End-to-End Performance Optimization of Java Server Workloads

Jong-Deok Choi

(jdchoi@us.ibm.com)

 © 2003 IBM Corporation

People

- Pratap Pattnaik, Manish Gupta
- Trey Cain, Jong-Deok Choi, Suhyun Kim, Kyung Ryu, Mauricio Serrano, Yefim Shuf, *Gilad Arnold, Ian Steiner, Richard Zhuang*
- Joefon Jann, Christoph von Praun, Stephen E Smith, Il Park
- Toshio Nakatani, Kazuaki Ishizaki, Tamiya Onodera

© 2003 IBM Corporation

Outline

- **End-to-End Optimization Project**
  - Workload and Server Configurations
- Methodology
- Performance Characteristics
  - Method Profiling, Cache, Branch Prediction, Synchronization
- Summary

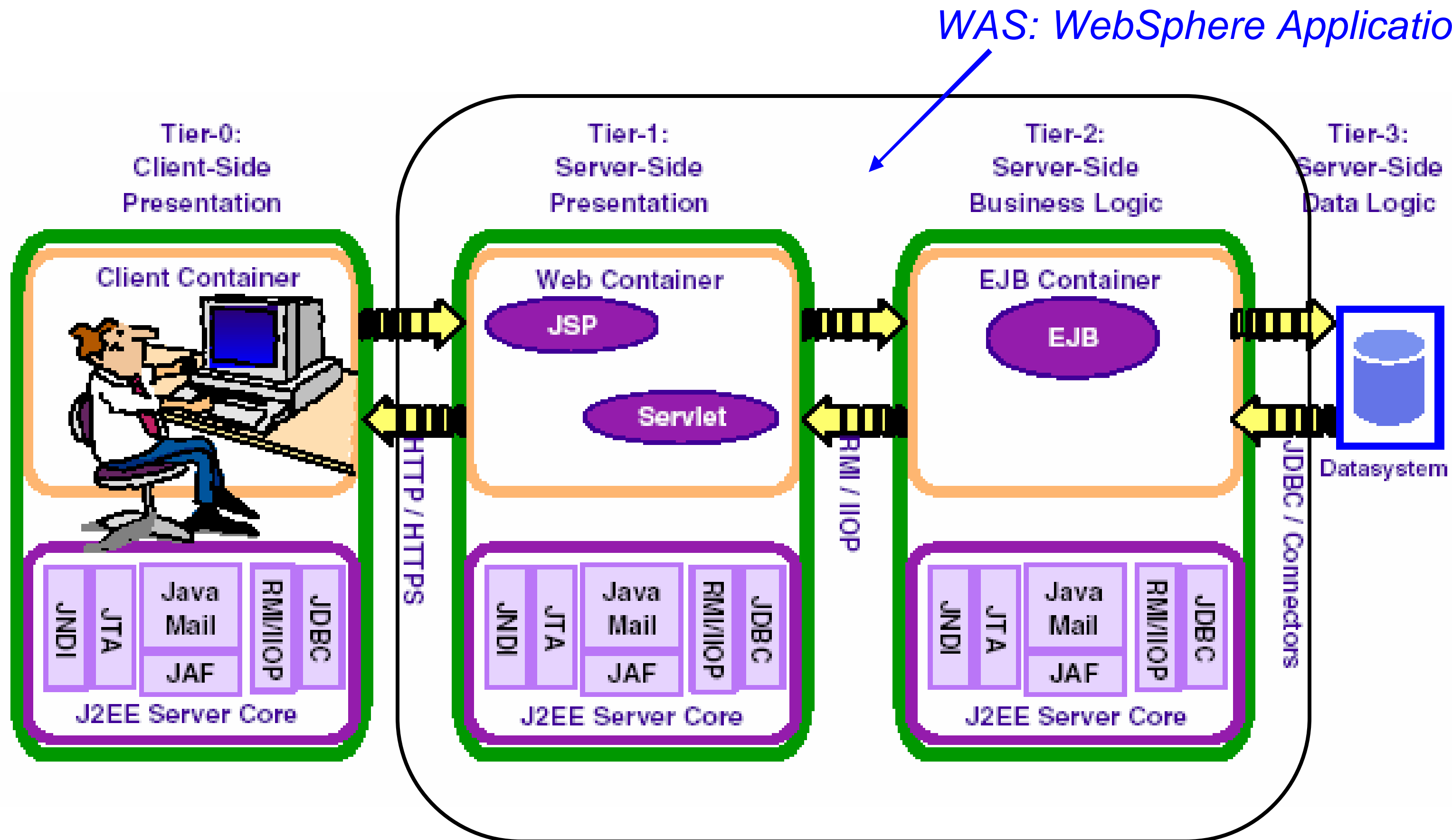
© 2003 IBM Corporation

End-to-End Optimization Project

- To understand and optimize the performance of the *whole-stack, end-to-end* SW/HW layers of *commercial middleware* applications (J2EE) on IBM's current and future *high-end servers*.

© 2003 IBM Corporation

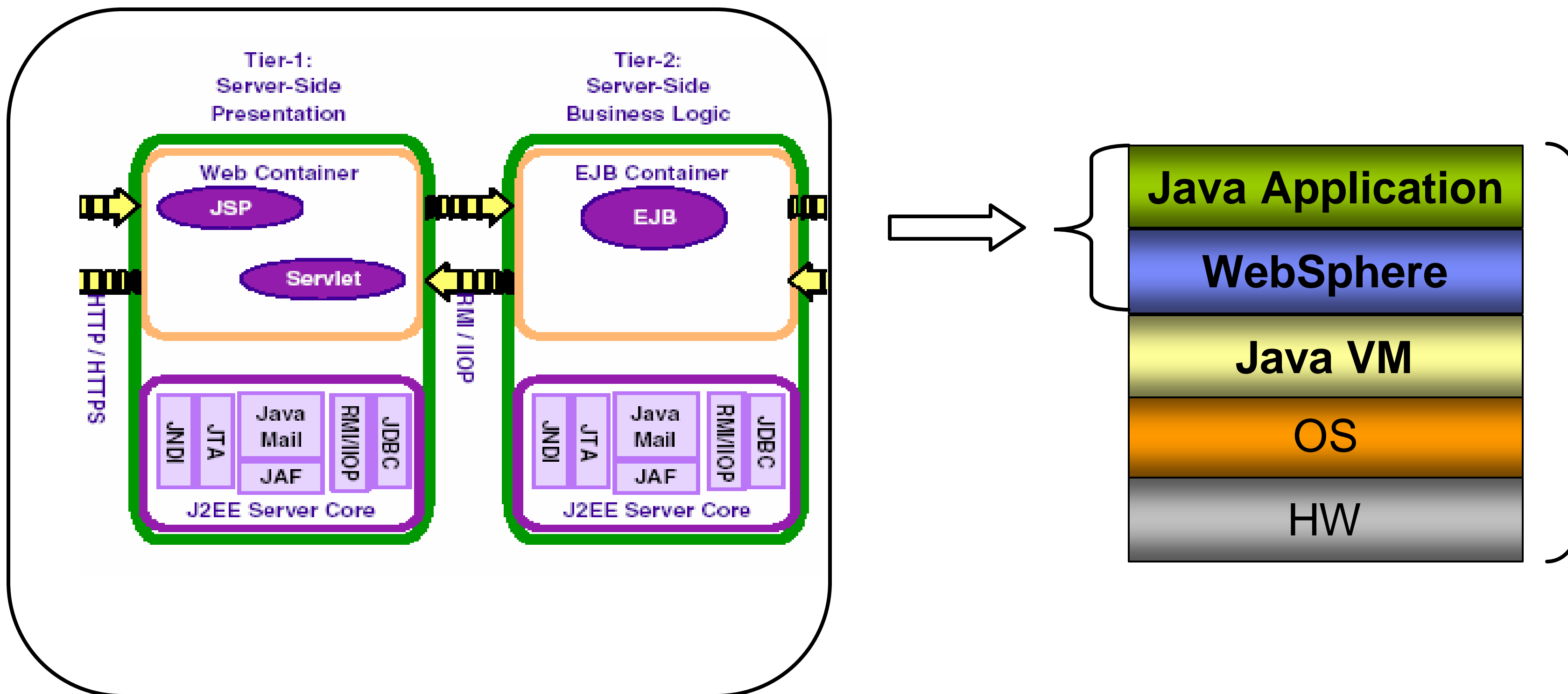
Workload: J2EE Multi-tier Server w/ Application



Source: Programming J2EE APIs with WebSphere Advanced by Osamu Takagiwa, et. al. Ibm.com/redbooks

