



# Integrating Web Services Architectures with Existing J2EE Architectures

Presented By: Brad Little  
Capstone Consulting

[www.capstoneC.com](http://www.capstoneC.com)

# Presentation Agenda

- J2EE
- Web Services
- Integration Discussion
- Case Study

# J2EE

# Application Evolution

- Mainframe
- Client Server
- Distributed Computing



# Client Server Applications

- Prior to mid 1990s, most applications executed on a single machine.
- The first attempt at distributing an application:
  - Application broken into a series of client-server processes.
  - Local copies of databases which were replicated and synchronized often.
- Problems with client-server approach:
  - Single point of failure with processes.
  - Hard to scale (“Ripple Effect”)
  - Proprietary protocols

# Distributed Application Goals

- High Availability
  - Systems need to be up 24/7/365 for client use.
- Scalability
  - Additional resources can be easily added to meet growing demand.
- Maintainability
  - Keep business logic in reusable units
  - Isolate database access from the user interface

# Application Layers

- Applications can be divided into 3 layers:
  - Presentation Logic
    - Users view of application (windows, browsers, command line)
  - Business Logic
    - The components that implement your business practices and policies
  - Data Access Logic
    - Retrieval and storage of data to RDBMS, files, legacy systems.
- By keeping these layers separate, reusability and maintainability are maximized.



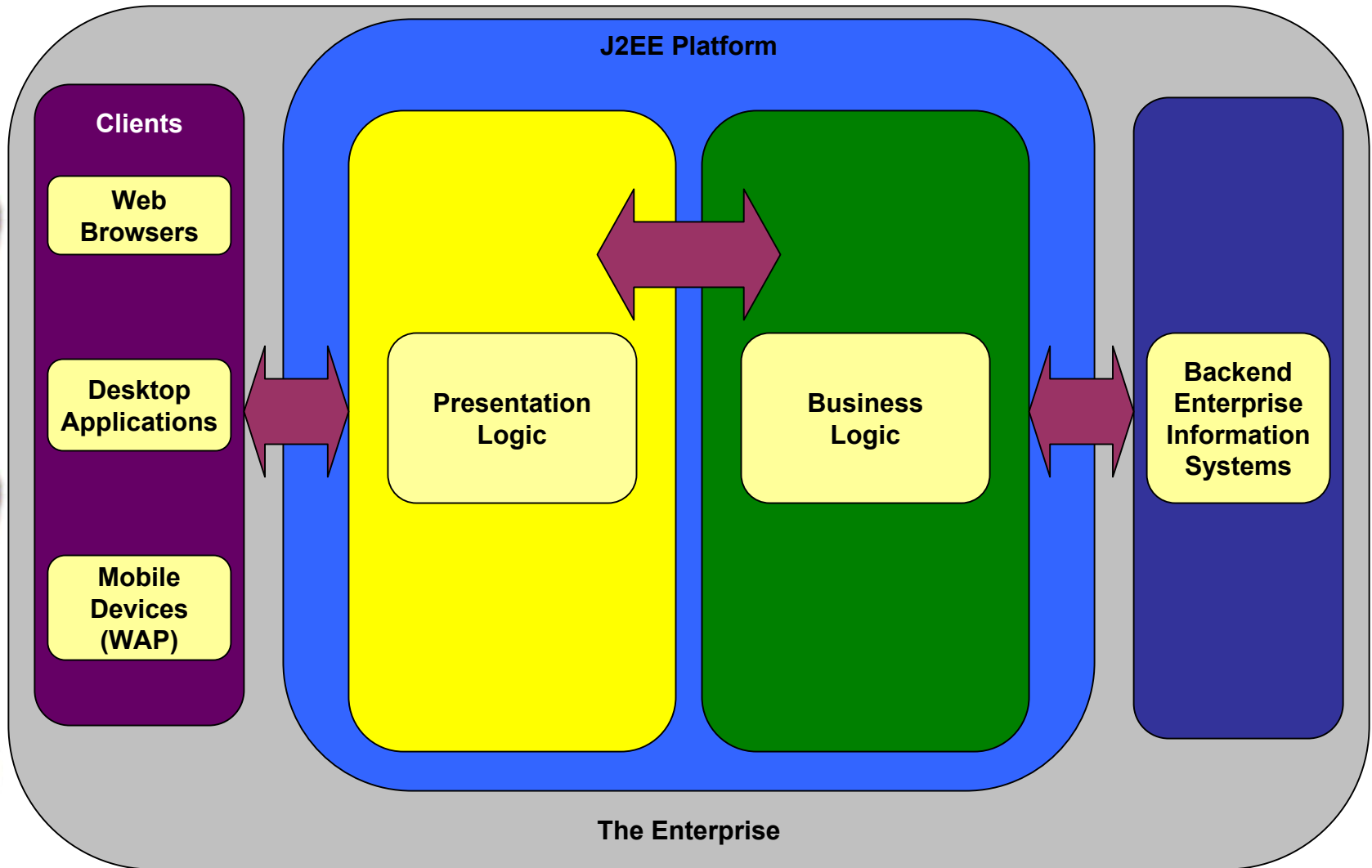
# What is J2EE?

- Bundle of Java™ technology standards
- Middleware infrastructure that serves as the building block upon which highly available, scalable, and maintainable enterprise systems can be built

# What is J2EE?

- A reference platform
- Implementations of this platform are referred to as Application Servers
- Market Leading Application Servers
  - BEA WebLogic Server
  - IBM WebSphere
  - Sun/Netscape iPlanet
  - Allaire Jrun
  - And others...

# High-Level J2EE Architecture



# Motivation for J2EE

- Faster time to market
- Shortage of developers
- Multiple platforms
- Distributed applications
- Reduced costs
- Component reuse
- Ease of integration

# Motivation for J2EE

- Easy-to-access services to their customers, partners, employees, and suppliers
- Standard for developing enterprise applications
- Making middleware easier, less complex coding

# J2EE Applications

- Becoming the de-facto standard for distributed component development
  - **Highly Available**, to meet the needs of today's global business environment.
  - **Scalable**, to insure that business transactions are promptly processed.
  - **Maintainable**, by breaking business logic into reusable components.
  - **Secure**, to protect the privacy of users and integrity of the enterprise.
  - **Reliable**, to ensure that business transactions are accurately processed.



# J2EE Platform Benefits

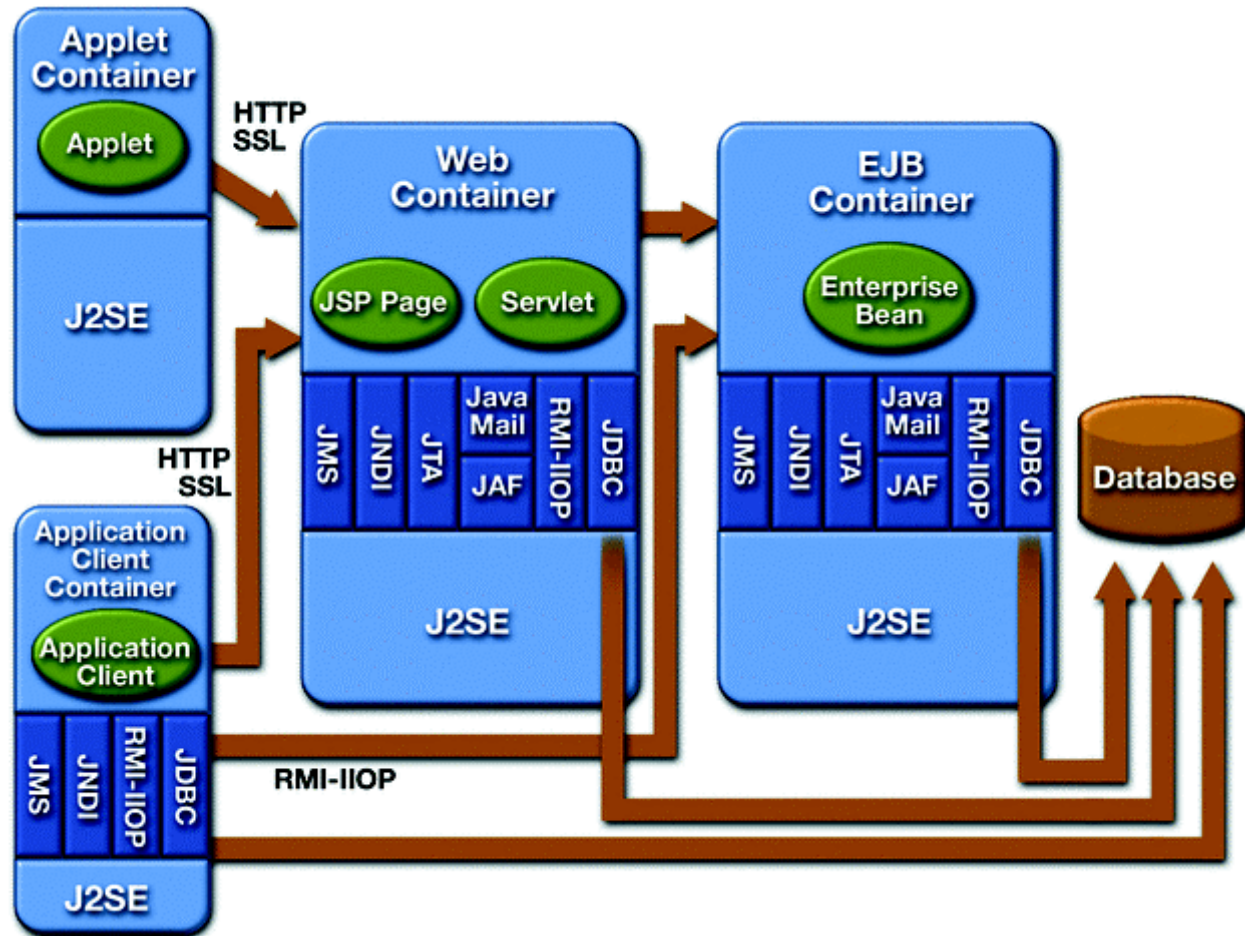
- Simplified architecture and development
  - Maps easily to application functionalities
  - Enables assembly-time and deploy-time behaviors
  - Supports separation of expertise
- Scalability and automatic load-balancing to meet demand variations (i.e. Connection Pooling for faster access to data)



