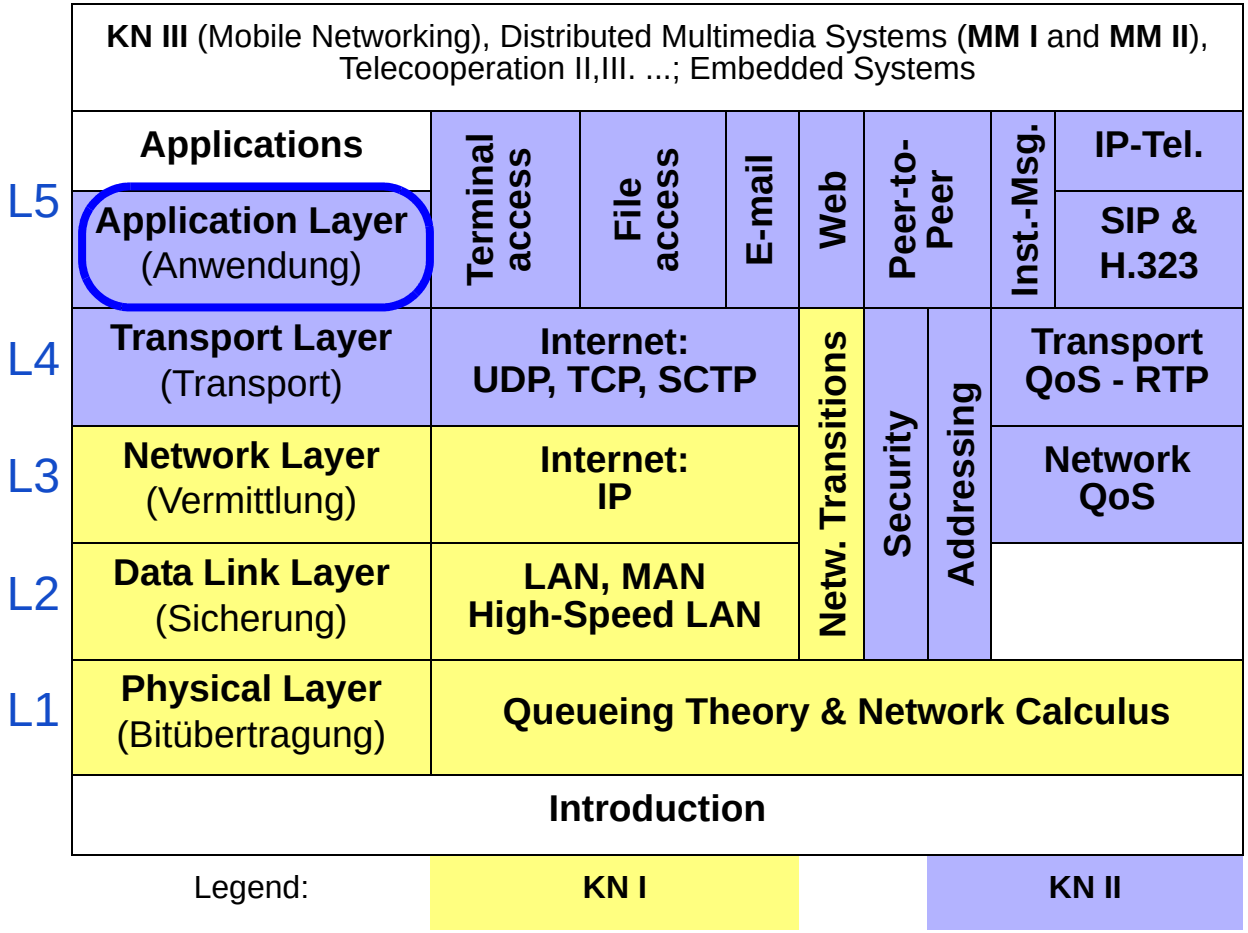
Communication Networks II
Application Layer

Prof. Dr.-Ing. **Ralf Steinmetz**

TU Darmstadt - Technische Universität Darmstadt,
Dept. of Electrical Engineering and Information Technology, Dept. of Computer Science
KOM - Multimedia Communications Lab
Merckstr. 25, D-64283 Darmstadt, Germany, Ralf.Steinmetz@KOM.tu-darmstadt.de
Tel.+49 6151 166151, Fax. +49 6151 166152

httc - Hessian Telemedia Technology Competence-Center e.V
Merckstr. 25, D-64283 Darmstadt, Ralf.Steinmetz@httc.de

Scope


KN III (Mobile Networking), Distributed Multimedia Systems (MM I and MM II), Telecooperation II,III. ...; Embedded Systems
Applications
Terminal access
File access
E-mail
Web
Peer-to-Peer
Inst.-Msg.
IP-Tel.
L5
Application Layer (Anwendung)
SIP & H.323
L4
Transport Layer (Transport)
Internet: UDP, TCP, SCTP
Netw. Transitions
Security
Addressing
Transport QoS - RTP
L3
Network Layer (Vermittlung)
Internet: IP
Network QoS
L2
Data Link Layer (Sicherung)
LAN, MAN High-Speed LAN
L1
Physical Layer (Bitübertragung)
Queueing Theory & Network Calculus
Introduction
Legend:
KN I
KN II

Overview

1. Application-Oriented Communication Services
2. Session Concept

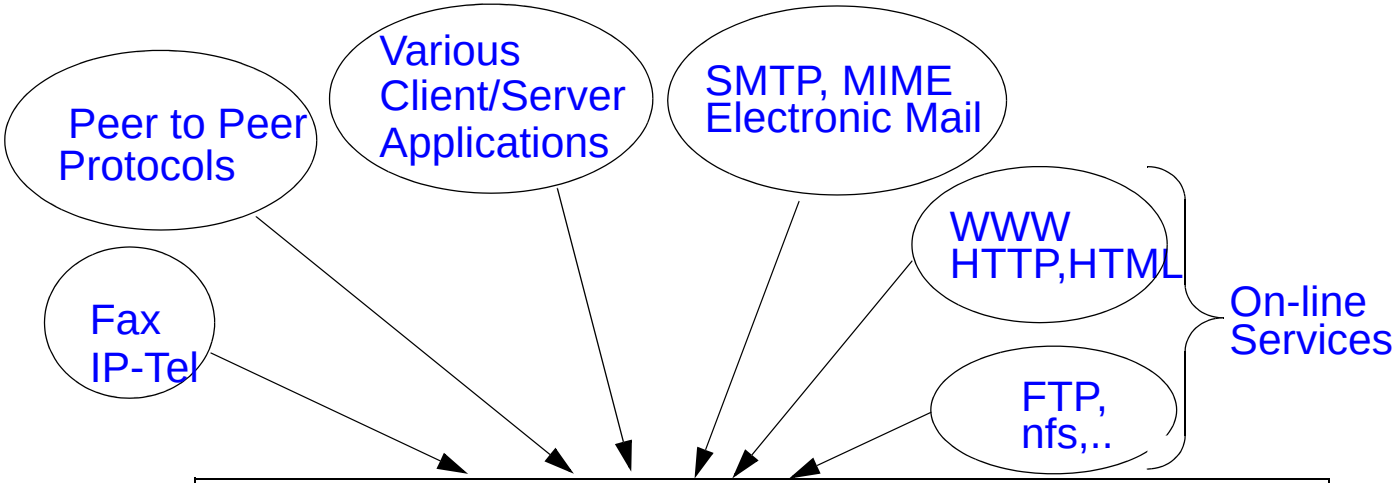
3. Data Presentation

4. Client / Server and Remote Procedure Call

5. Middleware - CORBA

6. Other: E.g. Microsoft .NET

1. Application-Oriented Communication Services



L5	**Application-oriented services**
L4	**Transport layer**
L3	**Network layer**
L2	**Data link layer**
L1	**Physical layer**

