

Definition

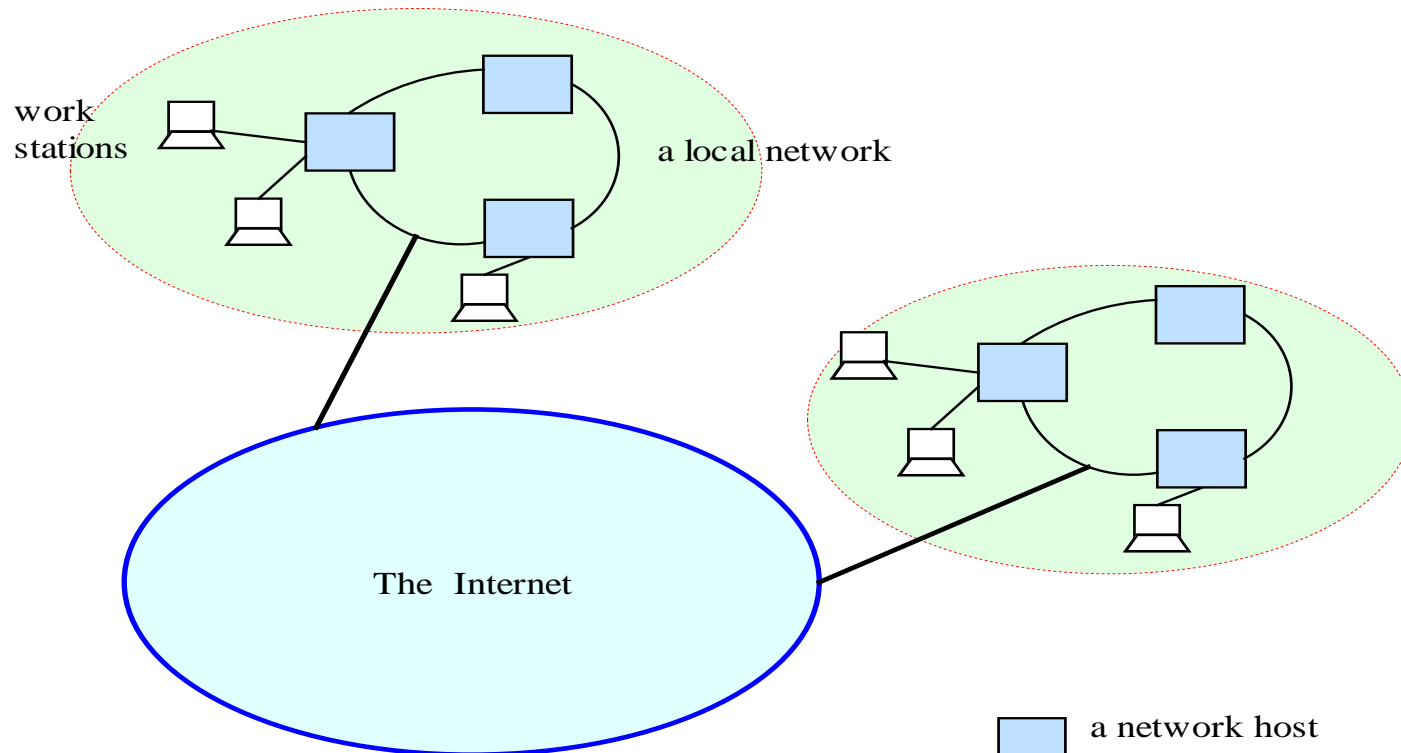
Distributed System

- A distributed system is a collection of independent computational resources and are connected with each other with the help of networks, and it is capable of resolving a task in a collaborative manner.

Distributed Computing

- Distributed computing deals with computational tasks to be performed on distributed system(s). Distributed computing uses multiple independent heterogeneous computational resources, which are communicating with each other over a network to perform a task. A distributed computational task is likely to involve heterogeneous resources like computer hardware, programming languages, operating systems and other resources.

Distributed Systems



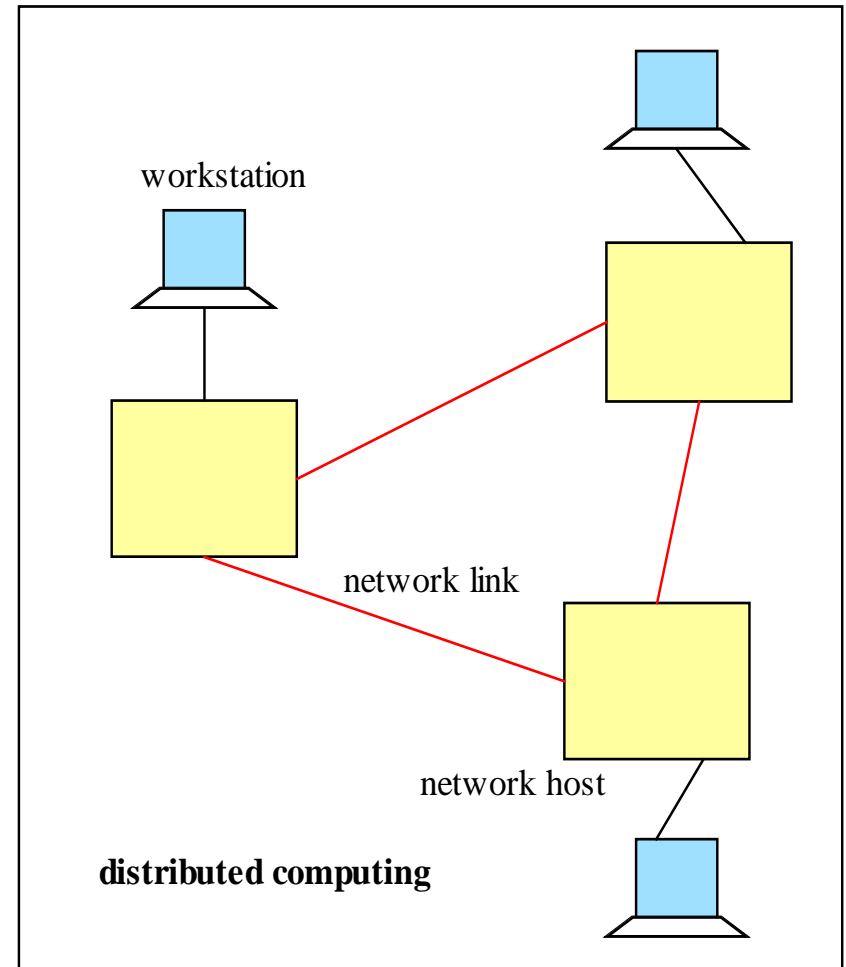
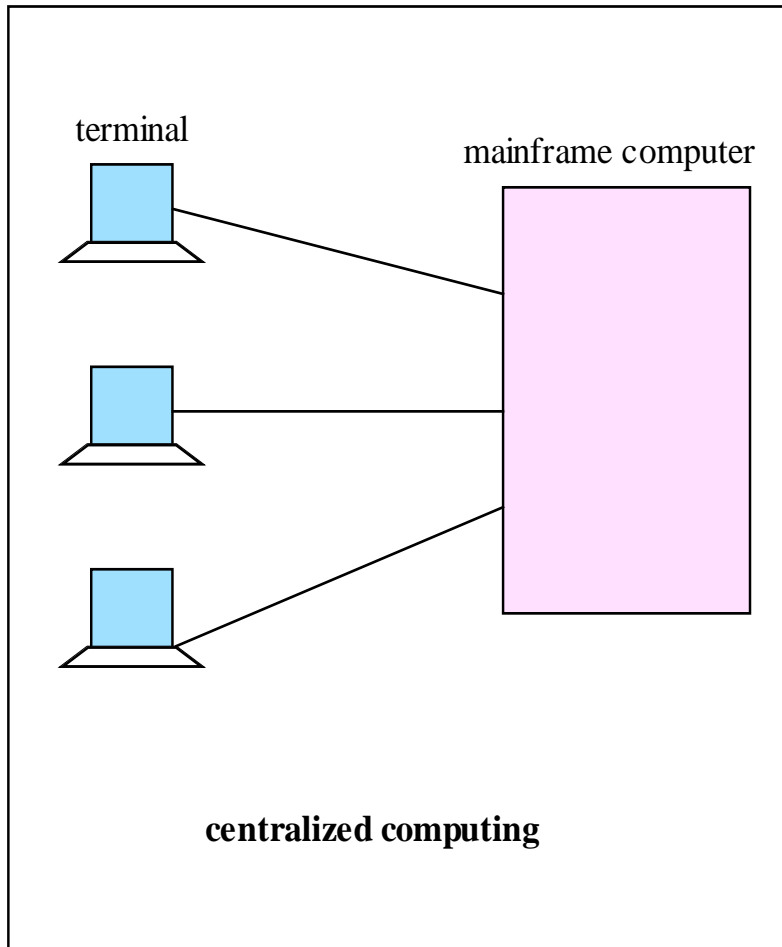
Computers in a Distributed System

- Workstations: computers used by end-users to perform computing
- Server machines: computers which provide resources and services
- Personal Assistance Devices: handheld computers connected to the system via a wireless communication link.

Examples of Distributed System:

- Network of workstations, Internet, Intranet

Centralized vs. Distributed Computing



Monolithic mainframe applications vs. distributed applications

based on <http://www.inprise.com/visibroker/papers/distributed/wp.html>

- **The monolithic mainframe application architecture:**
 - Separate, single-function applications, such as order-entry or billing
 - Applications cannot share data or other resources
 - Developers must create multiple instances of the same functionality (service).
 - Proprietary (user) interfaces
- **The distributed application architecture:**
 - Integrated applications
 - Applications can share resources
 - A single instance of functionality (service) can be reused.
 - Common user interfaces

Evolution of paradigms

- Client-server: Socket API, remote method invocation
- Distributed objects
- Object broker: CORBA
- Network service: Jini
- Object space: JavaSpaces
- Mobile agents
- Message oriented middleware (MOM): Java Message Service
- Collaborative applications

Why distributed computing?

- Economics: distributed systems allow the pooling of resources, including CPU cycles, data storage, input/output devices, and services.
- Reliability: a distributed system allow replication of resources and/or services, thus reducing service outage due to failures.
- The Internet has become a universal platform for distributed computing.

The Weaknesses and Strengths of Distributed Computing

In any form of computing, there is always a tradeoff in advantages and disadvantages

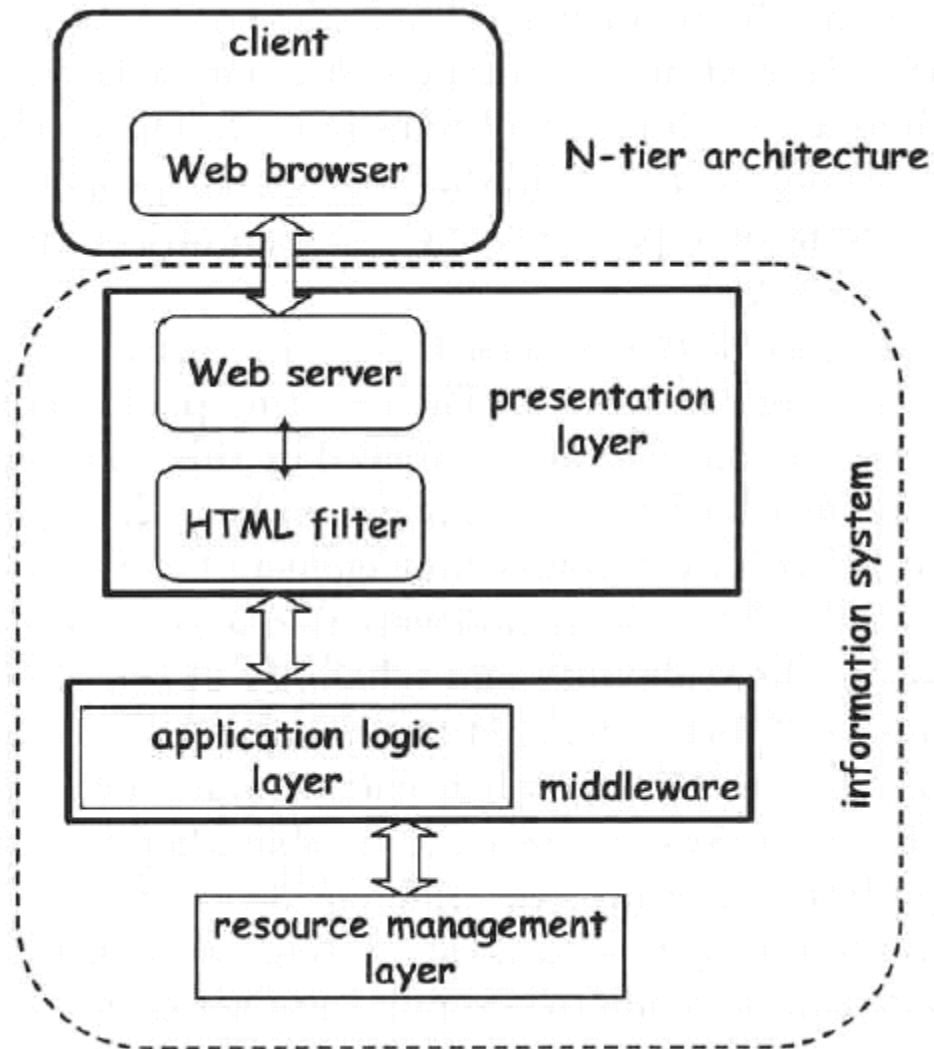
Some of the reasons for the popularity of distributed computing :

- **The affordability of computers and availability of network access**
- **Resource sharing**
- **Scalability**
- **Fault Tolerance**

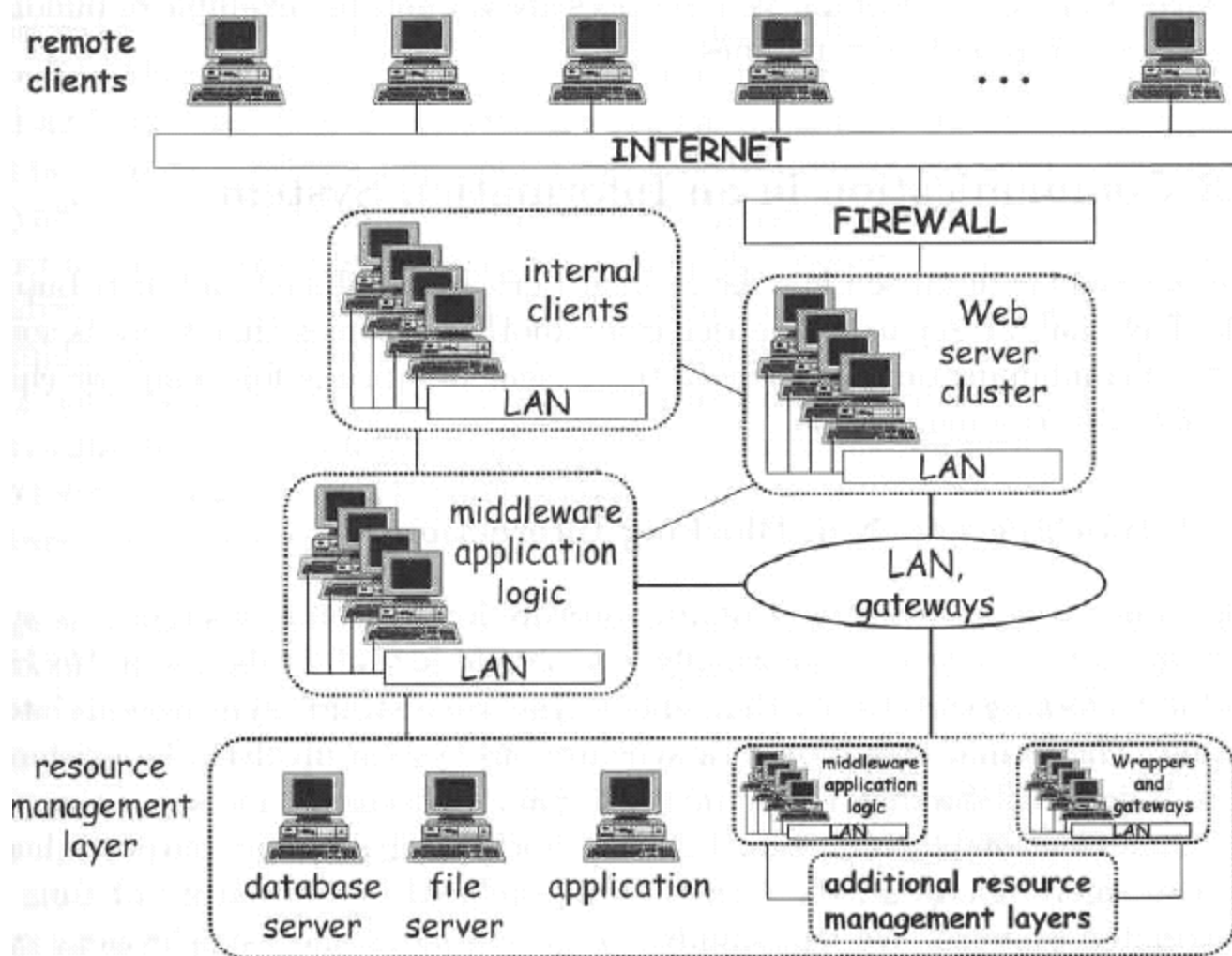
The Weaknesses and Strengths of Distributed Computing

The disadvantages of distributed computing:

- **Multiple Points of Failures:** the failure of one or more participating computers, or one or more network links, can spell trouble.
- **Security Concerns:** In a distributed system, there are more opportunities for unauthorized attack.



N-tier architecture by adding web server



N-tier Systems

Middleware

- Middleware facilitates and manages the interaction between applications across heterogeneous computing platforms.
- It is architectural solution to the problem of integrating a collection of servers and applications under a common service interface.
- Middleware offers programming abstraction that hide some of the complexities of building a distributed application.
- Through these programming abstractions, the developer has access to functionality that otherwise would have to be implemented from scratch.

Need for Middleware

- Example: There is a need to an application where part of the code is intended to run on one computer and another part must run on a different computer.
- Option – 1: To use sockets to open communication channel between the two parts of the application
 1. Need to worry about the channel itself:
 - Create the channel
 - Deal with any errors or failures
 2. Need to devise a protocol so that the two parts of the application can exchange information in an ordered manner. The protocol specifies who will be sending what, when and what is expected response.

Need for Middleware

3. Need to work out a format for the information being exchanged so that it can be correctly interpreted by both sides.
4. Once all above issues are resolved, we must develop an application that uses the communication channel:
 - Erroneous messages
 - Failure of application at other side of the channel
 - Recovery procedures to resume operations after failures
- Most of this work can be avoided by using middleware abstractions and tools.

Remote Procedure Call - RPC

Remote Procedure Call

The **remote procedure call** interface permits one to write client-server programs by calling functions on the client that appear to be functions on the server

- Developed by Sun Microsystems
- Provides a standard, partially automated method for some distributed computing tasks
- It is based on extending the notion of conventional, or local procedure calling, so that the called procedure need not exist in the same address space as the calling procedure.
- The two processes may be on the same system, or they may be on different systems with a network connecting them.
- By using RPC, programmers of distributed applications avoid the details of the interface with the network.

Remote Procedure Call

- RPC was introduced by Birell and Nelson in 1980 as a way to transparently call procedures on other machines, used to build 2-tier systems.
- RPC introduced the concept of
 - client,
 - interface definition language (IDL),
 - name and directory services,
 - dynamic binding,
 - service interface and so on.

