ECE151 – Lecture 15

Chapter 9

Distributed Object-Based Systems

CORBA

CORBA

CORBA: **C**ommon **O**bject **R**equest **B**roker **A**rchitecture

Background:

Developed by the **Object Management Group (OMG)** in response to industrial demands for object based middleware

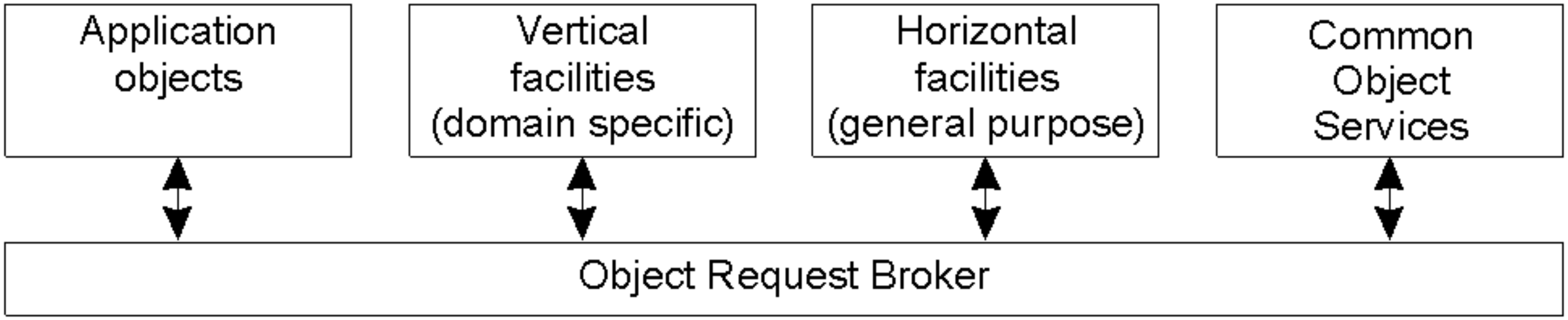
Currently in version #3

CORBA is a *specification*: different implementations of CORBA exist

Very much the work of a committee: there are over 800 members of the OMG and many of them have a say in what CORBA should look like

CORBA provides a simple distributed-object model, with specifications for many supporting services. In enterprises, it is legacy but is here to stay for a many years.

Overview of CORBA


Application objects
Vertical facilities (domain specific)
Horizontal facilities (general purpose)
Common Object Services
Object Request Broker

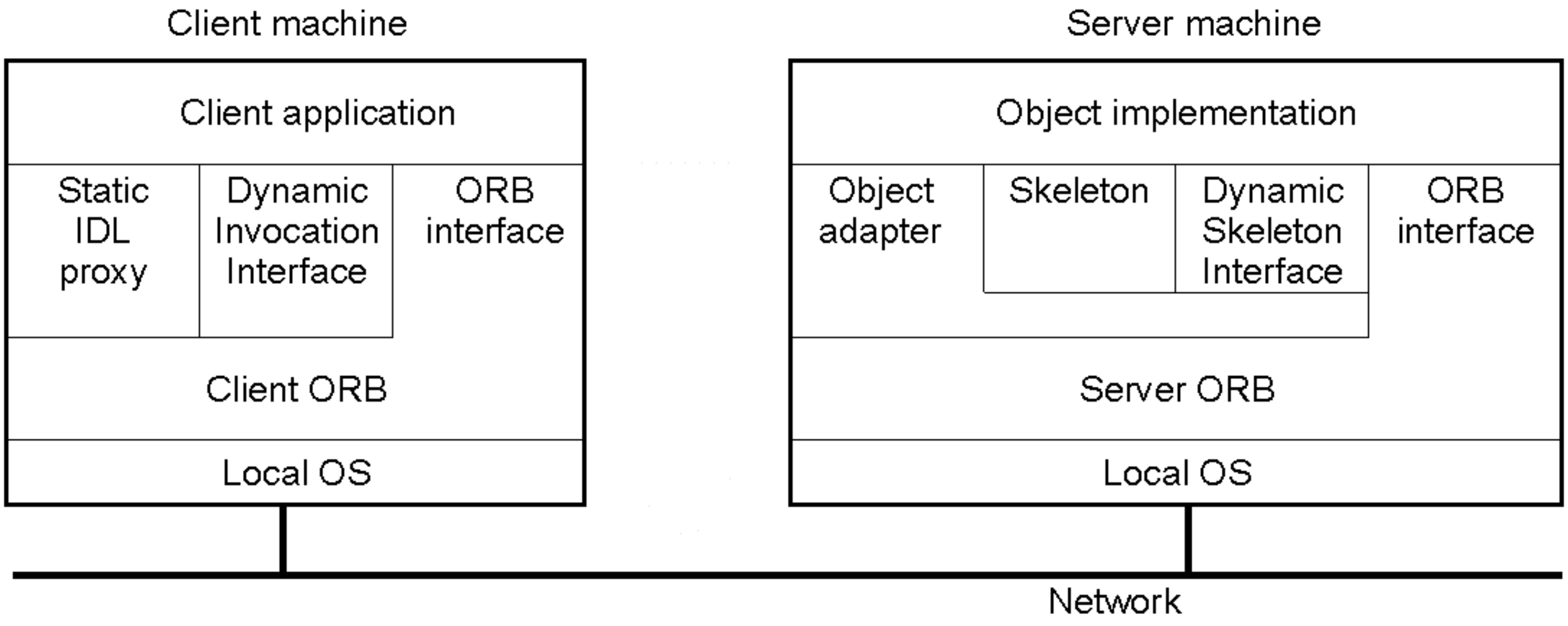
The global architecture of CORBA.

Object Model

Object Request Broker (ORB): CORBA's object broker that connects clients, objects, and services

Proxy/Skeleton: Precompiled code that takes care of (un)marshaling invocations and results

Dynamic Invocation/Skeleton Interface (DII/DSI): To allow clients to "construct" invocation requests at runtime instead of calling methods at a proxy, and having the server-side "reconstruct" those request into regular method invocations

