Introduction to Java Platform, Enterprise Edition (JEE)

Fahad R. Golra

ECE Paris Ecole d'Ingénieurs - FRANCE

Course Details

- Contents
  - 10 sessions of 3 hours each
  - 50% theoretical, 50% practical
- Assessment
  - Assignment - 20%
  - Course Project - 20%
  - Surprise Quiz - 10%
  - Final Exam - 50%

Course Contents

- JEE technology overview
- Servlets
- JSP, Expression Language
- Maven, JSTL
- Web Sockets
- Web Services
- EJB, JPA, Hibernate
- Java Servlet Faces

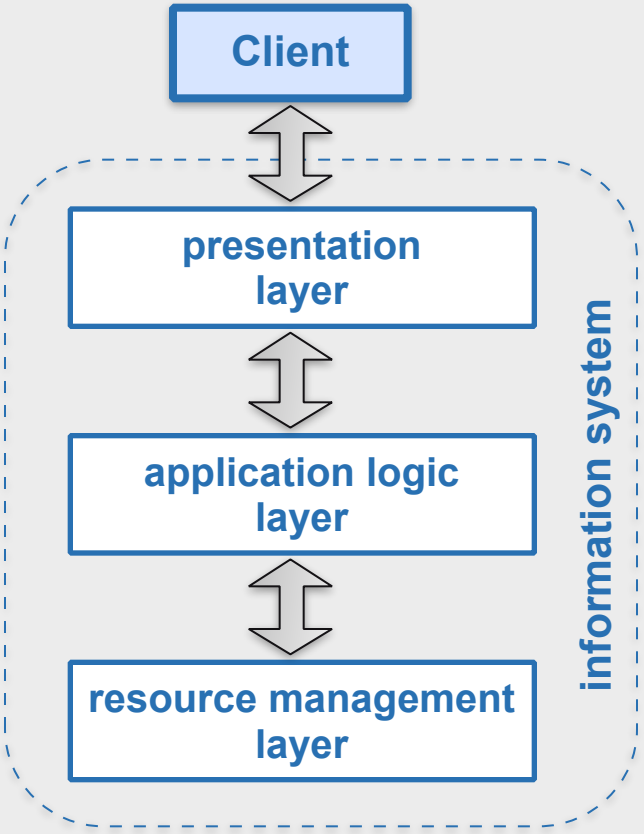
Lecture 1 - JEE Overview

- Multilayer architecture
- Transaction Processing
- 1-Tier, 2-Tier, 3-Tier & N-Tier architecture
- JEE Containers
- JEE architecture
- JEE APIs
- JEE Packaging

Multilayered software architecture

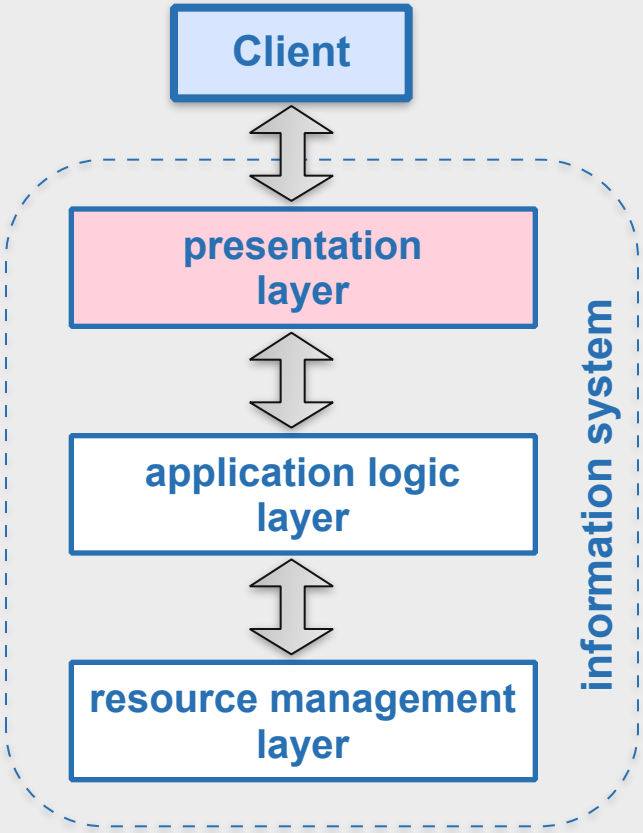
- A multilayered software architecture is a software architecture that uses many layers for allocating the different responsibilities of a software product.
- 'Layer' represents the orientation of the different physical or conceptual elements that make up an entire software solution
- 'Tier' represents the physical layout of the various mechanisms in a system's infrastructure

3 Layers of Information System



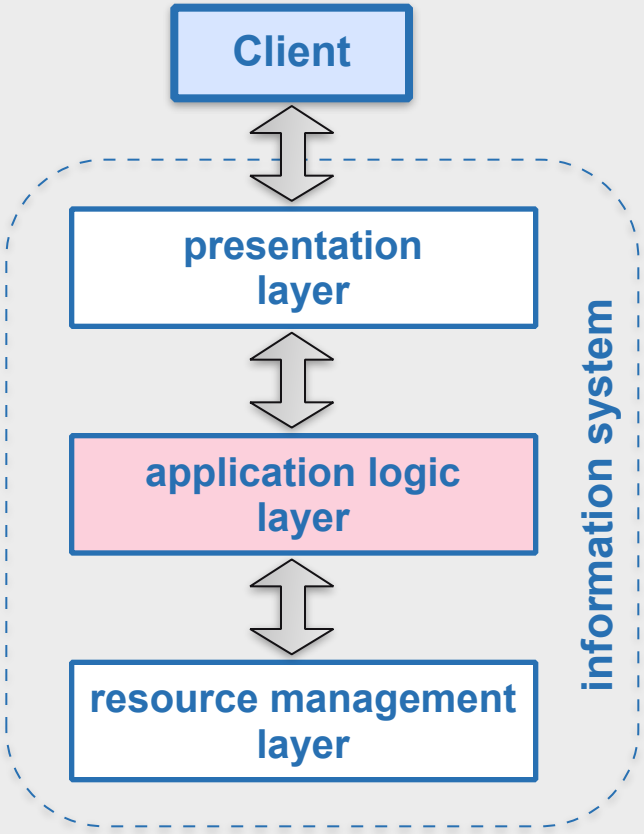
Presentation Layer

- Controls how the information system presents information to external entities and accepts it from them.
- External entities are users (UI) or other information systems (API)



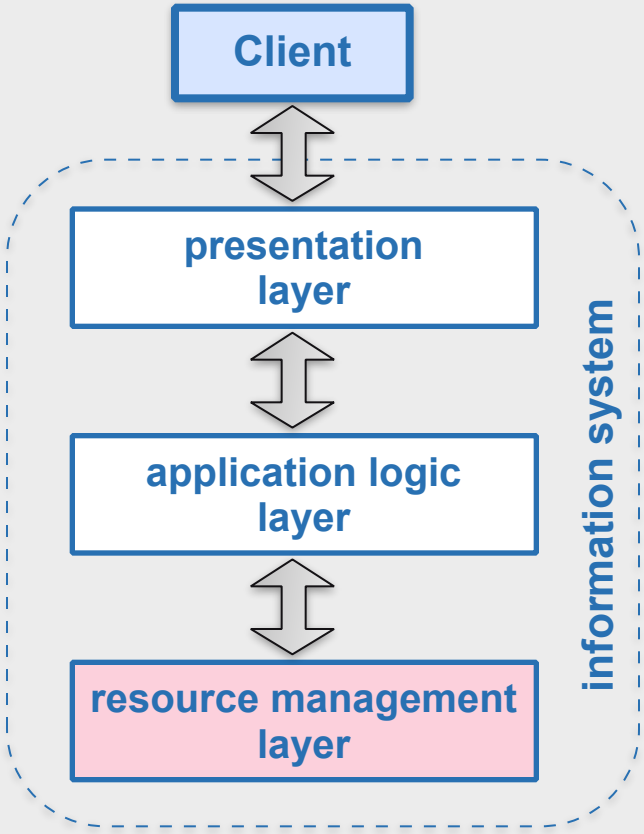
Application Logic Layer

- The program
- Business process
- Business logic
- Business rules



Resource Management Layer

- Implements the resource manager
- Takes care of ACID properties
- We will discuss them in coming slides



