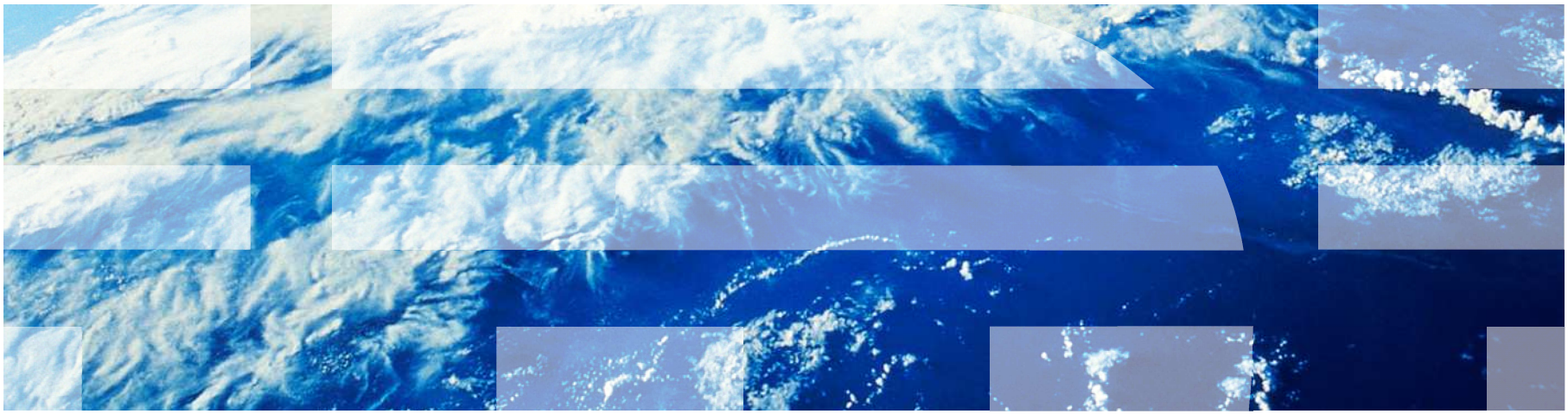
Inside IBM Java™ 7

Tim Ellison
Senior technical Staff Member
Java Technology Centre, UK.



* Java is the registered trademark of Oracle Corp

Important Disclaimers

THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY.

WHILST EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS PRESENTATION, IT IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED.

ALL PERFORMANCE DATA INCLUDED IN THIS PRESENTATION HAVE BEEN GATHERED IN A CONTROLLED ENVIRONMENT. YOUR OWN TEST RESULTS MAY VARY BASED ON HARDWARE, SOFTWARE OR INFRASTRUCTURE DIFFERENCES.

ALL DATA INCLUDED IN THIS PRESENTATION ARE MEANT TO BE USED ONLY AS A GUIDE.

IN ADDITION, THE INFORMATION CONTAINED IN THIS PRESENTATION IS BASED ON IBM’S CURRENT PRODUCT PLANS AND STRATEGY, WHICH ARE SUBJECT TO CHANGE BY IBM, WITHOUT NOTICE.

IBM AND ITS AFFILIATED COMPANIES SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS PRESENTATION OR ANY OTHER DOCUMENTATION.

NOTHING CONTAINED IN THIS PRESENTATION IS INTENDED TO, OR SHALL HAVE THE EFFECT OF:

- CREATING ANY WARRANT OR REPRESENTATION FROM IBM, ITS AFFILIATED COMPANIES OR ITS OR THEIR SUPPLIERS AND/OR LICENSORS

Agenda – Inside IBM Java 7

- High level goals for Java 7
- Base feature details
  - JSR 334 – Small language enhancements (Project Coin)
  - JSR 203 – More new I/O APIs for the Java platform (NIO.2)
  - JSR 292 – invokedynamic
  - JSR 166y – concurrency and collections updates
- Smaller features (TLS 1.2, UNICODE 6.0...)
- IBM feature details
  - Performance & platform exploitation – z196, POWER 7, ...
  - Garbage Collector updates & new policy - “balanced”
  - Technology Evaluation: WebSphere Real Time
  - Serviceability improvements & Tools overview
- Questions?

What is Java 7? The themes

- Major Java platform release, touching on all aspects of the language and JVM

From the draft JSR document (*), the goals:

- **Compatibility** - “Any program running on a previous release of the platform must also run *unchanged* on an implementation of Java SE 7”
- **Productivity** - “...promote best coding practices and reduce boilerplate... minimal learning curve...”
- **Performance** - “...new concurrency APIs... enable I/O-intensive applications by introducing a true asynchronous I/O API..”
- **Universality** - “...accelerate the performance of dynamic languages on the Java Virtual Machine.”
- **Integration** - “Java SE 7 will include a new, flexible filesystem API as part of JSR 203...”

(*) Available from jcp.org

New Language Constructs – project “coin”

- **It's a “bunch of small change(s)”**

- Strings in switch

```
switch(myString) {
   case “one”: <do something>; break;
   case “red”: <do something else>; break;
   default: <do something generic>;
}
```

- Improved Type Inference for Generic Instance Creation (diamond)

```
Map<String,MyType> foo = new Map<String,MyType>();
```

Becomes

```
Map<String,MyType> foo = new Map<>();
```

Coin continued...

- An omnibus proposal for better integral literals
  - Allow binary literals (`0b10011010`)
  - Allow underscores in numbers to help visual blocking (`34_409_066`)
  - Unsigned literals (`0xDEu`)

- Simplified Varargs Method Invocation
  - Moves warnings to method declaration, rather than on each user. Reduces unavoidable warnings.