Overview of RPCs

- Implicit network programming (sockets are explicit)
- Provides an API that looks like function calls

On the client:

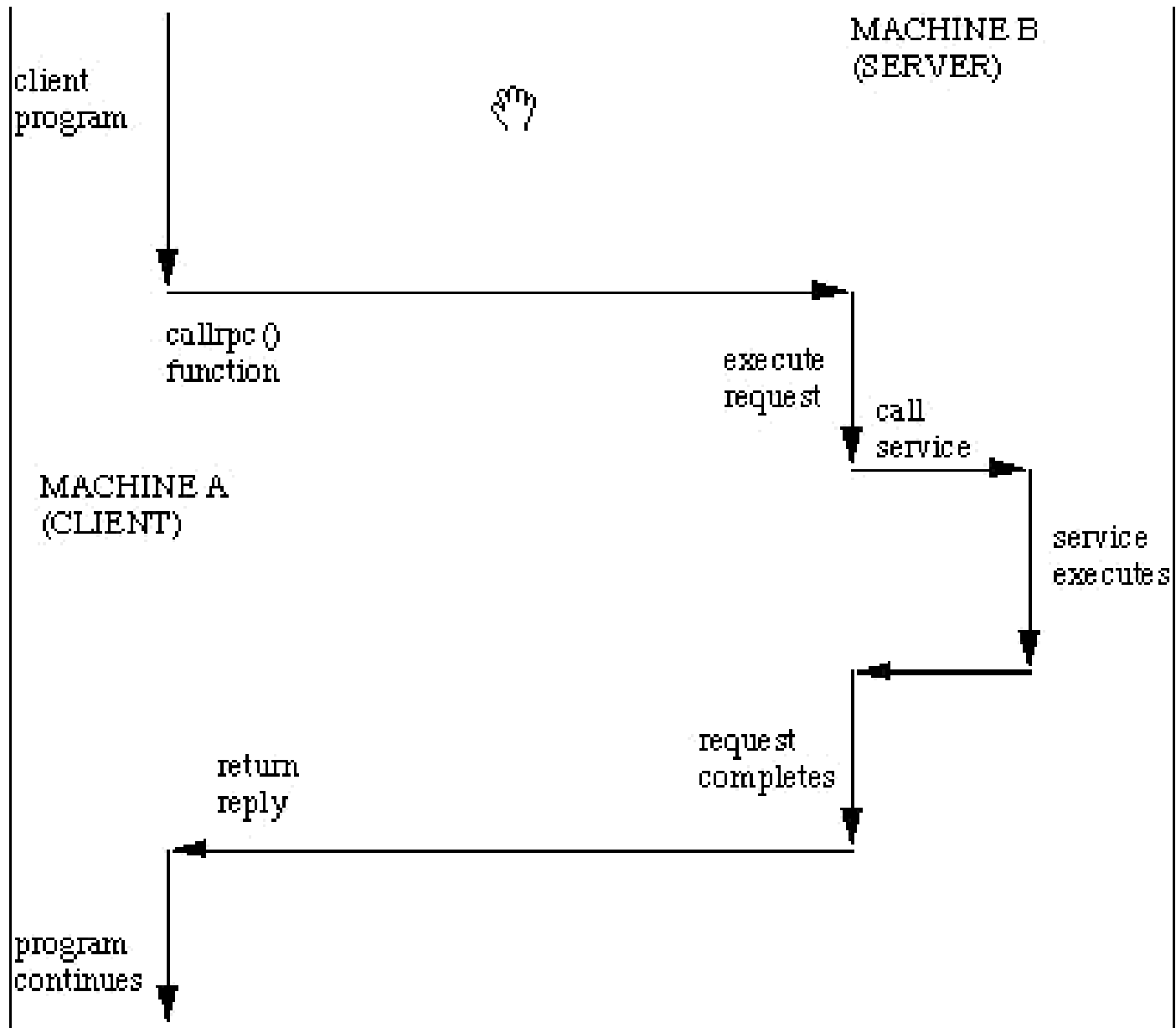
- Client stub procedure provides a function-call interface to applications running on the client
- Client stub uses sockets (datagram or stream) to interface with sockets on a server

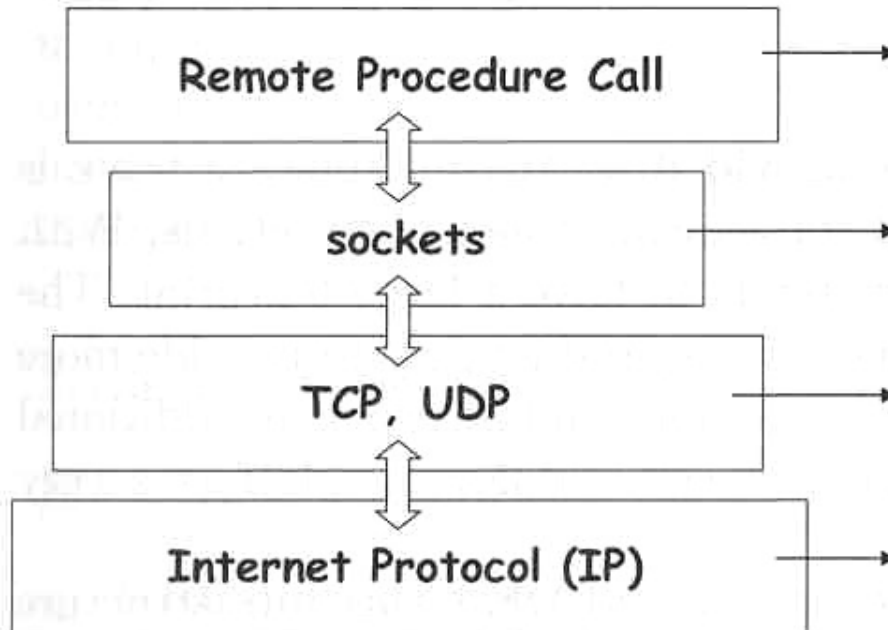
On the server:

- Server stub procedure interfaces to client stub through sockets
- Provides a procedure-like interface to application layer

RPC package handles:

- Reliability (retransmission, etc.)
- Data translation





Remote Procedure Call:

hides communication details behind a procedure call and helps bridge heterogeneous platforms

sockets:

operating system level interface to the underlying communication protocols

TCP, UDP:

User Datagram Protocol (UDP) transports data packets without guarantees

Transmission Control Protocol (TCP)

verifies correct delivery of data streams

Internet Protocol (IP):

moves a packet of data from one node to another

Types of middleware

RPC based systems

- Today, RPC systems are used as a foundation for almost all other forms of middleware, including Web Services middleware, e.g. SOAP provides a way to wrap RPC calls into XML messages exchanged through HTTP or some other transport protocol.

TP monitors

- It can be seen as RPC with transactional capabilities

Object brokers

- Most of them used RPC as underlying mechanism to implement remote object calls, e.g. CORBA

Object monitors

- TP monitors extended with OO interfaces

Message-oriented middleware

- Asynchronous RPC, TP monitors
- Queuing systems
- MOM: provides transactional access to the queues, persistent queues and a number of primitives for reading and writing to local and remote queues.

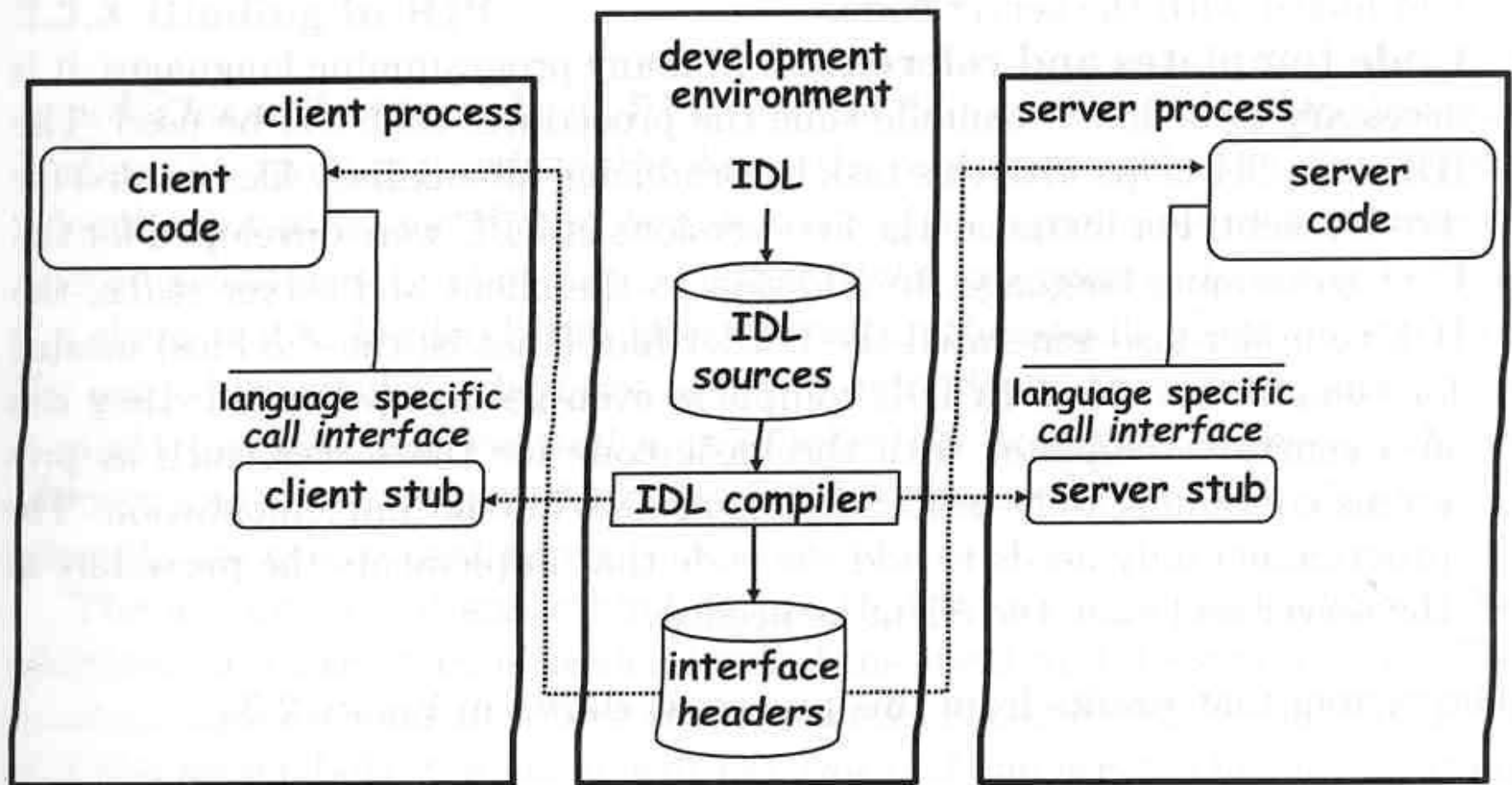
Message brokers

- Application logic can be attached to queues, thereby allowing designers to implement much more sophisticated interaction in an asynchronous manner.

How RPC works

1. To define IDL

- It provides an abstract representation of the procedure in terms of what parameters it takes as input and what parameters it returns as a response.



How RPC works

2. To compile IDL description

- Client stubs: every procedure signature in the IDL file results in a client stub.
- The stub is a piece of code to be compiled and linked with the client.
- When the client calls a remote procedure the call that is actually executed is a local call to the procedure provided by the stub.

How RPC works

The stub then takes care of:

- Locating server, i.e, binding the call to a server,
- Formatting the data appropriately, (which involves marshalling and serializing the data)
- Communicating with the server
- Getting a response.
- Forwarding that response as return parameter of the procedure invoked by the client

How RPC works


- Marshaling involves packing data into a common message format prior to transmitting the message over a communication channel, so that the message can be understood by the recipient.
 - An interprocess communication system may provide the capability to allow data representation to be imposed on the raw data.
 - Because different computers may have different internal storage format for the same data type, an external representation of data may be necessary.
 - Data marshalling is the process of (i) flattening a data structure, and (ii) converting the data to an external representation.
- Serialization consists of transforming the message into a string of bytes prior to sending the message through a communication channel.

Data Encoding Protocols

level of abstraction

data encoding schemes

Sample Standards



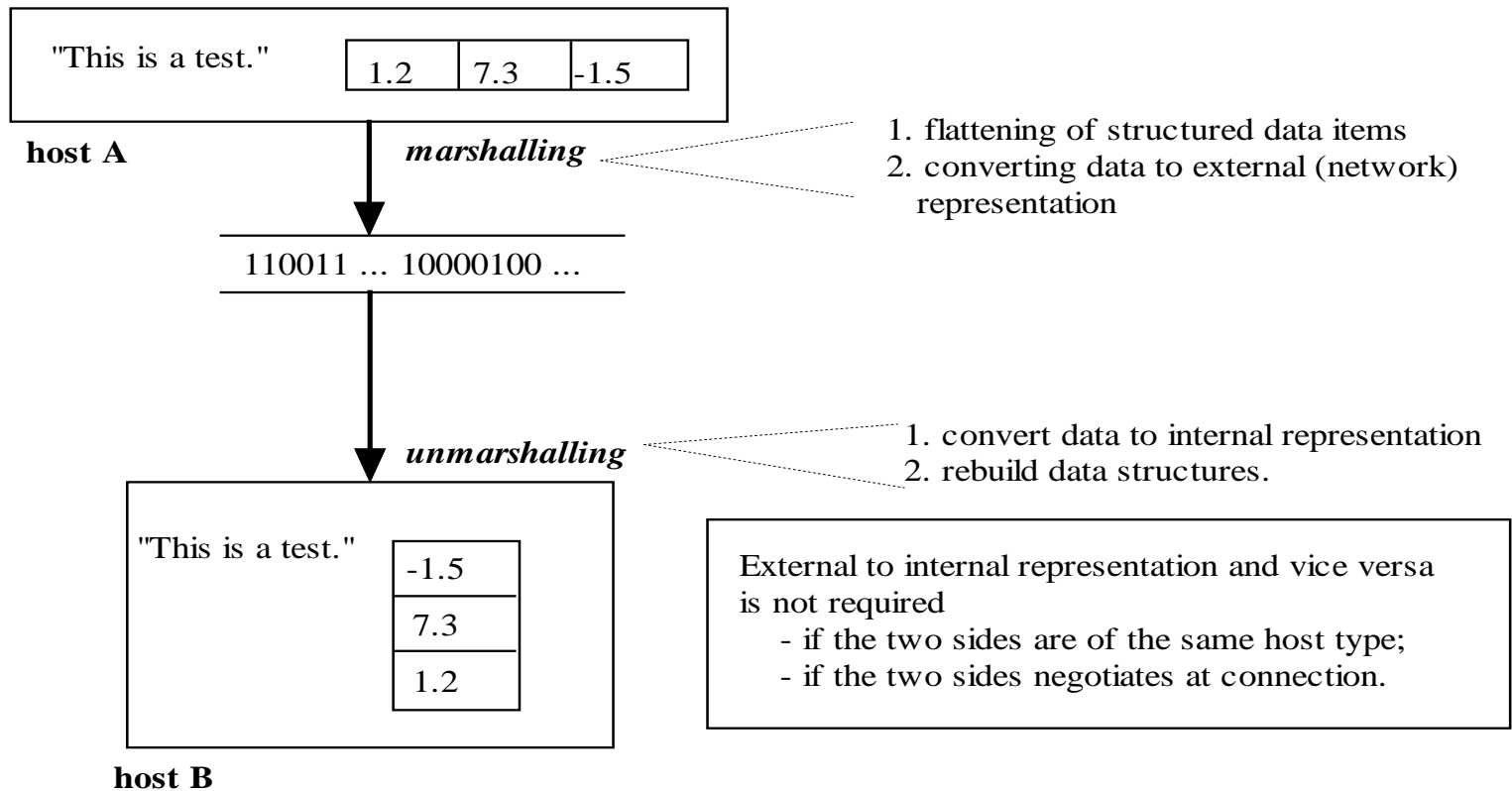
application specific data encoding language
general data encoding language
network data encoding standard

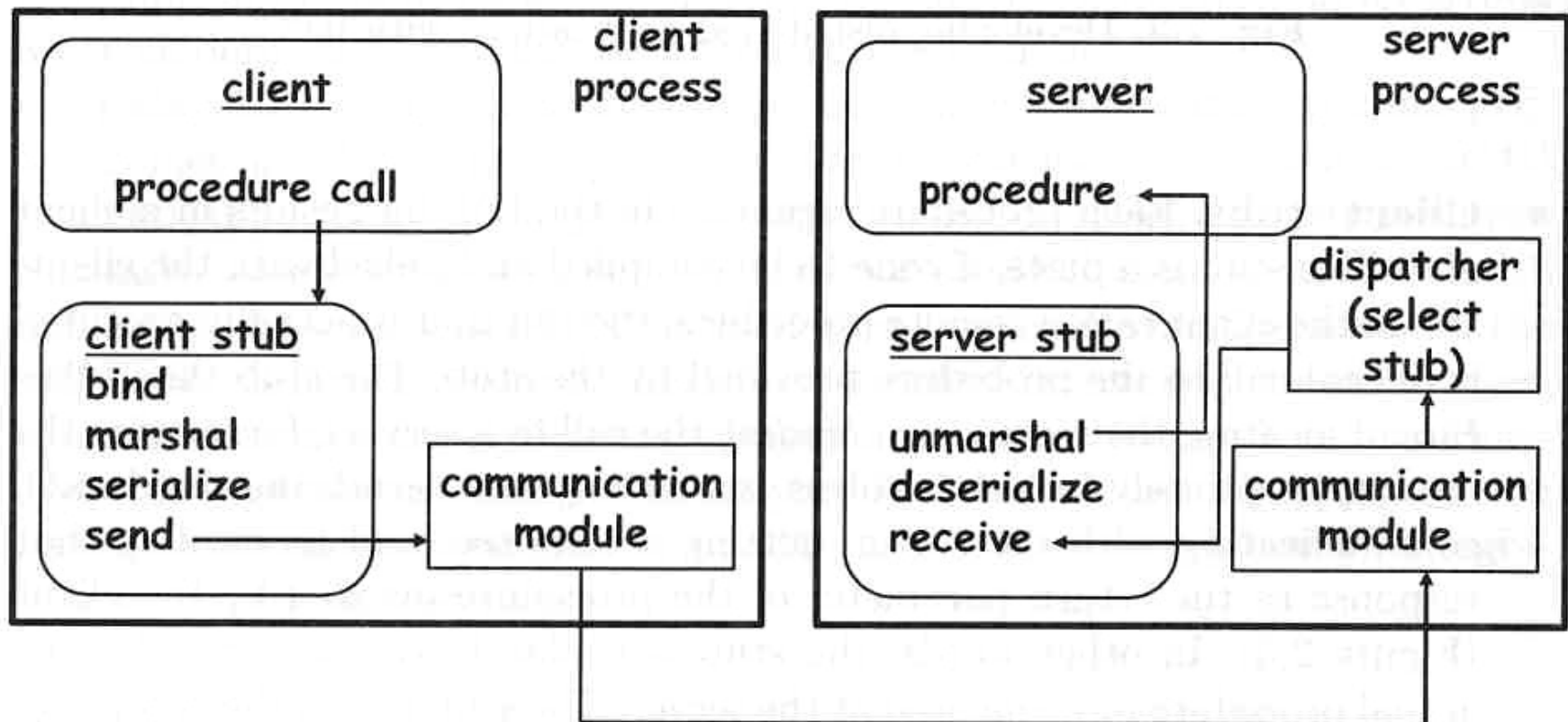
XML:(Extensible Markup Language)

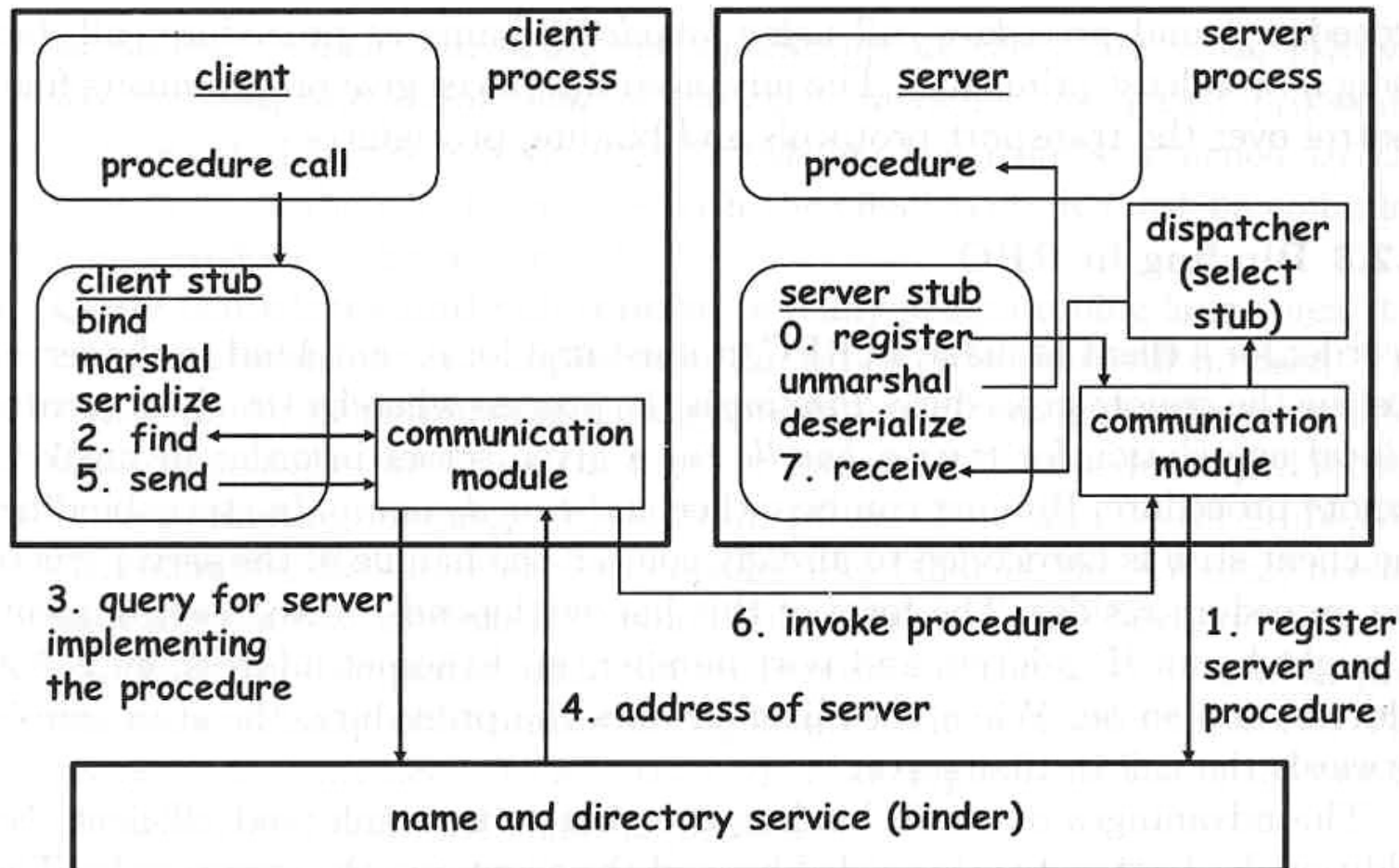
ASN.1(Abstract Syntax Notation)

Sun XDR(External Data Representation)

Data Marshalling







How RPC works

Client stubs

- The stub is a placeholder or proxy for the actual procedure implemented at the server.
- The stub makes the procedure appear as a normal local procedure, it does not implement the procedure.
- It implements all the mechanism necessary to interact with the server remotely for the purposes of executing that particular procedure.