Transaction Processing

- Business Transaction
  - Interaction in real world
  - Usually between enterprise & person or between enterprises
- Information processing that is divided into individual, indivisible operations, called transactions
- Performs function on (shared) database

- Online Transaction Processing (OLTP)
  - Runs a collection of transaction programs online

The ACID Properties

- A set of properties that guarantee that transactions are processed reliably
  - Atomicity
  - Consistency
  - Isolation
  - Durability

Atomicity

- All (commit) or nothing (abort)
  - "all or nothing": if one part of the transaction fails, the entire transaction fails
  - Example: transfer money between two bank accounts
- Must handle situations including power failures, errors, and crashes

Consistency

- Each transaction takes valid states to valid states:
  - Satisfy integrity constraints, triggers
- Sometimes the only notion of “valid” state is “a state that could have been produced by executing a sequence of transactions

Isolation

- Each transaction behaves as if it were executed in isolation at some instant in time
- AKA Serializability
  - Ensures that the concurrent execution of transactions results in a system state that would be obtained if transactions were executed serially
- Consistency + Isolation implies the data remains consistent even when multiple transaction programs execute concurrently

Durability

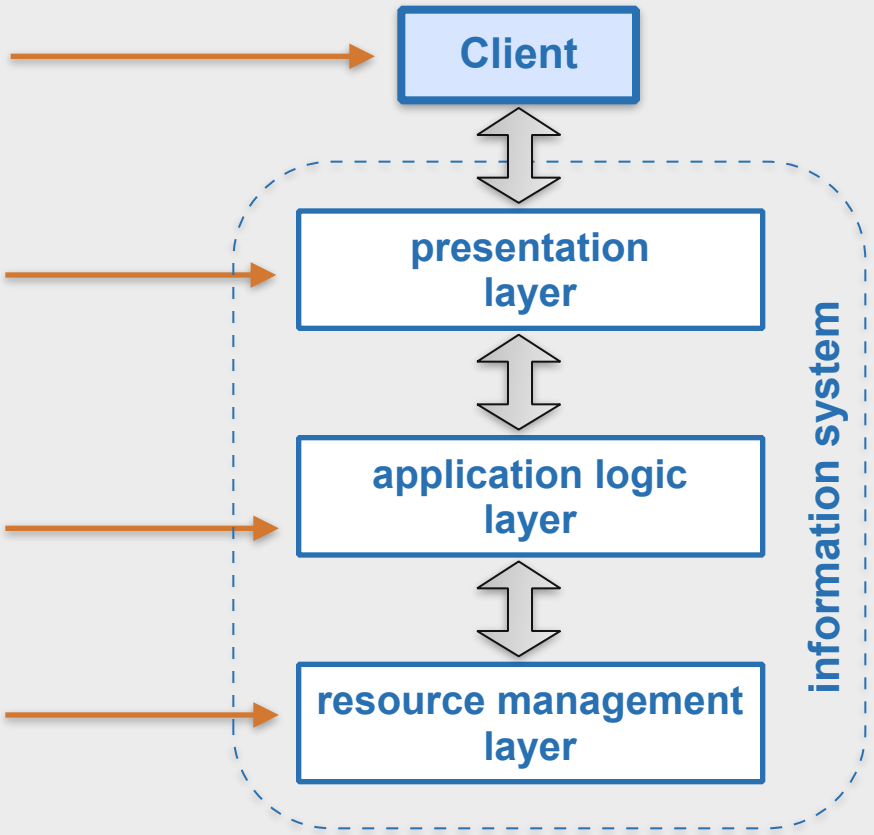
- The effect of a committed transaction will not be lost
  - Even in the event of power loss, crashes, or errors
- So data must be on stable storage before commit
- Usually done with a log (or journal) that must be forced before commit and used in case of crash recovery

Resource Manager

- How ACID transactions are implemented
  - Allocate resources to program executing a transaction e.g. a locked record is a resource
- Reclaim resources in appropriate state on commit or abort
- Handled at “Resource Management Layer”

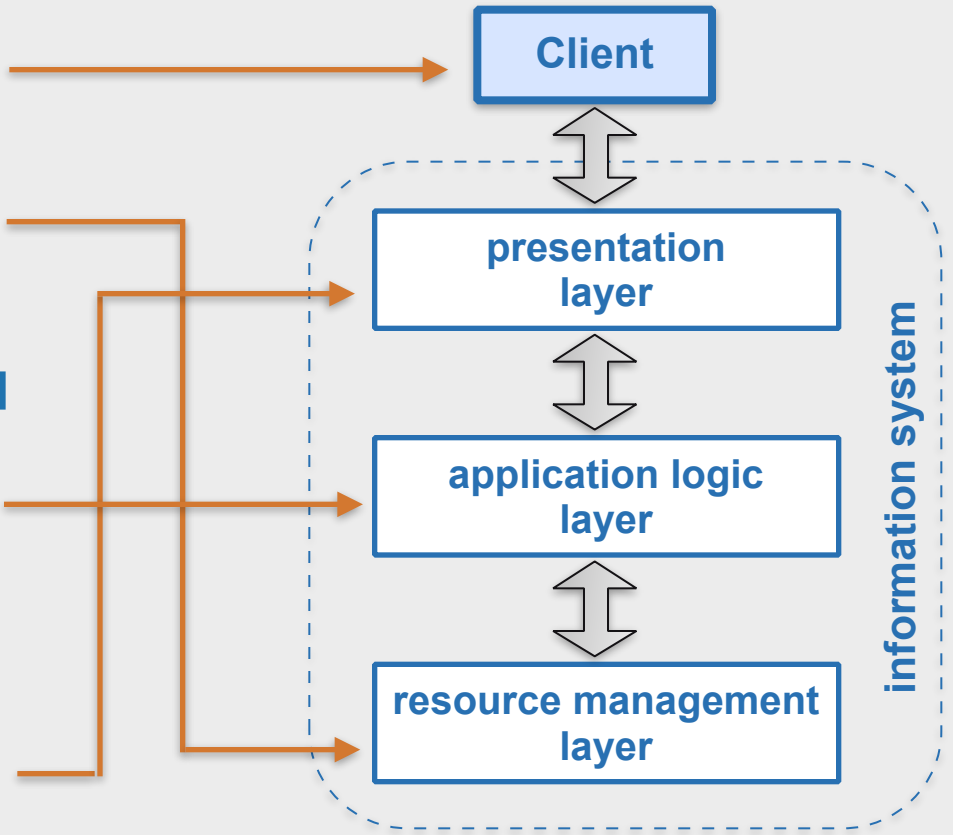
Top down design

1. define access channels and client platforms
2. define presentation formats and protocols for the selected clients and protocols
3. define the functionality necessary to deliver the contents and formats needed at the presentation layer
4. define the data sources and data organization needed to implement the application logic



