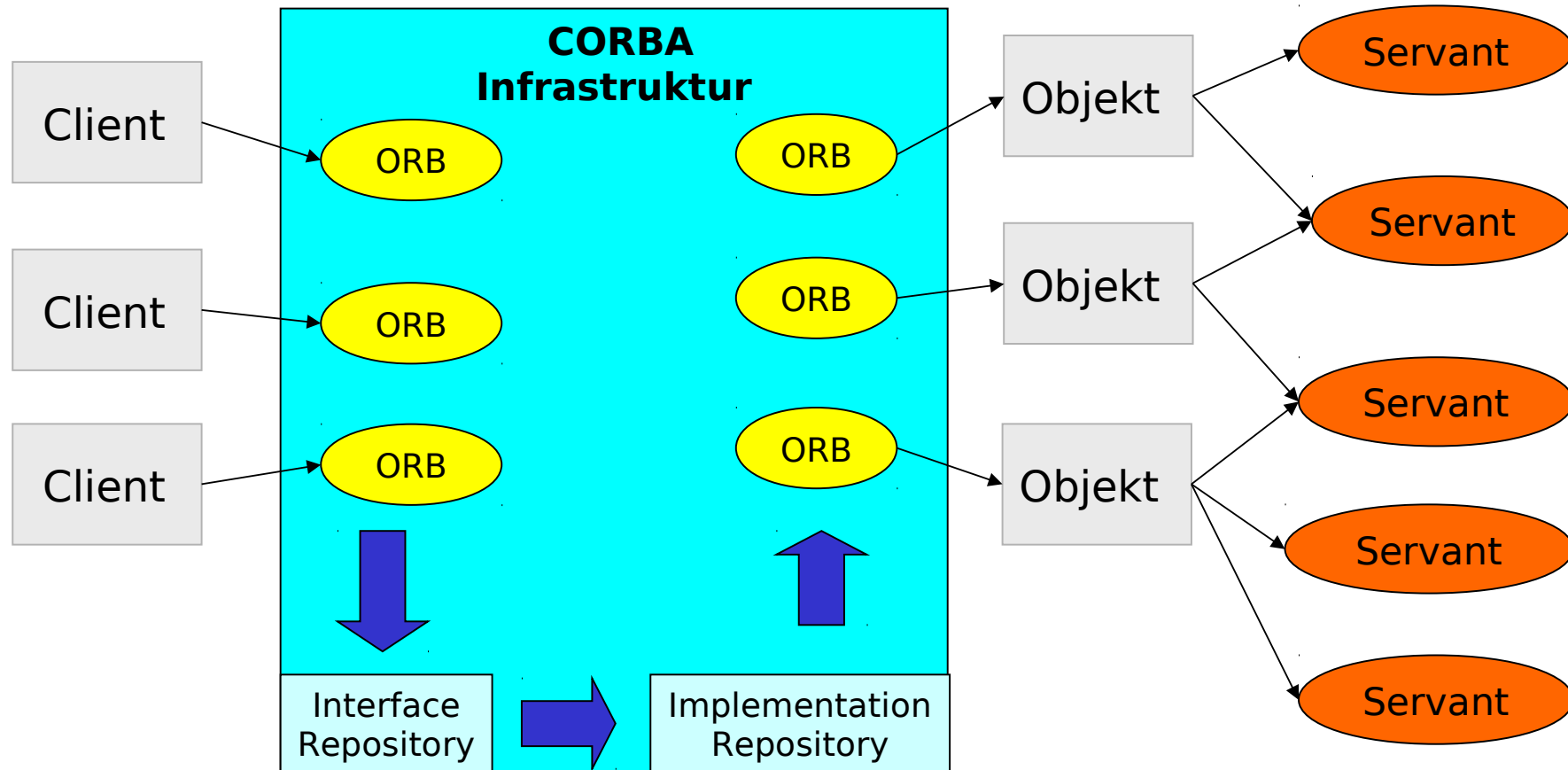


# **Vorlesung Software aus Komponenten**

## **3. Komponentenmodelle**

Prof. Dr. Hans-Gert Gräbe  
Wintersemester 2012/13

## Architektur im Überblick

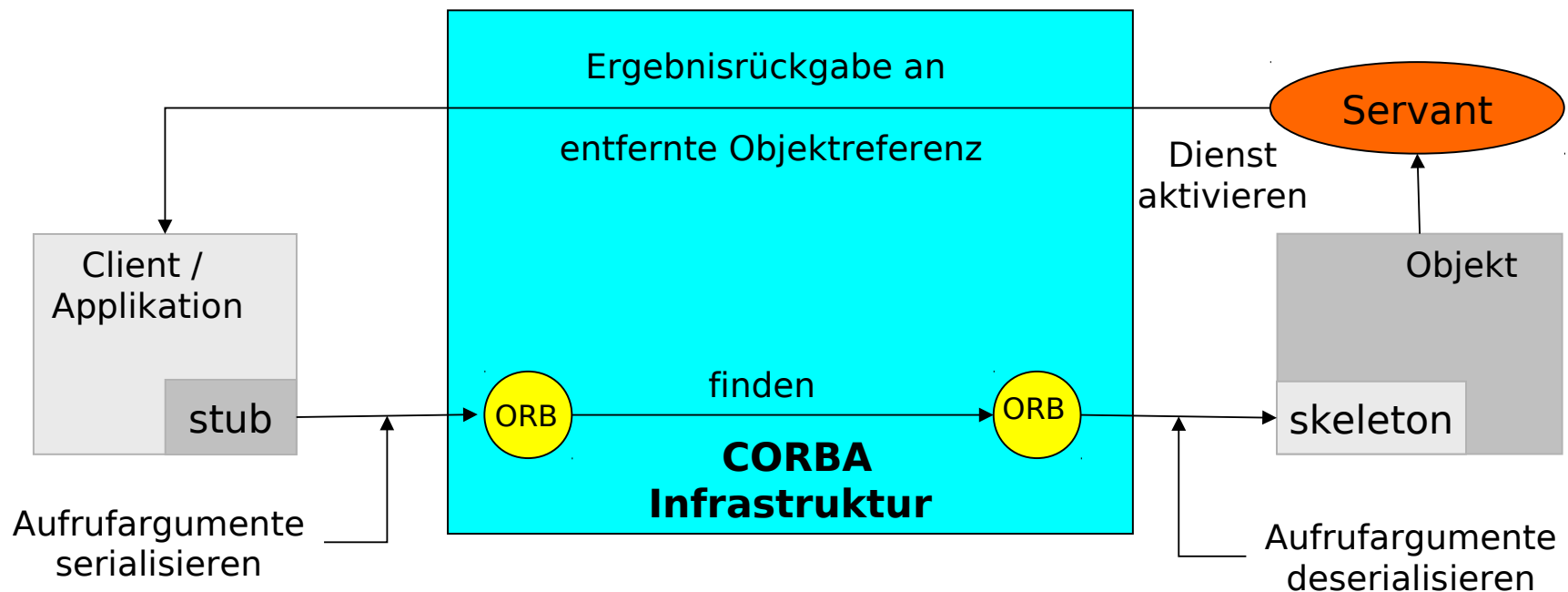


### Stummel (stubs) und Skelette (skeletons)

Methodenaufrufe erfolgen über Stummel-Skelett-Prinzip des RPC

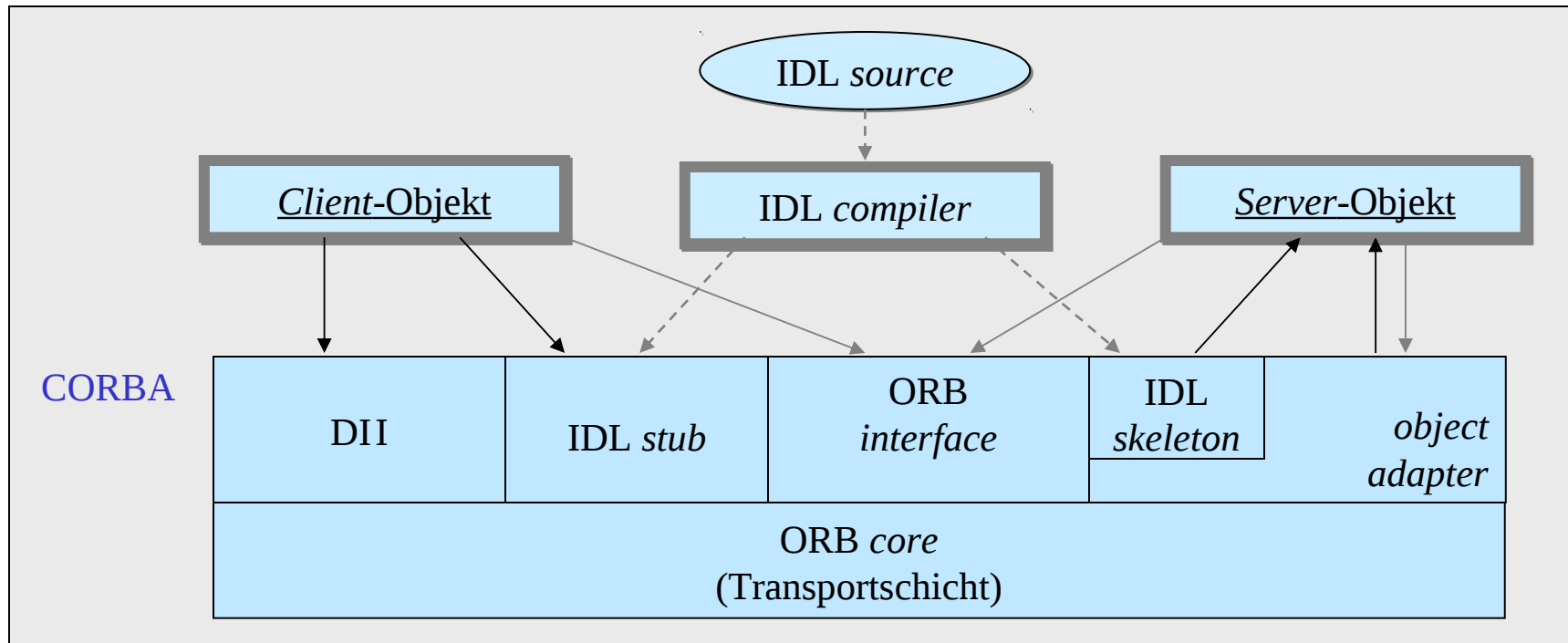
- mit OMG IDL Compiler aus Schnittstellenbeschreibung generierbar

Direkt nur für statische Methodenaufrufe einsetzbar (static invocation interface SII / static skeleton interface SSI)



### Dynamische Methodenaufrufe (dynamic invocation interface - DII)

- Erforderlich, um Methoden zur Laufzeit binden zu können
- Seit CORBA 2.0 auch Dynamik auf der Serverseite (dynamic skeleton interface - DSI)
- Verwenden eine **universelle Datenstruktur für Argumente**, um Methoden mit (statisch) unbestimmter Signatur zu behandeln
- Aus Geschwindigkeits-Gründen wird SII / SSI zusätzlich bereitgestellt.



### Symmetrie des CORBA-Modells

Keine Asymmetrie wie im Client-Server-Modell: Jeder Prozess kann sowohl Methodenaufrufe absetzen als auch empfangen.

Einzige Asymmetrie kommt durch den **Objektadapter**.

