Web Services

# SOAP

Lecture "XML in Communication Systems"
Chapter 12

Prof. Dr.-Ing. Norbert Luttenberger
Research Group for Communication Systems
Dept. of Computer Science
Christian-Albrechts-University in Kiel

## Recommended Reading

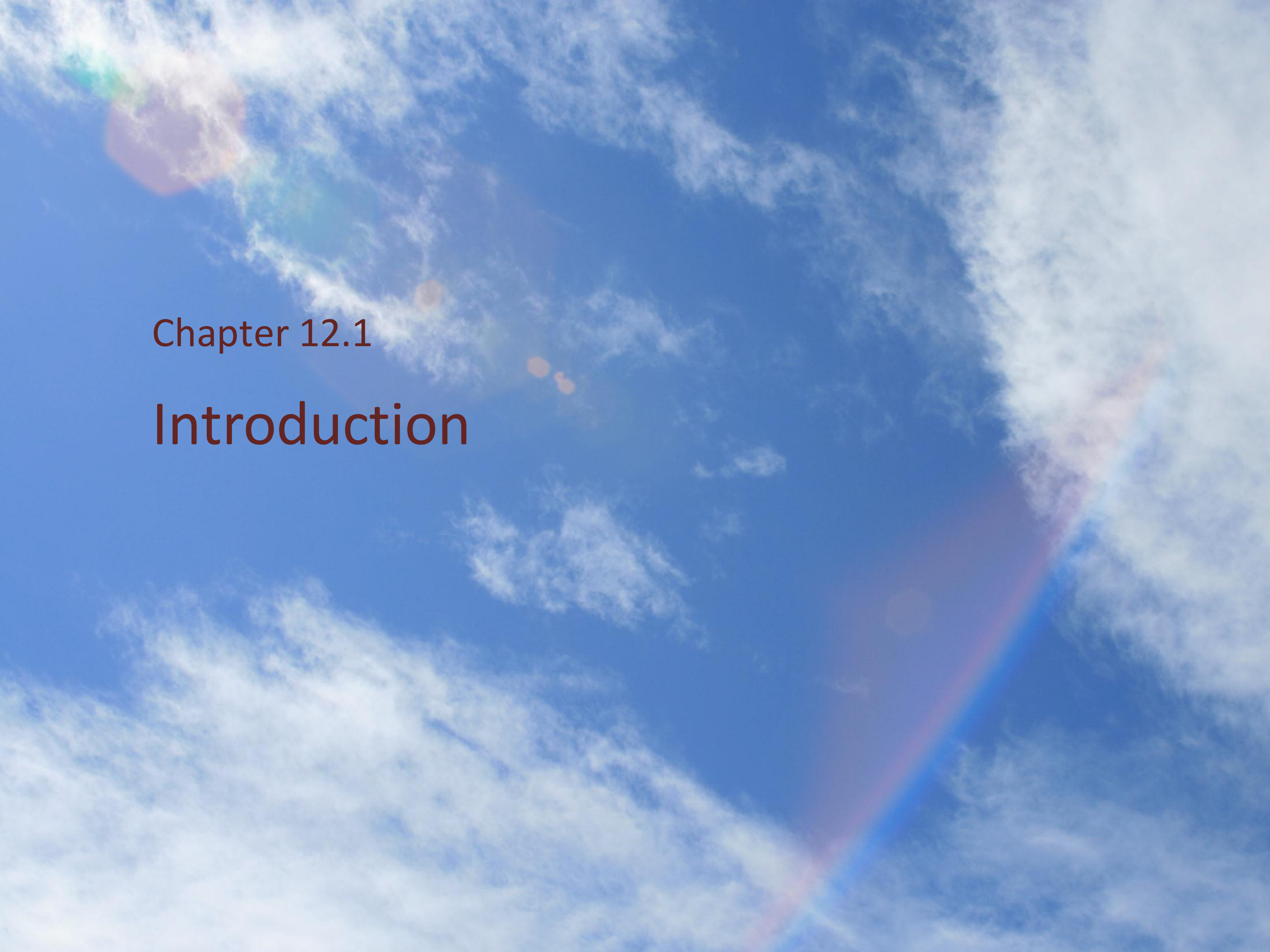
- Booth, D. et al.:
  Web Services Architecture
  http://www.w3.org/TR/ws-arch/

- Nilo Mitra, Yves Lafon (eds.):
  SOAP Version 1.2 Part 0: Primer (Second Edition)
  W3C Recommendation 27 April 2007
  http://www.w3.org/TR/soap12-part0/

- Ballinger, K., Ehnebuske, D., Gudgin, M., Nottingham, M., Yendluri, P.:
  Web Services Interoperability Organization Basic Profile Version 1.0
  http://www.ws-i.org/Profiles/BasicProfile-1.0.html

- Papazoglou, M.:
  Web Services: Principles and Technology.
  Pearson Education Ltd. 2008

## Overview



Chapter 12.1

# Introduction

## Introduction

- Former name: *Simple Object Access Protocol*
  - protocol for exchange of structured and typed data
  - between peers in a distributed environment
  - features:
    - XML message coding
    - stateless request/response communication
    - SOAP endpoints identified by URL
    - SOAP binding: definitions for transport of SOAP messages over different Internet protocols

