

[Lecturer: H. Muccini] <http://www.ics.uci.edu/~muccini/ics123>

ICS 123 (Fall 2002)

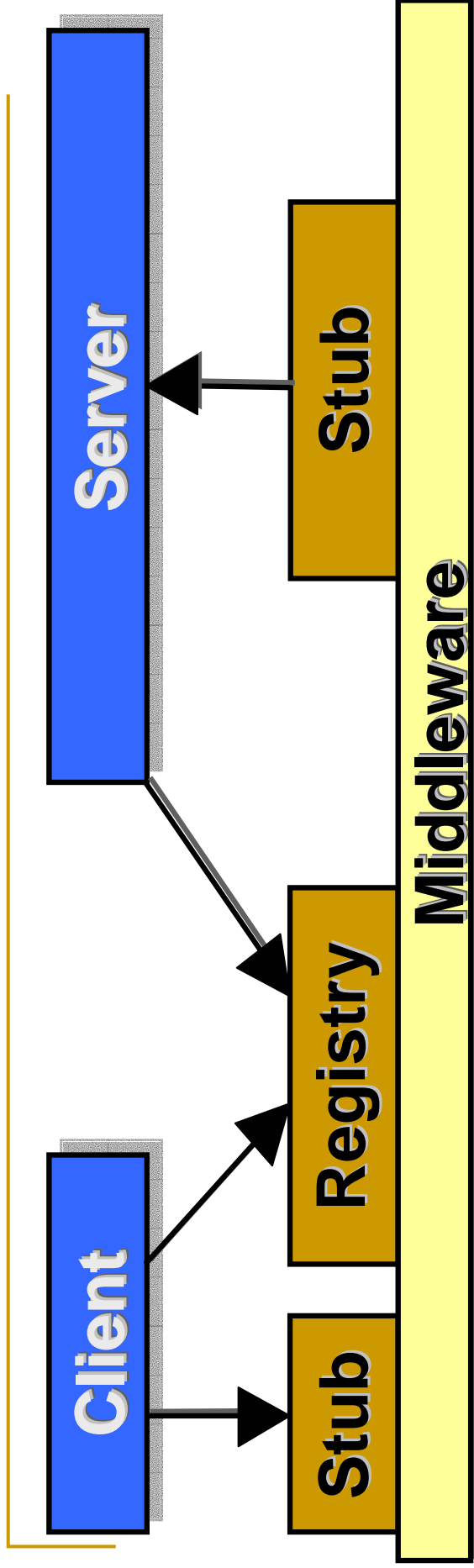
Software Architectures, Distributed Systems, and Interoperability

Lecture 13: CORBA

- CORBA Object Model
- CORBA Architecture
- CORBA vs. COM vs. Java/RMI

Object-Oriented Middleware

- IDL
 - They are more powerful
 - They handle failure
 - They support inheritance
- Presentation Layer
 - Common data representation
 - Marshalling and Unmarshalling
- Session Layer
 - Object activation
 - Object Binding



The Client side:

- Requires the service identifying the object it needs through an object reference
- The *Client Object* is connected to (and depends on) its *Client Stub*
- The Client Stub has the ability to locate the referenced object in the network

The Server side:

- The server object needs to be registered using the *Registry*
- Receives the request from the *Client Stub*
- The *Server Stub* maps the object reference to the active object, checks if the object is active. If it is not, it starts up the object. Then, it forwards the request to the object implementing the service.

CORBA

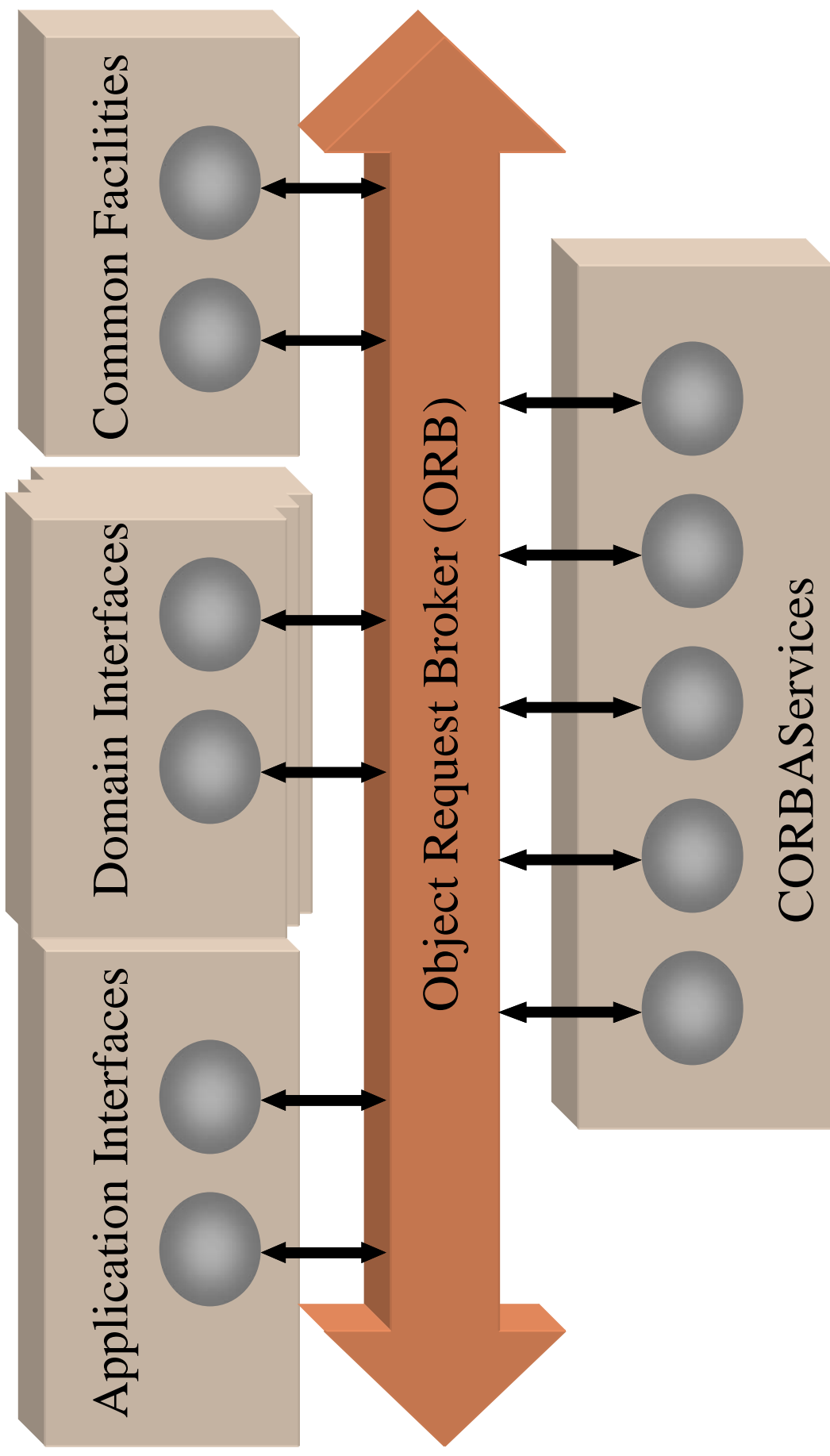
Who is the OMG?

