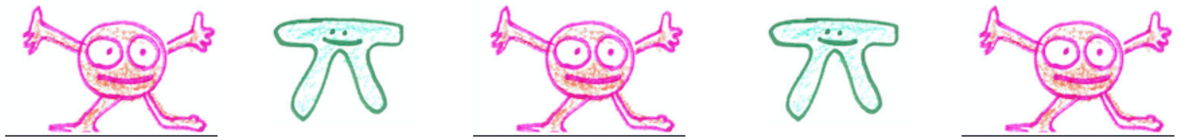
Open Problems in Session Types

Nobuko Yoshida
University of Oxford

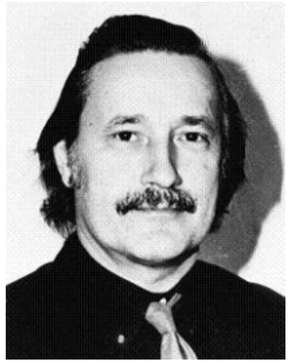
Open Problems in Concurrency Theory
27 June 2023



UNIVERSITY OF OXFORD

Types and Systems

from Conventional to Concurrent/Distributed Programming



1916-1975

- **Christopher Strachey** (sequential computation)
  - ***Types*** = **abstract computation (data types, polymorphism)**
  - **Structured programming = High-level programming**

Fundamental Concepts of Programming Languages

- **Session types** (concurrency & communication)
  - Structured programming = ***protocols***

Communications are Ubiquitous

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

microservices

Problems: Concurrency Bugs

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

Uber

The Go Gopher

Do not communicate by sharing memory; share memory by communicating

– Go Philosophy

Problems: Concurrency Bugs

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

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

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

deadlock

channel errors

The Go Gopher

Problems: Concurrency Bugs

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

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

Session Types

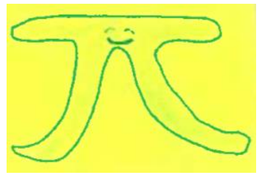
- Prevent concurrency bugs.
- Can abstract, implement and manage communications as **Protocols.**
- **Clean, Cheap** and **Retrofittable.**

Why Session Types, Why Now?

Significant academic and industry interests via fundamental breakthroughs

Milner, Honda, NY

Binary Session Types — ESOP'98



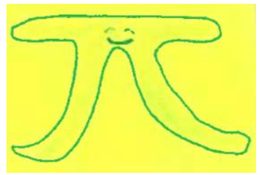
↓

Joined W3C Standardization — 2002

Why Session Types, Why Now?

Significant academic and industry interests via fundamental breakthroughs

Binary Session Types — ESOP'98



↓

Joined W3C Standardization — 2002

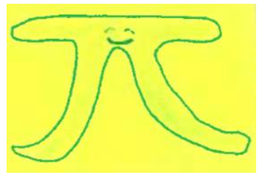
↓

Multiparty Session Types — POPL' 08

Why Session Types, Why Now?

Significant academic and industry interests via fundamental breakthroughs

Binary Session Types — ESOP'98



↓

Joined W3C Standardization — 2002

↓

Multiparty Session Types — POPL' 08

↓

largest open source company in the world

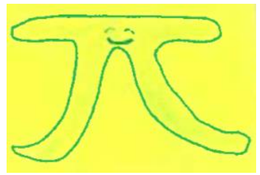
Red Hat

Scribble

Why Session Types, Why Now?

Significant academic and industry interests via fundamental breakthroughs

Binary Session Types — ESOP'98



Joined W3C Standardization — 2002

Multiparty Session Types — POPL' 08

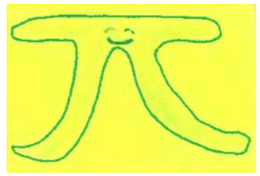
25M US$ large cyber infrastructures

middleware standardisations

Why Session Types, Why Now?

Significant academic and industry interests via fundamental breakthroughs

Binary Session Types — ESOP'98



Joined W3C Standardization — 2002

Multiparty Session Types — POPL' 08







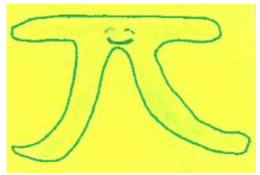



Why Session Types, Why Now?

Significant academic and industry interests via fundamental breakthroughs

ETAPS Test Time Award 2019

Binary Session Types — ESOP'98



Joined W3C Standardization — 2002

POPL Influential Paper Award 2018

Multiparty Session Types — POPL' 08

Foundations of Session Types

Three Major Linkages

- The Original Session Types by Honda, Vasconcelos and Kubo 98
- Linear Logic based Session Types

  Caires & Pfenning [2010] & Wadler [2012]

- Automata and Session Types
  - Danielou and NY 2012, Lange, Tuosto, NY 2015, Zavattaro et al., 2017, …
  - Model Checking (mCRL2)

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

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

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

Pi-Calculus vs Lambda-Calculus
Kohei Honda [1995]

λ in π

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

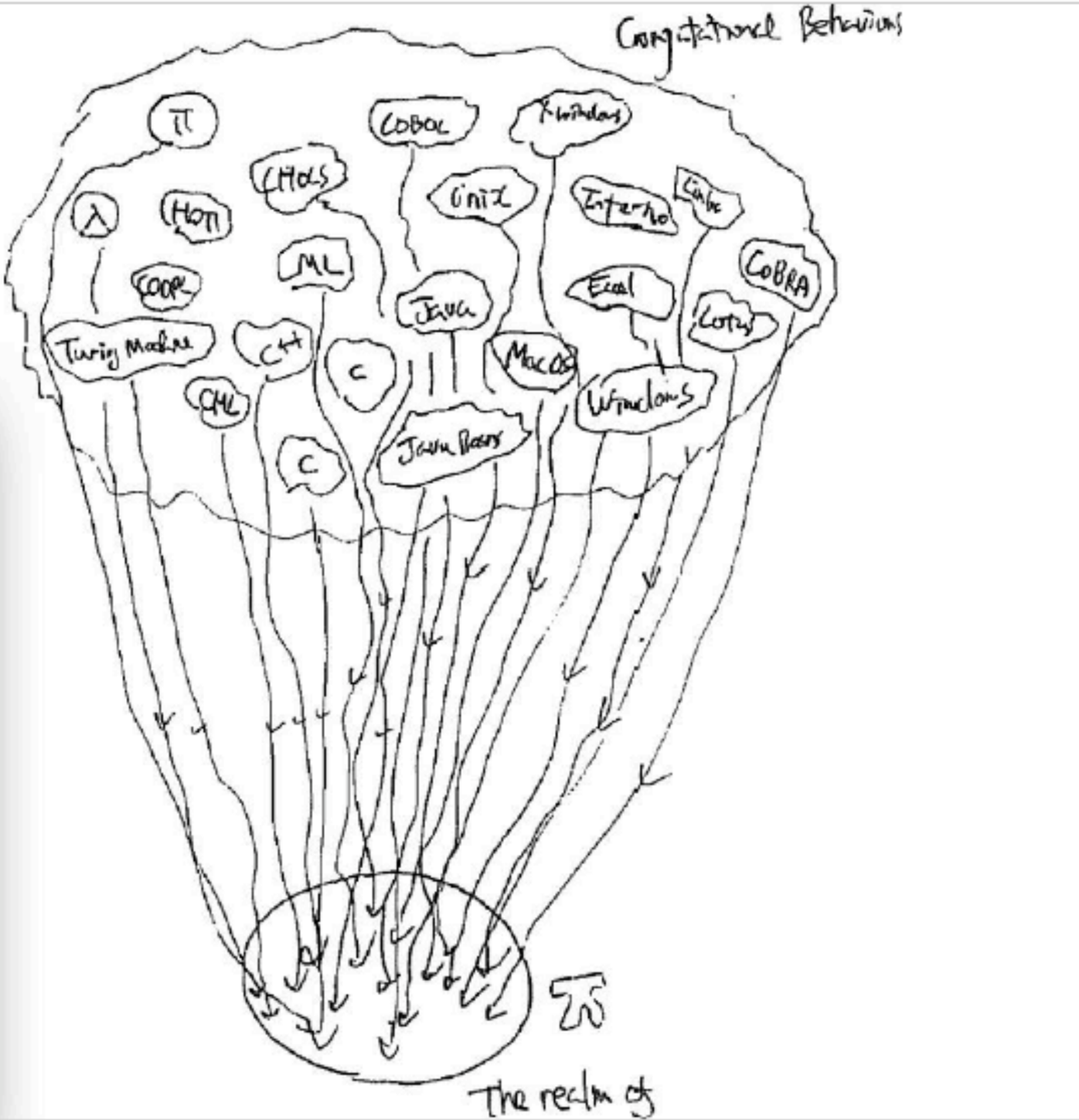
$[\![\lambda x.M]\!]_u \stackrel{def}{=} u(xu').[\![M]\!]_{u'}.$

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

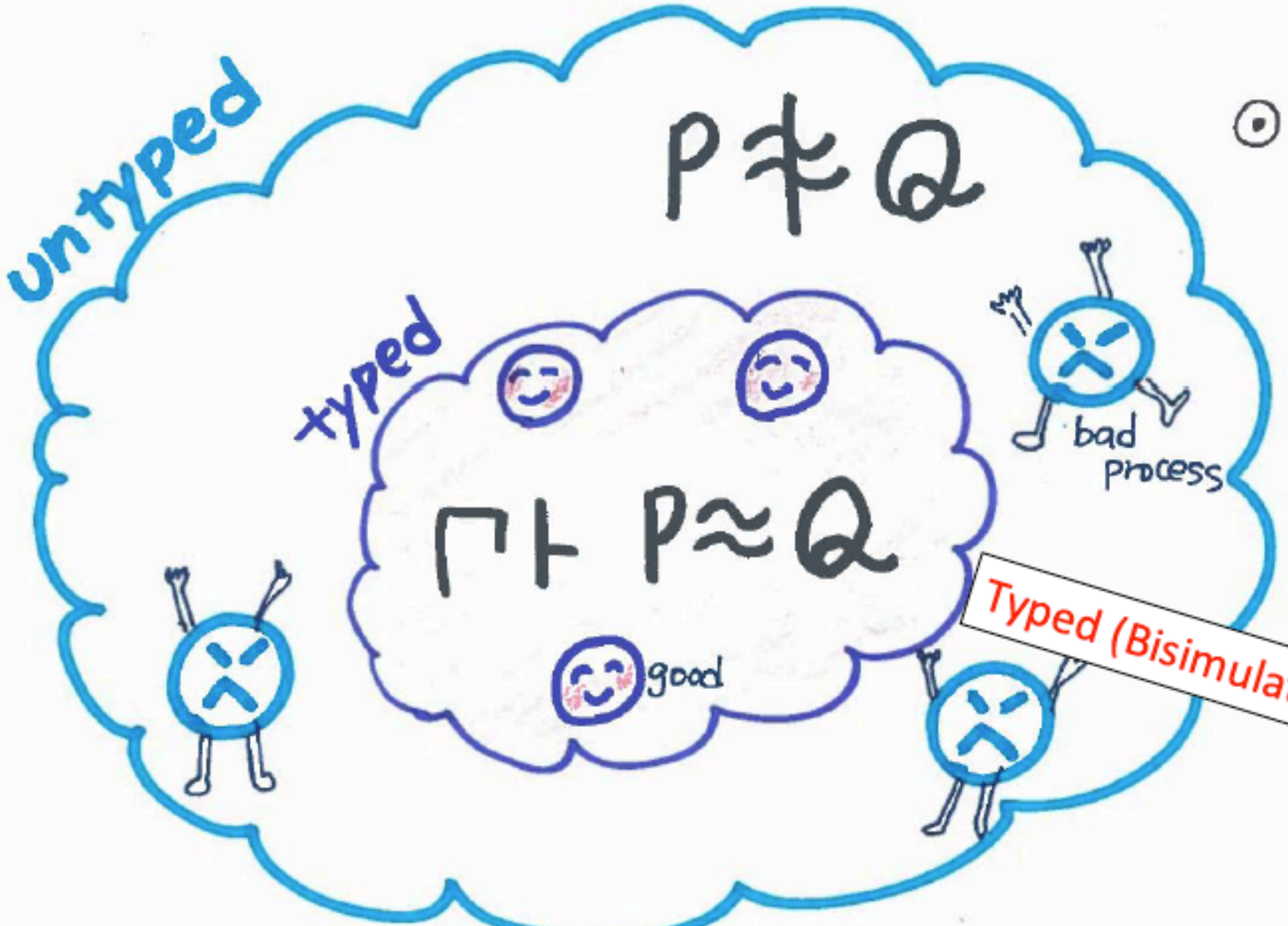
with $[\![x=N]\!] \stackrel{def}{=} !x(u').[\![N]\!]_{u'}.$

Pi-Calculus
Kohei Honda [1995]

- λ-calculus [MPW89, Milner90, Milner92, ...]
- Concurrent Object [Walker81]
- 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, ...



