Communication-Safe Web Programming in TypeScript with Routed Multiparty Session Types

Anson Miu (1)(2), Francisco Ferreira (1), Nobuko Yoshida (1), Fangyi Zhou (1)

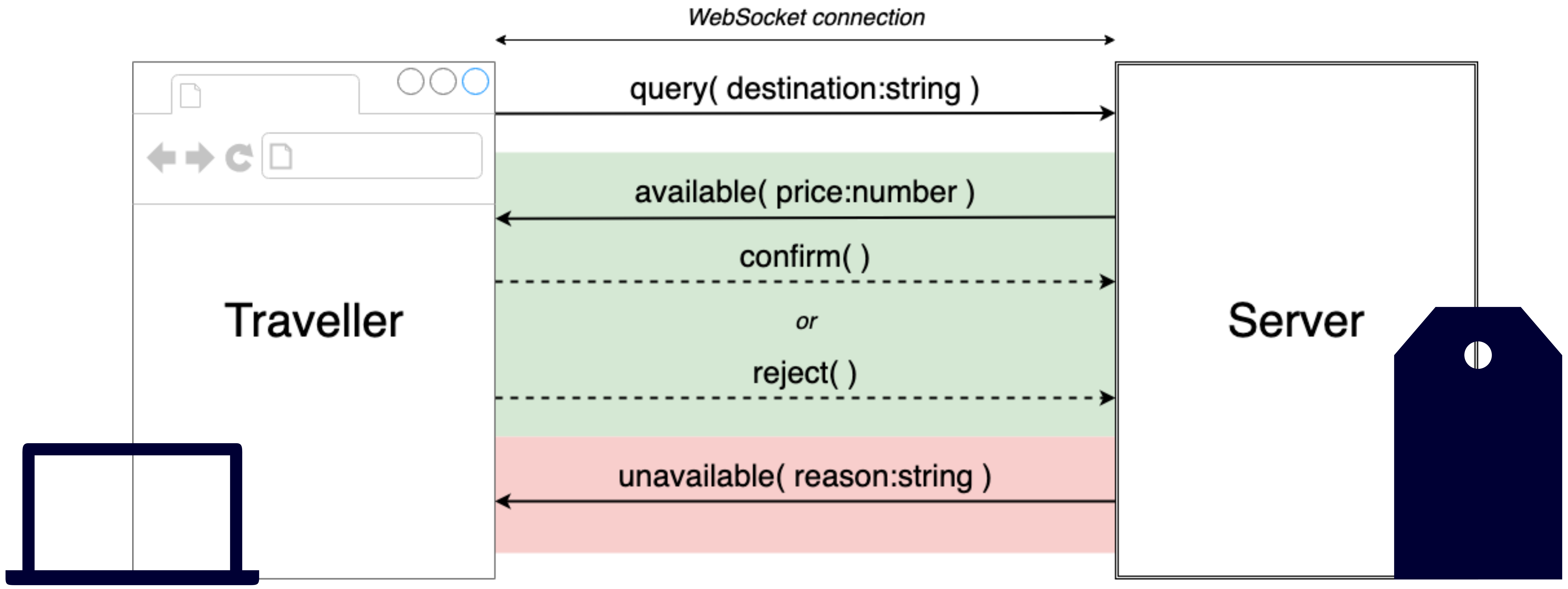
(1) Imperial College London (2) Bloomberg Engineering

CC 2021 - March 3, 2021

Communication-Safe Web Programming in TypeScript with Routed Multiparty Session Types

Example: “Travel Agency”

Endpoints interacting over WebSocket connections

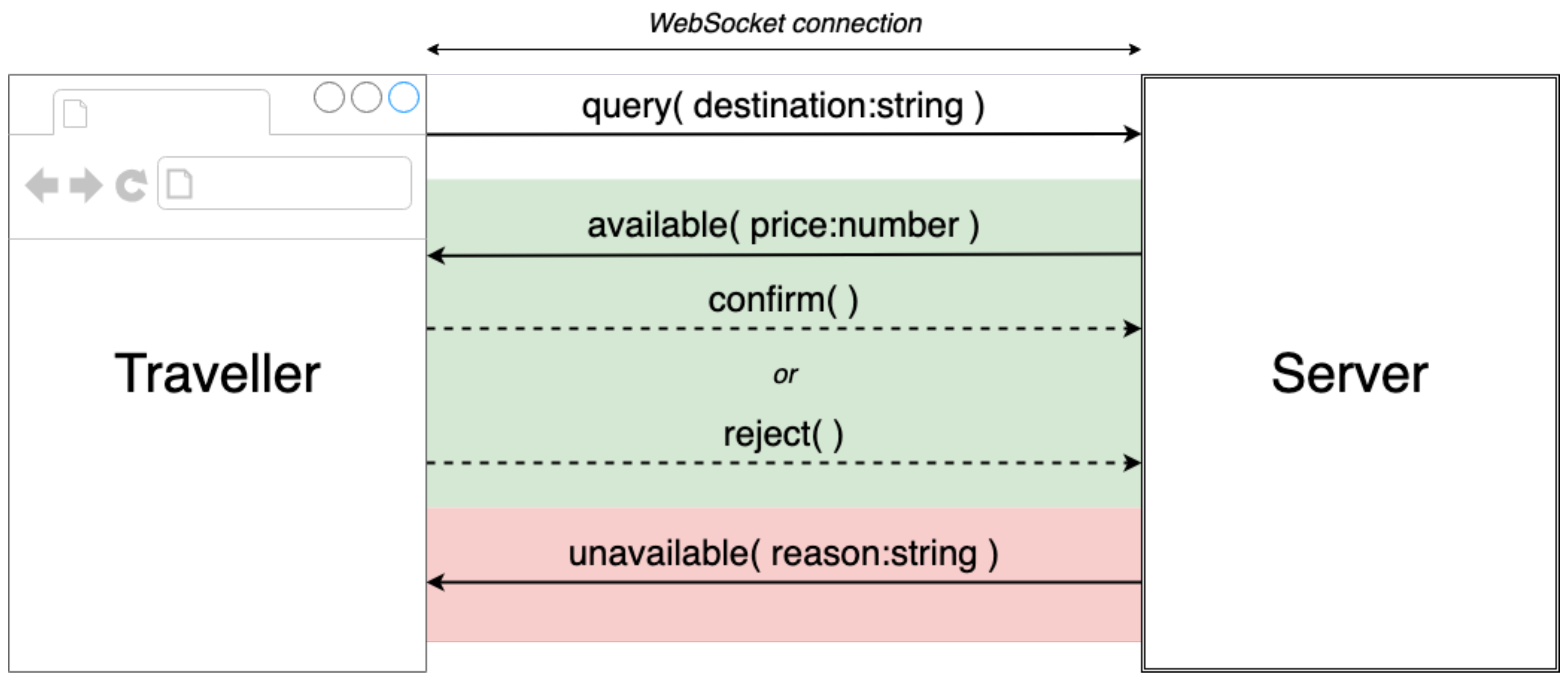


WebSocket connection
query(destination:string)
available(price:number)
confirm( )
or
reject( )
unavailable(reason:string)
Traveller
Server

Example: “Travel Agency”

Endpoints interacting over WebSocket connections

- Traveller asks Server about details for a particular destination
- If available:
    - Server receives 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
Server

Example: "Travel Agency"

Potential Communication Errors

- Traveller asks Server about 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`?

Example: "Travel Agency"

Potential Communication Errors

- Traveller asks Server about 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?

Example: “Travel Agency”

Potential Communication Errors

- Traveller asks Server about 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.

Session Cancellation

What if Traveller leaves the session prematurely before responding to the Server’s quotation?

Communication-Safe Web Programming in TypeScript with Routed Multiparty Session Types

Communication-Safe Web Programming in TypeScript with Routed Multiparty Session Types

Applying Multiparty Session Types

Towards Communication Safety

Global Type

Global Type

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

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

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

Applying Multiparty Session Types

Towards Communication Safety

Global Type

$\downarrow$ Projection

Local Types

$$G = \mu \mathtt{t}.\mathsf{Traveller} \to \mathsf{Server} : \mathsf{Destination}(\mathtt{string}).$$
$$\mathsf{Server} \to \mathsf{Traveller} \begin{cases} \mathsf{Available}(\mathtt{number}) : & G_{\mathsf{Available}} \\ \mathsf{Full}() : & \mathtt{t} \end{cases}$$

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

Global Type

$\searrow$

Server Local Type

$$T_{Server} = \mu \mathtt{t}.\mathsf{Traveller}\ \&\ \mathsf{Destination}(\mathtt{string}).$$
$$\boxed{\mathsf{Traveller} \oplus \begin{cases} \mathsf{Available}(\mathtt{number}) : & T_{\mathsf{Available}} \\ \mathsf{Full}() : & \mathtt{t} \end{cases}}$$

Selection

$$T_{\mathsf{Available}} = \boxed{\mathsf{Traveller}\ \&\ \begin{cases} \mathsf{Confirm}(\mathtt{Cred}) : & \mathtt{end} \\ \mathsf{Reject}() : & \mathtt{end} \end{cases}}$$

Branching

Applying Multiparty Session Types

Towards Communication Safety

Global Type

↓ Projection

Local Types

↓ Verify

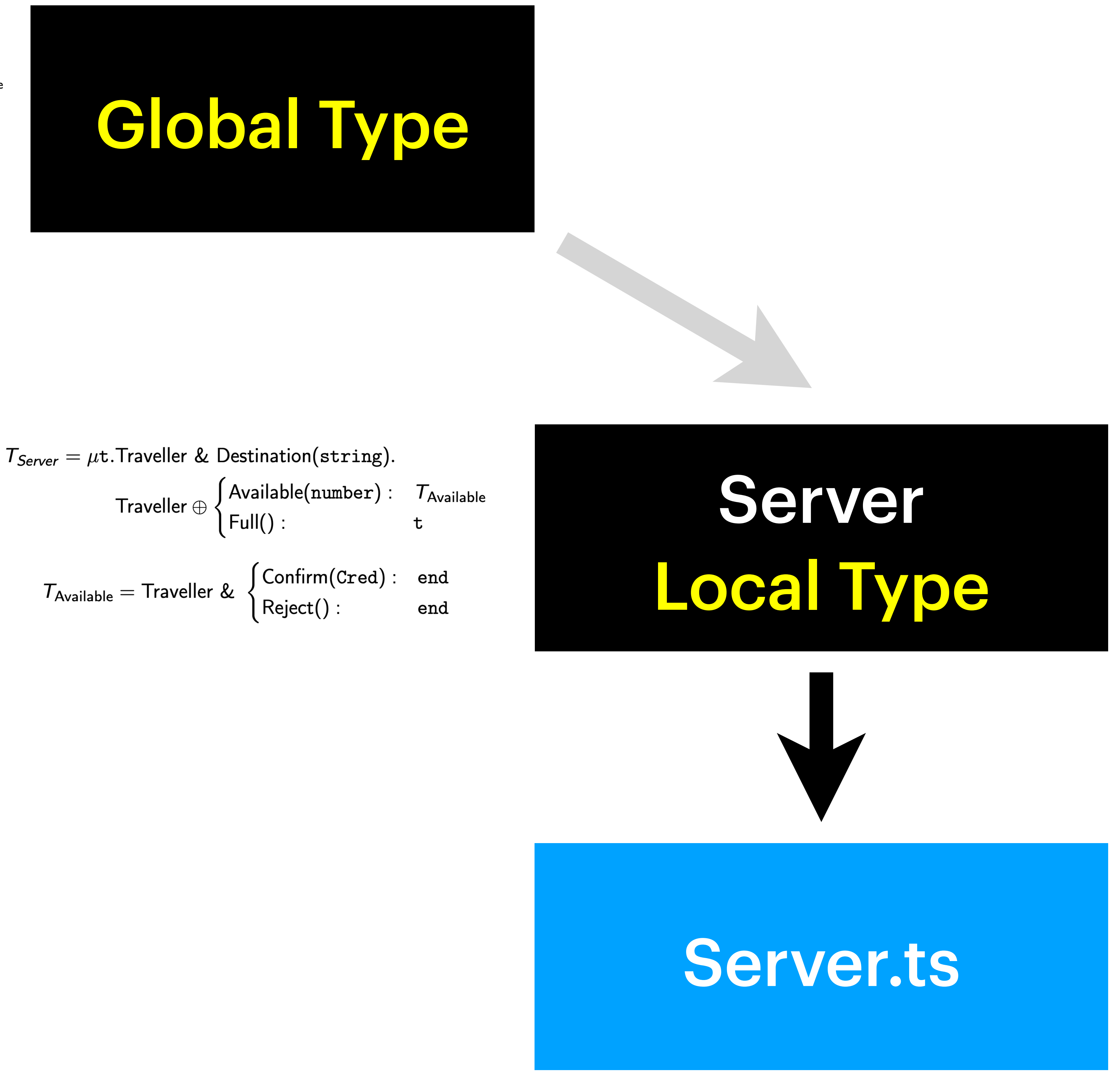
Endpoint Implementation

$$G = \mu\texttt{t}.\mathsf{Traveller} \to \mathsf{Server} : \mathsf{Destination}(\texttt{string}).$$
$$\mathsf{Server} \to \mathsf{Traveller} \begin{cases} \mathsf{Available}(\texttt{number}) : & G_{\mathsf{Available}} \\ \mathsf{Full}() : & \texttt{t} \end{cases}$$

