

# CORBA als Ordnung in verteilten Anwendungen

## Portable Object Adapter (POA) Konzept

Anton Böhm

[anton.boehm@itServe.ch](mailto:anton.boehm@itServe.ch)

**itServe AG**

P.O.Box

Länggass-Str.26

CH-3000 Bern 9

Tel. +41 31 305 16 16

Dipl. Arbeit

«Entwurf und Implementierung  
eines POA für JacORB»

Reimo Tiedemann 15.1.2000

**Developing Distributed Object  
Computing Applications with CORBA**

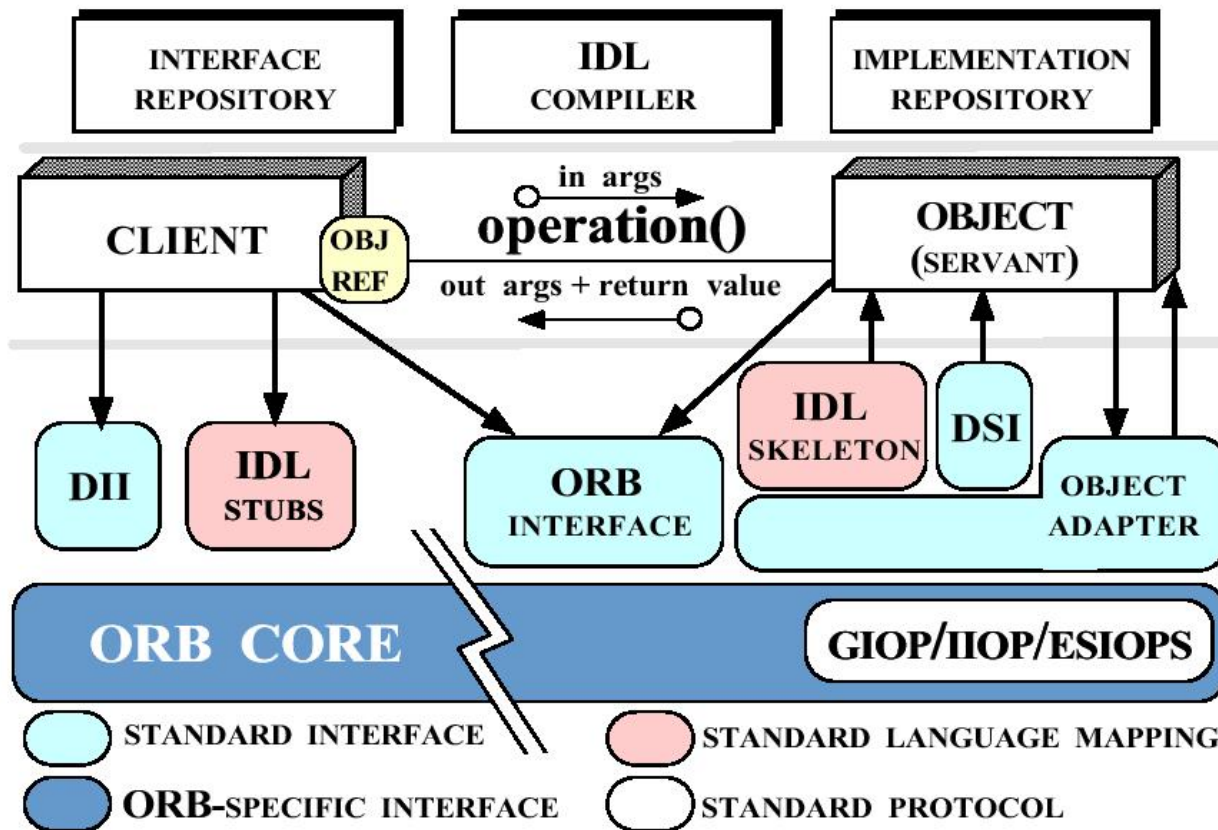
**Douglas C. Schmidt**

Associate Professor  
schmidt@uci.edu

[www.eng.uci.edu/~schmidt/](http://www.eng.uci.edu/~schmidt/)

Elec. & Comp. Eng. Dept.  
University of California, Irvine  
(949) 824-1901

# Advanced CORBA Features



## Features

- Portable Object Adapter
- Multi-threading
- Implementation Repository
- CORBA Component Model

[www.cs.wustl.edu/~schmidt/corba.html](http://www.cs.wustl.edu/~schmidt/corba.html)

# Aufruf eines CORBA-Objektes

⇒ CORBA-Standard definiert ein «Framework»:

Objekt-Implementierungen beim ORB anmelden

Zugriff mittels Objektreferenzen auf diese Objekte durchführen

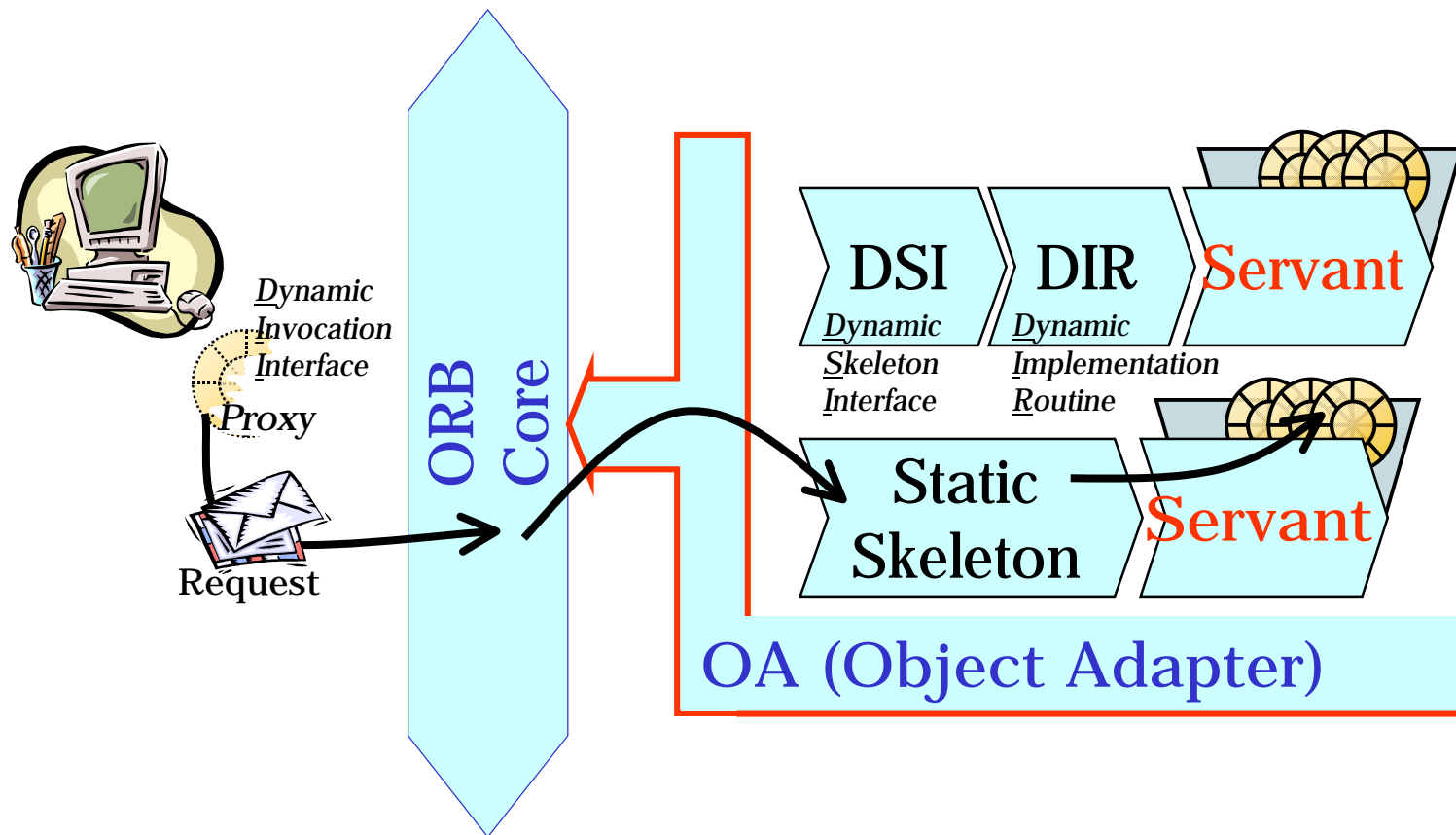
CORBA-Objekt = Virtuelle Entität,  
d.h. Abstraktions-Konzept eines Objektes