# J2EE Platform Benefits

- Integration with existing information systems
  - Accessing relational data from Java™
  - Managing and coordinating transactions across heterogeneous enterprise information systems
  - Accessing information in enterprise naming and directory services
  - Sending and receiving messages via enterprise messaging systems
  - Sending and receiving e-mail
  - Calling Common Object Request Broker Architecture (CORBA) services

# J2EE Architecture



# J2EE Application Components

- Application Clients
  - Java™ programming language programs (typically GUI programs) that execute on a desktop computer.
- Applets
  - GUI component that typically execute in a web browser.
- Servlets and Java Server Pages (JSPs)
  - Typically execute in a web server and respond to HTTP requests from web clients. May be used to generate HTML pages.
- Enterprise Java Beans (EJBs)
  - Components that execute in a managed environment that supports transactions. EJBs typically contain the business logic.

# J2EE Technologies

- RMI
  - Remote Method Invocation, Protocol for distributed communication
- Servlets
  - Java code which serves up dynamic web pages, replaces CGIs
- JSP
  - Java Server Pages, Combination of HTML and Java which can be intermixed, may be used in place of Servlets.
- JNDI
  - Java Naming & Directory Interface, Provides a common interface to communicate with different naming and directory services

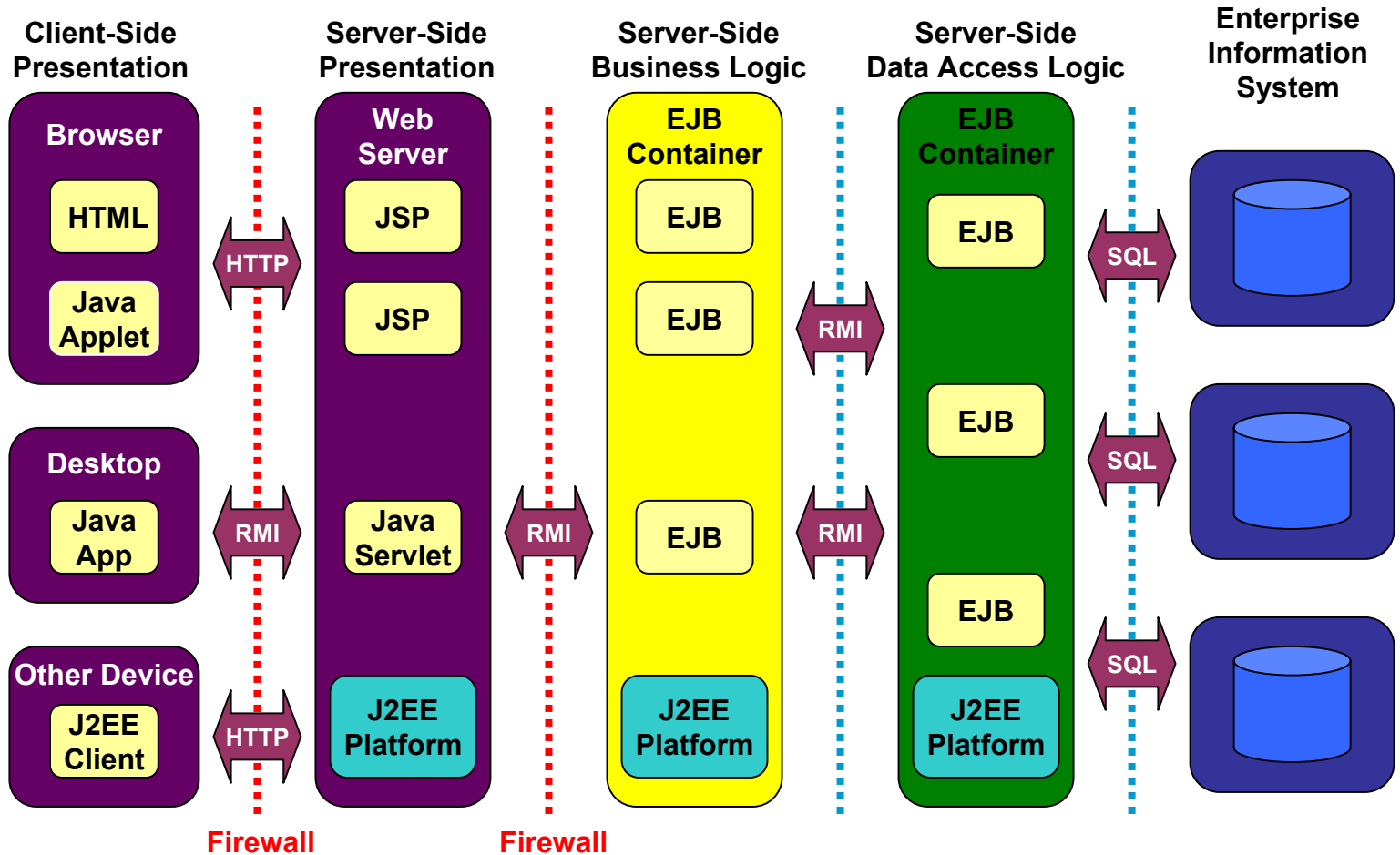
# J2EE Technologies

- **JDBC**
  - Java Database Connectivity, Provides a common interface to heterogeneous databases.
- **EJB**
  - Enterprise Java Beans, Distributed Java components.
- **JMS**
  - Java Message Service, Send and receive asynchronous messages.
- **JTA**
  - Java Transaction API, Provides a common interface for Java Transactions.
- **JTS**
  - Java Transaction Service, An OTS compliant transaction monitor.

# J2EE Technologies

- **JavaMail**
  - Standard way to communicate with mail servers.
- **JAF**
  - Java Activation Framework, Used in conjunction with JavaMail API, Integrates other mime types with Java
- **Java IDL**
  - Allows CORBA clients to access EJBs
- **JCA**
  - Java Connector Architecture, provides a standard way for back-end applications to plug into the J2EE platform.
- **XML**
  - eXtensible Markup Language, Allows users to define custom message and configuration formats.

# J2EE Architecture





# Web Services

# Motivation for Web Services

- Industry-wide uniformity with low cost of entry
  - There are many problematic existing standards for B2B wide-area collaboration
    - EDI
    - CORBA
- Allows us to use existing infrastructure
  - Technology we already have and are comfortable using
  - Building upon existing expertise and confidence

# Motivation for Web Services

- Allow partners, suppliers, and customers to access internal resources
- Utilize existing infrastructure
- Ease of integration
- Simple to implement

# Motivation for Web Services

- Component “wars”
- Language “wars”
- Firewalls
- No industry standard on interoperability

# Web Services Benefits

- Interoperable despite Component & Language “wars”
- B2B applications are cheaper and leverage the existing Internet
- EAI efforts become more transparent
- Loose application coupling
- Build services first – find partners later

# Web Services Technologies

- SAX
  - Simple API for XML, An event-driven interface in which the parser invokes one of several methods supplied by the caller when a "parsing event" occurs. "Events" include recognizing an XML tag, finding an error, encountering a reference to an external entity, or processing a DTD specification.
- DOM
  - Document Object Model, A tree of objects with interfaces for traversing the tree and writing an XML version of it, as defined by the W3C specification.

