



IBM Software Group

# Application Servers as Virtualization Environments

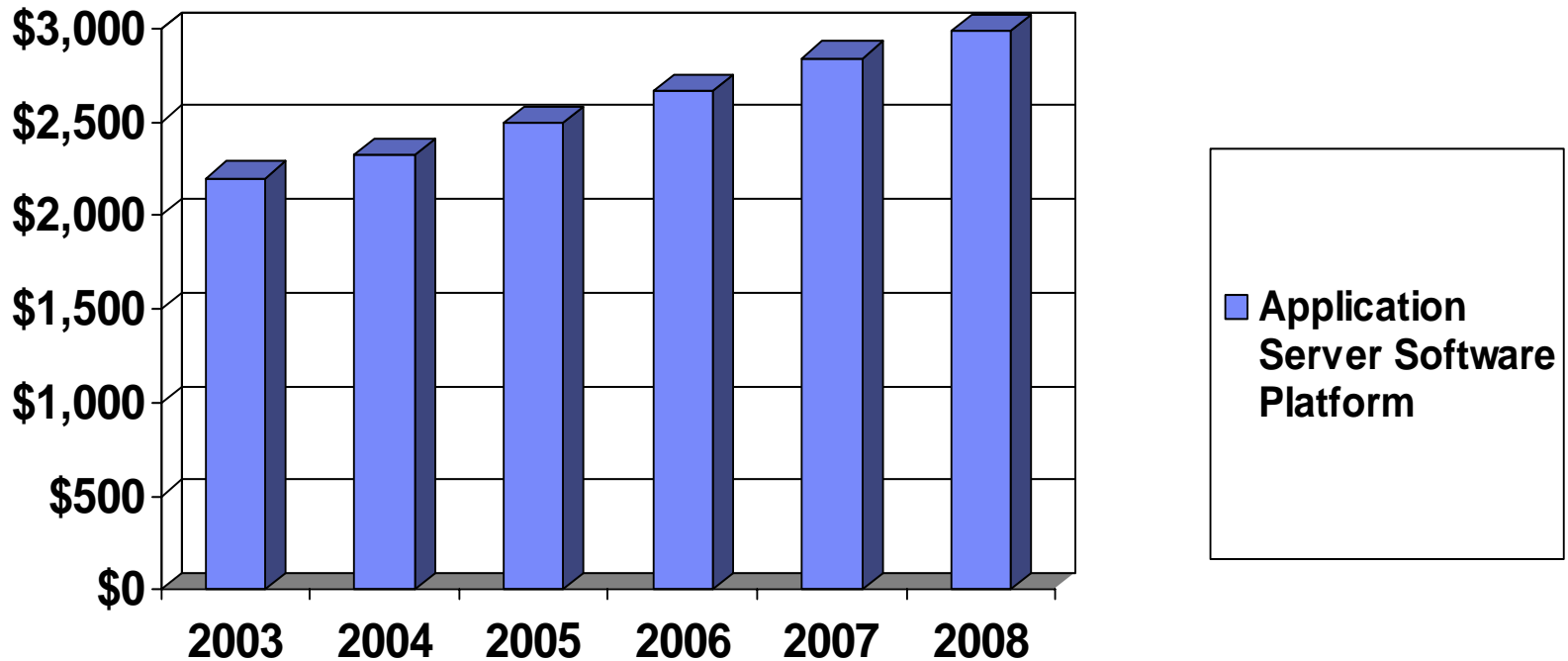
Martin Nally, CTO, IBM Rational

**WebSphere** software



@business on demand software

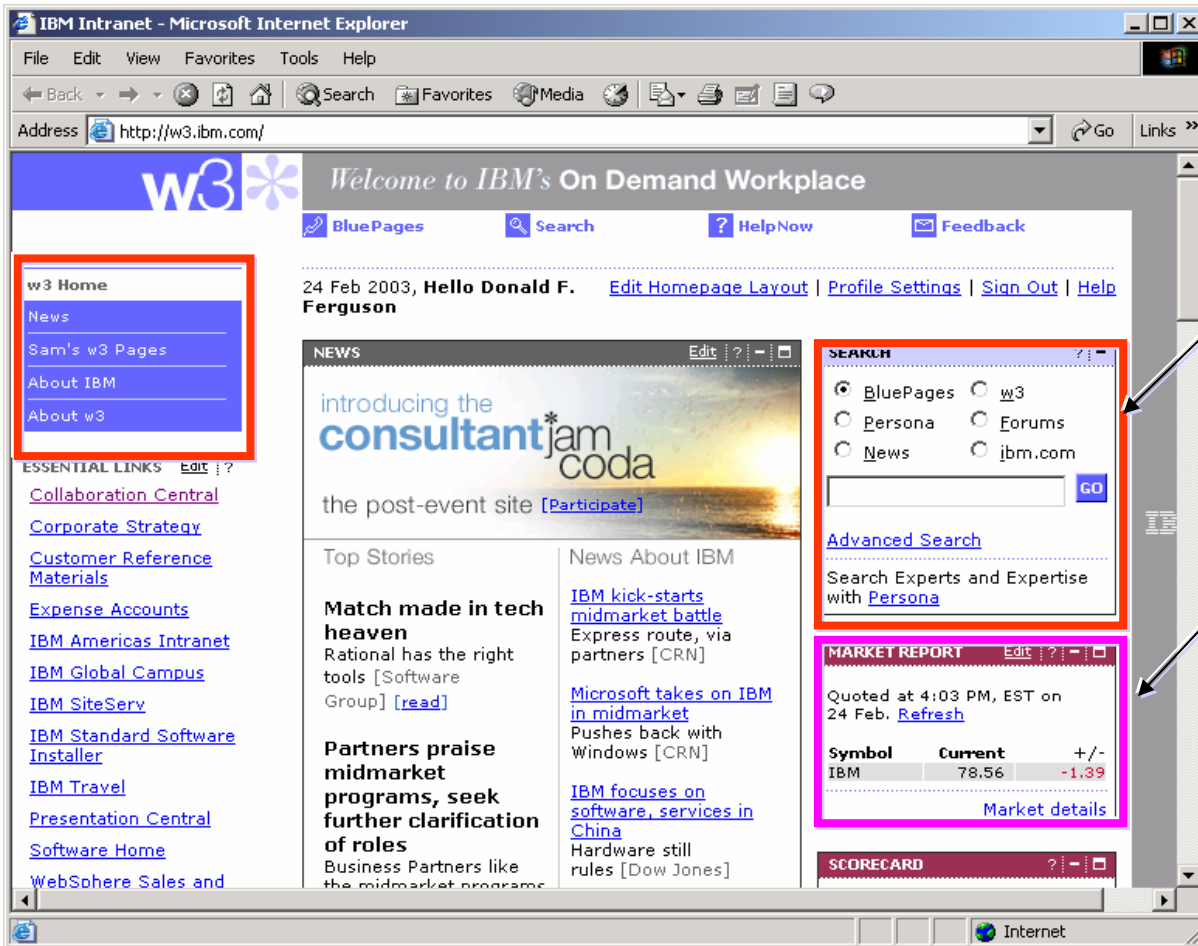
# Worldwide Application Server Software Platform Revenue Opportunity Forecast (US \$Millions)



Source: 2004 Worldwide Software Market Forecaster, IDC Corp., Sept. 28, 2004



# What sorts of Applications run on Application Servers?



## Personalize

- ▶ The site tells you
- ▶ Who you are?
- ▶ What they have?
- ▶ Marketing rules?

## Customize

- ▶ You tell the site
- ▶ What you want

## WCM

- ▶ Annotate content
- ▶ Pages, "tiles"
- ▶ Data, renders
- ▶ Structure assoc.
- ▶ Project management

## Writing a Web application without an application server

- Suppose I need to develop an application that services requests from many users, performs calculations, and accesses and updates data. Given the capabilities provided by the OS and JVM, here is an obvious possible architecture:
  - ▶ A simple application that accepts inputs from one user
  - ▶ One OS-level copy of the application for each user (because the application is stateful)
    - One copy of the Java VM for each application if the application is a Java application



# Writing a Web application without an application server

- Pros:
  - ▶ Exploits the OS support for data and code isolation, security
  - ▶ Application code is relatively simple and straightforward
  - ▶ Scales linearly with the number of users
    - To a point – contention for resource connections will be a problem
- Cons:
  - ▶ Inefficient
    - Each copy of the application will consume considerable memory resources, even though it is mostly idle.
    - If these application copies each consume resources, such as database connections, the solution will not scale
    - If an application “instance” is paged out by the OS, the time to restart (page in) is long
  - ▶ Hard to manage
    - How are these applications copies started, stopped and monitored?
    - How are these application copies allocated to machines?
      - One per machine?
      - Many on one machine?
    - How are these applications deployed across many machines?
    - How is user authorization for access to resources administered?
  - ▶ Gaps
    - How are user requests routed to the correct application copy to support “conversations” or “sessions”?



# Alternative architecture

- Write an application where each copy handles requests from multiple users
- Pros:
  - ▶ Potentially better use of resources and therefore better scaling
- Cons:
  - ▶ Complexity:
    - Serialize the processing of requests? - Doesn't scale
    - Spawn threads? Pool connections? - Complex to write
    - Protocol gateways (HTTP, messaging, ...)
    - Replication of code and data across machines
    - Logging for serviceability
    - Security
    - User access authorization for resources?
- If you do a really good job you will have implemented a whole application server inside each application
- This basically is the approach taken by application servers.
  - ▶ The twist is that the application server provides support for the repetitive stuff – the business application programmer “plugs-in” the business-specific logic



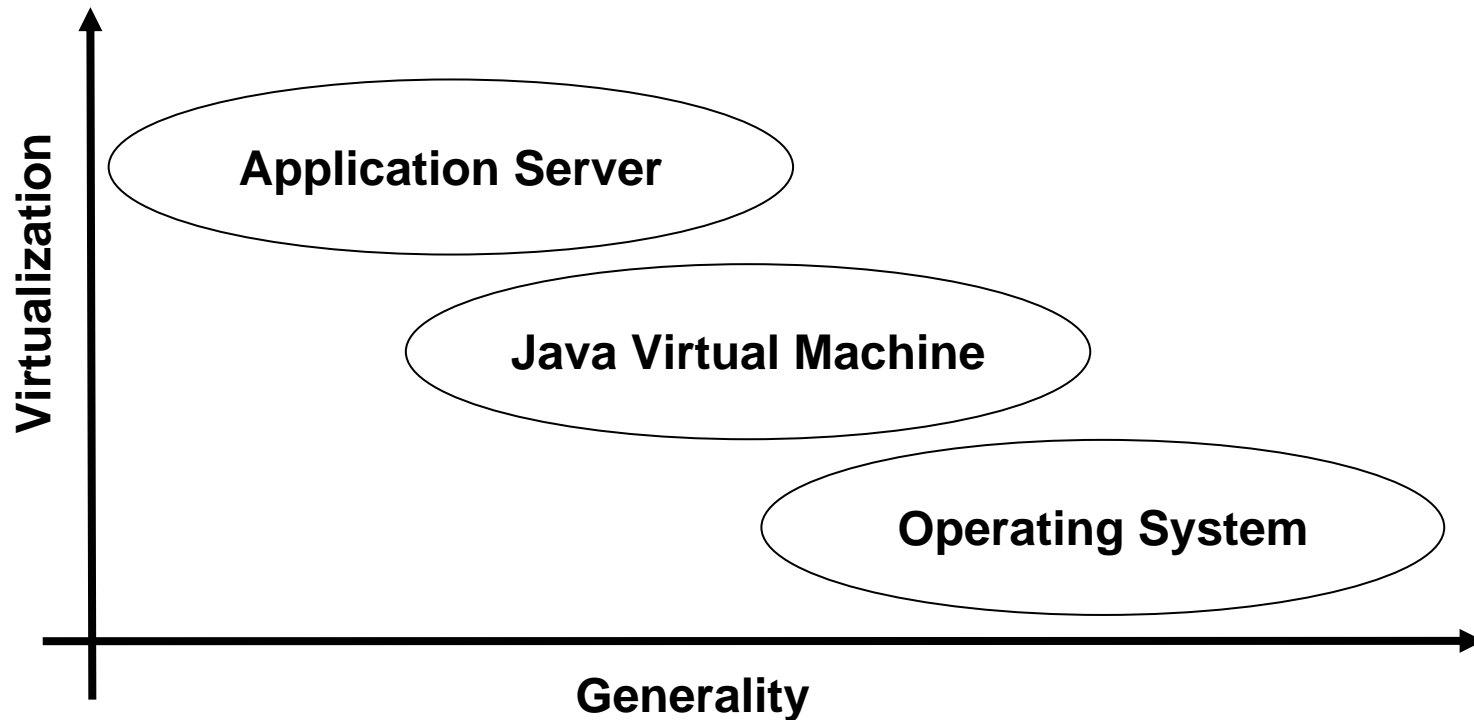
## 3rd architecture – CGI-BIN

- Use a web server to handle incoming HTTP requests
- For each “dynamic” request, the Web Server will spawn a normal OS process
  
