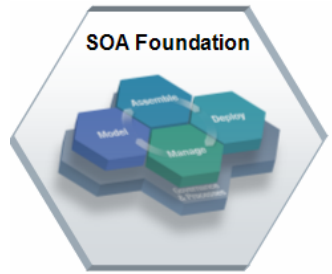
Best practices for managing the performance of composite applications

John Knutson
Market Manager - Web Services Management
IBM Tivoli
Knutson@uk.ibm.com



Last updated: Friday, 19 May 2006

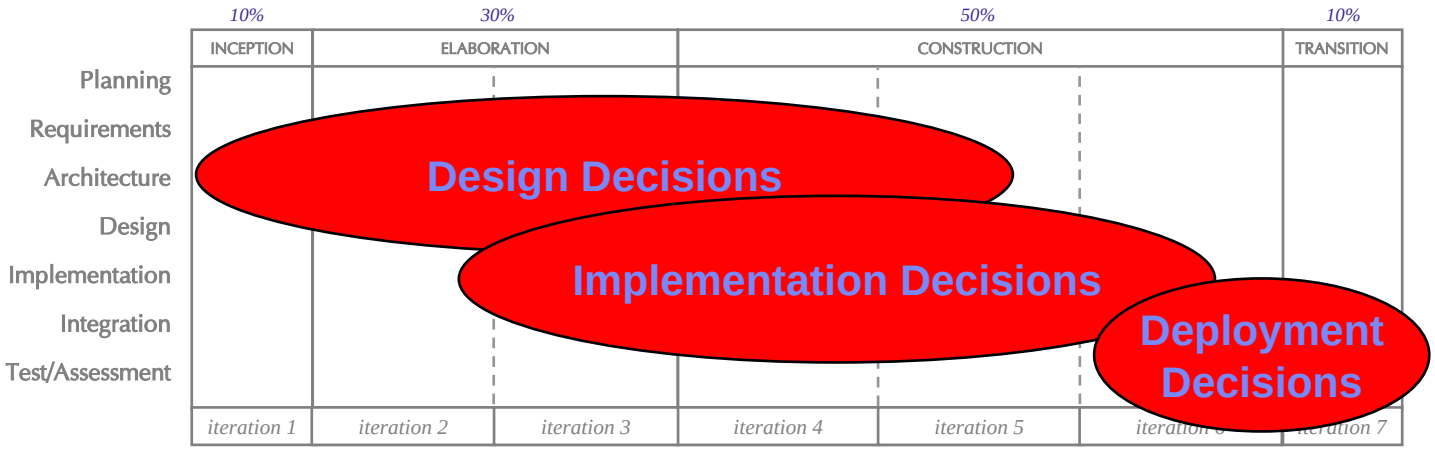
Agenda

- **Introduction**
- **Architecture & Design Best Practices**
- **Development Best Practices**
- **Testing Best Practices**
- **Deployment Best Practices**
- **Management Best Practices**
- **Conclusion**

Composite Application Performance

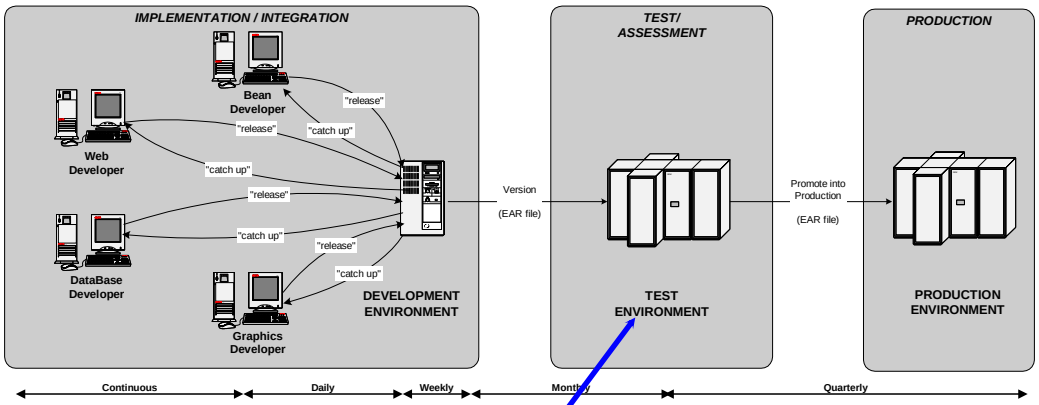
- **Performance and Availability problems . . .**
  - have moved to center stage
    - ⇨ *no longer simply an issue with firm's internal users*
    - ⇨ *affects customers, revenue, market share, brand equity*
  - but are typically . . .
    - ⇨ ***discovered late***
    - ⇨ ***created early***
- ***Life-cycle Best Practices are essential to . . .***
  - Uncovering latent problems that appear only under load conditions
  - Making better informed configuration decisions
  - Ensuring applications meet end user requirements for response time and availability on day one
  - Ensuring scalability to meet unexpected traffic peaks & future growth
  - Identifying thresholds to be used for production monitoring
  - Avoiding all those business impacts

Created Early …



"For every phase in which a defect is left beyond the point of detection, it becomes 80 to 1,000 times more expensive to remove."

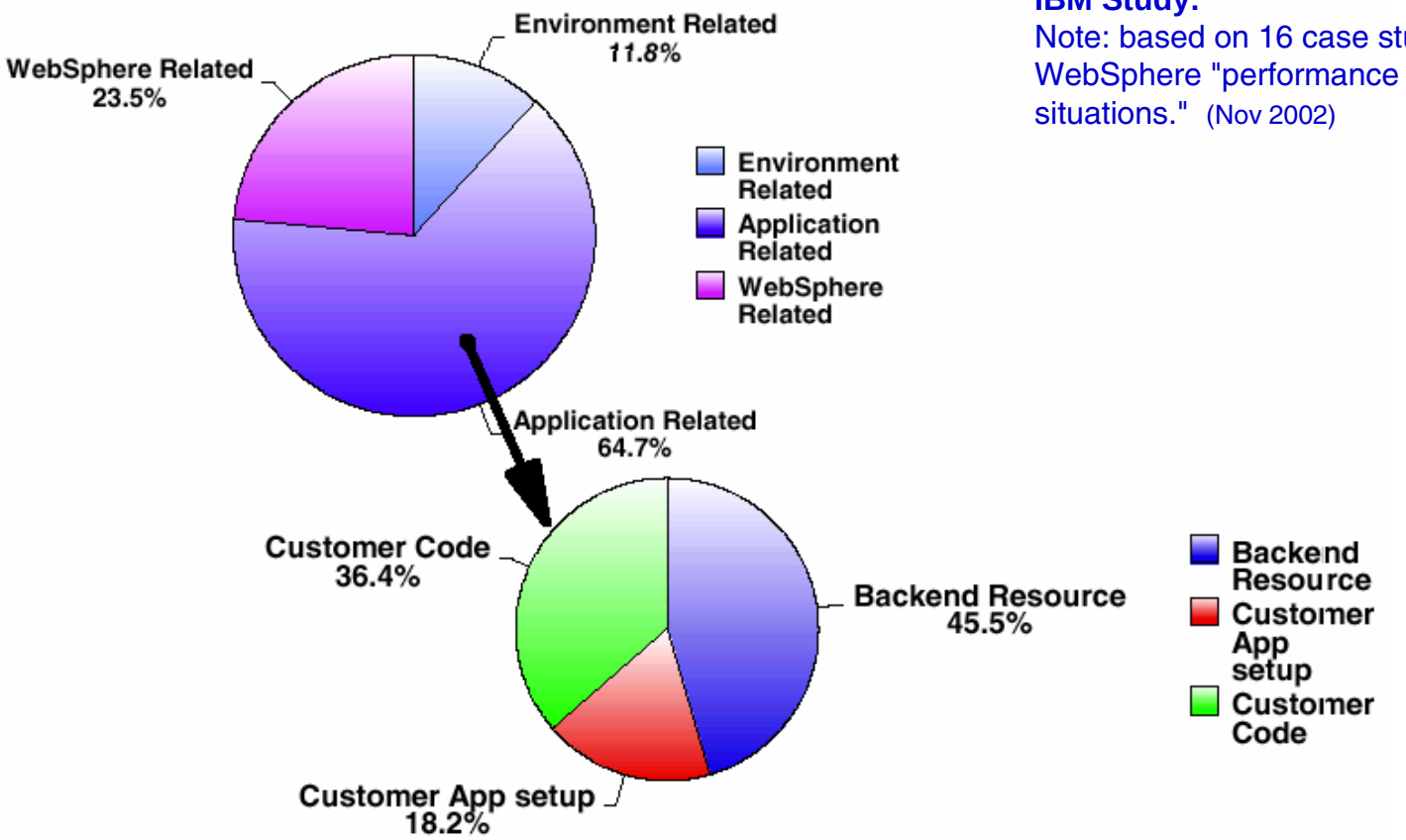
Gartner Aug-02



Last Chance To Catch & Prevent latent Production Problems

. . . Created Early …

IBM Study: "factors inhibiting performance & scale"

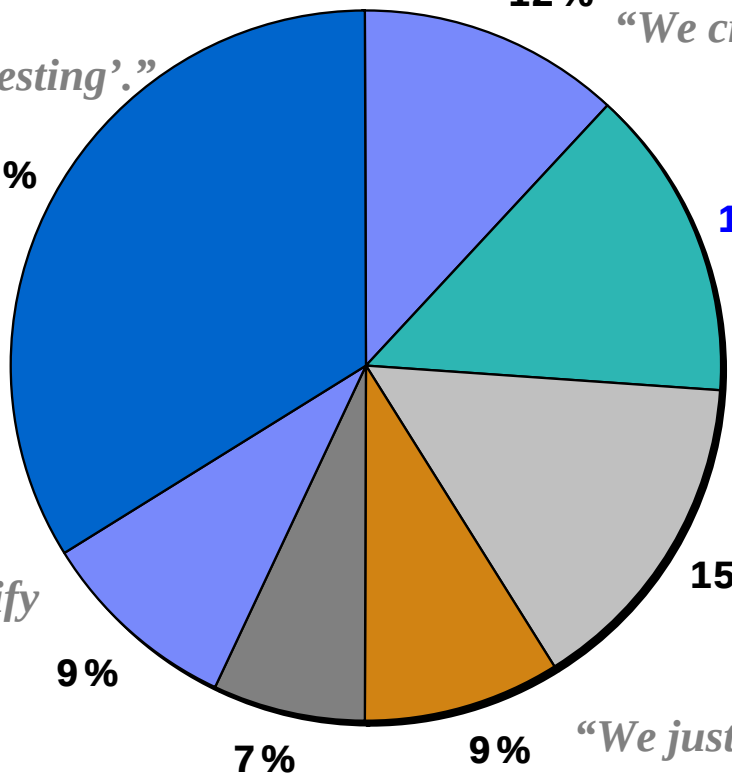


IBM Study:
Note: based on 16 case studies of WebSphere "performance critical situations." (Nov 2002)

SOURCE: Vanderham, Tim; IBM Corporation, presentation for session M21 at the WebSphere Technical Exchange 2002, Las Vegas, NV, Nov-2002.

… Discovered Late

Gartner asked, "What happens when you roll out a new application?"



"We cross our fingers and toes."

"We hear 'it worked fine in testing'."

"It meets all tested and expected response time measurements; users are happy."

"Our IT department is overwhelmed with calls."

"Our tools did not identify what was wrong."

"We just add bandwidth to get rid of the problem."

"Our company loses revenue, time and resources."

Theresa Lanowitz & Debra Curtis, Application Deployment: Tearing Down the Walls between AD and Operations, Gartner Group Presentation, Oct-2002.

Architecture & Design

Architecture & Design Phase

- **Requirements gathering**
  - Comprehensive
    - Best/worst case scenarios
  - Detailed business requirement
    - Projects work best when this is done!
- **Application Types**
- **Suitable architectures**
- **Composite Application environments**
  - Web Services and SOA
  - J2EE
  - Asynchronous Messaging
  - Brokers and ESBs
  - Legacy Systems

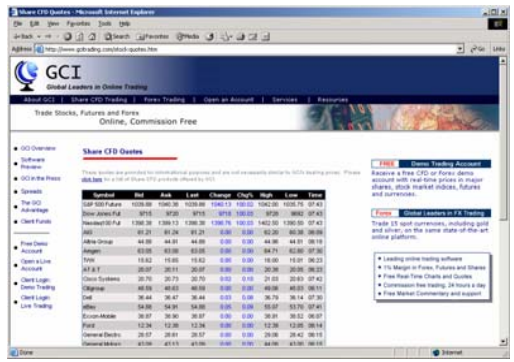
Application Types

- **5 typical composite application types**
  - Online trading
  - B2B
  - Online shopping
  - Information Sites
  - Self Service Portals

Online Trading

- **High complex transaction volumes**
- **Typically connected to a legacy system**
- **High security**
- **Low search traffic**
- **Time sensitive information**
- **Dramatic volume variation**
- **Long session duration**

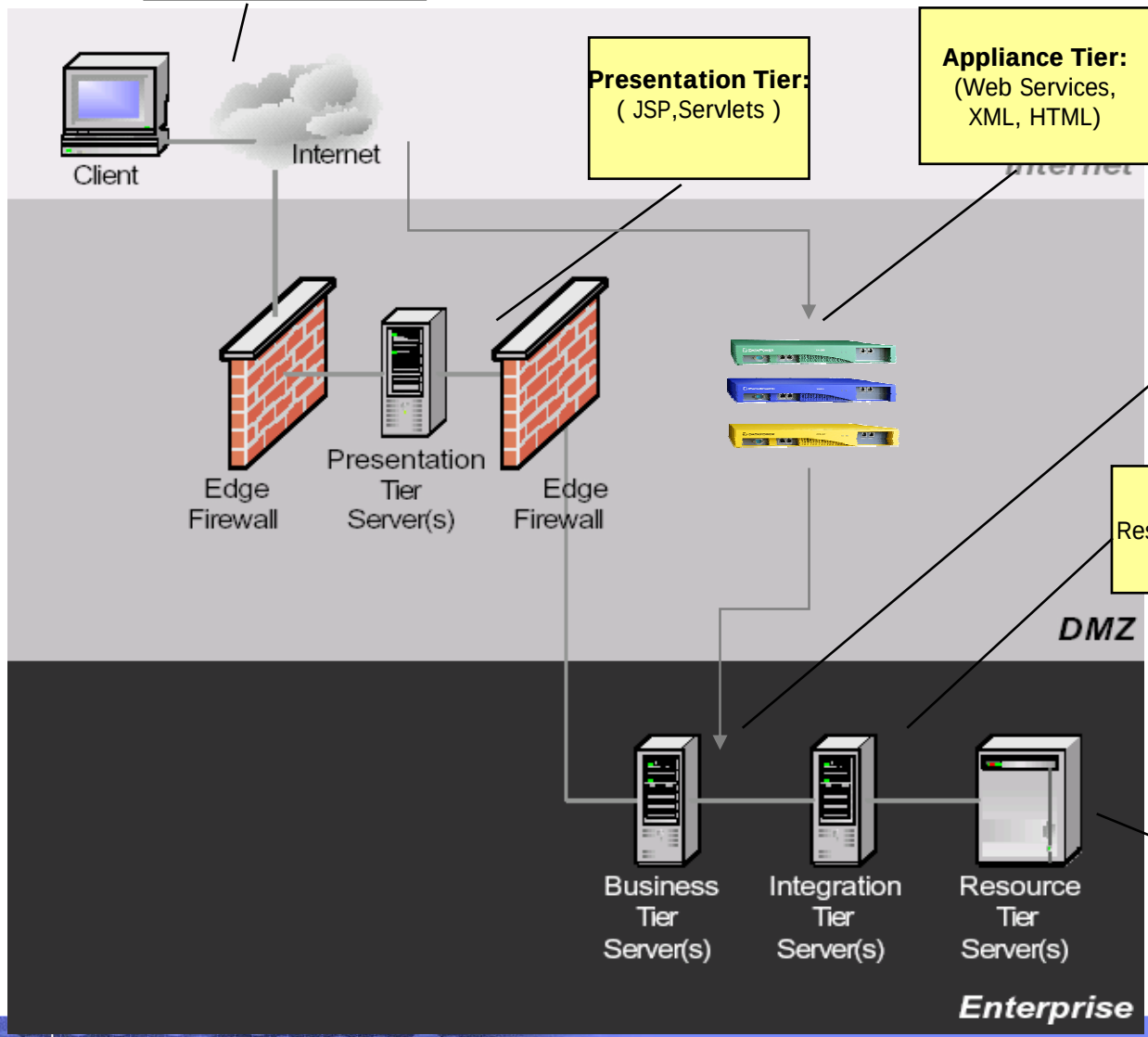
- **Limited Caching potential - requires a large test environment**
  - Lots of clients and data - to ensure test data is not cached
- **High security**
  - Large number of pages using SSL
  - Federated Identity
  - Consider a SOA Appliance



B2B Trading

- **Low utilization**
- **Constant traffic volumes**
- **Few complex transactions**
- **High interaction with legacy systems**
  - Often multiple back-ends
- **Often nbr. Logged-in users > nbr. of requests**
- **Medium caching potential**
- **Medium security**
  - Site and business dependent
  - Federated Identity
- **May use web services**
  - Requests and responses may contain large amounts of data
  - Session sizes can be very large

A multi-tier Composite J2EE Architecture Pattern



Client Tier:
User Interface
(HTML,Applets,Scripts)

Presentation Tier:
(JSP,Servlets)

Appliance Tier:
(Web Services,
XML, HTML)

