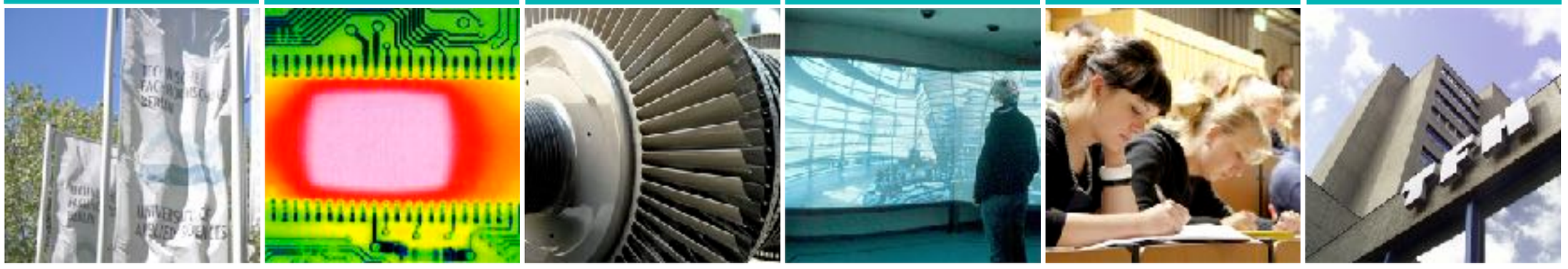
TECHNISCHE FACHHOCHSCHULE BERLIN
University of Applied Sciences

Summer School

RPC

EMR



Technische Fachhochschule Berlin

Client/Server-Modell: RPC

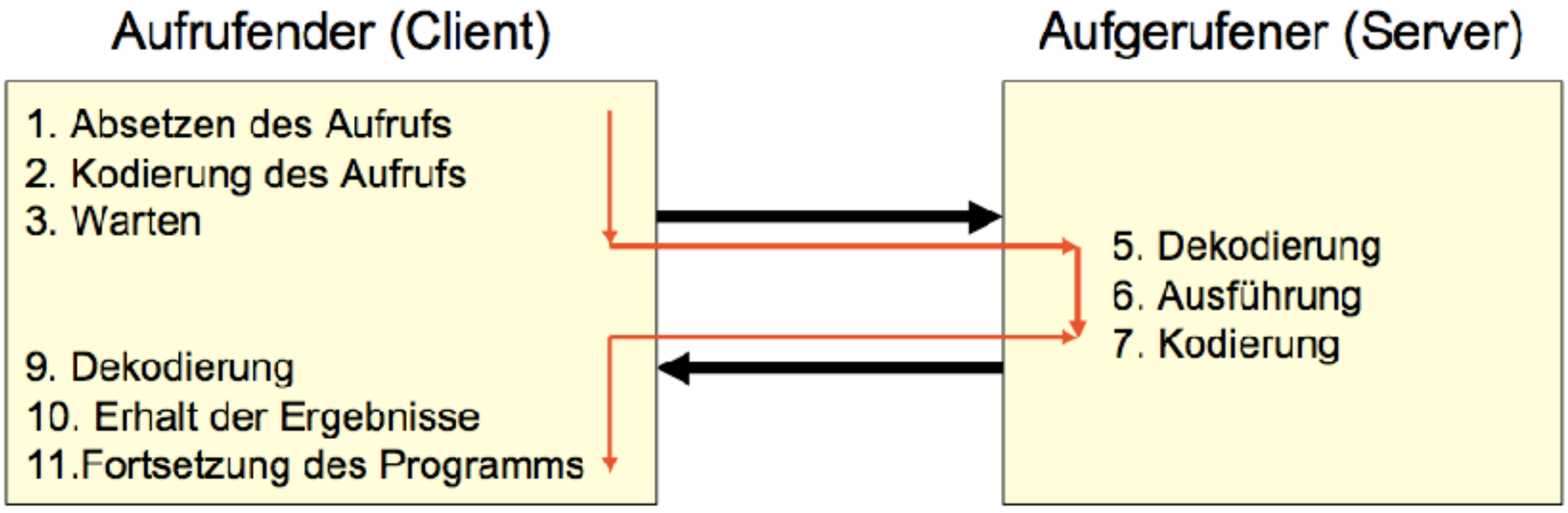
Remote Procedure Call (RPC)

- Erweiterung des Prozeduraufrufs zum Fernaufruf Vernetzung
- Ziel: Syntaktische und semantische Uniformität
  - Aufrufmechanismus
  - Sprachumfang
  - Fehlerfälle
- Definition (nach Nelson)
  - Synchrone Übergabe des Kontrollflusses
  - Ebene der Programmiersprache
  - Getrennte Adressräume
  - Kopplung über relativ schmalen Kanal
  - Datenaustausch: Aufrufparameter und Ergebnisse

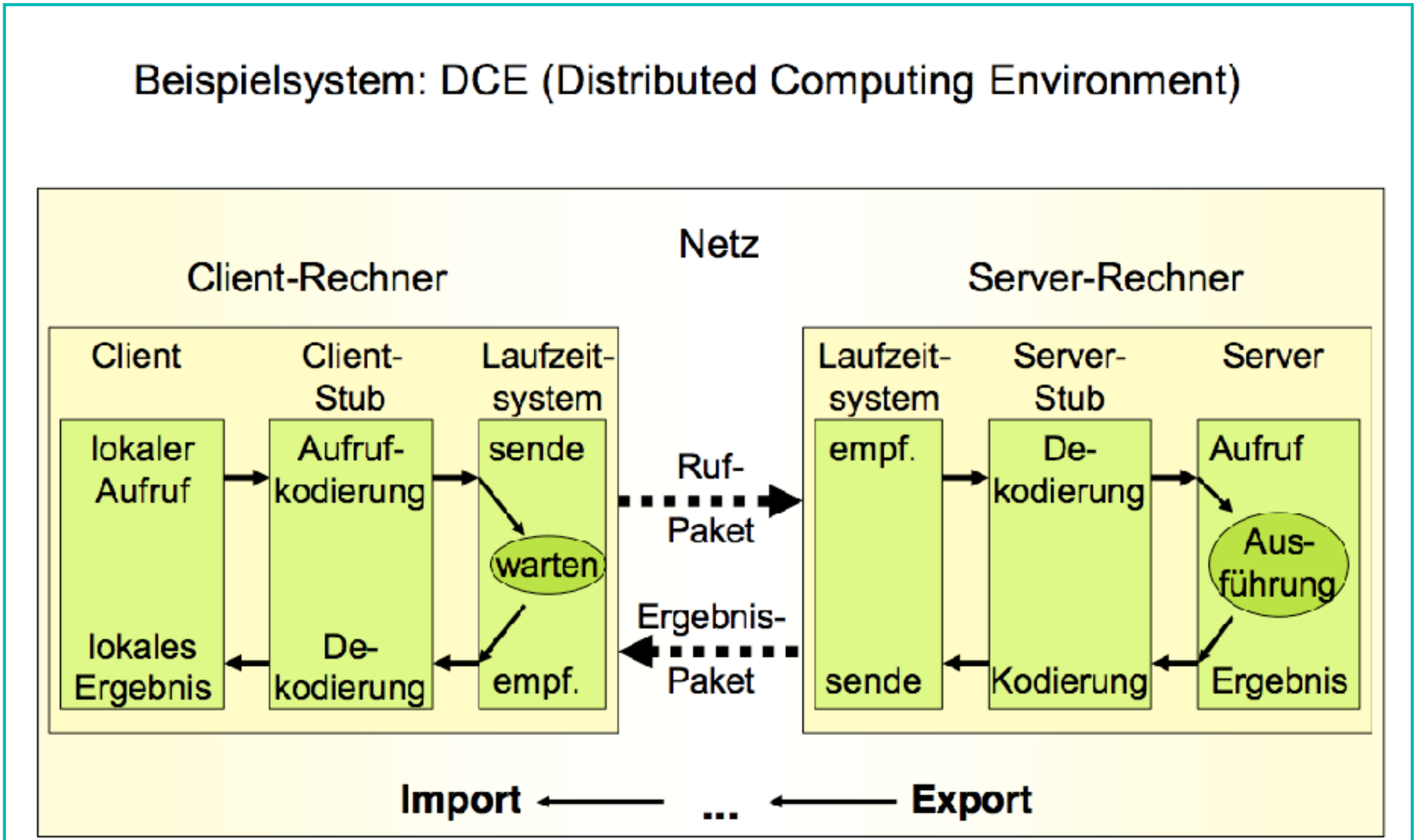
Client/Server-Modell: RPC

Remote Procedure Call (RPC)

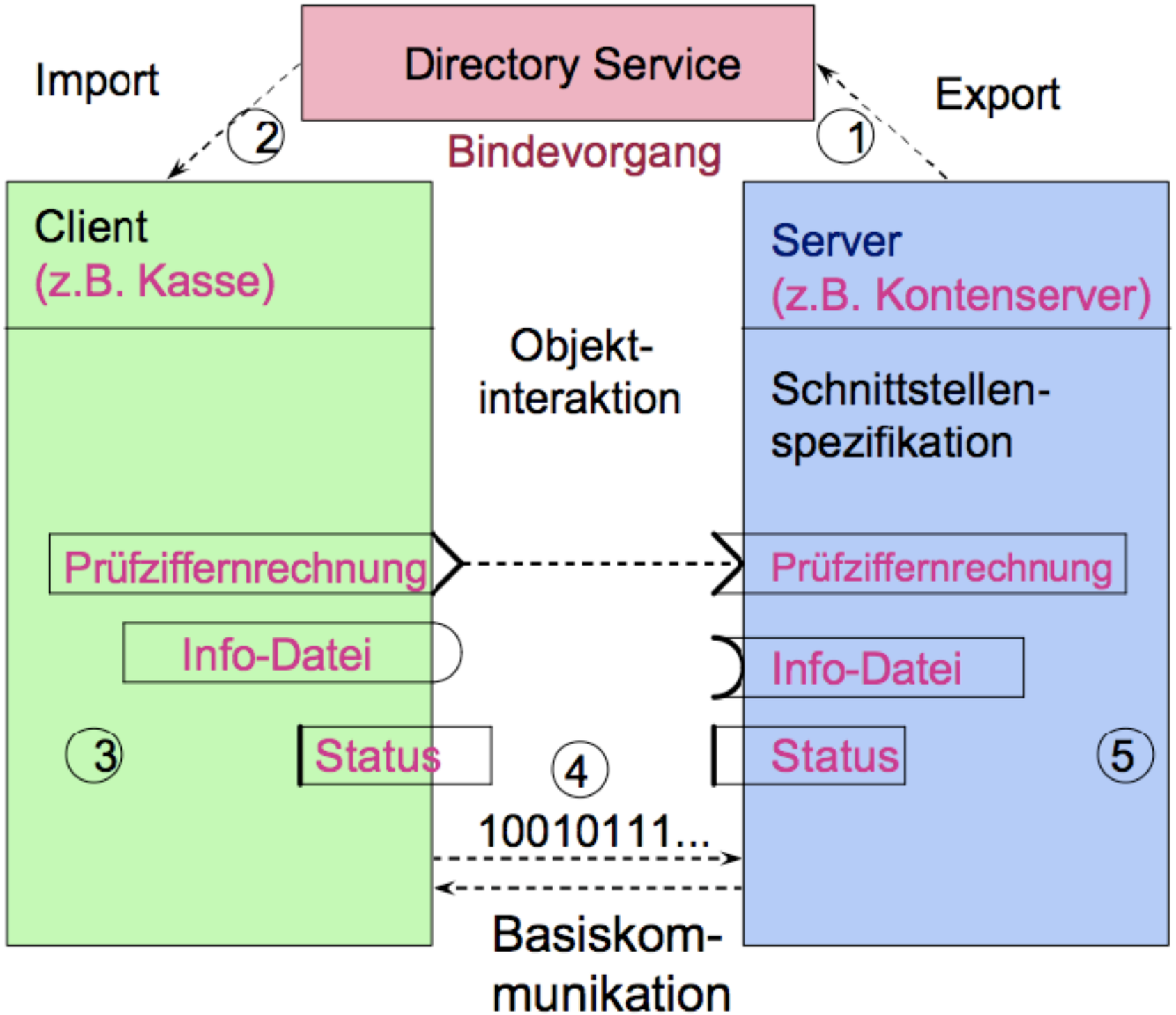
- Ablauf
  - Aufrufer im Wartezustand
  - Parameter- und Aufrufübertragung ins Zielsystem
  - Prozedurausführung
  - Rückmeldung
  - Fortsetzung der Programmausführung



