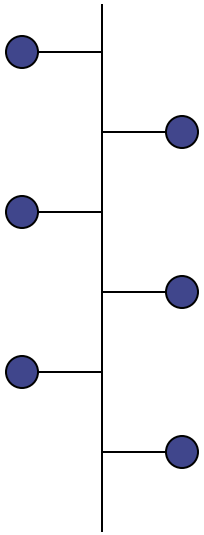
Rechnernetze und Verteilte Systeme

Introduction to Communication Networks and Distributed Systems



Unit 3: distributed Systems - Introduction

Prof. Dr.-Ing. Adam Wolisz

TKN Telecommunication Networks Group

We can now move MEANINGFULL Messages…

but how to use them?

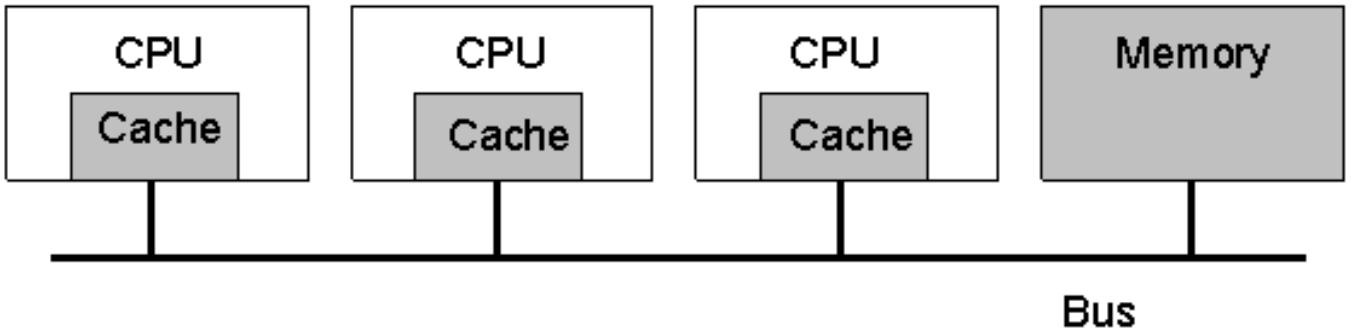
A Distributed System:

- *A distributed computing system consists of multiple **autonomous** processors that do not share primary memory but **cooperate** by sending messages over a communication network.*

  Henri Bal/ Colouris

Multiprocessors vs. Distributed systems

- A bus-based multiprocessor.

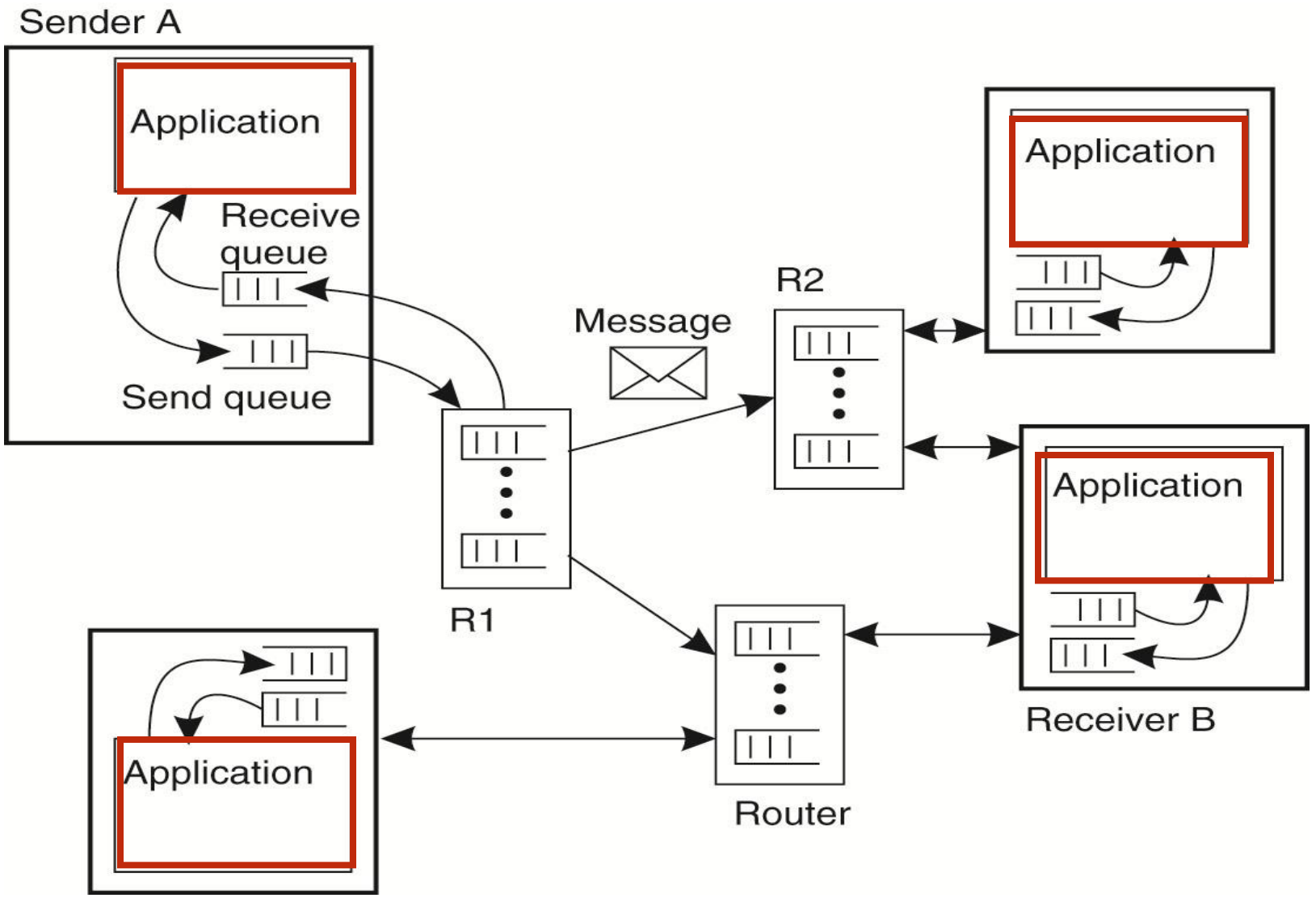


Typical features of a multiprocessor:

