IBM Java 6.0.1 in zEnterprise Technology Update

Session 09769

Ken Irwin, IBM GTS, Poughkeepsie, New York
Marcel Mitran, IBM SWG, Markham, Canada
Theresa Tai, IBM STG, Poughkeepsie, New York

IBM J9 2.6 Technology Innovation Highlight and z/OS Value-add

- IBM Strategy Initiative on Java
- IBM J9 2.6 Technology Innovation Highlight
- z/OS value-add
  - The newly announced z/OS V1.13 Batch Runtime
  - Maximizing zAAP, zIIP, and zAAP on zIIP Investment
- IBM Java Roadmap and Future
- SDK V6.x, V5.x and V1.4.2 Currency
- System zEnterprise z196 New Workload Video
  - http://www.centerline.net/review/#/3332_B

IBM and Java

- **Java is critically important to IBM**
  - Fundamental infrastructure for IBM's software portfolio
  - WebSphere, Lotus, Tivoli, Rational, Information Management (IM)
  - CICS, IMS and DB2
- **IBM is investing strategically for Java in Virtual Machines**
  - As of Java 5.0, single JVM support (JME, JSE, JEE)
  - New technology base (J9/TR Compiler) on which to deliver improved performance, reliability, serviceability
- **IBM also invests in, and supports public innovation in Java**
  - Eclipse, Apache (XML, Aries, Derby, Geronimo, Harmony, Tuscany, Hadoop …)
  - Broad participation in relevant open standards (JCP, OSGi)

IBM J9 2.6 Technology Innovation

System zEnterprise 196 and Java6.0.1

It's about performance and further improve the economics of running mission critical workloads on zEnterprise

- Leveraging 70+ new HW instructions
- HW Optimization Technology for Java
  - Reducing pressure on instruction cache and data cache
  - New architectural facilities designed for scalability, and concurrency
  - General optimizer and codegen improvements
- New GC Balanced Policy for large heap-intensive workloads
- Significant just-in-time (JIT) compiler performance enhancements
- z/OS Java Unique Security Enhancements
  - Integrated Cryptographic Service Facility (ICSF) exception handling
- New RAS features and new signal handling capability
- Enhancements to JZOS

Executive Summary

z196 and Java6.0.1: Engineered Together

- Up-to 2.1x improvement to Java throughput
- Reduced footprint
- Tighter integration with z/OS facilities
- Improved responsiveness in application behavior

J9 R2.6 Virtual Machine

- Significant enhancements to JIT optimization technology
- z196 exploitation of instructions and new pipeline
- New Balanced GC policy to reduce max pause times

z/OS Unique Enhancements

- JZOS 2.4.0
- z/OS Java unique security enhancements

Performance

- 2.1x improvement to multi-threaded workload
- 1.93x improvement to CPU-intensive workload

SHARE in Orlando 2011

Java Execution Environments and Interoperability

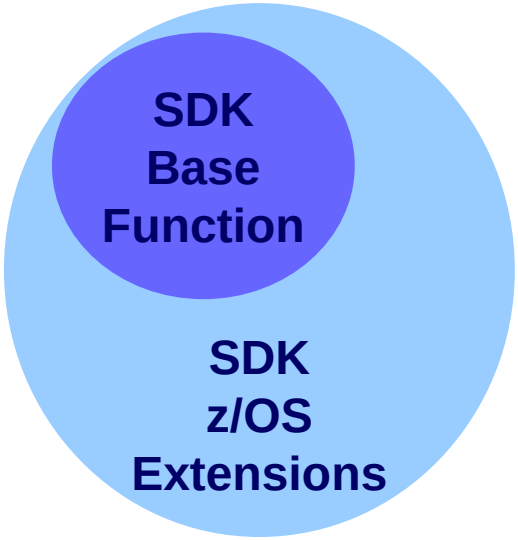
Capitalize on Pre-existing Assets, Artifacts, Processes, Core Competencies and Performance Strengths.

- **IBM Java Execution Offerings**
  - Transactional/Interactive
    - WebSphere for z/OS (WAS z/OS)
    - WebSphere Process Server for z/OS (WPS) for SOA BPM
    - JCICS
    - IMS Java
    - DB2 Stored Procedures
  - Batch Oriented
    - WebSphere Compute Grid (WAS-CG)
      - *WAS/JEE runtime extensions*
    - JZOS component of z/OS SDK
      - *JES/JSE-based environment*
- **Open Source or non-IBM vendor Application Server and Frameworks**
  - Tomcat, JBoss
  - iBatis, Hibernate, Spring
  - Ant
- **COBOL/Native Interoperability**
  - COBOL Invoke maps to JNI
  - RDz and JZOS** have tooling to map COBOL copy books to Java classes
  - JCICS
  - IMS Java, JMP/JBP
  - WAS CG, WOLA
  - etc

* See http://www-01.ibm.com/common/ssi/cgi-bin/ssialias?subtype=ca&infotype=an&supplier=897&letternum=ENUS211-252

** Alphaworks only, and hence currently un-supported

z/OS – System z Java Extensions



All SDKs support the ‘standards’, Java on z/OS extends the SDK

- Access to z/OS services
- Access to all types of data
- Access under control of z/OS security mechanisms
- Integration into existing operational infrastructure

Services available in JEE and JSE environments under the restrictions of the container.

System specific extension allow you to write robust middleware and applications that integrate with traditional z/OS operating environment

- Allow for maintaining platform independent design development.
- Platform specific implementations when required
- Allows for operational and resource optimization

e.g. JAAS wrapper of SAF (RACF, ACF2, or TopSecret), Traditional OS dataset access, Cryptographic hardware (Cards and CPACF), z/OS Console (modify and messages), z/OS system logger, JES job submission, DFSORT, SMF, etc.

The New z/OS Batch Runtime Environment

- z/OS V1.13 "real-time batch" support
  - A new z/OS base component
  - Enable concurrent batch and online data access
  - Provides the framework for
    - Java-to-COBOL interoperability
    - Transactional updates to DB2®
    - Sharing database connections between Java and COBOL
- New Java-COBOL interoperability capabilities are designed to enable re-use valuable COBOL assets by developing new and/or enhancing existing batch applications with Java
  - Example - use Java subroutines directly in lieu of Java stored procedures
- Leverage Specialty Engine zAAP

Getting the Most out of IBM zAAP, zIIP, zAAPzIIP with zEnterprise

- Significantly more productive with zEnterprise class of processors
  - You should expect lower utilization on the same workloads
  - Additional Performance and Throughput Gains with z196 and the new z114
- z/OS Management Facility (z/OSMF) exploitation of zAAP and zIIP (zAAPzIIP) engines
  - Parts of z/OSMF use the z/OS CIM Server
  - Java Workloads eligible for zAAP, or zIIP (with the zAAP on zIIP capability introduced with z/OS V1.11)
- New Batch Runtime with z/OS V1.13

IBM Java Road Map

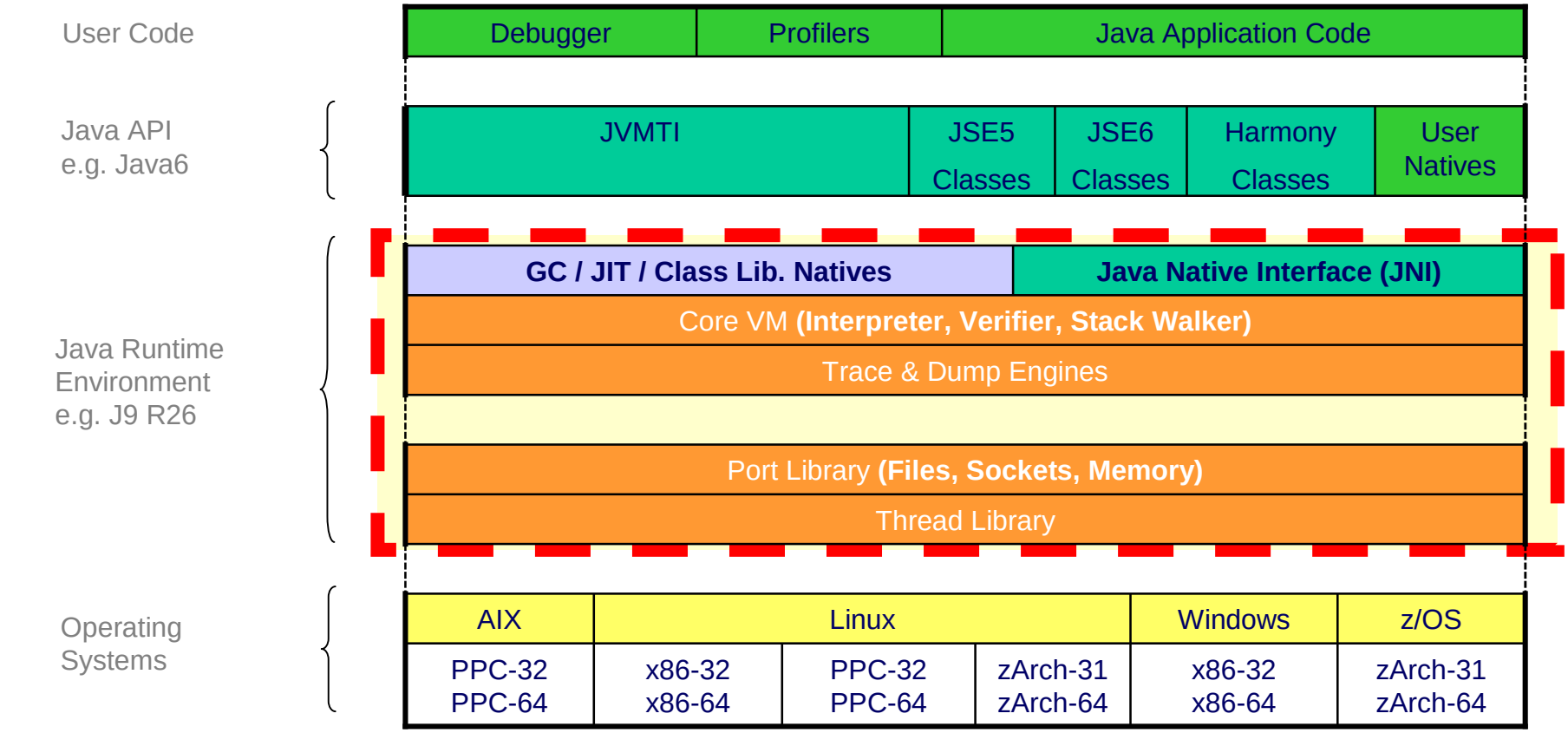
Page Intentionally Left Blank

IBM Java SDKs Currency

- Java Technology Edition V6.0.1 Build Level April 19, 2011 for z/OS
  - 31-bit and 64-bit SDK IBM J9 2.6 VM, a new level of JZOS (2.4.0), enhancements to z/OS Java security, and exploits z196 instructions. The existing Version 6 Release 0 Modification 0 remains orderable and in service
  - 31-bit - PTFs UK68991 and UK68998 / APARs PM40891 and PM40892
  - 64-bit - PTFs UK69000 and UK69001 / APARs PM40894 and PM40895
- Java Technology Edition V6.0 Build Level June 25, 2011 for z/OS
  - 31-bit - PTF UK65180/APAR PM33607/SDK6 SR9 FP1
  - 64-bit - PTF UK65285/APAR PM33609/SDK6 SR9 FP1
- Java Technology Edition V5.0 Build Level June 27, 2011 for z/OS
  - 31-bit - PTF UK69795/APAR PM43607/SDK5 SR12 FP5
  - 64-bit - PTF UK69796/APAR PM43609/SDK5 SR12 FP5
- Java Technology Edition V1.4 Build Level March 3, 2011 for z/OS
  - PTF UK66018/APAR PM35678/SDK1.4.2 SR13 Fixpack 9
  - No longer orderable
  - Has been withdrawn from Service effective September 30, 2011

