UNIT – 4

# CORBA

# CORBA

- AN OBJECT ORIENTED RPC MECHANISM
- HELPS TO DEVELOP “DISTRIBUTED SYTEMS” IN DIFF. PLATFORMS
- OBJECTS WRITTEN IN DIFF., LANG, CAN BE CALLED BY OBJECTS WRITTEN IN ANOTHER LANG.
- CORBA CLIENTS CAN ACCESS EJB OBJECTS

# CORBA

## Common Object Request Broker Architecture

***"How can distributed Object Oriented systems implemented in different languages and running on different platforms interact?"***

**CORBA is the OMG's component model**

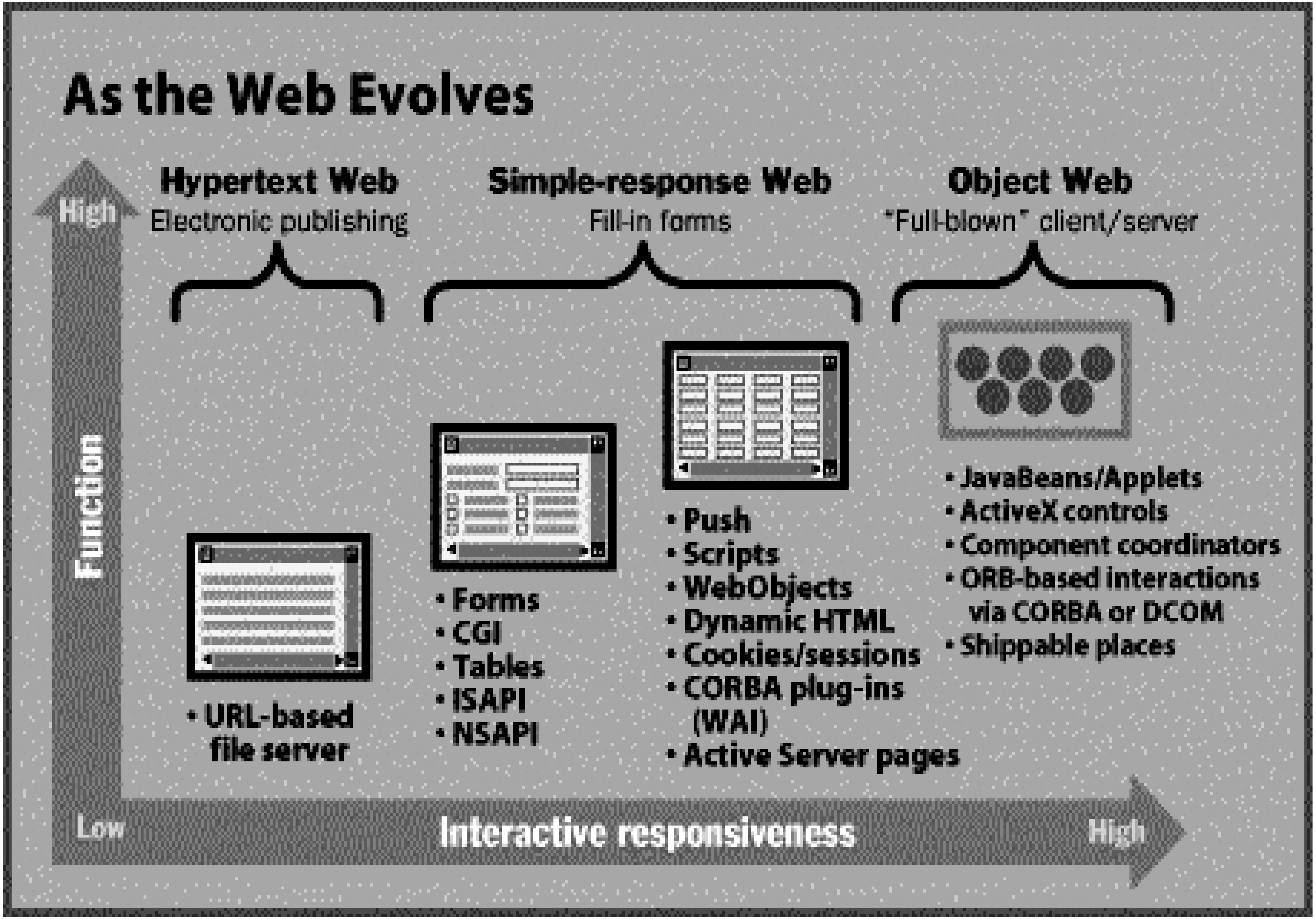
- IT IS AN PLATFORM & LANGUAGE INDEPENDENCE

IT IS WELL SUITED FOR OBJECT ORIENTED SYSTEMS

IDEA :

USE OF OO SYSTEMS RUNNIG DIFF. PLATFORMS & COMPILED IN DIFF., LANG .

AN INTERACTION BTWN THEM



## As the Web Evolves

**Hypertext Web**

Electronic publishing

- URL-based file server

**Simple-response Web**

Fill-in forms

- Forms
- CGI
- Tables
- ISAPI
- NSAPI

- Push
- Scripts
- WebObjects
- Dynamic HTML
- Cookies/sessions
- CORBA plug-ins (WAI)
- Active Server pages

