Internetanwendungstechnik

Common Object Request Broker Architecture (CORBA)

Gero Mühl

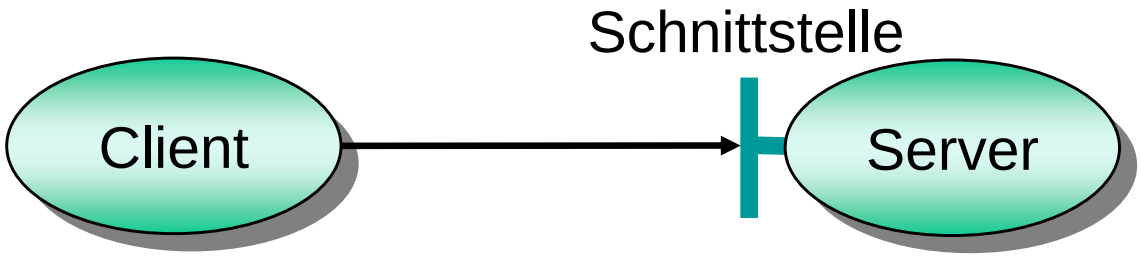
Technische Universität Berlin
Fakultät IV – Elektrotechnik und Informatik
Kommunikations- und Betriebssysteme (KBS)
Einsteinufer 17, Sekr. EN6, 10587 Berlin

Object Management Group (OMG)

- Gründung des Konsortiums im April 1989
- Ziele
  - Interoperabilität
  - Anwendungsintegration
  - Portabilität

  } in heterogenen Umgebungen auf der Basis des Objektmodells
- Entwicklungsprozess: Request for Proposal (RFP)
- Definition von Schnittstellen (und Architektur)
  - Keine eigenen Implementierungen
- Liaisons mit anderen Standardisierungsaktivitäten

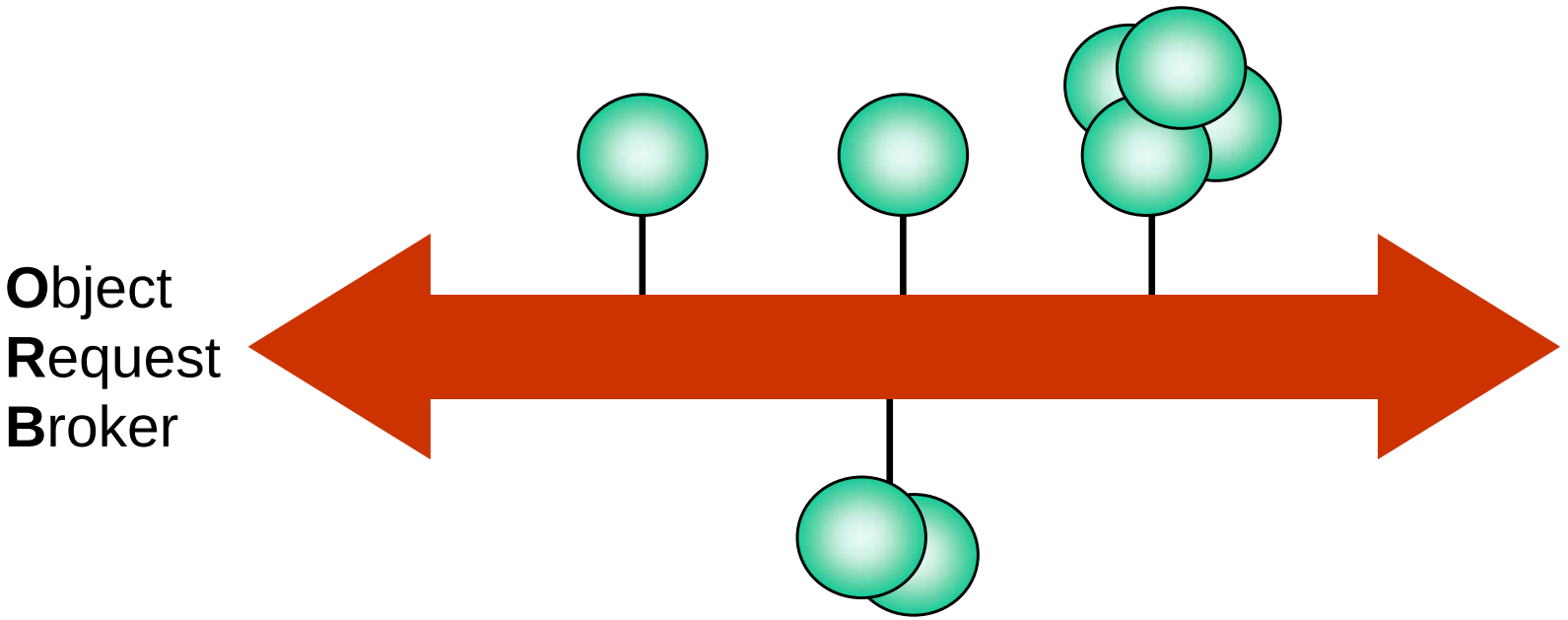
Objektmodell



- Objekt
  - Instanz einer Klasse
  - Hat einen Zustand
  - Zeigt ein Verhalten
  - Besitzt "Namen" und Lebenszeit
- Klasse
  - Schablone für Objekte
  - Verfeinerung durch Vererbungsmechanismus
  - Polymorphe Substitution
- Schnittstelle (Interface) eines Objektes
  - Methoden und Attribute
  - Kapselung der Implementierung

Object Request Broker (ORB)

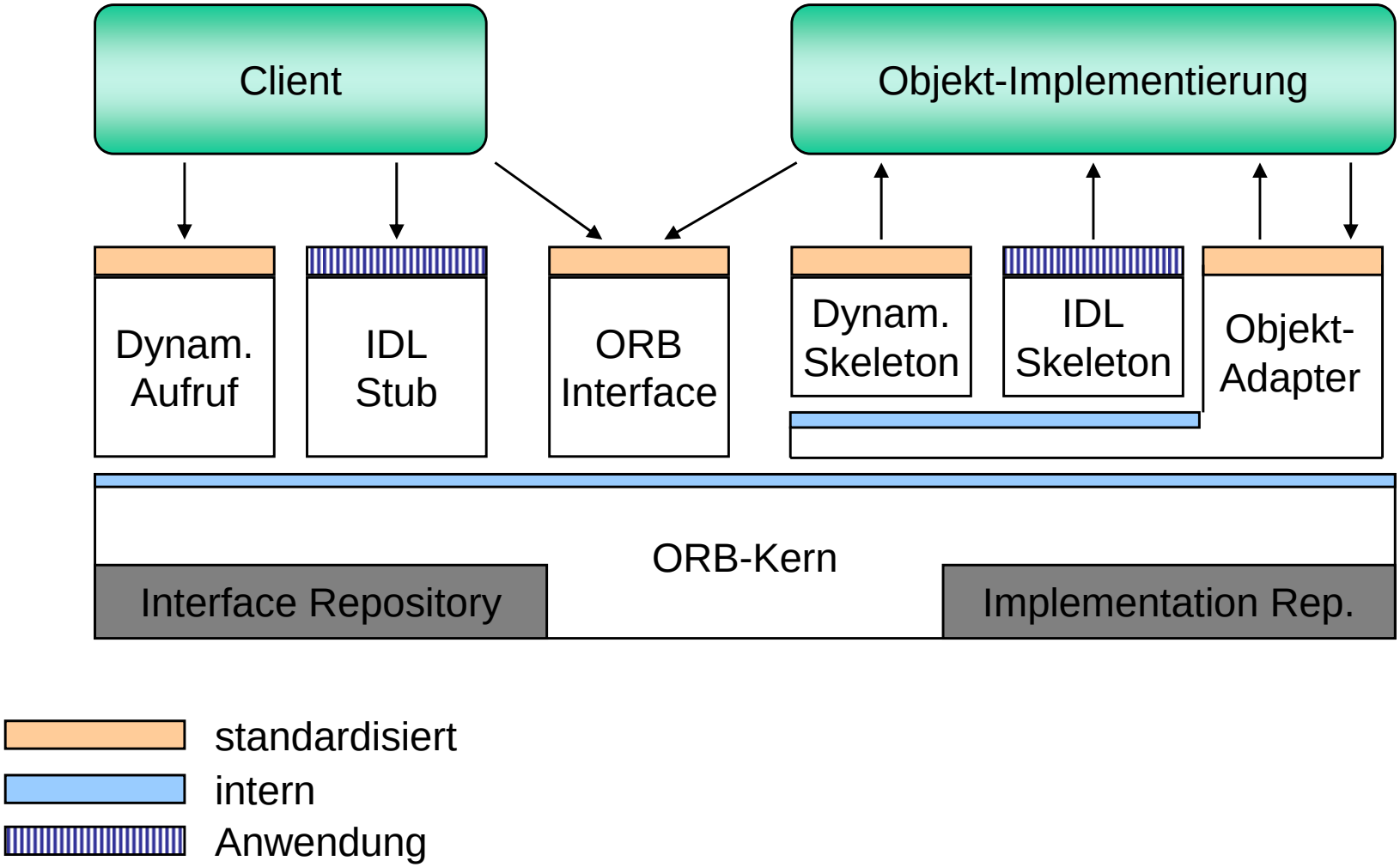
- "Software Bus" für verteilte Objektsysteme
  - Wie ein Hardware-Bus verschiedene H/W-Komponenten verbindet, so verbindet der ORB verschiedene S/W-Komponenten



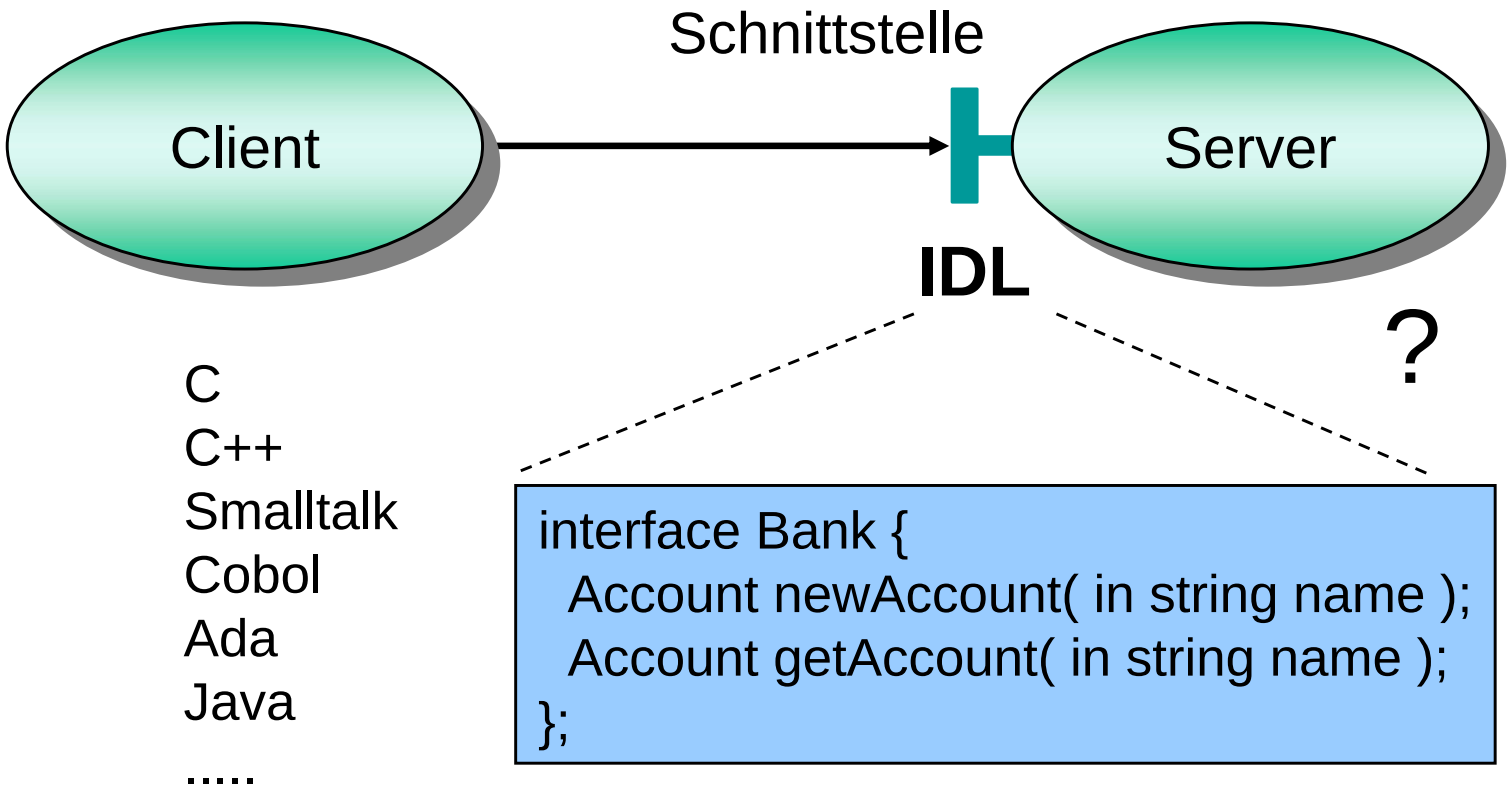
Object Request Broker (ORB)