IBM Java Performance on zEnterprise

JVM Architectural Overview



= User Code
= Java Platform API
= VM-aware
= Core VM

Java 6.0.1:

- Also referred to as Java6 R26, ships with WAS8 across platforms, or standalone on z/OS
- Fully compatible/compliant Java6 (JSE6)
- Includes new J9 R26 JRE (replacing J9 R24 in Java6.0.0)
  - Transparent z196 and new optimization exploitation
  - New balanced GC policy

IBM Java Runtime Environment

- IBM's implementation of Java 5 and Java 6 are built with **IBM J9 Virtual Machine** and **IBM Testarossa JIT Compiler** technology
  - Independent clean-room JVM runtime & JIT compiler

- Combines best-of breed from embedded, development and server environments… from a cell-phone to a mainframe!
  - Lightweight flexible/scalable technology
  - World class garbage collection – gencon, balanced GC policies
  - Startup & Footprint - Shared classes, Ahead-of-time (AOT) compilation
  - 64-bit performance - Compressed references & Large Pages
  - Deep System z exploitation – z196/z10/z9/z990 exploitation
  - Cost-effective for z - zAAP Ready!

- Millions of instances of J9/TR compiler

IBM Testarossa JIT
Deep System z Exploitation

Out-of-Order/Super-scalar Instruction Scheduler

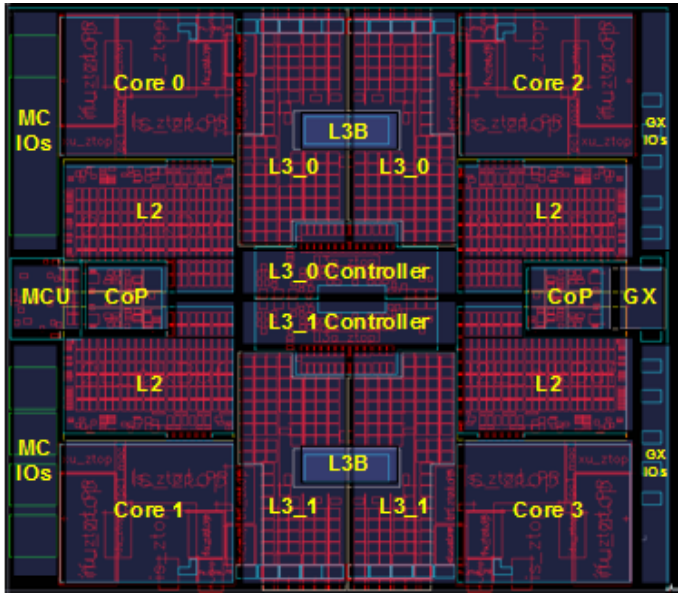
- z10/z9/z990 are in order superscalar dual pipelines, can dispatch up to 3 instrs/cycle
- z196 pipeline is OOO, permits 3 inst/group

Platform Tuned Optimizations

- idiom recognition, dynamic literal pool, etc

New Hardware Facility Exploitation

- z196: high-word, non-destructive, interlock, conditional load/store
- z10: traps, compare-and-branch, pre-fetch, Decimal Floating Point, etc
- z9: extended immediate support
- z990: long displacement support



IBM J9 2.6 and z196

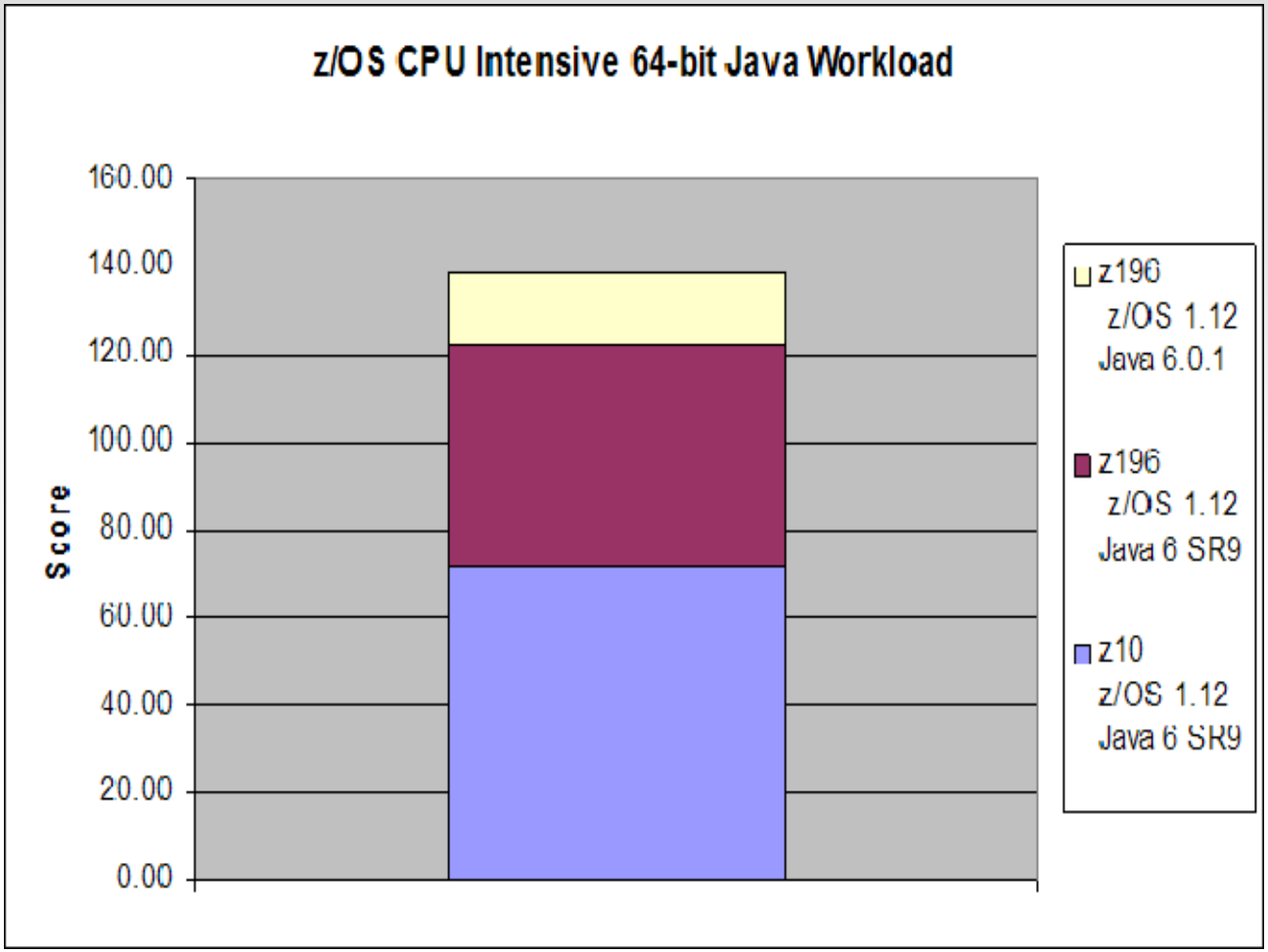
z196: Hardware for Java

- New Out-Of-Order pipeline design
- New larger cache structure
- Higher clock speed (~5.2GHz)

J9 R26: JRE for z196

- Reducing pressure on the data/instruction cache
  - Enables better exploitation of new OOO compute bandwidth
  - Mitigates effects of cache latencies for leveraging core speed
- Concurrency improvements
  - Better scalability
- General optimizer and codegen improvements
  - Reduced path-length

Performance on z/OS: CPU-Intensive Benchmark

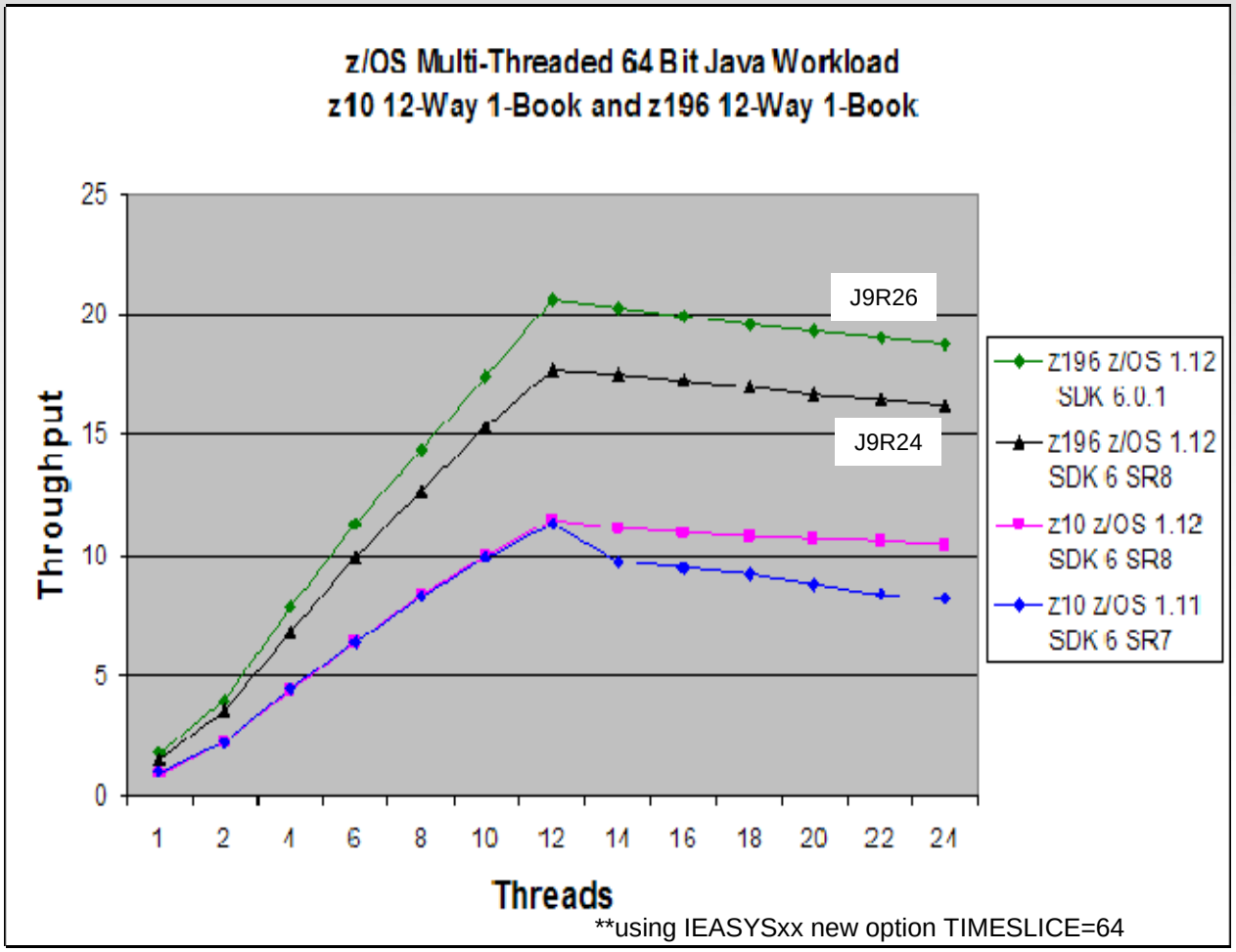


93% Aggregate improvement

- 14% Java 6.0.1 improvement
- 70% Hardware improvement

(Controlled measurement environment, results may vary)