Bottom up design

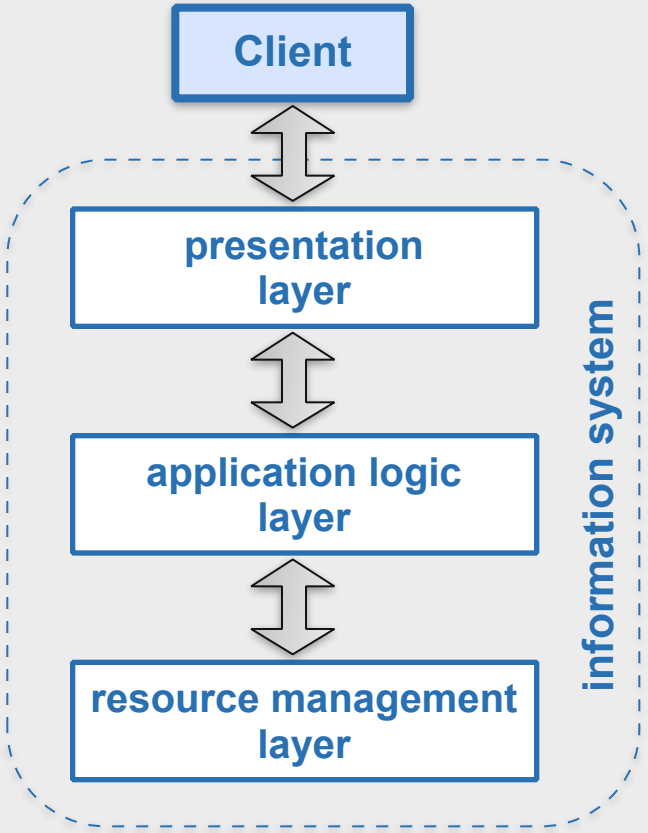
1. define access channels and client platforms
2. examine existing resources and the functionality they offer
3. wrap existing resources and integrate their functionality into a consistent interface
4. adapt the output of the application logic so that it can be used with the required access channels and client protocols



Note: Used for Legacy Systems

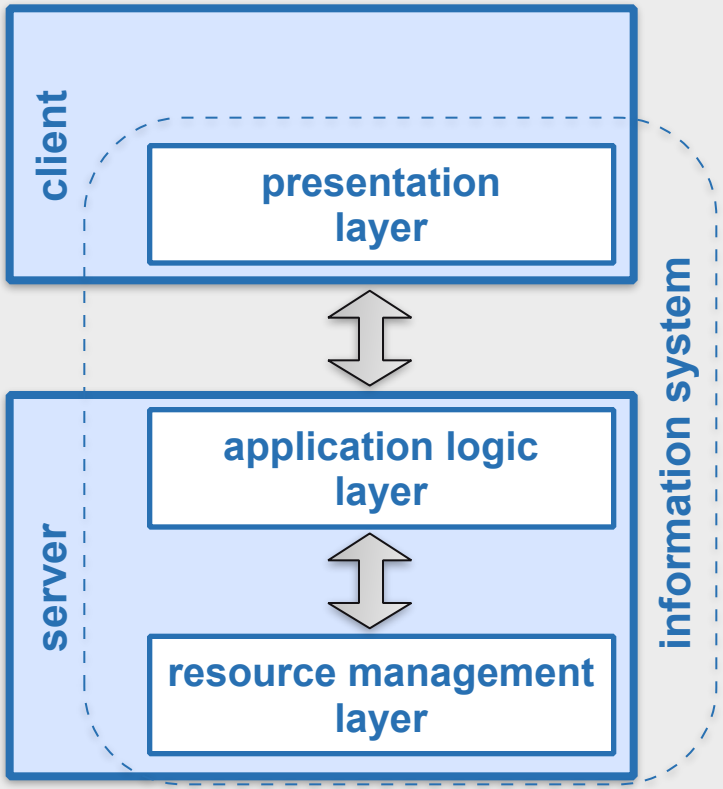
1-Tier Architecture

- System is necessarily monolithic
- May be highly efficient
- No stable service interface API
- Problem of Legacy Systems



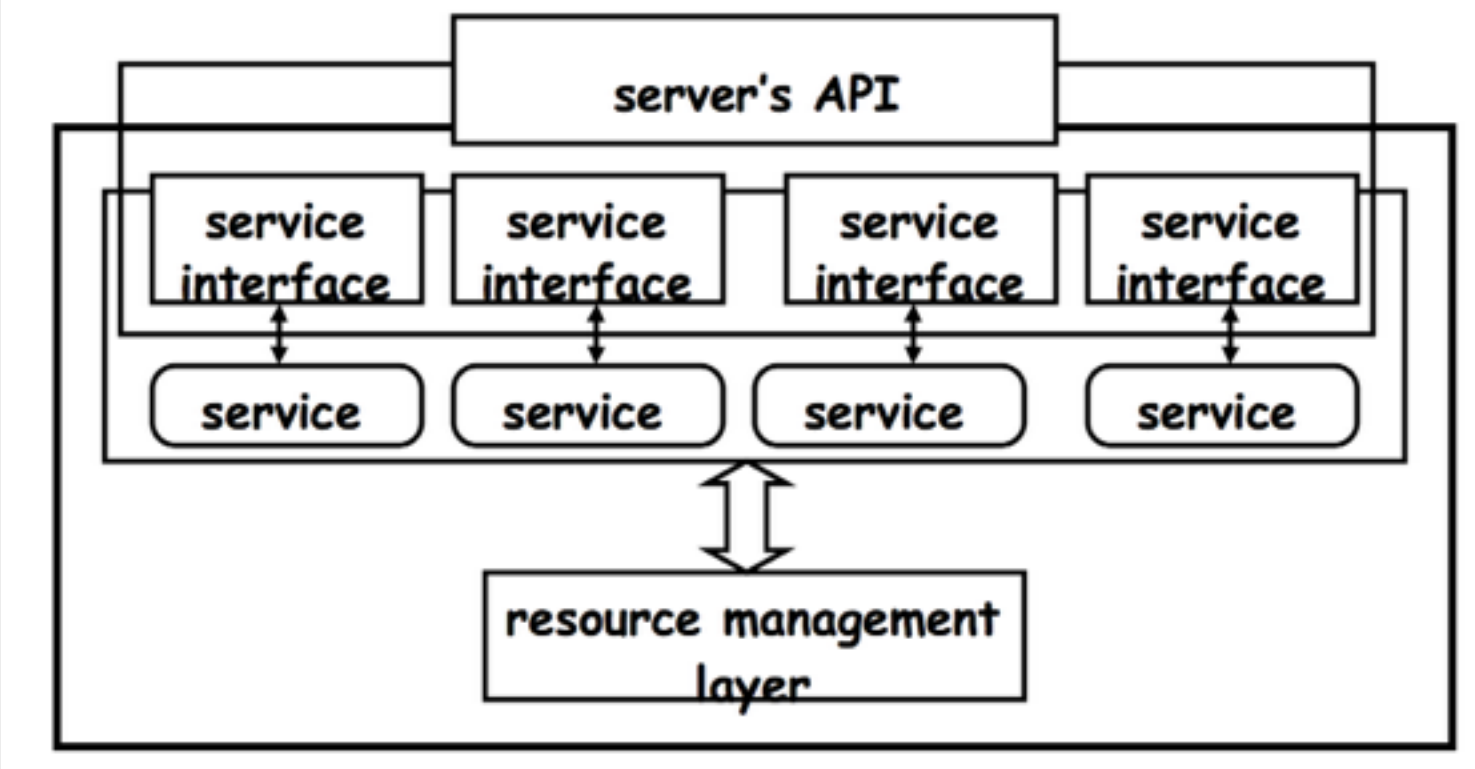
2-Tier Architecture

- Added flexibility in presentation layer
  - e.g. multiple specialised presentation layers add no complexity to application
- Encouraged stable, published APIs
  - So clients could be developed



