AOT LAB

Agent and **O**bject **T**echnology **Lab**
Dipartimento di Ingegneria dell'Informazione
Università degli Studi di Parma

Distributed and Agent Systems

Prof. Agostino Poggi

What is CORBA?

- CORBA (**C**ommon **O**bject **R**equest **B**roker **A**rchitecture) is a standard architecture for distributed object systems

- CORBA allows heterogeneous programs at different locations to communicate in a network through an "interface broker“

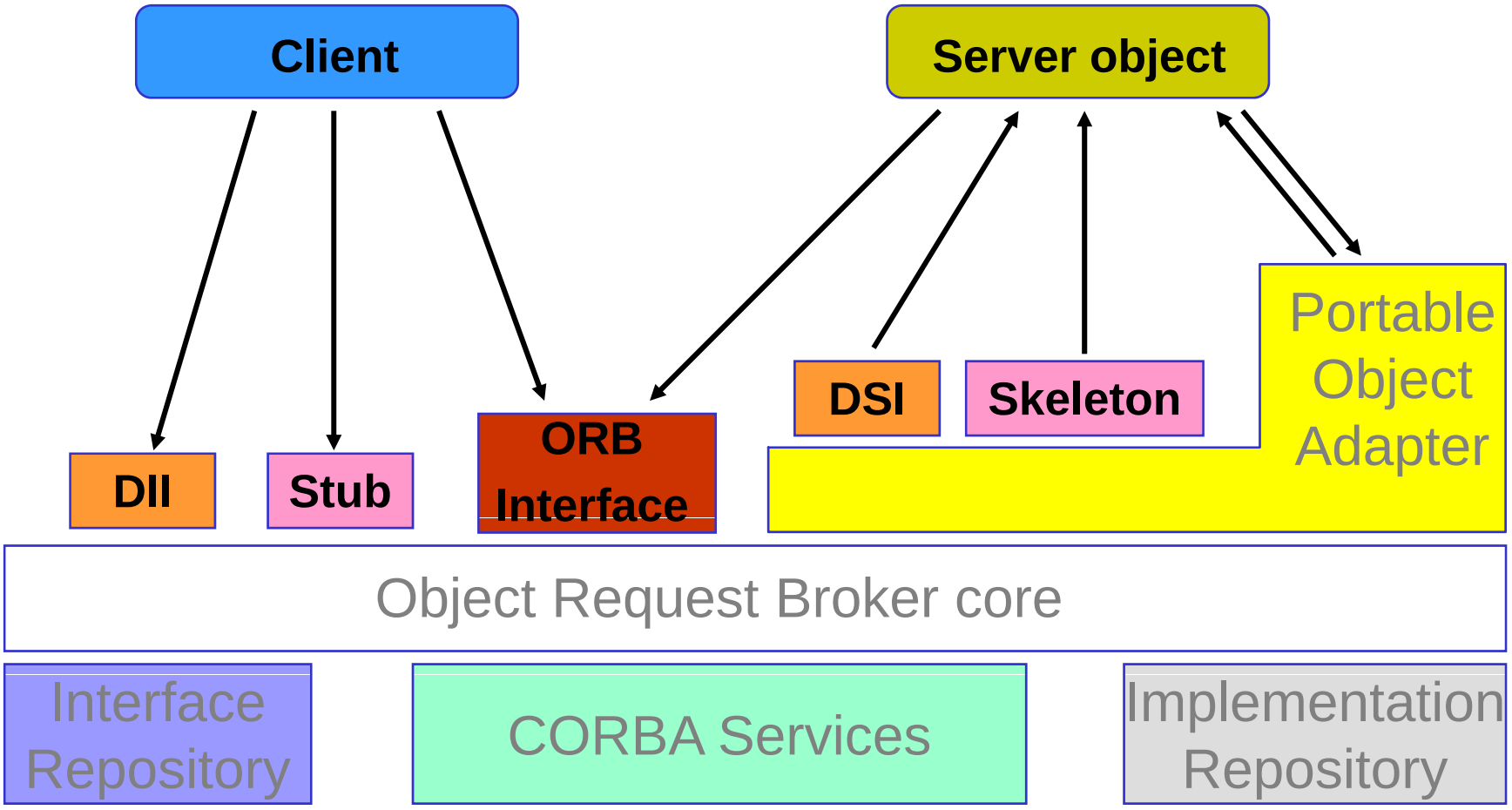
- CORBA is a defined by OMG and its main releases are: Corba 2 (1995) and Corba 3 (2002)