# Web Services Technologies

- XSLT
  - XML Stylesheet Language Transformations, a mechanism to convert an XML file document from one schema to another.
- UDDI
  - Universal Description, Discovery, and Integration, Globally unique public directory of business & services (*Green Pages*)
- WSDL
  - Web Services Description Language, an XML document describing the interface, semantics, and administration information of a call to a given web service



# Web Services Technologies

- SOAP
  - Simple Object Access Protocol, a spec for using XML documents to call business methods using various protocols, including HTTP.
- ebXML
  - electronic business eXtensible Markup Language, a collection of XML specs to provide an e-infrastructure for B2B collaboration and integration

# Web Services Technologies

- JAXP
  - Java API for XML Parsing, interface to the industry standard parsers SAX and DOM, including a “pluggable” interface to XSLT engines
- JAXR
  - Java API for XML Registries, spec for accessing registries in general, including UDDI (*future*)

# Web Services Technologies

- JWSDL
  - Java API for WSDL, spec for manipulating WSDL documents without direct document interaction (*future*)
- JAX/RPC
  - Java API for XML based RPC, used to send/marshall and receive/unmarshall method calls using XML based protocols, including SOAP (*future*)

# Web Services Technologies

- JAXM
  - Java API for XML Messaging, spec for interacting with XML messaging standards, including ebXML and SOAP (*future*)
- JAXB
  - Java Architecture for XML Binding, provides an API and tools that automate the mapping between XML documents and Java objects.



# Web Services Architecture

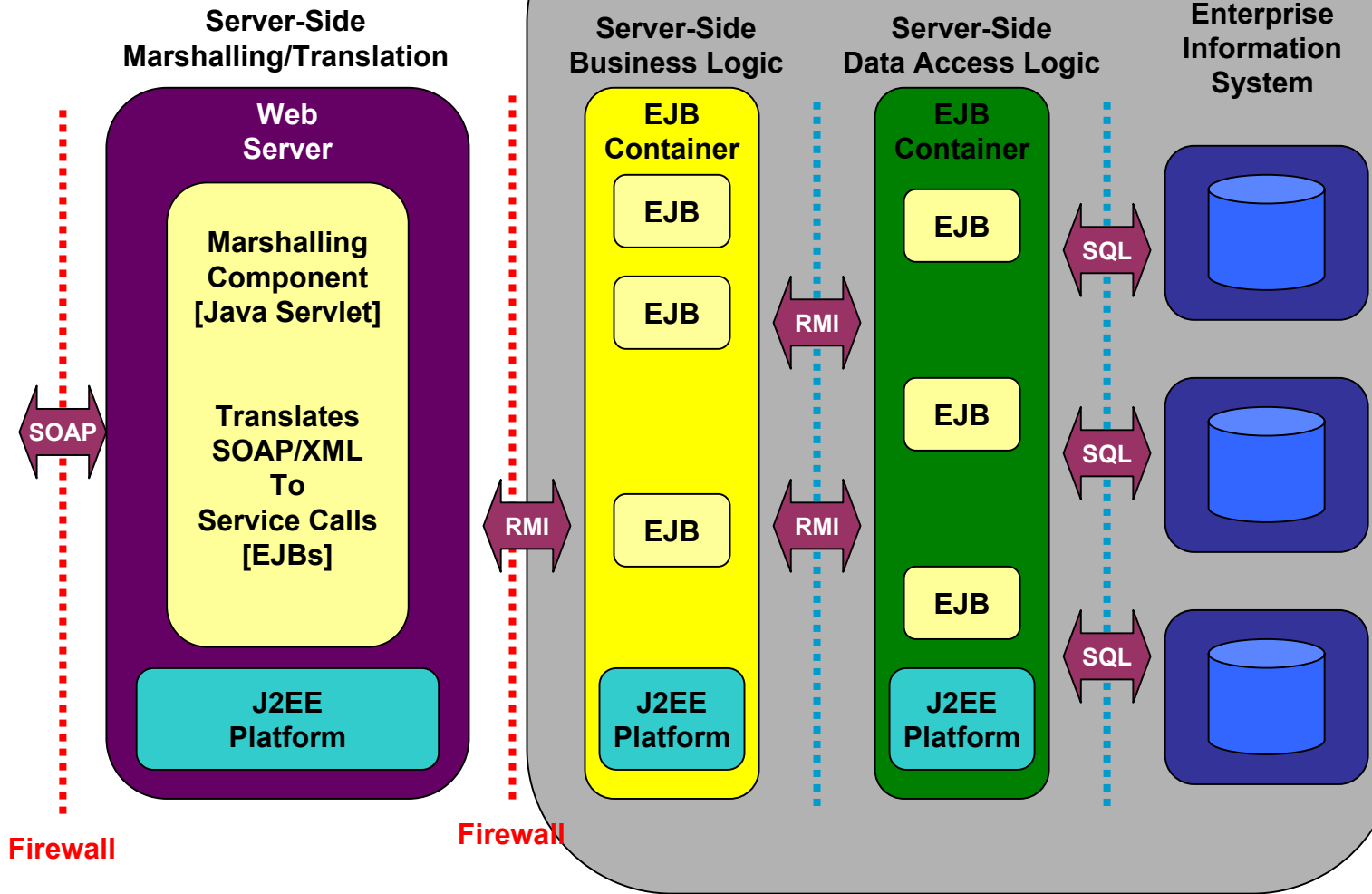
- 4 Key Functional Components
  - Service Implementation
  - Publishing
  - Discovery
  - Invocation

# Web Services Architecture

- Service Implementation
  - Existing Services
    - Consider wrapping existing services with an API representing a coarse grained web service
  - New Services
    - Build a service using familiar technologies (J2EE, JDBC, JMS, JNDI, JCA, etc...)
  - Create a Web Services interface to your service
    - Available tools:
      - Apache SOAP
      - BEA WebLogic 6.1 Beta
      - CapeClear CapeConnect
      - IBM Web Services Toolkit
      - The Mind Electric GLUE
      - SilverStream eXtend suite



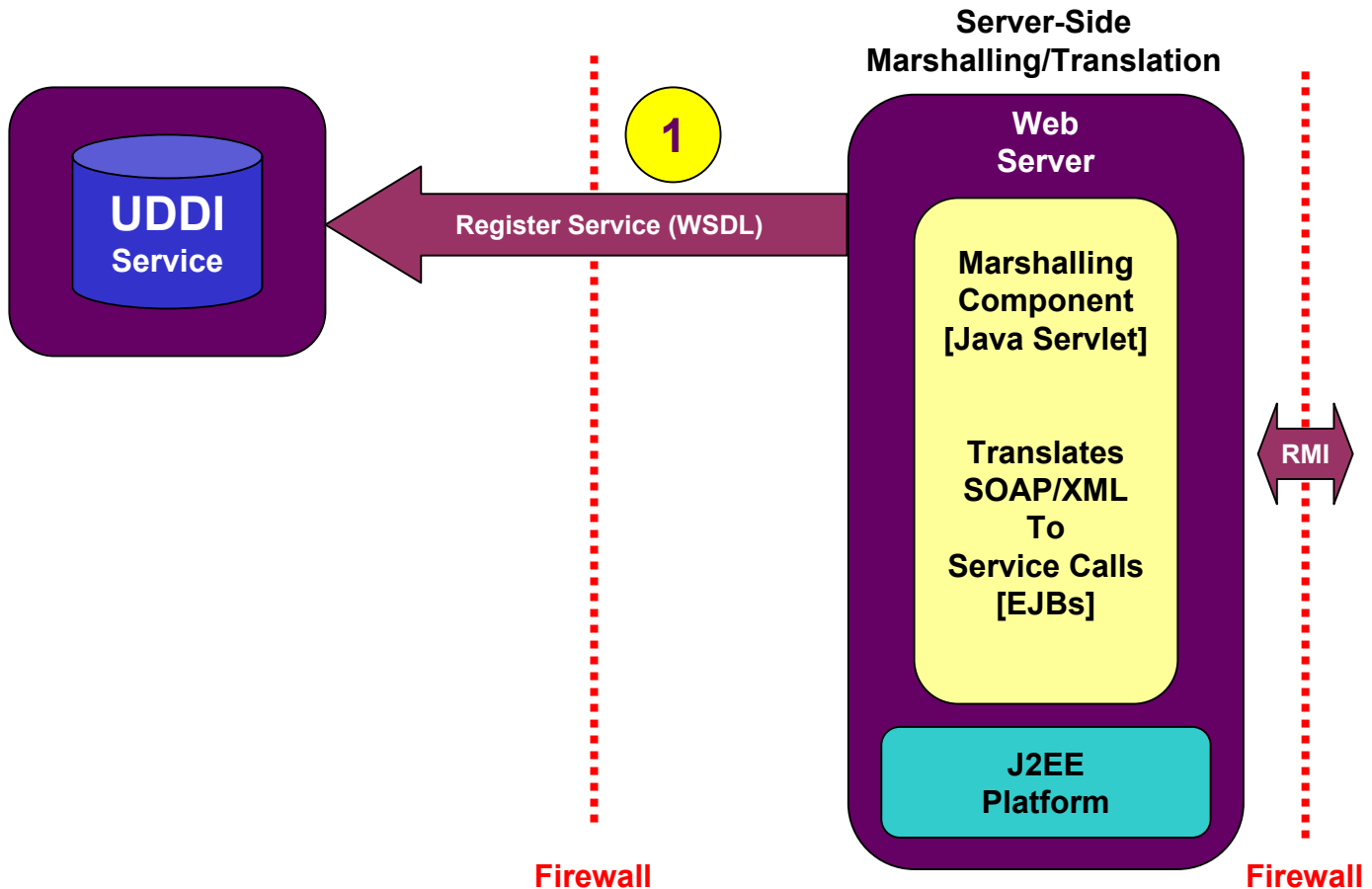
# Service Implementation



# Web Services Architecture

- Publishing
  - Author WSDL document
    - Create the appropriate WSDL document manually or using a tool that will generate it automatically using introspection on your Java™ interfaces and classes.
  - Publish WSDL document
    - Place the WSDL document on an accessible web server.
  - Publish WSDL existence to UDDI
    - Manually or programmatically, create appropriate entries into UDDI that announce the existence of your WSDL document and where to find it.

