
ORACLE

JavaOne™

ORACLE

CON2654: Java Performance: Hardware, Structures, and Algorithms

CREATE THE FUTURE

- Charlie Hunt
  HotSpot JVM Engineering
- Darryl Gove
  Compiler Performance Engineering

- 29 September, 2014

JavaOne ORACLE

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Oracle Solaris Studio

Java One Sessions	
Mon, 5:30 – 6:30pm	**Java Performance: Hardware, Structures, and Algorithms** [CON2654], Hilton Imperial Ballroom A
Wed, 3 – 4pm	**Simplifying Development of Mixed-Language Java and C++ Applications** [CON8109], Hilton Continental Ballroom B
OOW Sessions	
Wed, 4:45 – 5:30pm	**Engineering Insights: Best Practices for Optimizing Oracle Software for Oracle Hardware** [CON8108], Intercontinental Grand Ballroom C
Thurs, 12 – 12:45pm	**Code Analysis Tools for Achieving Consistent, Secure, and Reliable Product Quality** [CON8009], Intercontinental B
Wed, 11:45 - 12:45pm	**Create Quality, Secure, High-Performing Applications on Oracle Solaris** [HOL9805], Hotel Nikko - Mendocino I/II

- http://www.oracle.com/goto/solarisstudio

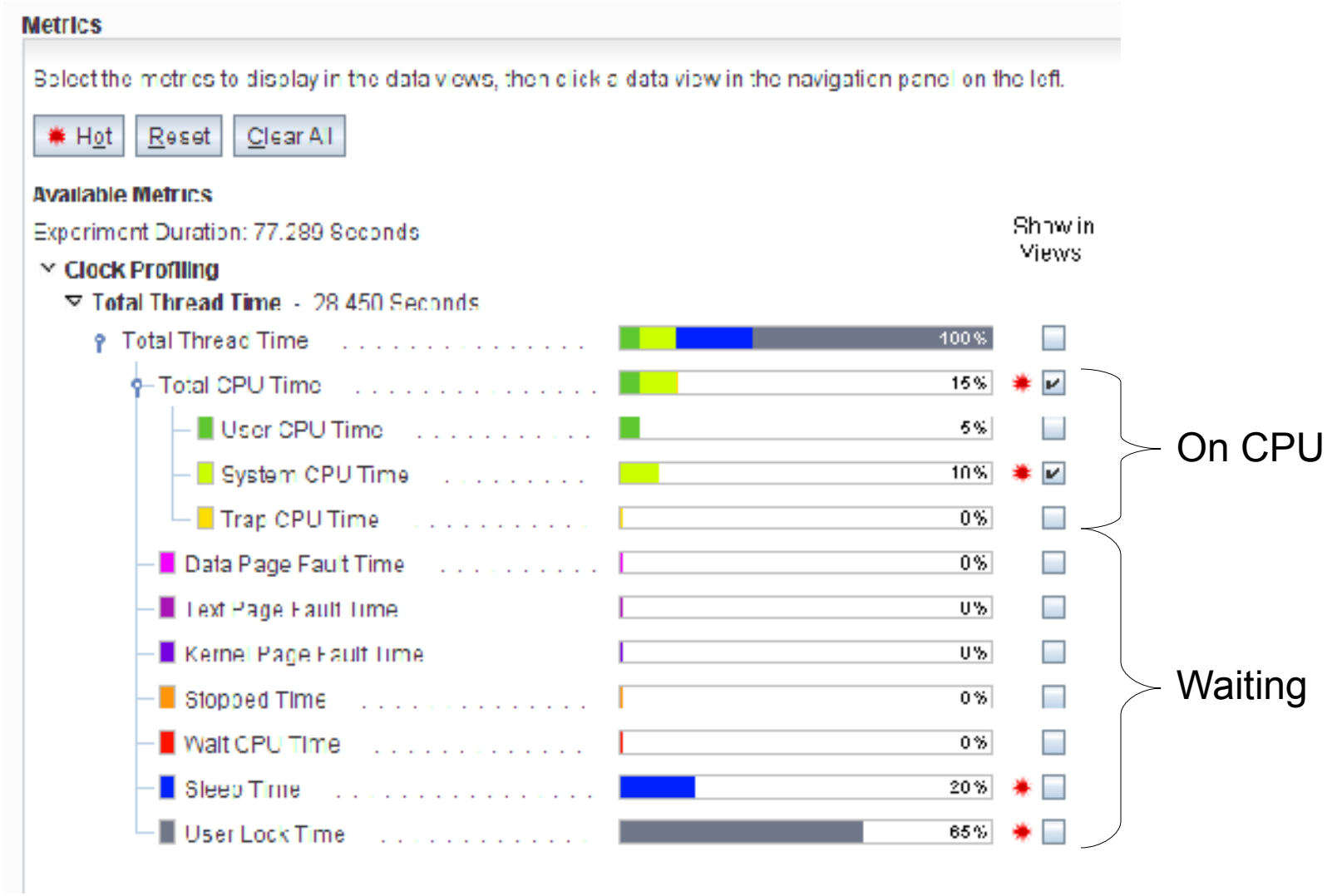
JavaOne

Program Agenda

1. Where to look for performance
2. The impact of hardware
3. Appropriate data structures
4. Efficient implementations
5. Concluding remarks

Where to Look for Performance

Overview



- Overview captures where the threads spend time
- Need to identify chunks of “time” to reduce.

Source Code View (High System Time)

Incl. Total CPU (sec.)	Incl. Sys. CPU (sec.)	Source File: flush.java Object File: flush.class (found as test.1.er/archives, Load Object: flush.class (found as test.1.er/archives,
0.	0.	10. for (int i=0; i<1000000; i++)
		11. {
3.402	2.362	12. out.write('a'); // one char
		13. }
0.	0.	14. out.close();
		15. }
0.	0.	16. catch (Exception e) {}

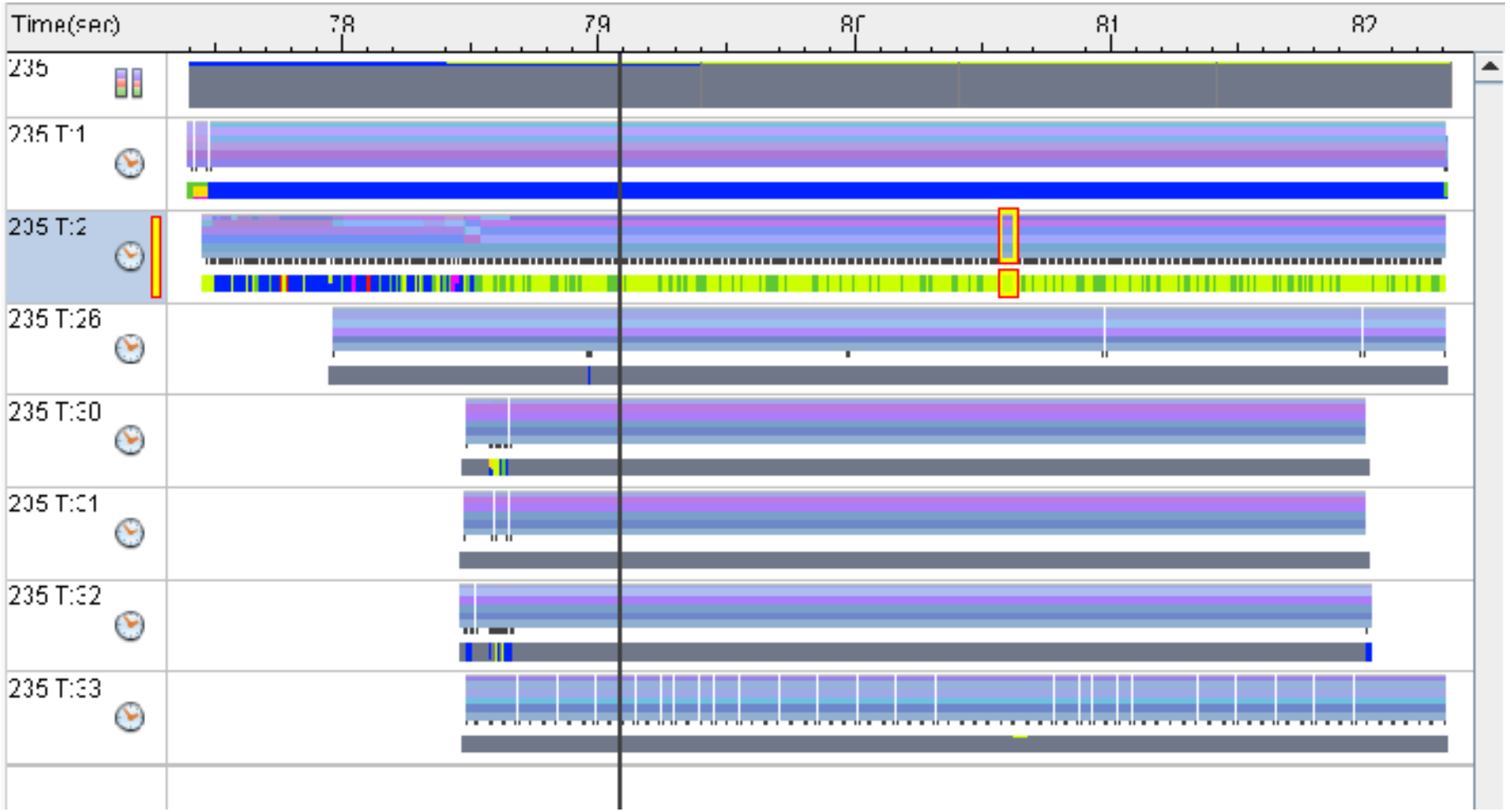
- Substantial system time spent on write of single byte

