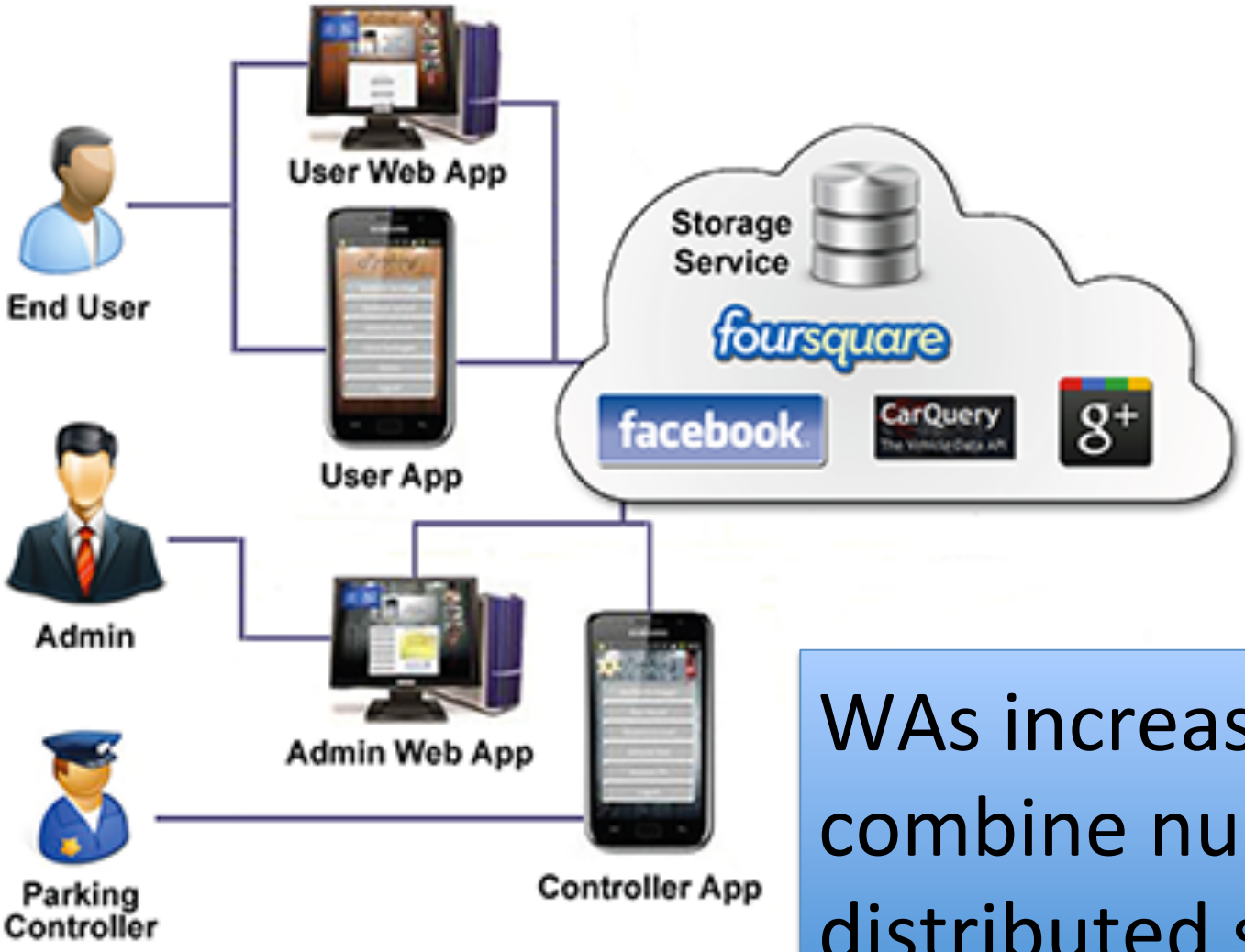
Presentation of Paper:

Session-Based Distributed Programming in Java

Martin Wolf
<mawo@martinwolf.eu>

Background and Motivations



WAs increasingly combine numerous distributed services.

