

The Architecture of CORBA

(**C**ommon **O**bject **R**equest **B**roker
Architecture)

Agenda

➤ **Motivation**

- The Broker Architecture
- CORBA Architecture
- ORB-Interoperability
- Conclusion
- Book-References

Distributed applications cause a lot of problems

- Participating systems may be heterogeneous
- Access to remote services has to be location transparent
- State of objects has to be kept persistent and consistent
- Remote objects have to be found and activated
- Security has to be dealt with

Distributed application have a lot of advantages

- Scalability
 - Server replication
 - Thin, heterogeneous clients
- Re-usability
- Partitioned functionality = easy updating of either clients or servers

so we want an architecture that...

- ...supports a remote method invocation paradigm

- ...provides location transparency

- ...allows to add, exchange, or remove services dynamically

- ...hides system details from the developer

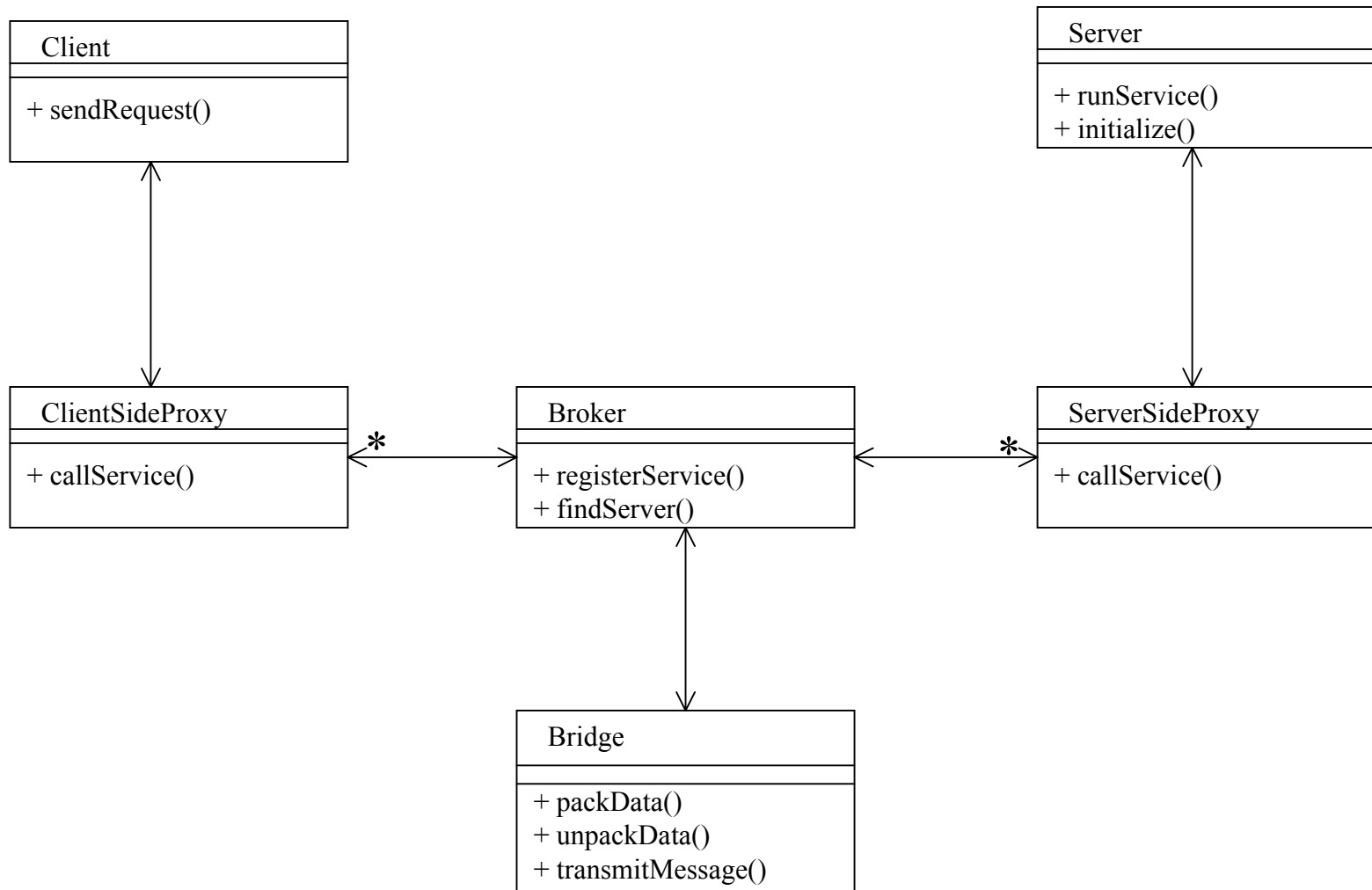
Agenda

✓ Motivation

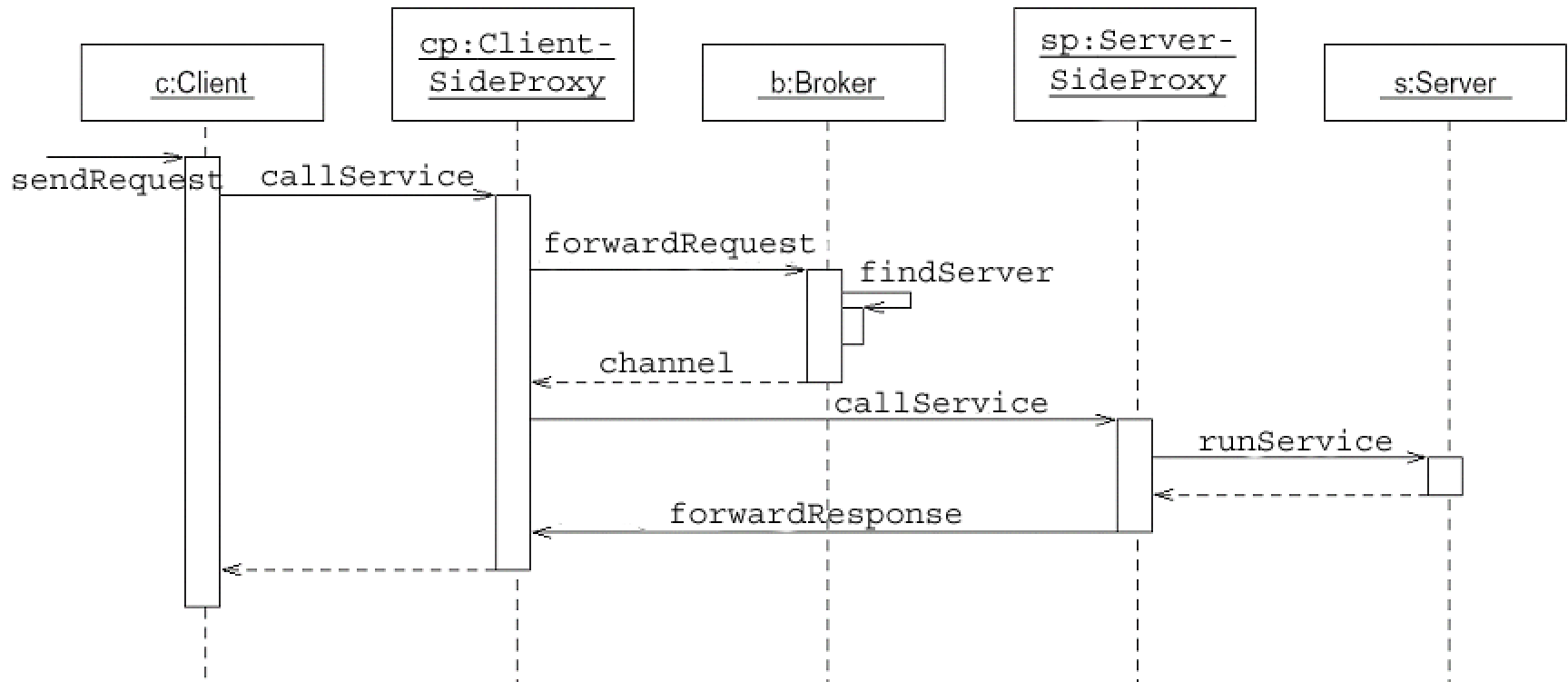
➤ **The Broker Architecture**

- CORBA Architecture
- ORB-Interoperability
- Conclusion
- Book-References

Broker Architecture



Broker Architecture Sequence Diagram



Agenda

- ✓ Motivation
- ✓ The Broker Architecture
- **CORBA Architecture**
 - ORB-Interoperability
 - Conclusion
 - Book-References