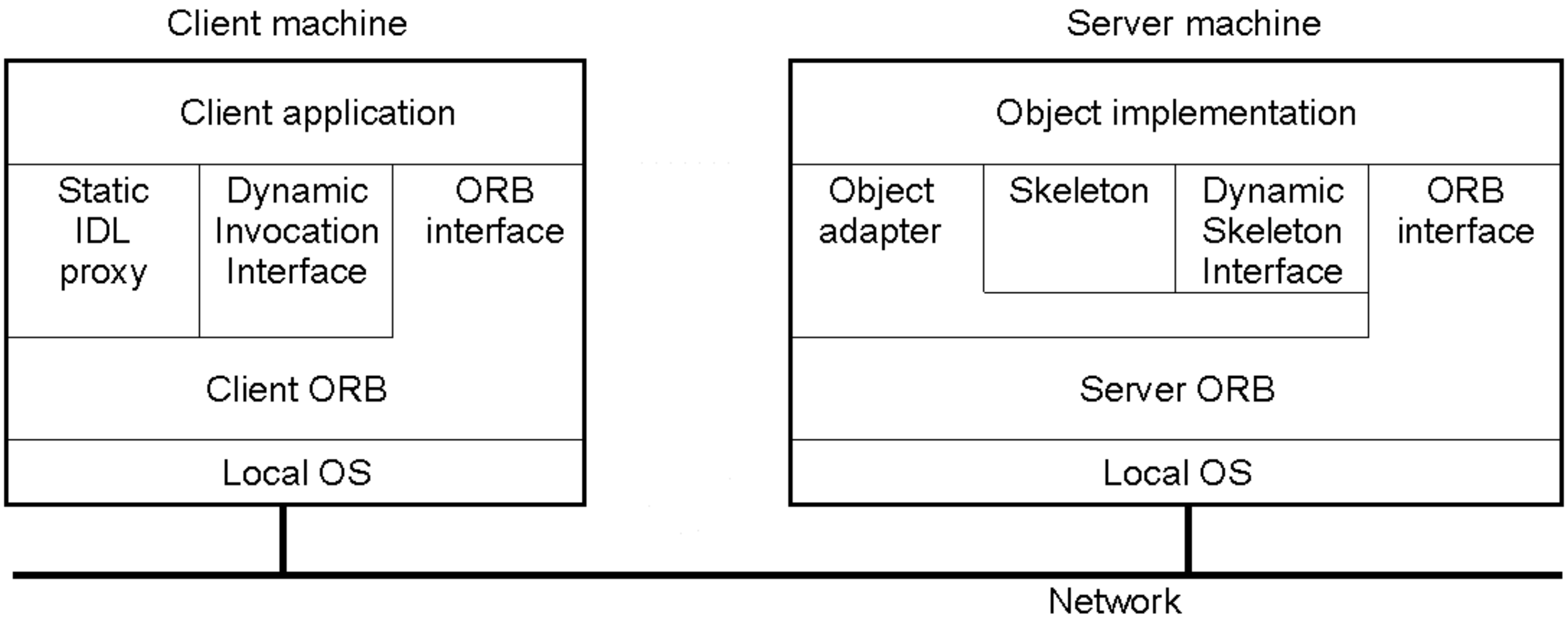


Object Model

Object adapter: Server-side code that handles incoming invocation requests.

Interface repository: Database containing interface definitions and which can be queried at runtime

Implementation repository: Database containing the implementation (code, and possibly also state) of objects. Effectively: a server that can launch object servers.



CORBA Object Model

CORBA has a “traditional” remote-object model in which an object residing at an object server is remote accessible through proxies

Observation: All CORBA specifications are given by means of interface descriptions, expressed in an IDL. CORBA follows an interface-based approach to objects:

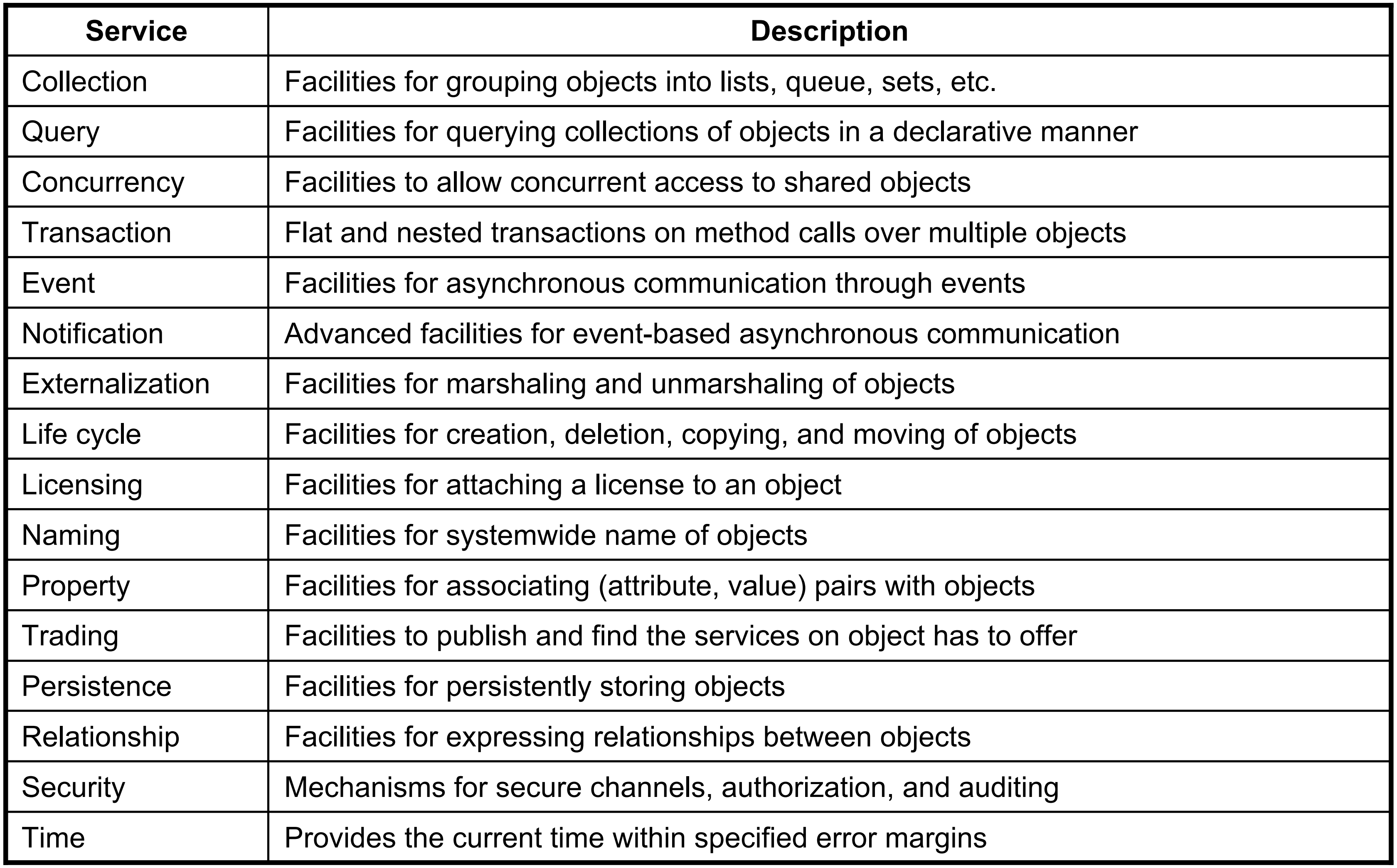
Not the objects, but interfaces are the really important entities

An object may implement one or more interfaces

Interface descriptions can be stored in an interface repository, and looked up at runtime

Mappings from IDL to specific programming are part of the CORBA specification (languages include C, C++, Smalltalk, Cobol, Ada, and Java.

