OMG

OMG(Object Management Group)

-Established in 1989 for vendor independent specifications for software industry(1997:700 members)

-Developing standard interfaces for Distributed Object Computing(i.e. CORBA/IIOP, Object Services, Internet Facilities, Domain Interface Specification)

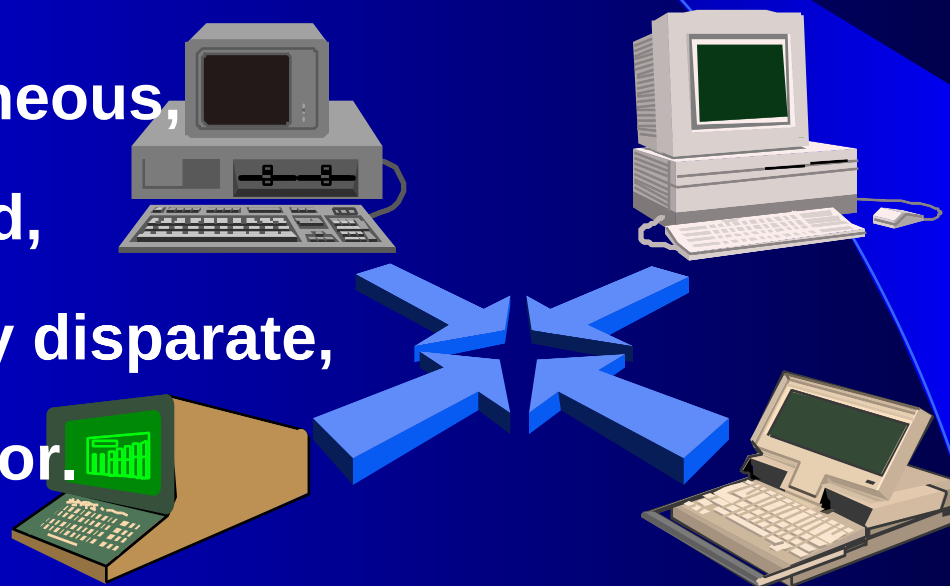
-Creating a component-based software marketplace across major platforms and operating systems

Source: http://www.omg.org/omg00/backgrnd.htm

OMG's View on the Problem in Sftware Engineering(1)

Constructing information-sharing distributed systems from diverse sources:

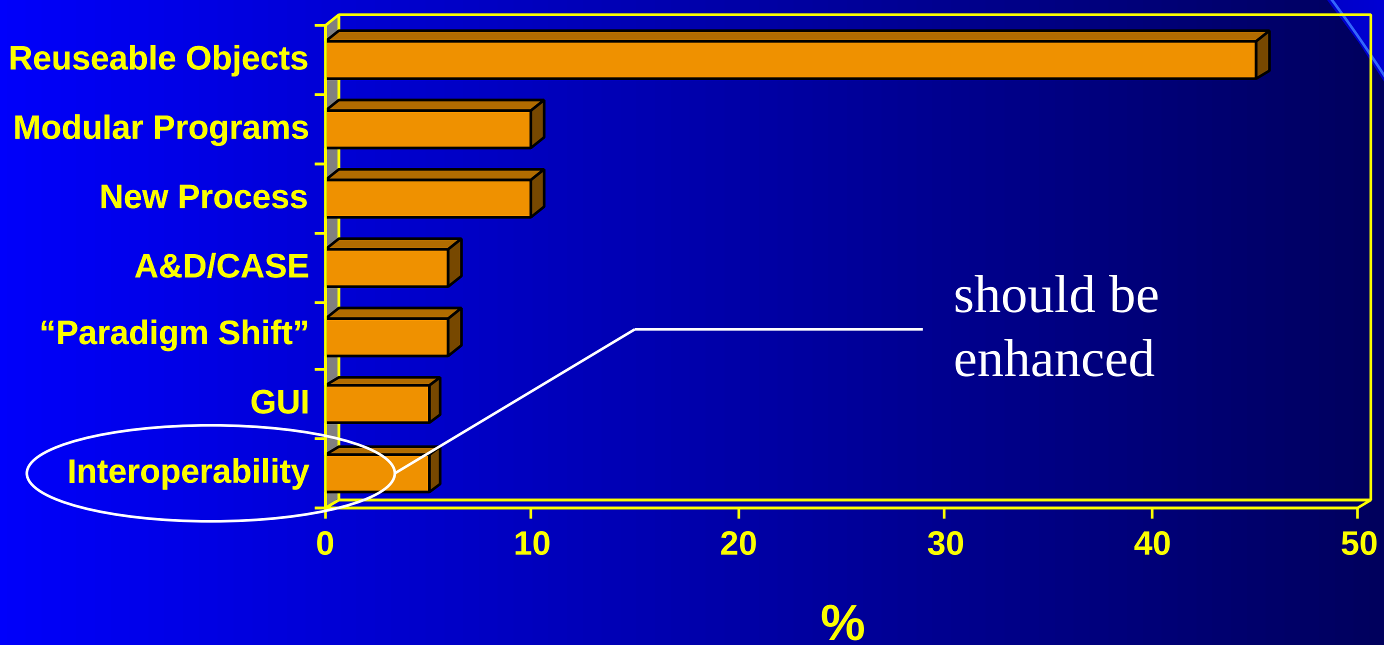
- ✍**heterogeneous,**
- ✍**networked,**
- ✍**physically disparate,**
- **-multi-vendor.**



Source: Richard Mark Soley, Creating Industry Consensusftp: ftp://ftp.omg.org/pub/presentations/consen.ppt

OMG's View on the Problem in Software Engineering(2)

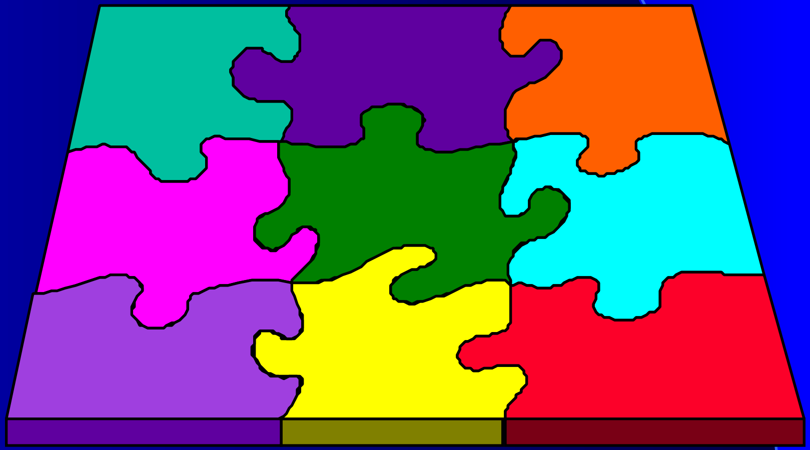
IT Professionals rate reusability as the major advantage of Object-Oriented Programming



Source: Richard Mark Soley, The Software Crisis: ftp://ftp.omg.org/pub/presentations/crisis.ppt

OMG's View on Solution(1)

- **The key isn't just technology, but integration**
- **Software should be built in small, reuseable, maintainable components with standardized, clearly-defined interfaces.**



Source: Richard Mark Soley, The Software Crisis: ftp://ftp.omg.org/pub/presentations/crisis.ppt

OMG's View on Solution(2)

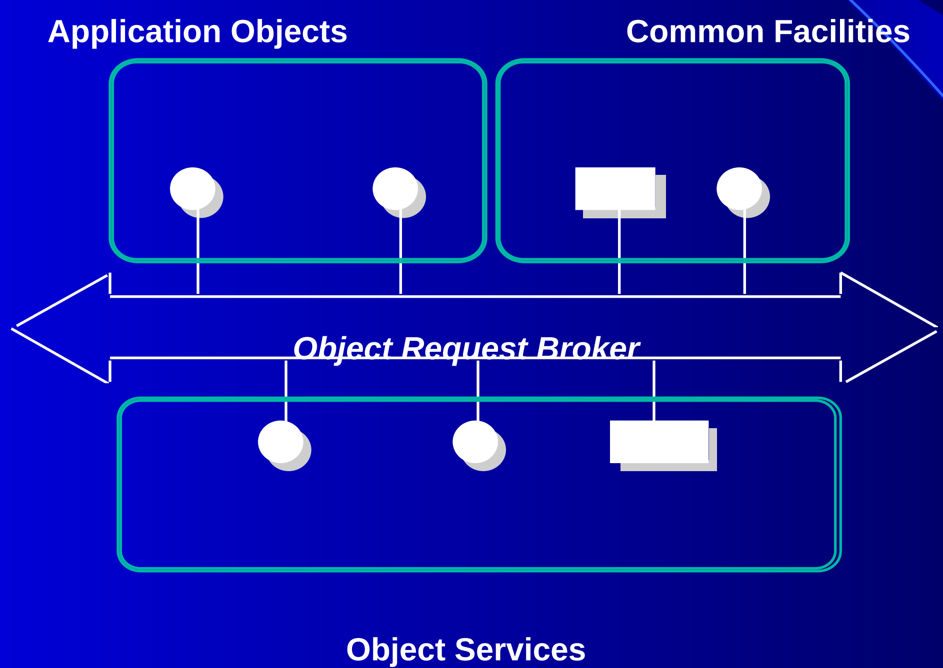
Develop a single architecture, using object technology, for distributed application integration for:

- Interoperability
- Portability
- Reusability

of object-based software in distributed and heterogeneous environment

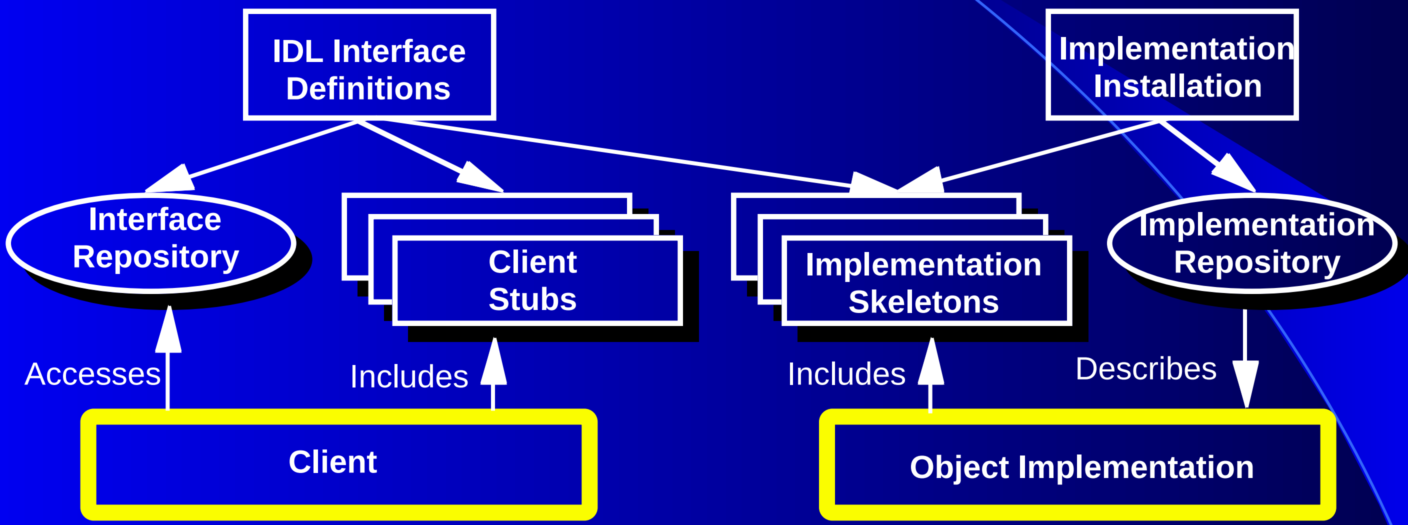
Source: Richard Mark Soley, Creating Industry Consensusftp: ftp://ftp.omg.org/pub/presentations/consen.ppt

CORBA(Common Object Broker Architecture)



Source: Richard Mark Soley, Creating Industry Consensusftp: ftp://ftp.omg.org/pub/presentations/consen.ppt

CORBA Interfaces



Source: Douglas C. Schmidt, A Tour of CORBA: ftp://ftp.omg.org/pub/presentations/corba.ppt

CORBA Components

Client stub:Each stub represents an object operation (a possible request) which a client invokes in a language-dependent manner (e.g., by calling a subroutine which represents the operation).

Dynamic Invocation:Alternatively, a client may dynamically construct and invoke request objects which can represent any object operation.

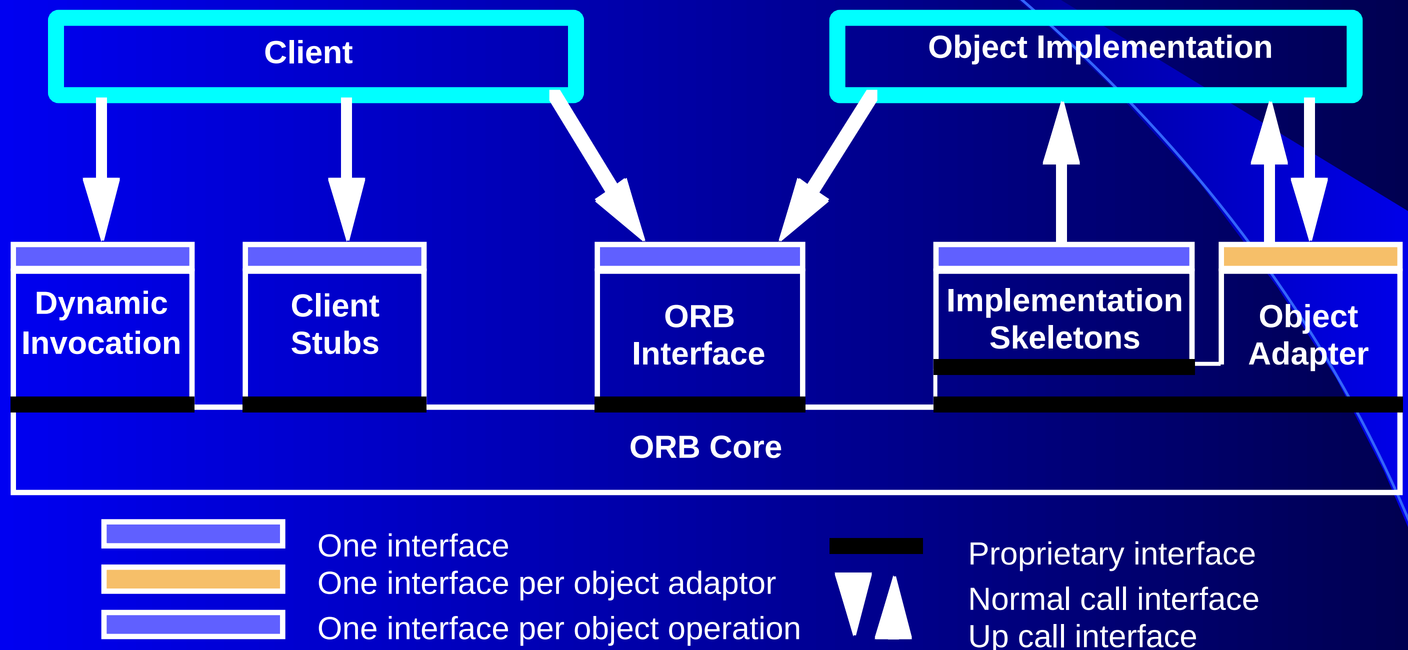
Implementation Skeleton:Each skeleton provides the interface through which a method receives a request.

Object Adapter:Each object adapter provides access to those services of an ORB (such as activation, deactivation, object creation, object reference management) used by a particular ilk of object implementation.

ORB Interface:The interface to the small set of ORB operations common to all objects, e.g., the operation which returns an object's interface type.

Source: Douglas C. Schmidt, A Tour of CORBA: ftp://ftp.omg.org/pub/presentations/corba.ppt

CORBA Components



Source: Douglas C. Schmidt, A Tour of CORBA: ftp://ftp.omg.org/pub/presentations/corba.ppt

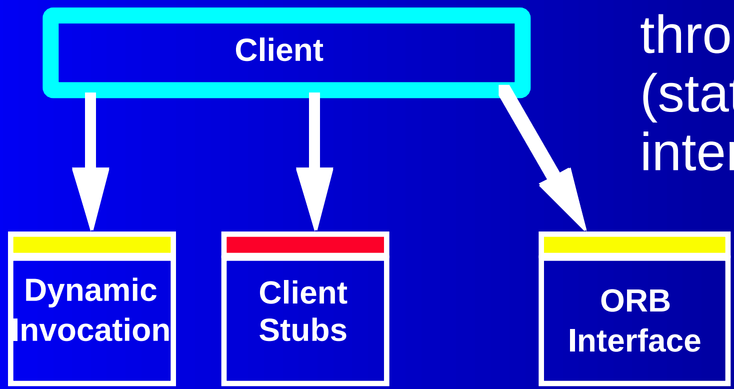
Client Side

Clients perform requests using object reference

Clients may issue requests through object interface stubs (static) or dynamic invocation interface.

Clients may access general ORB servi

- Interface Repository.
- Context Management.
- List Management.
- Request Management.



Source: Douglas C. Schmidt, A Tour of CORBA: ftp://ftp.omg.org/pub/presentations/corba.ppt

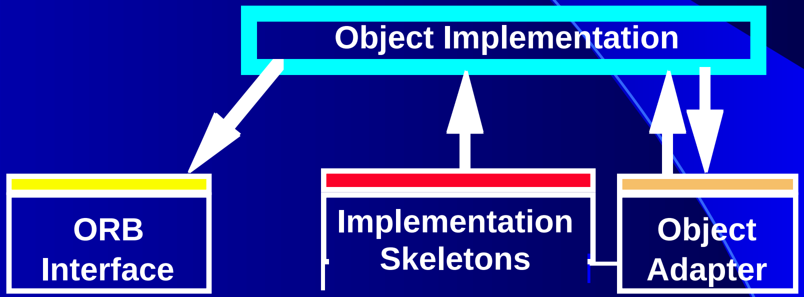
Implementation Side

Implementations receive requests through skeletons (without knowledge of invocation approach).

The Object Adapter adapts to vagaries of object implementation scheme.

