What’s Hot in BEA JRockit

Staffan Larsen

JRockit Chief Architect
BEA Systems

TS-2171

Goal

Learn about the technical direction of JRockit, some specific optimizations we do, as well as the diagnostics tools that are available.

Agenda

Quick Facts

Real-Time

Virtualization

Optimizations

Diagnostics

Q&A

JRockit Quick Facts

In a Nutshell

- Java™ Virtual Machine (JVM™) for enterprise wide usage
- 100% Java Platform, Standard Edition (Java SE platform) compatible
- Only JVM™ software—not class libraries
- Project started 1998, acquired by BEA 2002
- Development is still done in Stockholm
- Written (mostly) in C
- Free (as in beer)

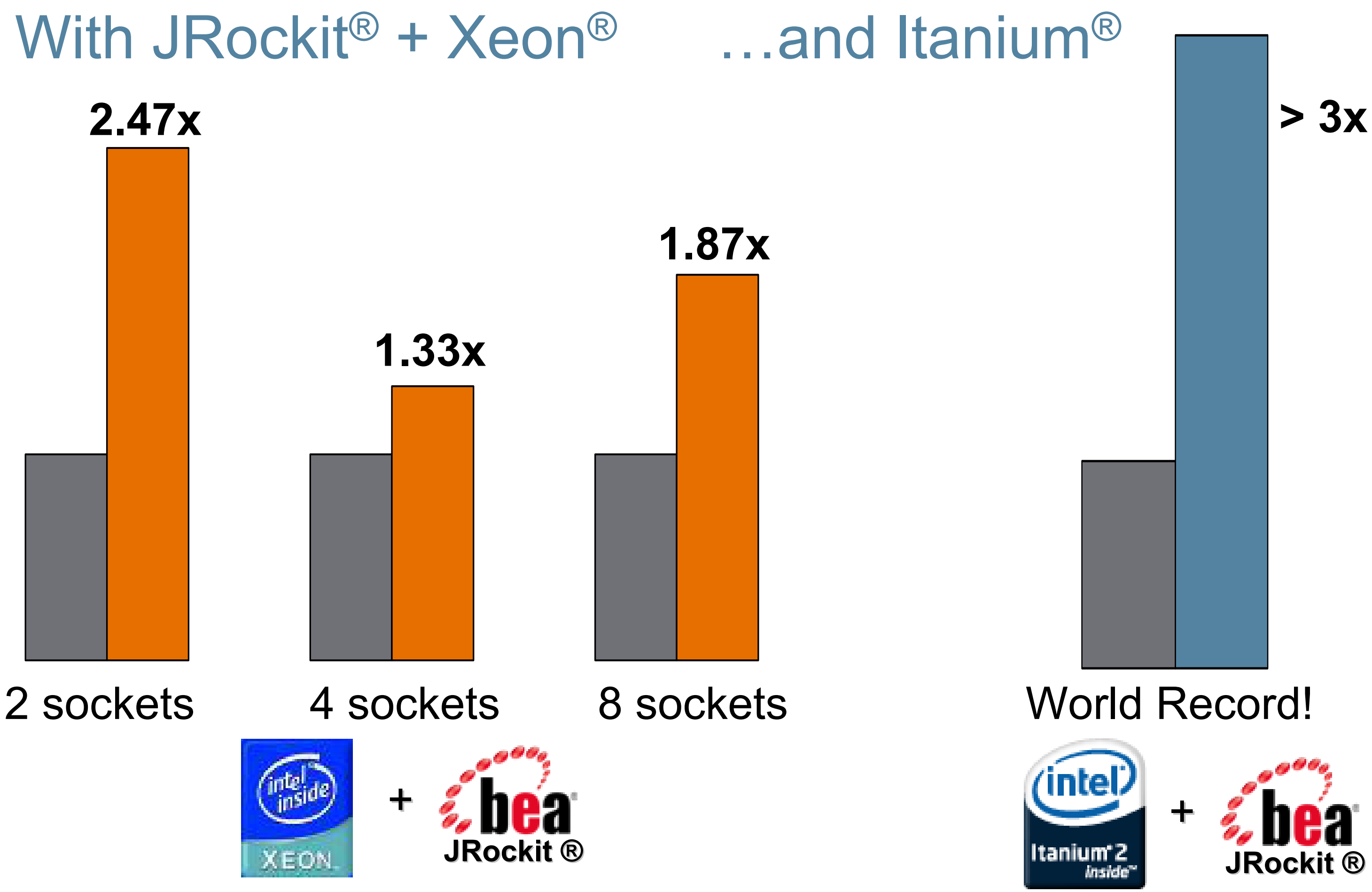
The terms “Java Virtual Machine” and “JVM” mean a Virtual Machine for the Java™ platform.

JRockit Quick Facts

Platform support

- Java Development Kit (JDK™) independent
  - With every new release, you get the same update on all JDK versions
- Available on 7 platforms…
  - Windows (x86, x86_64, ia64)
  - Linux (x86, x86_64, ia64)
  - Solaris™ Operating System (SPARC®)
- …and 3 JDK versions: 1.4.2, 5.0, and 6.0

BEA + Intel Leads Java Performance



Based on SPECjbb2005 bops of the top results from http://www.spec.org as of March 30, 2007. Xeon comparisons vs. best Opteron scores, Itanium vs. best non-Itanium score. See backup slides for full details.

Agenda

Quick Facts

Real-Time

Virtualization

Optimizations

Diagnostics

Q&A

Real-Time and Deterministic GC

Definitions

- *“Real-time”*—soft real-time
- *“Deterministic GC”* —GC with guaranteed upper bound for pause times

Real-Time

Java technology is moving towards “Real-time” applications

- SIP server—Telecom (VOIP)
  - 50–100 ms response times
  - Maximize # calls set up per second
  - Longer response times means dropped calls (busy signal)
- Trading processing—Financial services
  - 10–20 ms response times
  - Maximize trades per seconds
  - Lower response times means more trade wins

Real-Time

What the JVM software can help with

Max latency = time to process transaction + max pause time

- Deterministic GC provides max pause time guarantees
  - Example: “No pause should be longer than 10 ms”
- Requires no re-write of code
  - Unlike Real-Time Specification for Java (JSR 1)
- Tooling to find latency problems

JSR = Java Specification Request

Deterministic GC

Target applications and platforms

- Applications
  - Typical server applications
  - No code rewrites
  - Verified with 3–4 GB heap and 30% live data
- Platforms
  - Standard hardware and OS
  - Real-time OS not required

