ORACLE®

Oracle WebLogic Server Monitoring and Performance Tuning

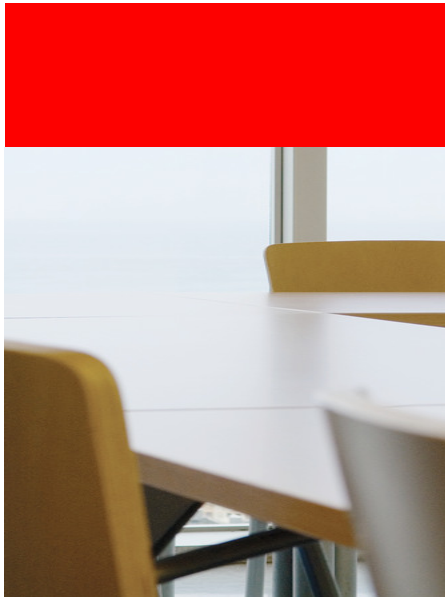
Duško Vukmanović
Principal Sales Consultant, FMW

Stuck Threads

- **A Label given to threads not returned to thread pool after a configured period of time (defaults to 600 secs)**
- Does not mean threads are literally stuck or locked (although possible) – they might be performing a long-running task
- This is an informational label – Administrators cannot stop the threads directly due to Java Threading design
- **The Server or** the **Work Managers can be configured to shutdown or go to an Admin State once a certain number of threads become stuck**

Taking Thread Dumps

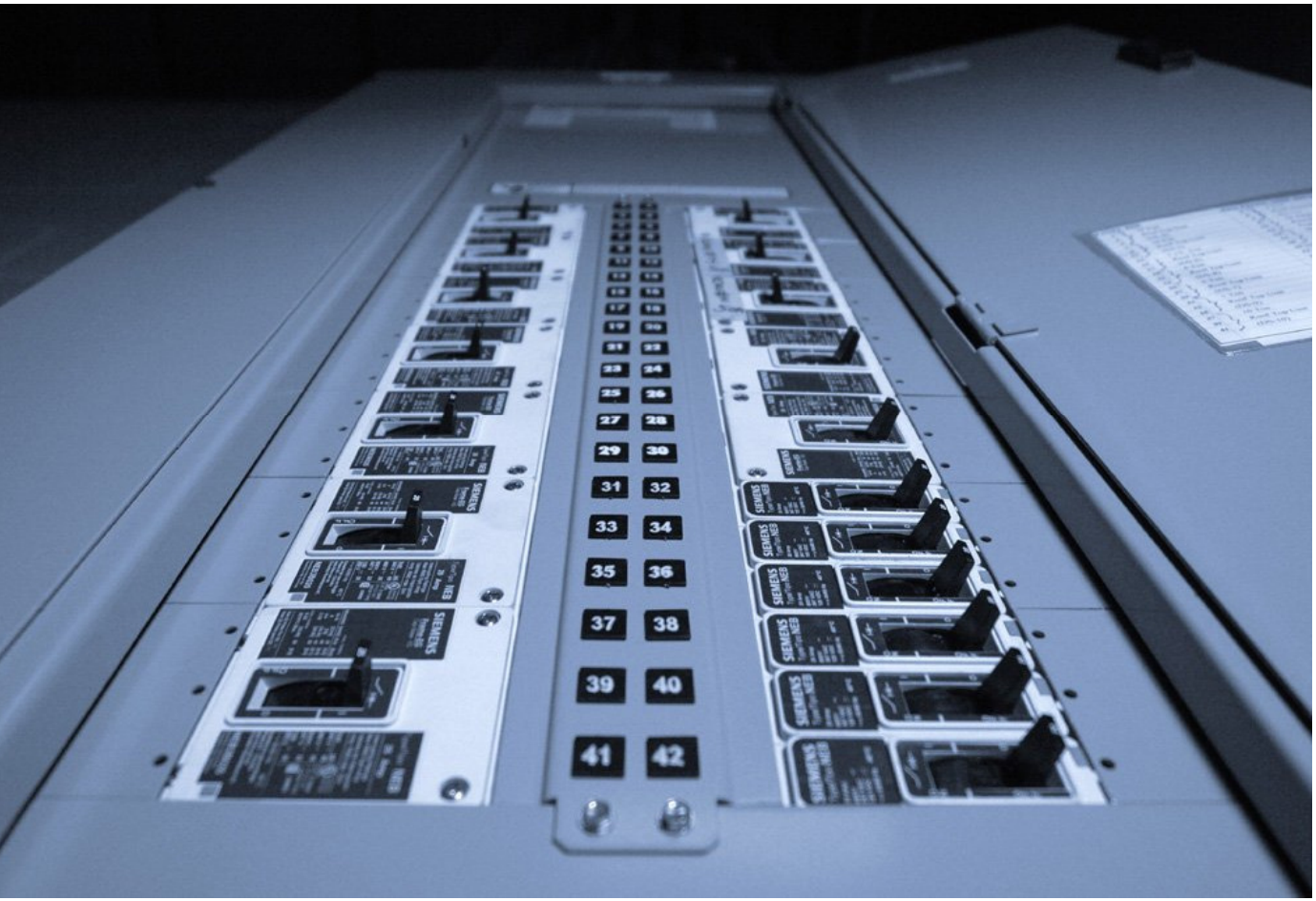
- Take thread dumps by using:
  - Windows: 'ctrl-break‘ or ‘ctrl-pause’
  - Unix: kill -3 <pid>
  - Admin Console → Server Instance → Monitoring → Dump Thread Stacks
- Thread Dumps provide:
  - Holistic view of the state of application server threads at that instant in time
  - Information about glaring issues such as
    - Hot spots within code which seem to be called often
    - Portions of code where the application seems to be hung
    - Locking and thread synchronization issues in an application



