CORBA

(*Common Object Request Broker Architecture*)

René de Vries

(rgv@cs.ru.nl)

Based on slides by M.L. Liu

Overview

- Introduction / context
- Genealogical of CORBA
- CORBA architecture
- Implementations
- Corba related specifications
- Conclusions
- References

Means to handle distributed application development

- Abstraction (Object Orientation)
- Infrastructure (services and object request broker)
- Divide and Conquer (Corba Component model (CCM))

Solution:
CORBA

What is CORBA?

- Acronym: Common Object Request Broker;
- A middleware
- Vendor-independent architecture;
- Set of protocol definitions;
- Standard architecture;
- CORBA enables:
  - distributed objects to interoperate in a heterogenous environment (transparancy)
  - programming language independence
  - deployment on different platforms

Who standardized CORBA?

- Developed by *Object Management Group* (OMG):
  - 800 members (Philips, HP, Sun) (no M$!)
  - Industrial consortium since 1989
- Goal:
  - Promotion of OO SE (UML), common architectural framework, development of distributed OO SW applications.

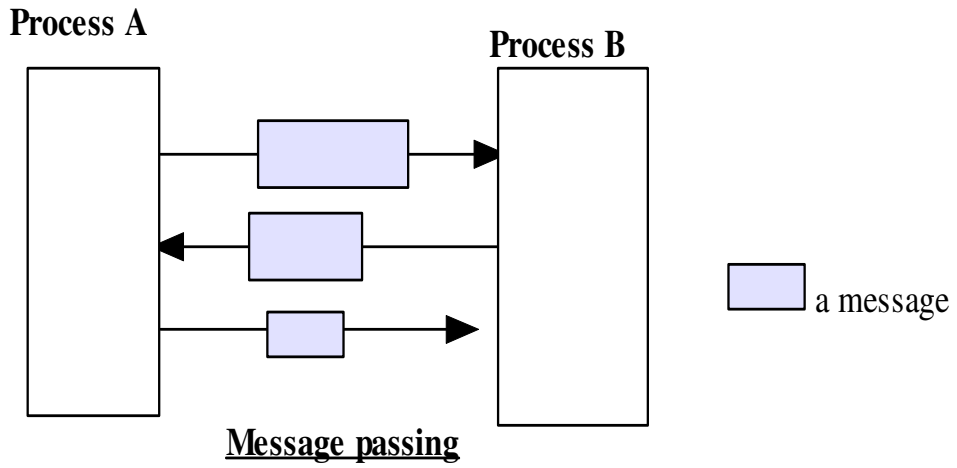
Genealogical

- Message passing
- Client-Server model
- Remote Procedure Calls (RPC) 1980
- Remote Method invocation (RMI) (Java/Sun)
- Object Orientation
  - Polymorphism (binding?)
  - Inheritance
- ORB 1990
- CORBA (release 1-3)

Message Passing mechanism

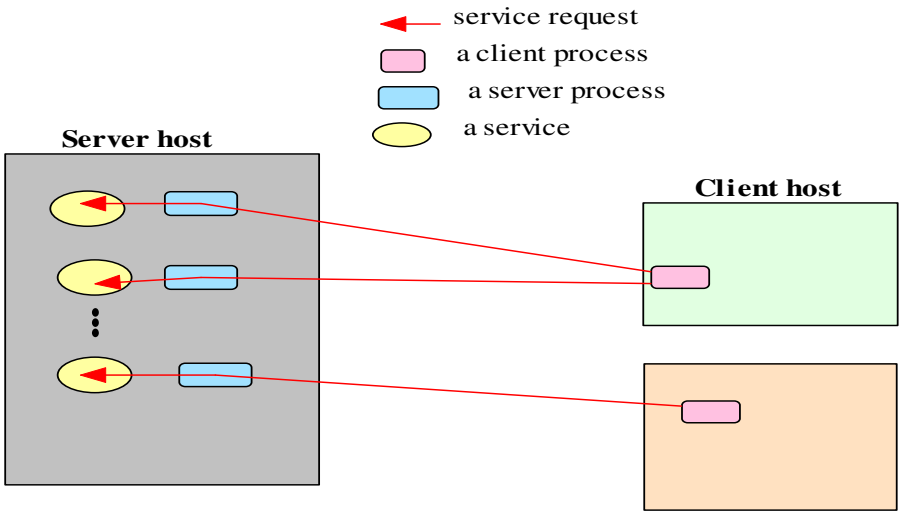
Message passing is the most fundamental paradigm for distributed applications.

- **A process sends a message representing a request.**
- **The message is delivered to a receiver, which processes the request, and sends a message in response.**
- **In turn, the reply may trigger a further request, which leads to a subsequent reply, and so forth. -**



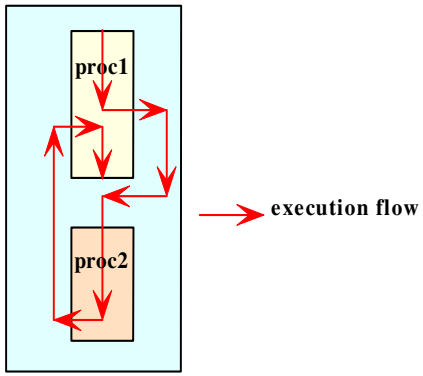
Client-Server Model

- The client-server model assigns asymmetric roles to two collaborating processes.
- The server waits passively for the arrival of requests. The other, the client, issues specific requests to the server and awaits its response.
- Examples (including message passing): FTP, HTTP etc.