Performance on z/OS: Multi-threaded Benchmark

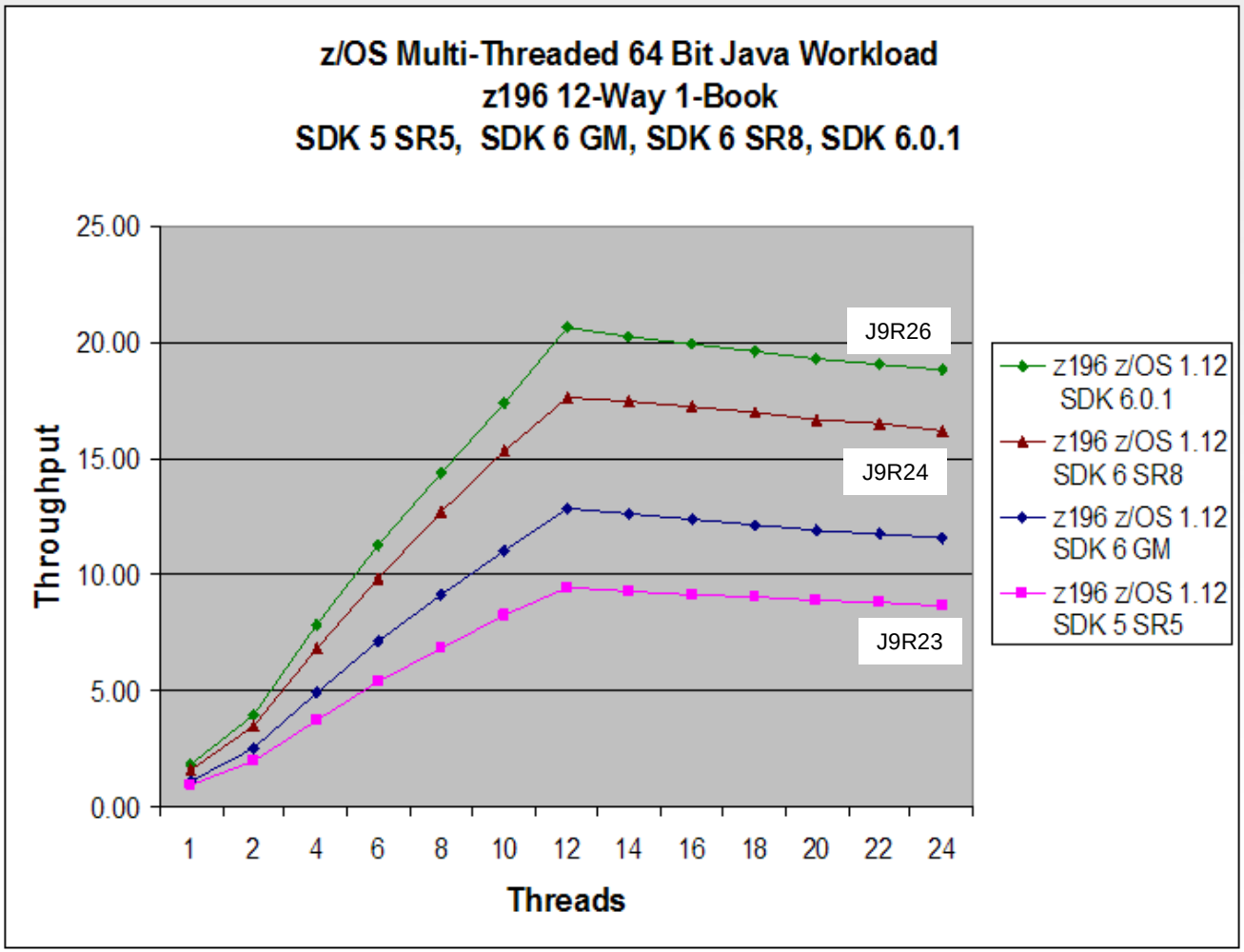


2.1x Aggregate improvement

- 16% Java 6.0.1 improvement
- 56% Hardware improvement
- 17% z/OS 1.12 improvement**

(Controlled measurement environment, results may vary)

z/OS Java SDK 6.0.1 Performance: 64 Bit Multi-threaded Benchmark



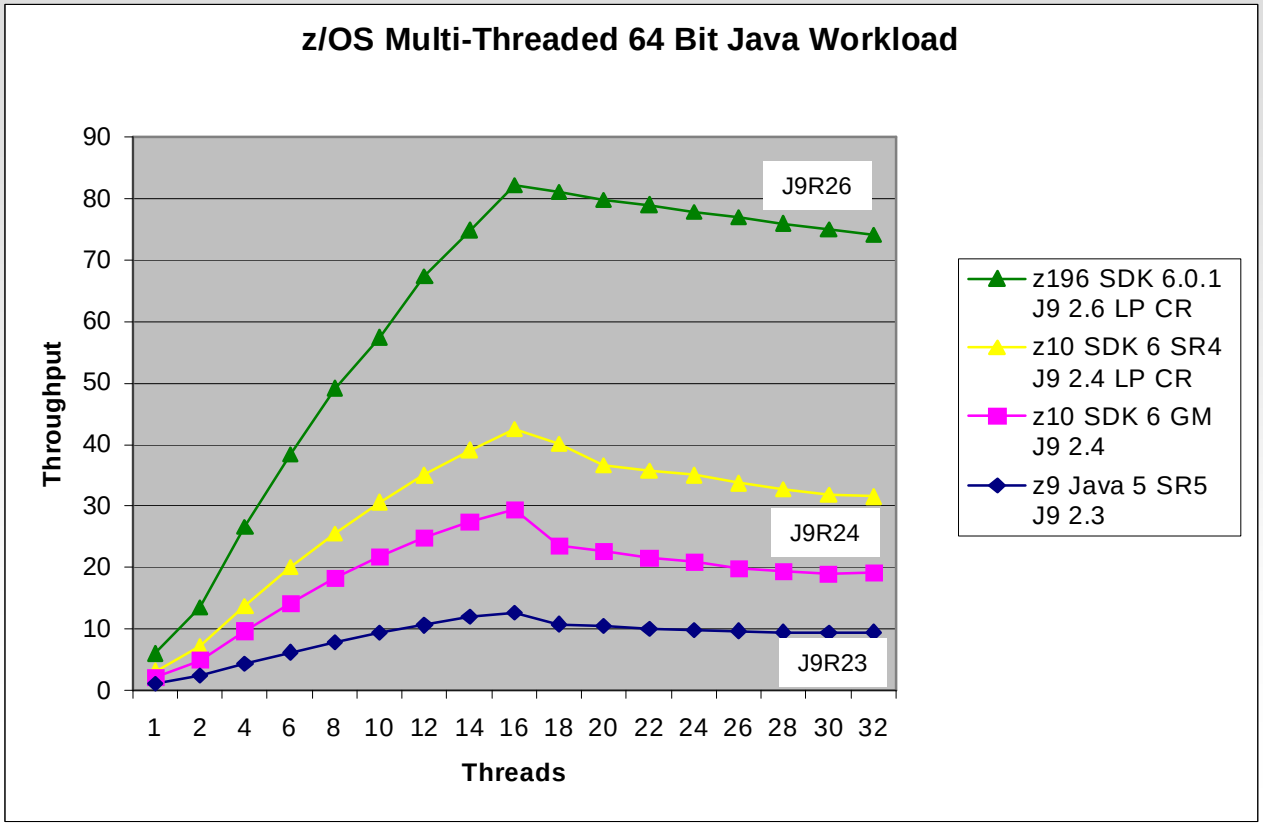
2.17x Aggregate Software improvement

- 16% Java 6.0.1 improvement
- 39% Java 6 SR8 versus Java 6 GM improvement
- 35% Java 6 GM versus Java 5 SR5

(Controlled measurement environment, results may vary)