- Pros:
  - ▶ Can write handler in any language, however you like
- Cons:
  - ▶ Spawning a new OS process for every request is expensive
  - ▶ Does not provide any support for scaling, connections, logging, or any of the other things application servers provide
    - Again can start writing this yourself



# Levels of Virtualization

- By providing support for particular application patterns, an application server can provide a higher degree of virtualization



# Levels of Virtualization – Operating Systems and JVMs

- Operating systems provide important general concepts to support application execution
  - ▶ Very general – most application code runs on some sort of OS
  - ▶ Basic function
    - OS Processes isolate data and code between applications
    - OS abstractions for resources like files and database connections
    - Virtual memory is supported
    - Some security support is provided (files, connections)
- Java Virtual machines provide some additional virtualization
  - ▶ Some applications cannot tolerate overhead of VM environment ?
  - ▶ Some more advanced function
    - Memory garbage collection
    - Portability of code and abstractions for resource access (databases, files, GUI, ...)
    - Some additional security enforcements



## Levels of Virtualization: Application Server Generality

- Application Servers target certain classes of application
  - ▶ Primarily B2C or B2B internet or intranet applications
  - ▶ Many users submitting business requests to an application that implements business logic and provides access to shared resources such as files and databases
  - ▶ In general, application servers do not provide value for all categories of application (e.g. video games, desktop utilities, device drivers, telemetry applications)
  - ▶ There are interesting exceptions – application servers have been extended to other problems (such as batch processing) and have been used in creative contexts (e.g. cars)



# Levels of Virtualization: Application Server Virtualization

- Application servers provide a virtual environment in which applications execute that maintains the following fictions:
  - ▶ There is only one user
  - ▶ There is no contention for resources
  - ▶ Users only try do things they are allowed to do
  - ▶ Data is magically persistent
    - Optional feature
  - ▶ Hardware is infinitely scaleable
  - ▶ Hardware never fails
    - Replication of state
- Application servers provide a virtual environment in which applications can be managed and monitored
  - ▶ Multiple applications/users serviced from a single JVM
    - Isolation is maintained (application code and user data)
  - ▶ Pools of machines can be used to run copies of an application
    - Multiple applications on one machine (e.g. to exploit affinities to each other or to data)
    - Applications spread over multiple machines (for scaling)
  - ▶ Pools of threads can support many users
  - ▶ Pools of connections can support scaling of access to shared resources
  - ▶ Hardware failures can be recovered either automatically or manually
  - ▶ Application servers have sophisticated management consoles.



# The Role of the Container

Security Header  
 Reliable Messaging Header  
 Atomic Transaction Header



SOAP Message



Policy Declarations



The Impl.

This is fragile,  
 changes over time,  
 complex for business programmers,  
 error prone,  
 etc.

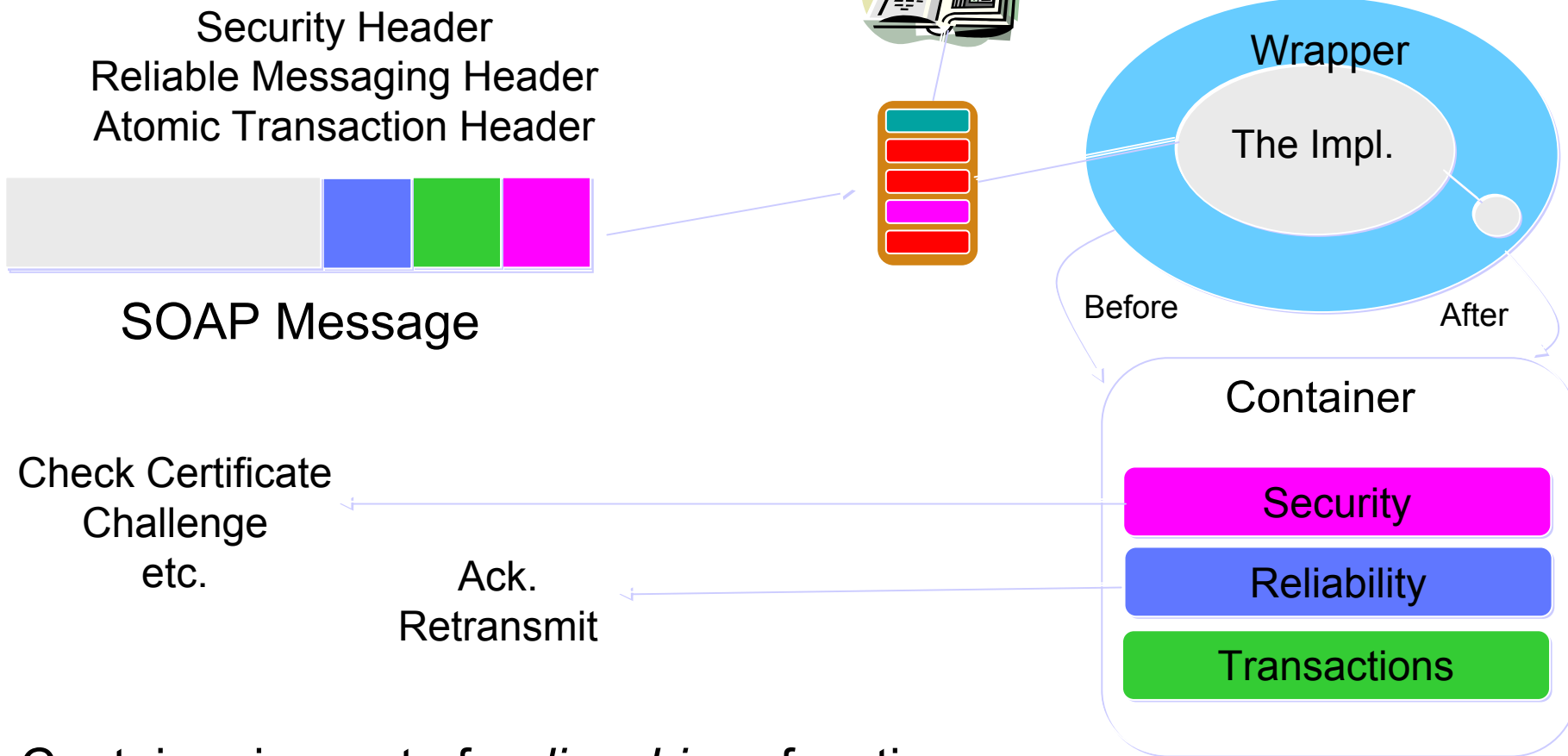
```
double deposit(Message m) {
    checkForDuplicate(m.seqNo);
    registerForTransaction(m.context);
    isCAValid(m);
    checkSignature(m);
    updatePerformanceInfo();

    balance += m.amount;

    // ... ..
    updatePerformanceInfo();
}
```



# The Role of the Container



Container is a set of *policy driven* functions.  
Interceptor pattern for business logic and “stubs.”  
Before and After factoring of code.

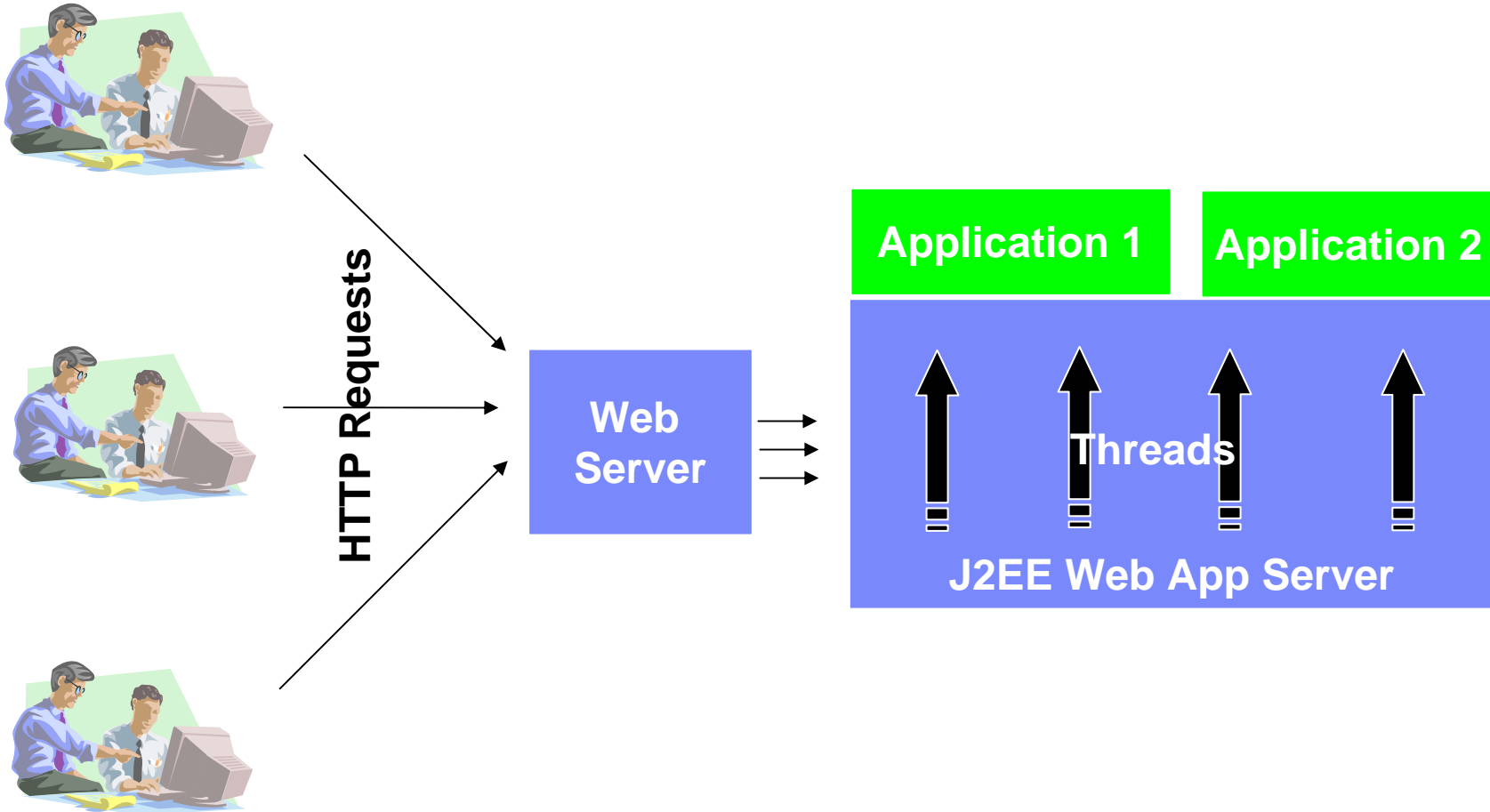


# Implementing a simple web application - Servlets

- In a J2EE web application server, HTTP requests whose responses must be calculated dynamically are handled by servlets.
- Servlets are Java classes that are written by application developers and are called by the web application server to handle HTTP requests and format responses.
- The servlet developer is not responsible for writing a “daemon” that listens for incoming HTTP requests – this is part of the web application server.
- The servlet developer is not responsible for creating the execution threads on which the application executes.