2. Session Concept

Approaches for developing distributed programs

1. COMMUNICATION ORIENTED approach

- **to define messages and formats**
- **to use e.g. client-server design**
  - defined as reaction to incoming messages
- **to use sockets**

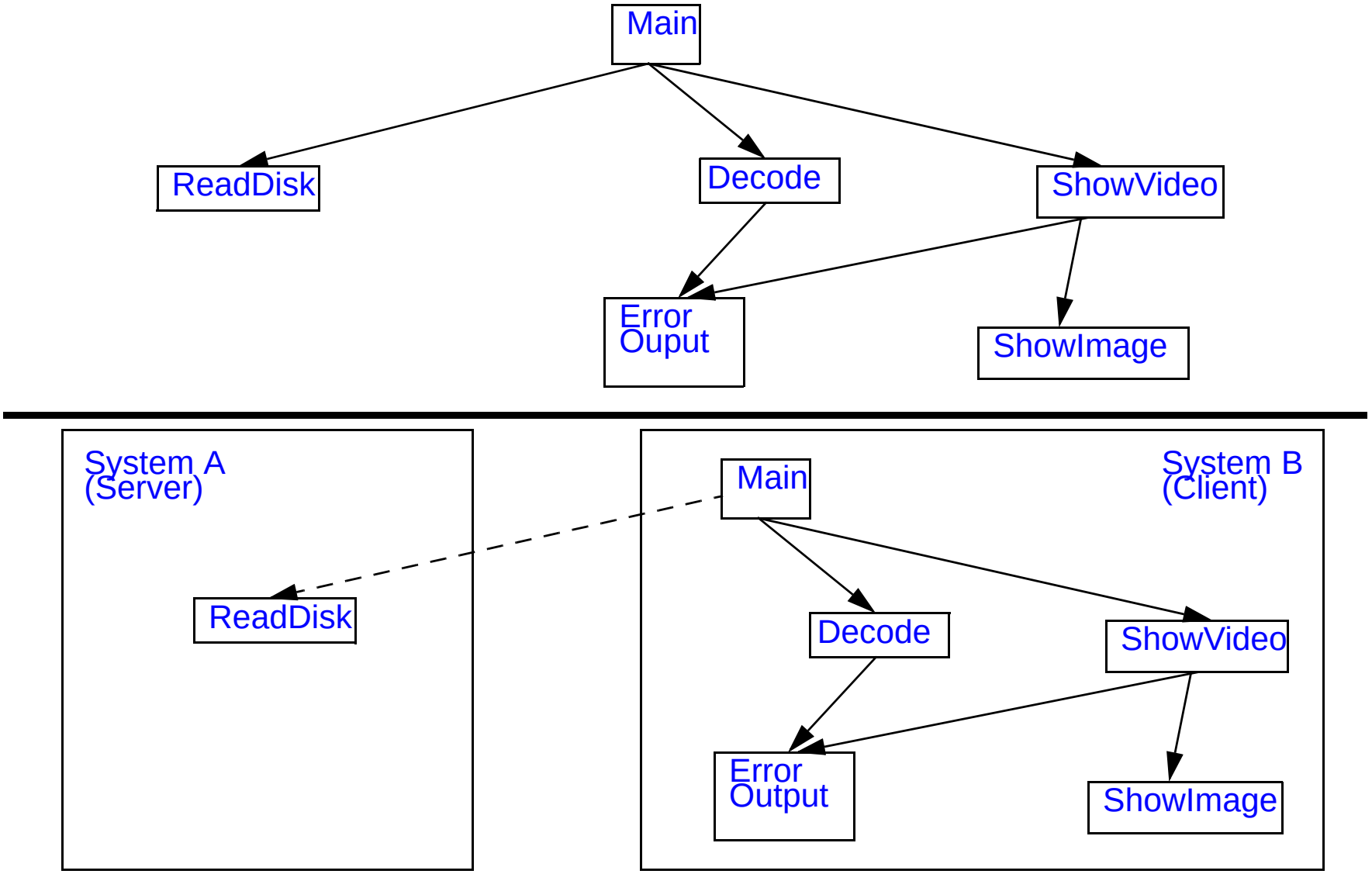
⇒ **evaluation:**

- benefits
  - when all communication is executed on an equal basis
- disadvantages
  - program design depends on type of communication
  - an error in the protocol may lead to complete redesign of the program
  - development of communication protocols may be complex

2. APPLICATION ORIENTED approach

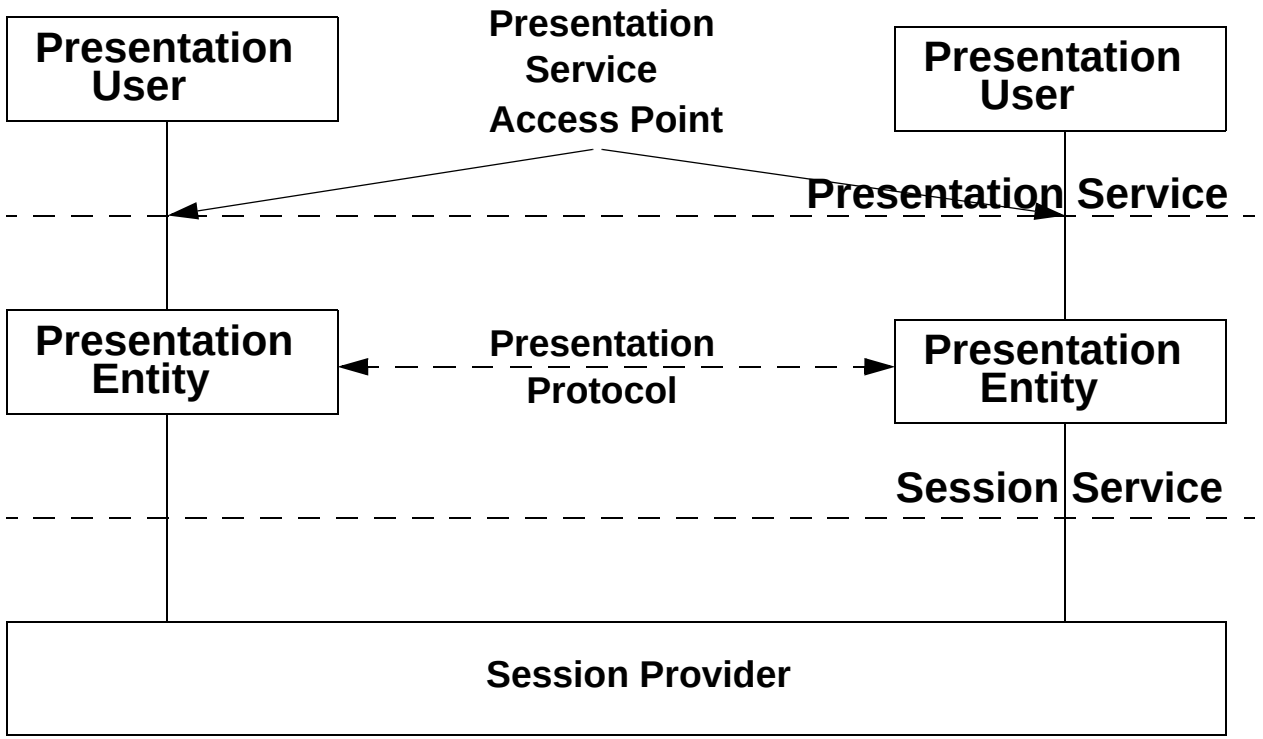
- **to use conventional program development**
- **to transfer modular approach to distributed programming**
- **functionality located in procedures/objects, not in communications**
- **communications between systems independent of programs e.g. by using the Remote Procedure Call concept**