ORB – Object Request Broker

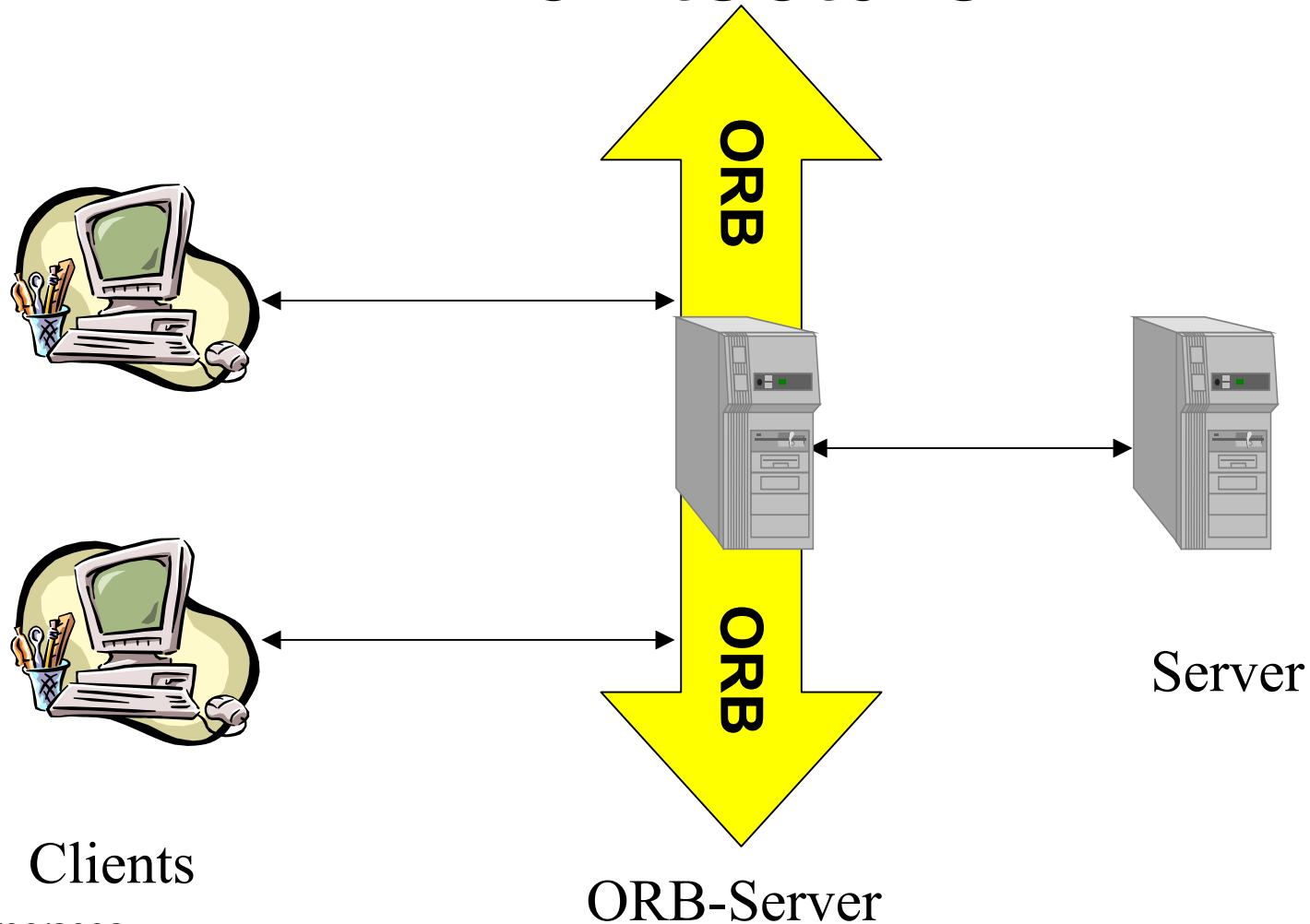
What is an ORB?

- Highway over which all CORBA-communication occurs

The ORB is responsible for:

- Data marshaling
- Object location management
- Delivering request to objects
- Returning output values back to client

Example for a CORBA-Architecture



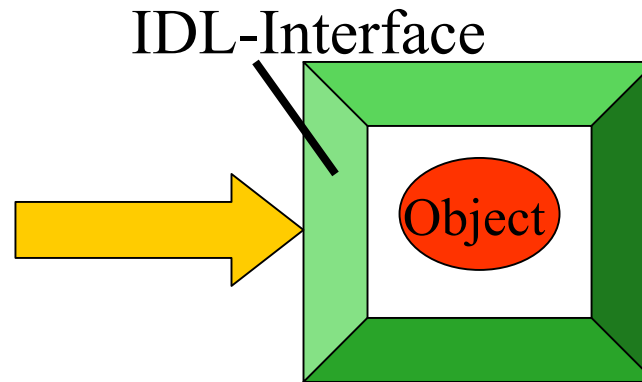
CORBA - Common Object Request Broker Architecture

- CORBA is a standard (not a product!)
- allows objects to transparently make requests and receive responses
- enables interoperability between different applications
 - on different machines
 - in heterogeneously distributed environments

CORBA design goals

- Independence of
 - hardware platform
 - programming language
 - operating system
 - specific Object Request Broker
 - degree of object distribution
- Open Architecture:
 - Language-neutral Interface Definition Language (IDL)
 - Language, platform and location transparent