Corba Services



Service	Description
Collection	Facilities for grouping objects into lists, queue, sets, etc.
Query	Facilities for querying collections of objects in a declarative manner
Concurrency	Facilities to allow concurrent access to shared objects
Transaction	Flat and nested transactions on method calls over multiple objects
Event	Facilities for asynchronous communication through events
Notification	Advanced facilities for event-based asynchronous communication
Externalization	Facilities for marshaling and unmarshaling of objects
Life cycle	Facilities for creation, deletion, copying, and moving of objects
Licensing	Facilities for attaching a license to an object
Naming	Facilities for systemwide name of objects
Property	Facilities for associating (attribute, value) pairs with objects
Trading	Facilities to publish and find the services on object has to offer
Persistence	Facilities for persistently storing objects
Relationship	Facilities for expressing relationships between objects
Security	Mechanisms for secure channels, authorization, and auditing
Time	Provides the current time within specified error margins

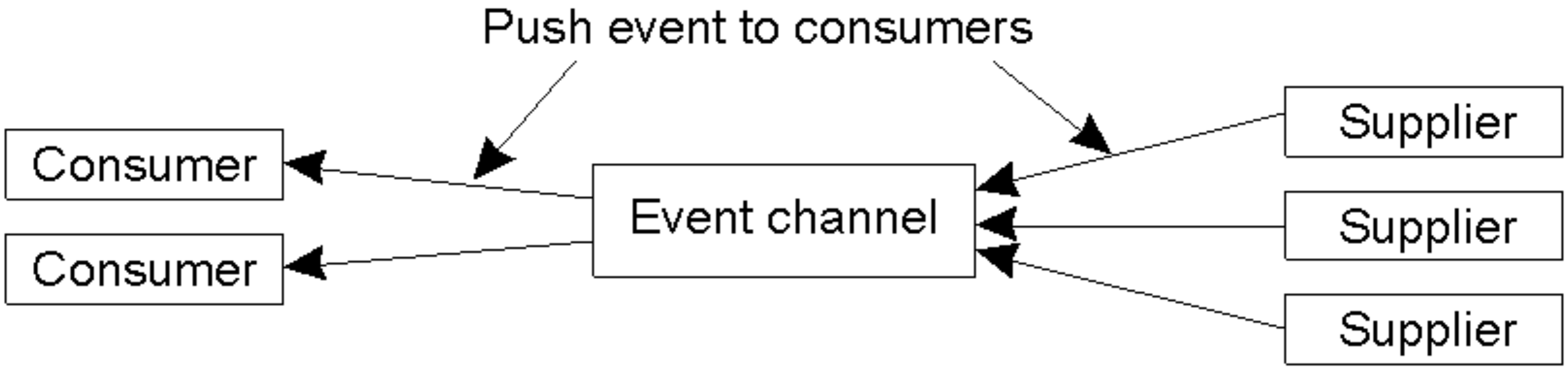
Overview of CORBA services.

Object Invocation Models

Request type	Failure semantics	Description
Synchronous	At-most-once	Caller blocks until a response is returned or an exception is raised
One-way	Best effort delivery	Caller continues immediately without waiting for any response from the server
Deferred synchronous	At-most-once	Caller continues immediately and can later block until response is delivered

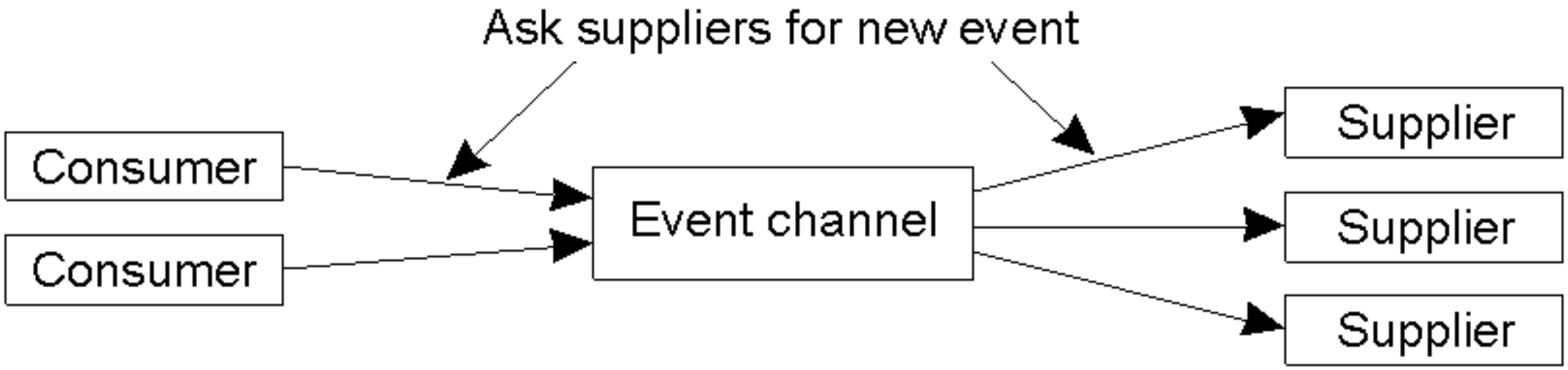
Invocation models supported in CORBA.

Event and Notification Services



The logical organization of suppliers and consumers of events, following the **push-style** model.

