From J2EE to Java EE

Peter Tröger

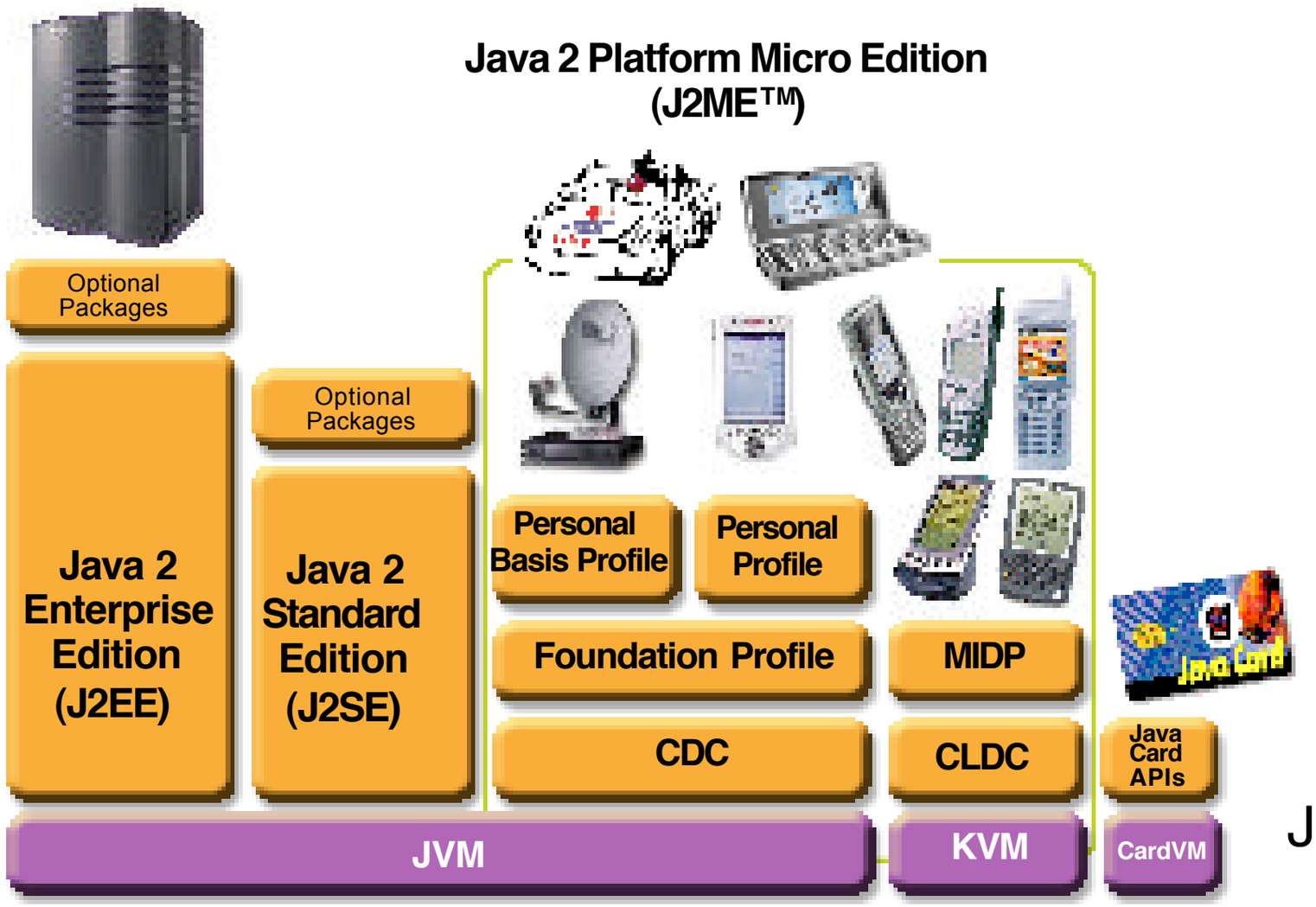
Enterprise Application Framework

- Single-Tier: Mainframe-based, dumb terminals, centralized model, monolithic application
- Two-Tier: Fat client talking to back-end database, standardized data access
- Three-Tier: Client communicates with presentation tier, which relies on the middle-tier application logic, which uses the data tier
    - Business logic and data model can be modified without changes for the client
    - RPC-based: Tight coupling of client and middle-tier
    - Object-based: business logic and data model encapsulated in objects (CORBA, RMI)
    - Current trend: Three tier model with web pages as presentation layer
- MVC is typically used in the presentation layer

Java EE Motivation

- Component-based distributed applications for the enterprise
  - Consider security, speed, transactional and reliable behavior
  - Avoid vendor-lock
  - Fast application design and development
    - “Write once, run everywhere”
- Problems to be solved
  - (Distributed) transactions, security (authentication, authorization, persistency (object-relational gap, caching), pooling (database connections, threads, objects), scalability, legacy integration

Counting Backwards - The Java (2) Platform



J2SE - Java 2 Standard Edition (Desktop)

Meanwhile Java SE

J2ME - Java 2 Micro Edition (Consumer device)

Meanwhile Java ME

J2EE - Java 2 Enterprise Edition (Server)

Meanwhile Java EE

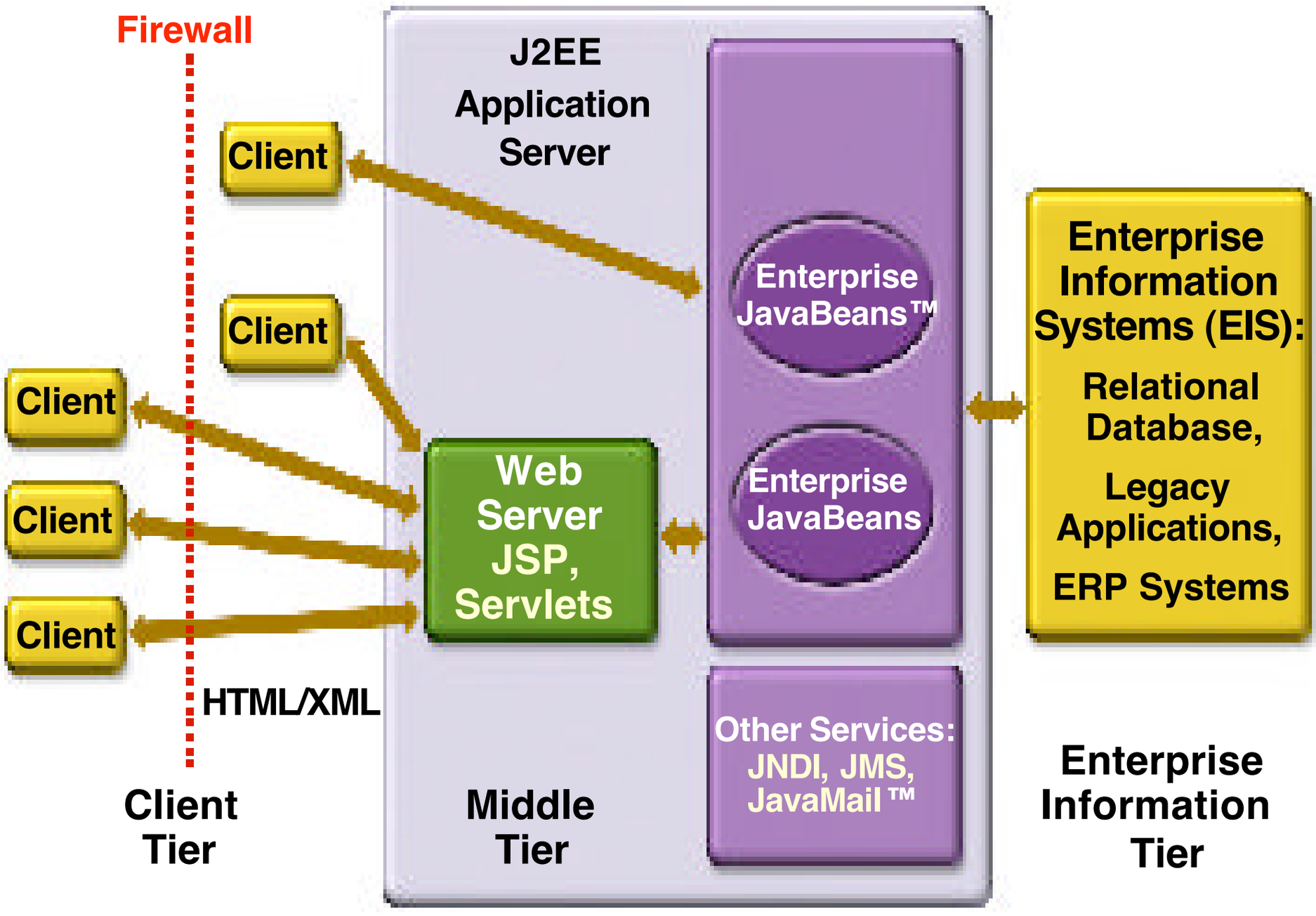
J2EE Platform

- First introduction in 1999, J2EE 1.4 approved by Java Community Process in November 2003
- API compatibility and portability for Enterprise Application Servers
- Standardized development and deployment of portable, distributed enterprise applications
- Multi-tier model for enterprise applications, standardized communication of application parts, integration of existing information systems
- Replaced by Java EE 5 in 2006
  - New EJB programming model based on annotations, better XML handling
  - More defaults, less descriptors
  - Underlying implementation of application servers remained the same

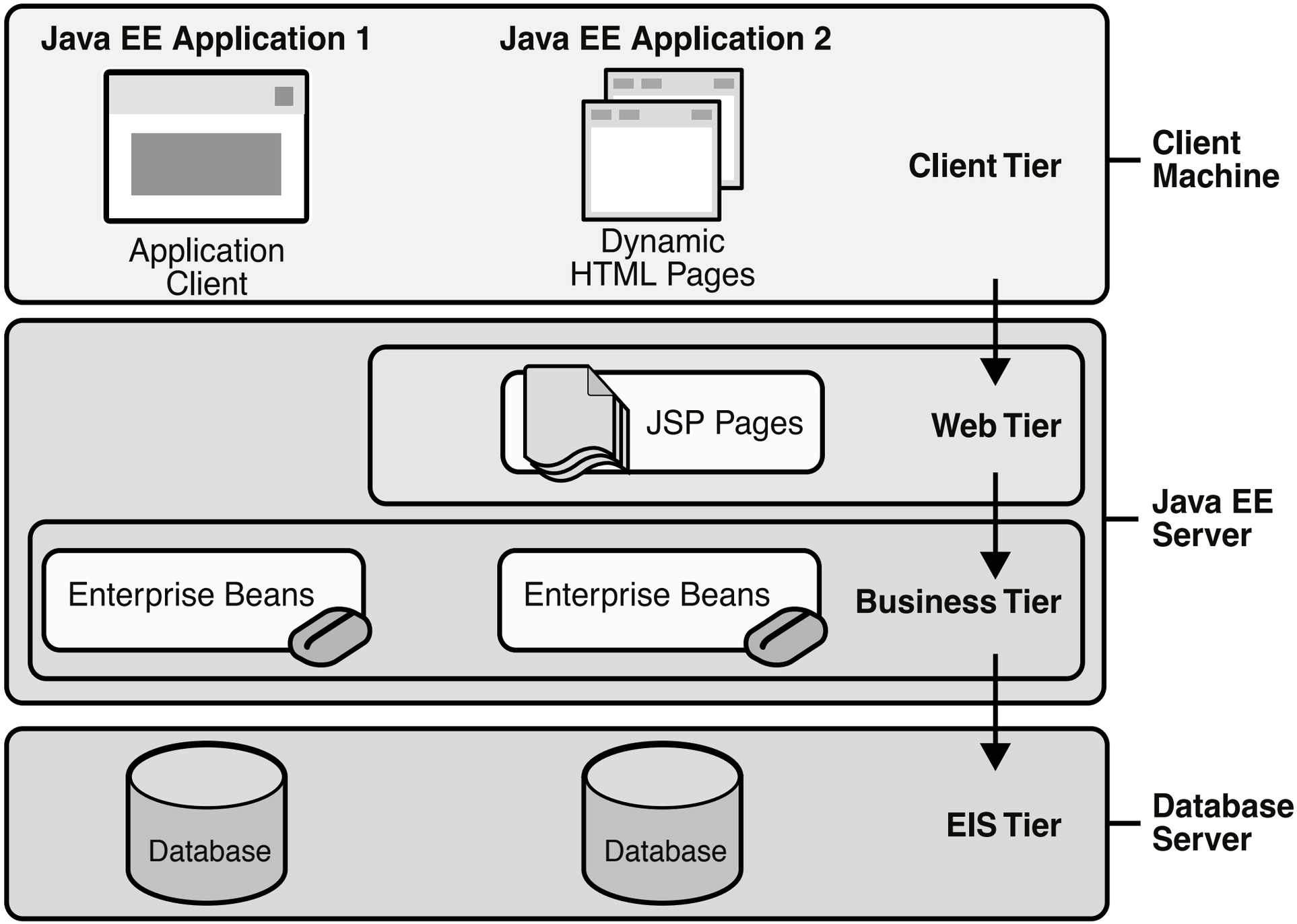
What Makes Up J2EE / JAVA EE

- API and technology specification
- Development and deployment platform
- Reference implementation as part of the SDK
    - Sun GlassFish for EE 5
- Compatibility test suite
- Brands, blueprints and best practises
    - EE 5 certified: Apache Geronimo, Bea WebLogic, Oracle Application Server, SAP NetWeaver, Sun GlassFish, ...
- Sample codes

