Prof. Dr. Claudia Müller-Birn
Institute for Computer Science, Networked Information Systems

Freie Universität Berlin

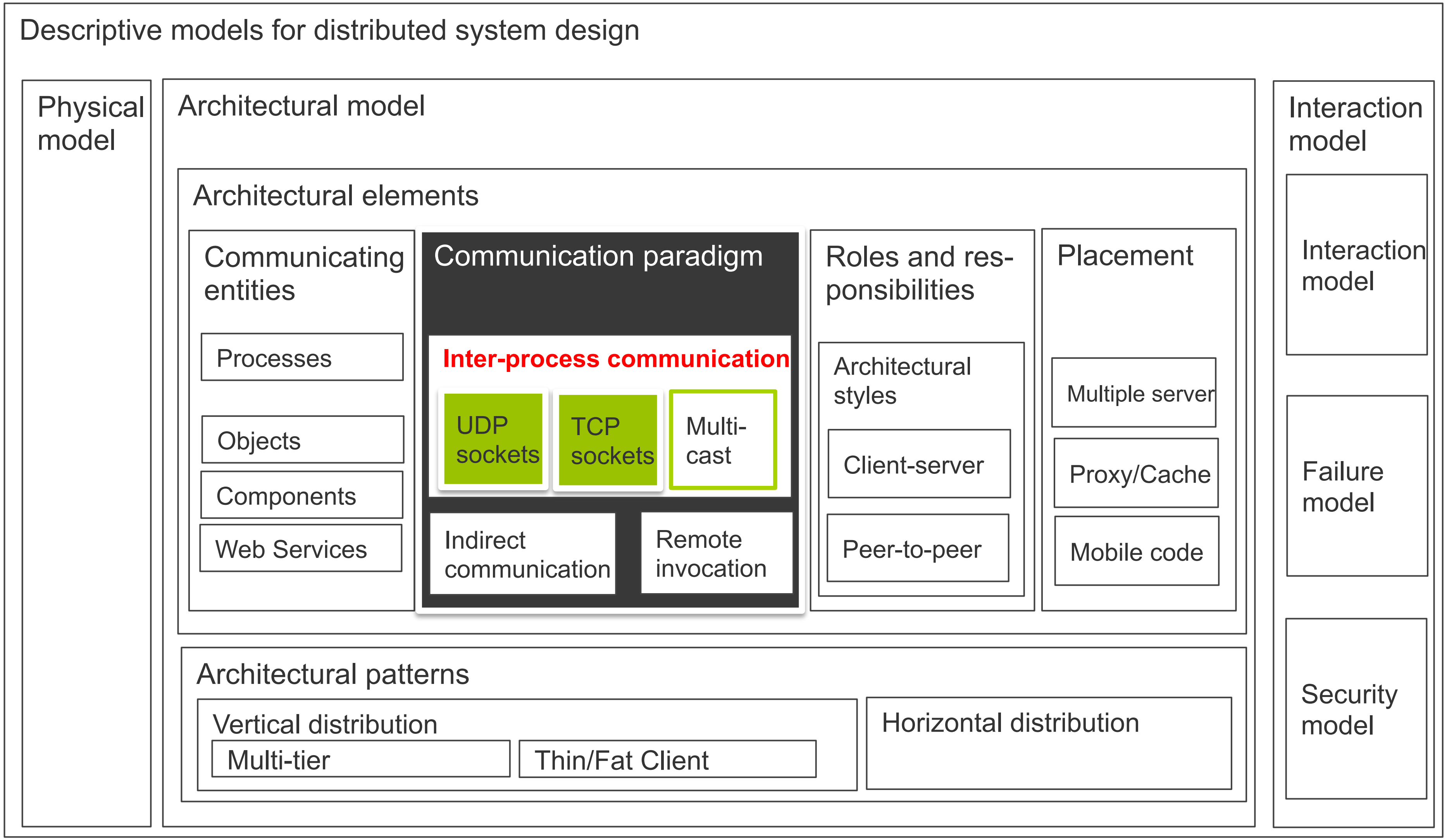
Structured communication (Remote invocation)

Nov 8th, 2011

Netzprogrammierung

(Algorithmen und Programmierung V)

Our topics last week



Descriptive models for distributed system design

Physical model

Architectural model

Architectural elements

Communicating entities

- Processes
- Objects
- Components
- Web Services

Communication paradigm

Inter-process communication

- UDP sockets
- TCP sockets
- Multi-cast
- Indirect communication
- Remote invocation

