Middleware technologies (CORBA, RMI & DCOM distributed computing)

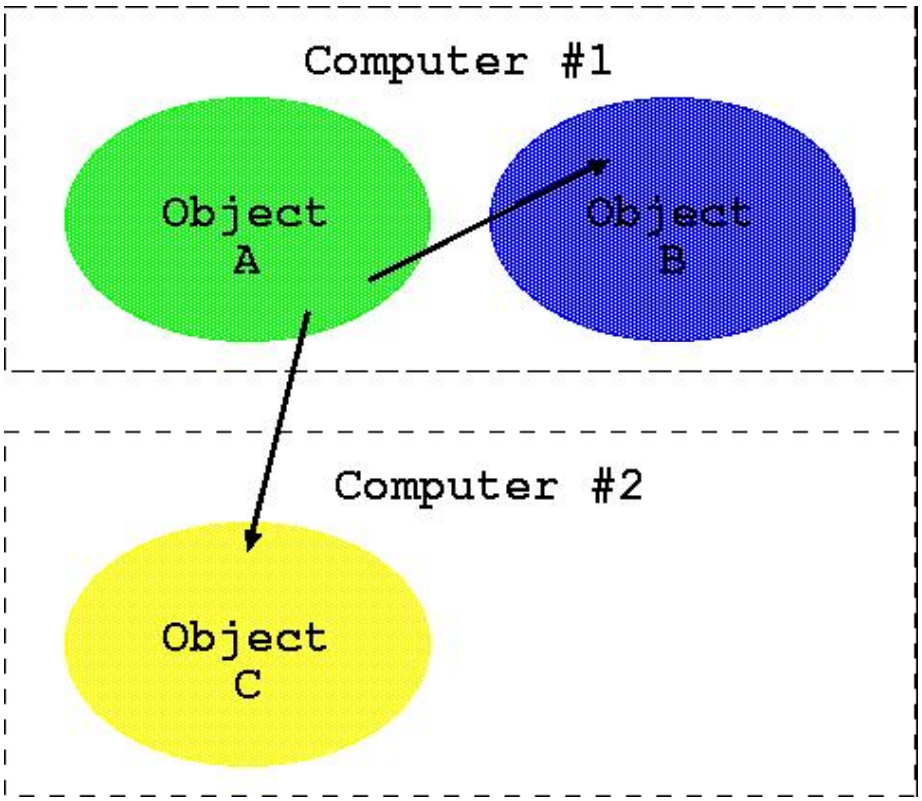
S.Chellammal

16-08-2013

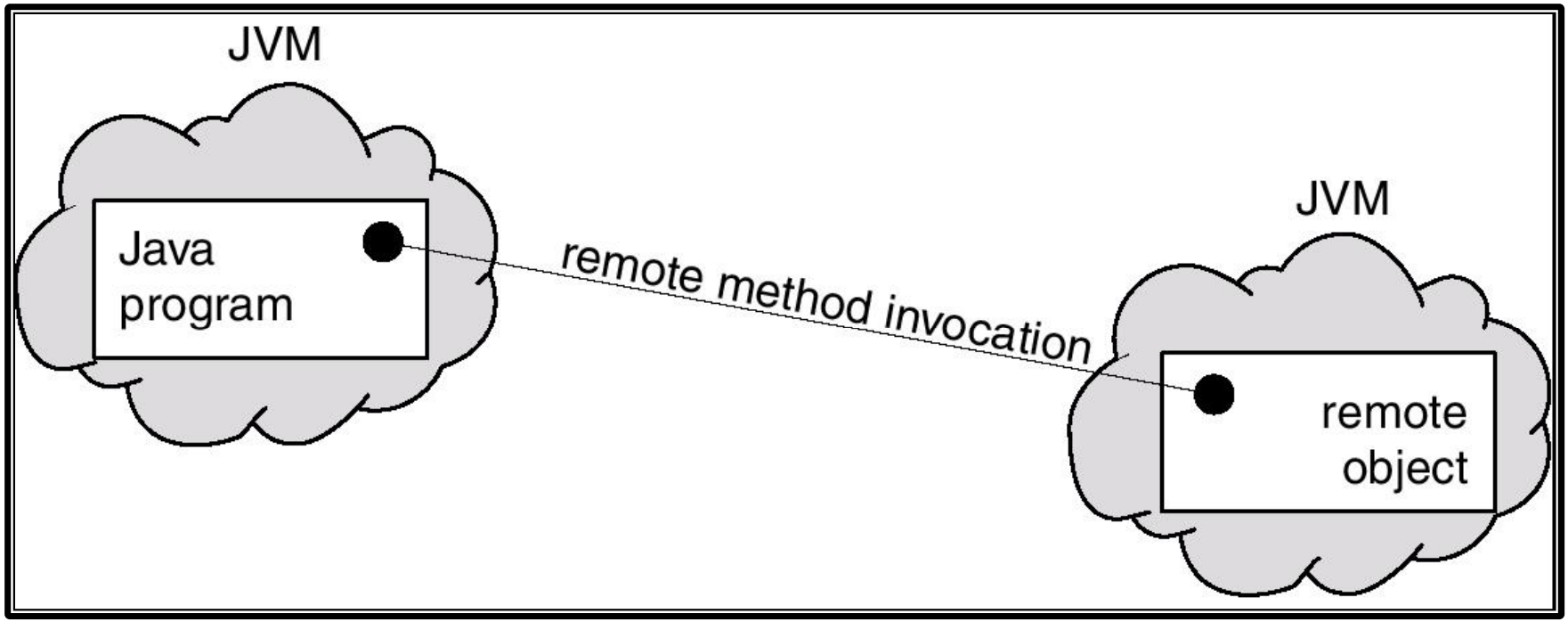
Distributed application/system

Distributed computing / **distributed system** is a software system in which components of the system are located on networked computers communicate and coordinate their actions by passing messages/RPC/message queues

Distributed object computing

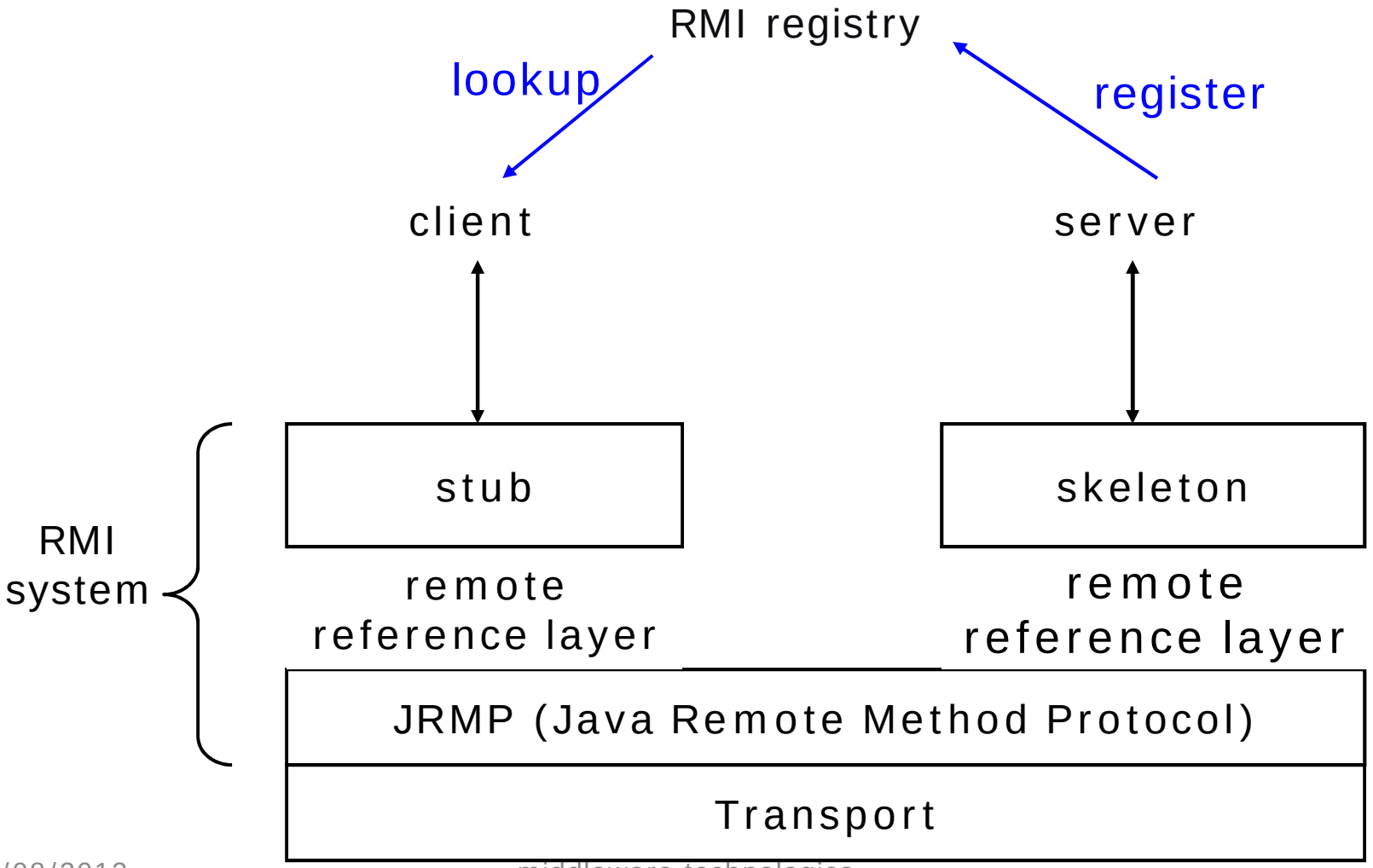


Remote Method Invocation

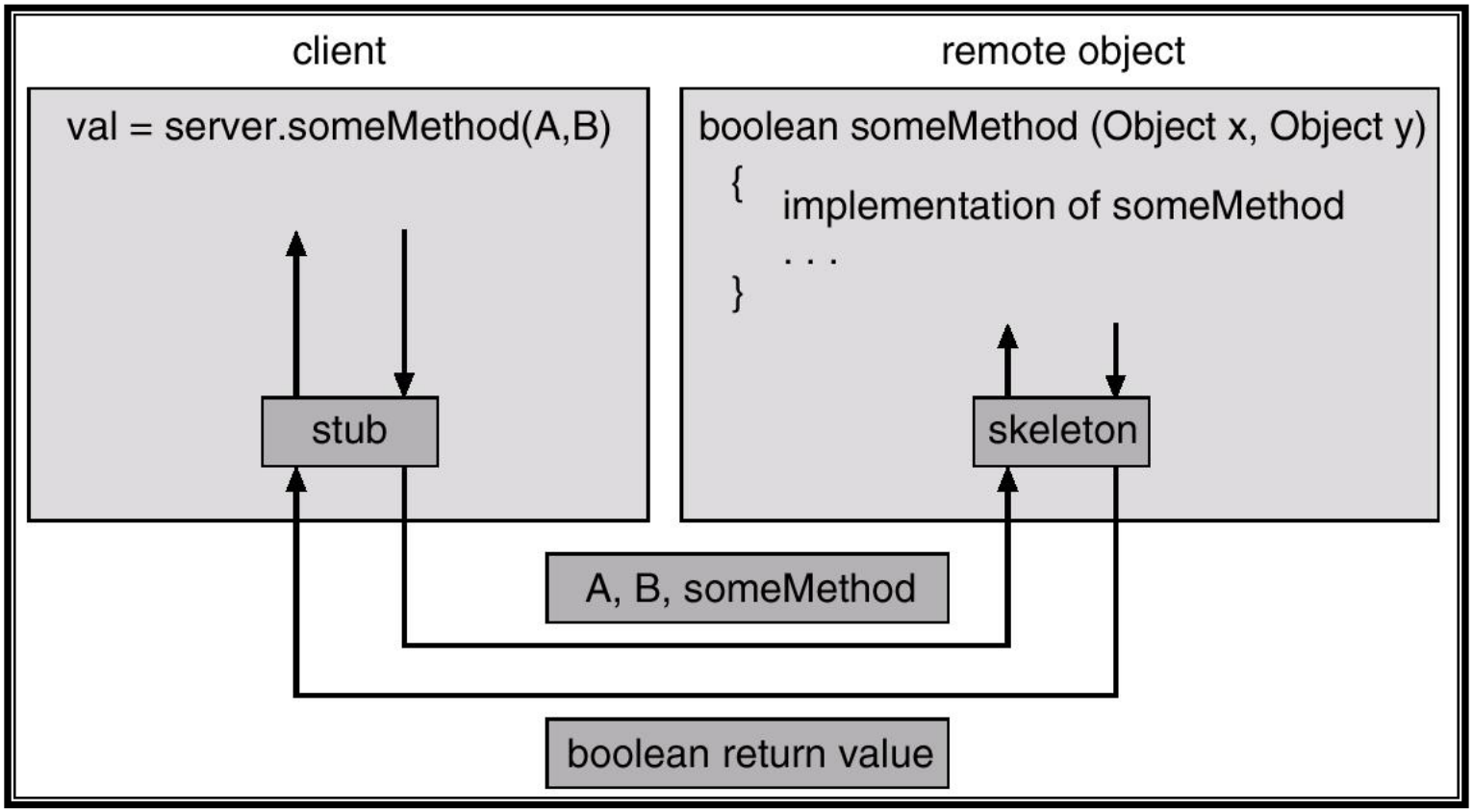


RMI allows a Java program on one machine to invoke a method on a remote object

RMI architecture



Stub & skeleton



Marshaling and UnMarshaling

Remote Reference Layer

RRL defines the invocation semantics

Connections: establishes connection between server and client

Stages:

1. server instantiates service
2. exporting to RMI
3. naming and registering

RMI Registry

Naming service & registry

Associates a unique name to server object

Steps to develop RMI

Step 1- Define remote interface

```
import java.rmi.Remote;
public interface Sum extends Remote
{
        public int add(int a, int b) throws RemoteException;
}
```

Step - 2

Implement the remote interface

```
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
public class SumImpl extends UnicastRemoteObject implements Sum
{
    SumImpl() throws java.rmi.RemoteException{}

    public int add(int a, int b) throws java.rmi.RemoteException
    {return a+b;}
}
```

Step - 3

- Compile interface
- Compile implementation
- Generate stub & skeleton using rmic
- copy interface file and stub file in client

Step – 4 Instantiate server & bind

```
import java.rmi.registry.LocateRegistry;
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.net.MalformedURLException;
public class RMIServer
{
    public static void main(String args[]) throws RemoteException,
    MalformedURLException
    {
            LocateRegistry.createRegistry(1099);
            Sum sum=new SumImpl();
            Naming.rebind("lookup_sum", sum);
            System.out.println("server ready");
    }
}
```

Step – 5 Develop client

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.NotBoundException;
import java.net.MalformedURLException;
public class RMIClient
{
    public static void main(String args[]) throws RemoteException, MalformedURLException,
    NotBoundException
    {
            Registry registry=LocateRegistry.getRegistry("localhost");
            Sum s = (Sum)registry.lookup("lookup_sum");
            System.out.println("client just made a call");
            System.out.println(s.add(3,6));
    }
}
```

Execute

Run server