Session: Example



Session: Task

To provide well understood data presentation for any communications between open systems



Session: Task (2)

Functions:

- **to transfer communication control services**
- **to allow the specification of complex data structures**
- **negotiation of required data structures**
- **to convert the local representation into a global one**

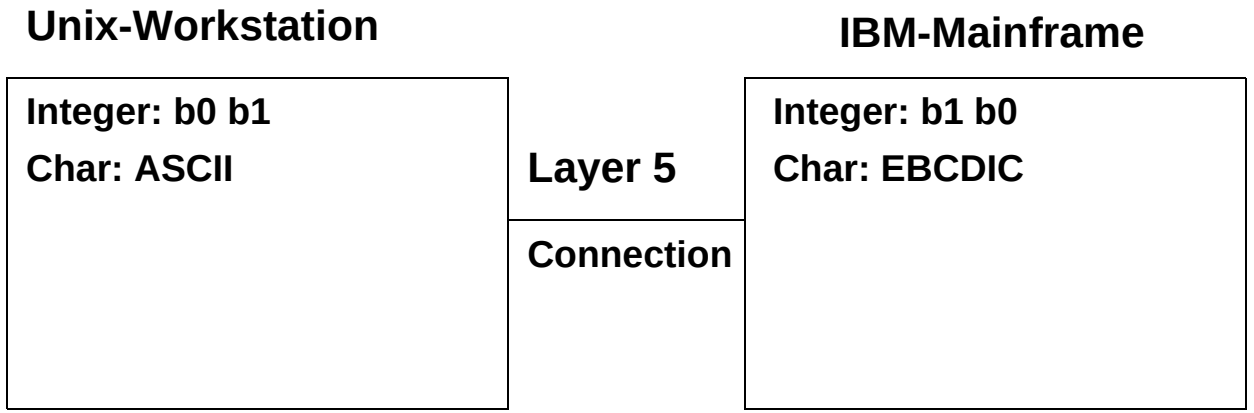
Because

- connection does not mean communication
- communication implies a common understanding

Example:

- **understanding the words**
  - Igel (German) - eagle (English)

Session: Task (3)



Situation:

- **even though correct communication at lower layers there is no further communication possible**

⇒ **Semantics are lost**
- heterogeneous software

⇒ **Coding regulations depend on compiler**
- distributed objects
  (Common Object Request Broker Architecture CORBA)

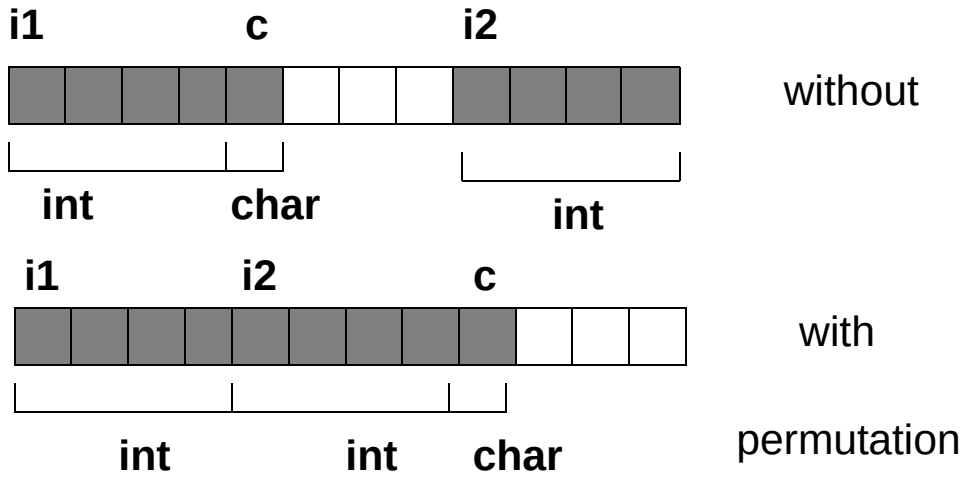
Session Example

Example:

```
struct {
   int    i1;
   char   c;
   int    i2;
}
```

- **char:** **one byte, no alignment conditions**
- **int:** **4 bytes, alignment according to an address divisible by 4**

compilation with and without permutation strategy



Coding Regulations

Type (c)	Coding Rules
INT	Length Coding type Arrangement Justification (with word border)
FLOAT	Length of mantissa Length of the exponential Exponential basis Coding type Arrangement Justification
CHAR	Coding type

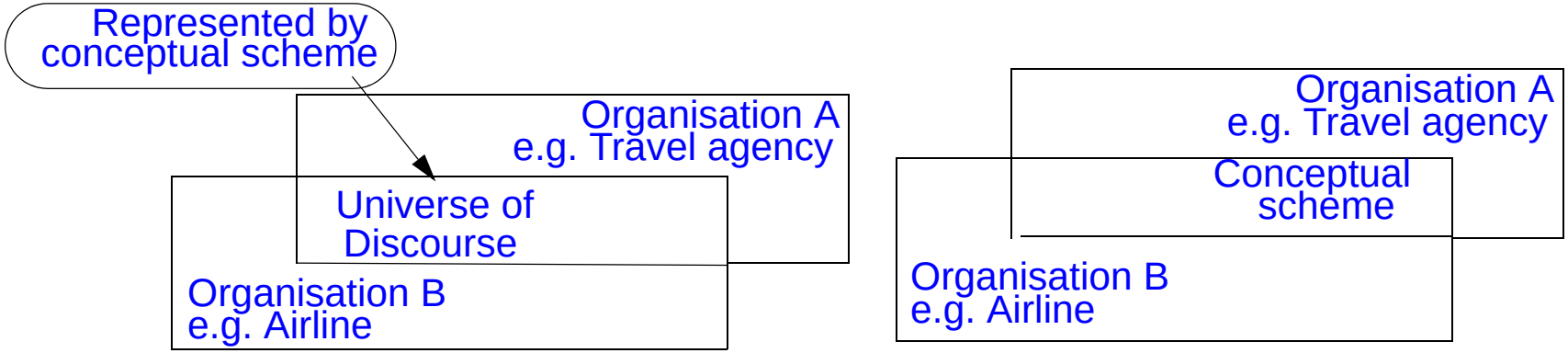
3. Data Presentation

Sender and receiver need common data presentation to allow understanding

- **'communication' of content not of bits**
- **needed for formats, data types, compression, coding, ...**

