



JavaOne™

java.sun.com/javaone

Java™ Platform Performance: Case Studies in Bottleneck Identification and Removal

Kumar Shiv, Intel

Paul Hohensee, Sun Microsystems

TS-6434



Trademarks And Abbreviations

(to get them out of the way ...)

- Java[™] Platform, Standard Edition (Java SE)
- Java HotSpot[™] Virtual Machine (HotSpot JVM[™])

Who Are These Guys?

➤ **Kumar Shiv**

- Principal Performance Architect, Managed Runtimes
- Intel, Hillsboro, OR

➤ **Paul Hohensee**

- Senior Staff Engineer, Hotspot JVM Tech Lead
- Sun Microsystems, Burlington, MA

Learn how performance bottlenecks are found and fixed in application code, Java library code and the JVM machine.

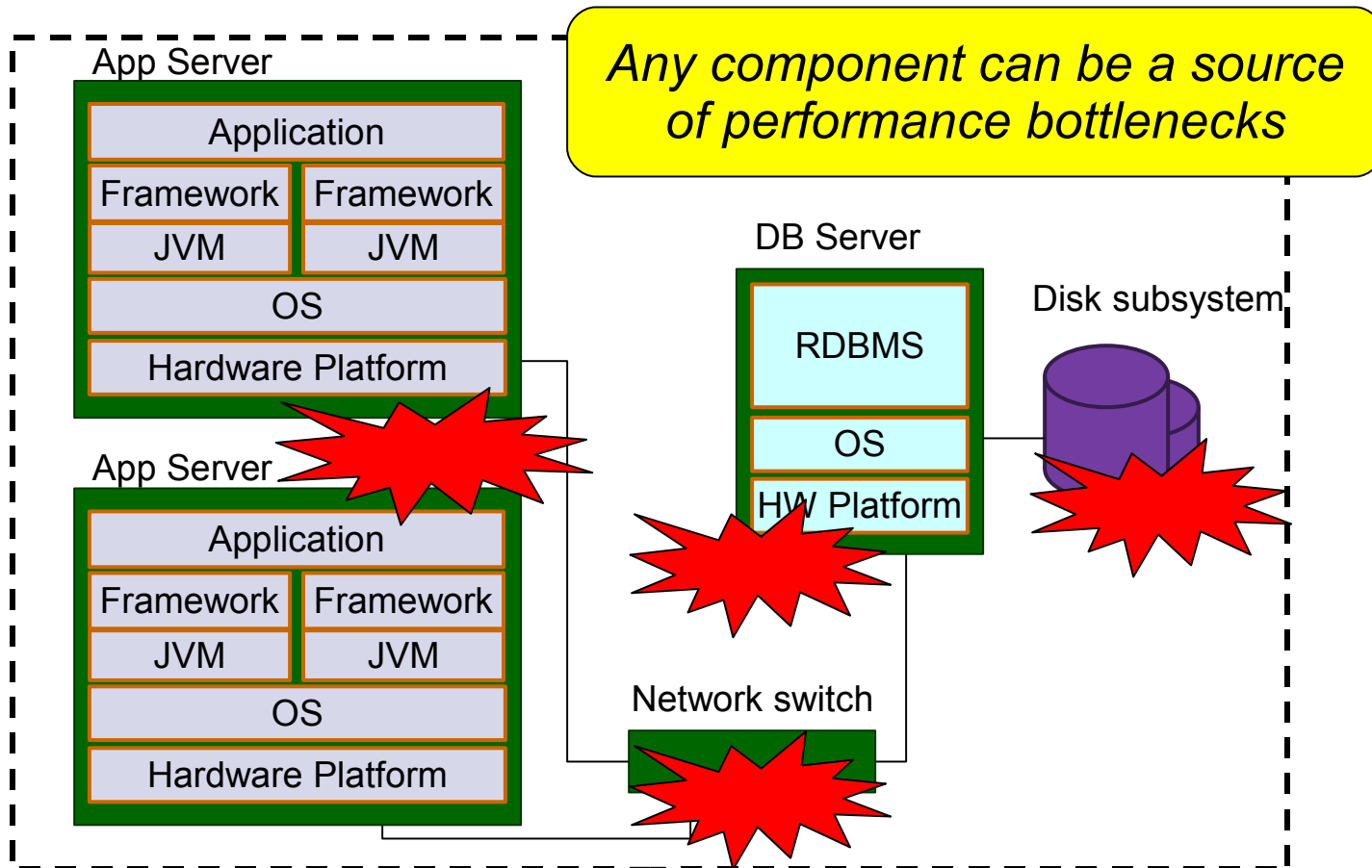
A large, light blue, stylized arrow pointing to the right, with a thick outline and a slight 3D effect.

GOAL

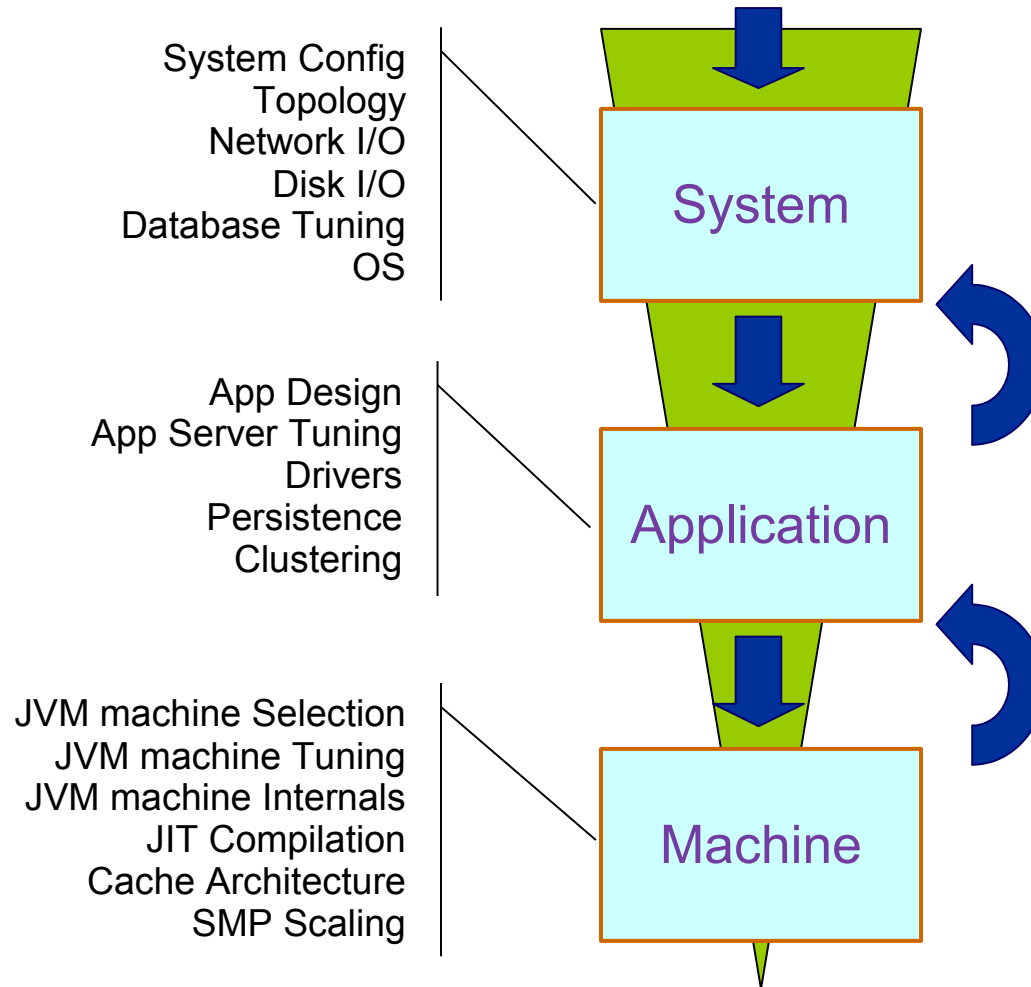
Agenda

- Methodology: Hardware, JVM Machine, Class Libraries, Applications
- Tools, Tools, Tools
- Case Study 1: SPECjvm2008, FFT Large
- Case Study 2: SPECjbb2005
- Demo
- Performance Analysis Takeaways
- Java Library and JVM Machine Performance Frontiers

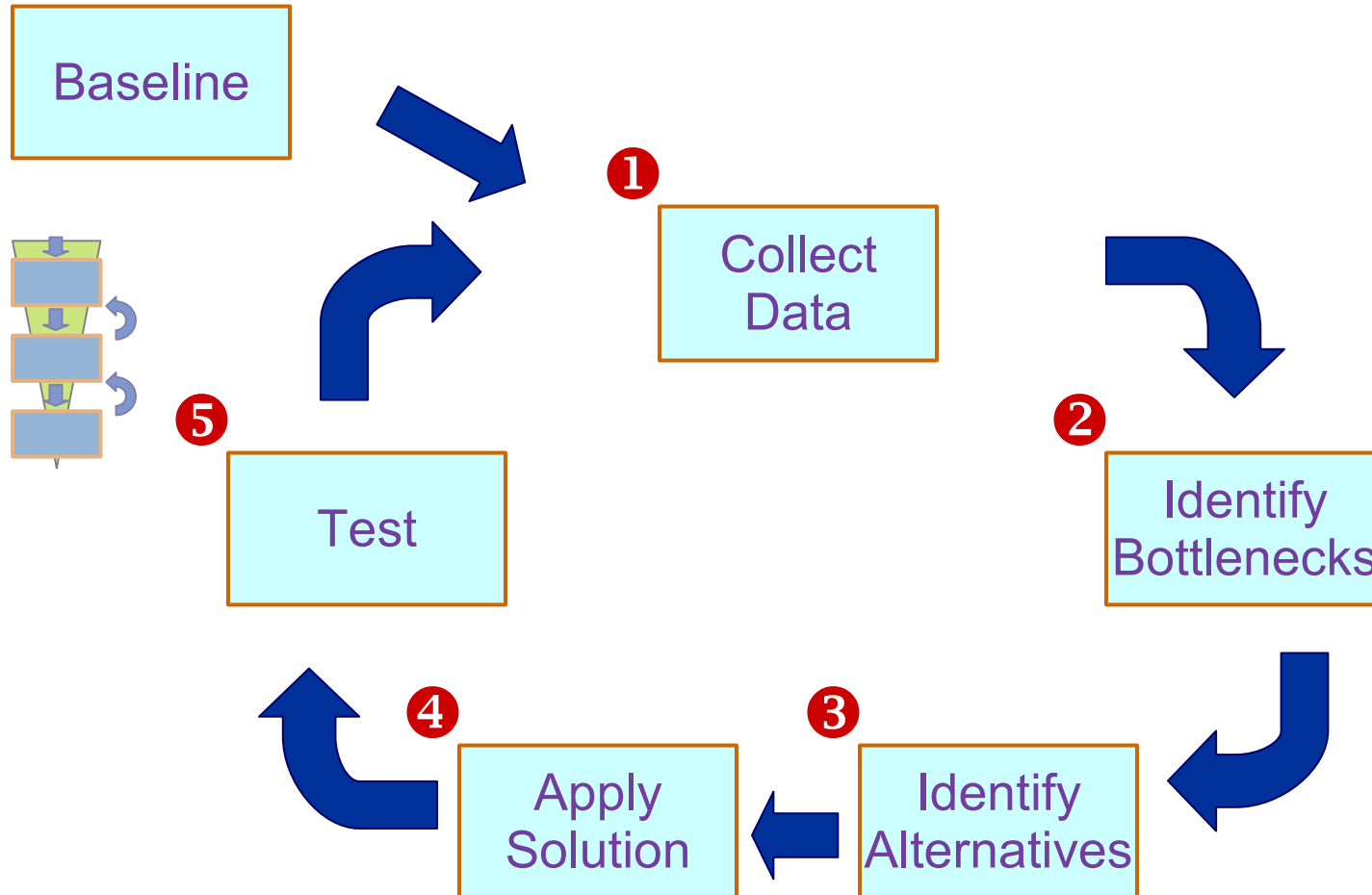
Multi-tier Solution Deployments Are Complex



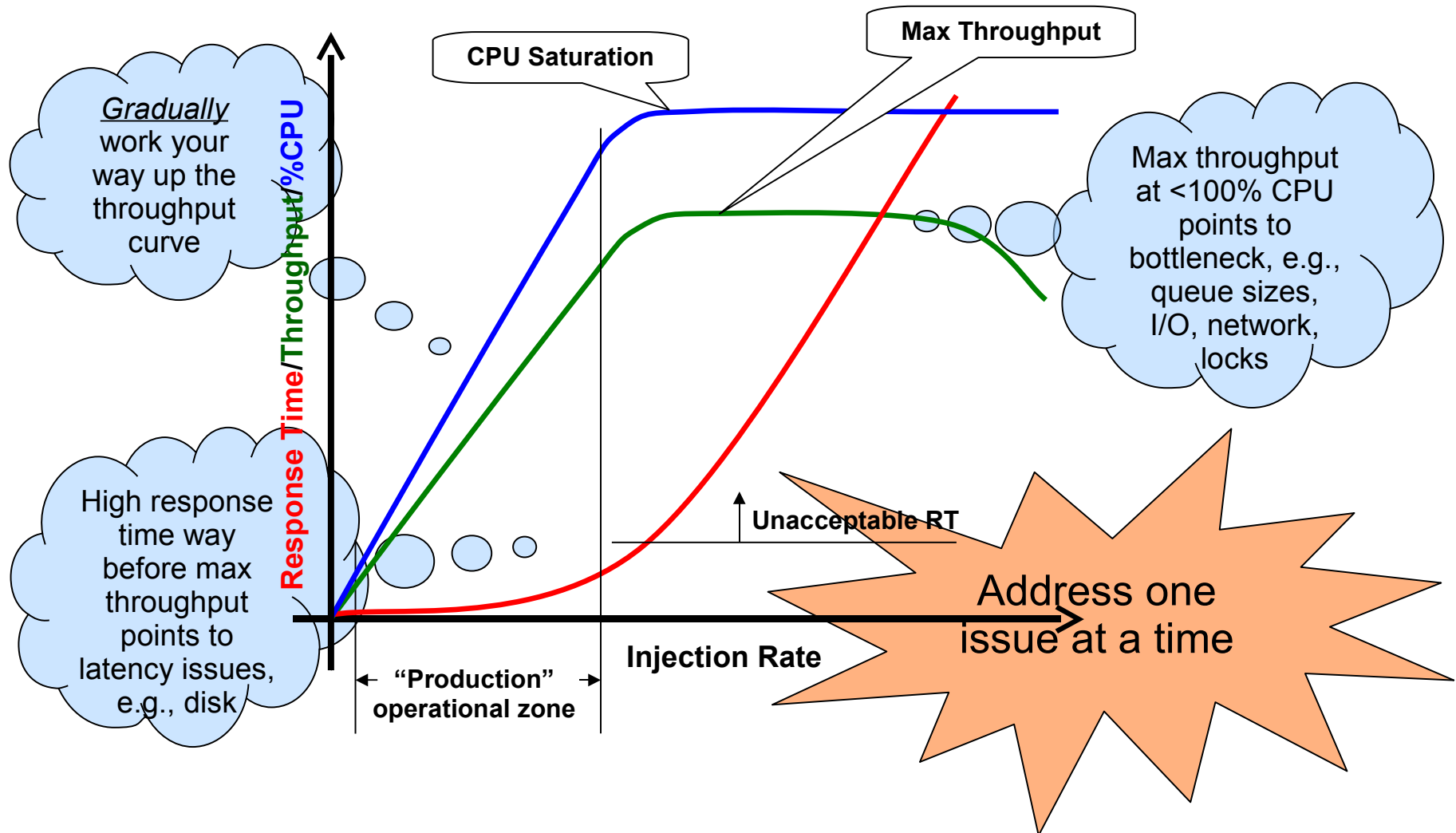
Top-Down Approach



Iterative Approach



Key Tool: A Throughput Curve



Agenda

- Methodology: Hardware, JVM Machine, Class Libraries, Applications
- Tools, Tools, Tools
- Case Study 1: SPECjvm2008, FFT Large
- Case Study 2: SPECjbb2005
- Demo
- Performance Analysis Takeaways
- Java Library and JVM Machine Performance Frontiers