# Implementing a simple web application



# Implementing a simple web application

```
package com.ibm.sampleservlet.vee;
```

```
import java.io.IOException;
import javax.servlet.Servlet;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```
public class VEEServlet extends HttpServlet implements Servlet {
    public VEEServlet() {
        super();
    }
```

```
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
        ServletException, IOException {
        response.getOutputStream().print("Hello");
    }
```

```
}
```

- Ignoring a little “boilerplate”, the single line of code in red is all that is required to write a “hello world” application that will run on an application server, respond to HTTP requests from browsers, and scale linearly to an arbitrary number of users by adding hardware



# Implementing a simple web application

- Servlets are stateless, like the HTTP protocol itself
- Programmers have to be aware that the “service” method of the same servlet instance may be executing on many threads concurrently, servicing requests from many users
  - ▶ Application server may use a single instance or may maintain a pool
  - ▶ Application programmers cannot store state in instance variables between different requests – this is different from “normal” OO programming
- This is a problem, because useful applications require state to be maintained between requests
  - ▶ What is in my shopping cart?



# HTTP Session State – a virtual stateful environment

- Application servers support the notion of “session”
  - ▶ A grouping of requests from a particular user across which state may be retained
- “Many strategies for session tracking have evolved over time, but all are difficult or troublesome for the programmer to use directly.” (J2EE Servlet specification)
  - ▶ Cookies – a data value sent by the server to the client that is transmitted back to the server by the client with each request
    - Cookies transmitted in HTTP headers
  - ▶ URL rewriting – adding extra information to the “submitted” URL that carries a session ID
    - <http://www.myserver.com/catalog/index.html;jsessionid=1234>
- Fortunately, Application servers automate the “troublesome” part of this



## HTTP Session State – a virtual stateful environment

- J2EE web application servers provide each application a “virtual stateful environment” where state can be simply stored during processing of one request and retrieved and modified during processing of a subsequent
  - ▶ The application executes in the context of a stateful session
  - ▶ Application servers handle the mechanics of session tracking for the application (cookies)
    - URL rewriting requires the programmer to be involved ☹
      - `<A HREF='`
        - `<%=response.encodeURL("/store/catalog")%>`
      - `'>Catalog</A>`
  - ▶ Replication of session state across clusters supports recovery from hardware failures



## HTTP Session State – a virtual stateful environment

```
protected void doGet(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException {
    if (request.getSession().getAttribute("beenHere") == null) {
        response.getOutputStream().print("Hello");
        request.getSession().setAttribute("beenHere", new Object());
    } else
        response.getOutputStream().print("Hello again");
}
```

- The code in green shows how to query the virtual stateful session associated with the request by the application server
- The code in red shows how to update the virtual stateful session associated with the request by the application server
- The application server will automatically maintain the state of the session (subject to configurable timeout parameters)
- The application server will ensure that successive requests in the session are routed to the machine that has the correct session states: "session affinity"
- Sophisticated application servers may replicate session state over multiple machines so that, in the case of a hardware failure, another machine may service subsequent requests of a session without interruption of service.



## Web Application container – a virtual secure environment

- J2EE web application servers support a model of security that is transparent to application developers
  - ▶ Programmatic security is also supported
- From an application programmer's perspective, the application runs in a virtual world where users only do things they are permitted to do
  - ▶ Application programmers do not need to code checks for security permissions – these checks are made by the application server
- The security enforced by the application server container is parameterized by declarative “rules” written in XML in a “deployment descriptor”
- Users are assigned to roles and rules define which roles have access to which capabilities
  - ▶ Usually users are defined in something like an LDAP server and associated with groups, like manager, supervisor, gold club member, frequent flier, ...
- J2EE security is role and resource based
  - ▶ Making sure that only John and his manager can see John's personnel records still requires programming, since “records” are below the resource level.



