Java Garbage Collector - Overview and Tuning

Iris Baron
IBM Java JIT on System Z
ibaron@ca.ibm.com

Session ID: 16181

#SHAREorg

SHARE in Pittsburgh 2014

Java Road Map

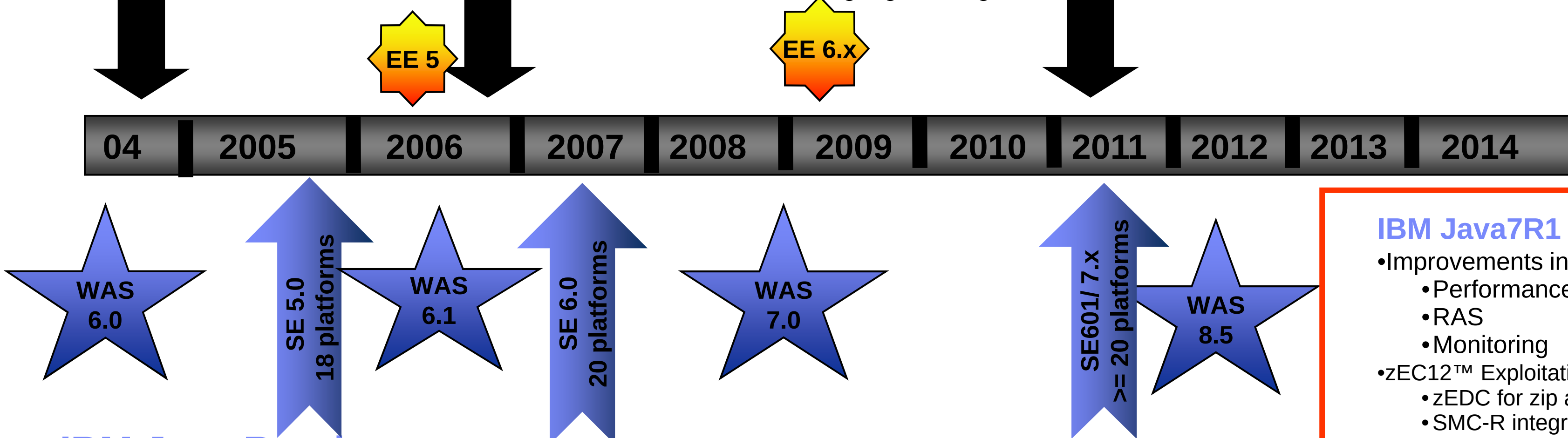
Language Updates

Java 5.0
- New Language features:
  - Autoboxing
  - Enumerated types
  - Generics
  - Metadata

Java 6.0
- Performance Improvements
- Client WebServices Support

Java 7.0
- Support for dynamic languages
- Improve ease of use for SWING
- New IO APIs (NIO2)
- Java persistence API
- JMX 2.x and WS connection for JMX agents
- Language Changes

Java 8.0**
- Language improvements
- Closures for simplified fork/join



IBM Java Runtimes

IBM Java 5.0 (J9 R23)
- Improved performance
  - Generational Garbage Collector
  - Shared classes support
  - New J9 Virtual Machine
  - New Testarossa JIT technology
- First Failure Data Capture
- Full Speed Debug
- Hot Code Replace
- Common runtime technology
  - ME, SE, EE

IBM Java 6.0 (J9 R24)
- Improvements in
  - Performance
  - Serviceability tooling
  - Class Sharing
- XML parser improvements
- z10™ Exploitation
  - DFP exploitation for BigDecimal
  - Large Pages
  - New ISA features

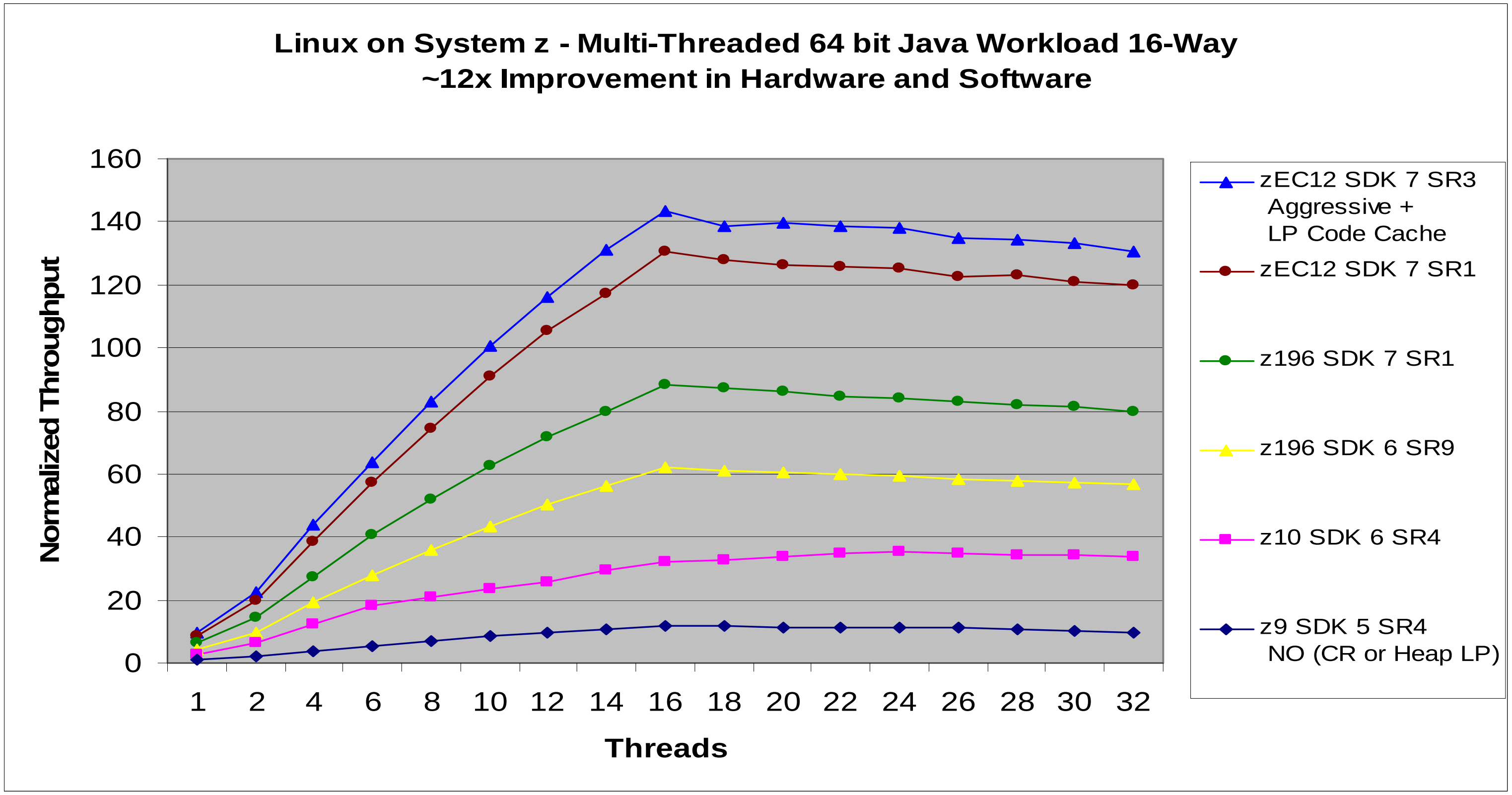
IBM Java 6.0.1/Java7.0 (J9 R26)
- Improvements in
  - Performance
  - GC Technology
- z196™ Exploitation
  - OOO Pipeline
  - 70+ New Instructions
- JZOS/Security Enhancements