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

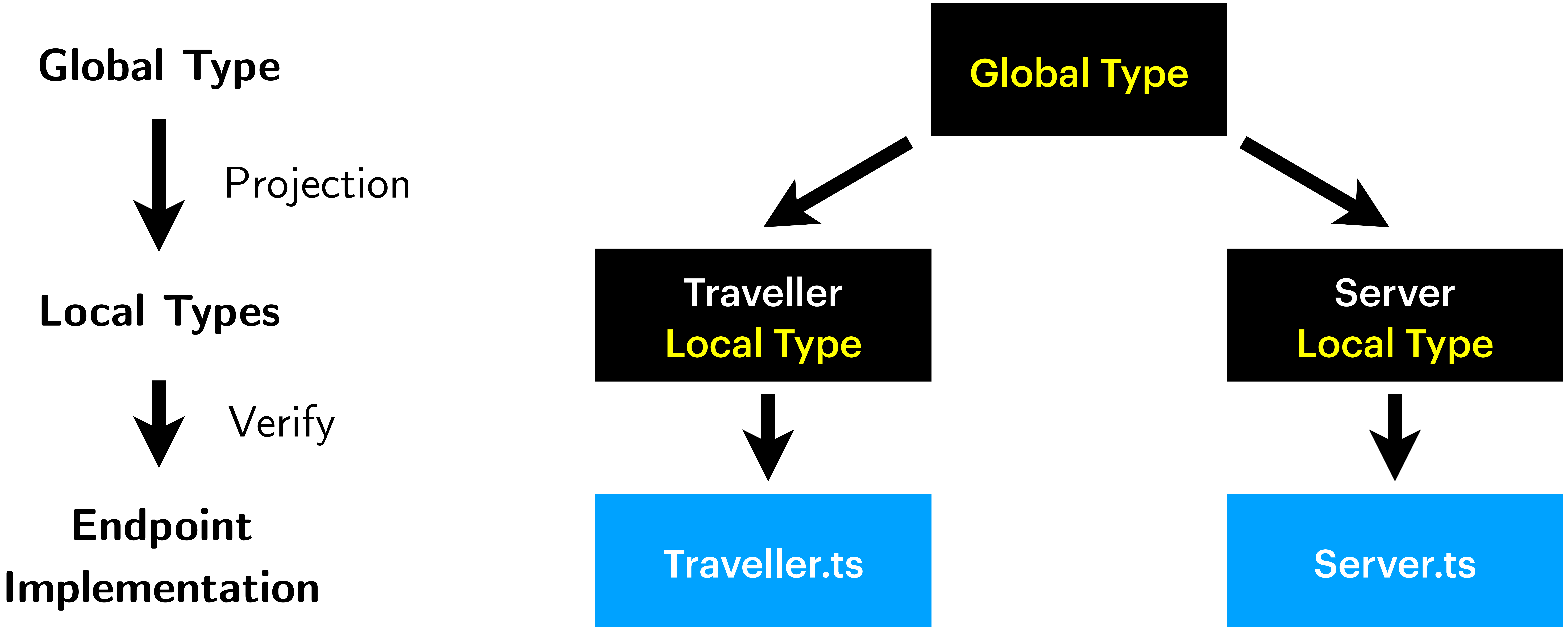
$$T_{Server} = \mu\texttt{t}.\mathsf{Traveller} \,\&\, \mathsf{Destination}(\texttt{string}).$$
$$\mathsf{Traveller} \oplus \begin{cases} \mathsf{Available}(\texttt{number}) : & T_{\mathsf{Available}} \\ \mathsf{Full}() : & \texttt{t} \end{cases}$$

$$T_{\mathsf{Available}} = \mathsf{Traveller} \,\&\, \begin{cases} \mathsf{Confirm}(\texttt{Cred}) : & \texttt{end} \\ \mathsf{Reject}() : & \texttt{end} \end{cases}$$



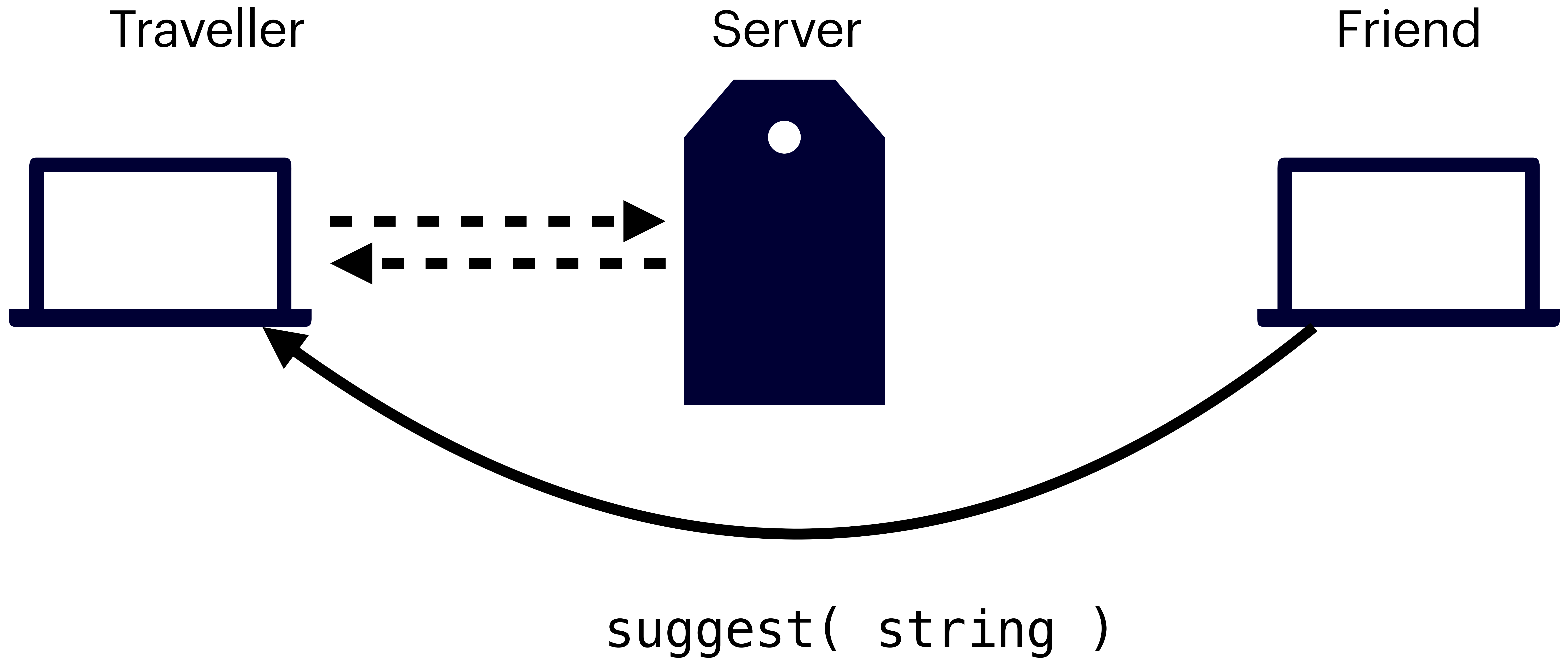
Applying Multiparty Session Types

Towards Communication Safety



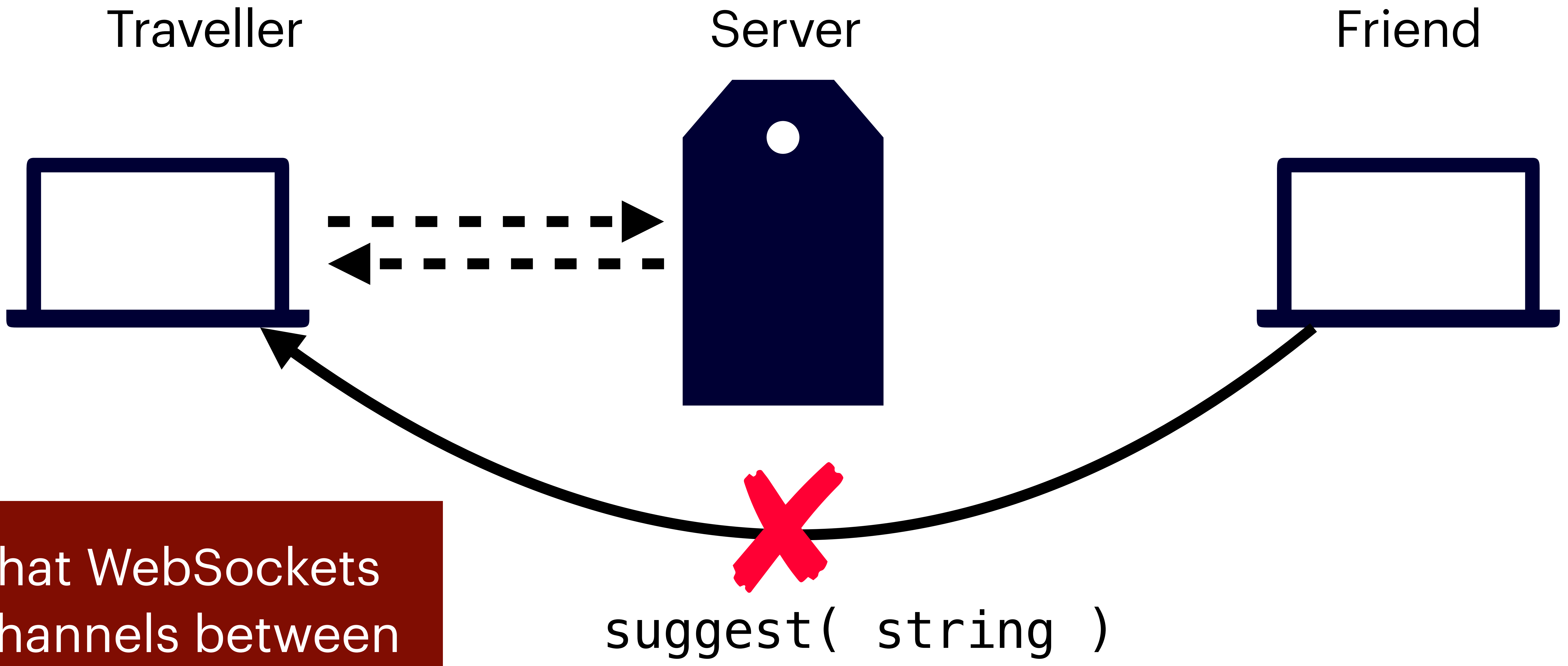
“Travel with a Friend”

Extending *Travel Agency* with Client-to-Client Interactions



“Travel with a Friend”

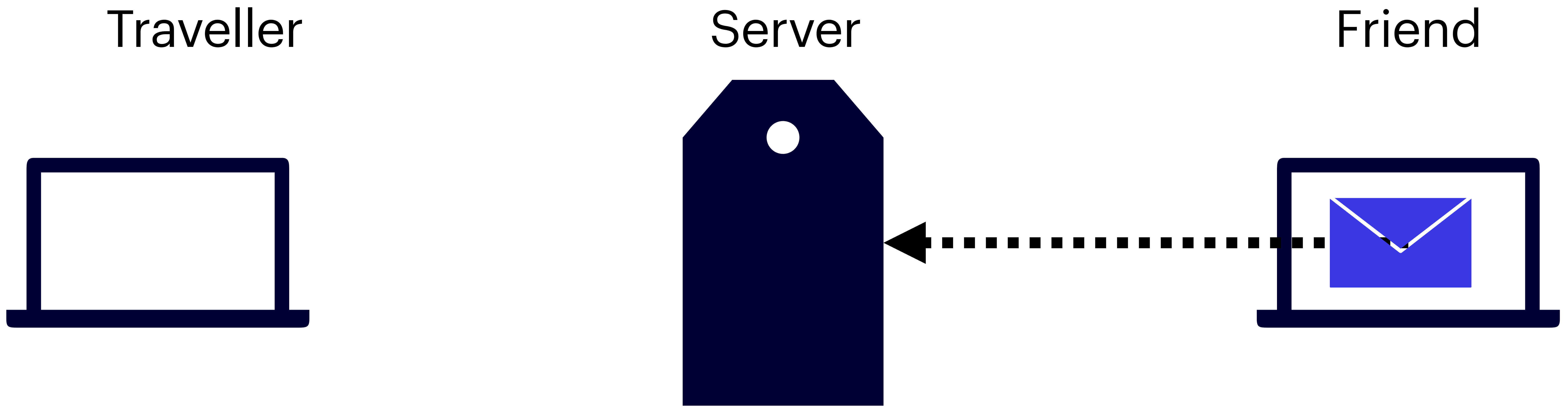
Extending *Travel Agency* with Client-to-Client Interactions



Recall that WebSockets define channels between client and server.

“Travel with a Friend”

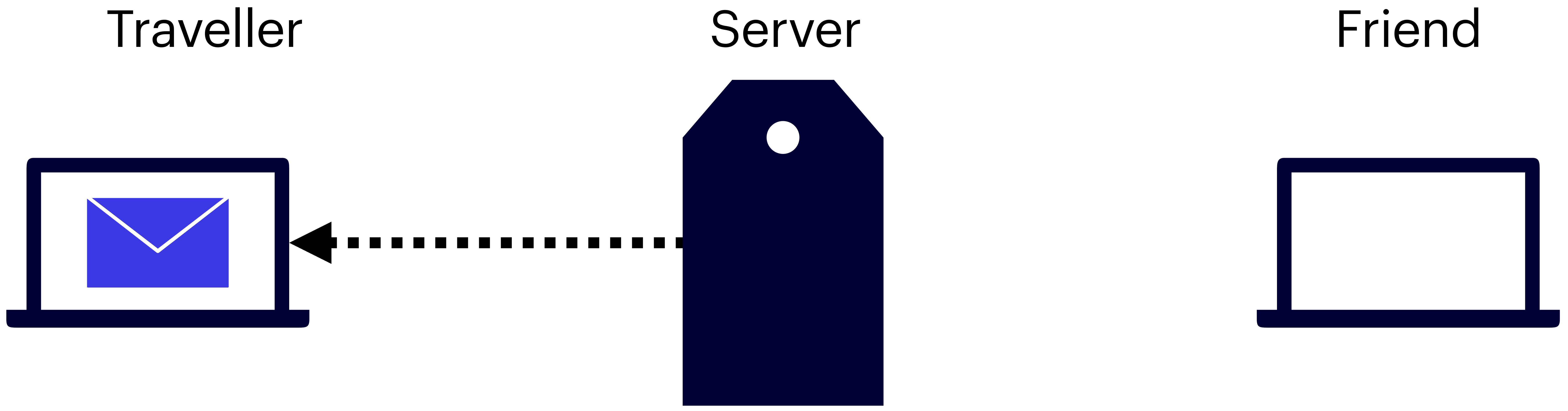
Extending *Travel Agency* with Client-to-Client Interactions



```
Traveller!suggest( string )
```

“Travel with a Friend”

Extending *Travel Agency* with Client-to-Client Interactions



```
Traveller!suggest( string )
```

How to **formalise** this routing mechanism?

