TDD25
Distributed Systems

Distributed Heterogeneous Applications and CORBA

Christoph Kessler

IDA
Linköping University
Sweden

Agenda

Distributed Heterogeneous Applications and CORBA

1. Heterogeneity in Distributed Systems
2. Middleware
3. Objects in Distributed Systems
4. The CORBA Approach
5. Components of a CORBA Environment
6. CORBA Services

Heterogeneity in Distributed Systems

Distributed applications are typically **heterogeneous**:

- **different hardware**: mainframes, workstations, PCs, servers, etc.;
- **different software**: UNIX, Windows, IBM OS/2, Real-time OSs, etc.;
- **unconventional devices**: teller machines, telephone switches, robots, manufacturing systems, etc.;
- **diverse networks and protocols**: Ethernet, wireless, FDDI, ATM, TCP/IP, UDP, HTTP, etc.

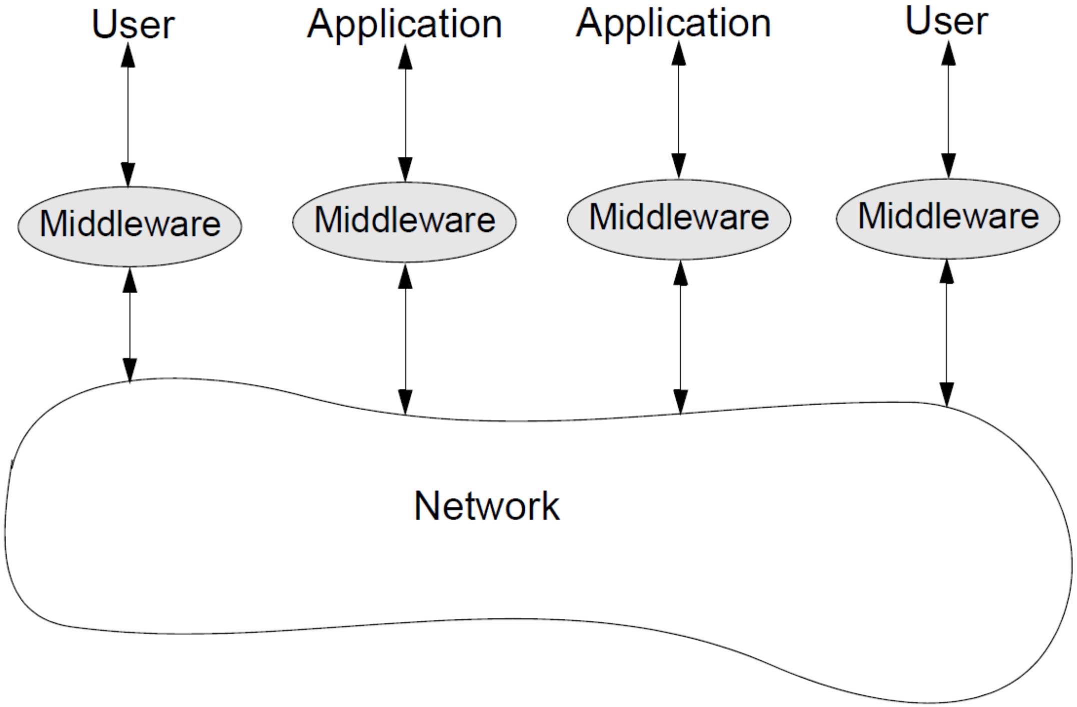
Why?

- Different hardware/software solutions are considered to be optimal for different parts of the system.
- Different users who have to interact are deciding for different hardware/software solutions/vendors.
- Legacy systems.

Middleware

A key component of a heterogeneous distributed client-server environment is **middleware**.

- Middleware is a set of **services** that enable applications and end users to interact with each other across a heterogeneous distributed system.
  - Middleware software resides above the network and below the application software.



Middleware

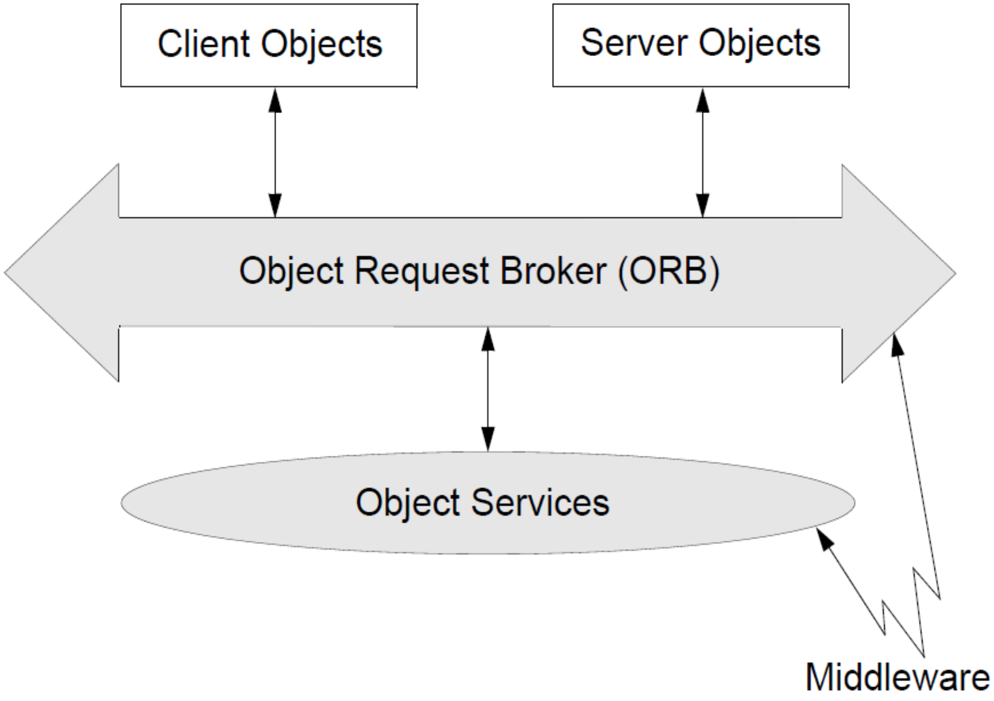
- Middleware should make aspects of a heterogeneous system, here the network, **transparent** to the applications and end users

  → Users and applications should be able to perform the same operations across the network that they can perform locally.
- Middleware should hide the details of computer hardware, OS, software components across networks.
- Different kind of software qualifies, to certain extent, as middleware, for example:
  - File-transfer packages (FTP) and email;
  - Web browsers;
  - CORBA

Remark: Middleware software also exists for other purposes than *network* abstraction, e.g. for system-independent message passing (MPI), portable CPU performance counter access (PAPI), etc., see TDDE65. Also the Java Virtual Machine (JVM) is a middleware. CORBA also abstracts from the client/server *programming language*, not only the network. Indeed, an important use case of CORBA is accessing legacy software/hardware systems.