IBM Java7R1
- Improvements in
  - Performance
  - RAS
  - Monitoring
- zEC12™ Exploitation
  - zEDC for zip acceleration
  - SMC-R integration
  - Transactional Execution
  - Runtime instrumentation
- Hints/traps
- Data Access Accelerator

IBM Java7.0SR3
- Improvements in
  - Performance
- zEC12™ Exploitation
  - Transactional Execution
  - Flash 1Meg pageable LPs
  - 2G large pages
  - Hints/traps

**Timelines and deliveries are subject to change.

Linux on System z and Java7SR3 on zEC12



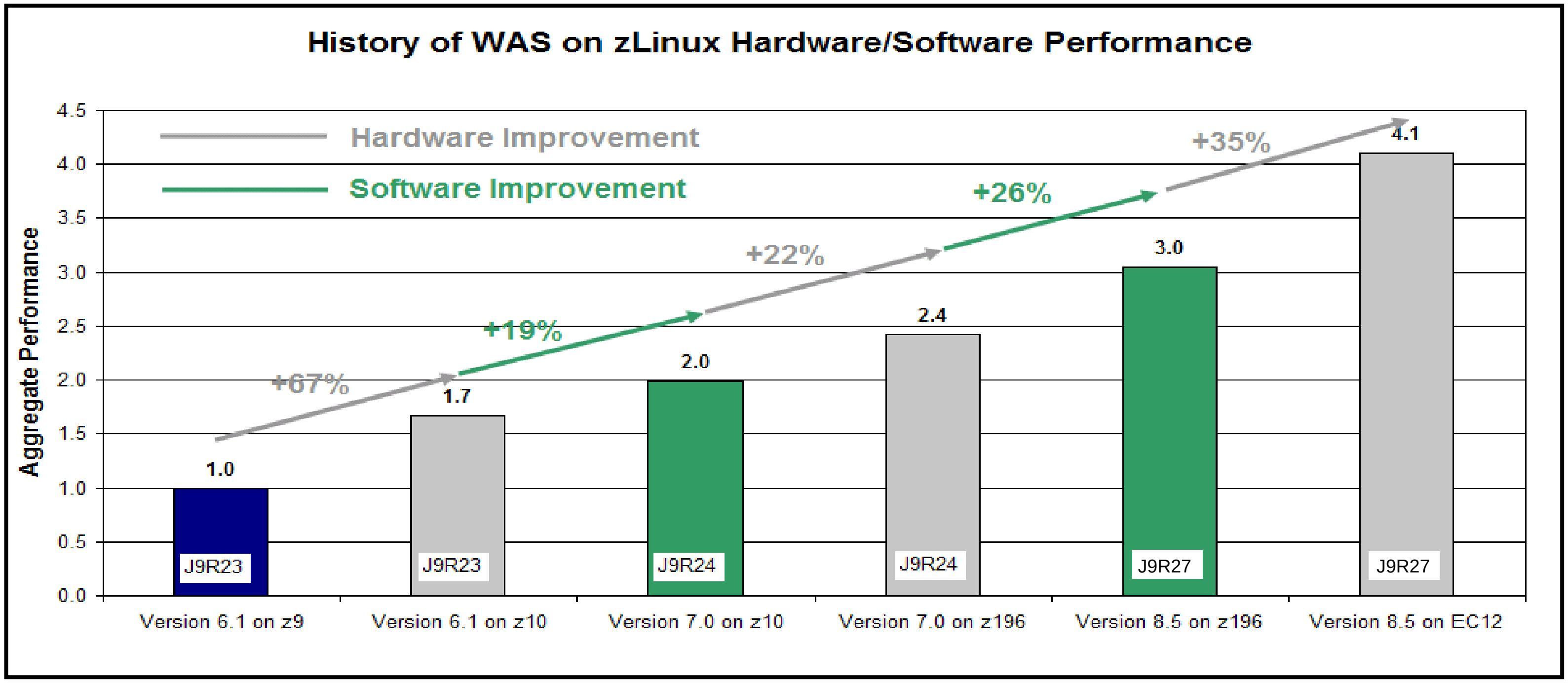
(Controlled measurement environment, results may vary)

~12x aggregate hardware and software improvement comparing Java5SR4 on z9 to Java7SR3 on zEC12