Contributions

- *STScript* - a toolchain that generates TypeScript APIs that statically guarantee communication-safe web development

- *RouST* - a new session type theory that supports multiparty communications with routing mechanisms

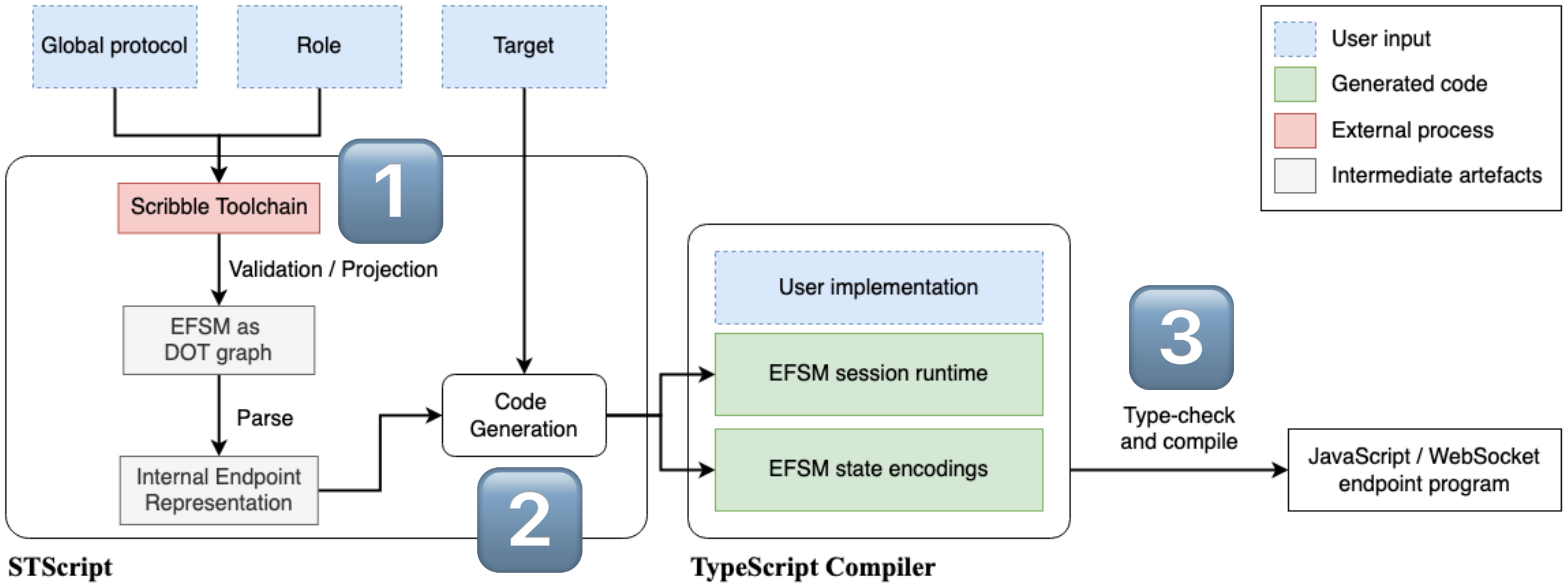
Contributions

- *STScript* - a toolchain that generates TypeScript APIs that statically guarantee communication-safe web development

- *RouST* - a new session type theory that supports multiparty communications with routing mechanisms

STScript

Session Type API Generation Toolchain for TypeScript



`https://github.com/STScript-2020/cc21-artifact`

1️⃣ Specify Communications Aspect

Using the Scribble Protocol Specification Language

```
type <typescript> “Credentials” from “./Payment” as Cred;
global protocol TravelAgency(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);
}}
```

