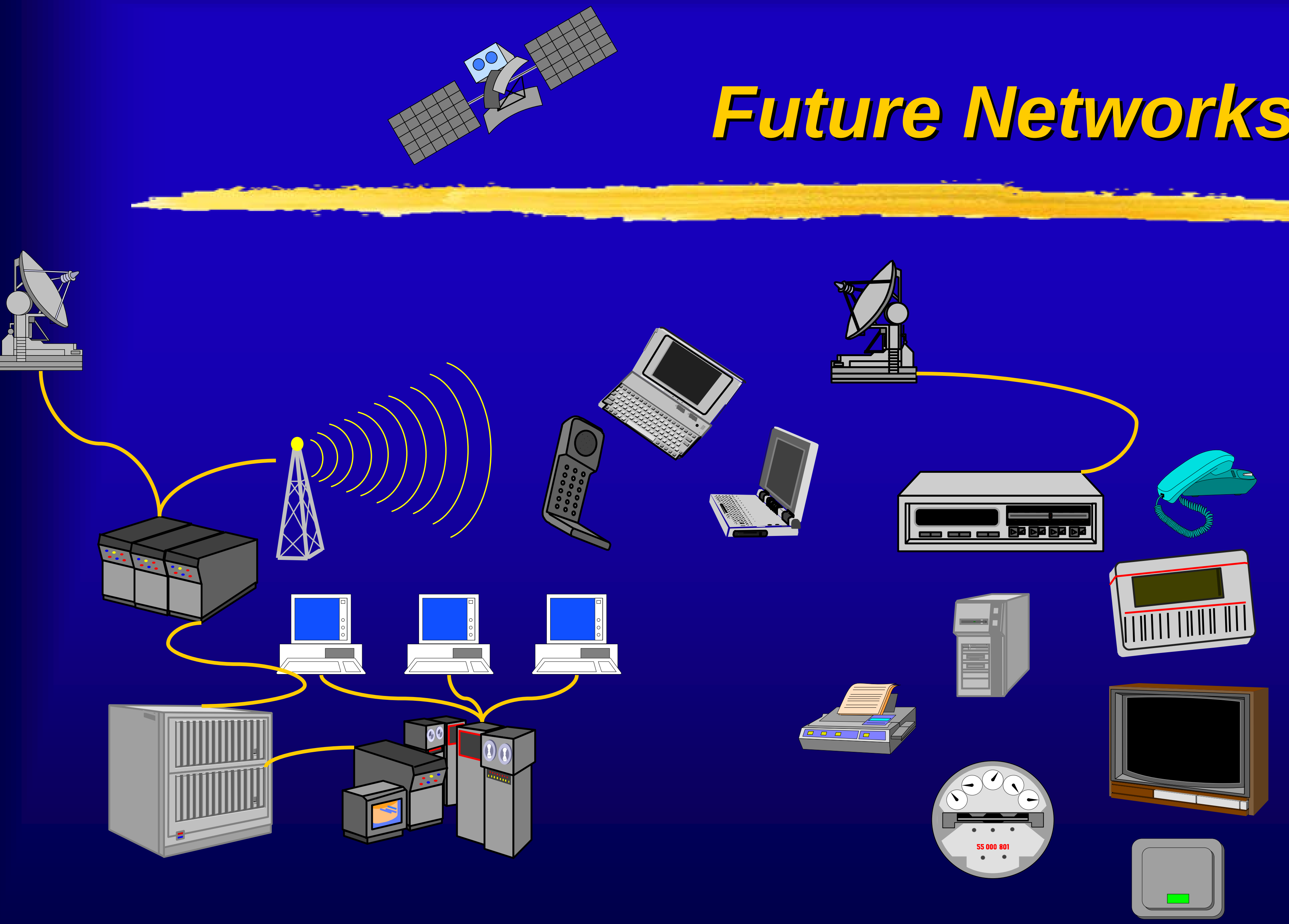
Distributed Applications with CORBA

Frank Kargl

Chaos Computer Club, Ulm, Germany

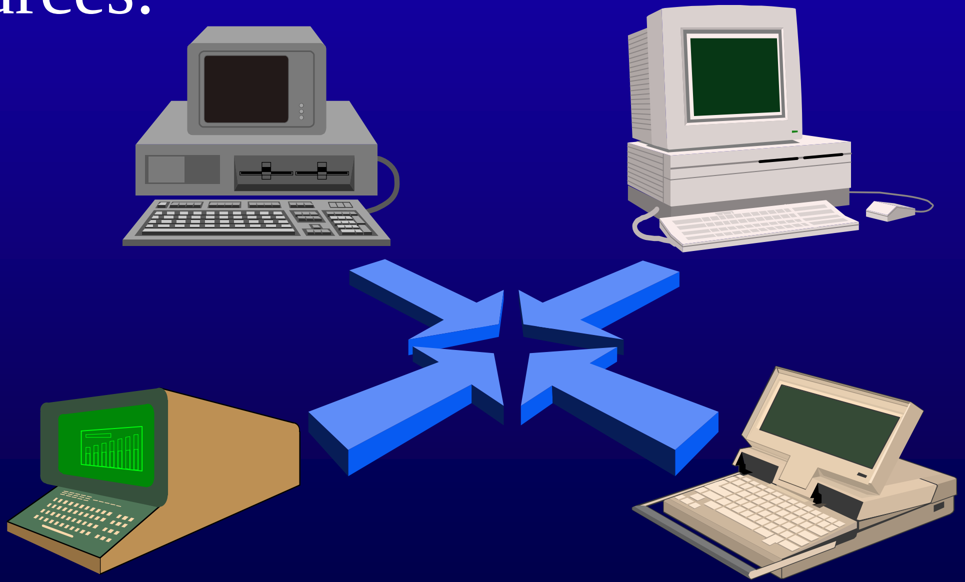
frank.kargl@ulm.ccc.de

Future Networks



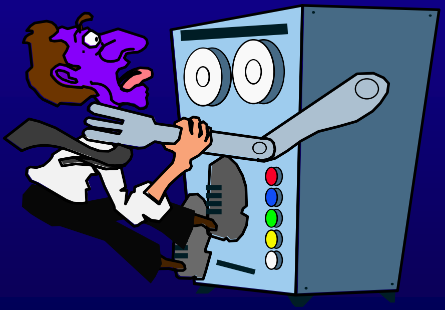
The Problem

- Application Integration and Distributed Processing are the same thing
- Constructing information-sharing distributed systems from diverse sources:
  - heterogeneous,
  - networked,
  - physically disparate,
  - multi-vendor.



Existing Tools?

- A major problem stands in the way:
  - Existing tools (Sockets, DCE, ONC) are too low-level; don't offer a unified view of all distributed applications.
  - Complexity of Distributed Systems grows beyond any boundaries.
  - Implementation and Management Overkill

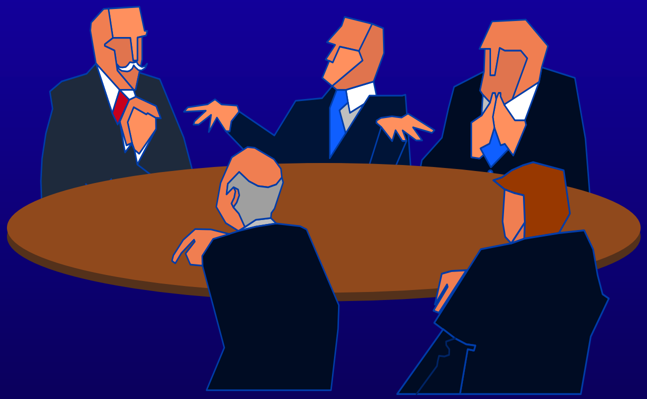


Object Management Group

- Not-for-profit company based in United States, with representation in United Kingdom, Japan & Germany.
- Founded April 1989.
- Small staff (15 full time); no internal development.
- Dedicated to creating and popularizing object-oriented standards for application integration based on existing technology.
- Object World subsidiary for market studies, training, seminars and conferences.

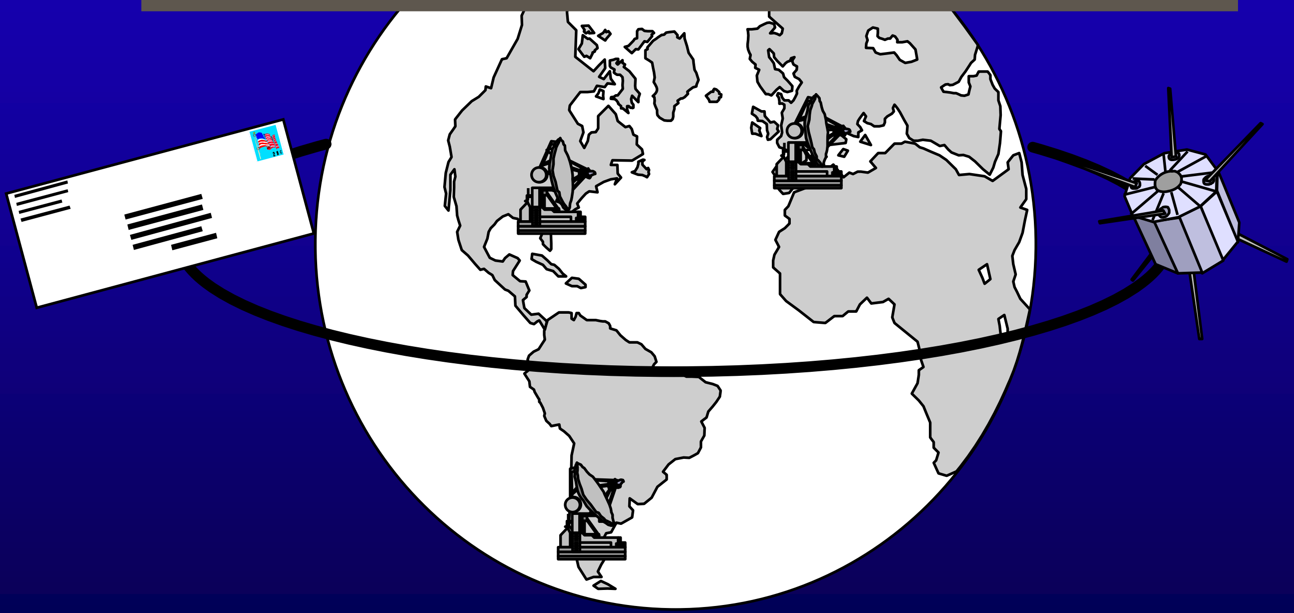
Technical Committee

- Representatives of all member companies.
- Determines direction of architecture & standards.
- Meets every eight weeks, hosted internationally.



Electronic Meetings

Discussions continue between meetings by electronic mail



Adoption Process

- RFI (Request for Information) to establish range of commercially available software.
- RFP (Request for Proposals) to gather explicit descriptions of available software.
- Letters of Intent to establish corporate direction.
- Task Force and End User evaluation & recommendation; simultaneous Business Committee examination.
- Board decision based on TC, End User, and BC recommendations.

Domain Task Forces

- Manufacturing:
  - Engineering Resource Planning (ERP), Product Data Management (PDM);
- Finance
  - Currency Standard, Party Management,Compass Project for Accounting
- Telecommunications
  - TMN/CORBA interworking, CORBA for IN, Audio/Video Streams
- Electronic Commerce
  - Electronic Payments, Negotiations
- Healthcare (CORBAmed)
  - Patient Identification Number (PID), Lexicon Query System, Life Sciences,…
- Transportation
  - air Traffic control,
  - inter-nodal transport.

Other Directions

- End-user SIG.
- Object-oriented analysis & design SIG
- Object-oriented database interface standards SIG
- Business Object Management SIG
- Manufacturing SIG
- Healthcare SIG
- Telecommunications SIG
- Financial SIG
- Security SIG

An Open Process

- OMG is an open, member-supported process, and share their work with other organizations doing related work:
  - X/Open
  - OSF, X Consortium, W3C
  - ESPRIT, NIST, NII
  - ISO, ITU
  - National bodies: ANSI, IEEE, JIPS
  - CFI, ODMG, COS, NMF, IMA, POSC

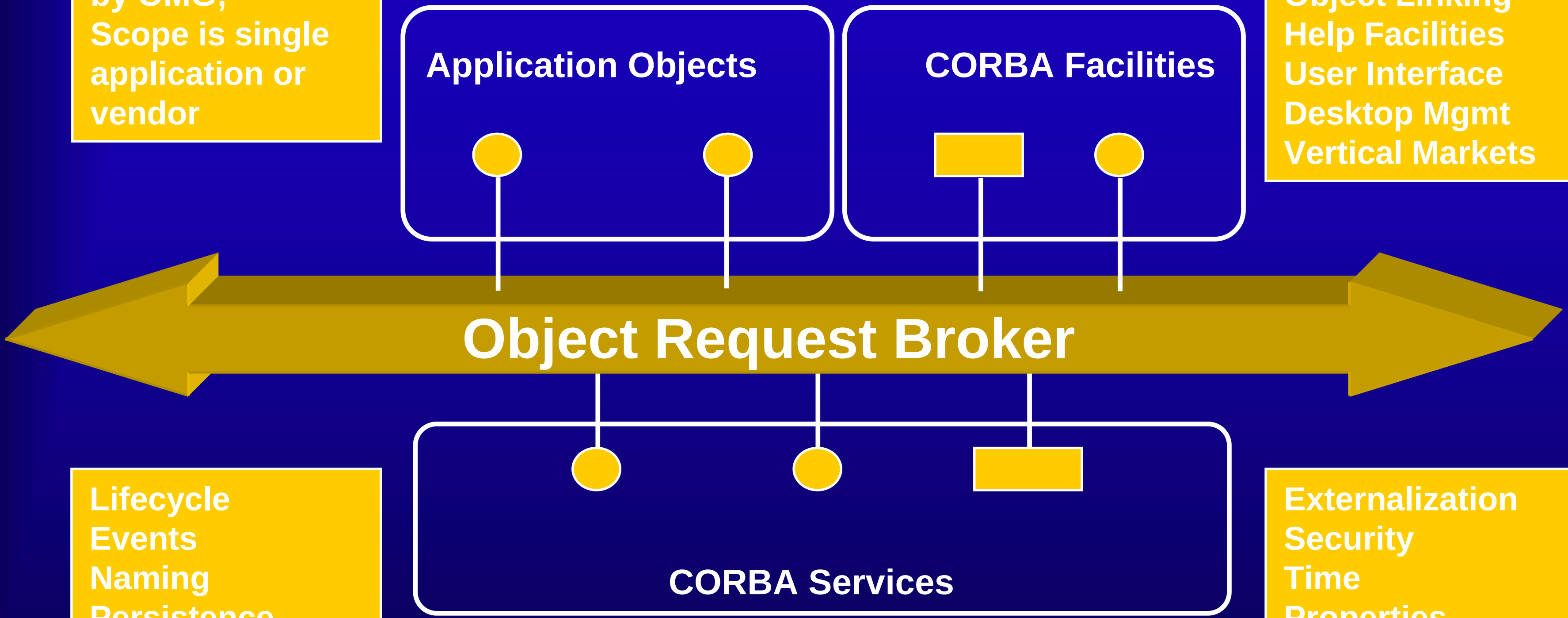
A Common Foundation

- Enable interoperability and Portability based on an object-oriented foundation which specifies:
  - A single terminology for object-orientation.
  - A common abstract framework or object model.
  - A common reference model or architecture.
  - Common interfaces & protocols.
- Foundation: Model published in OMA Guide

Object Model

- Stated Goal is "to define an object model that facilitates Portability of applications and type libraries, and Interoperability of software components in a distributed environment."
  - The model was completed in 1992 and published in the updated OMA Guide.
  - Key concepts are core, components, and profiles.

