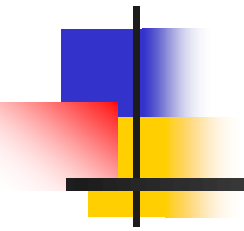
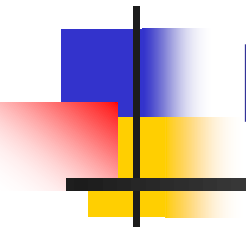


Distributed Objects

Remote Procedure Call (RPC)
Java SE: RMI
Industry Standard: CORBA

Distributed Objects

- Componentization is the process of breaking applications down into reusable components
  - Components may be distributable over the enterprise's network
- Java EE is a distributed component solution
  - Java SE supports socket programming, RMI
  - Adaptors for legacy connectivity
    - *Examples: CICS, SAP .. Enterprise Information Systems*
  - Almost any hardware, only Java language
  - Support CORBA, Web services
- Microsoft solutions include DCOM, .NET, MFC
  - For Microsoft-based servers and MS products
  - Supports CORBA, Web services

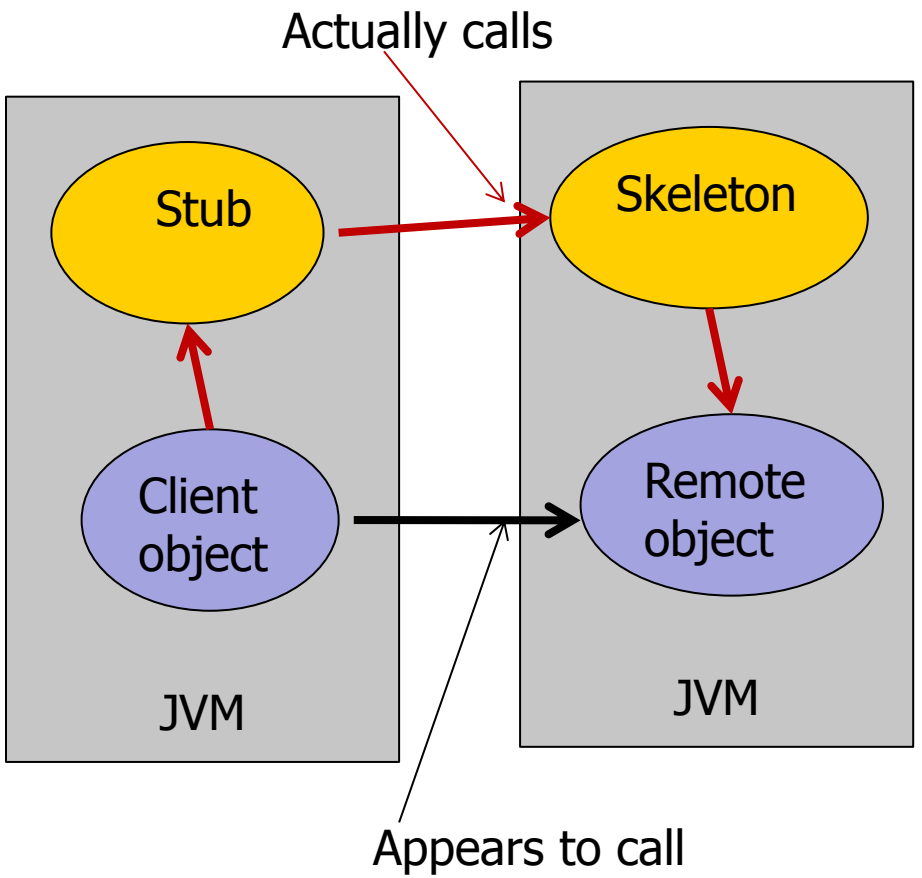


Remote Method Invocation (RMI)

A Java SE
solution for remote objects

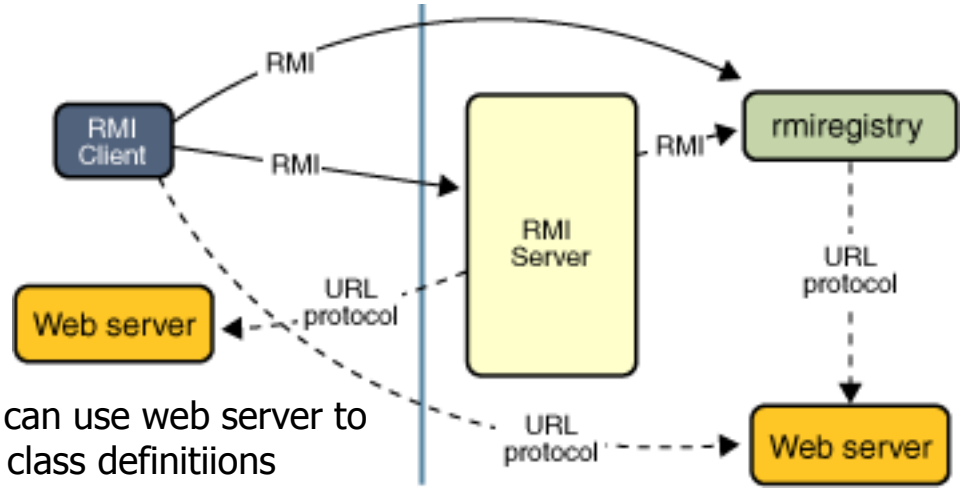
Remote Method Invocation (RMI)

