What’s Hot in IBM’s Virtual Machine for the Java™ Platform?

Trent Gray-Donald

Java SE JVM Lead
IBM
www.ibm.com

TS-3313

Goal

What's in it for you?

Learn about IBM's Java™ Virtual Machine offerings, the underlying design philosophy that enables scaling from a watch to a mainframe, and some of the core technology, including JIT and GC.

Agenda

Design Philosophy and History

JIT—Scalable Performance

GC—Flexible Collection Policies

Shared Classes—Smarter Resource Use

RAS—Reliability, Availability, Serviceability

Agenda

Design Philosophy and History

JIT—Scalable Performance

GC—Flexible Collection Policies

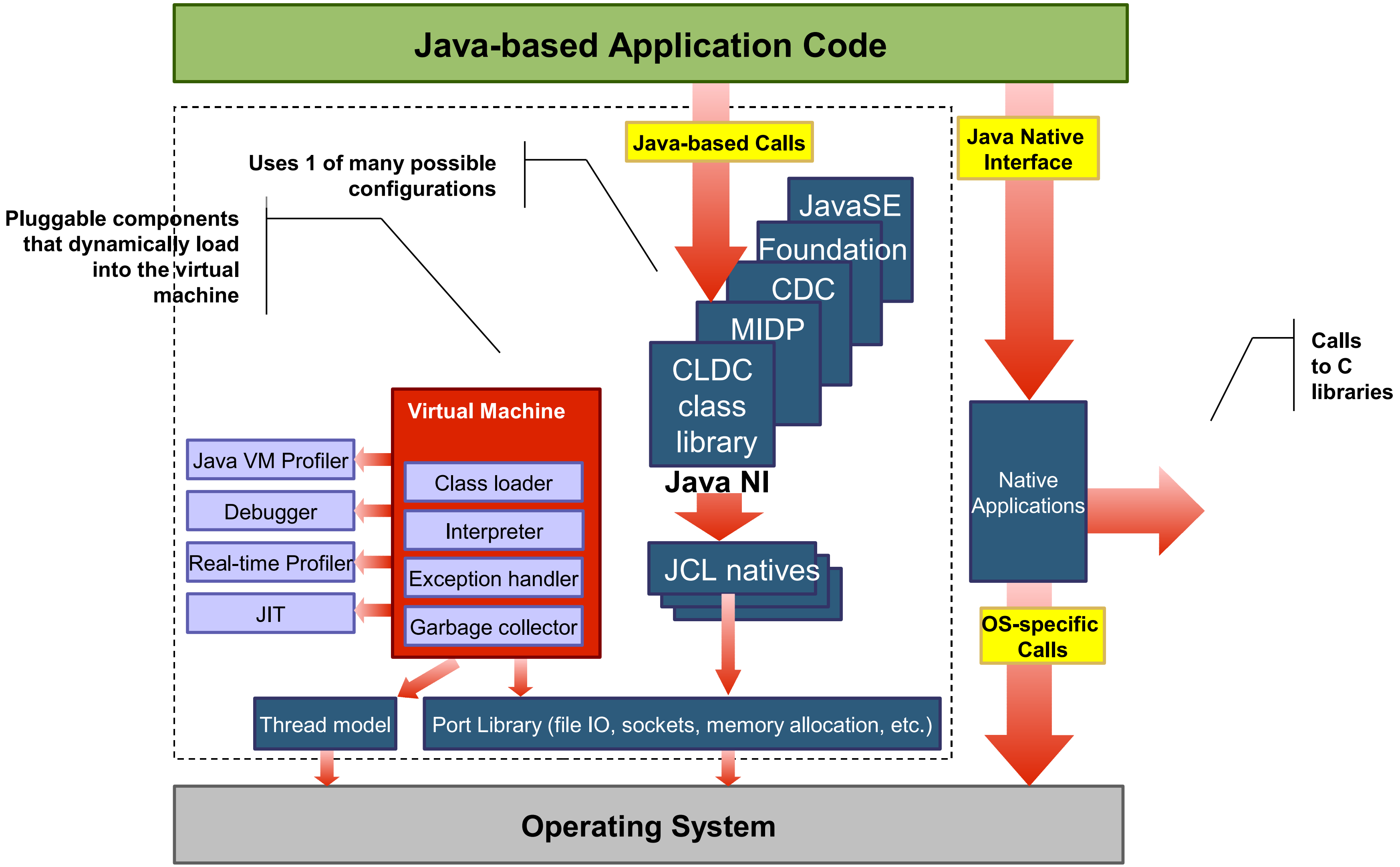
Shared Classes—Smarter Resource Use