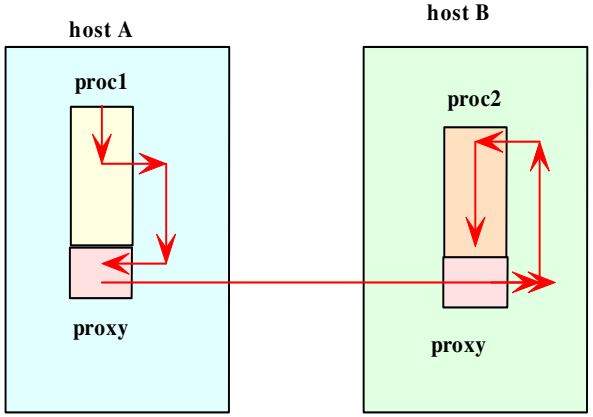
The Client-Server Paradigm, conceptual

Local Procedure Call and Remote Procedure Call



A local procedure call

1. proc1 on host A makes a call to proc 2 on host B.
2. The runtime support maps the call to a call to the proxy on host A.
3. The proxy marshalls the data and makes an IPC call to a proxy on host B.



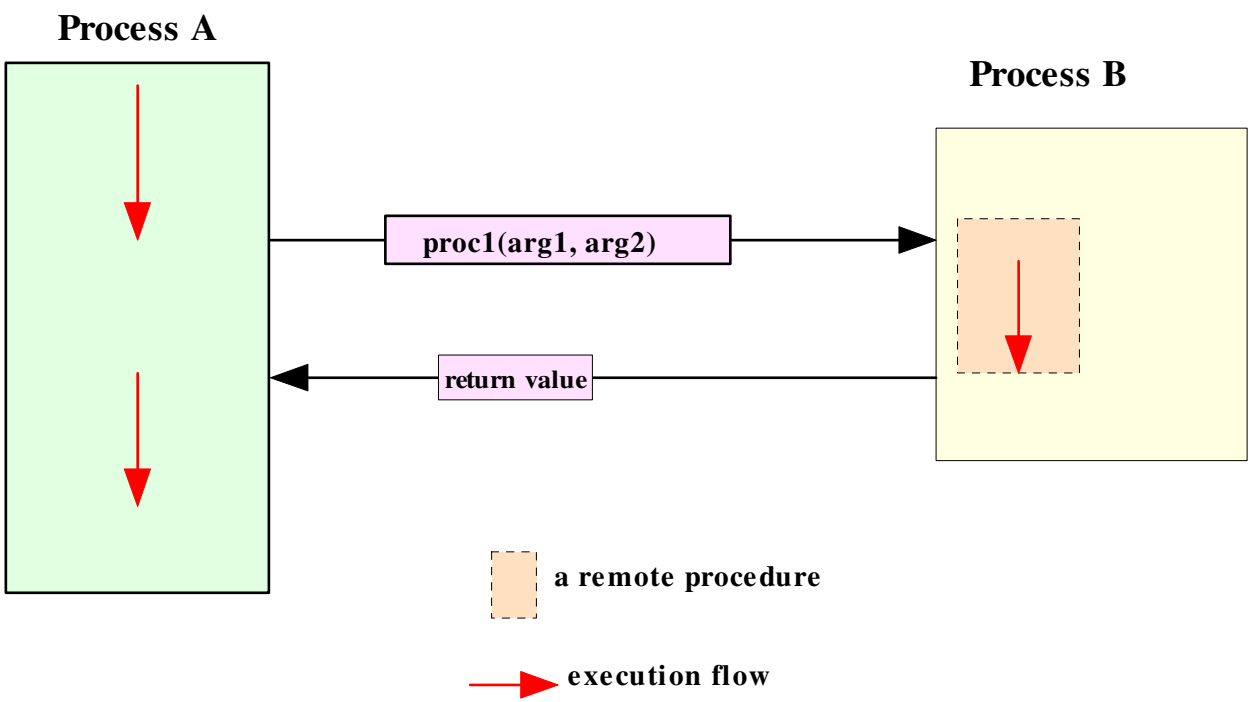
4. The proxy on host B unmarshalls the data received and issues a call to proc2.
5. The code in proc2 is executed and returns to the proxy on host B.
6. The proxy marshalls the return value and makes an IPC call to the proxy on host A.

7. The proxy received the return value, unmarshalls the data, and forwards the return value to proc1, which resumes its execution flow.

A remote procedure call
(the return execution path is not shown)