WebLogic Server Work Manager Overview

Real World Overload Protection

WebLogic Work Managers



WebLogic Server Overload Protection

Examples where this is applicable

- Protect against cascading failures
  - If maximum number of Database connections are in use, do not allocate new threads to service web requests that need a database connection

- Protect against hogging applications
  - If multiple applications are deployed to the same server, ensure badly behaving applications do not consume all resources

Work Management Concerns

Optimally tuning servers is hard!

- Thread pool size for optimum performance?
  - Workload can vary (time of day, event driven, etc.)
  - Handle overload conditions gracefully

- Prioritization of work
  - Across modules within single application
  - Across multiple applications
  - Across multiple classes of users
  - Ordered processing of requests (one request at a time)

Typical Solutions

Work Management Concerns

- Overprovision resources
  - Size thread pools for maximum load factor, usually requiring many load-test runs to optimize
- Deploy applications into different server instances
- Build additional machinery to detect overload condition and react

=> Sub-optimal ROI and increased Complexity

WebLogic Server Work Managers

Work Management on Autopilot

- Indicate your intent
- WebLogic is on autopilot adjusting to meet your goals

WebLogic Server Work Managers

Core Principles

- **Work Prioritization**
  - Applications define resource requirements via meta-data they can relate to, rather than low-level technical constructs (thread counts)
  - User-specific SLAs can be defined
- **Thread Pool Management**
  - Applications should not have to configure and maintain thread pools
    - WLS manages this internally and automatically
    - Without necessarily requiring Administrator configuration and sizing input
- **Overload Protection**
  - Standardized mechanism to respond to overload conditions

WLS Work Management

Key Components

- **New Thread Pool Implementation**
  - Single internally managed thread pool and priority-based request queue service all application requests
    - Request “Priority” dynamic and internally computed to meet application-defined goals
  - Thread Count **Self-Tuning**
    - Self-tuning thread pool monitors overall throughput every two seconds
    - Present thread count, measured throughput, and past history determines if thread count needs to change
    - New threads automatically added/removed as needed
    - Benefits Administrators and Operators - no need to conduct tedious performance testing or guesswork just to pick a static thread pool size that does not adapt to changing workloads

WLS Work Management

Key Components

- **Work Managers**
  - Runtime abstraction used by applications to define resource requirements
  - Work Manager Components
    - Request Class
      - **Fair-Share** – desired share of server resources for app
      - **Response Time** –desired app response time
      - **Context Based** – user-specific SLAs
    - Minimum Thread Constraint
    - Maximum Thread Constraint
    - Capacity
  - Specified in application descriptor (weblogic.xml, weblogic-ejb.xml, weblogic-application.xml)
  - Can be accessed programmatically via CommonJ API – JSR-237

