Web Development Based On Multiparty Session Types In PureScript

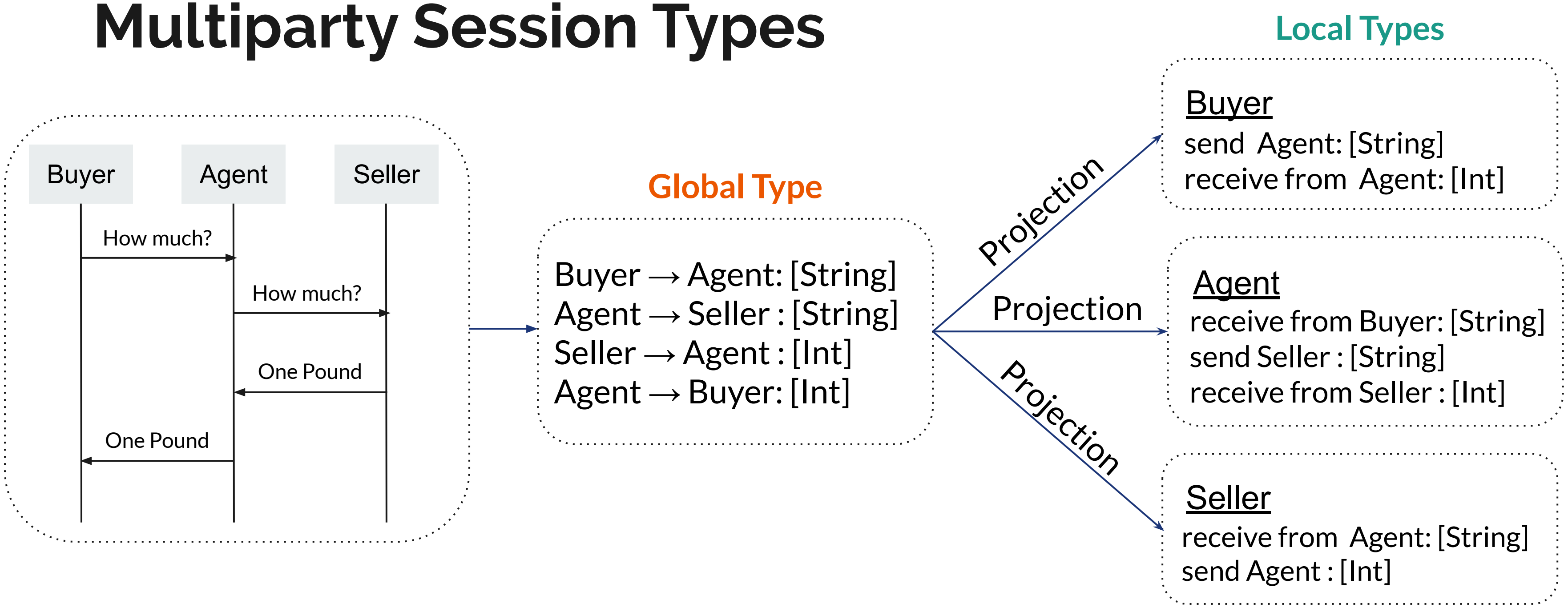
By Hei Yin Fong
Supervised by Nobuko Yoshida

Outline and Contributions of Project

- Extending Code Generation of Scribble-Purescript
- Improving Error Messages
- WebRTC Client-to-Client Session
- Benchmark of PureScript Session Runtime Library

Background

Multiparty Session Types



Global Type

Buyer → Agent: [String]
Agent → Seller : [String]
Seller → Agent : [Int]
Agent → Buyer: [Int]

Local Types

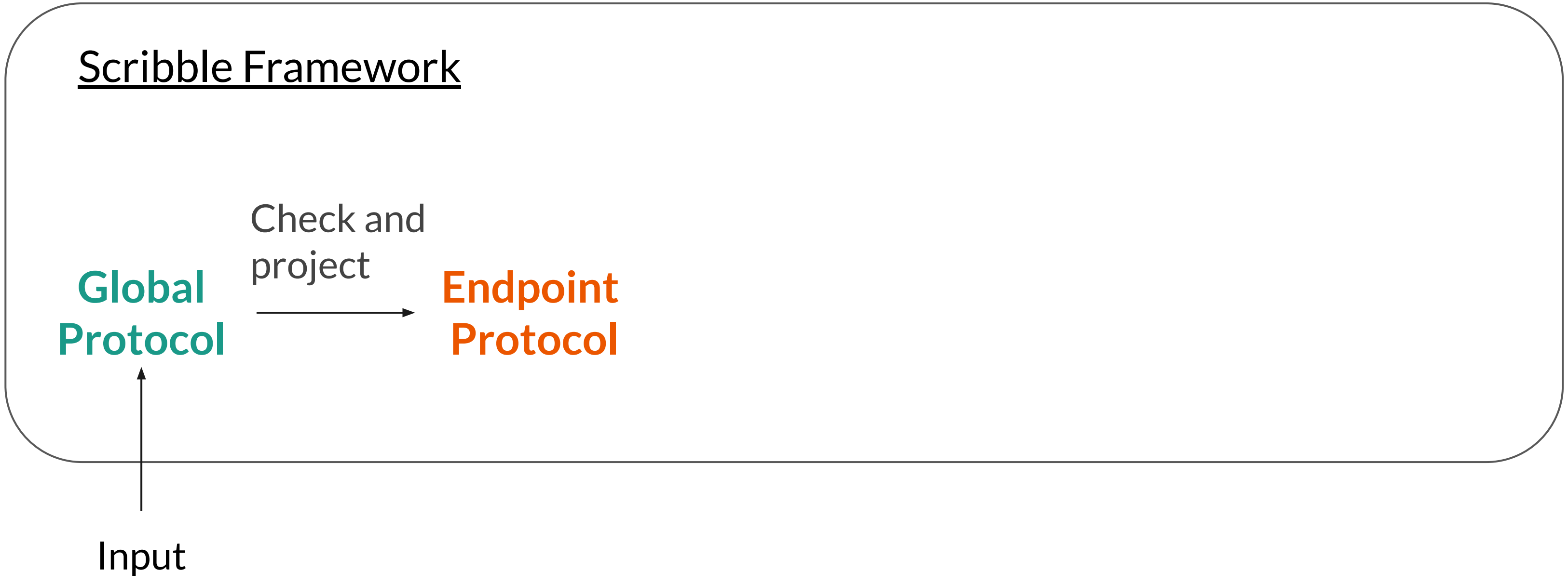
Buyer
send Agent: [String]
receive from Agent: [Int]

Agent
receive from Buyer: [String]
send Seller : [String]
receive from Seller : [Int]

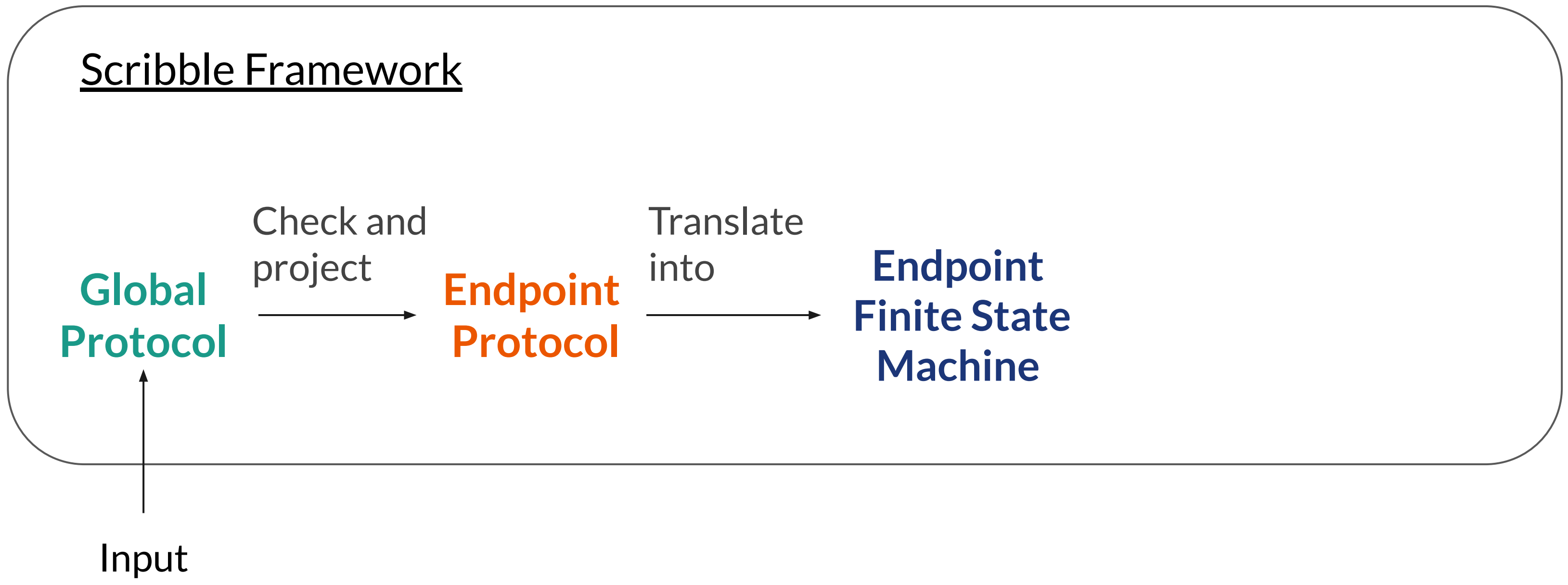
Seller
receive from Agent: [String]
send Agent : [Int]

