THE BUSY DEVELOPER'S GUIDE TO JVM TROUBLESHOOTING

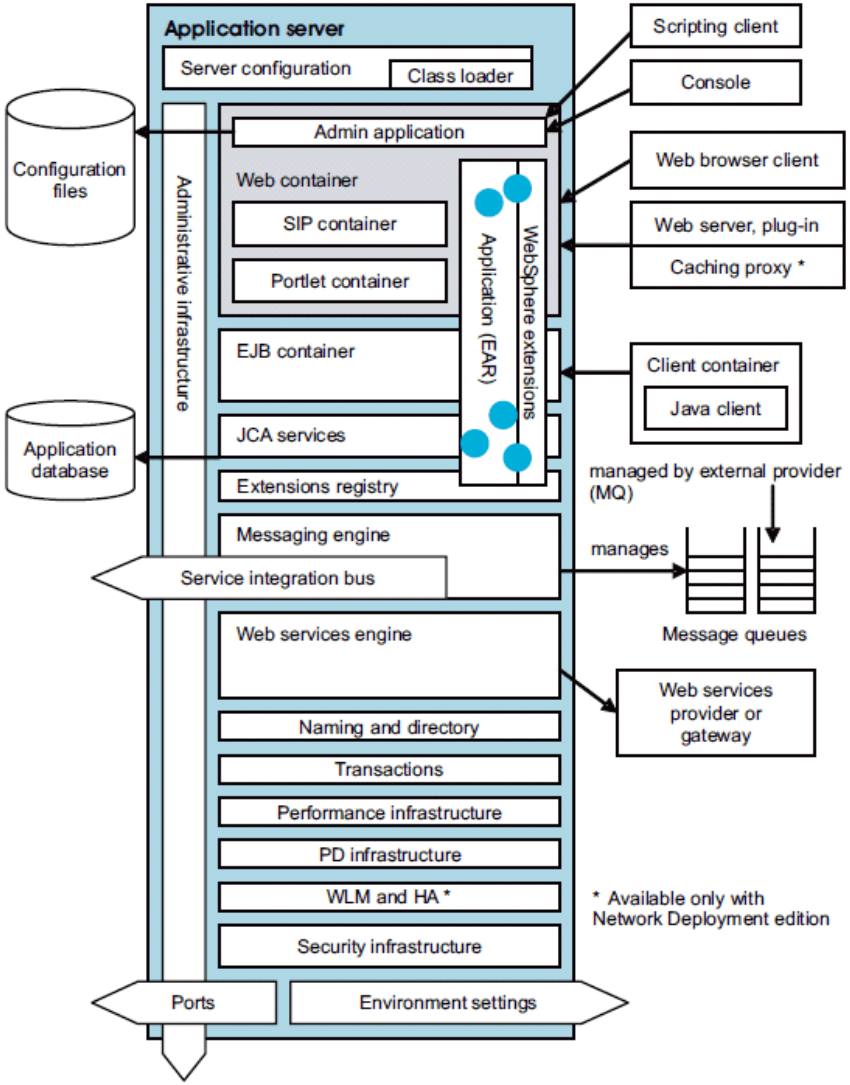
November 5, 2010

Rohit Kelapure
HTTP://WWW.LINKEDIN.COM/IN/ROHITKELAPURE
HTTP://TWITTER.COM/RKELA

Agenda

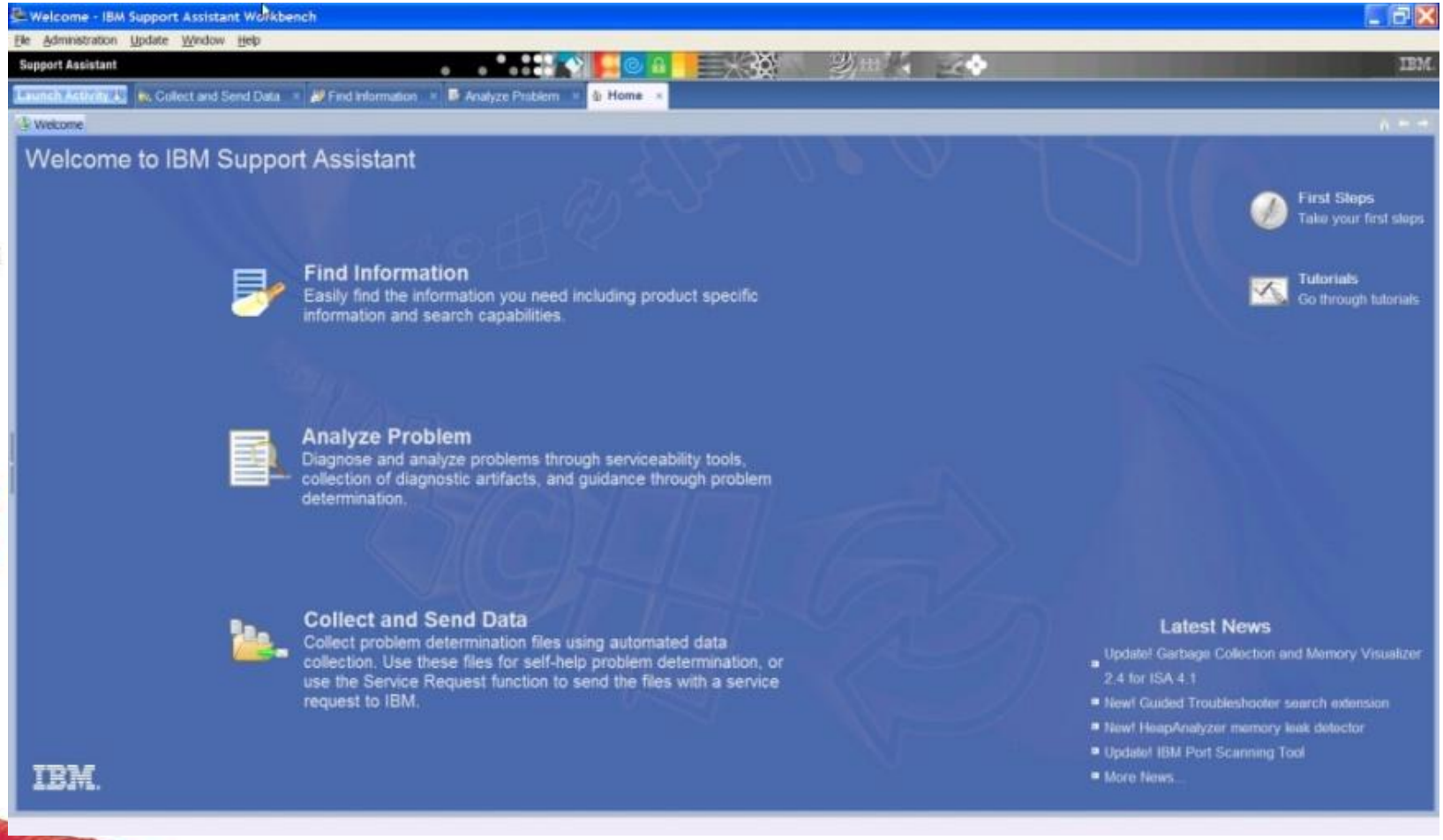
- Application Server component overview
- Support Assistant
- JVM Troubleshooting Tools
- Problem Determination Tools
- Scenario based problem resolution
- How customers get in trouble
- BadApp Demo
- Q&A