CORBA Key Feature

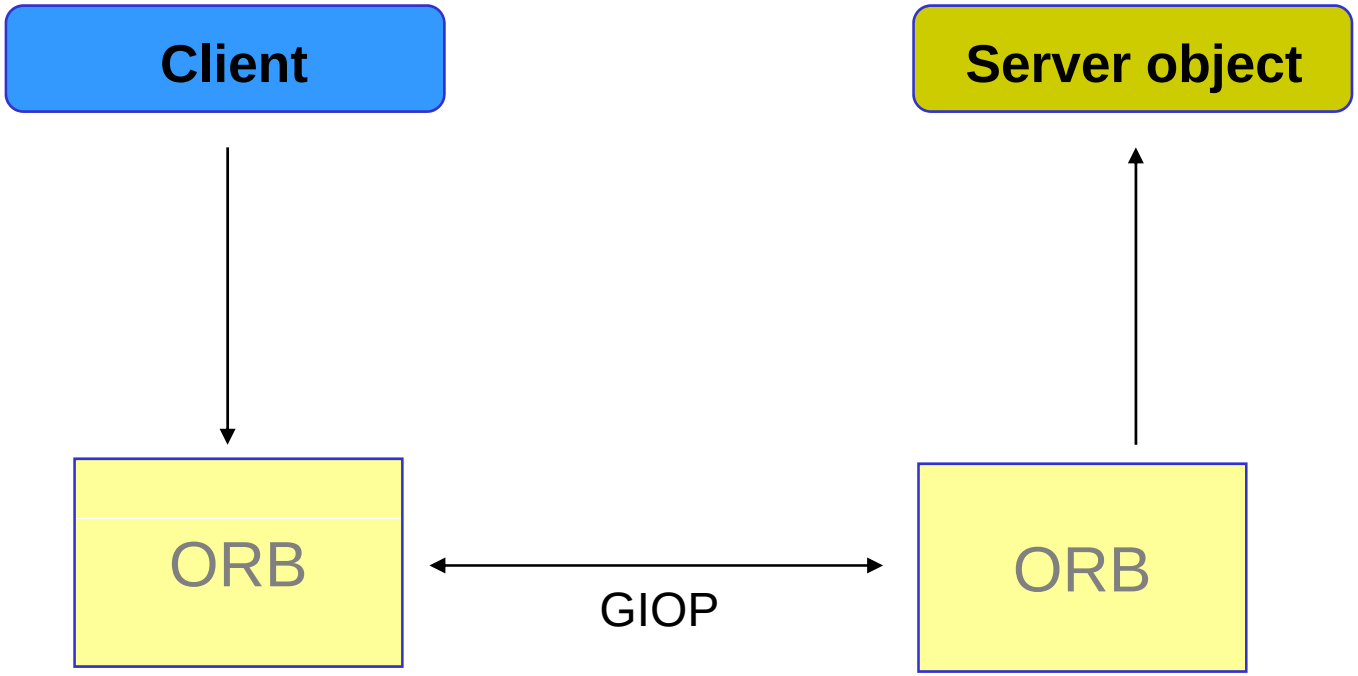
- The key feature of CORBA is the separation between object interface and object implementation
  - The interface to each object is public and language neutral
  - The implementation of an object is hidden from the rest of the system (that is, encapsulated) behind a boundary that the client may not cross
  - Clients access objects only through their advertised interface, invoking only those operations that the object exposes through its IDL interface

