Java Technology Centre

Reading the runes for Java runtimes

The latest IBM Java SDKs ... and beyond

Tim Ellison
tim_ellison@uk.ibm.com

Goals

- **IBM and Java**
- **Explore the changing landscape of hardware and software influences**
- **Discuss the impact to Java runtime technology due to these changes**
- **Show how IBM is leading the way with these changes**

IBM and Java

- **Java is critically important to IBM**
  - Provides fundamental infrastructure to IBM software portfolio
  - Delivers standard development environment
  - Enables cost effective multi platform support
  - Delivered to Independent Software Vendors supporting IBM server platforms

- **IBM is investing strategically in virtual machine technology**
  - Since Java 5.0, a single Java platform technology supports ME, SE and EE
  - Technology base on which to delivery improved performance, reliability and serviceability
    - Some IBM owned code (Virtual machine, JIT compiler, ...)
    - Some open source code (Apache XML parser, Apache Core libraries, Zlib, ...)
    - Some Sun licensed code (class libraries, tools, ...)

- **Looking to engender accelerated and open innovation in runtime technologies**
  - Support for Eclipse, Apache (Harmony, XML, Derby, Geronimo, Tuscany)
  - Broad participation of relevant standards bodies such as JCP and OSGi

IBM Java – 2009 key initiatives

- **Consumability**
  - Deliver value without complexity.
  - Ensure that problems with our products can be addressed quickly, allowing customers to keep focus on their own business issues.
  - Deliver a consistent model for solving customer problems.
- **“Scaling Up” - Emerging hardware and applications**
  - Provide a Java implementation that can scale to the most demanding application needs.
  - Exploit emerging hardware and software platforms while delivering industry leading performance.
- **“Scaling Down”**
  - A radically improved runtime in the areas of startup performance, memory footprint, configurability, isolation and security.
  - Provide a scalable agile programming platform.
- **Deterministic quality of service for the solution stack**
  - High throughput and guaranteed predictable performance for the full spectrum of real-time systems.
- **Open innovation**
  - Accelerate rate and pace of innovation in the Java community through open source efforts such as Apache Harmony and Eclipse

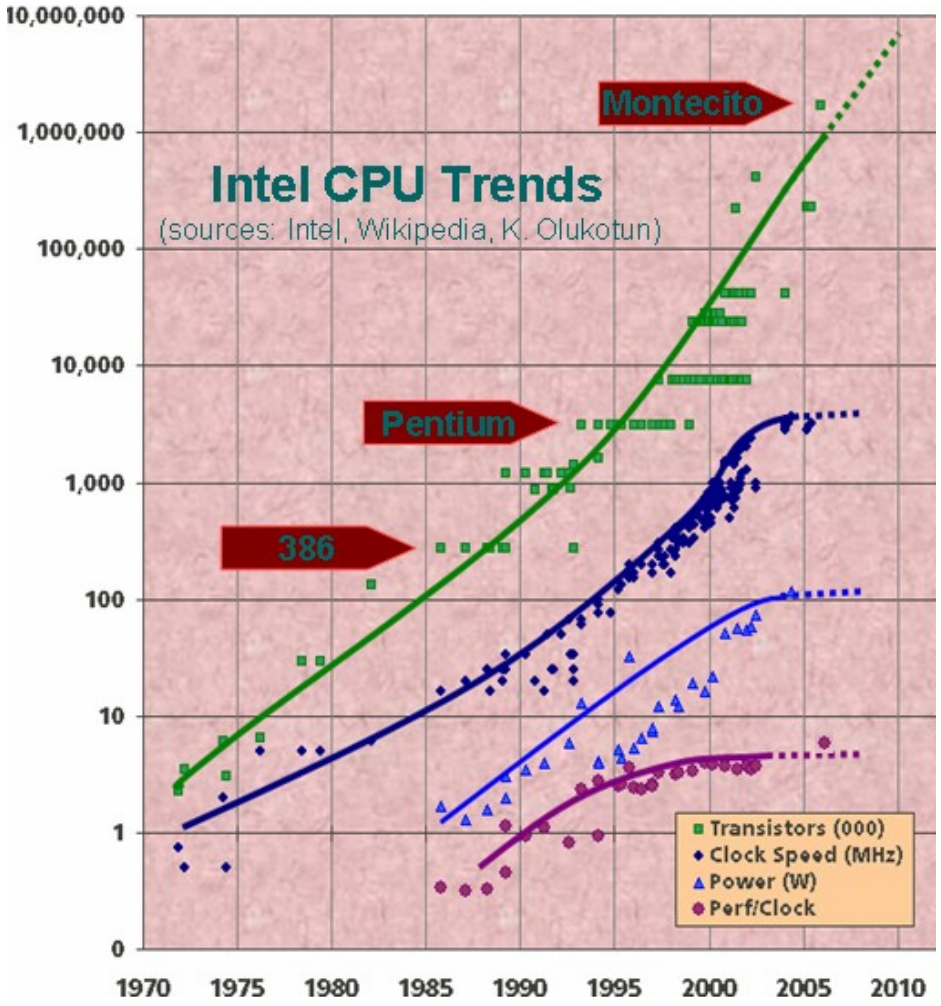
IBM Java technology – leading progress through innovation