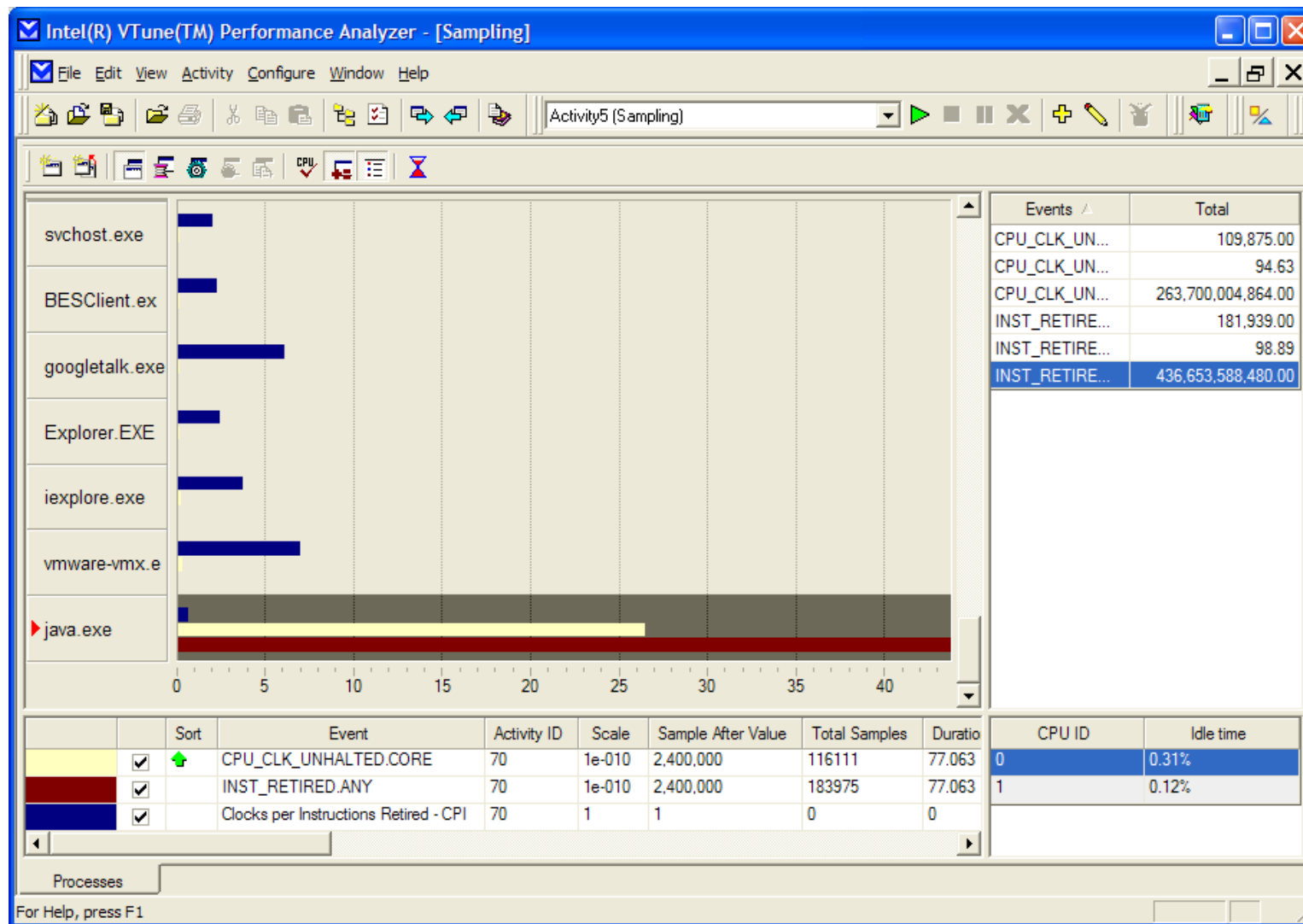
Tools, Tools, Tools

- SAR, iostat, other OS-supplied monitoring tools
- JDump and thread dumps
 - Often overlooked, easy to analyze with targeted scripts
- Dtrace: Solaris™ Operating System, OS X
 - <http://www.sun.com/bigadmin/content/dtrace>
- NetBeans™ software Profiler: Windows, Linux, Solaris, OS X
 - <http://www.netbeans.org>
- Project VisualVM: Windows, Linux, Solaris Operating System, Java HotSpot, JRockit™
 - <https://visualvm.dev.java.net>
- BEA™ Mission Control: Windows, Linux
 - <http://dev2dev.bea.com/jrockit/tools.html>
- Intel® VTune™ Performance Analyzer: Windows, Linux
 - <http://www.intel.com/software/products/vtune>
- Sun Studio Analyzer: Linux, Solaris operating system
 - <http://developers.sun.com/sunstudio>

VTune Performance Analyzer

- Helps identify and characterize performance issues by:
 - Collecting performance data from the system running your application
 - Organizing and displaying the data in variety of interactive views from system-wide down to source code or processor instruction perspective
 - Identifying potential performance issues and suggesting improvements
- Uses profiling APIs exposed by JVM machines
- Can profile pure and mixed managed and unmanaged code

VTune: Process View, XML Parser



VTune: Sampling

- Occasionally interrupts the processor
 - Periodic: typically 1000 times per second
 - Event-Based: Triggered by the occurrence of a user definable number of processor events
- Collects execution context
 - Execution memory address (Instruction Pointer)
 - OS process and thread ID
 - Executable module located at that address
 - If you have symbols for the module, post processing can identify the function or method at the memory address
 - Line numbers from symbol file can direct you to the relevant line of source code
- System wide



VTune: JIT'ed Code, XML Parser, traverseNode

VTune(TM) Performance Environment - [Source View - [C:\...erbench.tar\parserbench\JavaDOM.java]]

File Edit View Activity Configure Window Help

Activity1 (Sampling)

| Address | Line | Source | Penalties and Warnings | CPU | INST |
|---------|------|---|------------------------|-----|------|
| 0x120E5 | 201 | case Node.TEXT_NODE: | | | |
| 0x120E5 | 202 | if (node.getNodeValue() != null) { | | 23 | 22 |
| 0x120E6 | 202 | nop | | 4 | 5 |
| 0x120E7 | 202 | nop | | | |
| 0x120EC | 202 | mov eax, 0x55f17190h | | | |
| 0x120F1 | 202 | call \$-30335 (000000000ffff8a00h) | | | |
| 0x120F4 | 202 | cmp eax, 0x0h | | 1 | 1 |
| 0x120FA | 202 | je \$+0172h (0000000000000001f9h) | | | |
| 0x120FE | 202 | mov esi, DWORD PTR [esp+040h] | | 11 | 14 |
| 0x12103 | 202 | mov ecx, 0x55e31198h | | 7 | 2 |
| 0x12109 | 203 | mov ecx, DWORD PTR [ecx+01b0h] | | | |
| 0x12109 | 203 | thdCount[thd_id].contentBytes += node.getNo | | 19 | 8 |
| 0x1210C | 203 | cmp esi, DWORD PTR [ecx+08h] | | 4 | |
| 0x12112 | 203 | jae \$+0379h (000000000000000418h) | | 3 | |
| 0x12116 | 203 | mov edi, DWORD PTR [ecx+esi*4+0ch] | | 1 | |
| 0x12119 | 203 | mov ebx, DWORD PTR [edi+010h] | | | |
| 0x1211D | 203 | mov ecx, DWORD PTR [esp+010h] | | 1 | 1 |
| 0x12121 | 203 | mov DWORD PTR [esp+01ch], edi | | | |
| 0x12125 | 203 | mov DWORD PTR [esp+018h], ebx | | | |
| 0x12126 | 203 | nop | | | 1 |
| 0x12127 | 203 | nop | | | |
| 0x1212C | 203 | mov eax, 0x55f17190h | | | |
| 0x12131 | 203 | call \$-30399 (000000000ffff8a00h) | | | |
| 0x12134 | 203 | mov ecx, DWORD PTR [eax+010h] | | 9 | 6 |
| 0x12138 | 203 | mov ebx, DWORD PTR [esp+018h] | | | |
| 0x1213A | 203 | add ebx, ecx | | | |
| 0x1213E | 203 | mov edi, DWORD PTR [esp+01ch] | | 1 | |
| 0x12141 | 203 | mov DWORD PTR [edi+010h], ebx | | | |
| 0x12141 | 203 | imn \$+0125h (0000000000000001f9h) | | | |

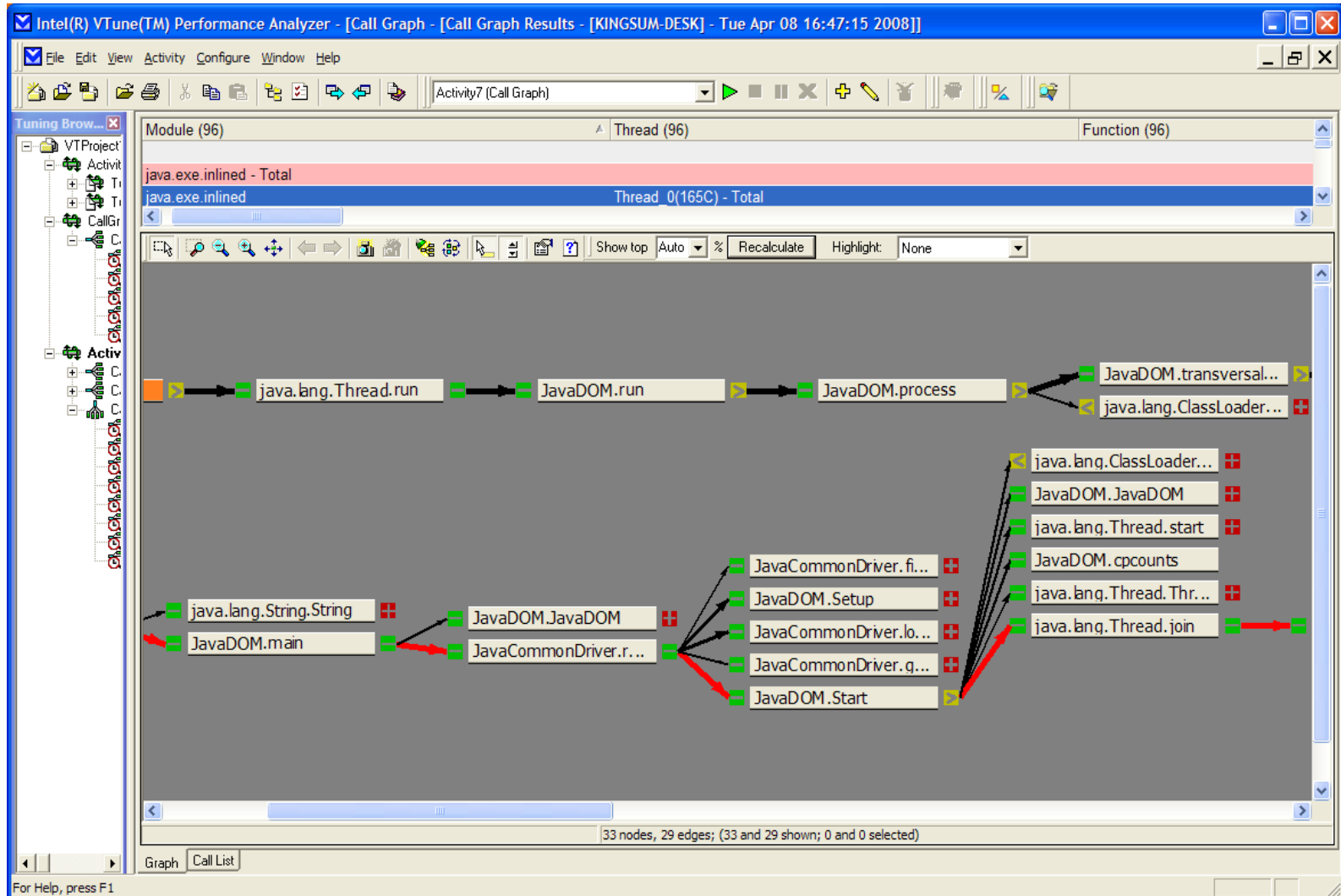
| Address | Size | Function | Class | CPU_CLK_UNHALTED.CORE (60) | INST_RETIRED.ANY (60) | Clocks per In... |
|---------|-------|------------------------|---------|----------------------------|-----------------------|------------------|
| ----- | ----- | --- Selected Range --- | ----- | | | .000 |
| 0x1206D | 0x590 | traverseNode | JavaDOM | 120 | 93 | 1.290 |

