Introduction to CORBA

1. **Introduction**
2. **Distributed Systems: Notions**
3. **Middleware**
4. **CORBA Architecture**

1. Introduction

•CORBA is defined by the OMG

•The OMG:

-Founded in 1989 by eight companies as a non-profit corporation.

-The consortium now includes over 800 members.

-Charter: establishment of industry guidelines and detailed object management specifications to provide a common framework for application development.

•OMG produces specifications, not implementations

•Implementations of OMG specifications can be found on over 50 operating systems

•CORBA consists of a standard framework for developing and maintaining ***distributed software systems***. Specifically, it provides

-A RPC mechanism allowing the invocation of operations across different hardware and operating system platforms.

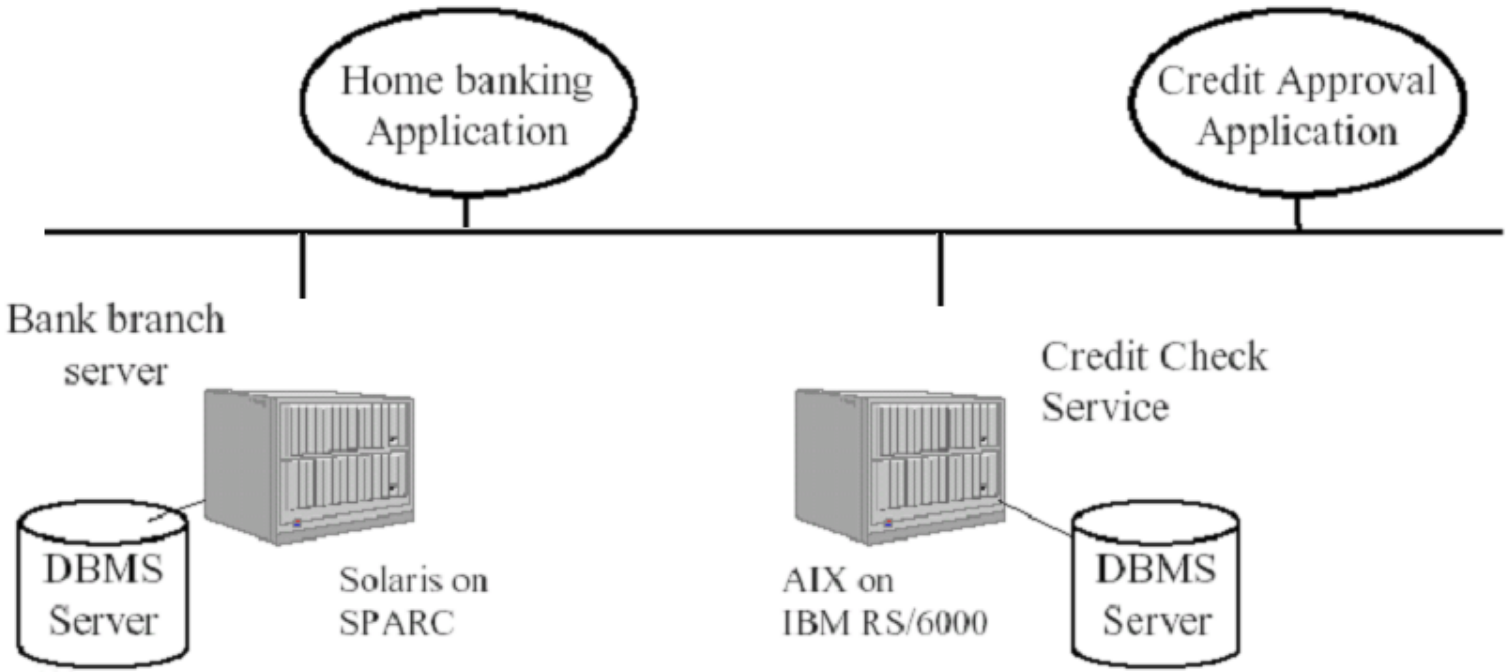
- A component model, the CORBA Component model (CCM).

2. Distributed Systems: Notions

Definition

A distributed system is a system supporting the development of applications and services that can exploit a physical architecture consisting of multiple, autonomous processing elements. Those elements do not share primary memory but cooperate by sending messages over the network.

Example-Distributed System



Centralized vs. Distributed Systems

Centralized Systems

- One component with non-autonomous parts
- Component shared by users all the time
- All resources accessible
- Software runs in a single process
- Single point of control
- Single point of failure

Distributed Systems

- Multiple autonomous components
- Components are not shared by all users
- Resources may not be accessible
- Software runs in concurrent processes on different processors
- Multiple points of control
- Multiple points of failure

Distributed System Requirements

- Heterogeneity
- Resource Sharing
- Openness
- Concurrency
- Scalability
- Fault Tolerance

Heterogeneity

- Hardware platforms
- Operating systems
- Networks
- Programming languages
- Data representation formats

Resource Sharing

- Ability to use any hardware, software or data anywhere in the system.
- Resource manager controls access, provides naming scheme and controls concurrency.
- Resource sharing model (e.g. client/ server or object-based) describing how
  - resources are provided,
  - they are used and
  - provider and user interact with each other.

Openness

- Openness is concerned with extensions and improvements of distributed systems.
- Detailed interfaces of components need to be published.
- New components have to be integrated with existing components.
- Differences in data representation of interface types on different processors (of different vendors) have to be resolved.

Concurrency

- Components in distributed systems are executed in concurrent processes.
- Components access and update shared resources (e.g. variables, databases, device drivers).
- Integrity of the system may be violated if concurrent updates are not coordinated.
- Lost updates
- Inconsistent analysis

Scalability

- Adaption of distributed systems to
  - –accomodate more users
  - –respond faster
- Usually done by adding more and/or faster components.
- Components should not need to be changed when scale of a system increases.
- Design components to be scalable!

Fault Tolerance

- Hardware, software and networks fail
- Distributed systems must maintain availability even at low levels of hardware/software/network reliability.
- Fault tolerance is achieved by
  - –recovery
  - –redundancy

