CORBA

CORBA Overview
CORBA

- Introduction
- Architecture
- OMG IDL
- ORB
- Object Model
- The Interoperability Architecture
- Language mappings

Object Management Group(OMG)

CORBA/OMG

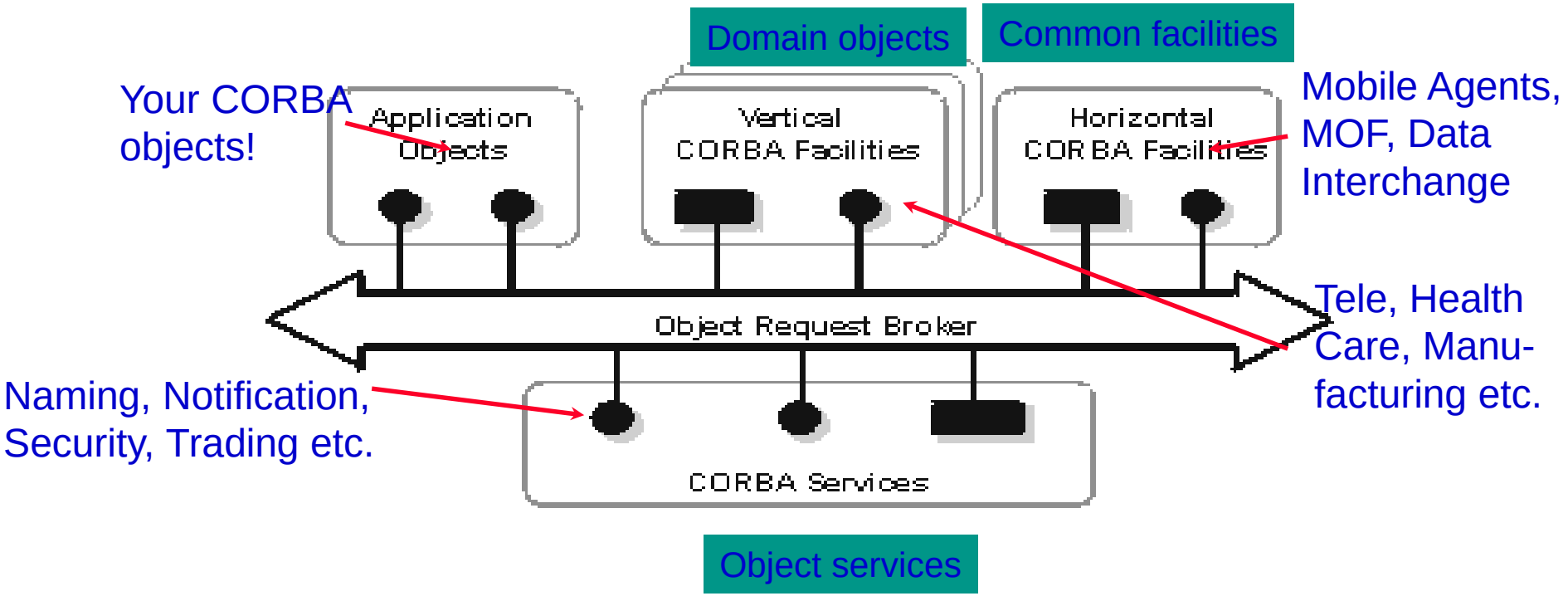
- CORBA is the acronym for Common Object Request Broker Architecture
- CORBA is OMG's open, vendorindependent *specification* for an architecture and infrastructure that computer applications use to work together over networks
- OMG is the world's largest computer industry consortium with over 800 members OMG began its work in 1989
- The goals of OMG are
  - promotion of the object-oriented approach to software engineering
  - development of a common architectual framework for writing distributed object-oriented applications based on interface specifications for the objects in the application

Object Management Group (OMG) OMA

- Besides CORBA OMG also controls the Object Management Architecture (OMA) specification and other specifications in the area Analysis and Design (XMI, UML etc.)
- OMA is the framework which all OMGs adapted technology fits
- OMA consists of two main parts
  - Core Object Model
  - The Reference Model

OMA Reference Model

The OMA Reference Model categories objects into four application areas



OMA defines the interfaces for the objects and leave the implementation to software vendors

OMA is OMGs vision for a software component environment where all their work fits

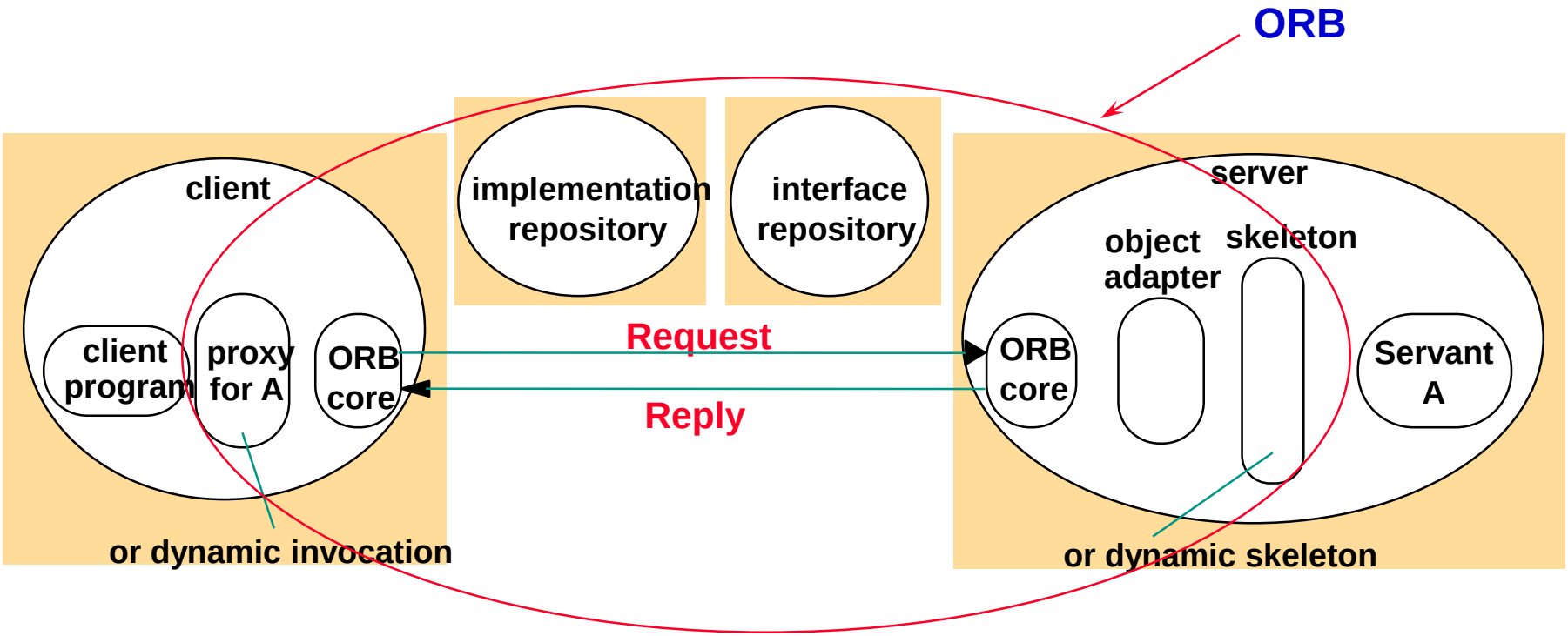
CORBA Overview
Introduction

- CORBA builds on the OMA Core Object Model and provides
  - syntax and semantics for IDL
  - A framework for interoperability: two specific protocols
  - a set of language mappings from IDL to prog.lang.
- CORBA is operating with transparency
  - Location transparency
  - Programming Language transparency
  - Platform/vendor transparency
  - Network HW/SW transparency