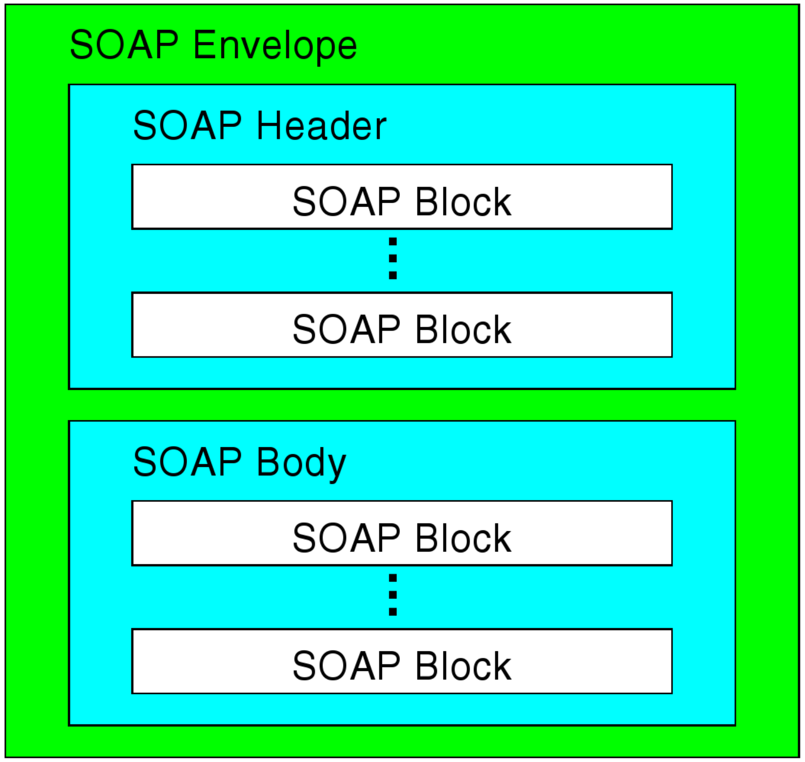
## Introduction

- Former name: *Simple Object Access Protocol* (cont'd.)
  - No explicit programming model, unlike DCOM and CORBA: no special components or tools needed to make an implementation
  - Can be implemented in any language and OS (Java, Perl, C++, VB, Windows, UNIX)
  - SOAP defines two types of messages:
    - Requests
    - Responses

## Introduction

- Former name: *Simple Object Access Protocol* (cont'd.)
  - versions: SOAP 1.1 (2000) and SOAP 1.2 (2003)
  - namespaces for SOAP 1.1
    - http://schemas.xmlsoap.org/soap/envelope/
    - http://schemas.xmlsoap.org/soap/encoding/
  - namespaces for SOAP 1.2
    - http://www.w3.org/2003/05/soap-envelope
    - http://www.w3.org/2003/05/soap-encoding
    - http://www.w3.org/2003/05/soap-rpc
    - …

## Introduction



elements

- `Envelope`
- `Header`
- `Body`
- `Fault`

attributes

- `actor` (SOAP 1.1) / `role` (SOAP 1.2)
- `mustUnderstand`
- `encodingStyle`
- `relay` (SOAP 1.2)

# Introduction

- Sample SOAP message (from the SOAP 1.2 standard)