z/OS Java SDK 6.0.1 Performance

Aggregate HW and SDK Improvement z10, z196, Java6 to Java6.0.1

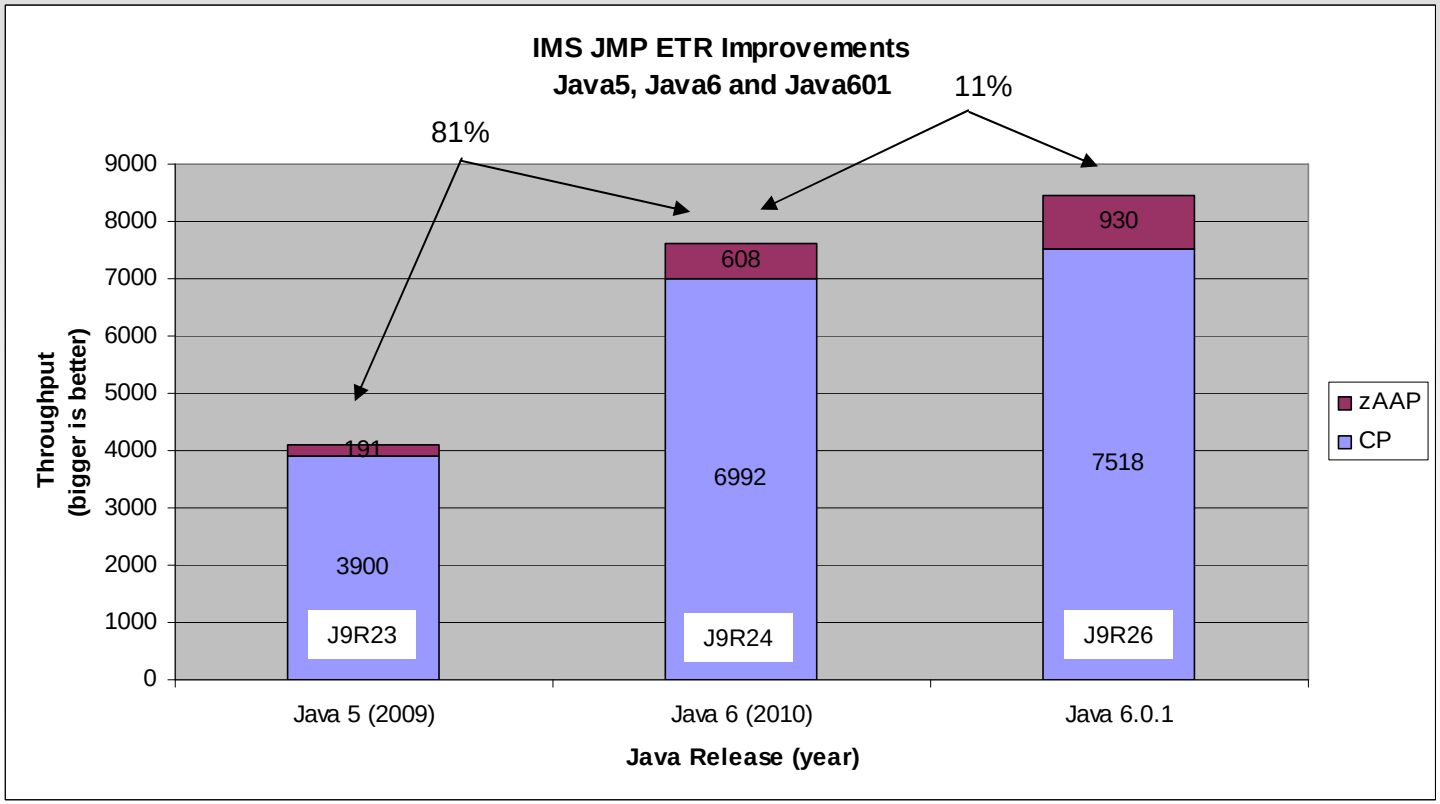


~7x Improvement from z10, z196, Java6 and Java6.0.1

Note: z9 (J9 R23) is the base for the aggregated HW and SDK performance improvements

(Controlled measurement environment, results may vary)

Performance – IMS JMP

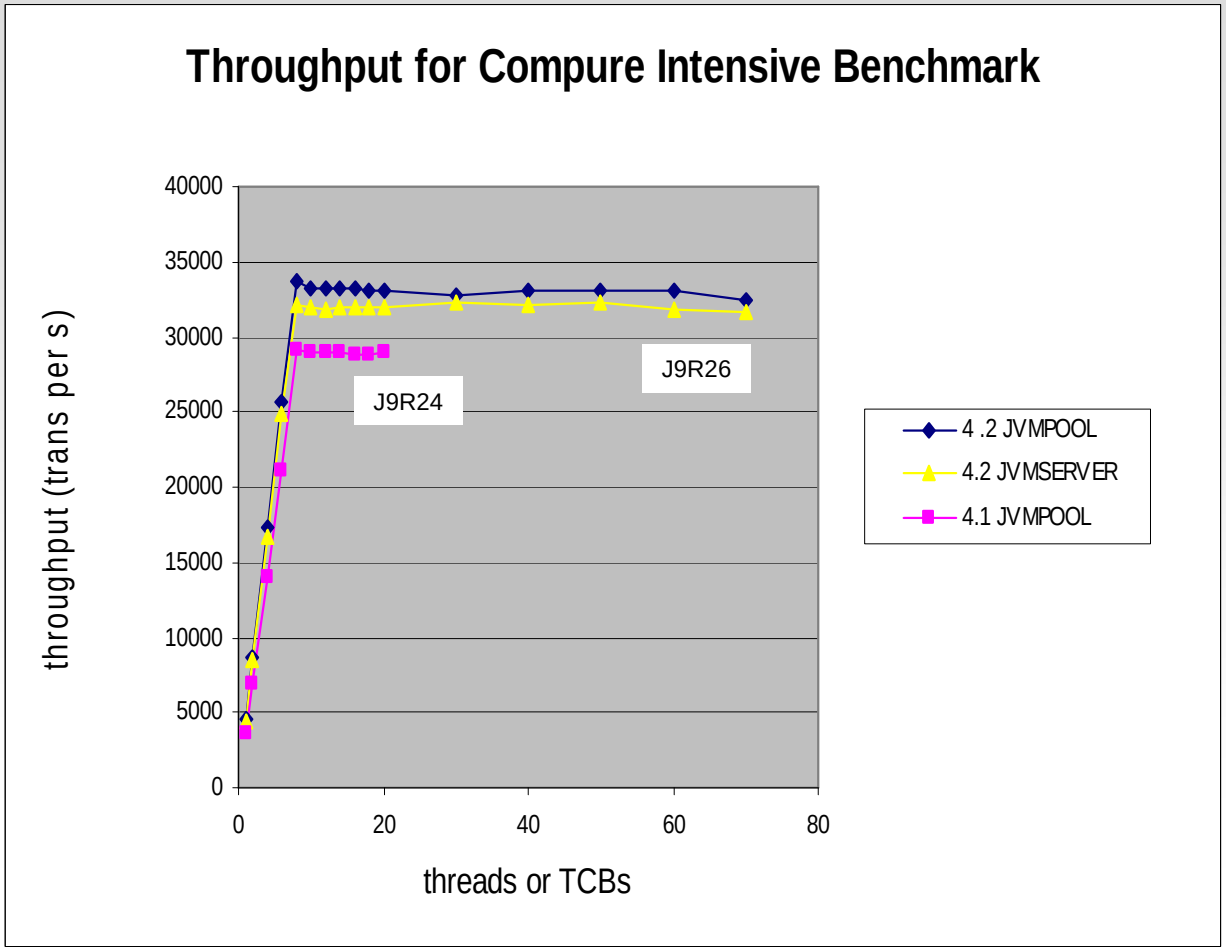


2 GCP + 2 zAAP

(Controlled measurement environment, results may vary)

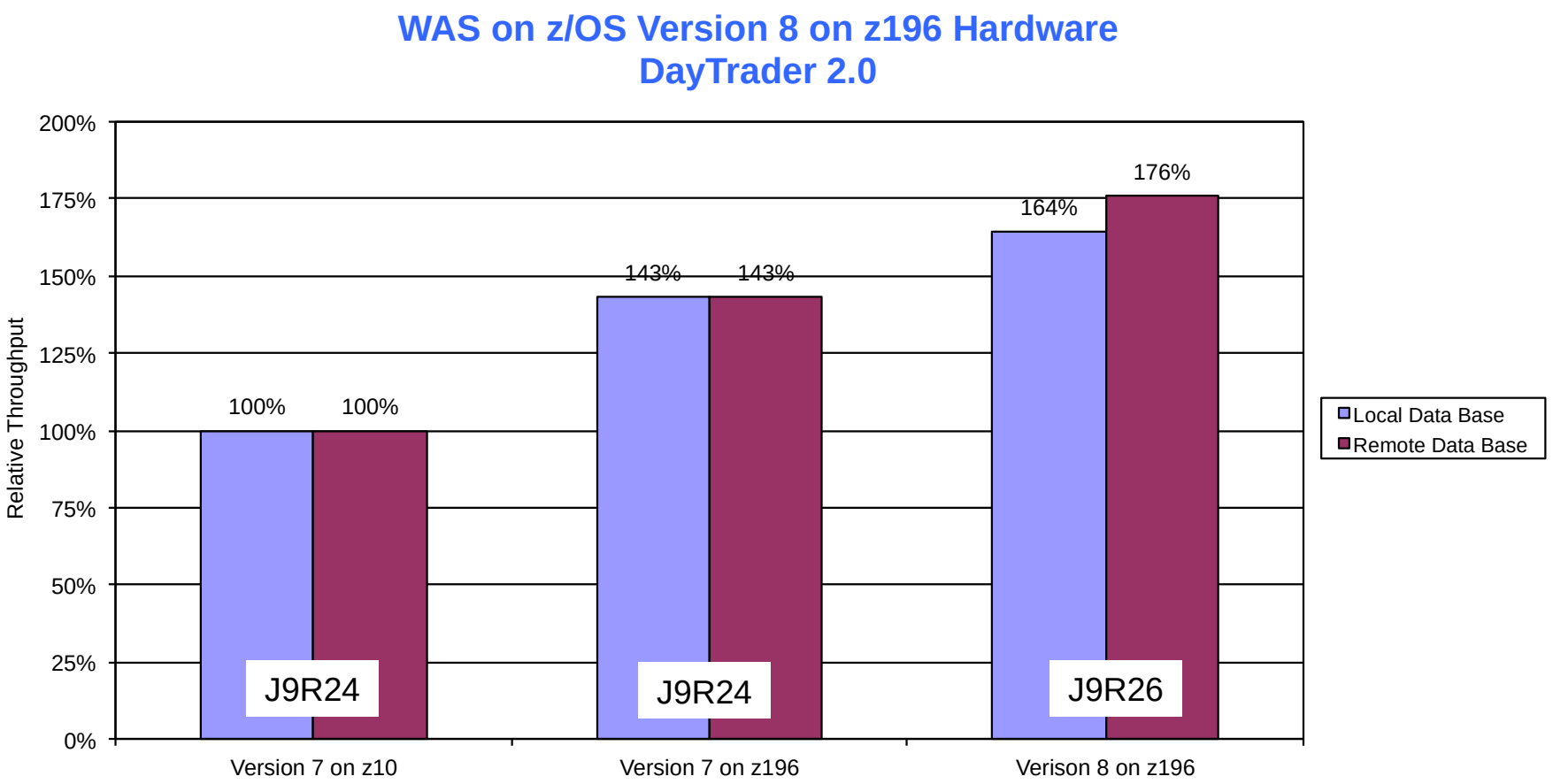
z196™ – z/OS V1.12

Comparing Java Throughput on CICS 4.2 with CICS 4.1



- CICS 4.2 JVMPOOL and JVMSERVER use 64-bit Java 6.0.1 relieving 31-bit storage constraint
- ~17% improvement to throughput with CICS 4.2/Java 6.0.1
- JVMSERVER slightly more expensive than JVMPOOL in CPU usage but requires less memory
- All configurations scale well

