Unbreakable and Self-Adaptive Java™ EE Application Service

Henning Blohm

henning.blohm@sap.com
SAP AG

Dirk Marwinski

dirk.marwinski@sap.com
SAP AG

TS-8486

Goal

Learn about approaches for improved robustness and availability in the Java™ Platform, Enterprise Edition

Agenda

- Availability and Robustness
- Business Application Anatomy
- Robustness via Isolation
- Availability via Isolation
- Robustness via Problem Prevention

Agenda

- Availability and Robustness
- Business Application Anatomy
- Robustness via Isolation
- Availability via Isolation
- Robustness via Problem Prevention

Availability
(Courtesy of Wikipedia)

- Availability:

> The degree to which a system, subsystem, or equipment is operable and in a committable state at the start of a mission, when the mission is called for at an unknown, i.e., a random, time

- Translation:
  - Can you access it?

Robustness
(Courtesy of Wikipedia)

- Robustness:

> Resilience of the system, especially when under stress or when confronted with invalid input

- Translations:
  - Once you access a system, will you be able to complete your work?
  - If something goes wrong, how much of the system will be impacted?

Common Robustness Issues

Issues Which May Cause a System to Become Unavailable

- Hardware or Operating System malfunctions
- Java VM bugs
  - e.g. JIT compiler problems cause the VM to crash
- Application server bugs
  - Resource leaks (Out-of-memory, file handles,…)
- Application problems
  - Leaks, hanging requests, inefficient code, …
- Robustness is limited—no matter how hard you try!

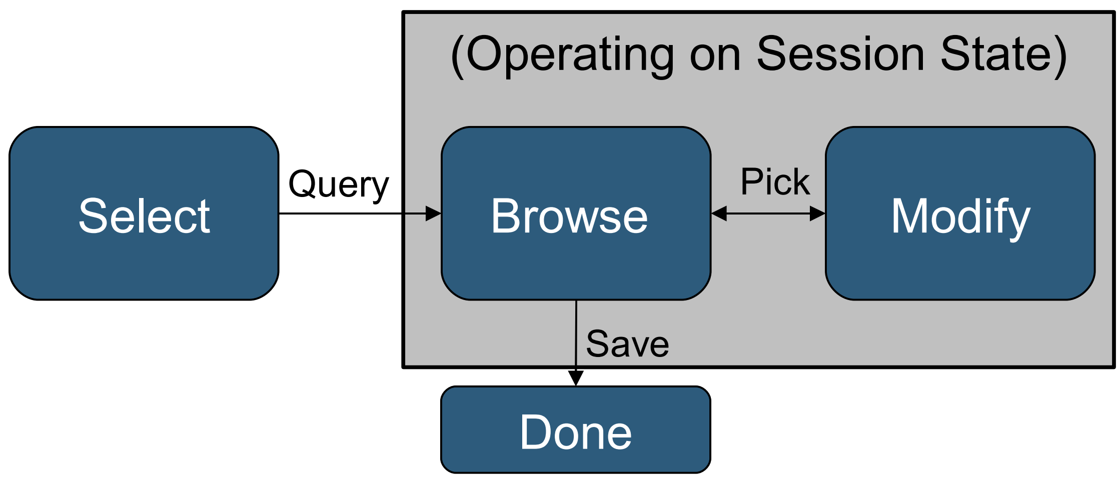
Agenda

- Availability and Robustness
- Business Application Anatomy
- Robustness via Isolation
- Availability via Isolation
- Robustness via Problem Prevention

Business Application Anatomy

Special Considerations for Business Applications

- Conversational state can become large:
  - Helps putting load off the database
- Typical pattern:
  - Select candidates
  - Modify some, mark others for deletion
  - Apply update to persistent store



Business Application Anatomy (Cont.)

- Business processes may require long running conversations
  - Complex user interactions
  - Orchestrated processes
- Conversational state must not be lost
  - Call center agent generally has no second chance to fill a form (failure implies a lost business!)