- Non-profit organization
- Founded April 1989
 - Java/RMI: 1997
 - Com/DCOM: 1993-1996
 - CORBA: 1995
- More than 800 members
- Dedicated to creating and popularizing object-oriented industry standards for application integration, e.g.
 - CORBA
 - UML
 - ...

Goals of CORBA

- Support distributed and heterogeneous object request in a way transparent to users and application programmers
- Facilitate the integration of new components with legacy components
- Open standard that can be used free of charge
- Based on wide industry consensus
- Languages supported:
 - C, C++, Smalltalk, Ada-95, Java and OO-Cobol

Object Management Architecture (OMA)



CORBAServices: Object location, object creation and mobility, concurrent access control, distributed transaction controller, ...

Roadmap on this lecture (1/2)

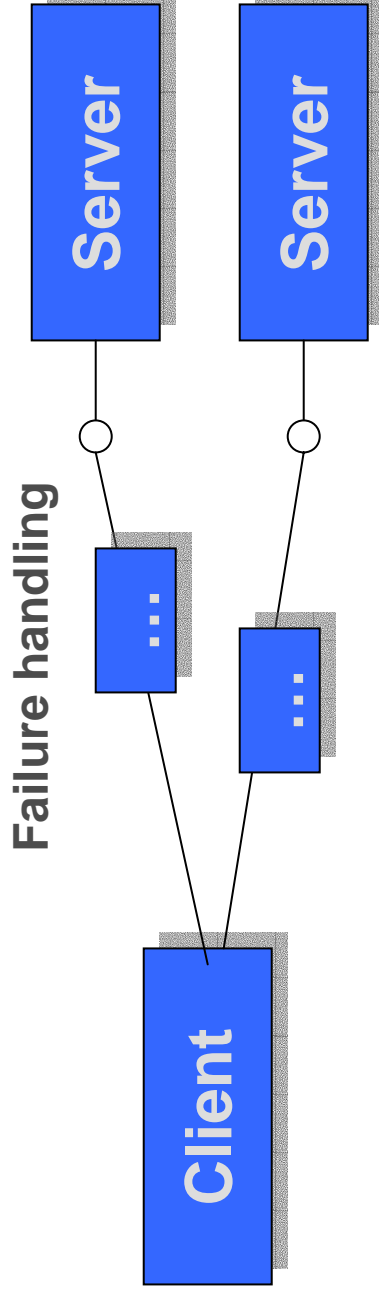
Inheritance



The CORBA Object Model Interface



- How the Client can require the operation it needs
- Parameter passing



Roadmap on this lecture (2/2)

- The CORBA Architecture

- Client Stub
- Server Skeleton
- Object Adapter
- Dynamic Invocation

- CORBA Vs. COM Vs. Java/RMI

Object Model and Interface Definition

- Objects
- Types
- Modules
- Attributes
- Operations
- Requests
- Exceptions
- Subtypes

Objects in CORBA

- Each object has one identifier that is unique within an ORB
 - Different from COM
- In order to make an object request, the client needs to have a reference to a server obj.
- References support location transparency
- Object references are persistent

Types in CORBA (in CORBA IDL)

- The object model is statically typed
 - Type safety is guaranteed
- Atomic types
 - Boolean, char, short, long, float and string
- New types can be built using the *typedef* construct combined with the keywords ***sequence***, ***struct***, ***array*** and ***union***

CORBA Object Model: Types

```
typedef struct _Address {  
    string street;  
    string postcode;  
    string city;  
} Address;  
typedef sequence <Address> AddressList;  
interface Team { ... };
```

The diagram illustrates the classification of CORBA types. The text 'Constructed types' has an arrow pointing to the `typedef struct` definition of `_Address`. The text 'Atomic types' has three arrows pointing to the `string` members: `street`, `postcode`, and `city`.

