CORBA:
Architecture, Concepts and S/W Development

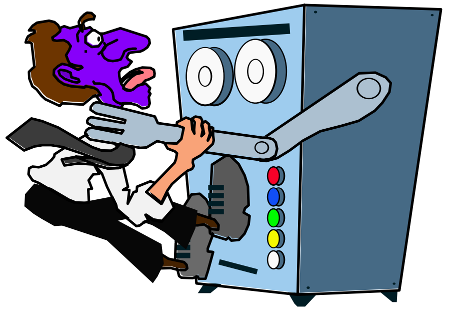
The Problem

- Today’s computing, telecommunication and service environments are inherently *distributed* and *heterogeneous*
  - inter-networked computing and telecommunication devices running different operating systems
- Distributed applications are needed in most areas
  - banking, retail, education, medical, government, telecommunication, management, etc.
- Developing distributed applications whose components collaborate *efficiently*, *reliably*, *transparently* and *scalably* is very hard

Existing Tools?

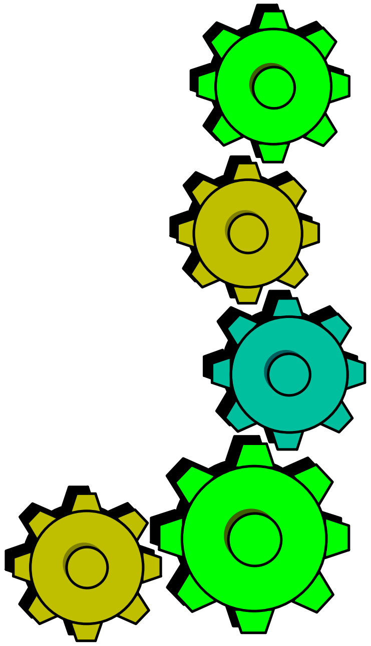
- A major problem stands in the way:
  - There is no single, commercially available, widely recognized & standardized integration approach and framework

Existing tools (Sockets, Sun RPC, DCE) are too low-level; do not offer a unified view of all distributed applications.



Distributed Object Technology

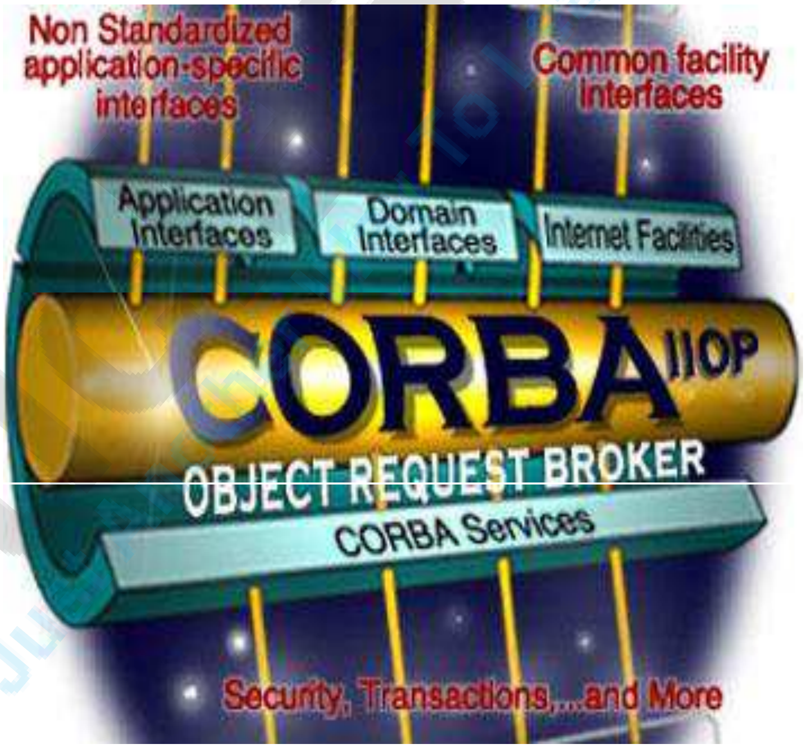
- It is widely believed that an approach based on distributed object technology simplifies the problem:
  - offers a single view of a distributed, heterogeneous system
  - three key concepts in object technology help integration of distributed systems:
    - **Encapsulation**
    - **Inheritance**
    - **Polymorphism**



Candidate Solution: CORBA

Goals:

- Simplify development of distributed applications
- Provide flexible foundation for higher-level services



Non Standardized application-specific interfaces
Common facility interfaces
Application Interfaces
Domain Interfaces
Internet Facilities
CORBA
IIOP
OBJECT REQUEST BROKER
CORBA Services
Security, Transactions,...and More

CORBA Overview

Contents

Background & History

- Object Management Group's (OMG) answer to the need for *interoperability* among the rapidly proliferating number of hardware and software products available today
- CORBA Specification Version 1.1 in 1991
  - failed to provide out-of-box multi-vendor interoperability among ORB implementations
- CORBA 2.0 in December 1997
  - defined true interoperability by specifying how ORBs from different vendors can interoperate
  - defined "CORBA interoperability and the IIOP protocol"
- CORBA 3.0 in June 2002
  - defines "CORBA Component Model"

