Lecture 16

CORBA

What is CORBA ?

- CORBA = Common Object Request Broker Architecture
- An extensive open standard for building *object-oriented distributed systems*, defined by the Object Management Group, that include middleware services.
- www.omg.org - OMG provides specifications not implementations.

- It is important to note that CORBA objects differ from typical programming objects in three ways:
  - CORBA objects can run on any platform;
  - CORBA objects can be located anywhere on the network;
  - CORBA objects can be written in any language that has IDL mapping.

Purpose of CORBA

- Using the standard protocol IIOP, two CORBA-complaint programs from any vendor, running on two separate computers, operating system, programming language, and network, can inter-operate.

- One of its most important uses is in servers that must handle large number of clients, at high rates, with high reliability.

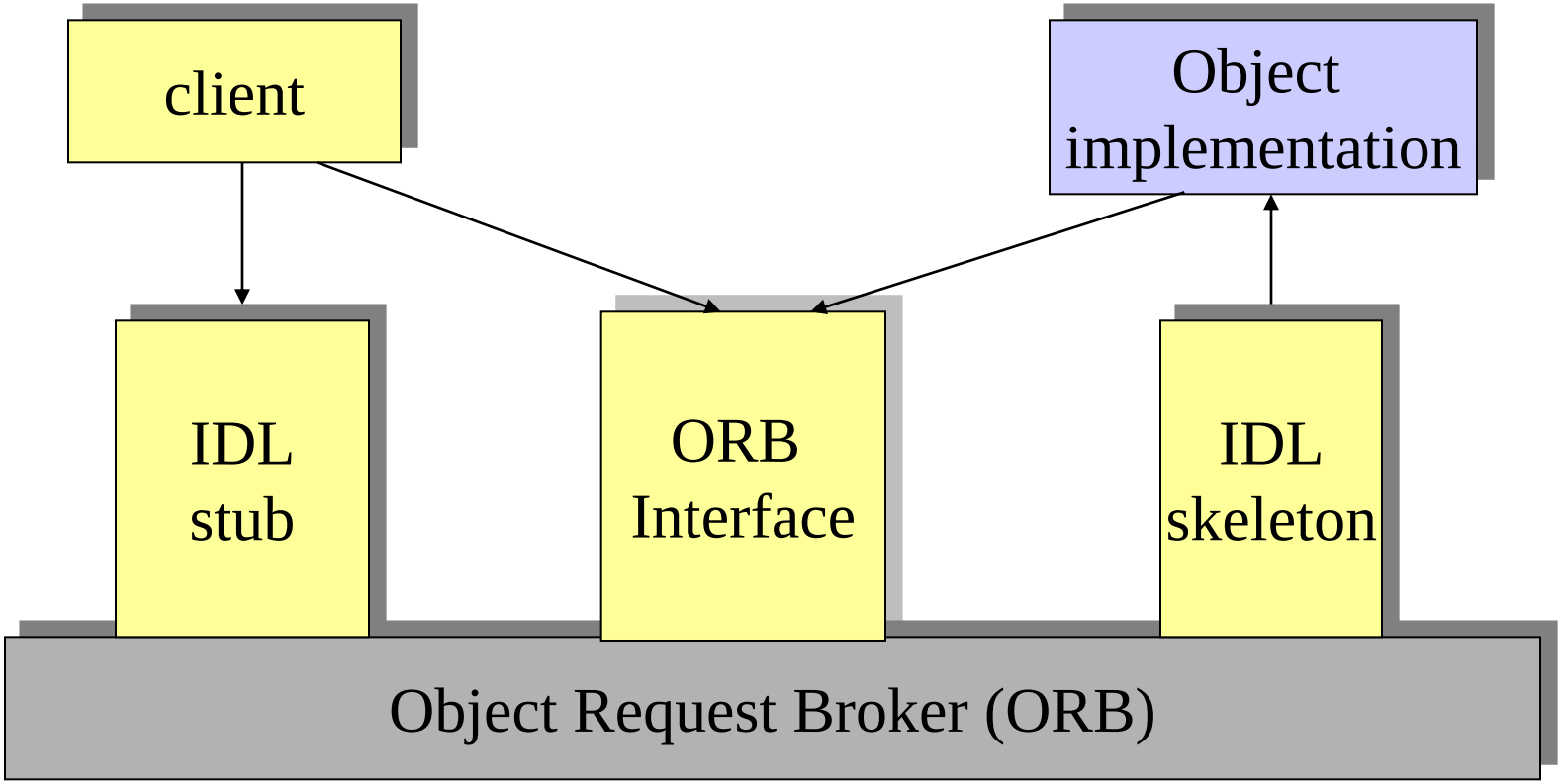
- Specialized versions of CORBA also run real-time systems, and small embedded systems.

