‘06

JavaPolis

11 - 15 DECEMBER ■ ANTWERP ■ BELGIUM

Adobe®

JBoss® a division of Red Hat

ORACLE®

Sun® microsystems

The Network is the Computer™

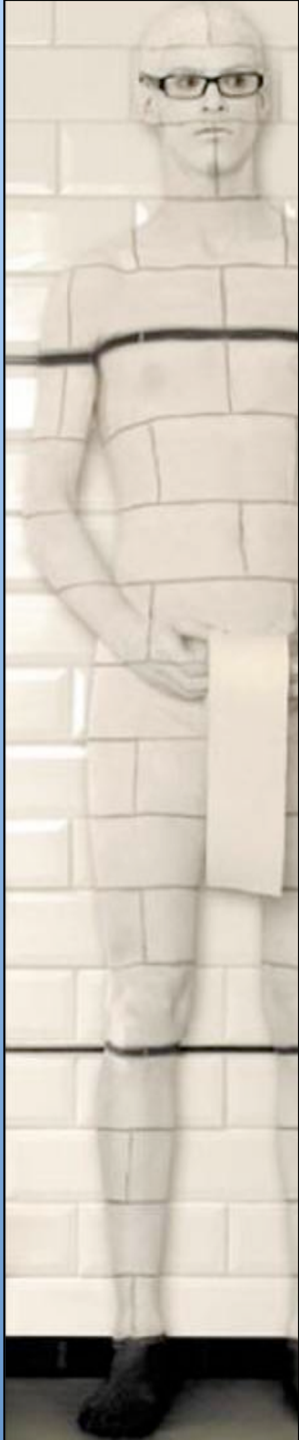
'06 JAVAPOLIS
11 - 15 DECEMBER ■ ANTWERP ■ BELGIUM

Why So Slow?

Debunking Speculative Tuning

Heinz Kabutz
Java Specialist
Maximum Solutions
http://www.cretesoft.com

Kirk Pepperdine
Performance Specialist
Kodewerk Ltd.
http://www.kodewerk.com

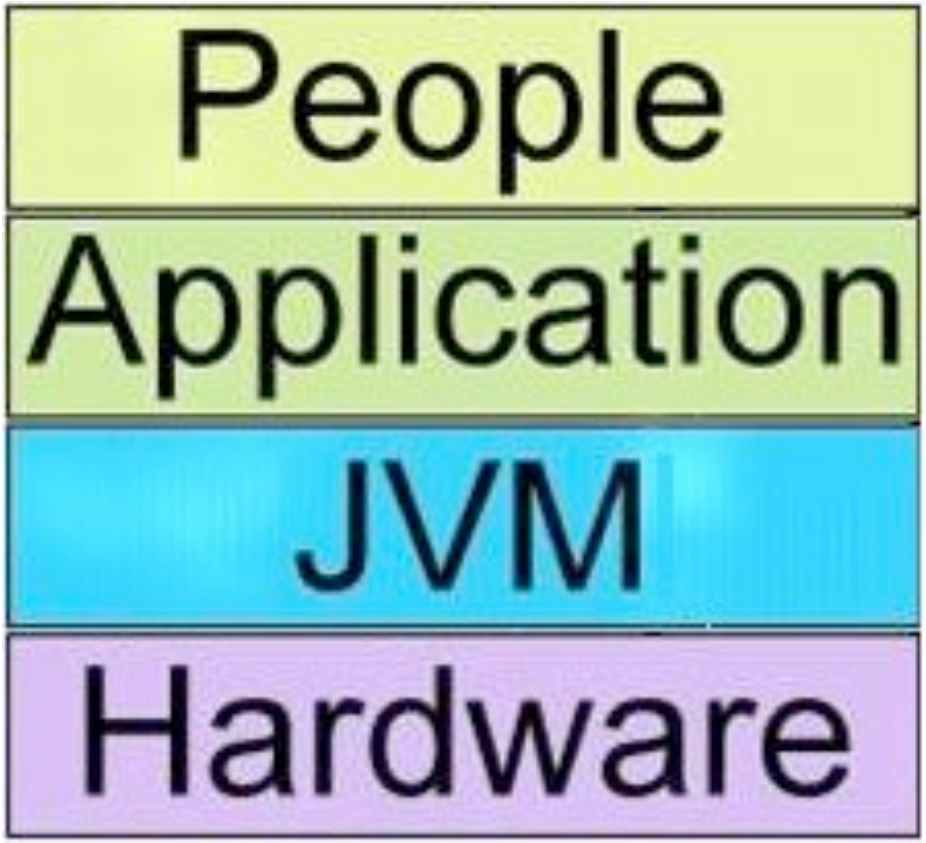


Our Typical Customer

- Customer JoGoSlo Ltd calls us in desperation
  - Millions of $$$$ invested
  - Users complain about poor performance
    - Customers are starting to abandon the project
- Developers in a panic
  - 6 man months already spent “tuning” with no results
  - Can almost reproduce the problem
  - Still have some ideas of what to do
  - But, management has lost confidence
- We have 5 days to diagnose problem and propose fix

Tuning Tool for Managers

Tuning Tool for Engineers – “The Box”



People

Application

JVM

Hardware

Heinz Kabutz

- Author of *The Java Specialists' Newsletter*
- Sun Java Champion
- http://www.cretesoft.com
- Lives in Greece
- Consults and trains companies about Java

maximum solutions
java training

JAVA™
CHAMPIONS

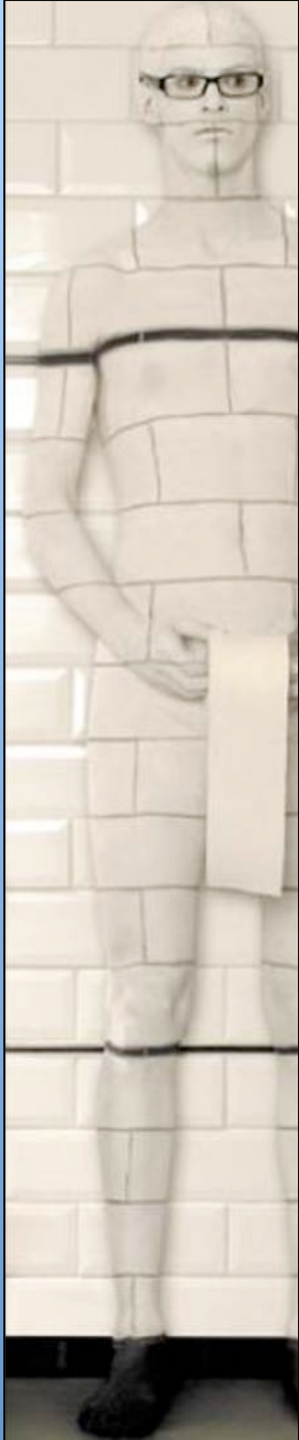
Kirk Pepperdine

- Engaged in performance tuning world wide.
- Co-author of www.javaperformancetuning.com
- Editor www.theserverside.com
- Sun Java Champion
- Speaks frequently about performance tuning
- http://www.kodewerk.com

Topics

- Dynamic nature of systems
- Measure don’t guess
- People
- Hardware/OS
- JVM
- Application
- External systems
- Putting it all together

Time to Setup



Time to Setup TipsDB

- Download from http://www.cretesoft.com/outgoing/javapolis.zip
- Set path to your JDK in the setenv.bat
- Go into tipsdb directory
- Call startDB.bat
- Call createDB.bat
- Call appserverStart.bat
- Connect to http://localhost:8080/tips/wildcard
- Connect to http://localhost:8080/tips/keyword

Dynamic Nature of Systems

Knowing what to measure and how to measure it makes a complex world much less so

Steven D. Levitt
Stephen J. Dubner
Authors of Freakonomics

Dynamic Nature of Systems

- Performance tuning is a complex task
  - Need to reverse engineer complex systems
  - Need right view of the system
    - Most useful view comes from measurements
- We will take introductory look at
  - What to measure
  - How to measure
  - How to understand the measurements

Importance of the Environment

- Need to understand all elements in the environment
- Changing elements of a system can change the dynamics of that system
  - E.g. different users, CPUs, network

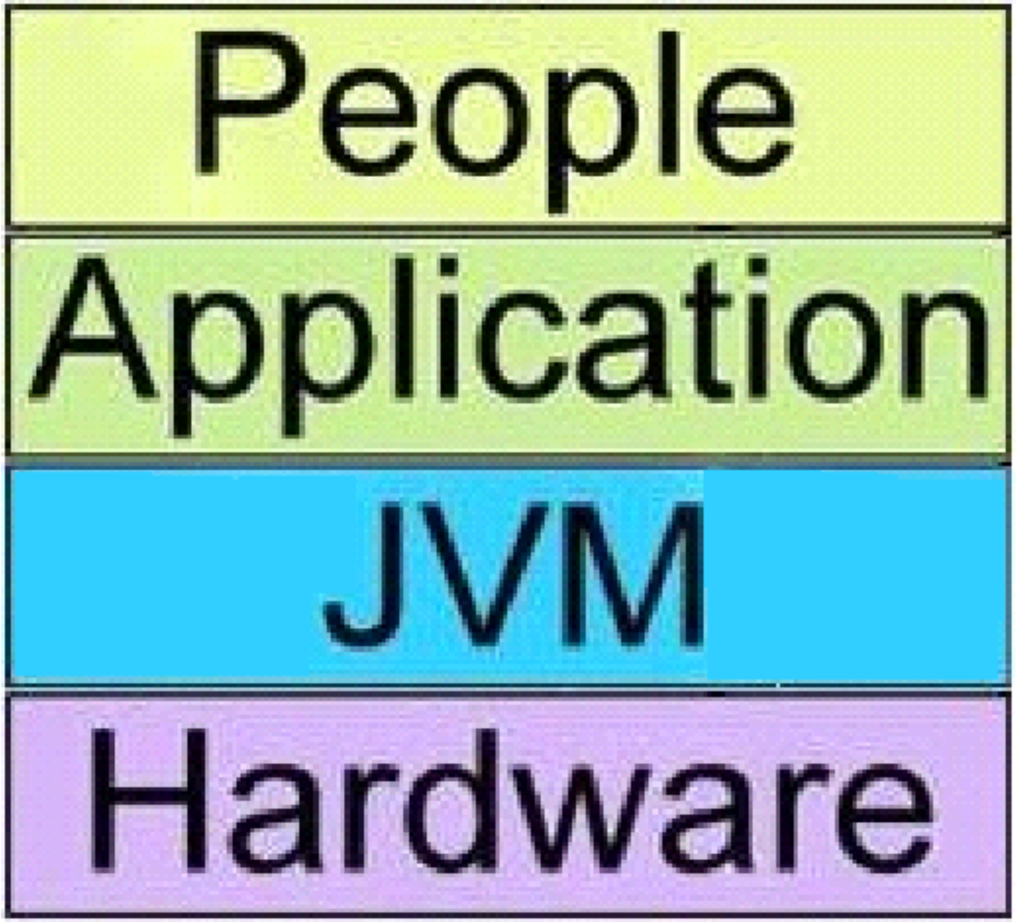
Importance of Tooling

- Tooling allows us to see what is otherwise invisible

Importance of process

- Process or ways of investigating the problem can change or hide the problem
- Systematic investigation

Holistic View



People

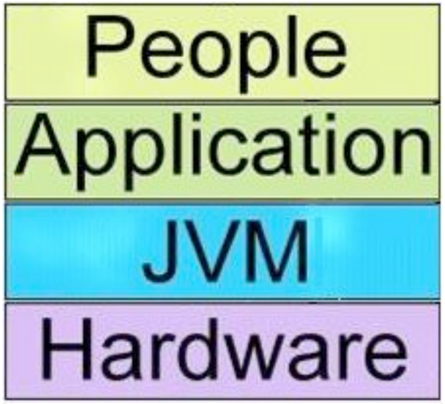
Application

JVM

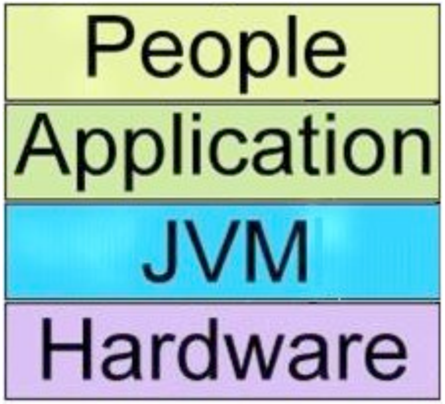
Hardware

Dynamic Nature of Systems

- Systems by their nature are dynamic
  - Mix of static and dynamic elements
- Static aspects of a Java based system
  - Not bottlenecks onto themselves
  - Hardware/OS
    - Defines the physical constraints of the system
  - Java Virtual Machine
    - Primarily a translation layer
  - Application
    - Expression of what is needed to be done



Dynamic Nature of Systems



People

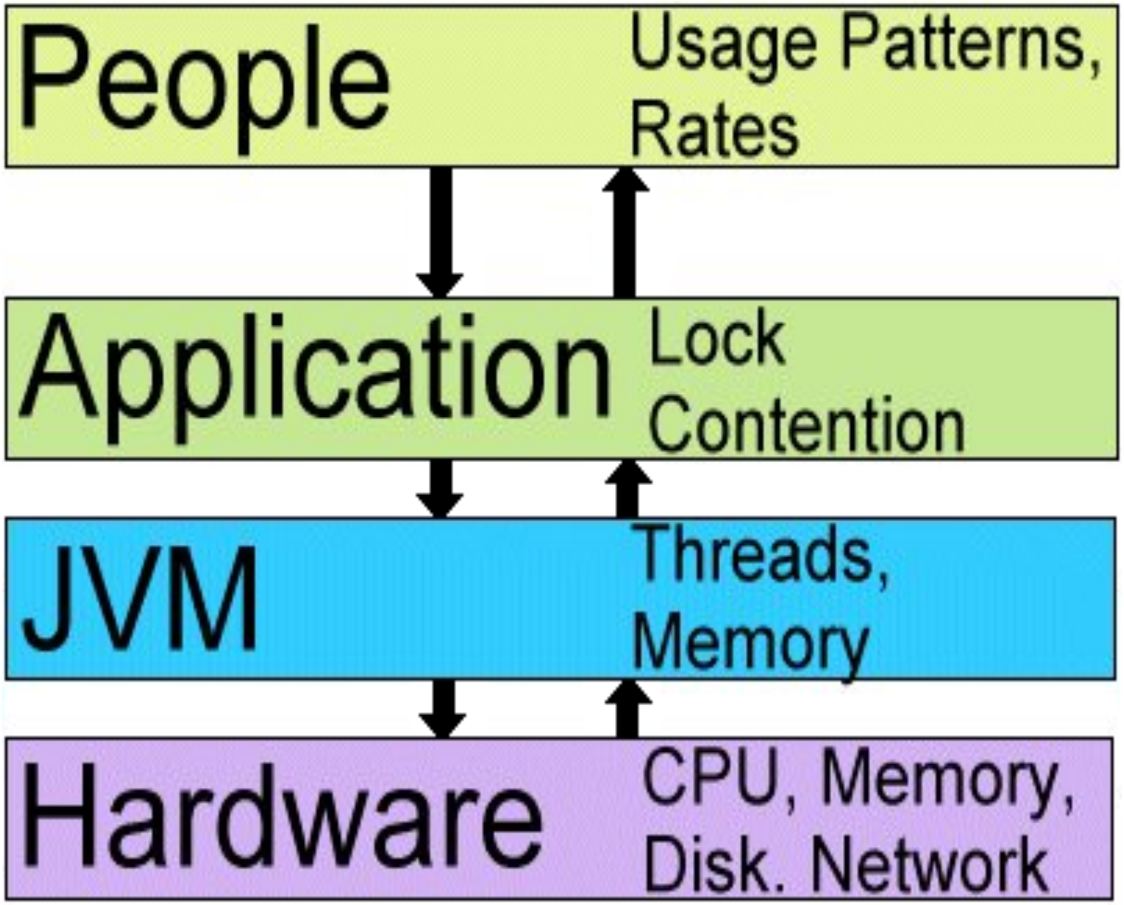
Application

JVM

Hardware

- Dynamic aspects of a system
  - People
    - Abstraction for system drivers
      - Batch processing
      - External systems
    - create flows through the system
      - maybe beyond the capacity of the system
      - Can put pressure on pinch points (or bottlenecks) in the system
- How does this work?