Inkarnation eines CORBA-Objektes durch den

«**Servant**»

verleiht dieser Abstraktion eine Substanz.

# Aufruf eines CORBA-Objektes



# Aufrufmodi

⇒ CORBA-Standard

«synchronous»

Identisch mit dem Prozeduraufruf, blockiert den Client bis zum Erhalten der Antwort.

«deferred synchronous»

Client wartet nicht auf das Ergebnis, sondern fährt mit seinem Prozess fort (später kann die Antwort abgeholt werden; Nutzung nur über das DII).

«oneway» auch als IDL-Schlüsselwort

Client sendet Anfrage ohne eine Antwort zu verlangen (ORB garantiert den Transfer nicht!!)

Als IDL-Default gilt der synchrone Aufruf

# BOA und POA

⇒ Portable Object Adapter (CORBA 2.3)

Aufbau nach dem Adapter-Muster

**ORB** -- **POA** -- **Servants**

Servant:

- «**incarnation**» und «**etherealization**»

Instanz durch Objekt-Implementierung

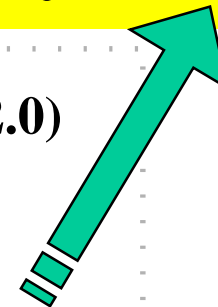
⇒ Basic Object Adapter (CORBA 2.0)

Objekt aktiv oder inaktiv

Servertypen:

«shared», «unshared» und «per-Method»  
«persistence»

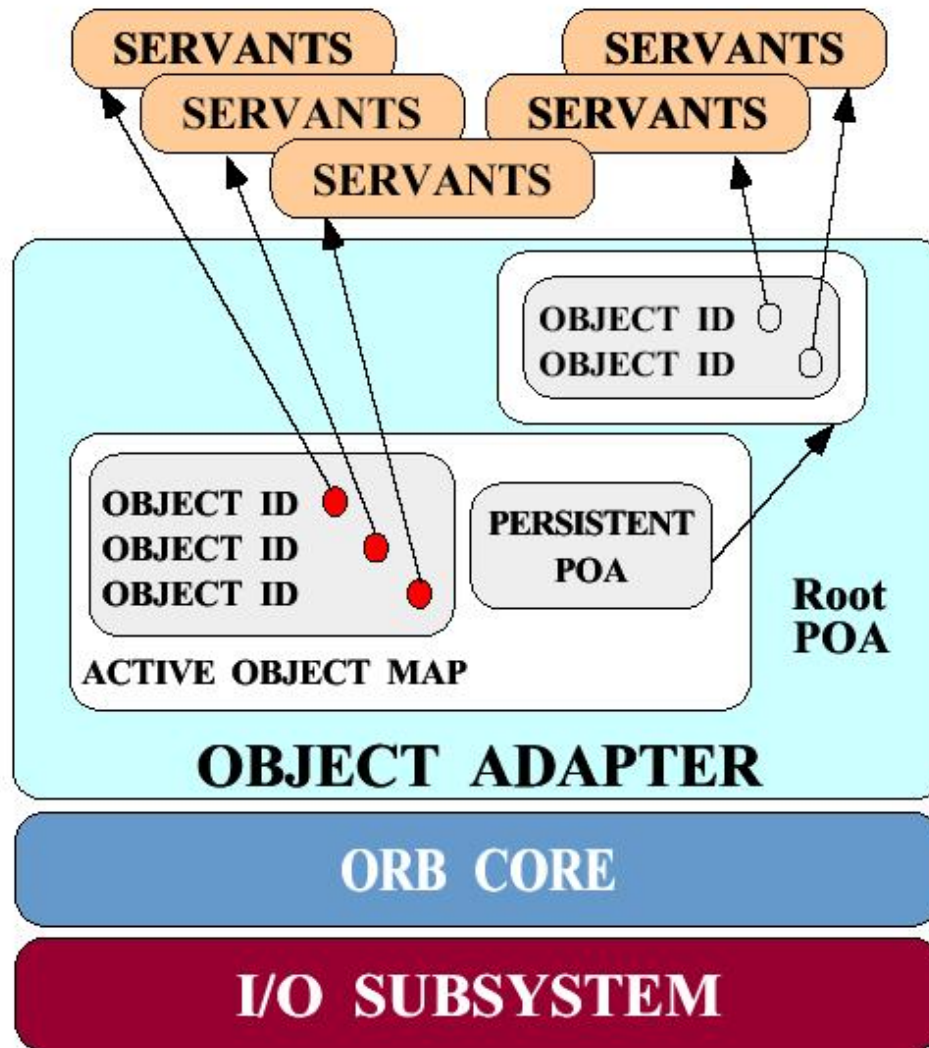
:( Portabilität der Server :(



- keine Unterstützung für persistente Objekte
- keine Unterstützung für implizite Servant-Aktivierung
- keine Kontrolle über die Objekt-Identität (Object-ID)
- kaum Konfigurationsmöglichkeiten des BOAs
- keine Möglichkeit, die Objektimplementation zur Laufzeit auszutauschen