- Physical access to a common memory
- Resources under same management
- Very fast (bus, switching matrix) based local communication

A Distributed System

[Tanenbaum]



A Distributed System:

- *A distributed computing system consists of multiple **autonomous** processors that do not share primary memory but **cooperate** by sending messages over a communication network.*

  Henri Bal/ Colouris

- A distributed system is one in which the **failure** of a computer which you didn‘t even know existed can render your own computer unusable.

  Leslie Lamport

==========

- A distributed system is a collection of **independent** computers that **appears** to its users as a **single coherent** system.

  A. S. Tannenbaum

Two important features of Distributed Systems

- **Autonomy**
  - A distributed system consists of ***autonomous,*** independent entities (***usually*** cooperative ones, ***sometimes*** antagonistic ones)
  - Each individual entity is – typically – a full-fledged, operational system of its own
  - Individual entities might follow local policies, be subject to local constraints…

- **Transparency**
  - The fact that the distributed systems is indeed a conglomerate of different (simpler) systems is of no interest and should be of no concern to the user

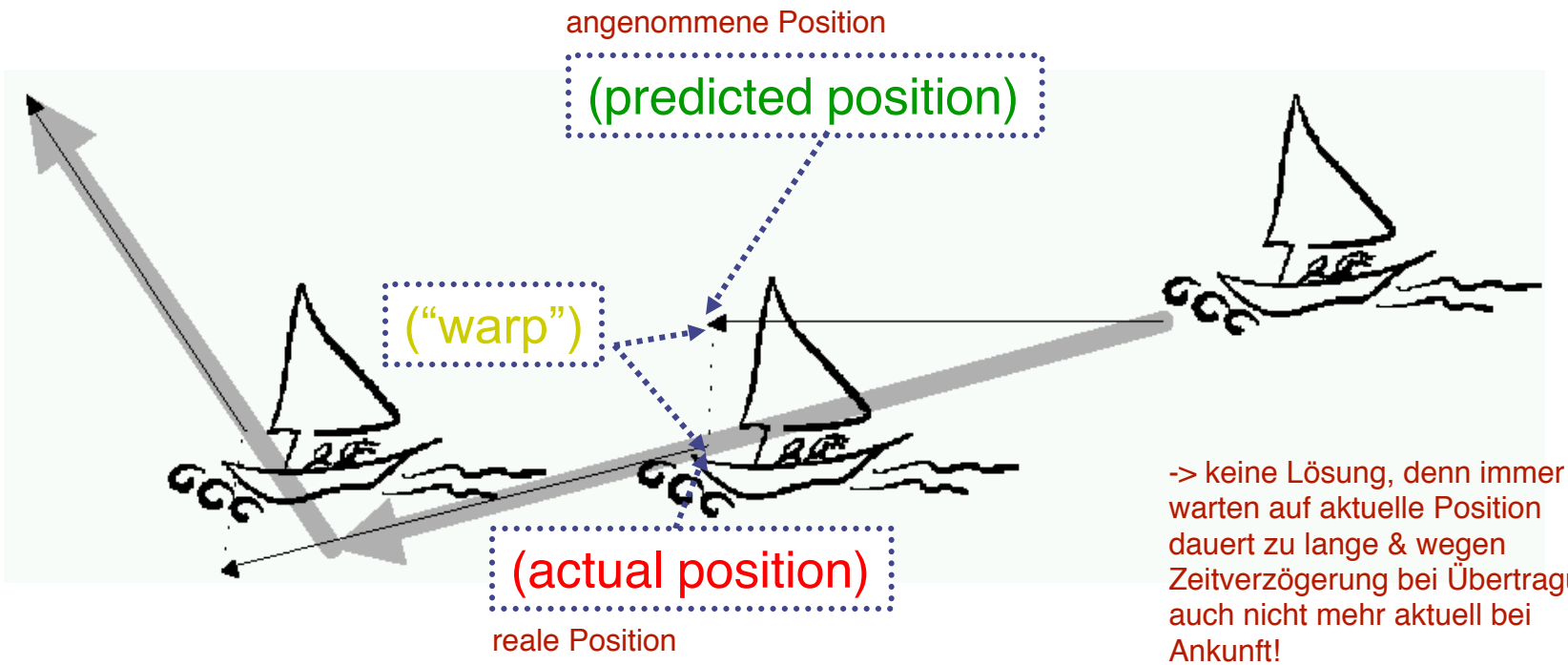
A dead Man that Shoots…

[Mouve, Mannheim, 2005]

- Consider a **distributed** action game: player A shoots player B.
  - Each of the players has a computer with a local copy of the game
  - Application of player A creates a bullet entity with a certain heading and velocity. It will then transmit the state of that bullet entity.
  - Upon receiving the state of the bullet, remote applications will start to check whether any entity that they control is hit by the bullet.
- There is network delay between the time A transmits the initial state of the bullet and the time B receives the state.
  - This network delay may be much larger (in the order of 100ms or more) than the amount of time that the bullet needs to hit its target!
  - During this time player B might take actions,
    - shoot at another player C, even though he's already dead.
    - move so that the bullet would not hit him (from his point of view!!!)
- Players A, B, and C may therefore disagree about whether a hit has been scored on B or not!