- **Through Java specifications**
  - Java 5.0
    - New Language features: autoboxing, enumerated types, generics, metadata, ...
  - Java 6.0
    - Performance Improvements, client web services support, ...
  - Java 7.0
    - Not yet defined, but likely
    - Modularity, support for dynamic languages, improve ease of use for SWING, new IO APIs (NIO2), Java persistence API, JMX 2.0 and WS connection for JMX agents, language changes, ...

- **Areas of IBM key focus areas**
  - Consumability
  - Scaling up
  - Scaling down
  - Deterministic quality of service
  - Open innovation

- **Exploitation of advances in hardware and software**

Hardware – processor trends



- **Transistor density**
  - increasing exponentially
- **Clock speed, Power consumption, Performance / clock speed**
  - leveling off

Source: Burton Smith, “Reinventing Computing”, Microsoft Manycore Workshop 2007

Hardware – increasing number of cores

IBM Power6 Systems

	BladeCenter® JS12 Express	BladeCenter JS22 Express	Power® 520 Express	Power 550 Express	Power 560 Express	Power 570	Power 575	Power 595
# of cores (GHz)	2 (3.8)	4 (4.0)	1, 2, 4[1] (4.2)	2, 4[1], 6, 8 (3.5, 4.2[2])	4, 8, 16 (3.6)	2 to 16 (3.5, 4.2, 4.4, 4.7, 5.0) 4 to 32 (4.2)	32 per node (4.7)	8 to 64 (4.2) 16 to 64 (5.0)

Challenges of multi-core / many-core trends

- **Multi-core selling point: “2x cores == 2x performance”**
  - Not strictly true, coordination of multiple CPUs has overhead
  - Scaling to low count multi-core has been fairly successful
  - Existing single-threaded applications gain some benefit from multi-core aware dynamic runtimes