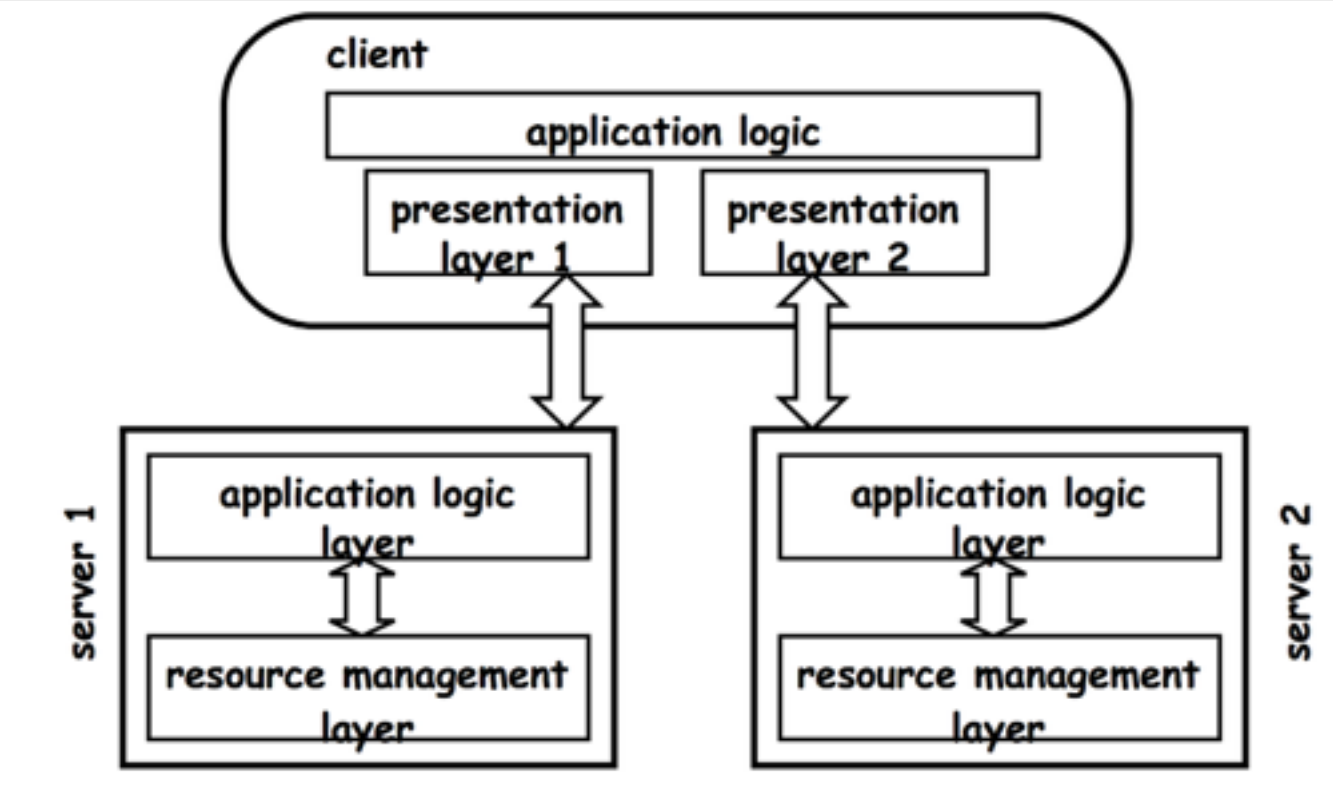
2-Tier Architecture disadvantages

- A single server doesn't scale



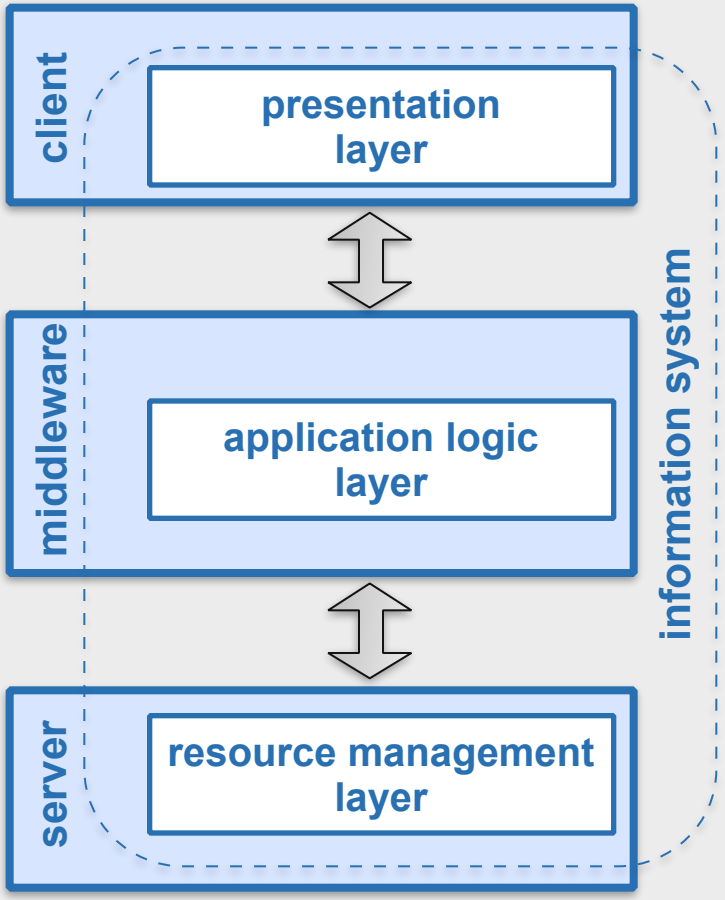
2-Tier Architecture disadvantages

- Integration of multiple services must be done at client

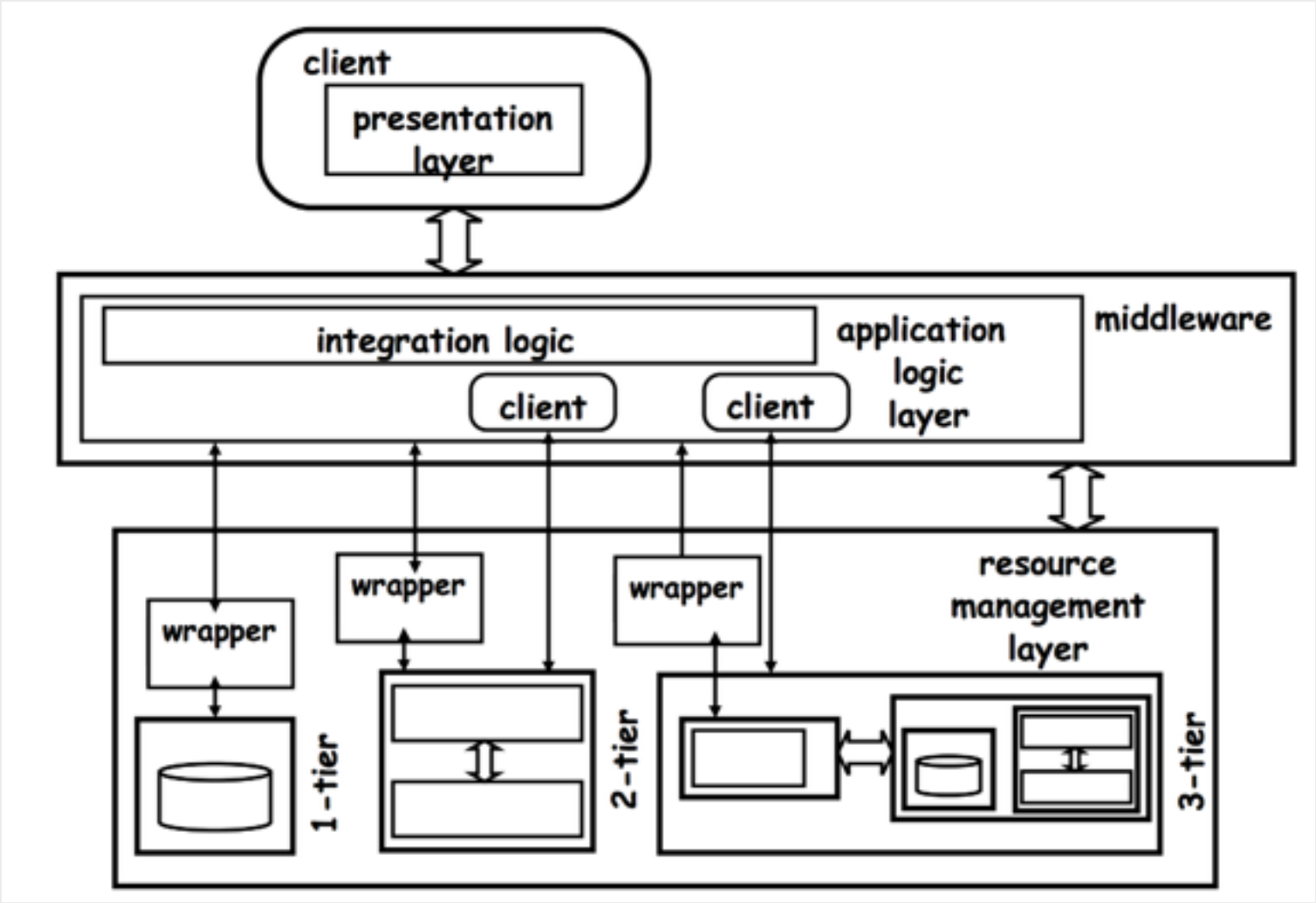


3-Tier Architecture

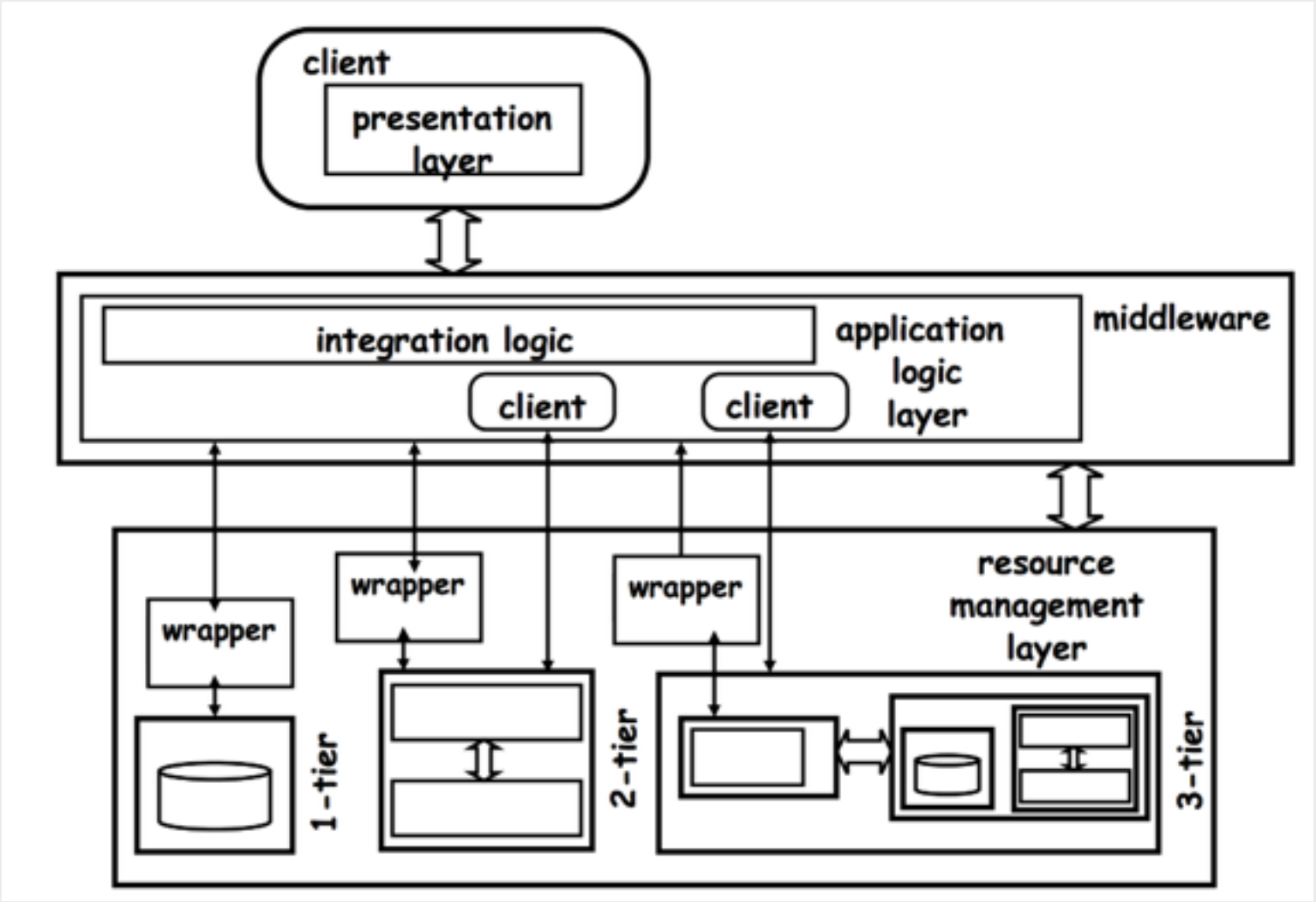
- Scalability at application layer
  - Multiple application servers
- Application Integration
  - Do it in the middle tier
- Encourage stable, published APIs for resource management



Integration in middle Tier



N-Tier architecture (Inductivity)



3-Tier 4-Tier 5-Tier

Enterprise application development considerations

- Distributed computing protocols (RMI, CORBA, IIOP)
- Load balancing
- Persistence, back-end integration
- Transaction processing
- Clustering
- Runtime re-deployment, Server restarting
- Multi-threading
- Resource pooling
- Security, performance, optimization
- …..

Java Editions

- Java Platform Micro Edition:
  - Mobile devices, set-top boxes etc
  - Restricted form of Java
- Java Platform Standard Edition:
  - Core libraries, what most people use in standard Java programming
- Java Platform Enterprise Edition:
  - Complete server-side enterprise-class development and deployment platform