- Einbettung von Objekt-Implementierungen ("Server-Objekte")
- Vergabe von Objektreferenzen
- Entgegennehmen von Aufrufen des Clients
- Transport der Aufrufe zum Server
- Ggf. Aktivierung eines Server-Objektes
- Übergabe des Aufrufs an das Server-Objekt
- Entgegennehmen von Ergebnissen und Transport / Rückgabe an den Client
- Unterstützung von Sicherheits- und Abrechnungsfunktionen

CORBA

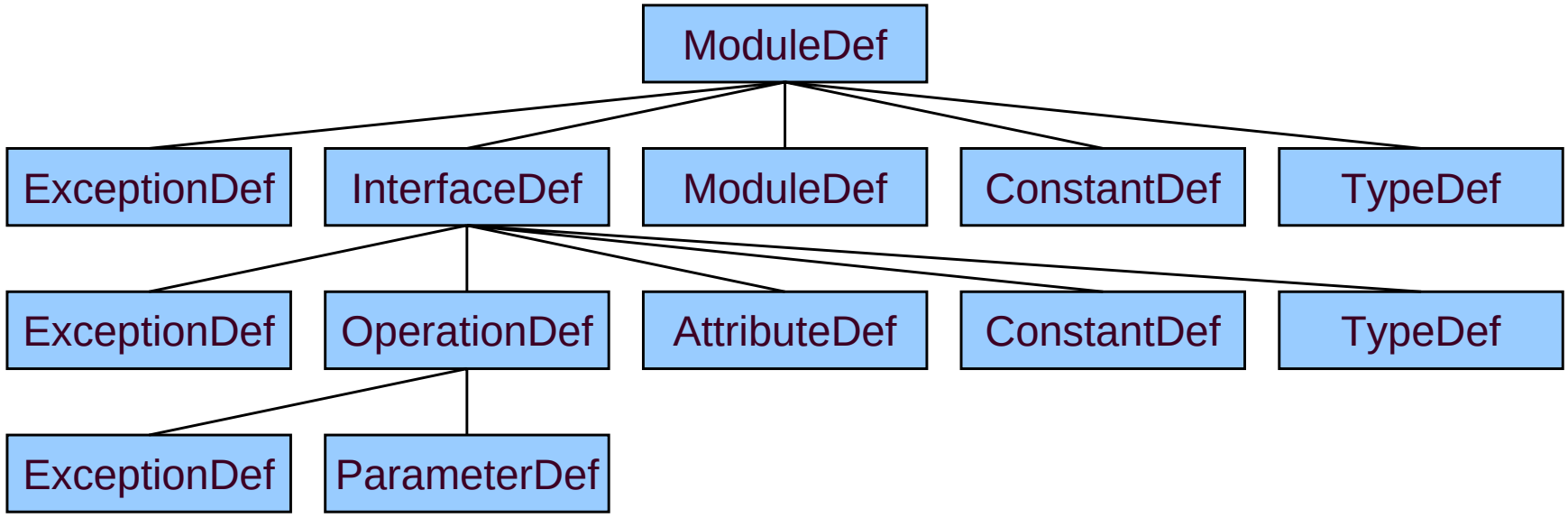

Client
Objekt-Implementierung
Dynam. Aufruf
IDL Stub
ORB Interface
Dynam. Skeleton
IDL Skeleton
Objekt-Adapter
ORB-Kern
Interface Repository
Implementation Rep.
standardisiert
intern
Anwendung

CORBA IDL


Schnittstelle
Client
Server
IDL
?
C
C++
Smalltalk
Cobol
Ada
Java
.....
interface Bank {
Account newAccount(in string name);
Account getAccount(in string name);
};

CORBA IDL

- Rein deklarativ
- Strenge Typbindung
- Scoping :: der Deklarationen



IDL – Modul

- Gruppiert zusammengehörige Schnittstellen
- Definiert eigenen Namensraum

```
module BankModul {
            // Reihenfolge ist beliebig
            // zusätzliche Konstanten, Typen, etc.
      const ...
      typedef ...
            // mögliche Exceptions
      exception kaputt { ... };
            // einige Schnittstellen
      interface Bank { ... };
      interface Account { ... };
            // auch weitere Module sind möglich
      module SubBankModul { ... };
};
```

IDL – Vererbung

- Eine Schnittstelle kann von beliebig vielen Schnittstellen erben
- Alle Definitionen der Ober-Schnittstelle verfügbar
- Elemente können neu definiert werden (nicht Attribute)
  - Originale durch Scoping weiter erreichbar (<interface>::<name>)

```
interface Bank { ... };
interface Telefon { ... };
interface TeleBank : Bank { ... };
interface TelefonBank : TeleBank, Telefon { ... }
```

IDL – Datentypen

basic

- `long`
- `unsigned long`
- `short`
- `unsigned short`
- `float`
- `double`
- `char`
- `boolean`
- `octet`
- `any`

constructed

- `struct`
- `union`
- `enumeration`
- `sequence`
- `string`
- `wstring`
- `template`
- `array`

IDL – Sprachbindung

CORBA IDL-Typen	C++ Mapping
`long, short`	`long, short`
`float, double`	`float, double`
`enum`	`enum`
`char`	`char`
`boolean`	`bool`
`octet`	`unsigned char`
`array`	array
`struct`	`struct`
`union`	class
`string`	`char *`
`wstring`	`wchar_t *  // 16 bit chars`
`sequence`	class
object reference	Pointer oder Objekt

Objektreferenzen

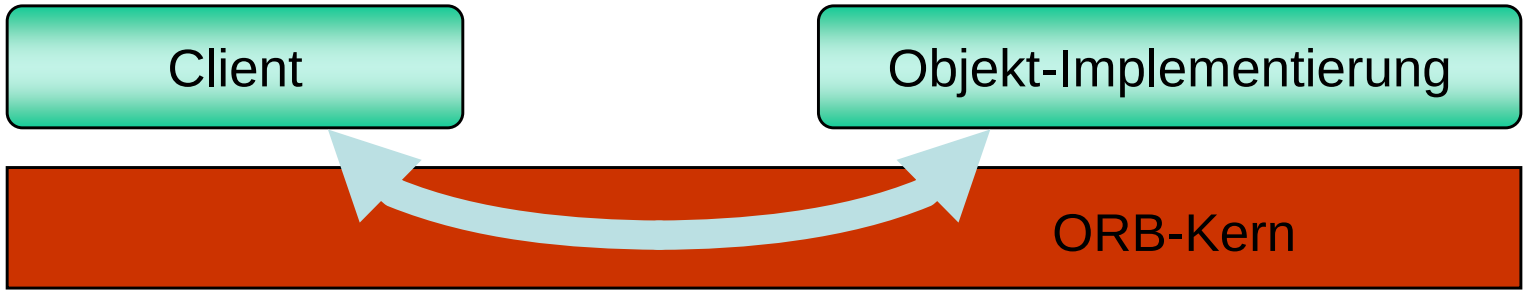
- Identifizieren eindeutig eine Objekt-Implementierung
- ORB bildet eine vom Client präsentierte Objektreferenz auf das zugehörige Objekt ab
- Referenzen können als Parameter in IDL übergeben werden
- Für jede Objektreferenz kann man sich vom ORB die Interface-Spezifikation geben lassen
- ORB-Interface bietet zur Umwandlung
  - object_to_string( ) // wandelt Referenz in einen String um
  - string_to_object( ) // wandelt einen String in eine Referenz um
- Auch die „String-Version“ kann ausgetauscht werden oder über Namensdienst ermittelt werden
- Wie findet ein Programm aber die Objektreferenz des Namensdienstes?

Objektreferenzen

- Bootstrapping
  - resolve_initial_references( )
- Funktion muss von jedem ORB angeboten werden
- Gibt initiale Objektreferenzen zurück für
  - Interface Repository
  - Naming Service
  - Trading Service
  - Notification Service
  - …
- CORBA-Standard lässt zukünftige Erweiterungen zu

ORB-Kern

- Transportiert Aufrufe und Ergebnisse zwischen Client und Objekt-Implementierungen hin und her
- Transparenz
  - Objekt-Lokation
  - Art der Objekt-Implementierung (Sprache, Plattform, ...)
  - Aktivierungszustand der Objekt-Implementierung
  - Verwendete Kommunikationsprotokolle



ORB-Kern Implementierungsvarianten

- Dezentraler ORB
  - ORB wird bei Client und Server in Form von Bibliotheksroutinen oder als Teil des Betriebssystems realisiert
- Zentraler ORB
  - ORB wird als separater Prozess realisiert
- Lokaler ORB (Spezialform)
  - Objektimplementierung als lokale Bibliothek vorhanden und keine Stubs notwendig

Statische Aufrufe

- Verwenden das Static Invocation Interface (SII)
- Schnittstellen stehen zur Compile-Zeit fest
- IDL-Compiler erzeugt aus den Schnittstellen die Stubs
- Typüberprüfung schon zur Compile-Zeit möglich
- Stub ist „Vertreter“ der lokal nicht vorhandenen Prozedur
- Kaum Unterschiede zum lokalen Aufruf

Dynamische Aufrufe

- Verwenden das Dynamic Invocation Interface (DII)
- DII ist unabhängig von der konkreten Schnittstelle des aufgerufenen Objekts
- Vorteil: Schnittstelle muss nicht zur Compile-Zeit vorliegen, da Aufruf zur Laufzeit zusammengestellt wird
- Allerdings deutlich mehr Code notwendig
- Folgende Angaben sind für einen Aufruf notwendig
  - Objekt, Operation, Argumente (Anzahl, Typ, Modus, Wert)
- Problem: Typüberprüfung erst zur Laufzeit möglich
- Normalerweise deutlich ineffizienter, als statische Aufrufe

Ablauf eines dynamischen Aufrufs

1. Objektreferenz beschaffen (z.B. über Name Service)
2. Schnittstelle bestimmen (z.B. im Interface Repository)
3. Request Pseudo Object erzeugen und füllen
   (= Behälter für die zum Aufruf gehörenden Informationen)
4. Request Pseudo Object an ORB übergeben
   (= Starten des Aufrufs)
5. Auf Ergebnis warten (optional)
6. Request Pseudo Object löschen oder wieder verwenden

Aufrufarten mit dem SII und dem DII

- Synchroner Aufruf
  - Client blockiert bis Antwort zurück (oder Fehler)
  - Aufruf- und Rückgabeparameter möglich
  - Möglich für statische und dynamische Aufrufe

- Oneway Aufruf (= "*fire and forget*“)
  - Beispiel: `oneway void sayGoodBye(...);`
  - Kein Rückgabewert oder Rückgabeparameter
  - Keine Exceptions bei Fehlern
  - Möglich für statische und dynamische Aufrufe
  - Semantik hängt vom ORB ab
    (können z.B. auch blockierend sein!)

Aufrufarten mit dem SII und dem DII

- Asynchroner Aufruf (Deferred Synchronous)
  - Client blockiert nicht
  - Aufruf- und Rückgabeparameter möglich
  - Nur möglich für dynamische Aufrufe
  - Zustand des Request-Objekts kann abgefragt werden
  - Bedarf für asynchrone Aufrufe mittels des SIIs → AMI

CORBA Messaging (ab CORBA 2.3)

- Ermöglicht asynchrone Aufrufe mittels des SIIs
  → Asynchronous Method Invocation (AMI)
  - Vorher waren asynchrone Aufrufe nur mit dem DII möglich!

