Type-safe Web Programming Using Routed Multiparty Session Types in TypeScript

Anson Miu

Supervisor: Prof. Nobuko Yoshida
Second Marker: Dr. Iain Phillips
With thanks to Fangyi Zhou and Dr. Francisco Ferreira

June 22, 2020

Contributions

Type-safe Web Programming Using
Routed Multiparty Session Types in TypeScript

1. **SessionTS: a Session Type API Code Generation Toolchain for Modern Web Programming**
   Targets standard industrial strength technologies and idiomatic web programming practices

2. **RoutedSessions: a New Theory of Multiparty Session Types with Routed Communication**
   Formalised new theory, proved correctness, implemented in SessionTS to support peer-to-peer interactions over server-centric network structures

Contributions

Type-safe Web Programming Using Routed Multiparty Session Types **in TypeScript**

1. **SessionTS: a Session Type API Code Generation Toolchain for Modern Web Programming**
   Targets standard industrial strength technologies and idiomatic web programming practices

2. **ROUTEDSESSIONS: a New Theory of Multiparty Session Types with Routed Communication**
   Formalised new theory, proved correctness, implemented in SessionTS to support peer-to-peer interactions over server-centric network structures

Contributions

Type-safe Web Programming Using
Routed Multiparty Session Types in TypeScript

1. **SessionTS: a Session Type API Code Generation Toolchain for Modern Web Programming**
   Targets standard industrial strength technologies and idiomatic web programming practices

2. **RoutedSessions: a New Theory of Multiparty Session Types with Routed Communication**
   Formalised new theory, proved correctness, implemented in SessionTS to support peer-to-peer interactions over server-centric network structures

Contributions

Type-safe Web Programming Using
Routed Multiparty Session Types in TypeScript

1. **SessionTS: a Session Type API Code Generation Toolchain for Modern Web Programming**
   Targets standard industrial strength technologies and idiomatic web programming practices

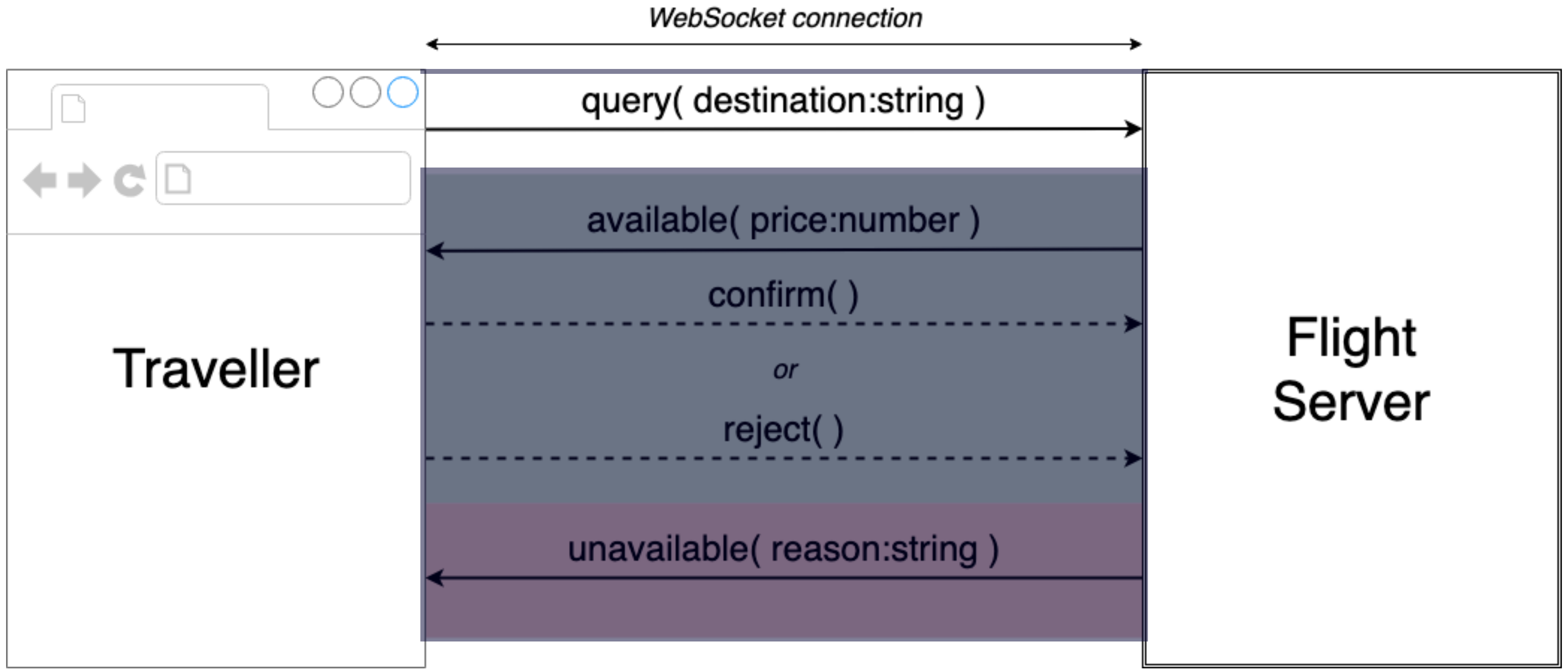
2. **RoutedSessions: a New Theory of Multiparty Session Types with Routed Communication**
   Formalised new theory, proved correctness, implemented in SessionTS to support peer-to-peer interactions over server-centric network structures

Problem

Communication Safety in Interactive Web Applications

Example: **Flight Booking Service**

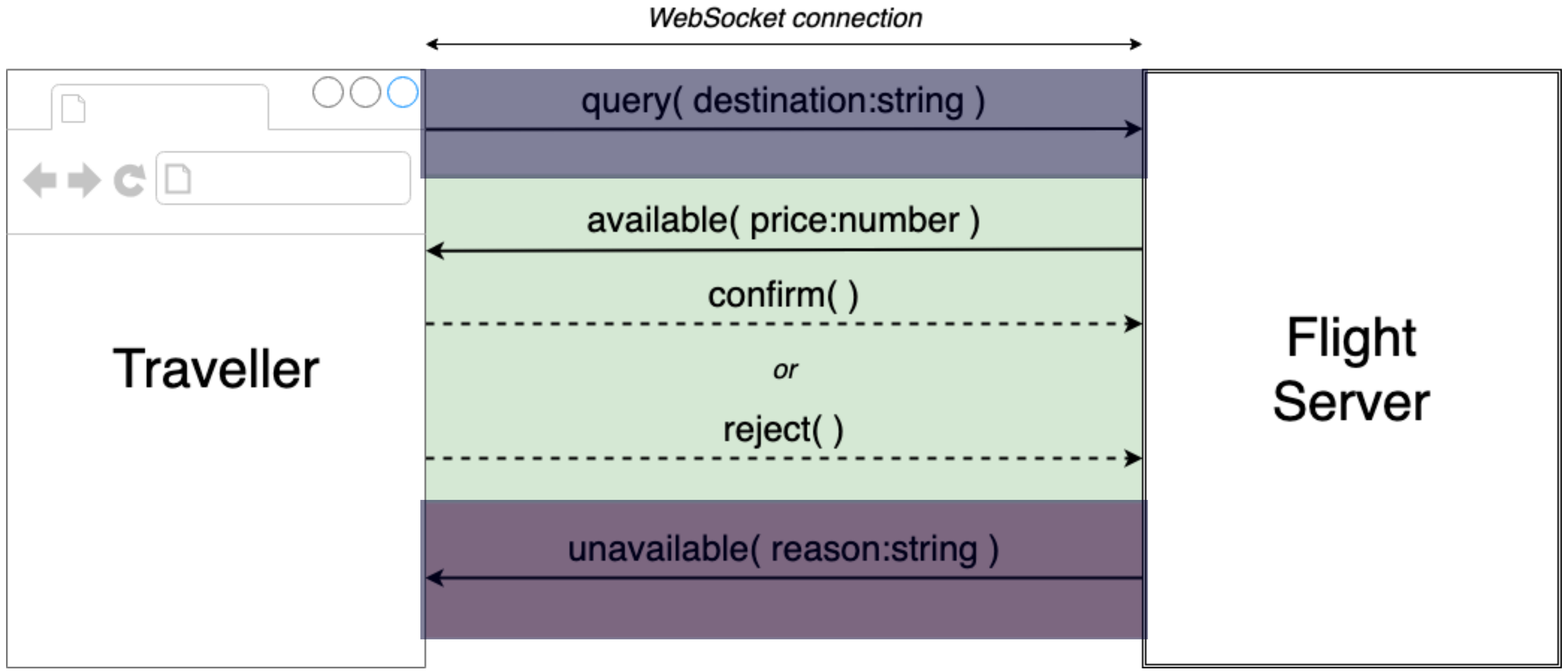
- Traveller asks Server about flight details for a particular destination.
- If available:
    - Server reserves seat
    - Server responds with price
    - Traveller responds with decision
    - If Traveller rejects, server releases seat
- Otherwise, Traveller can try again.



WebSocket connection
query(destination:string)
available(price:number)
confirm( )
or
reject( )
unavailable(reason:string)
Traveller
Flight Server

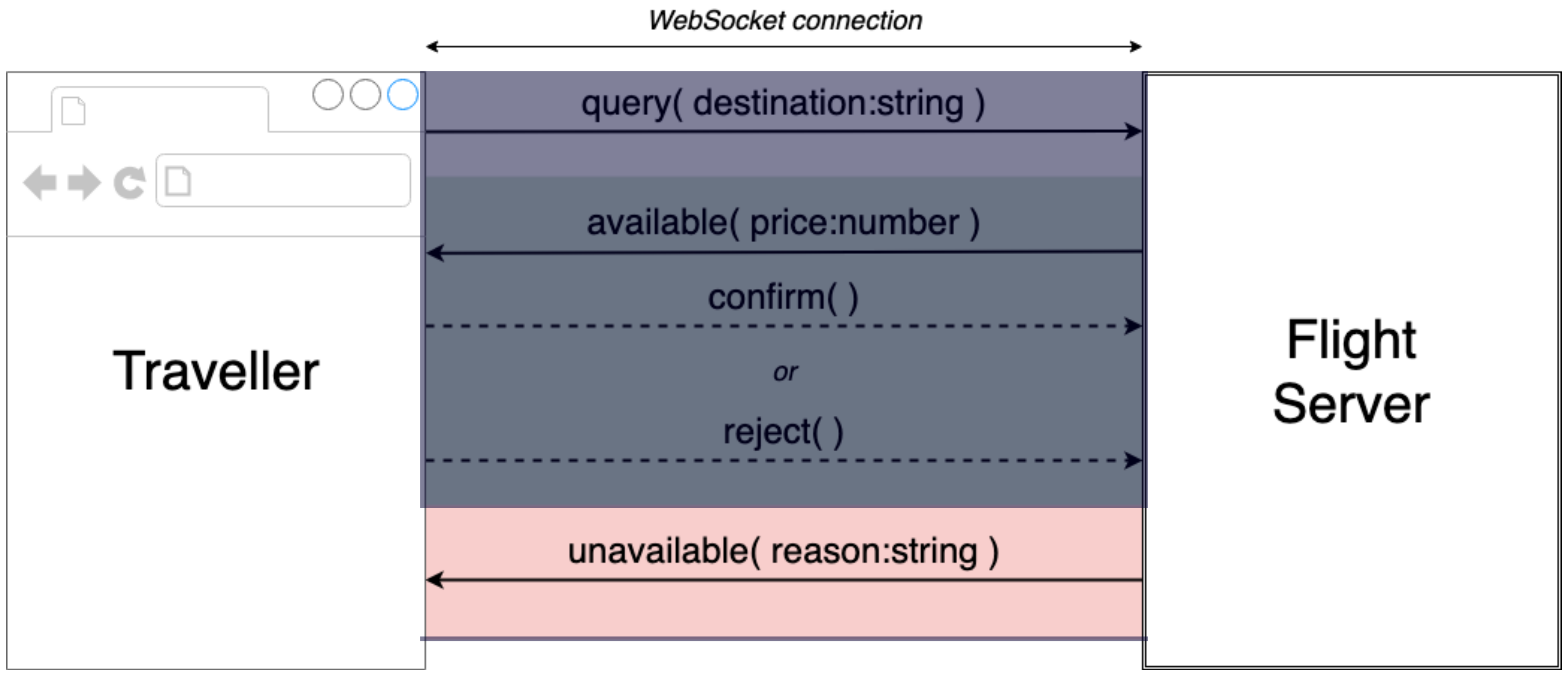
Example: **Flight Booking Service**

- Traveller asks Server about flight details for a particular destination.
- If available:
  - Server reserves seat
  - Server responds with price
  - Traveller responds with decision
  - If Traveller rejects, server releases seat
- Otherwise, Traveller can try again.


WebSocket connection
query(destination:string)
available(price:number)
confirm( )
or
reject( )
unavailable(reason:string)
Traveller
Flight Server

Example: **Flight Booking Service**

- Traveller asks Server about flight details for a particular destination.
- If available:
  - Server reserves seat
  - Server responds with price
  - Traveller responds with decision
  - If Traveller rejects, server releases seat
- Otherwise, Traveller can try again.


WebSocket connection
query(destination:string)
available(price:number)
confirm( )
or
reject( )
unavailable(reason:string)
Traveller
Flight Server

Problems: **Flight Booking Service**

- Traveller asks Server about flight details for a particular destination.
- If available:
  - Server reserves seat
  - Server responds with price
  - Traveller responds with decision
  - If Traveller rejects, server releases seat
- Otherwise, Traveller can try again.

Deadlocks

What if Traveller is waiting for quote whilst Server is waiting for destination?

Problems: **Flight Booking Service**

- Traveller asks Server about flight details for a particular destination.
- If available:
  - Server reserves seat
  - Server responds with price
  - Traveller responds with decision
  - If Traveller rejects, server releases seat
- Otherwise, Traveller can try again.

Communication Mismatch

What if Server sends string, but Traveller expects number?

Problems: **Flight Booking Service**

- Traveller asks Server about flight details for a particular destination.
- If available:
  - Server reserves seat
  - Server responds with price
  - Traveller responds with decision
  - If Traveller rejects, server releases seat
- Otherwise, Traveller can try again.

Channel Linearity Violation

What if Traveller sends query twice?
How many seats will be reserved?

Approach

using Multiparty Session Types

(1) Specify Communication

(2) Generate APIs from Specification

(1) Scribble Protocol Specification

```
type <typescript> “Credentials” from “./Payment” as Cred;
global protocol FlightService(role Traveller, role Server) {
    Destination(string) from Traveller to Server;
    choice at Server {
        Available(number) from Server to Traveller;
        choice at Traveller {
            Confirm(Cred) from Traveller to Server;
        } or { Reject() from Traveller to Server; }
    } or {
        Full() from Server to Traveller;
        do FlightService(Traveller, Server);
}}
```

$$G = \mu \mathtt{t}.\mathsf{Traveller} \to \mathsf{Server} : \mathsf{Destination}(\mathtt{string}).$$

$$\mathsf{Server} \to \mathsf{Traveller} \begin{cases} \mathsf{Available}(\mathtt{number}) : & G_{\mathsf{Available}} \\ \mathsf{Full}() : & \mathtt{t} \end{cases}$$

$$G_{\mathsf{Available}} = \mathsf{Traveller} \to \mathsf{Server} : \begin{cases} \mathsf{Confirm}(\mathtt{Cred}) : & \mathtt{end} \\ \mathsf{Reject}() : & \mathtt{end} \end{cases}$$

YOSHIDA, N., HU, R., NEYKOVA, R., AND NG, N. The Scribble Protocol Language. In *8th International Symposium on Trustworthy Global Computing* (2013), vol. 8358 of *LNCS*, Springer, pp. 22–41.

(1) Multiparty Session Types

```
type <typescript> “Credentials” from “./Payment” as Cred;
global protocol FlightService(role Traveller, role Server) {
  Destination(string) from Traveller to Server;
  choice at Server {
    Available(number) from Server to Traveller;
    choice at Traveller {
      Confirm(Cred) from Traveller to Server;
    } or { Reject() from Traveller to Server; }
  } or {
    Full() from Server to Traveller;
    do FlightService(Traveller, Server);
}}
```

$$G = \mu\mathtt{t}.\mathsf{Traveller} \to \mathsf{Server} : \mathsf{Destination}(\mathtt{string}).$$

$$\mathsf{Server} \to \mathsf{Traveller} \begin{cases} \mathsf{Available}(\mathtt{number}) : & G_{\mathsf{Available}} \\ \mathsf{Full}() : & \mathtt{t} \end{cases}$$

$$G_{\mathsf{Available}} = \mathsf{Traveller} \to \mathsf{Server} : \begin{cases} \mathsf{Confirm}(\mathtt{Cred}) : & \mathtt{end} \\ \mathsf{Reject}() : & \mathtt{end} \end{cases}$$

HONDA, K., YOSHIDA, N., AND CARBONE, M. Multiparty Asynchronous Session Types. In *35th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages* (2008), ACM, pp. 273–284.

(1) Scribble Protocol Specification

```
type <typescript> “Credentials” from “./Payment” as Cred;
global protocol FlightService(role Traveller, role Server) {
    Destination(string) from Traveller to Server;
    choice at Server {
        Available(number) from Server to Traveller;
        choice at Traveller {
            Confirm(Cred) from Traveller to Server;
        } or { Reject() from Traveller to Server; }
    } or {
        Full() from Server to Traveller;
        do FlightService(Traveller, Server);
}}
```

$$G = \mu \mathtt{t}.\boxed{\mathsf{Traveller} \to \mathsf{Server} : \mathsf{Destination}(\mathtt{string}).}$$

$$\mathsf{Server} \to \mathsf{Traveller} \begin{cases} \mathsf{Available}(\mathtt{number}) : & G_{\mathsf{Available}} \\ \mathsf{Full}() : & \mathtt{t} \end{cases}$$

$$G_{\mathsf{Available}} = \mathsf{Traveller} \to \mathsf{Server} : \begin{cases} \mathsf{Confirm}(\mathtt{Cred}) : & \mathtt{end} \\ \mathsf{Reject}() : & \mathtt{end} \end{cases}$$

YOSHIDA, N., HU, R., NEYKOVA, R., AND NG, N. The Scribble Protocol Language. In *8th International Symposium on Trustworthy Global Computing* (2013), vol. 8358 of *LNCS*, Springer, pp. 22–41.

HONDA, K., YOSHIDA, N., AND CARBONE, M. Multiparty Asynchronous Session Types. In *35th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages* (2008), ACM, pp. 273–284.

(1) Scribble Protocol Specification

```
type <typescript> "Credentials" from "./Payment" as Cred;
global protocol FlightService(role Traveller, role Server) {
    Destination(string) from Traveller to Server;
    choice at Server {
        Available(number) from Server to Traveller;
        choice at Traveller {
            Confirm(Cred) from Traveller to Server;
        } or { Reject() from Traveller to Server; }
    } or {
        Full() from Server to Traveller;
        do FlightService(Traveller, Server);
}}
```

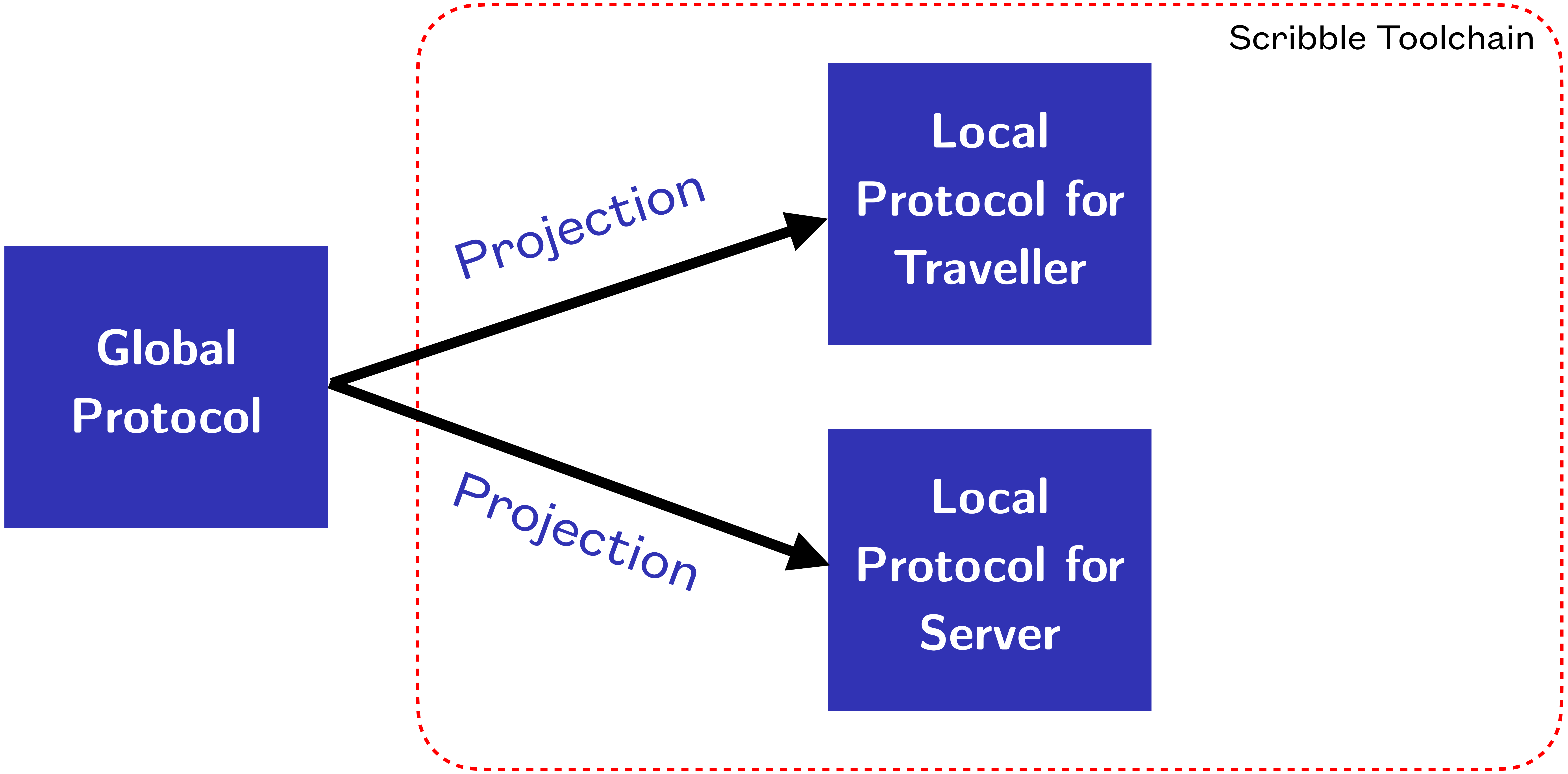
$$G = \boxed{\mu \mathtt{t}.}\,\mathsf{Traveller} \to \mathsf{Server} : \mathsf{Destination}(\mathtt{string}).$$

$$\mathsf{Server} \to \mathsf{Traveller} \begin{cases} \mathsf{Available}(\mathtt{number}) : & G_{\mathsf{Available}} \\ \boxed{\mathsf{Full}() : \qquad\qquad\qquad \mathtt{t}} \end{cases}$$

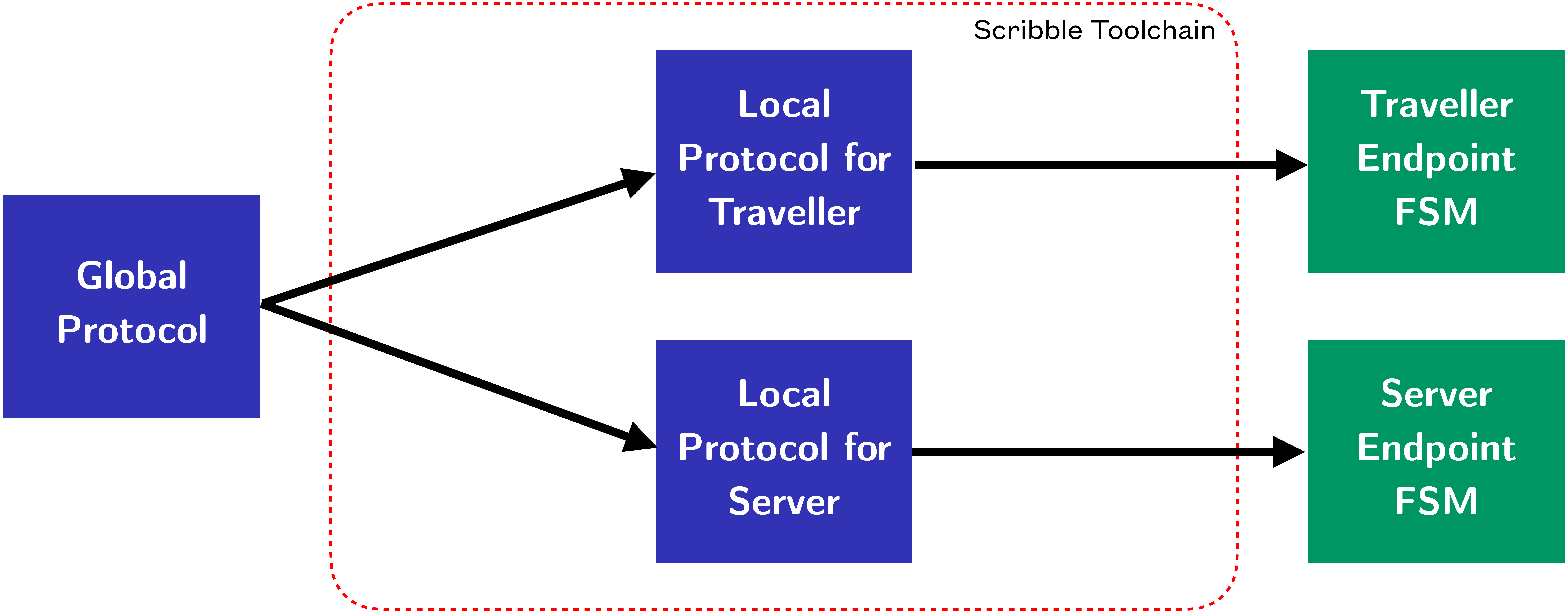
$$G_{\mathsf{Available}} = \mathsf{Traveller} \to \mathsf{Server} : \begin{cases} \mathsf{Confirm}(\mathtt{Cred}) : & \mathtt{end} \\ \mathsf{Reject}() : & \mathtt{end} \end{cases}$$

YOSHIDA, N., HU, R., NEYKOVA, R., AND NG, N. The Scribble Protocol Language. In *8th International Symposium on Trustworthy Global Computing* (2013), vol. 8358 of *LNCS*, Springer, pp. 22–41.
HONDA, K., YOSHIDA, N., AND CARBONE, M. Multiparty Asynchronous Session Types. In *35th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages* (2008), ACM, pp. 273–284.

(2) Endpoint API Generation

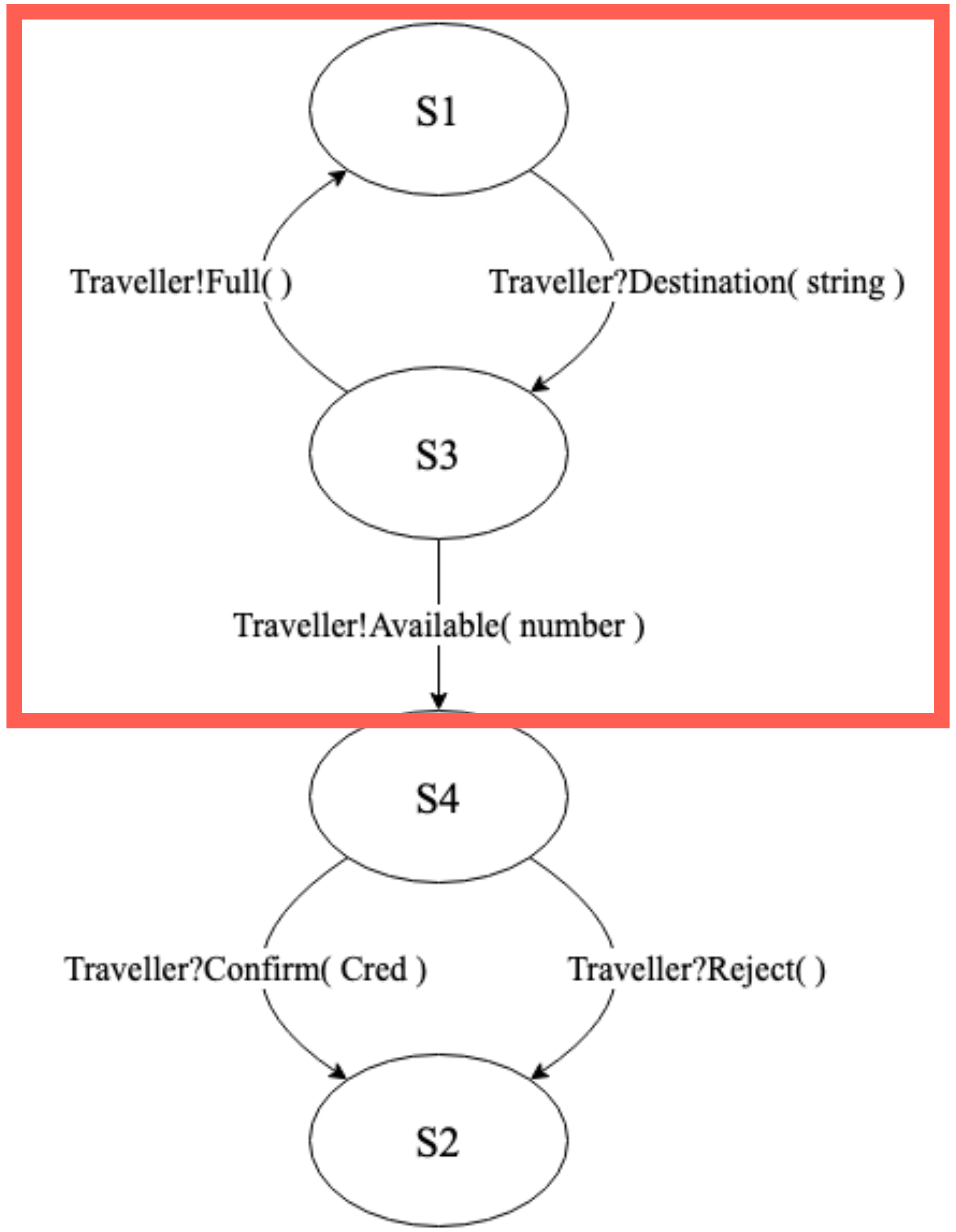


(2) Endpoint API Generation



(2) Endpoint API Generation

```
type <typescript> “Credentials” from “./Payment”
global protocol FlightService(role Traveller, role Server) {
  Destination(string) from Traveller to Server;
  choice at Server {
    Available(number) from Server to Traveller;
    choice at Traveller {
      Confirm(Cred) from Traveller to Server;
    } or { Reject() from Traveller to Server; }
  } or {
    Full() from Server to Traveller;
    do FlightService(Traveller, Server);
}}
```


S1
Traveller!Full( )
Traveller?Destination(string)
S3
Traveller!Available(number)
S4
Traveller?Confirm(Cred)
Traveller?Reject( )
S2

Server Endpoint FSM

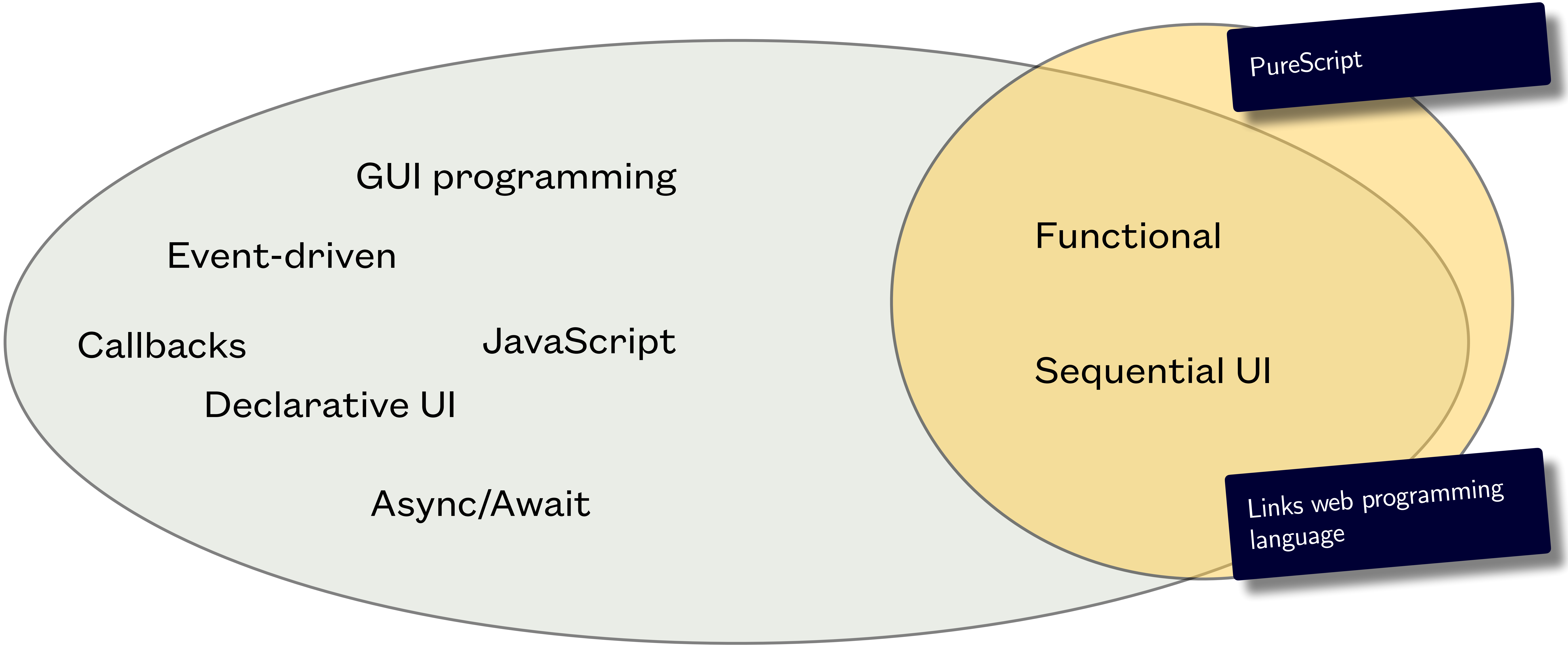
HU, R. Distributed Programming Using Java APIs Generated from Session Types. 22.
HU, R., AND YOSHIDA, N. Hybrid Session Verification through Endpoint API Generation. In *19th International Conference on Fundamental Approaches to Software Engineering* (2016), vol. 9633 of *LNCS*, Springer, pp. 401–418.
KING, J., NG, N., AND YOSHIDA, N. Multiparty Session Type-safe Web Development with Static Linearity. *Electronic Proceedings in Theoretical Computer Science 291* (Apr 2019), 35–46.