CORBA model

- A *CORBA object* is an entity that provides the operations defined in its *IDL interface* and has *an object reference*. The operations are *always* available to the clients, from the time the object instance is created until the time it is destroyed. There might be many instances of the same object type at a time (e.g., shopping cart).

- The *separation of interface from implementation*, enabled by OMG IDL, is the essence of CORBA - it enables interoperability. The *implementation* of an object - its running code, and its data - is hidden from the rest of the system (that is, *encapsulated*) behind a boundary that the client may not cross. Clients access objects only through their advertised interface, invoking only those operations that the object exposes through its IDL interface, with the corresponding parameters (input and output).

Architectural view



An object request broker (ORB) provides the interface between the client and the remote object – servant. In this figure, all run within the same process.

- In CORBA, every object instance has its own *unique* *object reference*, an identifying electronic token. Clients use the object references to direct their invocations, identifying to the ORB the exact instance they want to invoke.

- The client acts as if it's invoking an operation on the object instance, but it's actually invoking on the IDL stub which acts as a proxy. Passing through the stub on the client side, the invocation continues through the ORB (Object Request Broker), and the skeleton on the implementation side, to get to the object where it is executed.

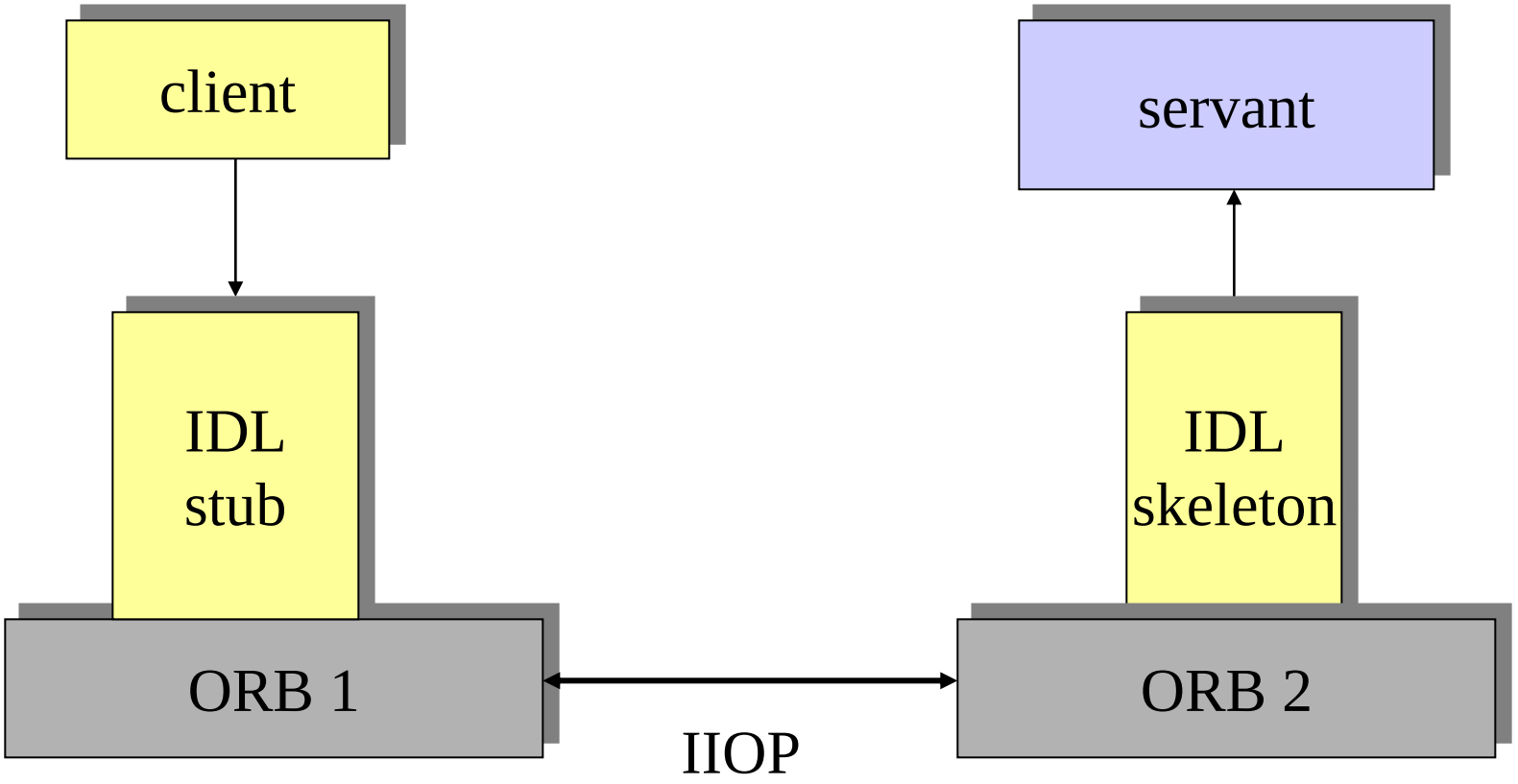
Advanced features

- *Activation* denotes that a CORBA server has allocated and configured CPU and memory resources to perform the operations of the object instance; *deactivation* denotes that these resources have been reclaimed. If the object has state, that state must be saved to *persistent storage* on deactivation and restored from storage on subsequent activation. Because this happens transparently, the client is unaware that a deactivation/activation cycle has occurred.

- Object activation and deactivation are visible to the server only.

- The servant, visible only to the server, is the executing CPU and memory resource that perform an object's operation. The association between a servant and an object is a run-time concept, and may be set as frequently as per-invocation.

