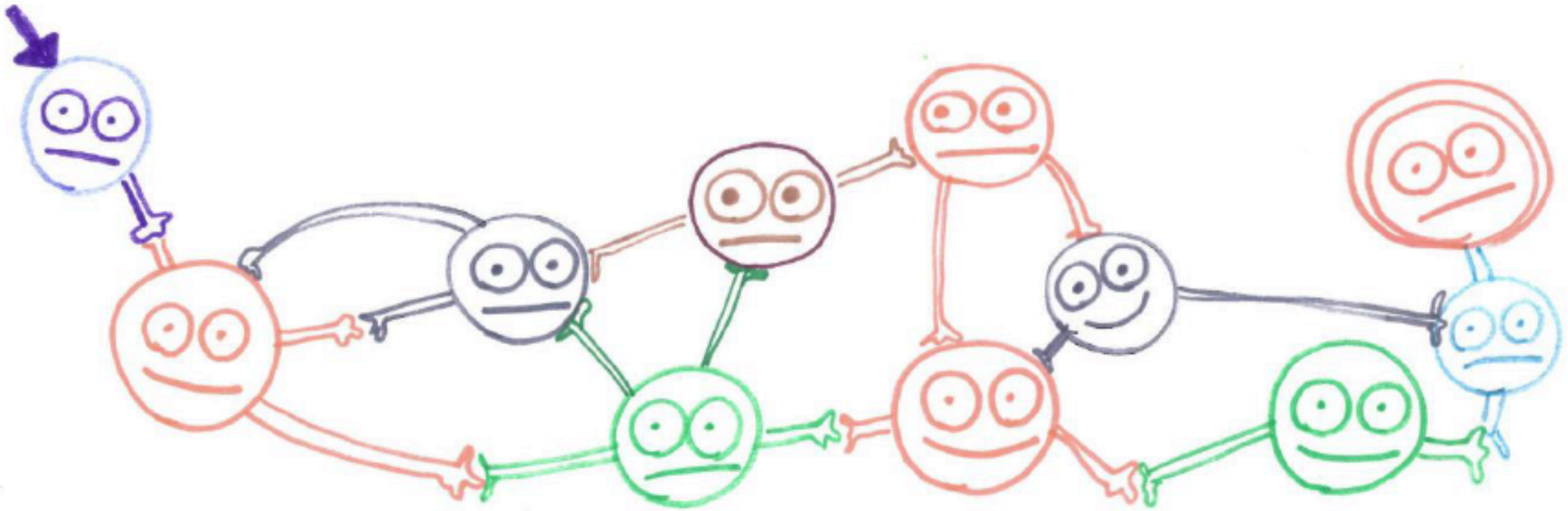
- Use Sess π to articulate boarder computations

LL-Based Session Types (Too Many to List)

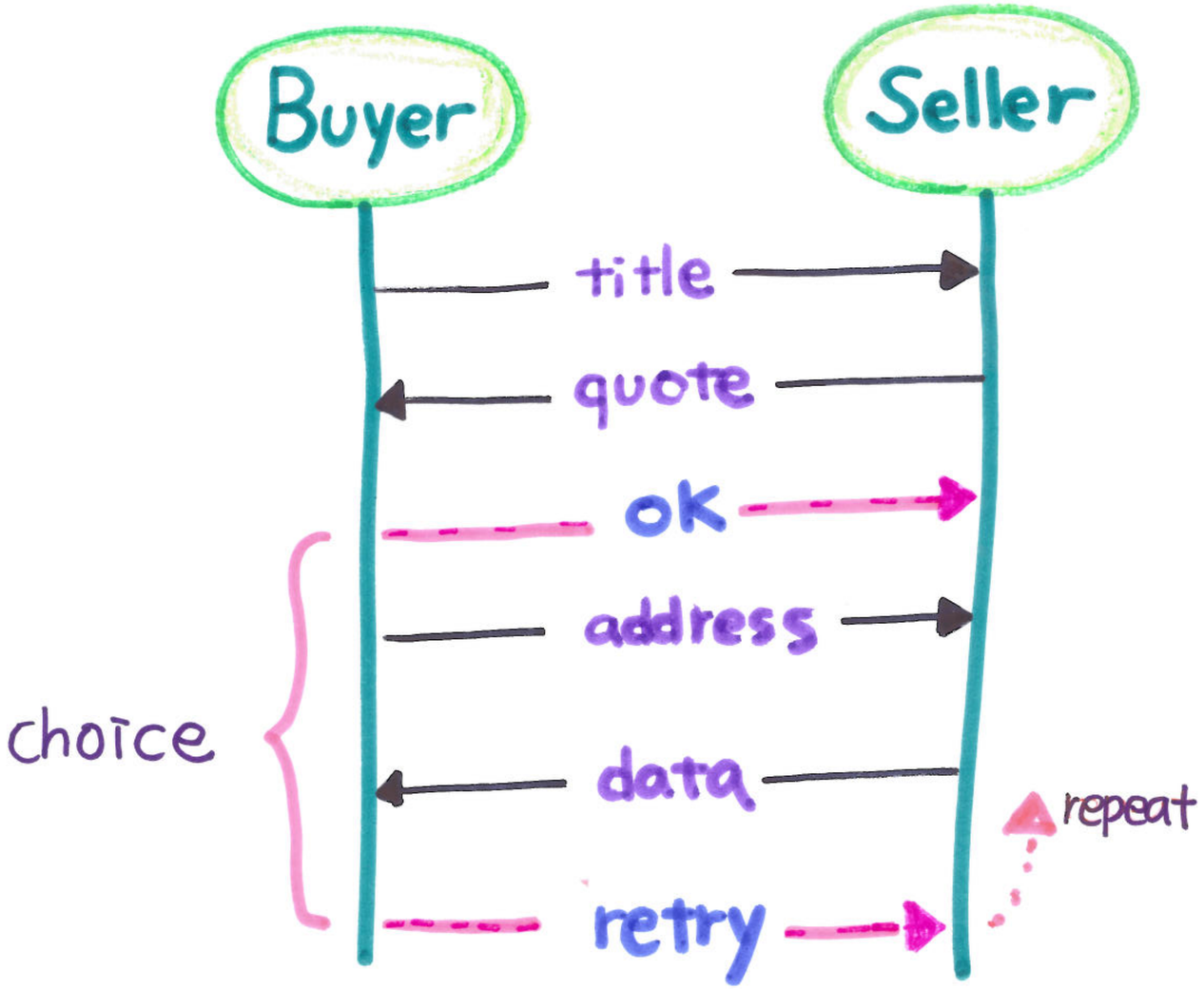
- Balzer and Pfenning, Manifest sharing with session types, ICFP'17
- Rocha and Caires, Propositions-as-types and shared state, ICFP'21
- Zien et al, Client-server sessions in linear logic, ICFP'21
- Frumin et al, A propositions-as-sessions interpretation of bunched implications in channel-based concurrency, OOPSLA'22
- Jacobs et al., Multiparty GV: Functional Multiparty Session Types With Certified Deadlock Freedom

Global Protocols

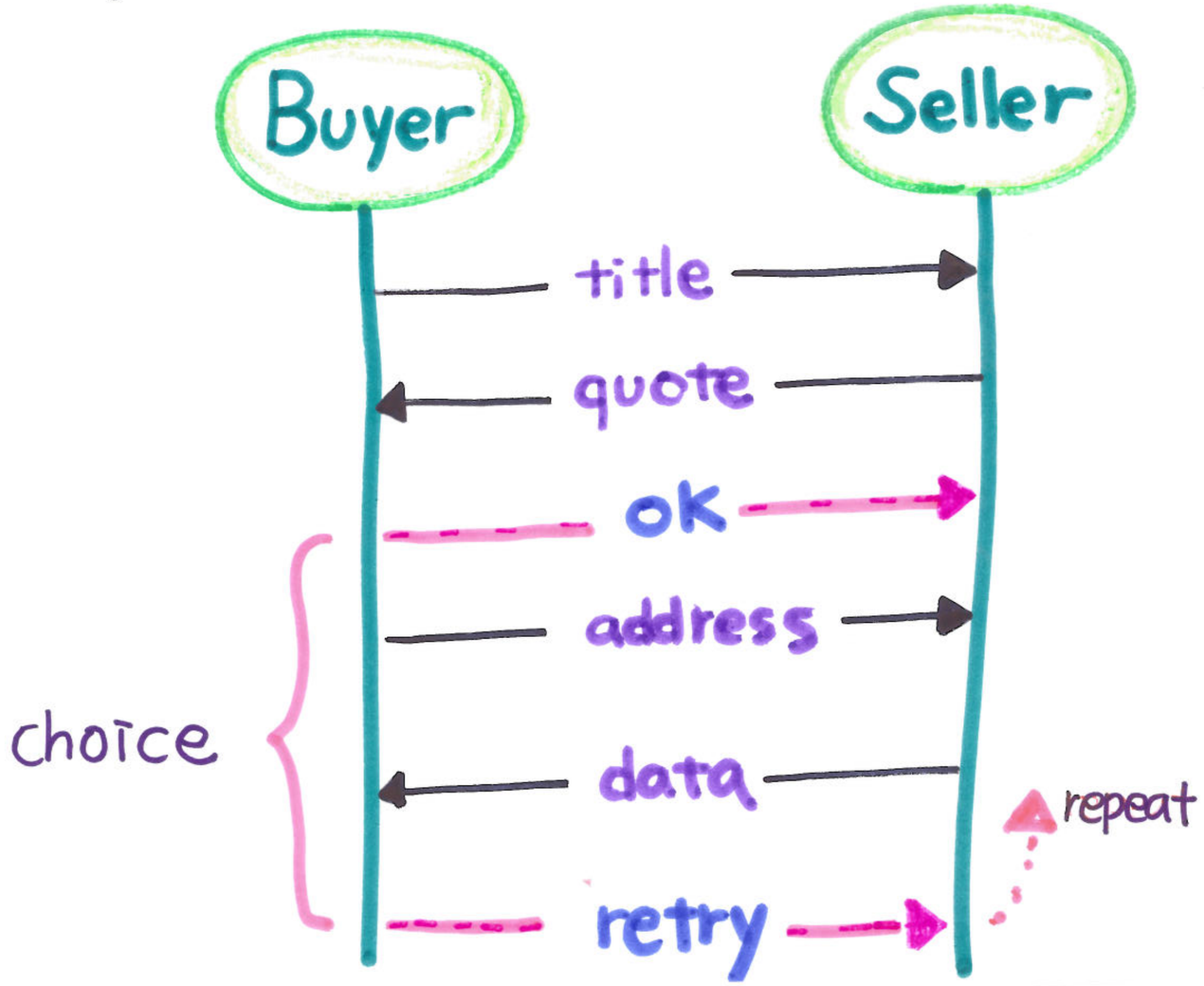
Linkage between Session Types and Communicating Automata



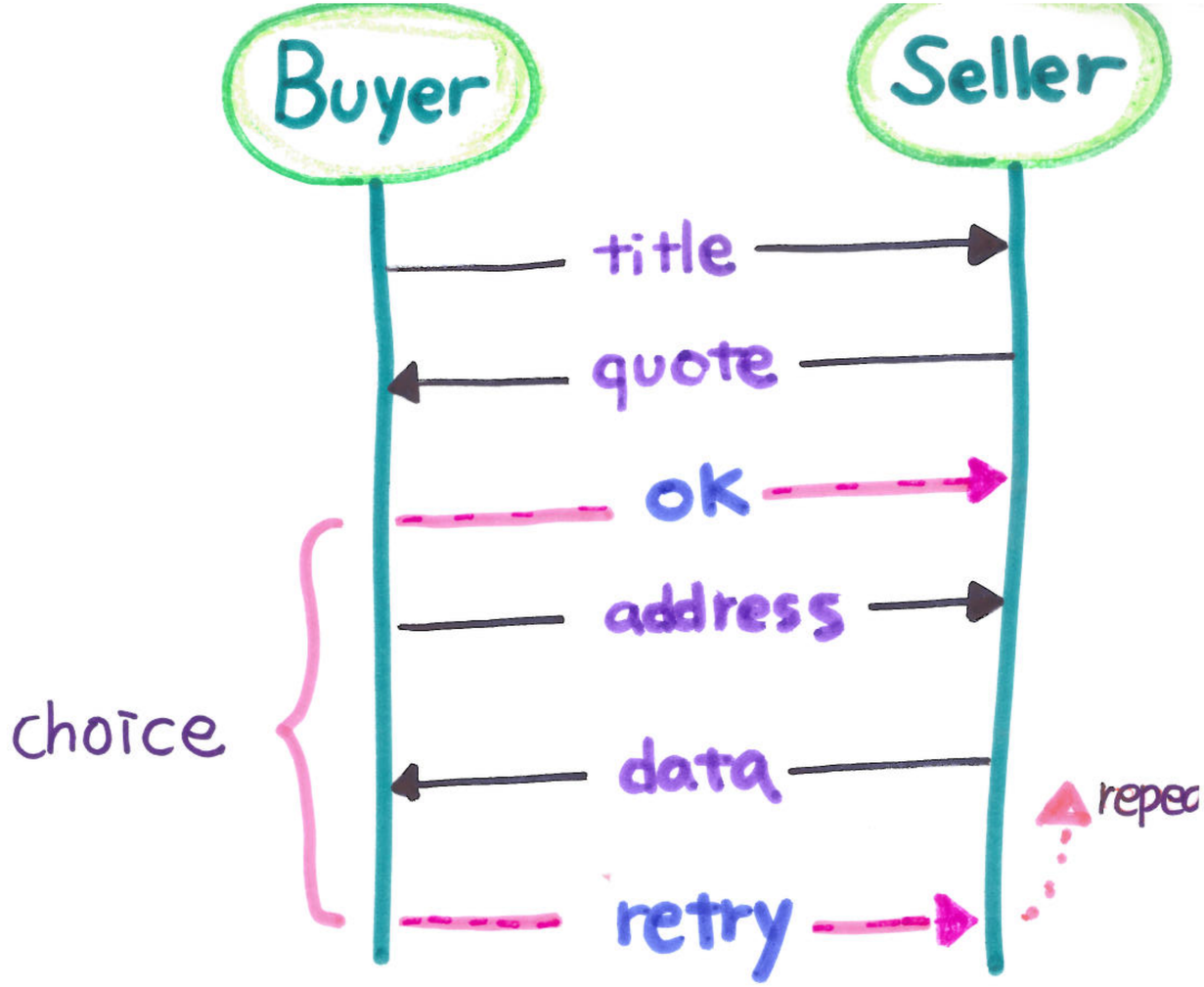
Binary Session Types: Buyer - Seller Protocol



