Enterprise Computing: An Overview

B. Ramamurthy

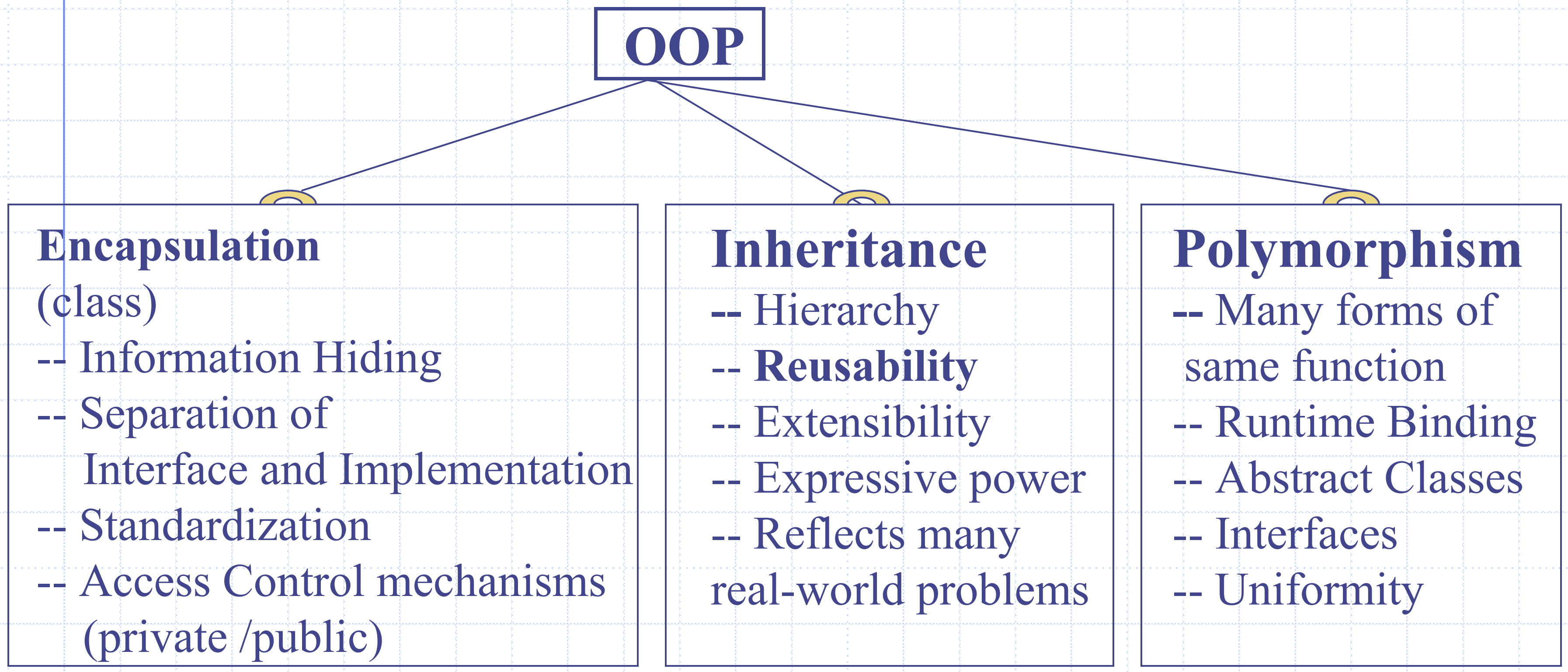
Introduction

- In this lecture we will trace through all the important developments leading to enterprise computing.
- During this process I will review many fundamental concepts such as object-oriented principles and request-reply model, distributed objects, remote method invocations, Java technology etc.
- Your task is to identify the concepts that you further need to study and work on them in the next two weeks.
- Those who are familiar with any of the concepts, share your experiences with the students in the class.

Topics of Discussion

- Object-Orientation (OO) Principles
- Unified Modeling Language (UML)
- Beyond objects
- Enterprise systems
- Middleware
- CORBA
- J2EE
- RMI
- Computation grid

Object-Oriented Principles (OOP)



OOP

Encapsulation
(class)
-- Information Hiding
-- Separation of
Interface and Implementation
-- Standardization
-- Access Control mechanisms
(private /public)

Inheritance
-- Hierarchy
-- **Reusability**
-- Extensibility
-- Expressive power
-- Reflects many
real-world problems

Polymorphism
-- Many forms of
same function
-- Runtime Binding
-- Abstract Classes
-- Interfaces
-- Uniformity

Why OO paradigm?

- OO Models let you structure your thoughts.
- Convenient for large software development
- Systematic approach to analyzing large problems
- Reuse through classes and inheritance
- Supports Application programmer Interface (API) concept
- Standardization (standard interface)
- Facilitates security , protection and access control

Unified Modeling Language

The Unified Modeling Language™ (UML) was developed jointly by Grady Booch, Ivar Jacobson, and Jim Rumbaugh with contributions from other leading methodologists, software vendors, and many users. The UML provides the application modeling language for:

- Business process modeling/ Requirement Analysis with use cases.
- Static Design with Class modeling and object modeling.
- Dynamic Design with sequence, collaboration and activity diagrams.
- Component modeling.
- Distribution and deployment modeling.

- See
  - **http://www.rational.com/uml/resources/whitepapers/index.jsp**
  - **http://www.cetus-links.org/oo_uml.html**

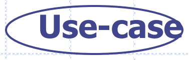
Phases of System Development

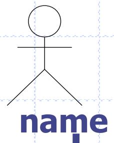
- Requirement Analysis
  - Functionality users require from the system
  - Use case model
- OO Analysis
  - Discovering classes and relationships
  - UML class diagram
- OO Design
  - Result of Analysis expanded into technical solution
  - Sequence diagram, state diagram, etc.
  - Results in detailed specs for the coding phase
- Implementation (Programming/coding)
  - Models are converted into code
- Testing
  - Unit tests, integration tests, system tests and acceptance tests.