- AMI unterstützt zwei Modelle
  - Callback Model
    - Client gibt dem Aufruf eine Objektreferenz für Antwort mit
    - ORB benutzt diese Objektreferenz, um Antwort abzuliefern
    - Ergebnis wird nicht im Kontext des Aufrufenden verarbeitet
  - Polling Model
    - Aufruf gibt sofort ein Pseudo-Objekt zurück
    - Client kann an diesem Objekt pollen bzw. warten
    - Ergebnis wird im Kontext des Aufrufenden verarbeitet

CORBA

Interoperabilität

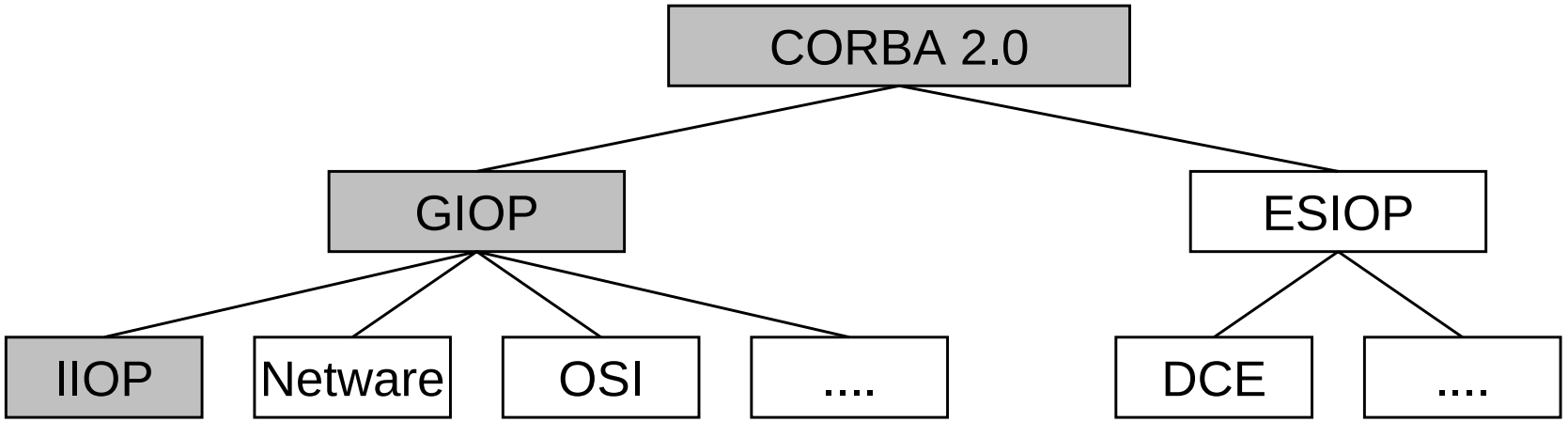
Interoperabilität – Zwischen Wem?

- Zwischen Objekten
  - ORB ermöglicht Methodenaufrufe
- Zwischen Programmiersprachen
  - IDL entkoppelt
- Zwischen ORBs
  - Allgemeines standardisiertes Inter-ORB-Protokoll (GIOP) (ab CORBA 2.0)
- Zwischen Verteilungsplattformen
  - Übergang in Gateways, z.B. von CORBA nach Microsoft DCOM

Interoperabilität – Zwischen ORBs

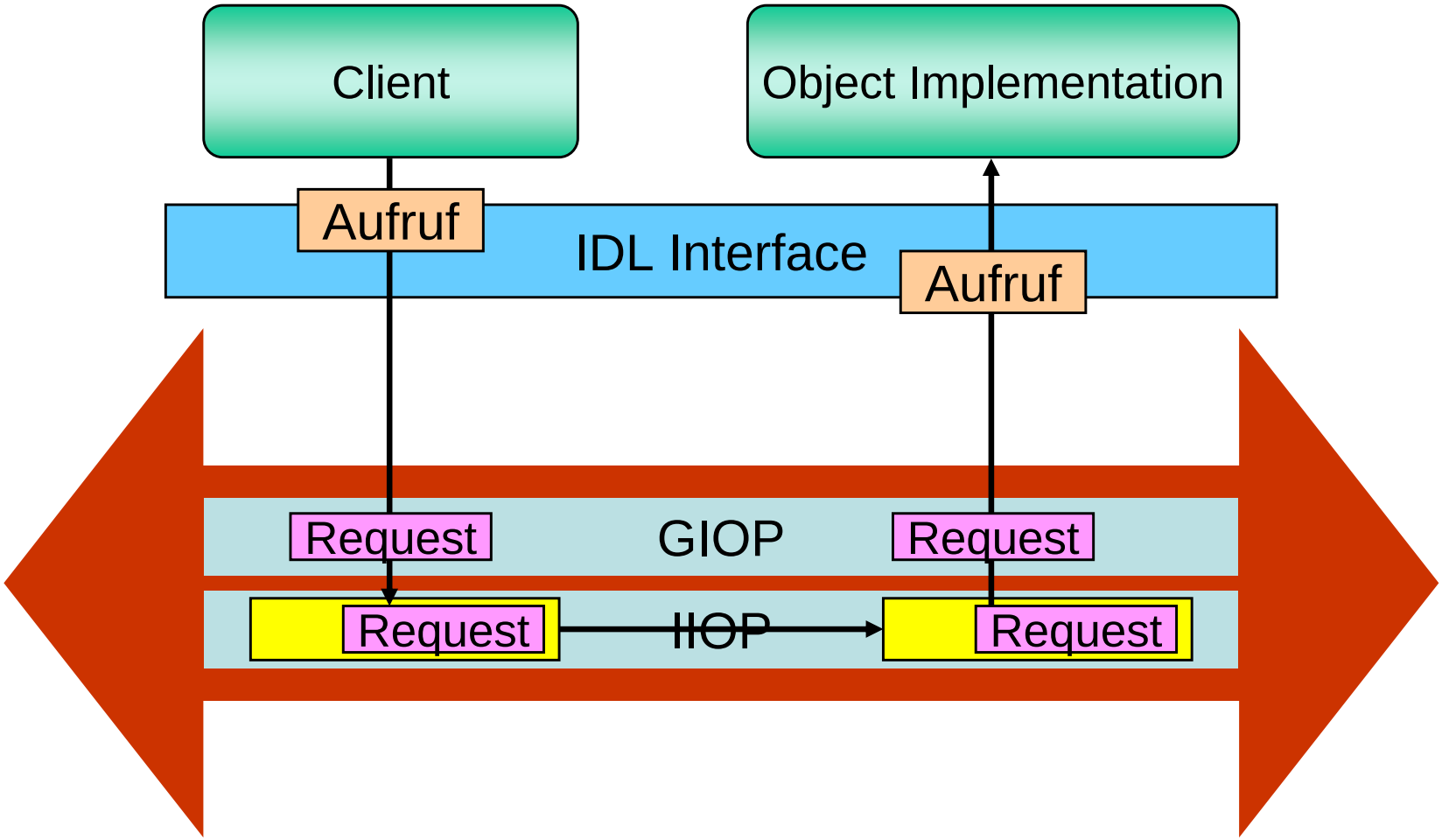
- General Inter-ORB Protocol (GIOP)
  - Spezifiziert Transfersyntax und Nachrichtenformate
    - Common Data Representation (CDR)
  - Für beliebiges verbindungsorientiertes Transportsystem
  - Wird auf konkretes Transportprotokoll abgebildet
- Internet Inter-ORB Protocol (IIOP)
  - GIOP auf TCP/IP
  - Ab CORBA 2.0 muss ein ORB GIOP und IIOP anbieten
  - Häufig auch als Intra-ORB-Protokoll benutzt
- Environment Specific Inter-ORB Protocol (ESIOP)
  - Z.B. für OSF DCE

Protokolle für die Interoperabilität



für CORBA 2.0 vorgeschrieben

ORB mit GIOP und IIOP



Internet Inter-ORB Protocol (IIOP)

- GIOP auf TCP/IP
  - Abbildung der GIOP-Protokollelemente auf TCP-Nachrichten
- Spezifiziert auch ein entsprechendes Protokollprofil
  - IIOP-Version
  - Name des Servers (string)
  - TCP-Portnummer des Servers (unsigned short)
  - Objekt-ID (sequence<octet>)

Interoperable Object References (IOR)

- Gemeinsames Zwischenformat für Objektreferenzen

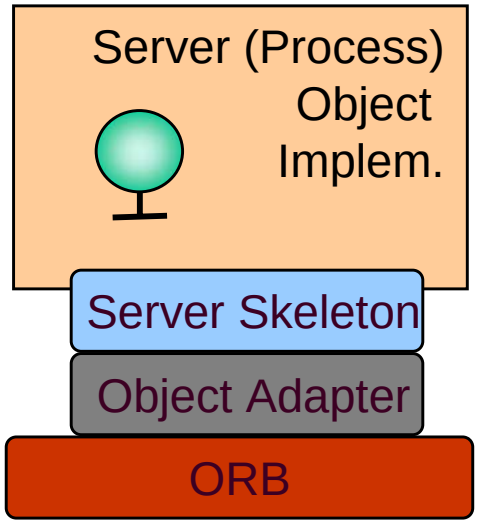
```
// IDL
module IOP {
   typedef unsigned long ProfileId;

   struct TaggedProfile {
         ProfileId tag;
         sequence<octet> profile_data;
   };

   struct IOR {
         string type_id;
         sequence<TaggedProfile> profiles;
   };
   ....
};
```

Interoperable Object Reference (IOR)

- Beispiel für IOR eines „Bank-Objekts“ erzeugt mit iordump() von MICO

Repo ID:	IDL:Bank:1.0
IIOP Profile	
Version:	1.0
Address:	inet:141.2.2.29:4653
Key:	2f 34 33 32 34 2f 39 36 37 32 32 38 37 34 33 2f 5f 30

Object Adapter (OA)



- Sitzt zwischen ORB und Skeletons
- Zusätzliche Indirektion → Flexibilität
- Spezialisierte OAs ermöglichen verschiedene Arten von Objektimplementierungen (*Servants*)
  - Objekte werden in einer Datenbank gespeichert und mit Datenbankanfragen manipuliert
  - Objekte im gleichen Adressraum
  - C++ Objekte
- Ermöglichen einen schlanken ORB-Kern

Object Adapter (OA)

- Aufgaben
  - Registrierung von Objektimplementierungen
  - Generierung von Objektreferenzen
  - Aktivierung der Server-Prozesse
  - Aktivierung und Deaktivierung der Objekte
  - Demultiplexen und Weitergabe der Aufrufe (Upcalls) mittels der Skeletons an den passenden Servant
  - Rückgabe der Ergebnisse
- Ursprünglich: Basic Object Adapter (BOA)
- Ab CORBA 2.2: Portable Object Adapter (POA)

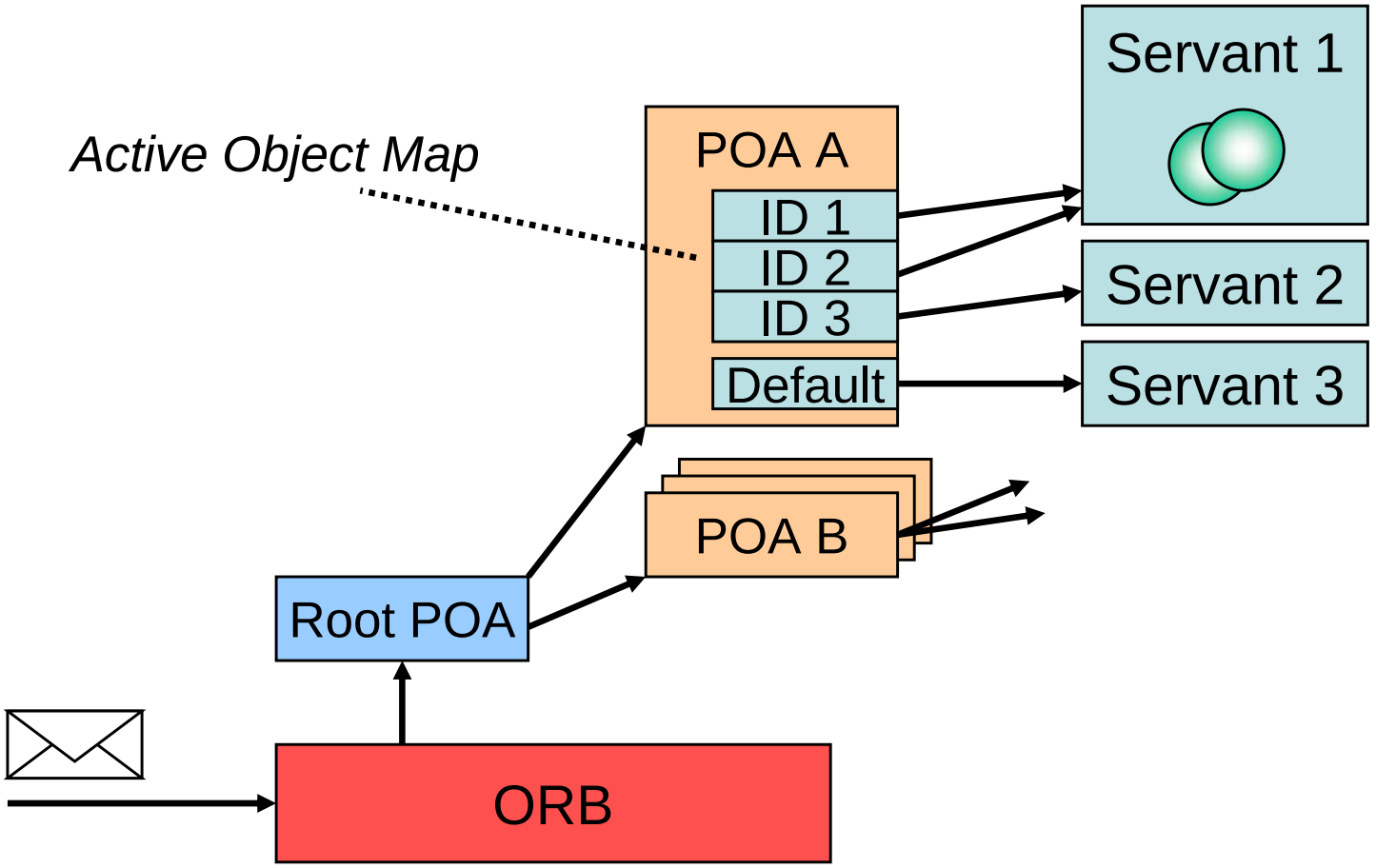
Basic Object Adapter (BOA)