CORBA works with interfaces

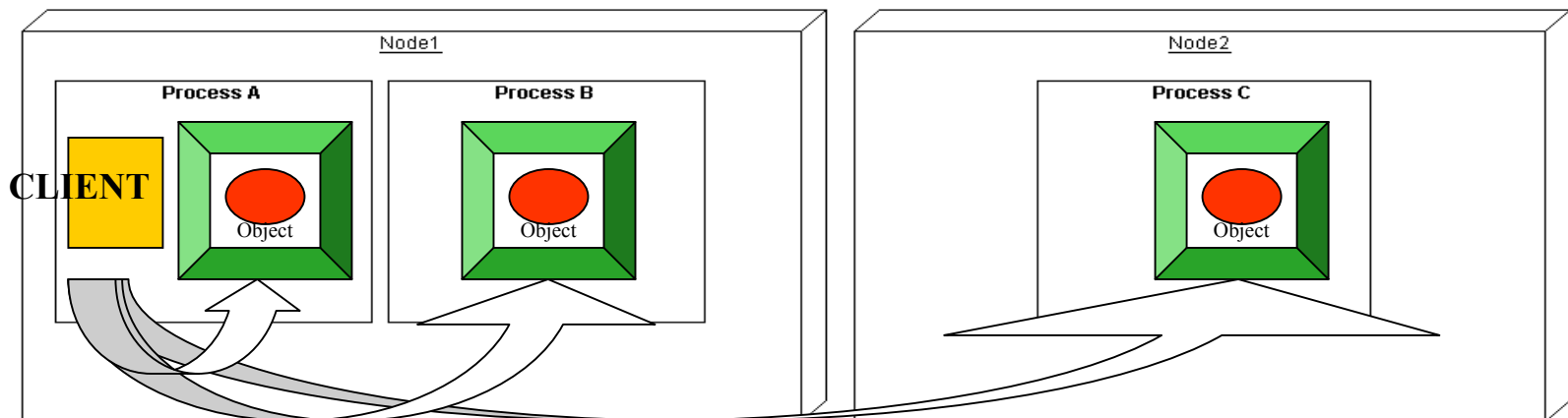


- all CORBA Objects are encapsulated
- Objects are accessible through interface only
- Separation of interfaces and implementation enables multiple implementations for one interface

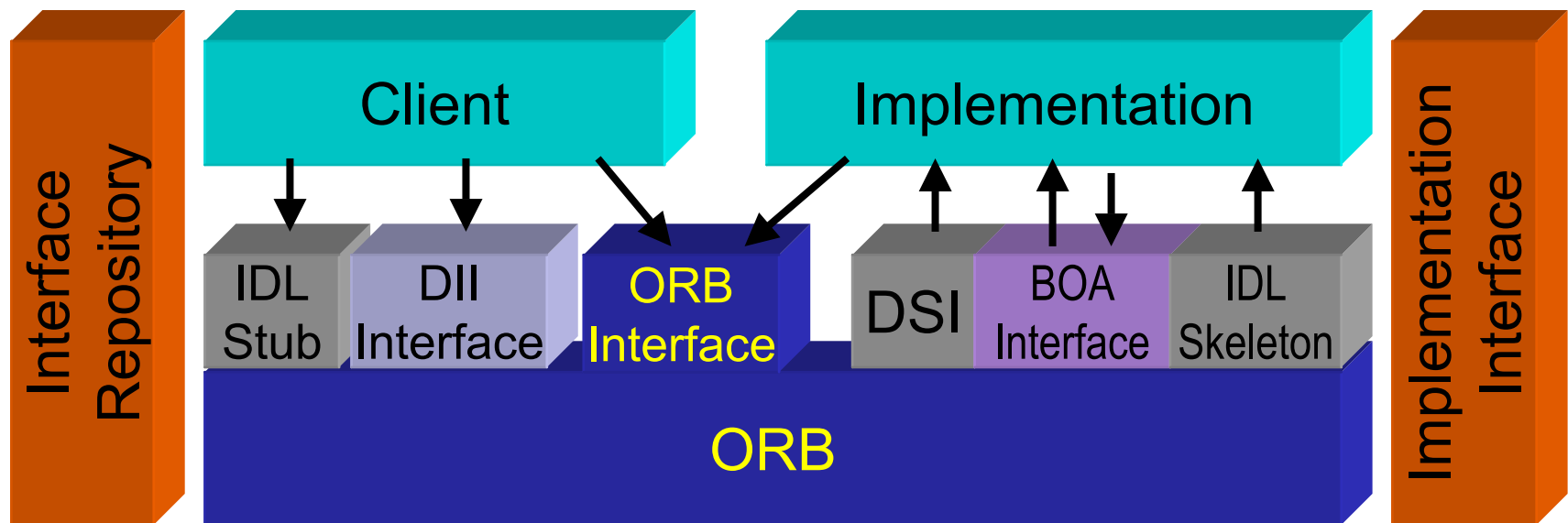
CORBA is *Location Transparent*

For the client it doesn't matter if the object he is operation on is running...

- on the same processor and even in the same process
- on the same processor, in a different process
- in different process on another processor



CORBA Architecture



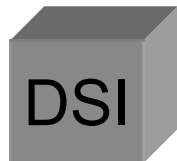
CORBA ORB Interfaces



ORB interface: contains functionality that might be required by clients or servers



Dynamic Invocation Interface(DII): used for dynamically invoking CORBA objects that were not known at implementation time



Dynamic Skeleton Interface(DSI): helps to implement generic CORBA servants



Basic Object Adapter(BOA): API used by the servers to register their object implementations