Resource Contribution



People

Usage Patterns, Rates

Application

Lock Contention

JVM

Threads, Memory

Hardware

CPU, Memory, Disk. Network

Forward Propagation of Actions

- People drive the application
- Application drives the JVM
  - Direct consequence of what the people are asking
  - And how application was coded
- JVM Drives the hardware
  - Direct consequence of what the application is asking
  - And how JVM was coded and configured
- Hardware executes instructions
  - Limited by speed and capacity

Backward Propagation of Problems

- Problem: hardware lacks capacity or is slow
  - people experience poor response times
- Problem: JVM is poorly configured
  - People experience poor response times
- Problem: Application suffers from contention
  - People experience poor response time
- Our starting point; people are experiencing poor response times
- How do we start our investigation?
  - It is at this point JoGoSlo ran into trouble

Performance Anti-pattern: Shot in the Dark

- Developers dove into the code
  - Found many ugly bits
    - Interactions with database
  - Wasted valuable time fixing them
    - None of the ugly code bits had any consequence on performance
  - Ignored key pieces of information
    - DBA reported millisecond response times
    - System sometimes recovered
- Developers started guessing at the cause of the problem

Solution to Shot in the Dark

Measure

Don’t Guess

Measure Don’t Guess

- Solid Measurements
  - Show you what needs to be done
  - Focus efforts
  - Facilitate planning
  - Instill confidence
  - Deflect finger pointing

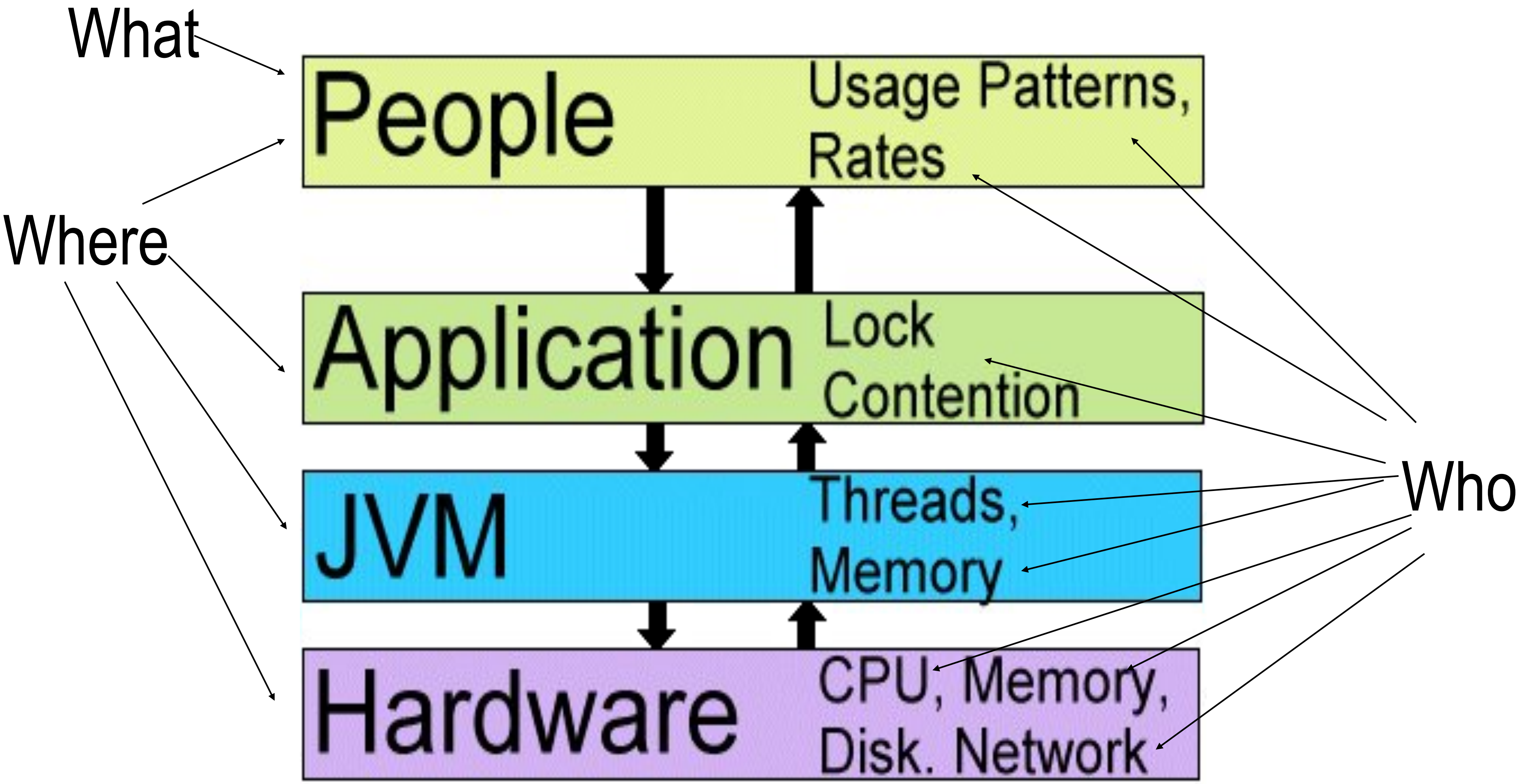
Measure Don’t Guess

- Review all performance requirements
- Construct a realistic test environment
- Use “The Box” as a roadmap
- Tackle one layer at a time
- Start with the people
- Start the investigation with the hardware
  - ➲ Work up the stack
- Let the user experience guide all decisions

Investigative W5

- Five questions asked by investigators:
  - Who ?
    - Who (which resource) is exhibiting the problem?
  - What ?
    - Observation: what do the users see?
  - Where ?
    - Which layer is exhibiting the problem?
  - When ?
    - Are there any peculiarities about when the problems occur?
  - Why ?
    - An explanation (hypothesis) of the observation from system perspective

Actors in the Performance Profile



Simple Process

- Form a hypothesis from observed behavior
- Devise a test to validate the hypothesis
- Measure for effect
- Make changes
- Test for desired effect
- Repeat until performance profile is in tolerance

People | Usage Patterns, Rates

- Provide the dynamics for the system
  - Use system in their own way
  - Use the system at their own leisure
- Need to capture the dynamics
- Usage pattern
  - Sequence of user actions
  - Timing information
    - Pauses between actions
    - Time of day for activity

People

- System utilization is an aggregate of all usage patterns
  - How system copes with the aggregation defines its performance profile
- Stress testing
  - Use mix of usage patterns to load the system
    - Ideally driven by a load testing tool
  - Measure system activity
    - Careful use of a selected tools
  - Must be run against a production like environment
- Goal: understand the user experience

Stress Testing Environment

- Production environment?
  - Not desirable and usually not an option
- Test environment should
  - Perfectly resemble your production environment
    - Data sizes, memory sizes, cache sizes, disk speeds, network speeds, should be the same
  - Be isolated
    - – Introduce other systems/processes in a controlled fashion

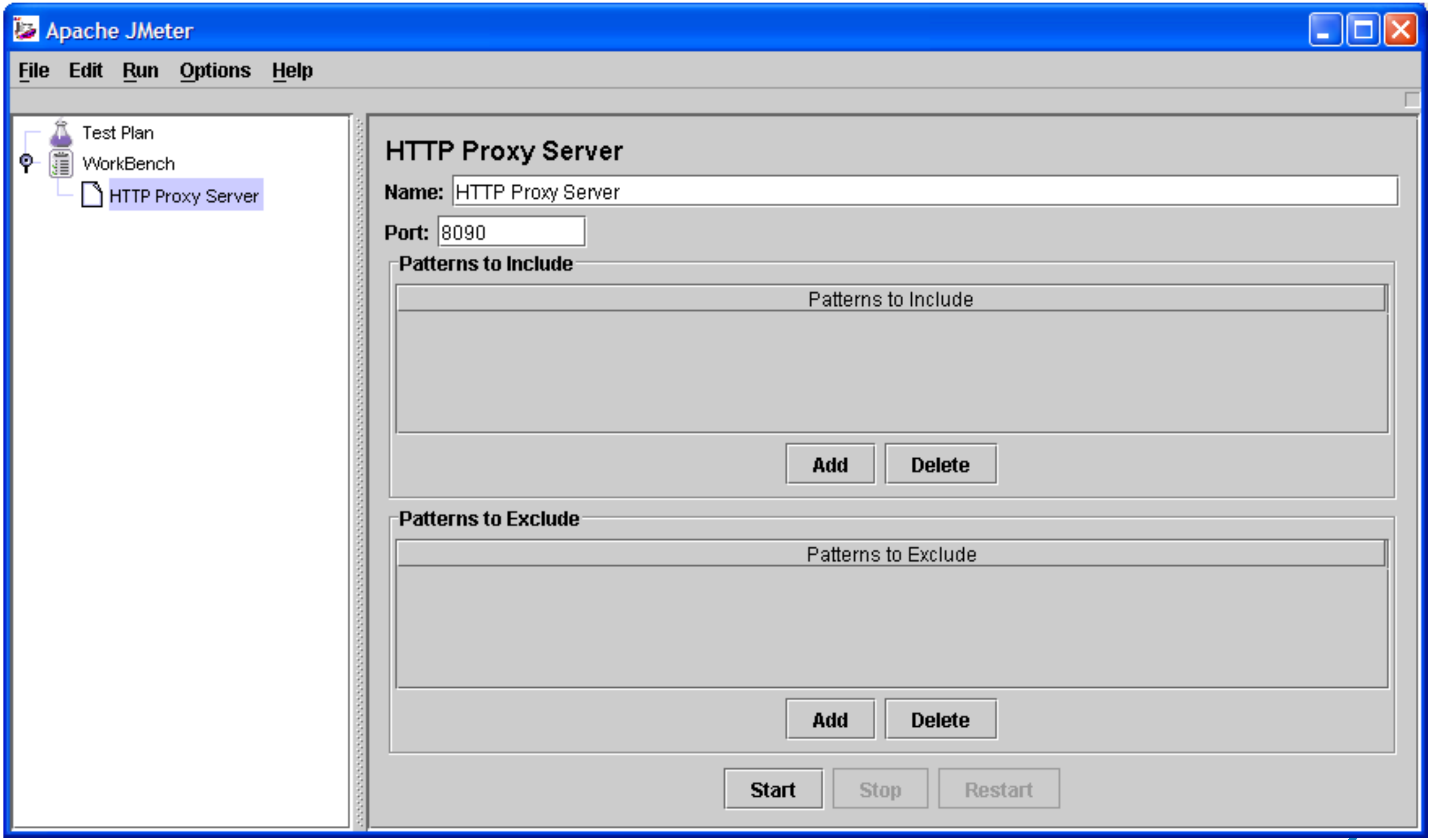
Stress Testing Environment

- Caching
  - Protects your application from an underlying slower technology
  - Reduces response times
  - May reduce the effects of I/O waits
- May need to simulate external systems
  - Do this with care
- Don’t extrapolate!
  - Difficult to know when you will hit the wall
  - E.g. Application using 15Mbits is moved from a gigabit to 10Megabit network
    - Shifts the bottleneck

Stress Testing

- Stress testing tool feature list
  - Easily scripted to support many users doing many different things
  - Supports randomization of inputs
  - Throttles request rates
  - Randomized request rates
  - Reports on response times (from clients perspective)
  - Vary loads
  - Generate high loads
- Introduced Apache JMeter to JoGoSlo