Binary Session Types: Buyer - Seller Protocol



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



P has T

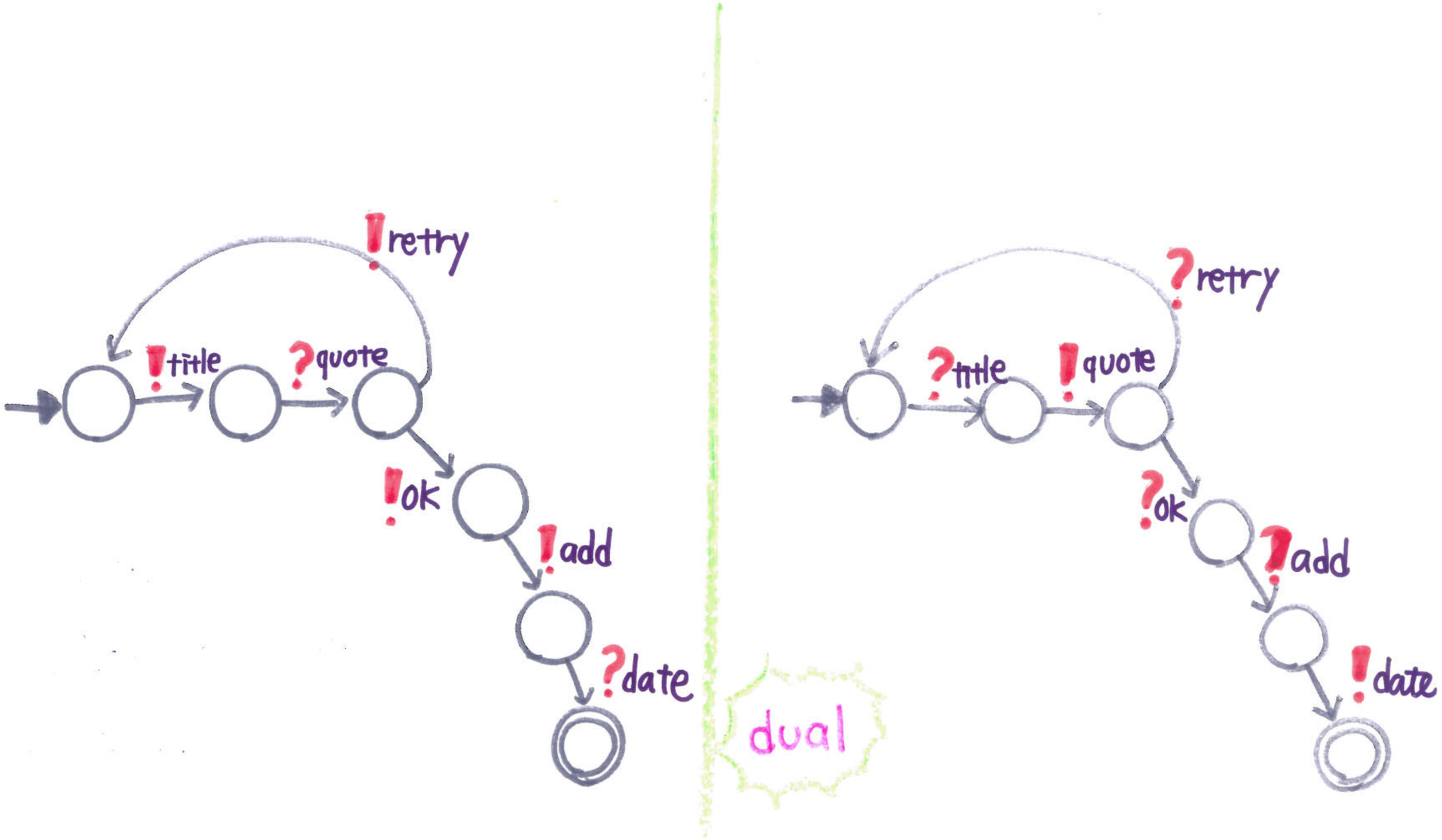
Q has $\overline{T}$ dual

P | Q typable

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

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

Communicating Automata [1980s] [Brand & Zafiropulo '83]



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

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

Q a finite set of states

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

$q0$ initial state

Σ a finite alphabet of messages

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

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

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

states queues

q_p w_{pq}

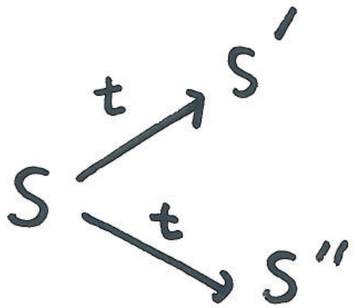
Send

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

Receive

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

Deteministic CFSM

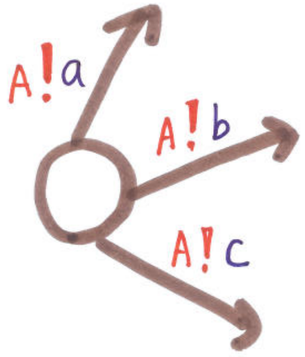


$\Rightarrow \quad s' = s''$

Basic CFSMs

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

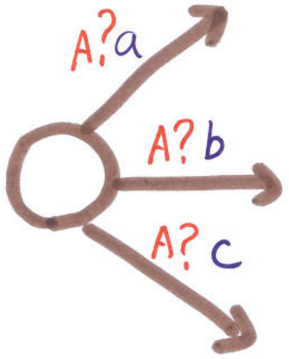
directed, has **no mixed states**



sending

✓

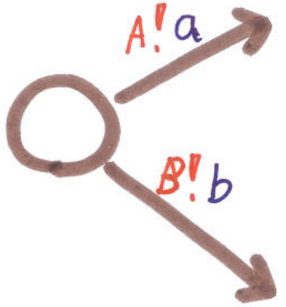
$T = A!\{a, b, c\}$



receiving

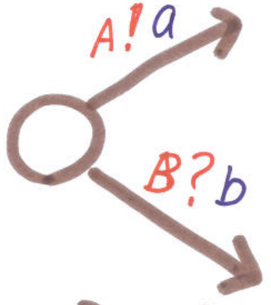
✓

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



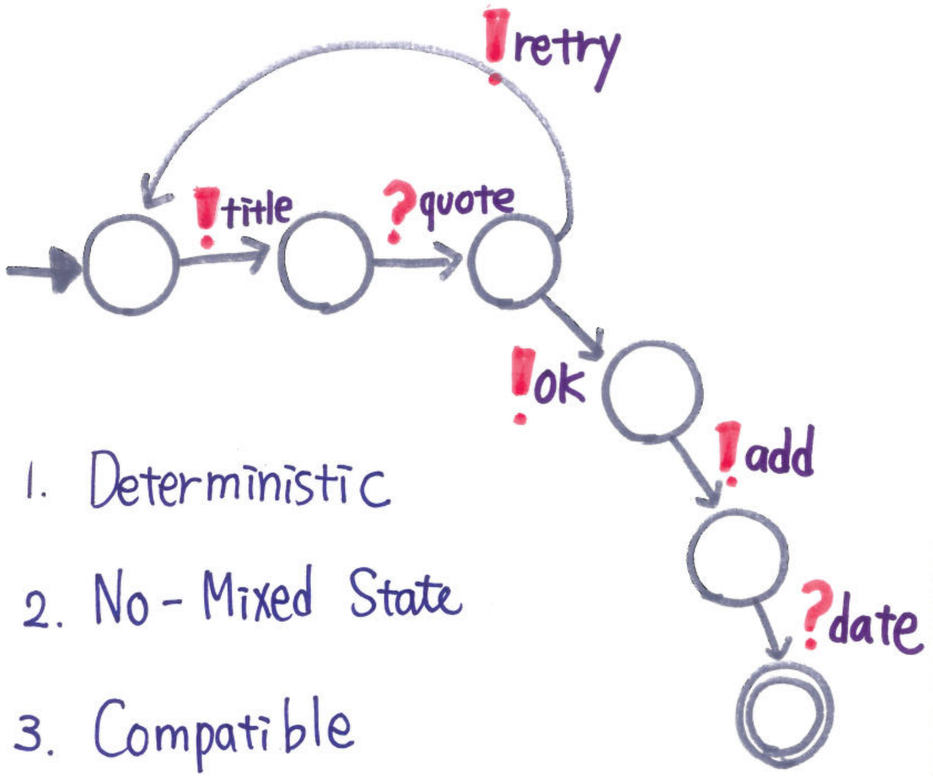
non directed

✗



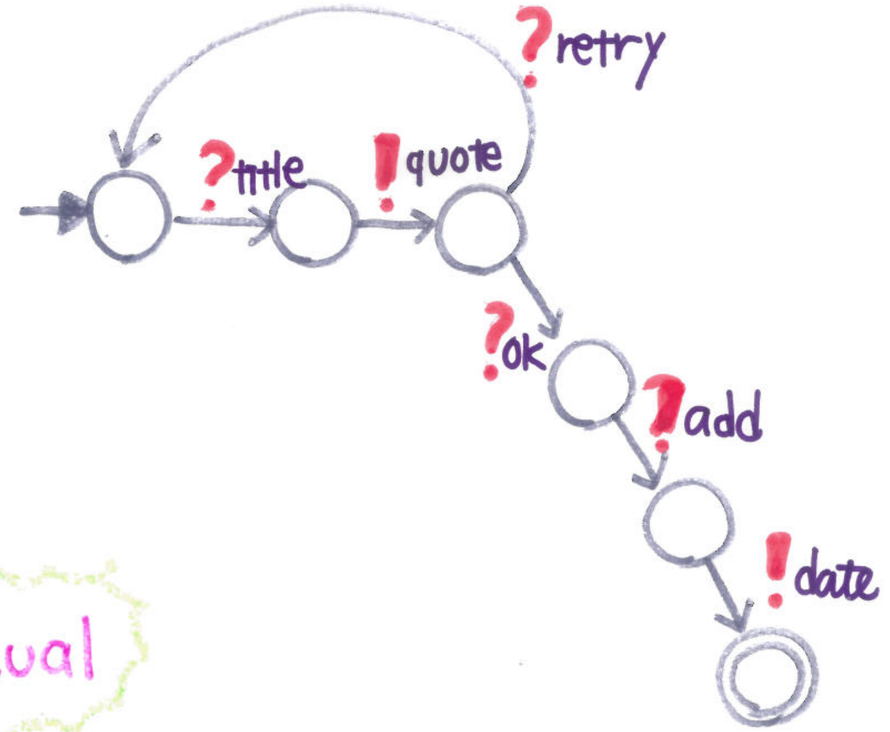
mixed

✗



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

dual



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

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

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

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

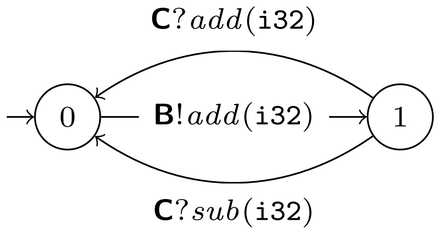
$$\mathsf{G} = \mu \mathbf{t}.\mathbf{A} \to \mathbf{B} : \left\{ add(\texttt{i32}).\mathbf{B} \to \mathbf{C} : \left\{ \begin{array}{l} add(\texttt{i32}).\mathbf{C} \to \mathbf{A} : \{add(\texttt{i32}).\mathbf{t}\} \\ sub(\texttt{i32}).\mathbf{C} \to \mathbf{A} : \{sub(\texttt{i32}).\mathbf{t}\} \end{array}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

Ring Protocol

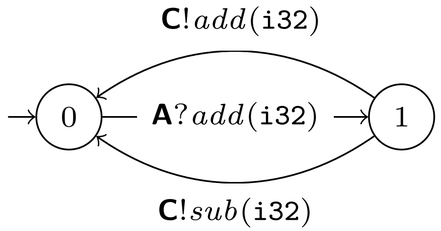
Example

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

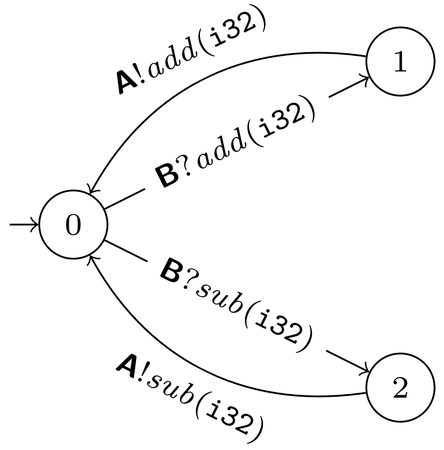
PROJECTION



PROJECTION



PROJECTION

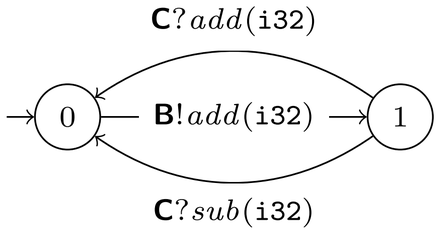


Ring Protocol

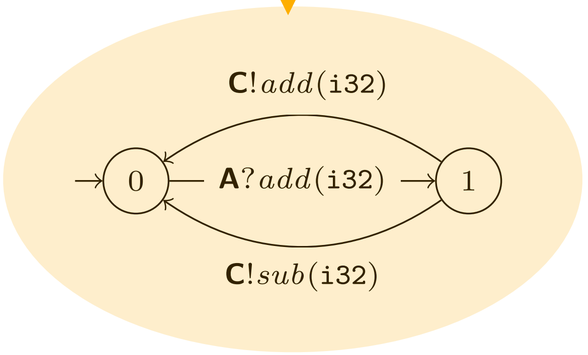
Example

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

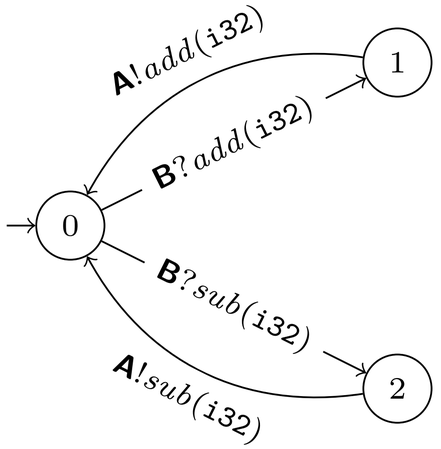
PROJECTION



PROJECTION



PROJECTION



Challenge

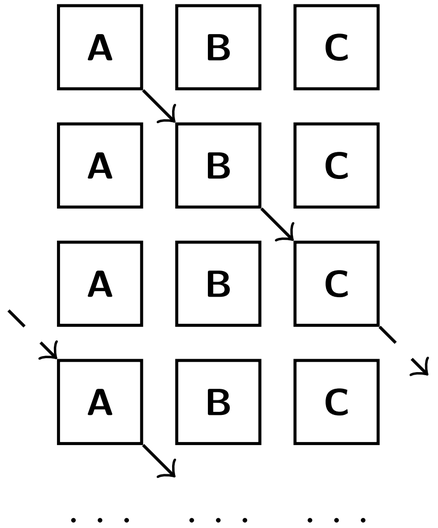
Asynchronous Orderings

- Global types are inherently synchronous

Challenge

Asynchronous Orderings

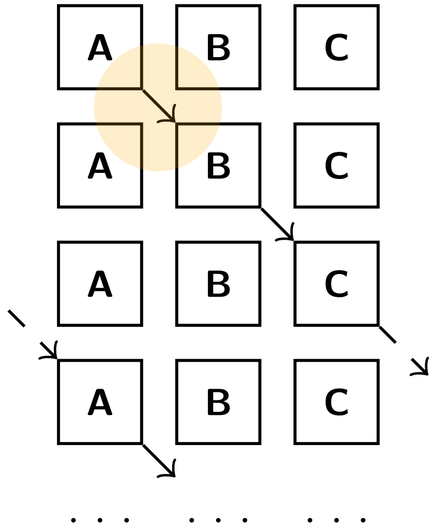
- Global types are inherently synchronous
  - Projection provides only one possible ordering