AOT LAB

CORBA Architecture



CORBA Architecture



Object Model

- Objects provide services
- Clients makes a request to an object for a service
- Client **doesn't need to know** where the object is, or anything about how the object is implemented
- Object interface must be known (public) and provides signature for each object method

Object Identification

- Objects have globally unique identifiers called **I**nteroperable **O**bject **R**eferences (IORs), which are machine generated and not intended for human use

- A named object has a well-known, unique, human-friendly name that clients use to access it
  - A server process advertises one or more named objects
  - Client processes have no named objects

IOR

- An IOR contains the information required for a client ORB to connect to a CORBA object or servant:
  - IIOP version
  - Host: Host TCP/IP address
  - Port: ORB TCP/IP port number
  - Key: Value identifying the object
  - Components: Additional information applicable to object method invocations, such as supported ORB services and proprietary protocol support

IOR

- In short, an IOR specifies
  - The wire protocol for talking to an object
  - The object's network location
- When a client gets an IOR of a CORBA object, it’s easy for it to transfer the IOR into a reference of that object

Object References

- Clients don’t have objects, they just have object references
- Object references are light-weight objects that have the same interface as remote objects, except their task is to forward all methods invocations to the server for remote object execution
- Object references can be persistent (saved for use later)

ORB

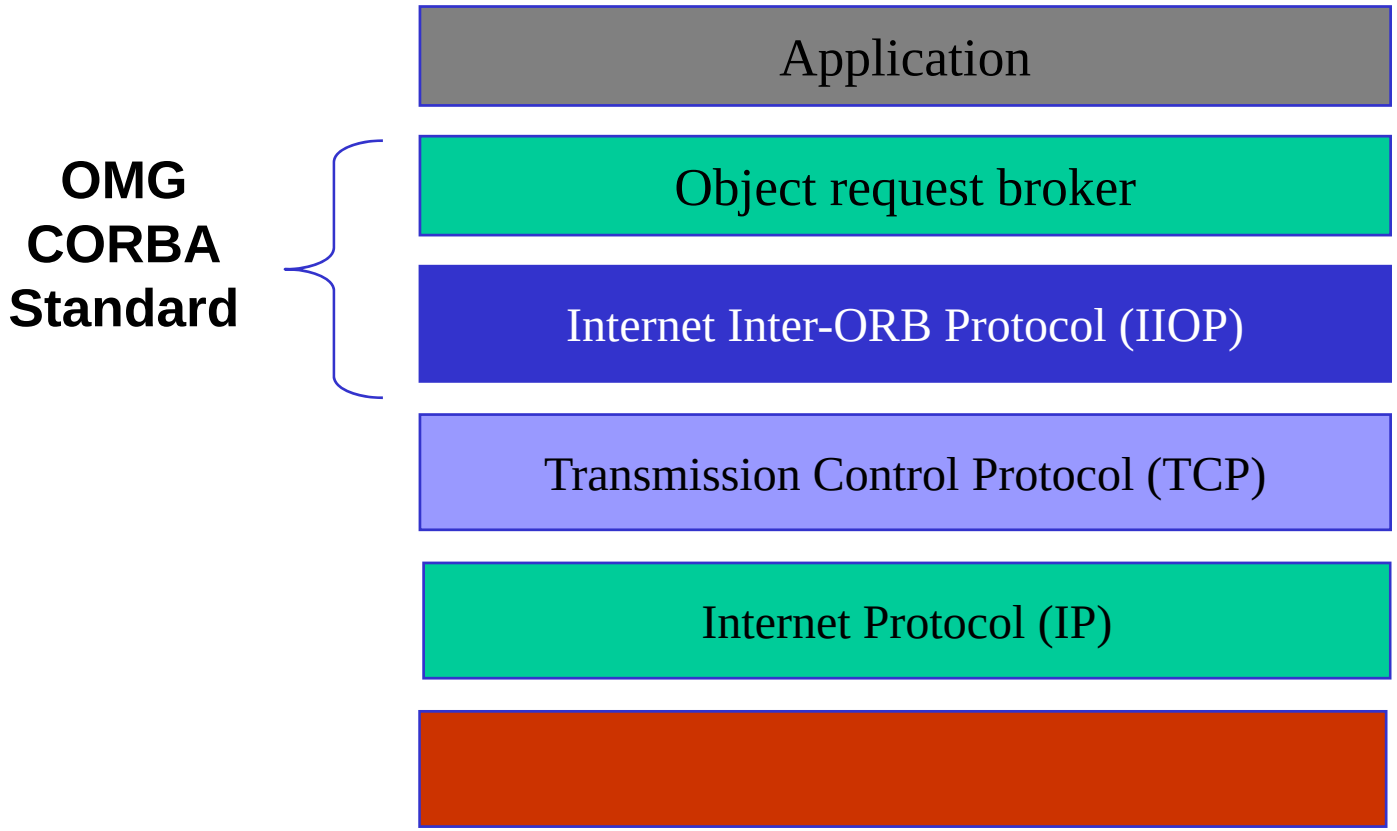
- The **O**bject **R**equest **B**roker (ORB) provides the middleware of communication between a client and server distributed objects
  - An ORB is an abstract entity that acts as the middleman in all remote method invocations
- An ORB finds a server that can handle a method invocation, passes the request to the server, receives the response and forwards it to the client
- The functions handled by an ORB are actually implemented in both client and server