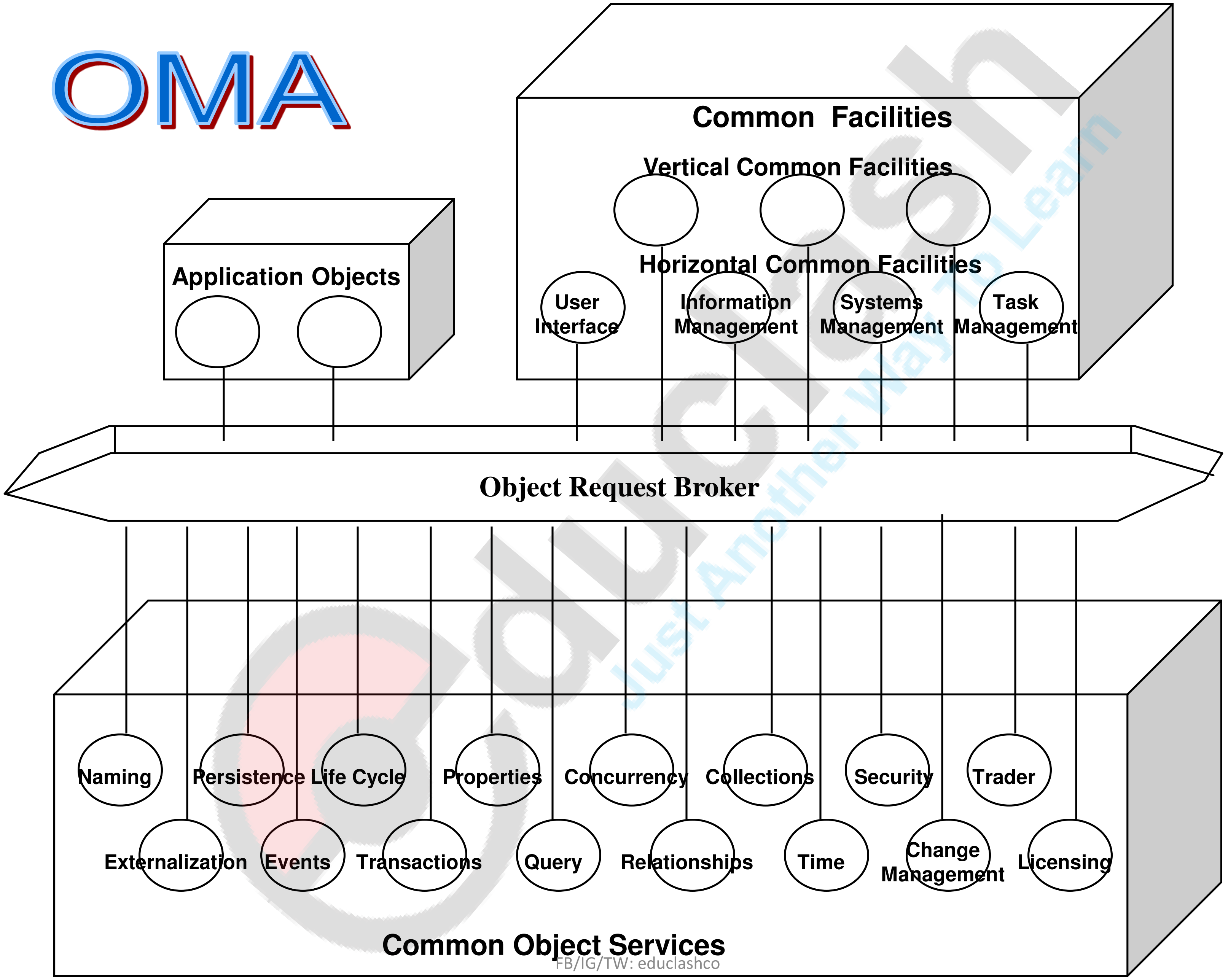
Object Management Group

- Non-profit Consortium based in US, with representations in UK, Japan & Germany
- Founded in April 1989
- 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
- No internal development carried out – produces standard specifications only
- Over 500 members from around the world

Object Management Architecture (OMA)

- A high-level vision of a complete distributed environment
- System Oriented Components
  - Object Request Broker (ORB)
  - Common Object Services
- Application Oriented Components
  - Application Objects
  - Common Facilities

OMA



Common Facilities
Vertical Common Facilities
Horizontal Common Facilities
User Interface
Information Management
Systems Management
Task Management
Application Objects
Object Request Broker
Naming
Persistence
Life Cycle
Properties
Concurrency
Collections
Security
Trader
Externalization
Events
Transactions
Query
Relationships
Time
Change Management
Licensing
Common Object Services

OMA Components (1)

- **Object Request Broker (ORB)**
  - The mechanism and interfaces that enables objects to make requests and receive responses.
  - Provides an *infrastructure* allowing objects to converse, independent of the specific platforms and techniques used to implement the objects.
- **Application Objects**
  - specific end-user client/server applications

OMA Components (2)

- **Common Object Services**
  - A collection of services for maintaining objects
  - Interfaces are provided to create objects, to control access to objects, to keep track of relocated objects, and to control the relationship between objects
  - Event Notification, Persistence, Lifecycles, Naming, Concurrency Control, Relationships, Transactions, Collections, Externalization, Time, Security, Query Service, Licensing, Trading, Change Management, Properties, etc.

OMA Components (3)

- **Common Facilities**
  - a set of generic application functions that can be configured to the specific requirements of a particular application
    - e.g., printing, document management, database, and electronic mail facilities
  - **Horizontal Common Facilities**
    - User Interface
    - Information Management
    - Systems Management
    - Task Management
  - **Vertical Common Facilities** support various vertical market segments:
    - Healthcare, Retailing, CAD, Telecom, Air Traffic, etc.

