Object oriented distributed architectures

Isidro Calvo
Dpto. Ingenieria de Sistemas y Automatica
Escuela Superior de Ingenieria de Bilbao
6th October 2005

Index

- Distributed Architectures in Control Systems
- Comparation of Distributed Architectures
- The CORBA-RT specification
- Hands-on Tutorial

Index

- **Distributed Architectures in Control Systems**
- Comparation of Distributed Architectures
- The CORBA-RT specification
- Hands-on Tutorial

Introduction

- Control systems increasingly distributed
  - Systems formed by several cooperating nodes
- Nodes more intelligent
  - Hardware more powerful
  - Price of the hardware is down
- Advantages:
  - More powerful
  - Robustness
  - Flexibility
  - Conceptually more simple
  - New possibilities (Built-in components to plug in, etc)
- Nodes exchange more information
  - Communications get more complicated

OO distributed architectures

- Benefits of object oriented technologies:
  - Flexibility, modularity, scalability, clarity...
- Added benefits:
  - Independence of the communications programming
  - Provide added value services
  - Introduce fault-tolerance mechanisms
  - Allow the construction of object oriented models of physical devices
- Distributed architectures more used:
  - CORBA, DCOM, Java / RMI
- Examples found in the industry:
  - CORBA-RT, OPC, ProfiNET, etc.

Index

- Distributed Architectures in Control Systems
- **Comparation of Distributed Architectures**
  - CORBA
  - DCOM
  - Java / RMI
- The CORBA-RT specification
- Hands-on work

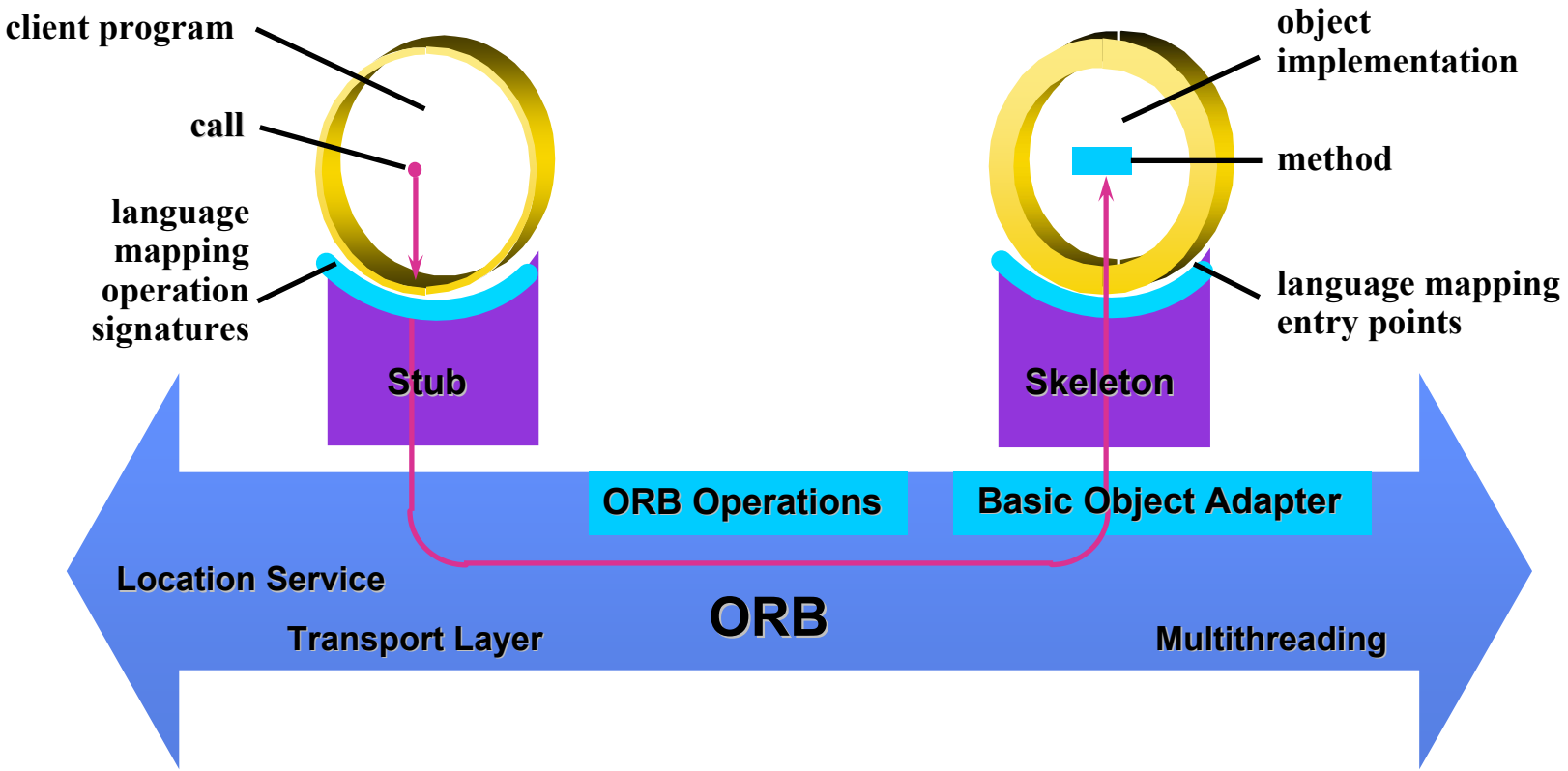
Index

- Distributed Architectures in Control Systems
- **Comparation of Distributed Architectures**
  - **CORBA**
  - DCOM
  - Java / RMI
- The CORBA-RT specification
- Hands-on work