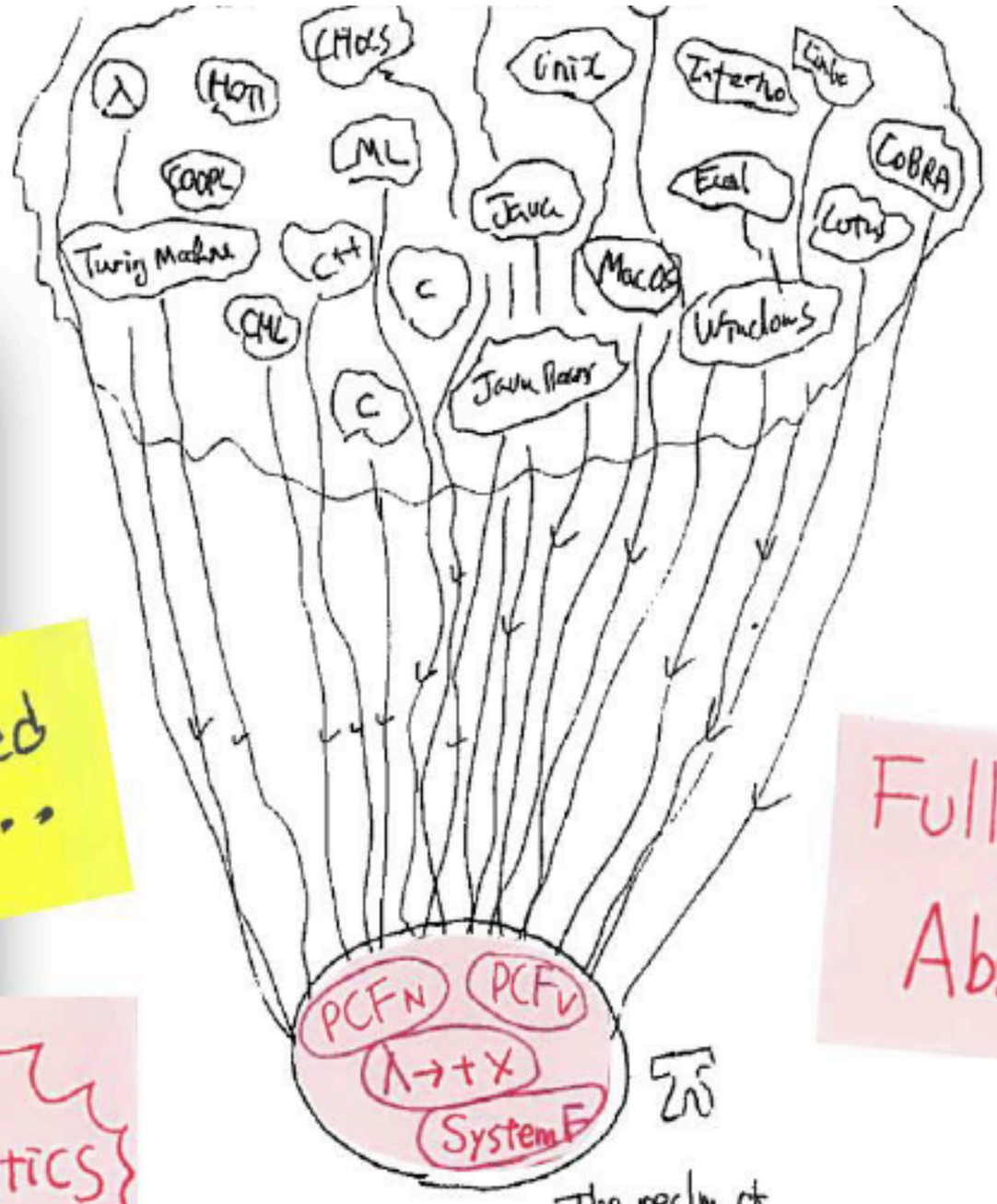
- Correctness of Encoding
- Limit environment Γ
  ⇒ Equate more processes
- Compositional

Typed (Bisimulation) Semantics

- 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 ...

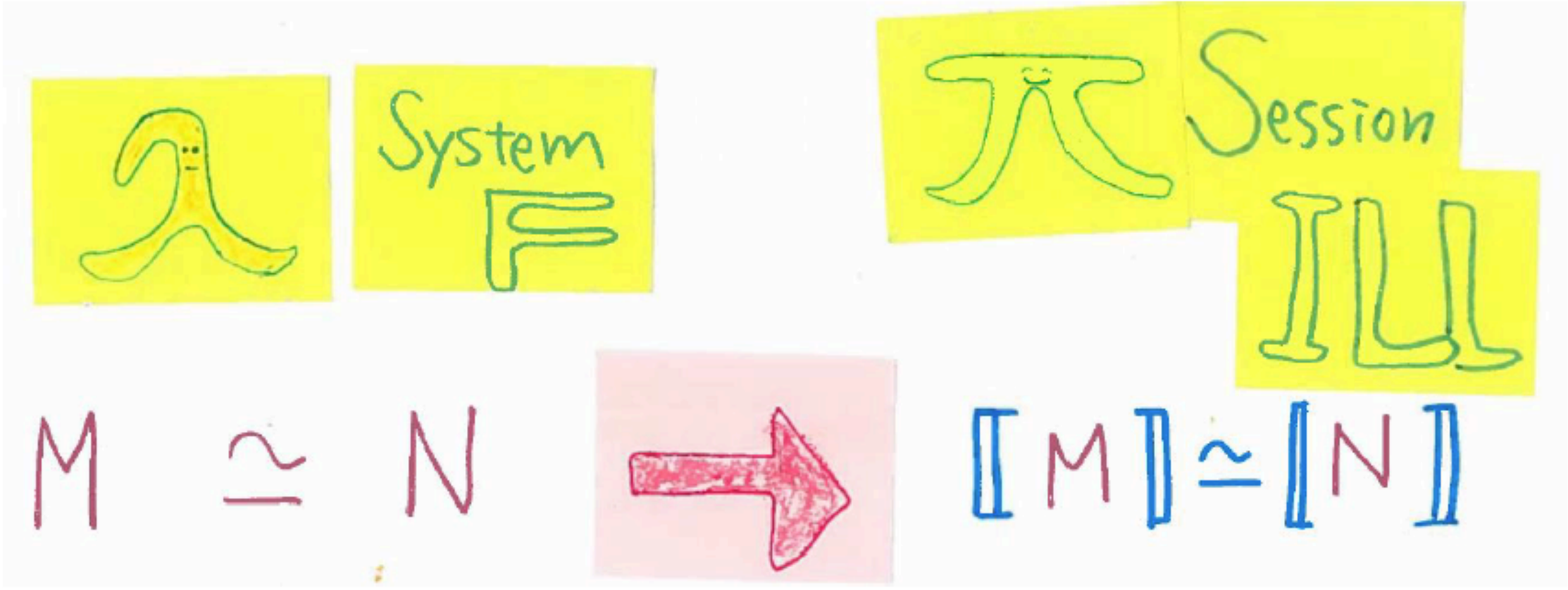
Game Semantics



The realm of Name Passing Interactions

Fully Abstract

Linear Logic-Based Session Types [Toninho & NY 2021]



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

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

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

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

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

System F

Session

$M \simeq N$

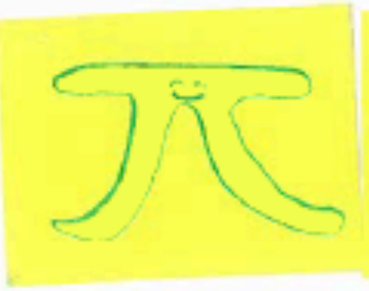
$[\![M]\!] \simeq [\![N]\!]$

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

$P \simeq Q$

Reverse

System F



Session

$M \simeq N$

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

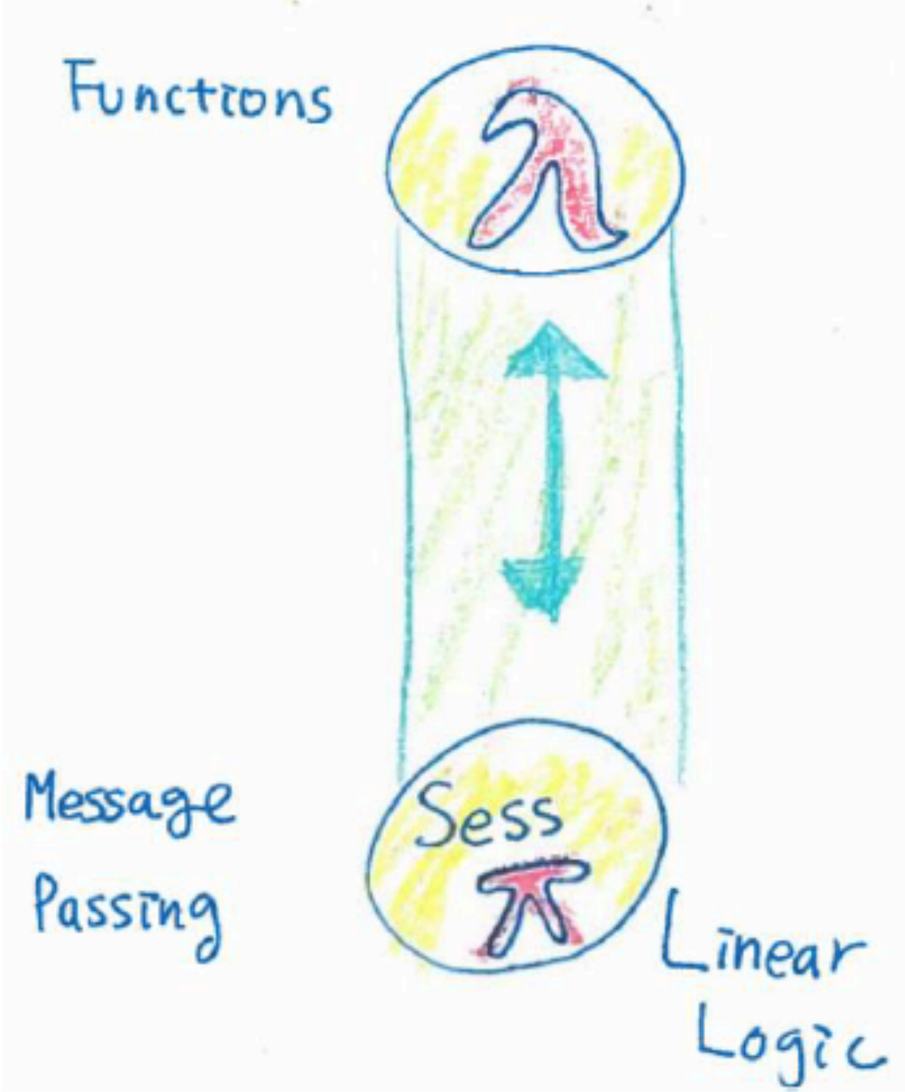
$[\![P]\!] \simeq [\![Q]\!]$

$P \simeq Q$

Reverse

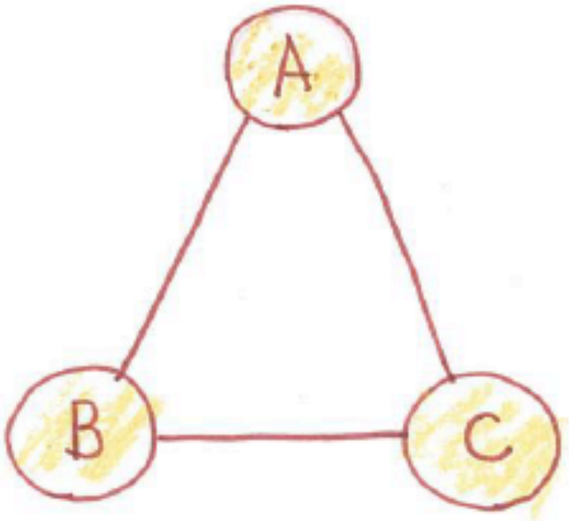
New

Summary

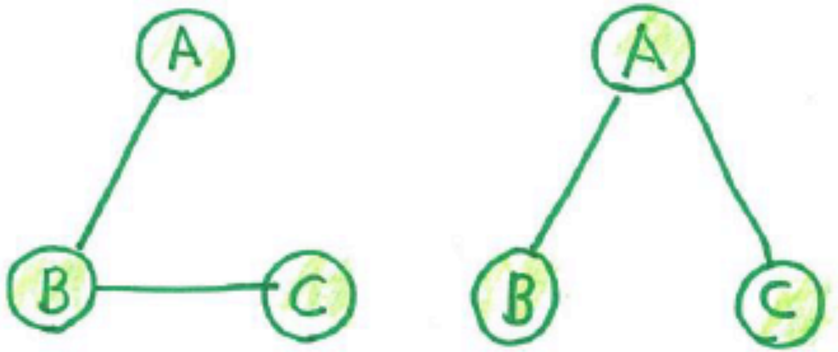


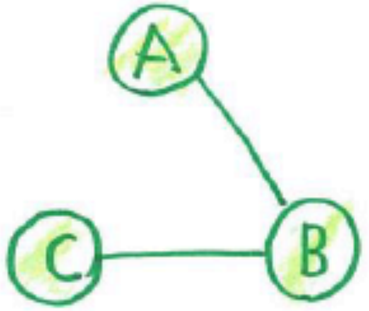
Distributability & Interconnectability

[Toninho & NY 19, Peters, Nestmann & Schmitt 23]



