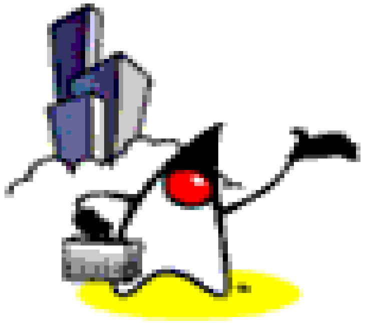
Java EE Overview

Sang Shin
www.JPassion.com
"Learn with JPassion!"

Agenda

- What is Java EE?
- Evolution of Enterprise Application Development Frameworks
- Why Java EE?
- Java EE Platform Architecture
- How to get started

What is Java EE?



Enterprise Computing

Challenges

Portability

Diverse Environments

Time-to-market

Core Competence

Assembly

Integration

Key Technologies

- J2SE™
- J2EE™
  - JMS
  - Servlet
  - JSP
  - Connector
- XML
  - Data Binding
  - XSLT

Products

App Servers

Web Servers

Components

Databases

Object to DB tools

Legacy Systems

Databases

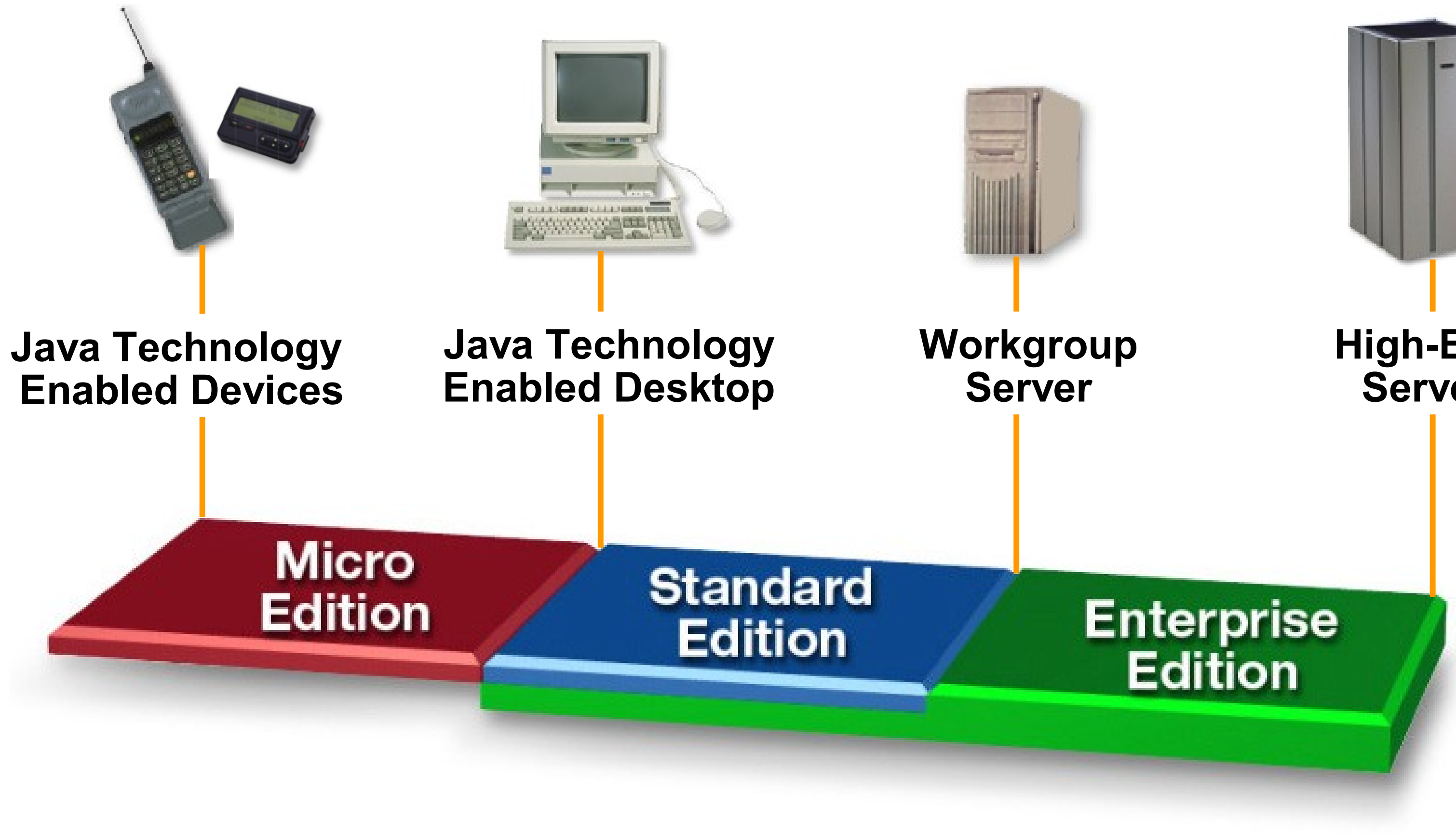
TP Monitors

EIS Systems

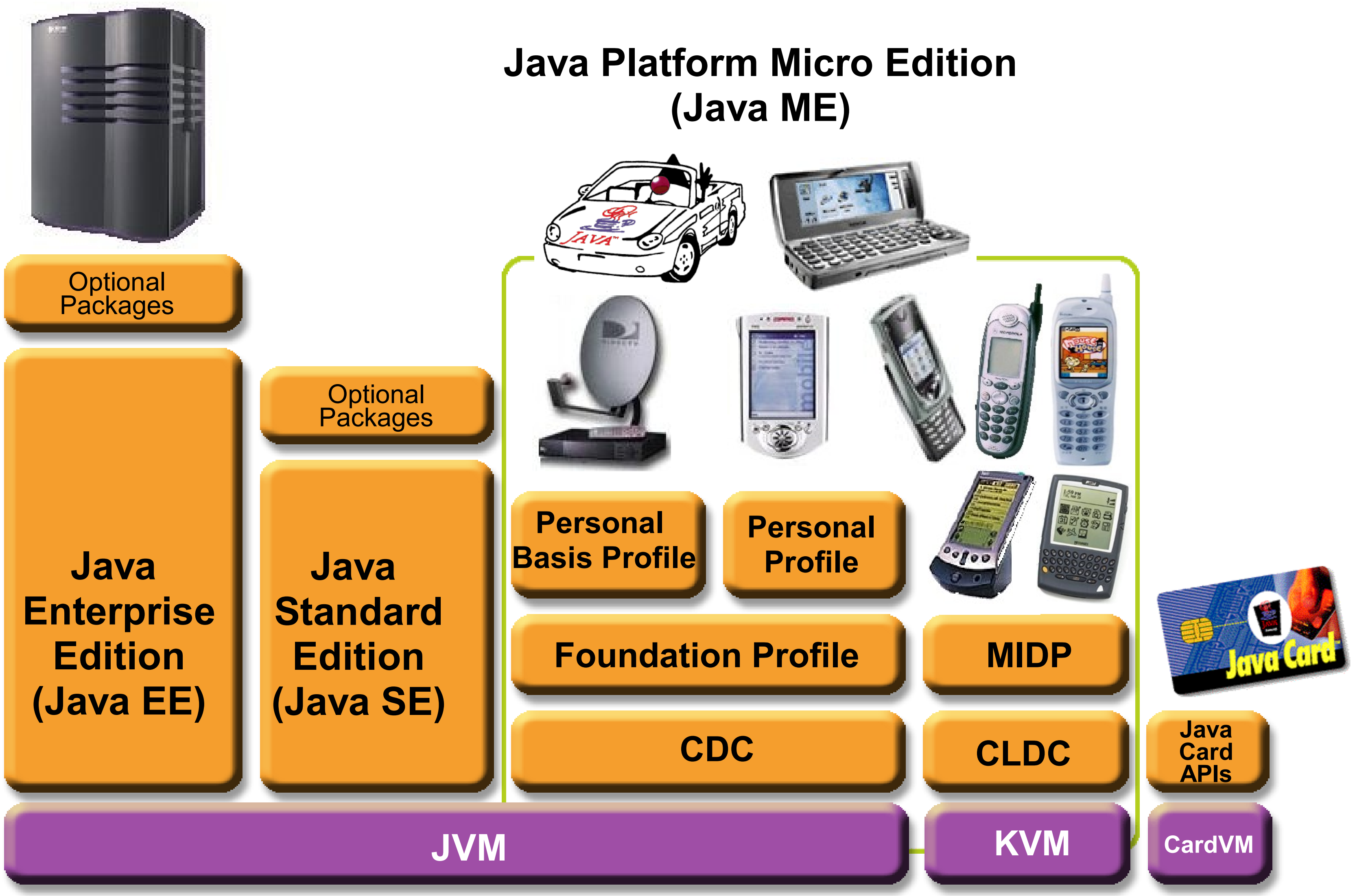
What Is the Java EE?

- Open and standard based platform for
- developing, deploying and managing
- n-tier, Web-enabled, server-centric, and component-based enterprise applications

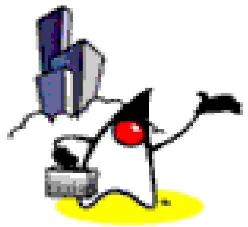