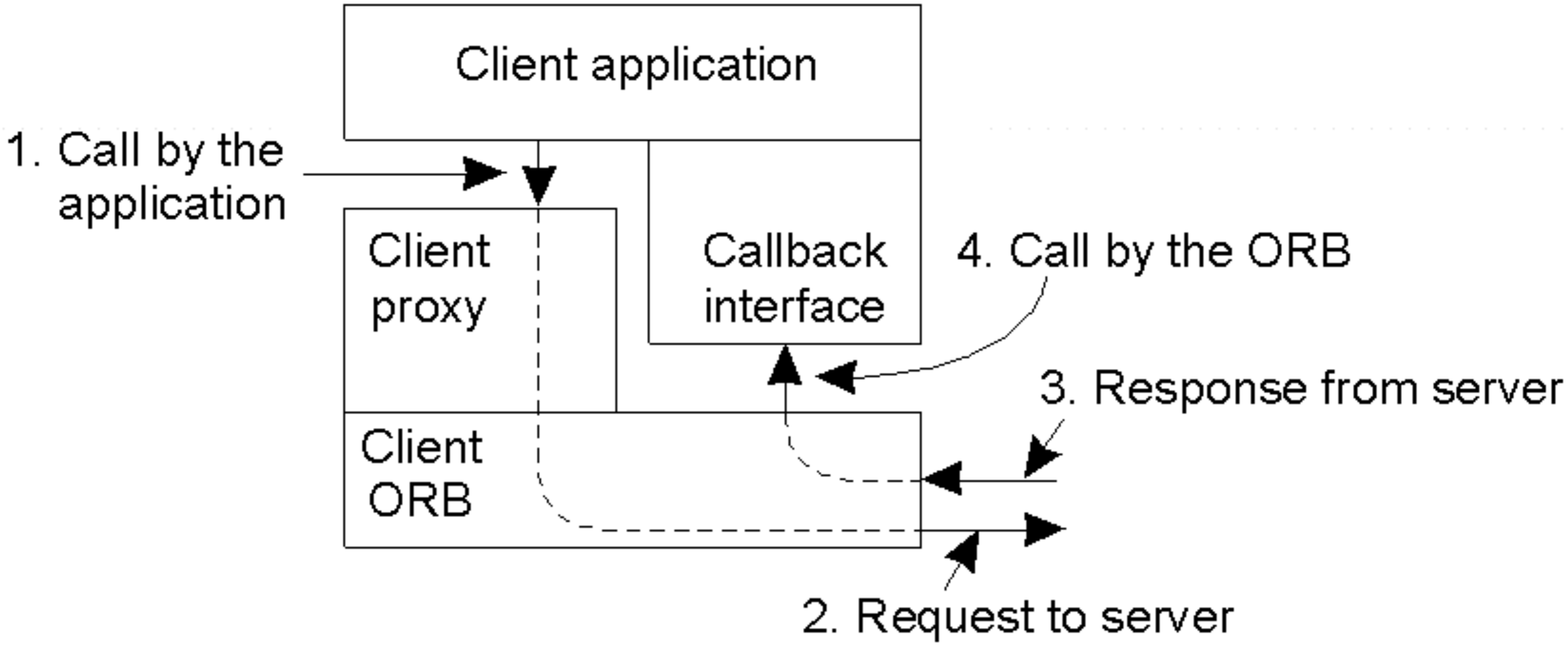
Event and Notification Services



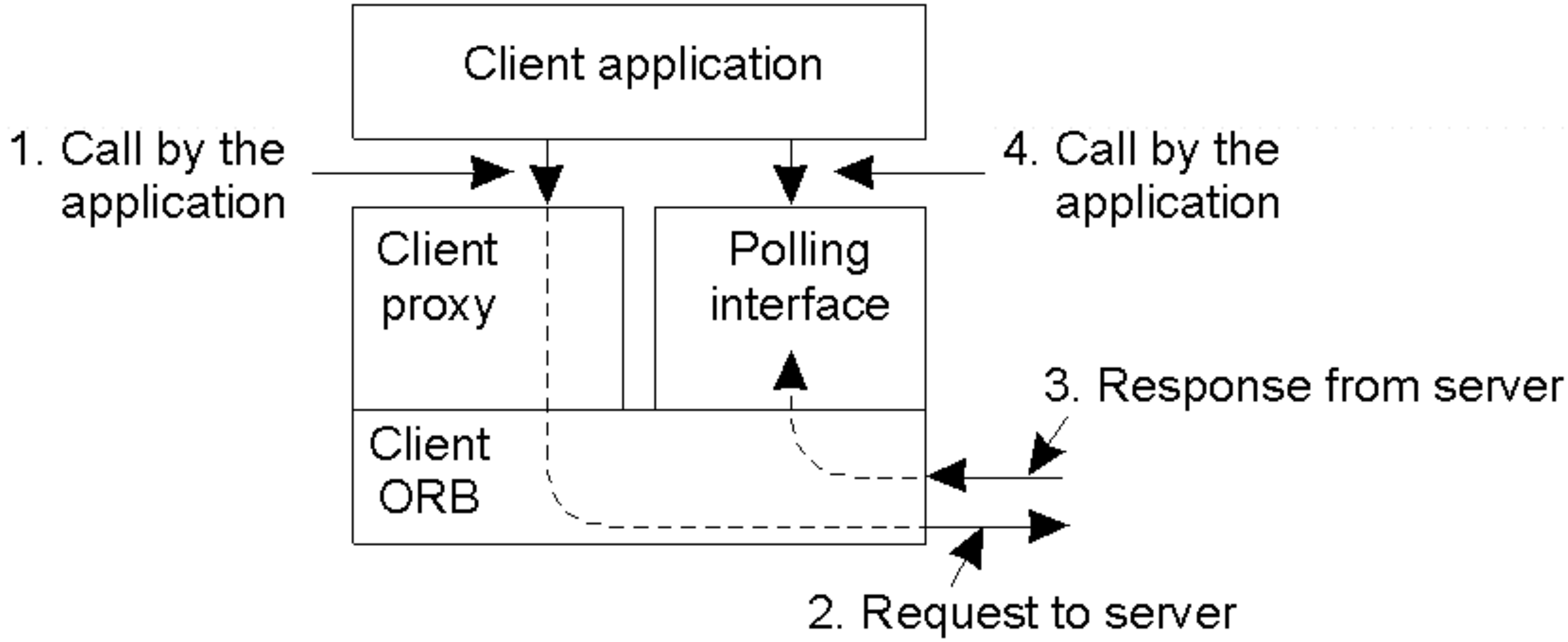
The **pull-style** model for event delivery in CORBA.

Messaging

CORBA's callback model for asynchronous method



Messaging



CORBA'S polling model for asynchronous method invocation.