Objects in Distributed Systems

- A distributed application can be viewed as a collection of **objects** (user interfaces, databases, application modules, customers).



Object:

- data surrounded by code;
- has its own attributes and methods which define the behavior of the object;
- objects can be clients, servers, or both.

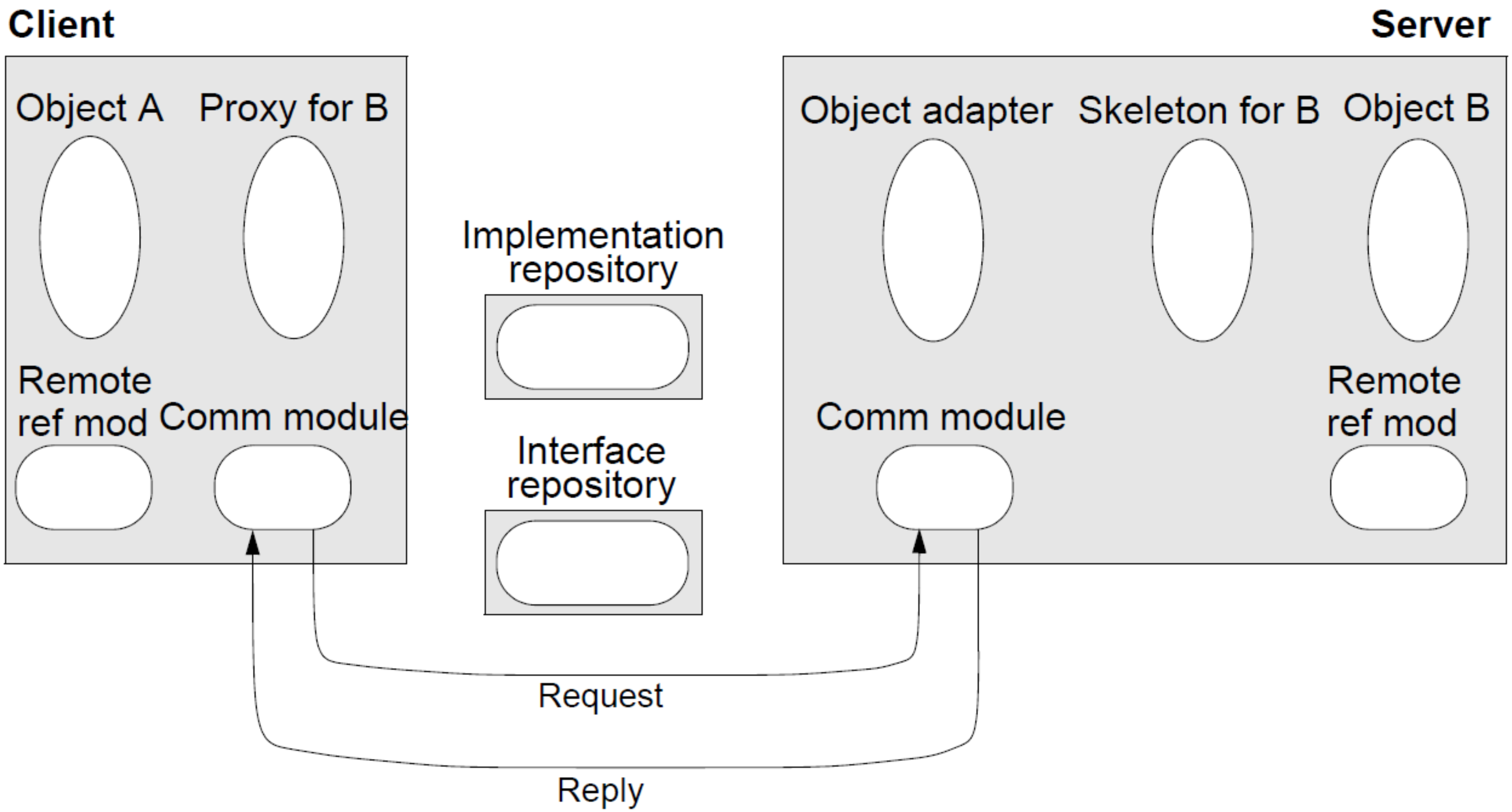
Object broker:

- allows objects to **find** each other and **interact** over a network;
- they are the backbone of the distributed system.