J2EE 1.4 Three Tier Architecture



Java EE 5 Three Tier Architecture



Java EE Applications

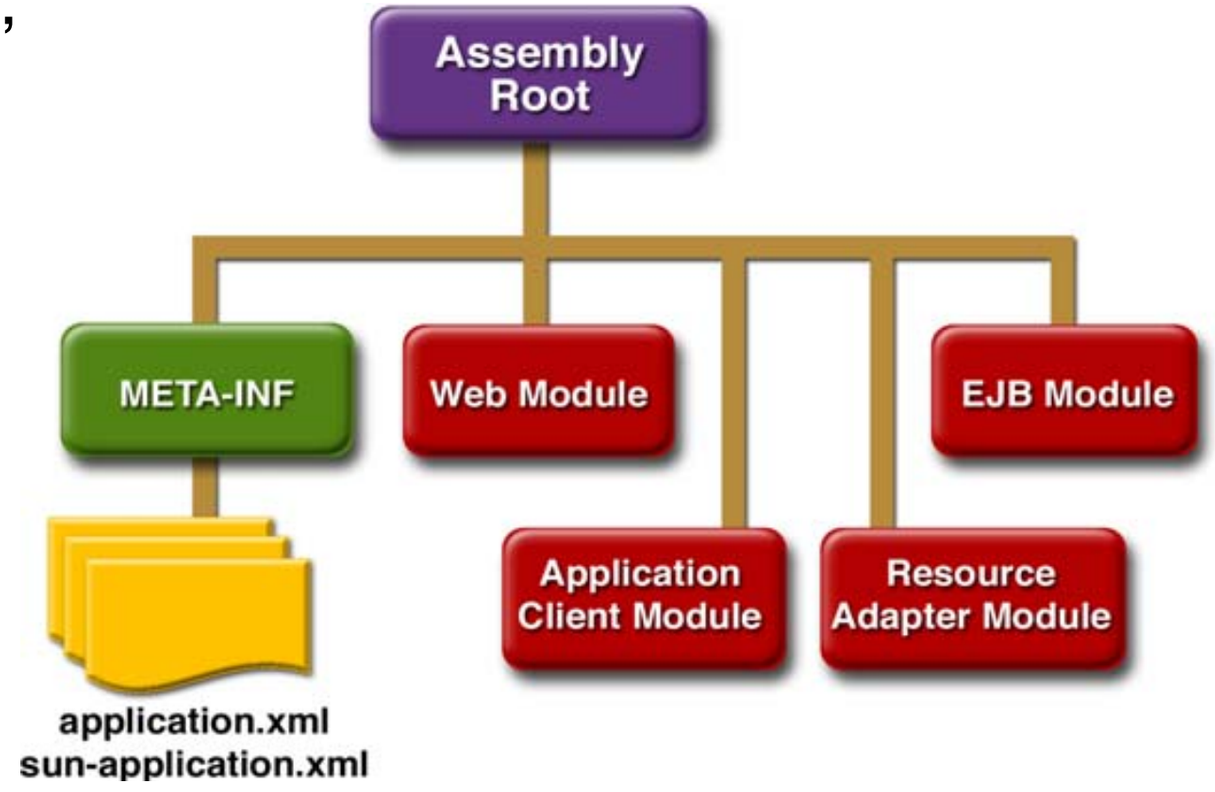
- Application logic is divided into components
- EE 5 component as self-contained functional software unit, written in Java
  - Communicates with other components
  - Executed and managed by application server
- Client-tier components, running on the client machine (applications, applets)
- Web-tier components running on the Java EE server (Java Servlet, JavaServer Faces, and JavaServer Pages)
- Business-tier components running on the Java EE server (Enterprise JavaBeans)
- Enterprise information system (EIS)-tier software runs on the EIS server

Java EE Modules

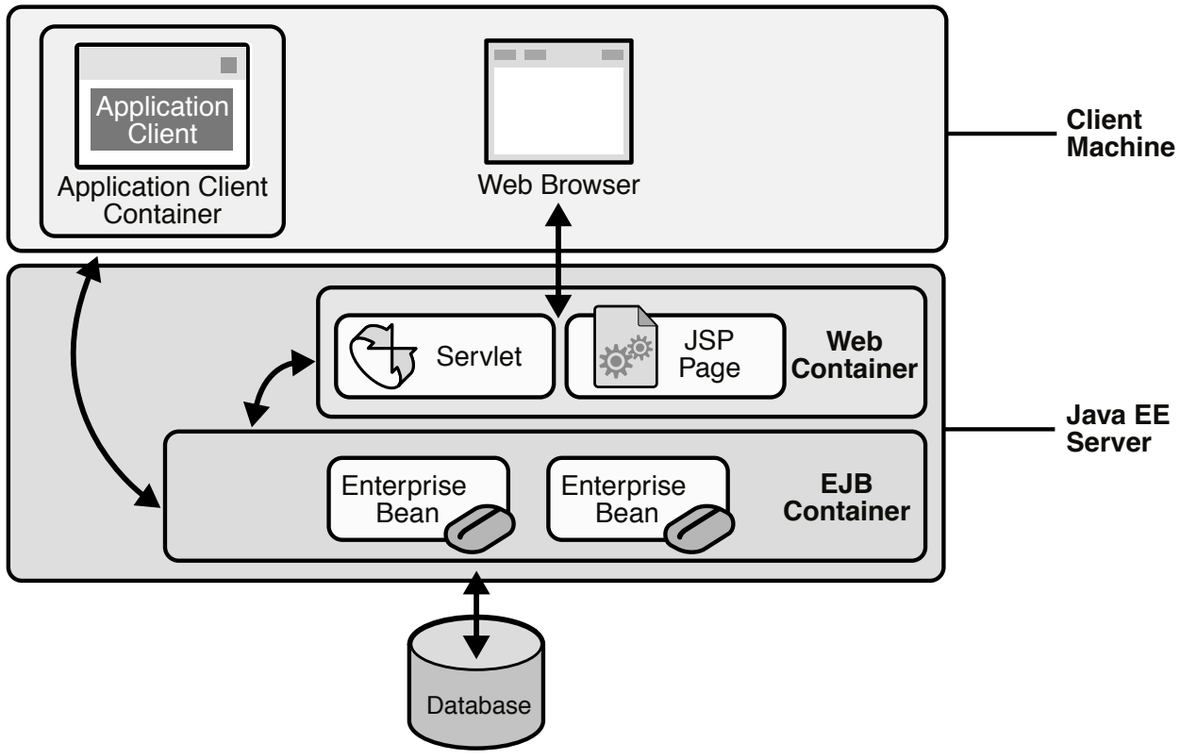
- Contains of one or more Java EE components for the same container type and one deployment descriptor
  - Transaction attributes, security authentication, ...
- Technically all JAR files, with own XML deployment descriptor
  - EJB modules (.JAR) - EJB class files
  - Web modules (.WAR) - Servlet class files, JSP files, class files, GIFs, HTML files
  - Application client modules (.JAR) - class files
  - Resource adapter modules (.RAR) - class files, native libraries, documentation; intended for JCA (EIS tier)

Packaging

- Application is delivered as Enterprise Archive File (EAR)
  - JAR file with new extension
  - Contains of J2EE modules and optional XML deployment descriptors
- Java EE descriptor is standardized, runtime deployment descriptor is not
  - Caching directives
  - Web application context root
  - ...



Java EE Containers



- Management of components in containers
  - Standardized runtime environment
  - Interpose on all method calls
  - Java EE server: runtime portion of a Java EE framework, provides containers
  - Provide specific container services for each component type, which can be expected by the component (like state management or pooling)
  - Enables platform-independent deployment
  - Container settings as part of the assembled application or preconfigured

Responsibilities

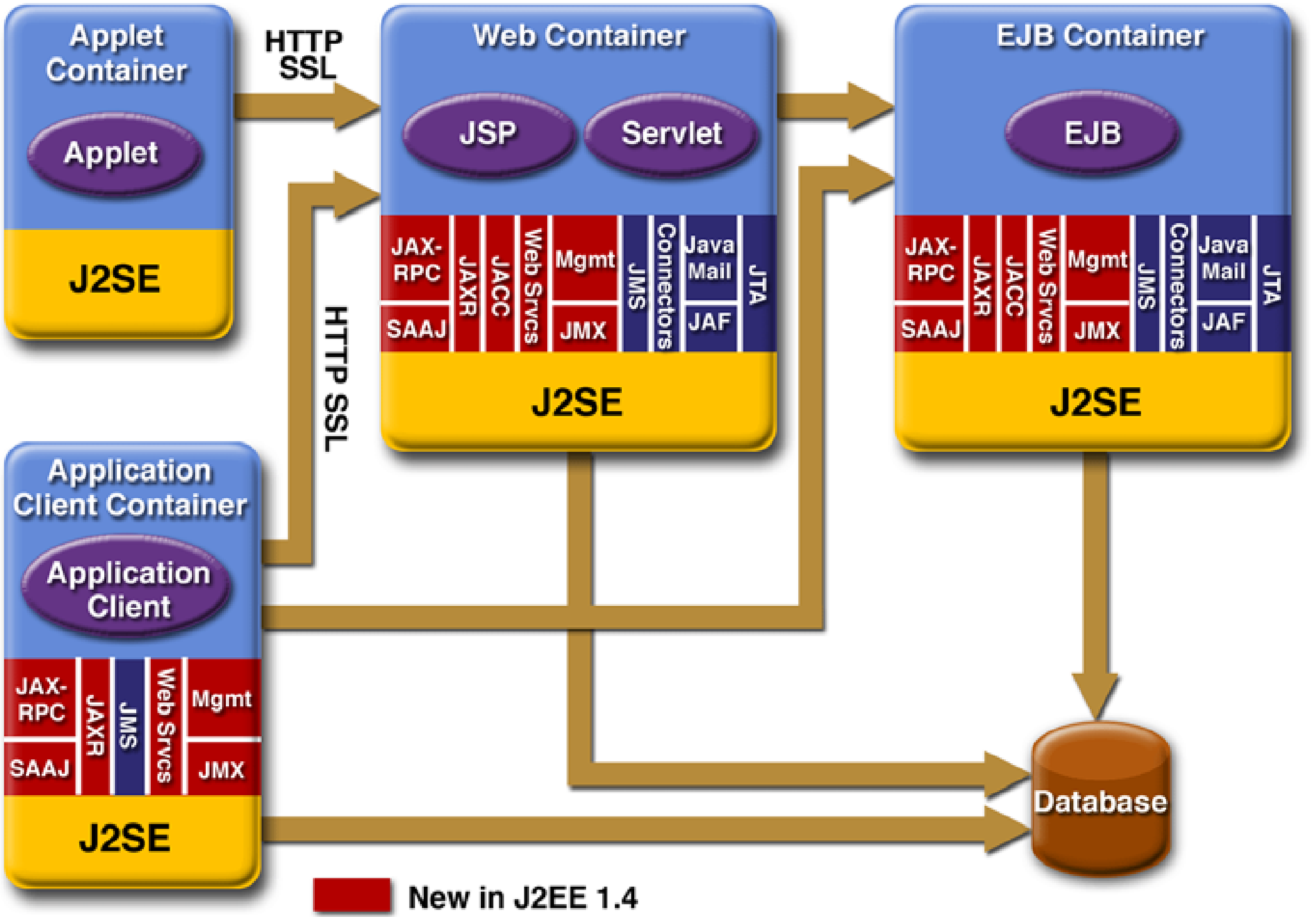
Containers Handle

- Concurrency
- Security
- Availability
- Scalability
- Persistence
- Transaction
- Life-cycle management
- Management