# Publishing





# Web Services Architecture

- Discovery

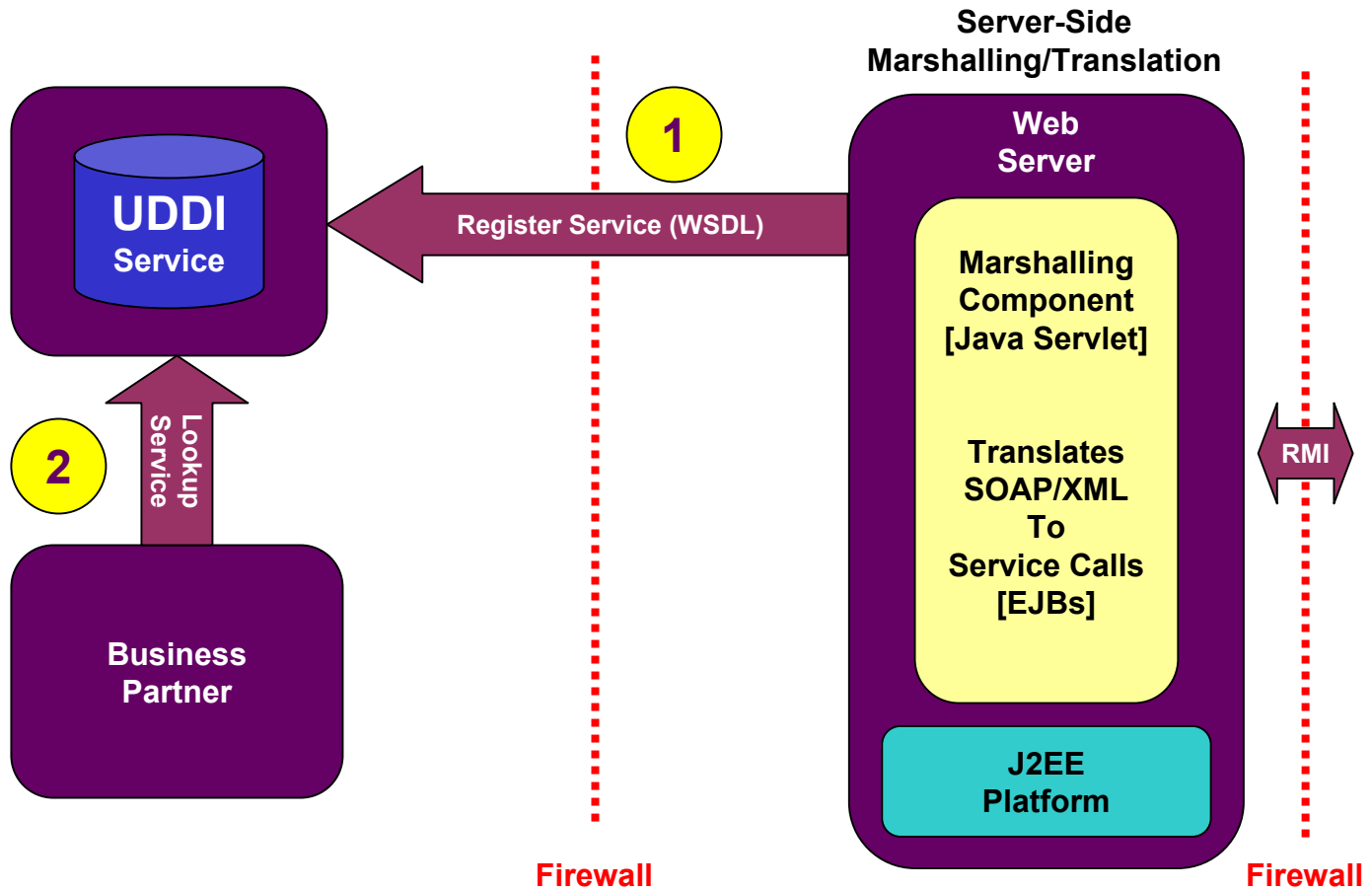
- Locate desired service

- Search UDDI to find needed service using pattern queries.
    - Retrieve the UDDI entry for the service to get the end-point URL for interface documentation

- Obtain the service interface document

- Download the WSDL document (or other documentation) from the end-point URL retrieved from UDDI.

# Discovery





# Web Services Architecture

- Invocation

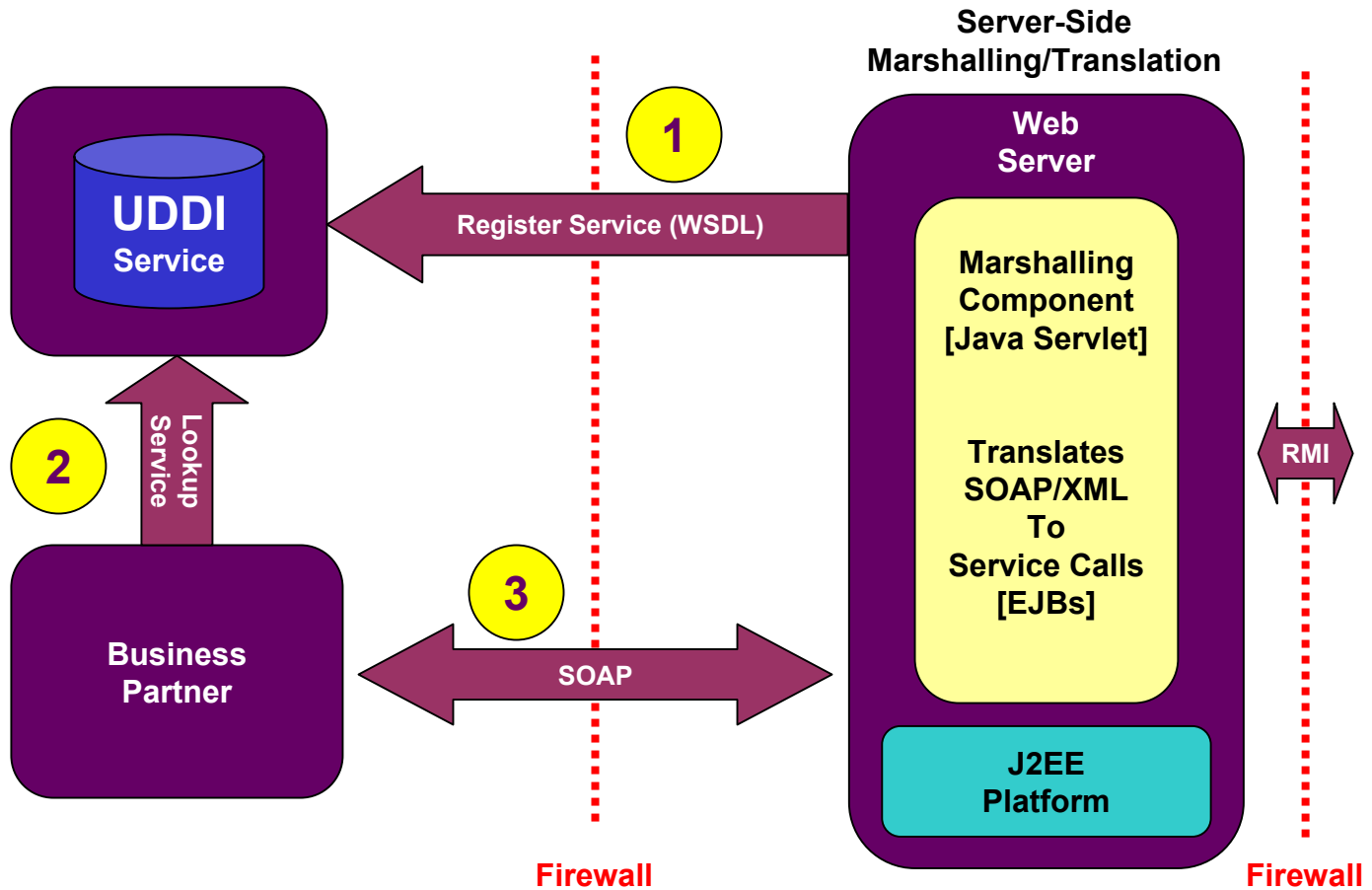
- Author a SOAP Client

- Using the interface outlined in the WSDL document, author a client component to invoke the SOAP service.
    - Consider using various industry tools for generating the needed client proxy stubs if the SOAP implementation you are using allows for it.