For Help, press F1

VTune: Call Graph

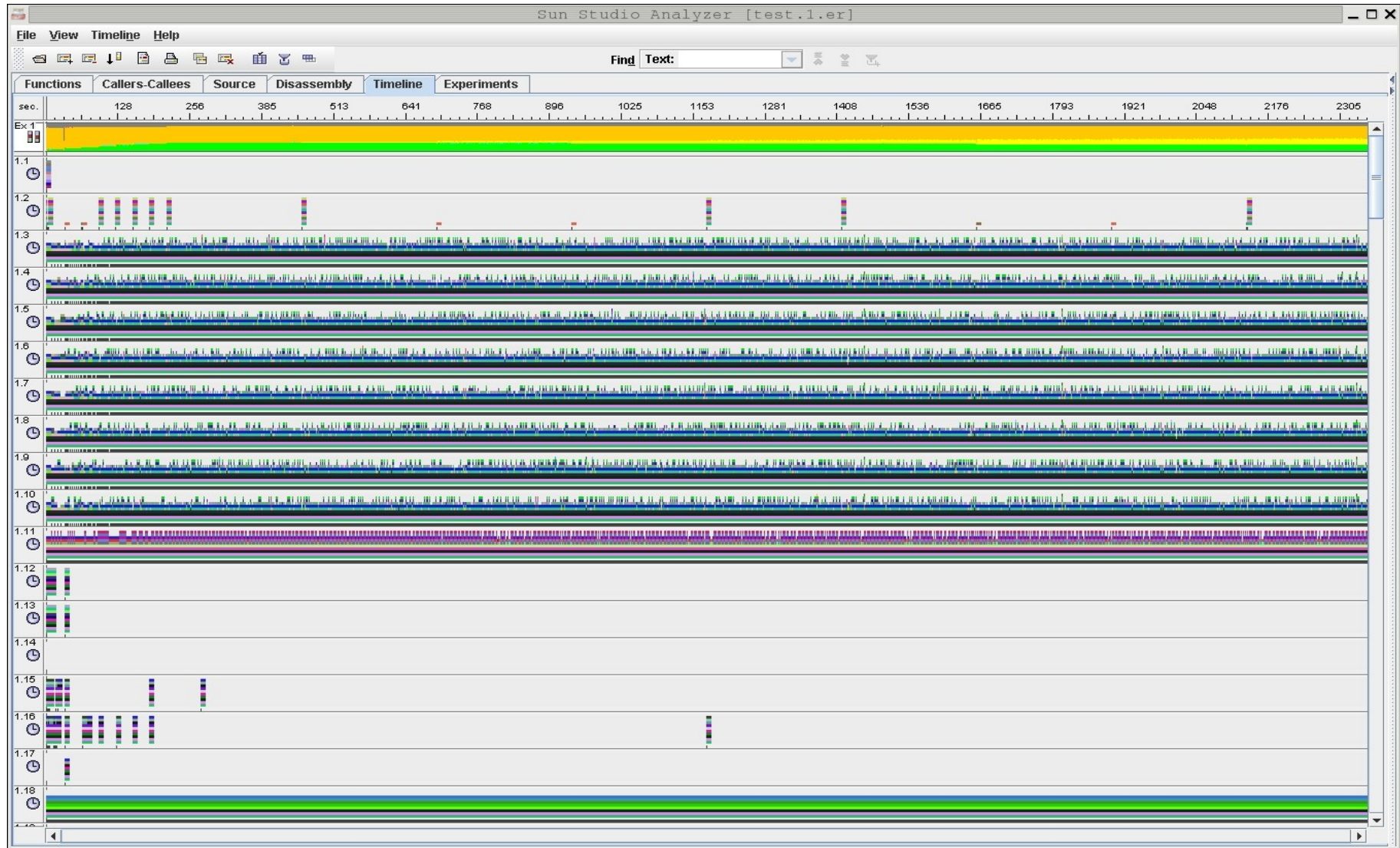
- Tracks method entry and exit points in your code
 - Periodic: typically 1000 times per second
 - Event-Based: Triggered by the occurrence of a user definable number of processor events
- Displays program flow, critical functions and call sequences
- Can use instrumentation and JVM machine profiling APIs
 - Mixed Mode – shows performance data for Java code and underlying OS APIs
 - Pure – shows performance data only for Java code

VTune: Call Graph, XML Parser

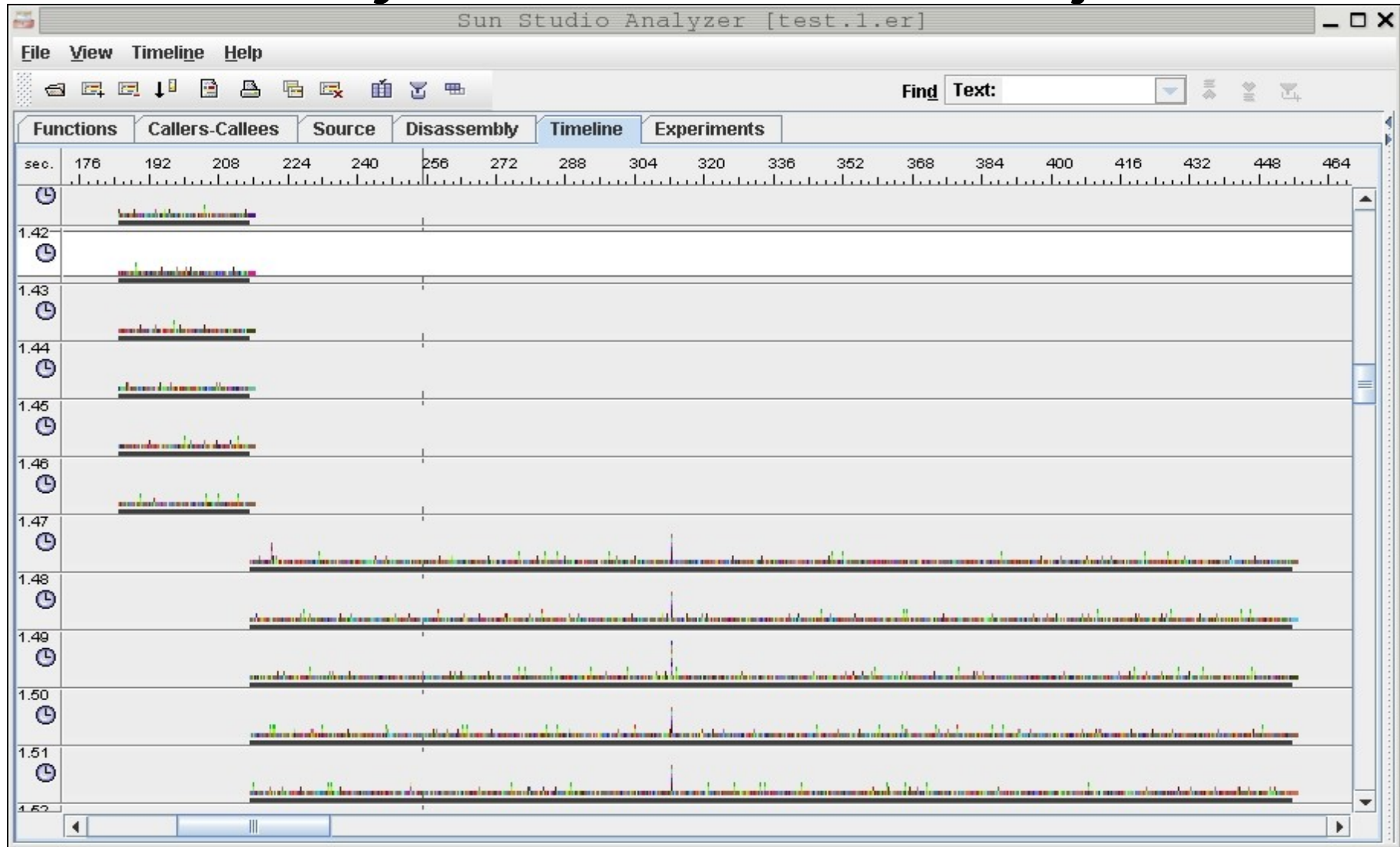




Studio Analyzer: JVM Machine Threads, SPECjbb2005

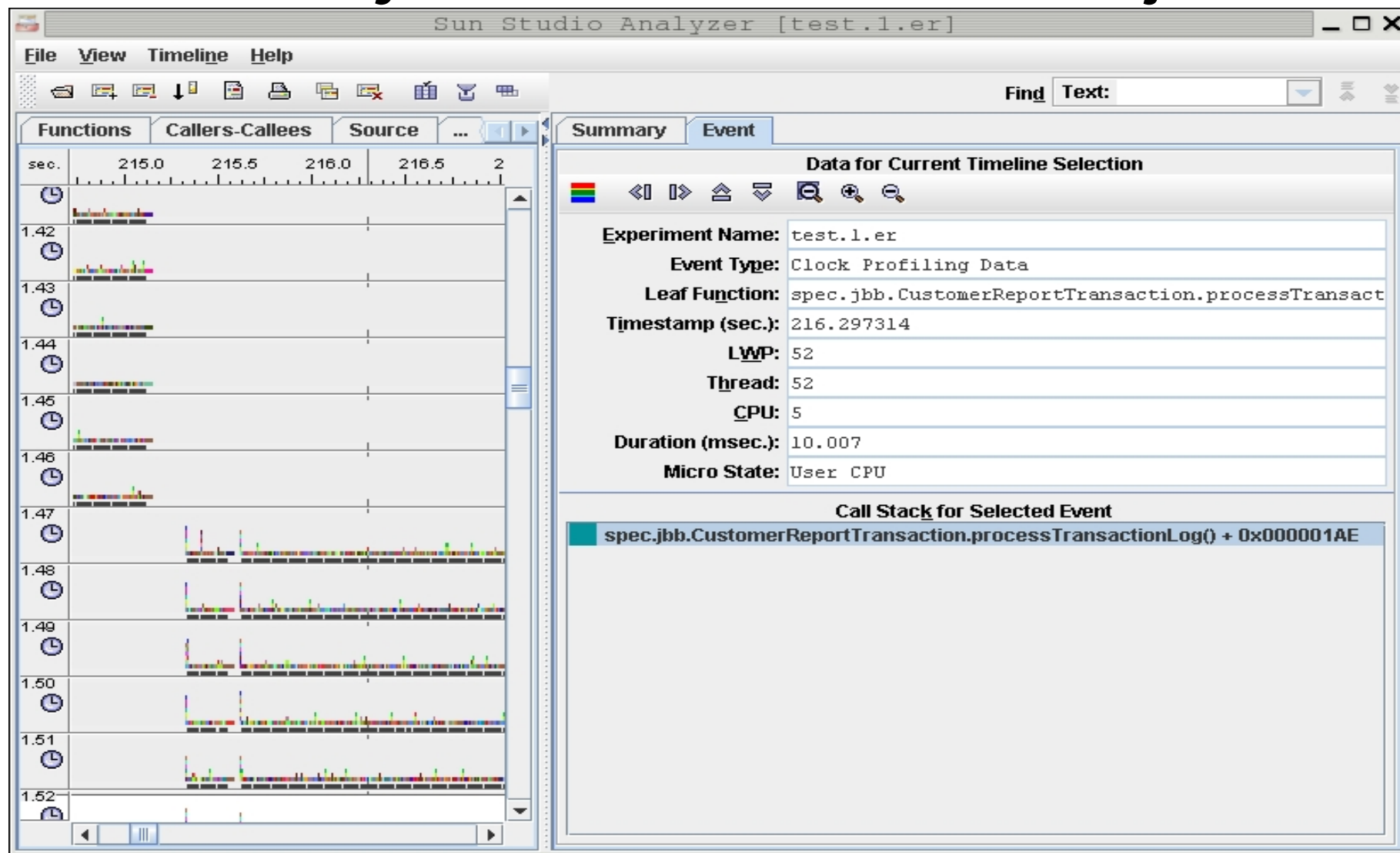


Studio Analyzer: Transition, SPECjbb2005





Studio Analyzer: Stack Trace, SPECjbb2005



Agenda

- Methodology: Hardware, JVM Machine, Class Libraries, Applications
- Tools, Tools, Tools
- Case Study 1: SPECjvm2008, FFT Large
- Case Study 2: SPECjbb2005
- Demo
- Performance Analysis Takeaways
- Java Library and JVM Machine Performance Frontiers

SPECjvm2008: scimark.FFT.large

- scimark is a Java technology-based floating point benchmark from NIST
- FFT is “Fast Fourier Transform”
- One dimensional, in-place algorithm with bit reversal
- $n \cdot \log(n)$ complexity
- Large datasets: 2 million double complex values (32mb)
- Data access pattern is in steps of 2^n ($n=20,19,18,\dots$)
- What's the story?
 - VTune data indicates low Instructions/Clock (IPC)
 - VTune cache miss profiles indicate very large cache miss rate
 - VTune data drill-down shows high cache miss rate inside one tight loop

Problem Analysis

➤ Reads and writes are in 2^n steps

```
double data = new double[2*1024*1024];
for (int j = 0; j < step; j++) {
    double sum = 0;
    for (int i = 0; i < data_size; i += step) {
        sum += data[i+j]; data[i+j] += sum;
    }
}
```

2^n access stride
in the inner loop

Cache Miss

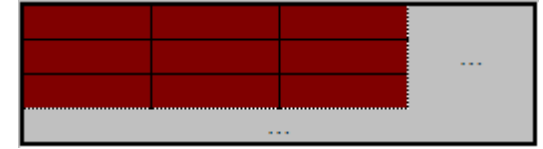
| | | |
|------|--------|--------|
| used | unused | unused |
| used | unused | unused |
| used | unused | unused |