OMG Object Model

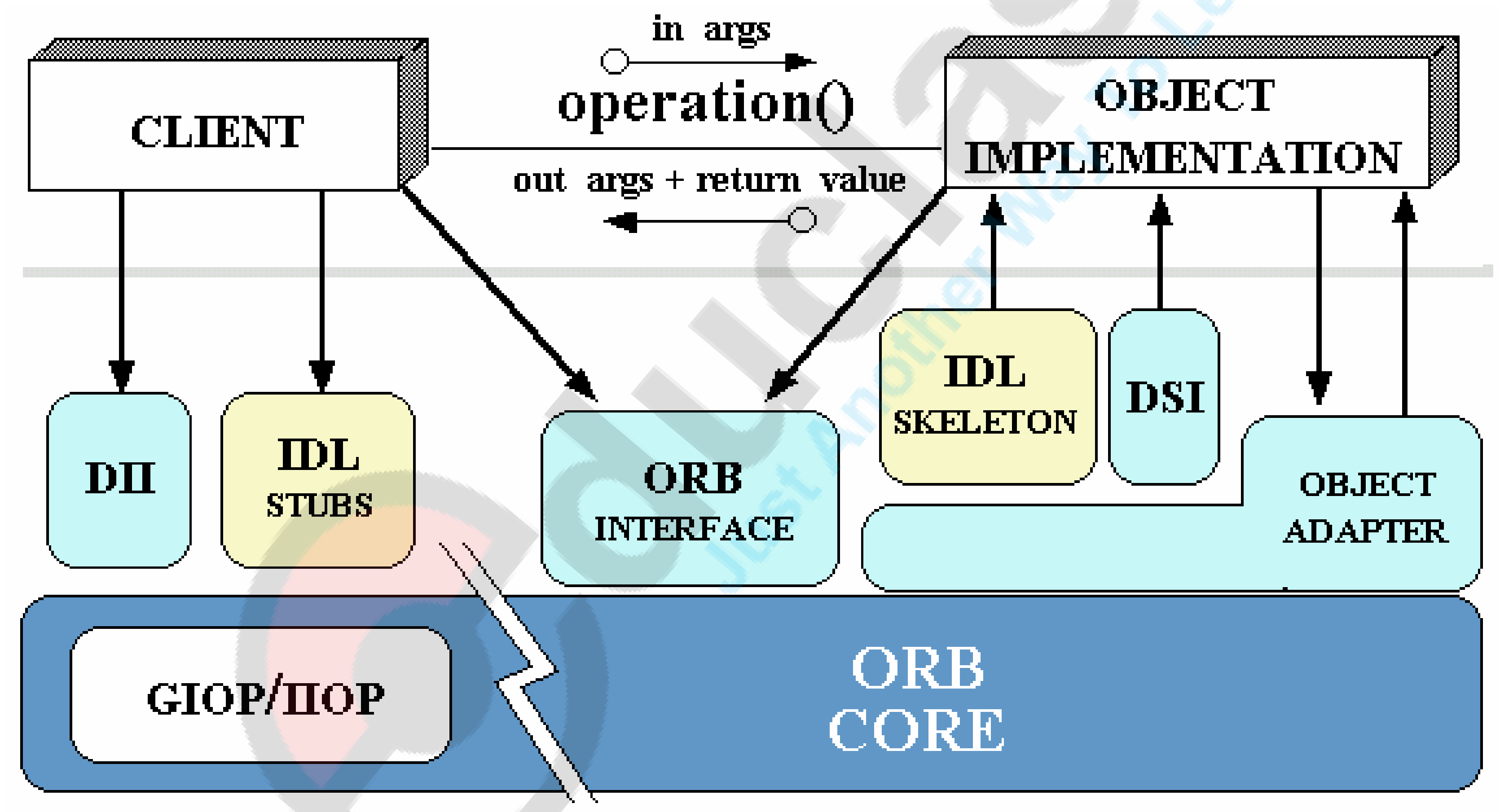
- Underlying specification for all OMG compliant technologies
- The goal is to support interoperability and application portability
- Provides the semantics that define the interfaces that are used to interact with the object
- The basic core concepts of Object Model:
  - **Objects**: instances of types
  - **Operations**: the actions that can be performed on data in objects and are defined by a signature (name, parameters)
  - **Subtyping**: defining a type via inheritance

Common Object Request Broker Architecture (CORBA)

- An OMA-compliant ORB specification of an architecture and interface that allows an application to make request of objects (servers) in a transparent, independent manner, regardless of platform, operating system or location considerations

- Based on the OMG Object Model

CORBA Architecture



CORBA Components (1)

- Object Implementation
- Client
- Object Request Broker (ORB) Core
- ORB Interface
- CORBA IDL Stubs & Skeletons
- Dynamic Invocation Interface (DII)
- Dynamic Skeleton Interface (DSI)
- Object Adapter
- Interface Repository
- Implementation Repository

CORBA Components (2)

- **Object Implementation**
  - defines operations that implement a CORBA IDL interface.
  - can be written in various languages including C, C++, Java, etc.
- **Client**
  - the program entity that invokes an operation on an object implementation
- **Object Request Broker (ORB)**
  - provides a mechanism for transparently communicating client requests to target object implementations
  - makes client requests appear to be local procedure calls