CORBA Overview
Introduction

- The key to *Location transparency* is the Object Request Broker (ORB)
- The key to *Programming Language Transparency* is an implementation neutral Interface Definition Language called OMG IDL that provides separation of interface and implementation
- The key to *Platform transparency and Network HW/SW transparency* is GIOP/CDR (Common Data Representation)

CORBA Overview
Architecture



The *Orb core* equals the *Communication Module* in the generic RMI architecture

The ORB can be implemented in many ways; stand-alone, distributed
To the programmer the ORB is a pseudo-object; interface to library functions

CORBA Overview
Architecture

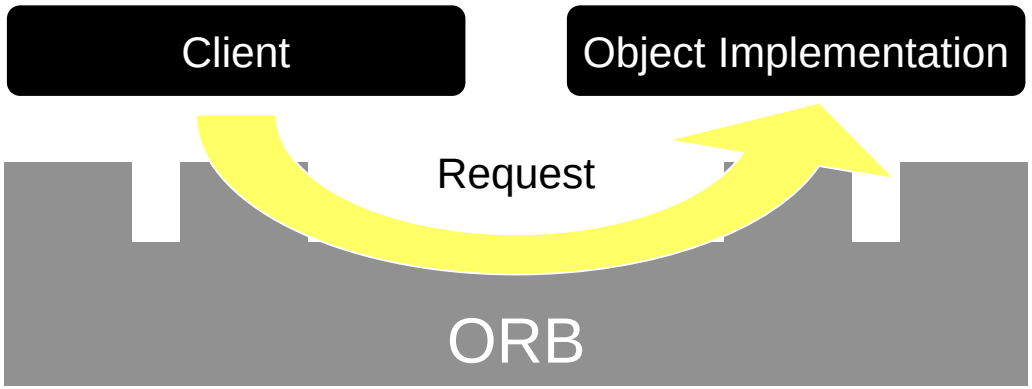
- Compared to the Generic RMI architecture there is only 3 new modules
  - Object Adapter
  - Implementation Repository
  - Interface Repository
- The communication protocol used by CORBA is based on the GIOP (General Inter-ORB Protocol) specification
- IIOP (Internet Inter-ORB Protocol) denotes the implementation of GIOP over TCP/IP

CORBA Overview
ORB - Components

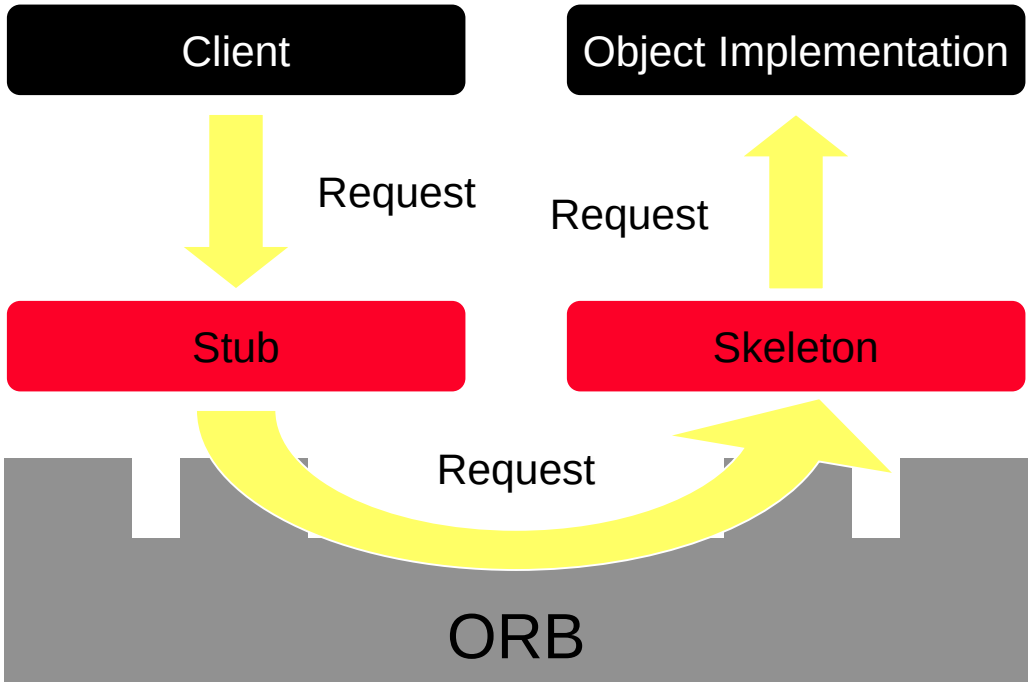
- Implementation Repository (optional)
- Interface Repository
- Client Stubs
- Server Skeletons
- Portable Object Adapter (POA)
- Dynamic Invocation Interface (DII)
- Dynamic Skeleton Interfaces (DSI)

Object Request Broker (ORB)

- Abstracts remote request and response mechanisms
- Transport for *distributing* method invocations