## Web Application container – a virtual secure environment

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>retail</web-resource-name>
    <url-pattern>/acme/retail/*</url-pattern>
    <http-method>GET</http-method>
    <http-method>POST</http-method>
  </web-resource-collection>
  <auth-constraint>
    <role-name>CONTRACTOR</role-name>
    <role-name>HOMEOWNER</role-name>
  </auth-constraint>
</security-constraint>
```

- In this example, only contractors and homeowners can access urls of the form “/acme/retail/\*”
- The application server will associate a role with an incoming request
  - ▶ Clients can authenticate using various forms of authentication (HTTP Basic Authentication, HTTP Digest Authentication, HTTPS Client Authentication, Form Based Authentication)
- The application server will check authorization constraints and will return an error if they are violated

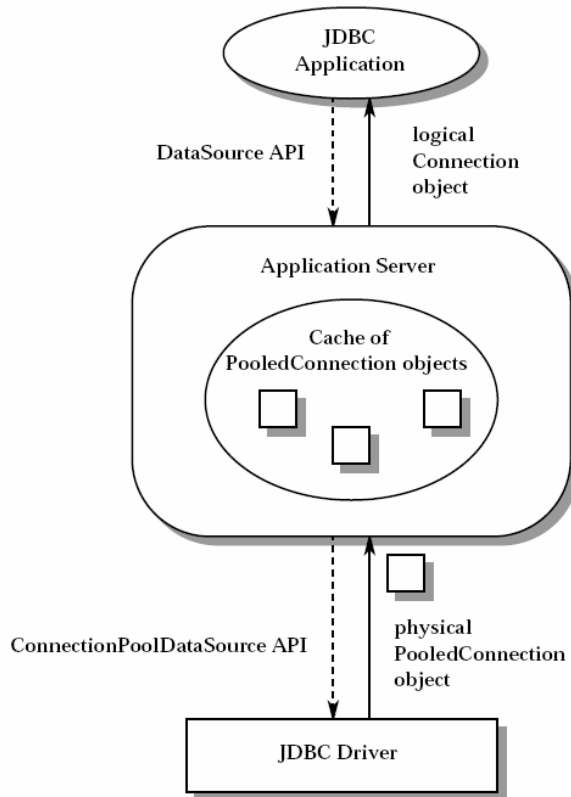


# Connection Pooling - Scalability and performance

- J2EE application servers provide a virtual world in which access to resources like databases and back-end systems is limitless
  - ▶ Applications on J2EE application servers are designed to scale to thousands or even 100s of thousands of concurrent users. Databases and other back-end resources are not designed to support 100s of thousands of concurrent connections.
  - ▶ Even if enough concurrent connections were supported, it takes significant time (including round-trip messages to a back-end datasource) to establish a new connection with the correct user-code, password etc. associated with it.
- J2EE application servers implement “connection pooling”
  - ▶ Applications request a connection
  - ▶ The server provides a connection from a pool
    - May require establishing a new connection to a resource, or there may be a suitable available connection in the pool (e.g. with matching user-code/password)
    - “Credential mapping” identifies the correct userid and password for each user
  - ▶ All application requests in the same transaction for the same data-source are automatically allocated the same connection
  - ▶ C.f. memory management in a JVM with garbage collection.
    - JVM understands cost trade-offs of memory, App server understands cost-trade-offs of connections
- Targets of connections are “virtual” – applications can be redirected to connect to different targets without modifying code
  - ▶ Redirect from a test database to a production database



# Connection pooling



# Using Connections

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    try {
        Context ctx = new InitialContext();
        DataSource ds = (DataSource)ctx.lookup("java:comp/env/VEEServlet_Con1");
        Connection con = ds.getConnection(); // automatic login
        PreparedStatement ps =
            con.prepareStatement("SELECT DEPTNAME FROM DEPARTMENT");
        ResultSet rs = ps.getResultSet();
        while (rs.next()) {
            response.getOutputStream().print(rs.getString("DEPTNAME"));
        }
    } catch (NamingException e) {
        e.printStackTrace();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

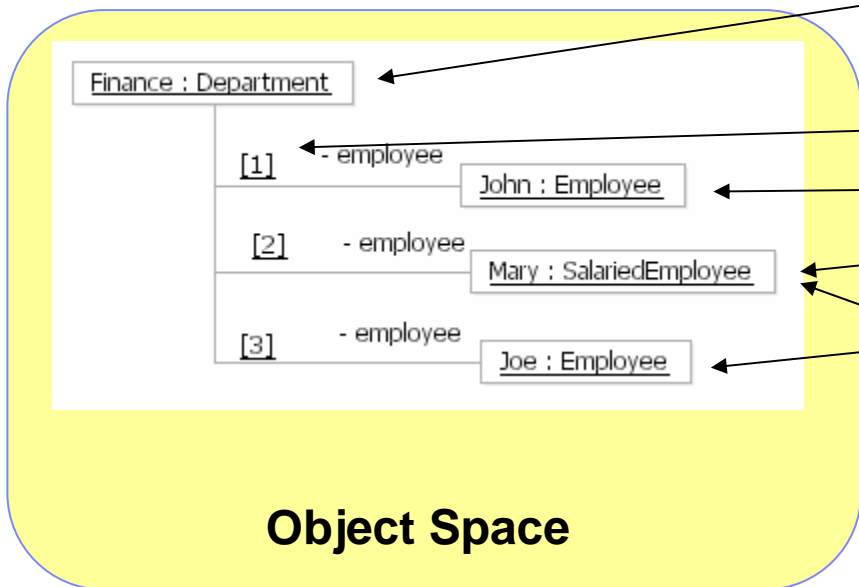
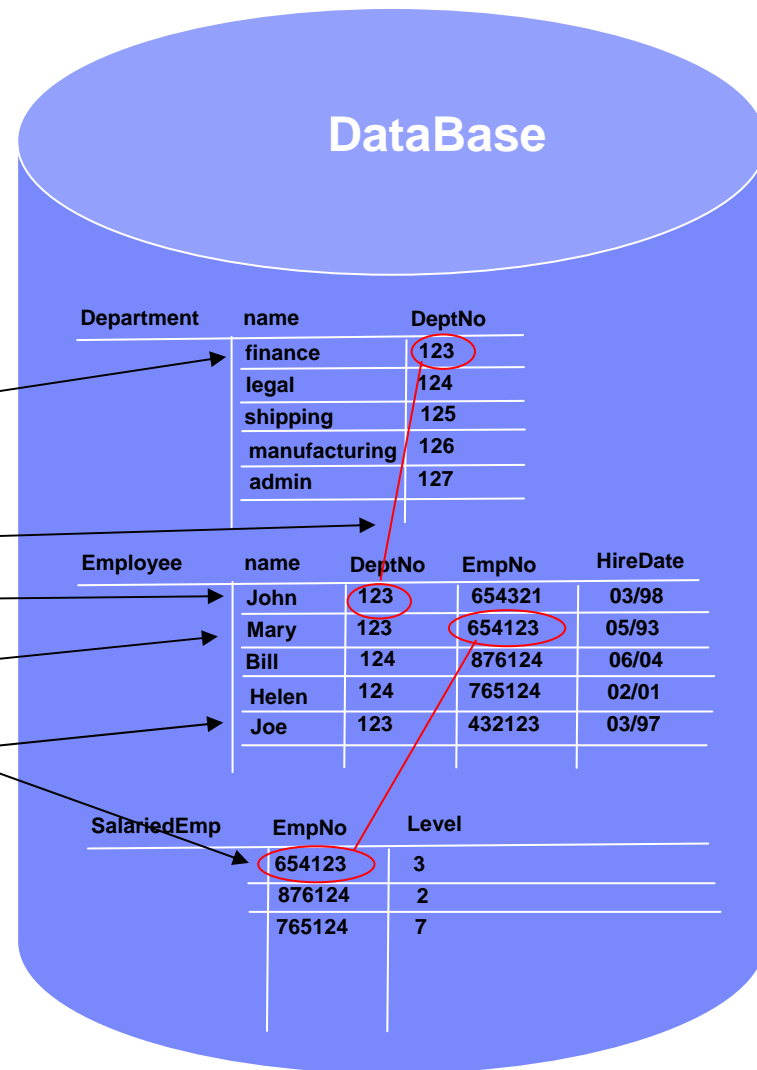
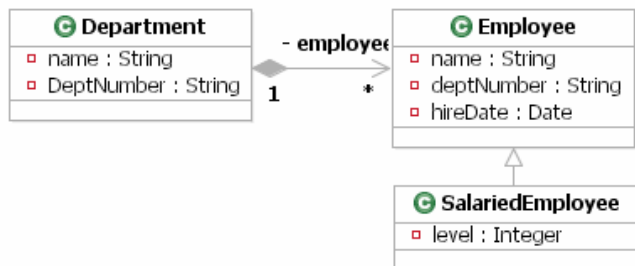


# Entity EJBs – virtualizing persistent data

- In the previous example, we showed how an application can access relational data using a virtualized database connection
- J2EE Application servers support a further level of virtualization of data access
- Applications do not code access to data resources directly. Instead they define:
  - ▶ A “logical” object model for the data
  - ▶ A mapping of the “logical” object model onto the underlying data model
    - Extensions allow mapping onto CRUD methods for non-relational data
- Application then interact with instances of this logical model
- The “virtual fiction” is that all data is in memory all the time
  - ▶ The container does on-demand retrieval of the data from the database