Fair Share

Work Manager Examples

- Desired share of server resources
- Thread usage become higher as fair share number increases
- Fair shares are relative to other fair shares defined in the system

```
<work-manager>
        <name>highfairshare_workmanager</name>
        <fair-share-request-class>
                <name>high_fairshare</name>
                <fair-share>80</fair-share>
        </fair-share-request-class>
</work-manager>
```

Response Time Goal

Work Manager Examples

- Desired response-time goal in milliseconds
- Response-time goals relative to other response goals and fair shares
- Workload is distributed to applications based on the ratio of their Fair Share
  - Two applications each set at 80 would result in each getting ~50% of the CPU

```
<work-manager>
    <name>highfairshare_workmanager</name>
    <fair-share-request-class>
      <name>high_fairshare</name>
      <fair-share>80</fair-share>
    </fair-share-request-class>
</work-manager>
```

Context Based

Work Manager Examples

- Currently look at security name and group of user submitting the request

```
<work-manager> <name>context_workmanager</name>
    <context-request-class>
        <name>test_context</name>
        <context-case>
            <user-name>platinum_user</user-name>
            <request-class-name>high_fairshare</request-class-name>
        </context-case>
        <context-case>
            <user-name>evaluation_user</user-name>
            <request-class-name>low_fairshare</request-class-name>
        </context-case>
    </context-request-class>
</work-manager>
```

How to use Work Managers

- **Coarse Grained**
  - Target at the entire server (“default” Work Manager)
  - Target entire applications or modules - e.g. weblogic.xml

```
<wl-dispatch-policy>myAppWorkManager</wl-dispatch-policy>
```

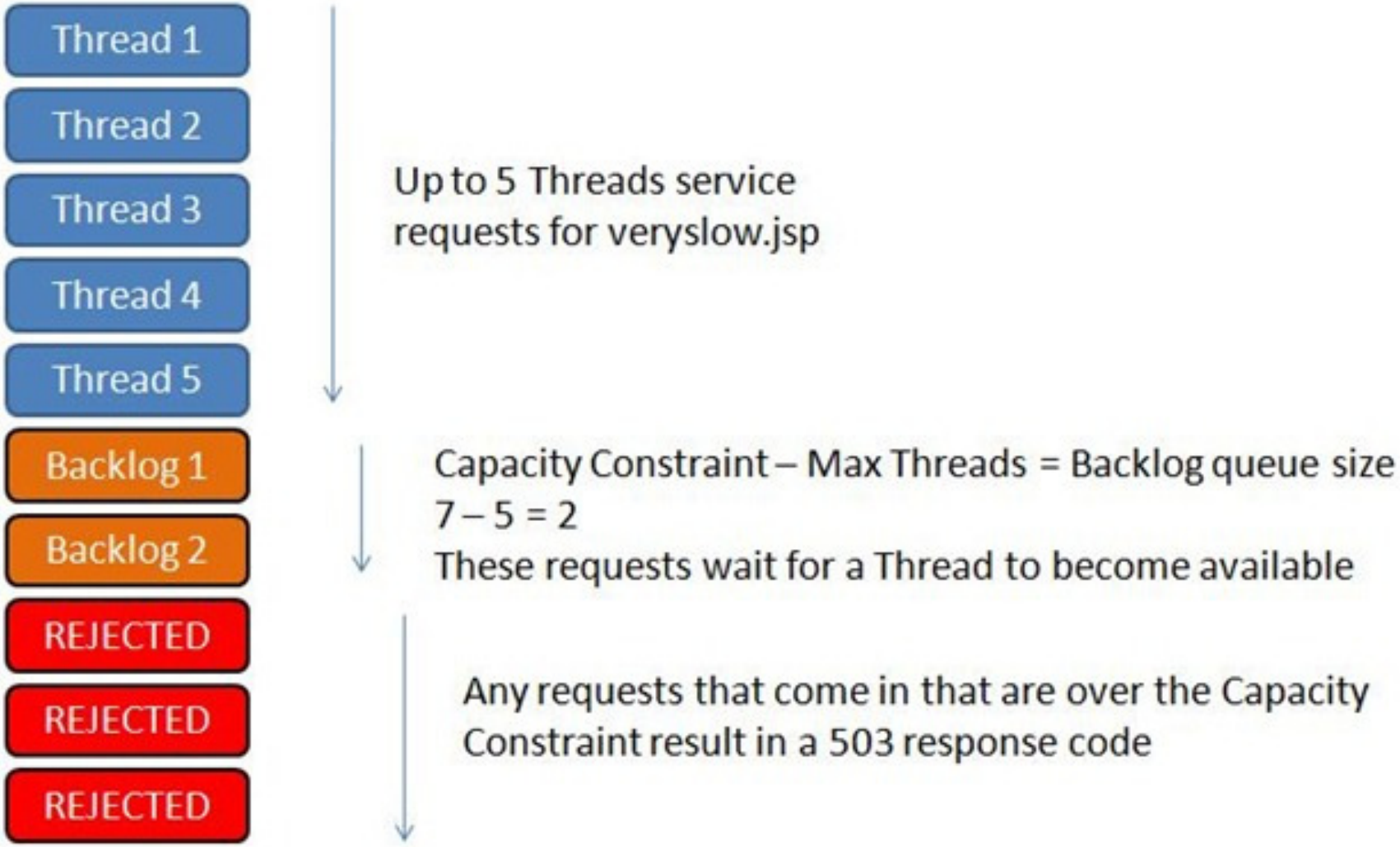
- **Fine Grained**
  - Target individual JSPs, Servlets, EJBs, MDBs

```
<servlet> ...
<init-param>
    <param-name>wl-dispatch-policy</param-name>
    <param-value>myCustomWorkManager</param-value> ...
```