Dead Reckoning

- Based on ocean navigation techniques
- Predict position based on last known position plus direction
  - Can also only send updates when deviates past a threshold



-> keine Lösung, denn immer warten auf aktuelle Position dauert zu lange & wegen Zeitverzögerung bei Übertragung auch nicht mehr aktuell bei Ankunft!

- When prediction differs, get "warping" or "rubber-banding" effect

The Distributed Stock Exchange…

- Consider a distributed stock exchange…
  - Brokers are distributed all over the world.
  - Each of the brokers runs a computer program
  - Each of the brokers sets his program to:
    Buy stocks of Company HOPE if they fall under XXX $
- The central unit (located at Wall street, NY) defines the value of the HOPE stocks and announces them …
- Due to different transmission delays Broker John – at Wall Street – will always react quicker than Paul in Frankfurt , or Wendong in Peking, or….
- But the relative advantage of Paul vs Wendong (or vice versa!) might dramatically change depending on traffic conditions….(e.g. local soccer league…)
  - What about Paul causing artificial load on Wendong´s computer by sending him numerous superfluous requests from a computer of his girl friend?

Pitfalls when Developing Distributed Systems

falsche Annahmen über verteilte Systeme (“ideale Annahme”)

- The network is reliable.
- The network is secure.
- The network is homogeneous.
- The topology does not change.
- **Latency is zero**.
- **Bandwidth (= bit rate!) is infinite**.
- Transport cost is zero.
- There is one administrator.

in Realität: dezentral, unabhängig und gelegentlich nicht konsistent

Characteristics of decentralized algorithms

- No machine has complete information about the system state.
- Machines make decisions based only on local information.
- Failure of one machine does not ruin the algorithm.

Schwierigkeit beim Entwurf eines verteilten Systems!

- There is no implicit assumption that a global clock (i.e. precise common understanding of time!) exists.

Problem: Nicht komplett synchronisierte Zeit (auch durch immer schnellere Systeme nicht gelöst, da dadurch auch immer kleinere zeitliche Abweichungen auffallen

Design Challenges: Openness

[J. Hou, Illinois]

- Degree to which new resource-sharing services can be added & used.
- Requires *publication* of interfaces for access to shared resources.

ein offenes System erfordert offene und bekannte Schnittstellen
-> gibt auch genug Beispiele, wo das fehlt!

- Requires *uniform communication* mechanism.

Problem: proprietäre Formate

- Conformance of each component to the published standard must be tested and verified.

Design Challenges: Scalability

[J. Hou, Illinois]

- A system is said to be scalable if it will remain effective when there is a significant increase in the number of **resources** and **users**:

  Vergrößerung des Systems im laufenden Betrieb

  - Controlling the cost of resources

    skalieren auch die Kosten der Interaktion? -> sollen nicht mehr als linear mit den Nutzern steigen

  - Controlling the performance loss
  - Preventing software resources running out (e.g., IP addresses)

Transparency in a Distributed System

[Tannenbaum]

Transparency	Description
Access	Hide differences in data representation and how a resource is accessed
Location	Hide where a resource is located Cloud Dienste!
Migration	Hide that a resource may move to another location
Relocation	Hide that a resource may be moved to another location while in use
Replication	Hide that a resource is replicated
Concurrency	Hide that a resource may be shared by several competitive users
Failure	Hide the failure and recovery of a resource

Umwandlung in andere Formate unsichtbar für Nutze

Note: There is a difference between the FEATURE, MECHANISM and POLICIES
Take openness and scaling into account…➔

Different forms of transparency in a distributed system (ISO, 1995)

Feature (Eigenschaft): sichtbar, Vorhandensein nachprüfbar, die Beispiele oben (Bsp. Bremsverzögerung beim Auto)

Mechanism: Abläufe (Bsp. welche Art von Bremsen verwendet)

Policies (Strategie / Prinzip): Regeln über Aktivierung von

The client – server approach..

The Client- Server approach

-> bringt mehr Struktur in verteilte Systeme

- Is the message passing/ stream interface enough? What about more complex cooperation patterns….

Client-Server-Architektur

- The client / server approach:
  - A process wants to perform a specific action, e.g. to print a file. The local computer can not do it (there is no printer), but there is someone another computer able to do it (the printer is connected to that other computer). The other one is the server.
  - The client transmits a request message to the server (including the file to be printed), asking the server to perform the service
  - The server receives this messages and performs (probably) the appropriate action (i.e. prints the file).
  - The results are send back to the client via a reply message.

Service

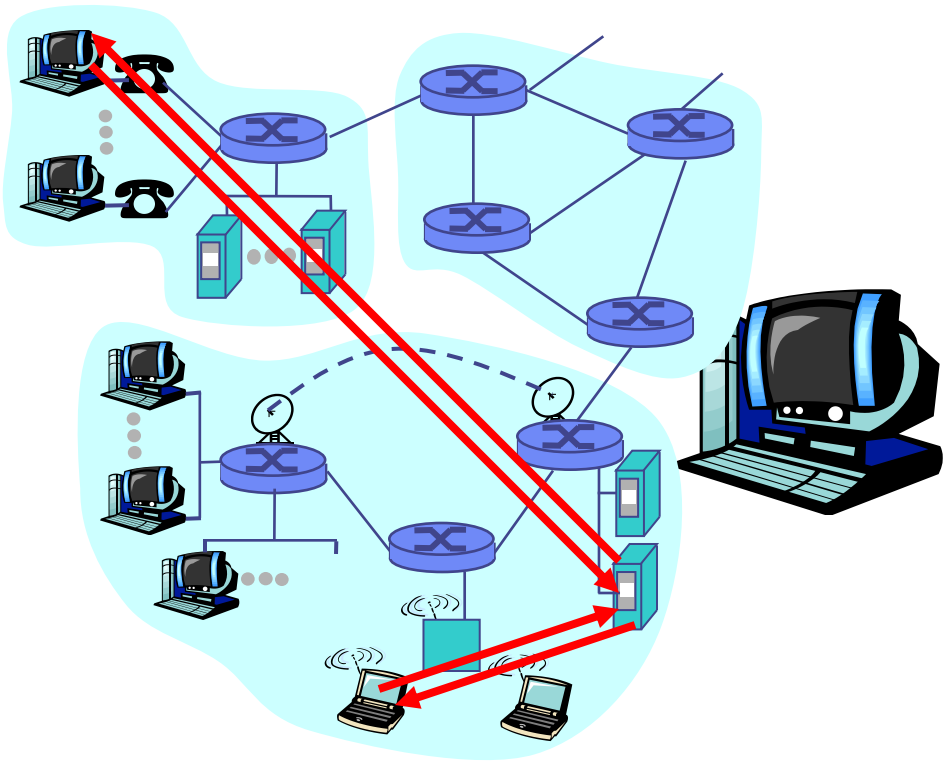
- Service: Any act or performance that one party can offer to another that is essentially intangible and does not result in the ownership of anything. Its production may or may not be tied to a physical product.