Source: Please add the source of your data here

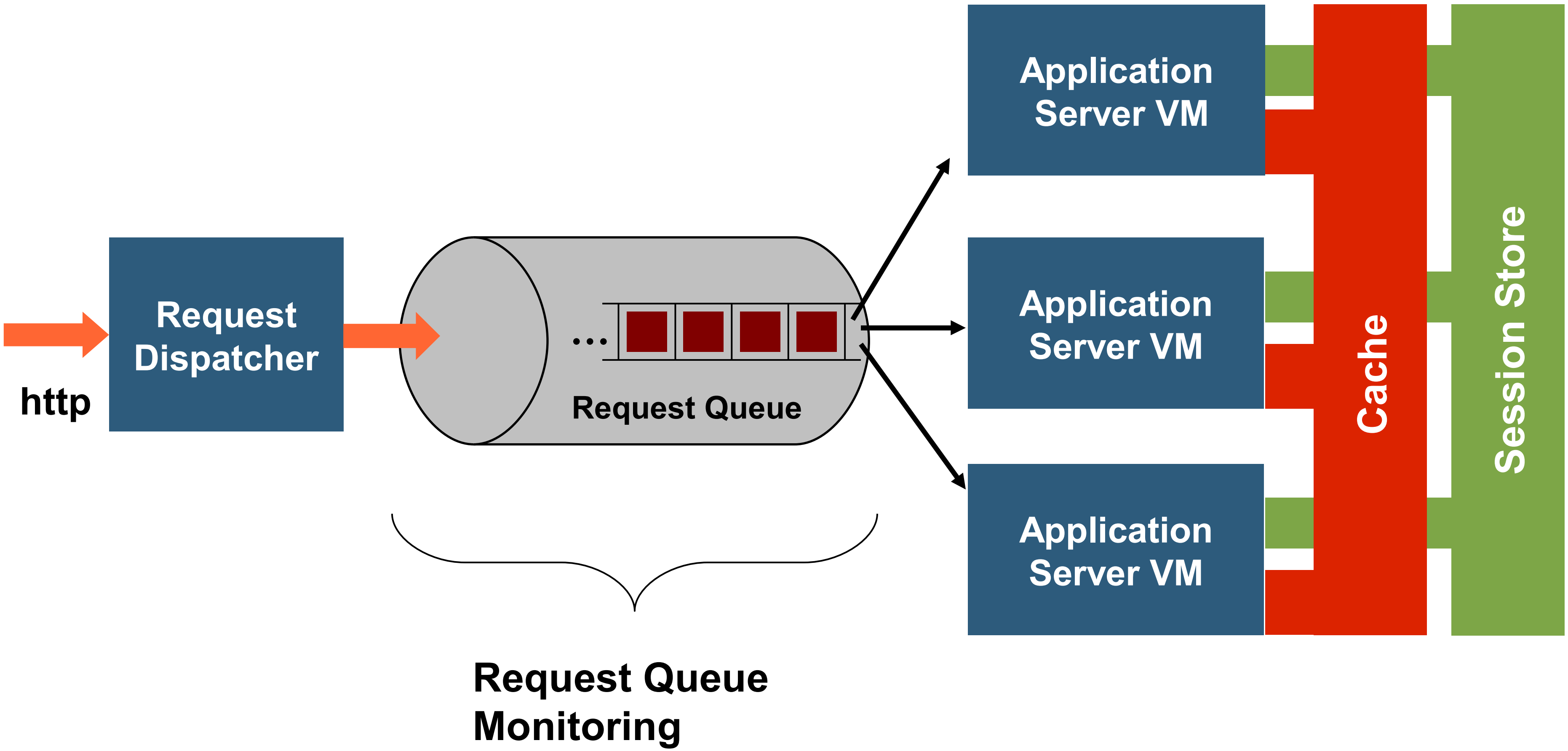
Business Application Anatomy (Cont.)