RAS—Reliability, Availability, Serviceability

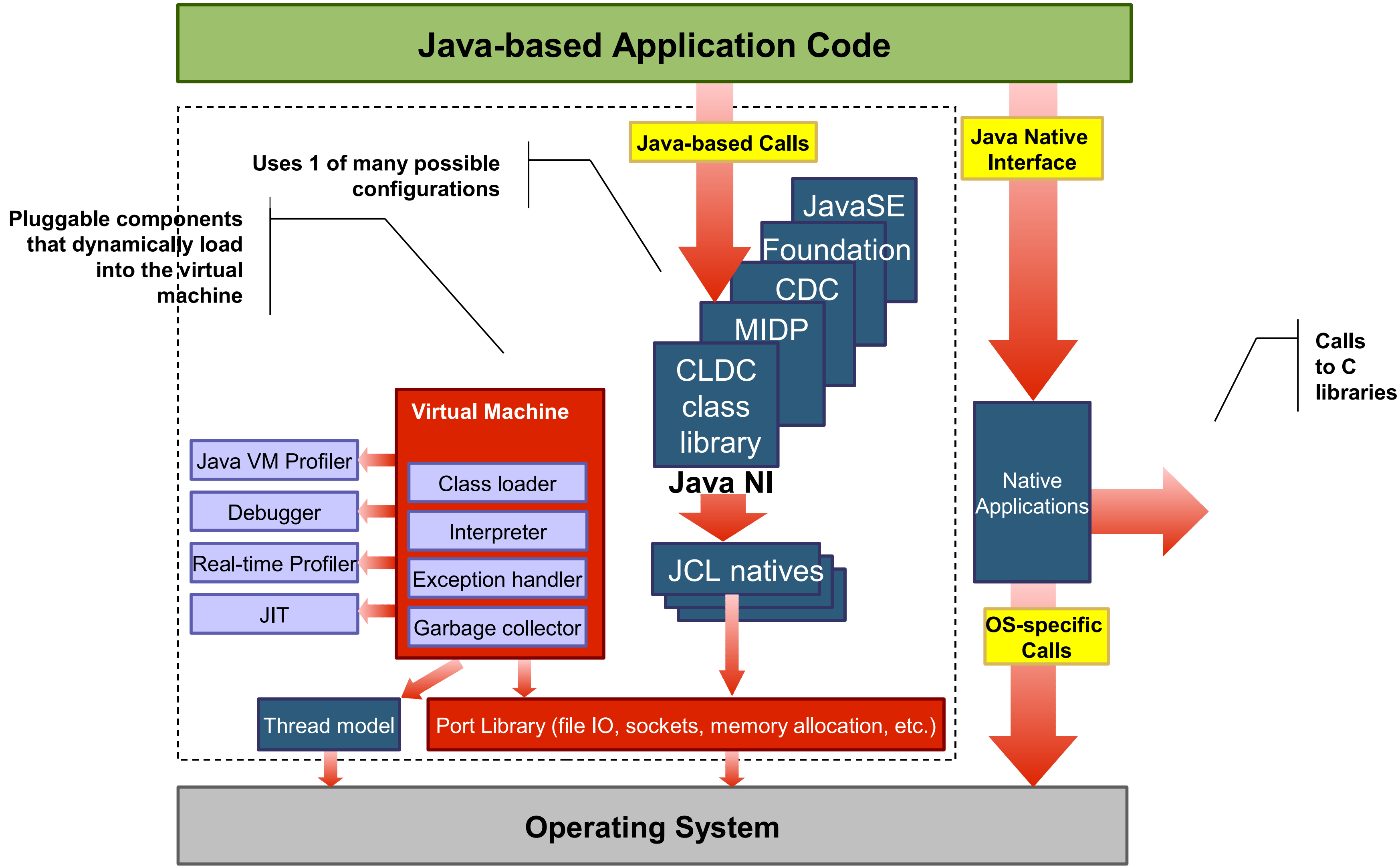
Design Philosophy and History

- Current offerings based on the J9 Virtual Machine
  - 3rd generation Java VM from IBM
  - Designed from the ground up to be a scalable solution for embedded, desktop, and server class hardware
- Common code base for all Java ME and Java SE products
  - Highly configurable—pluggable interfaces with different implementations depending on the target market
- Class library independence
- Supports latest language features (Java SE 5)
- Scaling to available hardware
  - Wide range—“From a watch to a mainframe”
    - Portable, configurable, flexible

