Session-based Distributed Programming in Java

Raymond Hu, Nobuko Yoshida

Imperial College London

Kohei Honda

Queen Mary University of London

Outline

- Session Types
- Aims
- An Example Application
- The Session Runtime and Benchmarks
- Conclusions and Future Work

Session Types

- Communication protocols (structured interaction sequences)

Session Types

- Communication protocols (structured interaction sequences)

Session Types

- Communication protocols (structured interaction sequences)

Asynchronous, reliable and ordered.

Session Types

- Communication protocols (structured interaction sequences)

A→B: “Hi, Bob.”

Session Types

- Communication protocols (structured interaction sequences)

A→B: “Hi, Bob.”

```
!<String>
```

Session Types

- Communication protocols (structured interaction sequences)

A→B: “Hi, Bob.”

`!<String>` *duals* `?(String)`

Session Types

- Communication protocols (structured interaction sequences)

A→B: “Hi, Bob.”

B→A: “Hi, Alice.”

`!<String>.?(String)` *duals* `?(String).!<String>`

Session Types

- Type systems for process calculi
  - Takeuchi et al. *An Interaction-based Language and its Typing System.* (PARLE '94)
  - Honda et al. *Language Primitives and Type Discipline for Structured Comm*… (ESOP '98)
  - Gay and Hole. *Subtyping for Session Types in the Pi Calculus*. (Acta Inf. 42: '05)
  - and many others…

- Integration into object calculi
  - Dezani-Ciancaglini et al. *A distributed Object-Oriented Language with Session*... (TGC '05)
  - Dezani-Ciancaglini et al. *Session Types for Object-Oriented Languages*. (ECOOP '06)
  - …

- Theoretical basis for Web services
  - Carbone et al. *A Theoretical Basis of Communication-Centred, Concurrent Prog*... ('06)
  - Carbone et al. *Structured Communication-Centred Programming for Web*… (ESOP '07)
  - …

Why Session Types?

➢ Communications programming is challenging

- RPC (RMI):
  - × Basic send-receive shape not readily composed to more general interaction structures
- (TCP) Sockets:
  - × No direct representation of protocols in program structure
  - × Program verification is more difficult
- Higher-order communications? (session delegation)

Why Session Types?

➢ Communications programming is challenging

- RPC (RMI):
  - × Basic send-receive shape not readily composed to more general interaction structures
- (TCP) Sockets:
  - × No direct representation of protocols in program structure
  - × Program verification is more difficult
- Higher-order communications? (session delegation)

Why Session Types?

➢ Communications programming is challenging

- RPC (RMI):
  - × Basic send-receive shape not readily composed to more general interaction structures
- (TCP) Sockets:
  - × No direct representation of protocols in program structure
  - × Program verification is more difficult
- Higher-order communications? (session delegation)

Outline

- Session Types
- Aims
- An Example Application
- The Session Runtime and Benchmarks
- Conclusions and Future Work

Aims

- Sessions and session types in a practical O-O language for distributed programming.
  - Sessions as a programming abstraction
  - Session types for communication-safety
  - Session language framework and runtime architecture

Aims

- Sessions and session types in a practical O-O language for distributed programming.
  - Benefits/restrictions of sessions?
  - Impact on language design?
  - Can we write useful and realistic apps?
  - – on
  - – Session types for communication-safety
  - – Session language framework and runtime architecture

Aims

- Sessions and session types in a practical O-O language for distributed progr[illegible]ing.
  - Sessions as a programm[illegible]
  - Session types for comm[illegible]
  - Session language framework and runtime architecture

- Asynchronous communication
- Distributed session peers
- Session subtyping, session interleaving
- Failure handling
- (Performance)
- …

Aims

- Sessions and session types in a practical O-O language for distributed programming.
  - Sessions as a programming abstraction
  - Session typ
  - Session lang cture

- What is session programming?
- How to integrate the above features?
- In Java?

Aims

- Sessions and session types in a practical O-O language for distributed programming.
  - Sessions as a programming abstraction
  - Session types for communication-safety (and more)
  - Session langua

- Integration of session types and O-O (linearity)
- Communication safety between distributed peers

Aims

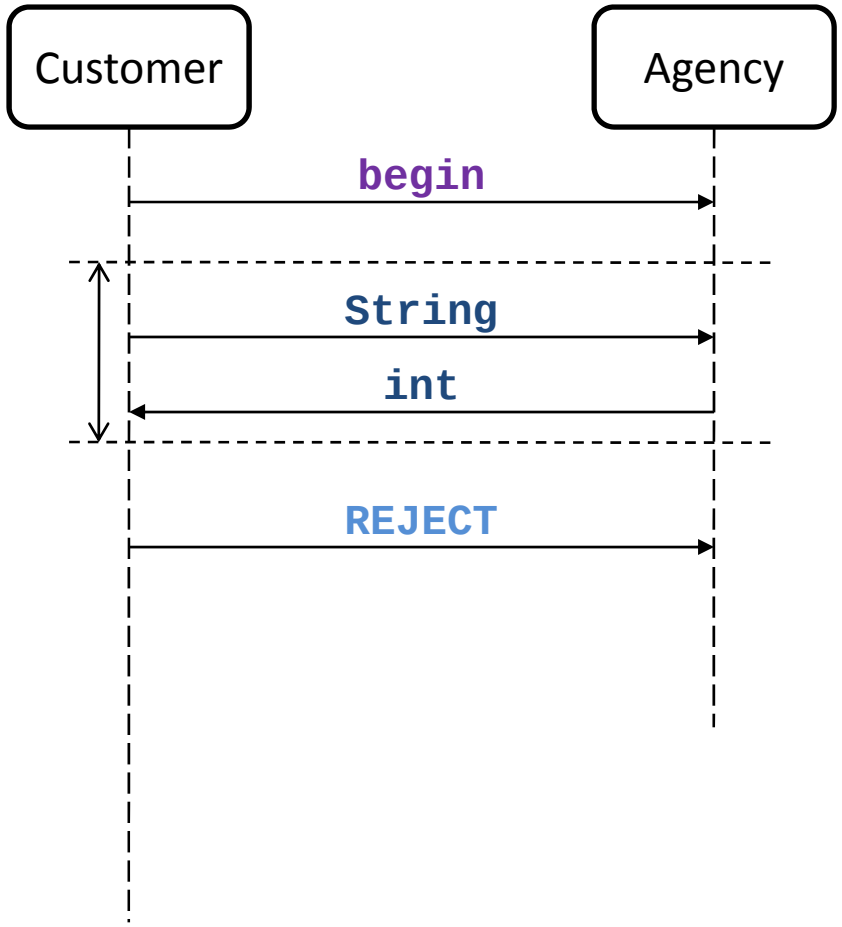
- Sessions and session types in a practical O-O language for distributed programming.
  - Sessions as a progr
  - Session types for c
  - Session language framework and runtime architecture

- Realising session abstraction over concrete transports
- Session delegation

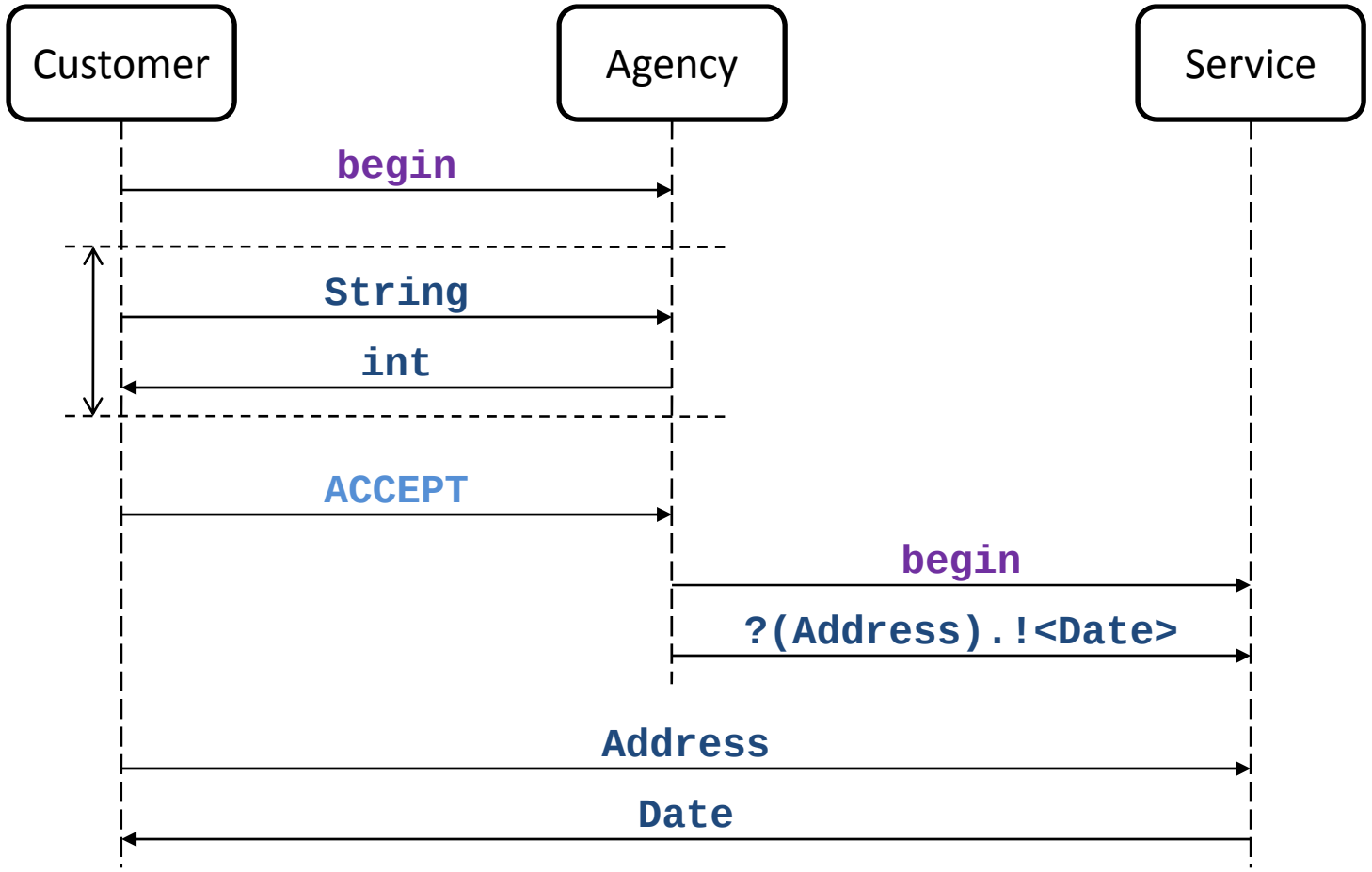
Outline

- Session Types
- Aims
- An Example Application
- The Session Runtime and Benchmarks
- Conclusions and Future Work