- Standardisierter Minimal-Adapter
- Ermöglicht verschiedene Server-Aktivierungsmodi
  - shared — Server bedient beliebige Anzahl von Objektinstanzen (wird am meisten benutzt)
  - unshared — Server bedient nur genau eine Objektinstanz
  - persistent — Server wird manuell gestartet (z.B. durch ein Skript bei jedem Neustart)
  - per method — für jeden Aufruf wird ein eigener Server gestartet
- Schwächen des BOA
  - unterspezifiziert, ungenau spezifiziert
  - fehlende Funktionalität
  - unklare Trennung von „Objekt“, „Server“, „Implementierung“
  - führt zu nicht-portablen Objektimplementierungen

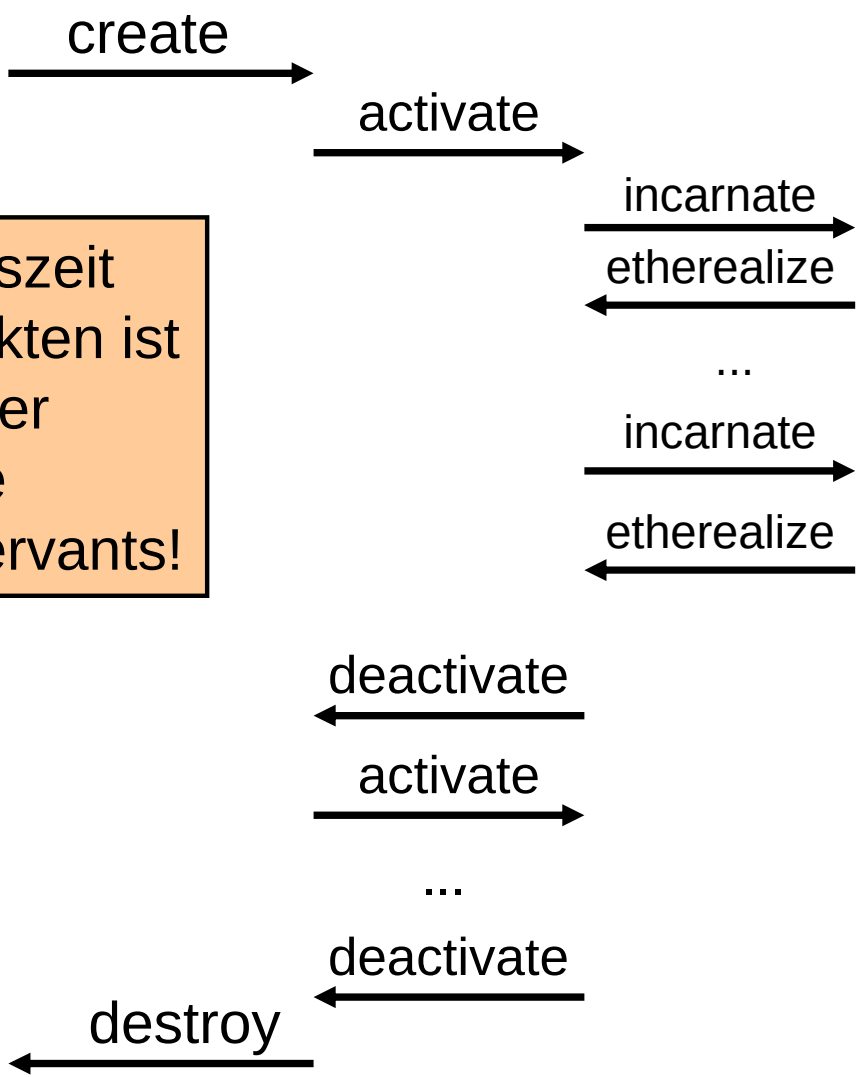
Portable Object Adapter (POA) (ab Version 2.2)

- Verbesserungen im Vergleich zum BOA
  - Eindeutige Spezifikation
  - Klare Trennung von „Objekt“, „Server“, „Implementierung“
  - Führt zu portablen Objektimplementierungen
  - Unterscheidung zwischen persistenten und transienten Objekten
    - persistent = Objekt lebt länger als erzeugender Serverprozess
    - transient = Objekt lebt nur so lang wie erzeugender Prozess
  - Erzeugung *virtueller* Objektreferenzen (z.B. für Dateien)
  - Transparente Aktivierung von Objekten im Bedarfsfall
  - Hierarchie von mehreren verschiedenen POAs möglich
    → „Root POA“

Portable Object Adapter (POA)



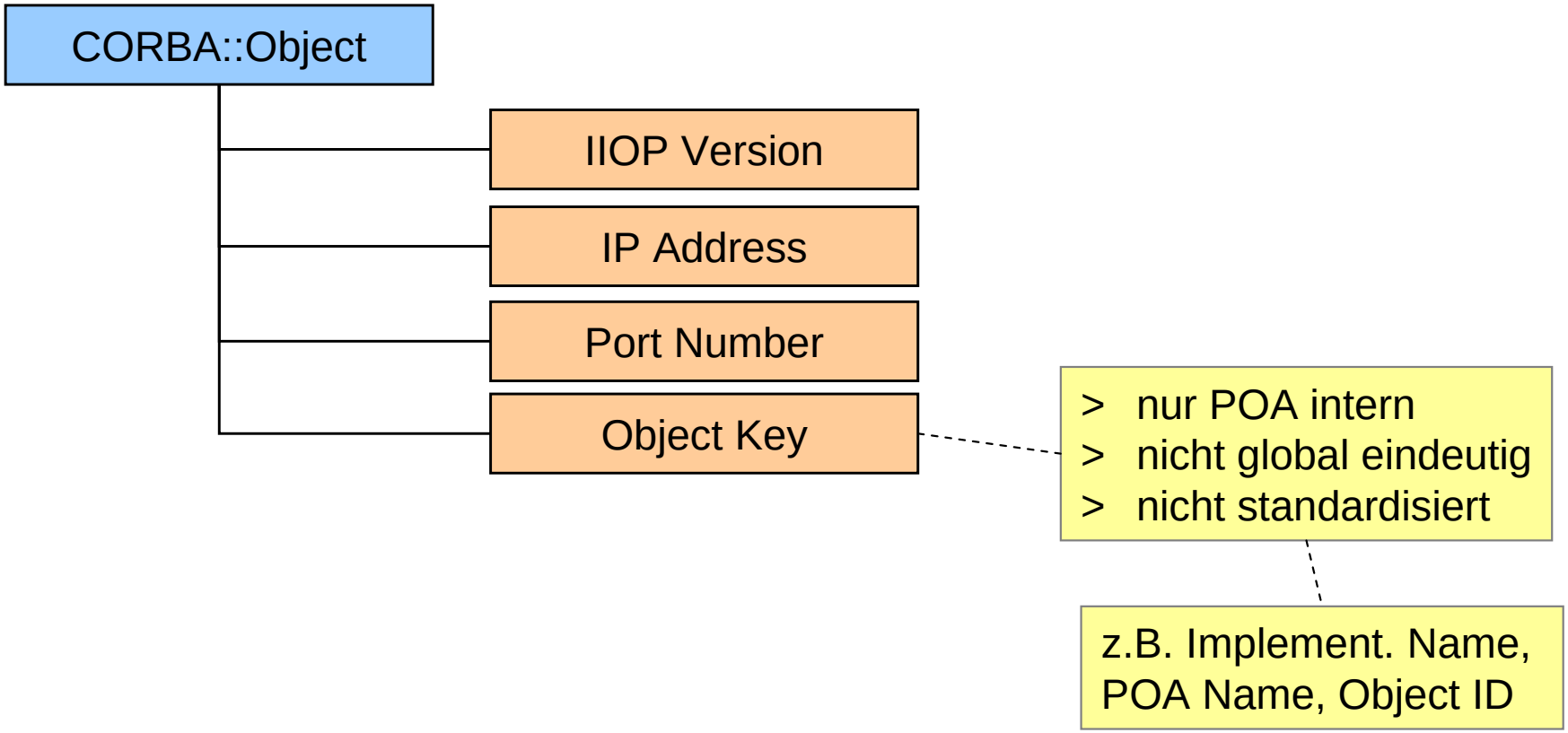
Lebenszyklus von CORBA Objekten



Merke: Die Lebenszeit von CORBA-Objekten ist unabhängig von der Lebenszeit der sie inkarnierenden Servants!

(POA)-Objektreferenz (für IIOP)

> Enthält alle Informationen zur Identifikation des zuständigen Servants



Interface Repository (IR)

- IR = Archiv für Schnittstellen-Beschreibungen
- Einträge sind (persistente) Objekte, die IDL-Definitionen enthalten
- IR ist selbst ein CORBA-Objekt (repository object)
- Kann zur Laufzeit benutzt werden, um Schnittstellen-Informationen zu erhalten, zum Beispiel für dynamische Aufrufe
- CORBA sagt nicht, wie Informationen ins IR gelangen
  - Meist durch IDL Compiler abgelegt
- Identifizierung von Schnittstellen-Definitionen
  - Durch (im IR) eindeutigen Namen
  - Durch global eindeutige RepositoryID
    - Generiert vom Eintragenden
    - Eindeutigkeit über IRs hinweg

Implementation Repository

- Verwendung meist mit persistenten Objekten
  - Server werden im Implementation Repository registriert
  - Läuft der benötigte Server nicht, wenn ein Request eintrifft, so wird er gestartet
- CORBA Standard macht keine Vorgaben zum Interface oder zum Inhalt → Verwendung ist ORB-spezifisch
- Enthält Angaben zu z.B.
  - Executable-Dateien
  - Zugriffsrechte
  - Aktivierungskommando
  - Aktivierungsparameter
  - …

Objects by Value (OBV) (ab CORBA 2.3)

- Vor OBV konnten Objektreferenzen, aber keine Objekte als Parameter übergeben werden
  - In bestimmten Fällen ist das Verschicken des ganzen Objekts sinnvoller (= object by value)
  - Beispiele: Graph, Liste ... mit speziellen Operationen
- Objekt als Wert-Parameter
  - Legt Kopie des Objektzustands beim Empfänger an
  - Spezielles Pseudo-Objekt "valuetype"
  - Eigene Identität, keine automatische Rückwirkung auf Original
- Aber was ist mit dem Verhalten (= dem Code) des Objekts?
  - Code muss vorhanden sein oder dynamisch geladen werden
  - Für Java und Smalltalk "einfach"
  - Für C++ mit dynamisch ladbaren Bibliotheken (DLL) (?)
  - Nicht weiter spezifiziert