- Programme, die als Servant eingesetzt werden, müssen sich beim ORB durch einen Objektadapter registrieren
- erster Standard: basic object adapter (BOA), deprecated seit 1998
  - war unterspezifiziert, deshalb Wildwuchs von Erweiterungen
- heute: portable object adapter (POA)
  - aktueller Standard ist Teil von CORBA 3.0.3, März 2004
- Mit der Registrierung „weiß“ der Objektadapter (und nur dieser), wie der Servant aktiviert wird
- jedes Objekt hat eine „Hausmaschine“, auch wenn ein Servant über mehrere Maschinen „verfügen“ kann
- Reine Applikationen, die keine Dienste zur Verfügung stellen, sondern nur welche nutzen, werden auch nicht registriert. Können damit auch nicht durch CORBA gestartet werden.

### 3.3. Corba

## Object Request Broker (ORB)

### Aufgaben des ORB

- Schlüsselkomponente und Kommunikationszentrale der Architektur
- vermittelt Methoden-Aufrufe zwischen Applikationen und Servanten
  - arbeitet nur mit den Schnittstellen (stub, skeleton)
  - Schnittstellen-Definition über OMG IDL

Verwendet dazu Informationen aus dem **Schnittstellen-Repository** (interface repository - IR)

CORBA Begriffe: **Dienste** werden über **CORBA-Objekte** angesprochen, deren Methoden mit den entsprechenden **Servanten** verbunden sind.

ORB verantwortlich für Suche von CORBA-Objekten, über deren Methoden der jeweilige Dienst erbracht wird

- Verwendet dazu Informationen aus dem **Repository der Implementierungen** (implementation repository)

## Aufgaben des ORB (Fortsetzung)

- Zusammenspiel von ORB, Objektadapter (Dienst) und Stummel (Nutzer) beim Auflösen dynamischer Methodenbindungen
  - ORB liefert relevante Interfacedefinitionen aus dem IR
  - Stummel löst dynamische Bindung von Aufrufen auf (DII)
  - Objektadapter realisiert Auflösung von Objektreferenzen
- Verwaltung der CORBA-Objekte
  - Aktivierung und Deaktivierung
  - Policy-Operationen
  - Verwaltung zugeordneter leichtgewichtiger Prozesse (Threads)
- **Unterscheide zwischen ORB als Klasse (Konzept der Architektur) und ORB-Instanzen (Laufzeitobjekte)**
  - Aufbau und Organisation einer Implementierung des ORB als Klasse ist von Anbieter und Einsatzgebiet abhängig.
  - Kann als einzelner Prozess oder verteilte Anwendung implementiert sein

## CORBA-Objekte

CORBA-Objekte sind Programmfragmente mit Eigenleben, charakterisiert durch

- Objektzustand (aktuelle Werte der Attribute)
- Funktionalität (verfügbare Methoden)

Dienste eines CORBA-Objekts können von Applikationen nur über den ORB in Anspruch genommen werden.

ORB organisiert Aktivierung und Deaktivierung von CORBA-Objekten und Diensten

- Anforderung entsprechender Ressourcen (CPU, Speicher)
- Sicherung der persistenten Bestandteile bei Deaktivierung
- Aktivierungs- und Deaktivierungsmuster können durch entsprechende Regeln (policies) festgelegt sein

CORBA-Objekte sind damit Zwischending zwischen Komponenten und Objekten im Sinne der Vorlesung

Jedes CORBA-Objekt hat einen Typnamen ( = Klasse in Java )

- Typname entspricht dem Schnittstellennamen in der IDL Deklaration
- Typname steht für einen abstrakten Datentyp als Menge von Methoden und deren Signaturen sowie Variablen (Attributen) und deren Typen

## CORBA Objektreferenzen

Statt Objekten werden normalerweise nur Referenzen übergeben

- Seit CORBA 2.3 können Objekte auch als Wertparameter übergeben werden
- Verwendet eine **standardisierte Serialisierung**

Referenz über Programmgrenzen statt Referenz innerhalb eines Programms

- Kann Objektänderungen durch die Evolution des Programmstatus im Ursprungsprogramm nicht verfolgen
- Damit teurer in der Handhabung als physische Referenzen und eher vergleichbar mit einer URL
  - Seit CORBA 2.3 existiert Standard zur Darstellung von Objektreferenzen als URL
- ORB Schnittstelle enthält **Methoden zum Umwandeln** zwischen physischen Referenzen und CORBA Objektreferenzen
- Problem des Call Backs – physische und CORBA Objektreferenz können im lokalen Kontext koexistieren.