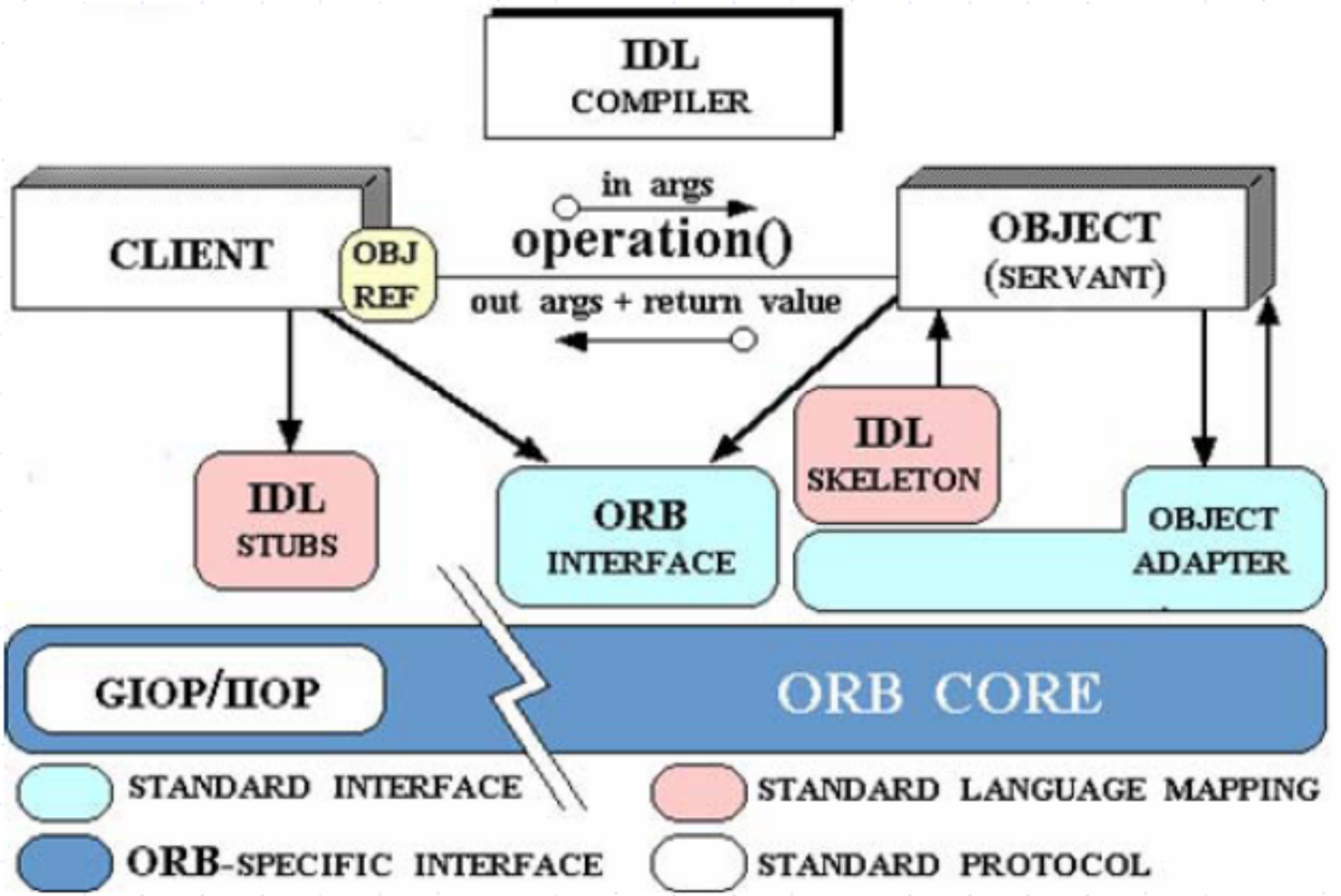
CORBA

- ***C**ommon **O**bject **R**equest **B**roker **A**rchitecture*
  - Open specification proposed by the OMG
    - http://www.omg.org
  - Objective:
    - Clients use remote objects as if they were local
  - Caracteristics:
    - Multilingual:
      - Ada, C, C++, Smalltalk, Java, Python, COBOL, …
    - Multiplatform:
      - Windows, Linux, Unix, MacOS, …
    - Interoperability between lenguages and platforms
    - Multiple vendors (some are freeware products)