Proxy-based Invocation



Static Invocation

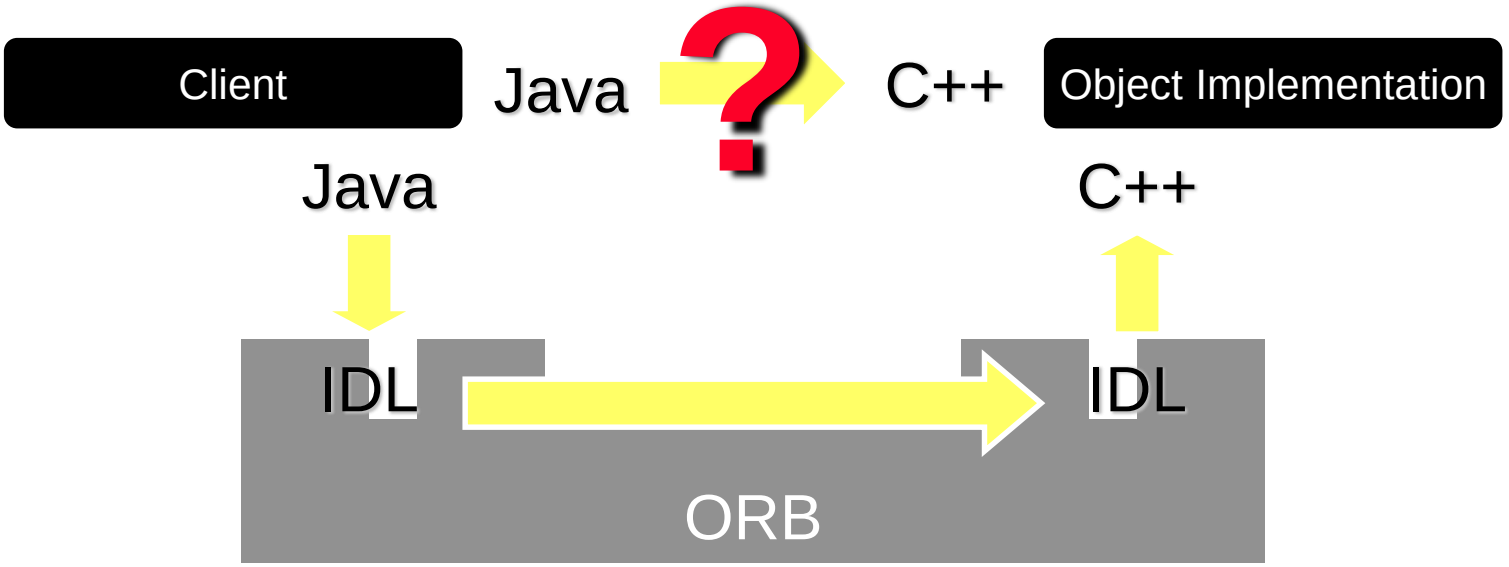
- Proxy objects generated to support distributed invocation
  - Interface defined using IDL
  - Stub and Skeleton classes
    - Language specific
  - Network and marshalling support
- Interface defined statically at compile time
  - Used when client is built

Dynamic Invocation Interface

- Alternative to static Stub/Skeleton calls
  - Don’t need Stubs when client is built
- Structure a generic invocation structure and submit to DII
- Asynchronous (*deferred synchronous*) calls
- Slower than static but more flexible
- Similar to Java Reflection

ORB Abstraction

- How is this possible in a heterogeneous environment?



Interface Definition Language

Interface Definition Language (IDL)

- Specification language
- Language independent interface
  - Declare interfaces to object methods
  - IDL maps to many high-level programming languages
- Design paradigm
  - Code to interface specified in the IDL regardless of implementation

OMG Language Mappings

- Mapping IDL to programming language
  - Many OMG standard mappings
    - C
    - C++
    - Smalltalk
    - Ada ‘95
    - COBOL
    - Java

Key IDL Language Elements

- Module
- Interface
- Attribute
- Operation
- Argument
- Exception
- Struct
- Typedef
- Sequence
- Any

Sample IDL Definition

```
// Quote system module
module QuoteSystem
{
  // Specify a data structure for quote
  struct Quote
  {
    string value;
  }
  // Specify interface to quote server
  interface QuoteServer
  {
    // Specify an stock exchange name attribute
    string exchange;
    // Unknown symbol exception
    exception UnknownSymbolException { string message; };
    // Lookup symbol
    Quote getQuote (in string symbol)
                    raises (UnknownSymbolException);
  };
};
```

specifies the scope/package QuoteSystem

define the struct for a Quote value

defines a QuoteServer object's interface

data member

return type

argument (direction and type)

declares that method throws an exception

Modules & Interfaces

- Module
  - Maps to a **package** in Java
  - Name space scoping
  - Module can contain multiple interfaces
- Interface
  - Maps to a set of related classes & interfaces

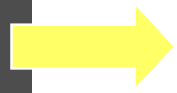
```
module QuoteSystem
{
  interface QuoteServer
  {
    ...
  };
};
```

```
QuoteSystem.QuoteServer
```

Struct

- Structure
  - Maps to a class in Java
  - Construct to hold logical blocks of data
  - Accessors and mutators
    - Generated for all data elements within structure

```
struct Quote
{
  string value;
}
```



```
public final class Quote
{
  public String value;
}
```

Attribute

- Maps to variable accessor and mutator methods
- In Java, maps to overloaded functions
  - Not JavaBean style get()/set(...) ☹
- Variables must be declared by developer
  - Not automatically generated by IDL compiler

```
string exchange;
```



```
String exchange();
void exchange(String arg);
```

Operations & Arguments

- Operation maps to a method
- Arguments for operations
  - Specify *direction*
    - IN (read in by method)
    - OUT (set by the method for return to caller)
    - INOUT (read and modified by the method)

```
Quote getQuote (in string symbol)
```



```
public Quote getQuote (String symbol)
```

Exception

- Maps to a Java exception
  - In IDL, no inheritance of exceptions
- Operation
  - *raises* instead of throws exceptions

```
Quote getQuote (in string symbol) raises UnknownSymbolException;
```



```
public Quote getQuote (String symbol)
                              throws UnknownSymbolException;
```

IDL to Java Mapping

Primitive Types:

IDL Type	Java Type
float	float
double	double
long , unsigned long	int
long long, unsigned long long	long
short, unsigned short	short
char, wchar	char
boolean	boolean
octet	byte
string, wstring	java.lang.String

IDL to Java Mapping

Complex Types:

IDL Type	Java Type
any	set of related classes
interface	set of related classes
sequence	array
struct	final class

Others:

IDL Type	Java Type
module	package
exception	exception class (inheriting from org.omg.CORBA.UserException)

Additional Notes

- IDL is case sensitive
  - Identifiers can't differ only by case

    boolean foobar

    interface FooBar
- No overriding or overloading of methods
  - Not all languages have these features
- Comments

