

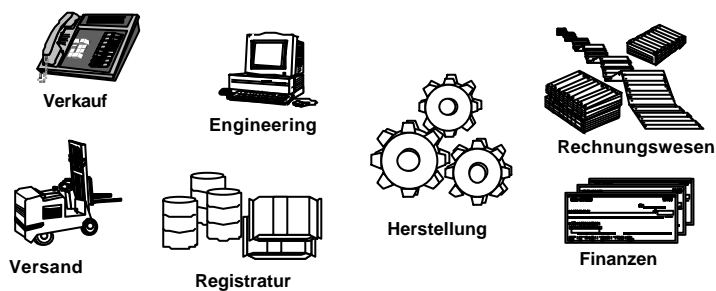
## 13.5 CORBA

---

Anmerkung: Teile aus Quellen der OMG

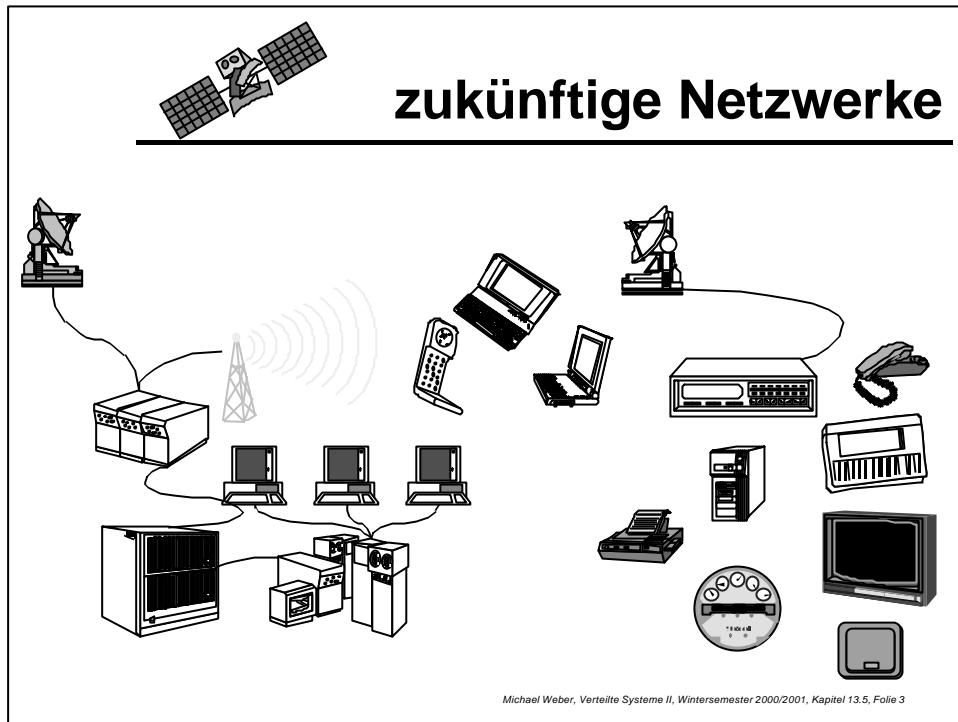
### Das Geschäftsmodell

---



- Jedes Programm ist ein Modell eines Teils des Gesamtunternehmens
- Warum sollten diese nicht zusammenarbeiten?

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 2



- ## Das Problem
- Anwendungsintegration und Verteilte Systeme sind die gleiche Problemstellung
  - Konstruktion von Verteilten Systemen aus unterschiedlichsten Quellen:
    - heterogen,
    - vernetzt,
    - physikalisch verteilt,
    - verschiedene Hersteller.
- Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 4

## **Existierende Werkzeuge?**

---

- Grundlegende Probleme:
  - Existierende Werkzeuge (Sockets, DCE, ONC) sind zu low-level.
  - Sie bieten keine gleichartige Sicht auf alle verteilten Anwendungen.
  - Die Komplexität verteilter Systeme wächst stetig.
  - Es entsteht ein Implementierungs- und Managementaufwand, der zunehmend unbeherrschbar wird.

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 5

## **Fokus auf Interoperabilität**

---

- Ausgangsüberlegung:  
es wird keinen Konsens geben bzgl.
  - Hardwareplattformen;
  - Betriebssystemen;
  - Netzwerkprotokollen;
  - Anwendungsformaten.
- Daraus folgt:  
es muss Konsens bzgl Interoperabilität geben

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 6

## Objekttechnologie

---

- Das Problem mit objektorientierter Technologie anzugehen verspricht dieses zu vereinfachen:
  - vereinheitlichte Sicht auf ein verteiltes, heterogenes System.
  - Schlüsseleigenschaften der Objektorientierung
    - Kapselung,
    - Polymorphismus,
    - Vererbung.

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 7

## Object Management Group

---

- Not-for-profit Firma
  - gebildet von Mitgliedsfirmen
  - Sitz in USA, Filialen in UK, Japan & Deutschland.
- Gegründet im April 1989.
- Wenige Angestellte (15 full time);
- keine Eigenentwicklungen.
- Mission:
  - Schaffung und Propagation von objekt-orientierten Standards zur Anwendungsintegration basierend auf existierenden Technologien.
  - Marktstudien, Weiterbildung, Seminare und Konferenzen.

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 8

## **Technical Committee**

---

- Repräsentanten der Mitgliedsfirmen.
  - ABM (all but Microsoft)
- Bestimmen die Weiterentwicklungsrichtung der Corba-Architektur und der Standards.

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 9

## **Adoptionsprozess**

---

- RFI (Request for Information) um eine Menge kommerziell verfügbarer Software zu einem Problemkreis bekannt zu machen.
- RFP (Request for Proposals) um explizite Spezifikationen und Beschreibungen verfügbarer Software für den Problemkreis zu sammeln.
- Letters of Intent um Firmendirektiven auszudrücken.
- Task Force und End User evaluation & recommendation; simultane Prüfung durch das Business Committee.
- Entscheidung des Board aufgrund der TC, End User, and BC Empfehlungen.

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 10

## Verfügbarkeit

---

- OMG nimmt Interfaces an (Adoptionsprozess) und publiziert diese.
- Interfaceimplementierung muss kommerziell verfügbar sein bei einem OMG Mitglied.
- Interfaces müssen frei verfügbar sein für alle, auch für Nicht-Mitglieder.
- Interfaces, die aus existierenden Produkten gewählt werden, müssen einen Wettbewerbsprozess durchlaufen.

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 11

## Domain Task Forces

---

- **Manufacturing:**
  - Engineering Resource Planning (ERP), Product Data Management (PDM);
- **Finance**
  - Currency Standard, Party Management, Compass Project for Accounting
- **Telecommunications**
  - TMN/CORBA interworking, CORBA for IN, Audio/Video Streams
- **Electronic Commerce**
  - Electronic Payments, Negotiations
- **Healthcare (CORBAmed)**
  - Patient Identification Number (PID), Lexicon Query System, Life Sciences,...
- **Transportation**
  - Air Traffic control,
  - inter-nodal transport.

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 12

## Special Interest Groups

---

- End-user SIG.
- Object-oriented analysis & design SIG
- Object-oriented database interface standards SIG
- Business Object Management SIG
- Manufacturing SIG
- Healthcare SIG
- Telecommunications SIG
- Financial SIG
- Security SIG

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 13

## Offenheit

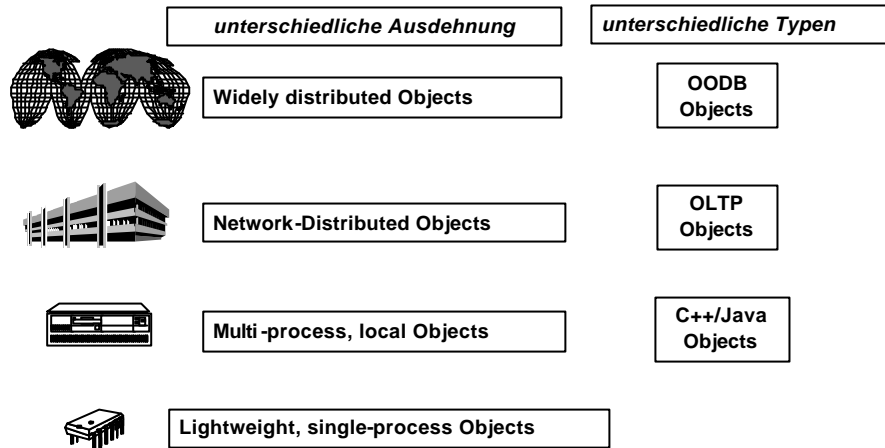
---

- Institutionalisierte Austausch der Ergebnisse mit:
  - X/Open
  - OSF, X Consortium, W3C
  - ESPRIT, NIST, NII
  - ISO, ITU
  - ANSI, IEEE, JIPS
  - CFI, ODMG, COS, NMF, IMA, POSC

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 14

# Skalierbare Architektur

---



Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 15

# vereinheitlichte Grundlage

---

- Interoperabilität und Portabilität durch:
  - vereinheitlichte Terminologie der Objektorientierung.
  - vereinheitlichtes abstraktes Objektmodell.
  - vereinheitlichtes Referenzmodell und Architektur.
  - vereinheitlichte Interfaces und Protokolle.
- Diese Modelle sind im OMA Guide beschrieben.