The Java™ Platform



The Java™ Platform



Evolution of Enterprise Application Frameworks



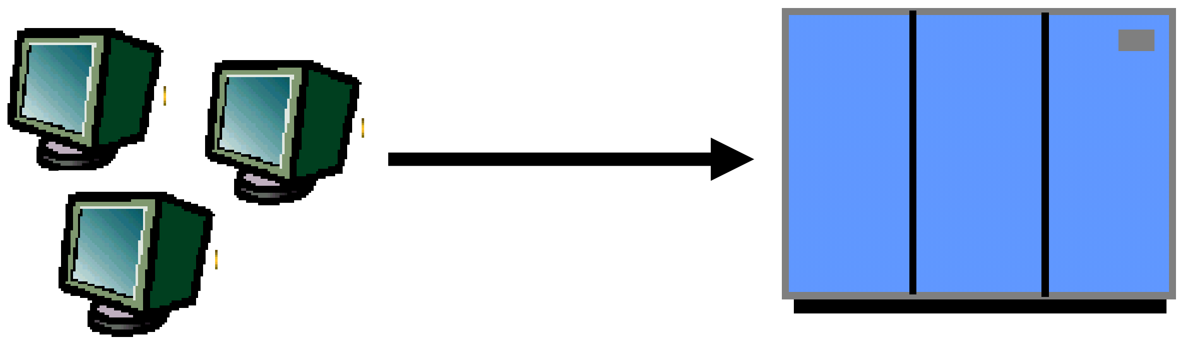
Evolution of Enterprise Application Framework

- Single tier
- Two tier
- Three tier
  - RPC based
  - Remote object based
- Three tier (HTML browser and Web server)
- Proprietary application server
- Standard application server

About Enterprise Applications

- Things that make up an enterprise application
  - Presentation logic
  - Business logic
  - Data access logic (and data model)
  - System services
- The evolution of enterprise application framework reflects
  - How flexibly you want to make changes
  - Where the system services are coming from

Single Tier (Mainframe-based)

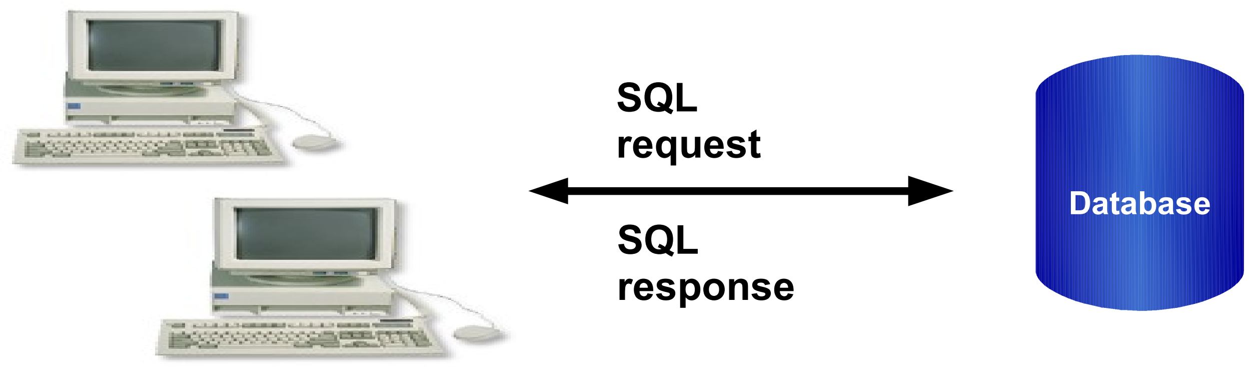


- Dumb terminals are directly connected to mainframe
- Centralized model (as opposed distributed model)
- Presentation, business logic, and data access are intertwined in one monolithic mainframe application