Challenge

Asynchronous Orderings

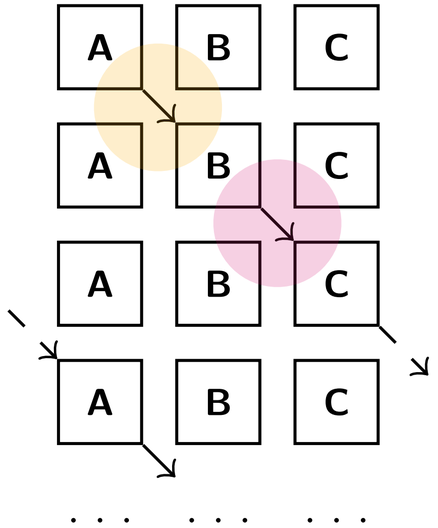
- Global types are inherently synchronous
  - Projection provides only one possible ordering



Challenge

Asynchronous Orderings

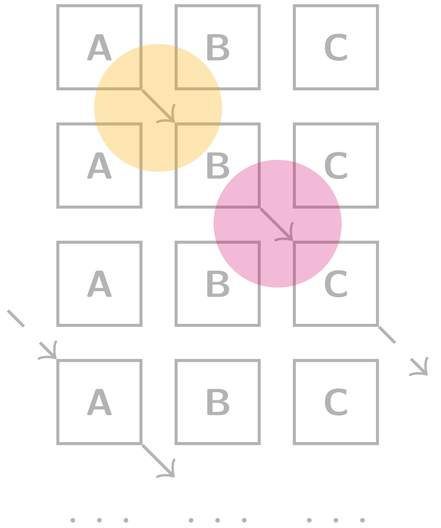
- Global types are inherently synchronous
  - Projection provides only one possible ordering

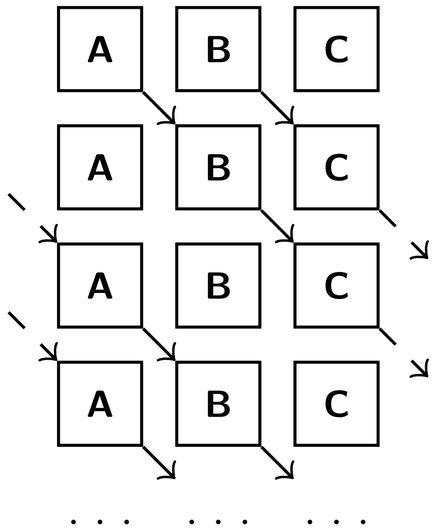


Challenge

Asynchronous Orderings

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

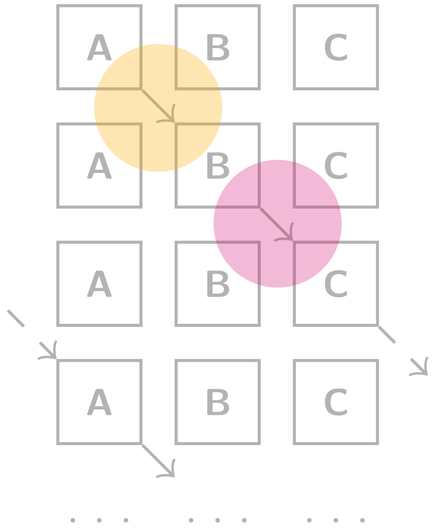


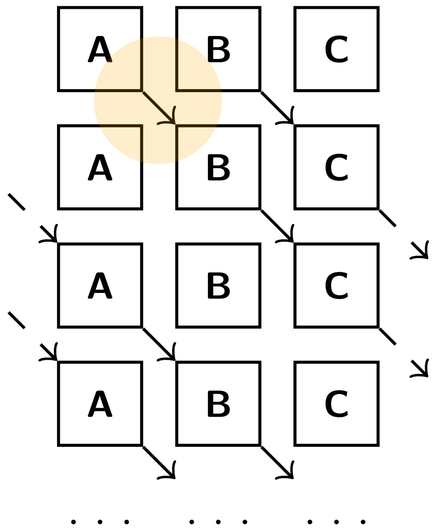


Challenge

Asynchronous Orderings

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

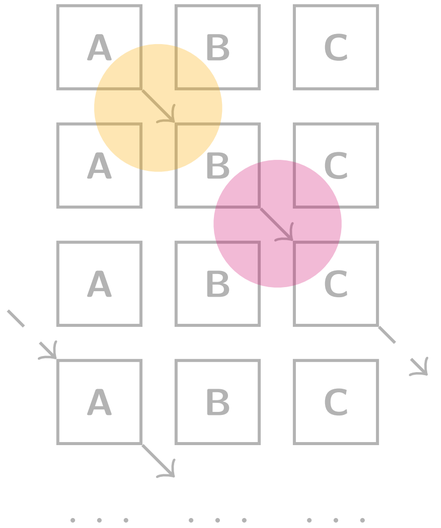


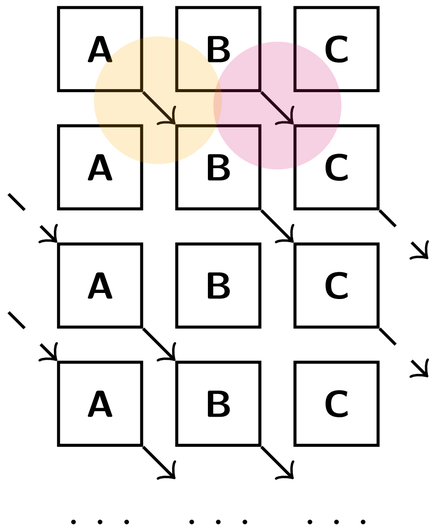


Challenge

Asynchronous Orderings

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

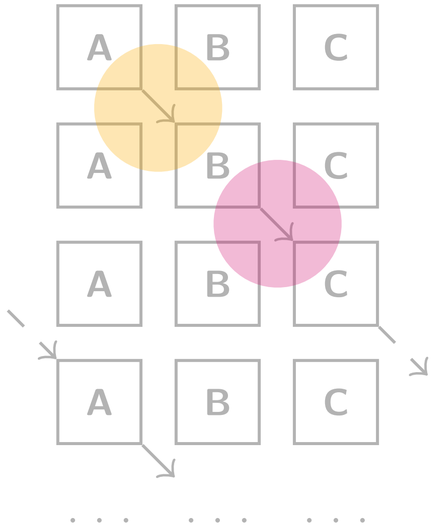


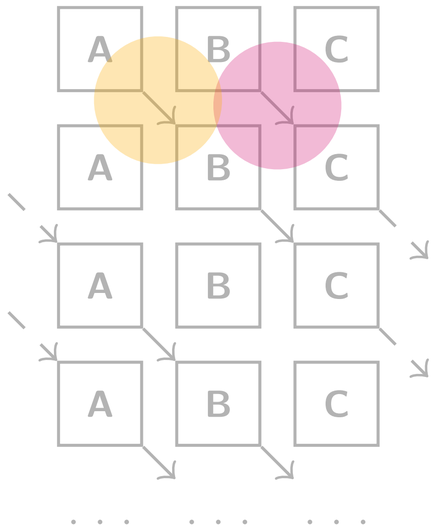


Challenge

Asynchronous Orderings

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

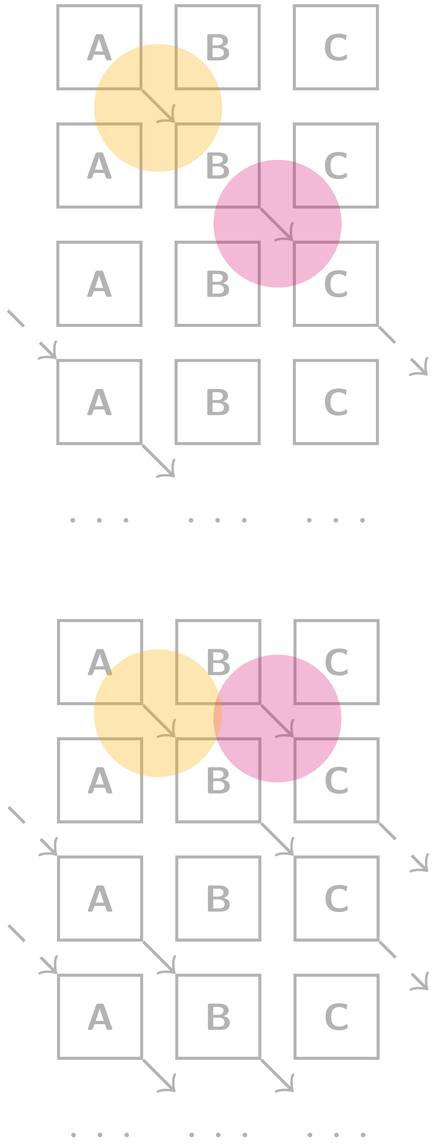




Challenge

Asynchronous Orderings

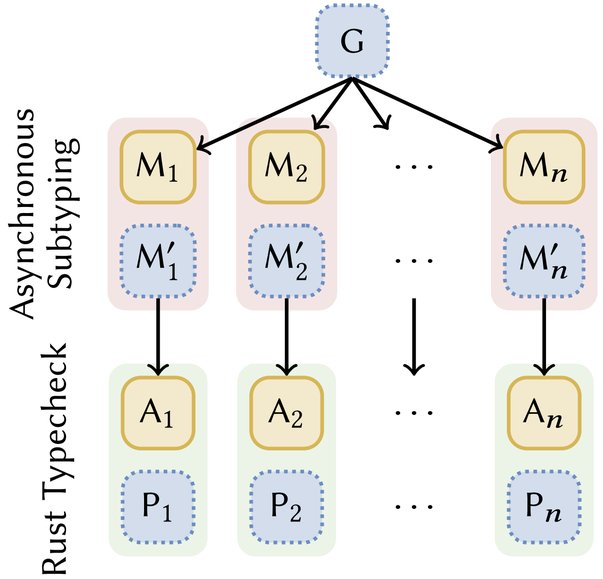
- Global types are inherently synchronous
  - Projection provides only one possible ordering
- Interactions can be reordered for efficiency while preserving safety
  1. Data dependencies must be preserved
  2. Sound and practical asynchronous reordering rules must be found



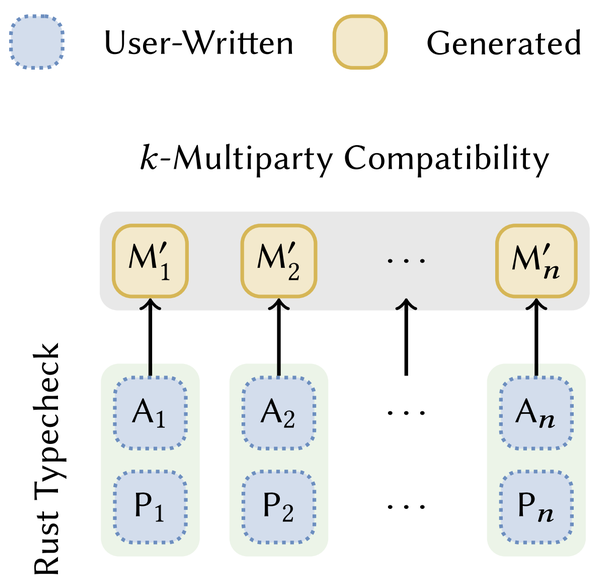
Rumpsteak Framework (Rust)

Three Approaches [CNV, PPoPP'21]

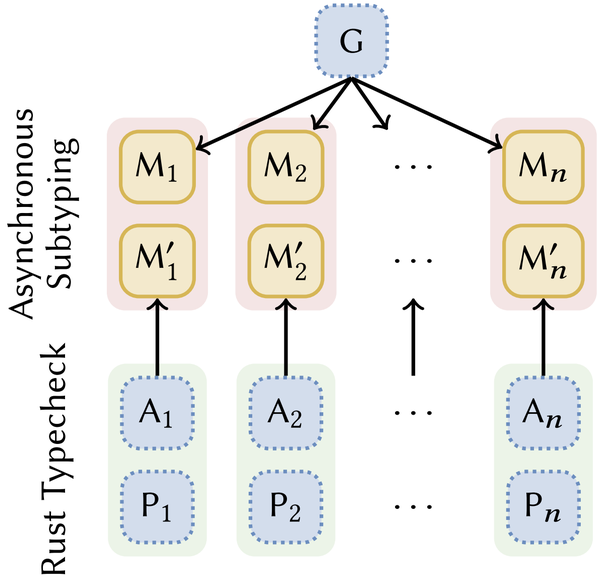
G Global Type **M** Finite State Machine (FSM) **M′** Optimised FSM **A** Rust API **P** Rust Process