Problem

```
os = new DataOutputStream(
        smtpSocket.getOutputStream()
);
os.writeBytes("HELO\n");
```

Background and Motivations: Abstractions

- Conversation: structured sequence of communications.
- Session: private channel.
- Session type: specification of the structure and message types.

Goal:

Demonstrate the impact of integrating session types into object-oriented programming in Java.

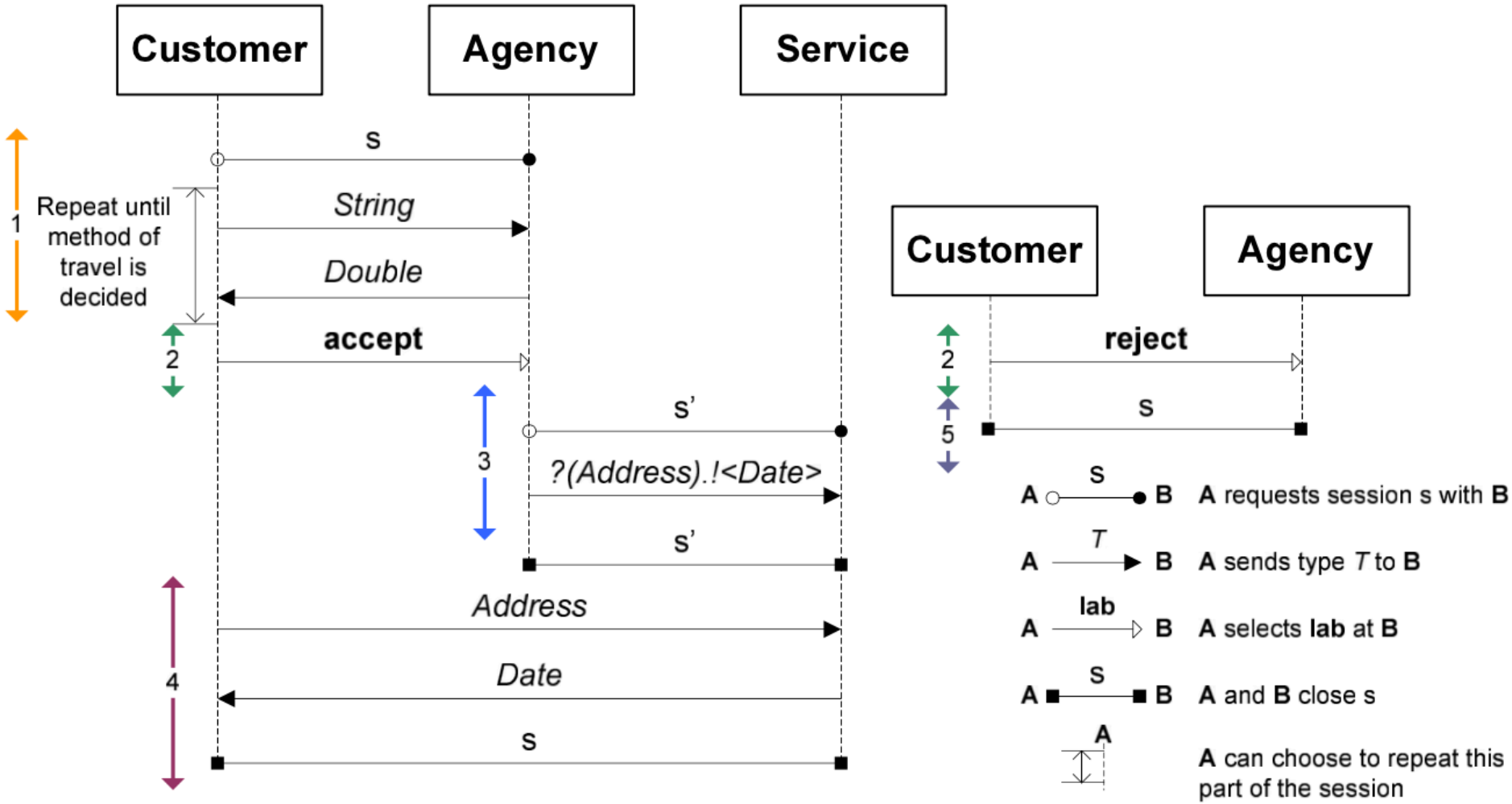
Preceding Works

- Theoretical studies of session types in object oriented core calculi.
- The implementation of a system-level object oriented language with session types for shared memory concurrency.