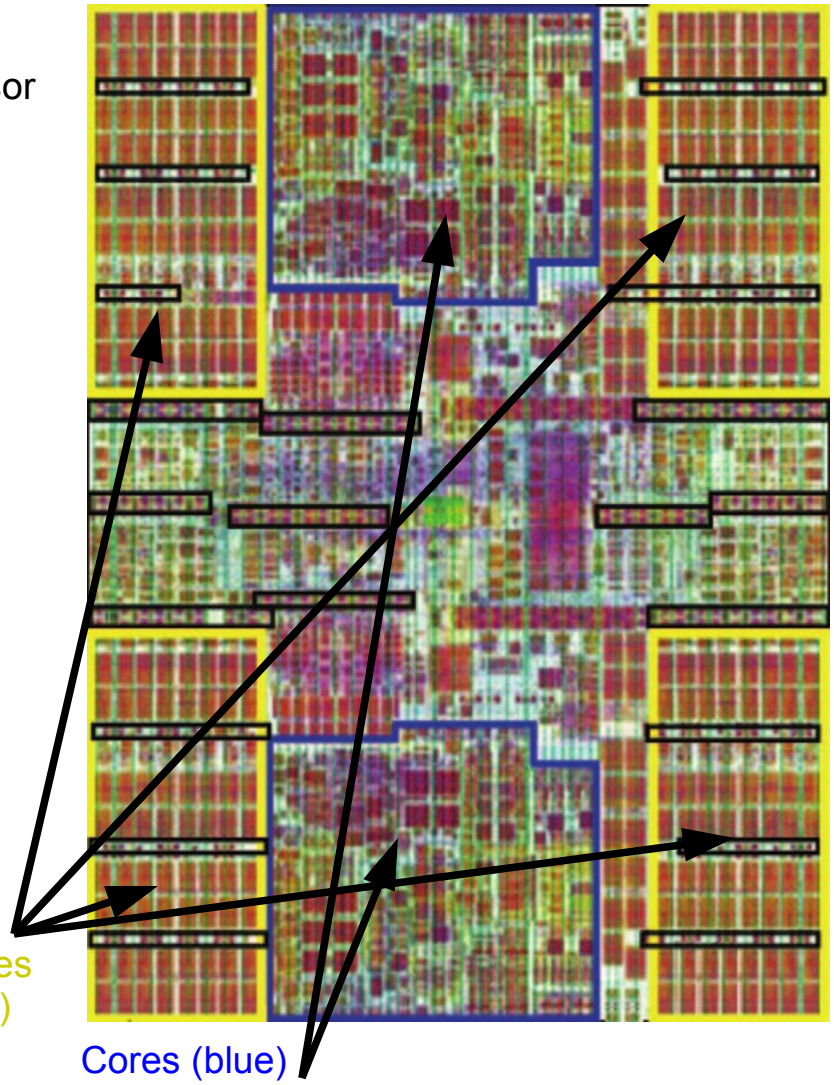
- **Many-core (8+ cores) requires smarter software**
  - New programming techniques required to deliver scale effectively to application programmers
  - High count multi-core can mean higher software overhead
  - Potentially asymmetric designs such as Cell (1 PPE, 8 SPEs), GPUs
  - Traditional managed runtimes often not well placed to exploit many-core architectures

- **Java runtimes need to evolve**
  - Support for concurrency starts from the bottom up
  - If the software you build on does not exploit concurrency...

IBM zSeries

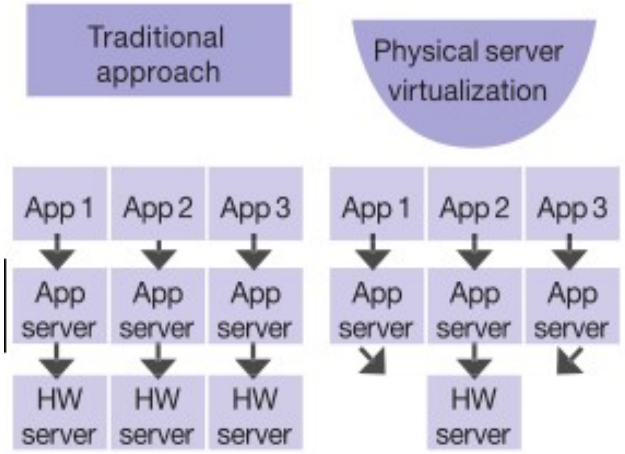
Hardware – memory

- **Non-Uniform Memory Access (NUMA)**
  - Access time depends on location relative to processor
  - Caches getting smaller, but faster
    - L1 cache 64KB reachable in 4 cycles
    - L2 cache 4MB reachable in 24 cycles
    - L3 cache 32MB reachable in 160 cycles
  - Data locality can have significant performance implications
    - Commercial workloads measured spending 45% of time on stalled memory requests
- **Applications can hit a "Memory Wall"**
  - Relative distance between main memory and CPU increasing (1000x)
  - Processor time cheap, memory access expensive
- **Managed runtimes control object placement**
  - Design of standard libraries and code generation need to be tuned to achieve data locality
  - Application programmers have few tools to help with their object placement



Secondary effects of multi-core / many-core

- **Machine consolidation**
  - Data centres increasing number of applications run in parallel on single physical machines
  - Reduce costs of administration, utilities (cooling, water, electricity), real estate, ...
- **Virtualization**
  - To the runtime, these may look like regular machines, but have different characteristics
  - How can managed runtimes contribute to the savings potential?
- **Simply running multiple copies of your application?**
  - No concept of shared code libraries or shared data
  - Isolation required between multiple applications in same runtime



- **Java heap size soon becomes a limiting factor**
  - Running multiple apps and app servers on a single machine often infeasible
  - Apps hit the memory wall waiting for heap accesses

public class **OutOfMemoryError**
extends VirtualMachineError

Thrown when the Java Virtual Machine cannot allocate an object because it is out of memory, and no more memory could be made available by the garbage collector.