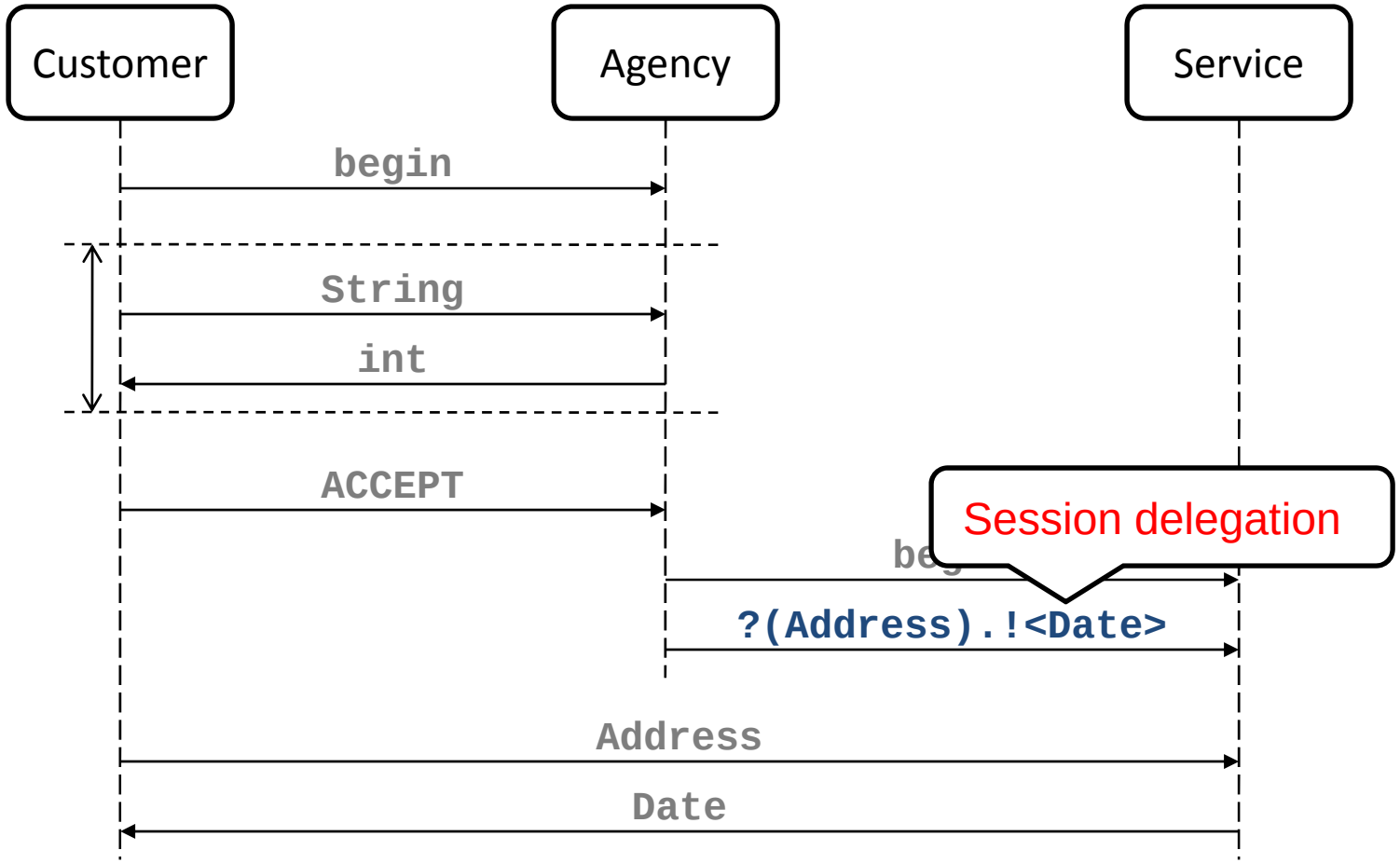
An Online Travel Agency



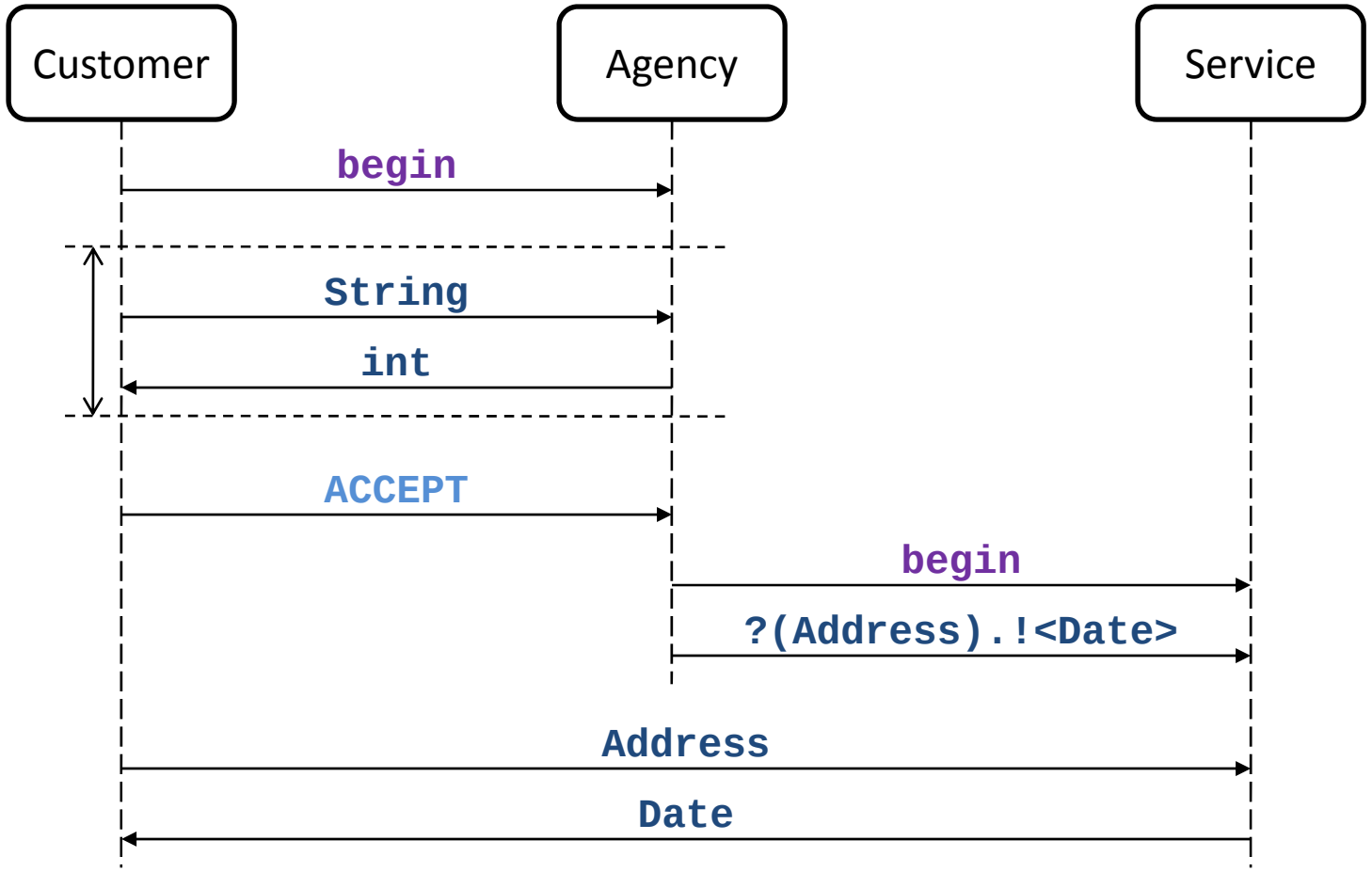
An Online Travel Agency



An Online Travel Agency

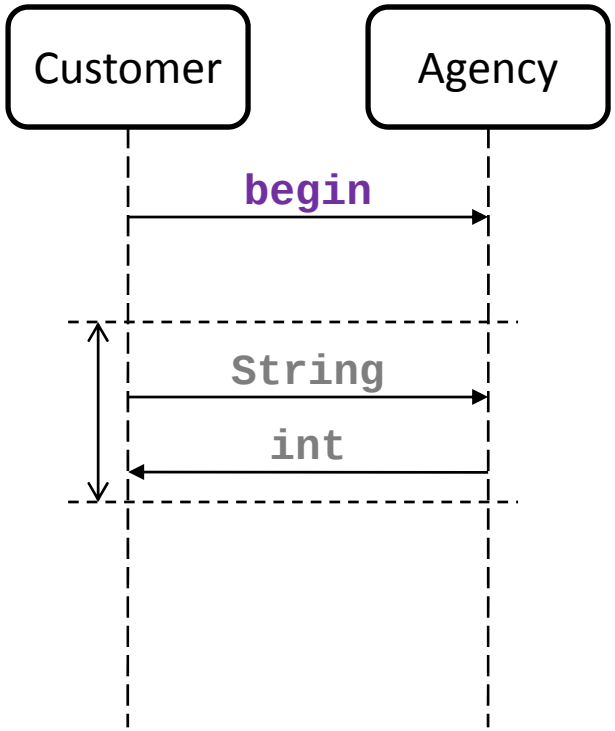


An Online Travel Agency



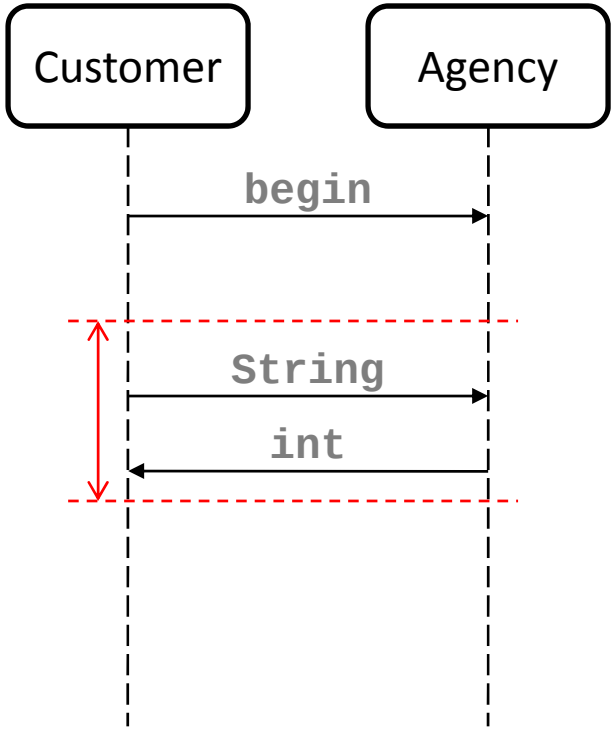
Session Type Declarations (1)

```
protocol p {
  begin

}
```