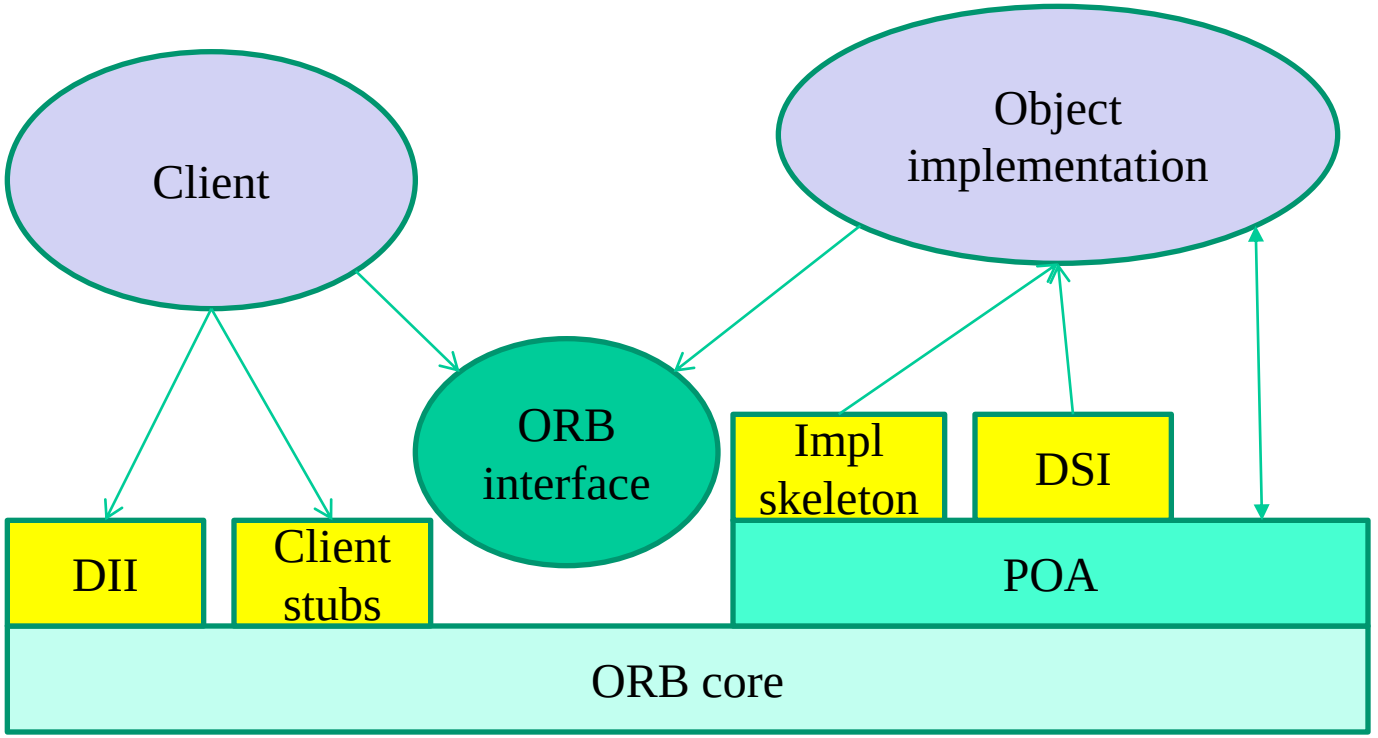
Remote invocation



ORB roles

- ORB helps the client to invoke a method on a remote object.
- The ORB is also the custodian of the Interface Repository (IR or IFR), an OMG-standardized distributed database containing IDL interface definitions.
- On the client side, the ORB offers a number of services: it provides interface definitions from the IFR, and constructs invocations for use with the Dynamic Invocation Interface (DII).
- On the object side, it involves locating the object, activating the object if necessary and then communicating the client's request… CORBA supports a number of activation patterns, so that different object or component types can activate and de-activate in the way that uses resources best.

ORB model



DII – Dynamic Invocation Interface
DSI – Dynamic Skeleton Interface
POA – Portable Object Adapter

- DII-based invocations are interpreted at run time, and not compiled in a previous step. Using the DII, a client can invoke an operation on a new type of object that it's just discovered (for example, using the Trader Service) - the client programmer must write code to retrieve the object's IDL interface definition from the IFR, and construct an invocation using interfaces defined on the ORB. The invocation itself is a CORBA object with its own object reference.
- Because the interface-dependent marshalling is done by the ORB and not by interface-specific compiled code, there is *only one* DII, serving *every* instance of *every* object type.
- On the other hand, there is a separate stub for each operation of each interface.

The ORB interface

- The ORB interface provides access to every ORB service *except* client invocations (which go through the stubs or the DII), and object activations and calls (which come through the object adapter, typically a POA). They are:
  - access to initial services such as the Naming Service, Trader Service, Root POA, and others, via **list_initial_services** and **resolve_initial_references**;
  - access to the IFR, and construction of DII invocations;
  - object reference operations, including conversion of Object References between session and stringified format;
  - policy operations including **create_policy**.