Systemarchitektur



Kommunikation: Grundkonzepte



Schnittstellenbeschreibung: IDL

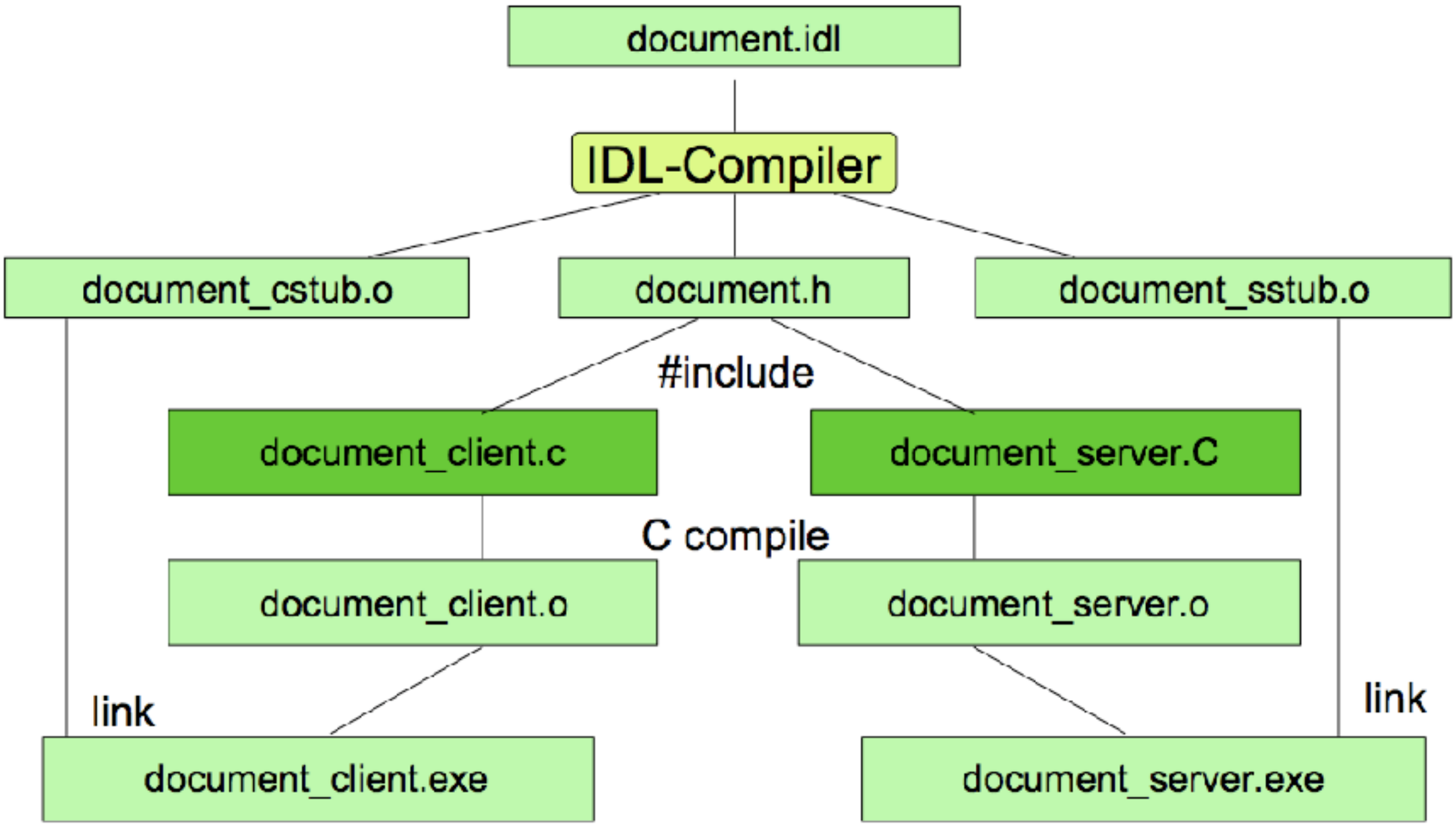
Interface Definition Language (IDL)

```
[
uuid(765c3b10-100a-135d-1568-040034e67831),
version(1.0),
]

interface DocumentServer                    // Schnittstelle für Dokumenten-Server
{
    import "globaldef.idl";                 // Import allg. Definitionen
    const long maxDoc=10;                   // Maximale Produktanzahl
    typedef [string] char *String;          // Datentyp für Character-Strings

    typedef struct {
        String documentName;                // Dokumentname
        String documentDescription;         // Textuelle Beschreibung
        long size;                          // Speicherumfang
    } DocumentDescription;                  // Dokumentbeschreibung

    typedef struct {
        DocumentDescription desc;           // Dokumentbeschreibung
        String header;                      // Dokumentkopf
        char *data;                         // Dokumentdaten
    } Document;                             // Dokument
```

Schnittstellenbeschreibung: IDL

```
[idempotent] long documentQuery (                  // Anfrage nach Dokumenten
        [in] String documentName[maxDoc],          // Dokumentnamen
        [out] DocumentDescription *dd[maxDoc],     // Beschreibungen
        [out] long *status);                       // Statuswert der Operation

long insertDocument (                              // Dokument einfügen
        [in] Document *d,                          // Neues Dokument
        [out] long *status);                       // Statuswert der Operation

long removeDocument (                              // Dokument löschen
        [in] String name,                          // Dokumentname
        [out] long *status);                       // Statuswert der Operation

long fetchDocument (                               // Beschaffen eines Dokuments
        [in] DocumentDescription *dd,              // Dokumentbeschreibung
        [out] Document *d);                        // Gesuchtes Dokument
```

Stub-Generierung

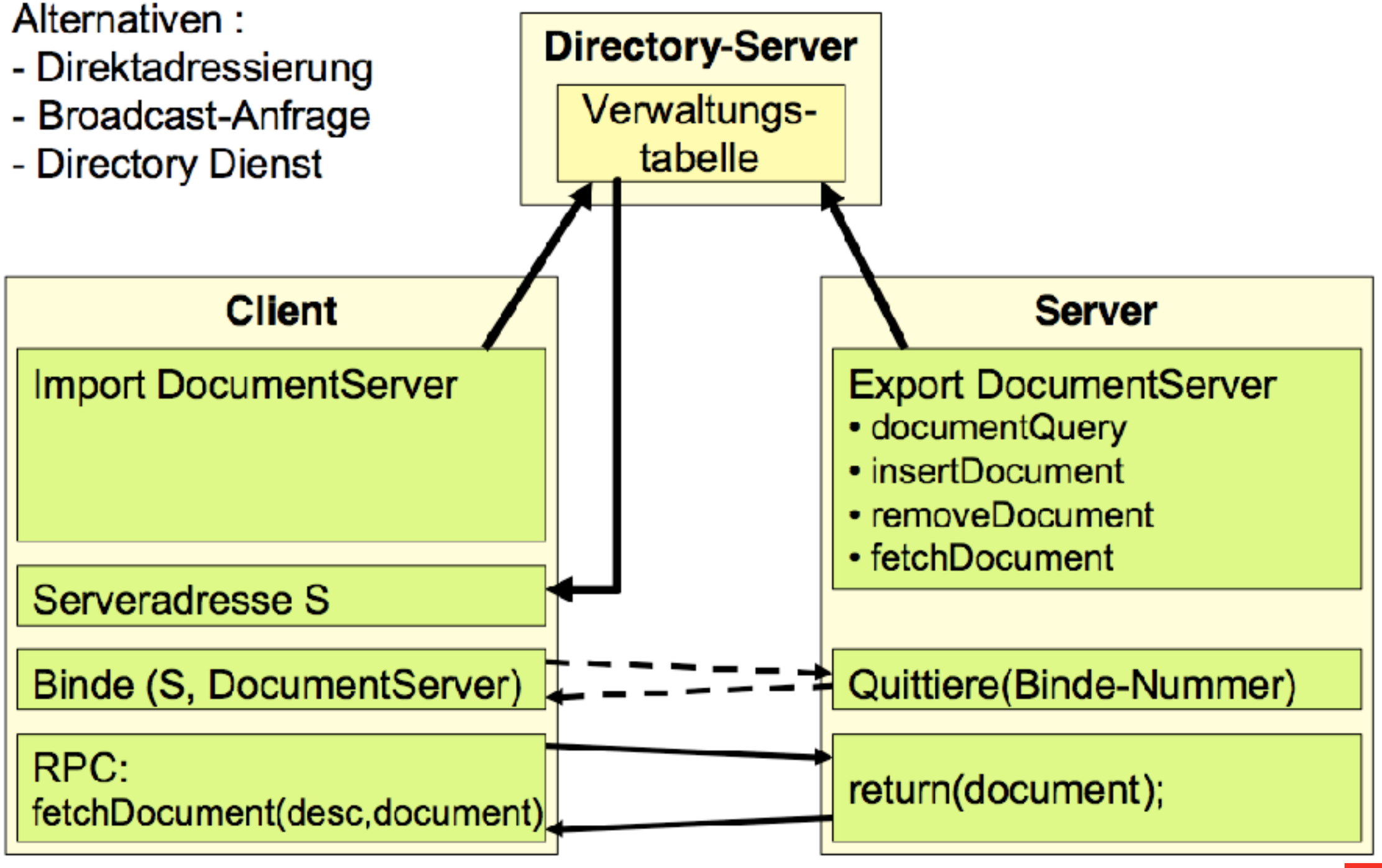


Variable Parametergrößen

```
typedef struct {
    DocumentDescription desc;       // Dokumentbeschreibung
    String header;                  // Dokumentkopf
    char *data;                     // Dokumentdaten
    long numSubdoc;                 // Anzahl der Teildokumente
    [size_is(numSubdoc)] ComplexDocument *subDocument[*];
                                    // Variables Array von Teildokumenten
} ComplexDocument;                  // Komplex strukturuertes Dokument
```

Bindevorgang

Alternativen :
- Direktadressierung
- Broadcast-Anfrage
- Directory Dienst



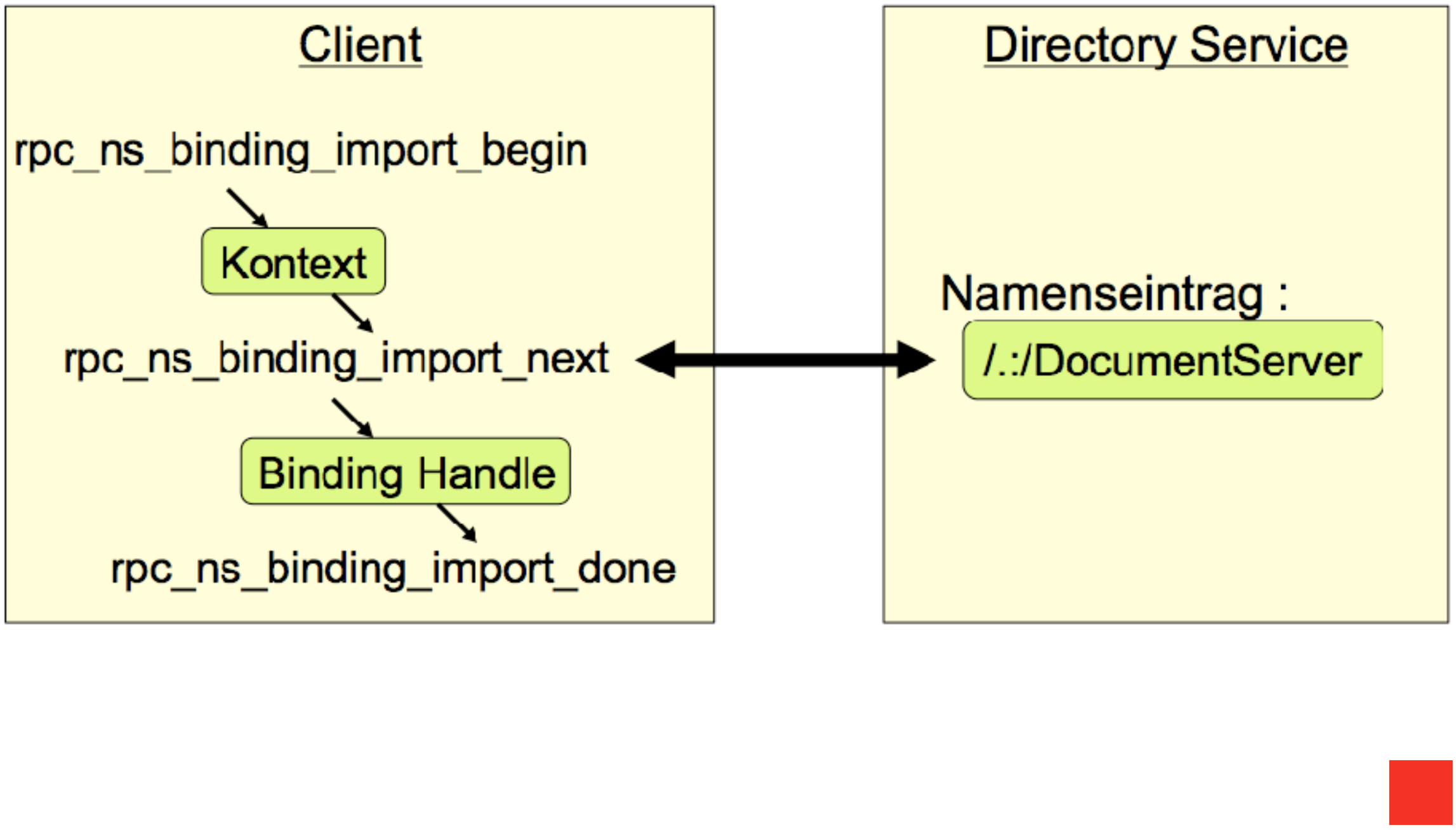
Exportieren durch Server

```
#include "DocumentServer.h"                  // vom IDL-Compiler erzeugt
#define entryName "/.:/DocumentServer"       // Name des Eintrags im DS
main()
{   unsigned status;                         // Status der Aufrufe
    rpc_binding_vector_t *bVec;              // Binding-Vektor
    // ... lokale Initialisierungen
    // *** Ermittle den Vektor der vorhandenen Binde-Kennungen: ***
    rpc_server_inq_bindings(&bVec, &status)
    // .. weitere Initialisierungsoperationen
    // ** Exportiere die Schnittstelle an den Directory Service : ***
    rpc_ns_binding_export(rpc_c_ns_syntax_default, entryName,
                          DocumentServer_v1_0_s_ifspec, bVec, NULL, &status);
    // ....
}
```

Importieren/Aufruf durch Client

```
setenv RPC_DEFAULT_ENTRY "/.:/DocumentServer"

#include "DocumentServer.h"          // Enthält u.a. „fetchDocument“

main()
{
    Document d;                      // zu beschaffendes Dokument
    DocumentDescription dd;          // Dokumentbeschreibung
    int status;                      // Statuswert

    inputDocumentDescription(&dd);          // Eingabe einer Dokumentbeschr.
    status = fetchDocument(&dd, &d);        // RPC; leitet automat. Bindevorg. ein
    if (status == OK) printDocument(&d);    // Ausdrucken des erh. Dokuments
}
```