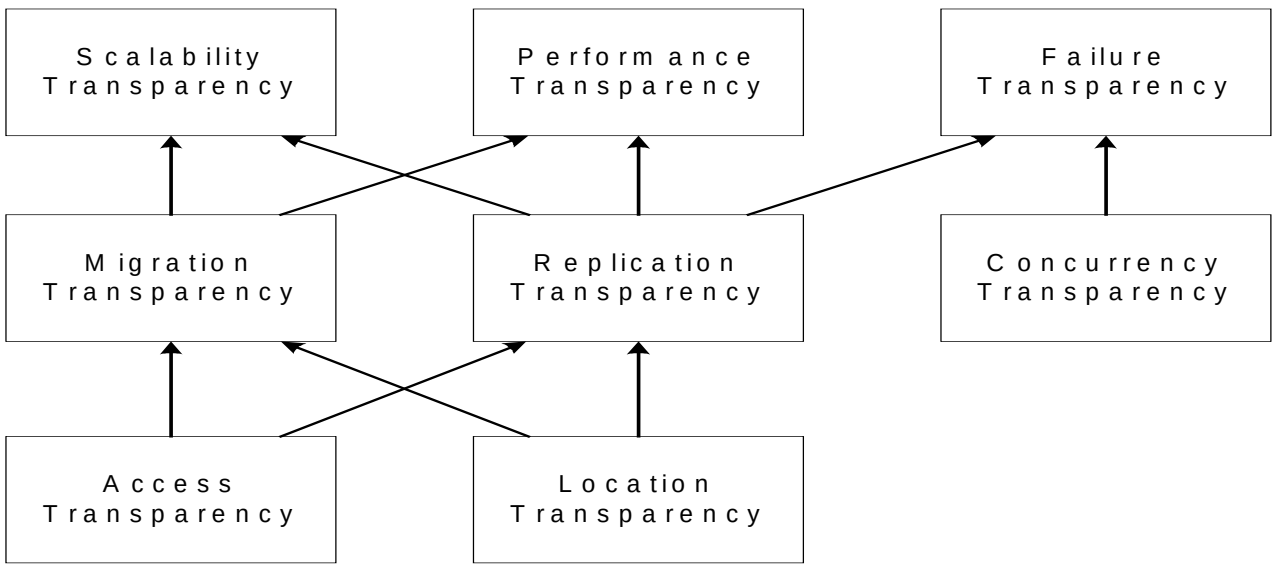
Transparency

•Distributed systems should be perceived by users and application programmers as a whole rather than as a collection of cooperating components.

-Transparency has different dimensions that were identified by ANSA as part of the International Standard of Open Distributed Processing (ODP).

-These represent various properties that distributed systems should have.

Dimensions of Transparency in Distributed Systems



Access Transparency

•Enables local and remote information objects to be accessed using identical operations.

-Example: File system operations in NFS.

-Example: Navigation in the Web.

-Example: SQL Queries

Location Transparency

•Enables information objects to be accessed without knowledge of their location.

-Example: File system operations in NFS

-Example: Pages in the Web

-Example: Tables in distributed databases

Concurrency Transparency

•Enables several processes to operate concurrently using shared information objects without interference between them.

-Example: NFS

-Example: Automatic teller machine network

-Example: Database management system

Replication Transparency

•Enables multiple instances of information objects to be used to increase reliability and performance without knowledge of the replicas by users or application programs

-Example: Distributed DBMS

-Example: Mirroring Web Pages.

Failure Transparency

•Enables the concealment of faults

•Allows users and applications to complete their tasks despite the failure of other components.

-Example: Database Management System

Migration Transparency

•Allows the movement of information objects within a system without affecting the operations of users or application programs.

-Example: NFS

-Example: Web Pages

Performance Transparency

•Allows the system to be reconfigured to improve performance as loads vary.

-Example: Distributed make.

Scaling Transparency

•Allows the system and applications to expand in scale without change to the system structure or the application algorithms.

-Example: World-Wide-Web

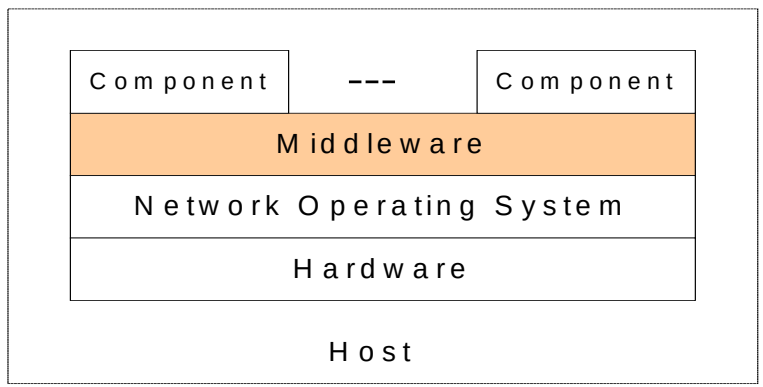
-Example: Distributed Database

3. Middleware Systems

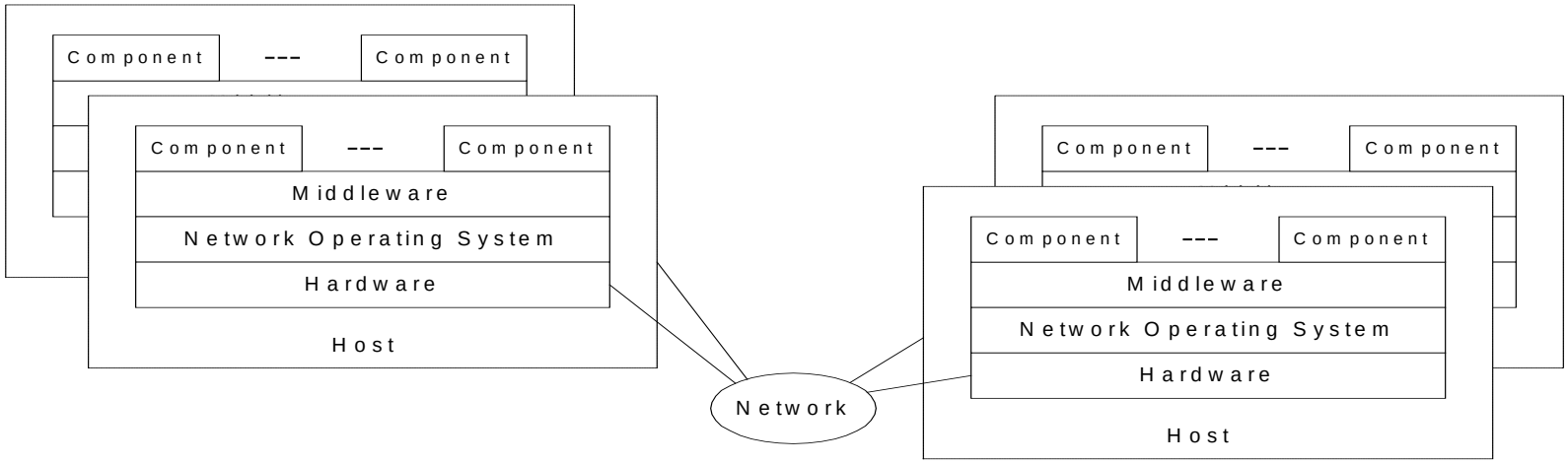
Definition: *Middleware is software that enables interprocess communication. It provides an API that isolates the application code from the underlying network communication formats and protocols (FAPs).*

-Middleware systems achieve the various forms of distribution transparencies by creating the *illusion of unity and homogeneity* within the network, what is called in other words the ***"single-system image".***