(a) Top-down



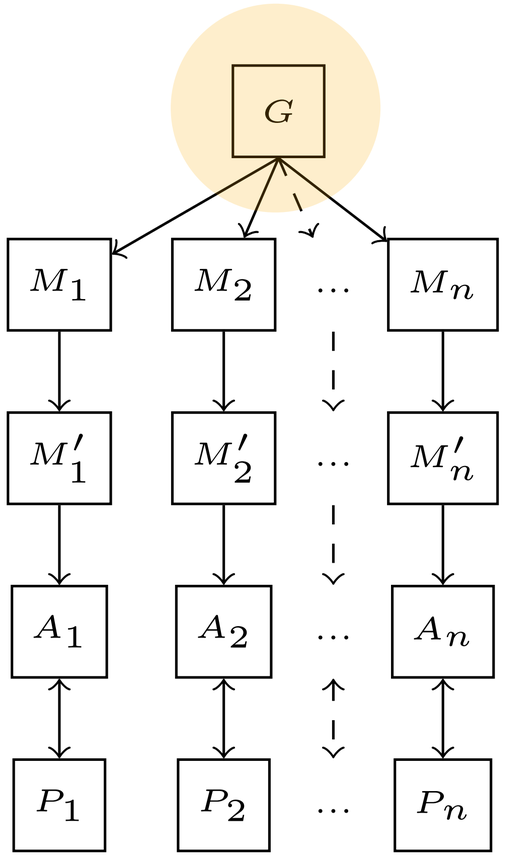
(b) Bottom-up



(c) Hybrid

Workflow

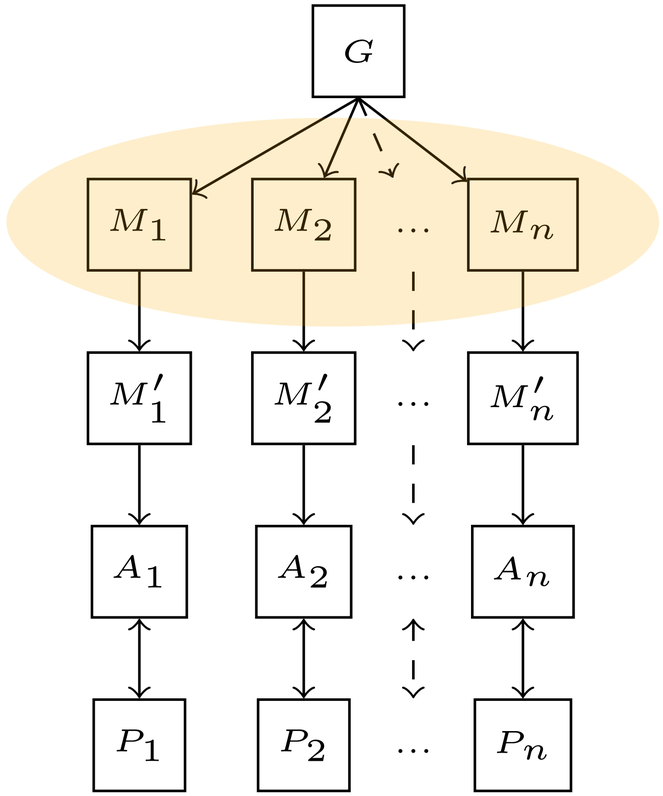
Top-Down Approach



Global Type

Workflow

Top-Down Approach

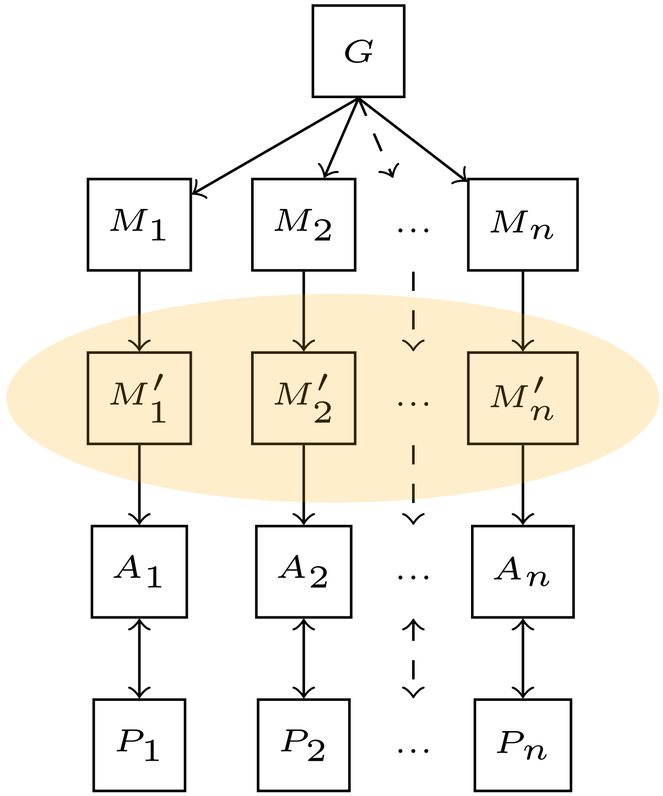


Global Type

Projected FSM

Workflow

Top-Down Approach



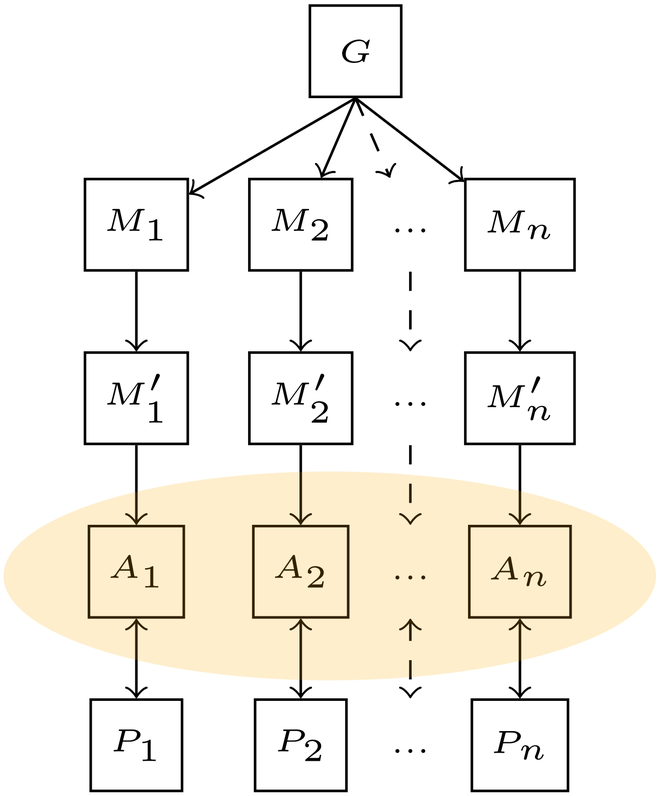
Global Type

Projected FSM

Optimised FSM

Workflow

Top-Down Approach



Global Type

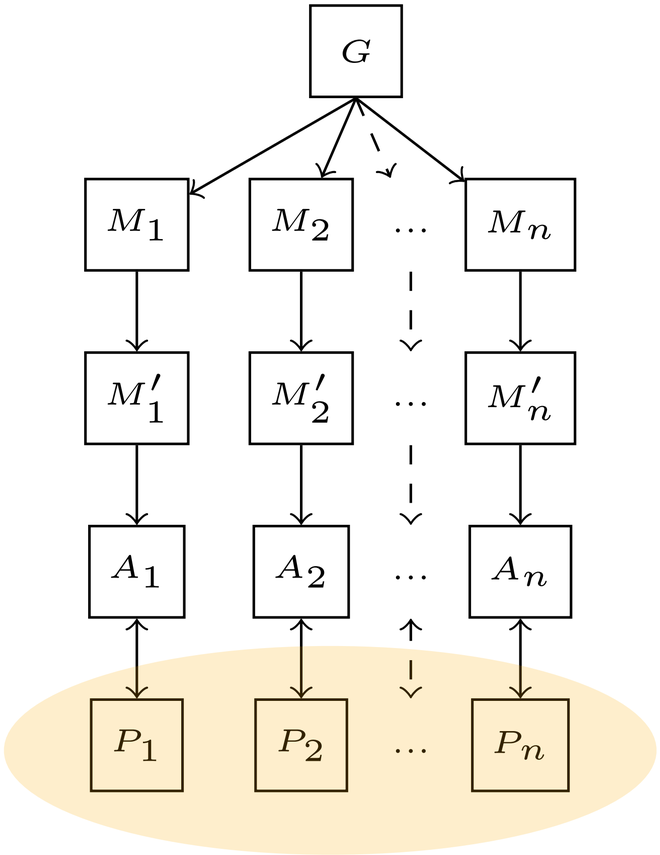
Projected FSM

Optimised FSM

Rust API

Workflow

Top-Down Approach



Global Type

Projected FSM

Optimised FSM

Rust API

Rust Implementation

Ring Protocol

Example

Global Type

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

Ring Protocol

Example

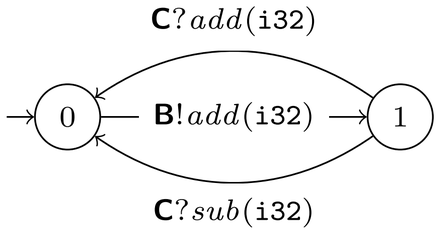
$$\mathsf{G} = \mu \mathbf{t}.\mathbf{A} \to \mathbf{B} : \left\{ add(\texttt{i32}).\mathbf{B} \to \mathbf{C} : \left\{ \begin{array}{l} add(\texttt{i32}).\mathbf{C} \to \mathbf{A} : \{add(\texttt{i32}).\mathbf{t}\} \\ sub(\texttt{i32}).\mathbf{C} \to \mathbf{A} : \{sub(\texttt{i32}).\mathbf{t}\} \end{array}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