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

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

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

1️⃣ Specify Communications Aspect

Using the Scribble Protocol Specification Language

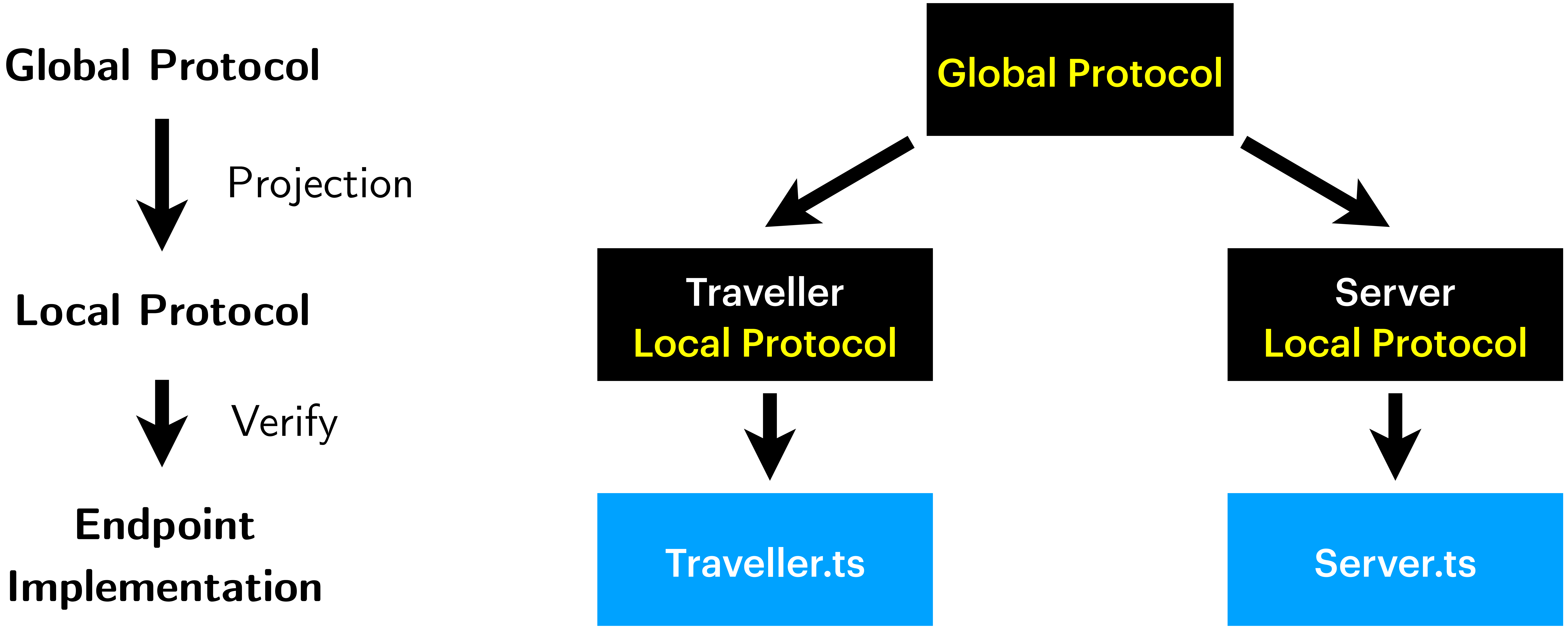
```
type <typescript> “Credentials” from “./Payment” as Cred;
global protocol TravelAgency(role Traveller, role Server) {
    Destination(string) from Traveller to Server;
    choice at Server {
        Available(number) from Server to Traveller;
        choice at Traveller {
            Confirm(Cred)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

$$G = \mu t.\boxed{\mathsf{Traveller} \to \mathsf{Server} : \mathsf{Destination}(\mathtt{string}).} \quad \mathsf{Server} \to \mathsf{Traveller} \begin{cases} \mathsf{Available}(\mathtt{number}) : & G_{\mathsf{Available}} \\ \boxed{\mathsf{Full}() : \qquad \mathtt{t}} \end{cases}$$

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

2️⃣ Endpoint API Generation

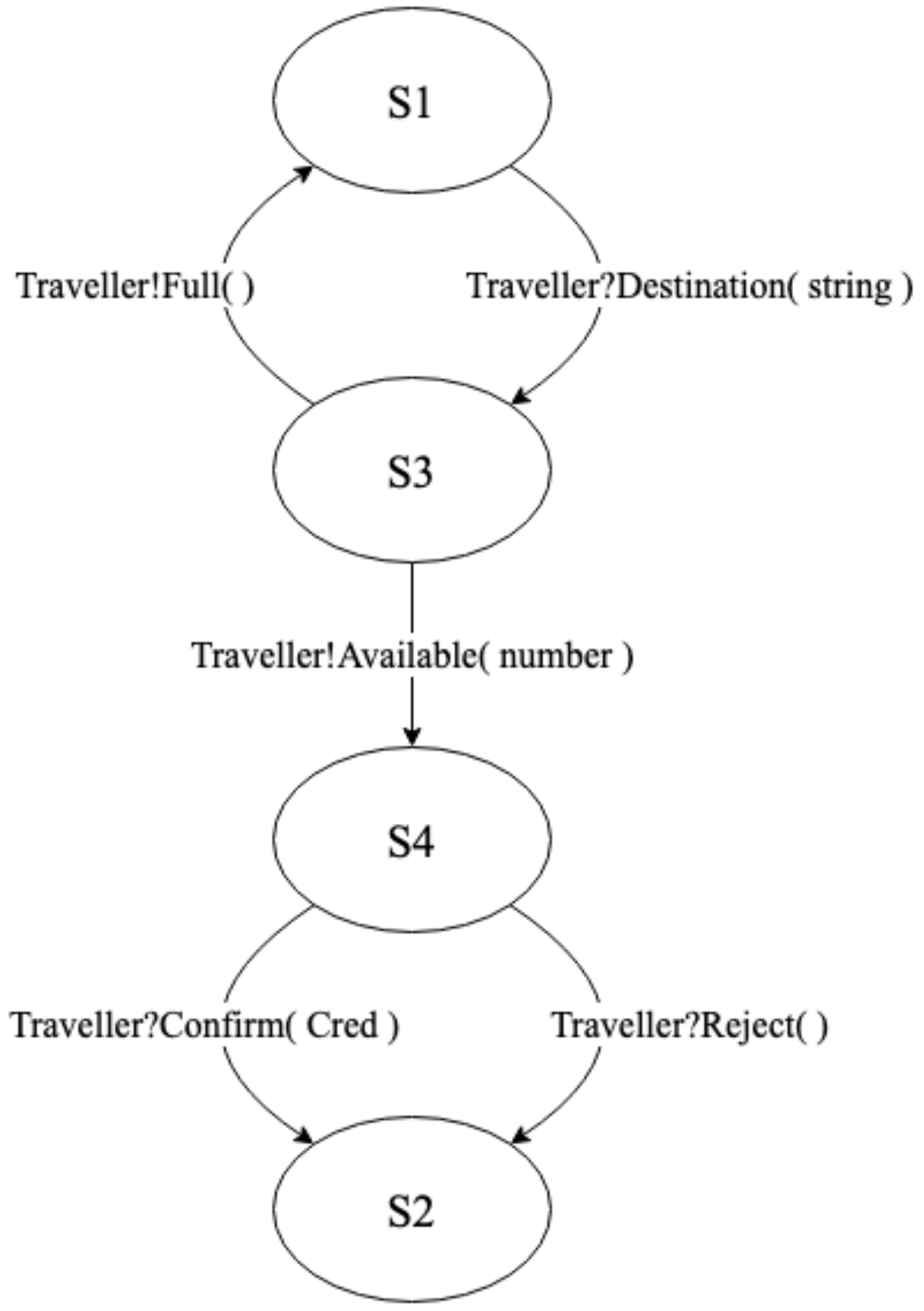
Local Protocol as Endpoint Finite State Machine (EFSM)

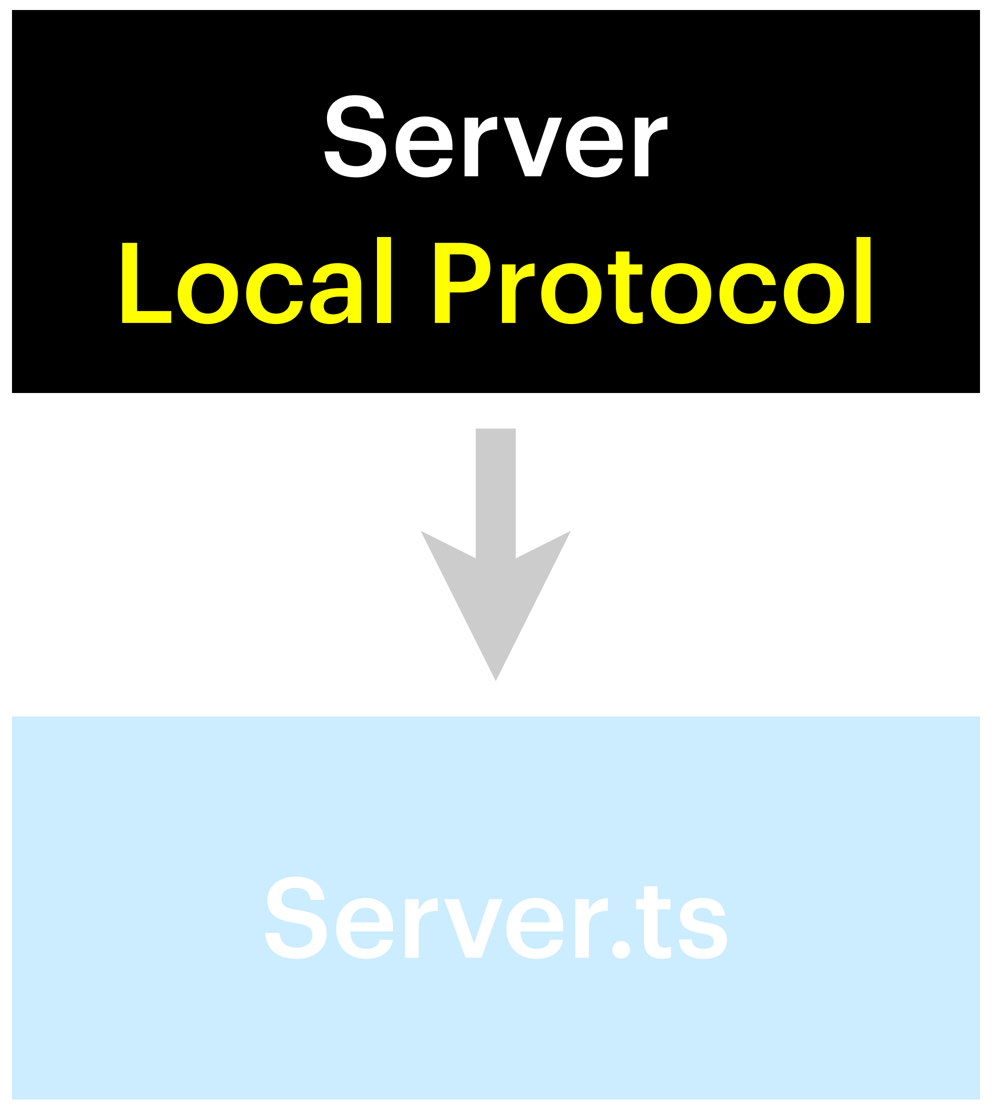


2️⃣ Endpoint API Generation

Local Protocol as Endpoint Finite State Machine (EFSM)

- Transitions represent IO actions, either send or receive
- Each state has its set of permitted IO actions
- Verify endpoint implementation to respect valid traces of its EFSM

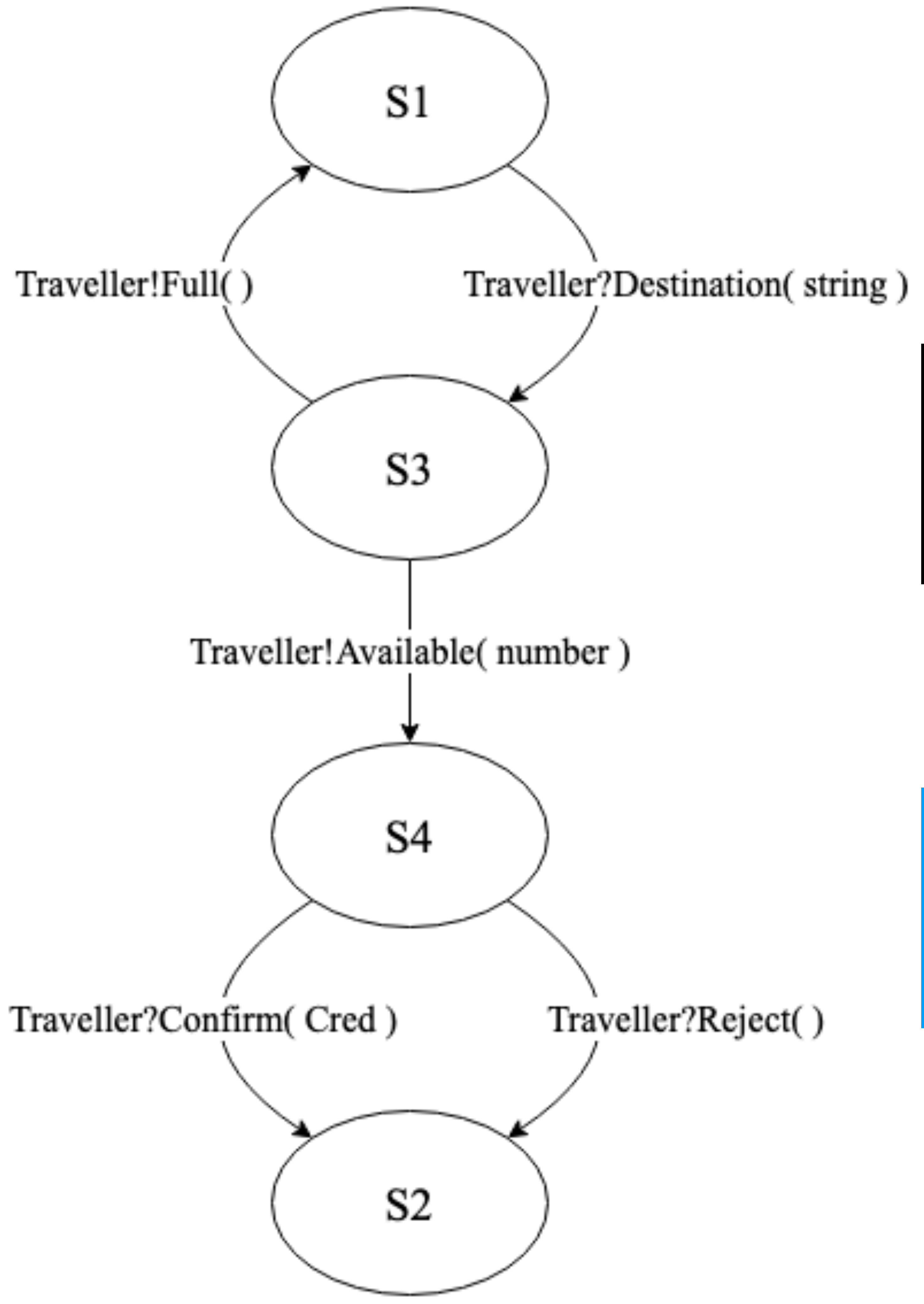

S1
Traveller!Full( )
Traveller?Destination(string)
S3
Traveller!Available(number)
S4
Traveller?Confirm(Cred)
Traveller?Reject( )
S2



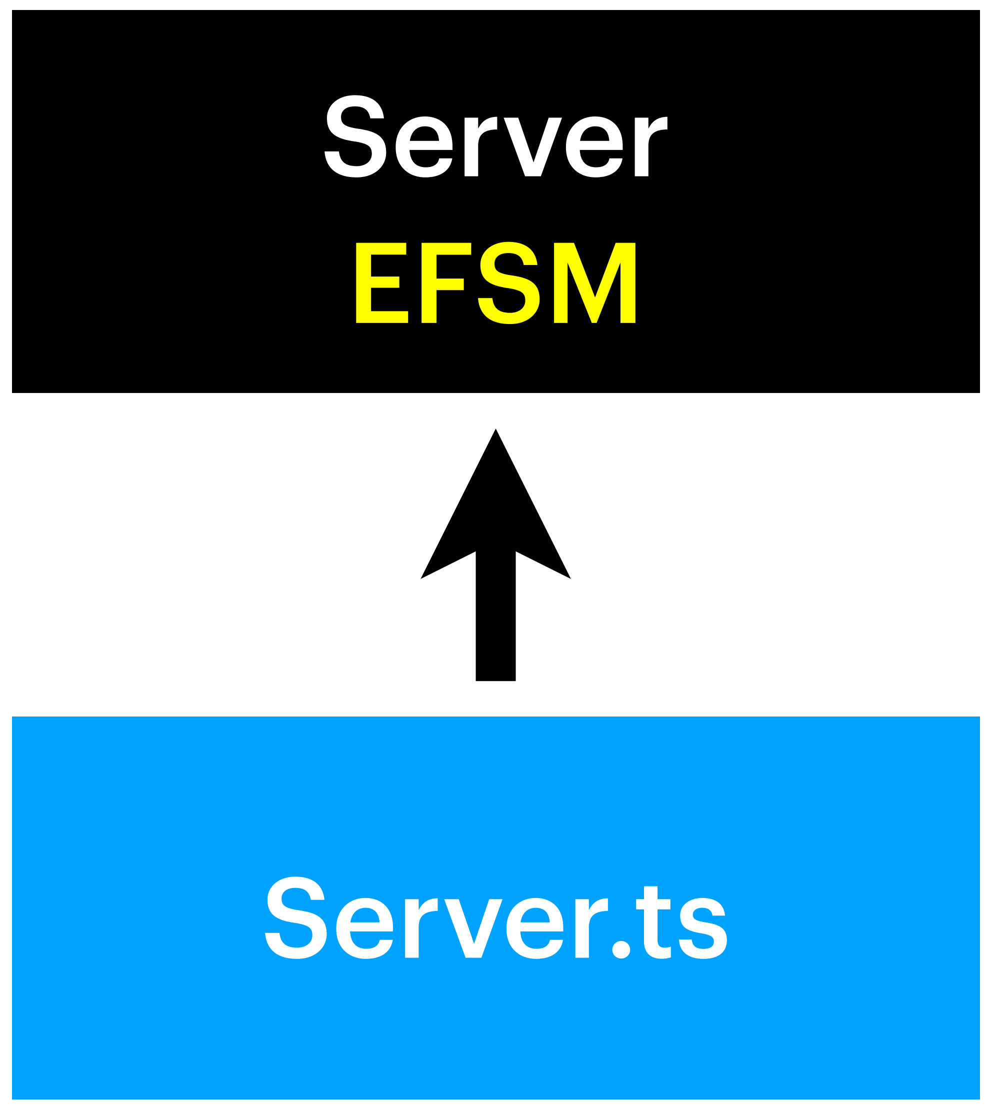
2️⃣ Endpoint API Generation

Local Protocol as Endpoint Finite State Machine (EFSM)

- Transitions represent IO actions, either send or receive