- **Java SE solution** when objects in a single application resided in multiple JVMs
  - Uses a proxy pattern: stub on client side has same client interface as remote object
  - Objects passed as args must be Serializable
  - Stub marshals args into byte stream
  - Skeleton unmarshals
  - Java SE 5 and later RMI infrastructure performs tasks of stub and skeleton
    - Previously generated by rmic



Distributed object application

- Java programs can use RMI for client server apps
  - To access remote objects
    - *Server registers instances in an **rmi registry** on server – bind()*
    - *Client locate object by name in the registry – lookup()*
  - Registry must be running for server and client to use
    - *On Windows:* `jdk\bin> start rmiregistry`
  - Communication handled by RMI infrastructure
    - *Method calls look like regular method calls*
  - Dynamic class loading
    - *RMI also has mechanism for loading class definitions as well as passing serialized objects*
    - *Clients can upload capabilities as classes to server*



RMI can use web server to load class definitiions

Accessing Remote objects

- Remote objects have methods that reside in one JVM and can be invoked by code in another JVM
  - Method must be defined in a Java interface that extends **java.rmi.remote**
    - *All methods throw exception* **java.rmi.remote.RemoteException** *in addition to application-specfic exceptions*
    - *The interface must be on classpath of both client and server*
    - *RMI passes stub (a proxy) for remote class to client as client does not have implementing class*

```
public interface Student extends Remote {
   public void setName(String name) throws RemoteException;
   public String getName() throws RemoteException;
   // set and get methods for other fields
}
```

  - Client looks up object and calls methods on it

```
public class StudentClient {
   // excerpt from method
   try {
      Student s1005 = (Student) Naming.lookup(
          "//" + HOST_NAME + "/Student1005" );
         System.out.println( s1005.getName() );
```

Remote objects

- Implementing class on server side must:
  - Define implementing class for remote interface
  - Extend **RemoteObject**
    - *Use subclass* **UnicastRemoteObject** *for CORBA compatibility*

```
public class StudentImpl
   extends UnicastRemoteObject implements Student{
   private int ID;
   private String name;
   private int credits = 0;
  public StudentImpl( int ID, String name ) throws RemoteException {
      this.ID = ID; this.name = name;
  }
  // get and set methods for all fields
   public String getName() throws RemoteException() {
      return name;
   }
     //
```

  - A server-side class must
    - Create remote objects
    - Register remote objects with RMI registry

RMI server-side programming

```
public class StudentEnrollment {
  public static void main(String[] args) {
    if (System.getSecurityManager() == null) {
      System.setSecurityManager(new SecurityManager());
    }
    try {
      // build up collection of students
      ArrayList<StudentImpl> students = new ArrayList<StudentImpl>();
      students.add(new StudentImpl(1001, "Maria Jones"));
      // ...
      for (StudentImpl s : students) {
        int id = s.getID();
        Naming.rebind("Student" + id, s);
      }
      System.out.println("Student objects bound.");
   } catch (Exception e) {
      e.printStackTrace();
   }
 }
}
```

RMI must have security specified

- Both client and server must:
  - Install security manager

To set up a generic policy manager:

```
// usually in main() method on server side
if (System.getSecurityManager() == null) {
   System.setSecurityManager(new SecurityManager());
}
```

  - Policy specifies who can access remote objects
  - Policy can be specified in policy files specified as argument to the JVM when program launched
    - *Typcially called server.policy and client.policy*

A generic policy file that grants lets client and server access registry on any socket:

```
grant {
  permission java.net.SocketPermission "*:1024-65535",
"connect,accept";
};
```

Command line options for RMI

- The java command takes options:
  - -cp or –classpath specfies where to find .class files
  - -D defines environment variables
    - *java.rmi.server.codebase is directory or jar where remote objects reside*
    - *RMI infrastructure downloads them available to client*
    - *java.security.policy defines securtiy policy for JVM*

Command to run server from command line:

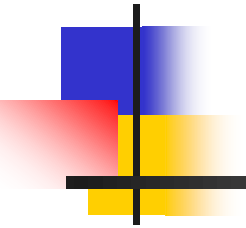
```
java -cp directory and jars containing compiled classes
     -Djava.rmi.server.codebase=
           directory from which clients download classes
     -Djava.security.policy=server.policy
     student.info.server
```