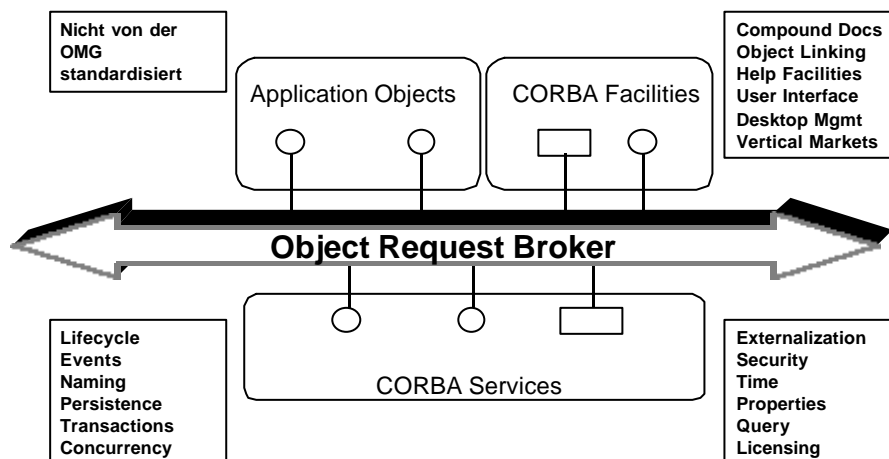
Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 16

# Objektmodell

- Definiertes Ziel:  
*"to define an object model that facilitates Portability of applications and type libraries, and Interoperability of software components in a distributed environment."*
- das Modell war 1992 fertig.
- Schlüsselkonzepte:
  - Core,
  - Components,
  - Profiles.

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 17

# Open Management Architecture



Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 18

## **Fundamentale CORBA Design Prinzipien**

- Trennung von Interface und Implementierung
  - Klienten hängen von Interfaces ab, nicht von (Server-) Implementierungen
- Lokationstransparenz
  - Dienstenutzung ist orthogonal zur Dienstlokation
- Zugriffstransparenz
  - Aufruf von Objektmethoden und -operationen
- getypte Interfaces
  - Objektreferenzen sind typisiert durch ihre Interfaces
- Mehrfachvererbung von Interfaces
  - Vererbung erweitert oder spezialisiert das Verhalten

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 19

## **Fundamentale CORBA Design Prinzipien**

- CORBA unterstützt zuverlässige Unicast-Kommunikation
  - oneway, twoway, deferred synchronous
- CORBA Objekte können in Client/Server, Peer-to-Peer oder Publish/Subscribe Anwendungsarchitekturen eingesetzt werden
  - z.B. der COS Event Service definiert ein Publish/Subscribe-Kommunikationsmodell

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 20

## Vorteile von CORBA

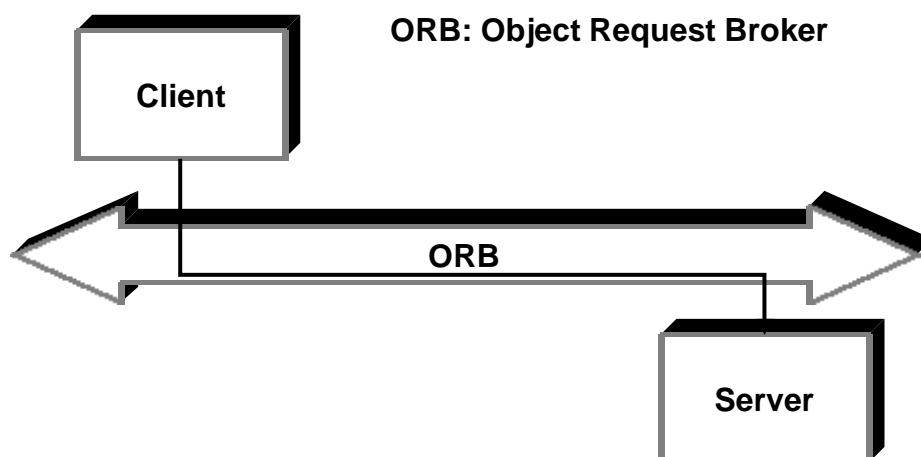
---

- Vereinfachtes Kommunikationsmodell
  - höhere Abstraktionsebene als traditionelle ungetypte Byteströme
- Vorteile des OO-Ansatzes analog zu den Vorteilen von OO-Sprachen bei nicht-verteilter Programmierung
  - Kapselung, Interfacevererbung, Polymorphismus, Ausnahmebehandlung

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 21

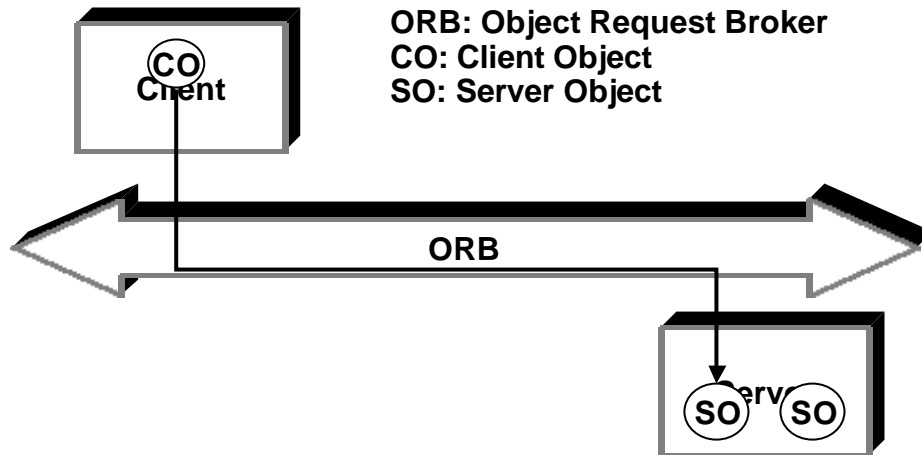
## CORBA Architektur

---



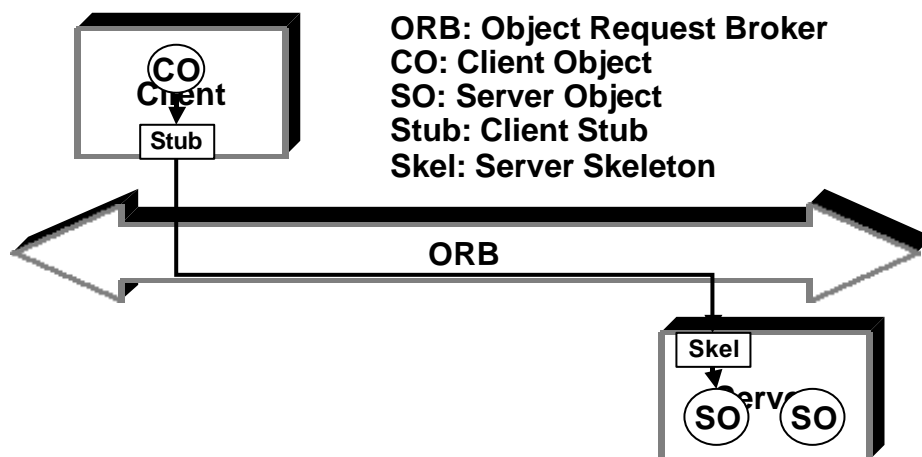
Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 22