Explizites Binden durch den Client

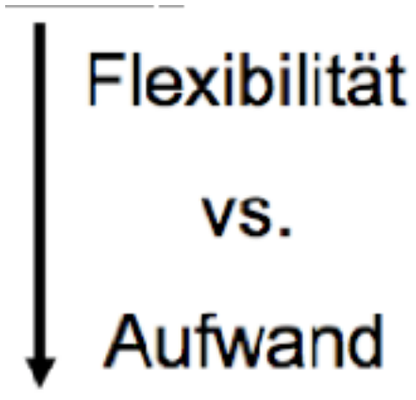


Explizites Binden durch den Client

```
#define entry "/.:/DocumentServer"          // Name beim Directory
main() {
    unsigned                status;         // Status der einzelnen Aufrufe
    rpc_ns_handle_t         context;        // Directorykontext
    rpc_binding_handle_t    binding;        // Gesuchtes Binding Handle
// *** Etabliere Directorykontext für den Bindevorgang: ***
    rpc_ns_binding_import_begin
            (rpc_c_ns_syntax_default, entryName,
             DocumentServer_v1_2_c_ifspec, NULL, &context, &status);
// *** Suche nach exportierendem Server: ***
    rpc_ns_bindung_import_next ( context, &binding, &status);
// *** (ggf. wiederholter Aufruf) ***
// *** Beende Interaktion mit dem Directory Service : ***
    rpc_ns_binding_import_done ( &context, &status);
// ** Prozeduraufruf mit explizitem Binden : ***
    status = DocumentQuery(binding, ...);
}
```

Binden: Details

- Caching von Binde-Information
  - Information auf Client-Seite global für alle Prozess
  - Erkennung veralteter Information (z.B. Timeout)
  - Begrenzte Informationshaltung auf Serverseite (Skalierbarkeit, Wiederanlauf)
- Zeitpunkt des Bindens
  - Übersetzungszeitpunkt
    - Linkzeitpunkt
    - Dynamisch
    - Gemischte Verfahren
      - Logische Namen
      - Erste Lokalisierung beim Binden zur Initialisierungszeit
      - Erneutes Lokalisieren bei Fehlern



Laufzeitunterstützung: Kommunikation

- Standard-Transportprotokolle
  - TCP/IP oder UDP/IP
  - Fehlerbehandlung, Reihenfolgetreue, Duplikaterkennung

- Spezielle Transportprotokolle
  - Kein Verbindungsaufbau (nur implizit) -> Antwortzeit
  - aktiver / passiver Verbindungszustand
  - Begrenzte Speicherung von Sequenznummern
  - Nur implizite Verbindungsauflösung (Timeout)
  - keine Zuordnung eigener Prozesse zu Verbindungen
    (Verbindungen nur pro Maschine -> weniger Verbindungen)

Transportprotokolle bei DCE

- TCP / IP
- UDP / IP
- Herstellerspezifisch

Beispiel:

```
rpc_server_use_protseq("ncacn_ip_tcp",
                       rpc_c_protseq_max_reqs_default, &status);

rpc_server_use_all_protseq
                       (rpc_c_protseq_max_reqs_default, &status);
```

Laufzeitunterstützung: Prozesse

- Prozessverwaltung
  - Zuordnung von Prozessen zu Prozedurausführungen, Lösung der Verklemmungsproblematik
  - “Lightweight” Prozess (Threads) :
    - Gemeinsamer Adreßraum
    - Schnelle Erzeugung und Prozessumschaltung
    - Große Anzahl von Prozessen möglich
      -> Einsatz bei RPC - Server - Implementierungen
      -> Einsatz auf bei Client, asynchron
  - Prozesszuordnung
    - Prozesserzeugung pro Aufruf oder
    - Prozess - Pool
  - Pufferweitergabe über Referenzen über weitere Protokollschichten
    -> Effizienz

Prozessverwaltung bei DCE

- Anlegen eines Prozesspools bei Server - Initialisierung

```
#define maxConcCalls 3
  rpc_server_listen ( maxConcCalls, & status);
    // max. Anzahl nebenläufiger Anrufe)
```

Prozesse auf Clientseite

- Prozesserzeugung :
  - expizit durch pthread_create
  - Übergabe von Startroutine und Parametern
  - Felder von Prozessen möglich
- RPC :
  - separat in Threads eingebettet
  - Rückgabewert über pthread_exit
- Synchronisation
  - blockierendes pthread_join
  - separat für alle threads

Prozesse auf Client-Seite

```
setenv RPC_DEFAULT_ENTRY "/.:/DocumentServer"

#include "DocumentServer.h"                  // Enthält u.a. „fetchDocument“
#define maxpar 3                             // Maximale Anz. paralleler Aufrufe

void docThread (pthread_addr_t arg) {        // Implementierung eines Threads
    int status;                              // Statuswert
    Document *d = malloc(sizeof(Document));  // Zu beschaffendes Dokument
    DocumentDescription *dd  = (DocumentDescription* ) arg;
                                             // Übergebene Dokumentbeschreibung
    status = fetchDocument(dd,d);            // RPC; leitet autom. Bindevorgang ein
    if (status != OK) exit (-1);             // Fehlerfall
    pthread_exit((pthread_addr_t) d);        // Rückgabe des Dokuments als Ergebnis
}
```

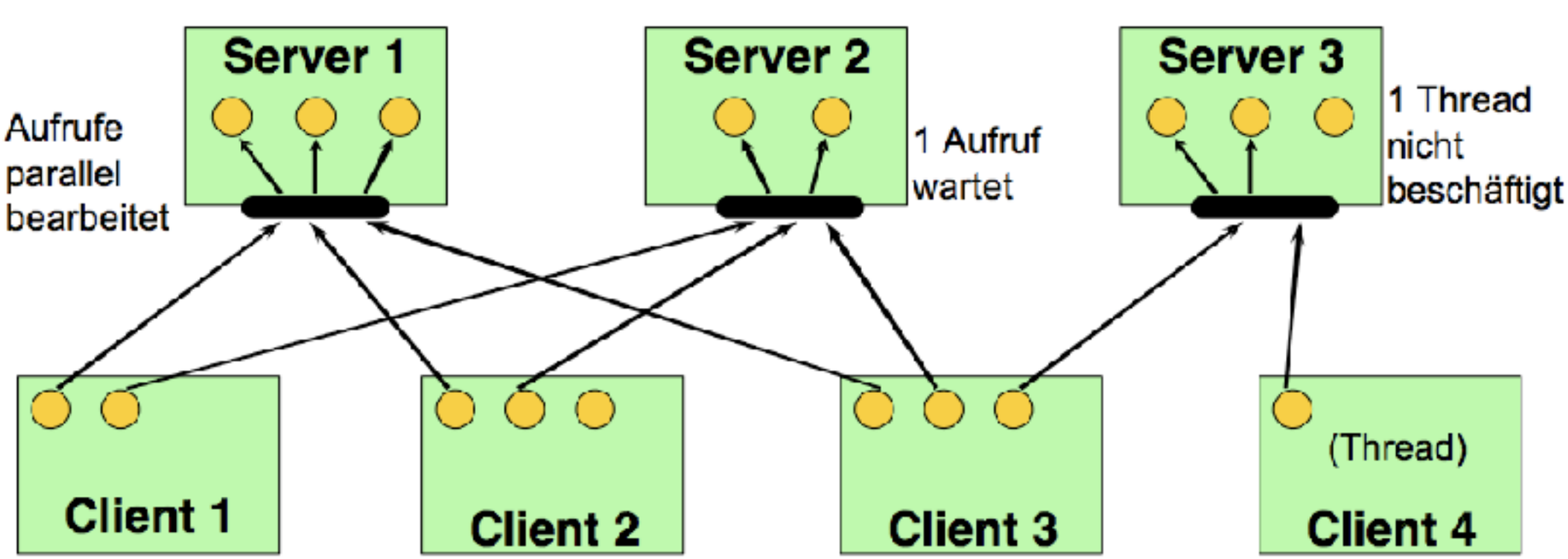
Prozesse auf Client-Seite

```
main()   {
    Document *d[maxpar];                        // zu beschaffende Dokumente
    DocumentDescription dd[maxpar];             // Dokumentbeschreibungen
    pthread_t thread [maxpar];
    int i;
    inputDocumentDescription (dd);     // Eingabe von Dokumentbeschreibungen
    for (i=0; i<maxpar; i++)
            pthread_create (&thread[i], pthread_attr_default, docThread,
                                                        (pthread_addr_t) &dd[i]);
                                        // Erzeugung der Bearbeitungsthreads
    for (i=0; i<maxpar; i++)  {                 // Warten auf alle Ergebnisse
            pthread_join (thread[i], &d[i]);    // Entgegennahme des Ergebnisses
            printDocument (d[i]);               // Ausdrucken des erh. Dokuments
    }
}
```

Einsatz von Threads: Gesamtsicht

- Client - Seite: Aufruf an mehrere Server gleichzeitig
- Server - Seite: Bearbeitung mehrerer Aufrufe

Beispiel:



Fehlerbehandlung beim RPC

- Fehlerfälle :
  - Fehler während der Prozedurabarbeitung
  - Übertragungsfehler
- Fehlerursachen :
  - Ausfall eines beteiligten Rechners
    - Server-Seite -> endloses Warten des Clients -> Timeout
    - Client-Seite -> Weitere Bearbeitung als “Orphan”
  - Unerreichbarkeit des Zielknotens -> dynamisches Binden
  - Veraltete Prozedur - Export - Information -> dynamisches Binden

Fehlerbehandlung beim RPC

- Fehlersemantic (Spector) :
  - Maybe
    - Höchstens einmalige Ausführung ohne Benachrichtigung im Fehlerfall -> nur für “unwichtige” Operationen
  - At - least - once
    - Mindestens einmalige Ausführung
    - nur bei idempotenten Operationen
  - At - most once :
    - Erkennung und Löschen von Duplikaten
    - Ausführung nur, wenn keine Rechnerausfälle vorliegen
  - Exactly - once
    - Ausführung genau einmal
    - auch bei Rechnerausfällen -> Transaktionskonzepte mit Wiederanlauf von Komponenten

Fehlersemantik

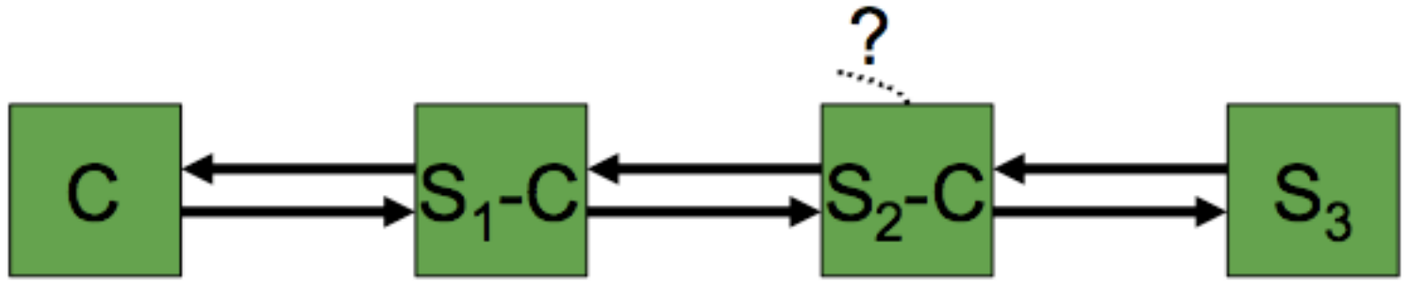
Fehlerarten / Fehlerklassen	Fehlerfreier Ablauf	Nachrichten-verluste	Zusätzlich Ausfall des Servers	Zusätzlich Ausfall des Clients
Maybe	Ausführg.: 1 Ergebnis : 1	Ausführg.: 0/1 Ergebnis : 0/1	Ausführg.: 0/1 Ergebnis : 0/1	Ausführg.: 0/1 Ergebnis : 0/1
At-Least-Once	Ausführg.: 1 Ergebnis : 1	Ausführg.: >=1 Ergebnis : >=1	Ausführg.: >=0 Ergebnis : >=0	Ausführg.: >=0 Ergebnis : 0
At-Most_Once Only-Once-Type-1	Ausführg.: 1 Ergebnis : 1	Ausführg.: 1 Ergebnis : 1	Ausführg.: 0/1 = 0/1 Ergebnis : 0/1	Ausführg.: 0/1 Ergebnis : 0
Exactly-Once Only-Once-Type-2	Ausführg.: 1 Ergebnis : 1	Ausführg.: 1 Ergebnis : 1	Ausführg.: 1 Ergebnis : 1	Ausführg.: 1 Ergebnis : 1

Fehlersemantik beim DCE RPC

Attribut	Semantik
default: at-most-once	Max. einmalige Ausführung mit Rückgabewert und ggf. expliziter Fehlermeldung
idempotent	Ggf. mehrmalige Ausführung mit Rückgabewert und ggf. expliziter Fehlermeldung
maybe	Ggf. mehrmalige Ausführung ohne Rückgabewert und ohne Fehlermeldung
broadcast	Ggf. mehrmalige Ausführung bei allen passenden Servern mit Rückgabewert des ersten abgeschlossenen Aufrufs