- LP=Large Pages for Java heap
- CR=Java compressed references
- Java7SR3 using -Xaggressive + 1Meg large pages

WAS on zLinux

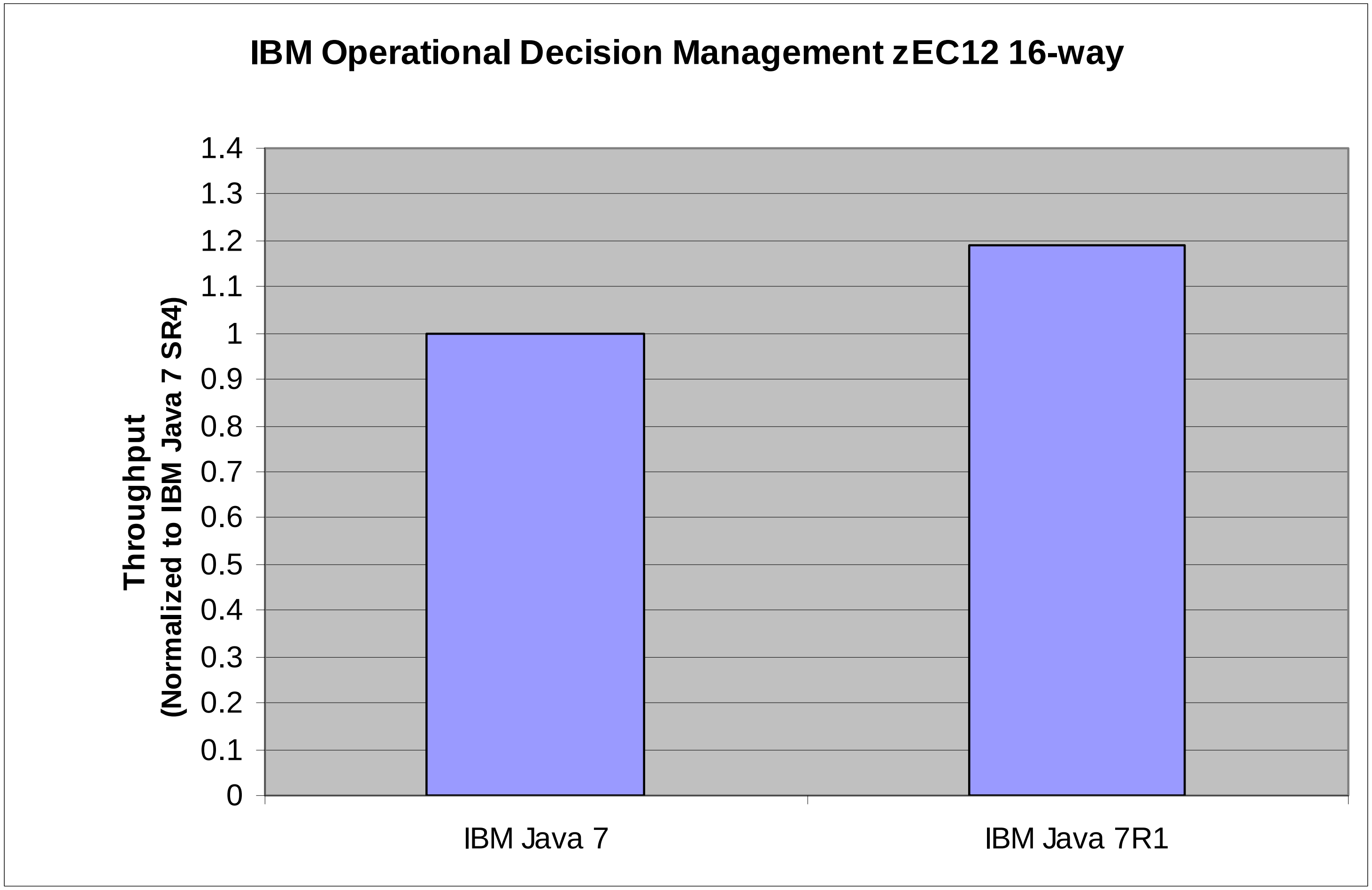
Aggregate HW, SDK and WAS Improvement: WAS 6.1 (Java 5) on z9 to WAS 8.5 (Java 7) on zEC12



(Controlled measurement environment, results may vary)

4x aggregate hardware and software improvement comparing WAS 6.1 Java5 on z9 to WAS 8.5 Java7 on zEC12

IBM Operational Decision Manager



(Controlled measurement environment, results may vary)

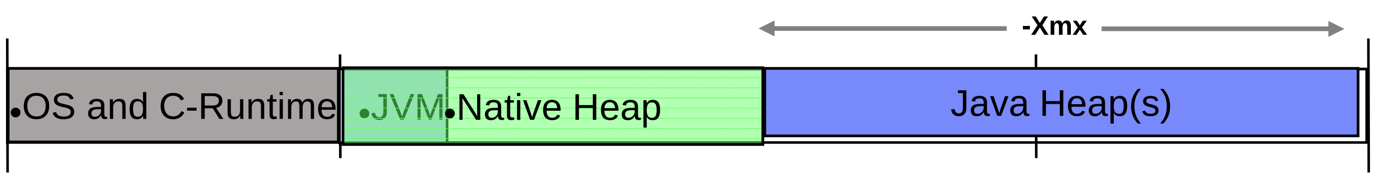
19% improvement to ODM with IBM Java 7R1 compared to IBM Java 7

Garbage Collection: Agenda

- **GC Overview**
- IBM JVM(J9) GC - Policies
- IBM JVM(J9) GC – Choosing the right policy
- IBM Monitoring and Diagnostic Tools
- Summary

Garbage Collector: Overview
Java Memory Usage

- Java is an Operating System (OS) process
  Some memory is used by the OS and C-language runtime
- Area left over is termed the “User Space” and is divided into:
  - Java Virtual Machine (JVM) runtime
  - Java Heap(s)
  - Native heap