Source Code View (Reduced System Time)

Incl. Total CPU (sec.)	Incl. Sys. CPU (sec.)	Source File: flush.java Object File: flush.class (found as test.1.er/archives, Load Object: flush.class (found as test.1.er/archives,
0.	0.	20. String text = "abcdefghij";
0.	0.	21. byte data[] = text.getBytes();
0.	0.	22. for (int i=0; i<100000; i++)
		23. {
		24.
0.340	0.250	25. out.write(data,0,10); // 10 chars
		26. }
0.	0.	27. out.close();
		28. }
0.	0.	29. catch (Exception e) {}

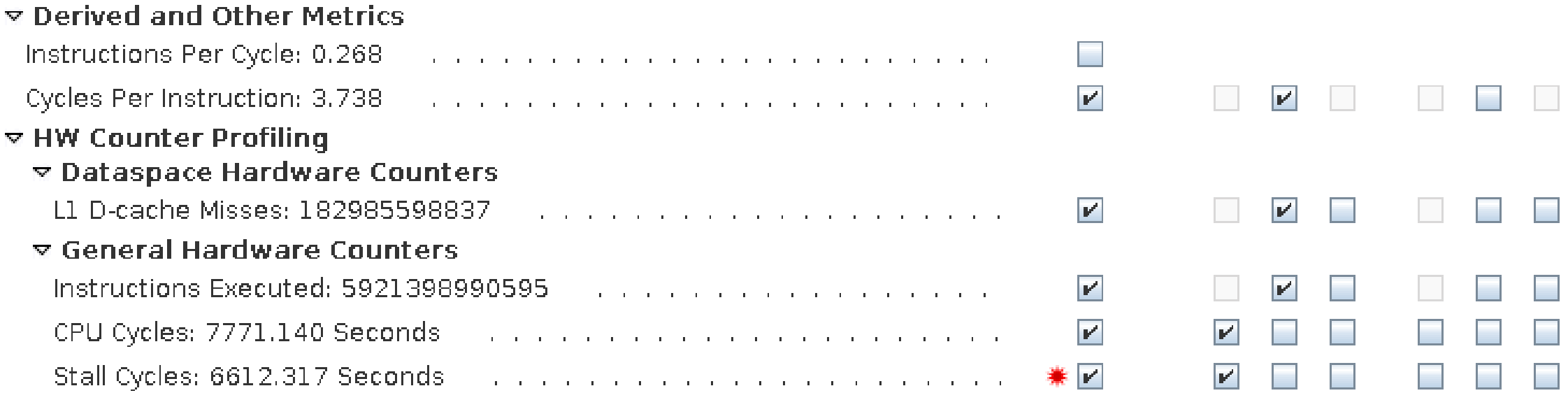
- Increasing size of writes (10x) reduces system time (10x)

Timeline View



- Sleep and User lock time often come from “idle” threads

Hardware Counters



- Hardware counters = processor events
- Can “explain” observed performance

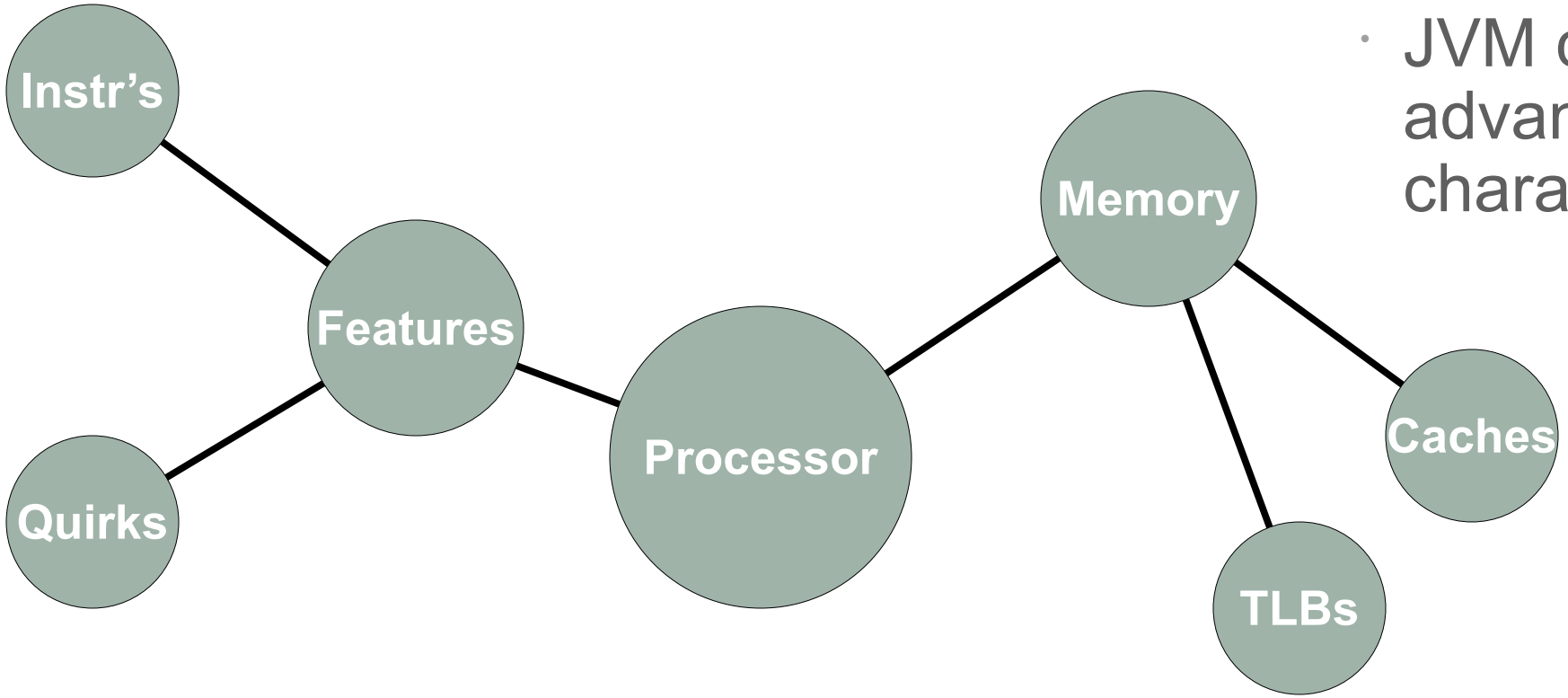
High CPI Indicates Stalls

Excl. User CPU (sec.)	Excl. CPI	Name
896.787	35.360	<Total>
27.029	143.224	java.util.TreeMap$ValueIterator
869.758	34.536	java.util.HashMap.getNode(int,