Probleme des RPC

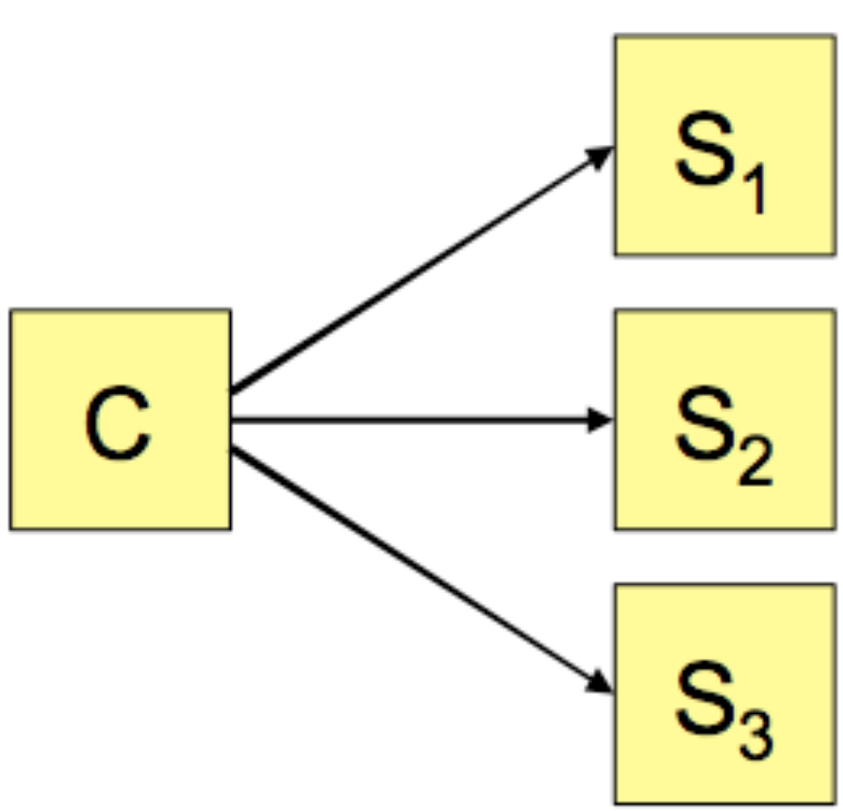
- Übertragung großer Datenmengen
  - Synchroner Mechanismus -> kleine Übertragungseinheit
  - Keine verbindungsorientierte Übertragung
  - Keine Flusskontrolle und Pufferung
  - Antwortzeit- statt Durchsatzoptimierung



- Verkettung von Bearbeitungseinheiten
  - Strenge Client- / Server Semantik
  - Keine vorzeitige Datenweitergabe
  - Kein direkter Kontrolltransfer über mehr als zwei Partner

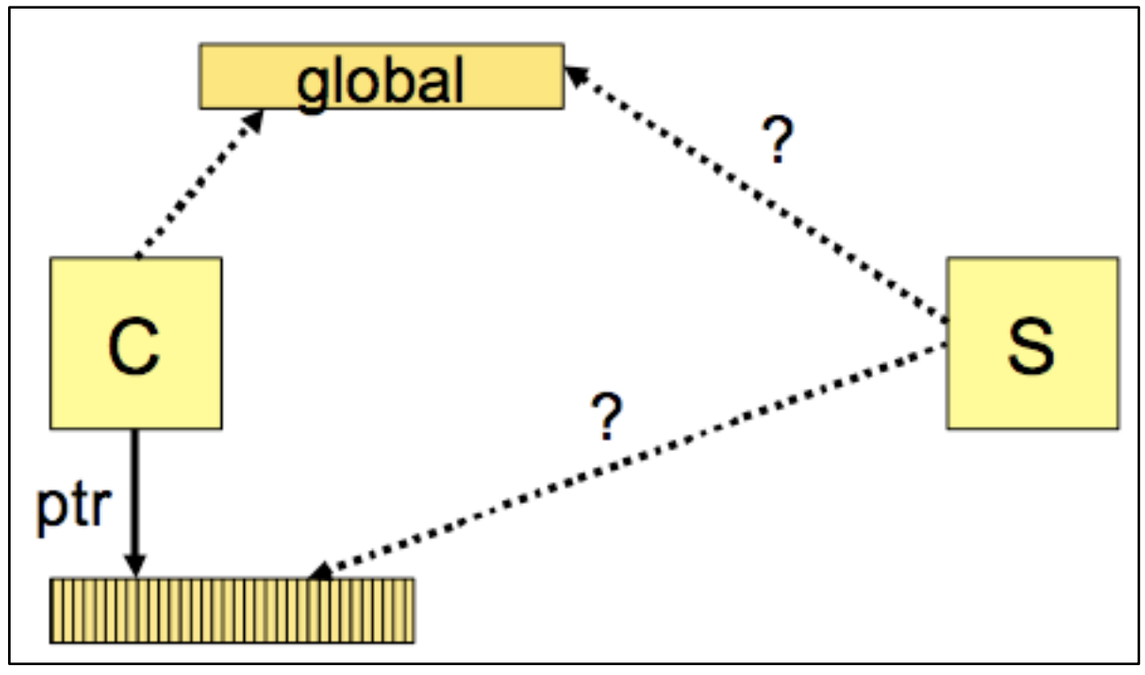
Probleme des RPC

- Vertauschung von Client- / Server-Rollen
  - Keine gleichberechtigten Kommunikationspartner
  - Keine zwischenzeitlichen Ergebnisrückmeldungen
  - Keine Rückfragen
- Multicast / Broadcast nicht unterstützt



Probleme des RPC

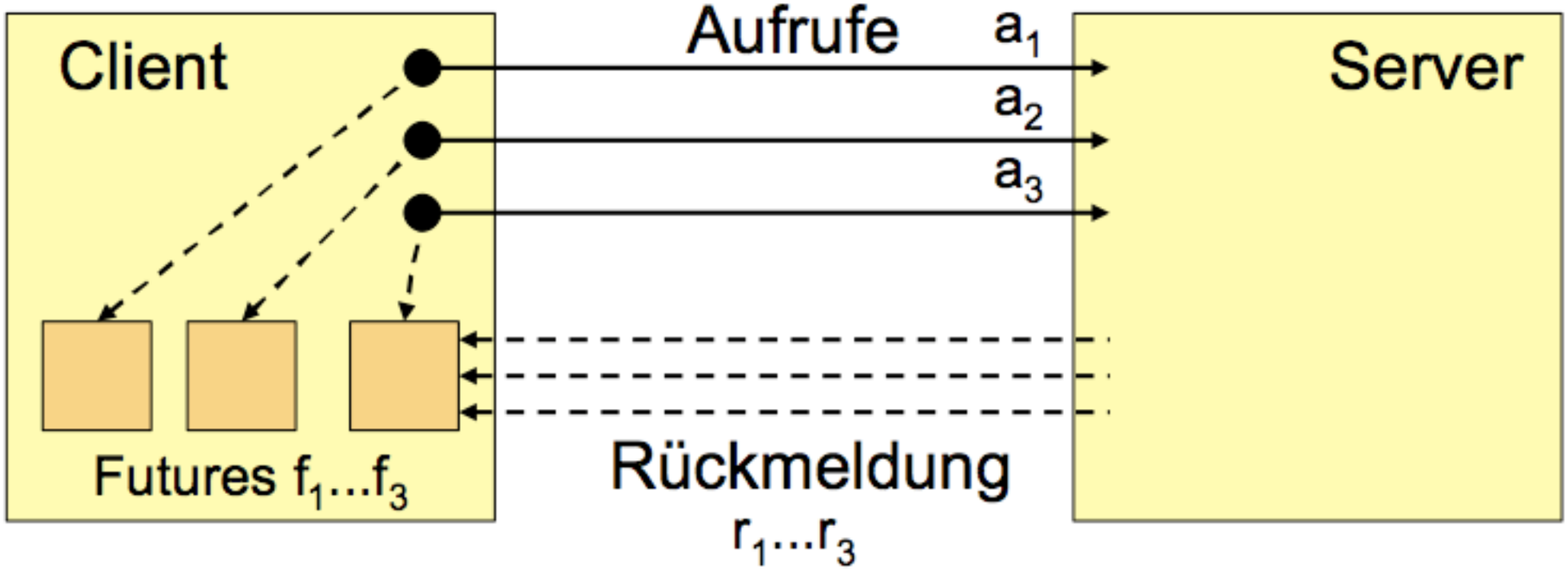
- Transparenzverletzungen
  - Variable Parameter und Typenanzahl (z.B. printf (“%s%d”,x1,x2);)
  - Zeigerparameter (z.B. char *x, ...)
  - Globale Variable
  - Fehlersemantik



RPC - Erweiterungen

- Asynchrone RPCs mit Rückantwort
- Massendatentransfer
- Rückaufrufe
- Fehlertoleranz
- Dynamische Codeinstallation
- Lokale RPC - Optimierung
- Objektorientierte Mechanismen

Asynchrone RPCs



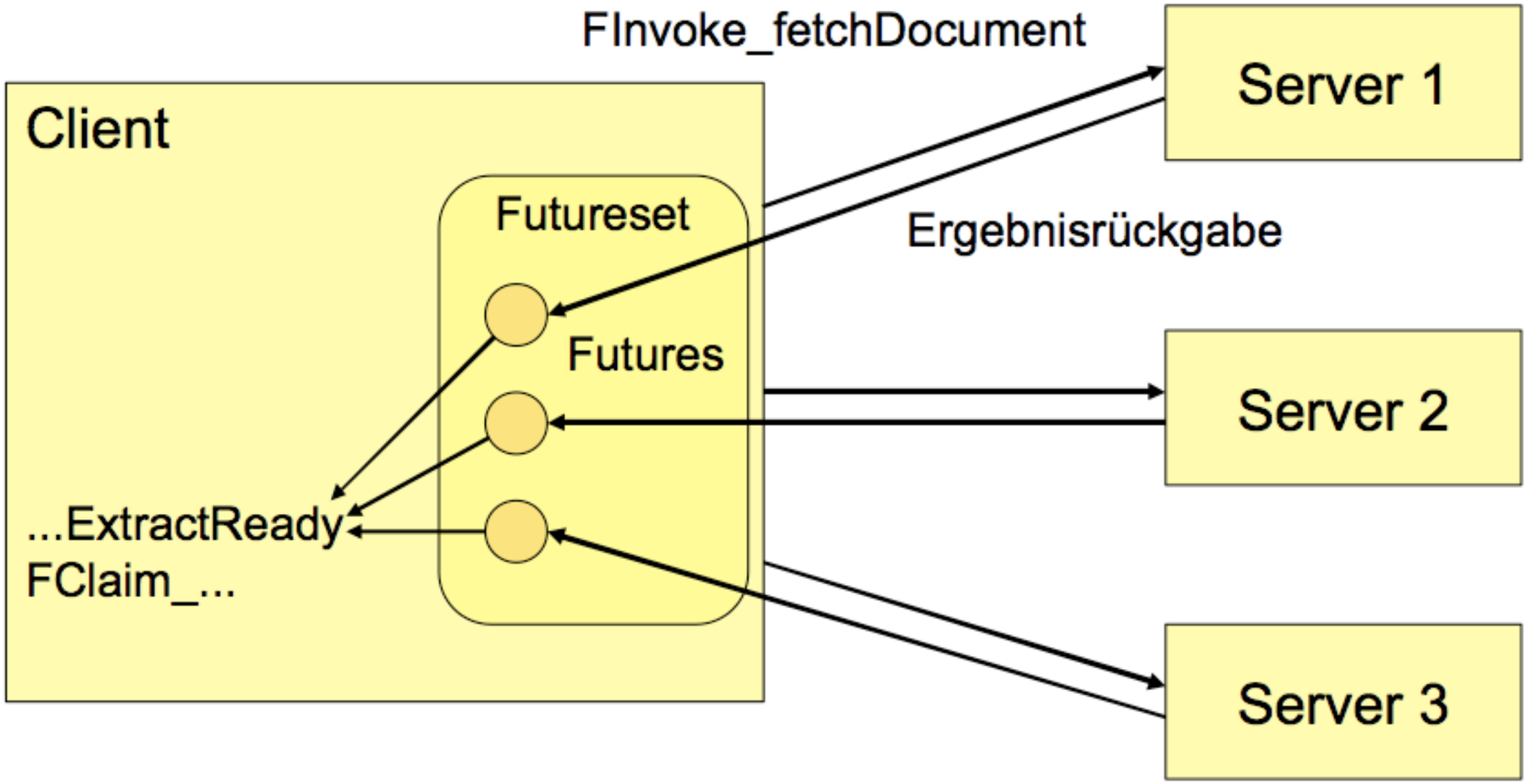
- Asynchrone Aufrufe liefern dem Client ein sogenanntes Future-Objekt
- Rückmeldungen (Ergebnisse) des Servers werden transparent an das jeweilige Future-Objekt geliefert
- Test- und Empfangsoperation auf Futures ermöglichen dem Client den Zugriff auf Ergebnisse asynchroner RPCs
- Spezielle Eigenschaften:
  - Volle Typisierung der Futures
  - Sofortiges Absenden asynchroner Aufrufe -> Antwortzeitoptimierung

Futures: Programmbeispiel

```
Future fd = FInvoke_fetchDocument(&dd);     // Asynchroner Aufruf
....                                        // Ausführung weiterer Op.
if (isReady (fd, timeout)) FClaim_fetchDocument (fd, &d, &status);
                                            // Beschaffen der Ergebnisse
    else ...                                // evtl. Ausnahmebehandlung
Discard (fd);                               // Löschen des Futures
```

```
Futureset fdFutureSet;                      // Definition Futureset
for (i=1; i<maxpar; i++) {
    Future fd = FInvoke_fetchDocument (&dd);    // Asynchroner Aufruf
    FuturesetAdd (&fdFutureSet, fd); }          // Einfügen des Futures in Futureset
for (i=1; i<maxpar; i++) {
    Future fd;                                  // Definition eines Futures
    FutureSetExtractReady(fdFutureSet, &fd);    // Auslesen eines Futures
    FClaim_fetchDocument(fd, &d, &status); }    // Beschaffen d. Ergebnisse etc.
```

Futures: Beispiel



Zeitverhalten

Kernaussage:

k asynchrone Aufrufe (mit Ergebnis) beim gleichen Server sind höchstens

$$\min (1 + t_s/t_a; k)$$

mal schneller als k synchrone Aufrufe.

t_s : Übertragungszeit

t_a : lokale Ausführungsdauer

$1 + t_s/t_a$ groß bei langer Übertragungszeit

t_s klein ⇨ $1 + t_s/t_a \rightarrow 1$ ⇨ keine Verbesserung

t_s groß ⇨ deutliche Verbesserung (z.B.: bei langsamen WANs)