The Basic Object Adapter provides for:

- management of references;
- method invocation;
- authentication;
- implementation registration;
- activation/deactivation.



Source: Douglas C. Schmidt, A Tour of CORBA: ftp://ftp.omg.org/pub/presentations/corba.ppt

CORBA IDL

Key component of the standard is **object oriented Interface Definition Language (IDL):**

- mappings will be provided for many languages/compilers;
- used to specify interface containing methods and attributes
- multiple-inheritance, public interface-structured specification language(e,g. C, C++, Smalltalk, COBOL, Modular 3, DCE etc.;
- not for implementation.
- **primary support for interoperability between static and dynamic requests mechanisms.**

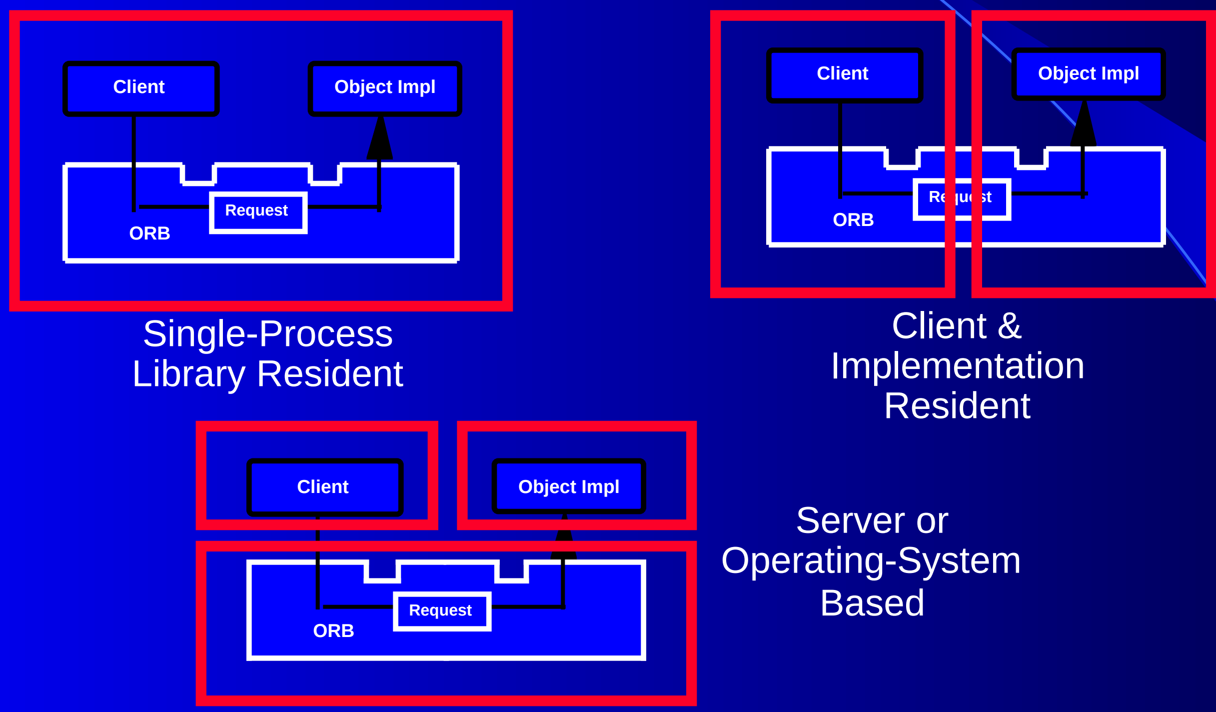
Source: Douglas C. Schmidt, A Tour of CORBA: ftp://ftp.omg.org/pub/presentations/corba.ppt

Examples ORB's

Client-and implementation-resident	ORB implemented as libraries (routines) resident in the clients and in the implementations.
Library-resident	ORB and implementations implemented as libraries (routines) resident in the client.
Server-based	ORB is implemented as a server (separate process) which brokers requests between client and implementation processes.
System-based	ORB is part of the operating system.

Source: Douglas C. Schmidt, A Tour of CORBA: ftp://ftp.omg.org/pub/presentations/corba.ppt

ORB Types



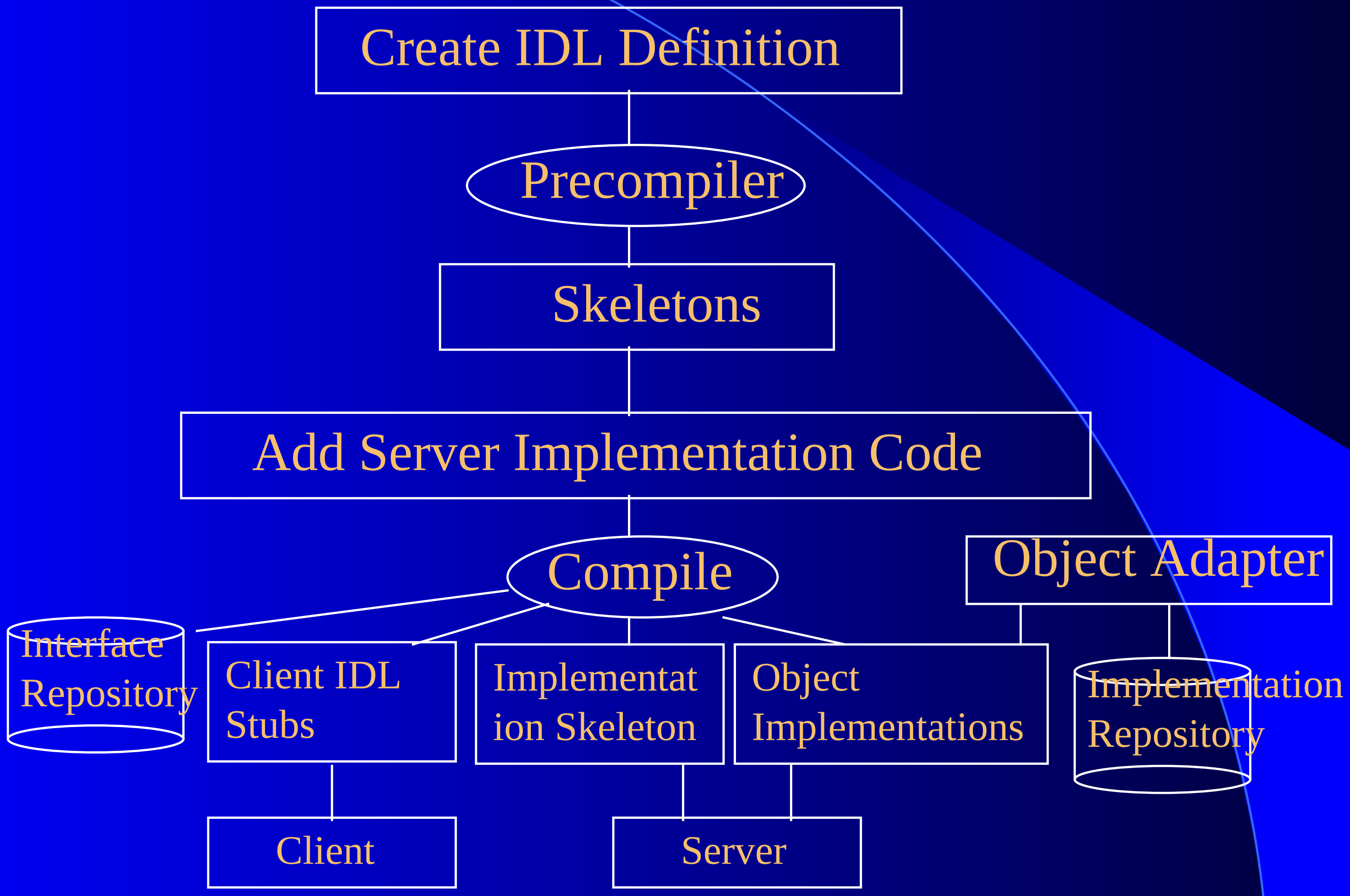
Source: Douglas C. Schmidt, A Tour of CORBA: ftp://ftp.omg.org/pub/presentations/corba.ppt

Example Adapters

Basic Object Adapter	Intended for implementations that are separate programs (processes) with no “ORB-like” services, the basic adapter provides for object reference generation and management, method invocation and request delivery, implementation registration, activation and deactivation.
OODB Adapter	As OODB’s provide some "ORB-like" services (e.g., object reference generation and management), this adapter is tuned to integrate OODB’s with ORB distribution and communication.
Library Adapter	Tuned for implementations resident in the client's process space, this adapter provides minimal implementation management and high-performance data transfer.

Source: Douglas C. Schmidt, A Tour of CORBA: ftp://ftp.omg.org/pub/presentations/corba.ppt

CORBA Static Method Invocation



Dynamic Invocation Interface

The Dynamic Invocation Interface (DII) allows clients to dynamically:

- **discover objects;**
- **discover objects' interfaces;**
- **create requests;**
- **invoke requests;**
- **receive responses.**

Major features of Dynamic Invocation Interface:

- **requests appear as objects themselves;**
- **requests are reusable;**
- **invocation may be synchronous or asynchronous; requests may be generated dynamically, statically or in combination approach.**

Source: Douglas C. Schmidt, A Tour of CORBA: ftp://ftp.omg.org/pub/presentations/corba.ppt

COSS(Common Object Service Specification)

Initial Object Services targets: (adoption 3Q93)

- Lifecycle: creation and deletion of objects.
- Persistence: long-term existence of objects, management of object storage.
- Naming: mapping of convenient object names to references to actual objects.
- Event Notification: registration of required and expected notification of event passage.