Scribble Code Generation

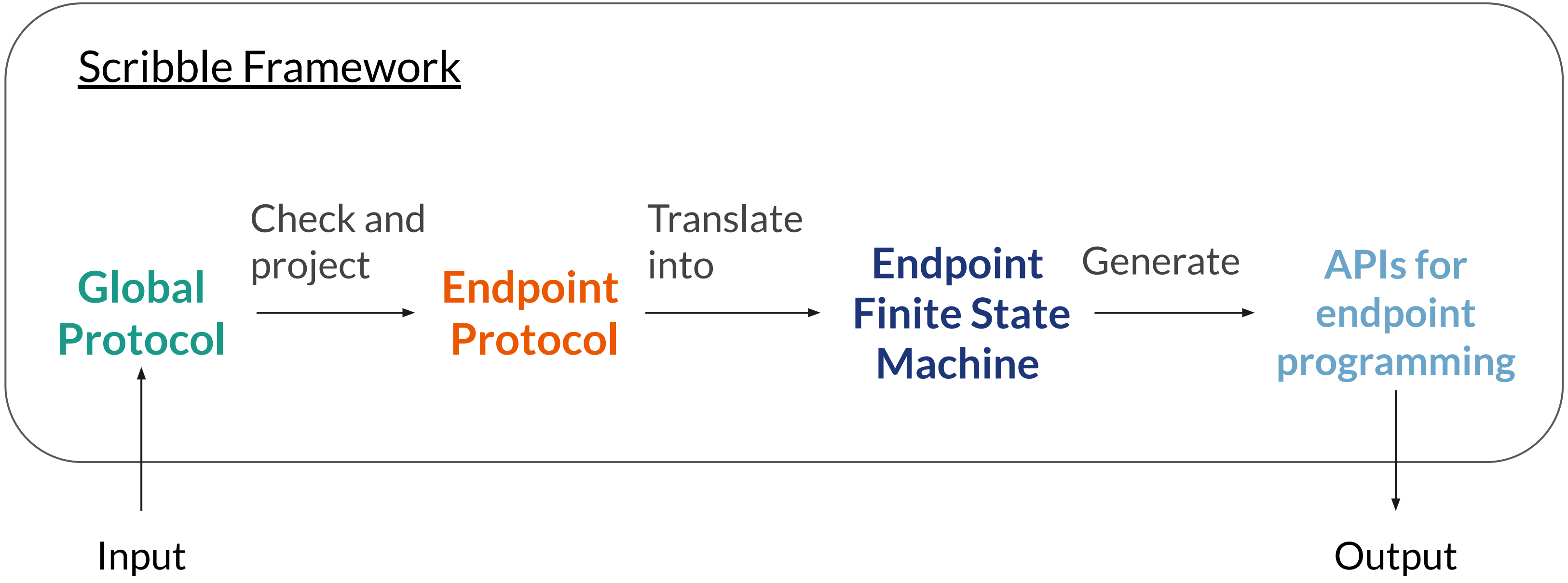
Scribble Code Generation



Scribble Code Generation



Scribble Code Generation



PureScript

- A **functional** language that compiles to **readable Javascript**
- Similar syntax as Haskell
- Extensive collection of libraries for the development of web applications, web servers and so on.

PureScript Code Generation

PureScript types are generated from the EFSMs:

- Each **state** is a type
- Each **transition** is a type class instance

Purescript Code Generation

Purescript types are generated from the EFSMs:

- Each **state** is a type
- Each **transition** is a type class instance

Send: message send
Receive: message receive
Select: label selection
Branch: label branching

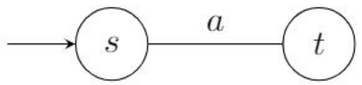
```
class Send r s t a
class Receive r s t a
class Select r s (ts :: RowList)
class Branch r r’ s (ts :: RowList)
```

EFSM transition as type classes

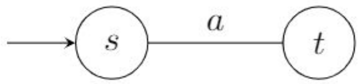
Send and **Receive** are type classes parameterised by r, s, t and a.

- r → role of the recipient/sender
- s → current state
- t → successor state
- a → message payload type

```
class Send r s t a | s ⇝ t r a
```



```
class Receive r s t a | s ⇝ t r a
```

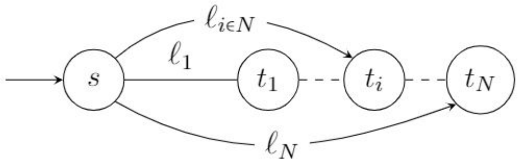


EFSM transition as type classes

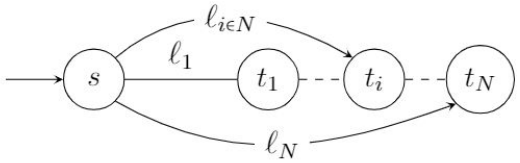
Select and **Branch** are type classes parameterised by r’, r, s and ts.

- s ⟶ current state
- ts ⟶ a collection of possible successor states

```
class Select r s! (ts :: RowList) |s! ⇝ ts r
```

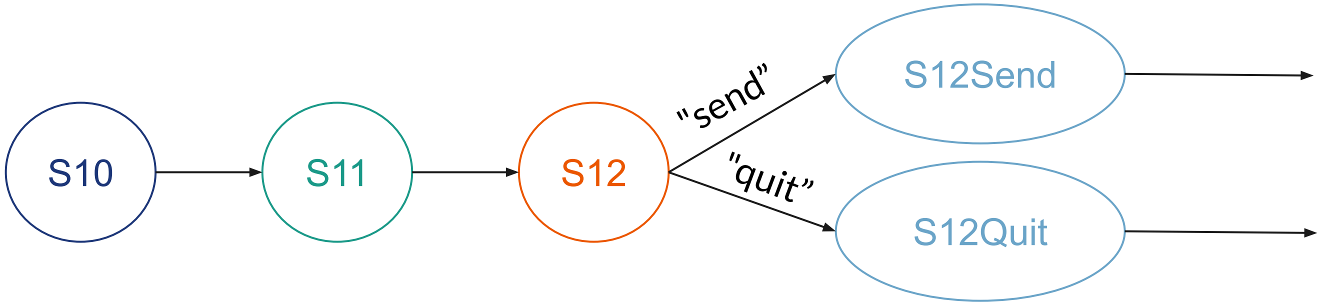


```
class Branch r  r s  (ts :: RowList) |s ⇝ ts r  r
```



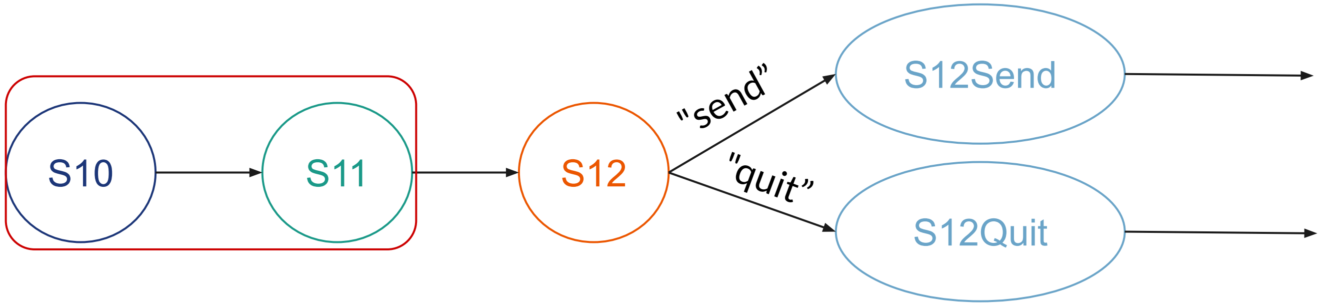
Purescript Generated APIs

```
instance sendS11 :: Send Server S10 S11 String
instance receiveS12 :: Receive Server S11 S12 String
instance selectS12 :: Select Server S12 (“send” ::
S12Send, “quit” :: S12Quit)
instance sendS12 :: Send Server S12Send S12 String
instance quitS12 :: Disconnect Server Client S12Quit S13
```



Purescript Generated APIs

```
instance sendS11 :: Send Server S10 S11 String
instance receiveS12 :: Receive Server S11 S12 String
instance selectS12 :: Select Server S12 (“send” ::
S12Send, “quit” :: S12Quit)
instance sendS12 :: Send Server S12Send S12 String
instance quitS12 :: Disconnect Server Client S12Quit S13
```



Purescript Session Runtime Library

- Uses the transition type classes as **type constraints** for functions connect, send, receive and etc.

```
send :: forall r rn c a s t m p.
   Send r s t a
=> RoleName r rn
=> EncodeJson a
…
=> a -> Session m c s t Unit
```

```
receive :: forall r rn c a s t m p.
   Receive r s t a
=> RoleName r rn
=> EncodeJson a
…
=> Session m c s t Unit
```

- **Statically checks** against the generated APIs
- Compilation success = protocol conformance

Improving Error Messages

Protocol Violation Example

```
global protocol Silly (role A,
role B)
{
  Connect() connect A to B;
  disconnect A and B;
}
```

Protocol Violation Example

```
global protocol Silly (role A,
role B)
{
  Connect() connect A to B;
  disconnect A and B;
}
```

```
silly = session
  (Proxy :: Proxy WebSockets)
  (Role :: Role A) do $
     connect (Role :: Role B) …
     send Connect
     disconnect (Role :: Role B)
```

Protocol Violation Example

```
global protocol Silly (role A,
role B)
{
  Connect() connect A to B;
  disconnect A and B;
}
```

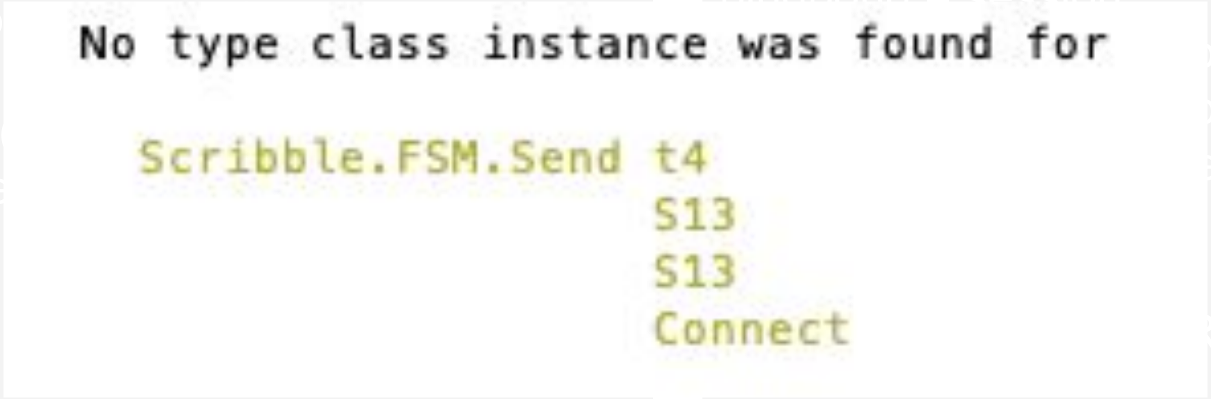
```
silly = session
  (Proxy :: Proxy WebSockets)
  (Role :: Role A) do $
     connect (Role :: Role B) …
     send Connect
     send Connect
     disconnect (Role :: Role B)
```