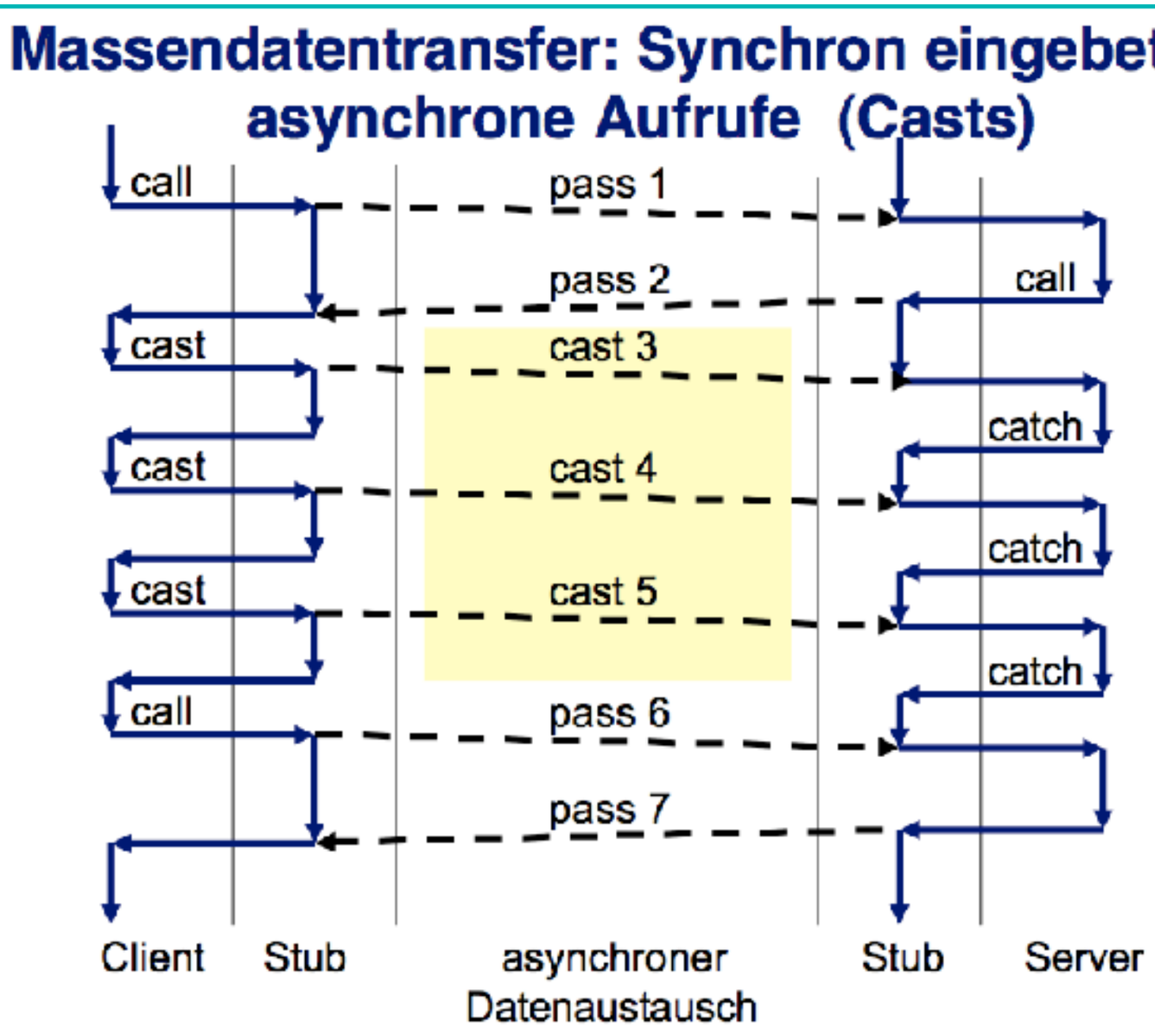
maximale Verbesserung um 1x Nachrichtenlaufzeit pro Aufruf, falls t_a klein, also um Faktor k bei k Aufrufen

(Nachrichten sequentiell gesendet, Aufrufe sequentiell ausgeführt)

RPC und Massendatentransfer

	RPC	Protokolle zum Massendatentransfer
Optimierung der Antwortzeit	sehr wichtig	geringe Bedeutung
Optimierung des Durchsatzes	geringe Bedeutung	sehr wichtig
Begrenzung der Server-Belastung	sehr wichtig	geringe Bedeutung
Speicherung von Zustandsinformation	soll möglichst vermieden werten	von großer Bedeutung zu Optimierungszwecken
Protokoll zum Verbindungsaufbau	nicht vorhanden oder implizit	wichtig zur Vereinbarung von Verbindungsparam.
Initiierung von Sendevorgängen	sofort nach Absetzen eines RPC-Aufrufs	ggf. erst nach Vorliegen großer Datenmengen

Massendatentransfer

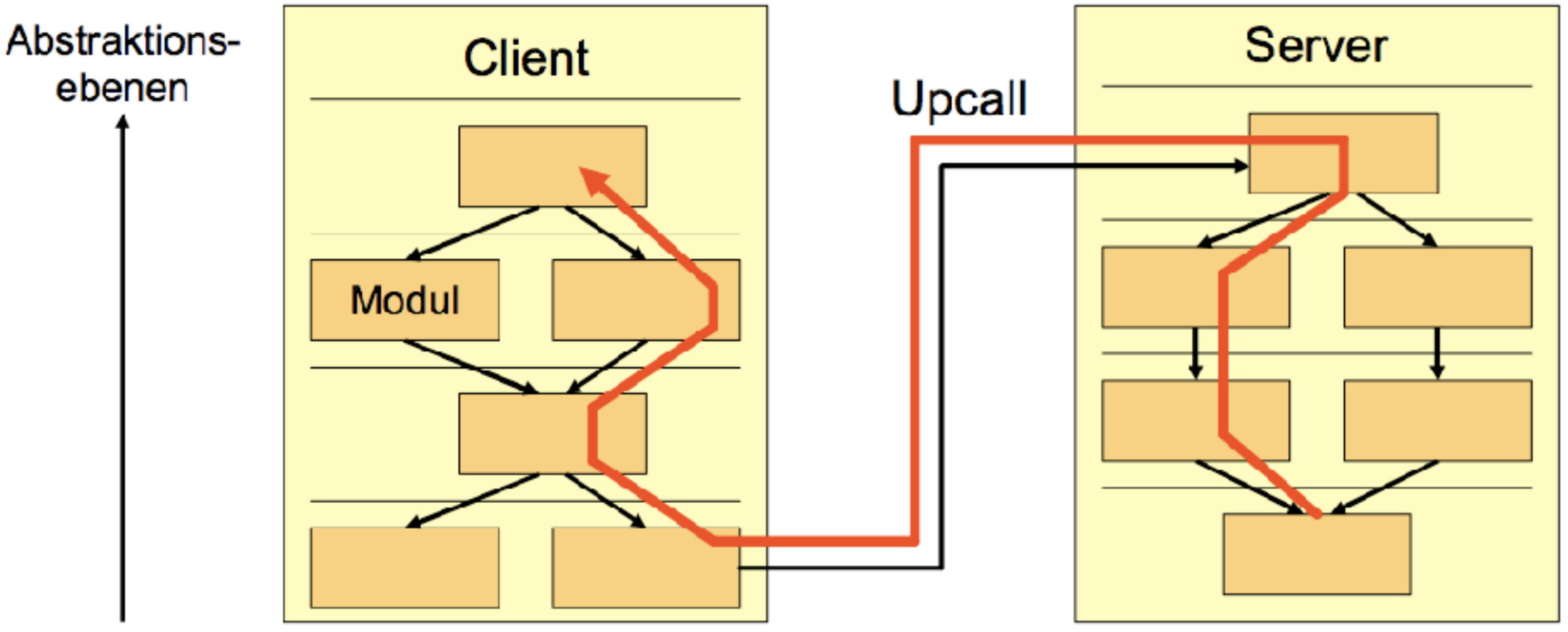


- casts sind unquittiert
- Selektive Wiederholung von Casts nach Synchronisation möglich
- Explizite Richtungssteuerung durch Kontrolltoken

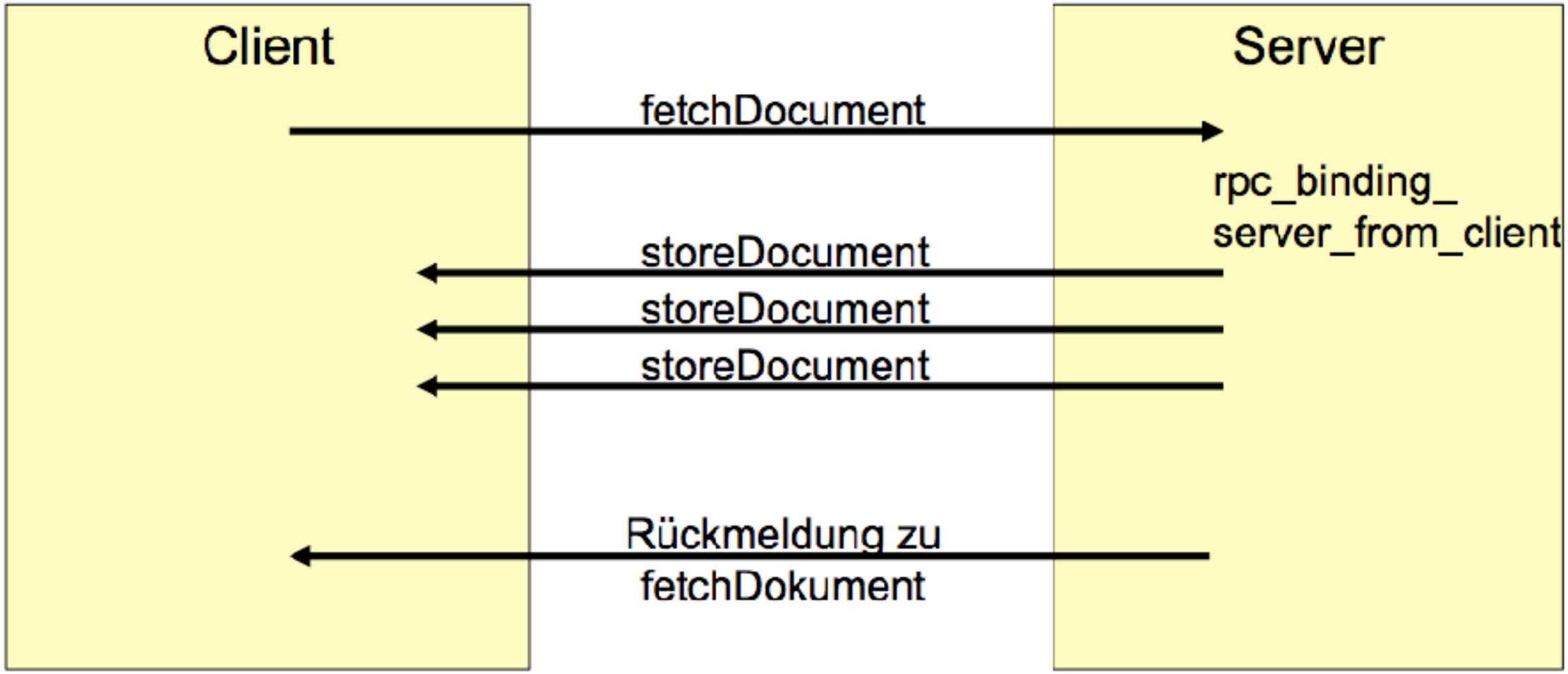
Rückwärtige Aufrufe

- Prinzip
  - Client bietet Aufrufschnittstelle an
  - Server kann während eines laufenden Aufrufs Rückaufrufe absetzen
  - Fortsetzung über mehrere Hierarchieebenen möglich
- Anwendung
  - Vorzeitige Ergebnisrückgabe
  - Anfordern weiterer Daten
  - Statusmeldungen

Prinzip: Distributed Upcalls



Beispiel: Beschaffen von Dokumenten

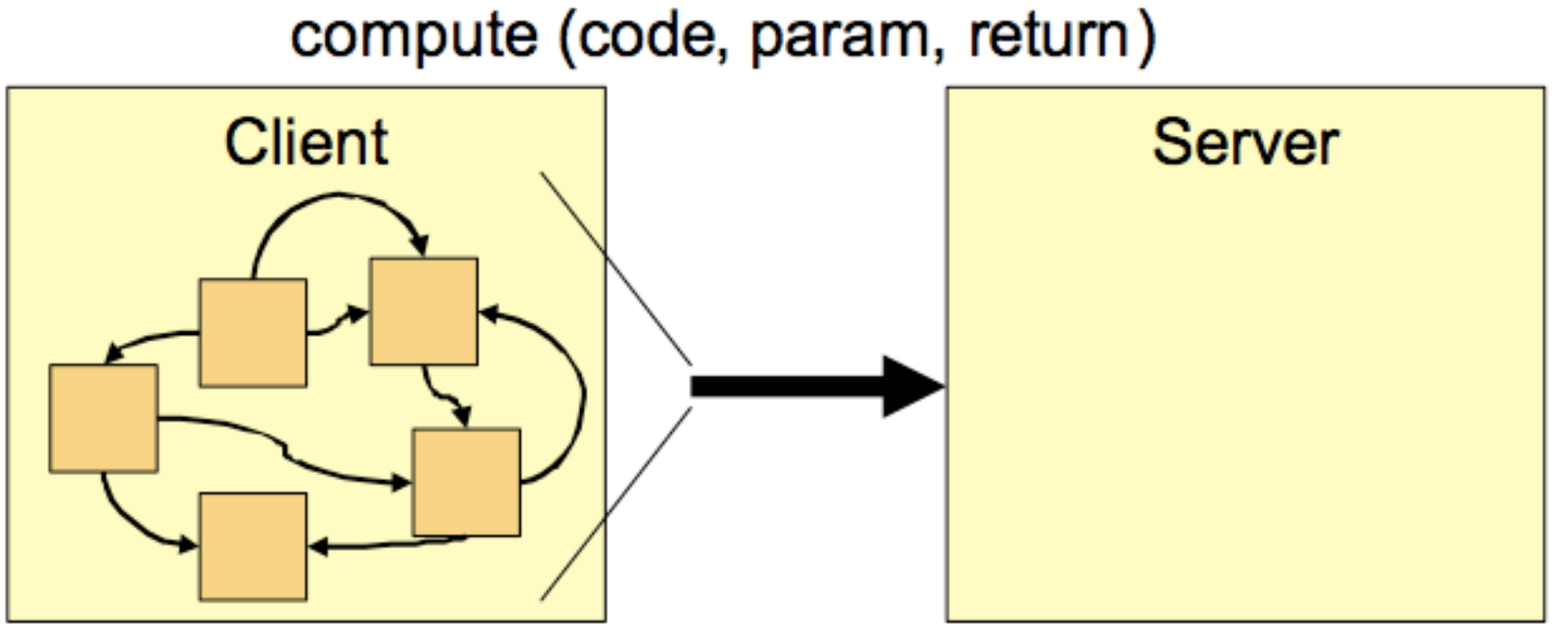


Implementierung beim DCE RPC

```
extern Document *findDocumentOnDisk();    // Beschaffen von Dokumenten

void fetchDocument (rpc_binding_handle_t bh, DocumentDescription *dd)
    // Operation zum Beschaffen von Dokumenten
    // explizites Binden, bh : Serveradresse
{
    Document *d;                                  // Ermitteltes Dokument
    rpc_binding_handle_t callback_handle;    // Clientadresse
    rpc_binding_server_from_client(bh, &callback_handle, &status);
    do {
        d = findDocumentOnDisk(dd);          // Suche des Dokuments
        if (d != NULL) storeDocument(callback_handle, d);   // Rückaufruf an Client
    } while (d != NULL);
}
```