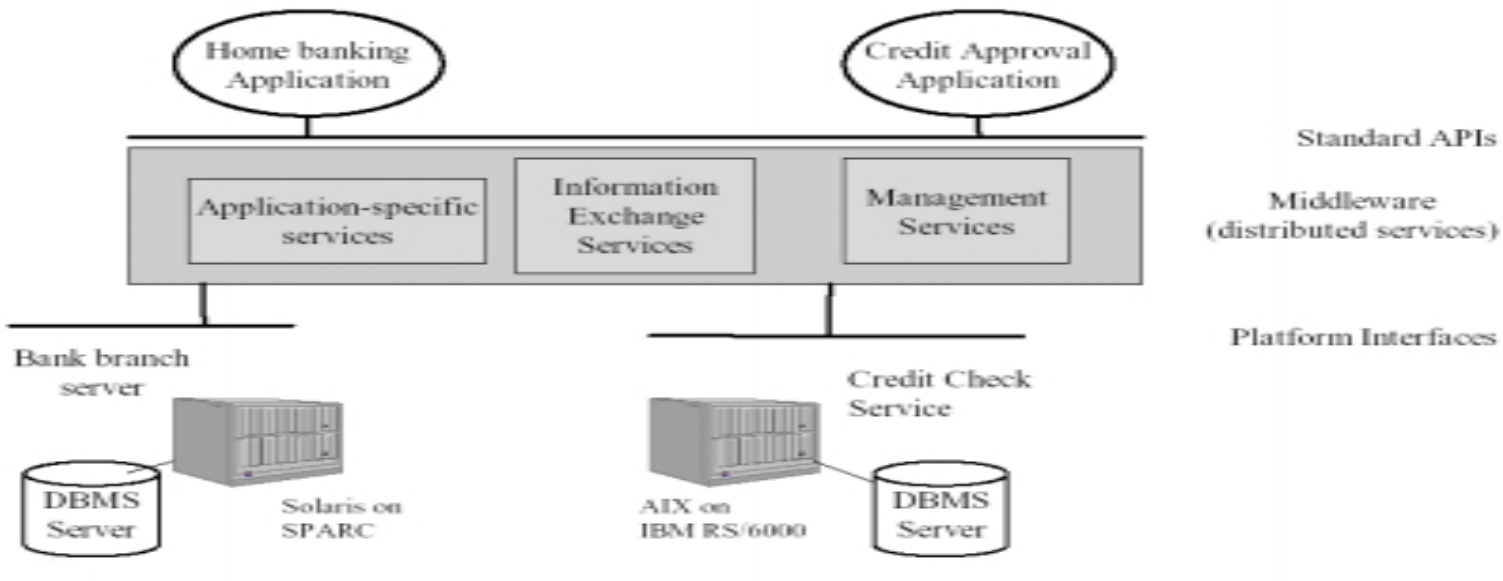
- They act as glue between clients and server processes by providing generic services on top of the OS.