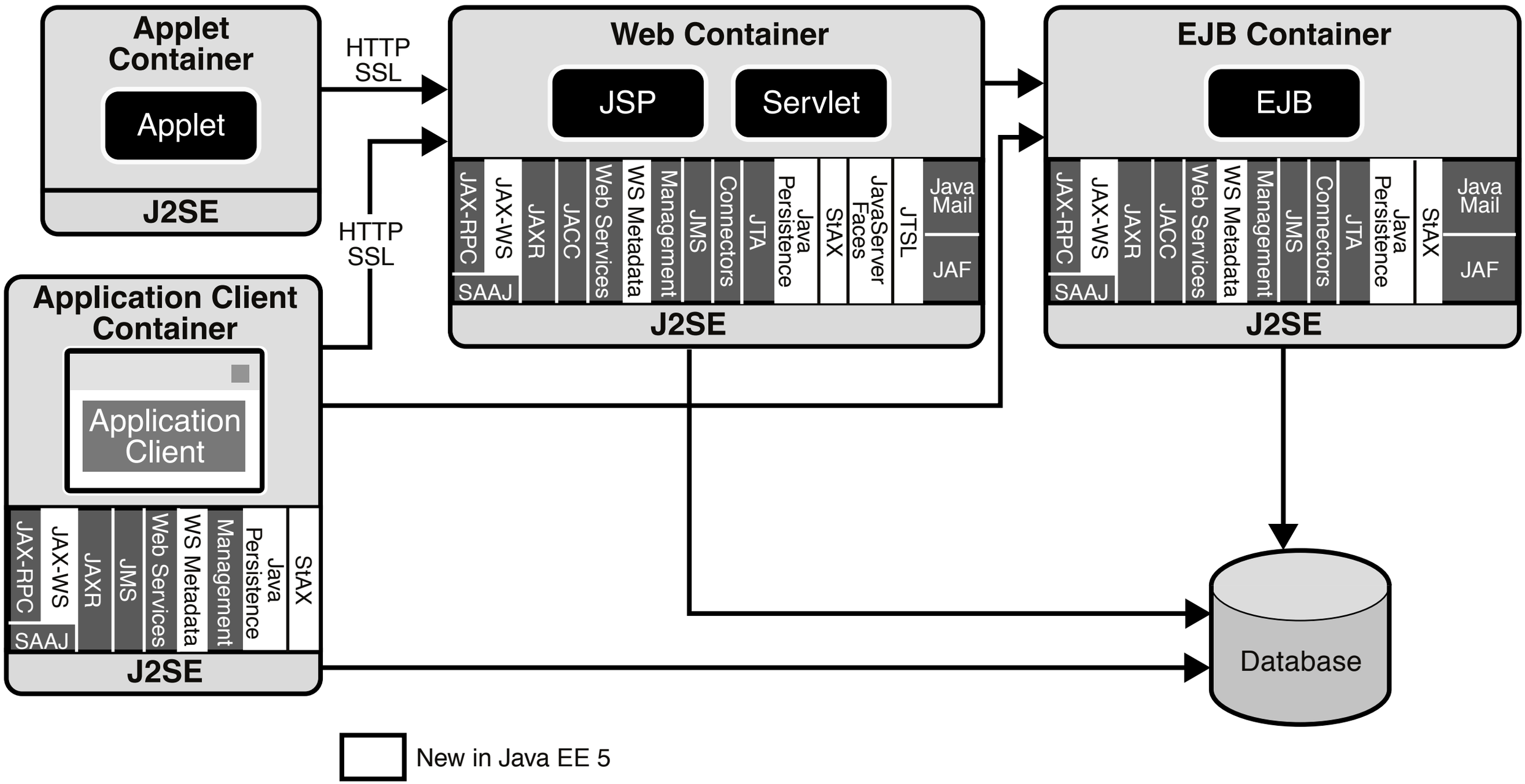
Components Handle

- Presentation
- Business Logic

J2EE 1.4 APIs


Applet Container
Applet
J2SE
HTTP
SSL
Web Container
JSP
Servlet
JAX-RPC
SAAJ
JAXR
JACC
Web Srvcs
Mgmt
JMX
JMS
Connectors
Java Mail
JAF
JTA
J2SE
EJB Container
EJB
JAX-RPC
SAAJ
JAXR
JACC
Web Srvcs
Mgmt
JMX
JMS
Connectors
Java Mail
JAF
JTA
J2SE
HTTP SSL
Application Client Container
Application Client
JAX-RPC
SAAJ
JAXR
JMS
Web Srvcs
Mgmt
JMX
J2SE
Database
New in J2EE 1.4

Java EE 5 APIs

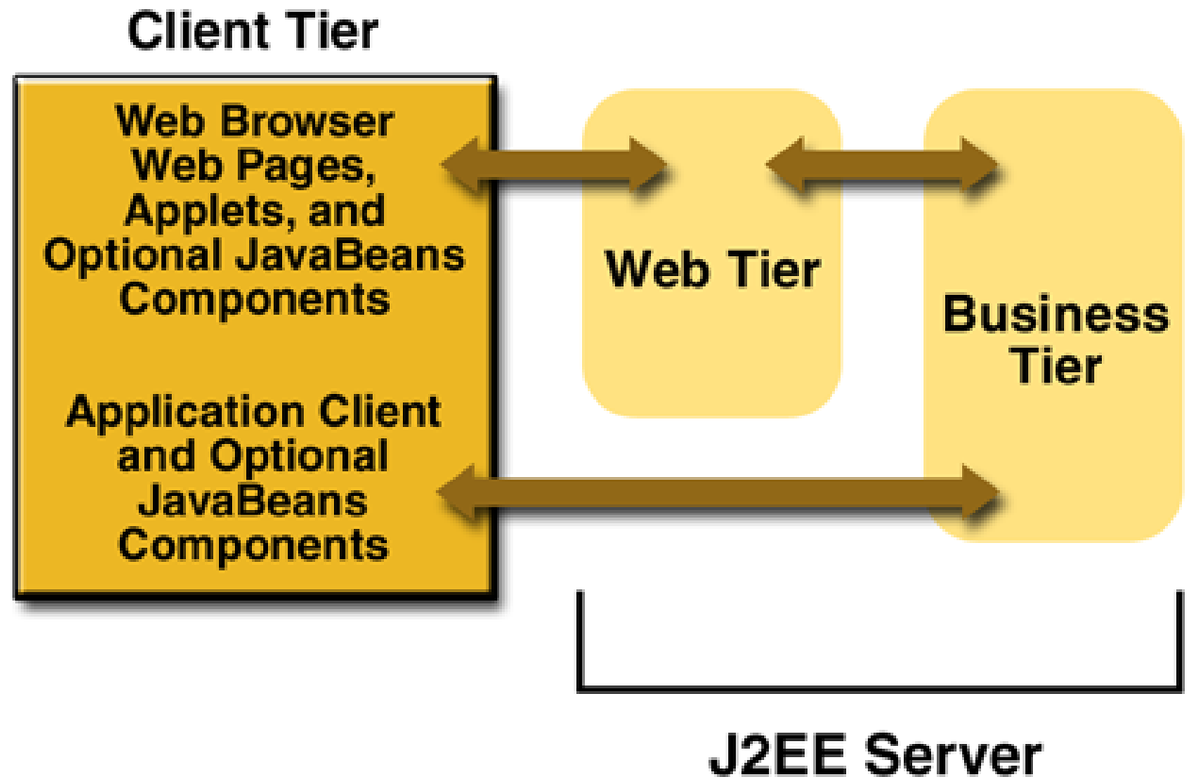


EE 5 APIs

- Enterprise JavaBeans Technology (EJB)
- Java Servlet Technology / JavaServer Pages Technology (JSP) / JavaServer Pages Standard Tag Library (**JSTL**) / **JavaServer Faces**
- Java Message Service API (JMS)
- Java Transaction API (JTA)
- Java Mail API & SPI / JavaBeans Activation Framework (JAF)
- Java API for XML Processing (JAXP)
- Java API for XML Web Services (**JAX-WS**) / SOAP with Attachments (SAAJ)
- Java API for XML Registries (JAXR)
- Java Architecture for XML Binding (JAXB)
- J2EE Connector Architecture (JCA)
- Java DataBase Connection (JDBC) API and SPI
- **Java Persistence API**
- Java Naming and Directory Interface (JNDI)
- Java Authentication and Authorization Service (JASS)

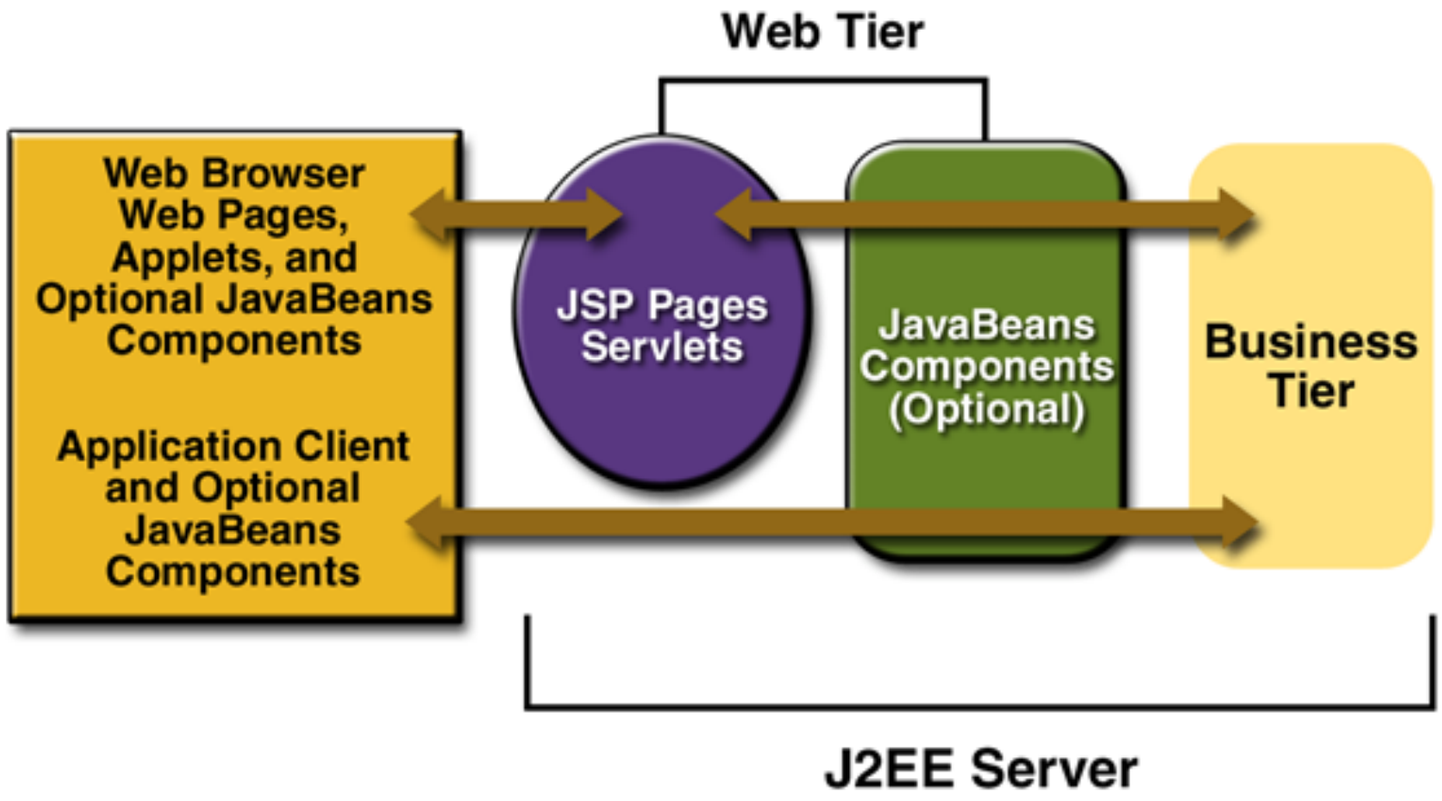
Client Tier

- Web clients (“thin client”)
  - Render web pages provided by web tier
  - Pages may contain Java applets (embedded client application, running in the browser)
- Application clients (“rich client”)
  - Direct connection to business tier (EJB)