  // comment

  /* comment */

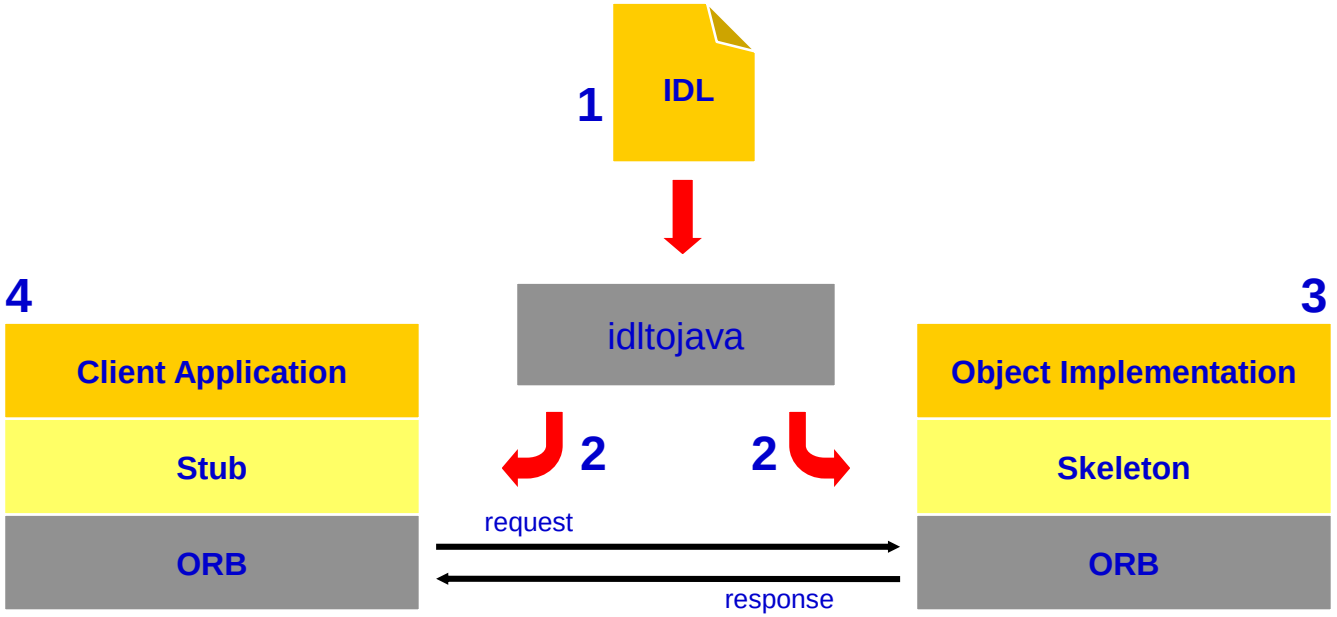
Developing CORBA Objects

- Define interface using IDL
- Process IDL to create *stub* and *skeleton* code
- Write code that implements the object (servant) and server to host it
- Write code that uses the object (client)

IDL Compilation

- IDL compilation
  - Generates code
    - Encapsulates underlying network code, marshalling
  - Complied to language dependent interfaces
- Stub (client side)
  - Proxy, reference to a “remote” object
- Skeleton (server side)
  - Manage interaction between proxy and server implementation

Development Steps



steps:

1	write the IDL file
2	compile with idltojava (stubs/skeleton generated automatically)
3	write object implementation (servant)
4	write client application

Object Adapters

Object Adapters

- Generate and interpret object references
- Activate and deactivate object implementations
- Handle method invocations via skeletons
- Basic Object Adapter (BOA)
- Portable Object Adapter (POA)

Portable Object Adapter (POA)

- Replaces BOA
  - Most commercial implementations still use BOA
- Expanded scope of OA to include
  - Activation policies
  - Threading models
  - Object life cycle (transient/persistent)
  - Pre/post invocation capabilities

Interoperable Object Reference (IOR)

- “Shareable” reference to a CORBA object
- Compatible with all CORBA-compliant ORBs
- Analogy: URL for object instances
- Location independent
  1) Save an IOR
  2) Go to another location
  3) Load the saved IOR
  4) Establish communication with the same object

CORBA Overview
ORB - Implementation Repository

- An Implementation Repository is responsible for locating and activating on demand registered CORBA servers
- An Implementation Repository stores a mapping from names of Object Adapters to servers address (host:port) along with scripts/batch files for starting the server if not running

The ORBcus IMR console is an ex. of GUI adm. of the IMR

CORBA Overview
ORB - Interface Repository

- The Interface Repository stores information about registered IDL Interfaces to clients and servers
  - names of methods
  - for each method the names and types of arguments and exceptions
  - the key is a IDL compiler generated unique identifier which is generated for each IDL type it compiles
- The Interface Repository is the fundament for reflection in CORBA
- The Interface Repository and the Dynamic Invocation Interface adds the power of reflection to CORBA

CORBA Overview
ORB - Interface Repository

- The Interface Repository can be access by both clients and servers
- The Interface Repository is often a autonom server/process which contain command line commands for feeding interface definitions into and deleting interface definitions from the Repository
- Example: ORBacus
  - irserv, irfeed, irdel

CORBA Overview
ORB - Portable Object Adapter

The component in the ORB architecture that maps the abstract concept of a CORBA Object onto concrete concepts provided by a specific prog. lang.

- An Object Adapter is responsible for
  - Generation and interpretation of IORs
  - Carrying out method invocations
  - Security on interactions
  - CORBA Object/implementation activation and deactivation
  - Mapping of IORs to corresponding object implementations
  - Registration of object implementations

Life cycle managment

CORBA Overview
ORB - Portable Object Adapter

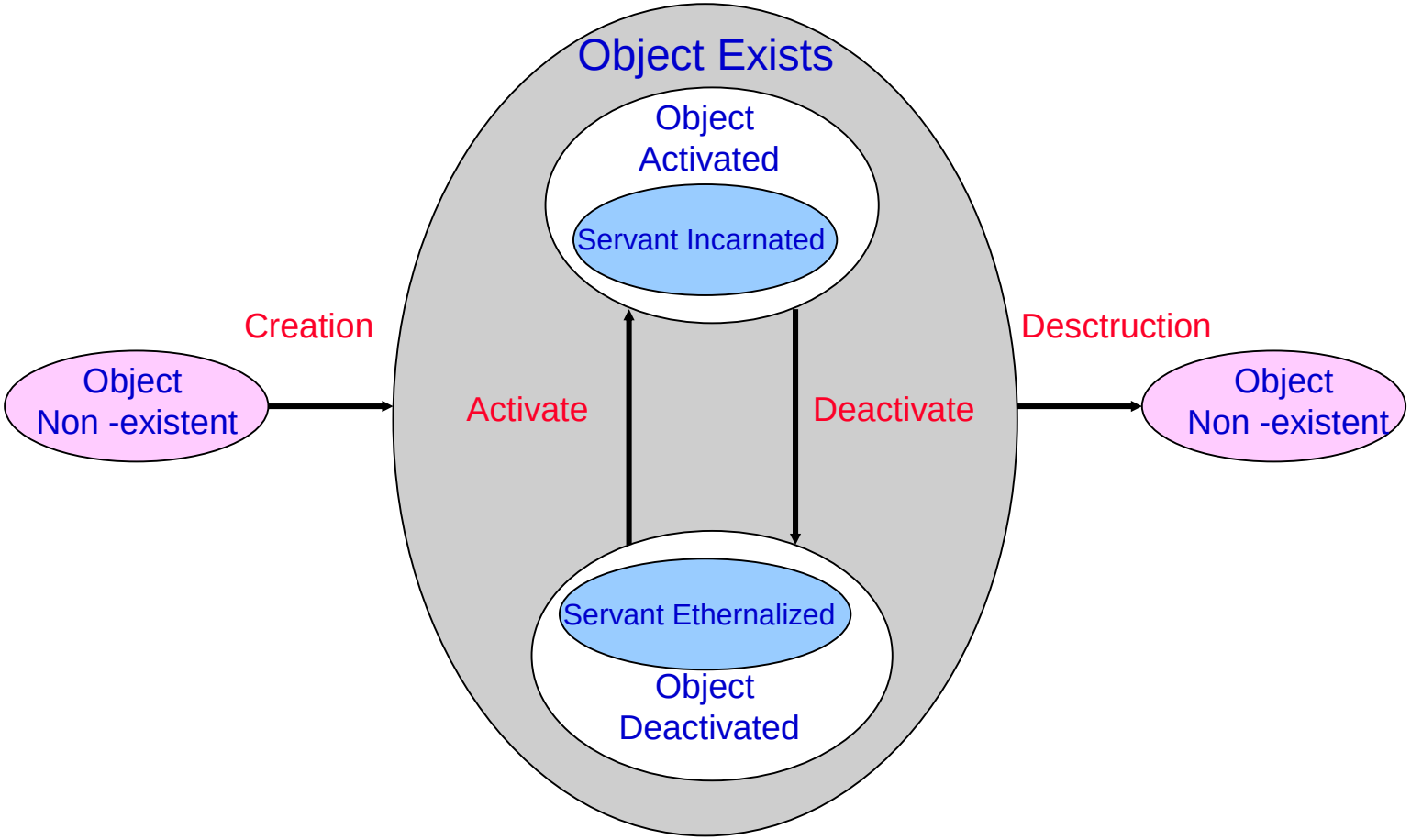
- Terminology
  - Servant: Implementation object providing the run-time semantic of one or more CORBA objects
  - ObjectID: An identifier, unique within a POA, that a POA uses to identify a CORBA Object
  - AOM: A table of Associations between ObjectIDs and servants
  - Incarnate: The action of providing a running servant to serve requests associated with a particular ObjectID
    - The POA will keep the association in AOM if configurated to do that