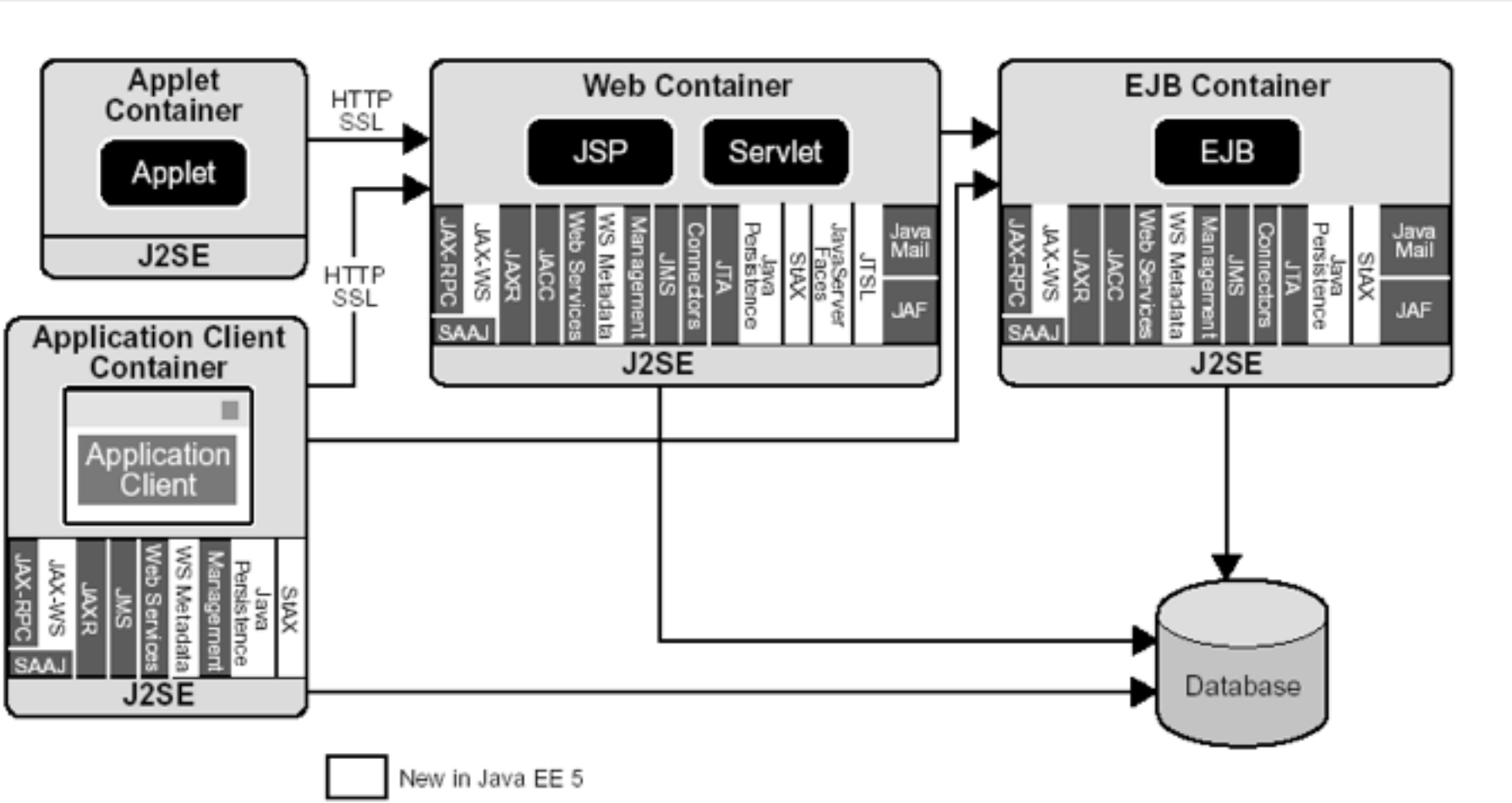
JEE

- Stands for “Java, Enterprise Edition”
- It is a collection of standards
  - JDBC, JNDI, JMX, JMS
- It is a component technology
  - Enterprise JavaBeans
- It is an “application server”
  - Following in the footsteps of Component Transaction Monitors

JEE Containers

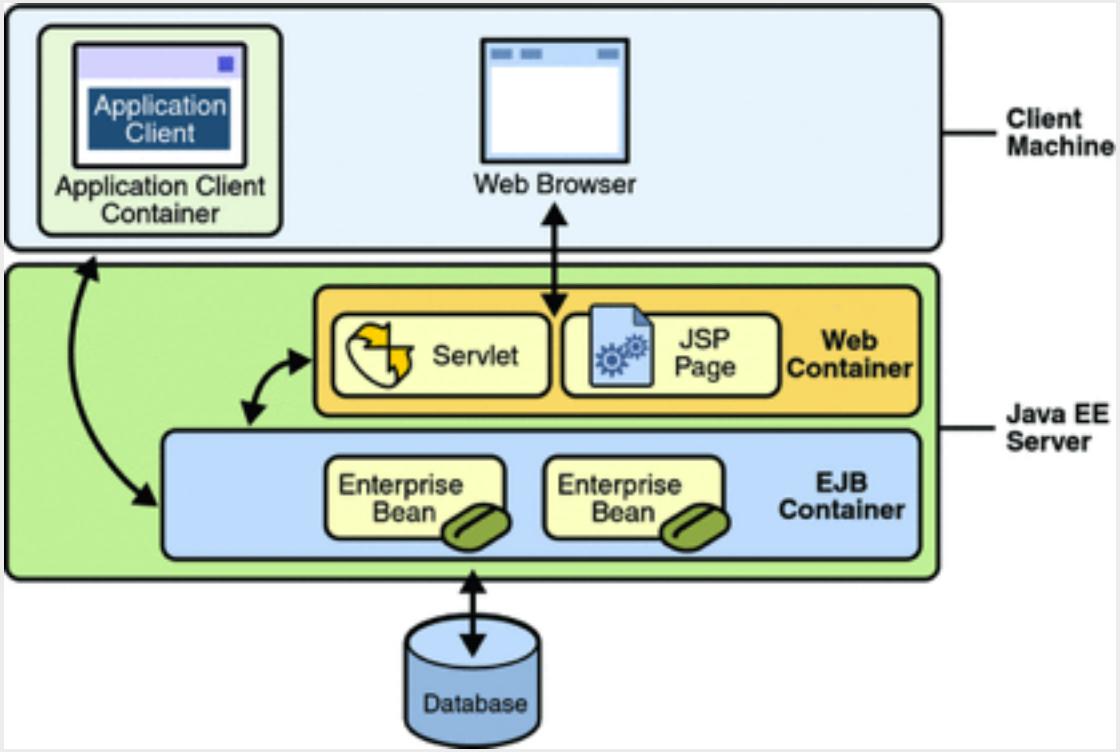
- JEE Infrastructure is divided into logical domains called containers which host components.
- A container supports services related to security, transaction management, Java Naming and Directory Interface (JNDI) lookups, and remote connectivity
- The container also manages some of the connectors because it is responsible for triggering events and instantiating components
- The container also manages non-configurable services such as enterprise bean and servlet life cycles, database connection resource pooling, data persistence, and access to the Java EE platform APIs

JEE Containers



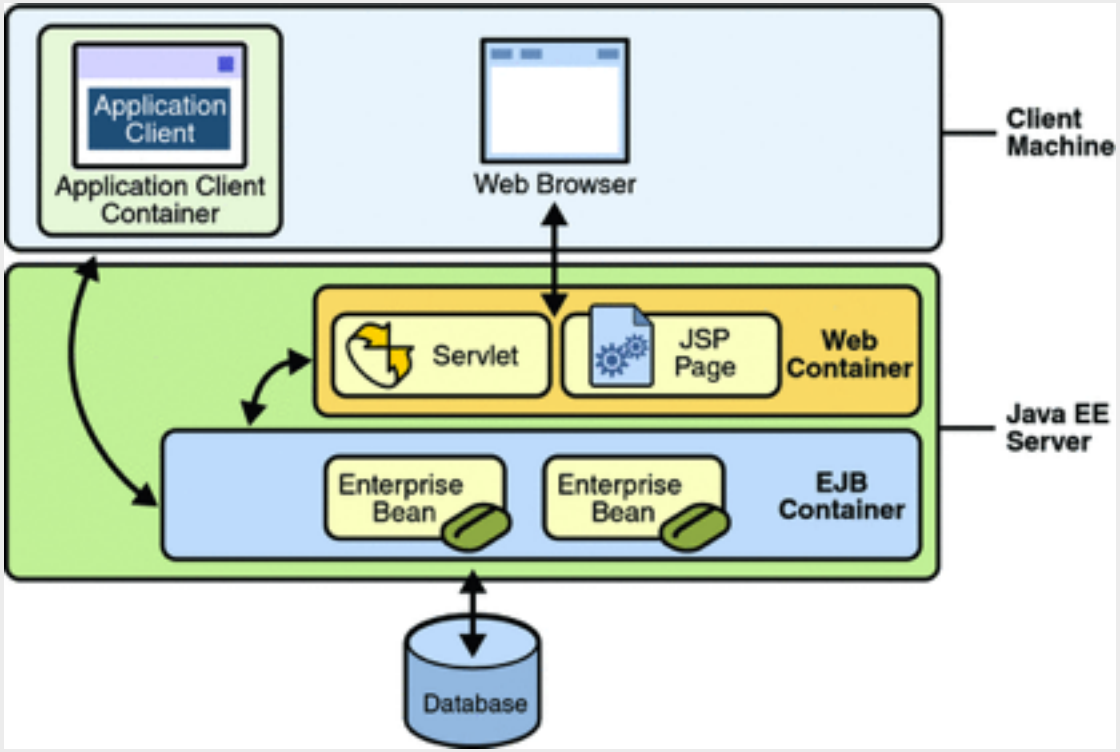
Java EE Server

- The runtime portion of a Java EE product. A Java EE server provides EJB and web containers.