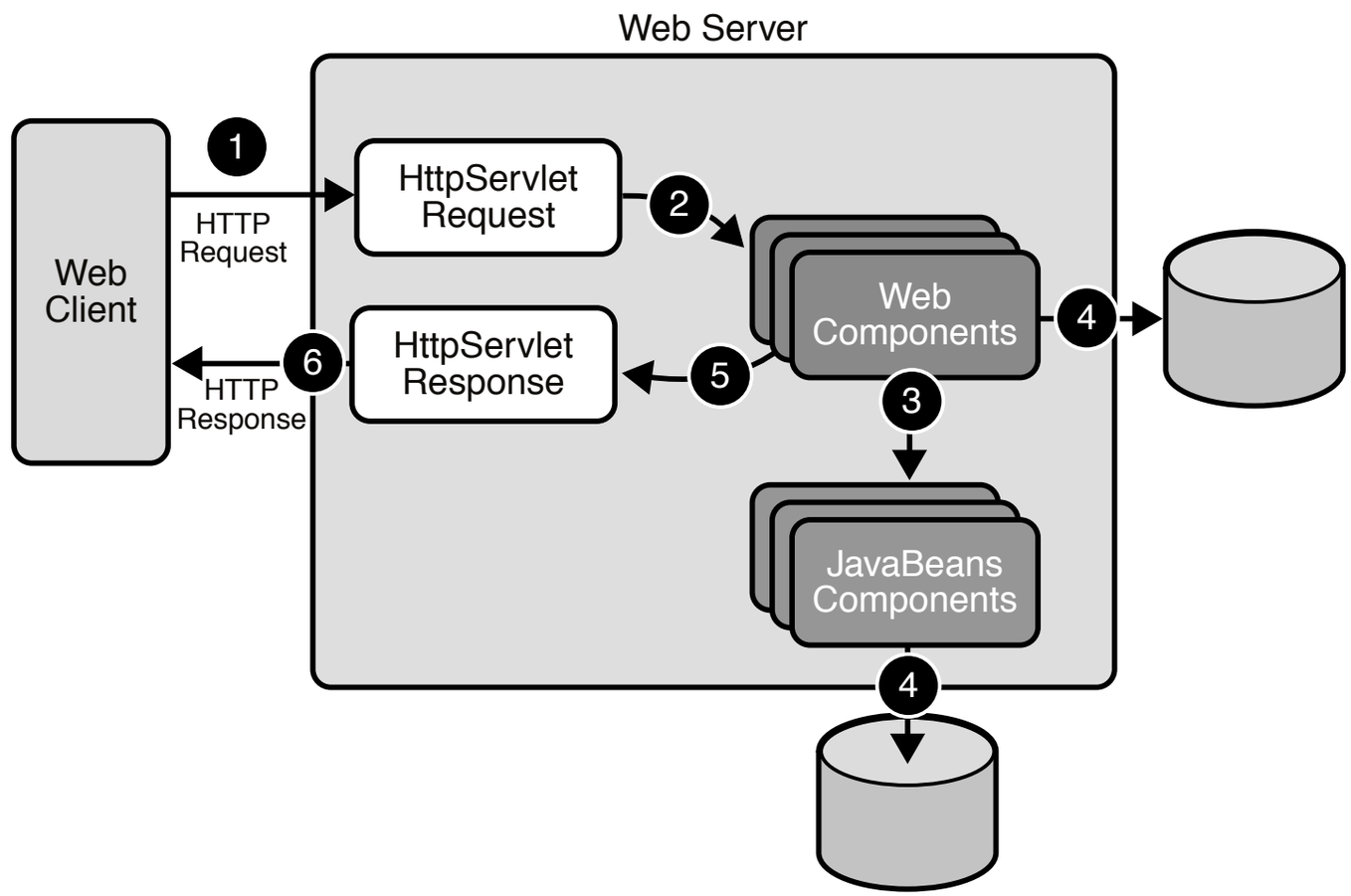
Web Tier

- Servlets: Java classes, which dynamically process requests and construct responses
- JavaServer pages: Text-based documents, includes servlet snippets
    - Static text data - HTML, WML, XML; JSP elements for dynamic content
    - Static HTML, applets, utility classes - bundled with J2EE application



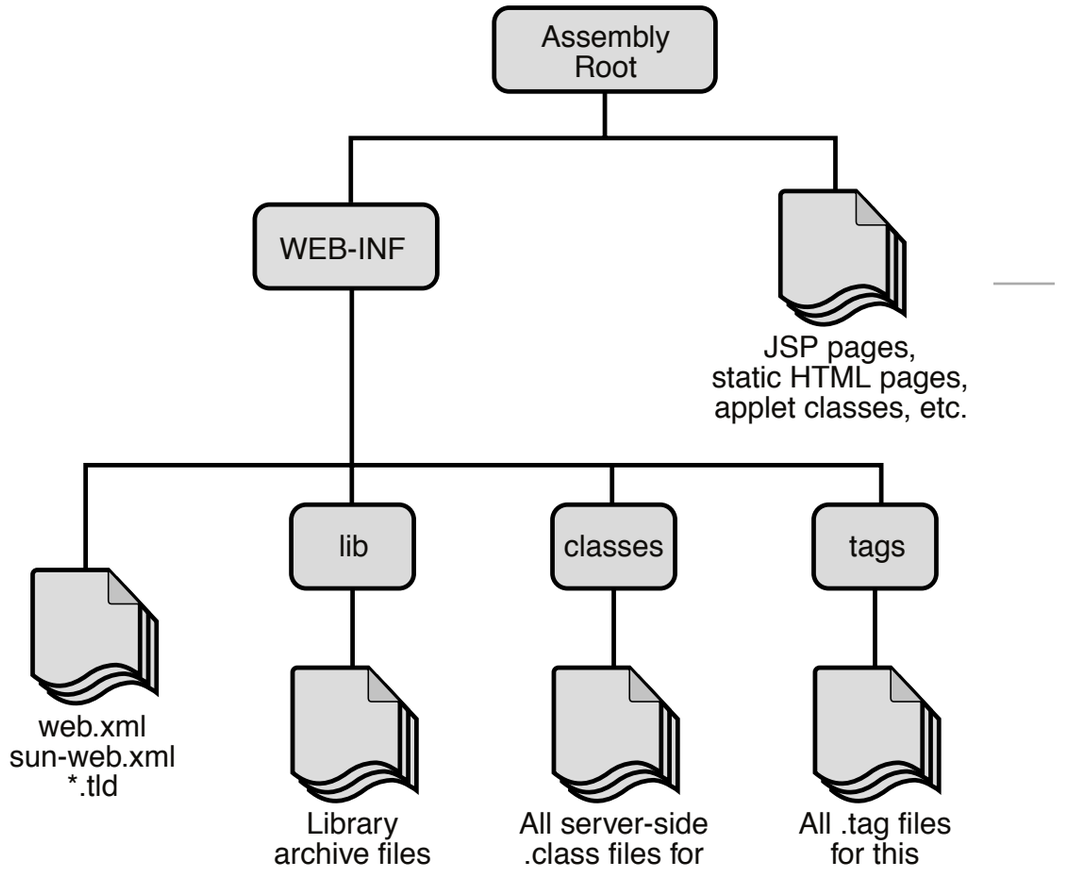
Web Applications

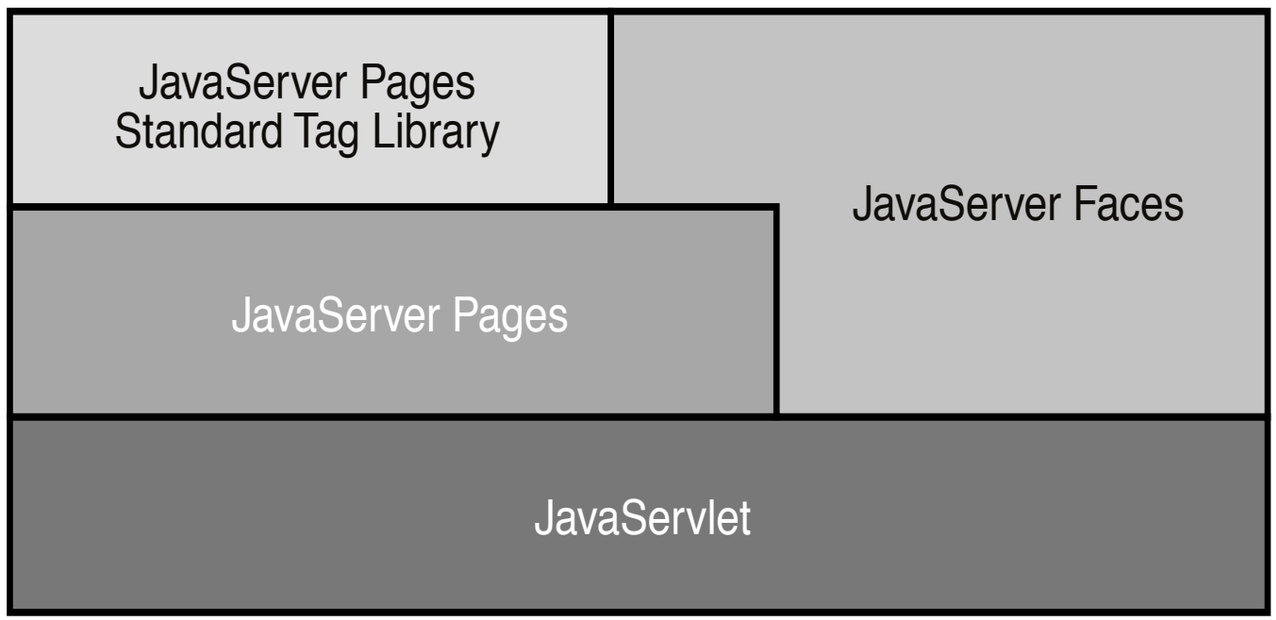
- Presentation-oriented web application (JSP pages for generation of markup)
- Service-oriented web application (web service endpoints with servlets)
- Web container as runtime platform
- Request dispatching
- Security
- Concurrency
- Life-cycle management
- API for components



Web Modules

- Web module as deployable unit
- .WAR file with portable format
  - see Servlet specification
- Support for unpackaged modules
- Context root for web module is configured in server-specific deployment descriptor

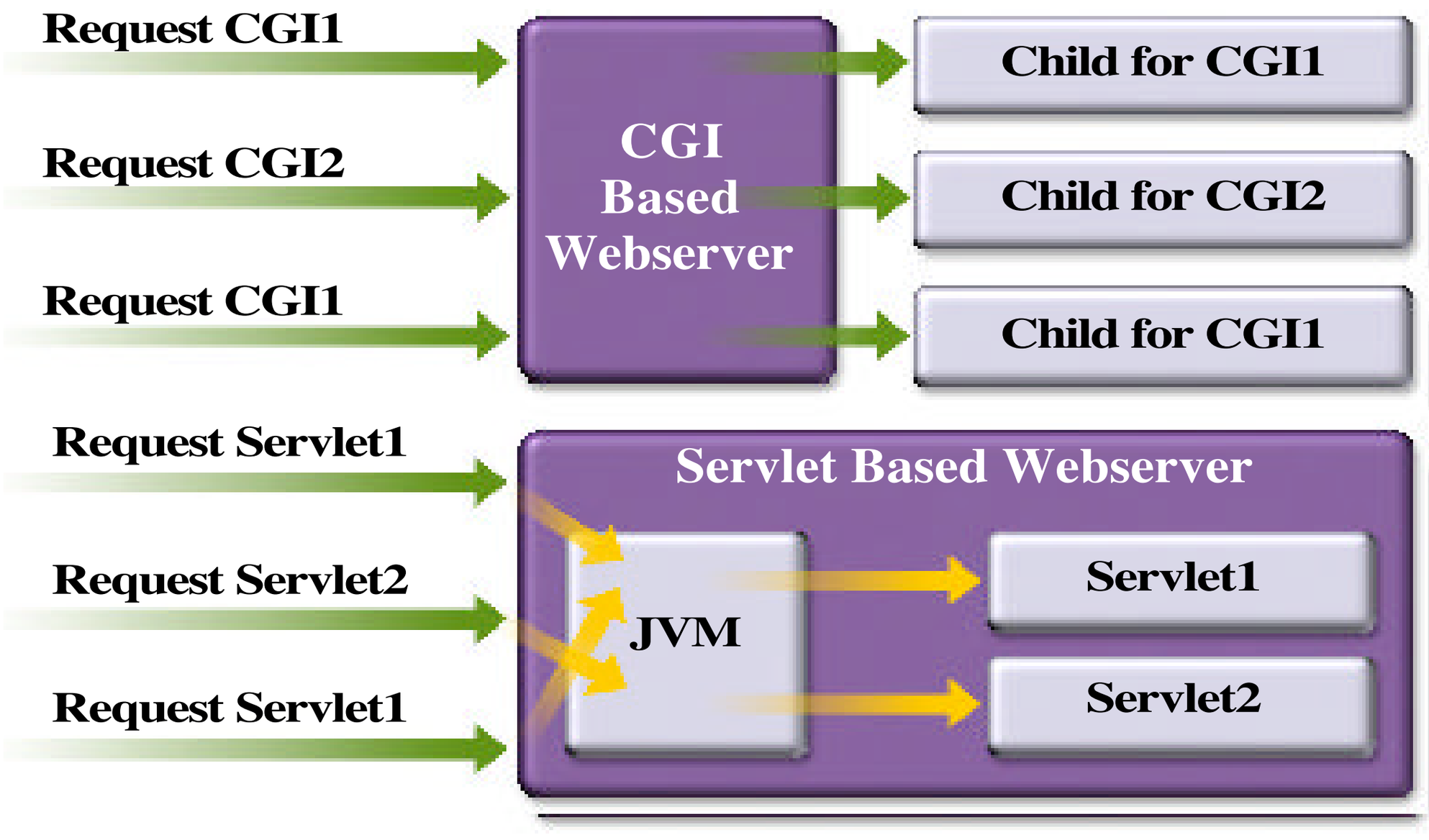




Java Servlets

- Portable way for dynamic web content
- For any kind of request-response pattern, HTTP-specific servlet classes (javax.servlet.http)
- Servlet life cycle: Instantiation through container, *init* method, service method, *destroy* method
- Possibility for event listeners in the WAR file (initialization, destruction, request, activation, passivation, invalidation)
- Concurrent access, synchronization in the code

Servlet vs. CGI



Servlet Life Cycle

- `public void init() throws ServletException {}`
- `public void destroy() {}`
- For `GenericServlet`, override
  `public void service(ServletRequest, ServletResponse)`
- For `HttpServlet`, override
  `public void doGet(...) | doPost(...) | doPut(...) | doDelete(...)`
  - `service()` method as dispatcher
- Request object contains all relevant data (e.g. `HttpServletRequest`)
- Response object filled by servlet (e.g. `HttpServletResponse`)