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    <m:reservation
        xmlns:m="http://travelcompany.example.org/reservation"
        env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
        env:mustUnderstand="true">
      <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</m:reference>
      <m:dateAndTime>2001-11-29T13:20:00.000-05:00</m:dateAndTime>
    </m:reservation>
    <n:passenger
        xmlns:n="http://mycompany.example.com/employees"
        env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
        env:mustUnderstand="true">
      <n:name>Åke Jógvan Øyvind</n:name>
    </n:passenger>
  </env:Header>
```

## Introduction

- Sample SOAP message (from the SOAP 1.2 standard)

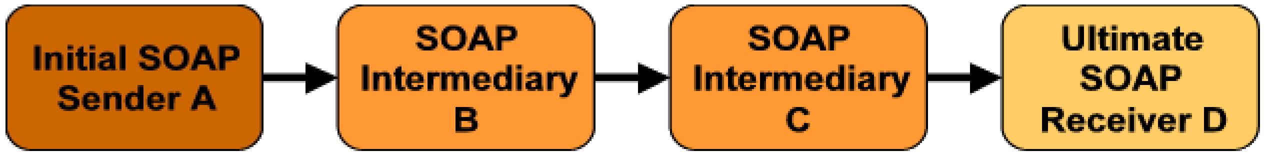
```
  <env:Body>
    <p:itinerary
      xmlns:p="http://travelcompany.example.org/reservation/travel">
      <p:departure>
        <p:departing>New York</p:departing>
        <p:arriving>Los Angeles</p:arriving>
        <p:departureDate>2001-12-14</p:departureDate>
        <p:departureTime>late afternoon</p:departureTime>
        <p:seatPreference>aisle</p:seatPreference>
      </p:departure>
      <p:return>  …  </p:return>
    </p:itinerary>
    <q:lodging
      xmlns:q="http://travelcompany.example.org/reservation/hotels">
      <q:preference>none</q:preference>
    </q:lodging>
  </env:Body>
</env:Envelope>
```

Chapter 12.2

# SOAP Processing

# SOAP Processing

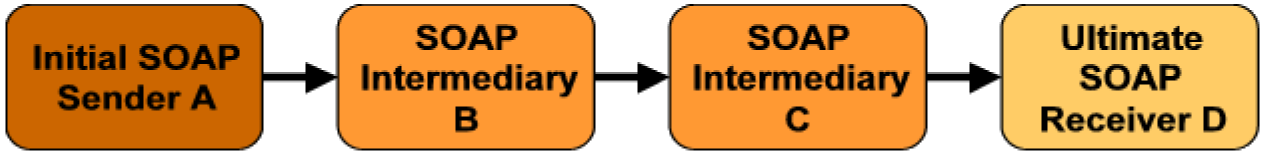
- SOAP Nodes
  - are sources and sinks of SOAP (request/response) messages, SOAP faults
  - different types of SOAP nodes