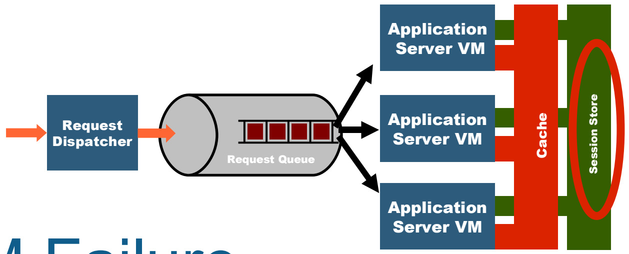
- Memory management is crucial
  - Session loss because one more user started working with the system is not acceptable (OutOfMemory)
  - Overload may be paid by performance penalty rather than non-availability
- Fail-over requirements extend to server resources
  - Stateful backend connections must be preserved

Agenda

- Availability and Robustness
- Business Application Anatomy
- Robustness via Isolation
- Availability via Isolation
- Robustness via Problem Prevention

Architecture Blueprint





User Session Isolation

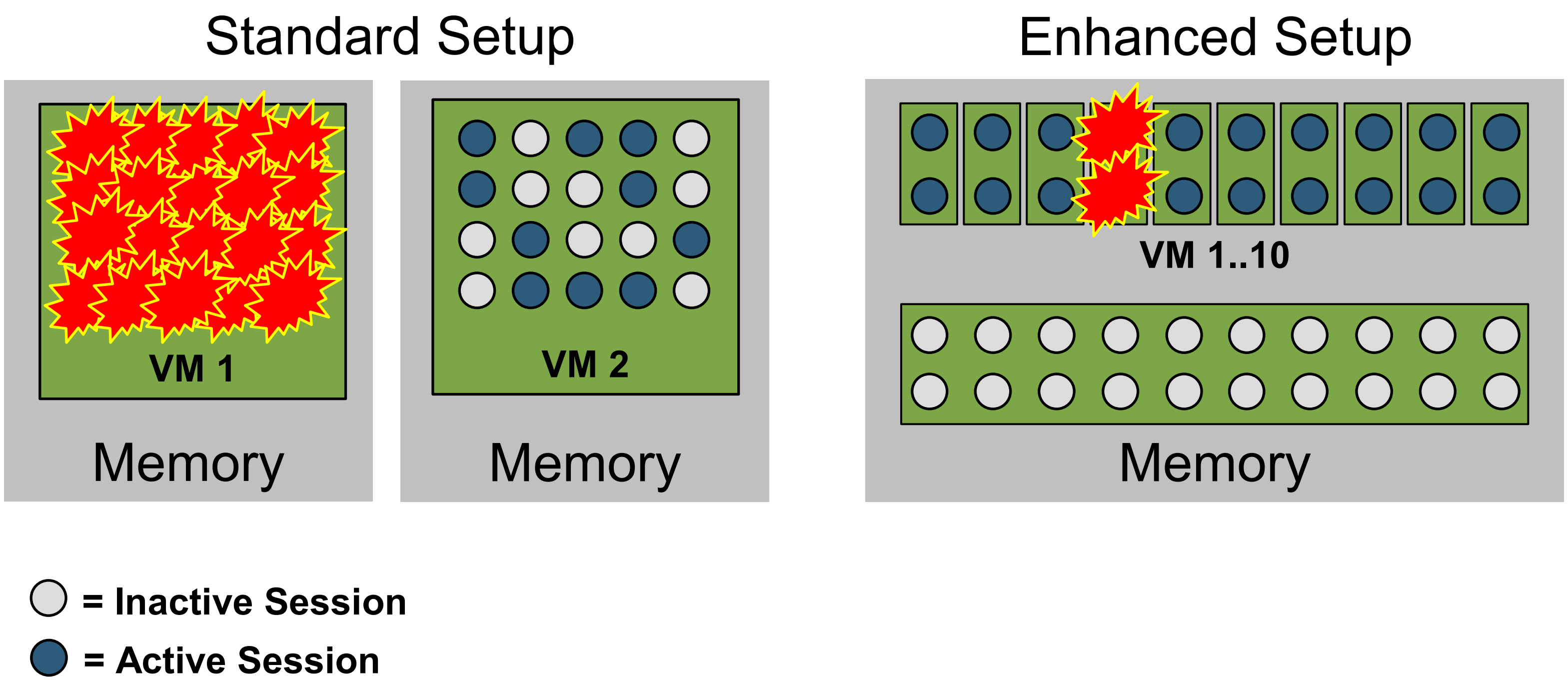
Session Data Should Survive a Java VM Failure