Apache JMeter

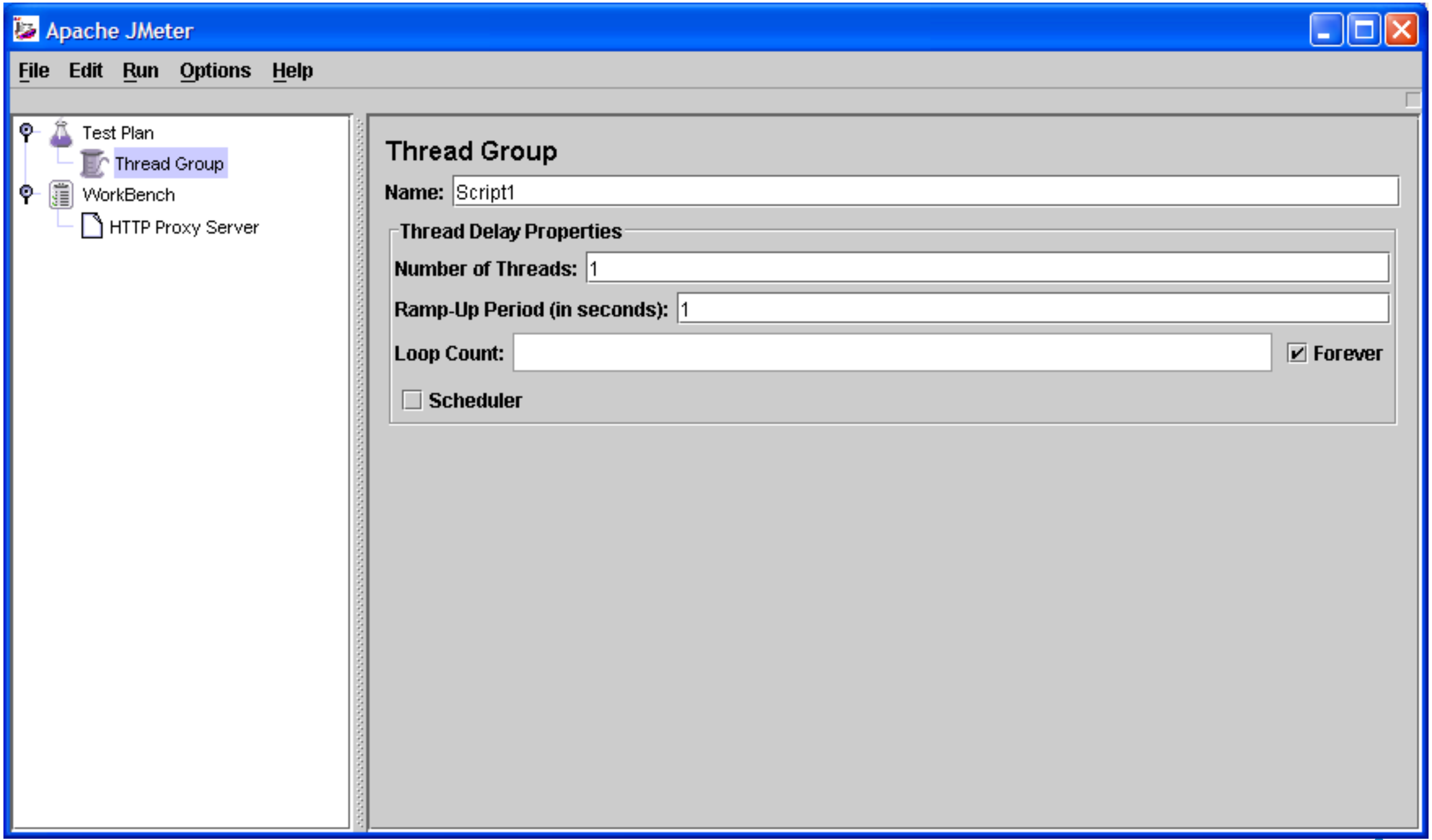


HTTP Proxy Server

Name: HTTP Proxy Server

Port: 8090

Apache JMeter



Thread Group

Name: Script1

Thread Delay Properties

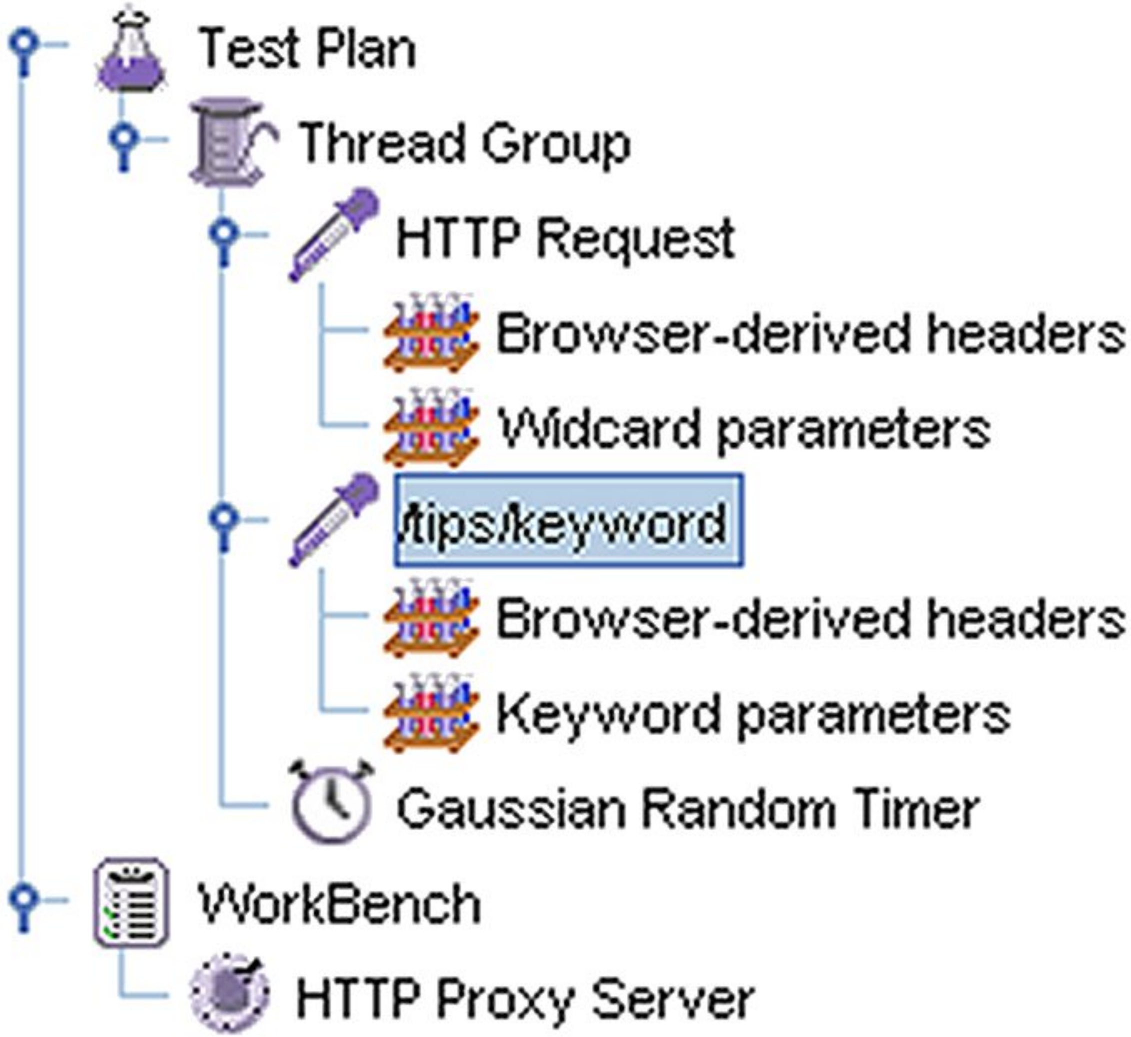
Number of Threads: 1

Ramp-Up Period (in seconds): 1

Loop Count: ☑ Forever

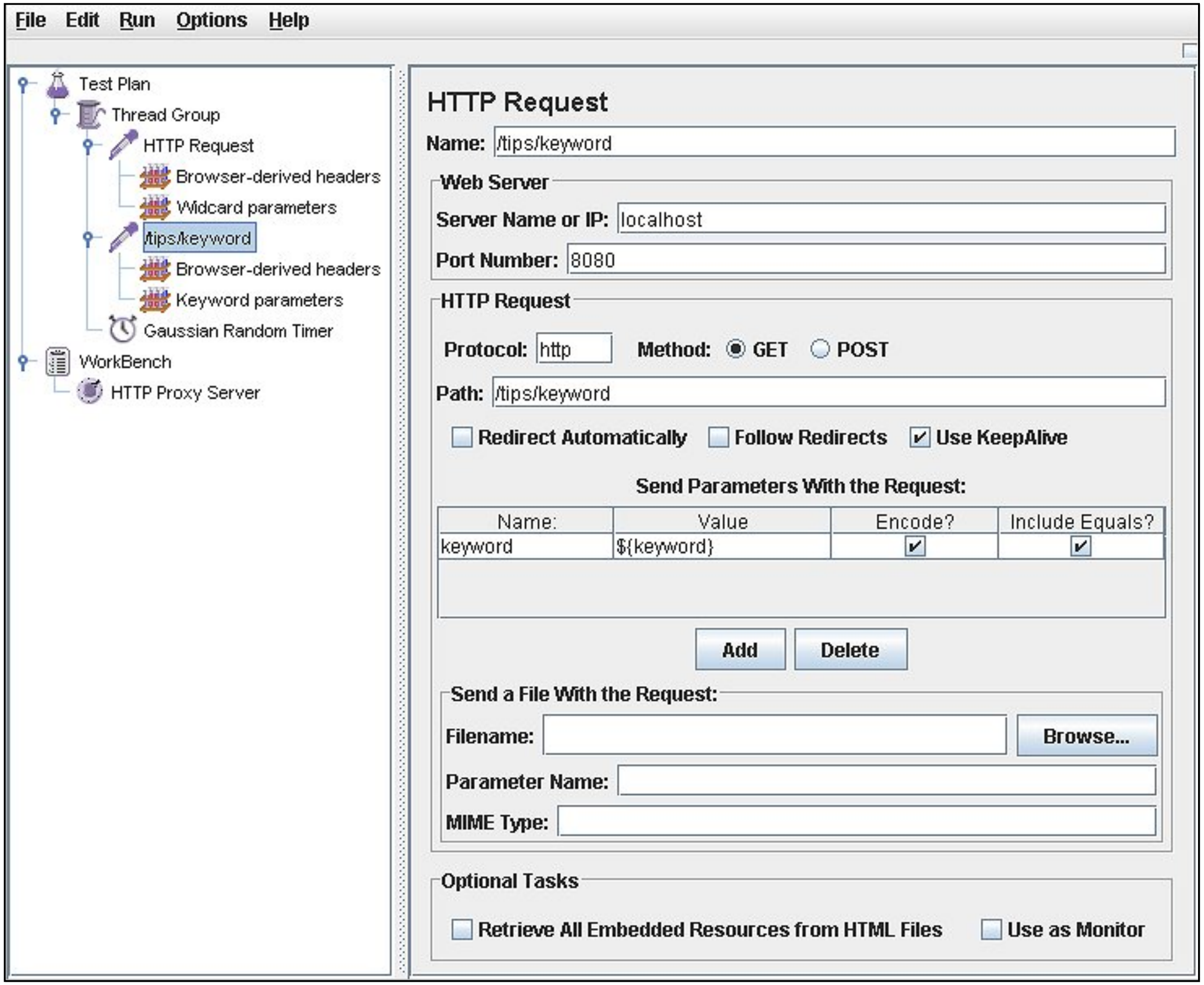
□ Scheduler

Apache JMeter



- Test Plan
  - Thread Group
    - HTTP Request
      - Browser-derived headers
      - Widcard parameters
    - /tips/keyword
      - Browser-derived headers
      - Keyword parameters
    - Gaussian Random Timer
- WorkBench
  - HTTP Proxy Server

Apache JMeter



HTTP Request

Name: /tips/keyword

Web Server

Server Name or IP: localhost

Port Number: 8080

HTTP Request

Protocol: http   Method: ● GET ○ POST

Path: /tips/keyword

□ Redirect Automatically □ Follow Redirects ☑ Use KeepAlive

Send Parameters With the Request:

Name:	Value	Encode?	Include Equals?
keyword	${keyword}	☑	☑

Send a File With the Request:

Filename:

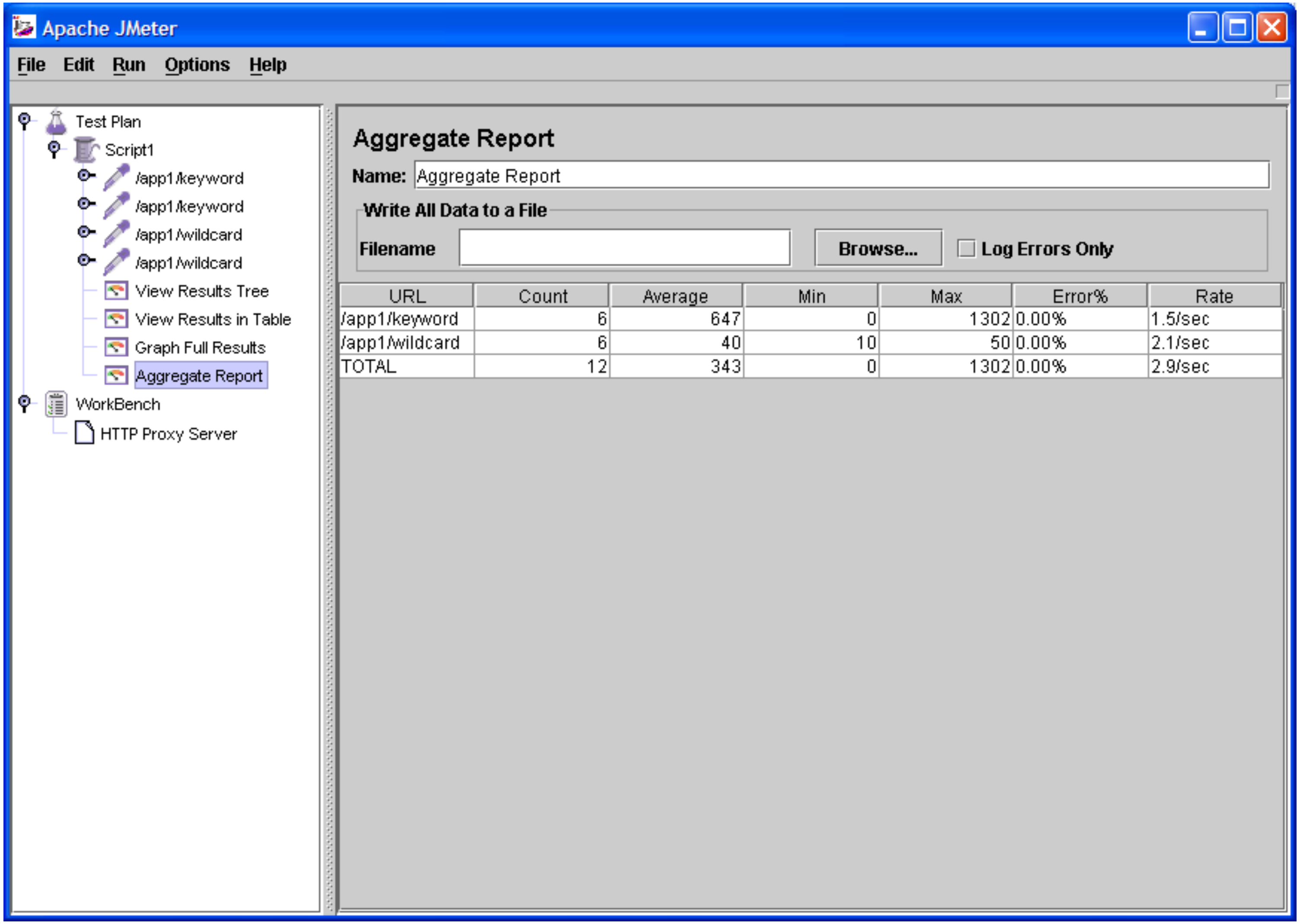
Parameter Name:

MIME Type:

Optional Tasks

□ Retrieve All Embedded Resources from HTML Files □ Use as Monitor

Apache JMeter



Aggregate Report

Name: Aggregate Report

Write All Data to a File

Filename

Browse...