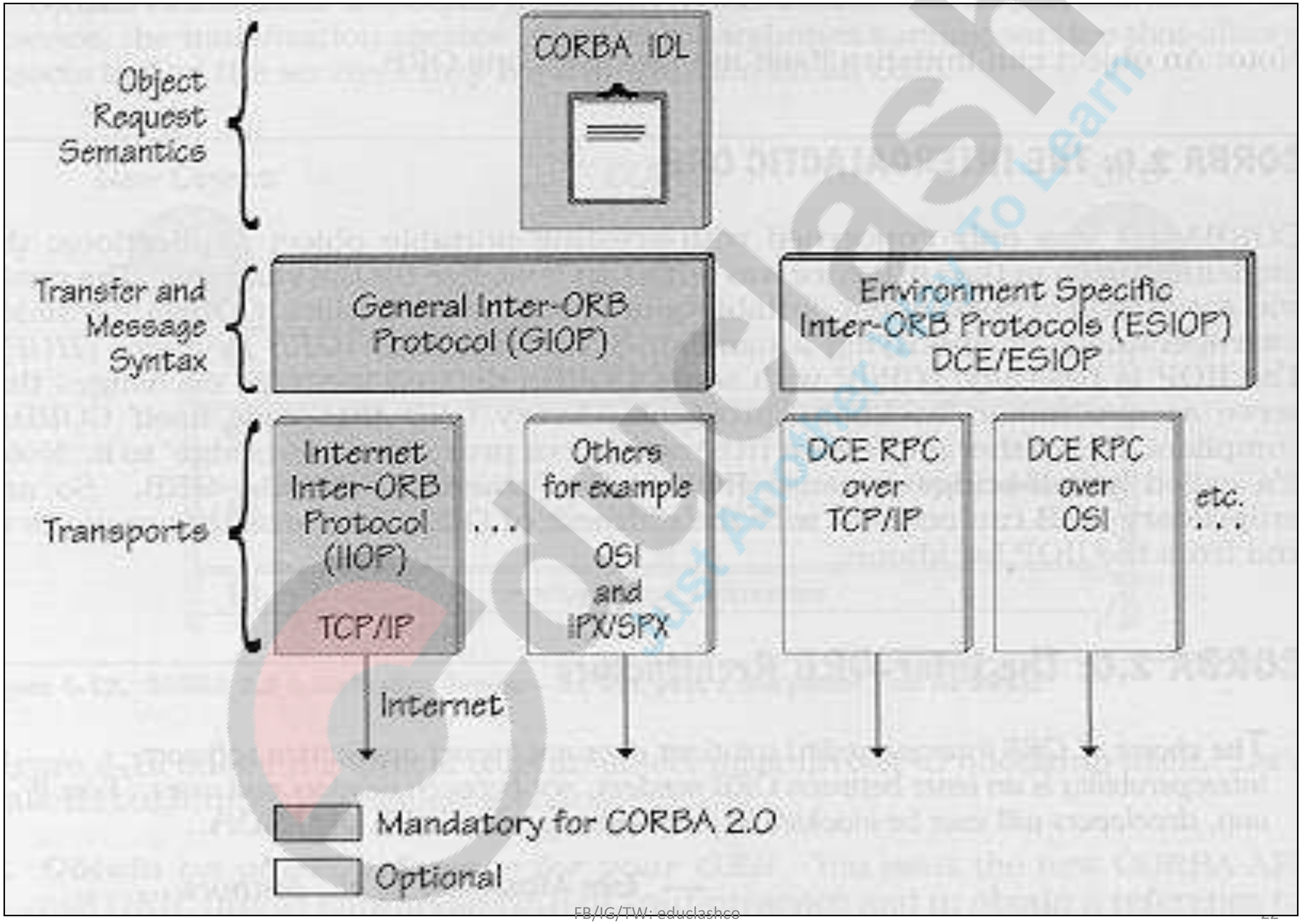
CORBA Components (3)

- **ORB Interface**
  - an abstract interface for an ORB
  - de-couples applications from implementation details
  - provides various helper functions such as converting object references to strings and vice versa, and creating argument lists for requests made through the dynamic invocation interface
- **CORBA IDL Stubs & Skeletons**
  - the static interface between client and server
  - generated by an IDL compiler
- **Dynamic Invocation Interface (DII)**
  - allows a client to directly access the underlying request mechanisms provided by an ORB

CORBA Components (4)

- **Dynamic Skeleton Interface (DSI)**
  - the server side's analogue to the client side's DII
  - allows an ORB to deliver requests to an object implementation that does not have compile-time knowledge of the type of the object it is implementing
- **Object Adapter**
  - provides the run-time environment for instantiating server objects, passing requests to them, and assigning them object references
- **Interface Repository**
  - a run-time database that contains machine-readable versions of the IDL-defined interfaces
- **Implementation Repository**
  - a run-time repository of information about the classes a server supports, the objects that are instantiated, and their IDs

CORBA Inter-ORB Architecture



Interoperability Protocols

- General Inter-ORB Protocol (GIOP)
  - specifies request format and transmission protocol that enables ORB-to-ORB interoperability
- Internet Inter-ORB Protocol (IIOP)
  - specifies a standardized interoperability protocol for the Internet
  - works directly over TCP/IP, no RPC necessary
  - can be used with any transport mechanism that meets certain requirements
- Environment-specific inter-ORB protocols (ESIOPs)
  - e.g., DCE

Commercial CORBA Implementations

- IONA Orbix, OrbixWeb, ORBacus
  - http://www.iona.com
- Sunsoft NEO
  - http://www.sun.com/software/neo/
- Borland Visibroker
  - http://www.borland.com/corba/
- Top Graph'x ORBriver
  - http://www.topgraphx.com/orbriver.htm
- Objective Interface ORBexpress
  - http://www.ois.com/products/prod-1.asp
- Vertel e*ORB
  - http://www.vertel.com