Module (in CORBA IDL)

A module is used to *restrict the scope*

In this way, we can have different elements (types, interfaces) with the same name

A module can contain type definition (as in the example), interface definitions, constants and exceptions



Two interfaces can have the same name

```
module Soccer {  
  typedef struct _Address {  
    string street;  
    string postcode;  
    string city;  
  } Address; Soccer::Address  
};  
module People {  
  typedef struct _Address {  
    string flat_number;  
    string street;  
    string postcode;  
    string city;  
    string country;  
  } Address; People::Address  
};
```

Attributes (in CORBA IDL)

Syntax: Attribute <static type> <name>

All attributes declared in the interface are accessible to clients
CORBA IDL supports the concept of constants (const)

```
interface Player;  
typedef sequence<Player> PlayerList;  
interface Trainer;  
typedef sequence<Trainer> TrainerList;  
  
interface Team {  
    readonly attribute string name;  
    attribute TrainerList coached_by;  
    attribute Club belongs_to;  
    attribute PlayerList players;  
    ..};
```

Clients cannot
change value

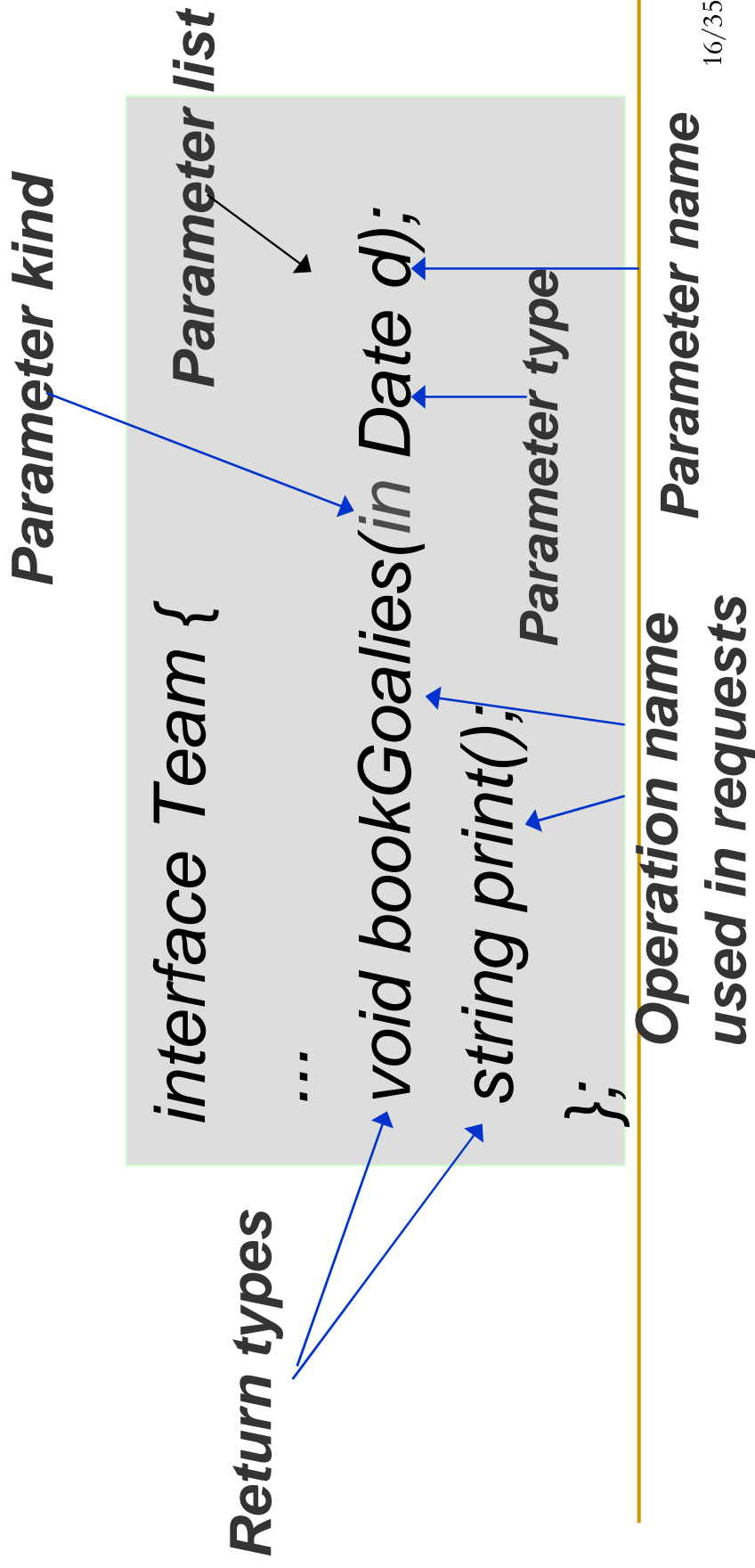
changeable **Attribute type** **Attribute name**

Operations (in CORBA IDL)

Syntax:

```
<return type><operation name>(<parameter>, <>, ...)  
<parameter> = <parameter kind><parameter type><parameter name>
```

CORBA does not support OVERLOADING



Exceptions (in CORBA IDL)

- ≈ 25 system Exceptions (e.g. network down, invalid object reference, out of memory)
- Type-specific Exceptions

Exception name

Exception data

```
exception PlayerBooked{sequence<Date> free;};  
  
interface Team {  
    ...  
    void bookGoalies(in Date d) raises (PlayerBooked);  
};
```

***Operations declare
exceptions they raise***

Subtypes and Inheritance

- CORBA supports multiple inheritance
- Inheritance is public
- The inheritance hierarchy has a single root, that is, the Object interface