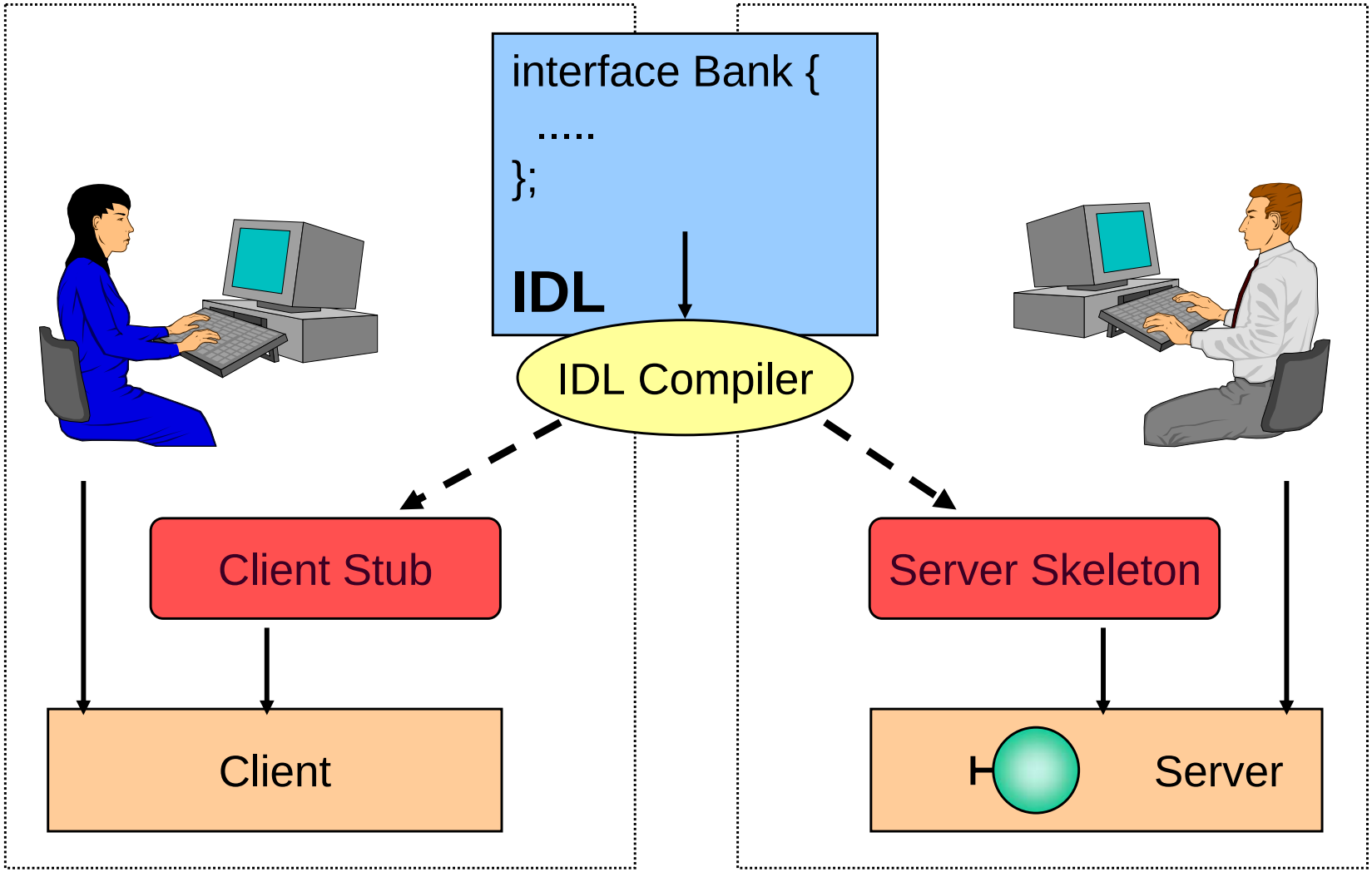
Produkte

- Kommerzielle Produkte (eine willkürliche Auswahl)
  - Orbix / IONA
  - VisiBroker / Borland
  - JavaIDL / SUN
  - Componentbroker / IBM
  - Web Logic Enterprise / BEA
  - …
  - http://www.cs.wustl.edu/~schmidt/corba-products.html
- Frei verfügbare Implementierungen (Open Source)
  - MICO C++ initiiert an Uni Frankfurt http://www.mico.org
  - JacORB Java initiiert an FU Berlin http://www.jacorb.org
  - Orbacus C++, Java Fa. IONA http://www.iona.com

CORBA

Entwicklung verteilter Anwendungen

Anwendungsentwicklung mit CORBA

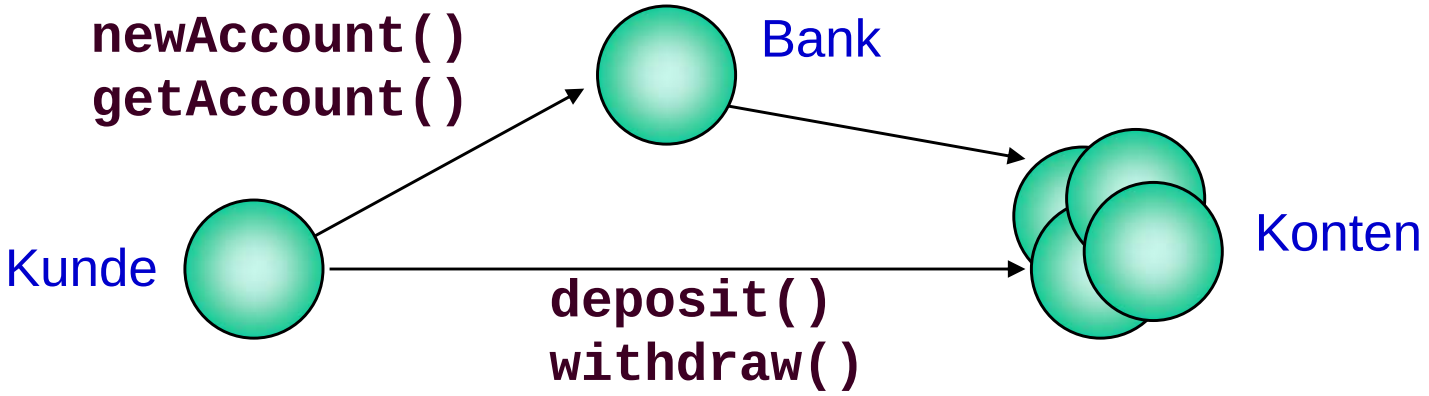


Ablauf

1. Definition der Schnittstellen in IDL
2. Mittels IDL-Compiler Stubs und Skeletons generieren
3. Implementierung des Klienten
4. Implementierung des Servants
   (d.h., der Objektimplementierung)
5. Implementierung des Servers
6. Registrierung des Servers beim ORB
7. Methodenaufrufe des Klienten auf dem Objekt

Beispiel – Bank-Szenario

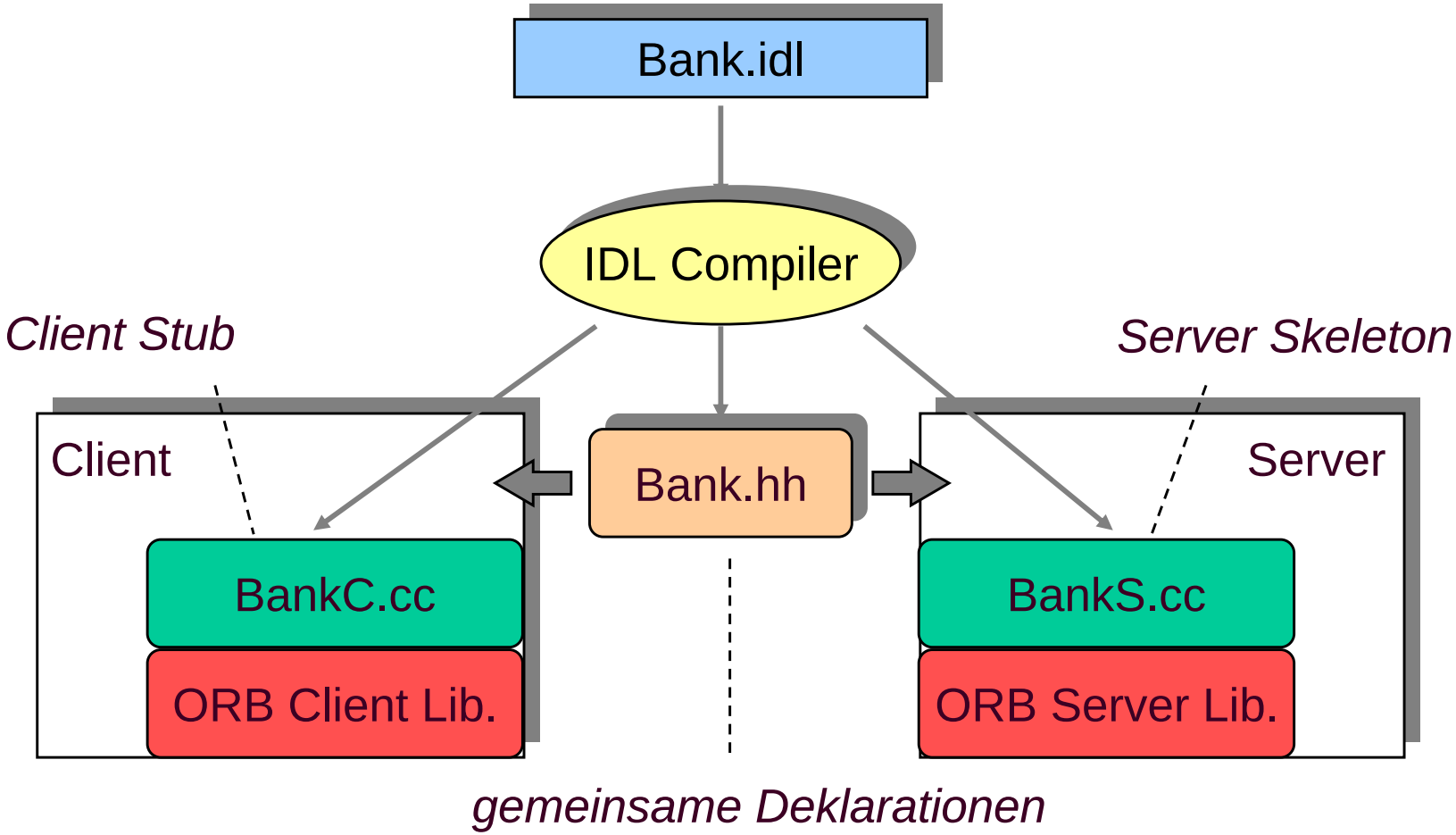
```
interface Bank {
  Account newAccount(in string name);
  Account getAccount(in string name);
};
```




```
interface Account {
  readonly attribute float balance;
  void deposit( in float amount );
  void withdraw( in float amount );
};
```

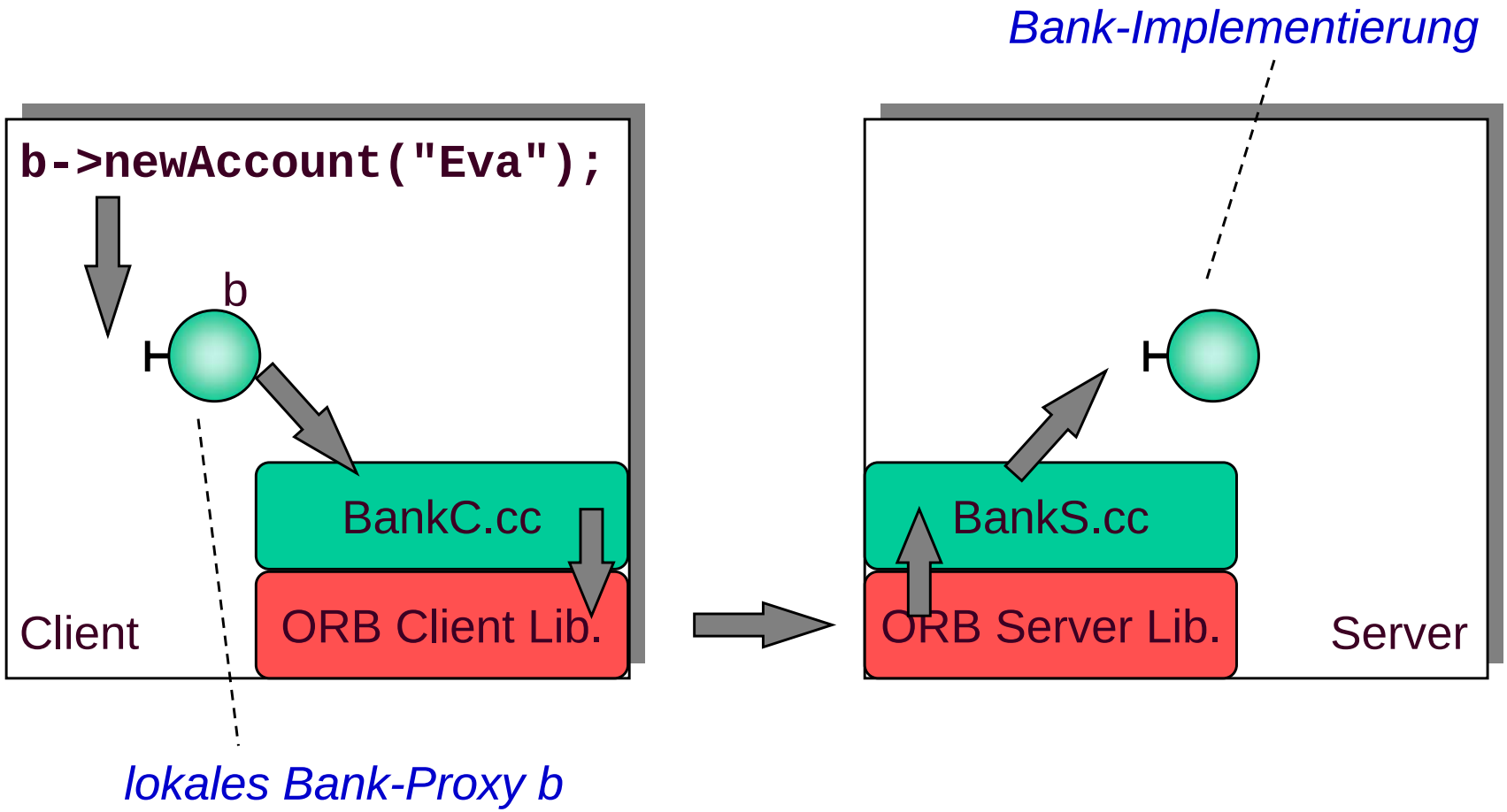
Beispiel – Bank Szenario

- IDL Compiler nach C++



Beispiel – Bank Szenario

- IDL Compiler nach C++



Beispiel – Bank Szenario

Client Code

```
#include "Bank.hh"
#include <iostream.h>
int main (int argc, char *argv[]) {
   // ORB initialisieren
   CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);
   // anbinden an das Bank-Objekt, IOR in argv
   CORBA::Object_var obj = orb->string_to_object(argv[1]);
   // Spezialisierung der Referenz per Downcast
   Bank_var b = Bank::_narrow(obj);
   // neues Konto anlegen
   Account_var a = b->newAccount("Eva");
   // Einzahlung durchführen und Kontostand ausgeben
   a->deposit(29.40);
   cout << "Kontostand: " << a->balance() << endl;
};
```