- Higher CPI indicates more stall time
- But time indicates benefit of fixing it

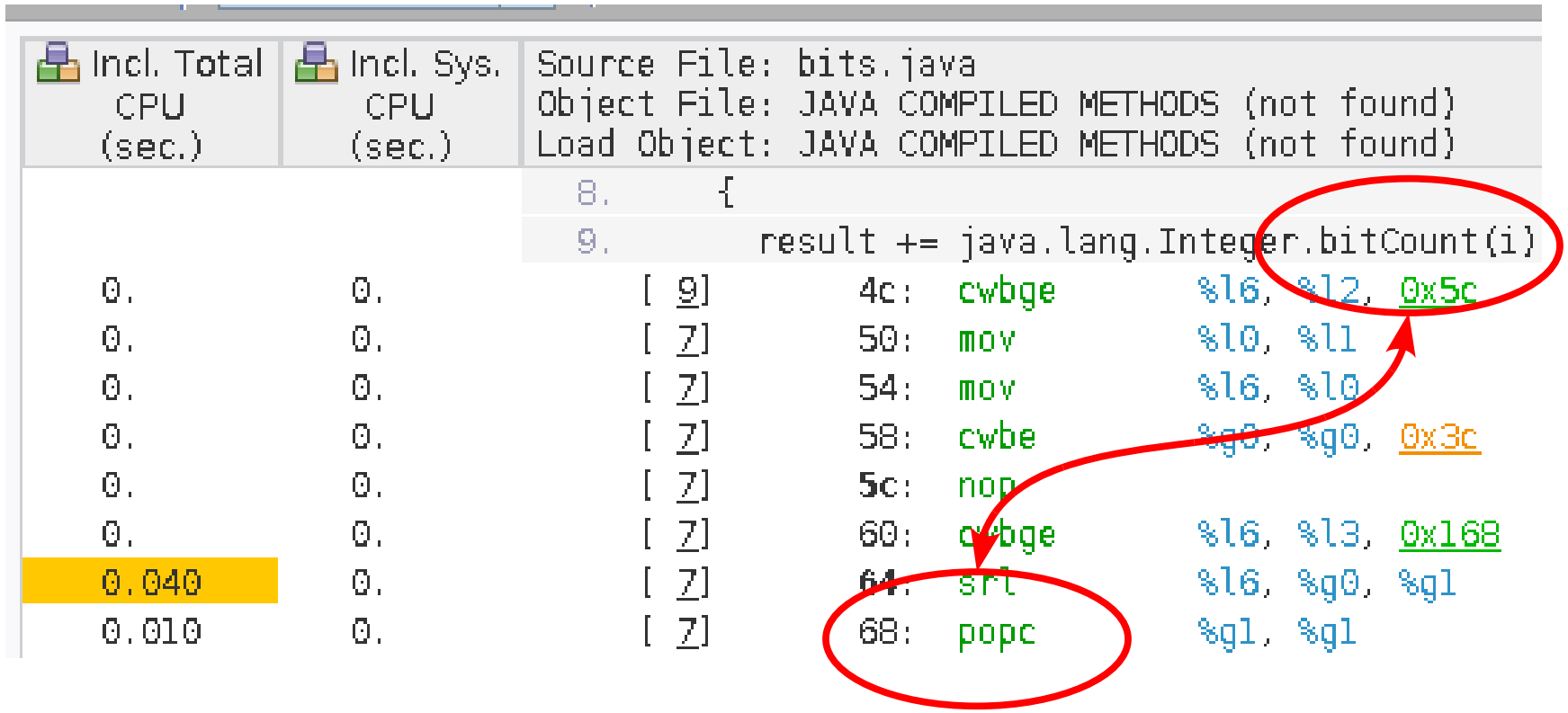
The Impact of Hardware

Hardware and the JVM



- JVM can take advantage of some characteristics

Hardware Intrinsics



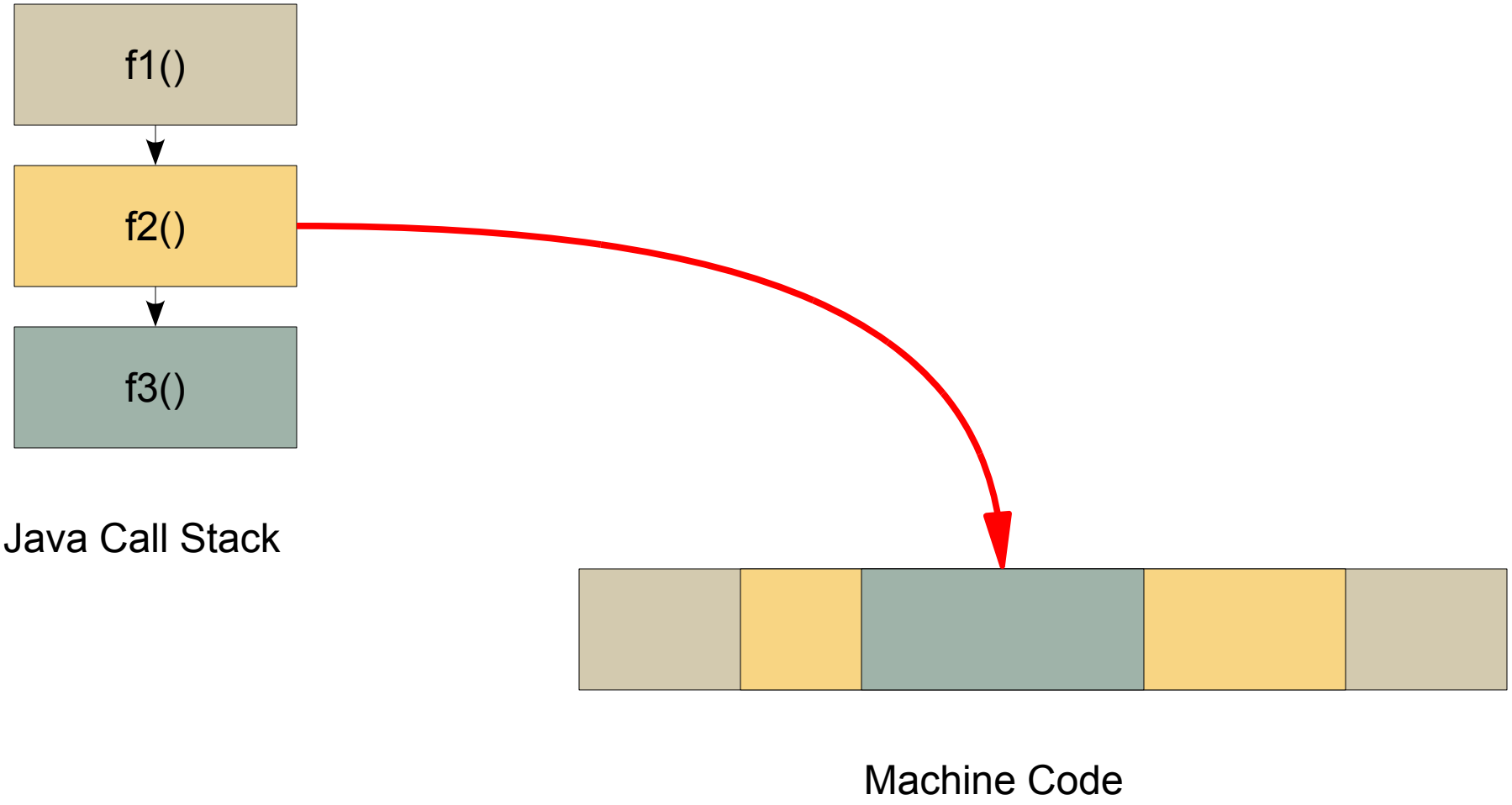
- JVM converts bitCount to popc instruction