Lebensdauer per Definition **unbestimmt**

- Wiederverwendung einer Referenz kann einen Fehler auslösen
- referenziertes Objekt muss nicht mehr existieren

### OMG IDL und Datentypen

OMG IDL unterscheidet primitive Datentypen und CORBA Objektreferenzen

- Basistypen (integer, float, char, string)
- zusammengesetzte Datentypen
  - Strukturen, Sequenzen, Aufzählungstypen
  - multi-dimensionale Felder fester Größe

Parameter eines primitiven Datentyps werden als Wertparameter übergeben

Umfang der Unterstützung ist von eingesetzter Programmiersprache abhängig

- CORBA Standard: Aufruf eines nicht unterstützten Typs erzeugt einen Fehler zur Übersetzungszeit
- Damit kann Sprache verwendet werden, die nur einen Teil des Standards implementiert, wenn auch nur dieser Teil verwendet wird.

## Java und CORBA

- CORBA als Interoperations-Standard muss an konkrete Programmiersprachen **gebunden** werden
- Seit CORBA 2.2 (1998) gibt es OMG IDL to Java binding sowie Java to OMG IDL reverse binding
  - Java als die wichtigste CORBA-Referenzimplementierung
- Reverse binding interessant für Anbindung von Nicht-Java-Systemen an Java-Systeme über IIOP-Standard von CORBA
- Heute Koexistenz in fast allen Applikationsserver-Produkten

Entwicklung einer CORBA-Anwendung unter JAVA  
(aus "Lehrbuch der Softwaretechnik" von Helmut Balzert)

- 1) Spezifikation der Schnittstelle in CORBA-IDL
- 2) Übersetzung mittels IDL-Compiler
  - Stummel- und Skelettklassen werden erzeugt
- 3) Implementierung der Operationen der Schnittstelle
- 4) Rahmenanwendung entwickeln
  - Erzeugt Objekt der Klasse
  - Objekt wird für Clients zugreifbar gemacht
- 5) Entwickeln des Clients

## Definition der Schnittstelle mit OMG IDL

```
module SemOrg { // Schnittstelle der Klasse Firma
  interface Firma {
    attribute string Name;
    attribute float Umsatz;
    // Operationssignatur
    float berechneGewinn( in float Kosten );
  };
};
```