Deterministic GC

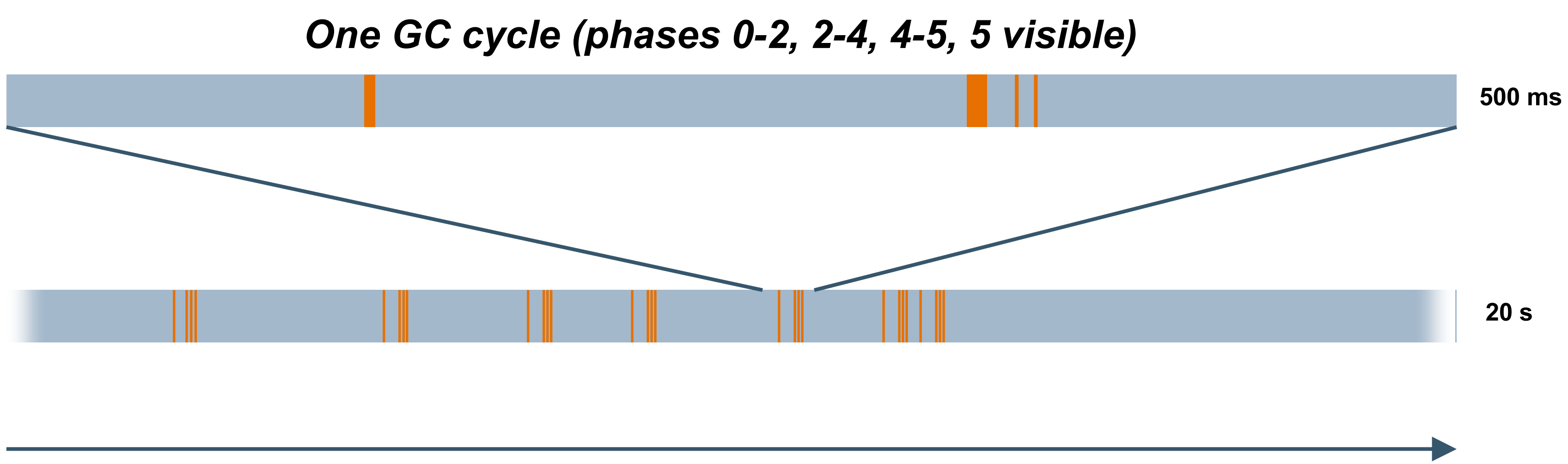
How does it work?

- Concurrent
- All work is broken up into pieces
  - Short but frequent pauses
- Divide and conquer
  - Take too long? Split and reschedule
- Speculate—redo
- Highly tuned
  - “The devil is in the details”

Deterministic GC

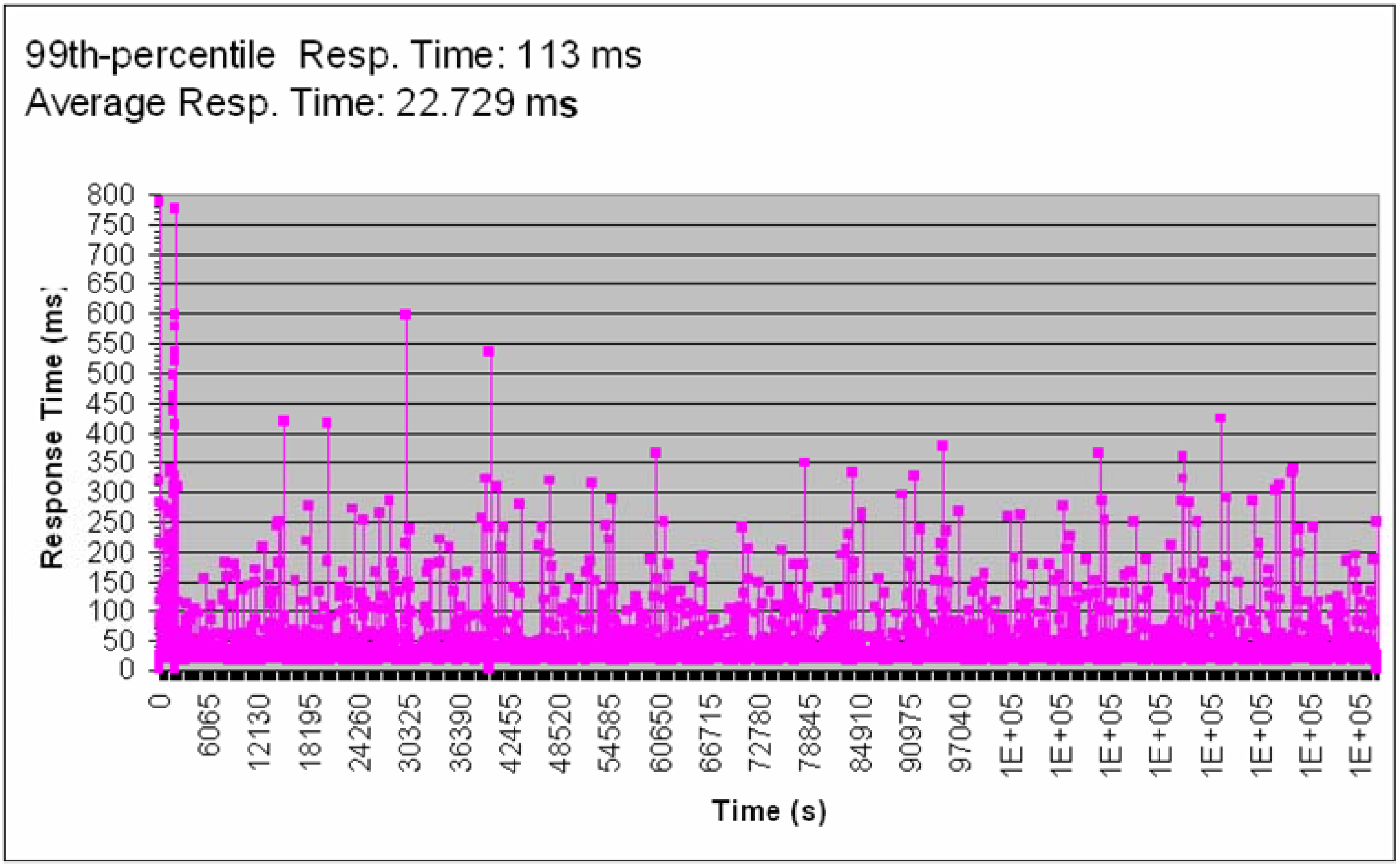
Divide and conquer

- Short but frequent pauses
- (Orange color is pause)