Hardware Intrinsics in the VM

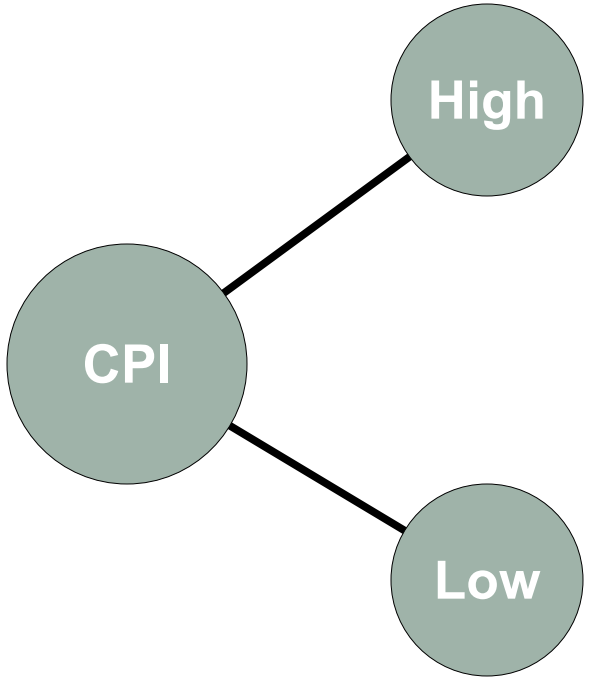
```
2341    switch (id) {
2342    case vmIntrinsics::_numberOfLeadingZeros_i:    n = new (C) CountL
2343    case vmIntrinsics::_numberOfLeadingZeros_l:    n = new (C) CountL
2344    case vmIntrinsics::_numberOfTrailingZeros_i:   n = new (C) CountT
2345    case vmIntrinsics::_numberOfTrailingZeros_l:   n = new (C) CountT
2346    case vmIntrinsics::_bitCount_i:                n = new (C) PopCou
2347    case vmIntrinsics::_bitCount_l:                n = new (C) PopCou
2348    case vmIntrinsics::_reverseBytes_c:            n = new (C) Revers
```

http://hg.openjdk.java.net/jdk8/jdk8/hotspot/file/87ee5ee27509/src/share/vm/opto/library_call.cpp

Inlining and optimisation



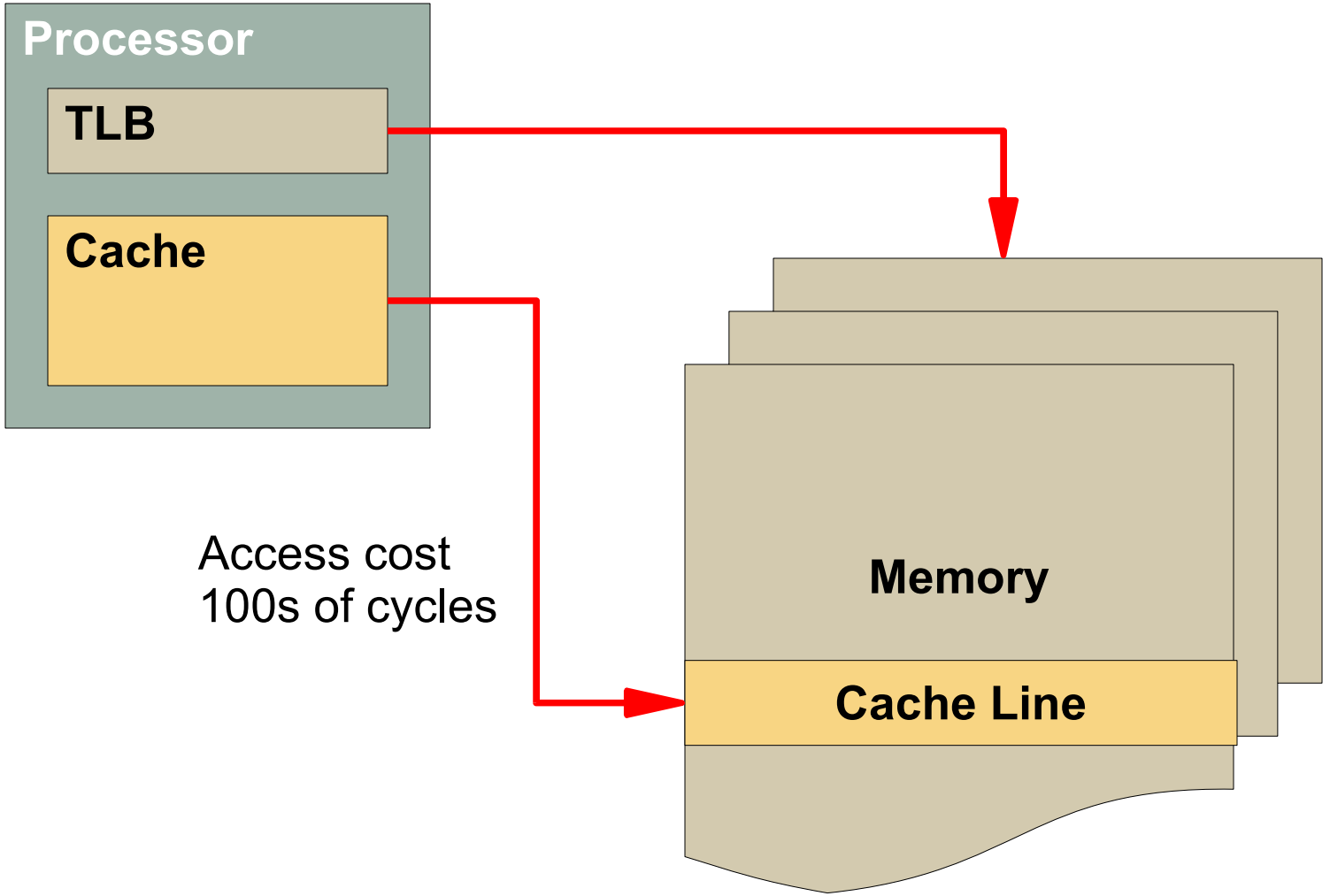
Generalisation of CPI & Performance



- High normally indicates **memory stalls**
  Suggests data structure efficiency

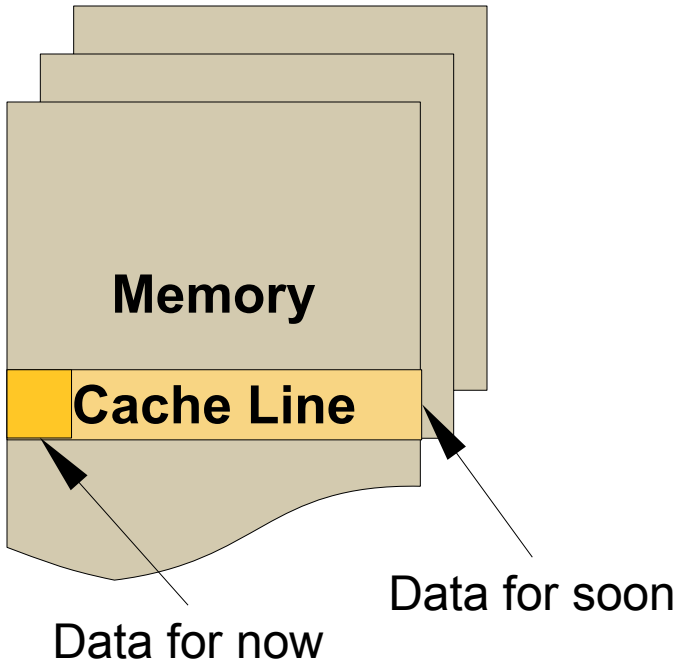
- Low CPI is **instruction issue** limited.
  Suggests algorithm efficiency