Benefits of RPC

1. The programming is easier since there is little or no network programming involved. The application programmer just writes a client program and the server procedures that the client calls.
2. If an unreliable protocol such as UDP is used, details like timeout and retransmission are handled by the RPC package. This simplifies the user application.
3. The RPC library handles any required data translation for the arguments and return values. For example, if the arguments consists of integers and floating point numbers, the RPC package handles any differences in the way integers and floating point numbers are stored on the client and server.

Sample Application

rpcgen print.x

- rpcgen uses its own language (RPC language or RPCL) which looks very similar to preprocessor directives.
- rpcgen exists as a standalone executable compiler that reads special files denoted by a .x prefix.

cc client.c print_clnt.c -o client

cc server.c print_svc.c -o server

./server&

- From client PC:

./client *hostname*

rpcgen

The default output of rpcgen is:

- A header file of definitions common to the server and the client
- A set of XDR routines that translate each data type defined in the header file
- A stub program for the server
- A stub program for the client

rpcgen

rpcgen can optionally generate :

- Various transports
- A time-out for servers
- Server stubs that are MultiThreading safe
- Server stubs that are not main programs
- C-style arguments passing ANSI C-compliant code
- An RPC dispatch table that checks authorizations and invokes service routines

Converting Local Procedures to Remote Procedures

```
/* msg.x: Remote msg printing protocol */  
program MESSAGEPROG  
{ version PRINTMESSAGEVERS {  
  int PRINTMESSAGE(string) = 1;  
} = 1;  
} = 0x20000001;
```

Converting Local Procedures to Remote Procedures

- Remote procedures are always declared as part of remote programs. The code above declares an entire remote program that contains the single procedure PRINTMESSAGE.

In this example, PRINTMESSAGE procedure is declared to be:

- the procedure 1,
in version 1 of the remote program
MESSAGEPROG, with the program number
0x20000001.

Converting Local Procedures to Remote Procedures

- Version numbers are incremented when functionality is changed in the remote program.
- Existing procedures can be changed or new ones can be added.
- More than one version of a remote program can be defined and a version can have more than one procedure defined.

Remote Method Invocation - RMI

Object Serialization

- Object Serialization is the technique by which Object persistence is realized.
- It controls how data that comprises an object's state information: member variables, whether public, private or protected is written as a sequence of bytes.
- The serialized object might be sent over a network (thru RMI), or saved to a disk so that it can be accessed at some point in the future.
- This allows objects to move from one JVM to another.

Object Serialization

- Serialization works by examining the variables of an object and writing primitive data types like numbers and characters to a byte stream.
- An Object may contain an object as a member variable.
- The object member variable would cease to function correctly if the object was left out, so the variable must be serialized as well.
- The set of all objects referenced is called a graph of objects, and object serialization converts entire graph to byte form.

Distributed Objects

- In classic client-server the server program listens on a fixed port.
- The client program connects to that port, and can then send messages (strings) to the server program.
- In distributed objects, the client object will send a method to the server object.
- The programmer does not deal with opening sockets and connecting to remote machines.

Issues

- How does the client find the server?
- Network connections must be made but to where and by who?
- How does the client object interact with the server object?
- How can we send a method request across the network?

Client-Dispatcher-Server

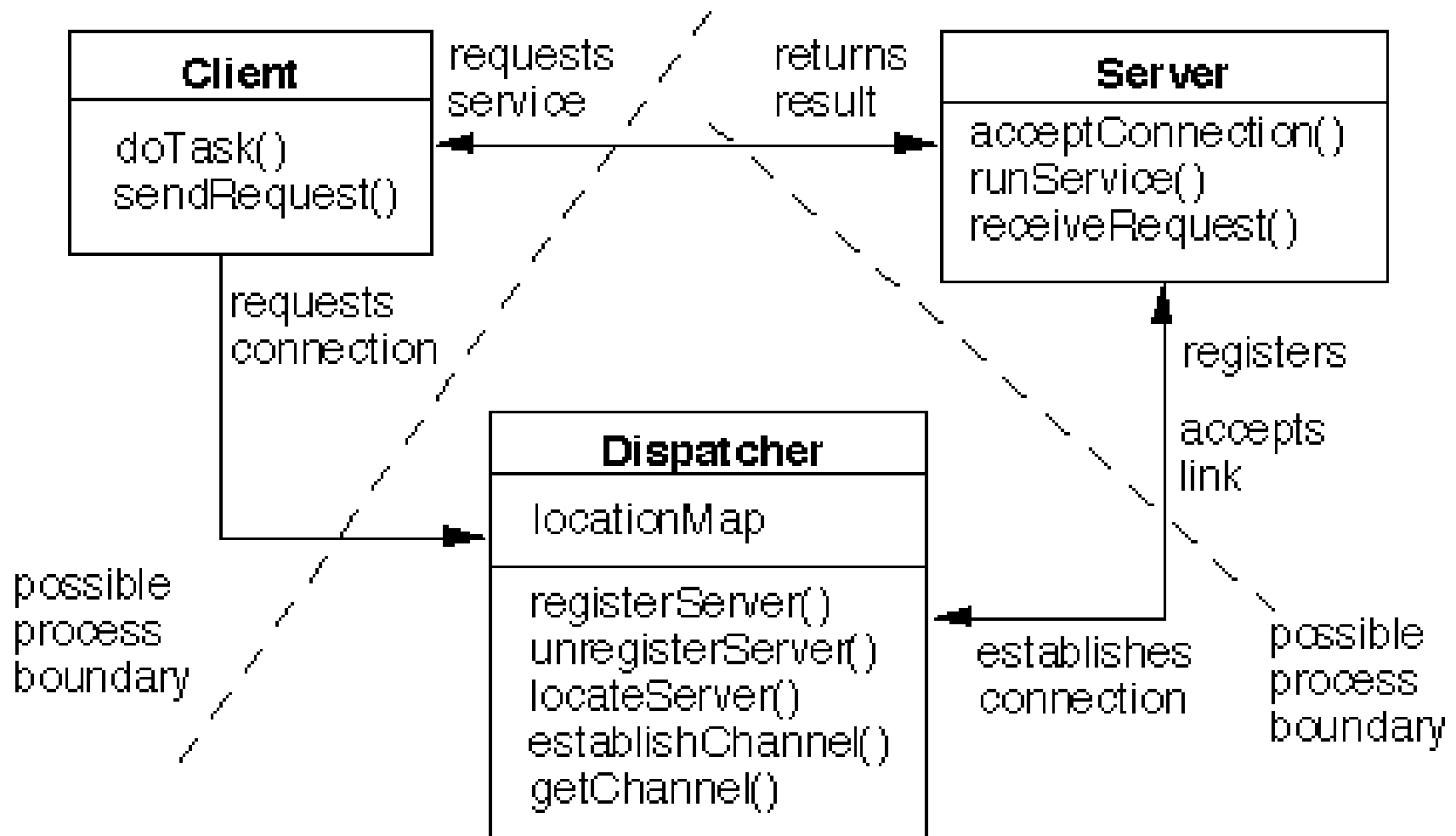
- A software system integrating a client with a set of distributed servers, with the servers running locally or distributed over a network

Problem

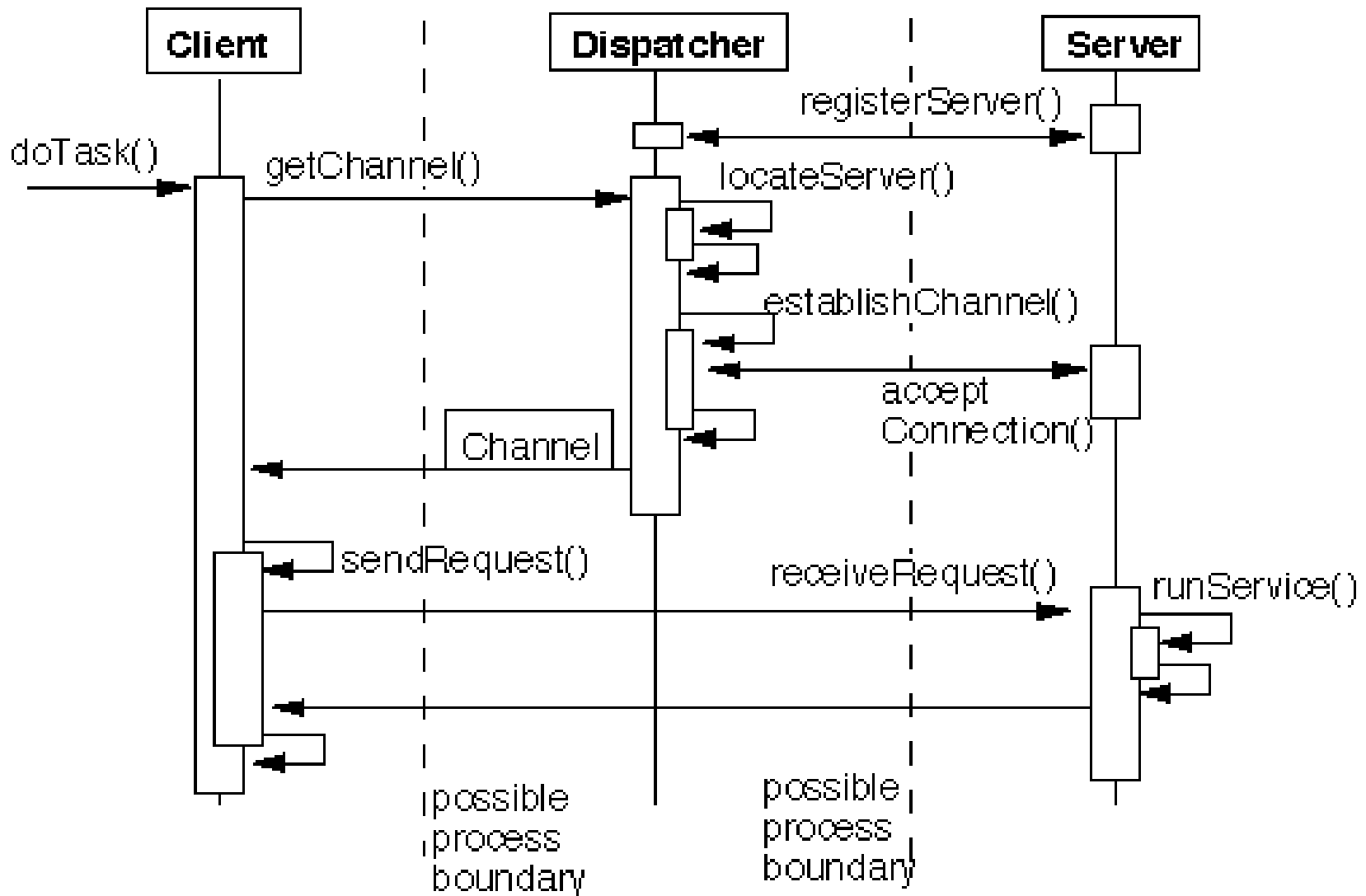
How to find the servers?

- A client should be able to use a service independent of the location of the service provider (server)
- The code implementing the functional core of a client should be separate from the code use to establish a connection with the server

Structure



Dynamics



Platforms available

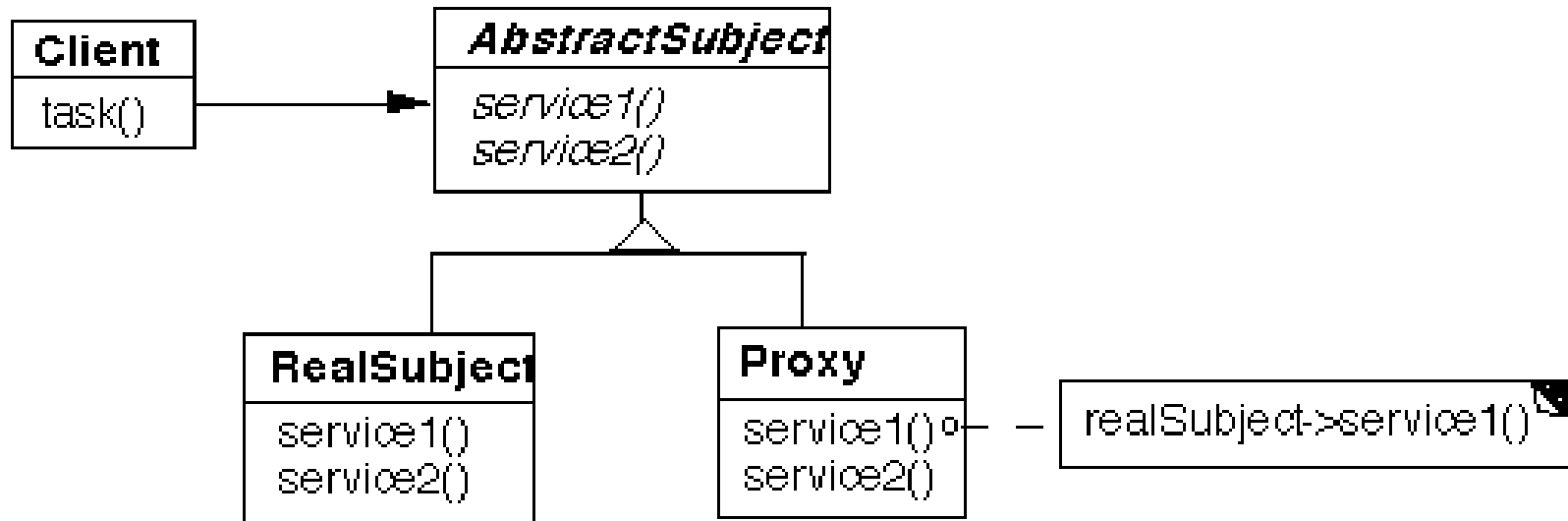
- Sun's implementation of remote procedure calls (RPC)
- Java's RMI
- OMG CORBA
- Microsoft .NET
- Sun's J2EE

Proxy

- The agency for a person who acts as a substitute for another person, authority to act for another.

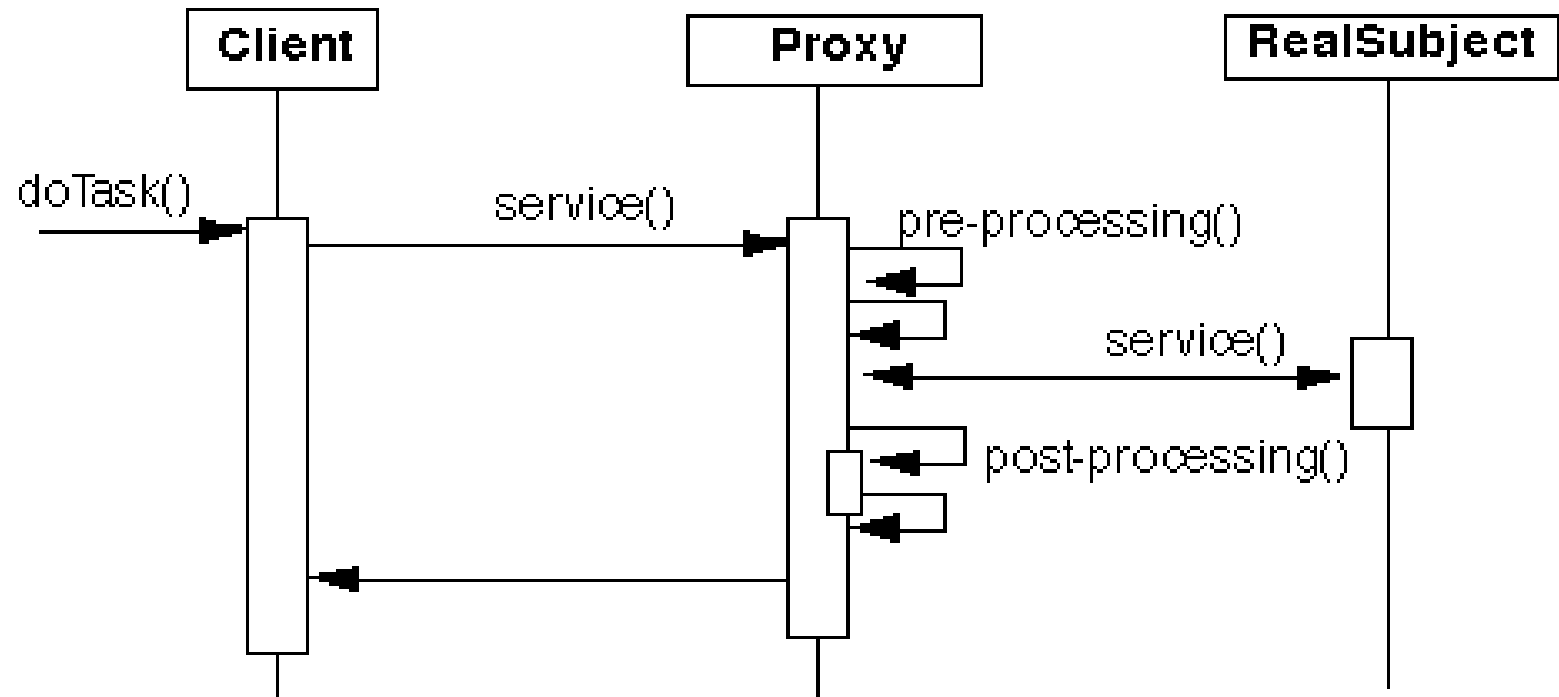
The Pattern

- The proxy has the same interface as the original object
- Use common interface (or abstract class) for both the proxy and original object
- Proxy contains a reference to original object, so proxy can forward requests to the original object



Structure

Dynamics



Reasons for Object Proxies

Remote Proxy

- The actual object is on a remote machine (remote address space)
- Hide real details of accessing the object
- Used in CORBA, Java RMI, RMI-IIOP