**Object Web**

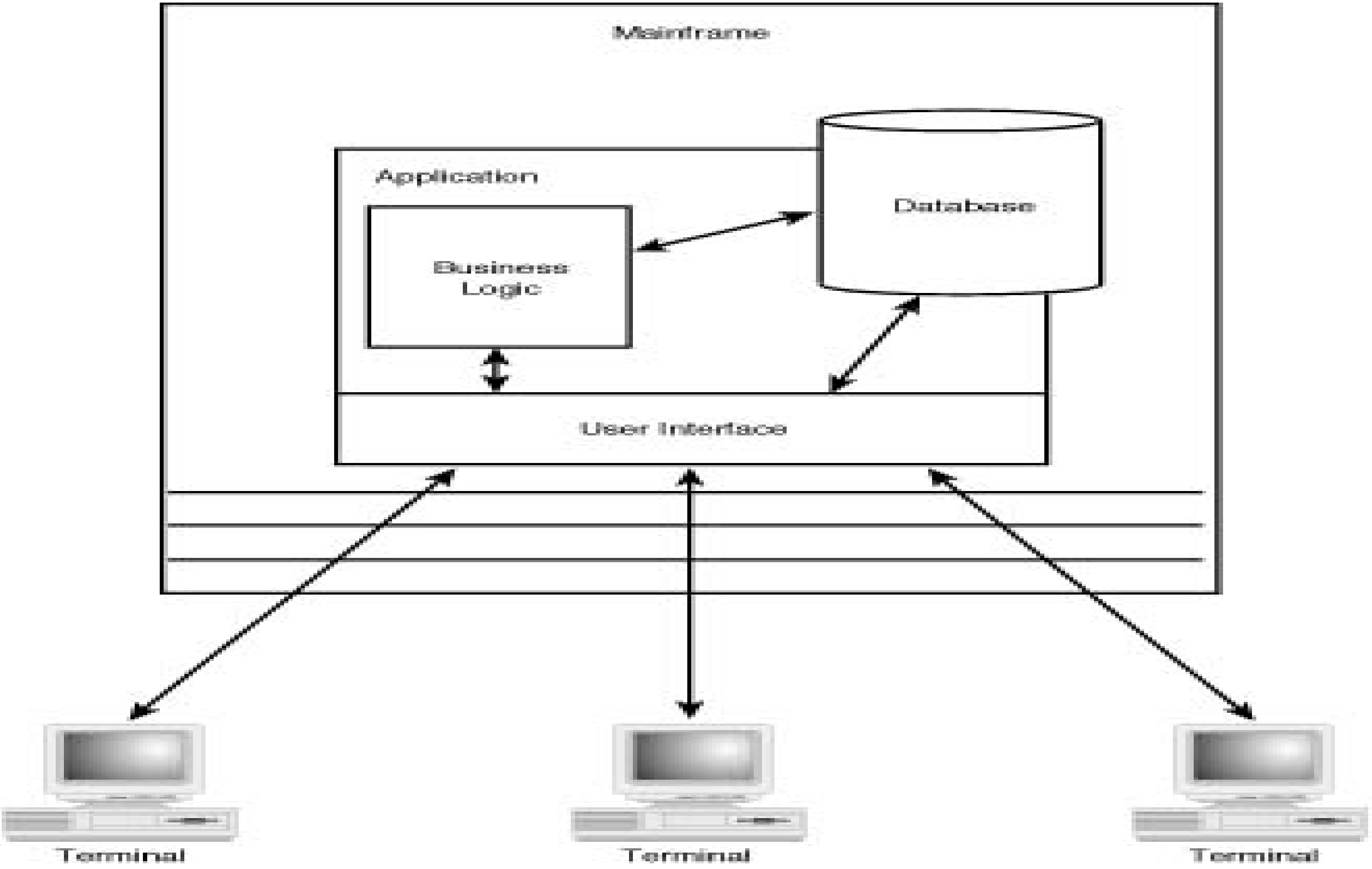
"Full-blown" client/server

- JavaBeans/Applets
- ActiveX controls
- Component coordinators
- ORB-based interactions via CORBA or DCOM
- Shippable places