Multi Party



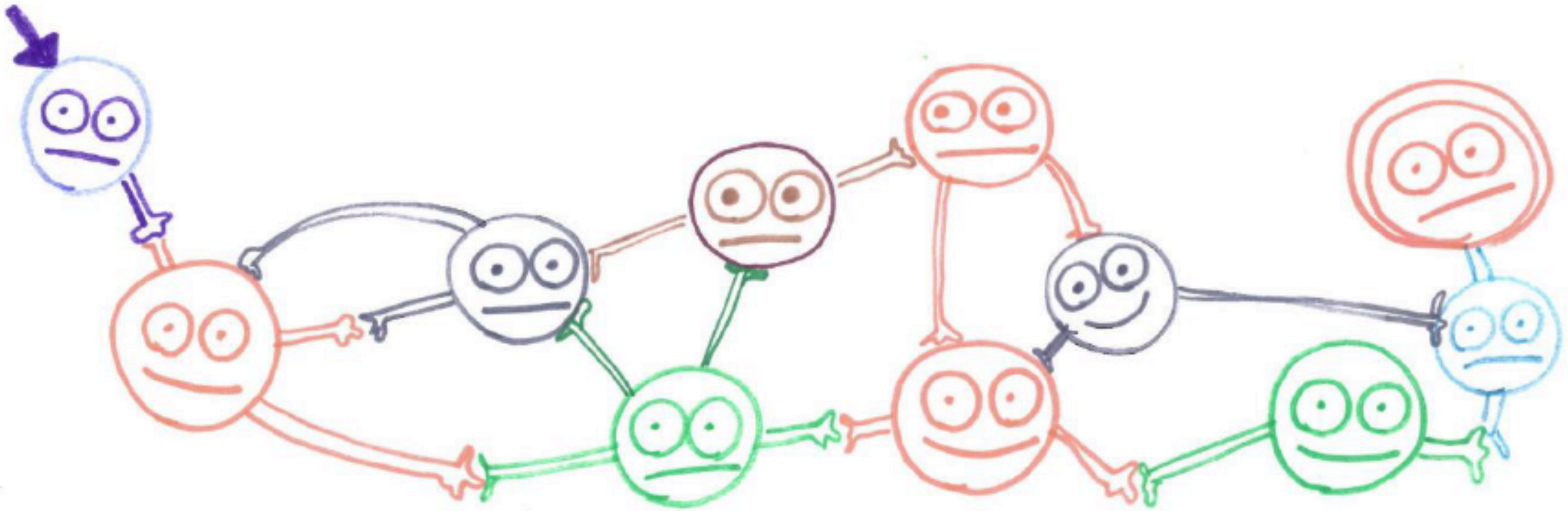


Linear Logical Session

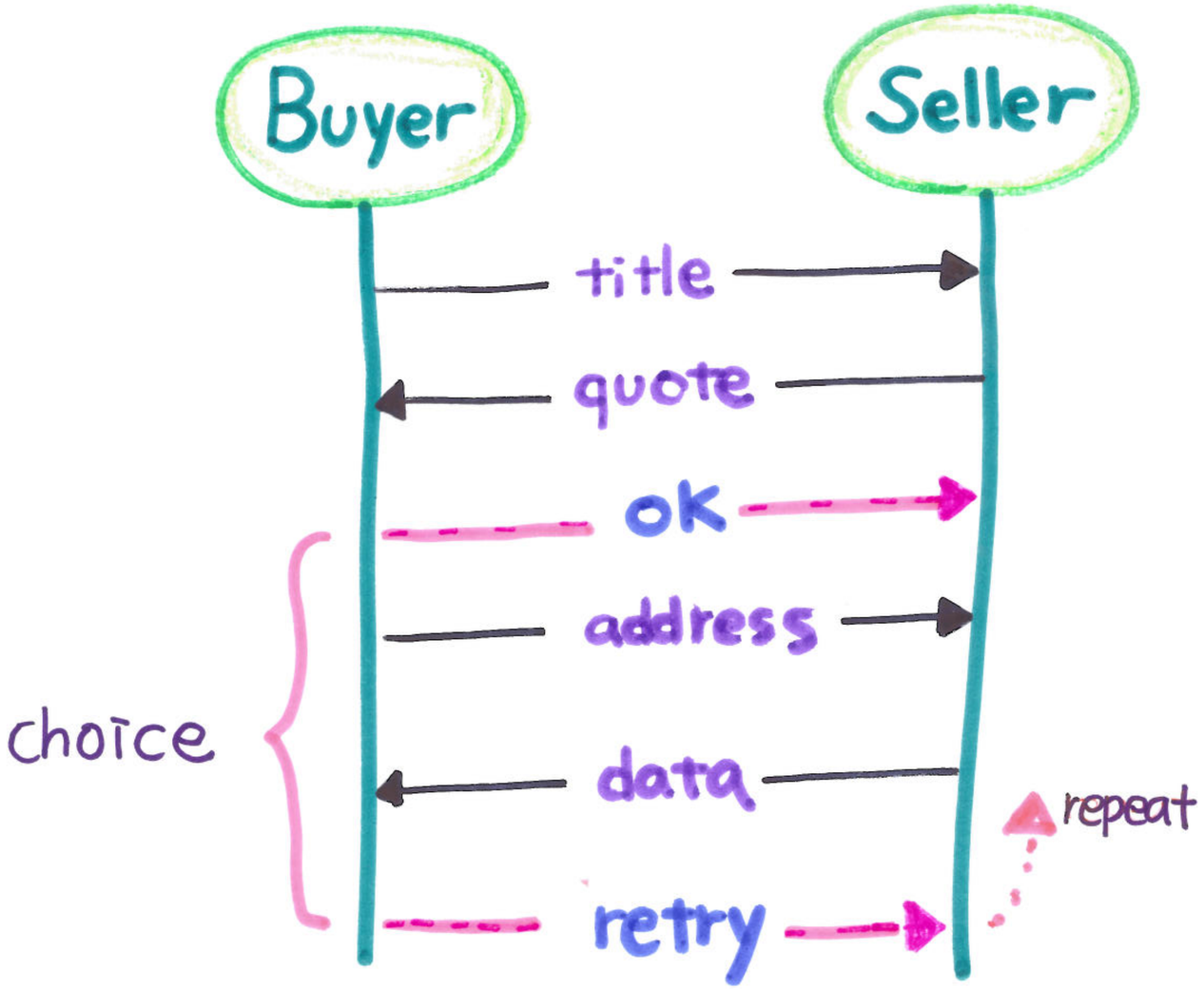
cf. [1995] Specification Structures and Proposition as Types

Communicating Automata

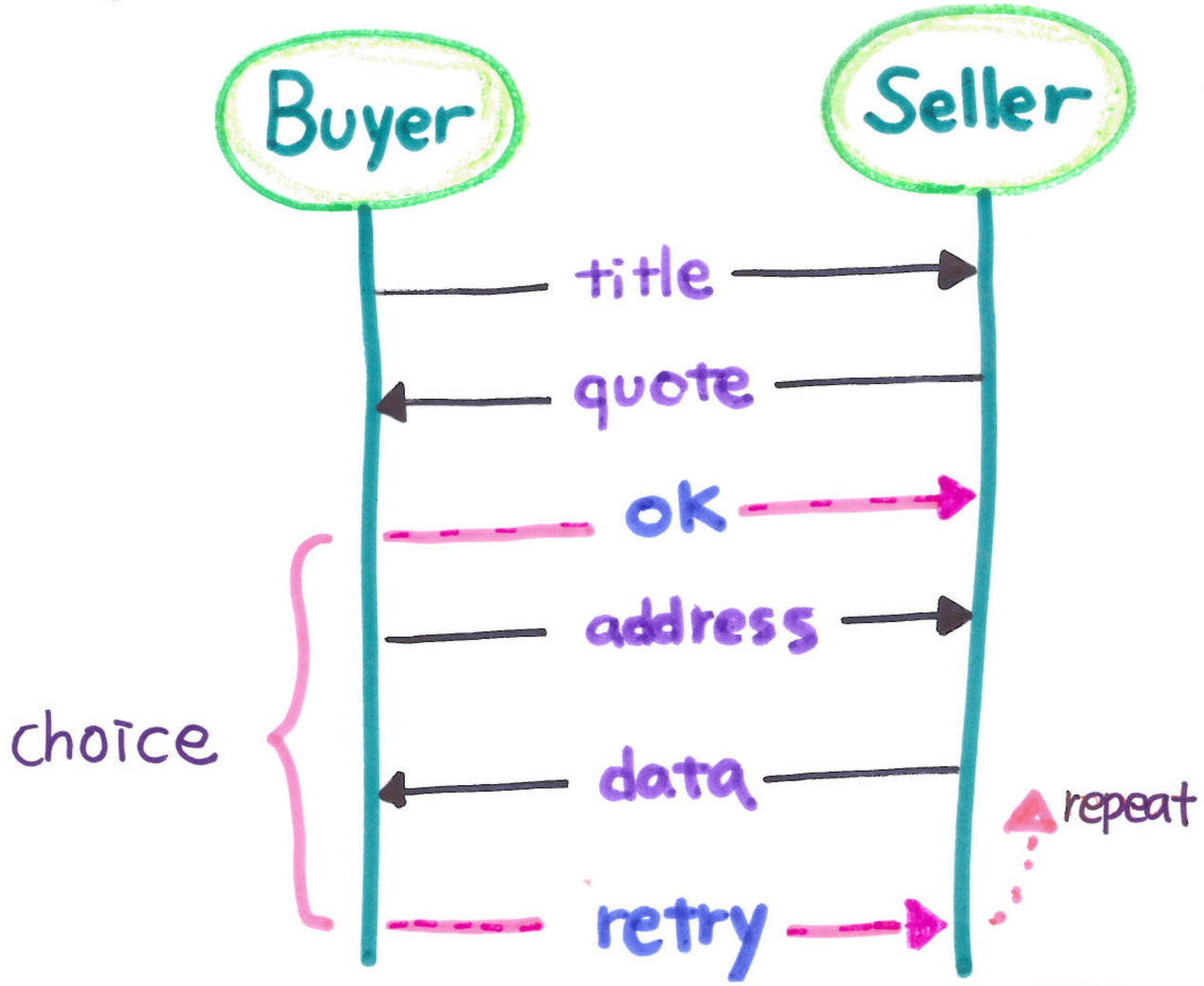
Linkage with Session Types



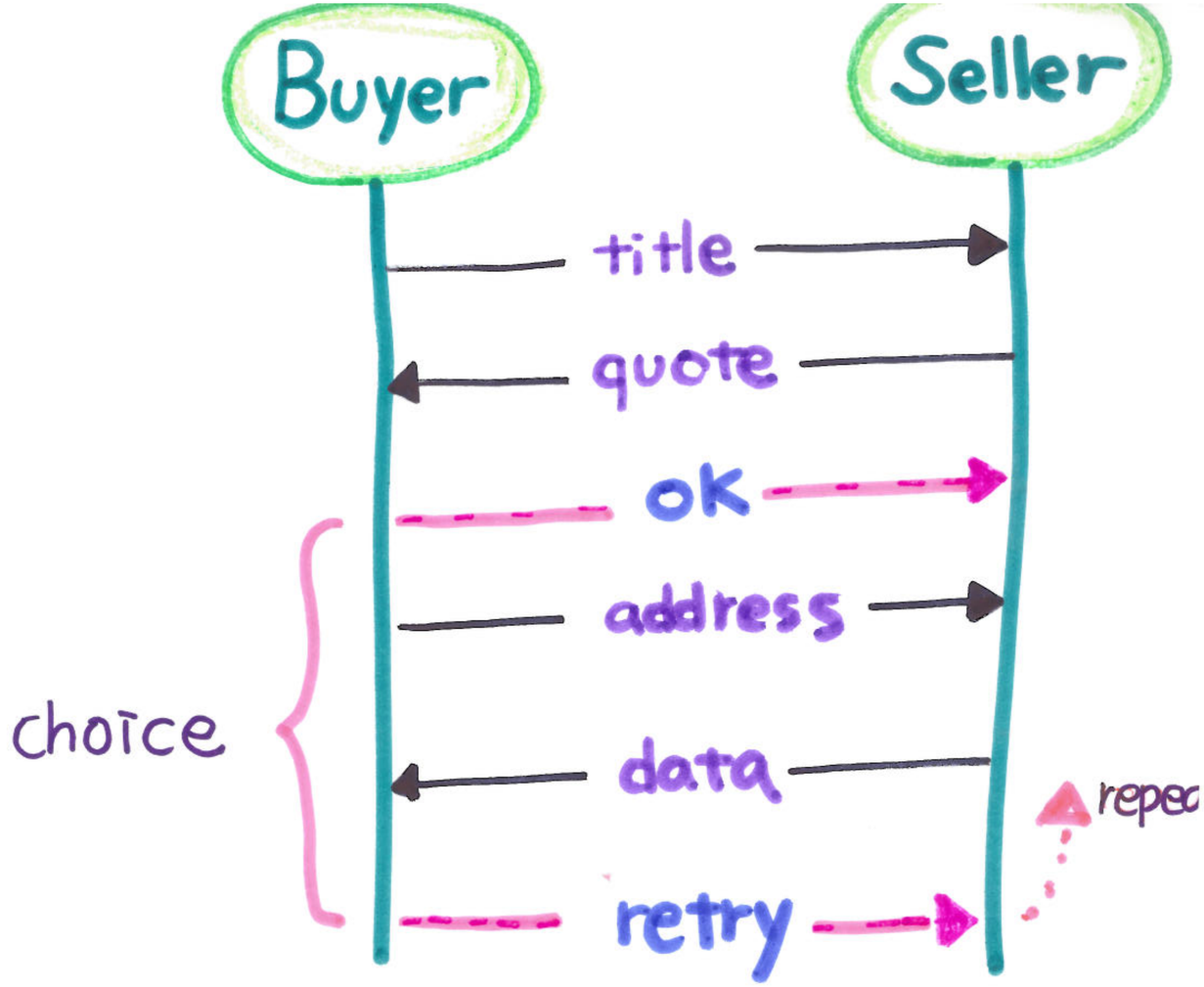
Binary Session Types: Buyer - Seller Protocol