- Make a SOAP Call

- Using the SOAP client, form a SOAP call and execute it.

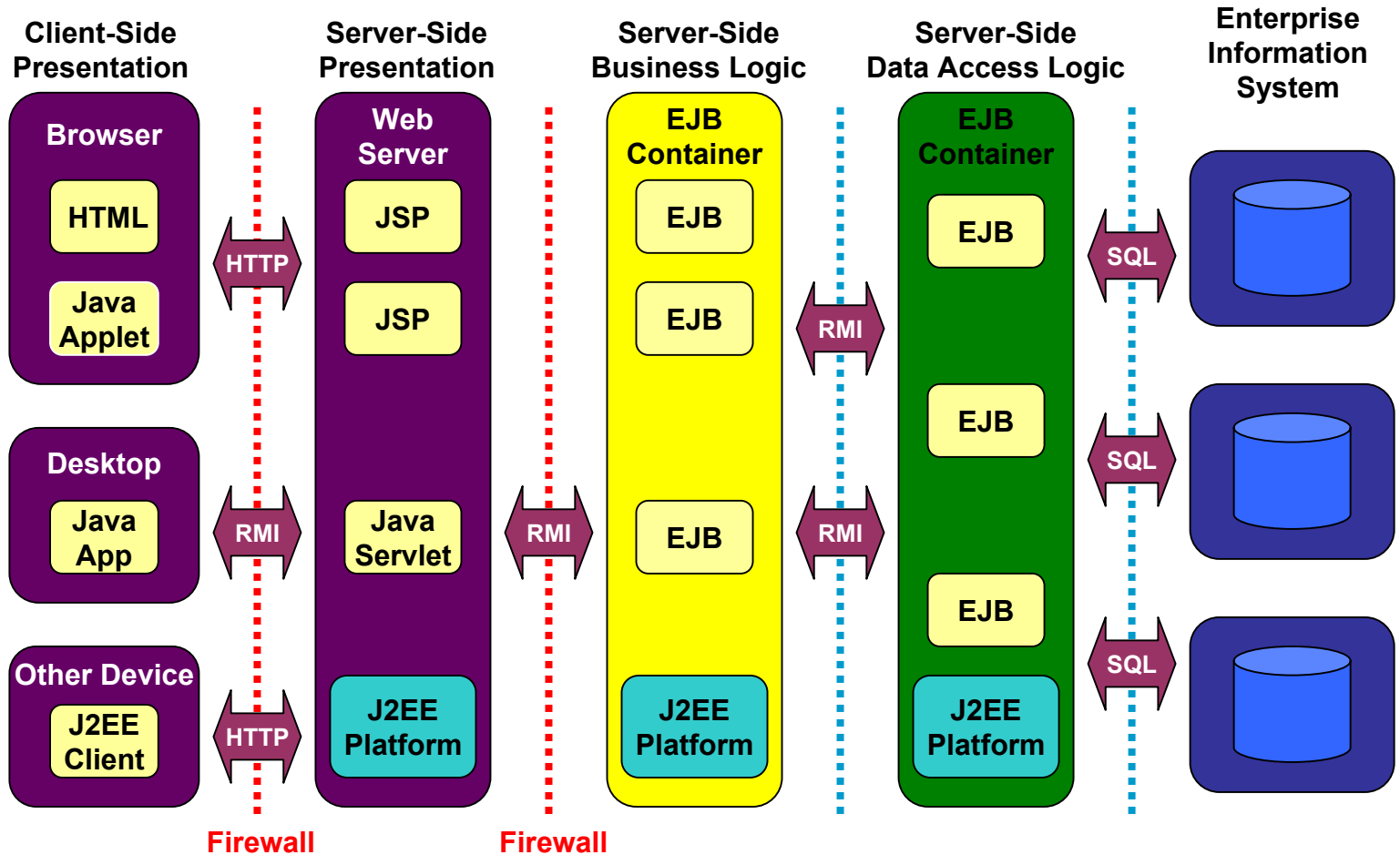
# Invocation





# Integration Discussion

# J2EE Architecture



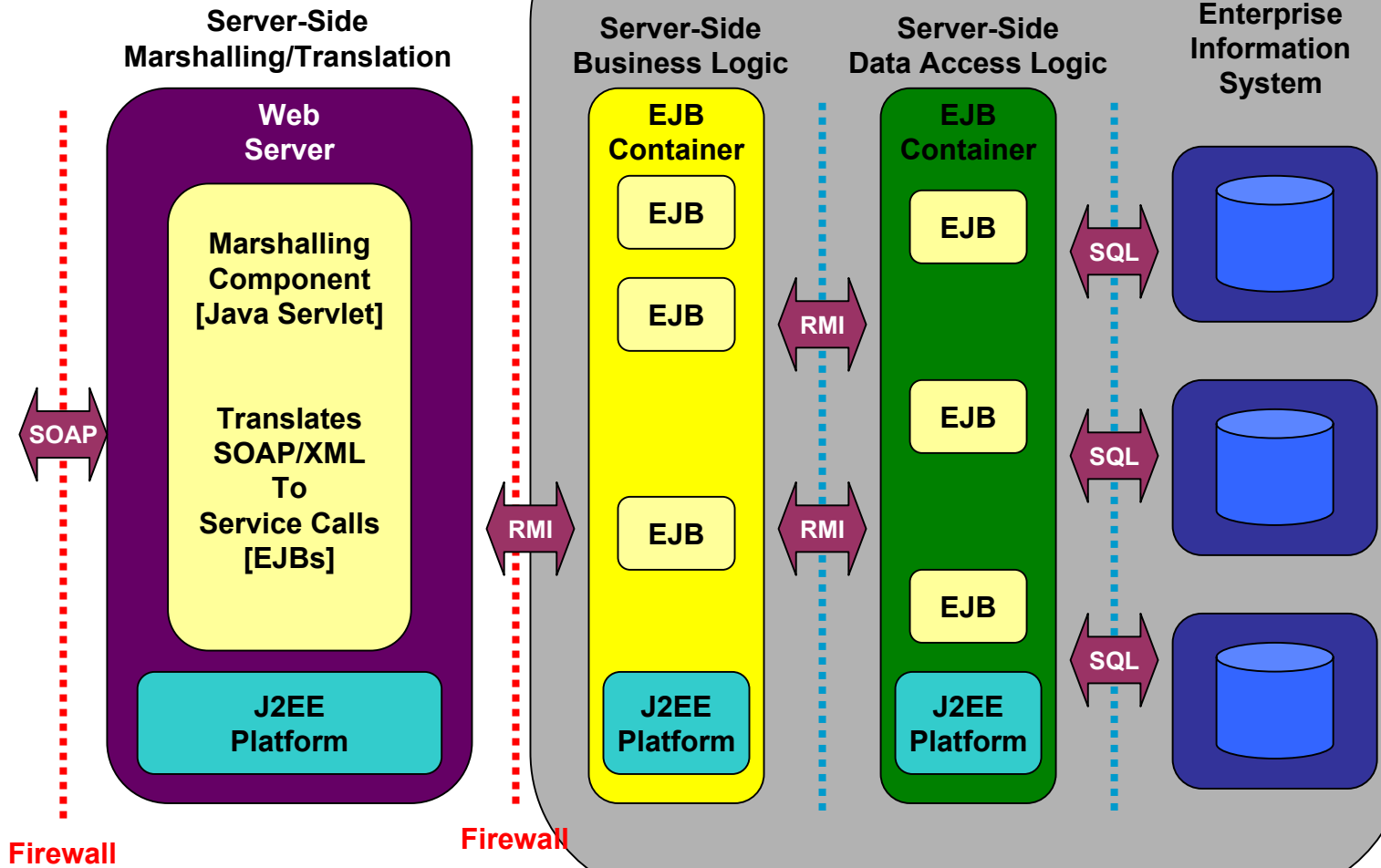


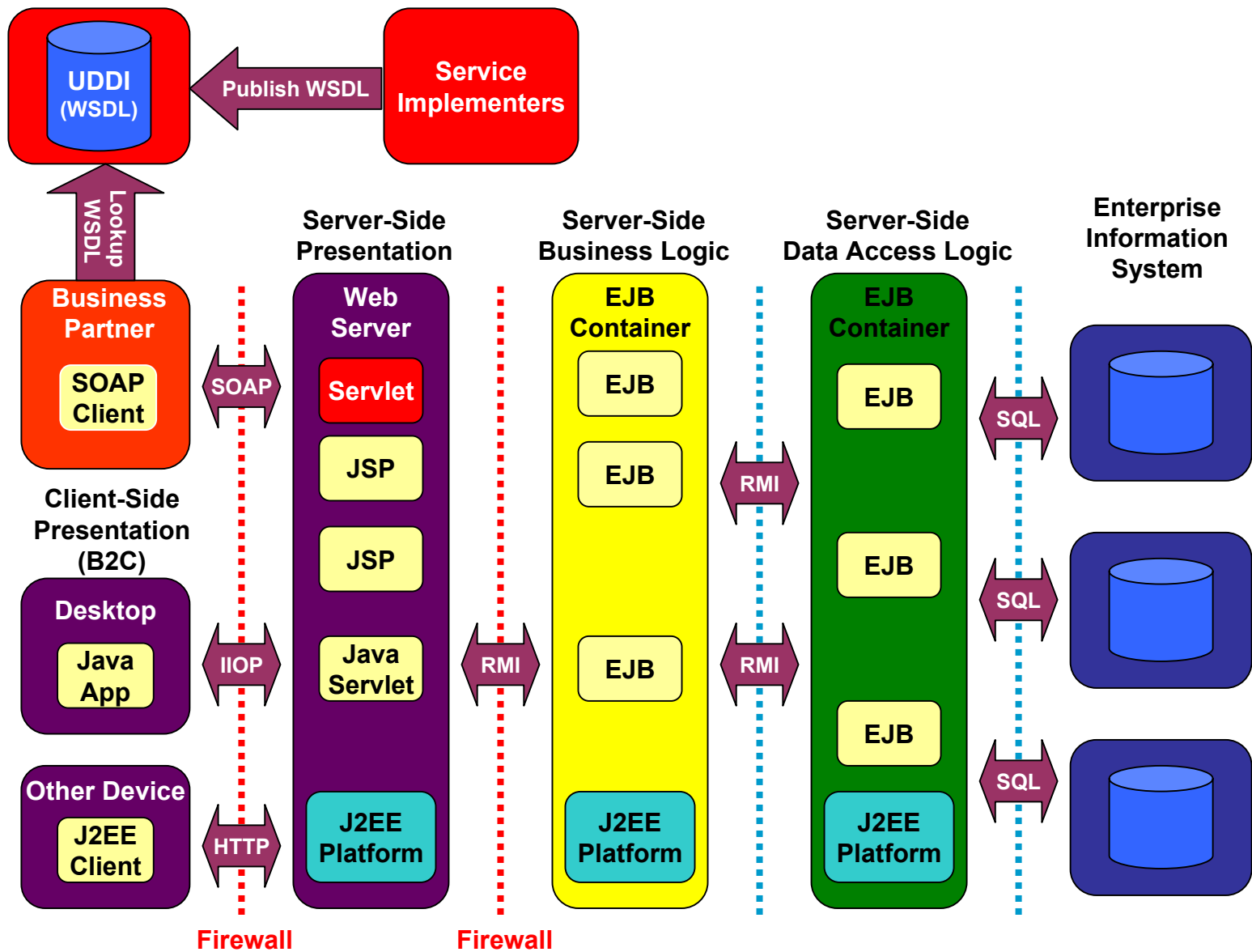
# Web Services Architecture

- Server Side Processing
  - SOAP request received over HTTP
  - Web Server delegates request to Servlet engine
  - Servlet engine invokes the data marshalling component.
  - Marshalling component invokes service component (Java Object or EJB)
  - Service returns result that is translated back into SOAP XML and returned to client