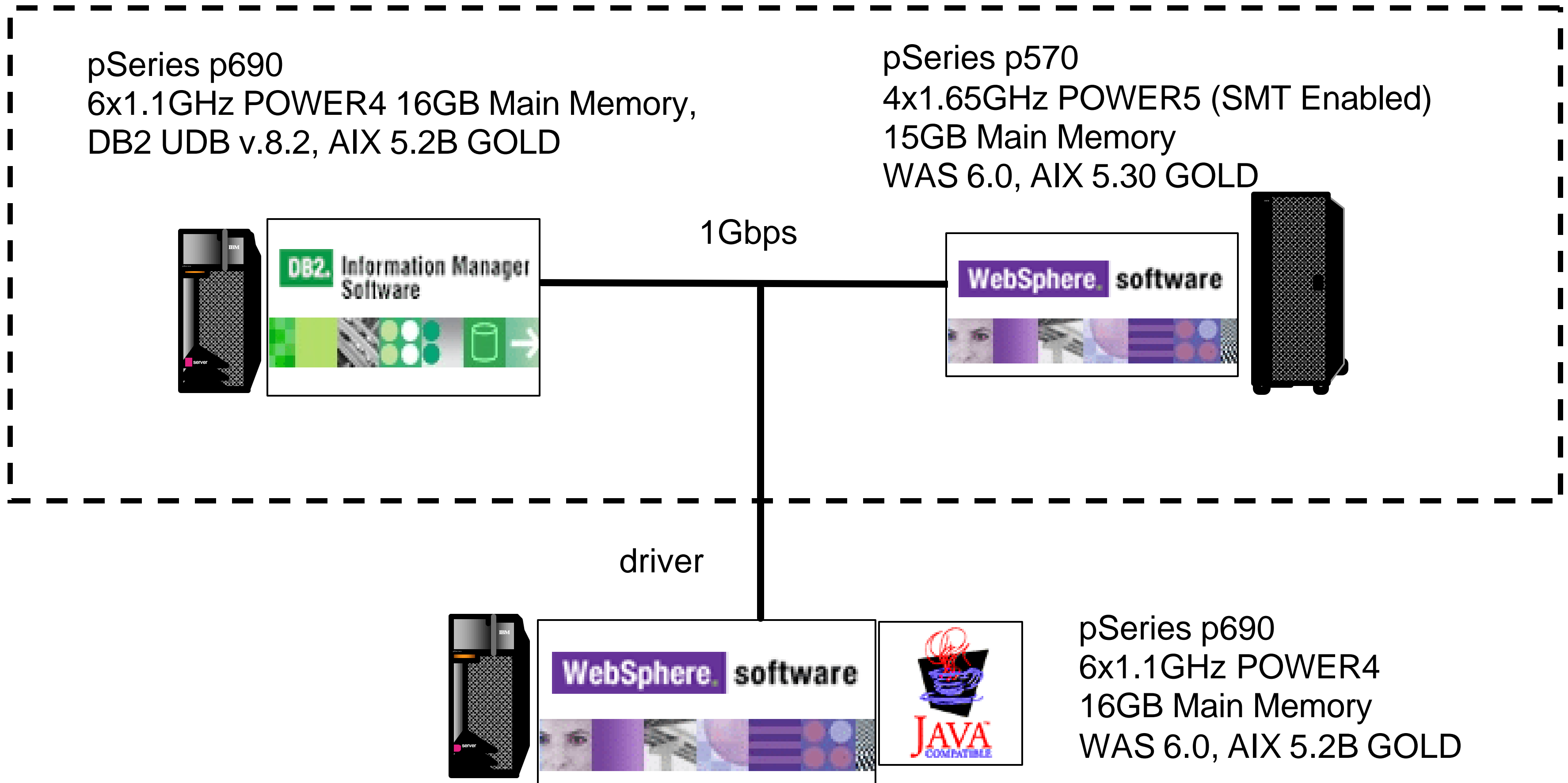
© 2003 IBM Corporation

J2EE Whole-Stack End-to-End Optimization



© 2003 IBM Corporation

Server Configuration for SpecJAS2004/Trade6



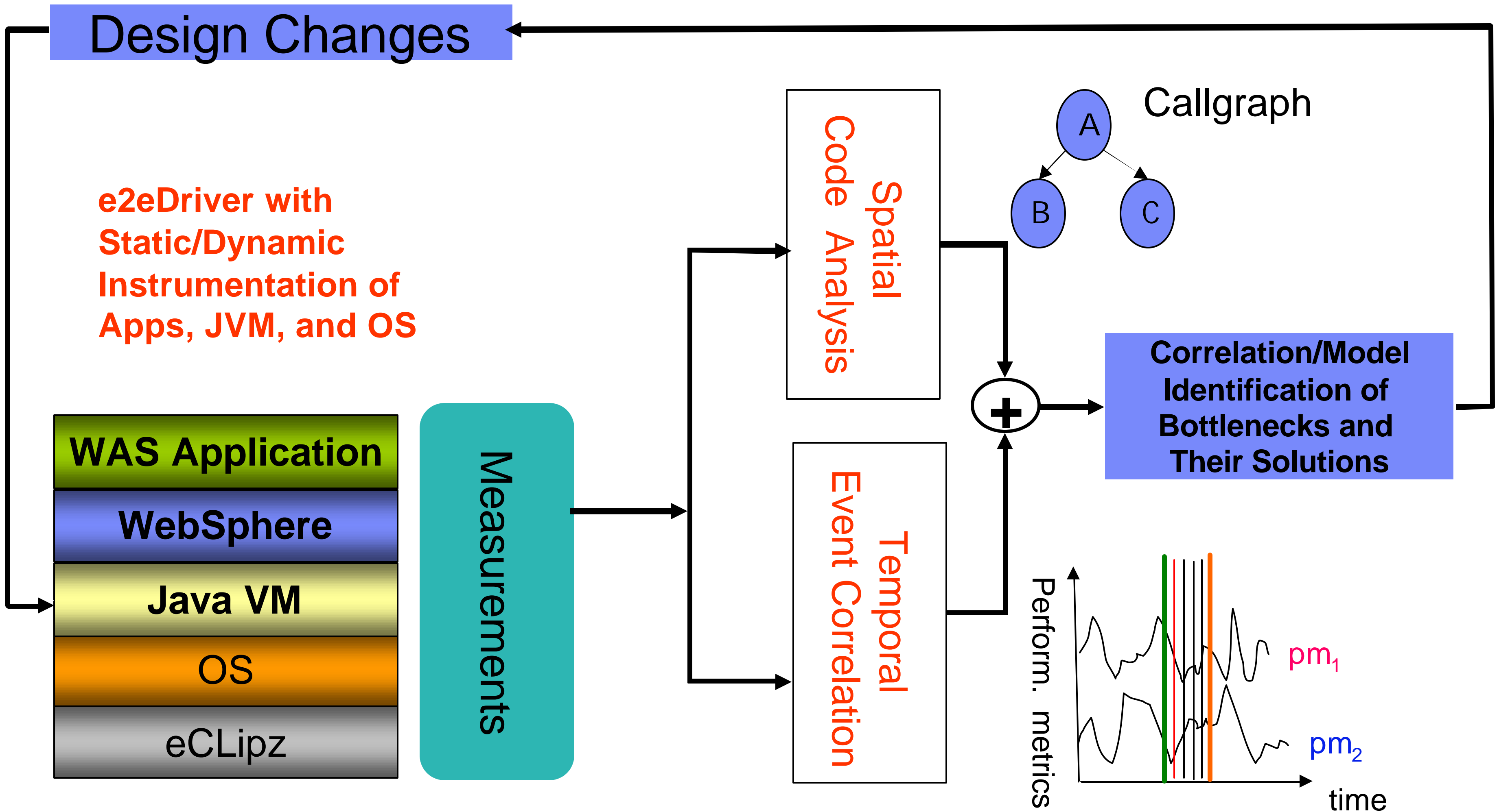
© 2003 IBM Corporation

Outline

- End-to-End Optimization Project
  - Workload and Server Configurations
- **Methodology**
- Performance Characteristics
- Summary

 © 2003 IBM Corporation

End-to-End Optimization Methodology



© 2003 IBM Corporation

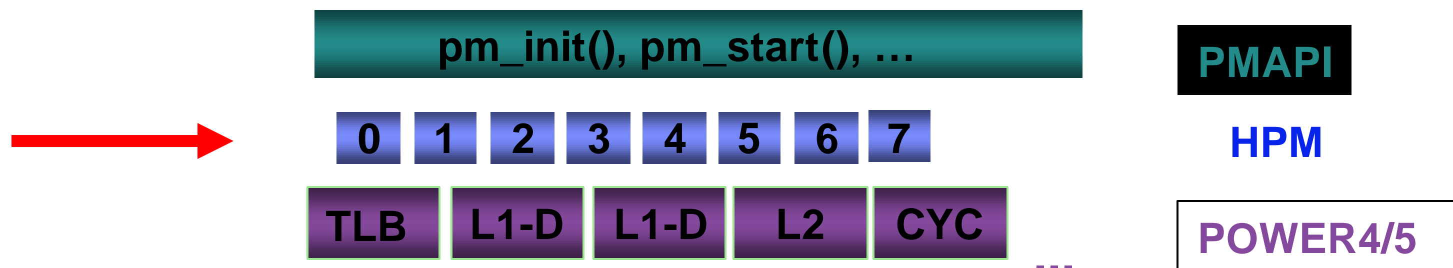
Performance Metrics and Tools

Layer	Metrics Examples	Method
WAS Application	Response time of each transaction	Application Response Management (ARM)
WebSphere	# of executing beans, # of activated beans	Performance Monitor Infrastructure (PMI) in WAS
Java VM	# of method calls, # of GC, # of object allocations, # of syncs	Java Instrumentation
OS	# of context switches	AIX trace facility, vmstat, sar
HW	# of inst., # of loads, # of D$ misses	HW Performance counters in POWER4/5

© 2003 IBM Corporation

Hardware Performance Monitor (HPM)

- **POWER4 has 8 HPM counters that can be programmed to count HW events**
  - The HW events are combined into logical *groups*
  - There are 61 groups, and 8 events per group (one event per counter)

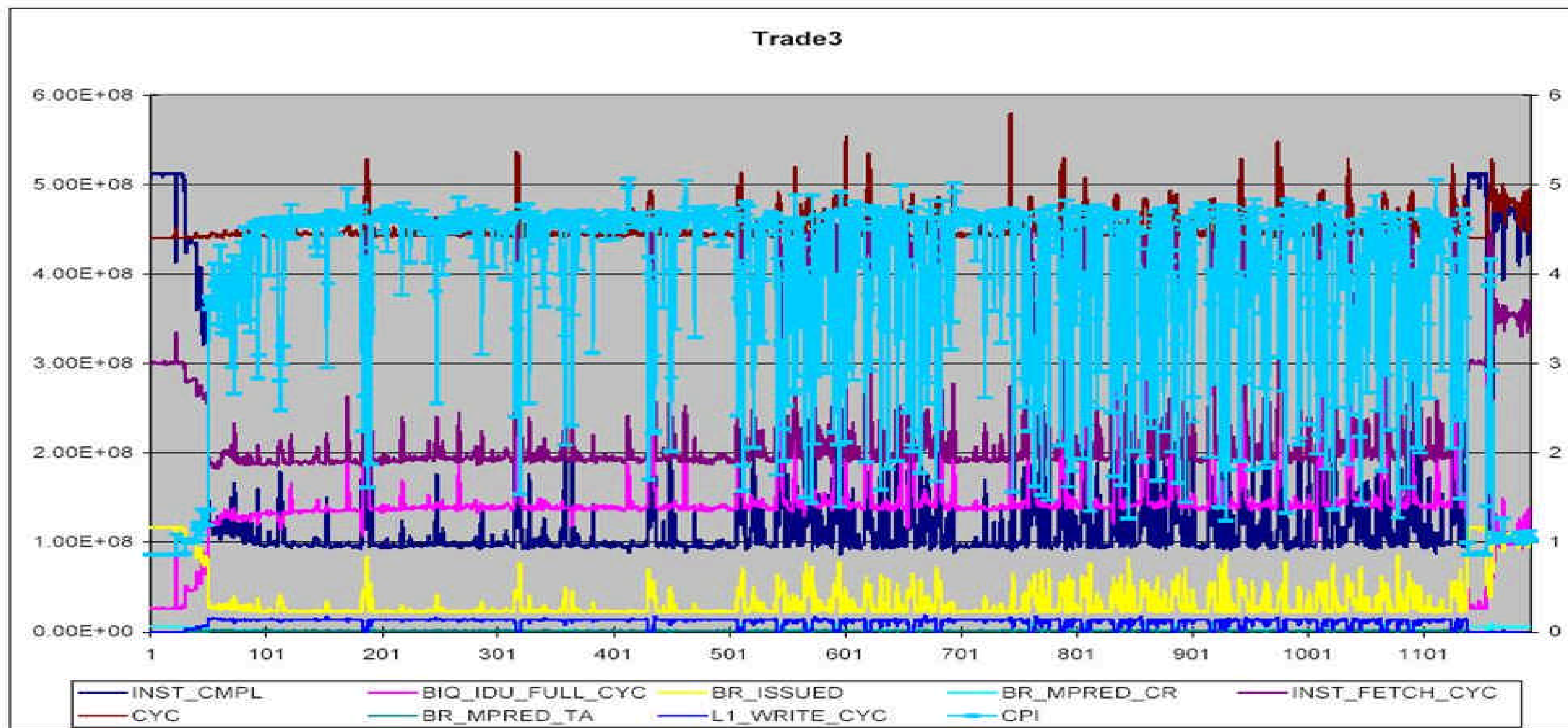


Group 56: CPI, TLB, L1-D cache

- **PM_DTLB_MISS -- Data TLB misses**
- **PM_ITLB_MISS -- Instruction TLB misses**
- **PM_LD_MISS_L1 - L1 D cache load misses**
- **PM_ST_MISS_L1 - L1 D cache store misses**
- **PM_CYC -------- Processor cycles**
- **PM_INST_CMPL -- Instructions completed**
- **PM_ST_REF_L1 -- L1 D cache store references**
- **PM_LD_REF_L1 -- L1 D cache load references**