Object Transparecy



Components of CORBA



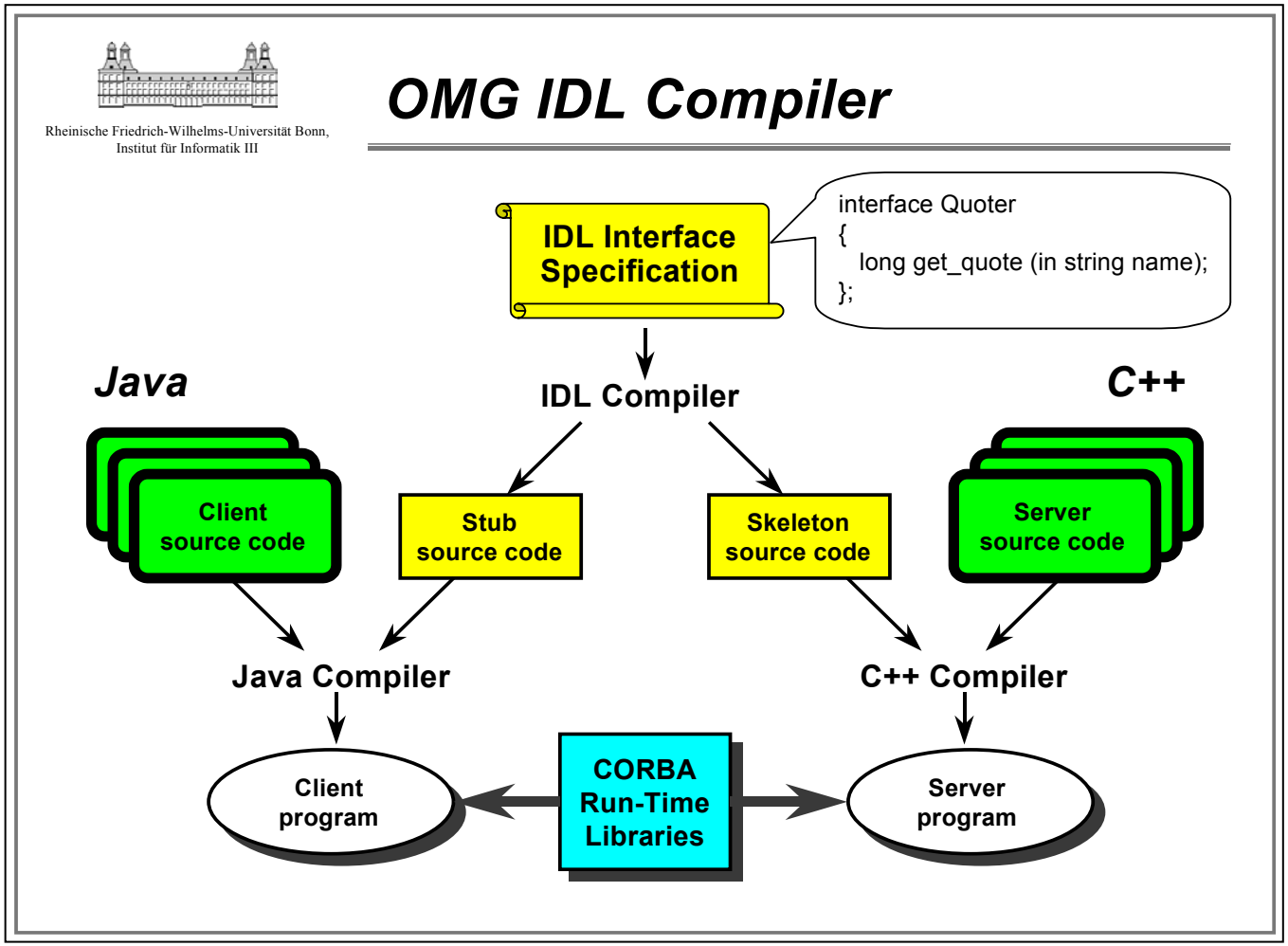
IDL

- ***I****nterface* ***D****efinition* ***L****anguage*
  - Allows remote clients to know what operations are available
  - Declarative-only language
  - IDL compilers generate the source code included in the servers (*Skeletons*) and clients (*Stubs*)

IDL (Example)

```
module RobotObjects {
        struct Position {
             double Cord_x, Cord_y, Cord_z;
        };
        exception Unknown{};

        interface Arm {
            Position GetCurrentPosition();
            void NewPosition(in Position FinalPosition)
                raises (Unknown);
        };

        interface Hand {
             readonly attribute string HandType;
             void ChangeHand(in long NewType) raises (Unknown);
        } ;
};
```

IDL Compilation



Object Adapters

- Link between CORBA objects and the ORB
  - Registration of CORBA objects
  - Generation of references for client objects
  - Execution of server objects
  - Management of the querys over the CORBA objects
  - Aceptation of the remote clients querys
  - Policies of the objects

Inter ORB Protocols

- Standard protocol (Defined by the OMG)
  - Allows the interoperation between products from different vendors
- Defines the messages between the ORBs to exchange information
- GIOP (General Inter ORB Protocol)
  - Specification to build particular protocols
  - IIOP (Internet Inter ORB Protocol)
    - Built over the TCP/IP protocols
  - CANIOP (CAN Inter ORB Protocol)
  - ...

CORBA Services

- Provide added functionality to the CORBA specification
  - Naming Service
  - Events Service
  - Notification Service
  - Concurrency Control Service
  - Security Service
  - ...

Index

- Distributed Architectures in Control Systems
- **Comparation of Distributed Architectures**
  - CORBA
  - **DCOM**
  - Java / RMI
- The CORBA-RT specification
- Hands-on work

COM / DCOM / COM+

- ***D****istributed* ***C****omponent* ***O****bject* ***M****odel*
  - Specification defined and owned by Microsoft Corporation
  - Architecture similar to CORBA
  - Some differences
    - Component oriented
  - Origins:
    - OLE (***O****bject* ***L****inking and* ***E****mbedding*): Exchange of information inside Windows

Inside DCOM

- Component-oriented technology
  - Not code oriented
  - Components are binary entities (already compiled)
  - Binary code inserted in different applications
  - Componets are totally encapsuled

COM IDL

- Separation between code and interface
  - Use of an ***I****nterface* ***D****efinition* ***L****anguage*
  - Similar to CORBA IDL
- Generation of a CLSID (*Class Identifier*)
  - Ex: {06DD38D3-D187-11CF-A80D-00C04FD74AD8}

Servers and Clients

- Servers:
  - Registration of the interface in the Windows registry under HKEY_CLASSES_ROOT
    - Ex:
      - **Name:** AudioVBScript
      - **CLSID:** {4EE17959-931E-49E4-A2C6-977ECF3628F3}
  - View [regedit]
- Clients:
  - Use the **CLSID** to create a pointer to an interface
  - The component may be used

DCOM Services

- MTS (***M****icrosoft* ***T****ransaction* ***S****ervice*)
  - Transactions Control
  - Security
  - Resource management
  - Concurrency
  - …
- MSMQ (***M****icrosoft* ***Q****ueuing* ***S****ervice)*
  - Distribution of events (messages) among several clients
  - Equivalent to the CORBA Event Service

Index

- Distributed Architectures in Control Systems
- **Comparation of Distributed Architectures**
  - CORBA
  - DCOM
  - **Java / RMI**
- The CORBA-RT specification
- Hands-on work

Java / RMI

- RMI *(**R**emote **M**ethod **I**nvocation)*
- Only used with Java platform
  - Over the JVM
- Integrated in the J2EE platform
  - Tools of J2EE:
    - Services (Events, security, transactions, …)

Comparation of technologies

- CORBA
  - Designed from scratch to build distributed systems
  - Multi-platform/lingual/vendor
  - Reliable, powerful and robust
  - Base of *CORBA minimal* and *CORBA-RT*
- DCOM
  - Excellent integration within Windows
  - Base of industry standards (OPC, Profinet)
- Java
  - Excellent integration with Java / Internet environments
  - Lower performance

Index

- Distributed Architectures in Control Systems
- Comparation of Distributed Architectures
- **The CORBA-RT specification**
- Hands-on Tutorial