# Service Implementation







# Financial Services Engine

## A Case Study



# Financial Services Engine

- The building block for financial transactions

**Financial Services Engine**

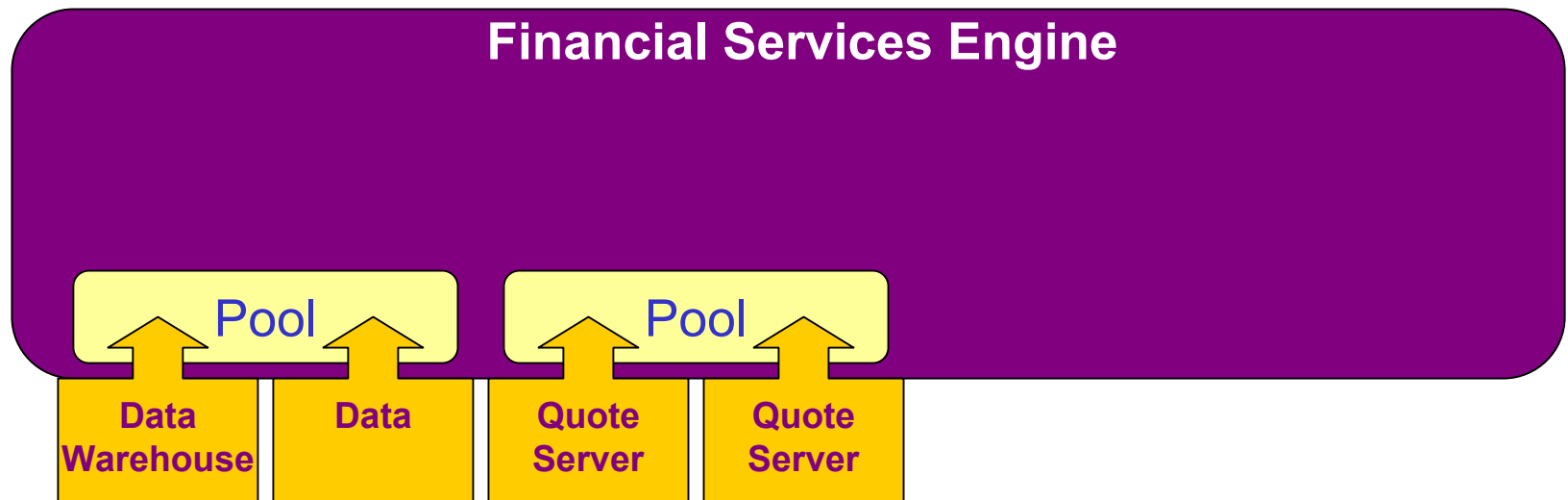
# Financial Services Engine

- Integrates with existing financial systems



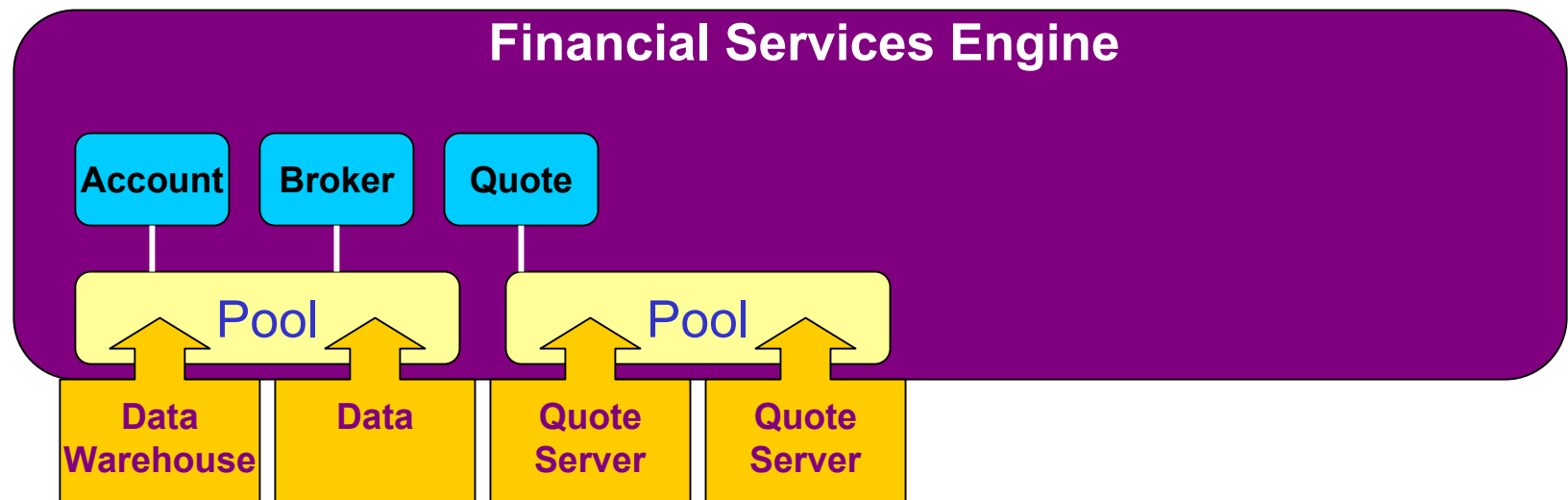
# Financial Services Engine

- Resource pooling provides scalability and redundancy