Contributions

- A compilation-runtime framework with features:
  - Integration of object-oriented and session programming disciplines.
  - Ensuring communication safety for distributed applications.
  - Supporting session abstraction over concrete transports.

Session-Based Programming Example



Session-Based Programming Example

```
protocol placeOrder {
  begin. // Commence session.
  ![     // Can iterate:
     !<String>. // send String
     ?(Double)  // receive Double
  ]*.
  !{    // Select one of:
     ACCEPT: !<Address>.?(Date),
     REJECT:
  }
}
```

Order protocol: Customer side.

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

Order protocol: Agency side.

Session-Based Programming Example

- Session Sockets and Communication
  - Session server socket(SJServerSocket)
    - SJServerSocketImpl.create(acceptOrder, port)
    - accept()
    - send/receive
  - Session server-address(SJServerAddress)
    - c_ca = SJServerAddress.create(placeOrder, host, port)
  - Session socket(SJSocket)
    - SJSocketImpl.create(c_ca)
    - request()
    - send/receive

Session-Based Programming Example: Iteration

```
boolean decided = false;
... // Set journey details.
s_ca.outwhile(!decided) {
  s_ca.send(journeyDetails);
  Double cost = agency.receive();
  ... // Set decided to true or
  ... // change details and retry
}
```

```
s_ac.inwhile() {
  String journeyDetails
        = s_ac.receive();
  ... // Calculate the cost.
  s_ac.send(price);
}
```

Session-Based Programming Example: Branching

```
if(want to place an order) {
  s_ca.outbranch(ACCEPT) {
    s_ca.send(address);
    Date dispatchDate = s_ca.receive();
  }
} else { // Don't want to order.
  s_ca.outbranch(REJECT) { }
}
```

```
s_ac.inbranch() {
  case ACCEPT: {
    ...
  }
  case REJECT: { }
}
```

Session-Based Programming Example: Handling Session Failures

```
try (s_ac, ...) {
  ... // Implementation of session 's_ac' and others.
} catch (SJIncompatibleSessionException ise) {
  ... // One of the above sessions could not be initiated.
} catch (SJIOException ioe) {
  ... // I/O error on any of the above sessions.
} finally { ... } // Optional.
```

Session-Based Programming Example: Session Delegation

```
protocol delegateOrderSession {
  begin.!<?(Address).!<Date>>
}
```

```
protocol receiveOrderSession {
  begin.?(?(Address).!<Date>)
}
```