Limitations of State of the Art

Not Widely Used

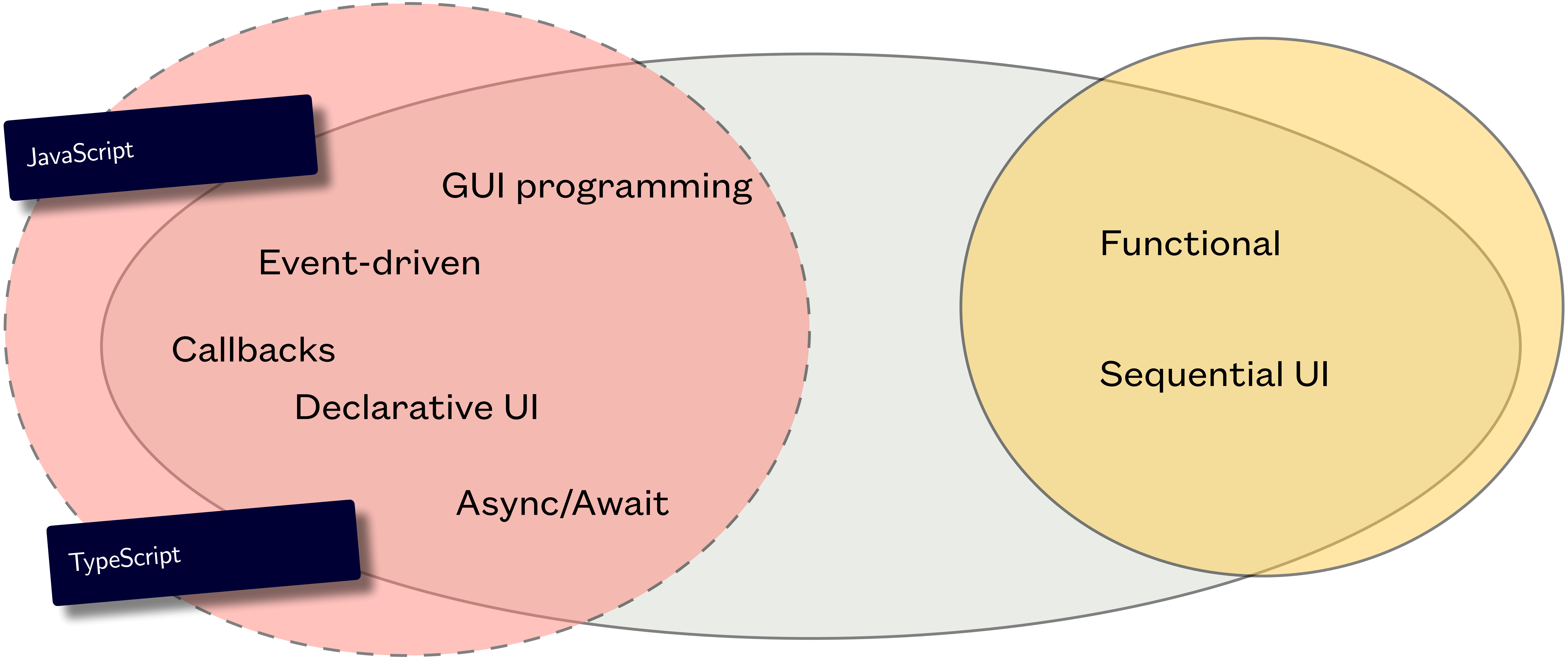
Only Server-Centric Protocols

Limitation 1: **Not Widely Used**



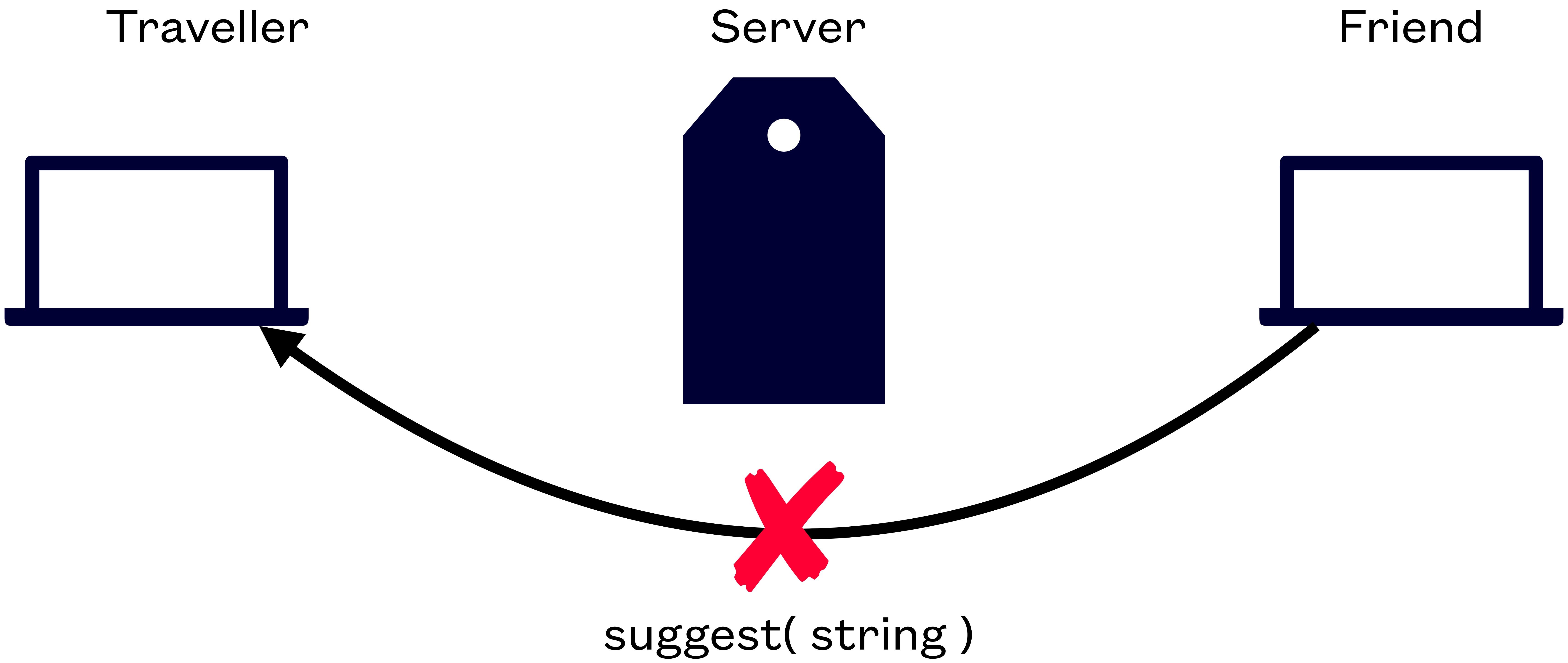
KING, J., NG, N., AND YOSHIDA, N. Multiparty Session Type-safe Web Development with Static Linearity. *Electronic Proceedings in Theoretical Computer Science 291* (Apr 2019), 35–46.
FOWLER, S. Model-View-Update-Communicate: Session Types meet the Elm Architecture. *ECOOP 2020*.

Limitation 1: **Not Widely Used**



KING, J., NG, N., AND YOSHIDA, N. Multiparty Session Type-safe Web Development with Static Linearity. *Electronic Proceedings in Theoretical Computer Science 291* (Apr 2019), 35–46.
FOWLER, S. Model-View-Update-Communicate: Session Types meet the Elm Architecture. *ECOOP 2020*.

Limitation 2: **Only Server-Centric Protocols**



KING, J., NG, N., AND YOSHIDA, N. Multiparty Session Type-safe Web Development with Static Linearity. *Electronic Proceedings in Theoretical Computer Science 291* (Apr 2019), 35–46.
FOWLER, S. Model-View-Update-Communicate: Session Types meet the Elm Architecture. *ECOOP 2020*.

Contributions

Type-safe Web Programming Using
Routed Multiparty Session Types in TypeScript

1. **SessionTS: a Session Type API Code Generation Toolchain for Modern Web Programming**
   Targets standard industrial strength technologies and idiomatic web programming practices

2. **RoutedSessions: a New Theory of Multiparty Session Types with Routed Communication**
   Formalised new theory, proved correctness, implemented in SessionTS to support peer-to-peer interactions over server-centric network structures

Contributions

Type-safe Web Programming Using
Routed Multiparty Session Types **in TypeScript**

1. **SessionTS: a Session Type API Code Generation Toolchain for Modern Web Programming**
   Targets standard industrial strength technologies and idiomatic web programming practices

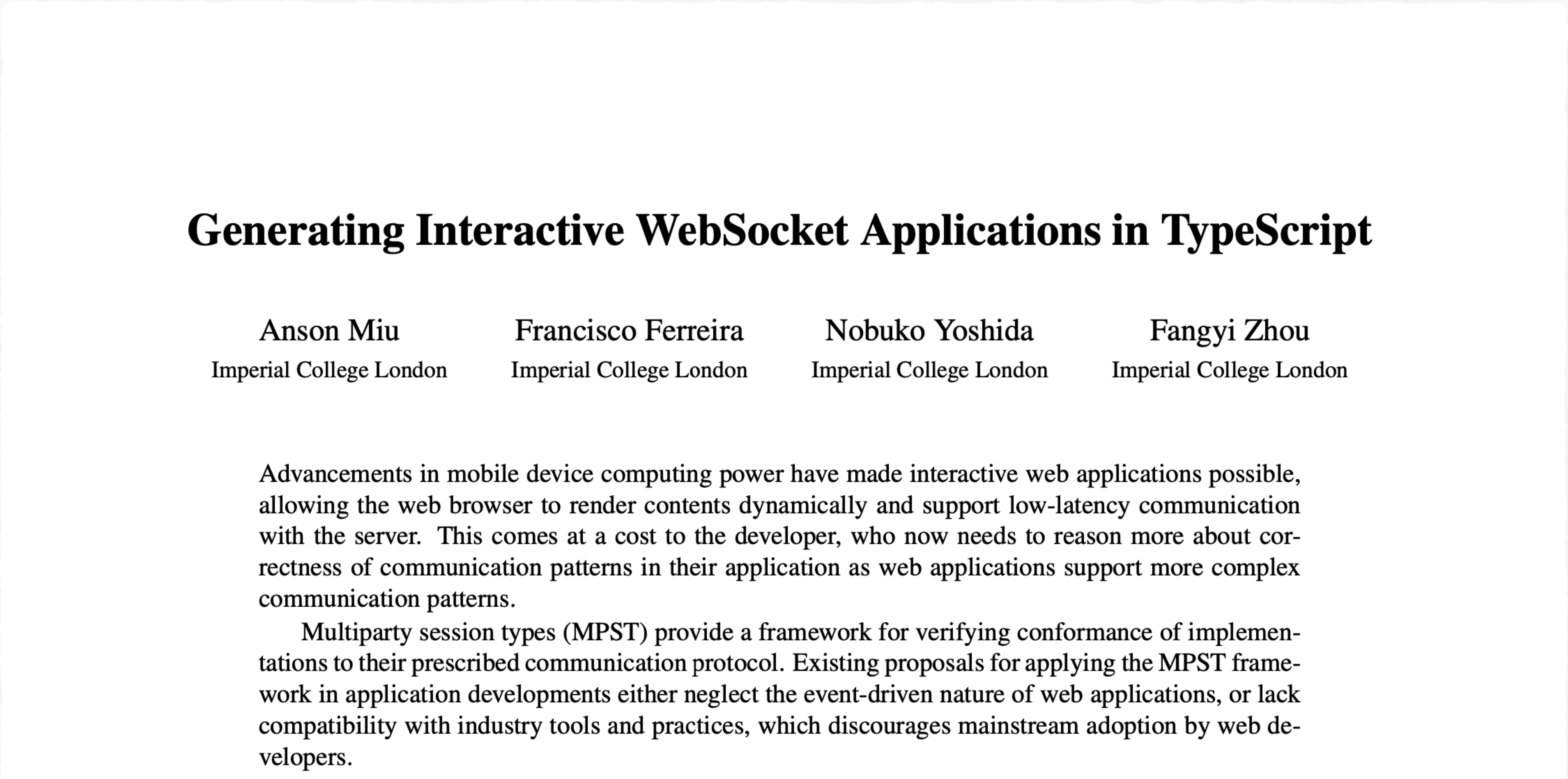
2. **RoutedSessions: a New Theory of Multiparty Session Types with Routed Communication**
   Formalised new theory, proved correctness, implemented in SessionTS to support peer-to-peer interactions over server-centric network structures

SessionTS

A Session Type API Code Generation Toolchain for Modern Web Programming

Initial Work

- Accepted to the 12th International Workshop on Programming Language Approaches to Concurrency- & Communication-cEntric Software (PLACES 2020)
- Published in the *Electronic Proceedings in Theoretical Computer Science* (EPTCS)



Generating Interactive WebSocket Applications in TypeScript

Anson Miu
Imperial College London

Francisco Ferreira
Imperial College London

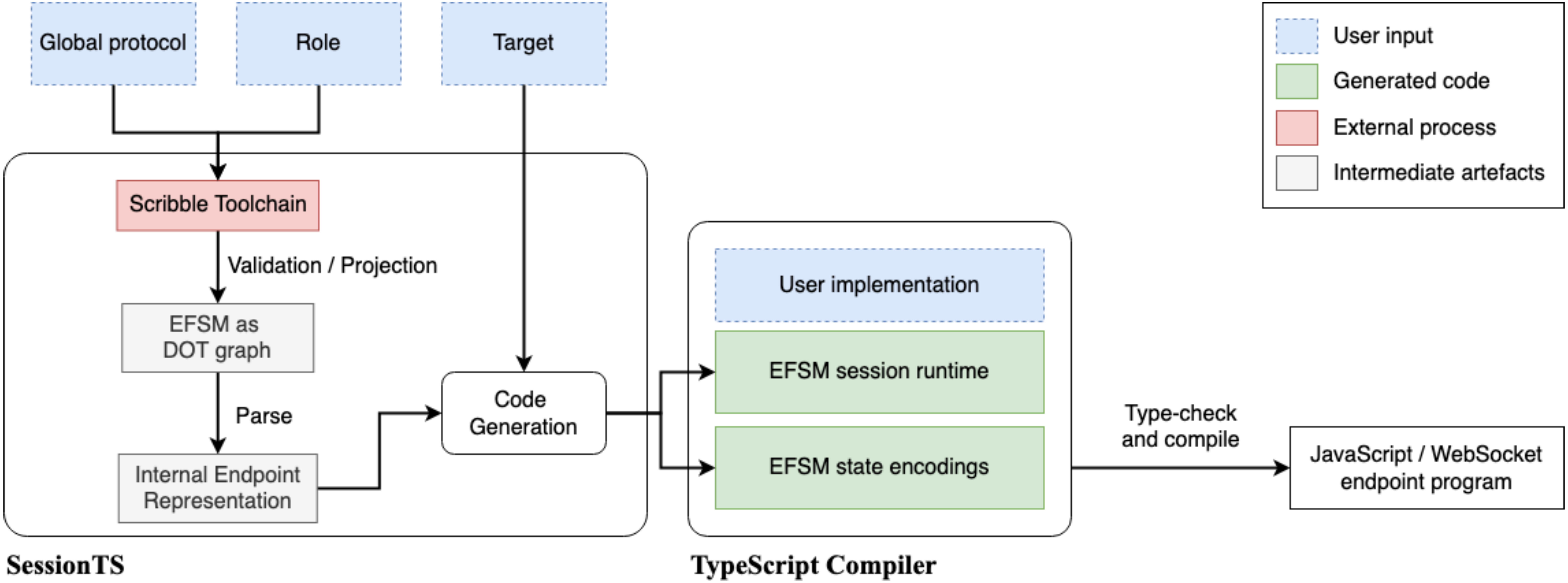
Nobuko Yoshida
Imperial College London

Fangyi Zhou
Imperial College London

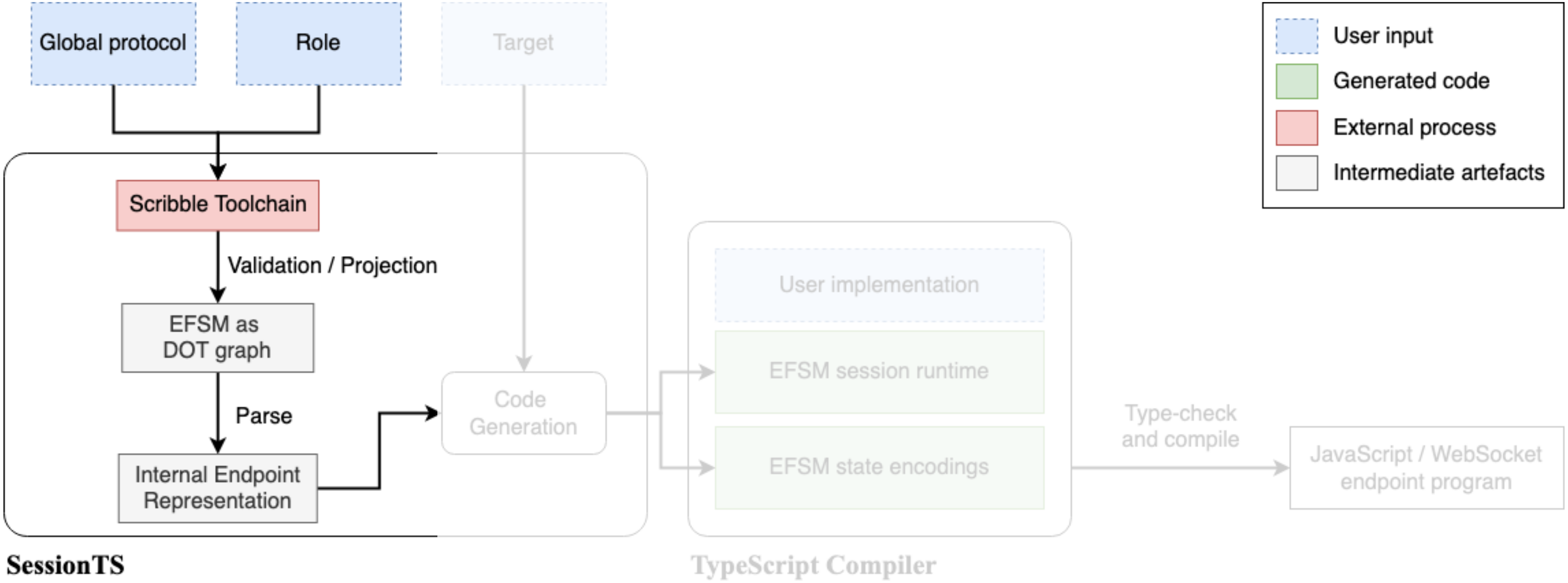
Advancements in mobile device computing power have made interactive web applications possible, allowing the web browser to render contents dynamically and support low-latency communication with the server. This comes at a cost to the developer, who now needs to reason more about correctness of communication patterns in their application as web applications support more complex communication patterns.

Multiparty session types (MPST) provide a framework for verifying conformance of implementations to their prescribed communication protocol. Existing proposals for applying the MPST framework in application developments either neglect the event-driven nature of web applications, or lack compatibility with industry tools and practices, which discourages mainstream adoption by web developers.

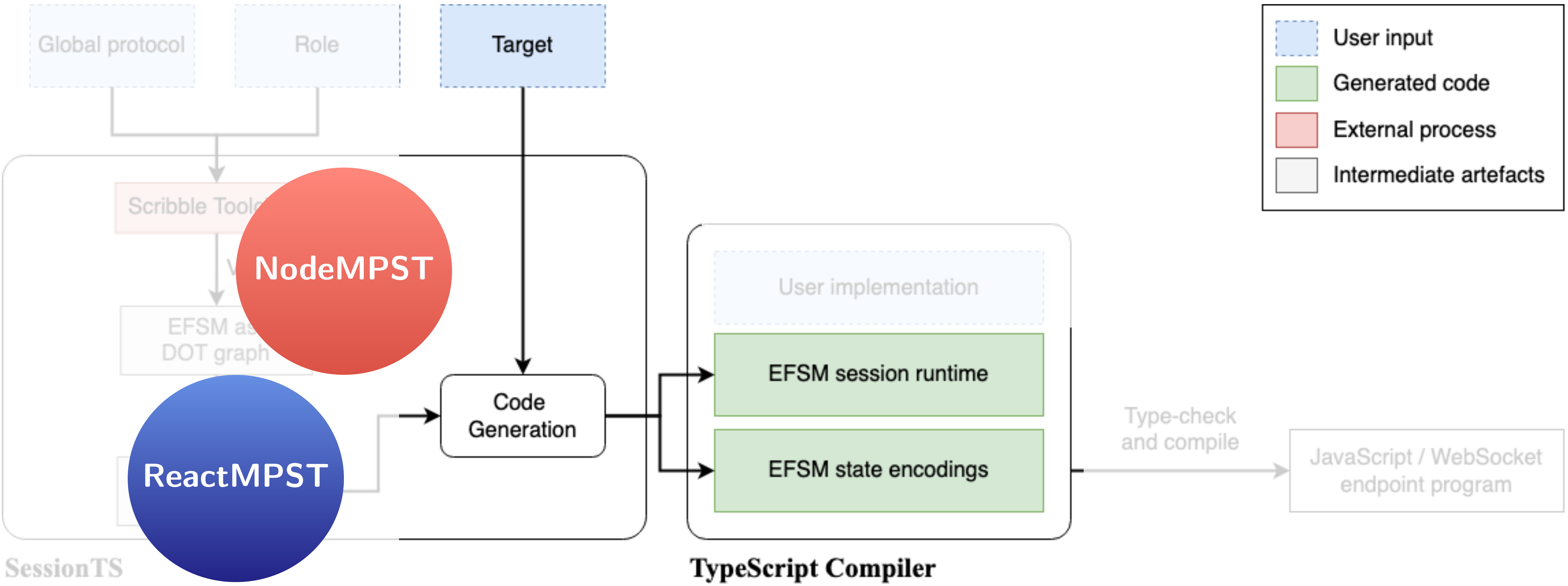
Workflow


Global protocol
Role
Target
Scribble Toolchain
Validation / Projection
EFSM as DOT graph
Parse
Internal Endpoint Representation
Code Generation
SessionTS
User implementation
EFSM session runtime
EFSM state encodings
TypeScript Compiler
Type-check and compile
JavaScript / WebSocket endpoint program
User input
Generated code
External process
Intermediate artefacts

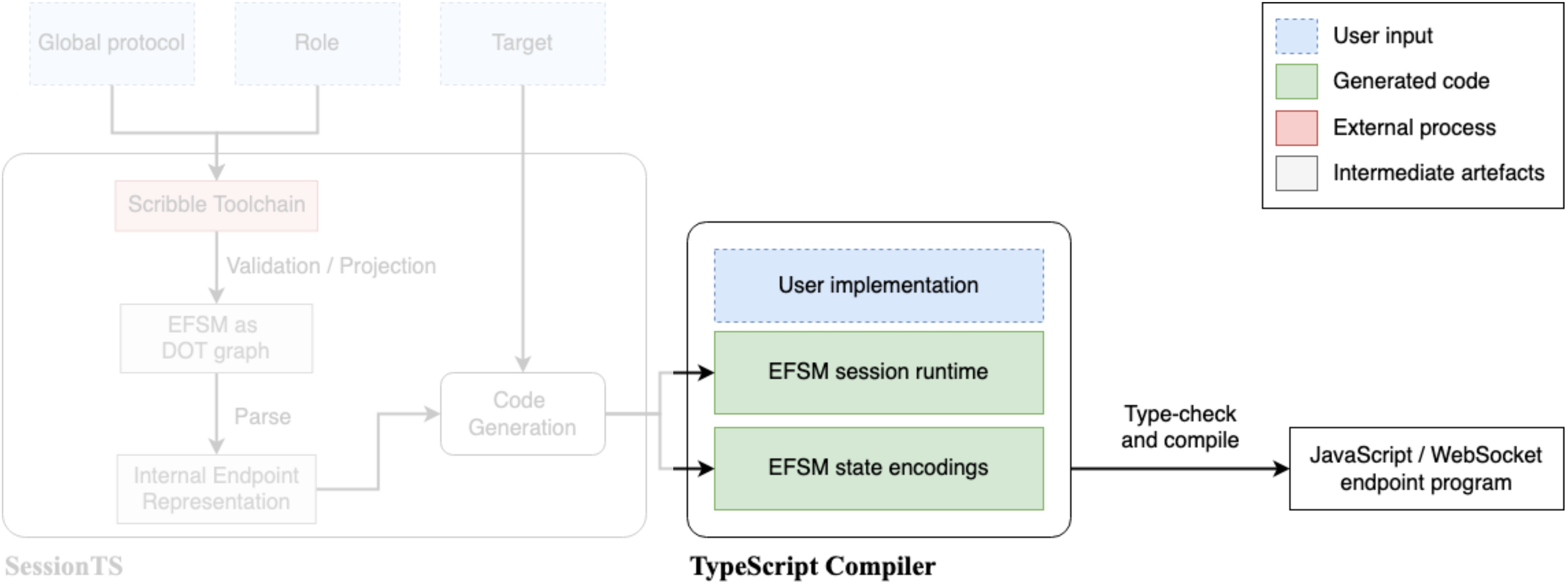
(1) Obtain EFSM from Protocol



(2) Generate APIs



(3) Implement APIs and Compile

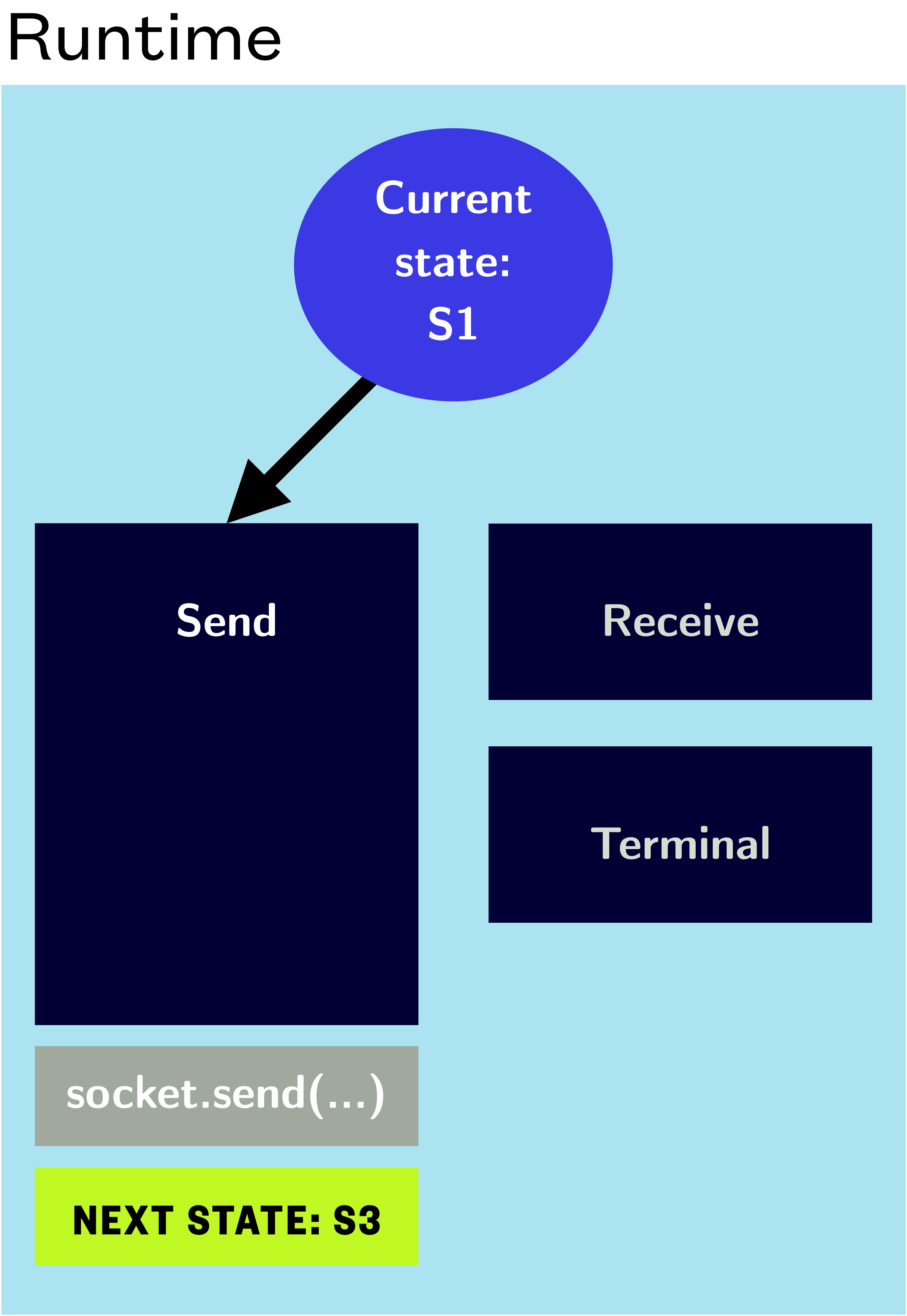


Demo

Type-safe Flight Booking Service

Design Philosophy

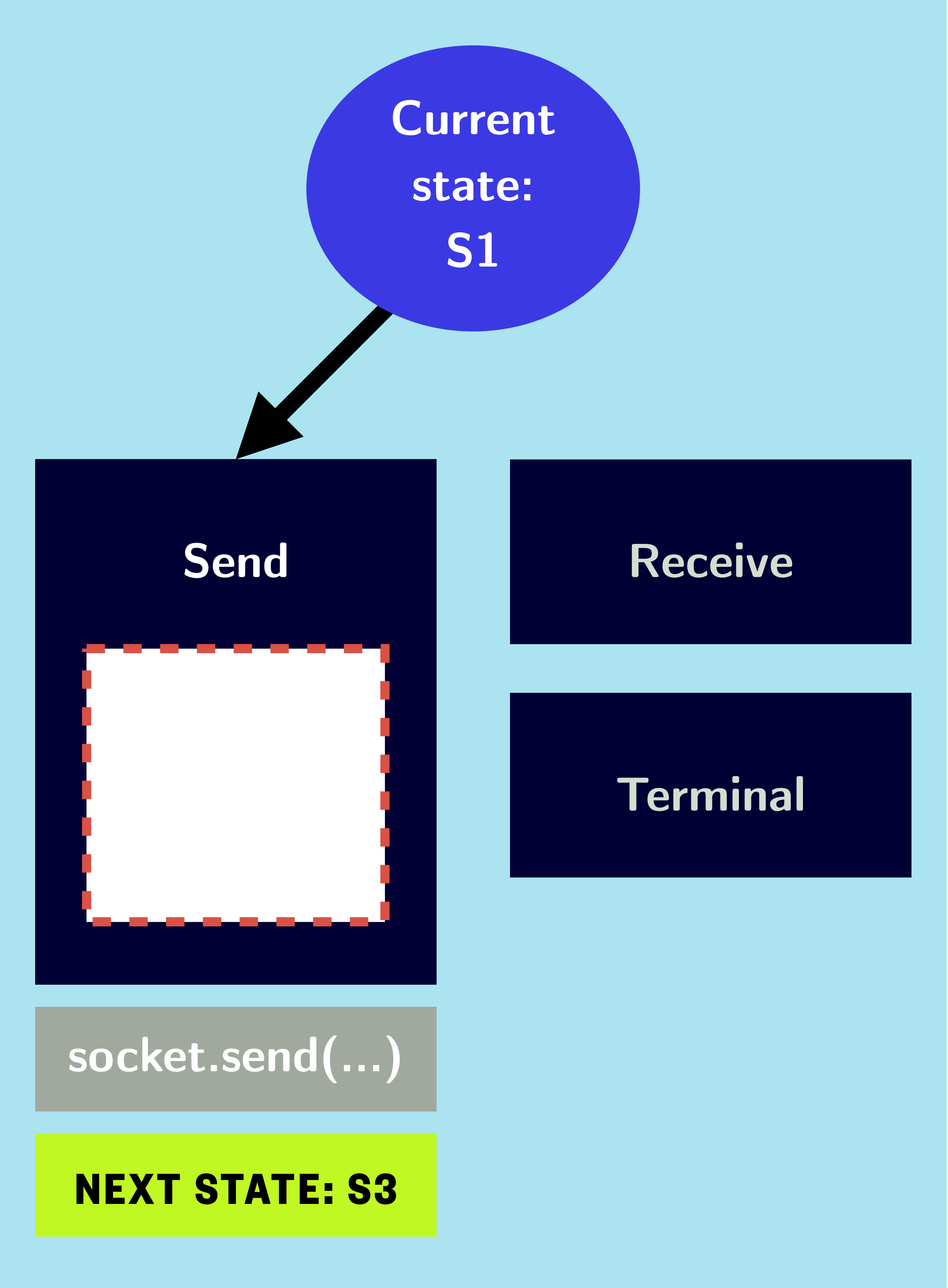
- We generate the session runtime to execute EFSM
  - Performs I/O action for current state
- We construct types for injecting business logic
  - What to send? How to handle receive?
- Developer instantiates session runtime with custom implementations



Design Philosophy

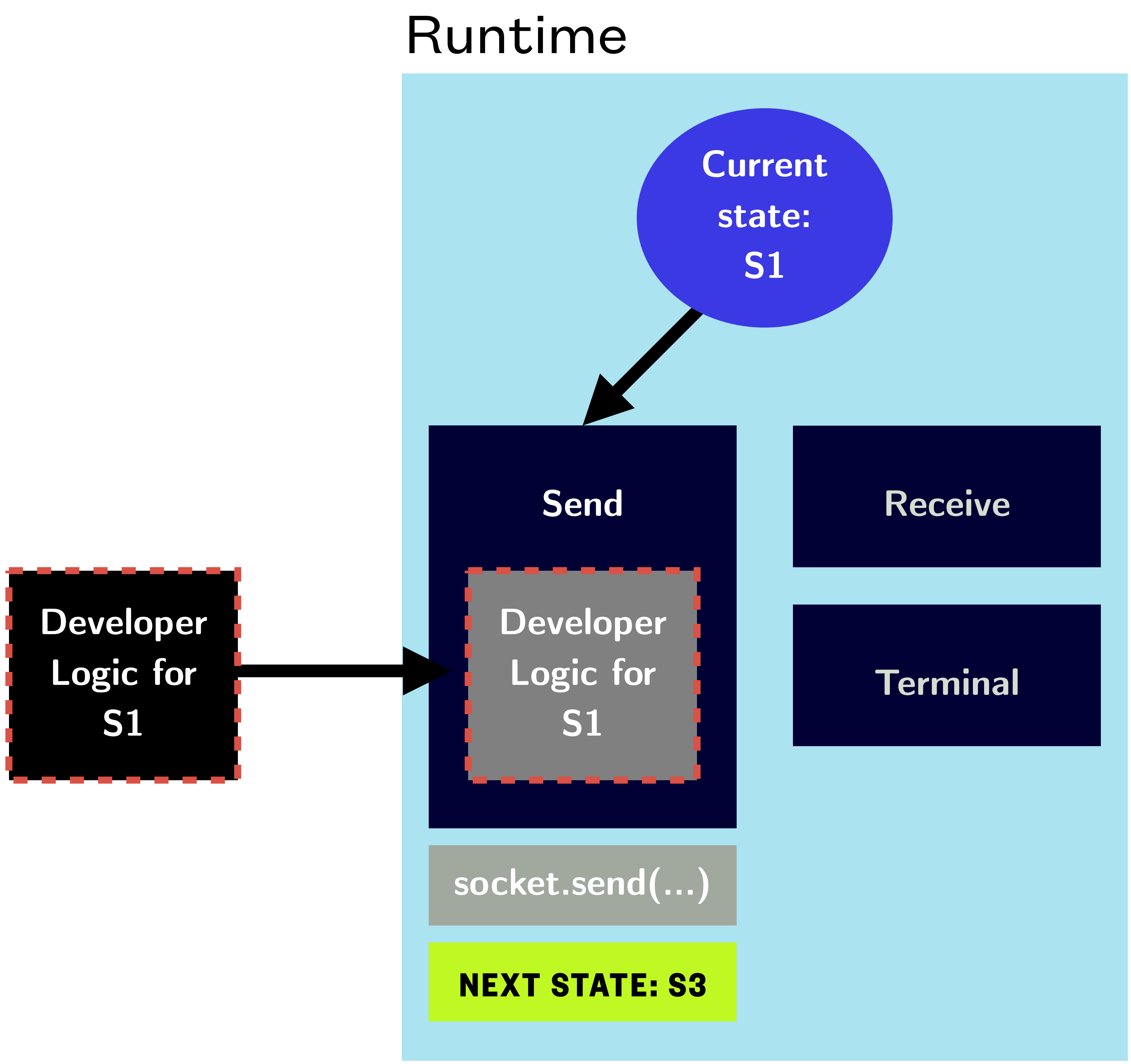
- We generate the session runtime to execute EFSM
  - Performs I/O action for current state
- We construct types for injecting business logic
  - What to send? How to handle receive?
- Developer instantiates session runtime with custom implementations

Runtime



Design Philosophy

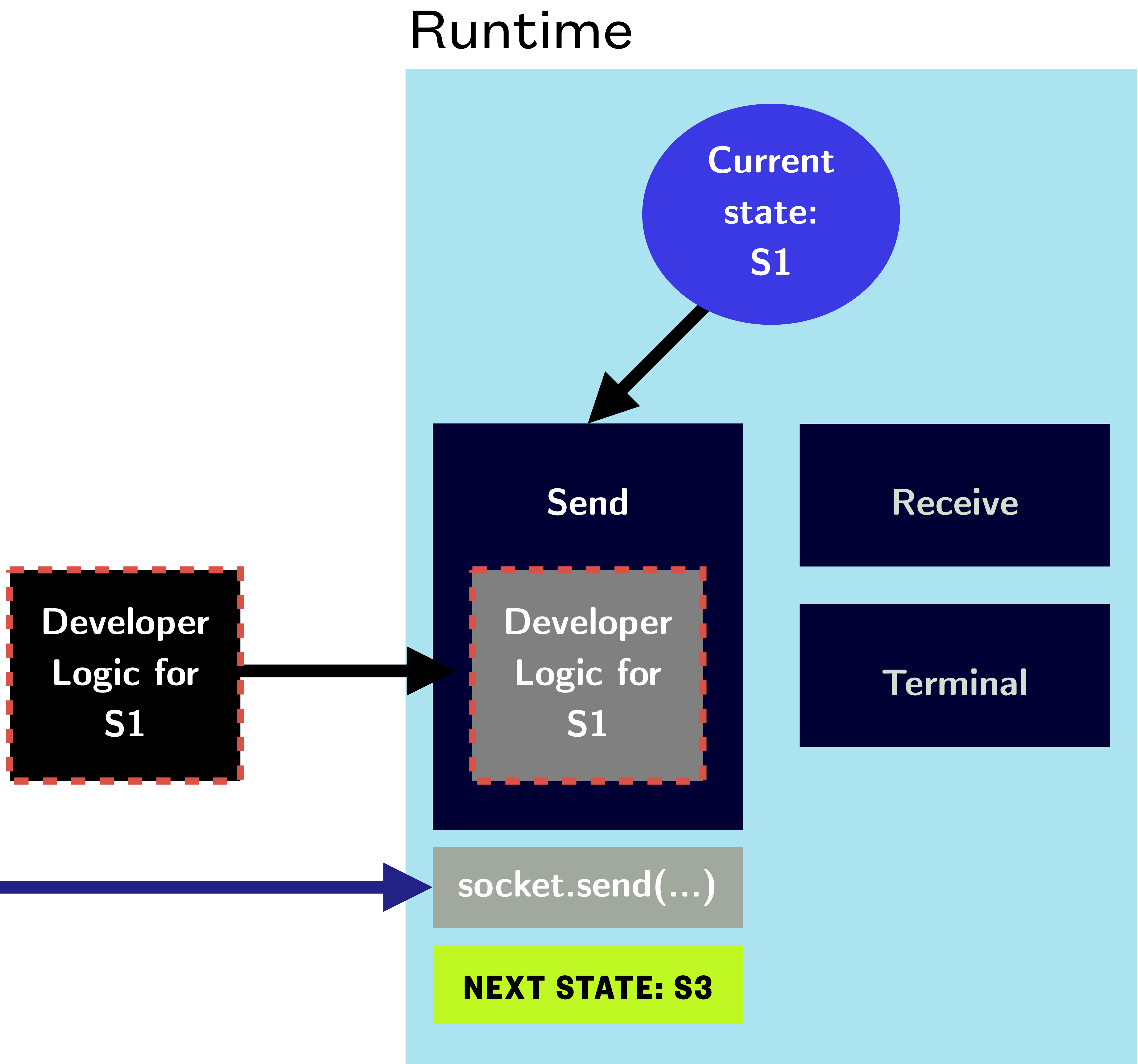
- We generate the session runtime to execute EFSM
  - Performs I/O action for current state
- We construct types for injecting business logic
  - What to send? How to handle receive?
- Developer instantiates session runtime with custom implementations



Design Philosophy

- We generate the session runtime to execute EFSM
  - Performs I/O action for current state
- We construct types for injecting business logic
  - What to send? How to handle receive?
- Developer instantiates session runtime with

Channel resources are not exposed, so channel reuse is **impossible** by construction.