Component Overview

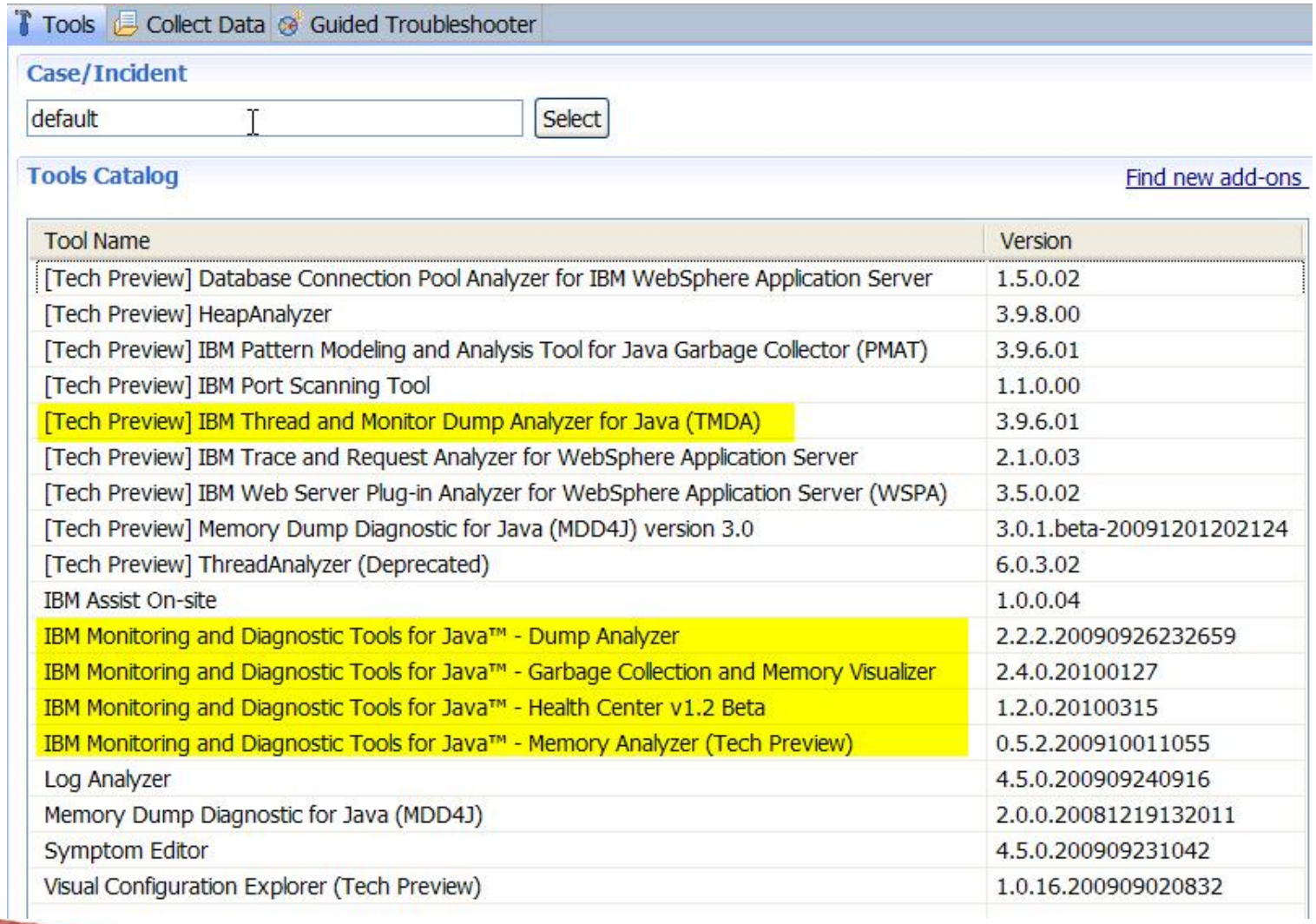


Support Assistant

Workbench to help with Problem Determination



Analyze Problem



Tool Name	Version
[Tech Preview] Database Connection Pool Analyzer for IBM WebSphere Application Server	1.5.0.02
[Tech Preview] HeapAnalyzer	3.9.8.00
[Tech Preview] IBM Pattern Modeling and Analysis Tool for Java Garbage Collector (PMAT)	3.9.6.01
[Tech Preview] IBM Port Scanning Tool	1.1.0.00
[Tech Preview] IBM Thread and Monitor Dump Analyzer for Java (TMDA)	3.9.6.01
[Tech Preview] IBM Trace and Request Analyzer for WebSphere Application Server	2.1.0.03
[Tech Preview] IBM Web Server Plug-in Analyzer for WebSphere Application Server (WSPA)	3.5.0.02
[Tech Preview] Memory Dump Diagnostic for Java (MDD4J) version 3.0	3.0.1.beta-20091201202124
[Tech Preview] ThreadAnalyzer (Deprecated)	6.0.3.02
IBM Assist On-site	1.0.0.04
IBM Monitoring and Diagnostic Tools for Java™ - Dump Analyzer	2.2.2.20090926232659
IBM Monitoring and Diagnostic Tools for Java™ - Garbage Collection and Memory Visualizer	2.4.0.20100127
IBM Monitoring and Diagnostic Tools for Java™ - Health Center v1.2 Beta	1.2.0.20100315
IBM Monitoring and Diagnostic Tools for Java™ - Memory Analyzer (Tech Preview)	0.5.2.200910011055
Log Analyzer	4.5.0.200909240916
Memory Dump Diagnostic for Java (MDD4J)	2.0.0.20081219132011
Symptom Editor	4.5.0.200909231042
Visual Configuration Explorer (Tech Preview)	1.0.16.200909020832