# Overview of the Portable Object Adapter (POA)

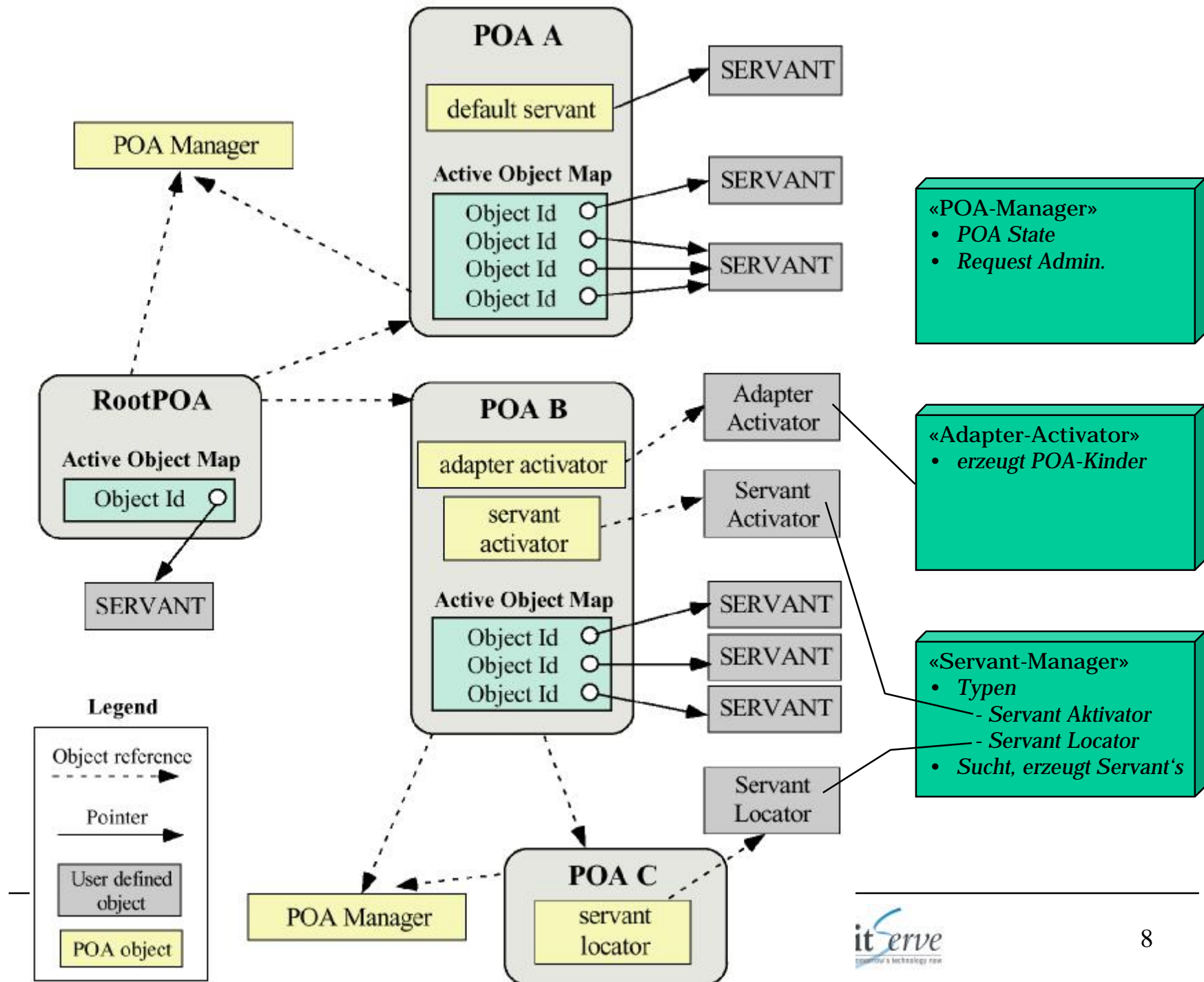


## POA Features

- Creates object refs
- Activates and deactivates objects
- Etherealizes and incarnates servants
- Maps requests to servants

The POA is very important for certain applications

- *e.g.*, telecom MIBs, enterprise servers





# Modell - Elemente

## ⇒ Servant

Ein Servant ist ein vom Anwendungsentwickler zur Verfügung gestelltes Exemplar einer Implementierungsklasse in einer konkreten Programmiersprache.

Er stellt die Implementierung für ein oder mehrere abstrakte CORBA-Objekte bereit. Aufrufe auf einer Objektreferenz werden vom ORB über einen POA an den verantwortlichen Servant vermittelt.

# Modell - Elemente

## ⇒ OR / IOR (Interoperable Object Reference)

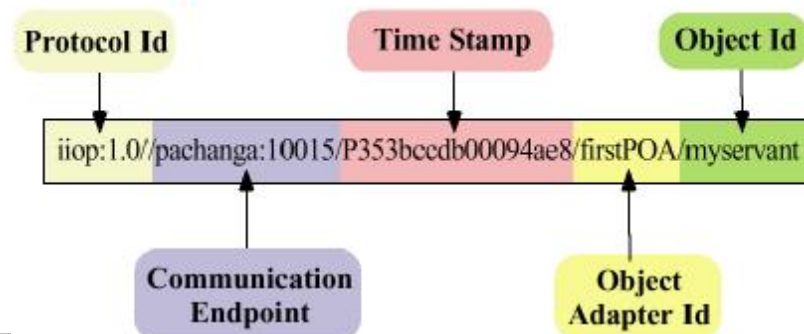
wird durch Serveranwendung generiert

Adressinformationen für verschiedene Protokolle  
(in den Profiles)

«Stub - Klasse» mit:

- **Repository-ID** (*most driven type*)
- Kommunikations-Informationen zum Server  
**Host-Adresse und Port**
- **Object-Key**

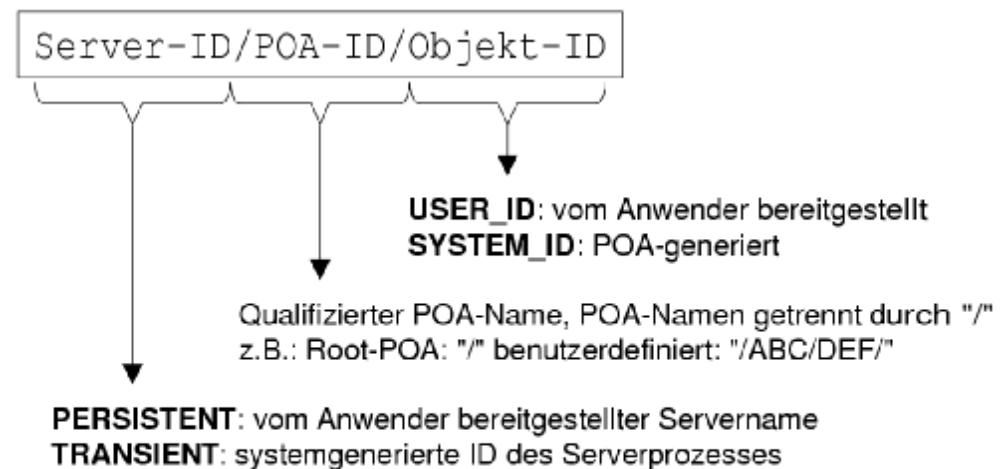
### Object Reference in URL Format



# Modell - Elemente

## ⇒ Object-Key

Der Objekt-Key enthält die Identität eines POAs als vollständig qualifizierten **POA-Namen** und die **Objekt-ID** eines abstrakten CORBA-Objektes innerhalb einer Sequenz von Bytes.



