

Vorlesung Software aus Komponenten

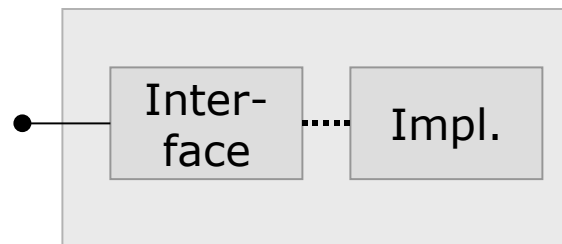
2. Grundlagen

apl. Prof. Dr. Hans-Gert Gräbe
Wintersemester 2008/09

Schnittstellen und Implementierungen

Drei wesentlich verschiedene Ansätze:

- 1 Impl. \Leftrightarrow 1 S (CORBA 2, SOM)
 - Objekt = Programmzustand (Kontrollfluss **und** Daten) und Implementierung **seiner** Schnittstelle
 - CORBA-Objektbegriff damit **zwischen** Komponente und Objekt im Sinne der Vorlesung
 - Schnittstelle kann durch Mehrfachvererbung entstanden sein



CORBA

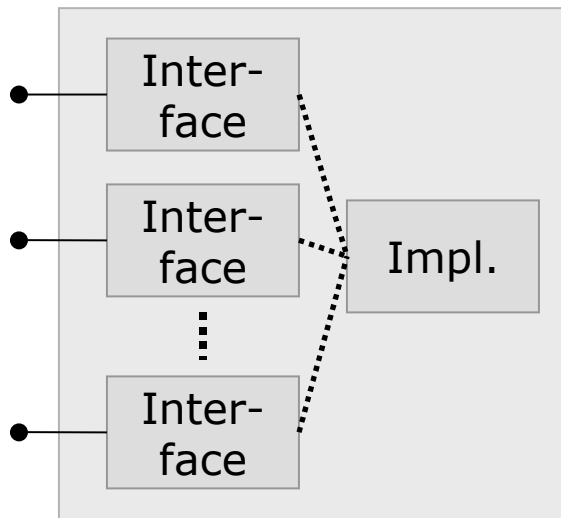
3.1 Kommunikationskonzepte

Von Prozeduren zu Objekten

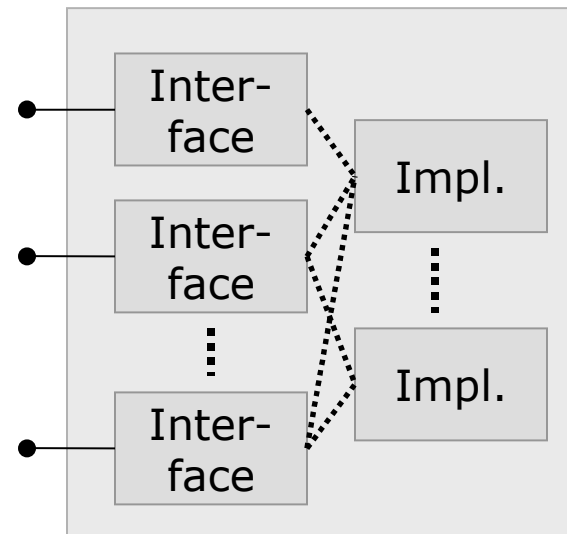
● 1 Impl. \Leftrightarrow * S (Java, CLR)

● * Impl. \Leftrightarrow * S (COM)

Java



COM



- mehrere Implementierungen derselben Schnittstelle möglich
- Implementierung kann mehr Funktionalität bereitstellen als durch die Schnittstelle definiert

3.1 Kommunikationskonzepte

Von Prozeduren zu Objekten

- CORBA: Traditioneller Objektansatz
 - Jedes Komponente (=Objekt) hat nur ein Interface
 - Mehrfachvererbung möglich
 - Erwartete Schnittstelle darf Subtyp der bereitgestellten sein
 - Zusatzeigenschaften können dynamisch herausgefunden werden
- COM: Komponente hat mehrere Schnittstellen in einer Liste
 - Schnittstellen bedienen mehrere Objekte
 - Interface unveränderbar
 - einmal veröffentlicht -- weder erweiter- noch änderbar
 - aber Schnittstellenliste kann dynamisch erweitert werden
 - einfache Schnittstellenvererbung möglich