Protocol is **Violated**

Protocol Violation Example

```
global pro
{
  Connect(
  disconne
}
```

```
pingpong = session
ckets)
) $
e Server) ...

Role Server)
```



```
No type class instance was found for

   Scribble.FSM.Send t4
                     S13
                     S13
                     Connect
```

Protocol is **Violated**

No Type Class Instance Is Found Error

```
instance initialS10 :: Initial A S11
instance connect :: Connect A B S11 S12
instance sendConnect :: Send B S12 S13 Connect
Instance disconnectS13 :: Disconnect B S13 S14
instance terminalS14 :: Terminal A S14
```

No Type Class Instance Is Found Error

```
instance initialS10 :: Initial A S11
instance connect :: Connect A B S11 S12
instance sendConnect :: Send B S12 S13 Connect
Instance disconnectS13 :: Disconnect B S13 S14
instance terminalS14 :: Terminal A S14
```

```
instance initialS10 :: Send B S13 <AnyState> Connect
```

Such instance is not found

Problem

- Information provided is **obscure**
- As complexity of a protocol increases → such error message becomes **a pain for debugging**

Problem

- Information provided is **obscure**
- As complexity of a protocol increases → such error message becomes **a pain for debugging**
- Therefore, we need to improve error messages to increase the usability of the library

Solution: Improve Error Messages with Custom Error

- Purescript provides support for creating custom type errors via the module `Prim.TypeError`

Solution: Improve Error Messages with Custom Error

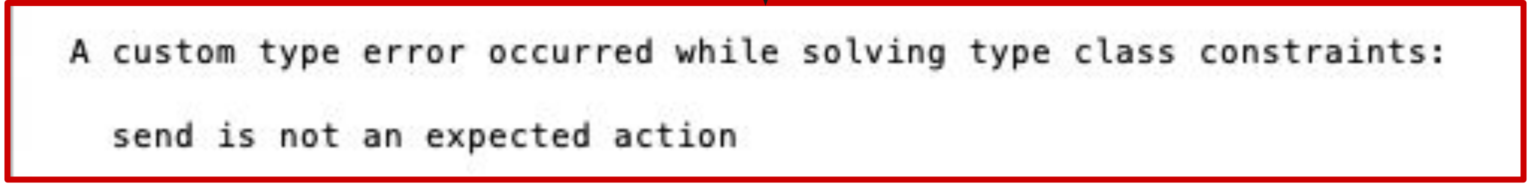
- Purescript provides support for creating custom type errors via the module `Prim.TypeError`
- Contains a `Fail` type class that embeds custom type errors

```
instance sendFail :: Fail (Text “send is not an expected
action”) => Send Server S13 <AnyState> Connect
```

Solution: Improve Error Messages with Custom Error

- Purescript provides support for creating custom type errors via the module `Prim.TypeError`
- Contains a `Fail` type class that embeds custom type errors

```
instance sendFail :: Fail (Text “send is not an expected
action”) => Send Server S13 <AnyState> Connect
```



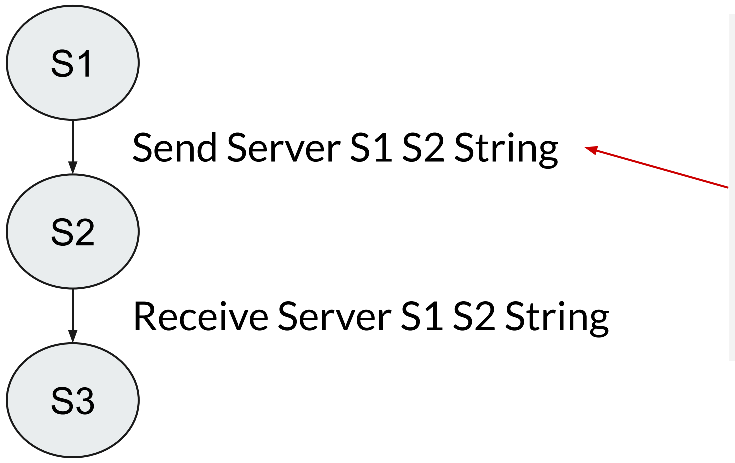
```
A custom type error occurred while solving type class constraints:

  send is not an expected action
```

Generating Failing Instances for Error Message

- Generate **all possible** ***incorrect*** **instances** inserted with custom type errors

Generating Failing Instances for Error Message

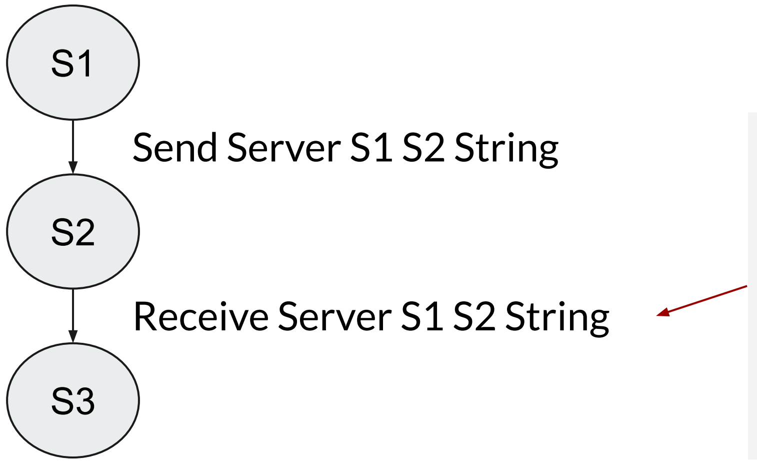
- Generate **all possible *incorrect* instances** inserted with custom type errors
- Method: Traversal of EFSM



```
instance .. :: Receive Server S1 S2 String
instance .. :: Select Server S1 ()
instance .. :: Branch Client Server S1 ()
instance .. :: Connect Client Server S1 S2
instance .. :: Disconnect Client Server S1 S2
instance .. :: Accept Client Server S1 S2
instance .. :: Terminal Client S1
```

Generating Failing Instances for Error Message

- Generate **all possible *incorrect* instances** inserted with custom type errors
- Method: Traversal of EFSM



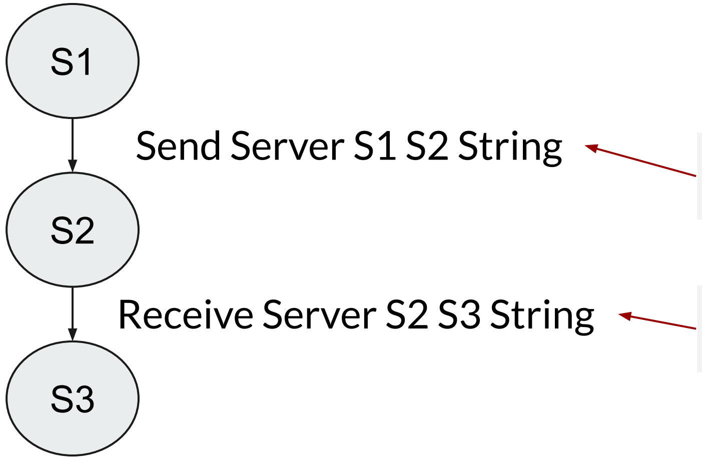
```
instance .. :: Send Server S2 S3 String
instance .. :: Select Server S2 ()
instance .. :: Branch Client Server S2 ()
instance .. :: Connect Client Server S2 S3
instance .. :: Disconnect Client Server S2 S3
instance .. :: Accept Client Server S2 S3
instance .. :: Terminal Client S2
```

Generating Failing Instances for Error Message

- Create incorrect instance for **incorrect type of message payload** but **correct action** **(type class)**

Generating Failing Instances for Error Message

- Create incorrect instance for **incorrect type of message payload** but **correct action** (type class)
- Instance Alternation



```
else instance .. :: Fail (..) => Send Server S1 S2 Int
```

```
else instance .. :: Fail (..) => Receive Server S1 S2 Int
```

Generating Custom Type Error Message

- Custom Type Error Message Structure:

```
Actual: <Type of Action> [<Type of Message Payload>]
Expected: <Type of Action> [<Type of Message Payload>]
```

For All Types of Incorrect Instances

For **Receive** and **Send** only

Generating Custom Type Error Message

- Combined with **parameterisation of message payload type**

```
instance sendCorrect :: Send Server S1 S2 String

else instance sendFail :: Fail(Above(Beside(Text
“Actual: Send message of type ”) (Quote b))
(Beside(Text “Expected: Send message of type
String”))) => Send Server S1 S2 b
```

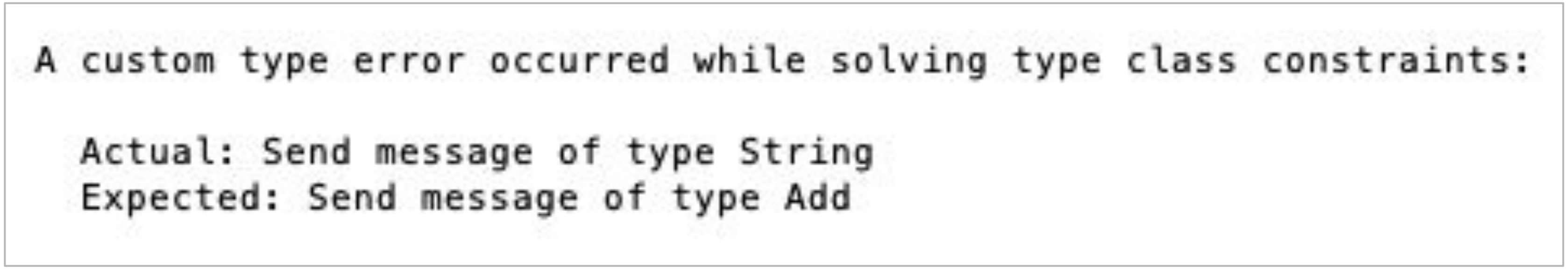
Generating Custom Type Error Message

- Combined with parameterisation of message payload type

```
instance sendCorrect :: Send Server S1 S2 String

else instance sendFail :: Fail(Above(Beside(Text
“Actual: Send message of type ”) (Quote b))
(Beside(Text “Expected: Send message of type
String”))) => Send Server S1 S2 b
```