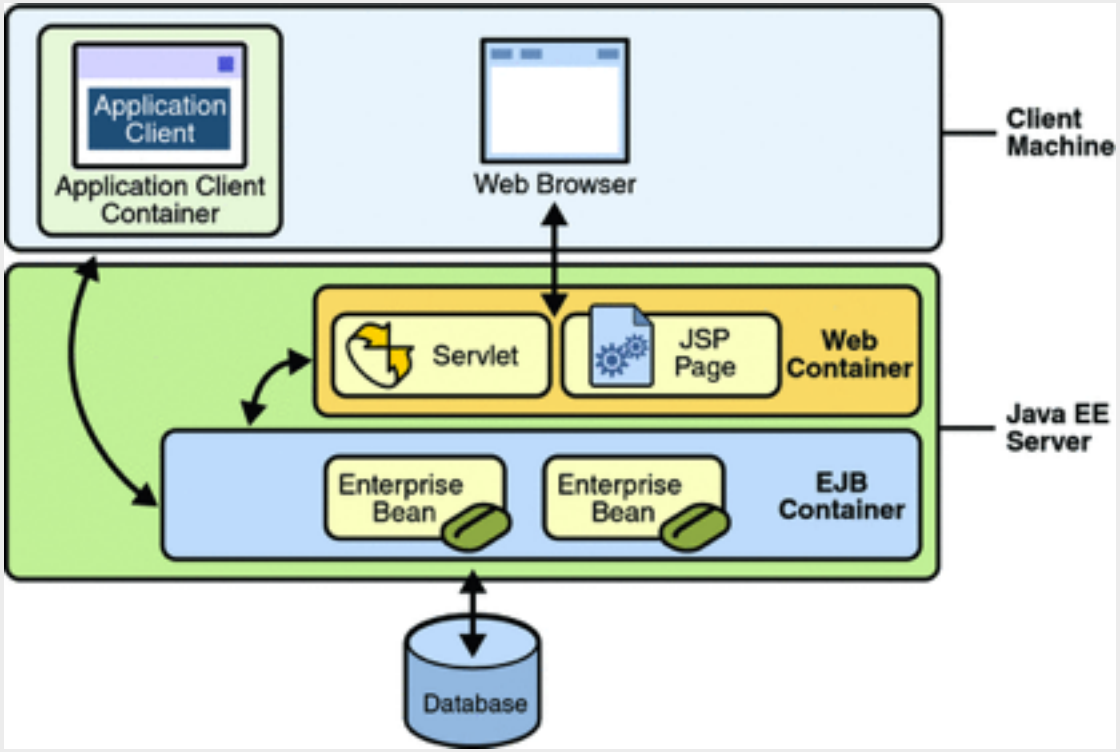
Enterprise JavaBeans (EJB) Container

- Manages the execution of enterprise beans for Java EE applications. Enterprise beans and their container run on the Java EE server.



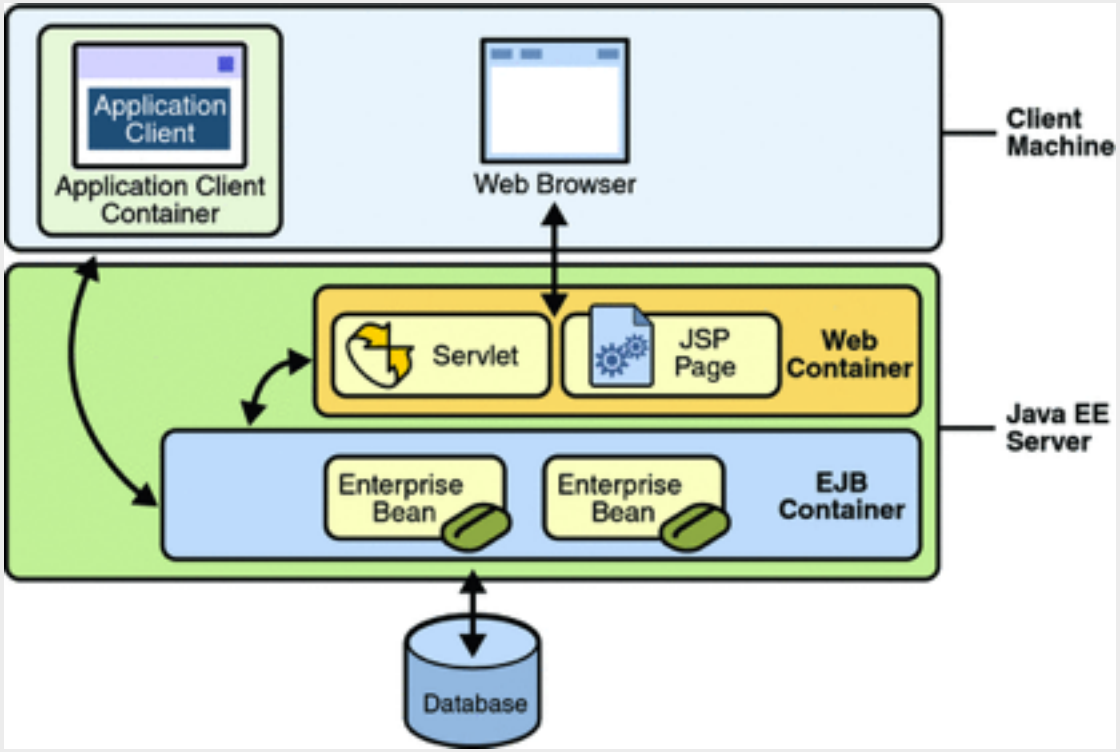
Web Container

- Manages the execution of JSP page and servlet components for Java EE applications. Web components and their container run on the Java EE server.



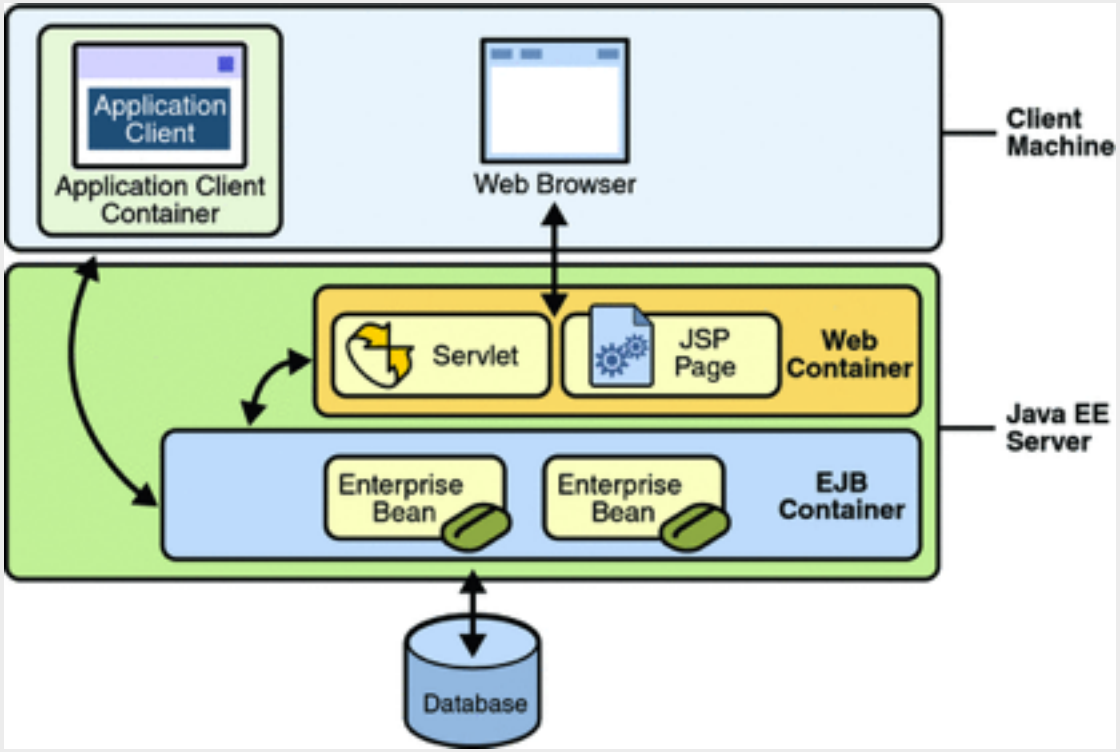
Application Client Container

- Manages the execution of application client components. Application clients and their container run on the client.