Object Adapters

- Object adapter allow an object implementation accesses services provided by the ORB
- Object adapters functions include:
  - Generation and interpretation of object references
  - Method invocation
  - Security of interactions
  - Object and implementation activation and deactivation
  - Mapping object references to implementations
  - Registration of implementations
- The default adapter is the Portable Object Adapter (POA), that is an object adapter that can be used with multiple ORBs with a minimum of rewriting needed to deal with different vendors' implementations

GIOP & IIOP

- The General Inter-ORB Protocol (GIOP) specifies a standard transfer syntax (low-level data representation) and a set of message formats for communications between ORBs
- The GIOP is specifically built for ORB to ORB interactions and is designed to work directly over any connection-oriented transport protocol that meets a minimal set of assumptions
- The GIOP may be mapped onto a number of different transports, and specifies the protocol elements that are common to all such mappings
- The Internet Inter-ORB Protocol (IIOP) specifies how GIOP messages are exchanged using TCP/IP connections


Application
OMG
CORBA
Standard
Object request broker
Internet Inter-ORB Protocol (IIOP)
Transmission Control Protocol (TCP)
Internet Protocol (IP)

IDL

- **I**nterface **D**efinition **L**anguage (IDL) is the language used to describe object interfaces
  - The same basic idea as a protocol definition file for RPC
- IDL is a declarative language, it only describes object interfaces
- IDL is language neutral
  - There are mappings for many object oriented languages (C, C++, Java, COBOL, Smalltalk, Ada, Lisp, Python, and IDLscript)
- IDL has a syntax similar to C++ and supports interface inheritance
  - All operations are effectively virtual

IDL Types

short,	Integers	char	8-bits codes
unsigned short		wchar	n-bits codes
long		string	chars array
unsigned long		wstring	wchars array
long long		array	
unsigned long long		sequence	
float	Floats	struct	
double		union	
long double		enum	
octet	Bytes	any	any data item
boolean	True, False	void	

IDL Types

```
long myArray[20];
double myMatrix[3][3];


typedef struct {
  string<64> name;
  enum { red, green, blue } color;
  sequence<octet> rawData;
} myStructType;


enum Color { red, green, blue };
const Color DFCOLOR = red;
```

```
union Bar;

typedef sequence<Bar> BarSeq;

union Bar switch(boolean) {
  case True:
    long l_mem;
  case False:
    struct Foo {
      double d_mem;
      BarSeq nested;
    } s_mem;
};
```