Future Object Services targets (underway): (2Q94)

- Transactions, Concurrency, Relationships, Externalization, Internationalization, Time.

COSS Key Features(1)

Naming Service

-Provides the ability to bind a name to an object relative to a naming context. A naming context is an object that contains a set of name bindings in which each name is unique

Event Service

-Provides basic capabilities taht can be configured together in a very flexible manner

-Both push and pull event delivery models are supported

-Event channel interface can be subtyped to support extended capabilities

-No extention is required to CORBA for Event Service

Source: http://www.omg.org/corba/sectrans.htm

COSS Key Features(2)

Life Cycle Service

-Defines conventions for creating, deleting,copying and moving objects

-Defines services and conventions that allow clients to perform life cycle operations on objects in different locations

-Defines an interface for a generic factory(factory object creates another object)

-Defines LifeCycleObject interface handling remove,copy and move operations

Persistent Object Service

-Provides a set of common interfaces to the mechanism used for retaining and managing persistent state objects

-Provide openness to the different clients and implementations (e.g. different strorage mechanism requirement in mobile computer and mainframs)

Source: http://www.omg.org/corba/sectrans.htm

COSS Key Features(3)

Transaction Service

-Supports multiple transaction models, including flat and nested models

-Supports interoperability between different programming models(e.g. an object and procedural code can share a sigle transaction)

-Supports both system-managed transaction propogation and application-managed propogation

-Supports multiple,concurrent transactions

Concurrency Control Service

-Enables multiple clients to coordinate their access to shared resources to maintain that resource in a consistent state

-Uses “lock” regulation strategy

-Support variety of lock modes as well as Intention Locks that support locking at multiple levels of granularity

Source: http://www.omg.org/corba/sectrans.htm

COSS Key Features(4)

Relationship Service

-Allows entities and relationships to be explicitly represented
-Role represents CORBA object in a relationship
-Relationship interface can be extended to add relationship specific attributes and operations
-Navigation of relationship can be a local operation in distributed implementation of the service

Externalization Service

-Defines protocols for externalizing(record the object state in memory or disk file) and internalizing(enter into a new object in the same or different process)
-Clients can request externalized data be stored in a file with the format specified in this service objects

Source: http://www.omg.org/corba/sectrans.htm

COSS Key Features(5)

Query Service

-Allows users and objects to invoke queries on collections of other objects
-Allows indexing and is based on existing standards for query(e.g. SQL-92, OQL-93, and OQL-93 Basic)
-Provides an architecture for nested and federated service that can coordinate multiple, nested query evaluators

Source: http://www.omg.org/corba/sectrans.htm

Lisensing Service

-Provides a mechanism for products to control the use of their intellectual property
-*Time* attribute allows licenses to have start/duration and expiration dates
-*Value mapping* allows producers to implement a licensing scheme according to units, allocation, or consumption
-*Consumer* attribute allows a license to be reserved or assigned for specific entities

COSS Key Features(7)

Property Service

-Provides the ability to dynamically associate named values with objects outside the static IDL-type system

-Defines operations to create and manipulate sets of name-value pairs or name-value-mode tuples

-Provides “batch” operations to deal with set of properties as a whole

-Provides client access and control of constraints and property modes

Time Service

-Enables the user to obtain current time together with an error estimate associated with it

-Ascertains the order in which “events” occured

-Generates time-based events based on timers and alarms

-Computes the interval between two events

-*TimeService* interface & *TimerEventService* interface

Source: http://www.omg.org/corba/sectrans.htm

COSS Key Features(8)

Security Service

-Identification and Authentication

-Authorization and Access Control

-Security Auditing to make users accountable for their security related actions

-Security of Communication between objects

-Nom-Repudiation provides irrefusable evidence of actions such as proof of origin of data to the recipient

Source: http://www.omg.org/corba/sectrans.htm

Adoption of COSS(1)

Ada Language Mapping:	19 Mar,1996
C++ Language Mapping:	06 Dec,1994
C++ Language Mapping 1.1	19 Mar,1996
COM/CORBA interworking	19 Mar,1996
CORBA 2.0	07 Dec,1994
CORBA Interoperability	30 Oct,1996
Common Secure IIOP	21 Nov,1996
Compound Presentation & Interchange	19 Mar,1996
Concurrency Service	06 Dec,1994
Event Notification Service	07 Dec,1993
Externalization Service	06 Dec,1994
IDL Fixed Point Extensions	20 Aug,1996
IDL Type Extensions	30 Oct,1996

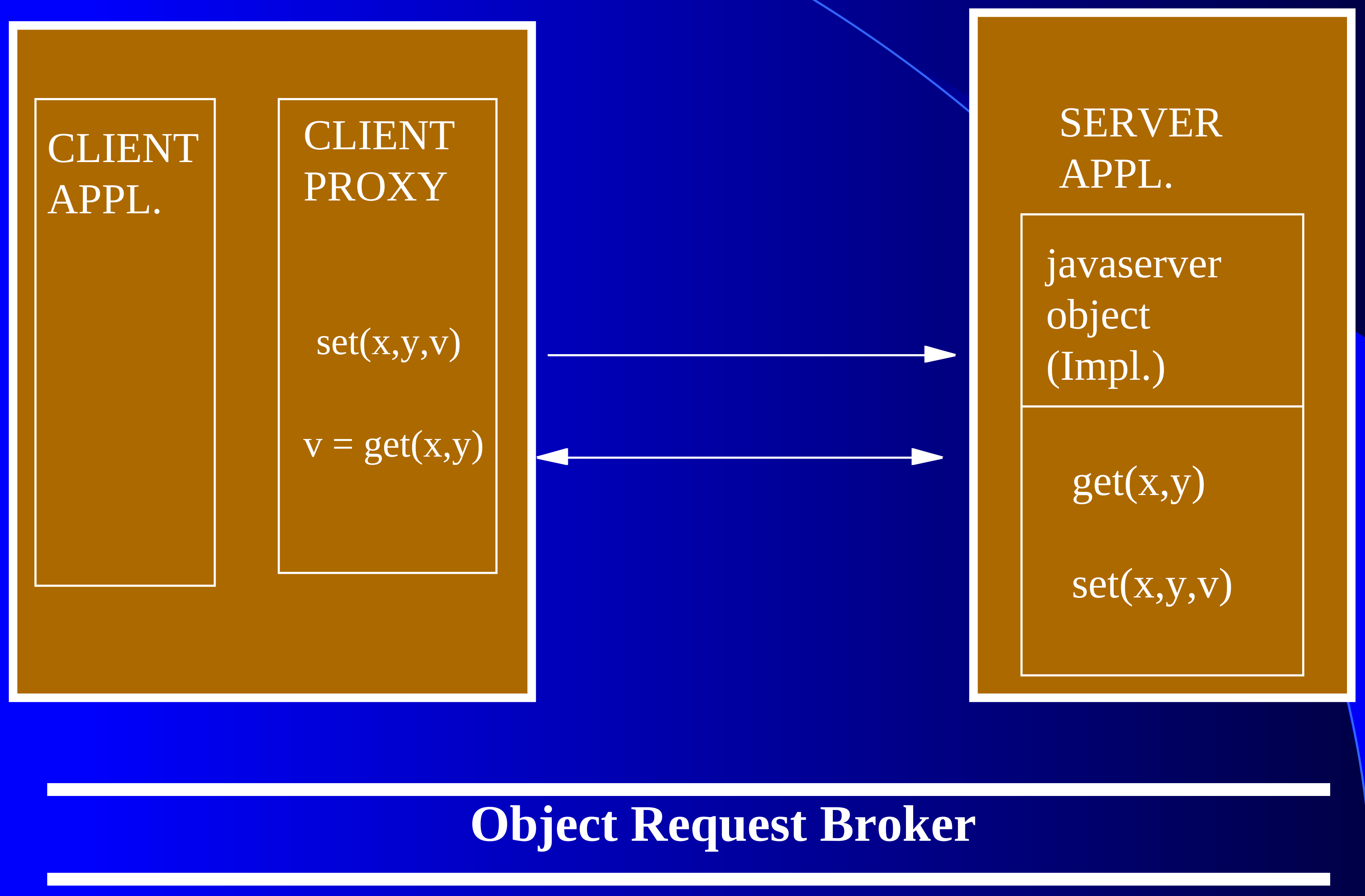
Adoption of COSS(2)

Licensing Service	10 Nov,1995
Lifecycle Service	07 Dec,1993
Naming Service	07 Dec,1993
Object Collections Service	30 Oct,1996
Object Query Service	28 Mar,1995
Object Security Service	21 Nov,1996
Object Trader Service	30 Oct,1996
Object Transaction Service	06 Dec,1994
Persistent Object Service	21 Apr,1994
Properties Service	10 Nov,1995
Relationship Service	06 Dec,1994
Security Service	19 Mar,1996
Smalltalk Language Mapping	28 Mar,1995
System Management Facility	21 Nov,1996
Time Service	19 Mar,1996