Implicit supertype: Object

Operation Inherited by Club

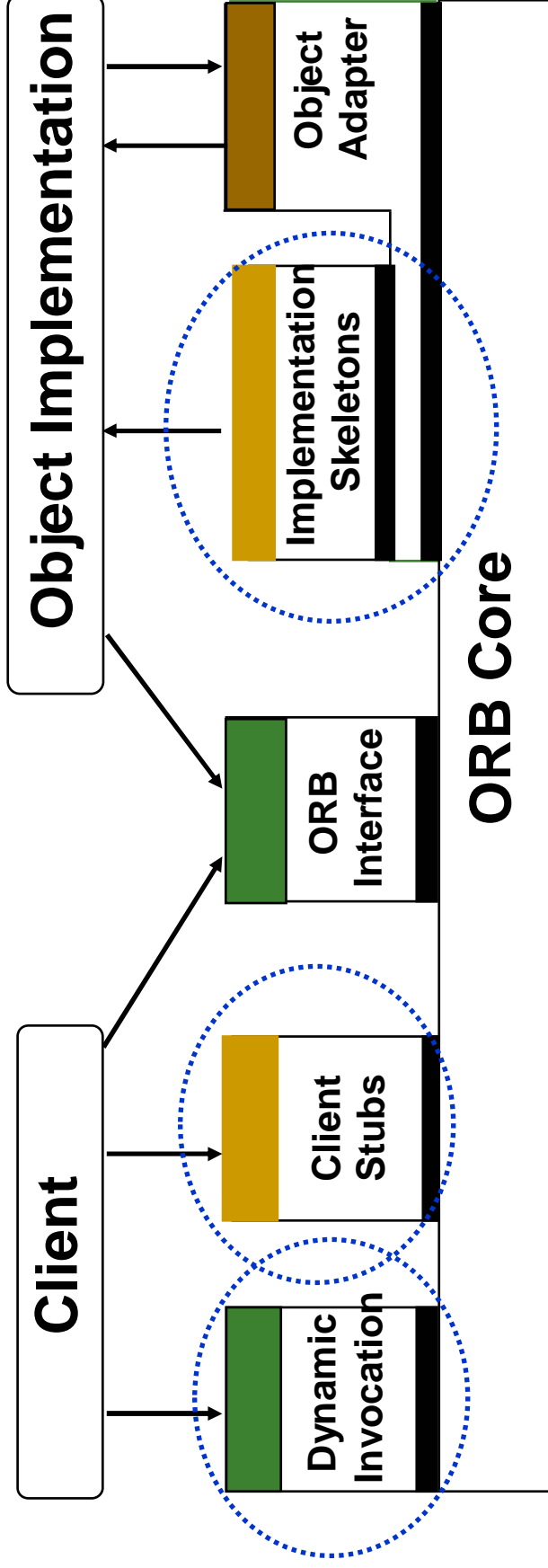
```
interface Organization {  
    readonly attribute string name;  
};  
interface Club : Organization {  
    exception NotInClub{};  
    readonly attribute short noOfMembers;  
    readonly attribute Address location;  
    attribute TeamList teams;  
    attribute TrainerList trainers;  
    void transfer(in Player p) raises(NotInClub);  
};
```

Summarizing: IDL

- **Modules**
- **Interfaces**
 - **Attributes**
 - **Types**
 - **Constants**
 - **Operations**
 - **Exceptions**

```
module Bank {  
    interface Account {  
        float balance();  
    };  
    interface AccountManager {  
        exception CantOpen {};  
        Account open(in string name)  
            raises (CantOpen);  
    };  
};
```