  - ▶ The container will automatically commit changes to the logical object model back to the database, usually when the transaction commits



# Entity EJBs – virtualizing persistent data



**Object Space**

# Maintaining Code and Data Isolation

- J2EE application servers provide their own form of application code isolation
  - ▶ Each application in a server is assigned a “class-loader”
    - Class-loaders form trees
  - ▶ This allows new version of an application to be installed without affecting other running applications in the same application server
- J2EE application servers provide their own forms of data isolation
  - ▶ Session state is kept on a per user, per session basis – applications cannot accidentally access session information for the wrong user
  - ▶ Credential mapping ensures that users access shared data under a suitable userID for their privilege levels.
    - You don’t (typically) authorize each user individually to each database/table/back-end application

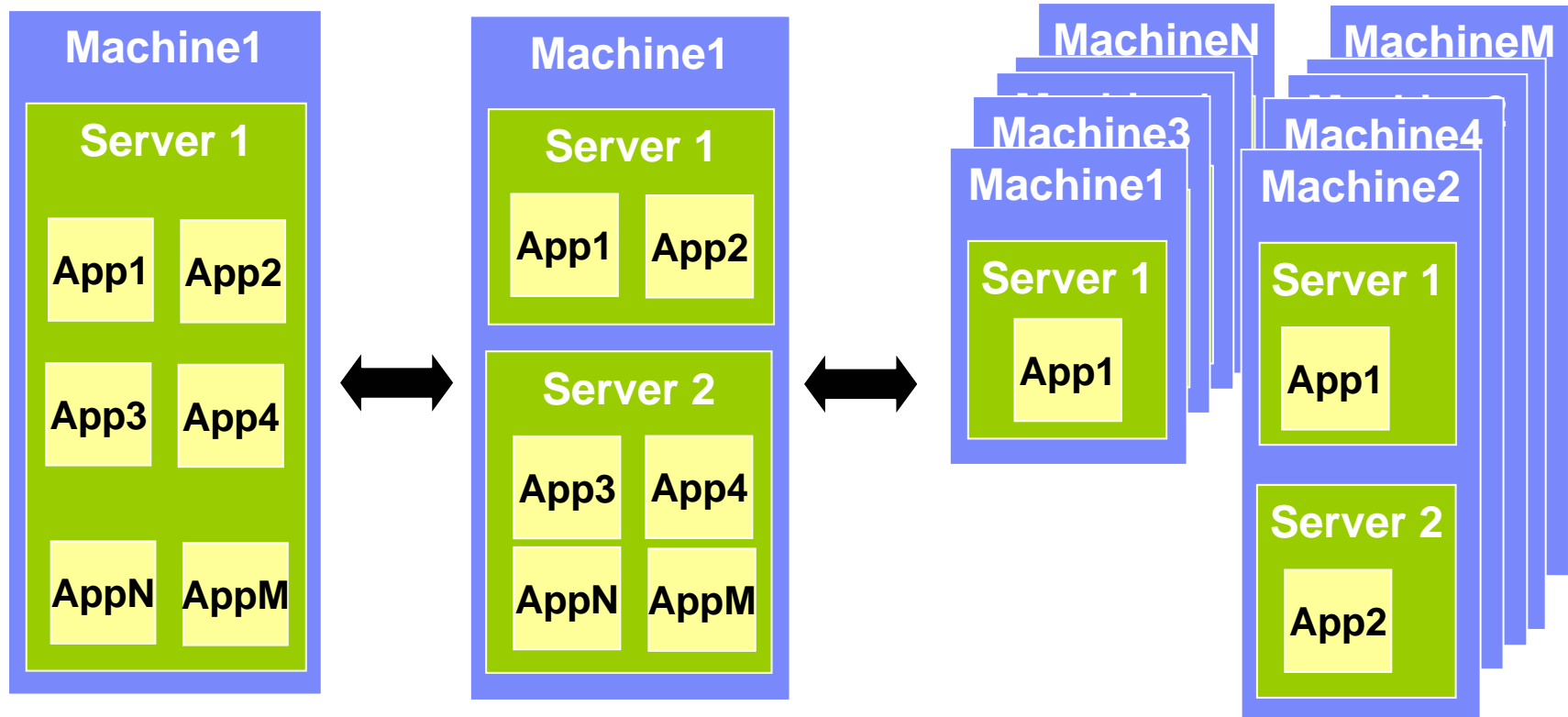


## Clusters and work-load management: Virtualizing Hardware

- One of the key features of application servers is scaling
  - ▶ If you need more capacity, add more hardware
- One of the challenges of having more hardware (and processes running on the hardware) is managing it all
  - ▶ If you can't deal with the management issues of adding machines, then virtualizing doesn't help.
- WebSphere Application Server (WAS) supports **Clusters**
  - ▶ Clusters are groups of machines running identical application software
  - ▶ WebSphere performs work-load management and fail-over across servers in a cluster
    - A cluster is many machines working like one super-sized one
  - ▶ Clusters are a key WebSphere concept for virtualization of hardware
- WAS also supports a management organization of machines called Network Deployment
  - ▶ Network Deployment organizes machines into cells with a single point of administration for configuration and application deployment
  - ▶ Having centralized administration *enables* virtualization via clusters



# Wide variety of configurations for virtualizing hardware



- Multiple servers on a machine is called vertical scaling
- Multiple servers spread over multiple machines is called horizontal scaling

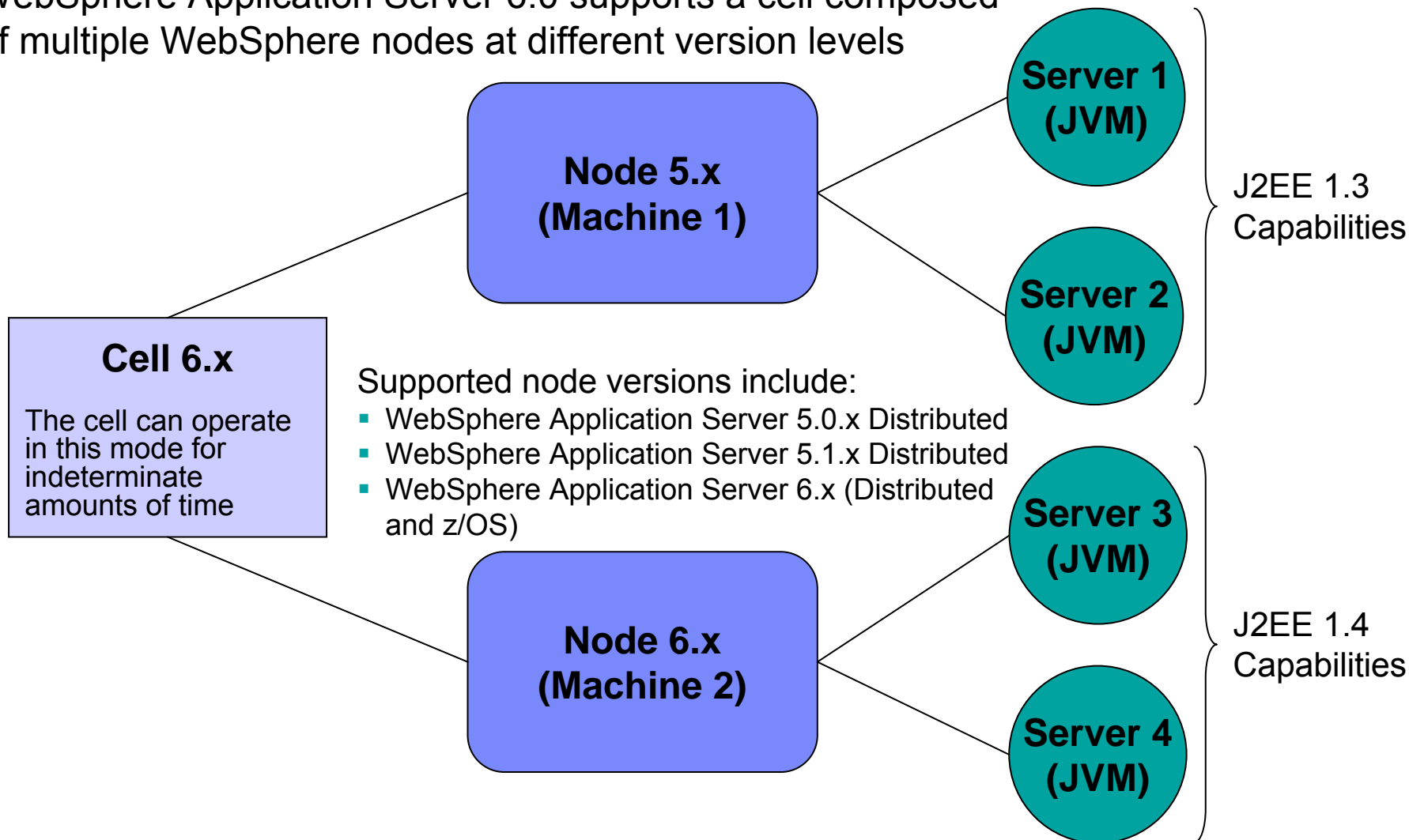
# Cells and Nodes and Servers

- Application Server
  - ▶ Corresponds to a JVM that executes WebSphere Application Server code plus Application code (servlets, JSPs, EJBs etc)
  - ▶ A single configuration of parameters, applications etc.
- Node
  - ▶ Corresponds to a machine on which WebSphere Application Server is installed. Several Application Servers can run on the same node
- Cell
  - ▶ A grouping of Nodes that are managed through a single administrative console. A special node in the cell, called the Deployment Manager, hosts all the configuration information for all nodes and application servers in the cell

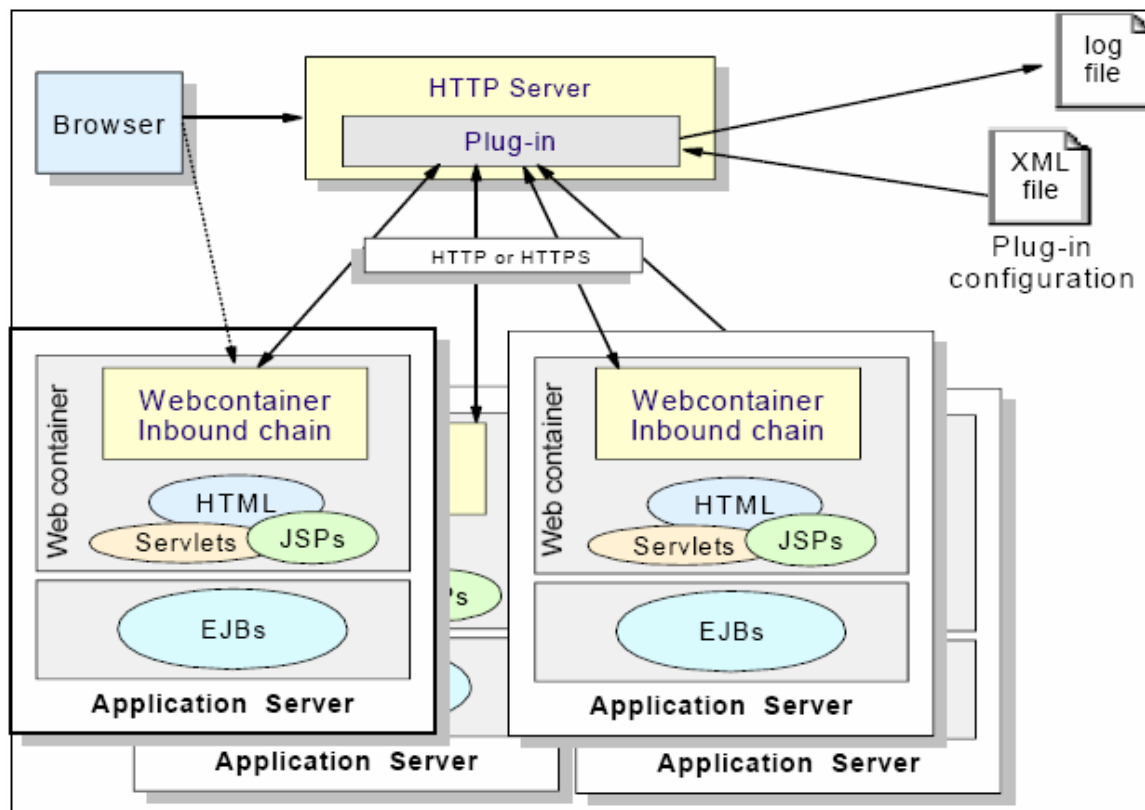


# Cells, Nodes and (Application) Servers

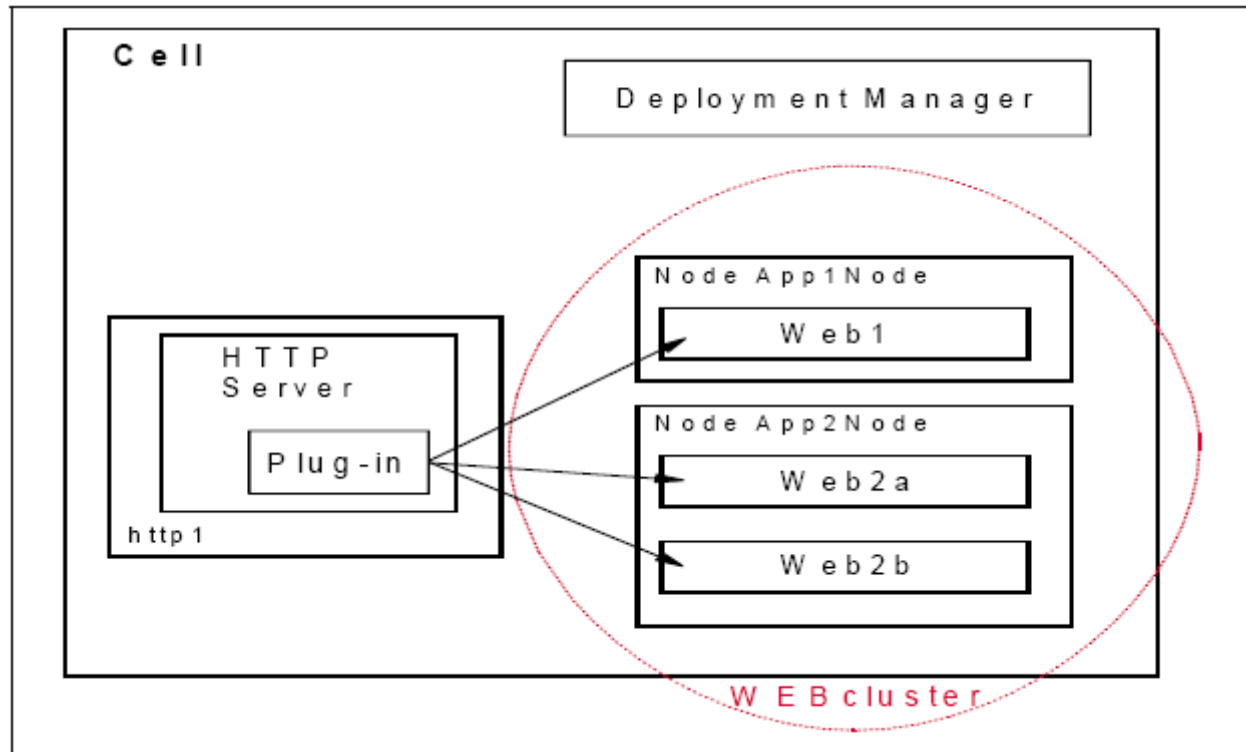
WebSphere Application Server 6.0 supports a cell composed of multiple WebSphere nodes at different version levels



# Application Server components and their interactions



# A web Cluster



# Web cluster

- The web containers in the application servers of a cluster act like a single virtual web container
- The WebSphere plug-in to the web server performs work-load management and failover across the individual application servers of the cluster
- Weighted round robin and random WLM algorithms are supported
- The individual application servers of the cluster with their weights are defined to the web server plug-in through a simple XML configuration file
  - ▶ This can be created and maintained through the deployment manager of the cell, either through interactive console or scripts – you don't have to type this all in 😊



# Web Server Plugin configuration fragment