Hardware – main memory growth

- **Main memory getting larger**
  - 64-bit computing common place
  - Java heaps allow for large number of individually addressable objects
- **Flash memory drives appearing**
  - Blurring distinction between main memory and disk
  - e.g. multiple 128Gb solid state drives (SSDs), low latency, fast read, slow write, low power consumption compared to HDD
- **Applications regularly holding large databases in memory**
  - Aim to increase performance, but lots of read-only objects cluttering heap
  - e.g. over 90% of (4G, 10Gig, 100Gig+) Java heap allocated is database

POWER6 BUILT ON Power

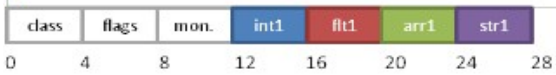
	BladeCenter® JS12 Express	BladeCenter JS22 Express	Power® 520 Express	Power 550 Express	Power 560 Express	Power 570	Power 575	Power 595
Min - max. memory	2 – 64 GB	4 – 32 GB	1 – 64 GB (max 16 per core)	1 – 256 GB (max 64 per proc card)	8 – 384 GB (max 96 per proc card)	2 – 768 GB (max 96 per proc card)	32 – 256 GB per node	16 – 4096 GB (max 512 per proc book)

Challenges of main memory growth trends

- **Garbage collection algorithms**
  - Designed with multi-megabyte heaps in mind
  - Pause times going into minutes
  - New approach required for today's 100's Gigabyte heaps

```
public class Foo32 {

   private int int1;          //4 byte primitive
   private float flt1;        //4 byte primitive
   private ArrayList arr1;    //4 byte refreference
   private String str1;       //4 byte refreference
}
```




```
public class Foo64 {

   private int int1;          //4 byte primitive
   private float flt1;        //4 byte primitive
   private ArrayList arr1;    //8 byte refreference
   private String str1;       //8 byte refreference
}
```

- **Object addressing with long long pointers**
  - Referencing objects with 64-bits causes “object bloat”
  - Decreased performance through sparse data and increased garbage collection frequency

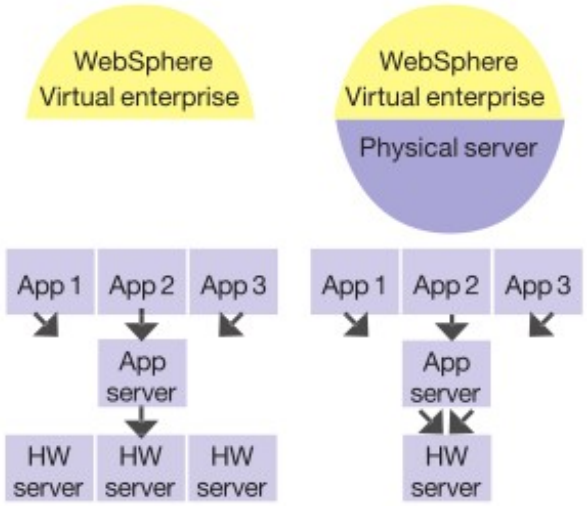
- **Java heap is flat**
  - Programming model does not distinguish read-only and read-write objects
  - No way to give hints to locality (other than usage patterns)