IDL Modules

- A module is a basic unit that defines a distributed application
  - Allows interfaces and associated definitions to be grouped
  - Defines a naming scope
- A module has three parts:
  - Variable definition
  - Exception definition
  - Interface definition

```
module M {
  interface B {
    typedef string ArgType;
    ArgType opb(in long i);
  };
};
module N {
  typedef char ArgType;
  interface Y : M::B {
    void opn(in ArgType s);
  };
};
```

IDL Exceptions

- Exceptions implement the standard way of processing errors in CORBA
- Exceptions are syntactically identical to structures
- An exception is defined by a name and a list of public data members that are to be transmitted with the exception

```
module CORBA {
  ...
  #define ex_body {\
    unsigned long minor;\
    completion_status status; }

  enum completion_status {
     COMPLETED_YES,
     COMPLETED_NO,
     COMPLETED_MAYBE };
  ...
  exception UNKNOWN
    ex_body;
  exception BAD_PARAM
    ex_body;
  ...
};
```

IDL Interface

- Interfaces of remote objects are defined by the interface declaration
- An IDL interface is a sequence of one or more constant and/or operation declarations
- An interface may be derived from any number of base interfaces

```
interface A {
  typedef long L1;
  short opA(in L1 l_1);
};

interface B {
  typedef short L1;
  L1 opB(in long l);
};

interface C : B, A {
  typedef A::L1 L3;
  B::L1 opC(in L3 l_3);
};
```

IDL Operation

- An operation declaration is defined by the following parts:
  - The optional operation (**oneway**) attribute
  - A return type
  - The operation name
  - An optional list of parameter declarations
  - A raises expression

IDL Operation

- Each parameter declaration needs a directional attribute:
  - **in**: the parameter is passed from client to server
  - **out**: the parameter is passed from server to client
  - **inout**: the parameter is passed in both directions

IDL Operation

- The operation attribute (**oneway**) specifies which semantics the communication service must provide for invocations of the operation
  - If the attribute is present, the invocation semantics are best-effort
    - Do not guarantee delivery of the call
    - The operation must not contain any output parameters and specify a void return type
  - If the attribute is not present, the invocation semantics are:
    - At-most-once if an exception is raised
    - Exactly-once if the operation invocation returns successfully

IDL Compiler

- IDL compilers implement language mappings in software
- Every ORB comes with one or more IDL compilers, one for each language that it supports
  - Because the mappings are standard and compilers implement this standard
  - Every vendor's IDL compiler will produce language code with the same mappings, allowing your client and server code to port from vendor to vendor without change (of course, as long as you resist vendor extensions)

IDL Compiler Output

- IDL compilers should produce at least two files:
  - Client stub
  - Object skeleton
- Stubs and skeletons act as proxies for clients and servers, respectively

AOT LAB

IDL Java Compiler Output

X.idl	
XPOA.java	Abstract class forming the server skeleton. It provides basic CORBA functionality The server implementer must write a “Servant” class, derived from this class
XStub.java	The client stub. The client implementer must link this with the client class.
XOperations.java	This contains the Java operations interface which corresponds to the IDL interface **X**
X.java	This contains the Java signature interface. It extends the operations interface and **org.omg.CORBA.Object** thus providing standard CORBA object functionality
XHelper.java	This final class provides auxiliary functionality. In particular, the narrow() method providing safe downcasting of CORBA object references to their most derived type
XHolder.java	This final class holds a public instance member of type **X**. Instances of this are needed when an argument of type **X** is an out or **inout** argument of an IDL operation