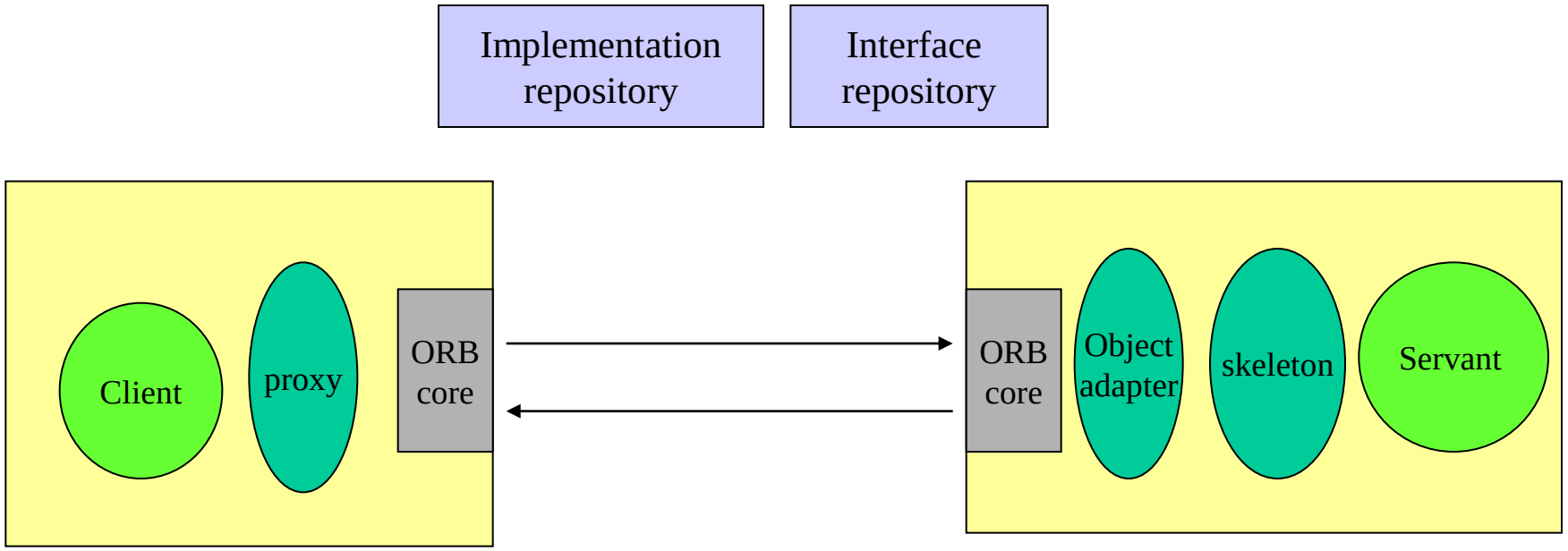
The Portable Object Adapter (POA) – added in CORBA 2.2

- It is the piece of the ORB that manages server-side resources for scalability. By deactivating objects' servants when they have no work to do, and activating them again when they're needed, we can stretch the same amount of hardware to service many more clients.
- The programmer fixes resource allocation/de-allocation patterns by setting POA Policies. These determine whether object references created by the POA are transient or persistent; whether activation is per-method-call or longer; and how multiple CORBA objects (either of a single type, or multiple types) map to servants.

- ORB is also the name of an interface that represents the ORB functionality that programmers need to access. It includes:
  - the method `init`, called to initialize the ORB;
  - the method `connect` used to register CORBA objects with the ORB;
  - other methods.

- An ORB implementor is free to locate the code that implements those services *anywhere:*
  - some ORBs run all of their functionality in a single process on the same machine as their clients and servers;
  - some ORBs spread their work over multiple processes on a single machine, and others spread their work over many machines using the network.

CORBA main components



The Security Service

- Includes authentication of users and servers.
- Access control can be applied to CORBA objects when they receive remote method invocations. These can be specified in access control lists (ACLs).
- To make a secure rmi, the clients' credentials are sent in the request message. The server validates them.
- The target object may also record details about the invocation in an audit log or store non-repudiation credentials.

Links

- http://www.omg.org/gettingstarted/corbafaq.htm