Roles and responsibilities

Architectural styles

- Client-server
- Peer-to-peer

Placement

- Multiple server
- Proxy/Cache
- Mobile code

Architectural patterns

Vertical distribution

- Multi-tier
- Thin/Fat Client

Horizontal distribution

Interaction model

- Interaction model
- Failure model
- Security model

Our topics today

UDP Style request-reply protocols

- Failure model of UDP request-reply protocol

Use of TCP streams to implement request-reply protocol

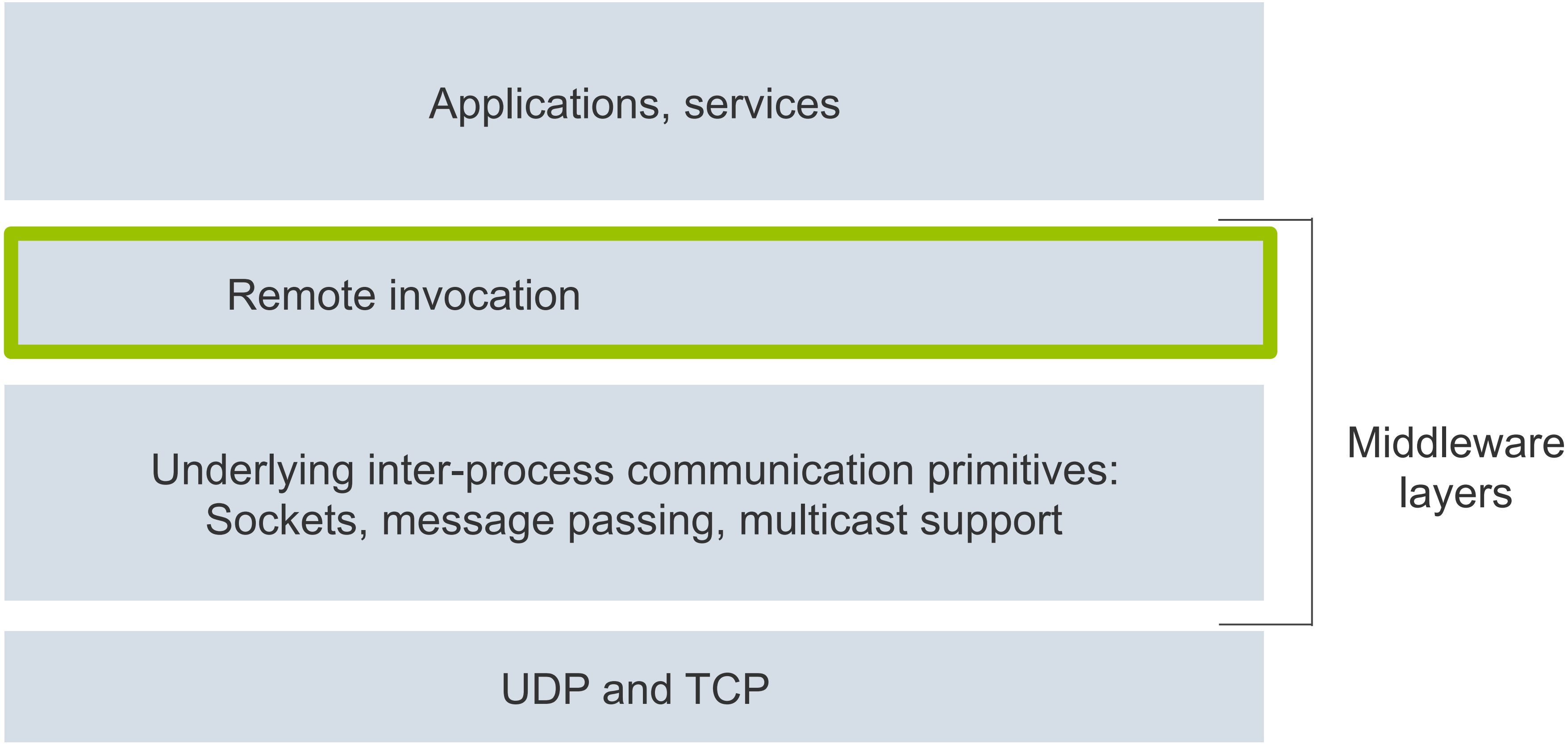
- HTTP: an example of a request-reply protocol

Remote procedure call

Remote Method invocation

- The distributed object model
- Implementation of RMI

Middleware layers



Applications, services

Remote invocation

Underlying inter-process communication primitives:
Sockets, message passing, multicast support