CORBA Overview
ORB - Portable Object Adapter

- Etherealize: The action of destroying the association between an ObjectID and a servant
- Activated: When a CORBA Object has been associated with a servant incarnating the object
- Deactivated: When a CORBA Object has no incarnating servant
- Default Servant: An object for which all incomming requests for ObjectIDs not in AOM are dispatched to
  - Only if configurated to do that

CORBA Overview

States of CORBA Object and Servant

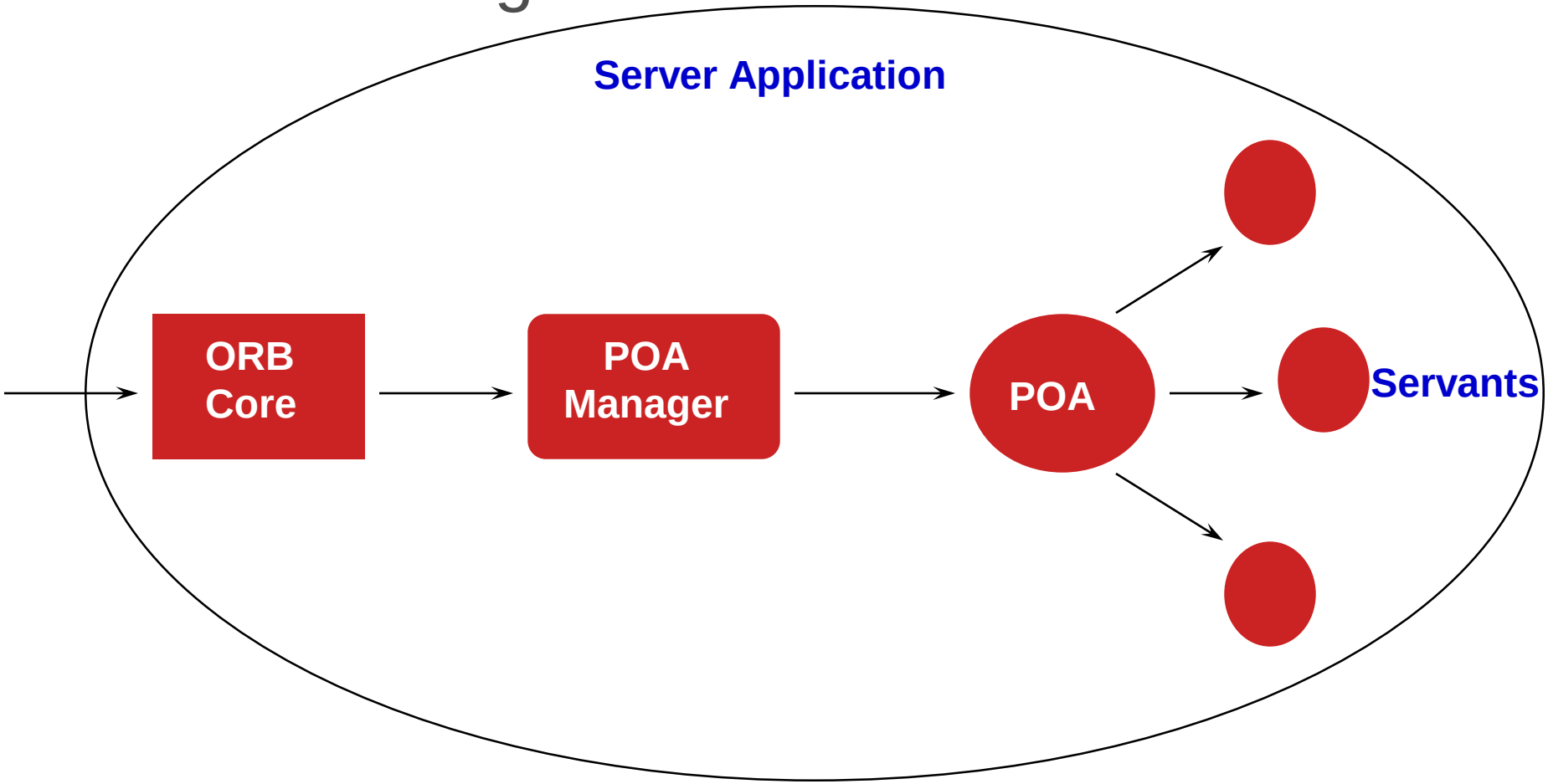


CORBA Overview
ORB - Portable Object Adapter

- An Object Adapter provides a public interface to object implementations
- An Object Adapter provides a private interface to the Skeleton
- An Object Adapter is build upon a private ORB-dependent interface to the ORB Core
- There are a variety of possible Object Adapters but it preferrable to use a few as possible as the object implementation is dependent upon them