Tools

Problem	Artifact	Monitoring & Analysis
Memory leaks Out of Memory errors Application Unresponsive	Verbose Garbage collection log (native_stdout.log)	• PMAT,GCMV • VisualGC • jps, jstat, jstatd, jinfo
High CPU, Crash, Hang, Performance bottleneck, Unexpected termination	Javadump, Javacore (javacore*.txt)	• Thread Monitor & Dump Analyzer (TMDA), • Samurai TDA • Jstack
Lock Contention Low CPU at high load	Threads (Connection to running JVM)	• Sun VisualVM • JConsole • IBM Health Center • Jrockit Mission Control
Memory Leak Out of Memory errors	Heapdump (*.phd, *.txt, *.hprof)	• MAT • HeapAnalyzer • JHat
Native Memory Leak Anomalies Unexpected Crash	System or core dump (core.dmp, user.dmp), Files must be processed with jextract tool	• Monitor - GCMV, Examine - pmap & VMMap, Track - DebugDiag, libumem, valgrind, cmalloc & NJAMD

Runtime Serviceability aids

- Troubleshooting panels in the admin console
- Performance Monitoring Infrastructure metrics
- Diagnostic Provider Mbeans
  - Dump Configuration, State and run self-test
- Application Response Measurement/Request Metrics
  - Follow transaction end-to-end and find bottlenecks
- Trace logs & First Failure Data Capture
- Runtime Performance Advisors
  - Memory leak detection, session size, …
- Specialized tracing and Runtime checks
  - Tomcat Classloader Leak Detection
  - Session crossover, Connection leak, ByteBuffer leak detection
  - Runaway CPU thread protection