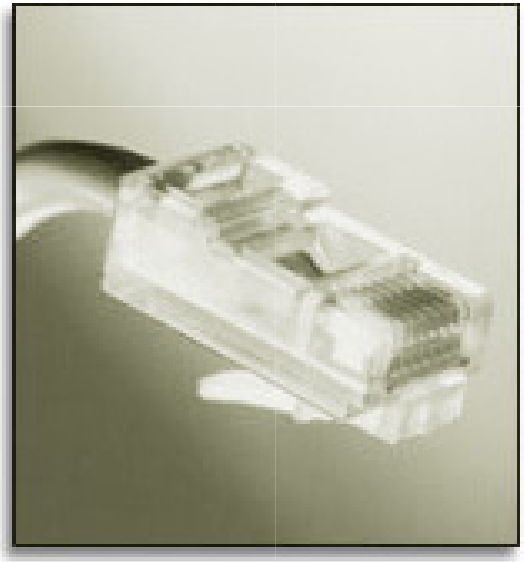
- **Programmatically via JNDI lookup**

```
InitialContext ic = new InitialContext();
commonj.work.WorkManager wm = (commonj.work.WorkManager)ic.lookup("java:comp/env/wm/myWM");
```

WLS behavior under test scenario

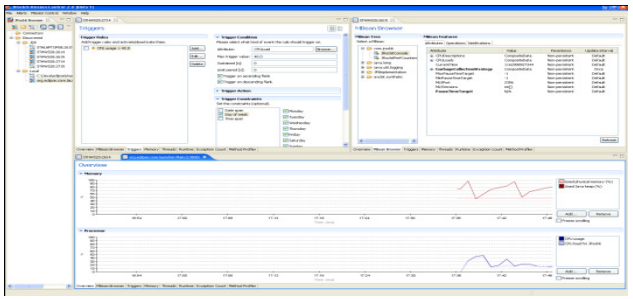


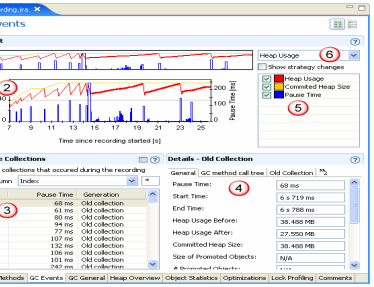
D E M O N S T R A T I O N



Oracle JRockit Mission Control

Monitoring