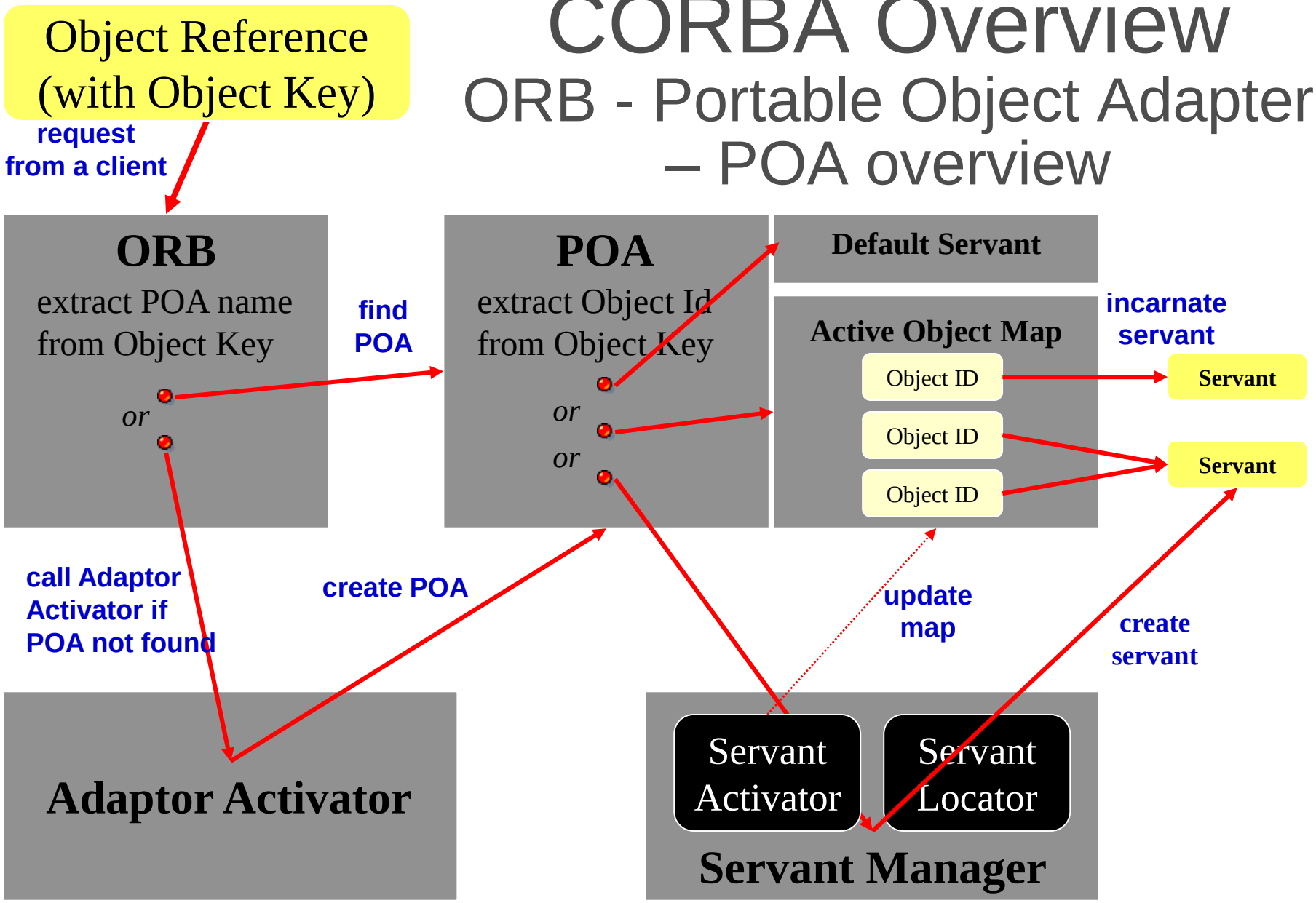
CORBA Overview

ORB - Portable Object Adapter - high-level architecture



CORBA Overview

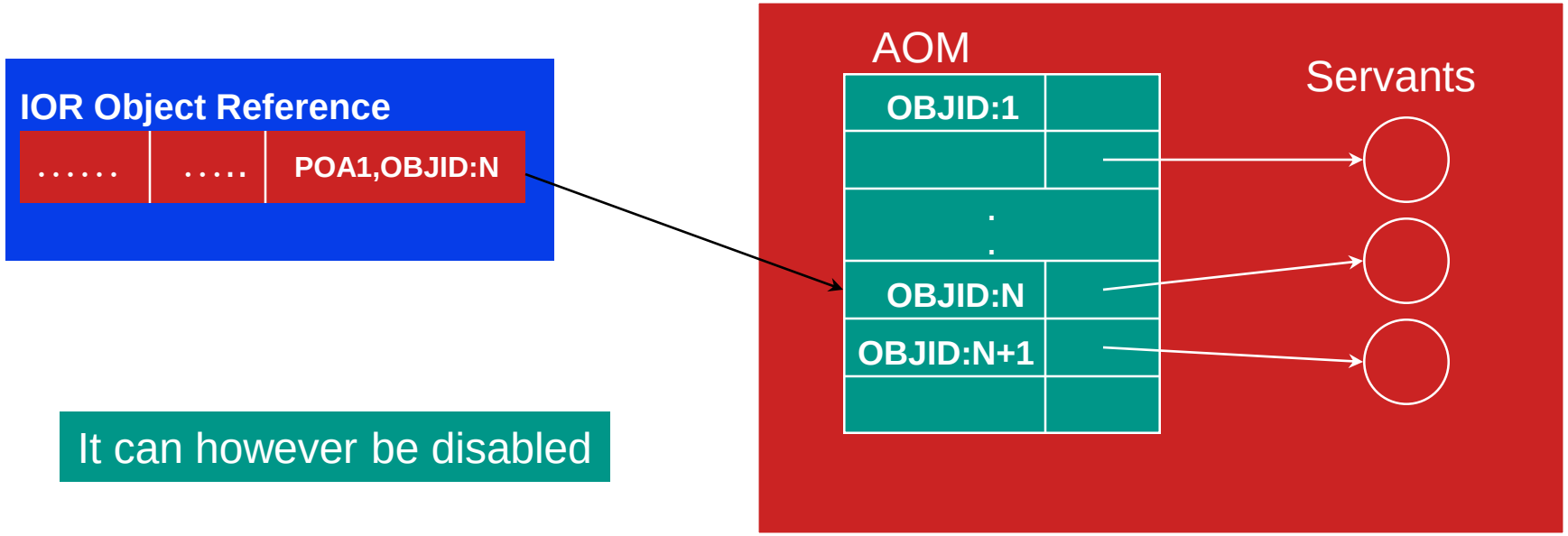
ORB - Portable Object Adapter – POA overview



CORBA Overview

ORB - Portable Object Adapter - Policies

Each POA maintain a *Active Object Map* (AOM) that map object IDs to servants



It can however be disabled

CORBA Overview
ORB - Portable Object Adapter - Policies

- Each POA is associated with 7 policies when its created – they are not changeable later
- The policies are not inherited from parent POA to its children POA upon creation
  - Given a set of standard policies
- Policies control implementation characteristics of the servants and object references
- Examples are
  - threading model for request dispatch
  - life time of references

CORBA Overview
ORB - Portable Object Adapter - Policies

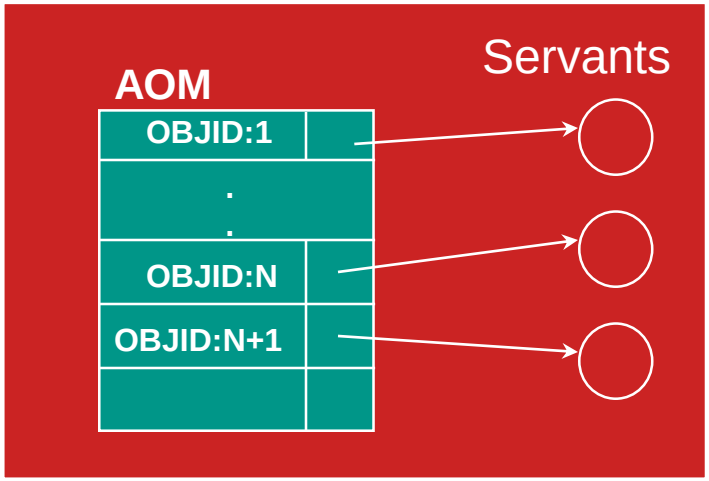
- LifespanPolicy
  - Transient object references
  - Persistent object references