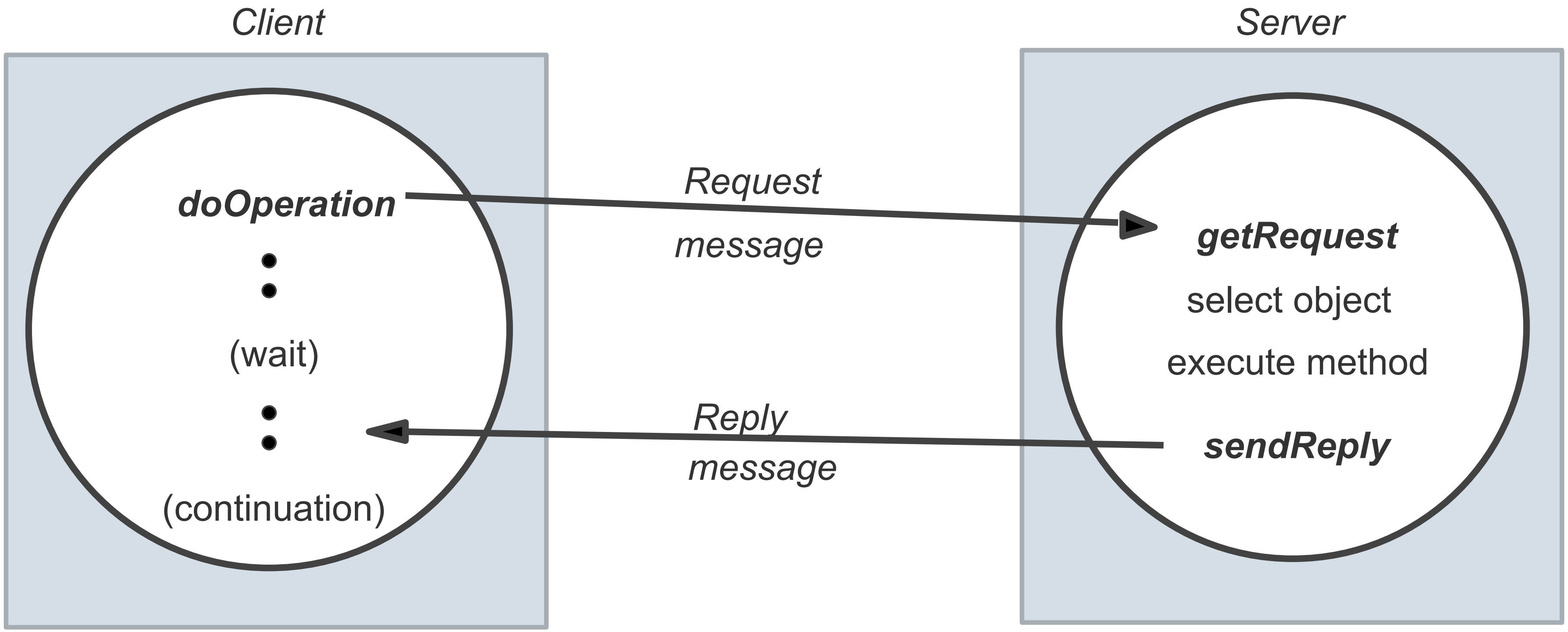
UDP and TCP

Middleware
layers

Remote invocation

UDP Style request-reply protocols

UDP style request-reply protocol



Client

doOperation

(wait)

(continuation)

Request

message

Reply

message

Server

getRequest

select object

execute method

sendReply

Operations of the request-reply protocol (UDP)

public byte[] doOperation (RemoteRef s, int operationId, byte[] arguments)

sends a request message to the remote server and returns the reply.

The arguments specify the remote server, the operation to be invoked and the arguments of that operation.

public byte[] getRequest ();

acquires a client request via the server port.

public void sendReply (byte[] reply, InetAddress clientHost, int clientPort);

sends the reply message reply to the client at its Internet address and port.

Request-reply message structure

messageType	*int (0=Request, 1= Reply)*
requestId	*int*
remoteReference	*RemoteRef*
operationId	*int or Operation*
arguments	*array of bytes*

Message identifiers

Unique message identifies is needed for any scheme that involves management of messages to provide additional properties such as

- reliable delivery
- request-reply communication

Parts of a message identifier

- requestID, which is taken from an increasing sequence of integers by the sending process
- an identifier for the sender process, for example, its port and Internet address

UDP Style request-reply protocols

Failure model of UDP request-reply protocol

Approaches to handle failures

Repeatedly request message

- doOperation sends the request message repeatedly until either it gets a reply or it is reasonable sure the the delay is due to lack of response from the server, rather than lost messages

Discarding duplicate request messages

- Server may receive more than one request message, e.g. server needs longer than the client’s timeout to execute the command and return reply
- Problem: Operation is more than once executed to the same request
- Protocol is designed to recognize successive messages (from the same client) with the same request identifiers

Approaches to handle failures (*cont.*)

Lost reply messages

- Problem: Server has already sent the reply when it receives a duplicate request it will need to execute the operation again to obtain the result
- *Idempotent* operation is an operation that can be performed repeatedly with the same effect as if it had been performed exactly once

History

- Refer to a structure that contains a record of (reply) messages that have been transmitted
- Entry contains: request identifier, message, identifier of a client

Possible Exchange Protocols

Name	*Messages sent by*		
	Client	*Server*	*Client*
R	*Request*		
RR	*Request*	*Reply*	
RRA	*Request*	*Reply*	*Acknowledge reply*

R	=	no response is needed and the client requires no confirmation
RR	=	a server's reply message is regarded as an acknowledgement
RRA	=	Server may discard entries from its history

(Identified by Spector[1982])

Request-reply protocols

Use of TCP streams to implement request-reply protocol

HTTP: an example of a request-reply protocol

HTTP specifies the messages involved in a request-reply exchange, the methods, arguments and results, and the rules for representing (marshalling) them in the messages

Fixed set of resources are applicable to all of server’s resources, e.g., GET, PUT, POST

Additional functions

- *Content negotiation*: clients’ requests can include information as to what data presentation they can accept (e.g. language)
- *Authentication*: Credentials are used to support password-style authentication

Client/server interaction HTTP over TCP (original version)

1. The client requests and the server accepts a connection at the default server port or at the port specified in the URL.
2. The client sends a request message to the server.
3. The server sends a reply message to the client.
4. The connection is closed.