Example Custom Type Error Message

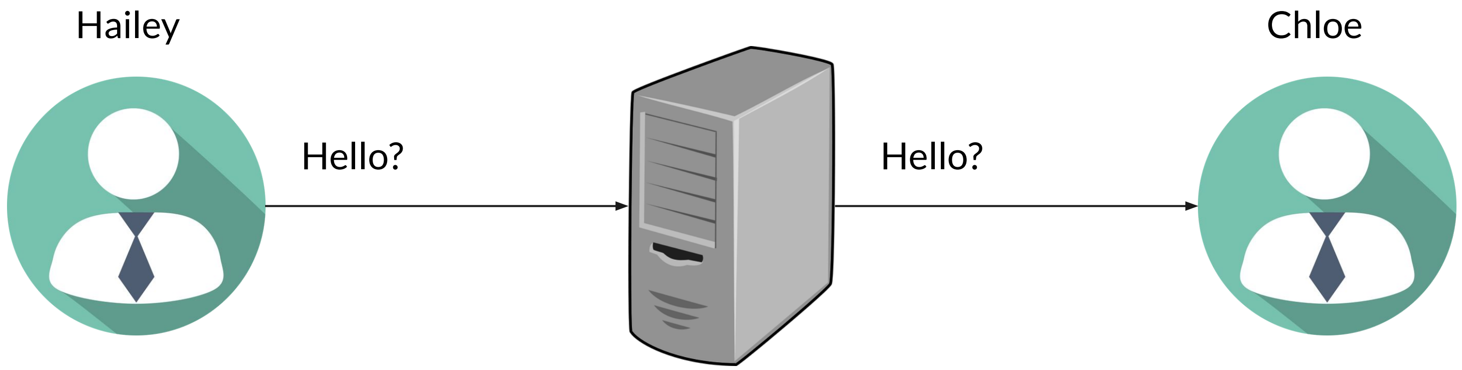
- Message Payload Type Mismatch



```
A custom type error occurred while solving type class constraints:

  Actual: Send message of type String
  Expected: Send message of type Add
```

Direct Client-to-Client Session

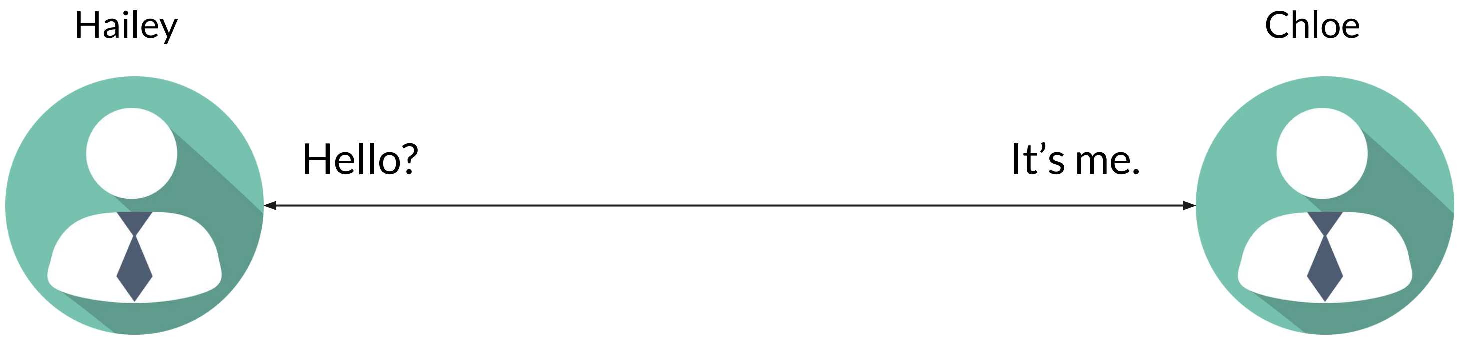
Client-to-Server Sessions



```
session
  (Proxy :: Proxy WebSockets)
  (Role :: Role Client) do $
    connect (Role :: Role Server) .....
```

```
session
  (Proxy :: Proxy WebSockets)
  (Role :: Role Client) do $
    connect (Role :: Role Server) .....
```

Direct Client-to-Client Session



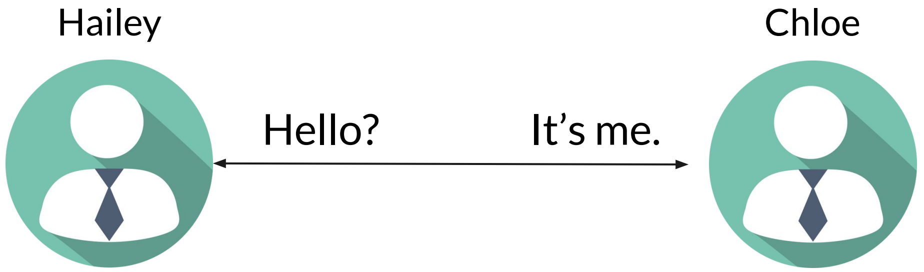
```
session
  (Proxy :: Proxy WebRTCConnection)
  (Role :: Role Client) do $
     connect (Role :: Role RemoteClient)
…..
```

Direct Client-to-Client Session

Hailey
Chloe
Hello?
It’s me.

```
session
  (Proxy :: Proxy WebRTCConnection)
  (Role :: Role Client) do $
    connect (Role :: Role RemoteClient)
.....
```

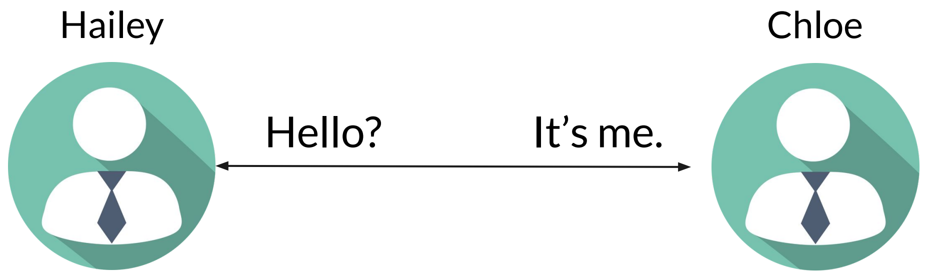
Direct Client-to-Client Session



✓ Specify Scribble Client-to-Client Protocol

```
session
  (Proxy :: Proxy WebRTCConnection)
  (Role :: Role Client) do $
    connect (Role :: Role RemoteClient)
…..
```

Direct Client-to-Client Session



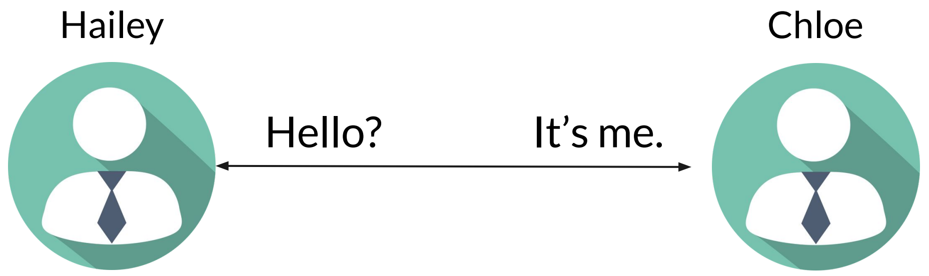
```
session
  (Proxy :: Proxy WebRTCConnection)
  (Role :: Role Client) do $
    connect (Role :: Role RemoteClient)
.....
```

✓ Specify Scribble Client-to-Client Protocol

↓

✓ Construct Client-to-Client Session using Purescript Session Runtime Library

Direct Client-to-Client Session



```
session
  (Proxy :: Proxy WebRTCConnection)
  (Role :: Role Client) do $
    connect (Role :: Role RemoteClient)
.....
```

✓ Specify Scribble Client-to-Client Protocol

↓

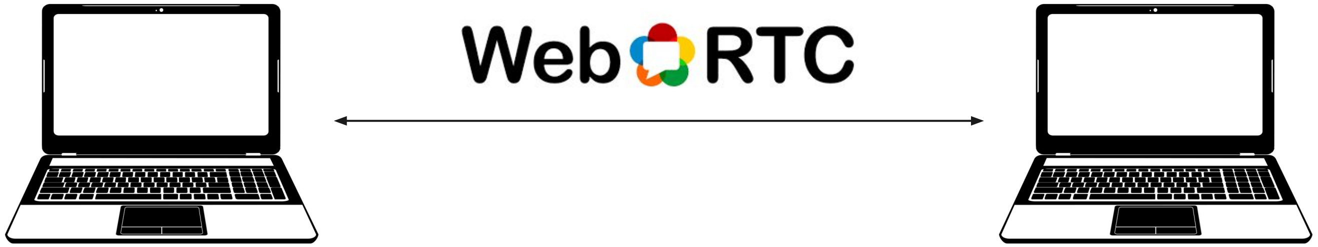
✓ Construct Client-to-Client Session using Purescript Session Runtime Library

↓

✓ Build Real-time Client-to-Client Application in Purescript

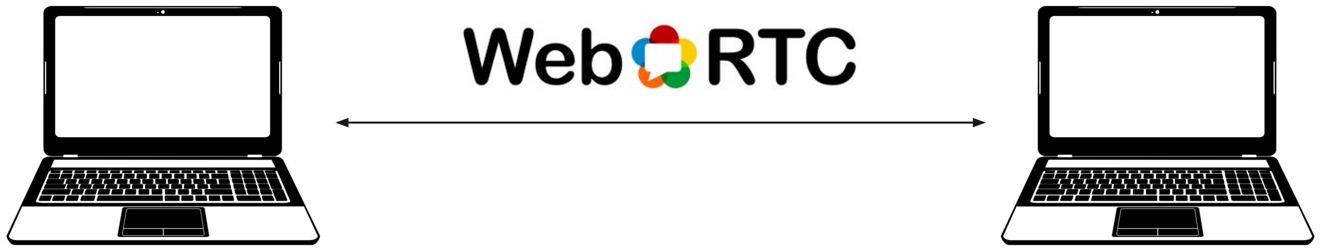
About WebRTC

- **Direct** connection between browsers



About WebRTC

- **Direct** connection between browsers via a collection of **Javascript API**
- Embedded in modern browsers, e.g. Chrome, Safari



Establishing a WebRTC Connection

1. **Signalling**
   - Clients connecting to a server, known as signalling server.