OMA Overview



Not standardized by OMG;
Scope is single application or vendor

Application Objects

CORBA Facilities

Compound Docs
Object Linking
Help Facilities
User Interface
Desktop Mgmt
Vertical Markets

Object Request Broker

CORBA Services

Lifecycle
Events
Naming
Persistence
Transactions
Concurrency

Externalization
Security
Time
Properties
Query
Licensing

Fundamental CORBA Design Principles

- Separation of interface and implementation
  - Clients depend on interfaces, not implementation
- Location transparency
  - Service use is orthogonal to service location
- Access transparency
  - Invoke operations on objects
- Typed interfaces
  - Object references are typed by interfaces
- Support of multiple inheritance of interfaces
  - Inheritance extends or specializes behavior

Fundamental CORBA Design Principles

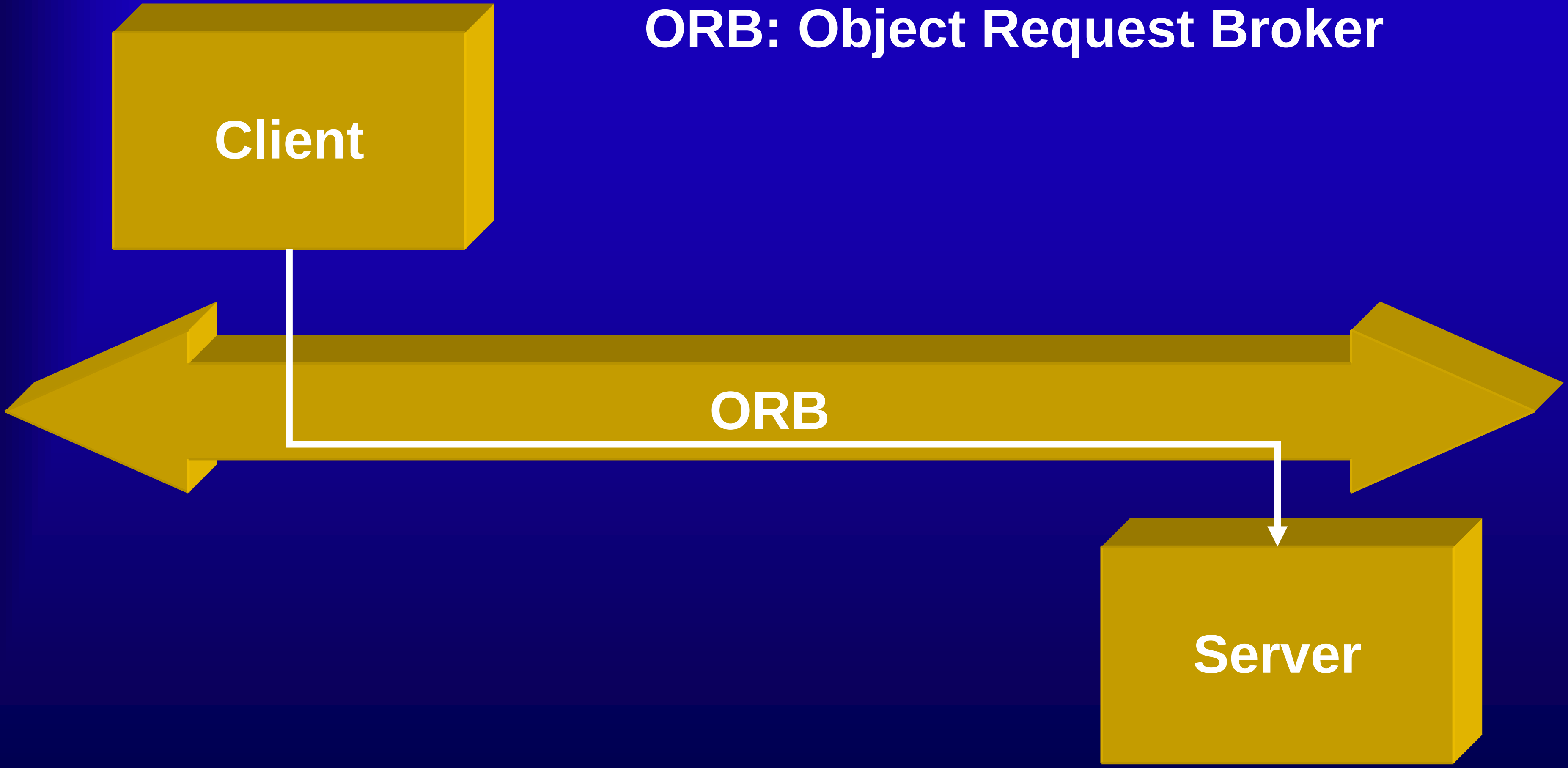
- CORBA supports reliable, uni-cast communication
  - oneway, twoway, deferred synchronous
- CORBA objects can also collaborate in a client/server, peer-to-peer or publish/subscriber manner
  - e.g. COS Event Service defines a publish/subscribe communication paradigm

CORBA Advantages

- Simplifies application interworking
  - higher level integration than traditional untyped bytestreams
- Benefits for distributed programming similar to OO languages for non-distributed programming
  - encapsulation, interface inheritance, polymorphism and exception handling

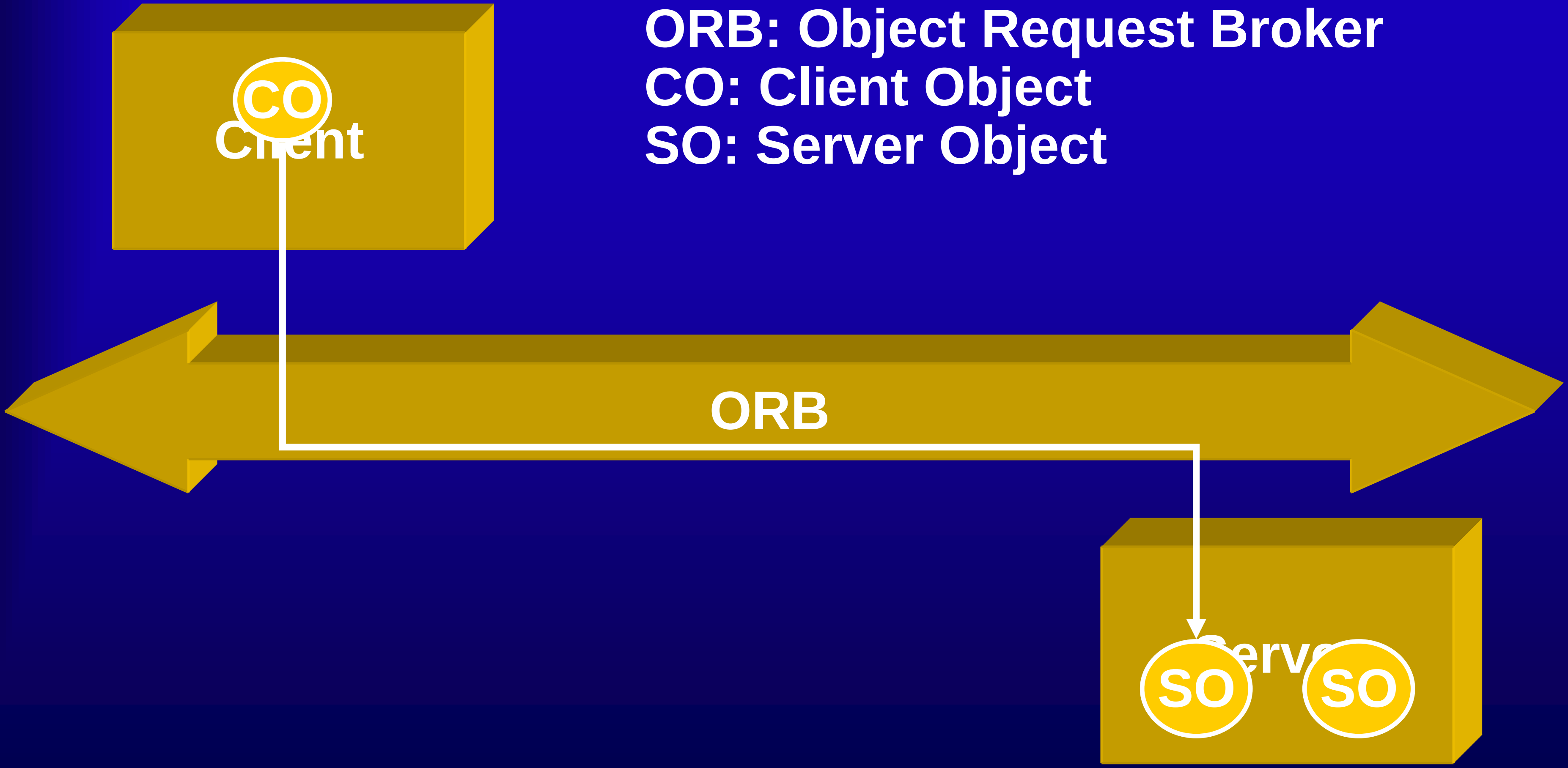
CORBA Architecture

ORB: Object Request Broker



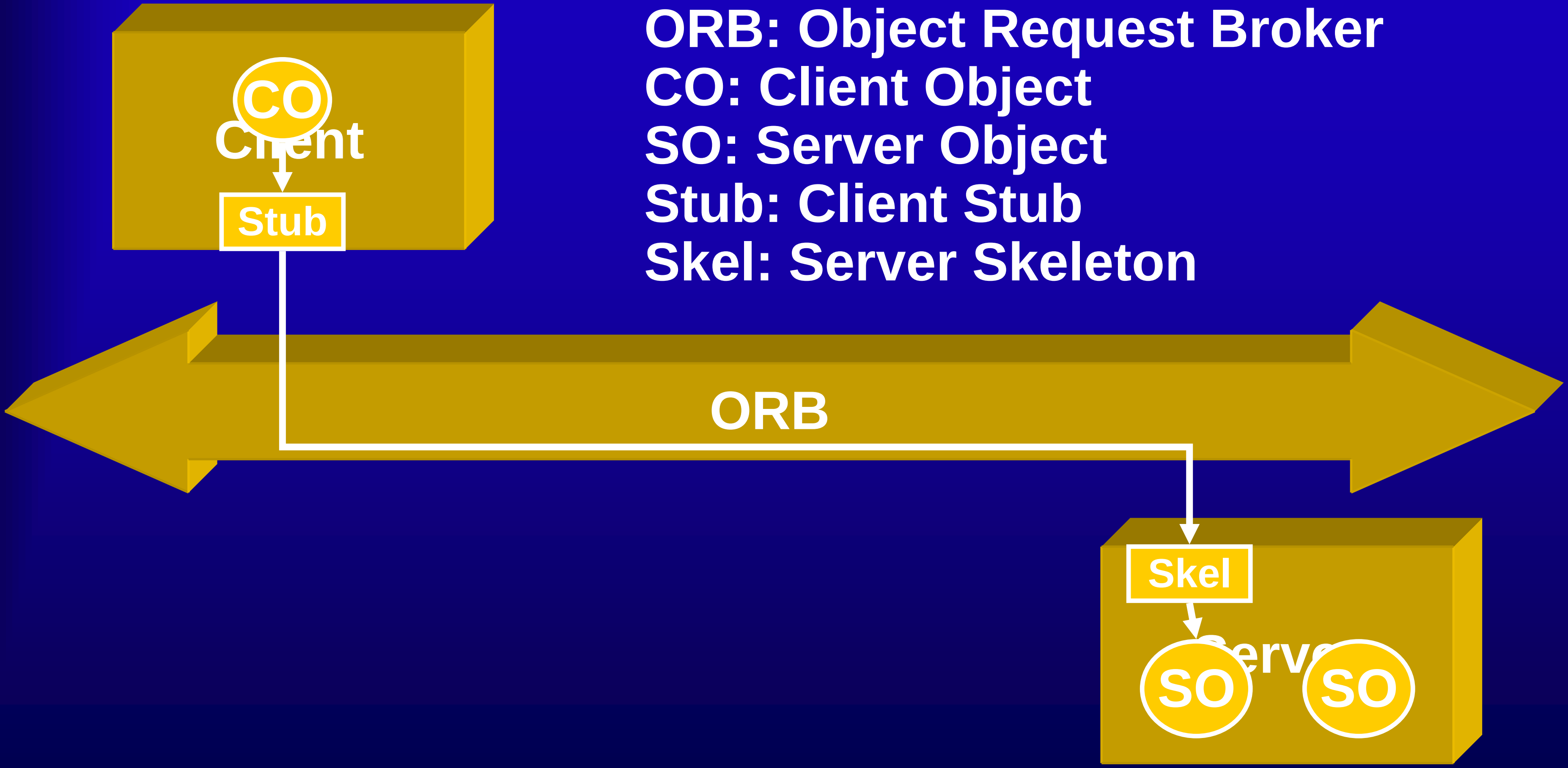
CORBA Architecture

ORB: Object Request Broker
CO: Client Object
SO: Server Object

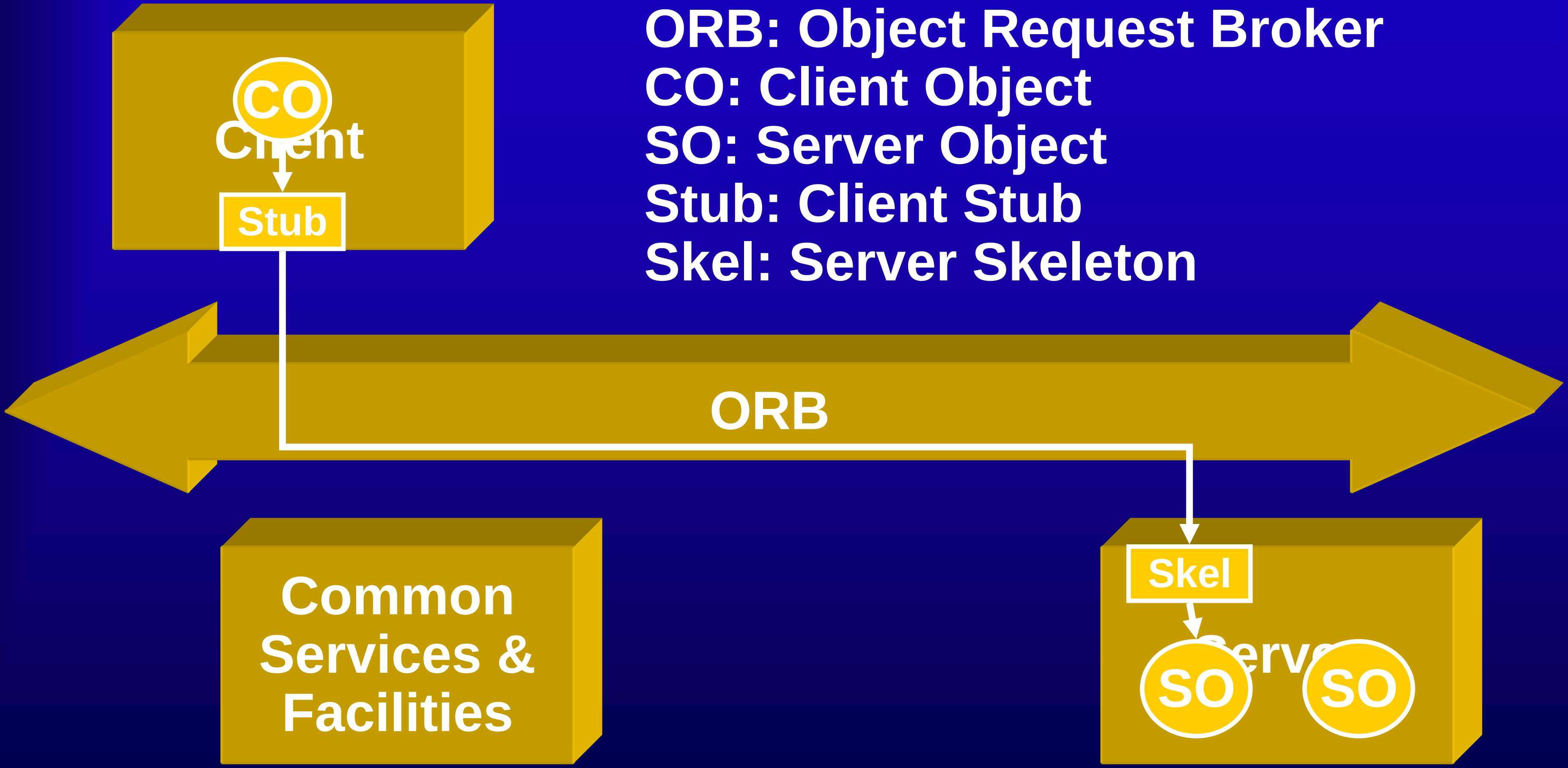


CORBA Architecture

ORB: Object Request Broker
CO: Client Object
SO: Server Object
Stub: Client Stub
Skel: Server Skeleton