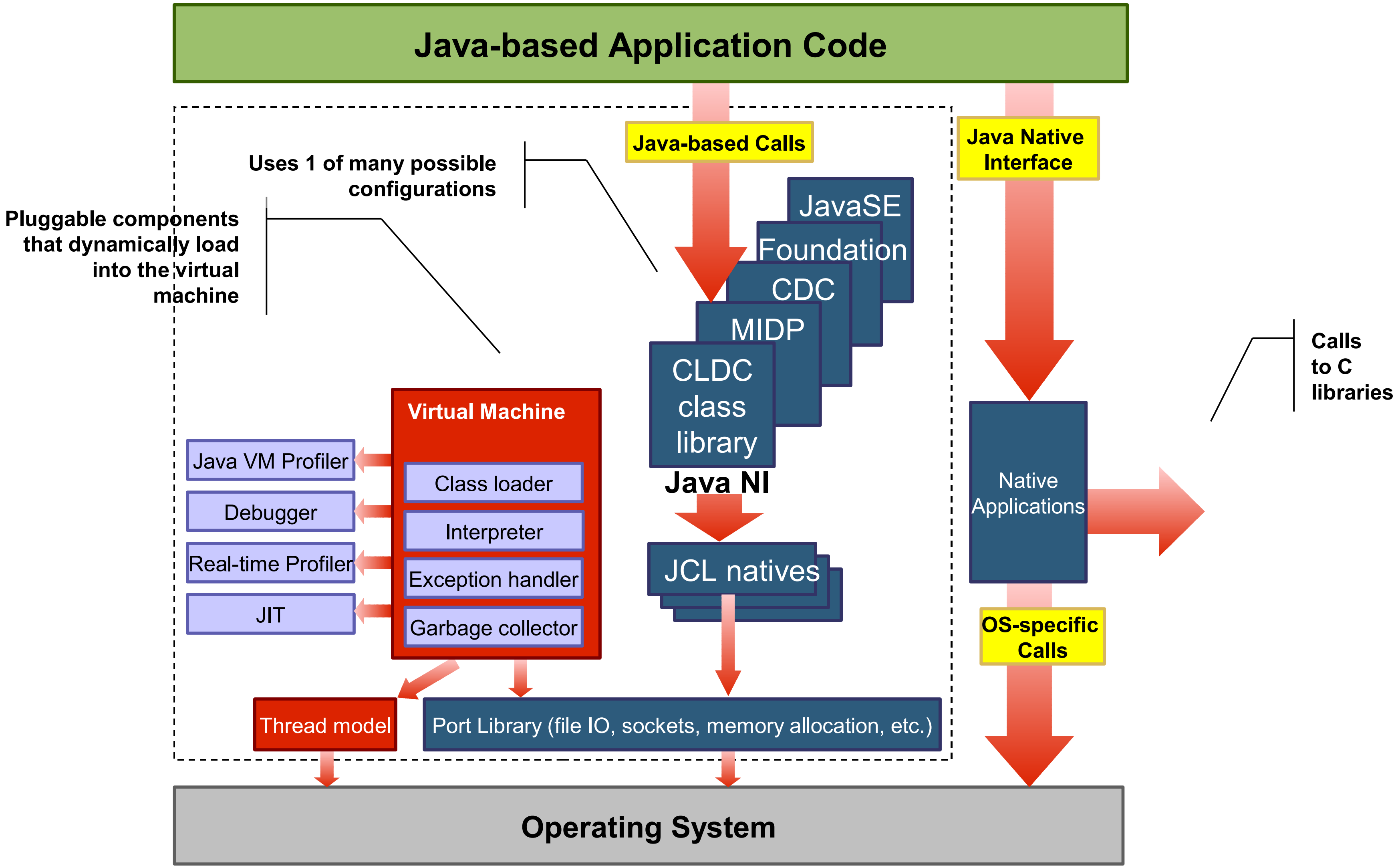
Java VM Architecture



Java VM Architecture



Java VM Architecture



Agenda

Design Philosophy and History

JIT—Scalable Performance

GC—Flexible Collection Policies

Shared Classes—Smarter Resource Use

RAS—Reliability, Availability, Serviceability

JIT Design Goals

- Java technology-centric design
- Flexible to meet different footprint goals
- Configurable optimization framework
- High-performance code with deep platform exploitation
- Complete solution: optimizing transformations fully operational in the presence of exception handling, security manager, stack trace, unresolved or volatile entities, etc.
- Dynamic recompilation with profile-directed feedback
- Fast Startup Times
- Support for Hot Code Replace and Full-speed Debug

Flexible JIT Configurations

- 3 ways to build the JIT
  - Full JIT 1–3M
    - Used with all the Java SE builds
  - Small JIT 300–600K
    - Generally bundled with the Java ME CDC offerings
    - Subset of the Full JIT
      - Optimizations from the Full JIT can easily be added to the Small JIT as small devices increase the amount of memory they have
  - Micro JIT 50–100K
    - Generally bundled with the Java ME CLDC offerings
    - Direct bytecode to native code (no intermediate language)