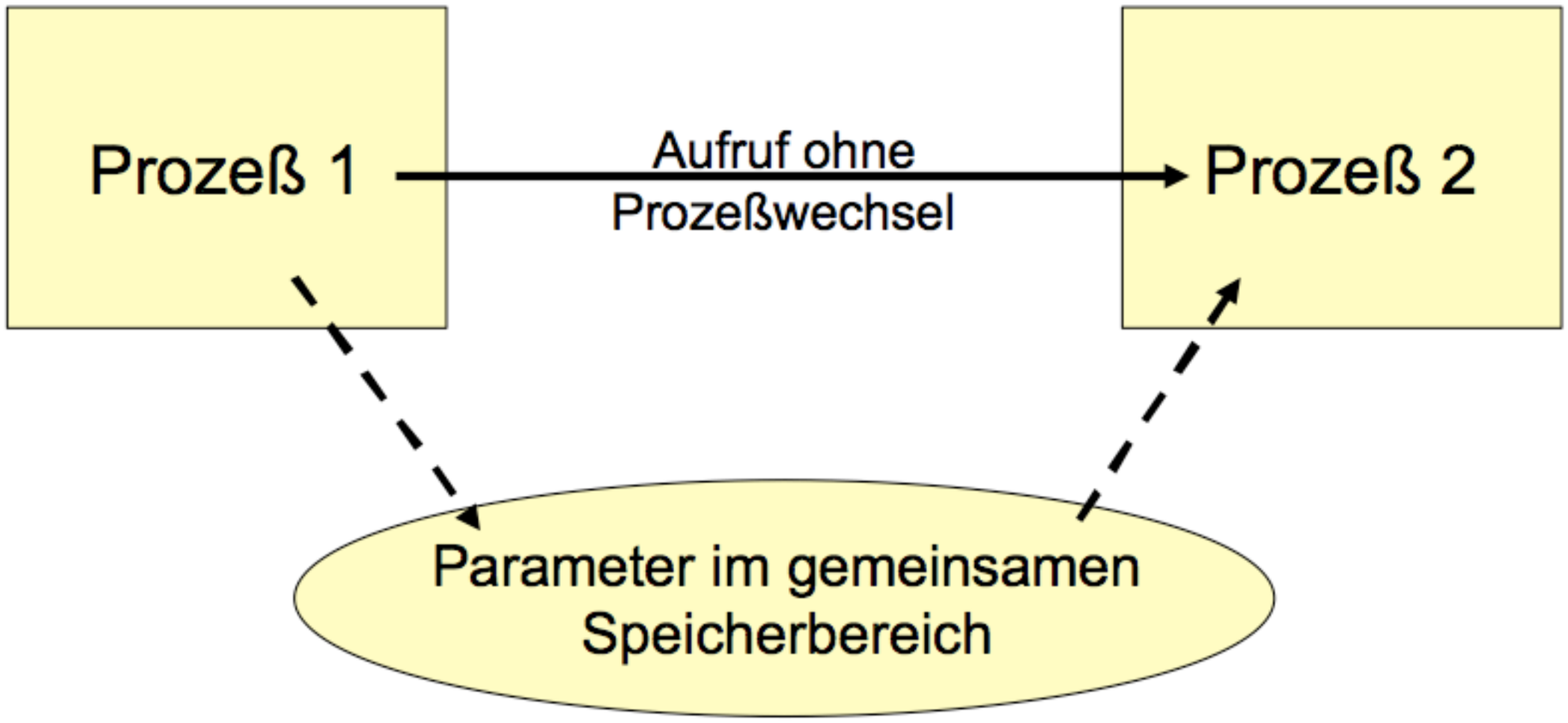
Dynamische Server-Codeinstallation

- REV = Remote Evaluation
- Server erhält Prozedurcode dynamisch (interpretiert oder übersetzt / bei homogenen Systemen)
- Dynamisch referenzierte Prozeduren ebenfalls zu übertragen ->Aufrufgraph
- Anwendung: CPU-intensive Berechnungsdienste
- Ähnlicher Mechanismus: Java Servlets(bei Java Applets Code -Installation auf Client-Seite)



Lokale RPC-Optimierung

- Effiziente RPCs innerhalb eines Rechners
- Beispiel: Lightweight RPC



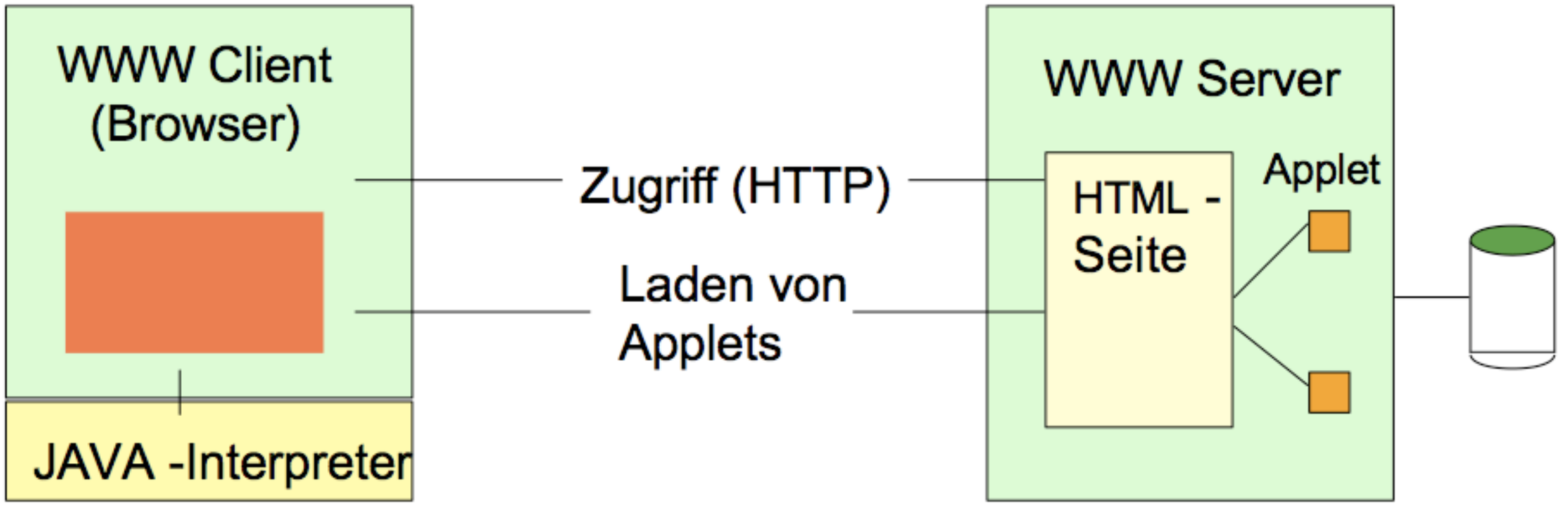
Objektorientierte Mechanismen: Java

- Objektorientierte Programmiersprache, an C++ angelehnt, jedoch einfacher und klarer strukturiert
- Laufzeitsystem mit Bytecode -Interpreter für Java (JVM -Java Virtual Machine) --> PlattformunabhängigkeitServer kann während eines laufenden Aufrufs Rückaufrufe absetzen
- Entwicklungsumgebung JDK (Java DevelopmentKit)Anwendung
- Dynamisch ladbare Applets, mit WWW integriert; Java Web Start als Alternative
- Entfernte Kommunikation zwischen Java-Objekten via RMI (RemoteMethodInvocation)
- Datenbankschnittstelle JDBC (Java Database Connectivity)

Java Enterprise Edition

- EJB (Enterprise JavaBeans)
- JDBC (Java Database Connectivity)
- JMS (Java MessagingService)
- Transaktionen (JTA, JTS -Java TransactionArchitecture / Service)
- JSP (Java Server Pages), Servlet API
- XML (Deployment Descriptor)
- JNDI (Java Naming and Directory Service)
- J2EE Connector(Schnittstellen für Legacy-Integration)
- JWS (Java Web Start)
- JDO (Java DataObjects)

Java Applets: Grundprinzip

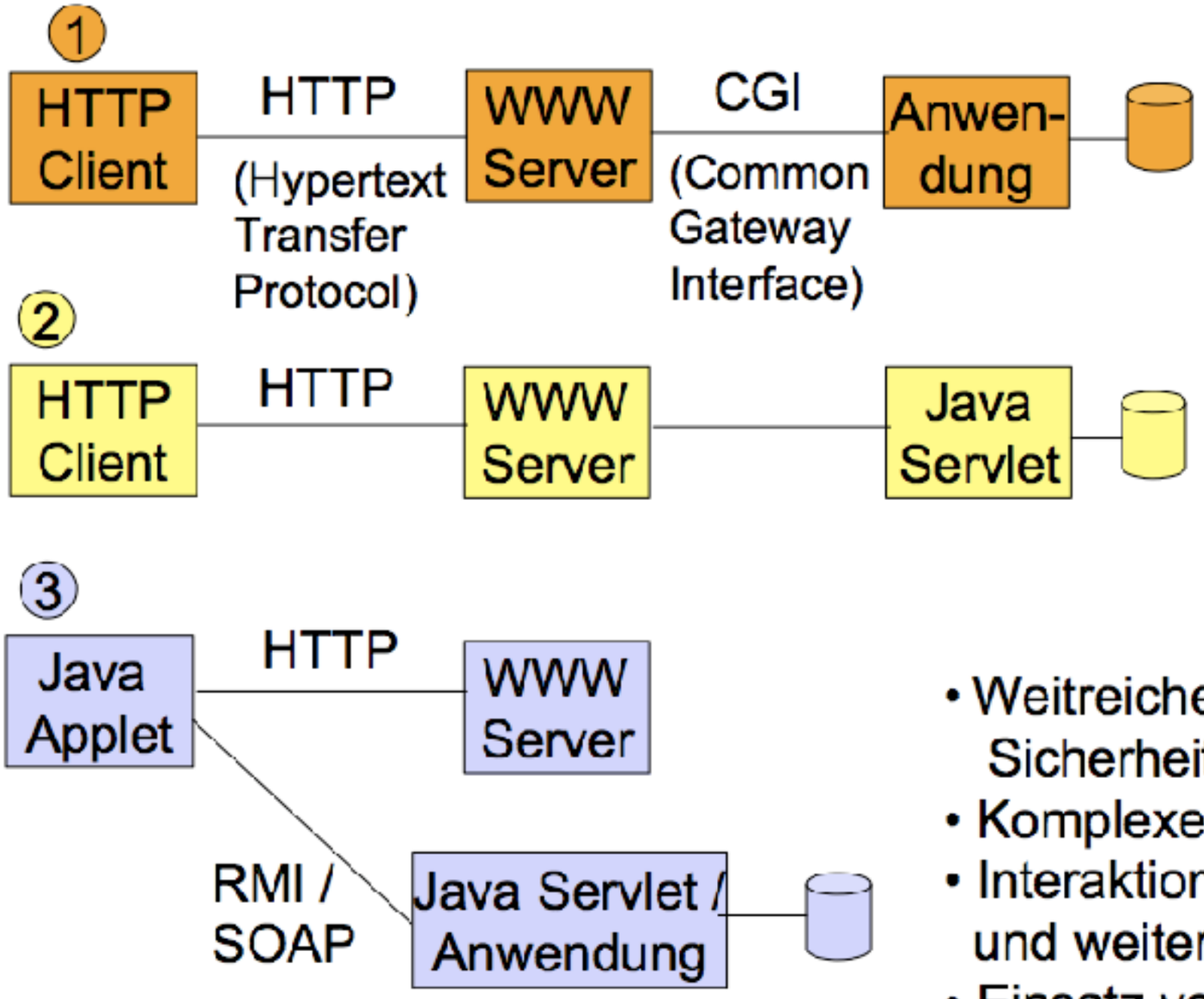


Alternative: Java Web Start: Ebenfalls dynamisches Laden von Java-Code zum Client, dort aber permanentes Caching mit Verknüpfung mit Arbeitsplatz-Oberfläche und automatischem Update -> verbesserte Performance