Interface Repository

- An Interface Repository provides persistent storage of IDL interface declarations
  - For an interface of a given type it can supply the names of the methods and for each method, the names and types of the arguments and exceptions
  - A facility for reflection in CORBA
  - If a client has a remote reference to a CORBA object, it can ask the interface repository about its methods and their parameter types
  - The client can use the dynamic invocation interface to construct an invocation with suitable arguments and send it to the server

Implementation Repository

- The Implementation Repository contains information that allows the ORB to locate and activate implementations of objects
  - Classes that a server supports
  - Objects that are instantiated
  - Object identifiers
- In addition to its role in the functioning of the ORB, the Implementation Repository is a common place to store additional information associated with implementations of ORB objects:
  - debugging information, administrative control, resource allocation, security, …

Client and Server Interaction

- Clients can call a server operation by using:
  - The **D**ynamic **I**nvocation **I**nterface (DII) that is an interface independent of the target object's interface
  - An IDL stub (the specific stub depending on the interface of the target object)
- Servers receive requests as an up-call through:
  - The **D**ynamic **S**keleton **I**nterface (DSI)
  - An IDL skeleton
- Dynamic invocation is useful because allows a client to use services of servers discovered at run time

DII

- A DII allows the dynamic construction of object invocations
  - A client may specify the object to be invoked, the operation to be performed, and the set of parameters for the operation
  - It is done through a call or a sequence of calls depending on the language mapping

DSI

- A DSI allows dynamic handling of object invocations
  - An object's implementation is reached through an interface that provides access to the operation name and parameters in a manner analogous to the client side's DII
  - Static or dynamic knowledge may be used and knowledge must be provided by implementation code
- A DSI behavior may vary substantially from one programming language mapping or object adapter to another
- A DSI may be invoked both through client stubs and through the DII

AOT LAB

CORBA Services

Object life cycle	Defines how CORBA objects are created, removed, moved, and copied
Naming	Defines how CORBA objects can have friendly symbolic names
Events	Decouples the communication between distributed objects
Relationships	Provides arbitrary typed n-ary relations between CORBA objects
Externalization	Coordinates the transformation of CORBA objects to and from external media
Transactions	Coordinates atomic access to CORBA objects
Concurrency control	Provides a locking service for CORBA objects in order to ensure serializable access
Property	Supports the association of name-value pairs with CORBA objects
Trader	Supports the finding of CORBA objects based on properties describing the service offered by the object
Query	Supports queries on objects

Mobile CORBA

- The key concepts in this specification are:
  - Mobile IOR
  - Home Location Agent
  - Access Bridge
  - Terminal Bridge
  - GIOP Tunneling Protocol

Mobile CORBA

- The Mobile IOR is a relocable object reference that hides from CORBA clients the mobility of terminals on which CORBA servers are running
  - Identifies the Access Bridge and the terminal on which the target object resides
  - Identifies the Home Location Agent that keeps track of the Access Bridge to which the terminal is currently attached
- The Home Location Agent keeps track of the current location of the mobile terminal
  - Provides operations to query and update terminal location
  - Provides operations to get a list of initial services and to resolve initial references in the home domain

- The Access Bridge is the network side end-point of the GIOP tunnel
  - Encapsulates GIOP messages to the Terminal Bridge and decapsulates the GIOP messages from the Terminal Bridge
  - Provides operations to get a list of initial services and to resolve initial references in the visited domain
  - May also provide notifications of terminal mobility events