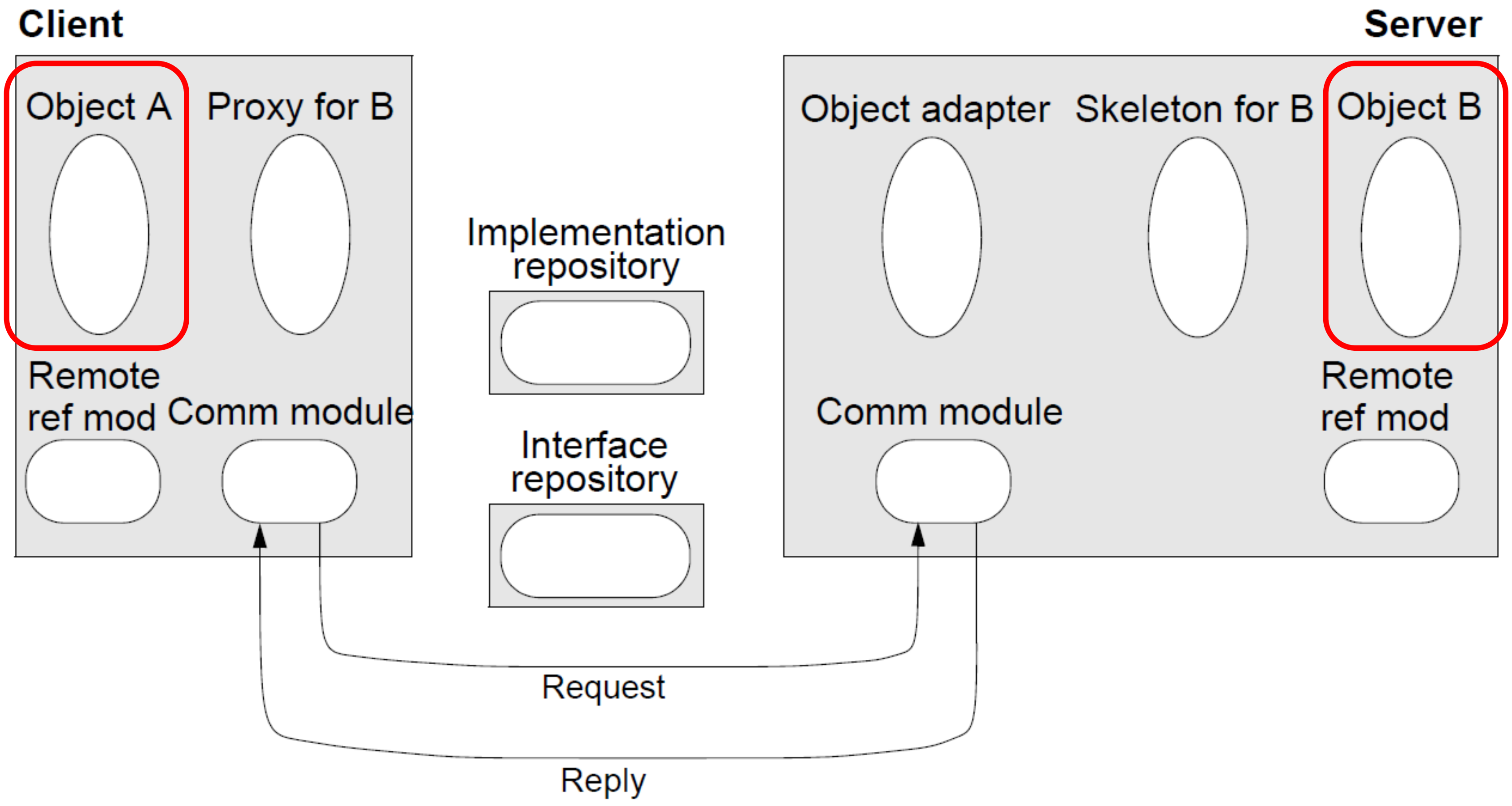
Object services:

- allow to create, name, move, copy, store, delete, restore, and manage objects.