    - Initial SOAP sender
    - Ultimate SOAP Receiver
    - SOAP Intermediary



Henrik Frystyk Nielsen: SOAP Message Path Modeling.
W3C Workshop on Web Services, 11-12 April 2001, San Jose, CA - USA

# SOAP Processing

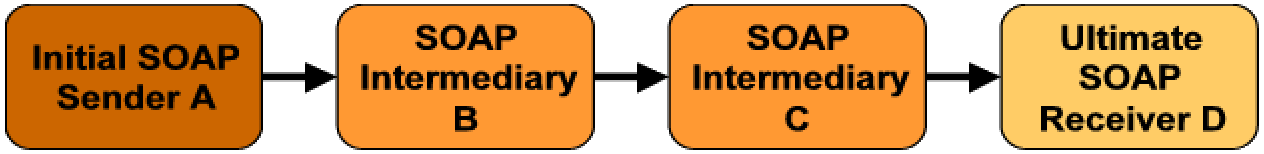
- Message path



  – "A SOAP message generated by A can indicate which part of a message is for B, C and D. However, it cannot indicate that the intermediaries are to be organized into a message path … "
  – "The SOAP targeting model is … a decentralized message-processing concept rather than a mechanism for transferring SOAP messages across the Web."

## SOAP Processing

- SOAP Intermediaries



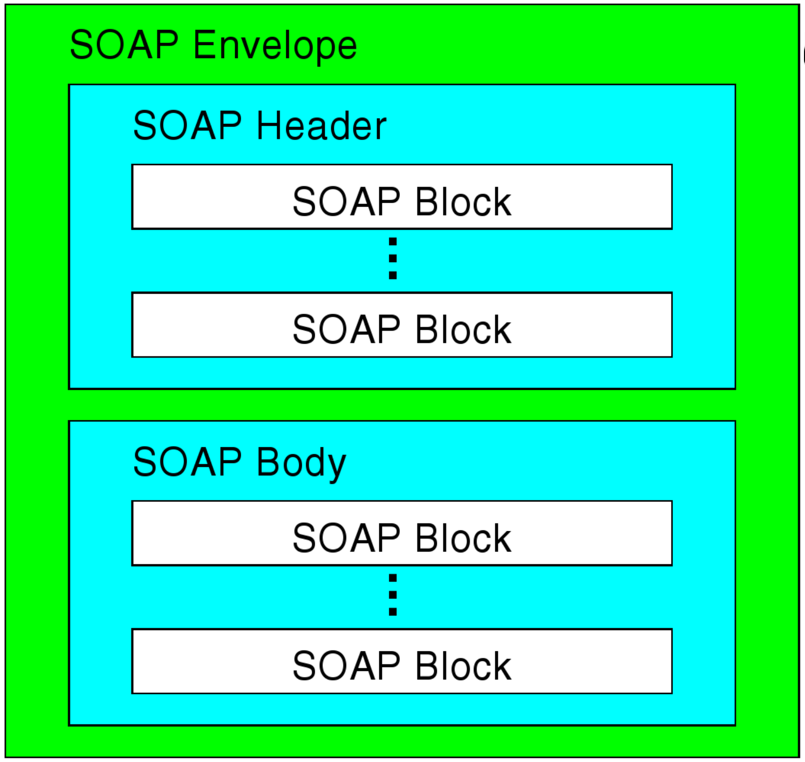
  - processing intermediaries deal with the message using the same rules as any other SOAP receiver and/or sender
  - protocol intermediaries don't take any part in the SOAP message path other than acting as a relay at the underlying protocol level.
  - assignment of roles to SOAP nodes

## SOAP Processing

- An aside: RFC-2616 (HTTP 1.1, 1999) on other kinds of intermediaries
  - A proxy is a forwarding agent, receiving requests for a URI in its absolute form, rewriting all or part of the message, and forwarding the reformatted request toward the server identified by the URI.
  - A gateway is a receiving agent, acting as a layer above some other server(s) and, if necessary, translating the requests to the underlying server's protocol.
  - A tunnel acts as a relay point between two connections without changing the messages; tunnels are used when the communication needs to pass through an intermediary (such as a firewall) even when the intermediary cannot understand the contents of the messages.