Client/server interaction HTTP 1.1 over TCP

Usage of persistent connections

Connections remain open over a series of request-reply exchanges between client and server

Connection may be closed by client or server any time by sending an indication to the other participant

RFC 2616, (Fielding et al. 1999)

HTTP methods

GET

- Requests the resource whose URL is given as its argument

HEAD

- Request is identical to GET but does not return any data
- Returns all the information about the data such as time of last modification

PUT

- Requests that the data supplied in the request is stored with the given URL as its identifier either as a modification of an existing resource or as a new resource

HTTP methods (*cont.*)

POST

- Is used to send data to the server to be processed in some way
- Designed to deal with
  - Providing a block of data to a data-handling process such as a servlet
  - Posting a message to a mailing list or updating member details
  - Extending a database with an append operation

Additional methods: DELETE, OPTIONS, TRACE

Message contents

HTTP request message

method	*URL or pathname*	*HTTP version*	*headers*	*message body*
GET	//www.dcs.qmw.ac.uk/index.html	HTTP/ 1.1		

HTTP reply message

HTTP version	*status code*	*reason*	*headers*	*message body*
HTTP/1.1	200	OK		resource data

Status code definitions and more:
http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

Remote invocation

Remote procedure call

Issues that are important to understand the concept

The style of programming promoted by RCP – programming with interfaces

The call semantics associated with RPC

The key issue of transparency and how it relates to remote procedure calls

Programming with interfaces

Modern programming languages provide a means of organizing a program as a set of modules that can communicate with one another.

Communication between modules can be by means of procedure calls between modules or by direct access to the variables in another module

In order to control possible interactions between modules, an interface is defined for each module which specifies the procedures and variables that can be assessed.

Advantages of using interfaces in distributed systems

Modular programming allows programmers to be concerned only with the abstraction offered by the service interface and they need not be aware of implementation details.

Extrapolating to (potentially heterogeneous) distributed systems, programmers also do not need to know the programming language or underlying platform used to implement the services.

Approach provides the natural support for software evolution in that implementations can change as long as the interface (the external view) remains the same.

RPC call semantics

Fault tolerance measures			*Call semantics*
Retransmit request message	*Duplicate filtering*	*Re-execute procedure or retransmit reply*	
No	Not applicable	Not applicable	*Maybe*
Yes	No	Re-execute procedure	*At-least-once*
Yes	Yes	Retransmit reply	*At-most-once*

RPC call semantics (*cont.*)

Maybe semantics

- RPC may be executed once or not at all, it means that faults are not tolerated
- Can suffer from omission and crash failures