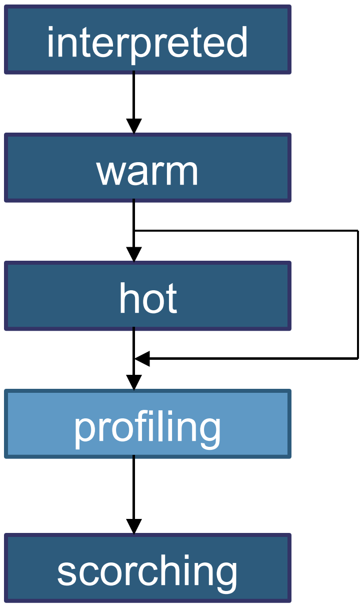
Configurable Optimization Framework

- Complete suite of classical and Java technology optimizations
- Platform neutral optimizer performs IL-IL transformations
  - Parameterized by platform-specific code to handle different CPU capabilities (e.g., number of registers)
- Multiple optimization strategies for different code quality/compile-time tradeoffs
  - Used to compose optimizations into a collection of transformations
  - Spend compile time where it makes biggest difference
  - Extremely flexible solutions and infrastructure
- Target processor optimizations

Subset of Classical Optimizations

- Loop versioning, loop unrolling
- Local and global register assigning
- Escape Analysis
- Devirtualization
- Scheduling technology shared with IBM's static compilers
- Class hierarchy optimizations

Adaptive Compilation in TR JIT



- Methods start out being interpreted
- After N invocations methods get compiled at ‘warm’ level
- Sampling thread used to identify hot methods
- Methods may get recompiled at ‘hot’ or ‘scorching’ levels
- Transition to ‘scorching’ goes through a temporary profiling step

Agenda

Design Philosophy and History

JIT—Scalable Performance

GC—Flexible Collection Policies

Shared Classes—Smarter Resource Use

RAS—Reliability, Availability, Serviceability

Garbage Collection

- Workloads
  - Transactional (e.g., web)
  - Batch (javac/Eclipse workspace build)
- Hardware varies tremendously
  - 1MB to 128GB
  - Single hardware thread up to 128-way multi-core Simultaneous MultiThreading (SMT) hardware
  - Memory Consistency differences
    - z/Series®
    - p/Series®

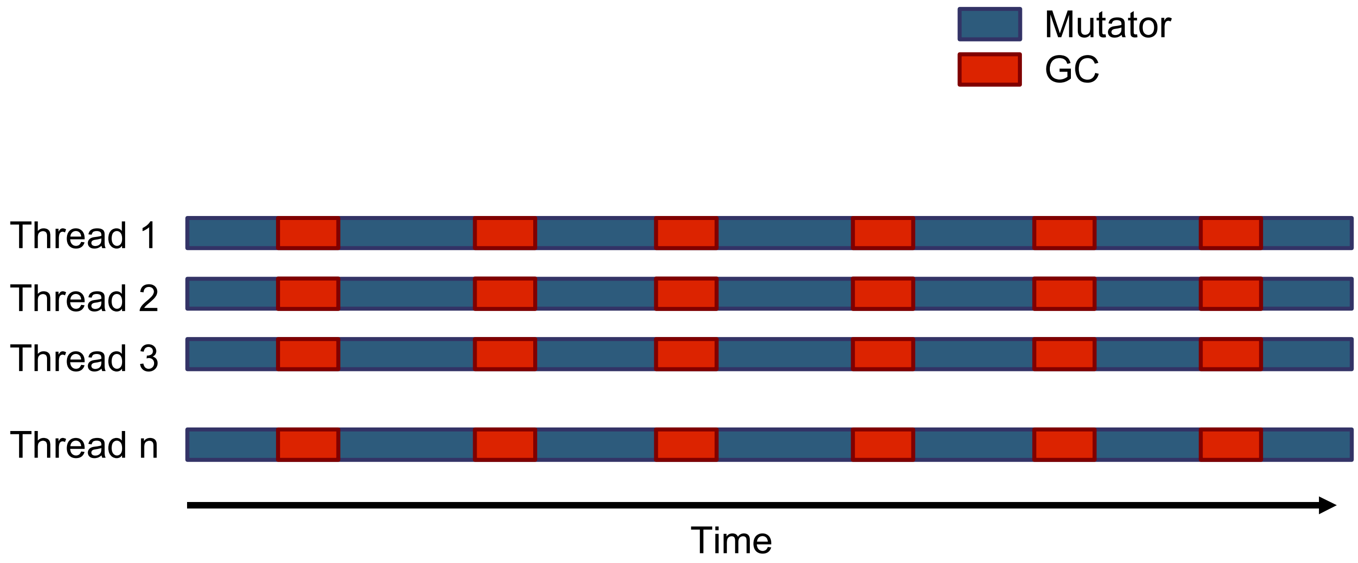
GC Policies in IBM SDK 5.0

- Optimize for Throughput –Xgcpolicy:optthruput (default)
  - Mark-Sweep-Compact tracing collector
- Optimize for PauseTime –Xgcpolicy:optavgpause
  - Concurrent Mark and Sweep
- Subpooling –Xgcpolicy:subpool
  - Mark-Sweep-Compact tracing collector
  - Designed to reduce heap lock contention on SMP systems
- Generational Concurrent –Xgcpolicy:gencon
  - Generational Copy-Collector with concurrent collection

GC Policies in IBM SDK 5.0

How do the policies compare?