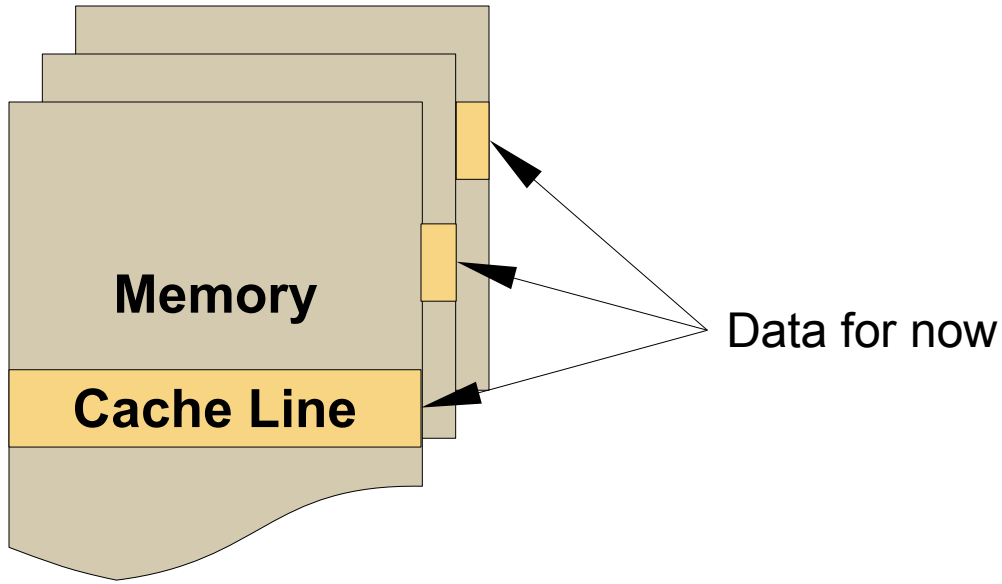
How Memory Works



- TLB maps to page in memory (KB or MB in size)
- Cache line fetched from page (often 64 bytes)

Memory Use Strategies





- Increase data density
- Increase memory level parallelism

Algorithm Cost

- Cost =

  O(operations) +

  O(cache misses) * cost/miss

- 1 cache miss ~= 100’s operations
- Worth adding operations to reduce cache misses

Appropriate data structures

Application Profile (Machine View)

Excl. User CPU (sec.)	Name
5146.620	<Total>
2031.491	spec.jbb.DeliveryTransaction.preprocess()
681.271	spec.jbb.CustomerReportTransaction.process
290.063	spec.jbb.Order.processLines(spec.jbb.Wareh
209.927	spec.jbb.Orderline.process(spec.jbb.Item,
139.638	spec.jbb.infra.Util.TransactionLogBuffer.p
130.621	spec.jbb.infra.Util.XMLTransactionLog.clea
126.969	spec.jbb.infra.Util.XMLTransactionLog.putL
126.318	spec.jbb.NewOrderTransaction.processTransa
103.452	ParallelTaskTerminator::offer_termination(
91.454	spec.jbb.StockLevelTransaction.process()

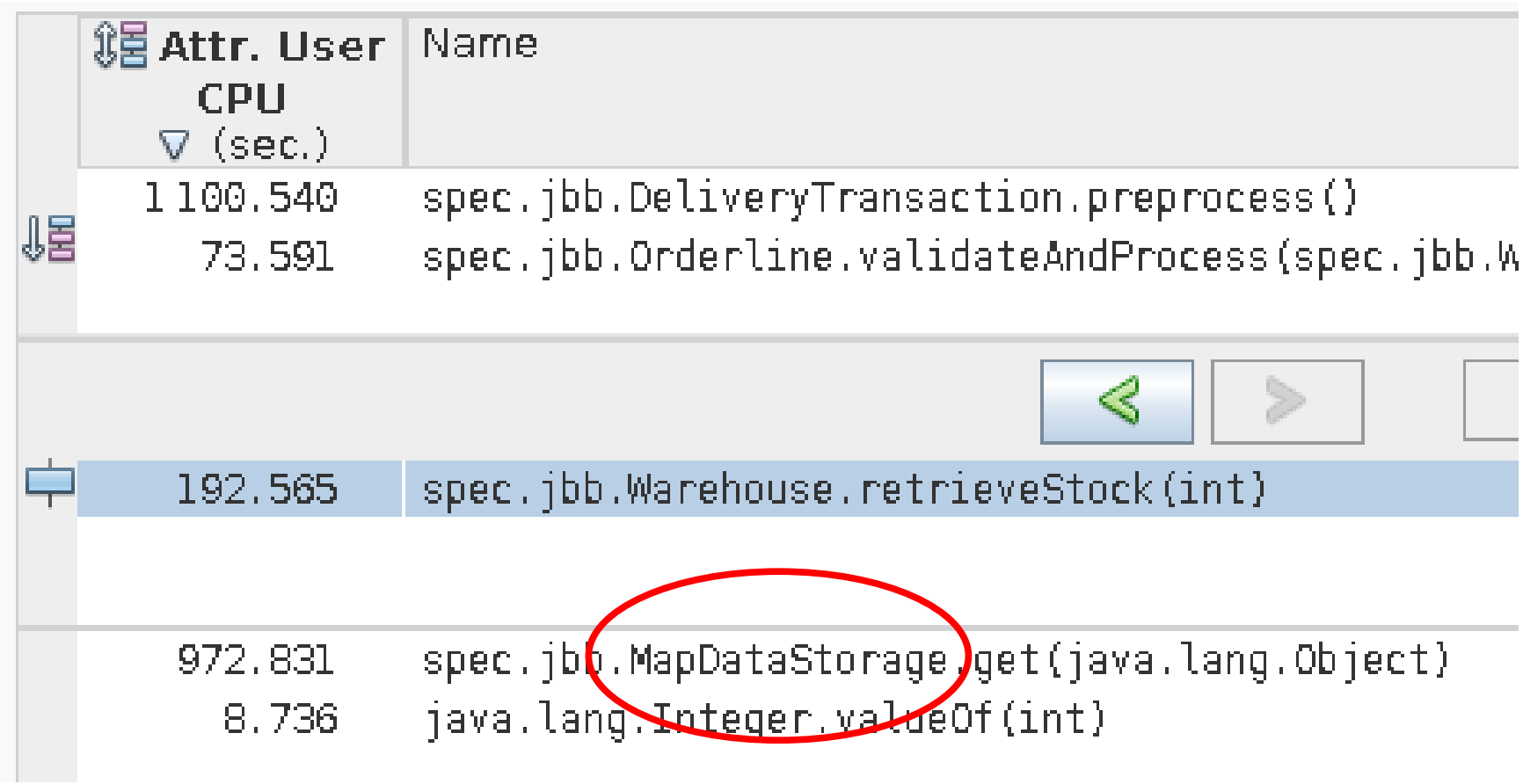
- 40% of total time

Source View of Hot Code

Incl. User CPU (sec.)	Source File: spec/jbb/DeliveryTransaction.java (not found) Object File: JAVA COMPILED METHODS (not found) Load Object: JAVA COMPILED METHODS (not found)
53.537	139. int requiredQuantity = orderline.getQuantity();
1.331	140. int itemId = orderline.getItemId();
1106.014	141. Stock stock = warehousePtr.retrieveStock(itemId);
35.035	142. int availableQuantity = stock.getQuantity();
7.936	143. if (availableQuantity >= requiredQuantity) {
1.271	144. stock.changeQuantity(-requiredQuantity);
0.	145. break;
	146. }
	147 }