D. Jobber, Principles and Practice of Marketing

- Focus is on the *output*, the *result* of the service
- NOT the means to achieve it

Client-Server Model

nach Prinzip eines verteilten Systems



server: Server

- always-on host
- permanent IP address
- server farms for scaling

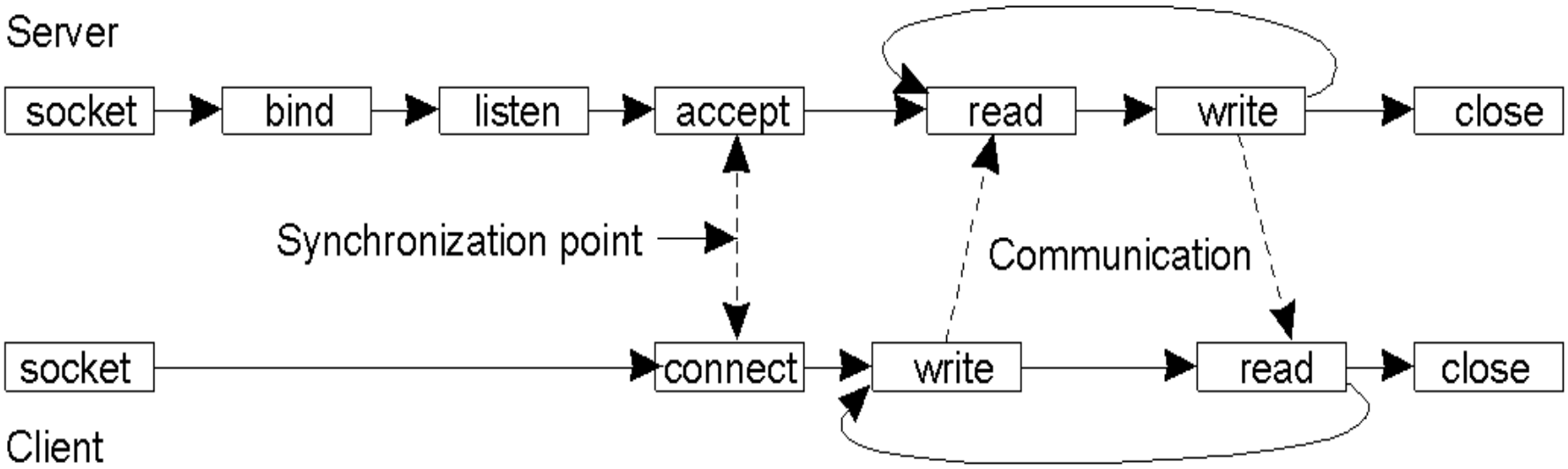
clients: PC, Smartphone, andere Geräte mit Internetzugang

- communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do not communicate directly with each other

  Kommunikation nur über Server (in der Regel)

Berkeley Sockets for Client-Server

eines verschiedener Muster zur
Kommunikation zwischen Client und Server



Remote Procedure Calls (RPC)

Prozedur = Funktion
Organisationsstruktur eines Programmes, wo Codeteile immer wieder verwendet werden können

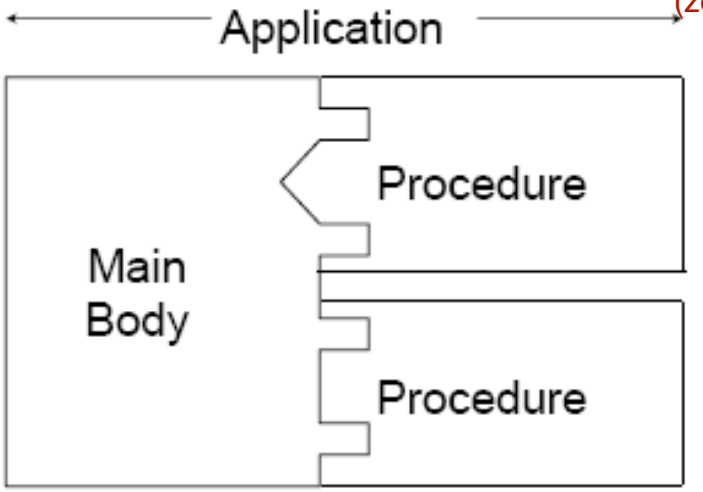
- Remote Procedural Calls are the preferred tool to implement the client-server model.
- In classical procedure calls the code of the procedure is located on the same computer (in the same address space) as the calling program, in an RPC the code is located on another computer.
- One major design goal of an RPC system is transparency: ideally the caller should not know if the callee is located locally or remotely. So in RPC we have to consider the following topics:

  Text

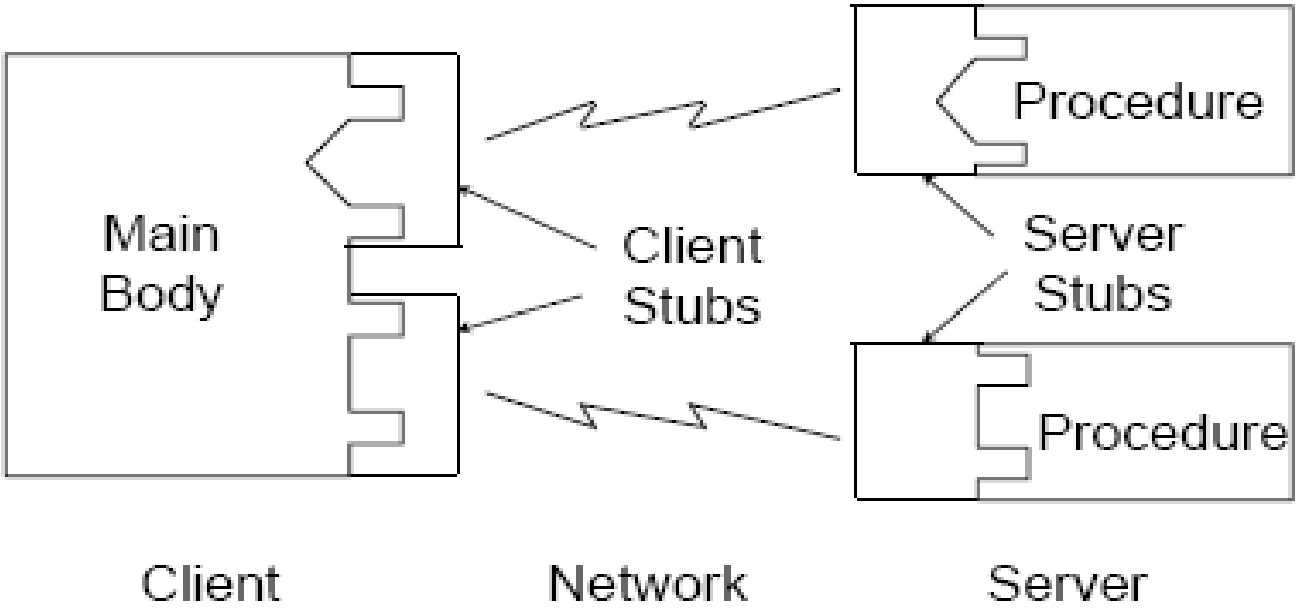
  – Parameter handling and marshalling
  – Semantics
  – Addressing
- An RPC system is attractive for the users because automatic support for the conversion from local to remote procedural call can be supported (see below).

Local vs. Remote Procedural Call

bei remote procedure Calls kann man nicht einfach mit Pointern arbeiten (zeigen in lokalen Speicher!) -> Stubs



Client Stub: tut für den Server als sei er der Client, gibt die Anfrage des Clients weiter (“Anknüpfungspunkt”)



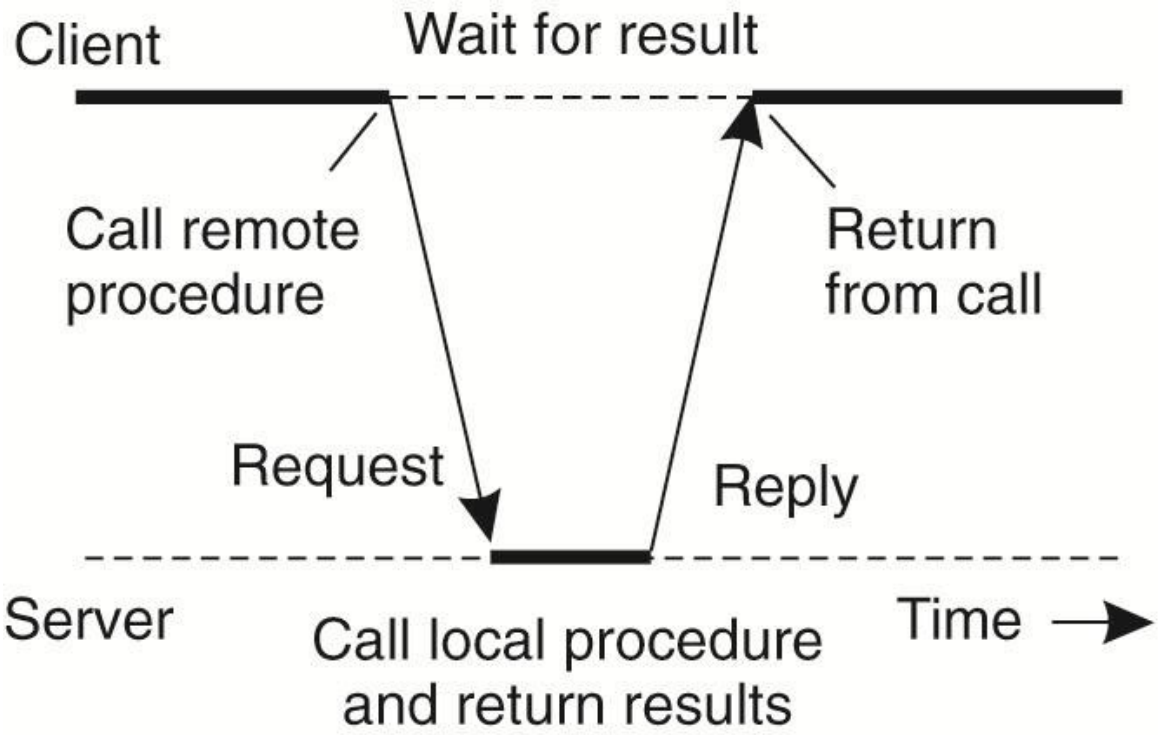
Basic RPC Operation

- The caller uses a specified procedure interface (like an ANSI-C function prototype), where all necessary procedure parameters are declared with their datatypes.
- In case of a local procedure, the callee performs the requested operation, in the RPC case the callee is a stub procedure that takes the given parameters, forms a message out of them (marshalling) and sends it to the appropriate server.
- In the server another stub procedure (also often called a skeleton procedure) receives the message, extracts the parameter values (unmarshalling) and calls the (local) procedure.
- The server stub takes the results of this call and sends them back to the client.
- The client stub returns the result extracted from the received message to its caller.