Profiling & performance tuning

Deploy

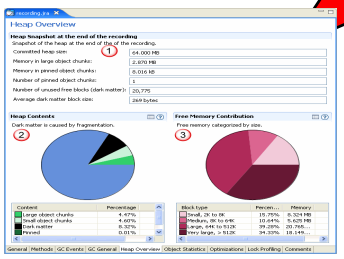
Development

Operations

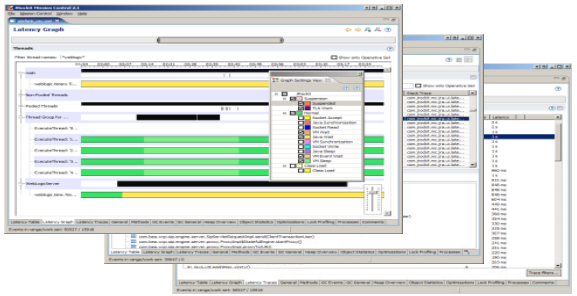
Alerts & triggers

Regression testing

Analysis



Escalate

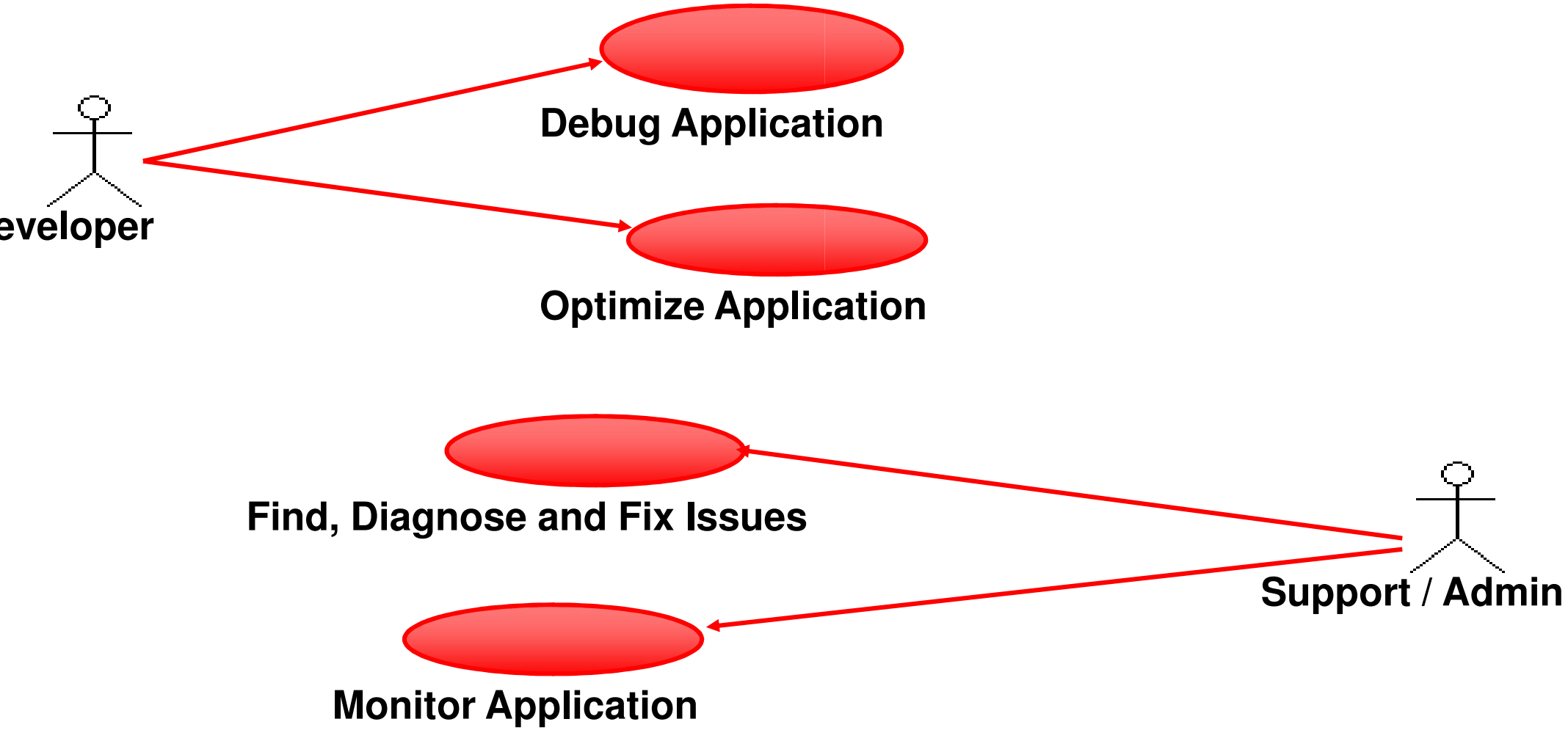


Production diagnostics

Troubleshooting

Use Cases

JRockit Mission Control



Runtime Monitoring & Profiling

JRockit Mission Control

What can you Monitor?