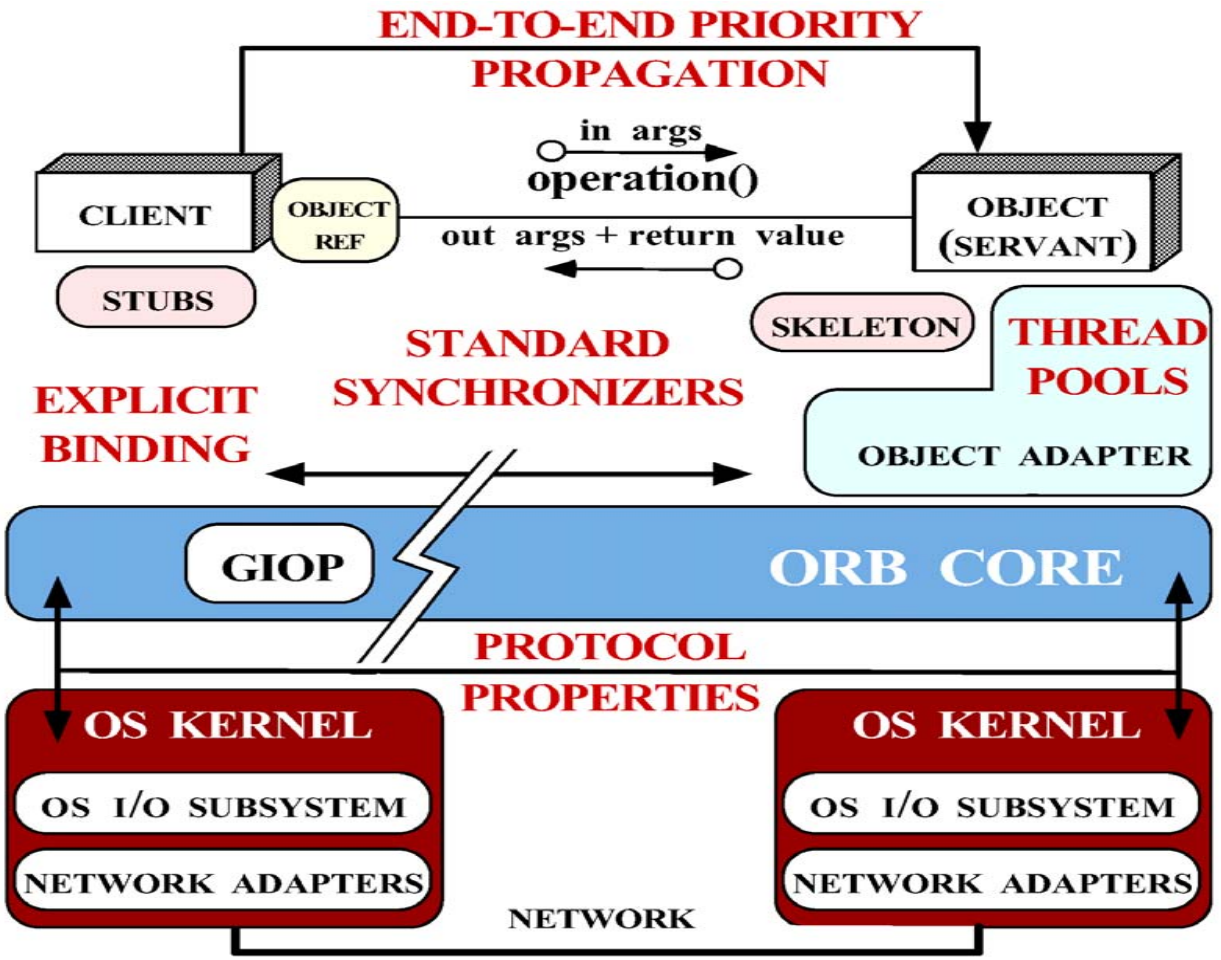
Limitations of standard CORBA

- Standard CORBA not designed to work in real-time systems
- It lacks:
  - Bandwidth management
  - QoS management policies
  - Control mechanims for the use of the resources
  - Integration mechanisms with the underlaying OS's
  - ...

CORBA-RT

- Extension of CORBA standard
- Define standard features to support end-to end predictability in fixed-priority applications
- New interfaces that allow applications to configure and control:
  - Processor resources:
    - Threadpools, priority mechanisms, intra-process mutexes
  - Communication resources:
    - Protocol properties, explicit bindings, private connections,...
  - Memory resources
    - Bounding the size of threadpools

ORB features for CORBA-RT



Index

- Distributed Architectures in Control Systems
- Comparation of Distributed Architectures
- The CORBA-RT specification
- **Hands-on Tutorial**