Binary Session Types: Buyer - Seller Protocol



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



P has T

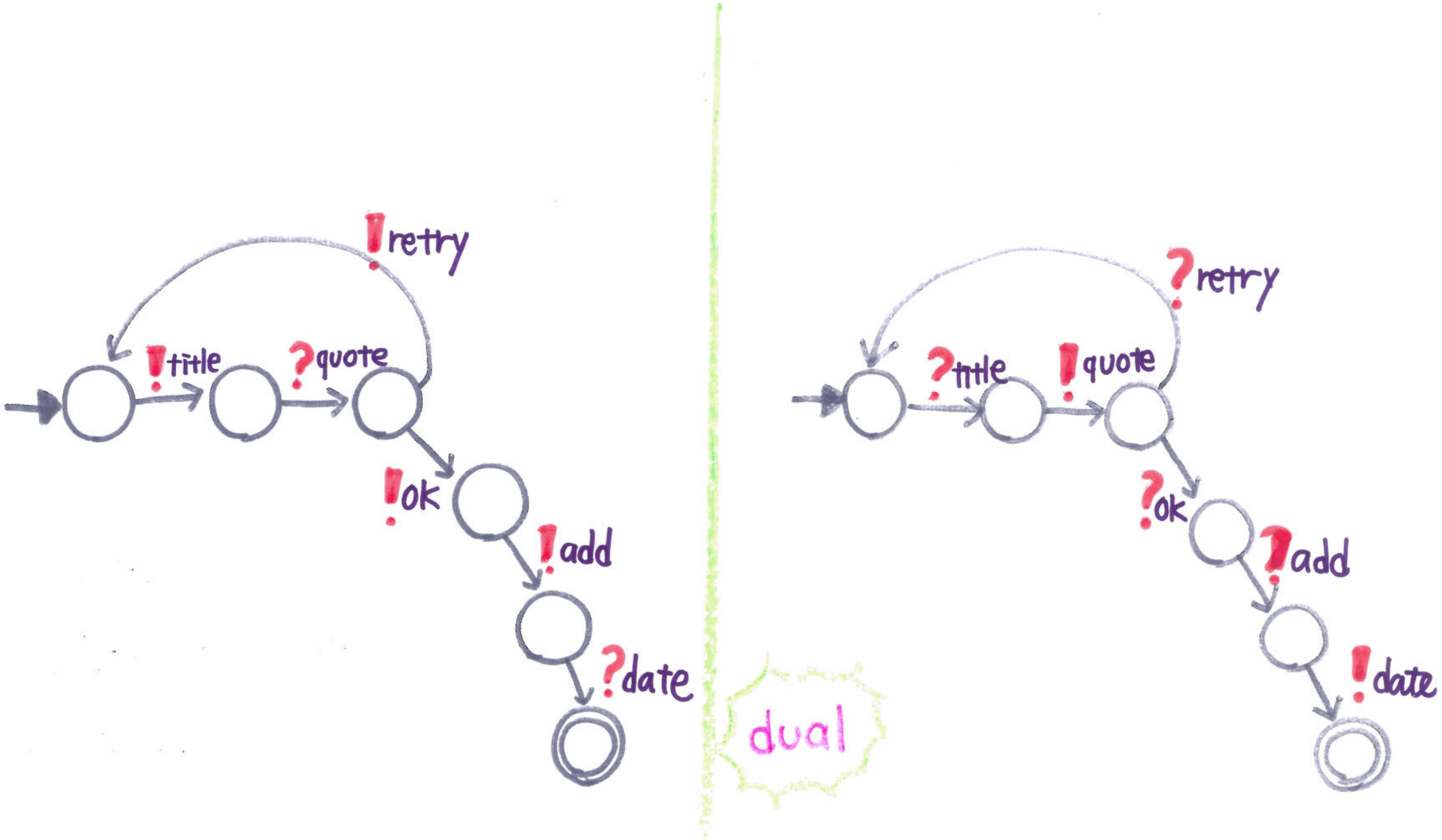
Q has $\overline{T}$ dual

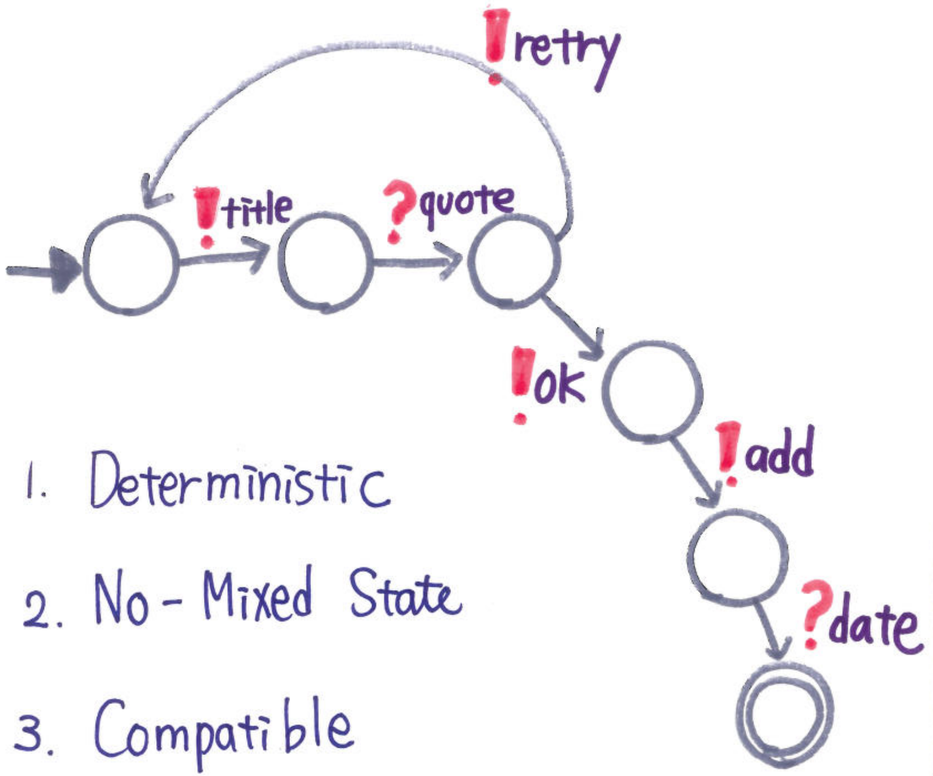
P | Q typable

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

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

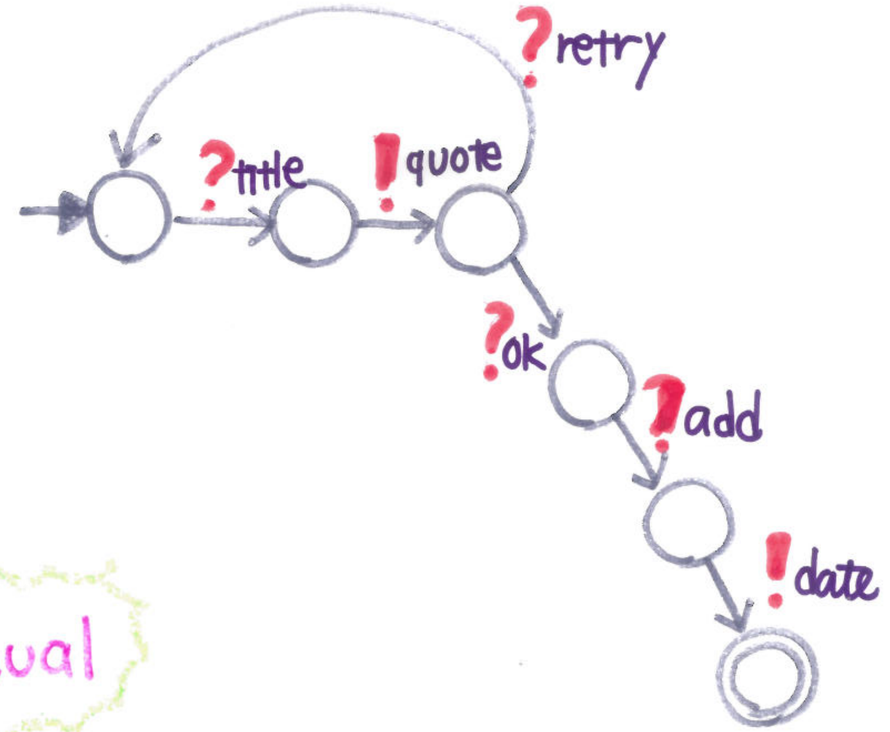
Communicating Automata [1980s] [Brand & Zafiropulo '83]





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

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

Ring Protocol

Example

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

Ring Protocol

Example

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

Ring Protocol

Example

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

Ring Protocol

Example

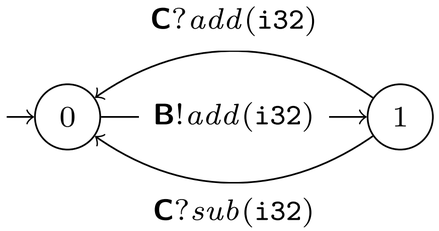
$$\mathsf{G} = \mu \mathbf{t}.\mathbf{A} \to \mathbf{B} : \left\{ add(\texttt{i32}).\mathbf{B} \to \mathbf{C} : \left\{ \begin{array}{l} add(\texttt{i32}).\mathbf{C} \to \mathbf{A} : \{add(\texttt{i32}).\mathbf{t}\} \\ sub(\texttt{i32}).\mathbf{C} \to \mathbf{A} : \{sub(\texttt{i32}).\mathbf{t}\} \end{array}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