➤ Set-associative caches; an address must be placed in one specific set

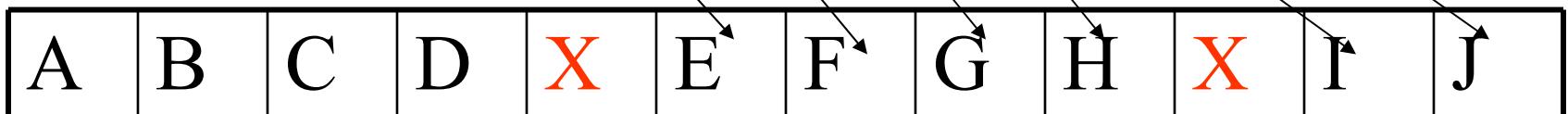
- Low bits of all addresses in a given set are identical, so a 2^n stride maps all addresses in the sequence onto the same set
- Only a small fraction of the cache is used

Change the source code

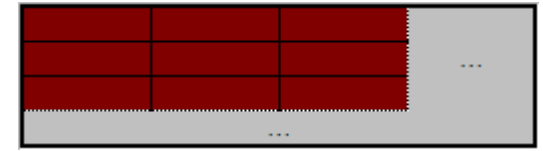


➤ Option 1: insert dummy element(s) into the array every k elements and change code to read/write from the new array correctly: ABCDEFGH => ABCDxEFGHx

- $\text{new_index} = \text{old_index} + \text{floor}(\text{old_index} / k)$
- Increases footprint as well as number of instructions executed per operation but allows entire cache to be used



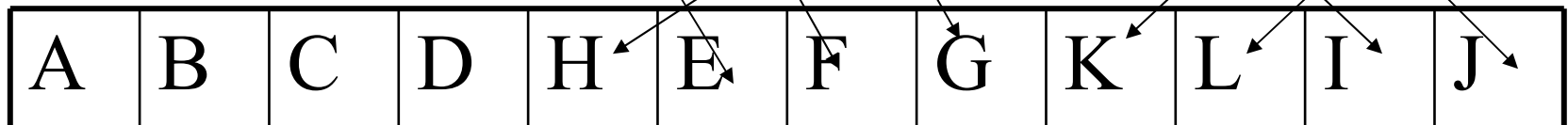
Change the source code



➤ Option 2: move data items around in some pattern:

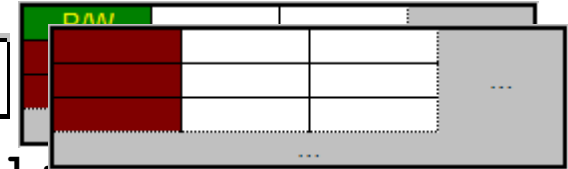
ABCDEFGH IJ KL... => ABCDHEFGKLIJ....

- More complex code necessary to correctly access needed element
=> many more instructions than option 1
- Footprint unchanged



Option 3: Loop Interchange + Scalar Promotion

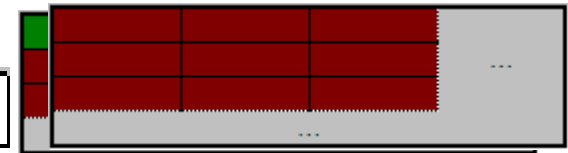
> From



```
double data = new double[2*1024*1024];
for (int j = 0; j < step; j++) {
    double sum = 0;
    for (int i = 0; i < data_size; i += step) {
        sum += data[i+j];
        data[i+j] = sum;
    }
}
```

Cache Miss

> To



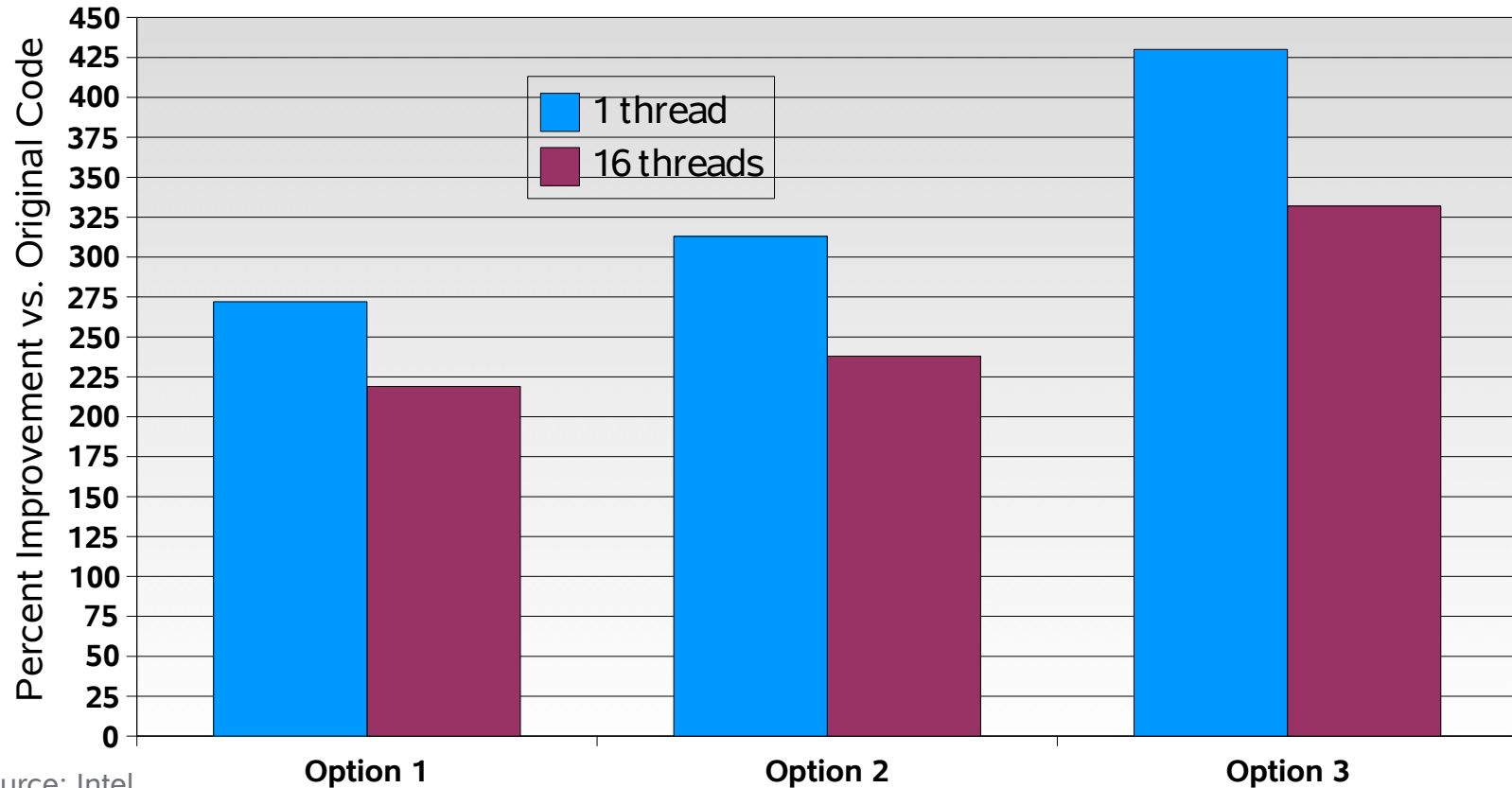
```
double data[] = new double[2*1024*1024];
double sum_array[] = new double[step];
for (int i = 0; i < data_size; i += step) {
    for (int j = 0; j < step; j++) {
        sum_array[j] += data[i+j];
        data[i+j] = sum_array[j];
    }
}
```

Promote sum
to an array

Interchange
loops

Cache Hit

Closing the Loop



Source: Intel

4 chip, 16 core Tigerton 2.93GHz 1066MHz bus

Measurements on other platforms show similar improvements

Option 3 is best

Minor footprint increase, smallest code change

Agenda

- Methodology: Hardware, JVM Machine, Class Libraries, Applications
- Tools, Tools, Tools
- Case Study 1: SPECjvm2008, FFT Large
- Case Study 2: SPECjbb2005
- Demo
- Performance Analysis Takeaways
- Java Library and JVM Machine Performance Frontiers

SPECjbb2005

- Emulates a three-tier client/server warehouse management system, emphasizing the middle tier
- Measures warehouse transaction rate, one thread per warehouse
- In theory, “embarrassingly parallel”
 - Measures hardware, JVM machine and OS scalability
 - Single- and multi-JVM machine run modes
 - Single-JVM machine mode is the real test of scalability
- Object allocation intensive
- Spoiler: *very* memory system sensitive
 - The smaller the caches, the more sensitive
 - The longer the main memory latency, the more sensitive
 - The more NUMA the memory system, the more sensitive

SPECjbb2005: Studio Analyzer, Profile

Sun Studio Analyzer [test.3.er]

File View Timeline Help

Find Text:

Functions Callers-Callees Source Disassembly Timeline Experiments

| Name | User (sec.) | CPU (%) |
|--|-------------|---------|
| <Total> | 18 081.498 | 100.00 |
| spec.jbb.DeliveryTransaction.preprocess() | 4 952.845 | 27.39 |
| spec.jbb.Order.processLines(spec.jbb.Warehouse, short, boolean) | 2 423.956 | 13.41 |
| spec.jbb.CustomerReportTransaction.process() | 1 344.871 | 7.44 |
| spec.jbb.NewOrderTransaction.processTransactionLog() | 633.483 | 3.50 |
| spec.jbb.infra.Util.TransactionLogBuffer.privText(java.lang.String) | 582.928 | 3.22 |
| spec.jbb.infra.Util.XMLTransactionLog.populateXML(spec.jbb.infra.U | 479.025 | 2.65 |
| java.math.BigDecimal.compareTo(java.math.BigDecimal) | 435.735 | 2.41 |
| spec.jbb.infra.Util.XMLTransactionLog.clear() | 427.099 | 2.36 |
| java.math.BigDecimal.layoutChars(boolean) | 329.250 | 1.82 |
| spec.jbb.infra.Util.TransactionLogBuffer.putInt(int, int, int, int) | 307.905 | 1.70 |
| gettimeofday | 304.283 | 1.68 |
| spec.jbb.StockLevelTransaction.process() | 296.948 | 1.64 |
| jshort_disjoint_arraycopy | 289.533 | 1.60 |
| java.math.BigDecimal.multiply(java.math.BigDecimal) | 271.100 | 1.50 |
| java.util.TreeMap\$AscendingSubMap\$AscendingEntrySetView.iterator() | 268.638 | 1.49 |
| spec.jbb.PaymentTransaction.processTransactionLog() | 250.355 | 1.38 |
| arrayof_jshort_disjoint_arraycopy | 237.186 | 1.31 |
| com.sun.org.apache.xerces.internal.dom.ParentNode.internalInsertBe | 232.212 | 1.28 |
| spec.jbb.infra.Util.TransactionLogBuffer.putText(java.lang.String, | 218.413 | 1.21 |

Summary Event

Selected Object:

Name: spec.jbb.DeliveryTran

PC Address: 29:0x000A0550

Size: 14080

Source File: (unknown)

Object File: spec.jbb.DeliveryTran

Load Object: <JAVA_COMPILED_METHOD

Mangled Name: spec.jbb.DeliveryTran

Aliases:

Metrics for Selected Ob

| | Exclusive |
|------------------|-------------------|
| User CPU: | 4952.845 (27.39%) |
| Wait: | 0. (0. %) |
| Total LWP: | 7130.368 (11.86%) |
| System CPU: | 4.643 (19.18%) |
| Wait CPU: | 2172.820 (27.30%) |
| User Lock: | 0. (0. %) |
| Text Page Fault: | 0. (0. %) |
| Data Page Fault: | 0.060 (1.79%) |
| Other Wait: | 0. (0. %) |



SPECjbb2005: Studio Analyzer, -XX:-Inline

Sun Studio Analyzer [test.4.er]

File View Timeline Help

Find Text:

Functions Callers-Callees Source Disassembly Timeline Experiments

| Name | User (sec.) | CPU (%) |
|--|-------------|---------|
| <Total> | 638.026 | 100.00 |
| java.util.HashMap.get(java.lang.Object) | 42.119 | 6.60 |
| java.lang.Integer.equals(java.lang.Object) | 24.457 | 3.83 |
| java.util.TreeMap.successor(java.util.TreeMap\$Entry) | 23.987 | 3.76 |
| spec.jbb.Warehouse.retrieveStock(int) | 16.852 | 2.64 |
| java.lang.String.<init>(java.lang.String) | 16.101 | 2.52 |
| spec.jbb.infra.Util.TransactionLogBuffer.clearBuffer() | 15.781 | 2.47 |
| java.lang.String.getChars(int, int, char[], int) | 14.490 | 2.27 |
| spec.jbb.Orderline.process(spec.jbb.Item, spec.jbb.Stock) | 14.190 | 2.22 |
| spec.jbb.CustomerReportTransaction.process() | 13.459 | 2.11 |
| spec.jbb.DeliveryTransaction.preprocess() | 12.469 | 1.95 |
| com.sun.org.apache.xerces.internal.dom.ParentNode.internalInsertBefore() | 11.778 | 1.85 |
| jshort_disjoint_arraycopy | 11.018 | 1.73 |
| gettimeofday | 10.427 | 1.63 |
| spec.jbb.infra.Util.TransactionLogBuffer.privText(java.lang.String, int) | 10.337 | 1.62 |
| spec.jbb.Orderline.getQuantity() | 10.077 | 1.58 |
| com.sun.org.apache.xerces.internal.dom.CharacterDataImpl.setNodeValueIn | 9.987 | 1.57 |
| spec.jbb.Stock.getQuantity() | 9.427 | 1.48 |
| spec.jbb.Customer.getBalance() | 8.636 | 1.35 |
| com.sun.org.apache.xerces.internal.dom.ParentNode.internalRemoveChild(o | 8.576 | 1.34 |

Summary Event

Selected Object:

Name: java.util.HashMap

PC Address: 29:0x00066790

Size: 352

Source File: /tmp/analyzer.HashM

Object File: java.util.HashMap

Load Object: <JAVA_COMPILED_METE

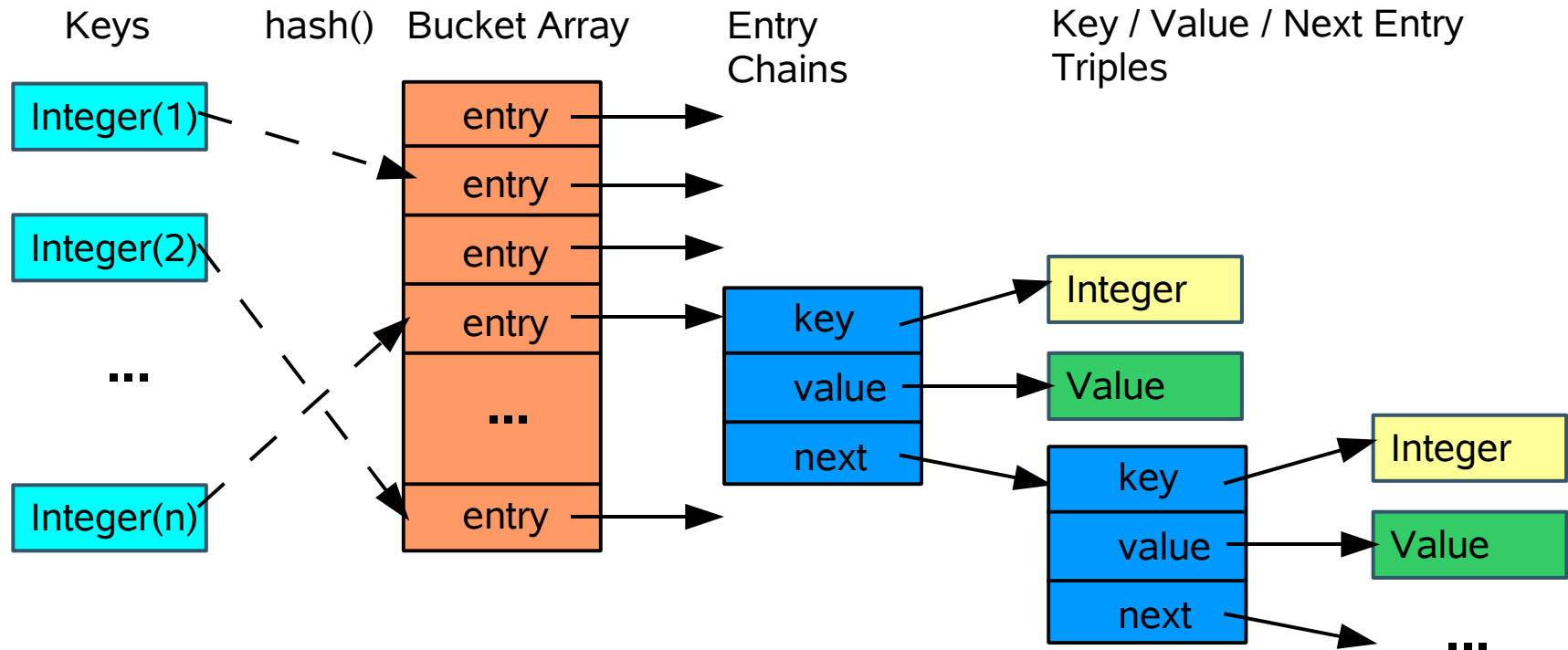
Mangled Name: java.util.HashMap

Aliases:

Metrics for Selected Object

| | Exclusive |
|------------------|-----------------|
| User CPU: | 42.119 (6.60%) |
| Wall: | 0. (0. %) |
| Total LWP: | 42.119 (1.47%) |
| System CPU: | 0. (0. %) |
| Wait CPU: | 0. (0. %) |
| User Lock: | 0. (0. %) |
| Text Page Fault: | 0. (0. %) |
| Data Page Fault: | 0. (0. %) |
| Other Wait: | 0. (0. %) |

HashMap Organization: Integer Keys



- HashMap is implemented as a chained hash table
- Chained hash tables have poor memory access locality
- Even if average chain length is one, could miss on every access: Bucket Array element, Entry, Key and Value objects

SPECjbb2005: HashMap.get(), Misses in Orange

```

public V get(Object key) {
    int hash = hash(key.hashCode());
    for (Entry<K,V> e = table[indexFor(hash, table.length)];
        e != null; e = e.next) {
        Object k;
        if (e.hash == hash &&
            ((k = e.key) == key || key.equals(k)))
            return e.value; // Possible miss
    }
    return null;
}

int index; HashMap table; ...; // Fill up table
Value value = table.get(index); // Many, many get()s!

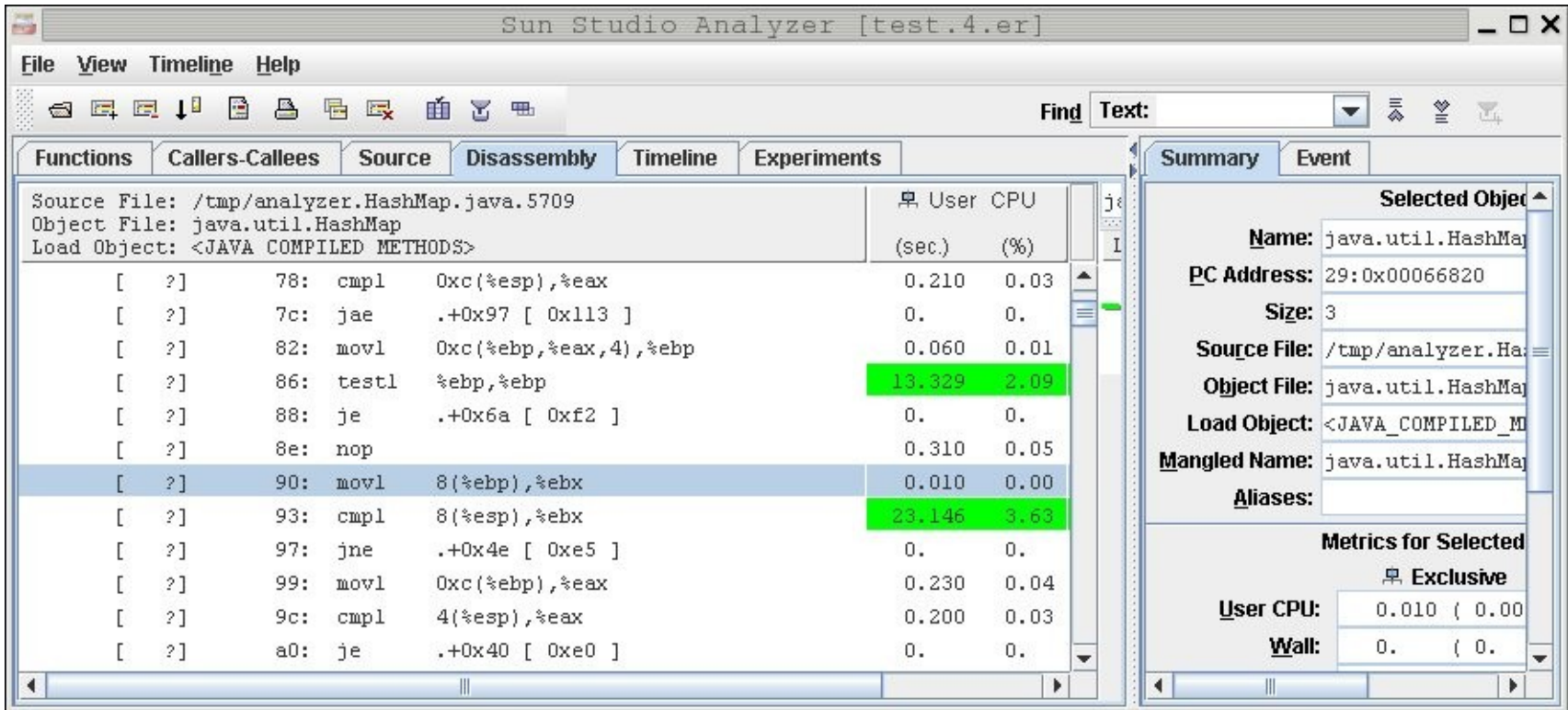
// But 'index' is auto-boxed, so the call
// to get() is actually

Value value = table.get(Integer.valueOf(index));

```

SPECjbb2005: Studio Analyzer, Assembly

- Bucket Array access
 - `table[indexFor(hash, table.length)]`
- Chain Entry access
 - `e.hash`



Sun Studio Analyzer [test.4.er]

File View Timeline Help

Find Text:

Functions Callers-Callees Source **Disassembly** Timeline Experiments

Source File: /tmp/analyser.HashMap.java.5709
Object File: java.util.HashMap
Load Object: <JAVA COMPILED METHODS>

| | | | | User (sec.) | CPU (%) |
|-------|-----|-------|-----------------------|-------------|---------|
| [?] | 78: | cmpl | 0xc(%esp),%eax | 0.210 | 0.03 |
| [?] | 7c: | jae | +.0x97 [0x113] | 0. | 0. |
| [?] | 82: | movl | 0xc(%ebp,%eax,4),%ebp | 0.060 | 0.01 |
| [?] | 86: | testl | %ebp,%ebp | 13.329 | 2.09 |
| [?] | 88: | je | +.0x6a [0xf2] | 0. | 0. |
| [?] | 8e: | nop | | 0.310 | 0.05 |
| [?] | 90: | movl | 8(%ebp),%ebx | 0.010 | 0.00 |
| [?] | 93: | cmpl | 8(%esp),%ebx | 23.146 | 3.63 |
| [?] | 97: | jne | +.0x4e [0xe5] | 0. | 0. |
| [?] | 99: | movl | 0xc(%ebp),%eax | 0.230 | 0.04 |
| [?] | 9c: | cmpl | 4(%esp),%eax | 0.200 | 0.03 |
| [?] | a0: | je | +.0x40 [0xe0] | 0. | 0. |

Summary Event

Selected Object

Name: java.util.HashMap
PC Address: 29:0x00066820
Size: 3
Source File: /tmp/analyser.HashMap.java.5709
Object File: java.util.HashMap
Load Object: <JAVA_COMPILED_METHODS>
Mangled Name: java.util.HashMap
Aliases:

Metrics for Selected

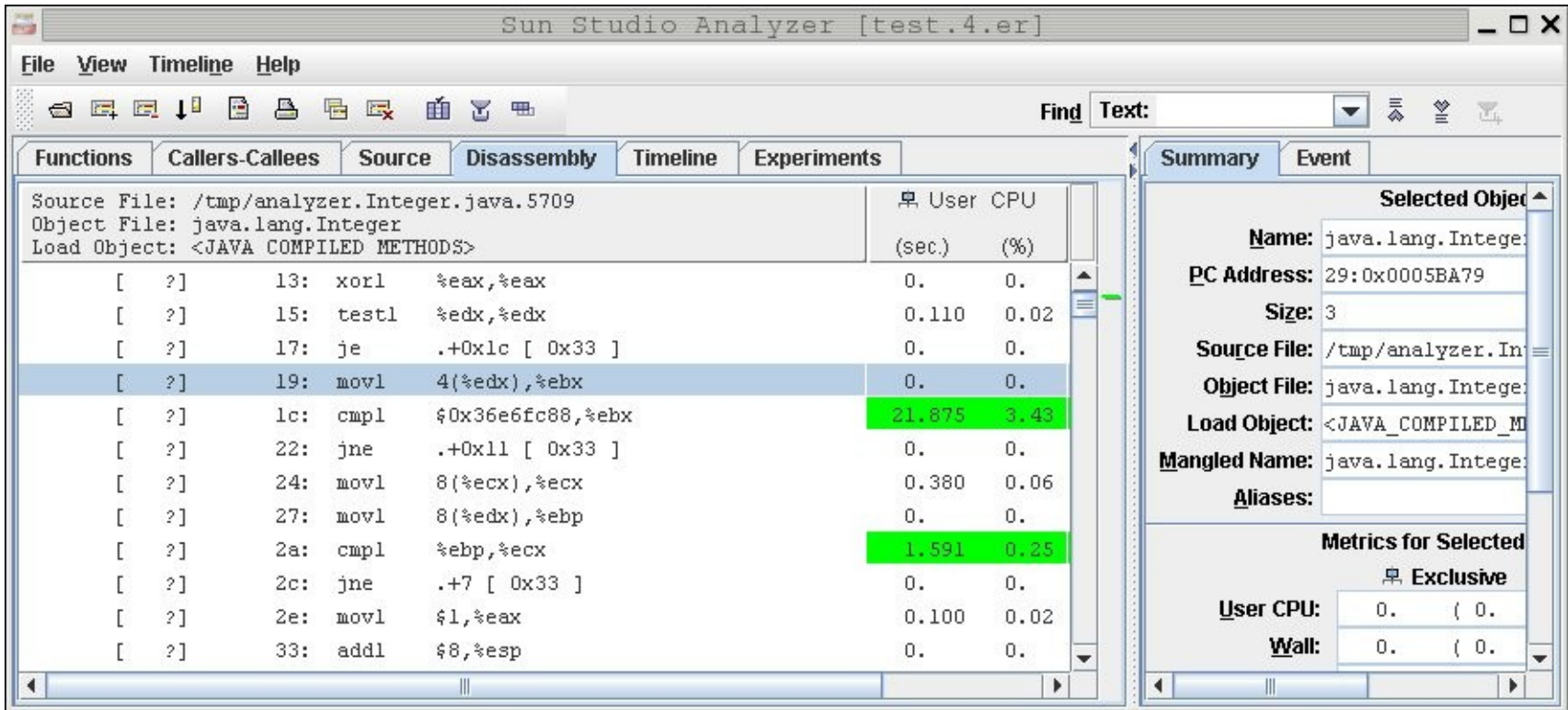
Exclusive

User CPU: 0.010 (0.00)
Wall: 0. (0.)