```
<ServerCluster CloneSeparatorChange="false" LoadBalance="Round Robin" Name="WEBcluster"
  PostSizeLimit="-1" RemoveSpecialHeaders="true" RetryInterval="120">
  <Server CloneID="vve2m4fh" ConnectTimeout="5" ExtendedHandshake="false" LoadBalanceWeight="2"
    MaxConnections="-1" Name="app1Node_Web1" WaitForContinue="false">
    <Transport Hostname="app1.itso.ibm.com" Port="9080" Protocol="http" />
    <Transport Hostname="app1.itso.ibm.com" Port="9095" Protocol="http" />
  </Server>
  .....
  <PrimaryServers>
    <Server Name="app2Node_Web2a" />
    <Server Name="app2Node_Web2b" />
    <Server Name="app1Node_Web1" />
  </PrimaryServers>
</ServerCluster>
```

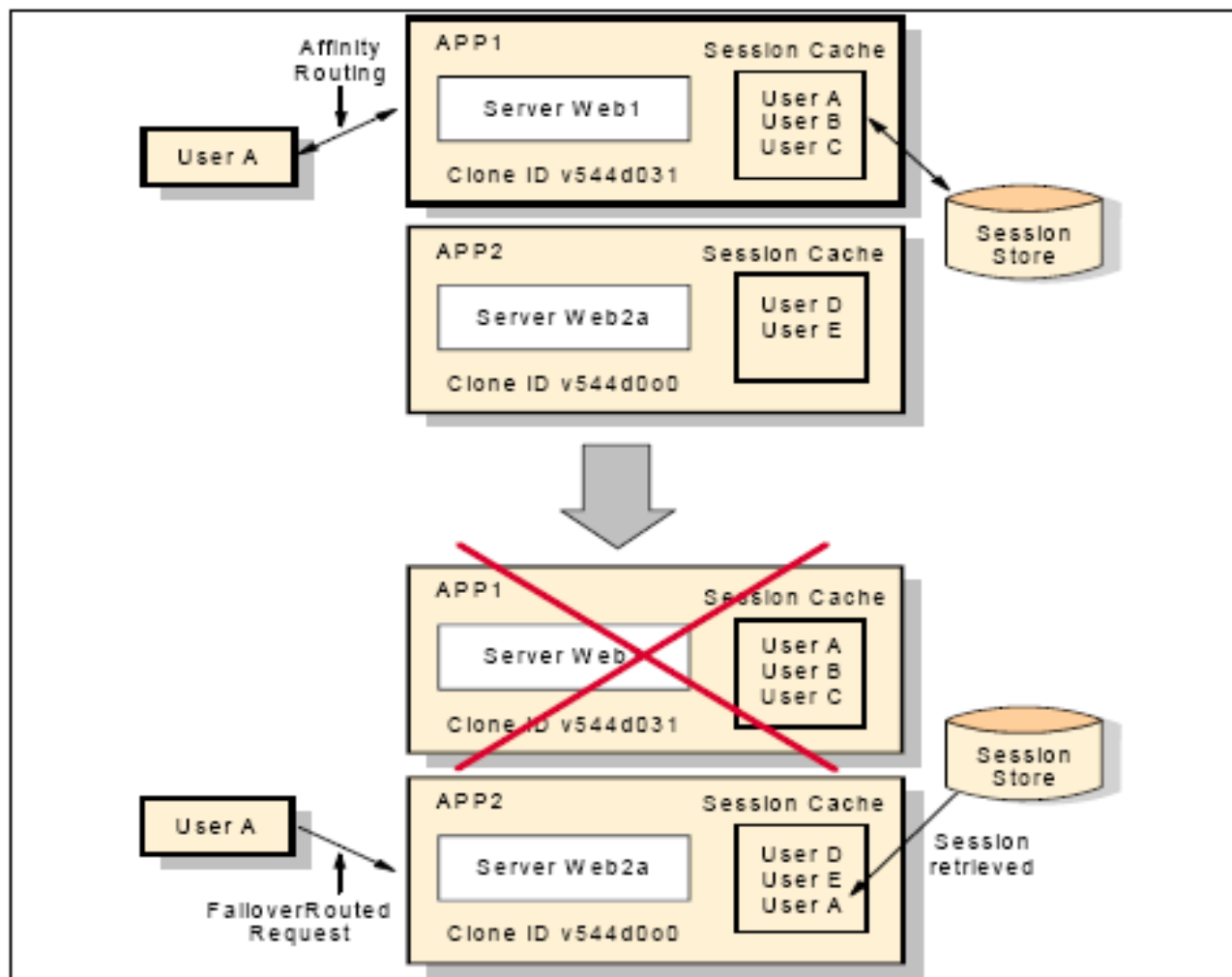


# Web Cluster failover

- Once a session is established, all subsequent requests in the same session are routed to the same application server
  - ▶ This is called “session affinity”
  - ▶ Why? Simplicity, performance and spec compliance (servlet 2.3 spec)
- However, in clusters, unavailable application servers are detected and other available application servers in the same cluster will be used until the application server “comes back”
  - ▶ This event is called “failover”
- Session state is correctly maintained in this situation
  - ▶ Application servers in a cluster can recover session state for sessions they did not initiate
  - ▶ Session state can be shared between application servers in a cluster in 2 ways
    - Via a database
    - Via in-memory replication between servers (even across nodes)



# Failover and session state



# Enterprise Java Beans (EJBs)

- In J2EE, servlets are used to implement application logic that returns HTTP results, usually in the form of HTML
  - ▶ Generally, servlets implement presentation logic
  - ▶ For technical reasons, servlets can also be used to implement web services that return XML-encoded data, rather than presentation
- In J2EE, Session EJBs are the abstraction for implementing business functions that return data
  - ▶ EJBs are the preferred way of implementing web services
- While servlets are “invoked” using HTTP requests that come through a web server, EJBs are “invoked” via requests that come through a CORBA Object request Broker” (ORB)
- Entity EJBs are the abstraction for shared, persistent data in data-stores or transactional back-ends – more later.

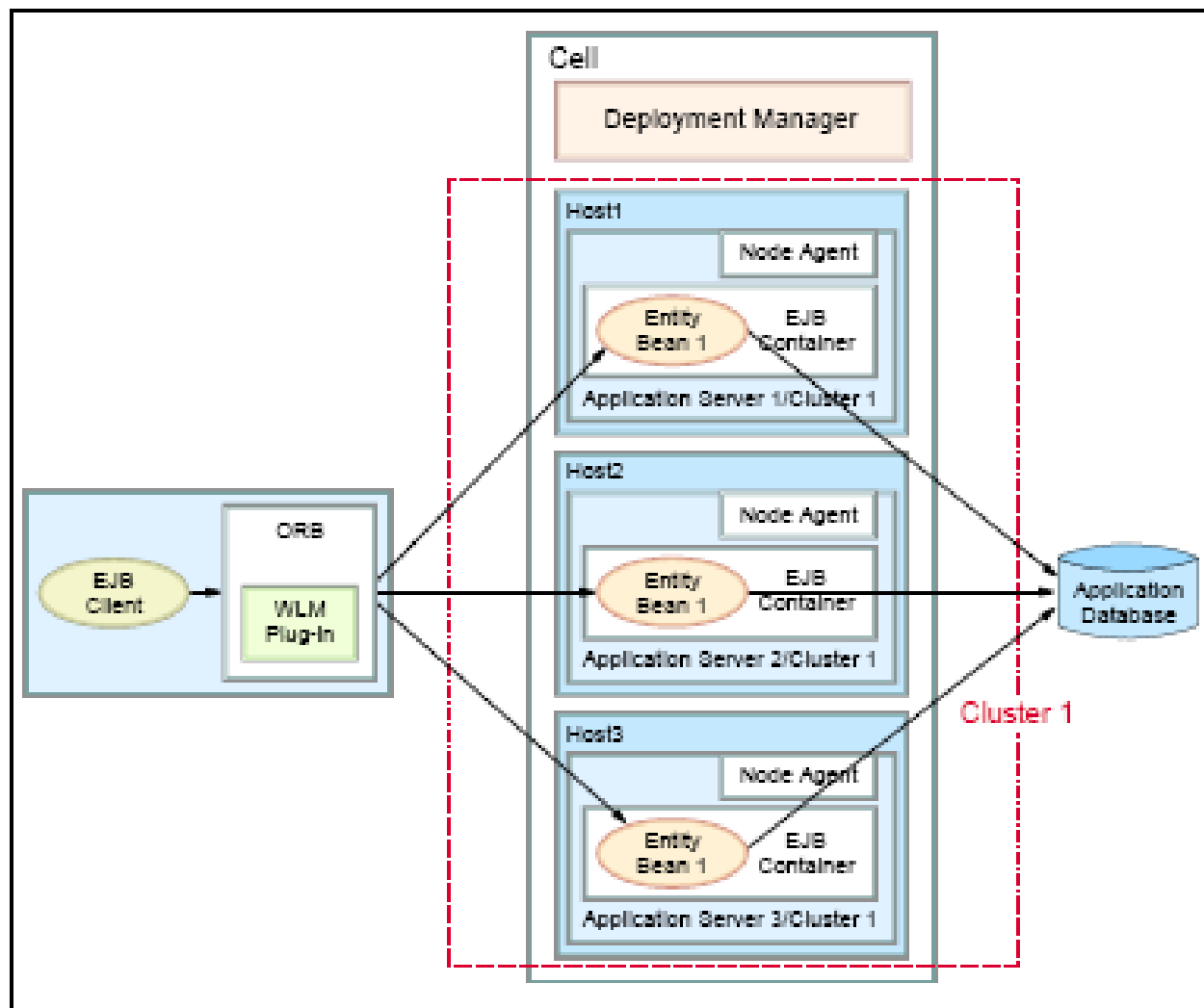


# EJBs and virtualization

- In WebSphere you can create clusters of EJB containers (inside application servers), analogous to web clusters
- In the case of EJB container clusters, work-load management and failover is controlled by a WLM plug-in to the ORB, rather than the web server.
- Stateless Session EJBs support the most flexible work-load management and virtualization.
- For entity EJBs, the concept of “transaction affinity” replaces the concept of “session affinity”.
- Stateful Session EJBs have a concept of session with failover which is somewhat analogous to the session concept for web applications



# EJBs and Virtualization



# Clusters also allow transparent application update

- Clustering enables continuous-availability