Beispiel – Bank Szenario

```
// Bank-Implementierung erbt ORB-Funktionalität
class Bank_impl : public virtual POA_Bank {
   Account_impl* m_accs[ ];       // zur Kontoverwaltung
   public:                        // wie in IDL vorgegeben
   Account_ptr newAccount( const char * );
   Account_ptr getAccount( const char * );
};

// dto. für Account-Implementierung
class Account_impl : public virtual POA_Account {
   protected float _balance;
   public:                        // wie in IDL vorgegeben
   void deposit( float amount );
   void withdraw( float amount);
   float balance( );
};
```

Servant Code

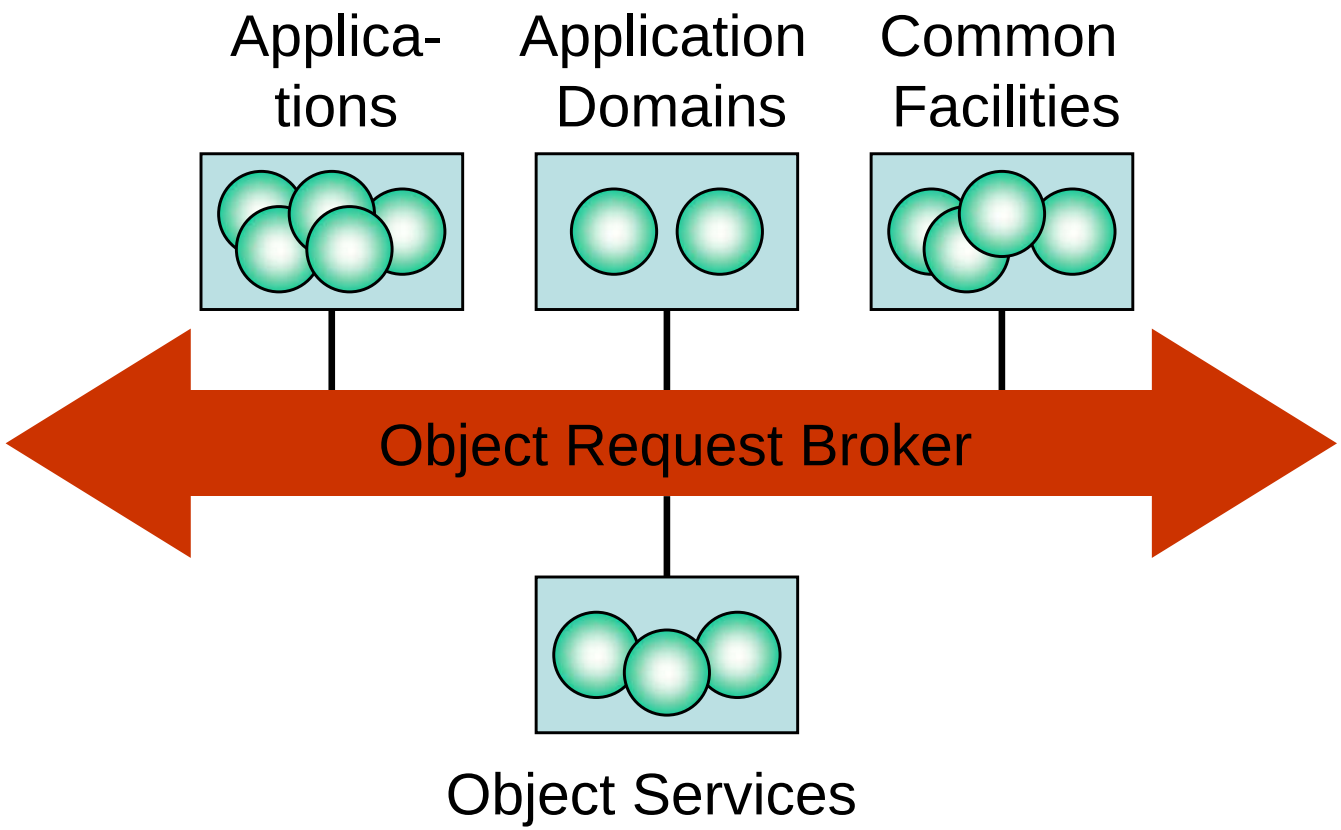
Beispiel – Bank Szenario

Server Code

```
#include "Bank.hh"
int main (int argc, char *argv[]) {
   // ORB initialisieren
   CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);
   // POA initialisieren
   CORBA::Object_var poaobj =
     orb->resolve_initial_references(“RootPOA“);
   PortableServer::POA_var poa =
     PortableServer::POA::narrow( poaobj );
   PortableServer::POAManager_var mgr = poa->the_POAManager();
   // Bank-Objekt anlegen und Objektreferenz erzeugen
   PortableServer::Servant bank_servant = new Bank_impl;
   CORBA::Object_var the_bank = bank_servant->_this();
   CORBA::String_var ior = orb->object_to_string(the_bank);
   mgr->activate();
   orb->run(); // Event loop des ORBs starten
};
```

CORBA

Object Management Architecture (OMA)



Object Services

- Basisdienste für verteilte, objektorientierte Anwendungen
- Spezifikation der Schnittstellen und der grundlegenden Funktionsprinzipien für folgende Services
  - Naming
  - Event
  - Security
  - Transactions
  - Trading
  - Lifecycle
  - Time
  - Licensing
  - Properties
  - Relationships
  - Notification
  - Persistence
  - Concurrency Control
  - Externalization

Common Facilities

- „horizontale“ anwendungsnahe Unterstützung
- Beispiele
  - System Management
  - UML (Unified Modeling Language)
  - Meta Object Facility (MOF)
  - Model Driven Architecture (MDA)

Application Domains

- „vertikale“ anwendungsnahe Unterstützung
- oft Bezug auf konkrete Anwendungsdomänen (Branchen)
- Beispiele
  - allgemeine Business Objects
  - Telekommunikation
  - Finanzindustrie
  - Medizin
  - Produktionsdaten

Application Objects

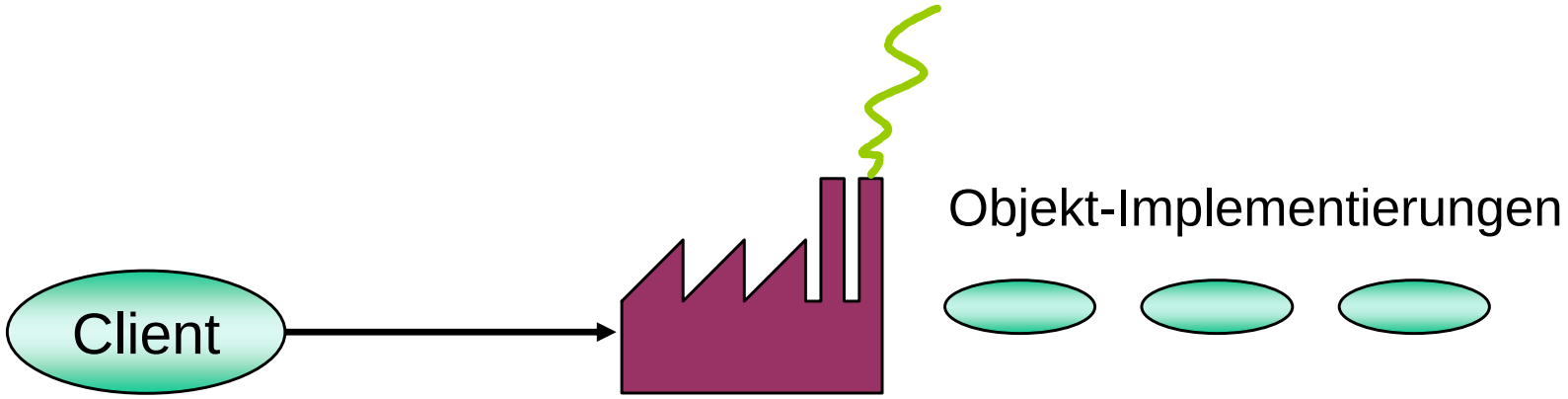
- keine Standardisierung von Anwendungsschnittstellen
- falls sich bestimmte Anwendungsschnittstellen eindeutig herauskristallisieren, wären auch solche Standards denkbar
- OMG betont die Wichtigkeit der Integration von "Altlasten"
  → Legacy Applications
  - Objektmodell bietet Kapselung und Kompatibilität *(zumindest konzeptionell)*
  - ORBs sind interoperabel
  - Interoperabilität von ORBs und „Nicht-ORBs“

Lifecycle Service

- Objekte erzeugen, löschen, kopieren, migrieren
- einige Objekte werden nicht durch den Lifecycle Service erzeugt, sondern z.B.
  - beim Systemstart oder
  - durch Scripts
- Anwendungen möchten aber evtl. auch neue CORBA-Objekte erzeugen, z.B.
  - kunden-spezifische Objekte
  - temporäre Objekte für einen Auftrag, so lange er bearbeitet wird
- Object Factory für neue Objekte

Object Factories

- Object Factories sind auch CORBA-Objekte
- erzeugt Objekte eines bestimmten Typs mit spezifischen Eigenschaften
  - allokiert Ressourcen für das Objekt
  - erzeugt eine entsprechende Objektinstanz
  - meldet das Objekt beim POA an
  - gibt Objektreferenz an den Client zurück
  - signalisiert dem POA, dass das Objekt nun aktiviert werden kann



Naming Service

- bildet "Name" auf "Objektreferenz" ab
- einfache, reine Abbildungsfunktion
  - ohne Attribute etc. (keine gelbe Seiten!)
  - wer mehr will, benutzt Trading Service
- Realisierung
  - kann mit beliebiger Technik realisiert werden
  - Name Server aus verschiedenen Domänen arbeiten föderativ zusammen
- Bootstrapping
  - Client kann sich zu Beginn die Objektreferenz des Name Service geben lassen

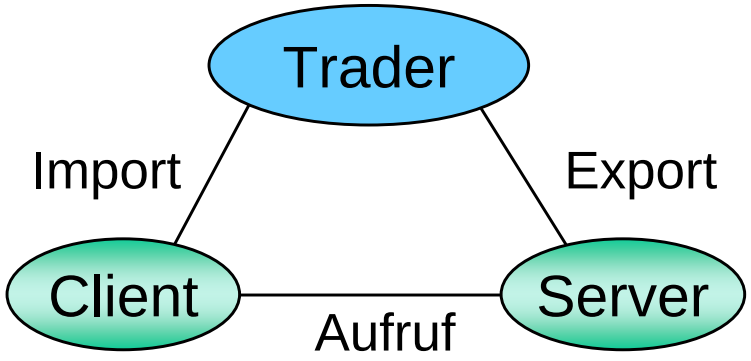
Naming Service

- keine Festlegung der Namenssyntax, sondern nur

```
struct NameComponent {
    string id;
    string kind;
};
typedef sequence <NameComponent> Name;
```