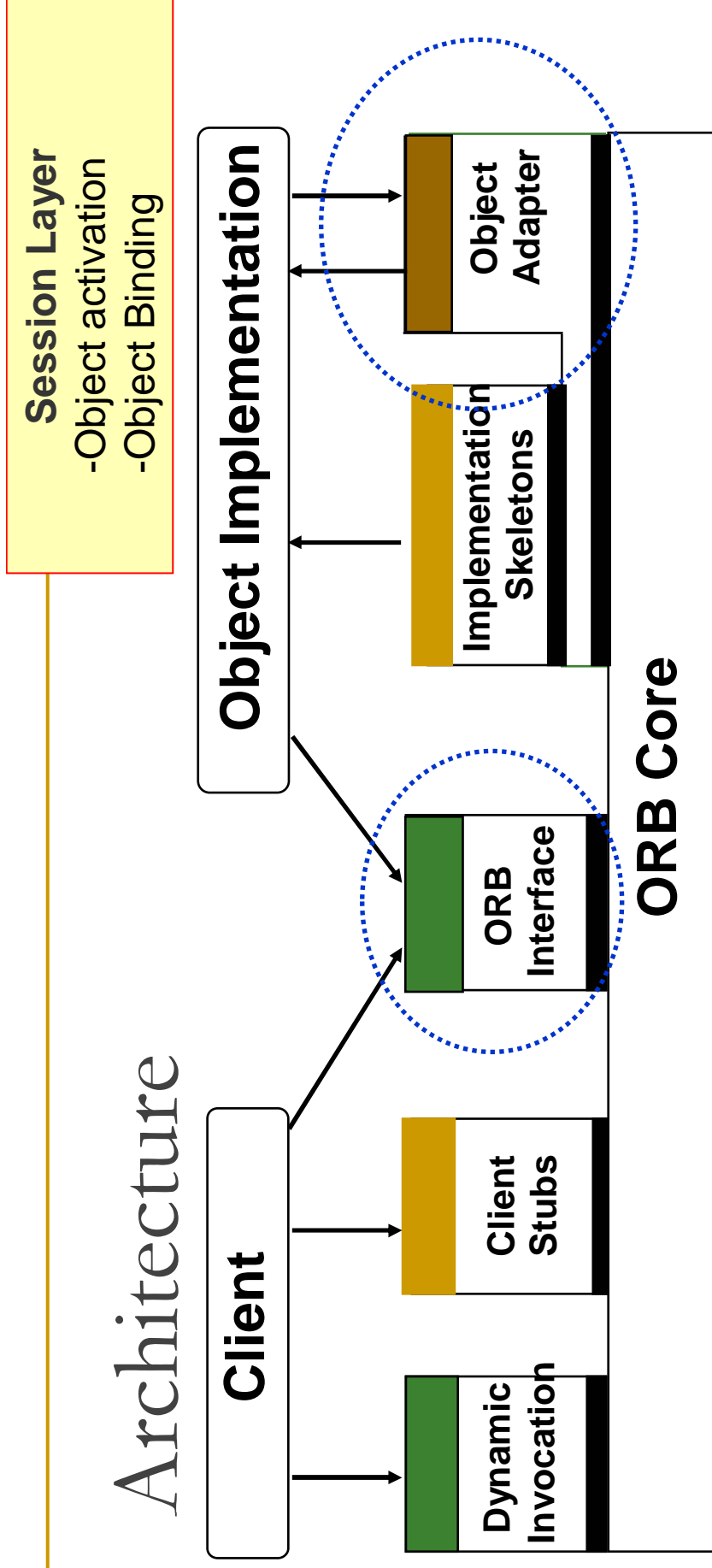
Architecture



- *Client Stub*, *implementation skeleton* and *Dynamic Invocation* interface are responsible for marshalling and unmarshalling
- Stub and skeleton are generated from the IDL using an IDL compiler

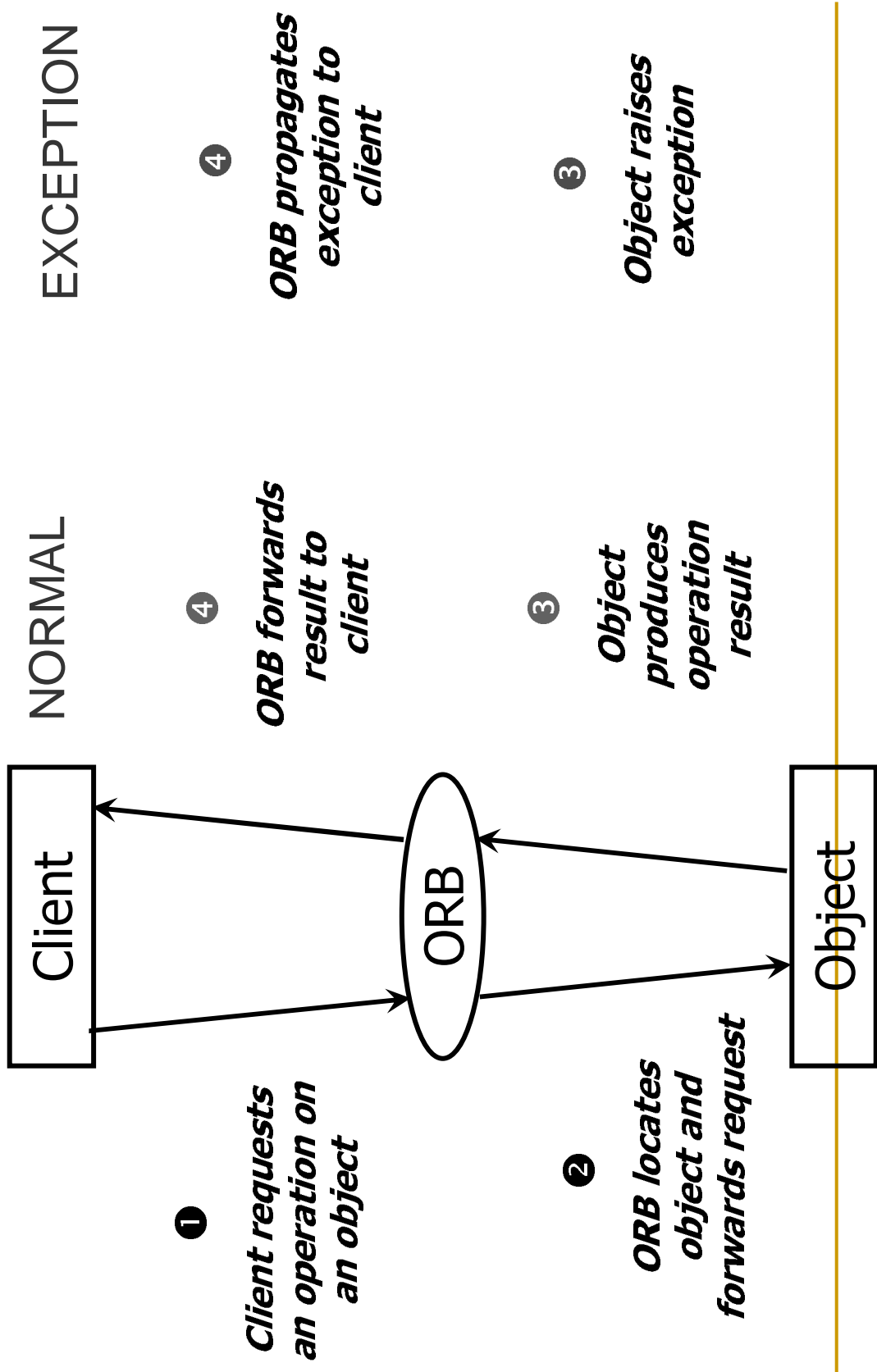
Presentation Layer
-Common data representation
-Marshalling and Unmarshalling