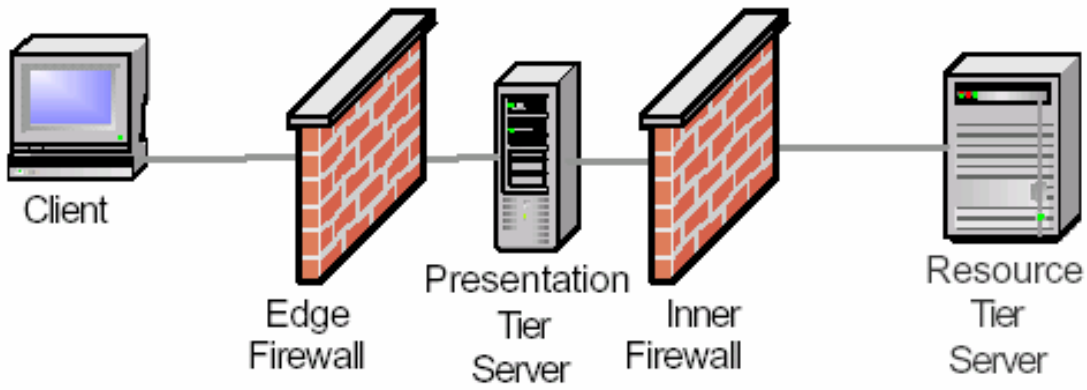
Business Tier:
Business logic
(EJB, BPEL and other Business Objects)

Integration Tier:
Resource adapters legacy, external systems,
ESB, JMS, JDBC, Connectors, Legacy

Resource Tier:
Databases, legacy resources

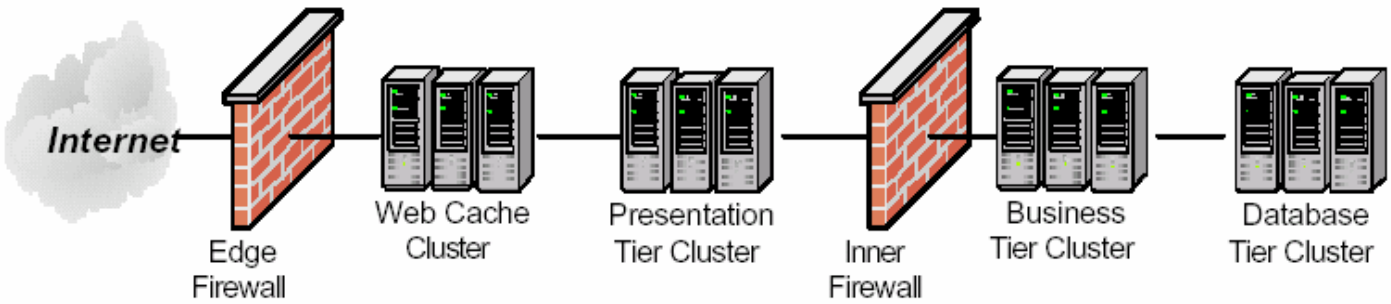
A simple J2EE Architecture Pattern

- Tailor your environment to your business requirements
- Each logical J2EE tier on a single machine or
- consolidate the presentation and business tiers
- + easier to design
- - scalability is limited
- - enterprise business logic is not protected behind two firewalls



A clustered J2EE Architecture Pattern

- **Choose a platform that supports clustering on each major tier**
- **Goals are to allow scalability through adding hardware to the cluster needing power**
- **Increasing availability by adding redundancy**
- **Clustering options:**
  - "Vertical" scaling (Scaling up)
    - The addition of computational and system resources, e.g. adding CPU's or memory to a single machine
  - Horizontal" scaling (Scaling out)
    - The server application is distributed across more than one machine instead of replacing an existing machine with a more powerful model

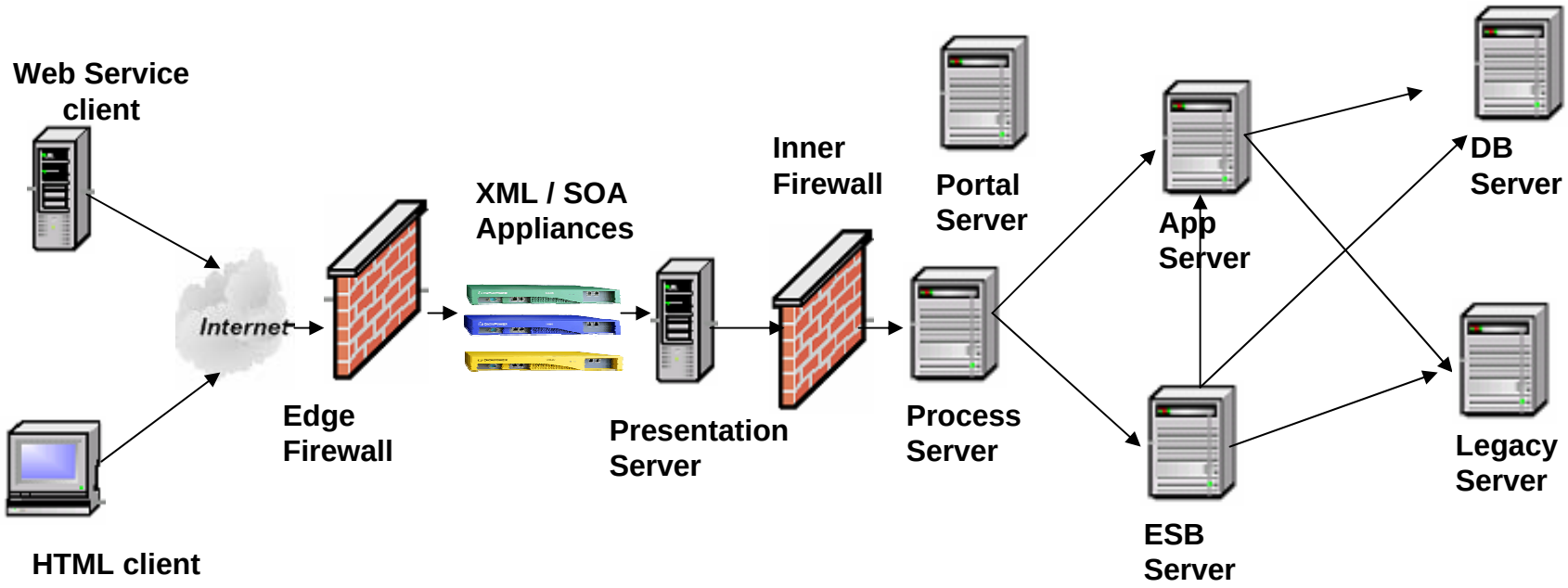


A SOA Architecture Pattern

- **Adopt SOA Principles, e.g. Service Contract, Service Reusability**
- **Implement an ESB for loose coupling and service abstraction**

Each component eligible for clustering

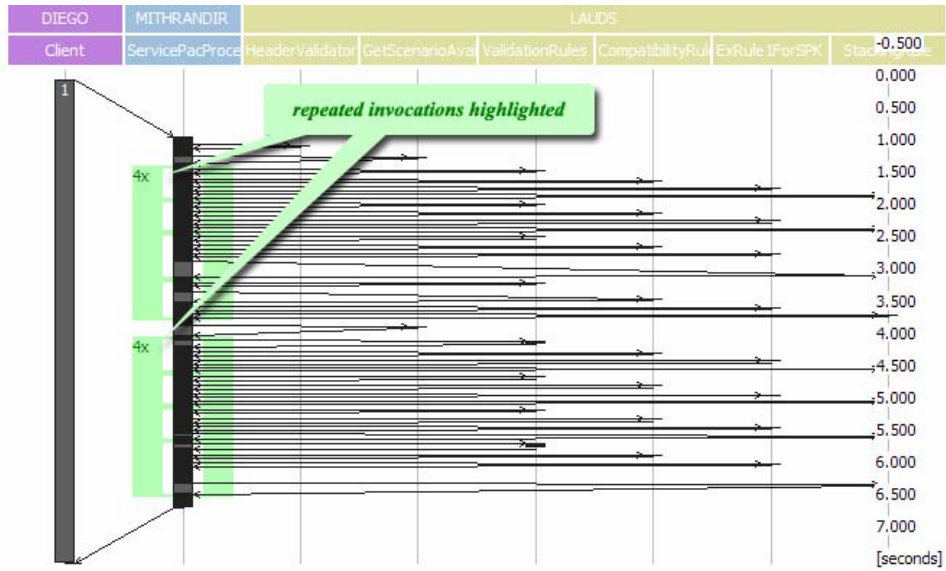
- **Some SOA Anti-Patterns**
- **Overly granular business services**
- **Remote access to local services**
- **Overuse of XML**
- **Use of loosely coupled services where tight coupling is required**



SOA Best Practices – Chatty Services

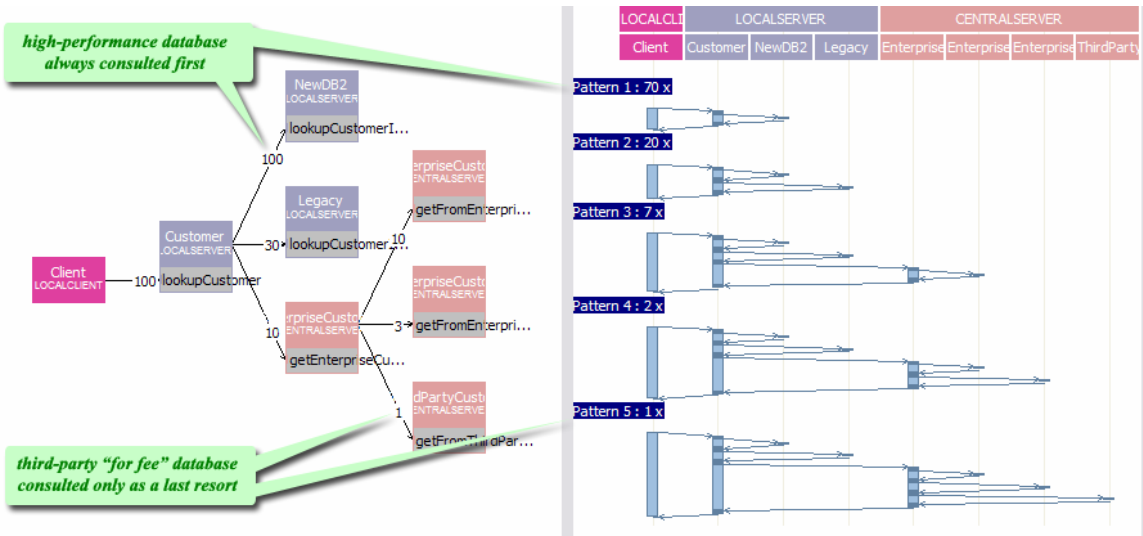
- **Problem**
  - Multiple Web services or a single service called multiple times, each call passing a tiny piece of data
  - Fine-grained service operations
- **Consequences**
  - Degradation in performance and costly development. Extra effort/knowledge to aggregate these too finely grained

- **Solutions**
  - Re-factoring the design to combine individual pieces of data in one document
  - Define services that map back to business goals using a good service modeling technique to determine appropriate coarse-grained services



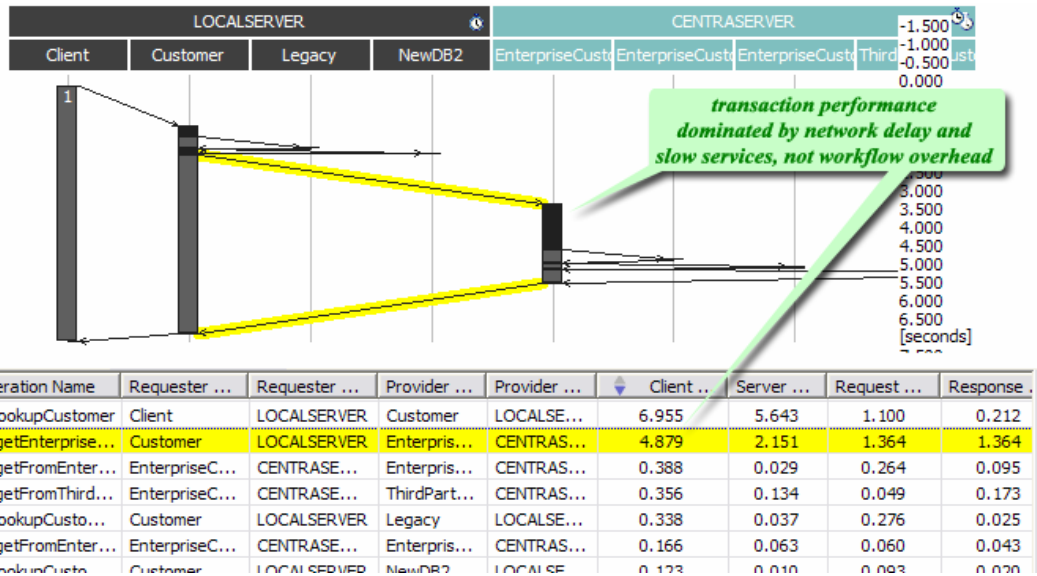
SOA Best Practices – Verify application cost structure

- **Problem**
  - Application design not optimized for IT infrastructure
- **Consequences**
  - Performance problems may arise from accessing slower services first
- **Solutions**
  - Customer information search should use local databases, corporate mainframes and, as a last resort, third-party (paid) service
  - Monitor service flows and validate that Execution Patterns match Use Cases



SOA Best Practices – Understand cost of XML

- **Problem**
  - Large XML may take excessive cpu resource to parse and network bandwidth to transmit
- **Consequences**
  - Longer response times or extra cost for more powerful infrastructure
- **Solutions**
  - Use XSLT to reduce message content
  - Use SOA / XML acceleration appliance



Operation Name	Requester ...	Requester ...	Provider ...	Provider ...	Client ...	Server ...	Request ...	Response ...
lookupCustomer	Client	LOCALSERVER	Customer	LOCALSE...	6.955	5.643	1.100	0.212
getEnterprise...	Customer	LOCALSERVER	Enterpris...	CENTRAS...	4.879	2.151	1.364	1.364
getFromEnter...	EnterpriseC...	CENTRASE...	Enterpris...	CENTRAS...	0.388	0.029	0.264	0.095
getFromThird...	EnterpriseC...	CENTRASE...	ThirdPart...	CENTRAS...	0.356	0.134	0.049	0.173
lookupCusto...	Customer	LOCALSERVER	Legacy	LOCALSE...	0.338	0.037	0.276	0.025
getFromEnter...	EnterpriseC...	CENTRASE...	Enterpris...	CENTRAS...	0.166	0.063	0.060	0.043

Development

J2EE Development Best Practices

- Use appropriate frameworks
- Static vs. Dynamic Pages
- Servlets
- Java Server Pages
- Logging
- HTTP Sessions
- Enterprise Java Beans
- Database Connections
- Object pools
- Efficient Java Code

Follow links to Backup for details

Profile your Applications

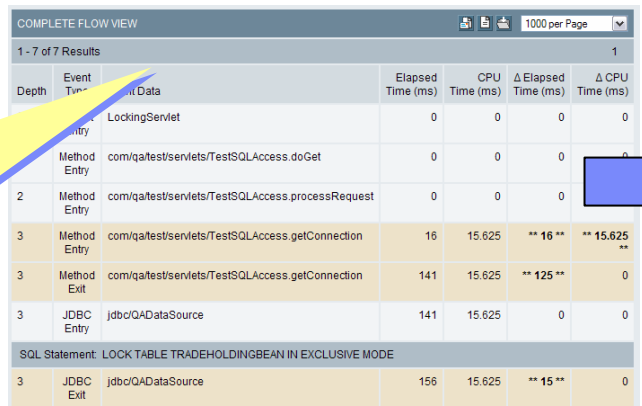
- **Provide detailed view of an applications behaviour**
  - Down to the method level
  - Which transactions/methods use the most CPU
  - Which packages allocate most object
  - Bottleneck analysis
  - Let developers debug problems using real production data

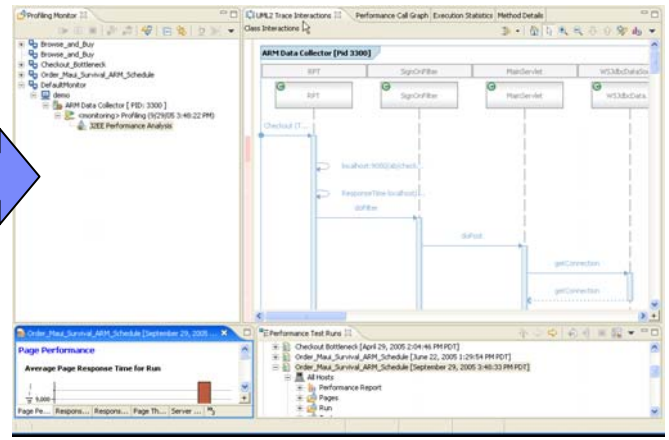
ITCAM Basic for WebSphere

ITCAM for WebSphere / RTT

Rational IDE plus IPOT

Trace information from ITCAM for WS can be exported directly to developers using IBM Performance Optimization Toolkit