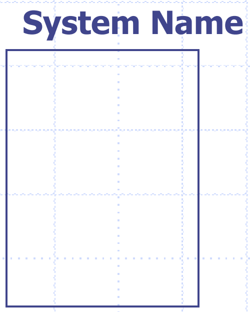
Use-Case Modeling

- In use-case modeling, the system is looked upon as a black box whose boundaries are defined by its functionality to external stimulus.
- The actual description of the use-case is usually given in plain text. A popular notation promoted by UML is the stick figure notation.
- We will look into the details of text representation later. Both visual and text representation are needed for a complete view.
- A use-case model represents the use-case view of the system. A use-case view of a system may consist of many use case diagrams.
- An use-case diagram shows (the system), the actors, the use-cases and the relationship among them.

Components of Use Case Model

- The components of a use case model are:
  - Use cases 

  - Actors 

  - System Modeled 

  - Stimulus

System

- As a part of the use-case modeling, the boundaries of the system are developed.
- System in the use-case diagram is a box with the name appearing on the top.
- Defining a system is an attempt to define the catalog of terms and definitions at an early stage of the development of a business model.

Actors

- An actor is something or someone that interacts with the system.
- Actor communicates with the system by sending and receiving messages.
- An actor provides the stimulus to activate an use case.
- Message sent by an actor may result in more messages to actors and to use cases.
- Actors can be ranked: primary and secondary; passive and active.
- Actor is a role not an individual instance.

Finding Actors

- The actors of a system can be identified by answering a number of questions:
  - Who will use the functionality of the system?
  - Who will maintain the system?
  - What devices does the system need to handle?
  - What other system does this system need to interact?
  - Who or what has interest in the results of this system?

Use Cases

- A use case in UML is defined as a set of sequences of actions a system performs that yield an observable result of value to a particular actor.
- Actions can involve communicating with number of actors as well as performing calculations and work inside the system.
- A use case
  - is always initiated by an actor.
  - provides a value to an actor.
  - must always be connected to at least one actor.
  - must be a complete description.
- Example?

Finding Use Cases

- For each actor ask these questions:
  - Which functions does the actor require from the system?
  - What does the actor need to do?
  - Could the actor's work be simplified or made efficient by new functions in the system?
  - What events are needed in the system?
  - What are the problems with the existing systems?
  - What are the inputs and outputs of the system?

Classes

- OO paradigm supports the view that a system is made up of objects interacting by message passing.
- Classes represent collection of objects of the same type.
- An object is an instance of a class.
- A class is defined by its properties and its behaviors.
- A class diagram describes the static view of a system in terms of classes and relationships among the classes.

Discovering Classes

- Underline the nouns in a problem statement.
- Using the problem context and general knowledge about the problem domain decide on the important nouns.
- Design and implement classes to represent the nouns.
- Underline the verbs. Verbs related to a class may represent the behavior of the class.
- You can also discover the classes from the use case diagram.

Examples

- Drawing package: Design a user interface for drawing various shapes: circle, square, rectangle.
- Football scores: Keep track of football score.
- General purpose counter: To keep of track of count for various applications.
- Library: Books, different categories of books, details of student borrower, library **personnel**.

Designing Classes

- A class represents a class of objects.
- A class contains the data declarations (“parts”) and methods (“behaviors” or “capabilities”).

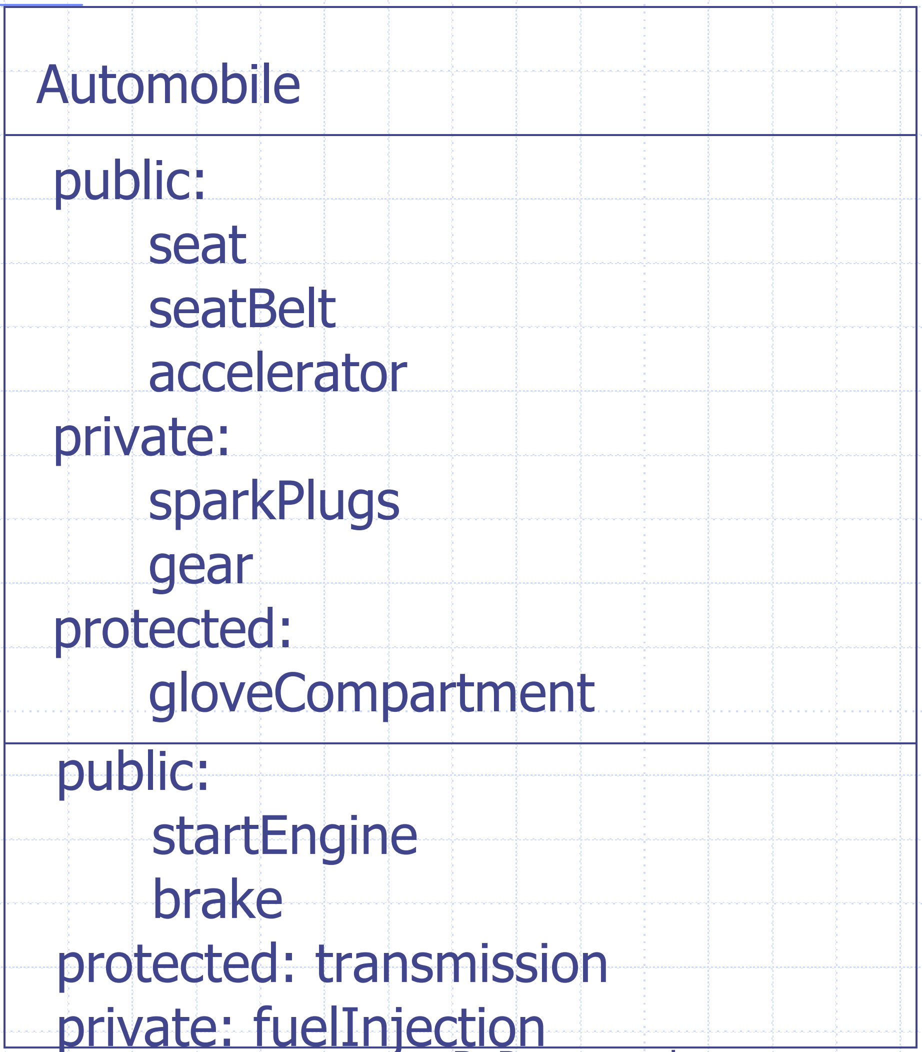
OO Design:

- Class properties or characteristics are answers to “What is it made of?” (It **has a** ____, ____, etc.)
- Behaviors, capabilities or operations are answers to “What **can** it **do**?” (verbs in the problem)

Classes are Blueprints

- A class defines the general nature of a collection of objects of the same type.
- The process creating an object from a class is called instantiation.
- Every object is an instance of a particular class.
- There can be many instances of objects from the same class possible with different values for data.
- A class structure implements encapsulation as well as access control: private, public, protected.

Class Diagram : Automobile



Automobile

public:
- seat
- seatBelt
- accelerator

private:
- sparkPlugs
- gear

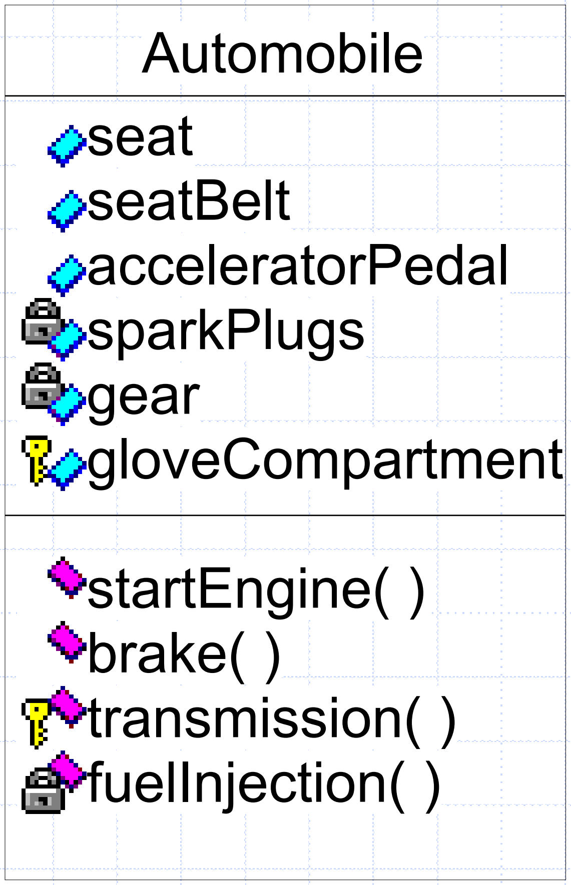
protected:
- gloveCompartment

public:
- startEngine
- brake

protected: transmission

private: fuelInjection

Automobile Class Using Rational Rose Tool



On to implementation

- You may define the methods of the class using sequence diagram and state diagram.
- Using these diagrams you can code the application.