CORBA Architecture



ORB: Object Request Broker
CO: Client Object
SO: Server Object
Stub: Client Stub
Skel: Server Skeleton

Interface Definition Language

- IDL separates the Interface from the Implementation
- Benefits of using an IDL
  - Ensure platform independence (e.g. NT, Unix)
  - Enforce modularity
  - Increase robustness
  - Enable language independence
- Many IDLs available
  - ASN.1, DCE IDL, ONC XDR, CORBA IDL

CORBA IDL

- Object-oriented, strongly typed, public interface specification language
- Independent of any particular language/compiler
- Mappings will be provided for many languages/compilers
  - C, C++, Smalltalk, COBOL, Modula3, DCE, Java, ...
- *Not* a programming language
  - similar to Java Interface / C++ abstract classes

CORBA IDL Elements

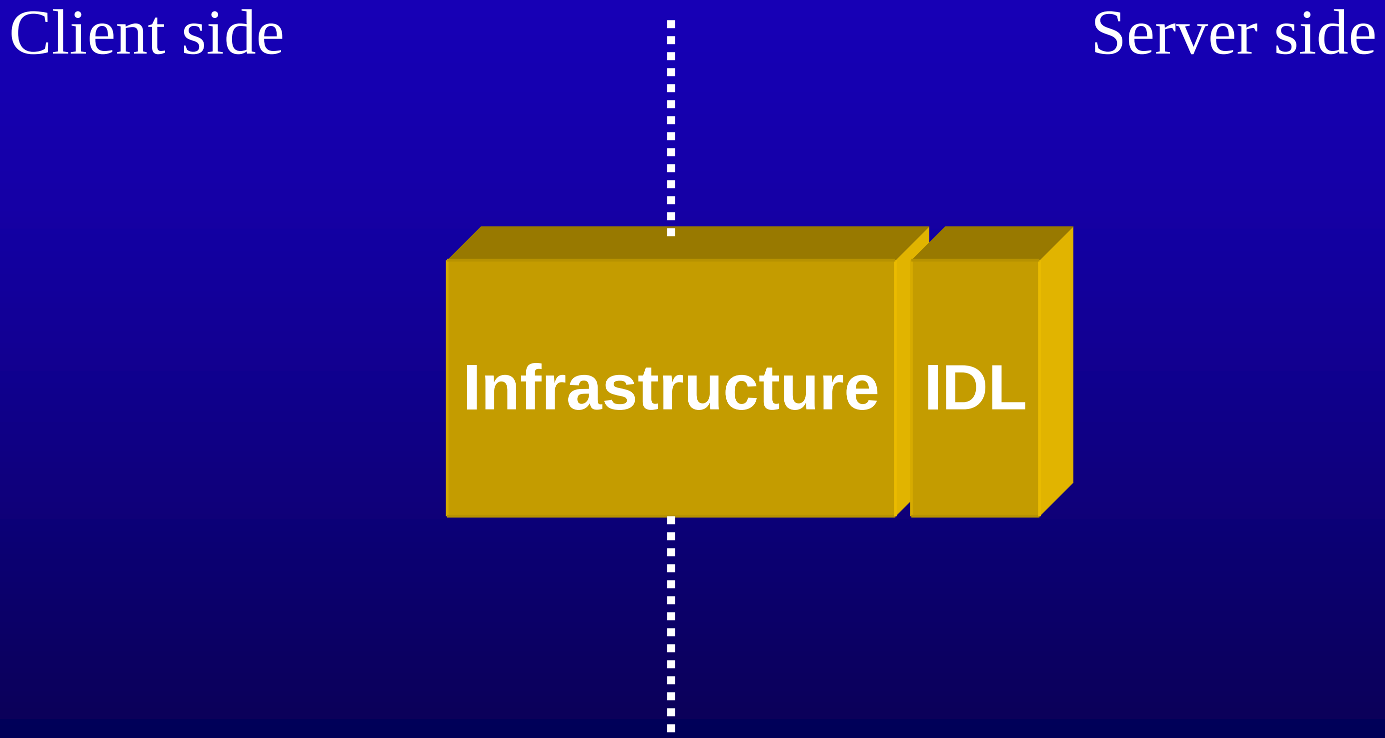
- `modules` and `interfaces`
- Operations and Attributes
- Single and multiple inheritance
- Basic types (`double`, `long`, `char`, etc.)
- `any` type
- Arrays and `sequence`
- `struct, enum, union, typedef`
- `consts`
- `exceptions`

Differences from C++ or Java

- No control constructs
- No data members
- No pointers
- No con-/destructors
- No overloaded operations
- No `int` data type
- Contains parameter passing modes
- Unions require a tag
- Different String type
- Different Sequence type
- Different exception interface
- No templates
- `oneway` call semantics
- `readonly` keyword

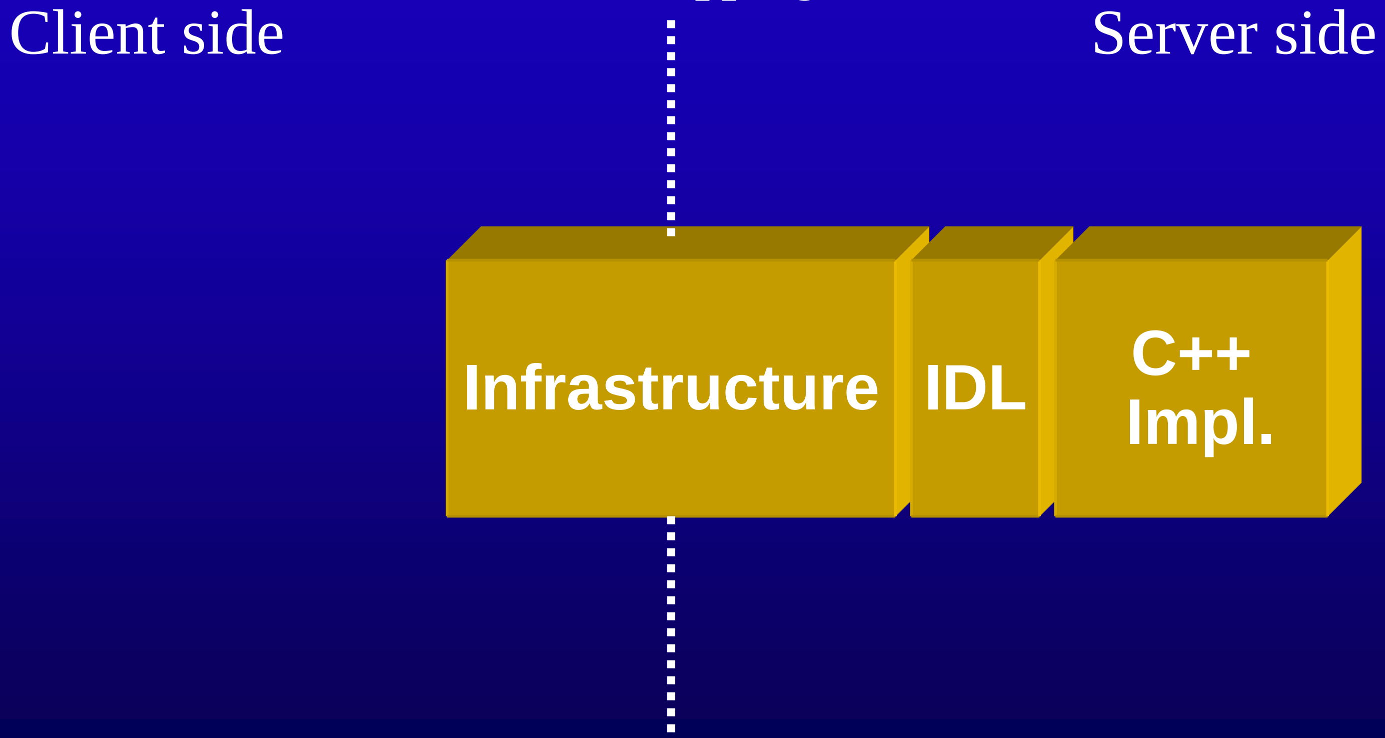
CORBA Interface Definition Language

IDL isolates interface from implementation



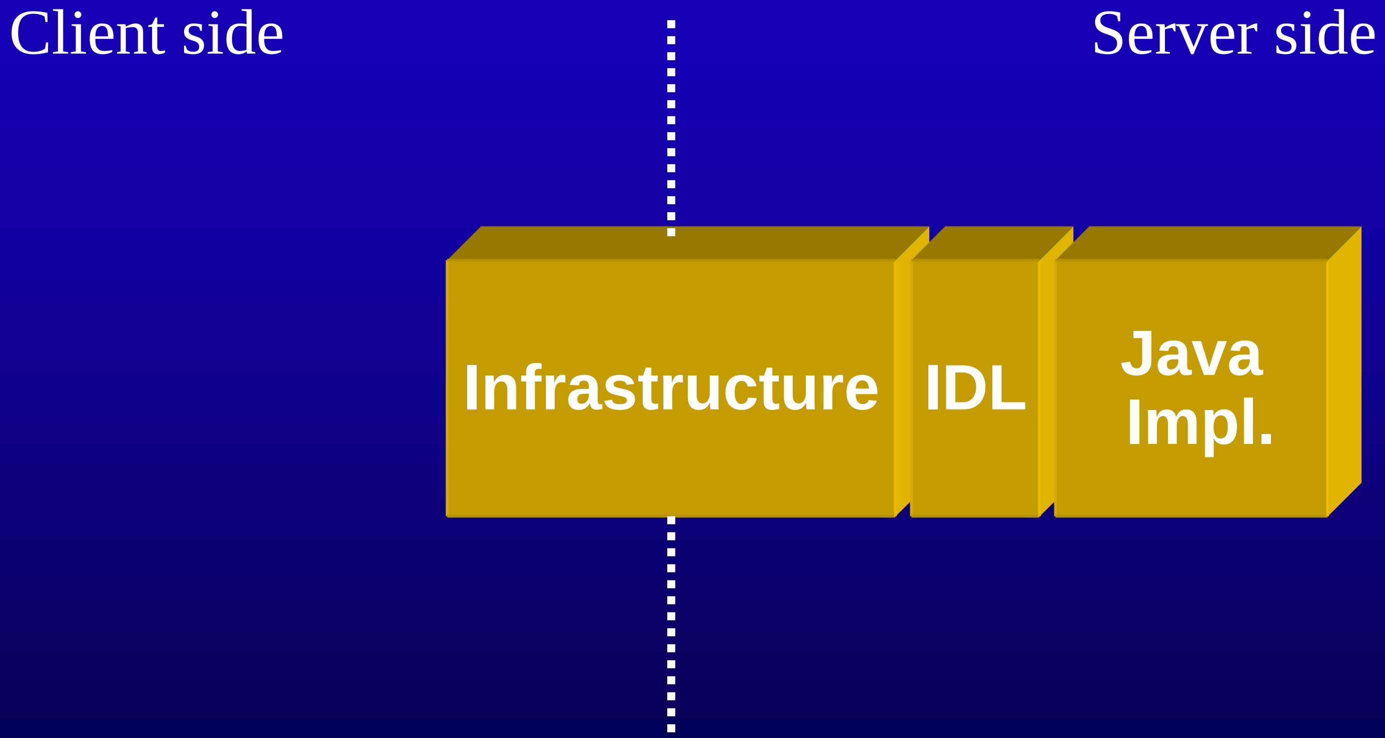
CORBA Interface Definition Language

You can then implement the interface e.g. in C++ using the C++ language mapping.



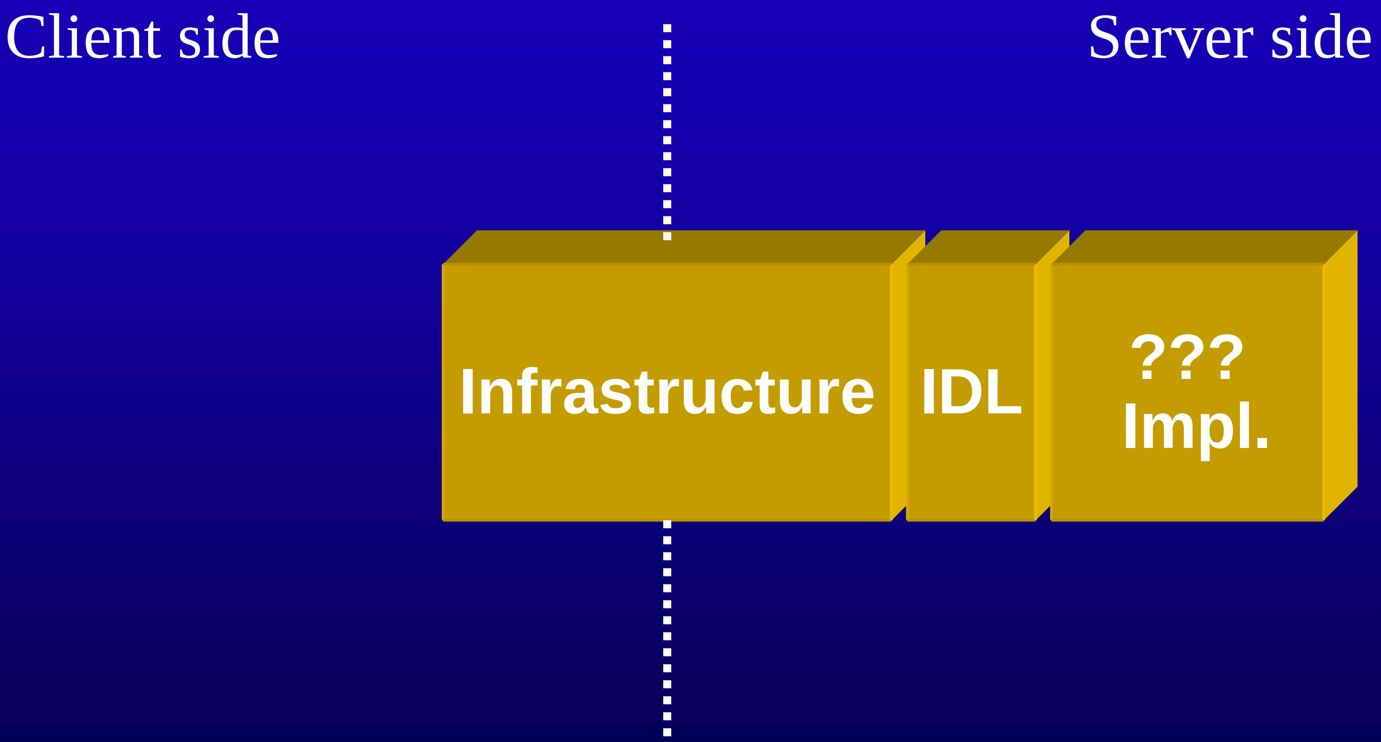
CORBA Interface Definition Language

Or perhaps you better like Java.