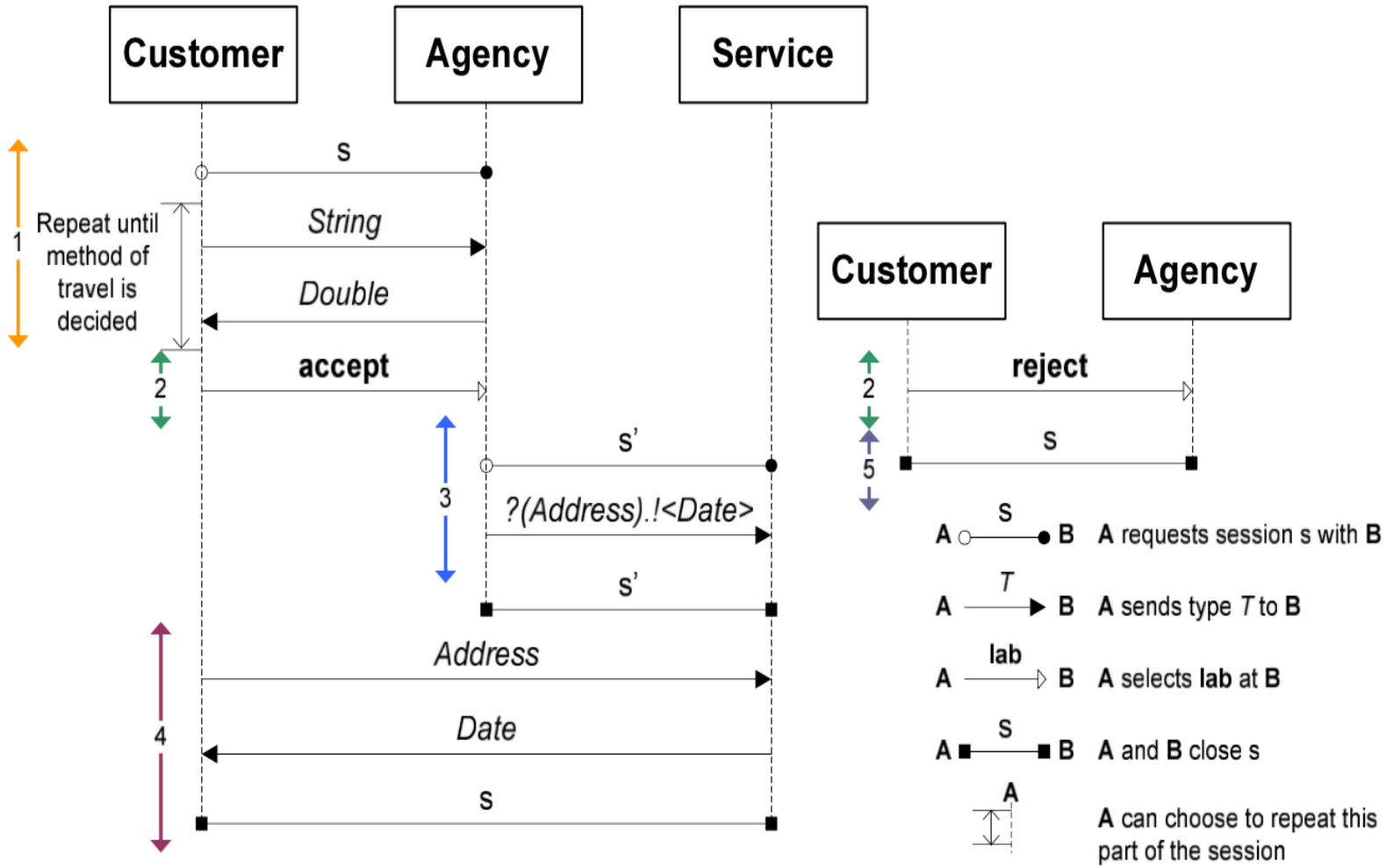
```
case ACCEPT: {
  SJServerAddress c_as = ... // delegateOrderSession.
  SJSocket s_as = SJSocketImpl.create(c_as);
  s_as.request();
  s_as.send(s_ac); // Delegation: Agency has finished with s_ac.
}
```

```
SJServerSocket ss_sa = SJServerSocketImpl.create(..., port)
SJSocket s_sa = ss_sa.accept();
SJSocket s_sc = s_sa.receive(); // Receive the delegated session.
```

```
Address custAddr = s_sc.receive();
Date dispatchDate = ... // Calculate dispatch date.
s_sc.send(dispatchDate);
```

Mechanisms for Session Delegation

- Indefinite redirection
- Direct reconnection
  - Resending protocol
  - Forwarding protocol


Customer
Agency
Service
s
String
Double
accept
s'
?(Address).!<Date>
s'
Address
Date
s
1
Repeat until method of travel is decided
2
3
4
Customer
Agency
reject
s
2
5
s
A B A requests session s with B
T
A B A sends type T to B
lab
A B A selects lab at B
s
A B A and B close s
A
A can choose to repeat this part of the session