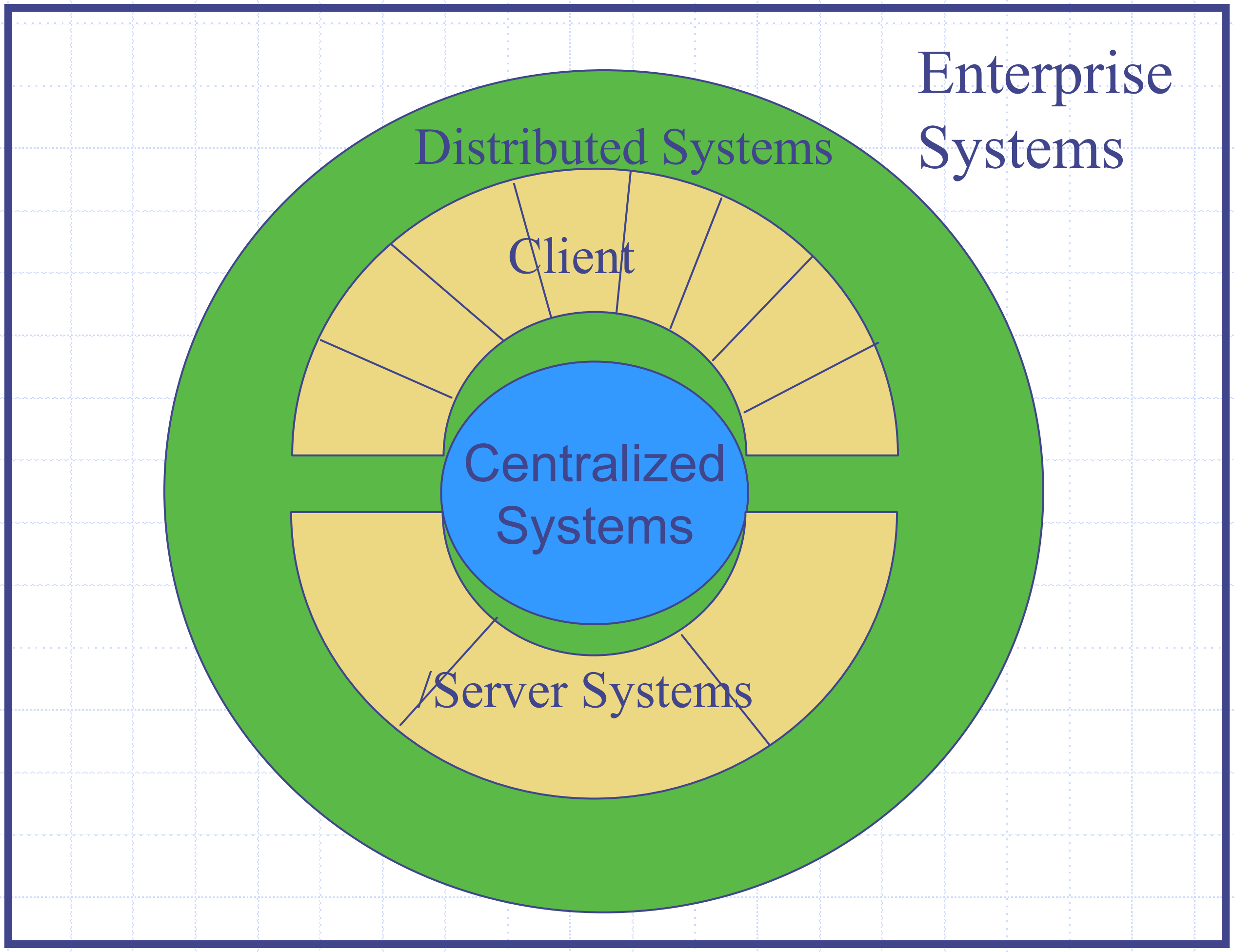
Beyond Objects

- Issues: Basic object-technology could not fulfill the promises such as reusability and interoperability fully in the context internet and enterprise level applications. Deployment was still a major problem and as a result portability and mobility are impaired.
- Solution: Middleware
- Common Object Request Broker Architecture (CORBA), Java 2 Enterprise Edition, Web services, .NET, computation grid

Enterprise Systems

- An enterprise is a very large organization.
- An enterprise system is a distributed system involving many large organizations.
- An example: AT&T, inktomi, amazon.com, UPS, and users operating in a supply chain model, make up an enterprise system.
- Inter .com ….

Evolution of Computing Systems



Distributed System as an Enterprise System

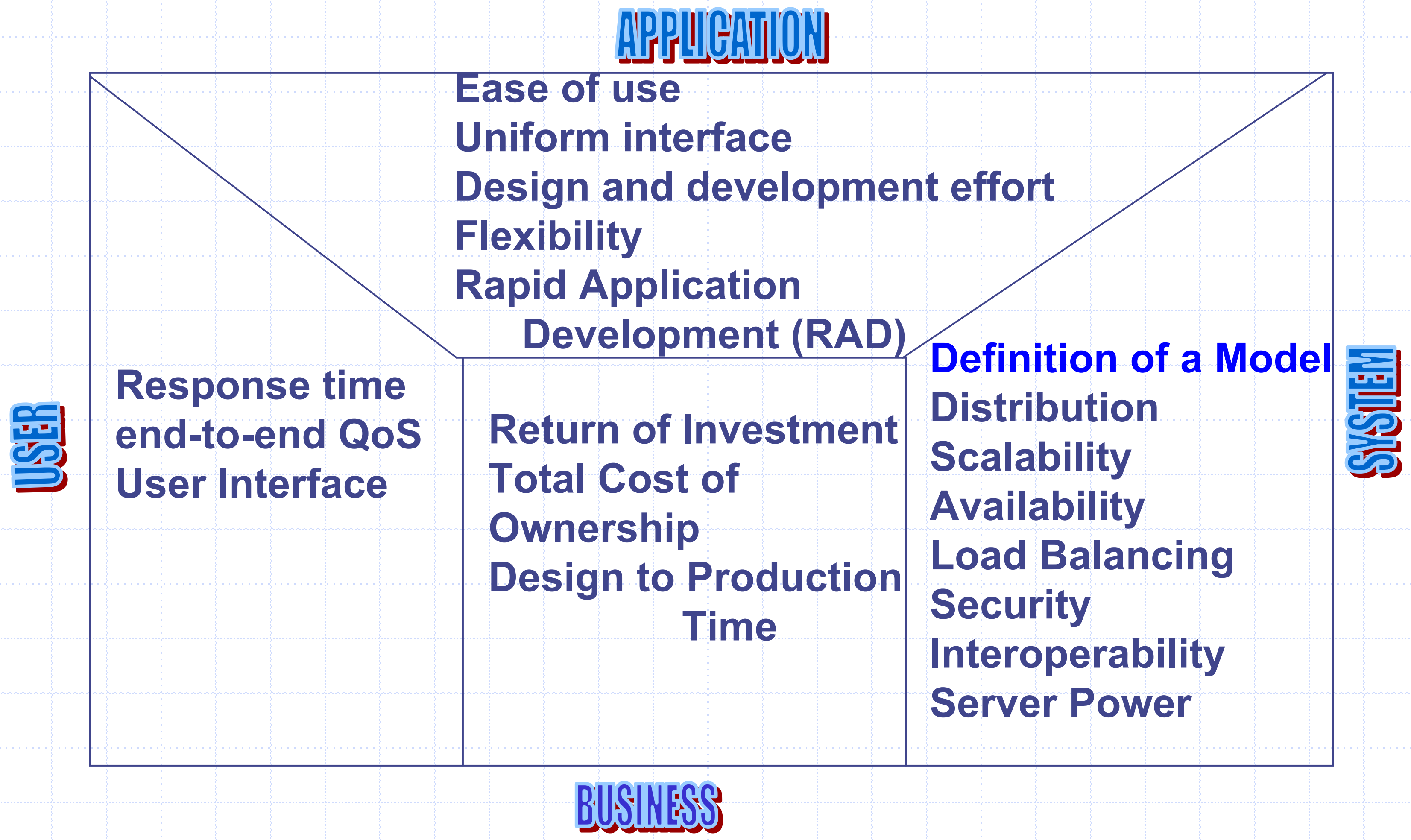
- **There are many problems in using traditional distributed system model for enterprise computing. Look at**