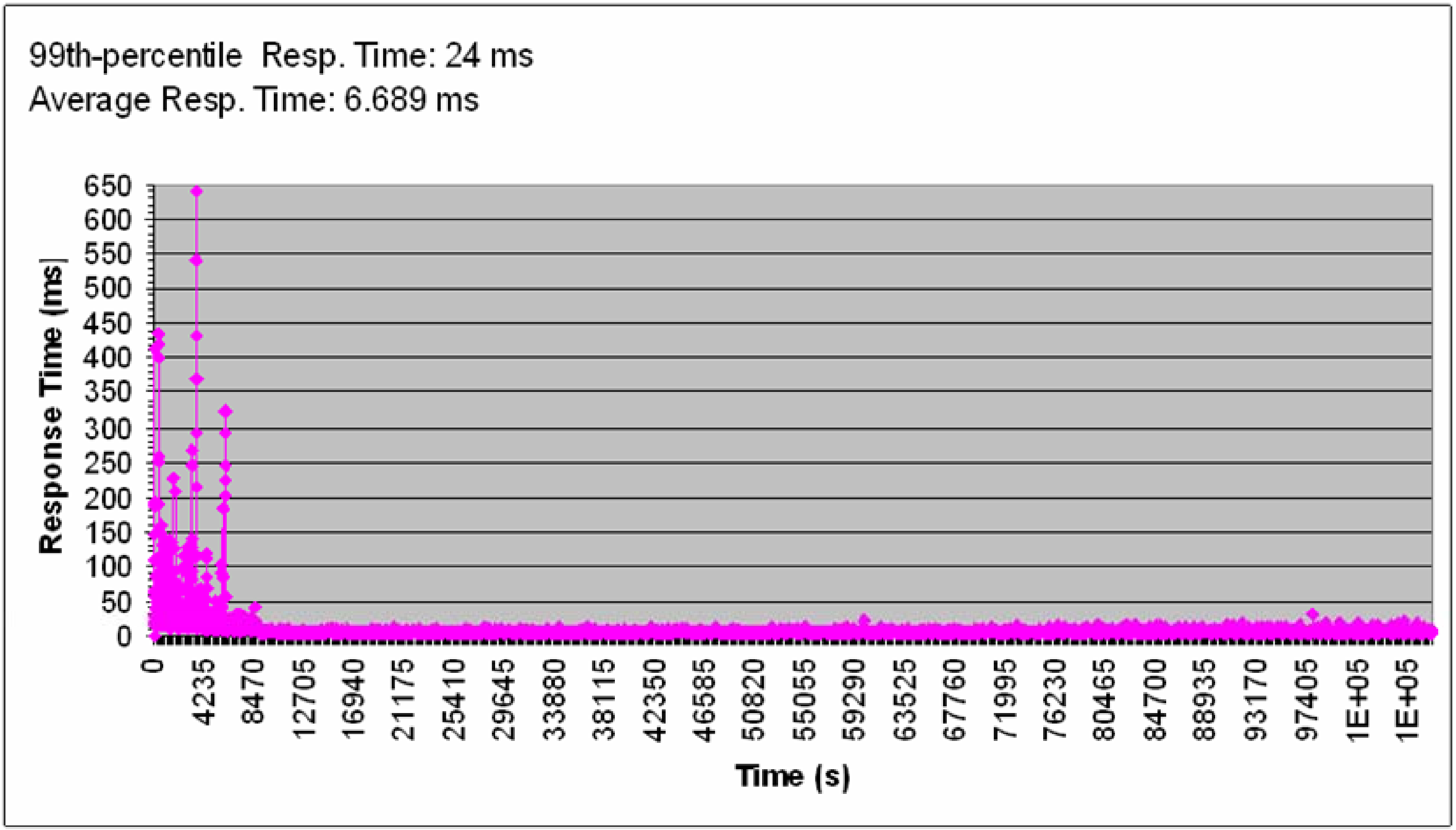
Standard GC

WLSS 2.2



Deterministic GC

WLSS 2.2



Agenda

Quick Facts

Real-time

Virtualization

Optimizations

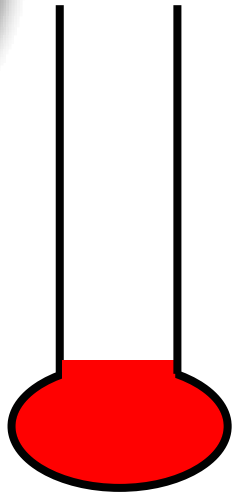
Diagnostics

Q&A

Datacenter Problems

A case for virtualization

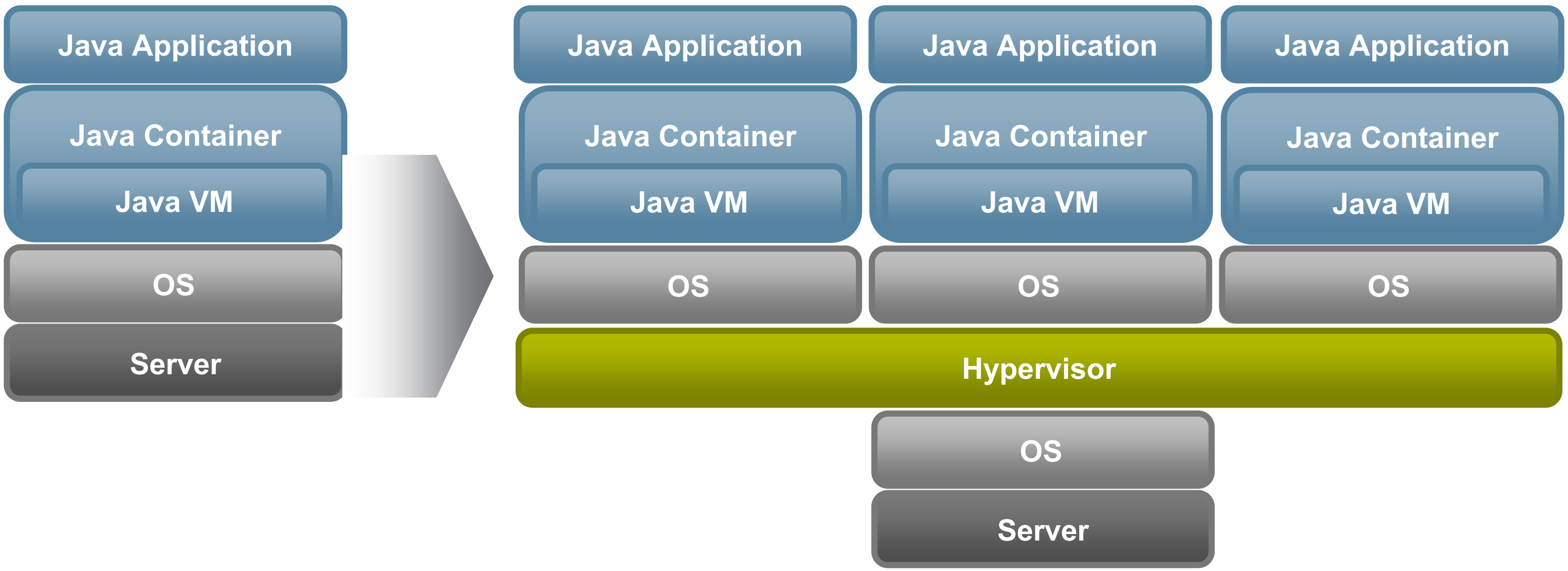
- Space
- Power
- Cooling
- Utilization
- Flexibility