# CORBA Architektur



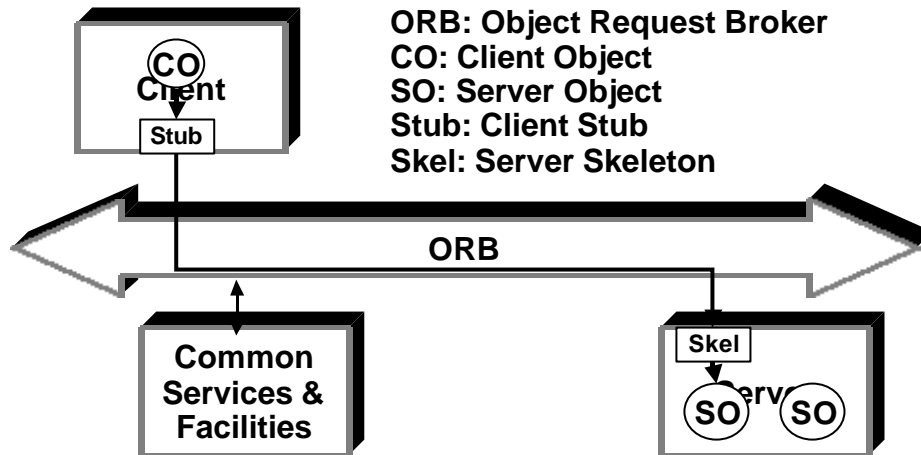
Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 23

# CORBA Architektur



Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 24

## CORBA Architektur



Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 25

## Interface Definition Language

- IDL separiert Interface und Implementierung
- Vorteile einer IDL
  - Plattformunabhängigkeit (z.B. NT, Unix)
  - verstärkte Modularisierung
  - erhöhte Robustheit
  - Programmiersprachen-Unabhängigkeit
- es gibt viele IDLs
  - ASN.1, DCE IDL, ONC XDR, CORBA IDL

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 26

## CORBA IDL

---

- Objektorientiert, streng typisiert, öffentliche Spezifikationssprache
- unabhängig von einer spezifischen Sprache bzw. Compiler
- Sprachbindungen für viele Sprachen/Compiler (language mappings)
  - C, C++, Smalltalk, COBOL, Modula3, DCE, Java, ...
- keine Programmiersprache
  - ähnlich zu Java Interfaces / C++ abstract classes

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 27

## CORBA IDL Elemente

---

- modules und interfaces
- Operationen und Attribute
- einfache und mehrfache Vererbung
- Basistypen (double, long, char, etc.)
- any type
- Arrays und sequence
- struct, enum, union, typedef
- consts
- exceptions

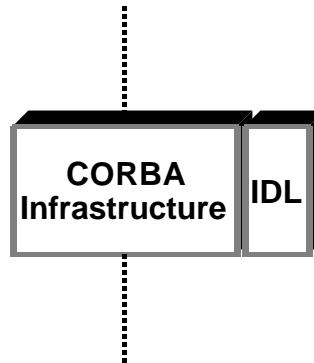
Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 28

# CORBA IDL

*IDL isoliert das Interface von der Implementierung*

Client side

Server side



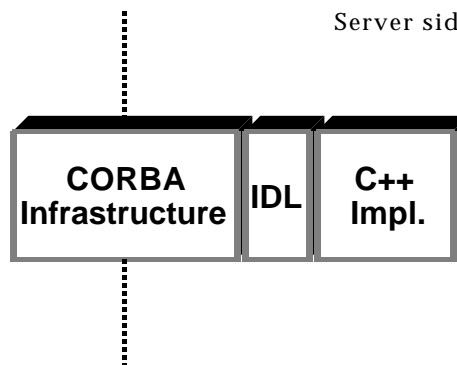
Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 29

# CORBA IDL

*die Interface-Implementierung erfolgt bspw. in C++ unter Verwendung des C++ Language Mapping.*

Client side

Server side



Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 30

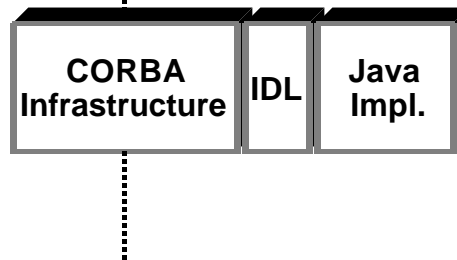
# CORBA IDL

---

*oder in Java.*

Client side

Server side



Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 31

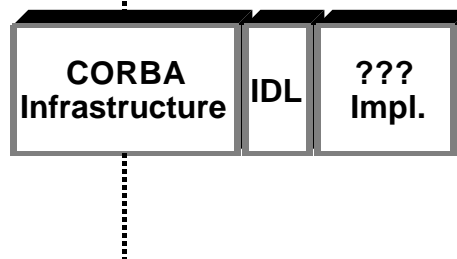
# CORBA IDL

---

*oder in C, Smalltalk, Cobol oder sonstigen "Exoten".*

Client side

Server side



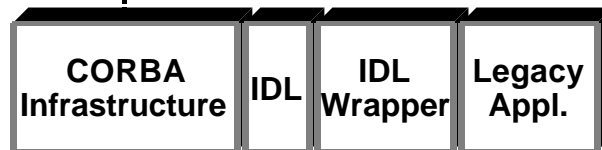
Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 32

## CORBA IDL

*Legacy-Applikationen (alte Anwendungen) können durch IDL Interfaces eingepackt werden (wrapping)*

Client side

Server side



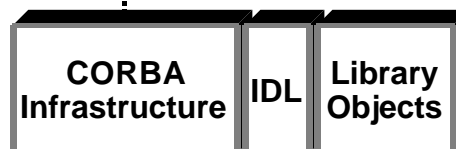
Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 33

## CORBA IDL

*oder Bibliotheken von Drittanbietern.*

Client side

Server side



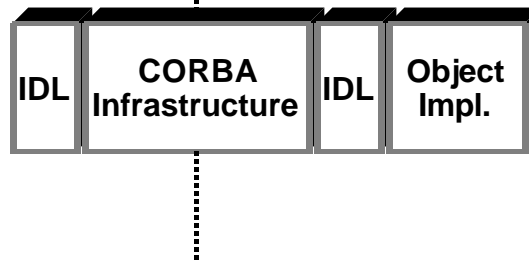
Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 34

## CORBA IDL

*die gleiche IDL-Beschreibung definiert  
das Klienten-Interface.*

Client side

Server side



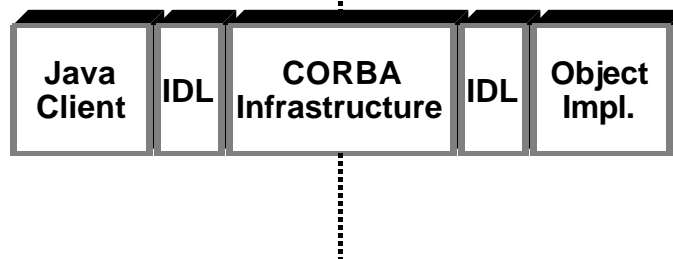
Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 35