Single-Tier: Pros & Cons

- Pros:
  - No client side management is required
  - Data consistency is easy to achieve
- Cons:
  - Functionality (presentation, data model, business logic) intertwined, difficult for updates and maintenance and code reuse

Two-Tier

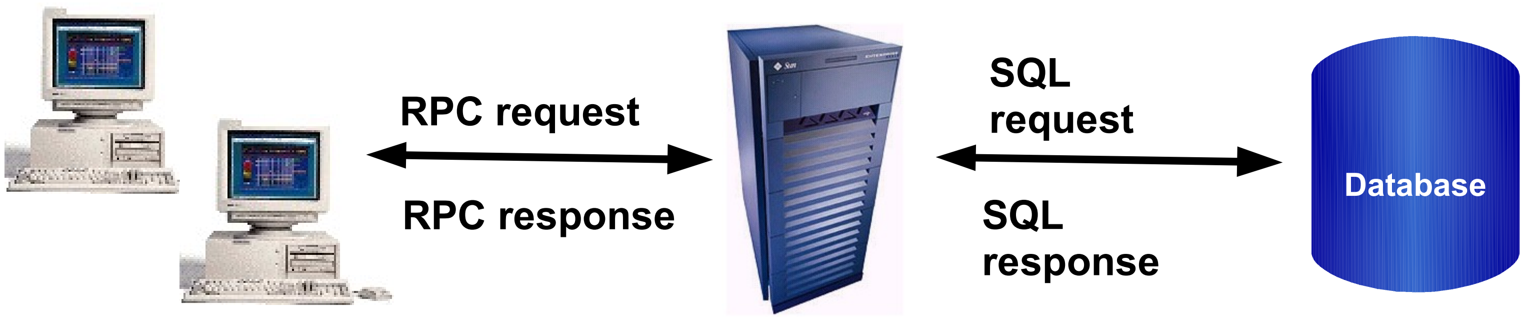


- Fat clients talking to back end database
  - SQL queries sent, raw data returned
- Presentation,Business logic and Data Model processing logic in client application

Two-Tier

- Pro:
  - DB product independence (compared to single-tier model)
- Cons:
  - Presentation, data model, business logic are intertwined (at client side), difficult for updates and maintenance
  - Data Model is “tightly coupled” to every client: If DB Schema changes, all clients break
  - Updates have to be deployed to all clients making System maintenance nightmare
  - DB connection for every client, thus difficult to scale
  - Raw data transferred to client for processing causes high network traffic

Three-Tier (RPC based)

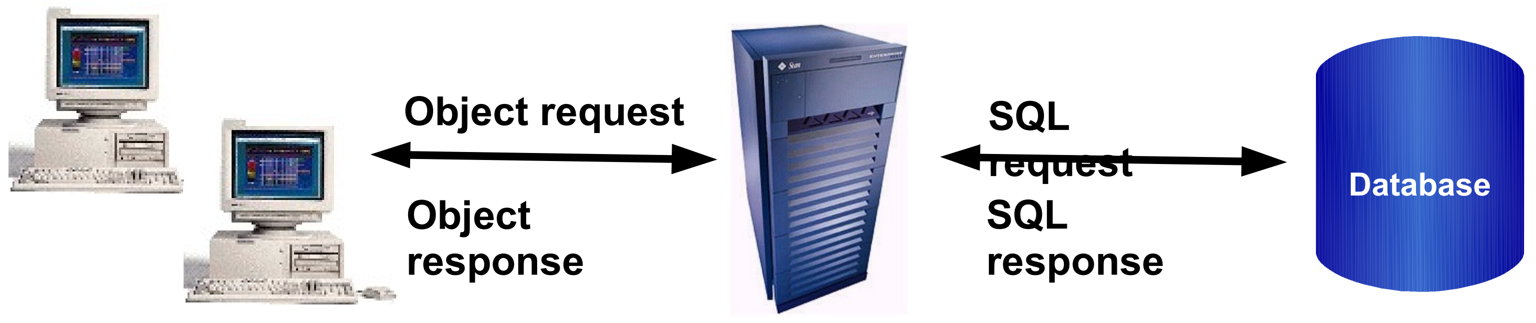


- Thinner client: business & data model separated from presentation
  - Business logic and data access logic reside in middle tier server while client handles presentation
- Middle tier server is now required to handle system services
  - Concurrency control, threading, transaction, security, persistence, multiplexing, performance, etc.

Three-tier (RPC based): Pros & Cons

- Pro:
  - Business logic can change more flexibly than 2-tier model
    - Most business logic reside in the middle-tier server
- Cons:
  - Complexity is introduced in the middle-tier server
  - Client and middle-tier server is more tightly-coupled (than the three-tier object based model)
  - Code is not really reusable (compared to object model based)

Three-Tier (Remote Object based)

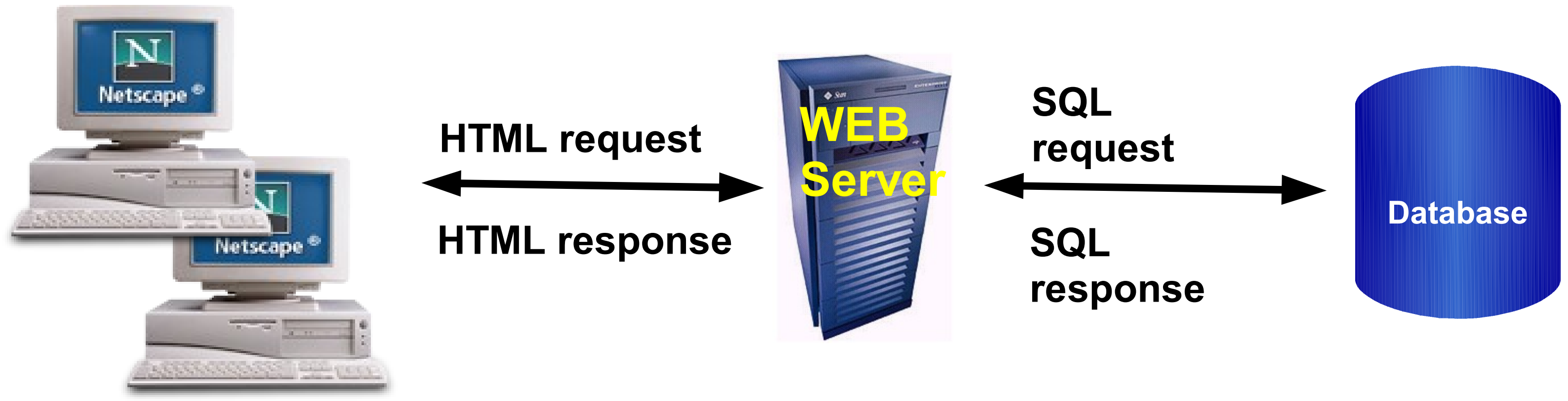


- Business logic and data model captured in objects
  - Business logic and data model are now described in “abstraction” (interface language)
- Object models used: CORBA, RMI, DCOM
  - Interface language in CORBA is IDL
  - Interface language in RMI is Java interface