- Solution: server virtualization

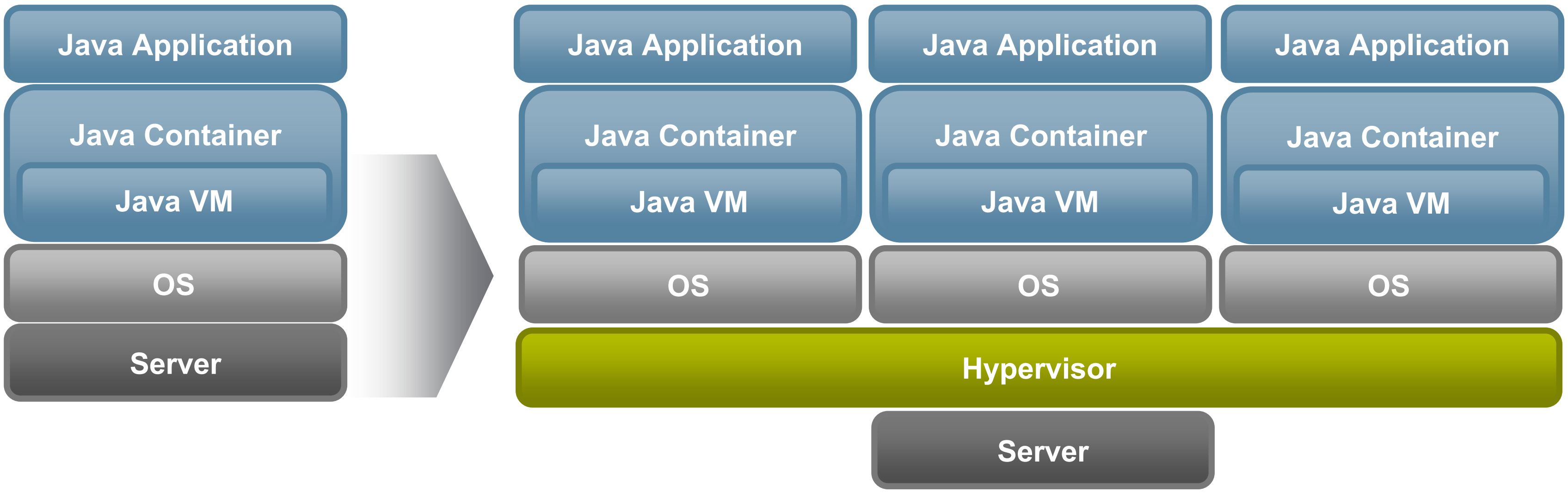
Basic Server Virtualization

e.g., VMware GSX, Microsoft Virtual Server



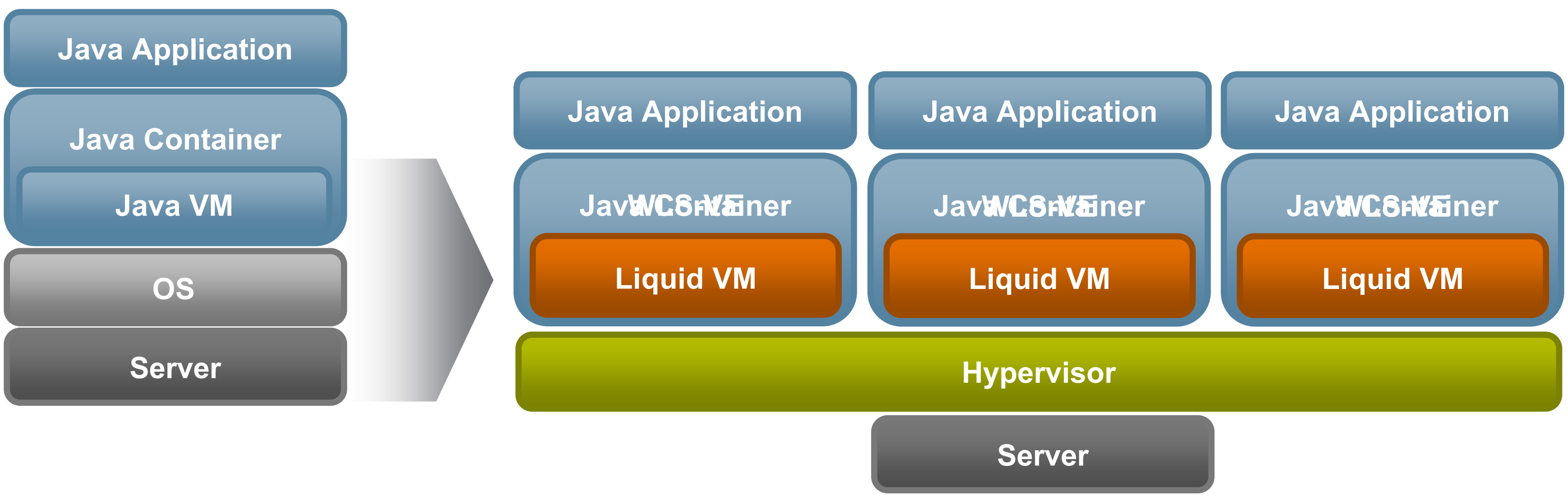
Next-Generation Server Virtualization

e.g., VMware ESX, Xen



New World: Liquid VM

Coming in Q2’07 in WebLogic Server Virtual Edition



java.sun.com/javaone

Rationale for LiquidVM

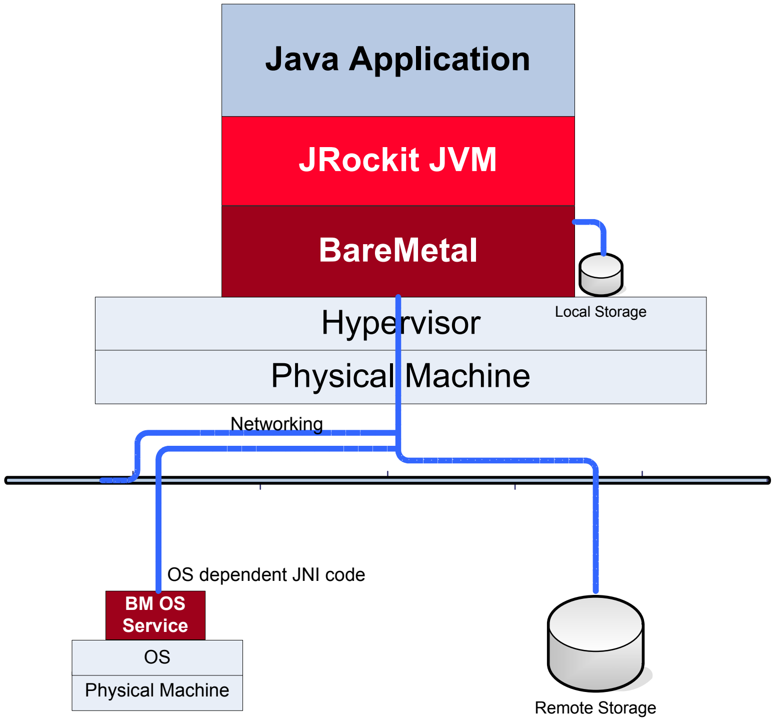
Efficient Java technology

- Remove unnecessary OS layer to gain efficiency
    - Footprint, performance, latency
- Sharing identical and free resources between VMs
    - Footprint efficiency
- Adaptive resource usage in a virtual environment
    - Add/remove memory at runtime
    - Add/remove CPUs at runtime
- Take advantage of hypervisor functionality in Java technology
    - Snapshots
    - Live migration

Bare Metal Technology Overview

From OS process to Java platform service on hypervisor

- *nix-like emulation layer for Java technology
  - Can run a JRockit JVM implementation for Linux
  - Not Linux-based
  - Networking, thread-scheduling, memory management, file storage
- Hypervisor support
  - Hardware-specific device drivers
- Not an OS in any normal sense
  - Only a single JVM implementation
  - No paging
  - No real device drivers
  - OS-dependent native code proxied
  - No GUI