Ring Protocol

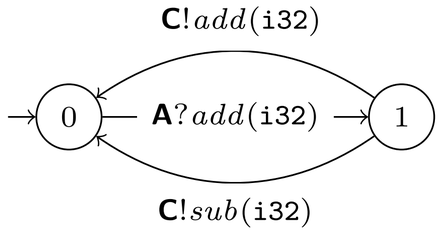
Example

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

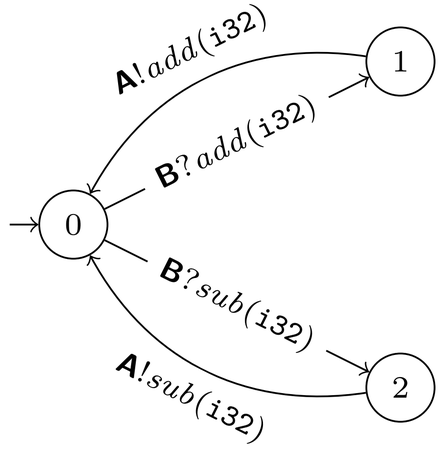
PROJECTION



PROJECTION



PROJECTION

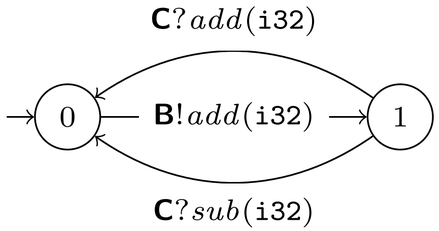


Ring Protocol

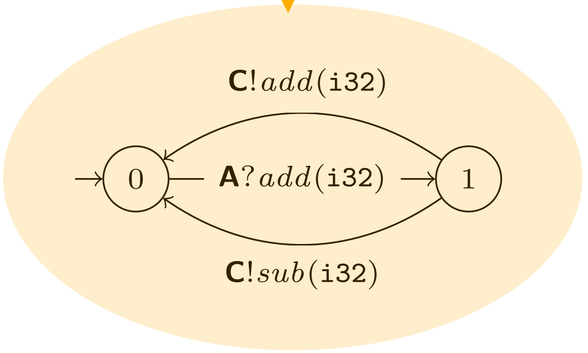
Example

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

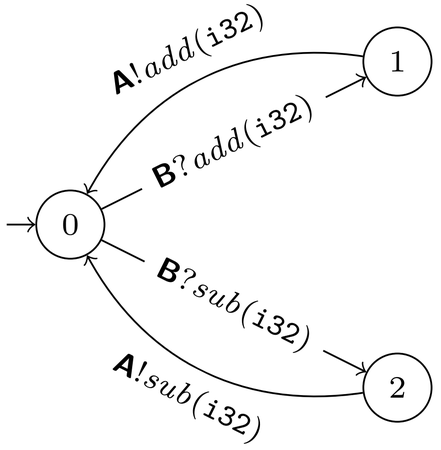
PROJECTION



PROJECTION



PROJECTION



Challenge

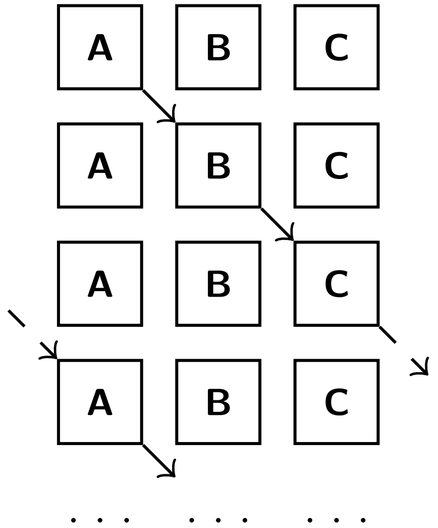
Asynchronous Orderings

- Global types are inherently synchronous

Challenge

Asynchronous Orderings

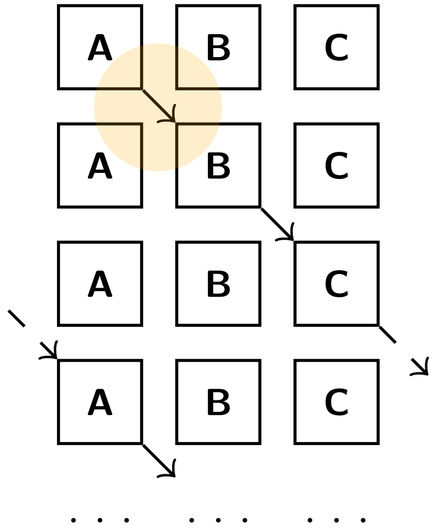
- Global types are inherently synchronous
  - Projection provides only one possible ordering



Challenge

Asynchronous Orderings

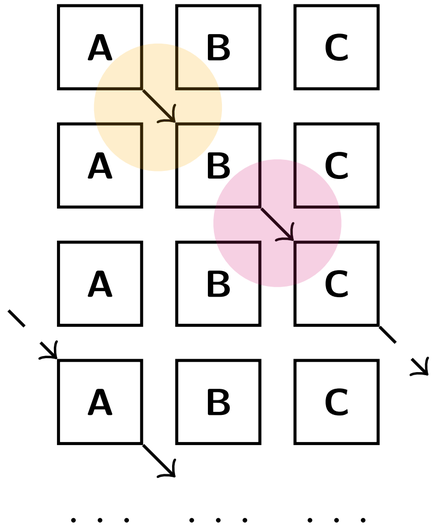
- Global types are inherently synchronous
  - Projection provides only one possible ordering



Challenge

Asynchronous Orderings

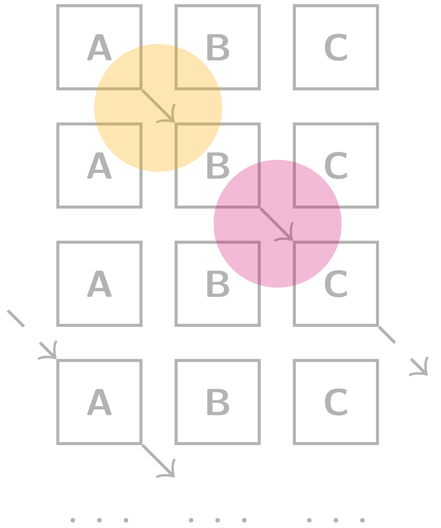
- Global types are inherently synchronous
  - Projection provides only one possible ordering

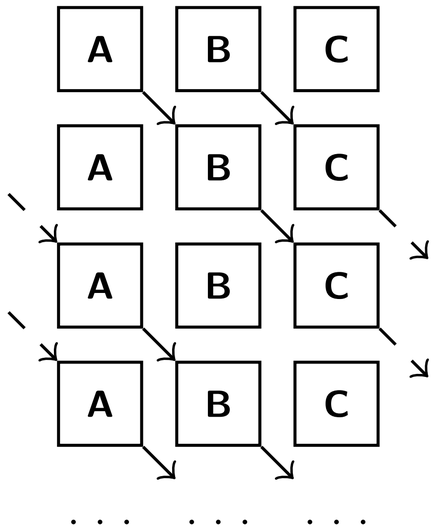


Challenge

Asynchronous Orderings

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

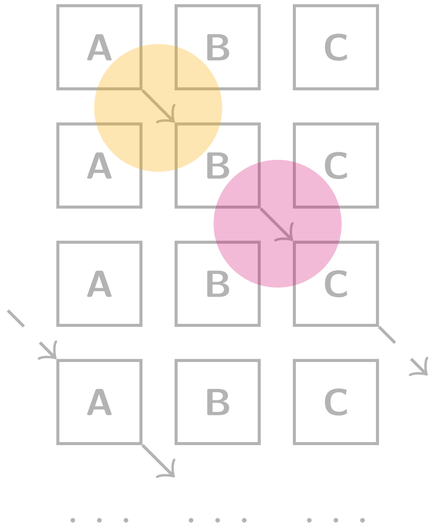


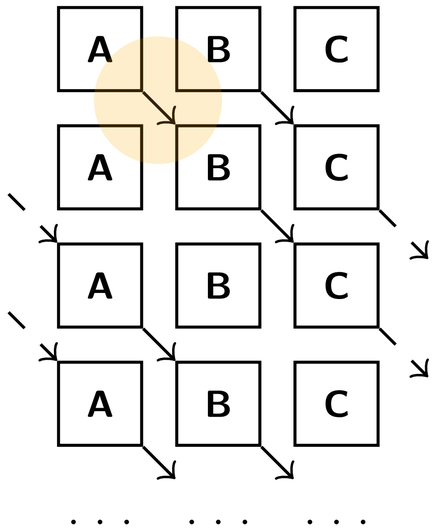


Challenge

Asynchronous Orderings

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

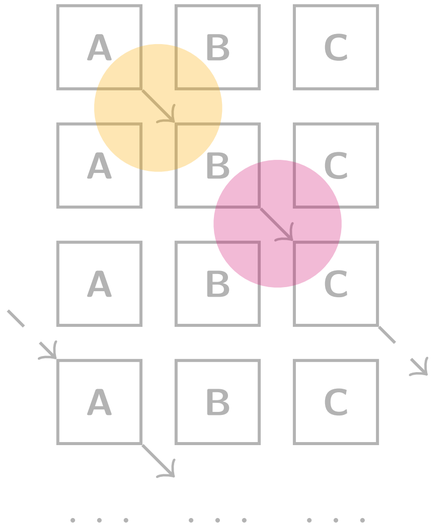


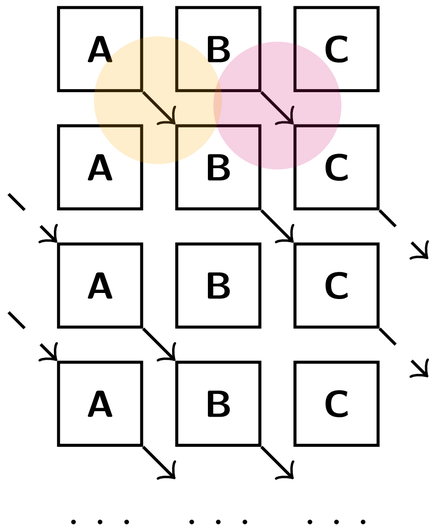


Challenge

Asynchronous Orderings

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

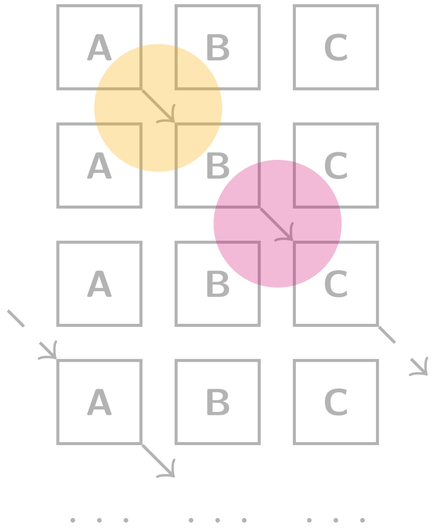


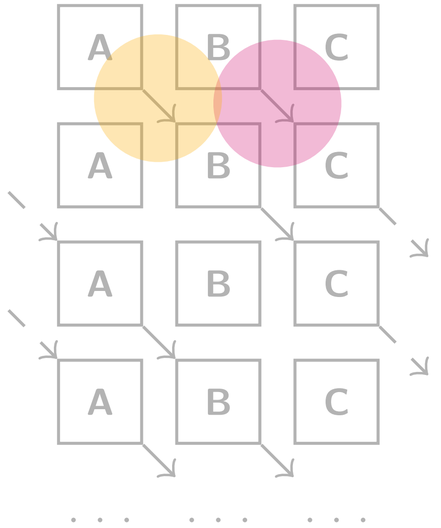


Challenge

Asynchronous Orderings

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

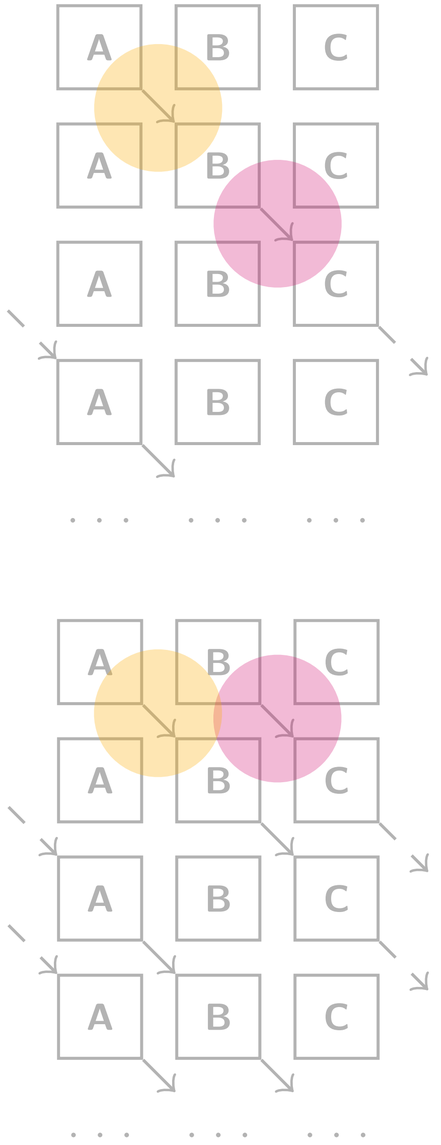




Challenge

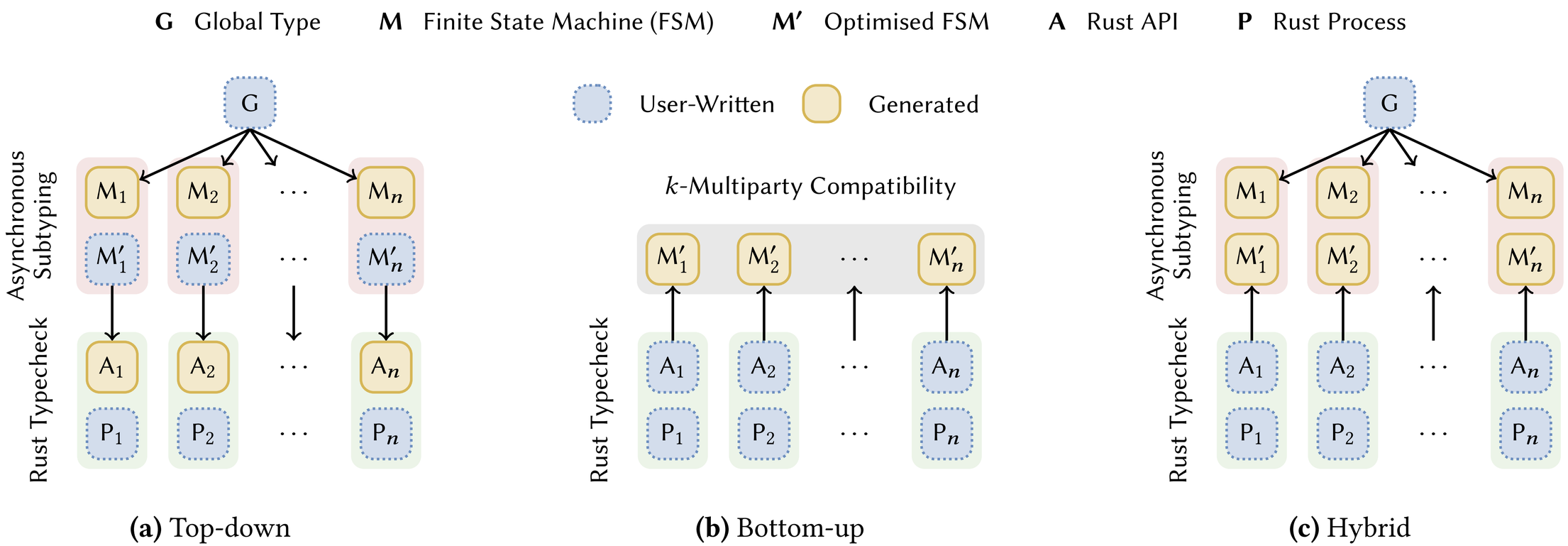
Asynchronous Orderings

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



Rumpsteak Framework (Rust)