Servlet Example

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorld extends HttpServlet {
  public void doGet(HttpServletRequest req,
                    HttpServletResponse res)
    throws ServletException, IOException
    {
      res.setContentType("text/html");
      PrintWriter out = res.getWriter();
      out.println("Hello World");
    }}
```

Information Sharing

- Through private helper objects
- Through sharing of objects that are attributes of a public scope
- Through a database
- Through invocation of other web resources
- Synchronization is task of the developer

```
import java.io.*; import java.util.*;
import javax.servlet.*; import javax.servlet.http.*;

public class HolisticCounter extends HttpServlet {
  static int classCount = 0;  // shared by all instances
  int count = 0;              // separate for each servlet
  static Hashtable instances = new Hashtable();  // also shared

  public void doGet(HttpServletRequest req, HttpServletResponse res)
                                throws ServletException, IOException {
    res.setContentType("text/plain");
    PrintWriter out = res.getWriter();
    count++;
    out.println("Since loading, this servlet instance has been accessed " +
                count + " times.");

    instances.put(this, this);
    out.println("There are currently " + instances.size() + " instances.");
    classCount++;
    out.println("Across all instances, this servlet class has been " +
                "accessed " + classCount + " times.");
}}
```

Client Sessions

- Session is created automatically
- Timeout period definition in WAR deployment descriptor

```
public class CashierServlet extends HttpServlet {
  public void doGet (HttpServletRequest request,
                     HttpServletResponse response)
              throws ServletException, IOException {
// Get the user's session and shopping cart
HttpSession session = request.getSession();
ShoppingCart cart = (ShoppingCart)session.getAttribute
                    ("cart");
...
// Determine the total price of the user's books
double total = cart.getTotal();
```

JavaServer Pages

- Combination of presentation and business logic code, as with PHP, ASP.NET, Ruby, ...
- JSP language supports access to server-side objects and tag libraries
- Mixture of static and dynamic content
  - Static data: HTML, SVG, WML, XML, ...
  - JSP elements, expressed in ‘standard’ syntax (.jsp) or XML syntax (.jspx)
- In case, container translates JSP page to a servlet class and compiles the class

JSP Syntax

- JSP expression: `<%= expression %>`
- JSP Scriptlet: `<% code %>`
- JSP Declaration: `<%! code %>`
- JSP page directive: `<%@ page att="val" %>`

- Always XML equivalent defined

```
<jsp:expression>
<jsp:directive.page att="val"\>
```

JSP Example

```
<%@ page contentType="text/html; charset=UTF-8" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML><HEAD>
<TITLE>Using JavaServer Pages</TITLE>
<BODY BGCOLOR="#FDF5E6" TEXT="#000000" LINK="#0000EE">
<UL><LI><B>Expression.</B><BR>
    Your hostname: <%= request.getRemoteHost() %>.
  <LI><B>Scriptlet.</B><BR>
    <% out.println("Attached GET data: " +request.getQueryString()); %>
  <LI><B>Declaration (plus expression).</B><BR>
    <%! private int accessCount = 0; %>
    Accesses to page since server reboot: <%= ++accessCount %>
  <LI><B>Directive (plus expression).</B><BR>
    <%@ page import = "java.util.*" %>
    Current date: <%= new Date() %>
</UL></BODY></HTML>
```

Further JSP Features

- HTTP request forwarding (`jsp:forward`)
- Use JavaBeans within the JSP page

```
<jsp:useBean id="test" class="hall.SimpleBean" />
<jsp:setProperty name="test" property="message"
                 value="Hello WWW" />
<jsp:getProperty name="test" property="message" />
```

  - Scope attribute (`PageContext (default), ServletRequest, HttpSession, ServletContext`)
- Inclusion on compile / execution time

```
<%@ include file="url" %>
<jsp:include page="relative URL" />
```

Extended Tags with JavaServer Faces (JSF)

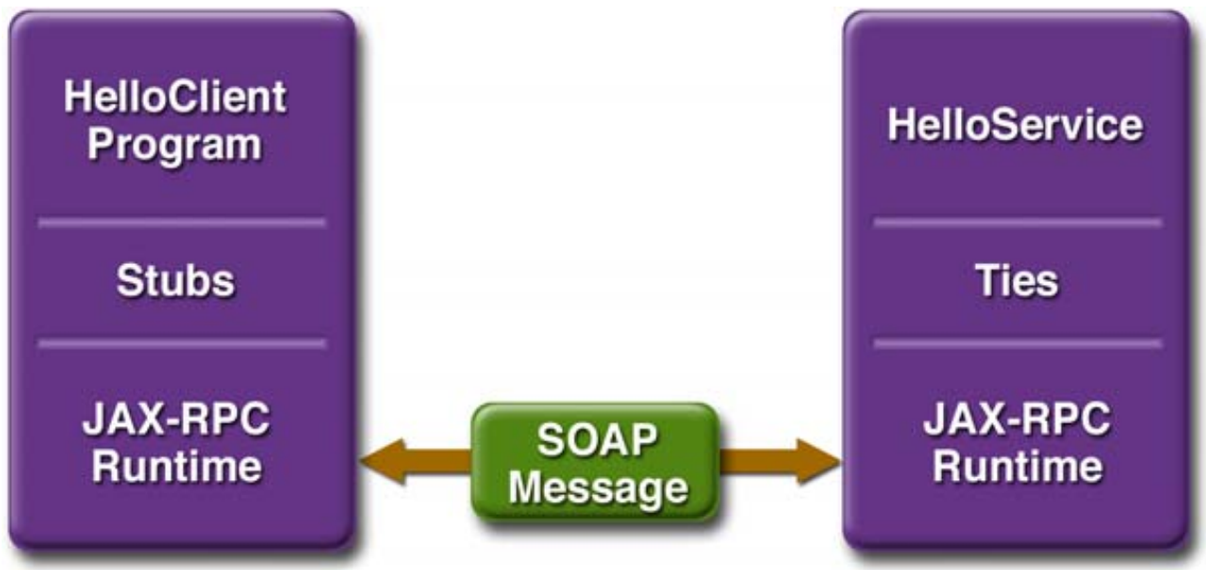
```
<HTML><HEAD> <title>Hello</title></HEAD>
  <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
  <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
  <body bgcolor="white">
  <f:view><h:form id="helloForm" >
      <h2>Hi. My name is Duke.  I'm thinking of a number from
          <h:outputText value="#{UserNumberBean.minimum}"/> to
          <h:outputText value="#{UserNumberBean.maximum}"/>.
          Can you guess it?</h2>
      <h:graphicImage id="waveImg" url="/wave.med.gif" />
      <h:inputText id="userNo" value="#{UserNumberBean.userNumber}">
        <f:validateLongRange minimum="#{UserNumberBean.minimum}"
                             maximum="#{UserNumberBean.maximum}" />
      </h:inputText>
      <h:commandButton id="submit" action="success" value="Submit" />
      <h:message style="color: red” id="errors1" for="userNo"/>
  </h:form></f:view>
</HTML>
```

Security in Web Applications

- HTTPS support is mandatory in J2EE-compliant web containers
- Security requirements: User identification and authentication, user account systems, authorization, audit logs, authentication schemes
- Declarative security (web.xml) vs. programmatic security (HttpServletRequest methods)

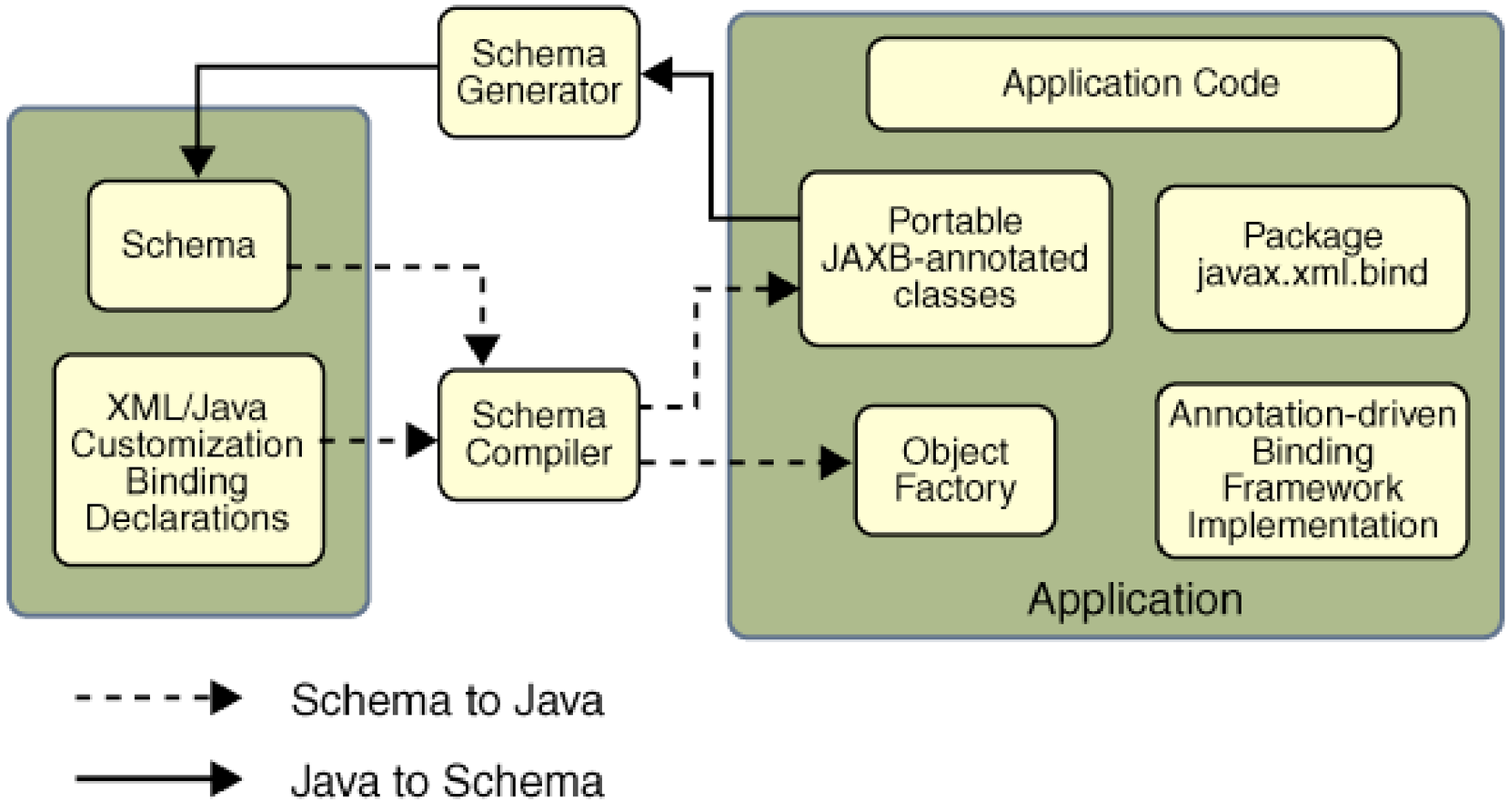
JAX-RPC

- XML-based RPC API for building web services
- No need for generating or parsing SOAP data (proxy / stub generation)
- Service Endpoint Interface (SEI) in Java, only JAX-RPC types allowed
- since June 2002 (before J2EE 1.4)
- No service-specific exceptions with DII
- J2ME supports only static stubs
- J2EE container manages service access (service.getPort())



JAX-WS

- Successor of JAX-RPC since EE 5, relies on Java 5 annotations
- Service Endpoint Interface (SEI) in Java, only JAXB types allowed
  - Binding between XML schema and Java types
- JAX-WS 2.0 supports WS-I Basic Profile 1.1



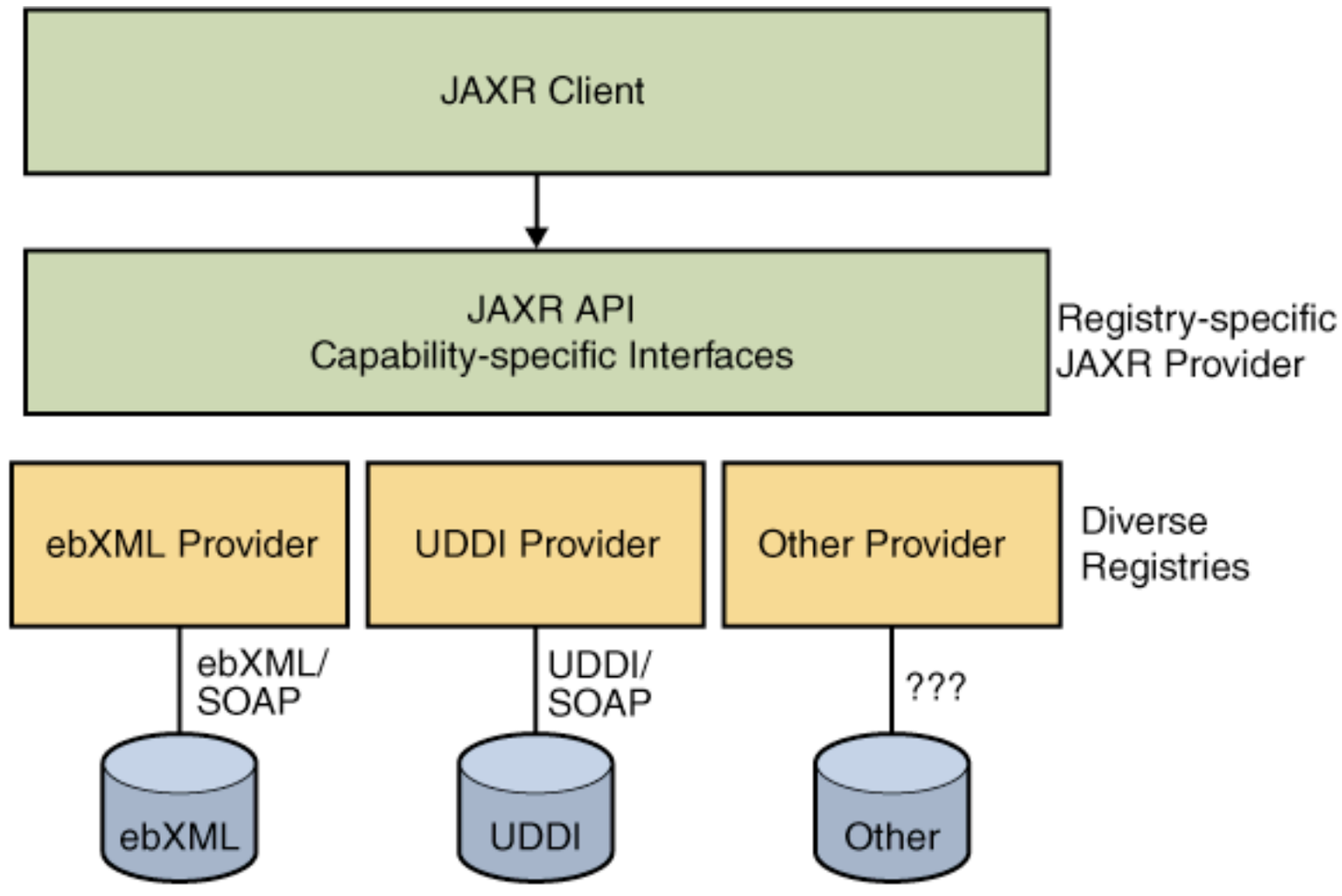
Coding a JAX-WS Web Service

- Code and compile the implementation class
- Use wsgen for the generation of descriptors and mapping files