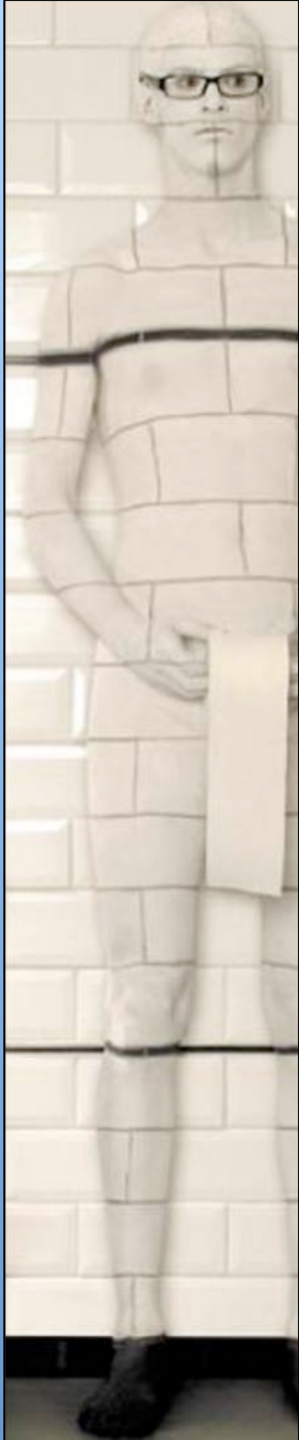
□ Log Errors Only

URL	Count	Average	Min	Max	Error%	Rate
/app1/keyword	6	647	0	1302	0.00%	1.5/sec
/app1/wildcard	6	40	10	50	0.00%	2.1/sec
TOTAL	12	343	0	1302	0.00%	2.9/sec

Apache JMeter Simple Setup

- Setup proxy
- Use browser to generate desired traffic
- Add in timers
- Randomize input
- Add in listeners
- Configure ThreadGroup properties
- Run load test

Practical



Instructions

- Start up apachejmeter.bat
- We'll skip the proxy setup. Load mixed.jmx JMeter plan
- Add random delay that ranges between 1 and 4 seconds between calls for both keyword and wildcard
- Add in a listener of your choice
- Use 2 threads (concurrent users)
  - Don't forget to set the repeat count
- Run and watch

Hardware

CPU, Memory, Disk. Network

- Hardware is our physical constraint
- If we don't have enough
  - Get more
  - Reduce utilization of what we have
    - Strength reduction (algorithms)
    - Trade one resource for another
      - Caching trades memory for I/O
- Judge utilization in relation to the task at hand
  - Reading 1 megabyte from disk should not stress a modern I/O channel
    - Are you really reading 1 meg?

Measuring Hardware Unix

- System activity
  - Maintained in kstat structures by the kernel
  - Collection of counters
- Reported on by command line tools
  - Includes vmstat, mpstat, iostat
  - Values reported as activity since last call
  - Provides instantaneous view on how hardware is coping with load

Measuring Hardware Windows

- System activity
  - Maintained in registry
  - Collection of counters
- Reported on by taskmgr and perfmon
  - Graphical windows on system performance
  - perfmon is configurable
  - Taskmgr has few configurations
    - You can (and should) turn on reporting of system time (CPU)

CPU

- High utilization is easily measurable
    - vmstat (Unix) or taskmgr (Windows)
- Different types of utilization
    - Application
    - JVM
    - System/OS

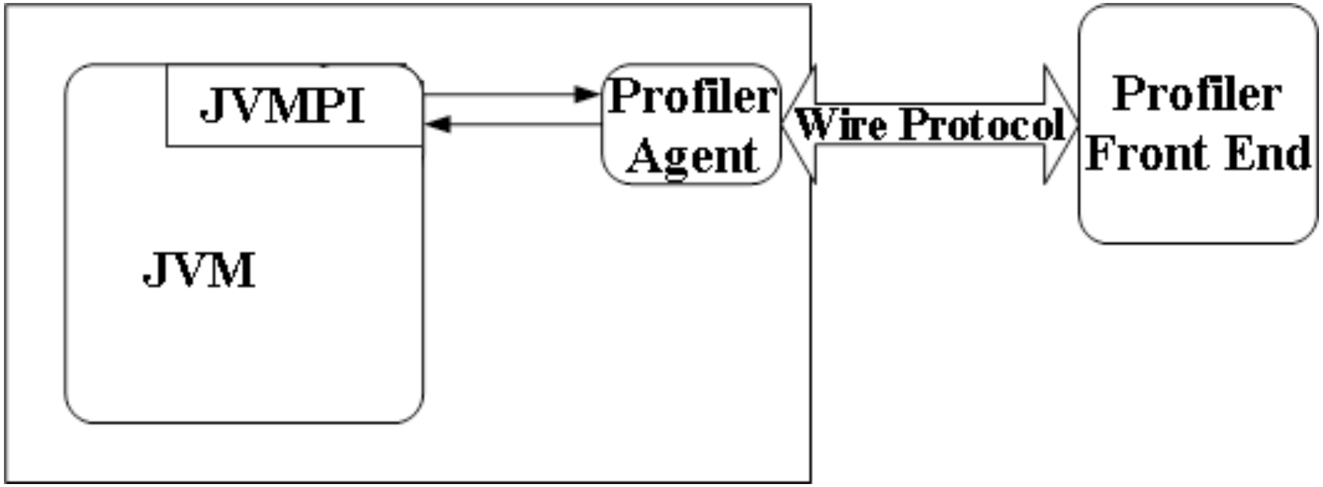
Application

- Source
  - Heavy workload
    - Add CPU
  - Remove processes from machine
  - Inefficient algorithms
    - Use method profiler to identify bottlenecks.
      - prof
      - hprof
      - NetBeans (JFluid)

Application Profiling

- JVMTI interface
  - New to 1.5
  - Combination of old JVMPI and JVMDB interfaces
  - Supported by –Xrunyourlib:parameters
    - Loads yourlib (dll or so)
    - Initializes with parameters

Application Profiling



Application Profiling

- -Xprof
  - Original execution profiler
  - Sampling profiler
    - Adds 1 to a counter for each method when it is found at the top of stack
    - Timings are inclusive
  - Reports on a thread bases
  - Dumps report to screen when thread dies

Application Profiling

- -Xrunhprof
  - Original heap profiler
  - Extended for thread and execution profiling
  - Built off of JVMTI interface but no wire protocol
  - Much more data than prof
    - Best viewed with a profiling tool (HPJMeter)

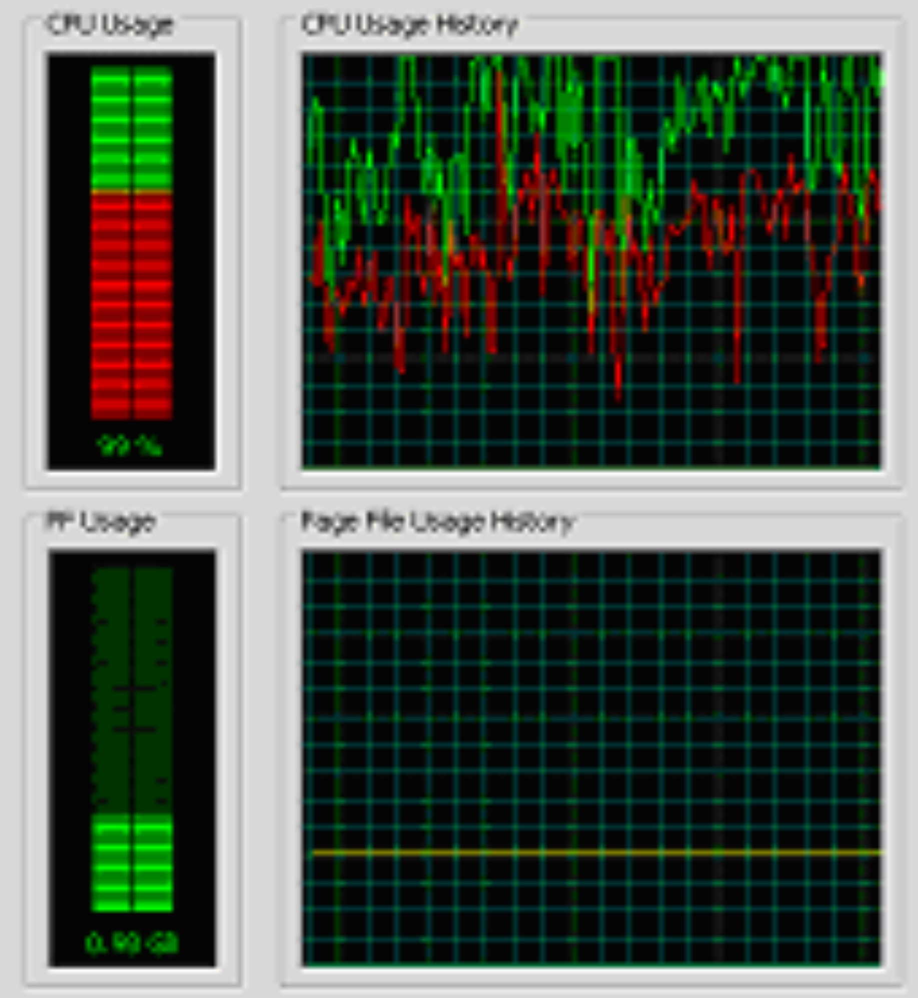
Java Virtual Machine

- Heavily threaded (measure with vmstat)
  - Runnable (r) queue consistently 2x number of CPUs
  - Stresses scheduler
  - Introduce thread pooling to limit activity
  - Reduce number of threads in current pools
- Java heap management
  - Monitor gc with –verbose:gc flag
  - View output with HPJTune

Operating System

- Context switching
  - Threads not completely using quantum
  - I/O
  - Lock acquisition
  - Interrupt handling
- Memory management
  - non-zero scan rates (sr) for more than a few seconds at a time
    - OS is thrashing

Operating System



Memory

- High utilization is easily measurable
  - memstat (Unix) or taskmgr (Windows)
  - Can look like high CPU utilization
- Real memory
- Virtual memory
  - An outdated optimization
- Ideally we want to pin JVM into real memory
  - Eliminate paging
  - Reduce memory utilization
  - Add memory

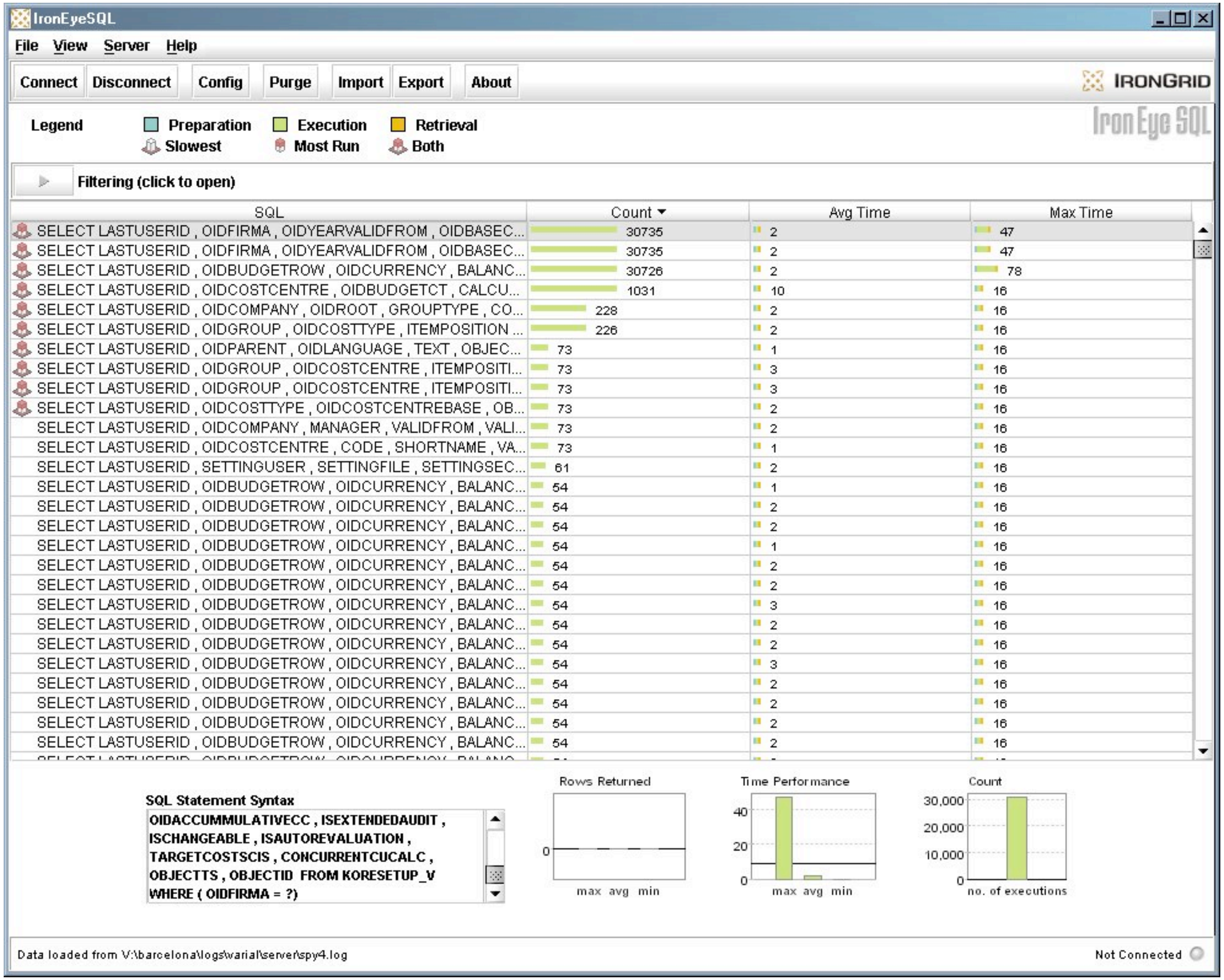
Disk and Network I/O

- Heavy utilization will most likely prevent application from fully utilizing CPU
- Source (iostat)
  - ➲ Reading/writing large data sets or many network calls
    - Use counters to calculate rates
    - Use I/O channel specs to understand capacity
    - For disk, introduce buffering in hardware or application
      - – E.g. Databases use paging
    - For network introduce caching
    - Bulk up operations
- Wrap I/O calls with timer