IBM Java solutions – optimizing for multi-core

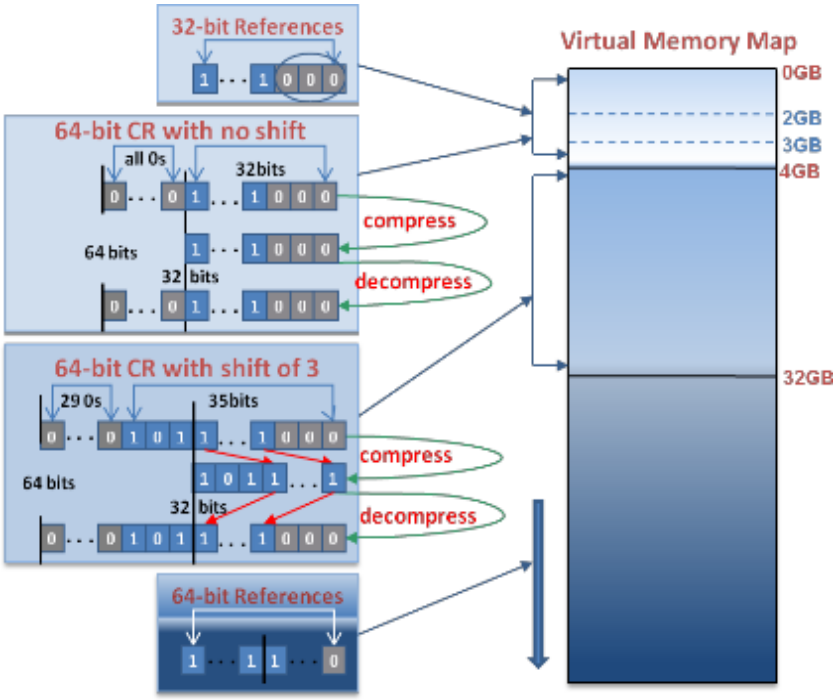
- **Implementation parallelism**
  - Runtime uses multiple helper threads for JIT, GC, etc.
  - `synchronized` lock acquisition highly optimized
  - Automatic application deadlock detection
  - IBM Lock Analyser for visualizing lock dynamics

- **Application level parallelism is still hard**
  - Parallel programming idioms recognized by the JIT
    - Use standard libraries e.g. JSR166 Concurrency Utilities
    - Code for clarity before performance
  - Research language “X10” adds parallel constructs to Java

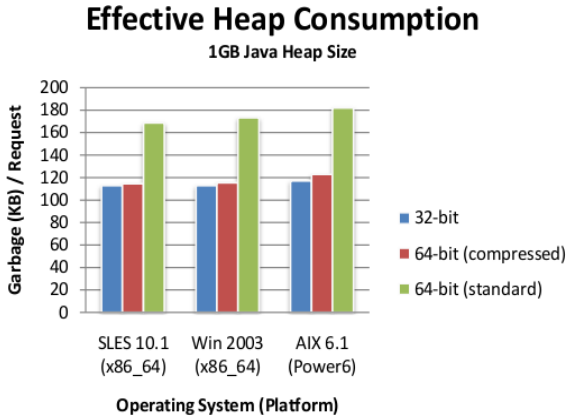
- **WebSphere Virtual enterprise**
  - Spread workloads across a pool of app servers
  - Spread app servers across pool of hardware
  - Seemless to the end user
  - Optimize utilization and quality of service



IBM Java solutions – optimizing memory usage



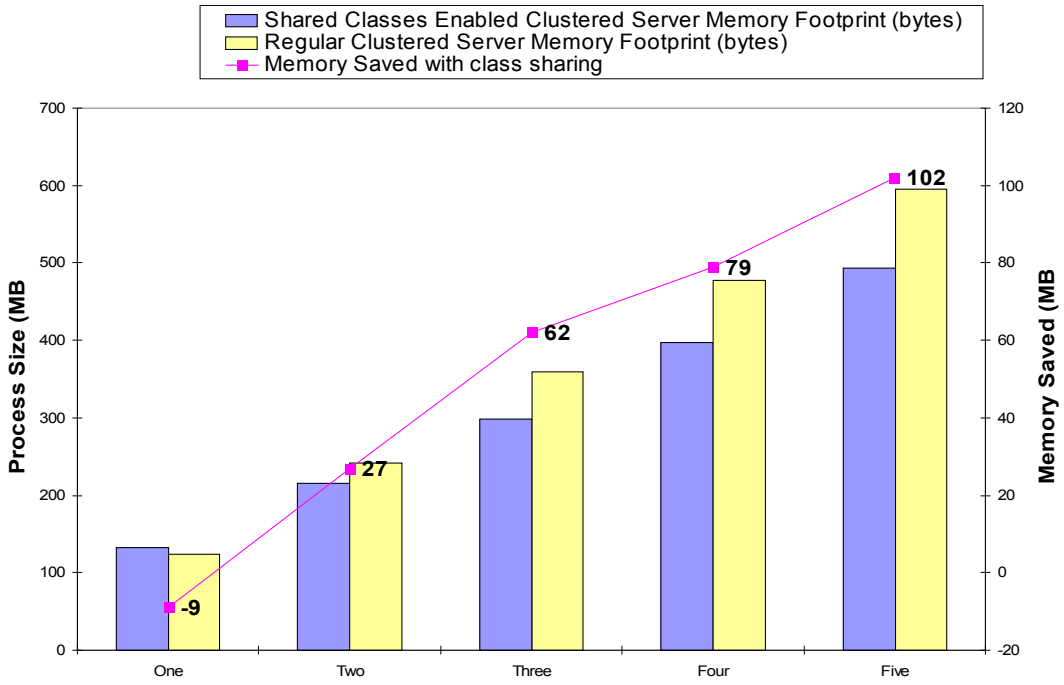
- **Compressed Reference Technology**
  - Addresses stored as a sequence of 32-bit values
  - Efficient bit-shifting algorithm
- **Dramatic improvements in 64-bit performance**
- **Improvements in 64-bit memory usage**
  - Heap sizes up to 28GB with the same physical memory overhead as an equivalent 32-bit deployment