Coin – multi-catch

- Developers often want to catch 2 exceptions the same way, but can't in Java 6:

```
try {
} catch(Exception a) {
    problemHandler(a);
} catch(Error b) {
    problemHandler(b);
}
```

- The following now works

```
try {
} catch(Exception|Error a) {
    problemHandler(a);
}
```

Coin – Automatic Resource Management

- Dealing with all possible failure paths is hard.
- Closing resources correctly in failure cases is hard.
- Idea: get the compiler to help, and define an interface on resources that knows how to tidy up automatically.

```
try(InputStream inFile = new FileInputStream(aFileName);
     OutputStream outFile = new FileOutputStream(aFileName)) {
    byte[] buf = new byte[BUF_SIZE];
    int readBytes;
    while ((readBytes = inFile.read(buf)) >= 0)
        inFile.write(buf, readBytes);
 }
```

NIO.2 – More new I/O APIs for Java (JSR 203)

- Goal: enable Java programmers to unlock the more powerful I/O abstractions.
- Asynchronous I/O
  - Delegated IO operations
  - Enable significant control over how I/O operations are handled, enabling better scaling.
  - Socket & file classes available.
  - 2 approaches to completion notification
    - java.util.concurrent.Future
    - CompletionHandler interface (competed() & failed() calls).
  - Flexible thread pooling strategies, including custom ones.

NIO.2 – New filesystem API

- Address long-standing usability issues & boilerplate
  - User-level modelling of more file system concepts like symlinks
  - File attributes modelled to represent FS-specific attributes (eg: owner, permissions...)
  - DirectoryStream iterates through directories
    - Scales very well, using less resources.
    - Allows glob, regex or custom filtering.
  - Recursive walks now provided, modelled on Visitor pattern.
- Model entirely artificial file systems much like Windows Explorer extensions
- File Change Notification
  - Improves performance of apps that currently poll to observe changes.

Directory Visitor - example

```
Files.walkFileTree(myPath, new SimpleFileVisitor<Path>() {
   public FileVisitResult visitFile(Path file,
                                    BasicFileAttributes attrs) {
        try {
            file.doWhatIWanted();
        } catch (IOException exc) {
            // do error handling
        }
        return FileVisitResult.CONTINUE;
    }
}
);
```

java.util.concurrent updates

- As multicore becomes more prevalent, data structures and algorithms to match are key.

- Major new abstraction: Fork/Join framework
  - Very good at 'divide and conquer' problems
  - Specific model for parallel computation acceleration, significantly more efficient than normal Thread or Executor -base synchronization models.
  - Implements work stealing for lopsided work breakdowns

- Other enhancements
  - TransferQueue – model producer/consumer queues efficiently
  - Phaser – very flexible synchronization barrier

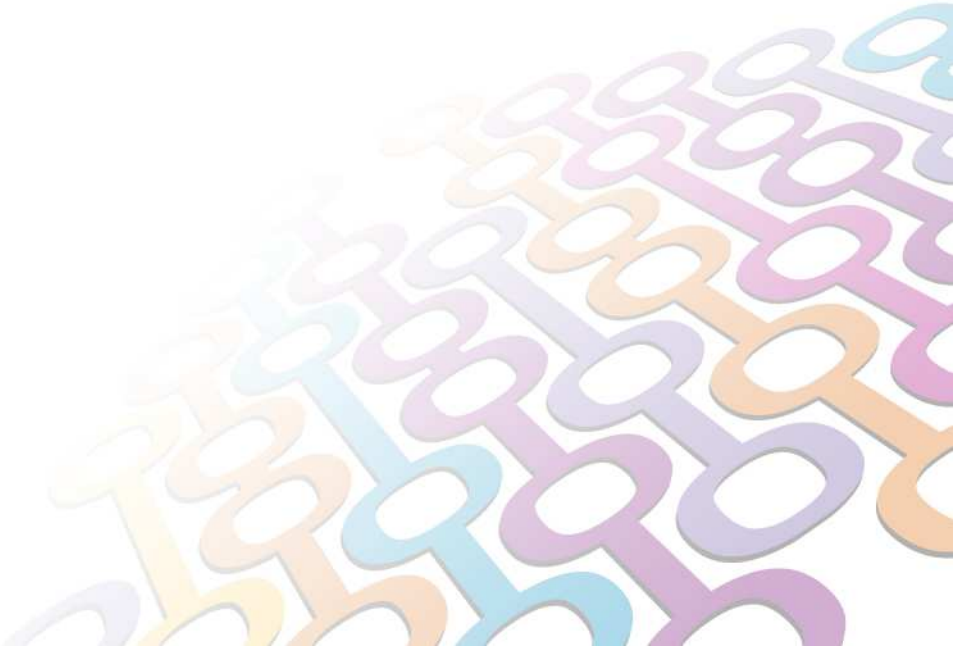
JSR 292 - invokedynamic

- The JVM managed runtime is becoming home to more languages (eg: jruby, jython, fan, clojure, etc..) but is missing some of the fundamentals that help make those languages go fast.
- JSR 292 decouples method lookup and method dispatch
  - Get away from being purely Java (the language) centric.
- Approach: introduce a new bytecode that executes a given method directly, and provides the ability at runtime to rewire what method that is.
  - Include a model for building up mutators (add a parameter, drop a parameter, etc..)
  - Ensure the JIT can efficiently exploit these constructs to ensure efficient code generation.