Reasons for Object Proxies

Virtual Proxy

- Creates/accesses expensive objects on demand
- You may wish to delay creating an expensive object until it is really accessed
- It may be too expensive to keep entire state of the object in memory at one time

Protection Proxy

- Provides different objects different level of access to original object

Cache Proxy (Server Proxy)

- Multiple local clients can share results from expensive operations: remote accesses or long computations

Firewall Proxy

- Protect local clients from outside world

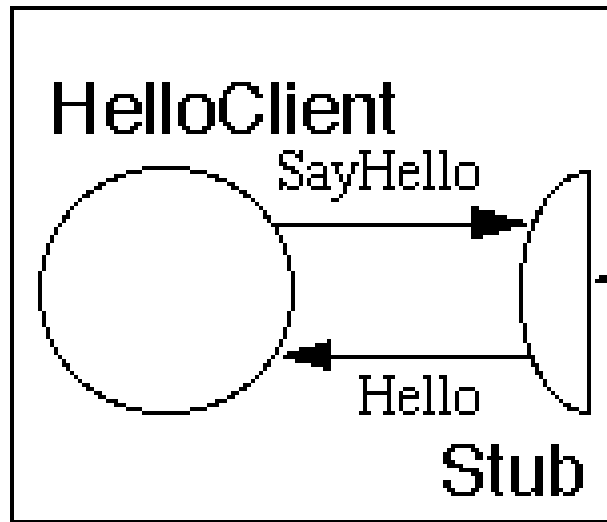
Synchronization Proxy

- Synchronize multiple accesses to real subject

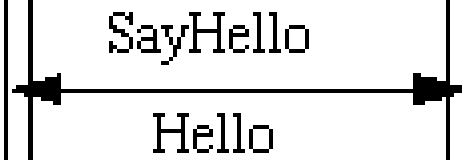
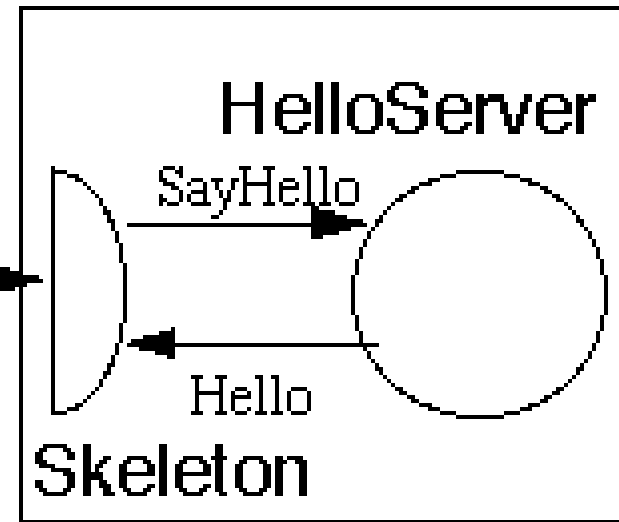
Java RMI

- First program

Machine A



Machine B



Example - 1

Example: HelloServer

Server Side

Step 1: Compile the source code

- Server side needs interface Hello and class HelloServer

```
[sanjay@dslabsrv17 HelloWorld]$ javac HelloInterface.java  
HelloServer.java
```

```
[sanjay@dslabsrv17 HelloWorld]$ Javac RegisterIt.java
```

```
[sanjay@dslabsrv17 HelloWorld]$
```

Example - 1

Step 2 . Generate Stubs and Skeletons

The rmi compiler generates the stubs and skeletons

```
[sanjay@dslabsrv17 HelloWorld]$ rmic -v1.2 -verbose
HelloServer
[loaded ./HelloServer.class in 0 ms]
[loaded
  /opt/jdk1.5.0_07/jre/lib/rt.jar(java/rmi/server/UnicastRemoteObject.class) in 1 ms]
[loaded
  /opt/jdk1.5.0_07/jre/lib/rt.jar(java/rmi/server/RemoteServer.class) in 1 ms]
```


Example - 1

- This produces the files (as JDKv1.2 is selected):

HelloServer_Stub.class

For Other versions following files will be created:

HelloServer_Skel.class

HelloServer_Stub.class

- The Stub is used by the client and server
The Skel is used by the server

Example - 1

Step 3

Run RMI Registry, For the default port number

```
[sanjay@dslabsrv17 HelloWorld]$  
rmiregistry&
```

For a specific port number

`rmiregistry portNumber &`

```
[sanjay@dslabsrv17 HelloWorld]$ rmiregistry 8090&  
[1] 23613
```

Example - 1

```
[sanjay@dslabsrv17 HelloWorld]$ netstat -a | more
```

Active Internet connections (servers and established)

Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State
tcp	0	0	*:32768	*:*	LISTEN
tcp	0	0	localhost.localdo:32769	*:*	LISTEN
tcp	0	0	*:1099	*:*	LISTEN
tcp	0	0	localhost.localdoma:783	*:*	LISTEN
tcp	0	0	*:sunrpc	*:*	LISTEN

Example - 1

```
[sanjay@dslabsrv17 HelloWorld]$ netstat -a
```

Active Internet connections (servers and established)

Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State
tcp	0	0	*:32768	*:*	LISTEN
tcp	0	0	localhost.localdo:32769	*:*	LISTEN
tcp	0	0	localhost.localdoma:783	*:*	LISTEN
tcp	0	0	*:sunrpc	*:*	LISTEN
tcp	0	0	*:x11	*:*	LISTEN
tcp	0	0	*:ssh	*:*	LISTEN
tcp	0	0	localhost.localdoma:ipp	*:*	LISTEN
tcp	0	0	localhost.localdom:smtp	*:*	LISTEN
tcp	0	0	*:8090	*:*	LISTEN
tcp	0	0	*:8443	*:*	LISTEN

Example - 1

Step 4: Register the server object with the rmiregistry by running `HelloServer.main()`

```
[sanjay@dslabsrv17 HelloWorld]$ java RegisterIt&  
[1] 24018
```

```
[sanjay@dslabsrv17 HelloWorld]$ Object  
  instantiatedHelloServer[UnicastServerRef [liveRef:  
    [endpoint:[10.100.84.17:32778](local),objID:[0]]]]  
HelloServer bound in registry
```

Important the above command will create a new thread that will not stop until you kill it!

Example - 1

Client side

- Place .java.policy in your home directory. **This is needed when running rmi in JDK 1.2, but not in JDK 1.1.x.**
- The client can be executed on the same machine or a different machine than the server

Step 1 . Compile the source code

- Client side needs interface Hello and class HelloClient

```
javac HelloInterface.java HelloClient.java
```

Example - 1

- **Step 2.** Make sure that the HelloServer_Stub.class is available
- **Step 3.** Run the client code

```
[sanjay@dslabsrv17 HelloWorld]$ java  
HelloClient
```

```
Hello World, the current system time is  
Mon Sep 04 13:38:49 IST 2006
```

Example - 1

Run HelloClient from a Window based computer:

- Copy .java.policy in the home directory of a user, e.g. C:\Documents and Settings\200512001

Example - 1

```
C:\Testing\HelloWorld>dir
```

```
Volume in drive C has no label.
```

```
Directory of C:\Testing\HelloWorld
```

```
09/04/2006  10:28a      <DIR>          .
09/04/2006  10:28a      <DIR>          ..
09/04/2006  10:20a             1,078 HelloClient.class
09/04/2006  01:30p             612 HelloClient.java
09/04/2006  10:18a             227 HelloInterface.class
07/29/2000  01:42p             221 HelloInterface.java
09/04/2006  01:00p             1,651 HelloServer_Stub.class
              5 File(s)              3,789 bytes
              2 Dir(s)          309,920,768 bytes free
```

```
C:\Testing\HelloWorld>java HelloClient
```

```
Hello World, the current system time is Mon Sep 04 13:57:01 IST
```

```
C:\Testing\HelloWorld>
```

Example - 1

Testing of HelloClient from another linux based computer:

```
[sanjay@dslab66 HelloWorld]$ ls -la
```

```
total 20
```

```
drwxr-xr-x      2 sanjay  student      4096 Sep  4 10:33 .
drwxr-xr-x      5 sanjay  student      4096 Aug 24  2005 ..
-rw-r--r--      1 sanjay  student      1078 Sep  4 10:20
    HelloClient.class
-rw-r--r--      1 sanjay  student        227 Sep  4 10:18
    HelloInterface.class
-rw-r--r--      1 sanjay  student      1651 Sep  4  2006
    HelloServer_Stub.class
```

```
[sanjay@dslab66 HelloWorld]$ export
```

```
    CLASSPATH=$CLASSPATH:/home/sanjay/rmi/HelloWorld/:
```

```
[sanjay@dslab66 HelloWorld]$ java HelloClient
```

```
Hello World, the current system time is Mon Sep 04 14:07:42 IST 2006
```

```
[sanjay@dslab66 HelloWorld]$
```

Example - 2

- Example: HelloServerSecured
- Server Side

Step 1

Install a policy file for socket permissions

- Place the following in a file called “.java.policy” in your home directory.
- **This is needed when running rmi in JDK 1.2, but not in JDK 1.1.x.**

Example - 2

```
grant    {           permission
  java.net.SocketPermission "*:1024-
  65535",
  "connect,accept,resolve";
  permission java.net.SocketPermission
  "*:1-1023",
  "connect,resolve";    };
```

Example - 2

Step 2: Compile the source code

- Server side needs interface Hello and class HelloServer

```
javac Hello.java HelloServer.java
```

Example - 2

Step 3 . Generate Stubs and Skeletons

The rmi compiler generates the stubs and skeletons

`rmic HelloServer`

- This produces the files:

`HelloServer_Skel.class`

`HelloServer_Stub.class`

- The Stub is used by the client and server
The Skel is used by the server

Example - 2

Step 4

Run RMI Registry

For the default port number

`rmiregistry &`

- For a specific port number

`rmiregistry portNumber &`

Example - 2

Step 4: Register the server object with the rmiregistry by running `HelloServer.main()`

`java HelloServer &`

- **Important** the above command will create a new thread that will not stop until you kill it!

Example - 2

- Client side
- The client can be executed on the same machine or a different machine than the server

Step 1 . Compile the source code

- Client side needs interface Hello and class HelloClient

```
javac Hello.java HelloClient.java
```

Example - 2

- **Step 2.** Make the HelloServer_Stub.class is available
- **Step 3.** Run the client code

```
java HelloClient
```

Example-3: Weather Information

```
[sanjay@dslabsrv17 weather]$ javac Weather.java
```

```
[sanjay@dslabsrv17 weather]$ javac  
WeatherInterface.java
```

```
[sanjay@dslabsrv17 weather]$ javac  
WeatherServer.java
```

```
[sanjay@dslabsrv17 weather]$ rmic WeatherServer
```

```
[sanjay@dslabsrv17 weather]$ java WeatherServer&
```

```
[3] 1632
```

```
C:\Testing\Weather>java WeatherClient 10.100.84.17
//10.100.84.17:4711/WeatherServer
List of Commands to get Weather Information :
all - Get all information available
wind - Get the wind speed
temp - Get the temperature
type - Get the weather type (for example "dry or cloudy"
help - View these commands again
quit - Quit the connection and the program.
```

```
>> all
```

```
-----
Todays weather:
-----
```

```
The temperature is 18 degrees Celsius
The wind is blowing with 10 m/s
It is raining...
There are some clouds in the sky.
The air is white and cold. It is a bit foggy.
```

```
>> wind
```

```
The wind blows with 10 m/s.
```

```
>> type
```

```
Today we have rain and clouds and fog.
```

```
>> temp
```

```
Temperature: 18 degrees celsius.
```

```
>> quit
```

```
Thank you for using Weather Information !
```

```
[sanjay@dslabsrv17 weather]$ netstat -a | more
Active Internet connections (servers and
    established)
```

Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State
tcp	0	0	*:32768		*:*
LISTEN					
tcp	0	0	localhost.localdo:32769		*:*
LISTEN					
tcp	0	0	*:4711		*:*
LISTEN					

Basic Issues

Multiple JVMs Running

- After testing (running and rerunning and rerunning and ...) your server you may end up with many JVMs that will not quit once you log out.
- It is very easy to accumulate lots of used orphaned processes.

Basic Issues

- Use the command:

```
/usr/bin/ps -o pid,stime,comm -usanjay
```

- Which will find all my processes
- Put the following in a file, make it executable
- Running the file will then kill all your Java processes

```
kill ` /usr/bin/ps -o pid,comm -u$USER |  
    egrep java | awk '{print $1}'`
```

Basic Issues

Port Contention

- Only one server can use a port at a time!
- Not everyone can use the same port number for the RMI registry.
- The RMI HelloServer example runs the RMI registry on the default port 1099
- You will need to find a port that is unused for your server!
- Ports you use must be in the range: 5001-65536

Run RMI registry on open Port

Enterprise Application Development and Deployment

Building Distributed Applications was Difficult

- Need to support:
 - Transactions,
 - resource-pooling,
 - security,
 - threading,
 - persistence,
 - life-cycle, etc...
- System programming at the expense of business logic
- Developers had to become specialists
- Proprietary APIs resulted in non-portable code

Problems in Scaling an Application Client

- communication overhead
- delay in accessing servers
- issues of priority and fairness
- synchronization
- running on different platforms
 - based on USER needs

Problems in Scaling an Application Server

- need for independence from client
- variety of client demands
- scalability
- transparency to client
- hardware based on
 - performance
 - infrastructure needs

Problems in Scaling an Enterprise System

- transaction service (commit/rollback)
- security
- load balancing
- thread management
- persistence
- middleware
- accounting and logging
- migrating from legacy systems

Common Object Request Broker Architecture - CORBA

- CORBA was developed by a consortium of companies (the Object Management Group) during the early 1990s to provide a common, language- and vendor-neutral standard for object distribution.
- CORBA as an architecture has been well accepted and successfully used in many projects.
- The CORBA architecture is built around a special layer, the object request broker (ORB), that facilitates communication between clients and objects.
- The ORB is responsible for handling the object requests from a client and passing over the parameters from method invocations.

Common Object Request Broker Architecture - CORBA

- Low-level communication between different object spaces (ORBs) is done by using the Internet-Inter ORB Protocol (IIOP).
- By using this standard protocol, a CORBA-based program from any vendor, on almost any computer, operating system, programming language, and network can interoperate with a CORBA-based program from the same or another vendor, on any other computer, operating system, programming language, and network.

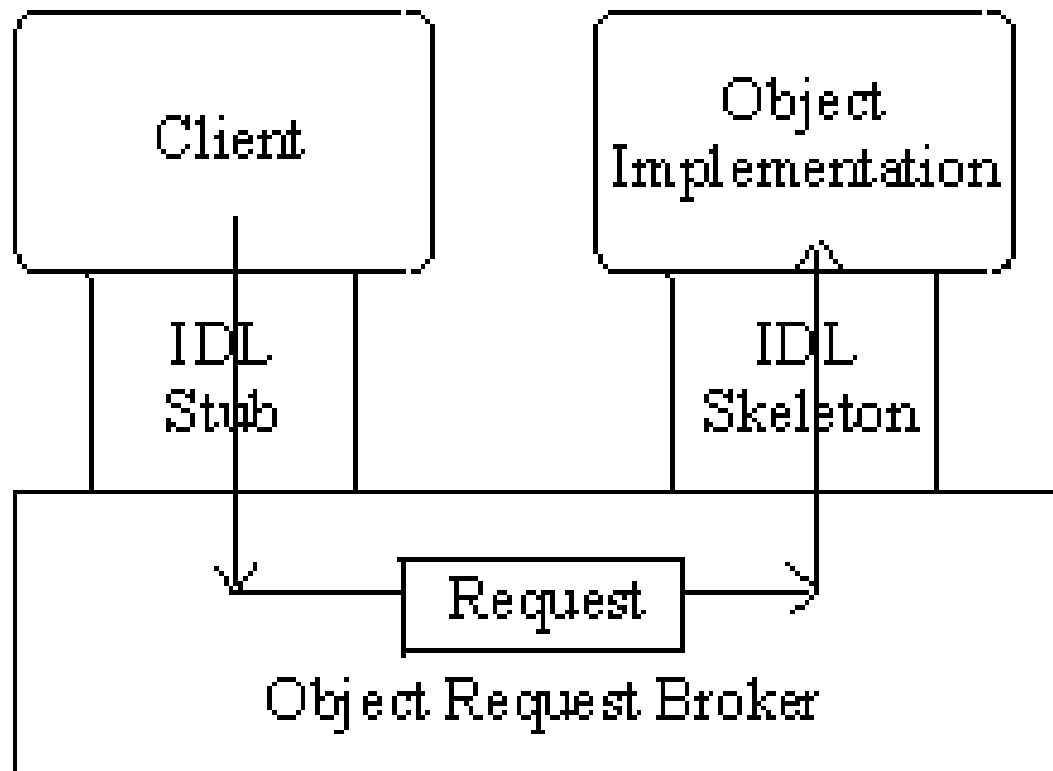


Figure 1: A request passing from client to object implementation

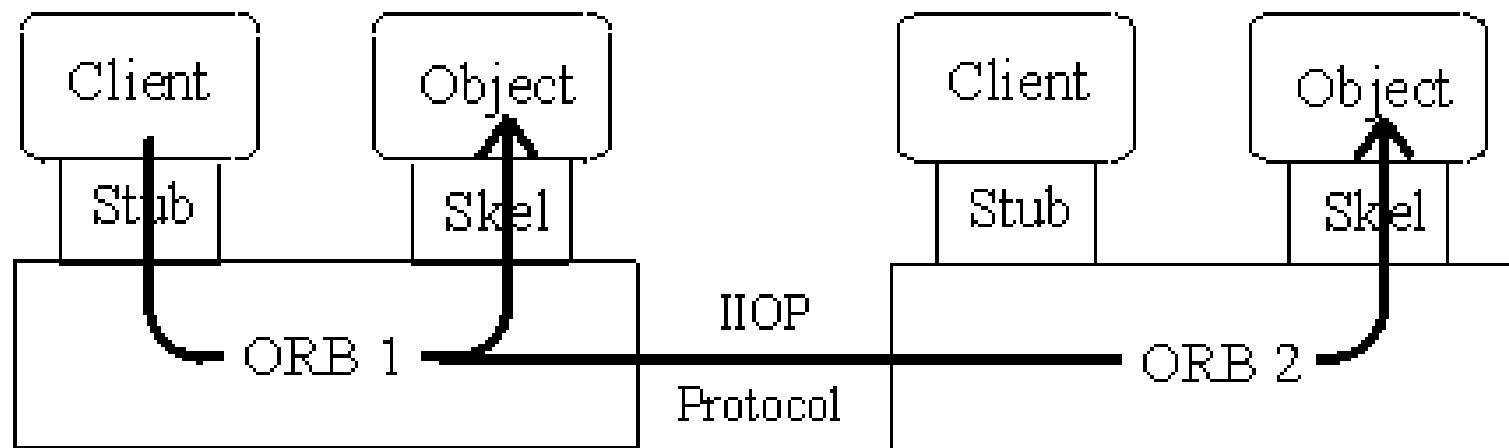


Figure 2: Interoperability uses ORB-to-ORB communication

Copyright © 2000 Object Management Group

CORBA

- Apart from the ORB, there are two other key building blocks in the CORBA model, the Interface Definition Language (IDL), which normalizes the differences caused by language or operating system dependencies; and the CORBA services, which provide standard ways for CORBA objects to interact, such as naming and transaction.
- It enables interoperability between different vendors' products, and that CORBA is language neutral. CORBA clients and servers can be written in a variety of computer languages, including Java, C++, C, Smalltalk, and Ada.
- This is possible by implementing remote interfaces for the CORBA distributed objects in IDL.

CORBA

- But when using CORBA to build distributed systems in Java, the development effort is higher, because many parts of the system have to be implemented in two languages: IDL and Java.
- The development tools and runtime environment for CORBA applications can also be expensive and may not fully implement the CORBA services.

RMI over IIOP (RMI-IIOP)

- RMI over IIOP (RMI-IIOP) combines the best features of RMI with those of CORBA. Like RMI, RMI-IIOP allows developers to use only Java.
- Developers do not have to develop in both Java and IDL. RMI-IIOP allows developers to build classes that pass any serializable Java object as remote method argument or return value.
- By using IIOP as communication protocol, RMI-IIOP applications are interoperable with other CORBA applications.
- The synthesis of these two technologies results in a unique combination of power and ease of use, the Enterprise JavaBeans.