```
package helloservice.endpoint;
import javax.jws.WebService;

@WebService
public class Hello {
  private String message = new String("Hello, ");

  public void Hello() {}

  @WebMethod
  public String sayHello(String name) {
    return message + name + ".";
  }
}
```

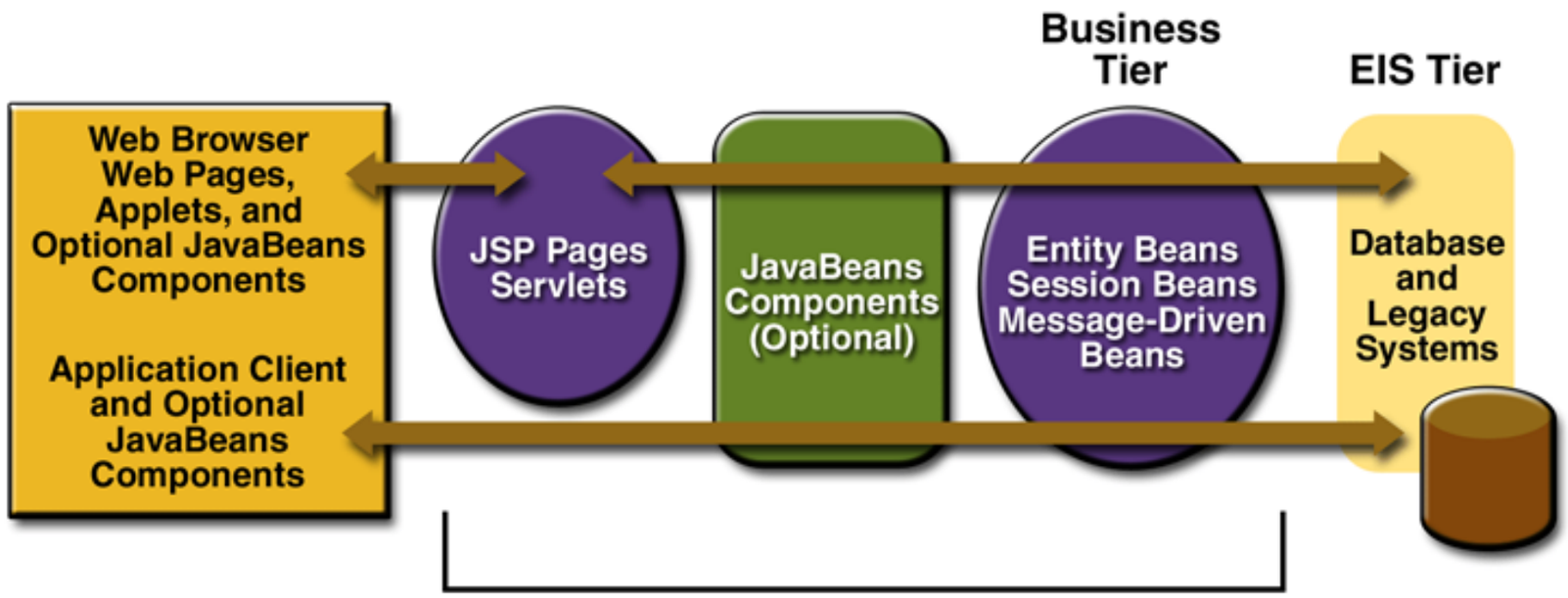
Java API for XML Registries

- JAXR: Uniform API to XML registries for web services
- ebXML registry and repository standard
- UDDI standard



Business Tier

- Contains reusable components with business code
  - Program logic which solves the need of a business domain functionality
- Business components are reflected as Enterprise JavaBean (EJB)
  - Different Bean types (Session, Message)
  - J2EE 1.4 Entity Beans replaced by Persistence API

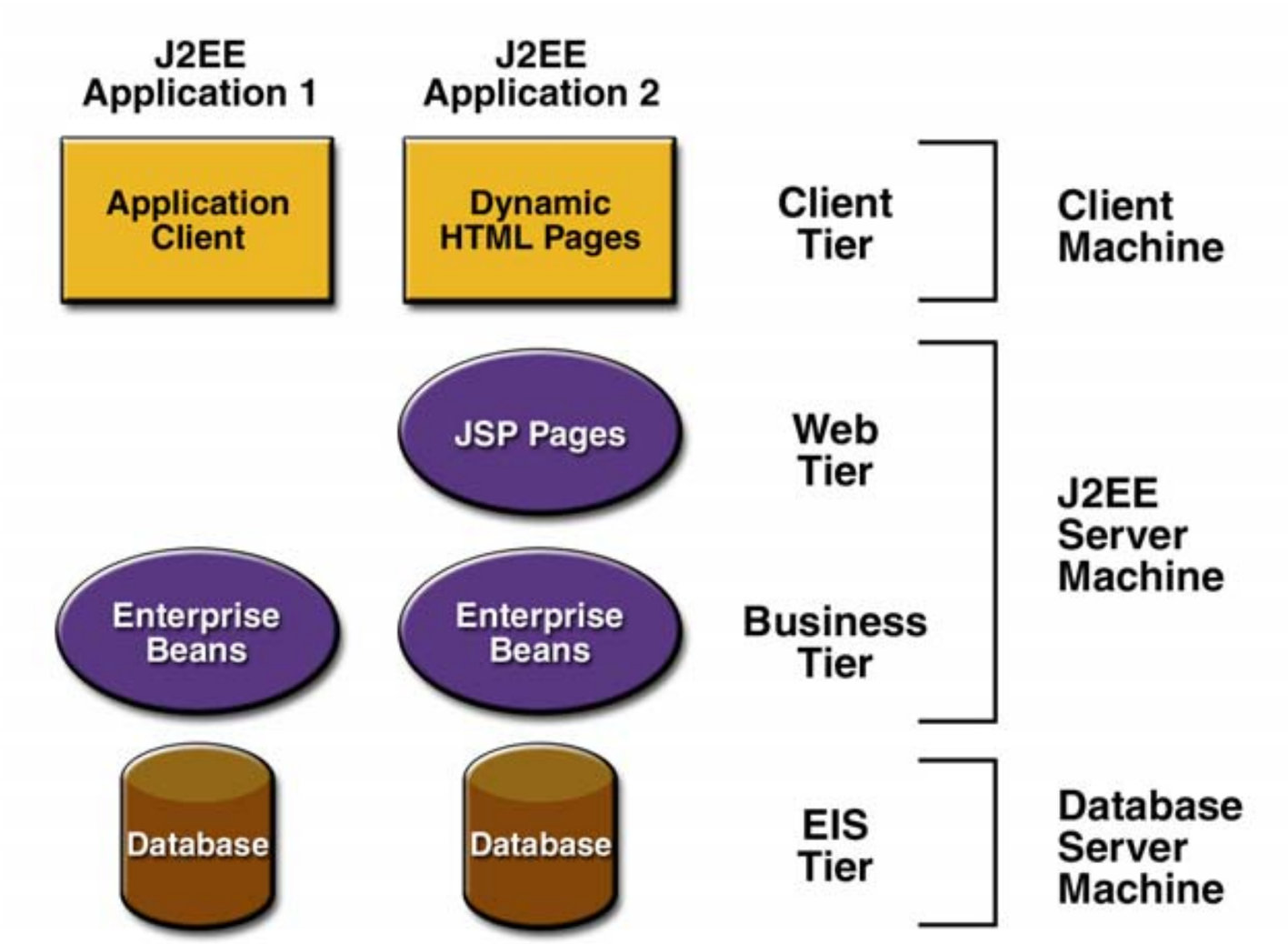


JNDI

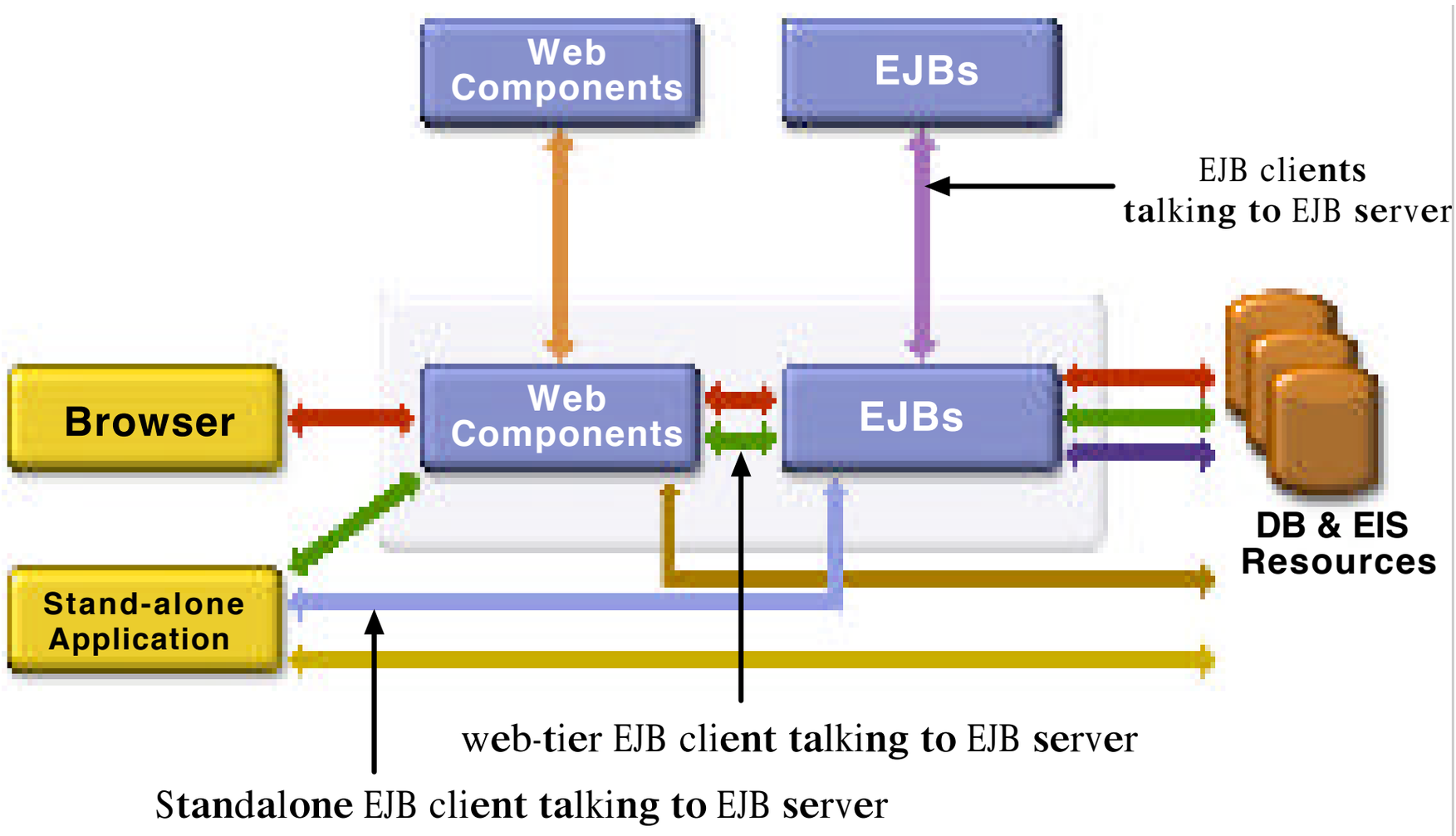
- Java Naming and Directory Interface
- Associate names and attributes with objects, provide way to access the objects by their name
- Binding: Association between a name and an object
- Context: Set of bindings, name can be bound to another subcontext (similar to FS directory structure)
- Entry point with `javax.naming.InitialContext` class
- Provider concept
  (LDAP, RMI registry, DNS, filesystem, NDS, NIS)

Enterprise Java Beans

- Development and deployment of scalable, transactional, server-side business logic components - the ‚heart‘ of J2EE / EE 5
- EJB container as runtime environment
- EJB client: servlet, application, other bean
- EJB 2.1 / EJB 3.0 specification



Java EE & EJB


Web Components
EJBs
EJB clients talking to EJB server
Browser
Web Components
EJBs
DB & EIS Resources
Stand-alone Application
web-tier EJB client talking to EJB server
Standalone EJB client talking to EJB server

EJB Design Principles

- EJB applications are loosely coupled
  - Access to other components / services (from other vendors) through arbitrary names
  - Can be authored without detailed knowledge of the environment
  - Assembling to an application without code change
- EJB behavior is entirely specified by interfaces
- EJB applications never manage resources
  - Access to external resource through their container
  - Container resource management (allocation, deallocation, sharing, pooling)
- Container configuration as administrative task, no programmatic interface; container provides system services

Annotations vs. Descriptors

- Only RUNTIME annotations used
  - Available from class file, read by runtime
- Deployment descriptors overwrite annotation configuration
- Lifetime management through annotated functions (@PreDestroy, @PostConstruct)