RPC Parameter Passing

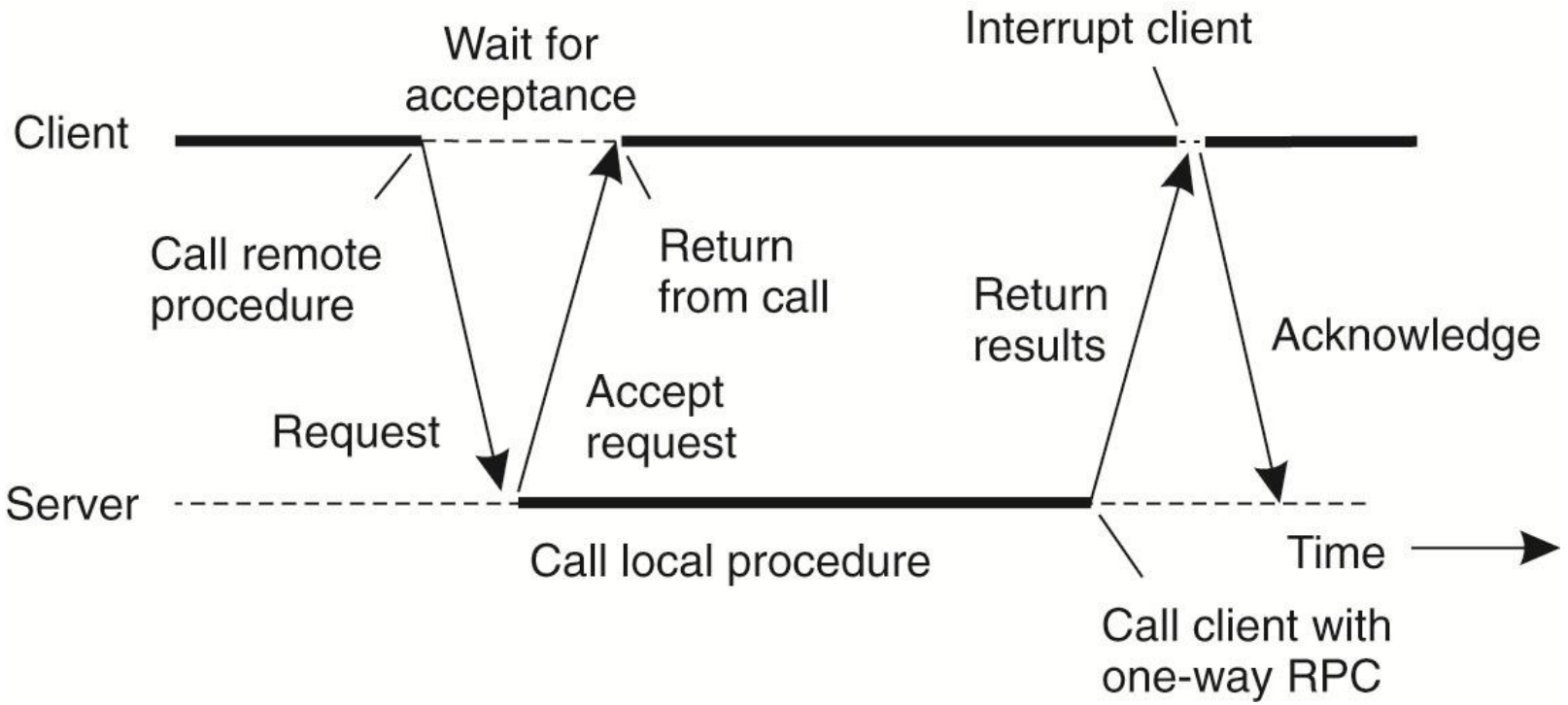
- Procedures in common programming languages have different types of parameters and calling conventions, which have to be treated in a RPC:
  - Simple call-by-value parameters are passed “as is“ (e.g. simple integer values)
  - Call-by-reference parameters are pointers; since different address spaces are used by sender and receiver, the denoted value (e.g. a buffer) has to be completely transmitted (so its length and its type must be known in advance). If the server changes some buffer values, the buffer must be retransmitted.
  - Complex data types using pointers (e.g. graphs, trees or lists) cannot (or only with difficulty) be transmitted.
- The stub procedures must use a common encoding convention for different parameter types.

“Traditional” synchronous RPC



(a)

A client and server interacting through asynchronous RPCs



(b)

Local Call vs. RPC - differences

- In the local case it is assumed that a procedure call returns correctly (unless the system fails).
- This assumption is not valid for RPC systems. Several problems can arise:
  - **Addressing (the client could not find an appropriate service)**
  - The client or the server can fail
  - **Message loss** muss bedacht und durch Systemdesign verhindert werden

- If the client could not find an appropriate server, a kind of exception handling is needed, thus violating the transparency requirement.

RPC Semantics (2)

- The client stub has three possibilities for further behavior if the result is missing:
  - He can retransmit his request until he receives a correct message (the server can be restarted or another server was found). In case that the server crashed after execution of the command, it could be executed twice. This is called „at least once semantics“.
  - He can stop after transmitting one message and report an error to the client. This is called „at most once semantics“
  - He can do anything else (e.g. make exactly 37 attempts), thus failing to give any guarantees to the client.