IFR - Interface Repository

Registry of fully qualified interface definitions



- provides type information necessary to issue requests using the DII

IDL - Interface Definition Language

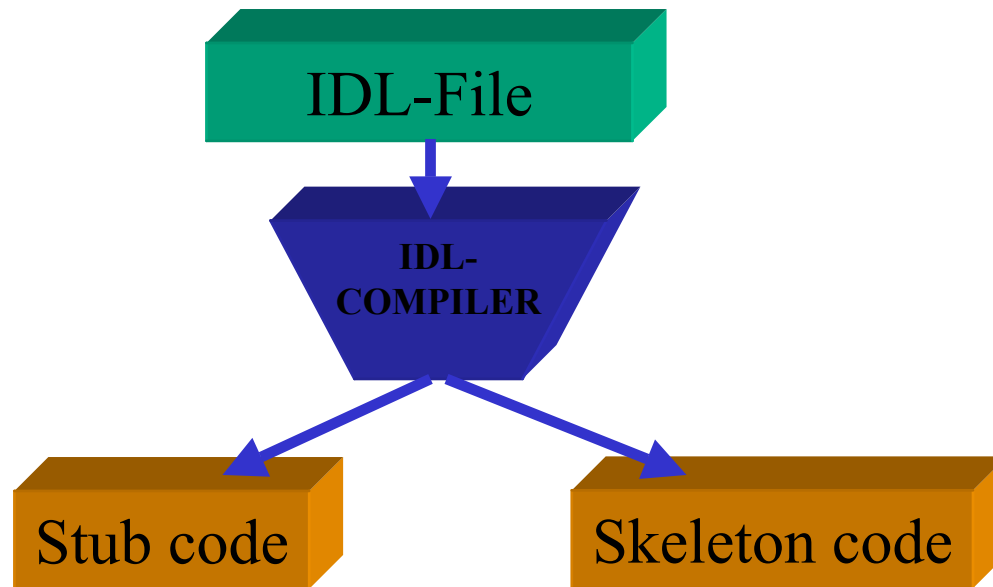
- Specifies the Interfaces
- is programming-language independent and does only contain data descriptions
- “Esperanto” of software

IDL-Compiler

translates the IDL-specifications into

- *IDLstubs* (client-side)
- *IDL-skeletons* (server-side)

in the desired programming language



Steps to develop a CORBA-Application

1. Writing the IDL source

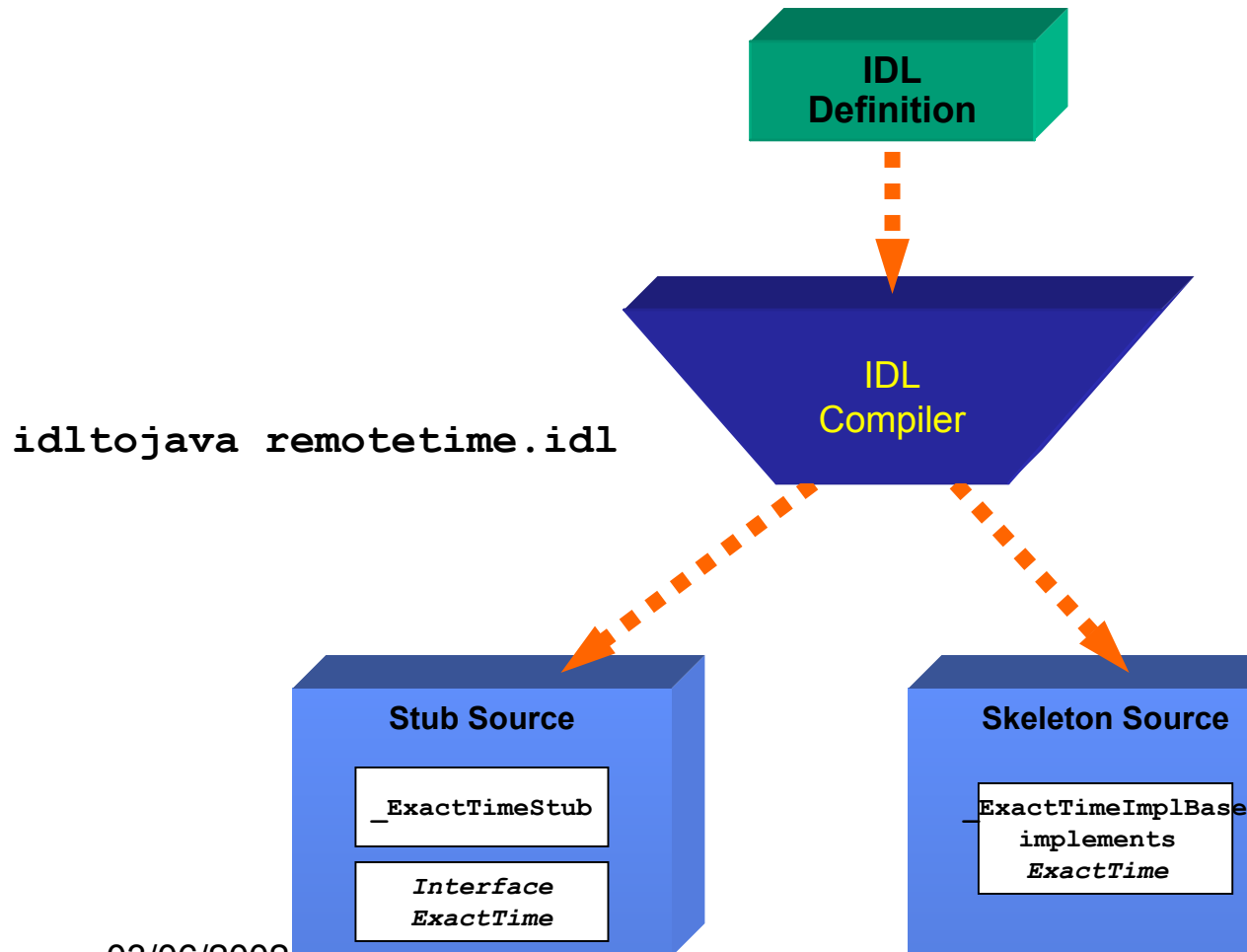


```
module remotetime {  
    interface ExactTime {  
        string getTime();  
    };  
};
```