- 20% of total time on code from one line

Routines Called by Hot Line



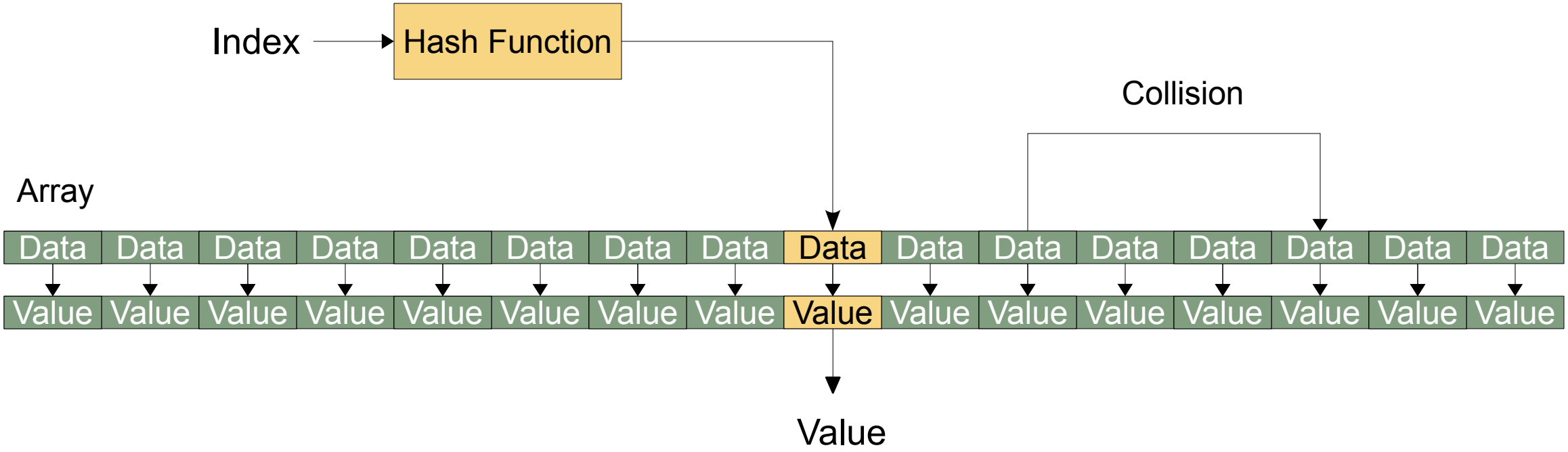
- Calls into HashTable to fetch result

Application Profile (User/Java View)

Excl. Total CPU (sec.)	Name
5,355.146	<Total>
941.699	java.util.HashMap.getNode(int, java.la
388.452	java.util.TreeMap.successor(java.util
328.560	java.util.TreeMap$Values.<init>(java.u
260.802	spec.jbb.CustomerReportTransaction.pro
230.972	java.util.TreeMap$PrivateEntryIterator
220.925	java.lang.Integer.equals(java.lang.Obj
208.576	spec.jbb.Warehouse.retrieveStock(int)

- 20% of total time

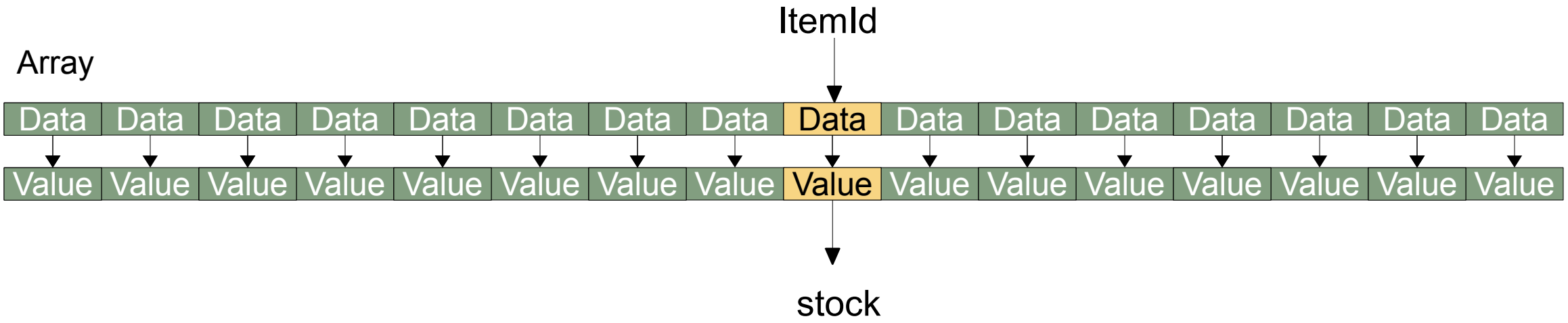
HashTable



- Look up = O(1)
- Cache misses = O(1)

ItemID is an Integer

```
    1.331    140.                    int itemId = orderline.getItemId();
1 106.014    141.                    Stock stock = warehousePtr.retrieveStock(itemId);
```



- Look up = O(1)
- Cache misses = O(1)

Application Profile with array

Excl. User CPU (sec.)	Name
5141.226	<Total>
2029.550	spec.jbb.DeliveryTransaction.preprocess()
600.480	spec.jbb.CustomerReportTransaction.process()
289.673	spec.jbb
209.677	spec.jbb
139.448	spec.jbb
130.451	spec.jbb
126.879	spec.jbb
126.098	spec.jbb
103.452	Parallel

Excl. User CPU (sec.)	Name
5145.980	<Total>
1694.395	spec.jbb.DeliveryTransaction.preprocess()
680.676	spec.jbb.CustomerReportTransaction.process()
226.108	spec.jbb.Orderline.process(spec.jbb.Item, spec

- ~300s less time, throughput increased by ~9%

Efficient implementations

Application Profile

Excl. User CPU (sec.)	Name
5141.226	<Total>
2029.550	spec.jbb.DeliveryTransaction.preprocess()
600.480	spec.jbb.CustomerReportTransaction.process()
289.673	spec.jbb.Order.processLines(spec.jbb.Warehouse, short, bool
209.677	spec.jbb.Orderline.process(spec.jbb.Item, spec.jbb.Stock)
139.448	spec.jbb.infra.Util.TransactionLogBuffer.putText(java.lang.
130.451	spec.jbb.infra.Util.XMLTransactionLog.clear()
126.879	spec.jbb.infra.Util.XMLTransactionLog.putLine(java.lang.Str
126.098	spec.jbb.NewOrderTransaction.processTransactionLog()
103.452	ParallelTaskTerminator::offer_termination(TerminatorTermina