Smaller Items

- Classloader changes
  - Enable parallel classloading capability via new “safe” API.
  - URLClassLoader gains a close() method.
- I18N - Unicode 6.0, Locale enhancement, Separate user locale and user-interface locale
- TLS 1.2 – Security updates.
- JDBC 4.1 – ARM awareness.
- Client (UI) updates
  - Create new platform APIs for 6u10 graphics features
  - Nimbus look-and-feel for Swing
  - Swing JLayer component
  - XRender support
- Update the XML stack

IBM-Unique Updates and Improvements



Performance

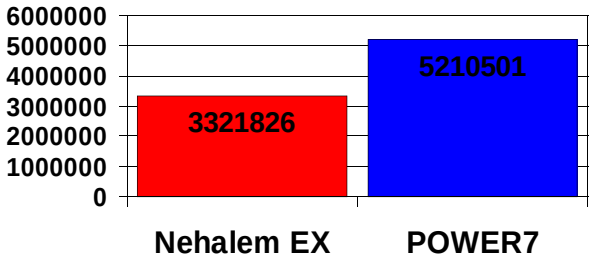
- "4 out of 5 publishes prefer J9"
  - http://www.spec.org/jbb2005/results/res2010q4/
  - 88% of SPECjbb2005 publishes in last year with J9
    - 94 with J9, 9 with HotSpot, 5 with Jrockit

SPECjbb2005

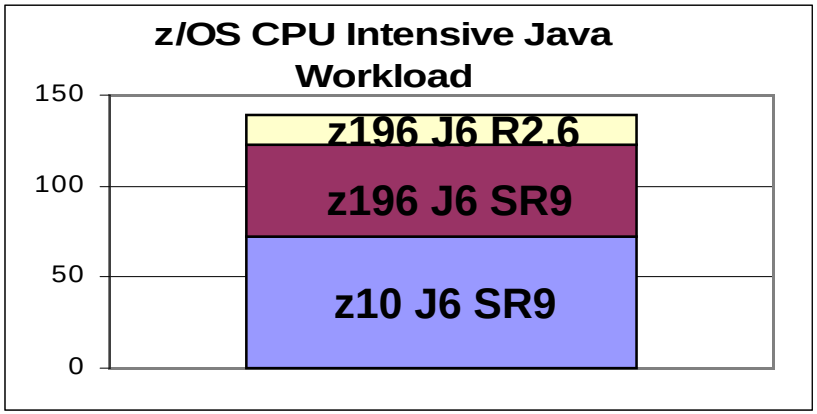
Company	BOPS	JVM	# cores	# chips	Published
Cisco Systems	1017141	IBM J9 VM (build	16	2	Oct-2010
Hewlett-Packard Company	1000188	IBM J9 VM (build	16	2	Dec-2010
IBM Corporation	318556	IBM J9 VM (build	4	1	Oct-2010
IBM Corporation	317811	IBM J9 VM (build	4	1	Oct-2010
SGI	3421167	Oracle Java HotS	64	8	Oct-2010

- POWER7 Exploitation
  - New prefetching capabilities
  - Extended divide instructions
  - Conversion between integer and float
  - Bit permutation and popcount instructions
  - BCD assist - Exploited through Java BigDecimal

P7 1.6X faster than Nehalem EX (8 sockets)



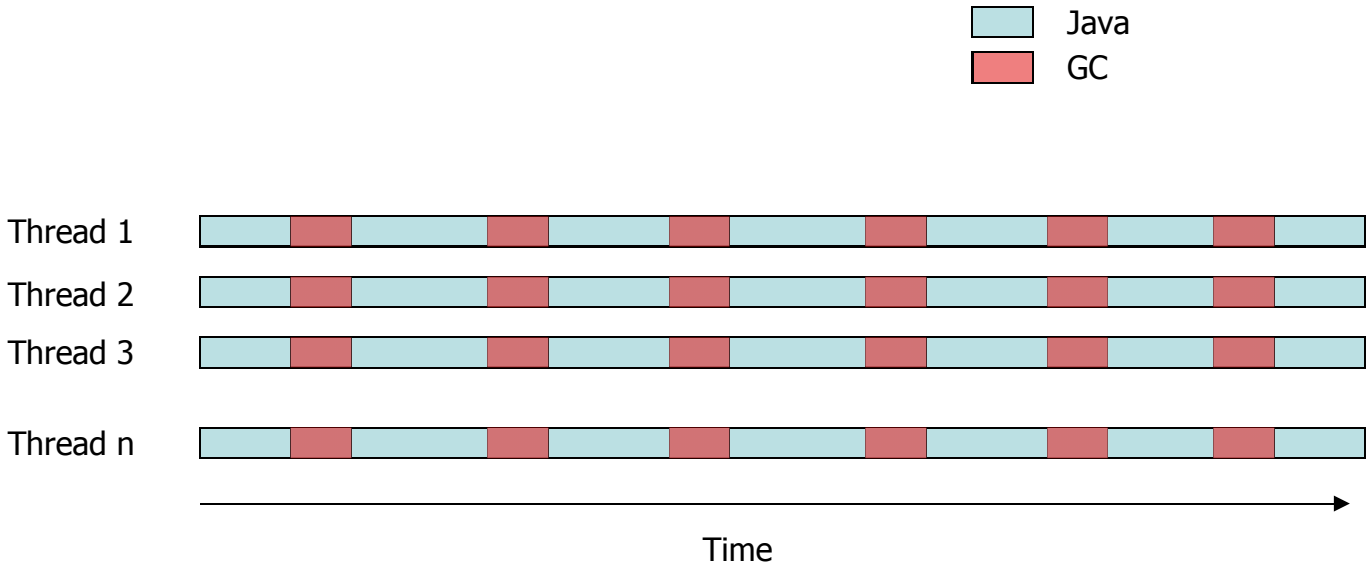
- System zEnterprise 196 Exploitation
  - 70+ new instructions
    - High-word facility
    - Interlock-update facility
    - Non-destructive operands
    - Conditional load/store
  - 93% Aggregate improvement
    - 14% Java 6.0.1 improvement
    - 70% Hardware improvement



GC policies since IBM Java 5

How do the policies compare?