Remote Procedure Model

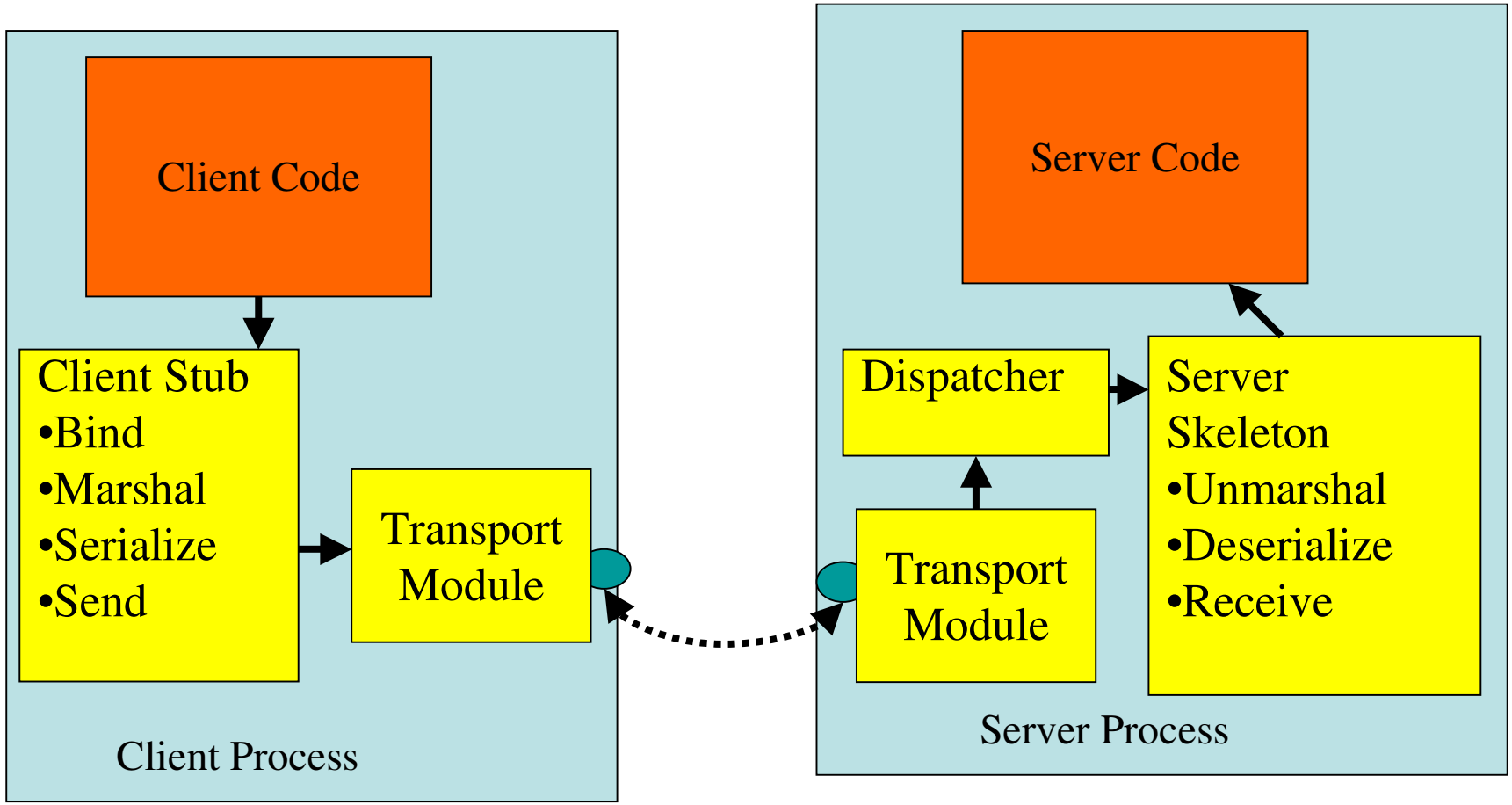
- **Programmer point of view.**
- **Remote Procedure Call (RPC) model interprocess communications proceed as procedure, or function, calls.**



Remote Procedure Call - 3

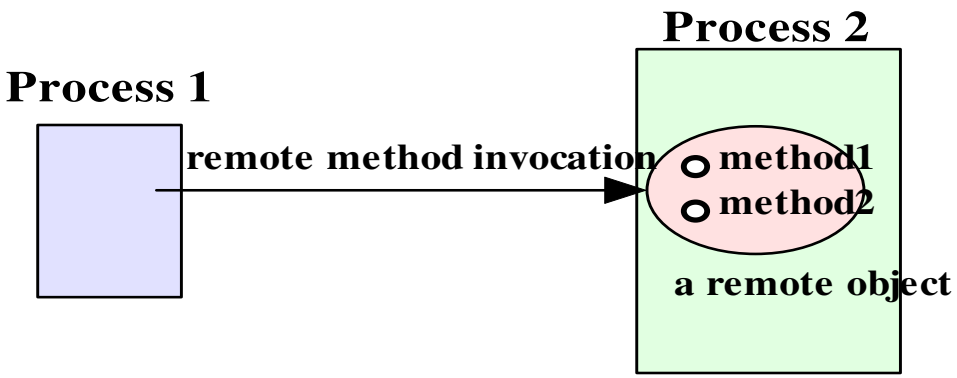
- **RPC allows programmers to build network applications using a programming construct similar to the local procedure call, providing a convenient abstraction for both interprocess communication and event synchronization.**
- **Since its introduction in the early 1980s, the Remote Procedure Call model has been widely in use in network applications.**
- **There are two prevalent APIs for Remote Procedure Calls.**
  - **The *Open Network Computing Remote Procedure Call*, evolved from the RPC API originated from Sun Microsystems in the early 1980s.**
  - **The *Open Group Distributed Computing Environment* (DCE) RPC.**
- **Both APIs provide a tool, *rpcgen*, for transforming remote procedure calls to local procedure calls to the stub.**

Functioning of RPC



Remote Method Invocation (RMI)

- Remote method invocation is the object-oriented equivalent of remote method calls.
- In this model, a process invokes the methods in an object, which may reside in a remote host.
- As with RPC, arguments may be passed with the invocation.