- Reduce the parallelism of active user sessions
  - An increased number of VMs per machine ensures that less active users are handled in each VM
- Separation of active from inactive sessions
  - Inactive (waiting) sessions must be secured in an area outside of the VM process

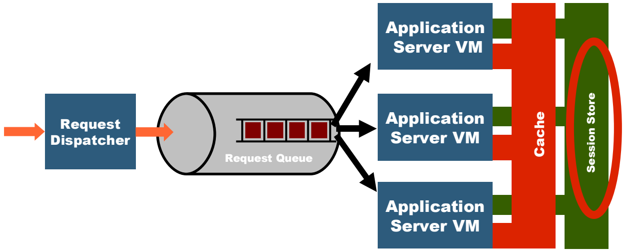
    (An active session means that a request is currently being processed on the server)

Reduced Parallelism

The Following Picture Illustrates Both Concepts

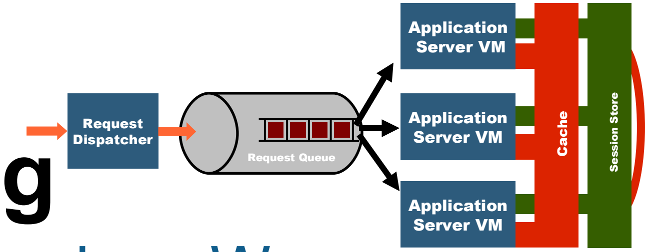


User Session Isolation



- User session isolation can be achieved by serializing and de-serializing it to the file system or to a database
- The approach is not optimal, because
  - Serialization is relatively slow and expensive
  - I/O is relatively slow
- This is not acceptable for many applications
- In order to extend the usage a fast way to move sessions and an efficient persistency mechanism is required
- This approach allows moving sessions from one VM to another

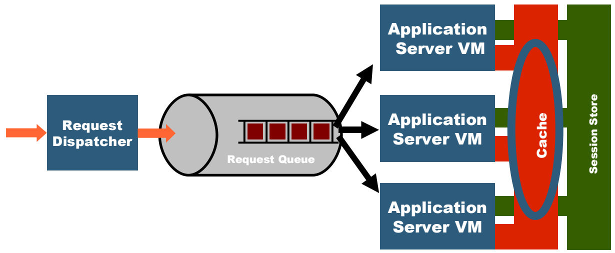
Fast Session Safeguarding



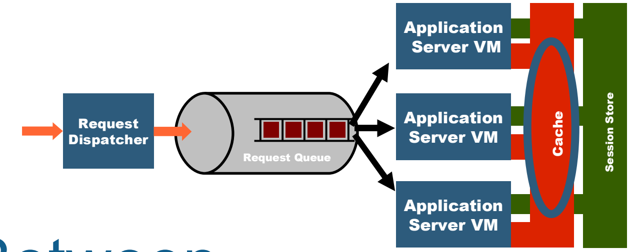
To Increase the Number of Secured Sessions We Have Optimized the Movement Process and the Persistency

- Use a shared memory area for persistency
  - Much faster than file system or database
  - Allows to shift sessions between VMs on one machine
- Do not use serialization
  - Use a memory copy approach which directly copies objects in and out of shared memory
- The drawback is that sessions cannot be moved between different machines