Idempotent operations

[mullender]

- An operation is idempotent if

1. Doing it twice has the same effect as doing it once
2. Doing it partially (several times, possibly) and then doing it whole has the same effect as doing it once

- Example: writing a block to disk
  - Doing it partially, results in a bad checksum for the block (so the block becomes unreadable)
  - Doing it whole makes the block readable
  - Doing it again doesn't matter (it's the same block again)

==========

If all RPC actions are idempotent, the RPC semantics "At least once" can be used, since every request can be repeated without harm.

wenn "At least once" verwendet werden soll, kann auch einfach dafür gesorgt werden, dass alle RCPs idempotent sind! -> Umbauen der Funktionen

Making an operation idempotent [mullender]

- Let's say you want to transfer $1000 to my (remote) bank account. Commands sent to the bank account are in blue.
- Here's the non-idempotent way:

  Add $1000 to my balance
- This is non-idempotent, because doing it twice (which I can only encourage) will give me $2000.
- The idempotent way:

  Keep trying to read my balance (idempotent) until that succeeds, call it *x*

  Add $1000 to it (a local operation)

  Keep trying to write *x*+$1000 to my balance (idempotent) until that succeeds

Features of client-server systems

- Advantages
  - The work can be *distributed* among different machines
  - The clients can access the server's functionality from a *distance*
  - The client and server can be *designed separately*
  - Simple, well known programming paradigm: call a function!
  - The server can be accessed *simultaneously* by many clients
  - Modification of the server easy …

- Disadvantages
  - Server can become a bottleneck
  - Latency in communication might be significant
  - Resources (e.g. disc space) of clients might be wasted
  - How to find e server for a given service?
  - The service might be disables if a server fails.

zu große Last auf einem einzelnen Server

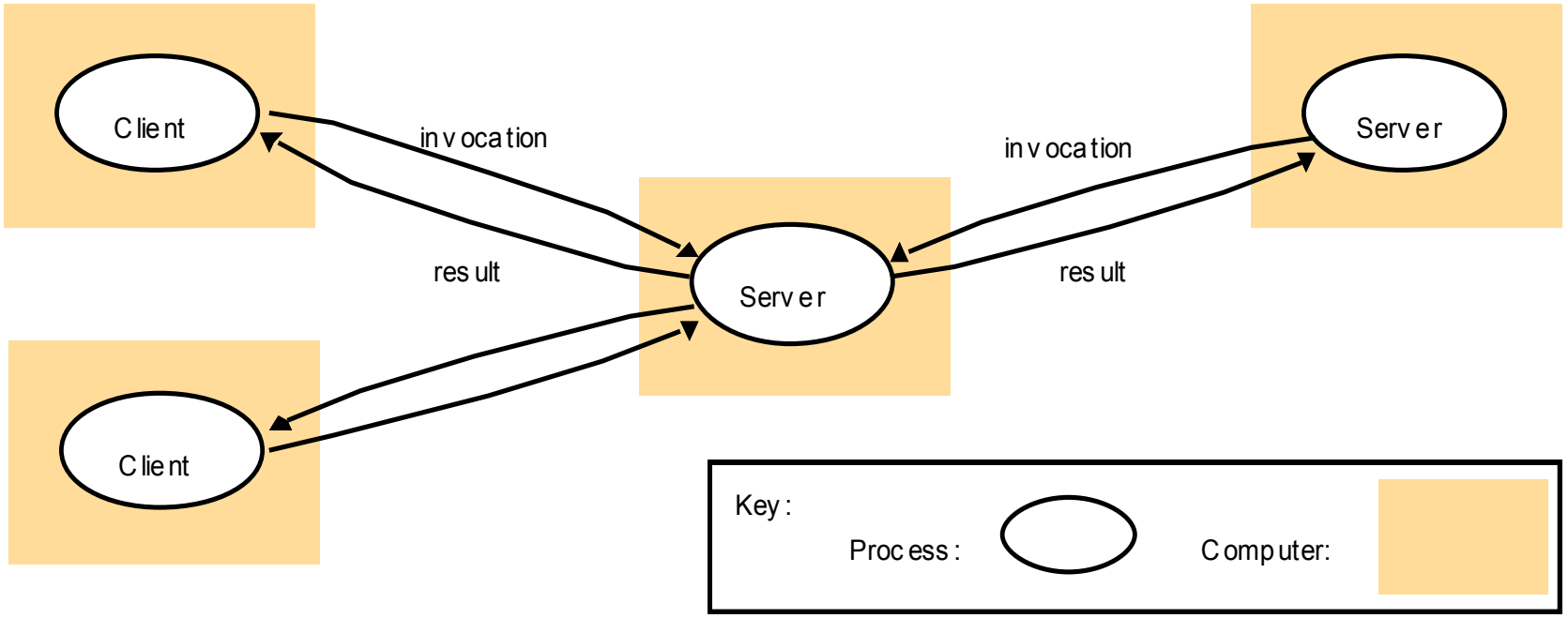
kann auch dazu führen, dass Funktionalität nicht verfügbar ist (zu kurzer Timeout!)

oder eine Funktion verschwindet, wenn der Server abgeschaltet wird!