>>Interface<<
ExactTime

getTime()

2.Creating stubs and skeletons



3.Implementing the server

```
// Server object implementation
class ExactTimeServer extends _ExactTimeImplBase {
    public String getTime(){
        return DateFormat.
            getInstance(DateFormat.FULL) .
            format(new Date(
                System.currentTimeMillis()));
    }
}
```

start up the ORB

create server object

register with ORB

obtain reference to
naming service

assign name to
server object reference

```
// Remote application implementation
public class RemoteTimeServer {
    // Throw exceptions to console:
    public static void main(String[] args)
        throws Exception {
        // ORB creation and initialization:
        ORB orb = ORB.init(args, null);
        // Create the server object and register it:
        ExactTimeServer timeServerObjRef =
            new ExactTimeServer();
        orb.connect(timeServerObjRef);
        // Get the root naming context:
        org.omg.CORBA.Object objRef =
            orb.resolve_initial_references(
                "NameService");
        NamingContext ncRef =
            NamingContextHelper.narrow(objRef);
        // Assign a string name to the
        // object reference (binding):
        NameComponent nc =
            new NameComponent("ExactTime", "");
        NameComponent[] path = { nc };
        ncRef.rebind(path, timeServerObjRef);
        // Wait for client requests:
        java.lang.Object sync =
            new java.lang.Object();
        synchronized(sync) {
            sync.wait();
        }
    }
}
```


4.Implementing the client

start up the ORB
(this time on client-side)