© 2003 IBM Corporation

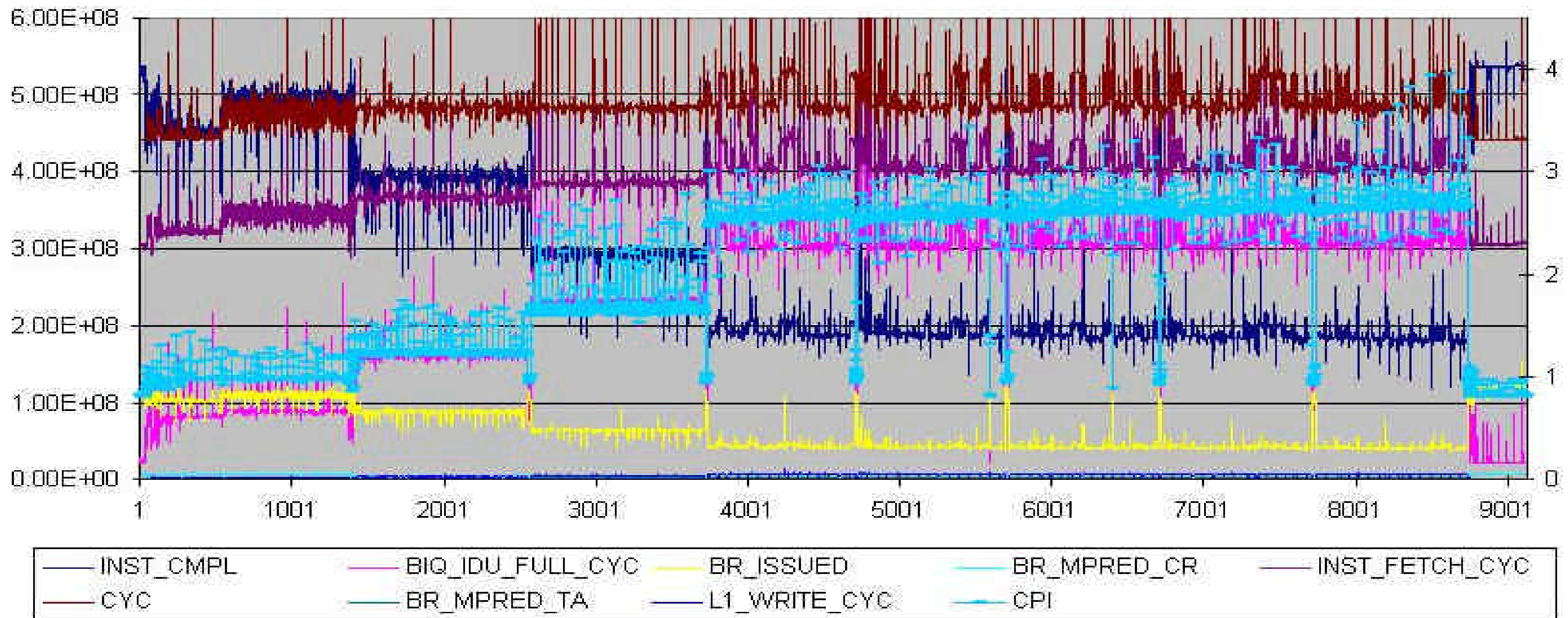
HW Performance Counters



1. **Trade3/WAS/Sovereign/AIX/Power4, System+User**
2. **hpm counters, group 3, ~100 seconds**
3. **Steady state CPI = ~4.5**

© 2003 IBM Corporation

HW Performance Counters



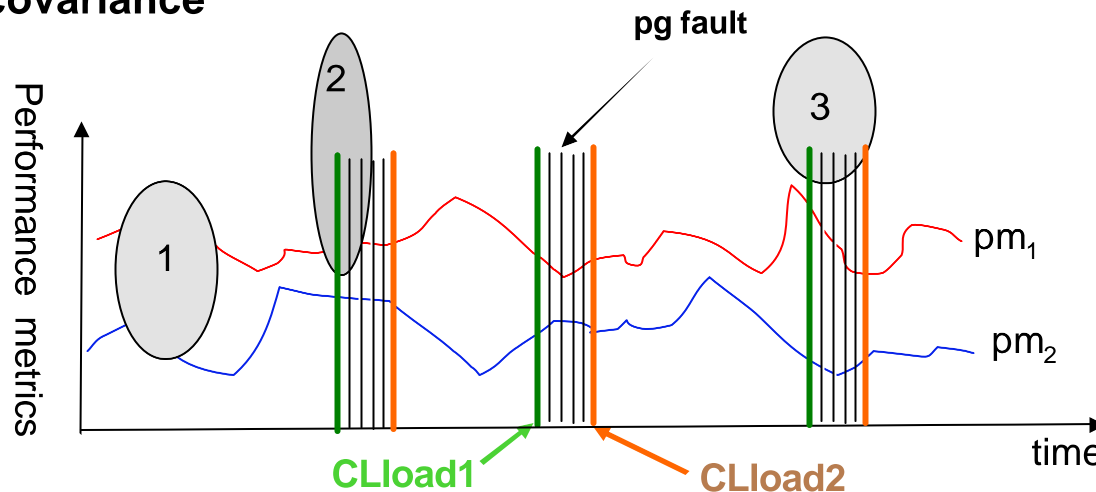
1. **SpecJBB/J9/AIX/Power4, System+User**
2. **hpm counters, group 3, ~900 seconds**
3. **Steady state CPI = ~2.5**

© 2003 IBM Corporation

Temporal Event Correlation

1. **Micro Event ⊗ Micro Event (e.g., performance metrics)**
   - CPI ⊗ TLB misses
2. **Micro Event ⊗ Macro Event**
   - TLB misses ⊗ Page fault at OS
3. **Macro Event ⊗ Macro Event**
   - Page fault at OS ⊗ Class Loading

- **Temporal event correlation employs various statistical tools such as covariance**



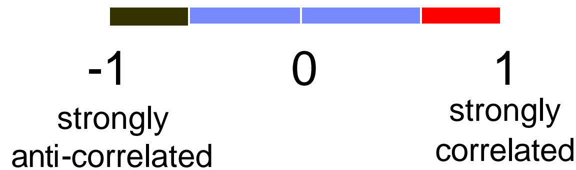
© 2003 IBM Corporation

Derived Metrics - Correlations

Given two vectors $X = \{x_i\}$ and $Y = \{y_i\}$

$$\mathrm{covar}(X,Y) = (1/n)\, \Sigma\, (x_i - \bar{x})\, (y_i - \bar{y})$$

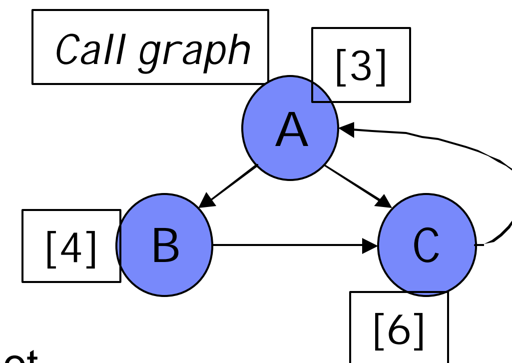
$$\mathrm{cc}(X,Y) = \mathrm{covar}(X,Y) \;/\; \mathrm{SQRT}\,[\mathrm{covar}(X,X) * \mathrm{covar}(Y,Y)]$$



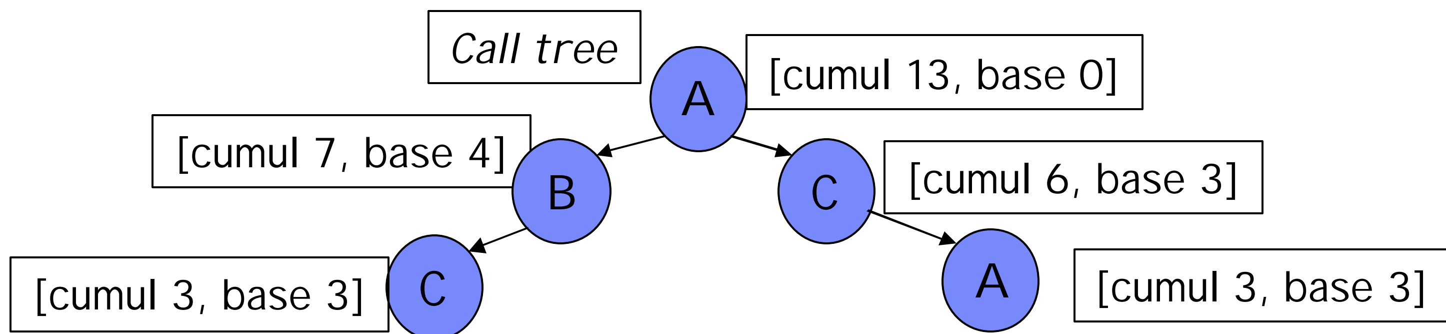
- Observe Trends: transient, steady-state, periodic
- Certain correlations are expected; spot the unexpected
- Needs systematic study

© 2003 IBM Corporation

Spatial Code Analysis



- **Profile-based hot-code analysis**
  - Context-insensitive
    - Identify *m* hot (frequently executed) methods
    - May fail to provide the contexts in which methods are hot
  - Context-sensitive
    - Identify *n* hot dynamic call chains
      - e.g. critical call-path information
    - May fail to recognize hot methods with uniform and low unit-cost