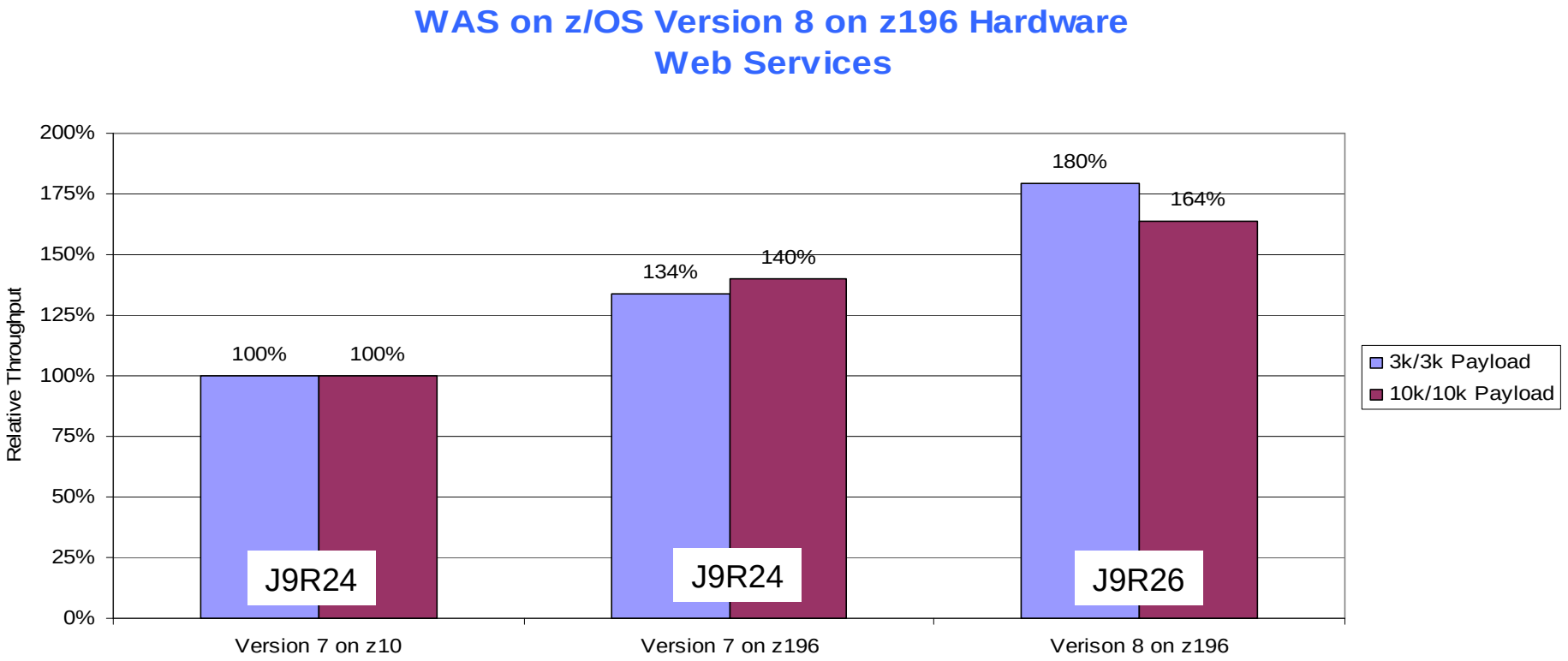
Performance on z/OS: WAS on z/OS



- z196 hardware measured 43% more throughput for local and remote database configurations
- Version 8 improved throughput of 2-tier configurations by another 15% for an aggregate benefit of 64%
- Version 8 improved throughput of 3-tier configurations by another 23% for an aggregate benefit of 76%

(Controlled measurement environment, results may vary)

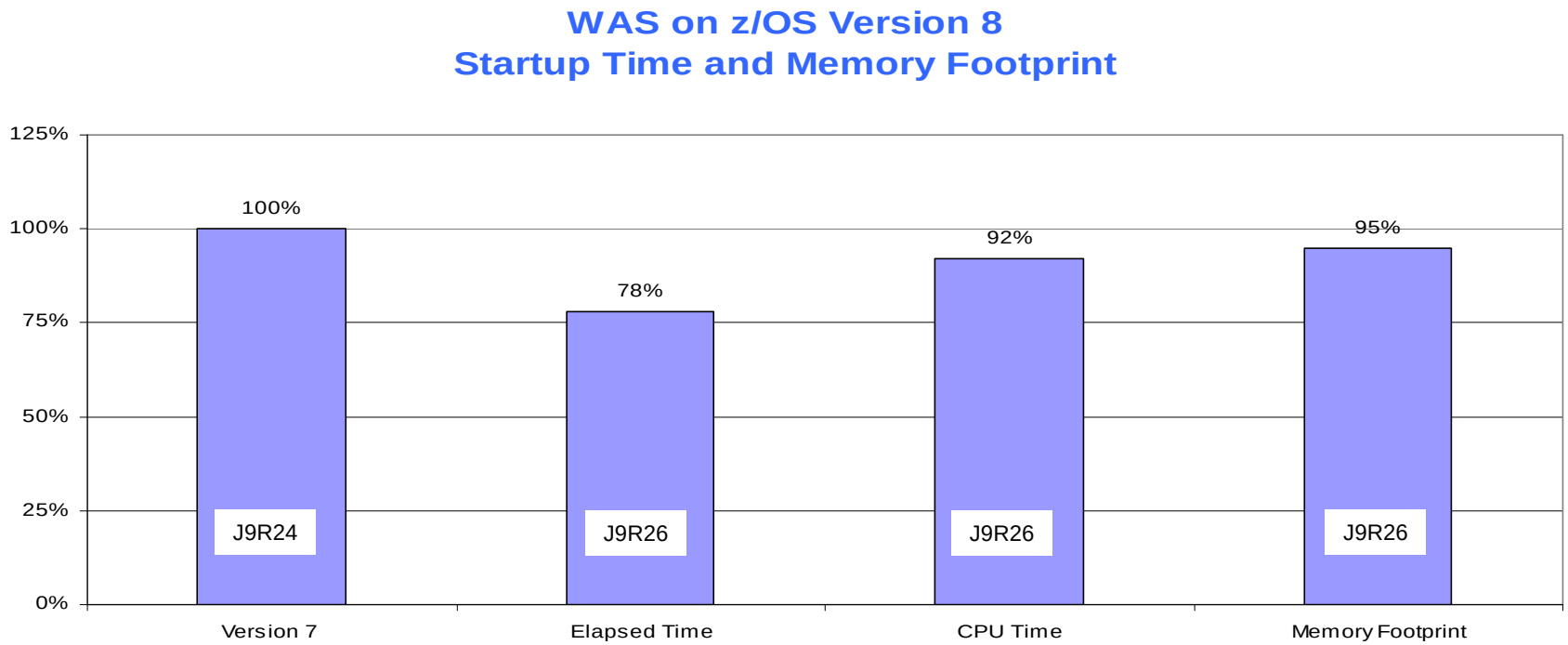
Performance on z/OS: WAS on z/OS



- z196 hardware measured 34% more throughput for small payload sizes (3kin/3kout) and 40% more for typical payload sizes (10kin/10kout) than z10 hardware
- Version 8 throughput improved over Version 7 by another 34% with the 3k/3k payload and 17% with the 10k/10k payload for and aggregate hardware and software benefit of +80% and +64% respectively
  - Improved JAXB parsing

(Controlled measurement environment, results may vary)

Performance on z/OS: WAS on z/OS

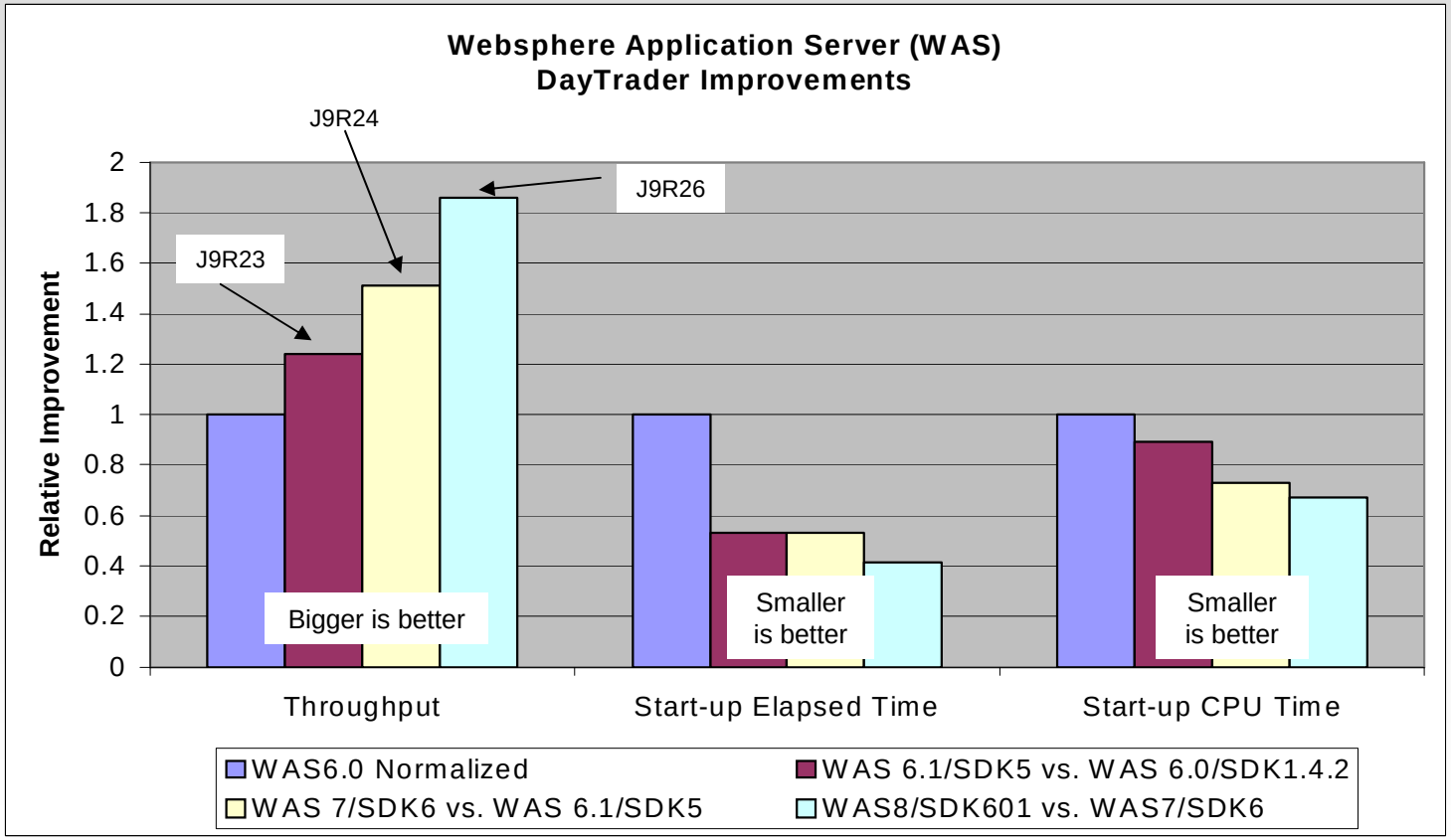


- Version 8 server startup time has been reduced by 22%, in elapsed time and 8% in CPU time compared to Version 7
  - Larger shared class cache reduced class load times
  - Optimized annotation scanning
  - Only delegate class loading to the JDK class loader instead of all class loaders for JDK classes
- Version 8 memory footprint has been reduced by 5%
  - Reductions in JVM native memory as well as class memory

(Controlled measurement environment, results may vary)