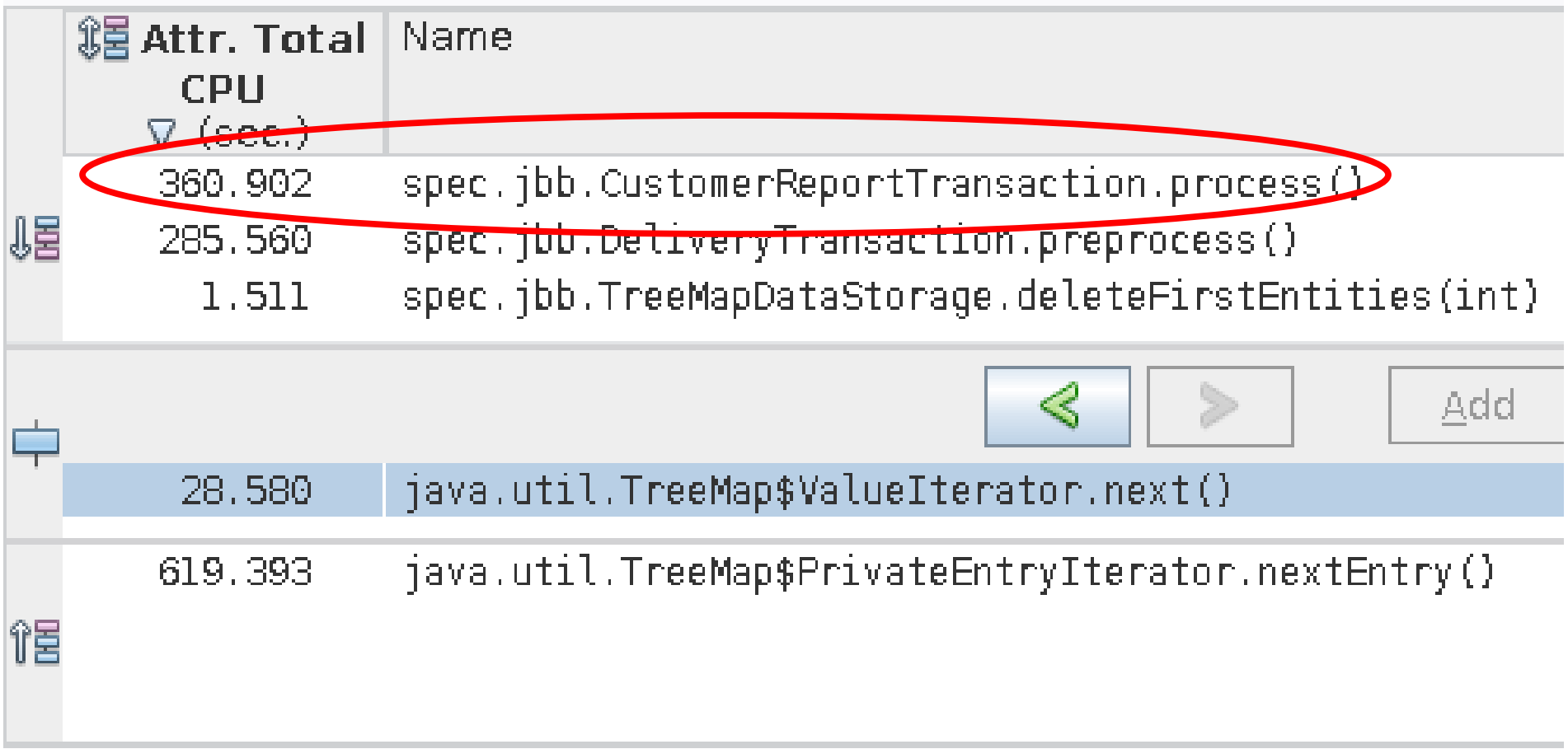
- ~12% of time

Hot Source Code

Incl. User CPU (sec.)	Source File: spec/jbb/CustomerReportTransaction.java (not found Object File: JAVA COMPILED METHODS (not found) Load Object: JAVA COMPILED METHODS (not found)
0.	235. int i = 0;
8.406	236. while (historyIter.hasNext()) {
438.337	237. history = (History) historyIter.next();
16.752	238. if (history.getCustomerId() == customerP1
1.241	239. histCount++;
0.560	240. payments[histCount] = history;
	241. }
	242. }