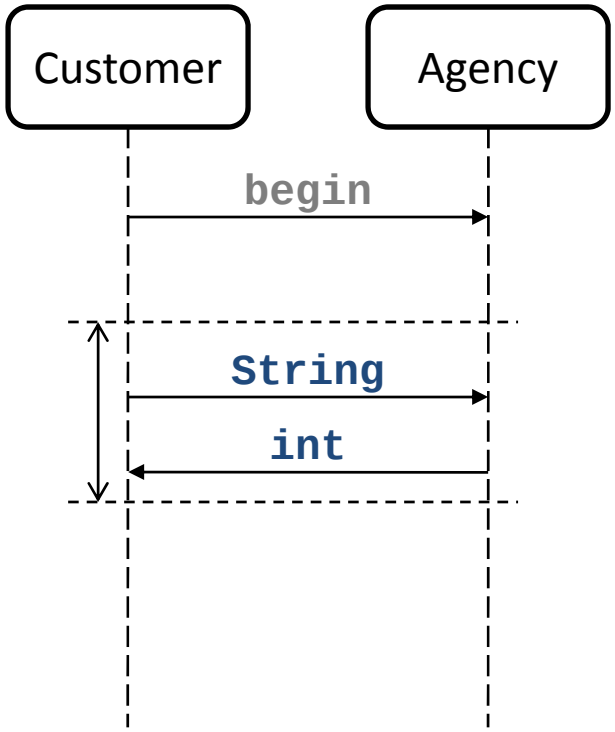
Session Type Declarations (1)

```
protocol p {
  begin.

    !<String>.
    ?(int)


}
```



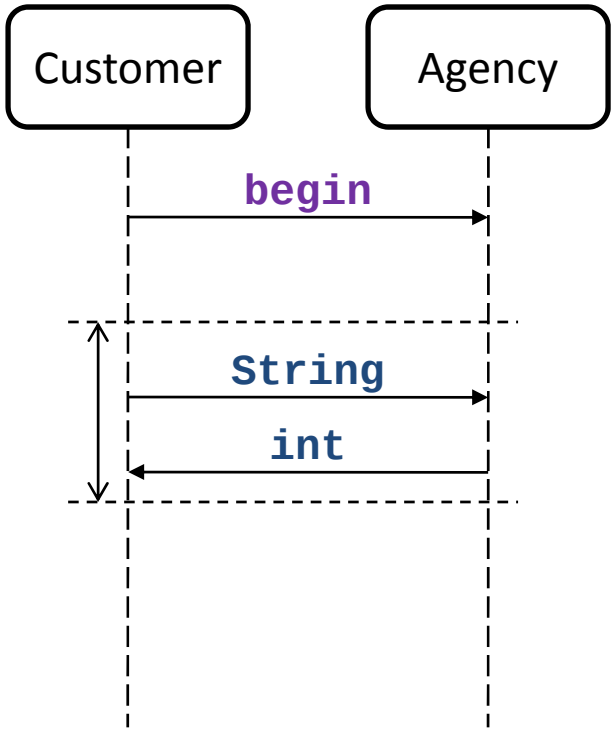
Session Type Declarations (1)

```
protocol p {
  begin.
  ![
    !<String>.
    ?(int)
  ]*.
  …
}
```



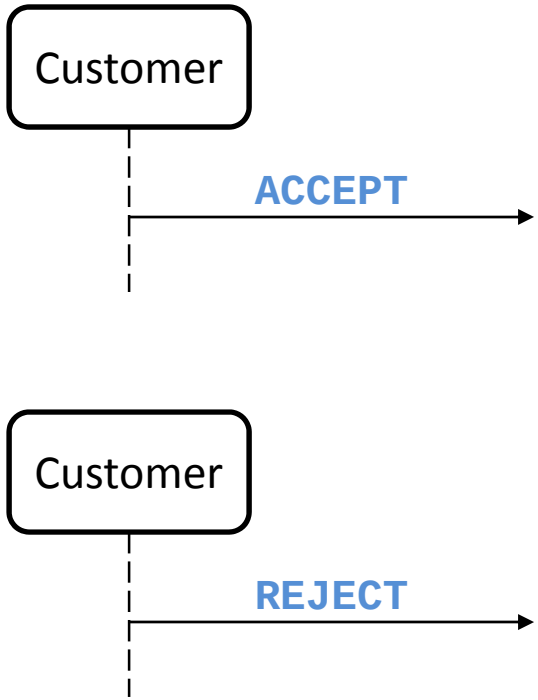
Session Type Declarations (1)

```
protocol p {
  begin.
  ![
    !<String>.
    ?(int)
  ]*.
  …
}
```



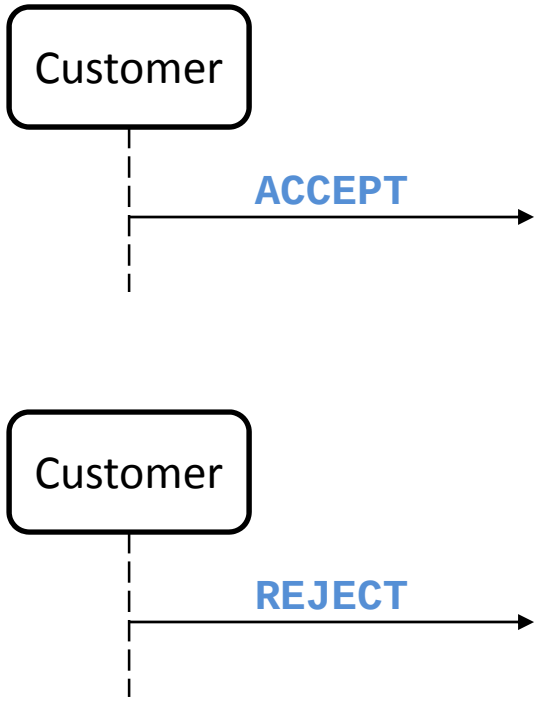

```
// Dual type.
protocol p {
  begin.
  ?[
    ?(String).
    !<int>
  ]*.
  …
}
```

Session Type Declarations (1)




```
!{
  ACCEPT: {
    !<Address>.
    ?(Date)
  }

}
```

Session Type Declarations (1)




```
!{
   ACCEPT: {
      !<Address>.
      ?(Date)
   },

   REJECT: {

   }
}
```

Implementing Customer (1)

```
protocol p {
  begin.

  ![
    !<String>.
    ?(int)
  ]*
  …
```

```
SJSocket s = SJSocket
    .create(p, agency, port);
```

Implementing Customer (2)

```
protocol p {
  begin.

  ![
    !<String>.
    ?(int)
  ]*
  …
```

```
SJSocket s = SJSocket
    .create(p, agency, port);

s.request();
```

Implementing Customer (2)

```
protocol p {
  begin.

  ![
    !<String>.
    ?(int)
  ]*
  …
```

```
SJSocket s = SJSocket
     .create(p, agency, port);

s.request();


   s.send("PARIS/EUROSTAR");
   cost = s.receive();
```

Implementing Customer (2)

```
protocol p {
  begin.

  ![
    !<String>.
    ?(int)
  ]*
  …
```

```
SJSocket s = SJSocket
      .create(p, agency, port);

s.request();

s.outwhile(…) {
   s.send("PARIS/EUROSTAR");
   cost = s.receive();
}
…
```

Implementing `Customer` (3)

```
!{
  ACCEPT: {
    …
  },

  REJECT: {

  }
}
```

```
s.outbranch(ACCEPT) {
  …
}
```

Implementing `Customer` (3)

```
!{
  ACCEPT: {
    …
  },

  REJECT: {

  }
}
```

```
if (cost < 100) {
  s.outbranch(ACCEPT) {
    …
  }
} else {
  s.outbranch(REJECT) {

  }
}
```

Implementing Customer (4)

```
protocol p {
  begin.
  ![
    !<String>.
    ?(int)
  ]*
  !{
    ACCEPT: {
      !<Address>.
      ?(Date)
    },
    REJECT: { }
  }
}
```

```
SJSocket s = SJSocket.create(p, …);

s.request();
s.outwhile(…) {
  s.send("PARIS/EUROSTAR");
  cost = s.receive();
}
if (…) {
  s.outbranch(ACCEPT) {
    s.send(…);
    date = s.receive();
  }
} else {
  s.outbranch(REJECT) { }
}
```

Implementing `Customer` (5)

```
protocol p { … }

SJSocket s = SJSocket.create(p, host, port);

try (s) {
  s.request();
  s.outwhile(…) {
    …
}
catch (SJIOException ioe) { … }

finally { … }
```

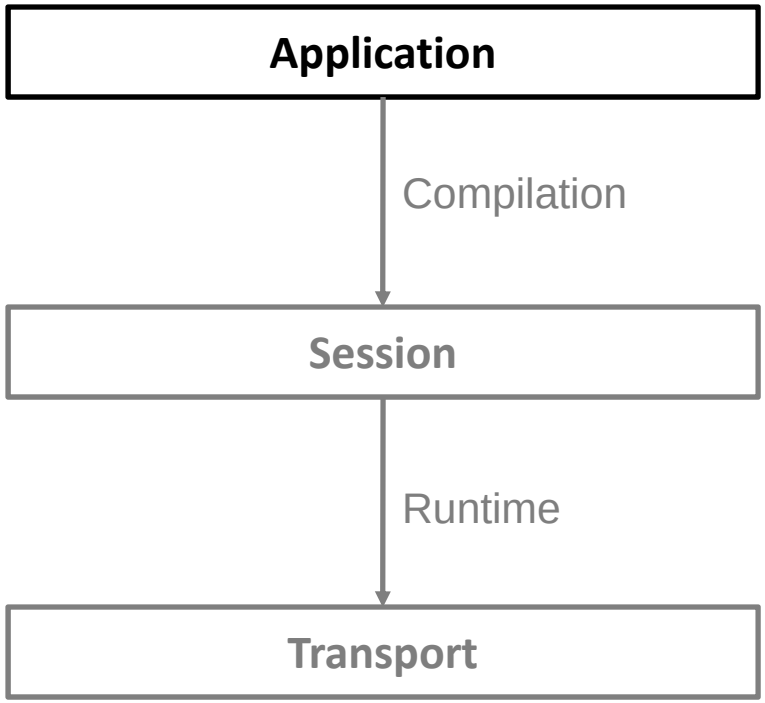
Implementing `Customer` (5)

```
protocol p { … }

SJSocket s = SJSocket.create(p, host, port);

try (s) {
  s.request();
  s.outwhile(…) {
    …
}
catch (SJIOException ioe) { … }
catch (SJIncompatibleSessionException ise) { … }
finally { … }
```

Outline

- Session Types
- Aims
- An Example Application
- The Session Runtime and Benchmarks
- Conclusions and Future Work