JDBC Monitoring

- Common problem is interactions with database
  - Can measure activity using JDBC interceptor
- P6Spy looks like a JDBC driver
  - Logs all JDBC calls
  - Logs can be viewed with IronEye

IronEye



Filtering (click to open)

SQL	Count ▾	Avg Time	Max Time
SELECT LASTUSERID , OIDFIRMA , OIDYEARVALIDFROM , OIDBASEC...	30735	2	47
SELECT LASTUSERID , OIDFIRMA , OIDYEARVALIDFROM , OIDBASEC...	30735	2	47
SELECT LASTUSERID , OIDBUDGETROW , OIDCURRENCY , BALANC...	30726	2	78
SELECT LASTUSERID , OIDCOSTCENTRE , OIDBUDGETCT , CALCU...	1031	10	16
SELECT LASTUSERID , OIDCOMPANY , OIDROOT , GROUPTYPE , CO...	228	2	16
SELECT LASTUSERID , OIDGROUP , OIDCOSTTYPE , ITEMPOSITION ...	226	2	16
SELECT LASTUSERID , OIDPARENT , OIDLANGUAGE , TEXT , OBJEC...	73	1	16
SELECT LASTUSERID , OIDGROUP , OIDCOSTCENTRE , ITEMPOSITI...	73	3	16
SELECT LASTUSERID , OIDGROUP , OIDCOSTCENTRE , ITEMPOSITI...	73	3	16
SELECT LASTUSERID , OIDCOSTTYPE , OIDCOSTCENTREBASE , OB...	73	2	16
SELECT LASTUSERID , OIDCOMPANY , MANAGER , VALIDFROM , VALI...	73	2	16
SELECT LASTUSERID , OIDCOSTCENTRE , CODE , SHORTNAME , VA...	73	1	16
SELECT LASTUSERID , SETTINGUSER , SETTINGFILE , SETTINGSEC...	61	2	16
SELECT LASTUSERID , OIDBUDGETROW , OIDCURRENCY , BALANC...	54	1	16
SELECT LASTUSERID , OIDBUDGETROW , OIDCURRENCY , BALANC...	54	2	16
SELECT LASTUSERID , OIDBUDGETROW , OIDCURRENCY , BALANC...	54	2	16
SELECT LASTUSERID , OIDBUDGETROW , OIDCURRENCY , BALANC...	54	1	16
SELECT LASTUSERID , OIDBUDGETROW , OIDCURRENCY , BALANC...	54	2	16
SELECT LASTUSERID , OIDBUDGETROW , OIDCURRENCY , BALANC...	54	2	16
SELECT LASTUSERID , OIDBUDGETROW , OIDCURRENCY , BALANC...	54	3	16
SELECT LASTUSERID , OIDBUDGETROW , OIDCURRENCY , BALANC...	54	2	16
SELECT LASTUSERID , OIDBUDGETROW , OIDCURRENCY , BALANC...	54	2	16
SELECT LASTUSERID , OIDBUDGETROW , OIDCURRENCY , BALANC...	54	3	16
SELECT LASTUSERID , OIDBUDGETROW , OIDCURRENCY , BALANC...	54	2	16
SELECT LASTUSERID , OIDBUDGETROW , OIDCURRENCY , BALANC...	54	2	16
SELECT LASTUSERID , OIDBUDGETROW , OIDCURRENCY , BALANC...	54	2	16
SELECT LASTUSERID , OIDBUDGETROW , OIDCURRENCY , BALANC...	54	2	16

SQL Statement Syntax

```
OIDACCUMMULATIVECC , ISEXTENDEDAUDIT ,
ISCHANGEABLE , ISAUTOREVALUATION ,
TARGETCOSTSCIS , CONCURRENTCUCALC ,
OBJECTTS , OBJECTID  FROM KORESETUP_V
WHERE ( OIDFIRMA = ?)
```

Data loaded from V:\barcelona\logs\varial\server\spy4.log

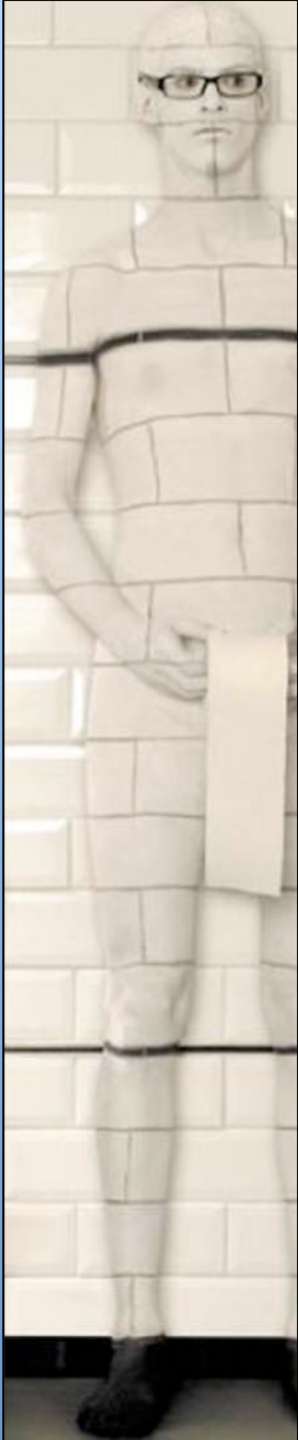
Not Connected

JAMon 2.2

- Specify JAMon JDBC driver
- Can be viewed using supplied WAR file
- To bind it in without code or config changes:
  - http://www.cretesoft.com/archive/newsletter.do?issue=136

'06 JAVAPOLIS
11 - 15 DECEMBER ■ ANTWERP ■ BELGIUM

Practical



Instructions

- Make sure Tips is running
- Use 30 threads (concurrent users)
  - Don’t forget to set the repeat count
- Run and watch the hardware
- What do we see?
- What do we do next?

JVM Threads, Memory

- If hardware is able to cope with the load, move to investigate JVM
- Threading
  - Maybe hints of problem when investigating hardware
  - Examine threading with kill –3 or ctrl-break
    - Dumps activity to console
    - Look for many busy threads
  - Control level of threading using thread pooling
    - Traffic calming

Java Heap Memory

- Java Virtual Machine C/C++ process
  - Structure depends upon OS
  - Shared text
  - Stack
  - Heap
    - Java Heap allocated from process heap
- Java object allocated from Java heap space
- Java heap space managed by garbage collection
  - Object that are no longer reachable will be collected
  - Memory that is no longer referenced will be returned to the free list

Java Heap Space

- C struct defines Java object
  - OOP
  - Contains references to other object
    - Depends on the class declaration

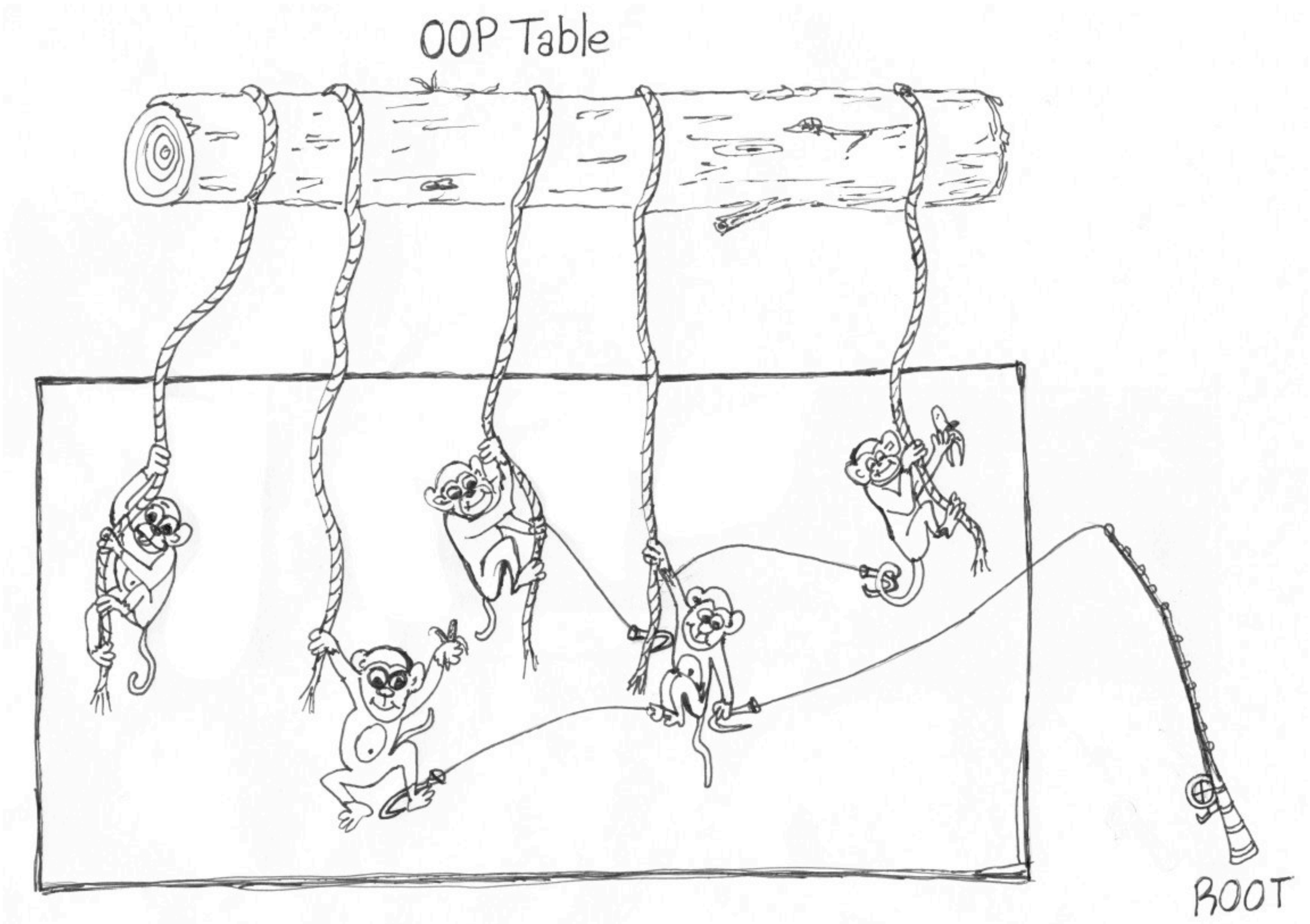
```
public class A {
    public Object x;
    public Object y;
}
```

```
struct OOP {
    int refCount;
    byte *refs;
} OOP, *OOP;
…
refs[0] = x;
refs[1] = y;
```

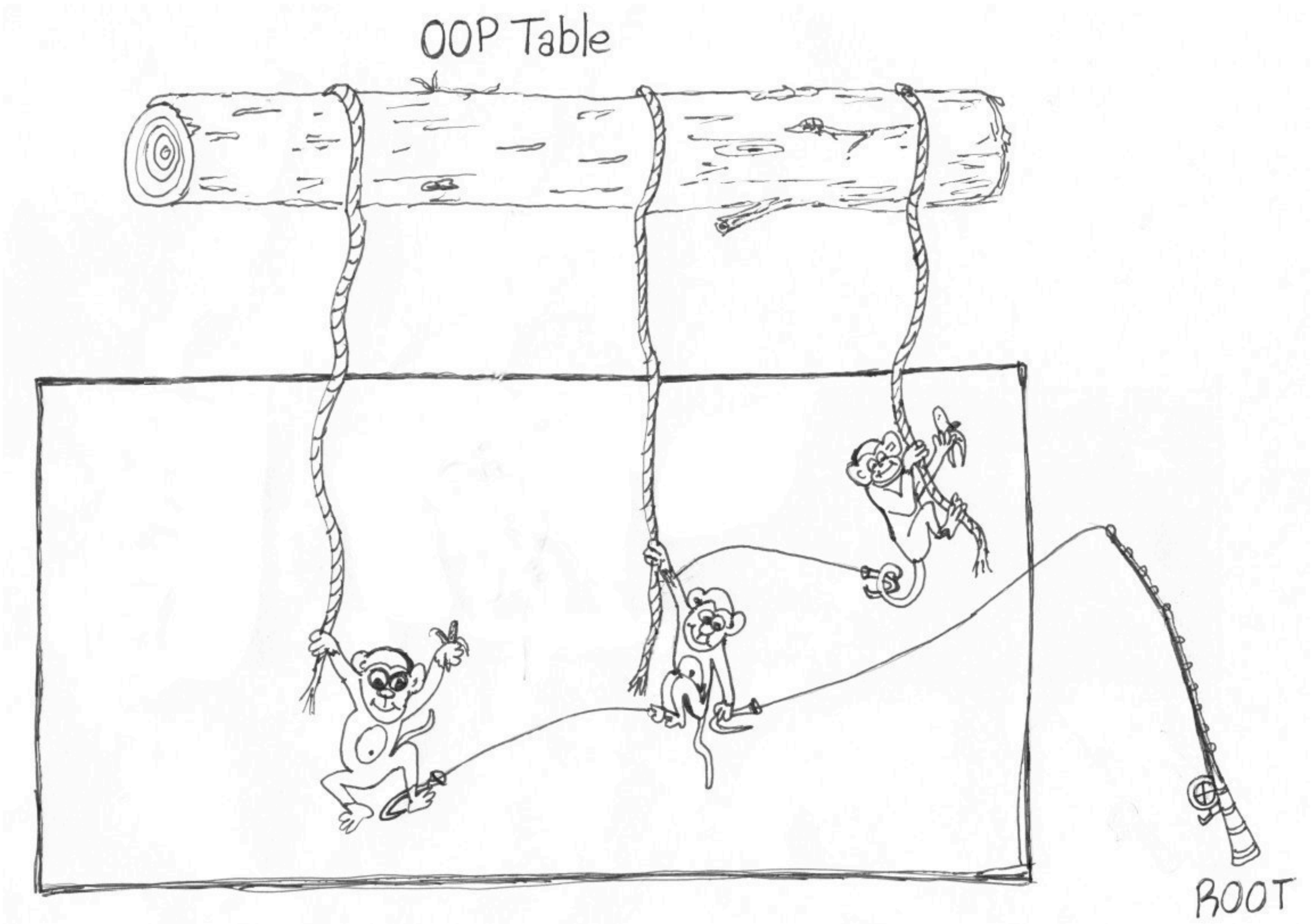
Java Heap Space

- Java heap maintains a references to OOP
  - Reference to all object maintained in OOP table
  - Root objects are at the top of object graphs
    - Define live objects
- Object not reachable from GC roots will be collected
  - Three step process known as Mark and Sweep:
    - Traverse OOP table and clear mark bit
    - Traverse object graphs starting at GC roots and set mark bit
    - Sweep across OOP table de-allocating OOP structures