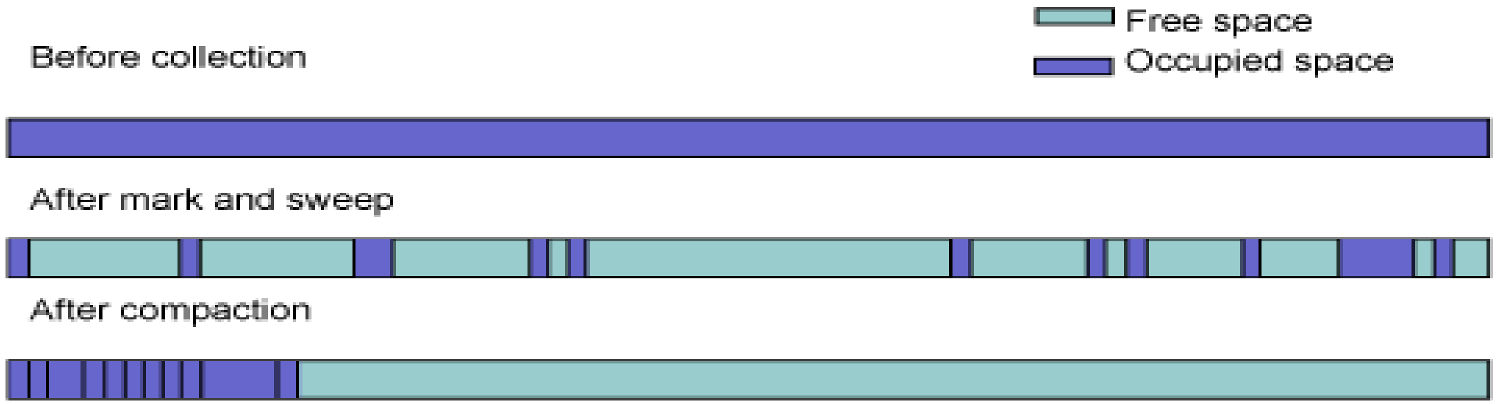
Garbage Collector: Overview

- The GC works on the Java Heap memory
- Responsible for allocation and freeing of:
  Java objects, arrays and classes
- Allocates objects using a contiguous section of Java heap
- Ensures the object remains as long as it is in use (“live”)
- Determines “liveness” based on a reference from another “live” object or from a known root
- Reclaims objects that are no longer referenced
- Ensures that any finalize method is run before the object is reclaimed

Garbage Collector: Overview

Two main GC technologies:

- Mark Sweep Collector
  - Mark: Find all live objects
  - Sweep: Reclaim unused heap memory
  - Compact: Reduce fragmentation (optional)



- Copy Collector

Garbage Collector: Overview

GC occurs under two scenarios:

- An *allocation failure*:

  Not enough contiguous memory available

- A programmatically requested GC cycle:

  Call to System.GC()

Garbage Collector: Overview
Performance Effect

- GC affects application's performance:
  - Pause times (responsiveness/Consistency)
  - Throughput
  - Footprint
- Typical tradeoffs:
  - Pause time vs. Throughput
    - Tradeoff frequency and length of pauses vs. throughput
  - Footprint vs. Frequency
    - Tradeoff smaller footprint vs. frequency of GC pauses/events

Garbage Collector: Overview
IBM J9 Garbage Collector Family

Why have many policies? Why not just *the best*?

- Cannot always dynamically determine what tradeoffs the user/application are willing to make
- Definition of a performance problem is user centric

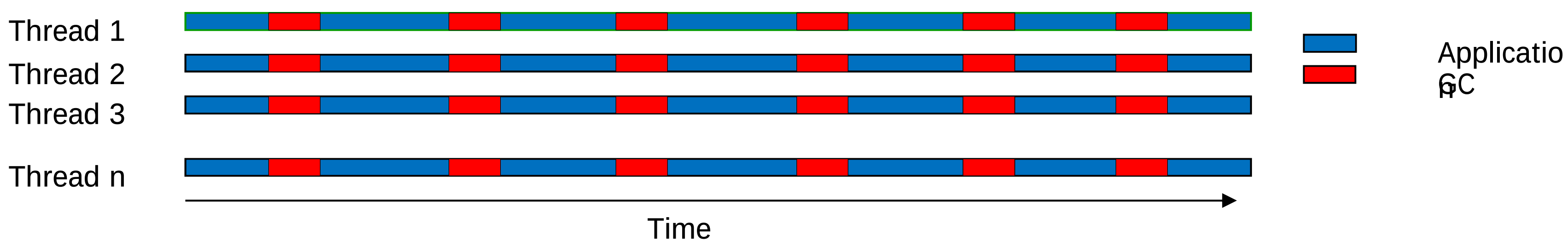
Policy	Notes
optthruput	default in Java5 and Java6
optavgpause	
gencon	default in Java601/Java7
balanced	added in Java601/Java7

Garbage Collection: Agenda

- GC Overview
- **IBM JVM(J9) GC - Policies**
  - optthruput
  - optavgpause
  - gencon
  - balanced
- IBM JVM(J9) GC – Choosing the right policy
- IBM Monitoring and Diagnostic Tools
- Summary

IBM J9 Garbage Collector: optthruput

- **Global Mark and Sweep Garbage Collection**
- Uses “flat” heap
- Single stop-the-world (STW) phase
  - Application “pauses” while GC is done
- Parallel GC via use of “GC Helper Threads”
  - “Parked” set of threads that wake to share GC work
  - Configurable using -Xgcthreads



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.