- Operationen
  - `void bind( in Name n, in Object obj );`
  - `Object resolve( in Name n );`
  - `void bind_context( in Name n,`
    `in NamingContext nc );`

Trading Service



- gelbe Seiten-Funktion mit attribut-basierter Abfrage
  → *„Gib mir einen Drucker mit Postscript und A4“*
- Standardisierung des Trading
  - ISO Open Distributed Processing (ODP)
  - OMG Trading ist weitgehend konform zum ISO Standard
- für offene, dynamisch erweiterbare Objektsysteme

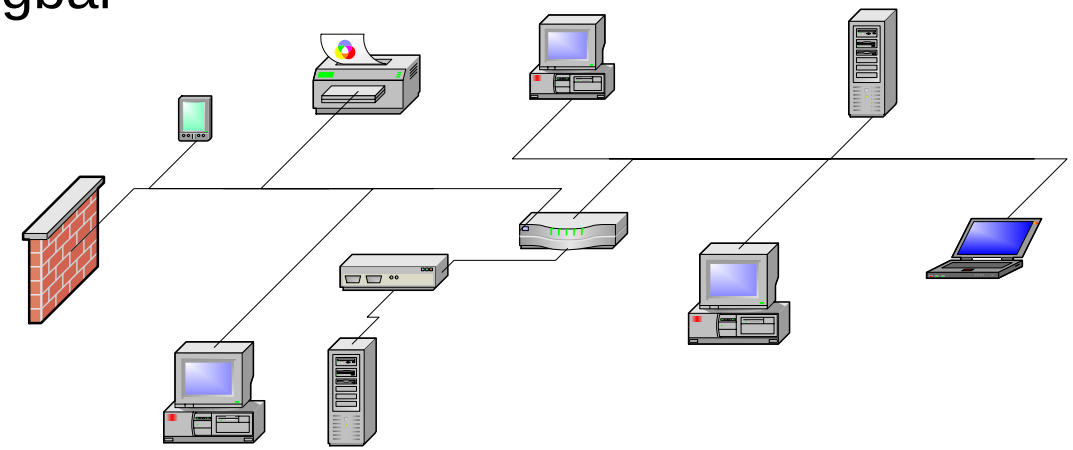
Event Service

- Normaler CORBA-vermittelter Aufruf ist synchron
  - Direkte zeitliche Kopplung zwischen Client und Server
- Asynchrone Aufrufe oft sinnvoller
  - Zeitliche Entkopplung von Aufrufer und Aufgerufenem
  - Möglich mit DII oder mit SII + AMI
  - Allerdings immer noch nur 1-zu-1 Kommunikation
- Häufig ist asynchrone *n*-zu-*m* Kommunikation sinnvoller und flexibler als 1-zu-1 Kommunikation
  - Möglich mit dem Event Service
  - Produzenten von Events veröffentlichen Events
  - Konsumenten von Event subskribieren sich für Events
  - Entkopplung durch Event Channel → Kein direkter Kontakt zwischen Produzenten und Konsumenten

Events und Systemmanagement

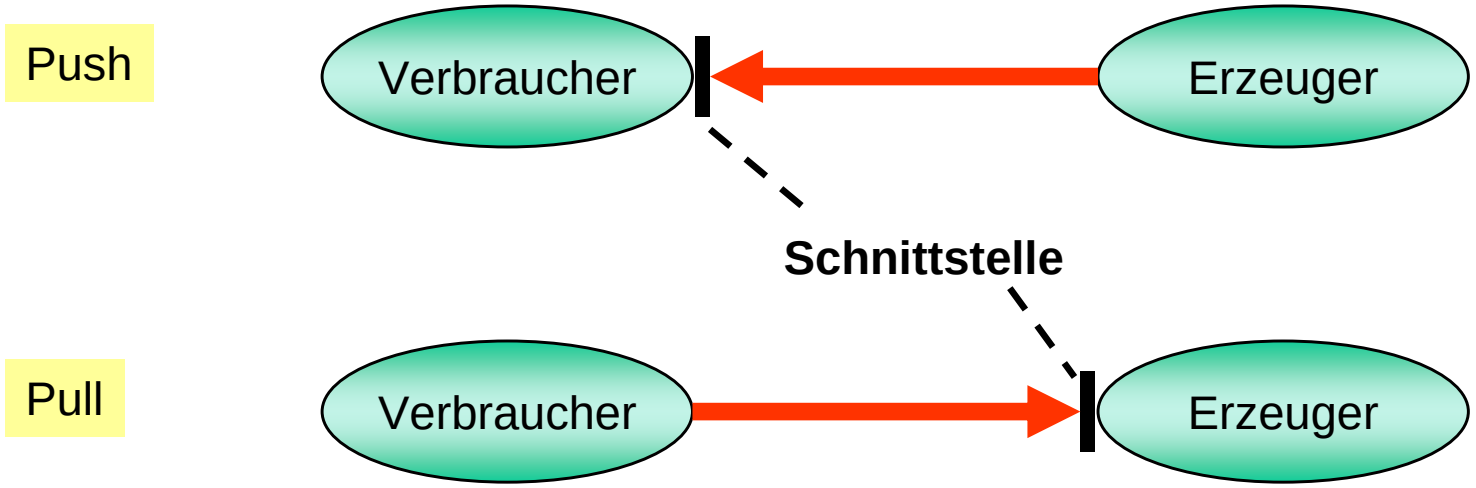
- Rechner eines verteilten Systems sollen überwacht werden
  → Monitoring

- Beispiele für Events
  - Ein Prozess wurde gestartet oder beendet
  - Platte zu 90% voll
  - Quelldatei verändert
  - Softwareupdates verfügbar
  - Ergebnis liegt vor



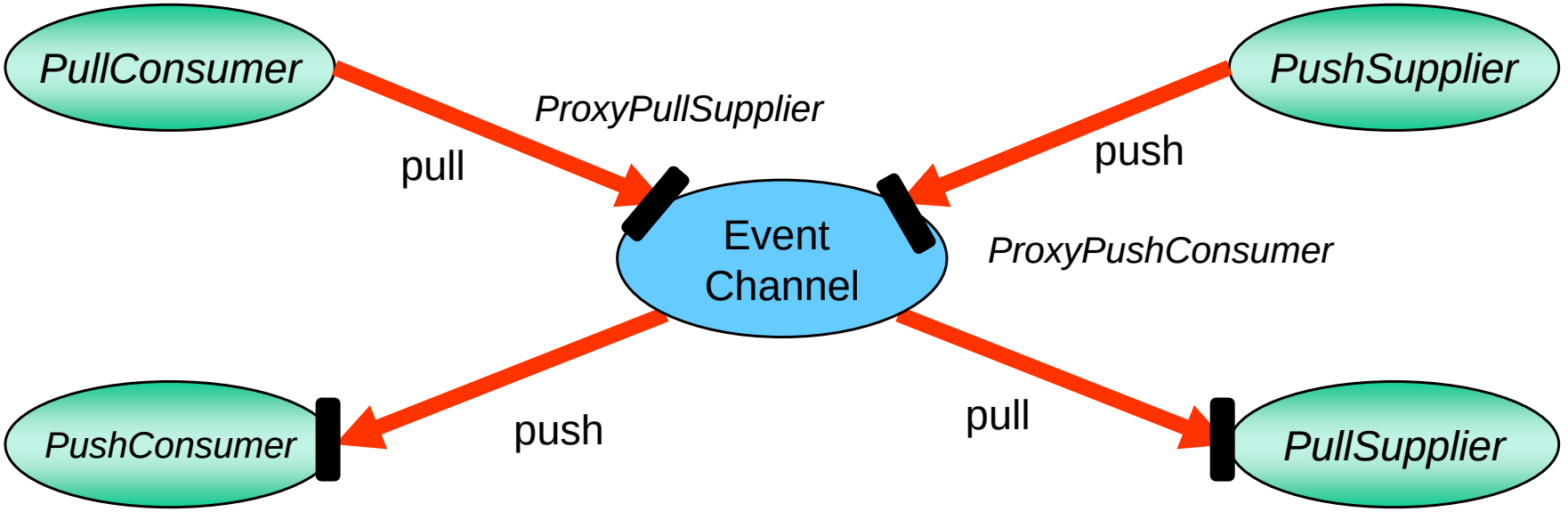
Verarbeitungsmodelle

- Überträgt Ereignismeldungen beliebigen Inhalts vom Erzeuger zu Interessenten
  - Entkoppelt Ereignis-Erzeuger und Ereignis-Verbraucher
  - Ermöglicht verschiedene Interaktionsmuster
- Zwei grundsätzliche Verarbeitungsmodelle
  - PUSH von Ereignissen: Initiative beim Erzeuger
  - PULL von Ereignissen: Initiative beim Verbraucher

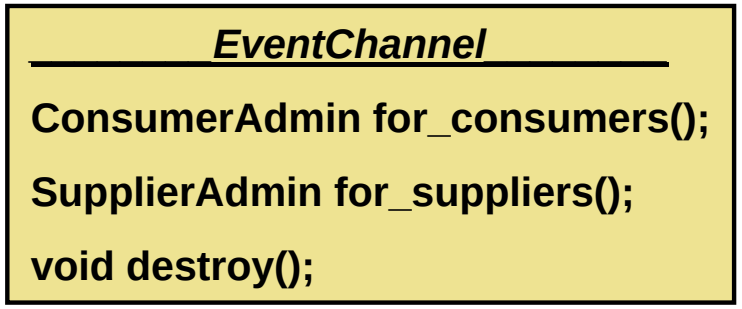


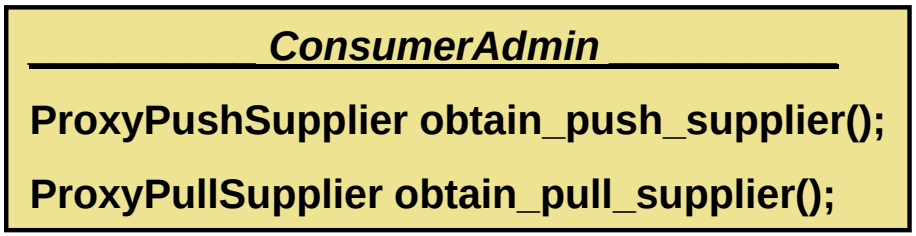
Kombinationen

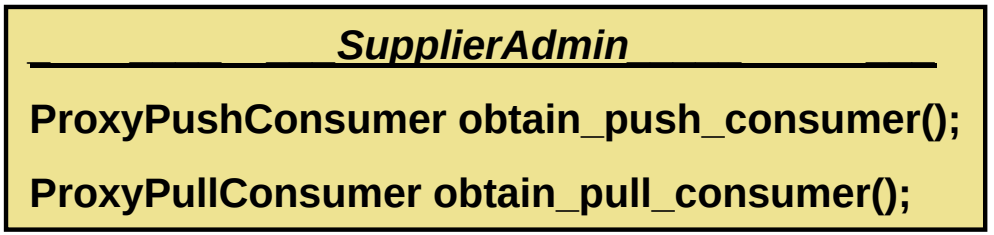
- Entkopplung durch zwischengeschalteten Event Channel
  - Beide Verarbeitungsmodelle an „beiden“ Seiten möglich
  - Mehrere Erzeuger und Verbraucher können sich registrieren
  - Übertragung der Ereignisse durch normale synchrone statische CORBA-Aufrufe