Vergleich: Dynamische Client-Installation

	Java Applets / Java Web Start	ActiveX-Controls	Script-Sprachen
Status	hersteller-unabhängig	proprietär (Microsoft)	proprietär: • Visual Basic Script (Microsoft) hersteller-unabhängig: • Javascript
Lade-vorgang	Als Bytecode beim ersten Aufruf; plattform-unabhängig	Als Binärcode; plattform-abhängig	Als Teil von HTML-Seiten (Quellcode)
Ausführung	Innerhalb JVM mit Schutz-mechanismen	Im Windows-Betriebssystem ohne Schutz-mechanismen	Durch Interpreter ohne Schutz-mechanismen

Internet/Intranet-Anwendungen: Server

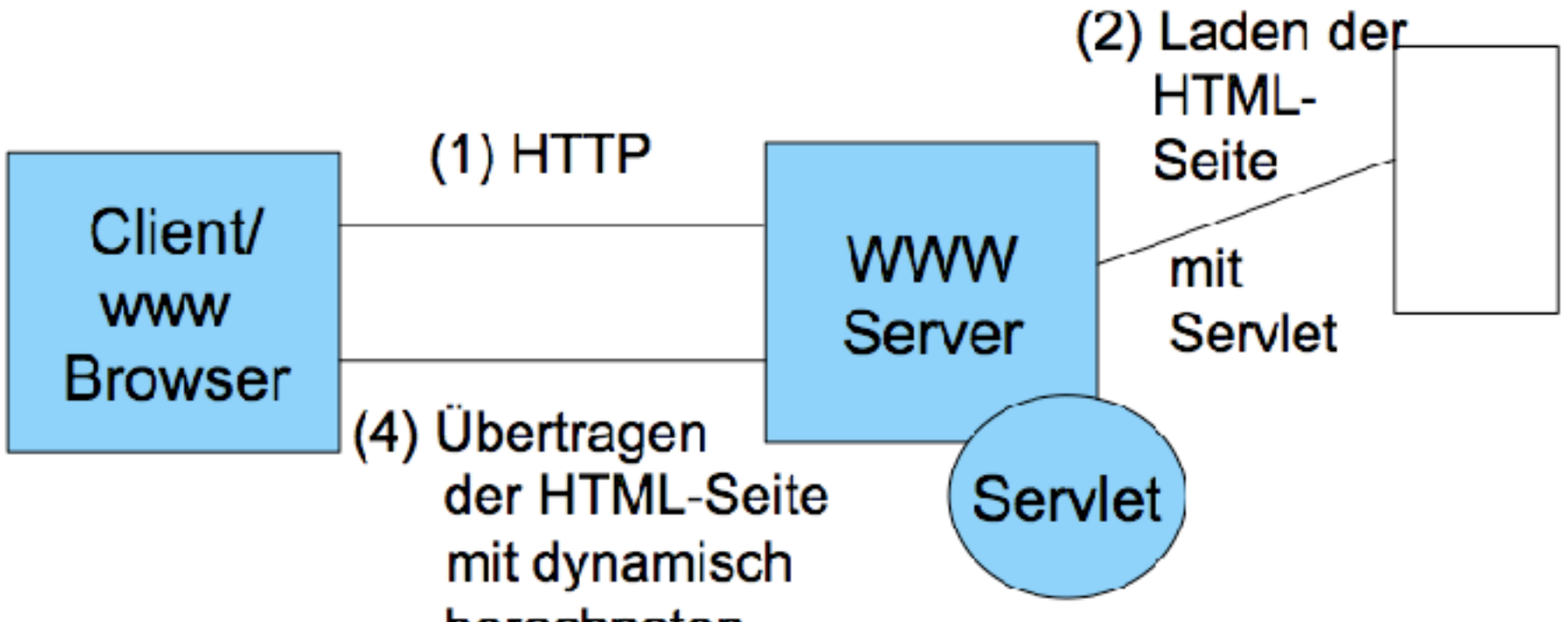


- Relativ umständlicher Parametertransfer
- Neuer Prozeß pro Aufruf => ineffizient

- Flexibler und effizienter
- Jedoch auf HTTP-Interaktion beschränkt

- Weitreichende Flexibilität; jedoch Sicherheit, Firewalls etc. zu beachten
- Komplexere Interaktionen möglich
- Interaktion mit Transaktionen und weiteren Diensten
- Einsatz von Komponenten-technologien (Enterprise JavaBeans)

Java Servlets: Ablauf

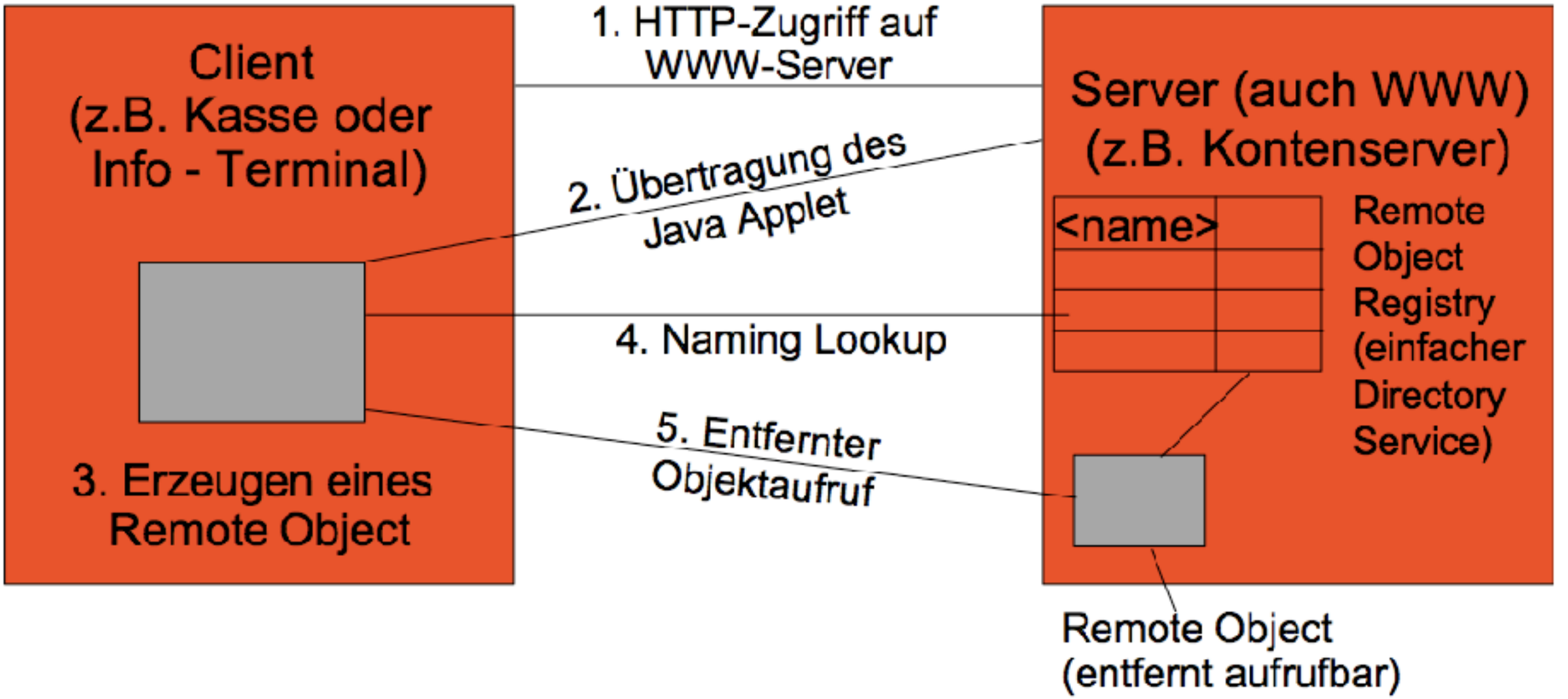


(3) Laden des Java-Servlet und Ausführung:
- Aufruf von Anwendungs-methoden
- Aufruf externer Programme
- Datenbankzugriff

dabei Multi-Threading möglich

=> nur für einfachere Anwendungen geeignet

Java RMI : Grundprinzip

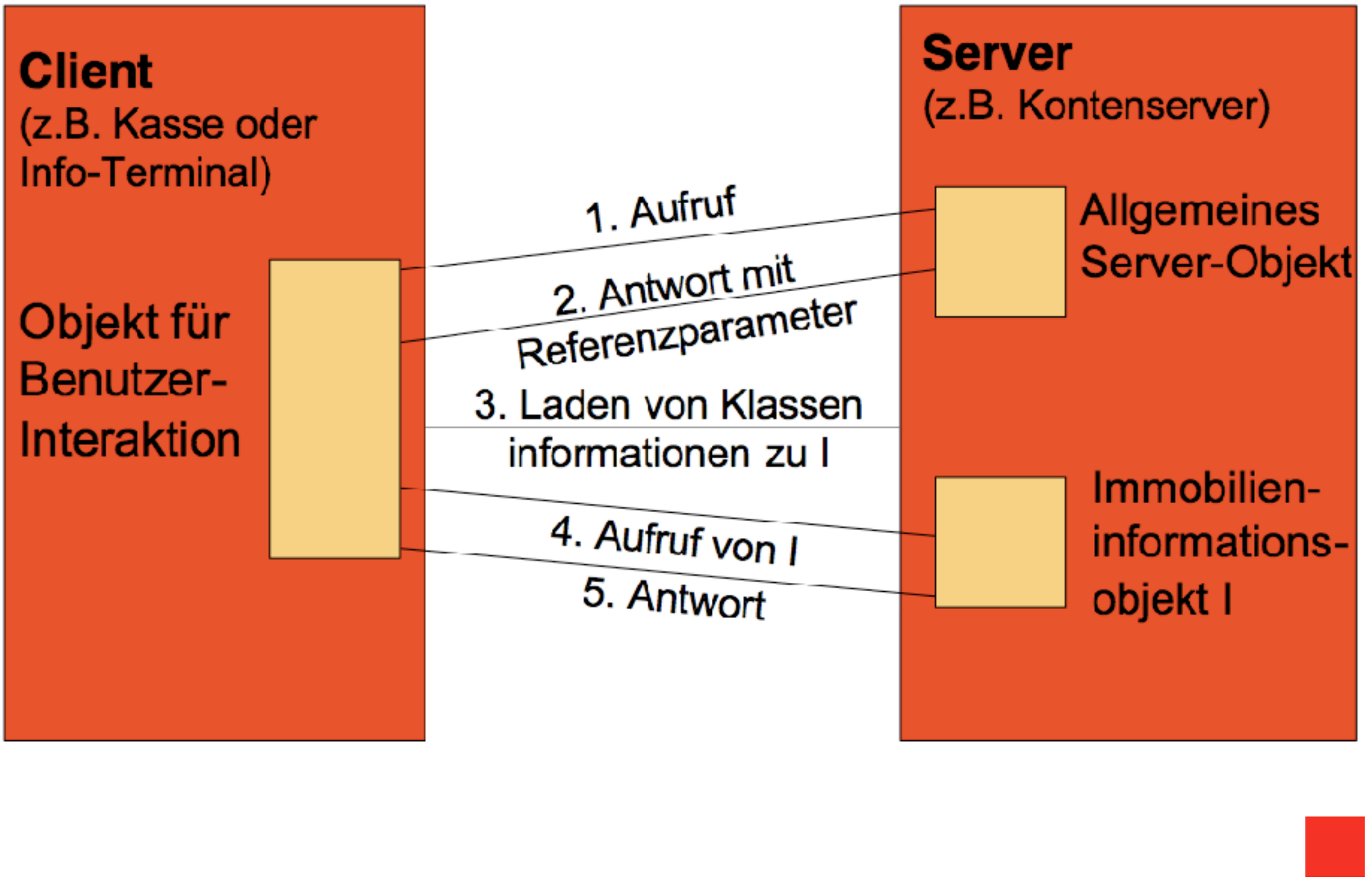


- Interaktion mit WWW-Server, dynamische Anfragen etc. möglich (z.B. für Investment -Informationen)
- Auch Rückaufrufe des Servers bei Client-Objekten (z.B. für Parametereingaben)

Java RMI : Eigenschaften

- RPC-artiger Kommunikationsmechanismus, in Objekte eingebettet
- Übergabe von Remote Objects stets als Referenzparameter, also dynamische Übergabe entfernter Objektreferenzen
- Übergabe anderer Objekte (“Local Objects“) als Wertparameter; keine Migration
- Dynamisches Laden von Klasseninformationen zu einem RemoteObject, auf das ein Client eine Referenz erhält
- Aufrufe generell synchron; asynchrone Aufrufe nur mittels Threads möglich

Beispiel : Remote Obejcts als Parameter



Beispiel: Schnittstellenbeschreibung

```
import java.rmi.*;
public interface Bank extends java.rmi.Remote {
      final long maxTransferAmount = 2000;                    // Maximaler Geldbetrag für Überweisungen
      float balanceQuery(AccountIdentification accountIdent) throws java.rmi.RemoteException;
      void transferRequest(AccountIdentification accountident, float amount, TransferOrder transOrder)
                                                                        throws java.rmi.RemoteException;
}

_____________________

import java.io.Serializable;
public class AccountIdentification implements Serializable // Identifikation für ein bestimmtes Konto
{     byte accountNumber[];                                 // Kontonummer
      long pin;                                             // PIN des Kontoinhabers
      String name;                                          // Name des Kontoinhabers
      public AccountIdentification(byte[] account, long pin, String name) {
            this.accountNumber = account;
            this.pin = pin;
            this.name = name;
      }
}
public class TransferOrder implements Serializable{        //Überweisungsformular
      String bankname;                                      // Bankname
      byte bankSortingCodeNumber[];                         // Bankleitzahl
      byte accountNumber[];                                 // Kontonummer
      String asPaymentFor;                                  // Verwendungszweck
      public TransferOrder(...) {...}
};
```