```
java RMIServer
```

Run Client

```
java RMIClient
```

RMI features

- RMI is object based
- It supports invocation of methods on remote objects.
- with RMI it is possible to pass objects as parameters to remote methods.
- Objects are passed by value with the help of object serialization

Middleware

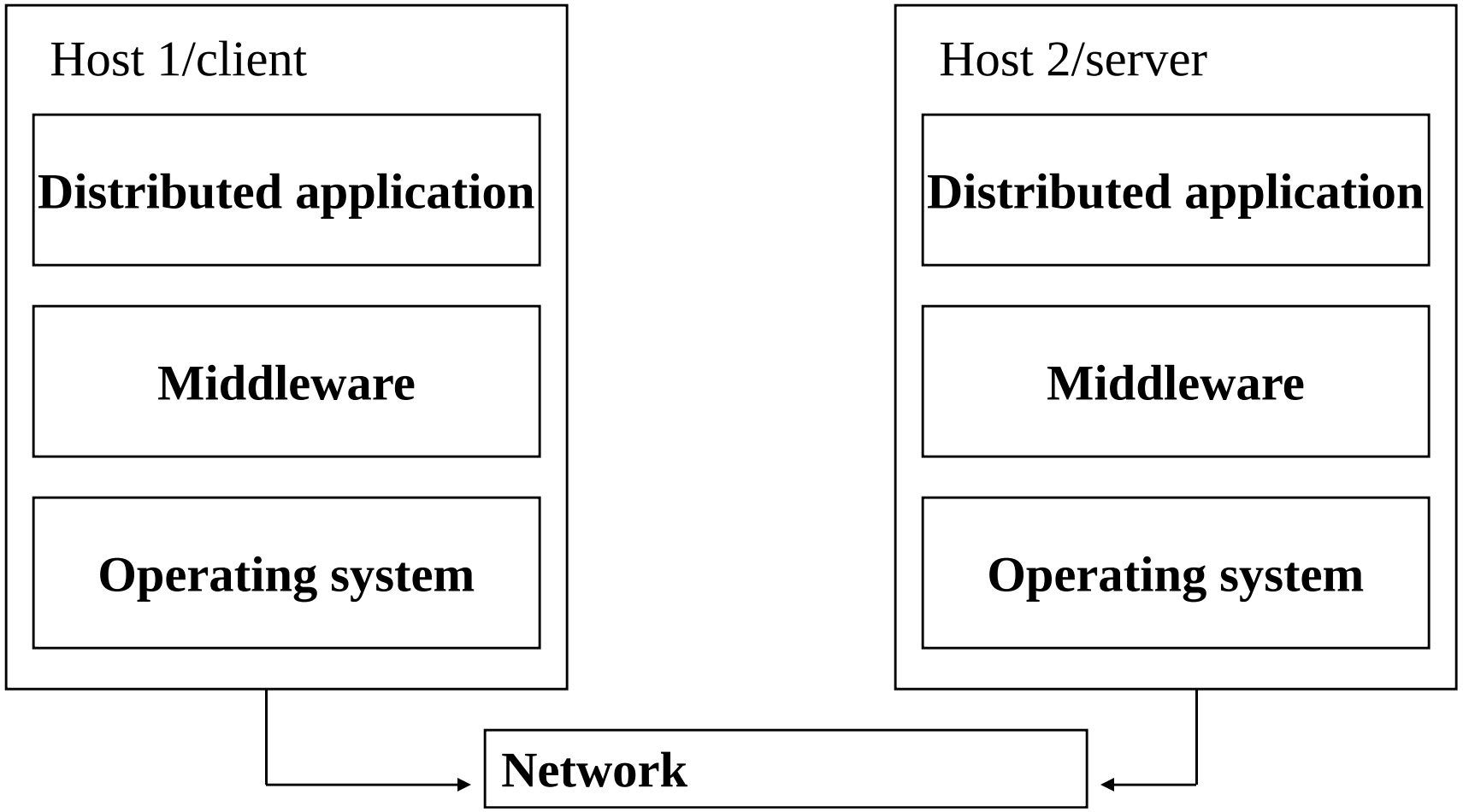
- Middleware is software that enables inter-process communication. It provides an API that isolates the application code from the underlying network communication formats and protocols (FAPs)

- Middleware acts as glue between autonomous components and processes (e.g., clients, server) by providing generic services on top of the OS.

Middleware

- Middleware enables applications running across multiple platforms to communicate with each other .
- Middleware shields the developer from dependencies on Network Protocols, OS and hardware platforms.
- Middleware is a software layer that lies between the operating system and the applications on each site of the system.

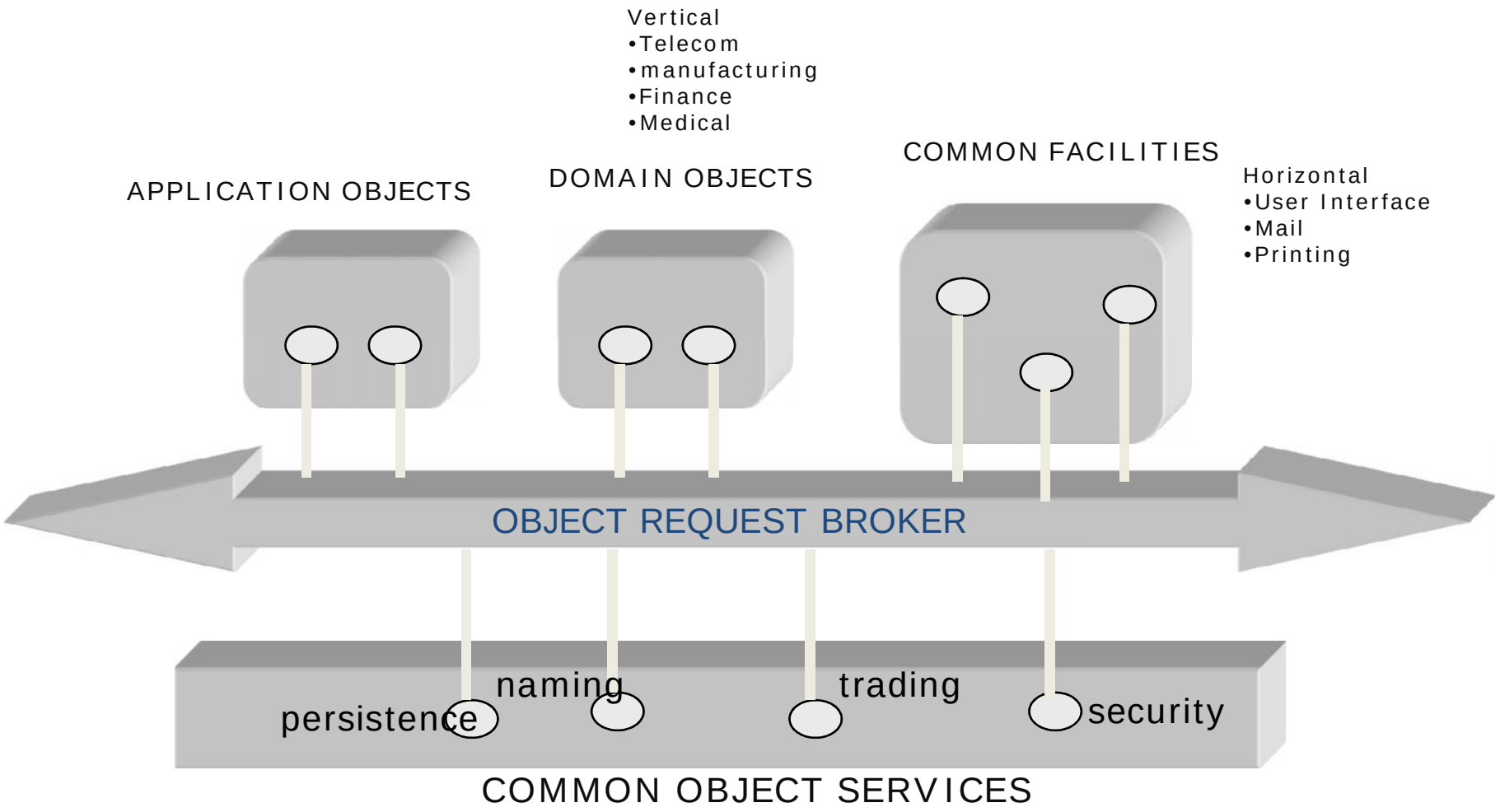
Middleware


Host 1/client
Distributed application
Middleware
Operating system
Host 2/server
Distributed application
Middleware
Operating system
Network

OMG & CORBA

- The Object Management Group consisting of over 600 companies evolved the CORBA specs :Since 1989
- CORBA is a specification for the distributed object bus architecture defined by OMG

Object Management Architecture



Object Management Architecture

- Reference model for distributed object computing
- Specifies the components/architecture
- OMA defines two models
    - Core Object Model
    - Reference model

Core object model

portability (to be able to create components that don't rely on existence and location of a

particular object implementation)

interoperability (to be able to invoke operations on objects regardless of where they are located, on which platform they execute, or in which

programming language they are implemented

Core Object Model contd.

is a classic object model (invoking operations on objects)

– Objects

– Operations

- Signatures
- Parameters
- return values –
- Non-object types (data types) –
- Interfaces
- – Substitutability When two interfaces can act as substitutes to each other

OMA reference model

Reference model defines 5 components

- ORB
- CORBA Services / CORBA Object Services (COS)/ Common Object Service Specification (COSS)
- Common facilities
- Domain Objects
- Application Objects

ORB chief responsibilities

- making the location transparency ie. routing the request from a client to object and routing the reply to destination

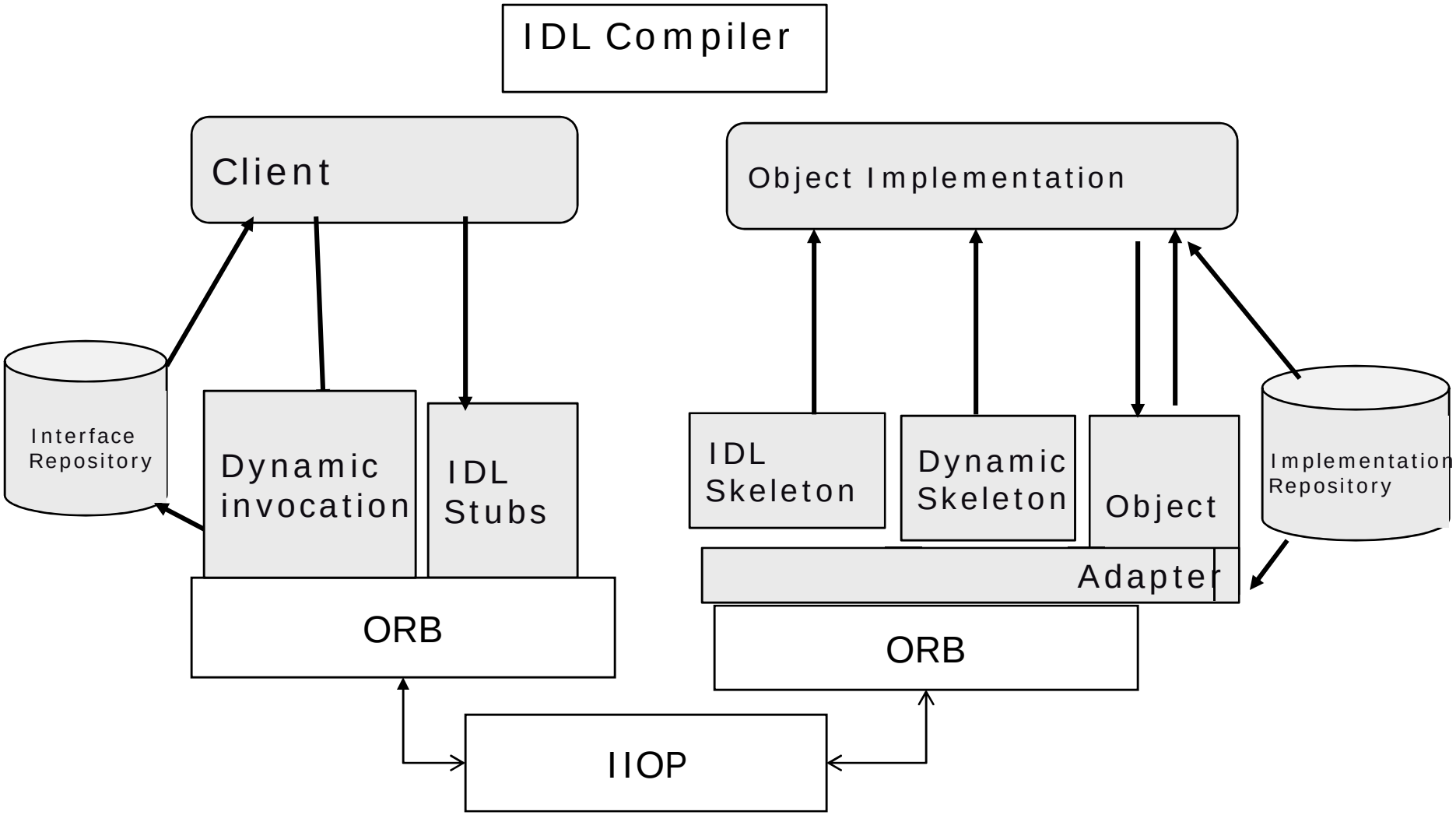
- management of the Interface Repository;
  a distributed database of IDL definitions
- Client side services for converting remote object references to and from strings
- Client side dynamic invocation of remote objects

- Server side resource management that is
- activation and deactivation of objects

ORB Components

- Client Stubs
- Server Skeletons
- Portable Object Adapter (POA)
- Dynamic Invocation Interface (DII)
- Dynamic Skeleton Interfaces (DSI)
- Interface Repository
- Implementation Repository

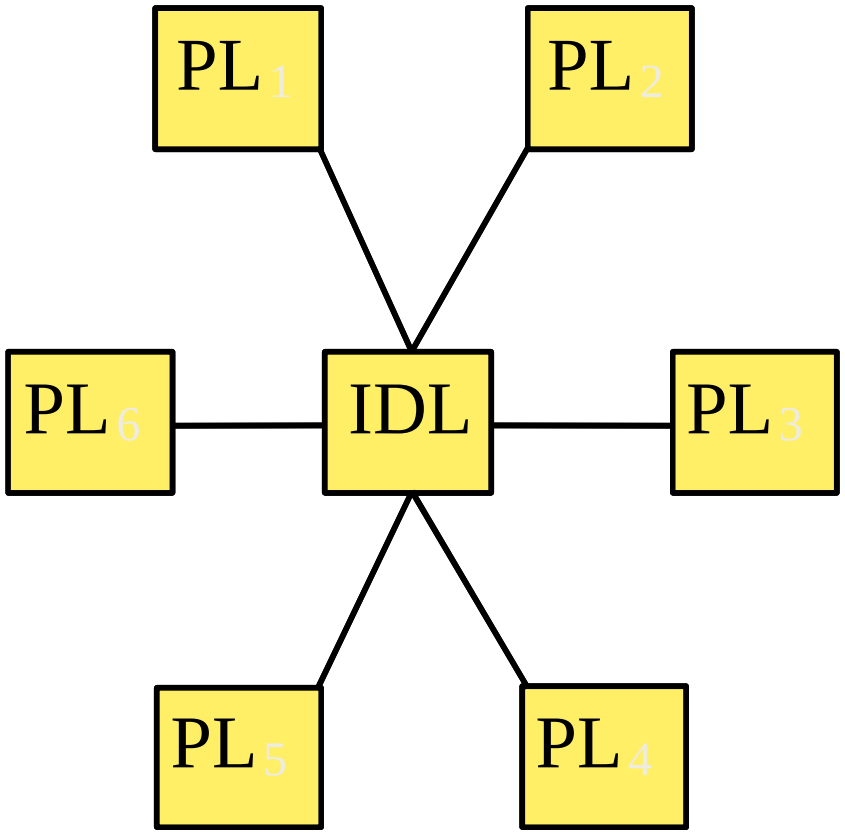
CORBA Architecture



Feature – 1 : Object Request Broker

- **Responsible for all communication between client and server**
  - Locating objects
    - Implementation specific
    - Known IOR(Inter-Object Reference)
    - Naming and Trading Services(DSN-like)
  - Transferring invocations and return values
  - Notifying other ORBs of hosted Objects
- **Must be able to communicate IDL invocations via IIOP**
- If an ORB is OMG compliant, then it is interoperable with all other OMG compliant ORBs
- **Interface Repository**
  - A Database of all of the IDL for compiled objects running on the ORB
- **Implementation Repository**
  - A Database containing policy information and the implementation details for the CORBA objects running on the ORB
- Load Balancing
- Fail-over support
- Security

Feature-2 : Interface Definition Language



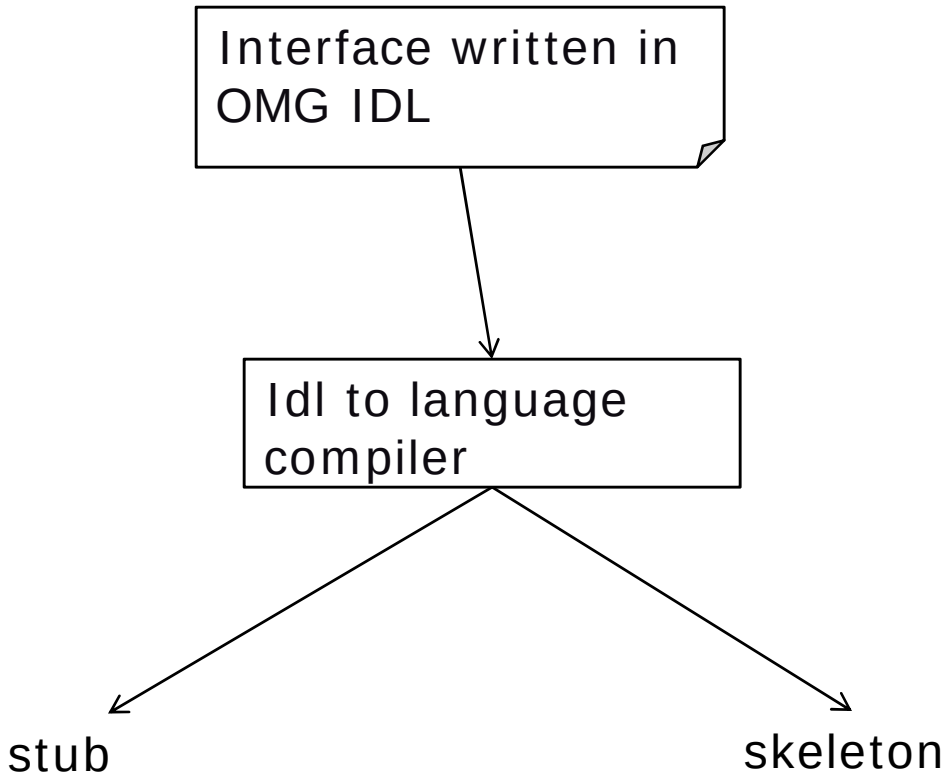