SJ Session Framework



SJ Session Programs

```
s.send(123);
s1.send(s2);
s.outwhile(…) { … }
```

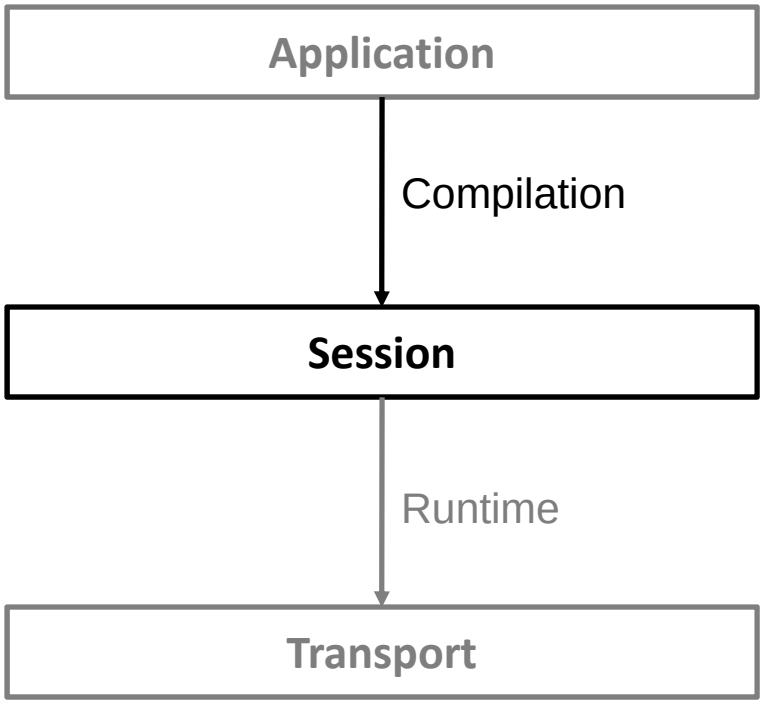
SJ Runtime Interface

```
sendInt(s, 123);
delegateSession(…);
outsync(s, …);
```

Transport Interface

```
writeBytes(…);
```

SJ Session Framework



SJ Session Programs

```
s.send(123);
s1.send(s2);
s.outwhile(…) { … }
```

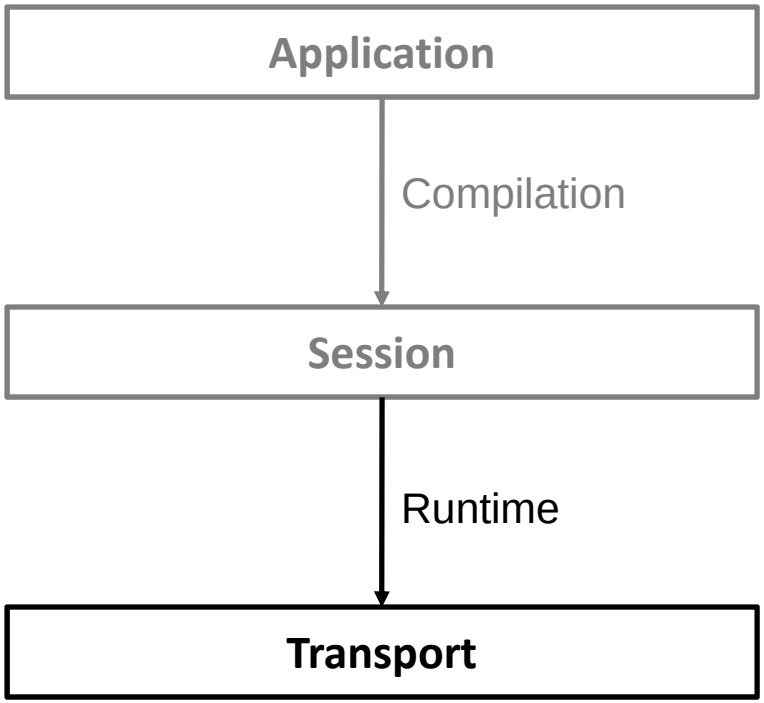
SJ Runtime Interface

```
sendInt(s, 123);
delegateSession(…);
outsync(s, …);
```

Transport Interface

```
writeBytes(…);
```

SJ Session Framework



SJ Session Programs

```
s.send(123);
s1.send(s2);
s.outwhile(…) { … }
```

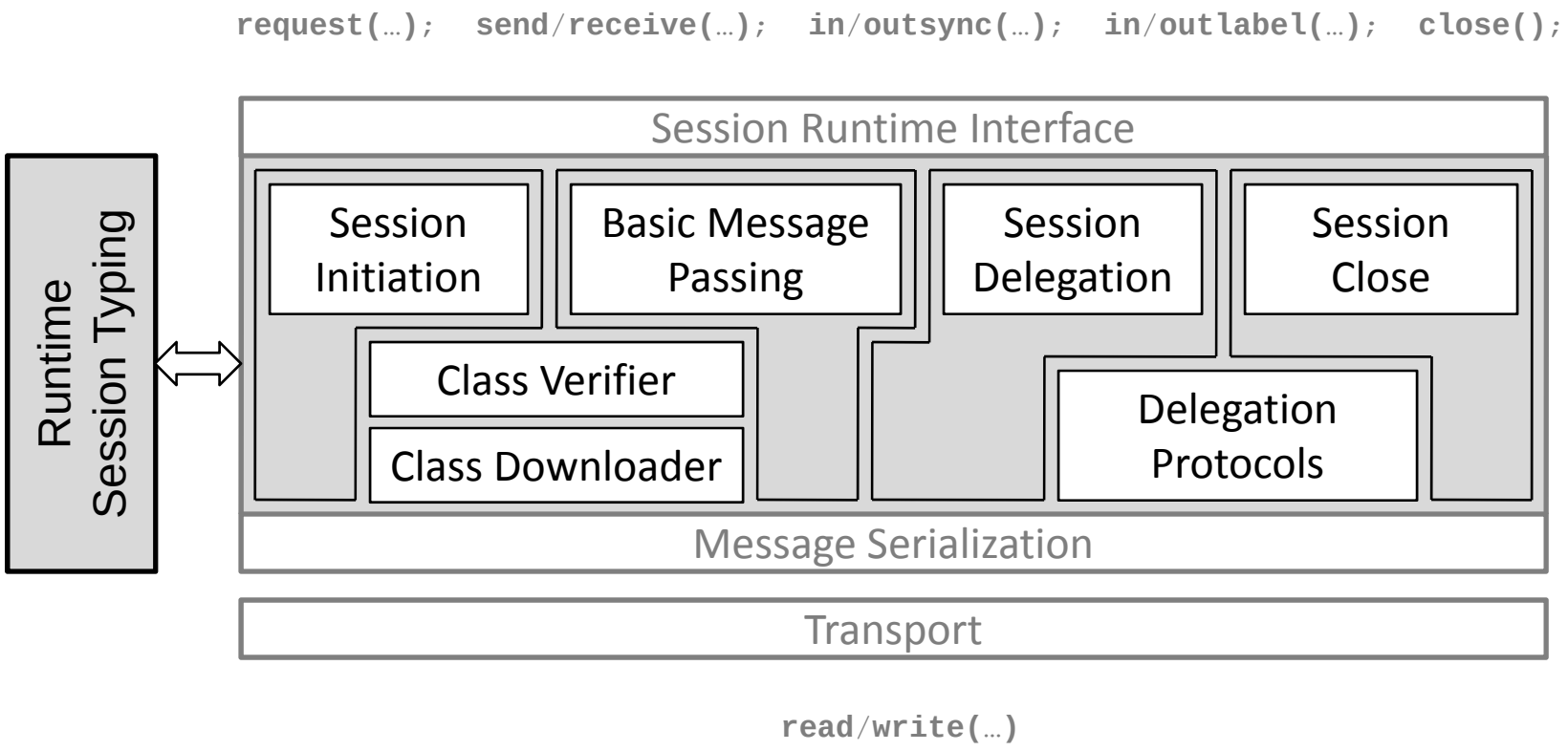
SJ Runtime Interface

```
sendInt(s, 123);
delegateSession(…);
outsync(s, …);
```

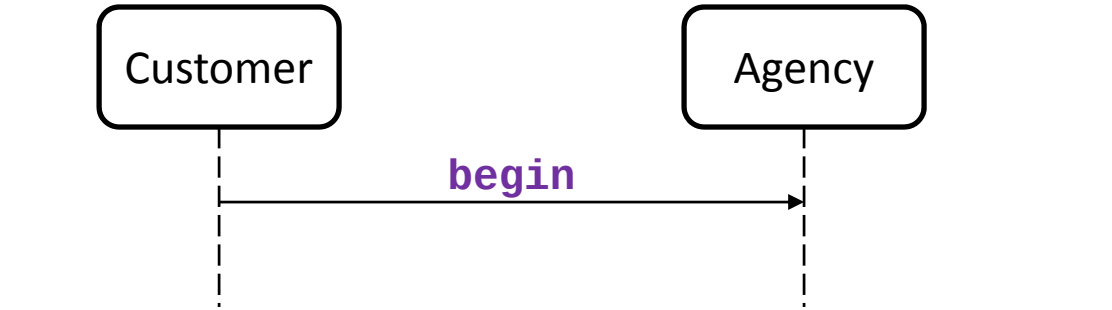
Transport Interface

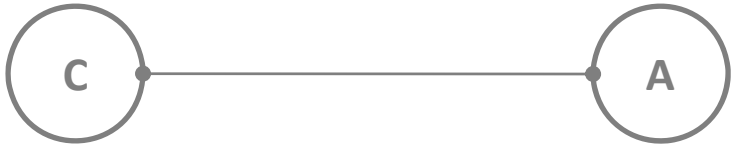
```
writeBytes(…);
```

SJ Session Runtime



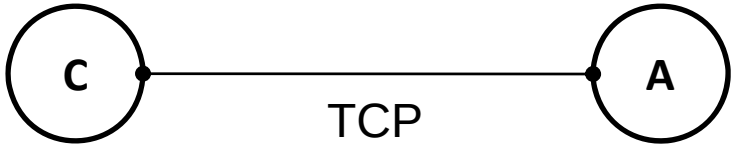
Sessions over TCP (Initiation)





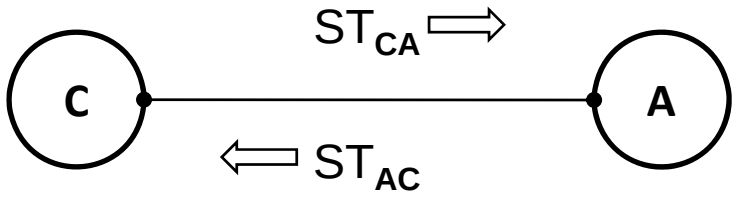

```
new Socket();          … = ss.accept();
```

Sessions over TCP (Initiation)




```
new Socket();          … = ss.accept();
```

Sessions over TCP (Initiation)