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

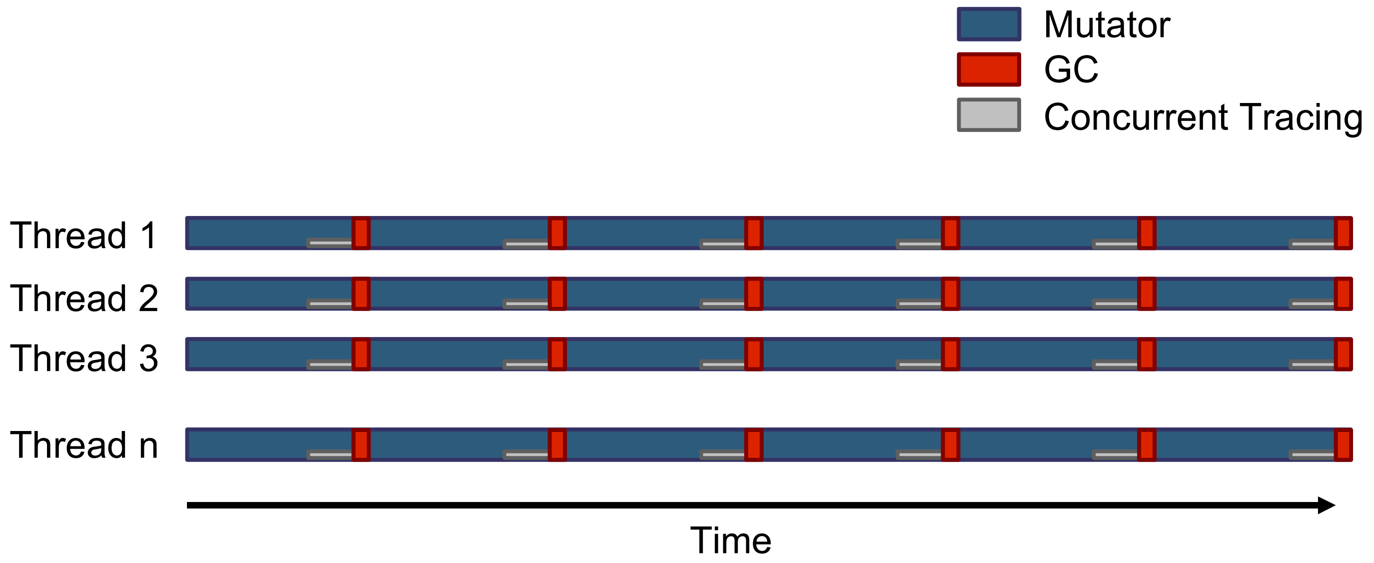


Picture is only illustrative and doesn't reflect any particular real-life application. The purpose is to show theoretical differences in pause times between GC policies.

GC Policies in IBM SDK 5.0

How do the policies compare?