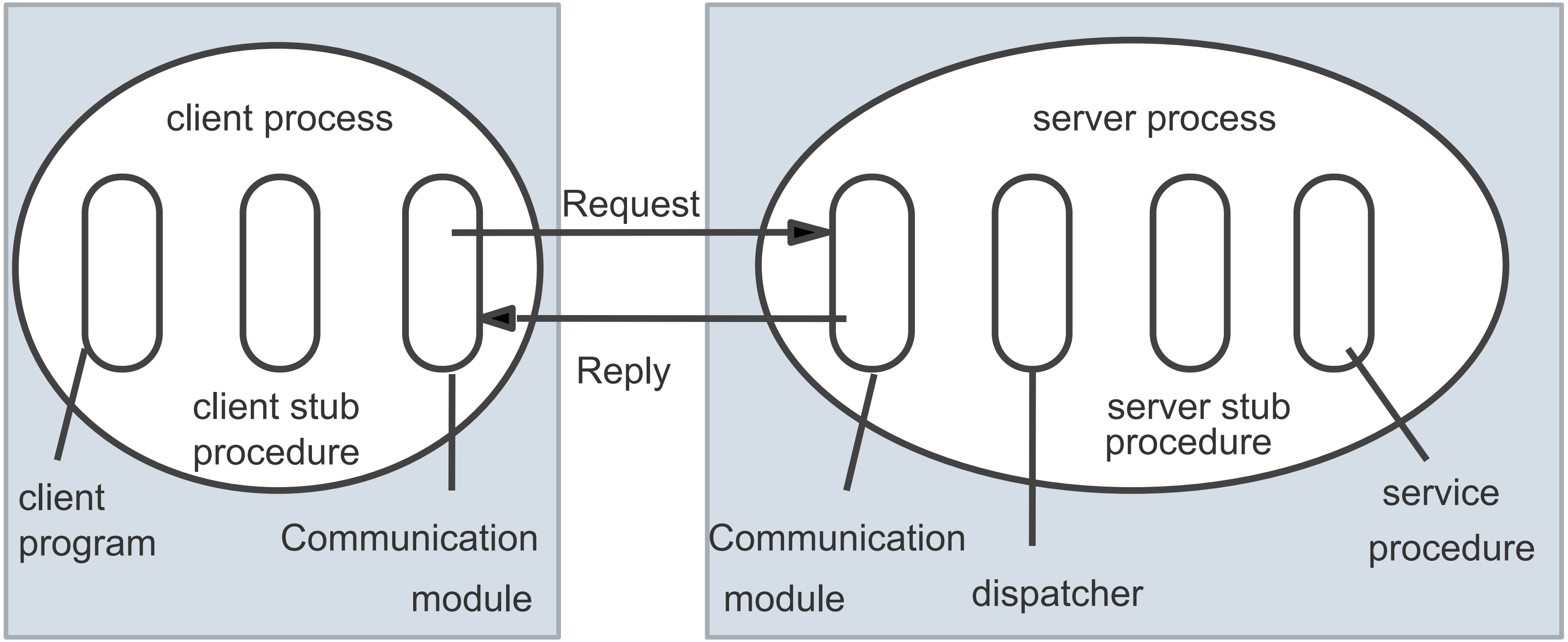
At-least-once semantics

- Invoker receives either a result, in which case the procedure was executed at least once, or an exception informing that no result was received
- Can suffer from crash failures and arbitrary failures

At-most-once semantics

- Caller receives either a result, then the procedure was executed once, or an exception, that no results has been received

Implementation of RPC



Remote invocation

Remote method invocation (RMI)

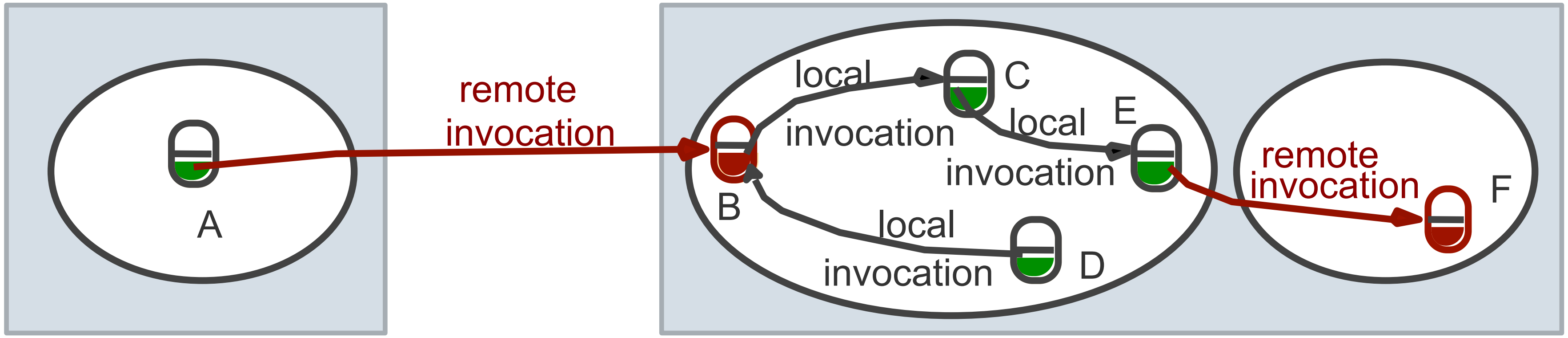
Commonalities of RMI and RPC

- Support of programming languages with interfaces
- Both are typically constructed on top of the request-reply protocol
- Offer semantics such *as at-least-once* and *at-most-once*
- Offer a similar level of transparency, means local and remote calls employ the same syntax but remote interfaces expose the distributed nature for example by supporting remote exceptions