Mark & Sweep GC



Mark & Sweep GC



Mark & Sweep GC

- Triggered on allocation failure
  - new Object(); fails
- Needs exclusive access to all of heap
  - Cannot share heap with application threads
  - Concurrency issue known as “stop-the-world” GC
- Single threaded
- Must manage entire heap space
  - Large heaps == long pauses

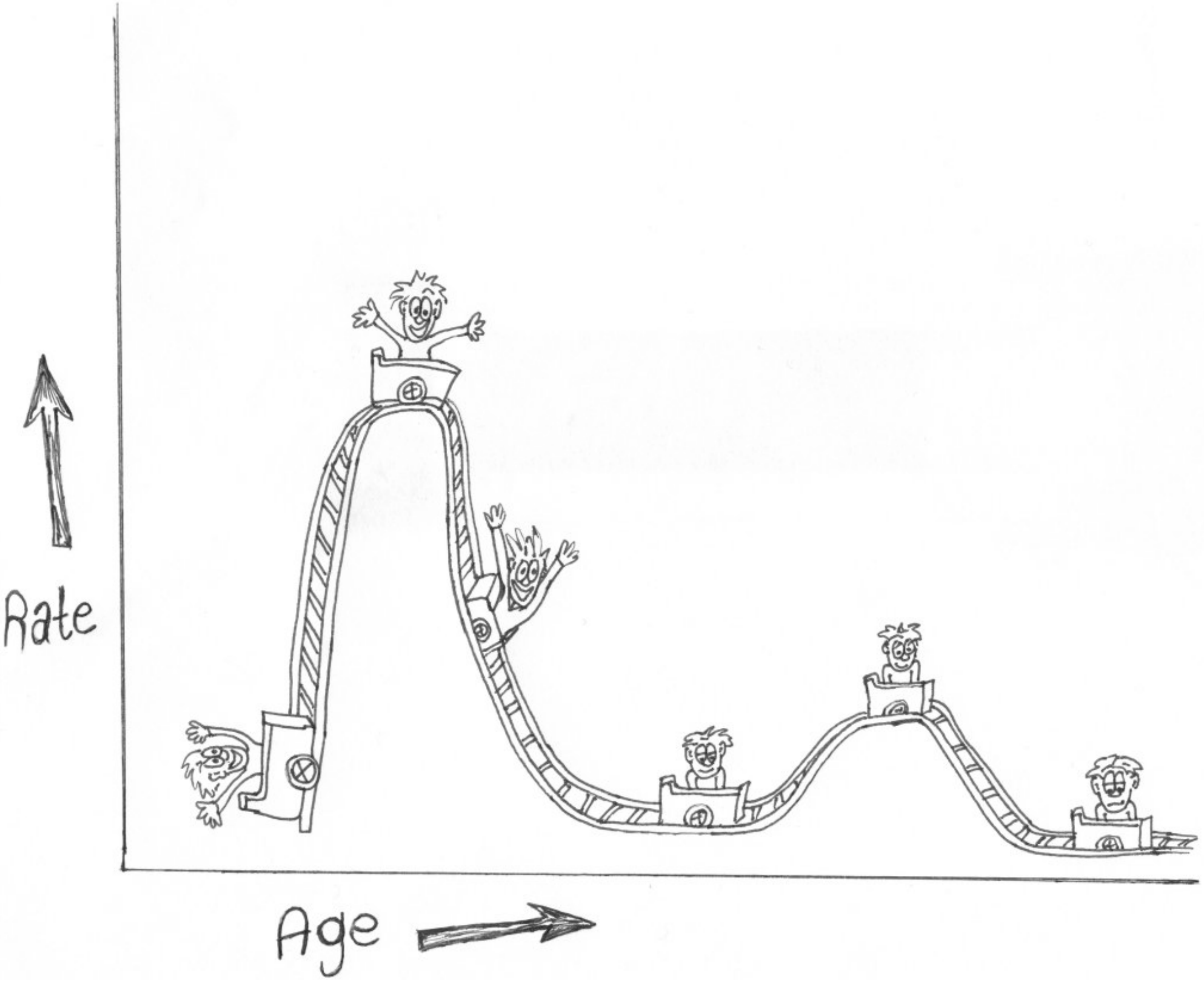
Mark & Sweep GC Optimizations

- When GC runs only 1 CPU is hot
  - Develop multi-threaded GC algorithms
  - Still have pause times but hopefully shorter
- Application pauses
  - Develop concurrent GC algorithms
  - Application and GC can run together
  - Reduced contention == reduced pause time
  - Higher overhead (ie trading CPU for shorter pause)

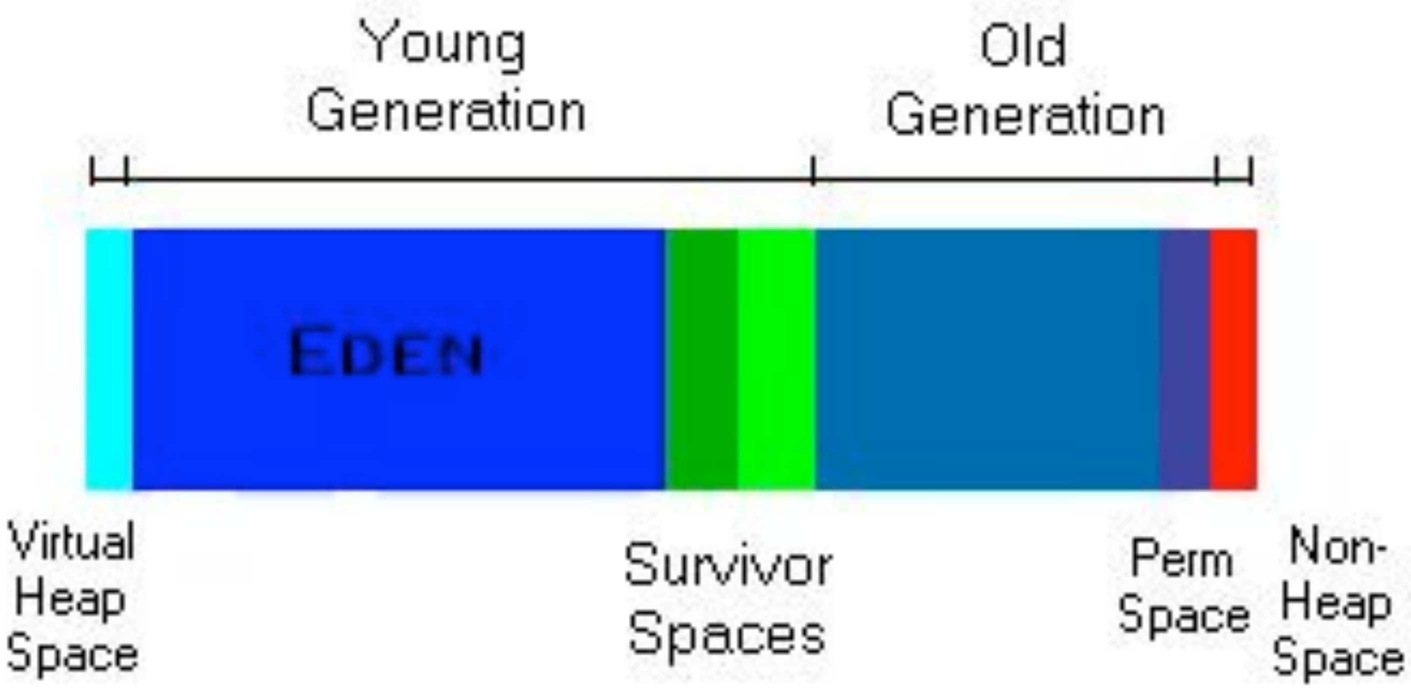
Mark & Sweep GC Optimizations

- Most Objects live for less than 100 μs or for a long time
  - IBM defines pinned clusters, wilderness (not so generational)
  - Sun/HP/JRocket added Generation Spaces
- Generational spaces
  - Choose a different collector for young and old
  - Collect young first
  - Collect old only when there will not be enough room for old objects

Object Lifespan



Sun Generational Spaces



Generational Spaces

- Heap sizing
  - Can size generational spaces using ratios or absolute sizes
- -Xmx defines maximum size of entire heap
- -XX:MaxNewSize=<N>
- -XX:NewRatio
  - Ratios: 8 for -client and 2 for -server
- -XX:SurvivorRatio
- -XX:PermSize=<size>
- -XX:MaxPermSize=<size>
- Old space is what is left over

Survivor Spaces

$$\text{Eden} = \text{New Size} - \frac{\text{New Size}}{\text{Survivor Ratio} + 2} * 2$$

e.g. new size = 2M, Survivor ratio = 8

Eden = 2M – 2M / (8 + 2) * 2

= 2048K – 204.8K

= 1843.2K

Each Survivor Space = 102.4K

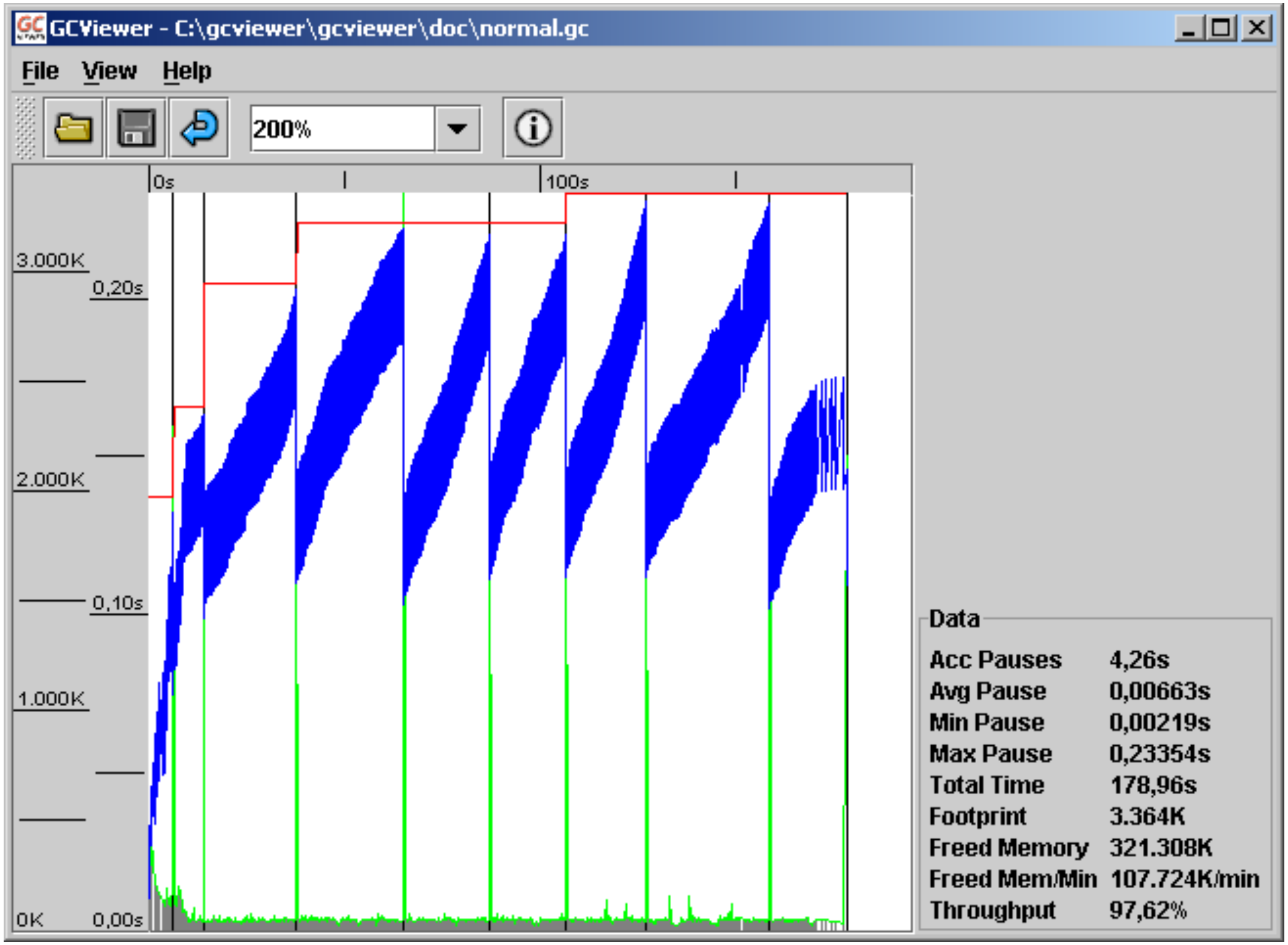
Monitoring GC

- -verbose:gc prints one log record for every GC event
    - -Xloggc:file
- Log entries provides a picture on how
    - your application is behaving
    - GC is coping
- Want to calculate GC throughput
- Want to find long GC pause times

GC Throughput

- “Time application is suspended by GC” divided by “total run time”
- E.g. 5 minutes of a 20 minute runtime is spent performing GC
- 25% efficiency
- GC bottleneck
- Requires many records to calculate
    - Better tooling
- GCViewer (TagTram)
- HPJTune (HP)