IDL

- IDL is used to define the public API that is exposed by objects in a server application. IDL defines this API in a way that is *independent* of any particular programming language.

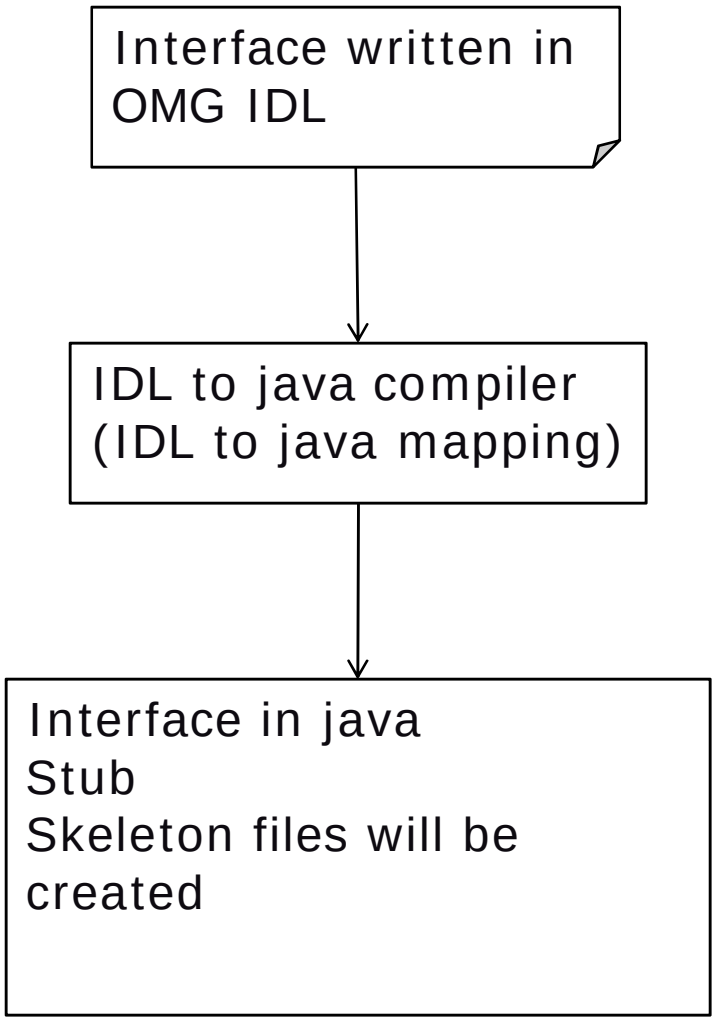
- However, for CORBA to be useful, there must be a mapping from IDL to a particular programming language.

- For example, the IDL-to-C++ mapping allows people to develop CORBA applications in C++

IDL to language compiler



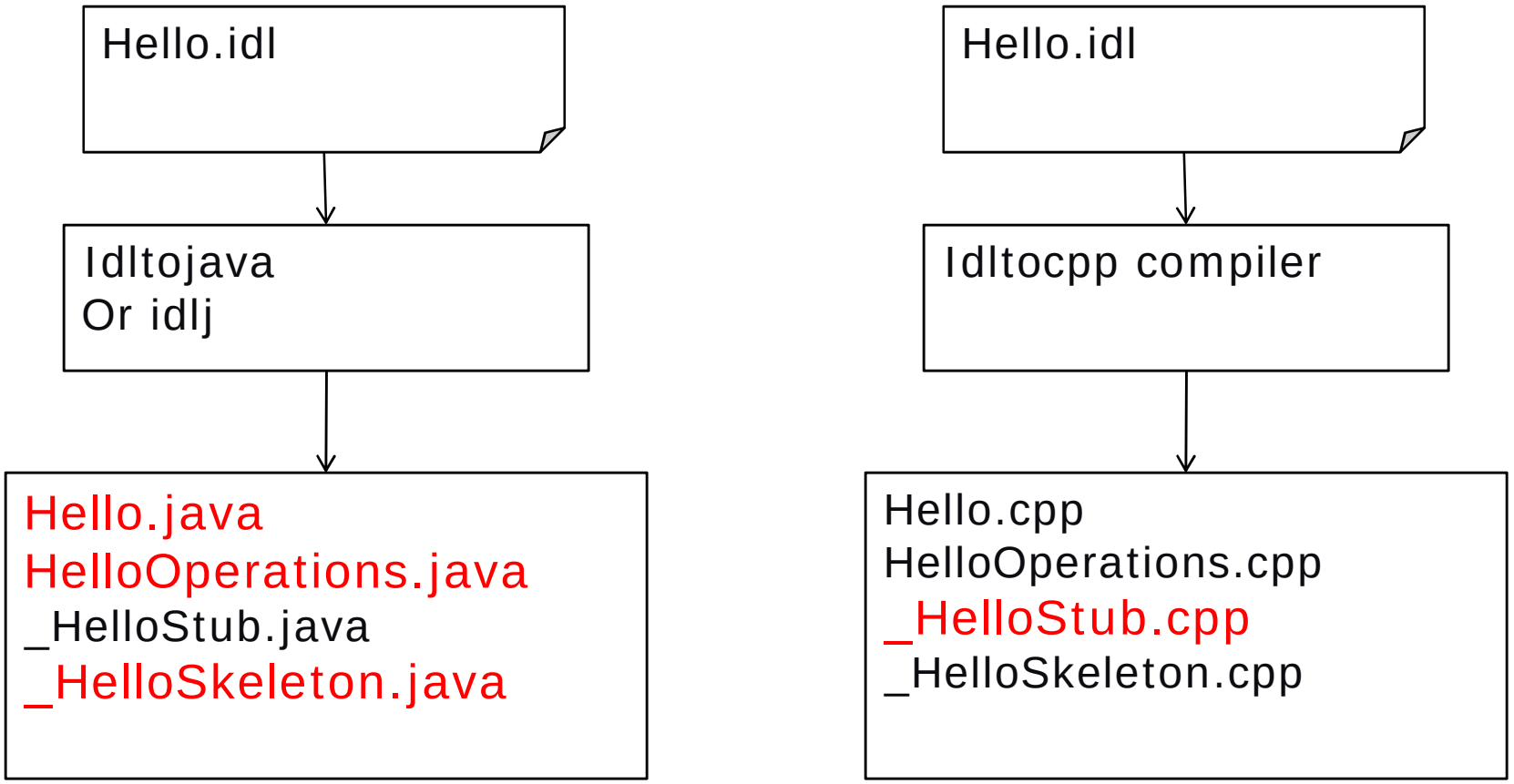
IDL Skeleton



Generated from an OMG IDL compiler to the server language of choice

- Unmarshals request data;
- Dispatch request to servant;
- Marshals reply data
- Servant is the actual CORBA object implementation in a chosen programming language
- Skeletons are used by the POA

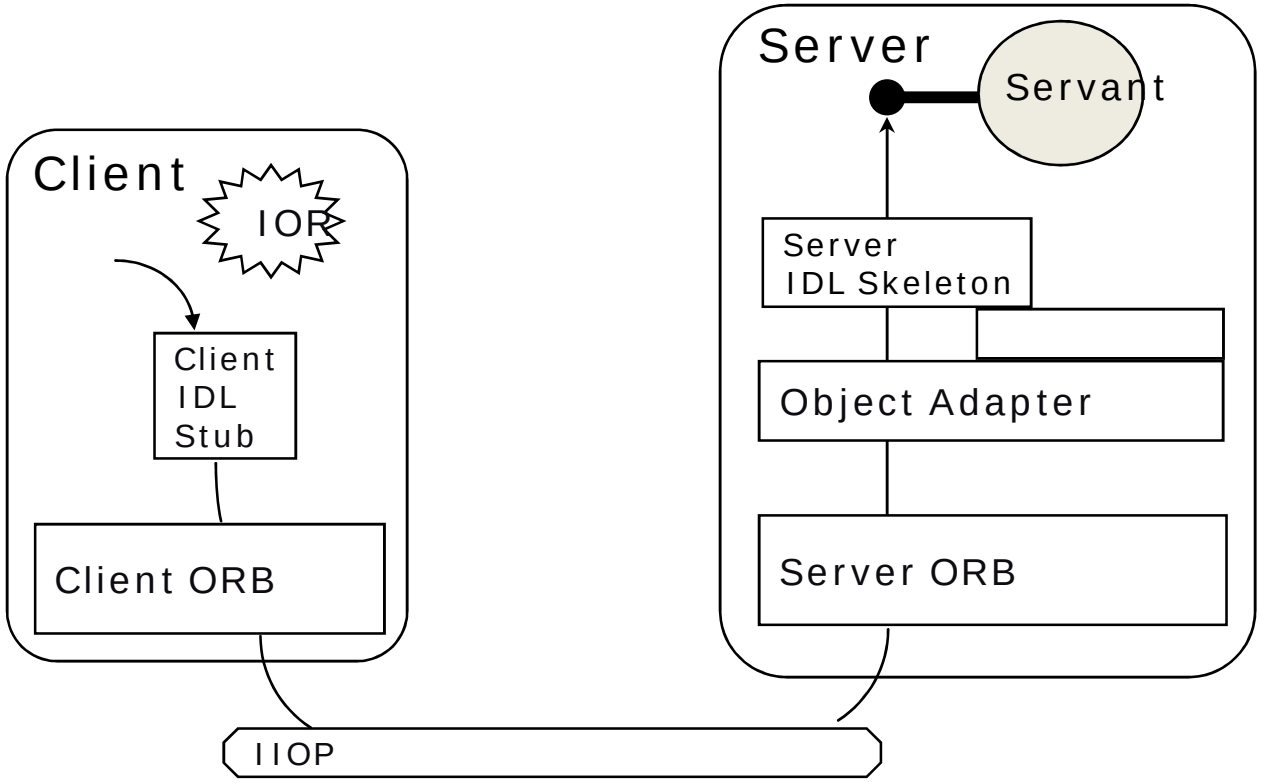
Achieving language independence



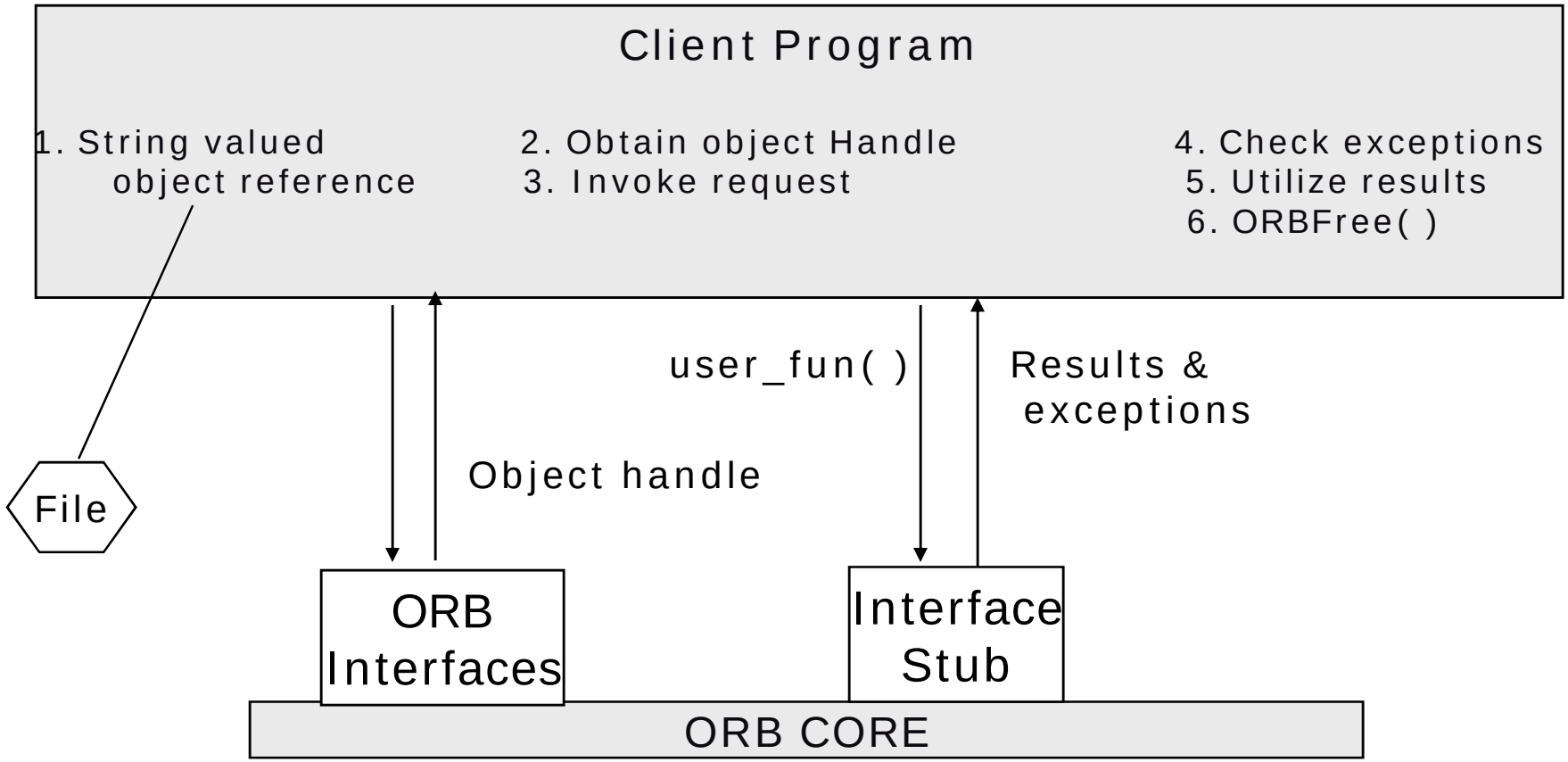
Functions of stubs and skeleton

- Refers to the process of translating input parameters/return values to a format in which it can be transmitted over the network.
- Unmarshaling is the reverse of marshaling.
- Stubs and skeletons contain code for marshaling and unmarshaling.

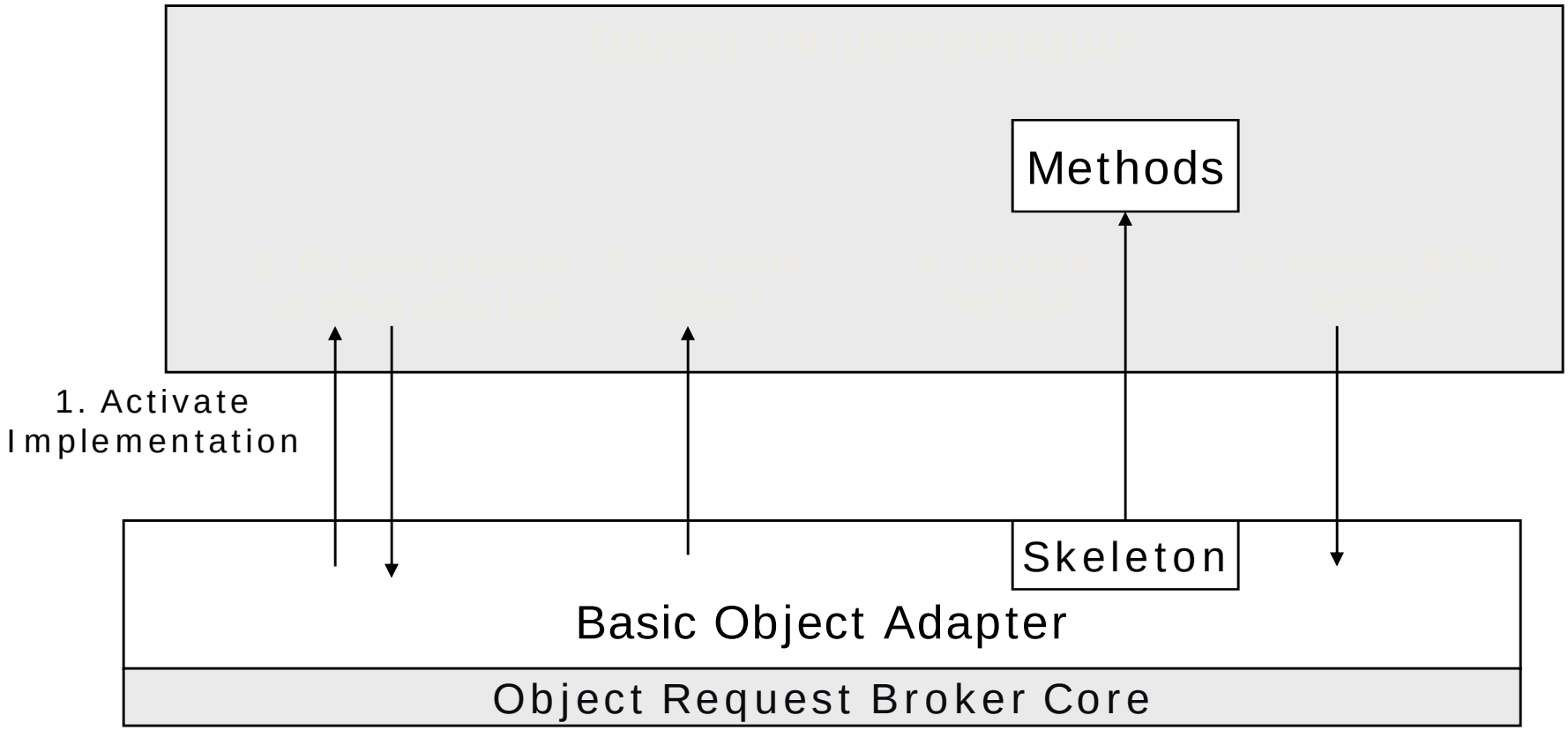
Static Invocation



Client Invocation Process Scenario



Object Implementation Invocation Scenario



Dynamic Invocation Interface

- Uses interface repository at run-time to discover interfaces.
- No need of pre-compiled stubs.
- Steps –
  - Obtain IOR of the interface name and get a reference to an object in the interface repository.
  - Obtain the method description.
  - Create the request to be passed.
  - Invoke the operation/method.

Dynamic Skeleton Interface

- Allows the ORB and OA to deliver requests to an object without the need of pre-compiled skeletons.

- Implemented via a DIR (Dynamic Invocation Routine).
- ORB invokes DIR for every DSI request it makes.

Dynamic Skeleton Interface

- OA up-calls the DIR servant and provides the request information (targeted object and operation name etc.).
- DIR asks IOR for the interface name of the targeted object and gets the meta-data information about it from interface repository.
- Creates the request to targeted object, using other parameters from the received request.
- Locates the Servant and send the request to it.
- Takes the return parameters and give them back to OA.

Object Adapter

- Different kind of object implementations -
  - objects residing in their own process and requiring activation.
  - others not requiring activation.
  - or some residing in same process as ORB.

- OA helps the ORB to operate with different type of objects.
- Most widely used OA - BOA (Basic OA)
- Recently standardized - POA (Portable OA)

Object Adapter

- Services provided by ORB via OA -
  - Registering implementations.
  - Generation and interpretation of object references.
  - Mapping *object references* to their corresponding implementation.
  - Activating and deactivating object implementation.
  - Invocation of methods via a skeleton.

GIOP & IIOP

- IT is a high level standard protocol for communication between ORB implementations. It is designed to directly work over any connection- oriented transport protocol

- IIOP - It is a specialized form of GIOP for TCP/IP networks.
- IIOP specifies how GIOP messages will be exchanged over TCP/IP network
- An ORB must support IIOP in order to be considered compliant with CORBA 2.0.
- It consists primarily of the specification for the IIOP IOR, which contains the host name and the port number.

IOR

- Interoperable Object Reference (IOR) - can be used to access the remote object from within any ORB framework, regardless of where the IOR was created.

- As mentioned above, the actual type of the object reference is defined in the client's programming language within the IDL client stub.

- The object reference will expose an interface that maps directly to the interface defined in the IDL using the IDL to language mapping.

A simple corba application

Step 1: Create a simple Hello interface in hello.idl