Defining J2EE

- The Java 2 Platform, Enterprise Edition (J2EE) specification describe a services-based application architecture within which
 - Transactional (DB updates),
 - Scalable (able to handle very large number of potential & simultaneous users),
 - Secure (users must be authorized),
 - Reliable (able to withstand planned and unplanned component failures)
 - portable Java components can be deployed and redeployed.

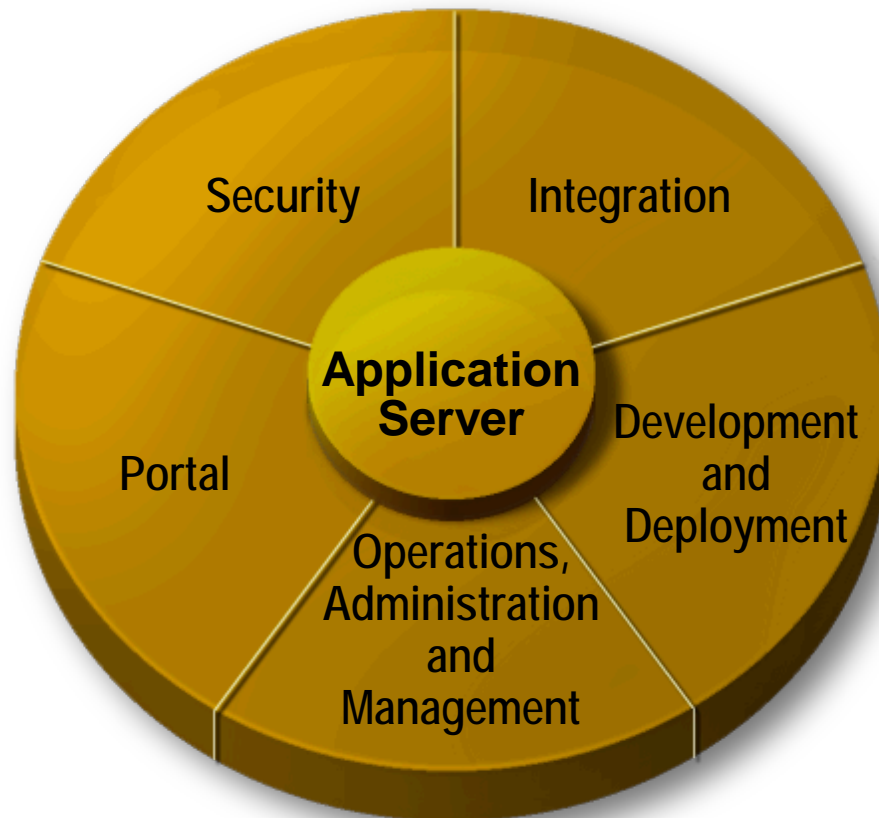
Java 2 Platform, Enterprise Edition (J2EE)

- J2EE defines an ARCHITECTURE for developing complex, distributed java applications
- Consists of:
 - Design guidelines for developing enterprise applications using J2EE
 - A reference implementation to provide an operational view of J2EE
 - A compatibility test suite for compliance testing of third party products
 - Several APIs (Application Programming Interfaces)
 - Technologies to simplify enterprise Java Development

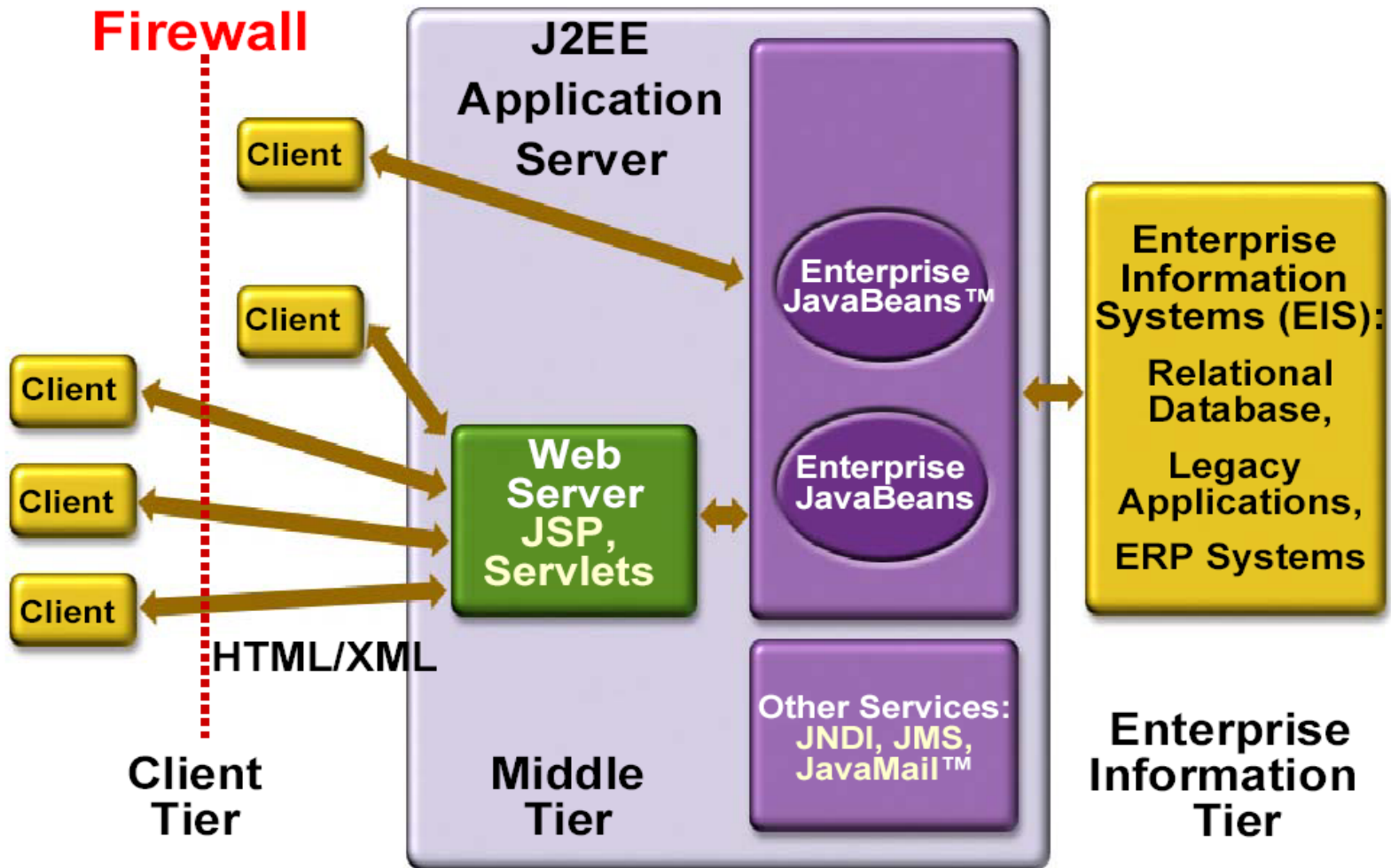
J2EE Application Server

- The Application server handles all system level programming
 - Security
 - Authorization
 - Authentication
 - Transactions
 - Threading
 - Object life time management
 - Caching
 - Object persistence
 - Database Connection pooling

A Typical J2EE Server



Several Clients one System



J2SE→J2EE

J2SE and J2EE

Basic libraries for java
development

I/o
GUIs
applets
etc

Based on J2SE

Can UTILIZE EJB

Uses other technologies

Defines a Specification

Components are constrained
to abide by **interface
specs** if they abide J2EE

Changes in J2EE 1.3 Specification

- New Container managed Persistence Model
- Support for Message Driven Beans
- Support for Enterprise Local Beans
- Finally, J2EE1.3 requires Support for J2SE1.3

Changes in J2EE 1.4 Specification

- Support for Web Services
- JAX-RPC and SAAJ APIs provide the basic web services interoperability support
- JAXR API support access to registries and repositories
- JMX API supports J2EE Management API

Benefits of the J2EE Approach

Allows developers to develop systems without regard of

- the operating system or hardware technology platforms (**platform independence**)
- The application server software that will be used to implement (execute) the business system (**application independence**)
- The physical locations from which the business system will be accessed (**location transparency**)

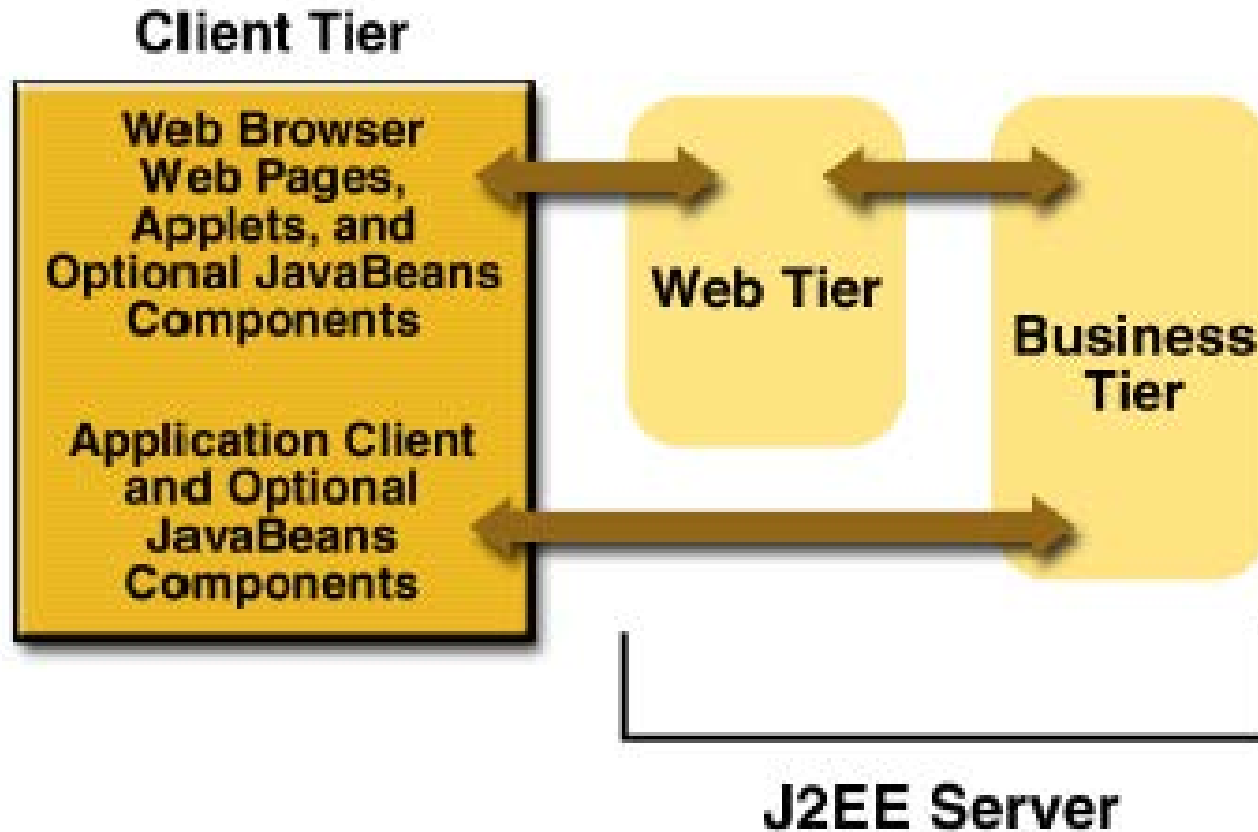
Benefits of the J2EE Approach

- Allows developers to specify the resources a business system will employ and set specific levels for these resources without having to write elaborate lines of code to achieve this (**attribute based programming**):
 - Example: developers can specify:
 - Security levels for different users of the system
 - How business system will connect to database and access data, access remotely located objects, manage transactions, intercommunicate with other component, etc
- Allows for Higher productivity of systems development team (cost-effective, rapid, re-use of pre-built components)

Benefits of the J2EE Approach

- Facilitates componentization in many ways:
 - J2EE offers a well thought-out approach of separating the development aspects of a component from its assembly specifics
 - J2EE offers a wide range of APIs that can be used for accessing and integrating products from third-party vendors, creating a market for software components
 - J2EE offers function-specific or highly specialized components optimized for specific types of roles

J2EE Communication as Tiers

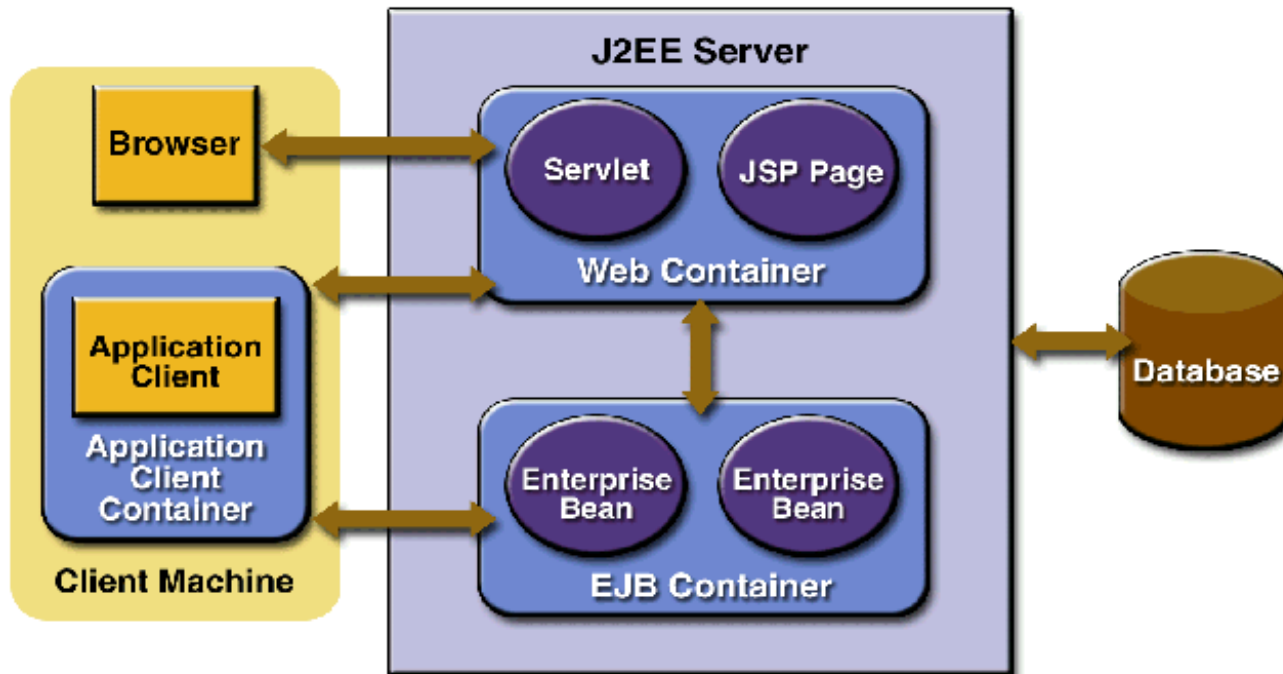


The Container Concept

- A Container is a software entity that runs within the server and is responsible for providing the execution environment for J2EE components
- A Container also manage the life-cycle of components deployed within it.
- The container is responsible for resource-pooling, enforcing security, and enforcing transaction management requirements

J2EE Containers

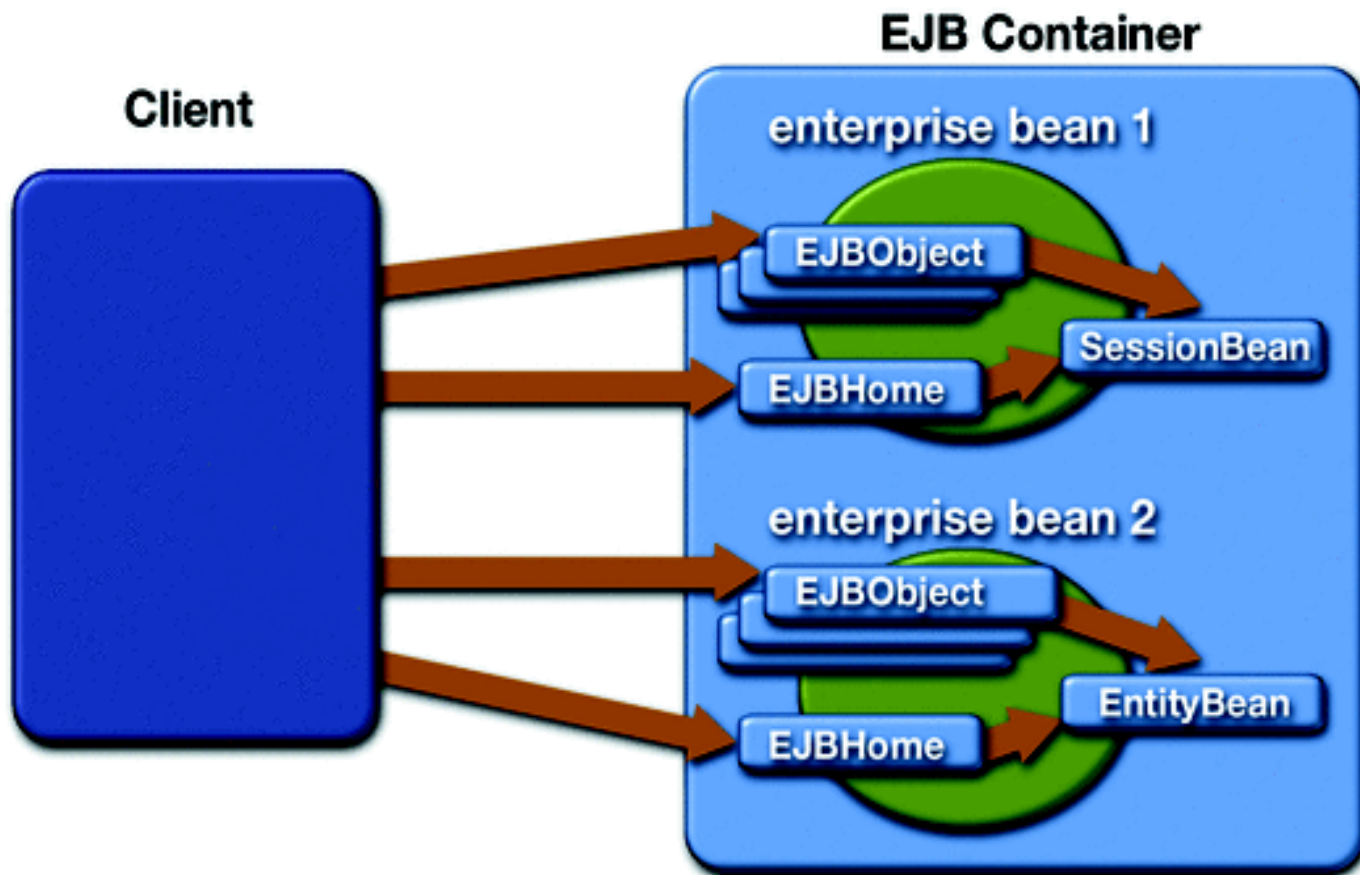
- Before a component can be executed, it must be assembled into a J2EE application and deployed into its container



EJB Container

- A container is provided by the *Application Server* vendor to provide basic services that are required by J2EE specification.
- An EJB programmer places their code here, and is assured a variety of basic services are available
- EJBs are fundamental links between presentation components (web tier) and business critical data and systems (EIS tier).