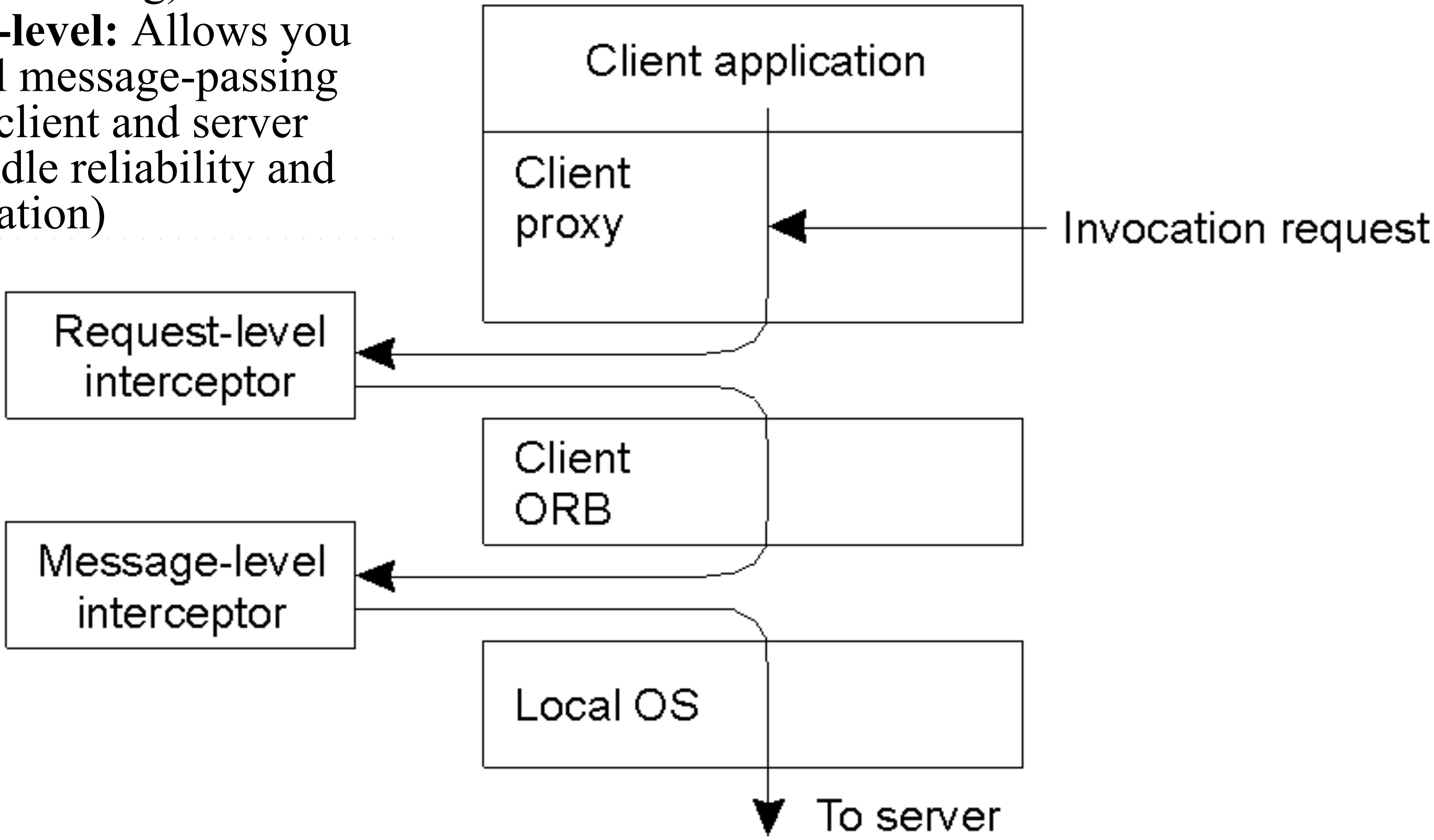
Interoperability

Message type	Originator	Description
Request	Client	Contains an invocation request
Reply	Server	Contains the response to an invocation
LocateRequest	Client	Contains a request on the exact location of an object
LocateReply	Server	Contains location information on an object
CancelRequest	Client	Indicates client no longer expects a reply
CloseConnection	Both	Indication that connection will be closed
MessageError	Both	Contains information on an error
Fragment	Both	Part (fragment) of a larger message

GIOP message types.

Interceptors

Request-level: Allows you to modify invocation semantics (e.g., multicasting)
Message-level: Allows you to control message-passing between client and server (e.g., handle reliability and fragmentation)



Logical placement of interceptors in CORBA.

Naming

Important: In CORBA, it is essential to distinguish specification-level and implementation-level object references

Specification level: An object reference is considered to be the same as a proxy for the referenced object having an object reference means you can directly invoke methods; there is no separate client-to-object binding phase

Implementation level: When a client gets an object reference, the implementation ensures that, one way or the other, a proxy for the referenced object is placed in the client's address space:

```
ObjectReference objRef;
objRef = bindTo(object O in server S at host H);
```

Object references in CORBA used to be highly **implementation dependent**: different implementations of CORBA could normally not exchange their references.

Interoperable Object References

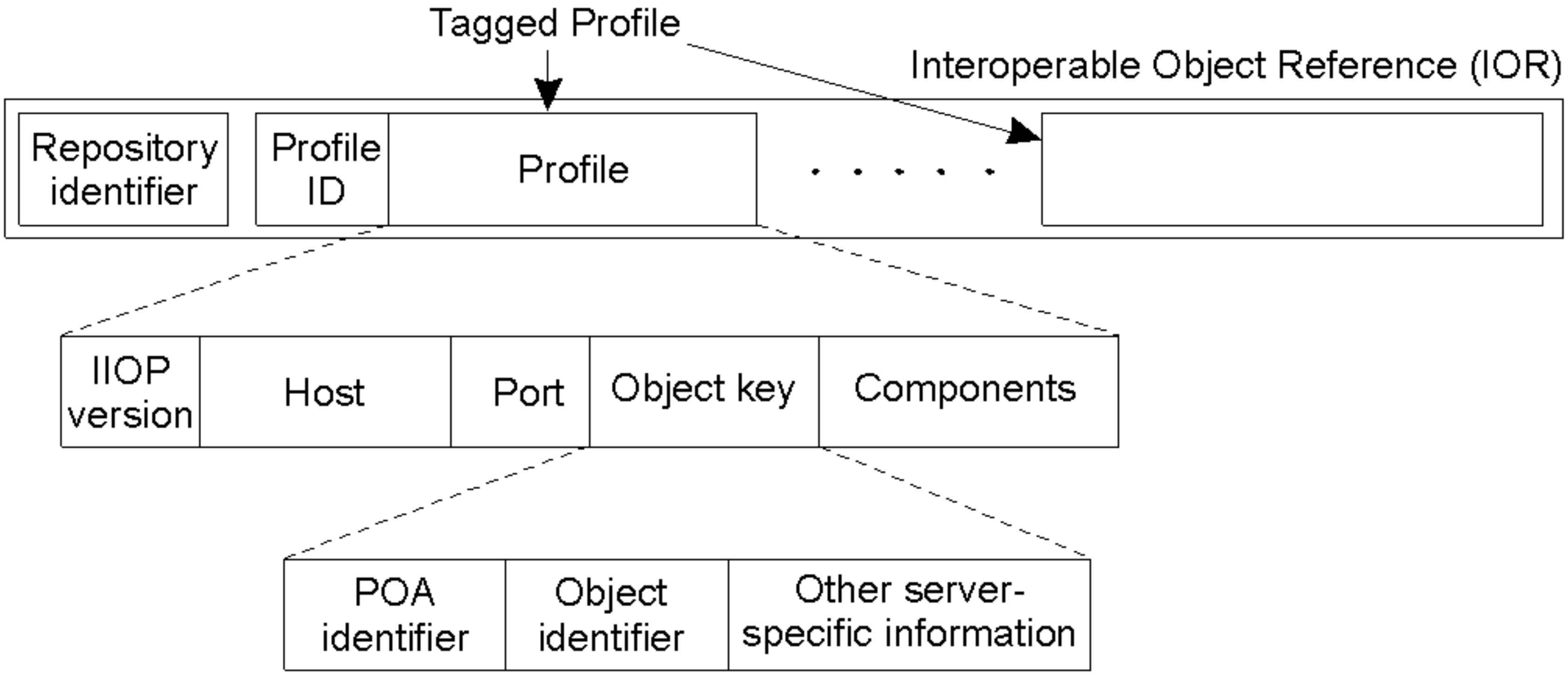
Observation: Recognizing that object references are implementation dependent, we need a separate referencing mechanism to cross ORB boundaries

Solution: Object references passed from one ORB to another are transformed by the bridge through which they pass (different transformation schemes can be implemented)

Observation: Passing an object reference *ref*A from ORB A to ORB B circumventing the A-to-B bridge may be useless if ORB B doesn't understand *ref*A