A Working Model of a Distributed System



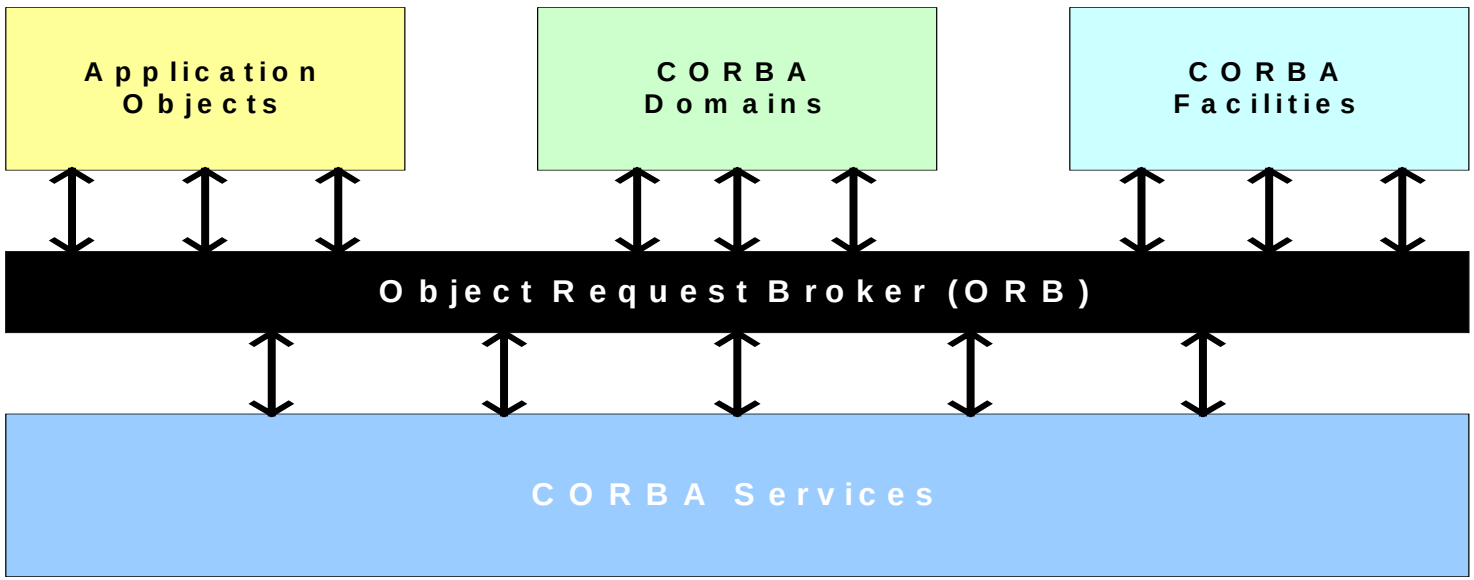
Example - Middleware



4. CORBA Architecture

CORBA Reference Model Architecture

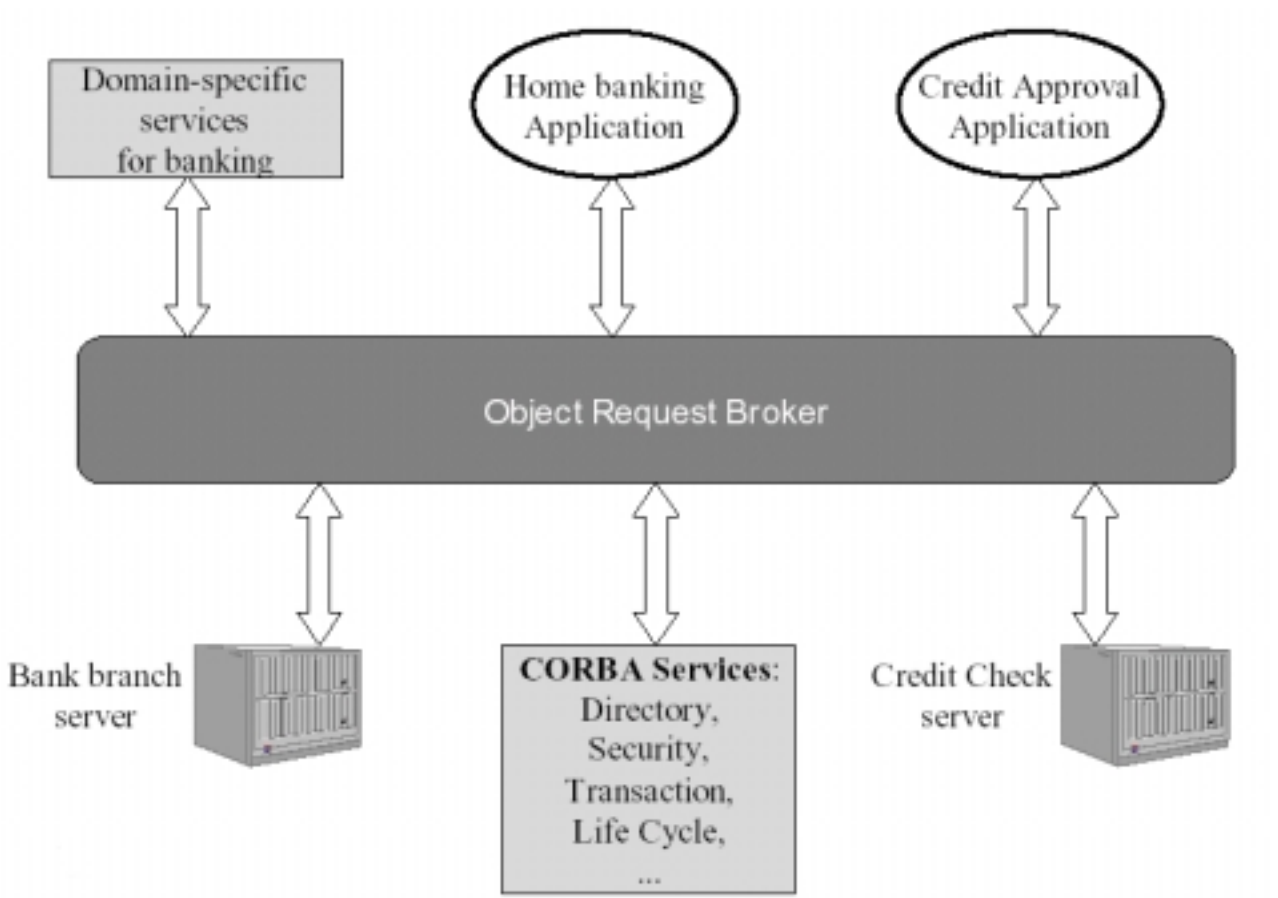
-The CORBA standard relies on the Object Management Architecture:



- A typical CORBA implementation may include:

• An Object Request Broker (ORB) implementation
• An Interface Definition Language (IDL) compiler
• Implementations of Common Object Services (COS), also called CORBA Services
• Common Frameworks, also called CORBA facilities
• An Internet Inter ORB Protocol (IIOP) implementation

Example - Distributed Middleware with CORBA



Examples of CORBA Products

ORB	***Description***
Orbacus	A popular Java and C++ ORB from Iona Technologies
WebSphere	A popular application server with an ORB from IBM
Netscape Communicator	-Netscape browsers have a version of VisiBroker embedded in them
Sun Java 2 Platform	Provides an ORB and two CORBA programming models: -RMI programming model -IDL programming model

Examples ORBs implementations

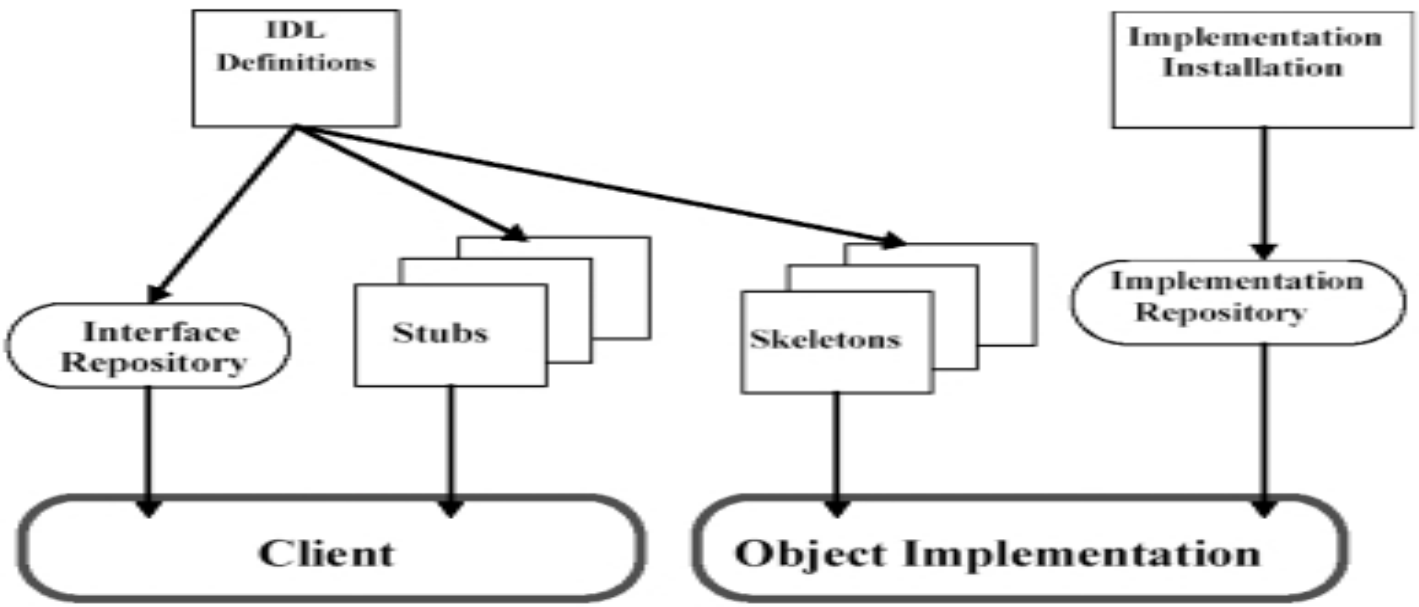
- Client- and Implementation-resident ORB
- Server-based ORB
- System-based ORB
- Library-based ORB

Interface Definition Language (IDL)

-IDL provides a programming language neutral way to define how a service is implemented.

- Is an intermediary language between specification languages such as the UML and programming languages such C, C++ etc.
- Provides an abstract representation of the interfaces that a client will use and a server will implement.