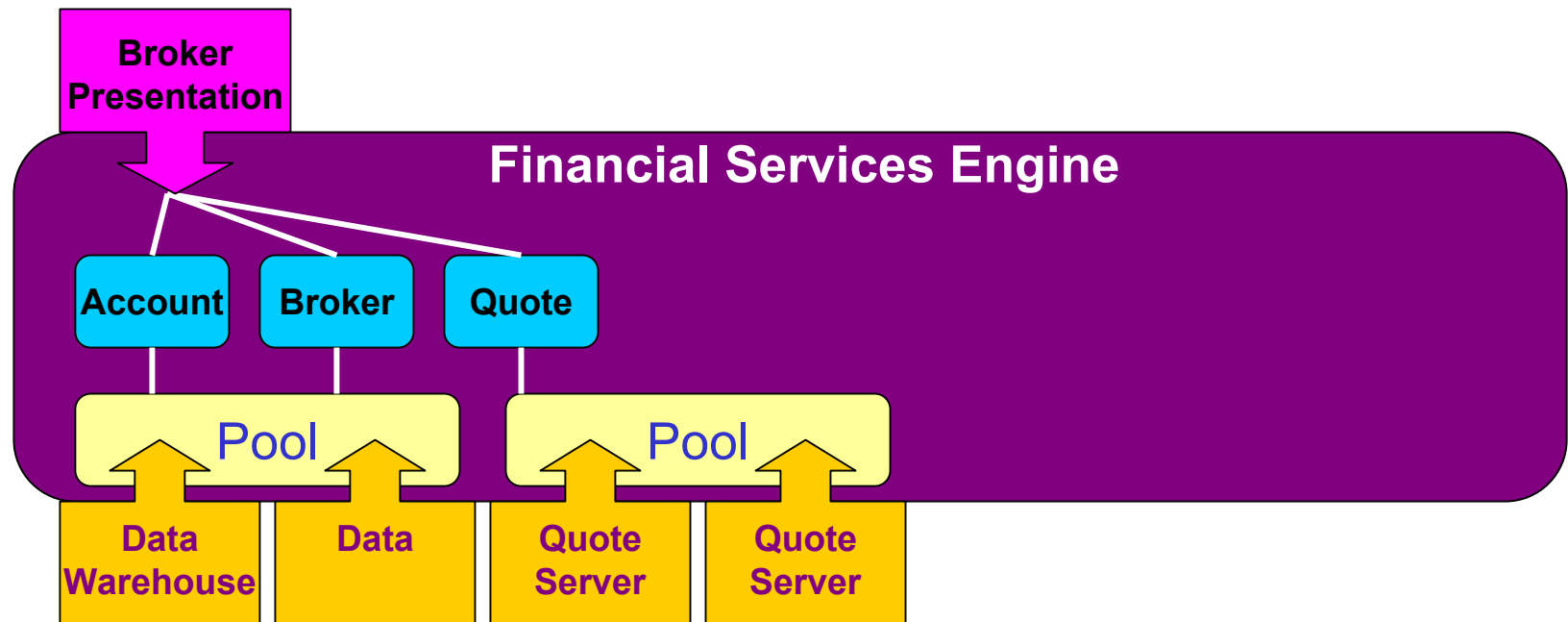
# Financial Services Engine

- Reusable components and objects model the business domain



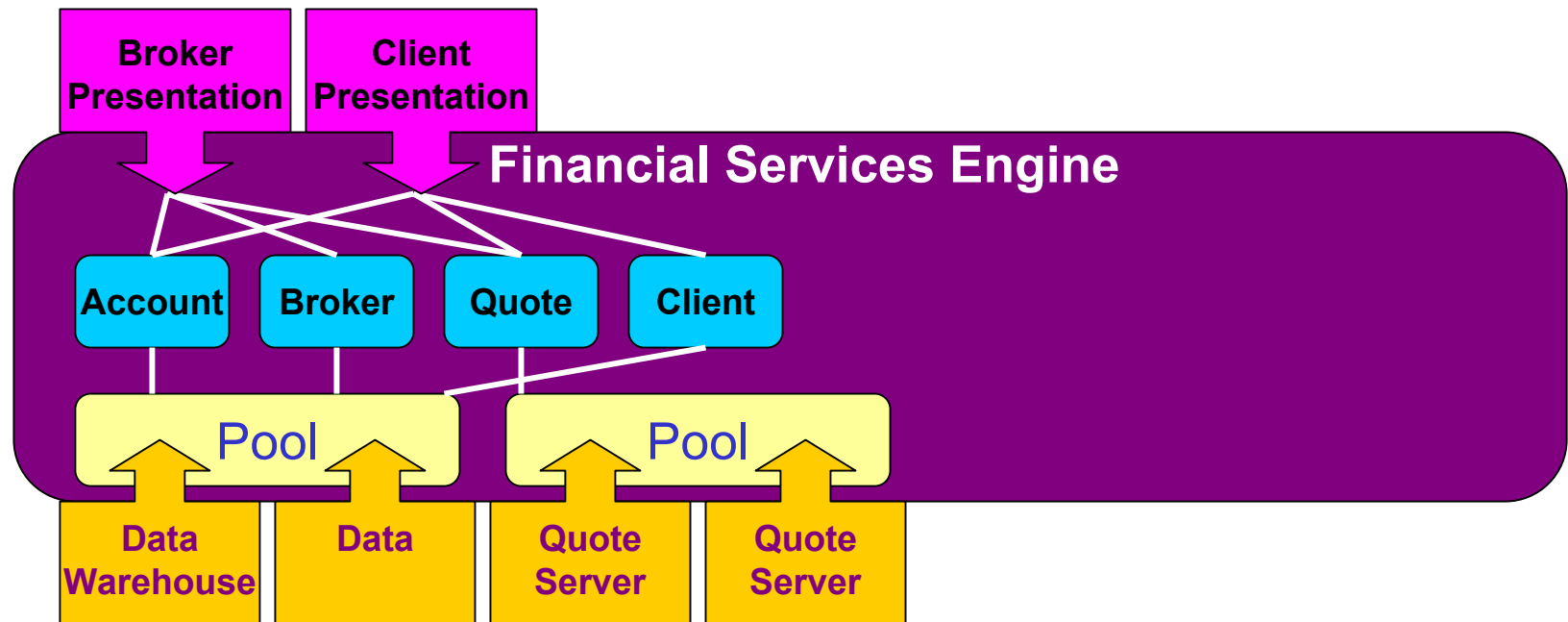
# Financial Services Engine

- The engine separates the business logic from the presentation logic allowing for greater reusability



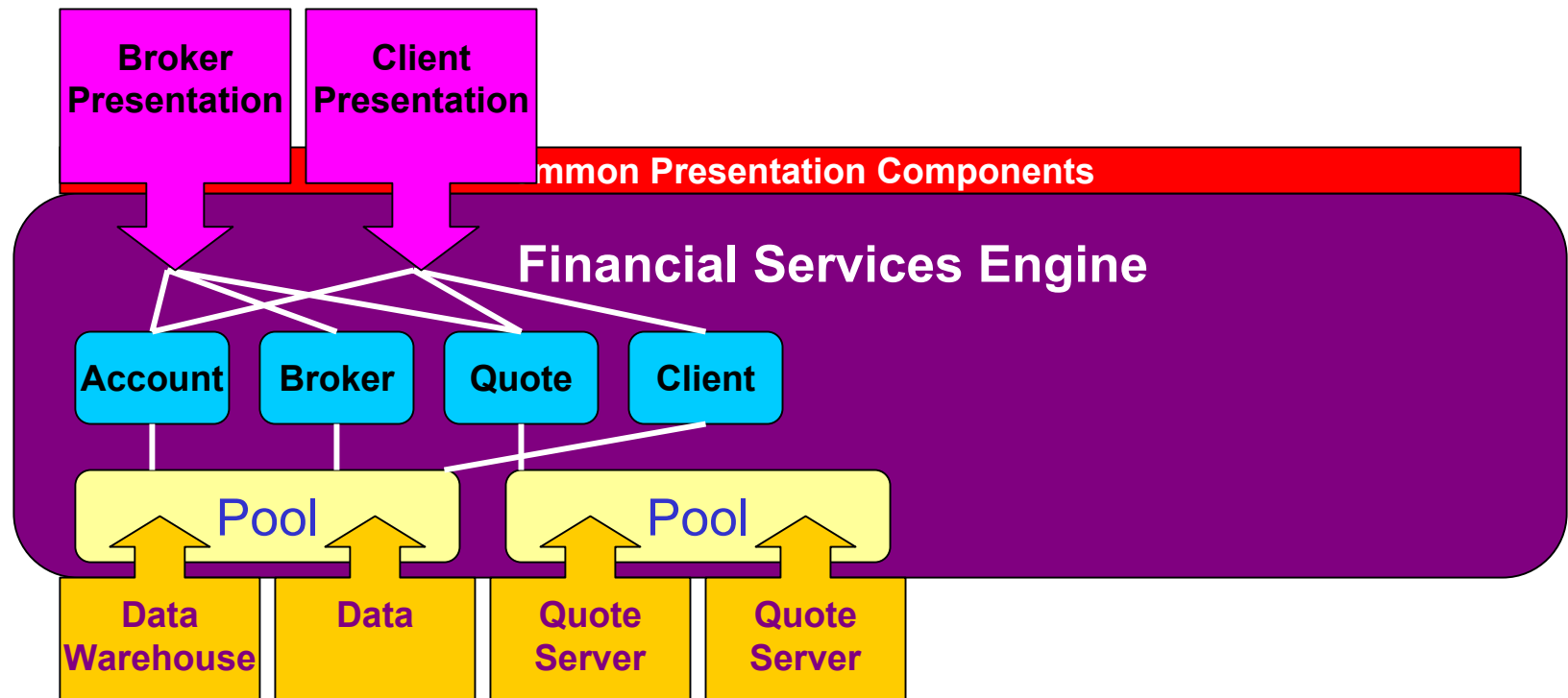
# Financial Services Engine

- As presentation requirements evolve, additional components can be easily integrated