Establishing a WebRTC Connection

1. **Signalling**
   - Clients connecting to a server, known as signalling server.
2. **Metadata Exchange**
   - Clients exchanging necessary network information via the signalling server

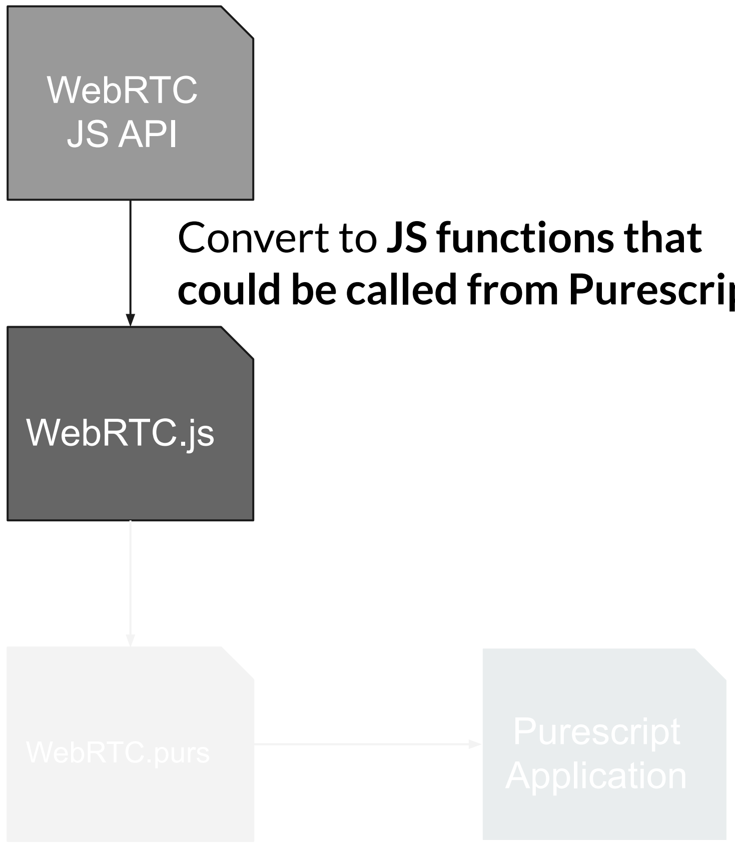
Establishing a WebRTC Connection

1. **Signalling**
    - Clients connecting to a server, known as signalling server.
2. **Metadata Exchange**
    - Clients exchanging necessary network information via the signalling server
3. **WebRTC Connection Establishment**
    - Once clients have reached an agreement on how to create the connection, and a webRTC connection is established

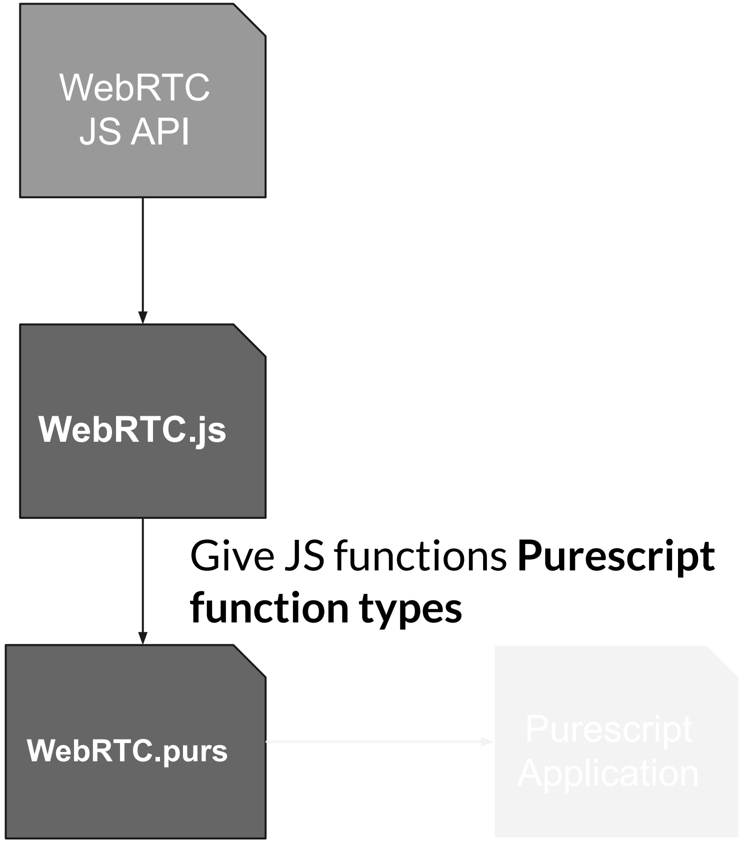
Back To Project

- Writing Purescript Bindings for the WebRTC Javascript API
- Implementing the Transport primitives for WebRTC in the Session Runtime Library
- Extending the Transport abstraction to incorporate WebRTC

Purescript Bindings for WebRTC via Foreign Function Interface

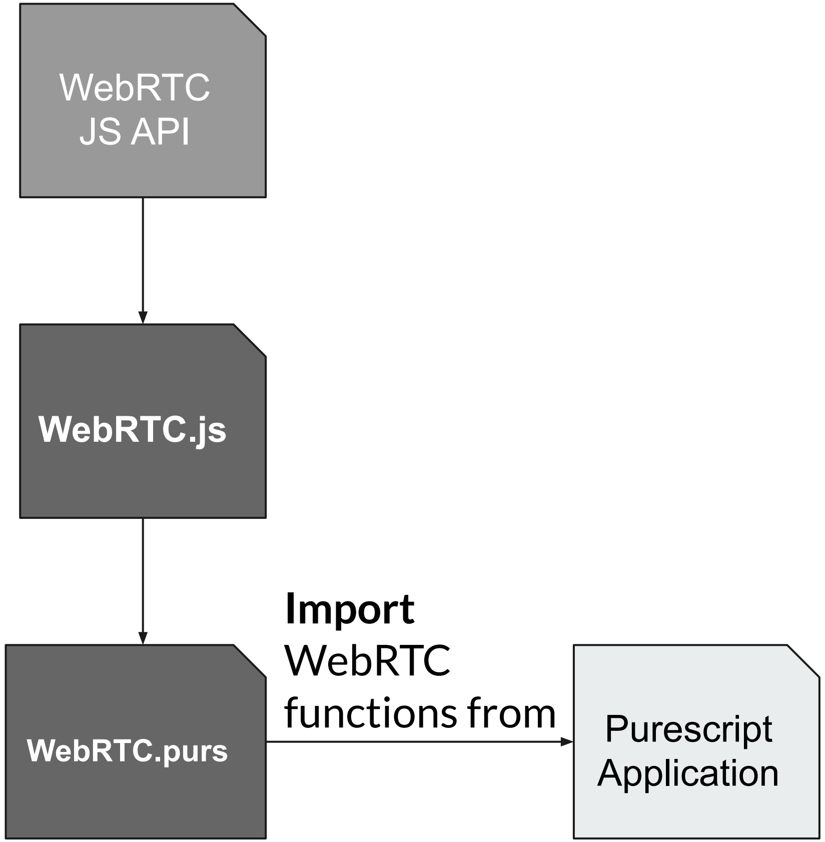

WebRTC JS API
Convert to **JS functions that could be called from Purescript**
WebRTC.js
WebRTC.purs
Purescript Application