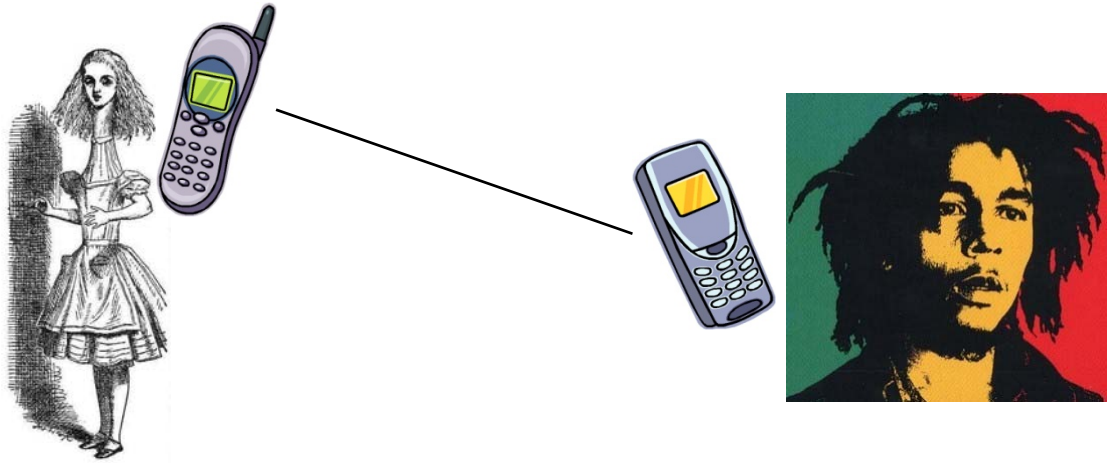



```
objectOut.writeObject(…);
```

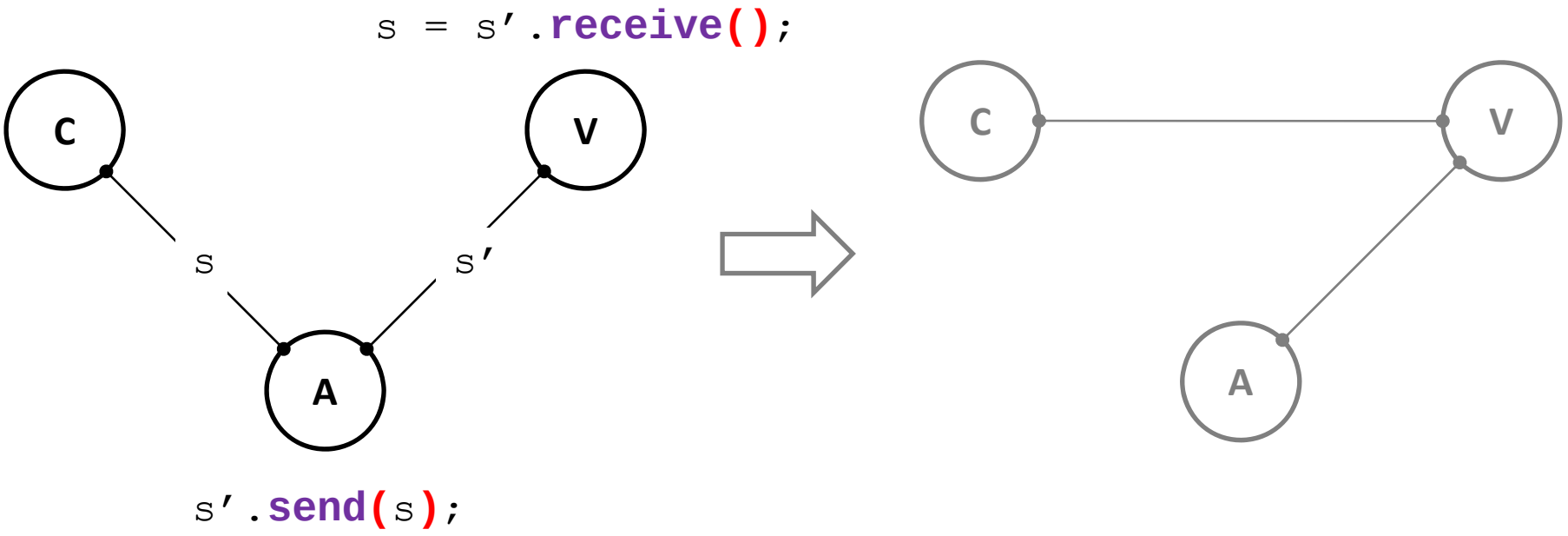
Implementing Session Delegation

- What is session delegation?
  - Higher-order communication:

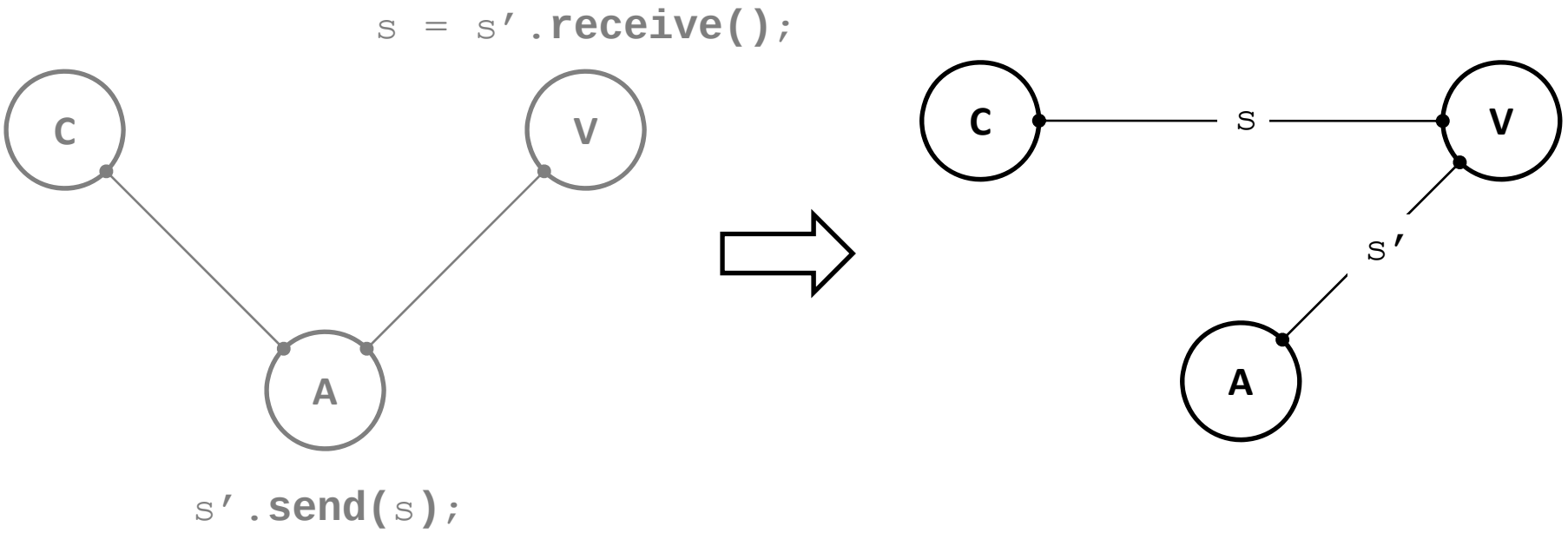
    type-safe connection dynamics.
  - Transparent to the passive party.



Implementing Session Delegation



Implementing Session Delegation



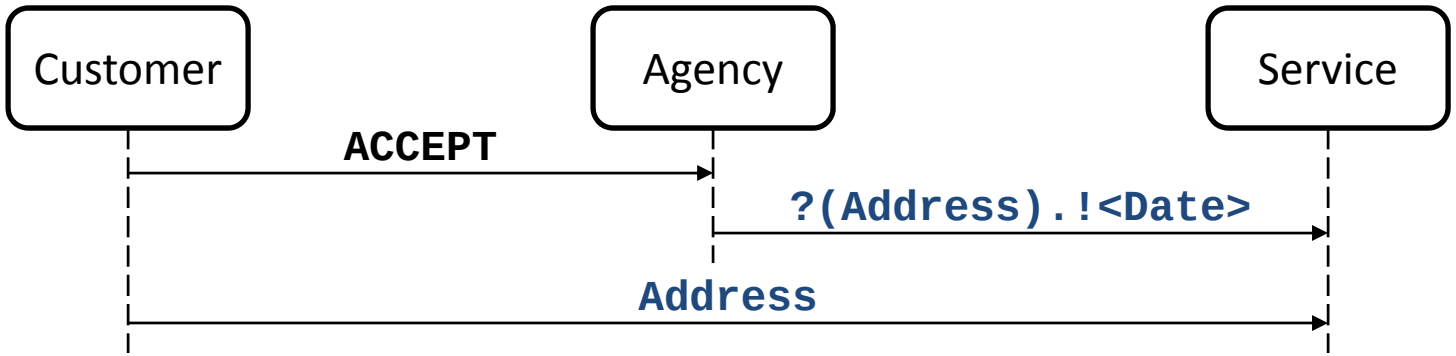
Implementing Session Delegation

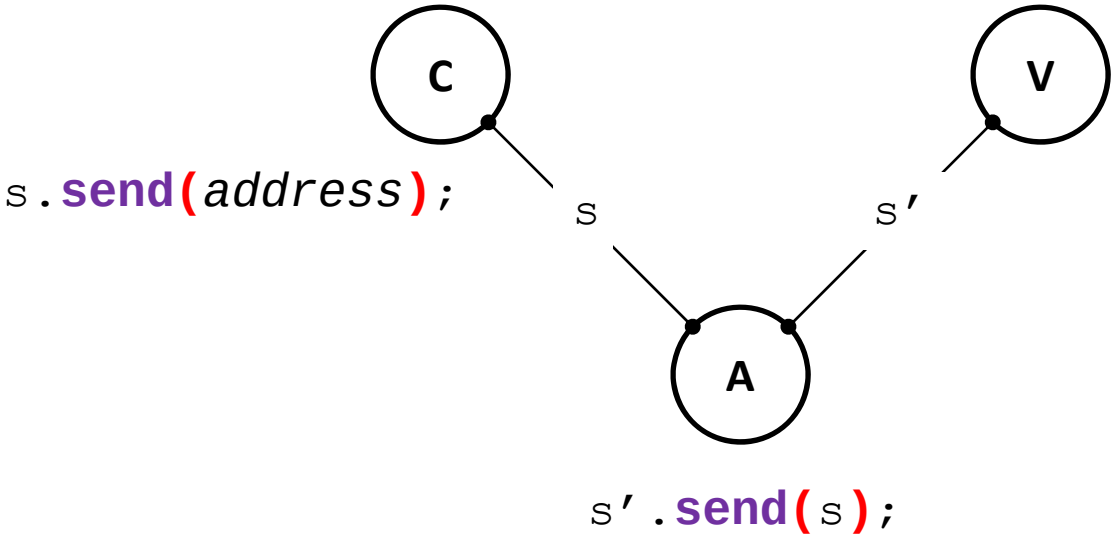
- What is session delegation?
  - Higher-order communication:

    type-safe connection dynamics.
  - Transparent to the passive party.

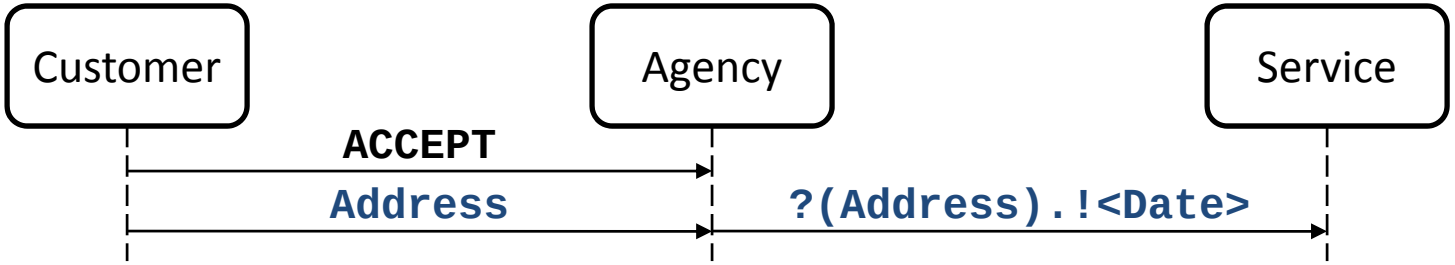
- The tricky part?
  - ➢ Asynchronous communication between distributed peers.