Free CORBA Implementations

- TAO - The ACE ORB
  - http://www.cs.wustl.edu/~schmidt/TAO.html
- JacORB
  - http://jacorb.inf.fu-berlin.de
- Mico
  - http://www.mico.org
- omniORB
  - http://omniorb.sourceforge.net
- OpenORB
  - http://sourceforge.net/projects/openorb/
- ORBit
  - http://orbit-resource.sourceforge.net

For more details on these, visit http://cmeerw.org/freeorbwatch/

Or http://patriot.net/~tvalesky/freecorba.html

S/W Development with CORBA

Contents

CORBA IDL (1)

- Used to specify the interfaces (operations and data) of an object
- Not a programming language itself
- Provides no implementation details
- IDL compilers are used to compile IDL definitions and generate language specific client and server stubs
- Actual body of client and server applications are written in ordinary programming languages (C, C++, Java, Smalltalk, etc.)

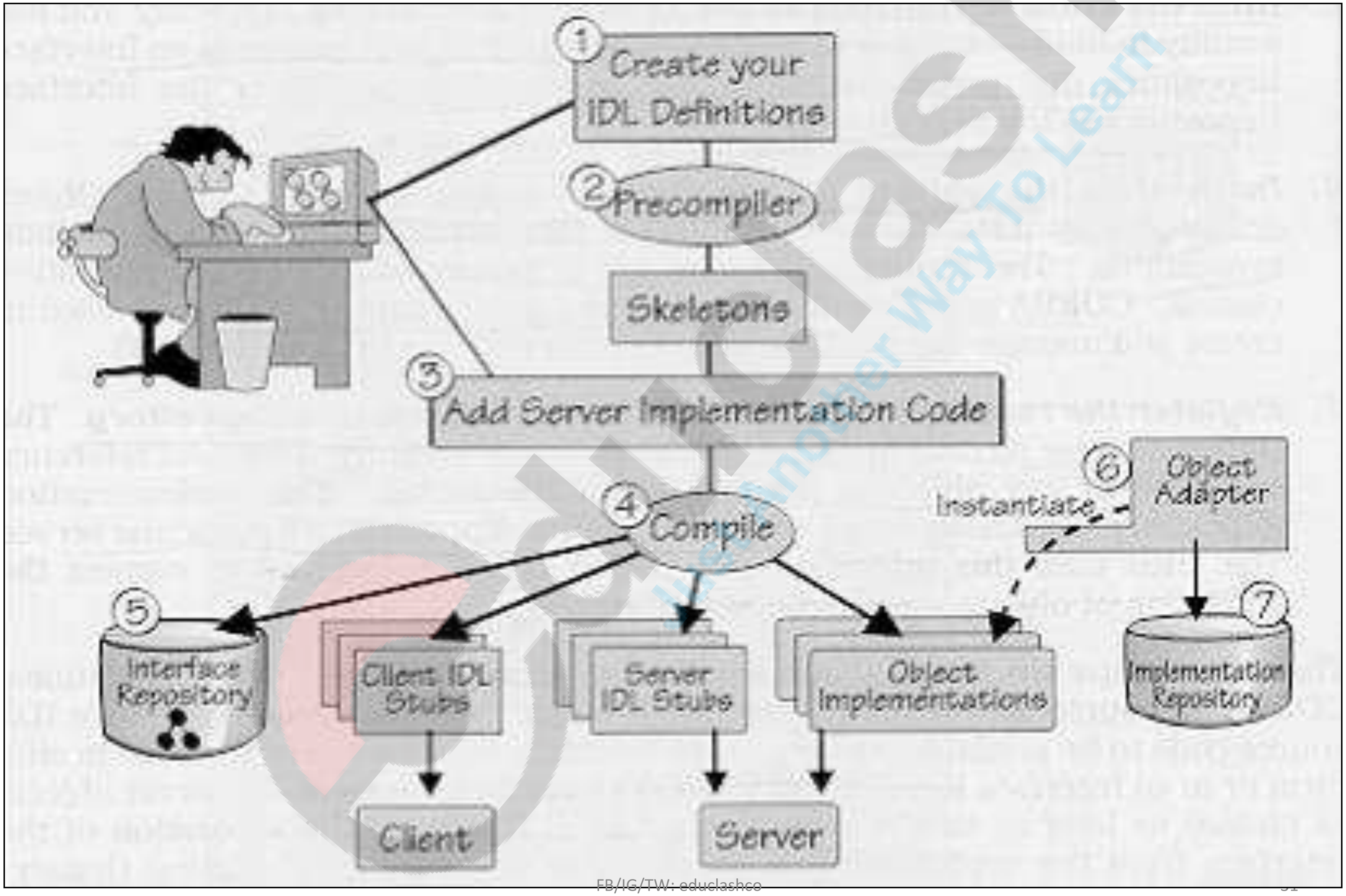
CORBA IDL (2)

- Basic Types: long, unsigned long, short, unsigned short, float, double, octet, char, boolean, any
- Constructed Types: structure, discriminated union
- Template Types : sequence, string, array
- Interface
  - Inheritance Specification
  - Type Declaration
  - Constant Declaration
  - Exception Declaration
  - Attribute Declaration
  - Operation Declaration
- Module (can be nested any number of times)
  - Type, Constant, and Exception Declaration
  - Interface Declaration
  - Module Declaration

An IDL Specification Example

```
// IDL in file grid.idl
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);
};
```