EFSM in Node

NodeMPST

- Send states = an union of selections
  - Selection :: (label, payload, successor)
- Receive states = labelled handlers
  - Handler :: payload → Successor

```
const logic = new Implementation.Initial({
  [Labels.S17.Destination]: async (dest) => {
    const result = await checkAvailable(dest);
    if (result.available) {
      return new Implementation.S19([
        Labels.S19.Available, [result.price], ...
      ]);
    } else {
      return new Implementation.S19([
        Labels.S19.Full, [], logic
      ]);
    }
  }
});
```

EFSM in Node

NodeMPST

- Send states = an union of selections
  - Selection :: (label, payload, successor)
- Receive states = labelled handlers
  - Handler :: payload → Successor

```
const logic = new Implementation.Initial({
  [Labels.S17.Destination]: async (dest) => {
    const result = await checkAvailable(dest);
    if (result.available) {
      return new Implementation.S19([
        Labels.S19.Available, [result.price], ...
      ]);
    } else {
      return new Implementation.S19([
        Labels.S19.Full, [], logic
      ]);
    }
  }
});
```

Session Types for GUI

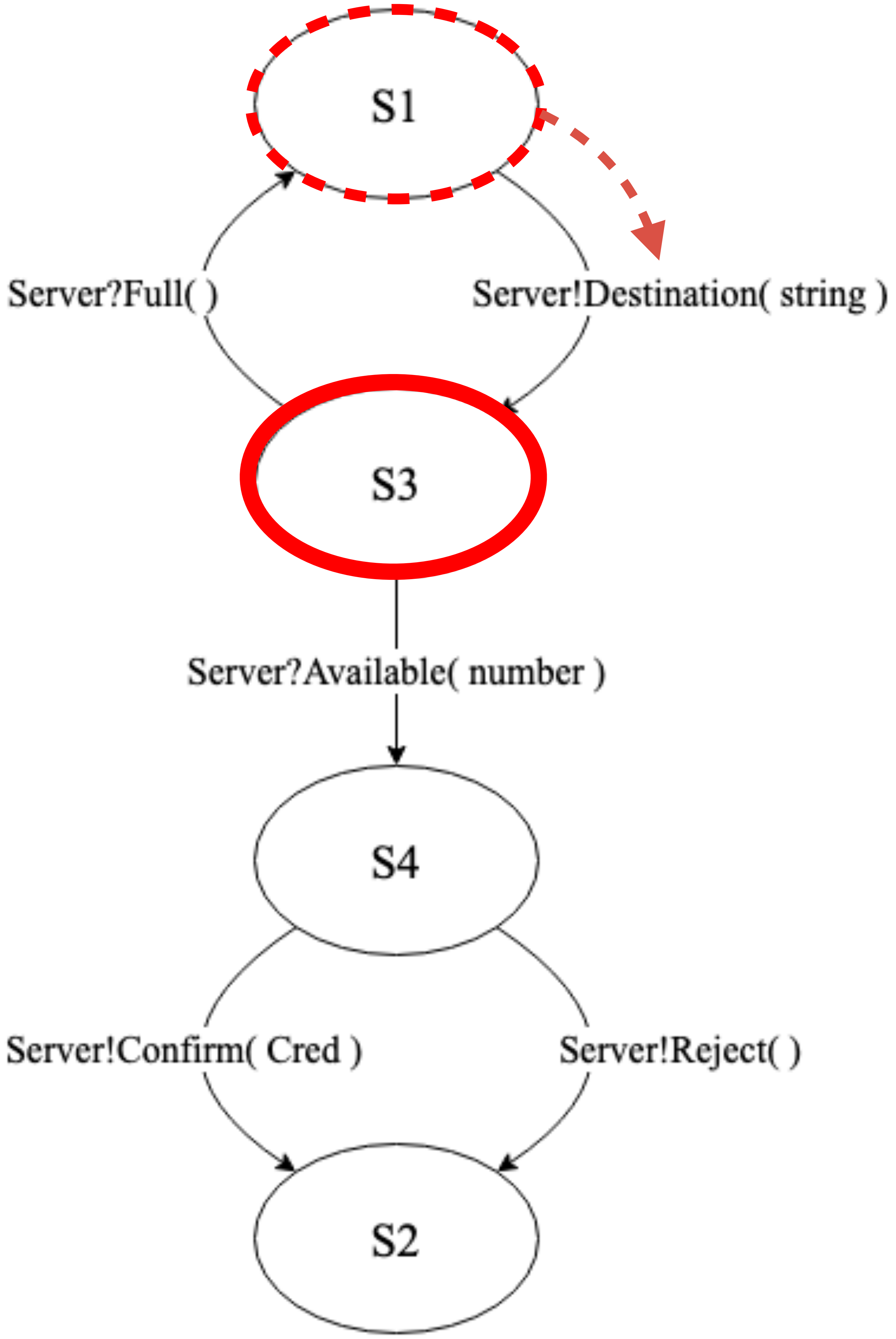
- Channel actions triggered by user interaction
  - User clicks button
  - User presses “Enter” on their keyboard
  - User hovers over HTML element, etc.

Session Types for GUI

ReactMPST

- Channel actions triggered by user interaction
  - User clicks button
  - User presses "Enter" on their keyboard
  - User hovers over HTML element, etc.

How to guarantee that user respects channel linearity?



Model-View-Update (MVU)

ReactMPST

- Model-View-Update (MVU)
- Each model **uniquely** defines:
  - Set of messages (e.g. "onClick")
  - View function (UI)
- The update function defines valid transitions (model x message) to other model types

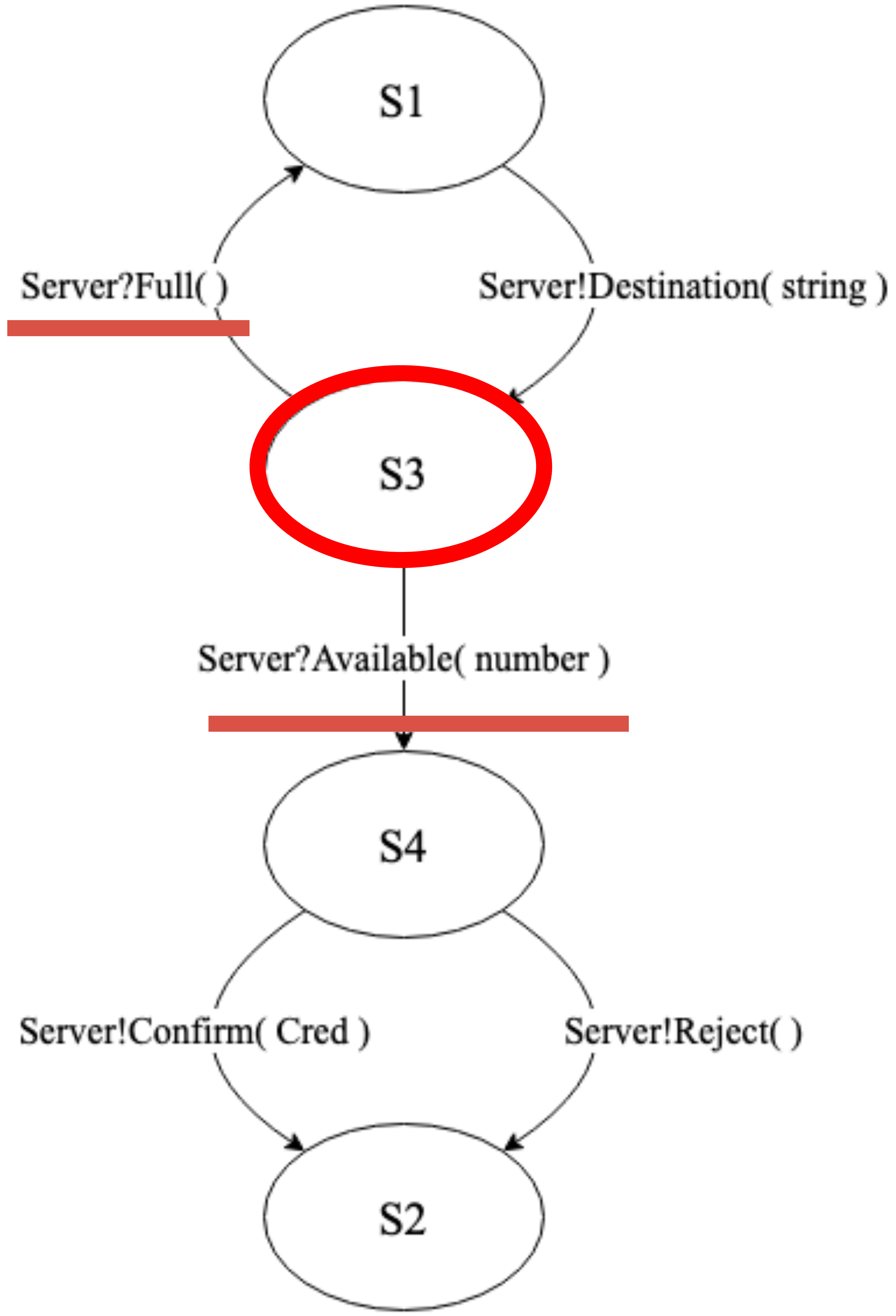
Correspondence between MVU, EFSM and React Components

FOWLER, S. Model-View-Update-Communicate: Session Types meet the Elm Architecture. *ECOOP 2020*.

MVU + Endpoint FSM

ReactMPST

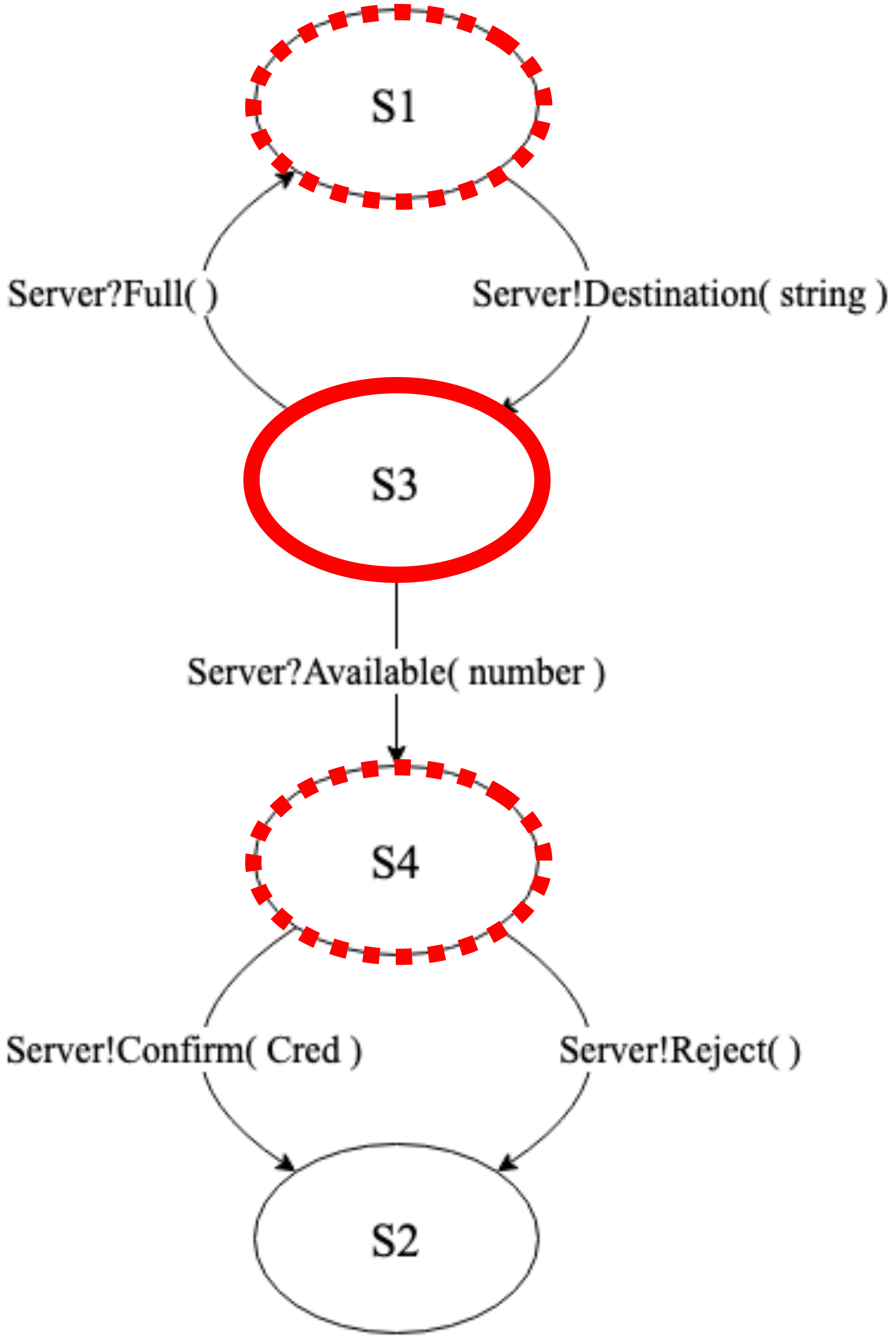
- Model-View-Update (MVU)
- Each model **uniquely** defines:
  - Set of messages (e.g. "onClick")
  - View function (UI)
- The update function defines valid transitions (model x message) to other model types



MVU + Endpoint FSM

ReactMPST

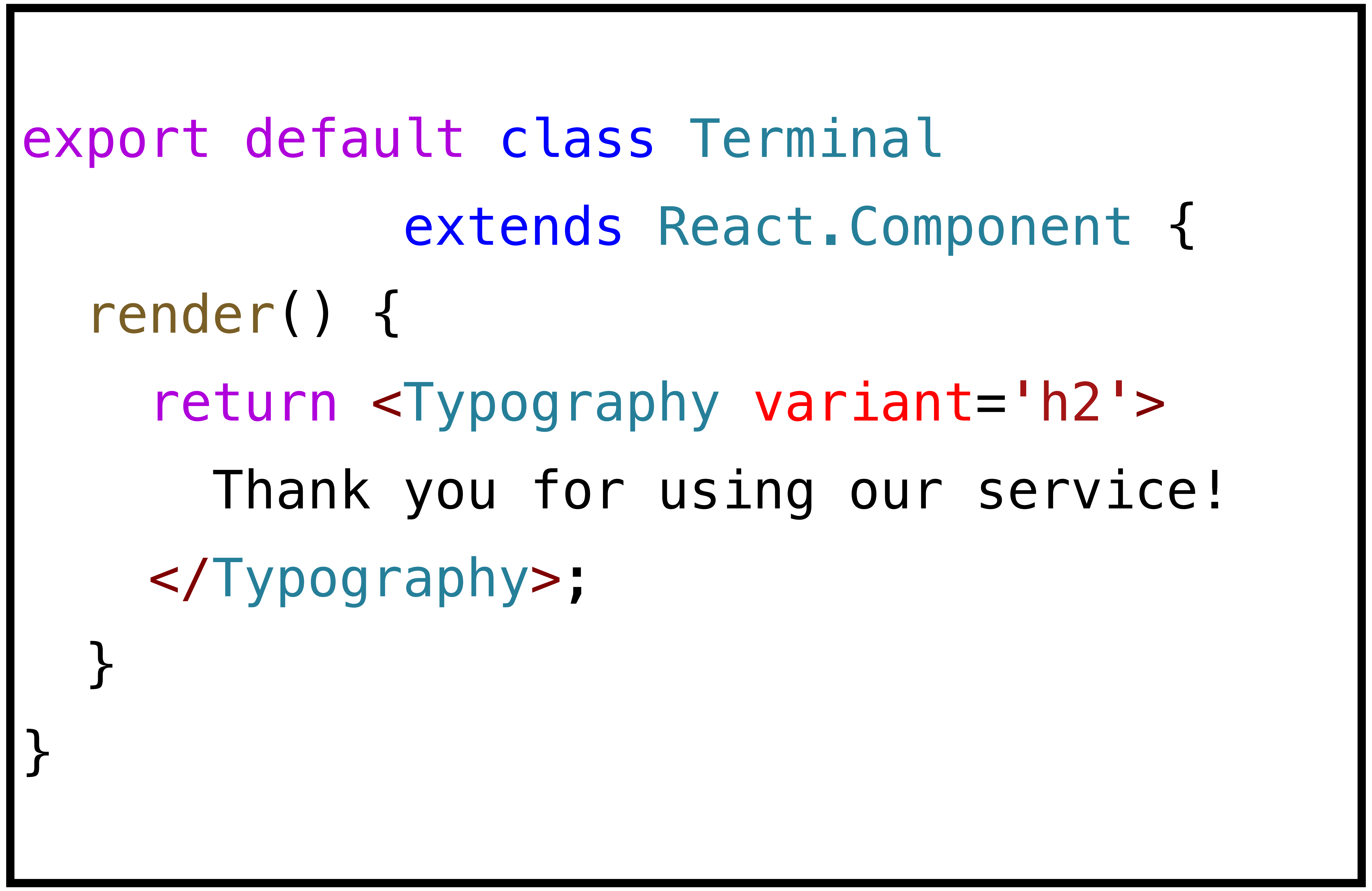
- Model-View-Update (MVU)
- Each model **uniquely** defines:
  - Set of messages (e.g. “onClick”)
  - View function (UI)
- The update function defines valid transitions (model x message) to other model types



MVU + React

ReactMPST

- Model-View-Update (MVU)
- Each model **uniquely** defines:
  - Set of messages (e.g. “onClick”)
  - View function (UI)
- The update function defines valid transitions (model x message) to other model types



```
export default class Terminal
               extends React.Component {
  render() {
    return <Typography variant='h2'>
      Thank you for using our service!
    </Typography>;
  }
}
```

EFSM in React

ReactMPST

- EFSM states = abstract React components
  - Developer implements the view function
- Send action = component factory
  - I/O bound to UI event on component
- Receive action = callback
  - Abstract method

EFSM in React

ReactMPST

- EFSM states = abstract React components
  - Developer implements the view function
- Send action = component factory
  - I/O bound to UI event on component
- Receive action = callback
  - Abstract method

```
const London = this.Destination('onClick', ev => {
    this.context.setDestination('London');
    return ['London'];
});

return (<div>
    ...
    <London>
        <Button size="small" color=“primary">
            Enquire
        </Button>
    </London>
    ...
</div>);
```

EFSM in React

ReactMPST

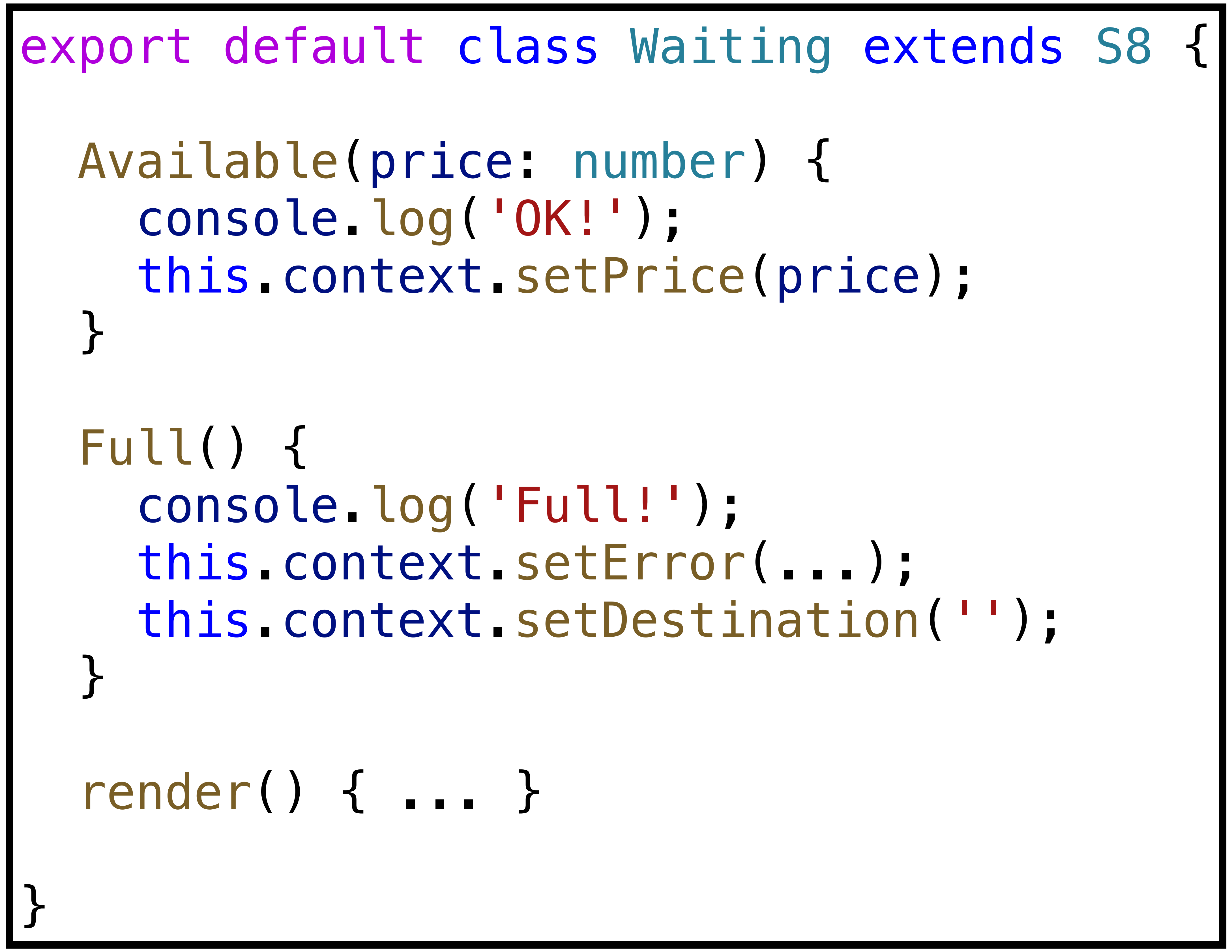
- EFSM states = abstract React components
  - Developer implements the view function
- Send action = component factory
  - I/O bound to UI event on component
- Receive action = callback
  - Abstract method

```
const London = this.Destination('onClick', ev => {
    this.context.setDestination('London');
    return ['London'];
});

return (<div>
    ...
    <London>
        <Button size="small" color=“primary">
            Enquire
        </Button>
    </London>
    ...
</div>);
```

EFSM in React

ReactMPST

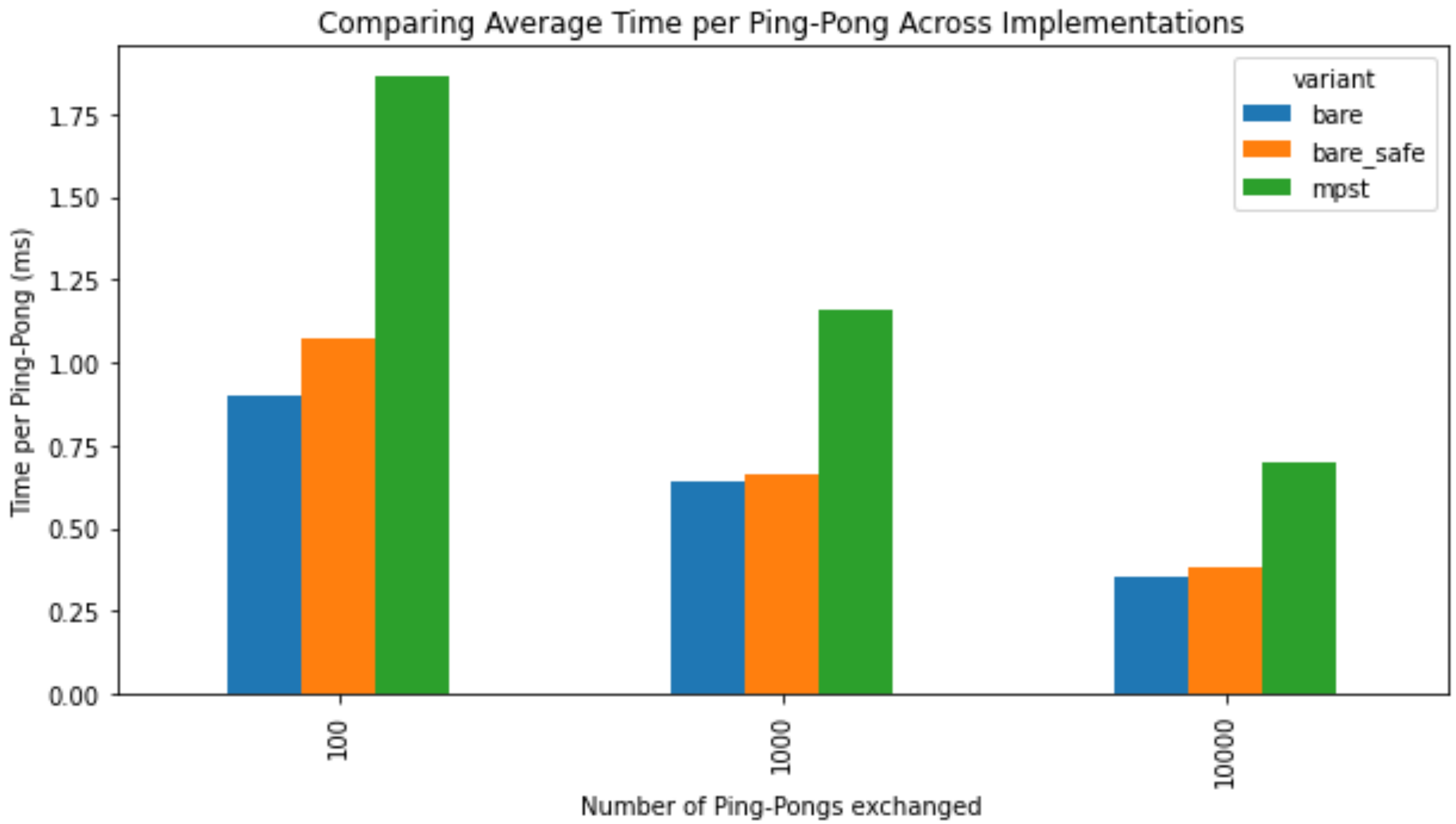
- EFSM states = abstract React components
  - Developer implements the view function
- Send action = component factory
  - I/O bound to UI event on component
- Receive action = callback
  - Abstract method



```
export default class Waiting extends S8 {

  Available(price: number) {
    console.log('OK!');
    this.context.setPrice(price);
  }

  Full() {
    console.log('Full!');
    this.context.setError(...);
    this.context.setDestination('');
  }

  render() { ... }

}
```

Evaluation

- **Expressiveness**
  - Flight Booking Service
  - Noughts and Crosses
- **Performance**
  - Micro-benchmarks of Ping Pong protocol with varying number of round trips
  - Overhead in message processing time



https://github.com/ansonmiu0214/SessionTS-Examples

https://github.com/ansonmiu0214/SessionTS-Benchmarks

Contributions

Type-safe Web Programming Using
Routed Multiparty Session Types **in TypeScript**

1. **SessionTS: a Session Type API Code Generation Toolchain for Modern Web Programming**
   Targets standard industrial strength technologies (TypeScript, Node.js, React.js) and idiomatic web programming practices (event-driven, callback-based, asynchronous).