## SOAP Processing

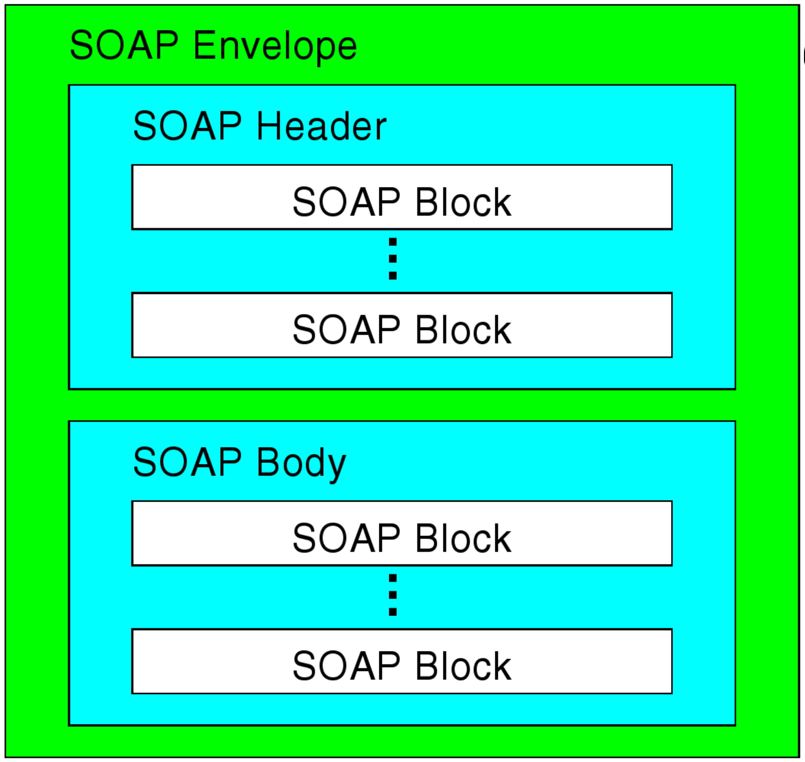
- Header block processing



- determined by the "role" a SOAP node adopts
- SOAP Header blocks may carry a related
  - `env:actor` (SOAP 1.1) resp.
  - `env:role` (SOAP 1.2) attribute (optional)

## SOAP Processing

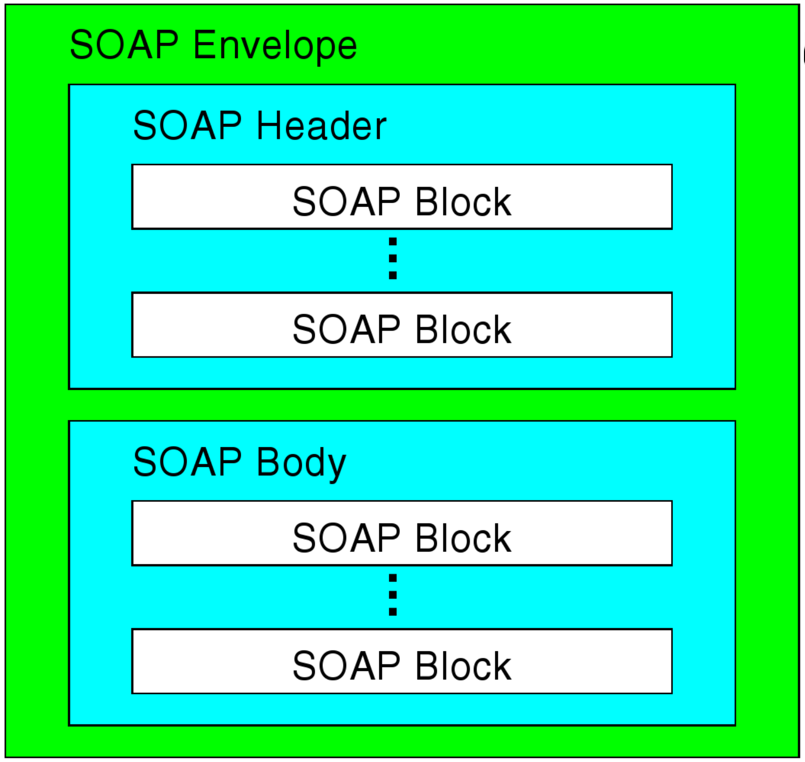
- Header block processing (cont'd.)

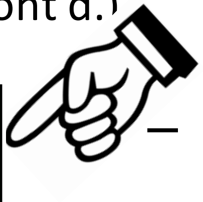


  - predefined roles (SOAP 1.2):
    - `next`
      for both all SOAP Intermediaries and the Ultimate SOAP Receiver
    - `none`
      no node in this message path
    - `ultimateReceiver`
      for the Ultimate SOAP Receiver

# SOAP Processing

- Header block processing (cont'd.)

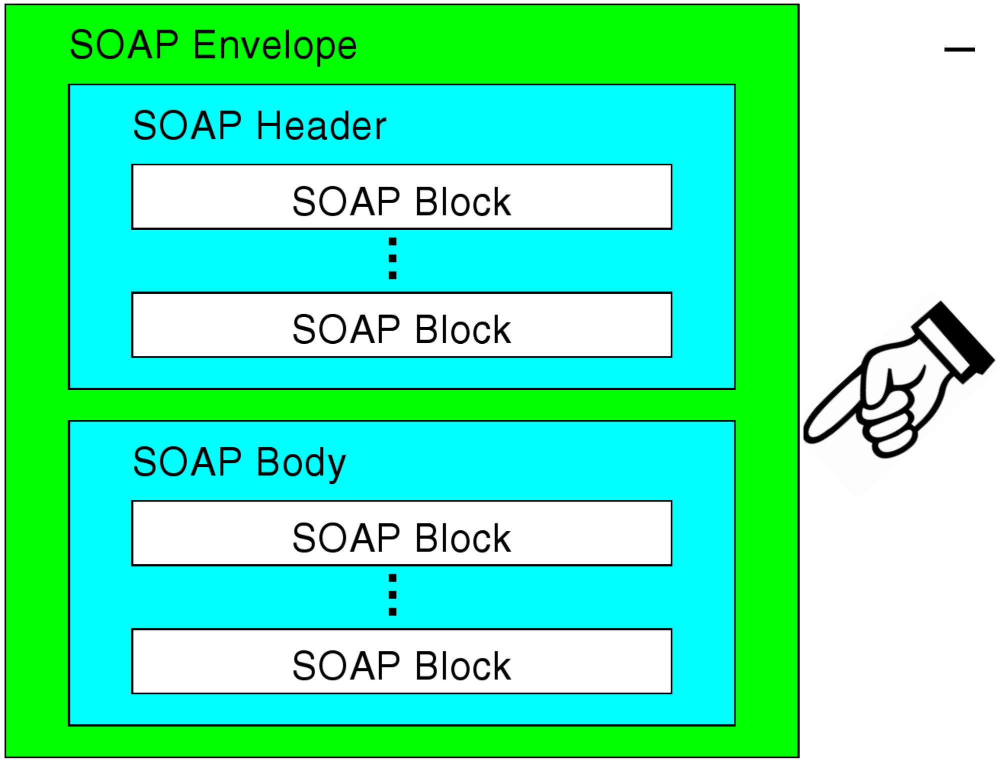




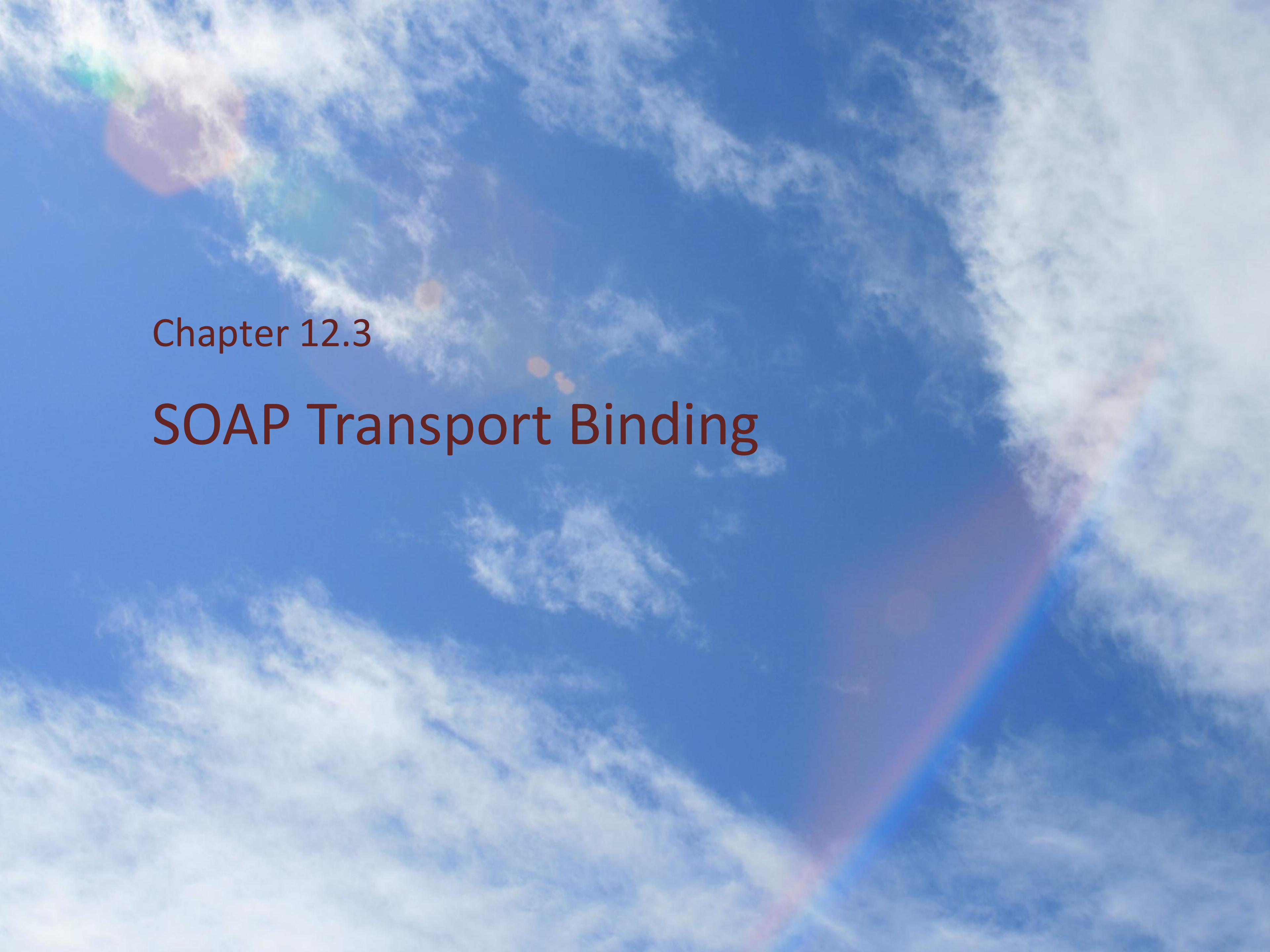
  - Optional attribute `env:mustUnderstand` → block must be processed!
    - SOAP fault has to be sent, if block cannot be processed.

# SOAP Processing

- Body block processing



  - Ultimate SOAP Receiver must process SOAP Body.



Chapter 12.3

# SOAP Transport Binding

## SOAP Transport over HTTP

- SOAP HTTP Binding
  - SOAP in Request-Response mode
    - HTTP Request carries SOAP Request and
    - HTTP Response carries SOAP Response.
  - HTTP features
    - method: POST
    - Content-Type: text/xml (SOAP 1.1) or application/soap+xml (SOAP 1.2)
    - SOAP 1.1: additional HTTP header line `SOAPAction` or SOAP 1.2: `action` parameter in the Content-Type (RFC-3902)

# SOAP Transport over HTTP