-Xgcpolicy:optavgpause

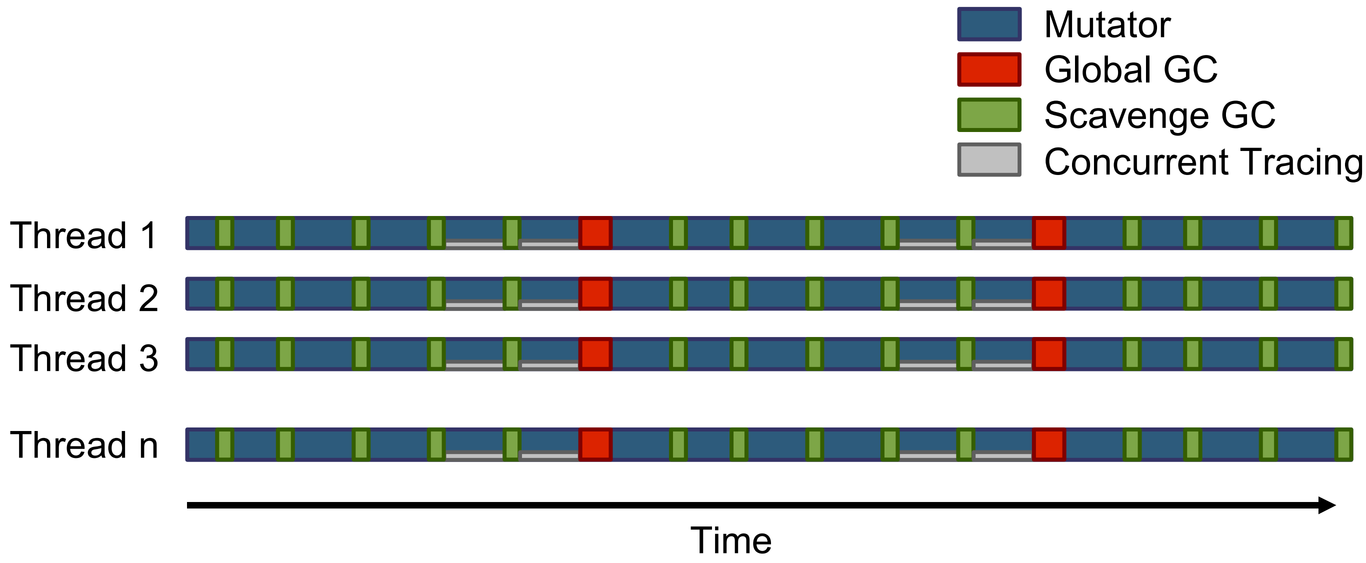


Picture is only illustrative and doesn't reflect any particular real-life application. The purpose is to show theoretical differences in pause times between GC policies.

GC Policies in IBM SDK 5.0

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.

Tuning

Allocation request details, time it took to stop all mutator threads.

Heap occupancy details before GC.

Details about the scavenge.

Heap occupancy details after GC.

```
<af type="nursery" id="1134" timestamp="Fri May 05 17:10:32 2006" intervalms="1882.416">
  <minimum requested_bytes="88" />
  <time exclusiveaccessms="0.109" />
  <nursery freebytes="0" totalbytes="56465408" percent="0" />
  <tenured freebytes="629133648" totalbytes="1191182336" percent="52" >
    <soa freebytes="617221968" totalbytes="1179270656" percent="52" />
    <loa freebytes="11911680" totalbytes="11911680" percent="100" />
  </tenured>
  <gc type="scavenger" id="1134" totalid="1142" intervalms="1882.915">
    <flipped objectcount="309209" bytes="10003828" />
    <tenured objectcount="150595" bytes="4840096" />
    <refs_cleared soft="234" weak="0" phantom="0" />
    <finalization objectsqueued="7" />
    <scavenger tiltratio="84" />
    <nursery freebytes="45832192" totalbytes="56475648" percent="81" tenureage="2" />
    <tenured freebytes="624027856" totalbytes="1191182336" percent="52" >
      <soa freebytes="612116176" totalbytes="1179270656" percent="51" />
      <loa freebytes="11911680" totalbytes="11911680" percent="100" />
    </tenured>
    <time totalms="86.619" />
  </gc>
  <nursery freebytes="45830144" totalbytes="56475648" percent="81" />
  <tenured freebytes="624027856" totalbytes="1191182336" percent="52" >
    <soa freebytes="612116176" totalbytes="1179270656" percent="51" />
    <loa freebytes="11911680" totalbytes="11911680" percent="100" />
  </tenured>
  <time totalms="87.229" />
</af>
```

What Policy Should I Choose?

I want my application to run to completion as quickly as possible.

-Xgcpolicy:optthruput

My application requires good response time to unpredictable events.

-Xgcpolicy:optavgpause

What Policy Should I Choose?

My application has a high allocation and death rate.

-Xgcpolicy:gencon

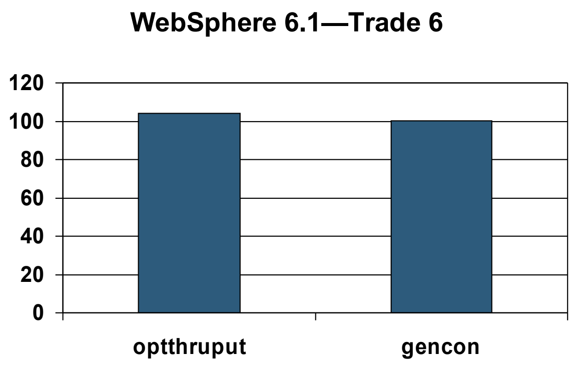
My application is running on big metal and has high allocation rates on many threads.