2. **RoutedSessions: a New Theory of Multiparty Session Types with Routed Communication**
   Formalised new theory, proved correctness, implemented in SessionTS to support peer-to-peer interactions over server-centric network structures

Contributions

Type-safe Web Programming Using
Routed Multiparty Session Types in TypeScript

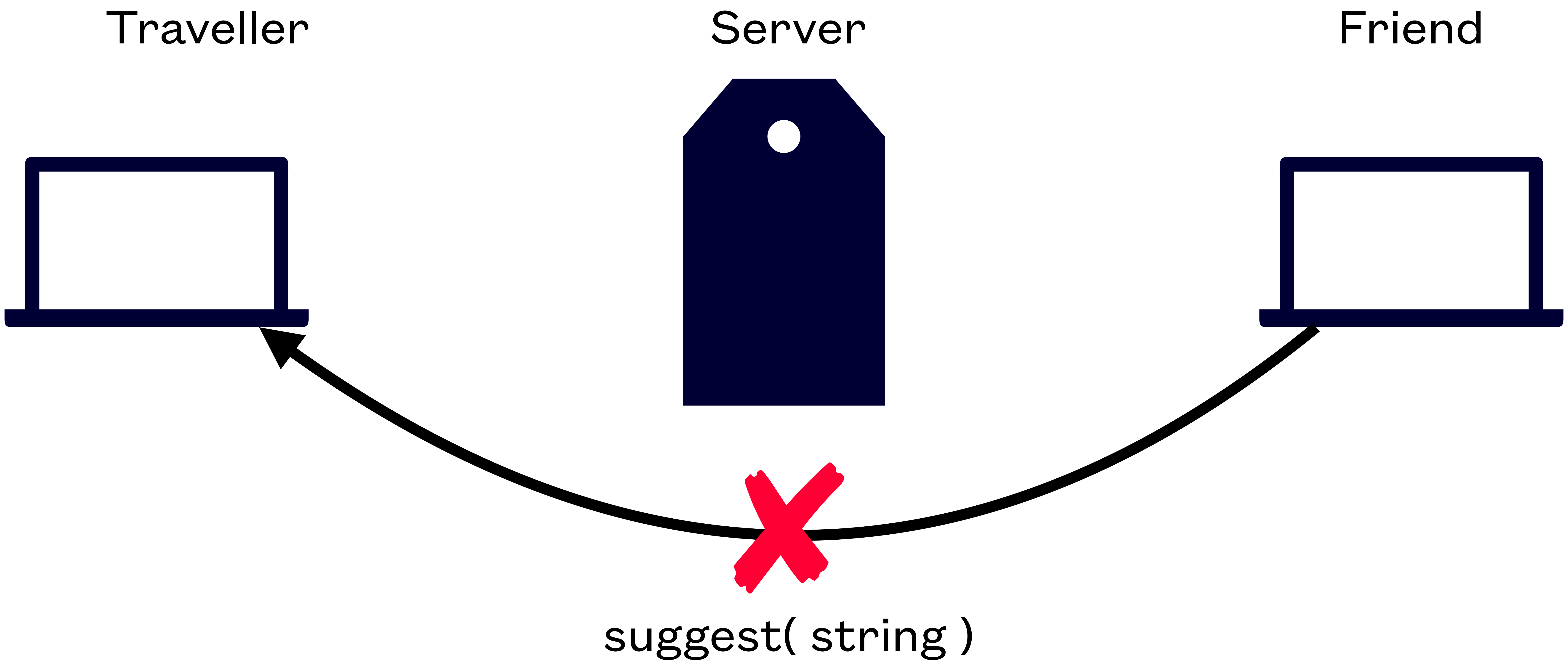
1. SessionTS: a Session Type API Code Generation Toolchain for Modern Web Programming
   Targets standard industrial strength technologies (TypeScript, Node.js, React.js) and idiomatic web programming practices (event-driven, callback-based, asynchronous)

2. **ROUTEDSESSIONS: a New Theory of Multiparty Session Types with Routed Communication**
   Formalised new theory, proved correctness, implemented in SessionTS to support peer-to-peer interactions over server-centric network structures

RoutedSessions

A New Theory of Multiparty Session Types with Routed Communication

Intuition



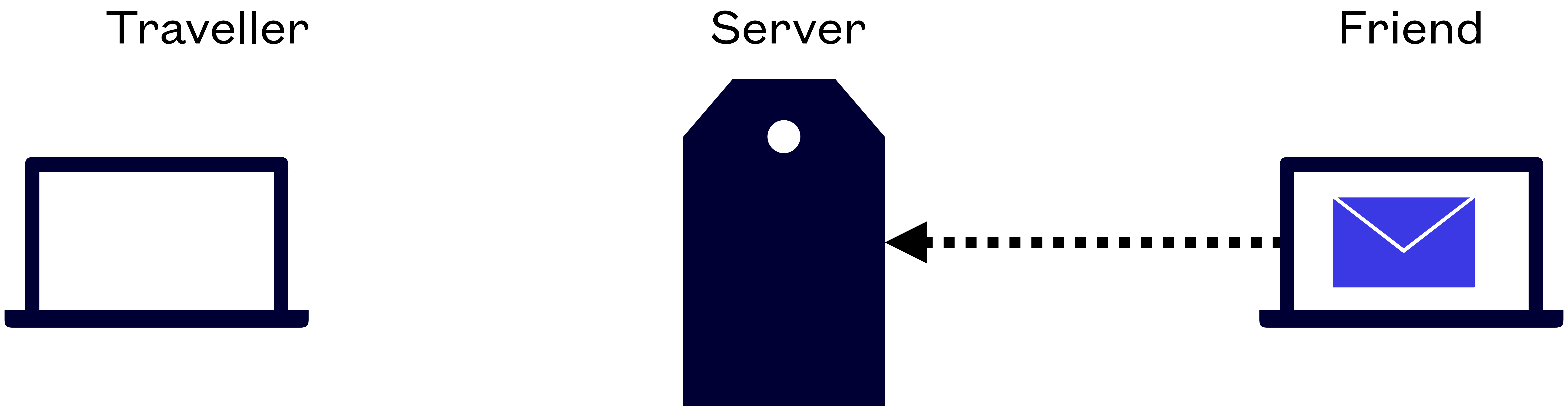
Traveller

Server

Friend

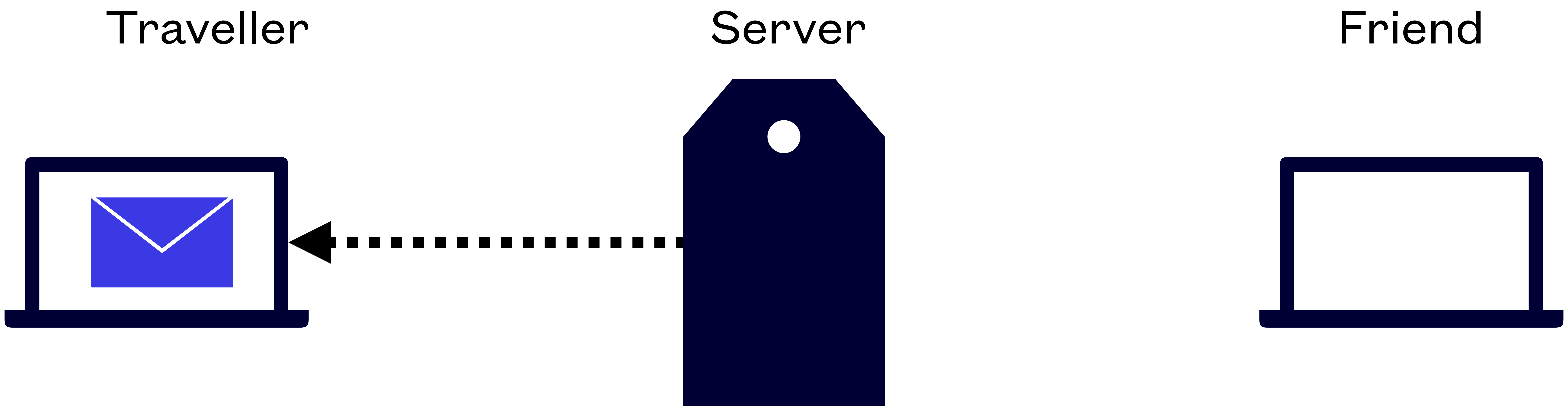
suggest(string)

Intuition



Traveller!suggest(string)

Intuition

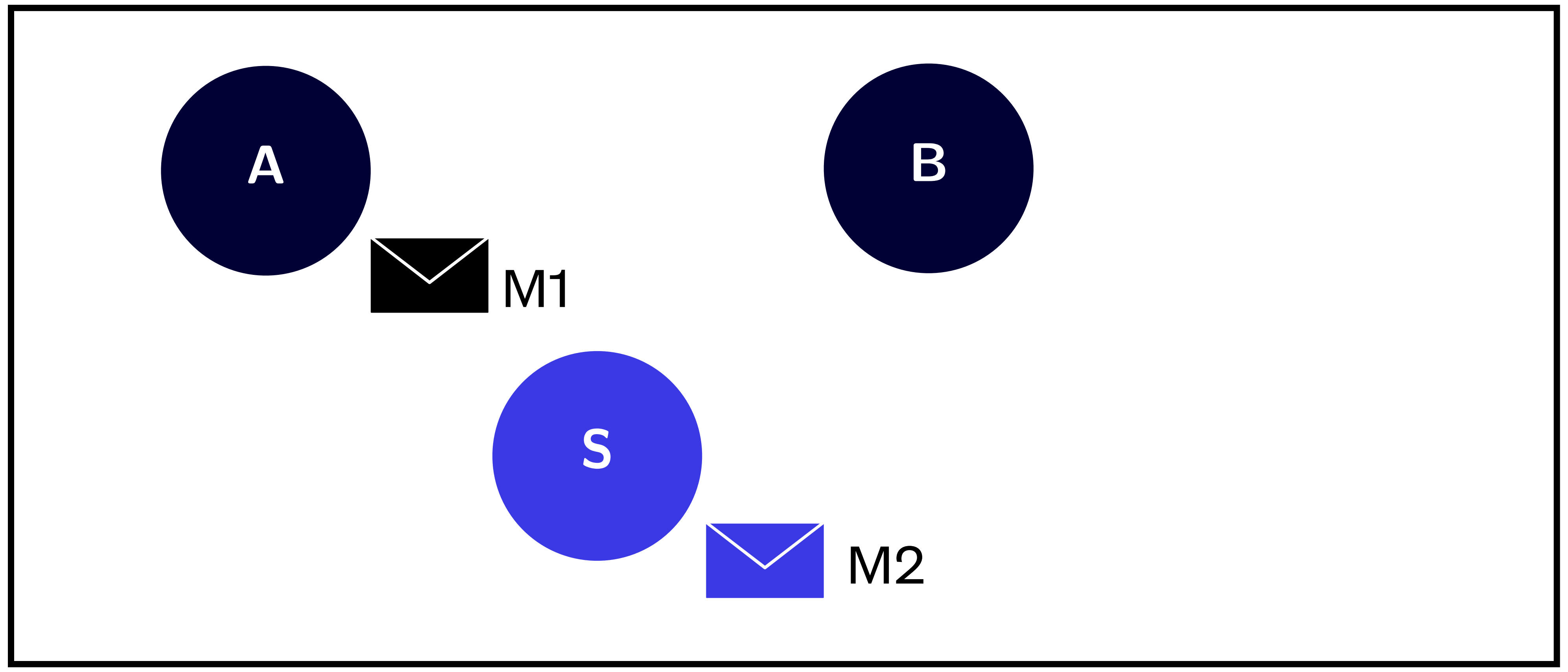


Traveller!suggest(string)

Challenges in Formalising “Routing”

“Original communication”

- A sends M1 to B.
- S sends M2 to B.

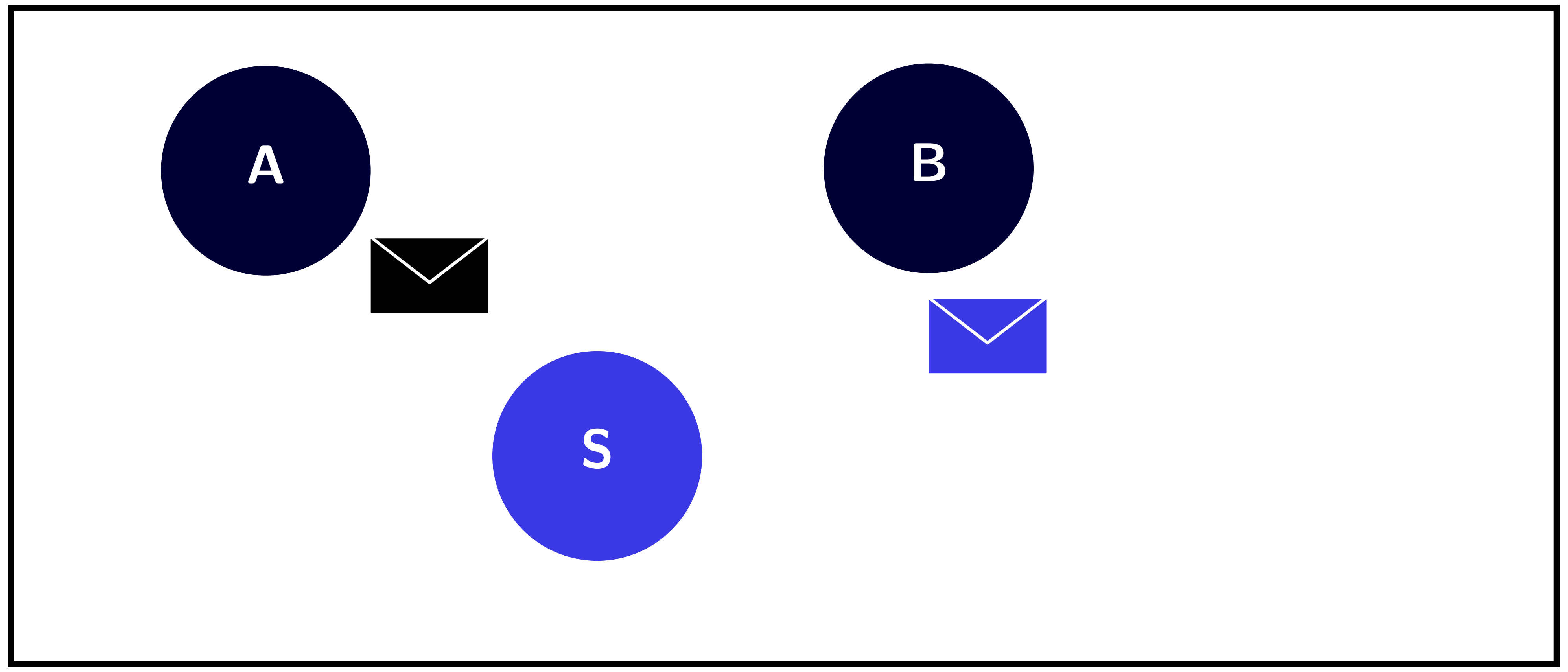


DENI´ELOU, P.-M., AND YOSHIDA, N. Multiparty Compatibility in Communicating Automata: Characterisation and Synthesis of Global Session Types. ICALP 2013.

Challenges in Formalising “Routing”

“Original communication”

- A sends M1 to B.
- **S sends M2 to B.**

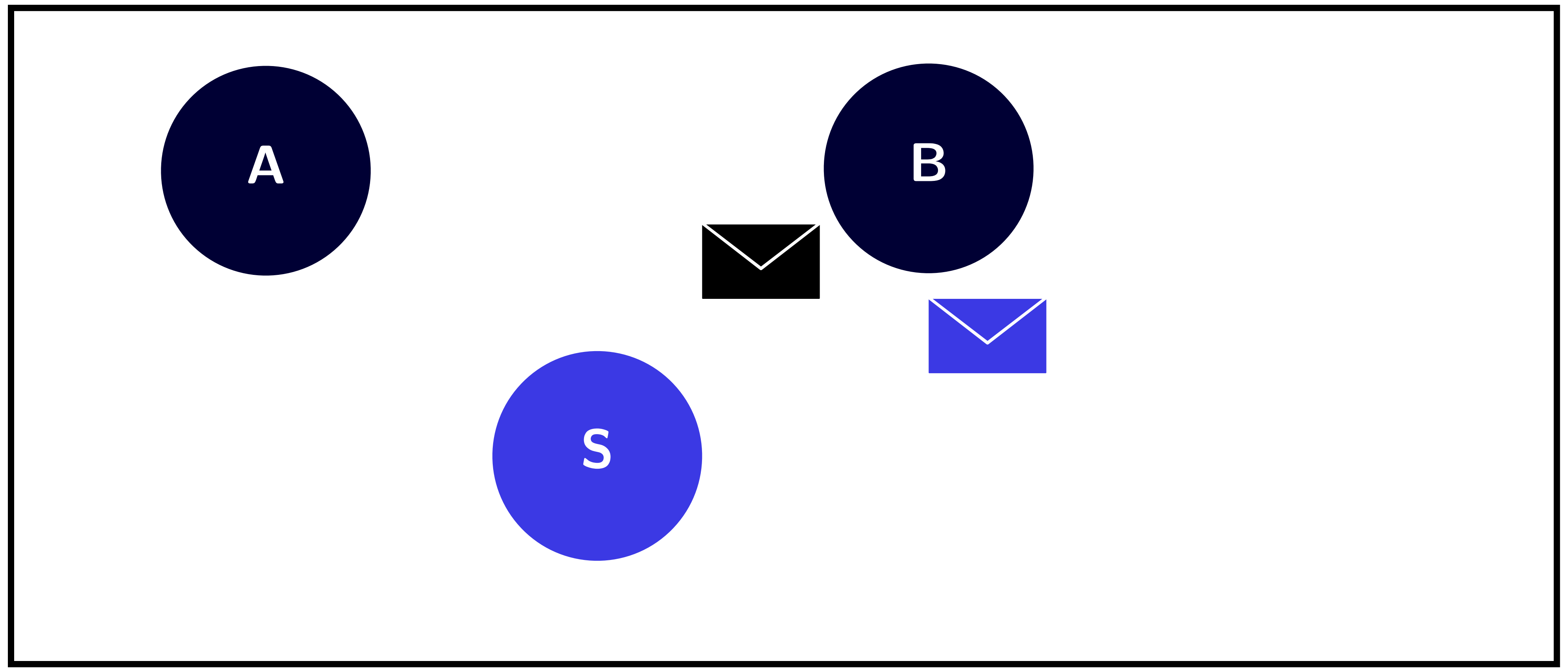


DENI´ELOU, P.-M., AND YOSHIDA, N. Multiparty Compatibility in Communicating Automata: Characterisation and Synthesis of Global Session Types. ICALP 2013.

Challenges in Formalising “Routing”

“Original communication”

- **A sends M1 to B.**
- **S sends M2 to B.**

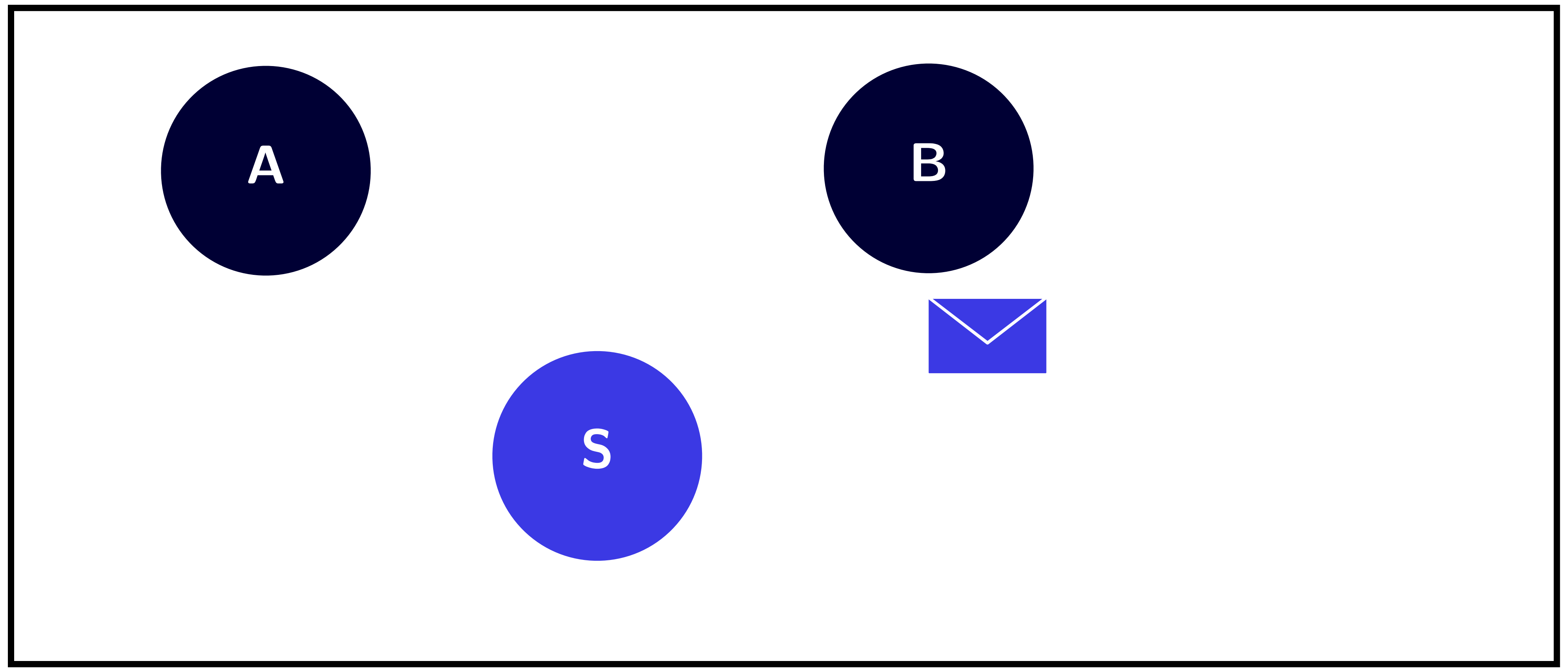


DENI´ELOU, P.-M., AND YOSHIDA, N. Multiparty Compatibility in Communicating Automata: Characterisation and Synthesis of Global Session Types. ICALP 2013.

Challenges in Formalising "Routing"

"Original communication"

- A sends M1 to B.
- **S sends M2 to B.**

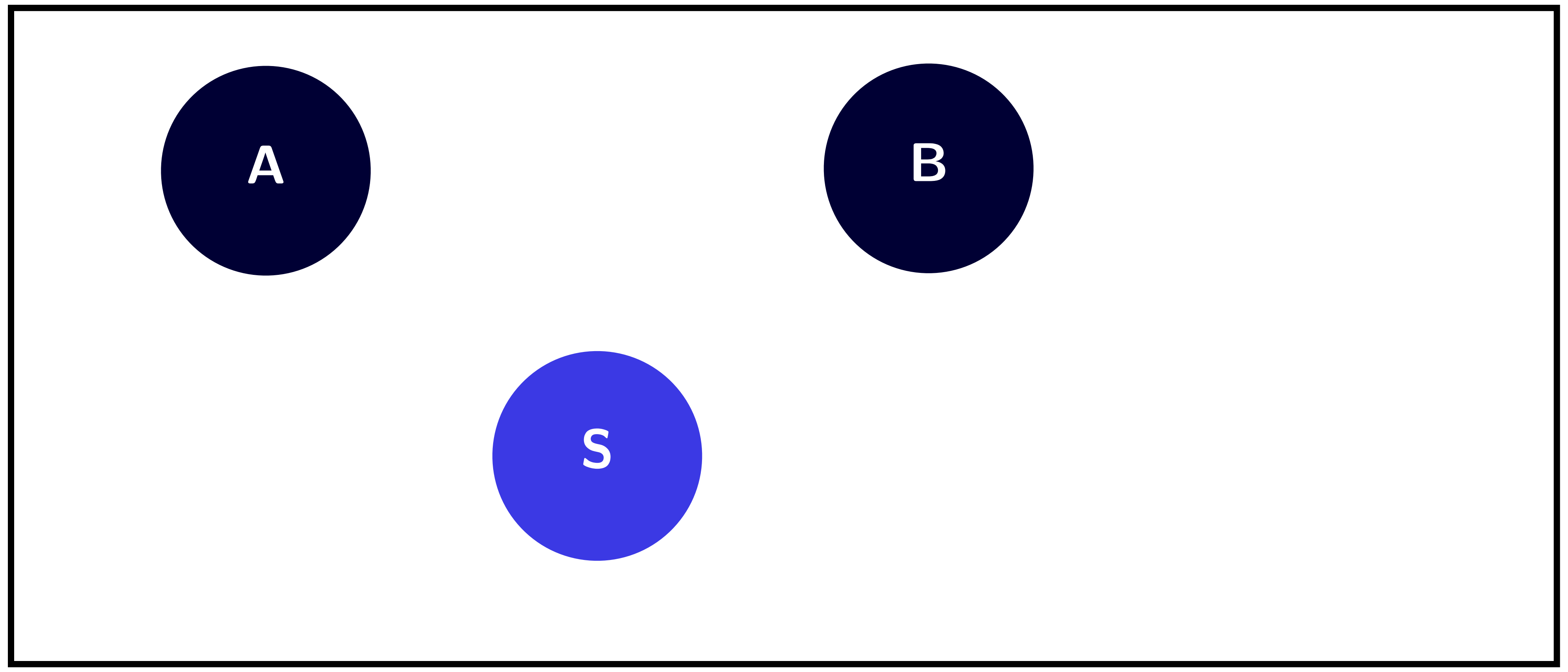


DENI´ELOU, P.-M., AND YOSHIDA, N. Multiparty Compatibility in Communicating Automata: Characterisation and Synthesis of Global Session Types. ICALP 2013.

Challenges in Formalising “Routing”

“Original communication”

- A sends M1 to B.
- S sends M2 to B.

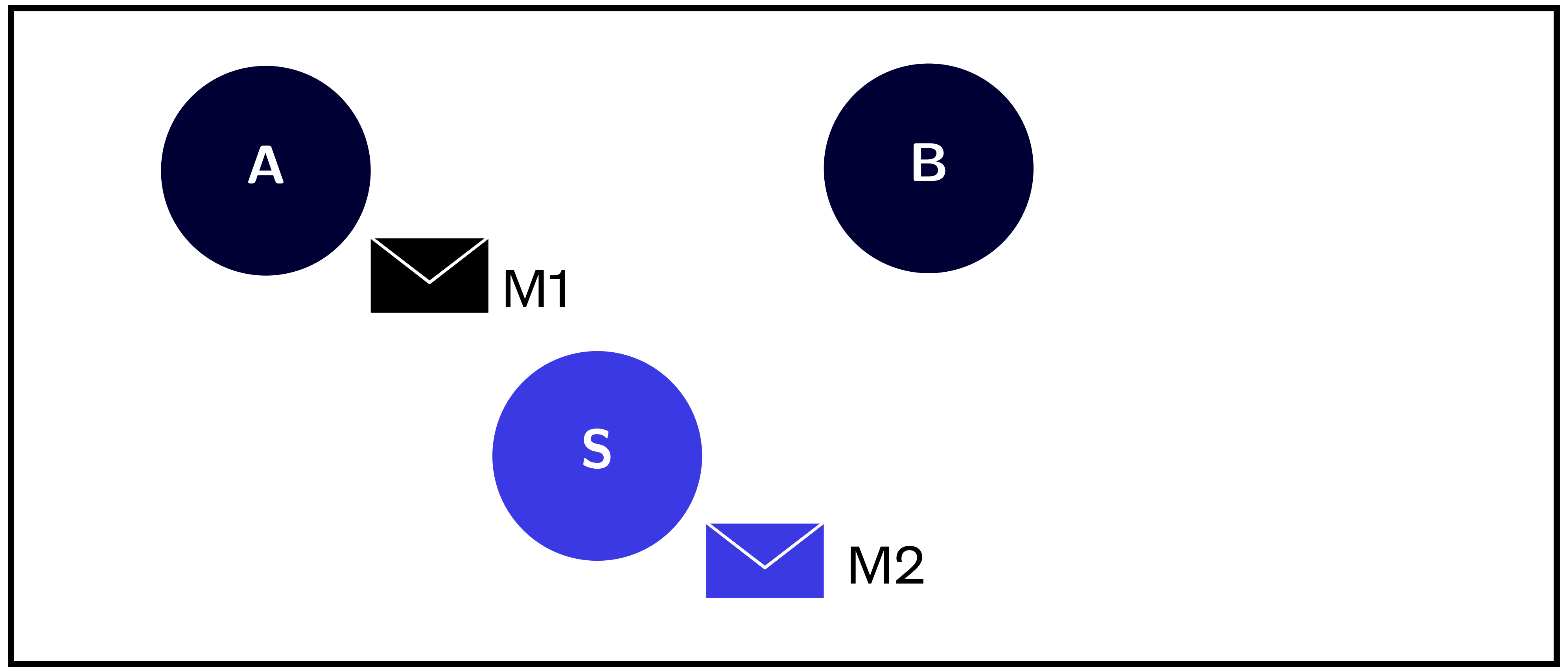


DENIÉLOU, P.-M., AND YOSHIDA, N. Multiparty Compatibility in Communicating Automata: Characterisation and Synthesis of Global Session Types. ICALP 2013.

Challenges in Formalising "Routing"

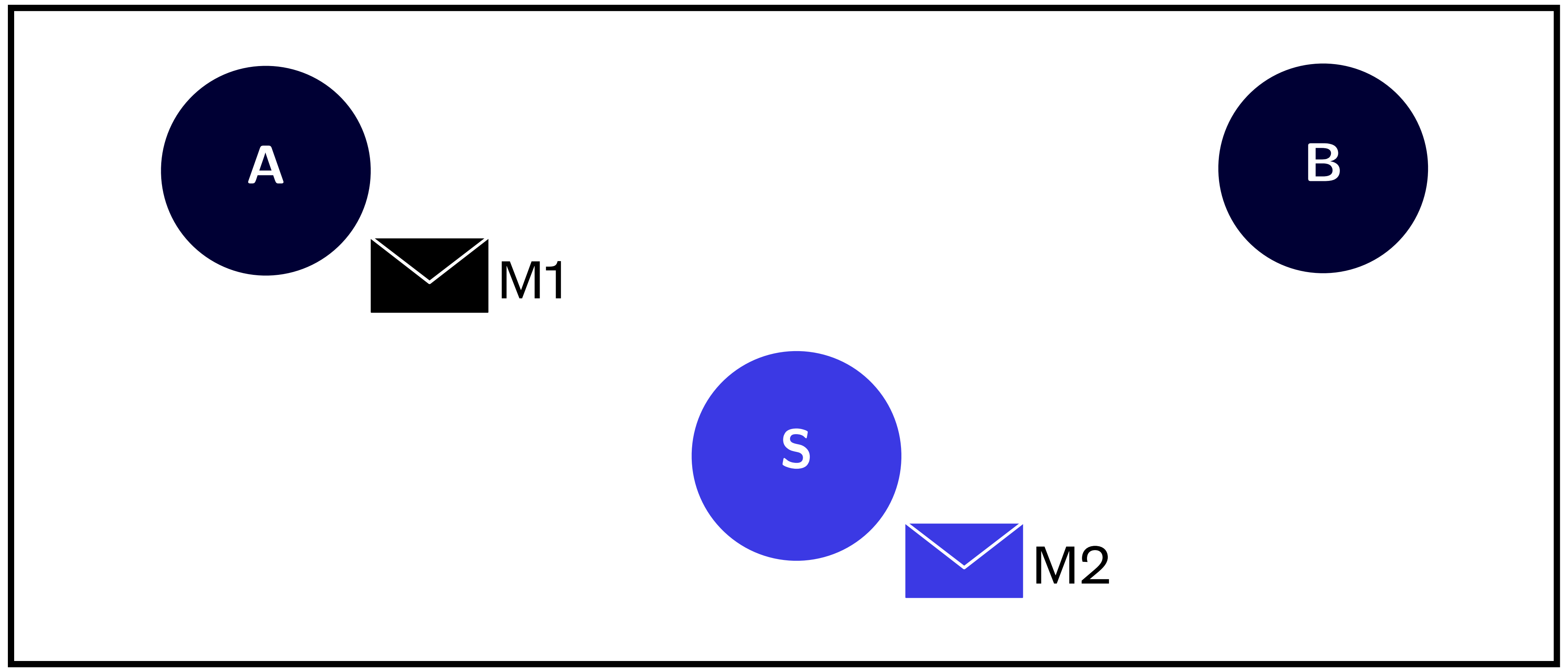
"Original communication"

- A sends M1 to B.
- S sends M2 to B.



"Naive Routed Communication"

- A sends M1 to S.
- S sends M1 to B.
- S sends M2 to B.

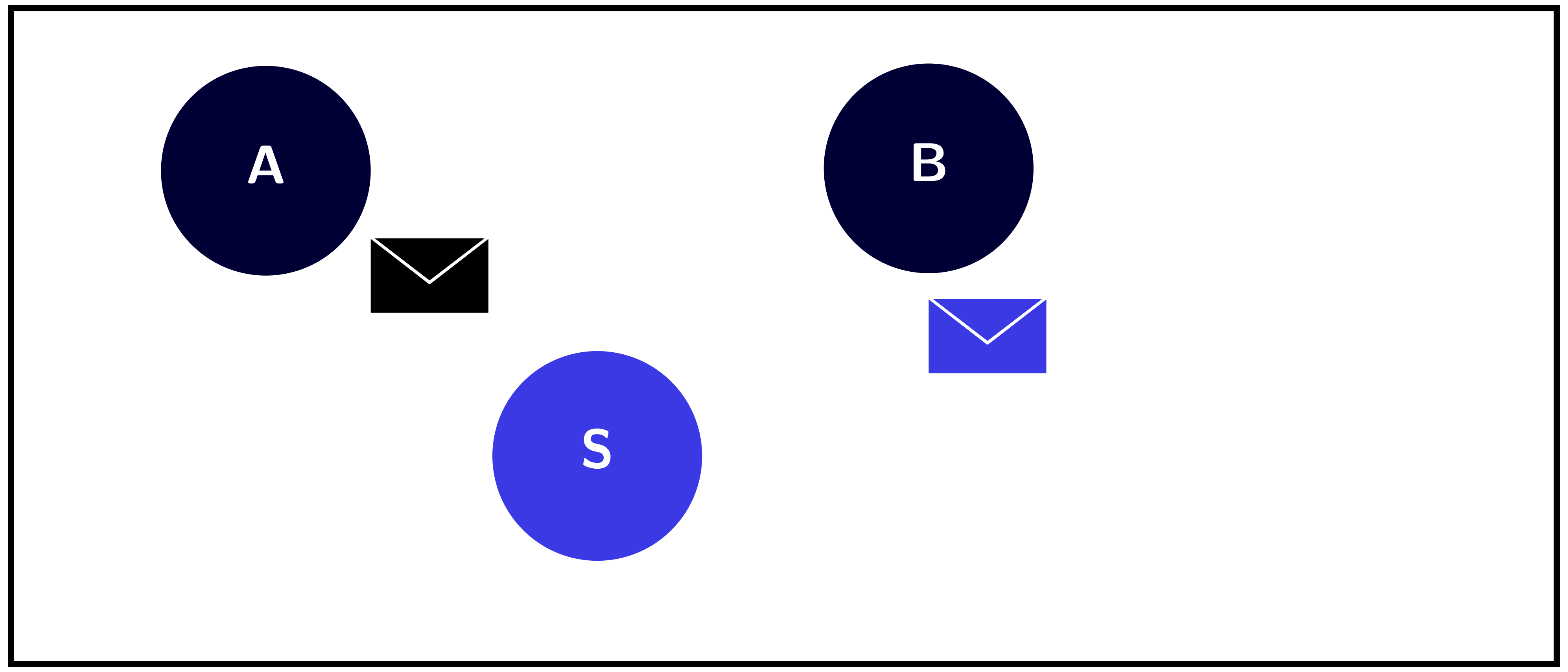


DENI´ELOU, P.-M., AND YOSHIDA, N. Multiparty Compatibility in Communicating Automata: Characterisation and Synthesis of Global Session Types. ICALP 2013.

Challenges in Formalising “Routing”

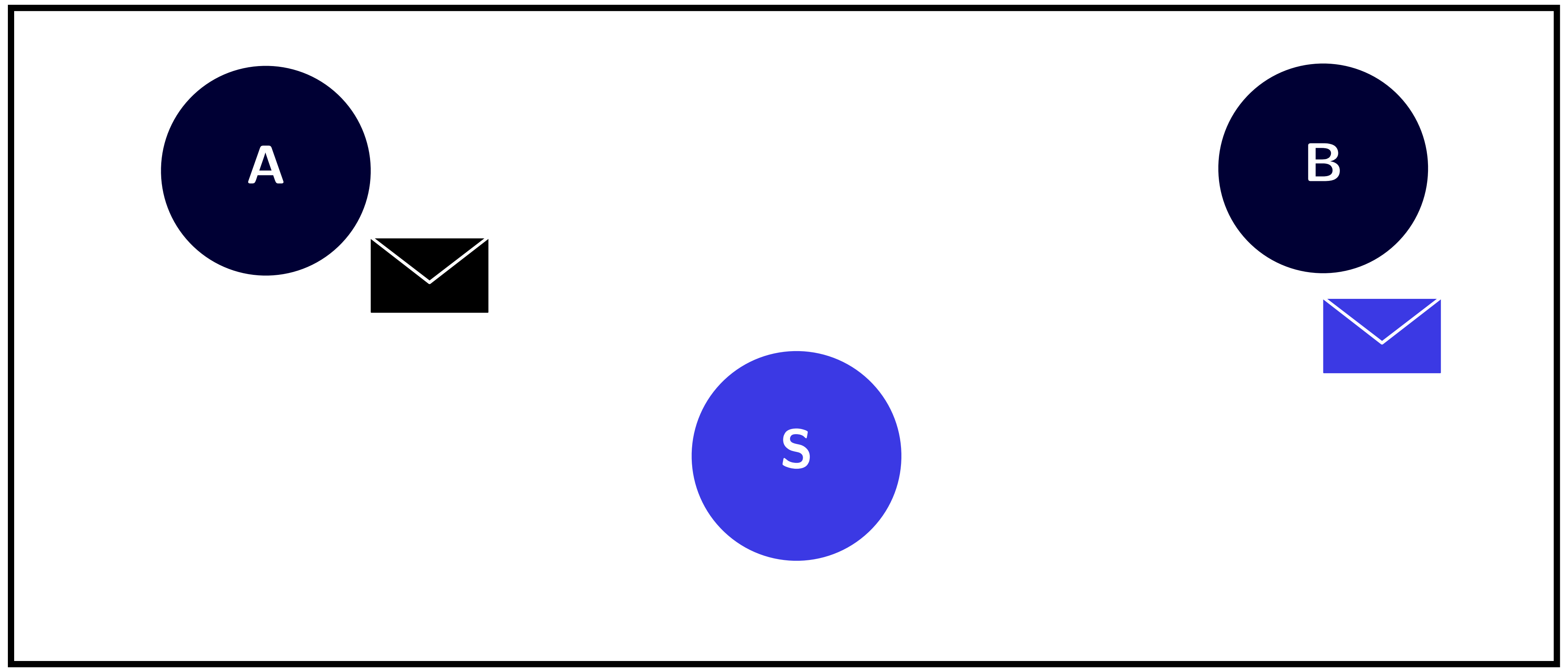
“Original communication”

- A sends M1 to B.
- **S sends M2 to B.**



“Naive Routed Communication”

- A sends M1 to S.
- S sends M1 to B.
- **S sends M2 to B.**

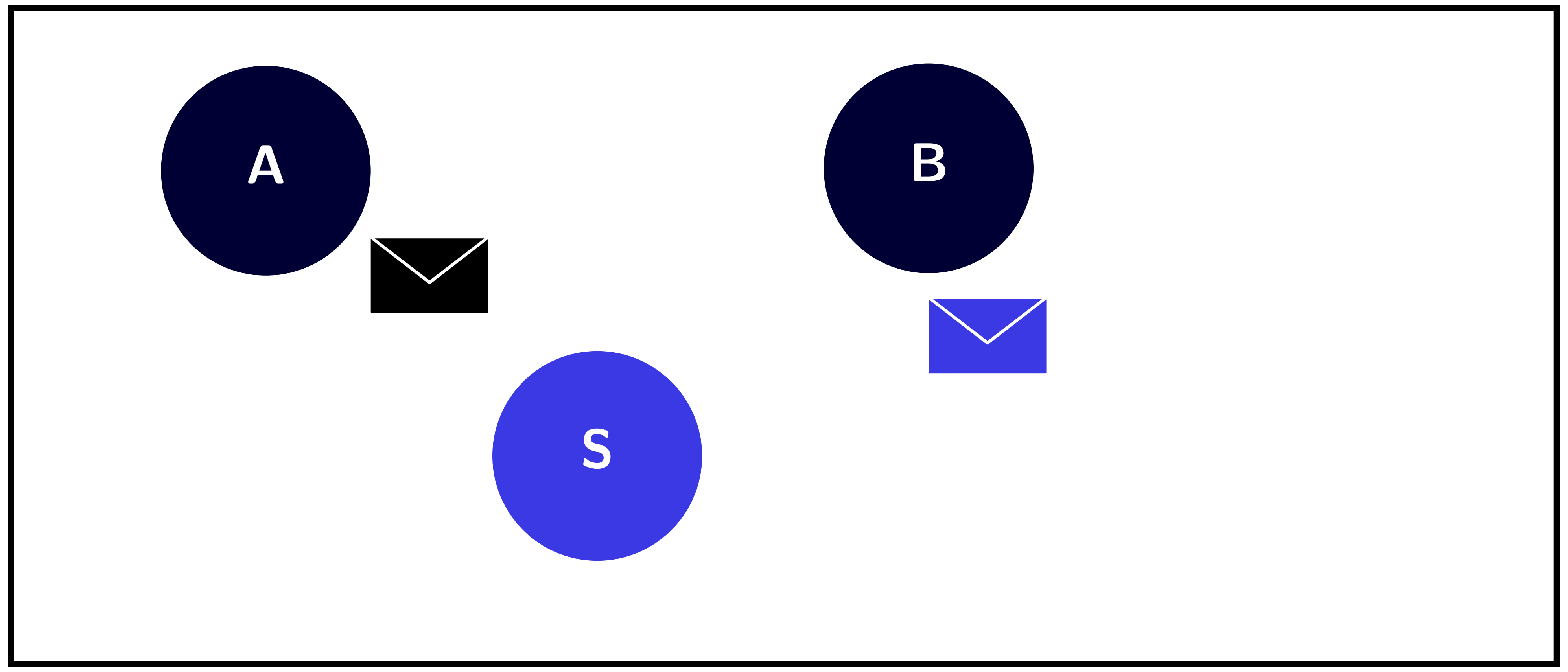


DENI´ELOU, P.-M., AND YOSHIDA, N. Multiparty Compatibility in Communicating Automata: Characterisation and Synthesis of Global Session Types. ICALP 2013.