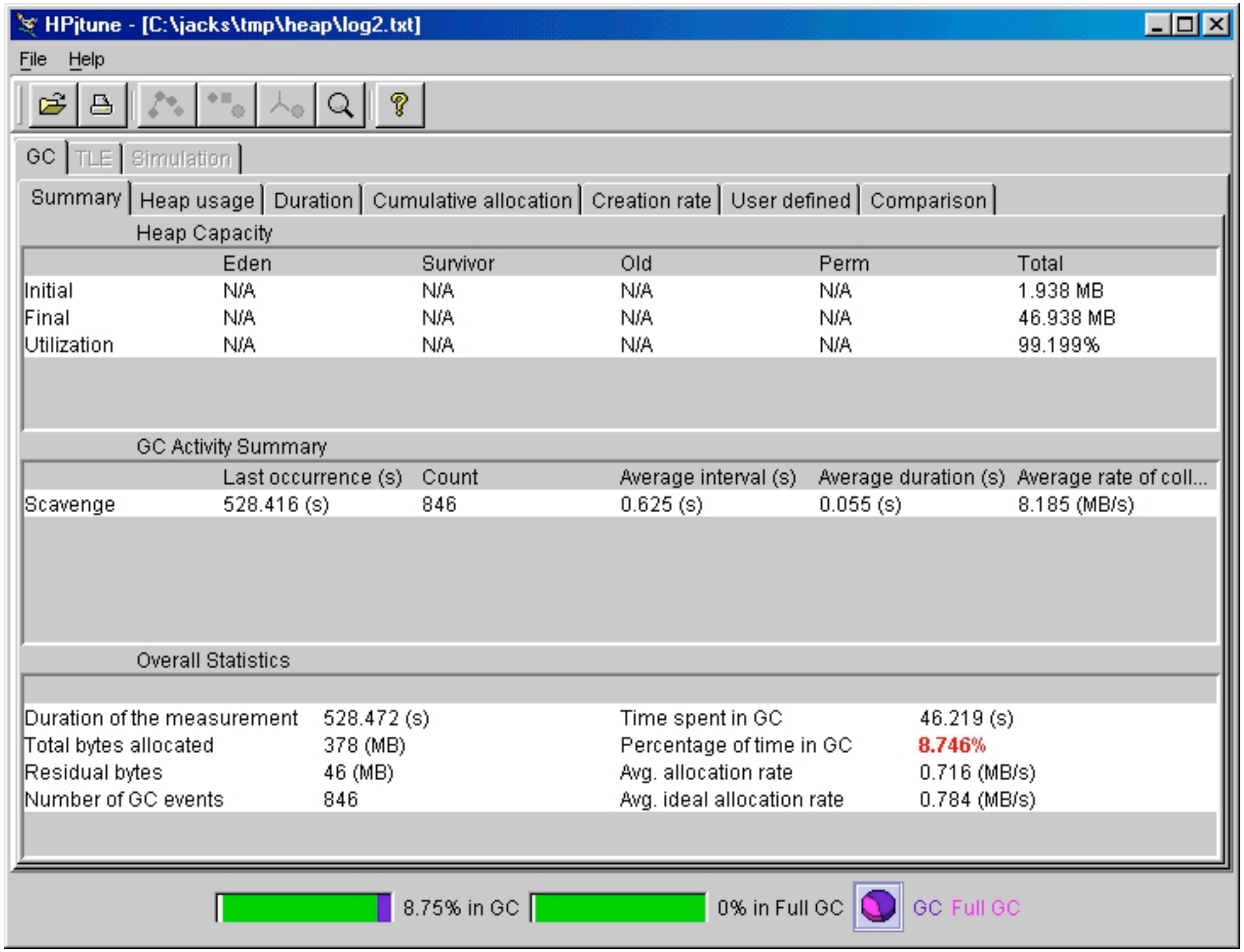
Tagtram GCViewer



Data

Acc Pauses	4,26s
Avg Pause	0,00663s
Min Pause	0,00219s
Max Pause	0,23354s
Total Time	178,96s
Footprint	3.364K
Freed Memory	321.308K
Freed Mem/Min	107.724K/min
Throughput	97,62%

HP JTune



HPjtune - [C:\jacks\tmp\heap\log2.txt]
File Help
GC TLE Simulation
Summary Heap usage Duration Cumulative allocation Creation rate User defined Comparison

Heap Capacity

	Eden	Survivor	Old	Perm	Total
Initial	N/A	N/A	N/A	N/A	1.938 MB
Final	N/A	N/A	N/A	N/A	46.938 MB
Utilization	N/A	N/A	N/A	N/A	99.199%

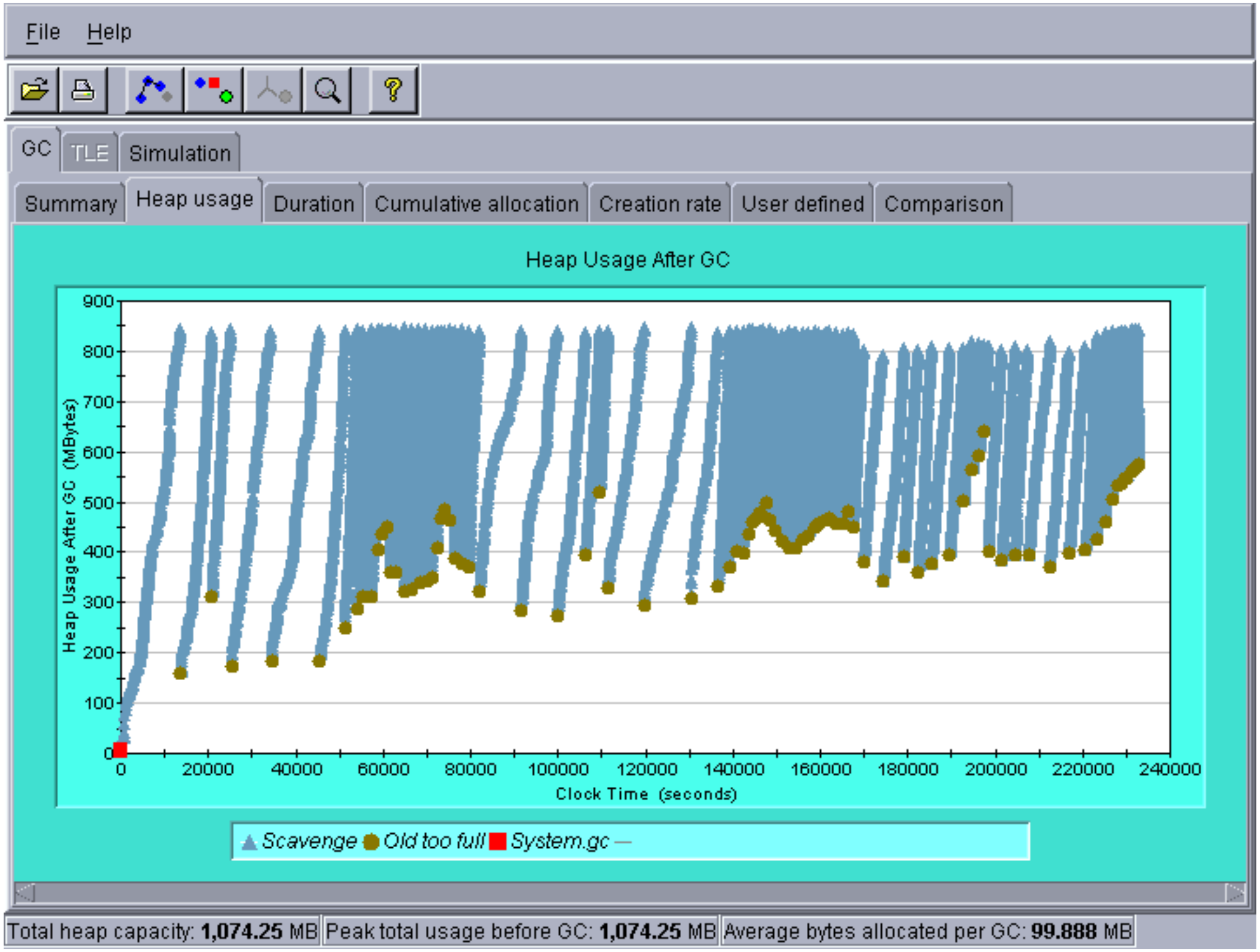
GC Activity Summary

	Last occurrence (s)	Count	Average interval (s)	Average duration (s)	Average rate of coll...
Scavenge	528.416 (s)	846	0.625 (s)	0.055 (s)	8.185 (MB/s)

Overall Statistics

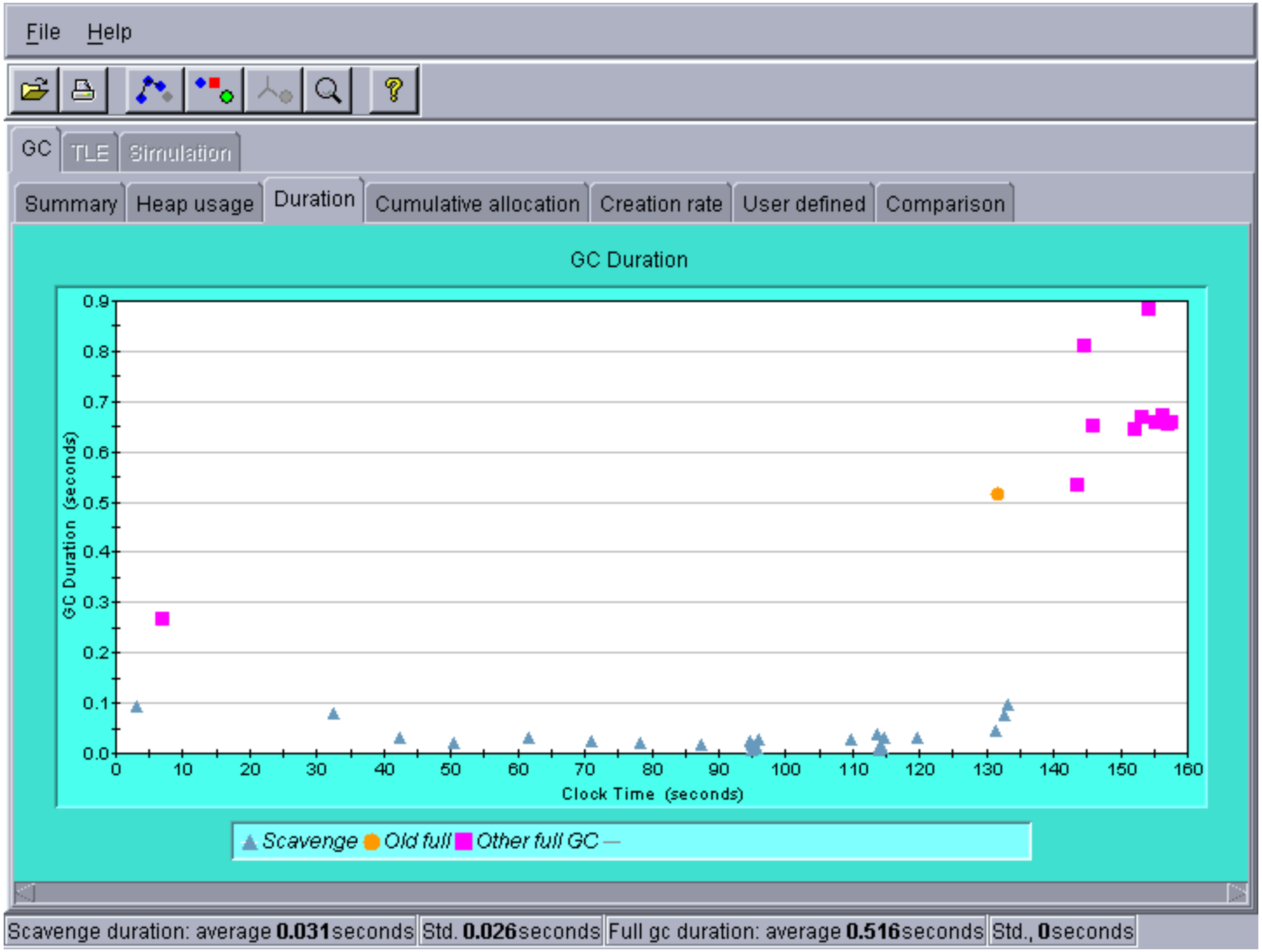
Duration of the measurement	528.472 (s)	Time spent in GC	46.219 (s)
Total bytes allocated	378 (MB)	Percentage of time in GC	8.746%
Residual bytes	46 (MB)	Avg. allocation rate	0.716 (MB/s)
Number of GC events	846	Avg. ideal allocation rate	0.784 (MB/s)

HP JTune Heap Usage After GC



Total heap capacity: **1,074.25** MB | Peak total usage before GC: **1,074.25** MB | Average bytes allocated per GC: **99.888** MB

HP JTune Pause Time



Scavenge duration: average **0.031**seconds | Std. **0.026**seconds | Full gc duration: average **0.516**seconds | Std., **0**seconds

Heap/GC Tuning

- Use graphics to decide how to tune memory
  - Let the user experience to temper your choices
- Strategy: eliminate full GC
  - Adjust size of total heap and survivor spaces
  - Tune other parameters as needed
- Strategy: eliminate long pauses
  - Use Parallel (if multi-cored)
  - Use concurrent if you can tolerate overhead

Heap/GC Tuning

- Tuning GC cannot eliminate
  - Extremely high rates of churn
  - Temporal or permanent memory leaks
- Need to fix the problem in the code
  - Use a memory profiler to direct your search
- -Xrunhprof:heap=all
  - Dumps heap when JVM exits
  - Dumps with kill -3 or ctrl-break
- -XX:+HeapDumpOnOutOfMemoryError
  - New for latest version of 1.6, 1.5, and 1.4