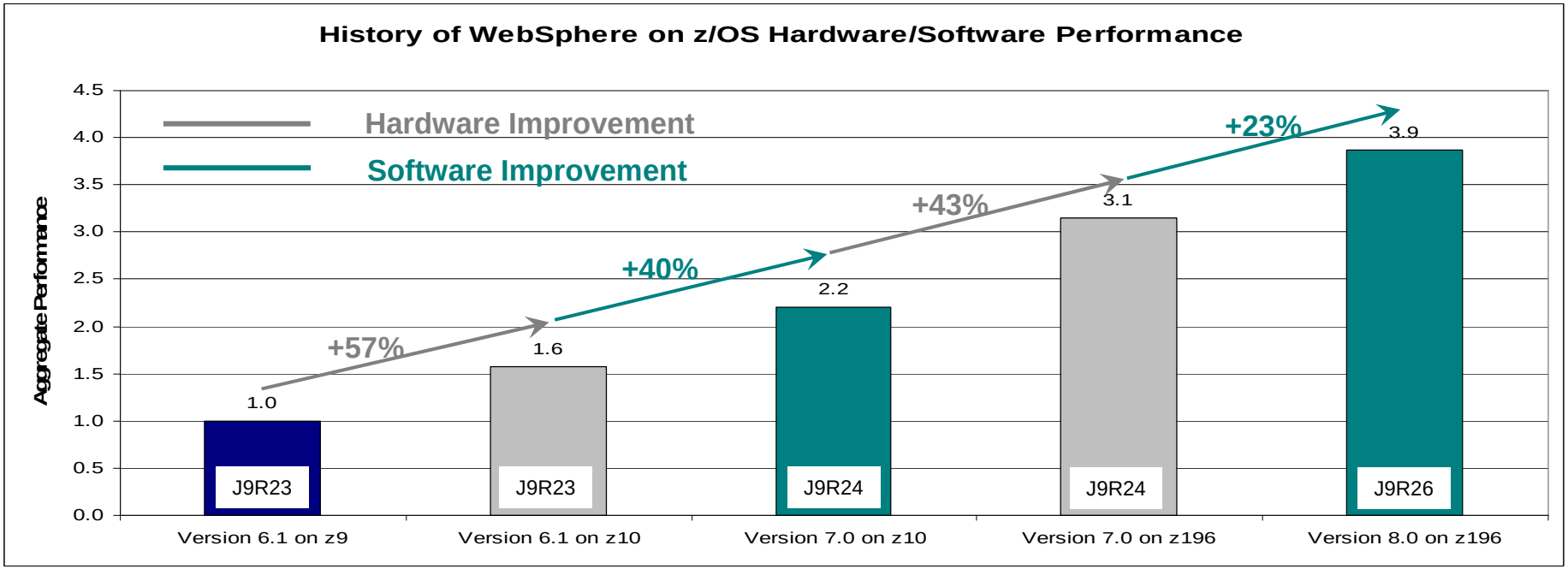
Performance – WAS on zOS

(WAS6.0 - WAS6.1 - WAS7.0 - WAS8.0 DayTrader on z/OS)



(Controlled measurement environment, results may vary)

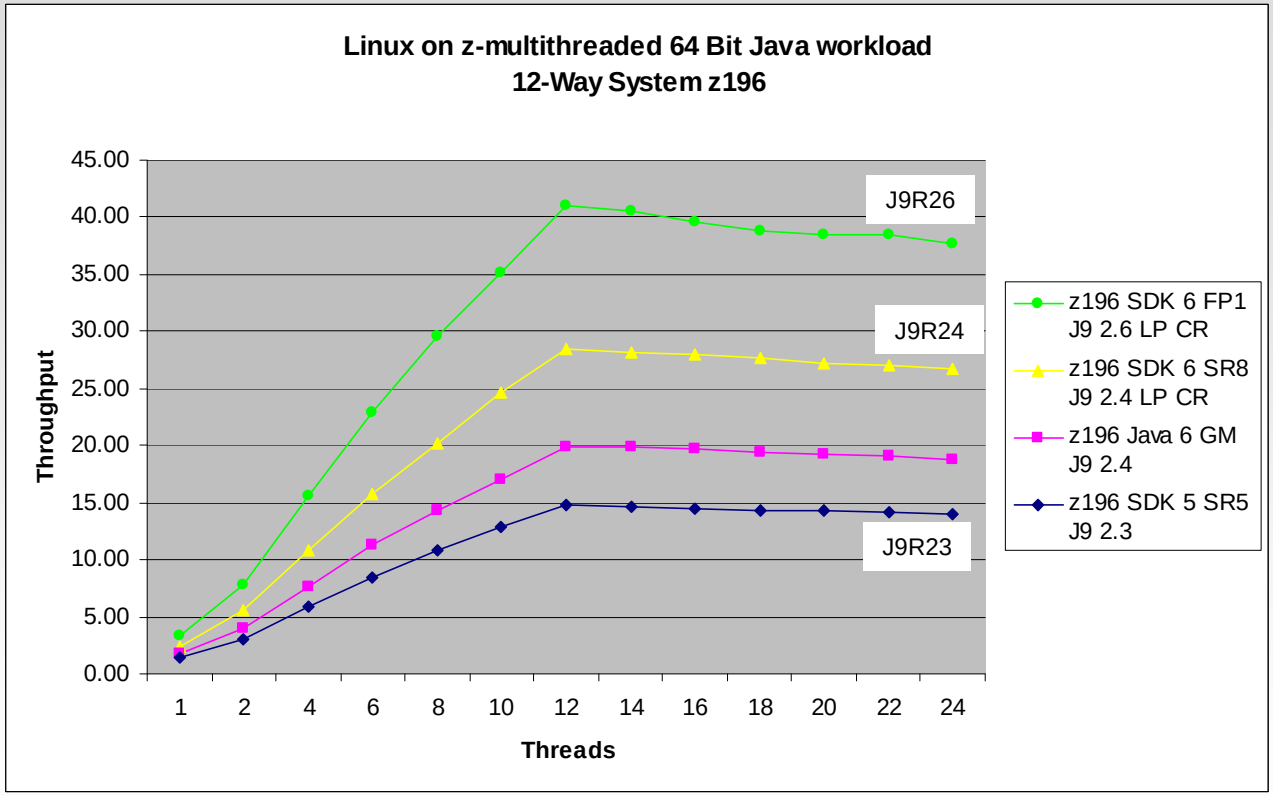
Performance on z/OS: WAS on z/OS



- This chart shows a history of improvements made from zSeries hardware (from z9 to z196) and software (from V6.1 to V8.0). The data is from measurements done using the DayTrader EJB workload.
  - The chart shows an aggregate performance improvement of almost 4x moving from WAS V6.1 on a z9 to WAS V8.0 on a z196.
  - The hardware component of this increase is about 2.25x (1.57 x 1.43)
  - The software component is about 1.72x (1.40 x .123)

(Controlled measurement environment, results may vary)

z/Linux Java6 R26 Performance: 64 Bit Multi-threaded Benchmark

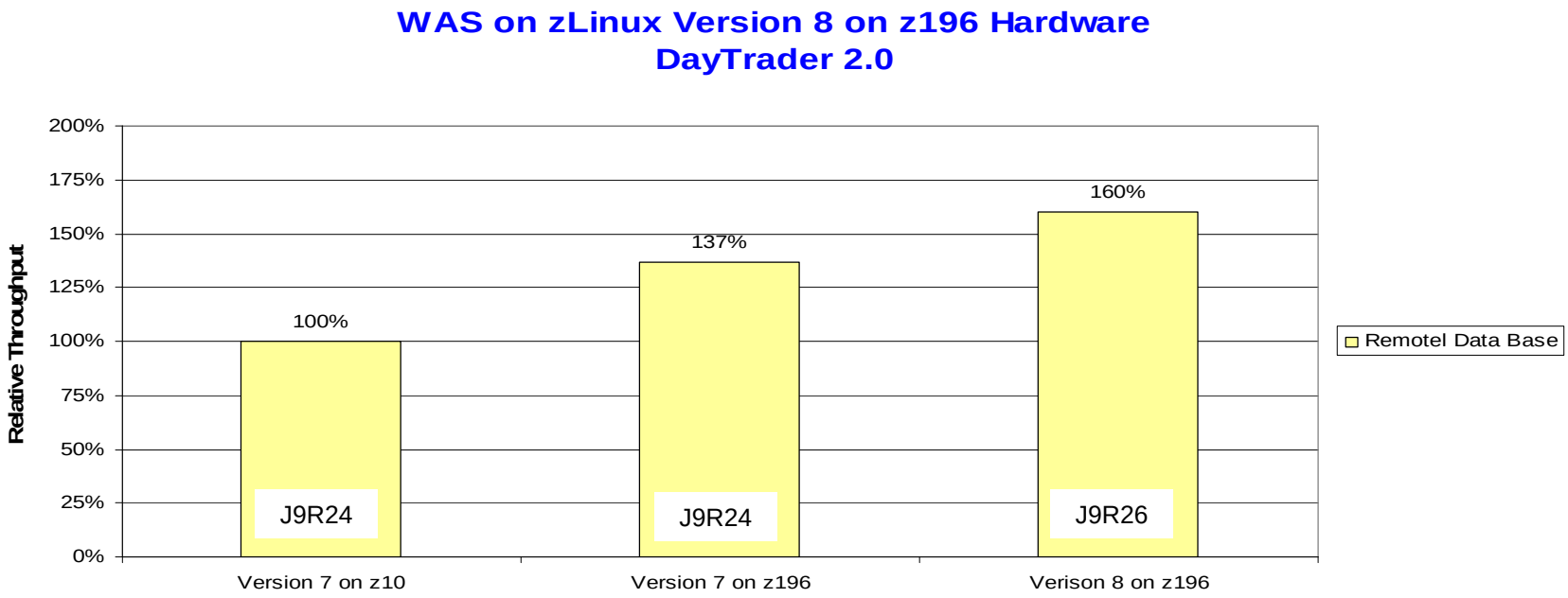


2.7x Aggregate Software improvement

- 42% Java6R26 vs Java6R24 improvement
- 42% Java6R24 SR9 vs Java6R24 GM
- 35% Java 6 GM versus Java 5 SR5

(Controlled measurement environment, results may vary)