```
module HelloApp
    {
    interface Hello
    {
    string sayHello();
    };
    };
```

A simple corba application

Step 2**:** Use the idltojava compiler as follows to compile the hello.idl file into the required Java mapping (idltojava Hello.idl)

The following files are created

Hello.java
HelloHelper.java
HelloHelper.java
_HelloImplBase.java
_HelloStub.java

A simple corba application

Step 3**:** Use the JDK 1.2 Java compiler and compile the foregoing classes (javac *.java)

Step 4: Create the HelloServer.java

- HelloServant extends the _HelloImplBase server-side skeleton automatically created by the idltojava compiler.
- Get the naming context and bind the object
- Step 5: Create the HelloClient.java
- Resolve binding to get reference to object and invoke sayHello()

A simple corba application

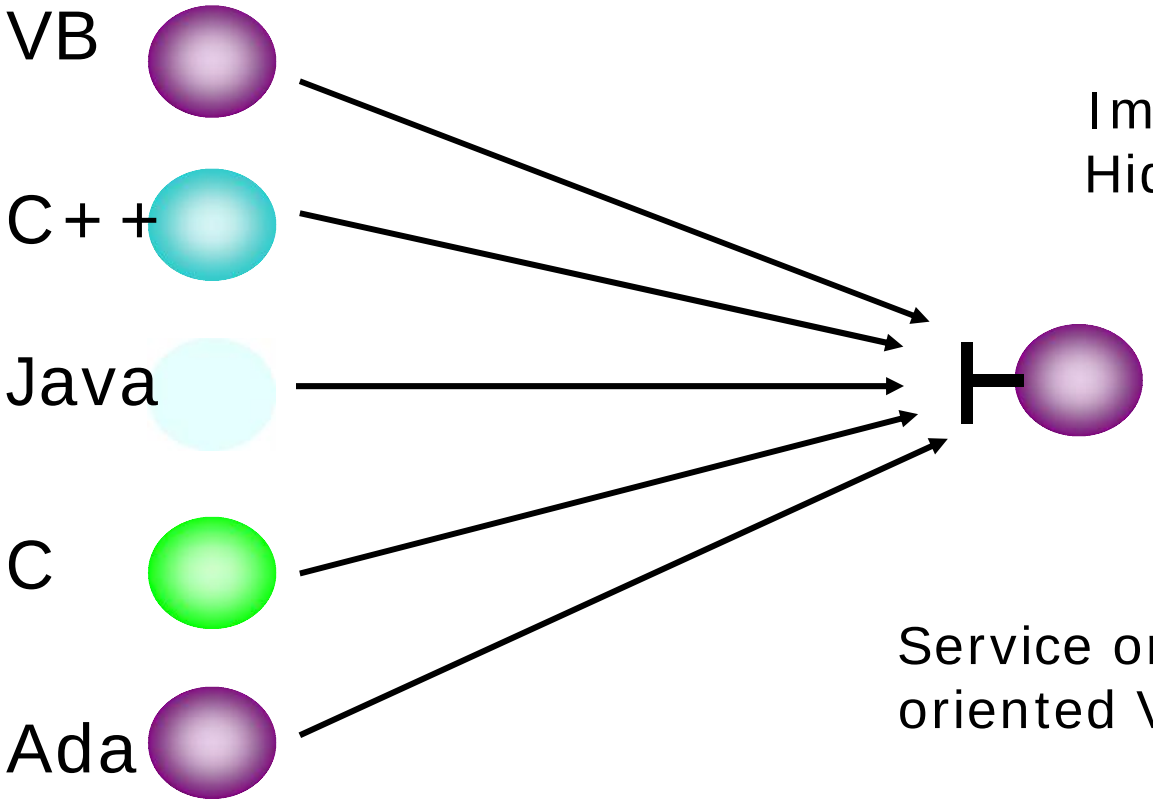
- Step 6: Compile HelloServer.java and HelloClient.java

- **Running the Application**
- Step 1: Start the transient naming service.
  tnameserv -ORBInitialPort 900

- Step 2: Start the HelloServer.
  java HelloServer -ORBInitialPort 900

- Step 3: Start the HelloClient.
  java HelloClient -ORBInitialPort 900

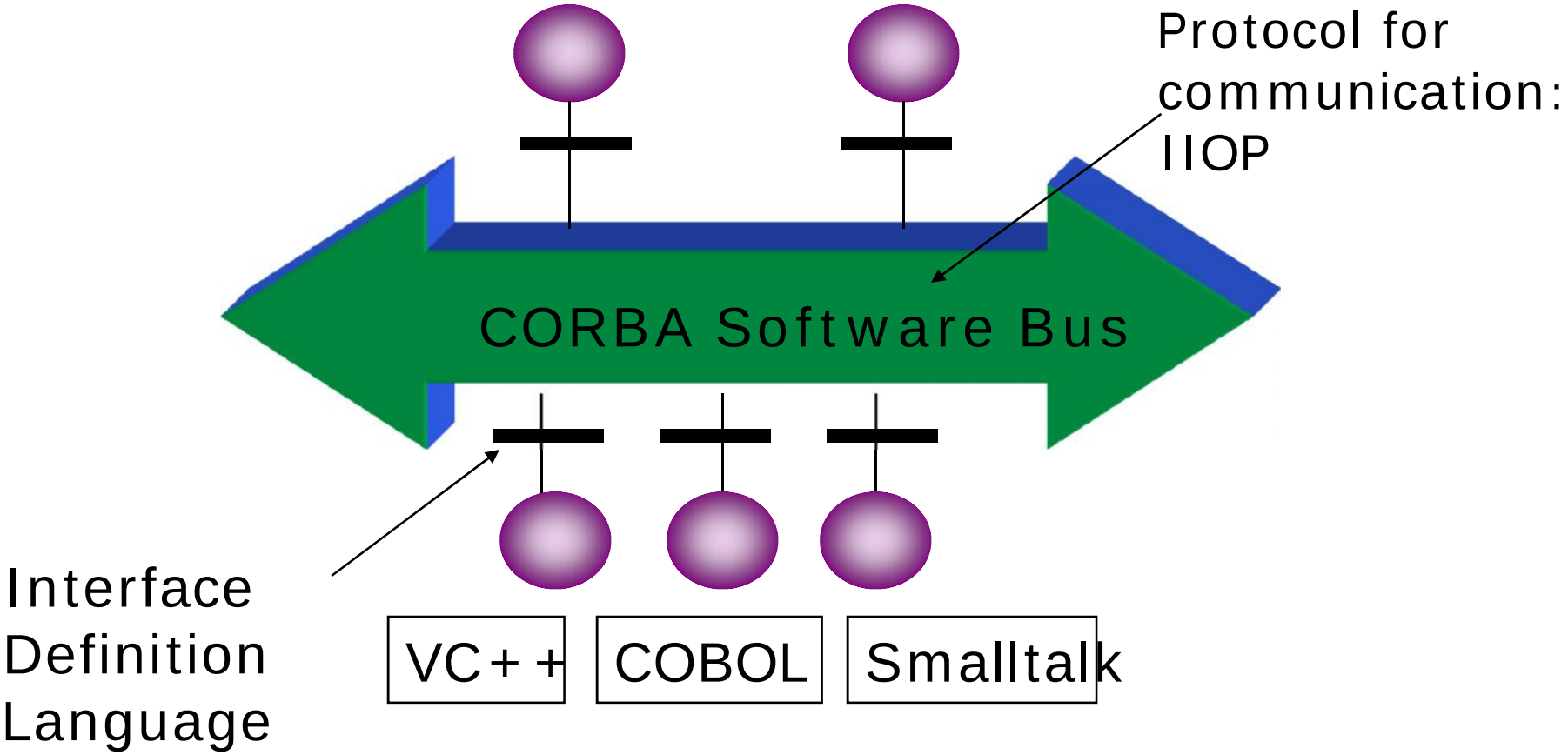
The First Key To CORBA – IDL & ORB



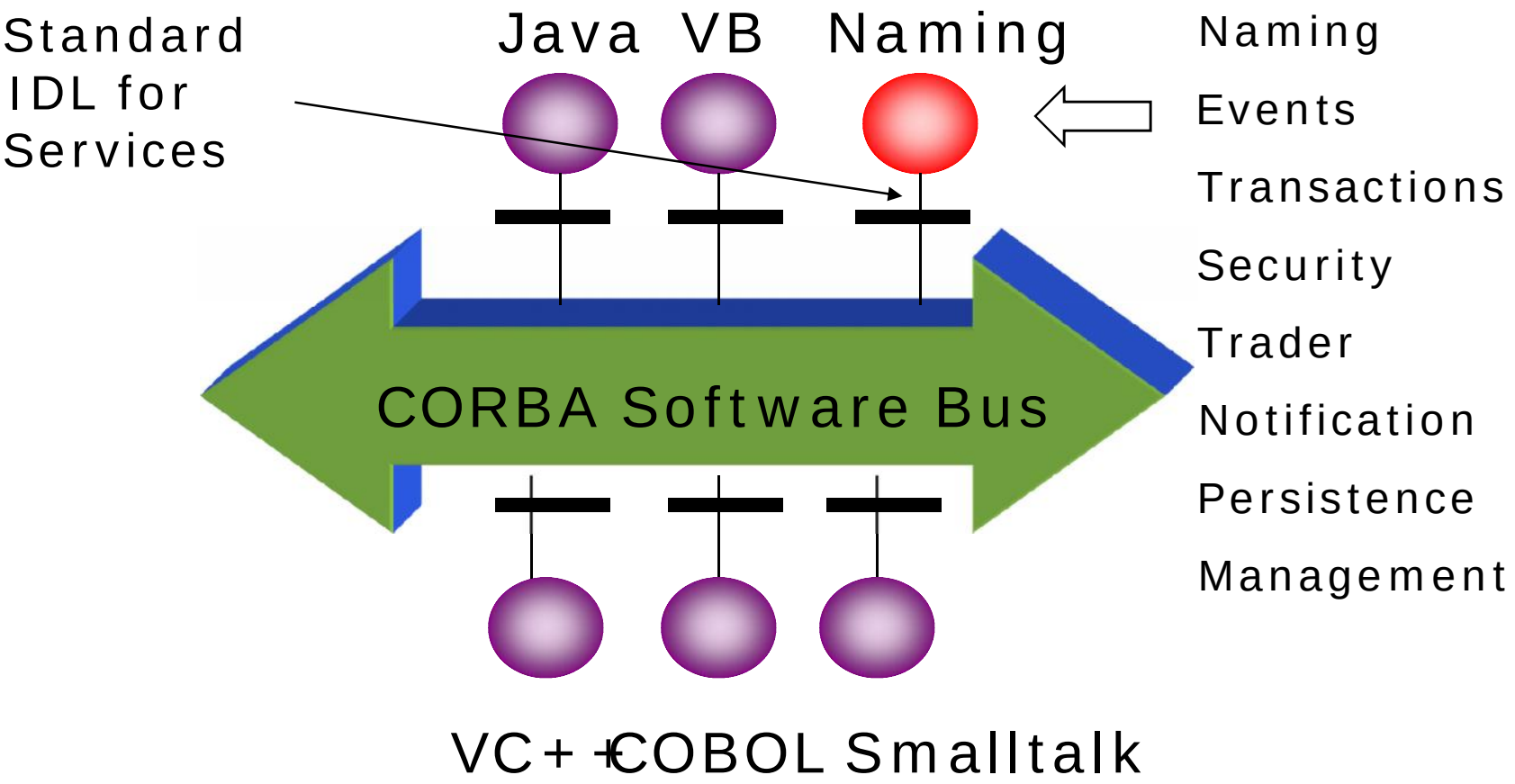
Implementation is
Hidden behind interface

Service or Contract-
oriented View

The Second Key to CORBA: IIOP



The Third Key to CORBA: *Services*



CORBA services

- Naming Service
- Trading Service
- Security Service
- Life Cycle Service
- Persistence Object Service
- Object Collection service
- Object transaction service
- Relationship service
- Event service
- Notification service
- Externalization service
- Time service

Naming service

Naming service provides a means for objects to be referenced by names within a given naming context

Naming service is implementation of OMG's Interoperable Name Service (INS) specification

It provides API to map object references to a hierarchical naming structure

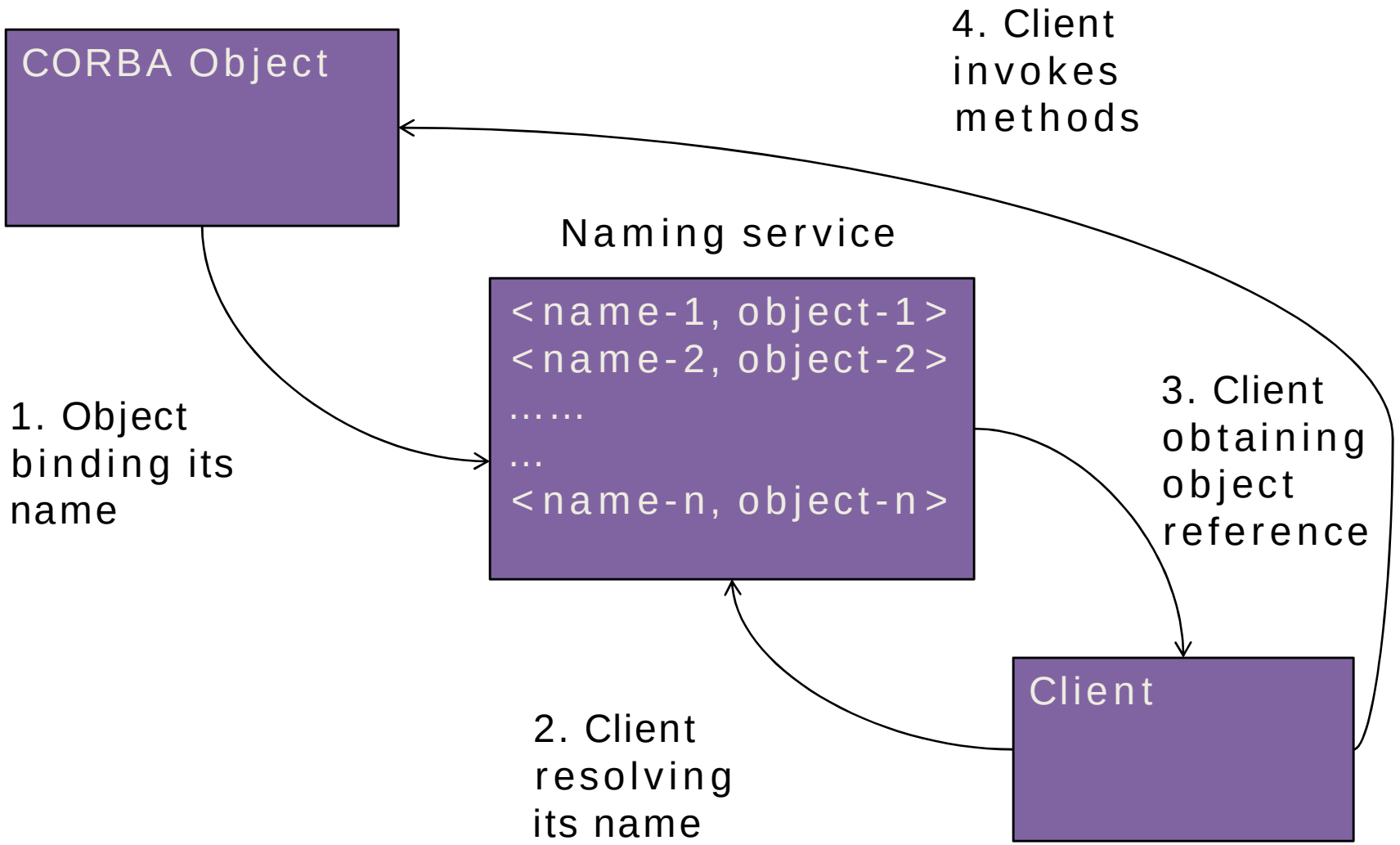
Naming context

A naming context is a scoping mechanism for names

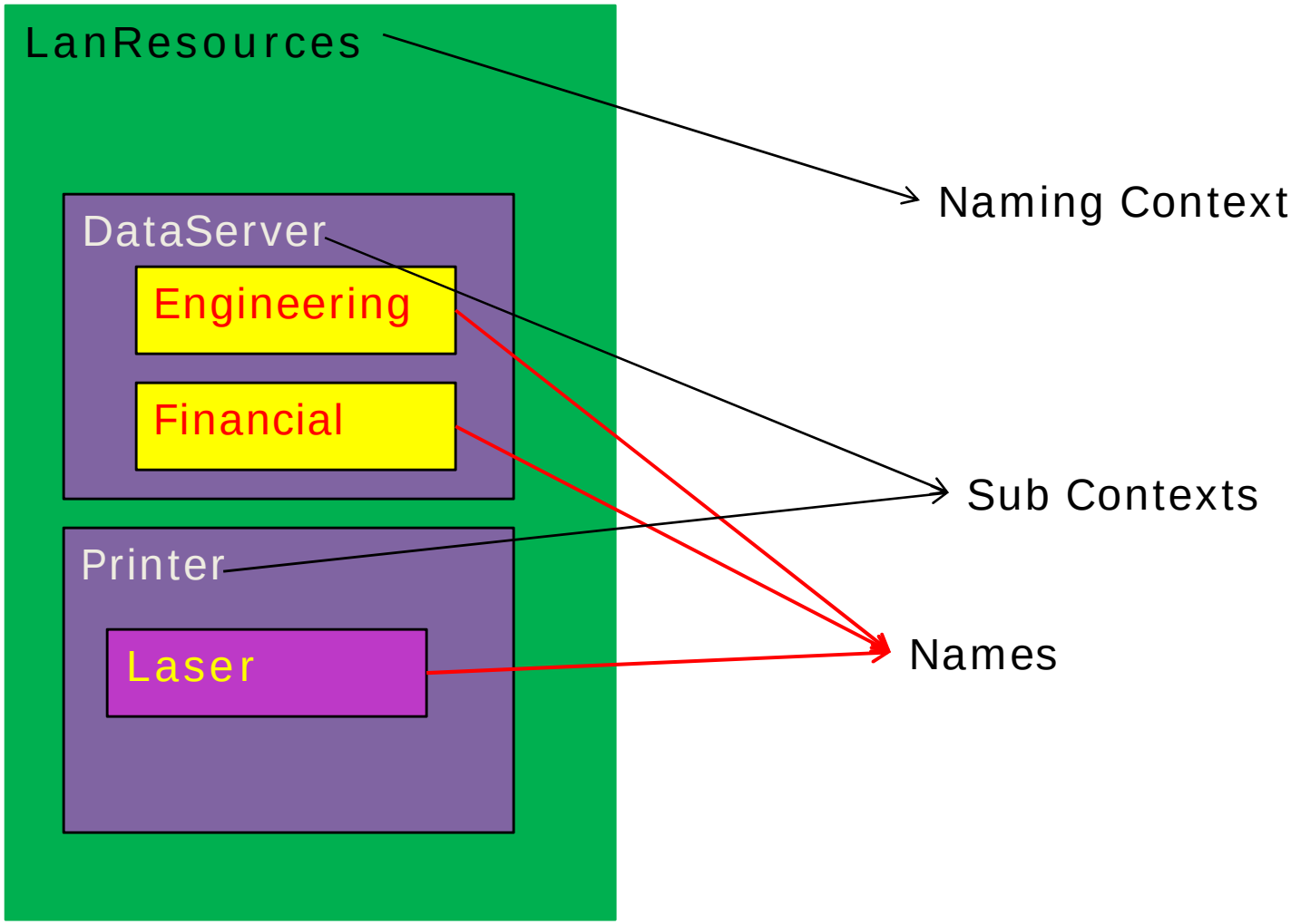
Within a particular context, names should be unique

Contexts can be nested to form compound context and names

Overview of naming service



Example


LanResources
DataServer
Engineering
Financial
Printer
Laser
Naming Context
Sub Contexts
Names

How to add named objects to naming service

Steps

1. Get the naming context from ORB
2. Create references to objects
3. Build required sub contexts
4. Create context
5. Create names with sub contexts
6. Bind in the context with objects mapped to their names

NamingContext

interface **NamingContext**
{
void **bind**(in Name n, in Object obj) raises(...);
void **rebind**(in Name n, in Object obj) raises(...);
void **bind_context**(in Name n, in NamingContext nc) raises(...);
void **rebind_context**(in Name n, in NamingContext nc) raises(...);
Object **resolve**(in Name n) raises(...);
void **unbind**(in Name n) raises(...);
NamingContext **new_context**();
NamingContext **bind_new_context**(in Name n) raises(...);
void **destroy**() raises(...);
void **list**(in unsigned long how_many, out BindingList bl, out BindingIterator bi);
};

Sample code

NamingContext base =//from ORB

//there are three objects having names *engineering, financial & laser*

org.corba.Object a = new ….eng object

org.corba.Object b = new ….fin object

org.corba.Object b = new ….laser object

Sample code contd.