From IDL to Executables

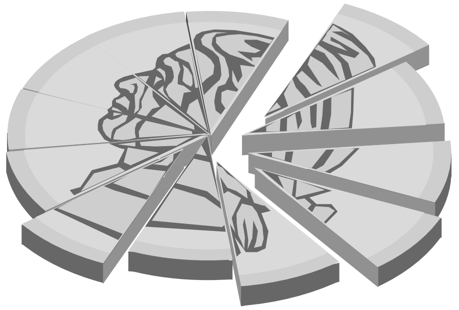


Application Development Steps using CORBA

1. Allocate Tasks
2. Define Remote Object Interfaces
3. Implement Remote Objects
4. Generate Client Stubs
5. Obtain Remote Object References
6. Invoke Remote Methods
7. Configure the System

Allocate Tasks

- Decide how to allocate responsibilities
- Client Side: UI & user input error checking
- Server Side: shared resources, software that controls access to the resources
- Multi-threading allows the user some GUI interaction event while a remote invocation is pending



Define Remote Object Interfaces

- Using IDL, define interfaces in terms of the operations that clients can invoke on the remote objects

```
interface cMSO : ObjectMIO {
    boolean get_attr(in Token my_token,
                     in SOID so_id,
                     in OID oid,
                     out AttrType attr)
            raises (InvalidSOID);
}
```

Implement Remote Objects (1)

- IDL compiler generate some of the remote object's server program (skeleton) which includes
  - definitions for the types defined in the IDL
  - codes for dispatching incoming requests
  - the empty bodies of the methods

```
CORBA::Boolean cMSO_i::get_attr (
            CORBA::Long my_token,
            SOID so_id,
            const char* oid,   AttrType*& attr,
   CORBA::Environment &IT_env) {
   // implementation body goes here
}
```

Implement Remote Objects (2)

- Implement the source code of these methods to provide the remote object’s capabilities
- Build a server program which activates a server
  - e.g., using CORBA Basic Object Adapter (BOA)

```
int main() {
    MIO_i *mio_ptr = new MIO_i("cMSO", "tigris", 1.2);
            cMSO_var cmso_ptr = new cMSO_i(mio_ptr);
    try {
        CORBA::Orbix.impl_is_ready("cMSO");
    }   catch (CORBA::SystemException &sysEx) {
        cerr << "Unexpected system exception" << endl;
        cerr << &sysEx;
    }
}
```

Generate Client Stubs

- Use IDL compiler to generate client-side stub
- Write a client program which invokes the remote object interfaces
- Stub relays an invocation to the real remote object via ORB

```
CORBA::Boolean cMSO:: get_attr (Token my_token,
                    SOID so_id, const char * oid,
                    AttrType*& attr,
                    CORBA::Environment &IT_pEnv)
    throw (CORBA::SystemException,
                    cMSO::InvalidSOID) {
{
    // compiler generated stub code here
}
```

Obtain Remote Object References

- Obtain a reference to an instance of the remote object's class
- Suggested mechanisms
  - object_to_string(), string_to_object(): convert ORB-specific object references to a standard string form and vice-versa
  - Naming Service Interface: allows remote object servers to register their objects by name

```
try {
    cmso_var = cMSO::_bind(":cMSO", host_name);
} catch (CORBA::SystemException &sysEx) {
    cerr << "Unexpected system exception" << endl;
    cerr << &sysEx;
}
```

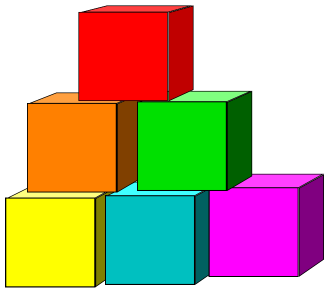
Invoke Remote Methods

- Invoke methods on an object as if it were a local object
- Stub handles all of the transport-level messaging and data marshaling

```
try {
    cmso_var->get_attr(my_token, so_id, buffer, attr);
} catch (const cMSO::InvalidSOID& userEx) {
    cerr << "Invalid Server ID" << endl;
    cerr << &userEx;
} catch (CORBA::SystemException &sysEx) {
    cerr << "Unexpected system exception" << endl;
    cerr << &sysEx;
}
```

Configure the System

- Check if an ORB daemon is running on the server host
- Install the remote server program on the server host
- Make its object reference available
- Install the compiled stub class files and a client program on each client host



Servers and Implementation Repository

- Each server has a unique name within its host machine
- The name of an object (object reference) contains:
  - A unique name within its server (object's marker)
  - Its server name (implementation name)
  - The host name of its server
- Implementation Repository maintains a mapping from a server's name to the file name of the executable code which implements that server --> the developer of the server must register it