## CORBA IDL

*der Klient kann dann z.B. in Java implementiert werden.*

Client side

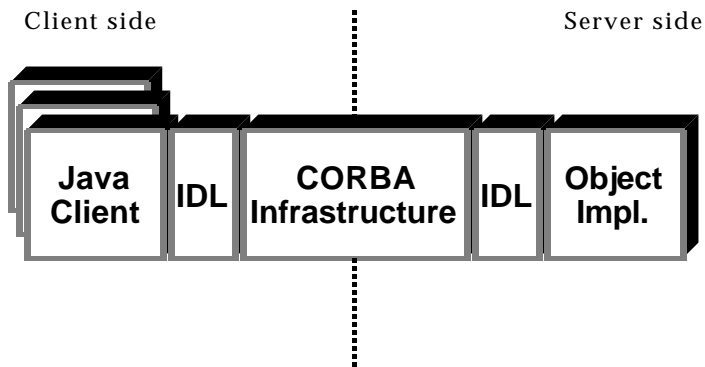
Server side



Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 36

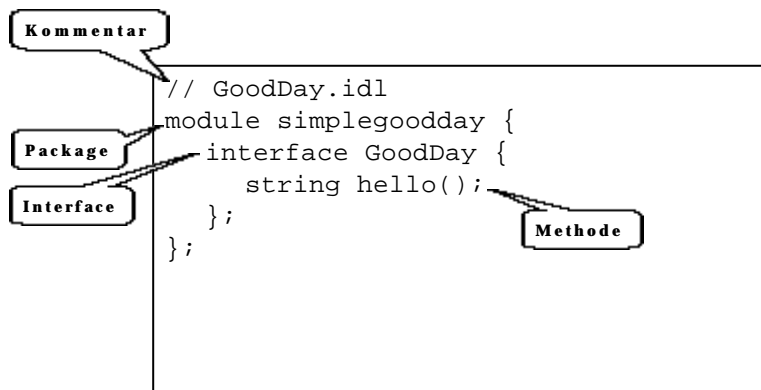
# CORBA IDL

*oder in C++, Ada, Smalltalk, ...*



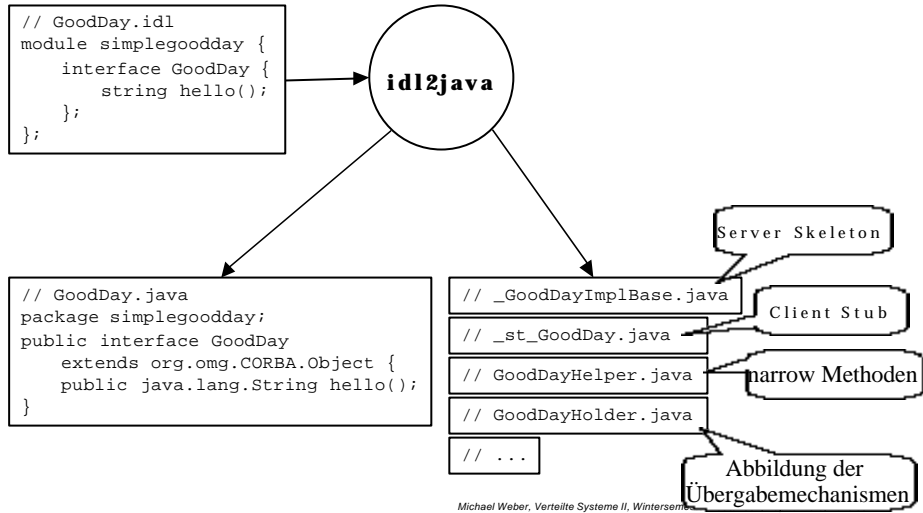
Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 37

# CORBA IDL Beispiel

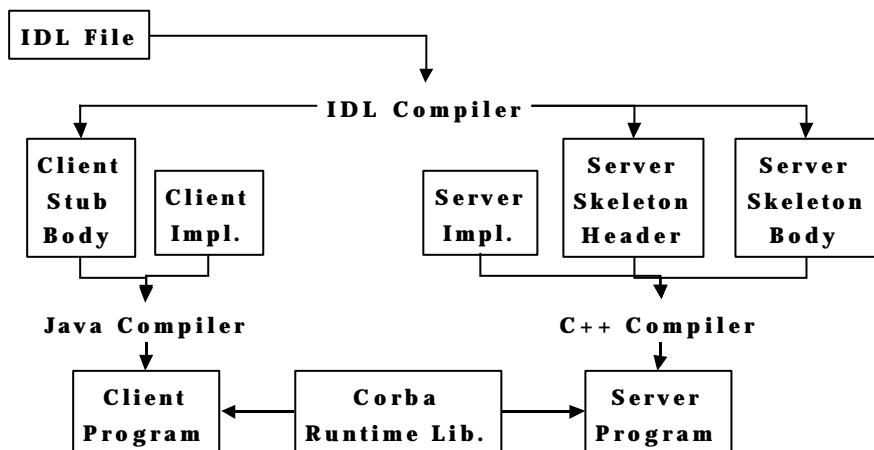


Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 38

# CORBA IDL Compiler



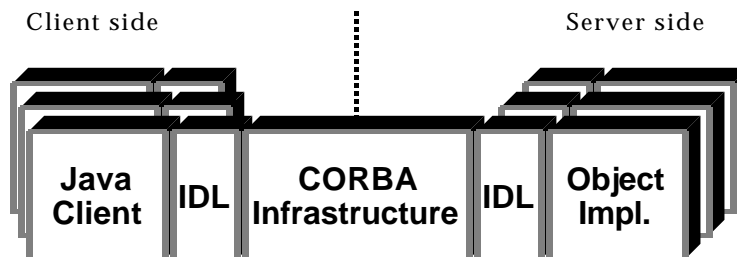
# CORBA IDL Compiler



## CORBA IDL

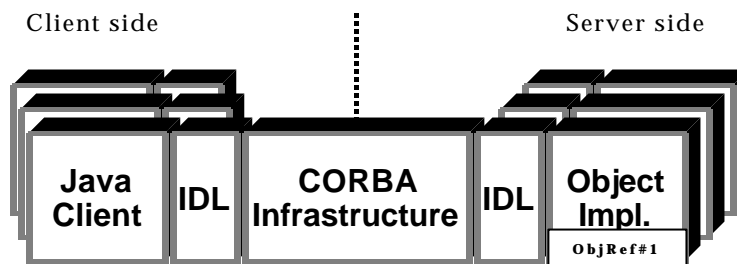
```
// complex.idl
module de {
  module uni-ulm {
    interface Meeting {
      readonly attribute string purpose;
      readonly attribute string participants;
      oneway void destroy();
    };
    interface MeetingFactory {
      Meeting CreateMeeting(in string purpose, in string part);
    };
    interface Room {
      enum Slot { am9, am10, pm12, pm1, pm2, pm3, pm4};
      const short MaxSlots = 8;
      typedef Meeting Meetings[MaxSlots];
      exception NoMeetingInThisSlot{};
      exception SlotAlreadyTaken{};
      readonly attribute string name;
      Meetings View();
      void Book(in Slot a_slot, in Meeting a_meeting)
        raises(SlotAlreadyTaken);
      void Cancel(in Slot a_slot) raises(NoMeetingInThisSlot);
    };
  };
};
```

## Object Invocation



Es gibt eine Reihe verschiedener  
Objektimplementierungen,  
jede mit ihrem eigenen IDL Interface.

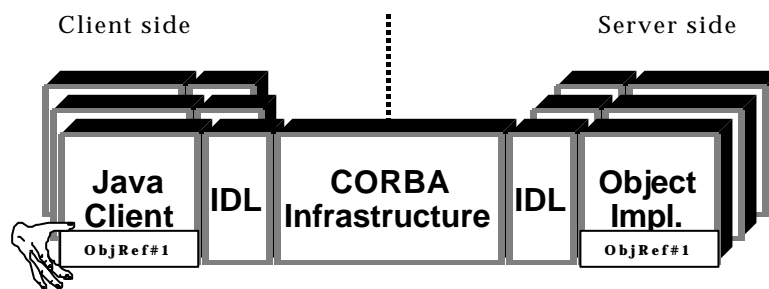
## Object Invocation



Jede Objektimplementierung erhält eine eindeutige Objektreferenz.

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 43

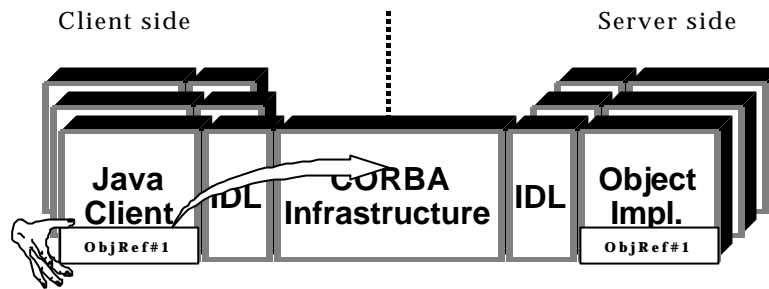
## Object Invocation



Klienten können die Objektreferenz über verschiedene Mechanismen erhalten.

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 44

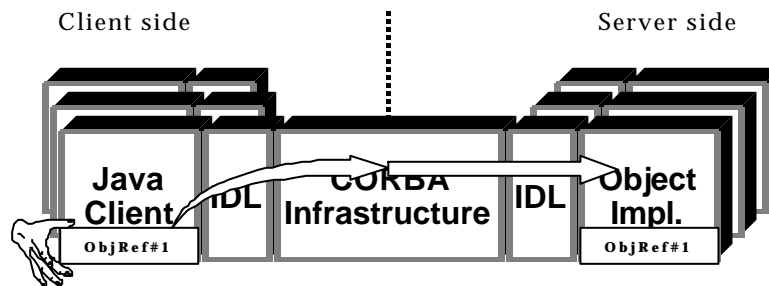
## Object Invocation



Klienten benutzen die Objektreferenz,  
um die CORBA Infrastruktur...