3.1 Kommunikationskonzepte

Von Prozeduren zu Objekten

- Java: Klassen können mehrere Schnittstellen implementieren, aber stärker am Vererbungskonzept orientiert
 - Default-Implementierungen von Interfaces durch abstrakte Klassen möglich
 - Klasse kann mehrere Interfaces implementieren, aber nur von einer abstrakten Klasse erben
- Problem der Namenskollision, wenn Methoden aus unterschiedlichen Schnittstellen denselben Namen haben
 - Java: Überladen und Überschreiben
 - qualifizierte Namensgebung ist möglich
 - COM und CLR: unterschiedliche Schnittstellen haben unterschiedliche Namensräume

Namensgebung und Auffinden von Diensten

- Dienste werden über ihren Namen identifiziert
 - OMG: UUID als Standard der Open Software Foundation (DCE)
 - genügend lange Zeichenkombinationen
 - COM (Microsoft) verwendet modifizierte Version: Global Unique Identifier (GUID)
 - Namensgebung für Interfaces (IID), Gruppen von Interfaces (categories = CATID) und Klassen (CLSID)
 - CLR: Identität durch private / public-key auf Komponentenebene
 - Java: Eindeutigkeit über zusammengesetzte Pfadnamen (Anlehnung an URL)
- Über den Namen muss wenigstens folgende Funktionalität zur Laufzeit abrufbar sein:
 - Typtest der Schnittstellen
 - Introspektion der Schnittstellen
 - dynamisches Erzeugen neuer Objekte

3.2. Erste Komponentenansätze

Komponentenkonzepte - Die Anfänge

Der dokumentenzentrierte Ansatz

- Idee: Nutzer wird nicht mit vielen verschiedenen Applikationen konfrontiert, sondern mit Dokumenten, die aus mehreren Teilen bestehen können. Diese Teile können unterschiedliche Applikationen zur Darstellung benötigen, kennen diese aber selbst.
- Erste Realisierung unmittelbar auf der Ebene von integrierten Textdokumenten
 - Hypercard (Apple)
 - Word mit Visual Basic und VBX (Microsoft)

3.2. Erste Komponentenansätze

Komponentenkonzepte - Die Anfänge

Visual Basic

- Dokument besteht aus (mehreren) Formularen
- Formular kann mit Kontrolleinheit ausgestattet sein
- Kontrolleinheiten interagieren über Basic-Skripte

Flexibilität und Produktivität dieses Konzepts führten zur Herausbildung des ersten Komponentenmarkts mit Komponenten etwa zur Tabellenkalkulation oder zur Prozessautomatisierung.

OLE als Weiterentwicklung dieses Ansatzes

- Formulare \Rightarrow Container für beliebige Anwendungen
- Kontrolleinheit \Rightarrow Dokumentenserver
- Container können hierarchisch ineinander geschachtelt werden

3.2. Erste Komponentenansätze

Komponentenkonzepte - Die Anfänge

Der webzentrierte Ansatz

- Idee: Einbettung von beliebigen Objekten in HTML-Seiten
 - z.B. Java Applets, Form-Bestandteile
- Einheitliche und erweiterbare Darstellung im Browser durch Plugin-Technologie
- Schritt weg vom OLE-Containerkonzept und zurück zum (nicht hierarchischen) Formularansatz von Visual Basic

Aktuelle Entwicklungsrichtungen

- COM (Microsoft)
- CORBA (Object Management Group)
- Java (Sun und inzwischen auch IBM)
- Webservices als lose gekoppeltes Konzept

Die OMG und CORBA

- Geschichte, Zielstellungen, Entwicklungsetappen
- Architektur
 - Objekte, Servanten, Anwendungen
 - Schnittstellensprache OMG IDL
 - Dynamische Methodenaufrufe (DII)
 - Symmetrie des CORBA-Modells
- Der Object Request Broker (ORB)
- CORBA-Objekte und Objektreferenzen
- CORBA IDL und Datentypen