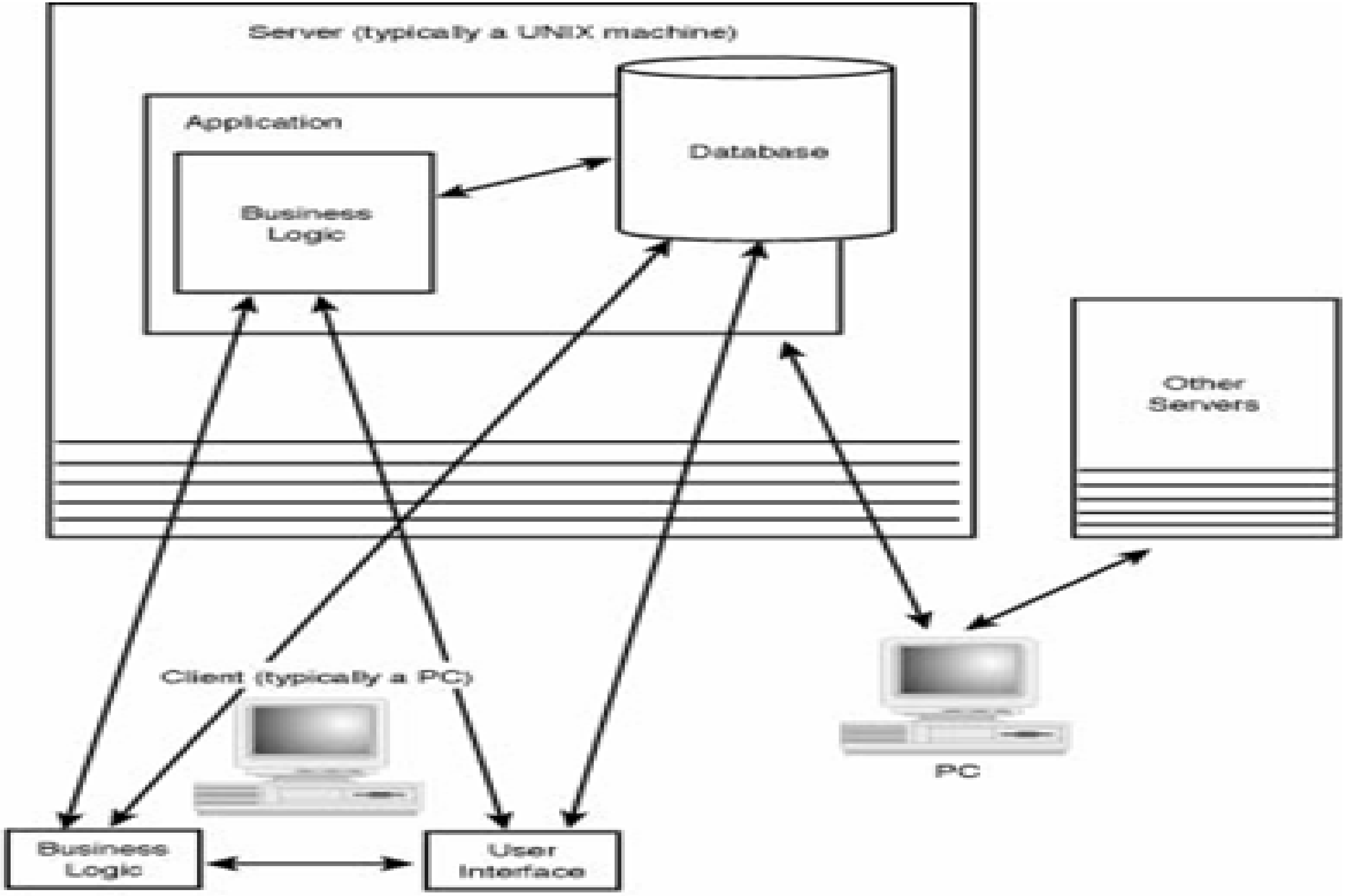
# HISTORY OF DISTRIBUTED SYSTEMS

- MONOLITHIC SYSTEMS & MAINFRAME
- 2TIER CLIENT/SERVER ARCHITECTURE
- MULTI TIER C/S (OR) DISTRIBUTED SYSTEMS

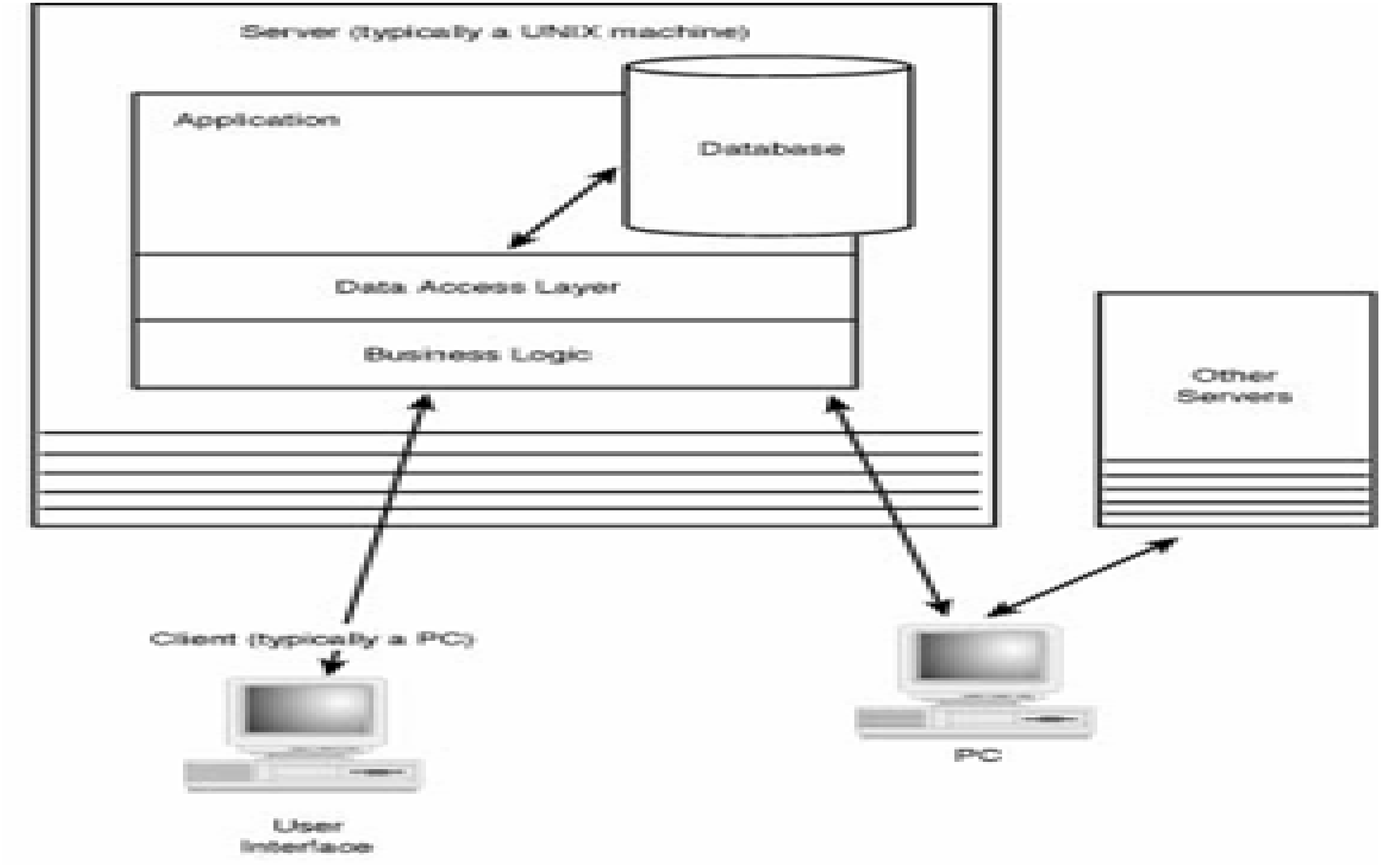
# Monolithic Systems and Mainframes



# Two-tier client/server architecture



# Multi-tier C/S – Distributed Systems



# PURPOSE

- CORBA provides a standard mechanism for defining the interfaces between components
- OMG provides directory and naming services, persistent object services, and transaction services
- Two features are *platform independence* and *language independence*