SemOrg.idl

**'idlj -f all SemOrg.idl'** erzeugt daraus Java-Package **SemOrg**

## 3.3. Corba

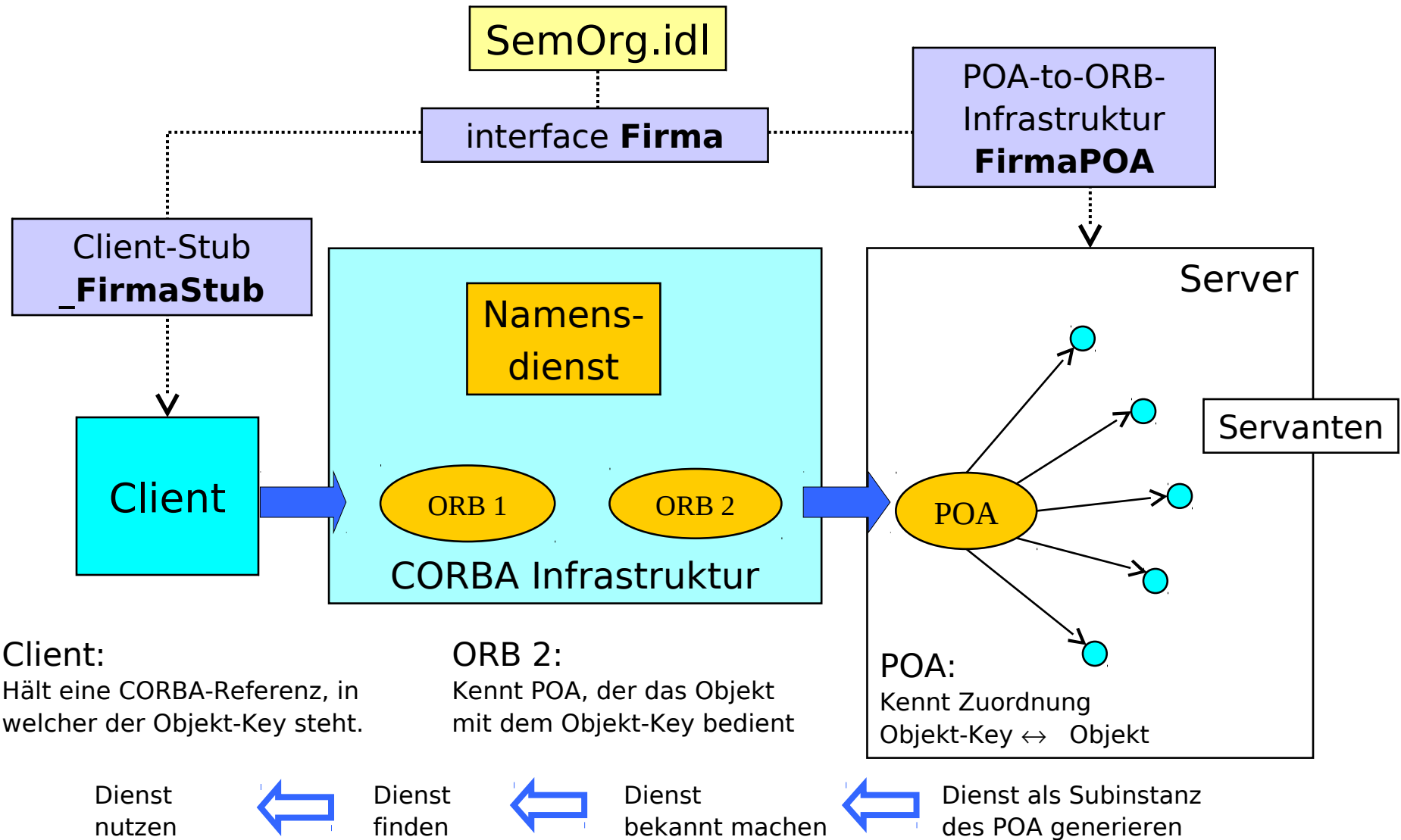
### CORBA-Beispiel Seminarorganisation

#### Vom IDL – Compiler erzeugte Klassen

- interface **FirmaOperations**
  - interface Firma als Java-Interface
- interface **Firma** extends SemOrg.FirmaOperations, ...
  - Schnittstelle mit speziellen Firma-Operationen und generischen CORBA.Object-Operationen
- class **\_FirmaStub** - voll generierte Stummel-Klasse für den Client
- final class **FirmaHolder** implements CORBA.portable.Streamable
- class **FirmaHelper**
  - Statische Methoden, um CORBA.Any auf Semorg.Firma zu casten
- abstract class **FirmaPOA** extends PortableServer.Servant implements SemOrg.FirmaOperations ....
  - portabler Objekt-Adapter
  - generierte Implementierung der ORB-seitigen Kommunikation zur Implementierung der Fachlogik in einer von FirmaPOA abgeleiteten Klasse (auf diese Methoden wird „abstract“ Bezug genommen)