 © 2003 IBM Corporation

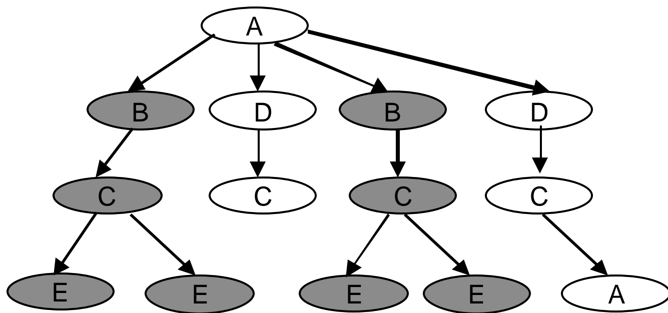
Context-Sensitive Analysis

- **SPECjAppServer2004: 50% of JIT'd code execution is in 224 "hottest" methods**
  - Method profiel is "flat"
  - Data profile is also "flat"
- **Profile-based hot-code analysis**
  - Identify *n* hot dynamic call chains
    - e.g. critical call-path information
- "Accurate, Efficient, and Adaptive Calling-Context Profiling," PLDI2006
  - X. Zhuang, M. Serrano, T. Cain, and J.-D. Choi

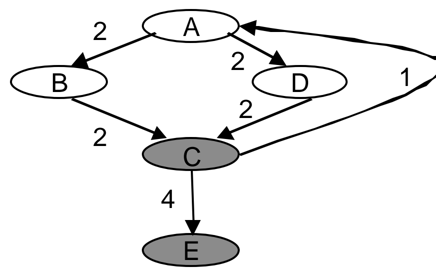
© 2003 IBM Corporation

Context-Senstive Analysis

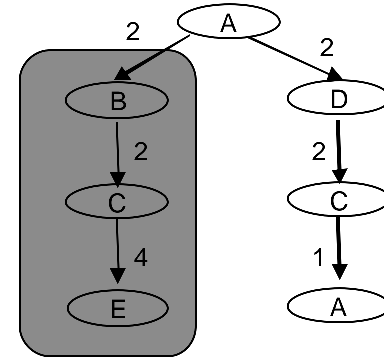
- Call sequence:
  - '->': method call, '<-': method return, '(A)' : 'A' is top-of-stack
  - A -> B -> C -> E, <-, (C) ->E, <, <-, <-, (A)-> D -> C, <-. <-, (A) -> B -> C -> E , <-, (C) ->E, <-, <-, <-, (A) -> D -> C, ->A, <-, <-, <-, (A)



Call Tree



Call Graph (edge profiling)



Calling-Context Tree (CCT)

- Call tree is too expensive: one node per each method call
- Call graph is too imprecise: cannot tell whether B or D is more responsible for the frequent calls of E by C
- CCT is not as expensive as call tree; on CCT it's clear B->C->E is the expensive call path.

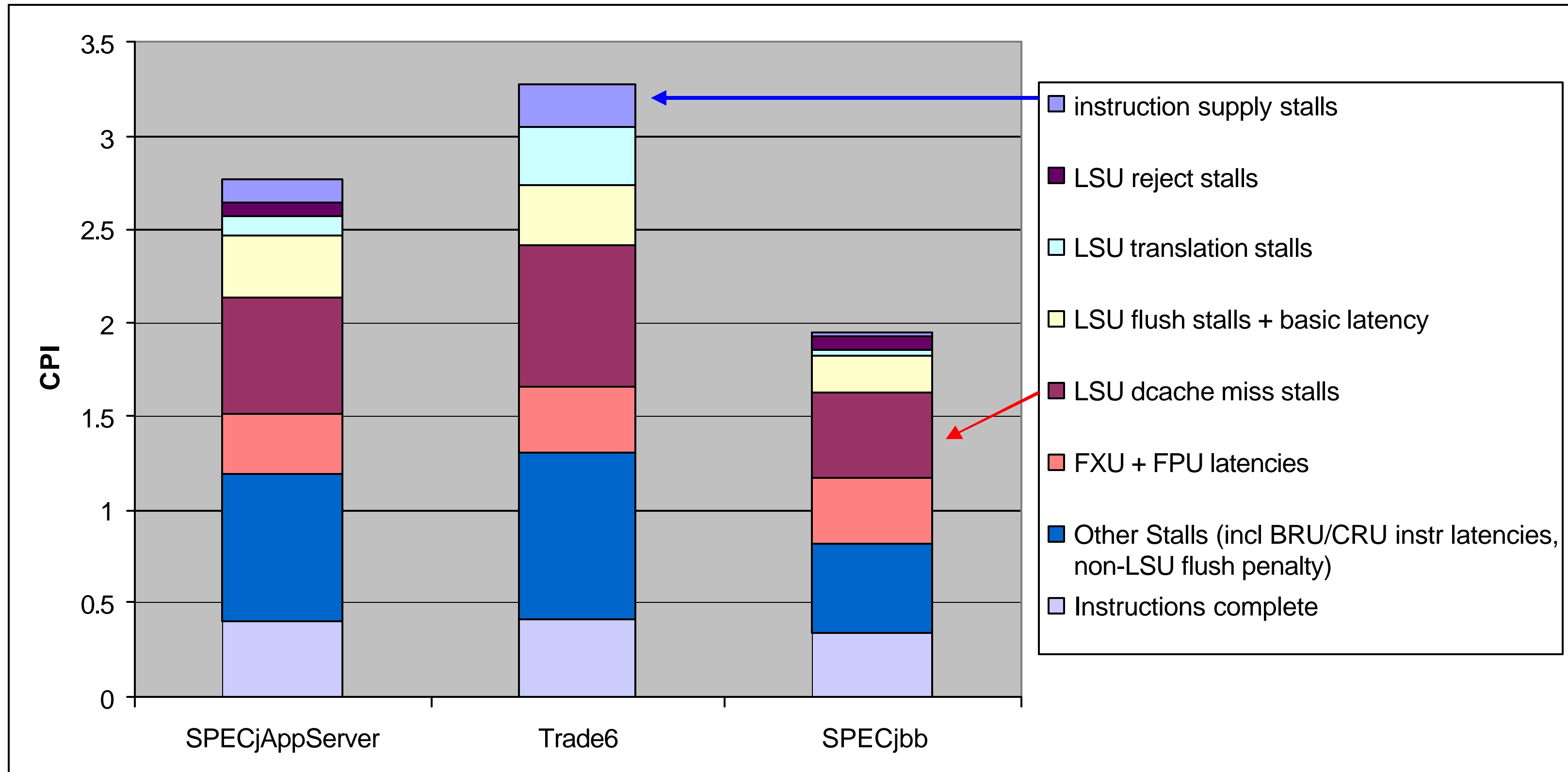
© 2003 IBM Corporation

Outline

- End-to-End Optimization Project
  - Workload and Server Configurations
- Methodology
- **Performance Characteristics**
- Summary