- Literatur: CORBA Spezifikation 3.0 (Juli 2002)
 - 1154 Seiten pdf-Dokument, siehe <http://www.omg.org>

Zur Geschichte der OMG (Object Management Group)

- **Ausgangspunkt 1989:** Wie kommuniziert man in einem verteilten OO-System über Sprach- und Plattformgrenzen hinweg?
 - selbst auf derselben Plattform lieferten C++-Compiler inkompatiblen Bytecode, verschiedene Objektmodelle in verschiedenen Programmiersprachen, Plattformunterschiede bei Socket-Kopplung
 - „Deep gaps everywhere“
- im April 1989 von 11 Firmen gegründet
- heute mit ca. 800 Mitgliedern eines der größten Konsortien der Computer-Industrie
 - vor allem Systemanbieter und Anwender objektorientierter Techniken

Zielstellung: „Standardisierung, koste es, was es wolle“, um Interoperabilität auf allen Ebenen in einem offenen Markt für „Objekte“ zu erreichen.

Zielstellungen der OMG

- Offene Interoperabilität zwischen einer Vielzahl von Sprachen, Implementierungen und Plattformen
- mehr standardisieren als „binäre“ Standards
- Flexibilität statt Binärkompatibilität
 - „teure“ Hochsprachenprotokolle
- **Nichtkommerzielle Vereinigung** zur Entwicklung von technisch exakten und in der Praxis realisierbaren Spezifikationen
- **Vereinbarung von Standards und Spezifikationen** der Infrastruktur für verteilte, objektorientierte Anwendungen
- **Aufstellung von Richtlinien** (guidelines) zur Entwicklung von Umgebungen, in denen heterogene Systemen (verschiedene Plattformen, Betriebssysteme u.ä.) zusammenarbeiten können
- Durch standardisierte, objektorientierte Softwarekonzepte die **Entstehung eines Marktes für Komponentensoftware forcieren**

Etappen der Entwicklung von CORBA

CORBA 1 (seit 1991) : **Standardisierung des ORB**

- erste Lösungen, um das Wirrwar zu entflechten
- Ansatz: Vermittlung zwischen Anfragen und Diensten durch einen Object Request Broker (ORB)
- CORBA = Common Object Request Broker Architecture
- **Meilenstein**: Schnittstellen-Definitionssprache (OMG IDL)

CORBA 2 (seit 1995 – 96) : **Interoperationsstandards zwischen ORBs**

- **Meilenstein**: Internet Inter-ORB Protokoll (IIOP)
- muss von jeder ORB-Implementierung unterstützt werden
- Für CORBA 2 existiert Vielzahl von Realisierungen verschiedener Anbieter und für verschiedene Plattformen

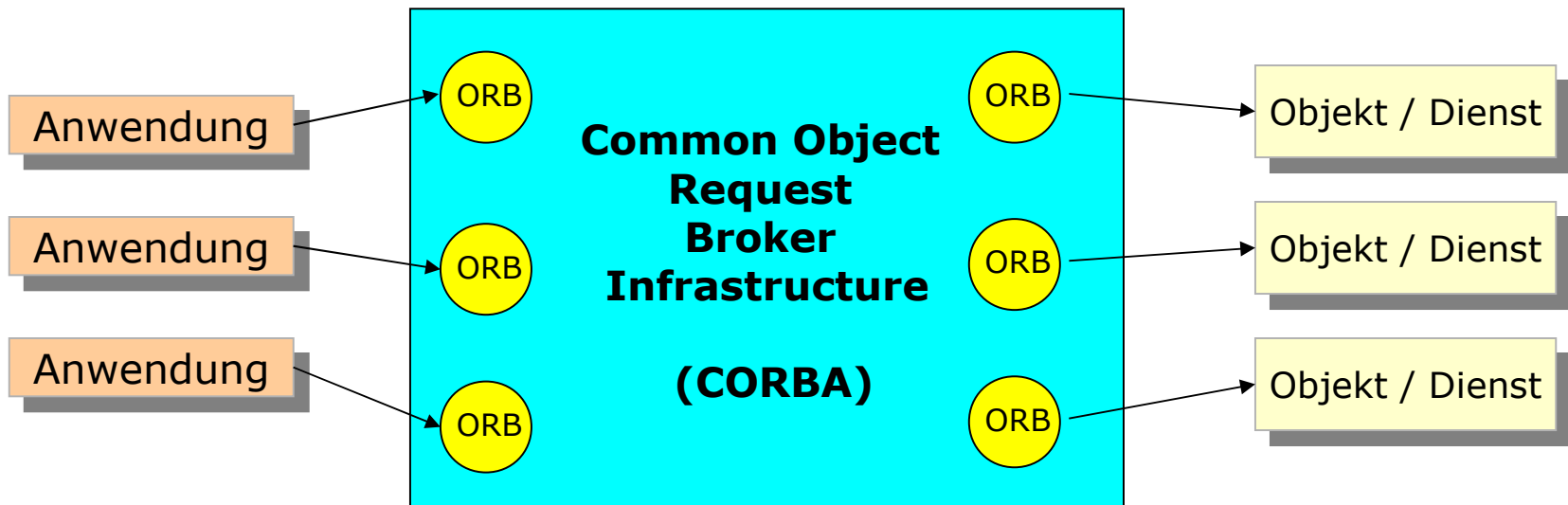
Etappen der Entwicklung von CORBA (Fortsetzung)