Three Approaches



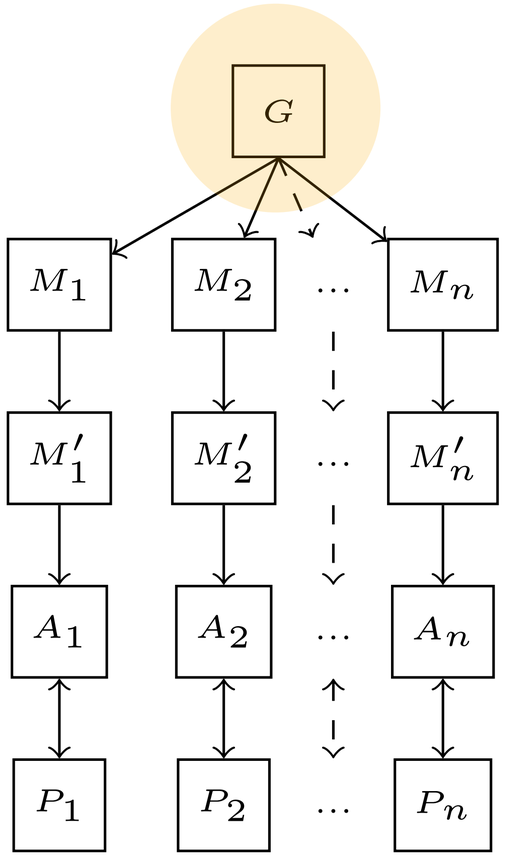
(a) Top-down

(b) Bottom-up

(c) Hybrid

Workflow

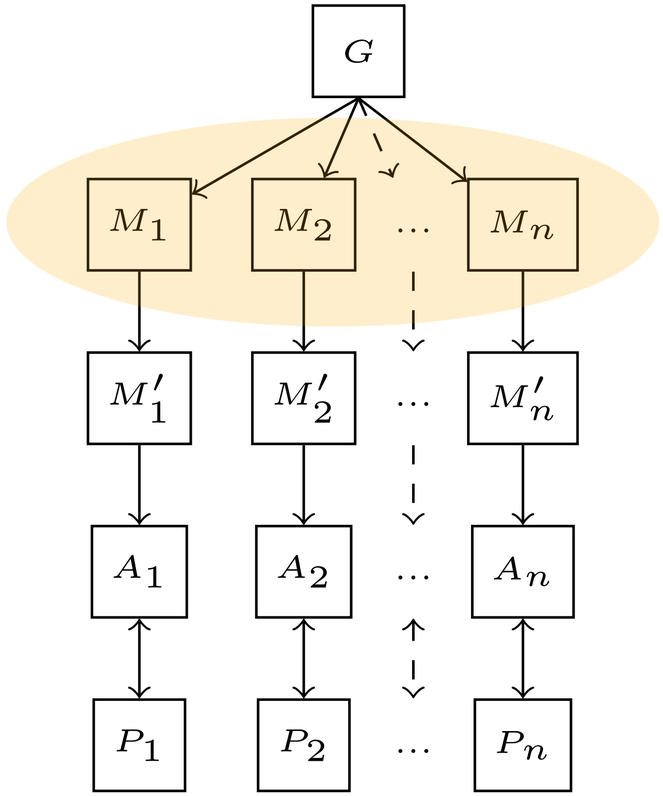
Top-Down Approach



Global Type

Workflow

Top-Down Approach

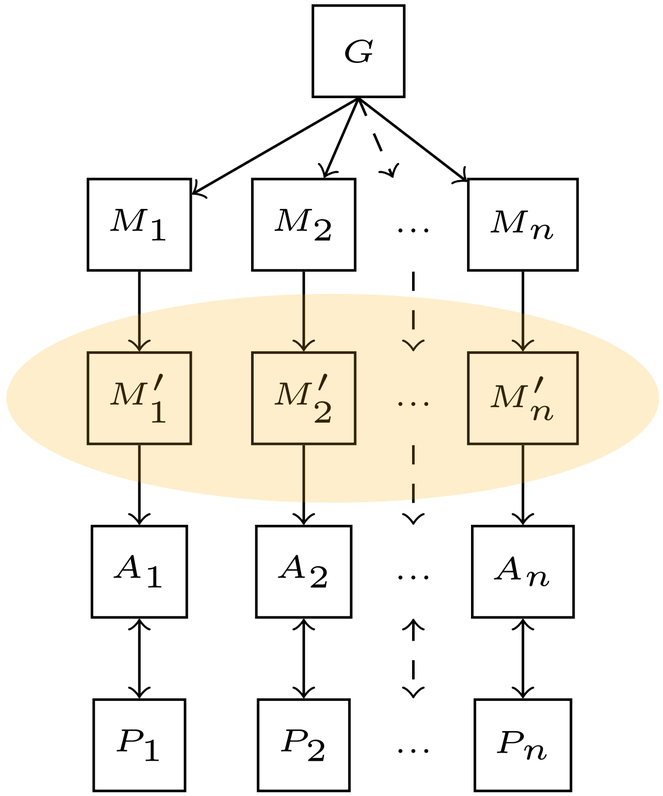


Global Type

Projected FSM

Workflow

Top-Down Approach



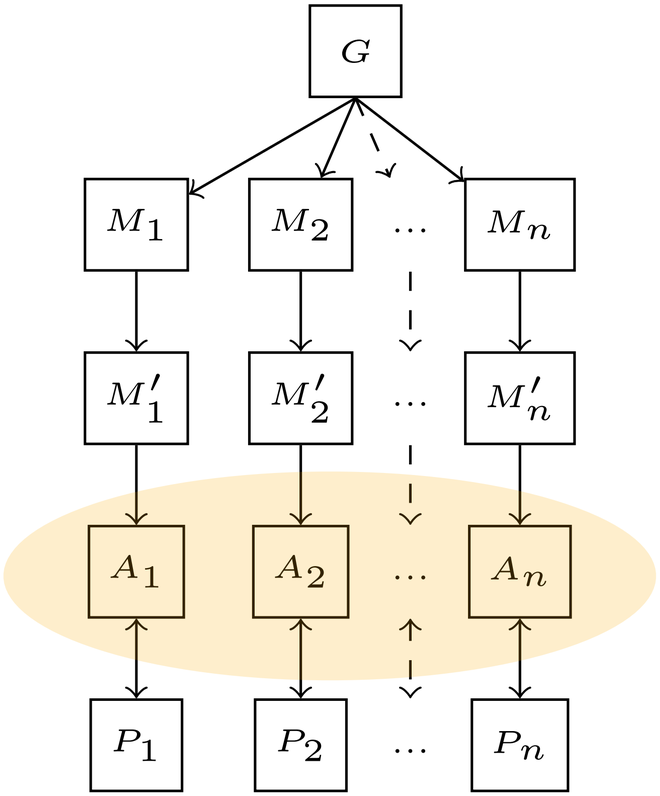
Global Type

Projected FSM

Optimised FSM

Workflow

Top-Down Approach



Global Type

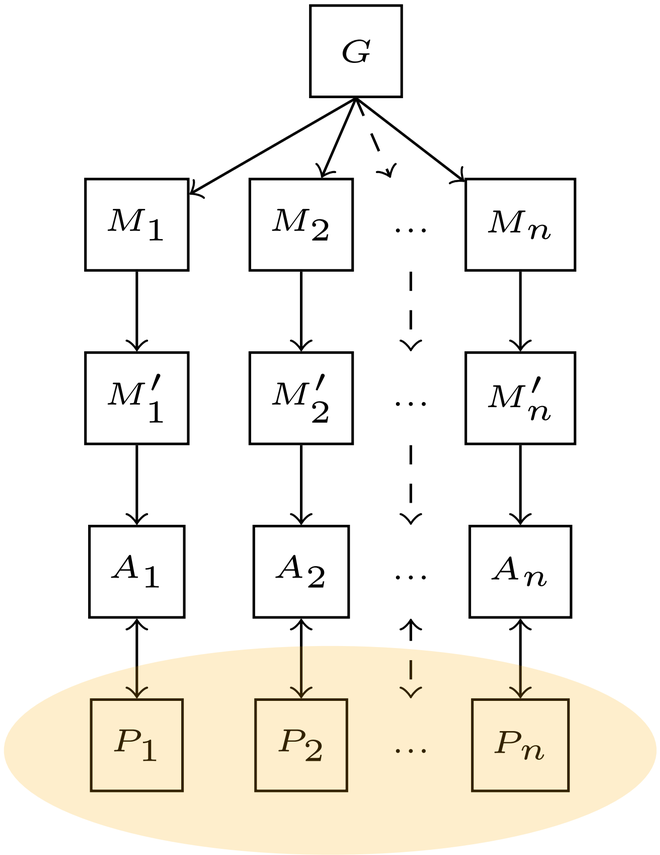
Projected FSM

Optimised FSM

Rust API

Workflow

Top-Down Approach



Global Type

Projected FSM

Optimised FSM

Rust API

Rust Implementation

Ring Protocol

Example

Global Type

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

Ring Protocol

Example

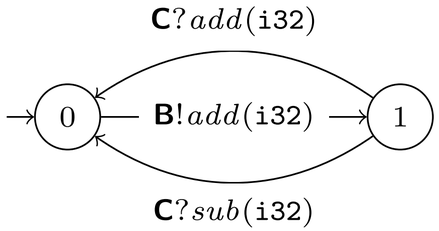
$$\mathsf{G} = \mu \mathbf{t}.\mathbf{A} \to \mathbf{B} : \left\{ add(\texttt{i32}).\mathbf{B} \to \mathbf{C} : \left\{ \begin{array}{l} add(\texttt{i32}).\mathbf{C} \to \mathbf{A} : \{add(\texttt{i32}).\mathbf{t}\} \\ sub(\texttt{i32}).\mathbf{C} \to \mathbf{A} : \{sub(\texttt{i32}).\mathbf{t}\} \end{array}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