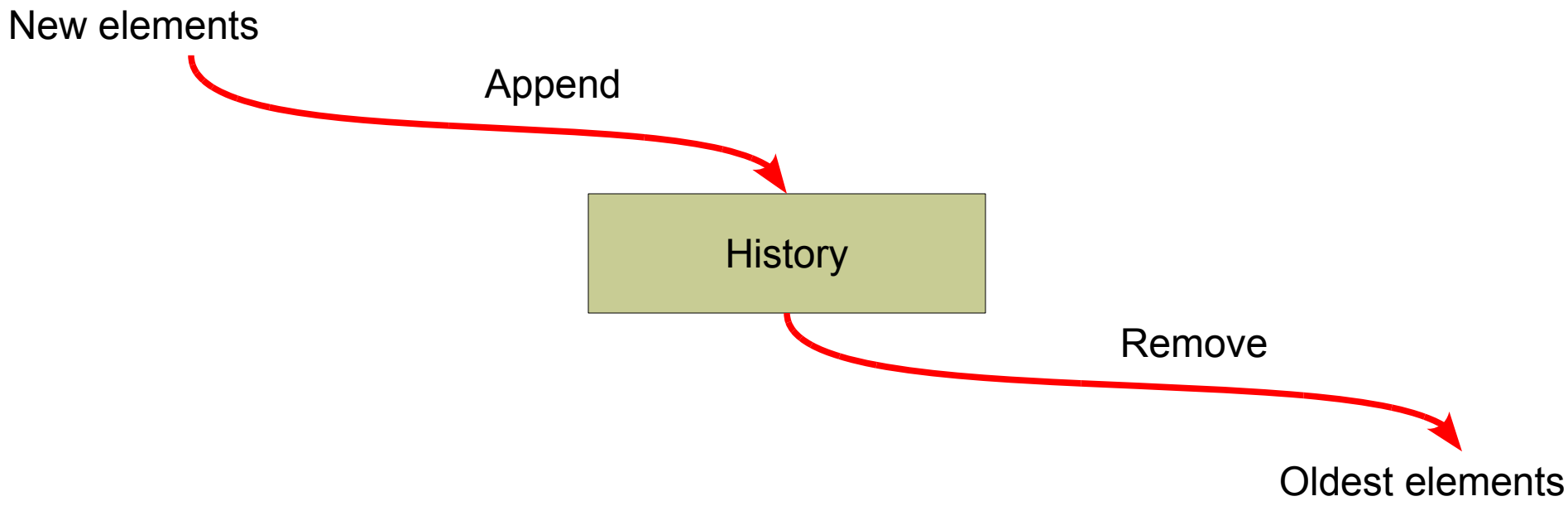
- Time spent iterating a HashMap

Hot Call Tree



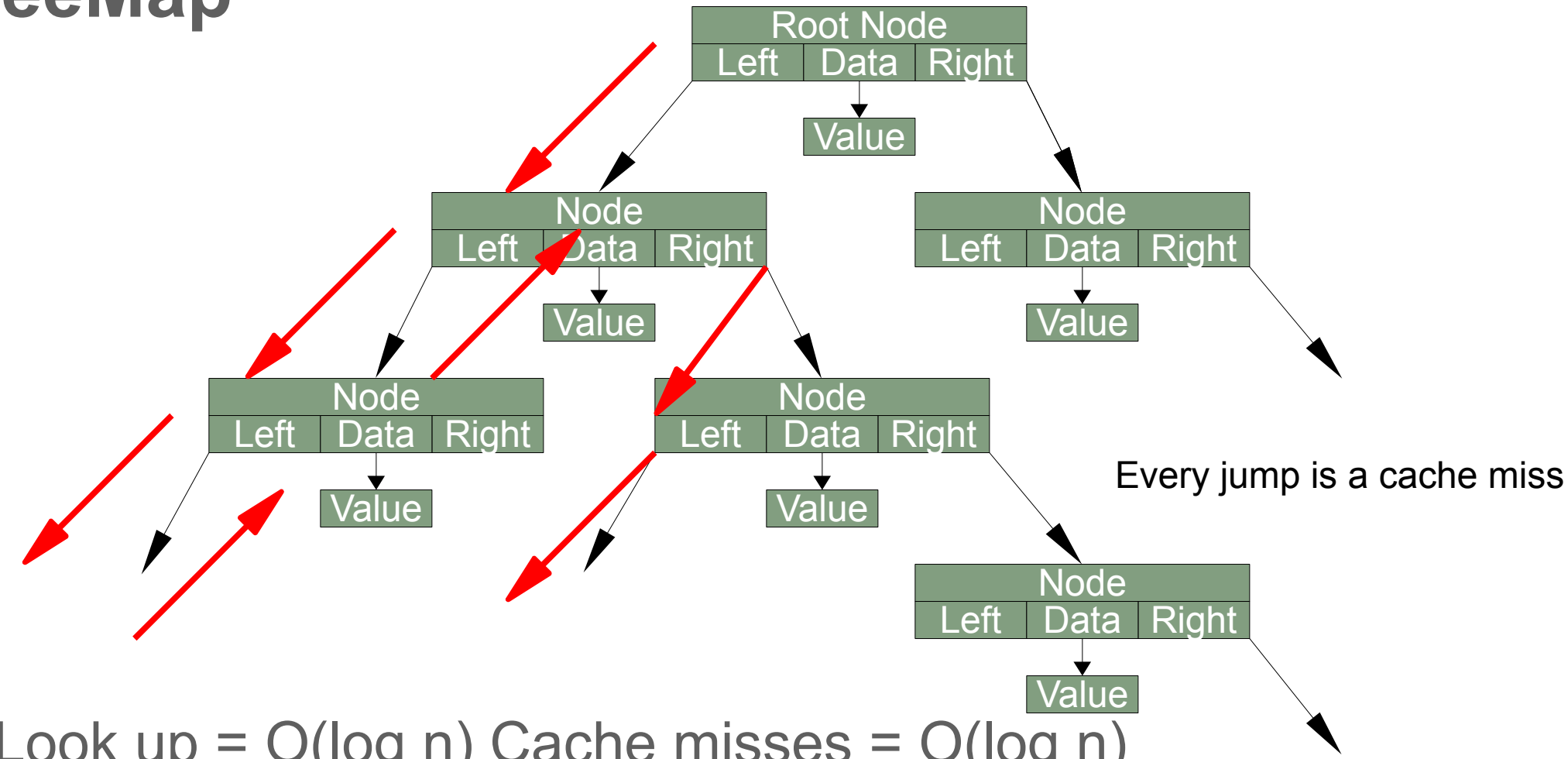
- Time spent iterating a HashMap

History



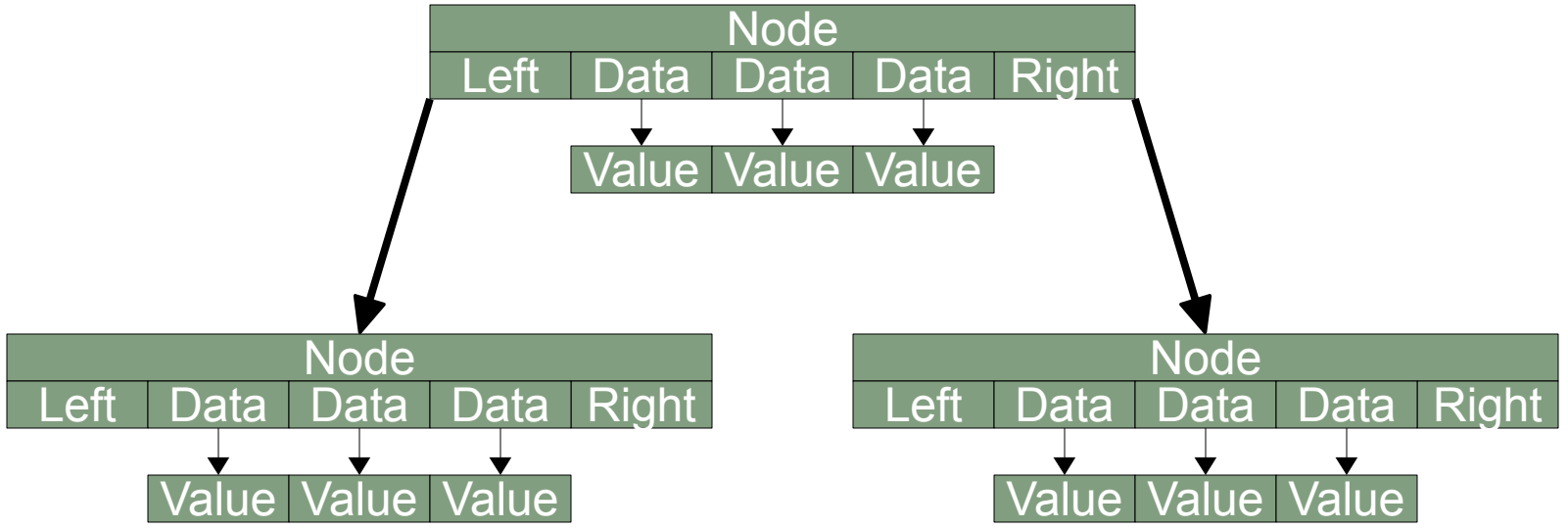
- History is constantly changing

TreeMap



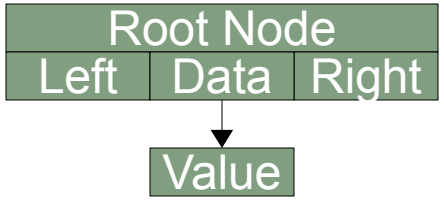
- Look up = O(log n) Cache misses = O(log n)
- Iteration = O(n) Cache misses = O(n)

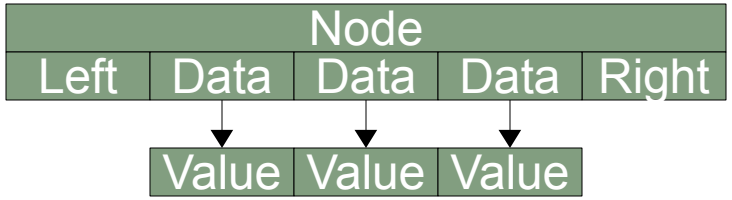
Using “Wide” Nodes

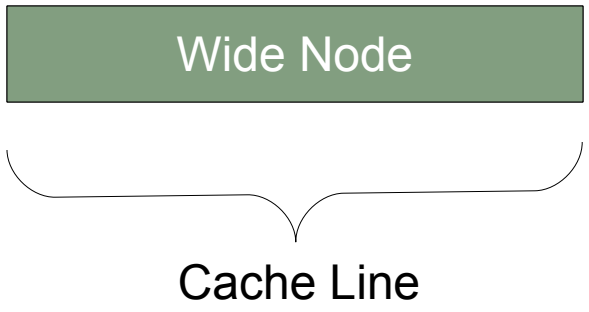


- Look up = O(log n) Cache misses = O(log n)
- Iteration = O(n), cache misses reduced by Wx (eg W=64)

Cache Line Utilisation







Application Profile with Wide Nodes

Excl. User CPU (sec.)	Name
5141.226	<Total>
2029.550	spec.jbb.DeliveryTransaction.preprocess()
600.480	spec.jbb.CustomerReportTransaction.process()
289.673	spec.jbb.O
209.677	spec.jbb.O
139.448	spec.jbb.i
130.451	spec.jbb.i
126.879	spec.jbb.i
126.098	spec.jbb.N
103.452	Parallel[Ta

Excl. User CPU (sec.)	Name
5159.009	<Total>
2117.191	spec.jbb.DeliveryTransaction.preprocess()
304.033	spec.jbb.Order.processLines(spec.jbb.Warehouse, short
277.184	spec.jbb.CustomerReportTransaction.process()

- 300s less time -> ~7% gain

Concluding Remarks

Hardware Tuning Opportunities

- Automatic JVM use of intrinsics
- Developer's knowledge of hardware
- Cache misses critical part of performance
  - Efficient use of loaded data
  - Minimise number of hops

Q&A

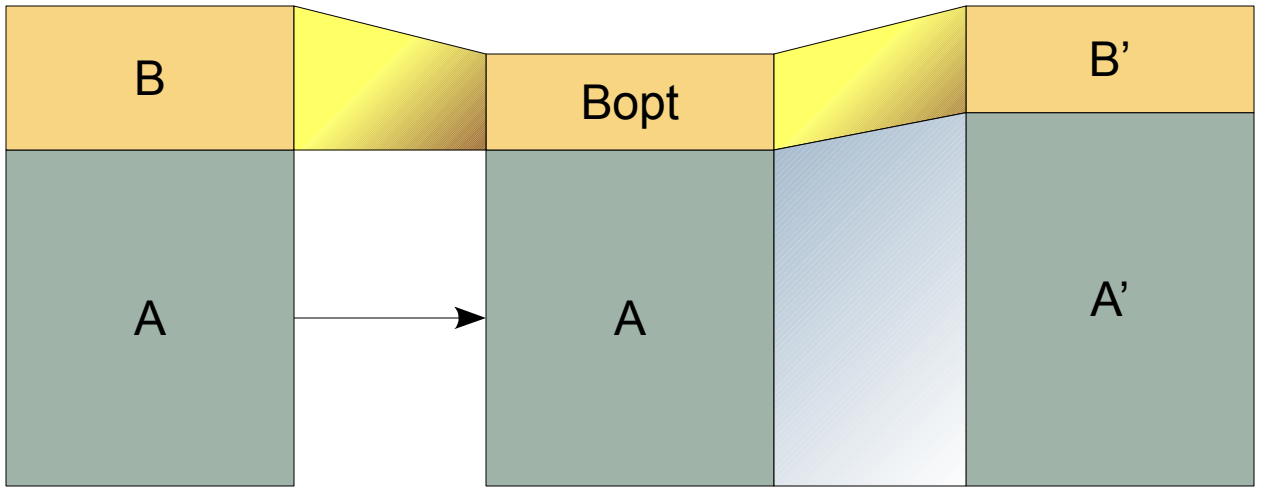
Hardware and Software
Engineered to Work Together

JavaOne™

ORACLE


ORACLE

Comparing Constant Time Profiles



Original profile

Profile of optimisation and original workload

New profile

- Unmodified code scales by S
  S = A' / A