```
POST /Service/Stockquotes HTTP/1.0
Content-Type: text/xml
SOAPAction: "urn:getQuote"

<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
   <env:Body>
      <ns:getQuote xmlns:ns="urn:example-delayed-quotes">
         <ns:symbol>NYSE:IBM</ns:symbol>
      </ns:getQuote>
   </env:Body>
</env:Envelope>
```

## SOAP Transport over HTTP

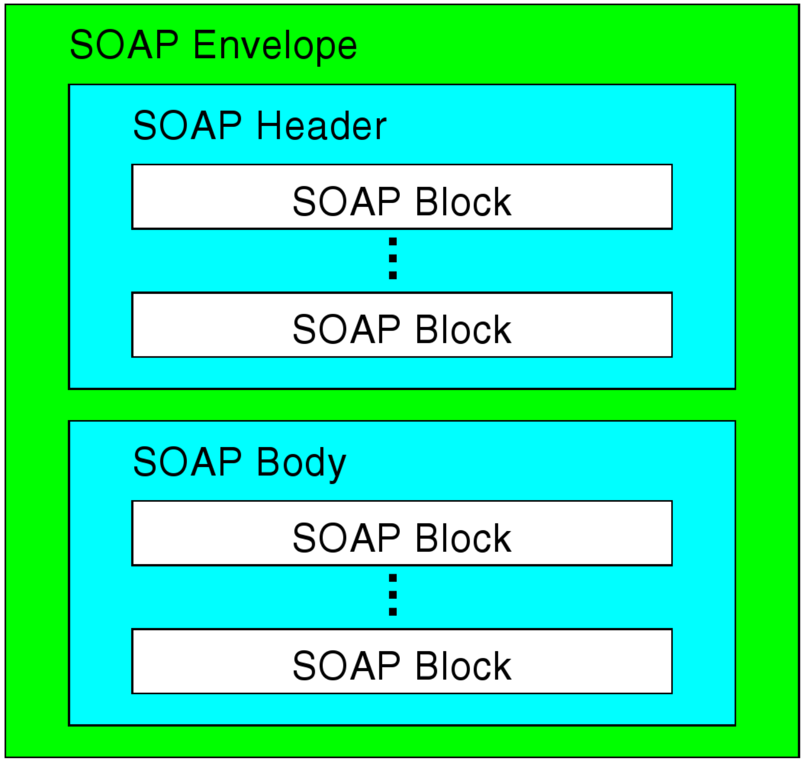
```
HTTP/1.0 200 OK
Content-Type: text/xml