SPECjbb2005: Studio Analyzer, Assembly

► Key access

- **key.equals(k)**
 - **key.class**
 - **key.value**



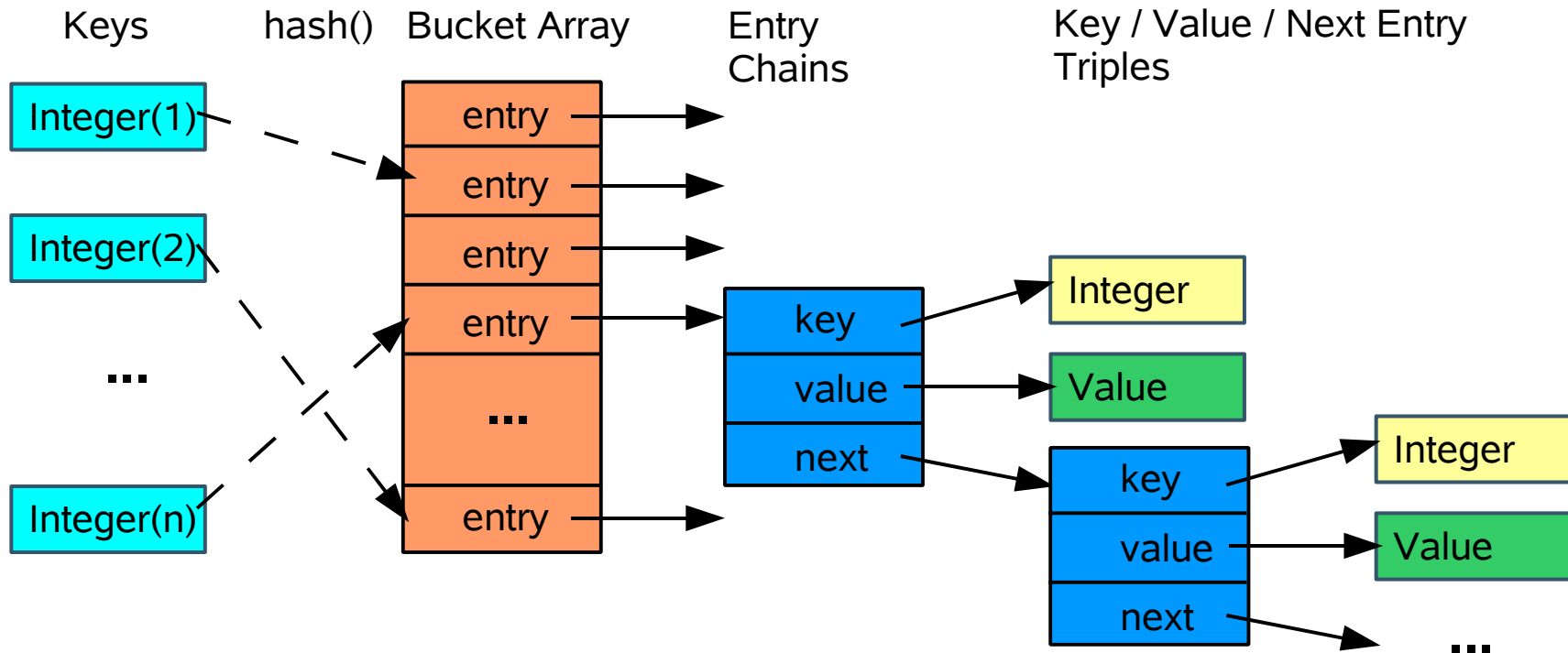
The screenshot shows the Sun Studio Analyzer interface. The main window displays assembly code for the method `java.lang.Integer.equals`. The assembly code is as follows:

| PC | Op | Inst | Time (sec.) | CPU (%) |
|----|-------|-------------------|-------------|---------|
| 13 | xorl | %eax,%eax | 0. | 0. |
| 15 | testl | %edx,%edx | 0.110 | 0.02 |
| 17 | je | .+0x1c [0x33] | 0. | 0. |
| 19 | movl | 4(%edx),%ebx | 0. | 0. |
| 1c | cmpl | \$0x36e6fc88,%ebx | 21.875 | 3.43 |
| 22 | jne | .+0x11 [0x33] | 0. | 0. |
| 24 | movl | 8(%ecx),%ecx | 0.380 | 0.06 |
| 27 | movl | 8(%edx),%ebp | 0. | 0. |
| 2a | cmpl | %ebp,%ecx | 1.591 | 0.25 |
| 2c | jne | .+7 [0x33] | 0. | 0. |
| 2e | movl | \$1,%eax | 0.100 | 0.02 |
| 33 | addl | \$8,%esp | 0. | 0. |

The right-hand pane shows the **Summary** tab for the selected object `java.lang.Integer`. The metrics for the selected object are:

| Metrics for Selected | |
|----------------------|---------|
| Exclusive | |
| User CPU: | 0. (0. |
| Wall: | 0. (0. |

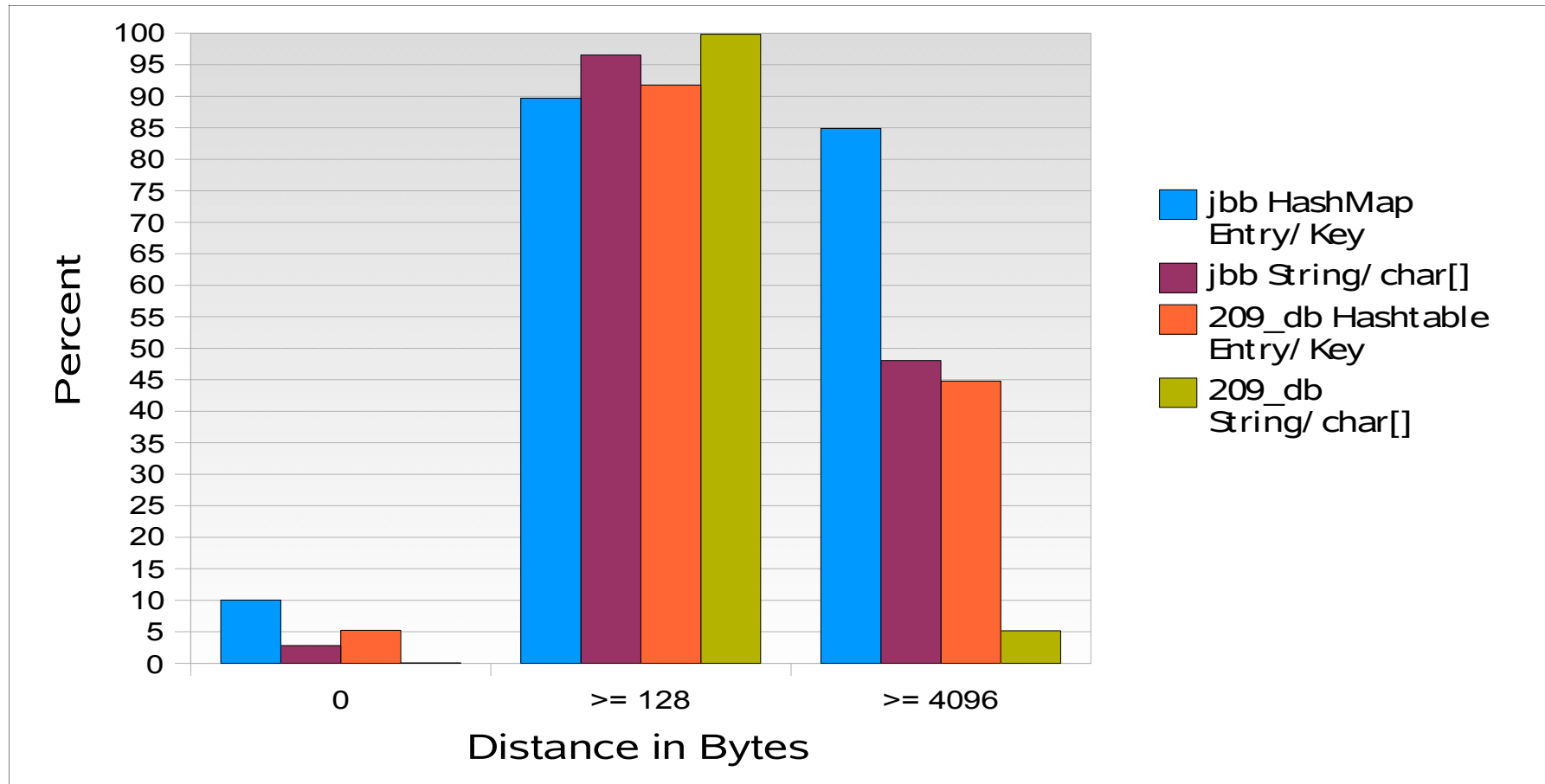
HashMap Organization: Integer Keys



Where in memory are all these guys?

Byte Distance: Parent End to Child Start

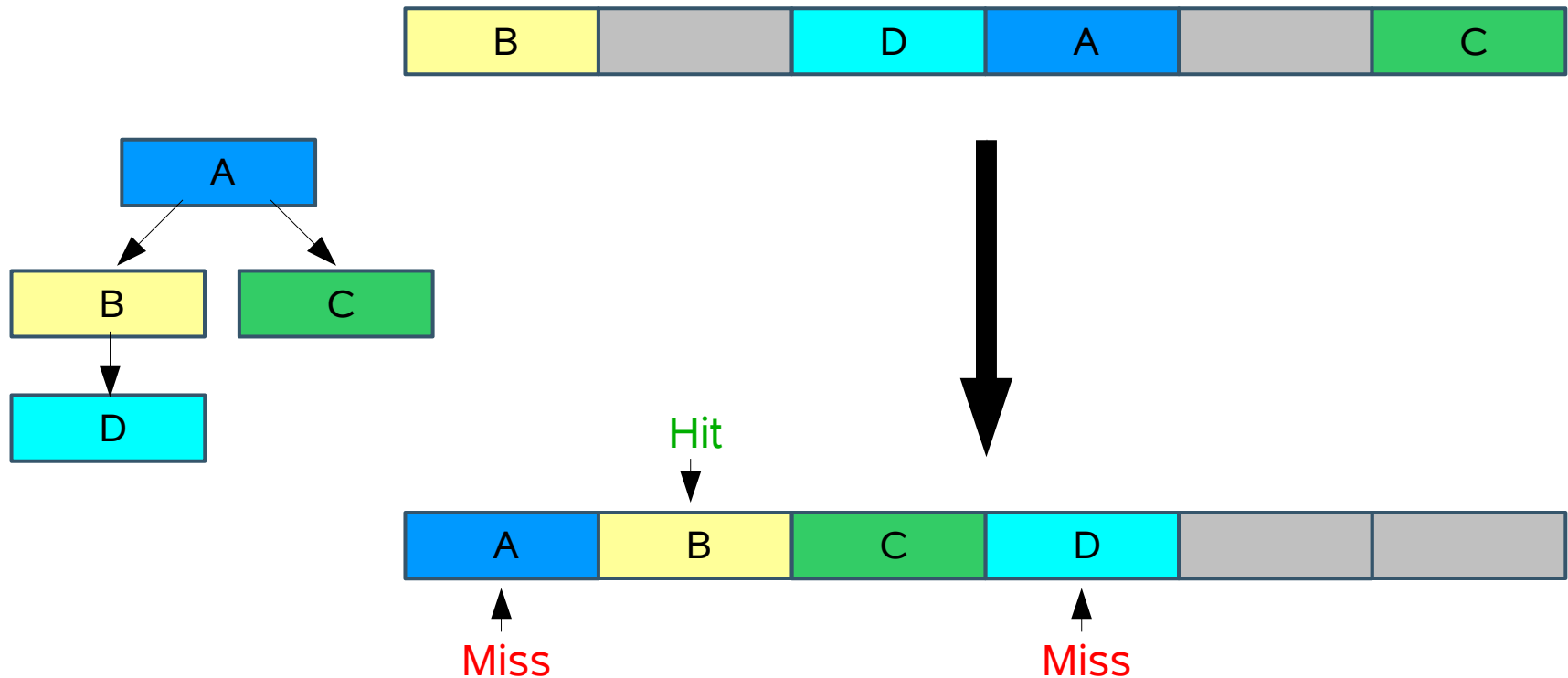
Percent of Object Instance Pairs a Given Distance Apart



Source: Sun Microsystems

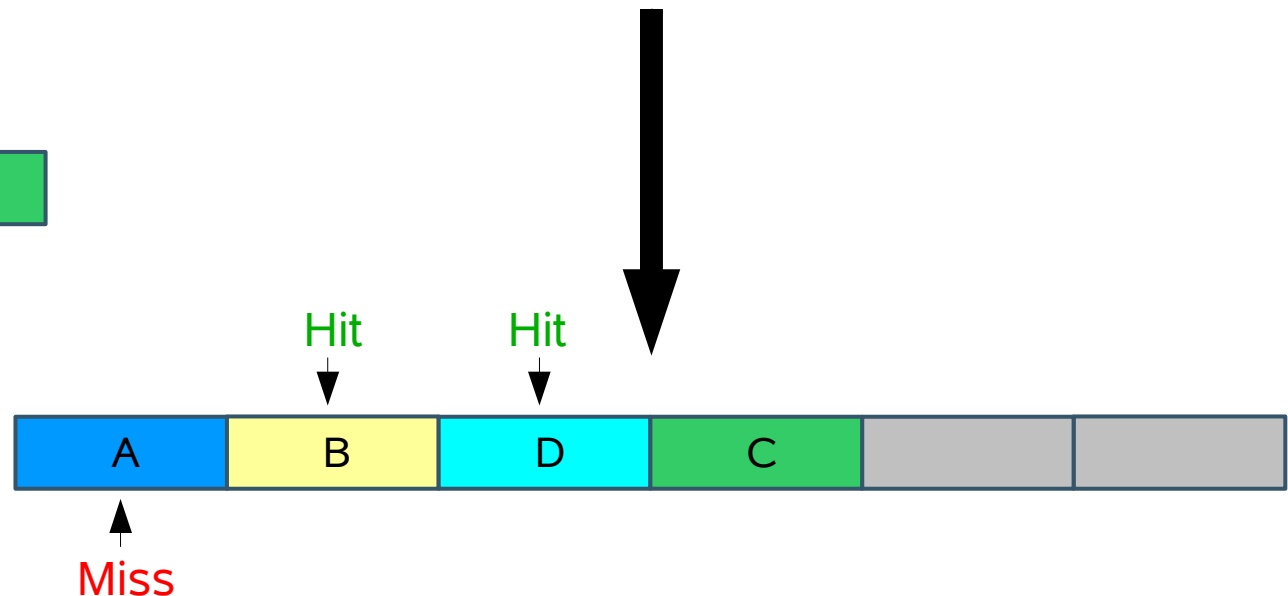
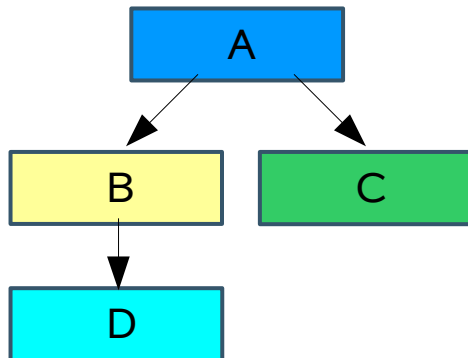
Breadth-First GC Object Copying

- Copy object
- Copy all its children
- For each child, copy all its children, ...