-Xgcpolicy:optthruput *(and –Xgcpolicy:subpool)*

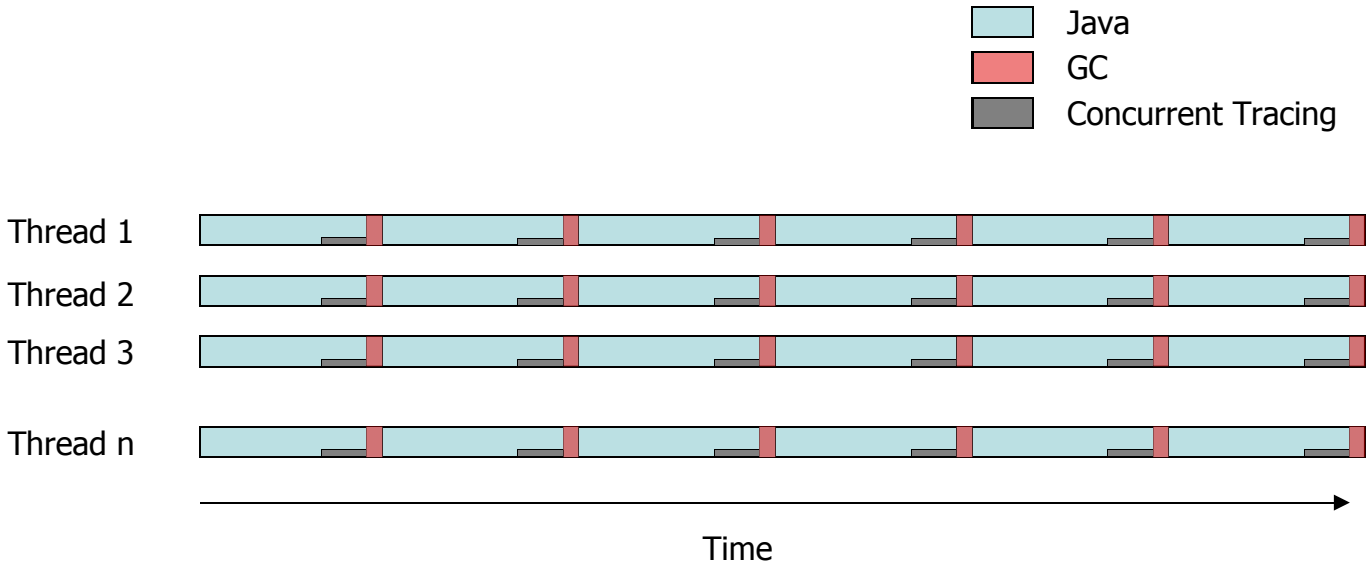


Picture is only illustrative and doesn't reflect any particular real-life application. The purpose is to show theoretical differences in pause times between GC policies.

GC policies since IBM Java 5

How do the policies compare?

-Xgcpolicy:optavgpause



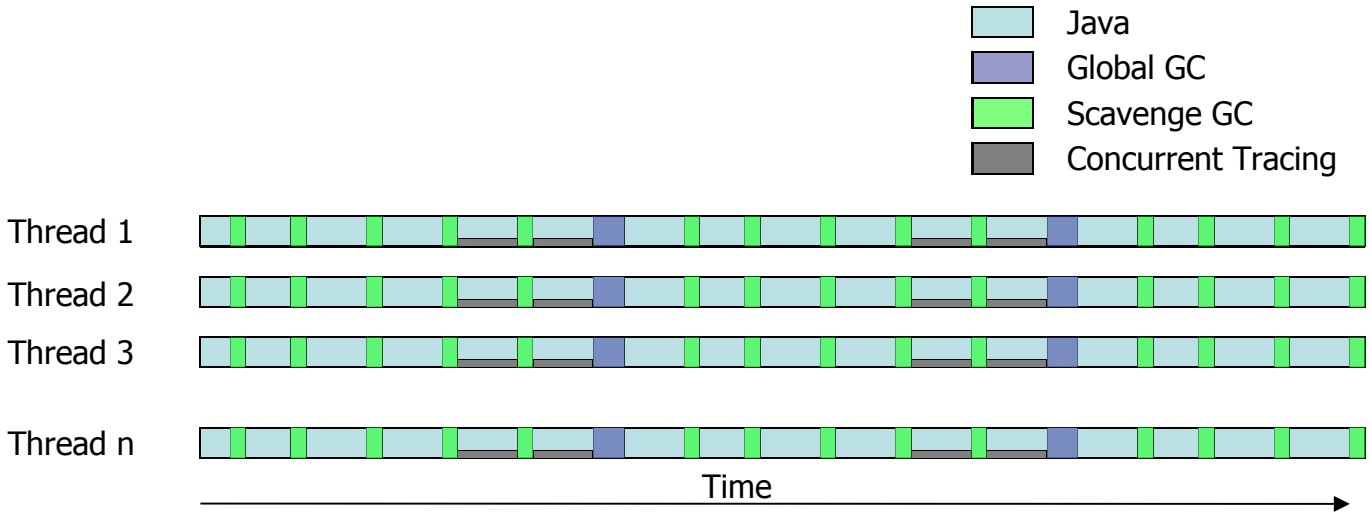
Picture is only illustrative and doesn't reflect any particular real-life application. The purpose is to show theoretical differences in pause times between GC policies.

GC policies since IBM Java 5

DEFAULT in Java 7

How do the policies compare?