Dipl. Arbeit  
«Entwurf und Implementierung eines  
POA für JacORB»  
Reimo Tiedemann 15.1.2000

Abbildung 6.2.: Zusammensetzung eines Objekt-Keys

# Modell - Elemente

## ⇒ Objekt-ID

Sie identifiziert ein abstraktes CORBA-Objekt innerhalb eines POAs. Objekt-IDs werden entweder vom **Anwendungsentwickler** zur Verfügung gestellt oder vom **POA** generiert.

Der POA verwaltet eine Objekt-ID als eine uninterpretierte Sequenz von Bytes und benutzt sie zur **Erzeugung** von **Objektreferenzen** und zur **Ermittlung** des **verantwortlichen Servants** für einen Objektaufruf.

- **POA-Namen** und **Object-ID** (= *Object-Key*)
  - **Object-ID** erstellt durch
    - a) Entwickler definiert z.B. «*Kölnisch-Wasser*»
    - b) POA generiert z.B. «*4711*»

# Modell - Elemente

## ⇒ POA

Ein POA ist die Programmkomponente, die die Auslieferung einer Anfrage an ein Servant innerhalb einer Serveranwendung realisiert und den Entwickler bei der *Erzeugung, Verwaltung und Zerstörung* von CORBA - Objekten unterstützt.

Jeder POA verwaltet einen **Namensraum** von Objekt-IDs für die eigenen CORBA-Objekte und einen Namensraum für weitere verschachtelte Kind-POAs. **Verschachtelte POAs** bilden somit einen **hierarchischen Namensraum** für CORBA-Objekte innerhalb einer Serveranwendung.

# Modell - Elemente

## ⇒ Policy

Eine Policy ist ein Objekt, das mit einem POA assoziiert wird und damit die Eigenschaft des POAs und die von ihm verwalteten CORBA-Objekte definiert.

Durch die Auswahl geeigneter Policy-Werte bei der Konfiguration kann der Anwendungsentwickler den POA an die speziellen Erfordernisse von Objekt-Implementierungen anpassen. Anzahl und Art der Policies, ihre Werte und deren Konsequenzen auf das Verhalten eines POAs sind durch die Spezifikation vorgegeben. Das Spektrum der mittels Policies konfigurierbaren POA-Eigenschaften reicht von der Festlegung einzelner Optionen, die den Ablauf der Anfragebearbeitung beeinflussen, bis hin zu verschiedenen Optionen, die die Verwaltung der CORBA-Objekte durch einen POA betreffen.



# Modell - Elemente

## ⇒ POA-Manager

Ein POA-Manager kapselt den Zustand eines POAs. Es können mehrere POAs einem POA-Manager zugeordnet werden.

Mit einem POA-Manager lässt sich der Strom von Anfragen steuern, der die Objekt-Implementierungen erreicht. Es kann bestimmt werden, ob die mit einem POA-Manager assoziierten POAs alle eintreffenden Anfragen

- bearbeiten,
- puffern oder
- zurückweisen

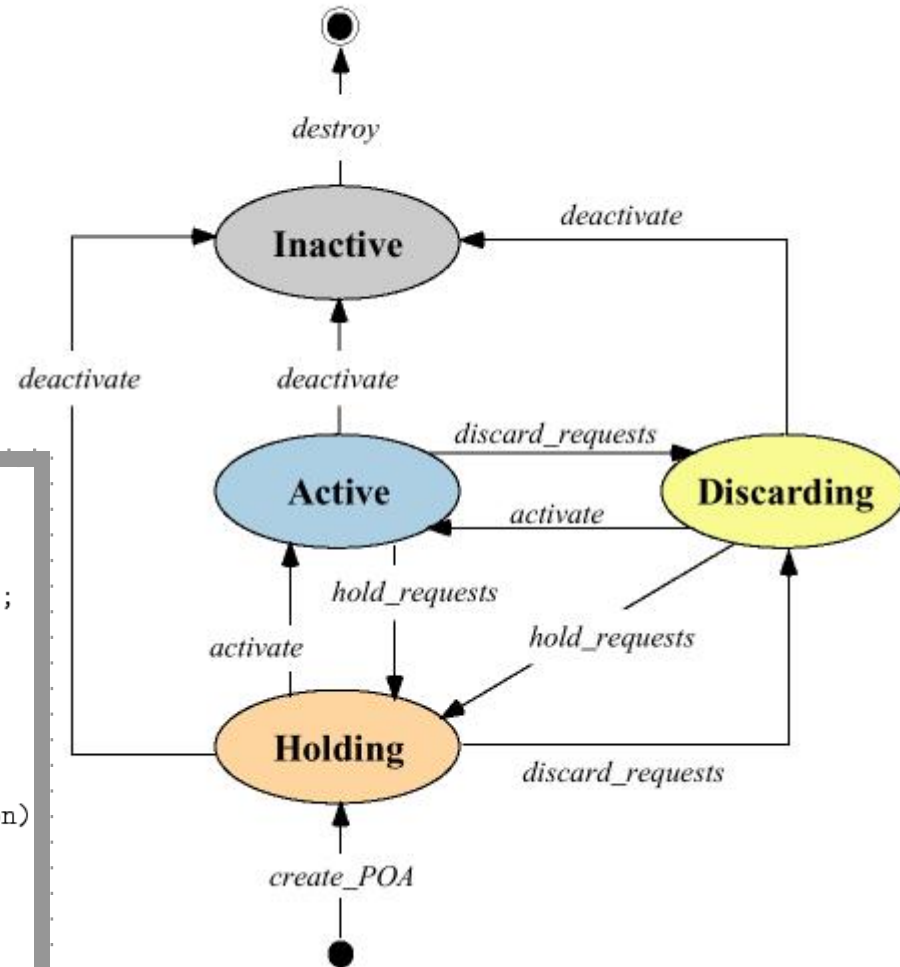
sollen. Außerdem können POAs über ihren POA-Manager deaktiviert werden.

# Modell - Elemente

## ⇒ POA-Manager

```
module PortableServer {  
  interface POAManager {  
    exception AdapterInactive {};  
    enum State { ACTIVE, HOLDING, DISCARDING, INACTIVE };  
    State get_state();  
  
    void activate() raises (AdapterInactive);  
    void hold_requests(in boolean wait_for_completion)  
      raises (AdapterInactive);  
    void discard_requests(in boolean wait_for_completion)  
      raises (AdapterInactive);  
    void deactivate(in boolean etherealize_objects,  
      in boolean wait_for_completion)  
      raises (AdapterInactive);  
  };  
};
```

## POA Manager Processing States



# Modell - Elemente

## ⇒ Servant-Manager

Ein Servant-Manager ist ein vom Anwendungsentwickler zur Verfügung gestelltes und mit einem POA assoziiertes lokales CORBA-Objekt, das den POA bei der Verwaltung von Objekt-Implementierungen und der Vermittlung von Anfragen unterstützen kann.

Der POA ruft Operationen auf einem Servant- Manager auf, um bei Bedarf, also im Falle einer Anfrage, den aktuellen Servant zu ermitteln bzw. zu erzeugen oder bei einer Objektdeaktivierung den Servant-Manager zu informieren.