Ring Protocol

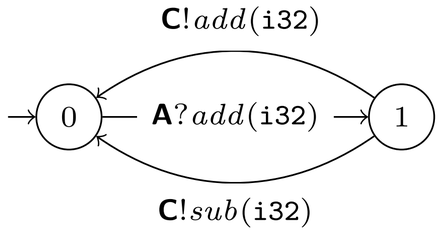
Example

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

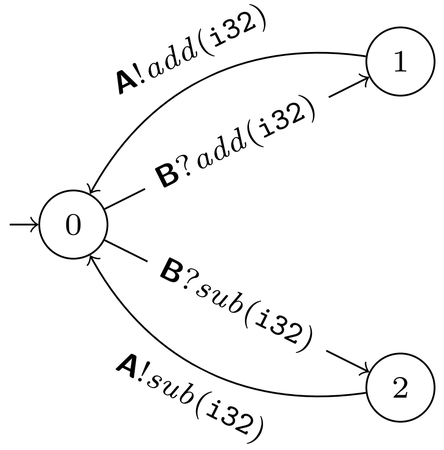
PROJECTION



PROJECTION

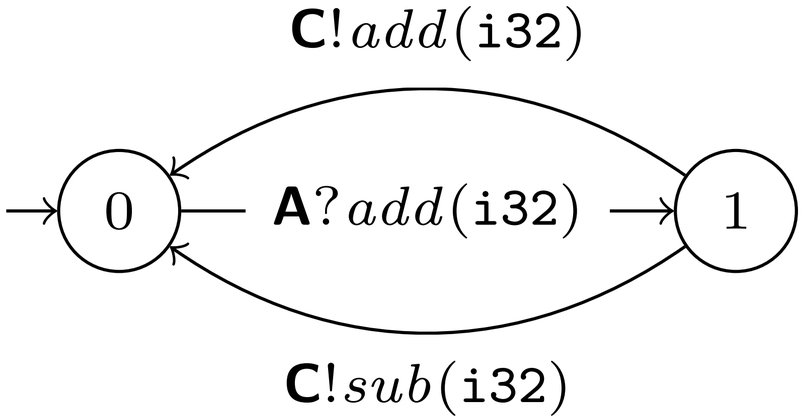


PROJECTION



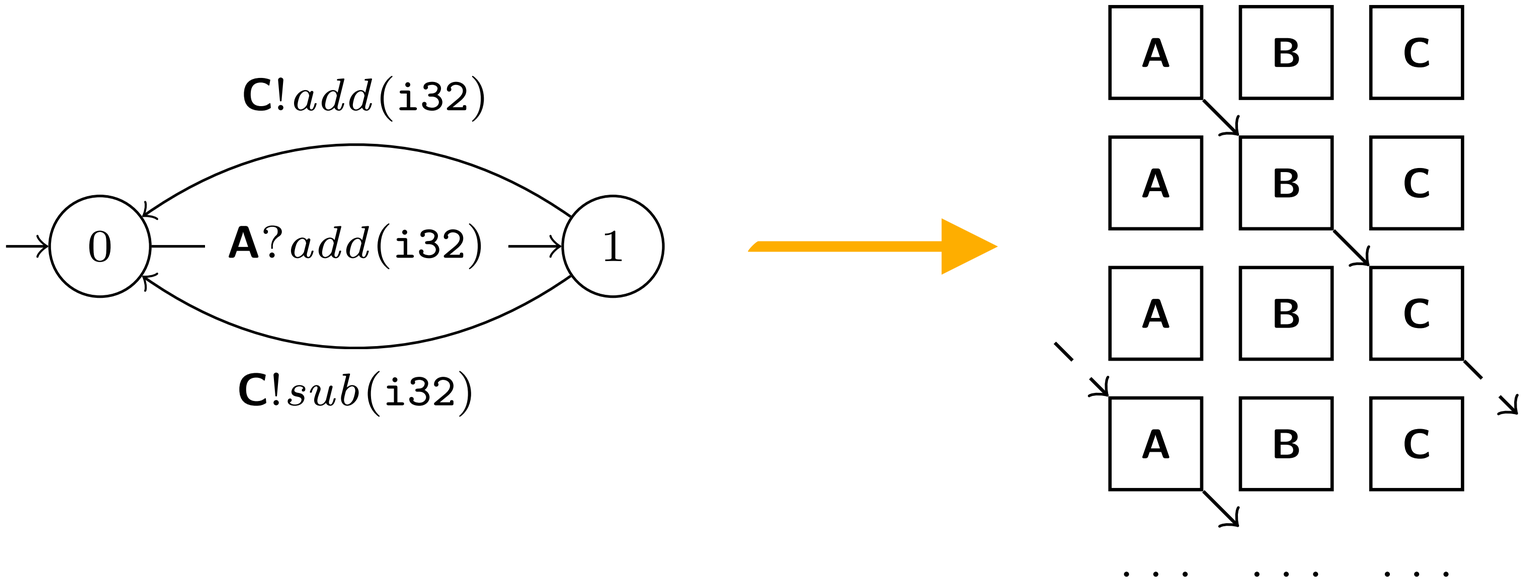
Ring Protocol

Example



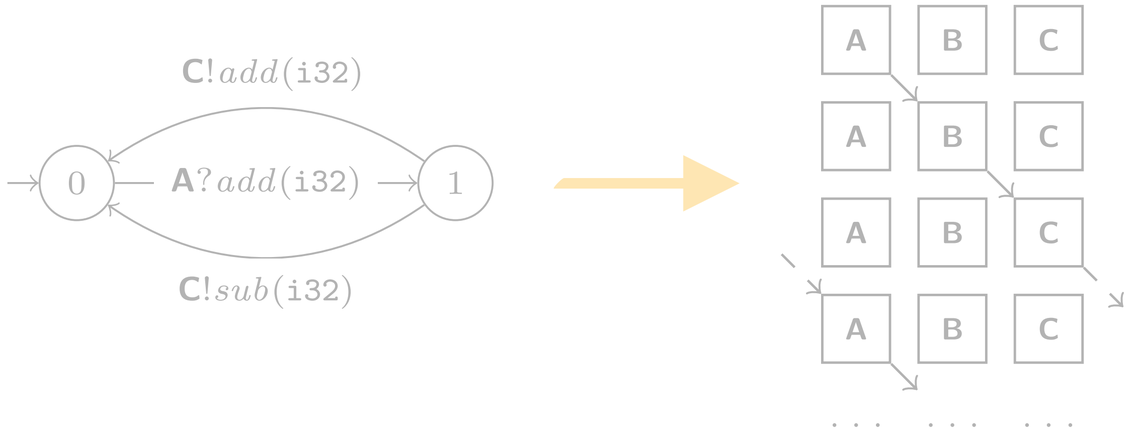
Ring Protocol

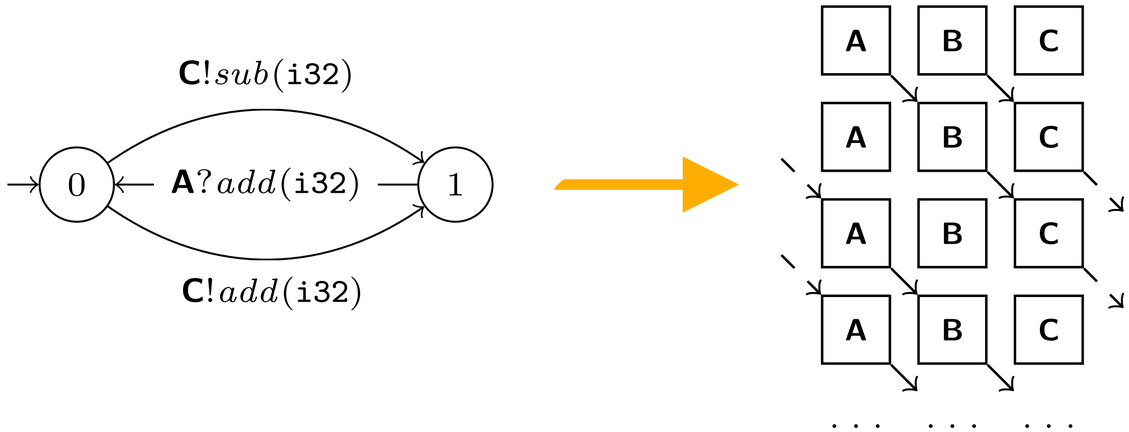
Example



Ring Protocol

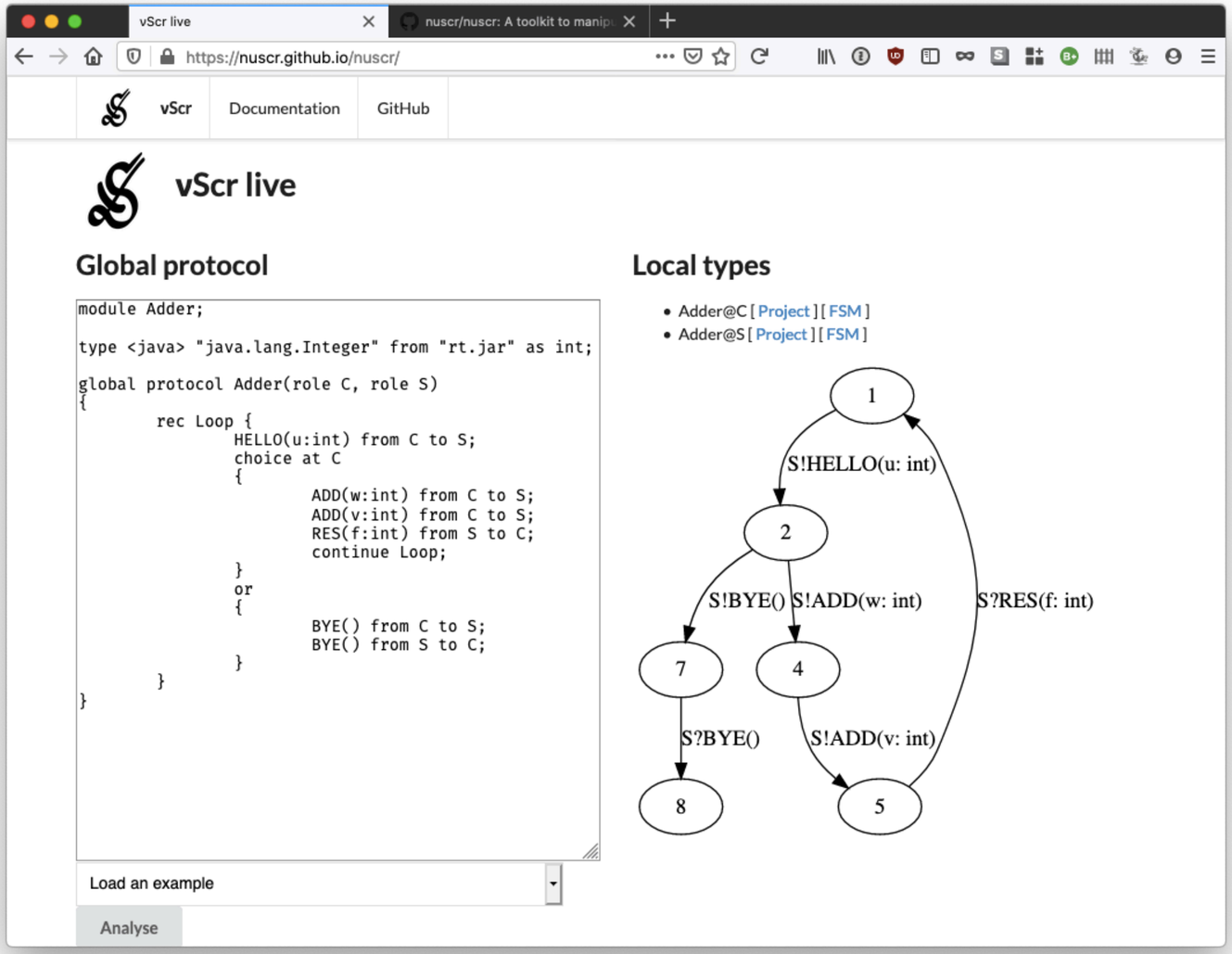
Example





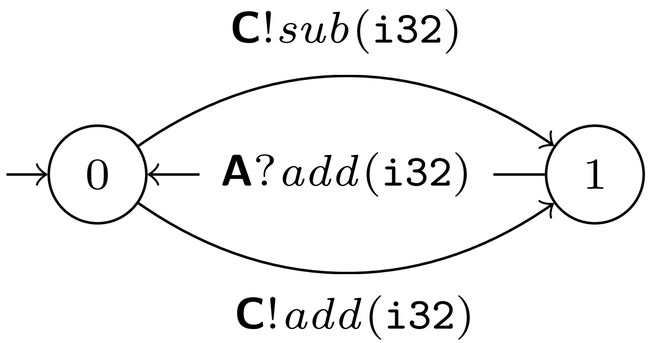
vScr An Extensible Toolchain for Multiparty Session Types