RMI: Objects in Distributed Systems

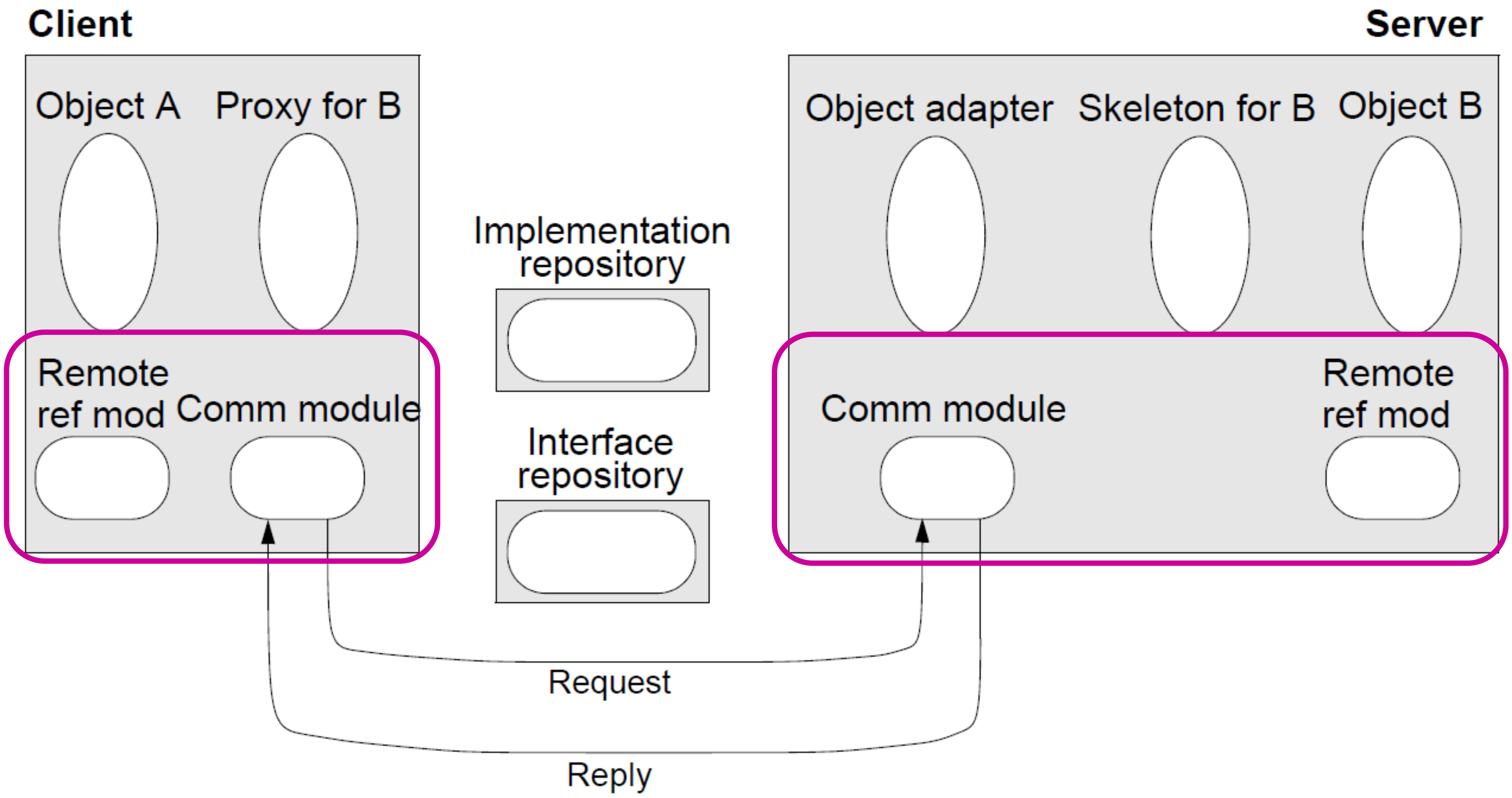


RMI: Objects in Distributed Systems



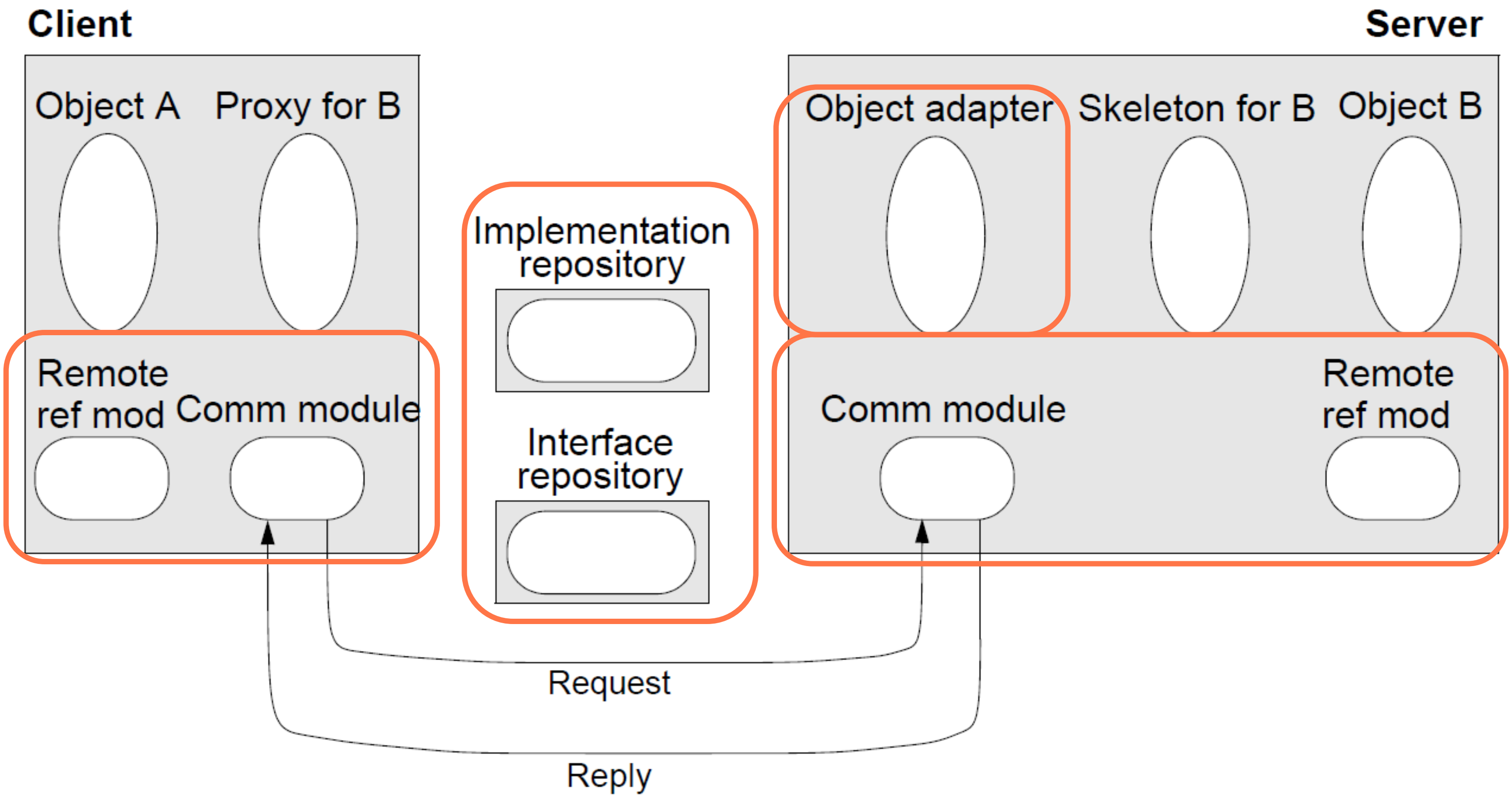
Distributed Application

RMI: Objects in Distributed Systems



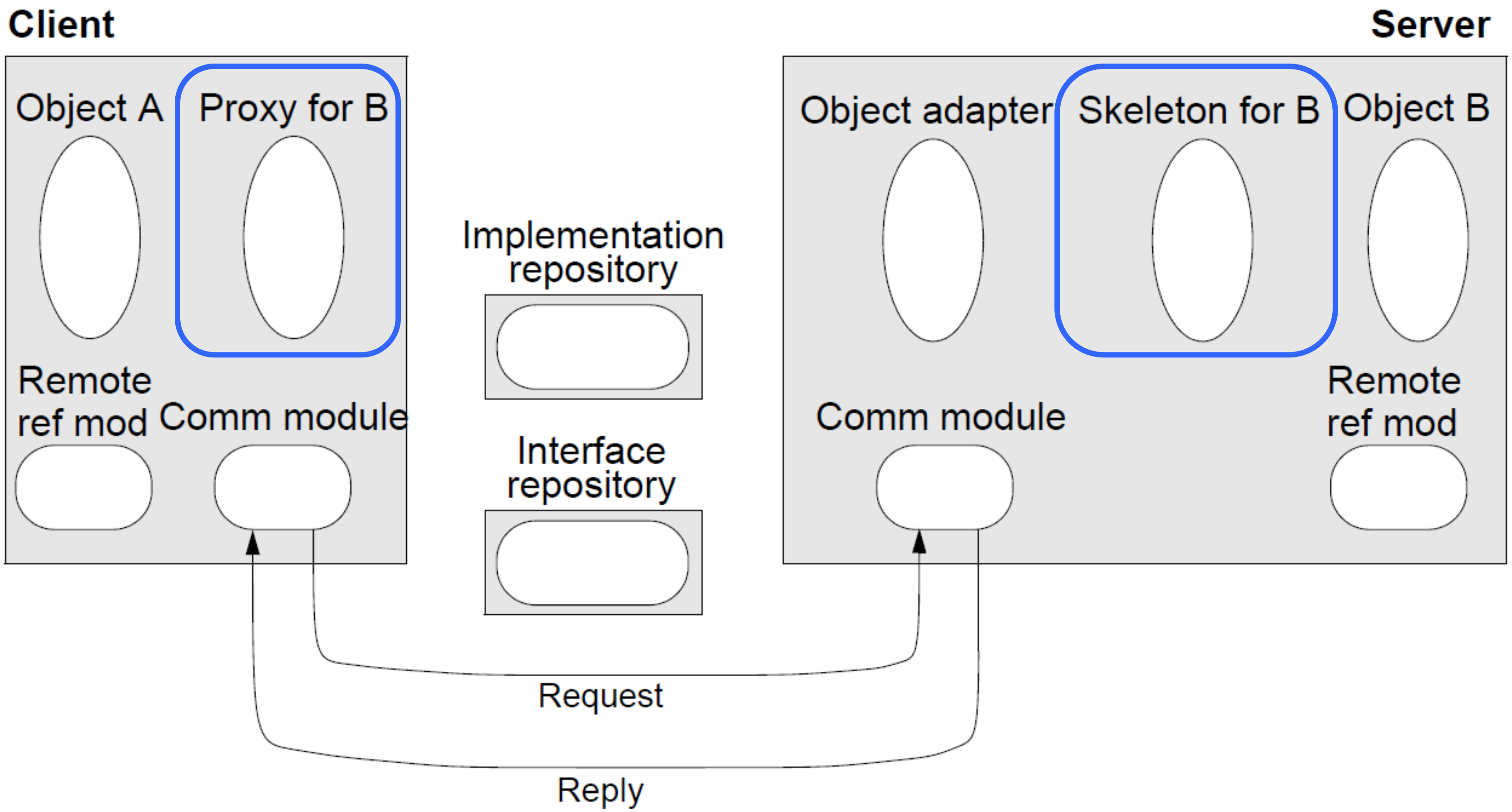
Object Request Broker (ORB)

RMI: Objects in Distributed Systems



Middleware

RMI: Objects in Distributed Systems



Border between Application and Middleware (Proxy objects, Glue code)

Interface Definition Language

An **interface** specifies how the clients can invoke operations on objects (regardless if server-side or local):

- the set of operations (methods)
- For each operation,
  - the input parameters with their data types
  - the return parameters with their data types
  - exceptions (= special return parameters indicating errors) where applicable

Interfaces are defined by using an **interface definition language (IDL)**.

- **CORBA IDL** is an example of such a language.

IDLs are **declarative languages**; they do not specify any executable code, but only declarations.

Interface Definition Language

- Middleware products (such as CORBA) provide interface compilers that parse the IDL description of the interface. An **IDL compiler** produces the following **code**:
    - classes corresponding to the stubs / proxies (in the programming language of the client)
    - classes corresponding to the skeletons (in the programming language of the server).
- Language mappings have to be defined which allow to generate proxies and skeletons in the implementation languages of the clients and of the server respectively.

CORBA

- **Object Management Group** (**OMG**):
  a non-profit industry consortium formed in 1989 with the goal to develop, adopt, and promote **standards** for the development of distributed heterogeneous applications.
  - https://www.omg.org/
  - One of the main achievements of OMG is the specification of a **Common Object Request Broker Architecture** (**CORBA**).

- The **CORBA** specification details the **interfaces** and **characteristics** of the Object Request Broker:
  - It **specifies** a set of **middleware functions (API)** which allow objects to communicate with one another no matter where they are located, who has designed them, and in which language they are implemented.
  - **OMG only provides a specification**; there are several products which, to a certain extent, implement the OMG specification.