IBM J9 Garbage Collector: optthruput Tuning: Fixed vs. Variable Heap Size

Main tuning settings:
-Xms (initial heap size)
-Xmx (maximum heap size)

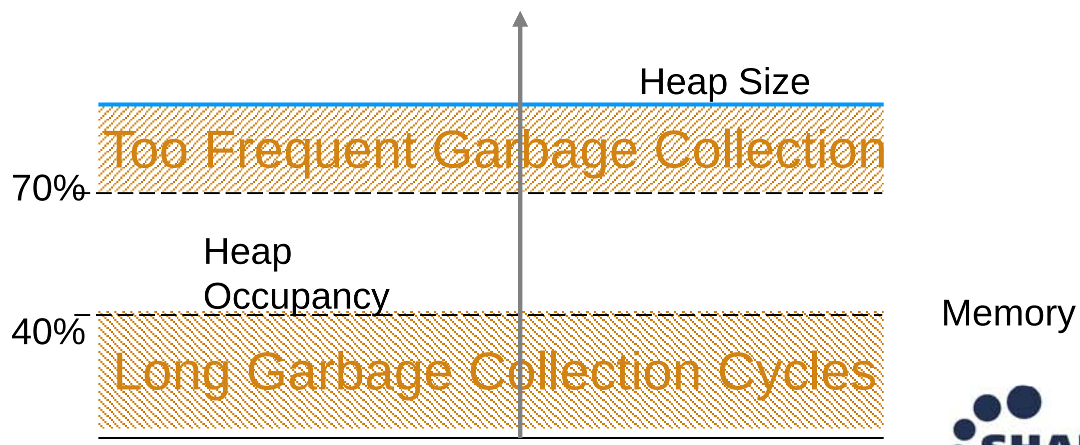
- Fixed heap size
  - Same values for -Xms and -Xmx
  - Best when memory usage is fairly constant and known

- Variable heap size
  - Different values for -Xms and -Xmx
  - GC adapts heap size
  - Best when memory usage varies over time or unknown
  - Provides flexibility and avoids OutOfMemoryErrors
  - Allows fine tuning of the heap usage

IBM J9 Garbage Collector: optthruput
Implementation: heap Expansion and Contraction

GC adapts heap size to keep occupancy between 40% and 70% using expansion and contraction:

- Occupancy over 70% causes frequent GC cycles
  - Which generally means reduced performance
  - Requires Expansion
- Occupancy below 40% means infrequent but longer GC cycles
  - Amount of live object might not change, but may have secondary effects
  - Requires more memory
  - Requires Contraction



IBM J9 Garbage Collector: optthruput Implementation: Compaction

- Heap expansion and contraction are relatively “cheap”
- GC (usually) optimizes heap occupancy at the cost of compaction cycles (“expensive”):
  - Expansion: for some expansions, GC may have already compacted to try to allocate the object before expansion
  - Contraction: GC may need to compact to move objects from the area of the heap being “shrunk”

IBM J9 Garbage Collector: optthruput
Recommended Deployment Scenarios

Best when:

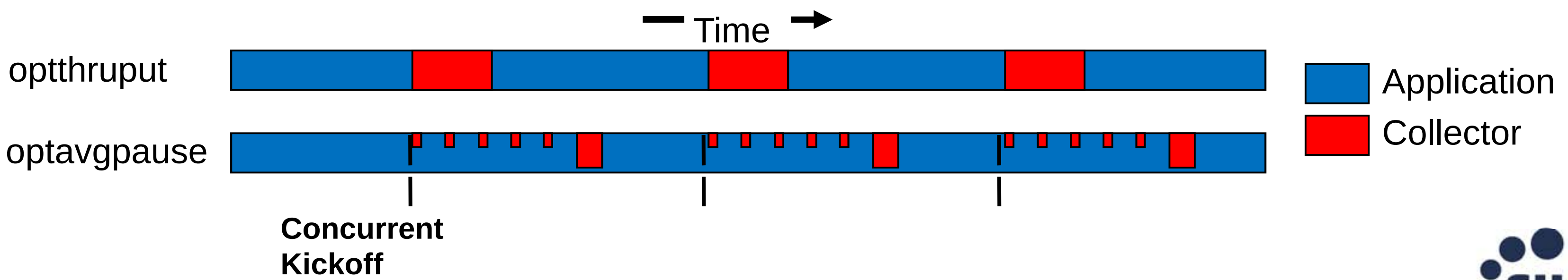
- Application is optimized for **throughput** rather than short GC pauses
- Pause time problems are not evident
- “Batch” type applications

Was default in Java5 and Java6, but not in Java7

IBM J9 Garbage Collector: optavgpause

- Reduces pause time spent inside STW GC vs optthruput
- Results in more consistent pauses
- Slight overhead on application's throughput performance
- Uses **Concurrent** Marking and Sweeping

  Carrying out some of the STW work while application is running
- Policy focuses on responsiveness criteria



IBM J9 Garbage Collector: gencon

Generational and Concurrent GC

- Best of both worlds: Throughput and Small Pause Times
- Shown most value with customers
- Default policy in Java6.0.1/Java7
- Handles short- and long-lived objects differently:
  Heap is split into two areas:
  - Objects created in the **nursery**
  - Objects that survive a number of collections are promoted to **tenured** area

IBM J9 Garbage Collector: gencon Motivation

- Most objects die young - focus on recently created objects

- Example: String concatenation

```
String str = new String ("String  ");
str += "Concatenated!";
```

  Results in the creation of 3 objects:
  - String object, containing "String "
  - A StringBuffer, containing "String ", and with "Concatenated!" then appended
  - String object, containing the result: "String Concatenated!"

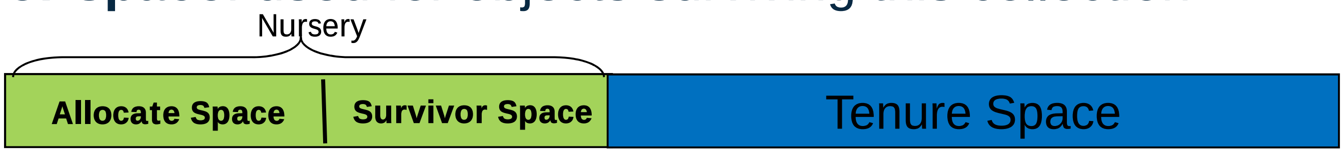
- Two of those three objects are no longer required!