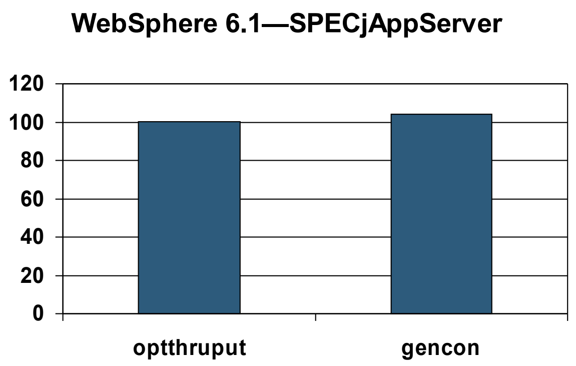
-Xgcpolicy:subpool

Danger! Caveat Emptor…

- Some WebSphere® applications perform better with gencon—however, some applications degrade in performance

- Peak throughput performance versus lower GC pause times tradeoffs possible





Numbers are approximate and only intended to show a general behaviour seen when running Trade6 compared to SPECjAppServer.

Garbage Collection FAQ

- "I want to reduce my maximum pause time."
  - Lock the new generation size to a fixed value
    - Fixed number of possible live objects per collect
  - Adjust the taxation rate of the concurrent collector
    - Amortize the cost of collection over longer periods
  - Disable compaction
    - Reduced large pauses at the expense of increased fragmentation

Garbage Collection FAQ

- "I want my system to handle occasional large objects allocations without Garbage Collecting every time."
  - Increase the size of the large object area
    - Reduce need for garbage collector to create space by pre-reserving heap
  - Adjust concurrent collector metering against regular allocates, large object allocates, or both
    - Determined by frequency of large object allocates

Garbage Collection—Tuning gencon

- Balancing nursery and the tenured space
- Automatic
  - Specify the minimum and maximum heap size (e.g., –Xms512m –Xmx1024m)
  - JDK 5 nursery will not automatically grow beyond 64MB
- Hand Tuning
  - Main factors are new object death rates, tenure space used
  - Recommended approach for performance-sensitive, server-side applications