Applet Container

- Manages the execution of applets. Consists of a web browser and Java Plug-in running on the client together.



JEE Architecture

- JEE specifications are interested in the activities:
  - Development
  - Deployment
  - Execution

- Components to develop the code of the different elements of an application
  - Web components
  - Business logic components
- Containers to host the different components of an application
  - Web container
  - Application client container
- The supporting services for cross functional aspects
  - Security, transactions, ...
  - Communications infrastructure

JEE APIs

- An application programming interface (API) specifies a software component in terms of its operations, their inputs and outputs and underlying types.
- Its main purpose is to define a set of functionalities that are independent of their respective implementation

JEE APIs

- JEE APIs can be divided into two broad categories:
  - Components
    - Business logic components
    - Presentation logic components
  - Services
    - Infrastructure services
    - Communication services

JEE APIs - Components

- Business logic components:
  - Enterprise JavaBeans
- Presentation logic components
  - Servlets
  - JSP
- These components are:
  - Configured via Deployment Descriptors
  - Deployed into containers

JEE APIs - Infrastructure Services

- JDBC (Java DataBase Connectivity) is an API for accessing relational databases.
- JNDI (Java Naming and Directory Interface) is an API to access naming services and business directories such as DNS, NIS, LDAP, etc.
- JTA / JTS (Java Transaction API / Java Transaction Services) is an API that defines standard interfaces with a transaction manager.
- JCA (Java EE Connector Architecture) is an API to connect to the enterprise information system (and Legacy systems)
- JMX (Java Management Extension) provides tools for managing and monitoring applications, system objects, devices (e.g. printers) and service oriented networks.

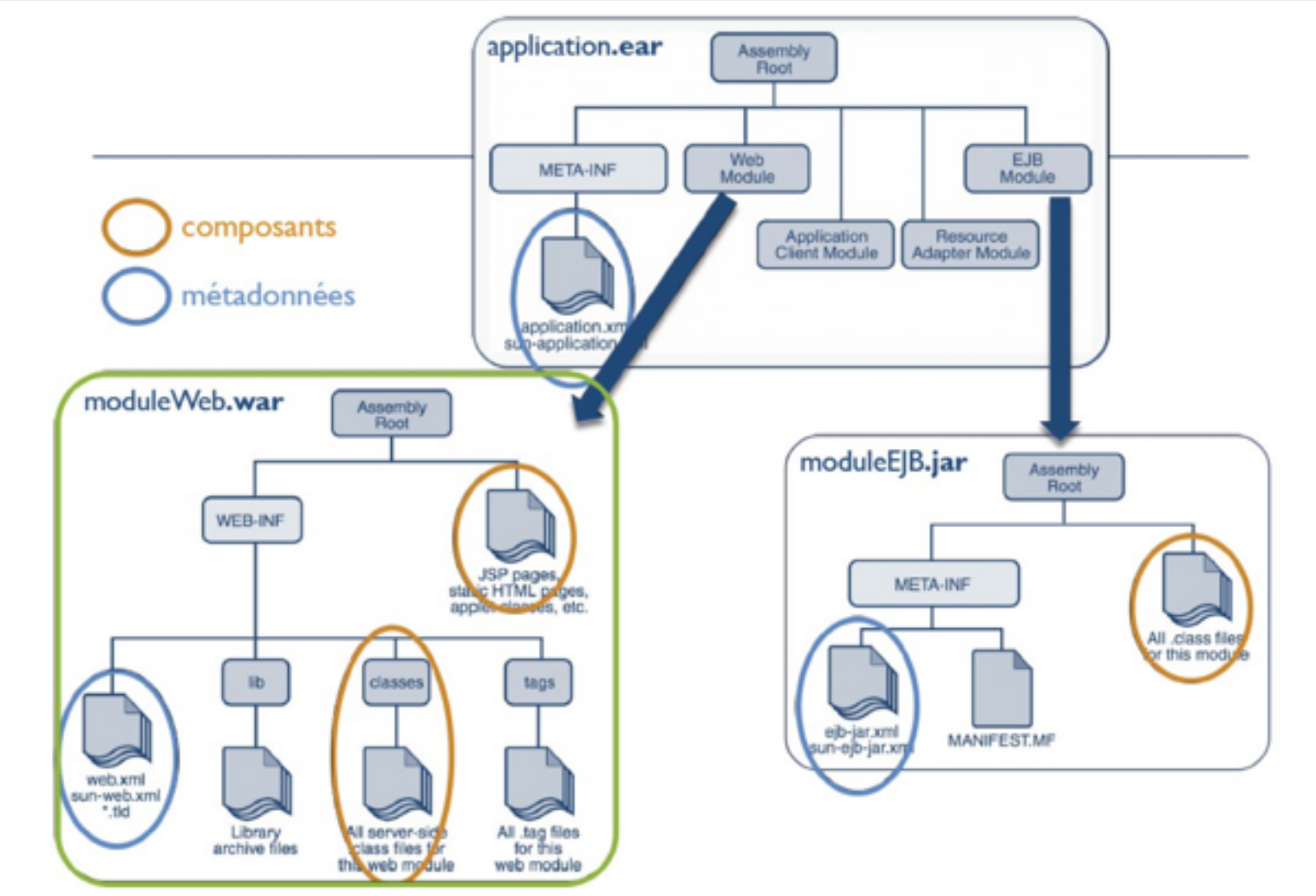
JEE APIs - Communication Services

- JAAS (Java Authentication and Authorization Service) is an API for the managing authentication and access rights.
- JavaMail is an API for sending email.
- JMS (Java Message Service) provides asynchronous communication capabilities (called MOM Middleware Message Object) between applications.
- RMI-IIOP (Remote Method Invocation over Internet Inter-ORB Protocol) is an API that allows synchronous communication between objects.

JEE Packaging

- Client application
  - jar archive
- Web
  - Gathers servlets and JSP and the resources required to execute them (classes, tag libraries, images, ...)
  - war archive + web.xml (optional in some cases)
- EJB
  - Gathers EJBs and their components (classes)
  - jar archive + ejb-jar.xml
- Enterprise Application
  - ear archive (includes several jar and war modules) + application.xml

JEE Packaging



Types of clients

- Fat client (Thick client) means a graphical client application running on the operating system of the user. A thick client generally has advanced processing capabilities and can have a sophisticated graphical interface.
- Thin client refers to an application accessible via a web interface (HTML) can be viewed using a web browser, where all the business logic is processed on the server side.
- Rich client (smart client) provide a graphical interface, described with a grammar description based on XML syntax, that allows a user's local applications to interact with server-based applications through the use of Web services.

Application Servers

- An application server is a server-side application execution environment
- It supports all the features that allow multiple clients to use the same application
- Application servers can provide:
  - Only a web container (eg Tomcat)
  - Only an EJB container (eg JBoss, Jonas, ...)
  - Both these containers (eg Websphere, Weblogic, …)