CPU Usage

Memory & Heap Usage

Garbage Collection Activity

Thread Usage and stack traces

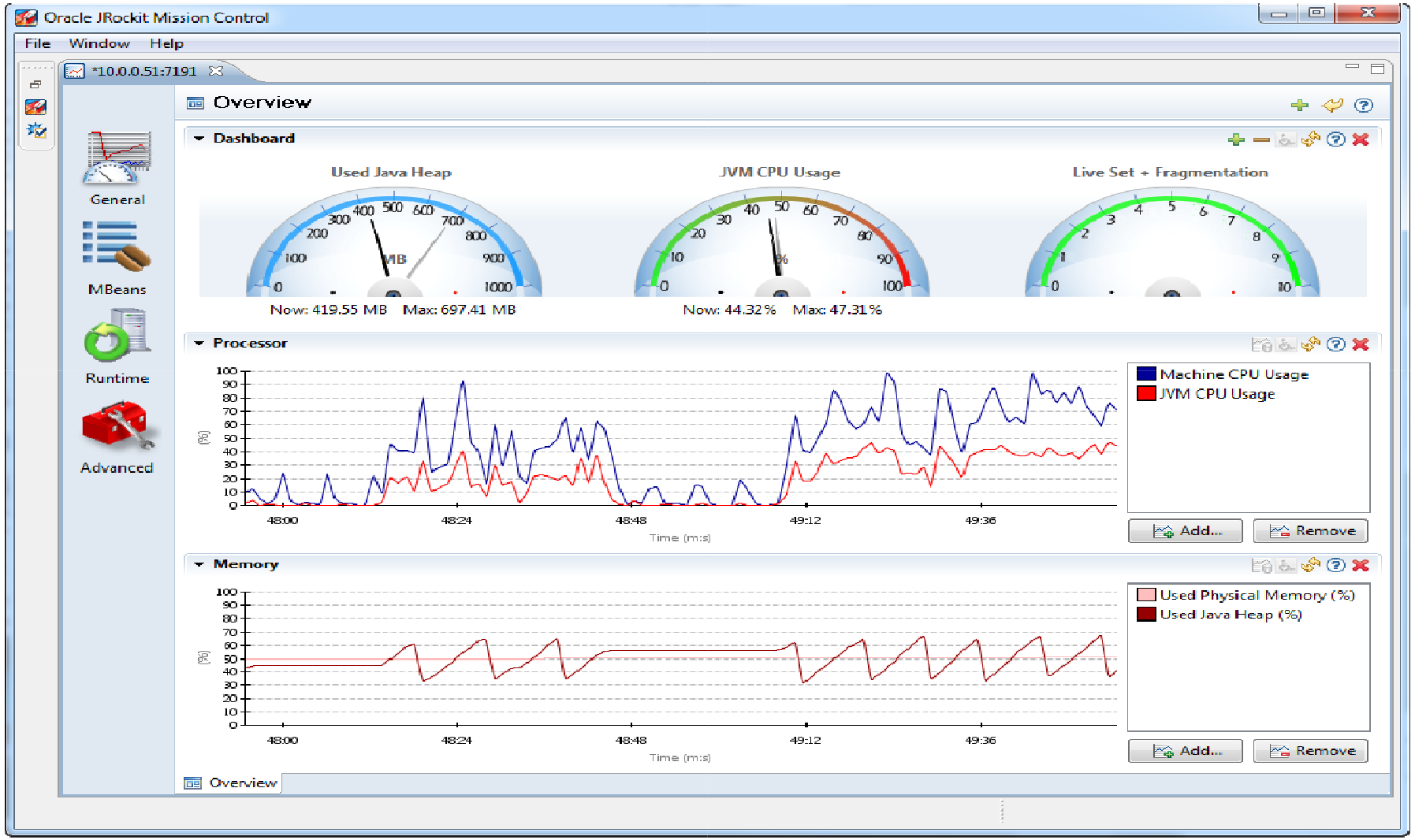
Mbeans with Mbean Browser

What can you Profile?