Remote method invocation

The distributed object model

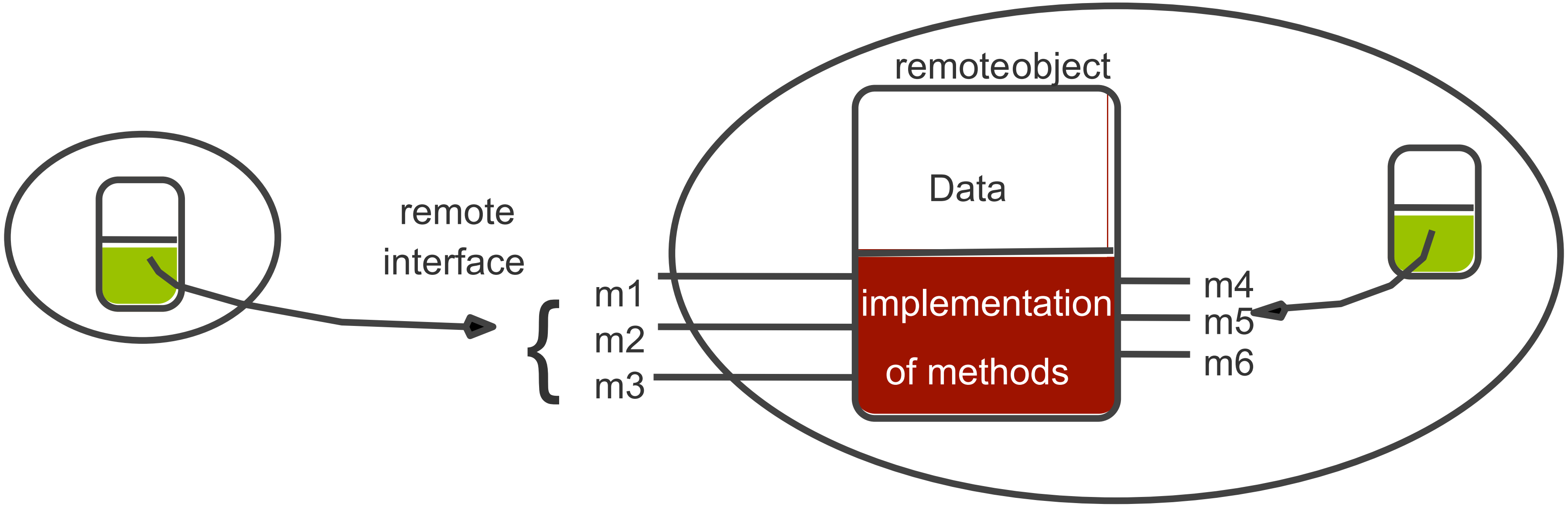
Remote and local method invocation



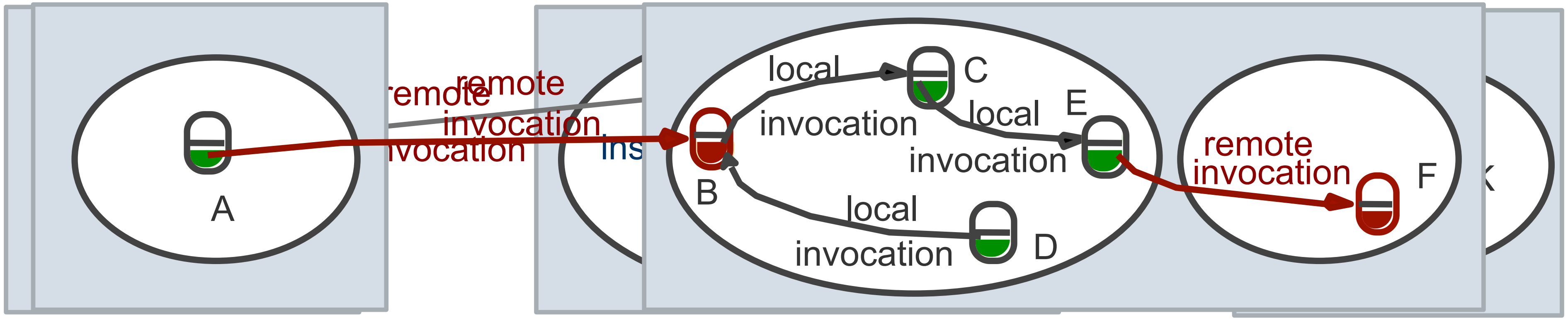
Remote object reference: Other objects can invoke the methods of a remote object if they have access to its remote object reference.

Remote interface: Every remote object has a remote interface that specifies which of its methods can be invoked remotely.