"[A Note on Distributing Computing]" by Jim Waldo, Geoff Wyant, Ann Wollarth and Sam Kendall of Sun labs.

-- current distributed system paradigm works well for small systems dealing with single address space but fails very badly for dynamically changing global address spaces.

- **We have seen advances in code mobility, data mobility,etc. But the distributed system architecture/principles are yet to evolve in any significant way.**
- **Focus on distribution.**

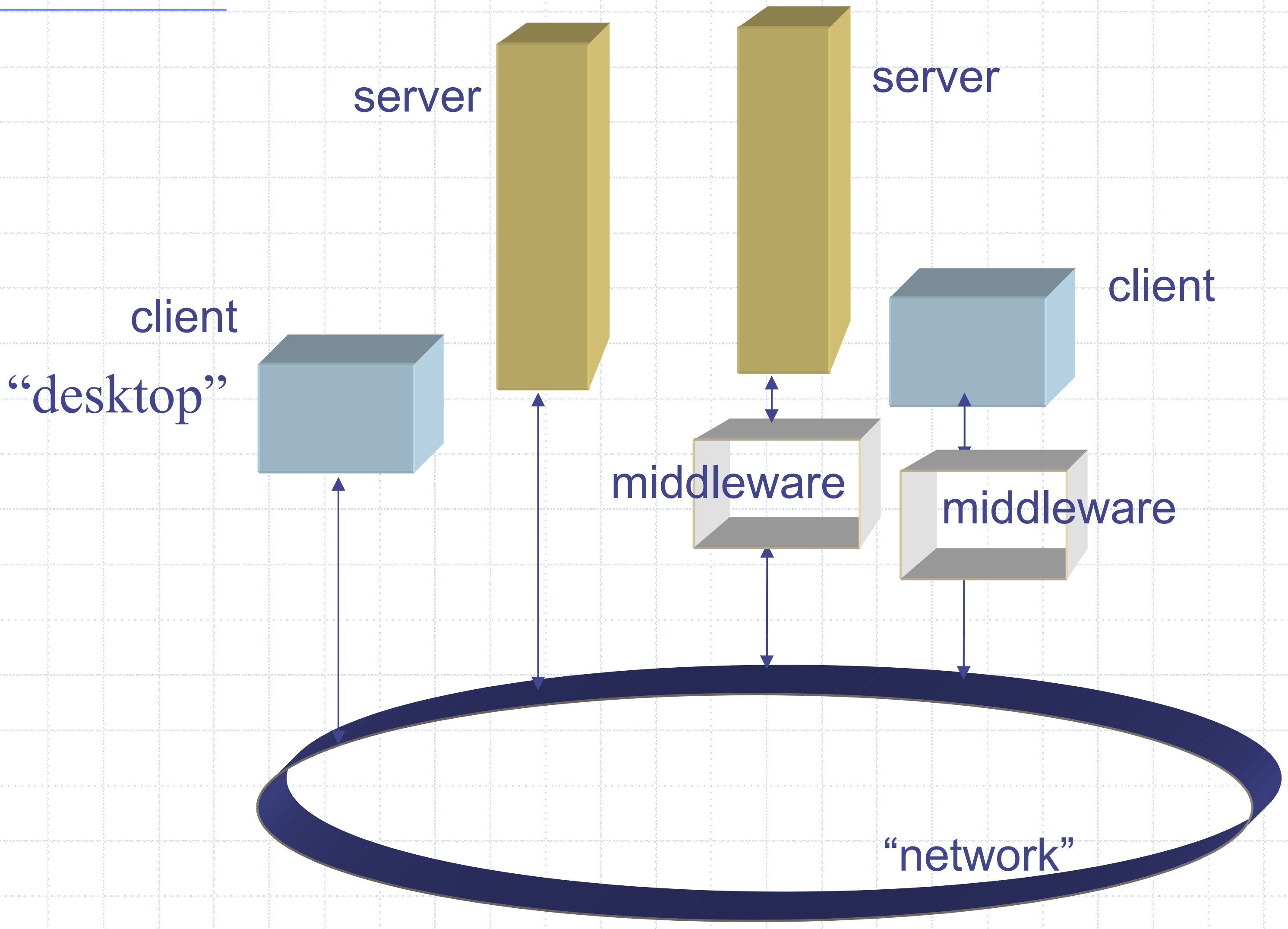
Issues in Enterprise Systems



Requirements for Enterprise Computing

- Accommodate changes gracefully - **scalability**, **dynamic reconfiguration**
- Maintain high **availability** at all times
- Offer good performance in terms of response time and end-to-end "QOS"
- Dependability and fault tolerance
- Simplicity
- ….