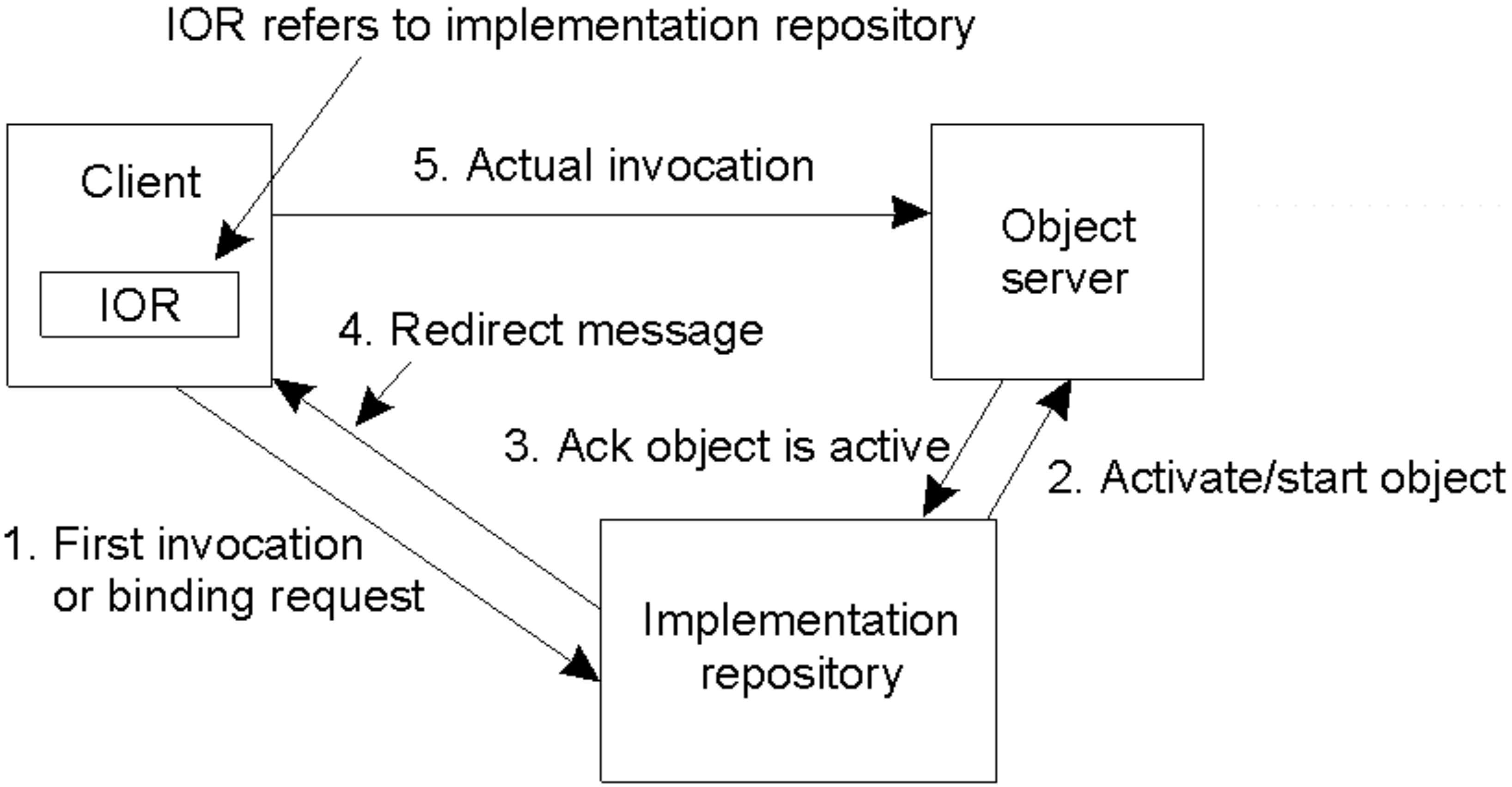
Object References

To allow all kinds of *different* systems to communicate, we standardize the reference that is passed between bridges:



Object References

Indirect binding in CORBA.



Naming Service

CORBA's naming service allows servers to associate a name to an object reference, and have clients subsequently bind to that object by resolving its name

Observation: In most CORBA implementations, object references denote servers at specific hosts; naming makes it easier to relocate objects

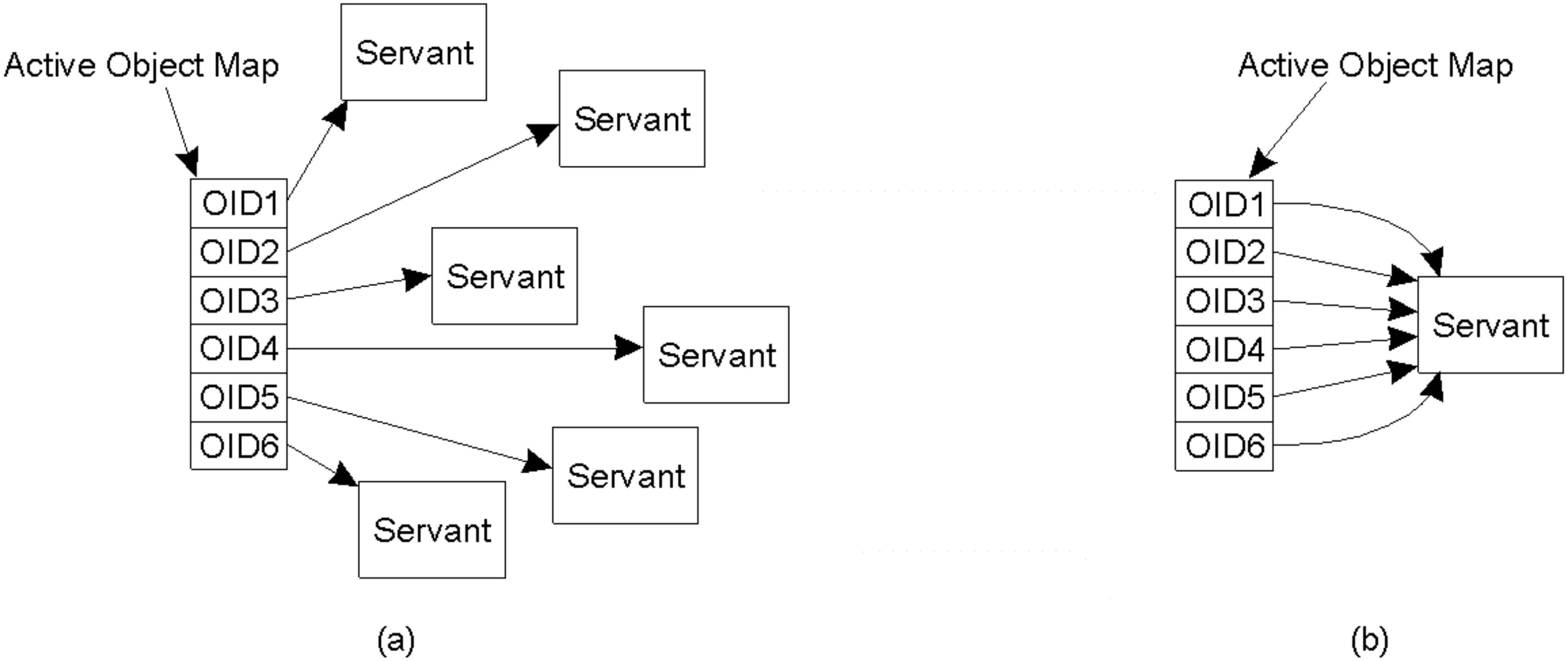
Observation: In the naming graph all nodes are objects; there are no restrictions to binding names to objects => CORBA allows arbitrary naming graphs

Question: How do you imagine cyclic name resolution stops?

Observation: There is no single root; an initial context node is returned through a special call to the ORB.

Also: the naming service can operate *across* different ORBs => **interoperable naming service**

Portable Object Adaptor



Mapping of CORBA object identifiers to servants.

a) The POA supports multiple servants.

b) The POA supports a single servant.

Portable Object Adaptor

```
My_servant *my_object;              // Declare a reference to a C++ object
CORBA::Objectid_var oid;            // Declare a CORBA identifier

my_object = new MyServant;          // Create a new C++ object
oid = poa ->activate_object (my_object);
                                    // Register C++ object as CORBA OBJECT
```

Changing a C++ object into a CORBA object.