One more tool and I am going to scream

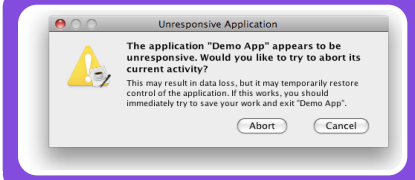
Most common JVM Problem Scenarios

Functional Problems

- Unexpected Exceptions, Compatibility

OOM Errors

- Java Heap ,Native Heap Classloaders

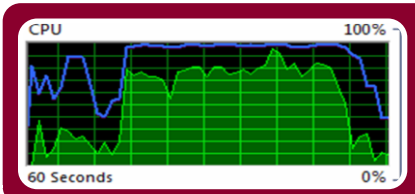


Hangs

- Synchronized resources, GC Pause times

Crash

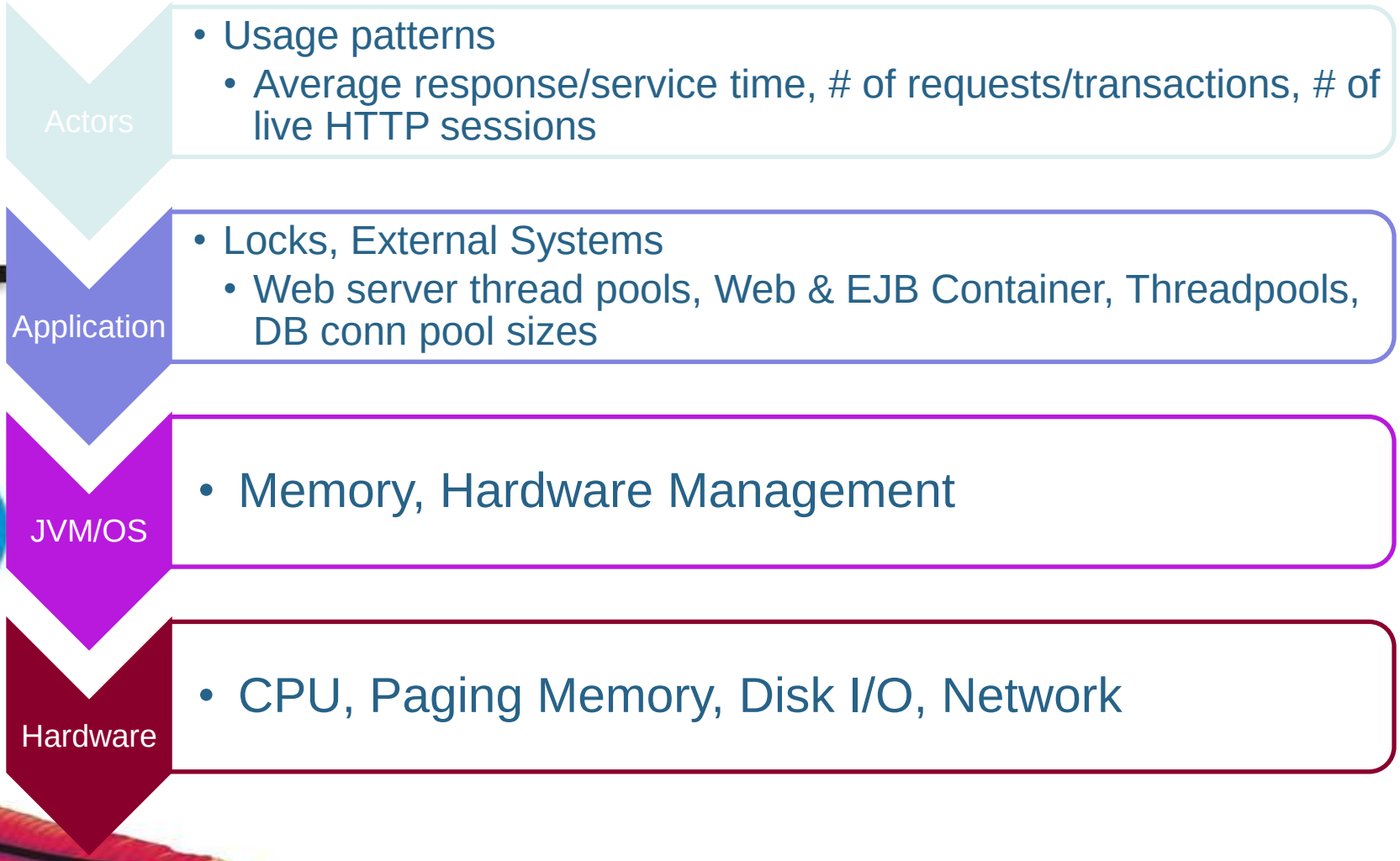
- JVM errors, JIT errors, JNI errors



High CPU

- Spin loops

Find Dominating consumer



WTF is wrong with my app

- Why does my app. run slow every time I do ?
- Why does my app. have erratic response times ?
- Why am I getting Out of Memory Errors ?
- What is my applications memory footprint ?
- Which parts of my app. are CPU intensive ?
- How did my JVM vanish without a trace ?
- Why is my application unresponsive ?
- What monitoring do I put in place for my app. ?

App runs slow when I do xxx ?

- Understand impact of activity on components
  - Look at the thread & method profiles
    - IBM Java Health Center
    - Visual VM
    - Jrockit Mission Control
- JVM method & dump trace - pinpoint performance problems.
  - Shows entry & exit times of any Java method
    - Method to trace to file for all methods in tests.mytest.package
  - Allows taking javadump, heapdump, etc when a method is hit
    - Dump javacore when method testInnerMethod in an inner class TestInnerClass of a class TestClass is called
  - Use Btrace, -Xtrace * –Xdump to trigger dumps on a range of events
    - gpf, user, abort, fullgc, slow, allocation, thrstop, throw …
    - Stack traces, tool launching

App. has erratic response times ?

- Verbose gc should be enabled by default
  - <2% impact on performance
- VisualGC, GCMV &PMAT : Visualize GC output
  - In use space after GC
    - Positive gradient indicates memory leak
      - Increased load (use for capacity plan)