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 45

## Object Invocation



über die angefragte  
Objektimplementierung zu informieren.

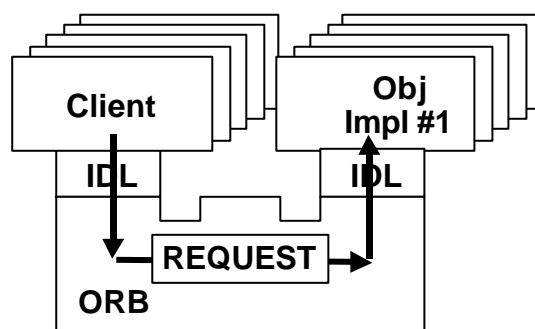
Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 46

## Rollen der CORBA Infrastruktur

- bietet einen lokalen, wohldefinierten Endpunkt für alle Objektaufrufe eines Klienten
- leitet den Aufruf an die Ziel-Objektimplementierung, lokal wie entfernt
- versteht IDL
- verwaltet ein Repository verfügbarer Objektinterfaces
- verwaltet ein Repository verfügbarer Objektimplementierungen
- föderiert diese Information im Gesamtsystem
- realisiert als ein Netz verbundener ORBs

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 47

## CORBA Architektur



ein Object Request Broker leitet den Aufruf von einem Klienten zur Objektimplementierung und das Resultat zurück zum Klienten.

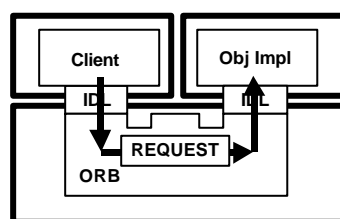
Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 48

## Möglichkeiten der ORB Realisierung

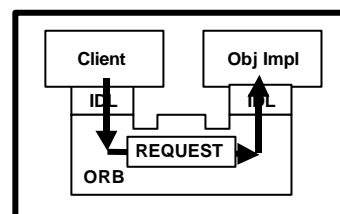
- Client- und Implementierungs-resident
  - ORB ist implementiert als Bibliothek
  - ORB ist resident in den Klienten und in der Objektimplementierungen
- Library-resident (single-process resident)
  - ORB und Objektimplementierungen sind als Bibliothek implementiert und Klienten-resident.
- Server-based
  - ORB ist als Server implementiert (separater Prozess)
  - ORB dient als Vermittler (broker) zwischen Klienten-Requests und Implementierungs-Prozessen (server).
- System-based
  - ORB ist Teil des Betriebssystems.

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 49

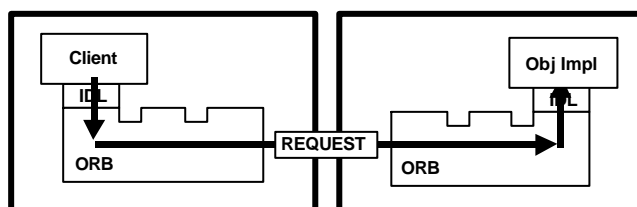
## verschiedene ORB Typen



Server or Operating-System Based



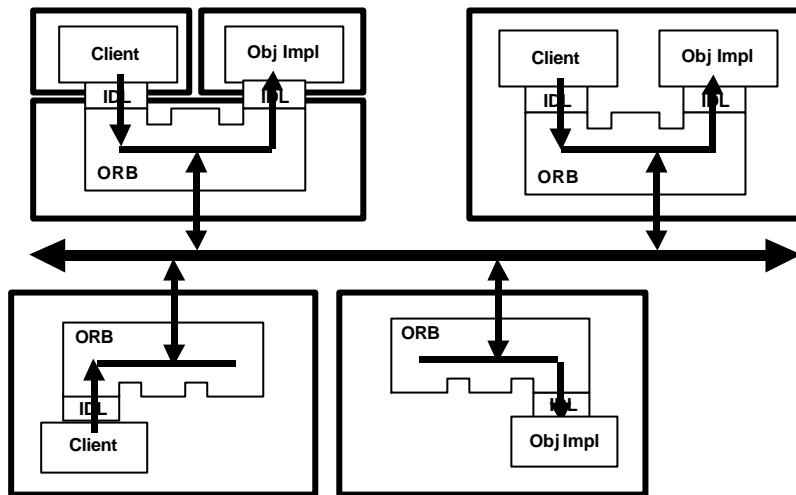
Single-Process Library Resident



Client & Implementation Resident

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 50

## ORB zu ORB Interoperabilität



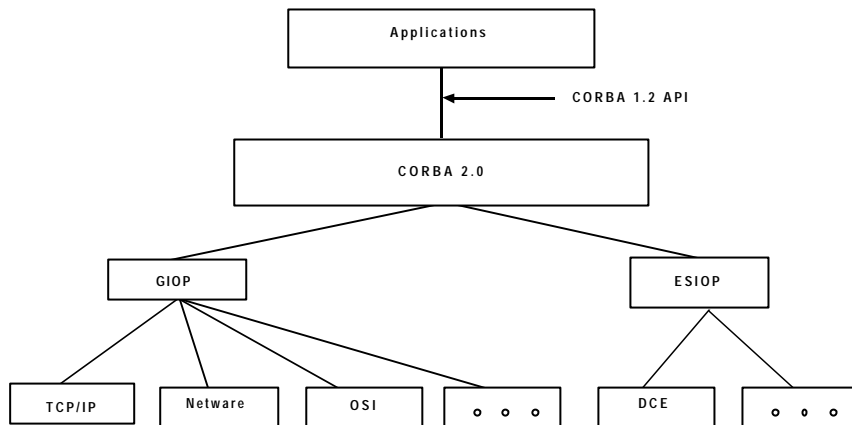
Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 51

## CORBA Interoperabilität

- CORBA 2.0 Interoperabilität umfasst:
  - Gesamtarchitektur für CORBA-CORBA Kommunikation;
  - API um Bridges einzufügen;
  - Multitransport Nachrichtenformat (General Inter-ORB Protocol, GIOP);
  - API für Gateways unter Verwendung von ESIOPs (Environment-Specific Inter-ORB Protocols);
- GIOP über TCP/IP (Internet Inter ORB Protocol, IIOB) ist zwingend um CORBA 2 konform zu sein
  - intern oder über eine Half-Bridge;
- DCE ESIOP ist optional.

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 52

## CORBA 2 Interoperability Spec



Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 53

## CORBA 2 Compliance



- Alle solche ORB-Produkte können mit jedem IIOP ORB interoperieren,
  - und dürfen das CORBA 2 Zeichen führen.

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 54

# GIOP

---

- **Common Data Representation (CDR)**
  - Transfersyntax
    - Mapping der IDL Datentypen in eine bikanonische low-level Representation
    - unterstützt variable Byteordnung und Alignment primitiver Typen
- **Nachrichtentransfer**
  - Request Multiplexing, d.h. auch Shared Connections
  - minimale Ordnungskriterien, d.h. auch asynchron mgl.
- **Nachrichtenformate**
  - Client: Request, CancelRequest, LocateRequest
  - Server: Reply, LocateReply, CloseConnection
  - Beide: MessageError

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 55

## Beispiel: GIOP Format

---

```
module GIOP {
  enum MsgType { Request, Reply, CancelRequest,
    LocateRequest, LocateReply, CloseConnection,
    MessageError };
  struct MessageHeader {
    char magic[4];
    Version GIOP_version;
    octet byte_order;
    octet message_type;
    unsigned long message_size;
  }
  struct RequestHeader {
    // ...
  }
}
```

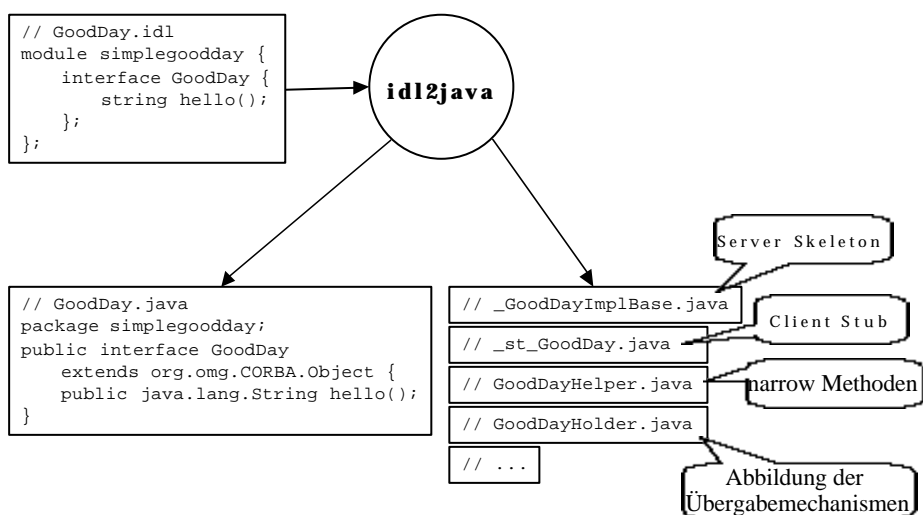
Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 56