Enabling Technology



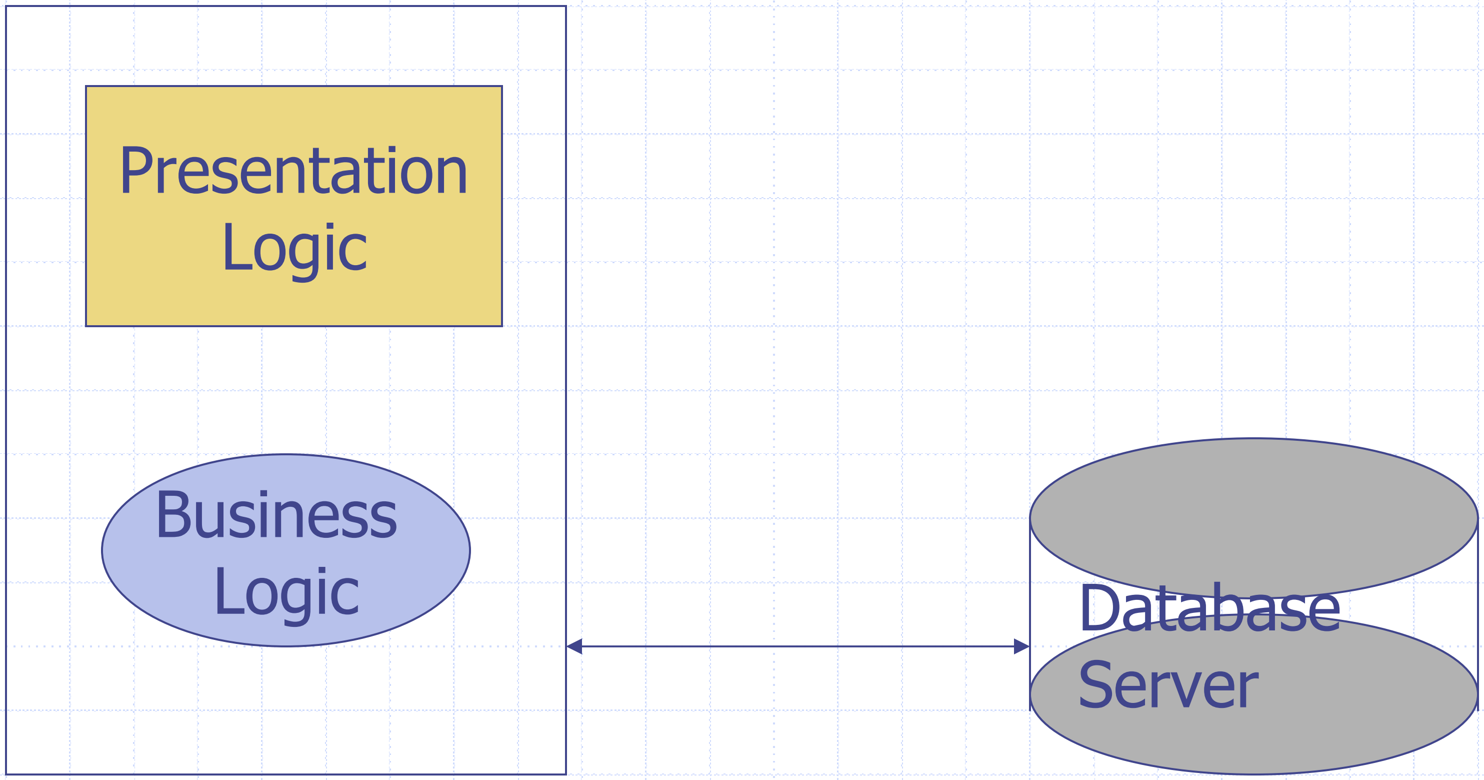
Middleware (as defined by NSF)

- Middleware refers to the software which is common to multiple applications and builds on the network transport services to enable ready development of new applications and network services.
- Middleware typically includes a set of components such as resources and services that can be utilized by applications either individually or in various subsets.
  - Examples of services: Security, Directory and naming, end-to-end quality of service, support for mobile code.
- OMG's CORBA defines a middleware standard.
- J2EE Java 2 enterprise edition is a middleware specification.
- Compute grid is a middleware framework.