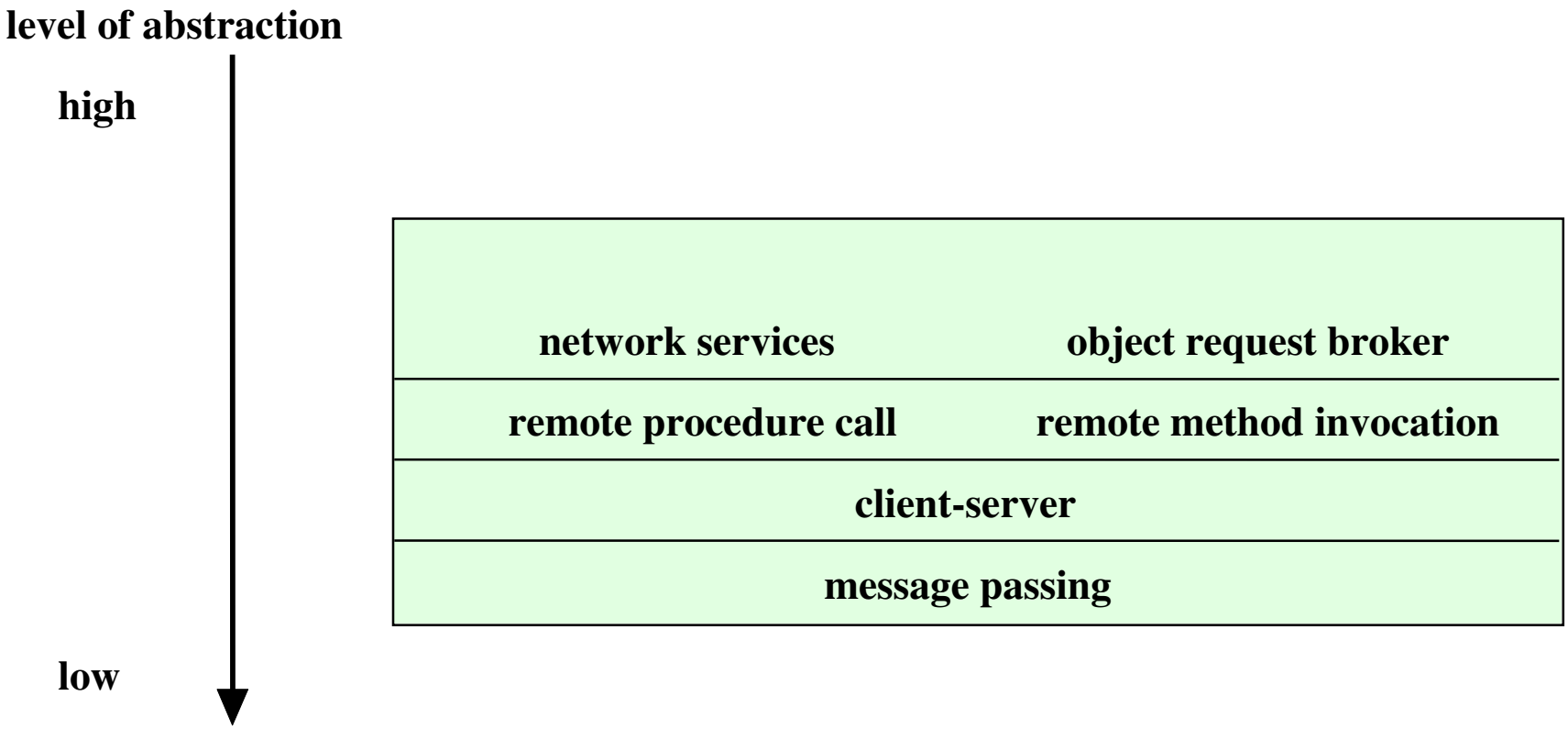
The Remote Method Call Paradigm

Do we need CORBA?