# IIOP

- IIOP ist eine Spezifikation der GIOP Semantik für TCP/IP Verbindungsmanagement
- verfügbar in allen CORBA 2 ORBs
- auch in vielen anderen Anwendungen, z.B.
  - Netscape Navigator (enthält Visibroker)
  - viele Application Server (z.B. BEA Weblogic)

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 57

## Implementierung von CORBA Anwendungen



Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 58

## erzeugtes Server Skeleton

```
// _GoodDayImplBase.java
package simplegoodday;
abstract public class _GoodDayImplBase extends
    com.inprise.vbroker.CORBA.portable.Skeleton implements simplegoodday.GoodDay {
    protected simplegoodday.GoodDay _wrapper = null;
    public simplegoodday.GoodDay _this() {
        return this;
    }
    protected _GoodDayImplBase(java.lang.String name) {
        super(name);
    }
    public _GoodDayImplBase() {
    }
    // ... Stuff deleted
    public static boolean _execute(vbj.simplegoodday.corba.GoodDay _self,
        int _method_id, org.omg.CORBA.portable.InputStream _input,
        org.omg.CORBA.portable.OutputStream _output) {
        switch(_method_id) {
        case 0: {
            java.lang.String _result = _self.hello();
            _output.write_string(_result);
            return false;
        }
        }
        throw new org.omg.CORBA.MARSHAL();
    }
}
```

## Erzeugter Client Stub

```
// _st_GoodDay.java
package simplegoodday;
public class _st_GoodDay extends
    com.inprise.vbroker.CORBA.portable.ObjectImpl implements simplegoodday.GoodDay {
    protected vbj.simplegoodday.corba.GoodDay _wrapper = null;
    public vbj.simplegoodday.corba.GoodDay _this() {
        return this;
    }
    public java.lang.String hello() {
        org.omg.CORBA.portable.OutputStream _output;
        org.omg.CORBA.portable.InputStream _input;
        java.lang.String _result;
        while(true) {
            _output = this._request("hello", true);
            try {
                _input = this._invoke(_output, null);
                _result = _input.read_string();
            }
            catch(org.omg.CORBA.TRANSIENT _exception) {
                continue;
            }
            break;
        }
        return _result;
    }
}
```

## erzeugte Helper Class

```
// GoodDayHelper.java
package simplegoodday;
abstract public class GoodDayHelper {
    public static simplegoodday.GoodDay narrow(org.omg.CORBA.Object object) {
        return narrow(object, false);
    }
    private static simplegoodday.GoodDay narrow(org.omg.CORBA.Object object, boolean is_a) {
        // implementation deleted
    }
    public static vbj.simplegoodday.corba.GoodDay bind(org.omg.CORBA.ORB orb) {
        return bind(orb, null, null, null);
    }
    public static simplegoodday.GoodDay bind(org.omg.CORBA.ORB orb, java.lang.String name) {
        return bind(orb, name, null, null);
    }
    public static simplegoodday.GoodDay bind(org.omg.CORBA.ORB orb, java.lang.String name,
        java.lang.String host, org.omg.CORBA.BindOptions options) {
        // implementation deleted
    }
}
```

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 61

## erzeugte Holder Class

```
package simplegoodday;
final public class GoodDayHolder implements org.omg.CORBA.portable.Streamable
{
    public simplegoodday.GoodDay value;
    public GoodDayHolder() {
    }
    public GoodDayHolder(simplegoodday.GoodDay value) {
        this.value = value;
    }
    public void _read(org.omg.CORBA.portable.InputStream input) {
        value = simplegoodday.GoodDayHelper.read(input);
    }
    public void _write(org.omg.CORBA.portable.OutputStream output) {
        simplegoodday.GoodDayHelper.write(output, value);
    }
    public org.omg.CORBA.TypeCode _type() {
        return simplegoodday.GoodDayHelper.type();
    }
}
```

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 62

## Implementierung von CORBA Anwendungen

Was muss der Programmierer noch tun:

- eine Implementierung von `hello()` schreiben, die `_GoodDayImplBase` erweitert.
- einen Server, der die Implementierung instantiiert.
- einen Klienten, der den Server ruft.

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 63

## Implementierung von Hello

```
// GoodDayImpl.java
package simplegoodday;
import java.util.Date;
import org.omg.CORBA.*;
import simplegoodday.*;
public class GoodDayImpl extends _GoodDayImplBase {

    private String location;

    public GoodDayImpl( String location ) {
        super();
        // initialize location
        this.location = location;
    }

    // hello method
    public String hello() {
        return "Hello World from " + location + "!";
    }
}
```

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 64

## Implementierung des Servers

```
// GoodDayServer.java
package simplegoodday;
import java.io.*;
import org.omg.CORBA.*;
import simplegoodday.*;
public class GoodDayServer {
    public static void main(String[] args) {
        try {
            ORB orb=ORB.init(args, null);          // init ORB
            GoodDayImpl goodDayImpl=new GoodDayImpl(args[0]);
                                                    // create a GoodDay Object;
            orb.connect(goodDayImpl);              // export the object reference
            System.out.println(orb.object_to_string(goodDayImpl));
                                                    // print stringified object reference
            java.lang.Object sync = new java.lang.Object(); // wait for requests
            synchronized (sync) {
                sync.wait();
            }
        } catch (Exception e) {
            System.err.println(e);
        }
    }
}
```

## Implementierung des Client

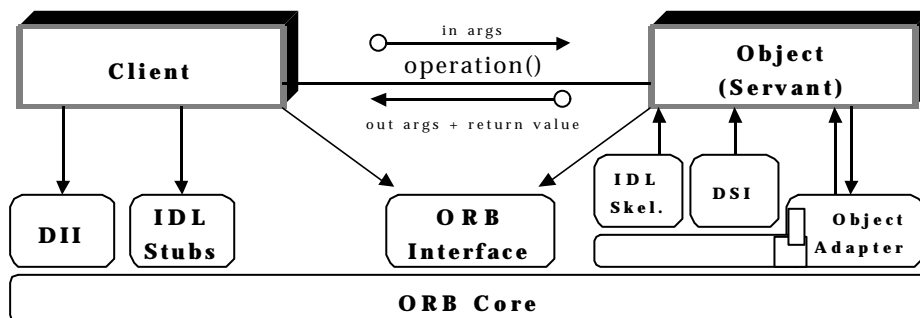
```
// GoodDayClient.java
package simplegoodday;
import java.io.*;
import org.omg.CORBA.*;
import simplegoodday.*;
public class GoodDayClient {
    public static void main(String args[]) {
        try {
            ORB orb=ORB.init(args, null);          //init orb
            org.omg.CORBA.Object obj=orb.string_to_object(args[0]);
                                                    // get object reference from command line
            GoodDay goodDay = GoodDayHelper.narrow(obj);
                                                    // cast the IIOR to a specific interface
            if (goodDay==null) { // check for errors
                System.err.println("stringified object reference" +
                    "is of wrong type");
                System.exit(-1);
            }
            System.out.println(goodDay.hello()); // call remote method
        } catch (SystemException ex) {
            System.err.println(ex);
        }
    }
}
```

## Fehlende Bestandteile

- Dynamisches Auffinden von Objekten (Object Discovery)
  - Naming Service
- Dynamische Objekt-Instantiierung
  - Basic Object Adapter (BOA) und Portable Object Adapter (POA)
- Parameterübergabe-Mechanismen
  - IDL: in, out, inout, oneway
- Klassen müssen \*ImplBase erweitern
  - tie approach
- Stub/Skeleton müssen bei Klient/Server vorhanden sein
  - Dynamic Invocation Interface (DII)
  - Dynamic Skeleton Interface (DSI)