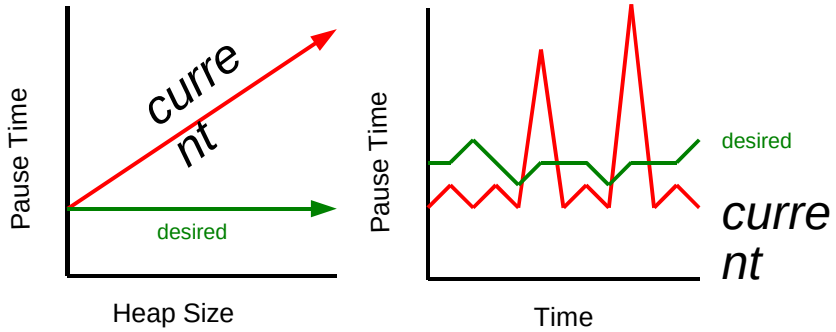
-Xgcpolicy:gencon



Picture is only illustrative and doesn't reflect any particular real-life application. The purpose is to show theoretical differences in pause times between GC policies.

Next-Gen Hardware and Software Challenges

- Meet customer needs for scaling garbage collection technology on large heaps



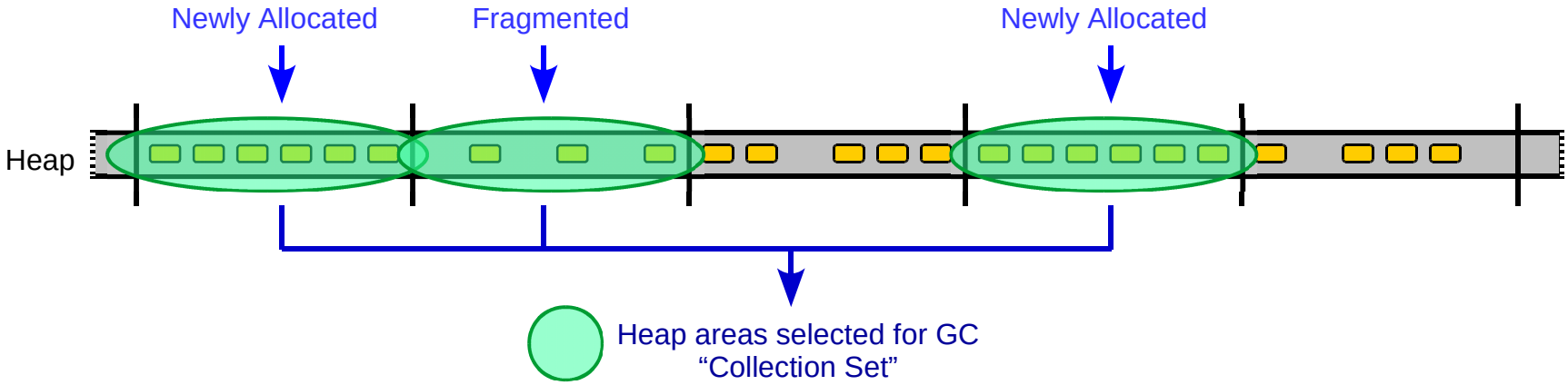
- Provide strong adaptive performance without expert advice
  - Flexible and adaptive behavior to provide a good first impression
  - Every tuning option increases complexity by an order of magnitude

- Showcase hardware capabilities through exploitation of platform facilities

- Maintain and increase technological competitive edge through innovation
  - Address new developments in industry quickly

-Xgcpolicy:balanced

- Incrementally collect areas of the heap that meet our needs – Partial Garbage Collect (PGC)
  - Reduced pause times
  - Freeing up memory
- Heap “collection set” selection based on best ROI (free memory) factors
  - E.g., Recently allocated objects, areas that reduce fragmentation



- Various technologies applied
  - Copy Forward (default)
    - High level of object mobility (similar to *gencon* GC policy)
  - Mark / Sweep / Compact
    - Separates the notion of “collection” vs. “compaction”

Specifics

- Suggested deployment scenario(s)
  - Larger (>4GB) heaps.
  - Frequent global garbage collections.
  - Excessive time spent in global compaction.
  - Relatively frequent allocation of large (>1MB) arrays.

- Fully supported on all IBM Java 7 64 bit platforms
  - First class citizen with other existing GC policies.

- We encourage use of the policy and welcome feedback!
  - The opportunity exists to work with the dev team.

Garbage Collection changes and improvements

- -Xgcpolicy:gencon is now the default
  - Provides a better out of the box performance for a most applications
  - Easily switch back to the old default with -Xgcpolicy:optthruput
- Object header size reduction & compressed references
  - Object headers are 4-16 bytes depending on object type and object reference size (32bit, 64bit compressed reference or 64bit)
  - “small 64 bit” (eg: 4-20GB) now have a 32-bit like footprint (new in a Java 6 SR).
  - Reduces garbage collection frequency
  - Provides better object locality
- Scalability improvements to all garbage collection policies on large n-way machines (#CPU > 64)
  - Decreases the time spent in garbage collections pauses
- Scalability improvements to allocation mechanism for highly parallel applications
  - -Xgcpolicy:subpool is now an alias for -Xgcpolicy:optthruput
- New format for verbose:gc output
  - Event based instead of summary based
  - Provides more detailed information for easier analysis

Try Out WebSphere Real Time (WRT)