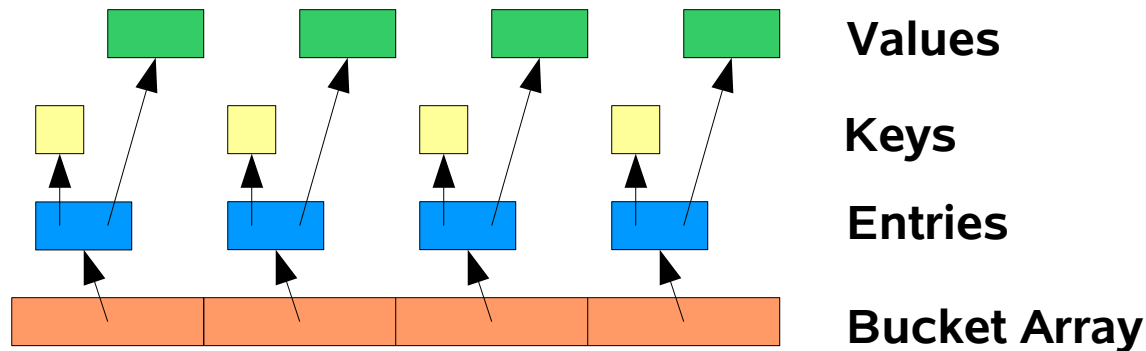


Depth-First GC Object Copying

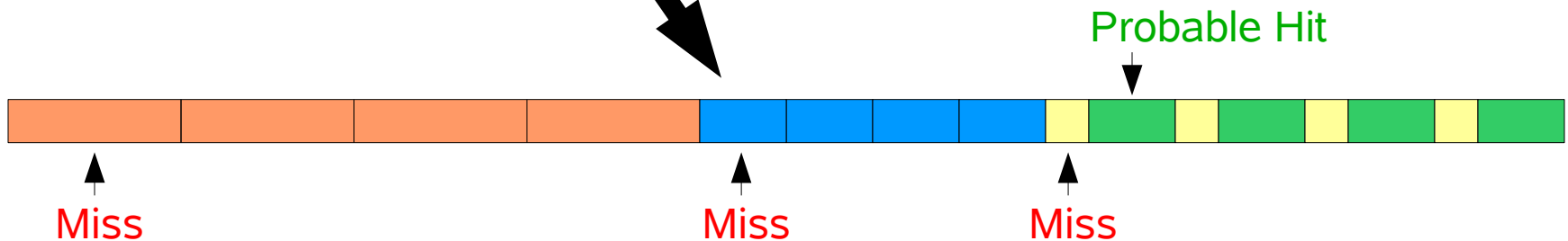
- > Copy object
- > Copy first child
- > Copy first child's first child, ...
- > Copy second child, then second child's first child, ...



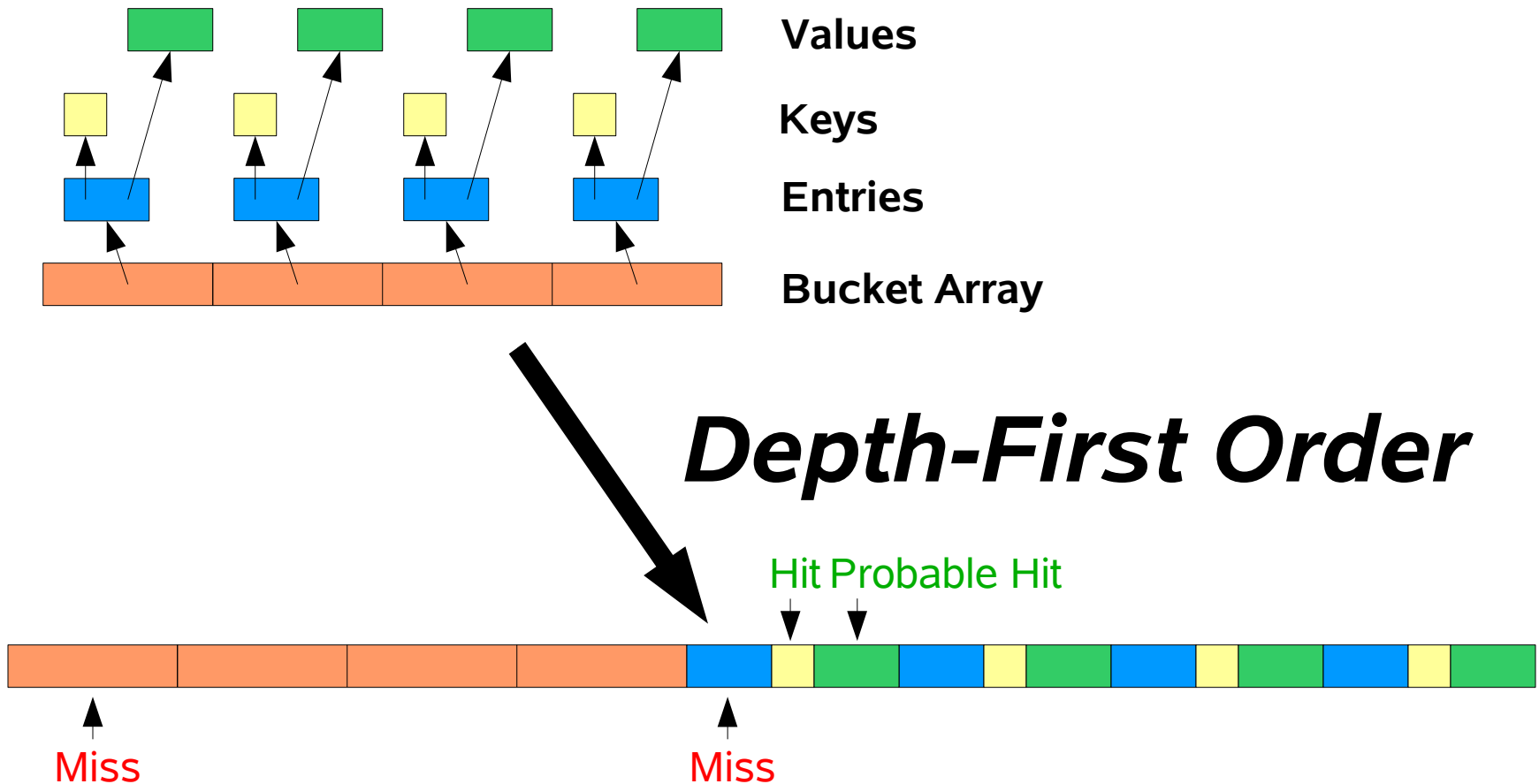
Breadth-First Copying: HashMap



Breadth-First Order

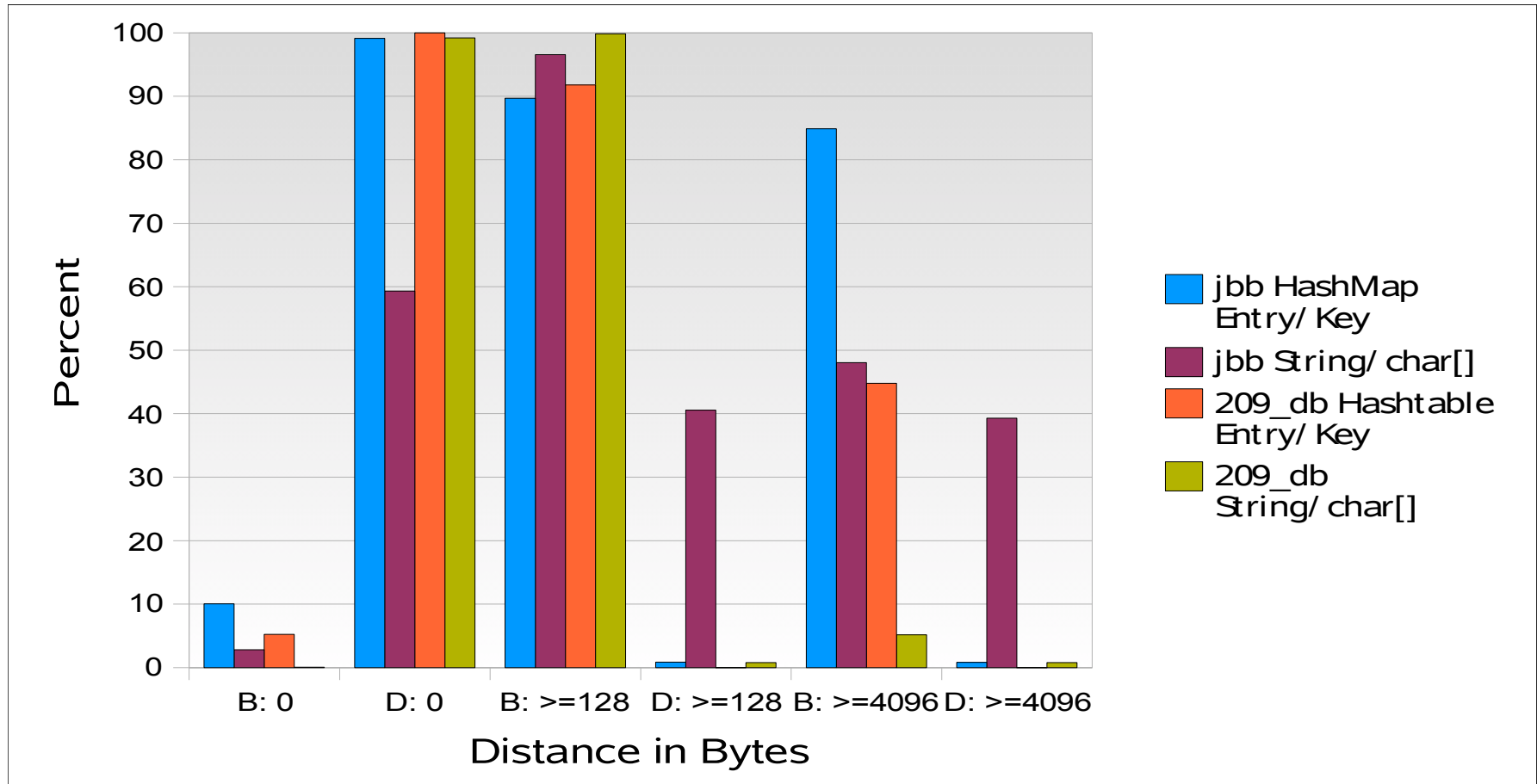


Depth-First Copying: HashMap



Byte Distance: Parent End to Child Start Breadth-First vs. Depth-First

Percent of Object Instance Pairs a Given Distance Apart



Source: Sun Microsystems

Breadth-First vs. Depth-First: SPECjbb2005

Sun Studio Analyzer [test.4.er]

File View Timeline Help

Find Text:

Functions Callers-Callees Source Disassembly Timeline Experiments

| Name | User (sec.) | CPU (%) |
|---|-------------|---------|
| <Total> | 638.026 | 100.00 |
| java.util.HashMap.get(java.lang.Object) | 42.119 | 6.60 |
| java.lang.Integer.equals(java.lang.Object) | 24.457 | 3.83 |
| java.util.TreeMap.successor(java.util.TreeMap\$Entry) | 23.987 | 3.76 |

Summary Event

Selected Object

Name: java.lang.Integer

PC Address: 29:0x0005BA60

Size: 64

Source File: /tmp/analyzer.Int

Object File: java.lang.Integer

Sun Studio Analyzer [test.2.er]

File View Timeline Help

Find Text:

Functions Callers-Callees Source Disassembly Timeline Experiments

| Name | User (sec.) | CPU (%) |
|--|-------------|---------|
| spec.jbb.Orderline.getQuantity() | 11.098 | 1.74 |
| java.lang.Integer.equals(java.lang.Object) | 10.958 | 1.72 |
| jshort_disjoint_arraycopy | 10.647 | 1.67 |
| gettimeofday | 10.517 | 1.65 |

Summary Event

Selected Object

Name: java.lang.Integer

PC Address: 29:0x0005BA20

Size: 64

Source File: /tmp/analyzer.Int

Object File: java.lang.Integer

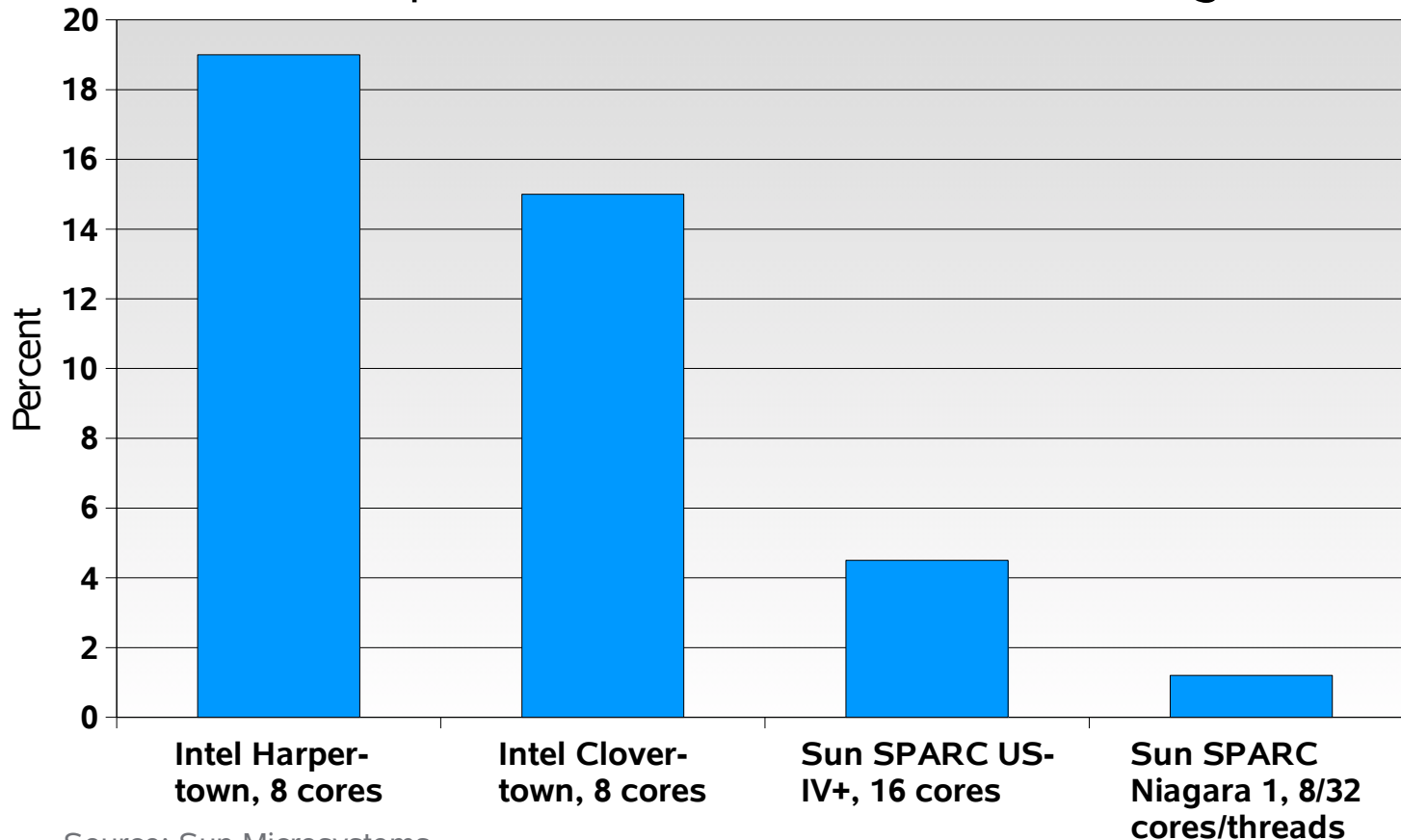
Breadth-First vs. Depth-First: SPECjbb2005