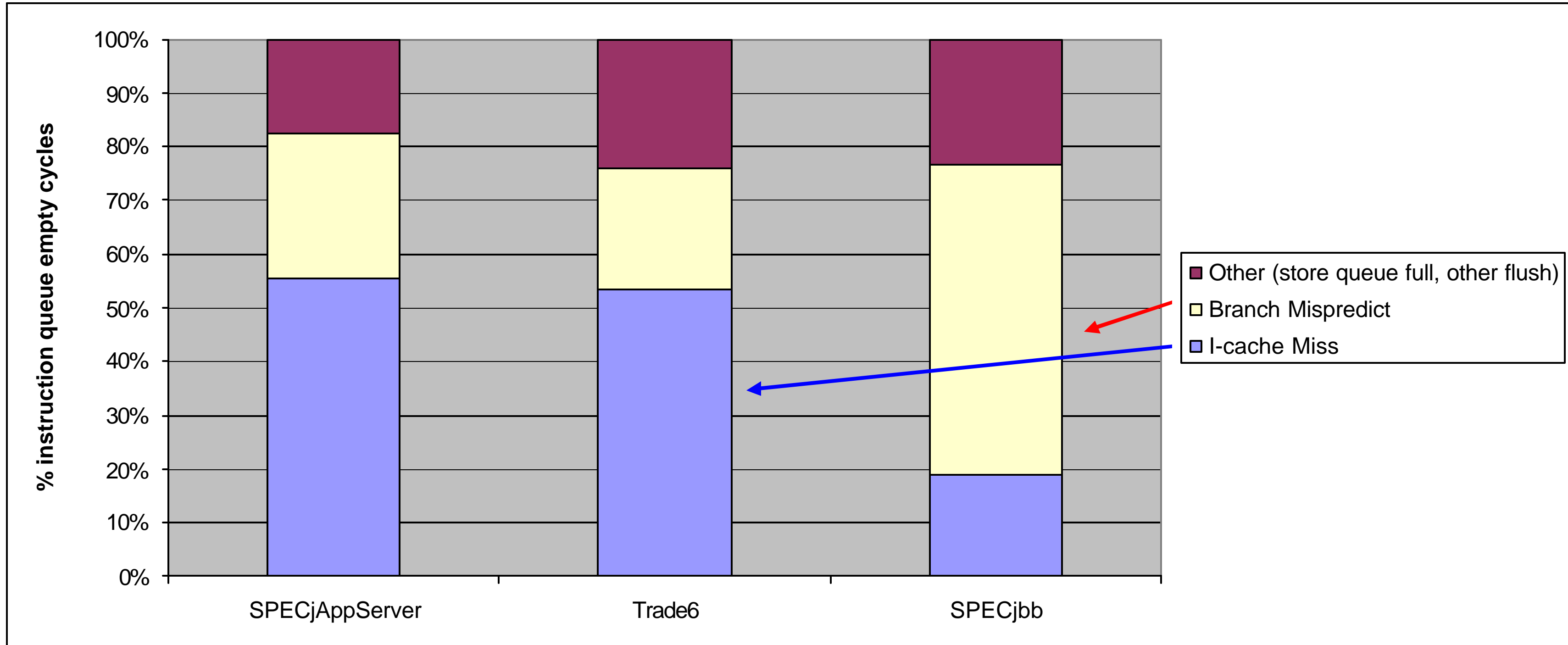
 © 2003 IBM Corporation

CPI Stacks



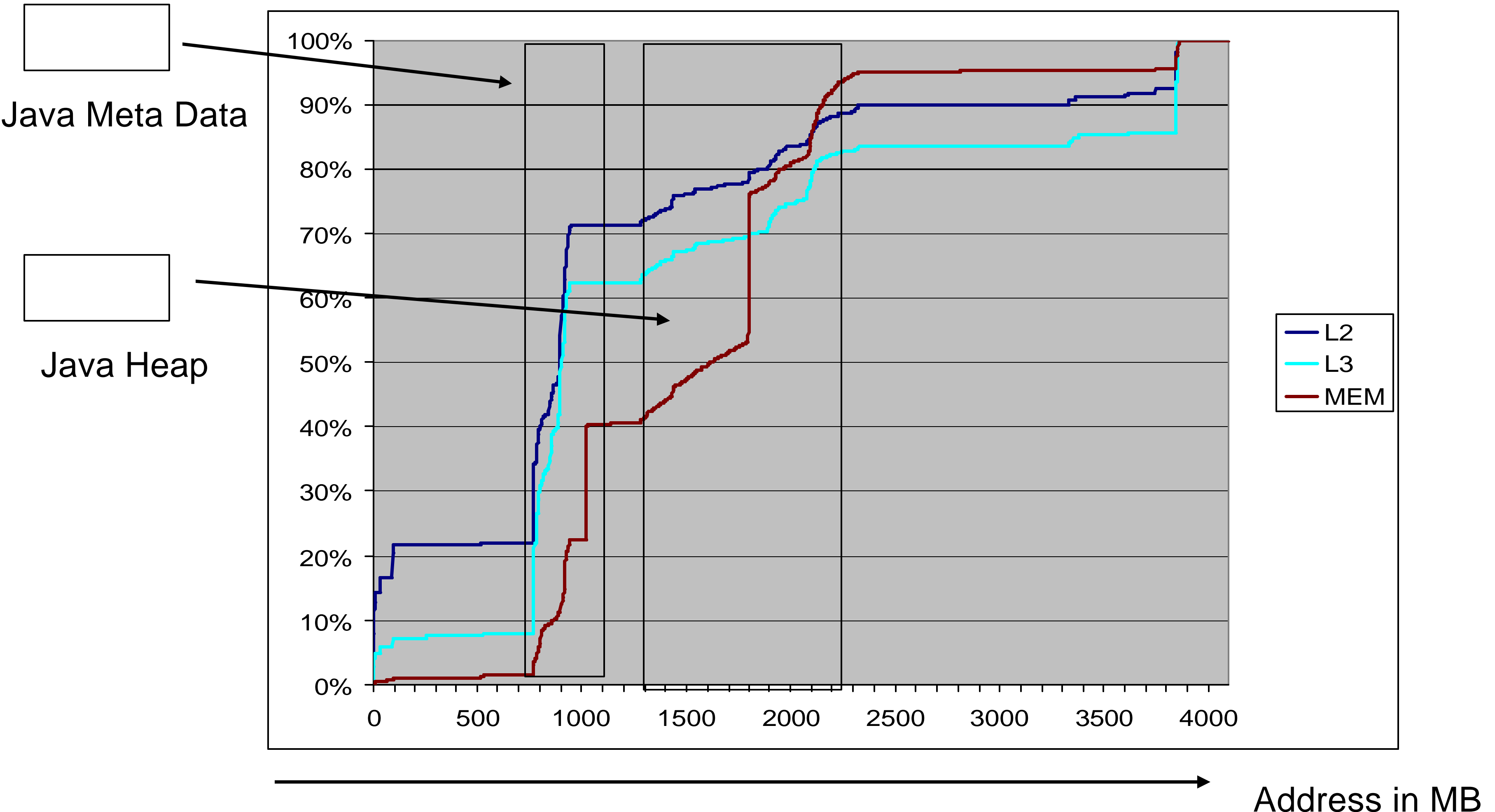
 © 2003 IBM Corporation

Instruction supply stall breakdown



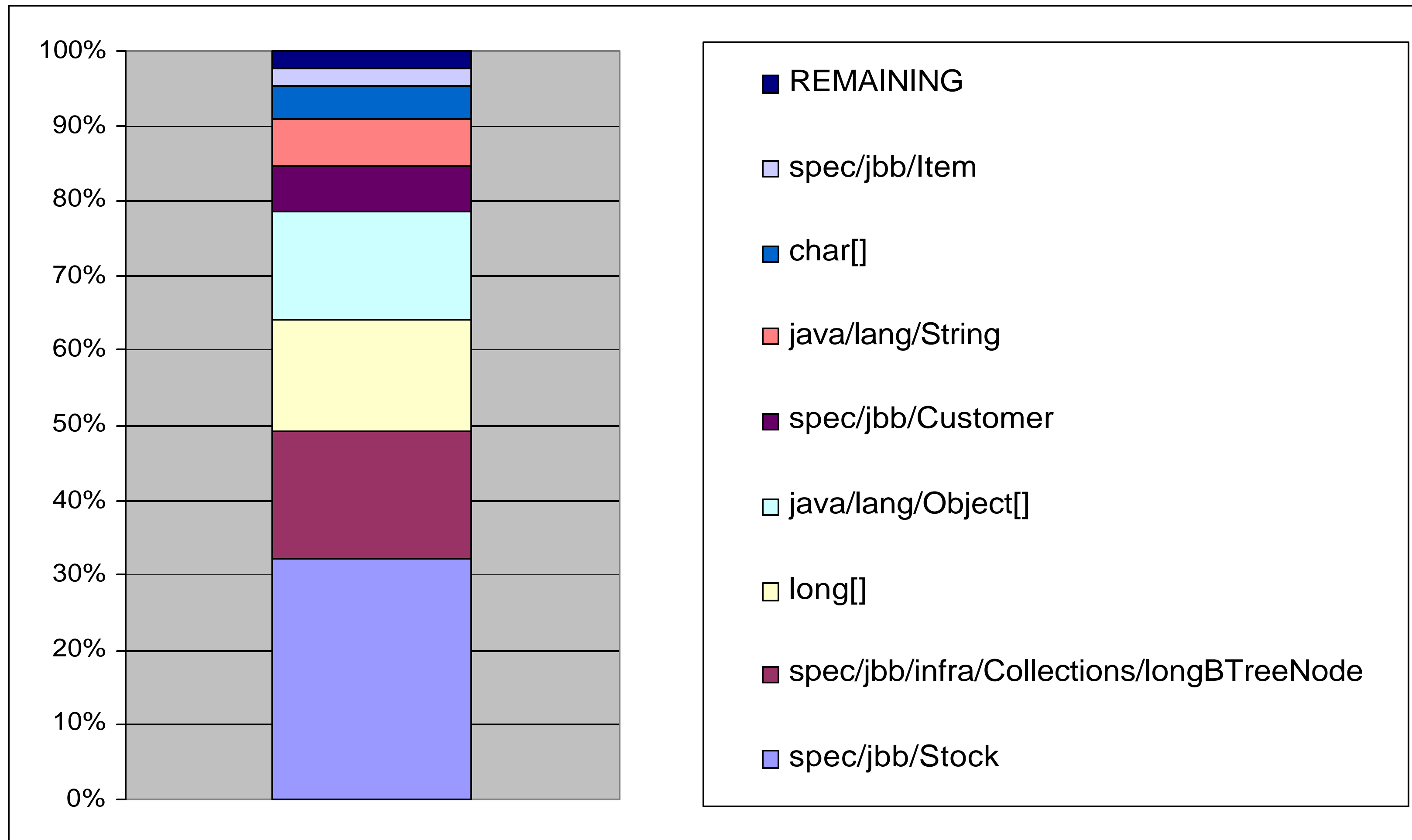
© 2003 IBM Corporation

L1 Miss Data Load Patterns: JAS2004



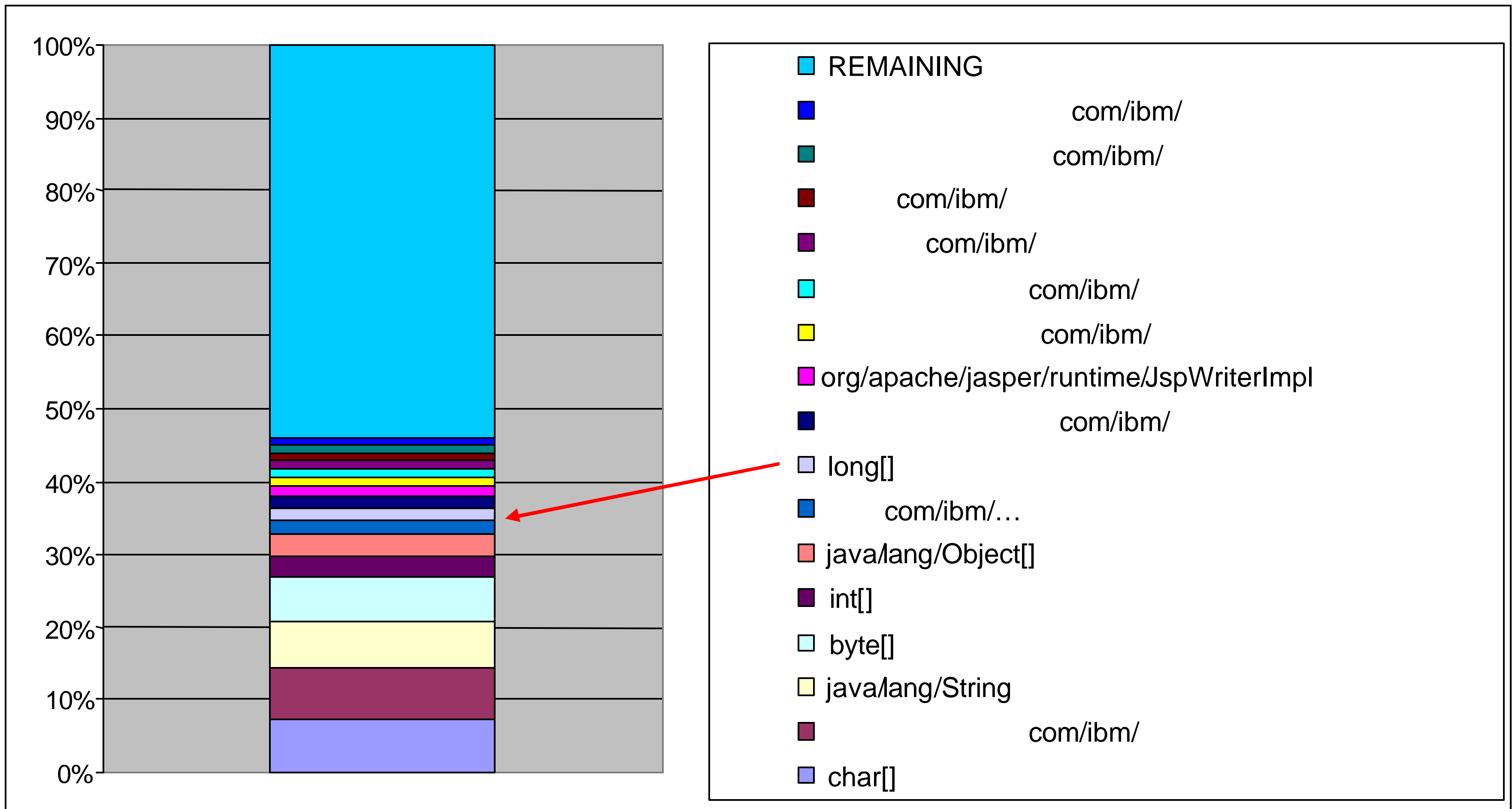
© 2003 IBM Corporation

Types of Java Heap misses – SPECjbb2000



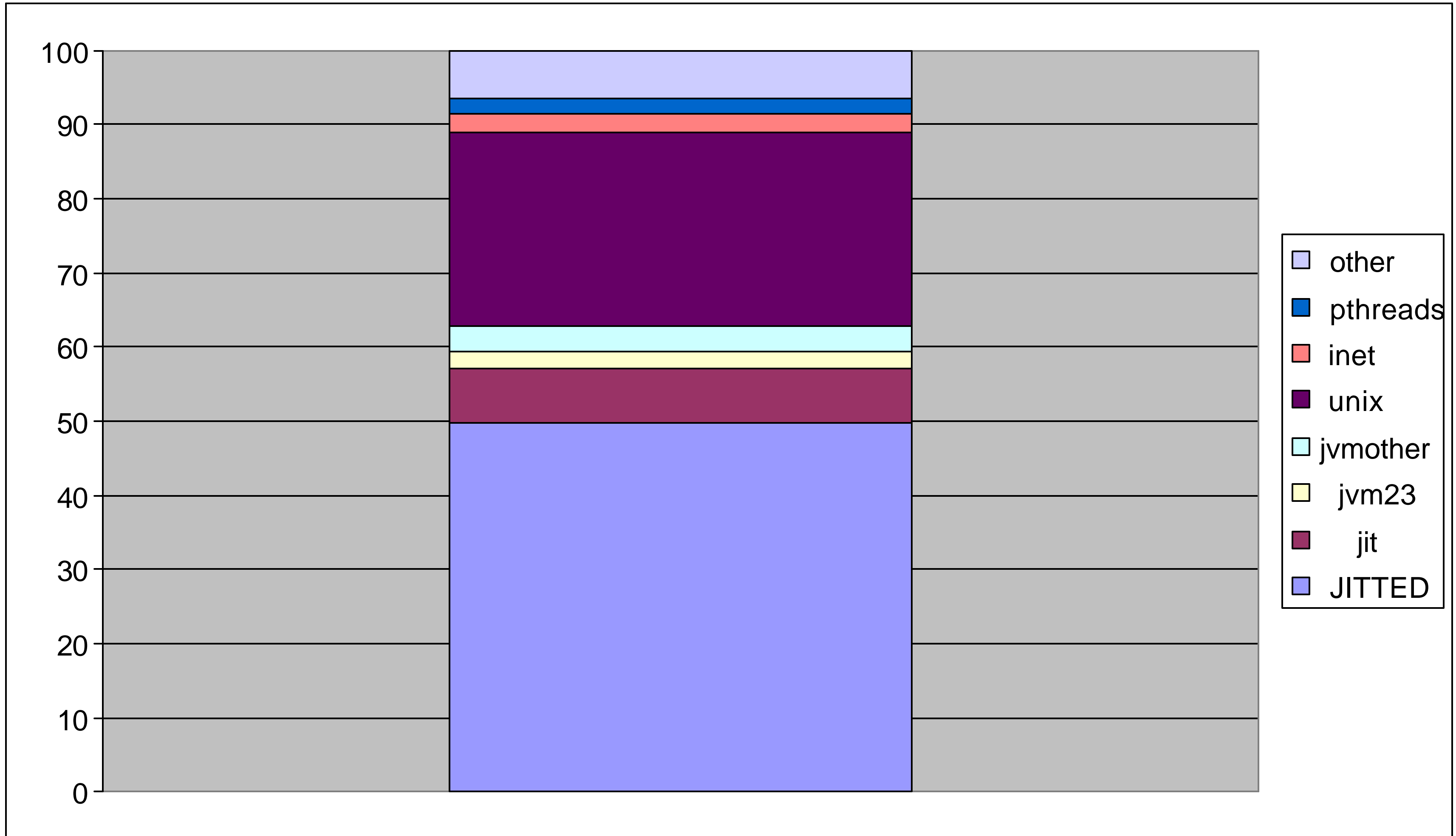
© 2003 IBM Corporation

Types of Java Heap misses – JAS 2004



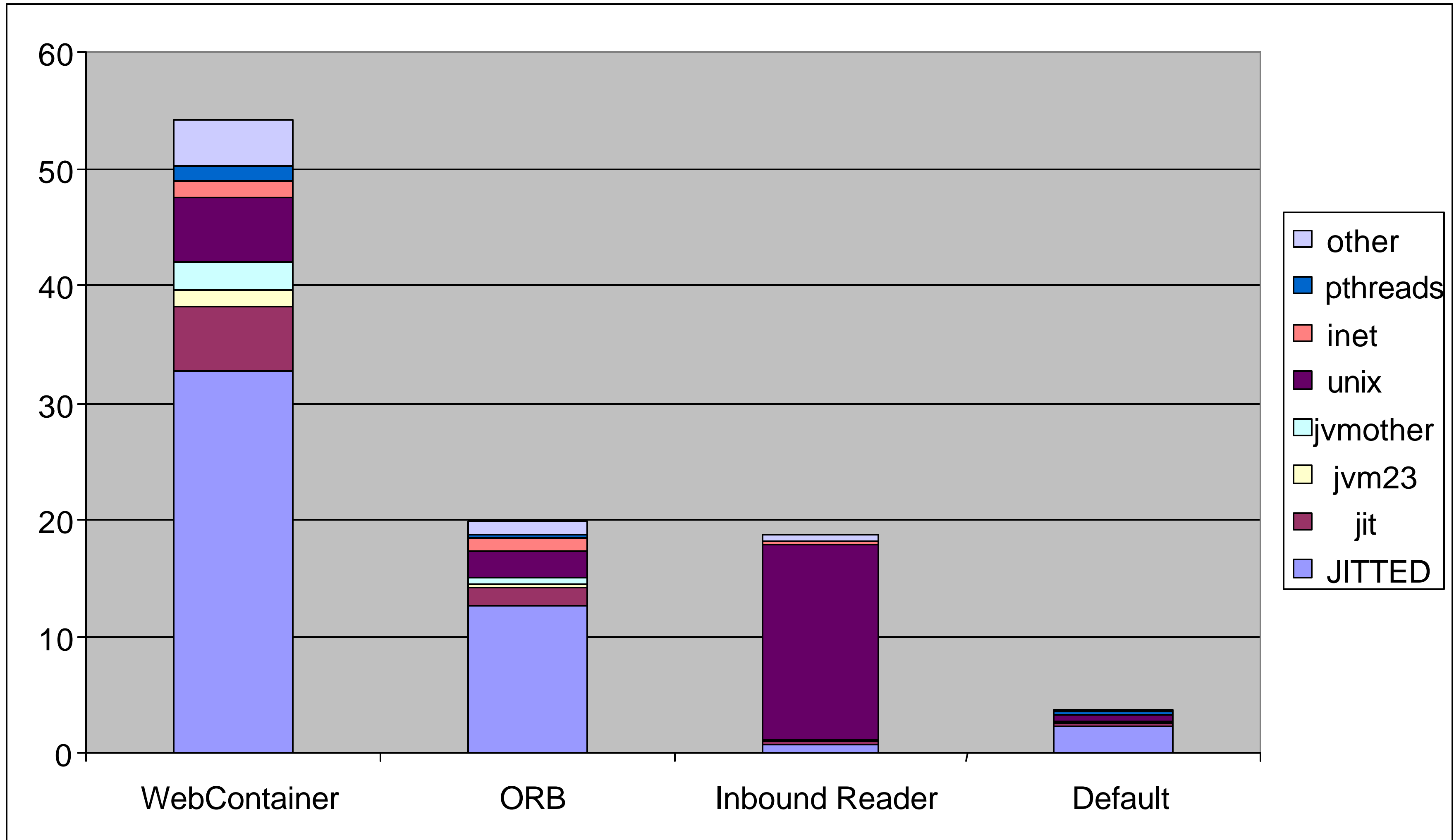