```
//create required subcontexts

NamingComponent lan = new NameComponent("LanResources");

NamingComponent data= new NameComponent("DataServer");

NamingComponent prn= new NameComponent("Printer");
```

Sample code contd.

```
//create context

NamingComponent[] path = {lan};

NamingContext c = base.bind_new_context(path);

// redefine path
path={lan, data, new NamingComponent("Engineering");
//binding first object, eng with names
c.bind(path,eng);
```

Sample code contd.

```
// redefine path
path={lan, data, new NamingComponent(“Financial”);
//binding second object, fin with names
c.bind(path,fin);


// redefine path

path={lan, prn, new NamingComponent(“Laser”);
//binding third object, prn with names

c.bind(path,prn);
```

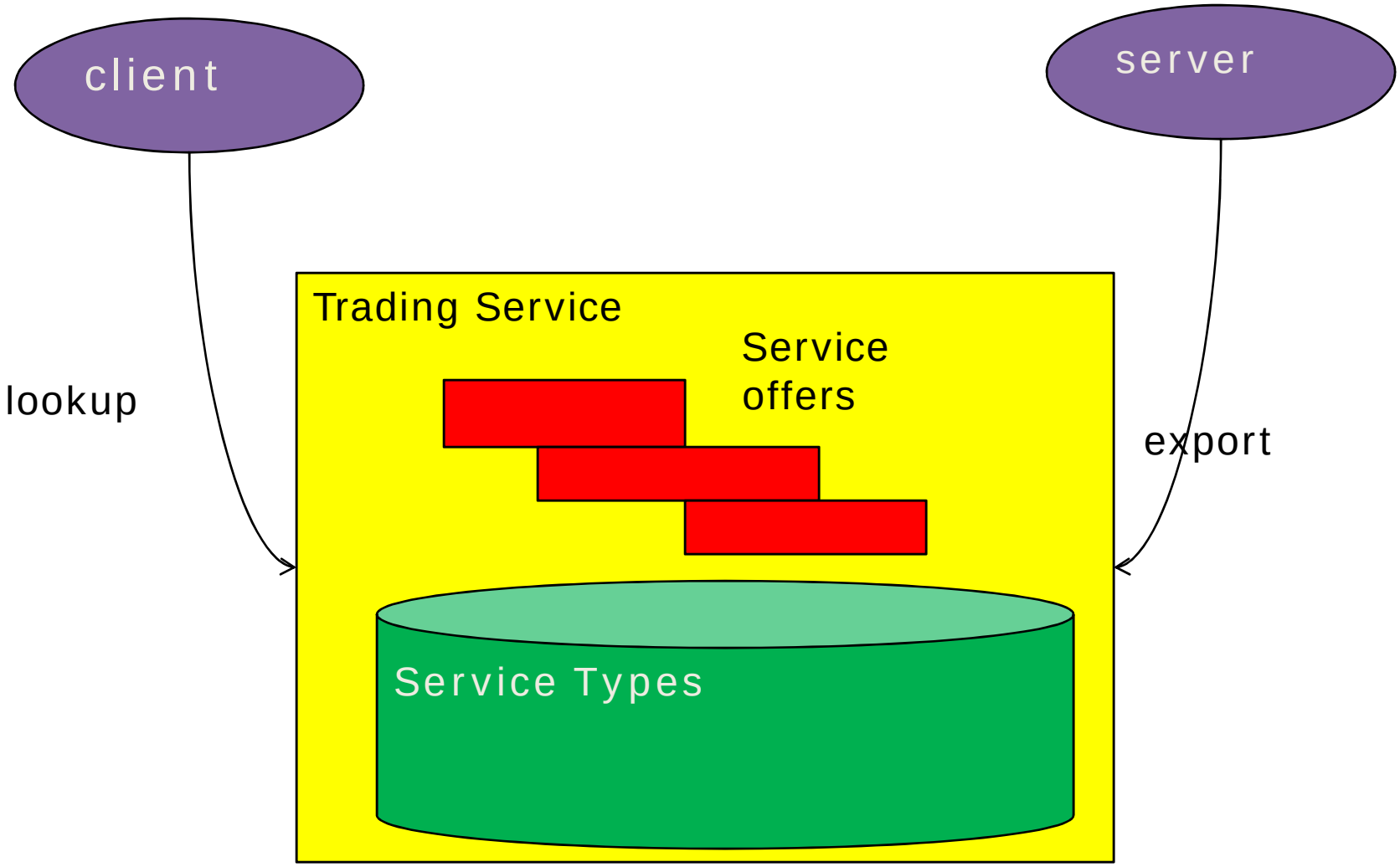
Client resolving name

Object resolve(name of the object);

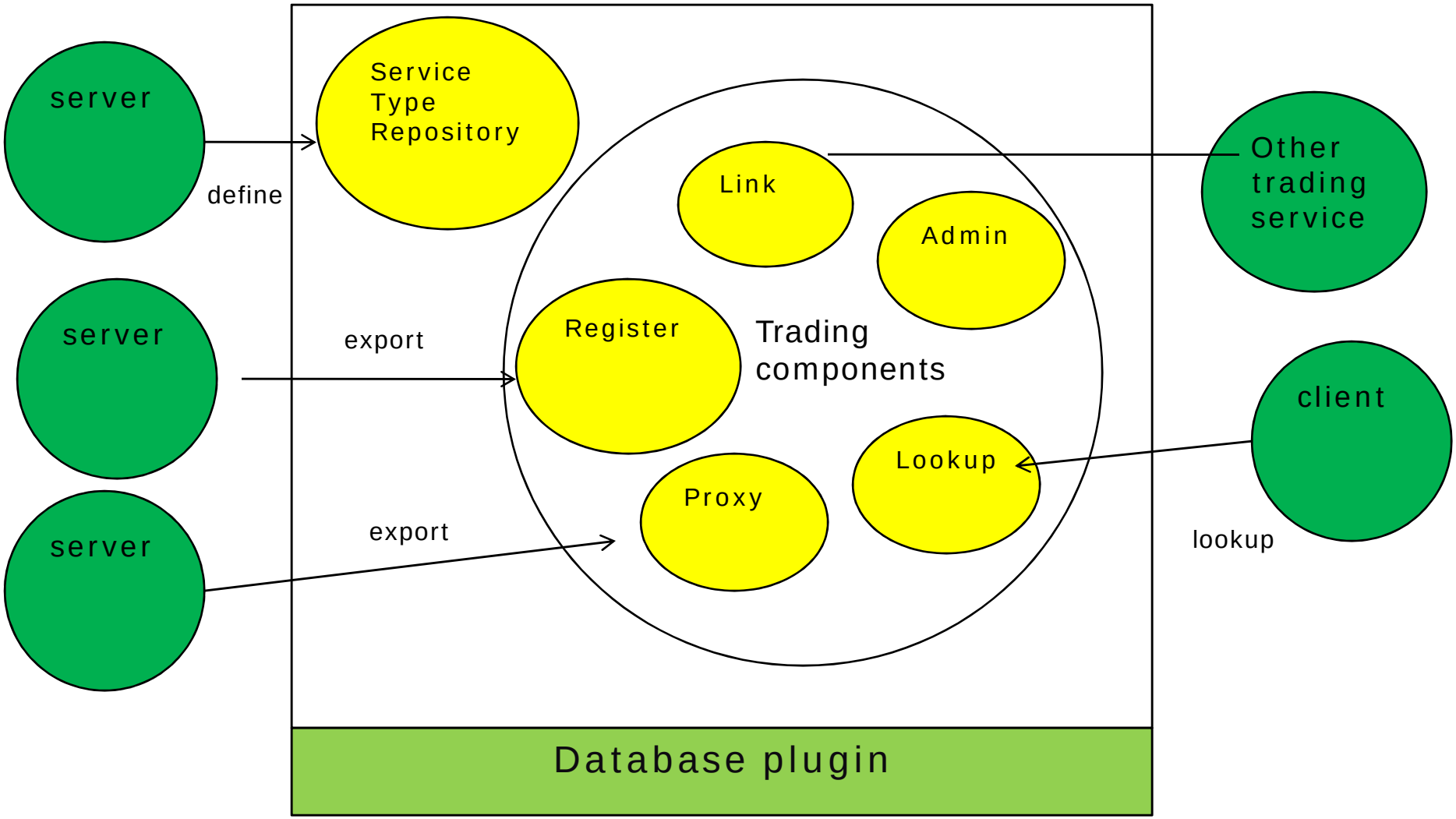
Trading service

- Trading service contains numerous service offers organized into different service offer types(categories)
- Trading service provides a way for a server to advertise its object reference

Overview of trading service



Architecture of trading service



Trading components

- **Service Type Repository**

  The Service Type Repository contains details of service types, as a kind of catalogue. Each service type definition has a unique name and zero or more properties. An appropriate service type must exist in the Service Type Repository before a server can export an offer

ServiceTypeRepository

- This interface is used to define service offer types
    - add_type() – used to define new service offer type
    - remove_type() – used to remove a service offer type

Register

When a service offer type is created, it possible to advertise corba object references using export method

export() operation is used to create a *service offer*, that is, an advertisement for an object. Parameters to this operation specify an object reference, the service offer type that it matches, and its properties

The return value from export() is a unique string that denotes an offer id. This is used to *withdraw* or *modify* the offer

withdraw() operation is used to withdraw (that is, delete) a service offer.

string export(in Object reference, in string type, in PropertySeq properties) raises(...);

void withdraw(in string offer_id)…..

Lookup

Lookup interface has only one operation, called query().

This operation is used to retrieve service offers (advertisements) from the Trading Service that match a specified constraint.

void query(service_type_name, constraint, preference, ... desired_props,...) raises(...); };

Other interfaces and details

The Proxy supports the delayed evaluation of offers and can be used to

encapsulate legacy systems. A proxy offer is like a normal service offer in that it has a service type and named properties. However, it does not contain an object reference leading directly to the interface providing the service; it contains a reference to an object supporting the Lookup interface (the Trader performs a secondary lookup on this interface, transparently to the client).

- **Link**
- The Link interface is used to federate traders.

- **Admin**
- The Admin interface is used to query and change the administrative properties of the trader

- **Database Plug-in**

- Persistence in the Trading Service is normally implemented with a database. The OpenFusion Trading Service supports many different databases through the use of JDBC plug-ins.

Other details

- **XML Import and Export**
- Trading Service enables offers to be both exported and imported as XML documents.
- Service type definitions can also be imported into the service type repository as XML documents.
- **Service Types**
- A *service type represents the information needed to describe a service. The* service type has a name This is usually meaningful within the the context of the business where the service will be used; in a video-on-demand system, for example, service types would probably have names such as programme, movie and sports_event. A service type definition contains:
- • an *interface name*
- • zero or more *named property types*
- • zero or more *super-types*

Programming trading service

```
org.omg.CORBA.Object obj =
orb.resolve_initial_references("TradingService");
org.omg.CosTrading.Lookup trader =
org.omg.CosTrading.LookupHelper.narrow(obj);
```

Step 1 - Add a service type in service type repository

Create a service offer type if a corresponding one doesn't already exist within the Trader Service. This example creates an Printer service offer type.

```
org.omg.CORBA.Object obj = trader.type_repos();
```

Trading service

```
//create properties
org.omg.CosTradingRepos.ServiceTypeRepository.PropStruct[] props = new
      org.omg.CosTradingRepos.ServiceTypeRepository.PropStruct[2];

props[0] = new  org.omg.CosTradingRepos.ServiceTypeRepository.PropStruct();
props[0].name = “resolution";
Props[0].value _type =…..primitive long
….
…..
type_repos_obj.add_type(
 "Printer" // Service Type
 "IDL:TraderDemo/PrintServer:1.0", // IDL type name
 props, // offer properties
 superTypes // no supertypes
);
```

Exporting service offer

```
PrintServer_Impl print_server_impl = new PrintServer_Impl();
PrintServer print_server = print_server_impl._this(orb);;
 org.omg.CORBA.Object trader = orb.resolve_initial_references("TradingService");
org.omg.CosTrading.Lookup lookup = org.omg.CosTrading.LookupHelper.narrow(trader);
org.omg.CosTrading.Register register = lookup.register_if();
3 org.omg.CosTrading.Property[] props = new org.omg.CosTrading.Property[3];
props[0] = new org.omg.CosTrading.Property();
Props[0].name = “resolution";
props[2].value = orb.create_any();
props[2].value.insert_long(100);
String id = reg.export(
);
```

Lookup into trader

```
// Trader Service reference, trader, acquired earlier
org.omg.CosTrading.Policy[] policies = new org.omg.CosTrading.Policy[0];
org.omg.CosTrading.LookupPackage.SpecifiedProps desiredProps =
new org.omg.CosTrading.LookupPackage.SpecifiedProps();
desiredProps.__default(org.omg.CosTrading.LookupPackage.HowManyP
rops.all);
org.omg.CosTrading.OfferSeqHolder offers = new org.omg.CosTrading.OfferSeqHolder();
org.omg.CosTrading.OfferIteratorHolder iter = new
org.omg.CosTrading.OfferIteratorHolder();
org.omg.CosTrading.PolicyNameSeqHolder limits = new
org.omg.CosTrading.PolicyNameSeqHolder();
 trader.query(
 "Printer", // the service type
 “resolution = 100", // the constraint to match
 "random", // the order to sort the results
 policies, // no special policies
 desiredProps, // set to return all properties
 50, // max offers to return
 offers, // offers returned
 iter, // remaining offers
 limits // polices applied by the trader
);
```

Look into trader