```
@Stateless
@Remote
public class HelloWorldBean {
  public String getHello() {
     return „Hello World“; }
  @PreDestroy cleanUp() {
     // logic before destruction of the instance
}}
```

Without Resource Injection

```
<resource-ref>
    <description>Catalog DataSource</description>
    <res-ref-name>jdbc/catalogDS</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
</resource-ref>
```

```
public javax.sql.DataSource getCatalogDS() {
    try {
        // Obtain JNDI context.
        InitialContext initCtx = new InitialContext();
        // Perform JNDI lookup to obtain the resource.
        catalogDS = (DataSource)
            initCtx.lookup("java:comp/env/jdbc/catalogDS");
    } catch (NamingException ex) {
}}...

public getProductsByCategory() {
    DataSource catalogDS = getCatalogDS();
    Connection conn = catalogDS.getConnection();
...}
```

With Resource Injection

```
private @Resource DataSource catalogDS;

public getProductsByCategory() {
    // Get a connection and execute the query.
    Connection conn = catalogDS.getConnection();
    ...
}
```

- Field is injected by the container before the component is made available
- Data source JNDI mapping is inferred from field name and data type
- No deployment descriptor needed, but possible for override
- Also useful for injecting JMS message destinations or Web Service references
- More powerful solutions for dependency injection from Open Source community (e.g. Spring framework, Pico container, Jakarta HiveMind)

EJB 3 Example

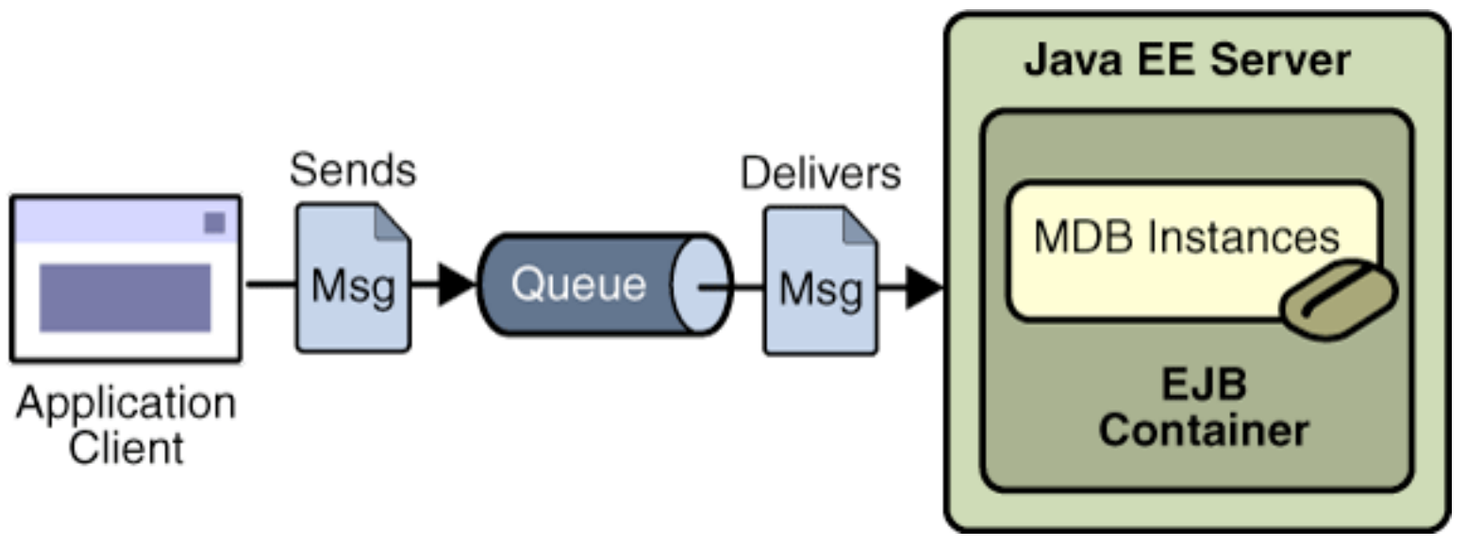
```
@Stateless public class PayrollBean
    implements Payroll {
  @Resource DataSource empDB;
   public void setBenefitsDeduction
     (int empId, double deduction) {
       ...
       Connection conn = empDB.getConnection();
       ... } ... }
-----------------------------------------------
@EJB ShoppingCart myCart;
Collection widgets = myCart.startToShop(“widgets”);
```

Session Beans

- Purpose: Performs a task for a client, similar to an interactive session; hides business task complexity
- Only one client per session bean at the same time
- Stateless Session Beans
  - No conversational state, container can assign any instance to a client
  - Class is annotated as `@Stateless`
- Stateful Session Beans
  - Unique state information per client-bean session (conversational state)
  - Class is annotated as `@Stateful`
  - Support for lifecycle callback functions

Message-Driven Beans

- Short-lived stateless JMS message listener component
  - Messages can be send by any J2EE component, not intended to be called directly by clients or other components
  - All instances of one message bean are equivalent, transaction-aware
- Class annotated with `@MessageDriven`, implements `MessageListener` for `onMessage()` method



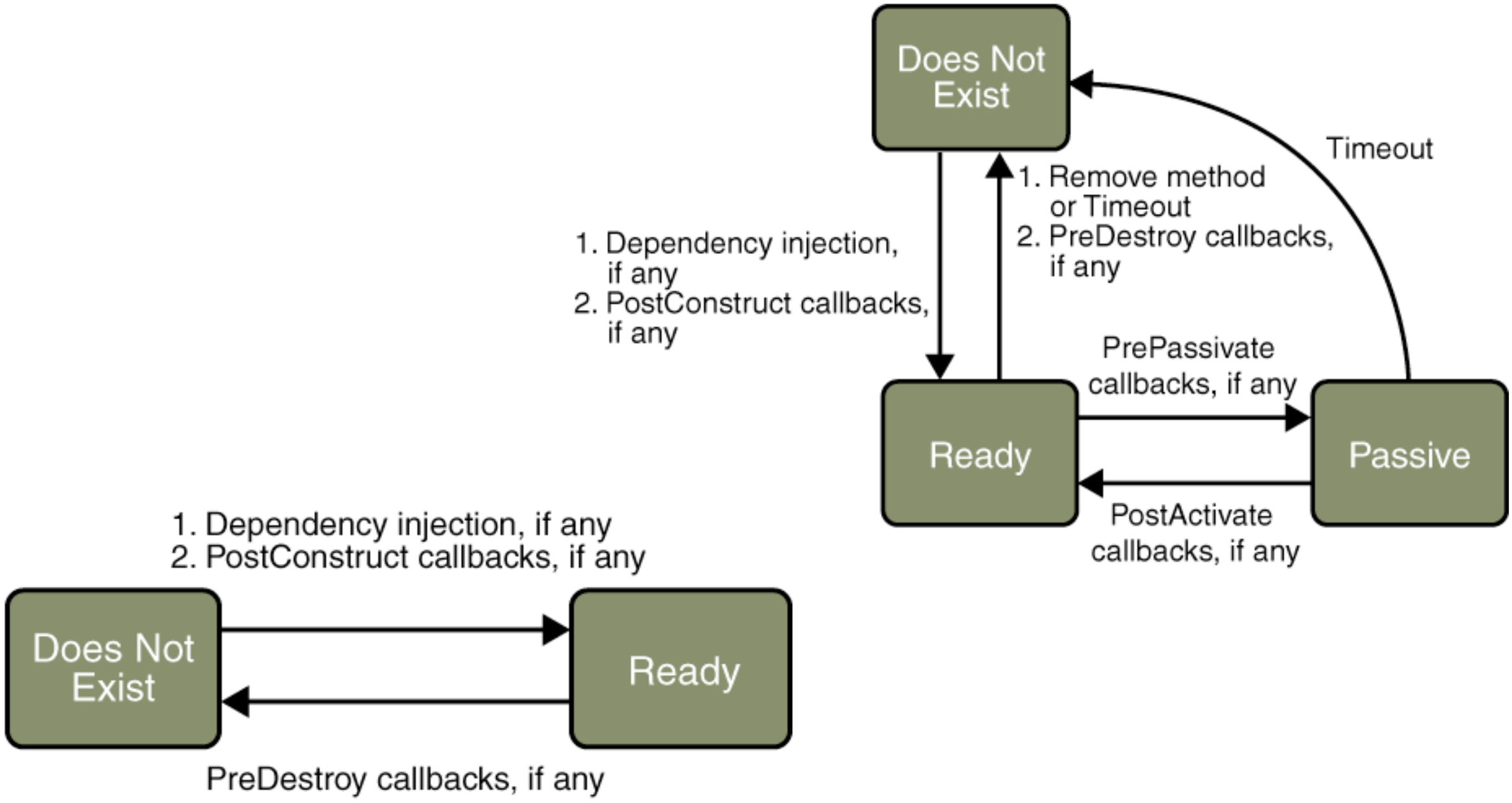
Defining Client Access

- Remote client access
  - Can run on different machine or JVM, location is transparent to client
  - Annotate business interface with `@Remote`
- Local client access
  - Client must run in the same JVM as the bean
  - Client and bean can manipulate the same parameter object
  - Annotate business interface with `@Local`
- Web service access (`@WebService , @WebMethod`)

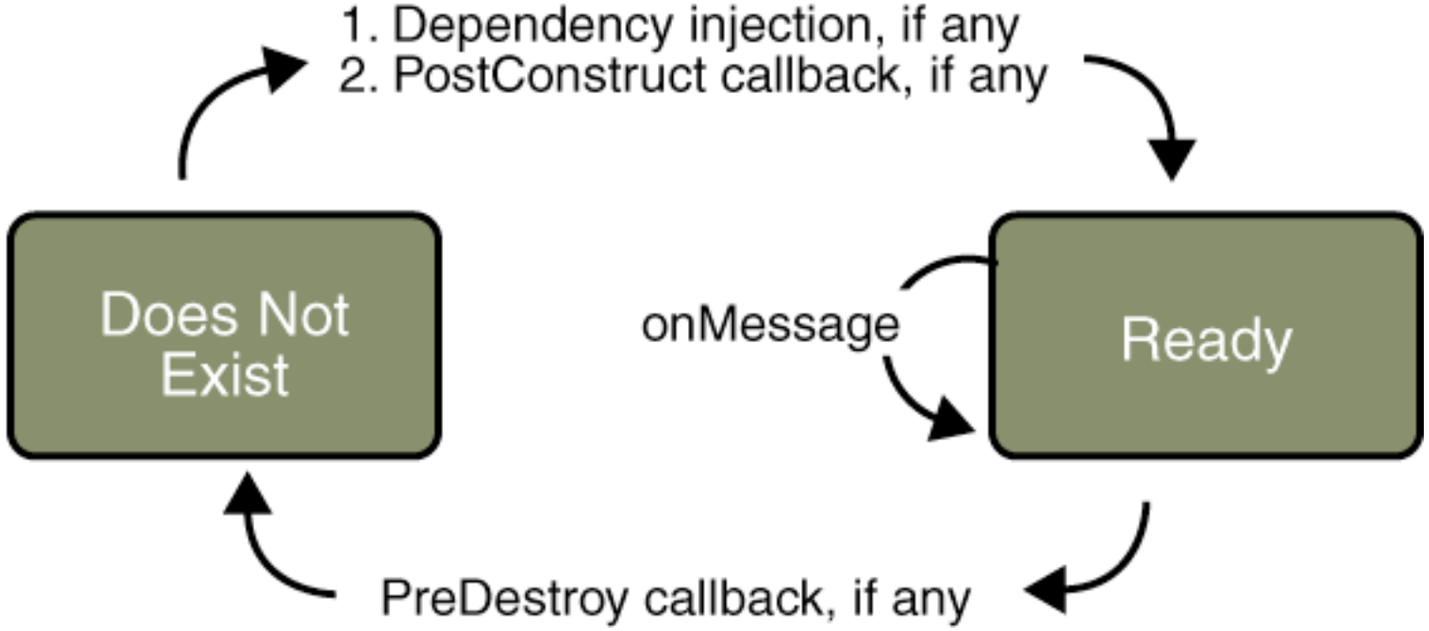
Web Service Example