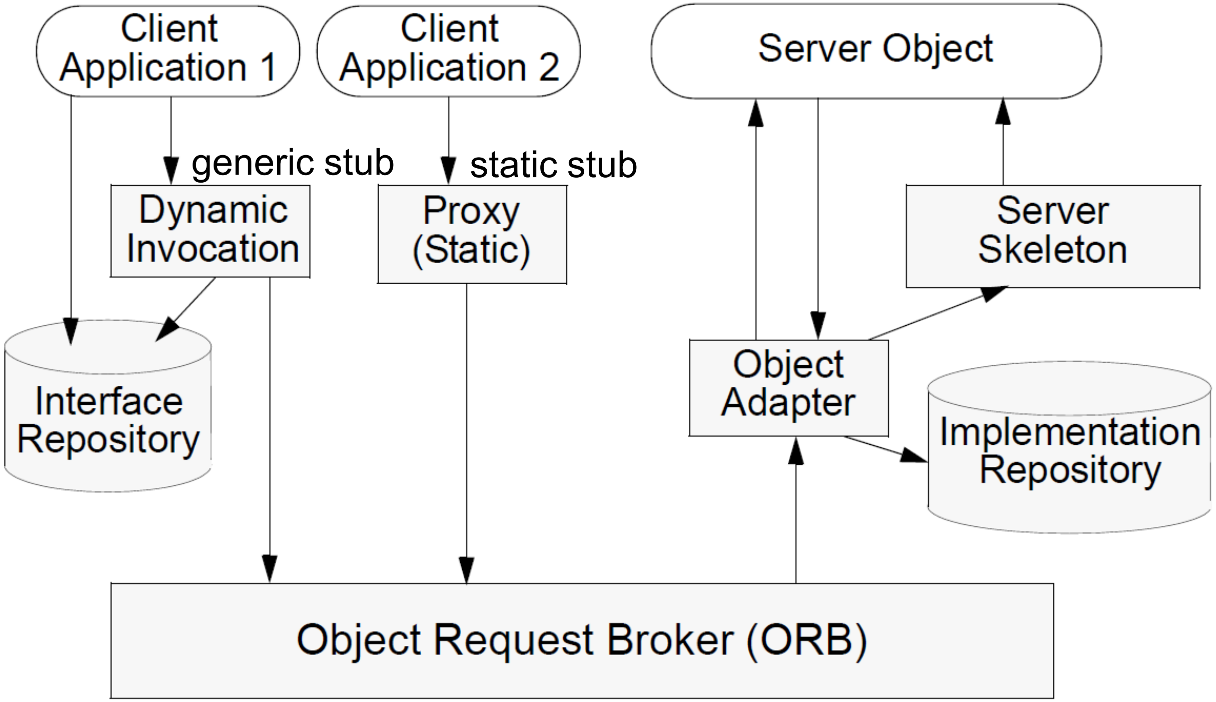
CORBA

Key concepts:

- CORBA specifies the middleware services used by application objects.
- An object can be a client, a server or both.
- Object interaction is through **requests**:
  - The information associated with a request is
    1. an **operation** to be performed
    2. **a target object**
    3. zero or more **arguments** (that match the operation's IDL type signature)
  - CORBA supports static as well as dynamic binding
    - Dynamic binding between objects uses a generic stub for run-time identification of callee objects based on argument types.
- The **interface** represents the **contract** between client and server;
  - to be written for each callable server class in CORBA IDL
  - proxies and skeletons (client and server stubs) are generated as result of IDL compilation.
- CORBA objects do not know the underlying implementation details; an **object adapter** maps the generic model to a specific implementation.

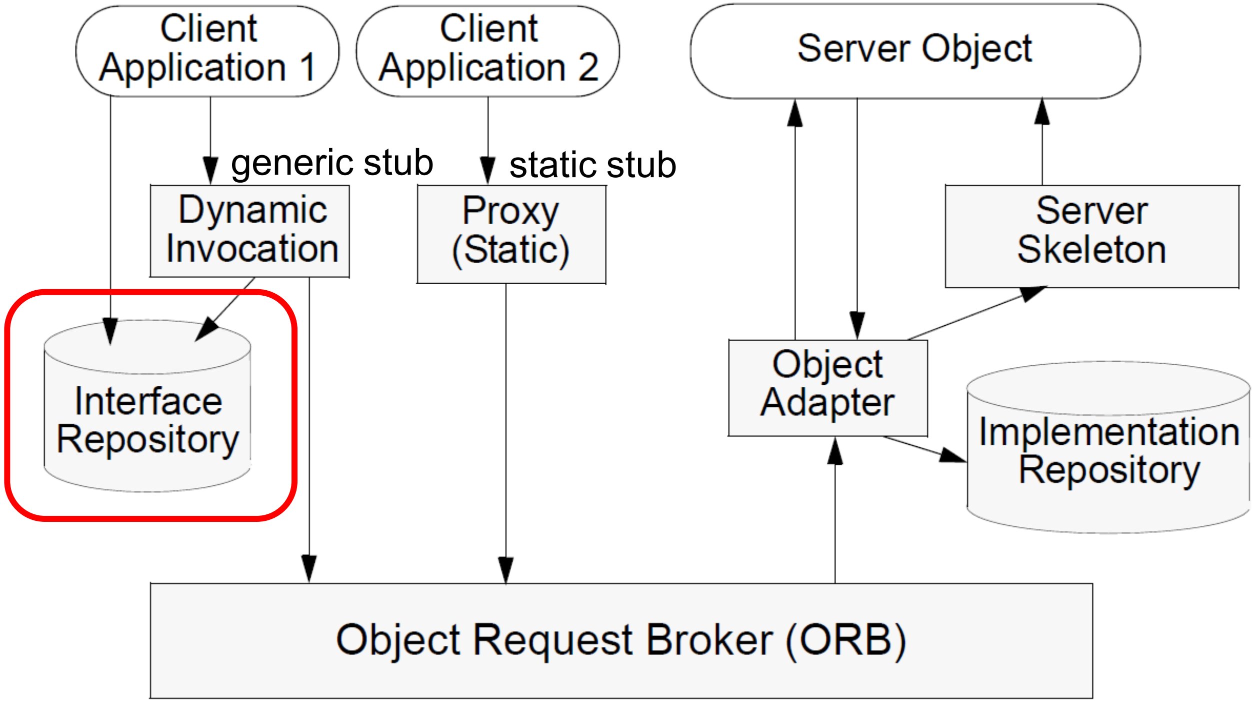
CORBA

Components of a CORBA environment:



CORBA

Components of a CORBA environment:

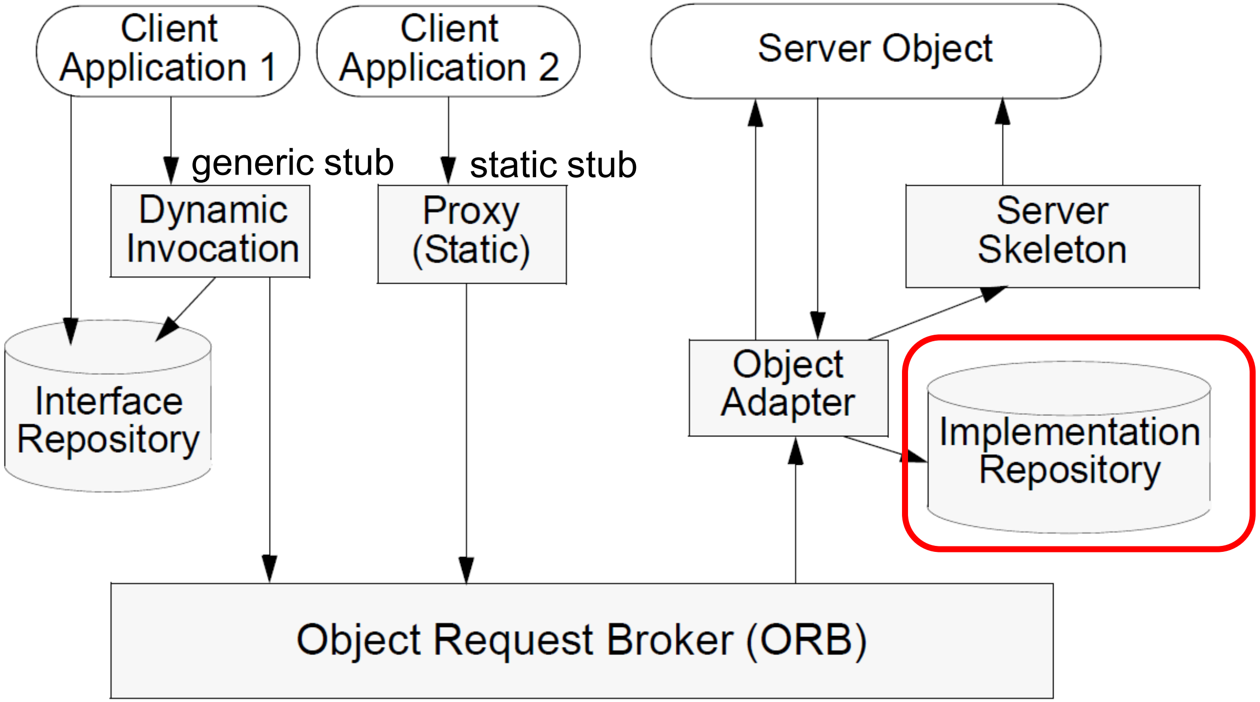


Interface Repository

- Provides a representation of interfaces for all server objects in the system. It corresponds to the server objects' IDL specification.
- Clients can access the repository to learn about server objects, the types of operations which can be invoked and the corresponding parameters.
- This is used for **dynamic invocation of objects**.

CORBA

Components of a CORBA environment:

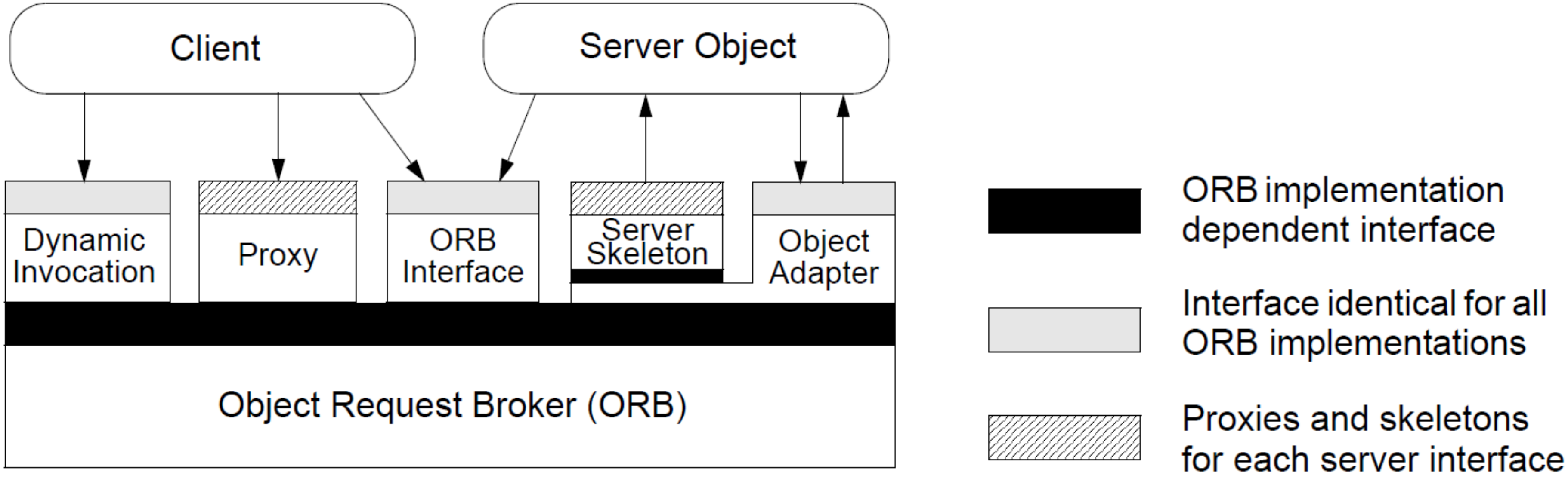


Implementation Repository

- Stores implementation details for the **objects** implementing each interface
  - the main information is a mapping from the server object's name to the binary file name which implements the respective service;
- the implementation repository is used by the **object adapter** (generic server-side entry point) to solve an incoming call and activate the right method (via a skeleton).

The Object Request Broker (ORB)

- **ORB and its interfaces:**



Client
Server Object
Dynamic Invocation
Proxy
ORB Interface
Server Skeleton
Object Adapter
Object Request Broker (ORB)
ORB implementation dependent interface
Interface identical for all ORB implementations
Proxies and skeletons for each server interface

The ORB, through its interfaces, provides mechanisms by which objects transparently interact with each other.

- Issuing of a request can be dynamic or static; it is performed through the client stubs (proxies) or through the dynamic invocation interface (generic stub). →
- Invocation of a specific server method is performed by the server skeleton which gets the request forwarded from the **object adapter**.
- The **ORB interface (API)** can be accessed **directly** by application objects for **services** like directory, naming, manipulation of object references.

Static and Dynamic Invocation

CORBA allows both static and dynamic invocation of objects.

- The choice is made depending on how much information, concerning the server object, is available at compile time.

Static Invocation

- Static invocation is based on compile time knowledge of the server's interface specification. This specification is formulated in IDL and is **compiled** into a proxy (client stub) code in the same programming language in which the client object is encoded.
- For the client, an object invocation is like a local invocation to a proxy method. The invocation is then automatically forwarded to the object implementation through the ORB, the object adapter and the skeleton.
- Static invocation is **efficient** at run time, because of the relatively low overhead.