- Each state has its set of permitted IO actions
- Verify endpoint implementation to respect valid traces of its EFSM



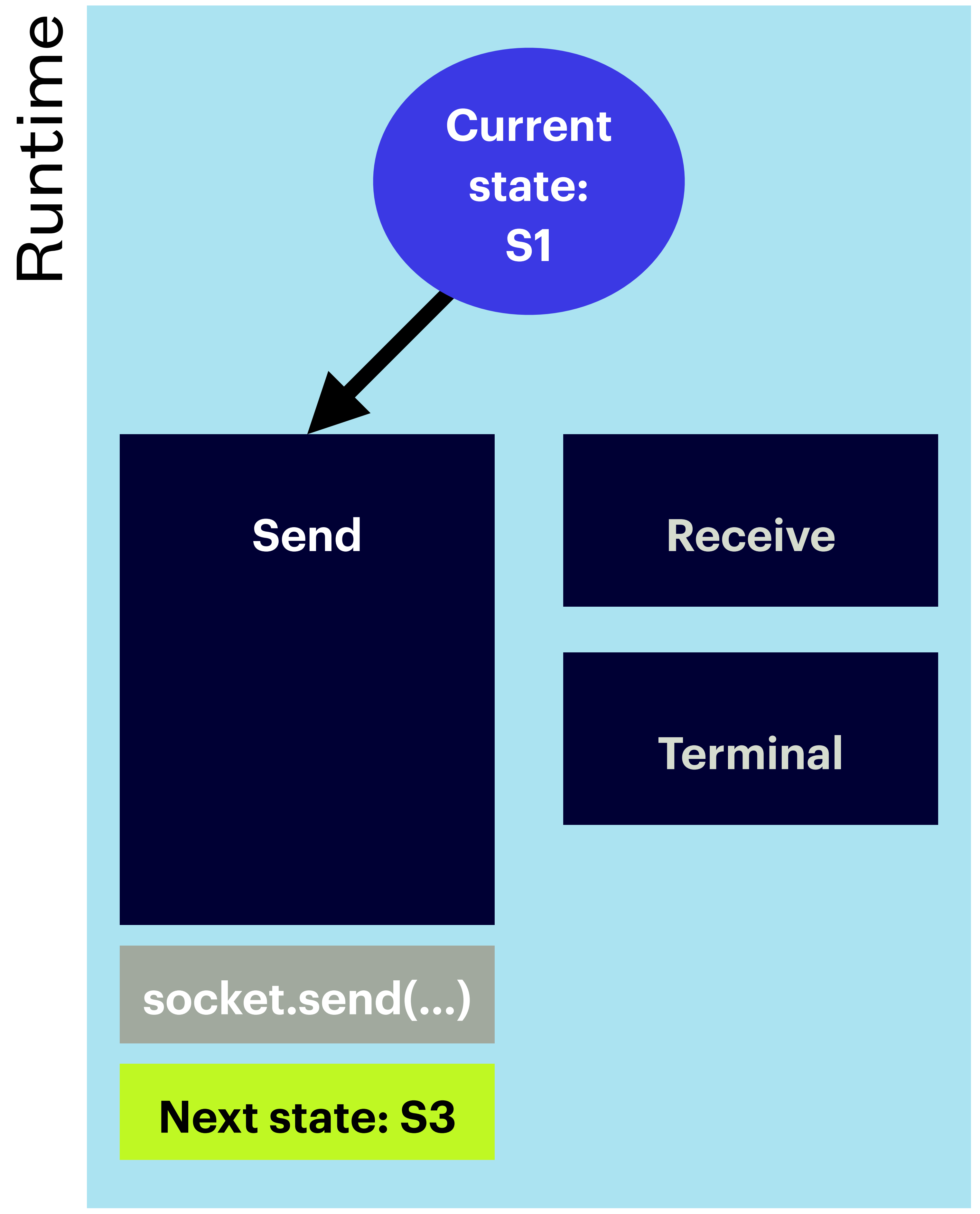
$$\begin{tikzcd} S1 \arrow[d, bend left, "\text{Traveller?Destination(string)}"] \\ S3 \arrow[u, bend left, "\text{Traveller!Full( )}"] \arrow[d, "\text{Traveller!Available(number)}"] \\ S4 \arrow[d, bend right, "\text{Traveller?Confirm(Cred)}"'] \arrow[d, bend left, "\text{Traveller?Reject( )}"] \\ S2 \end{tikzcd}$$



API Generation - Design Philosophy

Generate Correct-by-Construction APIs

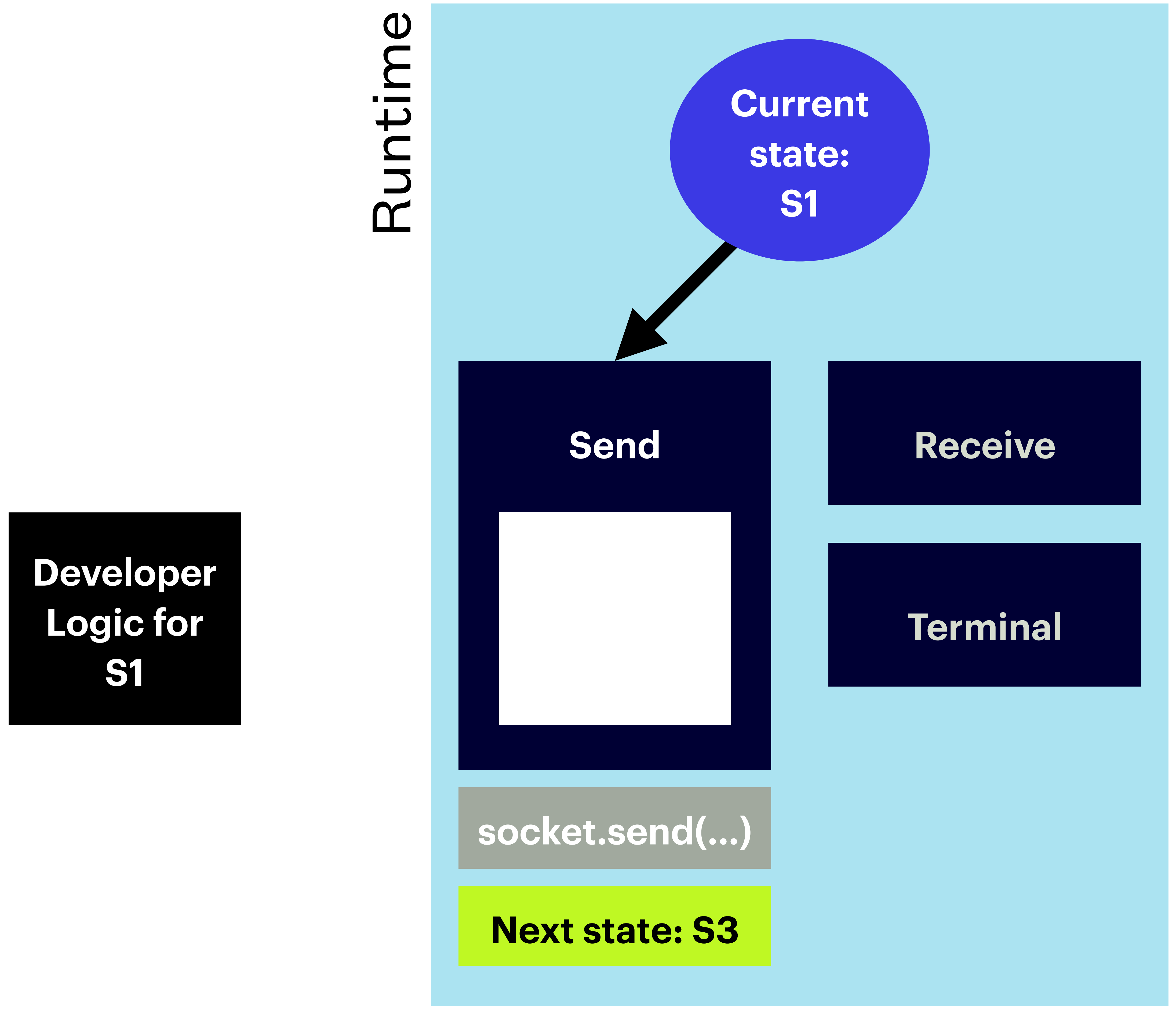
- We generate a session runtime to execute EFSM
  - Performs I/O action for current state
- We construct types for injecting business logic
  - What to send? How to handle receive?
- Developer instantiates session runtime with custom implementations



API Generation - Design Philosophy

Generate Correct-by-Construction APIs

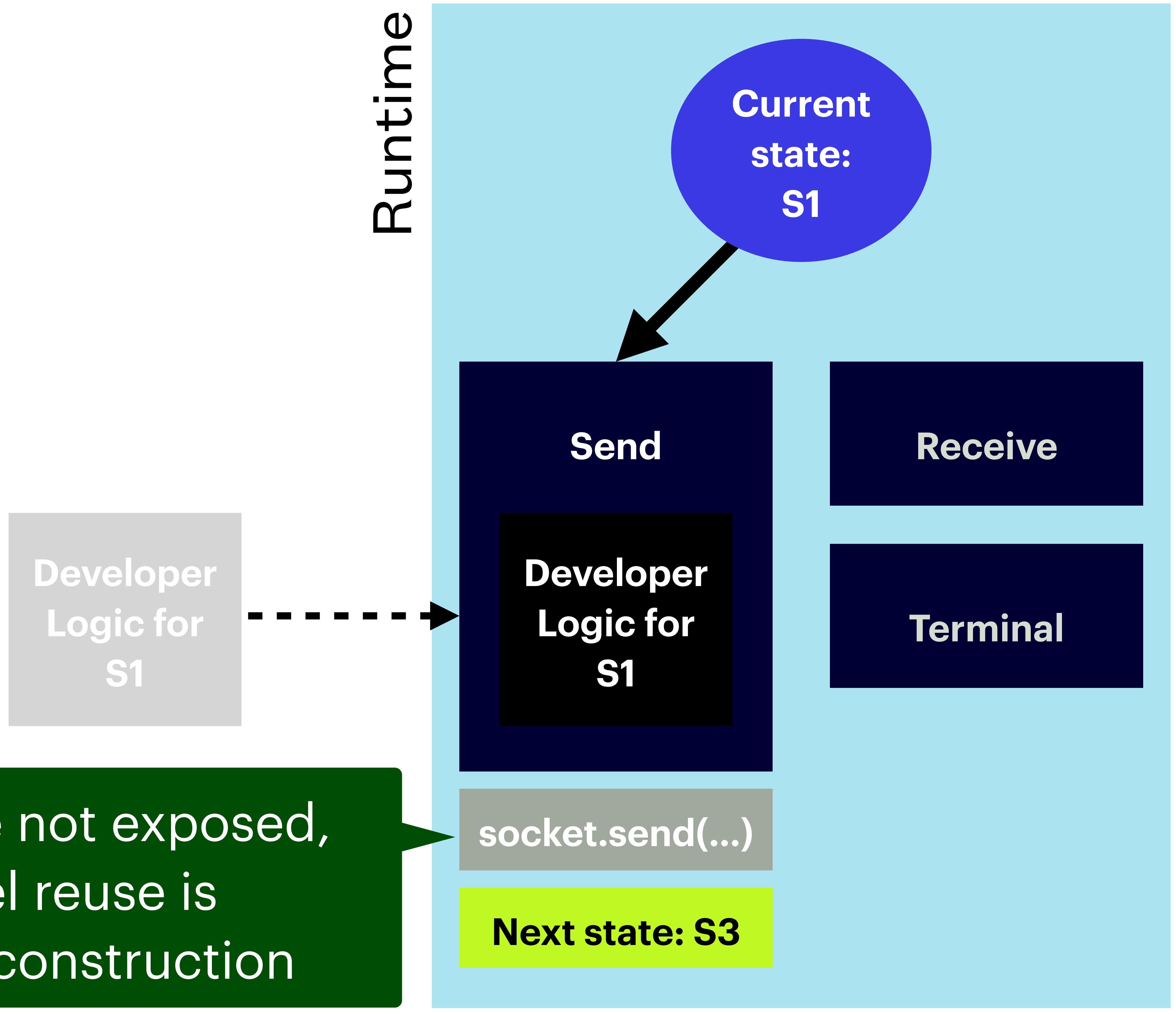
- We generate a session runtime to execute EFSM
  - Performs I/O action for current state
- We construct types for injecting business logic
  - What to send? How to handle receive?
- Developer instantiates session runtime with custom implementations



API Generation - Design Philosophy

Generate Correct-by-Construction APIs

- We generate a 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



Channel APIs are not exposed, thus channel reuse is **impossible** by construction

3️⃣ Callback-Style APIs for Static Linearity

API Generation for Node.js Endpoints

- Send = union type of **selections**
  - **Selection** = tuple of label, payload, successor state
- Receive = object literal of branches
  - Branch = callback named after the label
- We generate a factory object with overloads
  - Facilitate auto-completion in IDEs