Beispiel: Server

```
import java.rmi.*;
import java.rmi.registry.*;
import java.rmi.server.UnicastRemoteObject;

public class BankImpl extends UnicastRemoteObject implements Bank
{
    public BankImpl() throws RemoteException {}
    ...
    public  float balanceQuery(AccountIdentification accountIdent)     // Abfrage des Kontostands
    {       float balance;
            Account account;                                           // Kontovariable
            retrieveAccount (accountIdent.accountNumber, account);  // Konto aus der Datenbank holen
            checkAccount(account, accountIdent.pin, accountIdent.name); //Zugriffsberechtigung überprüfen
            getBalance(account, balance);                              // Kontostand in 'balance' eintragen
            return balance;
    }
    public void transferRequest(AccountIdentification accountIdent, float amount, TransferOrder transOrder)
    {...}
    public static void main(String args[])
    {       System.setSecurityManager(new RMISecurityManager());
            try {
                    BankImpl server = new BankImpl();
                    Registry registry = LocateRegistry.getRegistry();
                    registry.rebind("Bank", server);
            } catch (Exception e) {
                    System.out.println(e.getMessage());
                    e.printStackTrace();
            }
    }
}
```

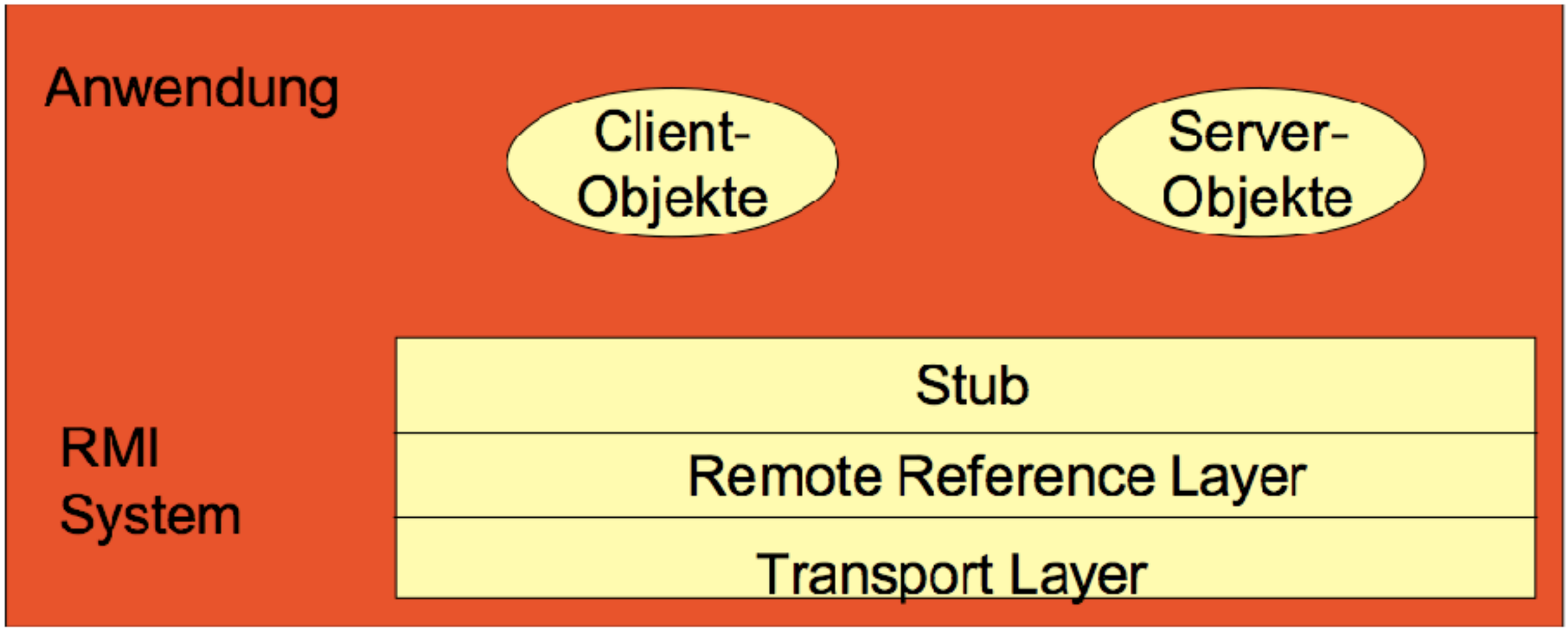
Beispiel: Client-Applet

```
import java.rmi.*;
import java.rmi.registry.*;
import java.applet.Applet;
import java.awt.Graphics;

public class BankAccessClient extends Applet {
    AccountIdentification accountIdent;
    float balance;                                  // Kontostand
    Bank remoteBankServer;                          // Bank-Interface
    String remoteBankServersURL;                    // URL des Bank-Interface

    public void init()
    {
        try {       remoteBankServersURL = "rmi://" + getCodeBase() + "/" + "Bank";
                    remoteBankServer = (Bank) Naming.lookup(remoteBankServersURL);
        } catch (Exception e) {
                    System.out.println(e.getMessage()); e.printStackTrace(); }
    }
    public void start() {
        inputAccountIdentification(accountIdent);       // Eingabe der Kontoidentifikation
        try {       balance = remoteBankServer.balanceQuery(accountIdent);
                                                        // Entfernte Abfrage des Kontostands
        } catch (Exception e) {
                    System.out.println(e.getMessage()); e.printStackTrace(); }
    }
    public void paint(Graphics g) {
        g.drawString("Kontostand=" + new Float(balance).toString(),10,10);}
    public void inputAccountIdentification(AccountIdentification accountIdent) {...}
}
```

Beispiel: Integration in WWW

```
<HTML>
<HEAD>
<TITLE>Bank Access</TITLE>
</HEAD>
<BODY>
<APPLET CODE="BankAccessClient.class" width=500
   height=500></APPLET>
</BODY>
</HTML>
```

Interne Systemarchitektur



Remote Reference Layer

- Verwaltung entfernter Objektreferenzen
- Aufruf replizierter Objekte
- Aktivierung von Objekten bei Bedarf

Interne Systemarchitektur

Transport Layer

- Verbindungsverwaltung (i.d.R. eine Verbindung zwischen einem Paar von Betriebssystemprozessen)
- Objektreferenz = <endpoint (IP-Adresse,Port); object ID>

Multithreaded Servers

- Default-Mechanismus für die Ausführung von Aufrufen unterschiedlicher Objekte
- Aufrufe desselben Clients i.d.R. sequentiell ausgeführt

Sicherheitsaspekte bei Java RMI

- Laden von Klassen nur vom Rechner des jeweiligen Applets (Ausnahme: Signierte Applets)
- Überprüfung von Applets durch Applet SecurityManager, Verbot lokaler Dateizugriffe sowie des Aufbaus fremder Netzverbindungen („Sandbox“); jedoch kontrollierte Zugriffserlaubnis möglich
- Zusätzliche digitale Signaturen für Applets
- Authentisierung und Verschlüsselung auf der Basis der Java CryptographicArchitecture
- Implementierung durch SecurityPackages, z.B. für DES(DataEncryptionStandard) oder RSA

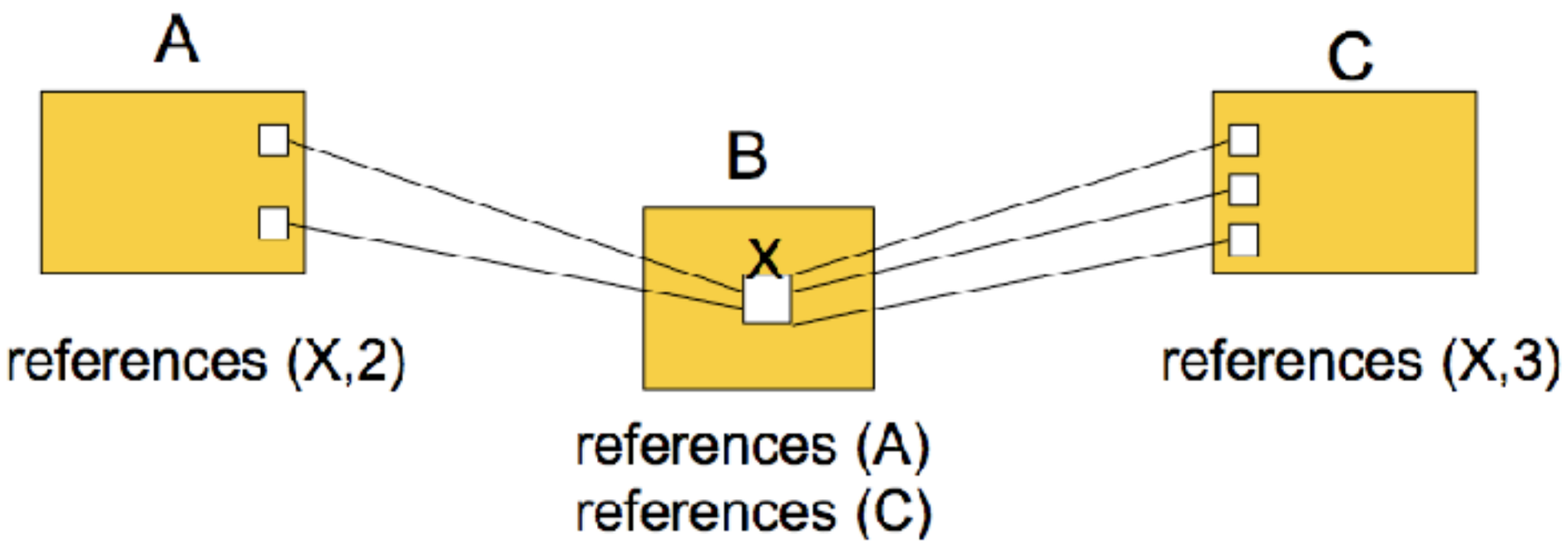
Garbage Collection

Automatische Speicherverwaltung für verteilte Java-Objekte;
Behandlung von Rechnerausfällen durch Testnachrichten

Basis: Reference Counting:

- Bei Erzeugung der ersten Referenz eines Prozesses auf ein entferntes Objekt: “referenced”-Nachricht an Server
- Bei weiteren Referenzen Inkrementierung eines lokalen Zählers
- Automatisches Löschen von Objekten, wenn keine Referenzen mehr existieren

Beispiel:



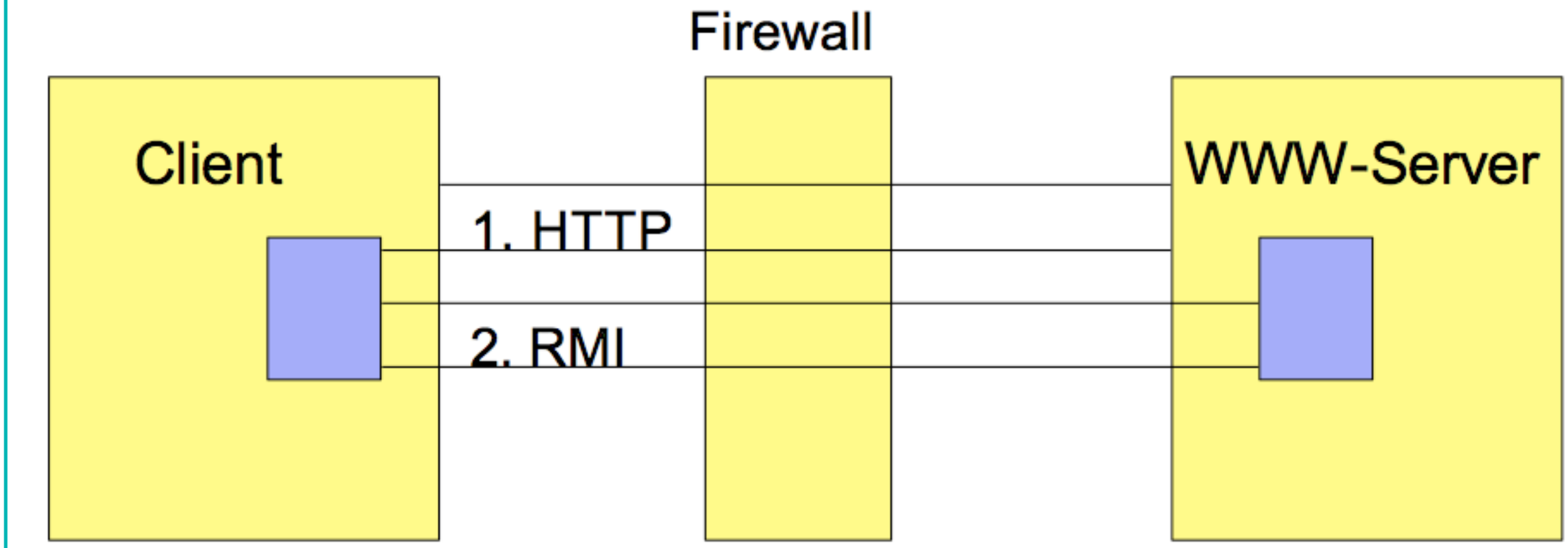
Registry Service

- Lokaler flacher Directory Server pro Server-Rechner
- Einfach nutzbar, aber nicht skalierbar

- Export von Objekten nur an lokalen Registry Service durch den Server
- Import netzweit möglich unter Angabe von Server-URL und Objektname; bei Applets jedoch nur Zugriff auf Server, von dem Applet geladen wurde

=> für einfache Anwendungen nutzbar, bei großen Applikationen jedoch Integration eines vollwertigen Directory Service sinnvoll (z.B. via JNDI-Schnittstellen)

RMI via Firewall



Problem:

- Firewall läßt nur HTTP-Aufrufe zu
- RMI arbeitet normalerweise über direkte TCP/IP-Sockets

=>

- RMI-Aufrufe können in HTTP POST requests eingebettet werden
- Dadurch kontrollierte RMI-Aufrufe via Firewall möglich

Erweiterte RMI-Aufrufmechanismen

- Basismechanismus: Unicast (Punkt-zu-Punkt)
- Persistente Referenzen: Referenzierung von Objekten auf externen Speichermedien, dynamische Aktivierung bei Aufrufen; von Java RMI bereits unterstützt

Replizierte Objektgruppen:

- Replikation von Objekten auf mehreren Rechnern
- Replizierte Aufrufe
- Ggf. auch atomic multicast (Aufruf an alle Replikate oder an keines (im Fehlerfall))