Challenges in Formalising “Routing”

“Original communication”

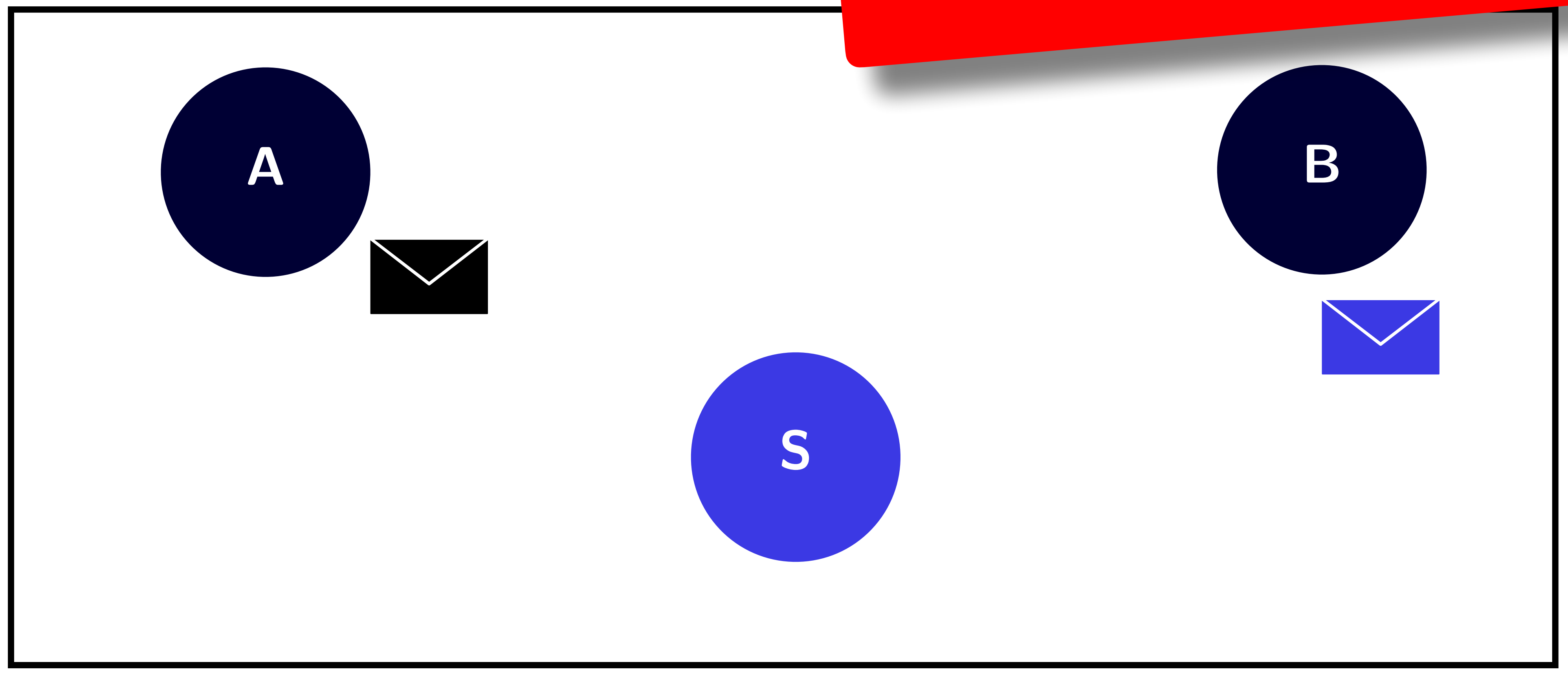
- A sends M1 to B.
- **S sends M2 to B.**



“Naive Routed Communication”

- A sends M1 to S.
- ***S sends M1 to B.***
- **S sends M2 to B.**

Not Possible!

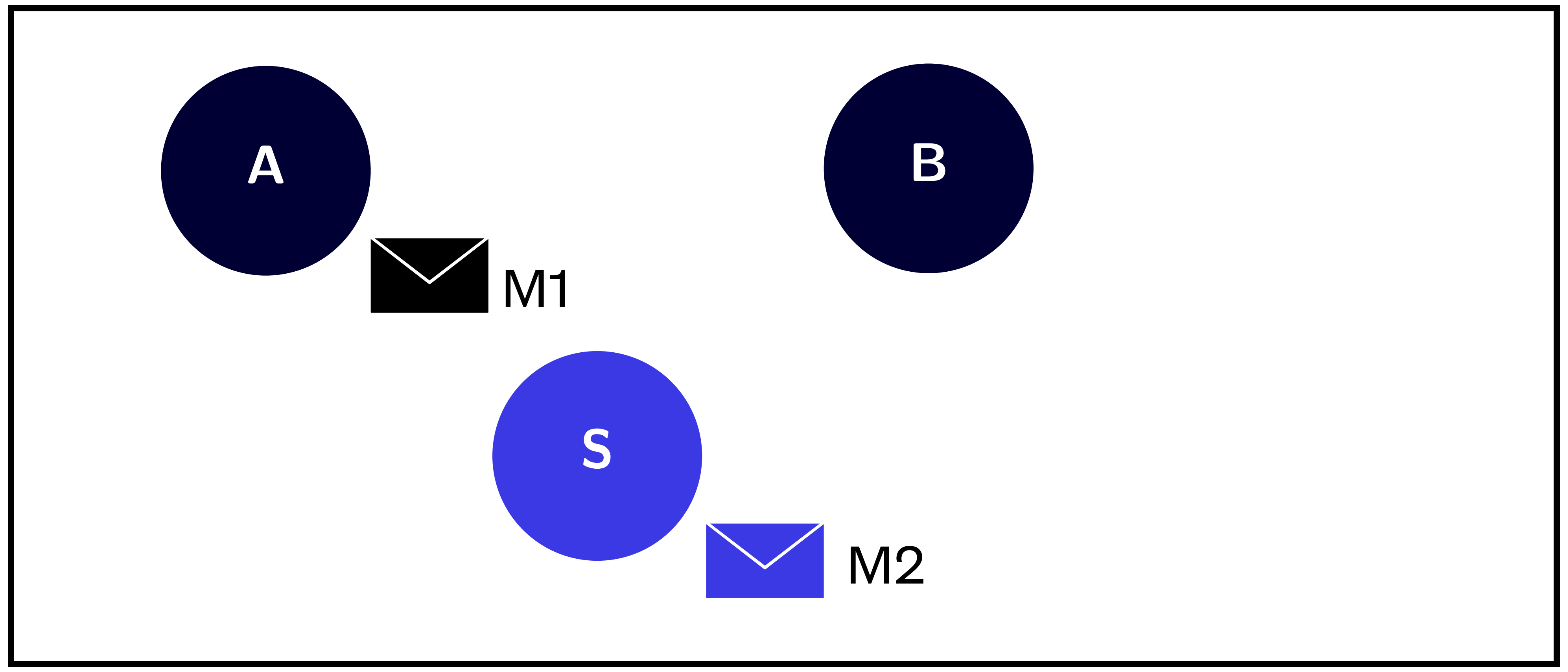


DENI´ELOU, P.-M., AND YOSHIDA, N. Multiparty Compatibility in Communicating Automata: Characterisation and Synthesis of Global Session Types. ICALP 2013.

Challenges in Formalising "Routing"

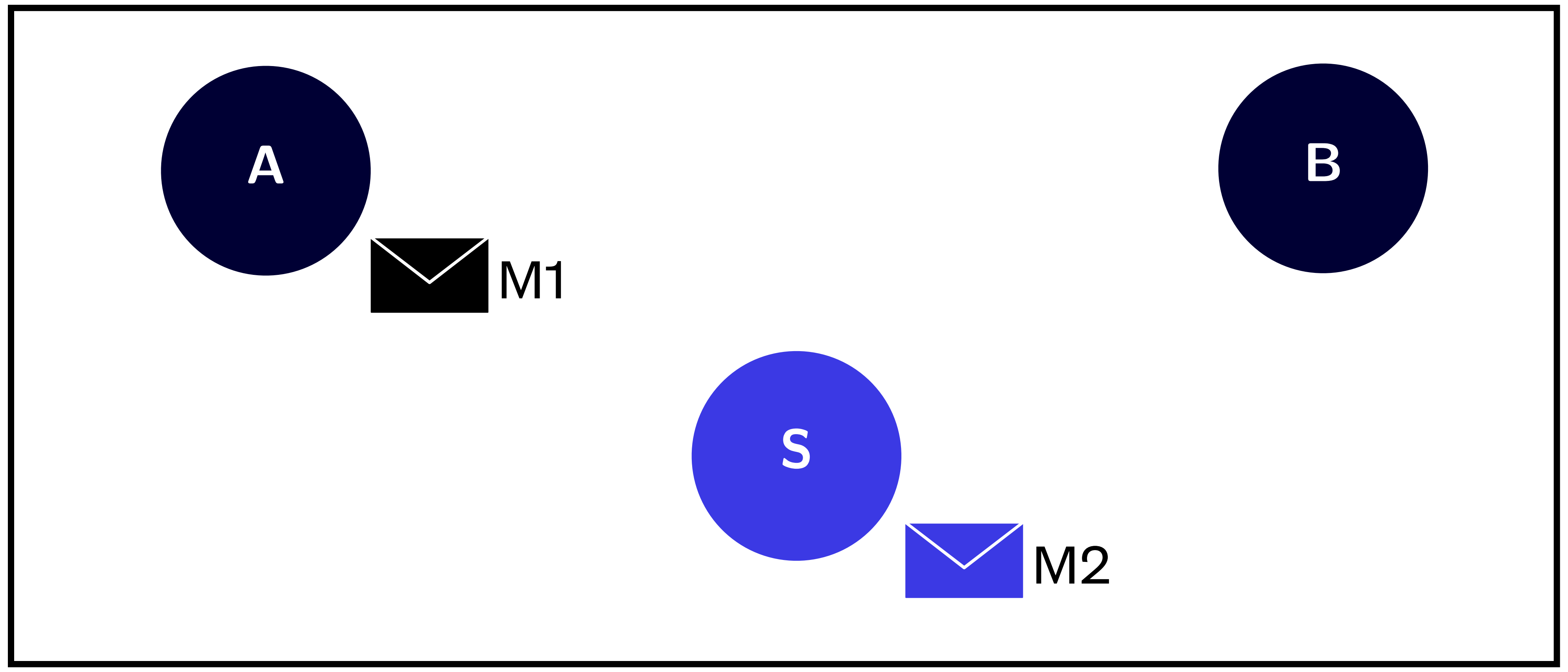
"Original communication"

- A sends M1 to B.
- S sends M2 to B.



ROUTEDSESSIONS

- A sends M1 to B via S.
- S sends M2 to B.

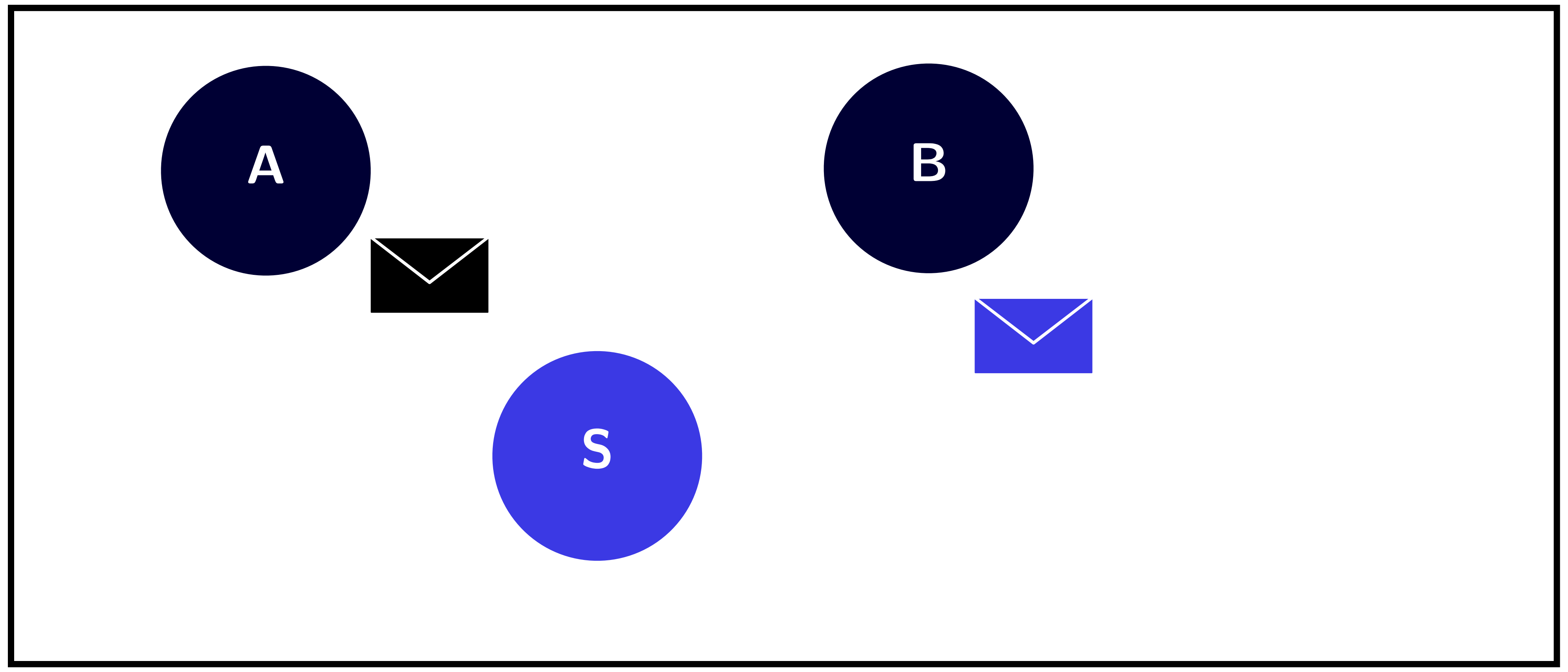


DENI´ELOU, P.-M., AND YOSHIDA, N. Multiparty Compatibility in Communicating Automata: Characterisation and Synthesis of Global Session Types. ICALP 2013.

Challenges in Formalising "Routing"

"Original communication"

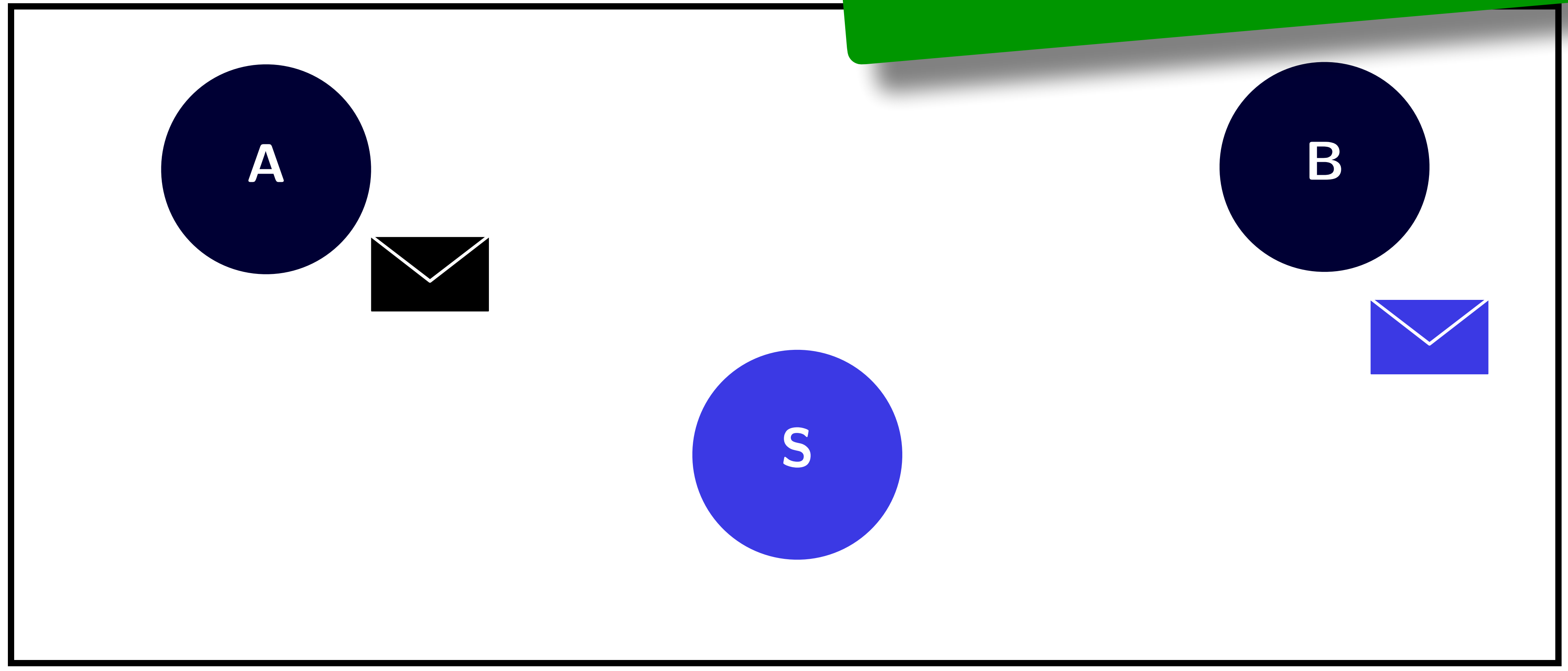
- A sends M1 to B.
- **S sends M2 to B.**



ROUTEDSESSIONS

- A sends M1 to B <u>via S</u>.
- **S sends M2 to B.**

via S allows this



DENI´ELOU, P.-M., AND YOSHIDA, N. Multiparty Compatibility in Communicating Automata: Characterisation and Synthesis of Global Session Types. ICALP 2013.

Formalising Routed Communication

"Original communication"

- A sends M1 to B.
- S sends M2 to B.

ROUTEDSESSIONS

- A sends M1 to B via S.
- S sends M2 to B.

Define new theory for ROUTEDSESSIONS

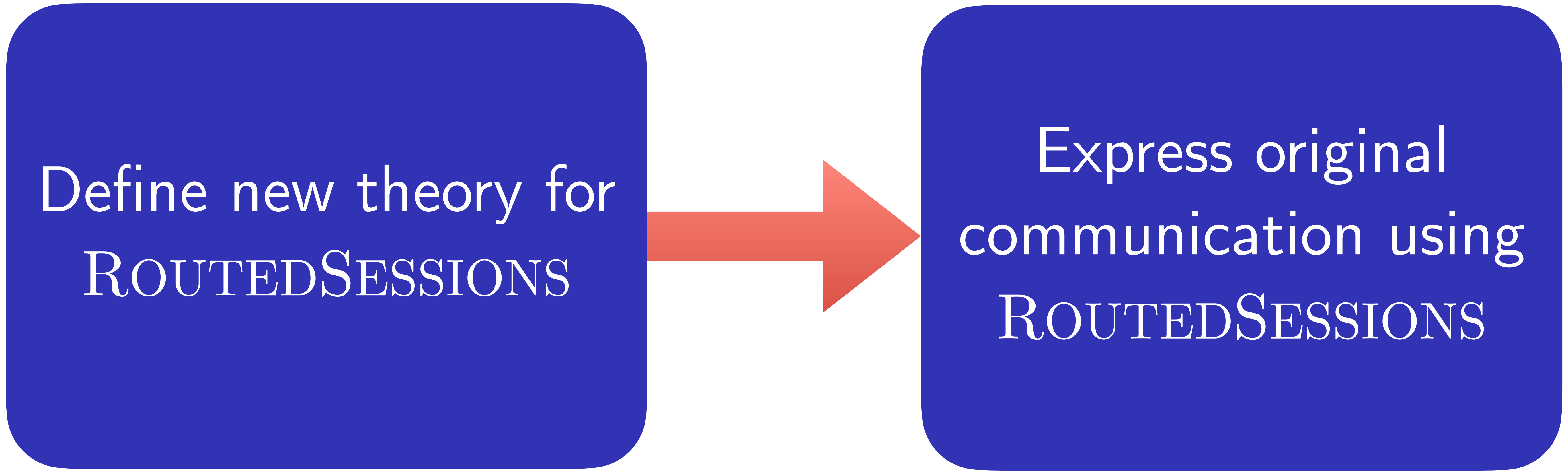
Formalising Routed Communication

"Original communication"

- A sends M1 to B.
- S sends M2 to B.

RoutedSessions

- A sends M1 to B via S.
- S sends M2 to B.



Define new theory for RoutedSessions

Express original communication using RoutedSessions

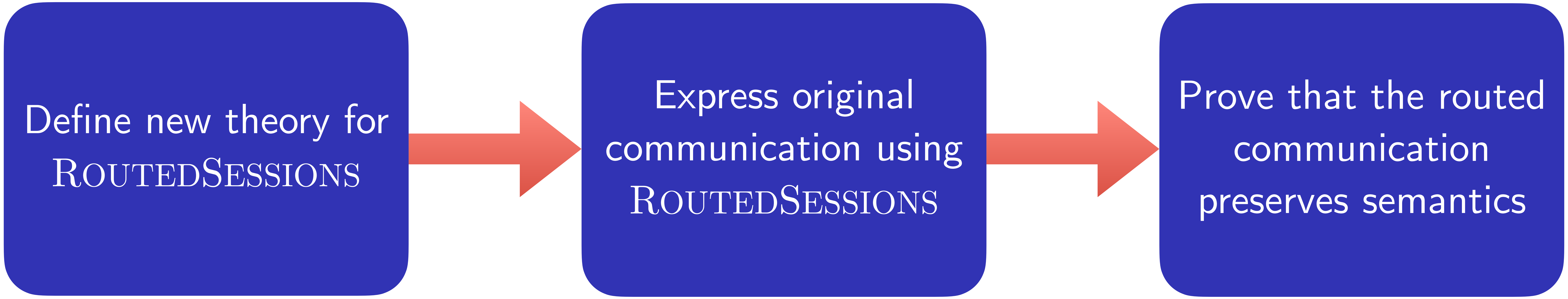
Formalising Routed Communication

"Original communication"

- A sends M1 to B.
- S sends M2 to B.

RoutedSessions

- A sends M1 to B via S.
- S sends M2 to B.



Define new theory for RoutedSessions

Express original communication using RoutedSessions

Prove that the routed communication preserves semantics

Global Types

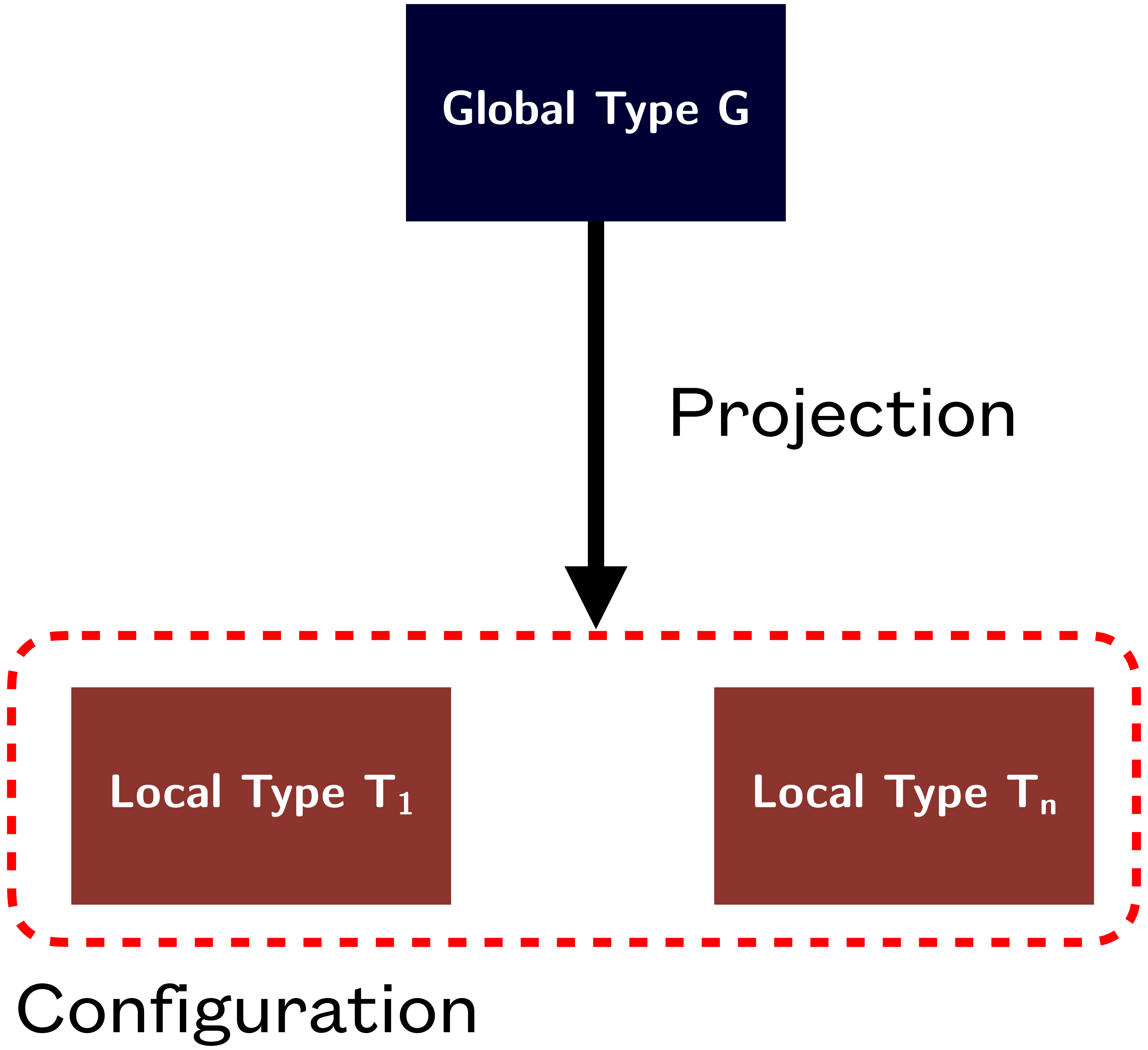
Syntax

$$
\begin{array}{lll}
G ::= & & \text{Global Types} \\
\quad | & \mathtt{end} & \text{Termination} \\
\quad | & \mathtt{t} & \text{Type Variable} \\
\quad | & \mu\mathtt{t}.G & \text{Recursive Type} \\
\quad | & \mathsf{p} \rightarrow \mathsf{q} : \{l_i : G_i\}_{i \in I} & \text{Direct Communication} \\
\quad | & \mathsf{p} \underset{\mathsf{s}}{\rightarrow} \mathsf{q} : \{l_i : G_i\}_{i \in I} & \text{Routed Communication}
\end{array}
$$

"via S"

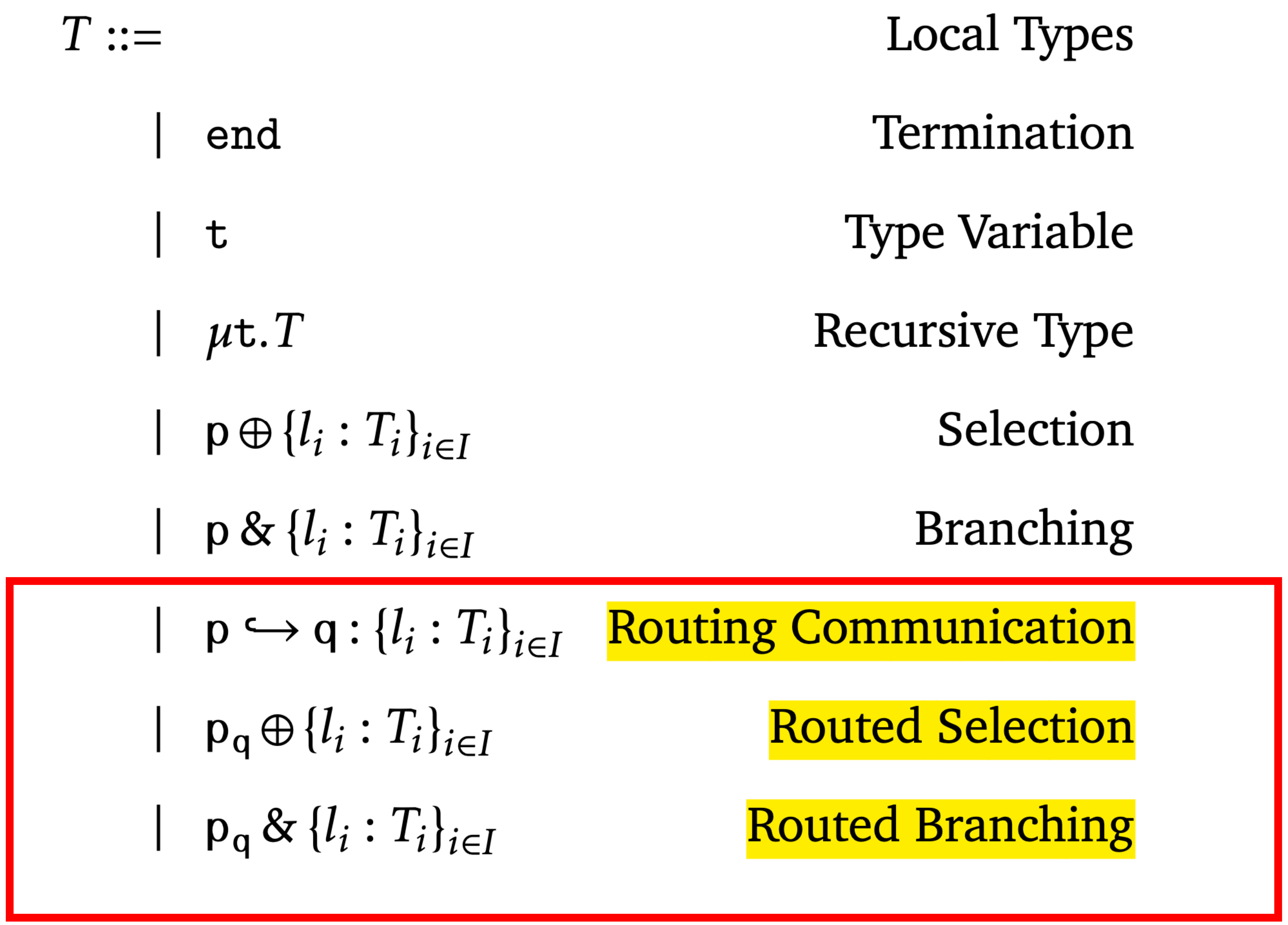
Figure 8.1: Global Types in ROUTEDSESSIONS

Projection



DENI´ELOU, P.-M., AND YOSHIDA, N. Multiparty Compatibility in Communicating Automata: Characterisation and Synthesis of Global Session Types. ICALP 2013.