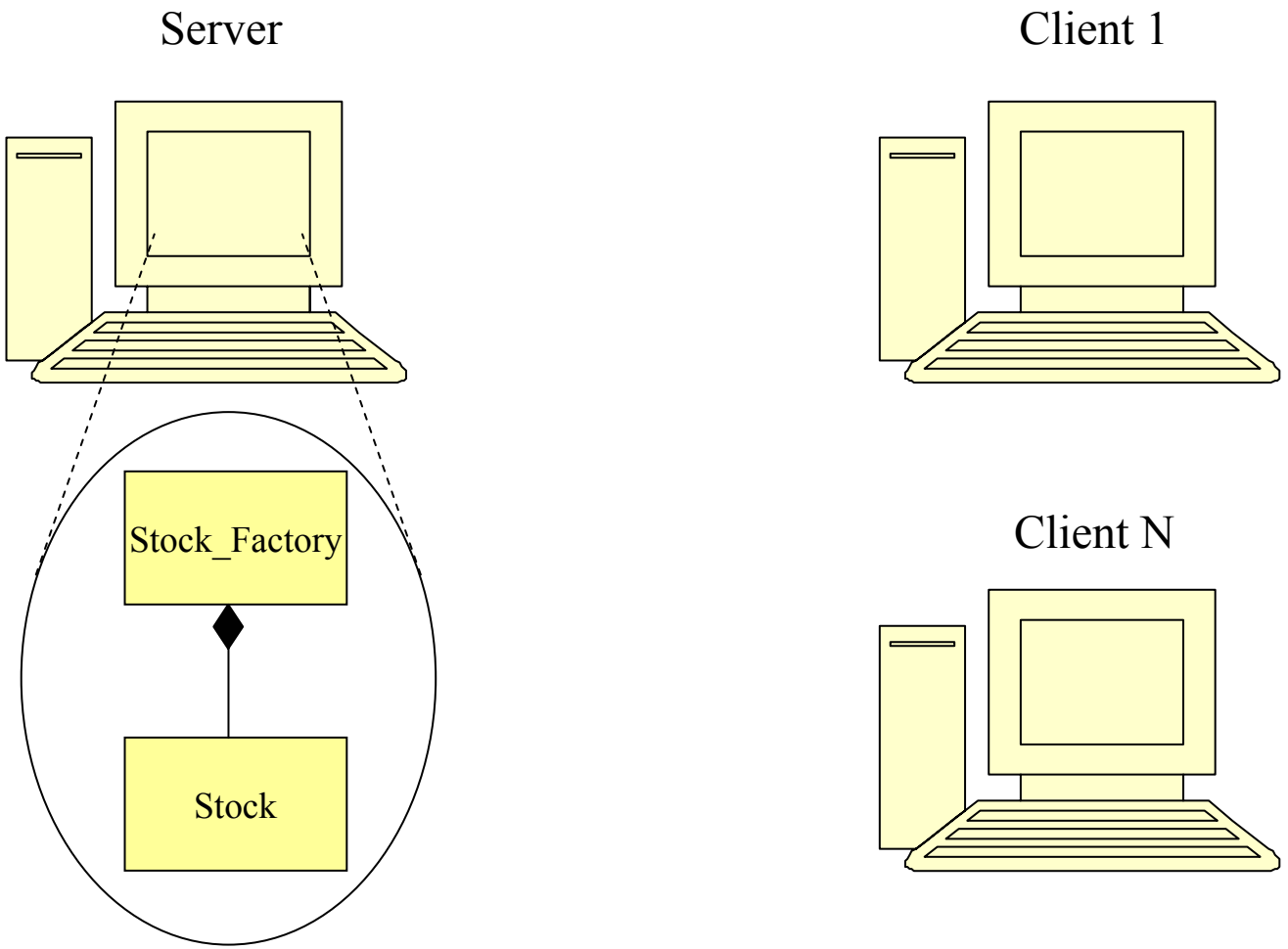
Hands-on tutorial

- Building a very simple CORBA application
- Using the CORBA Naming Service
- Interoperability with other ORB's (TAO – JacORB)
- Creation of a simple application

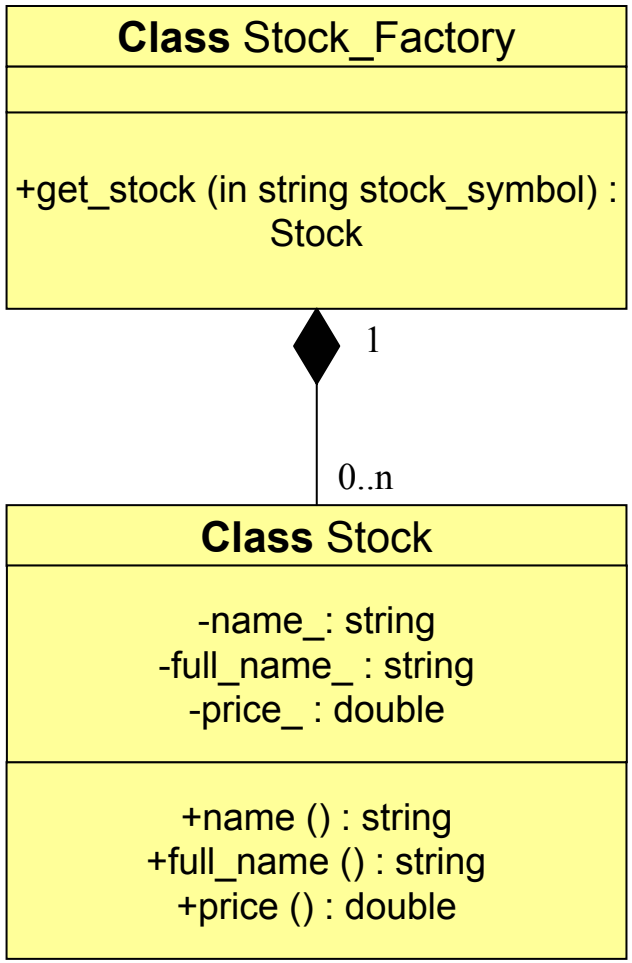
Hands-on tutorial

- **Building a very simple CORBA application**
- Using the CORBA Naming Service
- Interoperability with other ORB's (TAO – JacORB)
- Creation of a simple application

Simple Example (Description)



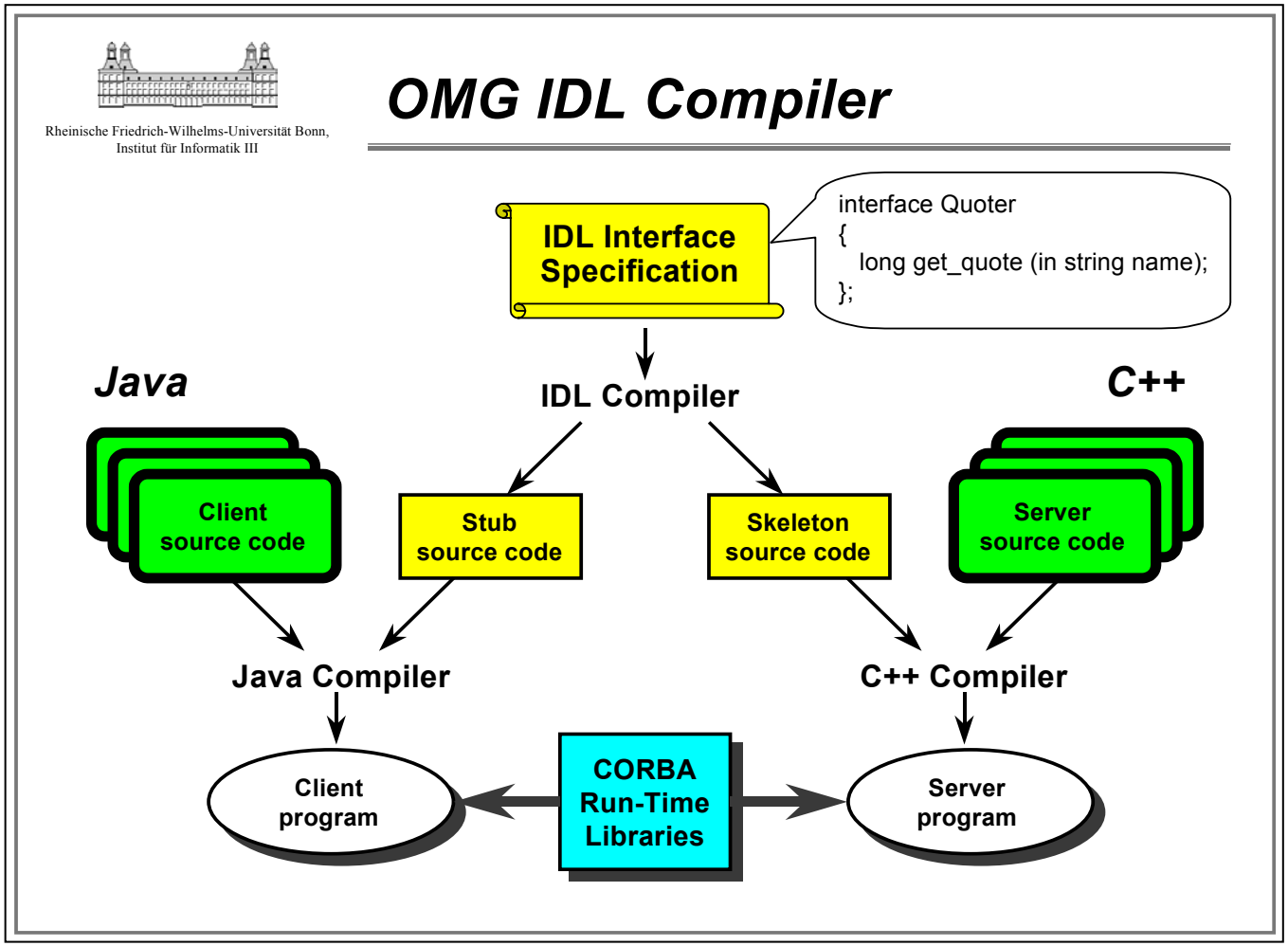
Simple Example (CORBA Objects)