ftp://ftp.software.ibm.com/software/webserver/appserv/was/WAS_V7_64-bit_performance.pdf

IBM Java solutions – optimizing memory usage

- **Class and code sharing (-Xshareclasses)**
  - Transparent and dynamic
  - Maintain isolated processes
  - Share class data across JVM processes
    - Applications share read-only class data
  - Share AoT JIT code
    - Improve start-up performance
    - Share JIT compiled code
  - Available in IBM Java5 and Java6 JDK's

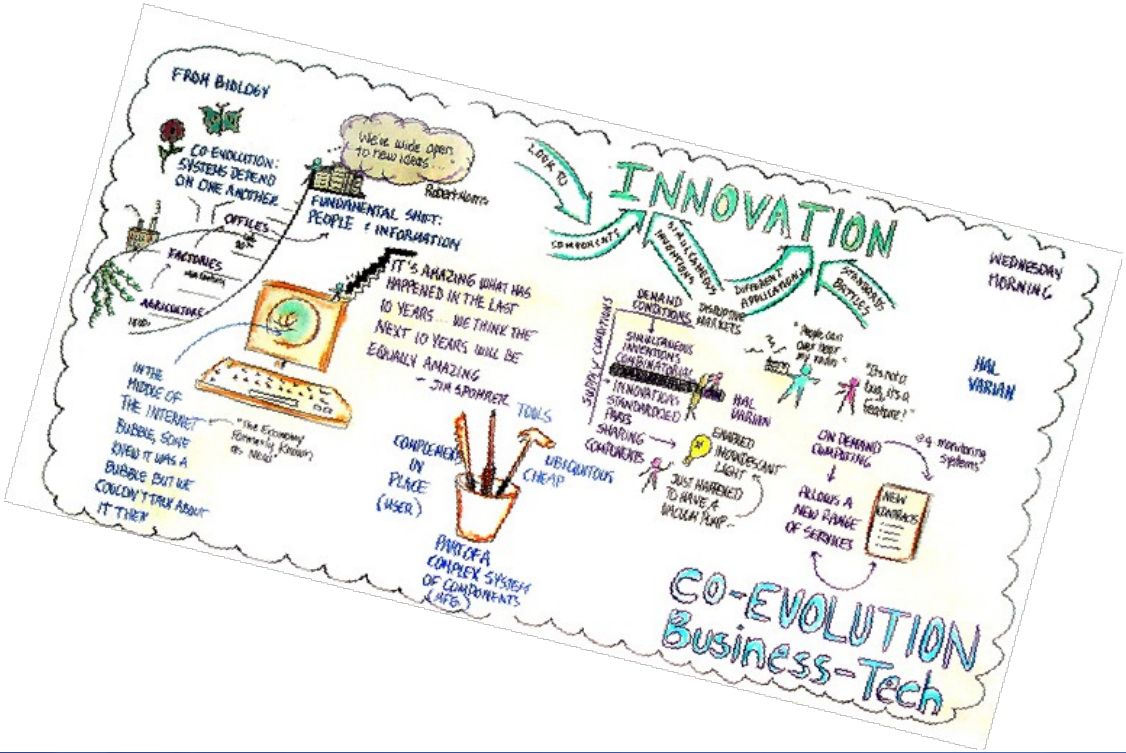


  - WebSphere instances are able to share non-changing class files amongst each other limiting the memory overhead of each JVM machine
  - Typically for a base WebSphere install each JVM uses about 30MB less memory with shared classes on its process size