CORBA 3 (12/2002) : **Komponenten- und Systemintegration**

- Höhere Abstraktionsebene
- neue Sprachebenen zur Beschreibung von Komponenten-Eigenschaften
- seit 1998 in der Entwicklung, aber als Ganzes erst Ende 2002 freigegeben
 - CORBA 2.3 ... 2.6 (2001) : Freigabe verschiedener Standards, auf die man sich auf dem Weg zu CORBA 3 zwischenzeitlich geeinigt hatte
- **Meilenstein:** CORBA Komponentenmodel (CCM)
 - Version 4.0, April 2006
- aktuelle Version CORBA/IIOP 3.0.3, März 2004 (<http://www.omg.org>)
- bisher kaum Implementierungen, die CORBA 3 voll unterstützen

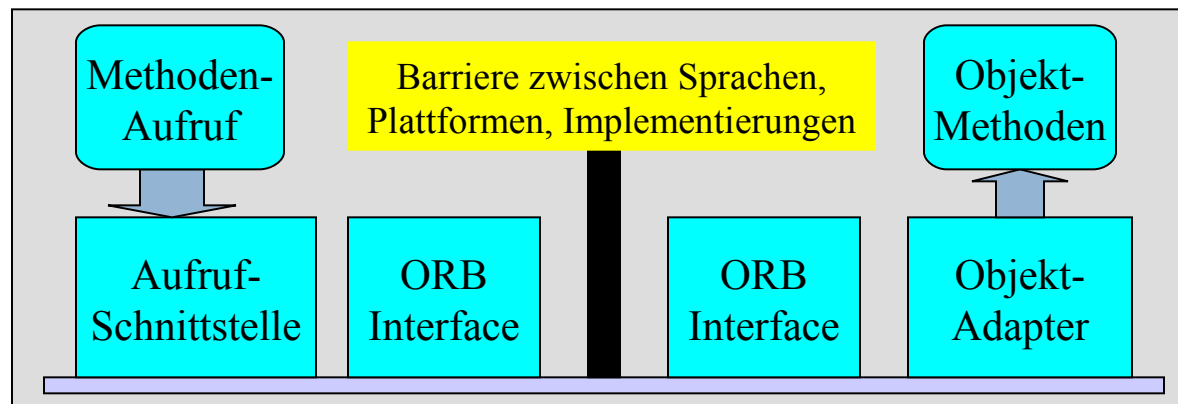
CORBA besteht im Grunde aus drei wichtigen Teilen:

- einer Menge von Aufrufschnittstellen (Invocation Interfaces)
- den Vermittlern (Object Request Brokers – ORBs) als den Schaltstellen der Kommunikation
 - mit einem spezifizierten Protokoll, dem internet inter-ORB protocol IIOP
- einer Menge von Objekt-Adaptern