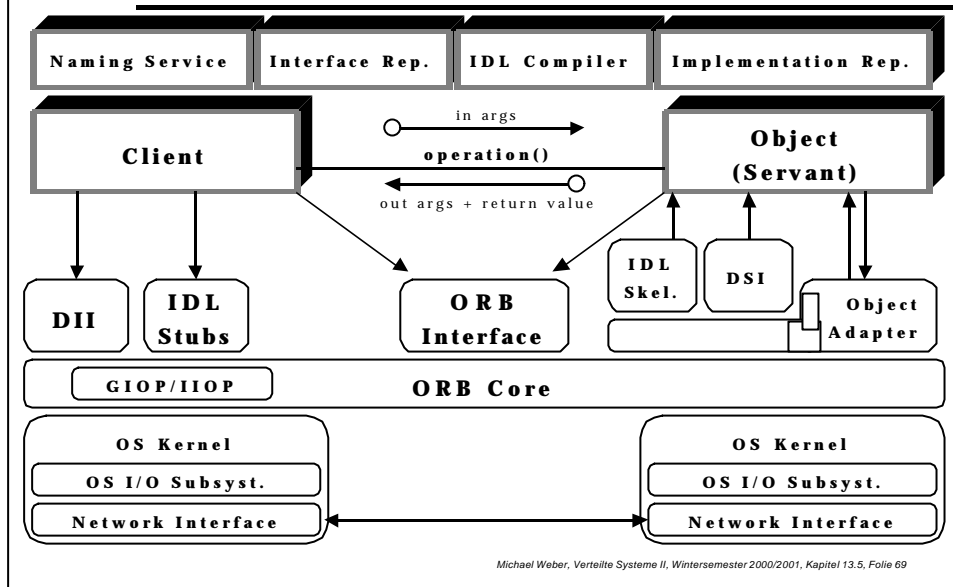
Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 67

## CORBA ORB Architektur



Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 68

# CORBA ORB Architektur



## Static Invocation Interface (SII)

- alle Operationen sind zur Compilezeit festgelegt
- sie sind Klient und Server durch Proxies bekannt (Stubs and Skeletons)
- Stubs verpacken Operationsaufrufe in Request-Nachrichten (marshalling)
- Skeletons entpacken die Nachrichten, rufen die Implementierung und rückübertragen die Antwort
- SII ist einfach, typsicher und effizient

## **SII Performance**

---

- Ca. 700 kByte/s bei in-Parametern  
(ca. 5,6 Mbit/s)
- ca. 520 kByte/s bei out-Parametern  
(ca. 4,1 Mbit/s)
- ca. 330 kByte/s bei inout-Parametern  
(ca. 2,6 Mbit/s)

Quelle: Diplomarbeit von Jochen Voegel,  
Messungen mit Standard PCs in 10Mb/s Ethernet

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 71

## **Dynamic Invocation Interface (DII)**

---

- DII erlaubt Klienten dynamisch:
  - Objekte zu finden;
  - Objekt-Interfaces zu finden;
  - Requests (Methodenaufrufe) zu erzeugen;
  - Methodenaufrufe abzusetzen;
  - Ergebnisse zu empfangen.

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 72

## Dynamic Invocation Interface (DII)

- ermöglicht Klienten den Aufruf von Objektoperationen, die erst zur Laufzeit bekannt sind.
  - e.g. MIB Browsers
- ermöglicht Klienten Argumente auf einen Request Stack zu “pushen” und Operationen über einen ASCII Namen zu identifizieren
  - Typchecks erfolgen über Metainformationen aus dem Interface Repository
- flexibler als SII (z.B. deferred synchronous Semantik)
- aber komplizierter, weniger typsicher, ineffizienter

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 73

## DII Beispiel (Ausschnitt aus einem Zählerprogramm)

```
int ctrInValue = 10; int ctrOutValue;
org.omg.CORBA.Object counter = // acquire a reference
    // Initialize the ORB.
ORB _orb = ORB.init();
    // Set up the argument (as an NVList)
NVList _arguments = _orb.create_list(1);
Any _input = _orb.create_any();
_input.insert_long(ctrInValue);
_arguments.add_value("inc", _input, ARG_IN.value);
    // Set up the result container (as a NamedValue)
Any _result = _orb.create_any();
_result.type(_orb.get_primitive_tc(TCKind.tk_long));
NamedValue _resultNV = _orb.create_named_value(null, _result, 0);
    // Create the request
Request _request = counter._create_request(null, "increment",
    _arguments, _resultNV);
    // Invoke the operation
_request.invoke();
    // Get a hold of the result
ctrOutValue = _result.extract_long();
System.out.println("Value returned from counter = " + ctrOutValue);
```

## **Static und Dynamic Skeleton Interface**

- SSI wird vom IDL Compiler generiert
  - SSI Aufgaben:
    - Operations-Dispatching
    - Parameter-Demarshalling
- DSI ist analog zu DII auf Klientenseite
  - vorwiegend zum Bau von ORB Bridges gedacht
  - Server kann beliebige Invokationen von CORBA Objekten behandeln

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 75

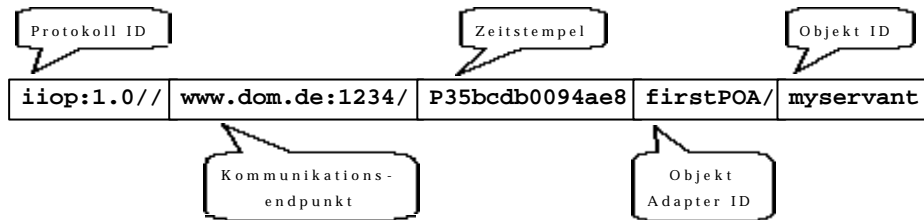
## **Objektreferenzen**

- transparenter Handle eines Objekts
  - identifiziert die Objektlokation
- Objektreferenz kann unter Prozessen weitergerichtet werden, auch entfernt
  - der CORBA ORB konvertiert die Objektreferenz in eine Netzwerk-Transferform
  - (Internet) Interoperable Object Reference ((I)IOR)
- Unterstützung von z.B.
  - Peer-to-Peer Interaktionen
  - verteilte Callbacks

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 76

## Beispiel: URL IOR

---



Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 77

## Basic Object Adapter

---

- Objektimplementierungen nutzen den Objektadapter, um sich beim ORB bekannt zu machen
- Objektadapter starten Implementierungen bei Bedarf (aus dem Implementation Repository)
- sie managen die Laufzeitumgebung des Objekts
- BOA ist unterspezifiziert
  - jeder ORB implementiert den BOA anders
  - d.h. hier ist CORBA nicht interoperabel !!

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 78

# Basic Object Adapter

```
// init ORB
ORB orb = ORB.init(args, null);

// init BOA
BOA boa = ((com.visigenic.vbroker.orb.ORB)orb).BOA_init();

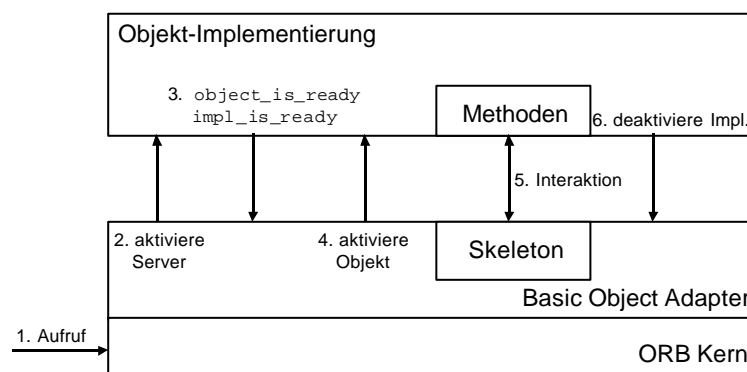
// create a GoodDay Object;
GoodDayImpl goodDayImpl = new GoodDayImpl( "corba/GoodDay", args[0] );

// export the object reference
boa.obj_is_ready( goodDayImpl );

// wait for requests
boa.impl_is_ready();
```

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 79

# BOA während der Laufzeit



Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 80

## Portable Object Adapter

---

- erzeugt Objektreferenzen
- aktiviert und deaktiviert Objekte
- suspendiert und reinkarniert Server
- bildet Aufrufe auf Server ab
- wichtig für z.B.
  - Telecom MIBs
  - Enterprise Server und Application Server