      - Memory leak (take HDs for PD.)
- Chose the right GC policy
  - Optimized for “batch” type applications, consistent allocation profile
  - Tight responsiveness criteria, allocations of large objects
  - High rates of object “burn”, large # of transitional objects
  - 12, 16 core SMP systems with allocation contention (AIX only)
- GC overhead > 10% → wrong policy | more tuning
- Enable compressed references for 64 bit JVM ?

Out Of Memory Errors ?

- JVM Heap sized incorrectly
  - NOT recommended Xms == Xmx
  - GC adapts heap size to keep occupancy [40, 70]%
- Determine heap occupancy of the app. under load
  - Xmx = 43% larger than max. occupancy of app.
    - For 700MB occupancy , 1000MB Max. heap is reqd. (700 +43% of 700)
- Analyze heapdumps & system dumps with dump tools
  - Lack of Java heap or Native heap
- Finding which methods allocated large objects
  - Prints stacktrace for all objects above 1K
- Enable Java Heap and Native heap monitoring
  - JMX and metrics output by JVM
- Classloader exhaustion

Applications memory footprint ?

- HPROF – profiler shipped with JDK – uses JVMTI
  - Analysis of memory usage -Xrunhprof:heap=all
- Performance Inspector tools - JPROF Java Profiling Agent
  - Capture state of the Java Heap later processed by HDUMP
- Use MAT to investigate heapdumps & system dumps
  - Find large clumps, Inspect those objects, What retains them ?
    - Why is this object not being garbage collected
      - List Objects > incoming refs, Path to GC roots, Immediate dominators
    - Limit analysis to a single application in a JEE environment
      - Dominator tree grouped by Class Loader
    - Set of objects that can be reclaimed if we could delete X
      - Retained Size Graphs
    - Traditional memory hogs like HTTPSession, Cache
      - Use Object Query Language (OQL)

CPU intensive parts of the app?