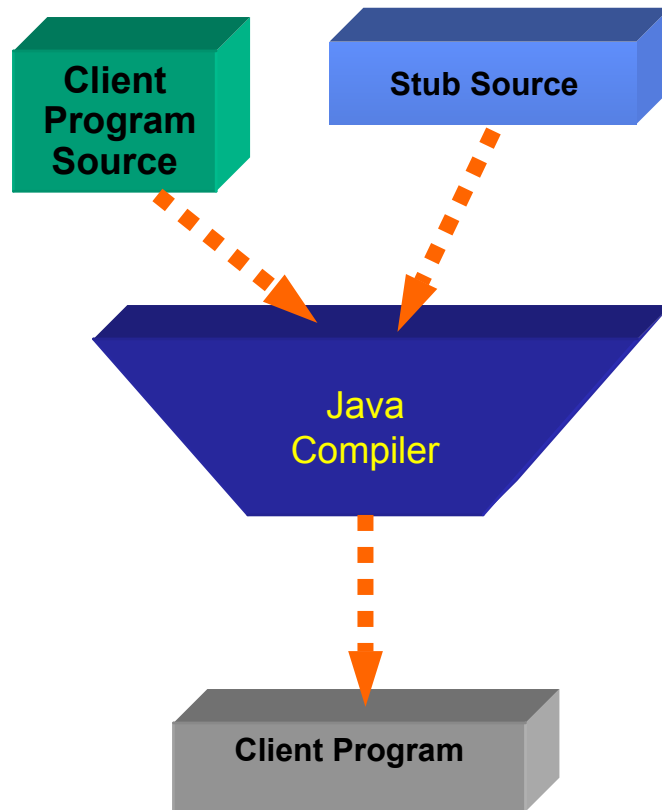
obtain reference to
naming service

create object reference
to servant object

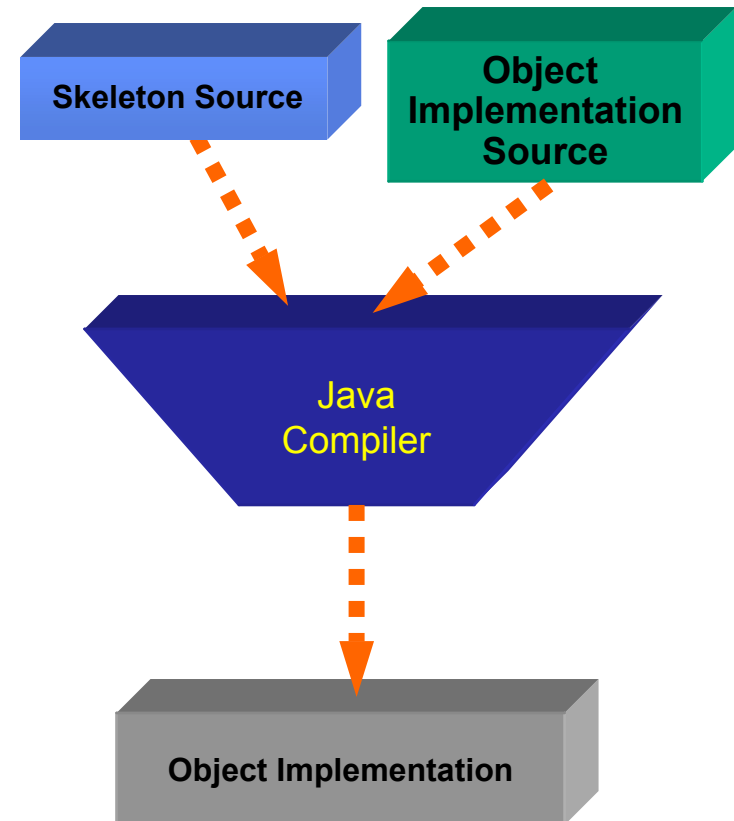
request services...

```
public class RemoteTimeClient {  
    // Throw exceptions to console:  
    public static void main(String[] args)  
        throws Exception {  
        // ORB creation and initialization:  
        ORB orb = ORB.init(args, null);  
        // Get the root naming context:  
        org.omg.CORBA.Object objRef =  
            orb.resolve_initial_references(  
                "NameService");  
        NamingContext ncRef =  
            NamingContextHelper.narrow(objRef);  
        // Get (resolve) the stringified object  
        // reference for the time server:  
        NameComponent nc =  
            new NameComponent("ExactTime", "");  
        NameComponent[] path = { nc };  