Application Data Sharing



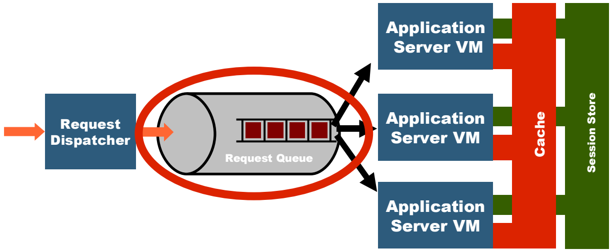
- Applications may require that several hundreds of megabytes of data are loaded into memory
  - Faster handling of large data sets (compared to data stored in SQL databases)
- Used to be stored in sessions—makes session serialization impossible (too big)
- Separating the data from the session and storing it in a file system or database is not really an option
  - Still too big
  - Data is already there



Application Data Sharing

Application Data Can Also Be Shared Between Different VMs Using a Shared Memory Approach

- Like sessions, application data can be stored in shared memory accessible to all VMs
- Survives a crash of a particular VM
  - Data does not have to be restored
- There is no need to copy the data when a session is moved to a different VM

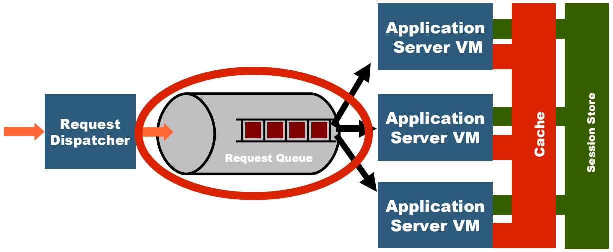


Request Queues

Separate Requests from a Particular VM as Long as it Is Waiting to Be Processed

- Queue of pending requests should be independent of a dedicated VM process
  - Dispatching to a dedicated Java VM should only happen if the request can be handled immediately
  - Pending requests will be stored in the request queue
- Pending requests should not be affected by a VM crash
  - User session to be moved to another VM
  - Only sessions with active requests are lost

Request Queues



- Implementation via shared memory
  - Requests are put into a queue by a dispatcher process
- They are taken out by a VM process when it has sufficient resources to process the request

Technical Prerequisites

The Concepts Introduced Above Require Extensions to the Underlying Virtual Machine, the Application Server, and the Application

- The blueprint architecture described above can be realized by using shared memory
- Several approaches possible in order to be able to access shared memory from Java technology
  - Access native operating system methods via Java Native Interface
  - Transparently add sharing features to the Java VM

Enhanced Java VM Capabilities

An Efficient Integration of Shared Memory Concepts into the Java VM

- Support of sharing approaches between several Java VMs on a single physical machine
  - Shared Java objects
- Fast mechanism to copy Java based objects into the shared memory
  - Standard serialization is too slow
  - Supports memory copy approach

Conclusions for Robust Applications

Robust Applications Do Not Come for Free—They Must Adhere to Certain Restrictions and Rules

- The user sessions should be
  - Kept small
  - Serializable but without custom serialization
- Do not use the session as a cache
  - Safeguarding it on every request can become too expensive
- Sessions which do not adhere to the rules are “sticky”
  - They still work, but lack the robustness advantages

Conclusions for Robust Applications

- Large data sets should be stored in shared memory
  - Allows the same data to be used from multiple Java VMs and therefore reduces the memory consumption
  - Data is protected against VM crashes

Agenda

- Availability and Robustness
- Business Application Anatomy
- Robustness via Isolation
- Availability via Isolation
- Robustness via Problem Prevention