A remote object and its remote interface



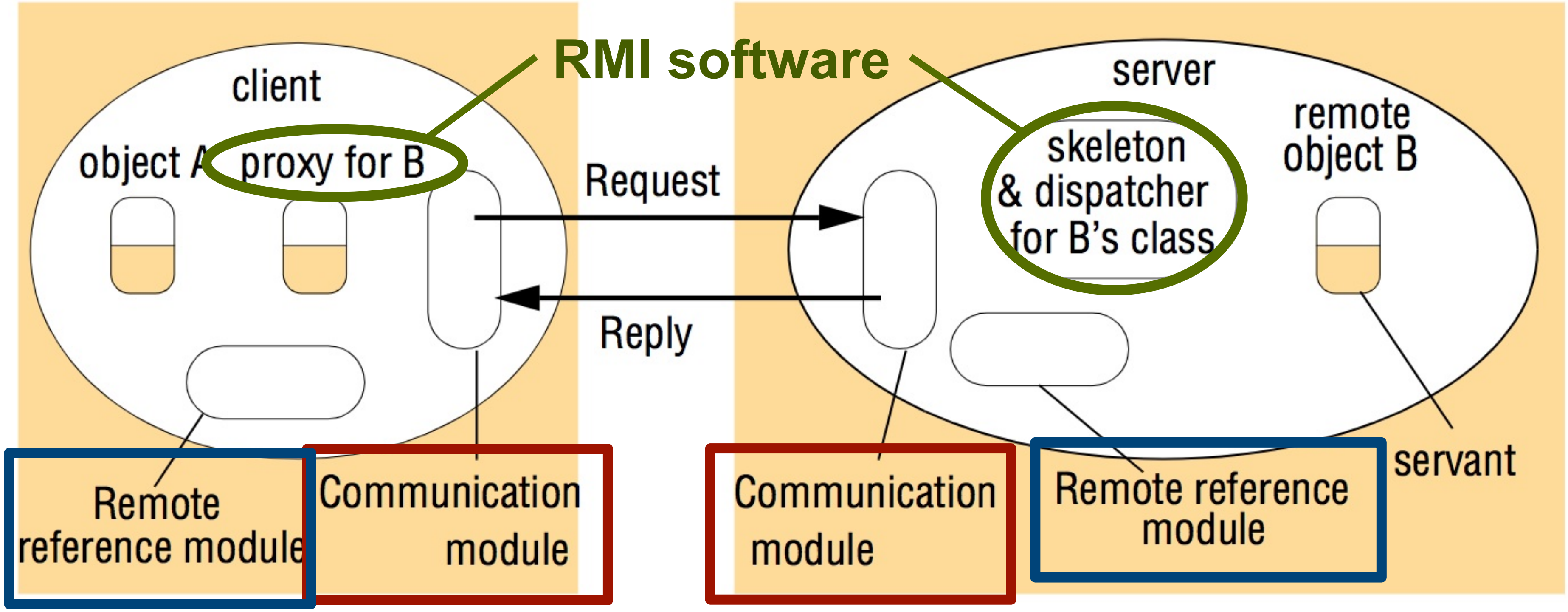
Instantiation of remote objects



Remote method invocation

Implementation of RMI

Generic RMI Modules



The Communication Module

Two cooperating communication modules carry out the request-reply protocol.

Content of request and reply messages

messageType
requestId
remoteReference

Communication modules provide together a specified invocation semantics.

The communication module in the server selects the dispatcher for the class of the object to be invoked, passing on the remote object’s local reference.

The Remote Reference Module

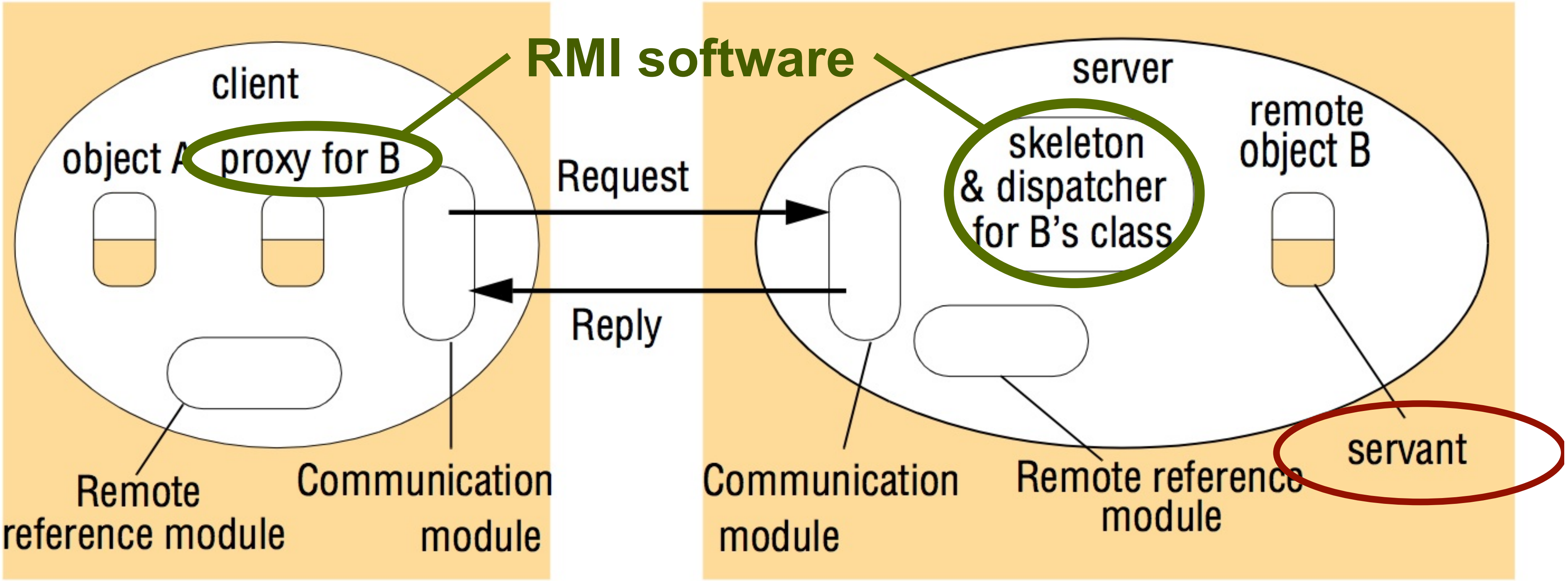
It is responsible for translating between local and remote object references and for creating remote object references.

The remote reference module holds a table that records the correspondence between local object references in that process and remote object references (which are system wide).