# Modell - Elemente

## ⇒ Servant-Manager (II)

Zur Identifizierung eines Servants wird dem Servant-Manager die Objekt-ID übergeben. Der Servant-Manager verwaltet

Objekt-ID-zu-Servant-Beziehungen

und entscheidet bei einem Aufruf, ob ein CORBA-Objekt existiert oder nicht.

Es gibt zwei verschiedene Arten eines Servant-Managers,  
einen Servant-Activator (first time) bzw.  
einen Servant-Locator (each time) .

Die zum Einsatz kommende Variante hängt von den Policies ab, mit denen ein POA konfiguriert wurde.

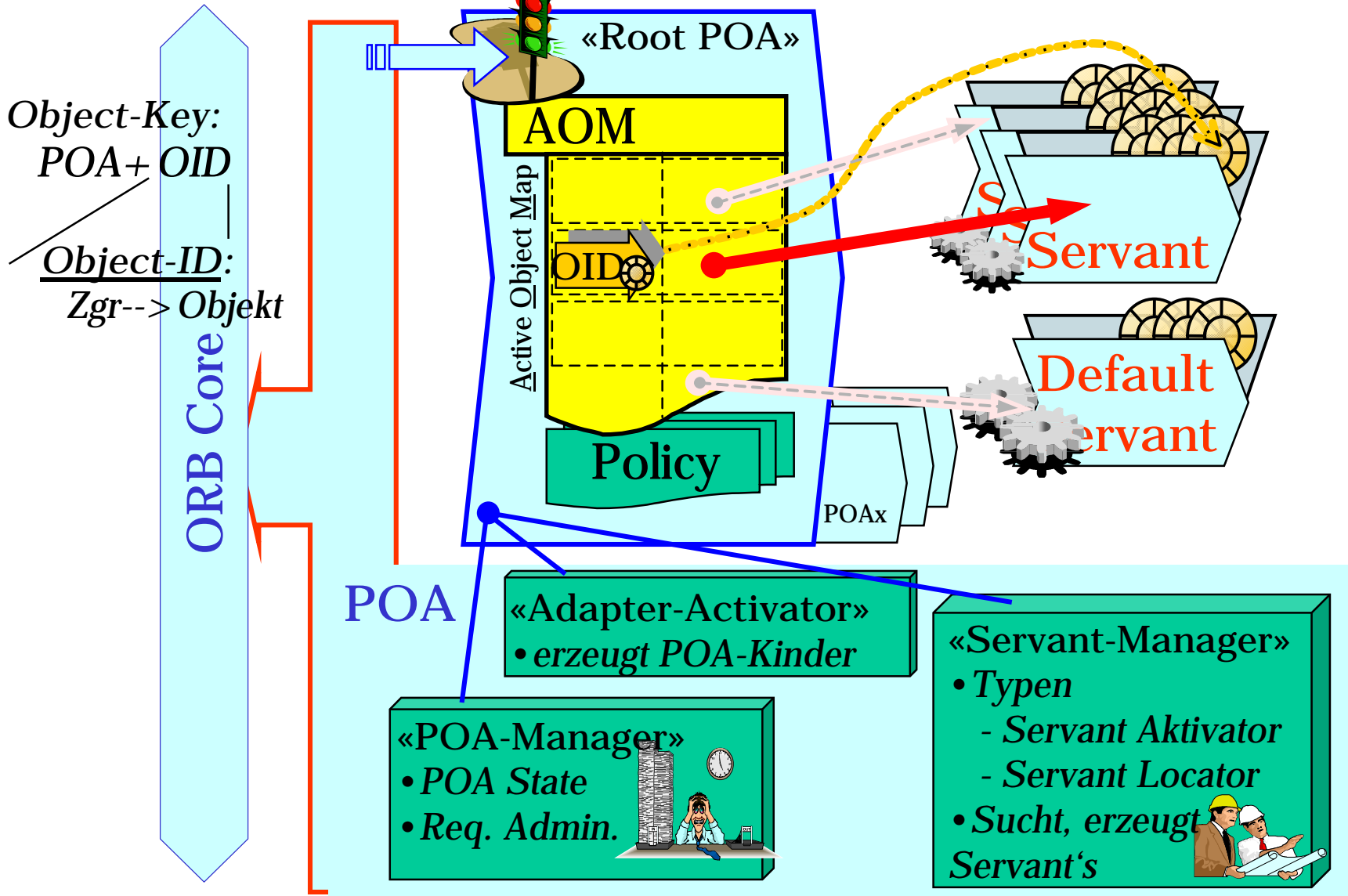
# Modell - Elemente

## ⇒ Adapter-Activator

Ein Adapter-Activator ist ein vom Anwendungsentwickler zur Verfügung gestelltes und mit einem POA assoziiertes lokales CORBA-Objekt, das für einen POA bei Bedarf einen Kind-POA erzeugen kann, wenn der für eine Anfrage zuständige POA noch nicht existiert.

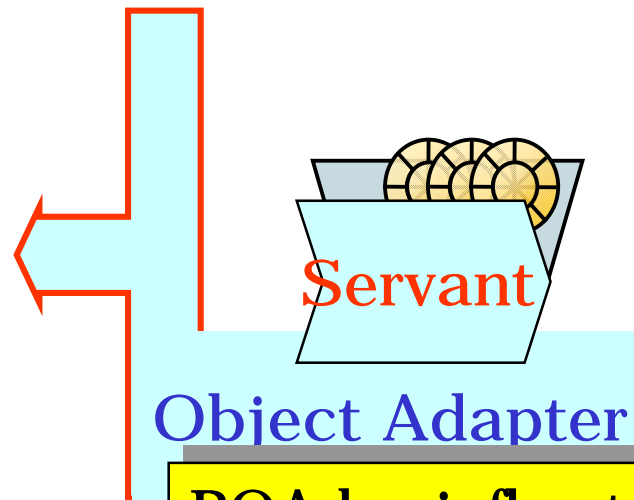
Ein Adapter-Activator ist für die Erzeugung eines POAs unter dem gewünschten Namen und für dessen Initialisierung verantwortlich.

# POA-Architektur





# POA - Policies



**POA beeinflusst das Objekt !**

**Objekt-Lebenszyklus**

- OR ist konzeptionell zeitlich unbegrenzt gültige Anfrage
- falls kein POA & Servant aktiv, dann kann der ORB einen Prozess starten

**Lösung: «well-known» Port & Daemon mit ImplementationRepository (ImR)**

# POA - Policies

- **LifespanPolicy**  
Können die in diesem POA implementierten CORBA-Objekte den Server-Prozess „überleben“ ?
- **IdUniquenessPolicy**  
Kann ein Servant mehrere Objekt-IDs unterstützen?
- **IdAssignmentPolicy**  
Wer erzeugt die Objekt-IDs, POA oder Applikation?
- **ImplicitActivationPolicy**  
Unterstützt der POA das implizite Aktivieren eines Servants?
- **ServantRetentionPolicy**  
Bleiben Servants nach Abarbeitung eines Requests aktiv und in der *Active Object Map* vermerkt?
- **RequestProcessingPolicy**  
Wie werden Requests vom POA bearbeitet und an einen Servant vermittelt?
- **ThreadPolicy**  
Wie werden Requests Threads zugeordnet?