Generic view:

- **Universe of Discourse**
  - part of the real world which is to be processed in the system
- **Conceptual scheme**
  - formal description of the universe of discourse

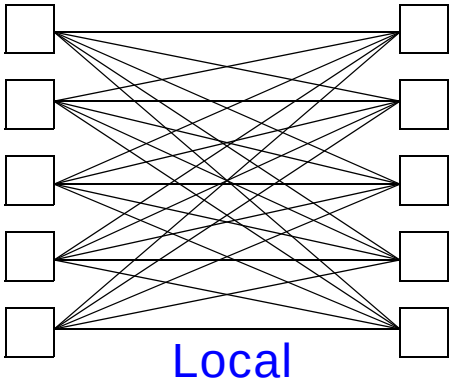


- **Requirements**
  - relation to the same universe of discourse
  - common conceptual scheme
  - comprehensible representation of the conceptual scheme's objects (i.e. data conversion) to both communication parties

Data Presentation: Methods

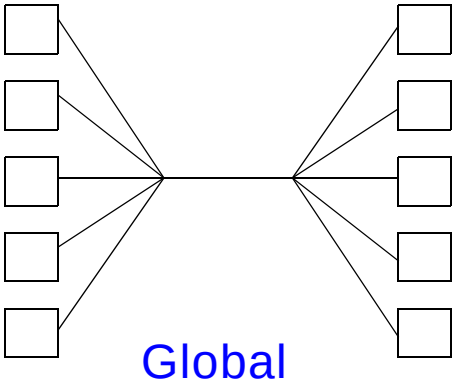
Local presentation of a communication partner

- **n x (n-1) conversion routines**
- **a maximum of one conversion per relationship**
  - local format f1 directly to local format f2



Global presentation

- **2 x n conversion routines**
- **2 conversions per relationship**
  - local f1 to global g,
  - global g to local f2
- **scheme:**



- **standards: XDR, ASN.1**

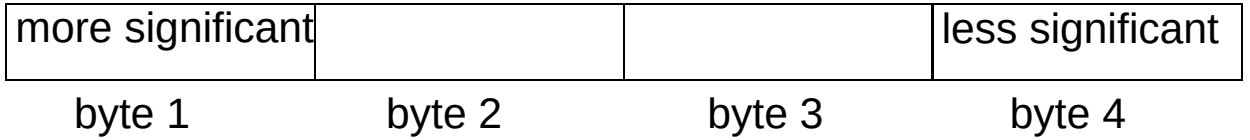
XDR, the Representation „Layer“ of the Internet

XDR: External Data Representation

- **presentation layer with very low functionality**

Example for a conversion issue: integers

1. **BIG-ENDIAN (byte 0 as the most significant (i.e. left)) versus LITTLE-ENDIAN (byte 0 as the least significant (i.e. right))**
   - comment: usually also relates to bits
   - Motorola 68x0, IBM 370 (Big Endian)

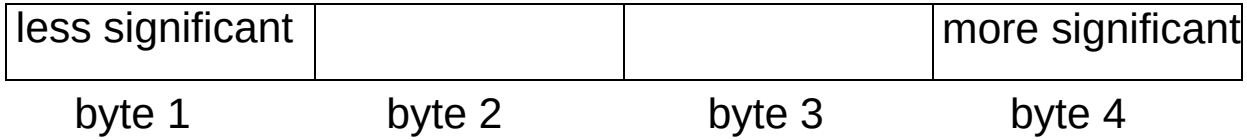


   - below the excerpt from a respective configuration header=file of an IBM RS6000:

```
/* Definitions for byte order, */
/* according to byte significance from low address to high. */
#define LITTLE_ENDIAN   1234    /* least-significant byte first
                                   (vax)          */
#define BIG_ENDIAN      4321    /* most-significant byte first
                                   (IBM, net)     */
#define PDP_ENDIAN      3412    /* LSB first in word, MSW first
                                   in long (pdp) */
#define BYTE_ORDER      BIG_ENDIAN
```

XDR, the Representation „Layer“ of the Internet (2)

2. Intel 80x86 (LITTLE ENDIAN)



- **all data are mapped to a pre-defined transfer syntax (no negotiations)**
  - all integers as 4-byte big-endians
  - floating-point numbers in IEEE format:
    - mantissa 23 bits
    - exponential 8 bits
    - algebraic sign 1 bit
  - texts in ASCII code
  - all data elements aligned with 4-byte limit

Disadvantage:

⇒ **Two systems which are completely identical have to convert twice**

XDR, the Representation „Layer“ of the Internet (3)

Essential component: XDR compiler

- **generates**
  - C data structures compatible with the XDR definition and
  - program pieces for coding and decoding

Summary/example of a typical data packet

start of the packet

Ethernet header
IP header
UDP header
RPC header
User data in XDR format
Ethernet checksum

end of the packet

Comment: XDR does NOT need any own header

4. Client / Server and Remote Procedure Call

Server

- provides services
- waits for incoming service requests from clients
- processes requests and sends results as response
- may use other servers to process request (becomes client in that case)

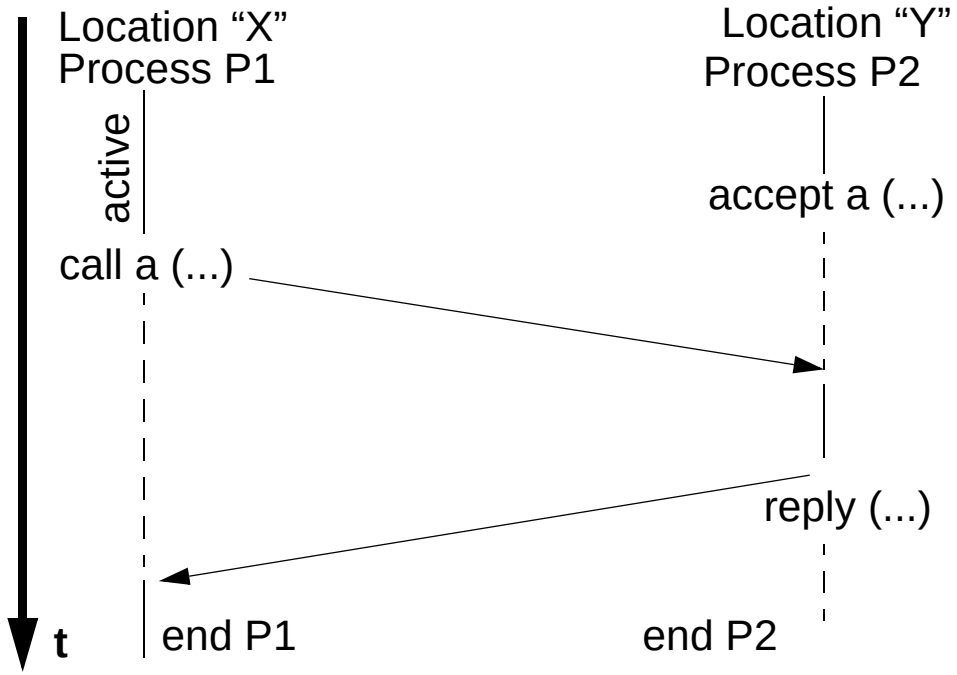
Client

- uses services provided by server
- sends requests to server
- (typically) waits for response from server