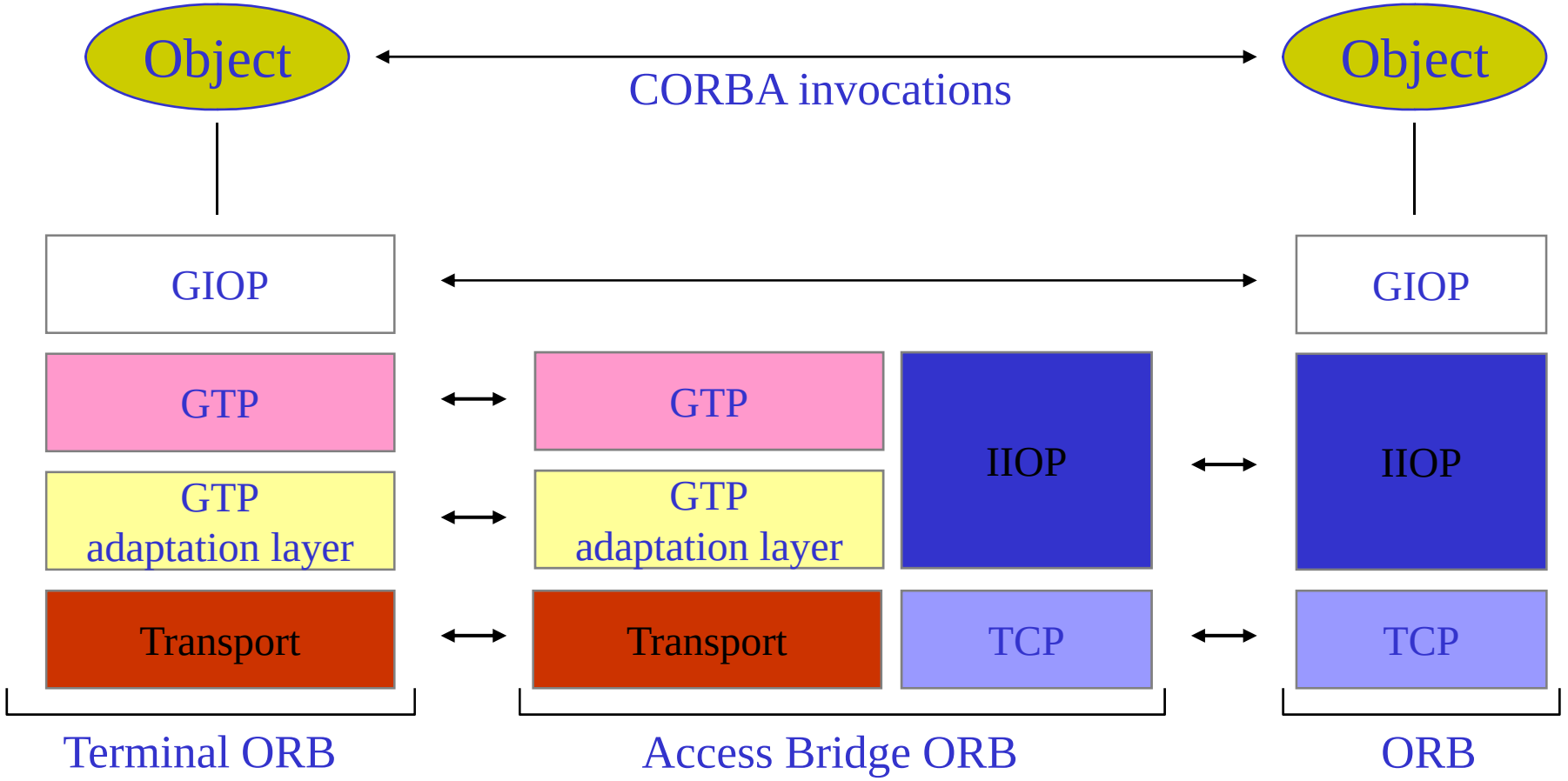
- The Terminal Bridge is the terminal side end-point of the GIOP tunnel
  - Encapsulates the GIOP messages to the Access Bridge and decapsulates the GIOP messages from the Access Bridge
  - May also provide a mobility event channel that delivers notifications related to handoffs and connectivity losses

Mobile CORBA

- The GIOP tunnel is the means to transmit GIOP messages between the Terminal Bridge and the Access Bridge by using the GIOP Tunnel Protocol (GTP)
- The GTP defines
  - How GIOP messages are transmitted
  - Necessary control messages to establish, release, and re-establish a GIOP tunnel
- The GTP is an abstract, transport-independent protocol on which are implemented four concrete tunneling protocols managing GTP messages over TCP, UDP, WAP WDP, and Bluetooth L2CAP