<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
   <env:Body>
      <ns:getQuoteResponse xmlns:ns="urn:example-delayed-quotes">
         <ns:result>91.35</ns:result>
      </ns:getQuoteResponse>
   </env:Body>
</env:Envelope>
```

Chapter 12.4

# Message Formats

# Message Formats



elements

- `Envelope`
- `Header`
- `Body`
- `Fault`

attributes

- `actor` (SOAP 1.1) / `role` (SOAP 1.2)
- `mustUnderstand`
- `encodingStyle`
- `relay` (SOAP 1.2)

## Message Formats

- SOAP header
  - is an optional component
  - has information about how the message is to be processed
  - can contain extensions to the message like transaction ids
  - can also contain security information
- SOAP body
  - contains the message referred to as "payload"
  - can contain the `encodingStyle` attribute
  - can also contain a `<Fault>` element

## SOAP Message Format

- **SOAP message**

  A SOAP message is the basic unit of communication between peer SOAP nodes.

- **SOAP envelope**

  The outermost syntactic construct or structure of a SOAP message defined by SOAP within which all other syntactic elements of the message are enclosed.

## SOAP Message Format

- **SOAP header**

  A collection of zero or more SOAP blocks which may be targeted at any SOAP receiver within the SOAP message path.

- **SOAP body**

  A collection of zero or more SOAP blocks targeted at the ultimate SOAP receiver within the SOAP message path.

- **SOAP fault**

  A special SOAP block which contains fault information generated by a SOAP node.

## SOAP Message Format

- **SOAP block**

  A syntactic construct or structure used to delimit data that logically constitutes a single computational unit as seen by a SOAP node. The type of a SOAP block is identified by the fully qualified name of the outer element for the block, which consists of the namespace URI and the local name. A block encapsulated within the SOAP header is called a header block and a block encapsulated within a SOAP body is called a body block.

## SOAP Message Format

- Formal definitions

  The document element information item has:

  - A local name of `Envelope`
  - Zero or more namespace qualified attribute information items
  - One or two element information item children in order as follows:
    - An optional `Header` element information item
    - A mandatory `Body` element information item.

# SOAP Message Format

- Formal definitions (cont'd.)

  The Header element information item has:

  - A local name of `Header`
  - Zero or more namespace qualified attribute information item children
  - Zero or more namespace qualified element information item children
  - Each SOAP header block element information item
    - MUST be namespace qualified
    - MAY have an `encodingStyle` attribute information item
    - MAY have an `actor` / `role` attribute information item
    - MAY have a `mustUnderstand` attribute information item

## SOAP Message Format

- Formal definitions (cont'd.)

  The Body element information item has:

  - A local name of `Body`
  - Zero or more namespace qualified element information item children
  - Each SOAP header block element information item
    - MUST be namespace qualified;
    - MAY have an `encodingStyle` attribute information item

  The namespaces for all Envelope elements are:

  - http://schemas.xmlsoap.org/soap/envelope/ (SOAP 1.1)
  - http://www.w3.org/2003/05/soap-envelope (SOAP 1.2)

# SOAP Message Format

- Sample SOAP fault (SOAP 1.1)

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Body>
    <env:Fault>
      <faultcode>env:Server</faultcode>
      <faultstring>Server Error</faultstring>
      <detail>My application didn't work</detail>
    </env:Fault>
  </env:Body>
</env:Envelope>
```

## SOAP Message Format

- Sample SOAP fault (SOAP 1.2)

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Body>
    <env:Fault>
      <env:Code>
        <env:Value>env:Sender</env:Value>
      </env:Code>
      <env:Reason>
        <env:Text>Sender Timeout</env:Text>
      </env:Reason>
      <env:Detail>Have waited 5 minutes</env:Detail>
    </env:Fault>
  </env:Body>
</env:Envelope>
```

## SOAP Message Format

- Fault-Codes
  - *VersionMismatch*:
    invalid SOAP Envelope
  - *MustUnderstand*:
    A `mustUnderstand` header block could not be processed
  - *DataEncodingUnknown* (nur SOAP 1.2):
    message encoding not supported
  - *Client* (SOAP 1.1) / *Sender* (SOAP 1.2):
    message invalid
  - *Server* (SOAP 1.1) / *Receiver* (SOAP 1.2):
    server-side error

# SOAP Message Format

- Message Encoding
  - given in the `encodingStyle` attribute (optional)
  - Mostly: literal encoding
    - XML serialization
    - recommended by WS-I

## SOAP Message Format

- Another example