Modes for Launching Servers

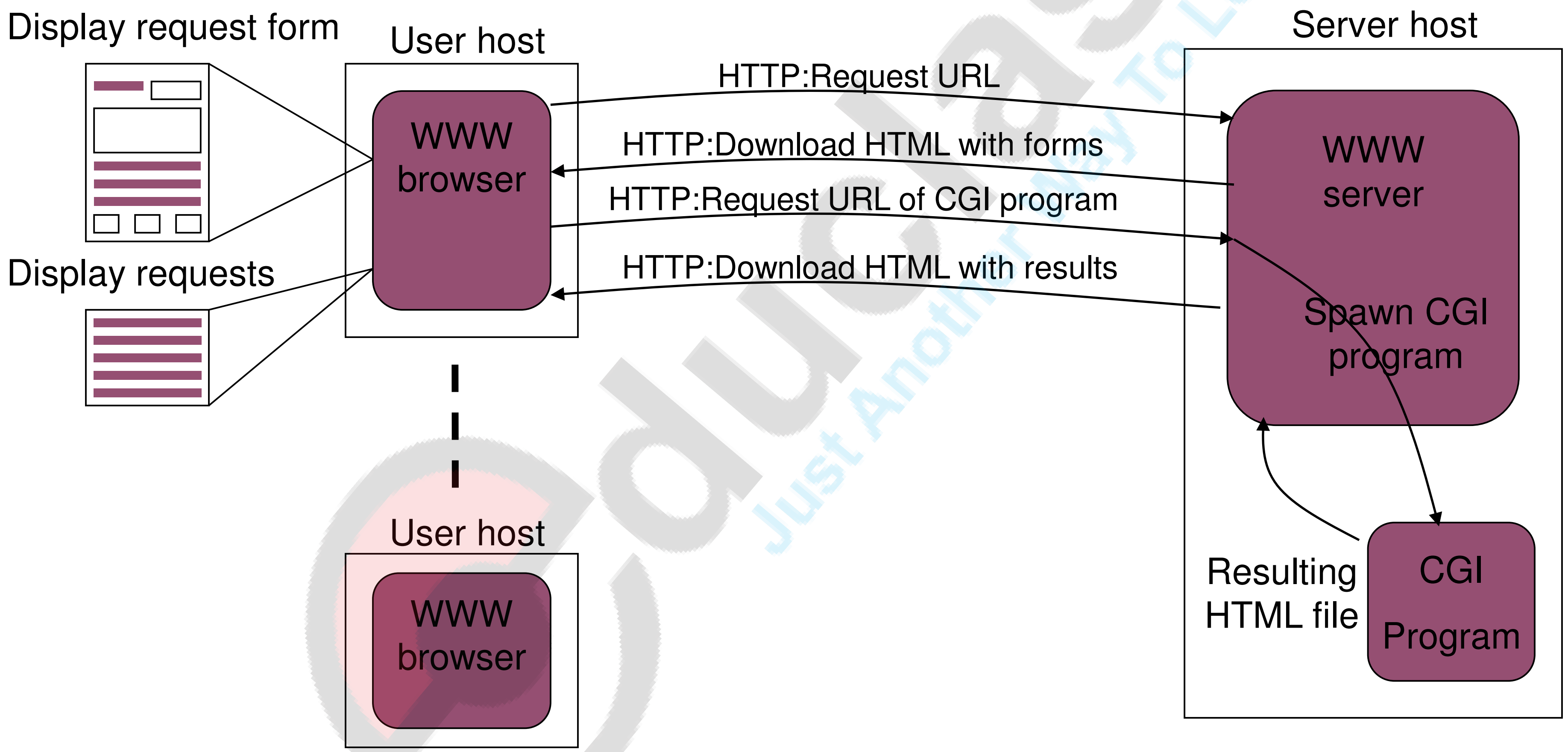
- Shared Mode
  - at most one process for any given server
- Unshared Mode
  - one process per active object
- Per-method-call Mode
  - a separate process for each operation call
- Persistent Mode
  - always active (i.e., not started by Object Adaptor)

Distributed Applications Development using Java/CORBA

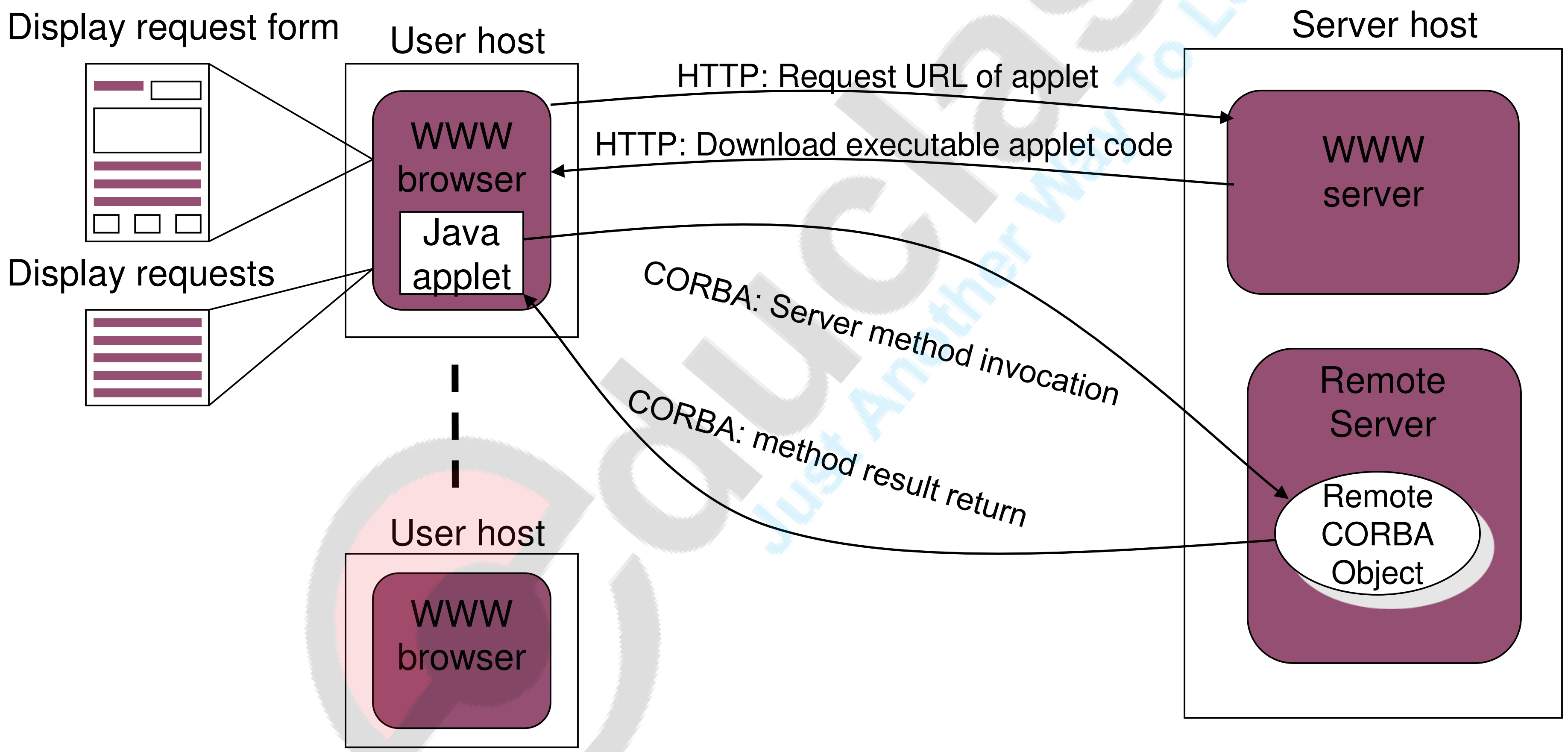
Introduction of CGI *vs*. Java/CORBA for Web-based applications