Table includes

- An entry for all remote objects held by the process
- An entry for each local proxy

Generic RMI Modules



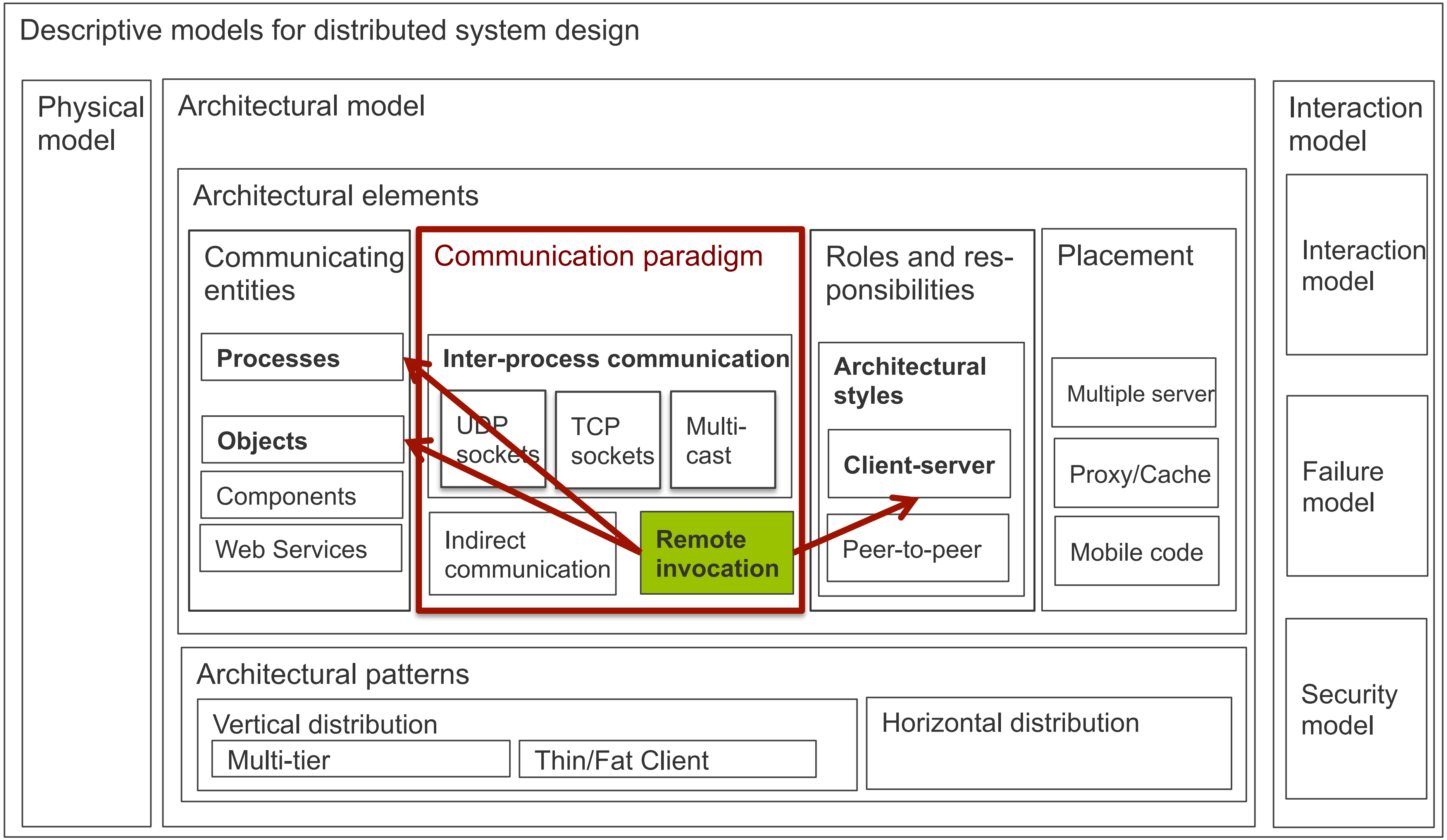
Remote method invocation

Summary

We have we learned?

- Basic communication primitives of UDP style request-reply protocols
- Basic message structure of the request-reply protocol
- Advantages of choosing TCP for request-reply protocols
- HTTP: an example of a request-reply protocol
- Issues that are important to understand the remote procedure calls
- HTTP methods and their properties
- Importance of interfaces for RPC
- RPC call semantics
- Commonalities and differences of RMI and RPC
- Generic RMI Modules

Our topics today



Descriptive models for distributed system design

Physical model

Architectural model

Architectural elements

Communicating entities

- **Processes**
- **Objects**
- Components
- Web Services

Communication paradigm

Inter-process communication

- UDP sockets
- TCP sockets
- Multi-cast

Indirect communication

Remote invocation

Roles and res-ponsibilities

Architectural styles

- **Client-server**
- Peer-to-peer

Placement

- Multiple server
- Proxy/Cache
- Mobile code

Architectural patterns

Vertical distribution

- Multi-tier
- Thin/Fat Client

Horizontal distribution

Interaction model

- Interaction model
- Failure model
- Security model

Next class

Distributed object component middleware I (Java RMI)

References

Main resource for this lecture:

George Coulouris, Jean Dollimore, Tim Kindberg: *Distributed Systems: Concepts and Design*. 5th edition, Addison Wesley, 2011