  - ▶ Updates can be made to an application and applied a running system without it stopping by *rippling* the cluster
  - ▶ The updated application is installed on one of the members of the cluster
  - ▶ The application on the member is stopped and re-started to load the new application into memory
  - ▶ The remaining cluster members continue to serve workload while the one server is being restarted
  - ▶ Each member is restarted in this fashion, in turn
  - ▶ After the last member is restarted the entire cluster is now updated with the changed application
  - ▶ Clients do not perceive any outage while the update is occurring
- Is this characteristic only possible if you virtualize above the OS level?



## WebSphere XD: A next level of sophistication of virtualizing hardware

- Customers have competing requirements:
- Need good application isolation, the ability to guarantee that high-priority applications will get high levels of service, that lower-priority applications will not get completely starved
  - ▶ An obvious strategy is to dedicate clusters of machines to individual applications.
- Hardware is cheap, but not that cheap. Need to optimize usage of machines, use available capacity to handle exceptional demand peaks.



# On Demand Operating Environment

Customer use case: Large Financial Company

## Conventional Distributed Environment

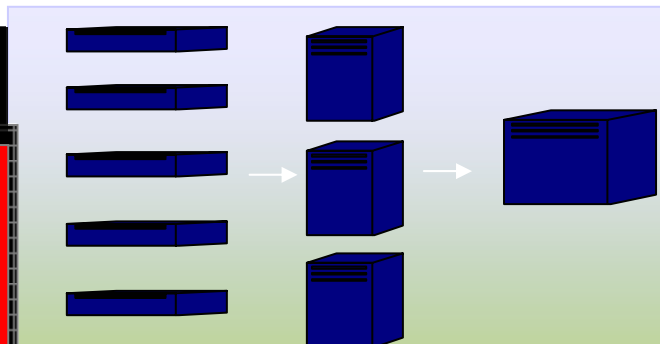
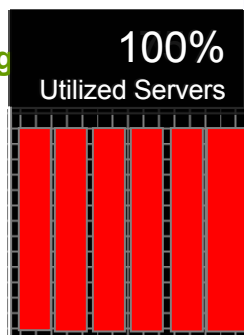
### Environment

- ▶ 30+ applications
- ▶ 100 application servers

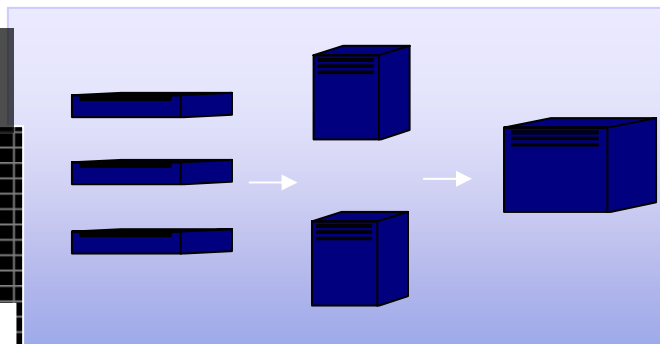
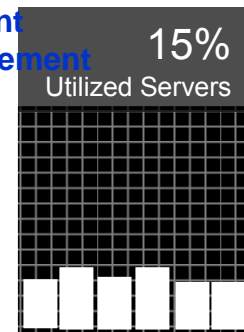
### Challenges

- ▶ Underutilized servers
- ▶ Inability to share resources across server pools – especially during peaks
- ▶ Inconsistent quality of service for business critical applications
- ▶ Human Intensive Monitoring and Managing Environment

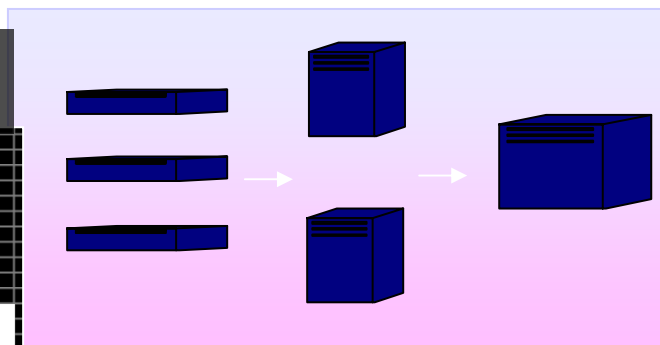
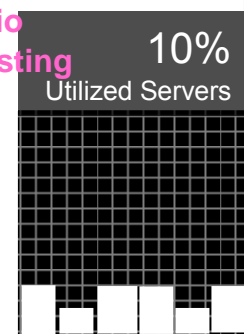
Stock Trading



Account Management



Portfolio Forecasting



# On Demand Operating Environment

Customer use case: Large Financial Company

## Conventional Distributed Environment

- Virtualized
  - ▶ Pool Resources (Node Groups)
  - ▶ Virtualized Applications (JAS)
  
- Autonomic
  - ▶ Operational Policies are attached to Application to reflect operational goals and importance of application
  - ▶ Autonomic Managers monitor environment for maximum utilization using business goals
  
- Results
  - ▶ Reduce total cost of ownership (doing more with same/less)
  - ▶ Increase stability and repeatability of Environment

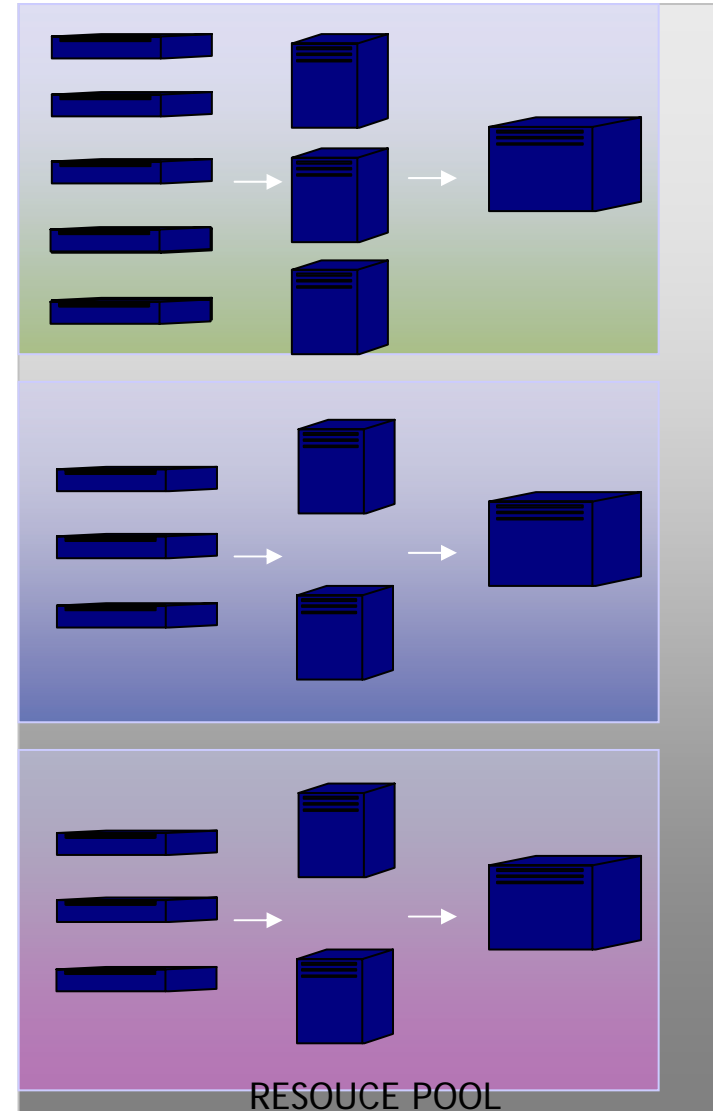
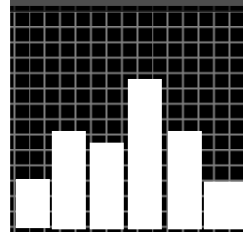
Stock Trading

Customer Support

Account Management

Risk Management

Portfolio Forecasting 55% Utilized Servers

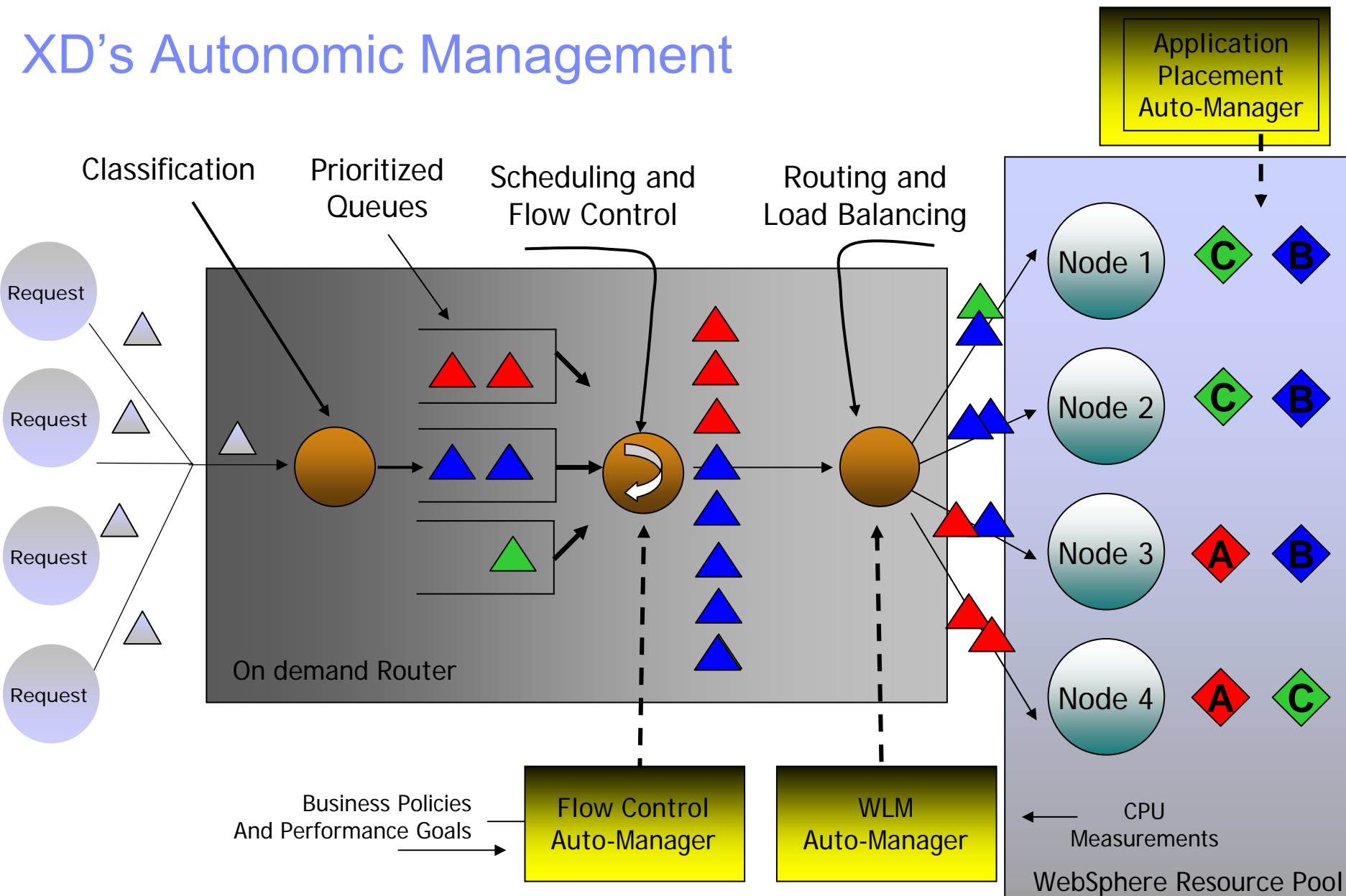


# What are challenges of this topology?

- Appropriate application isolation
  - ▶ How do we ensure that high-priority applications get high levels of service, and are not starved by lower-priority applications?
  - ▶ How do we ensure that lower-priority applications are not totally starved?
- WebSphere's answer - a new, sophisticated work-load manager called the On-Demand Router