JNI = Java Native Interface (JNI™)

Agenda

Quick Facts

Real-Time

Virtualization

Optimizations

Diagnostics

Q&A

JavaOne

Optimizations Agenda

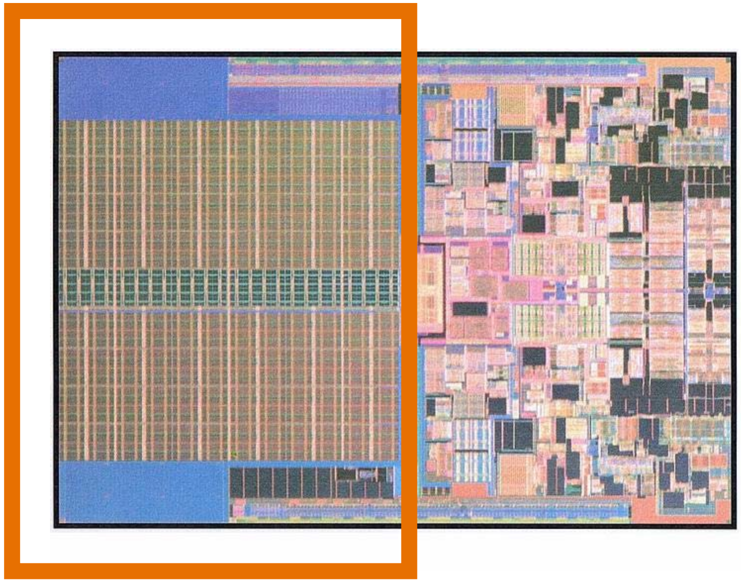
- Caches
  - Data alignment
  - Prefetching
- 64-bit heap efficiency
- StringBuilder

Memory Is Slow

The cache is your friend

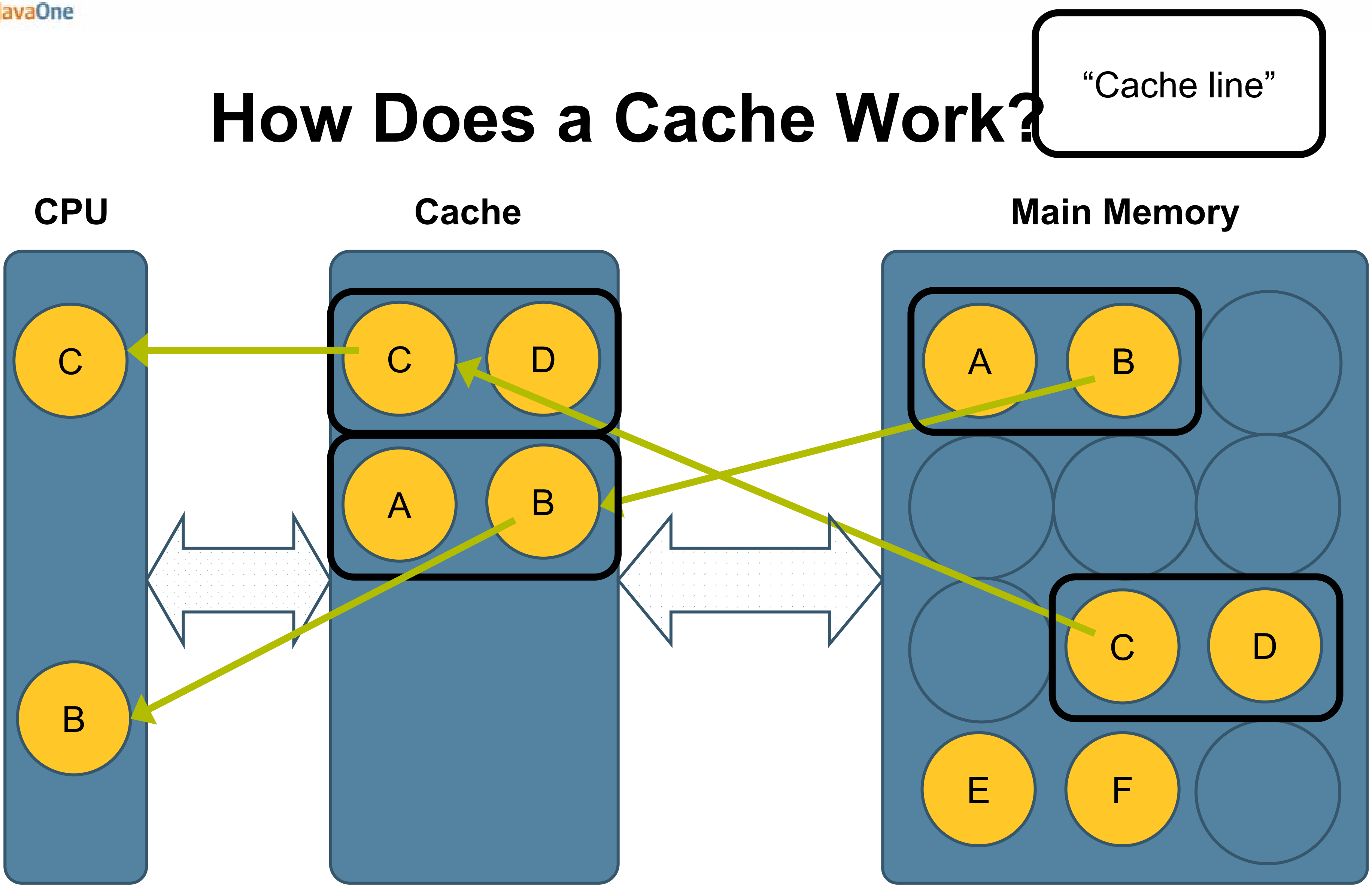
- Memory is slow—CPU is fast
- Cache line
- Cache size

Cache



Intel's Penryn 45nm chip

How Does a Cache Work?



Work With the Cache

Better cache usage

- Align data
  - Sequential data on the same cache line
  - Spatial locality should follow temporal locality
- Prefetch
  - Tell the cache ahead of time what the next item will be
  - Too often:
    - Destroys other data
    - Loads complete cache line—may take longer time

Data-Alignment

Use the cache line

- When GC moves objects—move referencing objects close together
- Will be used temporally close
- Move objects breadth-first

JavaOne

Special Case: Strings

Cache alignment

- Strings are very common
- Make sure String and char[] is on the same cache line
- When moving a String, move the char[] depth-first
- A string's character array is always used when the string is used

Prefetching at Allocation

Prime the cache

- JRockit has an area for each thread where new Objects are allocated
- The area is divided into smaller “chunks”
- When a chunk is used, next chunk is prefetched
- This means that the following operations will access memory already in the cache

Prefetching in the Garbage Collector

Faster parallel processing

- GC is parallel
- Work packets
- Prefetch the next item

Work packet	
Object 1	2) Process first Object
Object 2	1) Prefetch the next Object
Object 3	
Object 4	
...	