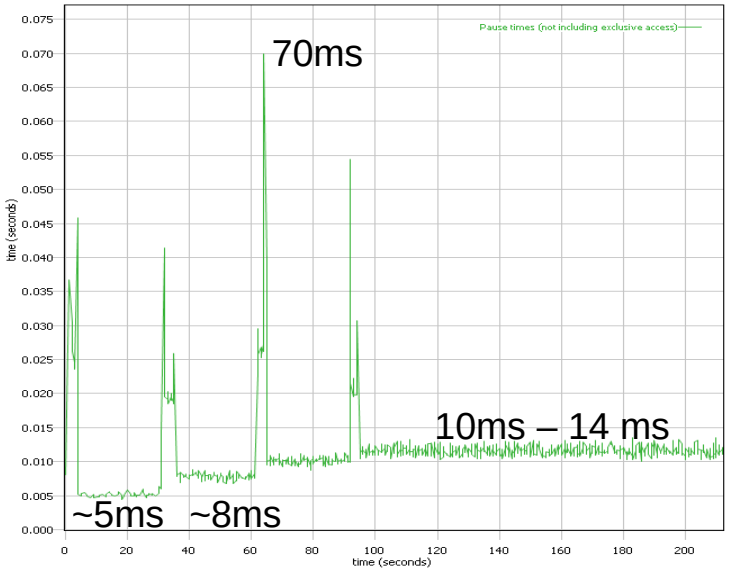
- WebSphere Real Time is a Java Runtime built with J9 technology that provides consistent performance
  - Incremental GC means consistently short (3ms) GC pause times
  - JIT compilations cannot block application threads
  - Also a Hard Real Time flavor that runs on Real-Time Linux (e.g. RHEL MRG, Novell SLERT)

- IBM Java 7 will include evaluation version of upcoming WRT-V3
  - New pause time target option lets you configure GC pause times
  - Throughput performance improvements
  - 32- and 64-bit Linux on x86, 32- and 64-bit AIX on POWER

- Just add -Xgcpolicy:metronome to your Java 7 command line to try it out!

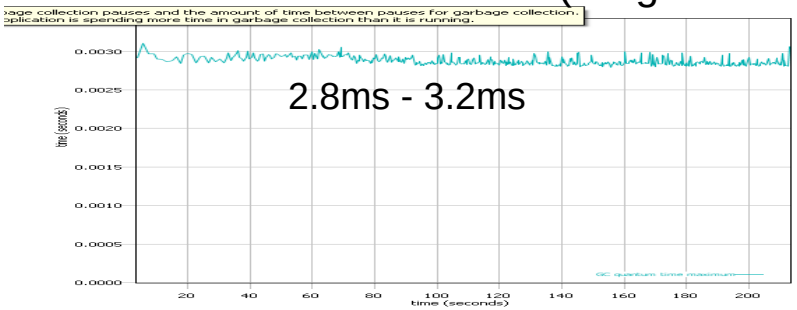
GC Pause Times: gencon and metronome

Gencon pause times

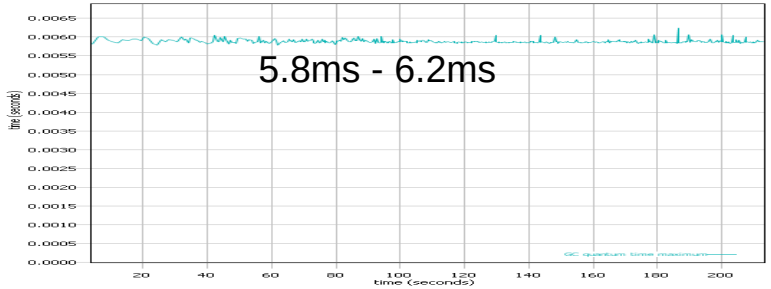


- Most GC policies have pause times ranging upwards of 10 – 100 ms
- Metronome controls pause times to as short as 3ms
- Throughput impact, varies by application

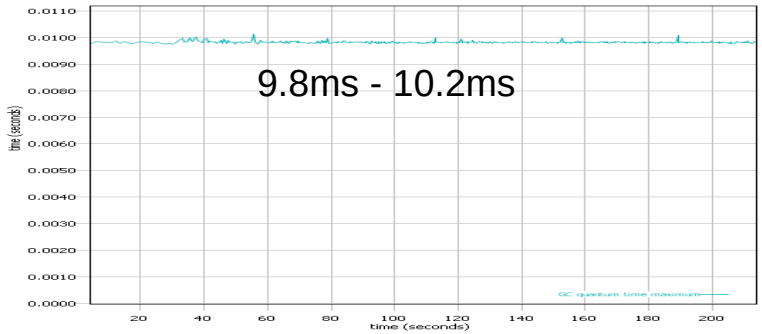
Metronome Pause Times (Target=3ms)



Metronome Pause Times (Target=6ms)



Metronome Pause Times (Target=10ms)