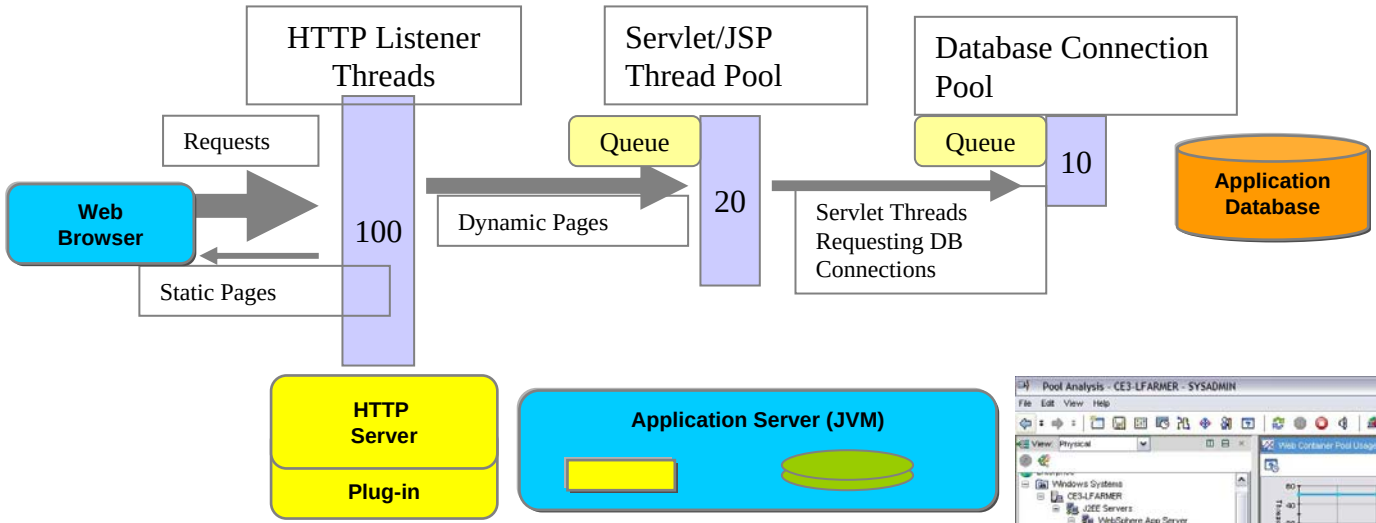
Deployment

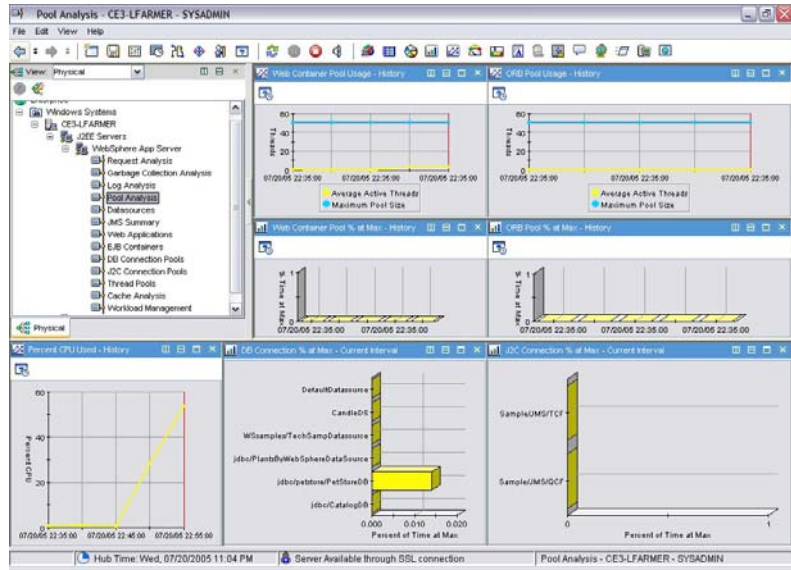
Deployment Best Practices

- The HA Pattern
- N-tier Model
- Vertical and Horizontal Scaling
- *Pool funneling*
- *JVM Heap*
- Http Server Patterns, Session state and Clustering
- Other Web Application Component Issues
- Monitoring

Pool Funneling

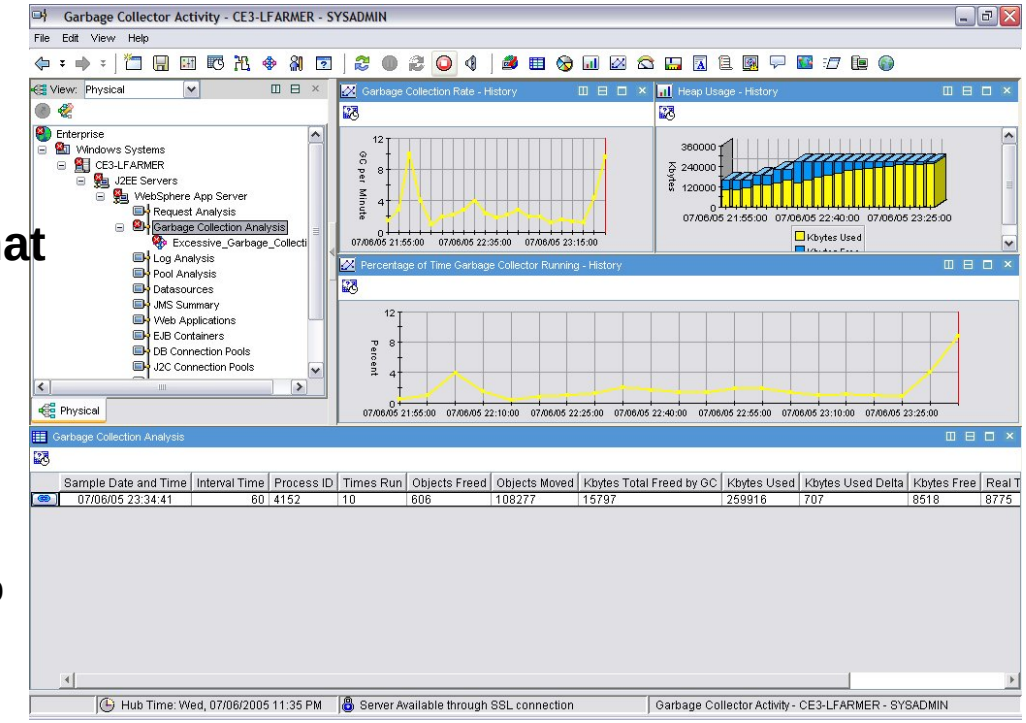
- **Create reducing thread pool funnel**
- **Minimize the size of thread pools**





JVM Heap

- **Set heap size large enough so that it is big enough enough to handle peak throughput**
  - Minimizes number of garbage collections
- **Set heap size small enough so that garbage collection does not run for long periods**
  - Application gets no CPU time during garbage collection
- **Minimum heap should be 25-50% of maximum**
- **Total sum of all JVM heaps on server less than total physical memory**



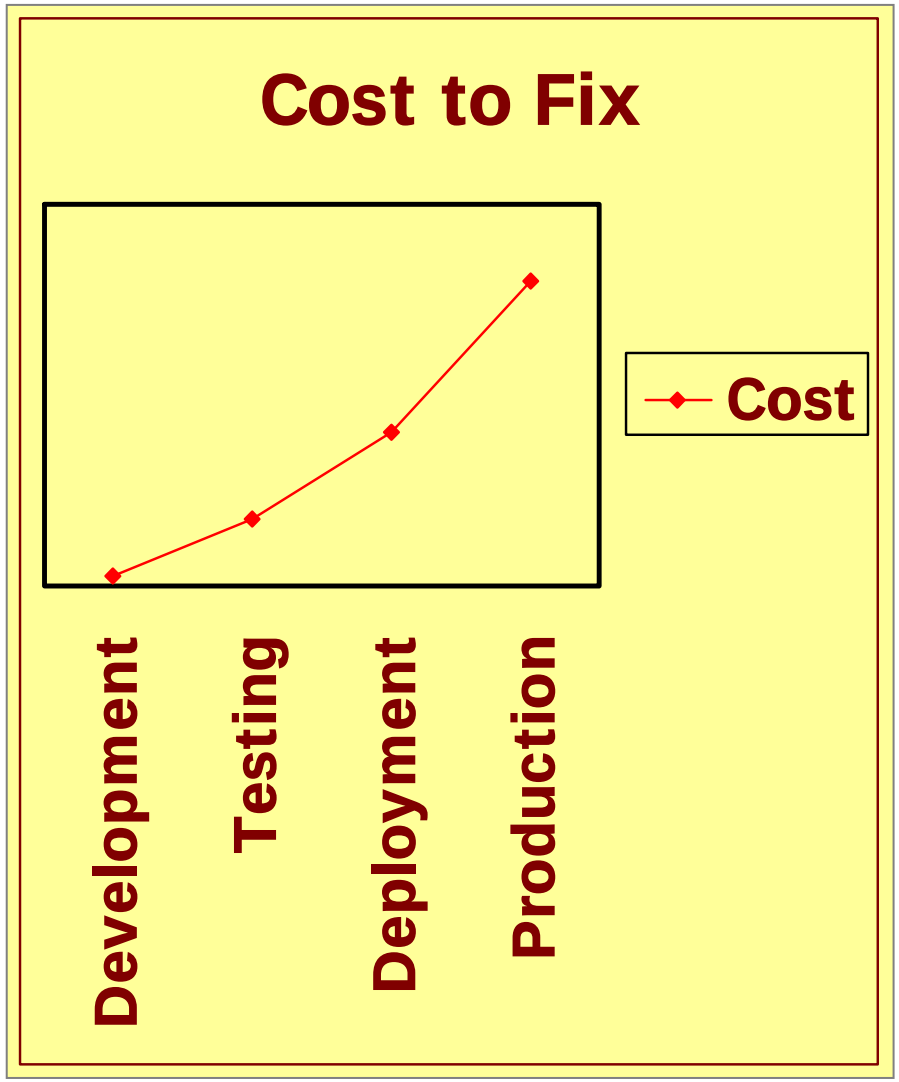
Testing

Performance Testing

- **Why bother?**
- **Who / what's involved?**
- **Types of testing**
  - Unit
  - System
  - Load
  - Stress
- **Application characteristics and testing**
- **Performance and testing**

Why Test?

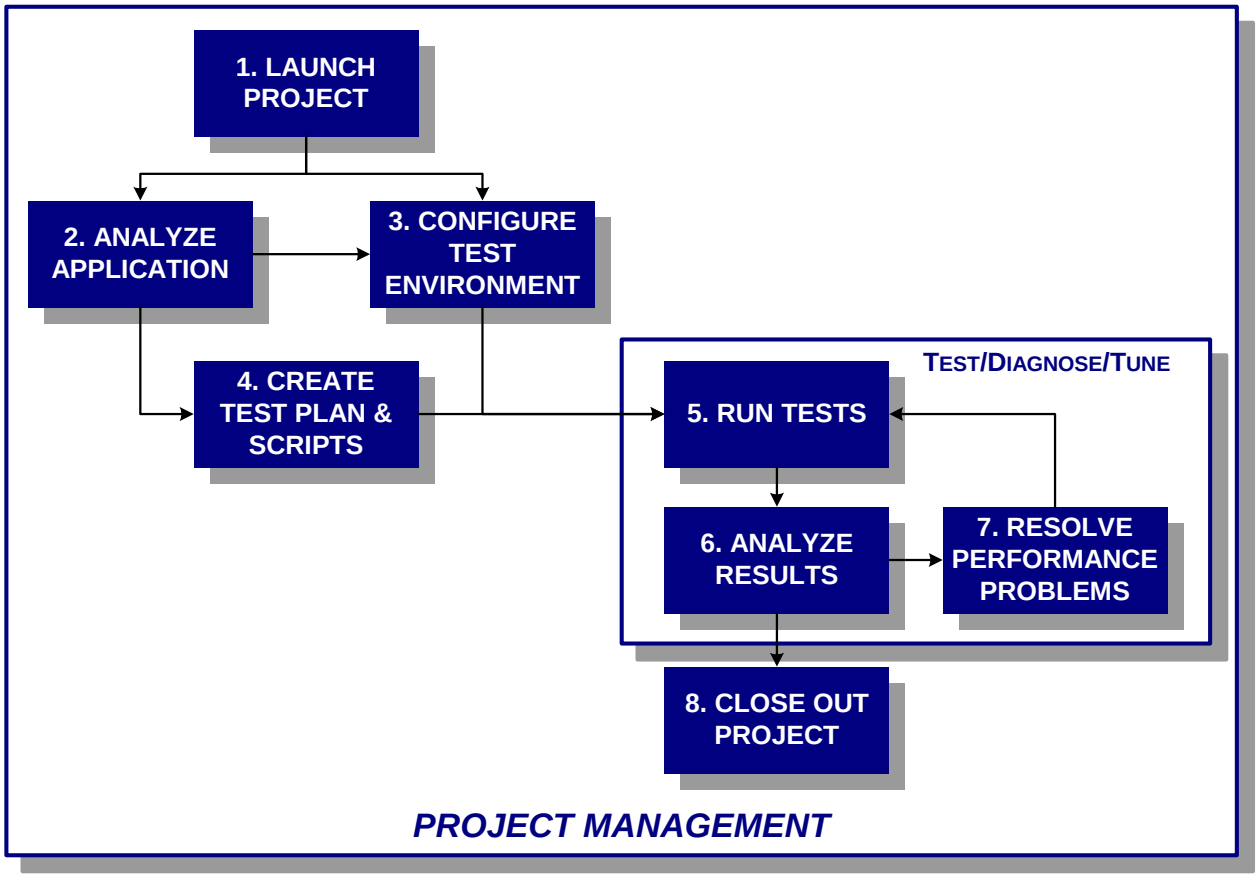
- **Ensures quality**
- **Maintains corporate image**
- **Improves productivity**
- **Reduces development costs**
- **Increases time available for new development**



Overall Best Practices

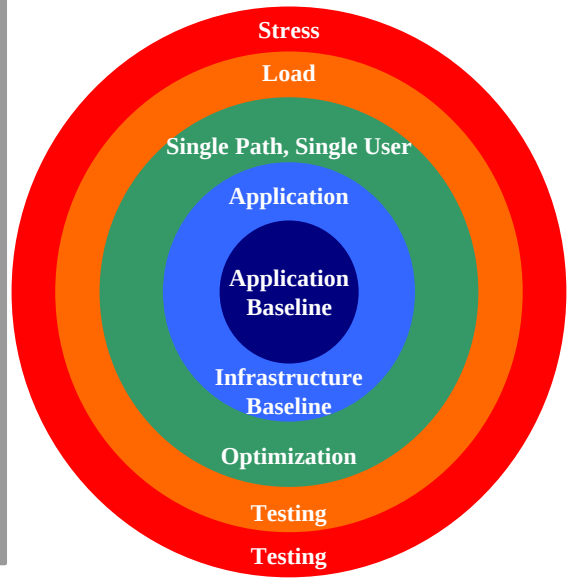
- **Plan for and fund Performance Testing**
- **Adopt an end-to-end – Composite Application - mindset**
- **Control as many variables as possible that can affect your performance measurements**
  - Start with well-designed test plan & scripts
  - Begin with a simple test environment, then systematically work up to increasing complexity and load
  - Test/Diagnose/Tune/Document all along the way

Overall Approach



Methodology

- ***A "light methodology" for repeatable services***
- ***Applied as a guide, not a bible – easily adapted as needed***
- ***Framework for best practices, consulting tools & templates***
- ***Provides a project management framework***



Performance Testing Checklist

- Who should be involved?
- Where should it be done?
- What do I need to know to get started?
- **Has the application been System Tested?**
- Load Testing
- Stress Testing

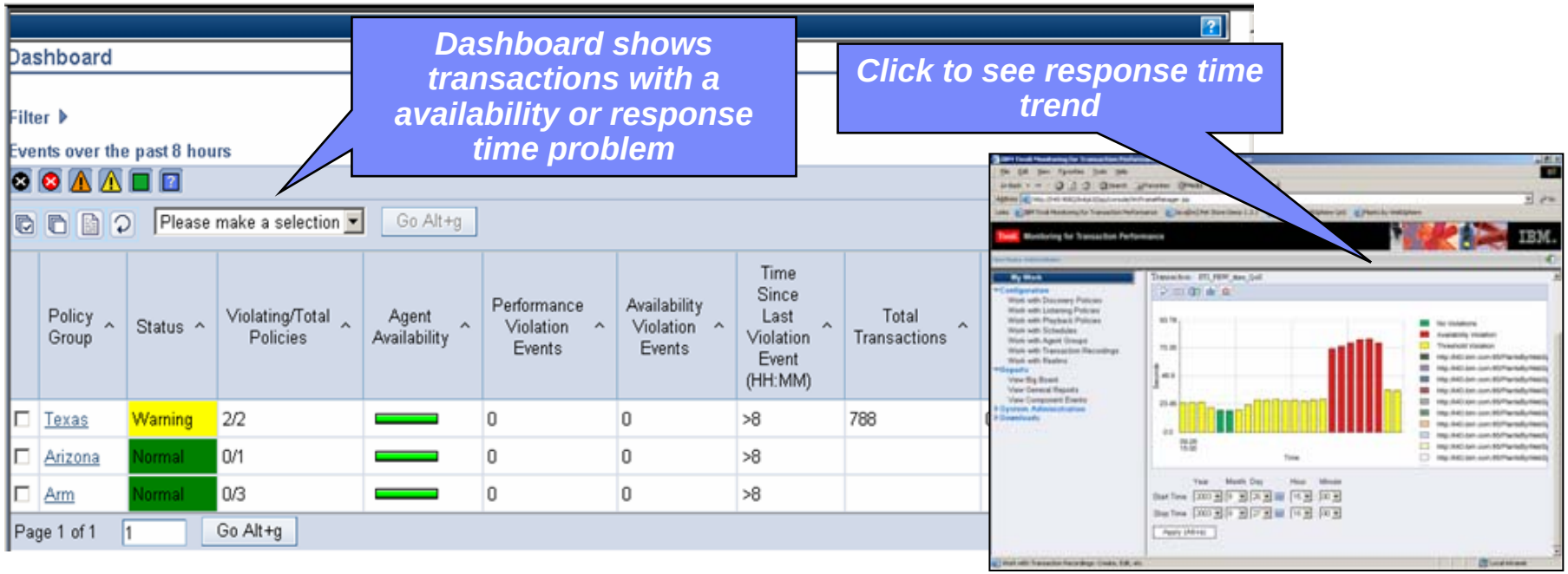
Test monitoring tools

- **Customer (End-User) Experience**
  - For SLA preparation
- **WebSphere Application Server**
  - Statistical, application trace
- **Operating System Resources**
  - For Load and SUT systems
- **Database Resources**
  - From DB and App Server perspectives
- **ESB and Message Queue Resources**
  - Queues, Channels and Brokers
- **Legacy Systems**
- **Application Systems**