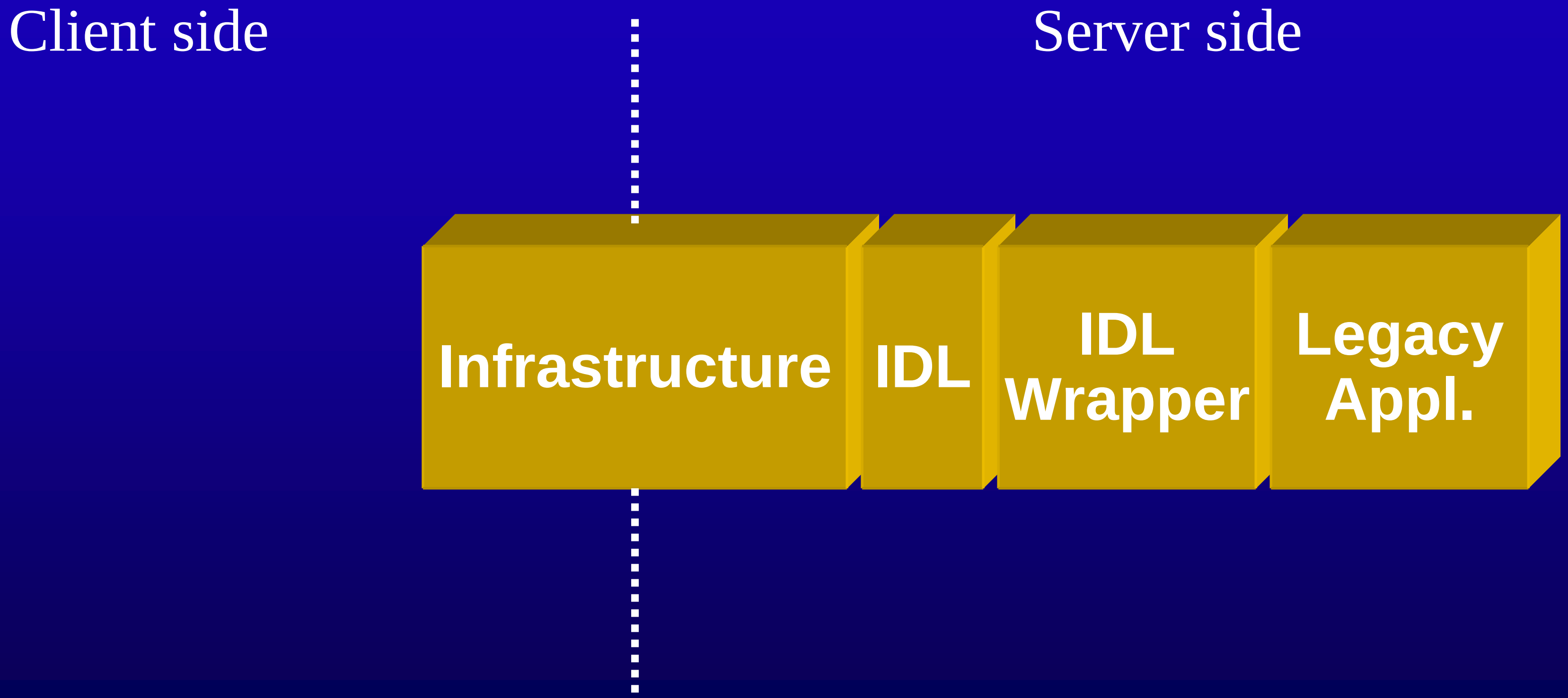
CORBA Interface Definition Language

Or C, Smalltalk, Cobol or a number of other languages.



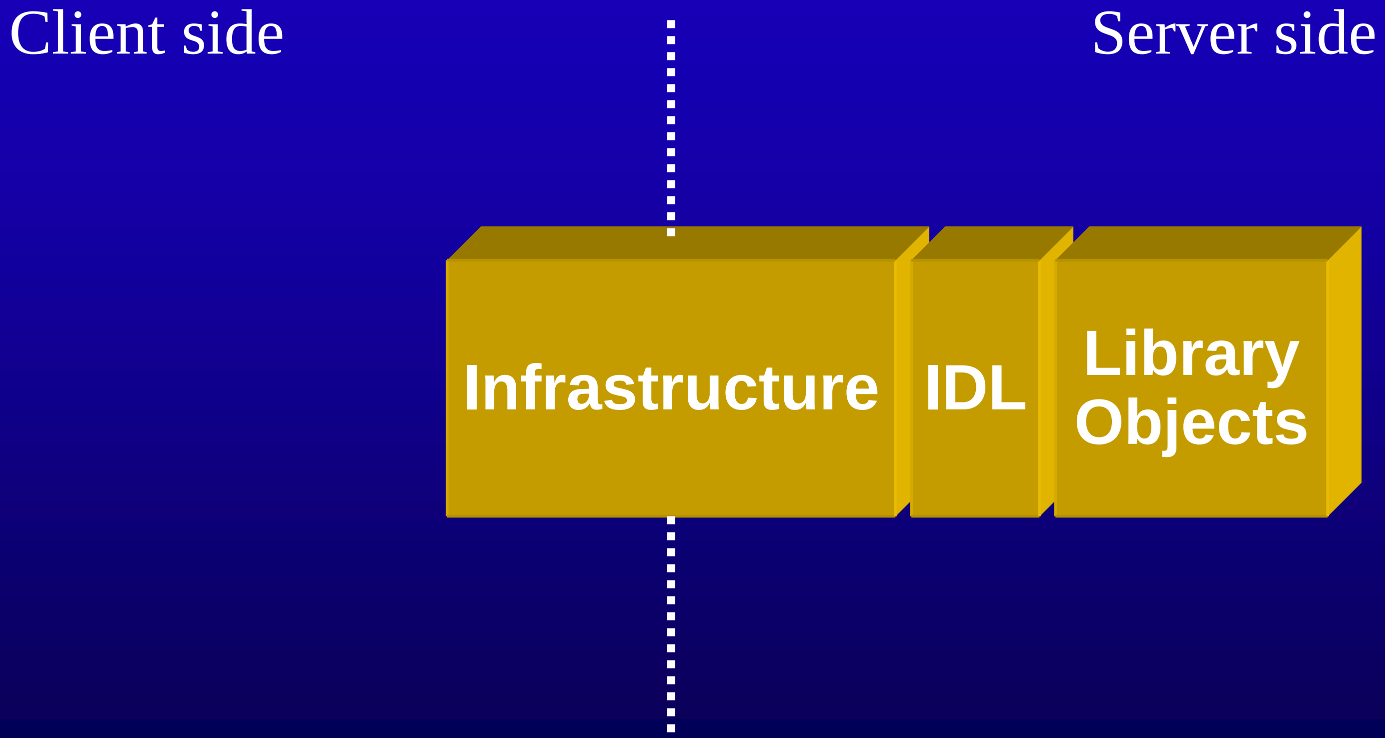
CORBA Interface Definition Language

You may also wrap legacy applications using IDL interfaces



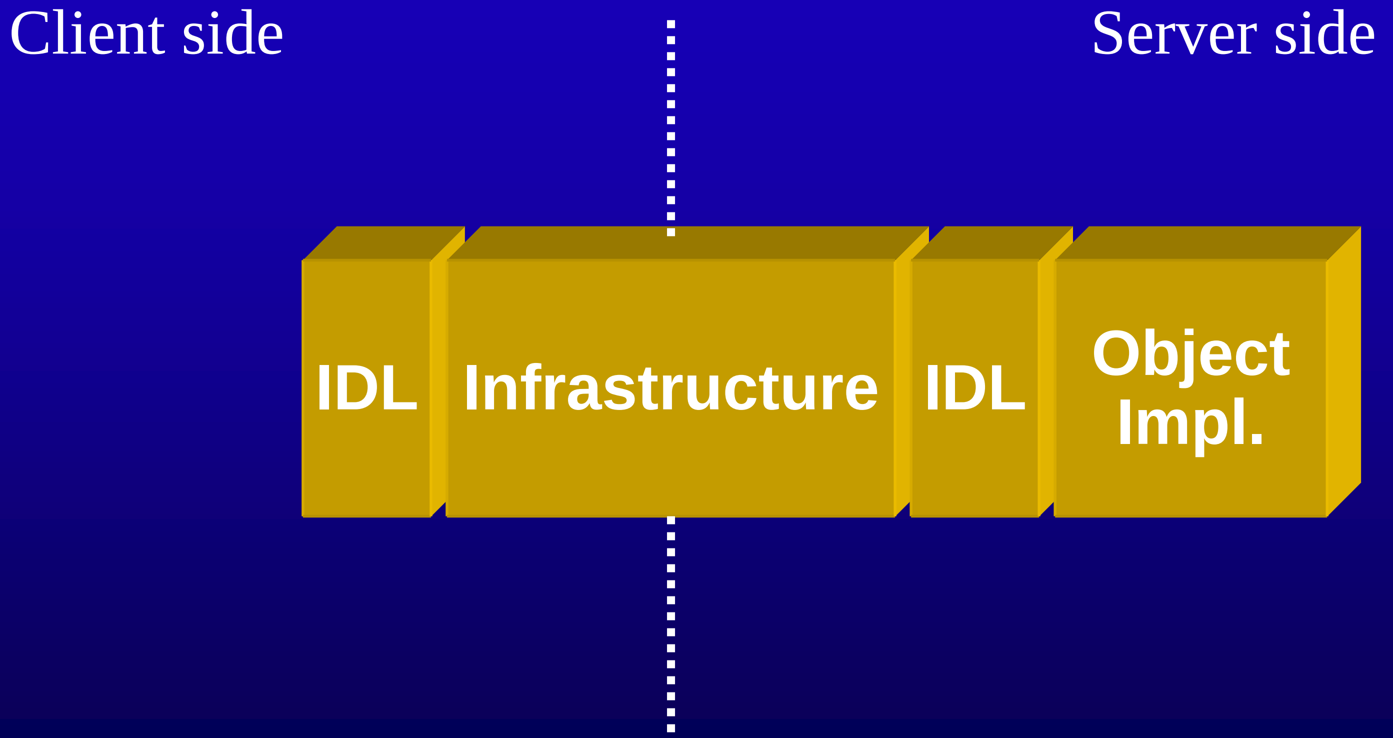
CORBA Interface Definition Language

Or 3rd party libraries.



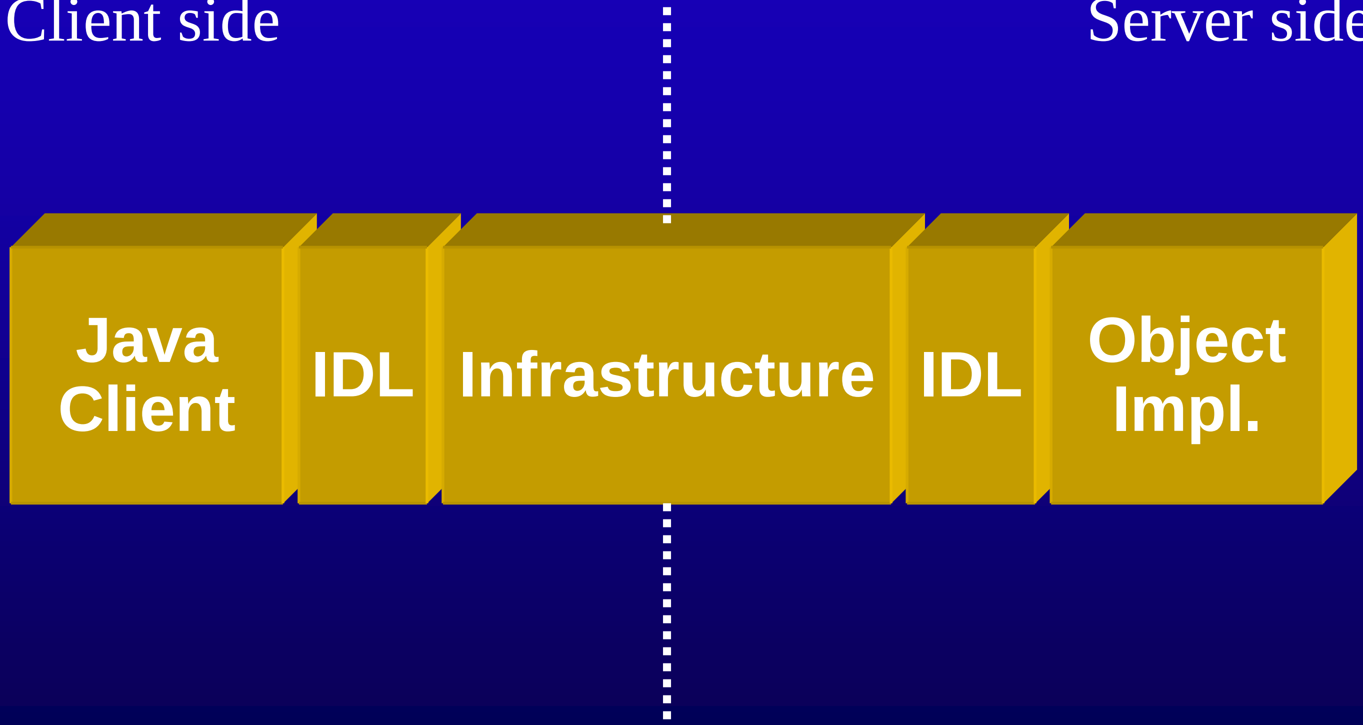
CORBA Interface Definition Language

The same IDL defines the client's interface.



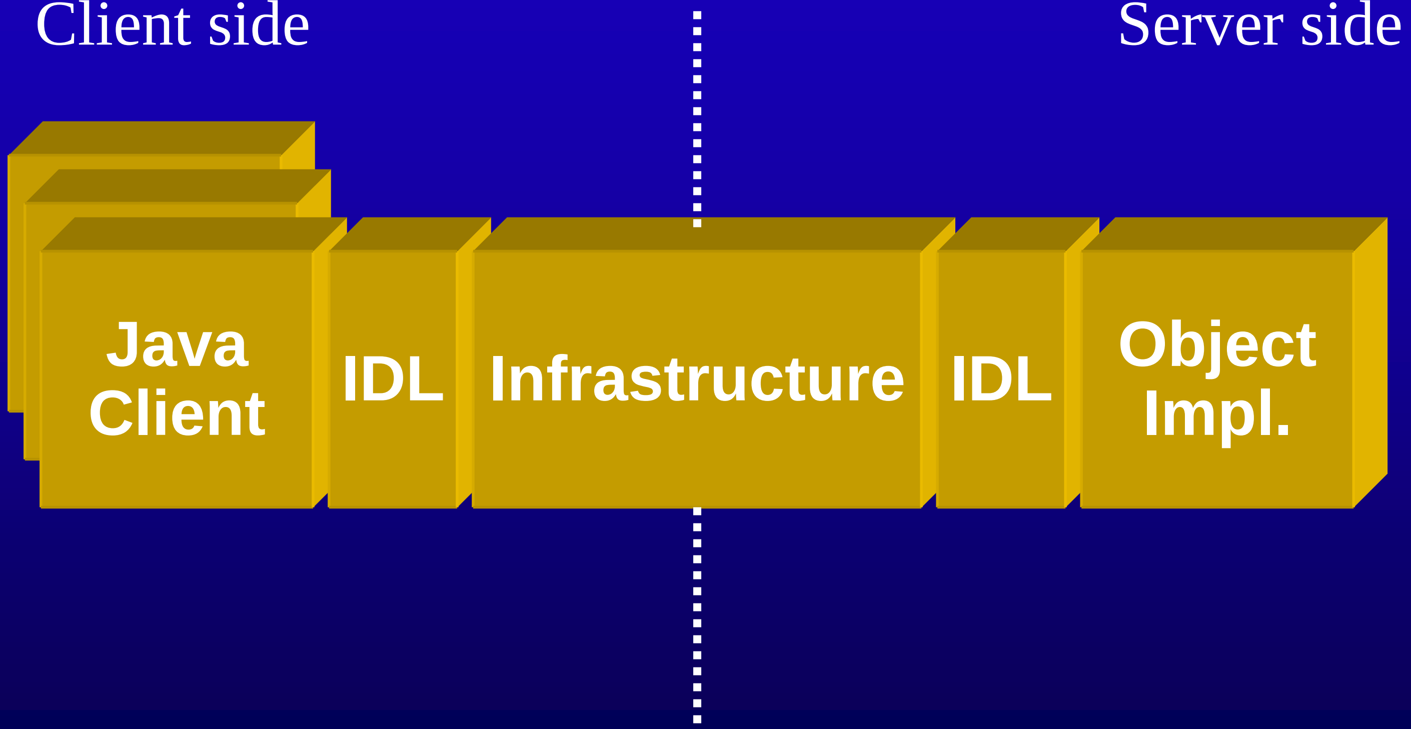
CORBA Interface Definition Language

Then you can implement the client e.g. using Java ...