For client conceptually similar to PROCEDURE CALL

- call procedure
- wait for result

Remote Procedure Call - RPC



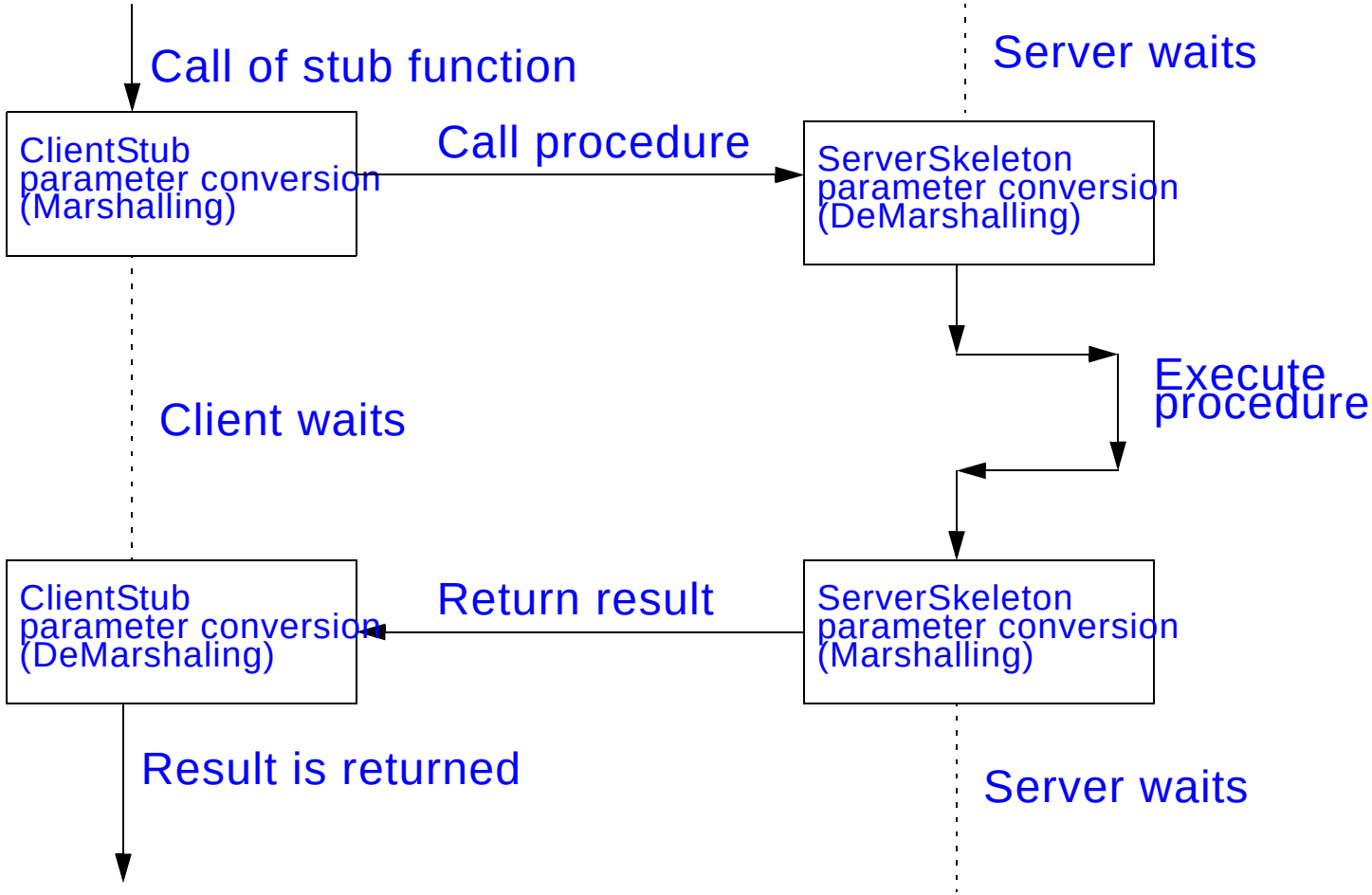
Concept

- **synchronization between client and server**
  - synchronous Remote Service Invocation (sRSI)
- **characteristic: e.g. limited parallelism**

Basic idea:

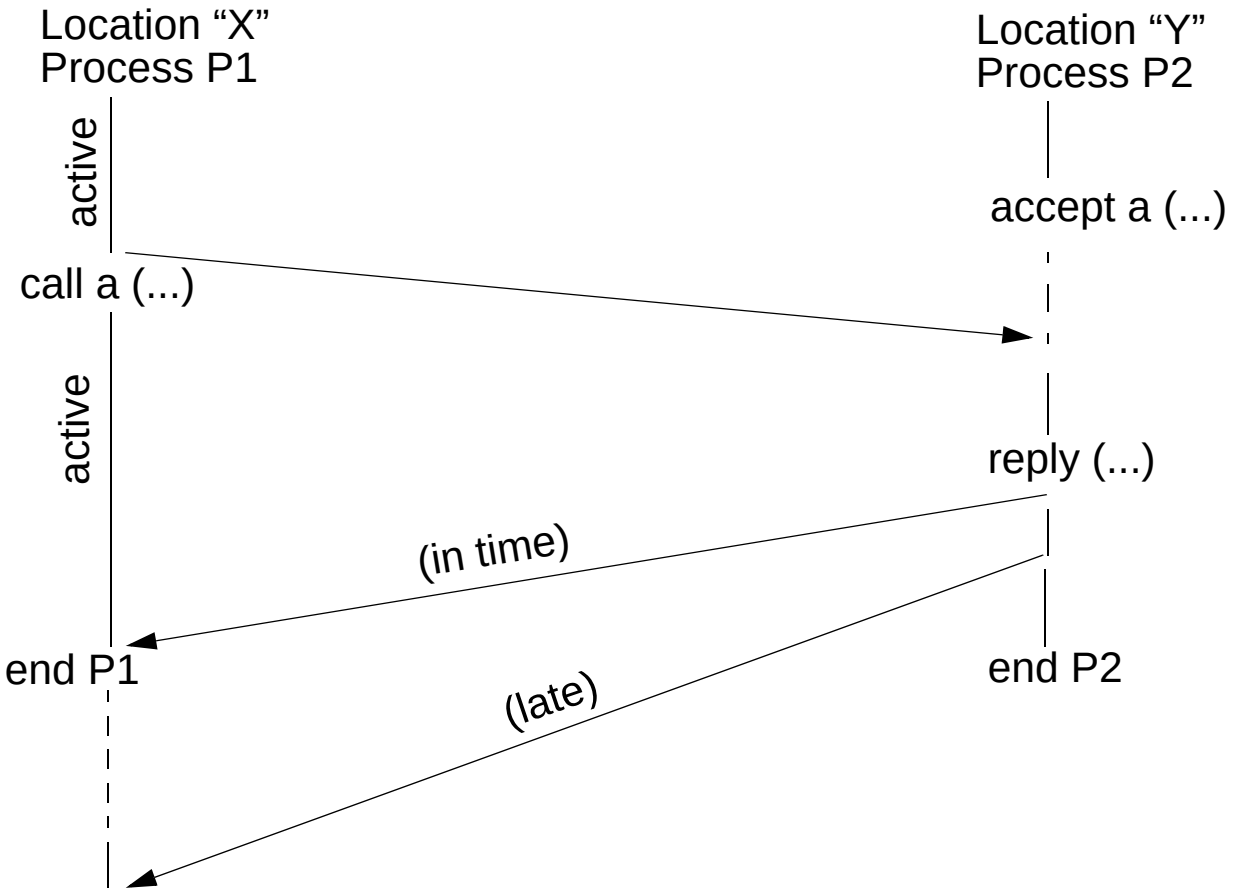
- **application cannot differentiate between**
  - remote procedure call and
  - local procedure call