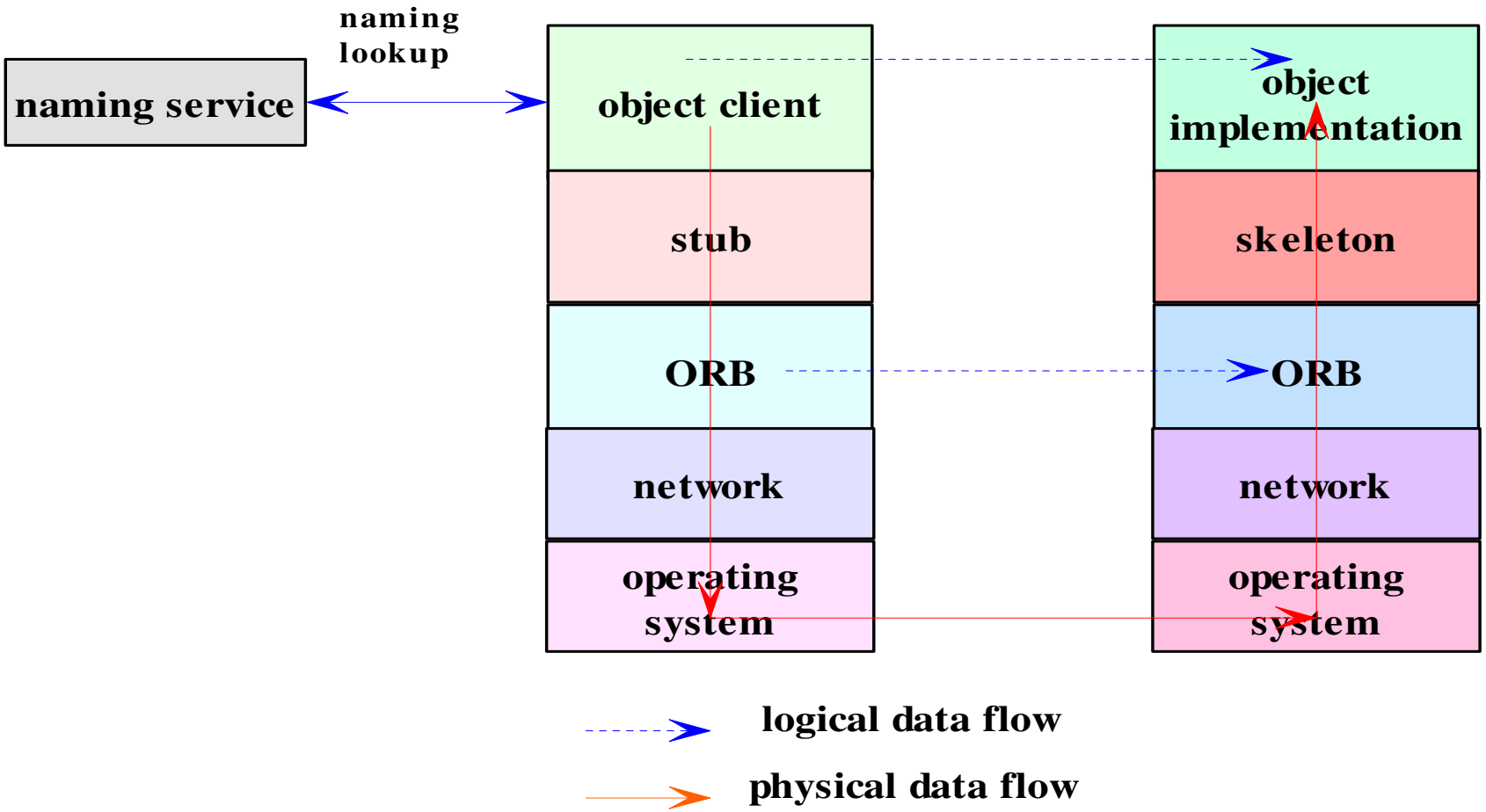
CORBA vs. Java RMI

- CORBA differs from the architecture of Java RMI in one significant aspect:
  - RMI is a proprietary facility developed by Sun MicroSystems, Inc., and supports objects written in the Java programming langugage only.
  - CORBA is an architecture that was developed by the Object Management Group (OMG), an industrial consortium.

Relating abstractions

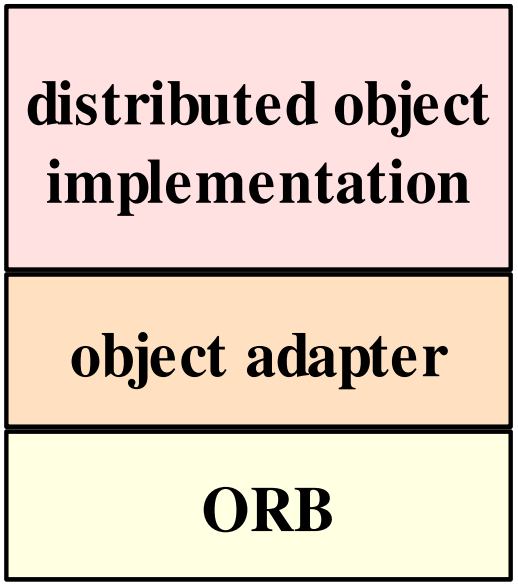


CORBA basic Architecture



Object Adapters

In the basic architecture of CORBA, the implementation of a distributed object interfaces with the skeleton to interact with the stub on the object client side. As the architecture evolved, a software component in addition to the skeleton was needed on the server side: an **object adapter**.



Object Adapter

- An object adapter simplifies the responsibilities of an ORB by assisting an ORB in delivering a client request to an object implementation (binding and polymorfism)
- When an ORB receives a client's request, it locates the object adapter associated with the object and forwards the request to the adapter.
- The adapter interacts with the object implementation's skeleton, which performs data marshalling and invoke the appropriate method in the object.

The ***Portable Object Adapter***

- There are different types of CORBA object adapters (BOA is deprecated)
- ***Portable Object Adapter***, or ***POA***, is a particular type of object adapter that allows an object implementation to function with different ORBs, hence the word portable.
- Support functionality:
  - Binding/mapping - Transparant activation
  - Persistency support - Object support functions