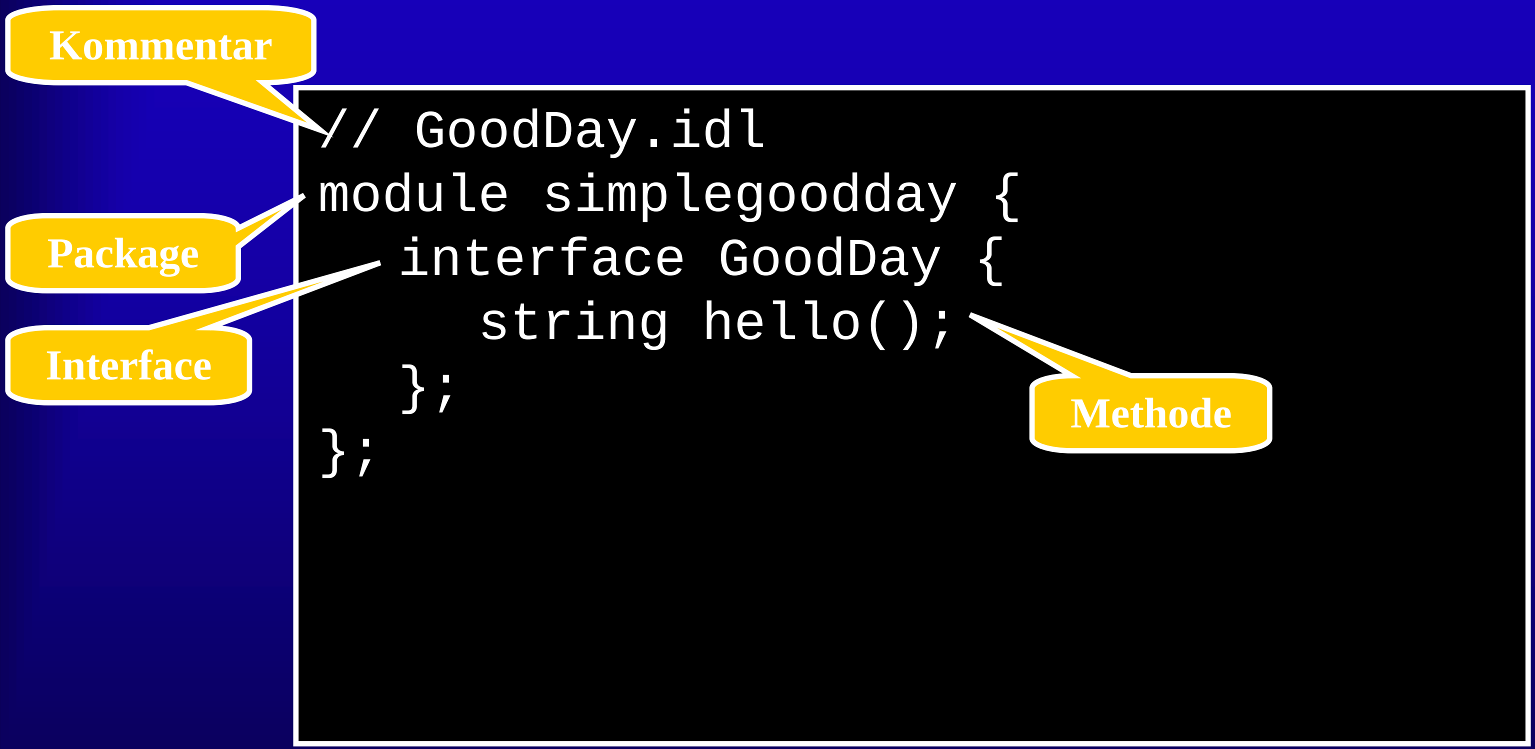


CORBA Interface Definition Language

or C++, Ada, Smalltalk, ...

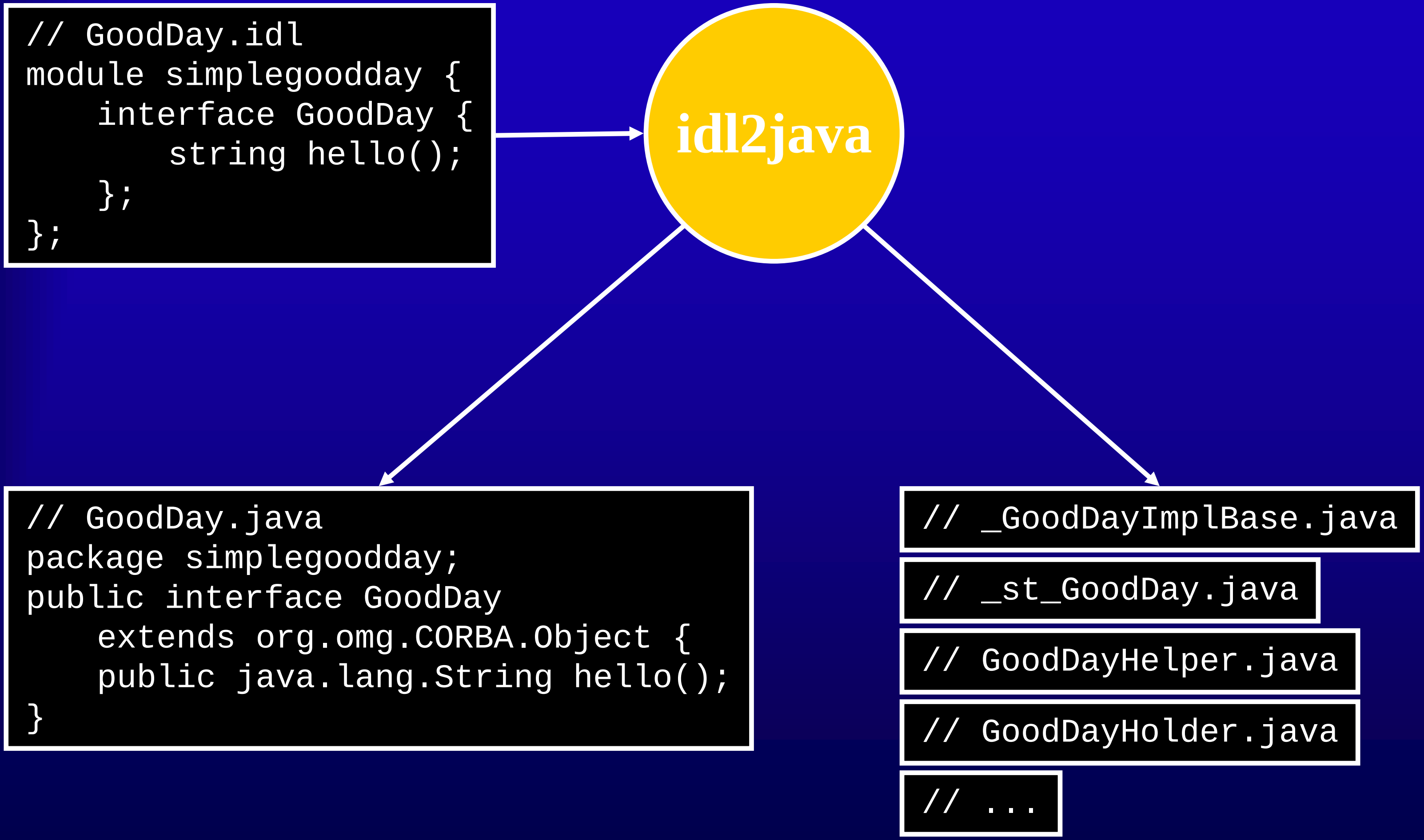


CORBA IDL Example



```
// GoodDay.idl
module simplegoodday {
   interface GoodDay {
      string hello();
   };
};
```

CORBA IDL Compiler



```
// GoodDay.idl
module simplegoodday {
    interface GoodDay {
        string hello();
    };
};
```

```
// GoodDay.java
package simplegoodday;
public interface GoodDay
    extends org.omg.CORBA.Object {
    public java.lang.String hello();
}
```

```
// _GoodDayImplBase.java
```

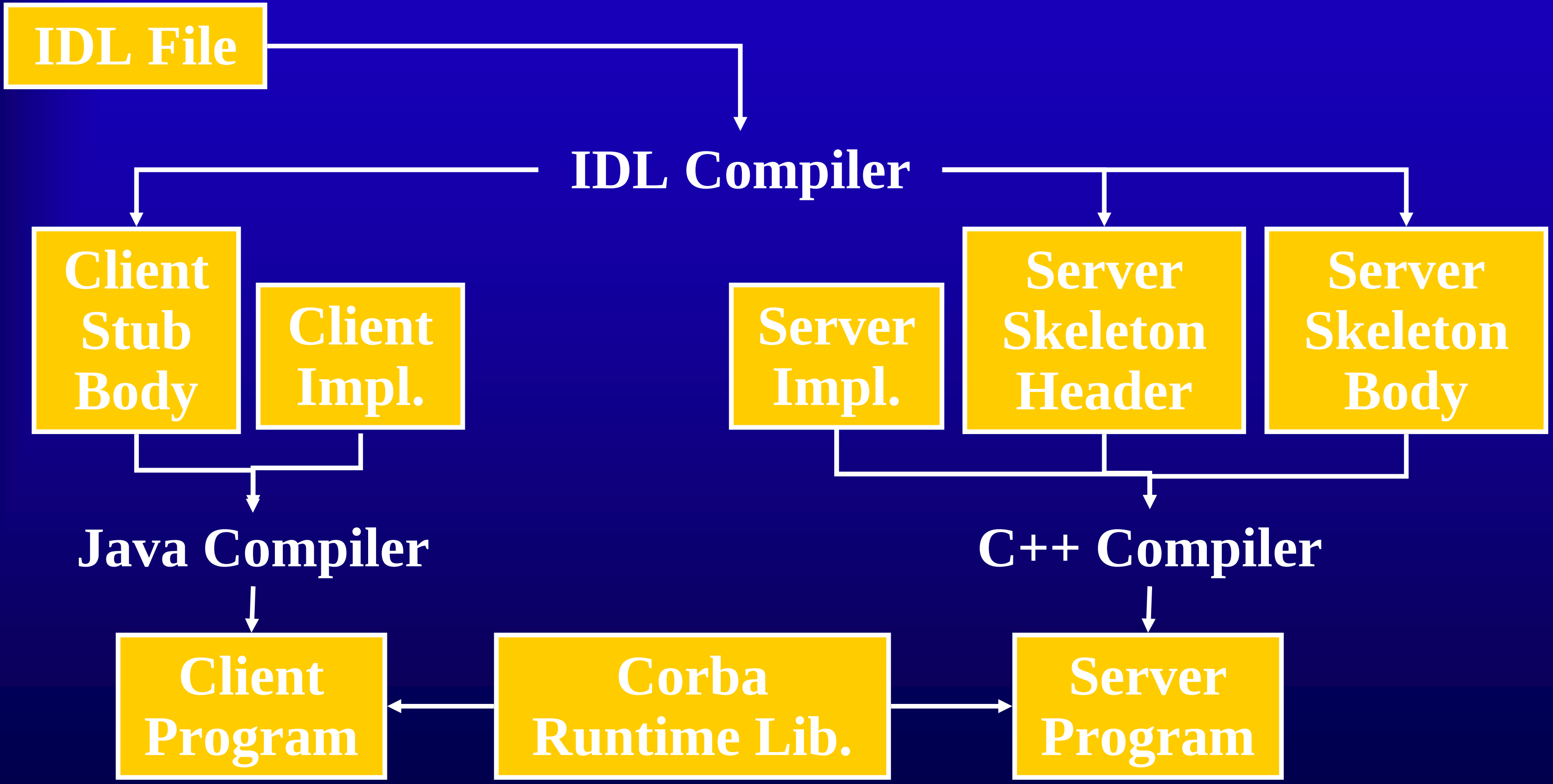
```
// _st_GoodDay.java
```

```
// GoodDayHelper.java
```

```
// GoodDayHolder.java
```

```
// ...
```

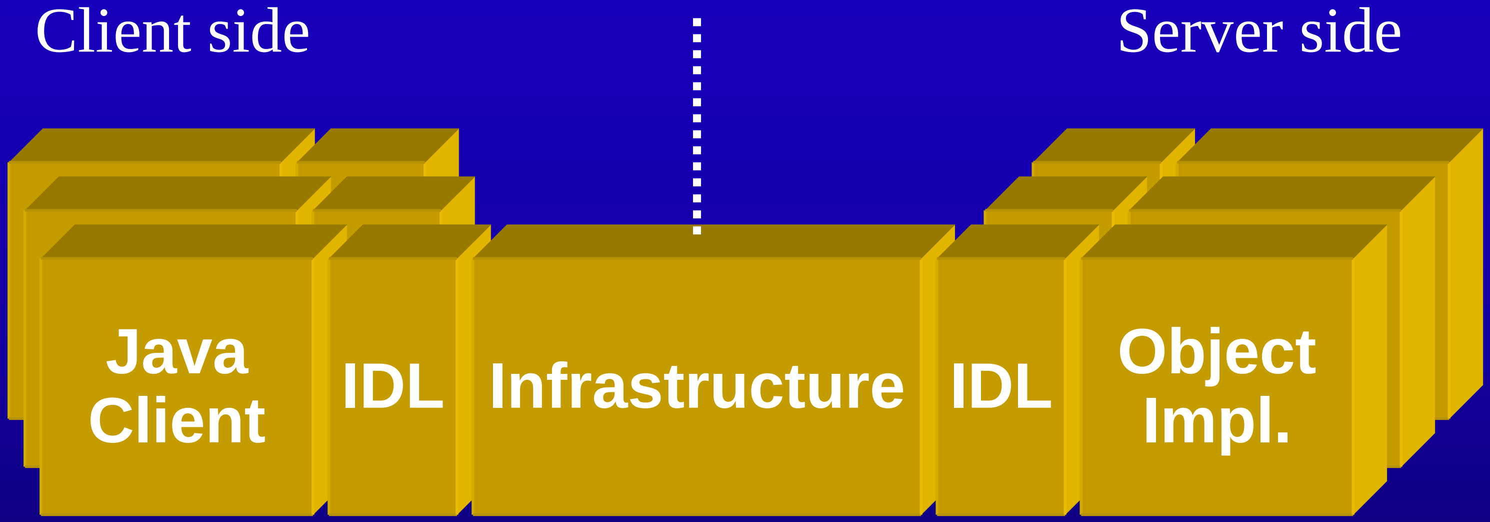
CORBA IDL Compiler



CORBA IDL

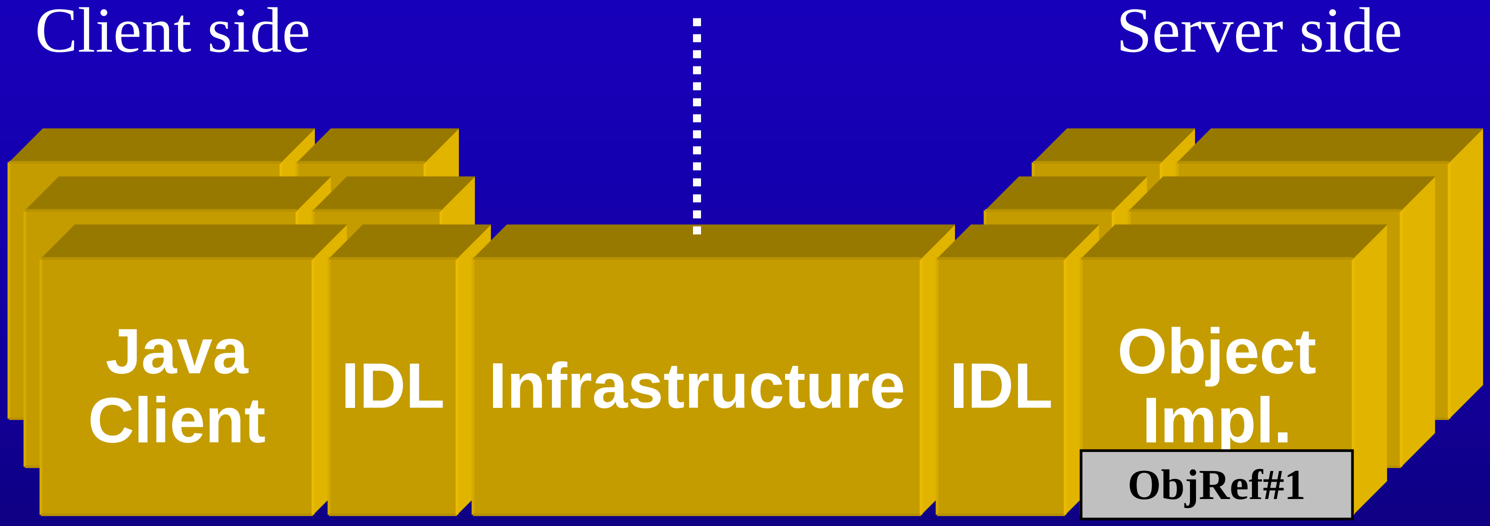
```
// complex.idl
module de {
    module uni-ulm {
        interface Meeting {
            readonly attribute string purpose;
            readonly attribute string participants;
            oneway void destroy();
        };
        interface MeetingFactory {
            Meeting CreateMeeting(in string purpose, in string part);
        };
        interface Room {
            enum Slot { am9, am10, pm12, pm1, pm2, pm3, pm4};
            const short MaxSlots = 8;
            typedef Meeting Meetings[MaxSlots];
            exception NoMeetingInThisSlot{};
            exception SlotAlreadyTaken{};
            readonly attribute string name;
            Meetings View();
            void Book(in Slot a_slot, in Meeting a_meeting)
                raises(SlotAlreadyTaken);
            void Cancel(in Slot a_slot) raises(NoMeetingInThisSlot);
        };
    };
};
```