Local Types



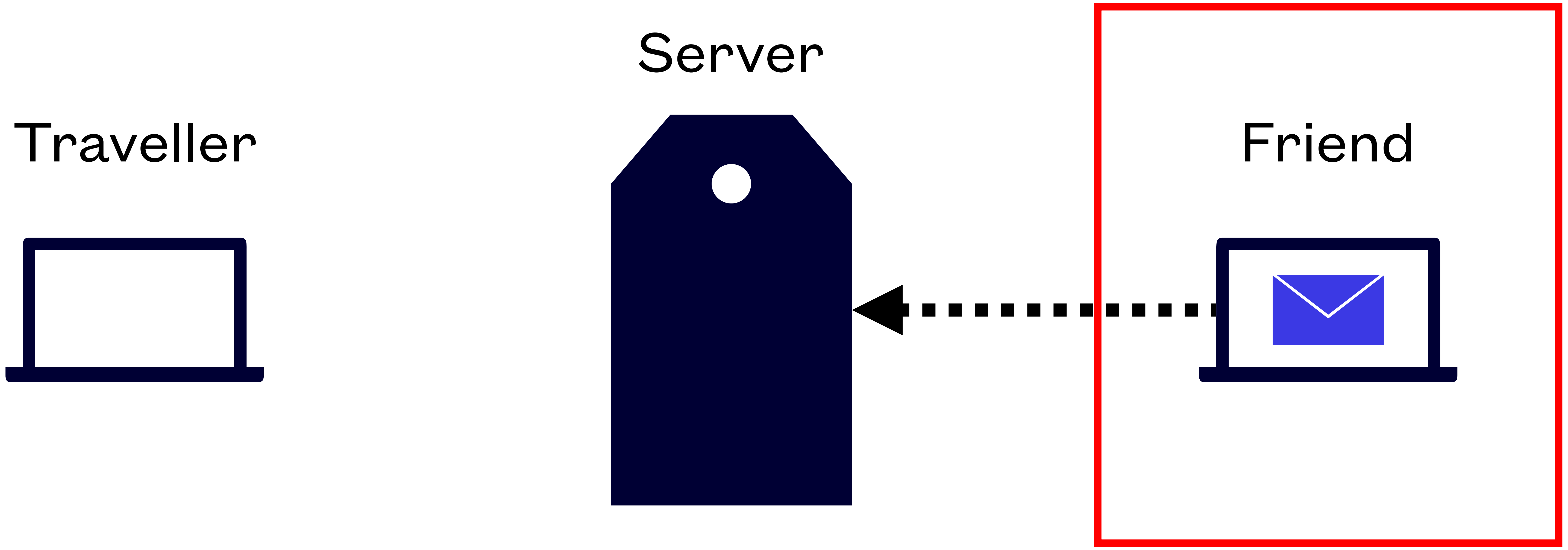
$T ::=$		Local Types
$\mid$	$\texttt{end}$	Termination
$\mid$	$\texttt{t}$	Type Variable
$\mid$	$\mu\texttt{t}.T$	Recursive Type
$\mid$	$\mathsf{p} \oplus \{l_i : T_i\}_{i \in I}$	Selection
$\mid$	$\mathsf{p} \,\&\, \{l_i : T_i\}_{i \in I}$	Branching
$\mid$	$\mathsf{p} \hookrightarrow \mathsf{q} : \{l_i : T_i\}_{i \in I}$	Routing Communication
$\mid$	$\mathsf{p}_{\mathsf{q}} \oplus \{l_i : T_i\}_{i \in I}$	Routed Selection
$\mid$	$\mathsf{p}_{\mathsf{q}} \,\&\, \{l_i : T_i\}_{i \in I}$	Routed Branching

Figure 8.2: Local Types in ROUTEDSESSIONS

Local Types

T ::=		Local Types
|	$\mathtt{end}$	Termination
|	$\mathtt{t}$	Type Variable
|	$\mu\mathtt{t}.T$	Recursive Type
|	$\mathtt{p} \oplus \{l_i : T_i\}_{i \in I}$	Selection
|	$\mathtt{p} \,\&\, \{l_i : T_i\}_{i \in I}$	Branching
|	$\mathtt{p} \hookrightarrow \mathtt{q} : \{l_i : T_i\}_{i \in I}$	Routing Communication
|	$\mathtt{p}_{\mathtt{q}} \oplus \{l_i : T_i\}_{i \in I}$	Routed Selection
|	$\mathtt{p}_{\mathtt{q}} \,\&\, \{l_i : T_i\}_{i \in I}$	Routed Branching

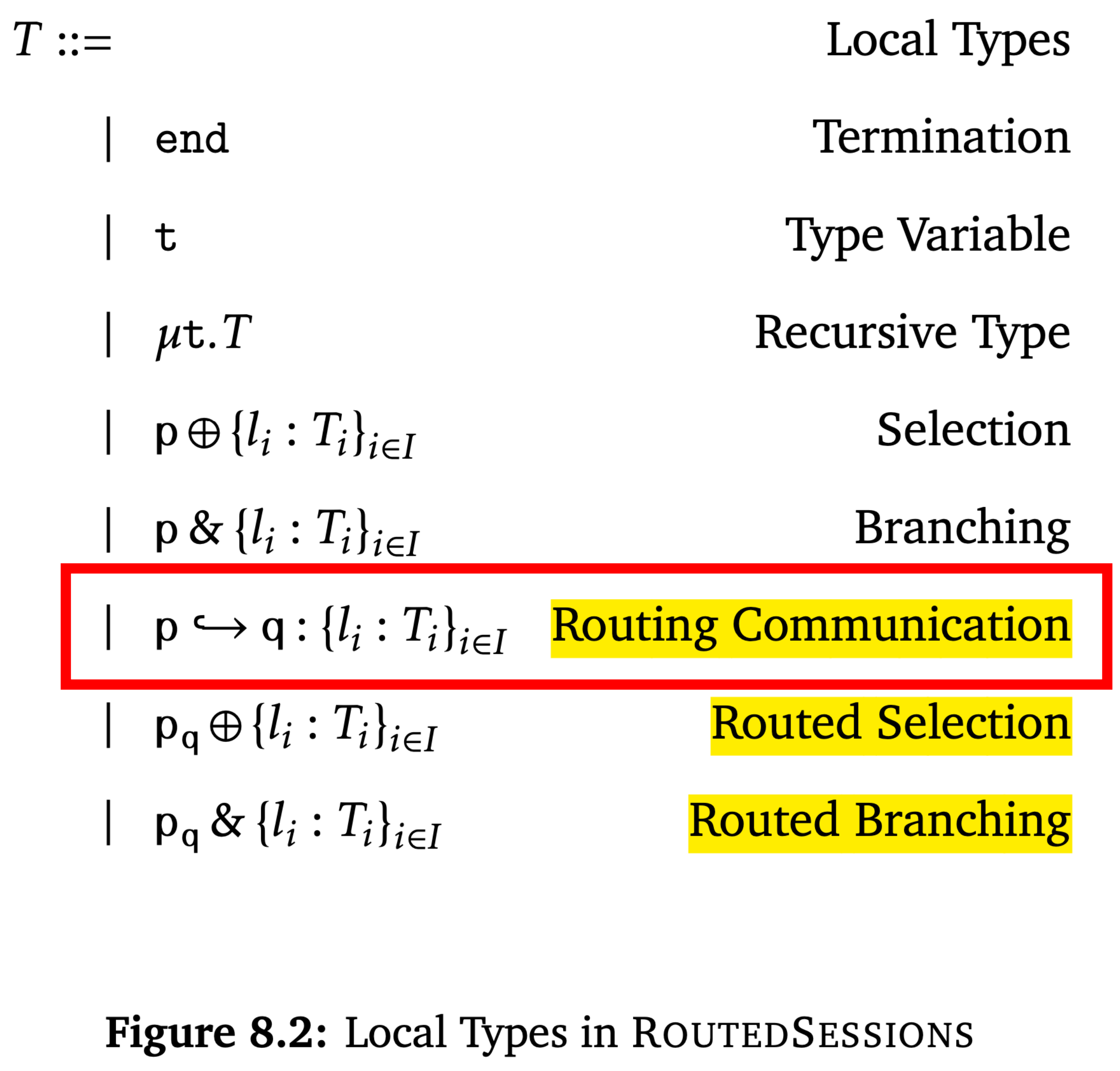
Figure 8.2: Local Types in ROUTEDSESSIONS



“I am sending a message to Server, intended for Traveller”

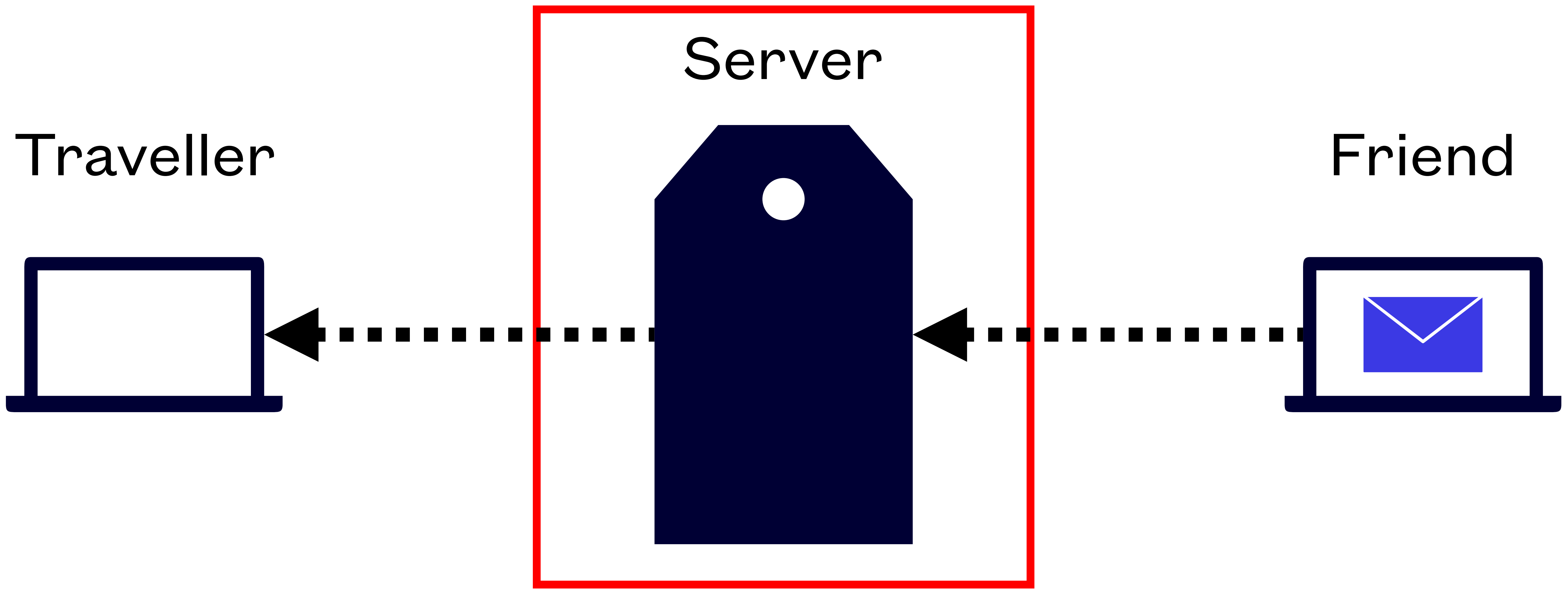
$\text{Traveller}_{\text{Server}} \oplus \text{M1}$

Local Types



$T ::=$	Local Types
$\mid$ `end`	Termination
$\mid$ `t`	Type Variable
$\mid \mu\texttt{t}.T$	Recursive Type
$\mid \mathsf{p} \oplus \{l_i : T_i\}_{i \in I}$	Selection
$\mid \mathsf{p} \,\&\, \{l_i : T_i\}_{i \in I}$	Branching
$\mid \mathsf{p} \hookrightarrow \mathsf{q} : \{l_i : T_i\}_{i \in I}$	Routing Communication
$\mid \mathsf{p}_{\mathsf{q}} \oplus \{l_i : T_i\}_{i \in I}$	Routed Selection
$\mid \mathsf{p}_{\mathsf{q}} \,\&\, \{l_i : T_i\}_{i \in I}$	Routed Branching

Figure 8.2: Local Types in ROUTEDSESSIONS



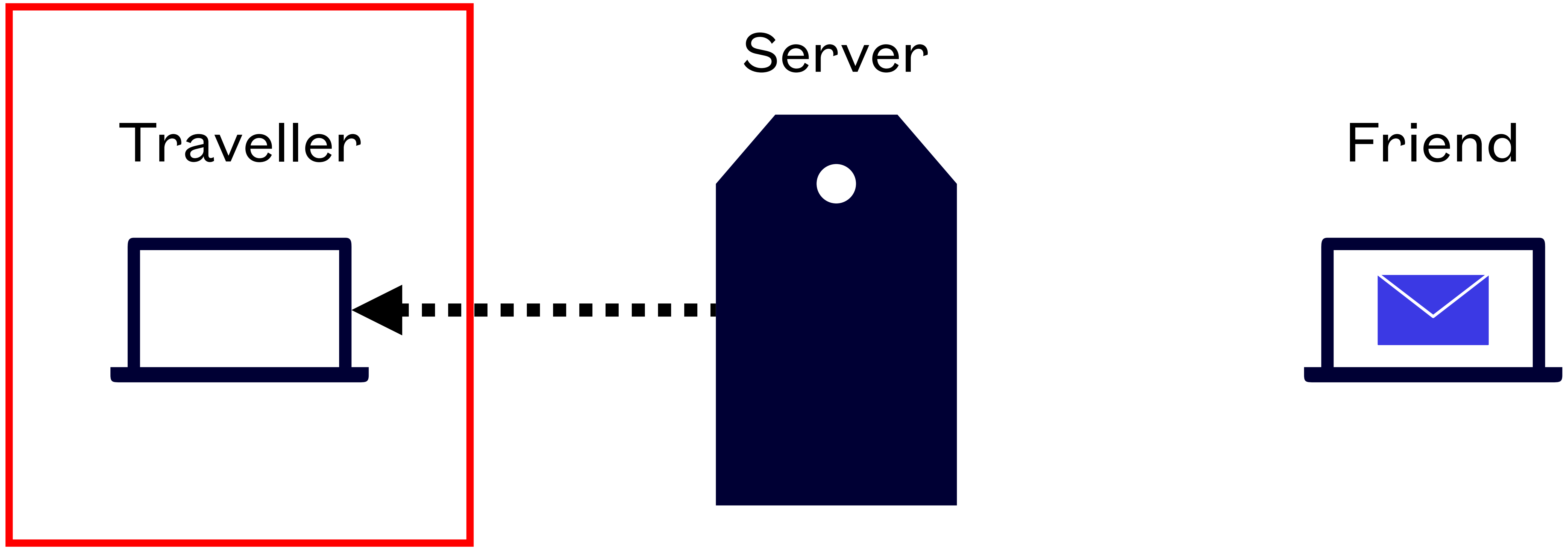
“I am routing a message from Friend intended for Traveller”

Friend ↪ Traveller : M1

Local Types

$T ::=$	Local Types
$\mid$ end	Termination
$\mid$ t	Type Variable
$\mid$ μt.T	Recursive Type
$\mid$ $\mathsf{p} \oplus \{l_i : T_i\}_{i \in I}$	Selection
$\mid$ $\mathsf{p} \,\&\, \{l_i : T_i\}_{i \in I}$	Branching
$\mid$ $\mathsf{p} \hookrightarrow \mathsf{q} : \{l_i : T_i\}_{i \in I}$	Routing Communication
$\mid$ $\mathsf{p}_\mathsf{q} \oplus \{l_i : T_i\}_{i \in I}$	Routed Selection
$\mid$ $\mathsf{p}_\mathsf{q} \,\&\, \{l_i : T_i\}_{i \in I}$	Routed Branching

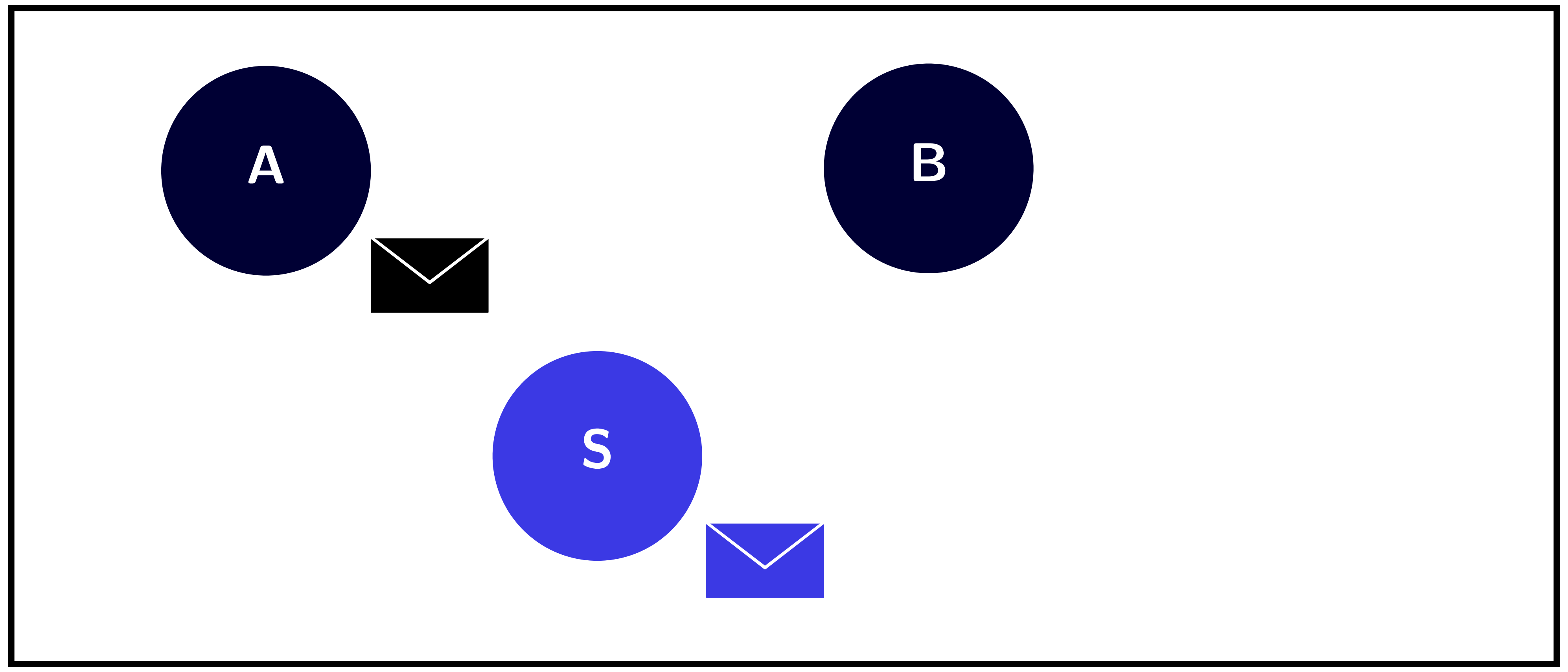
Figure 8.2: Local Types in RoutedSessions

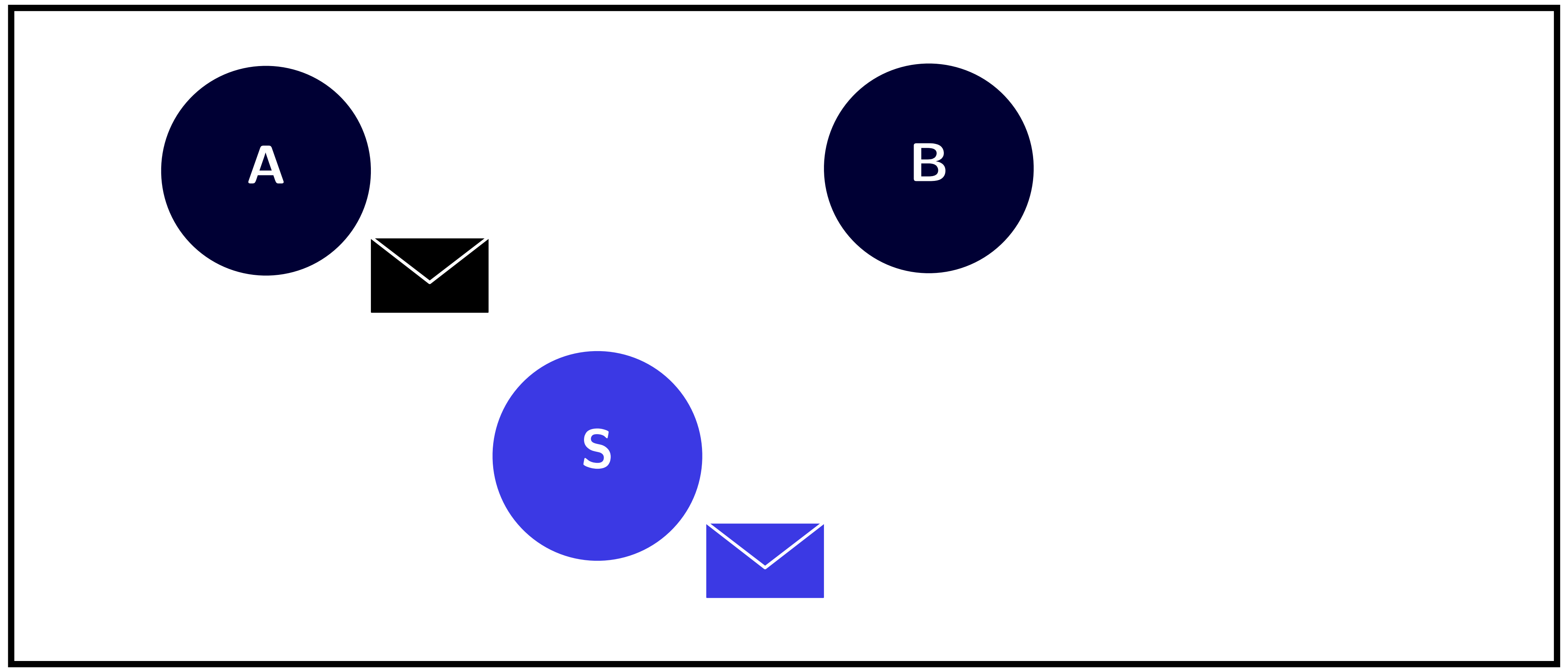


"I am receiving a message from Server, originally from Friend"

$\text{Friend}_{\text{Server}}$ & M1

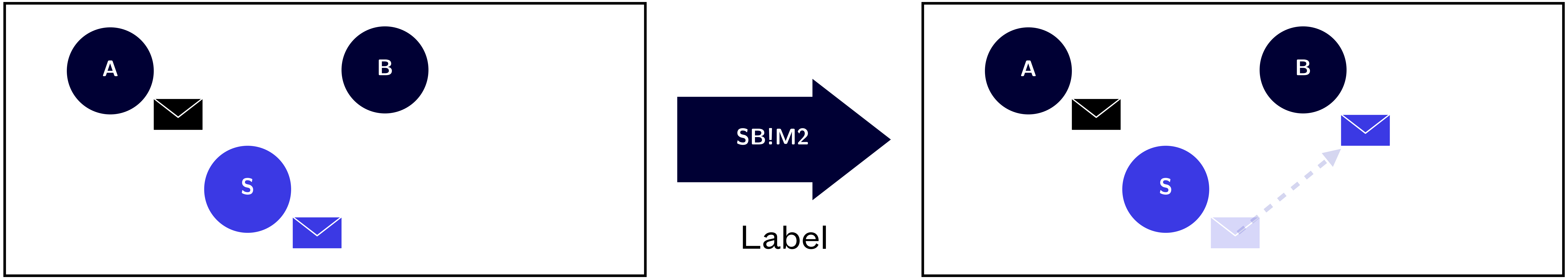
Semantics





DENI´ELOU, P.-M., AND YOSHIDA, N. Multiparty Compatibility in Communicating Automata: Characterisation and Synthesis of Global Session Types. ICALP 2013.

Semantics



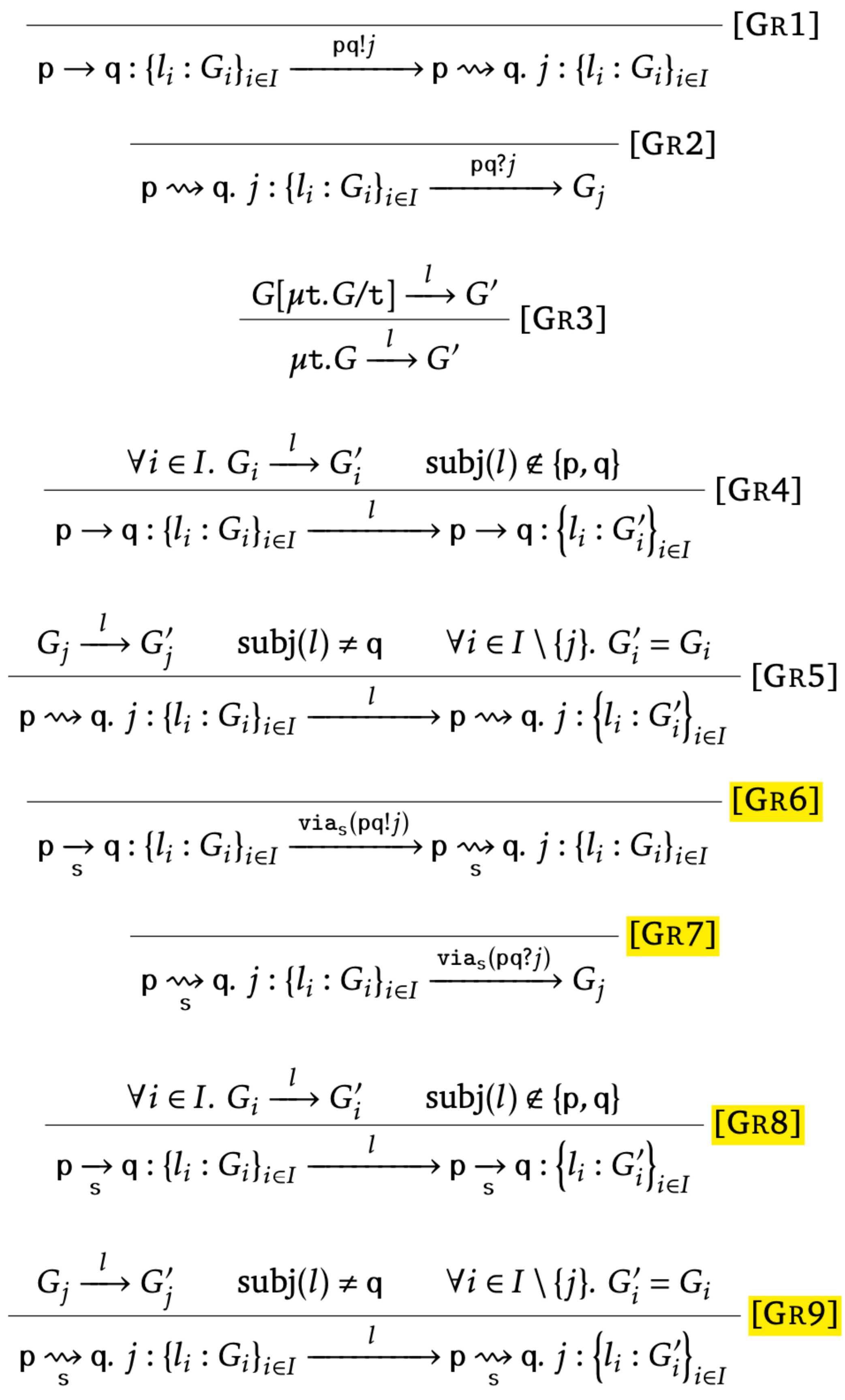
$l ::=$		Labels
$\mid$	$\texttt{pq}!j$	Direct Send
$\mid$	$\texttt{pq}?j$	Direct Receive
$\mid$	$\texttt{via}_{\texttt{s}}(\texttt{pq}!j)$	Routed Send
$\mid$	$\texttt{via}_{\texttt{s}}(\texttt{pq}?j)$	Routed Receive

Figure 8.3: LTS Labels in RoutedSessions

Labelled Transition System

DENI´ELOU, P.-M., AND YOSHIDA, N. Multiparty Compatibility in Communicating Automata: Characterisation and Synthesis of Global Session Types. ICALP 2013.

Semantics



$$\dfrac{}{\mathsf{p} \rightarrow \mathsf{q} : \{l_i : G_i\}_{i \in I} \xrightarrow{\mathsf{pq}!j} \mathsf{p} \rightsquigarrow \mathsf{q}.\ j : \{l_i : G_i\}_{i \in I}}\ [\textsc{Gr1}]$$

$$\dfrac{}{\mathsf{p} \rightsquigarrow \mathsf{q}.\ j : \{l_i : G_i\}_{i \in I} \xrightarrow{\mathsf{pq}?j} G_j}\ [\textsc{Gr2}]$$

$$\dfrac{G[\mu \mathtt{t}.G/\mathtt{t}] \xrightarrow{l} G'}{\mu \mathtt{t}.G \xrightarrow{l} G'}\ [\textsc{Gr3}]$$

$$\dfrac{\forall i \in I.\ G_i \xrightarrow{l} G'_i \qquad \mathsf{subj}(l) \notin \{\mathsf{p}, \mathsf{q}\}}{\mathsf{p} \rightarrow \mathsf{q} : \{l_i : G_i\}_{i \in I} \xrightarrow{l} \mathsf{p} \rightarrow \mathsf{q} : \left\{l_i : G'_i\right\}_{i \in I}}\ [\textsc{Gr4}]$$

$$\dfrac{G_j \xrightarrow{l} G'_j \qquad \mathsf{subj}(l) \neq \mathsf{q} \qquad \forall i \in I \setminus \{j\}.\ G'_i = G_i}{\mathsf{p} \rightsquigarrow \mathsf{q}.\ j : \{l_i : G_i\}_{i \in I} \xrightarrow{l} \mathsf{p} \rightsquigarrow \mathsf{q}.\ j : \left\{l_i : G'_i\right\}_{i \in I}}\ [\textsc{Gr5}]$$

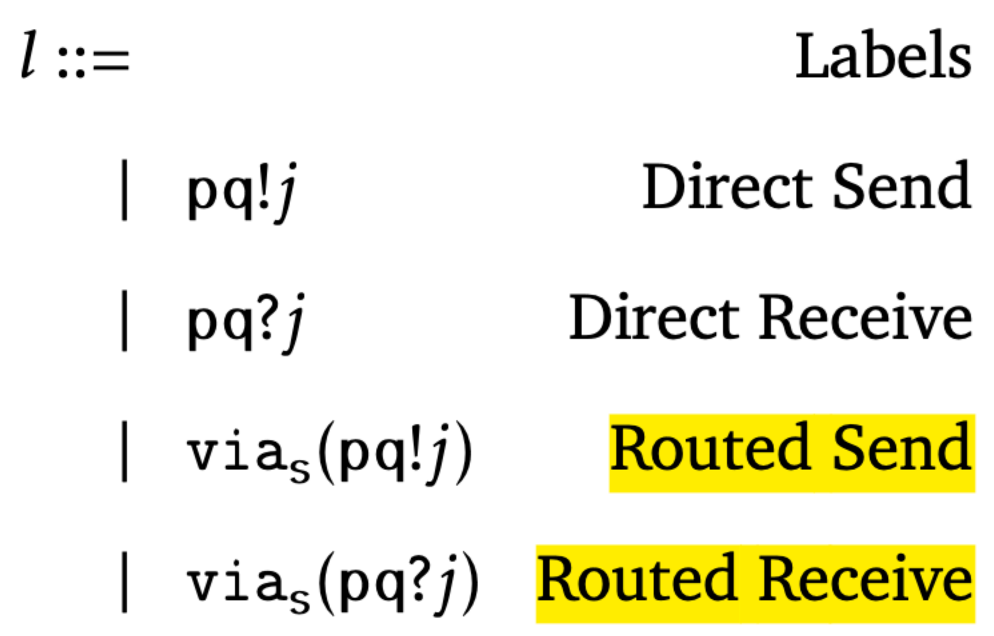
$$\dfrac{}{\mathsf{p} \underset{\mathsf{s}}{\rightarrow} \mathsf{q} : \{l_i : G_i\}_{i \in I} \xrightarrow{\mathtt{via}_{\mathsf{s}}(\mathsf{pq}!j)} \mathsf{p} \underset{\mathsf{s}}{\rightsquigarrow} \mathsf{q}.\ j : \{l_i : G_i\}_{i \in I}}\ [\textsc{Gr6}]$$

$$\dfrac{}{\mathsf{p} \underset{\mathsf{s}}{\rightsquigarrow} \mathsf{q}.\ j : \{l_i : G_i\}_{i \in I} \xrightarrow{\mathtt{via}_{\mathsf{s}}(\mathsf{pq}?j)} G_j}\ [\textsc{Gr7}]$$

$$\dfrac{\forall i \in I.\ G_i \xrightarrow{l} G'_i \qquad \mathsf{subj}(l) \notin \{\mathsf{p}, \mathsf{q}\}}{\mathsf{p} \underset{\mathsf{s}}{\rightarrow} \mathsf{q} : \{l_i : G_i\}_{i \in I} \xrightarrow{l} \mathsf{p} \underset{\mathsf{s}}{\rightarrow} \mathsf{q} : \left\{l_i : G'_i\right\}_{i \in I}}\ [\textsc{Gr8}]$$

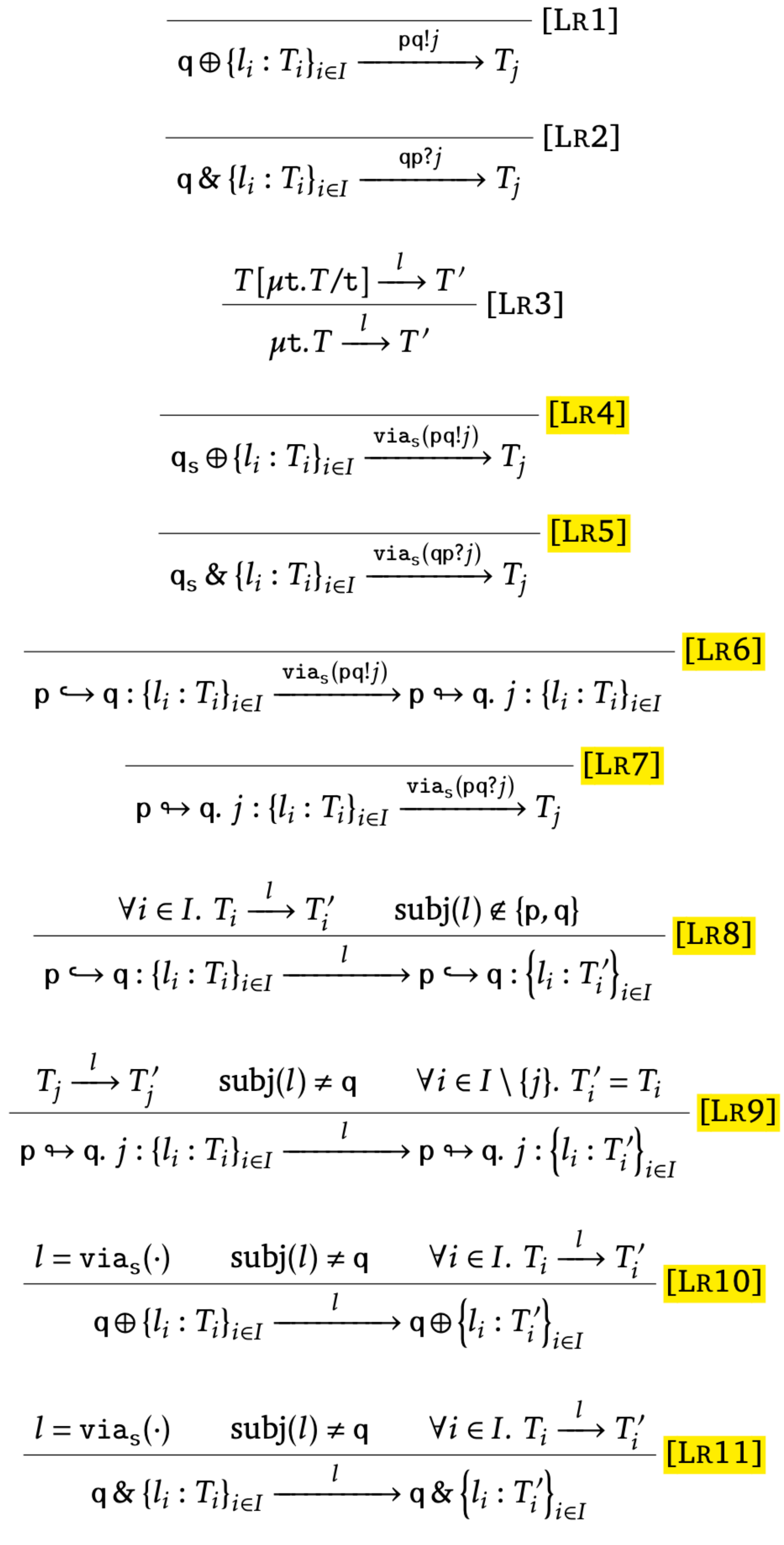
$$\dfrac{G_j \xrightarrow{l} G'_j \qquad \mathsf{subj}(l) \neq \mathsf{q} \qquad \forall i \in I \setminus \{j\}.\ G'_i = G_i}{\mathsf{p} \underset{\mathsf{s}}{\rightsquigarrow} \mathsf{q}.\ j : \{l_i : G_i\}_{i \in I} \xrightarrow{l} \mathsf{p} \underset{\mathsf{s}}{\rightsquigarrow} \mathsf{q}.\ j : \left\{l_i : G'_i\right\}_{i \in I}}\ [\textsc{Gr9}]$$

Figure 8.4: LTS Semantics over Global Types in ROUTEDSESSIONS



$l ::=$		Labels
$\mid$	$\mathsf{pq}!j$	Direct Send
$\mid$	$\mathsf{pq}?j$	Direct Receive
$\mid$	$\mathtt{via}_{\mathsf{s}}(\mathsf{pq}!j)$	Routed Send
$\mid$	$\mathtt{via}_{\mathsf{s}}(\mathsf{pq}?j)$	Routed Receive