RPC Example


Call of stub function
ClientStub
parameter conversion
(Marshalling)
Call procedure
Server waits
ServerSkeleton
parameter conversion
(DeMarshalling)
Execute
procedure
Client waits
ClientStub
parameter conversion
(DeMarshaling)
Return result
ServerSkeleton
parameter conversion
(Marshalling)
Result is returned
Server waits

Asynchronous Remote Service Invocation (aRSI)

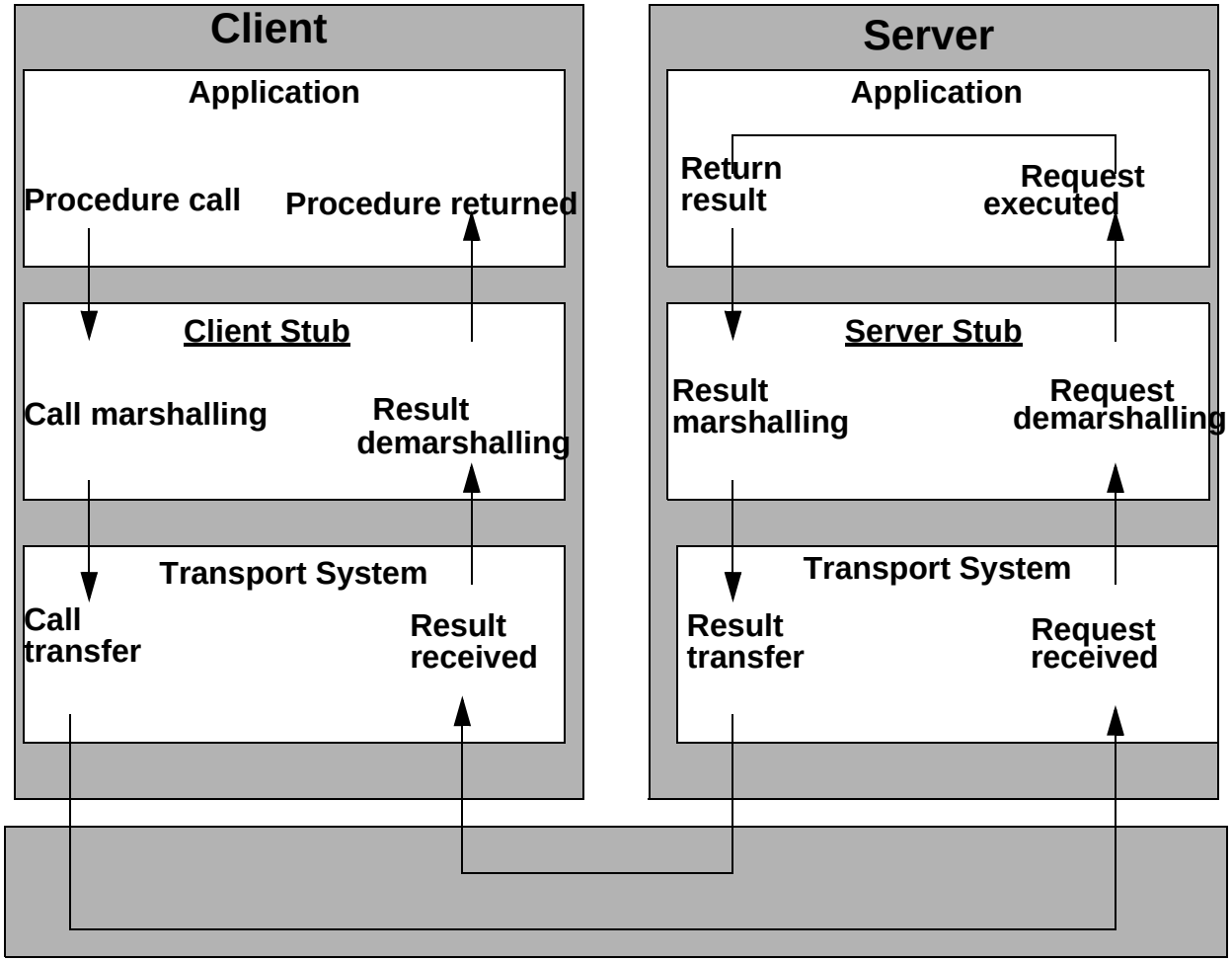
Alternatively to RPC



Characteristics (among others)

- **parallelism between client and server possible**
- **associating requests and respective results more difficult**

RPC: Cycle


Client
Application
Procedure call
Procedure returned
Client Stub
Call marshalling
Result demarshalling
Transport System
Call transfer
Result received
Server
Application
Return result
Request executed
Server Stub
Result marshalling
Request demarshalling
Transport System
Result transfer
Request received

RPC: Cycle (2)

Tasks of the “Stub” procedures:

- **to locate and bind server with/and client**
  - server registers its service at database (server) by providing its name (ASCII), network address and service number (any 32 bit number) (export)
  - client-stub sends request to database
    - its name (which is also the name of the server) in ASCII
  - database service returns network address and unique server identification (binding)
- **marshalling/demarshalling (parameter arrangement) of parameters and results (guarantees transparency)**
  - client
    - collects all parameters of an RPC call and packs them into a message
  - server
    - unpacks the parameters, performs function(s) and packs results into a message
  - client unpacks results
- **error treatment, error semantics**
- **communications**
  - transport system interface
  - data representation
  - authentication/encryption

RPC: Error Semantics

Various errors may occur, e.g.

- **requests or replies get lost or are garbled during data transfer**
- **client or server crashes while RPC is ongoing**

Several error classes can be distinguished

Maybe-semantics

- **server process may have been executed once**

At-least-once-semantics

- **server process is executed error-free at least once (if not more)**

At-most-once-semantics

- **server process is executed error-free at most once**

Exactly-once-semantics

- **server process is executed error-free exactly once (guaranteed) including transmission**

RPC: Idea and Reality

Basic idea:

- **application cannot differentiate between**
  - remote procedure call and
  - local procedure call

Problems:

- **transparency:**
  - parameter treatment
    (“call by reference”, “pointer”, procedures ...)
  - side effects
- **efficiency**
  - additional effort for “marshalling/demarshalling”
  - error treatment, for e.g. recovery after a “server crash”
- **conception**
  - client and server roles may change
    - e.g. in streaming

Implementations

- **e.g. SUN RPC (RFC 1057)**
- **e.g., RPC at Open Software Foundation’s (OSF) Distributed Computing Environment (DCE)**

5. Middleware - CORBA

Common Object Request Broker Architecture - CORBA

- **remark:**
  see also former slides in German (last German Version term WS 00/01)