System Architecture: Client-Server Model

[Colouris]

A server process can act as client for an other server

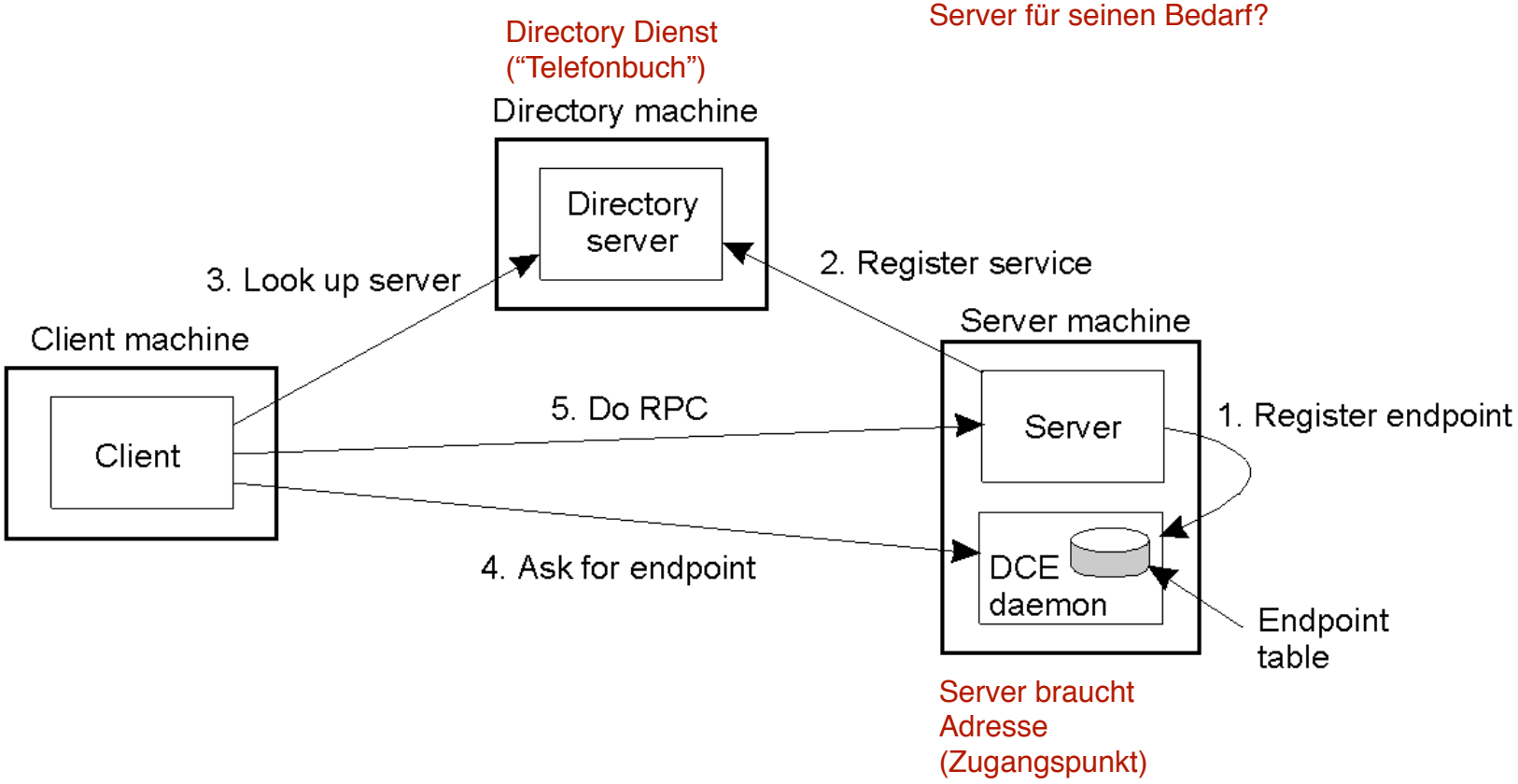


A service is not bound to a specific computer….
How to get to the right one?

Binding a Client to a Server

- Client-to-server binding in Distributed Computing Environment.

wie findet der Client den richtigen Server für seinen Bedarf?



Server braucht Adresse (Zugangspunkt)

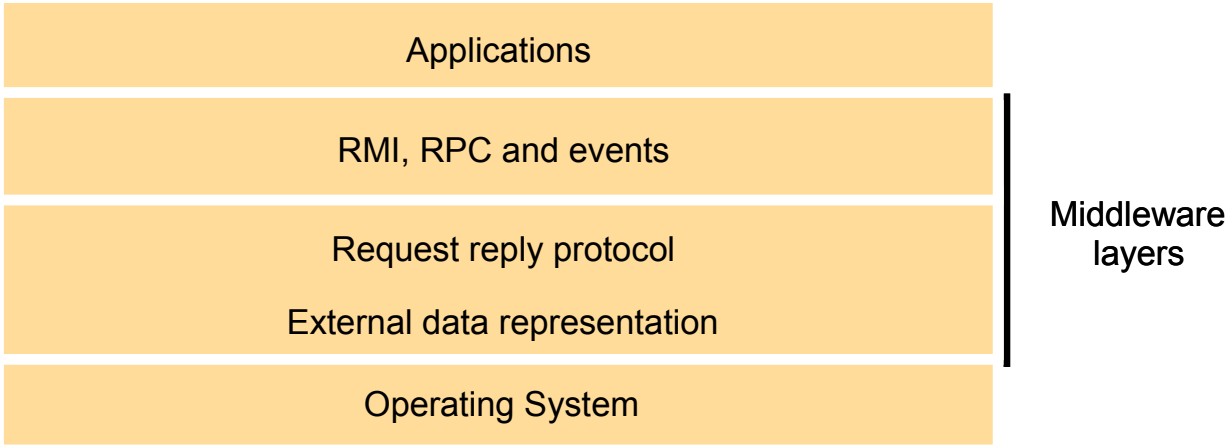
Erforderlich: Gleiche Begriffe für Dienste, damit diese gefunden werden können (vom Client, vom Nutzer…)

RPC as middleware component

[Karl,Paderborn]

Text

- RPCs present an abstracter view of a distributed system than a request/reply protocol directly realized with sockets
  - New programming model!
- Collection of software realizing such a new programming model is a ***middleware***
  - Can achieve, e.g., transparency towards location, communication protocols, hardware, operating system, different programming languages, …



An adapted communication model [Tanenbaum]