Three-tier (Remote Object based): Pros & Cons

- Pro:
  - More loosely coupled than RPC model
  - Code could be more reusable
- Cons:
  - Complexity in the middle-tier still need to be addressed

Three-Tier (Web Server)



- Browser handles presentation logic
- Browser talks Web server via HTTP protocol
- Business logic and data model are handled by “dynamic contents generation” technologies (CGI, Servlet/JSP, ASP)

Three-tier (Web Server based): Pros & Cons

- Pro:
  - Ubiquitous client types
  - Zero client management
  - Support various client devices
    - J2ME-enabled cell-phones
- Cons:
  - Complexity in the middle-tier still need to be addressed

Trends

- Moving from single-tier or two-tier to multi-tier architecture
- Moving from monolithic model to object-based application model
- Moving from application-based client to HTML-based client

Single-tier vs. Multi-tier

Single tier	Multi-tier
• No separation among presentation, business logic, database • Hard to maintain	• Separation among presentation, business logic, database • More flexible to change, i.e. presentation can change without affecting other tiers

Monolithic vs. Object-based

Monolithic	Object-based
• 1 Binary file • Recompiled, relinked, redeployed every time there is a change	• Pluggable parts • Reusable • Enables better design • Easier update • Implementation can be separated from interface • Only interface is published

Outstanding Issues & Solution

- Complexity at the middle tier server still remains
- Duplicate system services still need to be provided for the majority of enterprise applications
  - Concurrency control, Transactions
  - Load-balancing, Security
  - Resource management, Connection pooling
- How to solve this problem?
  - Commonly shared container that handles the above system services
  - Proprietary versus Open-standard based

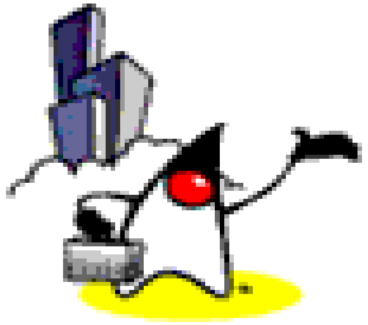
Proprietary Solution

- Use "component and container" model
  - Components captures business logic
  - Container provides system services
- The contract between components and container is defined in a well-defined but with proprietary manner
- Problem of proprietary solution: Vendor lock-in
- Example: Tuxedo, .NET

Open and Standard Solution

- Use "component and container" model in which container provides system services in a well-defined and as industry standard
- Java EE is that standard that also provides portability of code because it is based on Java technology and standard-based Java programming APIs

Why Java EE?



Platform Value to Developers

- Can use any Java EE implementation for development and deployment
  - Use production-quality standard implementation which is free for development/deployment
  - Use high-end commercial Java EE products for scalability and fault-tolerance
- Vast amount of Java EE community resources
  - Many Java EE related books, articles, tutorials, quality code you can use, best practice guidelines, design patterns etc.
- Can use off-the-shelf 3rd-party business components

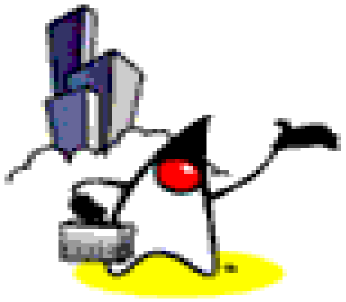
Platform Value to Vendors

- Vendors work together on specifications and then compete in implementations
  - In the areas of Scalability, Performance, Reliability, Availability, Management and development tools, and so on
- Freedom to innovate while maintaining the portability of applications
- Do not have create/maintain their own proprietary APIs

Platform Value to Business Customers

- Application portability
- Many implementation choices are possible based on various requirements
  - Price (free to high-end), scalability (single CPU to clustered model), reliability, performance, tools, and more
  - Best of breed of applications and platforms
- Large developer pool

Java EE APIs & Technologies



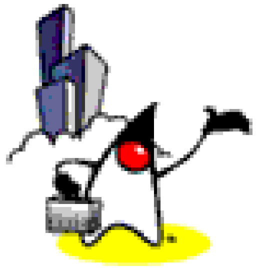
Java EE 1.4 APIs and Technologies

- J2SE 1.4 (improved)
- JAX-RPC (new)
- Web Service for Java EE
- Java EE Management
- Java EE Deployment
- JMX 1.1
- JMS 1.1
- JTA 1.0
- Servlet 2.4
- JSP 2.0
- EJB 2.1
- JAXR
- Connector 1.5
- JACC
- JAXP 1.2
- JavaMail 1.3
- JAF 1.0

