**L3 RPC**

**Remote Procedure Calls (RPC)**

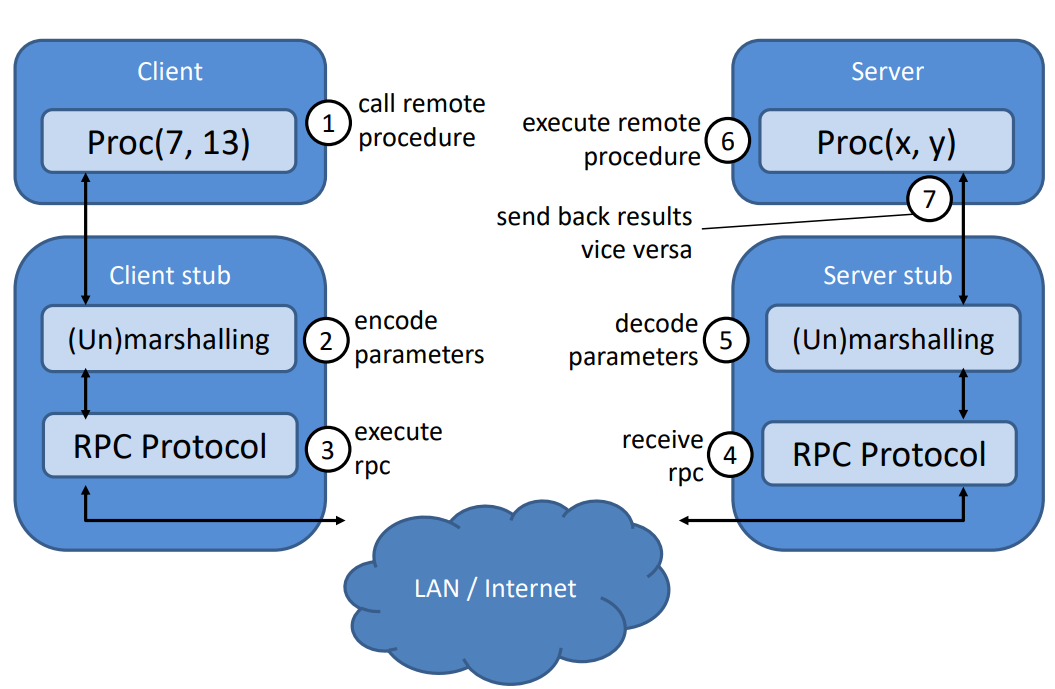
Definition:

* Calling a non-local procedure on a remote machine
* Appears like a local procedure call
* Hides complexity of network
  + Definition of message and content types
  + Marshalling/unmarshalling of parameters
  + Sending/receiving messages

OSI model: **rpc** -> Session Layer

Übung *RPC implementation* wegen OSI-Schichten in Implementation

RPC sequence - communication:



RPC parameter:

* **Call-by-value:** complete value handed to the procedure (primitives: string, int, Boolean)
  + Easy for remote procedure
* **Call-by-reference:** objects passed by reference (just a pointer is given to the procedure)
  + Serialize objects with Thrift, Protocol Buffers etc.

RPC binding:

* Static binding
* Dynamic binding

RPC errors: (vielleicht tricky Sochen noumol unschaugen)

* Lost request
* Lost reply
* Client crash
* Server crash

RPC failure semantics: (same wia errors)

four levels of failure semantics:

* maybe
* at least once
* at most once
* exactly once

(As-)Synchronous RPC:

During synchronous RPCs client is blocked until the result from the server arrives

Asynchronous RPC lets the client continue with its task

RPC problems:

RPC is **not** exactly like a local procedure call

* great area of error semantics
* Problems with scalability
* Security aspects

RPC technologies

* XML-RPC
* JSON-RPC
* SOAP
* Protocol Buffers
* Apache Thrift

**Remote Method Invocation (RMI)**

* Object-oriented RPC
* Procedure call on remote object
* Method parameters can be send two ways
  + call-by-value 🡪 Implement Serializable
  + call-by-reference 🡪 extend Remote interface
* Object reference identifies remote object

RMI vs. RPC:

RPC

* Procedures of remote processes are called
* **Service interface** provides set of procedures

RMI

* Objects in different processes communicate with each other
* **Remote interface** specifies methods of an object

RMI technologies:

* Java RMI
* CORBA
* COM 🡪 DCOM 🡪 .NET

RMI Problems:

* Locked into Java, Does not scale well, uses Java serialization 🡪 Versioning problem

**What is the downside of using delimiters to build/parse object properties?**

* The values of object properties should never include one of the delimiters.
* If a class is changed, parsing the old classes will probably fail

**What are alternative ways of doing the marshaling/unmarshaling?**

* Object meta information can be stored into the message → XML
* Object properties can be serialized in bytes, fixed length intervals can be used → Protocol Buffers