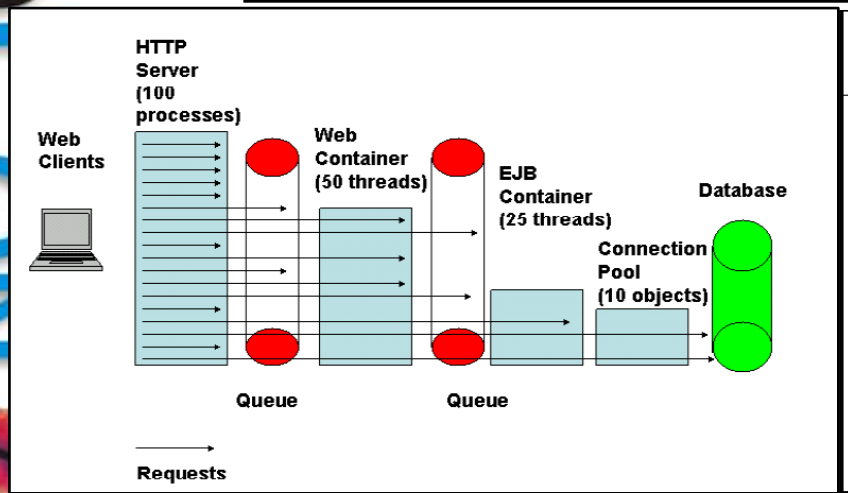
- HPROF - CPU spends most of its time
  - -Xrunhrof:cpu=samples, -Xrunhprof:cpu=time
- JPROF – method level execution times, who calls whom, etc.
  - Generate startup script & set the JVM argument
  - "-agentlib:jprof=rtarcf,callflow,logpath=./jprof" "-Xjit:disableInlining“
  - Output visualized using VPA
- ThreadDumps/Javacores - Poor mans profiler
  - Periodic javacores
  - Thread analysis – TMDA

How did my JVM vanish without trace ?

- JVM Process Crash Usual Suspects
  - Bad JNI calls, Segmentation violations, Call Stack Overflow
  - Native memory leaks - Object allocation fails with sufficient space in the JVM heap
  - Unexpected OS exceptions (out of disk space, file handles), JIT failures
- Monitor the OS process size
- Runtime check of JVM memory allocations –
  - Xcheck:memory
- Native memory usage - Create a core dump on an OOM
- JNI code static analysis -Xcheck:jni (errors, warnings, advice)
- GCMV provides scripts and graphing for native memory
  - Windows “perfmon“, Linux “ps” & AIX “svmon”
- Find the last stack of native code executing on the thread during the crash