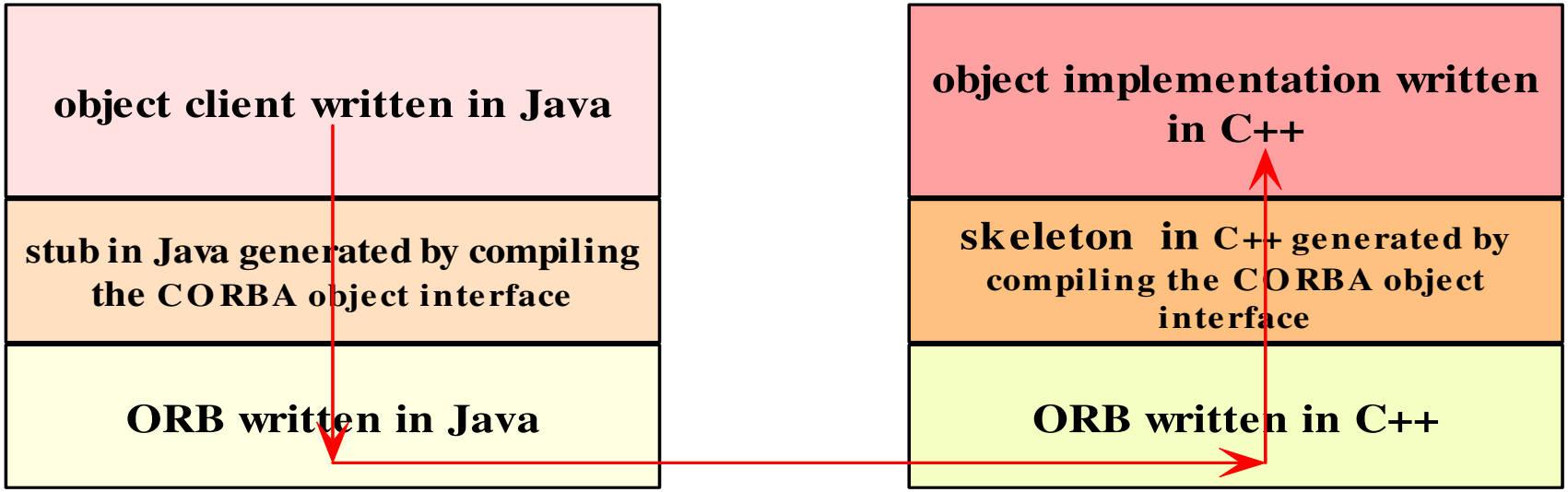
CORBA IDL

- A distributed object's signature is defined using an interface definition file.
- Since CORBA is language independent, the interface is defined using a universal language with a distinct syntax, known as the ***CORBA Interface Definition Language*** (***IDL***).
- The syntax of CORBA IDL is similar to Java and C++. However, object defined in a CORBA IDL file can be implemented in a large number of diverse programming languages, including C, C++, Java, COBOL, Smalltalk, Ada, Lisp, Python, and IDLScript.
- IDL support inheritance and polymorfism.
- For each of these languages, OMG has a standardized mapping from CORBA IDL to the programming language, so that a compiler can be used to process a CORBA interface to generate the proxy files needed to interface with an object implementation or an object client written in any of the CORBA-compatible languages.

The CORBA Interface file **Hello.idl**

```
01. module HelloApp
02. {
03.   interface Hello
04.   {
05.   string sayHello();
06.   oneway void shutdown();
07.   };
08. };
```