Command to run client from command line:

```
java -cp directory and jars containing compiled classes
     -Djava.security.policy=client.policy
     student.info.client
```

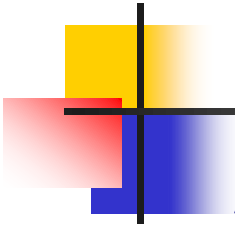
Running an RMI client-server

1. Make interface remote objects available to clients
   - Typically developers of server creates a JAR that holds interface definitions and specifies on when running where it resides
   - RMI mechanism downloads interfaces to clients
2. Start the RMI registry – typically runs on the server
   - From the command line enter **start rmiregistry**
     - *By default registry uses port 1099*
3. Start the server
   - Server specifies security manager
   - Server exports object and binds them to registry
4. Start the client
   - Client specifies security manager
   - Clients looks up remote objects and uses them through the stub/proxy provied by rmi infrastructure



CORBA

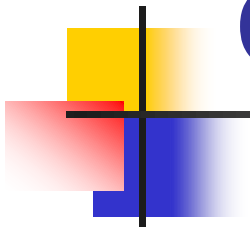
A language- and vendor-neutal solution for remote objects



Common Object Request Broker Architecture (CORBA)

- Vendor-neutral standard for remote procedure call for many OO languages from the Object Management Group (OMG)
  - Defined about 1990 for object oriented languages
    - *Originally used most widely for C++ and smalltalk*
    - *Now for C, C++, Java, COBOL, Smalltalk, Ada, Lisp, Python, Ruby , PL/1 Python …*
- Powerful but very complex
  - For building heterogeneous distributed object systems
- Separates object implementation from interfaces, much like RMI