# CORBA alternatives

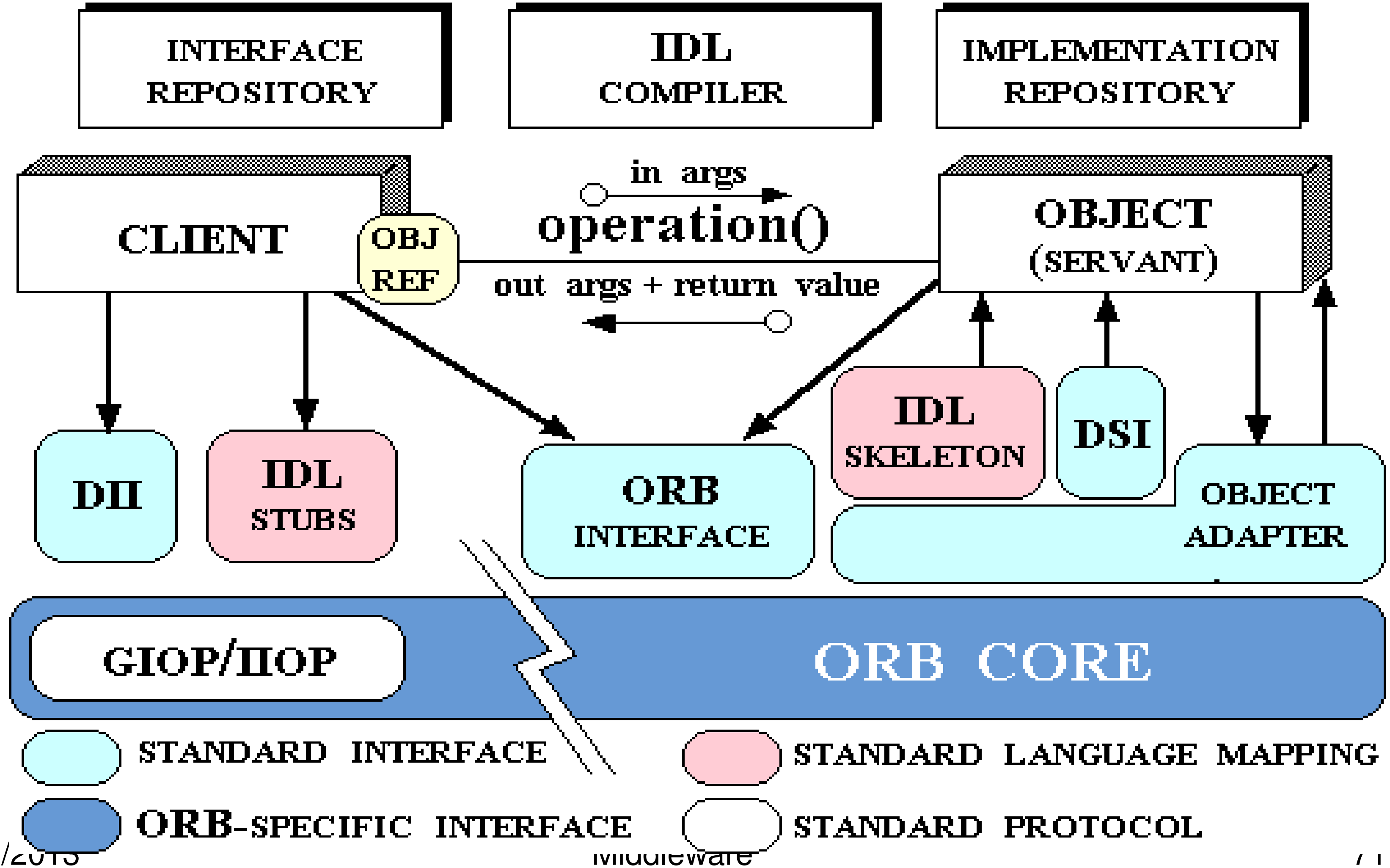
- **Socket Programming**
- **Remote Procedure Call**
    - **Function oriented interface**
- **OSF Distributed Computing Environment**
    - **OSF standard to RPC**
- **MS-DCOM**
    - **Restricted to only Microsoft technologies**
- **Java Remote Method Invocation**
    - **Passing of objects by value**
    - **Java only solution**

# Architecture overview

An Object Oriented architecture

- Object Request Broker
- Interface Definition Language
- Communications Model
- Clients and Servers, Stub and skeleton

# CORBA ARCHITECTURE



# Object Request Broker (ORB)

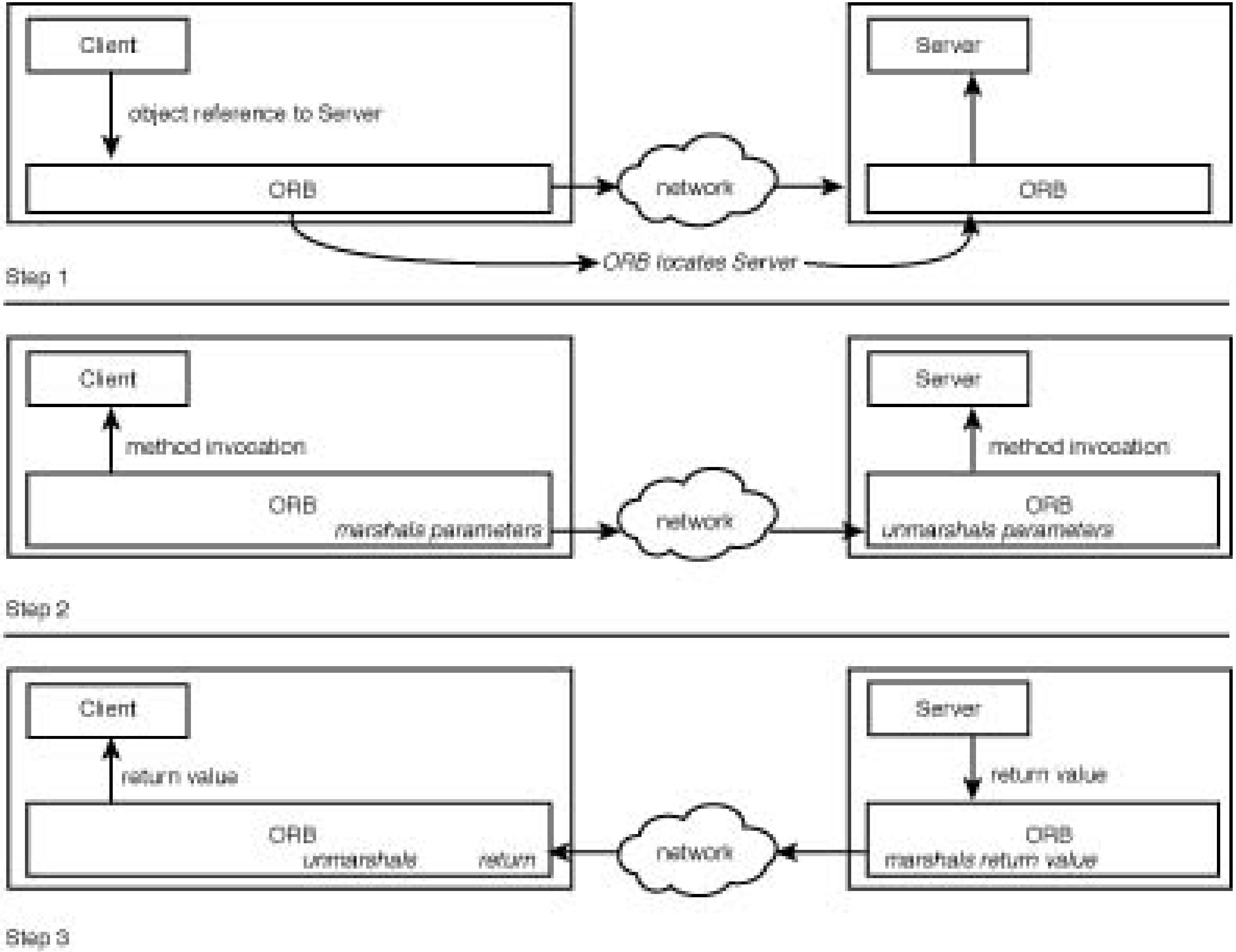
- Facilitates communication between objects
- Locates a remote object upon a reference
- Marshals and un-marshals parameters
- Platform independent  format