```
<env:Envelope xmlns:env="http://www.w3.org/2001/09/soap-envelope">
  <env:Header>
    <n:alertcontrol xmlns:n="http://example.org/alertcontrol">
    <n:priority>1</n:priority>
    <n:expires>2001-06-22T14:00:00-05:00</n:expires>
    </n:alertcontrol>
  </env:Header>
  <env:Body>
    <m:alert xmlns:m="http://example.org/alert">
    <m:msg>Pick up Mary at school at 2pm</m:msg>
    </m:alert>
  </env:Body>
</env:Envelope>
```

Chapter 12.5

# Remote Procedure Calls

## Remote Procedure Calls

- Messages are not a "natural" programming model
  - require that programmer worry about message formats
  - must be packed and unpacked
  - have to be decoded by client and server
  - often asynchronous
  - may require special error handling functions

## Remote Procedure Calls

- Procedure call: a more "natural" way to communicate
  - every language supports it
  - semantics are well defined and understood
  - natural for programmers to use
- Basic idea:
  - Let's just define a server as a module.
  - Let the server export a set of procedures that can be called by clients.
  - Clients just do procedure calls.

# Remote Procedure Calls

- Issues
  - How do we make **R**PCs invisible to the programmer?
  - What are the semantics of parameter passing?
  - How is binding done (locating the server)?
  - How do we support heterogeneity (OS, language)?

## Remote Procedure Calls

- The basic model for Remote Procedure Call (RPC)
  - described by Birrell and Nelson from Xerox PARC in 1984:

    A. D. Birrell and B. J. Nelson: *Implementing remote procedure calls.* ACM Transactions on Computer Systems, 2(1):39–59, February 1984.
  - goals: make RPC look as much like local PC as possible
  - used OS/language support

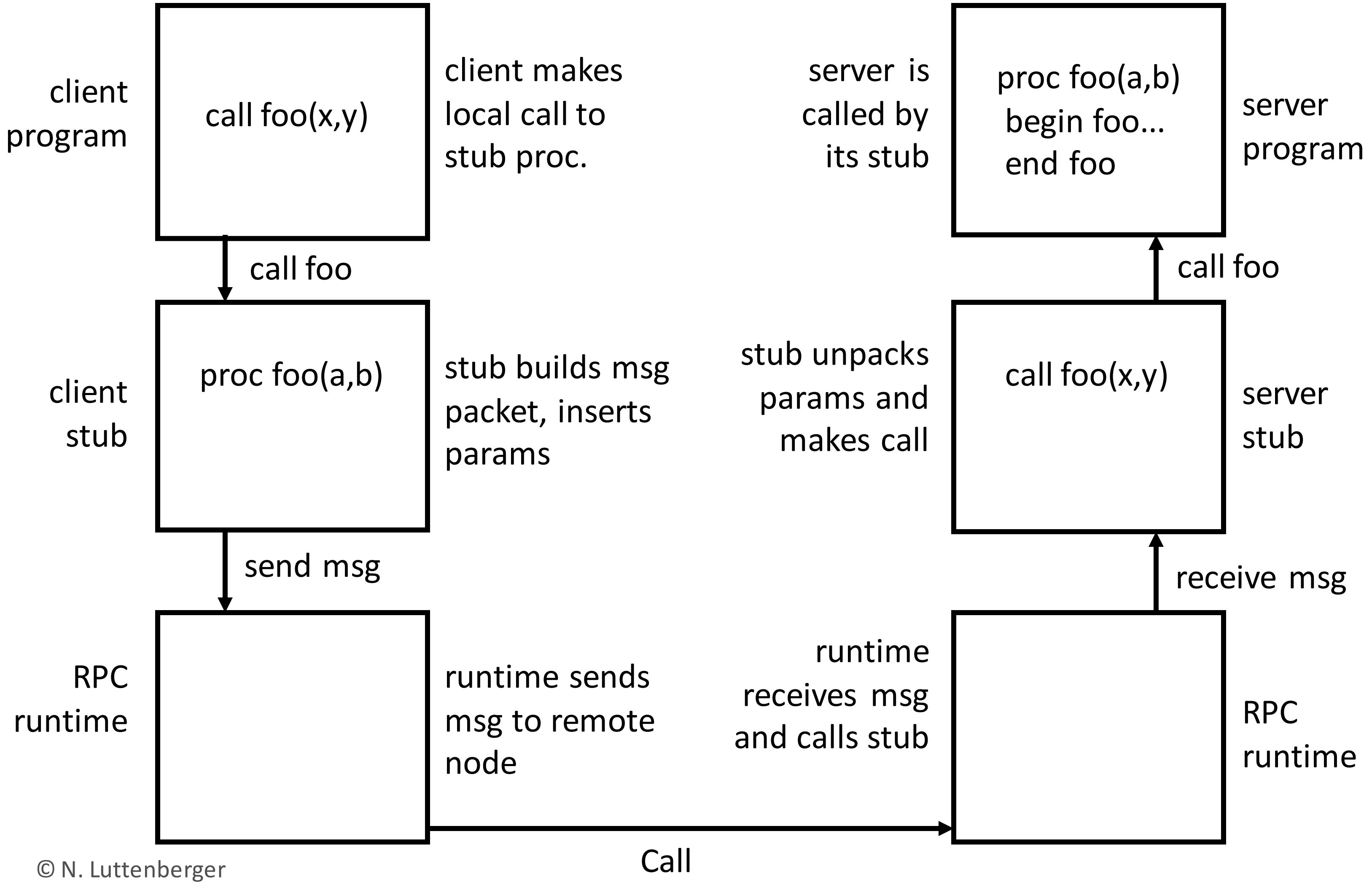
## Remote Procedure Calls

- Implementation: 3 components on each side
  - a user program (client or server)
  - a set of stub procedures
    - client-side stub appears to the client code as if it were a callable procedure
    - server-side stub appears to the server code as a calling client
    - stubs send messages to each other to make the RPC happen
  - RPC runtime support

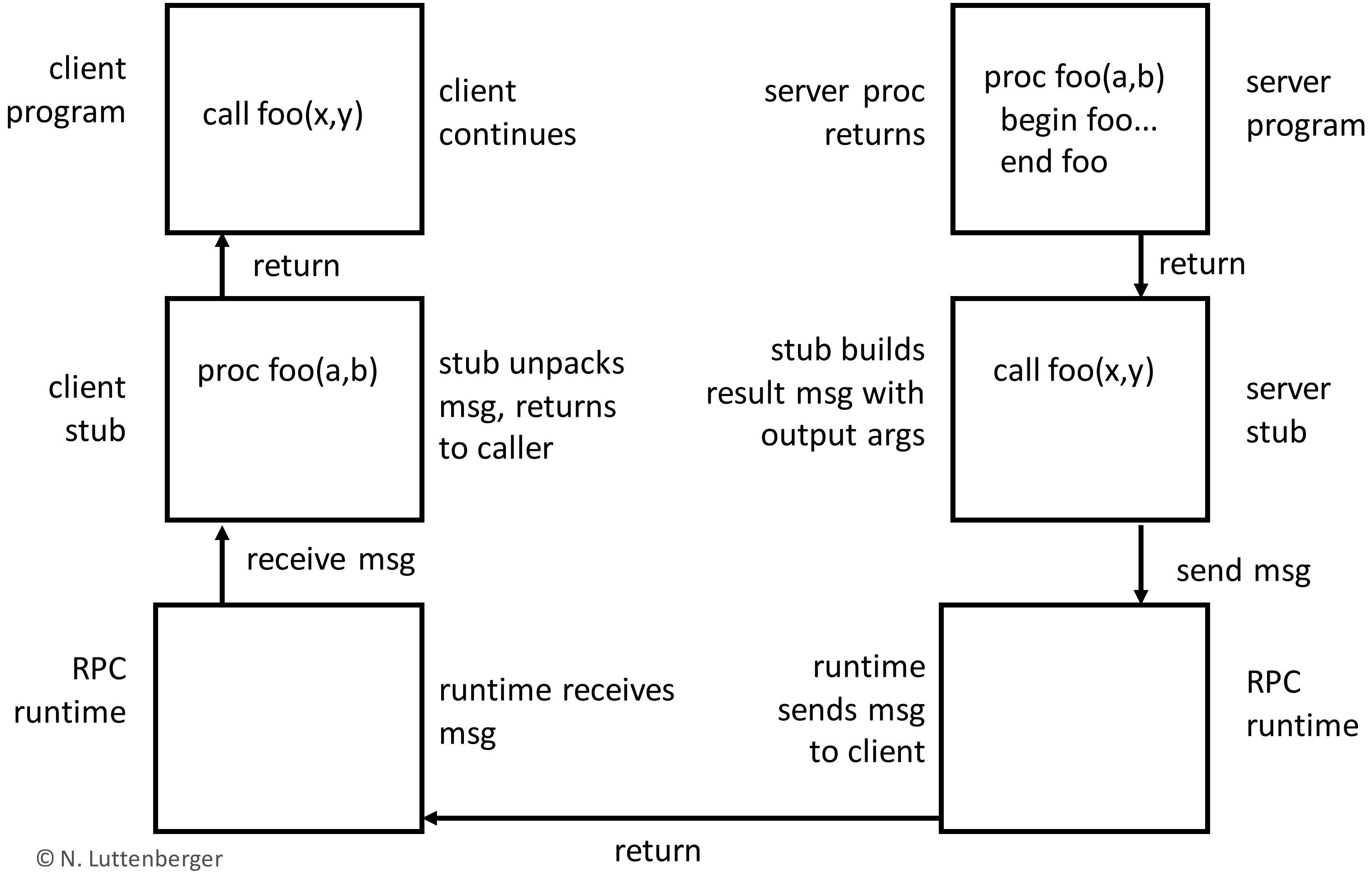
## Remote Procedure Calls

- Building a server
  - Server program defines the server's interface using an *interface definition language* (IDL)
  - IDL specifies the names, parameters, and types for all client-callable server procedures
  - A *stub compiler* reads the IDL and produces two stub procedures for each server procedure: a client-side stub and a server-side stub
  - The server code is linked with the server-side stubs; the client code is linked with the client-side stubs.

# Remote Procedure Calls


client program
call foo(x,y)
client makes local call to stub proc.
call foo
client stub
proc foo(a,b)
stub builds msg packet, inserts params
send msg
RPC runtime
runtime sends msg to remote node
Call
server is called by its stub
proc foo(a,b) begin foo... end foo
server program
call foo
stub unpacks params and makes call
call foo(x,y)
server stub
receive msg
runtime receives msg and calls stub
RPC runtime

# Remote Procedure Calls



# Remote Procedure Calls

- RPC binding
  - process of connecting the client and server
  - server, when it starts up, *exports* its interface, identifying itself to a network name server and telling the local runtime its dispatcher address
  - The client, before issuing any calls, *imports* the server, which causes the RPC runtime to lookup the server through the name service and contact the requested server to setup a connection.
  - The *import* and *export* are explicit calls in the code.

## Remote Procedure Calls

- RPC marshalling
  - packing of procedure parameters into a message packet
  - The RPC stubs call type-specific procedures to marshall and unmarshall all of the parameters to the call.
  - On the client side, the client stub marshalls the parameters into the call packet; on the server side the server stub unmarshalls the parameters in order to call the server's procedure.
  - On the return, the server stub marshalls return parameters into the return packet; the client stub unmarshalls return parameters and returns to the client.

# Remote Procedure Calls

- SOAP-based RPC
  - parameter passing by value only
  - binding via UDDI, WSDL
  - data model based on XML Schema

# Remote Procedure Calls

- SOAP-based RPC
  - namespace of the target object
  - method name
  - parameters

```
<env:Body>
  <m:GetLastTradePrice
    xmlns:m=“http://stocks.com/StockQuotes">
    <tickerSymbol>SUNW</tickerSymbol>
  </m:GetLastTradePrice>
</env:Body>
```

## Feedback for this Chapter

- Well understood – easy material
- Mostly understood – material is ok
- Hardly understood – difficult material