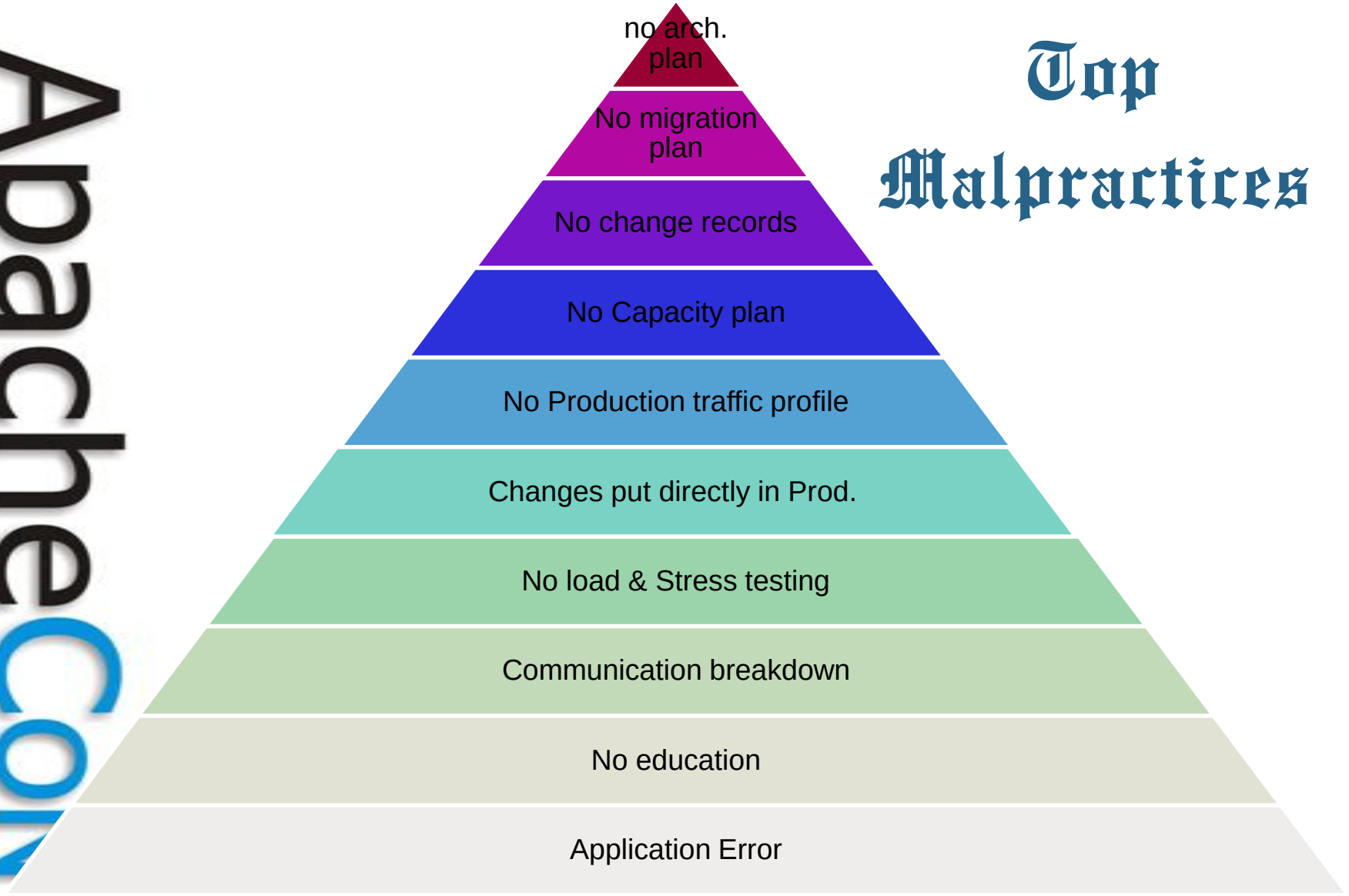
What do I monitor ?

JDBC Connection Pool	JVM Runtime	ServletSession Manager	SystemData	ThreadPool	Web Applications
AllocateCount ReturnCount CreateCount CloseCount FreePoolSize PoolSize JDBCTime UseTime WaitTime WaitingThreadCount PrepStmtCacheDiscardCount	HeapSize UsedMemory	ActiveCount CreateCount InvalidateCount LiveCount LifeTime TimeSinceLastActivated TimeoutInvalidationCount SessionObjectSize **	CPUUsageSinceServerStarted	ActiveCount ActiveTime CreateCount DestroyCount PoolSize DeclaredThreaHungCount	RequestCount ServiceTime ConcurrentRequests



CPU	Memory	Disk
Total % CPU User %CPU Wait %CPU System %CPU Avg. CPU RunQueue Context Switching CPU per processor CPU usage per process	Total Memory Real Memory Free Total Swap Space Swap Space Used Page In Page Out Pages/sec Memory usage per process Process% used FScache% used Scan Rate Page faults	Disk Service Time Per Disk Avg. Disk Queue Length Disk transfer per second %Busy Disk reads Disk writes

Top Malpractices



Test environment != Production

Demo, Q&A

- Eclipse Memory Analyzer
- Sun VisualVM
- Sun Visual GC
- Jconsole
- Samurai TDA
- Thread Monitor Dump Analyzer
- IBM Health Monitor
- Jrockit Mission Control

ApacheCon

Leading the Wave
of Open Source