```
const handleQuery = Session.Initial({
    Query: async (Next, destination) => {
      /* snip */
      if (response.status === "available") {
        return
          Next.Available([response.quote],
            /* snip */);
      } else {
        return
          Next.Full([], handleQuery);
      }
    },
});
```

3️⃣ Callback-Style APIs for Static Linearity

API Generation for Node.js Endpoints

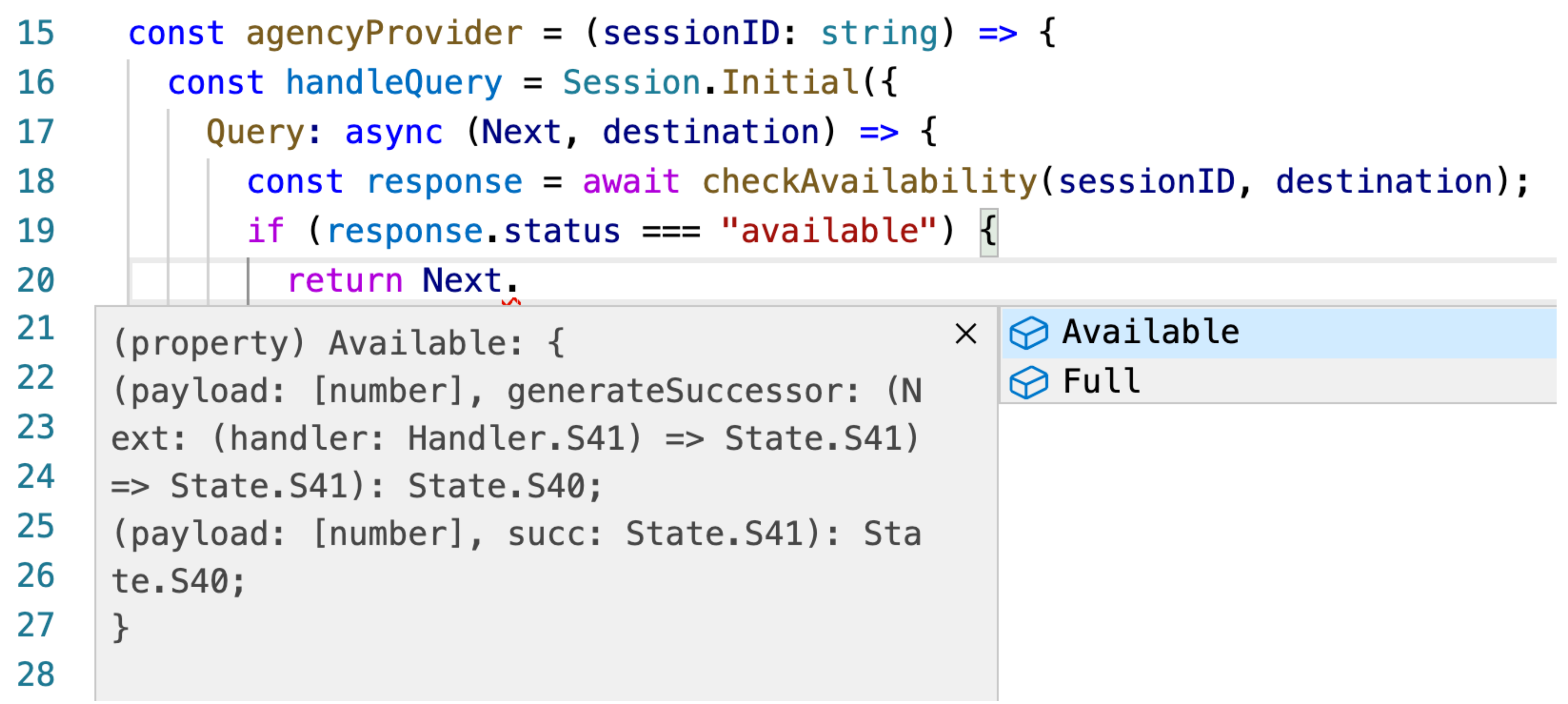
- Send = union type of selections
  - Selection = tuple of label, payload, successor state
- Receive = object literal of **branches**
  - **Branch** = callback named after the label
- We generate a factory object with overloads
  - Facilitate auto-completion in IDEs

```
/* snip */
return Next.Available([response.quote], Next => (
  Next({
    Confirm: async (End, credentials) => {
      // Handle confirmation
      await confirmBooking(sessionID, credentials);
      return End();
    },
    Reject: async (End) => {
      await release(sessionID);
      return End();
    },
  })
));
/* snip */
```

3️⃣ Callback-Style APIs for Static Linearity

API Generation for Node.js Endpoints

- Send = union type of selections
  - Selection = tuple of label, payload, successor state
- Receive = object literal of branches
  - Branch = callback named after the label
- We generate a factory object with overloads
  - Facilitate auto-completion in IDEs



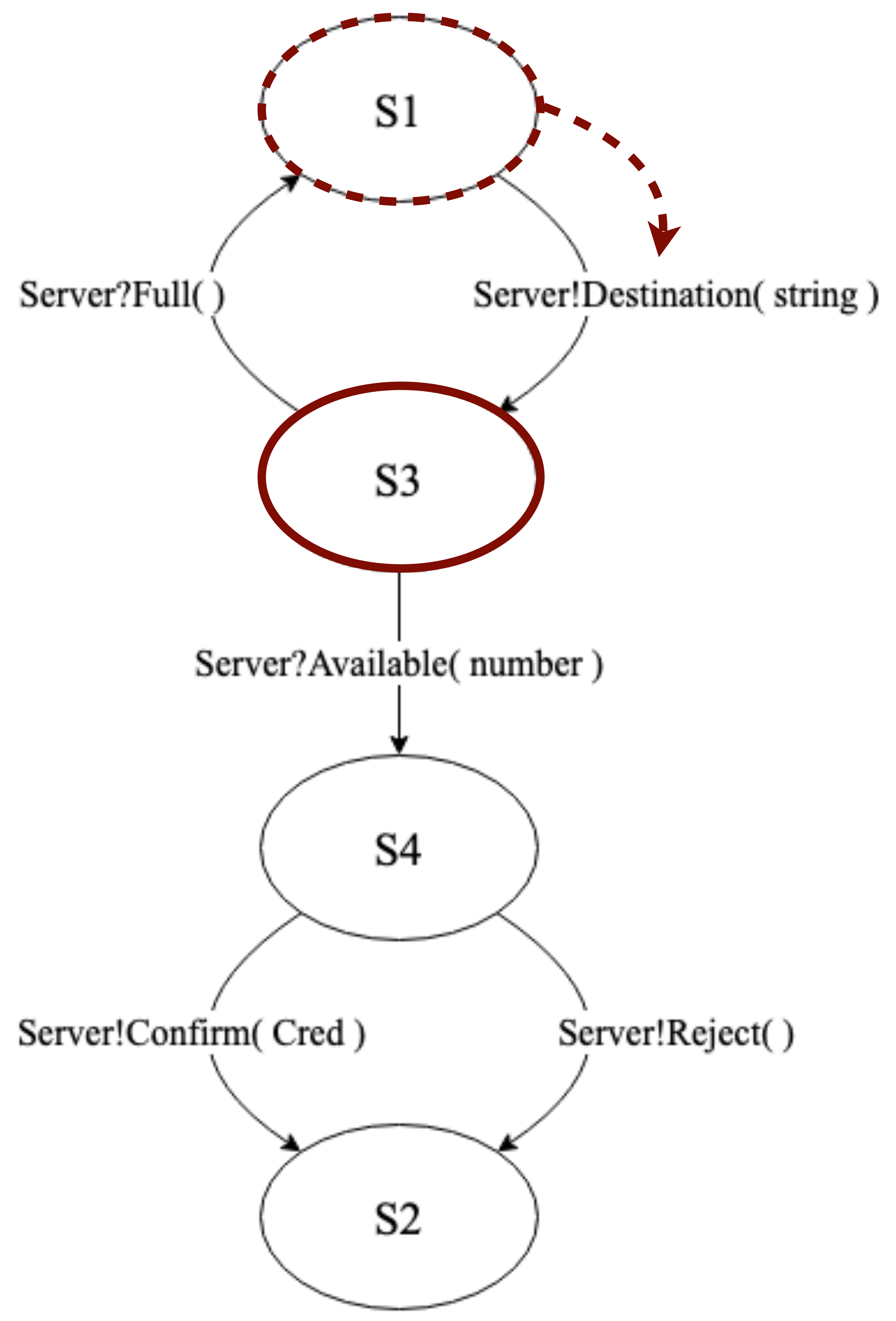
```
const agencyProvider = (sessionID: string) => {
  const handleQuery = Session.Initial({
    Query: async (Next, destination) => {
      const response = await checkAvailability(sessionID, destination);
      if (response.status === "available") {
        return Next.
```