```
 org.omg.CosTrading.Offer[] offer = offers.value;
if (offer.length() != 0)
 {
 //match offer and find best match
 }
```

Security service

Element	meaning
Subject	A human user or system entity which, may attempt an action within a secure system.
authentication	The act of establishing the identity of a subject. Once authenticated, the subject becomes a *Principal*.
Principal	An *authenticated subject*. Basically, this is any entity that directly or indirectly causes an invocation to be made against an object.
Credential	A container within a secure CORBA system for the security attributes associated with a principal.
Security association	The result of establishment of trust between a specific client and server, possibly enduring several invocations.

Security features authentication

- *An entity refers to human user or a program. An entity should prove its identify*
- an entity with the ability to use the resources of a system is called a *principal*
- *Authentication* is the process of verifying an entity's claimed identity.
- Note that clients can authenticate servers, and vice versa.
- Authentication can be either mandatory or optional depending on the security requirements of a given system.
- Successful authentication results in the principal being granted a set of *privilege attributes* (such as roles, groups, security clearance levels and so on); these are stored in a *credentials* object
- The credentials are considered during *authorization*.

Authorization

- This is the process of verifying whether or not a principal is allowed to perform a requested action in a system.
- An example of a commonly used authorization paradigm is Access Control Lists (ACLs).
- Although the flexibility of ACLs differ among CORBA Security products, ACLs typically allow access to be constrained at varying levels of granularity, such as per-process, per-object, per-interface or per-operation.
- Some products may also provide ACL functionality that allows access decisions to be made based on the values of parameters passed to IDL operations.
- Note that during authorization the set of *privilege attributes* that was determined for the principal during the authentication process is used to control access to system resources.

Data integrity

- **Data integrity.**
- This uses techniques such as message digests (a form of cryptographic checksum) to provide protection against malicious modification of messages.

Confidentiality

- This ensures the privacy of message exchanges so that only the intended recipients can read them.

Detection of misordering

- This prevents an attacker from rearranging messages in a different order to that in which they were sent.

Auditing/logging

- This involves keeping secure records of “who did what” so that access to system resources can be examined at a later time. Some security systems allow registration with a real-time management service that can perform appropriate system-defined alerts.

Delegation

- This is when one user or principal authorizes another to use their identity or privileges, potentially with usage restrictions. authorization process and be based on the current effective principal. Alternatively, some security products additionally allow authorization decisions to be based on delegation constraints associated with a request.

Non-repudiation

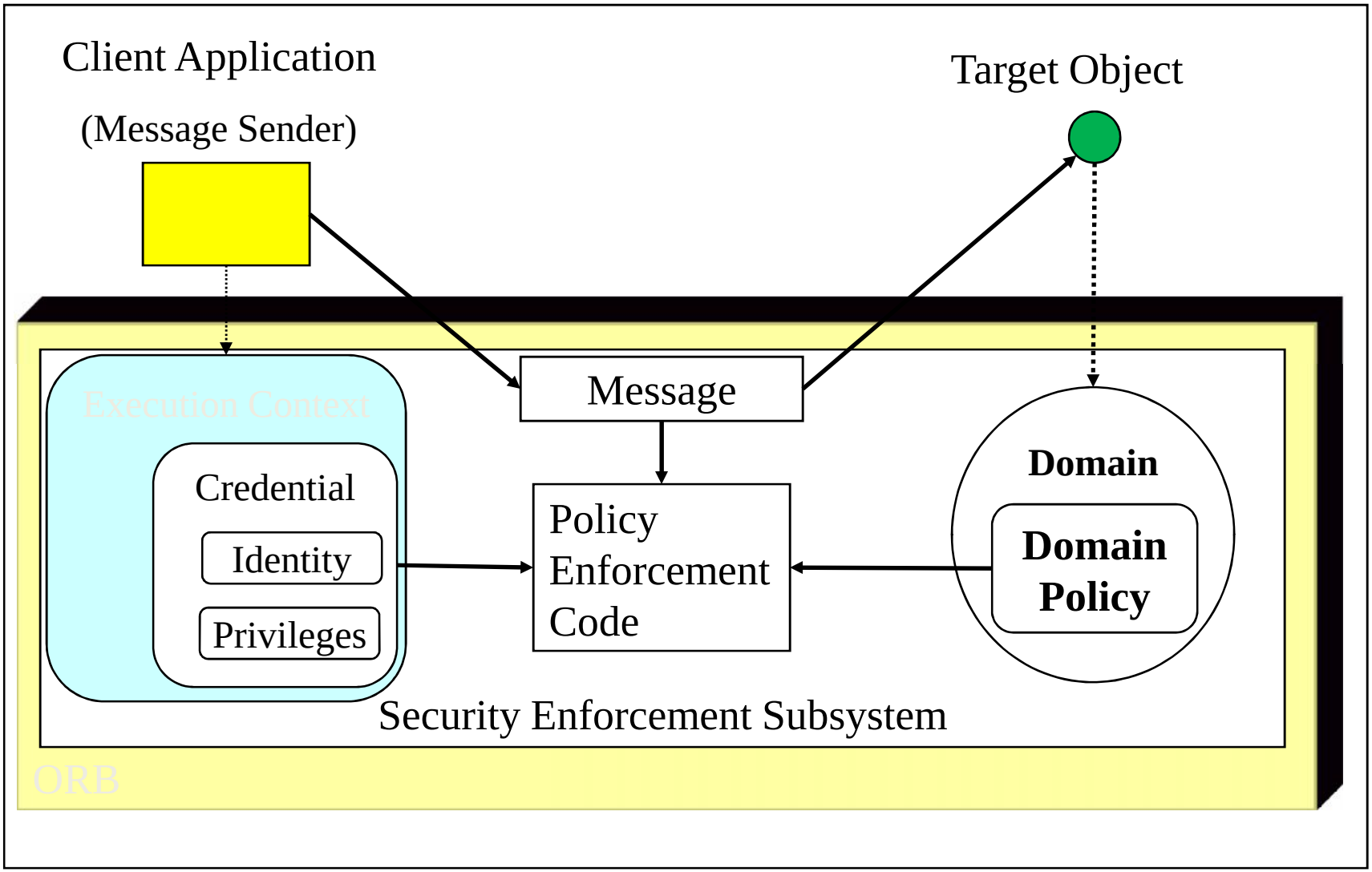
- **Non-repudiation.**
- *Non-repudiation* means the ability to prove whether or not a principal invoked a particular operation, so that the principal cannot later deny invoking an operation that he or she did, in fact, invoke..

Different levels of security