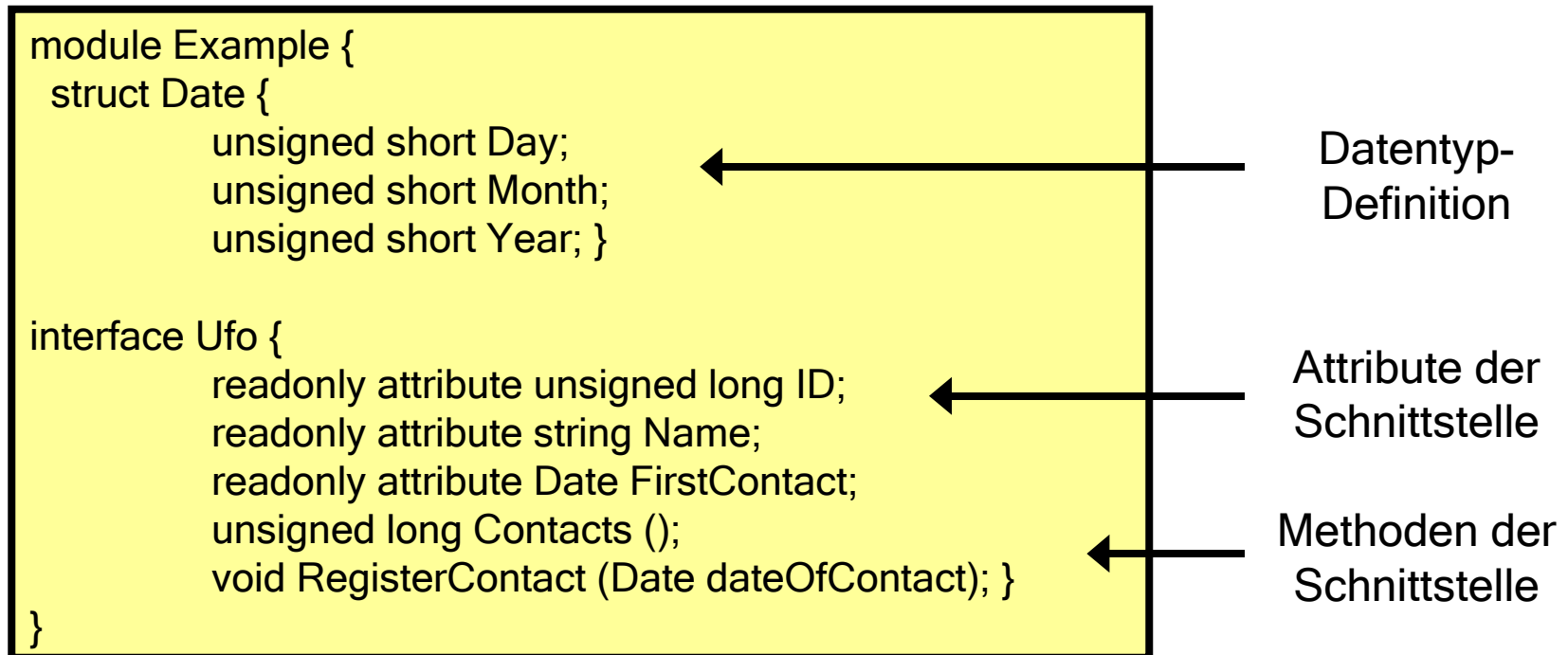
Laufzeitbindung von Methodenaufrufen

- Aufrufschnittstelle serialisiert Aufrufargumente
- ORBs suchen Zielobjekt, -methode, organisieren Transport der Argumente
- Objektadapter: dient der Aktivierung des Diensts im Objekt. Deserialisiert Argumente und ruft entsprechende Methode des Zielobjekts auf.



Wichtige Voraussetzungen

- Schnittstellen müssen in einer einheitlichen Sprache **definiert** werden (**Interface Definition Language - OMG IDL**)
 - wesentlicher Bestandteil des CORBA-Standards
 - ermöglicht generisches Serialisieren / Deserialisieren



Wichtige Voraussetzungen (Fortsetzung)

- alle Programmiersprachen, die den CORBA-Standard unterstützen, müssen **an OMG IDL gebunden** werden.
 - Mapping von Datentypen,
 - Übersetzung des OMG IDL Operationsformats in das sprachspezifische Aufruf-Format
 - Fehlerbehandlung
 - existieren Anbindungen für C, C++, SmallTalk, Cobol, Java, ...