© 2003 IBM Corporation

Misses by Component



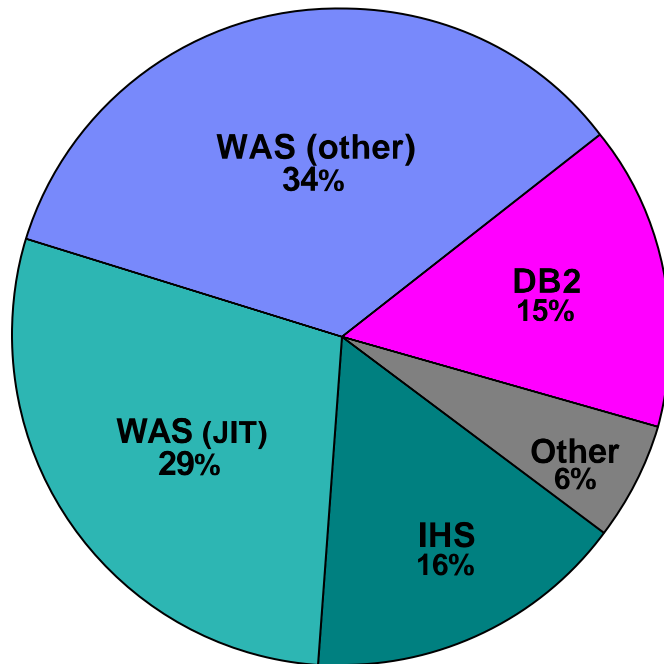
© 2003 IBM Corporation

Misses by Component



 © 2003 IBM Corporation

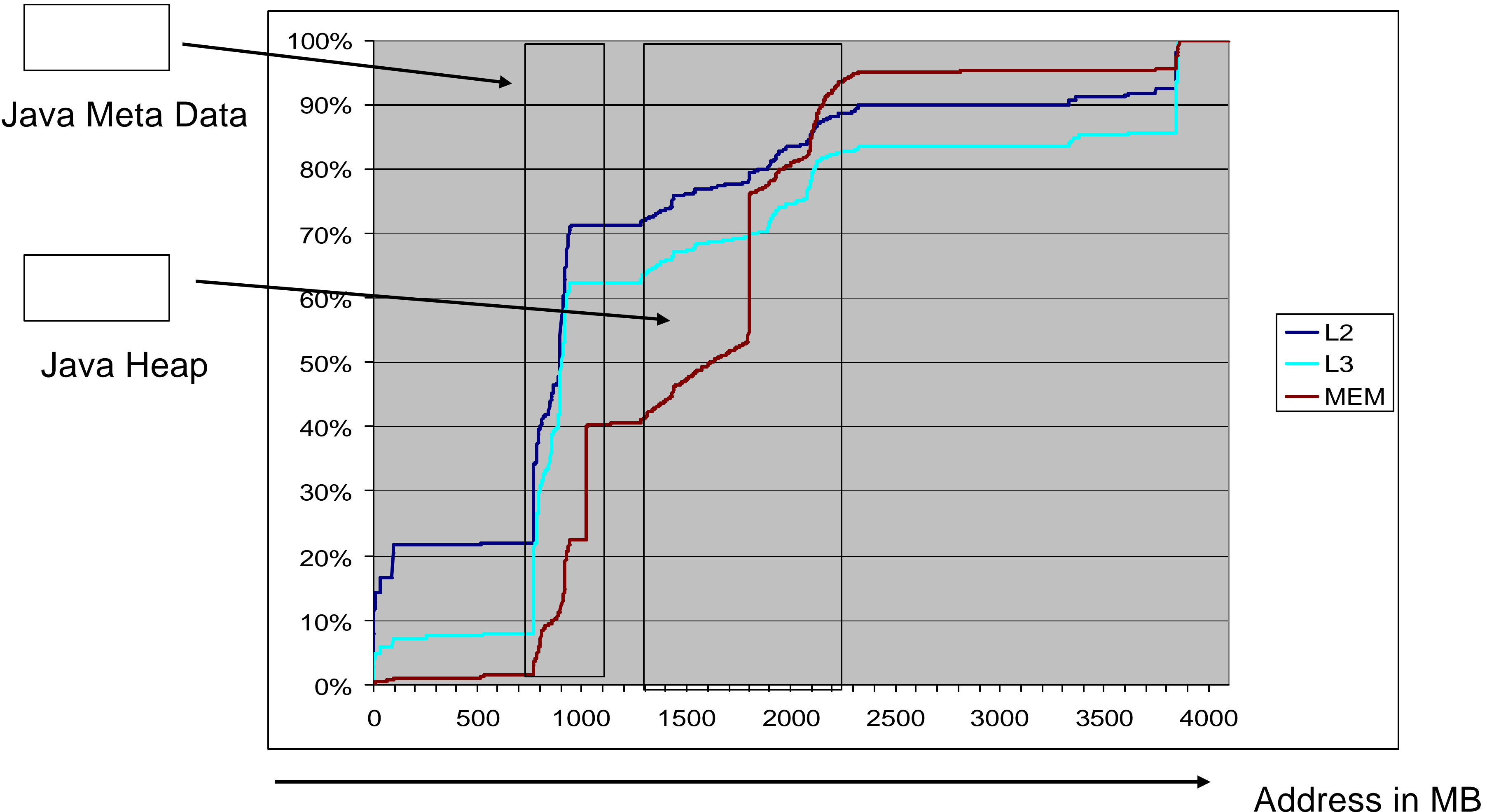
JIT Analysis



- **Data Collected from last 5 minutes of 60 minute run**
- **63% CPU time in WAS**
- **JIT'd Code in WAS (48% of WAS execution, 29% overall)**
  - Jas2004 JIT'd code: 3% of all JIT'd code
  - Enterprise Java Service <com.ibm.ejs>: 22% of JIT'd
  - WebSphere <com.ibm.ws>: 28% of JIT'd