- Other examples: transactions in banking /commerce, DB, web page request, GUI functions

IBM J9 Garbage Collector: gencon
Nursery Space Implementation

Nursery uses Copying GC :

- Nursery is split into two spaces:
  - **Allocate space**: used for new allocations and objects that survived previous collections
  - **Survivor space**: used for objects surviving this collection

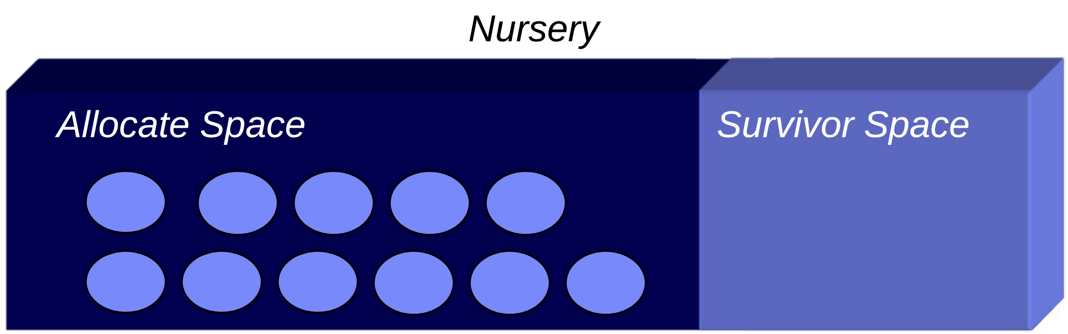


- Collection causes live objects to be:
  - copied from Allocate space to survivor space
  - copied to the Tenured space if they have survived sufficient collections

✓Small but frequently collected area
✓Reduces fragmentation
✓Improves data locality
✓Speeds up future allocations

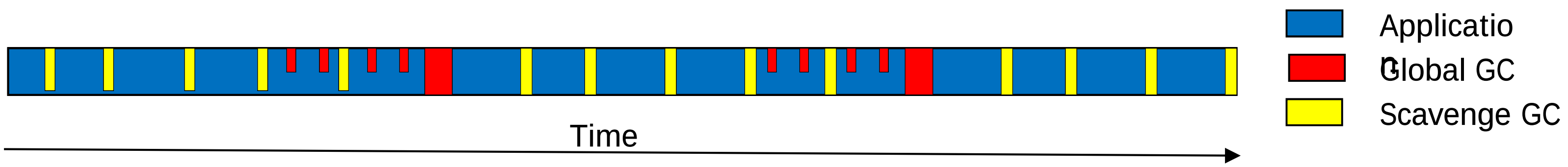
IBM J9 Garbage Collector: gencon
Nursery Space Implementation

- A 50/50 split between Allocate and Survivor spaces is wasteful
- Survivor space is “unusable” heap
  - Survivor space can be smaller than Allocate space
- “Tilt Ratio” - ratio between nursery spaces
  - Tilt adjusts automatically between 50% and 90%



IBM J9 Garbage Collector: gencon Tenure Space Implementation

- Tenure space uses Concurrent Mark and Sweep GC (same as optavgpause)
- Less frequently collected
- Same tuning options as for optthruput/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.

IBM J9 Garbage Collector: gencon Tuning: Nursery Space

- Copying data is a time consuming task
- Nursery collection time is proportional to amount of data copied

▶ Ideally Nursery size should be as large as possible!

The Larger the Nursery size

- the longer the time between collects
- the lower the GC overhead
- the fewer % of objects that survive
- very large objects are unlikely to be allocated directly into the tenured space

Disadvantages of very large nursery spaces:

- lots of physical memory is required

IBM J9 Garbage Collector: gencon Recommended Deployment Scenarios

Best when:

- Application allocates many short-lived objects
- Application is transaction-based
- The heap space is fragmented

- There is a net increase in memory usage when migrating to gencon

IBM J9 Garbage Collector: balanced

Goal: Improve responsiveness

- Reduced max pause times to achieve more consistent behaviour
- Incremental collection targets best ROI areas of the heap

Expands platform exploitation possibilities

- Virtualization : group heap data by frequency of access, direct OS paging decisions
- Dynamic reorganization of data structures to improve memory hierarchy utilization (performance)

IBM J9 Garbage Collector: balanced
Recommended Deployment Scenarios

Best when:

- Large (>4GB) heaps – to reduce long GC pause
- Frequent global garbage collections
- Excessive time spent in global compaction

Garbage Collection: Agenda

- GC Overview
- IBM JVM(J9) GC - Policies
- **IBM JVM(J9) GC – Choosing the right policy**
- IBM Monitoring and Diagnostic Tools
- Summary

IBM J9 Garbage Collector
Choosing the right policy

- Typical configuration
  - Pick a policy based on desired application behaviour
  - Tune heap sizes (use tooling)
  - Helper threads (-Xgcthreads)
  - Lots of other tuning knobs, suggest try hard to ignore, to avoid over-tuning
  - Monitor and re-tune if needed

- Rule of thumb:
  - If GC overhead is > 10%, you've most likely chosen the wrong policy

- Best practice:
  - Don't use System.gc()
  - Avoid finalizers
  - Memory leaks are possible even with a garbage collector

IBM J9 Garbage Collector
Choosing the right policy

Policy	Recommended usage	Notes
optthruput	throughput (batch applications)	default in Java5 and Java6
optavgpause	reduced pause times	
gencon	transactional workloads	default in Java601/Java7
balanced	optimized for large heaps	added in Java601/Java7

Garbage Collection: Agenda

- GC Overview
- IBM JVM(J9) GC - Policies
- IBM JVM(J9) GC – Choosing the right policy
- **IBM Monitoring and Diagnostic Tools**
  - Health Center
  - GCMV
  - Memory Analyzer

Session #16182:
Java Monitoring and Diagnostic Tooling
Thursday, Aug 7th, 4:15PM, room 304