Relative improvement, *not* absolute: see spec.org for the latter

Benefit varies by platform

Not just HashMap! TreeMap, String, etc. also benefit

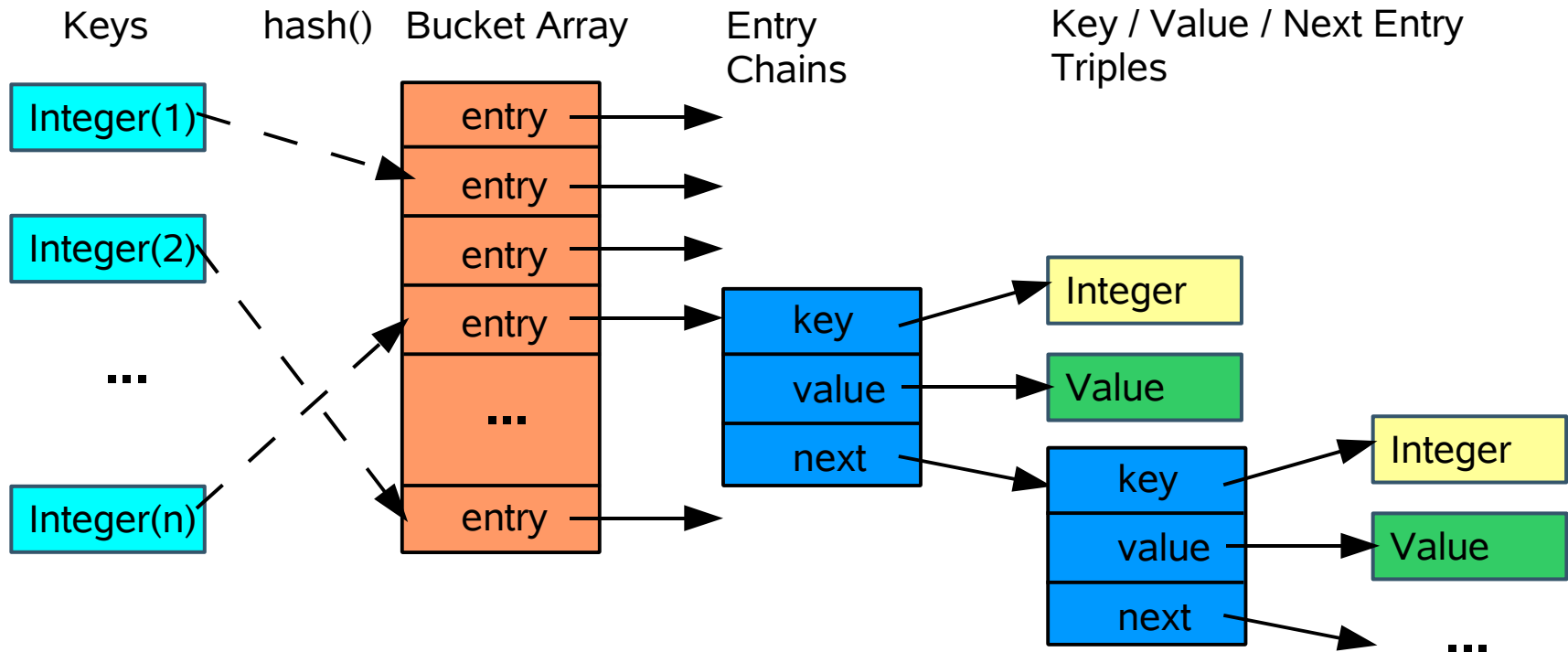
Depth-First Percent Improvement Over Breadth-First, Single JVM Machine



Breadth-First vs. Depth-First

- Any static policy will hurt some applications and help others
- A common pattern in Java technology applications is object arrays pointing to nodes
- Depth-First is better for this pattern
- Little or no degradation with respect to breadth-first on all other performance benchmarks we can get our hands on
- Many, e.g. SPECjvm98's 209_db, are much faster with depth-first
 - Strings and their char[]'s are co-located 100% of the time

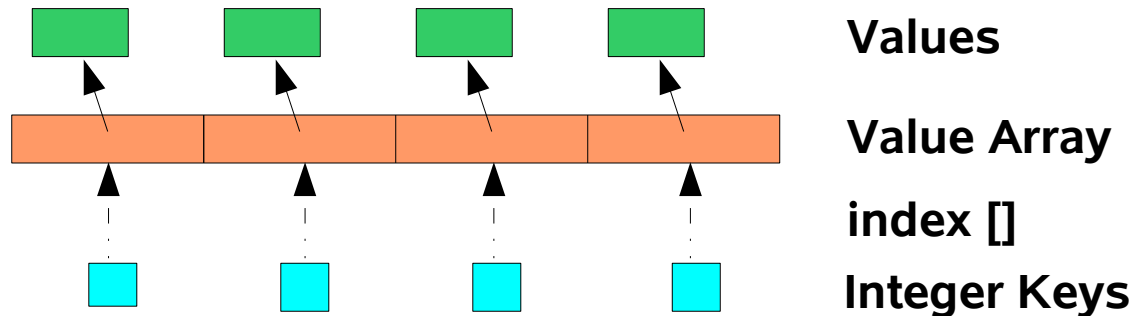
But Wait, There's More! Remember This?



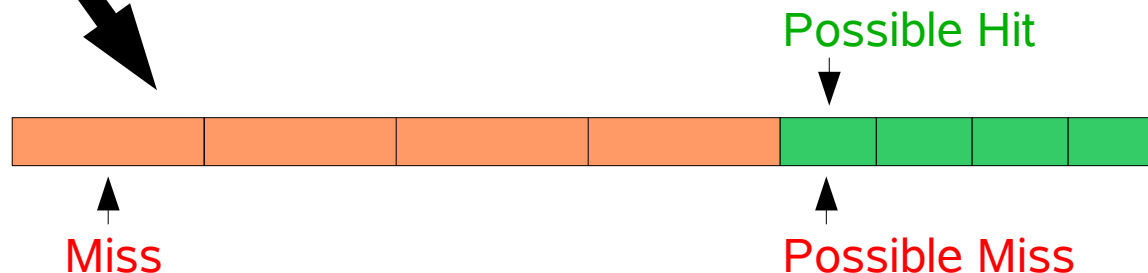
There *must* be a better way!

Yes, There Is: Specialize for Integer Keys

- Front the whole business with an Integer -> Value array cache, punt to backing HashMap on miss
- If front cache is big enough, the backing HashMap will stay out of processor cache: copying order doesn't matter



Breadth or Depth-First Order



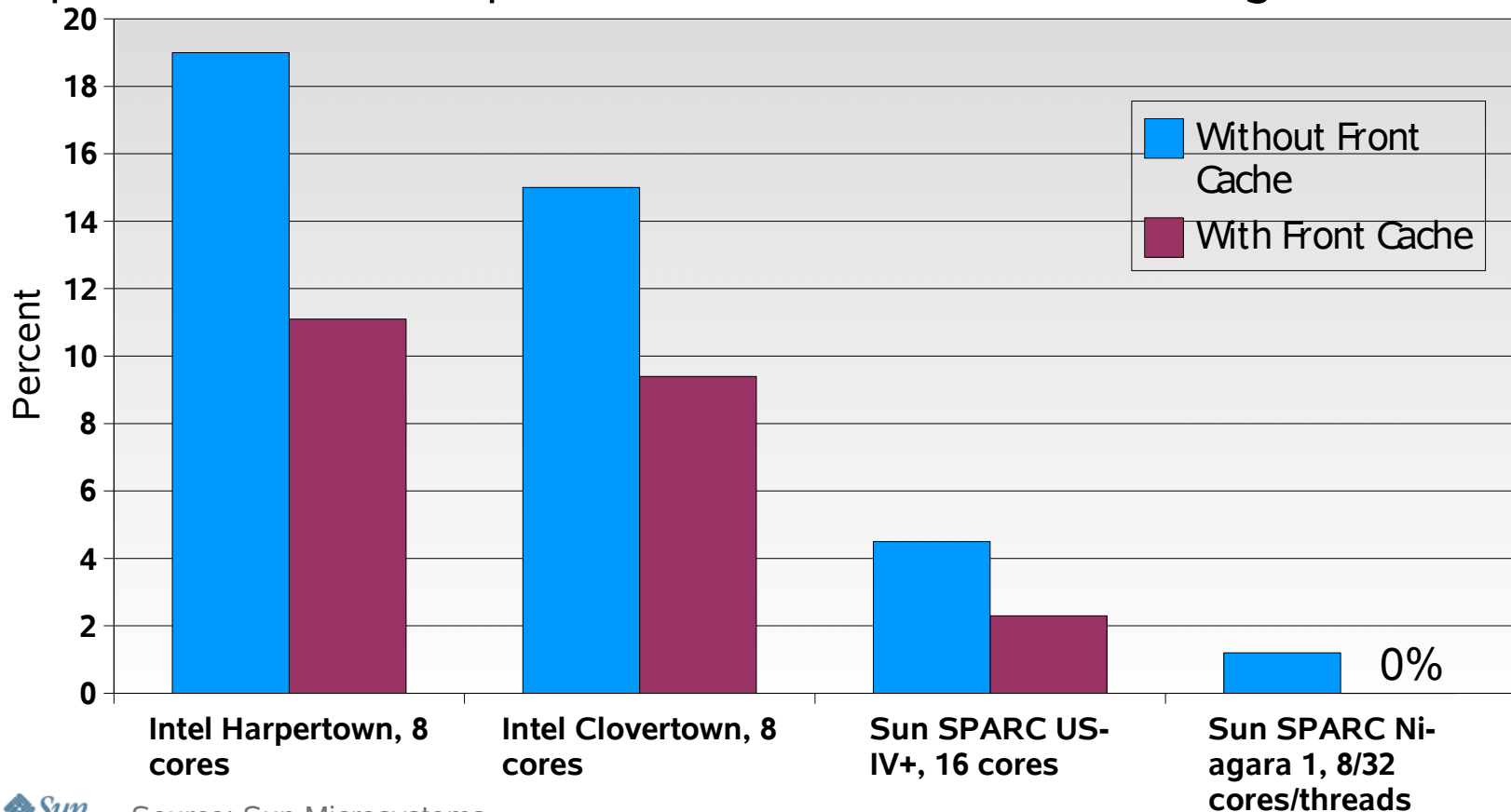
Breadth-First vs. Depth-First: SPECjbb2005

Relative improvement, *not* absolute: see spec.org for the latter

Benefit varies by platform

Not just HashMap! TreeMap, String, etc. also benefit

Depth-First Percent Improvement Over Breadth-First, Single JVM Machine



Demo

A large, light blue arrow pointing to the right, positioned behind the word "DEMO".

DEMO

Agenda

- Methodology: Hardware, JVM Machine, Class Libraries, Applications
- Tools, Tools, Tools
- Case Study 1: SPECjvm2008, FFT Large
- Case Study 2: SPECjbb2005
- Demo
- Performance Analysis Takeaways
- Java Library and JVM Machine Performance Frontiers

Performance Analysis Takeaways

- Incorporate performance monitoring and analysis into the development cycle
 - Including, to the extent possible, production
- Use the available tools: they work really well
- Learn and apply basic hardware design knowledge
- The memory system is not your friend
 - Most performance problems can be traced to its limitations
- Results vary by platform
 - Optimizing cache utilization matters most when average memory access time is high relative to processor speed
 - Test on as many platforms as possible
 - WORA => do your best to eliminate platform to platform variability

Agenda

- Methodology: Hardware, JVM Machine, Class Libraries, Applications
- Tools, Tools, Tools
- Case Study 1: SPECjvm2008, FFT Large
- Case Study 2: SPECjbb2005
- Demo
- Performance Analysis Takeaways
- Java Library and JVM Machine Performance Frontiers

Java Library and JVM Machine Performance Frontiers

- Completely dynamic and adaptive execution
 - Hardware-assisted always-on dynamic profiling
 - Adaptive garbage collection
 - Predictability and throughput goals
 - Real-time
- Ergonomics writ large
 - Execution history recording and feedback
- High performance computing
 - Photo-real graphics generation, speech analysis and generation, financial analysis
 - Auto-vectorization and concurrentization
 - Massively parallel Java platform
- Startup and footprint: minimize class loading overhead
 - Eliminate static variables and initializers, horizontal class dependencies
 - Minimize object allocation during startup
- Memory-optimal data structures: wide-node TreeMap, B+Trees

THANK YOU

Kumar Shiv, Intel
Paul Hohensee, Sun Microsystems

TS-6434