```
(property) Available: {
(payload: [number], generateSuccessor: (N
ext: (handler: Handler.S41) => State.S41)
=> State.S41): State.S40;
(payload: [number], succ: State.S41): Sta
te.S40;
}
```

Challenge - 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.

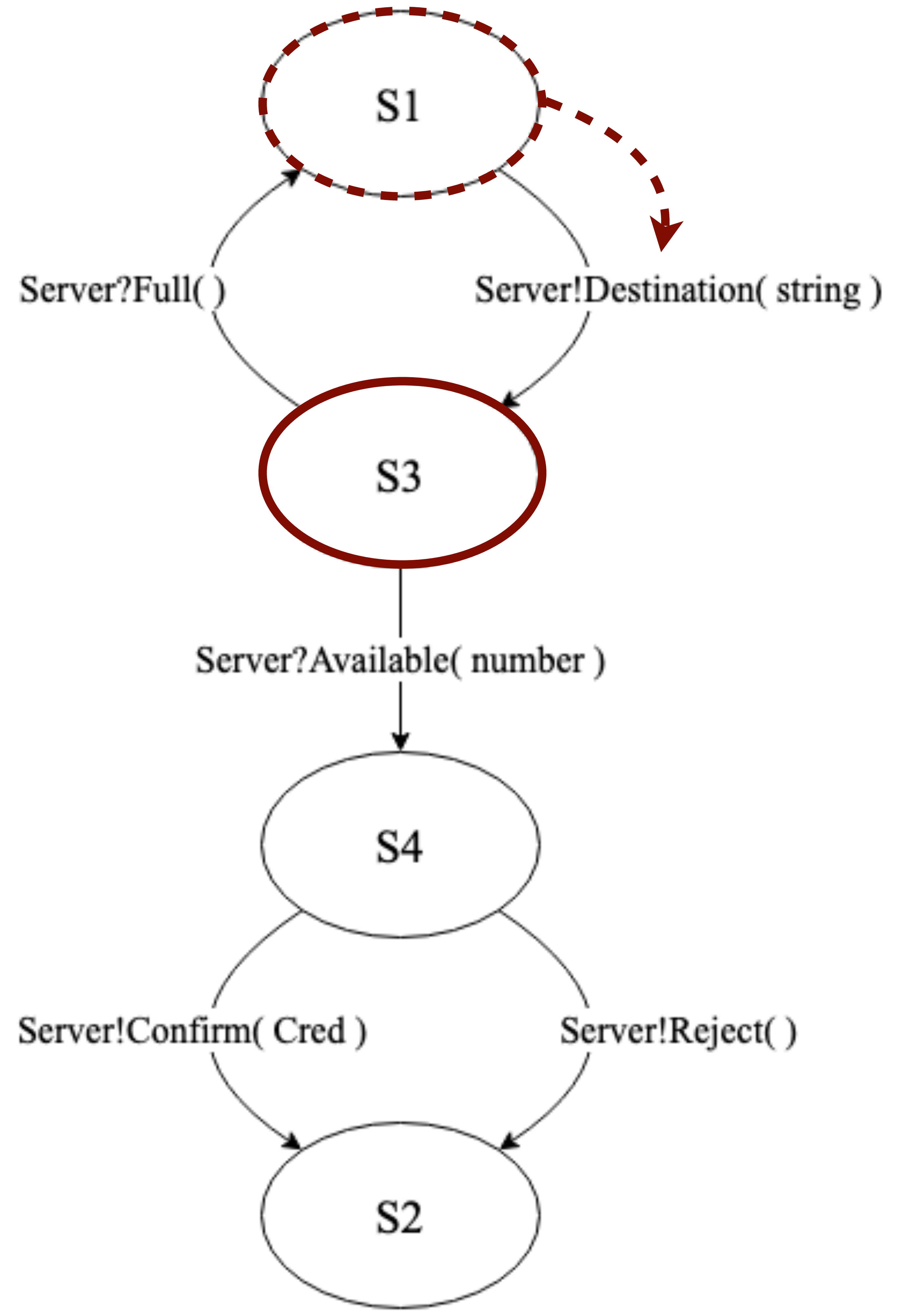
- How to guarantee that <u>users</u> respect channel linearity?



Challenge - 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.

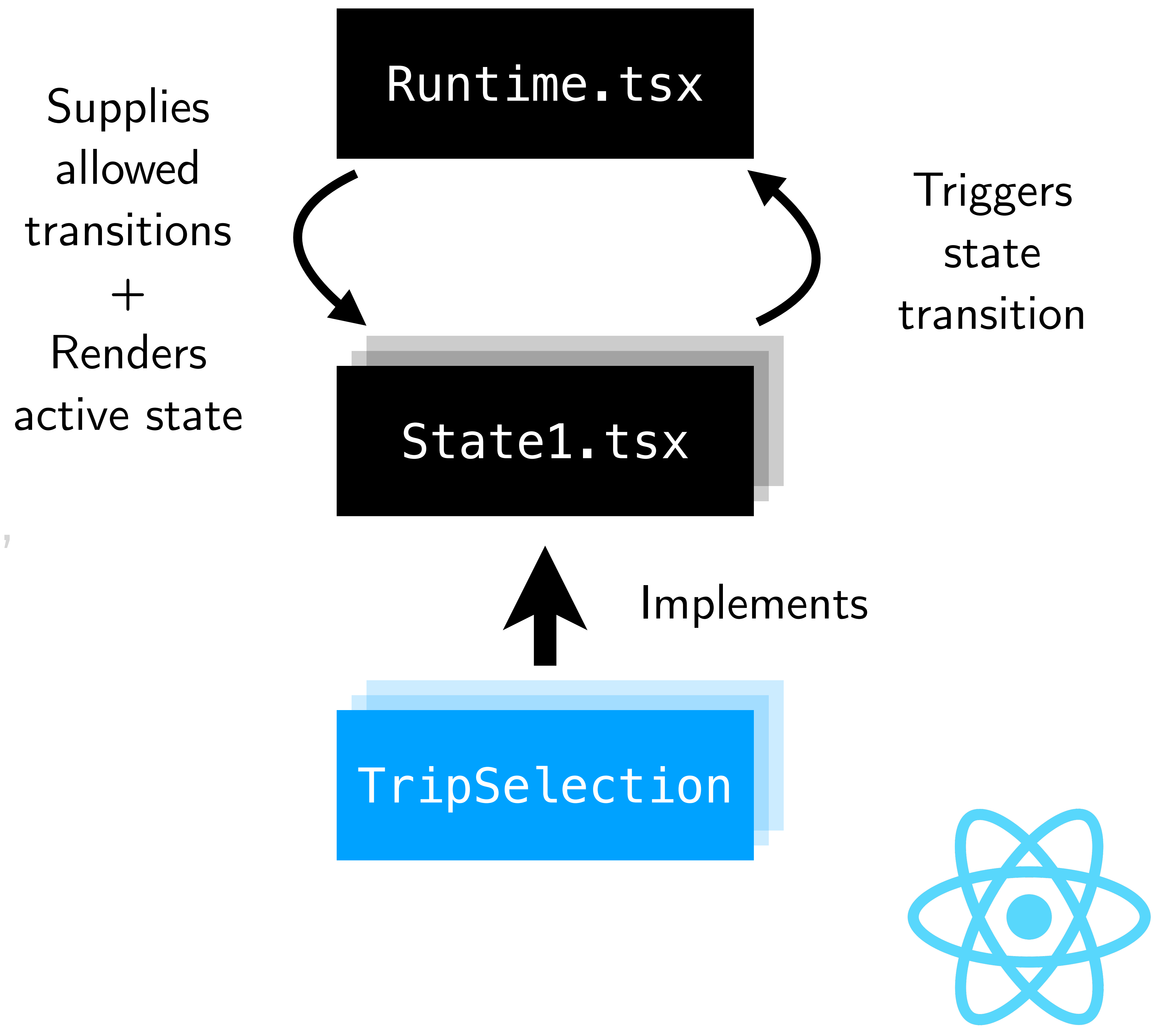
- **How to guarantee that <u>users</u> respect channel linearity?**



3️⃣ Safe, Interactive Channel Actions

API Generation for Browser Endpoints

- EFSM states = abstract React components
  - Developer inherits and overrides view function
- Runtime = React component
- Send = "component factories"
  - Generates a React component that, by construction, binds the permitted I/O action to a UI event
- Receive = named callbacks
  - Override abstract methods


Runtime.tsx
Supplies allowed transitions + Renders active state
Triggers state transition
State1.tsx
Implements
TripSelection

3️⃣ Safe, Interactive Channel Actions

API Generation for Browser Endpoints

- EFSM states = abstract React components
  - Developer inherits and overrides view function
- Runtime = React component
- Send = "component factories"
  - Generates a React component that, by construction, binds the permitted I/O action to an UI event
- Receive = named callbacks
  - Override abstract methods

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

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

3️⃣ Safe, Interactive Channel Actions

API Generation for Browser Endpoints

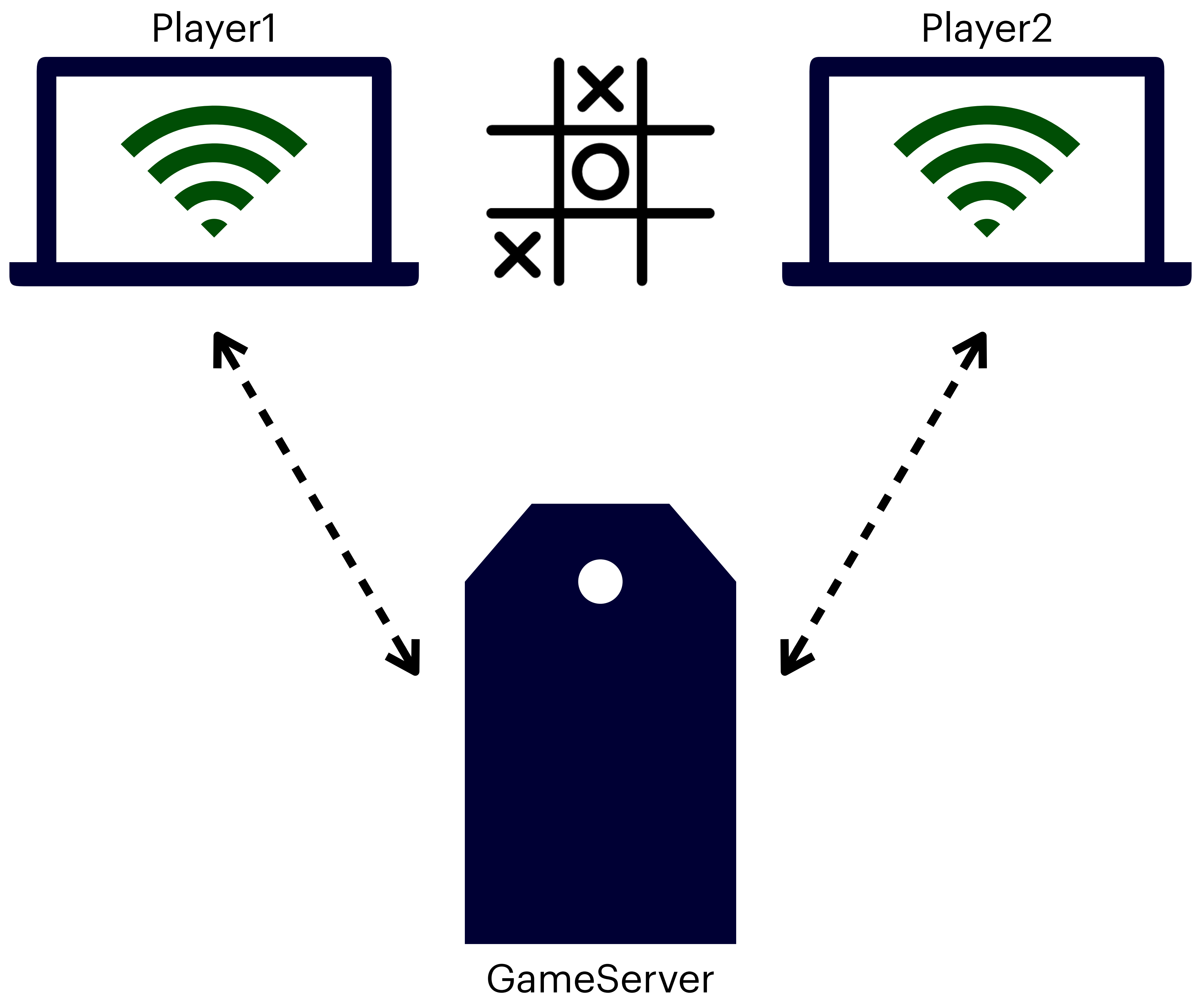
- EFSM states = abstract React components
  - Developer inherits and overrides view function
- Runtime = React component
- Send = "component factories"
  - Generates a React component that, by construction, binds the permitted I/O action to a UI event
- Receive = named callbacks
  - Override abstract methods