# POA - Policies (2)

## 1) Lifespan Policy

Servant's des POA existieren (*für den semantisch richtigen Objektzustand ist die Anwendung verantwortlich*)

- TRANSIENT: die Lebenszeit des Objektes ist durch die Lebenszeit des Servers, in dem es läuft, beschränkt.
- PERSISTENT: das Objekt kann länger existieren als der Server, in dem es ursprünglich gestartet wird.

## 2) Id Assignment Policy

- USER\_ID: Object Ids werden vom Programmierer vergeben
- SYSTEM\_ID: Object Ids werden von der CORBA-Implementierung vergeben

# POA - Policies (3)

Wird für den Root  
POA verwendet

## 3) Implicit Activation Policy

- IMPLICIT\_ACTIVATION: implizite Aktivierung von Servants, wenn auch SYSTEM\_ID und RETAIN gesetzt sind.
- NO\_IMPLICIT\_ACTIVATION: keine implizite Aktivierung von Servants.

## 4) Servant Retention Policy

- RETAIN: die Lebensdauer des Servants ist länger als ein Methodenaufruf (bewahren)
- NON\_RETAIN: die Lebensdauer des Servants ist auf einen Methodenaufruf beschränkt

# POA - Policies (4)

## 5) Request Processing Policy

- `USE_ACTIVE_OBJECT_MAP_ONLY`: der POA schaut nur in der active Object-Map nach bereits laufenden Servants.
- `USE_DEFAULT_SERVANT`: wird die Objekt-Id nicht in der active Object-Map gefunden, wird ein default Servant verwendet.
- `USE_SERVANT_MANAGER`: wird die Objekt-Id nicht in der active Object-Map gefunden, wird ein Servant-Manager herangezogen, um den Servant zu finden/erzeugen.

# POA - Policies (5)

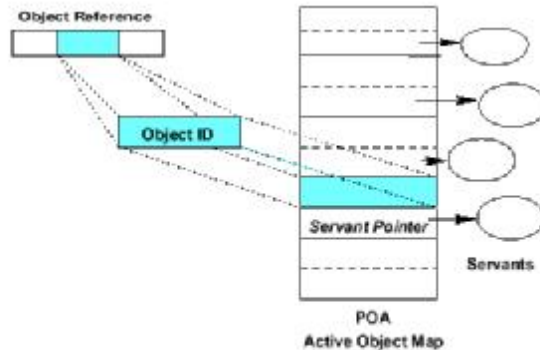


Abbildung 1: Servants in einem UNIQUE\_ID-POA.

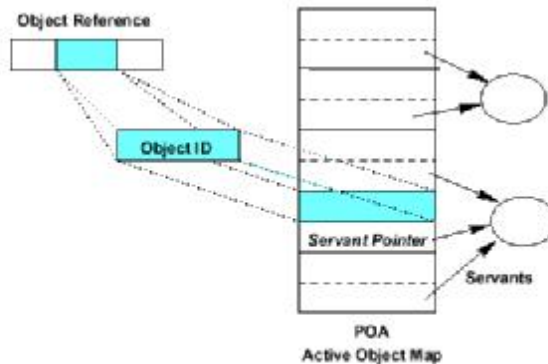


Abbildung 2: Servants in einem MULTIPLE\_ID-POA.

## 6) Object Id Uniqueness Policy

Die Eindeutigkeit der Objekt\_ID-zu-Servant-Beziehung innerhalb einer AOM lässt sich mit einer IdUniquenessPolicy konfigurieren.

- UNIQUE\_ID: ein Servant ist für genau eine Object-Id zuständig.
- MULTIPLE\_ID: ein Servant ist für mehrere Object-Ids zuständig.



# POA - Policies (6)

Wird für den Root  
POA verwendet

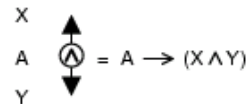
## 7) Thread Policy

Nebenläufigkeit; Multi-Threading sofern der POA es unterstützt

- ORB\_CTRL\_MODEL, SINGLE\_THREAD\_MODEL

# Einsatz der POA - Policies

Policy-Werte	Beispiele				Abhängigkeiten	Root-POA	Standard
	1	2	3	4			
PERSISTENT	■	■					
TRANSIENT			■	■		■	■
SYSTEM_ID			■	□		■	■
USER_ID	■	■		■			
NO_IMPLICIT_ACTIVATION	■	■		■			■
IMPLICIT_ACTIVATION			■			■	
RETAIN		■	■			■	■
NON_RETAIN	■			■			
USE_ACTIVE_OBJECT_MAP_ONLY		■	■			■	■
USE_SERVANT_MANAGER	■		■	■			
USE_DEFAULT_SERVANT		□	□	□			
UNIQUE_ID		■	■			■	■
MULTIPLE_ID		□	□				
SINGLE_THREAD_MODEL							
ORB_CTRL_MODEL						■	■



**Beispiel 1:** Große Anzahl persistenter Objekte, deren Zustand in einem Hintergrundspeicher verwaltet wird (Daten-Objekte).

**Beispiel 2:** Kleine Anzahl persistenter Objekte (bzw. ein Objekt), die öffentliche Dienste bereitstellen (Service-Objekte).