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 81

## POA Entwurfsziele

---

- Server sind portabel zwischen ORBs
- persistente und transiente Objekte
- transiente Objekte mit möglichst geringem Programmieraufwand und wenig Overhead
- transparentes Aktivieren und Deaktivieren von Servern
- implizite und explizite Serveraktivierung
- ein einzelner Server kann mehrere Objekte unterstützen
- mehrere (auch geschachtelte) Instanzen des POA in einem Serverprozess möglich
- das POA-Verhalten wird durch Creation Policies festgelegt
- Server können von Skeletons erben oder DSI nutzen

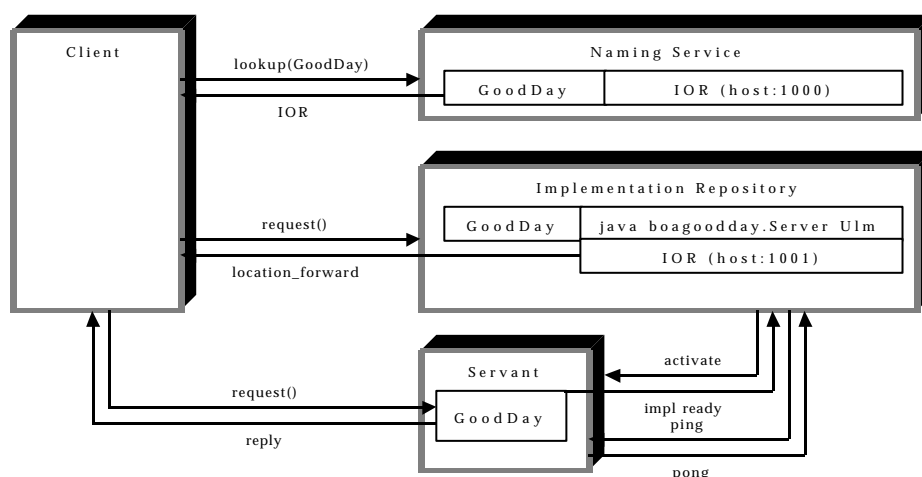
Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 82

# Implementation Repository

- erlaubt dem ORB angefragte Server zu aktivieren
- speichert Management-Information zu dem Objekten
- arbeitet primär mit persistenten Objektreferenzen
- aus Klientensicht ist das Verhalten aller IRs gleich
- die Realisierung ist spezifisch bzgl. ORB und Betriebssystemumgebung, also nicht portabel

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 83

# Implementation Repository



```
C:\> oadutil reg -i boagoodday::GoodDay -o GoodDay -java boagoodday.Server -a Ulm -p shared
```

## Server Activation Modes

---

- ein Server, der idle ist, wird automatisch gestartet, wenn eines seiner Objekte gerufen wird
- Server Activation Modes:
  - *shared* ⇒ ein Prozess für alle Objekte
  - *unshared* ⇒ jedes Objekt erhält seinen eigenen Prozess
  - *per-operation call* ⇒ jeder Aufruf erhält seinen eigenen Prozess
  - *persistent* ⇒ Prozess wird “von Hand” gestartet

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 85

## CORBA Services (COS)

---

- Basisdienste, die viele verteilte Anwendungen bzw. OO-Anwendungen brauchen
- Standard Interfaces sorgen für Portabilität und Interoperabilität der Basisfunktionen
- nicht alle Dienste werden von allen Herstellern implementiert

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 86

## COS

---

- Naming
- Events / Notification
- Life Cycle
- Persistent Object
- Relationship
- Externalization
- Transactions
- Concurrency Control
- Licensing
- Query
- Properties
- Security (incl. IIOP over SSL)
- Time
- Collections
- Trading
- Component (neu)

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 87

## COS

---

- Lifecycle
  - Erzeugen und Löschen von Objekten.
- Naming
  - Abbildung von Objektnamen auf Objektreferenzen.
- Event Notification
  - Registrierung von Listeners, Zustellung von Ereignissen.
- Persistence
  - Verwaltung persistenter Objekte, zugehöriges Speichermanagement.
- Relationships
  - Repräsentation und Konsistenzmanagement von Beziehungen zwischen Objekten.
- Externalization
  - Auslagerung von Objektrepräsentationen auf externen Speichermedien (z.B. CD), Internalisierung.

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 88

## COS

---

- **Transactions**
  - Verwendung von Objekten in transaktionaler Semantik.
- **Concurrency Control**
  - Management nebenläufiger Ausführungen.
- **Security**
  - Framework für verschiedene Sicherheitstechnologien.
- **Properties**
  - Objekte können mit (Zusatz-)Attributen versehen werden.
- **Query**
  - SQL Anbindung.
- **Licensing**
  - Lizenzverwaltung.
- **Trading**
  - Vermittlung von Objekten auf der Basis von Attributverhandlungen.

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 89

## CORBA Facilities

---

- **Spezifikation von Anwendungsdiensten**
- **horizontale Facilities**
  - können in vielen Bereichen eingesetzt werden
  - z.B. User Interface, Systemmanagement, Dokumentenmanagement, Informationsmodellierung, Workflow
- **vertikale Facilities**
  - sind spezifisch für ein Anwendungsgebiet
  - z.B. Telekommunikation, Medizin, Finanzwesen, Verkehr

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 90

## Zustand bzgl. Interoperabilität

- CORBA 1 Spezifikation war sehr unvollständig bzgl. Interoperabilität
- CORBA 2.x definiert eine brauchbare Interoperabilitäts-Spezifikation
  - spätere Erweiterungen erfassen die serverseitige Portabilität (d.h. POA spec)
- die meisten ORB Implementierungen unterstützen heute IIOP oder GIOP robust
- fast immer sind die Dienste (COS) nicht interoperabel

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 91

## Neuere Standardisierungen

- Standardisierte Spezifikationen:
  - IDL Sprachbindungen (z.B. C, C++, Java)
  - Naming Service, Event Service, Lifecycle Service
  - ORB Initialization Service
  - Portable Object Adapter API
  - Servant Mapping, d.h. welches Objekt läuft in welchem Server und wie wird es aktiviert
- die Portierung von Anwendungen von ORB zu ORB ist limitiert, bis Conformance Tests definiert und vorgeschrieben sind
  - <http://www.opengroup.org/testing/testsuites/vsorb.htm>

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 92

## Limitierungen

---

- Standard CORBA “kümmert” sich nicht um klassische VS-Probleme wie z.B.:
  - Latenz
  - Fehlertoleranz (RFP in Arbeit, z.B. von Sun, Oracle)
  - Realtime Erweiterungen (RFP in Arbeit)
  - Kausale Ordnungen
  - Deadlockbehandlung

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 93

## Limitierungen

---

- die meisten ORBs unterstützen den Übergabemechanismus “Object by Value” nicht
  - Lösung mit CORBA 2.3/3.0
- die meisten ORBs unterstützen nur folgende Semantik:
  - Objektreferenzen werden per Referenz übergeben
  - structs und unions werden per Wert übergeben
  - Object by Value muss von Hand durch Factories nachgebildet werden

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 94

## Limitierungen

- die meisten ORBs unterstützen keine asynchronen Methodenaufrufe oder Timeouts
- Versionierung wird in IDL durch Pragmas unterstützt
  - bei ONC RPC bzw. DCOM ist das in der Schnittstellenbeschreibung integriert

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 95

## Performance Einschränkungen

- Leistung ist eingeschränkt gegenüber “von Hand Programmierung” wegen:
  - zusätzliche Aufrufe zur Namensauflösung
  - Marshalling/Demarshalling Overhead
  - zusätzliche Datenkopien und Speichermanagement
  - Endpunkt- und Aufruf-Demultiplexing
  - Kontextwechsel und Synchronisations-Overhead
  - Trade off zwischen Performance und Erweiterbarkeit, Robustheit, Wartbarkeit

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 96

## CORBA 3

---

- **Verbesserte Java und Internet Integration**
  - Java-to-IDL Reverse Mapping
  - Firewall Spezifikation
  - CORBA Object URLs
- **Quality of Service Control**
  - Asynchrone Aufrufe, asynchrones Messaging
  - Invocation QoS Control
  - Realtime, Minimum, Fault Tolerance
- **CORBA Component Model**
  - Object by Value Übergabemechanismus
  - Component Container
    - Transaktionell, Persistent, Sicher
  - Auslieferungsformat
  - Spezifikation einer Scriptsprache zur Anpassung der Komponenten

Michael Weber, Verteilte Systeme II, Wintersemester 2000/2001, Kapitel 13.5, Folie 97