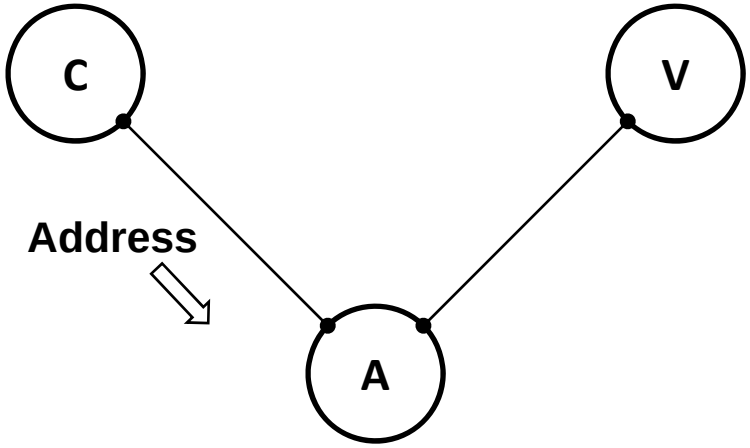
“Lost Messages”





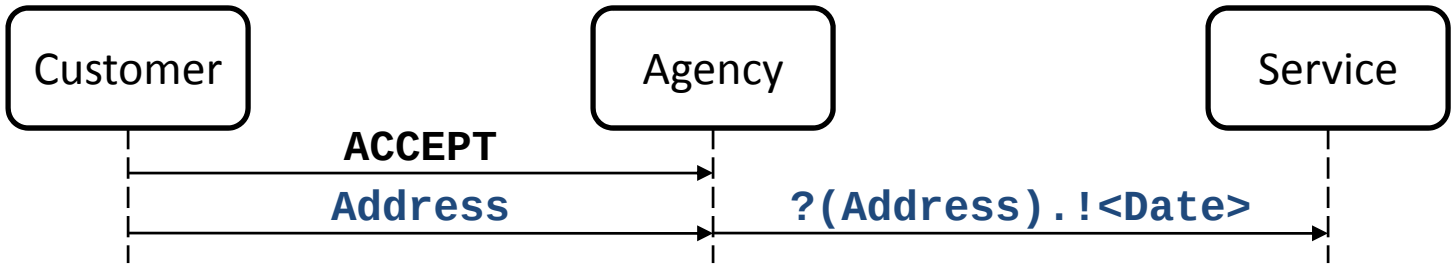
“Lost Messages”

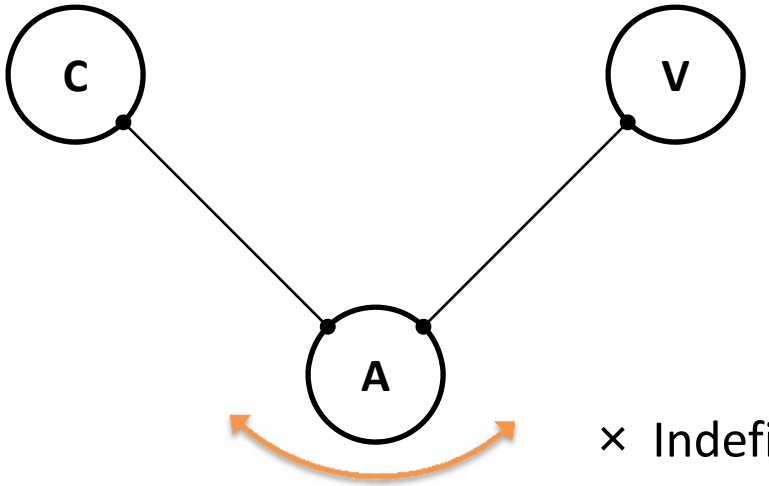





```
s'.send(s);
```

“Lost Messages”





× Indefinite Forwarding

Implementing Session Delegation

- Reconnection-based delegation:
  - (Bounded) “lost message” forwarding
  - “Lost message” resending

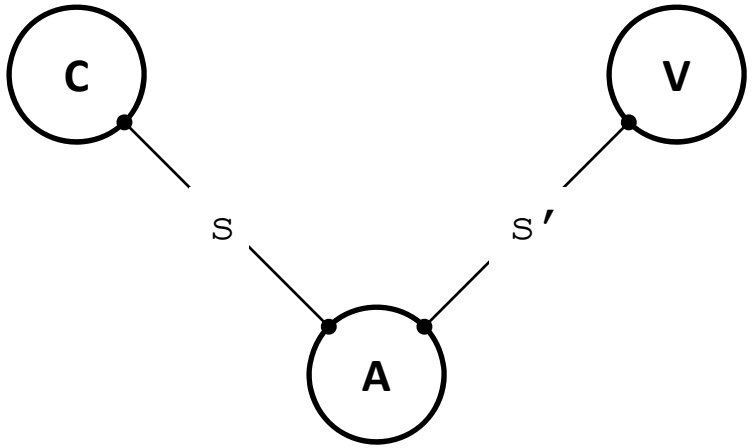
  (Runtime protocols for coordinating connection renegotiation and message rerouting/resending.)

  ➢Session types for runtime coordination

Delegation Protocol: “Resending”

```
// ….!{ACCEPT: }
s.send(address);
```

```
s = s′.receive();
```

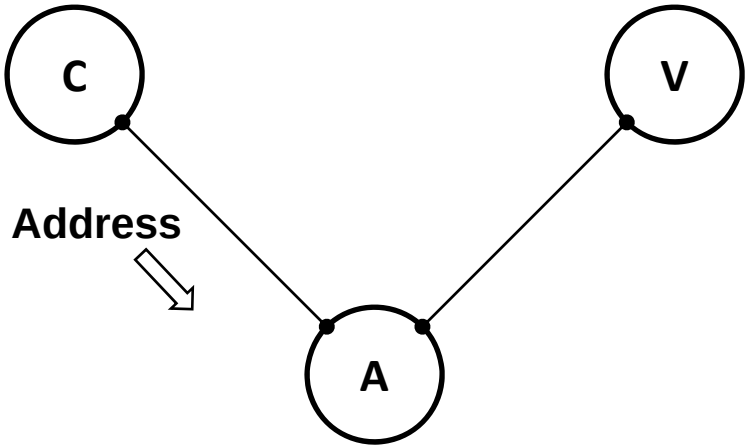



```
s.send(s′); // ….?{ACCEPT: }
```

Delegation Protocol: “Resending”

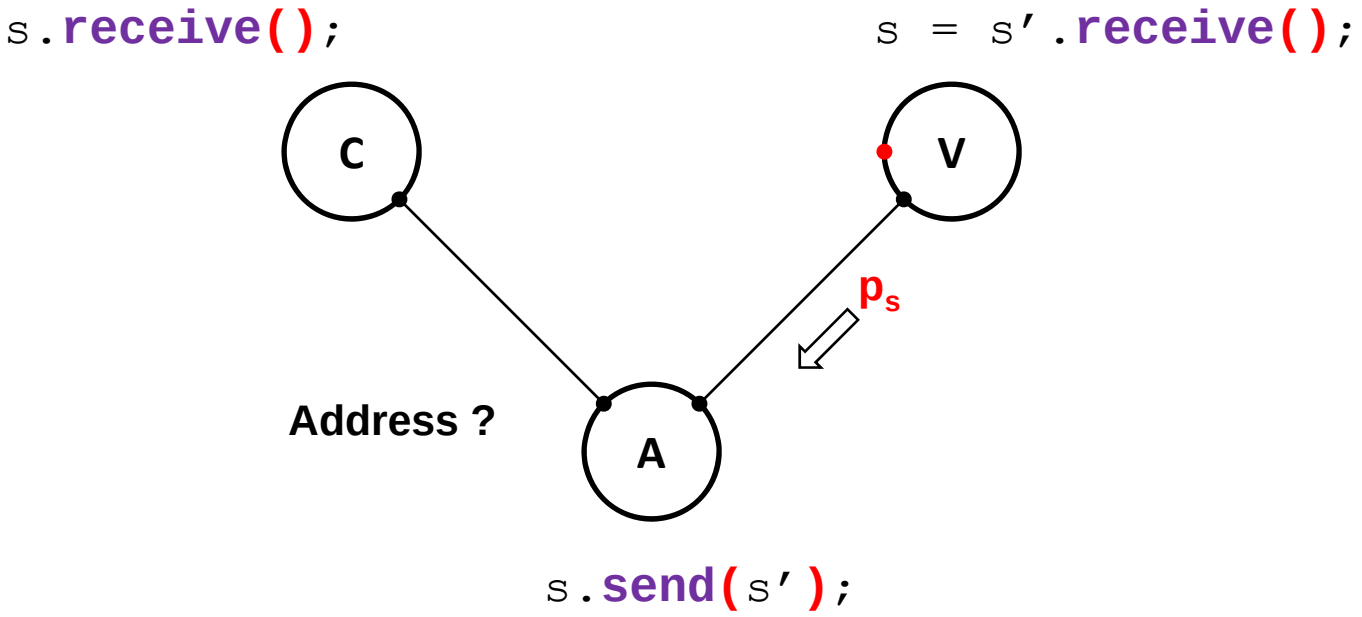
```
// ….!{ACCEPT:!<Address>}
s.receive();
```

```
s = s’.receive();
```

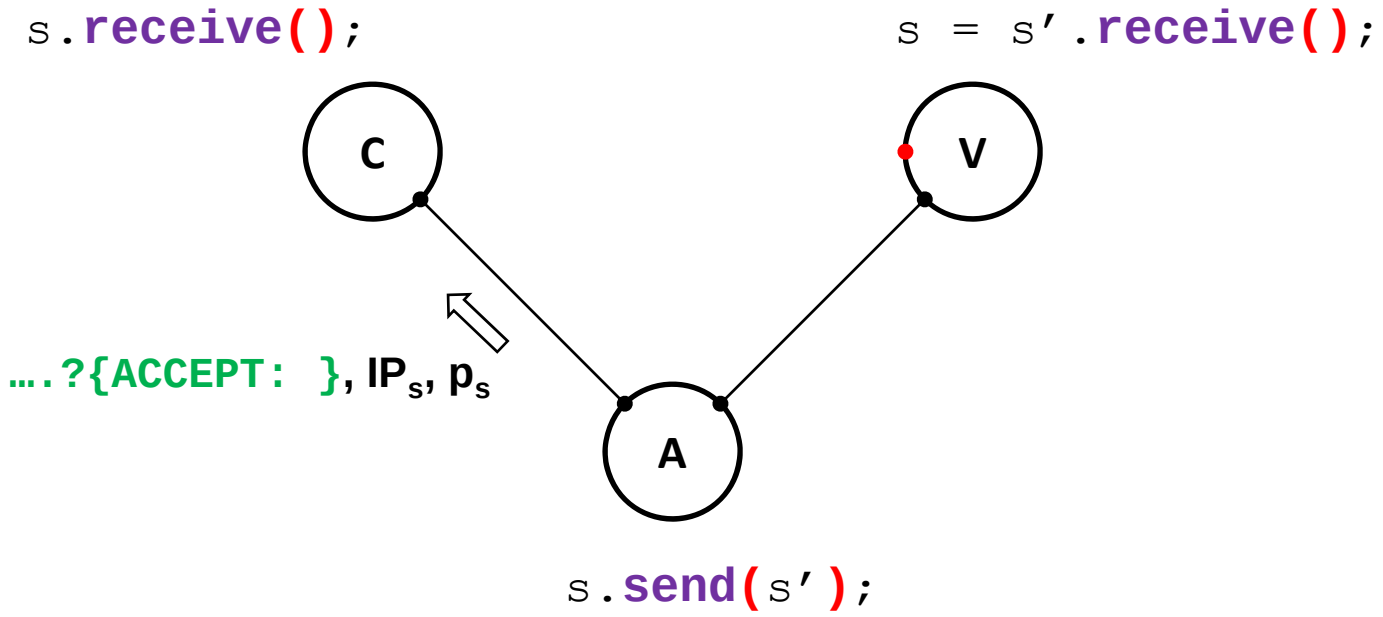



```
s.send(s’);
```

Delegation Protocol: “Resending”