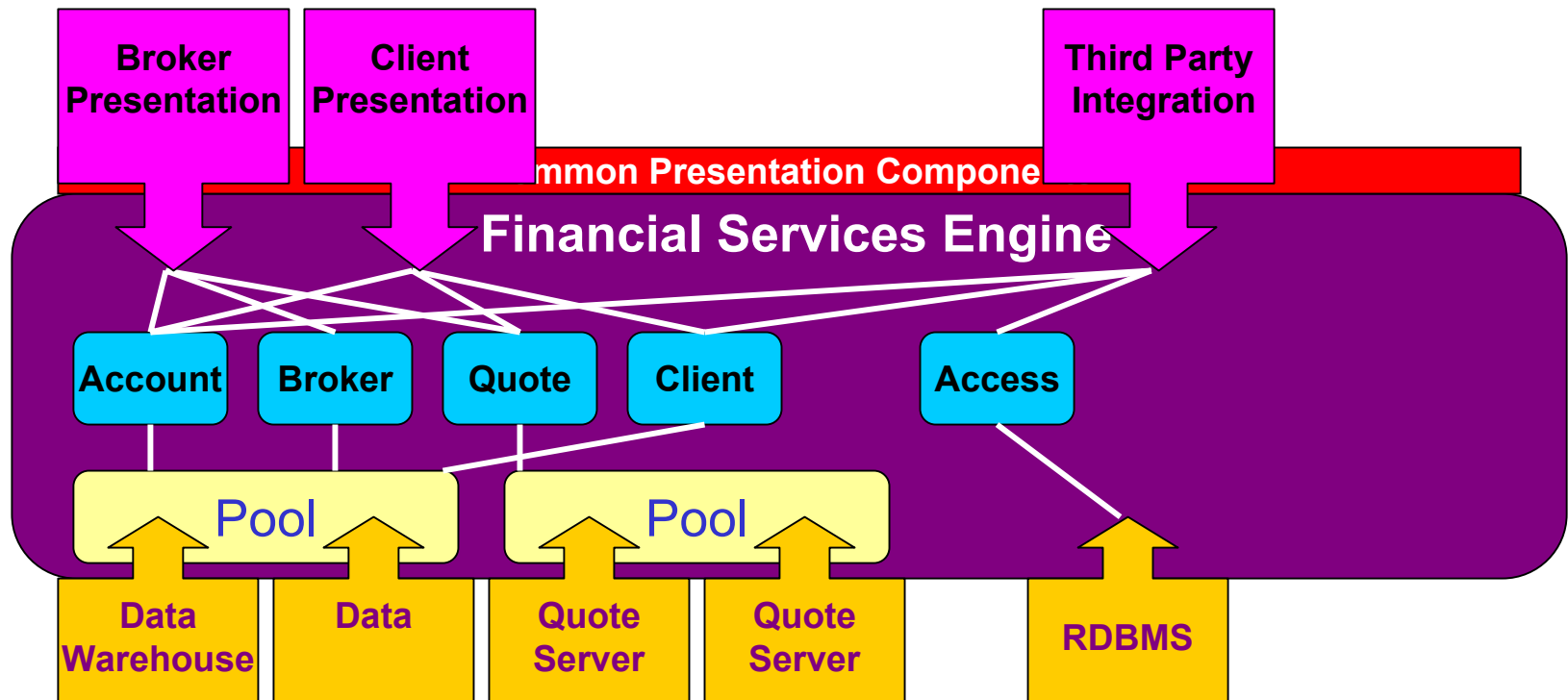
# Financial Services Engine

- Reuse occurs on the presentation side as well with common presentation components



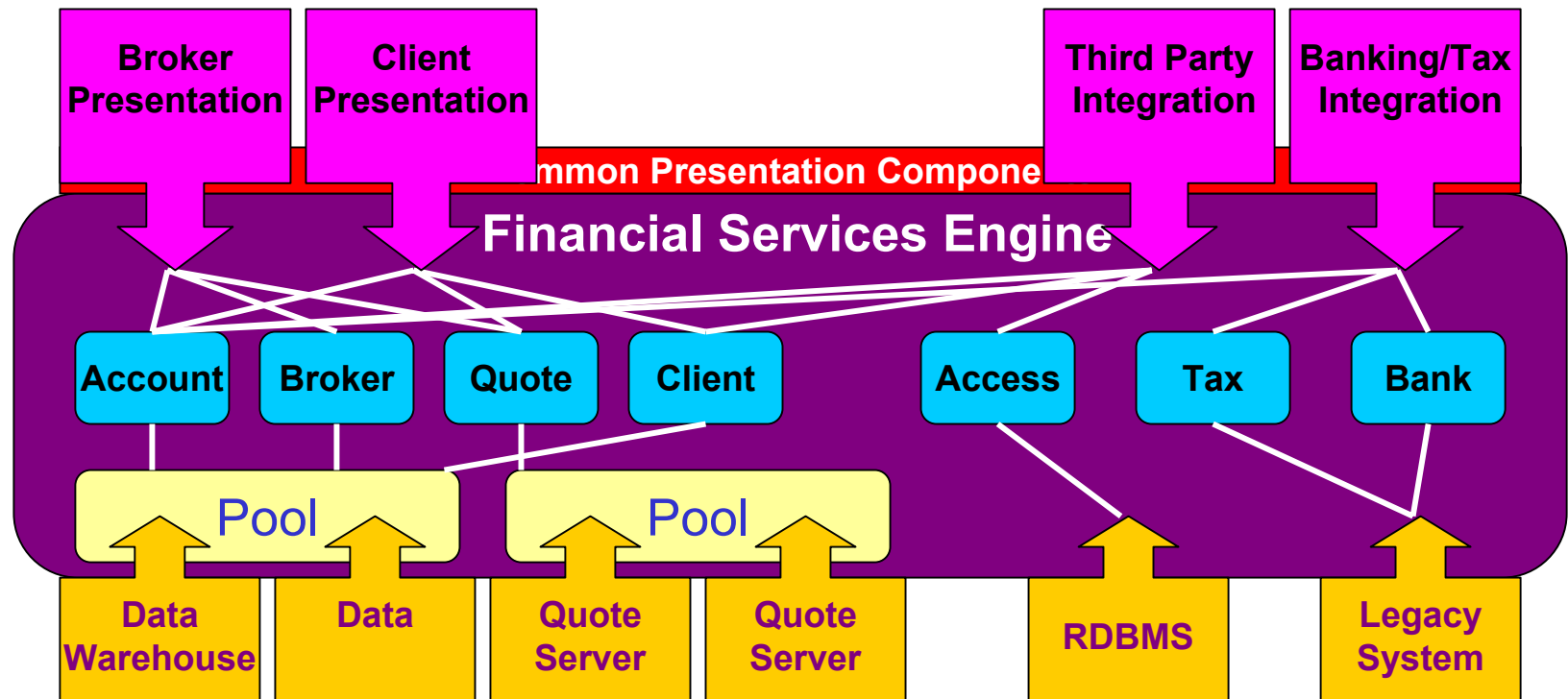
# Financial Services Engine

- Integration with other products is simplified by reusing existing components



# Financial Services Engine

- Integration with legacy systems also becomes simplified



# FSE Summary

- Separates business logic from presentation logic
- Represents your domain
- Promotes reusability
- Integration with other systems is simplified

# Web Services Best Practices

- Minimize the number of Internet round-trips
- Model services as a large operation
  - Model services as stateless
  - Transaction only lasts during a single call

# Web Services Summary

- Uses existing infrastructure
- Interoperability
- Allows you to build services first and find partners later



# Thank-You