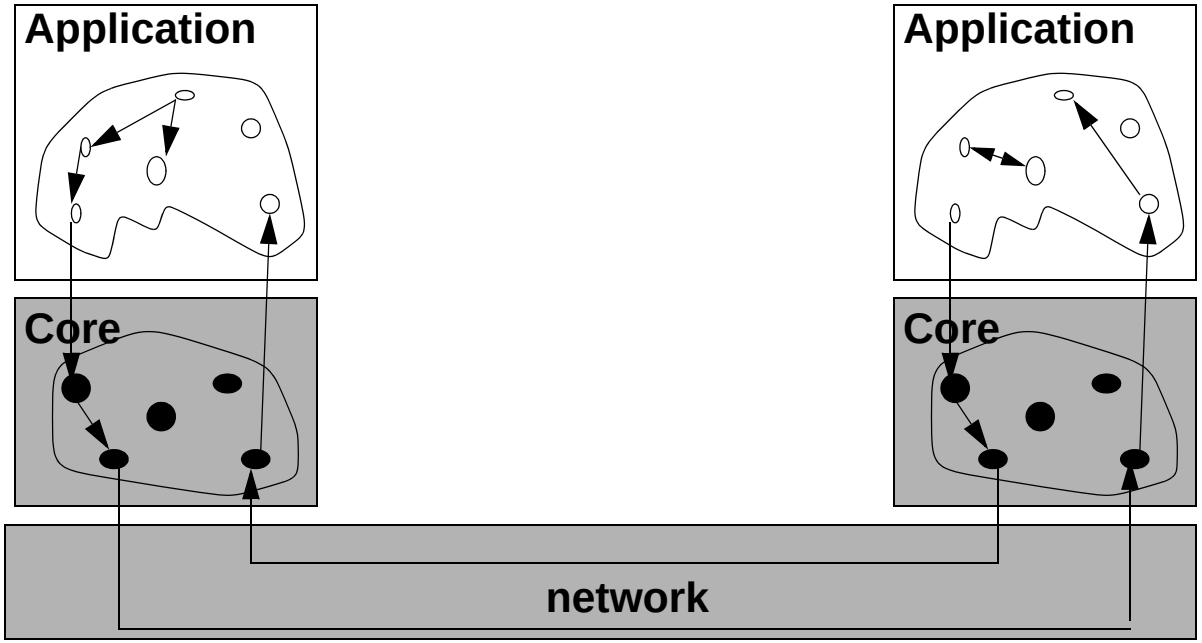
Middleware is

- **software/abstraction glue which allows separate applications to communicate TRANSPARENTLY**
- **i.e. to inter-operate independently from**
  - hardware and system devices, operating systems capabilities
  - communications infrastructure

History

- **1987 - Sun RPC**
- **1988 - Distributed Computing Environment (DCE) of Open Software Foundation (OSF)**
  - coined term “middleware”
  - incl. naming service, fault semantics, Interface Definition Language (IDL)
  - but: no object oriented model with inheritance, static binding of declared procedure, ..
- **today - Object oriented approach of Object Management Group (OMG)**
  - Object Management Architecture (OMA)
  - Common Object Request Broker Architecture (CORBA)

Object Oriented Software Development



Goals

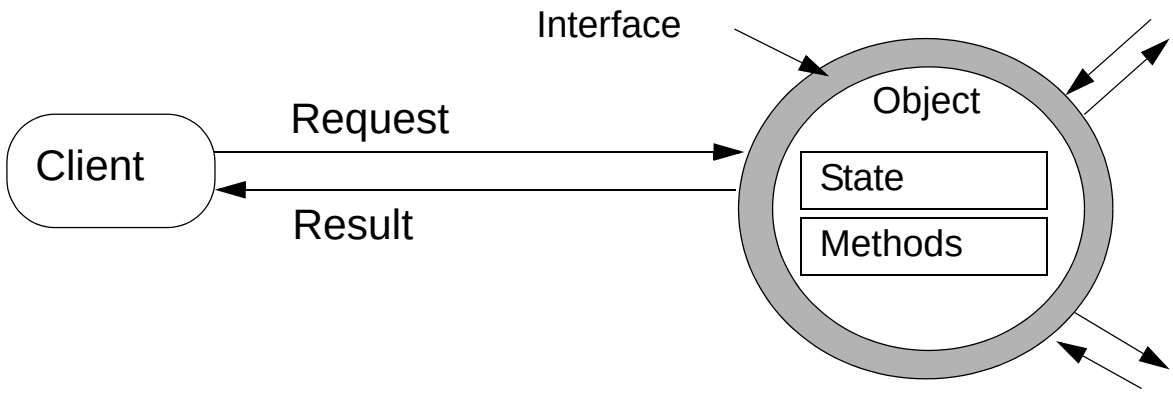
- functionality, efficiency, robustness,
- reuse, future enhancements possible

Alternative (unfortunately too often used) development methods

- “to develop from scratch”
- “Copy, Paste and Adapt individual code”
- “Combine generic parts taken from libraries”
- “Use objects, inherit from and instantiate framework components”