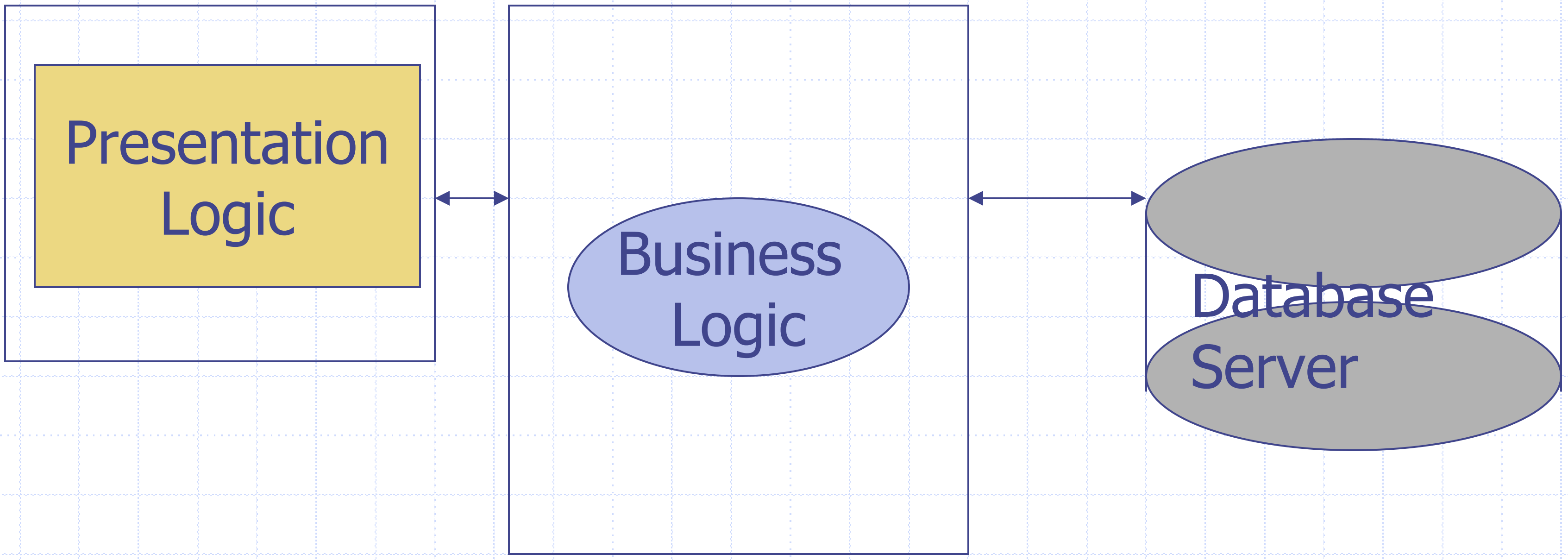
Component Technology

- We need an application architecture that works well in the new E-commerce age.
- Programmer productivity, cost-effective deployment, rapid time to market, seamless integration, application portability, scalability, security are some of the challenges that component technology tries to address head on.
- Enterprise Java Beans is Sun's server component model that provides portability across application servers, and supports complex systems features such as transactions, security, etc. on behalf of the application components.
- EJB is a specification provided by Sun and many third party vendors have products compliant with this specification: BEA systems, IONA, IBM, Oracle.