AOT LAB

Wireless CORBA Protocol Stack



Real-Time and Embedded CORBA

- Many applications require real-time QoS support and/or the deployment of some of their parts on embedded nodes with limited resources
- CORBA tries to cope with both the problems offering:
  - A set of policies and mechanisms for resource configuration/control that allow an end-to-end predictability of timeliness in a fixed priority system
  - Different solution for reducing the burden of a typical CORBA application

Real-Time Policies and Mechanisms

- Processor Resources
  - Thread pools
  - Priority models
  - Portable priorities
  - Synchronization
- Communication Resources
  - Protocol policies
  - Explicit binding
- Memory Resources
  - Request buffering

AOT LAB

Use CORBA with Embedded Nodes

- Use of a CORBA gateway between the CORBA world and the embedded nodes
  - Communication between the gateway and the embedded system is provided through a protocol supported by the embedded system
- Use of an IIOP engine on the embedded nodes (~15 Kbytes)
  - Communication between all the nodes of the system is through the IIOP protocol
- Use of a lightweight CORBA implementation that eliminates some of the functionalities of CORBA (from 0,15 - 5 Mbytes up to 30-60 Kbytes)

CORBA Application Development

- Step 1
  - Identification of client and server objects from requirements
- Step2
  - Define interfaces of the server objects
- Step 3
  - Generate client stubs and server stubs from the interface definitions
- Step 4
  - Use client stub to implement the client
  - Use the server stub to implement the server

IDL and Servant

```
module HelloApp {
  interface Hello {
    string sayHello();
    oneway void shutdown(); };
};
```

idlj **-fall** Hello.idl

- **Hello.java**
- **HelloOperations.java**
- **HelloHelper.java**
- **HelloHolder.java**
- **HelloStub.java**
- **HelloPOA.java**

Servant

```
class HelloServant extends HelloPOA {
  private ORB orb;
  public void setORB(ORB orb_val) {
    orb = orb_val; }
  public String sayHello() {
    return "\nHello World!!\n"; }
  public void shutdown() {
    orb.shutdown(false); }
}
```

Server

```
public class HelloServer {
  public static void main(String args[]) {
    try {
      ORB orb = ORB.init(args, null);
      POA rootpoa = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
      rootpoa.the_POAManager().activate();
      HelloServant helloImpl = new HelloServant();
      helloImpl.setORB(orb);
      org.omg.CORBA.Object ref = rootpoa.servant_to_reference(helloImpl);
      Hello href = HelloHelper.narrow(ref);
      org.omg.CORBA.Object objRef = corb.resolve_initial_references("NameService");
      NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
      NameComponent path[] = ncRef.to_name("Hello";);
      ncRef.rebind(path, href); orb.run(); }
    catch(Exception e) {
      System.err.println("ERROR : " + e);
      e.printStackTrace(System.out); } }
 }
```

Client

```
public class HelloClient {
  static Hello helloImpl;
public static void main(String args[]) {
    try {
     ORB orb = ORB.init(args, null);
     org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");
     NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
     String name = "Hello";
     helloImpl = HelloHelper.narrow(ncRef.resolve_str(name));
     System.out.println("Got a handle on the server object: " + helloImpl);
     System.out.println(helloImpl.sayHello());
     helloImpl.shutdown(); }
   catch(Exception e) {
     System.out.println("ERROR : " + e);
     e.printStackTrace(System.out); } }
}
```

Execution

1. Start the Java IDL name server:

   orbd -ORBInitialPort 1050 &

2. Start the Hello server:

   java HelloServer -ORBInitialPort 1050 -ORBInitialHost localhost &

3. Run the Hello application client from another window:

   java HelloClient -ORBInitialPort 1050 -ORBInitialHost localhost &

   Hello World!