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



Ring Protocol

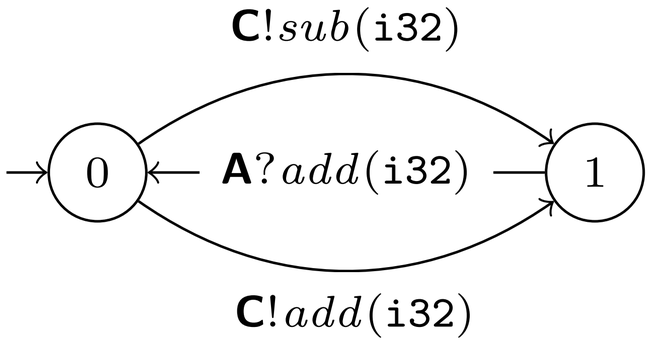
Rust API




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

Ring Protocol

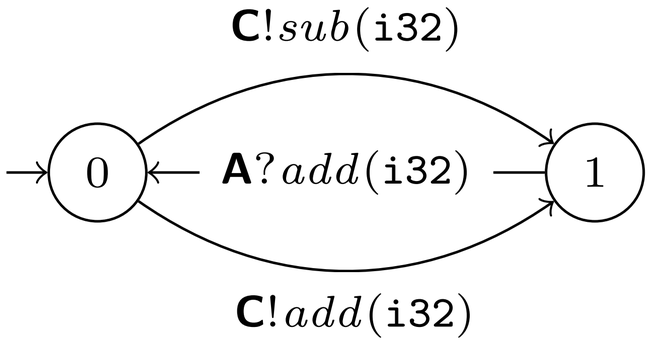
Rust API




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

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

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

Ring Protocol

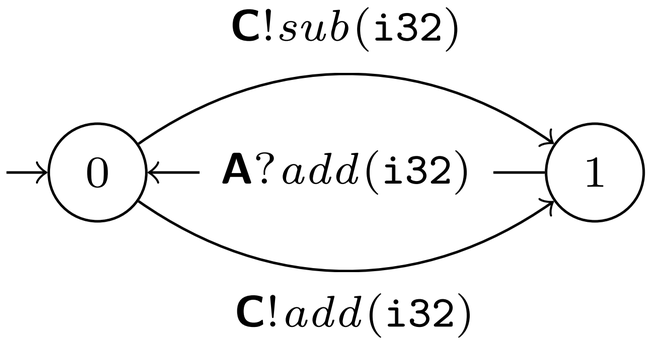
Rust API




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

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

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

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

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

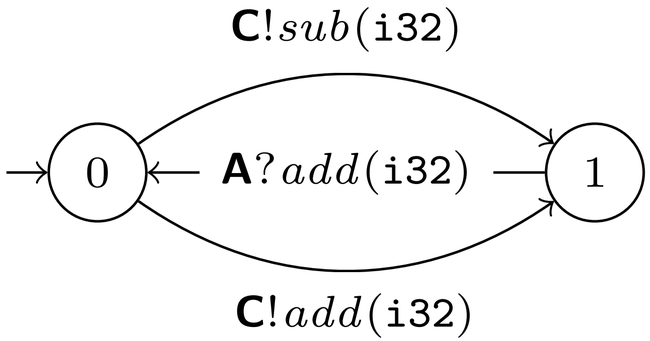
Ring Protocol

Rust API



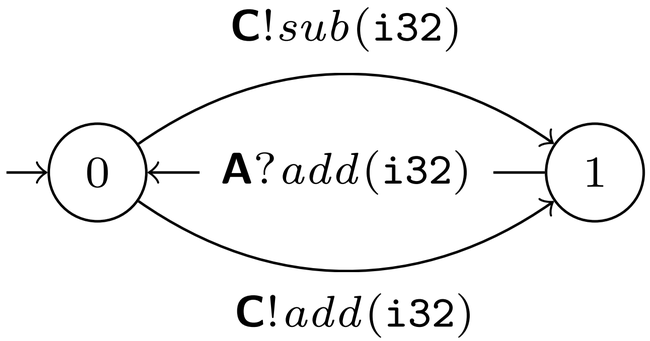
Ring Protocol

Implementation



Ring Protocol

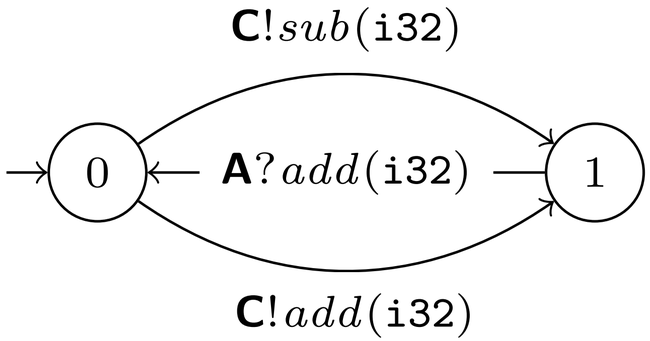
Implementation




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

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

Ring Protocol

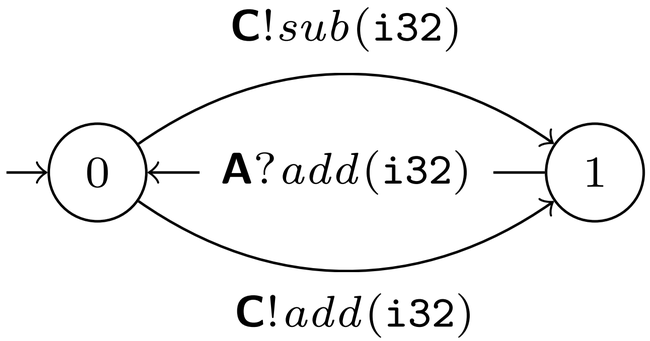
Implementation




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

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

Ring Protocol

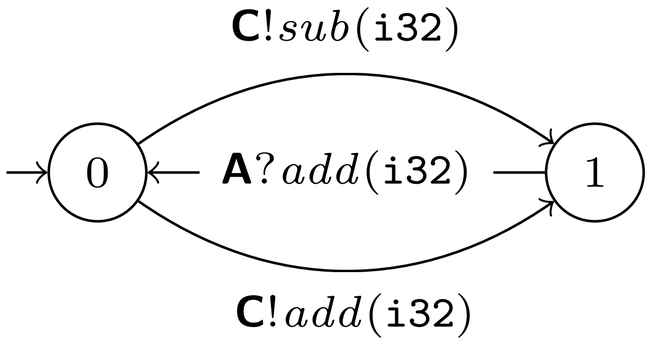
Implementation




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

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

Ring Protocol

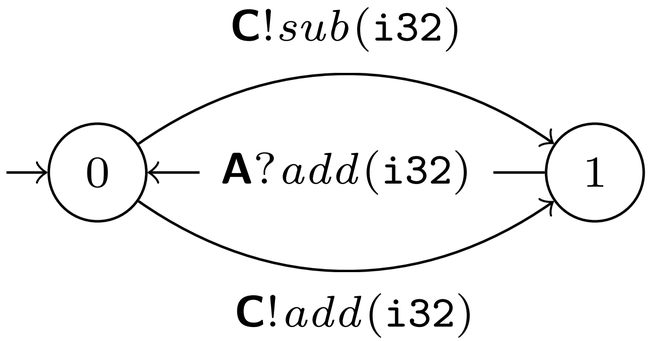
Implementation




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

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

Ring Protocol

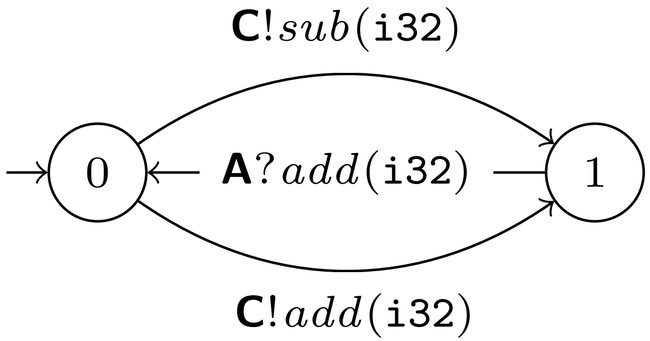
Implementation




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

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

Ring Protocol

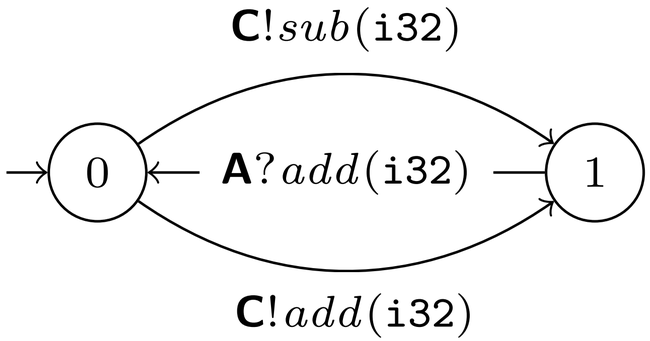
Implementation




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

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

Ring Protocol

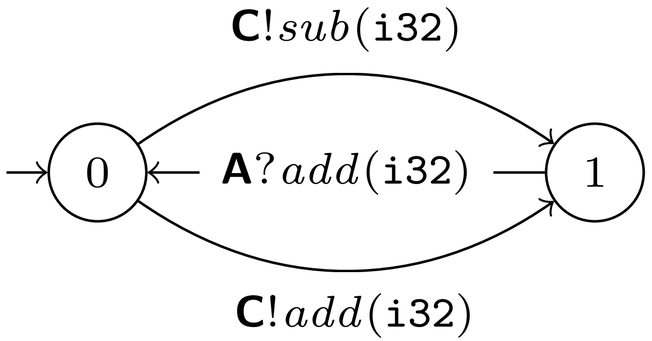
Implementation




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

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

Ring Protocol

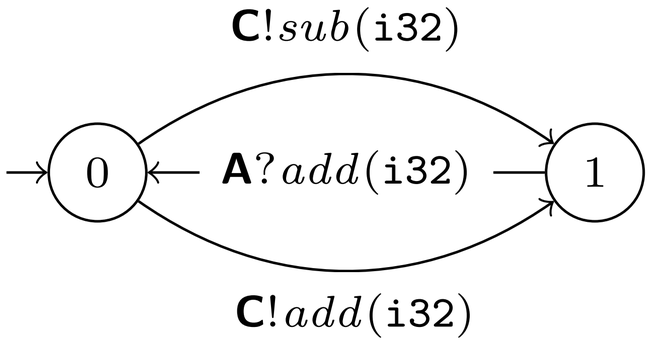
Implementation




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

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

Ring Protocol

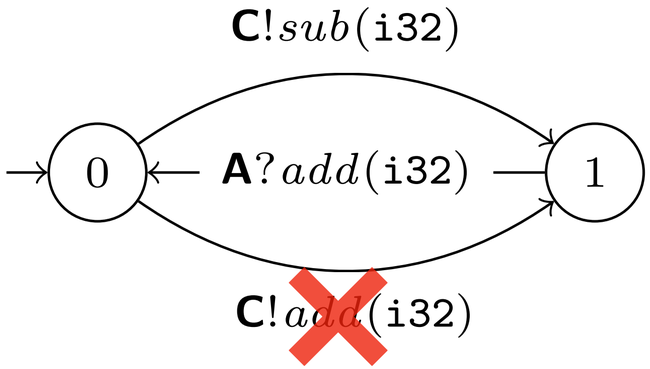
Implementation




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

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

Ring Protocol

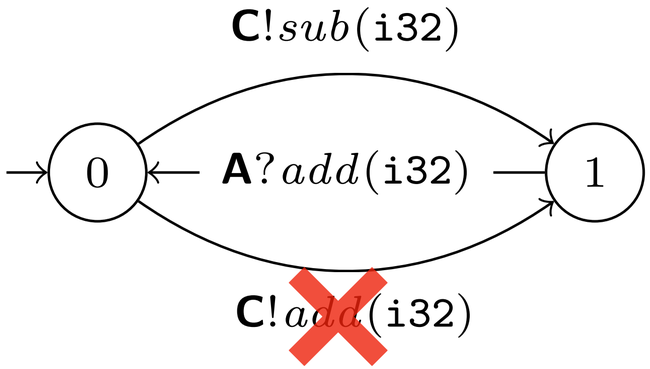
Implementation




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

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

Ring Protocol

Implementation



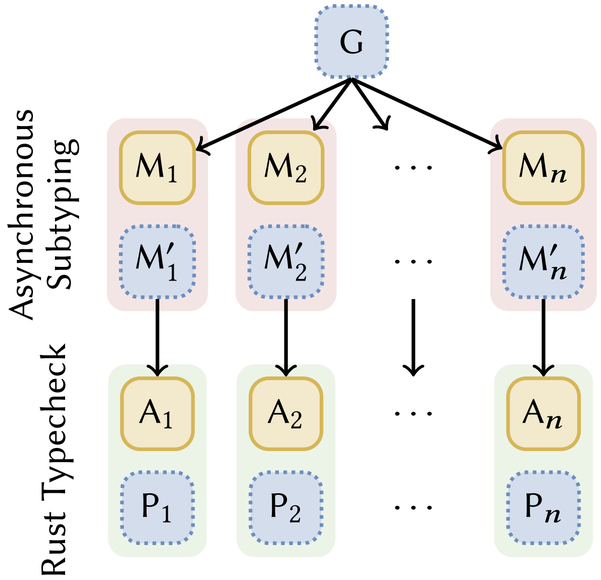

```rust
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

            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
}
```

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

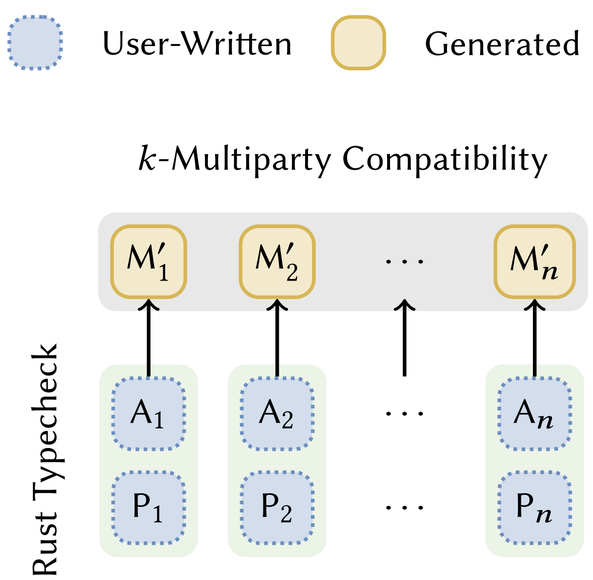
Rumpsteak Framework

Three Approaches

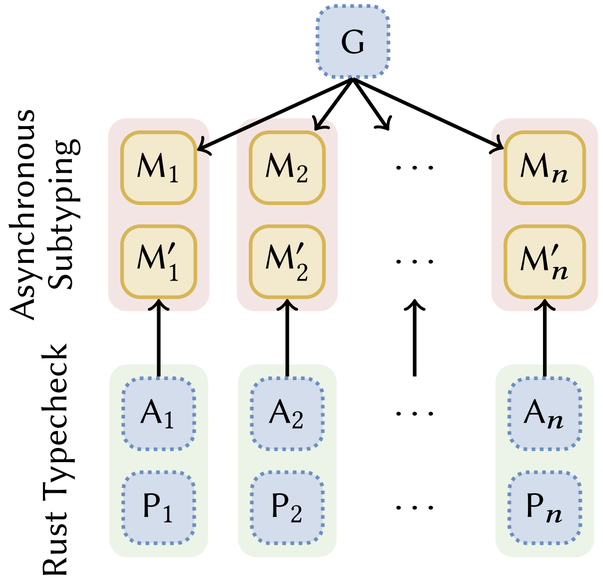
G Global Type **M** Finite State Machine (FSM) **M′** Optimised FSM **A** Rust API **P** Rust Process



(a) Top-down



(b) Bottom-up



(c) Hybrid

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?

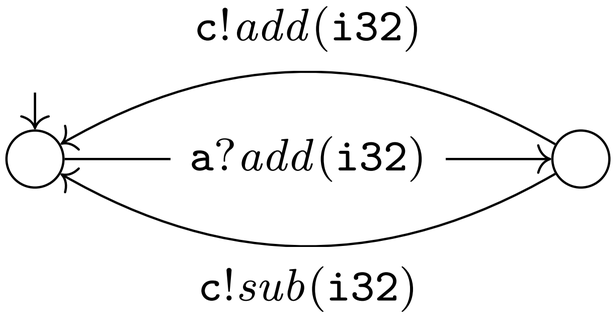
1. Asynchronous subtyping [Ghilezan, Pantvic, Prokic, Scalas and NY **POPL'2021**]

2. *k*-multiparty compatibility [Lange and NY, **CAV'2019**]

Safety

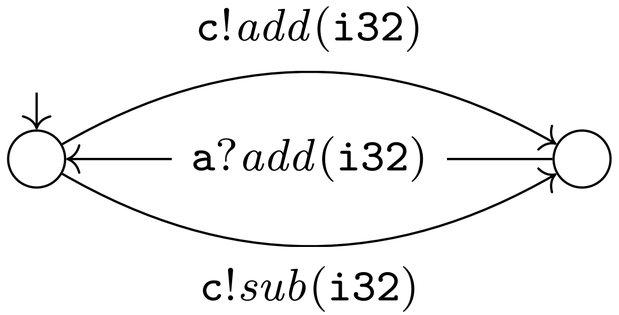
Asynchronous Subtyping

PROJECTED B



Safe?

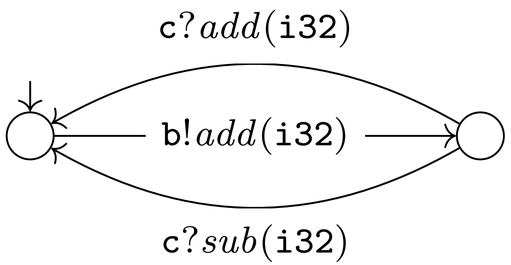
OPTIMISED B



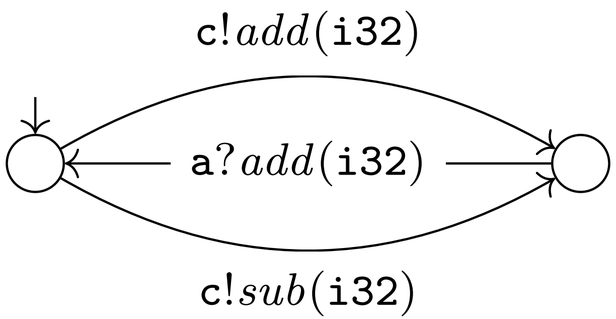
Safety

k-Multiparty Compatibility

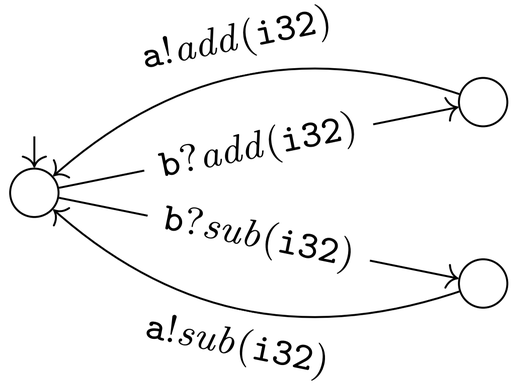
OPTIMISED A



OPTIMISED B

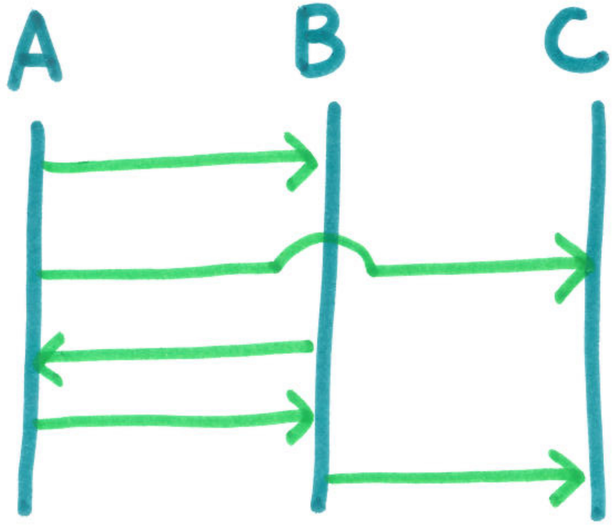


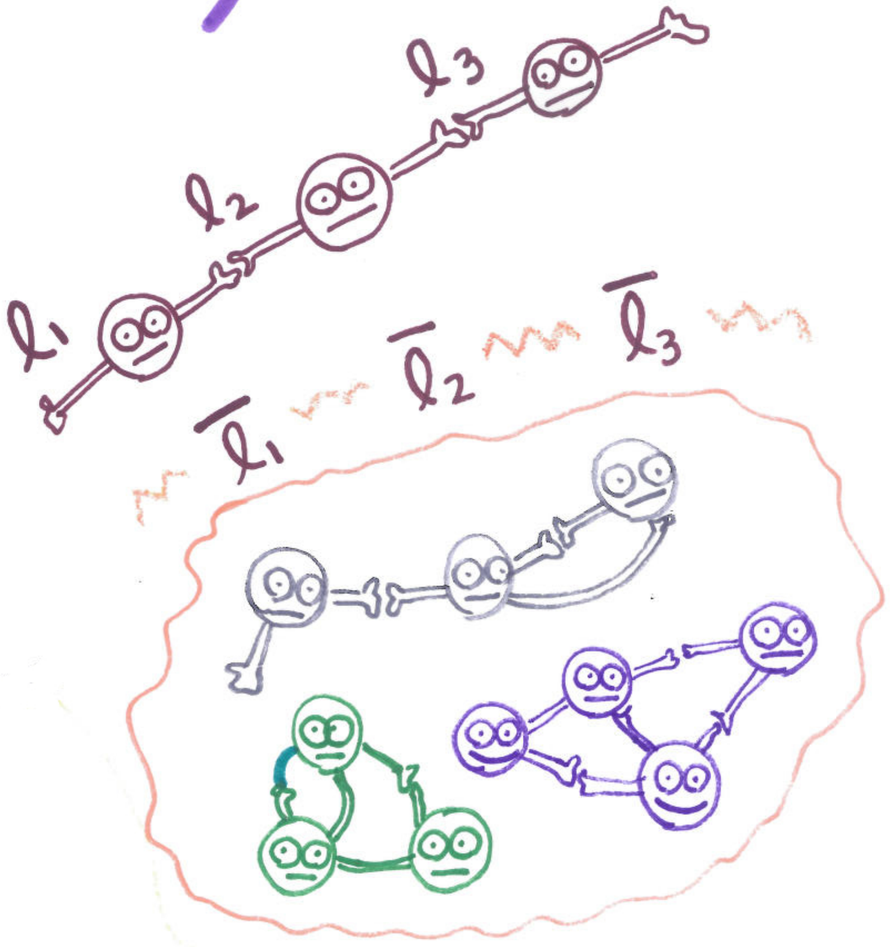
OPTIMISED C



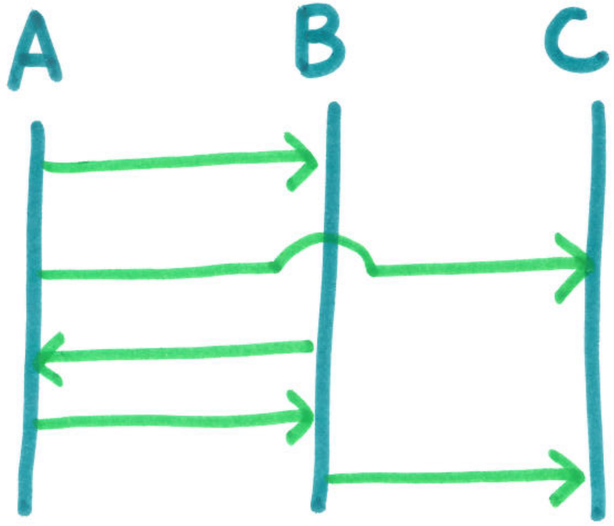
Safe?