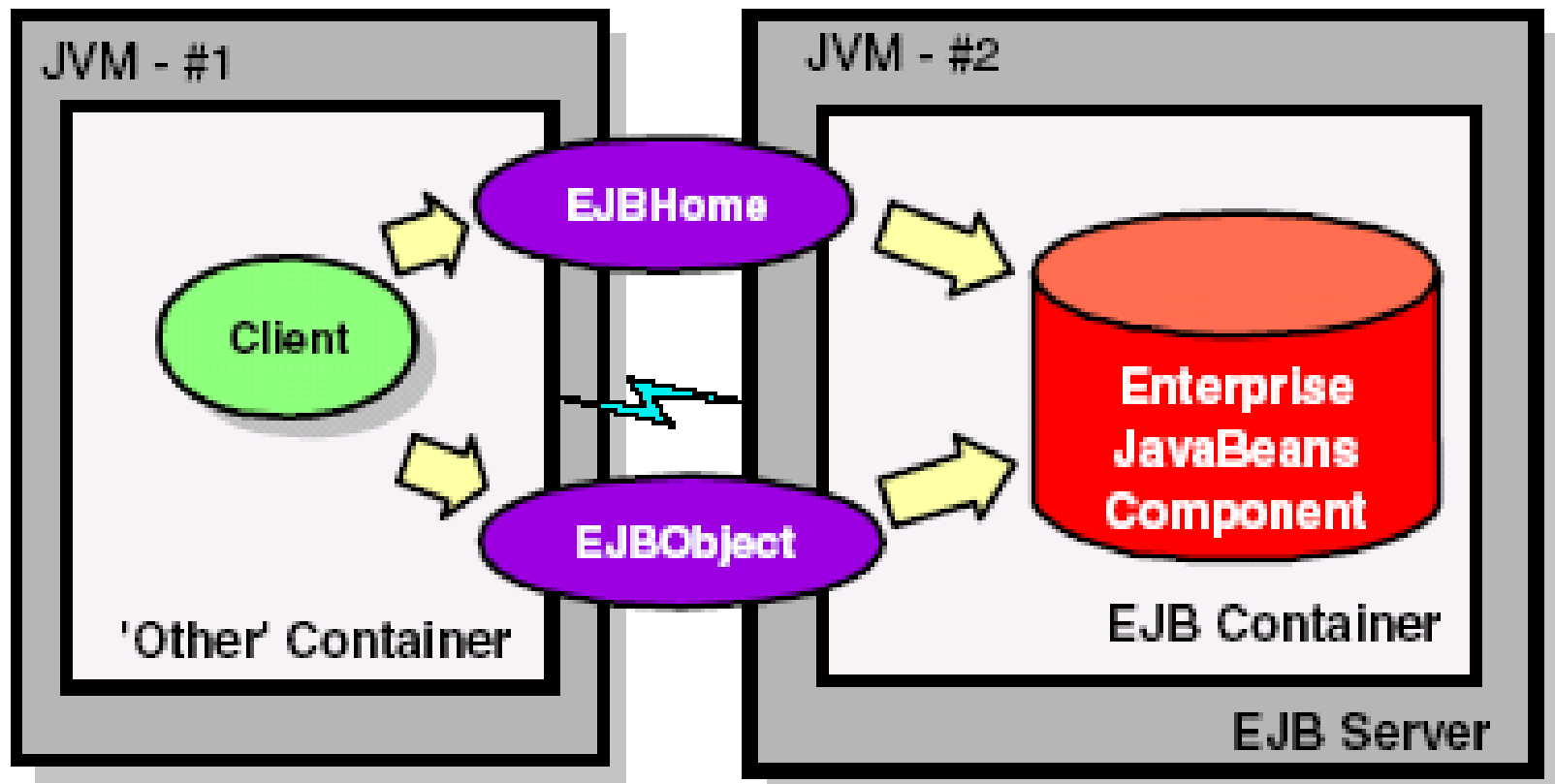
EJB Container



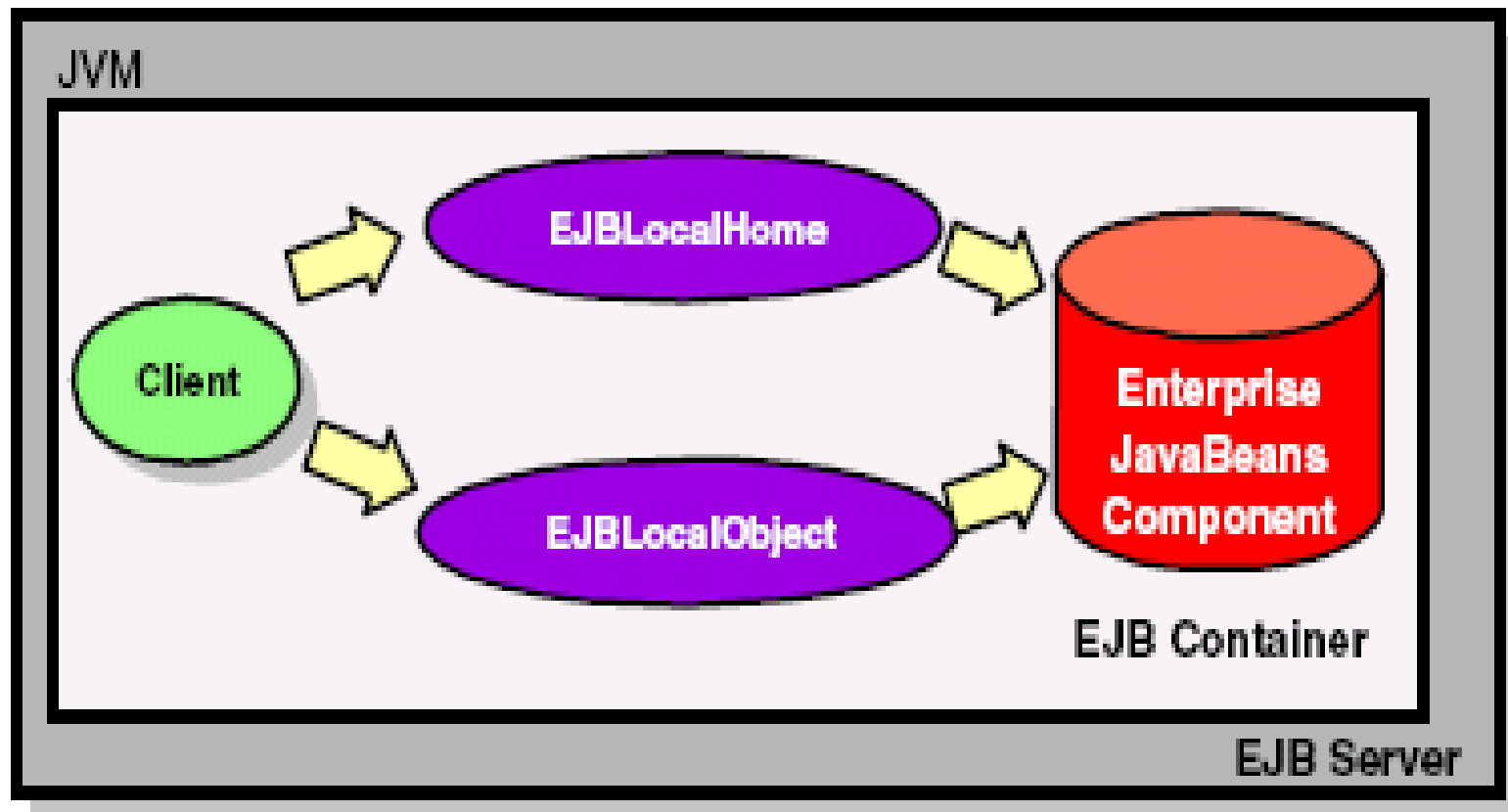
EJB Client

- From the EJB client perspective, all interactions are performed on objects that implement the home and component interfaces.

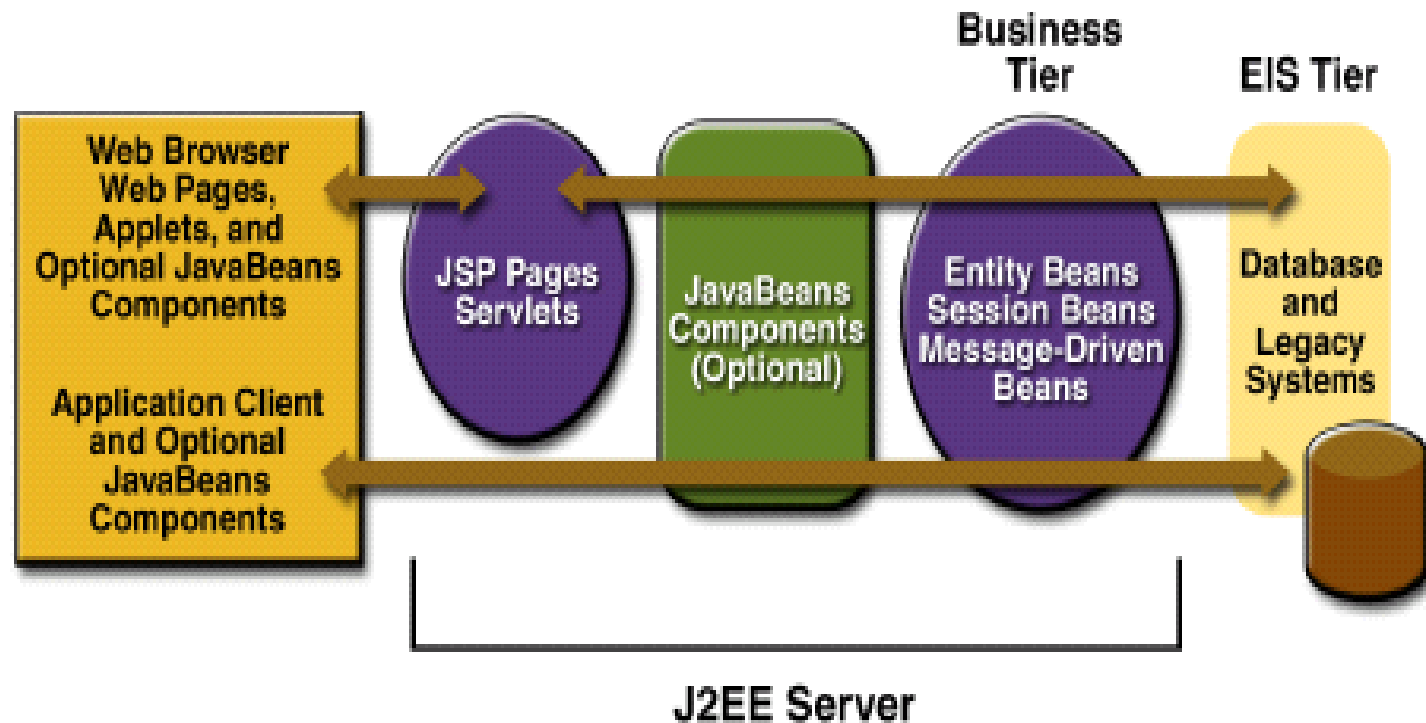
Client invocation to Remote Interfaces



Client invocation to Local Interfaces



Business Tier in EJB Container



Basic Services Provided by the EJB Container

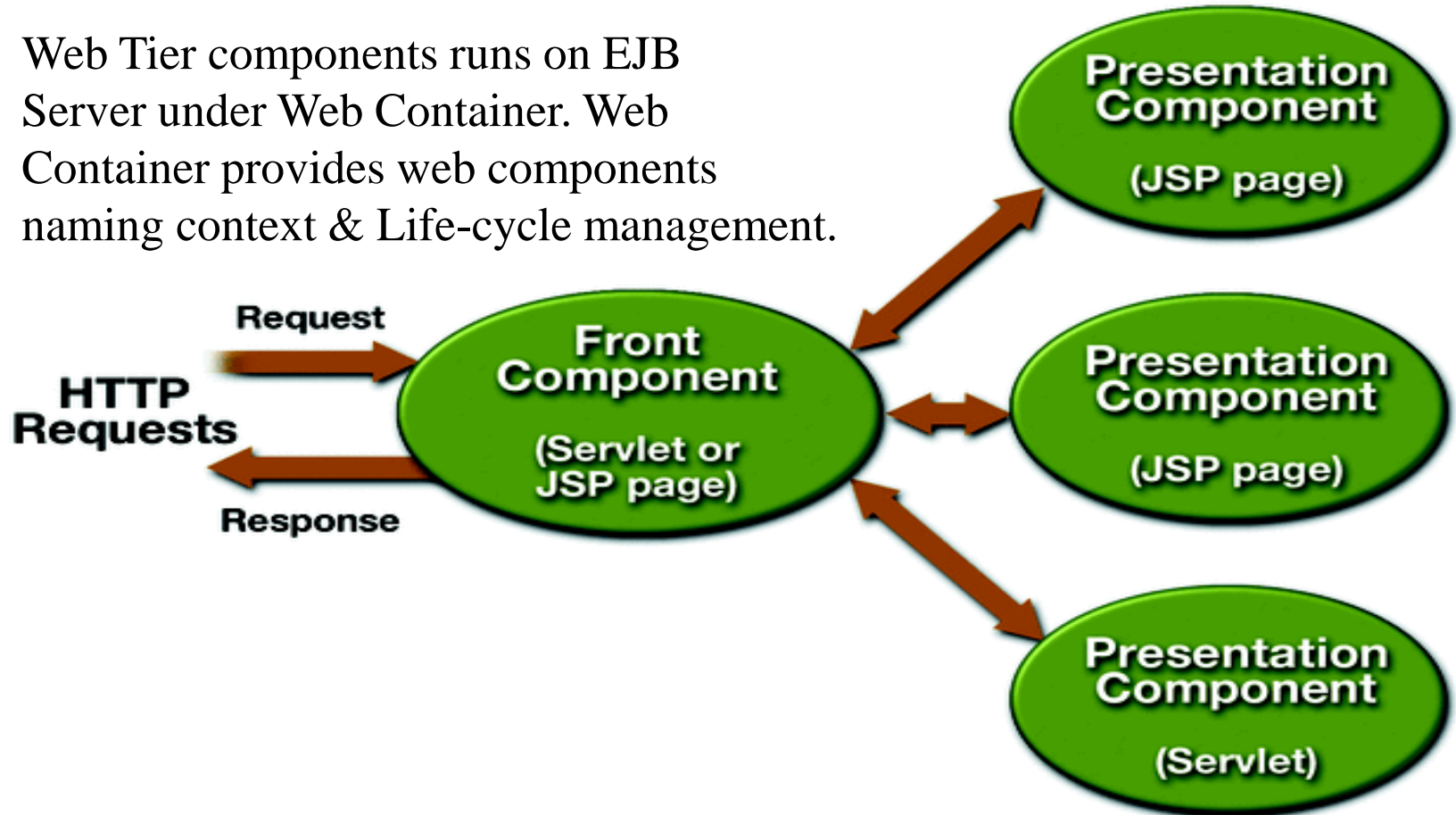
- Naming
- Transaction management
- Security
- Persistence
- Concurrency
- Life cycle management
- Messaging
- Remote client connectivity
- Database connection pooling

Web Container

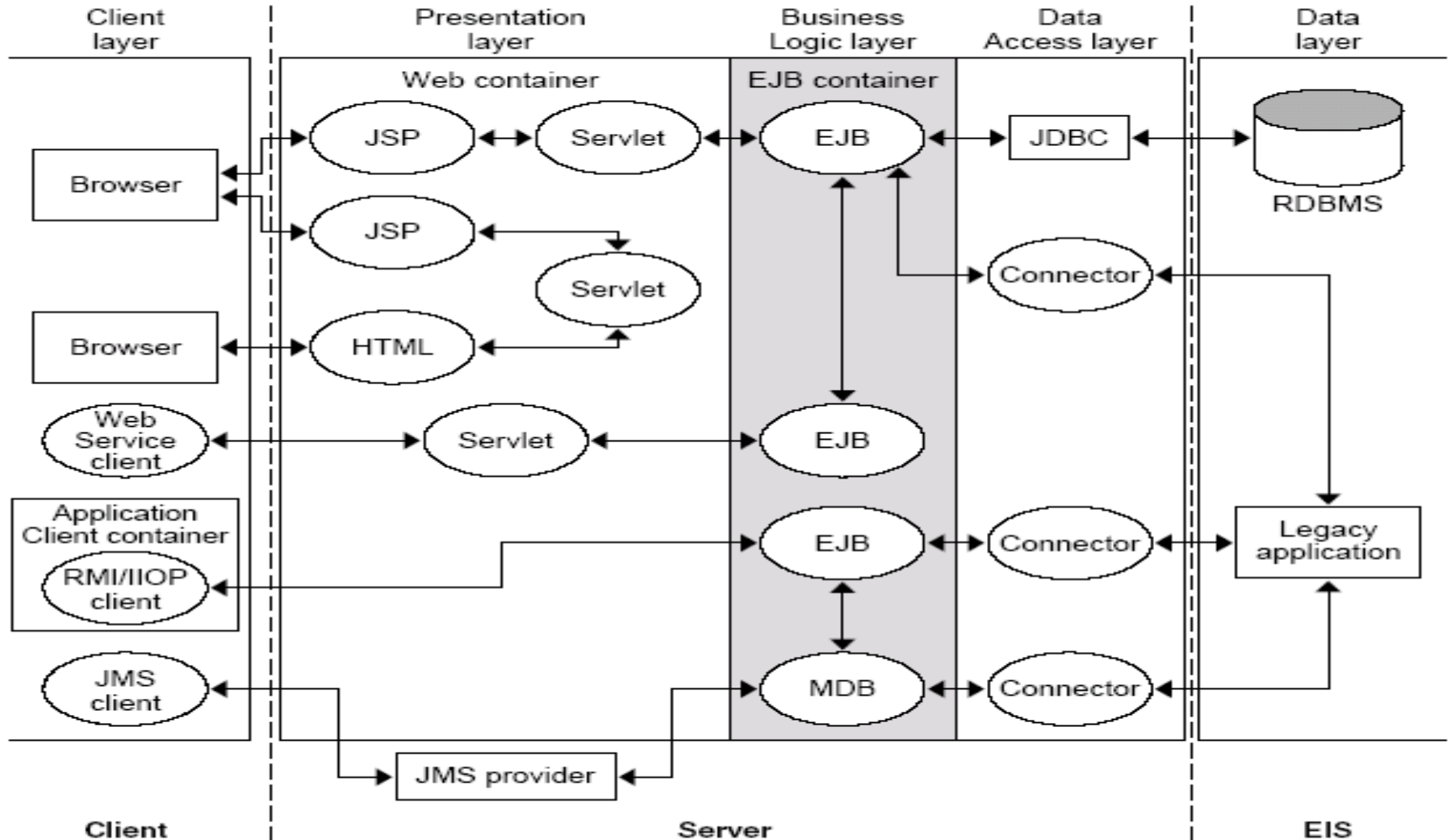
- Services supported by the web container
 - HTTP
 - JSP
 - Servlets

Web tier in web container

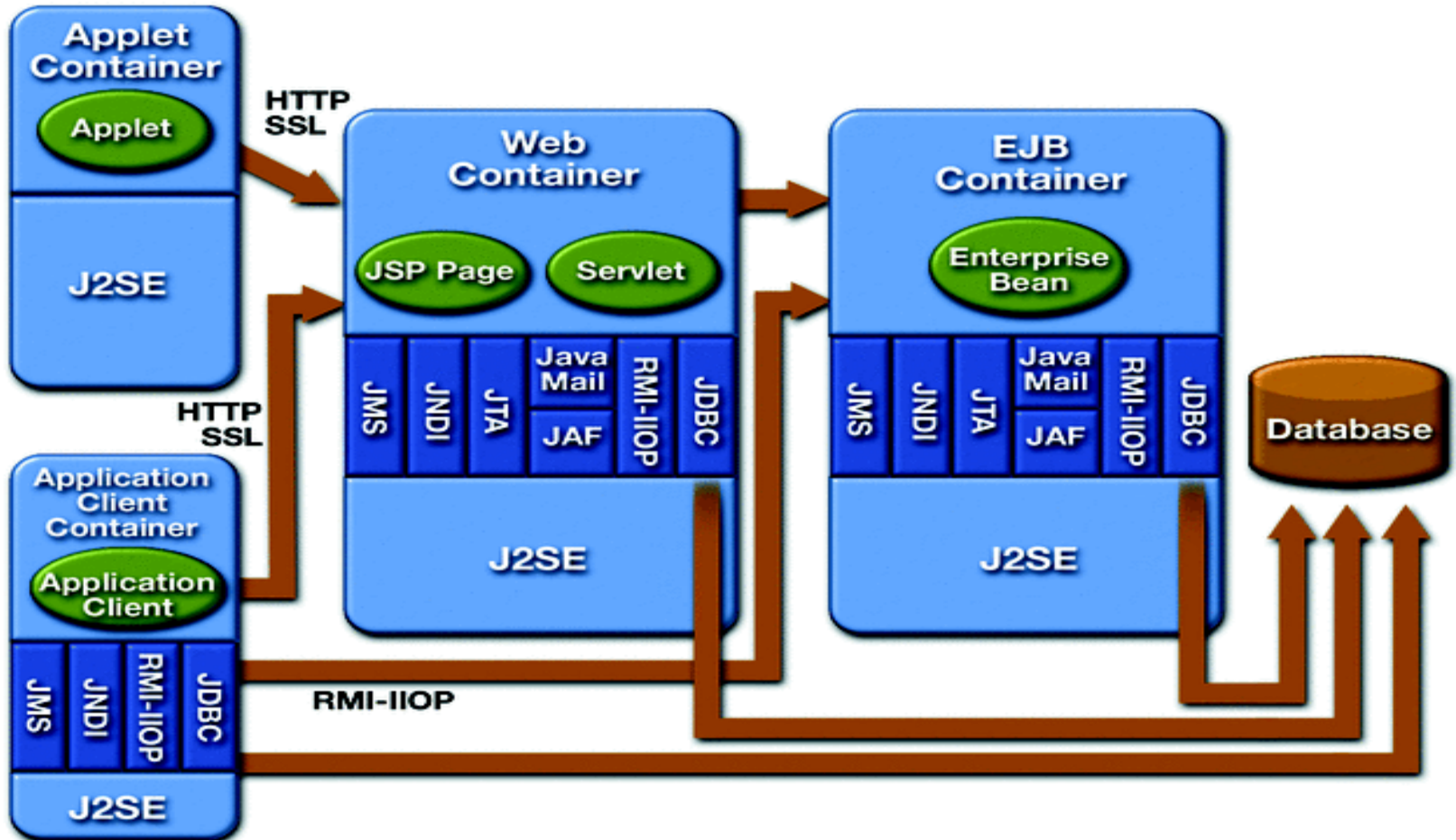
Web Tier components runs on EJB Server under Web Container. Web Container provides web components naming context & Life-cycle management.