Purescript Bindings for WebRTC via Foreign Function Interface

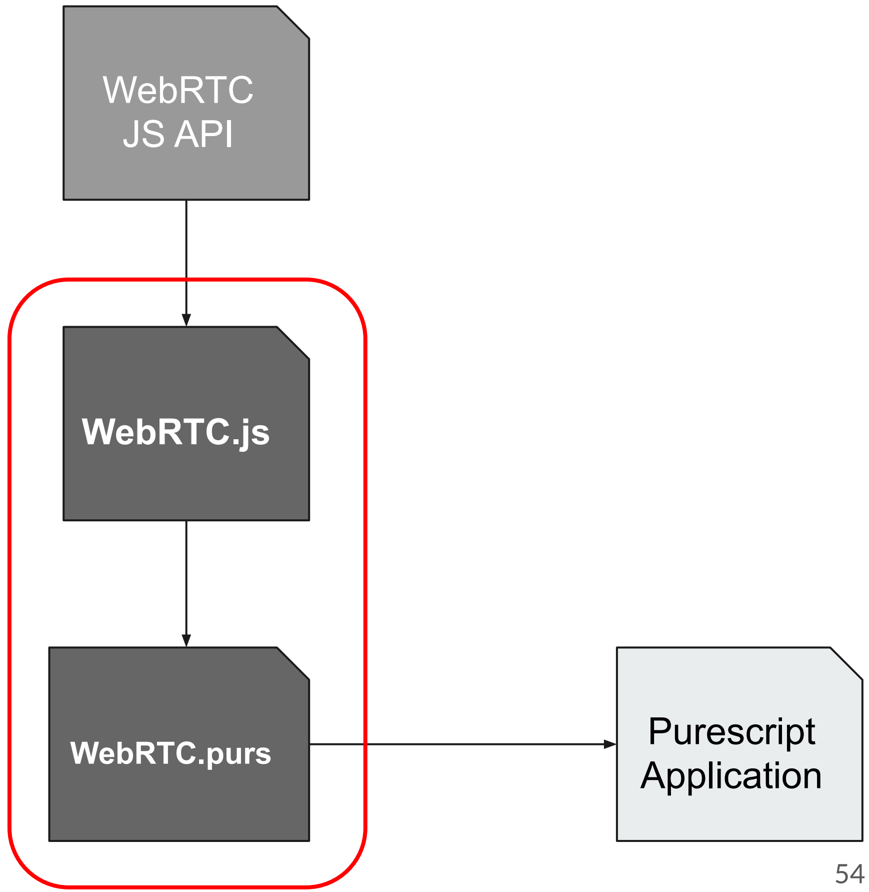


Give JS functions **Purescript function types**

Purescript Bindings for WebRTC via Foreign Function Interface

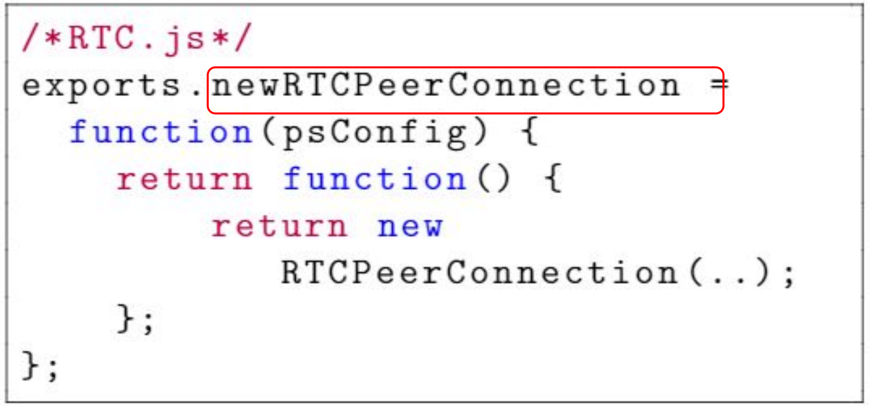


Purescript Bindings for WebRTC via Foreign Function Interface

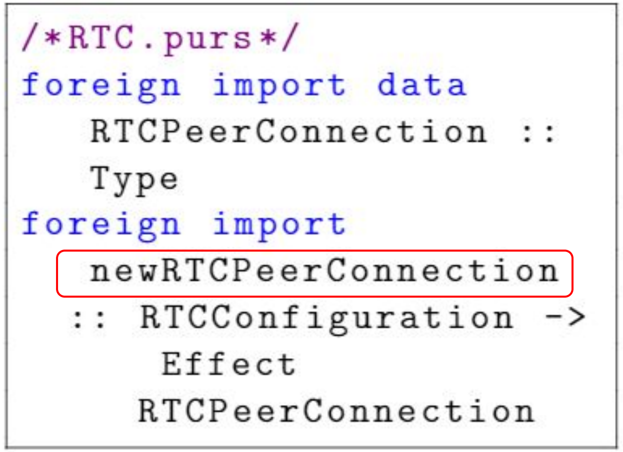


Example: RTCPeerConnection

```
connectionConfig = ...
localPeer = new RTCPeerConnection(connectionConfig)
```

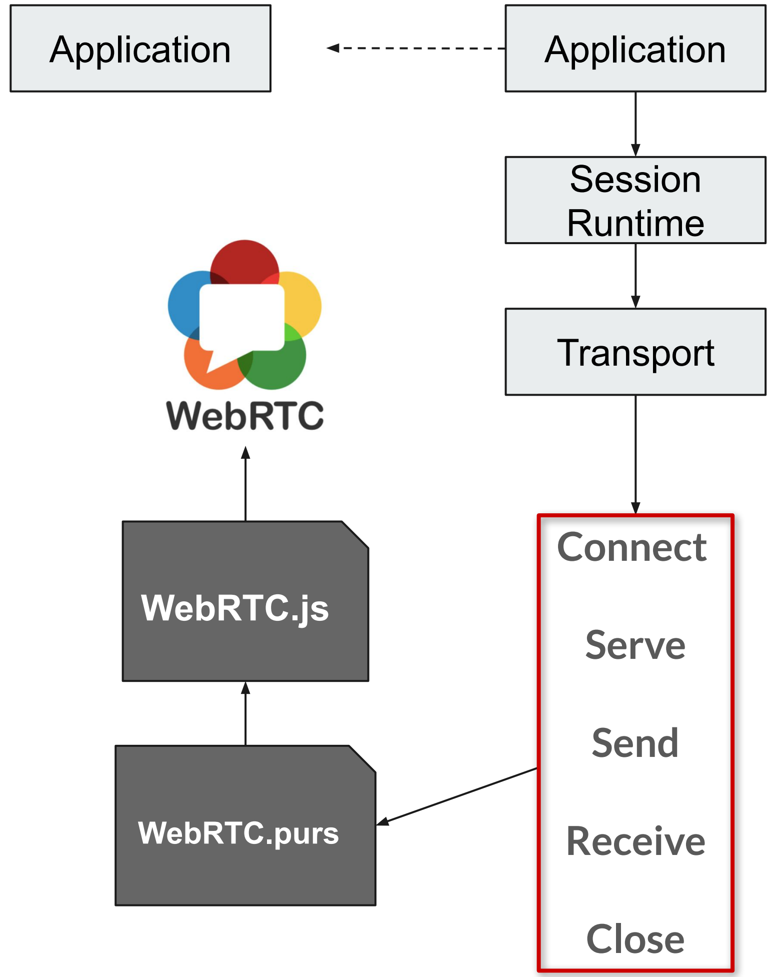


```
/*RTC.js*/
exports.newRTCPeerConnection =
  function(psConfig) {
    return function() {
        return new
           RTCPeerConnection(..);
    };
};
```

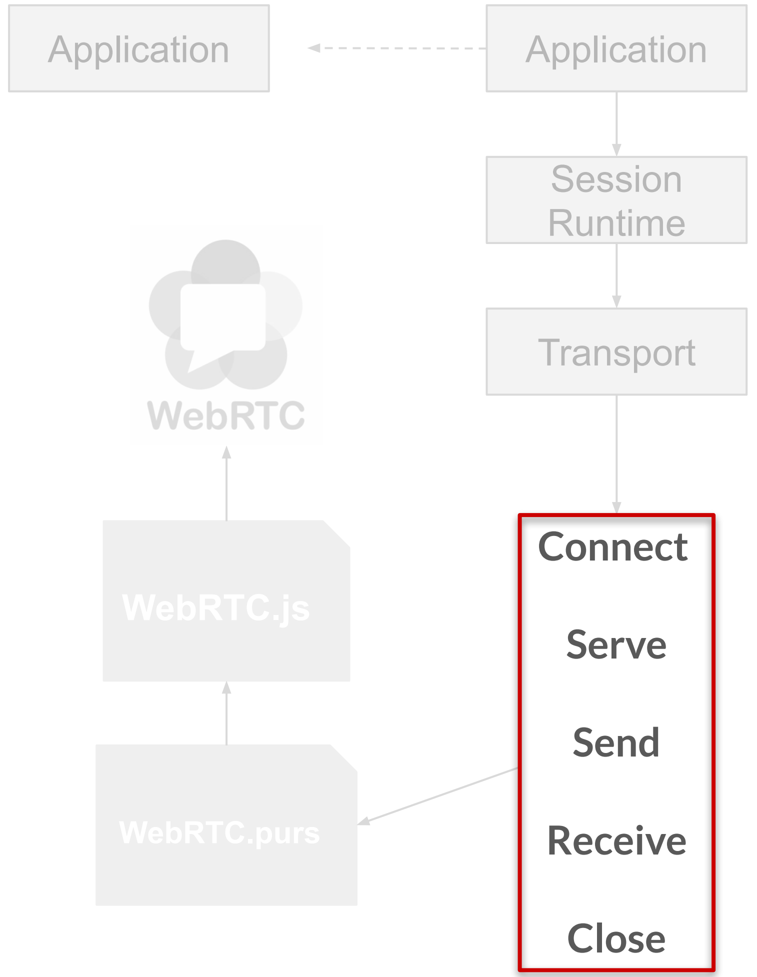


```
/*RTC.purs*/
foreign import data
   RTCPeerConnection ::
   Type
foreign import
   newRTCPeerConnection
  :: RTCConfiguration ->
      Effect
     RTCPeerConnection
```

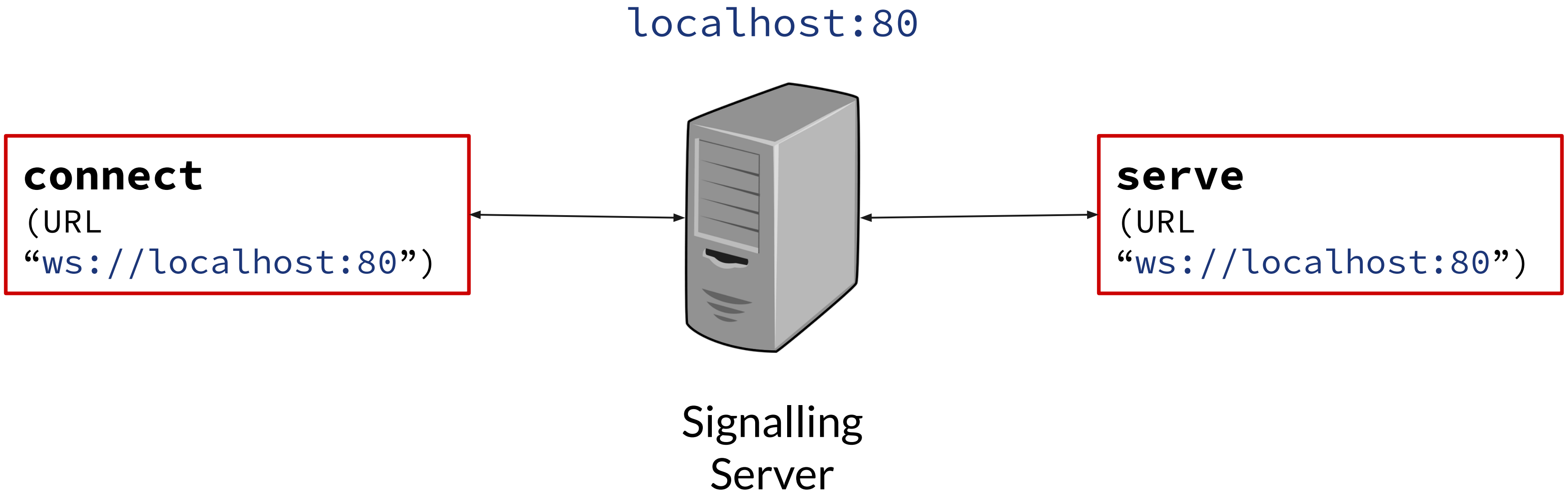
Implementing Transport Primitives for WebRTC