COSS Implementation

	SUN	DEC	IBM	HP	IONA	ExperSoft
Naming						ExtendedC++
Event						ExtendedC++
Life Cycle						ExtendedC++
Persistent						ExtendedC++
Transaction						
Concurrency						
Relationship						
Externalization						
Query						
Lisencing						
Property						
Time						ExtendedC++
Security		Kerberos/DCE				
Initialization						

A SAMPLE ***CORBA*** APPLICATION



Server-side OMG IDL Specification

// IDL Specification

interface grid {

readonly attribute short height;

readonly attribute short width;

void set(in short n, in short m, in long value);

long get(in short n, in short m);
}

Creating Server-Side Implementations

- Running the Grid Interface definition through the IDL Compiler generates a *client* stub and a *server* skeleton

  - The Client stub acts as a proxy and handles :
    object binding
    parameter marshalling

  - The server skeleton handles :
    object registration,
    activation,
    parameter demarshalling

Writing the Server Side Method Definitions

Server Side Method Implementations

```
public int get(short n, short m) {
  return (short) m_a[n][m];
}

public void set(short n, short m, int value) {
m_a[n][m] = value;
}

public short get_height() {
return (short)m_height;
}

public short get_width() {
return (short) m_width;
}
```

Writing the Main Server Program

The main program for the GridServer looks like :

```
public class javaserver1 {
 public static void main(String args[]) {
   .......
   try {
     .....
    _CORBA.Orbix.impl_is_ready("GrdSrv",0);
    _CORBA.Orbix.processEvents(-1);
    }
    catch(SystemException se) {
    ......
    System.exit(1);
    }
```

Object Activation

- If the service isn't running when a client invokes a method on an object it manages, the ORB will automatically start the service
- Services must be registered with the ORB, e.g.

  %putit -j TestSrv testServer.javaserver1
- Clients may bind to a service by using a location broker or by explicitly naming the server

Binding a Client to a Target Object

- Steps for binding a client to a target object

1. A CORBA client obtains an ‘object reference’ from a server
   - May use a *Name Service* or a *Locator Service*

2. This object reference serves as a local proxy for the remote target object
   - Object references may be passed as parameters to other remote objects

3. The Client may then invoke methods on its proxy

Client-Side Example

The main program for the GridServer looks like :

```
public class GridEvents extends GridPanel {
  public _gridRef   gRef;       // grid proxy object

 gRef = grid._bind(markerServer,hostName);
    .......
  cellVal = gRef.get(x,y);
    .......
  gRef.set(x,y,cellVal);
    ......

}
```

1996 : The Year of the ORB ?

"..1996 will be the year of the ORB. If not, it will be early 1997. If it does not happen by then, it's good-bye CORBA. It means that OLE would have won the battle of the ORB"

Robert Orfali : *The Essential Distributed Objects Survival Guide*

CORBA : Bad News

- Commercial ORBs are slow and inefficient
  - no concurrency control, no garbage collection, no load-balancing
- No MOM
  - OMG Common Facilities Task Force is currently working on it
- Server Code not portable
- IDL needs to support semantic-level extensions
- Standard CORBA does not support Meta-Classes

CORBA Event Services

- In CORBA, a standard method invocation results in synchronous execution of an operation provided by an object

  - Both requestor (client) and provider (server) must be present

- For many communications, a more *highly decoupled communication* model between objects is required

  - i.e. asynchronous communication with multiple suppliers and consumers

- Degree of Coupling : RPC, CORBA, MOM

The CORBA Event Service

- Allows objects to dynamically register or unregister their interest in specific events

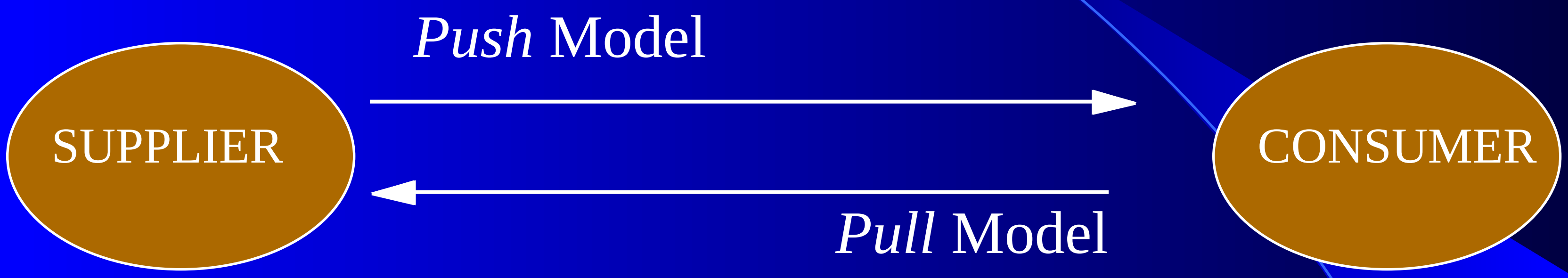
- *Event :* An Occurence within an object specified to be of interest to one or more objects

- *Notification* : A message an object sends to interested parties informing them that a specific event occurred

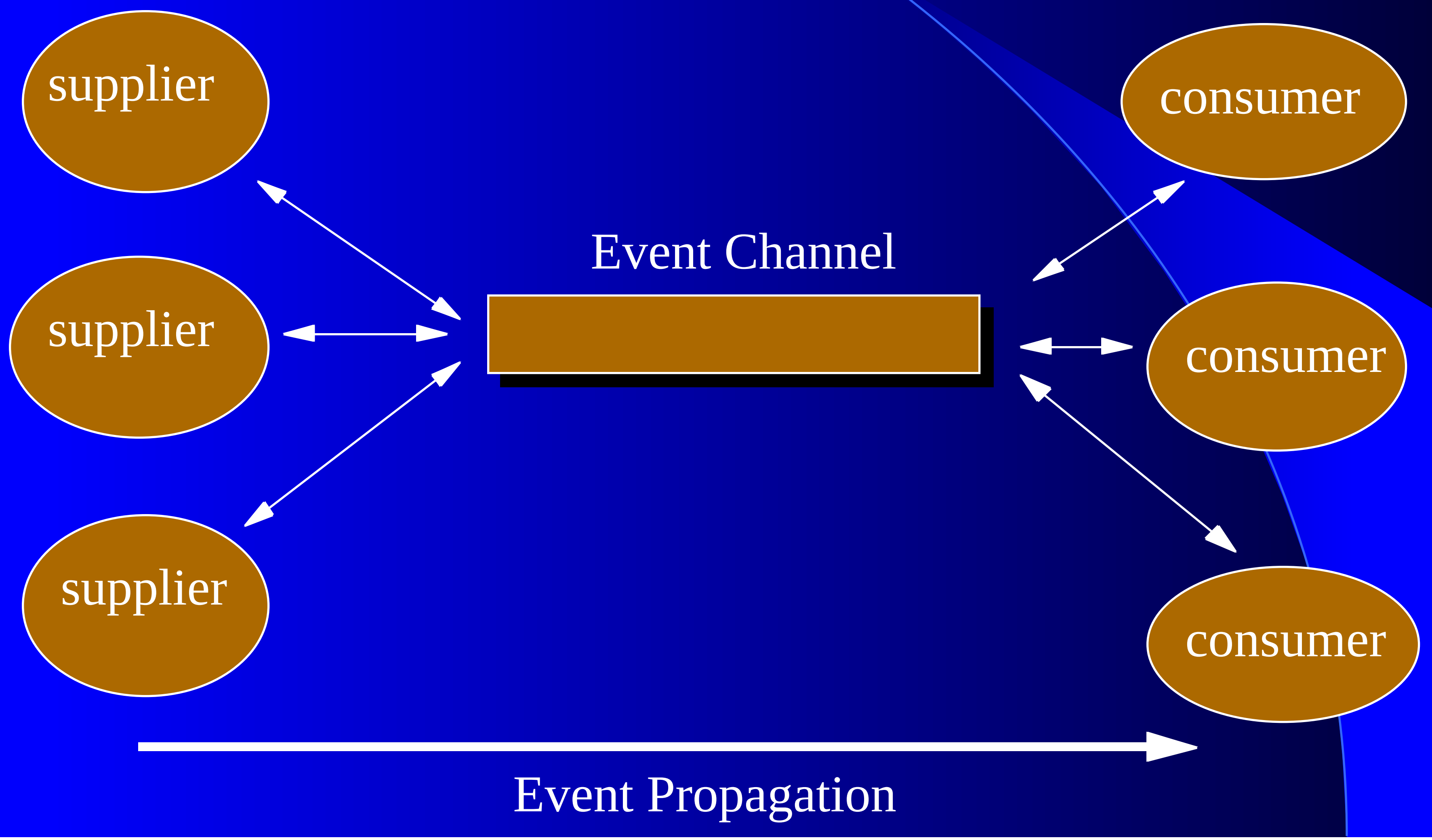
- Event Notification Models : *Pull vs Push Model*
  - Who Takes the Initiative ?