**Beispiel 3:** Kleine bis mittlere Anzahl transienter Objekte, die Hilfsfunktionalität bereitstellen.

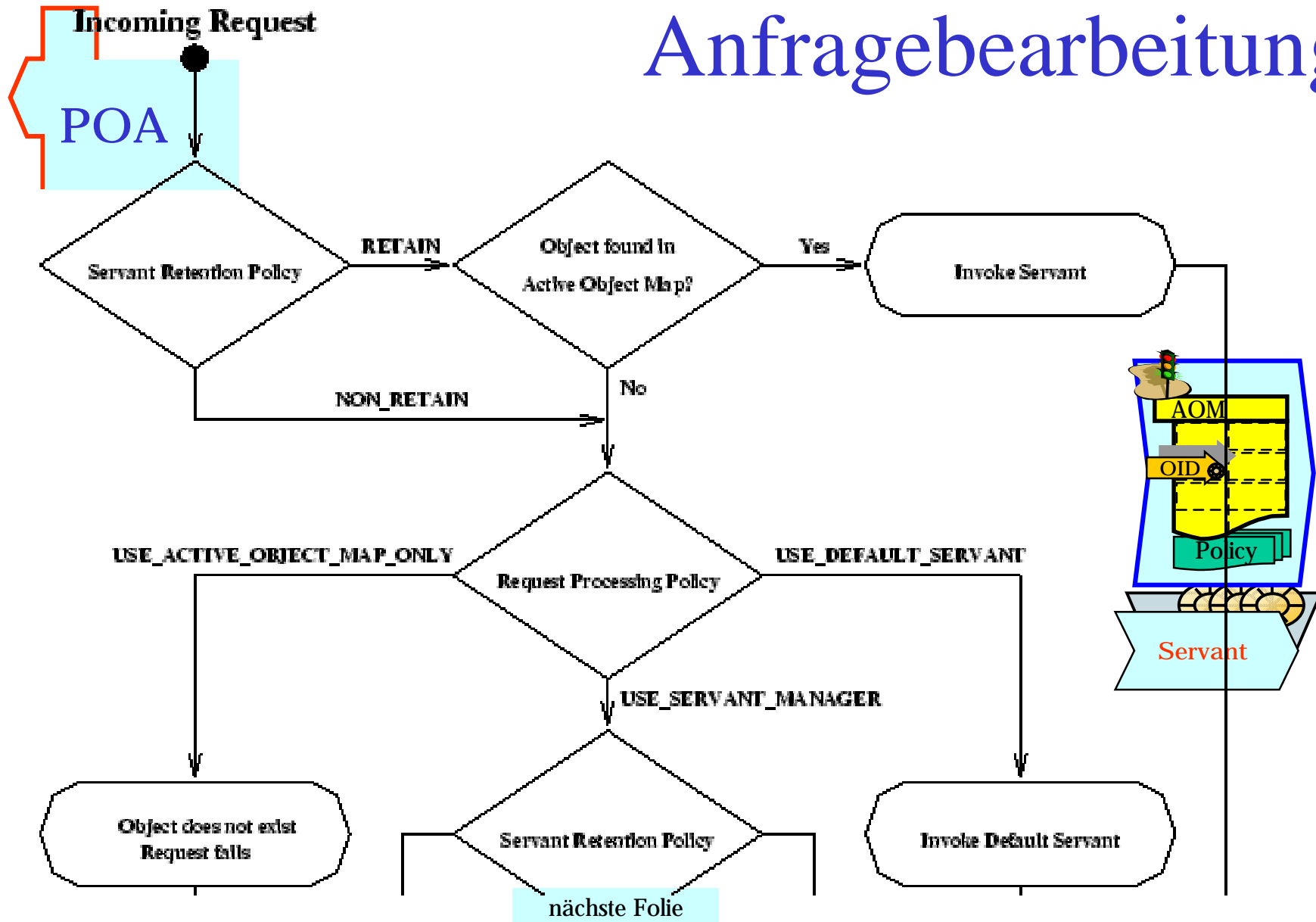
**Beispiel 4:** Unbekannte bzw. potentiell große Anzahl transienter Objekte, deren Rolle an den Serverprozeß gebunden ist.