```
@WebService(name="MySimpleWS");
public class RandomClass {
  @WebMethod
  public String sayHello(String s) {...}
  public void unpublished() {...}                    }
-------------------------------------------------------
@WebServiceRef(
wsdlLocation=
      "http://localhost:8080/SayHelloService?WSDL");)

static javaone.SayHelloService wsService;

public static void main(String[] args) {
  javaone.SayHello wsPort = wsService.getHello();
  wsPort.sayHello("FOSS.in Attendees");              }
```

Session Bean Lifecycle

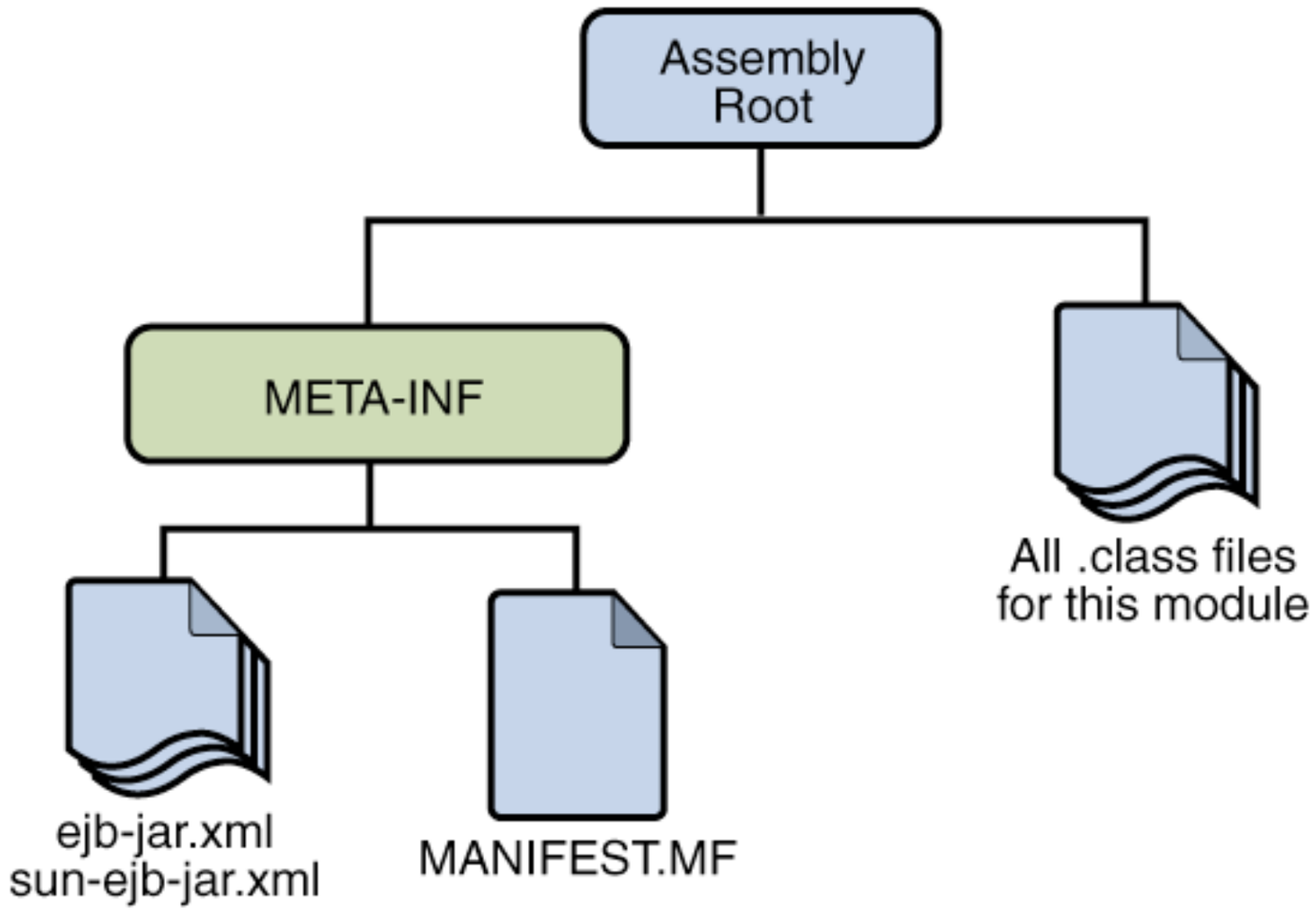


Message-Driven Bean Lifecycle



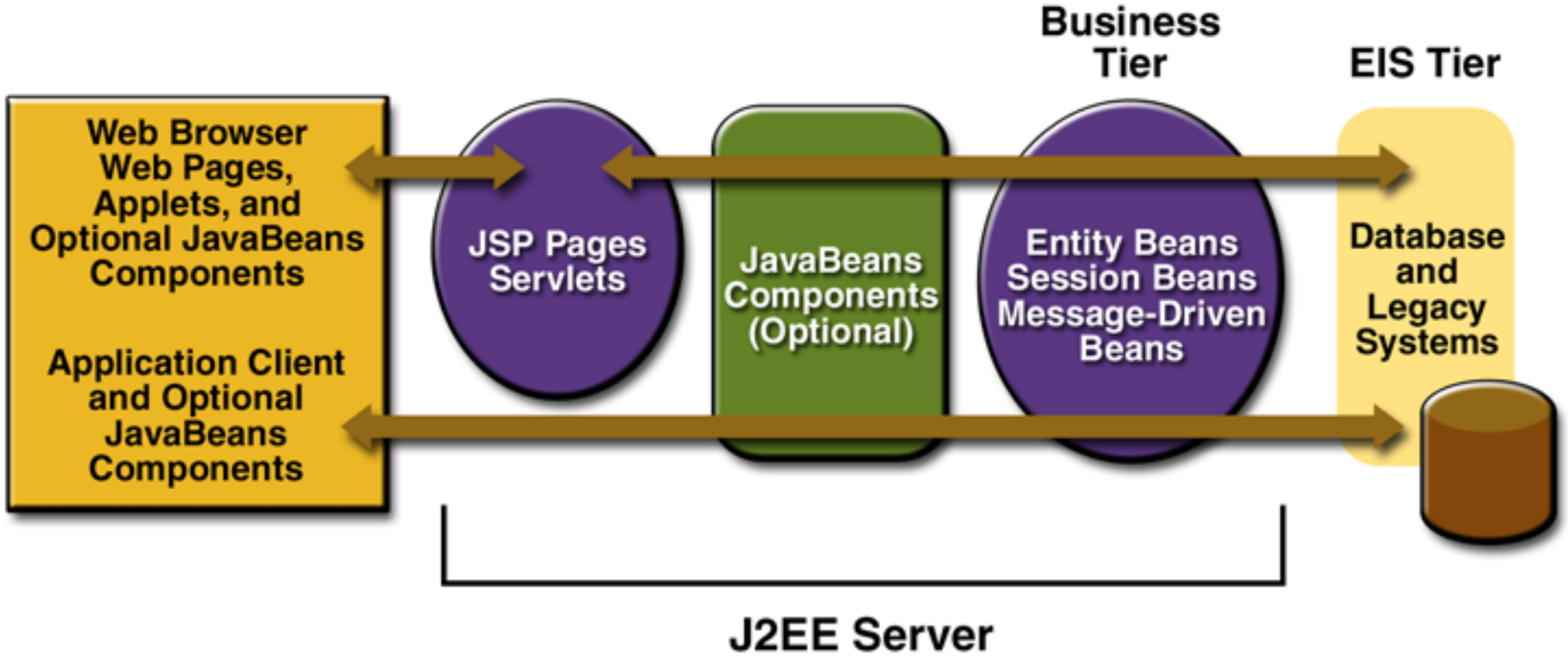
EJB Packaging

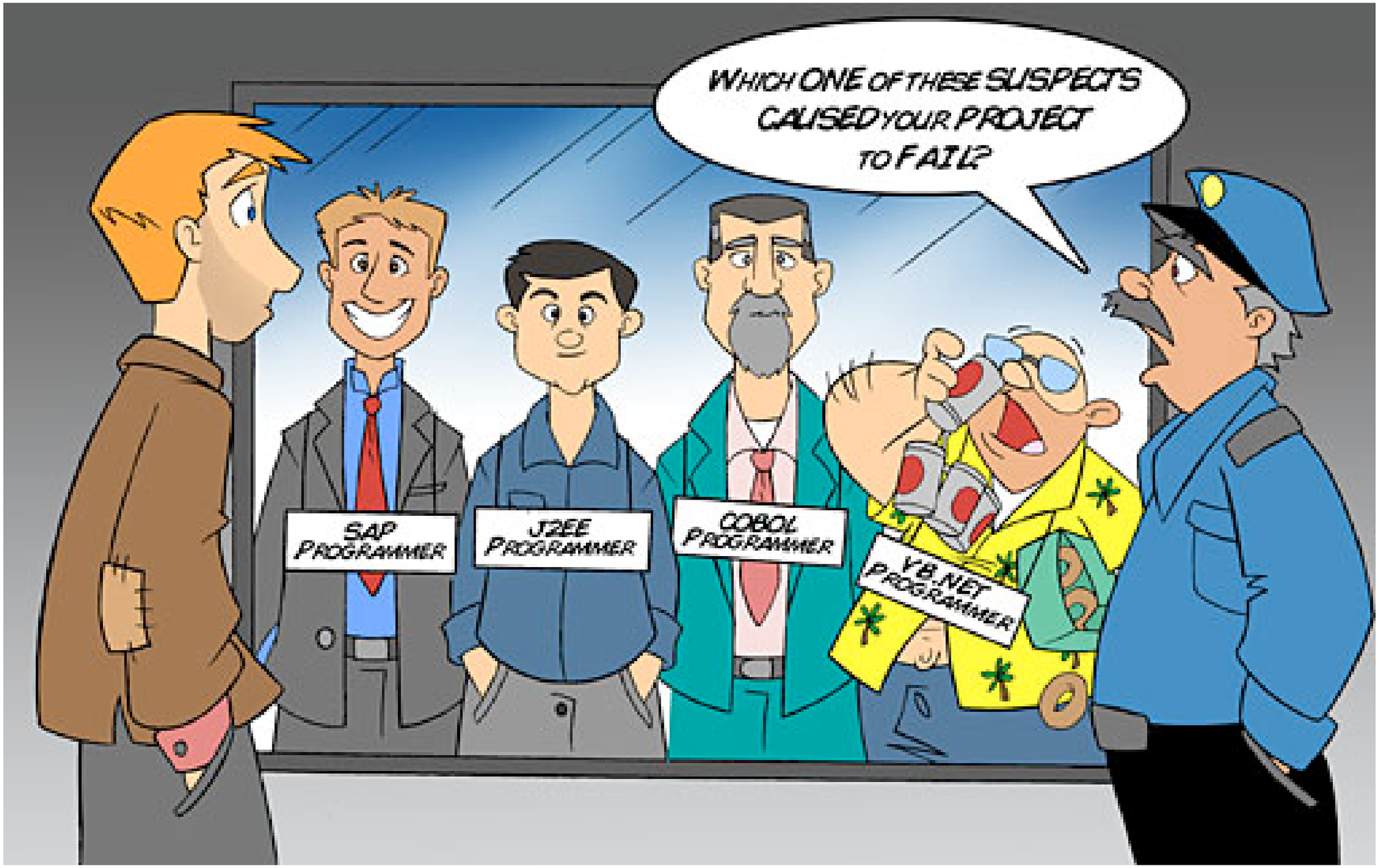
- Deployment descriptor (e.g. persistency type, transaction attributes)
- Enterprise bean classes, helper classes and interfaces



EIS Tier

- Coupling with Enterprise Information Systems
  - Enterprise resource planning (ERP) systems
  - Mainframe transaction processing systems or database systems
- Integration of existing data and infrastructure




WHICH ONE OF THESE SUSPECTS CAUSED YOUR PROJECT TO FAIL?
SAP PROGRAMMER
J2EE PROGRAMMER
COBOL PROGRAMMER
VB.NET PROGRAMMER