Compiler and Runtime Architecture: Layers

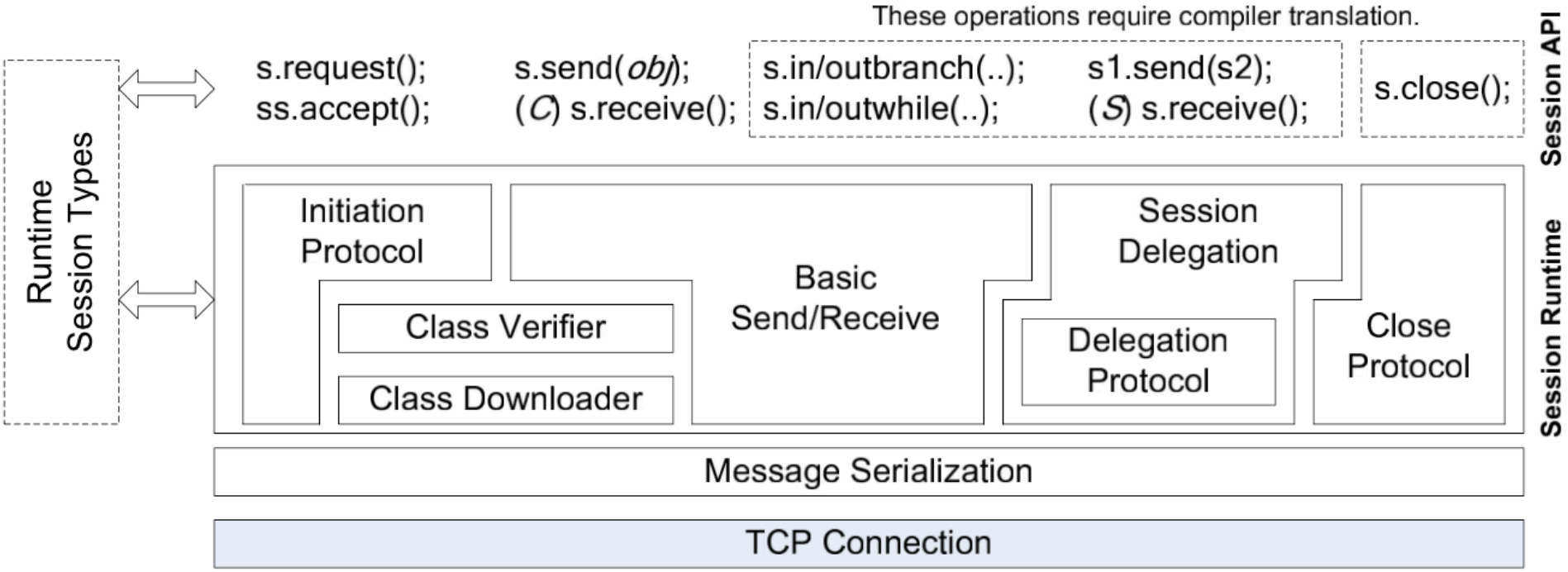
- Layer 1
  - SJ source code.
- Layer 2
  - Java translation and session runtime APIs.
- Layer 3
  - Runtime execution: JVM and SJ libraries.