Figure 8.3: LTS Labels in ROUTEDSESSIONS



$$\dfrac{}{\mathsf{q} \oplus \{l_i : T_i\}_{i \in I} \xrightarrow{\mathsf{pq}!j} T_j}\ [\textsc{Lr1}]$$

$$\dfrac{}{\mathsf{q}\ \&\ \{l_i : T_i\}_{i \in I} \xrightarrow{\mathsf{qp}?j} T_j}\ [\textsc{Lr2}]$$

$$\dfrac{T[\mu \mathtt{t}.T/\mathtt{t}] \xrightarrow{l} T'}{\mu \mathtt{t}.T \xrightarrow{l} T'}\ [\textsc{Lr3}]$$

$$\dfrac{}{\mathsf{q}_{\mathsf{s}} \oplus \{l_i : T_i\}_{i \in I} \xrightarrow{\mathtt{via}_{\mathsf{s}}(\mathsf{pq}!j)} T_j}\ [\textsc{Lr4}]$$

$$\dfrac{}{\mathsf{q}_{\mathsf{s}}\ \&\ \{l_i : T_i\}_{i \in I} \xrightarrow{\mathtt{via}_{\mathsf{s}}(\mathsf{qp}?j)} T_j}\ [\textsc{Lr5}]$$

$$\dfrac{}{\mathsf{p} \hookrightarrow \mathsf{q} : \{l_i : T_i\}_{i \in I} \xrightarrow{\mathtt{via}_{\mathsf{s}}(\mathsf{pq}!j)} \mathsf{p} \looparrowright \mathsf{q}.\ j : \{l_i : T_i\}_{i \in I}}\ [\textsc{Lr6}]$$

$$\dfrac{}{\mathsf{p} \looparrowright \mathsf{q}.\ j : \{l_i : T_i\}_{i \in I} \xrightarrow{\mathtt{via}_{\mathsf{s}}(\mathsf{pq}?j)} T_j}\ [\textsc{Lr7}]$$

$$\dfrac{\forall i \in I.\ T_i \xrightarrow{l} T'_i \qquad \mathsf{subj}(l) \notin \{\mathsf{p}, \mathsf{q}\}}{\mathsf{p} \hookrightarrow \mathsf{q} : \{l_i : T_i\}_{i \in I} \xrightarrow{l} \mathsf{p} \hookrightarrow \mathsf{q} : \left\{l_i : T'_i\right\}_{i \in I}}\ [\textsc{Lr8}]$$

$$\dfrac{T_j \xrightarrow{l} T'_j \qquad \mathsf{subj}(l) \neq \mathsf{q} \qquad \forall i \in I \setminus \{j\}.\ T'_i = T_i}{\mathsf{p} \looparrowright \mathsf{q}.\ j : \{l_i : T_i\}_{i \in I} \xrightarrow{l} \mathsf{p} \looparrowright \mathsf{q}.\ j : \left\{l_i : T'_i\right\}_{i \in I}}\ [\textsc{Lr9}]$$

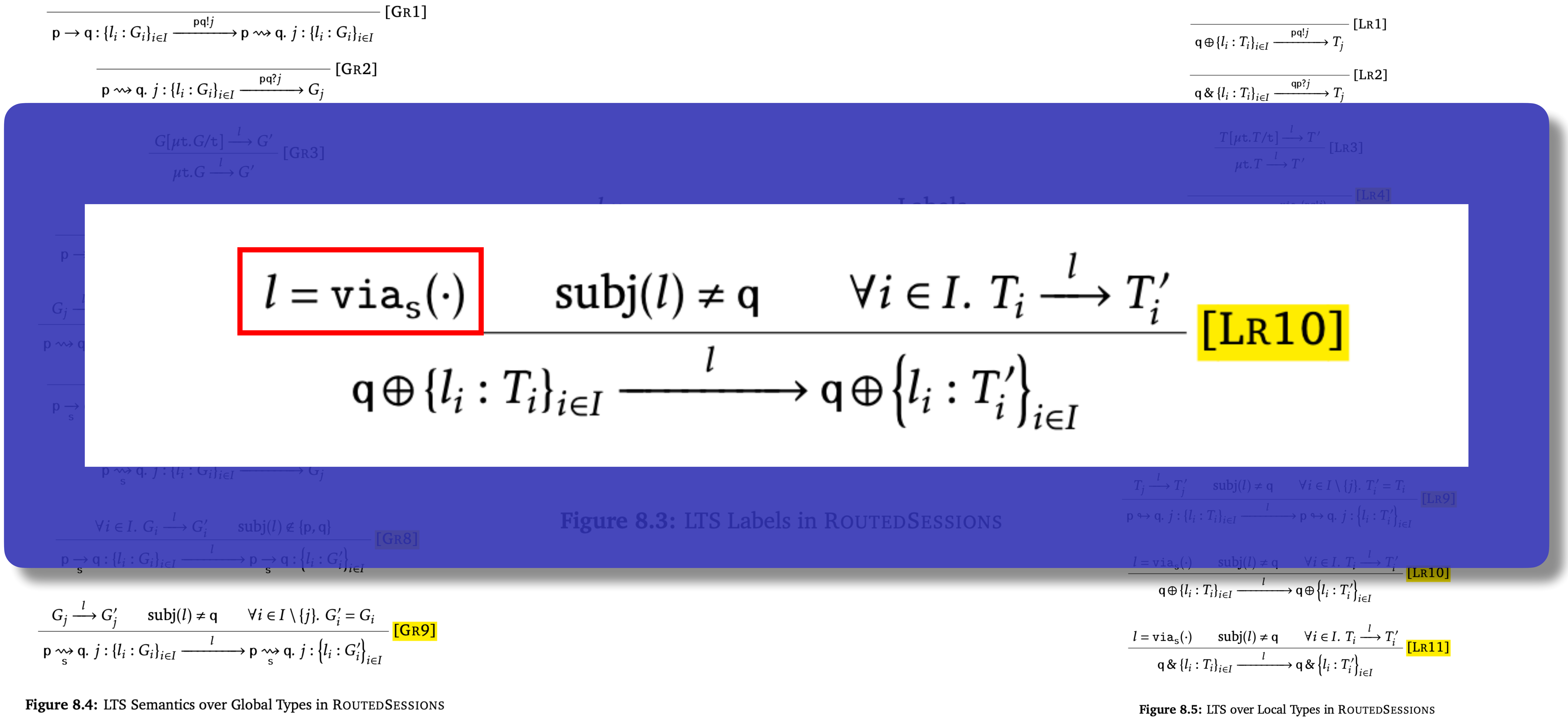
$$\dfrac{l = \mathtt{via}_{\mathsf{s}}(\cdot) \qquad \mathsf{subj}(l) \neq \mathsf{q} \qquad \forall i \in I.\ T_i \xrightarrow{l} T'_i}{\mathsf{q} \oplus \{l_i : T_i\}_{i \in I} \xrightarrow{l} \mathsf{q} \oplus \left\{l_i : T'_i\right\}_{i \in I}}\ [\textsc{Lr10}]$$

$$\dfrac{l = \mathtt{via}_{\mathsf{s}}(\cdot) \qquad \mathsf{subj}(l) \neq \mathsf{q} \qquad \forall i \in I.\ T_i \xrightarrow{l} T'_i}{\mathsf{q}\ \&\ \{l_i : T_i\}_{i \in I} \xrightarrow{l} \mathsf{q}\ \&\ \left\{l_i : T'_i\right\}_{i \in I}}\ [\textsc{Lr11}]$$

Figure 8.5: LTS over Local Types in ROUTEDSESSIONS

DENI´ELOU, P.-M., AND YOSHIDA, N. Multiparty Compatibility in Communicating Automata: Characterisation and Synthesis of Global Session Types. ICALP 2013.

Semantics



$$\frac{}{\mathsf{p} \to \mathsf{q} : \{l_i : G_i\}_{i \in I} \xrightarrow{\mathsf{pq}!j} \mathsf{p} \rightsquigarrow \mathsf{q}.\ j : \{l_i : G_i\}_{i \in I}}\ [\text{Gr1}]$$

$$\frac{}{\mathsf{p} \rightsquigarrow \mathsf{q}.\ j : \{l_i : G_i\}_{i \in I} \xrightarrow{\mathsf{pq}?j} G_j}\ [\text{Gr2}]$$

$$\frac{G[\mu \mathtt{t}.G/\mathtt{t}] \xrightarrow{l} G'}{\mu \mathtt{t}.G \xrightarrow{l} G'}\ [\text{Gr3}]$$

$$\frac{\forall i \in I.\ G_i \xrightarrow{l} G_i' \qquad \mathsf{subj}(l) \notin \{\mathsf{p}, \mathsf{q}\}}{\mathsf{p} \underset{\mathsf{s}}{\to} \mathsf{q} : \{l_i : G_i\}_{i \in I} \xrightarrow{l} \mathsf{p} \underset{\mathsf{s}}{\to} \mathsf{q} : \left\{l_i : G_i'\right\}_{i \in I}}\ [\text{Gr8}]$$

$$\frac{G_j \xrightarrow{l} G_j' \qquad \mathsf{subj}(l) \neq \mathsf{q} \qquad \forall i \in I \setminus \{j\}.\ G_i' = G_i}{\mathsf{p} \underset{\mathsf{s}}{\rightsquigarrow} \mathsf{q}.\ j : \{l_i : G_i\}_{i \in I} \xrightarrow{l} \mathsf{p} \underset{\mathsf{s}}{\rightsquigarrow} \mathsf{q}.\ j : \left\{l_i : G_i'\right\}_{i \in I}}\ [\text{Gr9}]$$

Figure 8.4: LTS Semantics over Global Types in RoutedSessions

$$\frac{\boxed{l = \mathtt{via}_{\mathsf{s}}(\cdot)} \qquad \mathsf{subj}(l) \neq \mathsf{q} \qquad \forall i \in I.\ T_i \xrightarrow{l} T_i'}{\mathsf{q} \oplus \{l_i : T_i\}_{i \in I} \xrightarrow{l} \mathsf{q} \oplus \left\{l_i : T_i'\right\}_{i \in I}}\ [\text{Lr10}]$$

Figure 8.3: LTS Labels in RoutedSessions

$$\frac{}{\mathsf{q} \oplus \{l_i : T_i\}_{i \in I} \xrightarrow{\mathsf{pq}!j} T_j}\ [\text{Lr1}]$$

$$\frac{}{\mathsf{q}\ \&\ \{l_i : T_i\}_{i \in I} \xrightarrow{\mathsf{qp}?j} T_j}\ [\text{Lr2}]$$

$$\frac{T[\mu \mathtt{t}.T/\mathtt{t}] \xrightarrow{l} T'}{\mu \mathtt{t}.T \xrightarrow{l} T'}\ [\text{Lr3}]$$

$$\frac{T_j \xrightarrow{l} T_j' \qquad \mathsf{subj}(l) \neq \mathsf{q} \qquad \forall i \in I \setminus \{j\}.\ T_i' = T_i}{\mathsf{p} \hookrightarrow \mathsf{q}.\ j : \{l_i : T_i\}_{i \in I} \xrightarrow{l} \mathsf{p} \hookrightarrow \mathsf{q}.\ j : \left\{l_i : T_i'\right\}_{i \in I}}\ [\text{Lr9}]$$

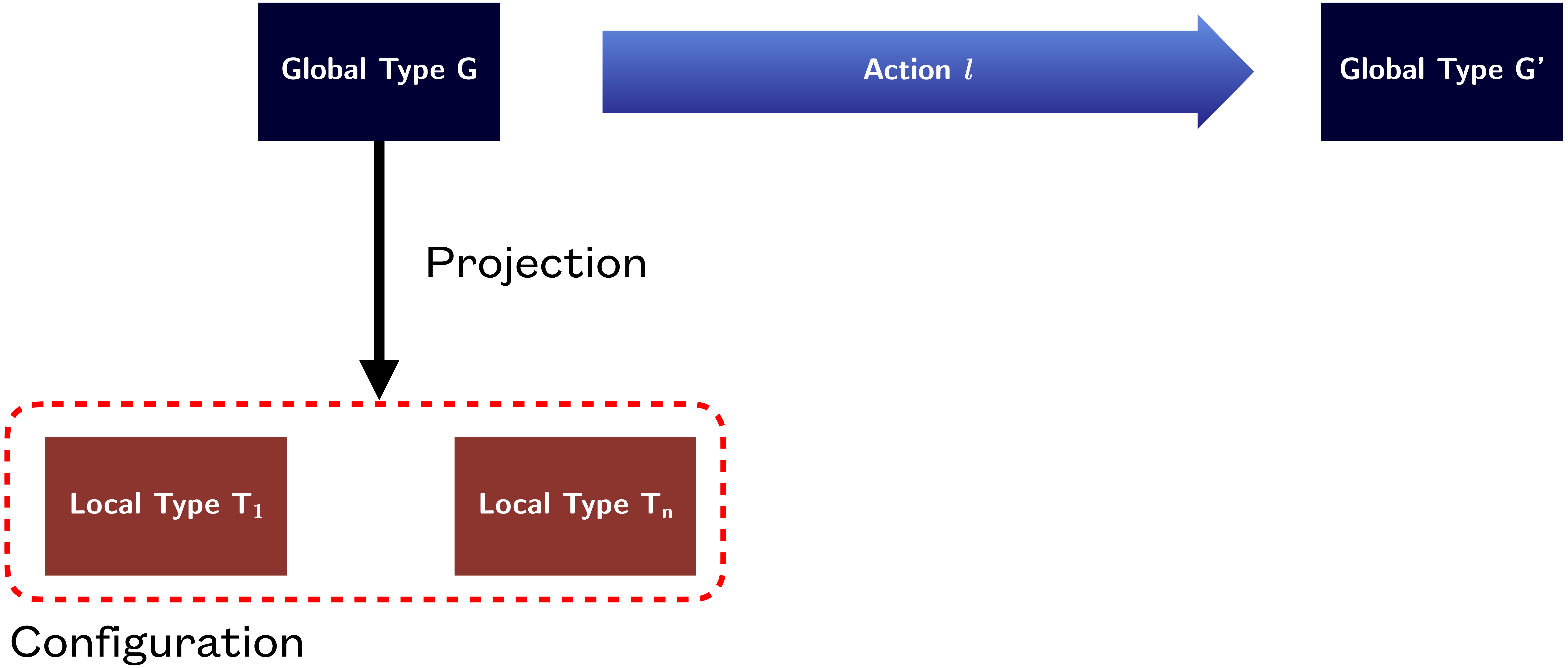
$$\frac{l = \mathtt{via}_{\mathsf{s}}(\cdot) \qquad \mathsf{subj}(l) \neq \mathsf{q} \qquad \forall i \in I.\ T_i \xrightarrow{l} T_i'}{\mathsf{q} \oplus \{l_i : T_i\}_{i \in I} \xrightarrow{l} \mathsf{q} \oplus \left\{l_i : T_i'\right\}_{i \in I}}\ [\text{Lr10}]$$

$$\frac{l = \mathtt{via}_{\mathsf{s}}(\cdot) \qquad \mathsf{subj}(l) \neq \mathsf{q} \qquad \forall i \in I.\ T_i \xrightarrow{l} T_i'}{\mathsf{q}\ \&\ \{l_i : T_i\}_{i \in I} \xrightarrow{l} \mathsf{q}\ \&\ \left\{l_i : T_i'\right\}_{i \in I}}\ [\text{Lr11}]$$

Figure 8.5: LTS over Local Types in RoutedSessions

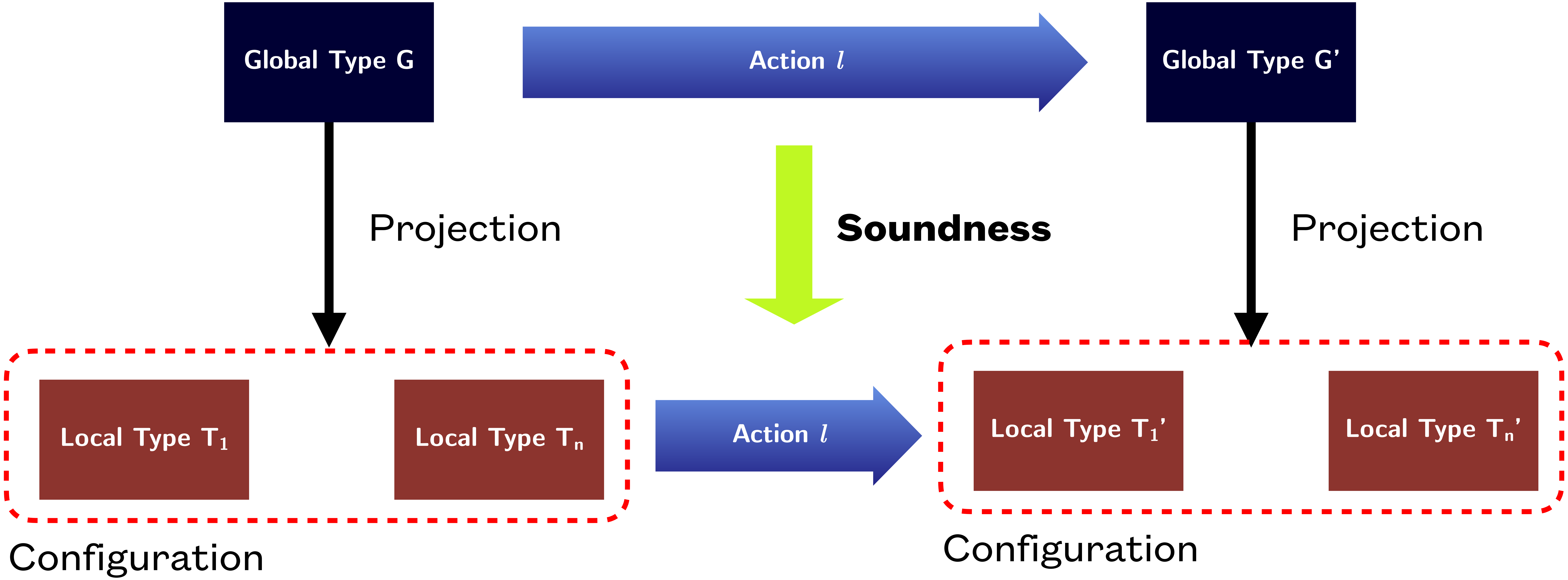
DENI´ELOU, P.-M., AND YOSHIDA, N. Multiparty Compatibility in Communicating Automata: Characterisation and Synthesis of Global Session Types. ICALP 2013.

Semantics



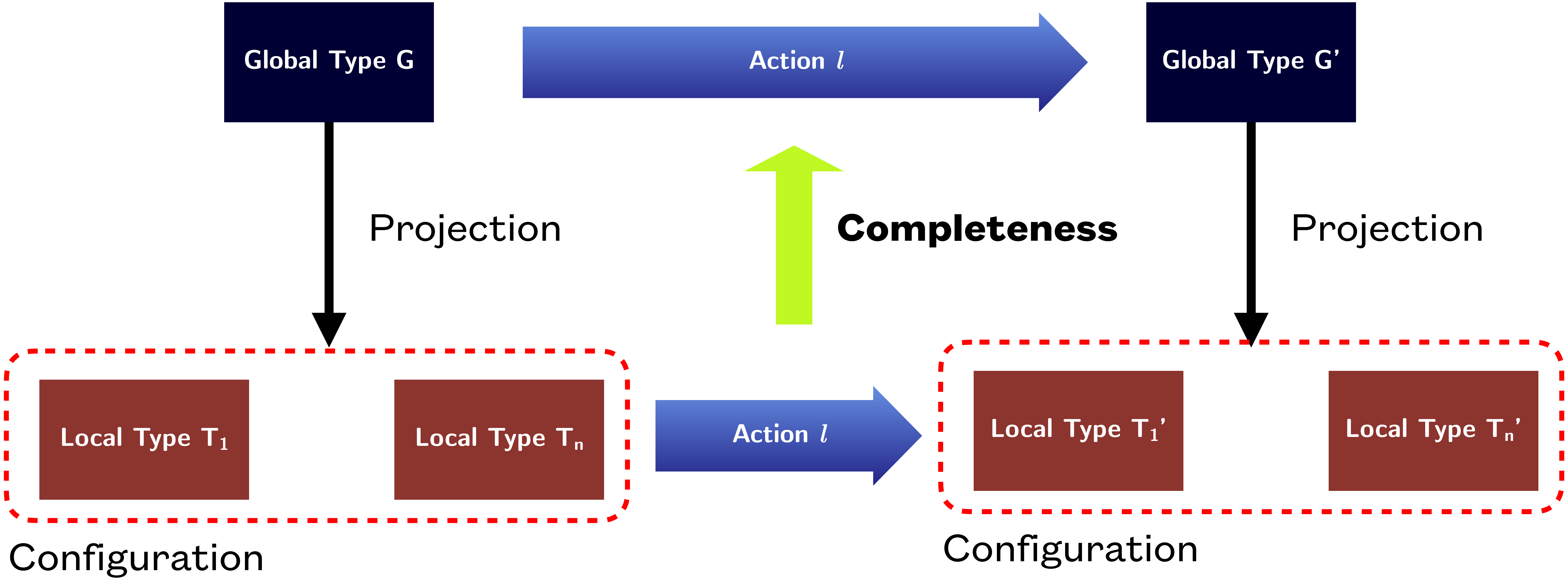
DENI´ELOU, P.-M., AND YOSHIDA, N. Multiparty Compatibility in Communicating Automata: Characterisation and Synthesis of Global Session Types. ICALP 2013.

Soundness



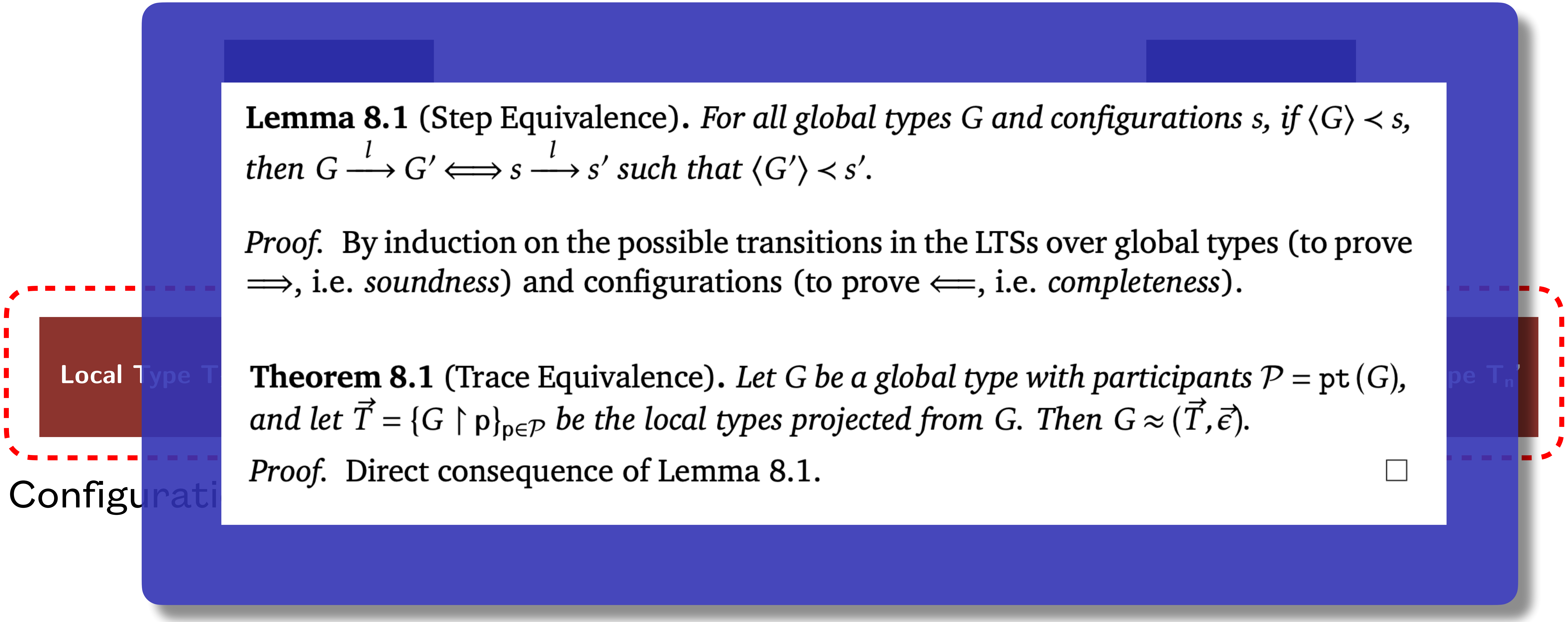
DENI´ELOU, P.-M., AND YOSHIDA, N. Multiparty Compatibility in Communicating Automata: Characterisation and Synthesis of Global Session Types. ICALP 2013.

Completeness



DENI´ELOU, P.-M., AND YOSHIDA, N. Multiparty Compatibility in Communicating Automata: Characterisation and Synthesis of Global Session Types. ICALP 2013.

Soundness & Completeness



Lemma 8.1 (Step Equivalence). *For all global types G and configurations s, if $\langle G \rangle \prec s$, then $G \xrightarrow{l} G' \Longleftrightarrow s \xrightarrow{l} s'$ such that $\langle G' \rangle \prec s'$.*

Proof. By induction on the possible transitions in the LTSs over global types (to prove $\Longrightarrow$, i.e. *soundness*) and configurations (to prove $\Longleftarrow$, i.e. *completeness*).

Theorem 8.1 (Trace Equivalence). *Let G be a global type with participants $\mathcal{P} = \mathtt{pt}(G)$, and let $\vec{T} = \{G \upharpoonright \mathtt{p}\}_{\mathtt{p} \in \mathcal{P}}$ be the local types projected from G. Then $G \approx (\vec{T}, \vec{\epsilon})$.*

Proof. Direct consequence of Lemma 8.1. □

DENI´ELOU, P.-M., AND YOSHIDA, N. Multiparty Compatibility in Communicating Automata: Characterisation and Synthesis of Global Session Types. ICALP 2013.

Towards RoutedSessions

Definition 8.9 (Encoding on Global Types)**.**

$$\llbracket \mathtt{end},\ \mathtt{s} \rrbracket \;=\; \mathtt{end} \qquad [\textsc{Enc-G-End}]$$

$$\llbracket \mathtt{t},\ \mathtt{s} \rrbracket \;=\; \mathtt{t} \qquad [\textsc{Enc-G-RecVar}]$$

$$\llbracket \mu\mathtt{t}.G,\ \mathtt{s} \rrbracket \;=\; \mu\mathtt{t}.\llbracket G,\ \mathtt{s} \rrbracket \qquad [\textsc{Enc-G-Rec}]$$

$$\llbracket \mathtt{p} \to \mathtt{q} : \{l_i : G_i\}_{i\in I},\ \mathtt{s} \rrbracket \;=\; \begin{cases} \mathtt{p} \to \mathtt{q} : \{l_i : \llbracket G_i,\ \mathtt{s} \rrbracket\}_{i\in I} & \text{if } \mathtt{s} \in \{\mathtt{p},\mathtt{q}\} \\ \mathtt{p} \xrightarrow[\mathtt{s}]{} \mathtt{q} : \{l_i : \llbracket G_i,\ \mathtt{s} \rrbracket\}_{i\in I} & \text{otherwise} \end{cases} \qquad [\textsc{Enc-G-Comm}]$$

```
encode :: RouterRole -> CanonicalTheory -> NewTheory
```

Towards RoutedSessions

Definition 8.9 (Encoding on Global Types).

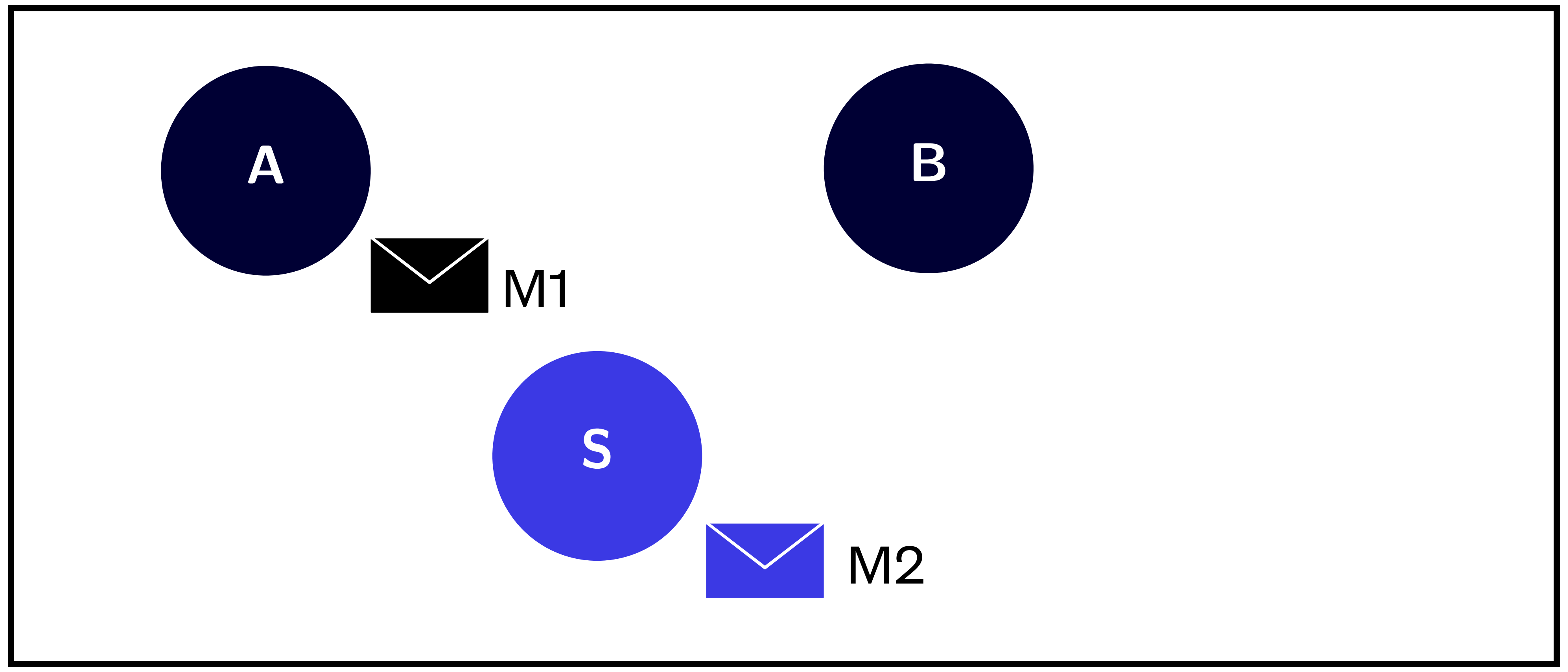
$$\begin{aligned}
[\![\mathsf{end},\ \mathsf{s}]\!] &= \mathsf{end} && [\textsc{Enc-G-End}]\\
[\![\mathsf{t},\ \mathsf{s}]\!] &= \mathsf{t} && [\textsc{Enc-G-RecVar}]\\
[\![\mu\mathsf{t}.G,\ \mathsf{s}]\!] &= \mu\mathsf{t}.[\![G,\ \mathsf{s}]\!] && [\textsc{Enc-G-Rec}]\\
[\![\mathsf{p} \to \mathsf{q} : \{l_i : G_i\}_{i\in I},\ \mathsf{s}]\!] &= \begin{cases} \mathsf{p} \to \mathsf{q} : \{l_i : [\![G_i,\ \mathsf{s}]\!]\}_{i\in I} & \text{if } \mathsf{s} \in \{\mathsf{p},\mathsf{q}\} \\ \mathsf{p} \xrightarrow[\mathsf{s}]{} \mathsf{q} : \{l_i : [\![G_i,\ \mathsf{s}]\!]\}_{i\in I} & \text{otherwise} \end{cases} && [\textsc{Enc-G-Comm}]
\end{aligned}$$

```
encode :: RouterRole -> CanonicalTheory -> NewTheory
```

Encoding Preserves Semantics

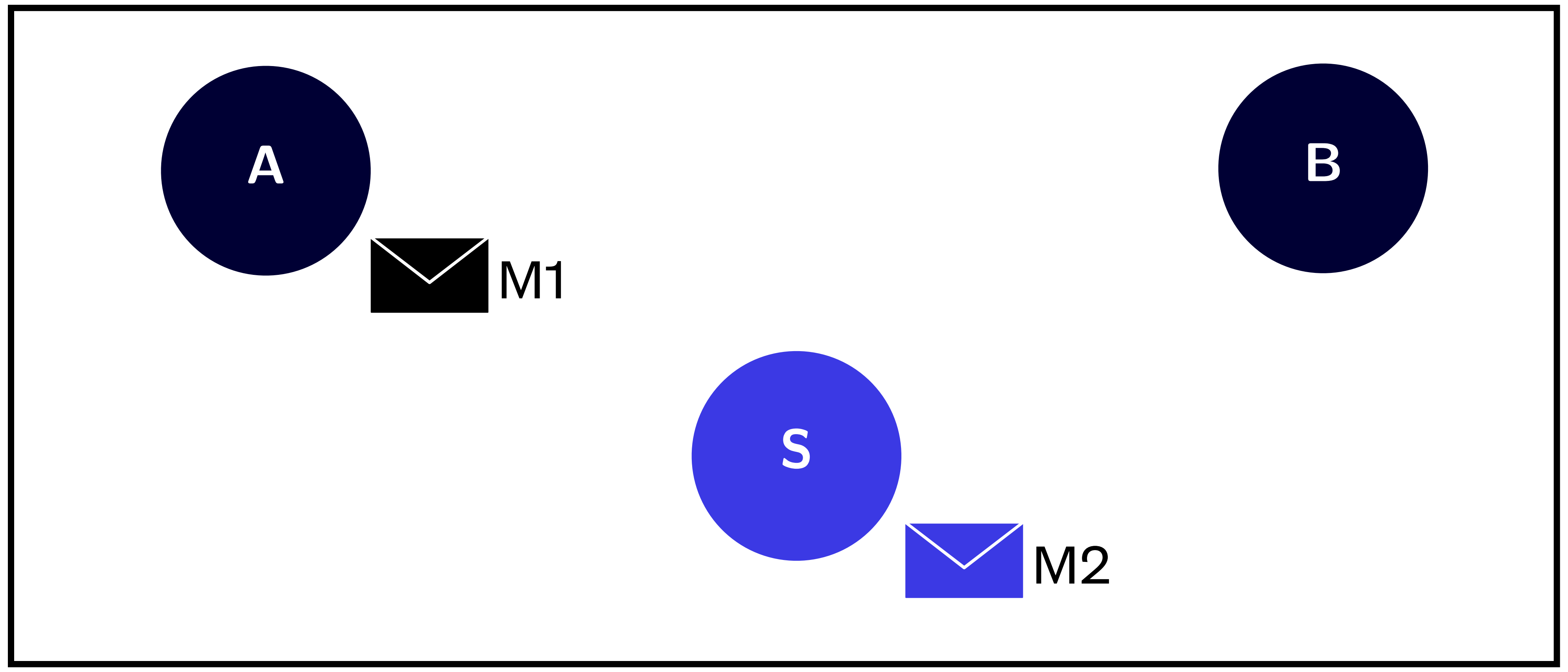
"Original communication"

- A sends M1 to B.
- S sends M2 to B.