- Various applications are being developed and used in Internet/Intranet environments
- Common Gateway Interface (CGI) has been used widely for providing simple Web-based client/server applications
- WWW, Java and CORBA can provide a powerful set of tools for developing and deploying distributed applications
- Java applets for WWW-downloadable client software and CORBA objects for server software

WWW CGI-based Application



WWW Java/CORBA-based Application



CGI *vs.* JAVA/CORBA Approach

	CGI	Java/CORBA
Flexibility	– UI and operations definitions are coupled in HTML forms. – Client limited to displaying UI and invoking remote operations. – Remote operation arguments are string only; no structured types.	– UI and remote operations are defined separately. – Client may perform other tasks in addition to managing UI and invoking remote operations. – Remote operation arguments may be of any type, including structured.
Maintainability	– Changes to an HTML form must be manually accounted for in its corresponding CGI program.	– Changes to IDL can be automatically propagated to both client and server via output of IDL copiler.
Server Configuration	– HTML files are installed on the WWW server host. – A CGI program is written to handle one distinct remote operation. – The WWW server spawns a new CGI program instance to handle each remote invocation.	– Java applet client software is installed on the WWW server host. – A remote object server program may be written to handle many distinct remote operations. – A single remote object server instance may handle many remote invocations.

CGI *vs.* JAVA/CORBA Approach

	CGI	Java/CORBA
Client Deployment	– Automatic, on demand, via WWW Platform independent.	– Automatic, on demand, via WWW Platform independent.
User Interface Intuitiveness	– Developer has limited control over UI component layout. – Developer limited to using HTML INPUT tag components.	– Developer has broad control over UI component layout. – Developer may create new components with available ones.
Responsiveness	– Simple UI update requires down loading a new HTML files. – Each remote invocation is slowed by the spawning of a CGI program. – Each remote invocation slowed by acquisition of needed resources. – Client is single-threaded.	– Simple UI updates are handled by the client applet. – Each remote invocation may be handled by an already executing server program. – Remote object server may already have acquired needed resource when remote invocation arrives. – Client can exploit multi-threading.

Application Development Steps using Java/CORBA

- The development steps are almost identical to using CORBA only
    1. Allocate Tasks
    2. Define Remote Object Interfaces
    3. Implement Remote Objects
    4. *Generate Remote Object Proxies*
    5. Obtain Remote Object References
    6. Invoke Remote Methods
    7. *Configure the System*
- The differences are in steps 4 and 7

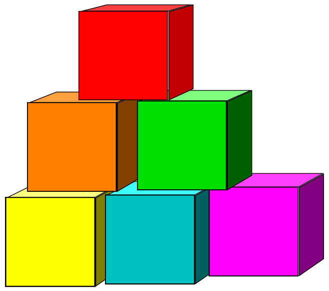
Generate Remote Object Proxies

- Use IDL-to-Java compiler to generate client-side proxy class (stub)
    - e.g., OrbixWeb IDL compiler
- Proxy relays an invocation to the real remote object via ORB

```
public boolean get_attr ( int my_token,
                          short so_id,
                          String oid, MSO.AttrType attr)
        throws MSO._cMSO.InvalidSOID,
        IE.Iona.Orbix2.CORBA.SystemException
{
   // compiler generated Java stub code here
 }
```

Configure the System

- Install the remote server program on the server host
- Make its remote object's reference available
- Install the compiled Java class files of each client applet on the WWW server host
- Install an HTML file for each applet on the WWW server host
- User sites need not install any executable or data files other than those for a Java-enabled browser



Summary

- CORBA is more than a “hype”
- Most widely-used distributed object technology today
- Applications
  - building distributed systems and applications
  - telecommunication network & management software
  - many other areas (such as healthcare, banking/finance, retail, and transportation) are beginning to use CORBA
- WWW, Java and CORBA combination can provide a powerful set of tools for developing and deploying distributed applications