Compiler and Runtime Architecture: Compiler Type Checking Example

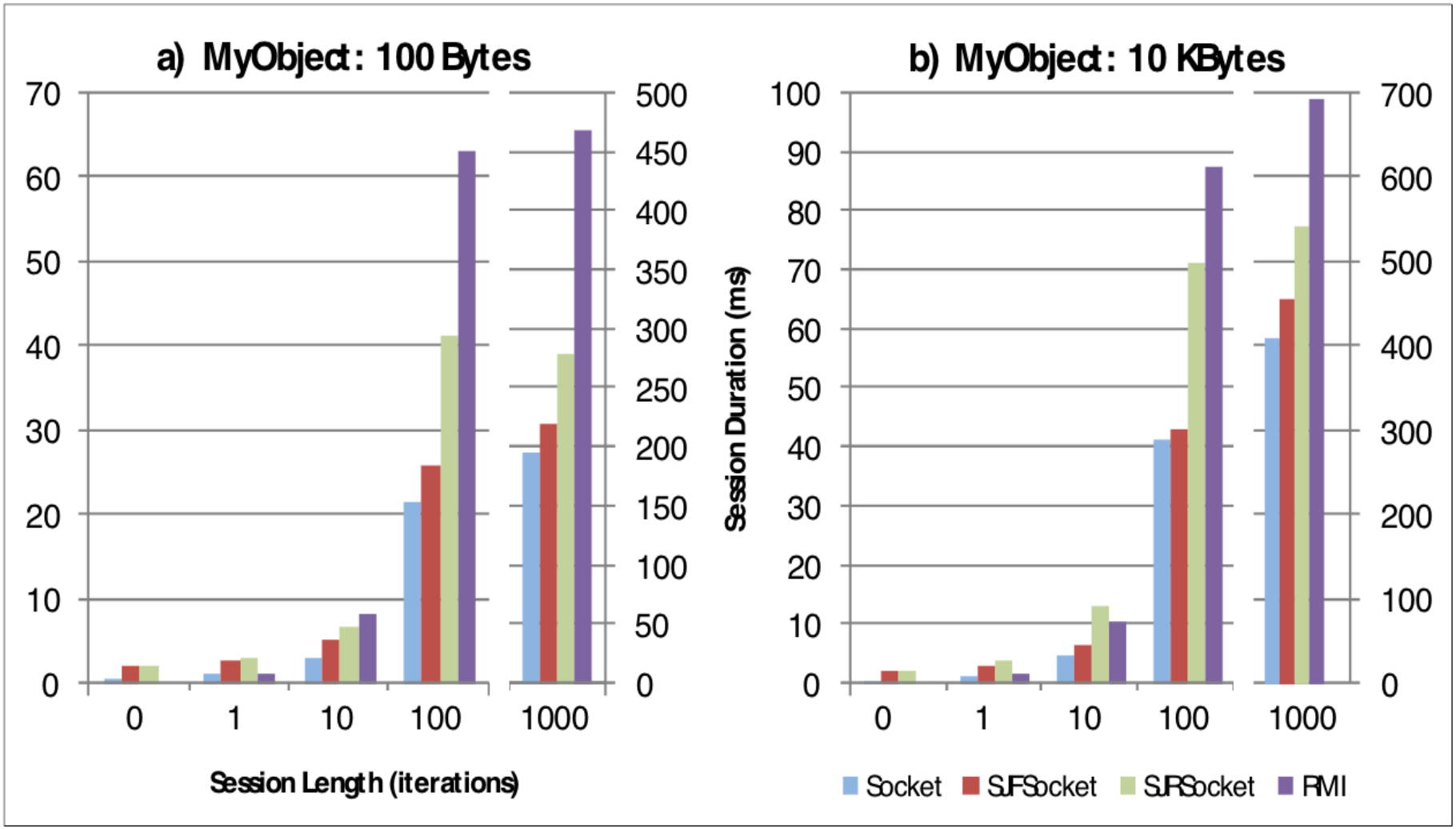
```
protocol thirstyPerson {
  begin.!{
     COFFEE: !<Euros>,
     TEA: !<Euros>
  }
}
...
if (...) { // Implemented...
  s.outbranch(COFFEE) { ... };
}
else { // ...a coffee addict.
  s.outbranch(COFFEE) { ... };
}
```

```
protocol vendingMachine {
  begin.?{
     COFFEE: ?(Money),
     TEA: ?(Money),
     CHOCOLATE: ?(Money)
  }
}
...
s.inbranch() {
  case COFFEE: { ... }
  case TEA: { ... }
  case CHOCOLATE: { ... }
}
```