- The concept of ORB is-When an application component wants to use a service provided by another component, it first must obtain an object reference for the object providing that service.

1. According to the OMG, **"CORBA allows applications to communicate with one another no matter where they are located or who has designed them."**
2. This standard also defined the Interface Definition Language (IDL) and the Application Programming Interface (API) that makes client/server object interactions work in a specific implementation of an ORB.

The OMG ORB model can be divided into two main parts:

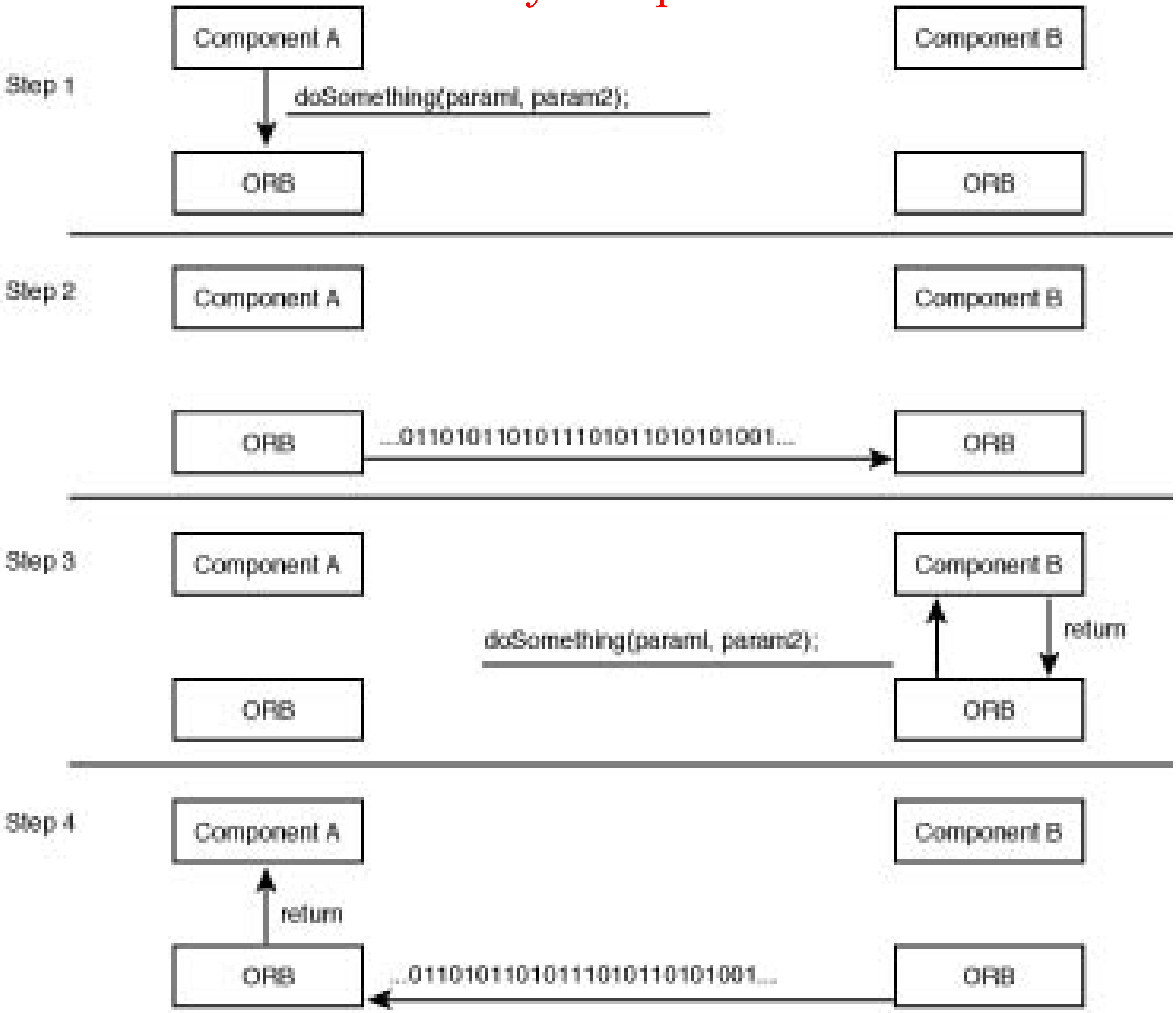
- Application oriented components
  - Application Interfaces
  - Domain Interfaces
  - Common Facilities
- System oriented components
  - Object Request Broker
  - Object Services


Client
object reference to Server
ORB
network
Server
ORB
ORB locates Server
Step 1
Client
method invocation
ORB
marshals parameters
network
Server
method invocation
ORB
unmarshals parameters
Step 2
Client
return value
ORB
unmarshals
return
network
Server
return value
ORB
marshals return value
Step 3

4/

***Marshaling*** refers to the process of translating input parameters to a format that can be transmitted across a network.