Application Isolation

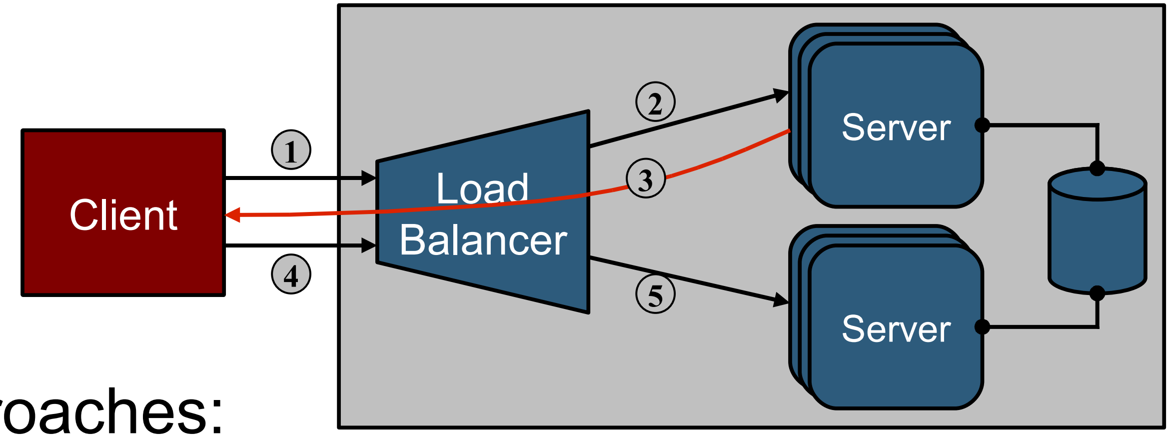
Protect Scenarios Against Each Other

- Failure protection
    - Restrict “bad” (unstable, leaking, experimental) code to separate server nodes
- Availability (SLA) for scenario
    - Reserve server nodes for scenario
    - Example:
        - Five nodes for management reporting, one for the portal
    - Example:
        - Front-end portal applications
        - Badminton booking app
- …but still one system

Application Isolation (Cont.)

(Approaches)

- Application isolation for the web may be implemented using enhanced load-balancing:



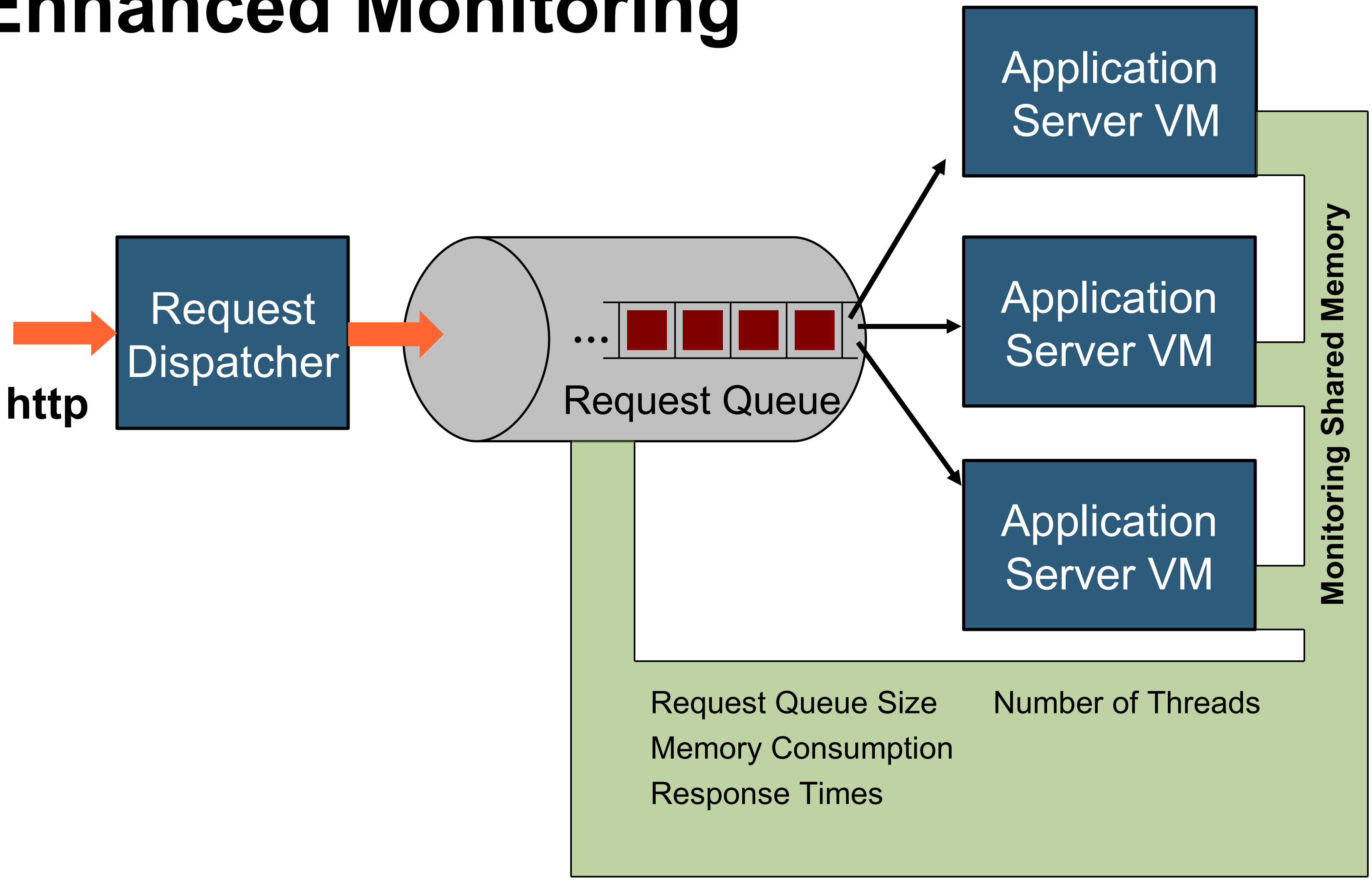
- More approaches:
  - WSRP based portal content
  - Web Services
  - Message queue separation