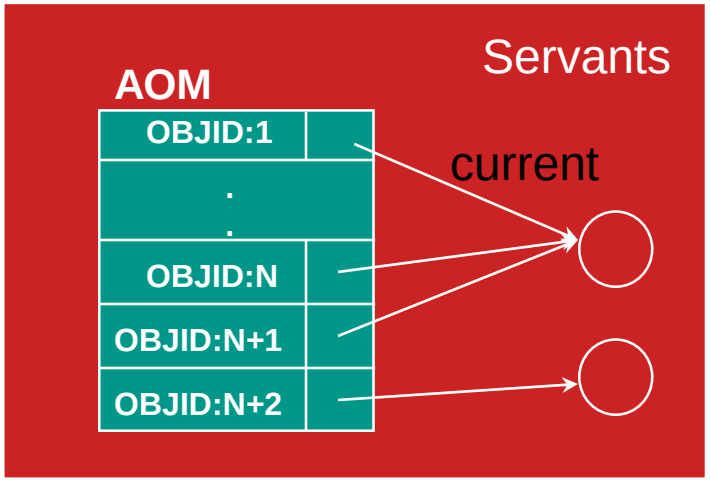
Detailed further in discussing the object model

- IdAssignmentPolicy
  - Object id provided by either the application or the system (USER_ID, SYSTEM_ID)
  - Persistent object references usually use IDs generated by the application (fx. Db primary key)
    - uses *activate_object_with_id* on POAs
  - Transient object references usually use IDs generated by the system

CORBA Overview
ORB - Portable Object Adapter - Policies

- IdUniquenessPolicy
  - How object references are mapped to servants
  - one servant for each Corba object (UNIQUE_ID)
  - one servant for more Corba objects (MULTIPLE_ID)





CORBA Overview
ORB - Portable Object Adapter - Policies

- Using MULTIPLE_ID offers you
  - scalability
  - possible slow access to servant state

Lets you develop the server as ‘Type’ based while clients see it as instance based!!!!!

- ImplicitActivationPolicy
  - Whether newly instantiated servants need be registered with the ORB (activation) manually or that happen automatically (NO_IMPLICIT_ACTIVATION, IMPLICIT_ACTIVATION)
  - Transient object references usually use implicit activation of servants
  - Persistent object references usually use explicit activation of servants

CORBA Overview
ORB - Portable Object Adapter - Policies

- RequestProcessingPolicy
  - Whether the POA uses static servant mapping (AOM) or servants are instantiated dynamically
  - Possible values
    - USE_ACTIVE_OBJECT_MAP_ONLY
    - USE_DEFAULT_SERVANT
    - USE_SERVANT_MANAGER
      - Servant Activator (RETAIN - servant for continues use)
      - Servant Locator (NON-RETAIN - servant for just the single operation - preInvoke()/postInvoke())

CORBA Overview
ORB - Portable Object Adapter - Policies

- ServantRetentionPolicy
  - Whether the POA keep the servants in memory all the time (RETAIN) or not (NON_RETAIN)
  - NON_RETAIN has to be used with USE_DEFAULT_SERVANT or USE_SERVANT_MANAGER RequestProcessing
  - If AOM the POA automatically calls a *default servant* or a *servant manager* if the requested object ID isn't in the AOM
  - This policy can be used to create the illusion of all objects running in the server - the *default servant* or *servant manager* just creates servants on request and maybe destroys them again

CORBA Overview
ORB - Portable Object Adapter - Policies

- ThreadPolicy
  - Whether the POAs processes request single threaded or whether the ORB chooses a threading model for request dispatch (ORB_CTL_MODEL, SINGLE_THREAD_MODEL)
  - Single-threaded means that all requests for that POA is serialized
  - If you choose to let the ORB decide you have to consult your ORBs documentation to see which threading the particular ORB practices

CORBA Overview
ORB - Portable Object Adapter - Policies

- ORBaCus provides more threading models
  - SingleThreaded Blocking (Client)
  - SingleThreaded Reactive (Client/Server)
  - Threaded (Client/Server)
  - Thread-per-Client (Server)
  - Thread-per-Request (Server)
  - ThreadPool (Server)
- Java ORB 1.4 doesn't support SingleThreaded model

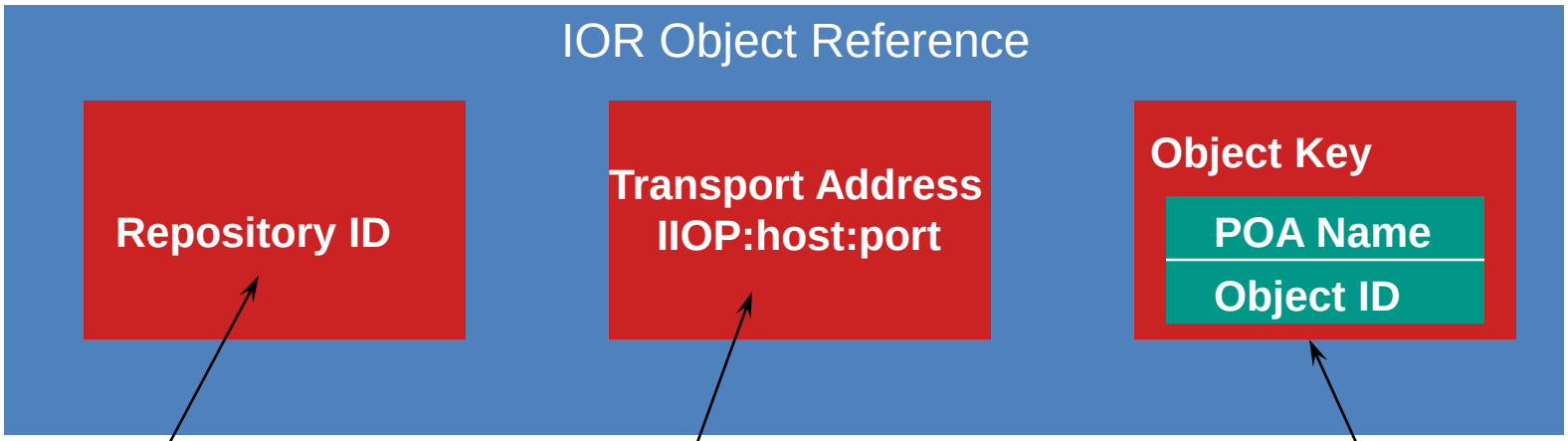
CORBA Overview
ORB - Portable Object Adapter - Policies

- Default values for the 7 policies for the RootPOA
  - transient
  - system_id
  - unique_id
  - retain
  - use_active_object_map_only
  - implicit_activation
  - orb_ctl_model

CORBA Overview
ORB - Portable Object Adapter - Policies

- Standard policies for POA creation
  - transient
  - system_id
  - unique_id
  - retain
  - use_active_object_map_only
  - no_implicit_activation
  - orb_ctl_model