- User-selected Java Methods
- User-selected Exceptions

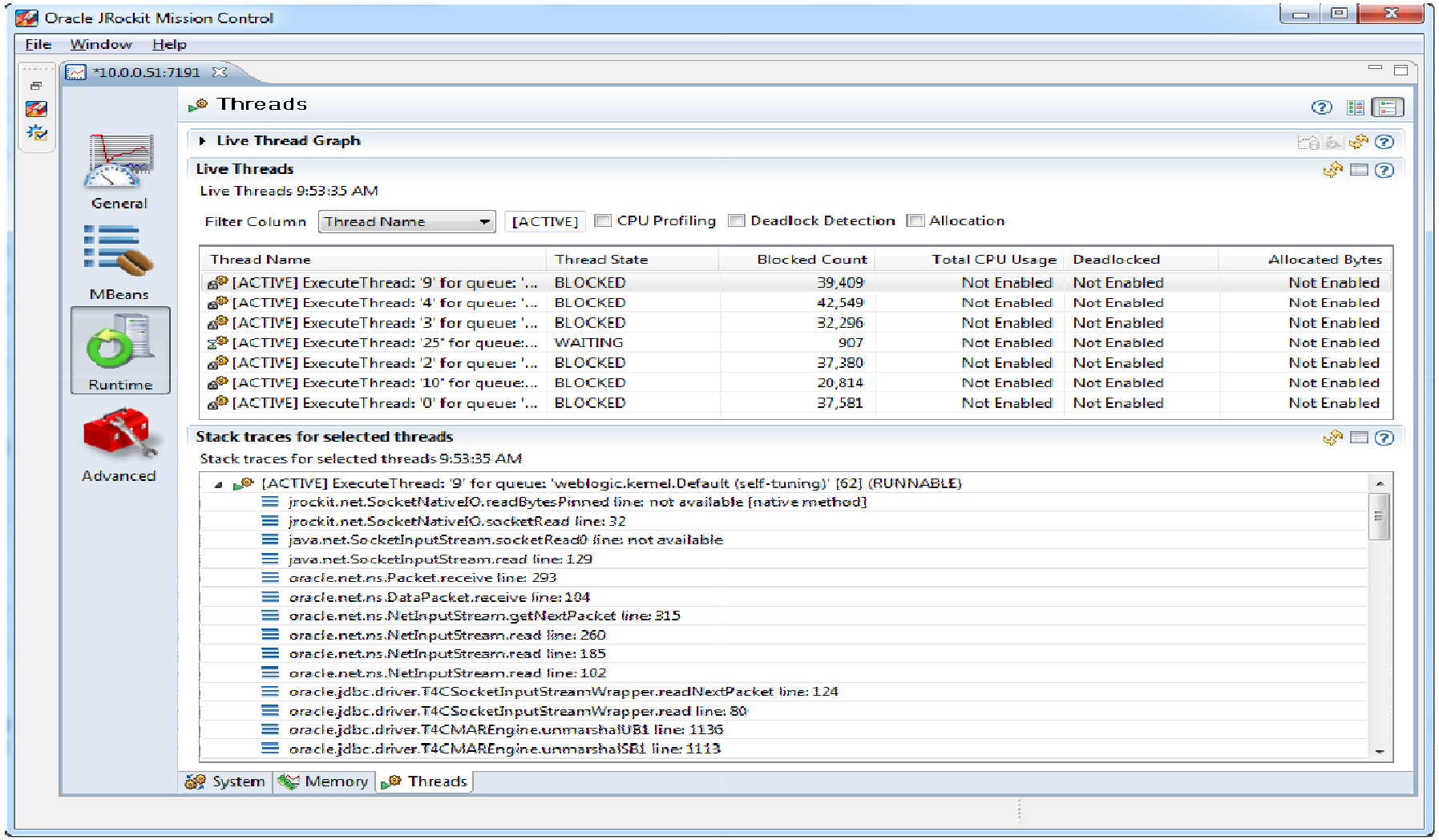
Oracle JRockit Mission Control

Monitoring Dashboard



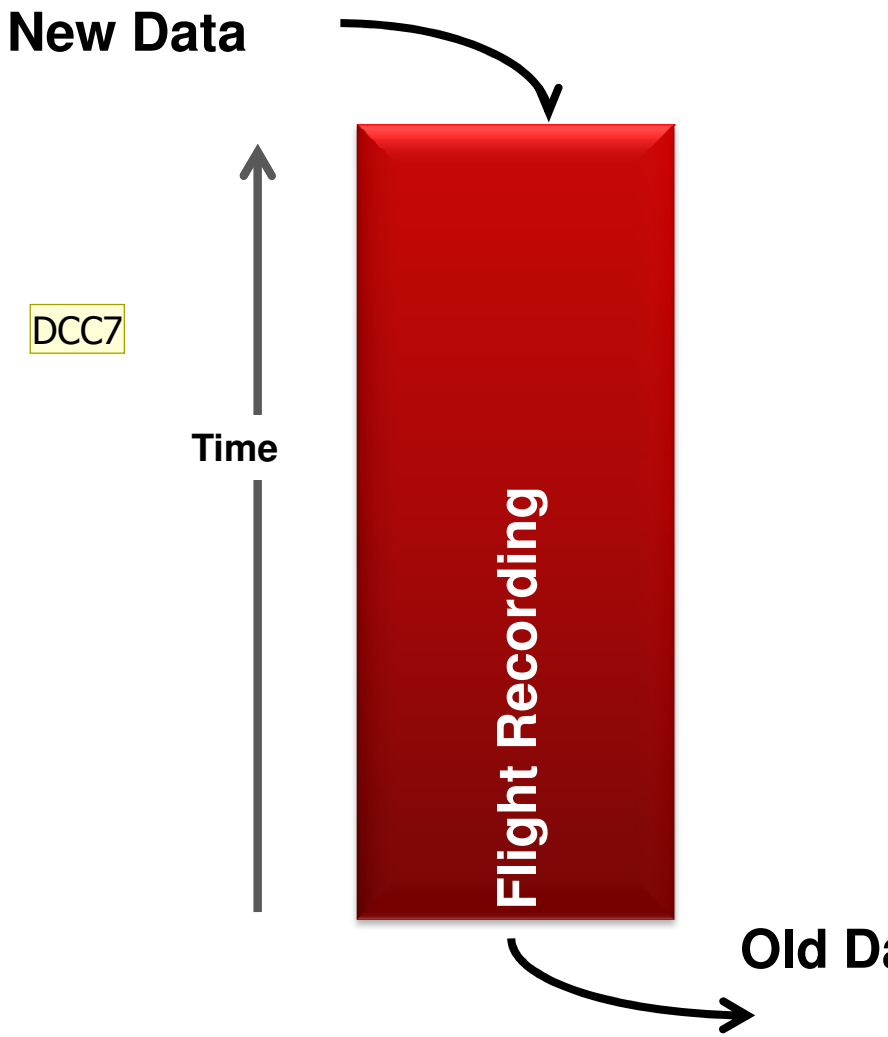
Oracle JRockit Mission Control

Monitoring Threads



What is the JRockit Flight Recorder?

- New in JRockit R28
- “Circular buffer” in JRockit JVM that stores diagnostic data
  - Always on
  - New data comes in and is stored, old data dropped off
- Built-in integration with JRMC
  - Replaces JRMC Runtime Analyzer and Latency Analyzer
- Very low/near zero overhead
  - Uses data already used by JVM
- Data can include events from the JVM and from any other event producer
  - WebLogic Server (WLDF)
  - Fusion Middleware (DMS)



Slide 24

DCC7 I fixed the color on this
David Cabelus; 5.5.2010.

Use Cases

JRockit Flight Recorder

- What it is designed for?
  - Provide diagnostic information in running production systems
  - Look back in time to see what happened after a crash
  - Capture most recent activity to enable analysis leading up to an issue
  - Capture data from all levels JVM, WLS, DMS, etc…
  - Offline/offsite analysis can be done using the JRMC GUI
  - JRockit dumps capture information to assist in crash-analysis

- What it is not designed for?
  - Large event payloads or very high volumes of events
  - Long history
  - Not a replacement for Debug logging or the server logging

Questions?