Ring Protocol

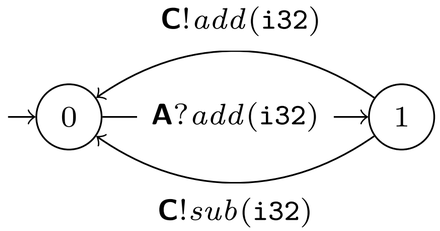
Example

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

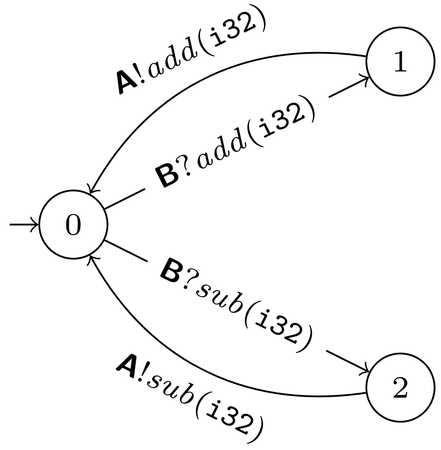
PROJECTION



PROJECTION

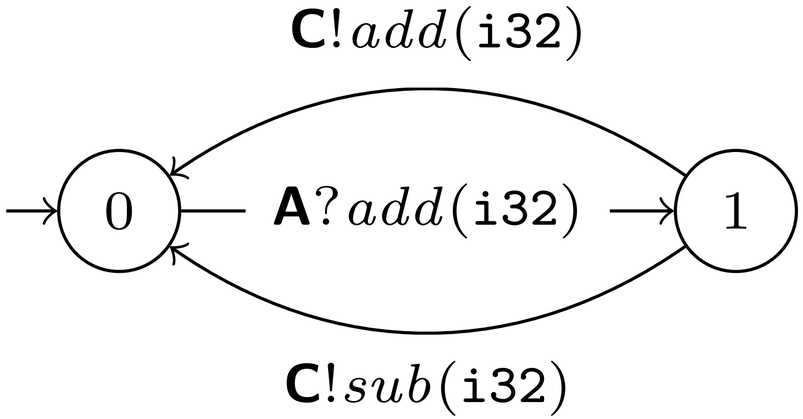


PROJECTION



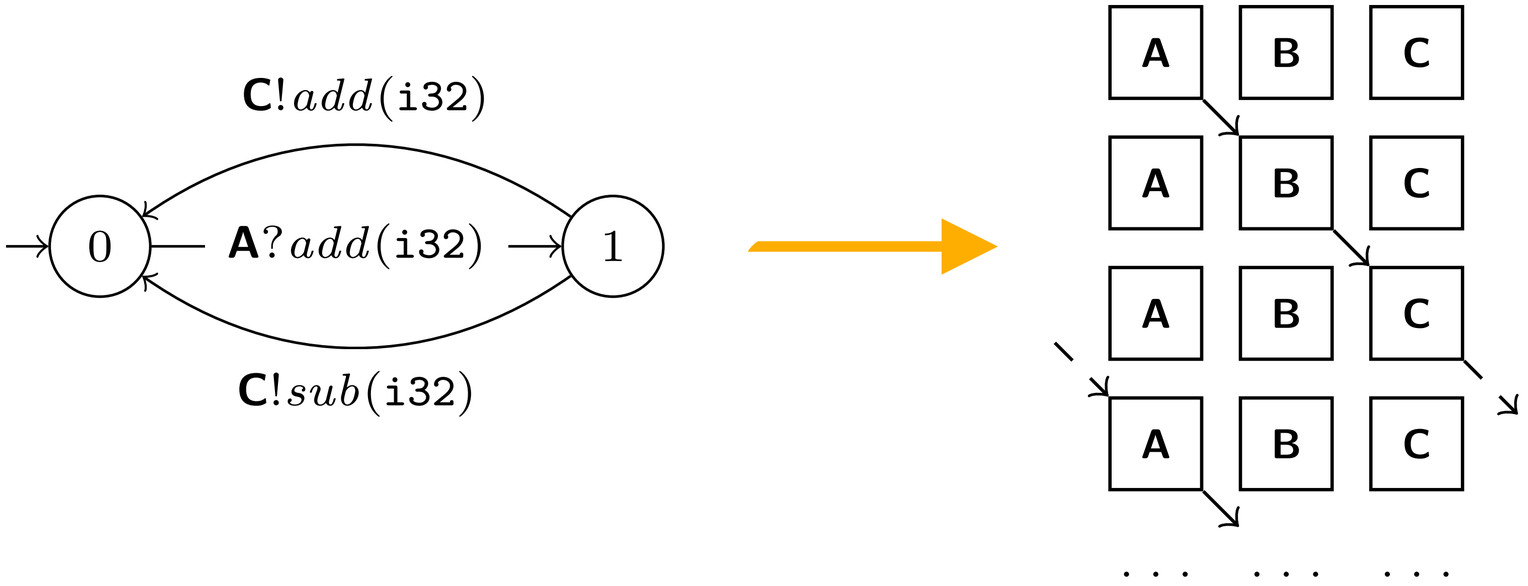
Ring Protocol

Example



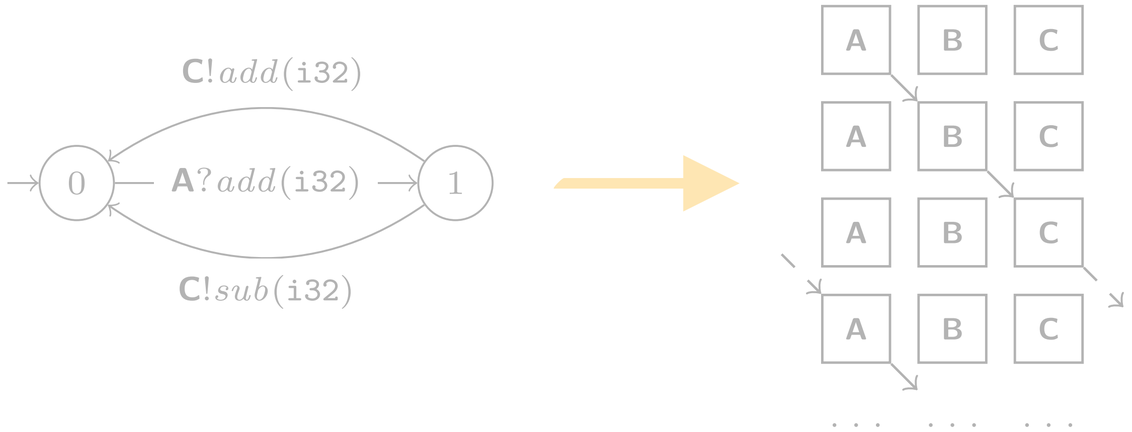
Ring Protocol

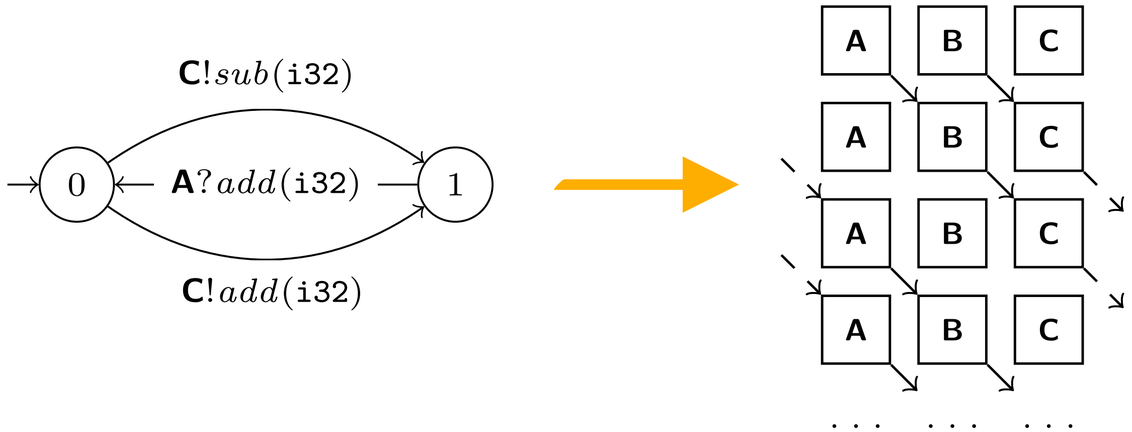
Example



Ring Protocol

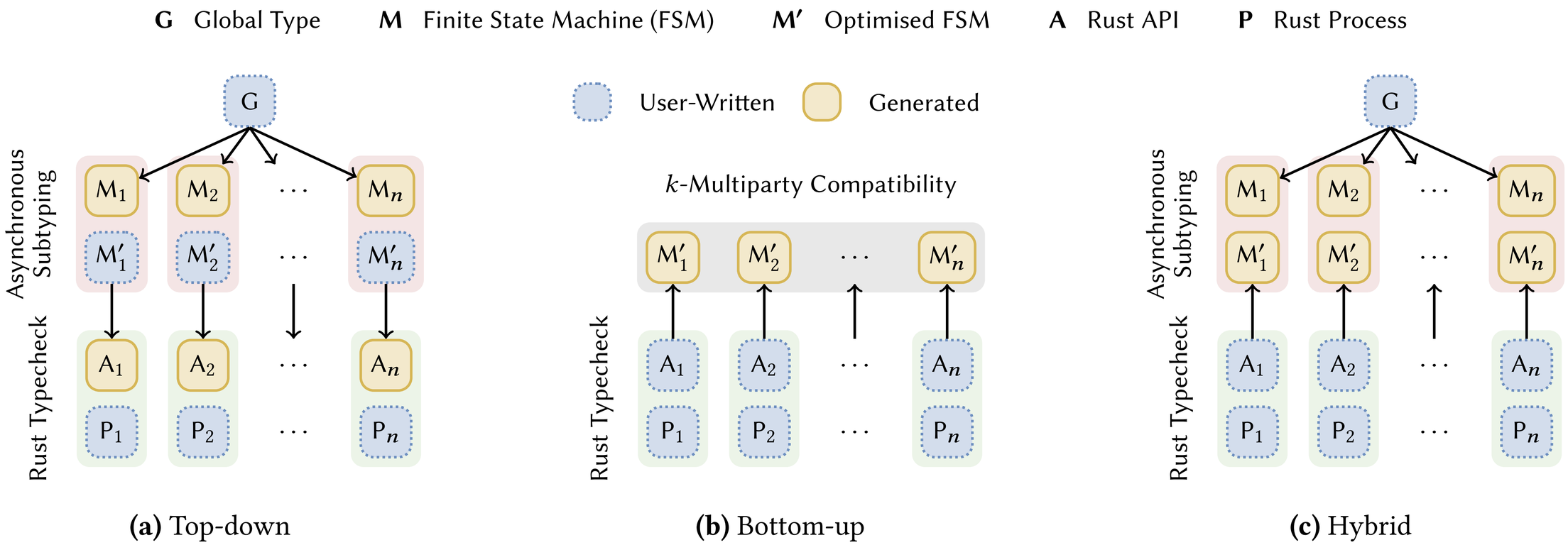
Example





Rumpsteak Framework

Three Approaches



(a) Top-down

(b) Bottom-up

(c) Hybrid

Theories for Communication Optimisation

Asynchronous Reordering Revisited

How do we check that asynchronous reorderings are safe?

Theories for Communication Optimisation

Asynchronous Reordering Revisited

How do we check that asynchronous reorderings are safe?

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

Theories for Communication Optimisation

Asynchronous Reordering Revisited

How do we check that asynchronous reorderings are safe?

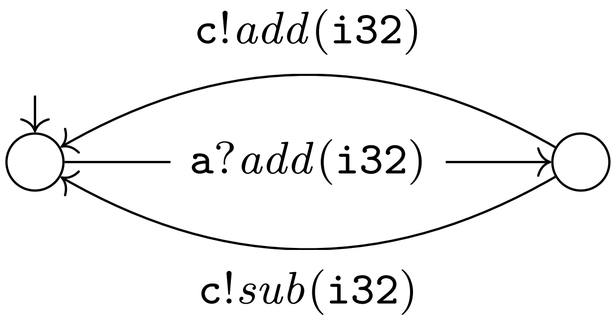
1. Asynchronous subtyping [Ghilezan, Pantvic, Prokic, Scalas and NY **POPL'2021**]

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

Safety

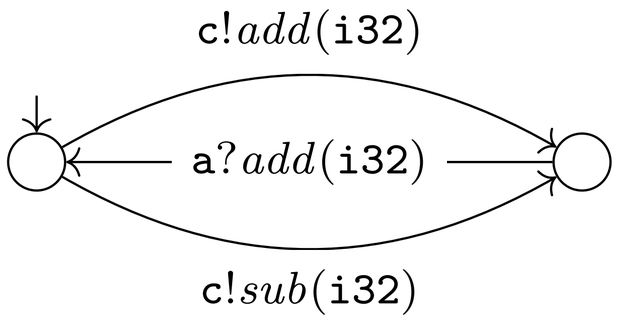
Asynchronous Subtyping

PROJECTED B



Safe?