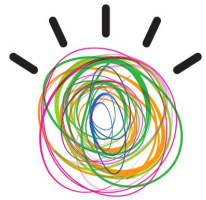
`http://www.ibm.com/developerworks/library/j-sharedclasses/`

IBM Java solutions – optimizing memory usage

- **Diagnostic tools**
  - Visualization of Java heap
  - Development and deployment decisions assisted by static and dynamic feedback
- **GC research**
  - NUMA aware algorithms with parallel hierarchical copying
  - Avoid moving large objects and read-heavy objects



IBM Java solutions – green computing

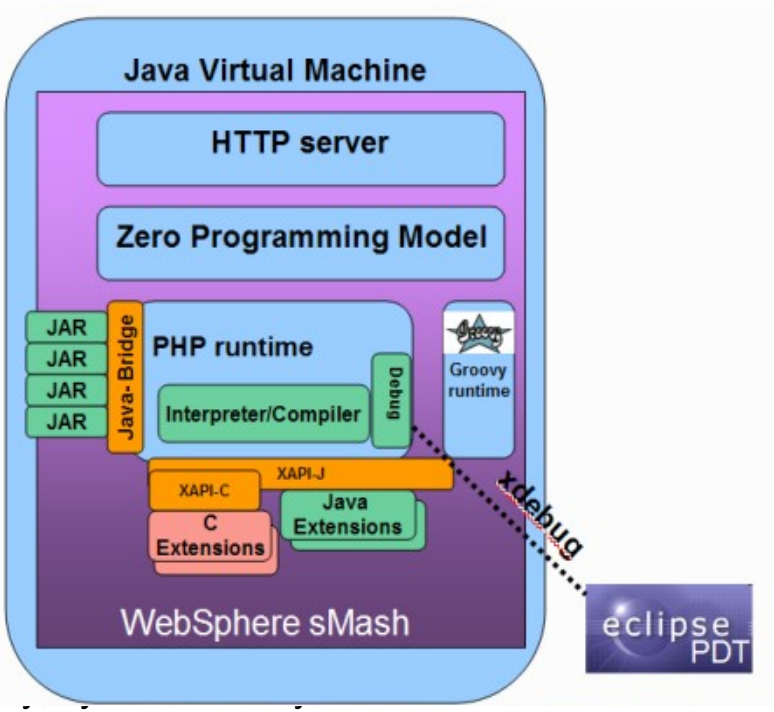


- **Increase MIPS per Watt**
  - Social, environmental, and economic impact
  - Increasing impact on IBM software and hardware design
  - IBM leads on industry benchmark SPECpower_ssj2008 (Last publication update: Wed Jan 14 17:01:06 EST 2009)
- **Consume less power when idle**
  - Leverage CPU / OS features to reduce energy consumption
- **Maximize utilization when running**
  - Optimize software to accomplish more in fewer cycles
  - Use Dynamic Voltage and Frequency Scaling
- **Avoid power hungry operations**
  - Garbage Collection
    - Avoid moving memory and exploit memory locality
  - Just In Time Compiler
    - Generate energy conservation aware code
    - Use processor binding to allow other cores to sleep

Let's build a smarter planet

Application Trends – dynamic runtimes

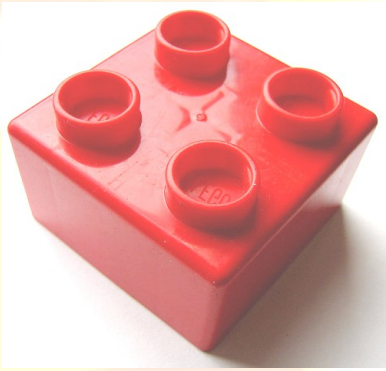
- **Mash-ups and Web2.0**
  - PHP estimated present on 20M+ web domains employing “LAMP” stack
  - Simple syntax and dynamic typing encourage situational applications
- **Mix Java and PHP assets and programmers**
  - Re-use frameworks, applications, extensions
- **Same-process interaction between Java and PHP**
  - Inter-language calls without IPC overhead
  - Data sharing without copies
- **Benefit from vast investment in VM Technology**
  - JIT, memory management, ...
  - Ongoing investment in Java RAS and tools
- **JVM Dynamic languages community**
  - Java world embracing scripting languages such as Jru
  - New bytecode `invokedynamic` scheduled for Java 7
    - “JSR 292: Supporting Dynamically Typed Languages on the JavaTM Platform”