Example IDL

```
module Quoter {
  exception Invalid_Stock_Symbol {};

  interface Stock;

  interface Stock_Factory {
    Stock get_stock (in string stock_symbol)
      raises (Invalid_Stock_Symbol);
  };

  interface Stock {
    readonly attribute string symbol;

    readonly attribute string full_name;

    double price ();
  };
};
```

IDL Compilation



IDL Compilation

```
TAO> tao_idl -GI Quoter.idl
TAO> dir
total 156
-rw-r—r--  1  user1 lab … QuoterC.cpp
-rw-r—r--  1  user1 lab … QuoterC.h
-rw-r—r--  1  user1 lab … QuoterC.i
-rw-r—r--  1  user1 lab … QuoterI.cpp      <- Quoter_i.cpp
-rw-r—r--  1  user1 lab … Quoter.idl
-rw-r—r--  1  user1 lab … QuoterI.h        <- Quoter_i.h
-rw-r—r--  1  user1 lab … QuoterS.cpp
-rw-r—r--  1  user1 lab … QuoterS.h
-rw-r—r--  1  user1 lab … QuoterS.i
-rw-r—r--  1  user1 lab … QuoterS_T.cpp
-rw-r—r--  1  user1 lab … QuoterS_T.h
-rw-r—r--  1  user1 lab … QuoterS_T.i
TAO>
```

Modifications in Quoter_i.h

- class Quoter_Stock_Factory_i
  - Add 2 private objects *Quoter_Stock_i*

```
private:
  Quoter_Stock_i rhat_;
  Quoter_Stock_i msft_;
```

- class Quoter_Stock_i
  - Modify the constructor to be:

```
Quoter_Stock_i (const char *symbol,
                const char *full_name,
                CORBA::Double price)
```

  - Add Stock private attributes

```
private:
  std::string symbol_;
  std::string full_name_;
  CORBA::Double price_;
```

Modifications in Quoter_i.cpp

- Includes: Depending on the application code
- Implementation Code:
  - Class Quoter_Stock_Factory_i:
    - ***Constructor*:** Create Stock objects available
    - ***get_stock*** method: Returns a pointer to a *Stock* object
  - Class Quoter_Stock_i
    - ***Constructor*:** Fills the private attributes (symbol_, full_name & price_)
    - ***symbol*** method: Returns a string pointer with the symbol
    - ***full_name*** method: Returns a string pointer with the full_name
    - ***price*** method: Returns a CORBA::Double object with the price

The Client (client.cpp)

- Include:
  - "QuoterC.h" + Necessary ".h" files
- Code:
  - Initialize the ORB
  - Create a reference to access a Stock_factory "remote" object
  - Use the object as if it were local
  - Capture possible CORBA exceptions (including CORBA::User exceptions)
  - Destroy the ORB object

The Server (server.cpp)

- Include:
  - "Quoter_i.h" + Necessary ".h" files
- Code:
  - Initialize the ORB
  - Create a reference to access the POA and activate it
  - Create the servant and activate it
  - Obtain an object reference and print it out
  - Run the ORB
  - Capture any possible CORBA exception
  - Destroy the POA and ORB objects

Code Compilation

- Generation of the makefiles with mwc.pl
  It is necessary to obtain the Simple.mpc file

```
TAO> mwc.pl -type gnuace                    <- generate makefile
```

- Check that the following files appear:

```
Makefile          Makefile.Simple_Client    Makefile.Simple_Server
```

```
TAO> make                                   <- compile files
```

- Check that the following files appear:

```
client            server
```

Execution

- Execution of the server in one console

```
TAO> server > ref.ior
```

- Check that the following file appears:

```
ref.ior
```

- Open the ref.ior file

```
IOR:010000001d000000000123fa1234ae562400000ab320000012234cbbb0…
```

- Execution of the client in another console

```
TAO> client file://ref.ior MSFT RHAT XXXX
The price of a stock in “Microsoft, Inc.” is $91
The price of a stock in “RedHat, Inc.” is $210
Invalid stock symbol <XXXX>
```

Hands-on tutorial

- Building a very simple CORBA application
- **Using the CORBA Naming Service**
- Interoperability with other ORB's (TAO – JacORB)
- Creation of a simple application

Why the Naming Service?