- Security: basic interfaces and data types which span all levels
- SecurityLevel1: the most basic, provides security features to all applications regardless of their level of participation
- SecurityLevel2: permits access to credentials and additional policy controls
- SecurityAdmin: permits manipulation of administrative features not necessarily related to invocation or other processing

User Authentication

CORBA Security Model



Subjects

- Security attributes
  - Identities : username, certificates
  - Privilege attributes : groups, roles
- Credentials are containers for security attributes
- Active entities in the system identifiable using “credentials”
- The PrincipalAuthenticator object authenticates subjects and assigns non-public security attributes

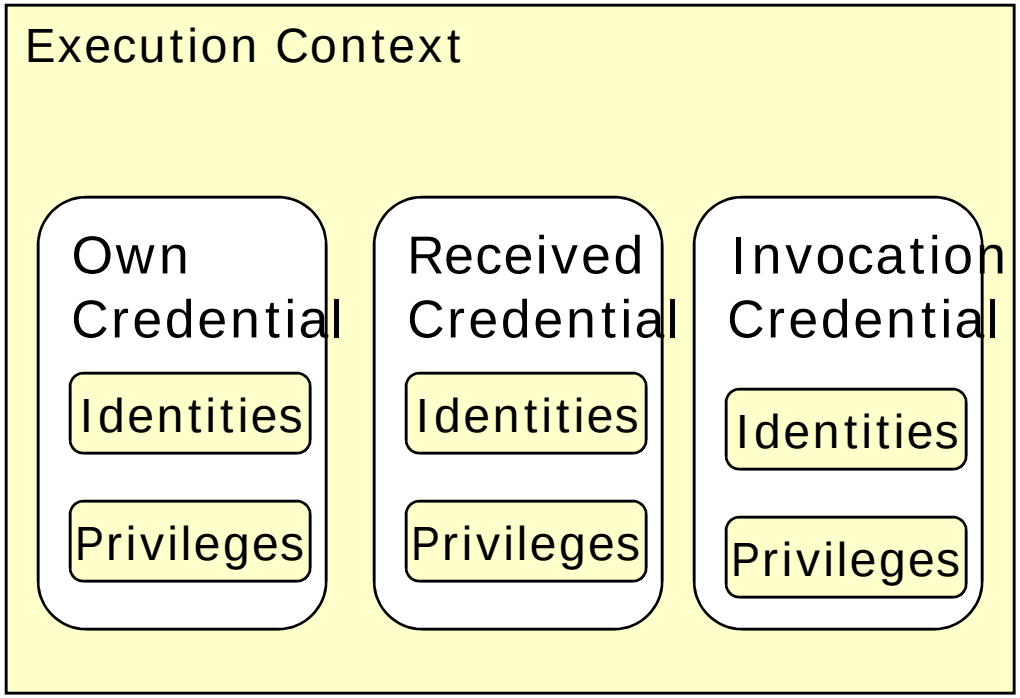
Actions

- Methods invocations
- ORB can look at each request or response and see whether it’s legal according to the security policy rules
- ORB provides security transparent

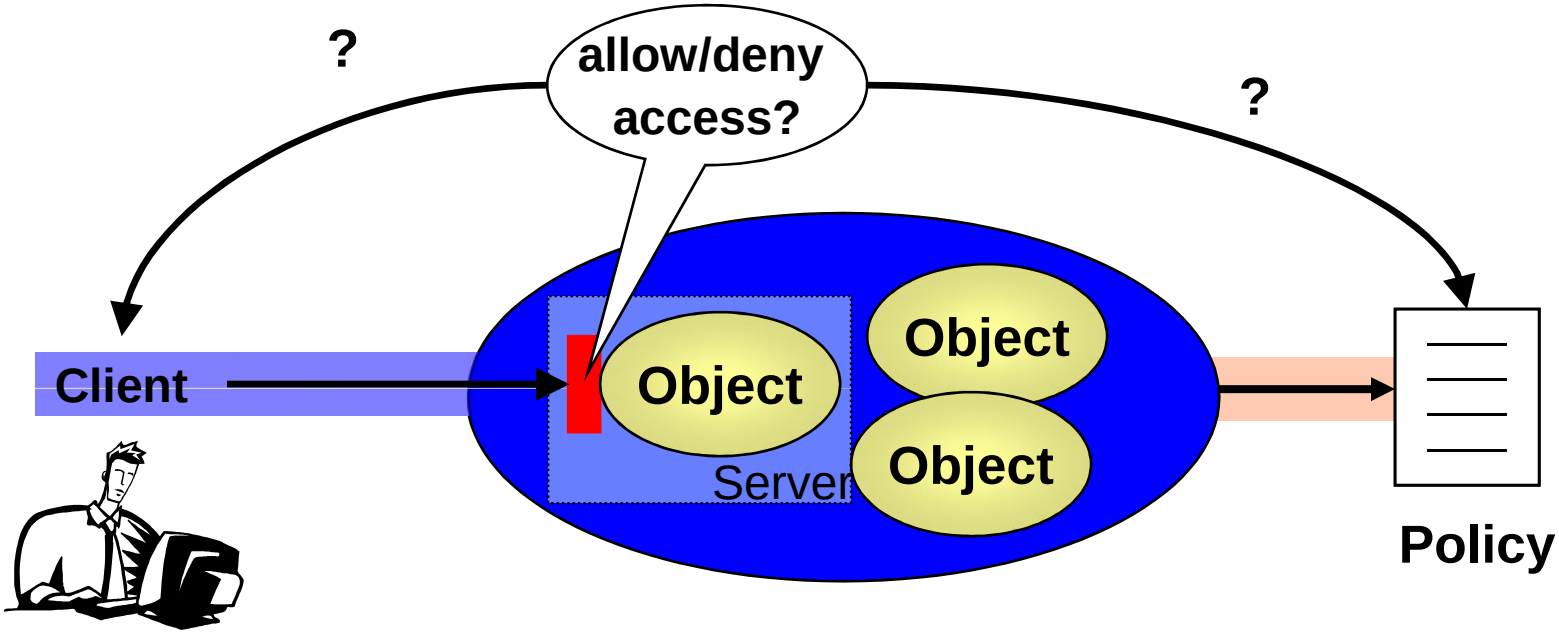
Execution Contexts

- Credentials are stored in execution contexts
- Own credentials
  - belongs to the current subject
- Received credentials
  - Belongs to the subject that most recently sent a message
- Invocation credentials
  - The subject identity that will be used when sending the next message

Execution Contexts



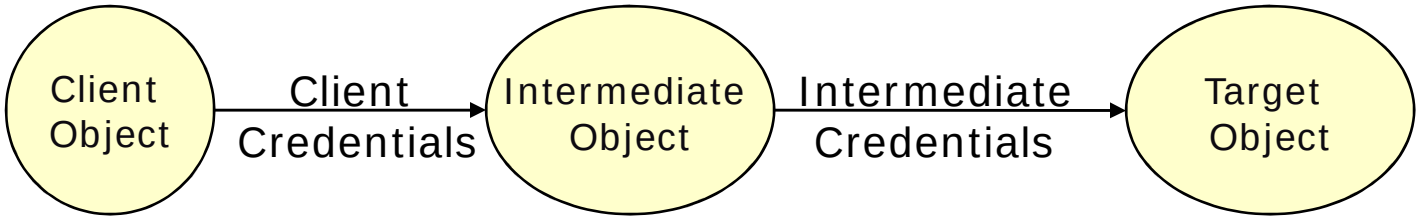
Access Control information



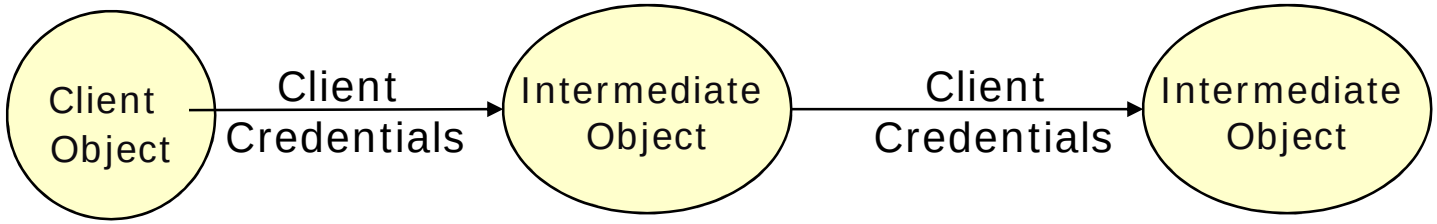
- Grouping of Objects with the same Policy in Policy-Domains

Delegation

No delegation

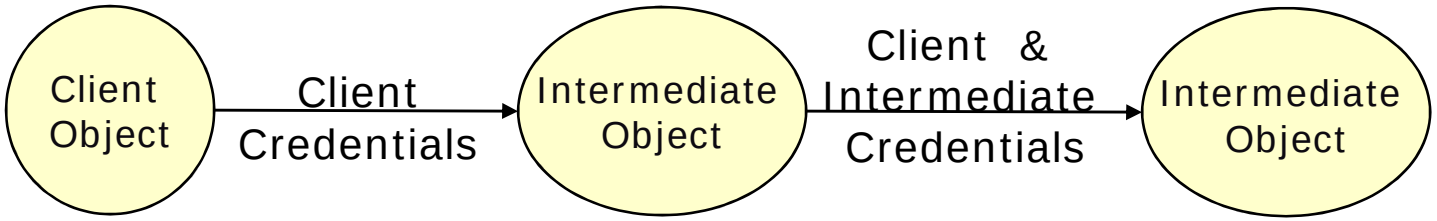


Simple delegation

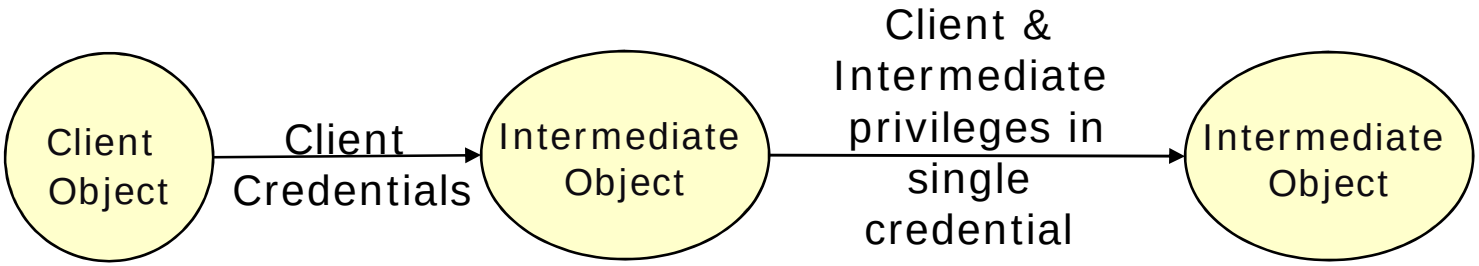


Delegation

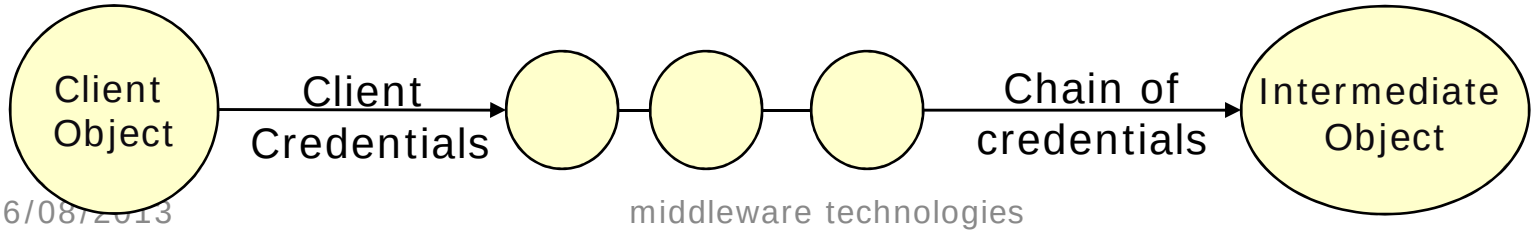
Composite delegation



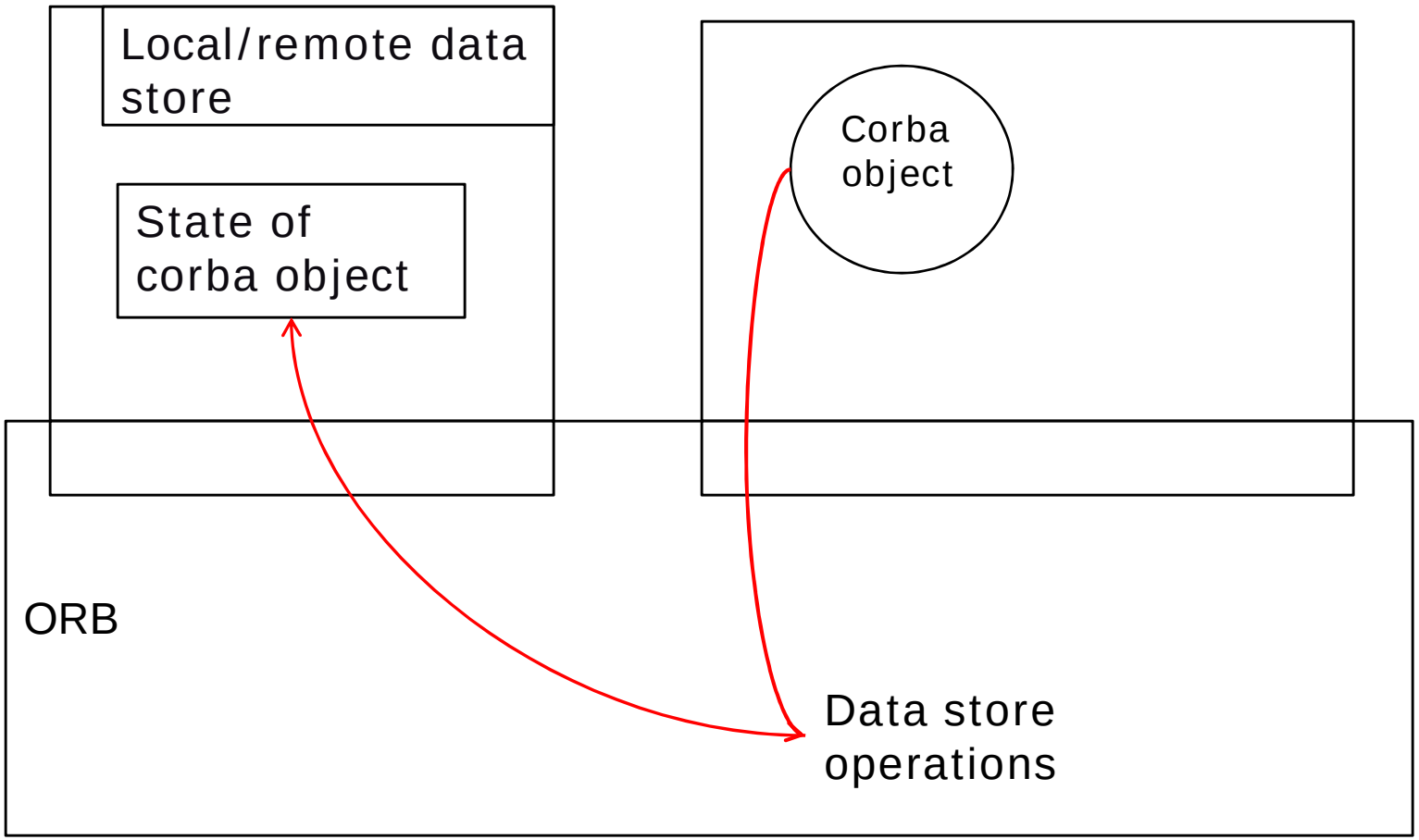
Combined privileges delegation



Traced delegation



Persistence – overview



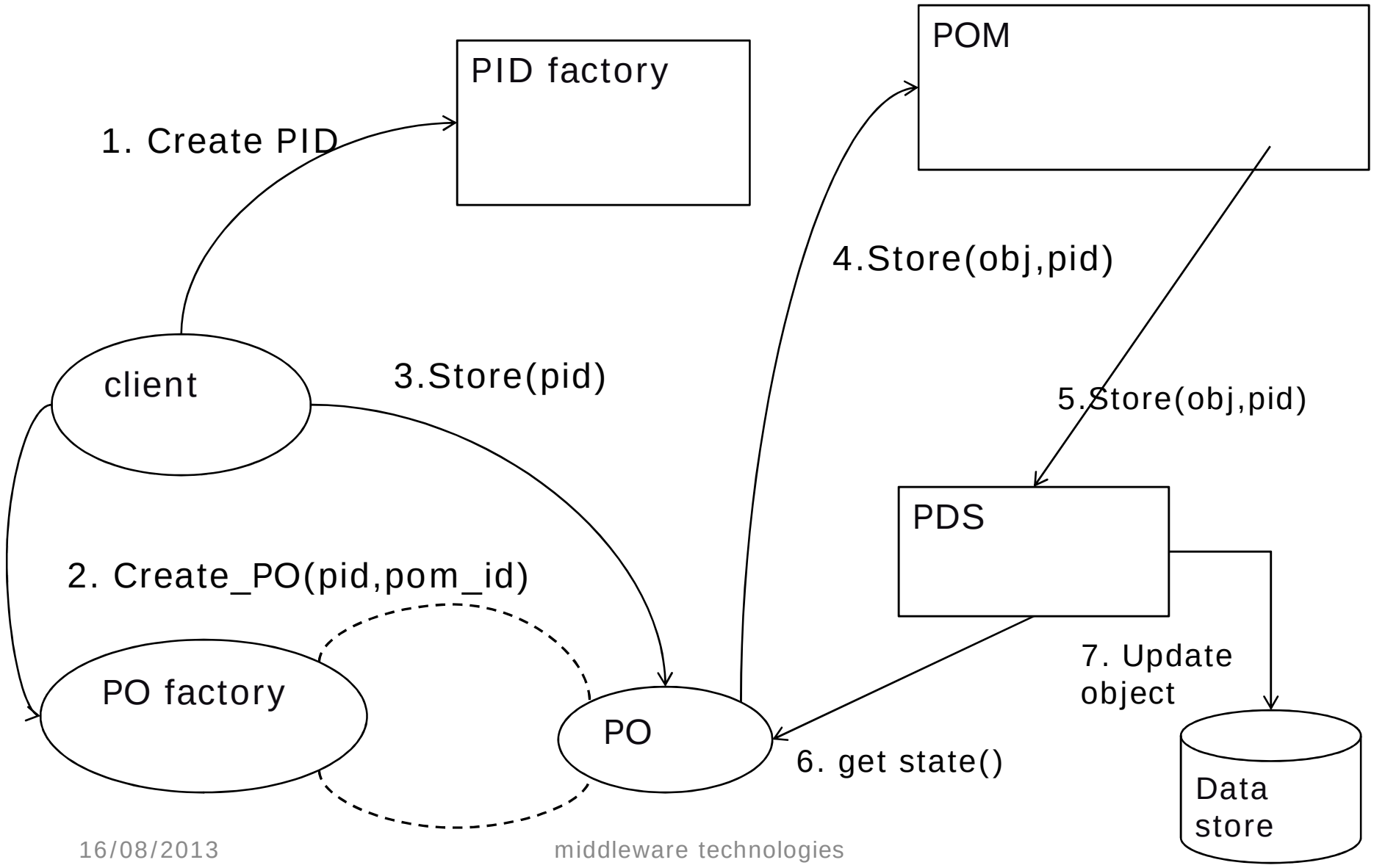
Persistence Object Service

POS - OMG's specification to provide common interfaces to the mechanisms used for retrieving and managing persistent state of corba objects.

POS abstractions/components

- Persistent ID – Object ID
- Persistent Object – Corba object
- Persistent Object Manager – redirects request from client to a particular mechanism used to control an object's persistence
- Protocol – provides interface between object and PDS
- Data store – maintains object's persistent state
- Persistent Data Service (PDS) – provides mechanism for persistence

Interactions



Interactions

Steps

1. Client performs initialization. Creation of PID by using factory object. create_pid()
2. Client creates Persistent Object (PO). The PID must be associated with PO. Here, PO factory creates PO create_PO(pid,pom_id)
3. Client seeks store(pid) on PO
4. In turn, the PO seeks store(obj, pid) to POM
5. In turn, the POM submits the request to PDS
6. PDS gets the current state of the object and
7. PDS updates the state in data store

Life cycle service

- Creation of objects
- Removal of objects
- Copying of objects
- Moving of objects

Creation of corba objects

Object creation is performed using factory objects

A factory is a corba object offering a method for creating new instances of a particular object type at a particular location

Life cycle service defines an IDL interface called ‘GenericFactory’. This interface contains a generic ‘create_object()’ method. By specifying a set of IDL ‘Criteria’ corba objects are created

Other life cycle operations

Object creating is handled by factory object. Other life cycle operations are executed on the objects itself.

An object support life cycle services has to implement LifeCycleObject interface

LifeCycleObject