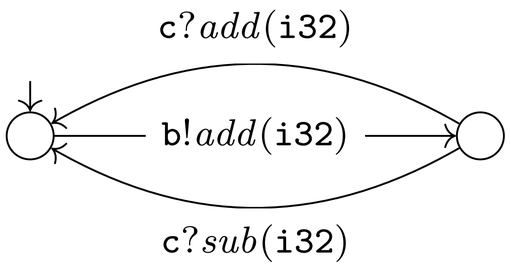
OPTIMISED B



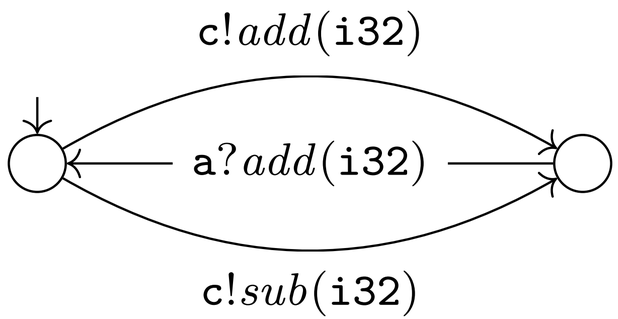
Safety

k-Multiparty Compatibility

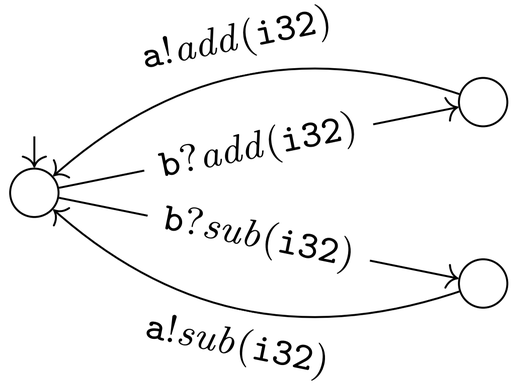
OPTIMISED A



OPTIMISED B

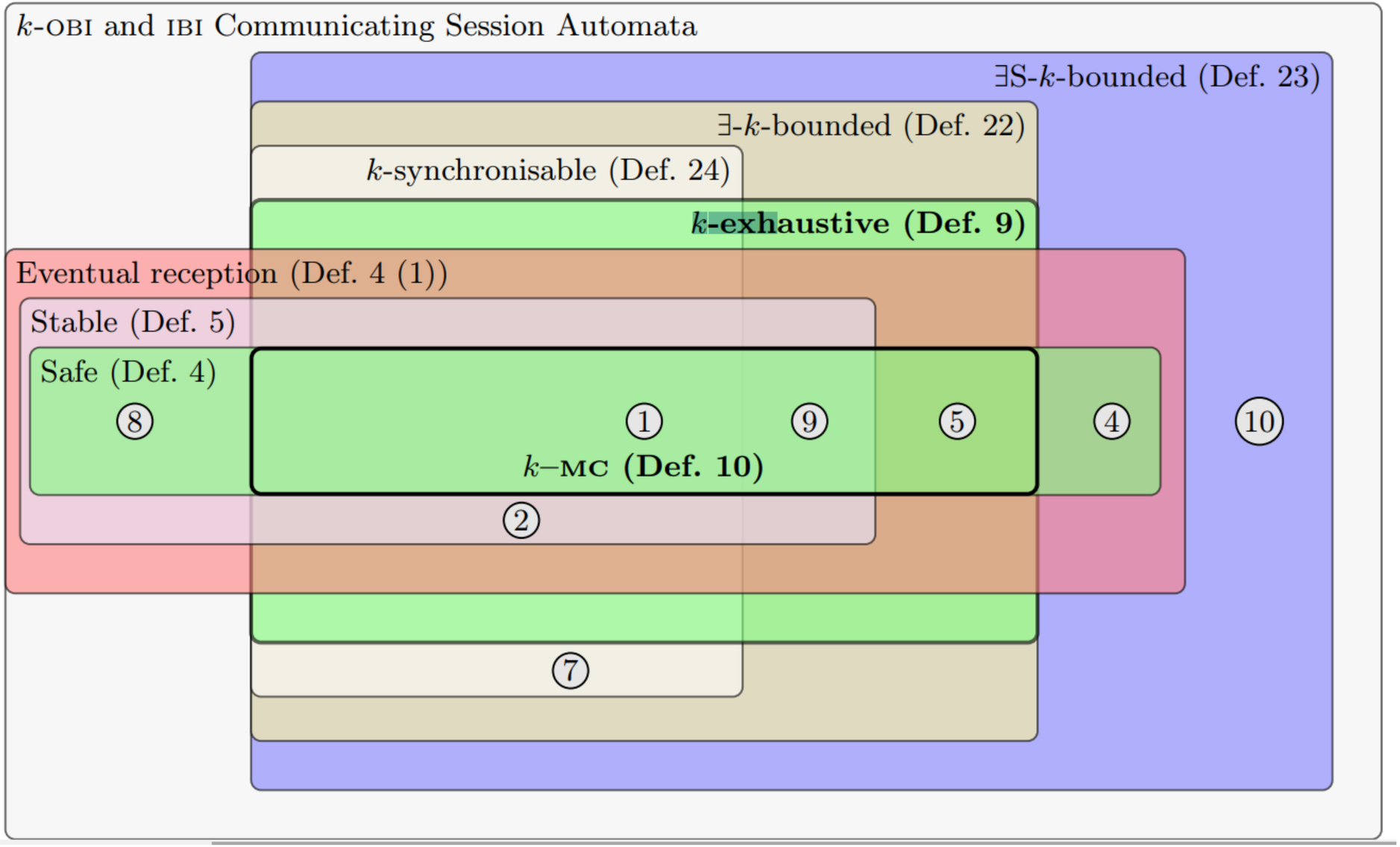


OPTIMISED C



Safe?

k-Multiparty Compatibility [CAV'19]



Asynchronous Subtyping

Existing work

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

Theorem [POPL 2021]

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

Asynchronous Subtyping

Existing work

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

Theorem [POPL 2021]

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

Asynchronous Subtyping

Existing work

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

Theorem [POPL 2021]

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

Asynchronous Subtyping

Existing work

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

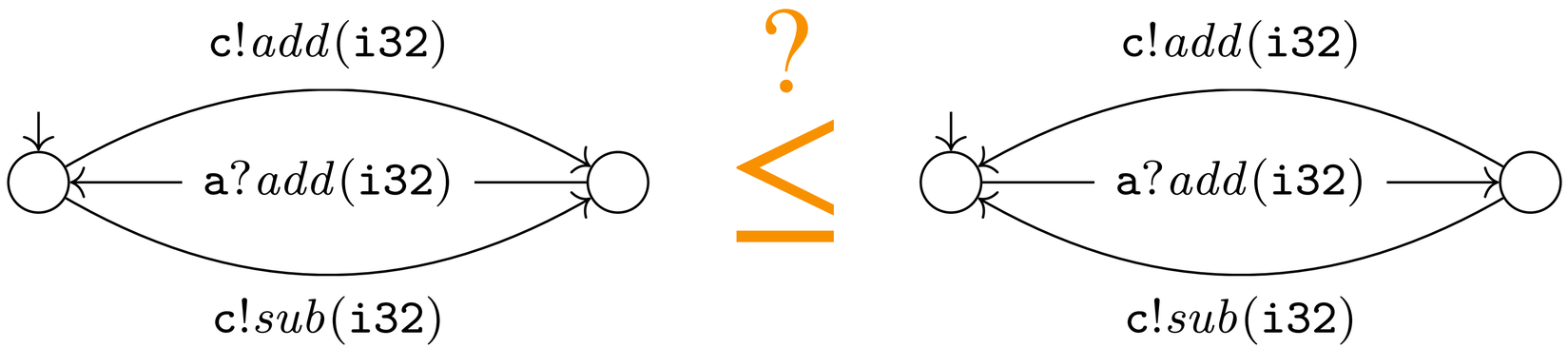
- Relation given by [Ghilezan et al., POPL 2021]
  - Sound ✅
  - Complete ✅
  - Decidable [Zavattaro et al., Lange and NY] ❌
- Our aim is a sound and decidable algorithm
- **Theorem [POPL 2021]**

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

Asynchronous Subtyping

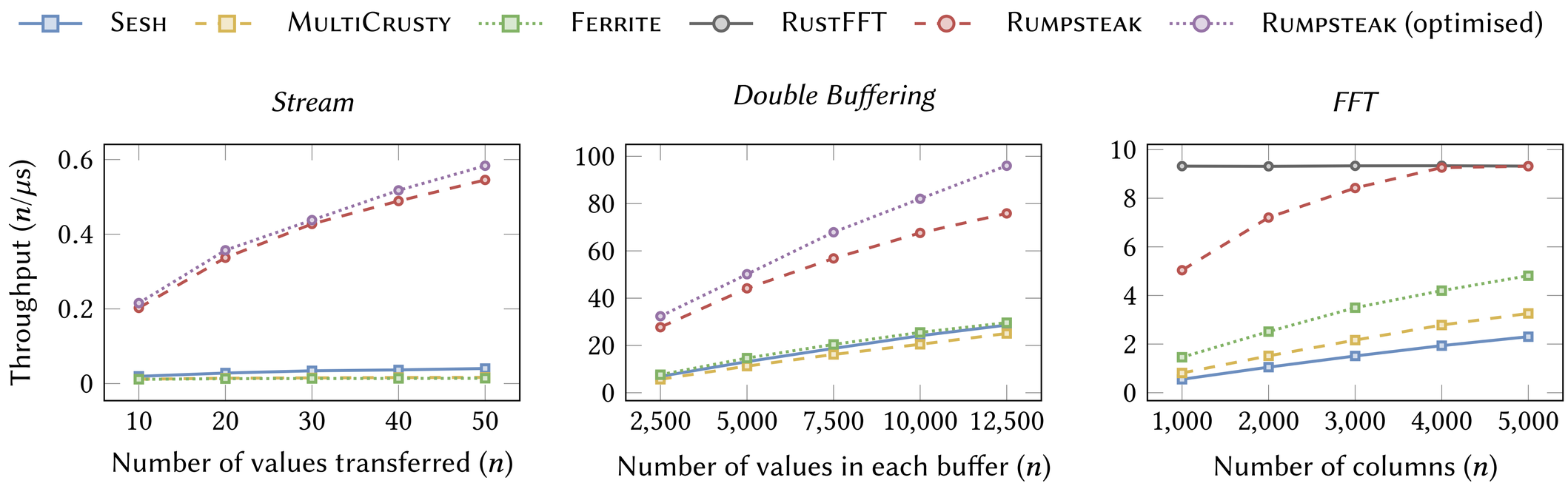
The Problem

- Choice and recursion make subtyping hard



Evaluation

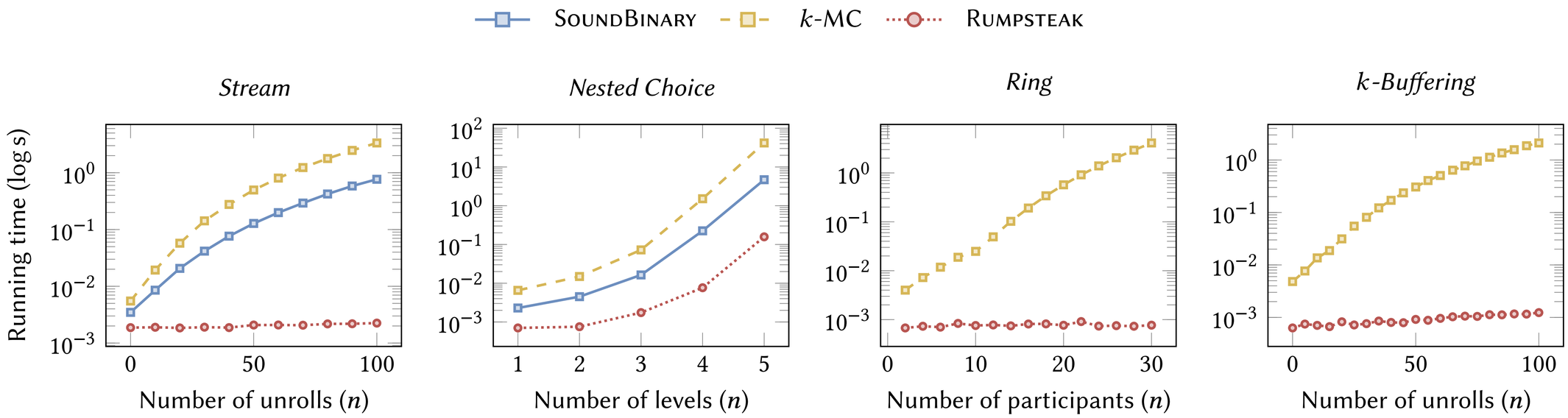
Rust Framework Benchmarks



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



Communicating Automata & Session Types

Asynchronous Subtyping Sound Algorithms [Bravetti, Lange and Zavattaro LMCS21, FoSSaCS'21]

Global Analysis (Bottom Up) k-MC [Langue and NY 19], Model-checking [Scalas & NY'19, Barwell et al 22]

Higher-Order Message Sequent Charts and Global Types (Top-Down)

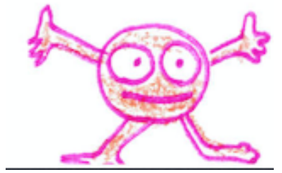
Complete Multiparty Session Type Projection with Automata [CAV'23, Li et al.] [ECOOP'23, Stutz]

Problem: A gap between end-point types and processes [Scalas and NY'19]

Session Types and Beyond

- What is 'Session Types'? Semantics (Behavioural Equivalences) and Expressiveness
- Mechanisations (Coq, Isabelle, etc)
- Fault Tolerance
- Security and Cryptography
- Probability
- Complexity (Computations and Verification)
- Applications: Programming Languages & Open Systems



Thank you! Questions?

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