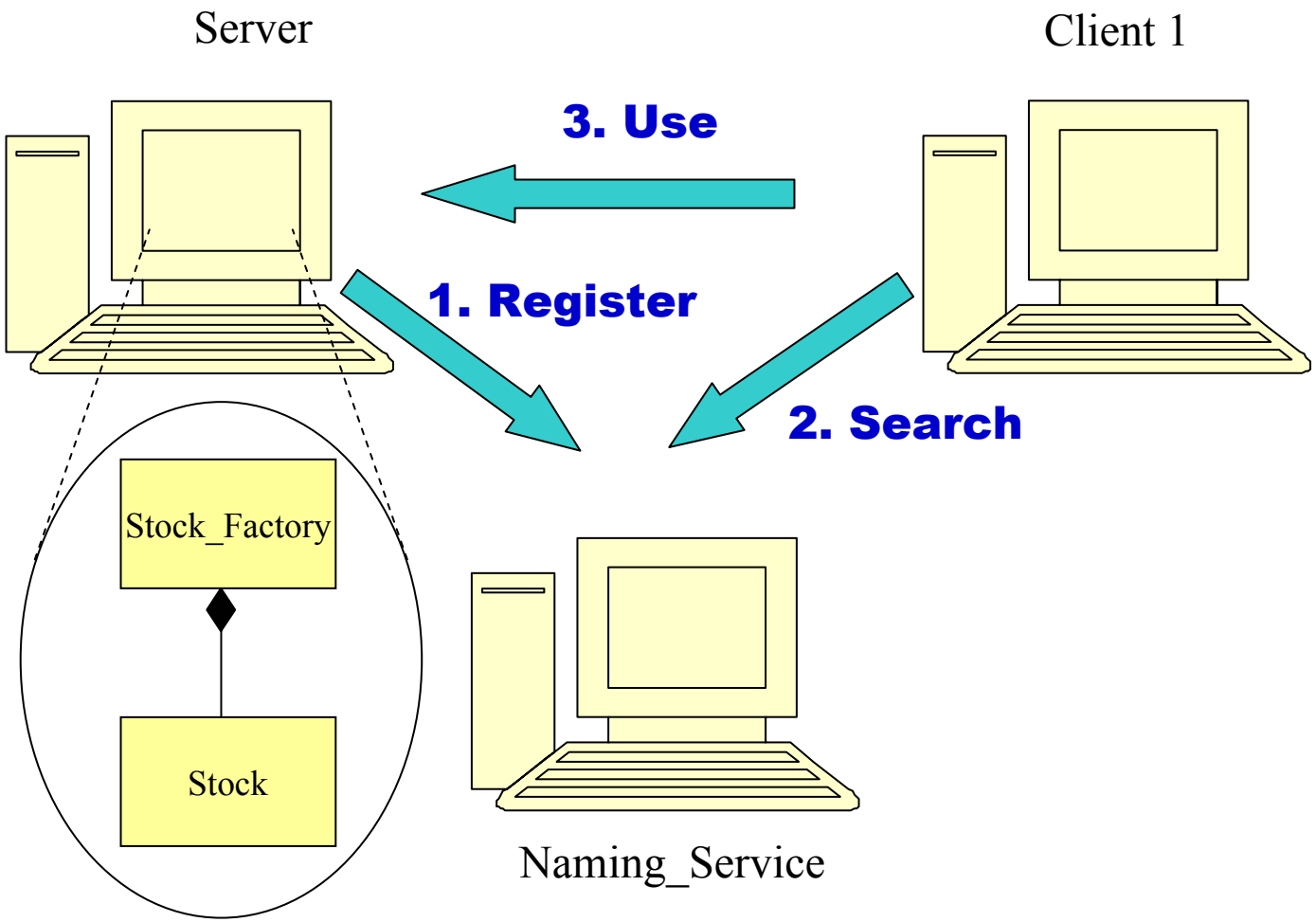
- So far remote objects were used through stringfied IORs
  - Methods ***object_to_string*** & ***string_to_object***
- However not very confortable
  - E.g. Require the IOR to be passed with a floppy disk between computers
- The Naming Service uses a table:
  - Name <-> IOR
- Servers register objects with a Name in the Naming Service
- Clients search for a Name and the Naming Service returns the IOR
- Clients use the IOR to perform operations with the remote objets

Same CORBA objects

- Same CORBA objects => Same IDL, Quoter_i.h & Quoter_i.cpp

```
module Quoter {
  exception Invalid_Stock_Symbol {};

  interface Stock;

  interface Stock_Factory {
    Stock get_stock (in string stock_symbol)
      raises (Invalid_Stock_Symbol);
  };
  interface Stock {
    readonly attribute string symbol;
    readonly attribute string full_name;
    double price ();
  };
};
```

Naming Service



The Server (server.cpp)

- Include:
  - **"orbsvcs/CosNamingC.h"** file
- Code:
  - Initialize the ORB
  - Create a reference to access the POA and activate it
  - Create the servant and activate it
  - **Create a reference to the Naming Service (Explicitly launched)**
  - **Registration in the Naming Service: Object_Name + Object_Reference (bind/rebind)**
  - Run the ORB
  - Capture any possible CORBA exception
  - Destroy the POA and ORB objects

The Client (client.cpp)

- Include:
  - **"orbsvcs/CosNamingC.h"** file
- Code:
  - Initialize the ORB
  - **Create a reference to the Naming Service (Explicitly launched)**
  - **Search the *"Object_Name"* in the Naming Service (resolve) and obtains a reference**
  - Use the object as if it were local
  - Capture possible CORBA exceptions (including CORBA::User exceptions)
  - Destroy the ORB object

Code Compilation

- Generation of the makefiles with mwc.pl
  It is necessary to obtain the Naming.mpc file

```
TAO> mwc.pl -type gnuace                    <- generate makefile
```

- Check that the following files appear:

```
Makefile          Makefile.Naming_Client    Makefile.Naming_Server
```

```
TAO> make                                   <- compile files
```

- Check that the following files appear:

```
client            server
```

Execution with multicast

- Execution of the Naming_Service

```
TAO> cd /home/TAO_CORBA/ACE_wrappers/TAO/orbsvcs/Naming_Service
TAO> Naming_Service –m 1
```

- Execution of the server in another console

```
TAO> server
```

- Execution of the client in a third console

```
TAO> client MSFT RHAT XXXX
The price of a stock in “Microsoft, Inc.” is $91
The price of a stock in “RedHat, Inc.” is $210
Invalid stock symbol <XXXX>
```

Execution without multicast

- Execution of the Naming_Service

```
TAO> cd /home/TAO_CORBA/ACE_wrappers/TAO/orbsvcs/Naming_Service
TAO> Naming_Service -m 0 -ORBEndpoint iiop://localhost:23456
```

- Execution of the server in another console

```
TAO> server -ORBInitRef
   NameService=corbaloc:iiop:localhost:23456/NameService
```

Execution of the client in a third console

```
TAO> client -ORBInitRef
   NameService=corbaloc:iiop:localhost:23456/NameService MSFT
   RHAT XXXX
The price of a stock in "Microsoft, Inc." is $91
The price of a stock in "RedHat, Inc." is $210
Invalid stock symbol <XXXX>
```