Management Best Practices

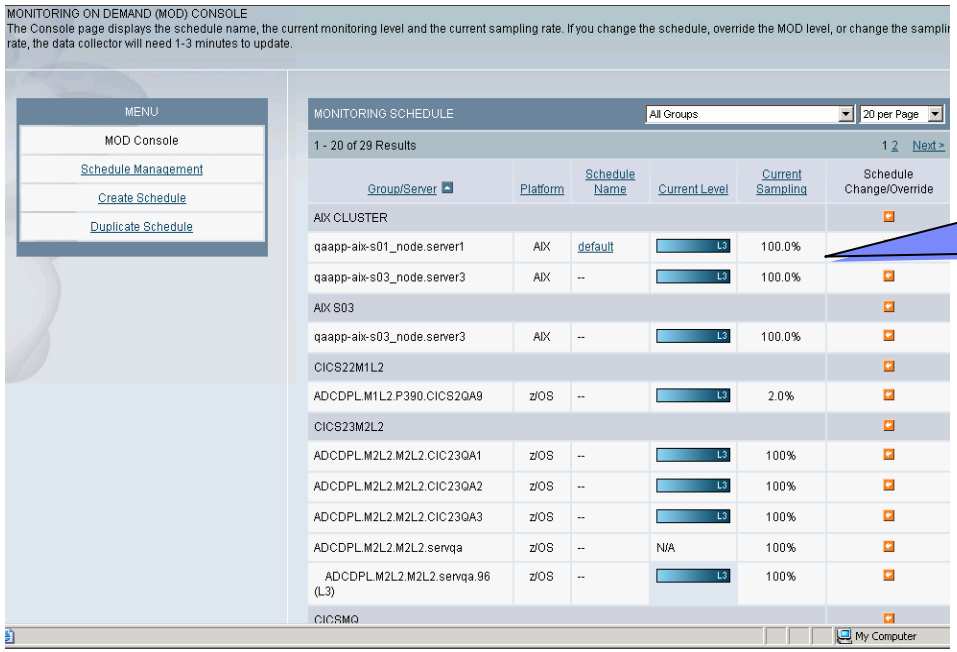
Best Practice - Spot trends before any impact to end users

- **Problem:**
  - Users report response-time problems before you know about them
  - Caught out by changes in usage patterns
- **Solution:**
  - Generate synthetic transactions to identify problems proactively
  - Historical and future trend analysis



Best Practice – Use appropriate monitoring granularity

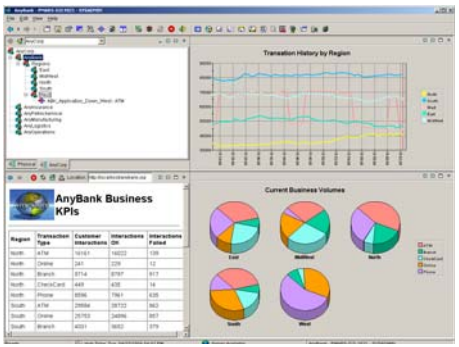
- **Problem:**
  - Need method-level tracing to determine cause of problem
  - Method-level tracing too expensive to run continuously
- **Solution:**
  - Choose tools that change monitoring levels on-the-fly
    - No need to recycle app server
    - Scheduled, triggered, targeted



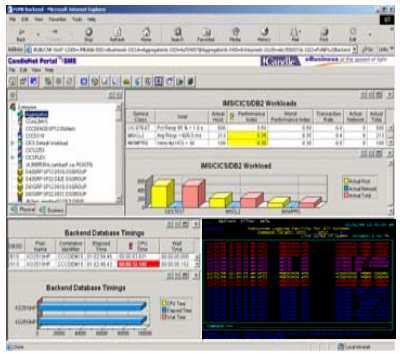
Change monitoring levels on-the-fly, without having to stop/start the application server

Best Practice – Monitor Logical views

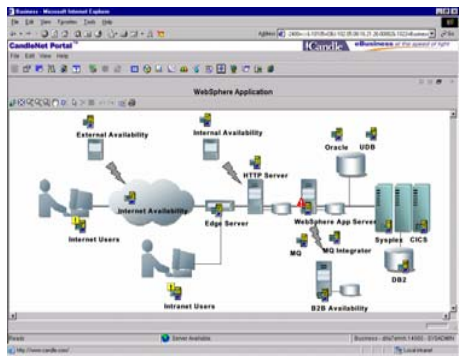
- **Problem:**
  - Too much detail – “can’t see the wood for the trees”
  - New systems added frequently – can’t keep up
- **Solution:**
  - Create logical workspaces / dashboards
  - Associate server groups with logical workspaces



Service Views



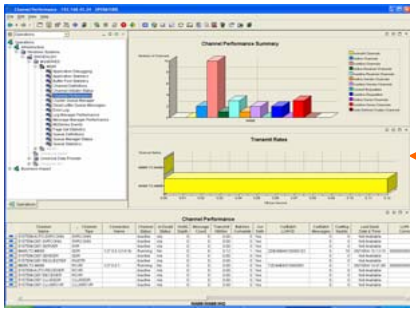
Infrastructure Views



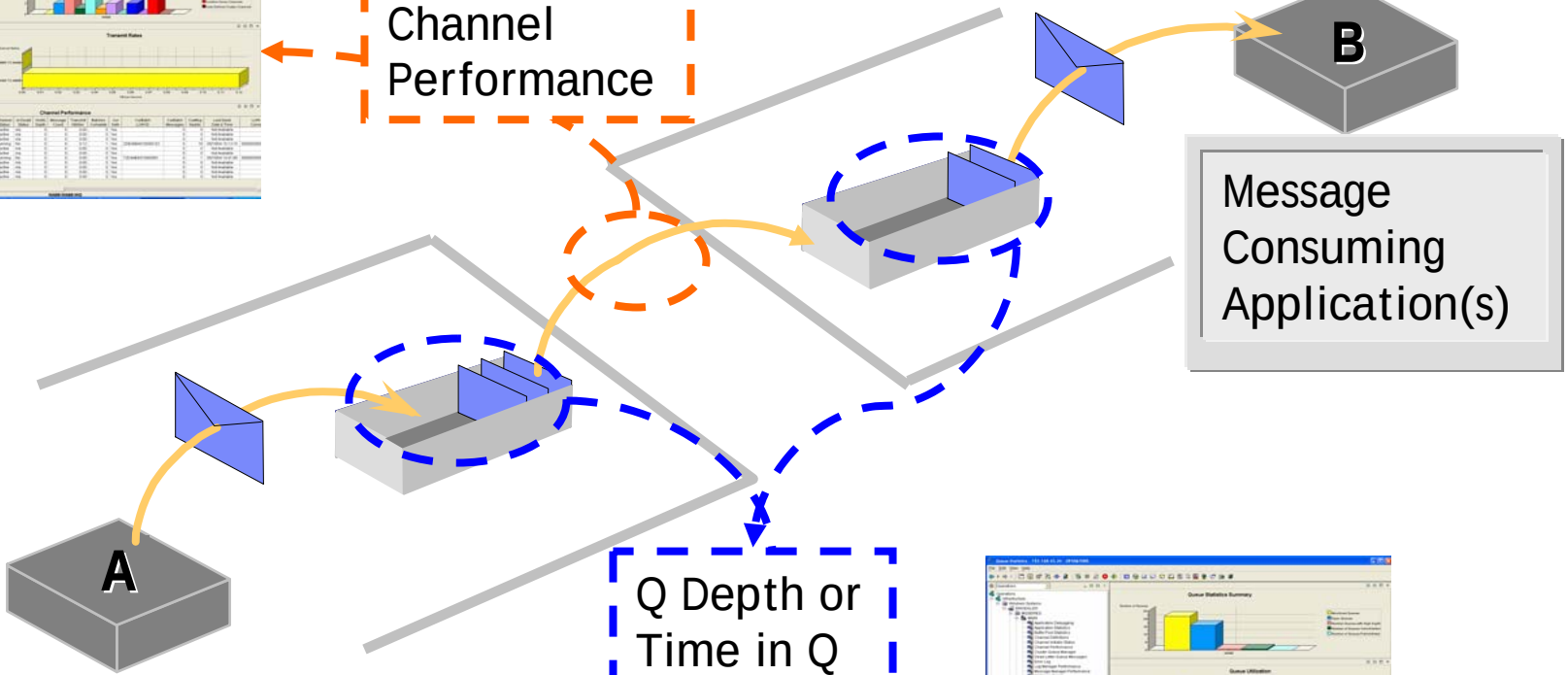
Application Views

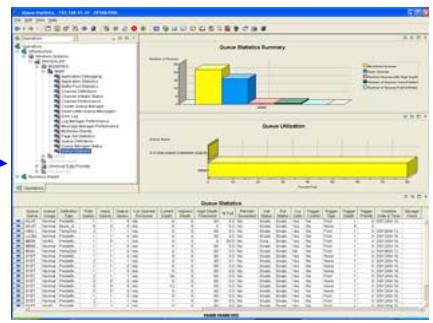
Business Views

Messaging – Key to Composite Application performance



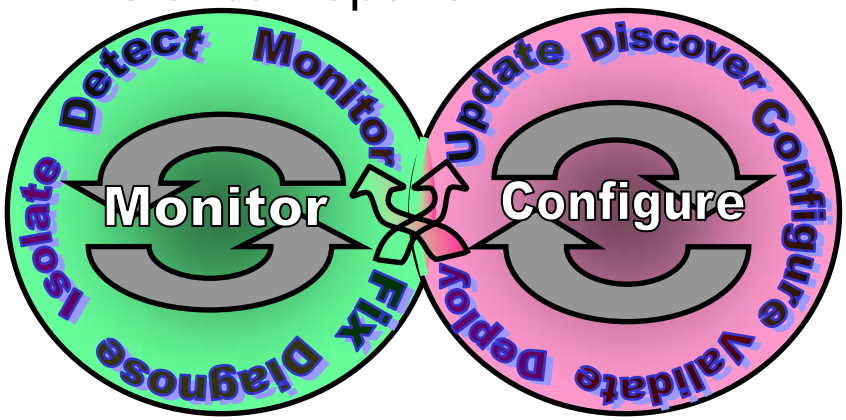
Channel Performance





WebSphere MQ – A Management Checklist

- **Monitoring**
  - WMQ health and status
  - Channel Status and performance
  - Queue Status and performance
  - Queue Sharing Group Status
  - Availability of all Queue Managers in the enterprise.
  - Queue/channel thresholds
  - Alerts and Real-time and historical reports

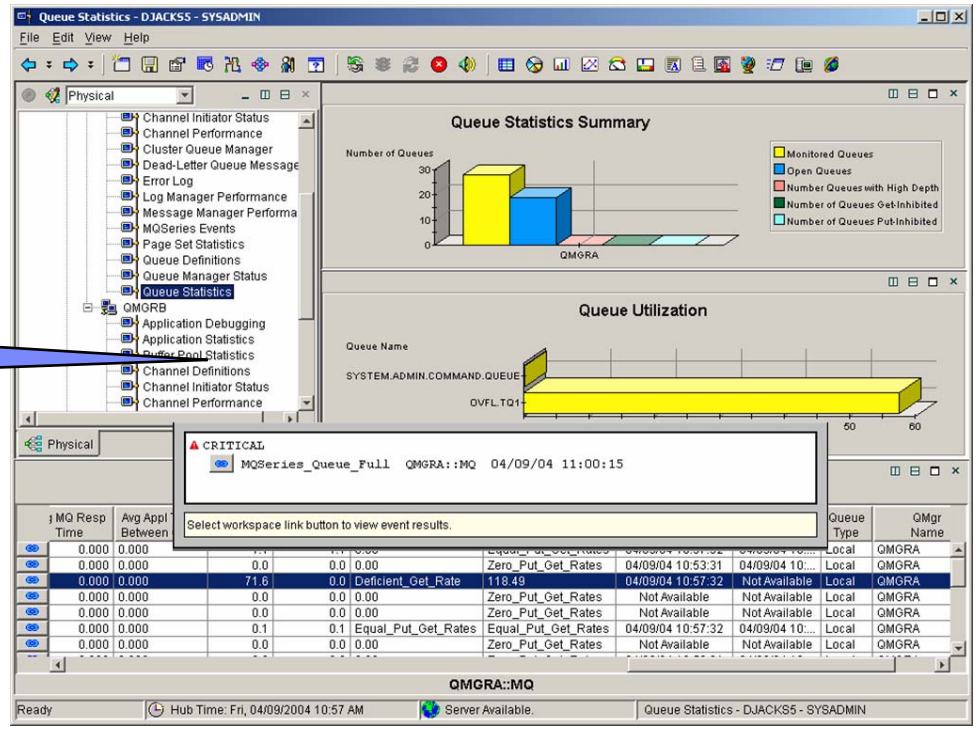


- **Configuration**
  - Discover deployed WMQ objects
  - Model and build Queue Managers and deploy throughout the enterprise
  - Configure all WMQ assets in the enterprise
  - Identify WMQ configuration discrepancies
  - Automated repair of error conditions
  - Recovery of WMQ Environments

Best Practice – Detect “Delayed" Messages

- **Problem:**
  - "Delayed" messages and “Queue Full” may indicate message consumers are not performing as expected or that message load is too great and additional instances should be created or started
- **Solution:**
  - Monitor how long messages have been resident on queues
  - Set thresholds on queues serving critical applications

Dashboard shows status and alerts for “Full” Queues or “Delayed” messages

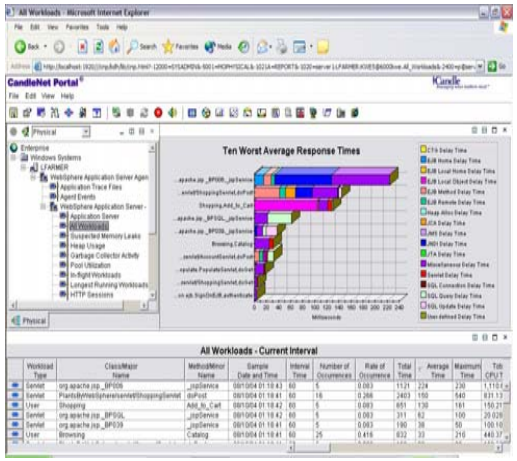


Production monitoring tools

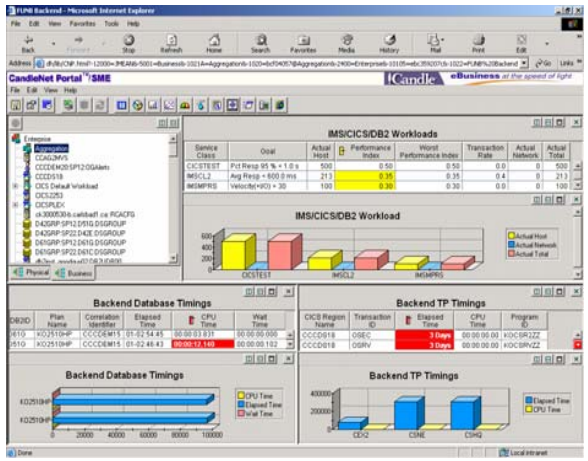
- **Customer (End-User) Experience**
  - End-To-End performance monitoring
  - Component decomposition
- **J2EE Servers**
  - Instrumentation (class loader hooks)
  - IBM API (PMI Application Server hooks) or JMX
  - JVM Profiler (Java hooks)
  - Application Server Log
- **ESB components**
  - Queues, Message and Channels
  - Message Brokers, Flows etc
- **Legacy and Application Systems**
  - CICS, IMS, SAP etc
- **Database Resources**
  - Connection Pools, Locks, Table Space etc
- **Operating System Resources**
  - Memory, CPU utilization etc.

Integration: Extensive Tivoli Enterprise Portal (TEP) workspace support

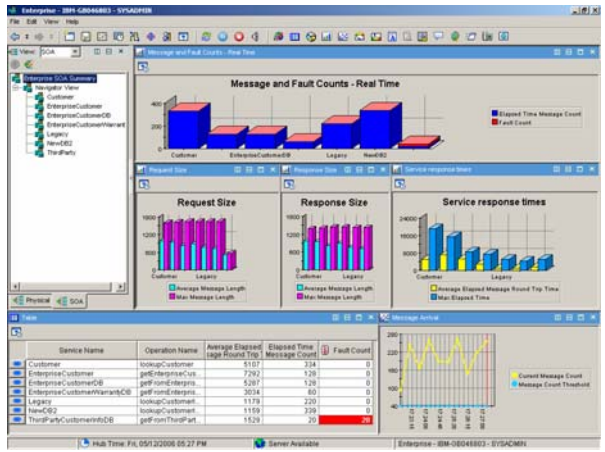
Multiple Customized Composite Application Management Views