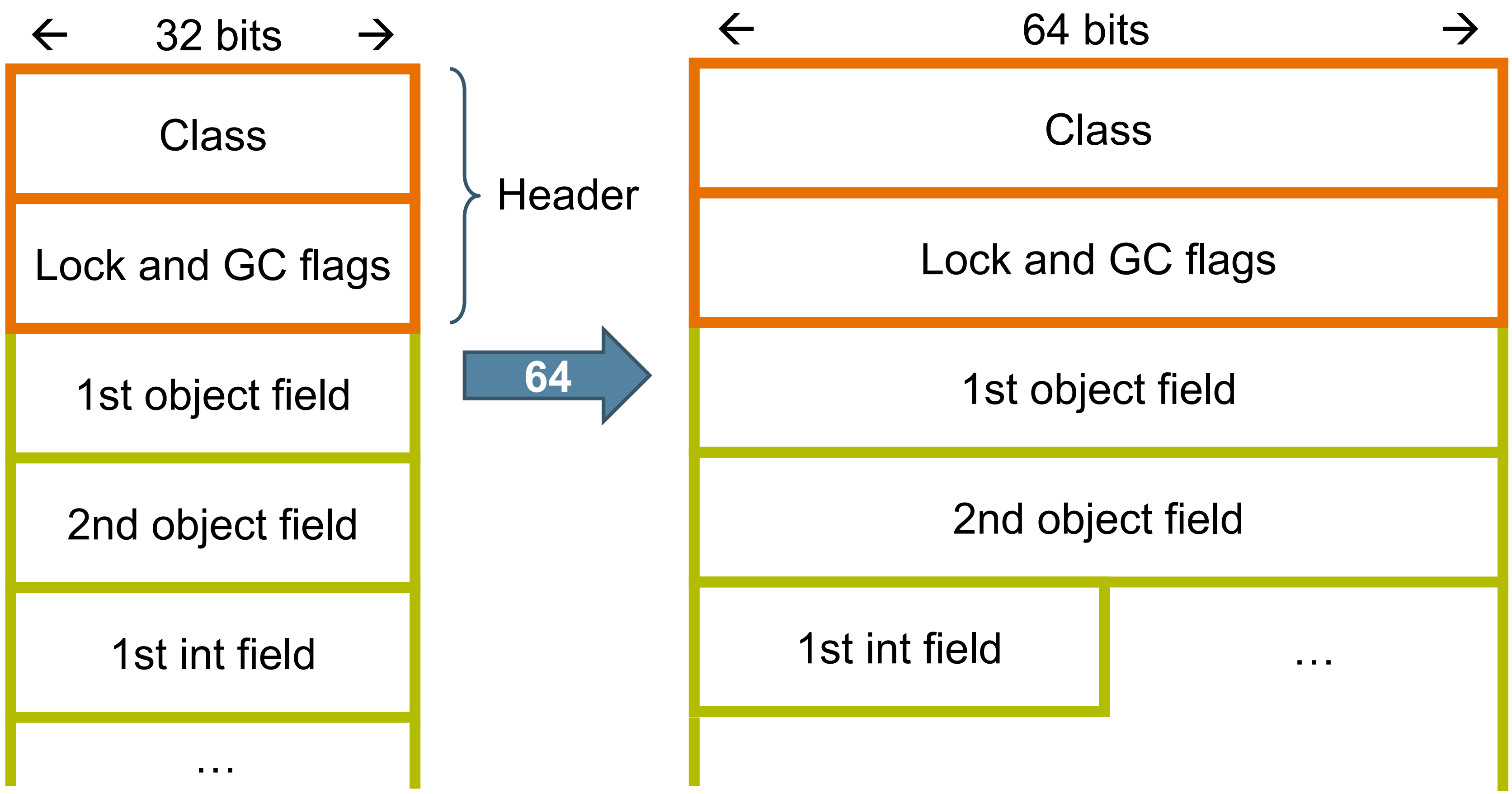
Going to 64-bit Heaps

More memory, more traffic

- 64-bits means more addressable memory
  - Larger heaps
- It also means longer (bigger) pointers
- Objects have pointers to other objects
  - More data to shuffle when objects are moved
- What can be done?

Object Details

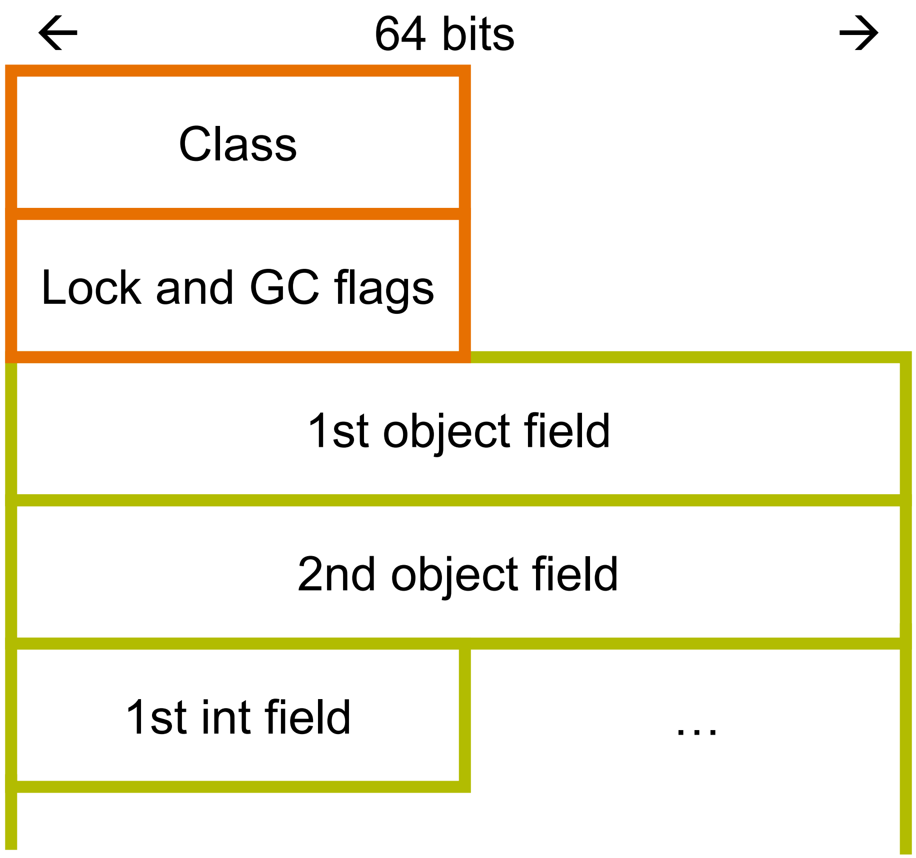
Memory layout



Smaller Object Header

Class reference

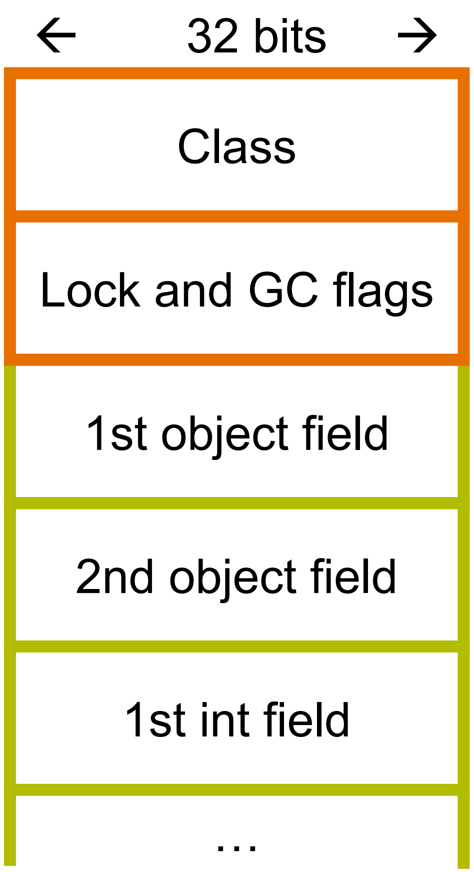
- Allocate classes in the lower 4GB of addressable memory
  - The top 32 bits are all zeros
- We only need 32 flag bits