Analize the Network Traffic

- It is possible to analize the Network Traffic with **EtherReal**
- Traffic inside the computer and/or with other computers

Hands-on tutorial

- Building a very simple CORBA application
- Using the CORBA Naming Service
- **Interoperability with other ORB's (TAO – JacORB)**
- Creation of a simple application

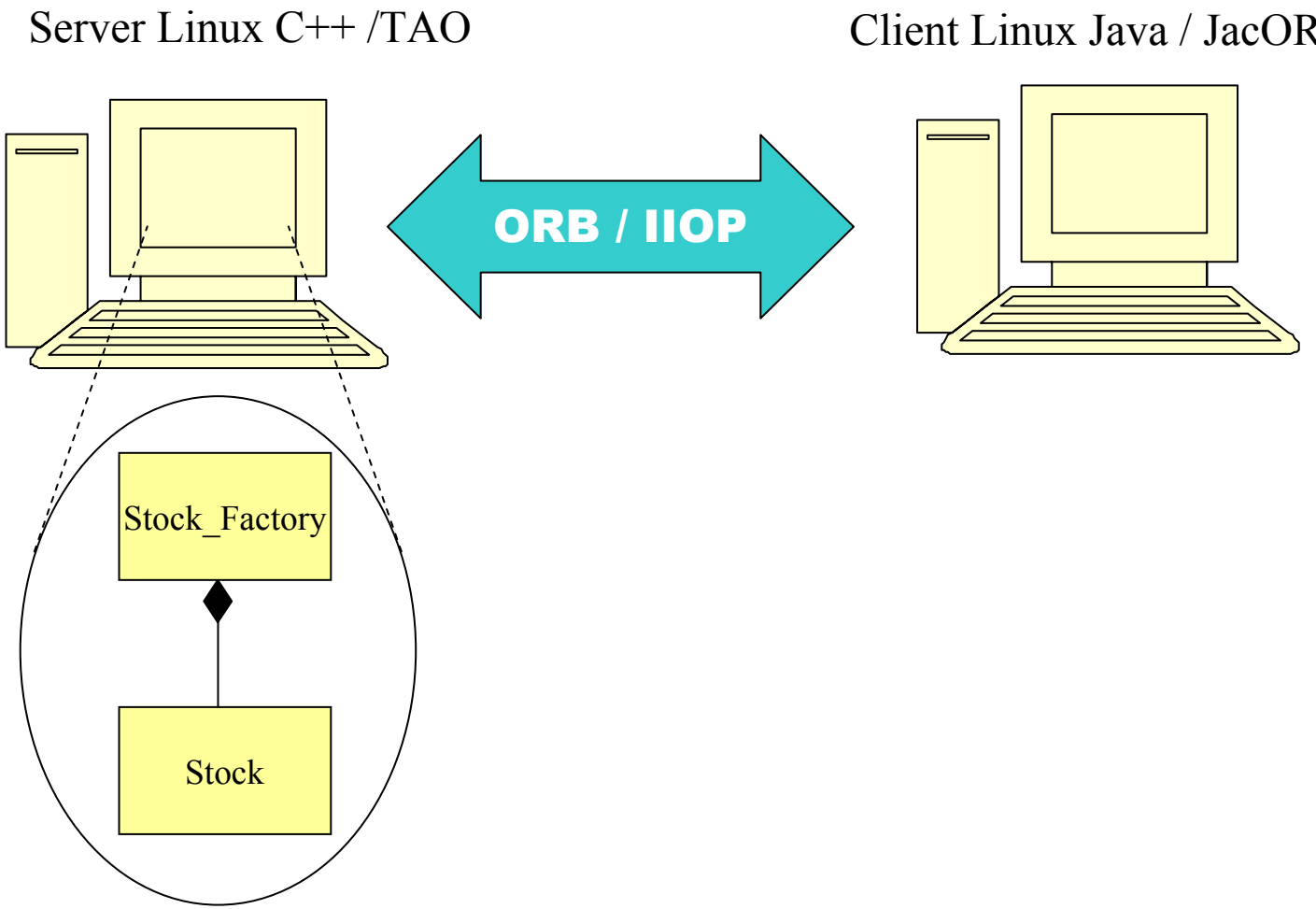
Interoperability

- One of the main advantages of CORBA is interoperatibility:
  - Different languages (C++, Java, …)
  - Different platforms (Linux, Windows, …)
  - Different products (TAO, JacORB, Visibroker, Orbix, MICO, …)

ORBs used

- Server written in C++ with TAO (ORB with real-time characteristics)
- Client written in Java with JacORB (Easy to build user friendly applications for operators)
- Both ORBs are freeware

Interoperability Example



Server (Same as before)

- Execution of the Naming_Service:

```
TAO> cd
   /home/TAO_CORBA/ACE_wrappers/TAO/orbsvcs/Nam
   ing_Service
TAO> Naming_Service -m 0 -ORBEndpoint
   iiop://x.x.x.x:2345x
```

- Execution of the server in another console

```
TAO> server -ORBInitRef
   NameService=corbaloc:iiop:x.x.x.x:2345x/Name
   Service
```

Client (Compiling the client)

- Open a session in another computer

```
TAO> ssh userx@port-grad01.det.nat.ua.pt
```

- Copy the jacorb.properties and the Client.java files in the local directory

```
TAO> cp ../common/Naming/jacorb.properties .
TAO> cp ../common/Naming/Client.java .
```

- Compile the Quoter.idl file with the JacORB IDL compiler

```
TAO> idl Quoter.idl
```

- Compile the Client.java file with javac

```
TAO> javac Client.java
```

- Move the resulting file (Client.class) to Quoter

```
TAO> mv Client.class Quoter
```

Client (Executing the client)

- Modify with the local address in the jacorb.properties file:

  `ORBInitRef.NameService=corbaloc::x.x.x:port/NameService`

- Execute

```
TAO> jaco Quoter.Client MSFT RHAT XXX
The price of a stock in “Microsoft, Inc.” is $91
The price of a stock in “RedHat, Inc.” is $210
Invalid stock symbol <XXXX>
```

Hands-on tutorial

- Building a very simple CORBA application
- Using the CORBA Naming Service
- Interoperability with other ORB's (TAO – JacORB)
- **Creation of a simple application**

Exercise IDL

```
interface Messenger {
    short send_message (
                in string user_name,
                in string subject,
                in string message);
};
```