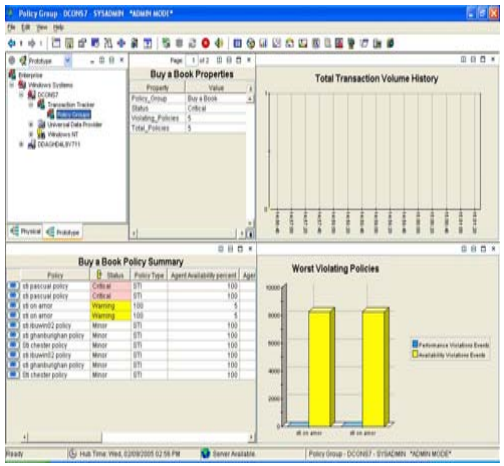
J2EE Workloads



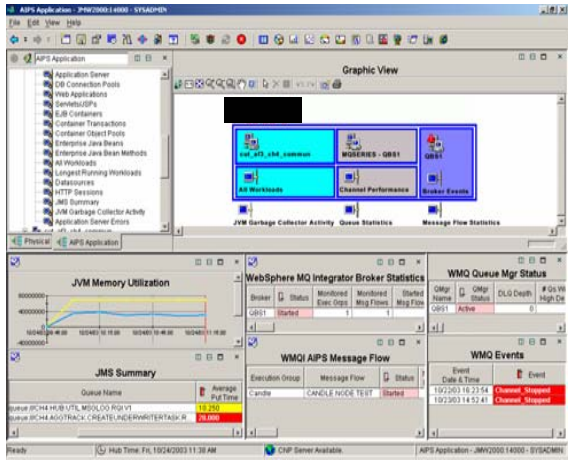
Legacy Systems



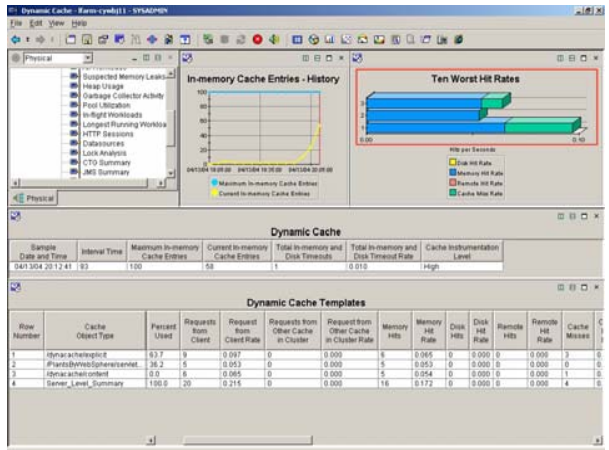
Web Services



End to End Transactions

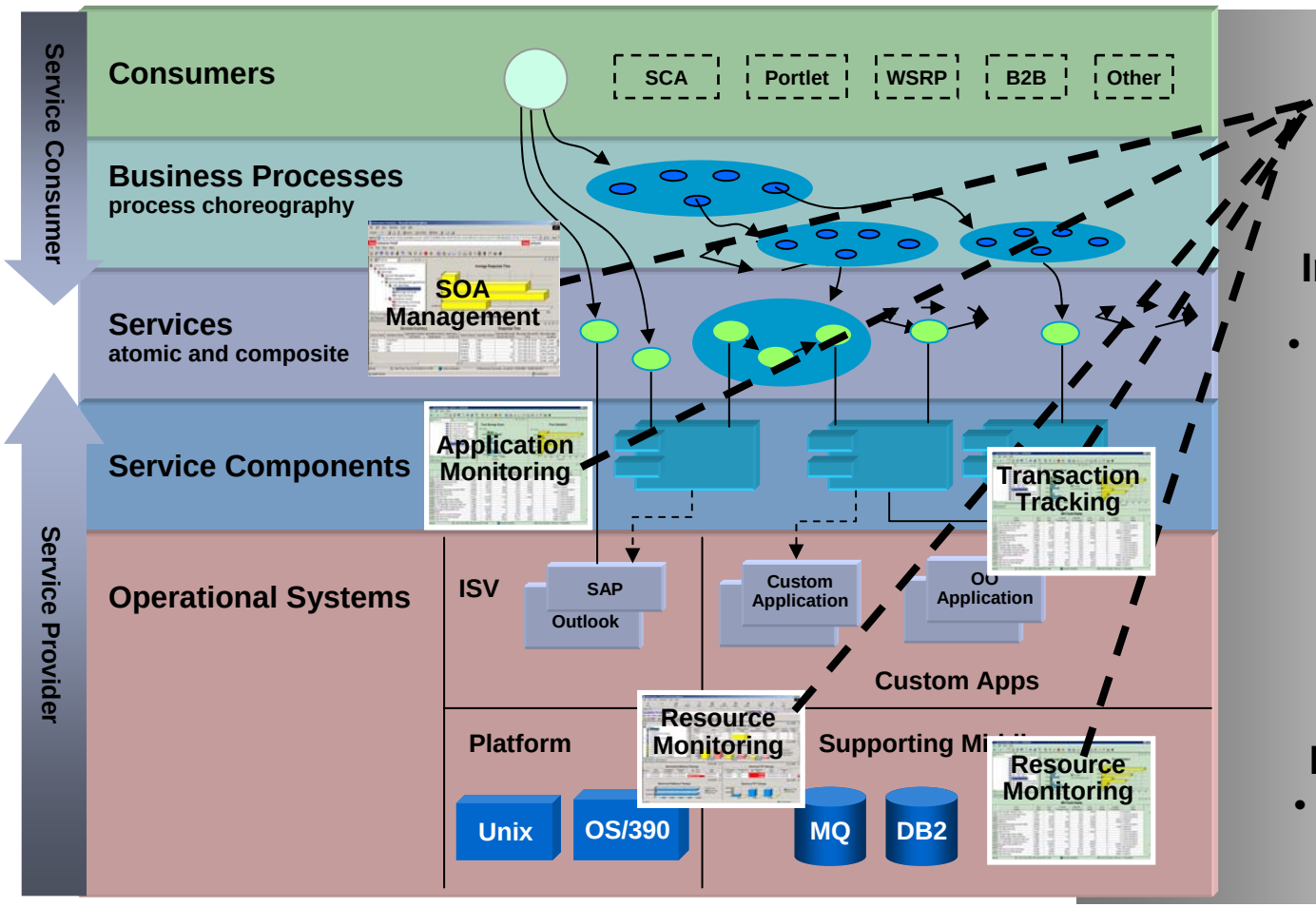


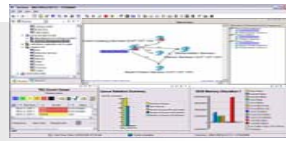
ESB Infrastructure



Databases

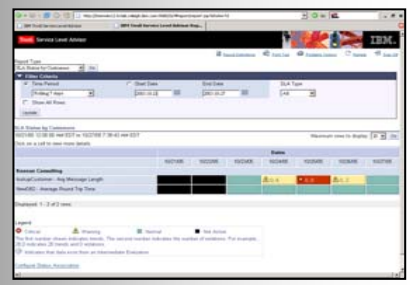
A Complete View Of Composite Application Management





Integrated Console

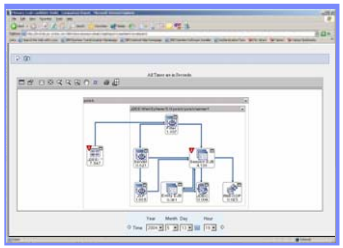
- Allow for seamless views across different layers of abstraction.



Integrated Reporting

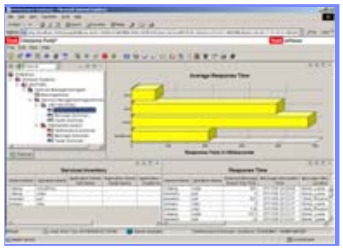
- Generate enterprise-wide service level reporting

ITCAM Family: Composite Application Management Solutions



ITCAM for Response Time Tracking (RTT)

Proactively recognizes and isolates transaction performance problems using robotic and real-time techniques and visualizes the transaction's path through your application systems, including response time contributions of each step. Supports service level management through direct feeds to Tivoli Service Level Advisor.



ITCAM for SOA

Discover, monitor, track and apply controls to web service messages. Helps IT Operators identify problems at web services layer, and provides automation capabilities for service mediation. Eclipse-based console adds subject matter expert views. Feeds Tivoli Service Level Advisor to support service-level management.

ITCAM for WebSphere

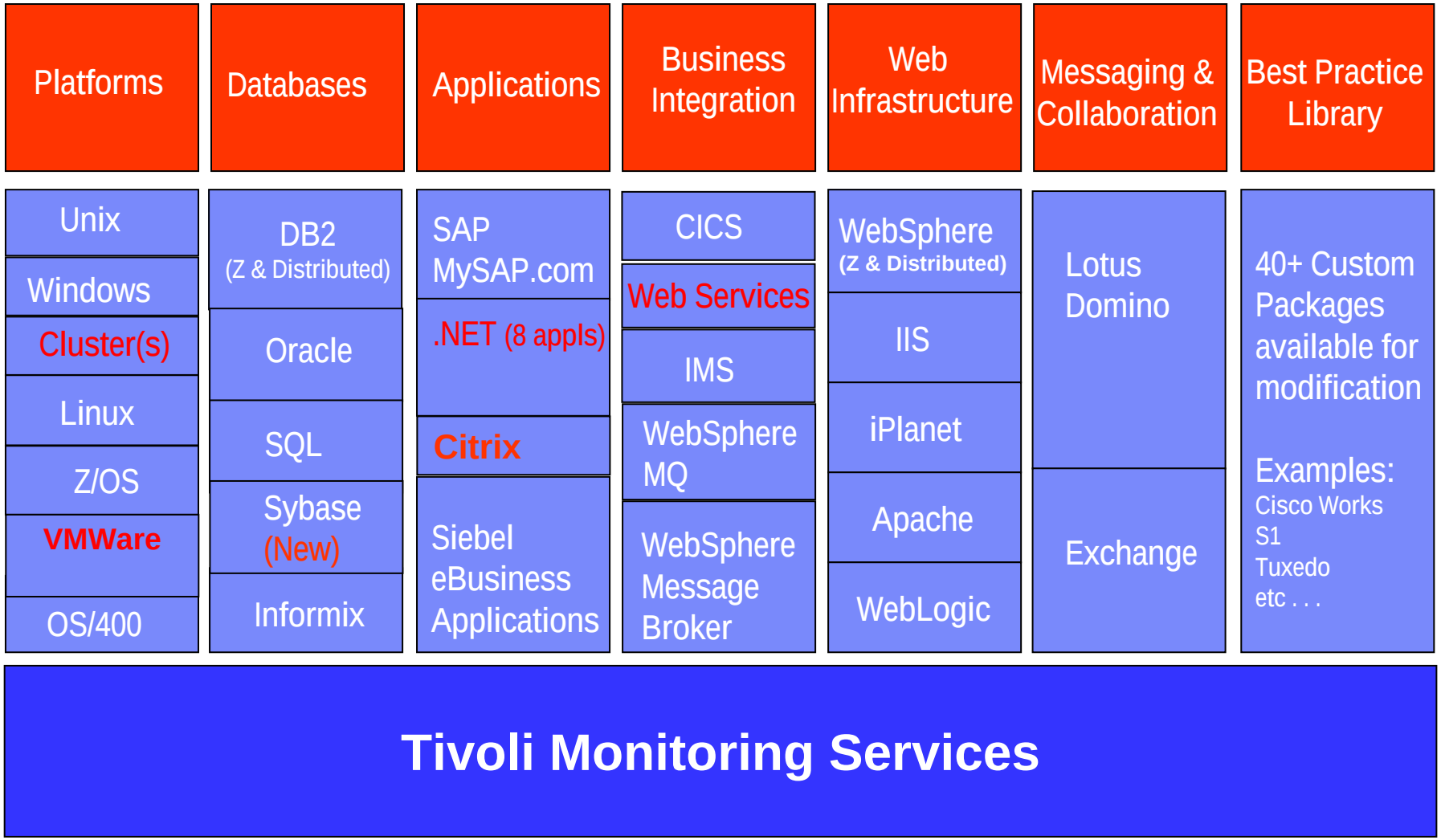
In depth analysis of J2EE applications to identify root cause of application problems. Detailed memory leak analysis for J2EE applications, and the ability to trace individual transactions that span into CICS and IMS

OMEGAMON XE for Messaging

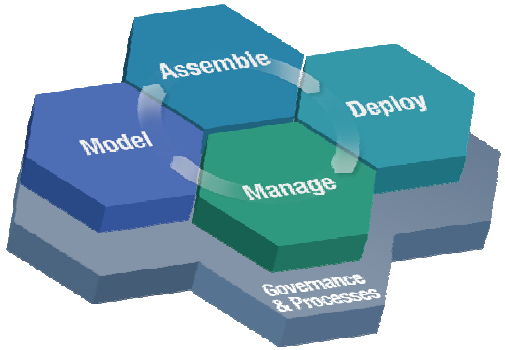
Messaging management solution that measures messaging performance, automates operational tasks, and provides problem determination for MQ environments and SOA environments in which MQ messaging is the backbone of a SOA implementation.

IBM Tivoli Monitoring

Ability to span your Composite Application environment



Summary



- **Adopt full life cycle approach**
  - Model, Assemble, **Test**, Deploy, Manage
- **Get help where necessary**
  - Best Practices – Choose them wisely – some decay rapidly
  - Monitoring Tools
  - Consulting and training
- **IBM Tivoli web site**
  - http://www-306.ibm.com/software/tivoli/

Thank You!

Questions?

Backup Material

Resources and Links

Application Types

Design Best Practices

Development Best Practices

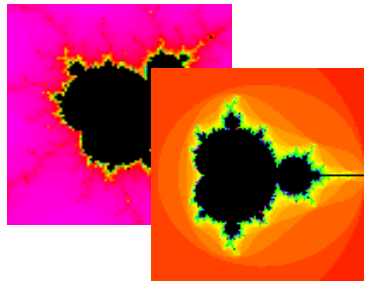
Deployment Best Practices

Resources and Links

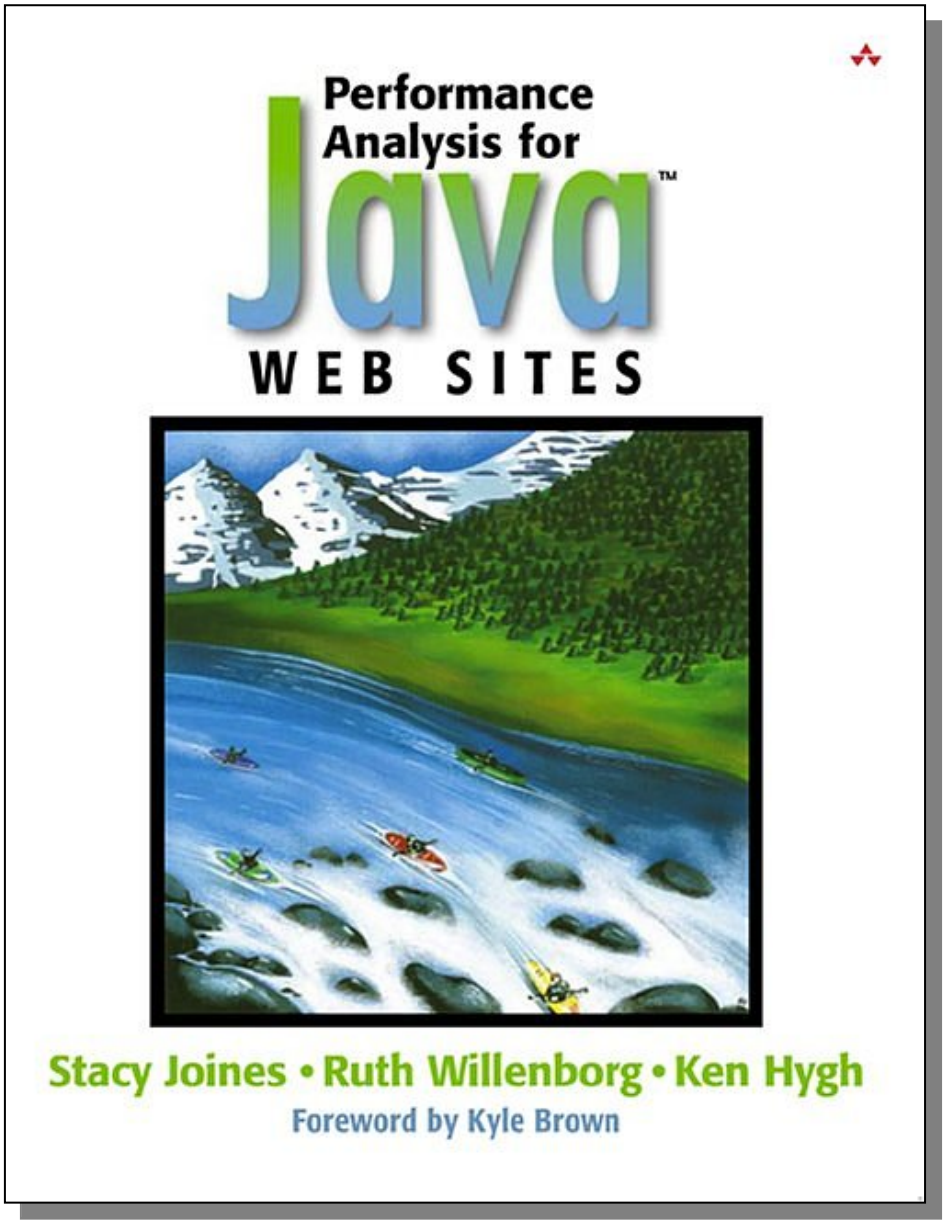
Best Practice Summary

- **Best Practice is an ongoing learning experience**
- **Plenty of resources**
  - developerWorks - IBM's resource for developers
  - WebSphere Resource usage
  - IBM Tivoli Open Process Automation Library (OPAL)
  - http://developer.java.sun.com/
  - http://www.theserverside.com/
  - http://www.jguru.com
  - Patterns
    - http://developer.java.sun.com/developer/technicalArticles/J2EE/patterns/
    - IBM Patterns for e-business
  - Anti-patterns
    - http://www-128.ibm.com/developerworks/webservices/library/ws-antipatterns

Design Patterns



- **Design Pattern and Principles**
  - Google *SOA Patterns*
- **Anti-patterns**
  - What not to do / How not to do it
    - http://www-128.ibm.com/developerworks/webservices/library/ws-antipatterns/



Read about Test Plans & Test Scripts Best Practices

Backup – Application Types

Online Shopping

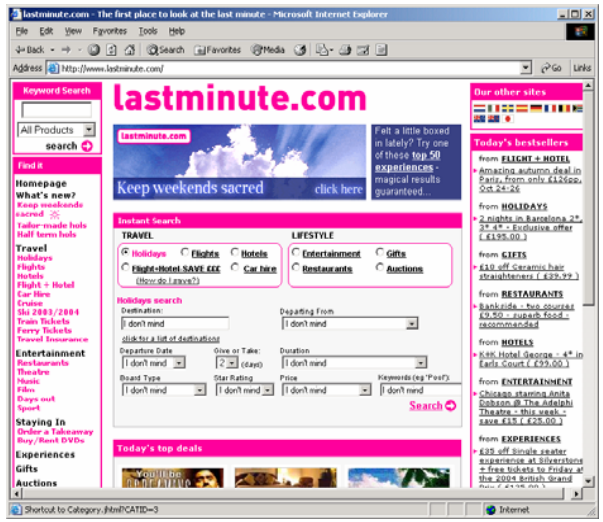
- **Low transaction rates**
- **High levels of “view-only” operations**
- **High traffic volumes for larger sites**
  - Daily and seasonal variations
- **Large numbers of concurrent users**
- **Large returned page sizes**
- **Stable, time in-sensitive data**
- **Relatively short session durations**
- **High caching potential**
- **Small number of secure pages**
  - Use minimum graphics
- **Need a large test database**
  - Ensure database is stressed
    - Eliminate database cache with variety
  - Use copy of production database

Information Sites

- **Highest number of page hits**
  - Some sites > 1M hits/minute
- **Regular content changes**
- **Usage spikes for interesting content**
- **Low security risk**
- **Very few transactions**
  - E.g. paid access to certain pages
- **Limited legacy connectivity**
- **Very high caching potential**
  - Cache refresh depends on data volatility
- **No security issues**
  - Very limited SSL usage
- **Need wide client base for testing**
  - Clients have little or no session data
  - Can reduce “think-time” at the test client