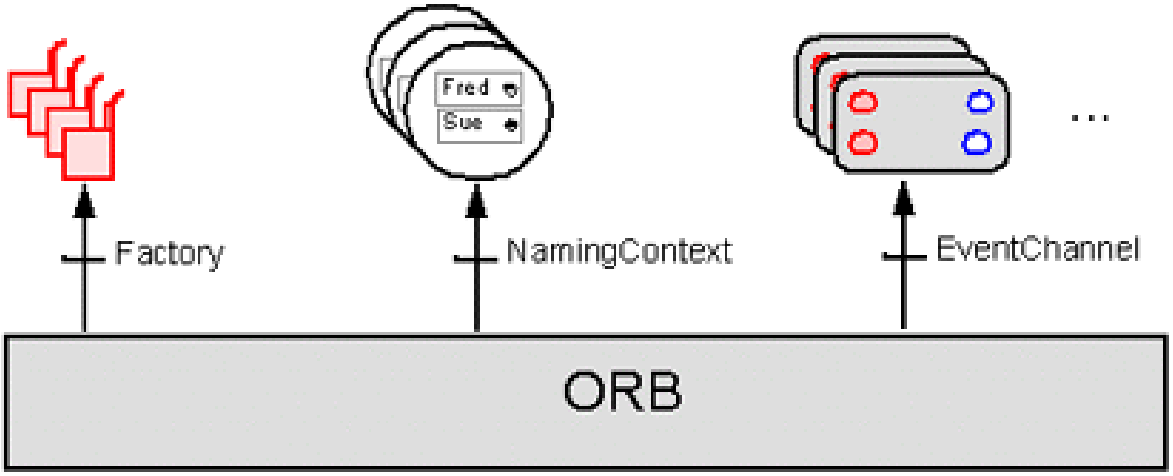
CORBA standards



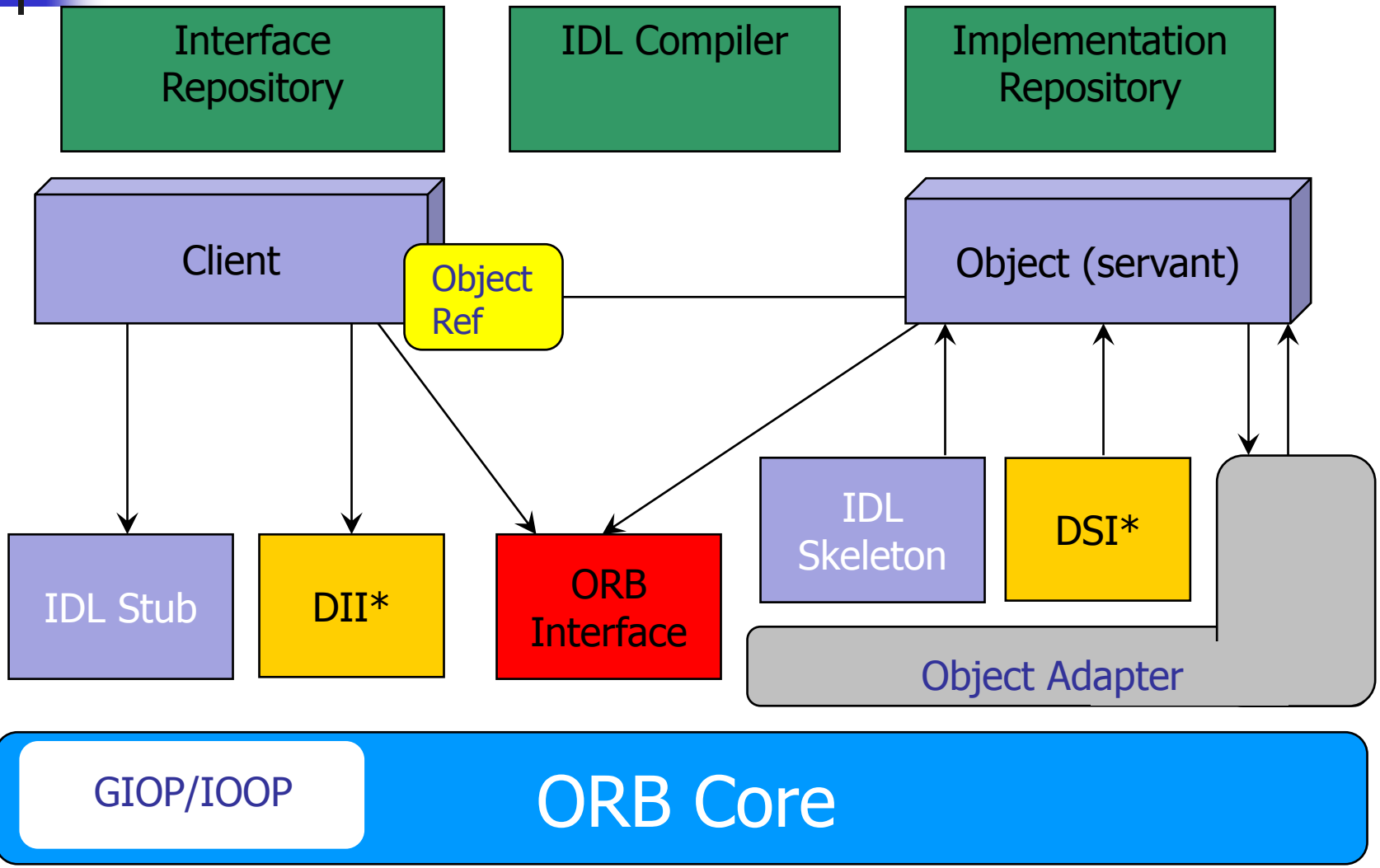
- Object Request Broker (ORB)
  - Implements requests to remote objects
    - *Locates objects on the network – location transparent to client*
    - *ORB finds object, sends request, waits for and returns*
    - *Performs translation between client and remote object langauge*
- Interface Definition Language (IDL)
  - Defines language bindings to neutral object description
  - Maps object references, attributes, data types, exceptions ...
- Internet Inter-Orb Protocol (IIOP)
  - Communication protocol based on TCP/IP
  - Adopted by Java EE and compatible with Java RMI
  - Implementation of general inter-orb protocol (GIOP)_

CORBA Services

- Client holds object reference to remote object
- ORB provides services to support remote procedure call:
  - Object file cycle
  - Naming to locate objects by symbolic names (CosNaming)
  - Events decoupe communcation between objects
  - Relationships between objects
  - Externalization: transformation between CORBA and external media
  - Transactions
  - Concurrency
  - ...



Overview of ORB architecture



*Dynamic Invocation Interface (DII) and DSI can generate stubs and skeletons at runtime.

Modern Distributed Objects

- CORBA-based solutions perform **call by reference**
  - To call a method on an object, obtain reference from a naming service or equivalent
  - To transfer objects and exceptions, serialize them
- Enterprise Java Beans in Java EE
  - Most Java EE Application Servers implement CORBA standards internally and can interoperate with CORBA ORBS
- Various enterprise-scale software products are accessible as CORBA orbs
  - *IBM CICS*
  - *…*

Emerging Solutions

- **Call by value** uses standards to define data format
- **XML** for data transfer
  - Web Services based on SOAP for data transfer and WSDL to itentify and describe service
  - Java EE standard for XML-based web servivces is JAX-WS
- Representational State Notation (**REST**)
  - Maps CRUD operations onto 4 of the HTTP methods PUT/GET/POST/DELETE
    - *URN identifies asset or data to act upon*
  - Usually combined with JavaScript Object Notation (JSON) for data transfer
  - Java EE standard for RESTful Web services is JAX-RS
- **Cloud** solutions
  - Componentization: deconstructing large apps into services
  - Virtualization: running services anywhere (virtual machines)