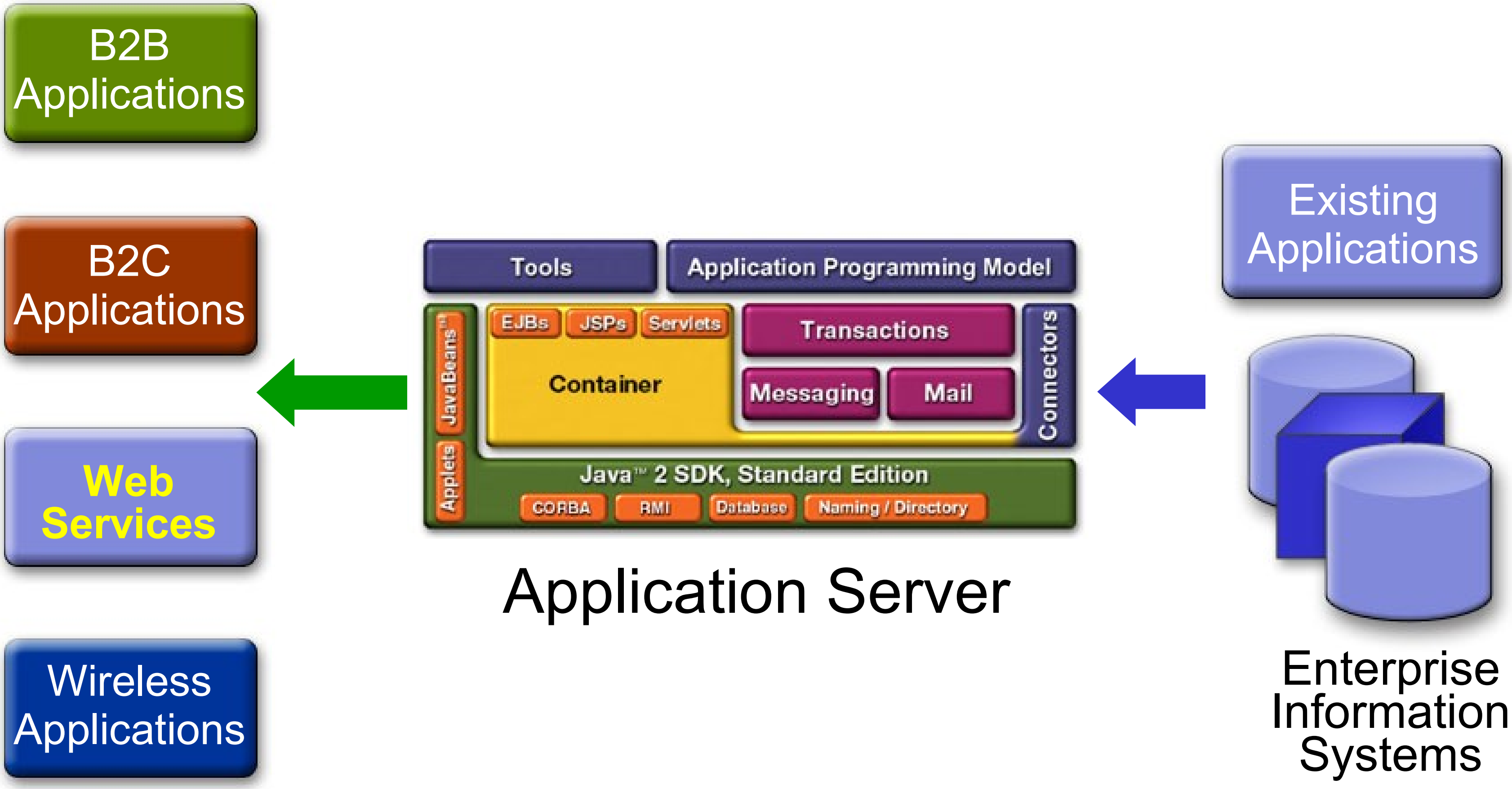
Java EE 5

- JAX-WS 2.0 & JSR 181
- Java Persistence
- EJB 3.0
- JAXB 2.0
- JavaSever Faces 1.2 – new to Platform
- JSP 2.1 – Unification w/ JSF 1.2
- StAX – Pull Parser – new to Platform