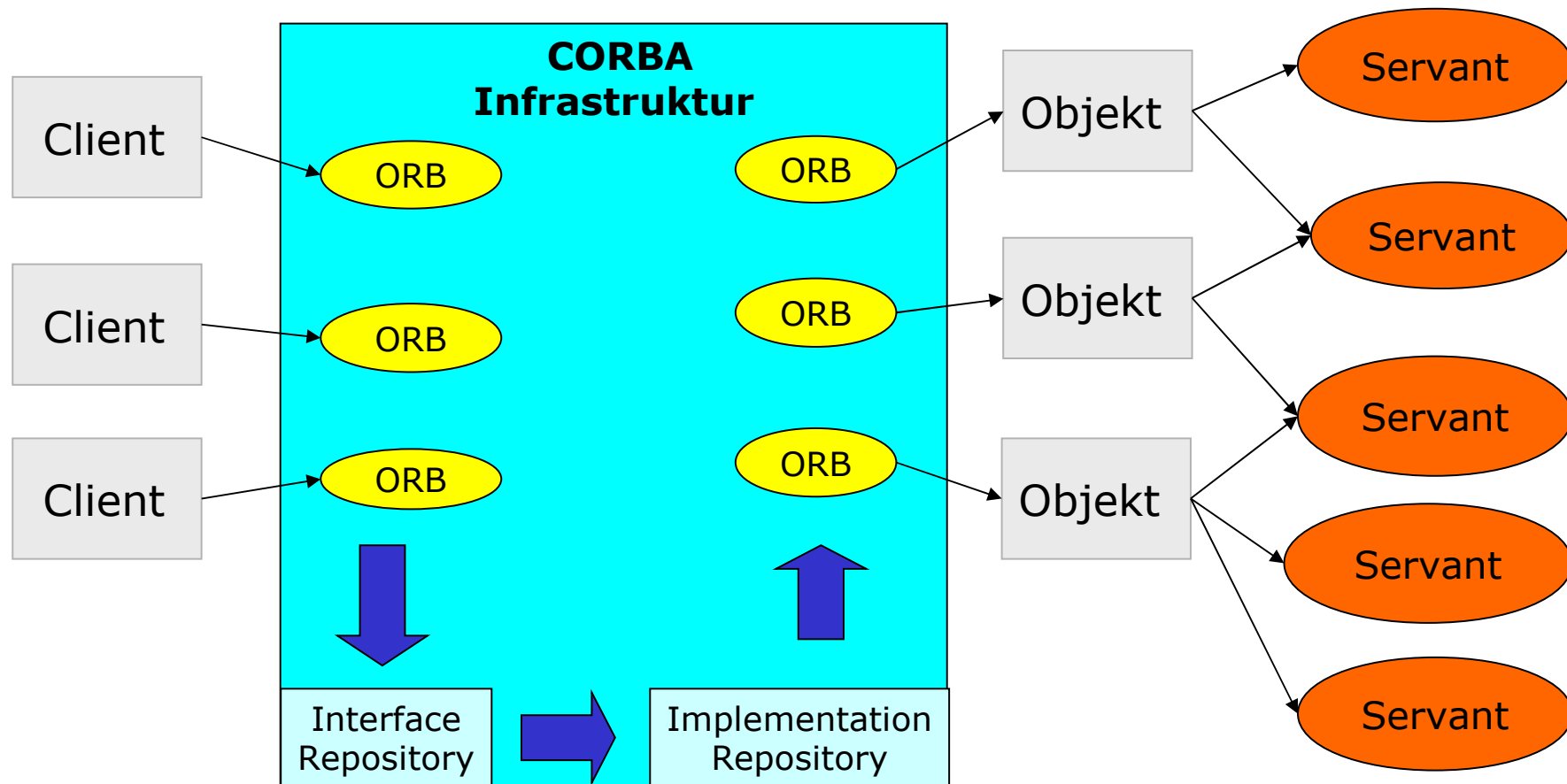
In OMG IDL beschriebene Schnittstellen werden dann