Self Service Portals

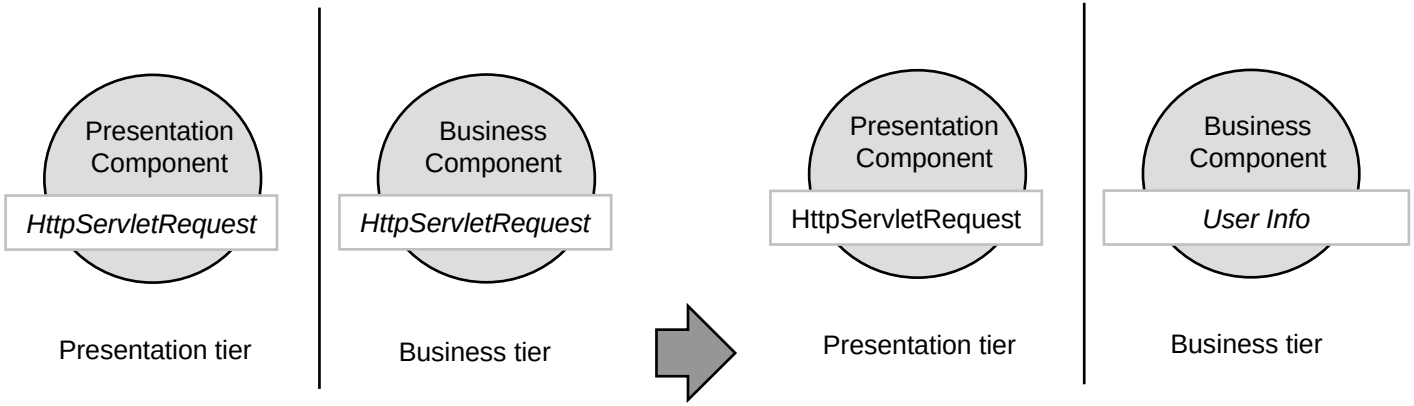
- **Personalized to a specific customer**
- **Potentially high security considerations**
  - E.g. home banking
- **Heavy reliance on legacy systems**
- **Medium transaction rates**
  - Increasing as more people use the web
- **Caching potential medium to low**
- **Security is high for certain types**
  - e-banking – long secure sessions
- **Include simulations with many users**
  - Users typically not very busy



Backup – Design Best Practices

Presentation Tier Best Practices

- **Problem:**
  - Exposing presentation-tier data structures (like HttpServletRequest) to business-tier
    - Increases coupling and reduces reusability.
- **Solution:**
  - Implementation details specific to one tier should not be introduced in another tier.
  - Copy the relevant state and data into more generic data structures and share those. (Transfer Object Pattern)



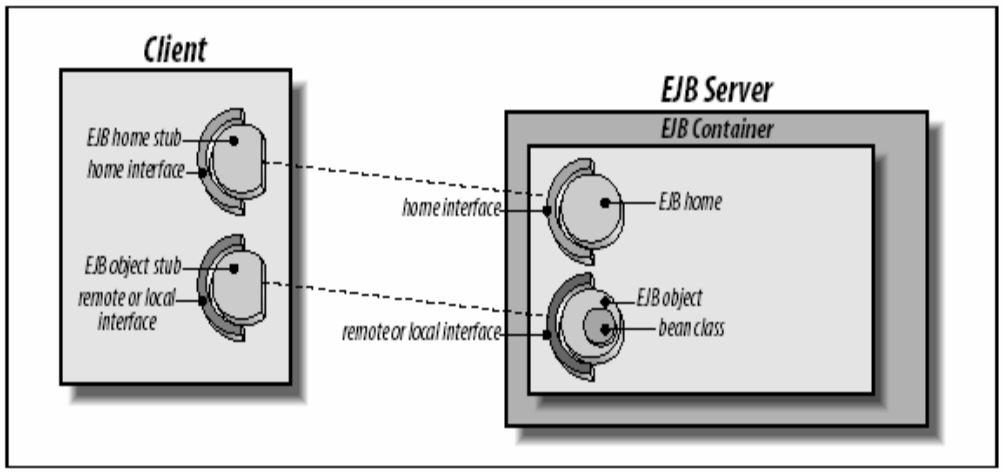
SOA Principles

- ***Service reusability*** **– Logic is divided into services with the intention of promoting reuse.**
- ***Service contract*** **– Services adhere to a communications agreement, as defined collectively by one or more service description documents.**
- ***Service loose coupling*** **– Services maintain a relationship that minimizes dependencies and only requires that they maintain an awareness of each other.**
- ***Service abstraction*** **– Beyond what is described in the service contract, services hide logic from the outside world.**
- ***Service composability*** **– Collections of services can be coordinated and assembled to form composite services.**
- ***Service autonomy*** **– Services have control over the logic they encapsulate.**
- ***Service statelessness*** **– Services minimize retaining information specific to an activity.**
- ***Service discoverability*** **– Services are designed to be outwardly descriptive so that they can be found and assessed via available discovery mechanisms.**

Credit: Thomas Erl

Business Tier Best Practices

- **Problem:**
  - Each time you access an component like an EJB, you need to get handles and references to these objects
  - Costly round-trips over the network.



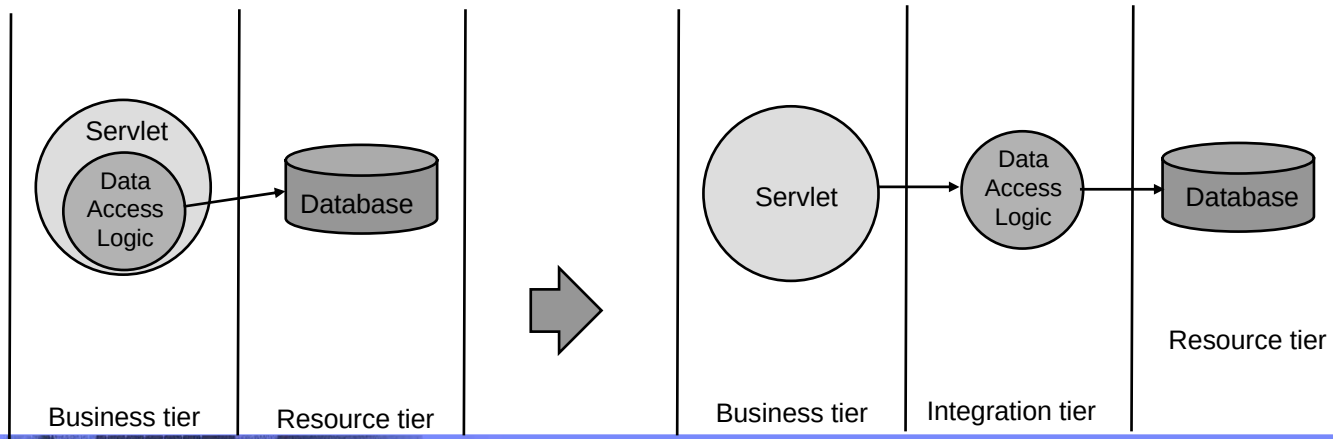
- **Solution:**
  - Cache these references (like EJBHome references) and handles.
  - Use patterns like Business Delegate and Service Locator.
  - The business delegate only needs to make the JNDI lookup the first time.
  - Use a Business Delegate to encapsulate access to a business service (hides the implementation details)

Business Tier Best Practices

- **Problem:**
  - Exposing all enterprise bean attributes via Get/Set methods.
  - Each method call is potentially remote, resulting in network overhead.
- **Solution:**
  - Use a value object (Transfer Object Pattern) to transfer aggregate data to and from the client.

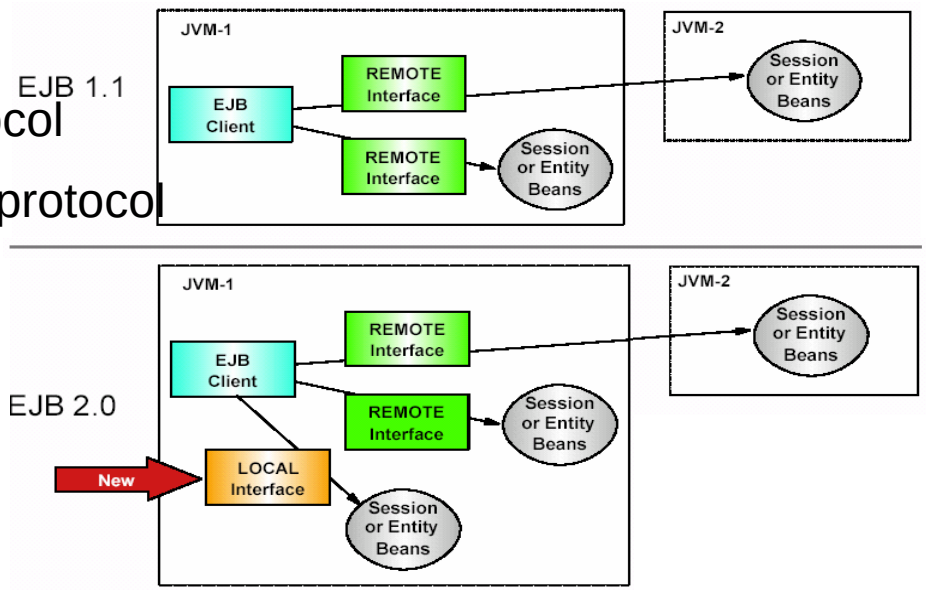
Data Access Best Practices

- **Problem:**
  - Data access logic is embedded directly within a class that has other unrelated responsibilities
- **Solution:**
  - Extract the data access code into a new class, an Data Access Object (DAO) class.
  - This encapsulates the data access and manipulation (Data Access Object Pattern)



Local EJB Interfaces

- **Formerly all EJBs must be considered remote**
  - Even if they were in the same JVM
  - Expensive in terms of time and memory
- **Now we can optionally declare Local Interface for an EJB**
  - EJBs within the same JVM can access each other directly
  - Pass-by-reference (not pass-by-values) semantics
- **Beans can be both local and remote**
  - A “local” interface defines the local protocol
  - A “remote” interface defines the remote protocol



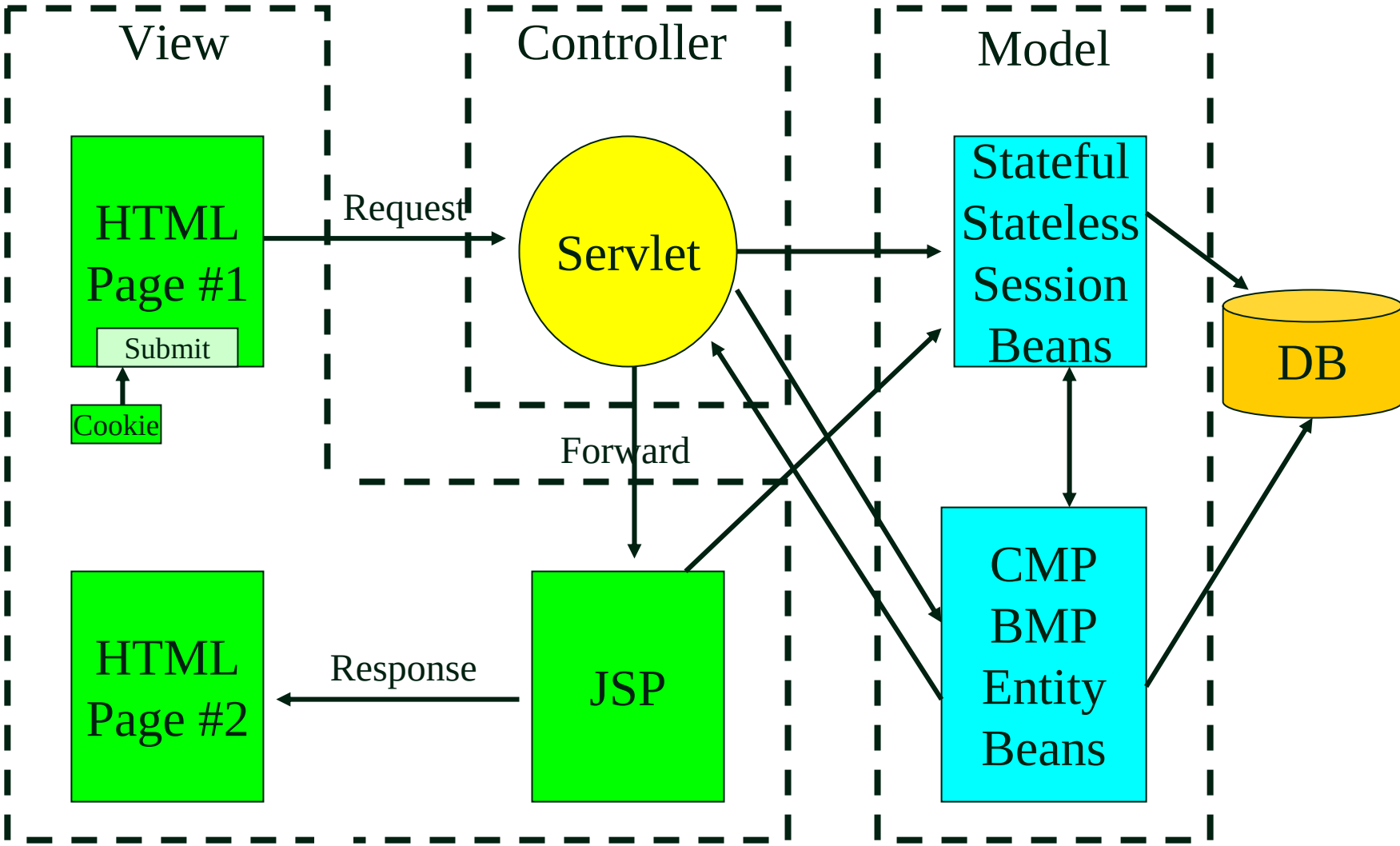
Warning: limits the web site scalability

Backup – Development Best Practices

Use appropriate frameworks

- **Model View Controller (MVC)**
  - STRUTS from Apache is popular
  - Model – responsible for the data or business process, business logic (Java Beans, EJBs)
  - View – responsible for the look and feel, GUI
  - (Java Server Pages JSPs)
  - Controller – is responsible for the flow of the application, traffic cop (Servlets)

Model-View-Controller Architecture



Static vs. Dynamic Pages

- **Use external HTTP Server**
- **Use WebSphere plug-in for routing requests**
  - HTTP server serve up static pages
  - Application server's Web Container serve up dynamic pages
  - Allows requests to be workload managed across clones
    - 1st checks for session affinity to route to a cloned application server
    - 2nd load balancing is used to route to a cloned application server
- **Avoid having "file serving servlet" serve up static pages**

Servlets

- **Drive application flow (Controller)**
  - JSP's generated by App Srv are memory efficient
- **Should serve up only a few web pages**
- **Small, lightweight Servlets <50 Lines of code**
- **Grab static resources during initialization init()**
  - JNDI context lookup for data sources, thread safe
- **Multi-threaded model, avoid single-threaded**
  - Do not use instance variables to hold state
- **Remove unneeded synchronize statements**
- **Synchronize on short sections of code**

Java Server Pages

- **Provide presentation content (View)**
- **Use Custom Tags rather than Java code**
- **Use HTML and XML for web pages**
- **Compiled into servlets so Pre-Compile them**
- **Eliminate default HTTP session creation**
  - <% page session="false" %>
- **Minimize <jsp:include> tag**

Logging

- **Have multiple logging levels**
- **Make logging levels configurable at runtime**
- **Keep production logging at a minimum**
- **Use “buffered writing” for logs**
- **Keep buffer memory minimal**
- **Avoid use of “servlet.log()” method calls**
- **Avoid use of “System.out.println” method calls**
- **Use third party logging frameworks**

HTTP Sessions

- **Keep session objects small <2KB**
- **Persist session data, index using Session ID**
- **Pass Session ID between Servlets**
  - Cookies
  - URL rewriting
  - Hidden form fields
- **Persist session data only for life of Session**
- **Make HTTP session data serializable**

HTTP Sessions (cont..)

- **Avoid using nontransferable data**
  - “transient” for variables prevent serialization
- **Do not create HttpSession objects by default on JSPs even though it is part of J2EE spec**
  - <%@page session=”false”%>
- **Reduce HTTP session timeout value**
  - Check for timeout set in code
- **Provide logout function on Web pages for closing session immediately**

HTTP Sessions (cont..)