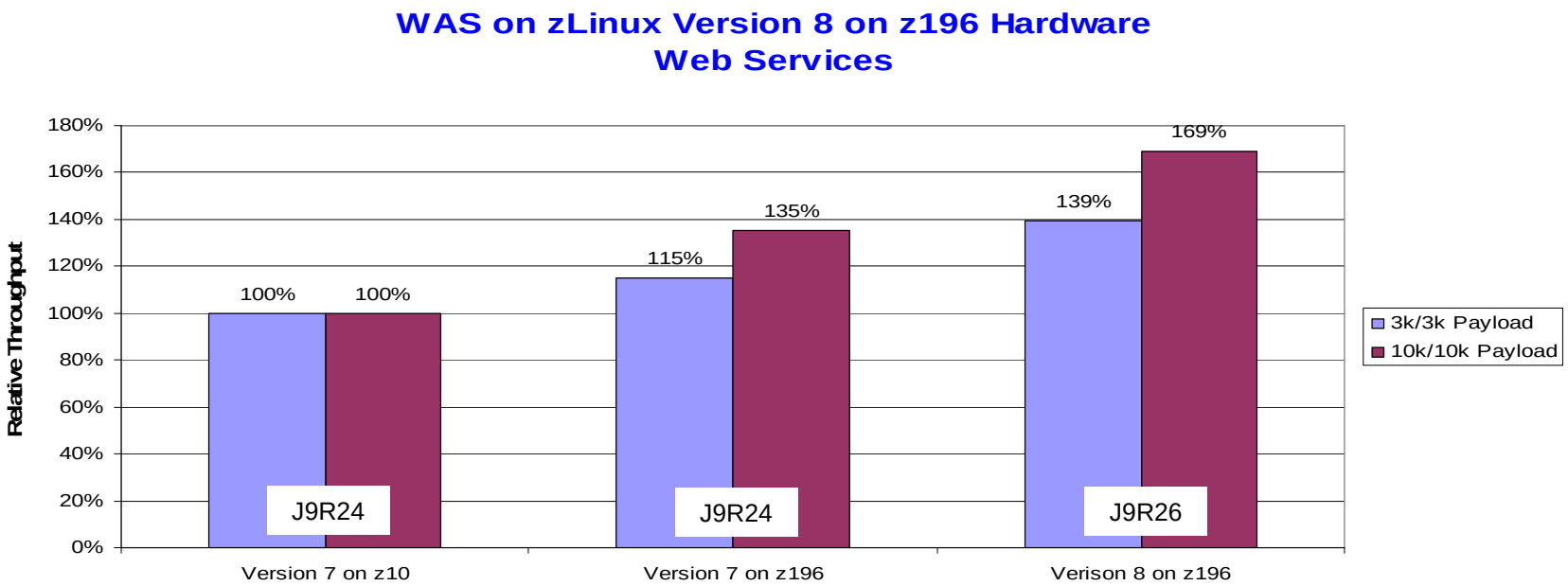
Performance – WAS8.0 on zLinux



- Upgrading from z10 to z196 improved throughput by 37% using our DayTrader 2.0 EJB benchmark.
- Additionally, upgrading to WAS V8.0 improved performance by another 17%. This increase is a result of improvements to the following areas:
  - JVM and JIT optimizations
  - OpenJPA code paths
- The combine hardware and software improvement is 60%.

(Controlled measurement environment, results may vary)

Performance – WAS8.0 on zLinux



- Upgrading from z10 to z196 improved throughput by as much as 35% using our SOABench webservices benchmark (15% for the 3k/3k payload and 35% for the 10k/10k payload).
- Additionally, upgrading to WAS V8.0 improved performance by another 21% for the 3k/3k payload and 25% for the 10k/10k payload. This increase is a result of improvements to the following areas:
  - JVM and JIT optimizations
  - JAXB fastpath optimizations
- The combine hardware and software improvement is 39% for the 3k/3k case and 69% in the 10k/10k case.

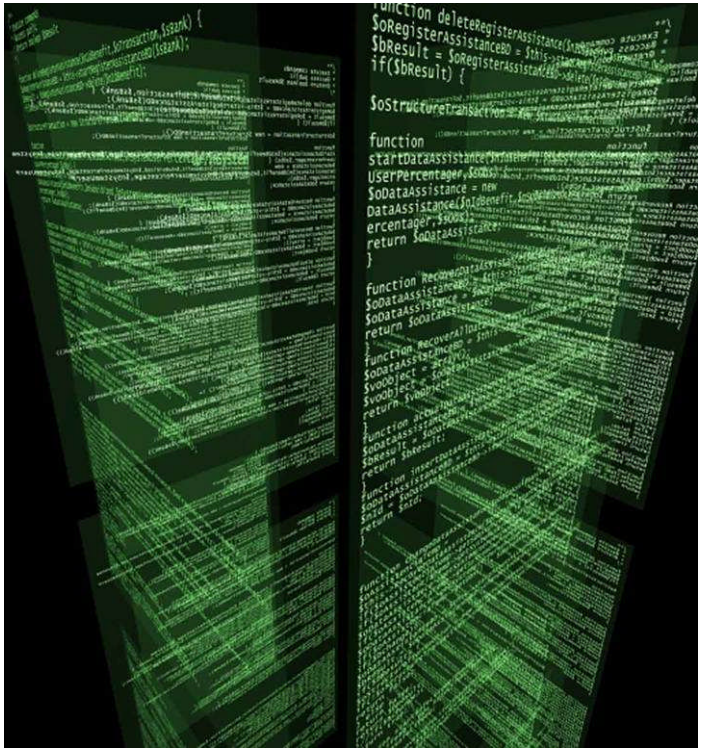
(Controlled measurement environment, results may vary)

IBM Java Consumability and Serviceability

R601 z/OS Unique functional enhancements

JZOS 2.4.0

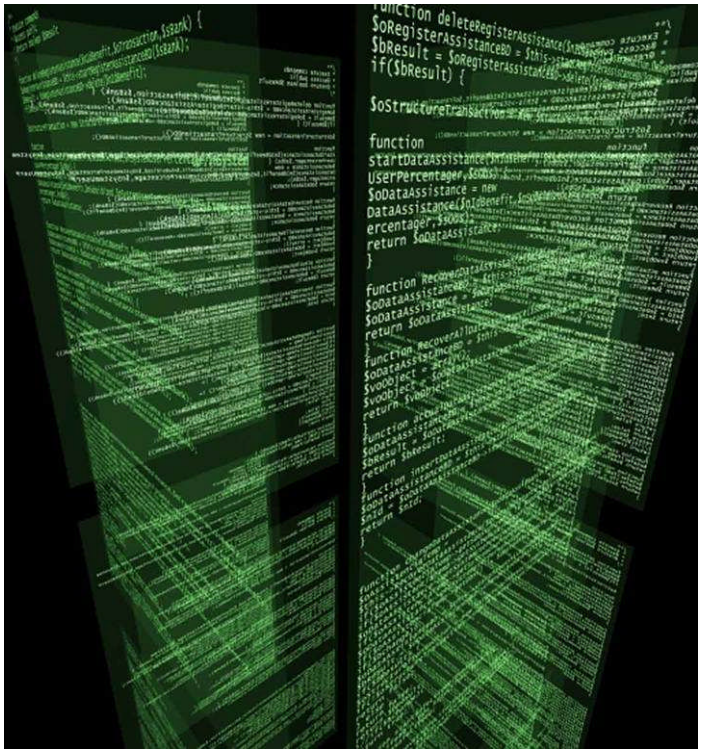
- Enables the submission of jobs to the MVS internal reader and retrieval of the submitted JOB ID
  - Text, record, and binary JCL will be supported.
  - The MVS internal reader attributes LRECL, RECFM, and Class will be configurable from this Java class.
- Enhances the zlogstream class to support
  - IXGBRWSE (read) and IXGDELT (delete)
  - InputStream/OutputStream Java wrappers
- Added the method ZFile.readDSCBChain() to support
  - Reading all of the DSCBs associated with a dsn/volume
  - Extended access volumes (Format-8 and Format-9 DSCBs.)
- Adds a new package com.ibm.jzos.wlm with selected z/OS Workload Manager (WLM) APIs
- Removes all JRIO code dependencies
  - JRIO is deprecated in z/OS V6.0.1 products



R601 z/OS Unique functional enhancements

z/OS Java Unique Security Enhancements

- IBMJCECCA provider support for AES Secure Keys
- RAS: Provide Enhanced ICSF Exception Handling in IBMJCECCA



IBM J9 2.6 Technology Enhancements:
Garbage Collection: Balanced Policy

Improved responsiveness in application behavior

- Reduced maximum pause times to achieve more consistent behavior
- Incremental result-based heap collection targets best ROI areas of the heap
- Native memory aware approach reduces non-object heap consumption

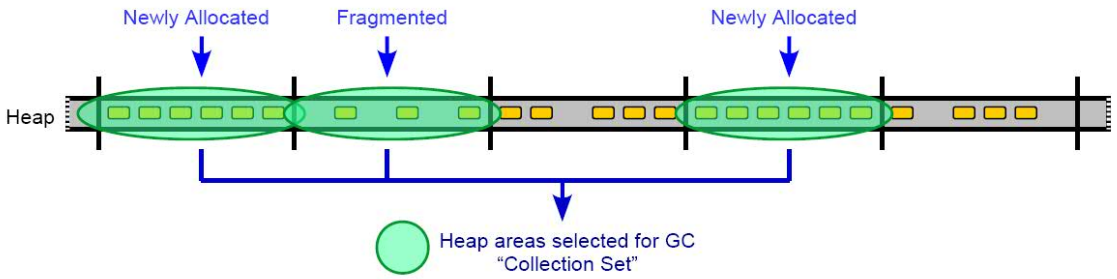
Next generation technology expands platform exploitation possibilities

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

Recommended deployment scenarios

- Large (>4GB) heaps
- Frequent global garbage collections
- Excessive time spent in global compaction
- Relatively frequent allocation of large (>1MB) arrays

Input welcome: Help set directions by telling us your needs



IBM J9 2.6 Technology Enhancements:
GC Policy Changes

Effective with Java R601

- Xgcpolicy: gencon is the default GC Policy
- Applies to both 31- and 64-bit JVMs

Benefits

- Long-lived objects are handled differently than short-lived objects
- Applications with many short-lived objects can see shorter pause times (better performance) while maintaining good throughput

Considerations

- Assess performance against current gcpolicy in use
- Consider returning to previous gcpolicy if needed

Other changes

- The subpool policy has been removed
- -Xgcpolicy:subpool is now an alias for -Xgcpolicy:optthruput

IBM J9 2.6 Technology Enhancements:
GenCon Policy Tuning

Gencon Defaults

- Default split is 25% newspace, 75% oldspace
- Both new and old space can grow and shrink dynamically

Old and New space size is definable

- -Xmns / -Xmnx
- -Xmos / -Xmox

Tuning Considerations

- In a fixed heap (ie -Xms = -Xmx), both areas are fixed in size
- Too small a nursery can cause
  too many local GCs
  premature tenuring >> more frequent Global GCs
- Too large a nursery can cause
  expensive local GCs
  smaller tenure space >> more frequent Global Gcs
- The tuning goal is to minimize the object survival rate
- If the rate is consistently too high, consider another policy
- Consider using the GCMV tool to help

IBM J9 2.6 Technology Enhancements:
RAS Enhancements

JVM Dump Support

- Environment Variables and ULIMITs included in javacores
- Native memory usage counters in javacore and from core dumps via DTFJ
- Multi-part TDUMPs on zOS 64

JVM Trace Support Improvements

- -Xtrace

  More internal tracepoints created
  New javastack and ceedump trigger points added

JVM Logging Improvements

- -Xlog

New with J9 2.6 all JVM error messages (JVMxxxxnnnE) plus specific information messages JVMDUMP006I, JVMDUMP032I and JVMDUMP033I are written to the system log by default.
The –Xlog option has been enhanced to provide controlled selection of Message routing to the system log.