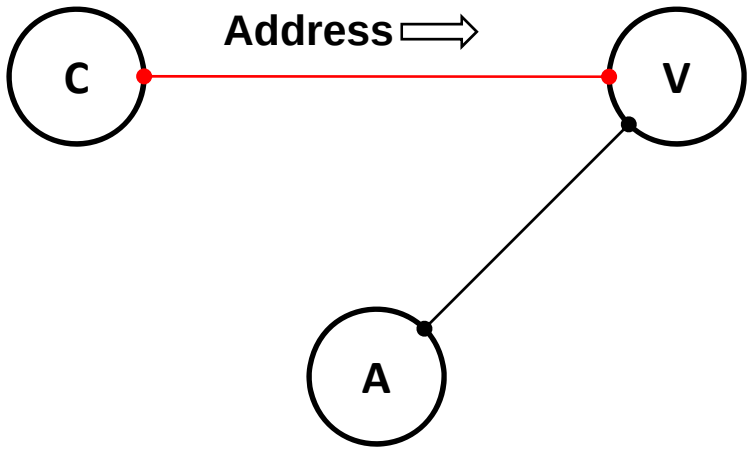


Delegation Protocol: “Resending”



Delegation Protocol: “Resending”

```
s.receive();                    … = s.receive(); // ?(Address)
```

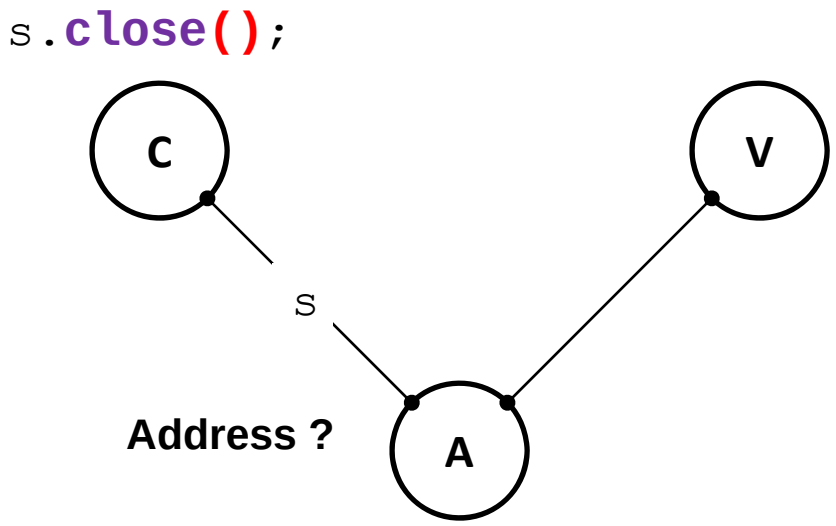


Session type “difference”:

```
….!{ACCEPT:!<Address>} - ….!{ACCEPT: } = !<Address>
```

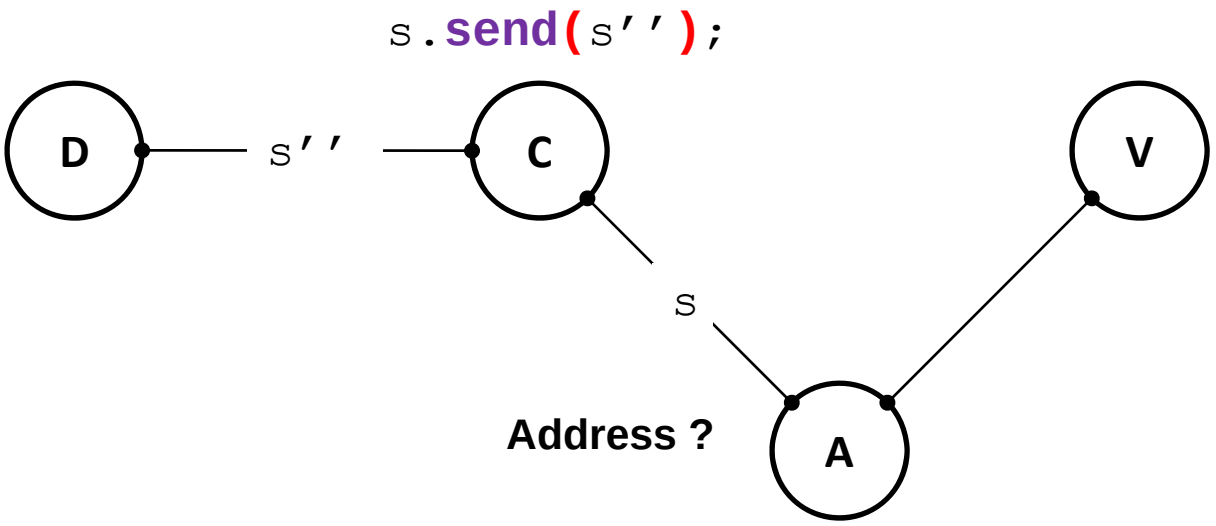
Implementing Session Delegation

- What if Customer has finished its side of the session?



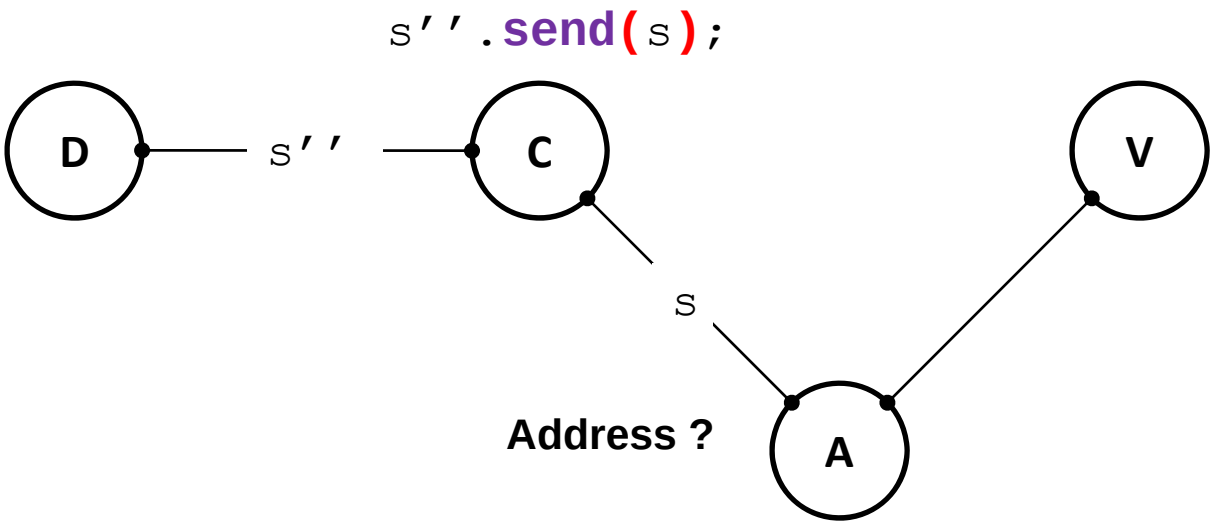
Implementing Session Delegation

- What if Customer is delegating another session to Agency at the “same” time?



Implementing Session Delegation

- What if Customer is delegating the *same* session to another party at the "same" time?



Benchmarks

Implemented client-server application

```
begin . ![ ?(MyObject) ]*
```

Benchmarks

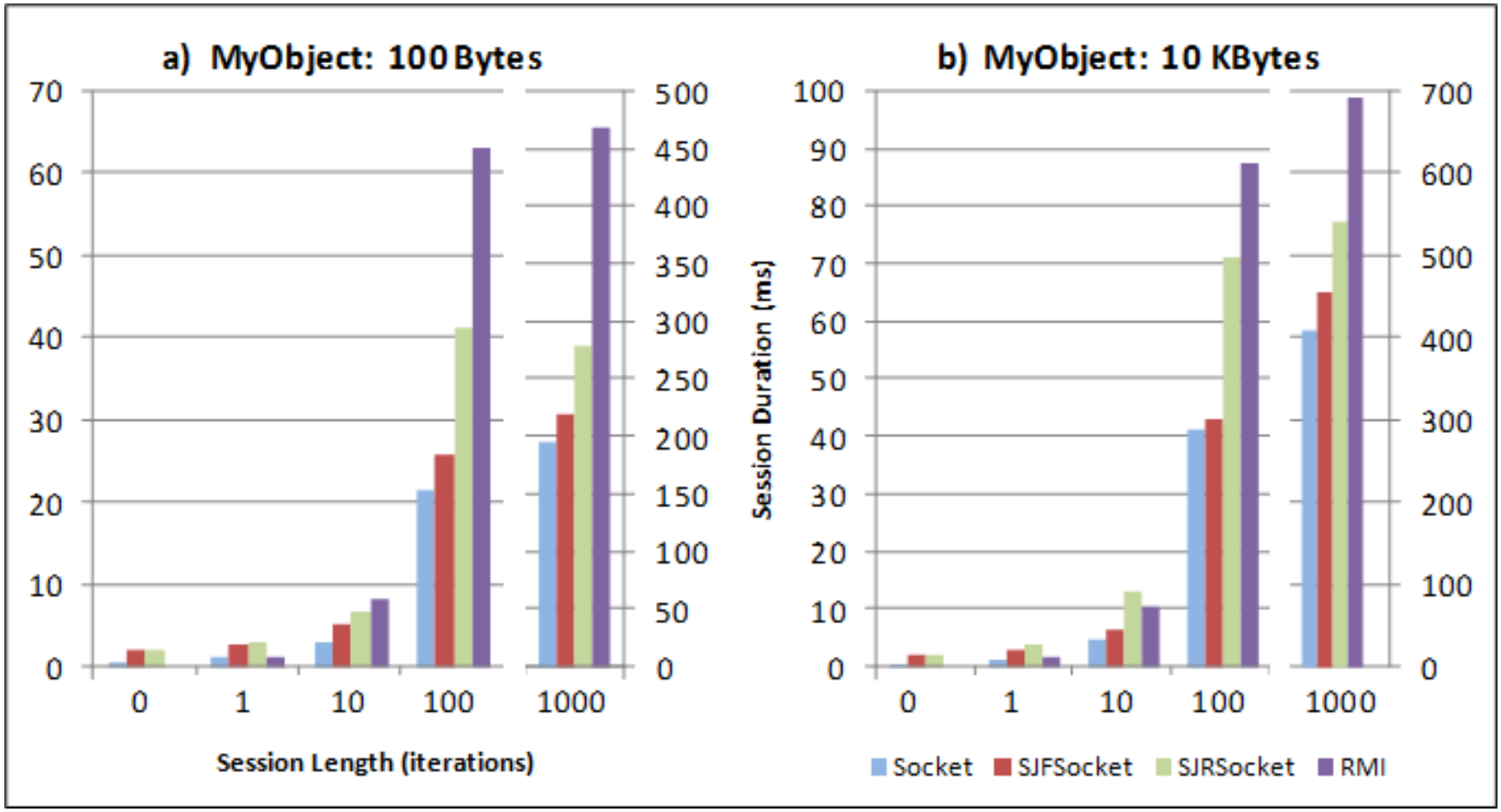
Implemented client-server application

```
begin . ![ ?(MyObject) ]*
```

using

- `Socket(java.net.*)`
- **SJ** Sockets (bounded-forwarding and resending -based)
- RMI: `MyObject remoteMeth(boolean b)`