Fault Tolerance

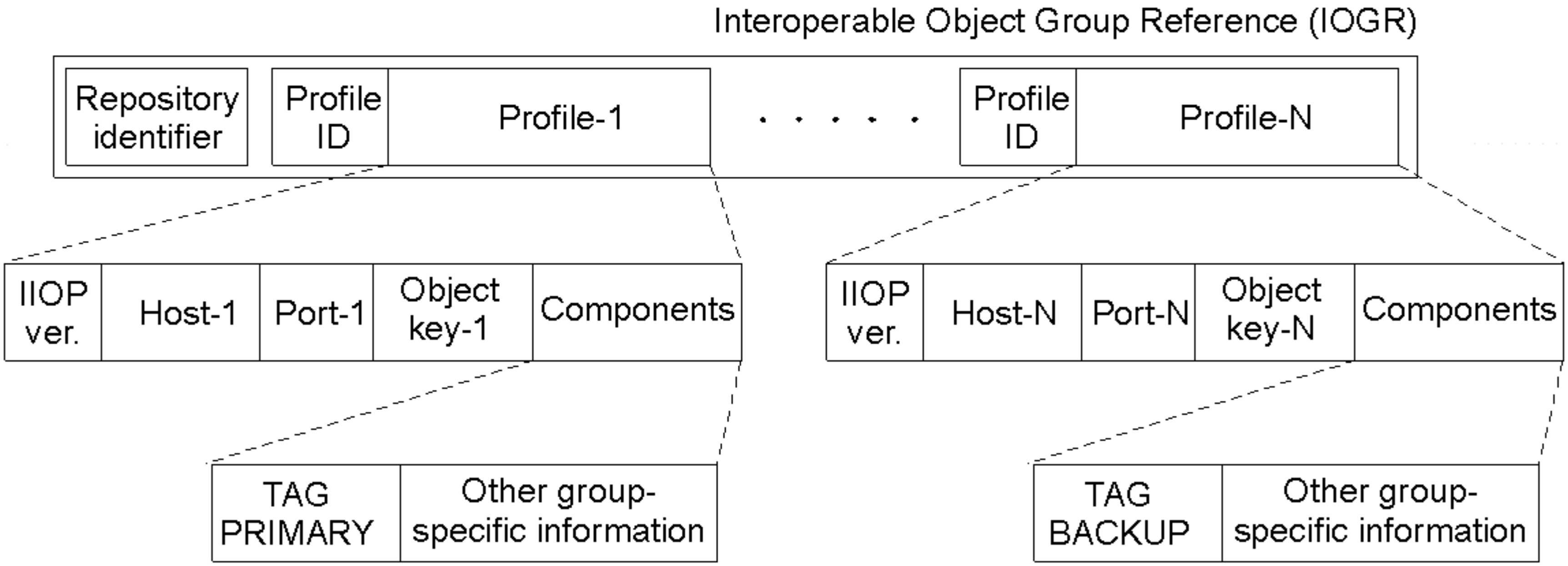
Essence: Mask failures through replication of objects.

Replicas form **object groups**.

Object groups are transparent to clients: they appear as "normal" objects.

This approach requires a separate type of object reference: **Interoperable Object Group Reference**:

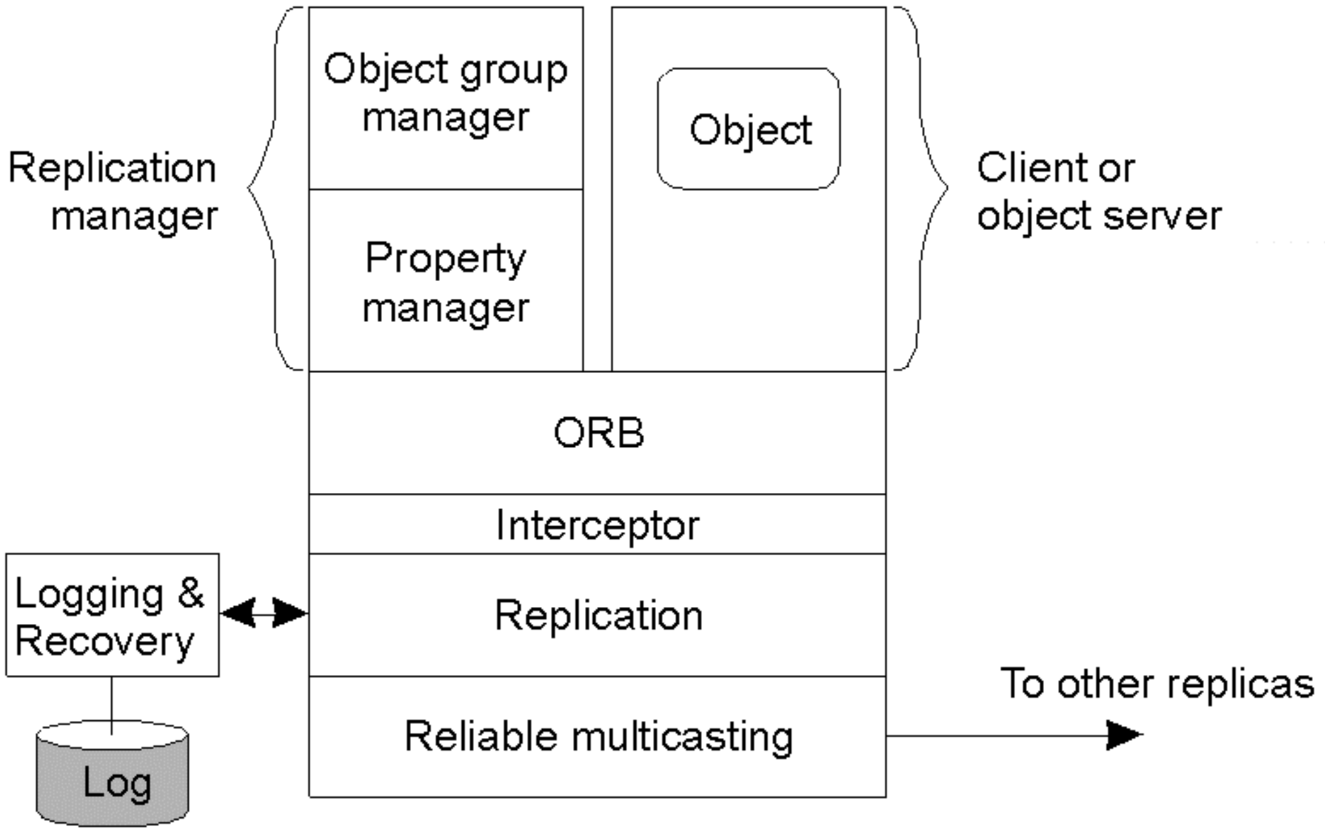
Object Groups



IOGRs have the same structure as IORs, but different *uses*. In IORs an additional profile is used as an alternative; in IOGR, it denotes another replica.