```
Interface LifeCycleObject
{
    LifeCycleObject copy(in FactoryFinder there, in Criteria c)
                        raises (….)

    void move(in FactoryFinder there, in Criteria c))



};
```

Other life cycle operations

copy() – creates a copy of the object at some location. A reference to the newly created object is returned

move() – this operation moves the object to another location

remove() – deletes the object

move() method

Life cycle service specifies a ‘FactoryFinder’ interface

move() method is called on the object

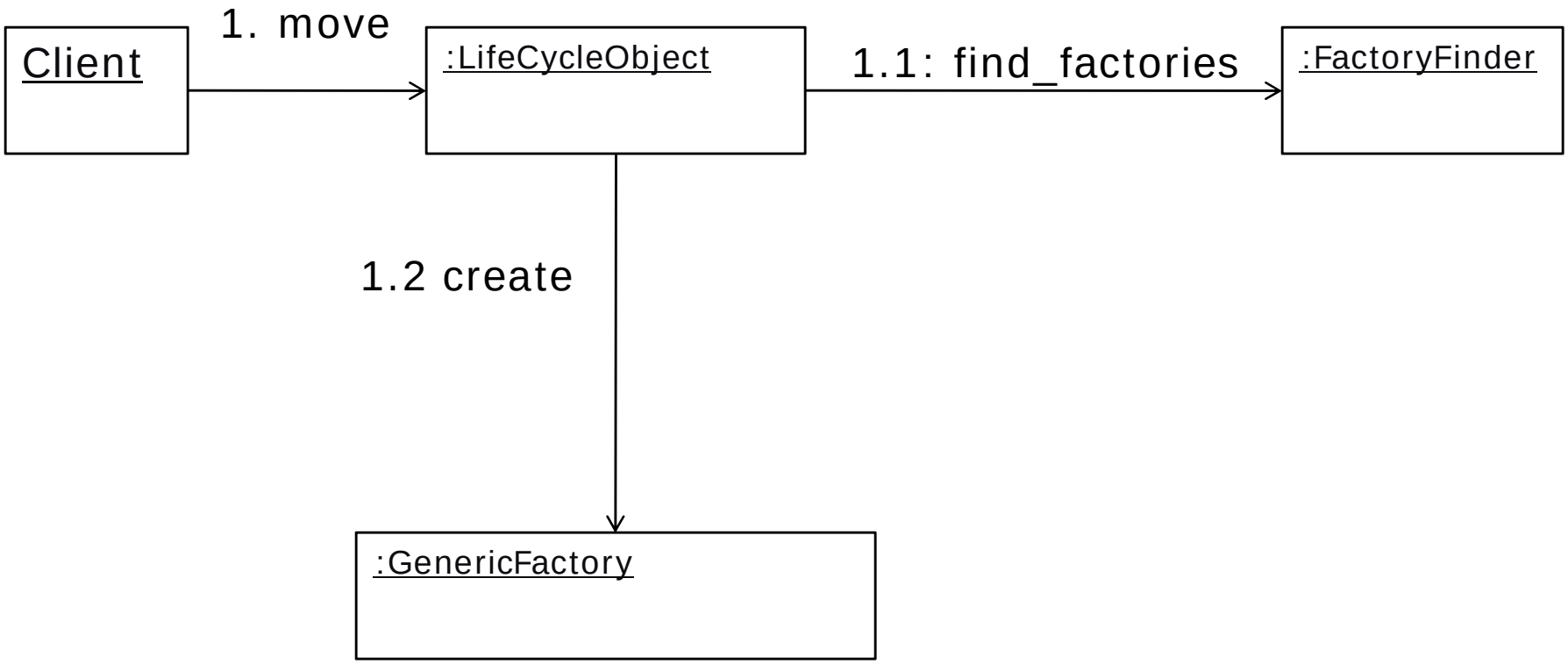
An instance of FactoryFinder is passed as parameter

The move() operation is supposed to as the factory finder for a factory.

The factory creates another instance of the original object at a location.

The method transfers the state of the original object to new one and makes the original reference referring to the newly created object.

Diagram


Client
1. move
:LifeCycleObject
1.1: find_factories
:FactoryFinder
1.2 create
:GenericFactory