Architecture

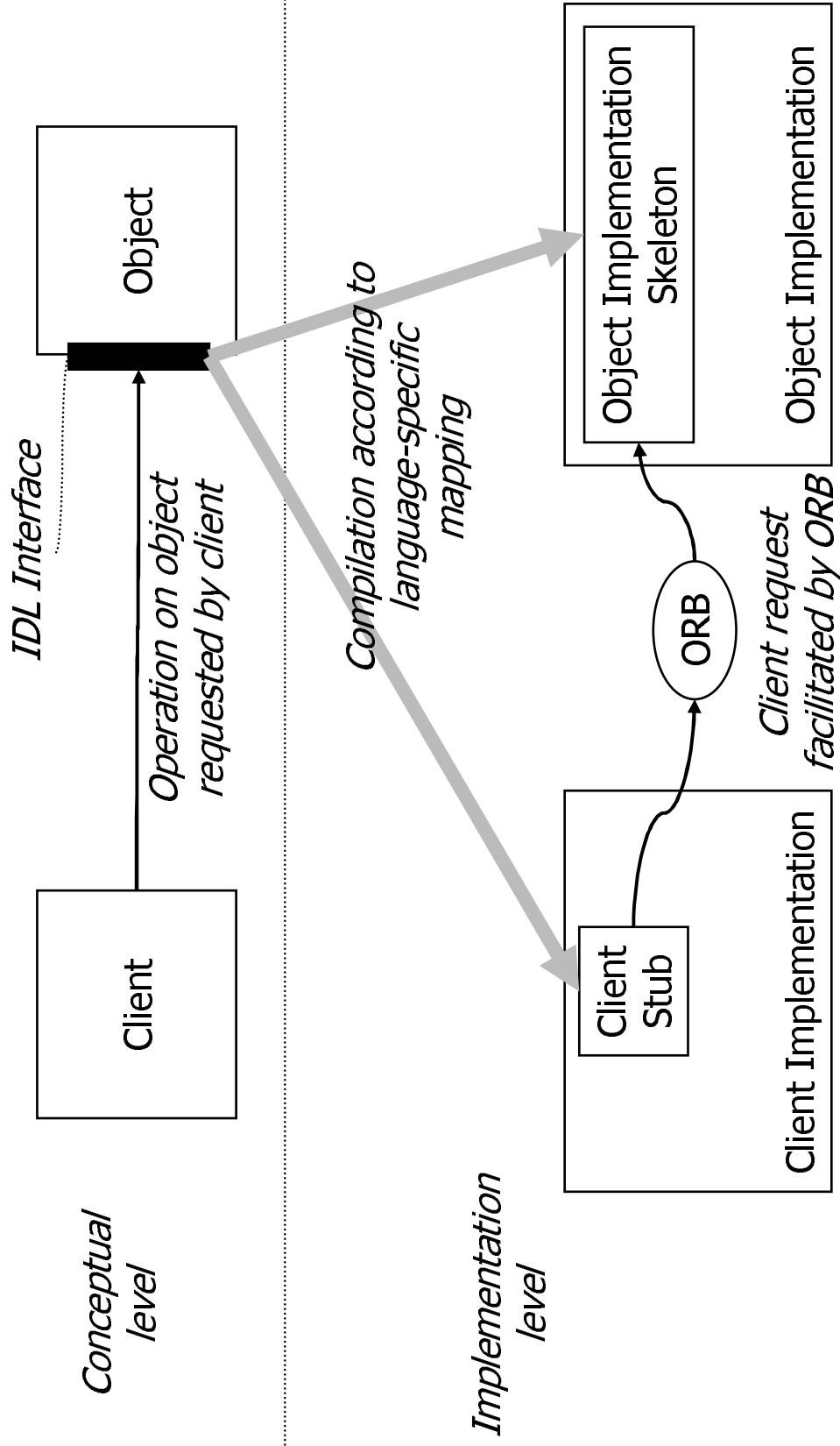


- The Object Adapter handles:
 - The server obj. registration, activation and deactivation, references
 - There are many Object Adapter:
 - Basic Object Adapter (BOA)...
 - Replaced by the Portable Object Adapter (POA)... it also supports persistent objects
- The object interface is used to initialize clients and servers

CORBA: Normal and Exception Behavior



The CORBA Object Model



CORBA vs. COM vs. Java/RMI

Comparison Roadmap

Object Model: **Architecture:**

- **Objects** ■ **Stub**
- **Types** ■ **Skeleton**
- **Operations** ■ **Proxy**
- **Requests** ■ **SCM**
- **Exceptions** ■ **...**

CORBA and COM: The Object Model

Differences:

- **INTERFACES:** The inheritance root in CORBA is the interface “Object” . COM has “IUnknown” as root interface.
- **TYPES:** CORBA supports more type’s construction mechanisms:
 - COM: records and arrays
 - CORBA: sequence, struct, array and union
- **REQUESTS:** COM operations return a 32-bit integer, while CORBA can return different types. COM requests can be local to the same process, local to the same machine, remote (using RPC)
- **IDL:** CORBA uses CORBA IDL while COM uses MIDL
- **INHERITANCE:** COM does not support multiple inheritance (p. 105), while CORBA does
 - Multiple interfaces in COM provide similar expressiveness as multiple inheritance in CORBA
- **FAILURE HANDLING:** COM uses the HRESULT. CORBA uses system and type-specific exceptions. COM failure handling is less powerful than CORBA’s (p. 104)
- **ID:** COM uses unique UUID and CLSID

CORBA and COM: The Object Model

Similarities:

- **OPERATION VISIBILITY:** Like COM, CORBA does not have any primitives to define the visibility of operations
- **OPERATIONS TYPE:** CORBA and COM operations type are very similar, since both present input, output and input/output parameters

Notes:

- **RPC:** Microsoft did not invent the distribution mechanisms by scratch but extended OSF/DCE's RPC
- COM is for reuse and evolvability, using binary encapsulation and binary compatibility
- **STUBS:** COM proxies are equivalent to CORBA client stub
- **MODULE:** COM has not the concept of "module" owned by CORBA:
 - CORBA modules are for scoping purpose
 - COM does not need, since it uses UUIDs and CLSIDs
- Corba ORB is not portable to other ORB products

Feature	COM/DCOM	CORBA/IIOP
Security	Shipping : NTLM NT5: MIT Kerberos, SSL/Public Key	Shipping: none Beta: Multiple SSL variations
Languages	Visual C++, Visual J++, Visual Basic, Cobol	C, C++, ADA-95 JAVA, Smalltalk, OO-Cobol No Visual Basic
Multiple Transports	TCP IP, IP, IPX, SPX, HTTP, many others	TCP only