WebSphere sMash

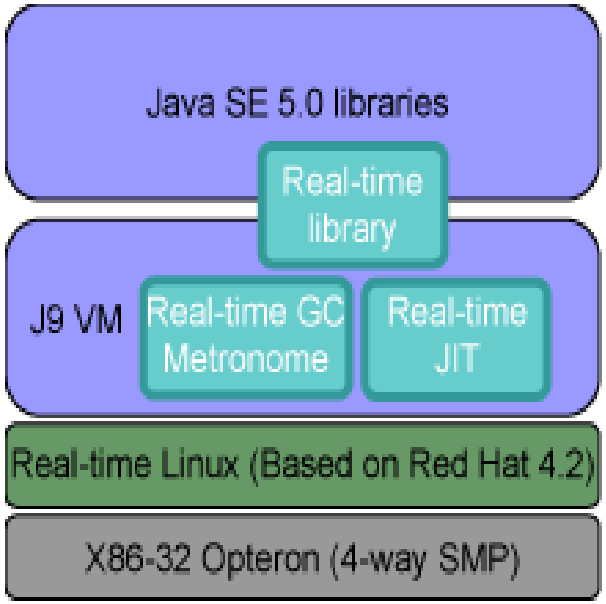
Application Trends – modularity and right sized runtimes

- **Modularity**
  - Java system components (JARs and DLLs) do not support multi-variant runtimes well
  - Applications typically written to a particular deep stack and JAR extensions
    - e.g. Java EE 5 on Java SE 6 with JUnit 4.5 JAR and json JAR and ...
    - Leads to dependency management problems (akin to DLL hell)
  - Modules provide versioned interfaces, dependencies, life-cycle management, ...
    - Applications depend upon software assemblies of modules
    - OSGi modules are compatible with regular JARs
    - Apache Harmony has Java SE composed of ~30 OSGi modules
  - Java standardization efforts in flux
    - JSR 277 : The Java Module System
    - JSR 291 : Dynamic Component Support for Java SE
- **Right-sized runtimes**
  - Apply Java technology modules in new configurations
  - Convergence of Java ME and Java SE technologies
  - Assemblies suited to different application goals
    - Embedded profile, server profile, batch processing, transaction based, multi-core aware, ...
    - Different implementations of equivalent modules
    - e.g. Google Android



Application Trends – Deterministic Quality of Service

- **Real-time Java technology**
  - Provide predictable runtime characteristics
  - Deterministic quality of service (DQoS) to time sensitive applications
- **If the software you are built on does not support DQoS ...**
- **Real-time stack**
  - Operating system
  - JIT compiler
  - Memory manager
  - Class libraries
- **IBM WebSphere Real Time**
  - RTOS required
  - Unique GC algorithms
  - Ahead of Time compilation
  - Augmented class libraries



Summary

- **Hardware**
  - New architectures require fundamental software changes for exploitation
  - Virtualization brings opportunities and challenges for managed runtimes
  - IBM software and hardware engineers collaborate closely to create best of breed
- **Software**
  - Java application programming hidden from increasing complexity
  - Additional dynamic languages such as PHP running on VMs
  - Growing complexity requires sophisticated visualization and diagnostics
- **Environmental**
  - Principal costs of enterprise computing changing
  - Social responsibility
  - Efficiencies benefit applications and environment
- **Future trends**
  - Reuse assets in new configurations
  - New guaranteed levels of service extend programming paradigms
  - IBM well positioned to define the future technology direction

Java Technology Centre

Reading the runes for Java runtimes

The latest IBM Java SDKs ... and beyond

Tim Ellison
tim_ellison@uk.ibm.com