  - ▶ Policy-driven configuration:
    - Specify target levels of service for throughput and response-time
    - Monitoring of queue depths and response times of application servers to measure server load and response-time
  - ▶ Maximize usage of hardware while assuring that application policies will be met, or that in the case of saturation, stated goals and policies will govern how degradation happens
  - ▶ Dynamic Clusters
    - Automatic creation and configuration of server instances on groups of Nodes
    - Starting and stopping of server instances "on-demand"

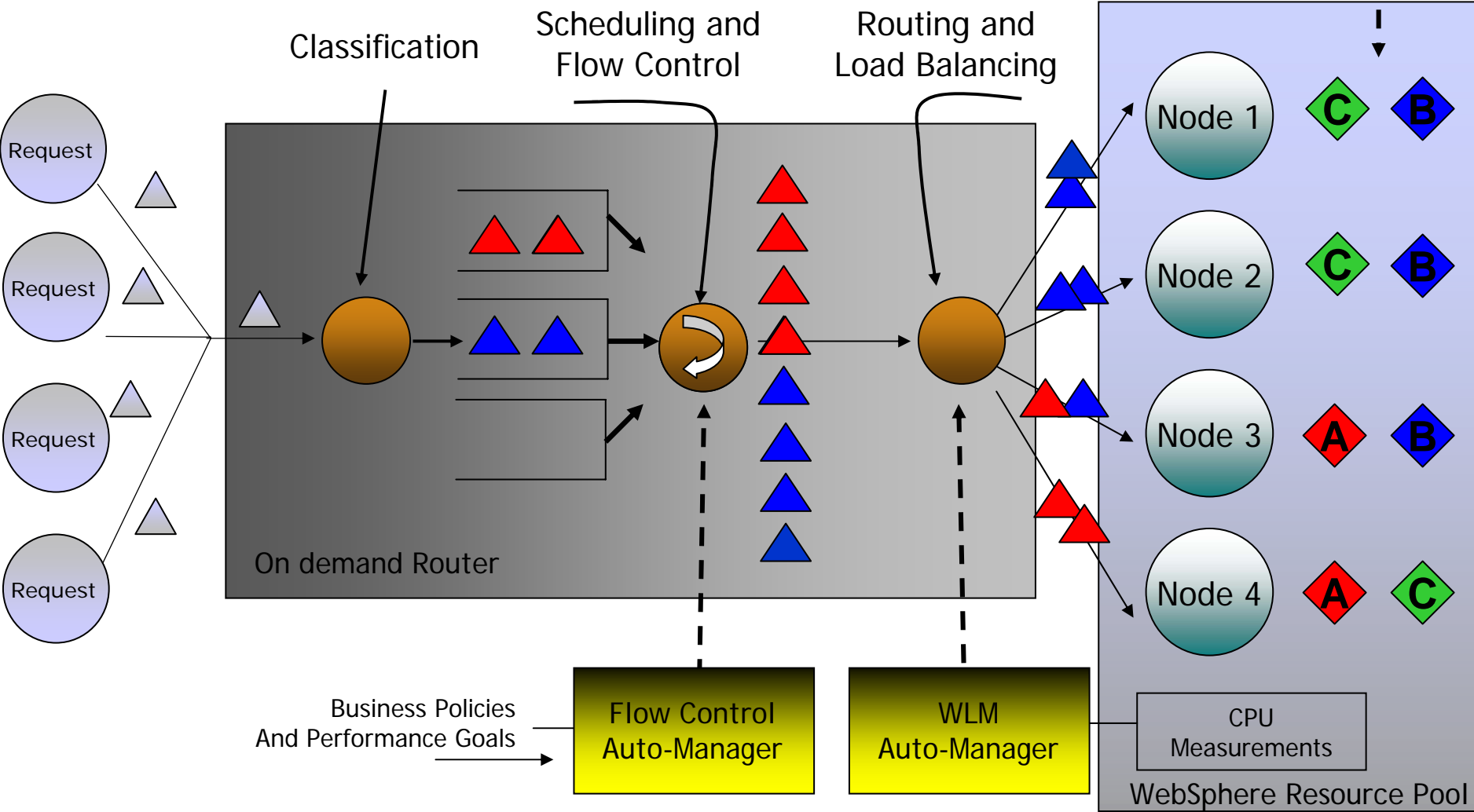


# XD's Autonomic Management

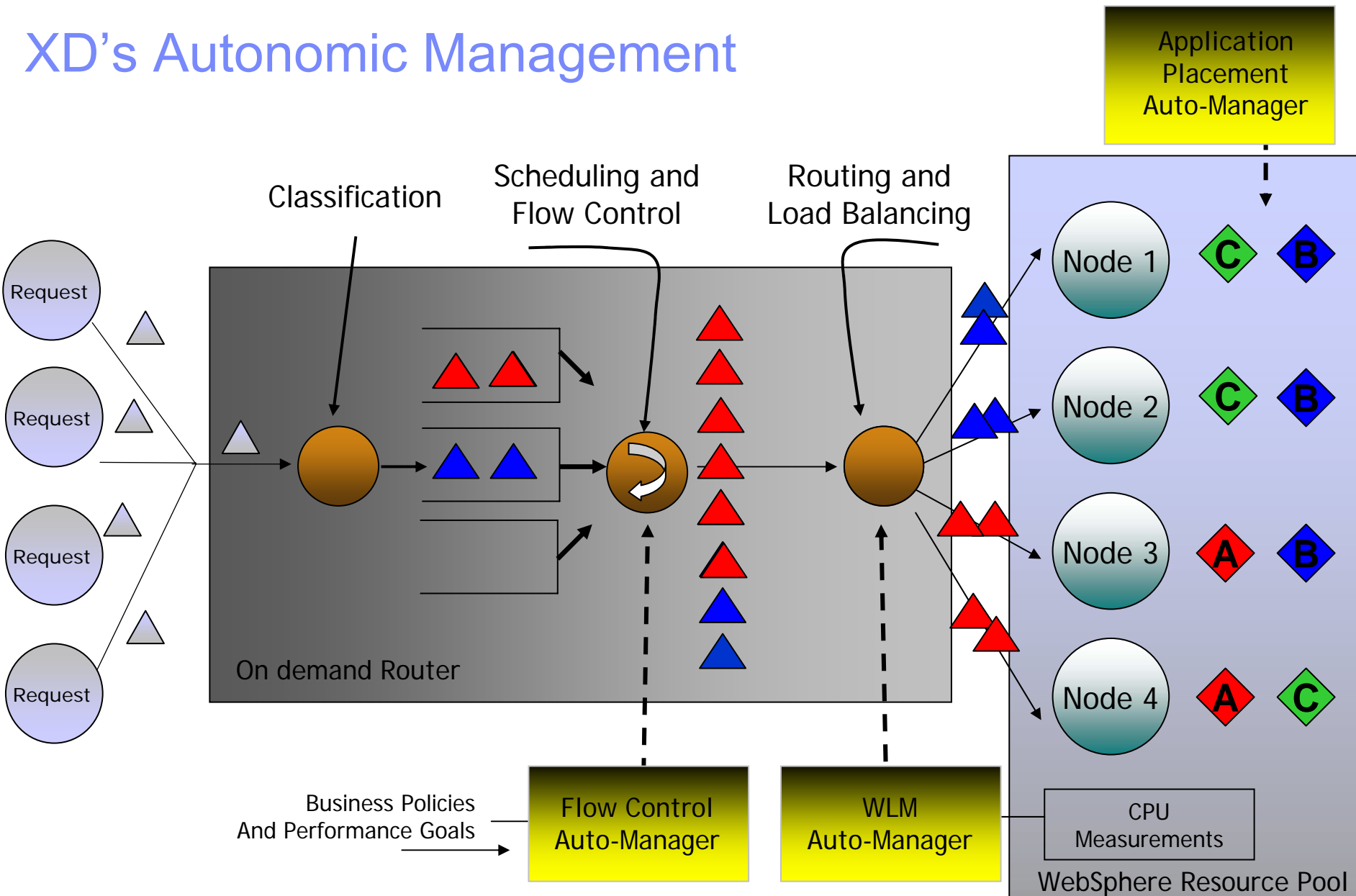


# XD's Autonomic Management

Application Placement Auto-Manager

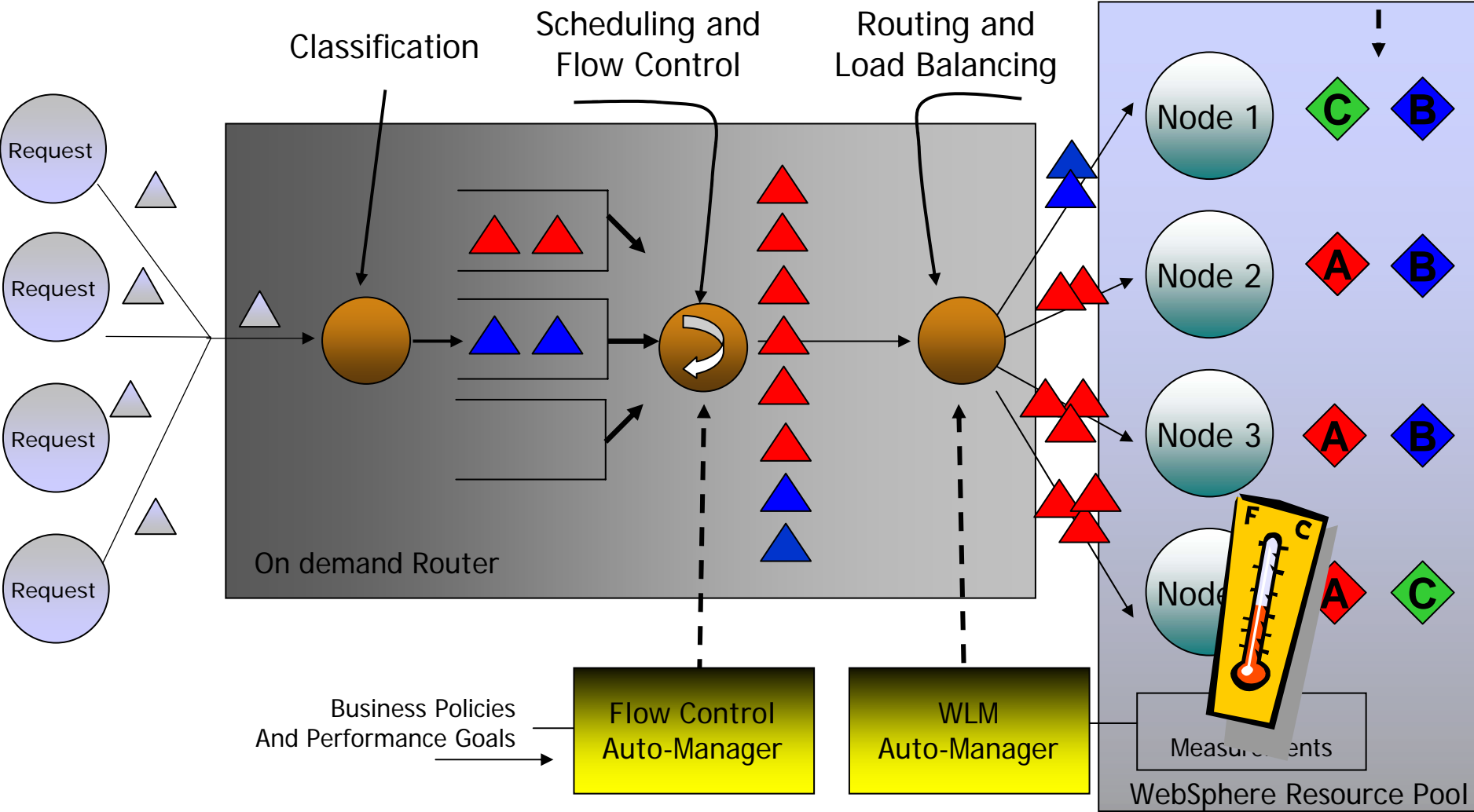


# XD's Autonomic Management

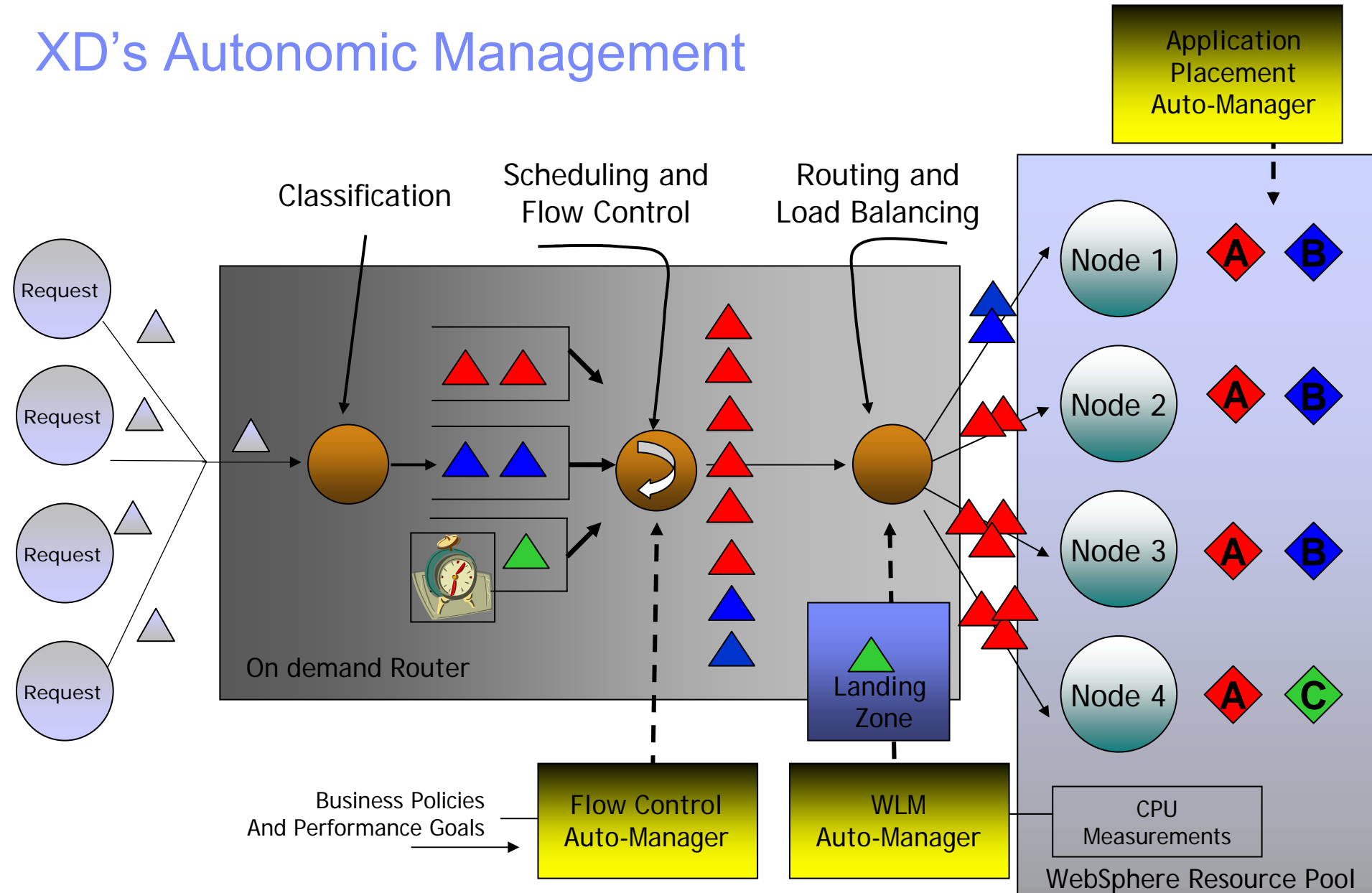


# XD's Autonomic Management

Application Placement Auto-Manager



# XD's Autonomic Management



# Summary

- Application servers provide a virtual environment for executing internet and intranet applications (and some others)
- Application servers present a simple virtual environment to application programmers
- Application servers virtualize across many physically individual computers that may even be running different operating systems (e.g. for WebSphere: Windows, UNIX, zOS)
  - ▶ Importantly, application servers provide management techniques for provisioning (installing software), monitoring and managing applications across many machines
- Application Servers use some virtualization techniques that you don't see at the level of an Operating System or JVM



# Some Questions

- Q: J2EE Application Servers run on JVMs which run on Operating Systems. Each of these layers is performing virtualization – is this working?
- A: Yes, but it's not perfect. There have been and continue to be problems
  - ▶ Sometimes virtual memory systems collide:
    - For example, JVM mark and sweep GC can uselessly page in rarely-used memory pages swapped out by the OS
  - ▶ Sometimes application isolation systems collide or at least lack synergy
    - Simple JVM implementation strategies do not allow sharing of code between OS instances (work goes on here)
      - Why does it take time to start a VM? Shouldn't the “standard” part just be there and only the application-specific part require starting? (c.f. OS and DLLs)
    - OS process isolation mechanisms are only indirectly exploited by application servers which invent their own class-loader schemes
- Q: Could some of the redundancy be removed? Could these layers work better together, or even be coalesced?
- A: ?



# Some References

- Java Servlet Specification:  
<http://java.sun.com/products/servlet/download.html>
- J2EE 1.4 Specification:  
<http://java.sun.com/j2ee/1.4/download.html#platformspec>
- WebSphere Application Server V6 Scalability and Performance Handbook:  
<http://www.redbooks.ibm.com/abstracts/SG246392.html?Open>
- WebSphere Application Server V6 System Management & Configuration Handbook: <http://publib-b.boulder.ibm.com/Redbooks.nsf/RedbookAbstracts/sg246451.html?Open>



# Thank-you!

# Questions?