Java RMI and CORBA - COM

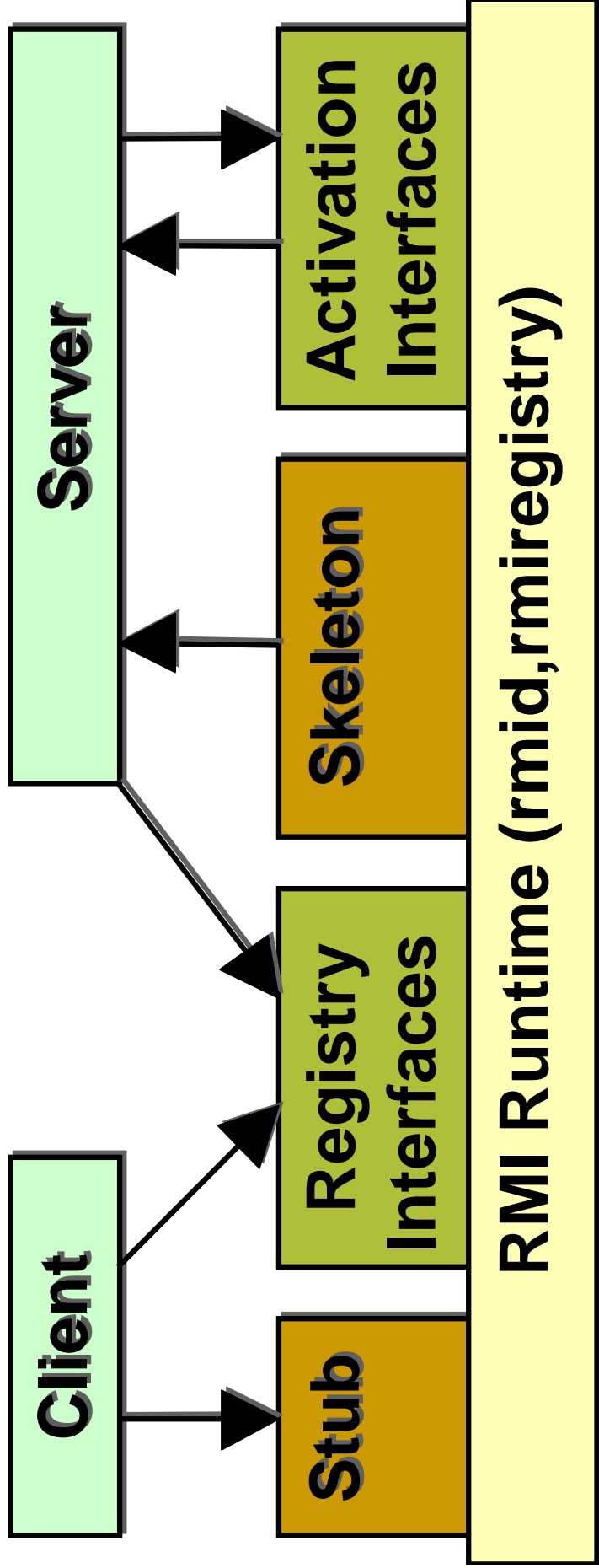
- **RMI:**
 - Requires objects to be programmed in Java
 - Objects are platform independent
 - The IDL is in Java
 - extend the **java.rmi.Remote** class
 - any method that can be remotely invoked in Java/RMI may throw a **java.rmi.RemoteException**
 - Java needs to call a Security Manager
 - Many object services are being defined within Enterprise JavaBeans
 - CORBA provided these services since the OMA architecture
 - COM provides services through COM+
 - It can be used in any hw platform with a JVM
 - CORBA may be run on different O.S.
 - COM mainly under Windows

Java RMI and CORBA, COM

- Object Reference
 - CORBA identifies objects through object references
 - COM objects are identified by interface pointers
 - Java/RMI objects are identified by references
- Inheritance root
 - CORBA → *Object*
 - COM → *IUnknown*
 - Java/RMI → *Remote*
- Inheritance and Failure Handling
 - Both CORBA and Java/RMI support multiple inheritance at the IDL or interface level
 - CORBA and Java/RMI IDLs can specify exceptions in the IDLs while DCOM does not