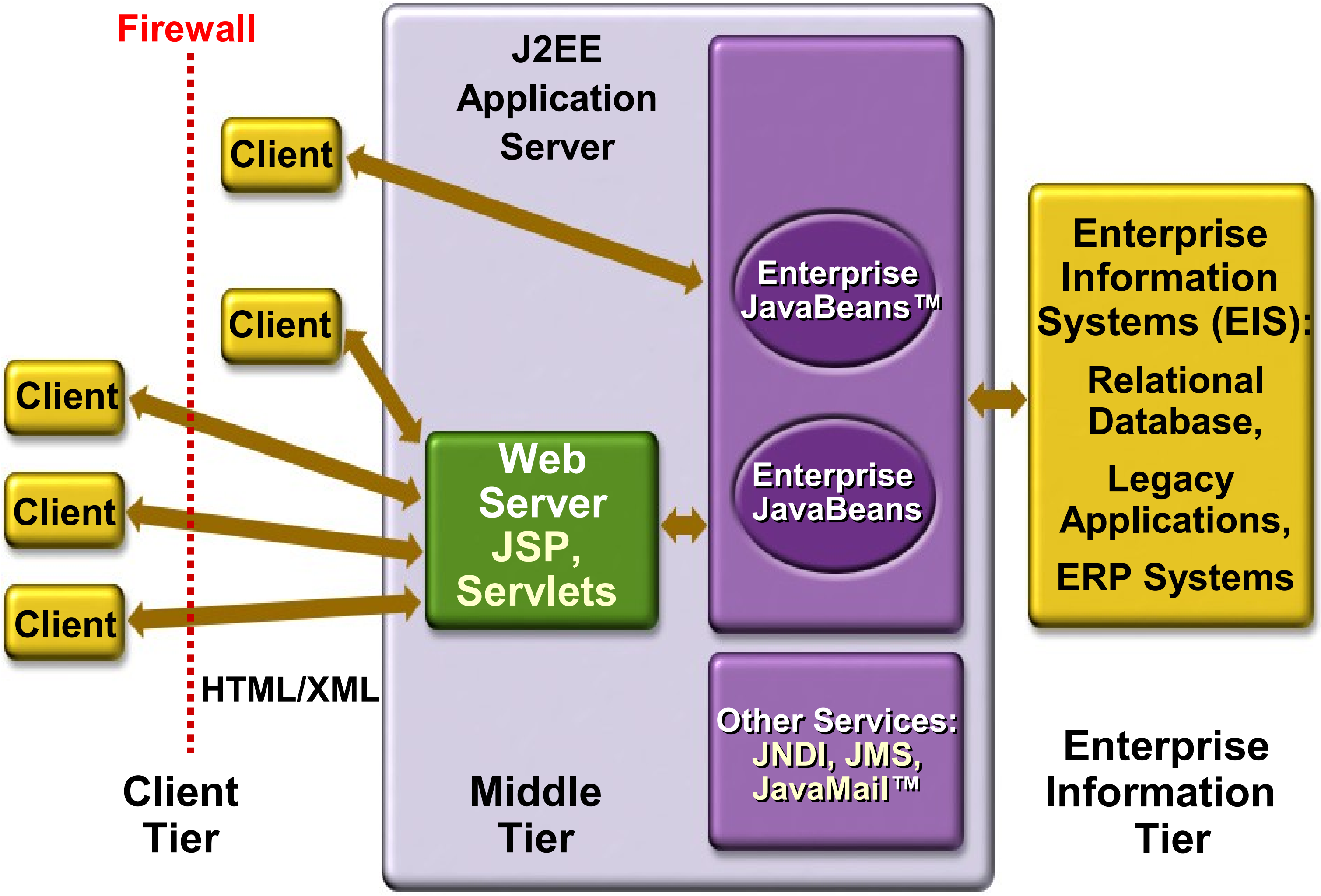
J2EE is an End-to-End Architecture



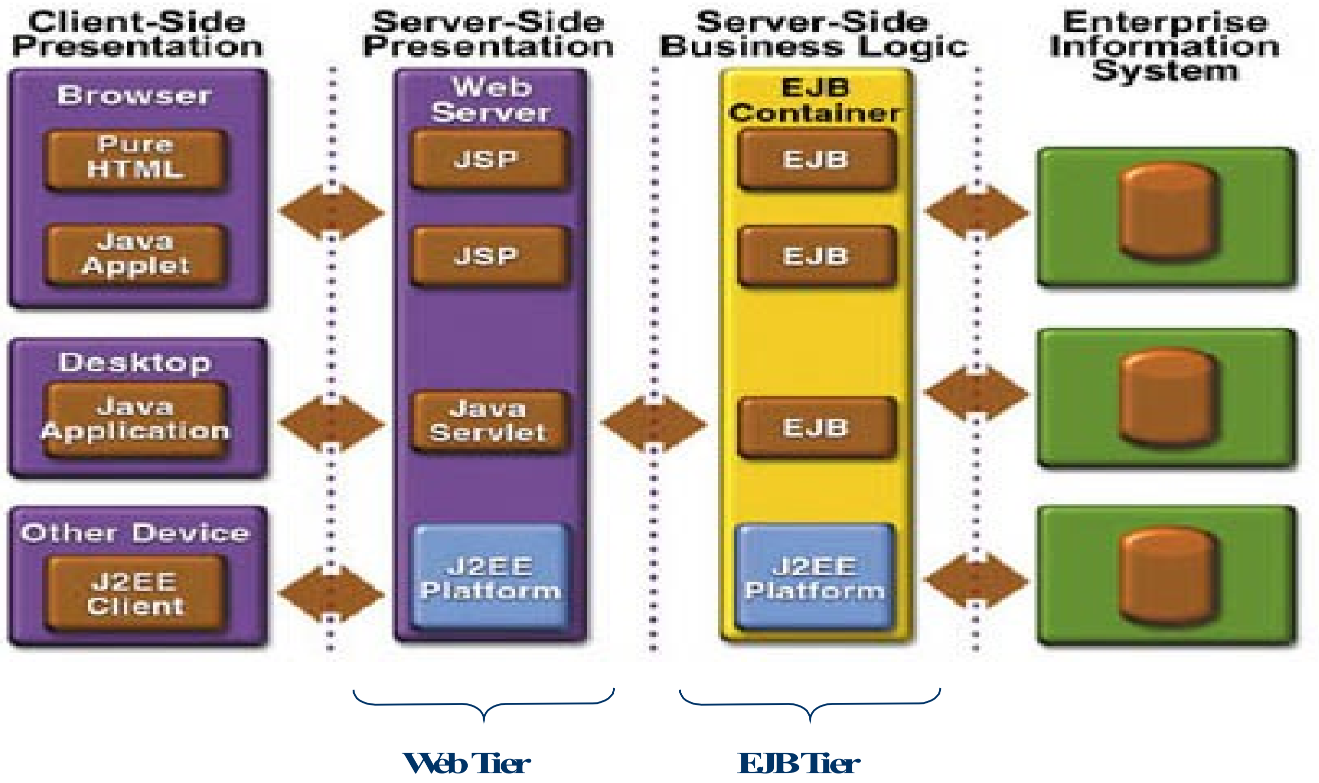
The Java EE Platform Architecture



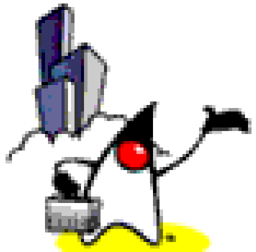
Java EE is End-to-End Solution



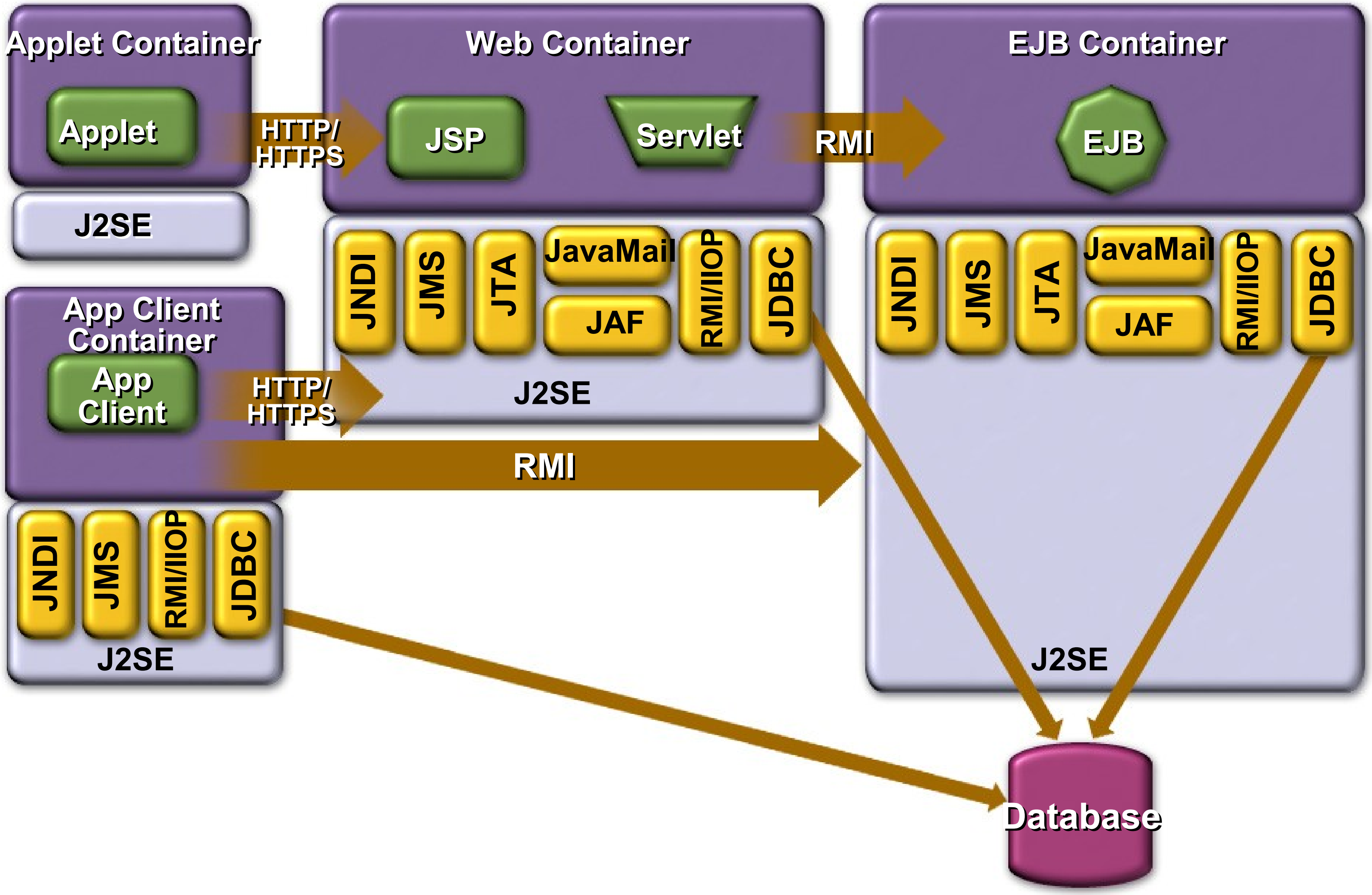
N-tier Java EE Architecture



Java EE Component & Container Architecture



Java EE Containers & Components



Containers and Components

Containers Handle

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

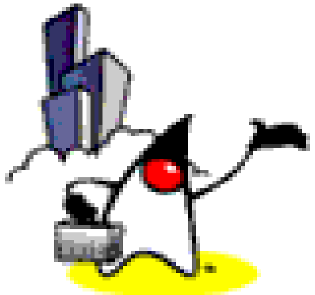
Components Handle

- Presentation
- Business Logic

Containers & Components

- Containers do their work invisibly
  - No complicated APIs
  - They control by interposition
- Containers implement Java EE