J2EE Components in J2EE Container



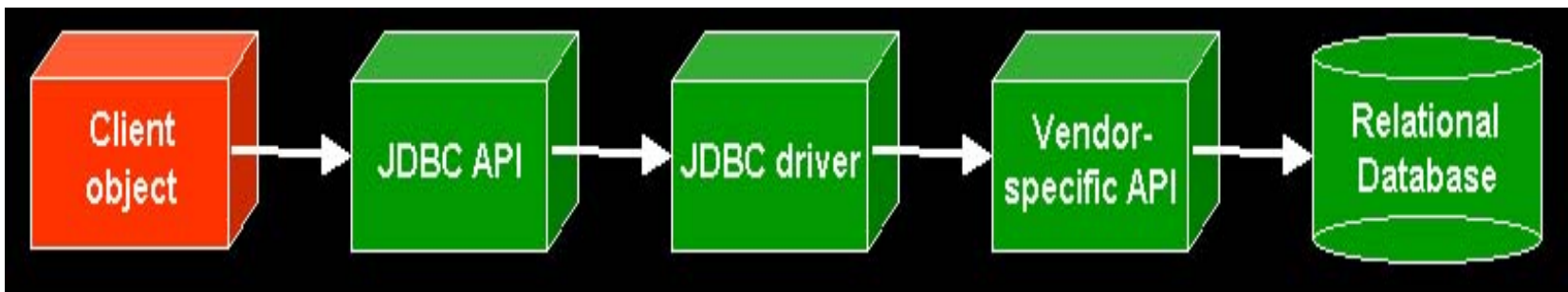
J2EE Component Technology



Component Technologies

(JDBC)

- JDBC (Java Database Connectivity)
 - Allows access of variety of databases in a uniform way.
 - Enables data manipulation from relational databases.

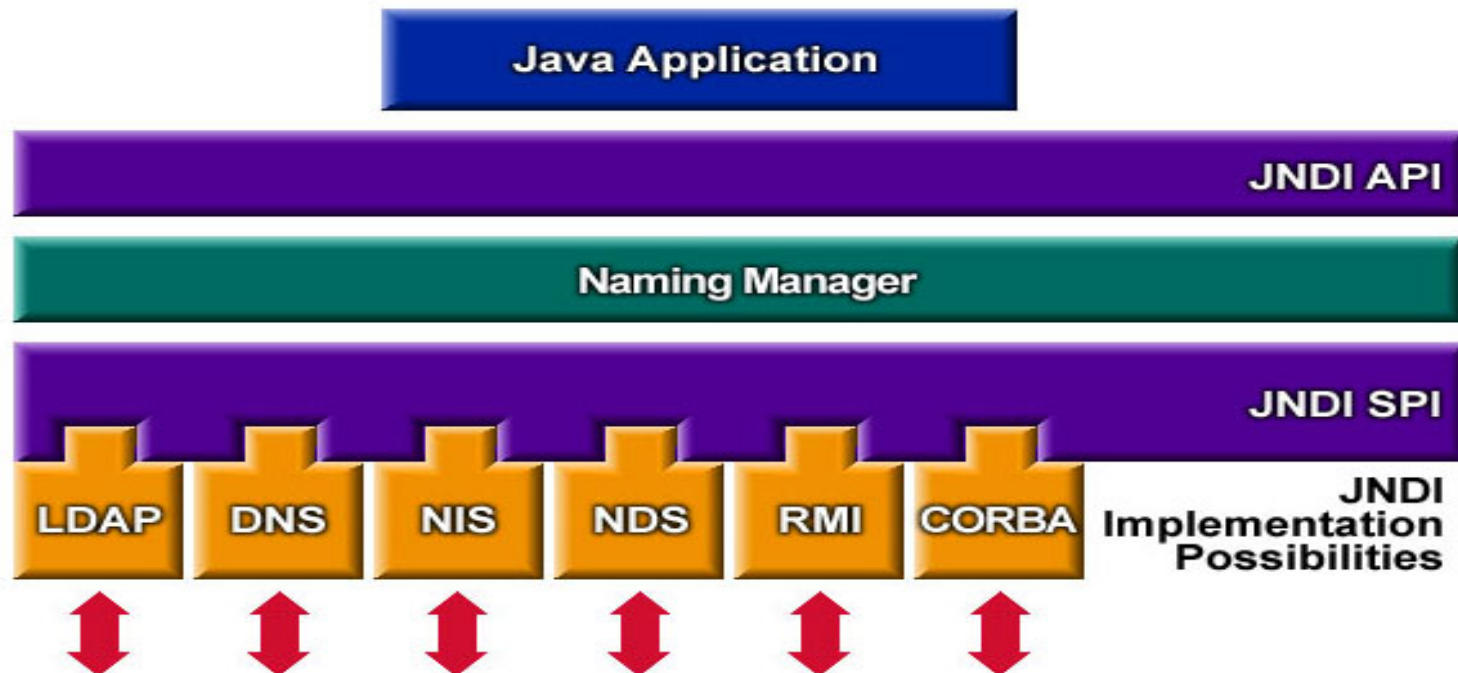


Component Technologies

(JNDI)

(Java Naming and Directory Interface)

- Standardized access to various enterprise wide naming and directory services, such as DNS, LDAP, local file systems, etc.



Component Technologies

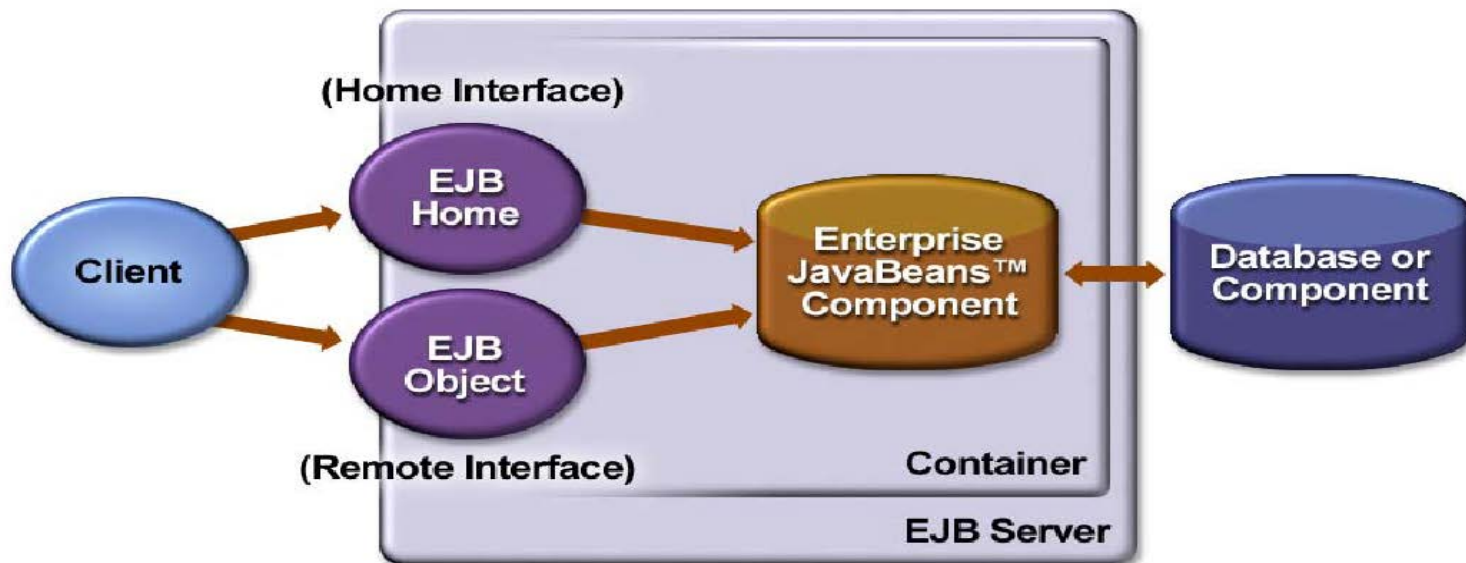
(JNDI)(Continued...)

- All objects are stored in a hierarchical way
- The way the Home interface for EJBs are located
- Used to store connection pools for JDBC
- `Context.lookup("java:comp/env/jdbc/Pool1")`
 - Private context from within components
 - Defined with a `<resource-ref>` in deployment descriptors
- `Context.lookup("Pool1")`
 - Global context from the outside

Component Technologies

(EJB)

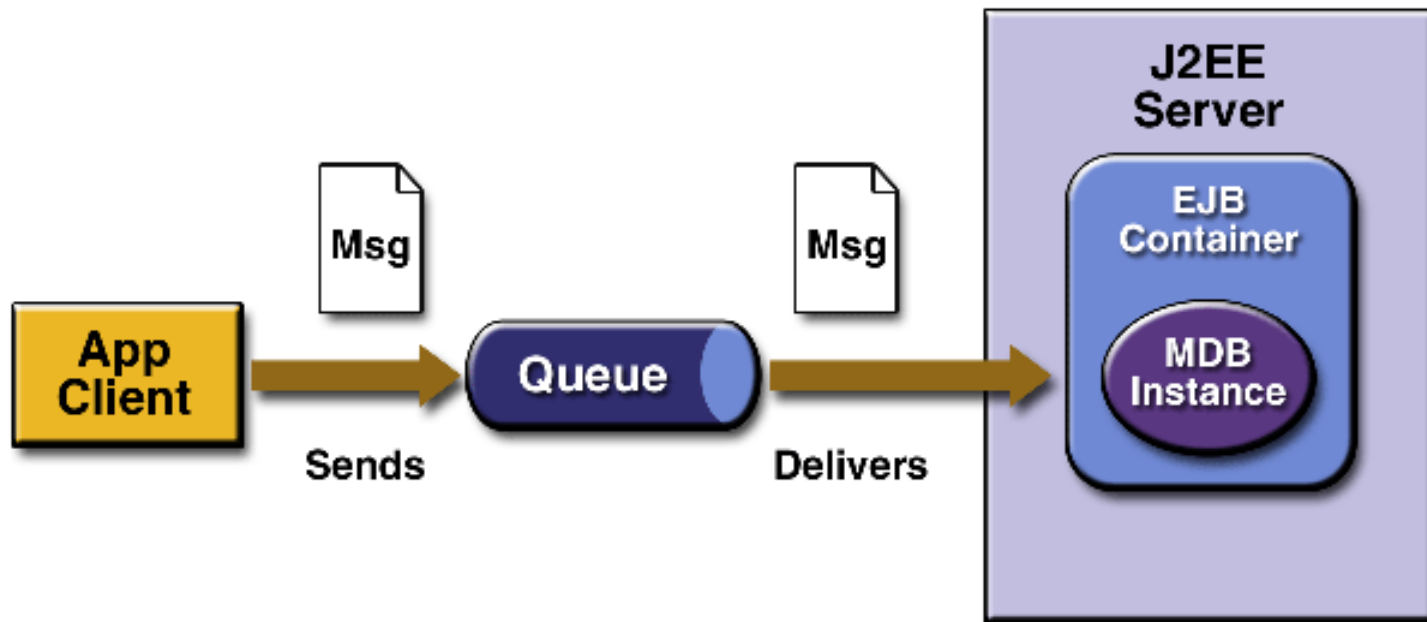
- EJB (Enterprise JavaBean)
 - The flexible business component model
 - Conceals application complexity and enables the component developer to focus on business logic.
 - Types: Session, Entity, and Message-Driven.



Component Technologies

(JMS)

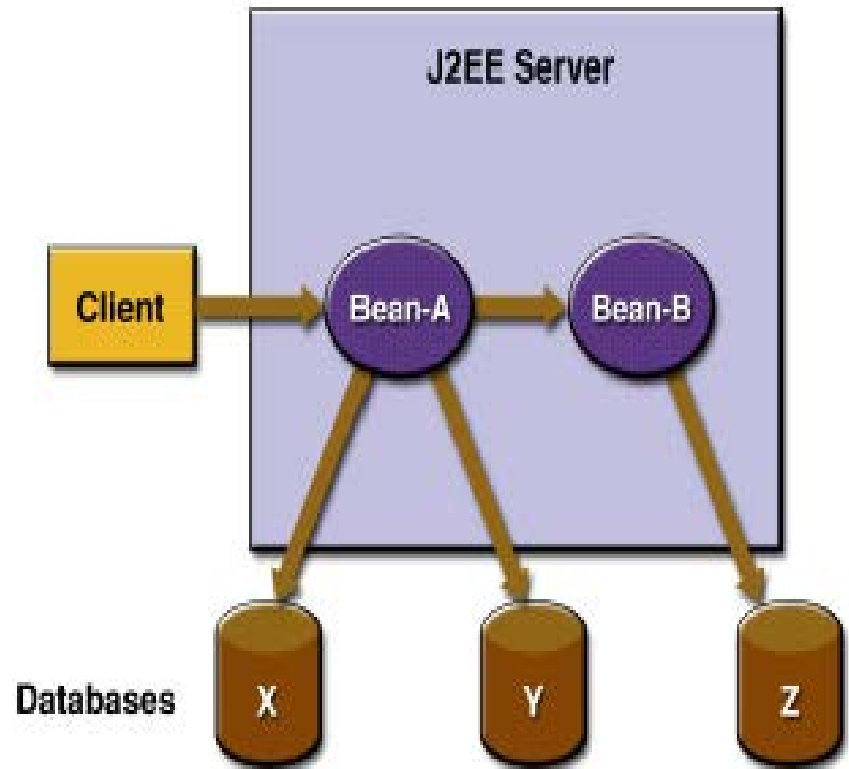
- JMS (Java Messaging Service)
 - a messaging standard that allows J2EE app components to create, send, receive, and read messages.
 - Enables distributed communication that is loosely coupled, reliable, and asynchronous.



Component Technologies

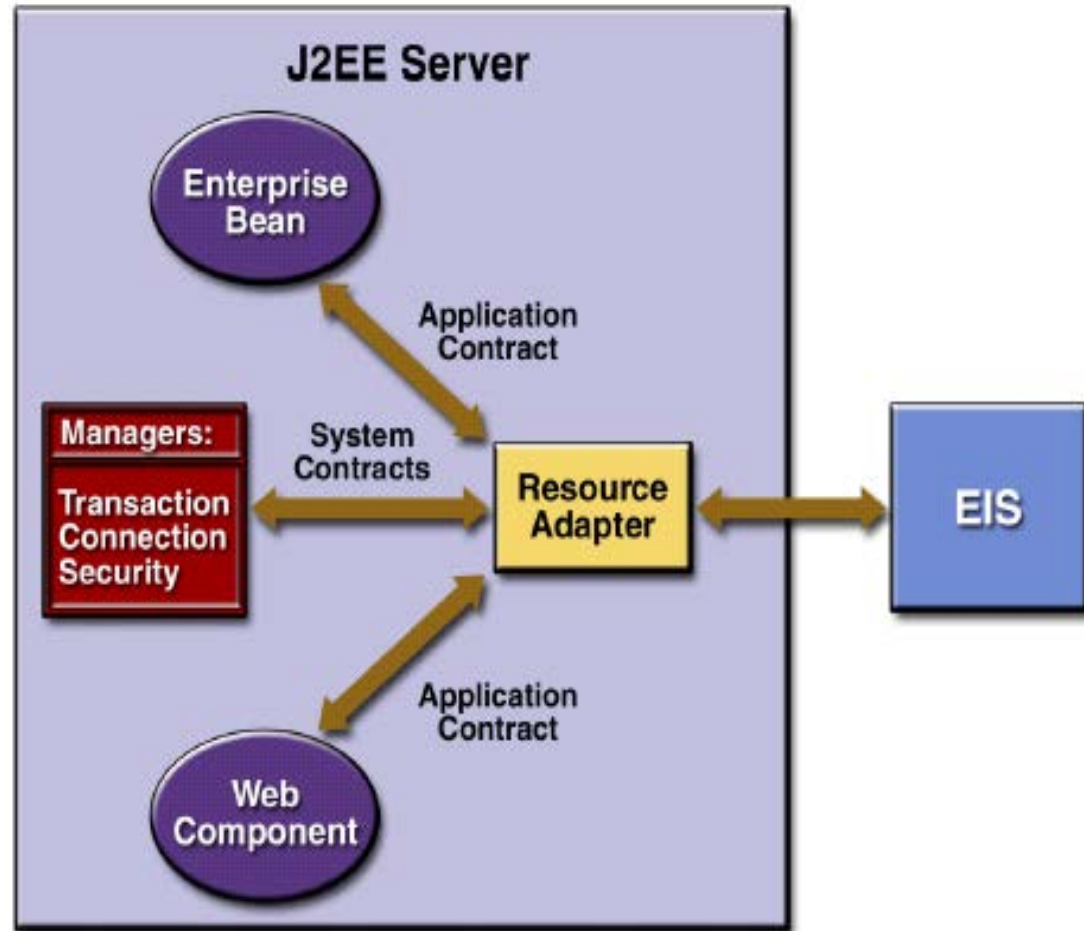
(JTA)

- JTA (Java Transaction API)
 - Provides a standard interface for demarcating transactions. e.g., a transaction requiring two separate database access operation that depend on each other.
 - Provides a way for J2EE components and clients to manage their own transactions, and for multiple components to participate in a single transaction.