Cross-language CORBA application

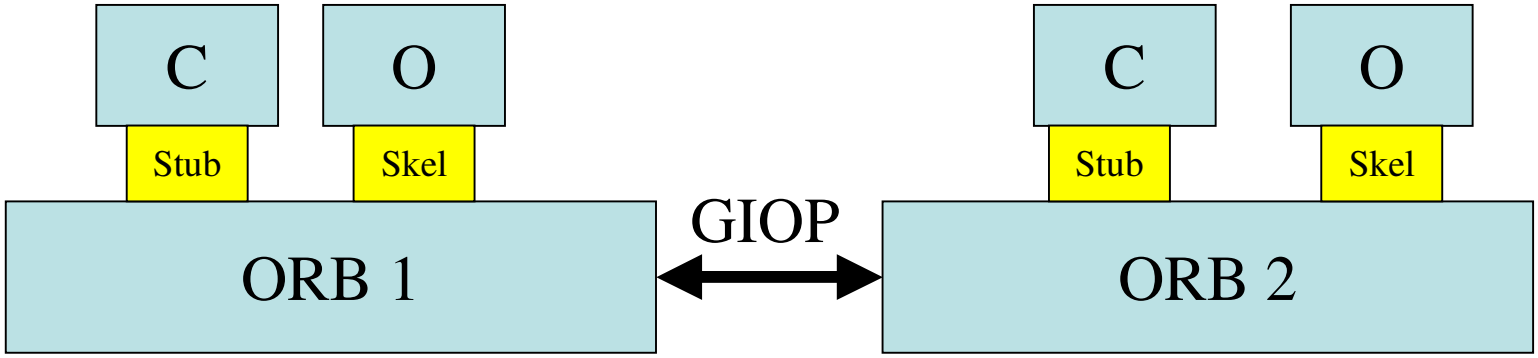


ORB Core Feature Matrix

http://www.jetpen.com/~ben/corba/orbmatrix.html

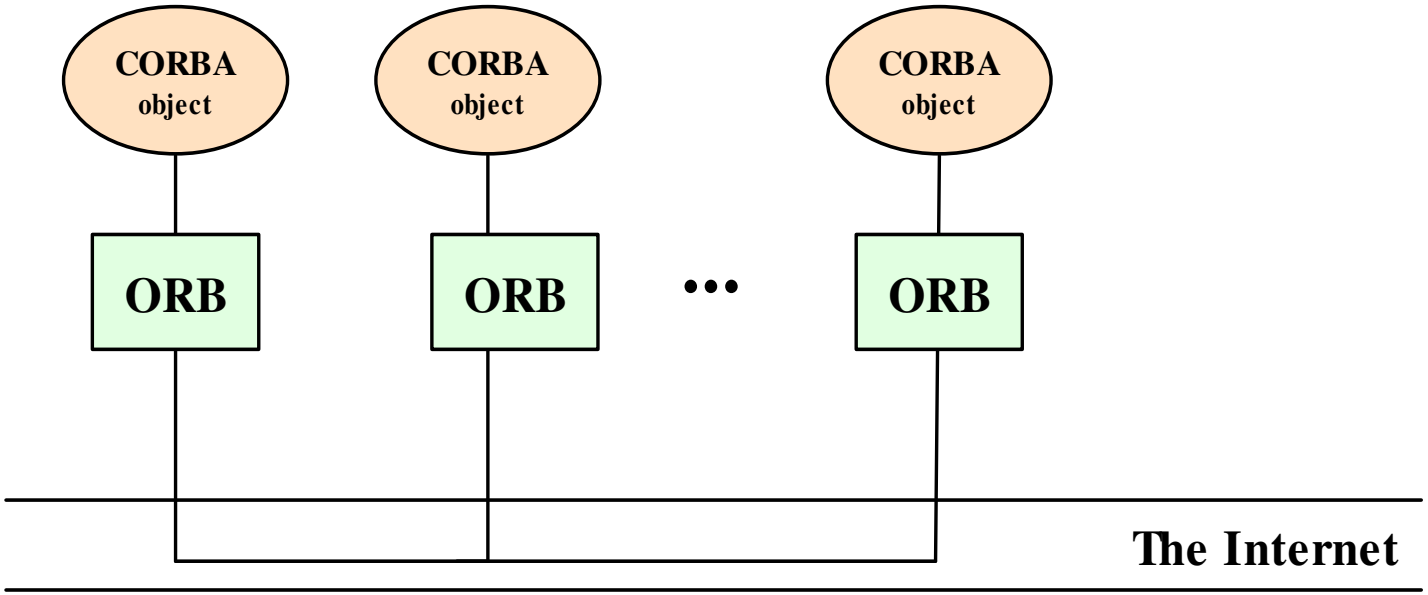
Inter-ORB Protocols

- To allow ORBs to be interoperable, the OMG specified a protocol known as the ***General Inter-ORB Protocol*** (***GIOP***), a specification which "provides a general framework for protocols to be built on top of specific transport layers."
- A special case of the protocol is the ***Inter-ORB Protocol*** (***IIOP***), which is the GIOP applied to the TCP/IP transport layer.



Object Bus

An ORB which adheres to the specifications of the IIOP may interoperate with any other IIOP-compliant ORBs over the Internet. This gives rise to the term "***object bus***", where the Internet is seen as a bus that interconnects CORBA objects



CORBA Object Services

(related specifications)

CORBA specify services commonly needed in distributed applications, some of which are:

- ***Naming Service***:
- ***Concurrency Service***:
- ***Event Service***: for event synchronization;
- ***Logging Service***: for event logging;
- ***Scheduling Service***: for event scheduling;
- ***Security Service***: for security management;
- ***Trading Service***: for locating a service by the type (instead of by name);
- ***Time Service***: a service for time-related events;
- ***Notification Service***: for events notification;
- ***Object Transaction Service***: for transactional processing.

Each service is defined in a standard IDL that can be implemented by a developer of the service object, and whose methods can be invoked by a CORBA client.

CORBA Naming Service

- To export a distributed object, a CORBA object server contacts a Naming Service to ***bind*** a symbolic name to the object The Naming Service maintains a database of names and the objects associated with them.
- To obtain a reference to the object, an object client requests the Naming Service to look up the object associated with the name (This is known as ***resolving*** the object name.)
- The API for the Naming Service is specified in interfaces defined in IDL, and includes methods that allow servers to bind names to objects and clients to resolve those names.

Naming Service

(Binding Objects)

- A CORBA distributed object is exported by an ***object server***.
- An ***object client*** retrieves a reference to a distributed object from a naming or directory service, to be described, and invokes the methods of the distributed object.

CORBA Object References

- A CORBA distributed object is located using an ***object reference***. Since CORBA is language-independent, a CORBA object reference is an abstract entity mapped to a language-specific object reference by an ORB, in a representation chosen by the developer of the ORB.
- For interoperability, OMG specifies a protocol for the abstract CORBA object reference object, known as the ***Interoperable Object Reference*** (***IOR***) protocol.
- Example of the string representation of an IOR [5]:


```
IOR:000000000000000d49444c3a677269643a312e3000000
00000000001000000000000004c000100000000015756c74
72612e6475626c696e2e696f6e612e696500000963000002
83a5c756c7472612e6475626c696e2e696f6e612e69653a67
7269643a303a3a49523a67726964003a
```

Interoperable Object Reference (IOR)