- Summary

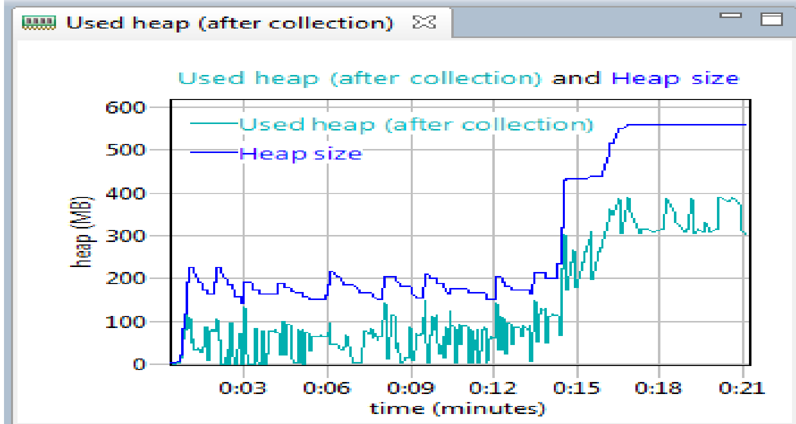
Garbage Collection
IBM Monitoring and Diagnostic Tools for Java

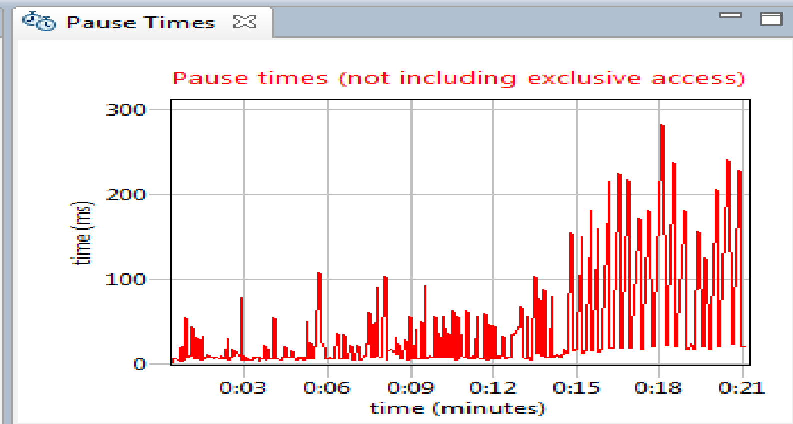
Health Center

- Motivating questions:
  - What is my JVM doing? Is everything ok?
  - Why is my application running slowly?
  - Why is it not scaling?
  - Am I using the right options?
- Overview
  - Lightweight live monitoring tool with very low overhead
  - Understand how your application is behaving, diagnose potential problems with recommendations
  - Visualize garbage collection, method profiling, class loading, lock analysis, file I/O and native memory usage
  - Suitable for all Java applications running on IBM's JVM

Garbage Collection
IBM Monitoring and Diagnostic Tools for Java

Health Center





Summary

Allocation failure count	148
GC Mode	Default (optthruput)
Largest memory request	39.1 KB
Mean garbage collection pause	30.0 ms
Mean heap unusable due to fragmentation	26.3 MB
Mean interval between collections	0.0013 minutes
Proportion of time spent unpaused	62.6%
Rate of garbage collection	76044 MB/minutes
System (forced) garbage collection count	111
Time spent in garbage collection pauses	37.4%

Garbage Collection
IBM Monitoring and Diagnostic Tools for Java

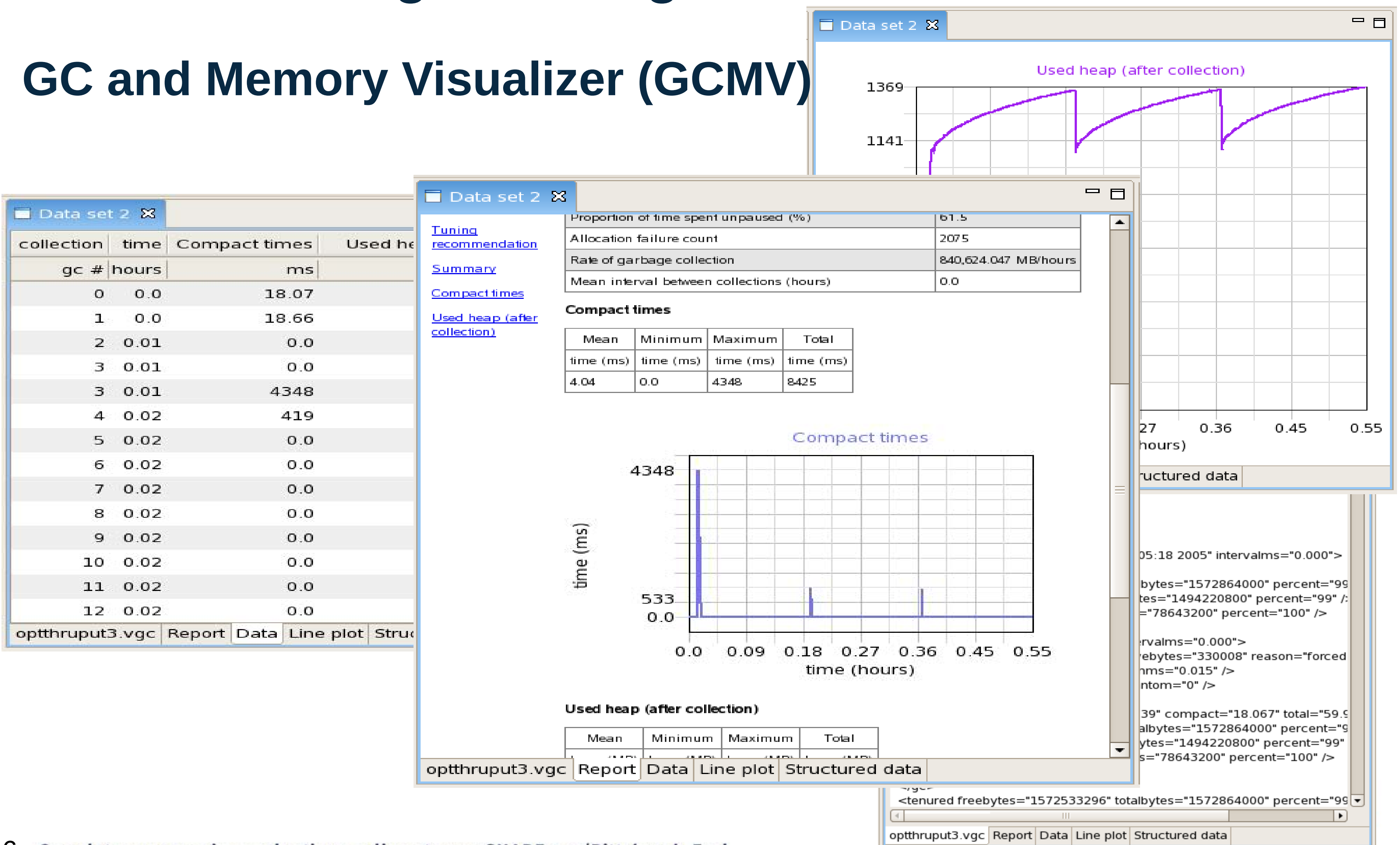
GC and Memory Visualizer (GCMV)