- *Suppliers*, *Consumers*, *Event Channels*

Direct Event-Style Communication



CORBA Event Architecture



Cosumers and Suppliers

- The OMG event service defines two roles for objects :

1. The *Supplier* Role
   - Suppliers generate event data

2. The *Consumer* Role
   - Consumers process event data

- Event data are communicated between suppliers and consumers by issuing standard CORBA requests

The Event Channel

- Event channels are standard CORBA objects, and communication with an event channel is accomplished using standard CORBA requests

- Event Channel Semantics

  - ***Generic*** versus ***Typed*** Event Communication

- Event Channel Levels of Service

  - presistent store, buffer ?

Generic and Typed Event Communication

• There are 2 orthogonal approaches that OMG event-based communication may take :

1. ***Generic***

- All communication is by means of generic ***push*** or ***pull*** operations

- These operations involve single parameters or return values that package all the events into a generic **corba** ***any*** data structure

2. ***Typed***

- In the typed case, communication is via operations defined in IDL

- Event data is passed by means of typed parameters, which can be a powerful means of filtering event information.

The OMG CORBA Event Services Specification

- Service Qualities and Semantics for the Event Channel Specification
  - Fast Sender and Slow Receivers ?

- A minimalist form of MOM Communications into CORBA
  - no message priorities,
    filters,
    transaction protection,
    reception confirmation
    Time-to-Live Stamps
    Sophisticated Queue Management

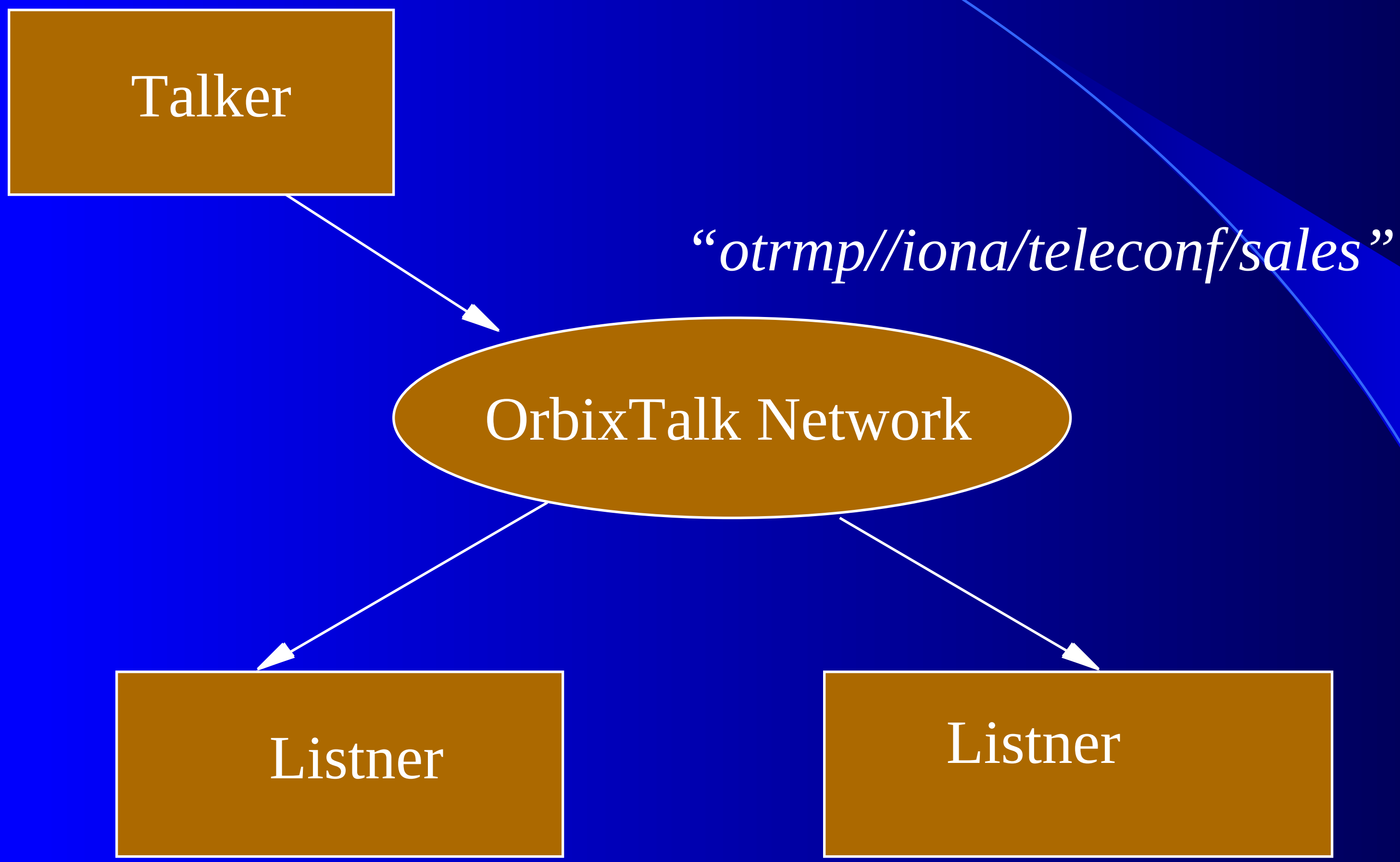
OrbixTalk

- OrbixTalk provides the first major extension of CORBA that allow clients to communicate with application objects using messages
- Asynchronous, Decoupled Messaging, and Events
- Event-Driven Systems

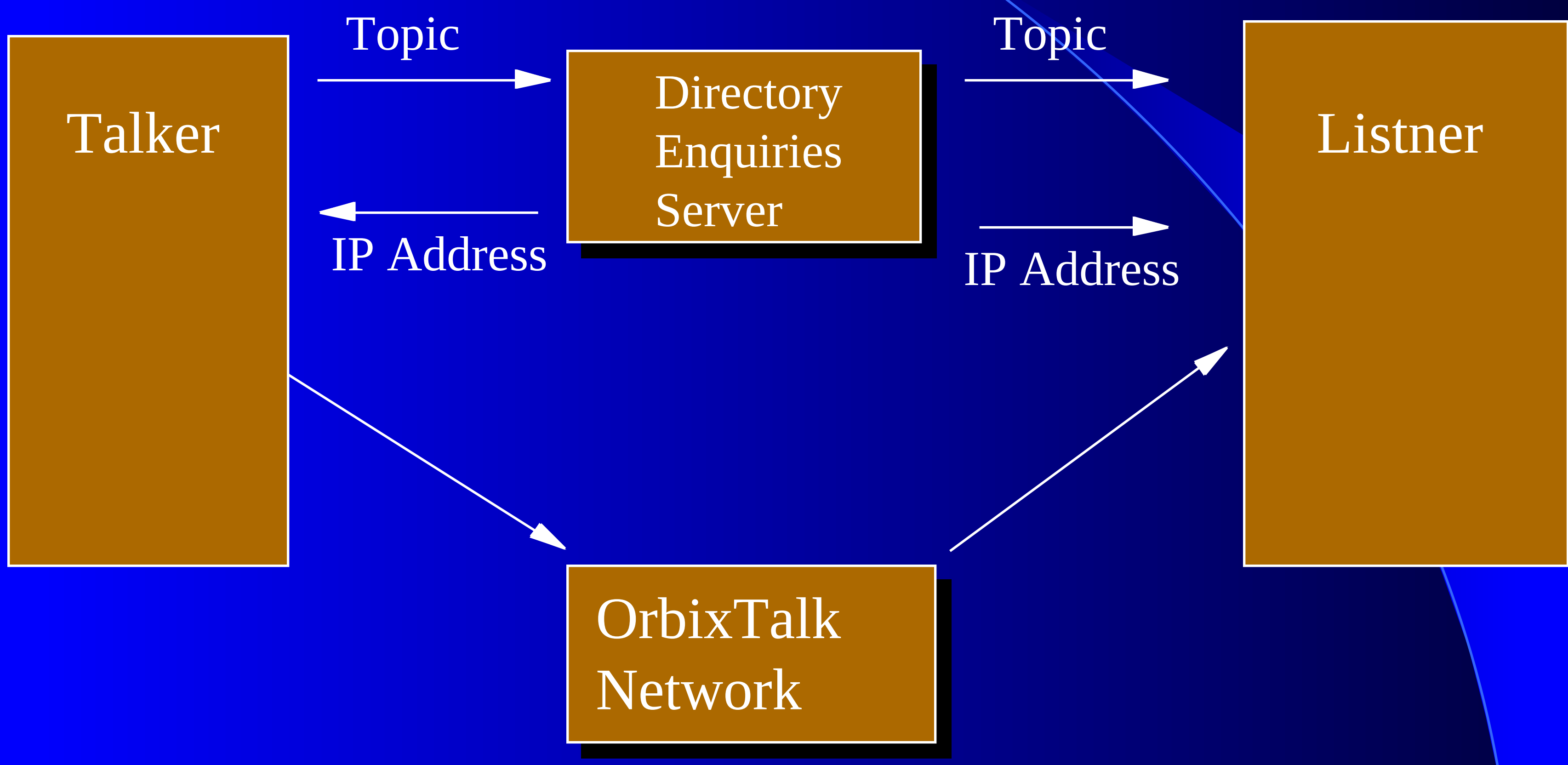
OrbixTalk

- Supports reliable and guaranteed messaging semantics via its Messagestore persistence technology
- Programming Model based on : *Talker*, *Listners* and *Topics*