Interface and Implementation Repositories

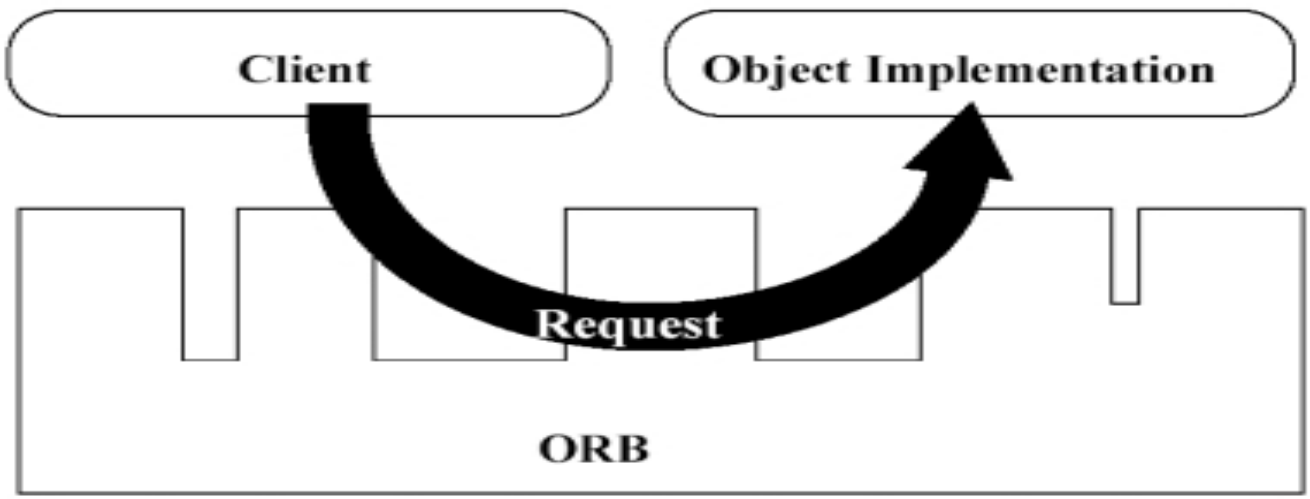


Object Request Broker

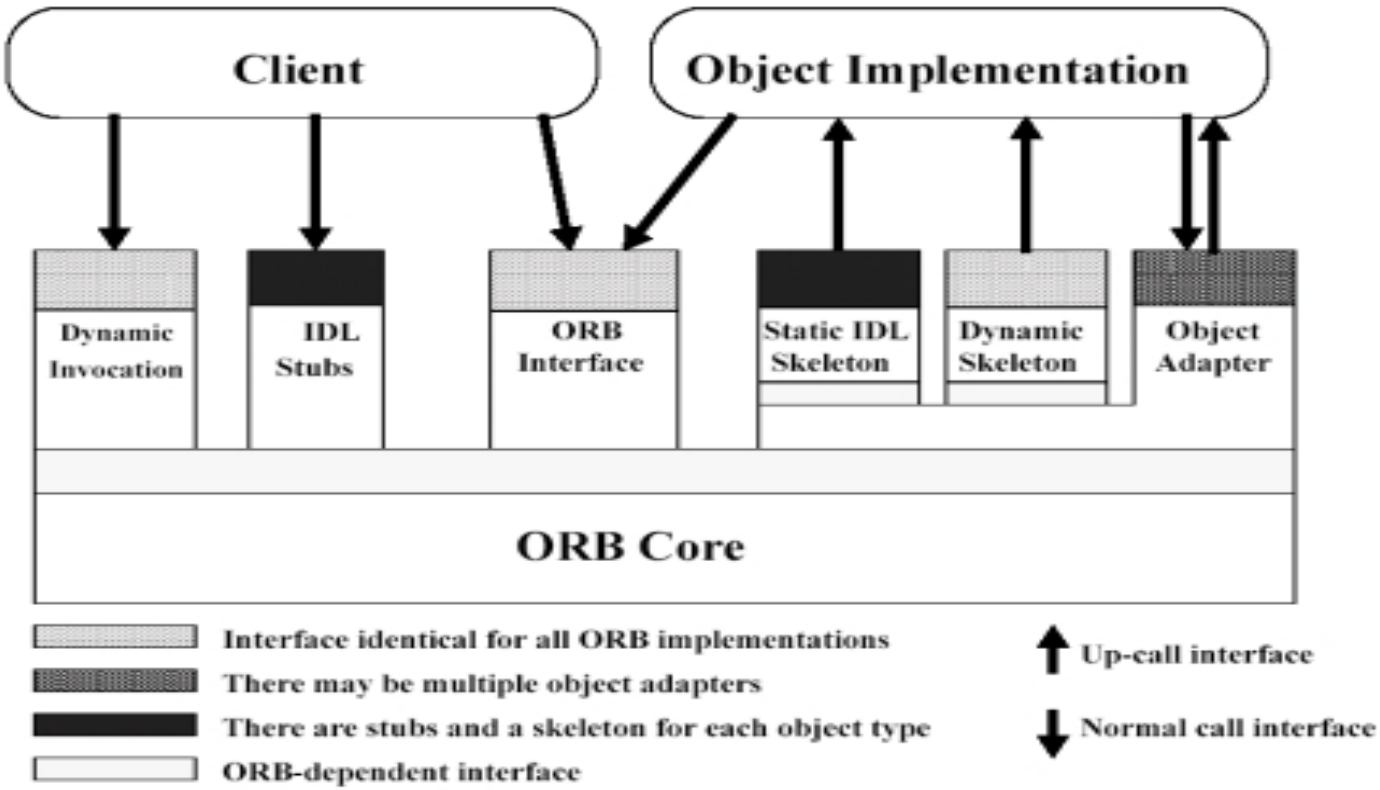
-The interface the client sees is completely independent of where the object is located, what programming language it is implemented in, or any other aspect that is not reflected in the object's interface.

-The ORB is responsible for:

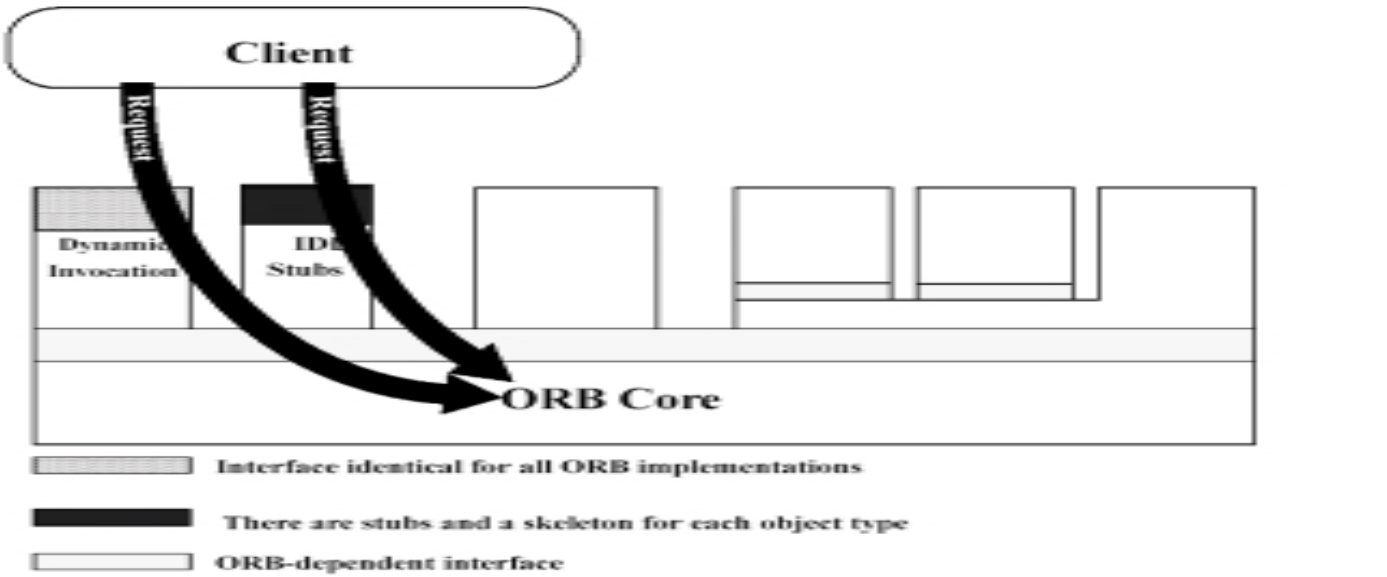
- finding the object implementation for the request,
- preparing the object implementation to receive the request,
- communicating the data making up the request.