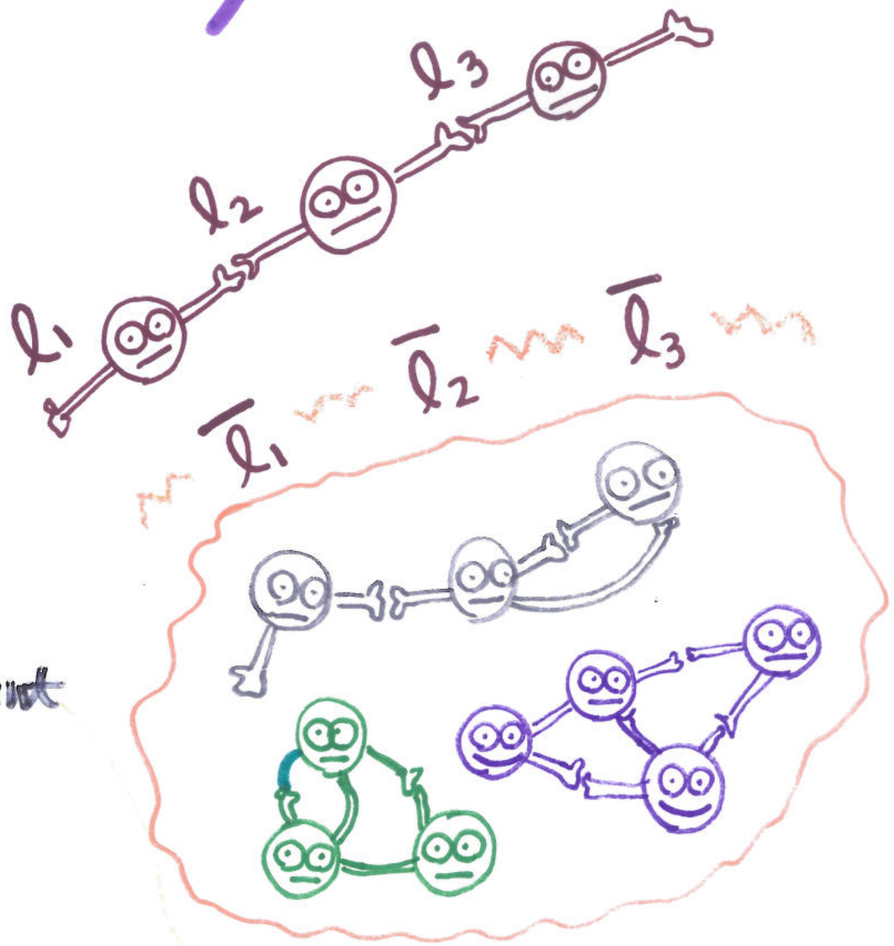
Multiparty Compatibility





Multiparty Compatibility





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

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

if M_i does action ℓ

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

after some $\rightsquigarrow$

Definition 4 (k-Safety). *S is* k-safe *if the conditions below hold for all $s = (\boldsymbol{q}; \boldsymbol{w}) \in RS_k(S)$:*

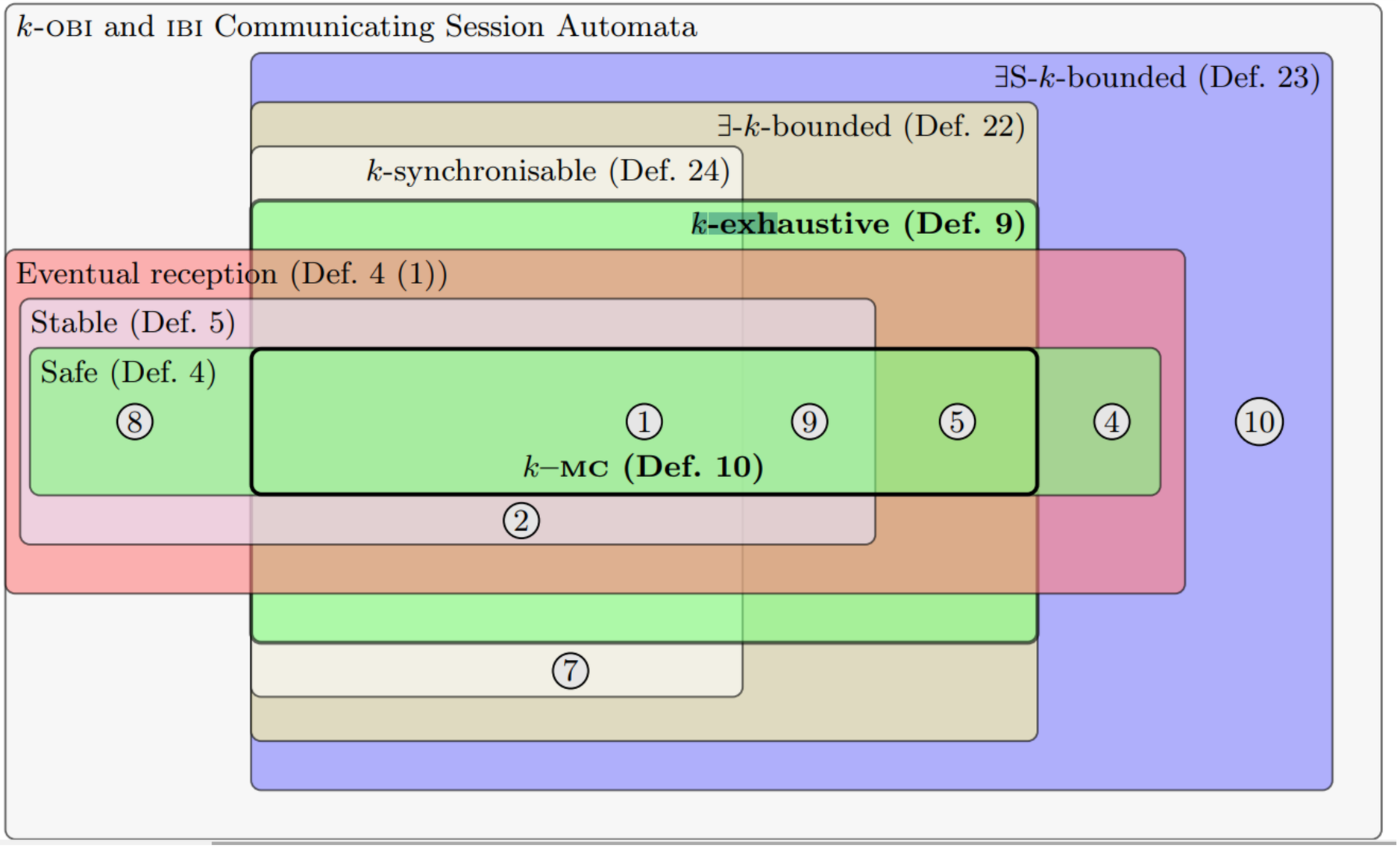
1. *For all $\mathtt{pq} \in \mathcal{C}$, if $w_{\mathtt{pq}} = a \cdot w'$, then $s \rightarrow_k{}^* \xrightarrow{\mathtt{pq}?a}_k$.*
2. *For all $\mathtt{p} \in \mathcal{P}$, if $q_{\mathtt{p}}$ is a* receiving *state, then $s \rightarrow_k{}^* \xrightarrow{\mathtt{qp}?a}_k$ for some $\mathtt{q} \in \mathcal{P}$ and $a \in \Sigma$.*

We say that S is safe *if it validates the unbounded version of k-safety (∞-safe).*

Definition 5 (Stable). *S has the* stable property *(SP) if $\forall s \in RS(S) : \exists (\boldsymbol{q}; \boldsymbol{\epsilon}) \in RS(S) : s \rightarrow^* (\boldsymbol{q}; \boldsymbol{\epsilon})$.*

Definition 9 (k-Exhaustivity). *S is k-*exhaustive *if for all $s = (\boldsymbol{q}; \boldsymbol{w}) \in RS_k(S)$ and $\mathtt{p} \in \mathcal{P}$, if $q_{\mathtt{p}}$ is a sending state, then $\forall (q_{\mathtt{p}}, \ell, q'_{\mathtt{p}}) \in \delta_{\mathtt{p}} : \exists \phi \in \mathcal{A}^* : s \xrightarrow{\phi}_k \xrightarrow{\ell}_k$ and $\mathtt{p} \notin \phi$.*

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 [Bravetti et al., Lange and NY] ❌

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 [Bravetti et al., Lange and NY] ❌
- 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 [Bravetti et al., Lange and NY] ❌
- 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

Session Type Prefix

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

Asynchronous Subtyping

Reduction Rules

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

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

Asynchronous Subtyping

Reduction Rules

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

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

Asynchronous Subtyping

Reduction Rules

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

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

Asynchronous Subtyping

Reduction Rules

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

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

Asynchronous Subtyping

Reduction Rules

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

Asynchronous Subtyping

Reduction Rules

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

Asynchronous Subtyping

Reduction Rules

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

Asynchronous Subtyping

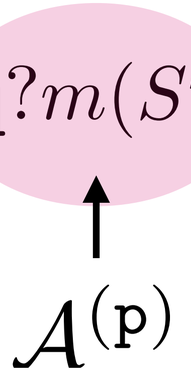
Reduction Rules

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

Asynchronous Subtyping

Reduction Rules

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



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

✅

Asynchronous Subtyping

Reduction Rules

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

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

Asynchronous Subtyping

Reduction Rules

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

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

Asynchronous Subtyping

Reduction Rules

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

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

Asynchronous Subtyping

Reduction Rules

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

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

Asynchronous Subtyping

Reduction Rules

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

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

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

Asynchronous Subtyping

Reduction Rules

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

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

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

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

Theorems

Termination, Soundness & Complexity

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

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

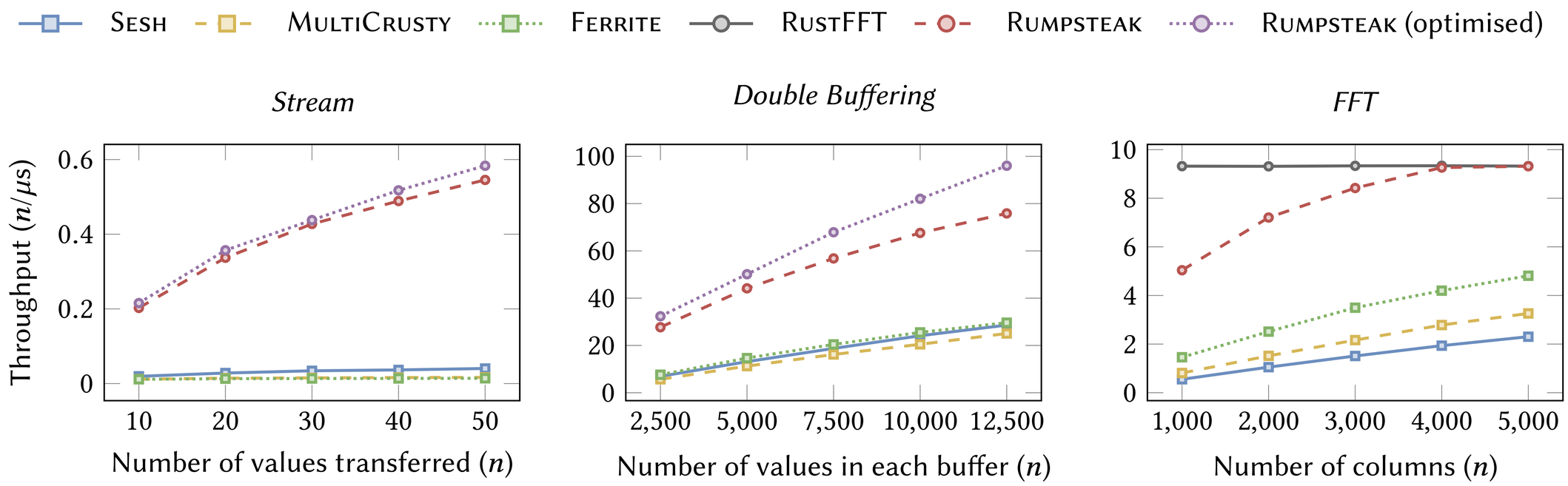
Theorem 5 (Soundness)**.** *Our subtyping algorithm is sound.*