# Konvertierungsoperationen

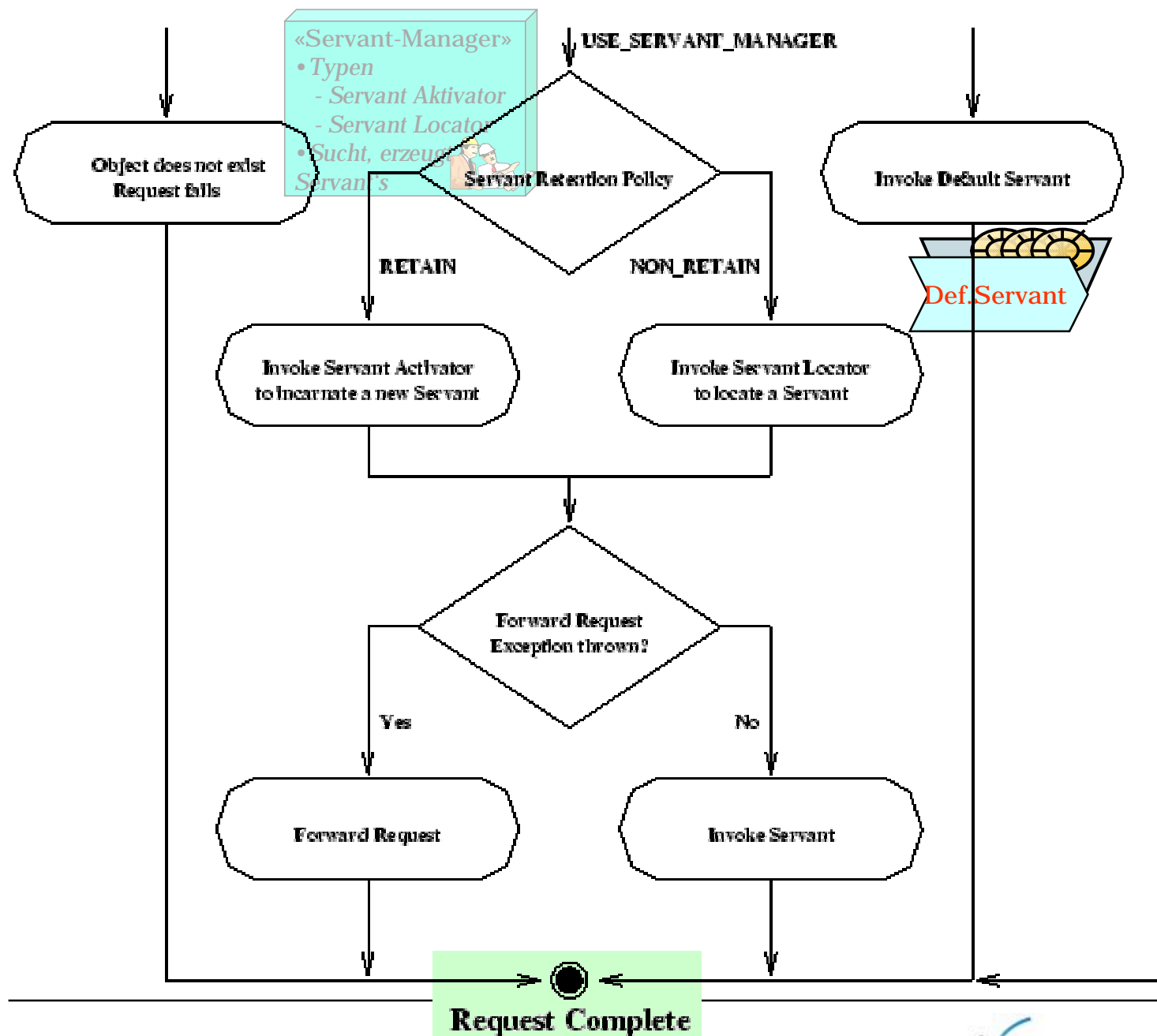
## Object, ObjectId u. Servant

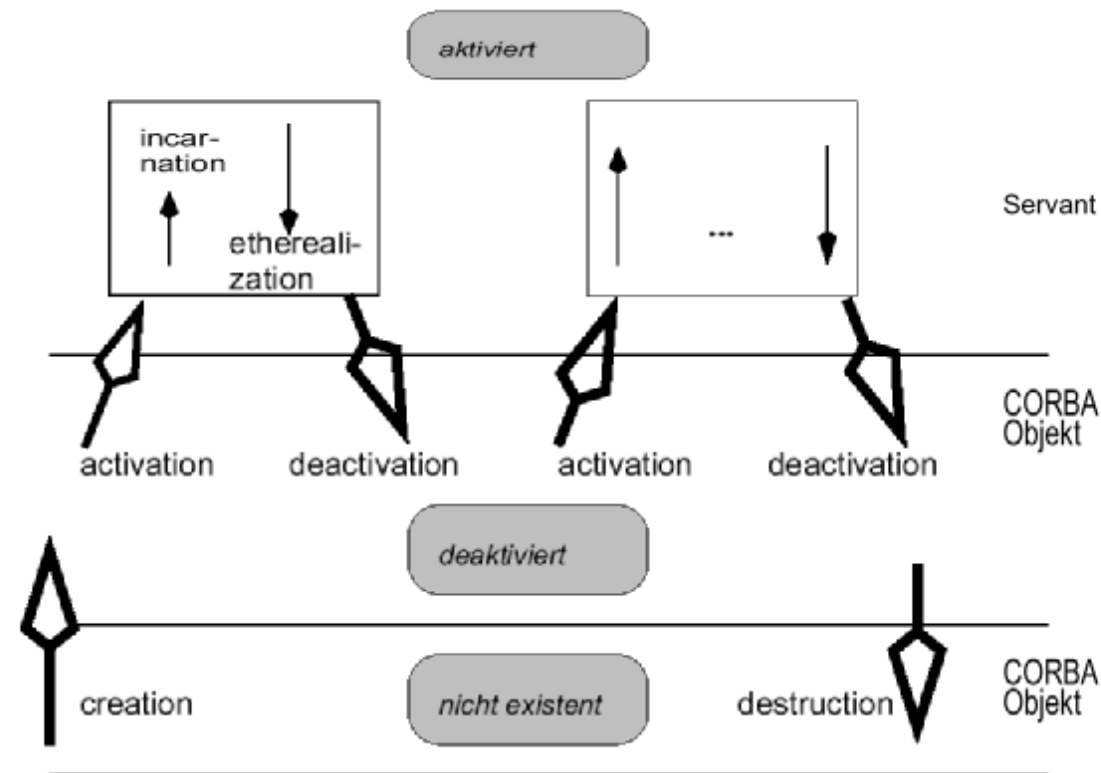
```
module PortableServer {  
    interface POA {  
        ObjectId servant_to_id(in Servant servant)  
            raises(ServantNotActive, WrongPolicy);  
        Object servant_to_reference(in Servant servant)  
            raises(ServantNotActive, WrongPolicy);  
        -----  
        Servant reference_to_servant(in Object reference)  
            raises(ObjectNotActive, WrongAdapter,  
                WrongPolicy);  
        ObjectId reference_to_id(in Object reference)  
            raises(WrongAdapter, WrongPolicy);  
        -----  
        Servant id_to_servant(in ObjectId oid)  
            raises(ObjectNotActive, WrongPolicy);  
        Object id_to_reference(in ObjectId oid)  
            raises(ObjectNotActive, WrongPolicy);  
    }  
}
```

# Anfragebearbeitung



(II)





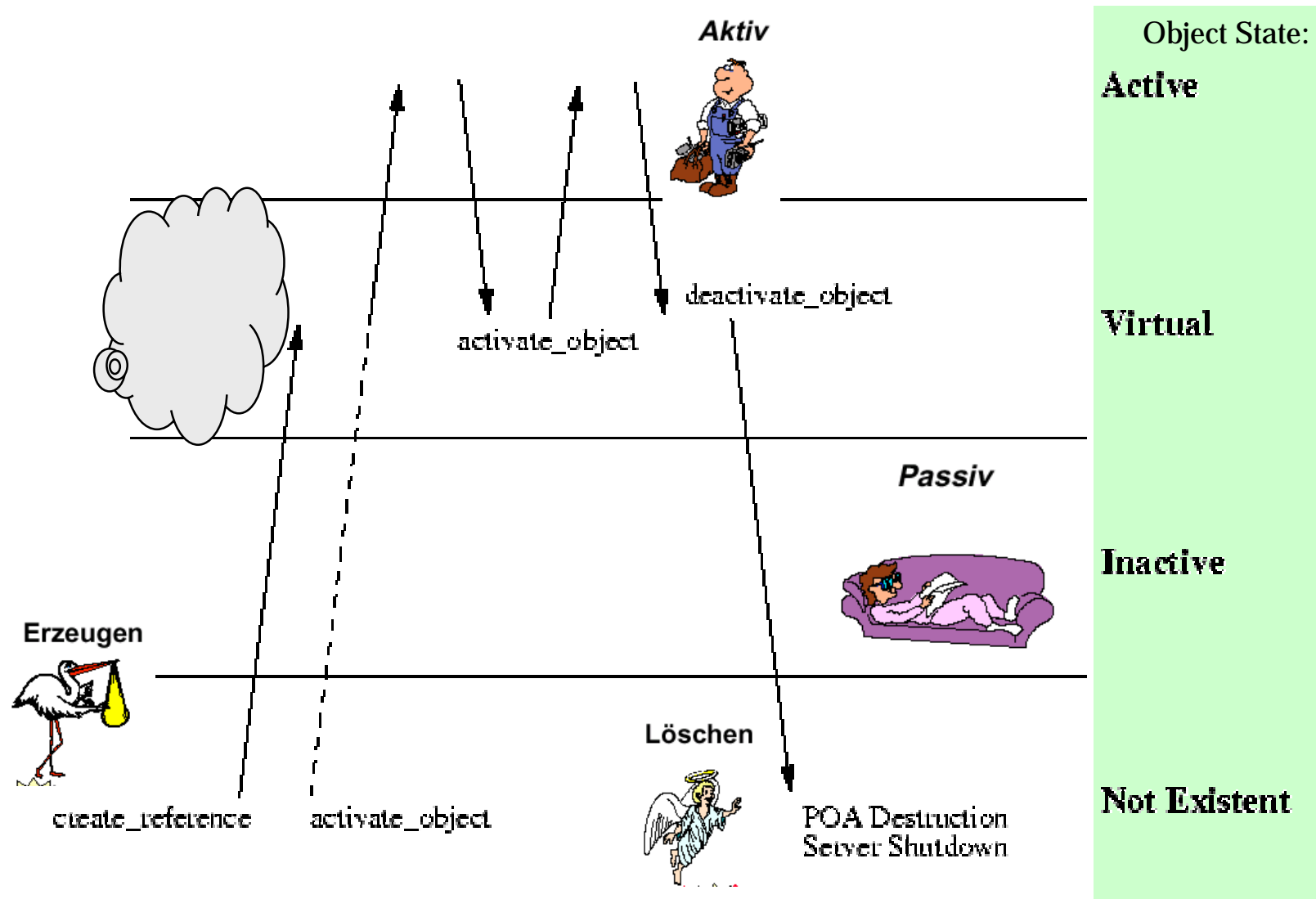
Ein CORBA-Objekt wird erzeugt. Es kann aktiviert (mit einem Servant assoziiert) werden, z.B. sobald ein Request vorliegt.

Ein Objekt kann während seiner Lebenszeit Requests durch mehrere Servants bearbeiten lassen, die nacheinander das CORBA-Objekt verkörpern (*incarnate*) und es wieder verlassen (*etherealize*).

CORBA-Objekte können innerhalb ihrer Lebenszeit aktiv und inaktiv sein, bis sie schließlich zerstört werden.



# Lifecycle Transient-Object



# Lifecycle Persistent-Object