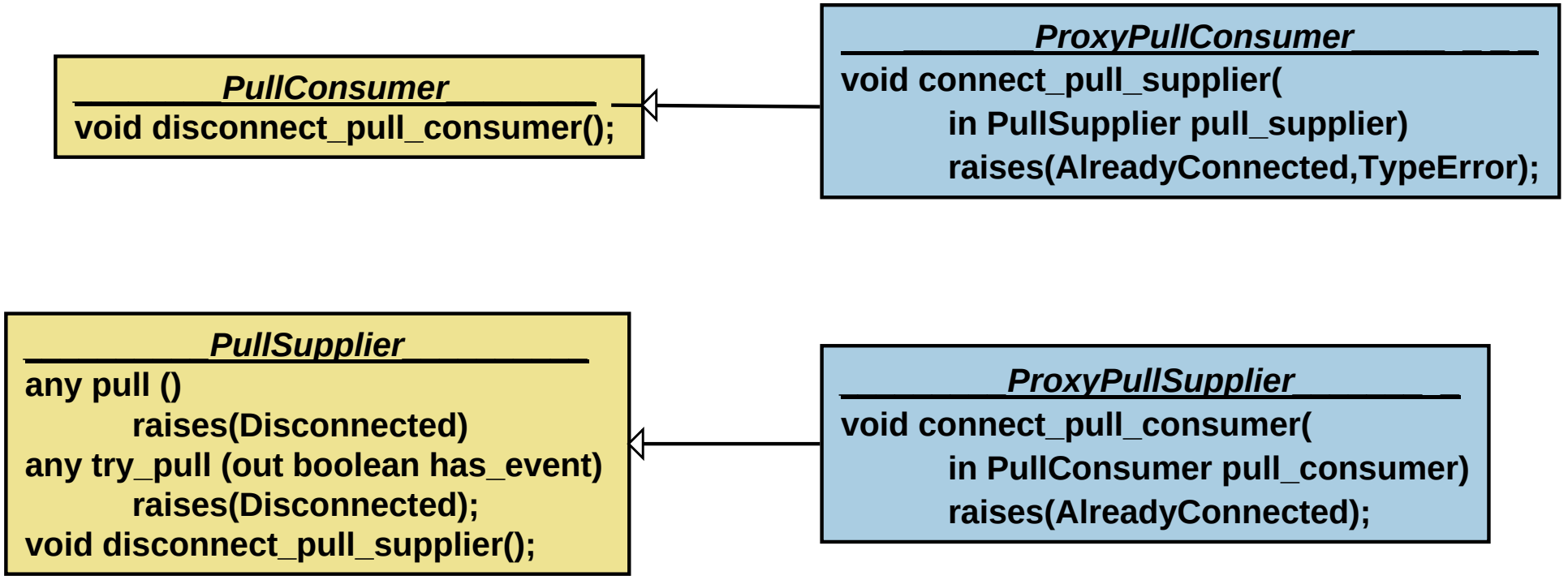
Schnittstellen zur Administration



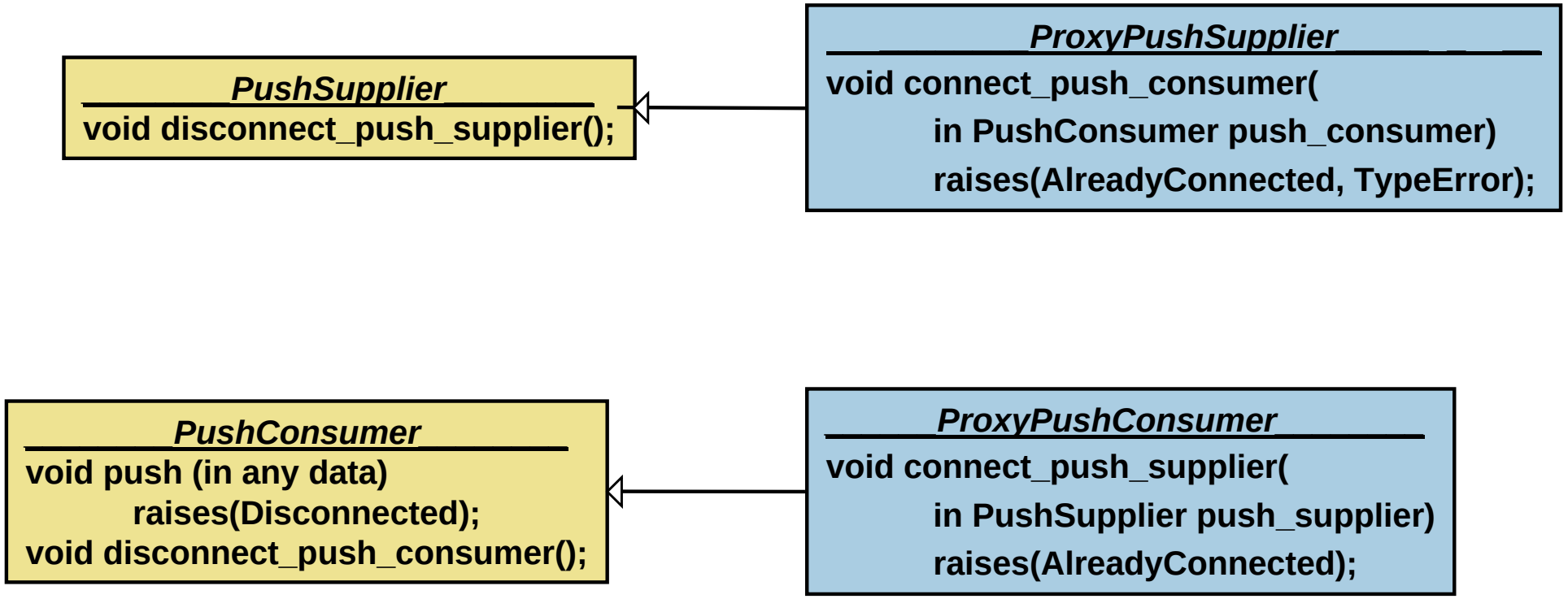




Schnittstellen für das Pull-Modell



Schnittstellen für das Push-Modell



Beispielcode

```
// Create EventChannel
event_channel = event_channel_factory -> create_event_channel();
```

```
// Code for PushSupplier
supplier_admin = event_channel -> for_suppliers();
proxy_push_consumer = supplier_admin -> obtain_push_consumer();
proxy_push_consumer -> connect_push_supplier(this);
proxy_push_consumer -> push(data);
proxy_push_consumer -> disconnect_push_supplier(this);
```

```
// Code for PullConsumer
consumer_admin = event_channel -> for_consumers();
proxy_pull_supplier = consumer_admin -> obtain_pull_supplier();
proxy_pull_supplier -> connect_pull_consumer(this);
data = proxy_pull_supplier -> pull();
proxy_pull_supplier -> disconnect_pull_consumer(this);
```

Schwächen des Event Service

- Unzureichende Filterung
  - Alle Nachrichten im Kanal werden ausgeliefert
  - Keine zusätzliche Filterung durch den Event Service auf Wunsch des Konsumenten möglich
  - Filterung nur über Kanal-Komposition möglich
- Vorgesehene Ereignis-Arten erwiesen sich als unpraktisch
  - untyped events keine Angaben über Datentypen
  - typed events strenge Typbindung der Daten
- Unklare Semantik
  - Wie lange sollen Ereignisse im Event Channel bleiben?
  - Wie können Prioritäten vergeben werden?
  - ...
- Nachfolger → Notification Service
  - Aufwärts kompatible Erweiterung des Event Service

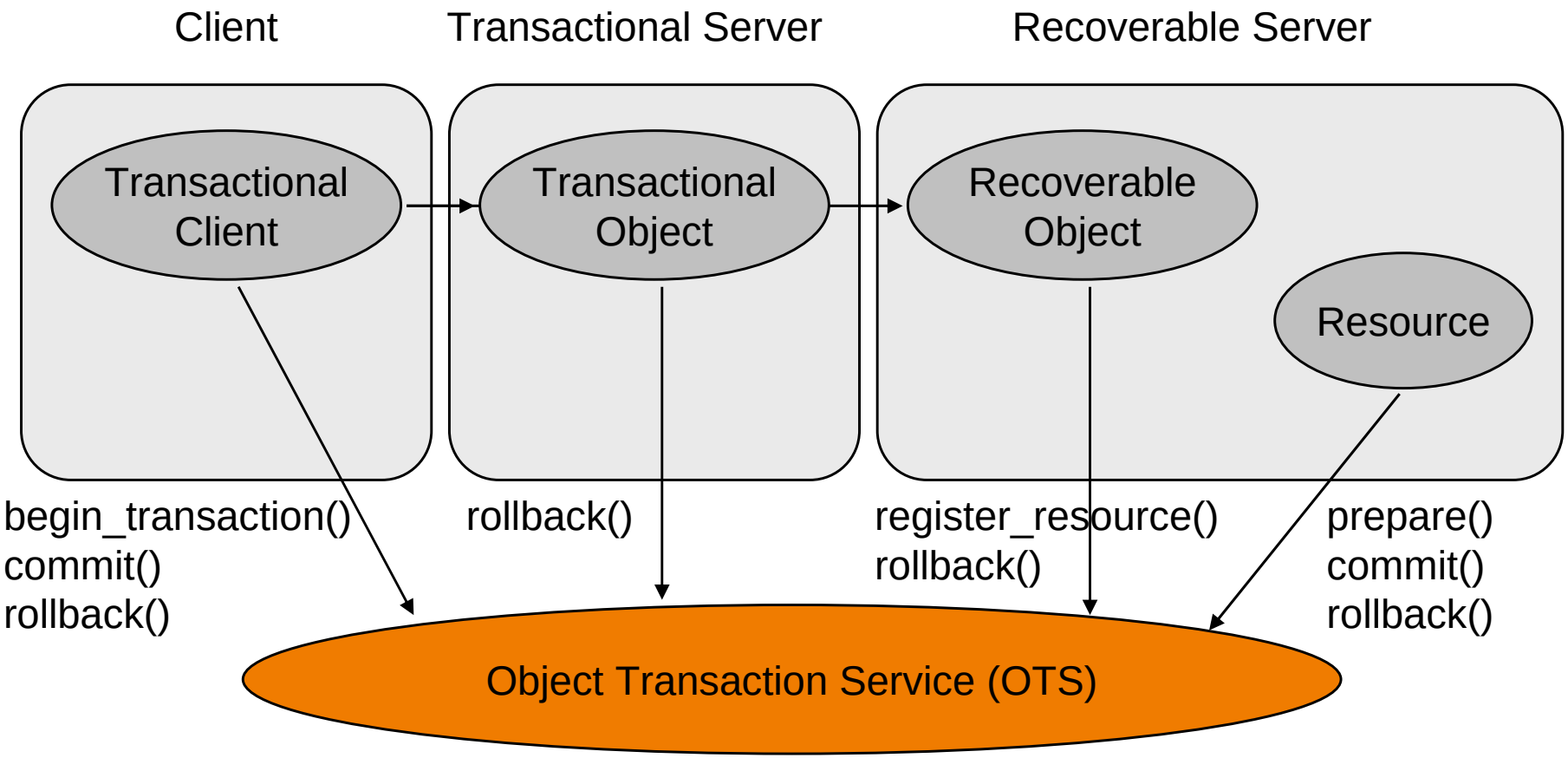
Notification Service

- Filterung
  - Selektion erwünschter Ereignisse durch Filter-Objekte
  - Inhaltsbezogene Filterung durch Filter-Sprache
- Structured Events
  - Vorgegebene Datenstruktur mit festen und optionalen Feldern
  - Effizienter als untyped und typed events
- Quality of Service
  - Garantien für die Auslieferung von Ereignissen
    best effort, persistent
  - Lebensdauer von Ereignissen (Verfallsdatum)
  - Prioritäten für bestimmte Ereignisse (-32767,...,32767)
  - Auslieferungsreihenfolge (Any, FIFO, Priority, Deadline)
  - Gruppierung bei der Auslieferung (Batching of Notifications)

Object Transaction Service (OTS)

- Unterstützt flache und geschachtelte Transaktionen
- Interoperabilität mit OTS anderer Hersteller
- Interoperabilität mit legacy transaction systems
- Kompatibilität mit existierenden Standards
  - X/Open XA, OSI TP, LU 6.2
- 2PC-Protokoll zum Abschluss der Transaktion

Komponenten einer OTS-Anwendung



Literatur

1. Object Management Group (OMG). The Common Object Request Broker: Architecture and Specification, Version 3.0.3. formal/04-03-12, March 2004.
2. Object Management Group. *CORBA Event Service Specification, Version 1.2*. OMG Document formal/04-10-12, October, 2004.
3. Object Management Group. *CORBA Notification Service, Version 1.1*. OMG Document formal/04-10-11, October, 2004.
4. Object Management Group. *CORBA Transaction Service Specification, Version 1.4*. OMG Document formal/03-09-02, September, 2003.
5. D. C. Schmidt and S. Vinoski. *Object Adapters: Concepts and Terminology*. C++ Report, 9(11), Nov. 1997.
6. D. Schmidt and S. Vinoski. *An Introduction to CORBA Messaging*. C++ Report, 10(11), Nov. 1998.
7. D. C. Schmidt and S. Vinoski. *Programming Asynchronous Method Invocations with CORBA Messaging*. C++ Report, 11(2), Feb. 1999.

Literatur

8. D. C. Schmidt and S. Vinoski. *Time-Independent Invocation and Interoperable Routing*. C++ Report, 11(4), Apr. 1999.
9. D. C. Schmidt and S. Vinoski. *Distributed Callbacks and Decoupled Communication in CORBA*. C++ Report, 8(9):48--56, 77, Oct. 1996.
10. D. C. Schmidt and S. Vinoski. *The OMG Events Service*. C++ Report, Feb. 1997.
11. D. C. Schmidt and S. Vinoski. *Overcoming Drawbacks in the OMG Events Service*. C++ Report, June 1997.

Fragen?