        ExactTime timeObjRef =  
            ExactTimeHelper.narrow(  
                ncRef.resolve(path));  
        // Make requests to the server object:  
        String exactTime = timeObjRef.getTime();  
        System.out.println(exactTime);  
    }  
}
```

Client Implementation



Servant Object Implementation



Agenda

- ✓ Motivation
- ✓ The Broker Architecture
- ✓ CORBA Architecture
- **ORB-Interoperability**
 - Conclusion
 - Book-References

ORB Interoperability

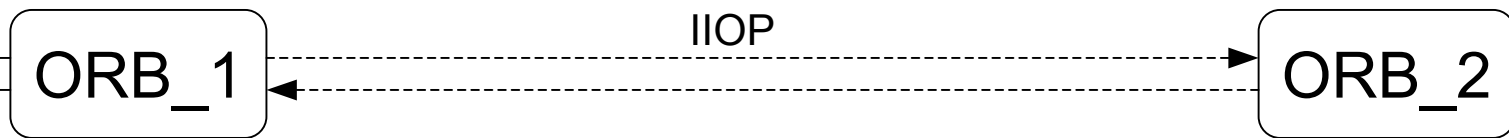
Interoperability: ability of distinct hard- or
software components to interoperate

different forms of interoperability:

- between objects
- between programming languages
- between computers
- **between ORBs**

Possibilities for ORB-interoperability:

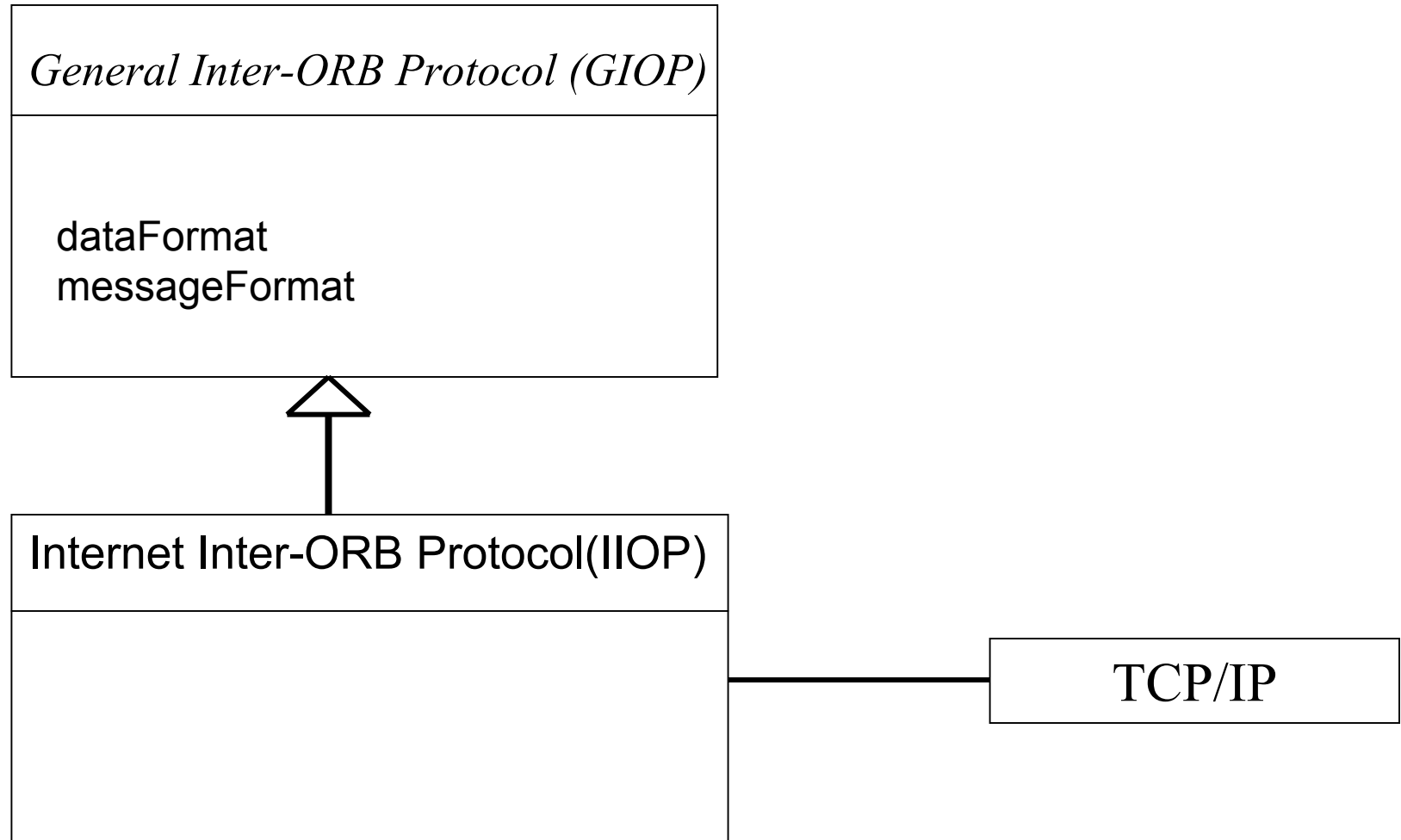
- all ORBs support a common protocol (GIOP)