- mit einem **OMG IDL Compiler** übersetzt
- im **Schnittstellen-Repository** abgelegt
- durch **Methoden der ORB-Schnittstelle** angesprochen

Wichtige Voraussetzungen (Fortsetzung)

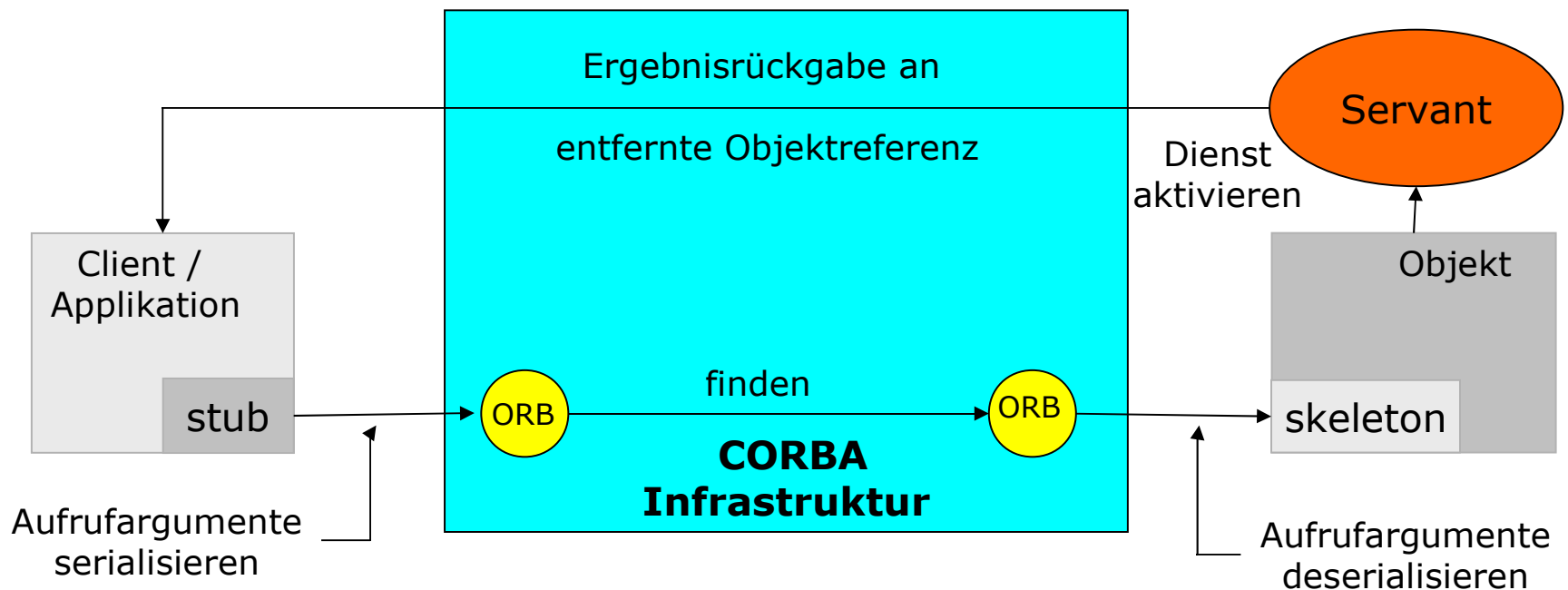
- Programmfragmente stellen **Implementierungen** für solche Schnittstellen (oder Teile davon) bereit
 - heißen **Objekt-Servanten** (object servant)
 - werden im **Implementations Repository** registriert
 - Servanten werden bei Bedarf geladen und/oder gestartet
 - Objektadapter teilen dem ORB mit, welche Objekte von welchen Servanten bedient werden.
 - Eine Serverumgebung (typ. Prozess) kann mehrere Servanten bedienen.
 - *: * - Beziehung zwischen Objekten und Servanten
 - Objektbegriff hat damit leicht anderen Fokus als in der Vorlesung

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.