  - Look the same to components
  - Vendors making the containers have great freedom to innovate

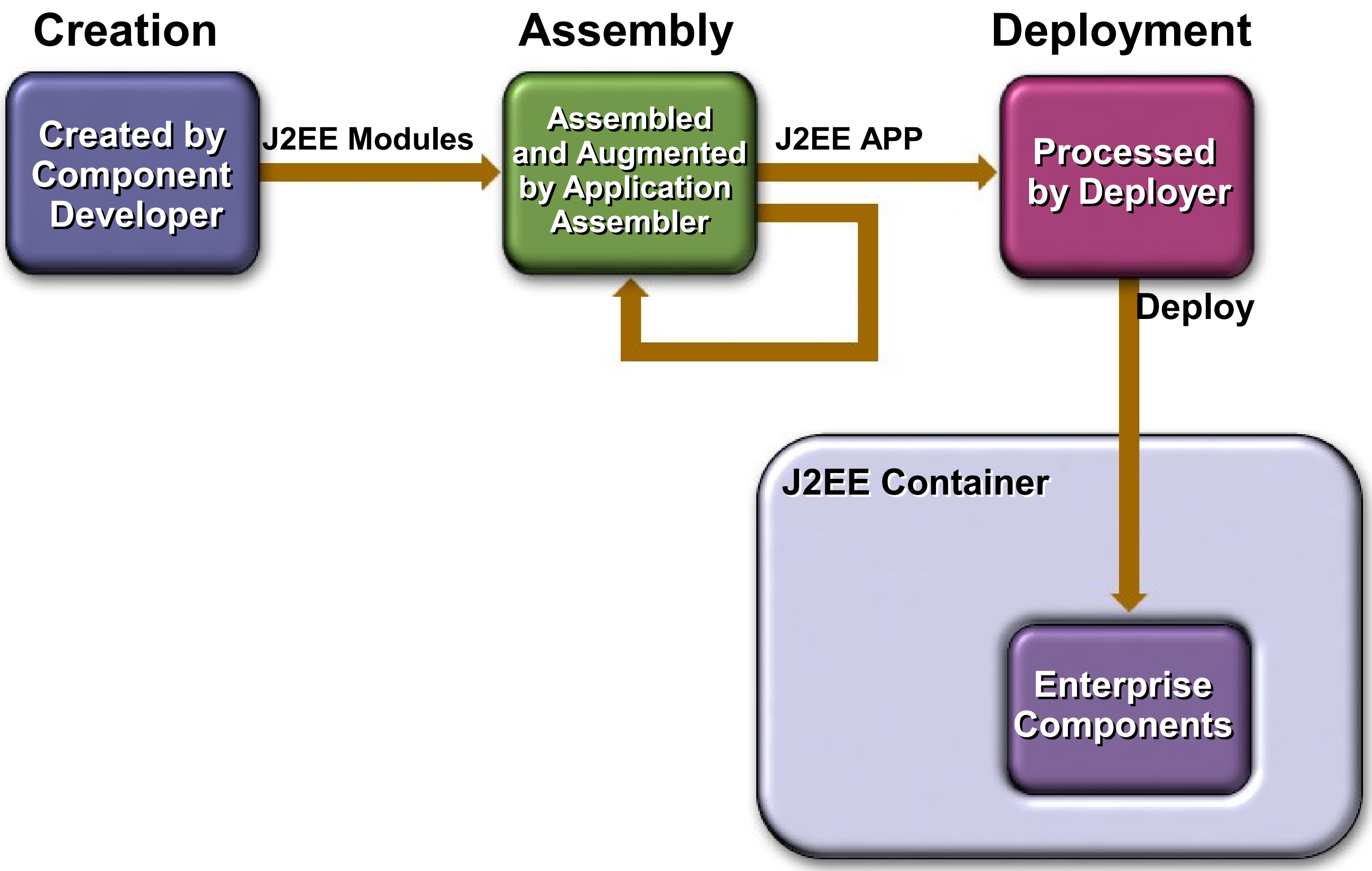
Java EE Application Development & Deployment Life Cycle



Java EE Application Development Lifecycle

- Write and compile component code
  - Servlet, JSP, EJB
- Write deployment descriptors for components
  - From Java EE 5, you can use annotations
- Assemble components into ready-to-deployable package
- Deploy the package on a server

Life-cycle Illustration



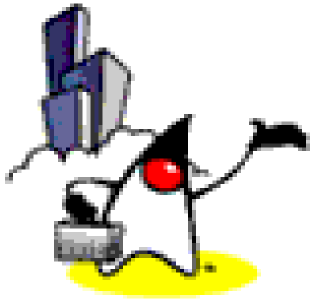
Java EE Development Roles

- Component provider
  - Bean provider
- Application assembler
- Deployer
- Platform provider
  - Container provider
- Tools provider
- System administrator