Implementing Transport Primitives for WebRTC



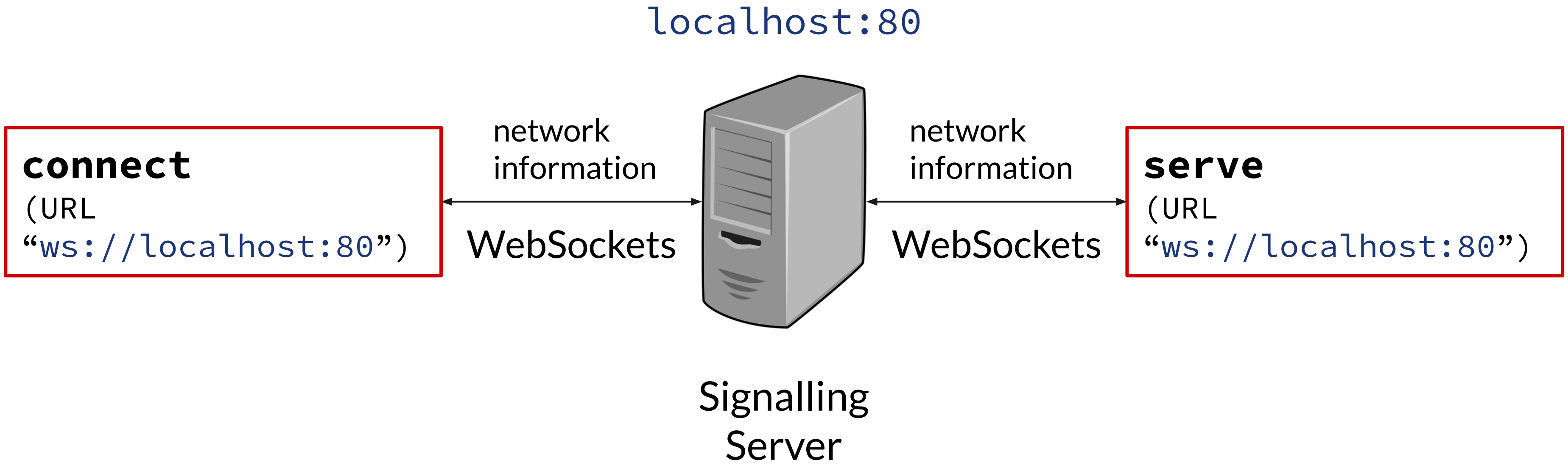
Overview of Connect and Serve



```
connect
(URL
"ws://localhost:80")
```

```
serve
(URL
"ws://localhost:80")
```

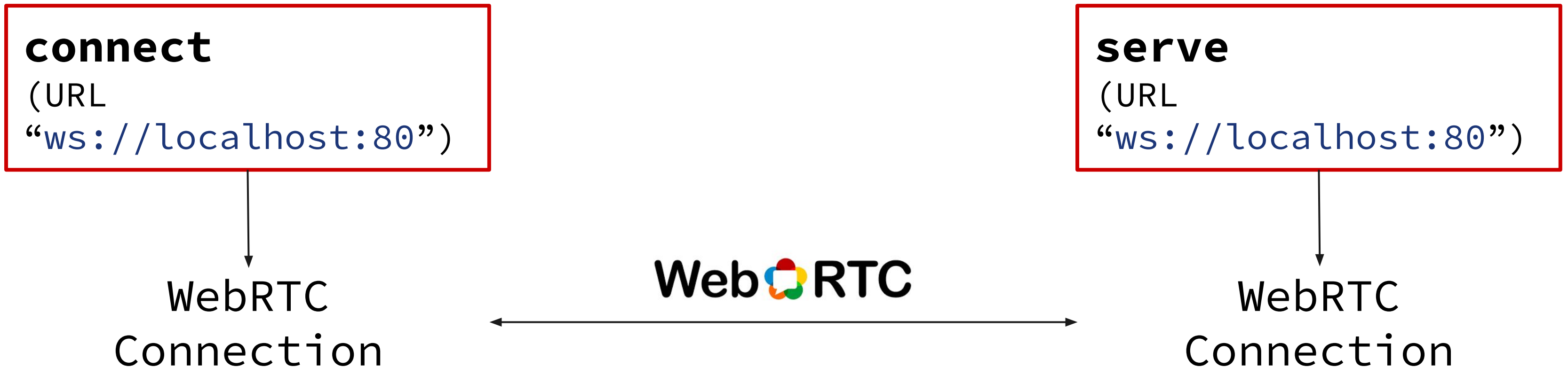
Overview of Connect and Serve



```
connect
(URL
"ws://localhost:80")
```