- **Consider Memory-to-Memory Replication**
  - Single Replica
  - N-way peer-to-peer
  - Client/Server
  - Persist on timer, each request, on demand
  - Can be encrypted

Enterprise Java Beans

- **Ensure you need distributed enterprise objects**
- **Avoid stateful session beans & remove unused ones**
  - Stateful non-web based clients
- **Use stateless session bean facades**
  - do not have servlets call multiple entity beans directly
    - Expensive remote method calls
    - Transactional, 2 database calls (get data, update database)
- **Access Entity beans from Session beans**
  - Reduces the number of remote method calls
  - Single transaction context

Enterprise Java Beans (con't)

- **Use local interfaces for co-located beans**
  - This must be part of initial architecture
- **Use caching**
  - Homes - Reduce expensive JNDI calls
  - If using custom finders
- **Use CMP 2.0 where possible**
  - Better performance and helps referential integrity
  - Portable over multiple databases
  - Easier to develop and maintained by container
  - If you need BMP, sub-class CMP 2.0 bean
- **Transactions**
  - Abort transactions with setRollbackOnly() method
  - Use declarative rather than programmatic

Enterprise Java Beans (con't)

- **Read-Only getter methods**
  - Consider group methods
- **Cache EJB Home**
  - Reduce expensive JNDI calls
- **Cache Bean content in custom finders**
- **Reduce Tx Isolation Level**
  - TRANSACTION_READ_UNCOMMITTED
  - TRANSACTION_READ_COMMITTED
  - TRANSACTION_REPEATABLE_READ
  - TRANSACTION_SERIALIZABLE

Database Connection Pools

- **Use database connection pools**
- **Use JNDI lookups for naming**
- **Use vendor supplied JDBC drivers**
- **When finished with database connections return them to pool immediately**
- **Verify exceptions do not leave connections open**
  - Return connections in “finally” clause
- **Set up data-source once and cache it**
  - Object initialization (Servlet.init(), ejbActive())

Object pools

- **If application launches threads, create thread pool**
- **Set configurable limits**
- **Avoid waiting indefinitely for an Object**
  - Gives appearance of frozen web site, display error page
- **Set “hard maximums”**
  - Don’t create indefinitely
- **Set timeouts for retrieving connections**
  - Gives appearance of frozen web site, display error page

Efficient Java Code

- **Minimize creation objects, pool and reuse them**
  - Check condition statements and loops
- **Write efficient code, reduce memory and leaks**
  - Avoid creating Strings use StringBuffer instead
  - Use System.arraycopy()
  - Remove hash tables when emptied
  - Remove unused local variables
  - Size collections appropriately
- **Use static and final variables**
  - use static finals, declared only once
  - Place them in singletons, declared only once

Efficient Java Code (cont…)

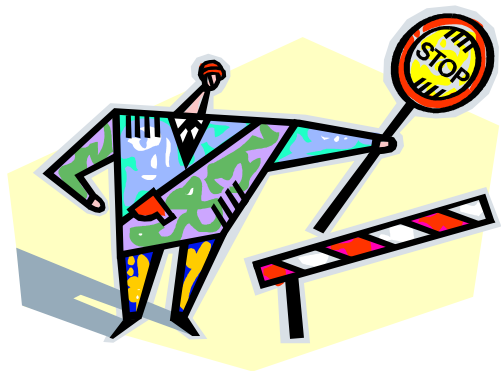
- **Determine if reflection is really necessary**
  - Elegance vs. performance
- **Avoid very complex class structures and hierarchies**
- **Use buffered IO for small amounts of data**

```
Writer writer = new BufferedWriter(new
      FileWriter(new File("file.txt")));

Writer.write(“Hello World”);

Writer.close();
```


Backup – Deployment Best Practices

High Availability

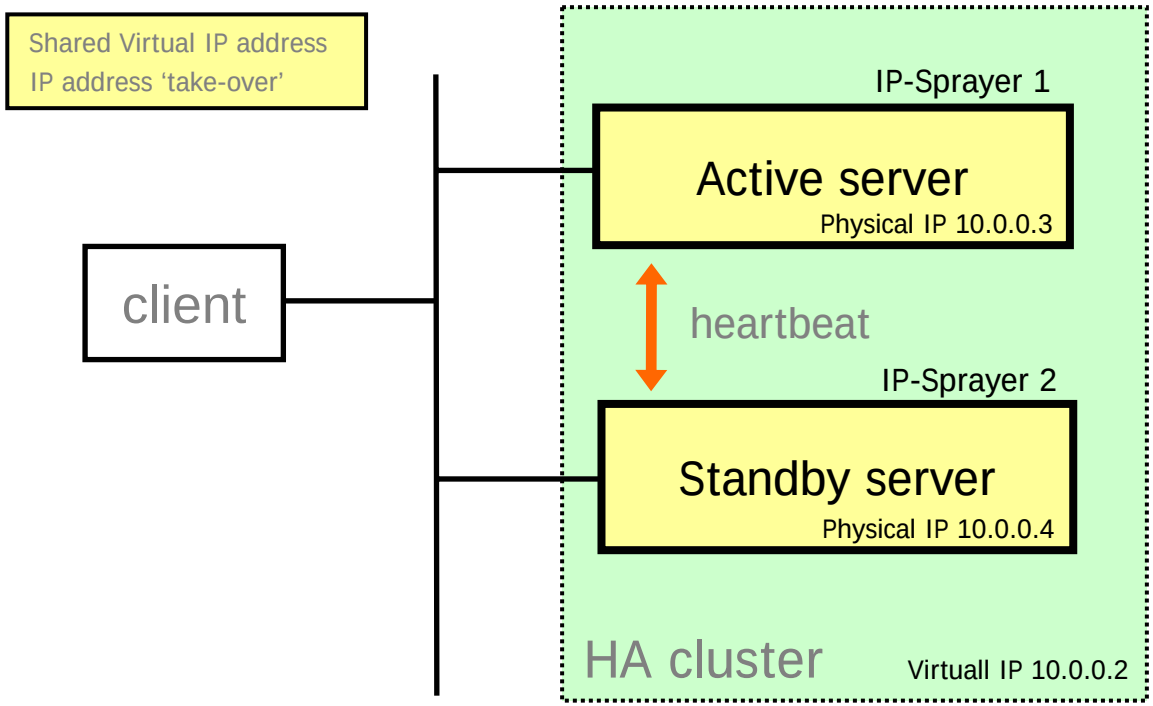
- **An application or service that is provided in a near-continuous fashion (can never guarantee 100%)**
  - **Failures**
  - Disk failure
  - Machine crash
  - Process failure
  - Network failure
  - Human error
  - Planned maintenance
  - Loss of the data center



Application must be designed and built to be scalable

The HA Pattern

- configure two or more load balancers
- one LB monitors the other and takes over load distribution if the primary LB fails

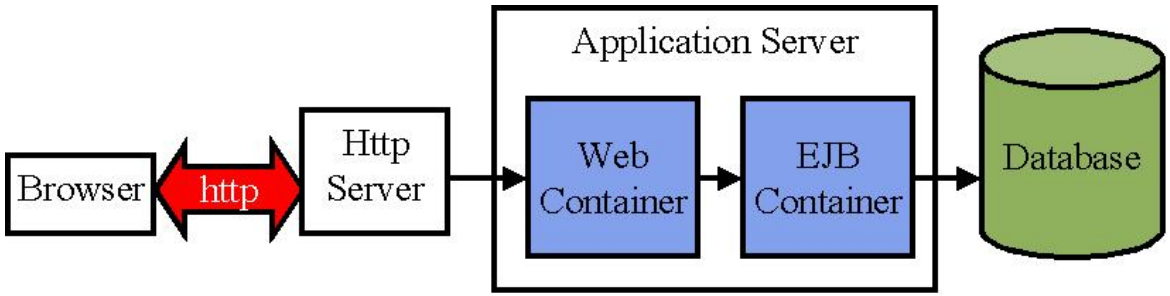


IBM WebSphere Edge Server Network Dispatcher

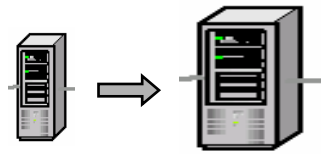
N-tier Model

Tier

- a logical partition of the separation of concerns in the system
- assigned its unique responsibility in the system
- is loosely coupled with the adjacent tier

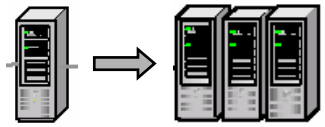


Vertical Scaling



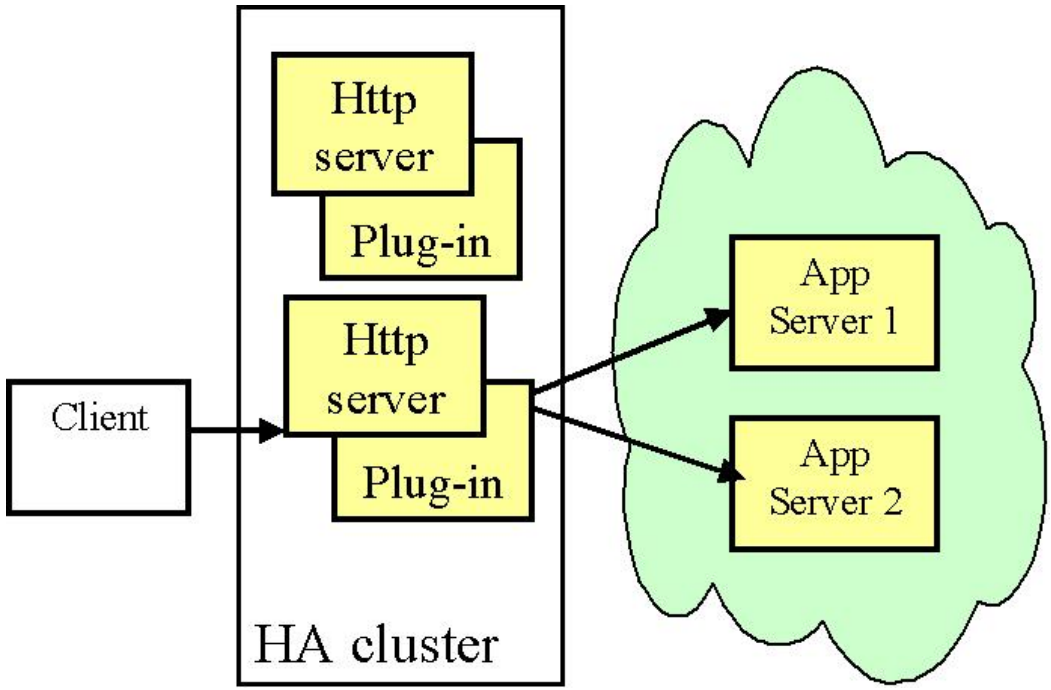
- **“Vertical” scaling (Scaling up)**
  - The addition of computational and system resources, e.g. adding CPU’s or memory to a single machine
- **Strengths**
  - Improved throughput on large machines (up-size)
  - Easy to maintain and support
- **Weaknesses**
  - Limited available resources on single machine
  - Decreasing return on investment in throughput
- **Threats**
  - Hardware as Single point of failure
  - Memory/CPU overhead if have too many clones
- **Opportunities**
  - Can be combined with other topologies

Horizontal Scaling



- **“Horizontal” scaling (Scaling out)**
  - The server application is distributed across more than one machine instead of replacing an existing machine with a more powerful model
- **Strengths**
  - Distribution of load among machines
  - Failover support if one machine goes down
  - Increase in throughput is closer to linear than Vertical Scaling
- **Weaknesses**
  - More difficult to setup and maintain
  - More $$ for hardware and software
- **Threats**
  - Requires session persistency
  - Increased network traffic between machines
- **Opportunities**
  - Can increase number of machines as needed, highly scalable
  - Can be combined with Vertical Scaling

High Availability Cluster



plugin-cfg.xml

<PrimaryServers>
<Server Name="HAClusterServer1"/>
<Server Name="HAClusterServer3"/>
</PrimaryServers>
<BackupServers>
<Server Name="HAClusterServer2"/>
<Server Name="HAClusterServer4"/>
</BackupServers>

Clustering software such as HACMP, MC/Serviceguard, MSCS

Web Application Component Issues

- Web Containers
- HTTP Session State
- EJB Containers
- Thread pools
- JVM Heap Usage
- Application Server Clones
- HTTP Servers
- Database Servers
- Network Components

Web Containers

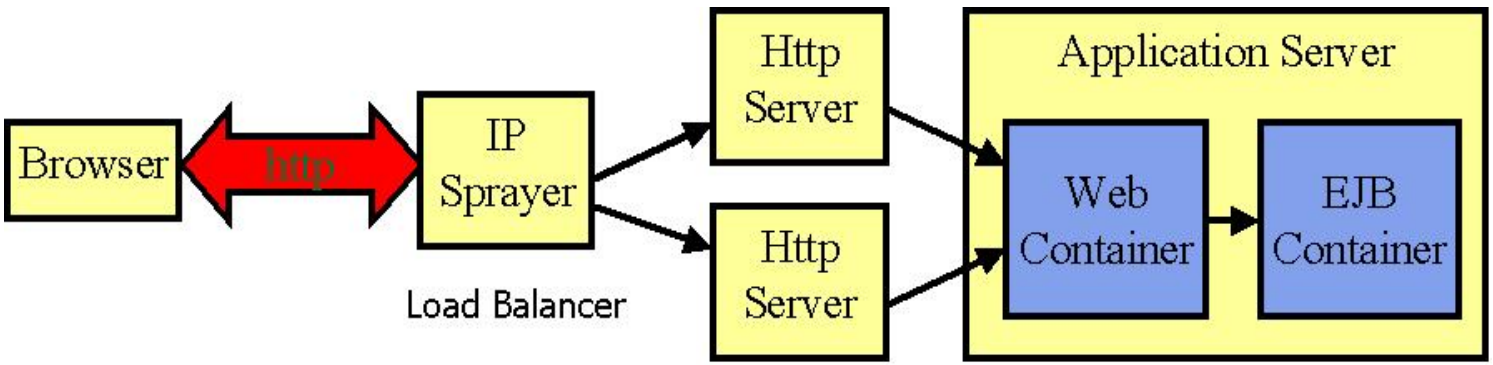
- **Load servlets at startup**
- **Precompile all JSPs so servlets get created**
- **Turn off reload interval for Servlets**
- **Turn off reload interval for JSPs**

HTTP Servers

- **Configure number threads and process appropriately**
- **Set timeouts accordingly**
- **Verify Logging Level set correctly**
- **Verify keep-alive set correctly**
- **Only use SSL (Secure Socket Layer) when necessary, overhead**

Http Server Cluster

- Load balancers (”IP Sprayers”) takes incoming requests and distributes them across muliple servers.
- uses routing algoithms
  - round-robin
  - weighted load distribution algorithm
- needs to know about availability and handle ”affinity routing”
- Some vendors* of load balancers implements the LB in hardware, others provide LB software

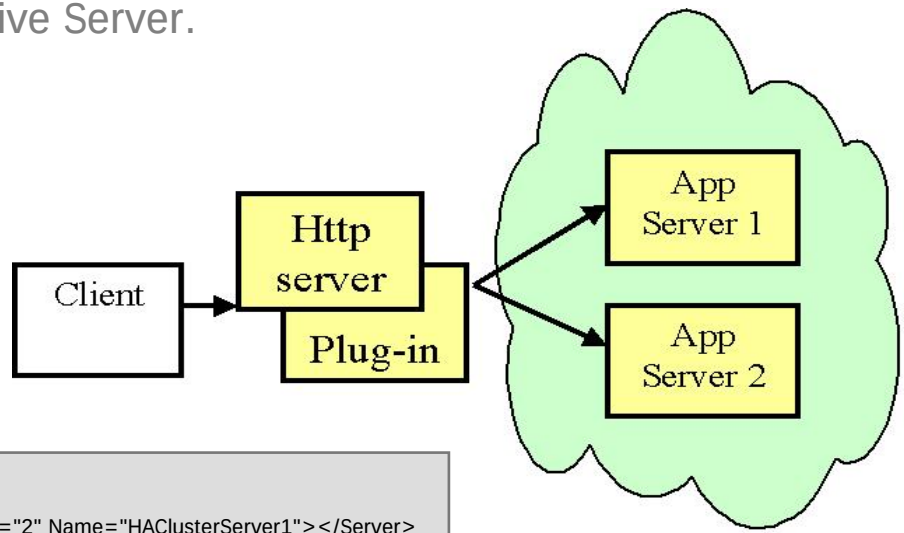


**) IBM WebSphere Edge Server Network Dispatcher*

Routing of Servlet Requests with HTTP plug-in

HTTP-plugin:

- mechanism used to route HTTP requests to WebSphere application servers
- provide distribution of requests and failover
- handshaking: by adding special header information to the HTTP request and response.
- uses an XML configuration file to determine information about the WebSphere domain it is serving. This configuration file is initially generated by the WebSphere Administrative Server.

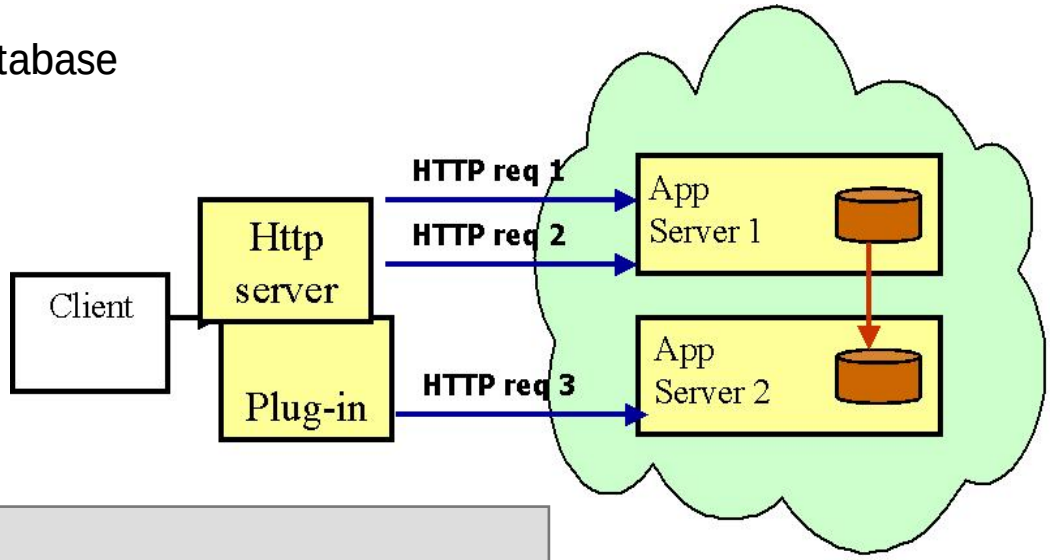


plugin-cfg.xml

```
<ServerCluster Name="HACluster">
<Server CloneID="u307p2vq" LoadBalanceWeight="2" Name="HAClusterServer1"></Server>
<Server CloneID="u307p48r" LoadBalanceWeight="3" Name="HAClusterServer2"></Server>
<Server CloneID="u307p62u" LoadBalanceWeight="4" Name="HAClusterServer3"></Server>
<Server CloneID="u307p71m" LoadBalanceWeight="5" Name="HAClusterServer4"></Server>
</ServerCluster>
```

Http Session Storage

- manage HTTP-session affinity
- maintain information about a client across multiple HTTP requests
- each application server is assigned a unique cloneID
- the plugin will bind cloneID and client's sessionID when it receives the client's first request
- a user's session information can be kept on a server in two ways:
  - in memory
  - persistent to database