ROUTEDSESSIONS

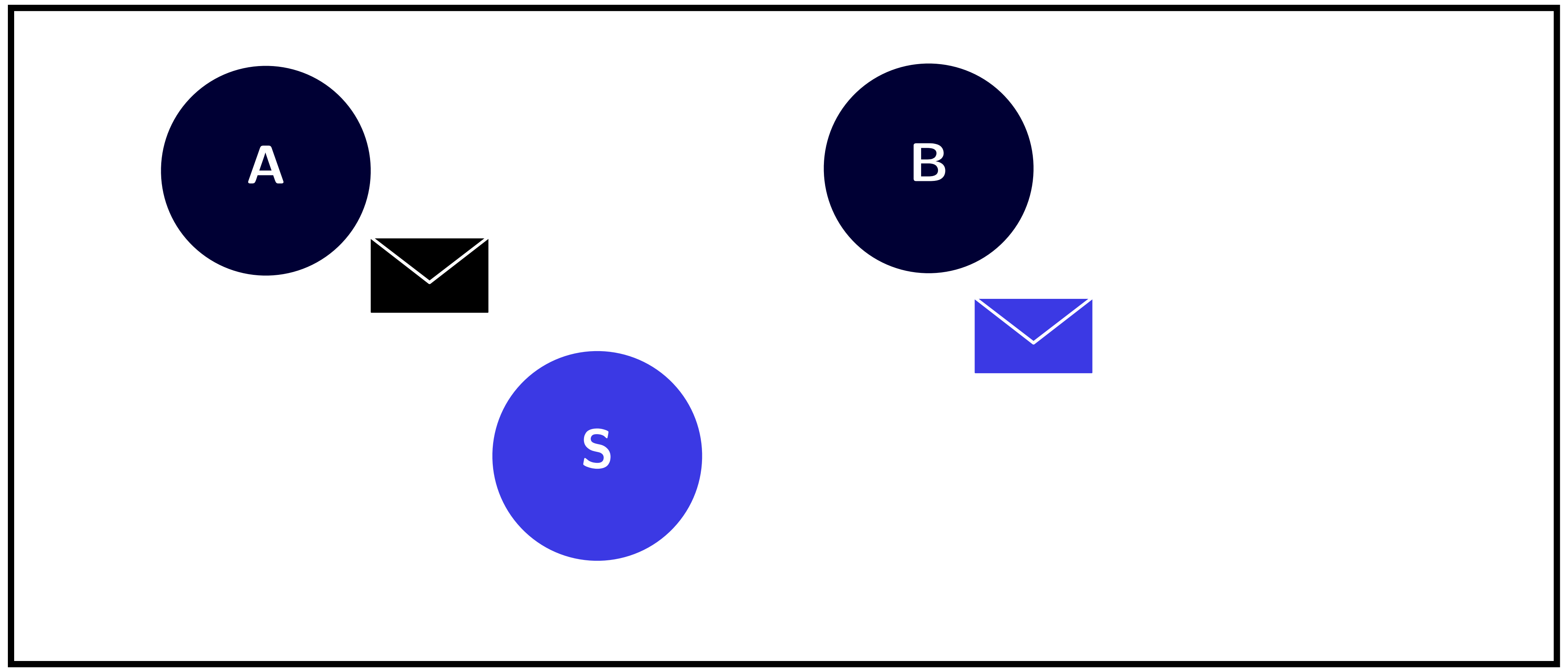
- A sends M1 to B <u>via S</u>.
- S sends M2 to B.



Encoding Preserves Semantics

"Original communication"

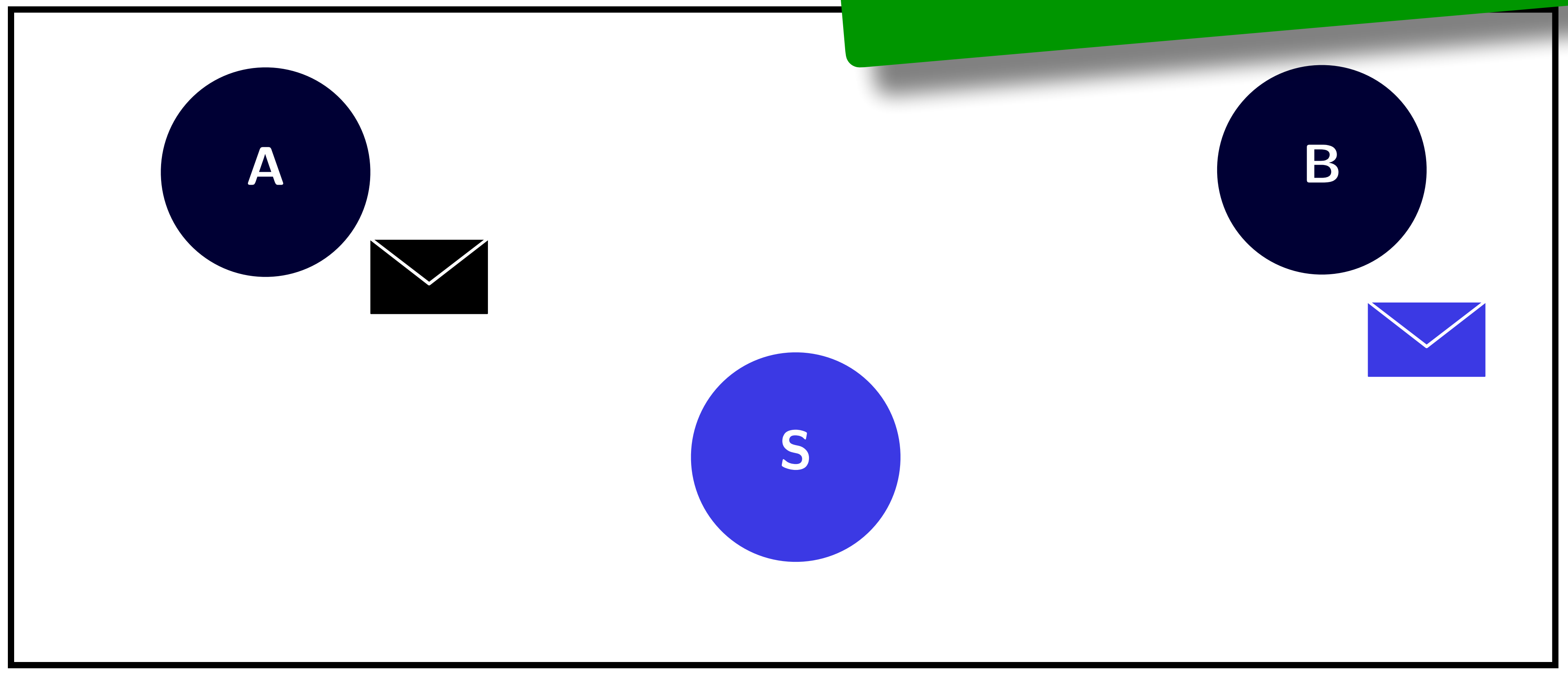
- A sends M1 to B.
- **S sends M2 to B.**



ROUTEDSESSIONS

- A sends M1 to B via S.
- **S sends M2 to B.**

via S allows this

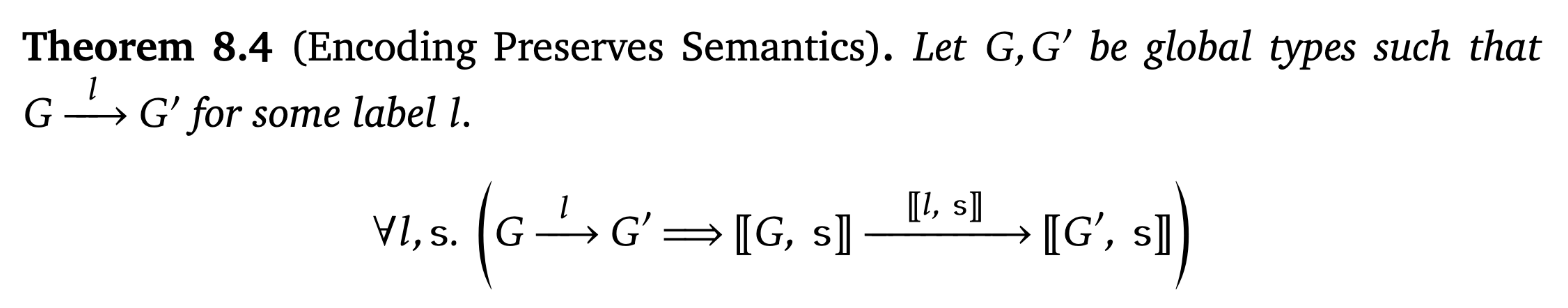


Encoding Preserves Semantics

“Original communication”

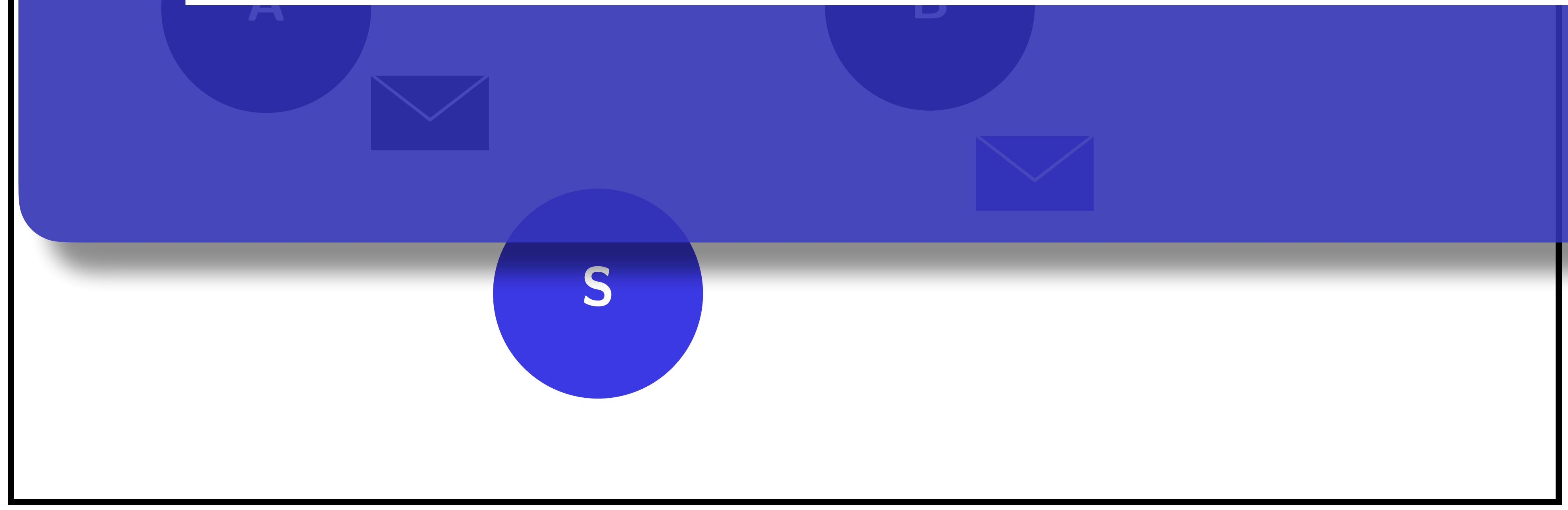
RoutedSessions

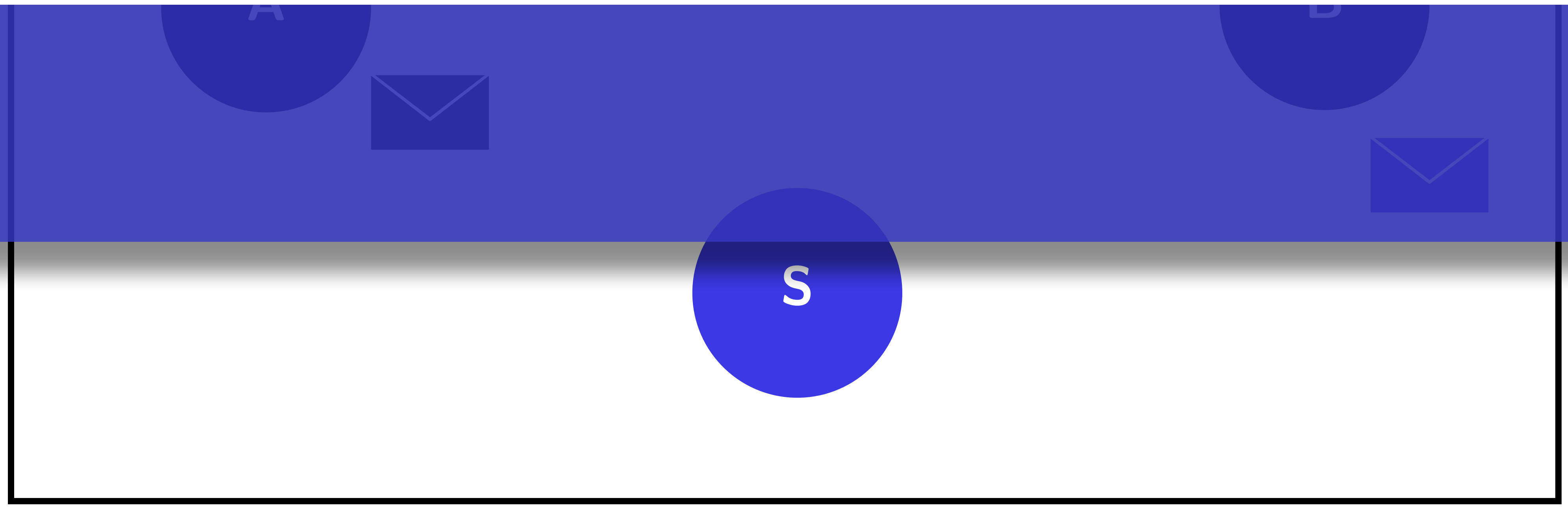
- A
- S



Theorem 8.4 (Encoding Preserves Semantics). *Let G, G' be global types such that $G \xrightarrow{l} G'$ for some label l.*

$$\forall l, \mathsf{s}.\ \left(G \xrightarrow{l} G' \Longrightarrow [\![G,\ \mathsf{s}]\!] \xrightarrow{[\![l,\ \mathsf{s}]\!]} [\![G',\ \mathsf{s}]\!] \right)$$





Evaluation

Chapter 8

Appendix A.1

- **Theorem 8.1: Trace Equivalence**
  Extended semantics on routed multiparty session types is sound and complete w.r.t. projection.

- **Theorem 8.2: Deadlock Freedom**
  Well-formed communication protocols do not get stuck.

- **Theorem 8.3: Encoding Preserves Well-formedness**
  If the original communication is well-formed, so is the encoded routed communication.

- **Theorem 8.4: Encoding Preserves Semantics**
  If the global type makes a step, the encoded global type makes a compatible step.

Evaluation

Chapter 8

Appendix A.1

- **Theorem 8.1: Trace Equivalence**
  Extended semantics on routed multiparty session types is sound and complete w.r.t. projection.

- **Theorem 8.2: Deadlock Freedom**
  Well-formed communication protocols do not get stuck.

- **Theorem 8.3: Encoding Preserves Well-formedness**
  If the original communication is well-formed, so is the encoded routed communication.

- **Theorem 8.4: Encoding Preserves Semantics**
  If the global type makes a step, the encoded global type makes a compatible step.

Evaluation

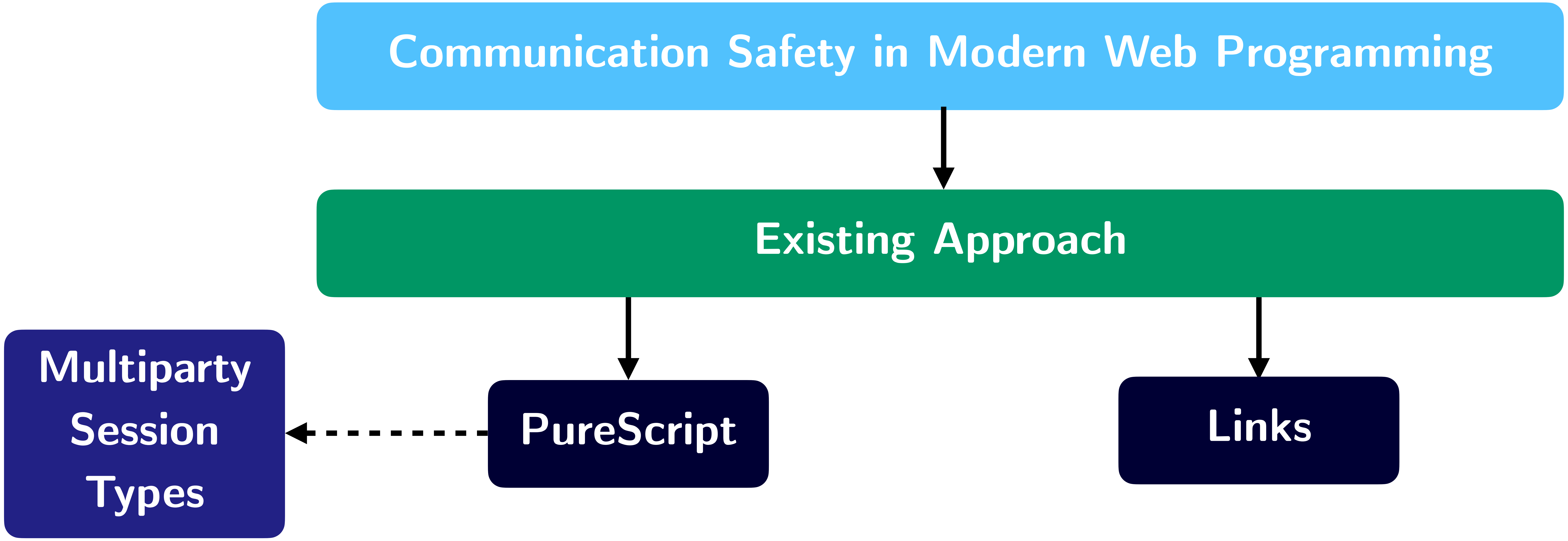
Chapter 8

Appendix A.1

- Lemma 8.1: Step Equivalence
- Lemma 8.2: (LTS) Preservation of Well-formedness
- Lemma 8.3: Progress for Well-formed Global Types
- Lemma 8.5: Encoding Preserves Projection
- Lemma A.1: Local LTS Preserves Merge
- Lemma A.3: Commutativity between Encoding and Substitution
- Lemma A.8: Encoding on Global Types Preserves Merge

Concluding Remarks

Communication Safety in Modern Web Programming



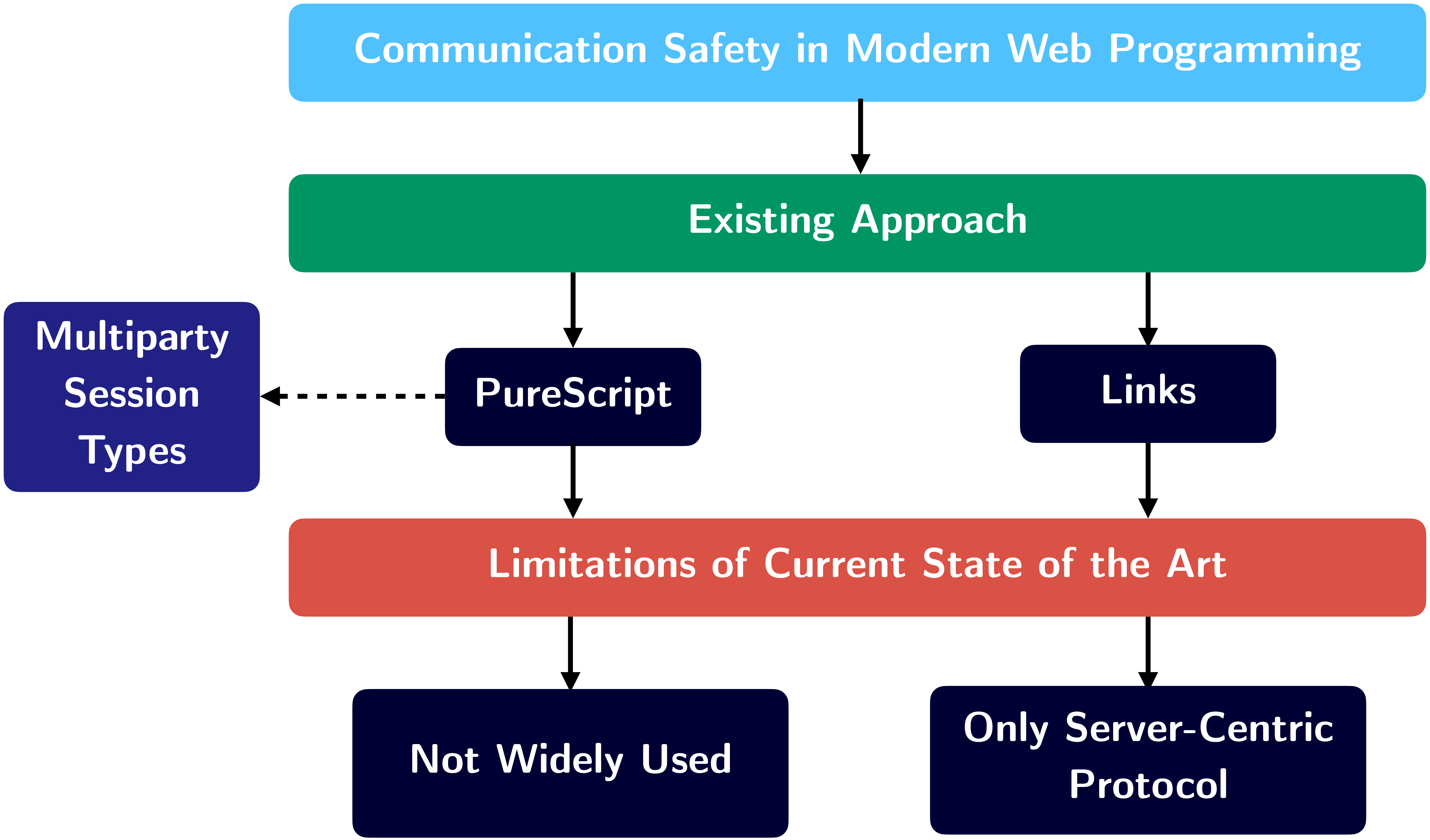
Communication Safety in Modern Web Programming

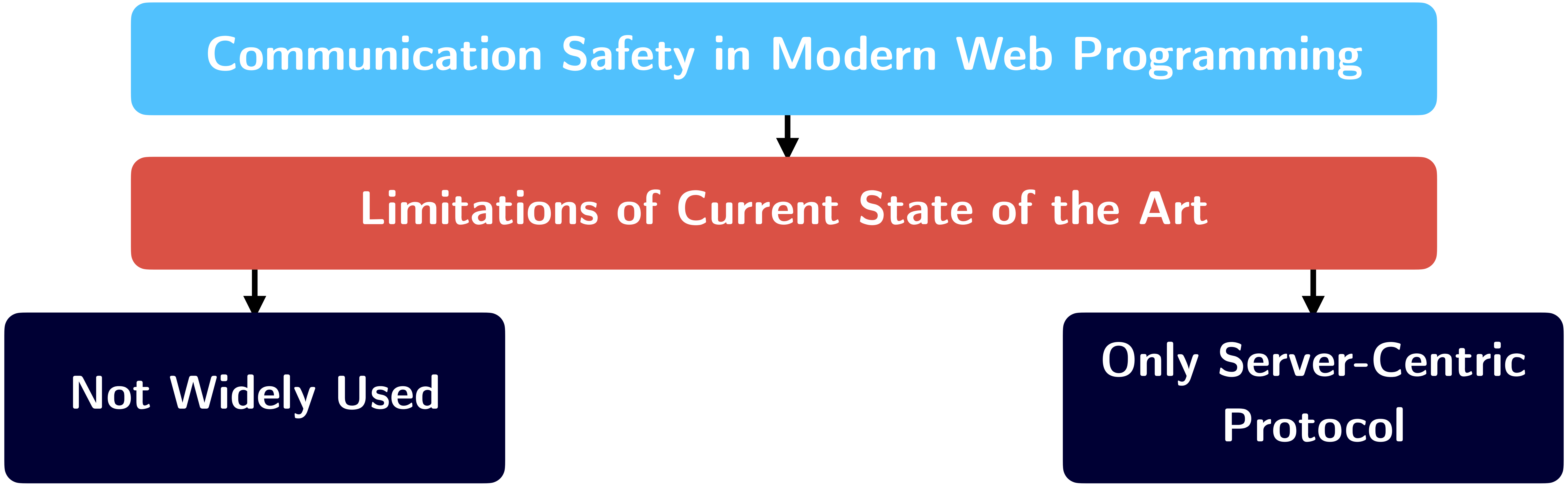
Existing Approach

Multiparty Session Types

PureScript

Links


Communication Safety in Modern Web Programming
Existing Approach
Multiparty Session Types
PureScript
Links
Limitations of Current State of the Art
Not Widely Used
Only Server-Centric Protocol

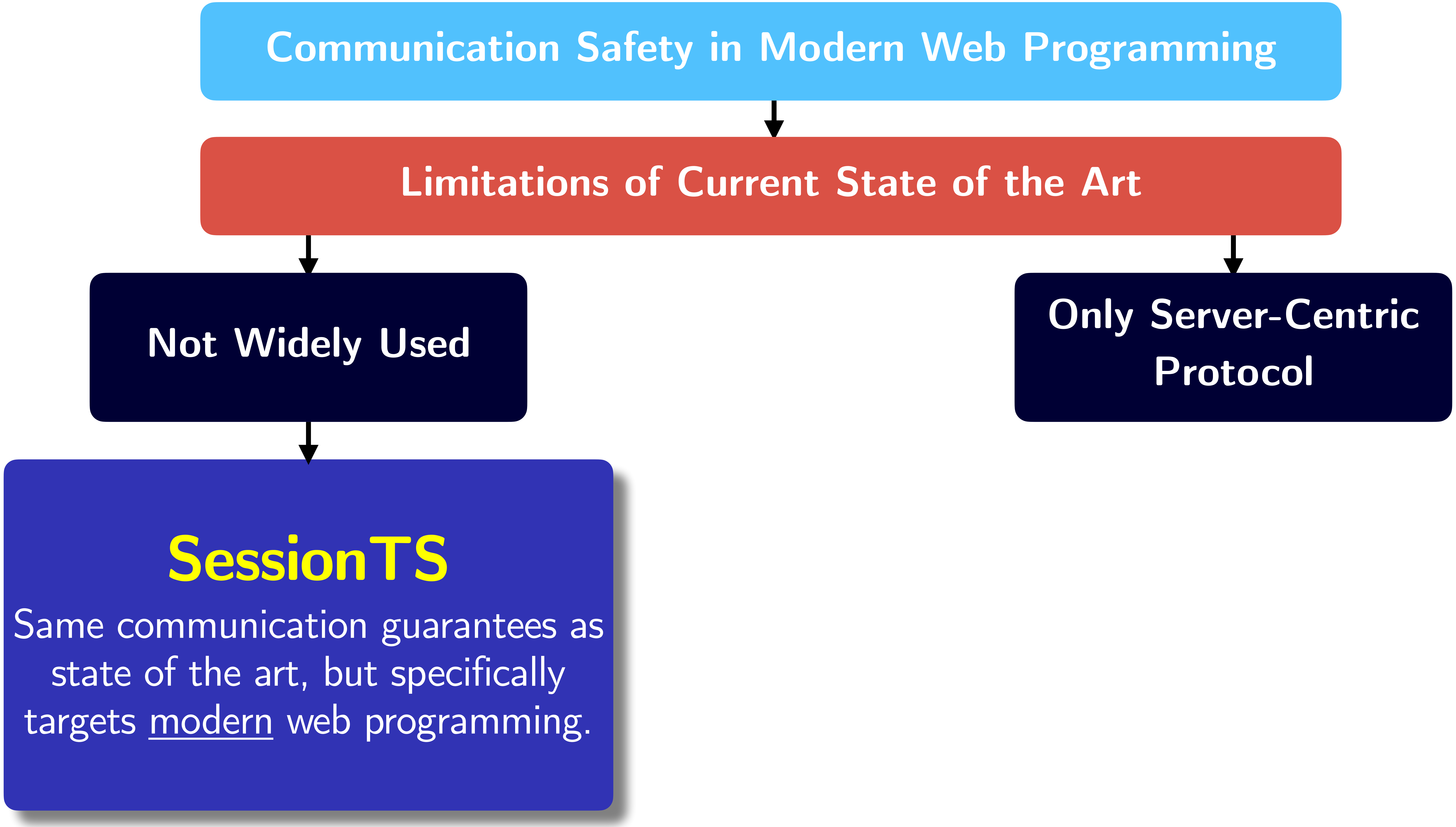


Communication Safety in Modern Web Programming

Limitations of Current State of the Art

Not Widely Used

Only Server-Centric Protocol



Communication Safety in Modern Web Programming

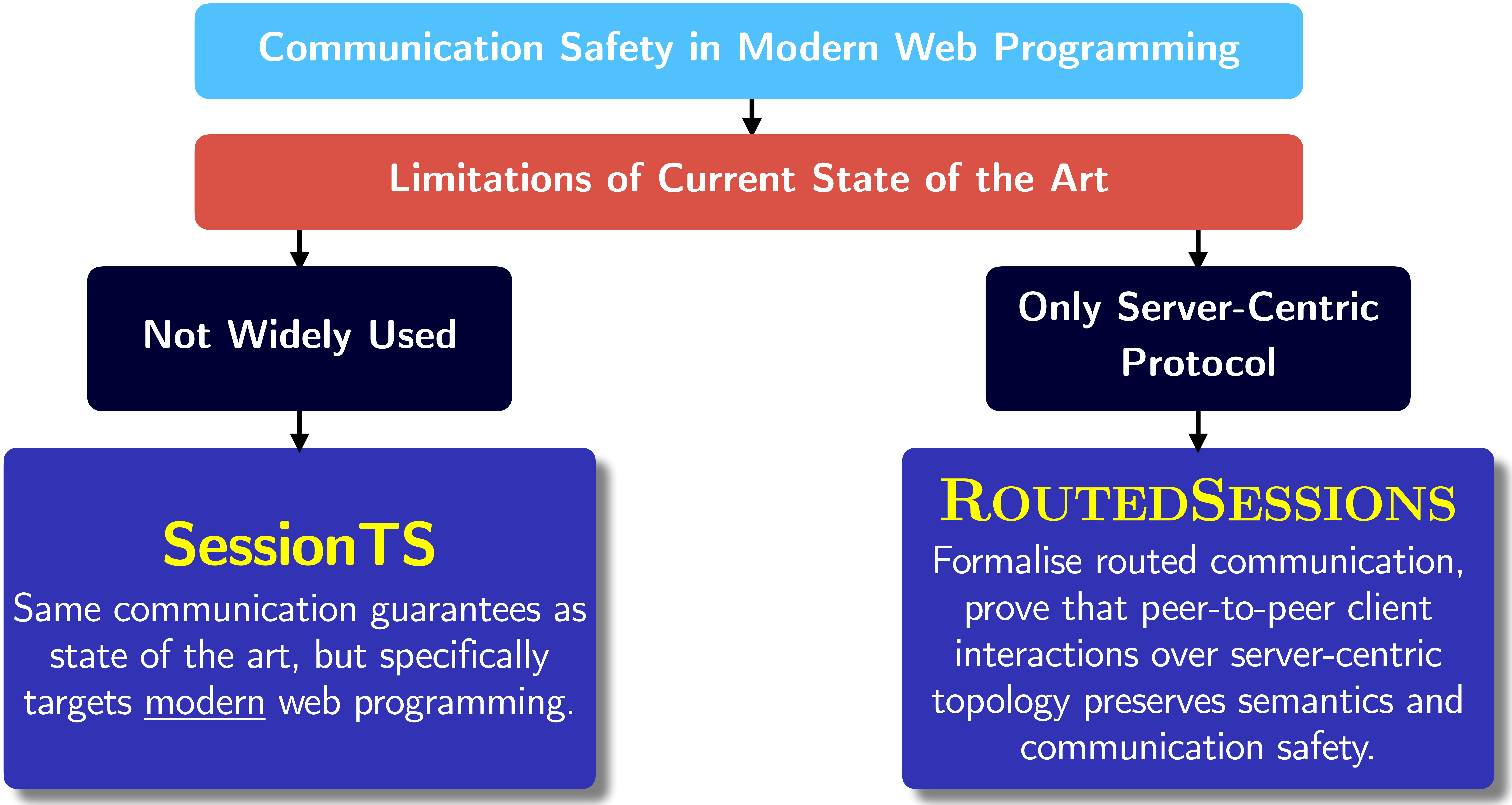
Limitations of Current State of the Art

Not Widely Used

Only Server-Centric Protocol

SessionTS

Same communication guarantees as state of the art, but specifically targets modern web programming.



Communication Safety in Modern Web Programming

Limitations of Current State of the Art

Not Widely Used

Only Server-Centric Protocol

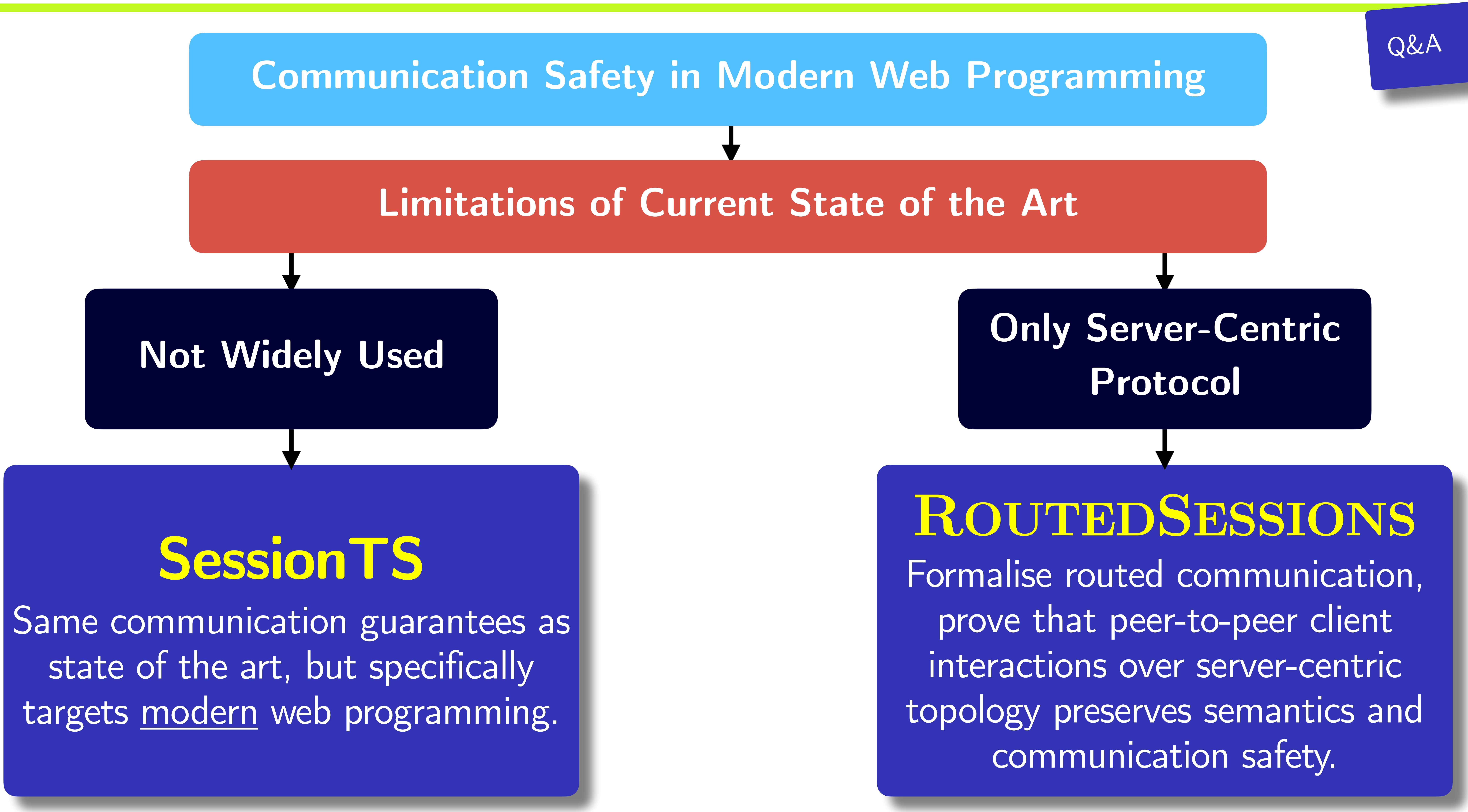
SessionTS

Same communication guarantees as state of the art, but specifically targets <u>modern</u> web programming.

RoutedSessions

Formalise routed communication, prove that peer-to-peer client interactions over server-centric topology preserves semantics and communication safety.

https://github.com/ansonmiu0214/TypeScript-Multiparty-Sessions



Communication Safety in Modern Web Programming

Limitations of Current State of the Art

Not Widely Used

SessionTS

Same communication guarantees as state of the art, but specifically targets modern web programming.

Only Server-Centric Protocol

RoutedSessions

Formalise routed communication, prove that peer-to-peer client interactions over server-centric topology preserves semantics and communication safety.

https://github.com/ansonmiu0214/TypeScript-Multiparty-Sessions