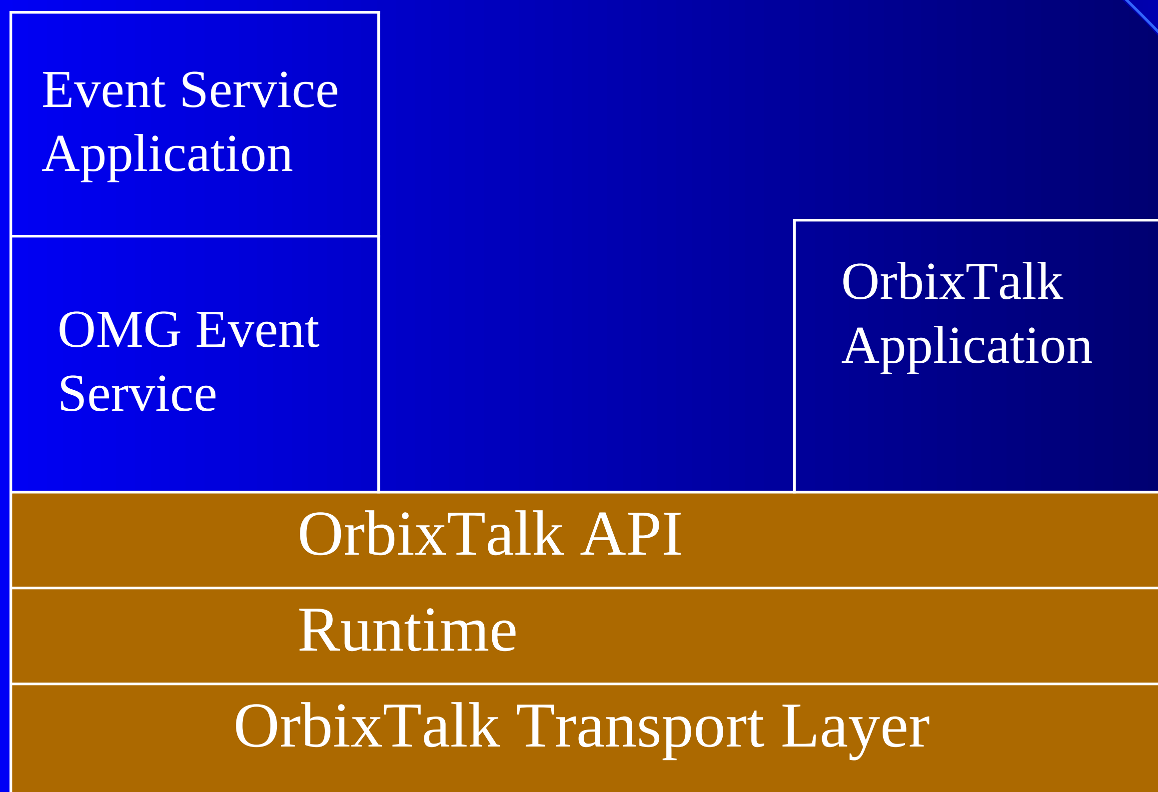
Talker and Listner OrbixTalk Applications



OrbixTalk Architecture



The OrbixTalk Architecture



Programming with OrbixTalk

- OrbixTalk Talkers quote stock prices and OrbixTalk Listners listen for stock price quotes

// IDL

```
interface StockPrice {
  oneway void quote(in float f);
}
```

- *oneway :* Since talker objects may invoke only oneway operations

Listner and Talker Application Structure

- The Listner Application :

  1. Creates StockPrice Objects
  2. Registers them as listners
  3. Waits to Receive Messages

- The Talker Application :

  1. Creates StockPrice Objects
  2. Registers them as Talkers
  3. Objects ‘talk’ by quoting stockprices

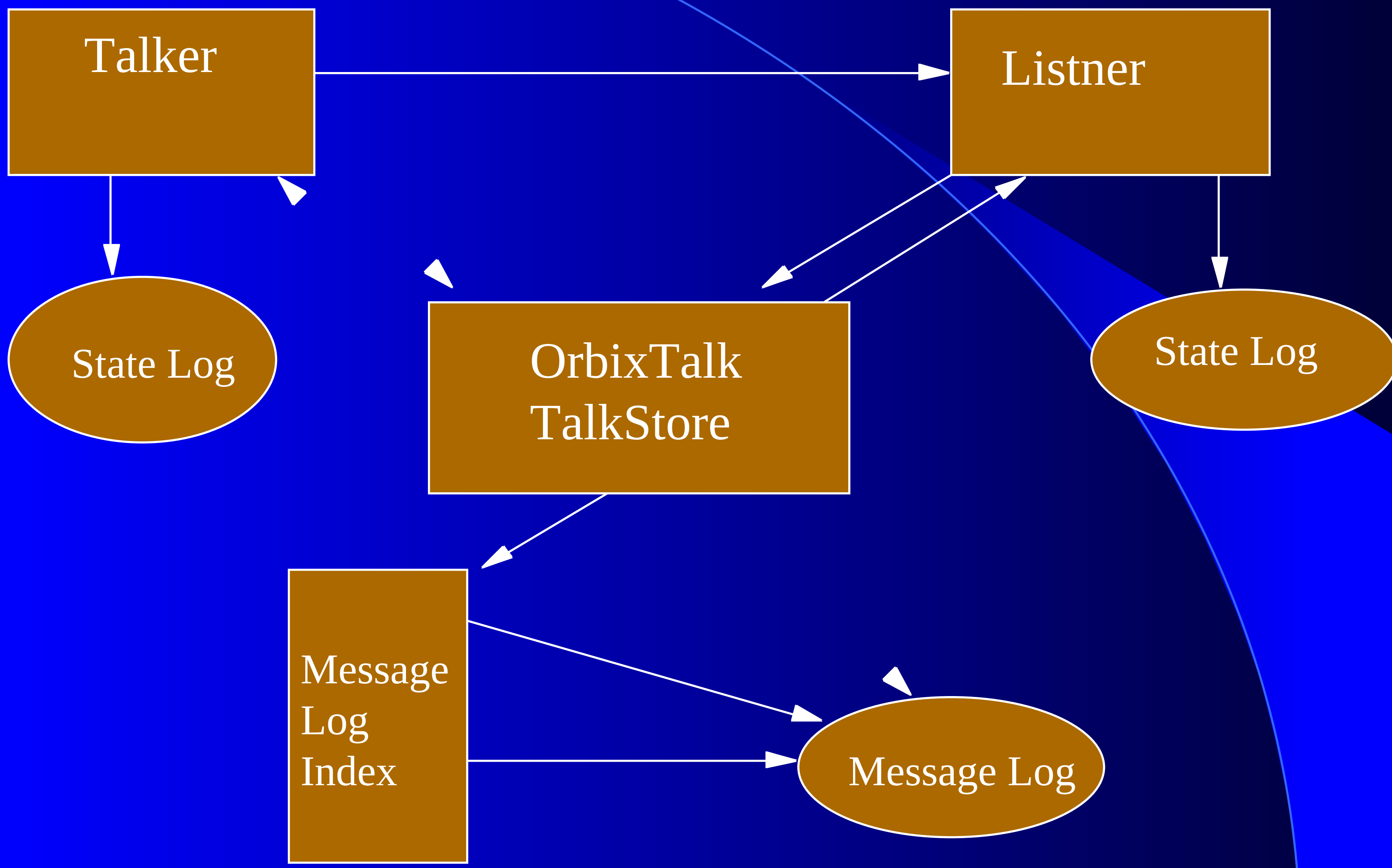
OrbixTalk : Listner

```
main() {
OrbixTalk_var orbixTalkMgr;
StockPrice_var sunStock;
.......
try {

  OrbixTalkMgr = OrbixTalk::initialize();

    sunStock = new Stock_i("otrmp//sun");                    // 1. Create Objects

                    ......

    orbixTalkMgr->registerListener(sunStock);          // 2. Register these objects as Listners
                    .......


      CORBA::Orbix.processEvents();            //3. Process Incoming and Outgoing Events

} catch ()
                    ......
```

OrbixTalk : Talker

```
main() {
OrbixTalk_var orbixTalkMgr;
StockPrice_var sunStock;
.......
try {

  OrbixTalkMgr = OrbixTalk::initialize();

    CORBA::Object_ptr obj =  ......                    // 1. Create stockPrice Proxy Objects
                  ......

   sunStock = orbixTalkMgr->registerTalker(sunStock);   // 2. Register these objects
                                                              as Listners
                   .......

     sunstock->quote((CORBA)::Float j);
    CORBA::Orbix.processEvents(1000);              //3. Process Incoming and Outgoing
                                                       Events - every second
} catch ()
                   ......
```