Object Invocation



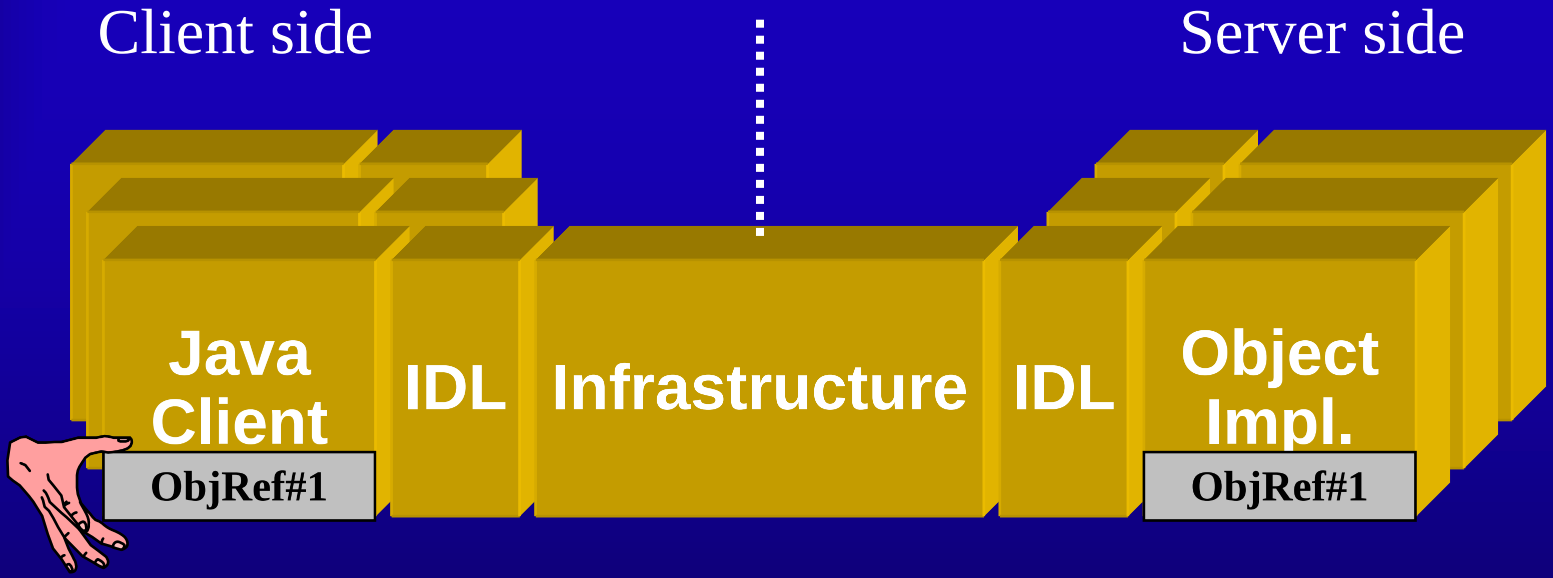
There will be a number of different Object Implementations, each with its own IDL interface...

Object Invocation



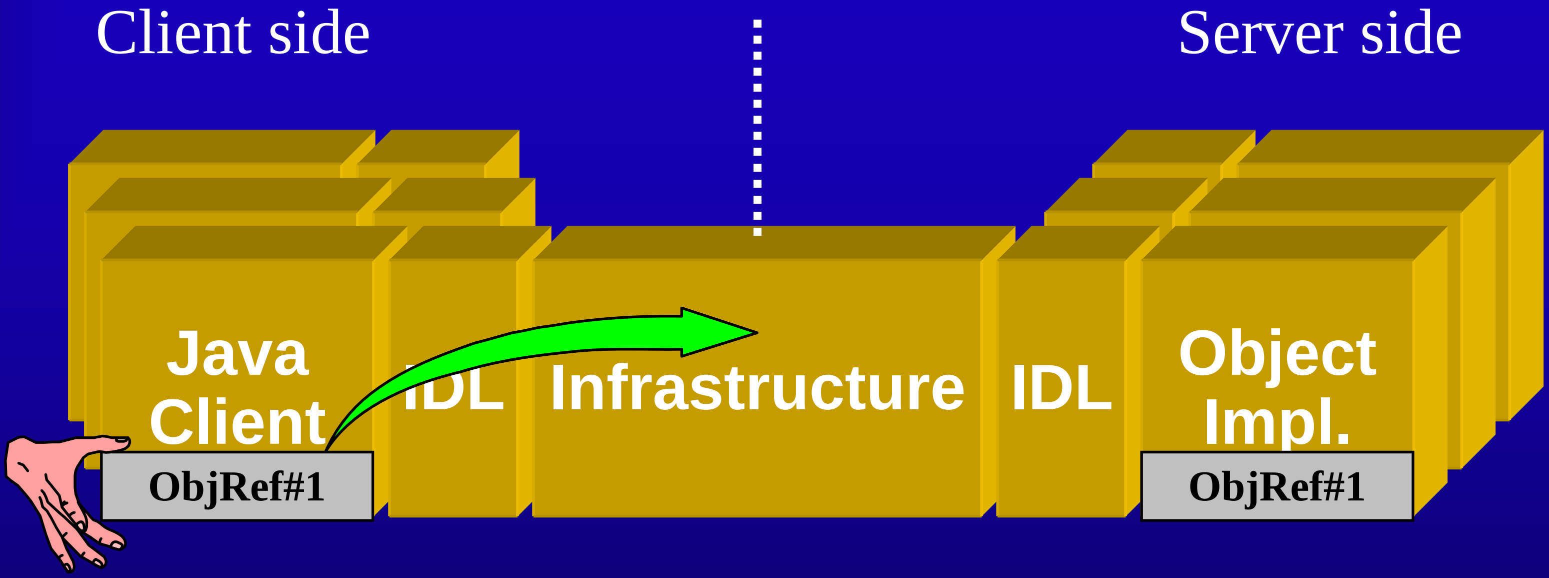
Each Object Implementation is assigned a unique Object Reference...

Object Invocation



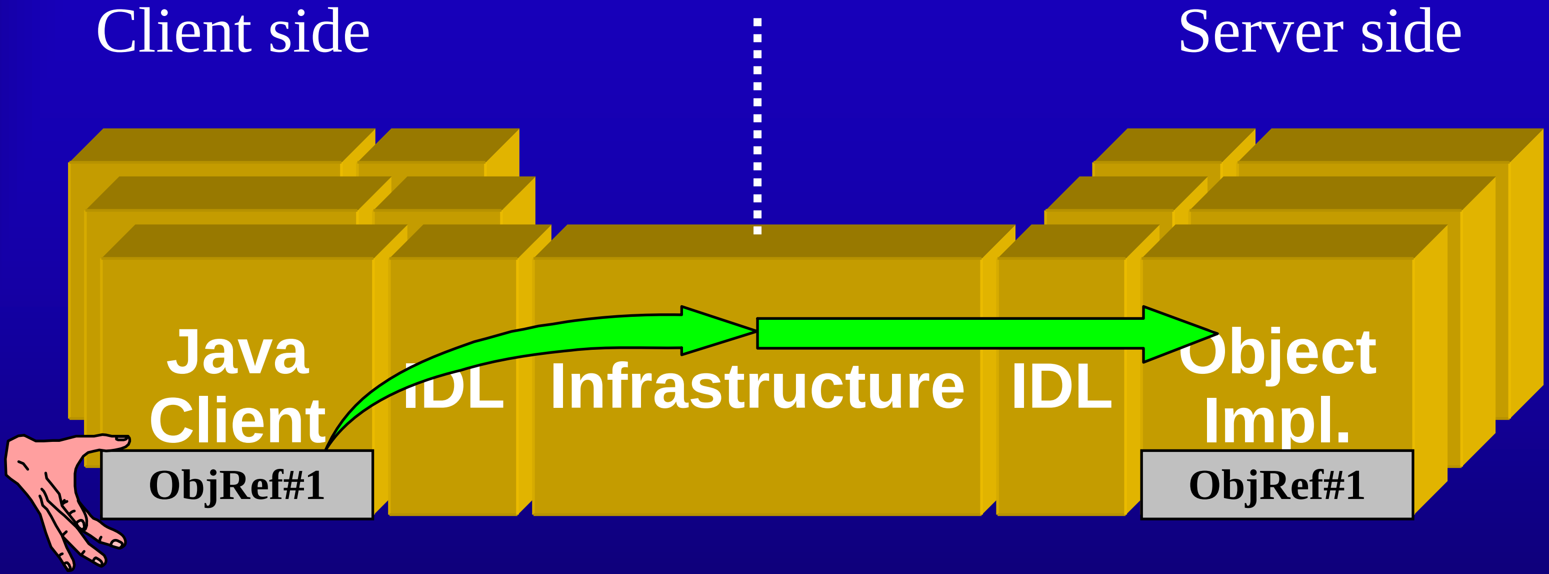
Clients obtain an Object Reference via several ways.

Object Invocation



Clients use the Object Reference to inform the infrastructure...

Object Invocation



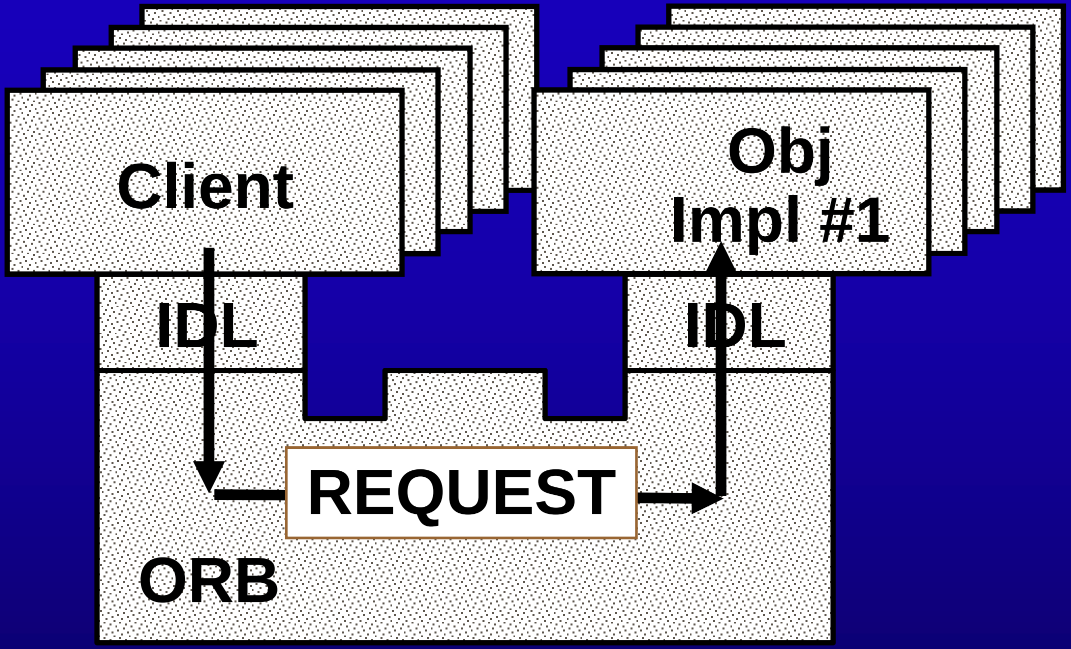
about the Target Object Implementation for their request.

Roles of Infrastructure

- Provides a Local, Well-Known Point of Contact for All Object Invocations a Client may make
- Passes invocation to Local or Remote Target Object Implementation
- Understands IDL; Maintains Repository of available Object Interfaces
- Also Maintains Repository of Available Implementations
- Federates this information across the System

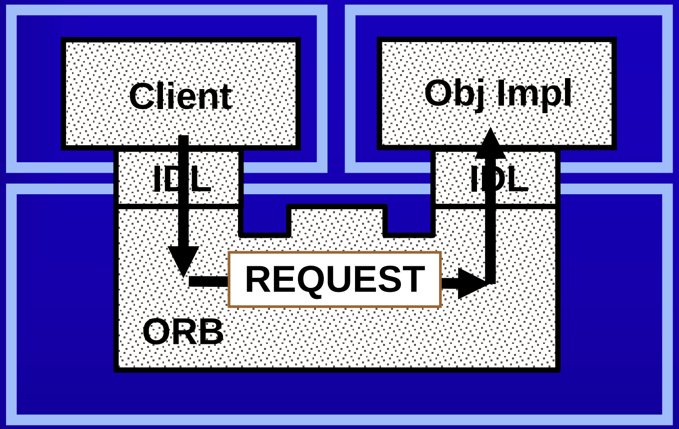
A WEB OF INTERCONNECTED ORBs

CORBA Architecture

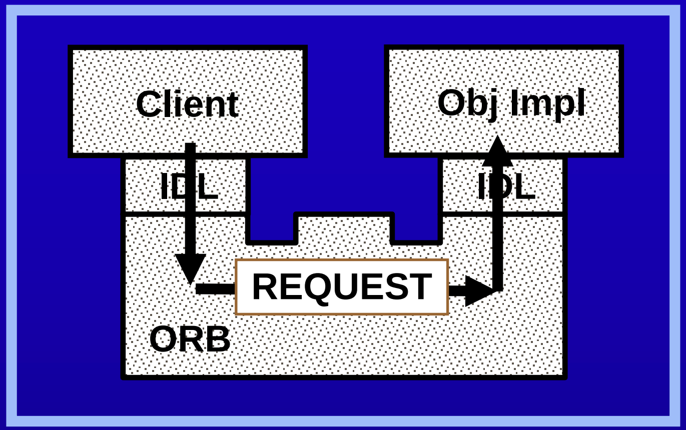


An Object Request Broker relays the Invocation from Client to Object Implementation, and the result back to the Client.