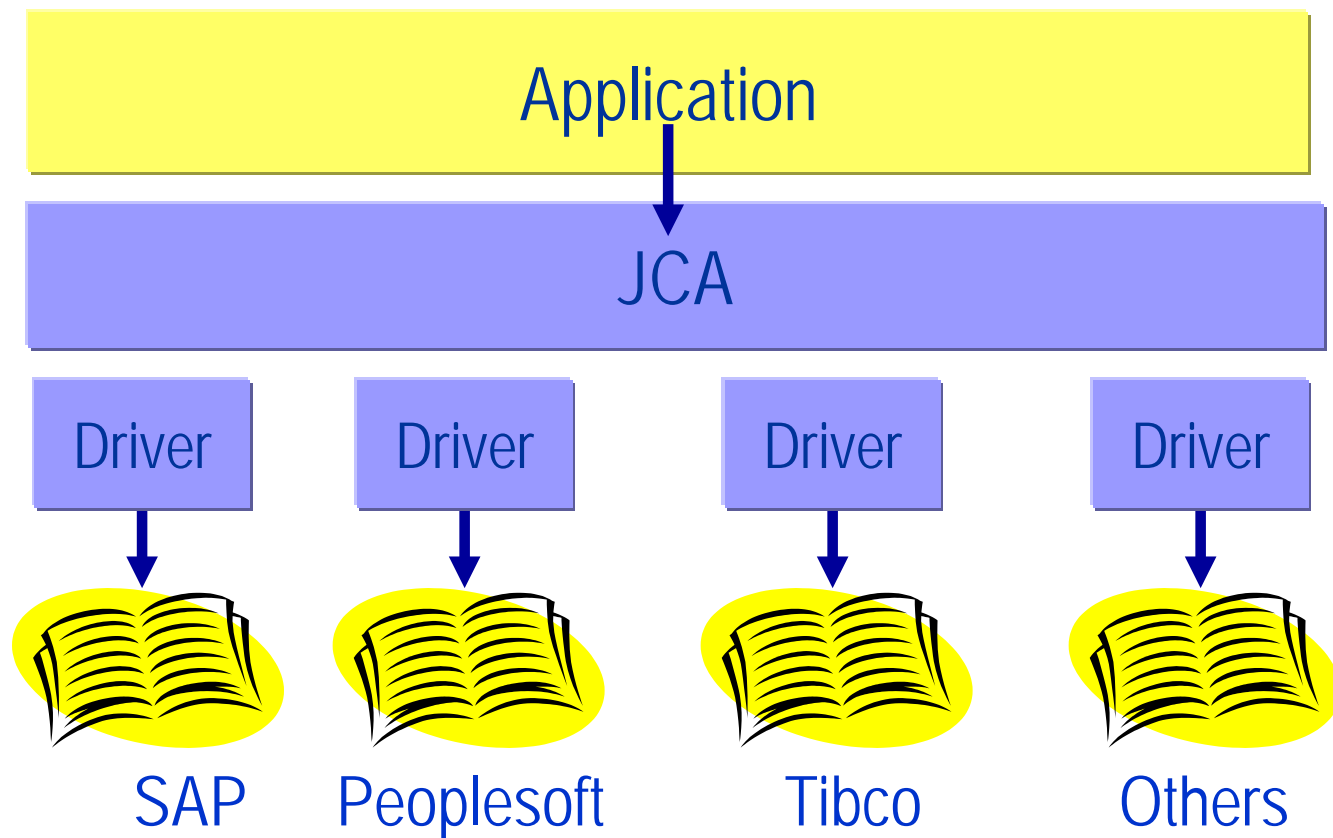
Component Technologies (JCA)

- J2EE Connector Architecture enables J2EE components to interact with EIS like-ERP, Mainframe Transaction Processing and non-relational Databases



Component Technologies

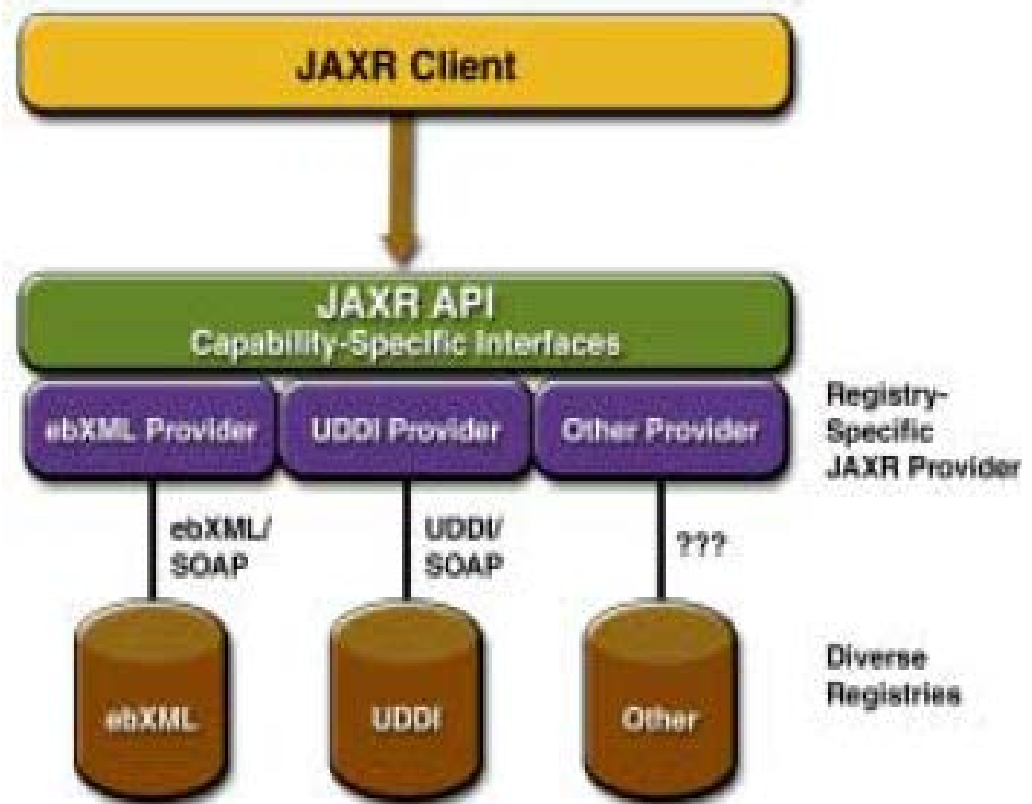
(JCA)(Continued..)



Component Technologies

(JAXR)

- JAXR enables Java software programmers to use a single, easy-to-use abstraction API to access a variety of XML registries.



Component Technologies

(JAX-RPC)

- Specify APIs for supporting XML based RPC for the Java platform
- The JAX-RPC specification describes the support for message handlers that can process message requests and responses.
- In XML based RPC, a remote procedure call is represented using an XML based protocol. SOAP 1.1 specification defines an XML based protocol for exchange of information in a decentralized, distributed environment.

Component Technologies

(JAAS)

JAAS (Java Authentication and Authorization Service)

- A Java version Pluggable Authentication Module (PAM), extending the J2EE security architecture to support user-based authorization.
- JAAS can be used for two purposes:
 - for *authentication* of users, to reliably and securely determine who is currently executing Java code
 - for *authorization* of users to ensure they have the access control rights (permissions) required to do the actions performed.

Component Technologies

(RMI-IIOP)

- RMI-IIOP
(Remote Method Invocation - Internet Inter-ORB Protocol)
 - combines the programming ease of Java RMI with CORBA's IIOP for easier integration of J2EE applications with legacy applications

Component Technologies

(XML)

- XML (eXtensive Markup Language)
 - Is a cross-platform, extensible, and text-based standard for representing data.
 - Parties are free to create their own tags to describe the data, set up schemas to specify which tags can be used in a particular kind of XML document, and use XML style sheets to manage the display and handling of the data.
- Java API for XML Processing (JAXP)
- Simple API for XML Parsing (SAX)
- Document Object Model (DOM)
- What does it get me?
 - You decide the parser that best fits your organization.

Component Technologies

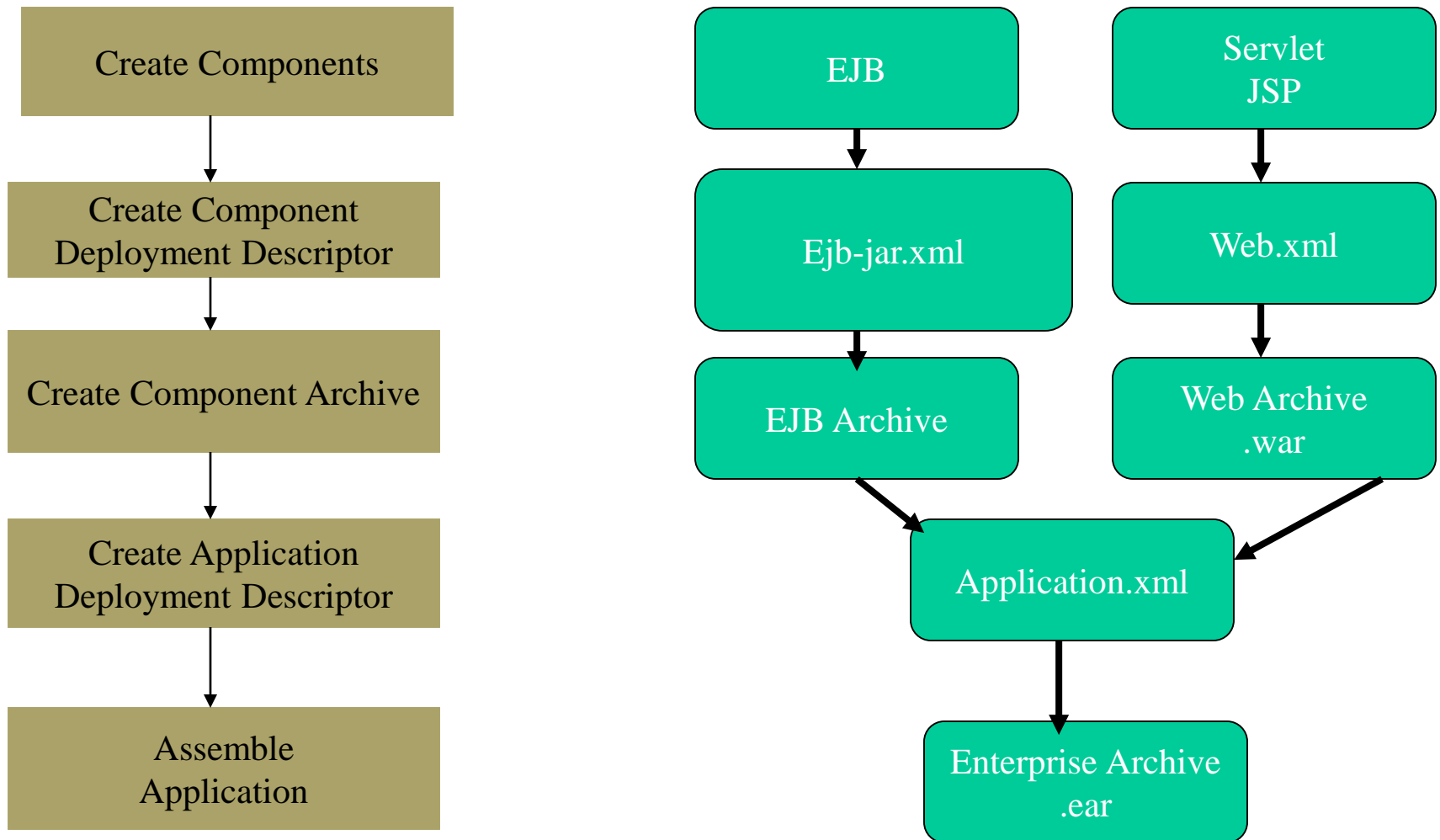
(JavaMail, JAF)

- JavaMail
 - can be used to send internet mails.
 - Supports provision of new protocols & functionality
 - Can be used to implement IMAP4, POP3, and SMTP.
- JAF (JavaBeans Activation Framework)
 - helps determine data types, encapsulate access, discover operation
 - Constructs to support MIME (Multi-purpose Internet Mail Extensions) data types, as used by the JavaMail API.

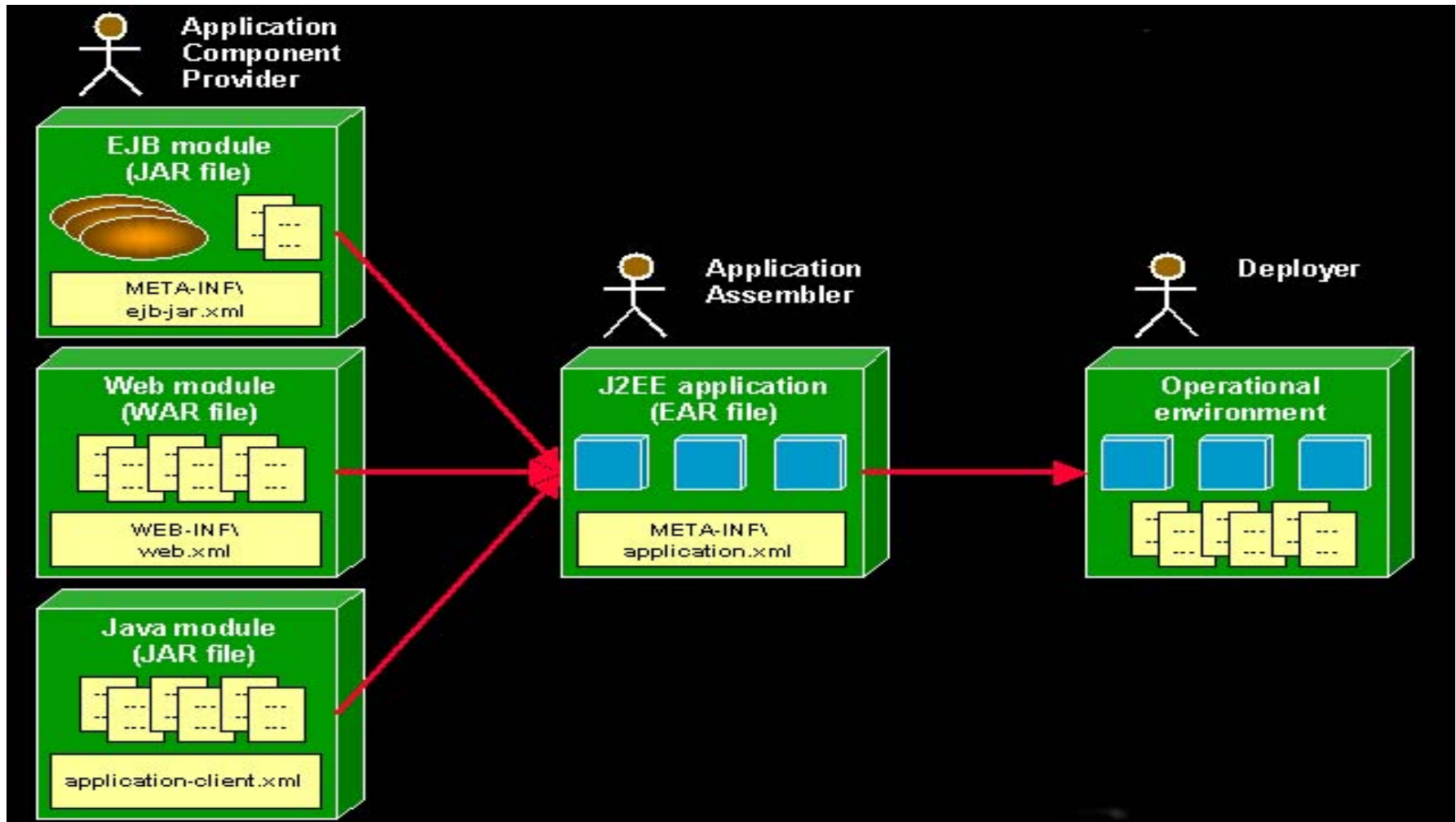
J2EE Components packed into Modules

- Enterprise Beans (jar file, ear file)
 - Classes
 - Deployment descriptor
- Web Components (war file)
 - Servlet classes, JSPs, HTML, images
 - Web application deployment descriptor
- Application Clients
 - Classes
 - Application client deployment descriptor

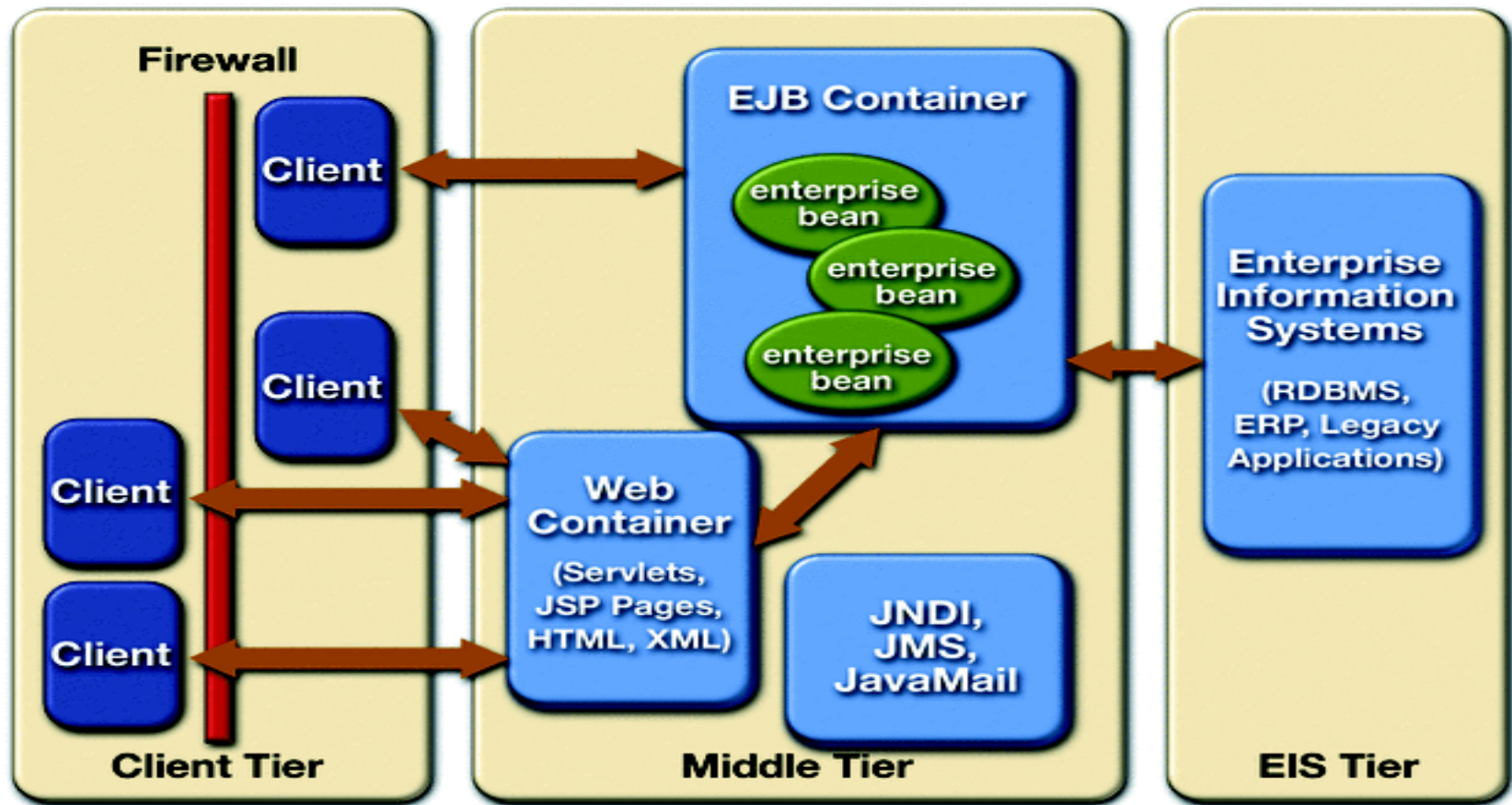
J2EE Application Creation Steps



J2EE Application Creation Steps (Cont.)



J2EE Application Scenario



References

1. “Web Services: Concepts, Architectures and Applications”, Gustavo Alonso, Fabio Casati, Harumi Kuno, Vijay Machiraju, Springer Verlag
2. "Distributed Computing: Principles and Applications" by M L Liu, Pearson Education
3. <http://www.sun.com>

References for RMI

- Default Policy Implementation and Policy File Syntax (JDK 1.2)

Sun Site: <http://java.sun.com/products/jdk/1.2/docs/guide/security/PolicyFiles.html>

- Getting Started Using RMI,

Sun Site: <http://java.sun.com/products/jdk/1.2/docs/guide/rmi/getstart.doc.html>

- Java Remote Method Invocation Specification

Sun HTML Site:

<http://java.sun.com/products/jdk/1.2/docs/guide/rmi/spec/rmiTOC.doc.html>

References for RMI

- JDK 1.2 API Documentation

Sun Site: <http://java.sun.com/products/jdk/1.2/docs/api/overview-summary.html>

- Permissions in JDK1.2

Sun Site: <http://java.sun.com/products/jdk/1.2/docs/guide/security/permissions.html>

- Remote Method Invocation (RMI) Tools Documentation (rmic, rmiregistry)

Sun Site: <http://java.sun.com/products/jdk/1.2/docs/tooldocs/tools.html#rmi>