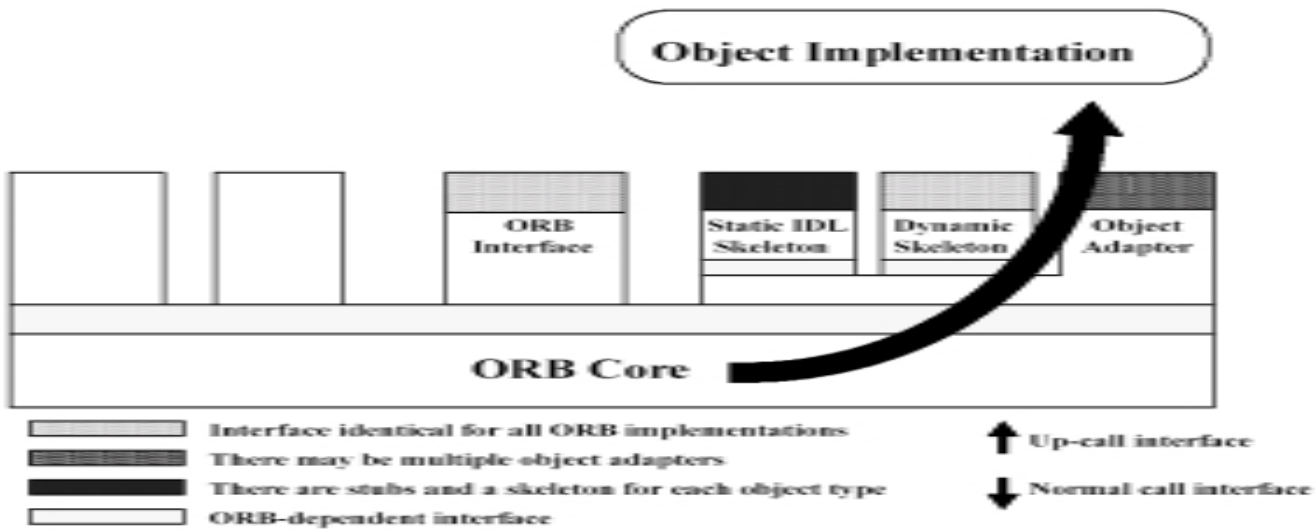
Structure of Object Request Interfaces



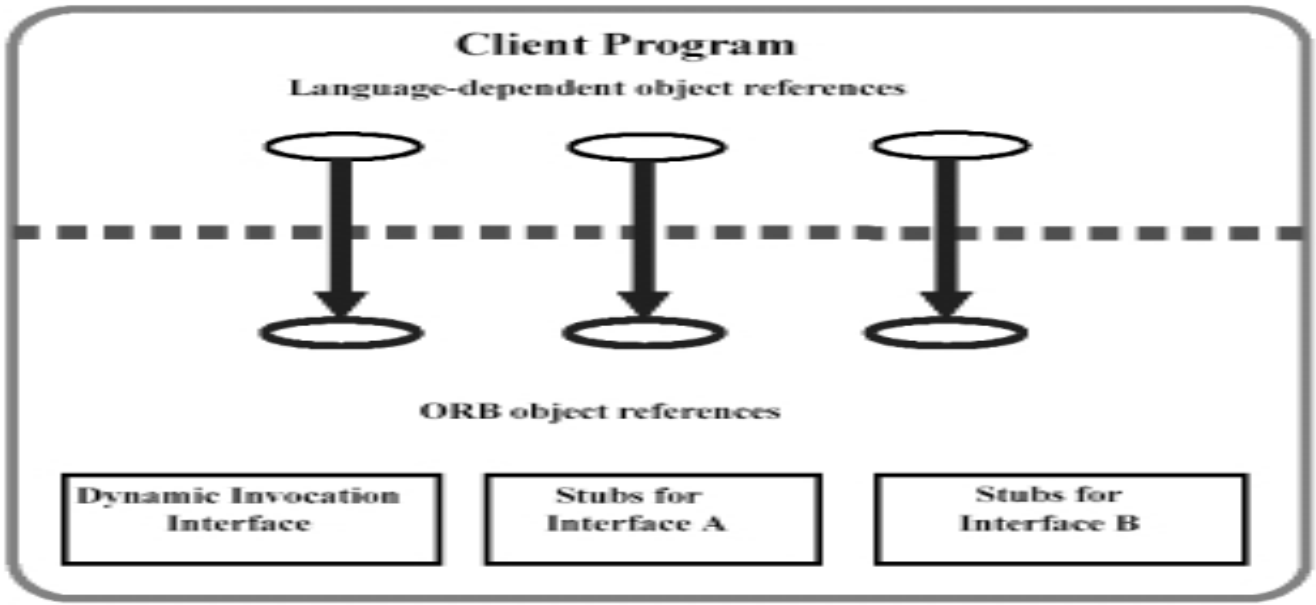
-A Client using the Stub or Dynamic Invocation Interface



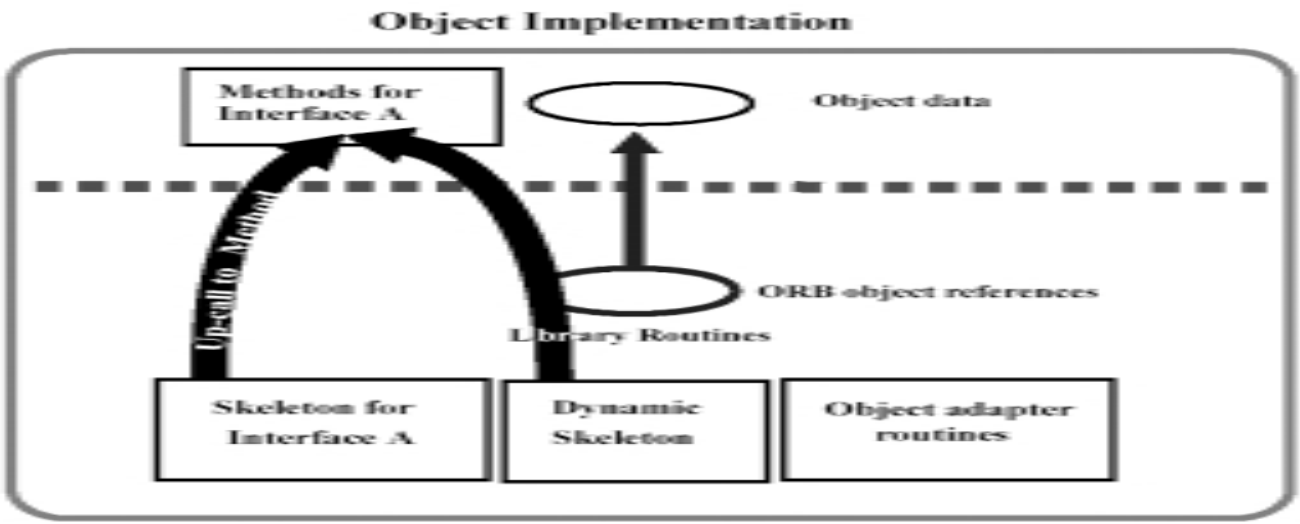
-An Object Implementation Receiving a Request