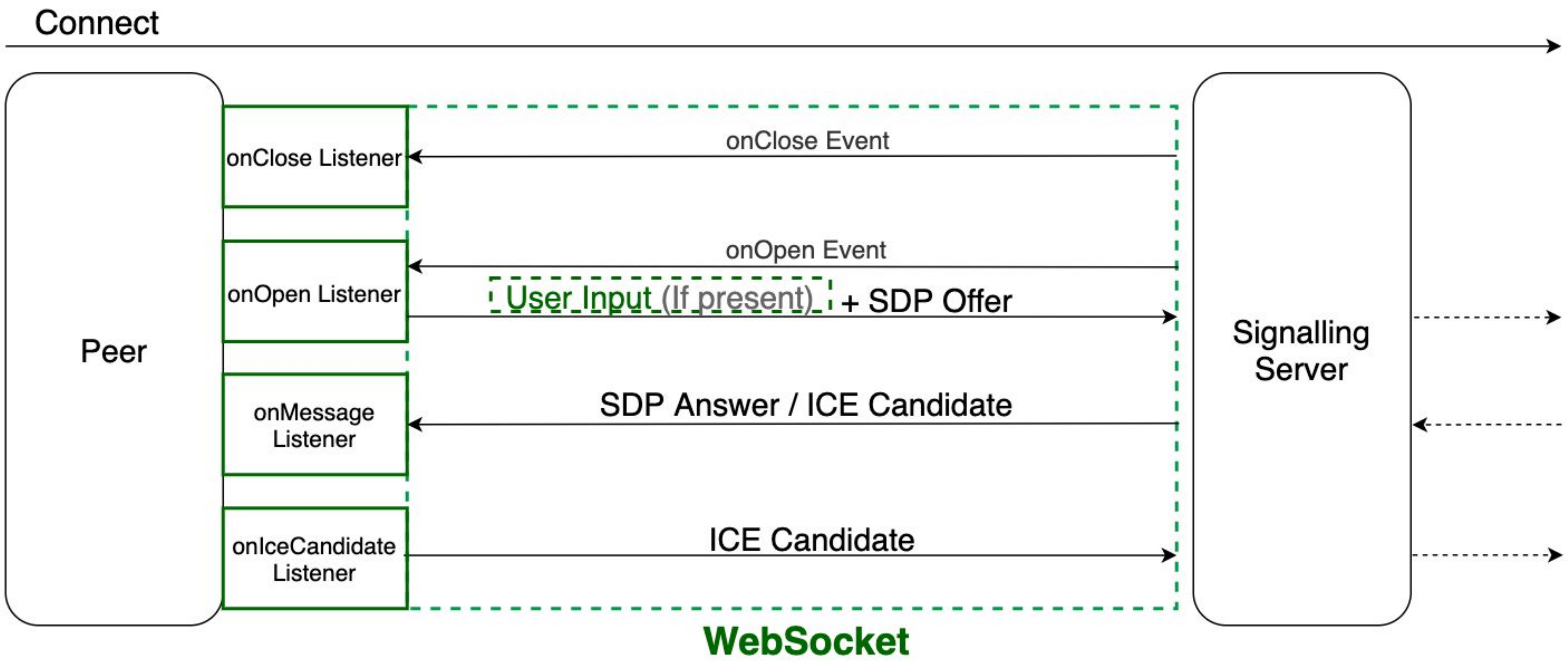
```
serve
(URL
"ws://localhost:80")
```

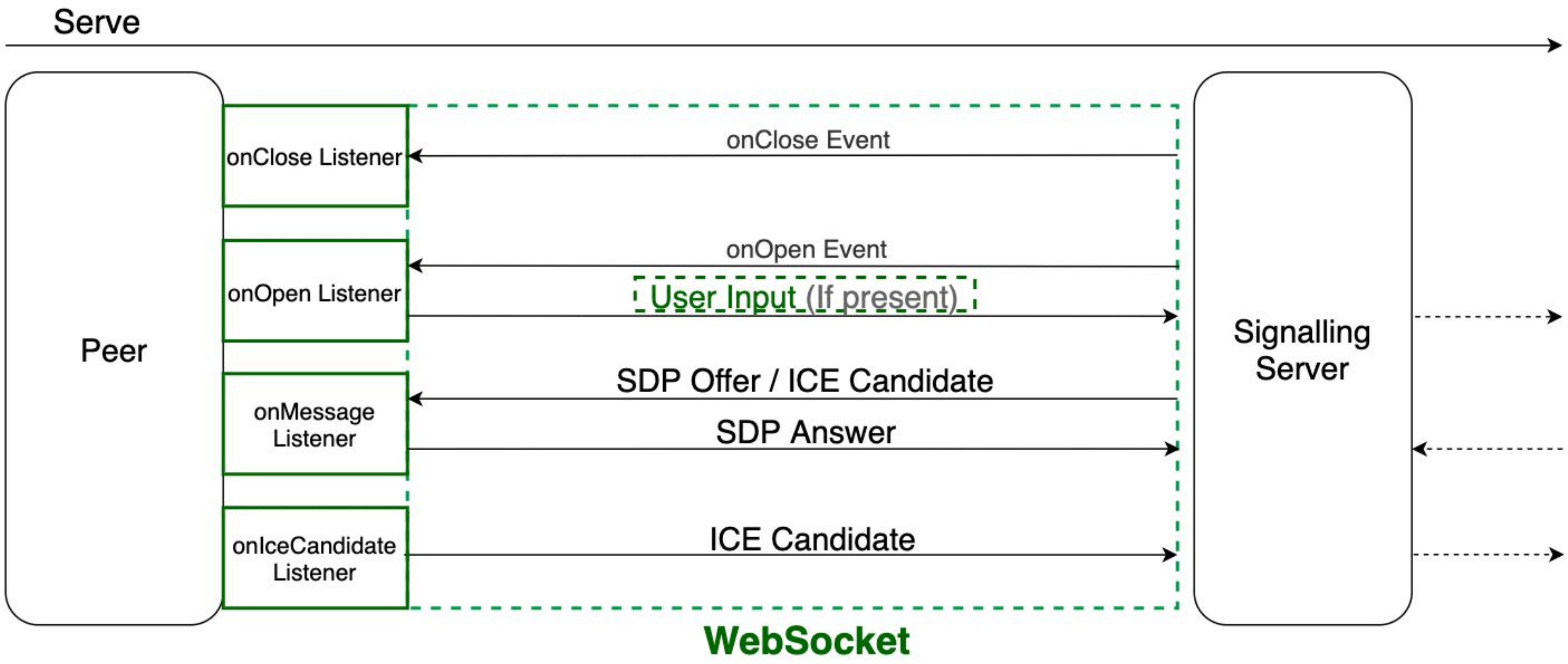
Overview of Connect and Serve



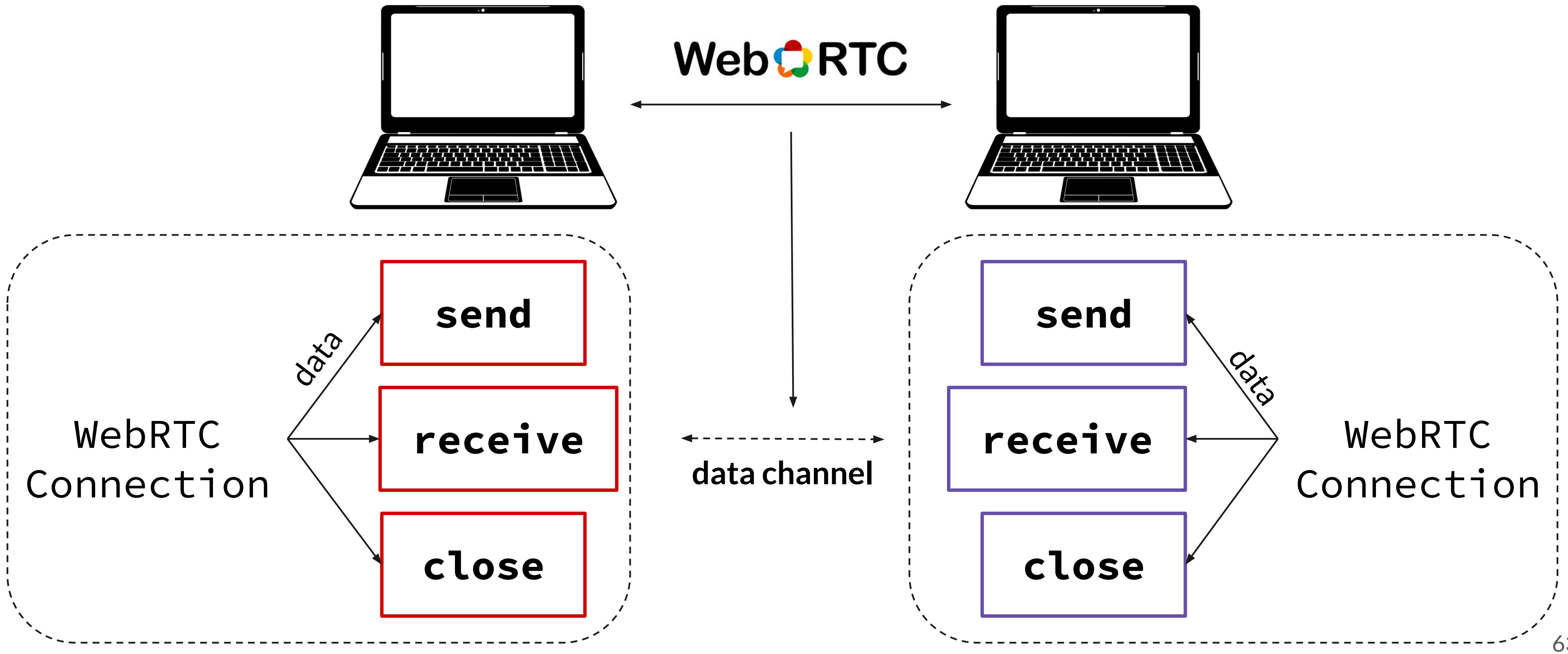
```
connect
(URL
"ws://localhost:80")
```

```
serve
(URL
"ws://localhost:80")
```


Connect
Peer
onClose Listener
onOpen Listener
onMessage Listener
onIceCandidate Listener
onClose Event
onOpen Event
User Input (If present) + SDP Offer
SDP Answer / ICE Candidate
ICE Candidate
Signalling Server
WebSocket


Serve
Peer
onClose Listener
onOpen Listener
onMessage Listener
onIceCandidate Listener
onClose Event
onOpen Event
User Input (If present)
SDP Offer / ICE Candidate
SDP Answer
ICE Candidate
Signalling Server
WebSocket

Overview of Send, Receive and Close



Extending Library's Transport Abstraction

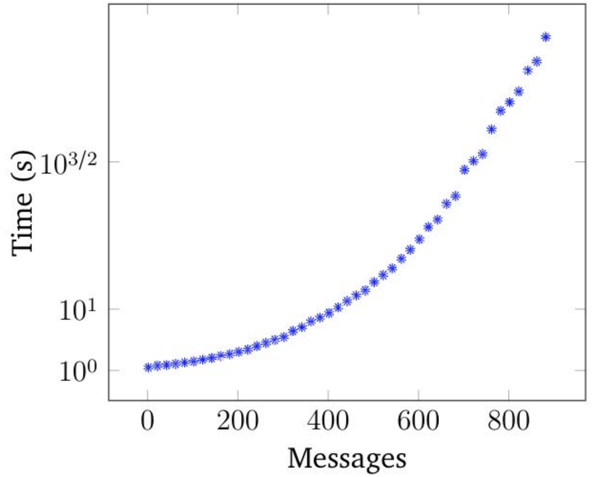
```
session
  (Proxy :: Proxy WebRTCConnection)
  (Role :: Role Client) do $
    connect (Role :: Role RemoteClient)
…..
```

Instance Instantiation of Transport Type Classes

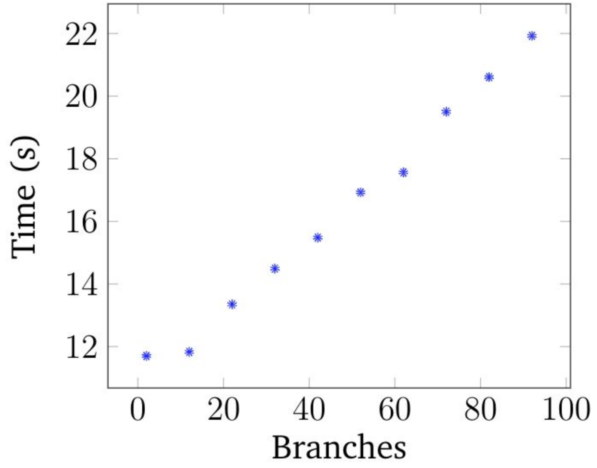
```
instance webRTCURLTransport :: Transport WebRTCConnection URL where
  send = \ws -> liftAff <<< (send ws)
  receive = liftAff <<< receive
  close = liftAff <<< close

instance webRTCURLTransportClient :: TransportClient WebRTCConnection
    URL ("loginMsg" :: Maybe String, "connInfo" :: {thisPeerId ::
   String, remotePeerId :: String}) where
  connect p x = liftAff $ connect p x.loginMsg x.connInfo

instance webRTCURLTransportServer :: TransportServer WebRTCConnection
    URL ("loginMsg" :: Maybe String, "connInfo" :: {thisPeerId ::
   String, remotePeerId :: String}) where
  serve p x = liftAff $ serve p x.loginMsg x.connInfo
```

Benchmark of Purescript Session Runtime Library

Benchmark Results - PingPong and Branching

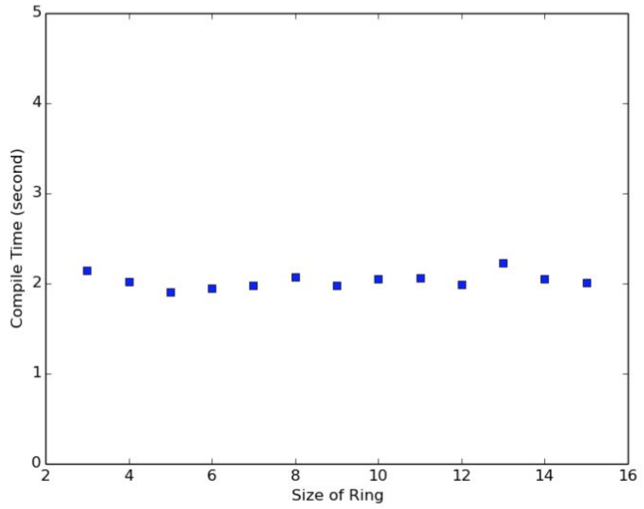


Ping-Pong

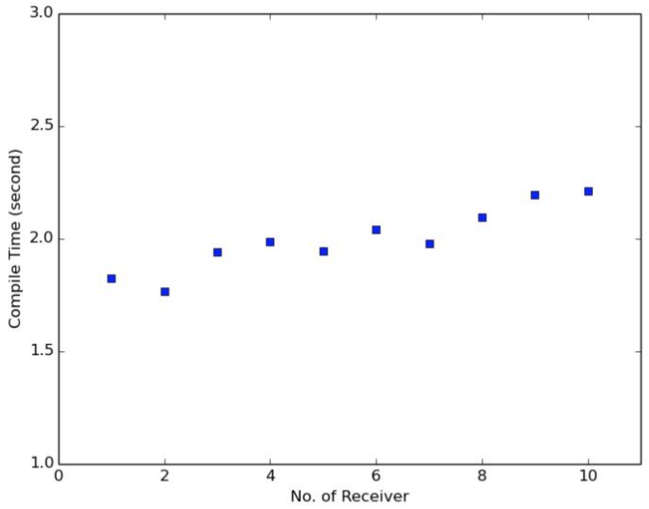


Branching

Benchmark Results - Ring and One-To-Many



Ring



One-to-Many

Demo

Challenges

Conclusion

- Improving error messages → Improves library’s **usability**
- Incorporating WebRTC → expands library’s **functionality**
- Benchmark → shows library’s **performance**

Conclusion and Future Work

- Improving error messages → Improves library's usability
- Incorporating WebRTC → expands library's functionality
- Benchmark → shows library's scalability
- Extend WebRTC to allow **audio** and **video** mediat data exchange
- Apply same approach to **Haskell**

Thank you