- **Not-JIT'd Code in WAS (the other 52% of WAS execution time)**
  - 15% in kernel
  - 12% in libdb2.a
  - 11% in libmqmcs_r.a
  - 9% in libj9vm22.so
- **50% of JIT'd code execution is in 224 "hottest" methods**
  - Method profiel is "flat"
  - Data profile is also "flat"

 © 2003 IBM Corporation

L1 Miss Data Load Patterns: JAS2004

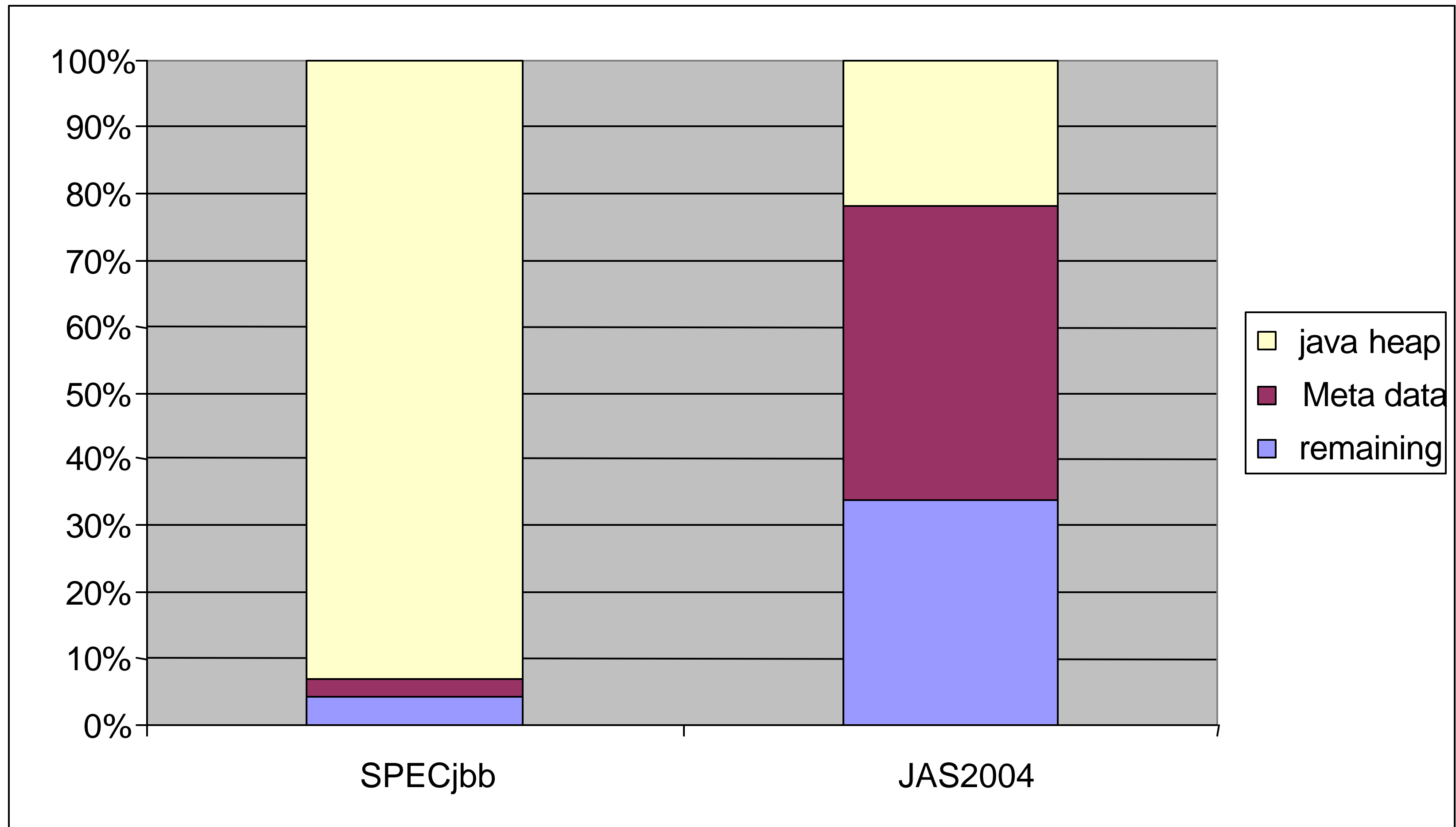


© 2003 IBM Corporation

Data Cache Misses

© 2003 IBM Corporation

Loads from L3 Classified by Region



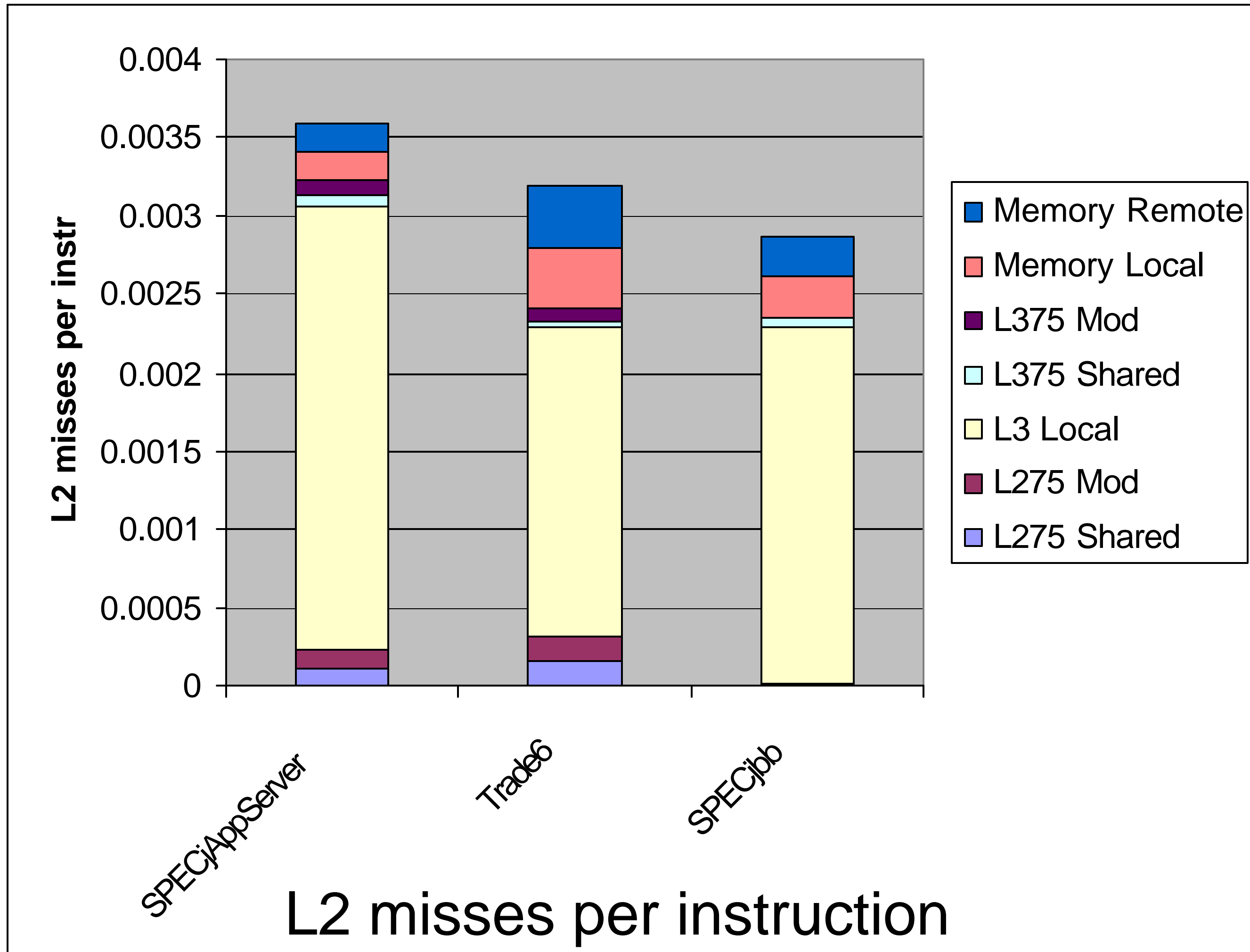
 © 2003 IBM Corporation

What is Meta-Data?