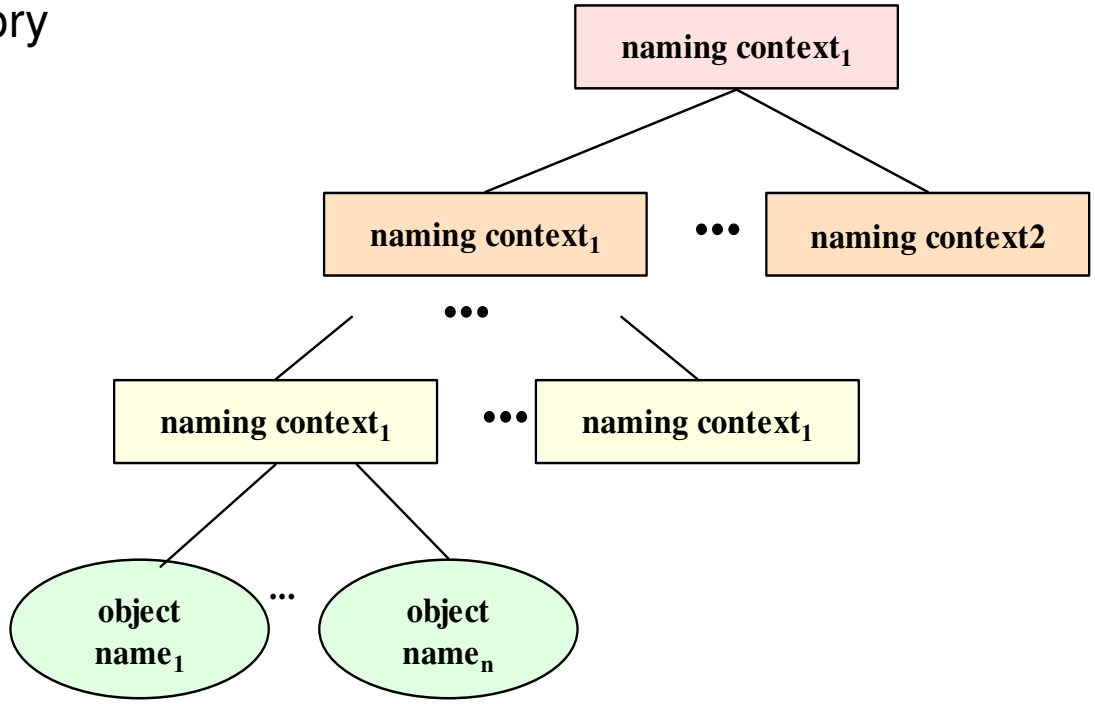
An IOR is a string that contains encoding for the following information:

- The type of the object.
- The host where the object can be found.
- The port number of the server for that object.
- An object key, a string of bytes identifying the object.

  The object key is used by an object server to locate the object.

CORBA Naming Service

To be as general as possible, the CORBA object naming scheme is necessary complex. Since the name space is universal, a standard naming hierarchy is defined in a manner similar to the naming hierarchy in a file directory



A Naming Context

- A naming context correspond to a folder or directory in a file hierarchy, while object names corresponds to a file.
- The full name of an object, including all the associated naming contexts, is known as a *compound name*. The first component of a compound name gives the name of a naming context, in which the second component is accessed. This process continues until the last component of the compound name has been reached.
- Naming contexts and name bindings are created using methods provided in the Naming Service interface.

A CORBA object name

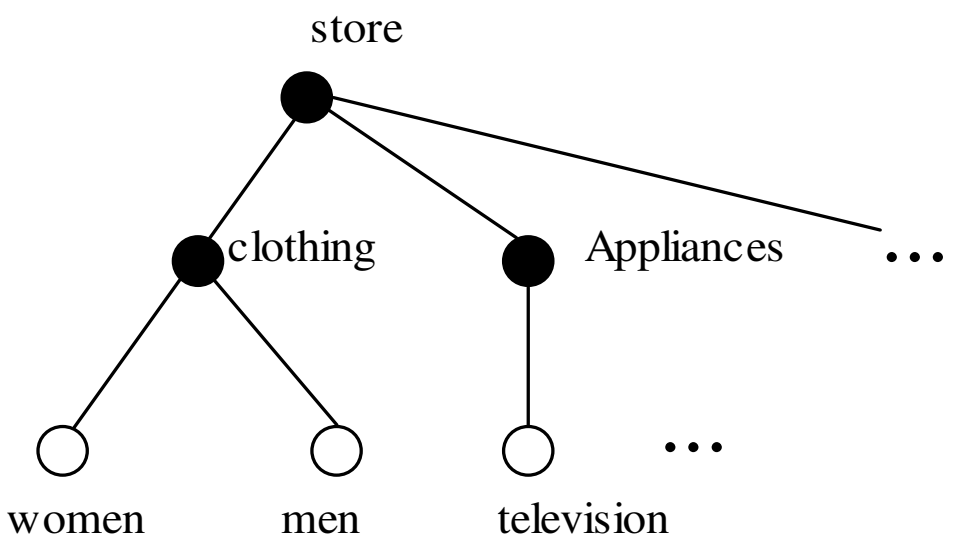
The syntax for an object name is as follows:

```
<naming context > …<naming context><object name>
```

where the sequence of naming contexts leads to the object name.

Example of a naming hierarchy

As shown, an object representing the men's clothing department is named store.clothing.men, where store and clothing are naming contexts, and men is an object name.



Interoperable Naming Service

The ***Interoperable Naming Service*** (***INS***) is a URL-based naming system based on the CORBA Naming Service, it allows applications to share a common initial naming context and provide a URL to access a CORBA object.

ORB products

There are a large number of proprietary as well as experimental ORBs available:

(See CORBA Product Profiles, http://www.puder.org/corba/matrix/)

- Orbix IONA
- Borland Visibroker
- PrismTech's OpenFusion
- Web Logic Enterprise from BEA
- Ada Broker from ENST
- Free ORBs (Orbit, OmniORB)

Conclusion

- CORBA enables:
  - Interoperablility
  - Platform independence
  - Object Oriented distributed application development
  - Widely supported
- CORBA is not new, just a next step

References

- OMG (www.omg.org)
- *Brose, Vogel, Duddy*
  "Java Programming with CORBA"
- *Aloso,Casati, Kuno, Machiraju*
  "Web Services - Concepts, Architectures and Applications"
- *Liu*
  "Distributed Computing - principles and applications"