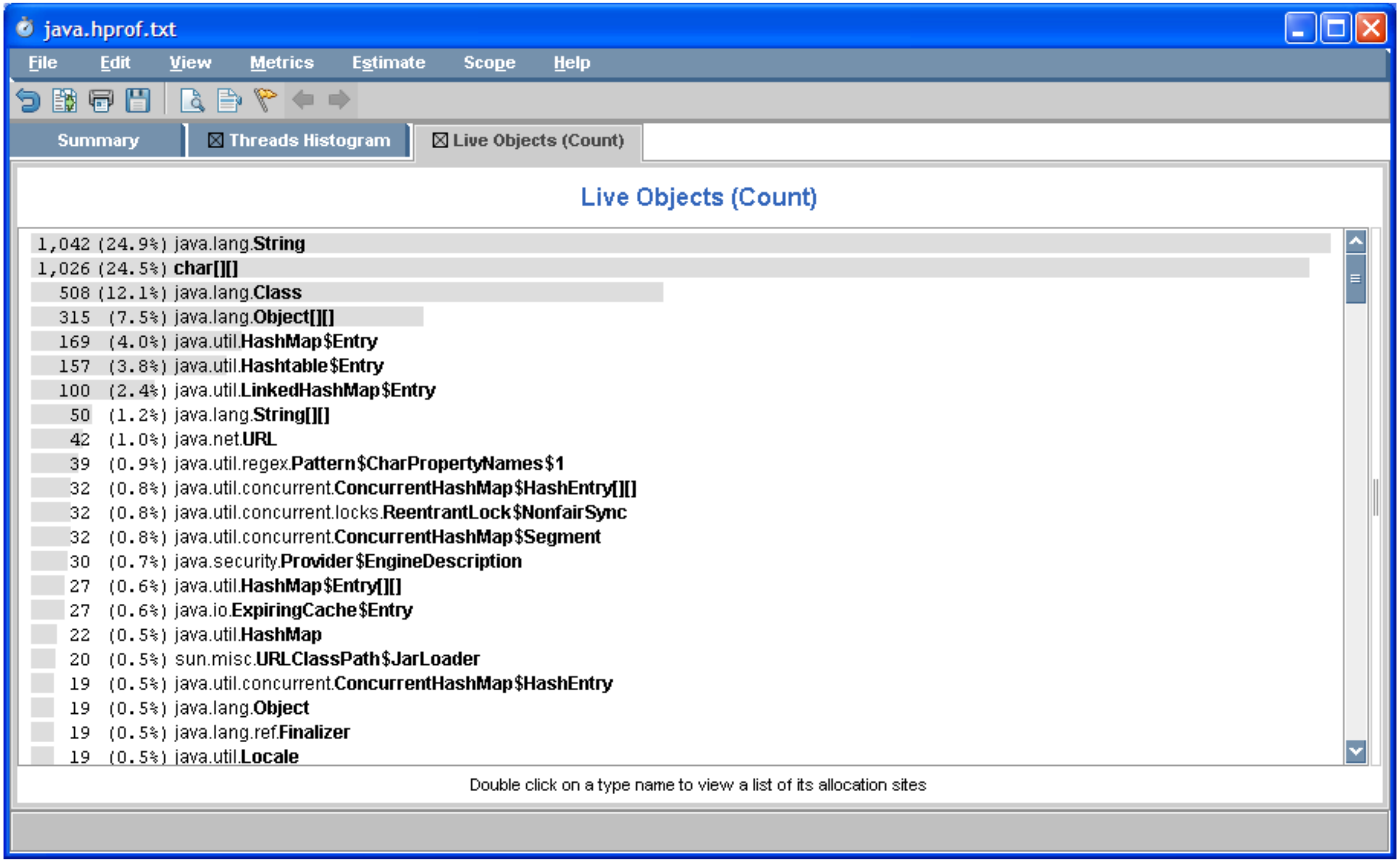
Heap Dump

- Contains enough information to reconstruct a picture of memory
- Picture contains references to all objects
  - Dead objects held by OOP table
  - Live objects
- Call GC twice before dumping heap
- Data volume and complexity calls for tooling
  - HPJMeter

HPJMeter

- Read hprof dump
  - Limited to single snapshot
- Provides rudimentary views of heap
  - Live object numbers and sizes
  - Dead objects numbers and sizes
- Can guess at memory leaks
  - Single snapshot analysis is limited
  - Can be good enough if you are methodical
- Memory leaks usually are found in collections
  - Strategy: focus on collections

HPJMeter Live Object View

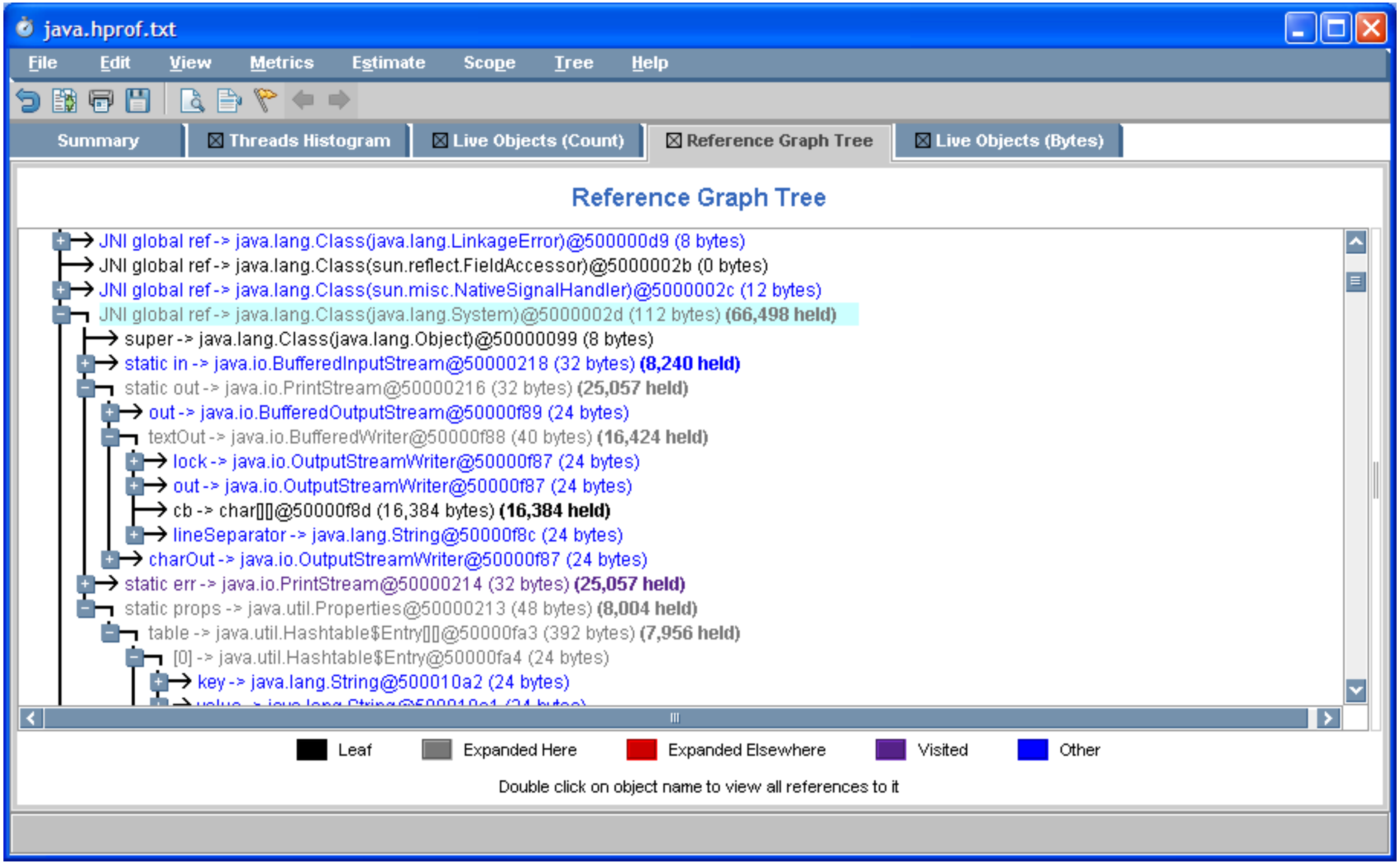


Live Objects (Count)

```
1,042 (24.9%) java.lang.String
1,026 (24.5%) char[][]
  508 (12.1%) java.lang.Class
  315  (7.5%) java.lang.Object[][]
  169  (4.0%) java.util.HashMap$Entry
  157  (3.8%) java.util.Hashtable$Entry
  100  (2.4%) java.util.LinkedHashMap$Entry
   50  (1.2%) java.lang.String[][]
   42  (1.0%) java.net.URL
   39  (0.9%) java.util.regex.Pattern$CharPropertyNames$1
   32  (0.8%) java.util.concurrent.ConcurrentHashMap$HashEntry[][]
   32  (0.8%) java.util.concurrent.locks.ReentrantLock$NonfairSync
   32  (0.8%) java.util.concurrent.ConcurrentHashMap$Segment
   30  (0.7%) java.security.Provider$EngineDescription
   27  (0.6%) java.util.HashMap$Entry[][]
   27  (0.6%) java.io.ExpiringCache$Entry
   22  (0.5%) java.util.HashMap
   20  (0.5%) sun.misc.URLClassPath$JarLoader
   19  (0.5%) java.util.concurrent.ConcurrentHashMap$HashEntry
   19  (0.5%) java.lang.Object
   19  (0.5%) java.lang.ref.Finalizer
   19  (0.5%) java.util.Locale
```

Double click on a type name to view a list of its allocation sites

HPJMeter Leak Detection



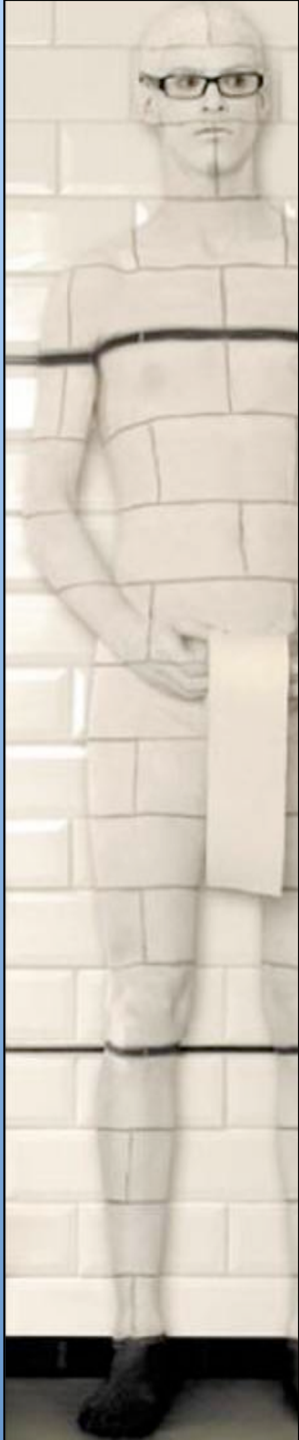
Reference Graph Tree

```
+→ JNI global ref -> java.lang.Class(java.lang.LinkageError)@500000d9 (8 bytes)
 → JNI global ref -> java.lang.Class(sun.reflect.FieldAccessor)@5000002b (0 bytes)
+→ JNI global ref -> java.lang.Class(sun.misc.NativeSignalHandler)@5000002c (12 bytes)
-  JNI global ref -> java.lang.Class(java.lang.System)@5000002d (112 bytes) (66,498 held)
    → super -> java.lang.Class(java.lang.Object)@50000099 (8 bytes)
   +→ static in -> java.io.BufferedInputStream@50000218 (32 bytes) (8,240 held)
   -  static out -> java.io.PrintStream@50000216 (32 bytes) (25,057 held)
      +→ out -> java.io.BufferedOutputStream@50000f89 (24 bytes)
      -  textOut -> java.io.BufferedWriter@50000f88 (40 bytes) (16,424 held)
         +→ lock -> java.io.OutputStreamWriter@50000f87 (24 bytes)
         +→ out -> java.io.OutputStreamWriter@50000f87 (24 bytes)
          → cb -> char[]@50000f8d (16,384 bytes) (16,384 held)
         +→ lineSeparator -> java.lang.String@50000f8c (24 bytes)
      +→ charOut -> java.io.OutputStreamWriter@50000f87 (24 bytes)
   +→ static err -> java.io.PrintStream@50000214 (32 bytes) (25,057 held)
   -  static props -> java.util.Properties@50000213 (48 bytes) (8,004 held)
      -  table -> java.util.Hashtable$Entry[]@50000fa3 (392 bytes) (7,956 held)
         -  [0] -> java.util.Hashtable$Entry@50000fa4 (24 bytes)
            +→ key -> java.lang.String@500010a2 (24 bytes)
            +→ value -> java.lang.String@500010a1 (24 bytes)
```

⬛ Leaf ▩ Expanded Here 🟥 Expanded Elsewhere 🟪 Visited 🟦 Other

Double click on object name to view all references to it

Practical



Instructions

- Let’s profile heap with JVM switch –Xrunhprof:heap=all
  - For fun, add switch –Xloggc:gc.log
- Restart application server and run JMeter plan
- Confirm that there is a memory leak with HPJTune
  - Open gc.log
  - Look at “Heap Usage After GC”
  - Look at “GC Duration”
- Open HPJMeter and find the leak
  - You may need to shut down everything first

Application Lock Contention

- The only problem left is lock contention
- Characterized by inability to utilize CPU
  - Similar to I/O bound (call to external system)
- High system time (% of total)
  - Locks are a kernel resource
- Find by performing a thread dump (kill -3)
  - For live lock you may need many thread dumps
- Techniques to educe lock contention is an emerging topic

People	Usage Patterns, Rates

- If you haven’t found anything
  - ➲ Re-investigate the people
    - Are they really doing what you think they are doing?
    - Read logs
    - Visit the floor and watch
    - Re-do usage patterns
      - – Compare JMeter scripts with real life
    - Re-test
  - ➲ Validate that QA == Production
    - Even the smallest difference can hide the problem

JoGoSlo Reload

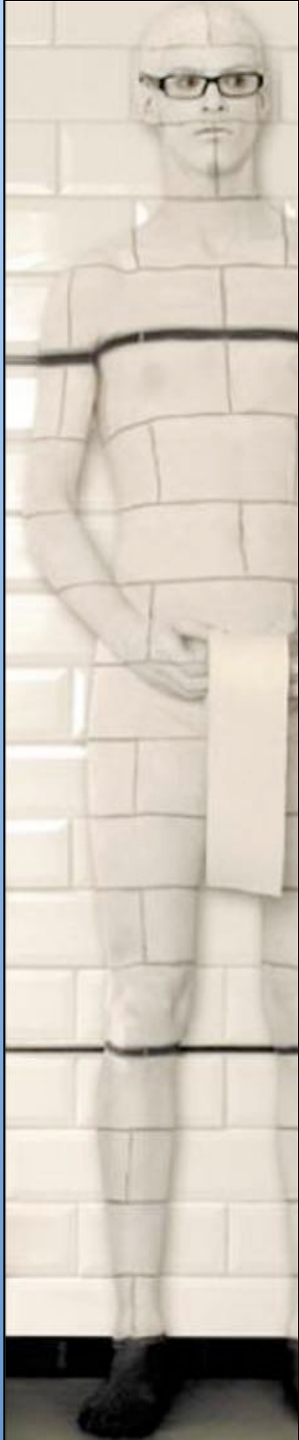
- Introduced Apache JMeter
- Introduced HPJTune to monitor memory
- Confirmed memory leak hypothesis
  - Resting the application allowed application to recover
  - Recovery was tied to HttpSessionState timeout
    - Developers were working on persistence framework
- Isolated memory leak to single usage pattern
  - Filtered off a vast majority of the application
  - Identified, fixed and re-tested with-in budget

Summary

- Systems are dynamic, code is static
- Be methodical
- Review performance requirements
- Prepare stress testing environment
- Define Usage patterns
- Investigate hardware, JVM, and Application
- Use measurements from tooling to direct your efforts
- Let the user experience guide your decisions

Measure
Don’t
Guess

Q&A



Thank you for your attention!

Adobe®

JBoss®
a division of Red Hat

ORACLE®

Sun® microsystems
The Network is the Computer™