Compiler and Runtime Architecture



SJ Runtime Performance



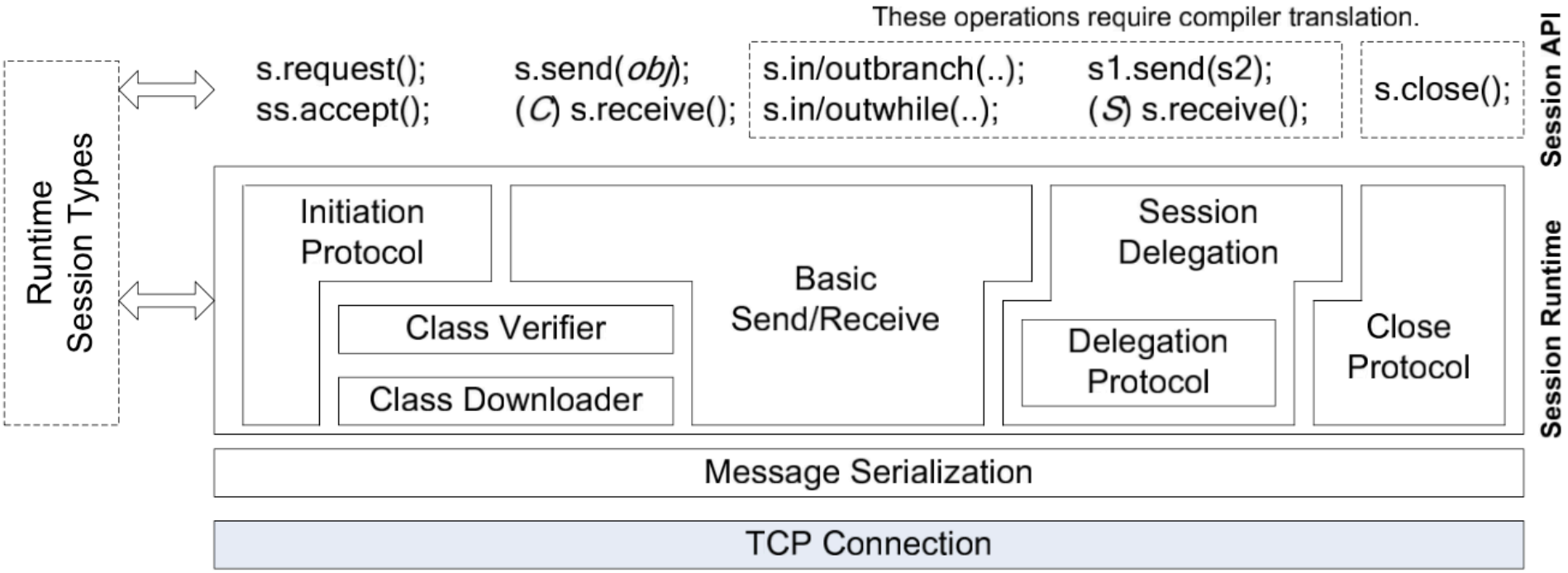
Protocol: `begin.?[!<MyObject>]*`

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

Conclusion and Future Work

- Benchmarks show the feasibility of session based communication and the session runtime architecture.
- Future work:
  - Optimizations that utilize session type information.
  - Incorporation of transports other than TCP into the session runtime.
  - Dynamically handle deeper security issues.

Q&A


These operations require compiler translation.
Runtime Session Types
s.request();
ss.accept();
s.send(obj);
(C) s.receive();
s.in/outbranch(..);
s.in/outwhile(..);
s1.send(s2);
(S) s.receive();
s.close();
Session API
Initiation Protocol
Class Verifier
Class Downloader
Basic Send/Receive
Session Delegation
Delegation Protocol
Close Protocol
Session Runtime
Message Serialization
TCP Connection

References

- Figures borrowed from article:
  - Raymond Hu, Nobuko Yoshida, and Kohei Honda. 2008. Session-Based Distributed Programming in Java.
- Parking system example:
  - http://dsgwords.blogspot.dk/2013/07/collaborative-mobile-application-and.html