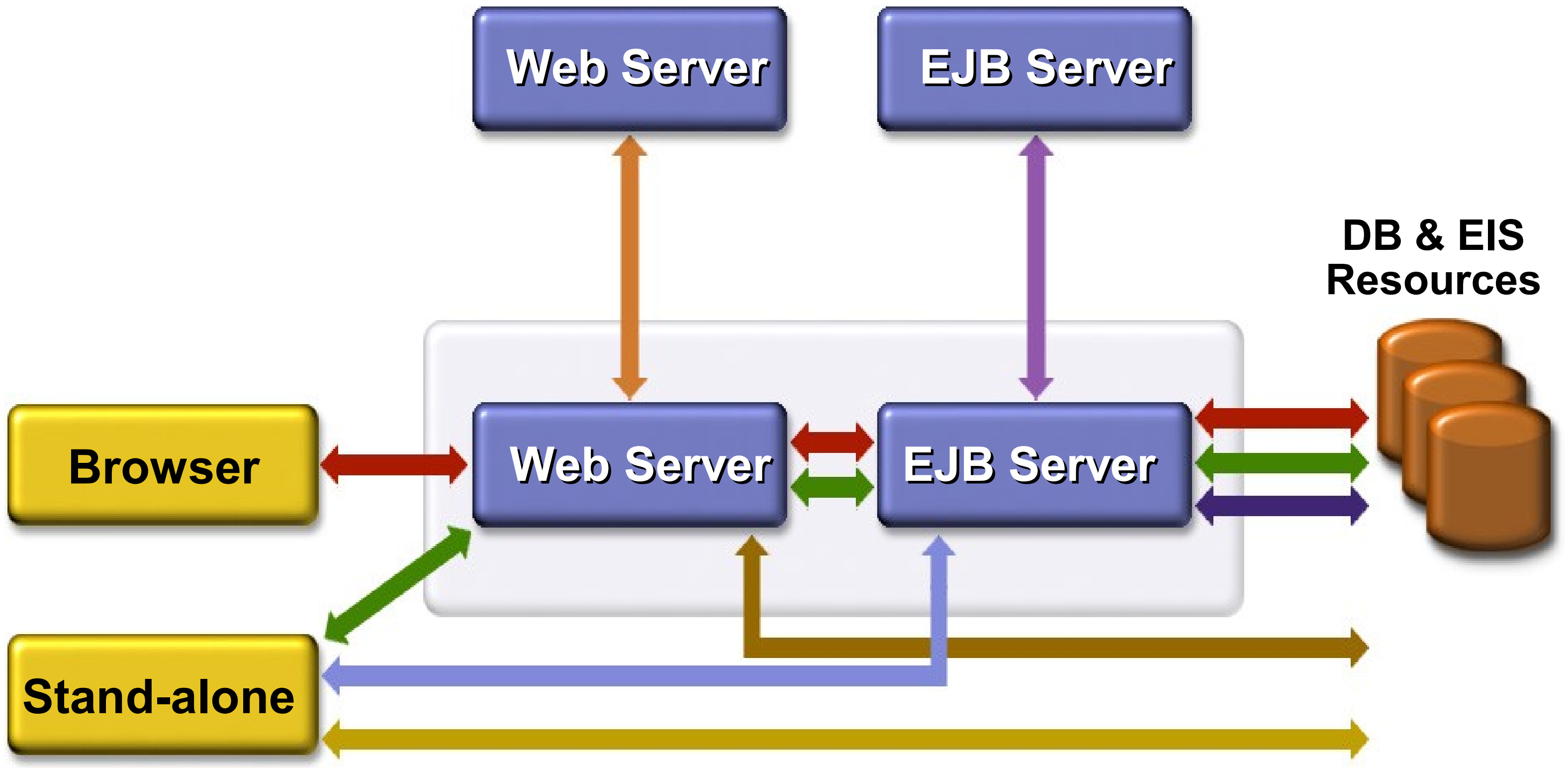
The Deployment Descriptor

- Gives the container instructions on how to manage and control behaviors of the Java EE components
  - Transaction
  - Security
  - Persistence
- Allows declarative customization (as opposed to programming customization)
  - XML file
- Enables portability of code

Java EE Application Anatomies



Possible Java EE Application Anatomies



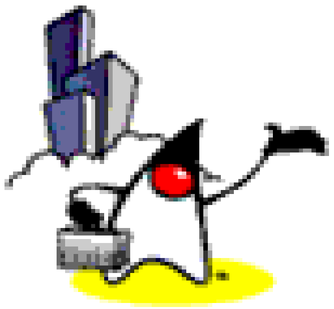
Java EE Application Anatomies

- 4-tier Java EE applications
  - HTML client, JSP/Servlets, EJB, JDBC/Connector
- 3-tier Java EE applications
  - HTML client, JSP/Servlets, JDBC
- 3-tier Java EE applications
  - EJB standalone applications, EJB, JDBC/Connector
- B2B Enterprise applications
  - Java EE platform to Java EE platform through the exchange of JMS or XML-based messages

Which One to Use?

- Depends on several factors
  - Requirements of applications
  - Availability of EJB tier
  - Availability of developer resource

How to Get Started



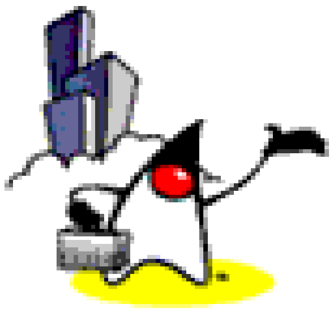
Step1: For Beginners and Intermediate Java EE Programmers

- Follow along with this course
- Start using Java EE IDE of your choice
- Try open source IDE's
  - NetBeans IDE (netbeans.org)
  - Eclipse

Step2: Next Step (For Advanced Java EE Programmers)

- Learn practical open-source solutions
  - Spring framework (for light-weight framework)
  - Hibernate (for O/R mapping)
  - JDO (for transparent persistence)
  - Struts, WebWork, Tapestry (for Web-tier frameworks)
  - JUnit (for unit testing)
  - Log4j (for logging)
  - Many more

Summary & Resources



Summary

- Java EE is the platform of choice for development and deployment of n-tier, web-based, transactional, component-based enterprise applications
- Java EE is standard-based architecture
- Java EE is all about community
- Java EE evolves according to the needs of the industry

Thank you!

Sang Shin
Michèle Garoche
http://www.javapassion.com
"Learning is fun!"