Static and Dynamic Invocation

Dynamic Invocation

- Dynamic invocation allows a client to invoke requests on an object without having compile-time knowledge of the object's interface.
- The object and its interface (methods, parameters, types) are detected at run-time.
- The **dynamic invocation interface** (DII) allows to inspect the interface repository and **dynamically construct invocations** corresponding to the server's interface specification.
  - It is a **generic** stub, like an **interpreter** in contrast to to the compiled fixed-function stub for static calls
- The execution **overhead** of a dynamic invocation is **huge**.
- Once the request has been constructed and arguments placed, its invocation has the same effect as a static invocation.
  - From the server's point of view, static and dynamic invocation are identical; the server does not know how it has been invoked.
  - The server invocation is always issued through its skeleton, generated at compile time from the IDL specification.

CORBA Services

Goal:

- Provide a **portable** execution environment with **standardized**, **reusable** functionality atop heterogeneous hardware and system software
- Avoid that programmers hardcode their own solution (no reuse), even in a platform-specific way ☹

The following → **services**
have been specified in the OMG CORBA **standard**
(however, some products only implement part of them):

- **CORBA Naming Service**
- **CORBA Trader Service**
- **CORBA Transaction Management Service**
- **CORBA Concurrency Control Service**
- **CORBA Security Service**
- **CORBA Time Service**.
- and others

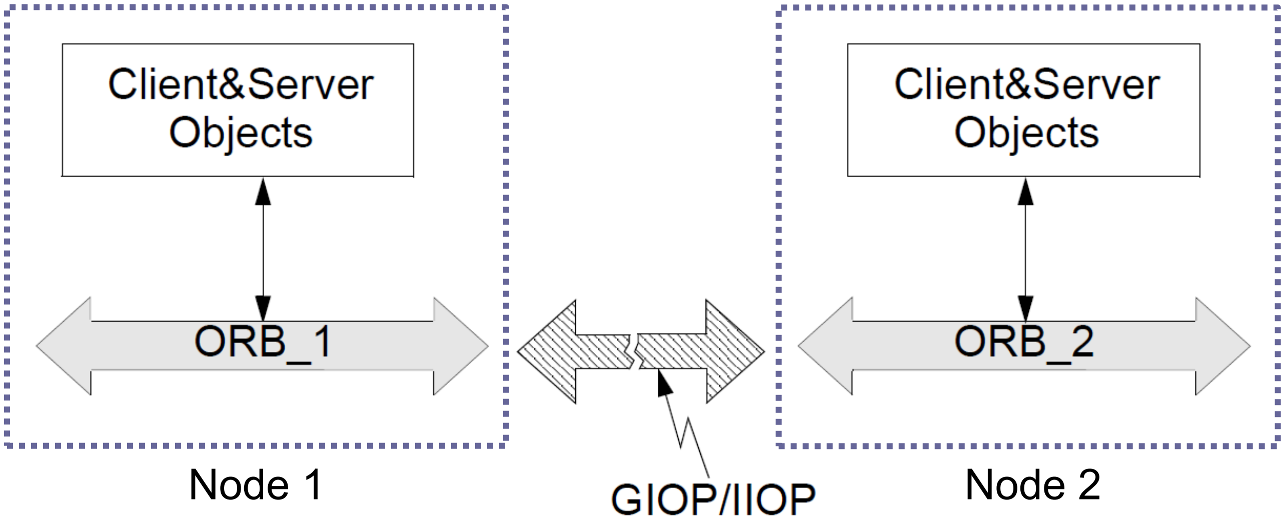
CORBA Services

- **CORBA Naming Service and Trader Service**:
  - The basic way an object reference is generated is at creation of the (server) object when the reference is returned (to the client calling the constructor).
    - An **interoperable object reference** contains, in particular, the referenced object's server IP address and port number
    - Can be passed around to other nodes and called from there
  - Object references can be stored together with associated information (e.g. names and properties).
  - The **naming service** allows clients to find objects based on names.
  - The **trader service** allows clients to find objects based on properties.
- **CORBA Transaction Management Service**: provides two-phase commit coordination among recoverable components using transactions.
- **CORBA Concurrency Control Service**: provides a lock manager that can acquire and free locks for transactions or threads.
- **CORBA Security Service**: protects components from unauthorized users; it provides authentication, access control lists, confidentiality, etc.
- **CORBA Time Service**: provides interfaces for synchronizing time; provides operations for defining and managing time-triggered events.
- ...

Inter-ORB Architecture

Implementations of ORBs differ from vendor to vendor

→ how do we solve interaction between objects running on different CORBA *implementations*?



General Inter-ORB Protocol (GIOP): (defined in CORBA 2.0)
GIOP specifies a set of message formats and portable common data representations for interactions between ORBs and is intended to operate over **any** connection-oriented transport protocol.

- **Internet Inter-ORB Protocol (IIOP):** IIOP is a particularization of GIOP; it specifies how GIOP messages have to be exchanged over a **TCP/IP** network.

Additional Material on CORBA

- Coulouris et al., “Distributed Systems – Concepts and Design” (5th edition), Chapter 8
  or:
- Tanenbaum, van Steen: Distributed Systems. Chapter 9.

- Extra slide set on CORBA with Java example code
  - for background reading
  - on the course web page

- CORBA documentation by OMG https://www.omg.org/

The Legacy of CORBA

- CORBA has influenced many later frameworks for ***portable* RMI abstraction**
  - **Java RMI** (language-specific) – slow
  - **Enterprise Java Beans** (**EJB**) (language-specific)
    - Heavyweight, replaced by **Spring** framework, but still in use
  - **RESTful services** (Representational State Transfer) – text-based (XML, JSON, ...) client-server communication API
    - Example: **Web services**
      - No object abstraction as in CORBA
      - Interfaces, data type specifications, and portable request and reply messages are all encoded in XML atop HTTP and parsed/interpreted at runtime
    - Very high overheads
    - Limited middleware beyond communication (fewer portable services)
  - ...
- The runtime efficiency of CORBA static calls with compiled stubs/skeletons has never been matched by these.
- CORBA is still in use today for interfacing to legacy SW/HW systems

Acknowledgments

- Most of the slide contents is based on a previous version by Petru Eles, IDA, Linköping University.