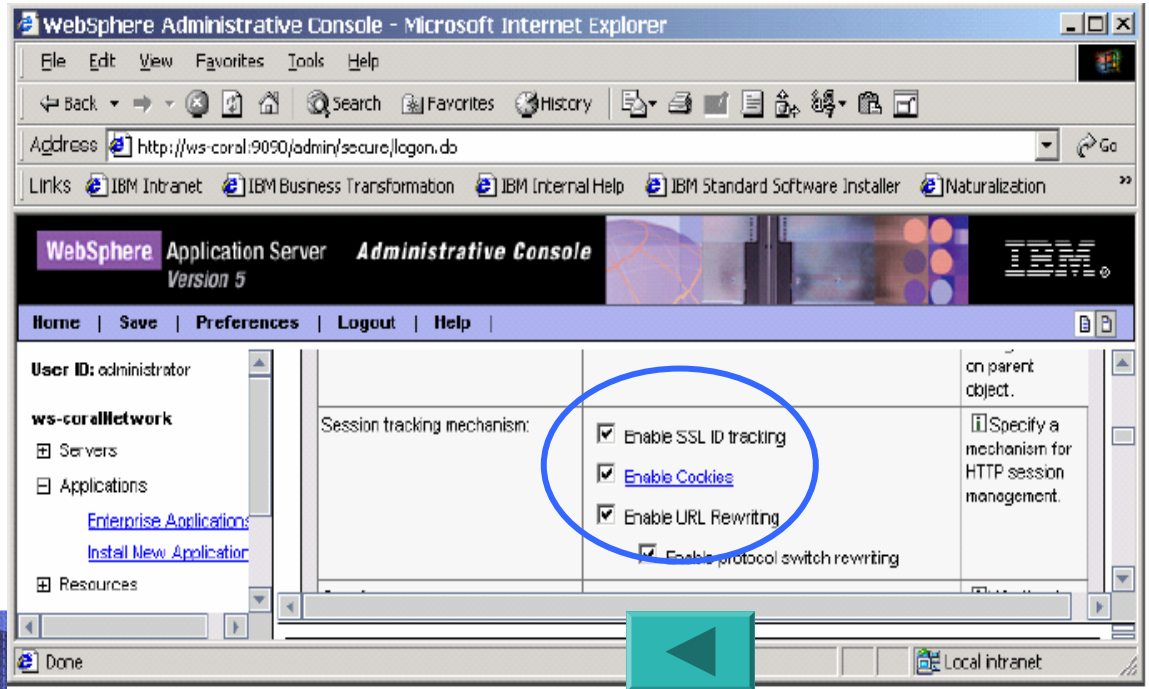


plugin-cfg.xml

```
<ServerCluster Name="HACluster">
<Server CloneID="u307p2vq" LoadBalanceWeight="2" Name="HAClusterServer1"></Server>
<Server CloneID="u307p48r" LoadBalanceWeight="3" Name="HAClusterServer2"></Server>
<Server CloneID="u307p62u" LoadBalanceWeight="4" Name="HAClusterServer3"></Server>
<Server CloneID="u307p71m" LoadBalanceWeight="5" Name="HAClusterServer4"></Server>
</ServerCluster>
```

Http Session Storage

- manage HTTP-session affinity
- maintain information about a client across multiple HTTP requests
- each application server is assigned a unique cloneID
- the plugin will bind cloneID and client's sessionID when it receives the client's first request
- a user's session information can be kept on a server in two ways:
  - in memory
  - persistent to database



Enterprise Java Bean Containers

- **Configure EJB pool size large enough to handle maximum EJB requests**
- **Set the cleanup interval appropriately for application**
- **Use Local interface when available**
  - Available from EJB 2.0 specification
  - Requires previous knowledge EJBs and where located

Application Server Clones

- **Clone servers that are being under utilized based on CPU utilization**
- **Maximum number of clones are achieved if CPU utilization approaches 90 %**
- **As add clones to system add resources accordingly**
  - Database connections
  - Heap memory space
  - Network Capacity

Database Servers

- **Verify Indices set on databases**
- **Verify following resources set accordingly**
  - Buffer pools
  - Cursors
  - Sockets
  - Caching used
  - Multiple disks
  - Stripe Logs

Network Components

- **Routers**
- **Firewalls**
- **Proxy Servers**
- **Network Interface Cards (NICs)**
- **Load Balancers**

Web Response Monitoring

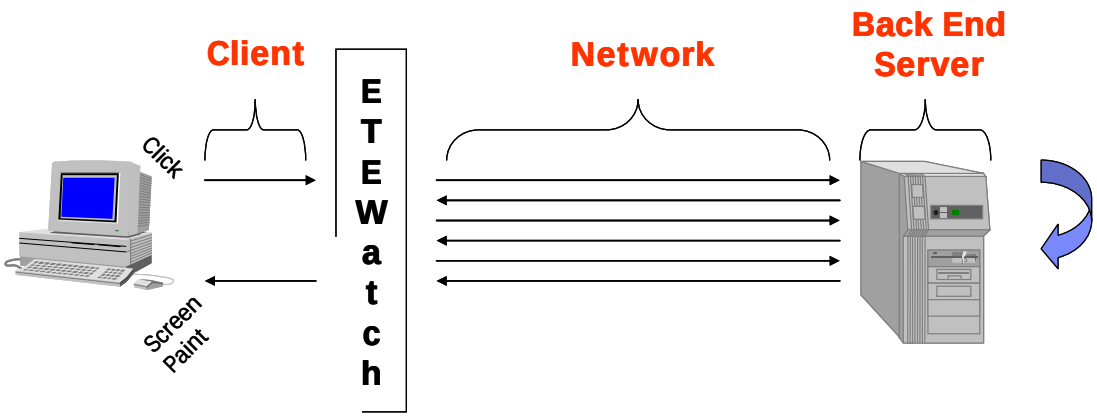
Passive Server-Side Monitoring at the page level

Detailed Web Server metrics

- For each Web Page and Object
  - Round trip response time
  - Load, resolve and Web application time
  - Network vs. Application time
  - Inbound vs. outbound traffic rates
  - Cache request time
- Inbound vs. Outbound Traffic byte counts and Object size
- Location Web Page or Object was pulled from (Web Server or Cache)
- Video Response and Play time
- Audio Response and Play time
- Browser Type and Version

Client based Response Time Monitoring

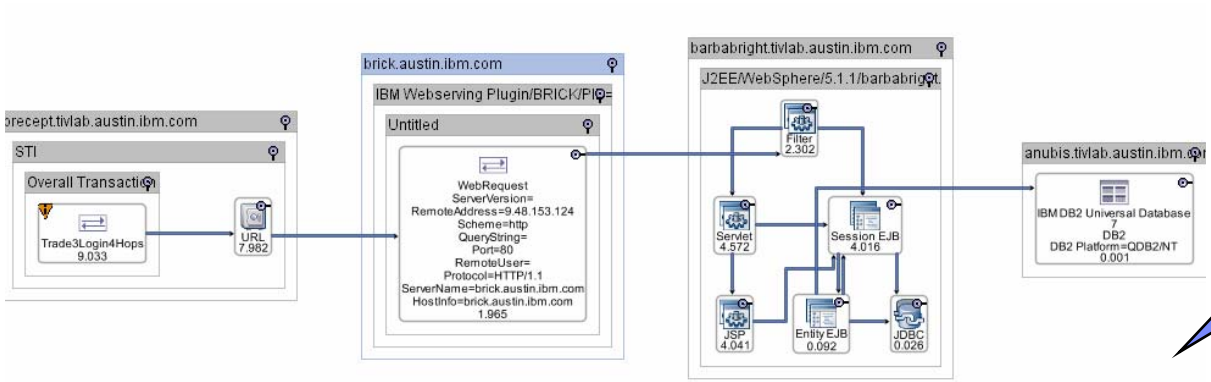
Real end user response times



Metrics collected

- Total Response Time
- Segmented Response Time
- Browse Time
- Workstation Hostname
- Application Name
- Transaction Name
- Application Destination IP Address
- IP Packet Size

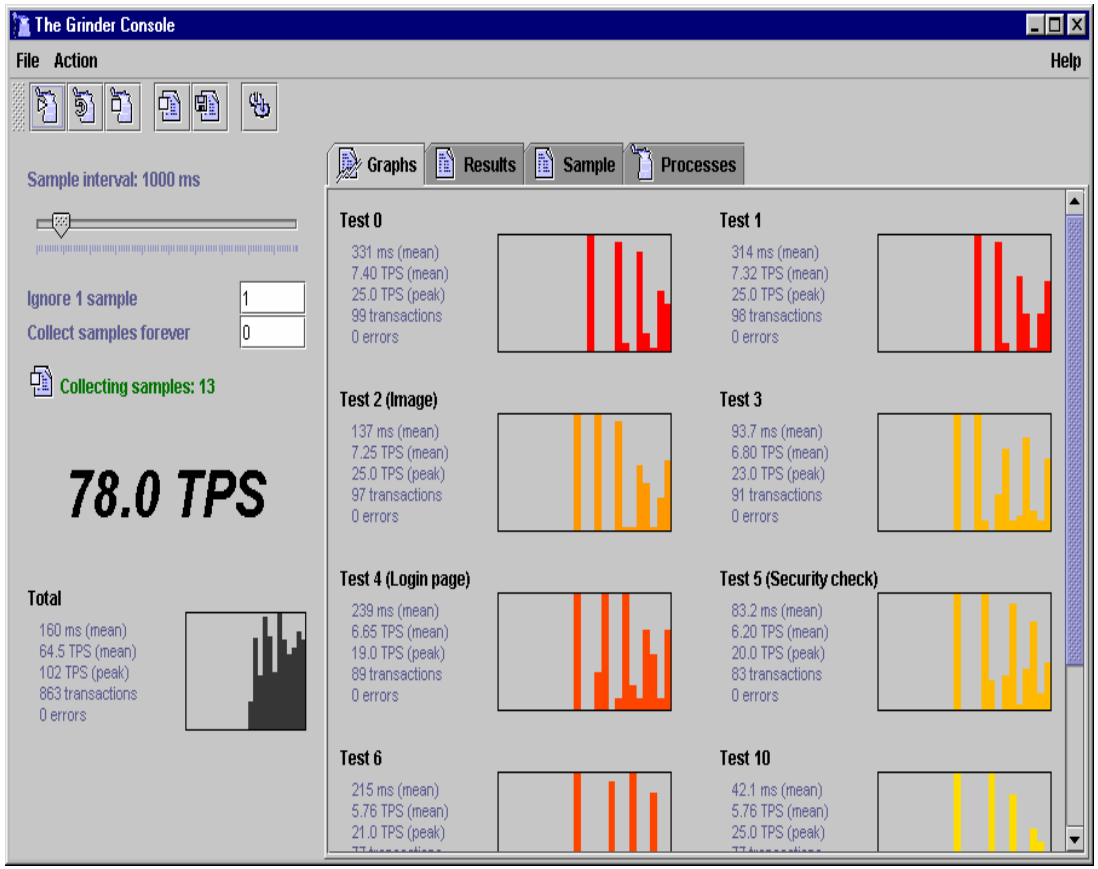
- **Client Time** — Processing time spent on the client
- **Network Time** — Client to Back End Server and back
- **Server Time** — Total Back End Application Time



*Problems are automatically identified by finding resources that are performing **differently** than normal*

Load Generation Tools

- **Software**
  - IBM Rational Performance Test
  - Mercury – LoadRunner
  - Segue – Silkperformer
  - Others
- **Free-ware**
  - Jmeter
  - Grinder
  - OpenSTA
  - Microsoft WSTT
  - Many others



Backup - Testing

Who should be involved?

- **Quality Assurance / Performance Test Team**
- **Lead Application Architect**
- **Application Developers**
- **Data Base Administrators**
- **Various System Admins and SMEs for J2EE, Messaging, ESB, OS, etc**
- **Network Administrator**
- **Operations**

▲*Yes, that is just about everyone, isn't it . . . .*

Where should it be done?

- **Separate, dedicated Test Environment**
- **Test Environment mirrors Production configuration (infrastructure, scalability, availability and application portfolio)**
  - Has own separate, dedicated network
  - Has a load generation tool available
  - Has monitoring tools available that can view all application infrastructure components which could impact performance
    - (HTTP, J2EE, DB, OS, Network, Messaging and ESB, legacy etc.)

What do I need to know to get started?

- **Ensure Functional/Integration Testing finished**
- **Know when testing/tuning is finished**
  - Set Performance Goals
- **Understand your user population & behaviors**
  - Establish Peak Load Estimates
  - Establish Throughput Estimates
  - Establish Response Time Estimates
  - Determine mix of expected Workloads

System Testing

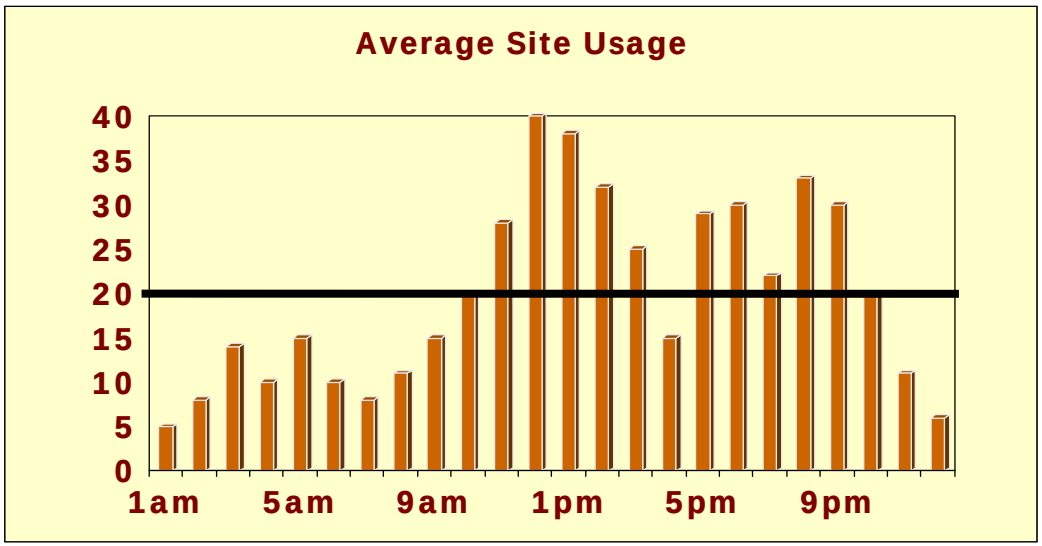
- **Testing functionality**
  - Easiest when requirements are detailed
  - Ideally all tests should pass before performance testing
  - Test both positive and negative paths

70% of new software systems fail in some way upon deployment!

Standish Group 1998

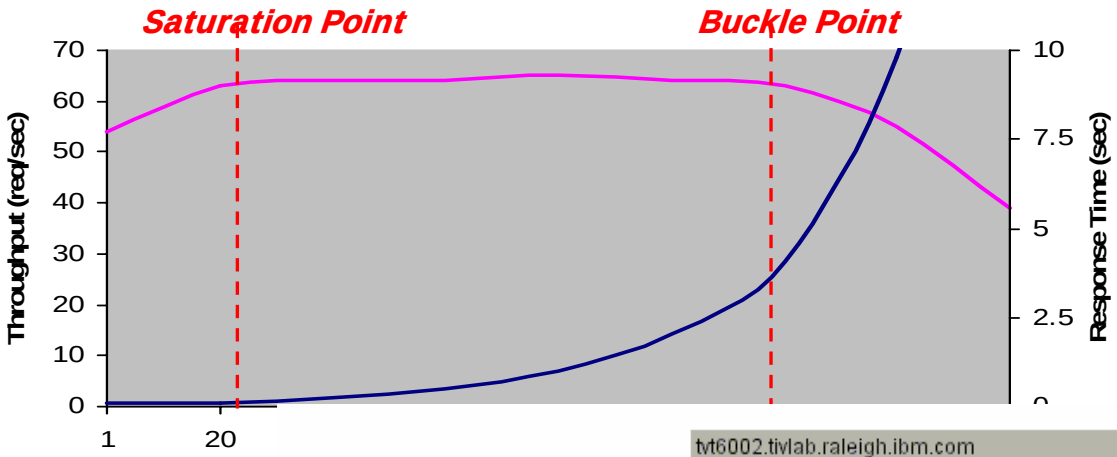
Load Testing

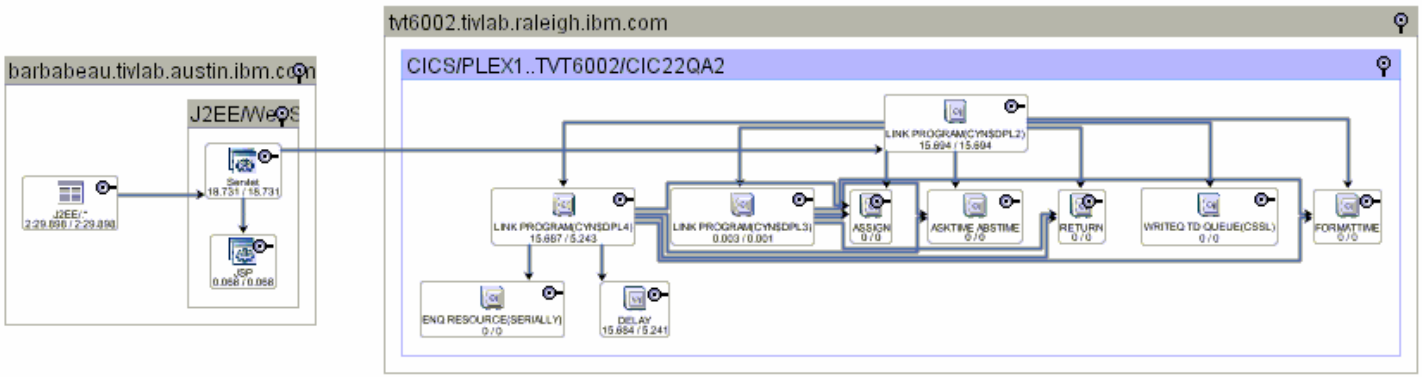
- **Ensure site can handle expected load**
- **Plan for daily, weekly and seasonal variations**
- **Ensure application performs as expected**
  - Should have expected throughput values
  - Understand peak and normal usage
  - Need to define different transaction types
  - Must collect the correct statistics
    - HTTP Hits/day could be misleading
- **May result in tuning**
- **May result in re-working/re-designing**



Stress Testing

- **Run to capacity and see where it breaks!**
- **Composite Applications have multiple buckle points**
- **Composite Applications break at the weakest links – understand how to stress them using transaction decomposition**





Stress Test Analysis

- **End-to-End analysis**
  - Investigate all aspects of the application
    - Network, Appliances, Web Server, Application Server, ESB, Database, Legacy