-The Structure of a Typical Client



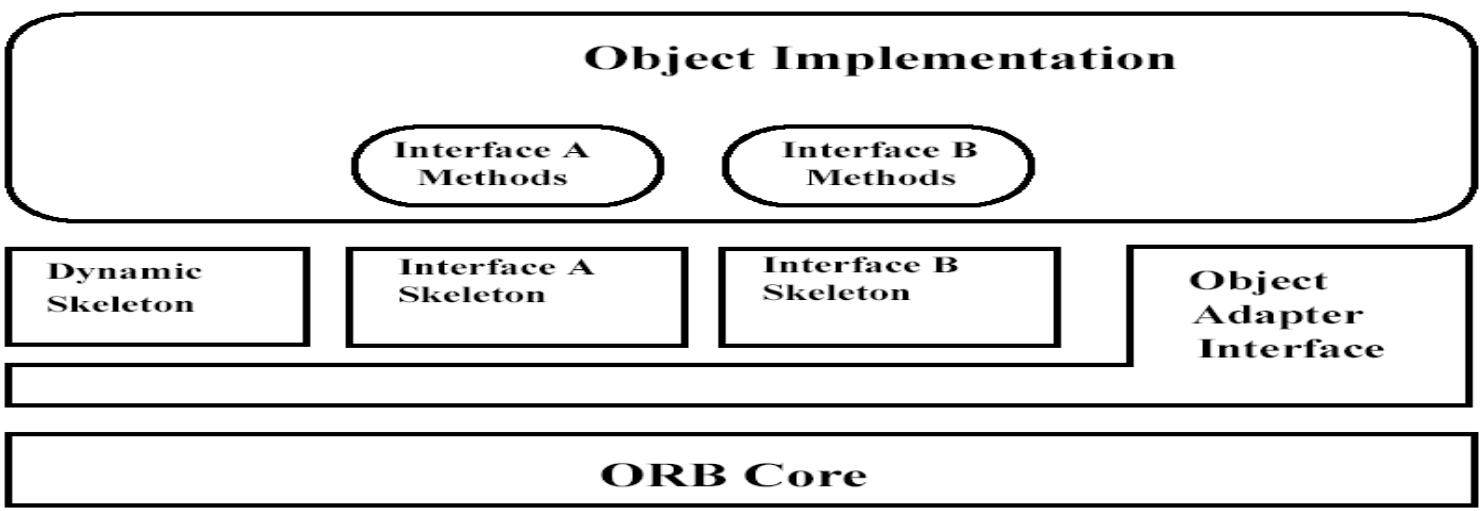
-The Structure of a Typical Object Implementation



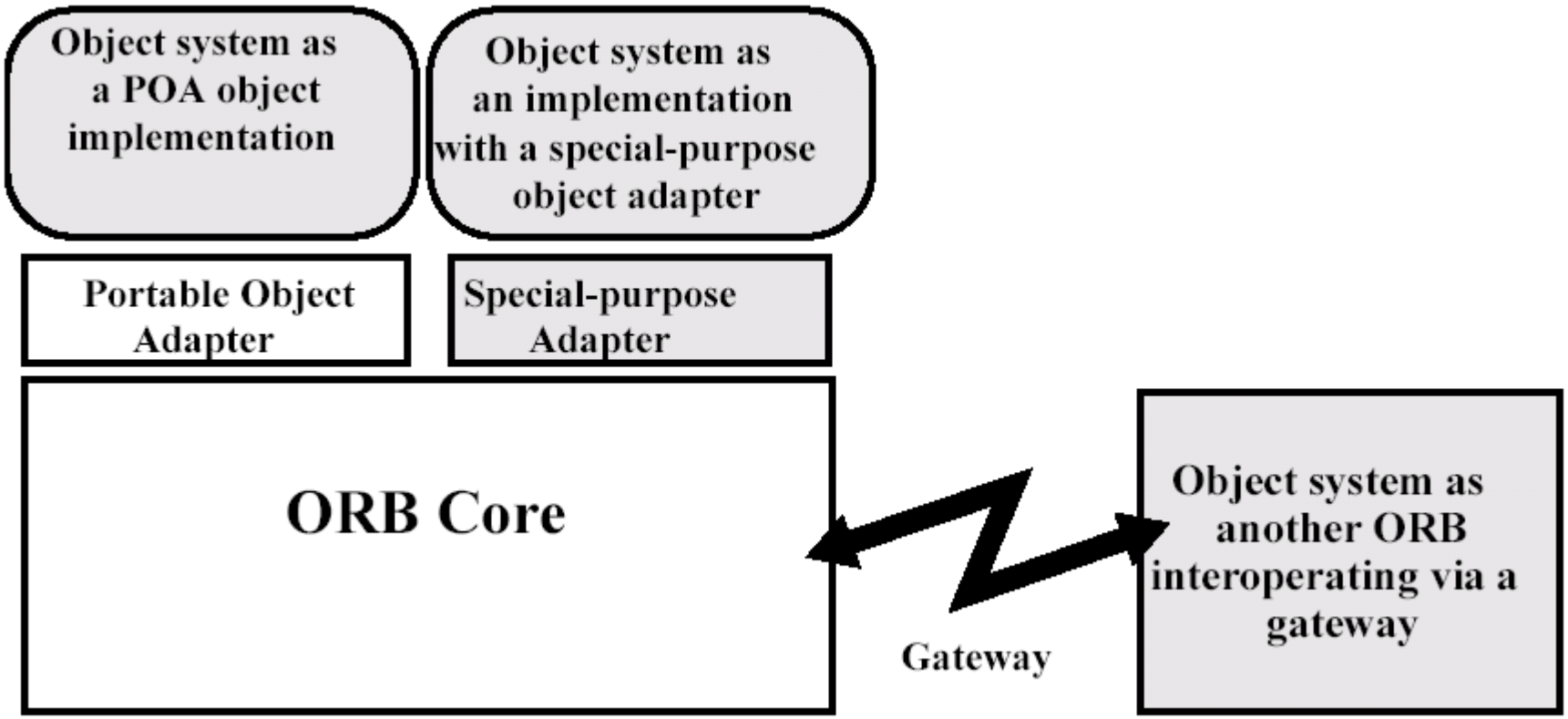
Object Adapter

- An object adapter is the primary means for an object implementation to access ORB services such as object reference generation.
- Object adapters are responsible for the following functions:
  - –Generation and interpretation of object references
  - –Method invocation
  - –Security of interactions
  - –Object and implementation activation and deactivation
  - –Mapping object references to the corresponding object implementations
  - –Registration of implementations