- Motivating questions:
  - How is the GC behaving? Can I do better?
  - How much time is GC taking?
  - How much free memory does my JVM have?

- Overview
  - Analyzes Java verbose GC logs, providing insight into application behaviour
  - Uses ps -p $PID -o pid,vsz,rss output to plot native footprint
  - Visualizes a wide range of GC data and Java heap statistics over time
  - Provides the ability to detect memory leaks and optimize GC
  - Recommends tuning options to improve performance

Garbage Collection
IBM Monitoring and Diagnostic Tools for Java

GC and Memory Visualizer (GCMV)

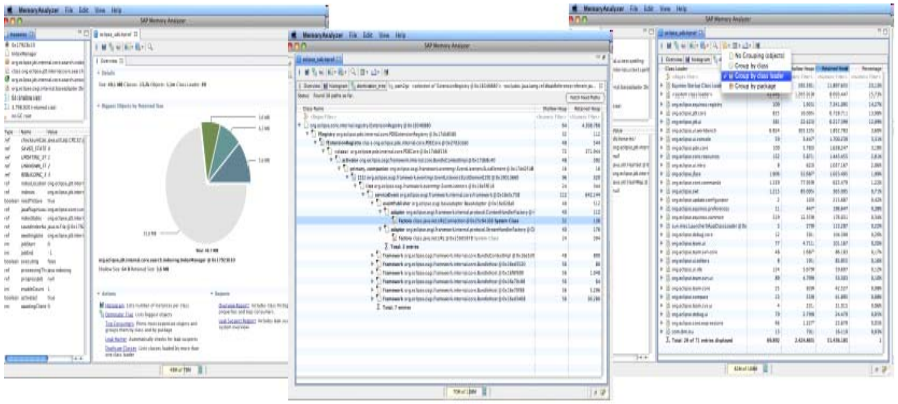


 Complete your session evaluations online at www.SHARE.org/Pittsburgh-Eval

Garbage Collection
IBM Monitoring and Diagnostic Tools for Java

Memory Analyzer

- Motivation questions:
  - Why did I run out of Java memory?
  - What's in my Java heap? How can I explore it and get new insights?



- Overview
  - Tool 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, footprint analysis and insight into wasted space
  - Provides SQL like object query language (OQL)

Garbage Collection: Summary

- Garbage Collector for Java
- IBM JVM(J9) GC - Policies
  - optthruput
  - optavgpause
  - gencon
  - balanced
- IBM JVM(J9) GC – Choosing the right policy
- IBM Monitoring and Diagnostic Tools for Java
  - Health Center
  - GCMV
  - Memory Analyzer

Session #16182:
Java Monitoring and Diagnostic Tooling
Thursday, Aug 7th, 4:15PM, room 304

Iris Baron
ibaron@ca.ibm.com


Thank You

Summary of Links

- Documentation
  - http://www.ibm.com/developerworks/java/jdk/docs.html
  - http://www.redbooks.ibm.com/redpapers/pdfs/redp3950.pdf
- zOS SDK
  - http://www.ibm.com/servers/eserver/zseries/software/java
- System z Linux SDK
  - http://www.ibm.com/developerworks/java/jdk/linux/download.html
- GC Tuning documentation
  - http://www.ibm.com/developerworks/views/java/libraryview.jsp?search_by=java+technology+ibm+style:
  - http://www-01.ibm.com/support/docview.wss?uid=swg27013824&aid=1
  - http://www.ibm.com/developerworks/websphere/techjournal/1106_bailey/1106_bailey.html#sec-ng
  - http://www.ibm.com/developerworks/websphere/techjournal/1108_sciampacone/1108_sciampacone.html
- IBM Support Assistant
  - http://www.ibm.com/software/support/isa/

References

- Java 7
  - Project Coin
    - https://www.ibm.com/developerworks/mydeveloperworks/blogs/javaee/entry/5_minute_guide_to_project_coin9?lang=en
  - NIO.2
    - http://www.ibm.com/developerworks/java/library/j-nio2-1/index.html
  - Fork/Join
    - http://www.ibm.com/developerworks/library/j-jtp11137/index.html

- WAS 8.5
  - What's new
    - http://www.ibm.com/developerworks/websphere/techjournal/1206_alcott/1206_alcott.html
  - Using Java 7 in WAS 8.5
    - http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.installation.base.doc/ae/tins_installation_jdk7.html
    - http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.nd.multiplatform.doc/ae/rxml_managesdk.html
  - Migration Tools
    - http://www.ibm.com/developerworks/websphere/downloads/migtoolkit/index.html