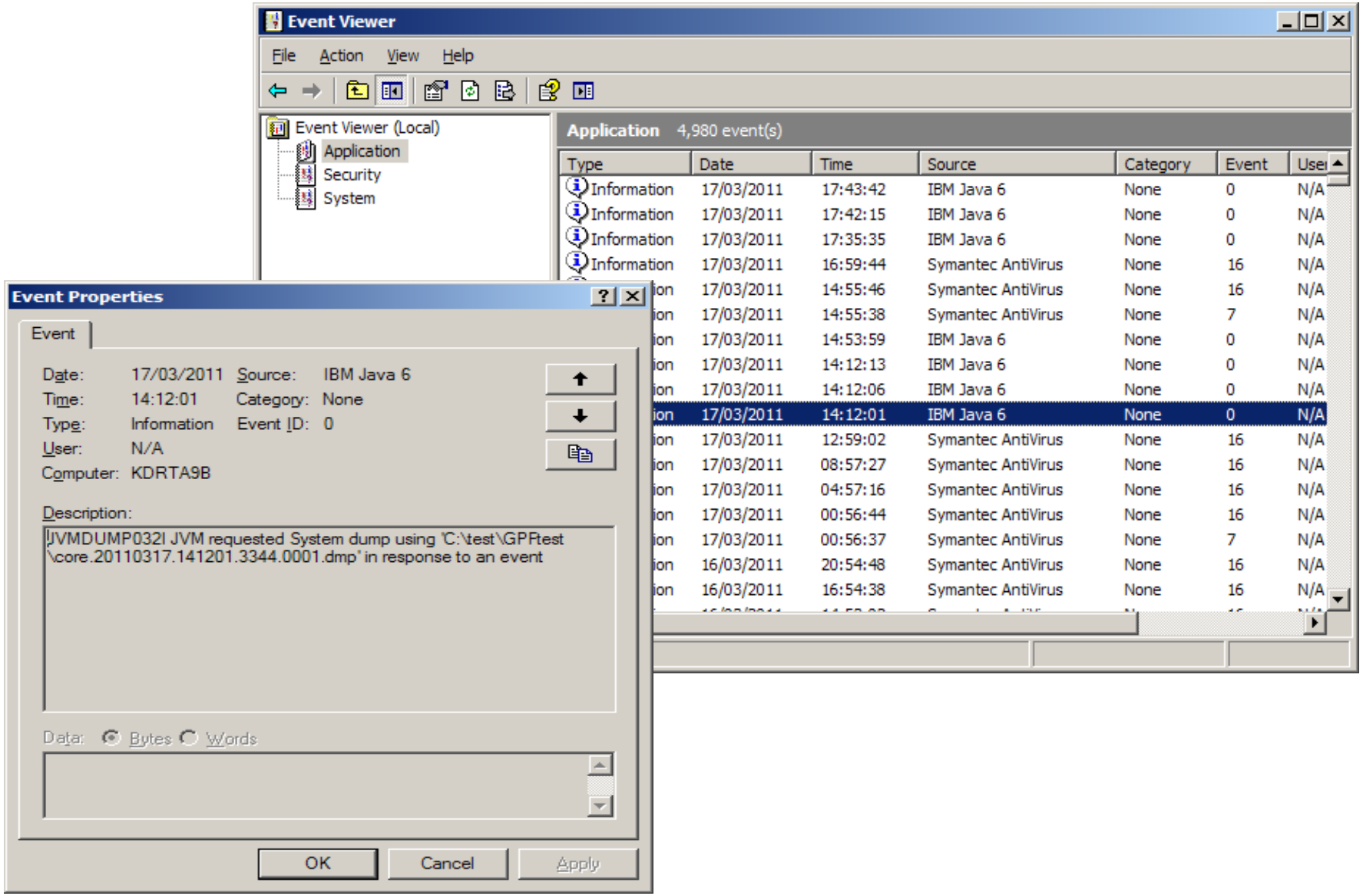
Consumability and RAS Enhancements

- -Xdump
  - Native stack traces in javacore
  - Environment variables and ULIMITs in javacore
  - Native memory usage counters in javacore and from core dumps via DTFJ
  - Multi-part TDUMPs on zOS 64
- -Xtrace
  - Tracepoints can include Java stacks (jstacktrace)
- -Xlog
  - Messages go to the Event log on Windows, syslog on Linux, errlog or syslog on AIX, MVS console on zOS.

Native memory usage counters

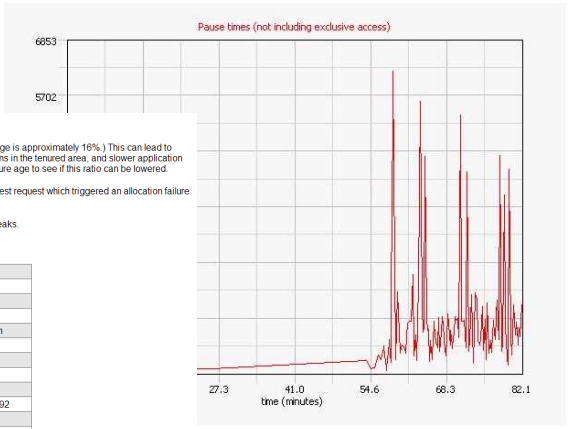
```
NATIVEMEMINFO subcomponent dump routine
=======================================

JRE: 555,698,264 bytes / 1208 allocations
|
+--VM: 552,977,664 bytes / 856 allocations
|   |
|   +--Classes: 1,949,664 bytes / 92 allocations
|   |
|   +--Memory Manager (GC): 547,705,848 bytes / 146 allocations
|   |   |
|   |   +--Java Heap: 536,875,008 bytes / 1 allocation
|   |   |
|   |   +--Other: 10,830,840 bytes / 145 allocations
|   |
|   +--Threads: 2,660,804 bytes / 104 allocations
|   |   |
|   |   +--Java Stack: 64,944 bytes / 9 allocations
|   |   |
|   |   +--Native Stack: 2,523,136 bytes / 11 allocations
|   |   |
|   |   +--Other: 72,724 bytes / 84 allocations
|   |
|   +--Trace: 92,464 bytes / 208 allocations
|   |
```

Example JVM message in Windows Event log



Garbage Collection and Memory Visualizer (GCMV)

- Tool to analyze Java verbose GC logs
- Graphs Recommendations use heuristics to guide you towards issues that may be limiting performance.
- Show garbage collection and Java heap statistics over time.
- Not only for memory errors, very good for performance tuning.



Tuning recommendation

A high proportion of the nursery is tenured each collection. (The average is approximately 16% .) This can lead to longer pause times for collections in the nursery, more frequent collections in the tenured area, and slower application access to these objects. Consider increasing the nursery size or the tenure age to see if this ratio can be lowered.

The application seems to be using some quite large objects. The largest request which triggered an allocation failure (and was recorded in the verbose gc log) was for 3643992 bytes.

The memory usage of the application does not indicate any obvious leaks.