Benchmarks (Low Latency)



Outline

- Session Types
- Aims
- An Example Application
- The Session Runtime and Benchmarks
- Conclusions and Future Work

Conclusions

- http://www.doc.ic.ac.uk/~rh105/sessionj.html

- Practical, object-oriented language for session programming.

- SJ runtime supports transport-independent session abstraction.

- Asynchronous communication, distributed session initiation, session subtyping, session failure, class downloading, …

- Mechanisms for distributed session delegation.

Related Work: Singularity OS

Sing# Channel Contracts

```
contract NicDevice {
  out message DeviceInfo(…);
  in message RegisterForEvents(
      NicEvents.Exp:READY evchan);
  …
  state IO_CONFIGURE_BEGIN: one {
    RegisterForEvents ? →
      SetParameters ? →
        IO_CONFIGURE_ACK;
  }
  …
  state IO_CONFIGURE_ACK: one {
    InvalidParameters ! →
      IO_CONFIGURE_BEGIN;
    Success ! → IO_CONFIGURED;
  }
  …
```

Session Types

```
class DeviceInfo { … }

…

protocol NicDevice {
  begin.
  …
  rec IO_CONFIGURE_BEGIN[
    ?(@NicEvents_Exp).
    ?(SetParameters).
    !{
      InvalidParameters: #IO_CONF…,
      Success:
    }
  ]
  . …
```

Future Work

- Further features and language extensions, e.g. code generation, session exceptions, parallel sub-sessions, multiparty sessions, session-typed service directories, …
- Integration of additional transports under session abstraction, e.g. shared memory, named pipes, … multiplexing, …
- Session types for runtime optimisation, e.g. message batching, asynchronous send promotion, buffer sizes, …
- W3C Web services Choreography

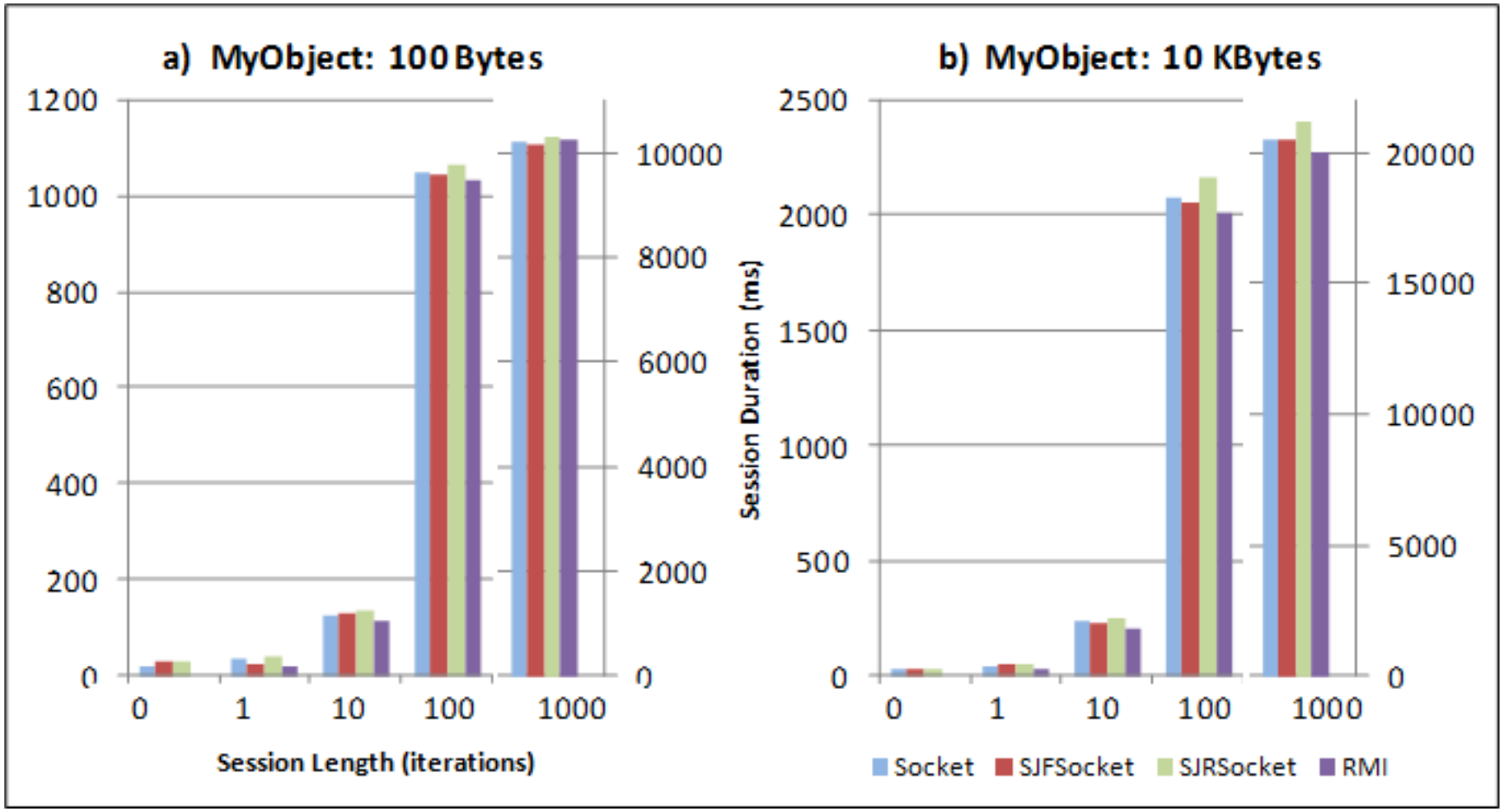
Examples: Chat Server

```
protocol p_recmsgs {
  begin.?[?(String)]*
}
protocol p_sendmsgs {
  begin.?[!<String>]*
}

protocol p_events {
  rec X[
    ?{
      PING:
        ![
          !{
            USER_JOINED:
              !<int>.!<String>,
            USER_LEFT:
              !<int>
          }
        ]*
      .#X,
      BYE:

    }
  ]
}

protocol p_control {
  begin
  .!<int>.?(String)
  .?(@p_recmsgs)
  .?(@p_sendmsgs)
  .@p_events
}
```

Examples: File Server

```
protocol p_binsend {
  ![!<byte[]>]*
}

protocol p_binrecv {
  [?(byte[])]*
}

protocol p_control {
  ?[
    ?{
      GET:
        ?(String)
        .?{
          ACTIVE:
            ?(begin.@p_binsend),
          PASSIVE:
            !<begin.@p_binrecv>
        },
      PUT:
        ?(String)
        .?{
          ACTIVE:
            ?(begin.@p_binrecv),
          PASSIVE:
            !<begin.@p_binsend>
        }
    }
  ]*
}

protocol p_rftp {
  begin
  .?(String).?(String)
  .!{
    LOGIN_OK: @p_control,
    LOGIN_FAIL:
  }
}
```

Benchmarks (Higher Latency)



Implementing `Customer` (6)

```
                                        // Session type checking.
s.request();
s.outwhile(…) {
  s.send("PARIS/EUROSTAR");
  cost = s.receive();
}
if (…) {
  s.outbranch(ACCEPT) {
    s.send(…);
    date = s.receive();
  }
} else {
  s.outbranch(REJECT) { }
}
```

Implementing `Customer` (6)

```
s.request();
s.outwhile(…) {
  s.send("PARIS/EUROSTAR");
  cost = s.receive();
}
if (…) {
  s.outbranch(ACCEPT) {
    s.send(…);
    date = s.receive();
  }
} else {
  s.outbranch(REJECT) { }
}
```

```
// Session type checking.
begin.
![
  !<String>

]*

!{ACCEPT:
  !<Address>

}

!{REJECT: }
```

Implementing `Customer` (6)

```
s.request();
s.outwhile(…) {
  s.send("PARIS/EUROSTAR");
  cost = s.receive();
}
if (…) {
  s.outbranch(ACCEPT) {
    s.send(…);
    date = s.receive();
  }
} else {
  s.outbranch(REJECT) { }
}
```

```
// Session type checking.
begin
![
  !<String>.
  ?(…)
]*

!{ACCEPT:
  !<Address>.
  ?(…)
}

!{REJECT: }
```

Implementing `Customer` (6)

```
s.request();
s.outwhile(…) {
  s.send("PARIS/EUROSTAR");
  cost = s.receive();
}
if (…) {
  s.outbranch(ACCEPT) {
    s.send(…);
    date = s.receive();
  }
} else {
  s.outbranch(REJECT) { }
}
```

```
// Session type checking.
begin.
![
  !<String>.
  ?(int)
]*

!{ACCEPT:
  !<Address>.
  ?(Date)
}

!{REJECT: }
```

Implementing `Customer` (6)

```
s.request();
s.outwhile(…) {
  s.send("PARIS/EUROSTAR");
  cost = s.receive();
}
if (…) {
  s.outbranch(ACCEPT) {
    s.send(…);
    date = s.receive();
  }
} else {
  s.outbranch(REJECT) { }
}
```

```
// Session type checking.
begin.
![
  !<String>.
  ?(int)
]*.
!{
   ACCEPT:
     !<Address>.
     ?(Date),

   REJECT:
}
```

Implementing `Customer` (6)

```
                                        // Linear endpoint usage.
s.request();
s.outwhile(…) {
  s.send("PARIS/EUROSTAR");
  cost = s.receive();
}
if (…) {
  s.outbranch(ACCEPT) {
    s.send(…);
    date = s.receive();
  }
} else {
  s.outbranch(REJECT) { }
}
```