***Unmarshaling*** is the reverse of marshaling; this process converts data from the network to output parameters.

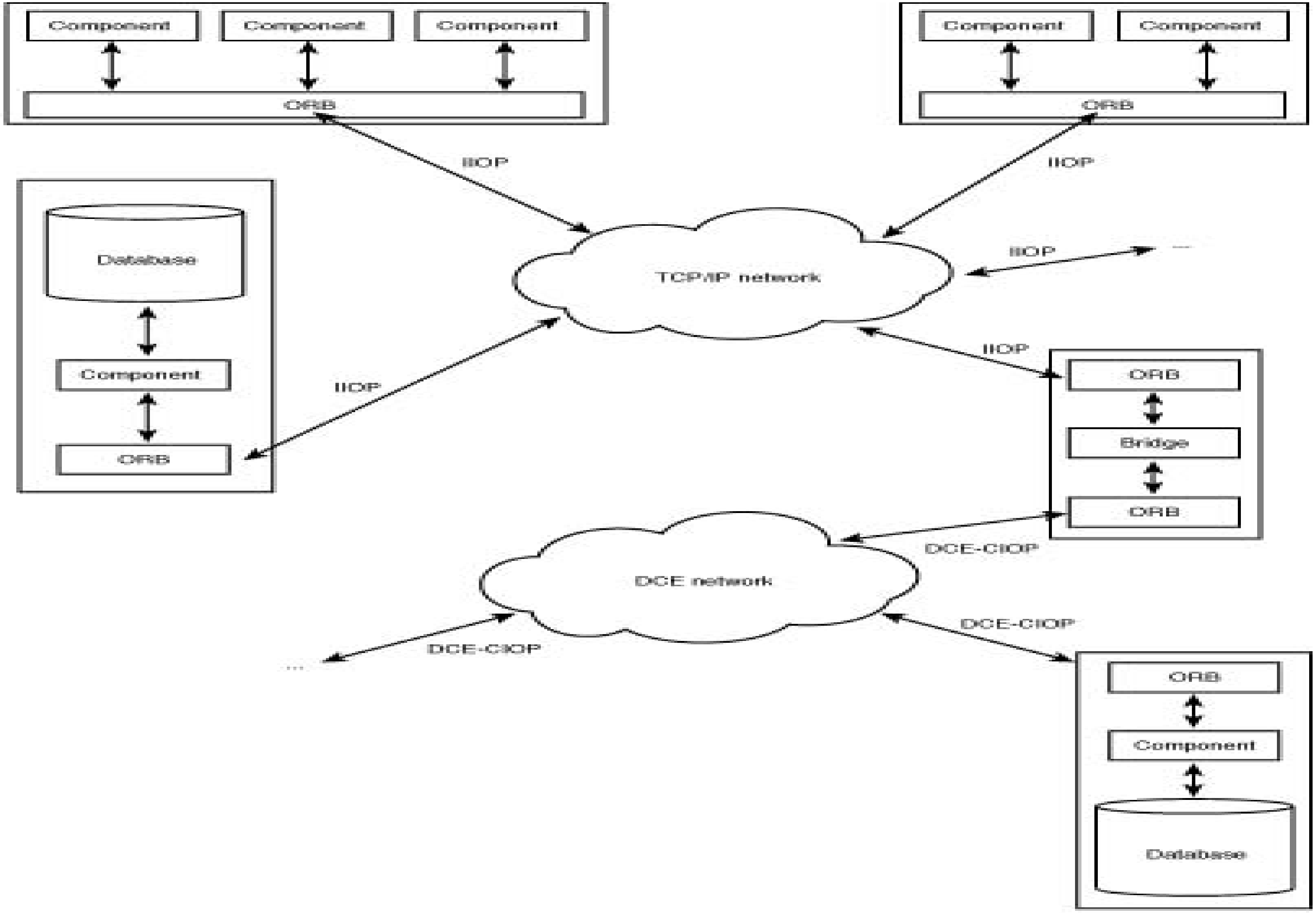

Step 1
Component A
Component B
doSomething(paraml, param2);
ORB
ORB
Step 2
Component A
Component B
ORB
...0110101101011101011010101001...
ORB
Step 3
Component A
Component B
doSomething(paraml, param2);
return
ORB
ORB
Step 4
Component A
Component B
return
ORB
...0110101101011101011010101001...
ORB

# CORBA Networking( COMMUNICATIONS) Model

- network model
- Object model

# N/W MODEL

- Inter-ORB protocols
  - ❖General Inter-ORB Protocol (GIOP)
  - ❖Internet Inter –ORB Protocol (IIOP)
- Applications are built on top of IIOP
- GIOP protocols rest on TCP/IP, DCE protocols
- New layer – ORB Protocol layer


Component
Component
Component
ORB
Component
Component
ORB
IIOP
IIOP
Database
Component
ORB
TCP/IP network
IIOP
...
IIOP
IIOP
ORB
Bridge
ORB
DCE-CIOP
DCE network
DCE-CIOP
DCE-CIOP
...
ORB
Component
Database