Summary

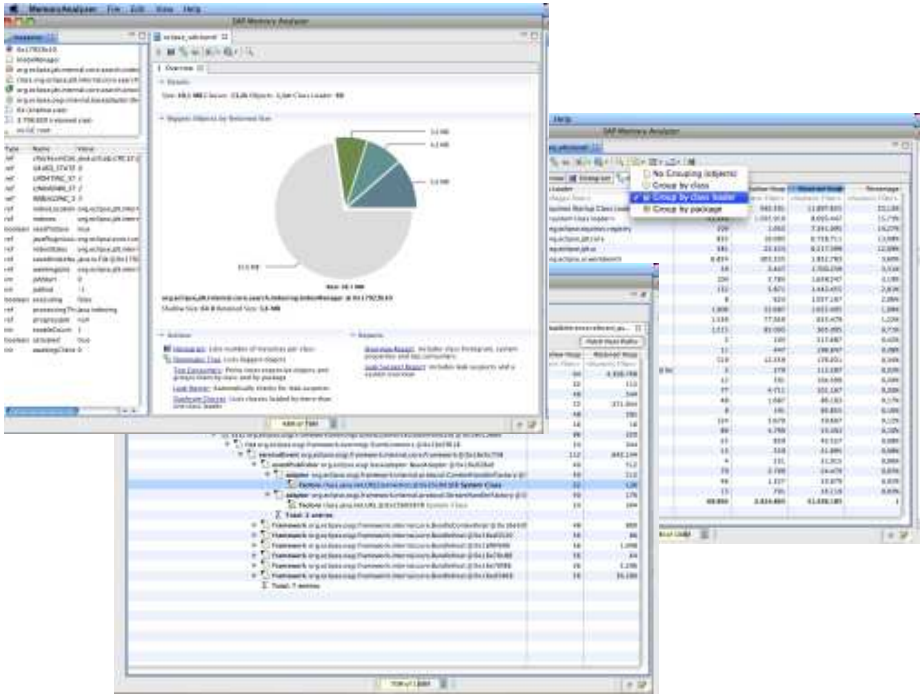
Allocation failure count	1286
Concurrent collection count	1
Forced collection count	0
Full collections	0
GC Mode	gencon
Global collections - Mean garbage collection pause (ms)	693
Global collections - Mean interval between collections (minutes)	0.18
Global collections - Number of collections	21
Global collections - Total amount tenured (MB)	10290
Largest memory request (bytes)	3643992
Minor collections - Mean garbage collection pause (ms)	127
Minor collections - Mean interval between collections (ms)	56.1
Minor collections - Number of collections	1266
Minor collections - Total amount flipped (MB)	11756
Minor collections - Total amount tenured (MB)	9168
Proportion of time spent in garbage collection pauses (%)	75.8
Proportion of time spent unpaused (%)	24.2
Rate of garbage collection	0.163 MB/sec

Diagnostics Collector

- At JVM start it runs a diagnostic configuration check
- Runs as a separate process when the JVM detects a ‘dump event’
  - GPF
  - Java heap OutOfMemoryError
  - Unexpected signal received
  - (optionally) JVM start, JVM stop
- Knows all possible dump locations and searches to gather all dumps into a single zip file
- Collects system dumps, Java dumps, heap dumps, verbose GC logs
- If system dump found jextract runs automatically
- Requires IBM SDK for Java version 5.0 or above

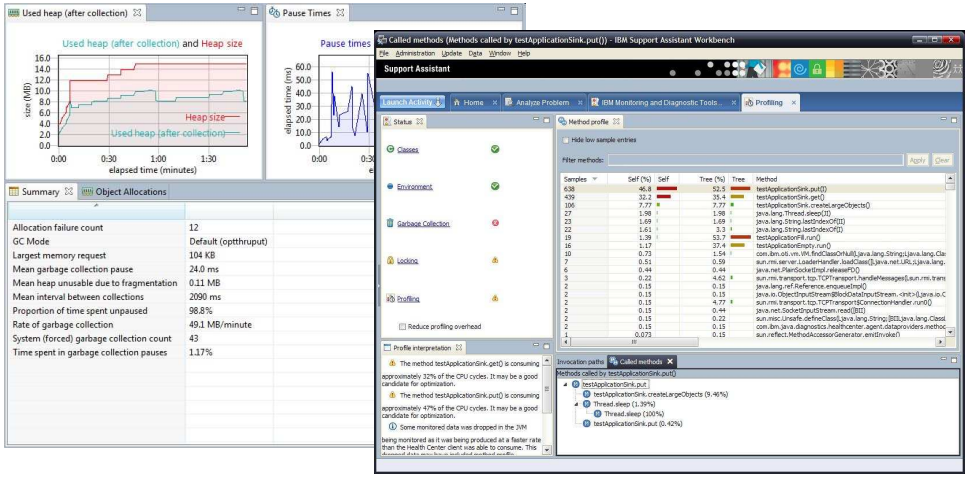
Memory Analyzer

- Eclipse project for analyzing heap dumps and identifying memory leaks from JVMs
- Works with IBM system dumps, heapdumps and Sun HPROF binary dumps
- Provides memory leak detection and footprint analysis
  - Objects by Class, Dominator Tree Analysis, Path to GC Roots, Dominator Tree by Class Loader
- Provides SQL like object query language (OQL)
- Provides extension points to write analysis plugins



Health Center

- Live monitoring tool with very low overhead
- Understand how your application is behaving
- It provides access to information about method profiling, garbage collection, class loading, locking and environment data
- Diagnose potential problems, with recommendations
- Works at the JVM level – no domain-specific (e.g. J2EE) information
- Suitable for all Java applications



Summary & Conclusion

- Base Java 7 Features
  - NIO.2, java.util.concurrent, etc...
- IBM feature details
  - Performance & Platform Exploitation
  - Garbage Collection Updates & “Balanced” GC policy
  - Serviceability improvements
- Free Tools Overview

Copyright and Trademarks