### 3.3. Corba

## CORBA-Beispiel Seminarorganisation



#### Implementierung des Servanten

```
package SemOrg;
public class FirmaImpl extends FirmaPOA {
    private String derName;
    private float derUmsatz;
    public FirmaImpl() {} // Konstruktor
    public float berechneGewinn(float Kosten)
        { return this.Umsatz - Kosten; }
    // get-/set-Operation für Attribut Name
    public String Name() { return this.derName }
    public void Name(String neuerName)
        { this.derName = neuerName; }
    //analog für Umsatz ...
}
```

### 3.3. Corba

## CORBA-Beispiel Seminarorganisation

### Default-Server-Modell: Portable Servant Inheritance Model

#### Aufgaben des Servers

- Verbindung zur ORB-Struktur herstellen über eigenen Broker
- Instanz des Servanten **FirmaImpl** anlegen und in der CORBA-Struktur verankern: als POA registrieren und den POAManager aktivieren.
- CORBA-Objektreferenz des Servanten besorgen
- Objektreferenz auf einen Namens-Kontext von Namens-Server der ORB-Infrastruktur holen und den Servanten als „eineFirma“ registrieren.
- Auf Anfragen warten

#### Aufgaben des Client

- Verbindung zur ORB-Struktur herstellen über eigenen Broker
- CORBA-Objektreferenz auf den Servanten „eineFirma“ über den Namens-Kontext besorgen
- Dienste des Servanten in Anspruch nehmen

### Serverkomponente

```
package SemOrg;
import java.io.*;
import org.omg.CosNaming.*;
import org.omg.PortableServer.*;

public class SemOrgServer {
    public static void main(String[ ] args) {
        try {
            // Kontakt zur ORB-Infrastruktur herstellen: ORB-Referenz holen
            org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, null);
            // POA-Referenz vom ORB besorgen und POA aktivieren
            POA poa=POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
            poa.the_POAManager().activate();
            // Servanten anlegen als reales Java-Objekt
            FirmaImpl impl = new FirmaImpl();
```

### 3.3. Corba

## CORBA-Beispiel Seminarorganisation

```
// CORBA-Objektreferenz auf den Servanten besorgen und „casten“
org.omg.CORBA.Object implObjServer = poa.servant_to_reference(impl);
Firma eineFirmaS = FirmaHelper.narrow(implObjServer);
// vom ORB Referenz auf Namensdienst-Server besorgen und „casten“
org.omg.CORBA.Object ncObj =
    orb.resolve_initial_reference("NameService");
NamingContextExt ncRef = NamingContextExtHelper.narrow(ncObj);
// Servanten unter dem Namen "eineFirma" im Namensdienst bekannt
// machen. Verbindung zwischen "eineFirma" und dem Java-Objekt impl
// kennt nur poa
ncRef.rebind(ncRef.to_name("eineFirma"), eineFirmaS);
System.out.println(„SemorgServer gestartet ...“);
} // end try
catch (Exception e) { ... }
} // end main
}
```

### 3.3. Corba

## CORBA-Beispiel Seminarorganisation

#### Client-Komponente

```
package SemOrg;
import org.omg.CosNaming.*;

public class SemOrgClient {
    public static void main(String[ ] args) {
        try {
            // Verbindung zur ORB-Infrastruktur
            org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, null);

            // vom ORB Referenz auf Namensdienst besorgen
            org.omg.CORBA.Object ncObj = orb.resolve_initial_reference("NameService");
            NamingContextExt ncRef = NamingContextExtHelper.narrow(ncObj);
```

### 3.3. Corba

## CORBA-Beispiel Seminarorganisation

// CORBA-Objektreferenz auf das Servanten-Objekt „eineFirma“ besorgen

```
org.omg.CORBA.Object firmaObj = ncRef.resolve_str("eineFirma");
```

```
Firma eineFirmaC = FirmaHelper.narrow(firmaObj);
```

```
System.out.println("Handle auf das Server-Objekt"+eineFirmaC);
```

//Aufrufe durchführen

```
System.out.println(eineFirmaC.Name());
```

```
...
```

```
}
```

```
catch (Exception e) { ... }
```

```
}
```

```
}
```