CORBA Overview
Object Model - remote object references



Identifies Interface Type

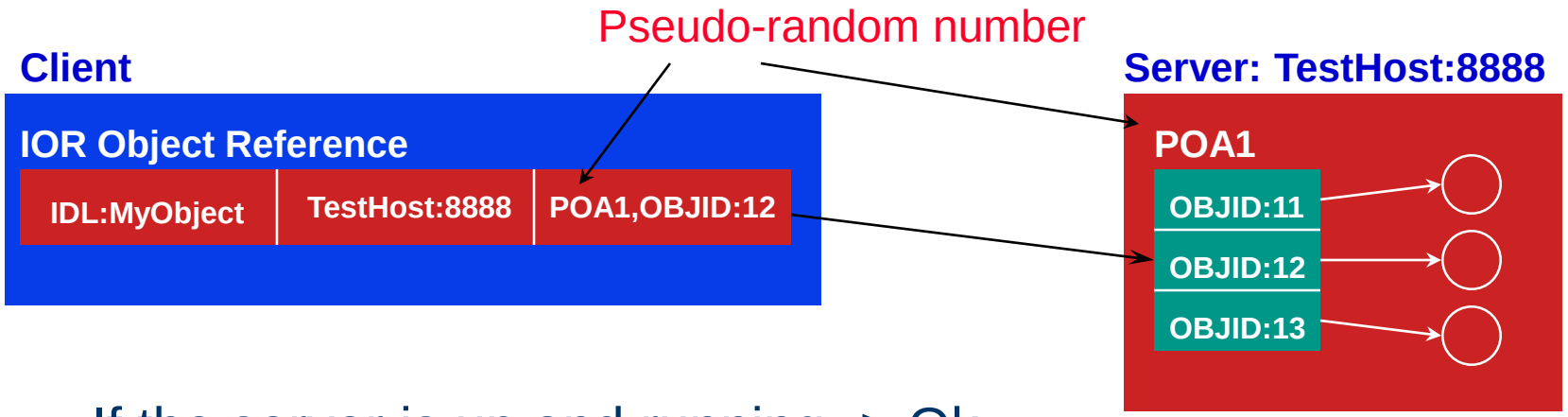
Identifies a transport endpoint

Identifies the object within POA

Is in proprietary format for a given ORB

Interoperability is not affected as the server is the only one looking in it

CORBA Overview
Object Model - remote object references



If the server is up and running -> Ok

If the server is down - > OBJECT_NOT_EXIST

If the server is running but not the right adapter ID (check for pseudo-random number) -> OBJECT_NOT_EXIST

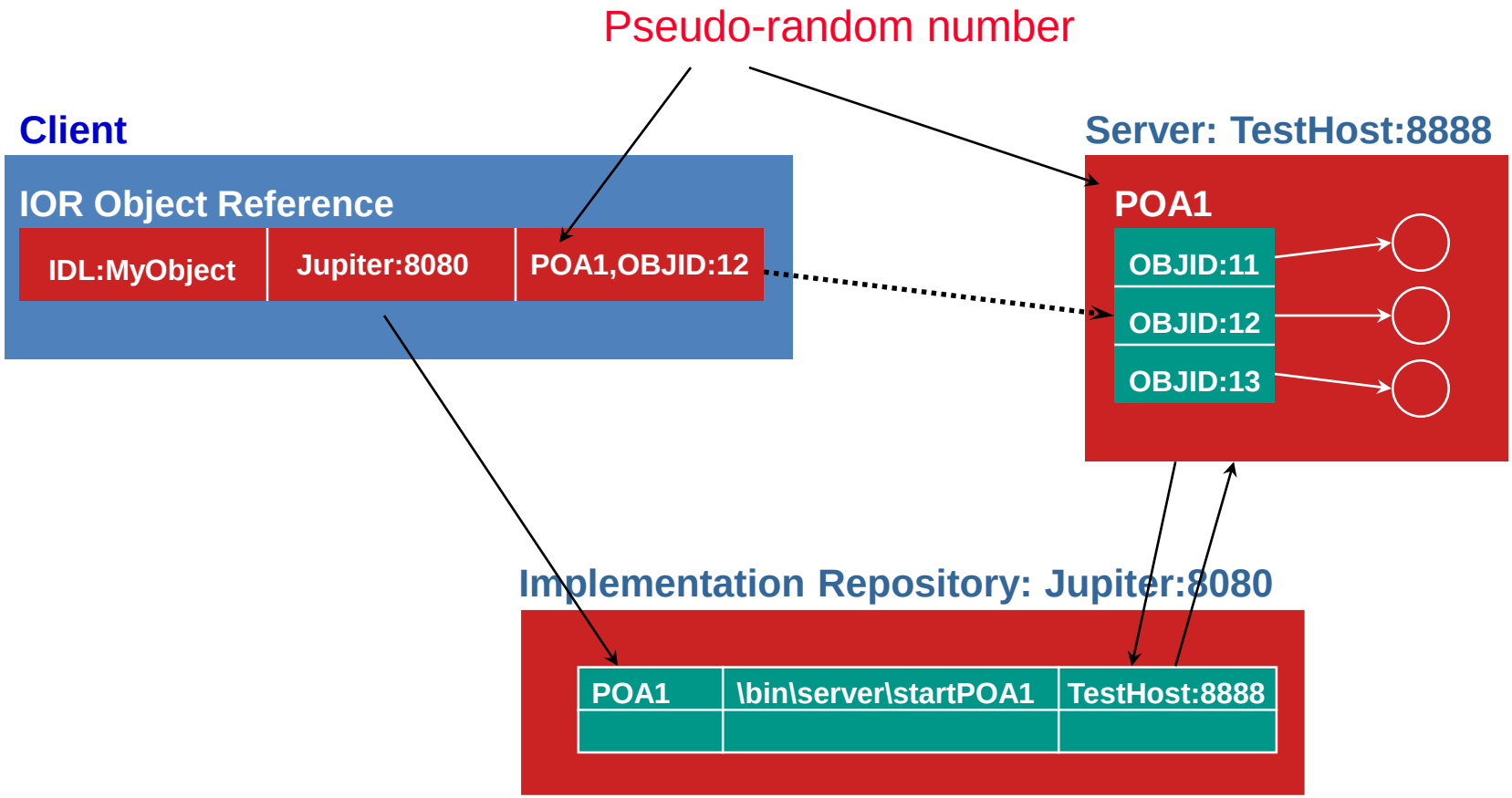
If the server is running but not the right ORB (check for vendor specific tag in IOR identifying ORB) ->OBJECT-NOT_EXIST

CORBA Overview
Object Model - remote object references

- Persistent object references are implemented by usage of the Implementation Repository
- IOR *Host:port* contains the Implementation Repository server process information
  - More host:port occurences allow for replicated Implementation Services
- Implementation Repository acts as a level of indirection and delivers at runtime the address of the POA server process to the client

CORBA Overview

Object Model - remote object references



Next Time
CORBA Part II

- CORBA Services
- CORBA Naming Service
- CORBA Transaction Service
- CORBA Concurrency Service
- RMI/IIOP
- CORBA 3.0 - Whats new?