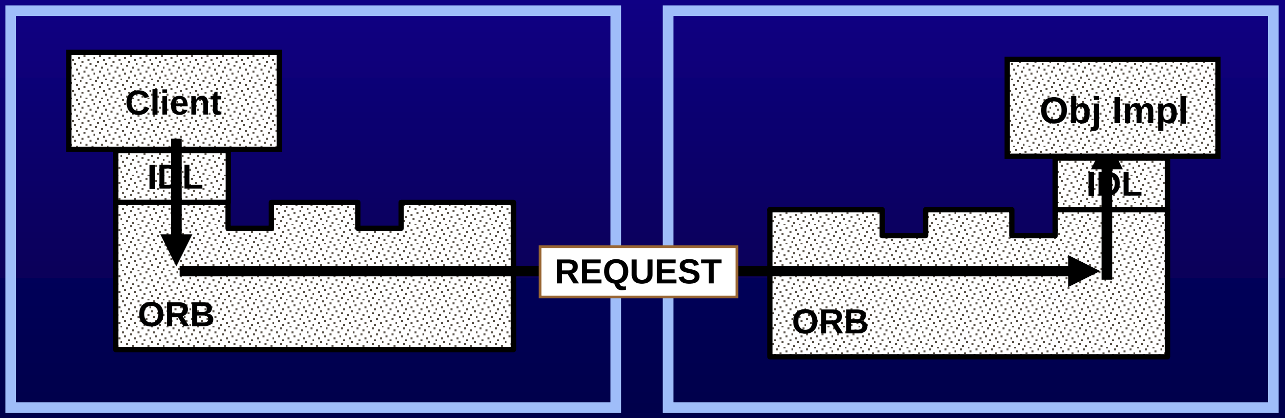
Different ORB Types



Server or Operating-System Based

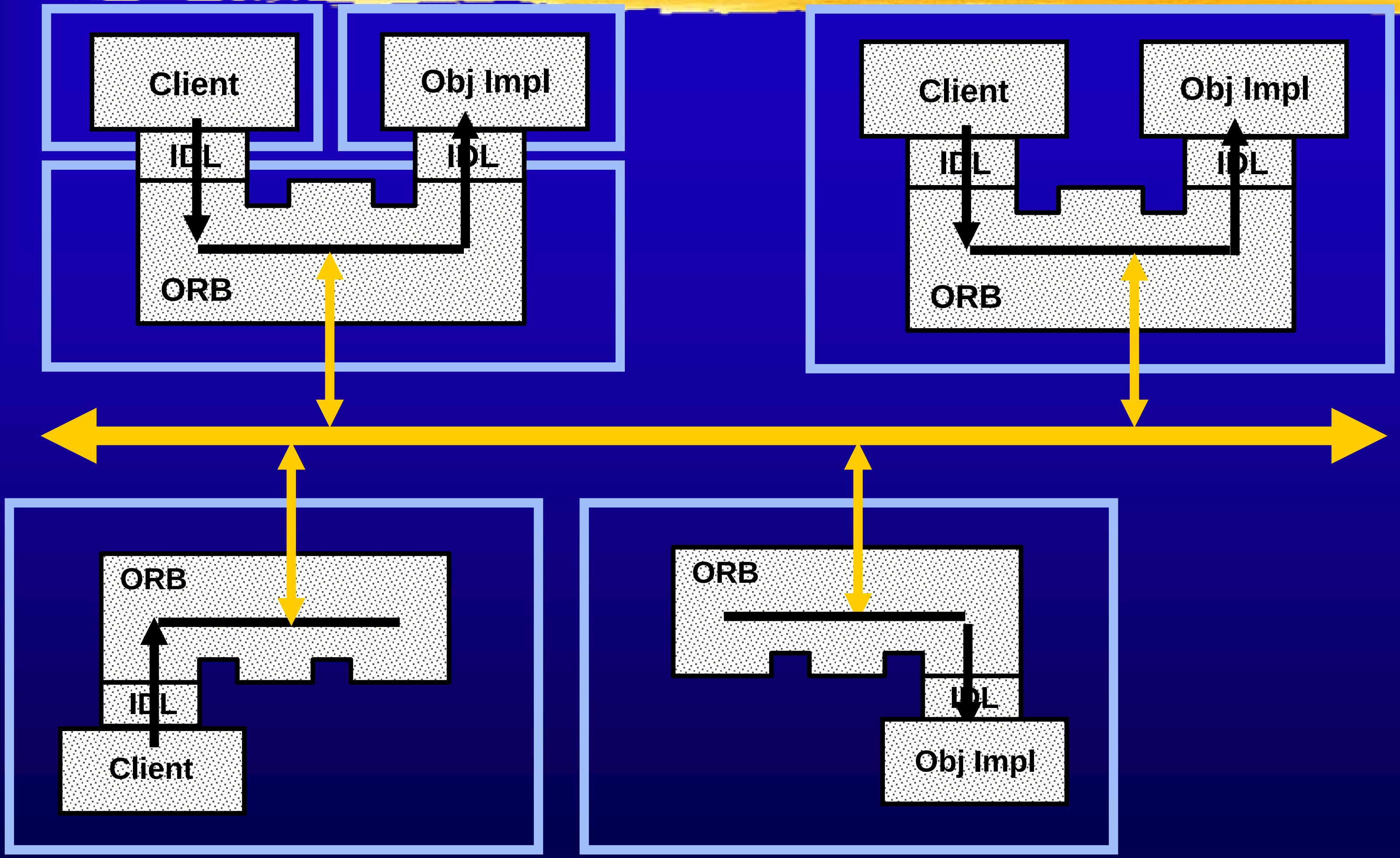


Single-Process
Library Resident



Client &
Implementation
Resident

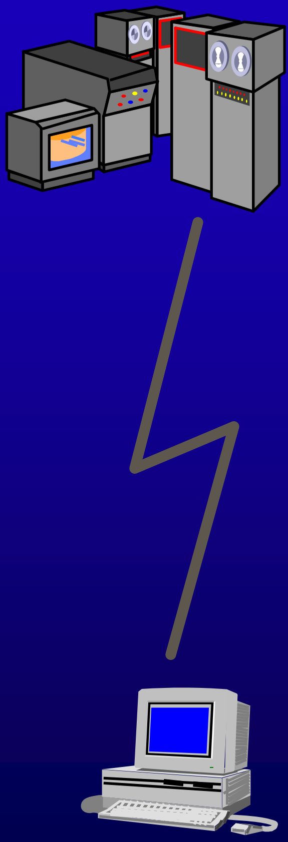
ORB to ORB Interoperability



CORBA Interoperability

CORBA 2.0 Interoperability Comprises:

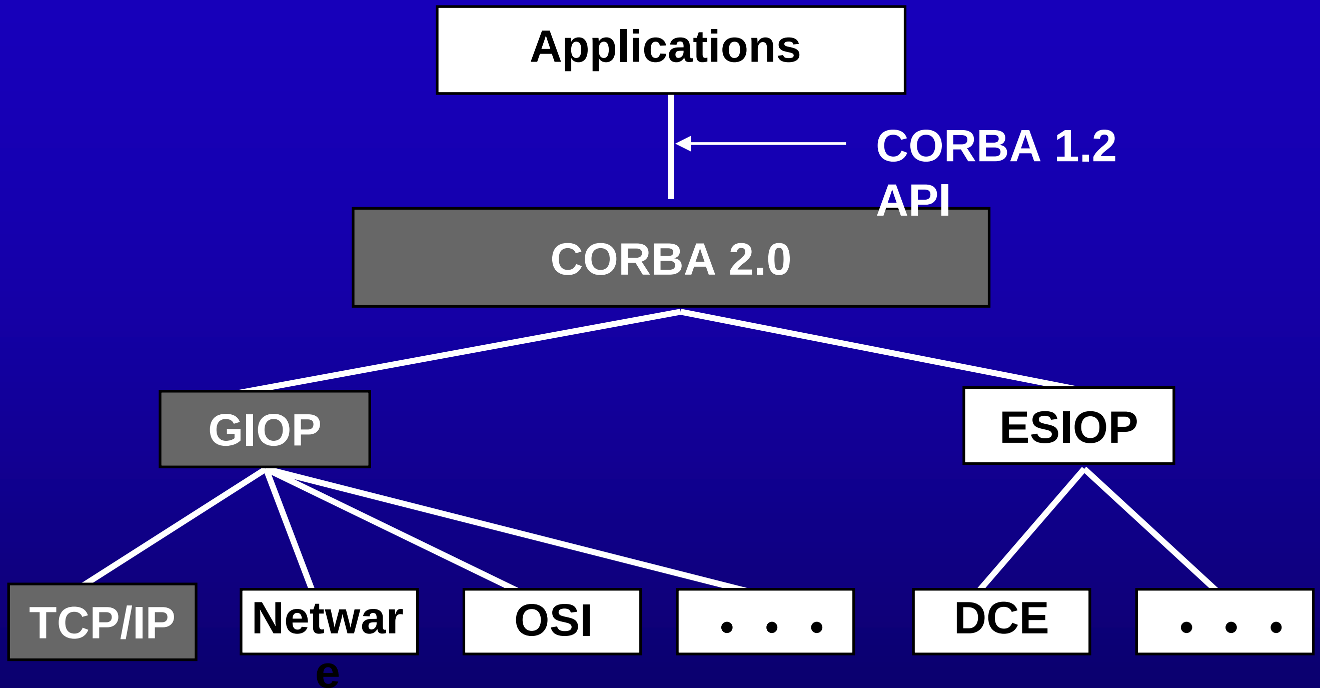
- An overall architecture for CORBA-CORBA communications;
- An API for adding bridges;
- A general multi-transport message format (General Inter-ORB Protocol or GIOP);
- An API for gateways using ESIOPs -- (Environment-Specific Inter-ORB Protocols)



UNIVERSAL, OUT-OF-THE-BOX INTEROPERABILITY:

- GIOP over TCP/IP (IIOP) is *mandatory* for compliance either internally or via a half-bridge;
- DCE ESIOP is optional for all implementations.

CORBA 2.0 Interoperability Spec



Applications

CORBA 1.2 API

CORBA 2.0

GIOP

ESIOP

TCP/IP

Netware

OSI

. . .

DCE

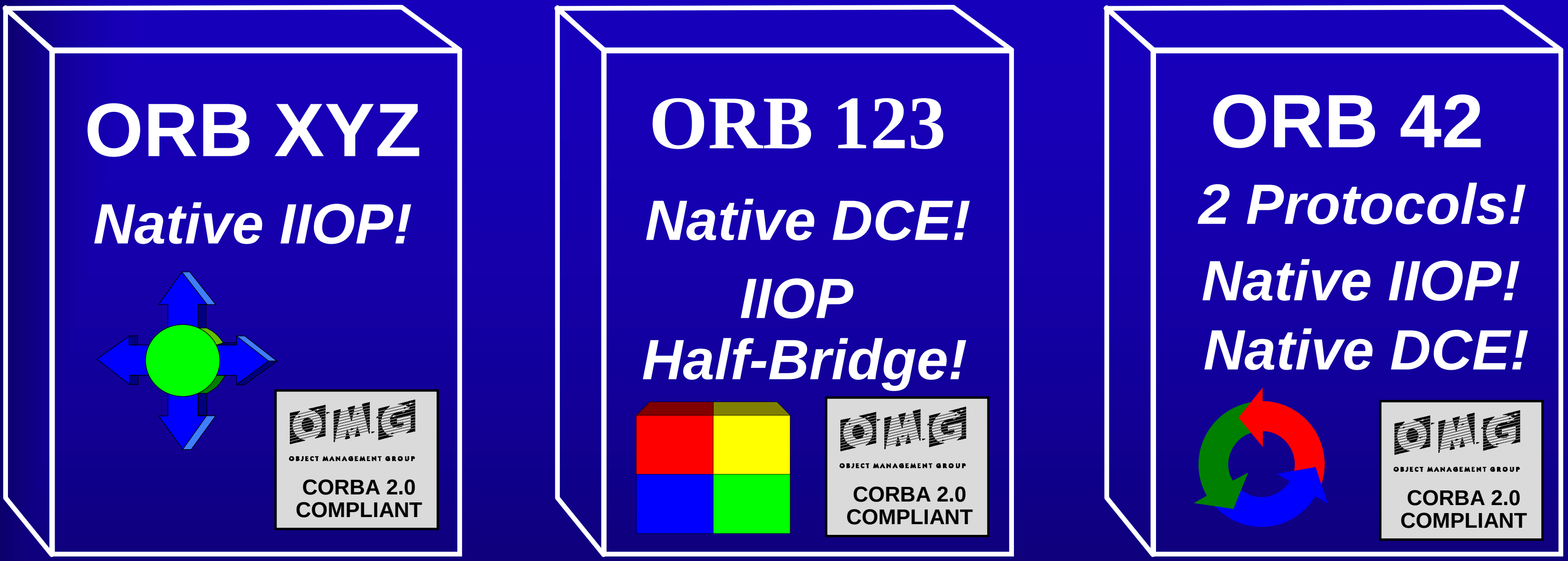
. . .

A General Inter-ORB Protocol (GIOP) is a message format hosted on any network transport. ***TCP/IP is required for CORBA 2.0, either native or via a half-bridge.***

An Environment Specific Inter-ORB Protocol (ESIOP) is an optional approach for special environments (real time systems, existing base of DCE, etc.)

▨ Mandatory portion: provides “out-of-the-box” interoperability

CORBA 2.0 Compliance



ORB XYZ

Native IIOP!

OMG OBJECT MANAGEMENT GROUP CORBA 2.0 COMPLIANT

ORB 123

Native DCE!

IIOP Half-Bridge!

OMG OBJECT MANAGEMENT GROUP CORBA 2.0 COMPLIANT

ORB 42

2 Protocols!

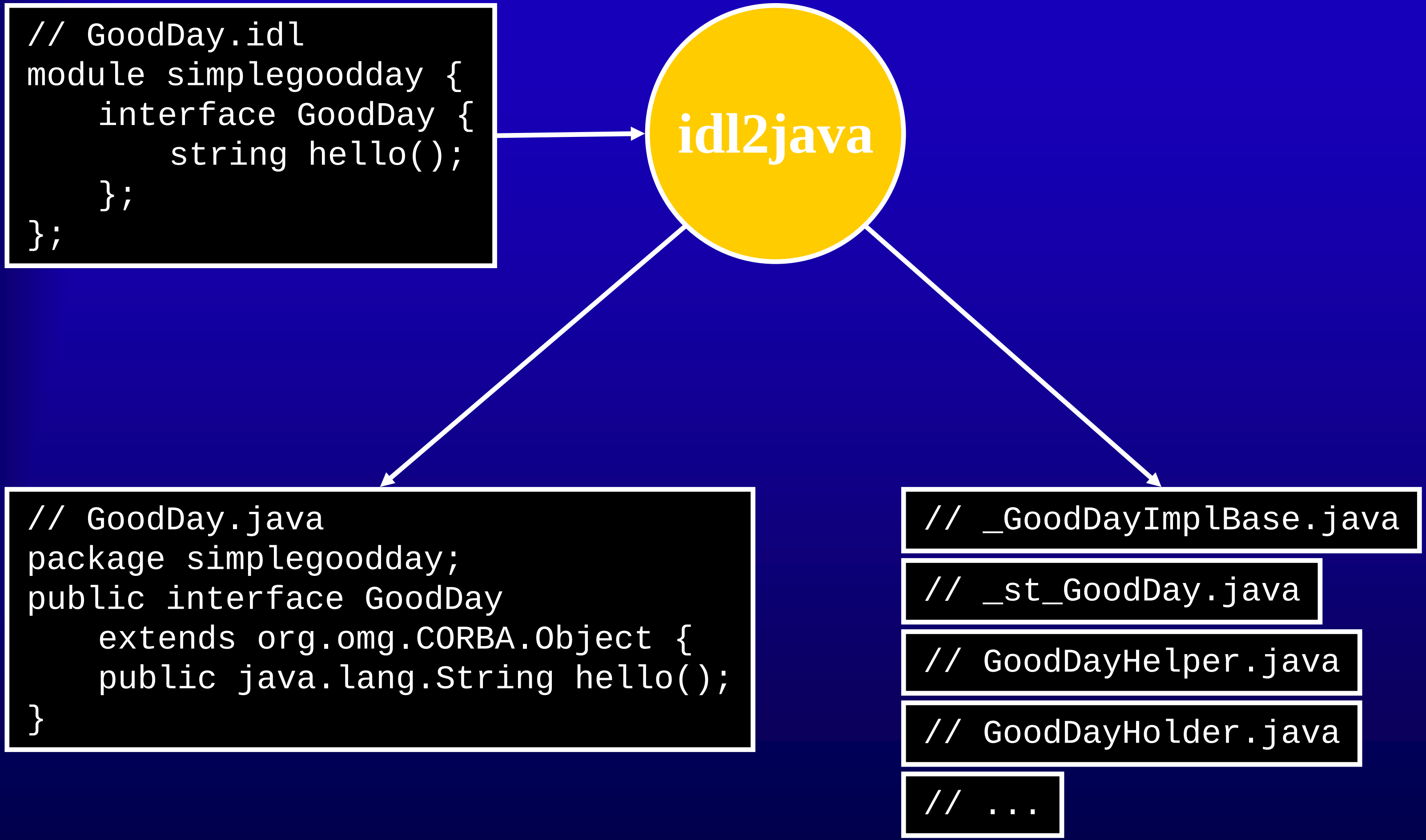
Native IIOP!

Native DCE!

OMG OBJECT MANAGEMENT GROUP CORBA 2.0 COMPLIANT

- These products can all interoperate with every IIOP ORB,
- and they can all display the CORBA 2.0 brand.