IBM J9 2.6 Technology Enhancements:
RAS Enhancements: javacore contents

Environment Variables

--

_CXX_WORK_SPACE=(32000,(150,150))

_CXX_PMSGS=EDCPMSGE

MAIL=/usr/mail/CHAMBER

_CC_CNAME=CCNDRVR

PATH=/u/sovbld/bldsys:/usr/local/perl/bin:/u/java/bin:/bin:/usr/sbin:/u/chamb....

_C89_WORK_SPACE=(32000,(150,150))

_CXX_WORK_UNIT=SYSDA

_CXX_INCDIRS=/usr/include //DD:SYSLIB //'PP.ADLE370.ZOS180.SCEEH.NET.H'……

_C89_PNAME=EDCPRLK

TMPDIR=/tmp

SSH_CLIENT=9.20.183.84 1846 22

SHELL=/bin/sh

IBM J9 2.6 Technology Enhancements:
RAS Enhancements: javacore contents

ULIMITs

User Limits (in bytes except for NOFILE and NPROC)

---

type	soft limit	hard limit
RLIMIT_AS	unlimited	unlimited
RLIMIT_CORE	0	unlimited
RLIMIT_CPU	unlimited	unlimited
RLIMIT_DATA	unlimited	unlimited
RLIMIT_FSIZE	unlimited	unlimited
RLIMIT_LOCKS	unlimited	unlimited
RLIMIT_MEMLOCK	32768	32768
RLIMIT_NOFILE	1024	1024
RLIMIT_NPROC	16382	16382
RLIMIT_RSS	unlimited	unlimited
RLIMIT_STACK	10485760	unlimited
RLIMIT_MSGQUEUE	819200	819200

IBM J9 2.6 Technology Enhancements:
RAS Enhancements: javacore contents

- 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
```

IBM J9 2.6 Technology Enhancements:
RAS Enhancements: tdump collection

- Multi-part TDUMPs
  - targeted for the 64bit JVM user
  - Existed in Java 6 also, so this should be a refresher

If you specify a template for the IEATDUMP file name, append the &DS token to enable multiple dumps. The &DS token is replaced by an ordered sequence number, and must be at the end of the file name. For example, X&DS generates file names in the form X001, X002, and X003.

 If you specify a template without the &DS token, .X&DS is appended automatically to the end of your template by the JVM. If your template is too long to append .X&DS, a message is issued. The message advises that the template pattern is too long and that a default pattern will be used.

 If you do not specify a template, the default template is used. The default template is: %uid.JVM.%job.D%y%m%d.T%H%M%S.X&DS

***Dump files MUST be merged before use with IPCS or JDmpview

IBM J9 2.6 Technology Enhancements:
RAS Enhancements: -Xlog

- Using –Xlog to control system log messages

-Xlog[:help]|[:<options>]
Sub-options are:
error log all error (JVMxxxnnnE) messages (default)
warn log all warning (JVMxxxnnnW) messages
info log all information (JVMxxxnnnI) messages
config log all configuration messages (there are none yet!)
vital log VM-selected messages, eg for location of dump files (default)
all log all messages
none turn off logging

Examples
To log error and warning messages
-Xlog:error,warn.
To turn off logging
-Xlog:none.

IBM J9 2.6 Technology Enhancements:
Verbosegc Updates

Verbosegc Format has Changed

- Still XML, but built on a different model
- Old format was transactional and nested
  - *Worked well for optthruput*
  - *Not so well for optavgpause and gencon*
- New format is event driven and flatter
  - *Provides a better fit for concurrent GC work*

Added BONUS! Verbose GC output now contains a listing of arguments passed to the JVM

- Helpful for diagnosing OOMs, long GC cycles, etc
- Helps when a matching javacore isn't available

Verbosegc: Old Format

- Pre R601 trace entry:

```
<con event="kickoff" timestamp="Mar 22 10:38:49 2011">
<kickoff reason="Kickoff threshold reached" />
</con>
<af type="nursery" id="16" timestamp="Mar 22 10:38:49 2011" intervalms="21.127">
<gc type="scavenger" id="17" totalid="19" intervalms="21.165">
</gc>
<time totalms="6.952" />
</af>
<con event="collection" id="1" timestamp="Mar 22 10:38:49 2011" intervalms="1526.710">
<gc type="global" id="2" totalid="19" intervalms="660.545">
<timesms mark="1.410" sweep="0.437" compact="0.000" total="4.002" />
</gc>
<time totalms="4.080" />
</con>
```

Verbosegc: New Format

- R601 trace entry:

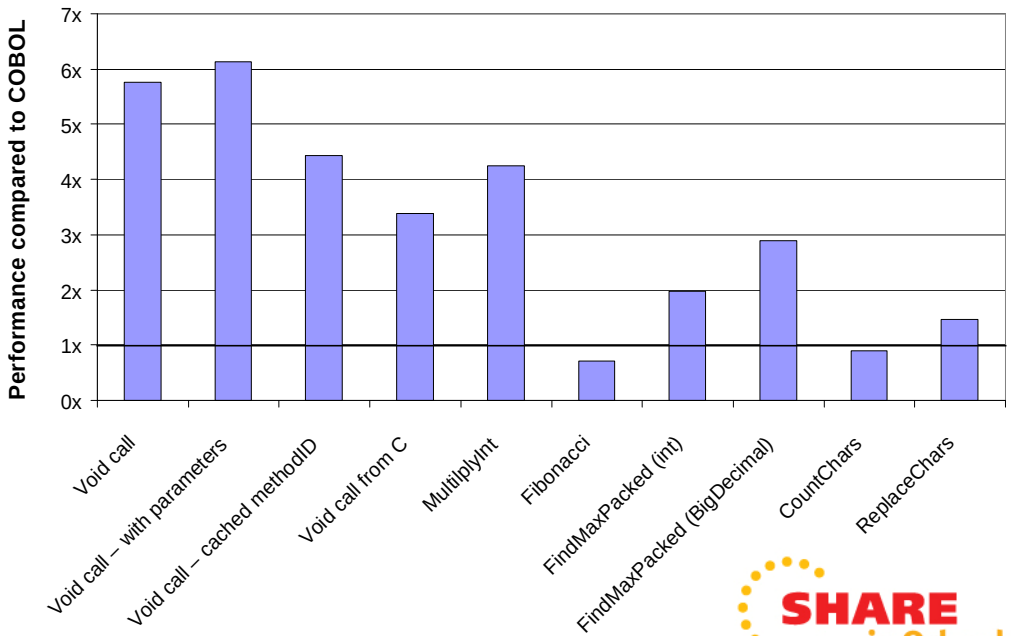
```
<concurrent-kickoff id="235" timestamp="2011-03-22T10:40:51.701">
<kickoff reason="threshold reached" targetBytes="933651" thresholdFreeBytes="127933" />
</concurrent-kickoff>
<af-start id="240" totalBytesRequested="65544" timestamp="2011-03-22T10:40:51.705" intervalms="23.097" />
<gc-start id="242" type="scavenge" contextid="241" timestamp="2011-03-22T10:40:51.705">
<gc-op id="244" type="scavenge" timems="3.990" contextid="241" timestamp="2011-03-22T10:40:51.709">
<gc-end id="246" type="scavenge" contextid="241" durationms="4.161" timestamp="2011-03-22T10:40:51.709">
<af-end id="250" timestamp="2011-03-22T10:40:51.709" />
<concurrent-collection-start id="253" timestamp="2011-03-22T10:40:51.719" intervalms="1337.041" />
<gc-start id="254" type="global" contextid="237" timestamp="2011-03-22T10:40:51.719">
<gc-op id="258" type="mark" timems="1.433" contextid="237" timestamp="2011-03-22T10:40:51.722">
<gc-op id="259" type="sweep" timems="0.394" contextid="237" timestamp="2011-03-22T10:40:51.722" />
<gc-end id="260" type="global" contextid="237" durationms="3.277" timestamp="2011-03-22T10:40:51.723">
<concurrent-collection-end id="263" timestamp="2011-03-22T10:40:51.723" />
```

Inter-language Communication: Signal Handling with -XCEEHDLR (31-bit z/OS only)

- New feature in Java 6.0.1
  - Requested by Java/COBOL interoperability batch mode environment
- Switches JVM from POSIX to LE Signal Handling for
  - SIGBUS
  - SIGFPE
  - SIGILL
  - SIGSEGV
  - SIGTRAP
- A condition triggered while executing a JNI component causes the JVM to convert the Language Environment condition into a Java ConditionException
  - Allows Java application to see/catch LE conditions
  - com.ibm.le.conditionhandling.ConditionException exception is thrown

Inter-language Communication
Java/COBOL Inter-Operability Performance

- COBOL → Java compared to COBOL → COBOL
  - Java void() method shows 6x more overhead
  - Caching methodID reduces overhead to 4.4x
  - Max operation on a set of packed decimals:
    - 2x slowdown when Java transformed decimals into ints
    - 2.9x slowdown when Java transformed decimals into BigDecimals
  - Fibonacci (42 adds in a loop) shows Java performs 40% better than COBOL
- +96% of the program is eligible for zAAP offload
- Best practices:
  - Do as much work in Java as possible
  - Have as few COBOL ↔ Java transitions as possible



Inter-language Communication: JNI Best Practices

- Common performance pitfalls
  1. Not caching method IDs, field IDs, and classes
     - Avoid redundent calls to `FindClass(), GetFieldID(), GetMethodId(), and GetStaticMethodID()`
  2. Triggering array copies
     - Assume arrays are buffered, hence be precise about which elements you really need to avoid needless copying
  3. Reaching back instead of passing parameters
     - When possible, flatten object fields into parameters of call. Avoid using JNI services to get to object fields
  4. Choosing the wrong boundary between native and Java code
     - Assume Native ⇔ Java call overhead 10x slower than Native ⇔ Native or Java ⇔ Java
  5. Using many local references without informing the JVM

See http://www.ibm.com/developerworks/java/library/j-jni/

Other changes of interest

- Service has requested a TDUMP be captured by default on OutOfMemory exceptions.

- Jmpview no longer requires TDUMPs to be pre-processed by jextract.
  - This eliminates the requirement to use a jvm build level matching that captured within the dump.
  - Support has been backported to more current versions of Java6 and Java5, but requires using an R601 build level to execute jdmpview. Support exists beginning with:
    - Java6 SR9
    - Java5 SR12

Documentation

Download a copy of the

" IBM SDK Java Technology Edition Version 6 Supplement" for more details:

http://public.dhe.ibm.com/common/ssi/ecm/en/zsl03118usen/ZSL03118USEN.PDF

R601 31-bit:
http://www-03.ibm.com/systems/z/os/zos/tools/java/products/sdk601_31.html

R601 64-bit:
http://www-03.ibm.com/systems/z/os/zos/tools/java/products/sdk601_64.html

SHARE
Technology · Connections · Results

SHARE in Orlando
2011