```
export default class Waiting extends S8 {

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

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

  // View function
  render() { /* snip */ }

}
```

3️⃣ Error Handling for Web Applications

Session Cancellation

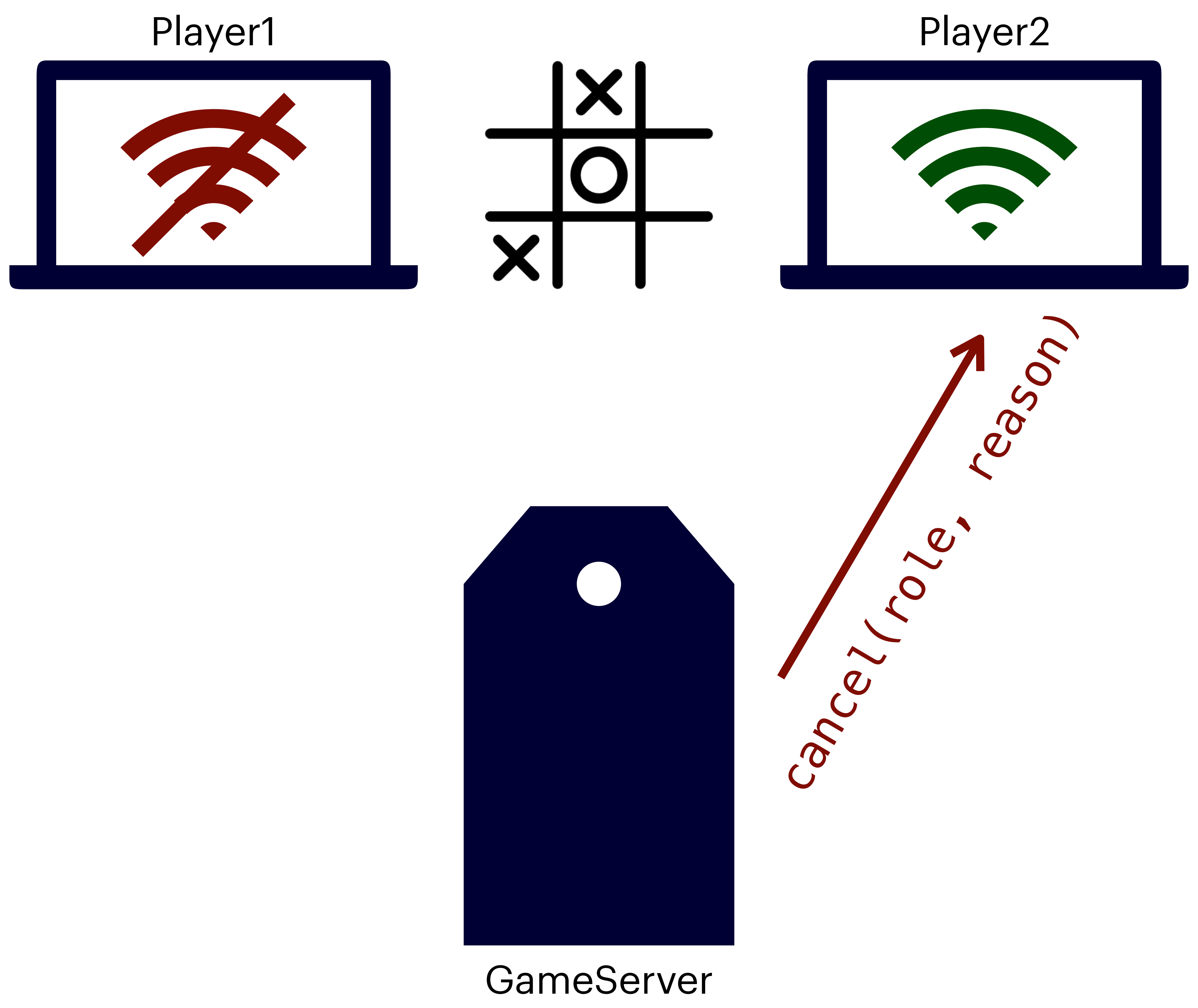
- Session cancellation is unavoidable - e.g. browser disconnects prematurely
- Server signals to other browser roles when a browser role disconnects
- We generate seams for developers to inject custom business logic
  - Server = callback function for cleanup
  - Browser = React component



3️⃣ Error Handling for Web Applications

Session Cancellation

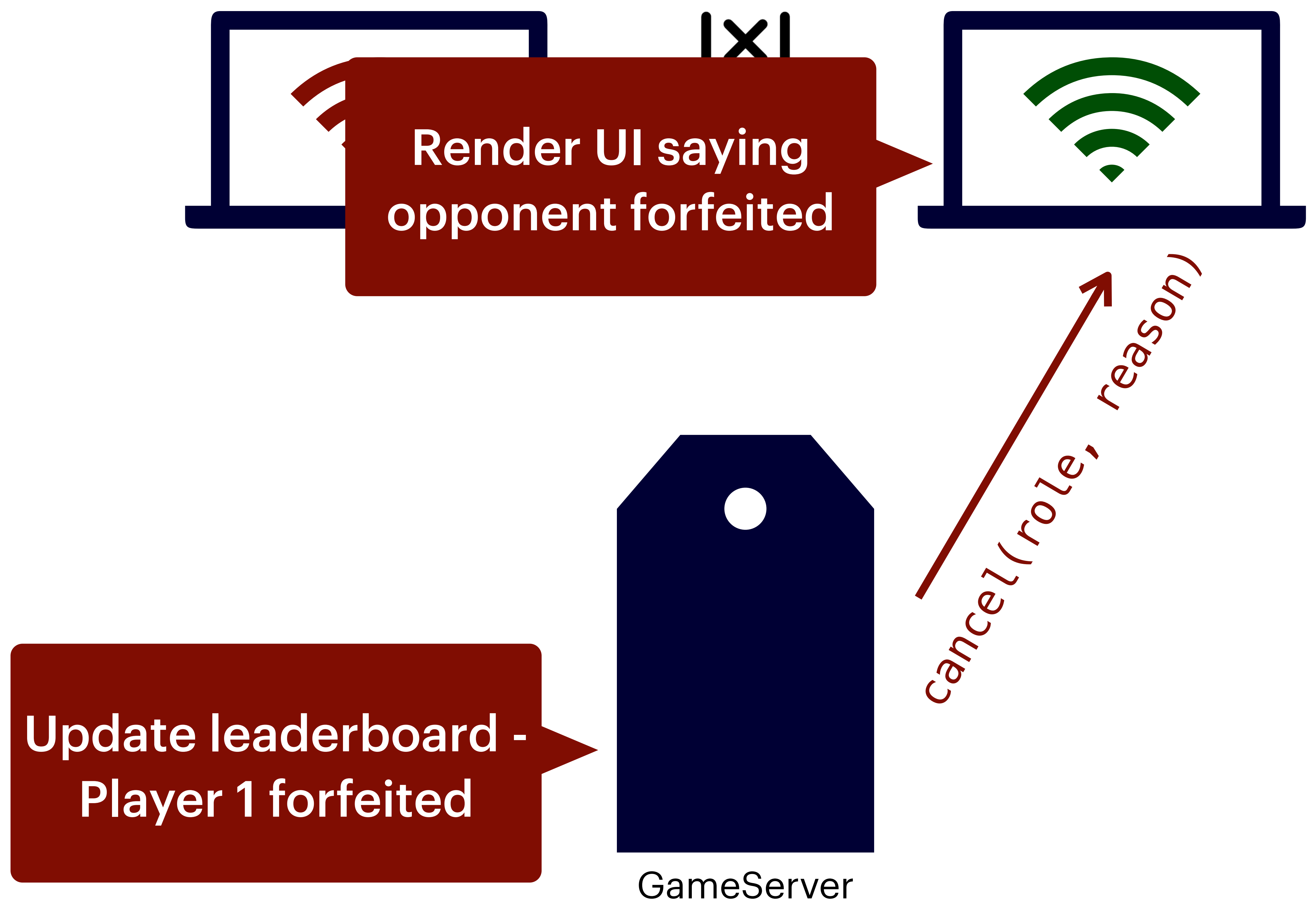
- Session cancellation is unavoidable - e.g. browser disconnects prematurely
- Server signals to other browser roles when a browser role disconnects
- We generate seams for developers to inject custom business logic
  - Server = callback function for cleanup
  - Browser = React component



3️⃣ Error Handling for Web Applications

Session Cancellation

- Session cancellation is unavoidable - e.g. browser disconnects prematurely
- Server signals to other browser roles when a browser role disconnects
- We generate seams for developers to inject custom business logic
    - Server = callback function for cleanup
    - Browser = React component


Render UI saying opponent forfeited
cancel(role, reason)
Update leaderboard - Player 1 forfeited
GameServer

Contributions

- *STScript* - a toolchain that generates TypeScript APIs that statically guarantee communication-safe web development

- *RouST* - a new session type theory that supports multiparty communications with routing mechanisms

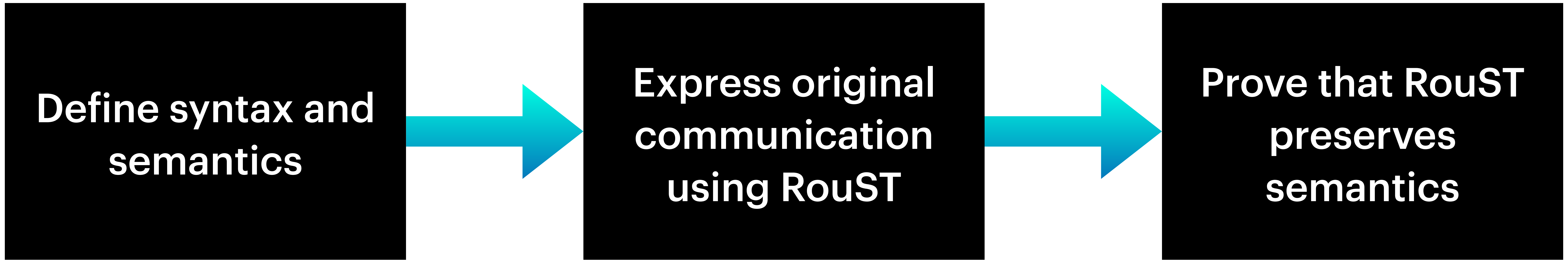
Contributions

- *STScript* - a toolchain that generates TypeScript APIs that statically guarantee communication-safe web development

- *RouST* - a new session type theory that supports multiparty communications with routing mechanisms

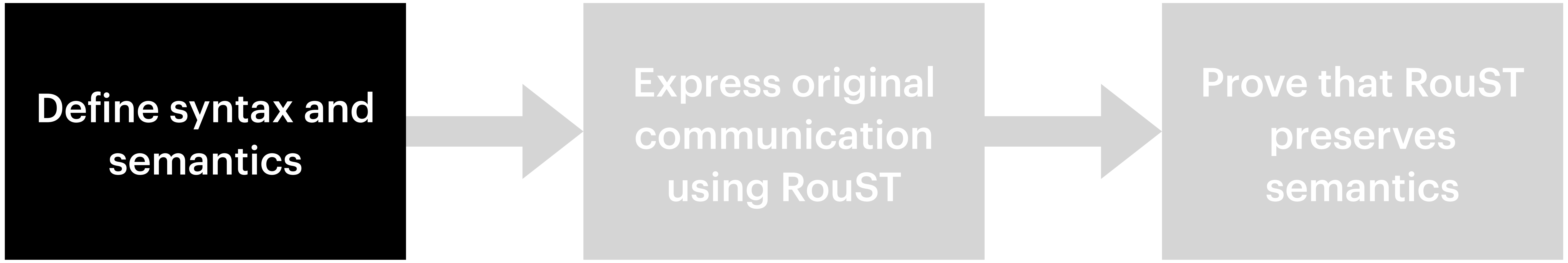
RouST

A Theory of Routed Multiparty Session Types


Define syntax and semantics
Express original communication using RouST
Prove that RouST preserves semantics

RouST

A Theory of Routed Multiparty Session Types


Define syntax and semantics
Express original communication using RouST
Prove that RouST preserves semantics

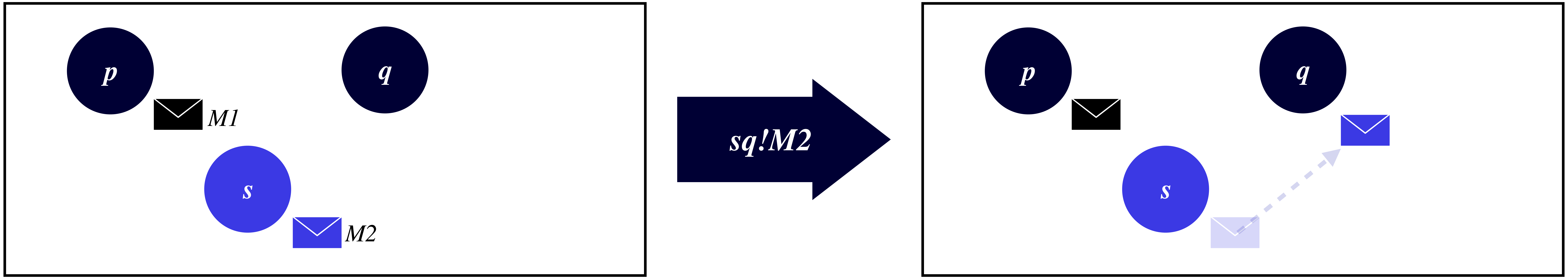
RouST - Syntax

G	$::=$		Global Types
	$\mid$	$\mathtt{end}$	Termination
	$\mid$	$\mathbf{t}$	Type Variable
	$\mid$	$\mu\mathbf{t}.G$	Recursive Type
	$\mid$	$\mathsf{p} \to \mathsf{q} : \{l_i : G_i\}_{i \in I}$	Direct Communication
	$\mid$	$\mathsf{p} \,{-}\mathsf{s}{\to}\, \mathsf{q} : \{l_i : G_i\}_{i \in I}$	Routed Communication

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

RouST - Semantics

Labelled Transition System (LTS)



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

RouST - Semantics

Labelled Transition System (LTS)

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

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

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

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

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

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

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

$$\frac{\forall i \in I.\ G_i \xrightarrow{l} G'_i \qquad \mathrm{subj}(l) \notin \{\mathsf{p}, \mathsf{q}\}}{\mathsf{p} - \mathsf{s} \to \mathsf{q} : \{l_i : G_i\}_{i \in I} \xrightarrow{l} \mathsf{p} - \mathsf{s} \to \mathsf{q} : \{l_i : G'_i\}_{i \in I}}\ [\textsc{Gr8}]$$

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

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

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

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

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

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

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

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

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

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

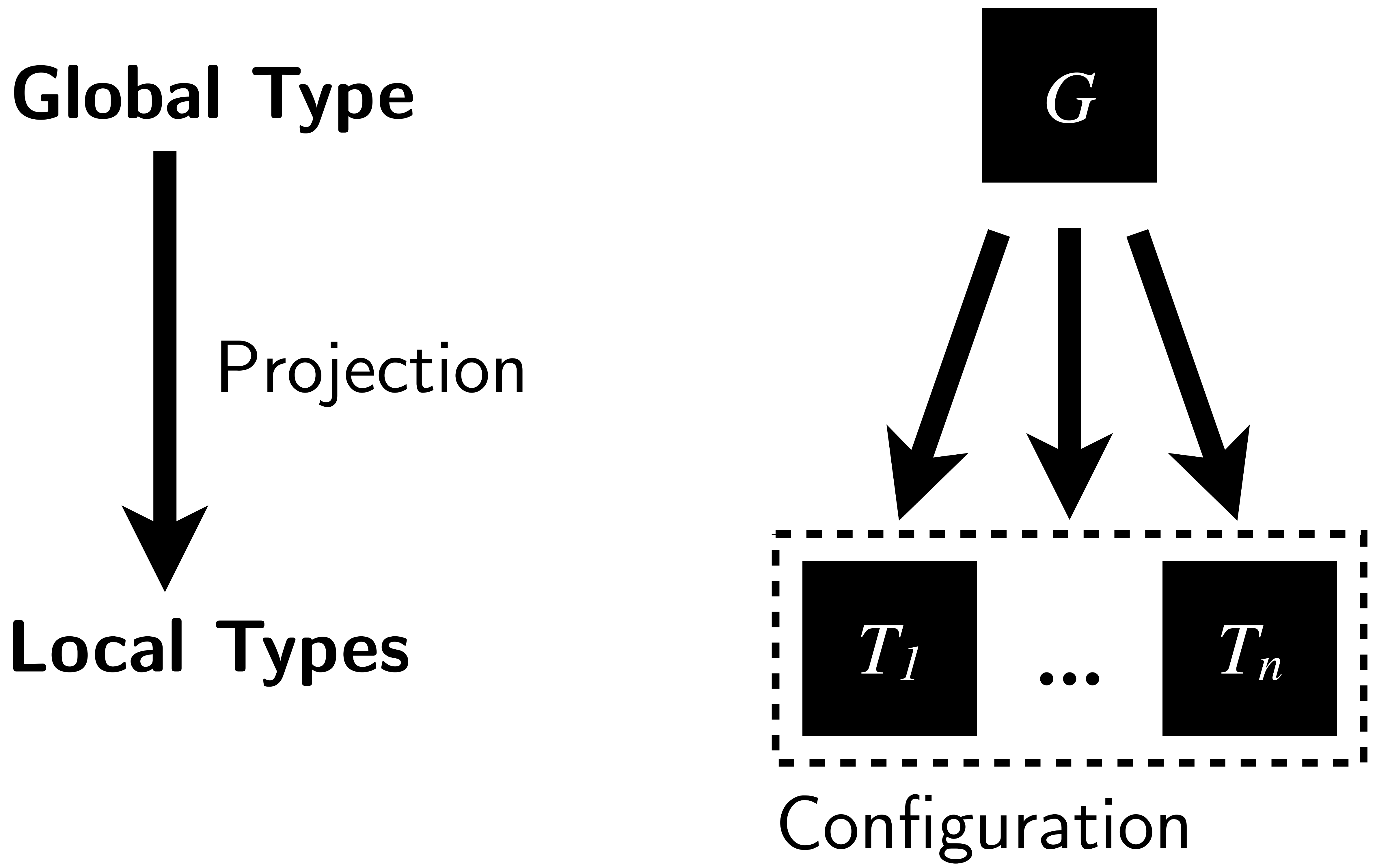
$$\frac{T_j \xrightarrow{l} T'_j \qquad \mathrm{subj}(l) \neq \mathsf{q} \qquad \forall i \in I \setminus \{j\}.\ T'_i = T_i}{\mathsf{p} \looparrowright \mathsf{q}.\ j : \{l_i : T_i\}_{i \in I} \xrightarrow{l} \mathsf{p} \looparrowright \mathsf{q}.\ j : \{l_i : T'_i\}_{i \in I}}\ [\textsc{Lr9}]$$

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

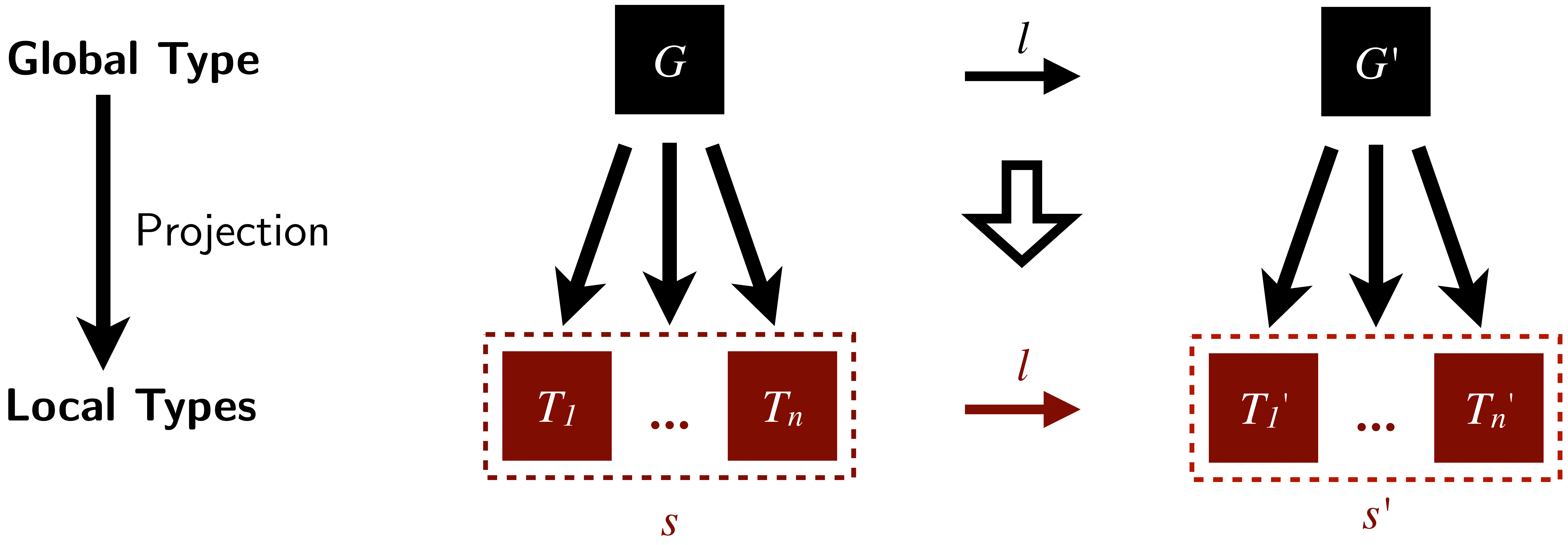
$$\frac{l = \mathtt{via}\langle \mathsf{s} \rangle (\cdot) \qquad \mathrm{subj}(l) \neq \mathsf{q} \qquad \forall i \in I.\ T_i \xrightarrow{l} T'_i}{\mathsf{q} \,\&\, \{l_i : T_i\}_{i \in I} \xrightarrow{l} \mathsf{q} \,\&\, \{l_i : T'_i\}_{i \in I}}\ [\textsc{Lr11}]$$

Soundness and Completeness

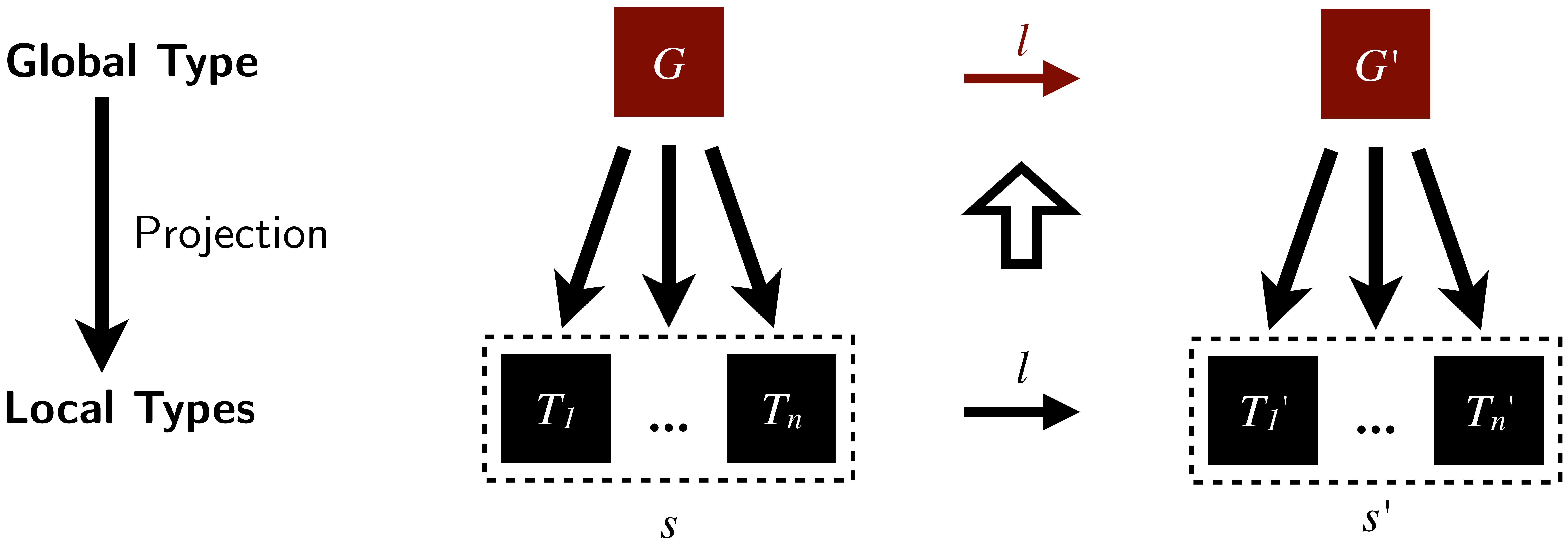
Projected Configurations of Global Types



Soundness and Completeness

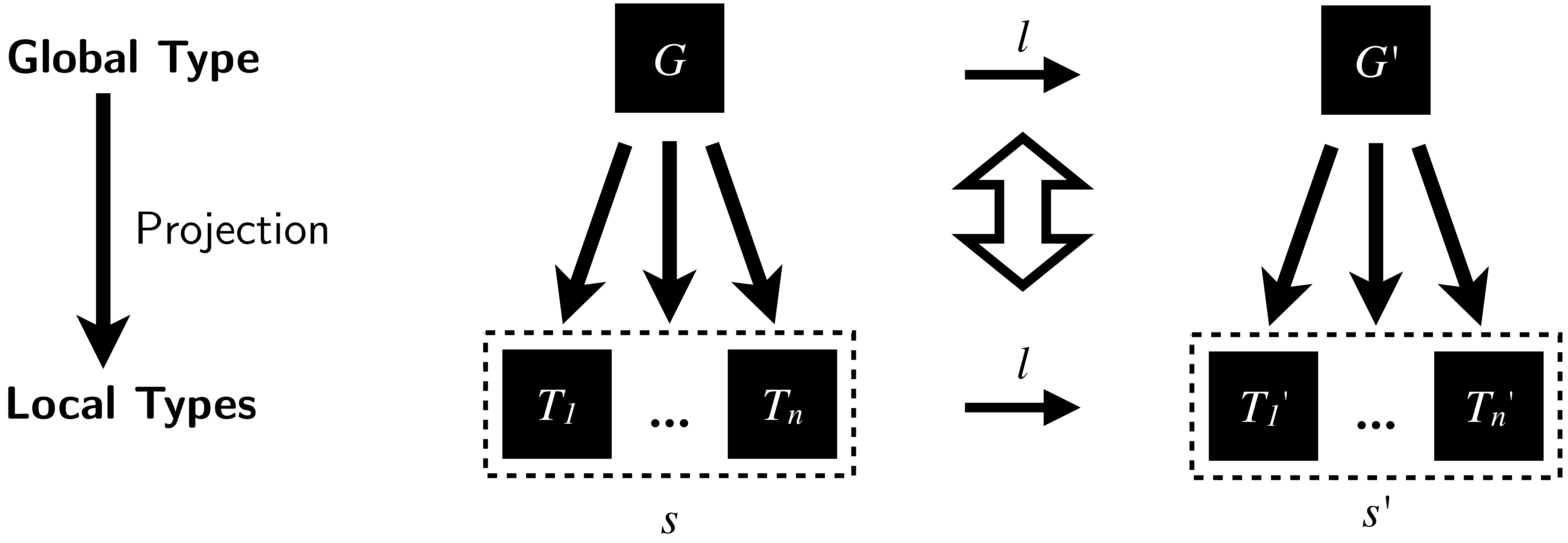


Soundness and Completeness



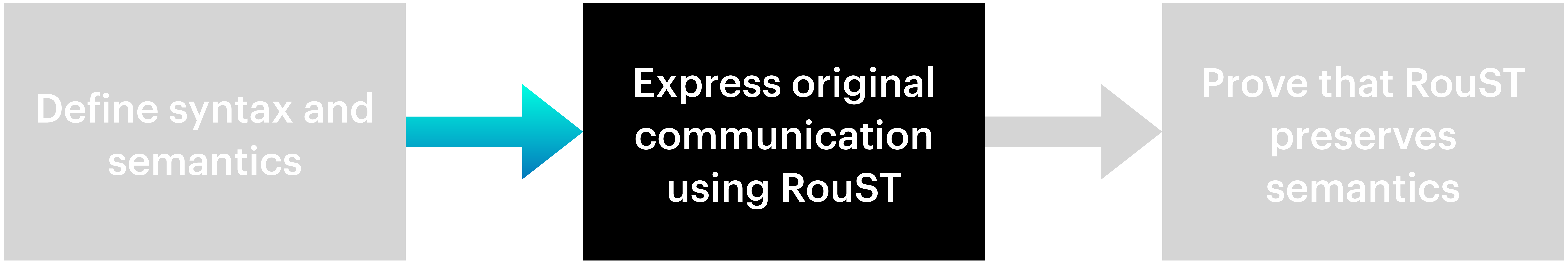
Soundness and Completeness

Theorem 4.6, see full paper for proof



RouST

A Theory of Routed Multiparty Session Types


Define syntax and semantics
Express original communication using RouST
Prove that RouST preserves semantics

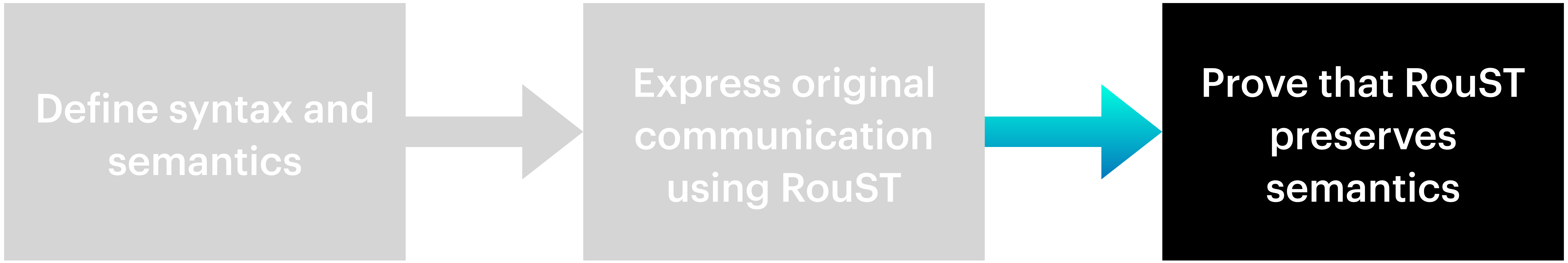
Towards *RouST*

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

```
Encoding :: MPST -> Role -> RouST
```

RouST

A Theory of Routed Multiparty Session Types


Define syntax and semantics
Express original communication using RouST
Prove that RouST preserves semantics

RouST - Preservation of Semantics

Theorem 4.12, see full paper for proof

$$\begin{tikzcd}
\text{MPST} \arrow[d, "\text{Encoding}"] & \boxed{G} \arrow[rr, "l"] \arrow[d] & {} \arrow[d, phantom, "\Updownarrow"] & \boxed{G'} \arrow[d] \\
\text{RouST} & \boxed{\llbracket G, s \rrbracket} \arrow[rr, "{\llbracket l, s \rrbracket}"] & {} & \boxed{\llbracket G', s \rrbracket}
\end{tikzcd}$$

Contributions

- *STScript* - a toolchain that generates TypeScript APIs that statically guarantee communication-safe web development

- *RouST* - a new session type theory that supports multiparty communications with routing mechanisms

Contributions

- *STScript* - a toolchain that generates TypeScript APIs that statically guarantee communication-safe web development

- *RouST* - a new session type theory that supports multiparty communications with routing mechanisms

Thank you!

Full paper available at

`https://arxiv.org/abs/2101.04622`