Implementing CORBA Applications



```
// GoodDay.idl
module simplegoodday {
    interface GoodDay {
        string hello();
    };
};
```

```
// GoodDay.java
package simplegoodday;
public interface GoodDay
    extends org.omg.CORBA.Object {
    public java.lang.String hello();
}
```

```
// _GoodDayImplBase.java
```

```
// _st_GoodDay.java
```

```
// GoodDayHelper.java
```

```
// GoodDayHolder.java
```

```
// ...
```

Implementing CORBA Applications

```
// _GoodDayImplBase.java
package simplegoodday;
abstract public class _GoodDayImplBase extends
    com.inprise.vbroker.CORBA.portable.Skeleton implements simplegoodday.GoodDay {
  protected simplegoodday.GoodDay _wrapper = null;
  public simplegoodday.GoodDay _this() {
    return this;
  }
  protected _GoodDayImplBase(java.lang.String name) {
    super(name);
  }
  public _GoodDayImplBase() {
  }
  // ... Stuff deleted
  public static boolean _execute(vbj.simplegoodday.corba.GoodDay _self,
    int _method_id, org.omg.CORBA.portable.InputStream _input,
    org.omg.CORBA.portable.OutputStream _output) {
    switch(_method_id) {
    case 0: {
      java.lang.String _result = _self.hello();
      _output.write_string(_result);
      return false;
    }
    }
    throw new org.omg.CORBA.MARSHAL();
  }
}
```

Implementing CORBA Applications

```java
// _st_GoodDay.java
package simplegoodday;
public class _st_GoodDay extends
    com.inprise.vbroker.CORBA.portable.ObjectImpl implements simplegoodday.GoodDay {
  protected vbj.simplegoodday.corba.GoodDay _wrapper = null;
  public vbj.simplegoodday.corba.GoodDay _this() {
    return this;
  }
  public java.lang.String hello() {
    org.omg.CORBA.portable.OutputStream _output;
    org.omg.CORBA.portable.InputStream _input;
    java.lang.String _result;
    while(true) {
      _output = this._request("hello", true);
      try {
        _input = this._invoke(_output, null);
        _result = _input.read_string();
      }
      catch(org.omg.CORBA.TRANSIENT _exception) {
        continue;
      }
      break;
    }
    return _result;
  }
```

Implementing CORBA Applications

```
// GoodDayHelper.java
package simplegoodday;
abstract public class GoodDayHelper {
  public static simplegoodday.GoodDay narrow(org.omg.CORBA.Object object) {
    return narrow(object, false);
  }
  private static simplegoodday.GoodDay narrow(org.omg.CORBA.Object object, boolean is_a) {

    // implementation deleted

  }
  public static vbj.simplegoodday.corba.GoodDay bind(org.omg.CORBA.ORB orb) {
    return bind(orb, null, null, null);
  }
  public static simplegoodday.GoodDay bind(org.omg.CORBA.ORB orb, java.lang.String name) {
    return bind(orb, name, null, null);
  }
  public static simplegoodday.GoodDay bind(org.omg.CORBA.ORB orb, java.lang.String name,
    java.lang.String host, org.omg.CORBA.BindOptions options) {

    // implmenetation deleted

  }
}
```

Implementing CORBA Applications

```
package simplegoodday;
final public class GoodDayHolder implements org.omg.CORBA.portable.Streamable {
  public simplegoodday.GoodDay value;
  public GoodDayHolder() {
  }
  public GoodDayHolder(simplegoodday.GoodDay value) {
    this.value = value;
  }
  public void _read(org.omg.CORBA.portable.InputStream input) {
    value = simplegoodday.GoodDayHelper.read(input);
  }
  public void _write(org.omg.CORBA.portable.OutputStream output) {
    simplegoodday.GoodDayHelper.write(output, value);
  }
  public org.omg.CORBA.TypeCode _type() {
    return simplegoodday.GoodDayHelper.type();
  }
}
```

Implementing CORBA Applications

What we have to do:

- Write an implementation of hello() extending _GoodDayImplBase.
- Write a server instantiating this implementation.
- Write a client to call the server.

Implementing CORBA Applications

```
// GoodDayImpl.java
package simplegoodday;
import java.util.Date;
import org.omg.CORBA.*;
import simplegoodday.*;
public class GoodDayImpl extends _GoodDayImplBase {

    private String location;

    public GoodDayImpl( String location ) {
        super();
        // initialize location
        this.location = location;
    }

    // hello method
    public String hello() {
        return "Hello World from " + location + "!";
    }

}
```

Implementing CORBA Applications

```
// GoodDayServer.java
package simplegoodday;
import java.io.*;
import org.omg.CORBA.*;
import simplegoodday.*;
public class GoodDayServer {
    public static void main(String[] args) {
        try {
            ORB orb=ORB.init(args, null);                      // init ORB
            GoodDayImpl goodDayImpl=new GoodDayImpl(args[0]); // create a GoodDay Object;
            orb.connect(goodDayImpl);                // export the object reference
            System.out.println(orb.object_to_string(goodDayImpl));
                                                     // print stringified object reference
            java.lang.Object sync = new java.lang.Object();   // wait for requests
            synchronized (sync) {
                sync.wait();
            }
        } catch (Exception e) {
            System.err.println(e);
        }

    }
}
```

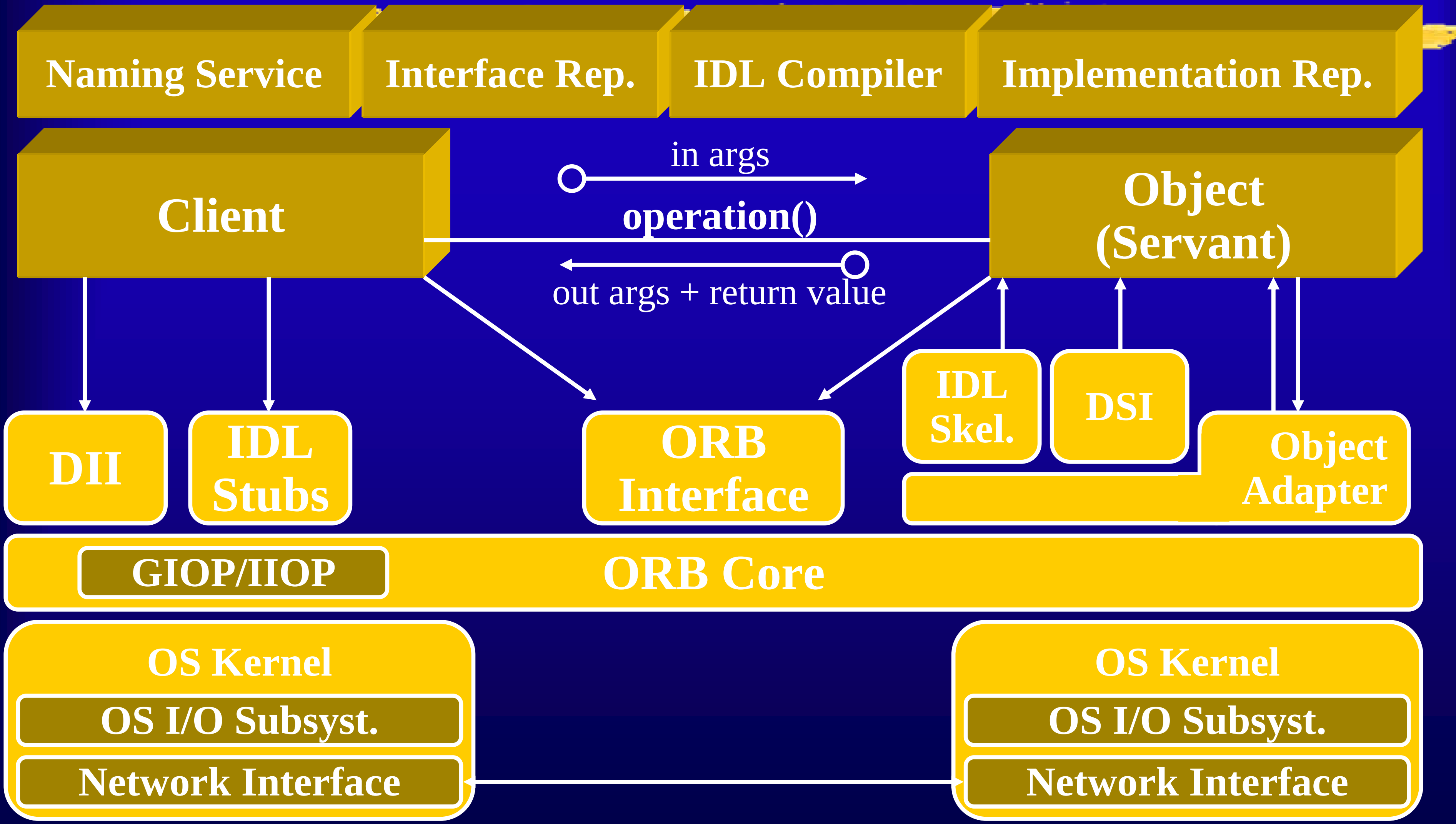
Implementing CORBA Applications

```
// GoodDayClient.java
package simplegoodday;
import java.io.*;
import org.omg.CORBA.*;
import simplegoodday.*;
public class GoodDayClient {
    public static void main(String args[]) {
        try {
            ORB orb=ORB.init(args, null);                         //init orb
            org.omg.CORBA.Object obj=orb.string_to_object(args[0]);
                                        // get object reference from command line
            GoodDay goodDay = GoodDayHelper.narrow(obj);
                                        // cast the IIOR to a specific interface
            if (goodDay==null) {        // check for errors
                System.err.println("stringified object reference" +
                                                     "is of wrong type");
                System.exit(-1);
            }
            System.out.println(goodDay.hello());  // call remote method
        } catch (SystemException ex) {
            System.err.println(ex);
        }
    }
}
```

What’s missing here?

- Dynamic Object Discovery
  - Naming Service
- Dynamic Object Instantiation
  - Basic and Portable Object Adapter
- Passing parameters
  - IDL: in, out, inout, oneway
- Classes must extend *ImplBase
  - tie approach
- Stubs/Skeletons must be present at Client/Server
  - Dynamic Invocation Interface (DII)
  - Dynamic Skeleton Interface (DSI)

CORBA ORB Architecture



CORBA Services

- Basic Services, needed by every OO application
- Standard Interfaces provide portability and interoperability for these basic functions
- Not all services implemented by all vendors
- Interoperability should be provided

COS

- Naming
- Events / Notification
- Life Cycle
- Persistent Object
- Relationship
- Externalization
- Transactions
- Concurrency Control
- Licensing
- Query
- Properties
- Security (incl. IIOP over SSL)
- Time
- Collections
- Trading

COS

- Lifecycle: creation and deletion of objects.
- Naming: mapping of convenient object names to references to actual objects.
- Event Notification: registration of required and expected notification of event passage.
- Persistence: long-term existence of objects, management of object storage.
- Relationships: representation and consistency management of relationships between objects.
- Externalization: ability to store object representation on removable media and allow re-internalization later.

COS

- Transactions: merger of OLTP and distributed objects.
- Concurrency Control: management of concurrent execution in distributed environment.
- Security: framework for many underlying security technologies.
- Properties: assign properties to objects.
- Query: common query interface to objects.
- Licensing: license management.
- Trading: trading of different object references by attributes

Evaluation

- Learning curve
- Interoperability
- Portability
- Feature Limitations
- Performance

Learning Curve

- CORBA introduces the following:
  - New concepts (e.g. IOR, stubs, object adapters)
  - New components and tools (e.g. IDL compiler, ORB, implementation rep.)
  - New features (e.g. exception handling, inheritance)
- Time spent learning this must be amortized over many projects

Interoperability

- CORBA 1 was woefully incomplete with respect to interoperability
- CORBA 2.x defines a useful interoperability specification
  - later extensions deal with portability issues for server-side (i.e. the POA spec)
- Most ORB implementations now support IIOP or GIOP robustly
  - but not all higher services are interoperable

Portability

- To improve portability, the latest CORBA specification standardizes
  - IDL language mappings (e.g. C, C++, Java)
  - Naming service, event service, lifecycle service
  - ORB initialization service
  - Portable Object Adapter API
  - Servant mapping
- Porting applications from ORB-to-ORB will be limited, however, until conformance tests become common-place
  - `http://www.opengroup.org/testing/testsuites/vsorb.htm`

Feature Limitations

- Standard CORBA doesn’t yet address key “inherent” complexities of distributed computing, e.g.
  - latency
  - fault tolerance (RPF is underway on this)
  - causal ordering
  - deadlock

Feature Limitations

- Most ORBs do not support passing objects by value
  - Solution with CORBA 2.3/3.0
- Most ORBs still support only the following semantics:
  - ORs are passed by reference
  - structs and unions are passed by value
  - objects by value must be hand-crafted using “factories”

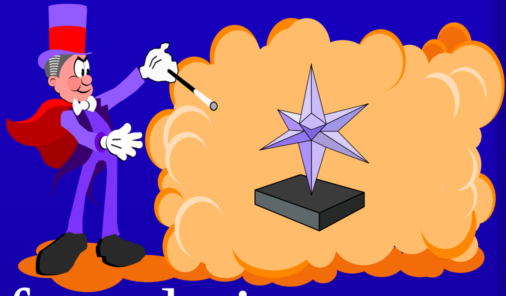
Feature Limitations

- Most ORBs do not yet support asynchronous method invocation or timeouts
- Versioning is supported in IDL via pragmas
  - not language inherent like in ONC RPC or DCOM

Performance Limitations

- Performance may not be as good as hand-crafted code for some applications due to
  - additional remote invocations for naming
  - marshaling/demarshaling overhead
  - data copying and memory management
  - endpoint and request demultiplexing
  - context switching and synchronization overhead
  - Trade off between performance and extensibility, robustness, maintainablility

For Your Developers



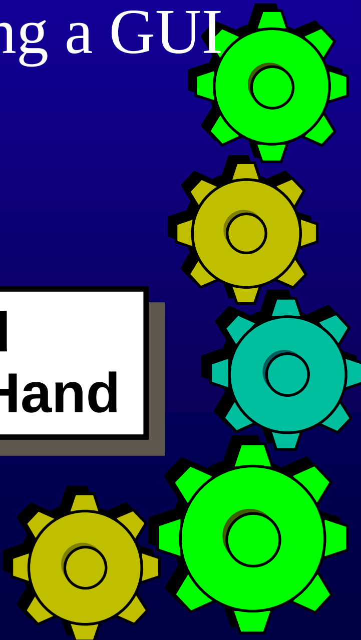
- Much more than Client-Server
- CORBA provides a sophisticated base
- CORBA Services provide necessary OO foundation
- CORBA Facilities will standardize building blocks
- Developers create or assemble Application Objects

Object Services and Common Facilities accessed via standard OMG IDL Interfaces

Develop Clients and Servers Independently using the Best Tools for Each Task

For Your Users

- Purchase Server Objects from Multiple Vendors and Integrate Under One or More Client Applications
- Seamlessly Integrate In-House and Purchased Objects
- Acquire & Maintain a Single Set of Business Objects Accessed by the Entire Enterprise
- Each Division Accesses These Common Objects Using a GUI Built
for its Own Needs



APPLICATION: A Set of Clients and Servers Activated and Connected at Run Time to Attack the Problem at Hand

CORBA Implementations

- Many ORBs available
  - Orbix from IONA
  - Visibroker from Inprise
  - BEA Web Logic Enterprise
  - Component Broker from IBM
  - CORBAplus from Expertsoft
  - ORB Express
  - Open Source ORBs (TAO, ORBacus, omniORB, MICO, ...)
- In *theory* CORBA facilitates vendor-and platform independent application collaboration
- In *practice* interoperability and portability still an issue

CORBA 3

- Improved Java and Internet Integration
  - Java-to-IDL reverse mapping
  - Firewall specification
  - CORBA Object URLs
- Quality of Service Control
  - Asynchronous Invocation/Messaging
  - Invocation QoS Control
  - Realtime, Minimum, Fault Tolerance
- CORBA Component Model
  - Objects passed by value
  - Component Container
    - Transactional, Persistent, Secure
  - Distribution Format
  - Scripting Language Specification

Summary of CORBA Features

- Object Request Broker (ORB)
- Interface Definition Language (IDL)
- Language Mappings (C, C++, COBOL, Java)
- Static and Dynamic Invocation Interfaces
- Static and Dynamic Skeleton Interfaces
- Interface and Implementation Repositories
- Basic and Portable Object Adapter
- CORBA Services

More Information

www.omg.org