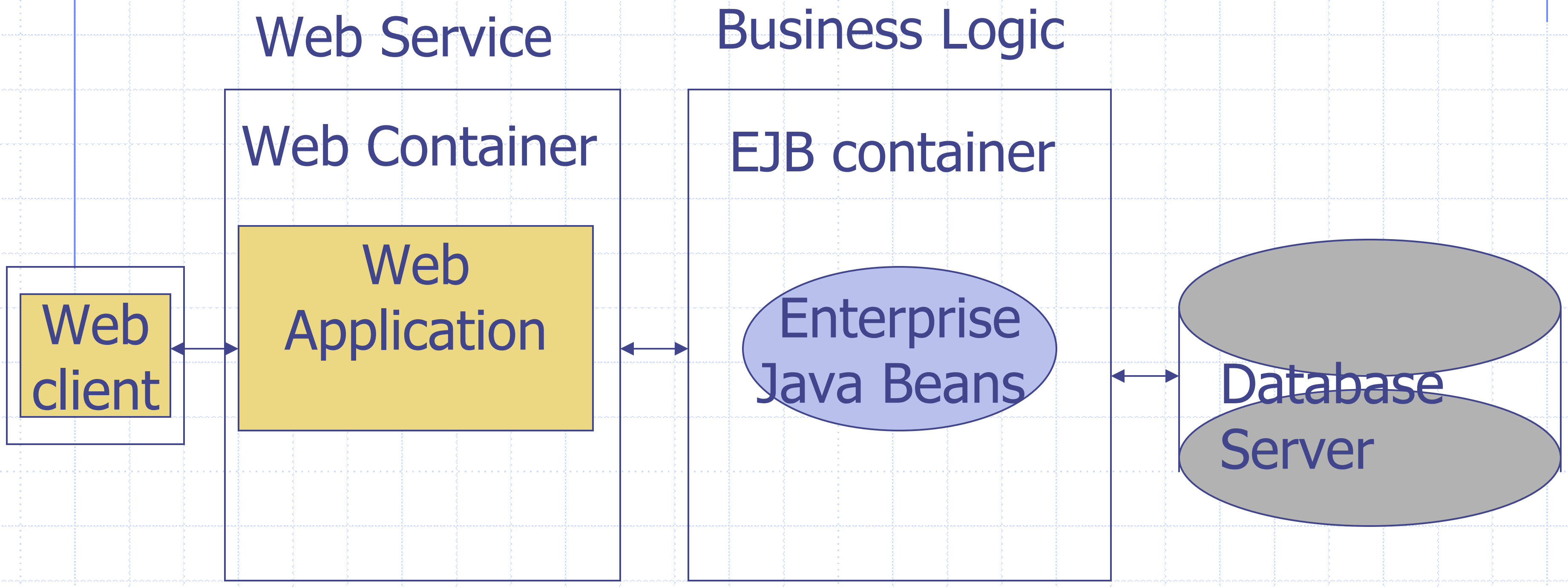
Two-tier applications



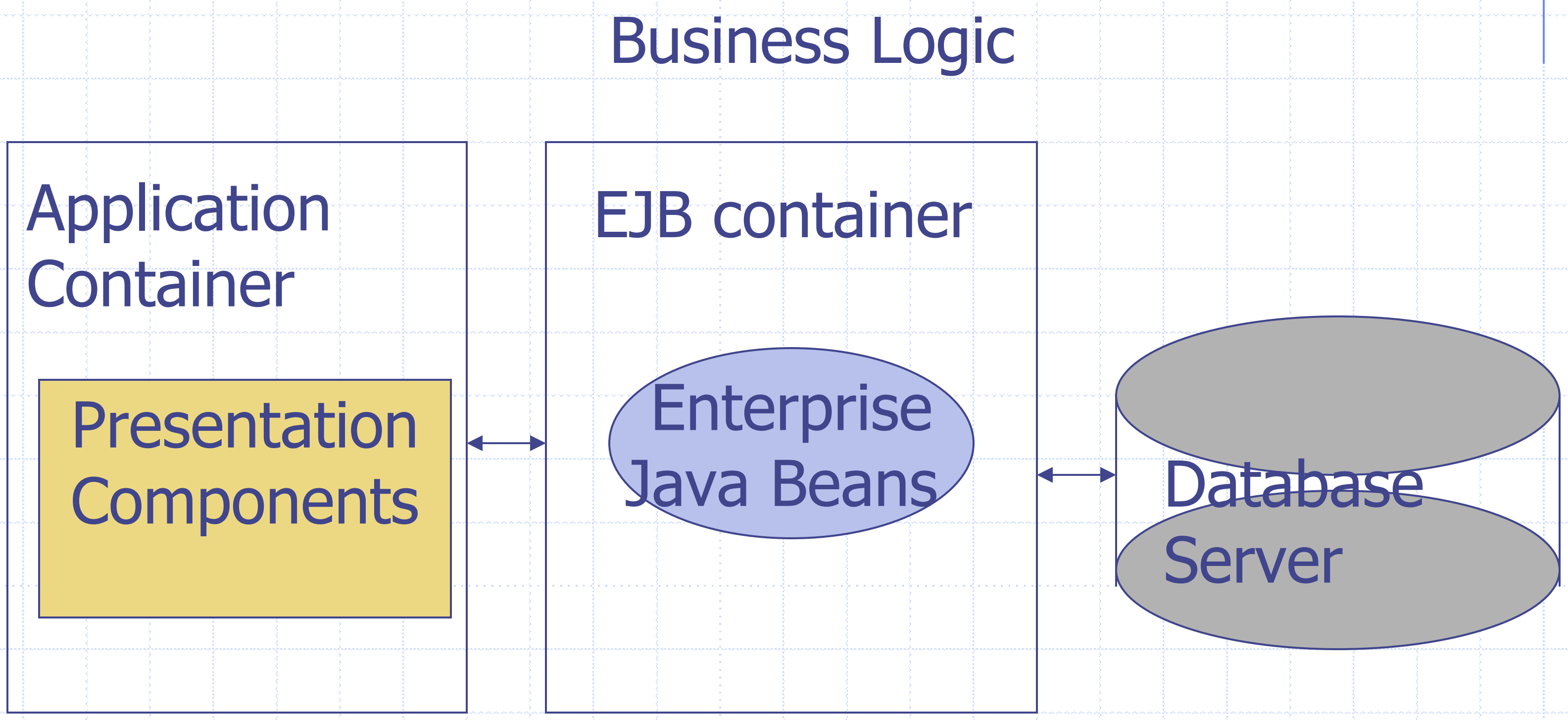
Three-tier Applications



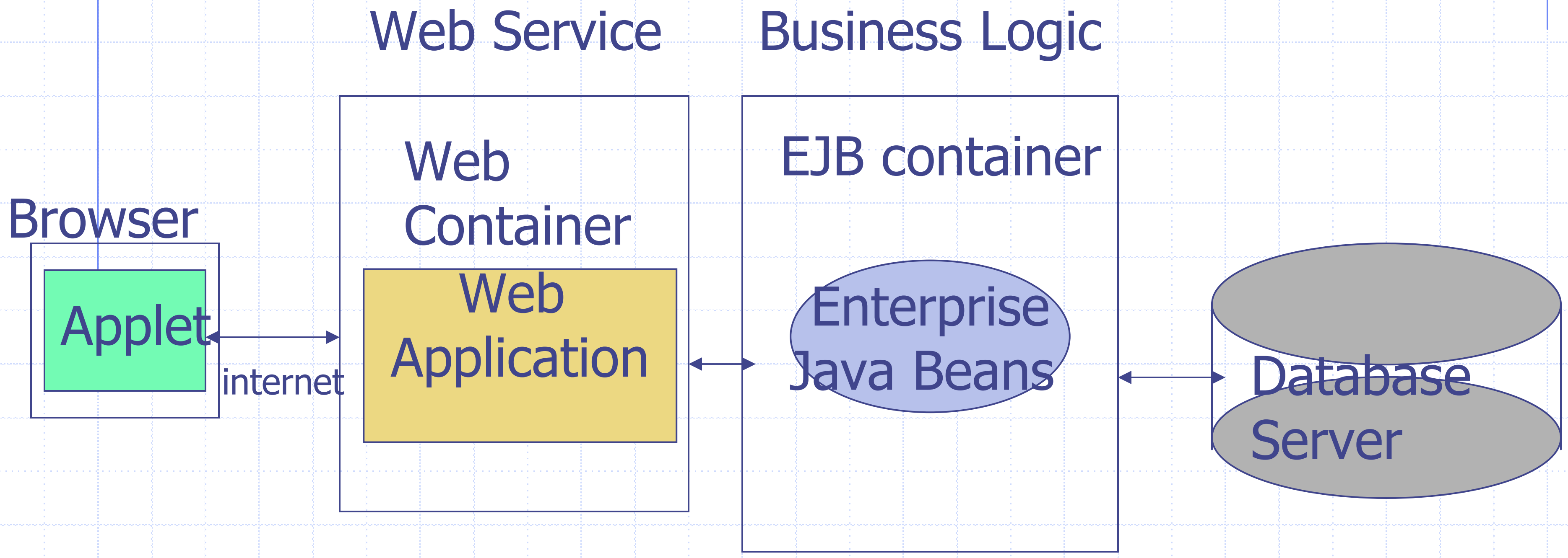
J2EE Application Programming Model for Web-based applications



J2EE Application Programming Model for Three-tier Applications



J2EE Application Programming Model for Web-based Applets


Web Service
Business Logic
Browser
Applet
internet
Web Container
Web Application
EJB container
Enterprise Java Beans
Database Server