An Example Architecture

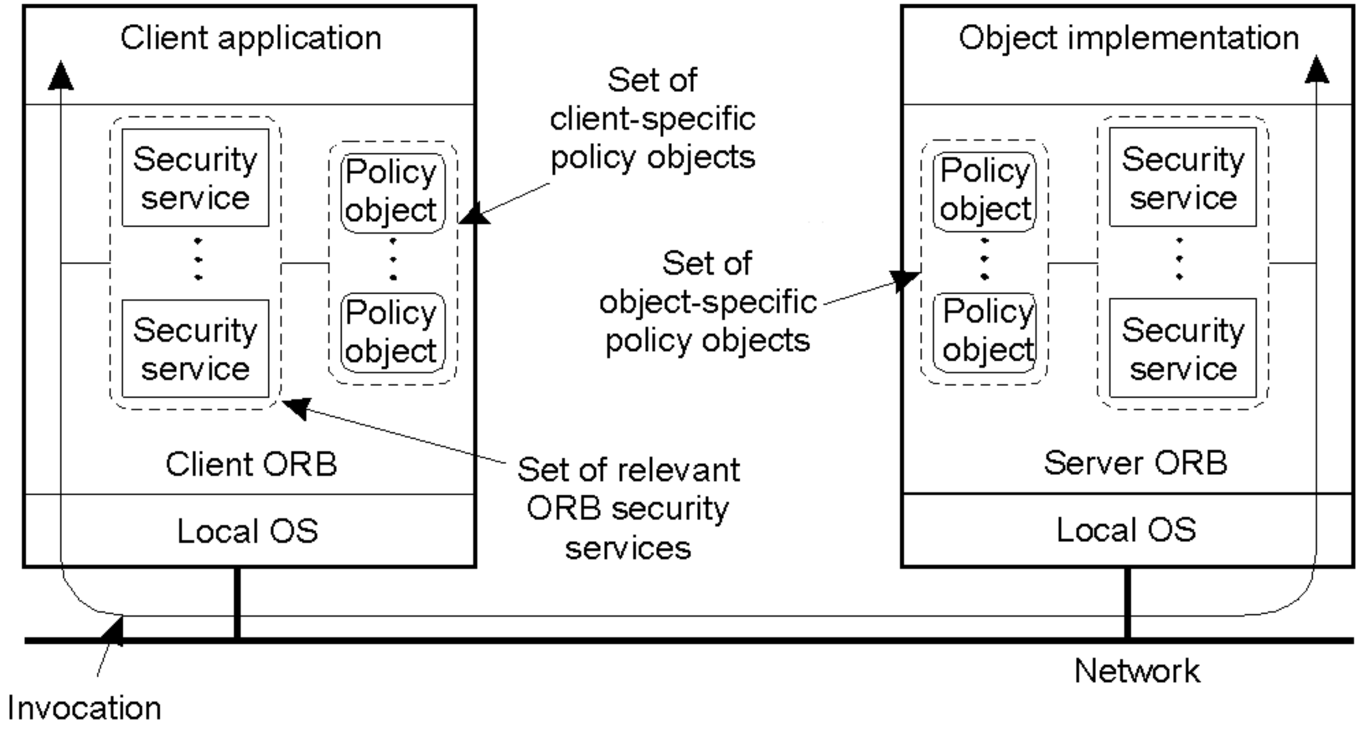
An example architecture of a fault-tolerant CORBA system.



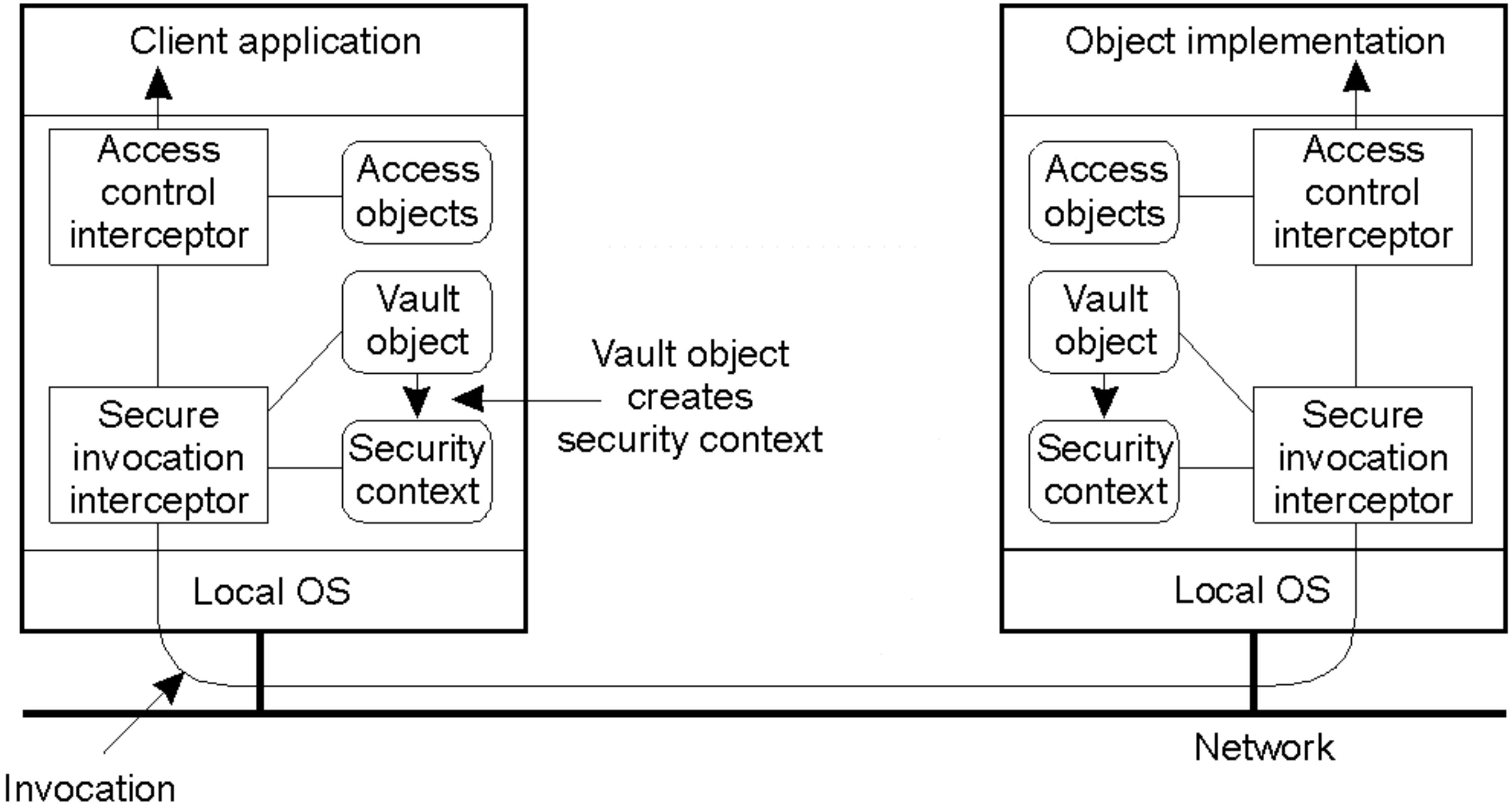
Security

Allow the client and object to be mostly unaware of all the security policies, except perhaps at binding time; the ORB does the rest. Specific policies are passed to the ORB as (local) objects and are invoked when necessary:

Examples: Type of message protection, lists of trusted parties.



Security (2)



The role of security interceptors in CORBA.