Compressed References

Object references

- If heap is less than 4GB in size we can do the same trick for object references
- Can’t always get a 4GB chunk in lower memory
  - Keep the heap-offset separately
  - Add heap-offset to the reference before loading
- Only if **Heap Size < 4GB**



StringBuilder Inefficiency

Common pattern

- String concatenation is very common
- Consider this JavaServer Pages™ (JSP™) technology code

```
<TD><%= db.getName() %></TD>
```

- The JSP™ technology compiler will turn this into:

```
StringBuilder tmp = new StringBuilder();
tmp.append("<TD>");
tmp.append(db.getName());
tmp.append("</TD>");
tmp.toString();
```

StringBuilder Inefficiency

Multiple copies

```
public StringBuilder append(String str) {
    int newCount = count + str.length();
    if (newCount > capacity)
        expandCapacity();
    str.getChars(0, str.length(), value, count);
    count = newCount;
    return this;
}
```

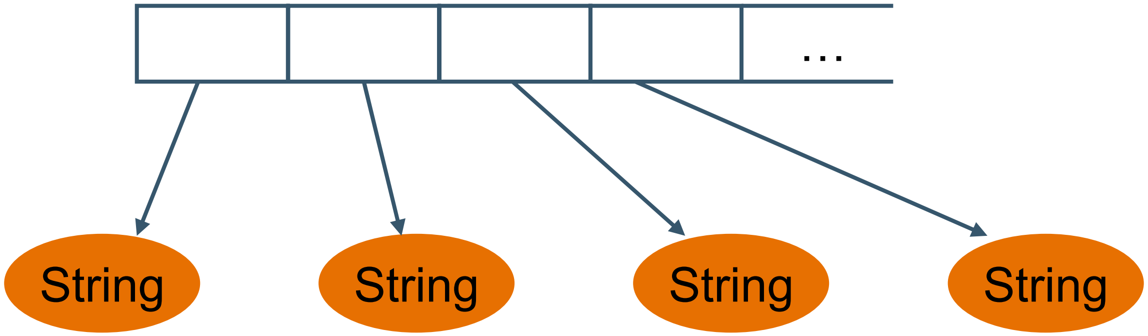
- Each append() can run out of space → data will be copied multiple times
- Lots of calls → slower code

java.sun.com/javaone

StringBuilder Inefficiency

Solution

- Create string data lazily
- append() stores each string
- toString() creates and copies the data
- More append() versions cause fewer calls



Patent pending

JavaOne

StringBuilder Inefficiency

Optimized code

```
StringMaker tmp = new StringMaker();
tmp.append("<TD>", db.getName(), "</TD>");
tmp.toString();
```

```
public StringMaker append(String str) {
  strings[size++] = str;
  return this;
}
```

Patent pending

StringBuilder Inefficiency

Optimized code

```
public String toString() {
   for(int i = 0; i < size; i++) {
    numChars += strings[i].length();
   }
   char value[] =  new char[numChars];
   for (int i = 0; i < size; i++) {
     // copy string data to value array
   }
   return new String(value);
 }
```

Patent pending

StringBuilder Inefficiency

Summary of solution

- Less objects allocated
- Less data copied
- Less calls made

→ Faster!