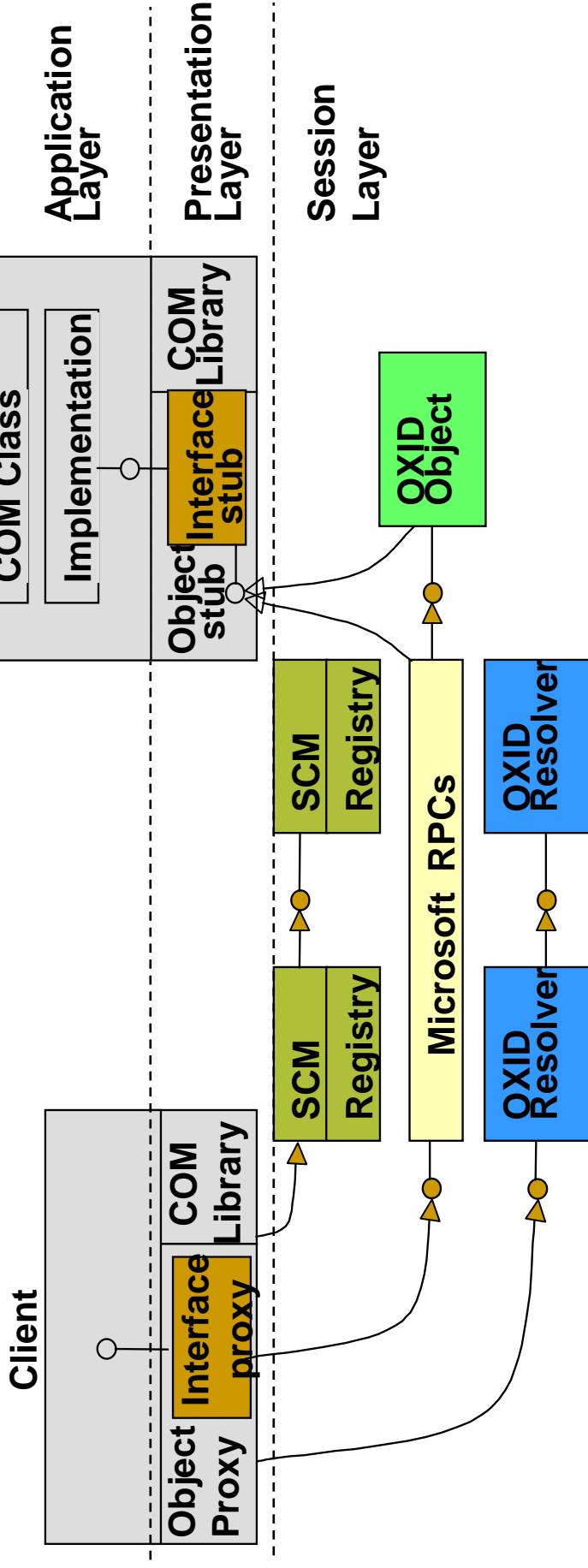
DCOM - IDL	CORBA - IDL	Java/RMI - Interface definition
<pre> ... library SimpleStocks { importlib("stdole32.tlb "); [uuid(BC4C0AB0-5A45- 11d2-99C5- 00A02414C655), dual] interface IStockMarket : IDispatch { HRESULT get_price ([in] BSTR p1, [out, retval] float * rtn); } ... </pre>	<pre> module SimpleStocks { interface StockMarket { exception Remote(); float get_price(in string symbol) raises Remote(); }; }; </pre>	<pre> package SimpleStocks; import java.rmi.*; import java.util.*; public interface StockMarket extends java.rmi.Remote { float get_price(String symbol) throws RemoteException; } </pre>
File : StockMarketLib.idl	File : StockMarket.idl	File : StockMarket.java

The RMI Architecture



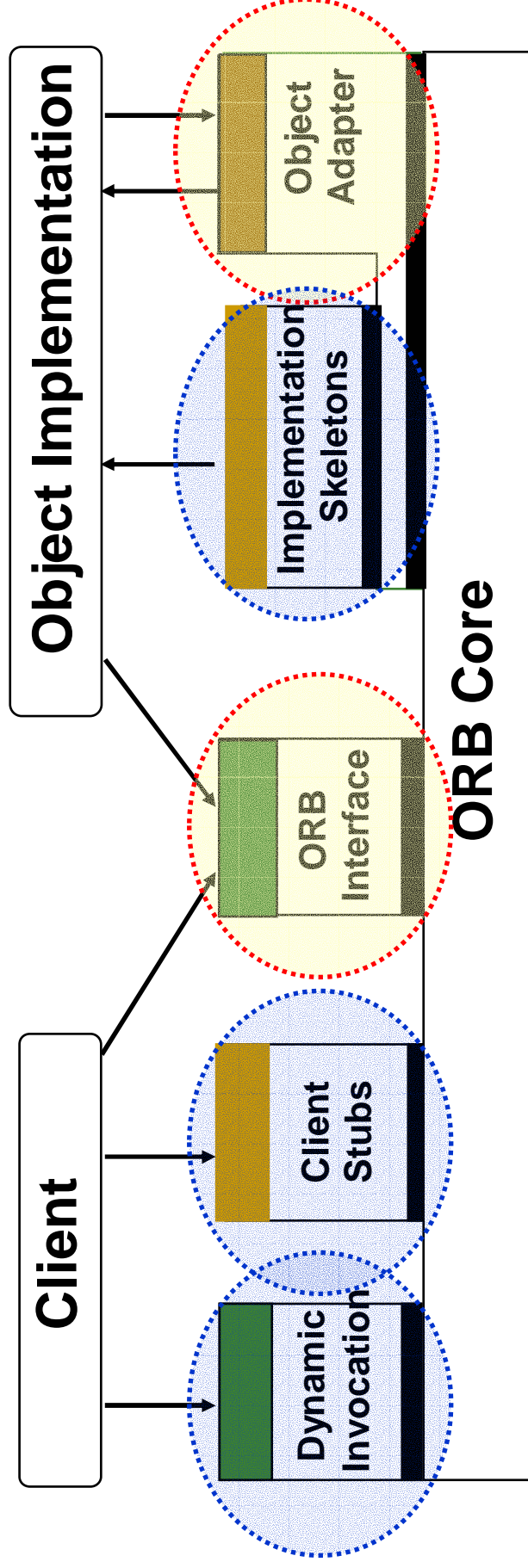
- Stub and Skeleton are responsible for marshalling and unmarshalling. They are automatically generated using the *rmic* command
- The Registry is used to register and access the Server object (rebind, lookup)
- The Activation Interfaces are used to activate components on demand

The COM Architecture



- The “*Presentation*” layer is responsible for marshalling and unmarshalling and create or locate an interface pointer to clients
- The *Service Control Manager (SCM)* is responsible for activation
- The *OXID* is the protocol representing the **how to** communication information

The CORBA Architecture



- Marshalling and unmarshalling
- Activation and Deactivation

To conclude

- CORBA, COM and RMI
 - CORBA: for mission-critical and high-availability applications. Unix and mainframes
 - COM: distributed systems for Windows operating systems
 - Java/RMI: Internet and e-commerce, portable applications