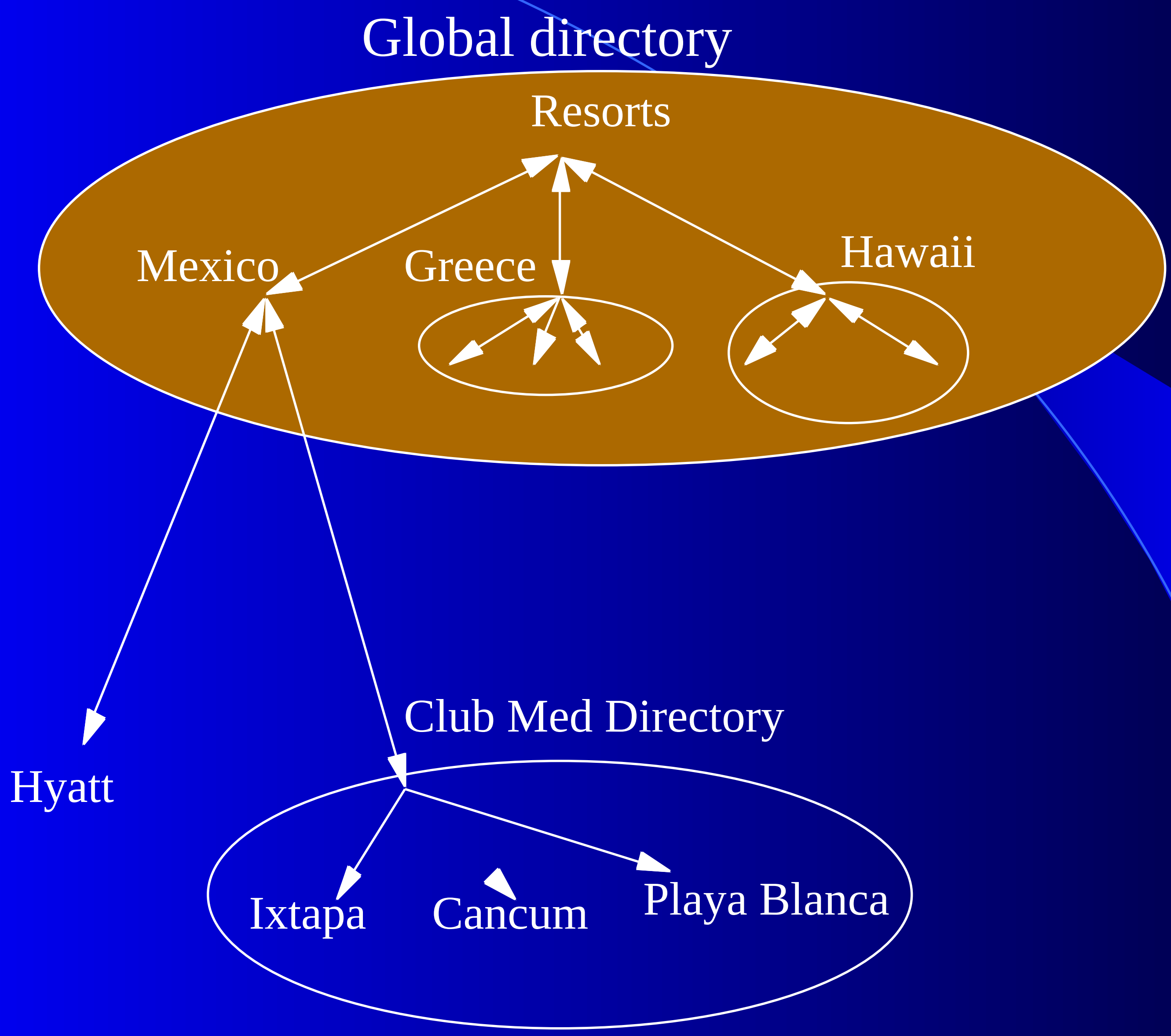
OrbixTalk Messagestore Architecture



The CORBA Object Naming Service

- Is the principal mechanism for objects on an ORB to locate other objects
- Naming_Service : {Human_Names} → Object_References
- *Name Binding* : Name-to-Object Association
- *Naming Context* : Namespace in which the object's name is unique
- Every object has a unique reference ID
- It transparently encapsulates : DCE CDS, ISO X.500, Sun NIS+

What's in a CORBA Object Name ?

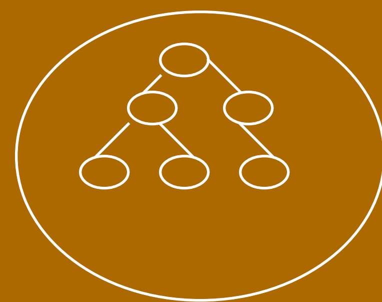


A Compound Name

Resort	Mexico	Club Med	Playa Blanca
Context Name	Context Name	Context Name	Simple Name

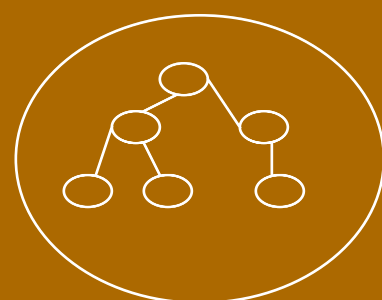
The Object Naming Service Interface

Naming Context



- resolve()
- list()
- destroy()
- new_context()
- unbind()
- bind()
- rebind()
- bind_context()
- rebind_context()
- bind_new_context()

Binding Iterator



- next_one()
- next_n()
- destroy()

The CORBA Object Life Cycle Service

- Provides operations for creating, copying, moving, deleting objects

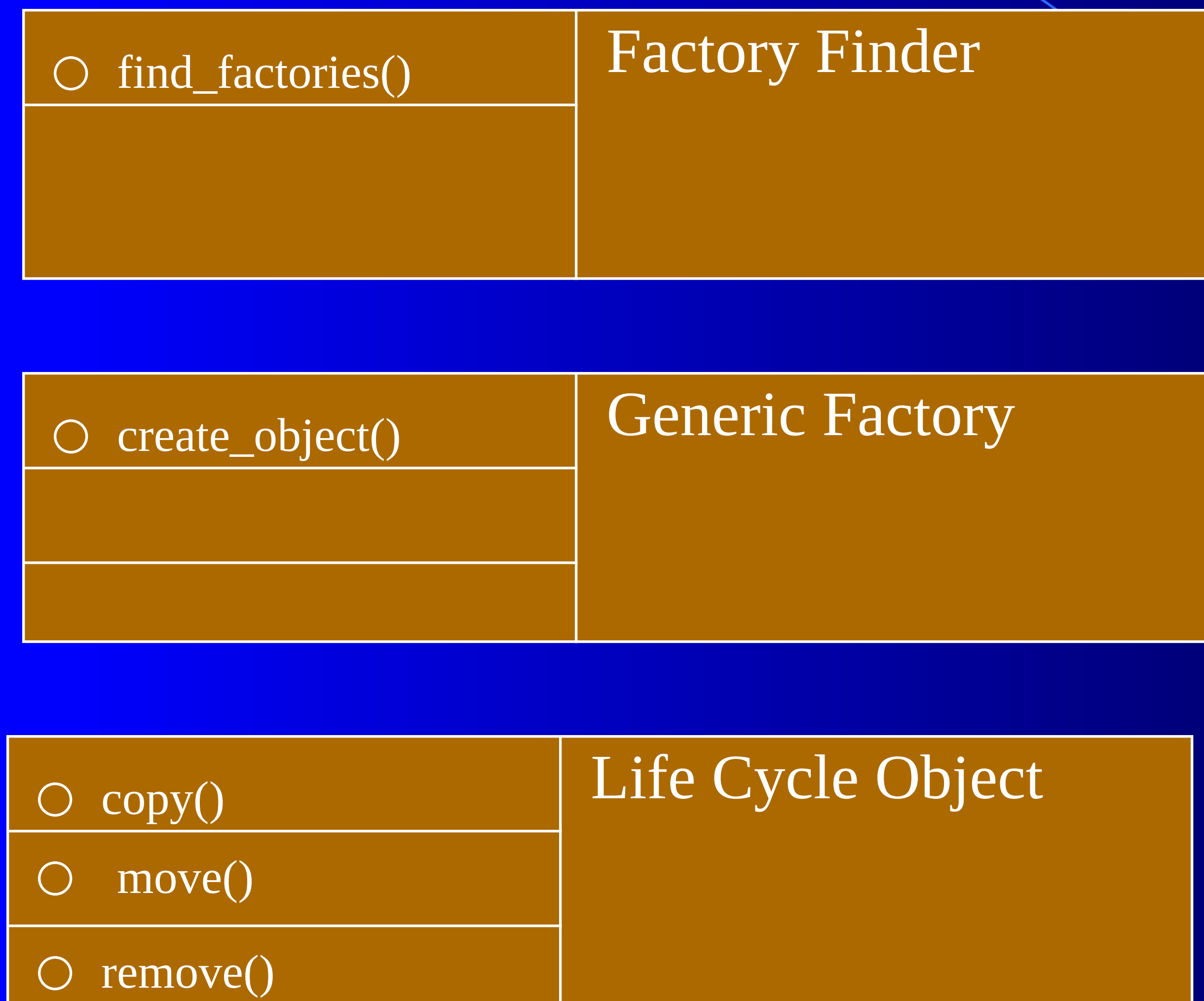
- To create a new object, a client must find a *factory* object -
  - An object that knows how to instantiate an object of that class

- Issue a *create* request - and get back an object reference

- A client can also create an object by cloning an exisiting object with a *copy* operation

- The factory objects must allocate resources, obtain object references, and register the new objects with the Object Adapter and Implementation Repository

The Object Life Cycle Interface



Summary

- CORBA helps to reduce complexity of developing distributed applications

  - However there are many hard issues remaining...

- Products lag standards by about 16 months

  - Commercial ORB implementations