Agenda

Quick Facts

Real-Time

Virtualization

Optimizations

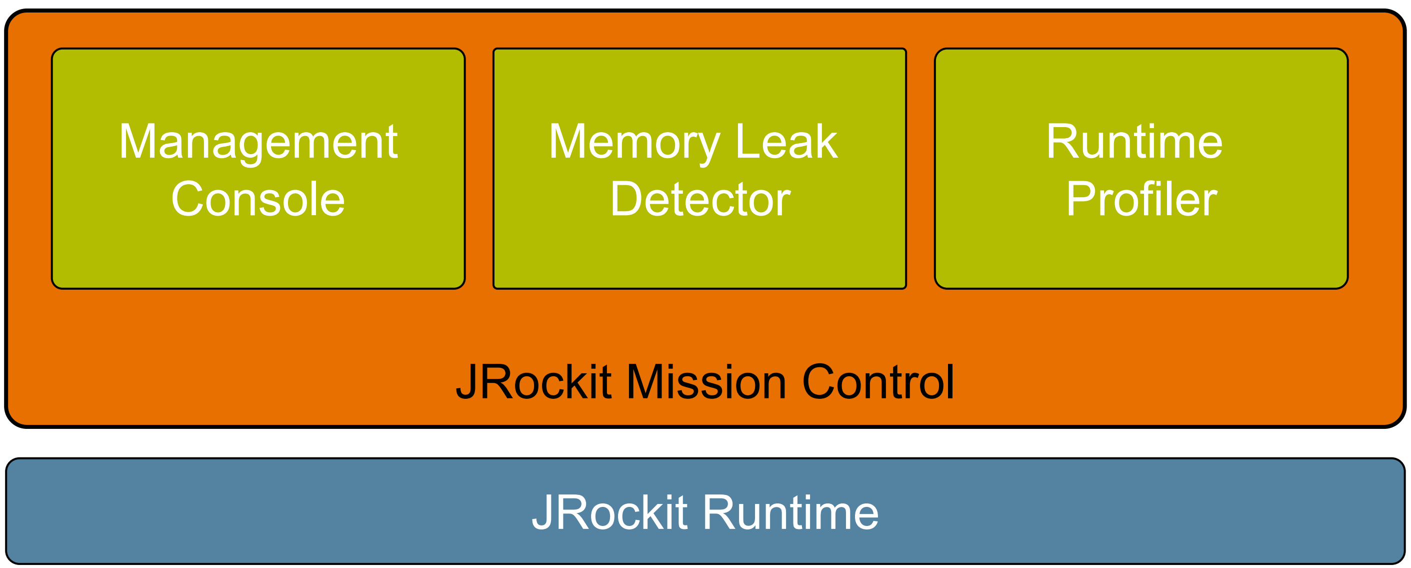
Diagnostics

Q&A

Diagnostics and Operations

JRockit mission control

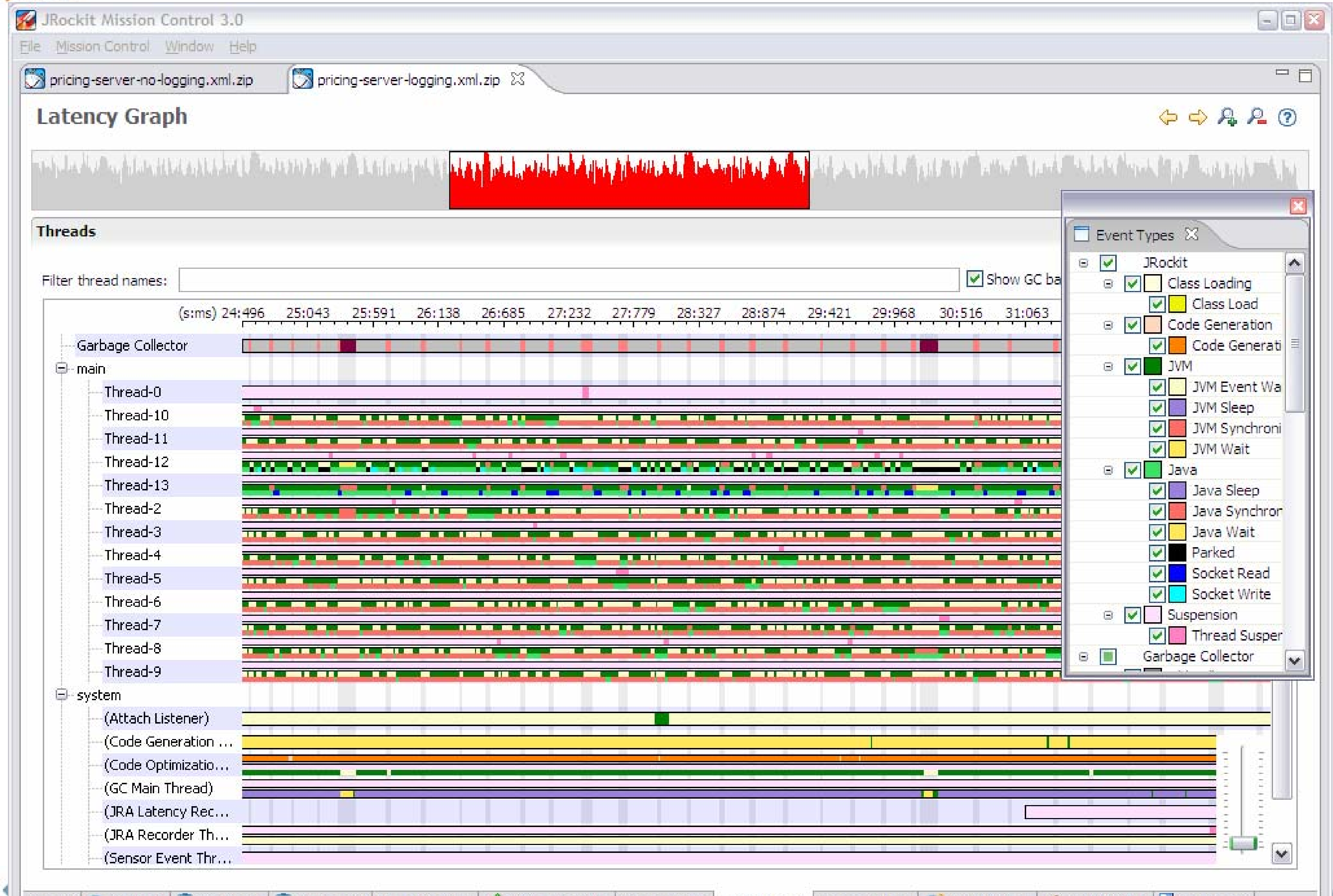
- Production-time monitoring
- Extremely low-performance cost
  - Tight JVM software coupling



Latency Analysis Tool

Where is 'dead' time spent?

- Visualize thread and transaction execution time lines
- Find sources of latency spikes
- Nanosecond resolution for real-time apps
- Very small performance and latency overhead
- Built into the JVM software core


JRockit Mission Control 3.0
File Mission Control Window Help
pricing-server-no-logging.xml.zip
pricing-server-logging.xml.zip
Latency Graph
Threads
Filter thread names:
Show GC ba
(s:ms) 24:496 25:043 25:591 26:138 26:685 27:232 27:779 28:327 28:874 29:421 29:968 30:516 31:063
Garbage Collector
main
Thread-0
Thread-10
Thread-11
Thread-12
Thread-13
Thread-2
Thread-3
Thread-4
Thread-5
Thread-6
Thread-7
Thread-8
Thread-9
system
(Attach Listener)
(Code Generation ...
(Code Optimizatio...
(GC Main Thread)
(JRA Latency Rec...
(JRA Recorder Th...
(Sensor Event Thr...
Event Types
JRockit
Class Loading
Class Load
Code Generation
Code Generati
JVM
JVM Event Wa
JVM Sleep
JVM Synchroni
JVM Wait
Java
Java Sleep
Java Synchror
Java Wait
Parked
Socket Read
Socket Write
Suspension
Thread Susper
Garbage Collector

Diagnostic Commands

Simple JVM software information

- Set of built-in diagnostics features
- Always accessible
    - jrcmd tool
    - Java Management Extensions (JMX™)
    - Console
- Example
    - jrcmd <pid> print_threads
    - jrcmd <pid> verbosity set=exceptions=info

Summary

- Java technology + real-time = true
- Virtualization makes Java technology efficient
- Cache and memory optimizations
- Visualizing latency problems

For More Information

- BEA's booth in the Pavilion
- http://dev2dev.bea.com/jrockit/
- http://forums.bea.com/

Q&A

What’s Hot in BEA JRockit

Staffan Larsen

JRockit Chief Architect
BEA Systems

TS-2171

Backup Slides

BEA + Intel SPECjbb2005 data

- 2 socket x86
  - Intel Xeon X5355, JRockit 5.0 P27.2.0, 222509 bops @ 55627 bops/JVM software
  - AMD Opteron® 2220, JRockit 5.0 P27.1.0, 88934 bops @ 44467 bops /JVM software
- 4 socket x86
  - Intel Xeon 7140M, JRockit 5.0 P27.1.0, 217334 bops @ 54334 bops/JVM software
  - AMD Opteron 8220SE, JRockit 5.0 P27.1.0, 163384 bops @ 40846 bops/JVM software
- 8 socket x86
  - Intel Xeon 7140M, JRockit 5.0 P27.2.0, 336653 bops @ 42082 bops/JVM software
  - AMD Opteron 885, JRockit 5.0 P26.4.0, 180418 bops @ 22552 bops/JVM software
- World Record
  - SGI Altix 4700, 128 dual-core Itanium CPUs, JRockit 5.0 P27.1.0, 4231610 bops @ 66119 bops/JVM software
  - Sun Fire™ E25K server, 72 dual-core UltraSPARC® IV+ CPUs, Sun JVM software 1.5.0_08, 1387437 bops @ 19270 bops/JVM software
- All measurements in SPECjbb2005 bops, the results are the best Intel-based vs. best non-Intel based results from www.spec.org as of March 30, 2007. SPEC and the benchmark name SPECjbb2005 are trademarks of the Standard Performance Evaluation Corporation.