- every ORB uses it's own protocol,
bridges are used to transfer the protocols



If two ORBs cooperate, they need a common language

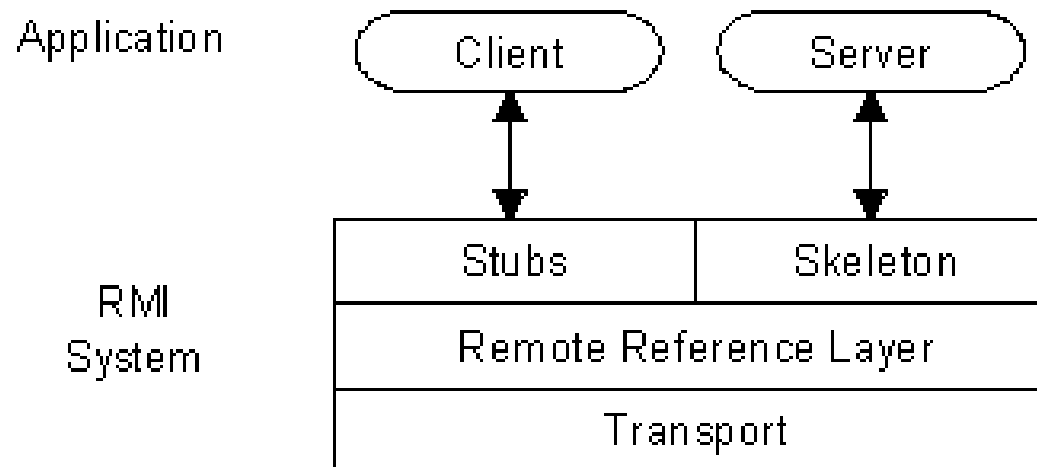


IIOP - Internet Inter-ORB Protocol

- standard communication protocol between ORBs
- describes how agents open TCP/ IP connections and use them to transfer GIOP messages
- Specifies common format for:
 - object references, known as the Interoperable Object Reference (IOR)
 - Messages exchanged between a client and the object

RMI – Remote Method Invocation

Basic architecture



CORBA vs. RMI

- RMI is simpler to work with

- RMI interfaces are defined in Java. RMI-IIOP allows you to write all interfaces in Java



- CORBA interfaces are defined in IDL

- RMI was designed for a single language where all objects are written in Java



- CORBA is language independence

- RMI objects are garbage collected automatically



- CORBA objects are not garbage collected

Agenda

- ✓ Motivation
- ✓ The Broker Architecture
- ✓ CORBA Architecture
- ✓ ORB-Interoperability
- **Conclusion**
 - Book-References

Advantages of CORBA

- CORBA allows methods on a remote object to be accessed as if they were on the local machine
- CORBA is a mature technology, support and tools are widely available
- Can deal with heterogeneous systems
- Legacy-systems can be integrated

Agenda

- ✓ Motivation
- ✓ The Broker Architecture
- ✓ CORBA Architecture
- ✓ ORB-Interoperability
- ✓ Conclusion
- **Book-References**

Book references

- Alan Pope: The CORBA Reference Guide, Addison Wesley, 1997
- CORBA 2.0 – Praktische Einführung für C++ und Java, Addison Wesley, 1996
- Buschmann et al.: Pattern-Oriented Software Architecture (Vol.1), Wiley, 1996
- Bill McCarty, Luke Cassady-Dorion: Java Distributed Objects, SAMS, 1998
- Bruce Eckel: Thinking in Java (2nd edition), Prentice Hall, 2000