1. Incoming request
2. Default balancer decision
3. Redirect reply to URL carrying server group identification
4. Request to appropriate server group
5. Corresponding balancer decision

Agenda

- Availability and Robustness
- Business Application Anatomy
- Robustness via Isolation
- Availability via Isolation
- Robustness via Problem Prevention

Enhanced Monitoring



Enhanced Monitoring

- Monitoring and state information stored external to the application server VM in order to
  - Survive the crash of one Java VM
  - Allow access to that information from all Java VM processes handling user requests
- Data can still be accessed when a VM becomes unstable (e.g. high garbage collection times due to low memory)

System Health Indicators

Indicators Visualize an Unhealthy System and Enable Manual or Automatic Measures to Be Taken

- Through deep integration with the monitoring capabilities of the Java VM the health state can be estimated
  - Memory situation (garbage collection times)
  - Thread situation
  - Response times
  - Request queue size
- Those health states can be visualized and measured
  - Via visual indicators the administrators can treat the problem
  - Automatic heuristics can react properly in many cases

Robustness Via Self-Adaptive Problem Prevention

A Self-Healing Application Server Can Prevent Problems by Taking Automatic Measures

- Garbage collection times increase
  - Memory is getting low, reduce parallelism
  - Restart server node if this does not help
  - Prevent out-of-memory crash
- Response time increase
  - Possible overload situation
  - Reduce parallelism
  - Prevent timeout situations

Dealing with Unhealthy Java VMs

Unhealthy Java VMs Can be Dealt with Automatically

- Make sure that no additional requests are dispatched to it
- Move all user sessions to other Java VMs
- Restart the Java VMs
- Afterwards, add it again to the available Java VMs

For More Information

- SAP NetWeaver Application Server Java: https://www.sdn.sap.com/irj/sdn/developerareas/java
- Norbert Kuck: Increasing the Robustness of the Java™ Virtual Machine, JavaOneSM Conference 2005, TS-7179
- Thomas Smits: Unbreakable Java, JDJ Volume 9 Issue 12 (12/2004), http://jdj.sys-con.com/read/47362.htm
- Peter Kulka: High-end Java™ EE Application Servers for Enterprise Scale Business Suites, JavaOne Conference 2006, TS-4830

DEMO

Shared Memory Based Failover

Q&A

Unbreakable and Self-Adaptive Java™ EE Application Service

Henning Blohm

henning.blohm@sap.com
SAP AG

Dirk Marwinski

dirk.marwinski@sap.com
SAP AG

TS-8486