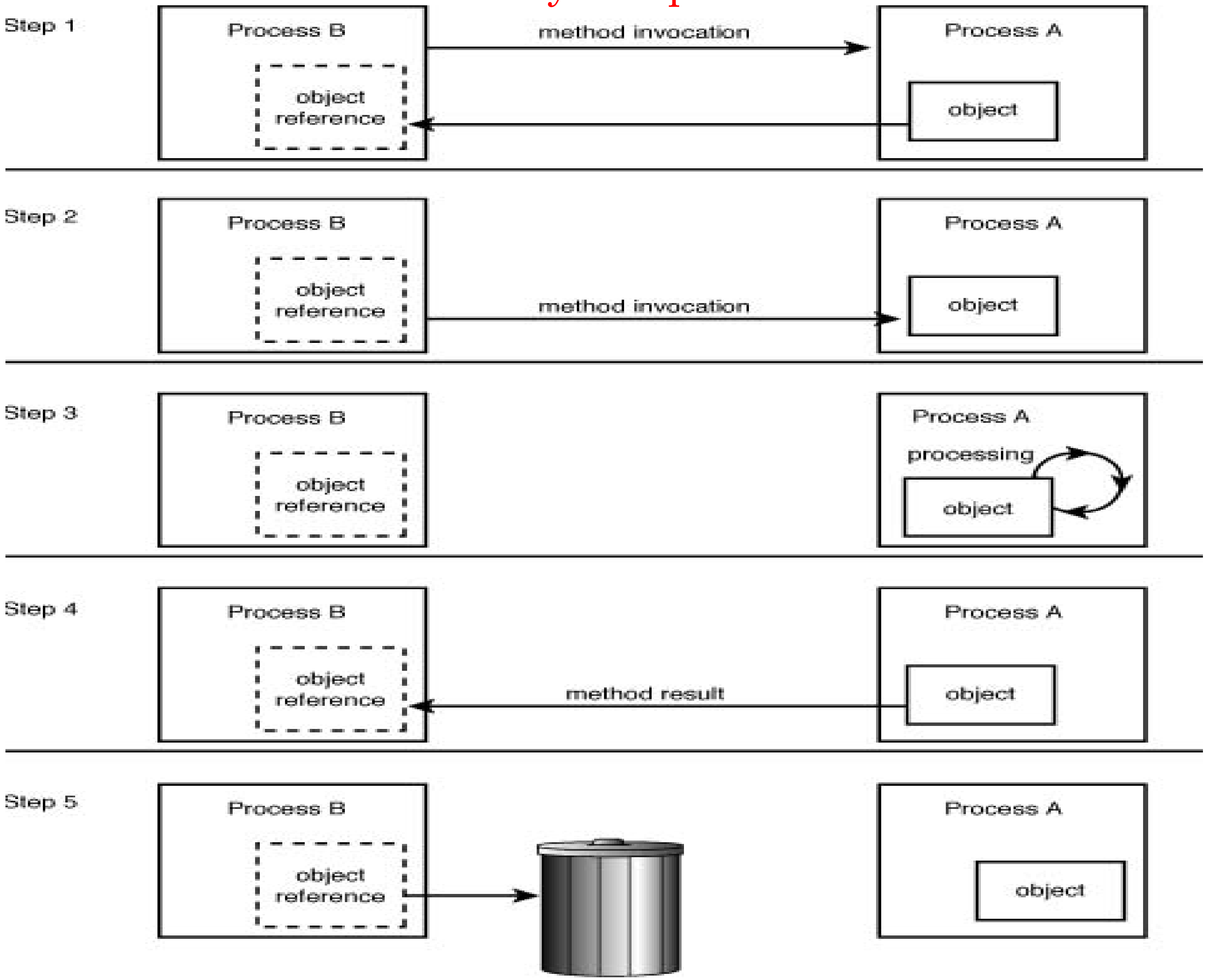
# CORBA Object Model

- Object Distribution
  - ✓ A CORBA client, a remote method call looks exactly like a local method call
- Object Reference
- Basic Object Adapters

# OBJECT REFERENCE

- 2 possible ways
  - ➢ call by value
  - ➢ Call by reference

OMG’S ORB USES CALL BY REFERENCE


Step 1
Process B
object reference
method invocation
Process A
object
Step 2
Process B
object reference
method invocation
Process A
object
Step 3
Process B
object reference
Process A
processing
object
Step 4
Process B
object reference
method result
Process A
object
Step 5
Process B
object reference
Process A
object

# BASIC OBJECT ADAPTERS

- purpose :To interface an object's implementation with its ORB.
- PROVIDES 3 object ADAPTER types
  - ✓ Basic Object Adapters

    provides set of methods for accessing ORB functions
  - ✓Library Object Adapters
  - ✓OO Database Adapters

    useful for accessing objects in persistent storage

# IDL

- Provides interface b/w various CORBA objects
- Is a generic language
- Has its own language constraints
- IDL is not a procedural language; **it can define only interfaces, not implementations**
- **as its name suggests, is the language used to define interfaces between application components**

# IDL Ground Rules

- **Case Sensitivity**
- **IDL Definition Syntax**
- **The Module**
- In addition to ,
  - Comments
  - Use of C preprocessor
  - Coupling & cohension

# PRIMITIVE TYPES

- Void
- Boolean
- Char & wchar
- Float
- Double & long types
- String
- Const modifier
- Integer types
  - Long & long long
- Unsigned ..,.,.,.
- Short
- Unsigned short
- octet

# Constructed types

- The Enumerated type
- The Structure type
- The Union type
- Term : A *discriminator,* as used in an IDL union, is a parameter that determines the value used by the union.
- THE INTERFACE TYPE and so on

# Building a CORBA Application

The outline of the process is this:

**1**. Define the server's interfaces using IDL.

**2**. Choose an implementation approach for the server's interfaces

**3**. Use the IDL compiler to generate client stubs and server skeletons for the server interfaces

**4**. Implement the server interfaces.

**5**. Compile the server application.

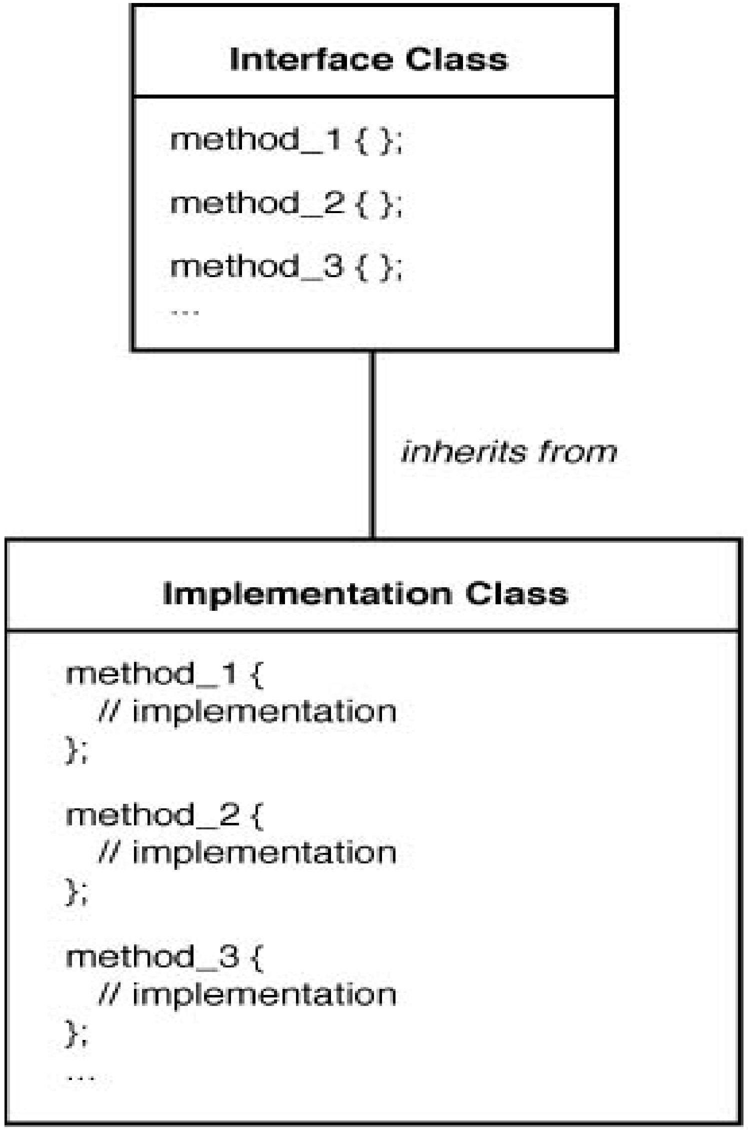
**6**. Run the server application.

# Defining the Server Interfaces
# ( Define Interface as an IDL file )

-
  ```
  module SimpleStocks {
   interface StockMarket {
    float get_price( in string symbol );
   };
  };
  ```

# Choosing an Implementation Approach

- **Inheritance mechanism,** in which a class implements an interface by inheriting from that interface class
- The **delegation mechanism**, in which the methods of the interface class call the methods of the implementing class (delegating to those methods).



Interface Class

method_1 { };

method_2 { };

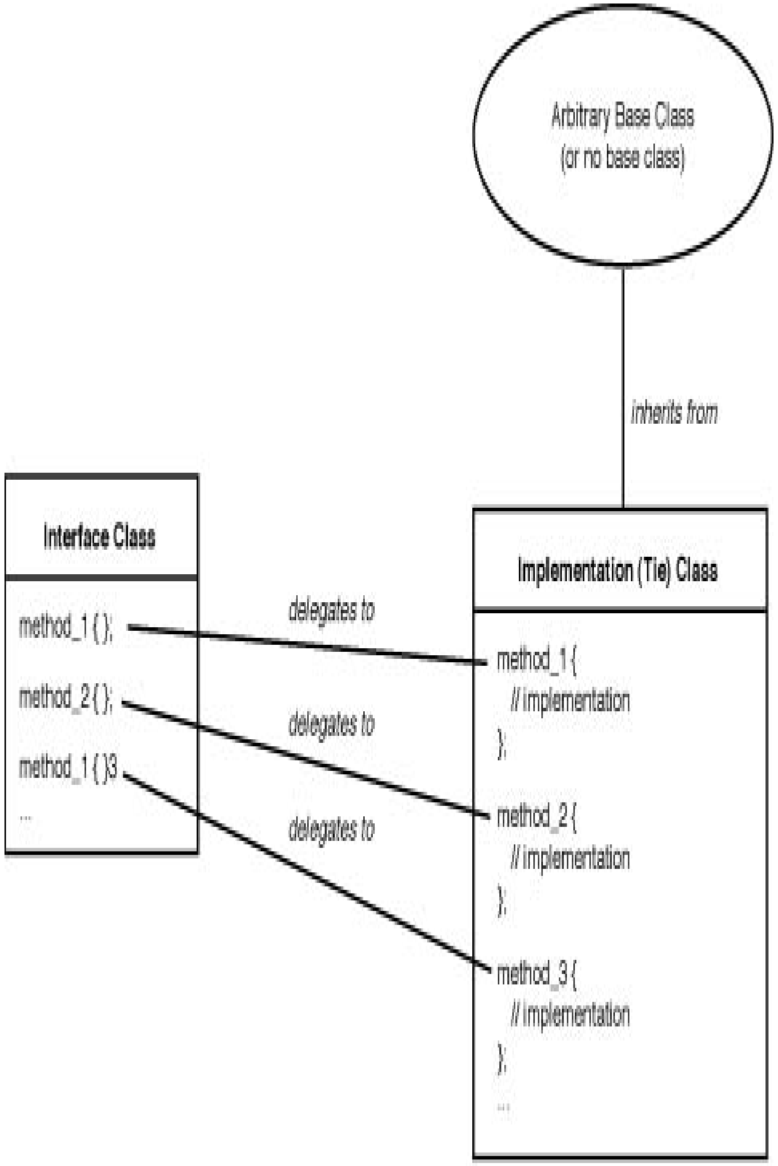
method_3 { };

...

inherits from

Implementation Class

```
method_1 {
   // implementation
};
method_2 {
   // implementation
};
method_3 {
   // implementation
};
...
```



Arbitrary Base Class
(or no base class)

inherits from

Interface Class

method_1 { };

method_2 { };

method_1 { }3

...

delegates to

delegates to

delegates to

Implementation (Tie) Class

```
method_1 {
  // implementation
};
method_2 {
  // implementation
};
method_3 {
  // implementation
};
...
```

Use the IDL compiler to generate client stubs and server skeletons for the server interfaces.

When compiled with the **idltojava** compiler, the stubs and skeletons are placed in a package named after our **module** – **modulename**

**Eg., stockapp,weatherapp**

# Implement the server interfaces.

```
import org.omg.CORBA.*;     // All CORBA applications
    need these classes.
    import SimpleStocks.*;      // The package containing our
    stubs
public class StockMarketImpl extends
    _StockMarketImplBase {
  public float get_price( String symbol ) {
        float price = 0;
        for(int i = 0; i < symbol.length(); i++) {
          price += (int) symbol.charAt( i );
        }
        price /= 5;
        return price;
      }
  public StockMarketImpl() { super(); }
```

5. Implement and Compile the server application.

**6. Compile and Run the Client application.**