The Structure of a Typical Object Adapter



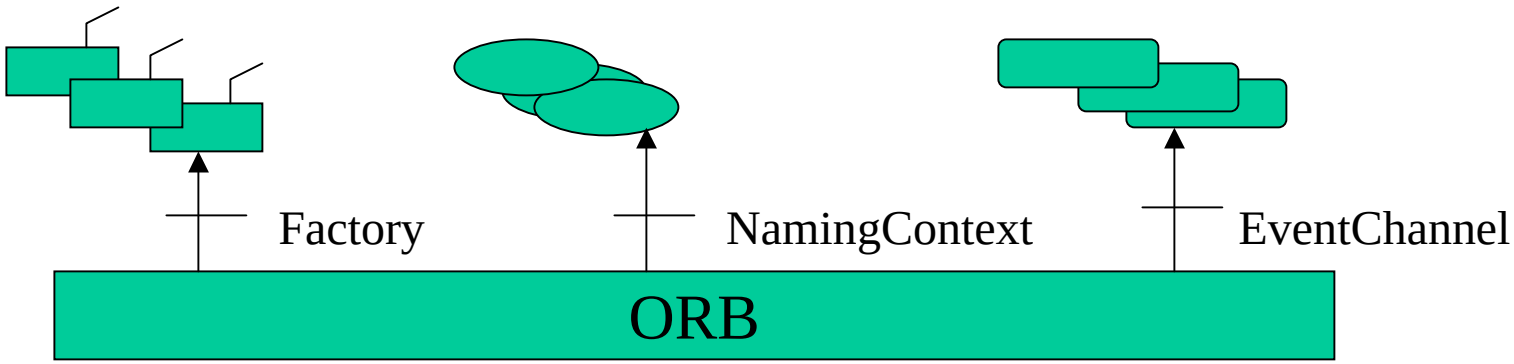
Different Ways to Integrate Foreign Object Systems



Common Object Services (COS)

-Up to 15 services are currently available that assist the ORB.

- They are defined on top of the ORB, as standard CORBA objects with IDL interfaces



-Popular services include:

- *Naming Service*: provides a way for CORBA clients (and servers) to find objects on the network.
- *Event Service*: stores and delivers events sent by clients or servers to their target objects.
- *Security Service*: provides a means to authenticate messages, authorize access to objects, and provide secure communications.
- *Transaction Service*: defines a means to control an action against a database or other subsystem.

ORB Interoperability Architecture

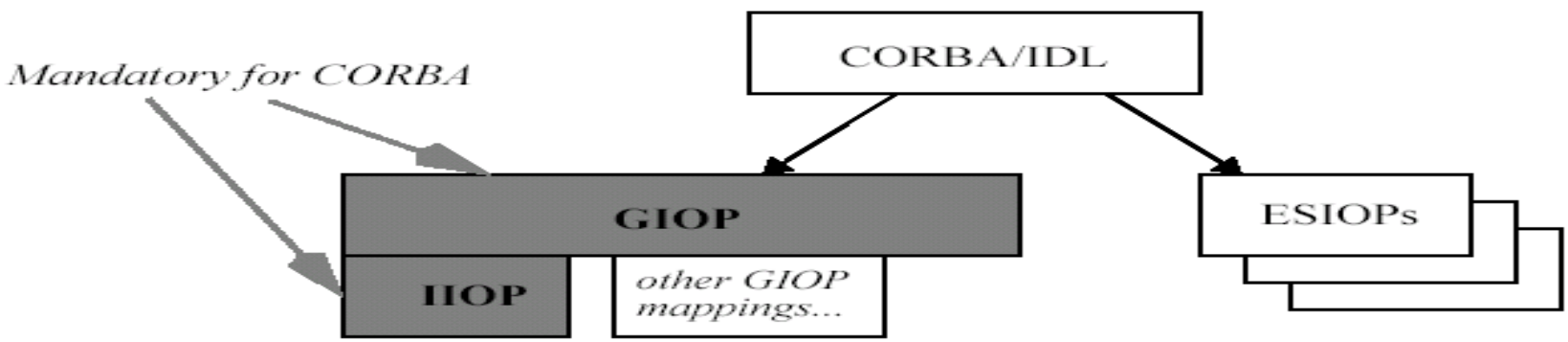
-One of the goals of the CORBA specification is that client and object implementations are portable.

- However, the reality of CORBA products today is that CORBA clients are portable, but object implementations need some rework to port from one CORBA product to another.

-Interoperability is more important in a distributed system than portability.

-CORBA 2.0 added interoperability as a goal in the specification, under the form of interoperability protocols:

- Inter-ORB Bridge Support
- General Inter-ORB Protocol (GIOP)
- Internet Inter-ORB Protocol (IIOP)
- Environment-Specific Inter-ORB Protocols (ESIOPs)



Inter-ORB Bridge Support

- The interoperability architecture clearly identifies the role of different kinds of domains for ORB-specific information.
  - Where two ORBs are in the same domain, they can communicate directly.
  - When information in an invocation must leave its domain, the invocation must traverse a bridge.
- The role of a bridge is to ensure that content and semantics are mapped from the form appropriate to one ORB to that of another, so that users of any given ORB only see their appropriate content and semantics.

General Inter-ORB Protocol (GIOP)

- The General Inter-ORB Protocol (GIOP) specifies a standard transfer syntax (low-level data representation) and a set of message formats for communications between ORBs.
  - The protocol is specifically built for ORB to ORB interactions and is designed to work directly over any connection-oriented transport protocol that meets a minimal set of assumptions.
  - The protocol allows portable implementations with small memory footprints and reasonable performance, with minimal dependencies on supporting software other than the underlying transport layer.
  - While versions of the GIOP running on different transports would not be directly interoperable, their commonality would allow easy and efficient bridging between such networking domains.

Internet Inter-ORB Protocol (IIOP)

-GIOP (General Inter-ORB protocol) specifies how ORBs communicate, including how messages are sent, how parameters are marshaled for remote object invocations etc.

-IIOP (Internet Inter-ORB Protocol) is a TCP/IP implementation of GIOP.

- IIOP specifies a standardized interoperability protocol for the Internet, providing "out of the box" interoperation with other compatible ORBs based on the most popular product- and vendor-neutral transport layer.

- IIOP is used in other systems that do not even attempt to provide the CORBA API (e.g "RMI over IIOP", EJB etc.)
- Because they all use IIOP, programs written to these APIs can interoperate with each other and with CORBA programs.