Agenda

Design Philosophy and History

JIT—Scalable Performance

GC—Flexible Collection Policies

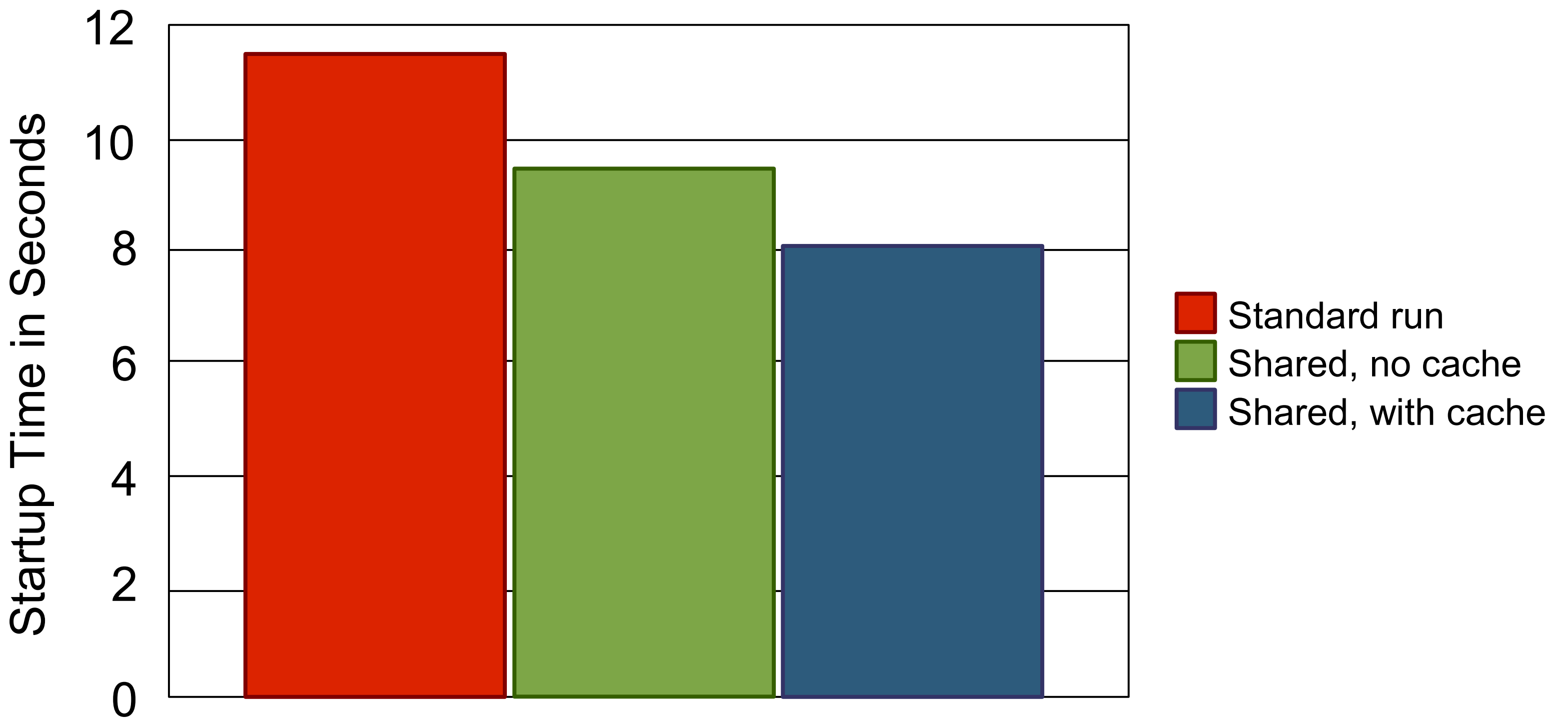
Shared Classes—Smarter Resource Use

RAS—Reliability, Availability, Serviceability

Shared Classes

- Server environments where multiple JVMs exist on the same box
- Improves startup time and memory footprint
- Sharing of class data—granularity is .class file
- Multiple sharing strategies
  - Standard classloaders (including Application Classloader) exploit the feature when enabled
  - API to extend custom ClassLoaders available

Shared Classes Performance



Agenda

Design Philosophy and History

JIT—Scalable Performance

GC—Flexible Collection Policies

Shared Classes—Smarter Resource Use

RAS—Reliability, Availability, Serviceability

Reliability, Availability, Serviceability (RAS)

- Top IBM focus
  - Necessary for effective support of over 1,800 IBM products on top of Java
- Problem types are varied
  - Hangs
  - Spins
  - Unexpected Java code exceptions/errors
  - Crashes (because of user Java NI code or Java VM code)
  - Performance issues

RAS Tools—Java NI

- Difficult problem area
  - Different Java VM behaviours
    - Arrays—copy vs. pin
    - References (reuse, lifetimes)
  - Non-Java technology paradigm
    - Explicit exception checks
    - Allocation failures
- -Xcheck:jni
  - Validates a full range of Java NI errors seen internally and in customer code
  - Range from critical to pedantic

RAS Tools –Xdump and DTFJ

- Trigger informative dumps (Java based/ system/heap) on a multitude of events
  - e.g., OutOfMemoryError produces javacore.txt and a heap dump
- Events include class load, exception throw, thread start/stop, GC
- Dump Tools Framework for Java Technology (DTFJ)
  - Toolkit to allow programmatic introspection into different RAS artifacts (system dumps, etc.)
  - TS-3881 for a deep dive

RAS Tools –Xdump File Types

- Javacore.txt (human and machine readable)
  - Java VM version
  - Java VM arguments, OS paths
  - Major Java VM structure addresses
  - Java-based Threads with stack trace
  - Monitors with owners, blockers, waiters
  - Classloaders and classes
- Heapdump.phd (machine readable)
  - Dense Java-based heap contents
- System dumps (OS specific)

RAS Tools –Xdump Examples

- -Xdump:java:events=throw,filter=MyException
- -Xdump:heap:events=unload,filter=MyClass
- -Xdump:java:events=load,range=4..7

```
JVMDUMP006I Processing Dump Event "load", detail "java/lang/reflect/GenericDeclaration" -
Please Wait.
JVMDUMP007I JVM Requesting Java Dump using C:\150\jre\bin\javacore.20060425.010831.2720.txt
JVMDUMP010I Java Dump written to C:\150\jre\bin\javacore.20060425.010831.2720.txt
JVMDUMP013I Processed Dump Event "load", detail "java/lang/reflect/GenericDeclaration".
```

RAS Tools –Xtrace

- Provides tiered tracing of Java VM internal program flow and Java technology-level method execution
- Always on for some internal trace in JDK™ 5.0 software
  - Very useful for First Fail Data Capture (FFDC) purposes
- GC statistics kept in separate rolling trace buffer and dumped into javacore*.txt files whenever triggered

Observations on the Future

(NB: No guarantee re: future products or research)

- Quality of service
  - Pause time (not just GC!)
  - Performance
  - RAS
- Real-time GC
  - Metronome (1ms max pause time)
- Multi-core CPUs
- NUMA
- 64-bit Java VMs and hybrid solutions

Summary

Design Philosophy and History

JIT—Scalable Performance

GC—Flexible Collection Policies

Shared Classes—Smarter Resource Use

RAS—Reliability, Availability, Serviceability

Q&A

What's Hot in IBM's Virtual Machine for the Java™ Platform?

Trent Gray-Donald

Java SE JVM Lead
IBM
www.ibm.com

TS-3313