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

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

Evaluation

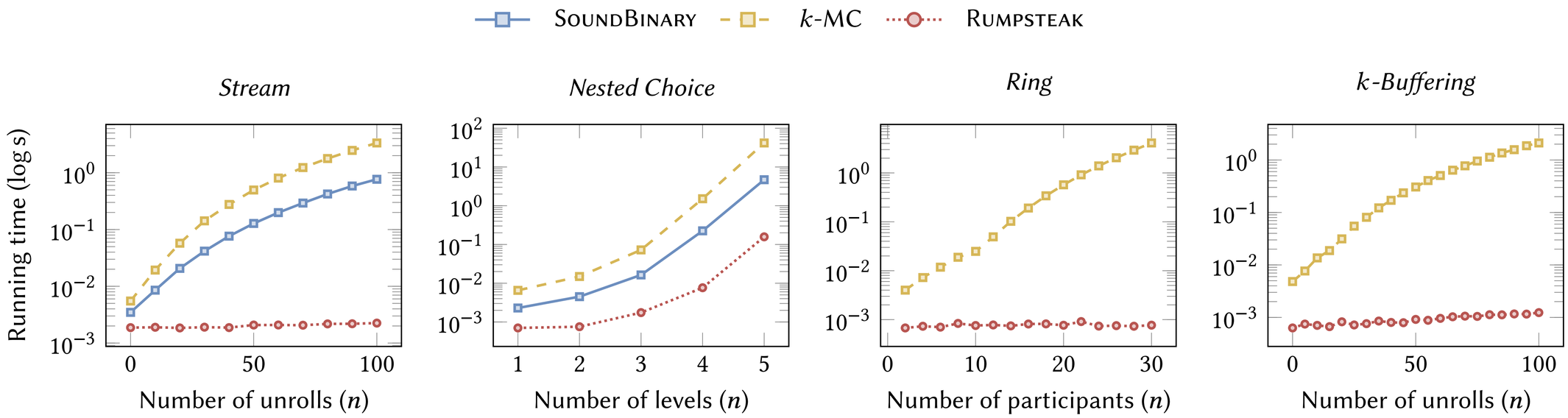
Rust Framework Benchmarks



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

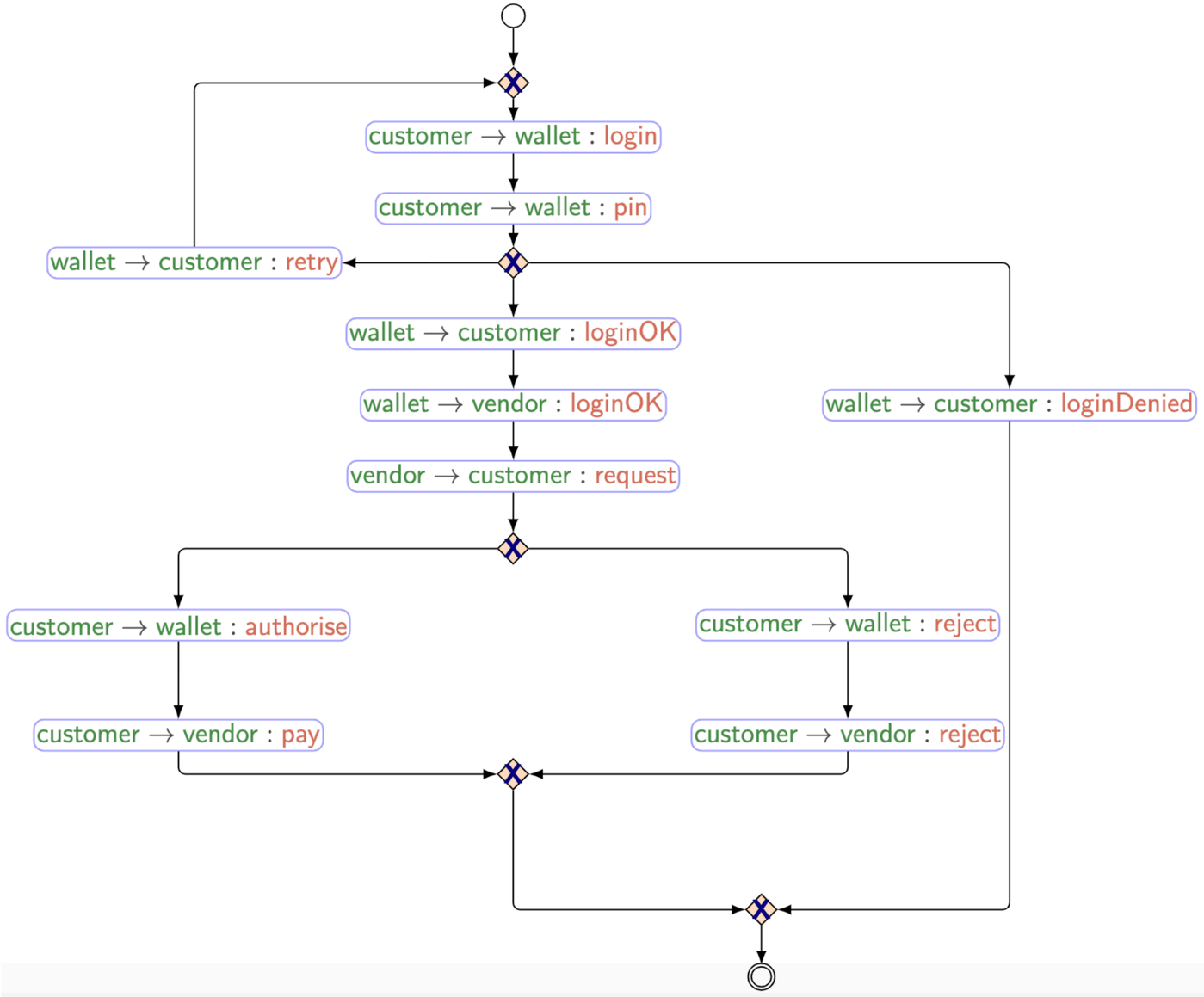
Evaluation

Asynchronous Reordering Benchmarks



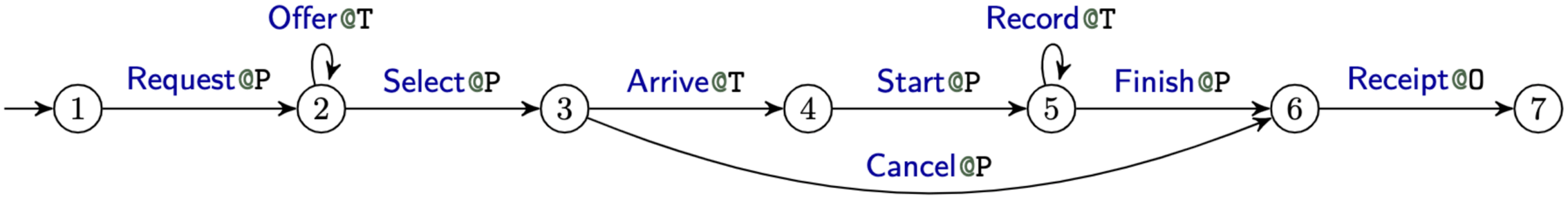
Choreography Automata

Tuosto et al



Swam protocols

Kuhn et al [ECOOP 2023]



Top-down

- Less is more [Scalas and NY, POPL'19]
  - End-Point Projections with Sums (by Scalas and NY, 2019)
    - Plain projection [POPL'08] has no issue (**Two Buyer Protocol**)
    - Meageability is sound but the proof of subject reduction was incorrect (*balanced* called *coherence* is too strong)
    - $$\frac{\Gamma' = \{s[\mathbf{p}] : S_{\mathbf{p}}\}_{\mathbf{p} \in I} \quad s \notin \Gamma \quad \text{consistent}(\Gamma') \quad \Theta \cdot \Gamma, \Gamma' \vdash P}{\Theta \cdot \Gamma \vdash (\nu s : \Gamma')\, P} \; [\text{T-}\nu\text{CLASSIC}]$$

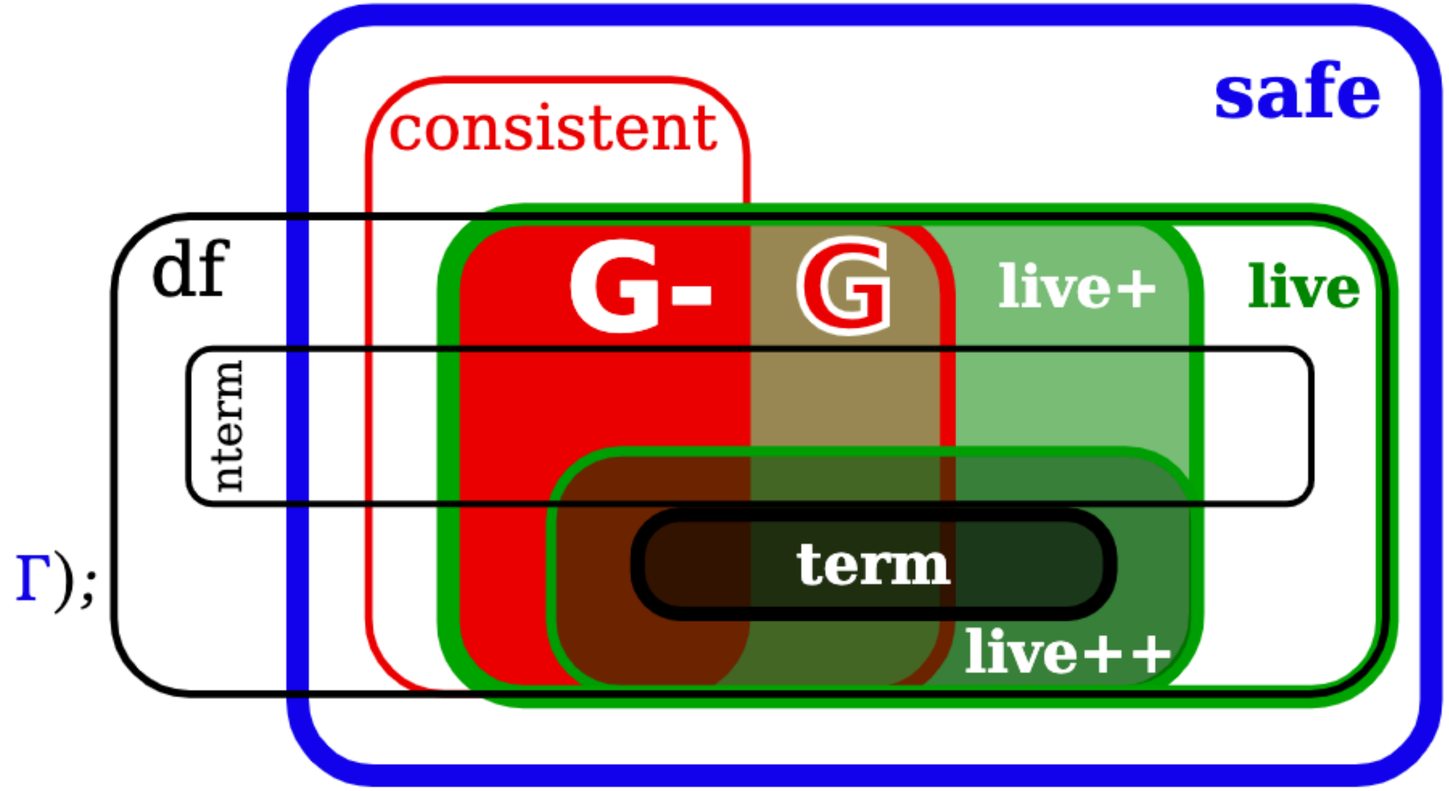
Bottom-up

- Promoting a use of model checking
- Typing rule for **bottom-up**

$$\frac{\Gamma' = \{s[\mathbf{p}] : S_{\mathbf{p}}\}_{\mathbf{p} \in I} \quad \varphi(\Gamma') \quad s \notin \Gamma \quad \Theta \cdot \Gamma, \Gamma' \vdash P}{\Theta \cdot \Gamma \vdash (\nu s : \Gamma')\, P}$$

End Point Projection and Merge Operators

- A Diagram from SY19



Proving Subject Reduction with Global Types

- Change consistency up-to-subtyping [DSHY'17]
- Use projection for the initial (starting) processes then use safety for running processes [HFDG'21, LNY'22]
- Use global types directly as a type of a multiparty session (precise synchronous multiparty session types) [GJPSY'19]
- Use **association** between the global and local types up-to-subtyping [BHYZ'22]

$$\frac{G \sqsubseteq_s \Gamma' \quad s \notin \Gamma \quad \Theta \cdot \Gamma, \Gamma' \vdash P}{\Theta \cdot \Gamma \vdash (\nu s{:}\Gamma')\, P}$$

Top-Down (Global Protocols)

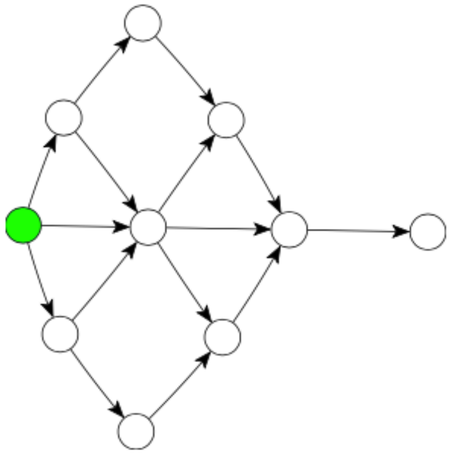
- **Global protocols help to visualise the whole interactions**
- **Correctness by contraction (safe, deadlock-free and live)**
- **Helpful to prove subject reductions**
- **Effective inductive projection [Tirore et at 23]**
- **Complete Projection of MPSTs with Automata [Li et al 23]**
- **Comositionariity [Gheri & NY OOPSLA 2023, Dezani et al (many)]**
- **Asynchronous subtyping: many other progress on sound algorithms [TACAS'23, Bocchi et al., ESOP'23, Li et al.,]**
- Users need to write a protocol
- Realisability is usually non-trivial, often difficult to produce a sound and complete set of deadlock-free and live local behaviours
- Global types might be getting complicated (Choreography Automata, Swams, Refinements, Timers, etc).

Bottom Up (Local Protocols)

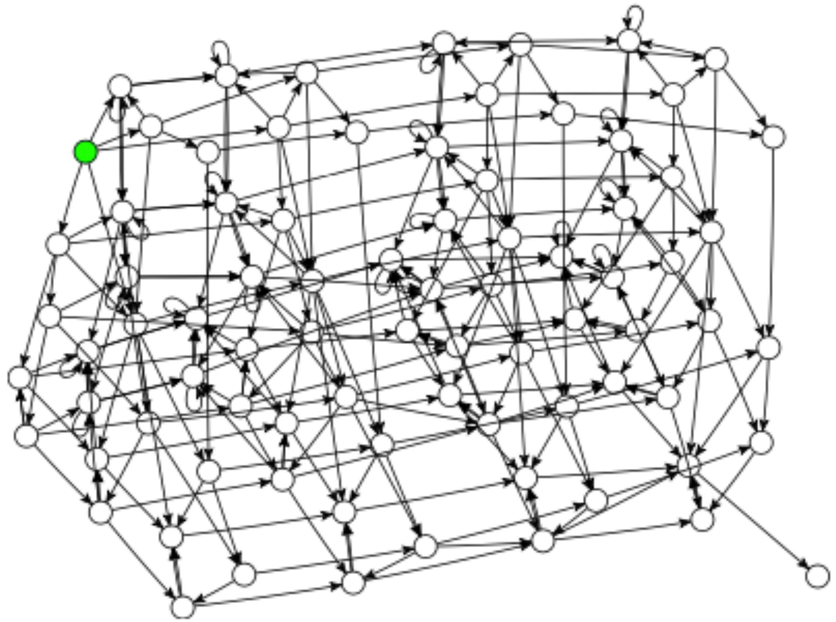
- **Sound and Complete w.r.t. the Properties by construction**
- **No need to write a global protocol**
- **Easy to extend the syntax of local types**
- State Explosion Problem (off-the-shelf model checking tools)
- Extraction of behavioural types from programs is very tedious and hard
- Gap between type-level properties and process behaviours (liveness)
- Undecidable for Asynchronous MPST (Communicating Automata) —limit to synchrony (no buffer)
- Need to check a combination of safety and deadlock-freedom (or other property)

Why model-checking tool (mCRL2)?

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



No Failure



with Failure

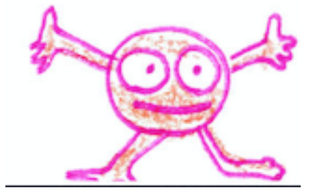
Future Directions

- Applications
  - Beyond Academic Trials, Industry production level applications
  - Benchmark Sets (Distributed Systems)
  - Failures / Recovery
  - Cyber Physical Systems, Quantum Computing, (Federate) ML, HPC, …
- Foundations
  - Typed Behavioural Semantics
  - Relationship with (Concurrent ?) Effect Systems





Thank you! Discussions?



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