Objects in Distributed Systems

Abstraction



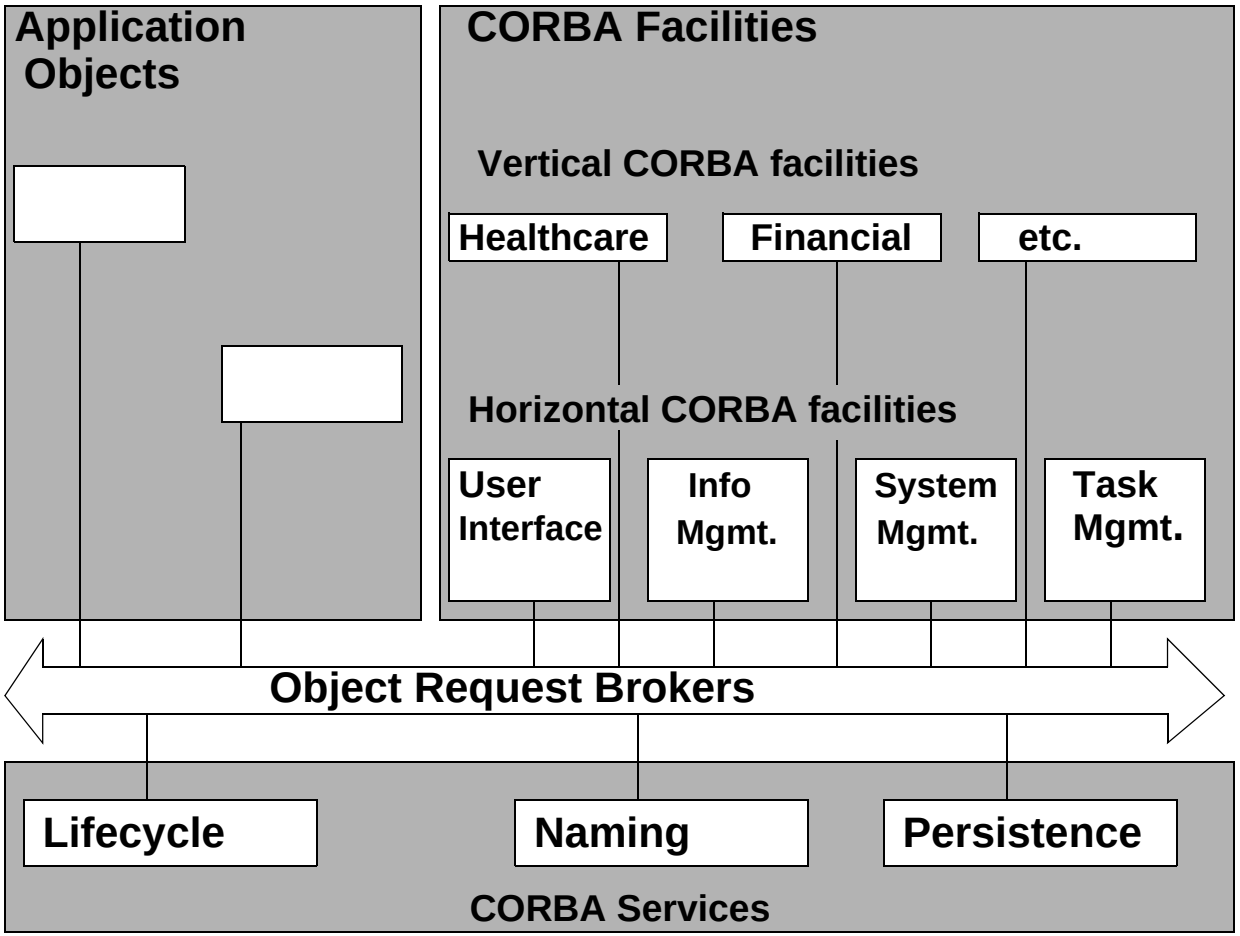
Request specifies

Target Object	Operation	Parameter	Context

Characteristics

- **object addressed via**
  - unique system wide (location independent) identifier
- **usage of Interface Definition Language IDL**
- **abstraction from local environment**
- **like the object oriented paradigm**
  - inheritance, instantiation of object classes, polymorphism

Object Management Architecture OMA



Object Management Group (OMG)

- **1989 established**
- **as independent group (more than 400 companies involved)**

Object Management Architecture OMA (2)

OMA defines components:

- **Object Request Broker (ORB)**
- **Object Services**
- **Common Facilities**
- **Application Objects**
- **Notation via: Interface Definition Language (IDL)**

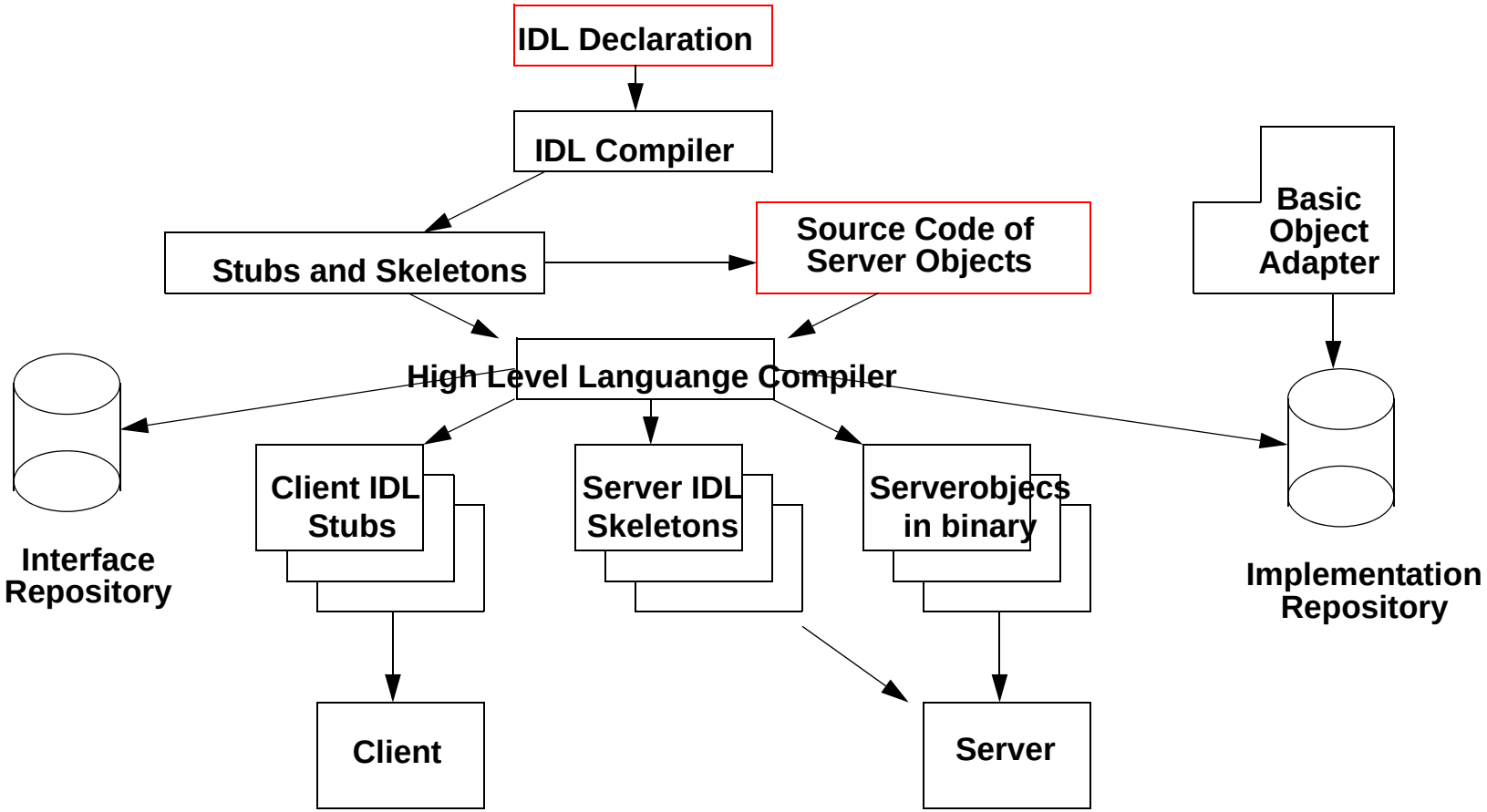
Interface Definition Language

- **Language to define the interfaces, syntax similar to C++**

Object Request Broker (ORB)

- **Client**
  - does NOT contact server directly
  - contacts object bus
- **Client does not care about**
  - transport services or protocols
  - object creation, management, storage at server side
- **Server and client similar**
  - i.e. should be the same

Interface Definition Language - Environment



Interface Repository:

- stores interface information of objects used by clients at run-time

Implementation Repository:

- allows ORB to localize and activate object implementations

6. Other: E.g. Microsoft .NET

.NET

- **Microsoft**
  - „ As a result of the changes in how businesses and consumers use the Web, the industry is converging on a new computing model that enables a standard way of building applications and processes to connect and exchange information over the Web“ – Bill Gates

.NET: is a software platform

- **new APIs and libraries**
- **.NET Framework with**
  - Common Language Runtime
  - Unified Classes
  - Application service provider: ASP.NET

.NET: authentication system

- **it is now called .NET MyServices**
- **e.g. rent software instead of license it**

.NET: standardized method by which applications can "talk" to each other

- **via XML**
- **Web Services describe the way computers can exchange information**
- **regardless of the platform on which they run**

Microsoft .NET Structure