- JVM data structure not directly accessible by user application:
  - Object type information, Class information, Dispatch table, …
  - Mostly accessed via indirection
  - Heavily used in Java
    - Invokevirtual, invokeinterface, checkcast, instanceof, …

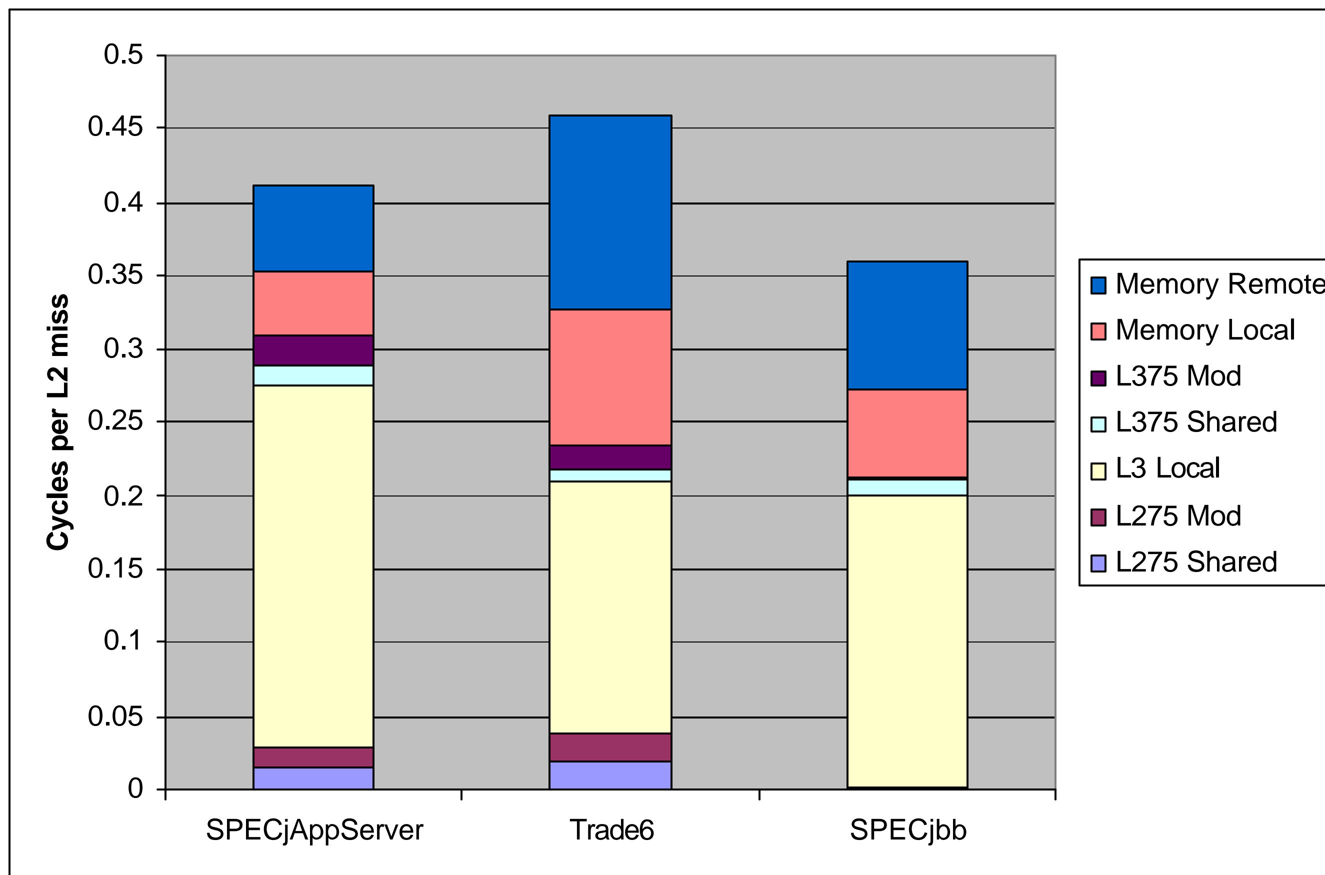
© 2003 IBM Corporation

Capacity or Communication?



 © 2003 IBM Corporation

Capacity or Communication?



L2 miss cycles per instruction (approx.)

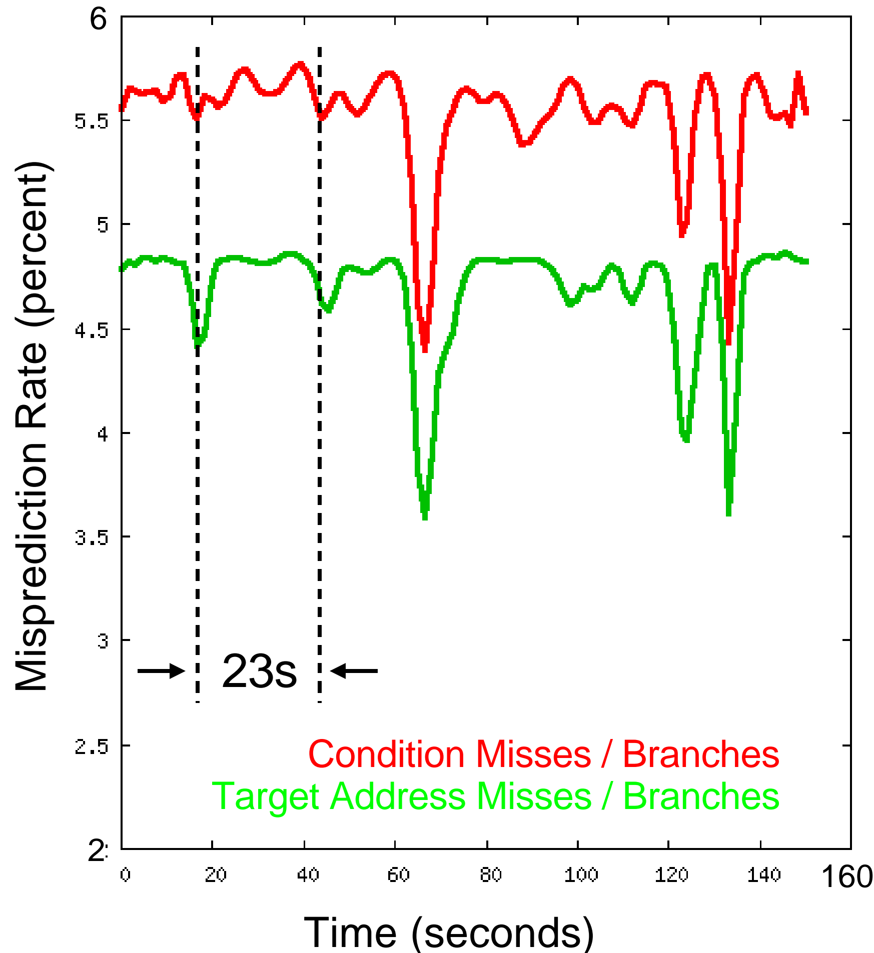
© 2003 IBM Corporation

Data Cache Performance: Summary

- Memory ops are common - almost 50% of instructions
- Stronger Load performance, relatively weaker Store performance
- Mostly capacity misses w/o many communication misses
- Object Meta-Data accounts for a large portion of D$ misses
  - Invokevirtual, invokeinterface, checkcast, instanceof, …

© 2003 IBM Corporation

Branch Misprediction



- **Relatively high as expected**
  - Correlated with GC events
- **Target Address (TA) Misses are strongly correlated with L1 I$ miss rate (0.9)**
  - TA misses *could* lead to fetching useless instructions, evicting useful data & instructions
- **No apparent L1 D$ pollution**
  - Low correlation between “speculation” rate and L1 D$ misses

Relatively high mispr. rate, insignificant correlation with CPI

 © 2003 IBM Corporation

Comparison (Aggregate View), AIX/Power4

A: TrWasSovS+U: Trade3, WAS, Sovereign, System+User, 4 CPUs
B: TrWasJ9S+U: Trade3, WAS, J9, System+User, 4 CPUs
C: TrWasJ9U: Trade3, WAS, J9, User, 4 CPUs
D: JbbJ9S+U: SpecJBB, J9, System+User, 4 CPUs
E: TPC-C: Native (C) Code 32 CPUs

	A: TrWasSovS+U	B: TrWasJ9S+U	C: TrWasJ9U	D: JbbJ9S+U	E: TPC-C
1: CPI	*3.846*	*3.780*	*3.313*	*1.79*	*3.682*
2: BR/Inst	*24.43 %*	*23.38 %*	*23.27 %*	*18.96 %*	*19.0140 %*
3: MPRED_CR/BR	**6.48 %**	**5.23 %**	**5.28 %**	**5.36 %**	**4.3095 %**
4: MPRED_TA/BR	**4.96 %**	**4.64 %**	**5.38 %**	**1.35 %**	**1.7619 %**
5: MPRED/BR	**11.44 %**	**9.87 %**	**10.64 %**	**6.71 %**	**6.0714 %**
6: MPRED/Inst:	**2.79 %**	**2.27 %**	**2.48 %**	**1.27 %**	**1.1546 %**
7: MPRED_CR/Inst	*1.58 %*	*1.22 %*	*1.23 %*	*1.02 %*	*0.7619 %*
8: MPRED_TA/Inst	*1.21 %*	*1.09 %*	*1.25 %*	*0.26 %*	*0.3927 %*

Small Java on J9 (D) shows very good CPI (1.79)

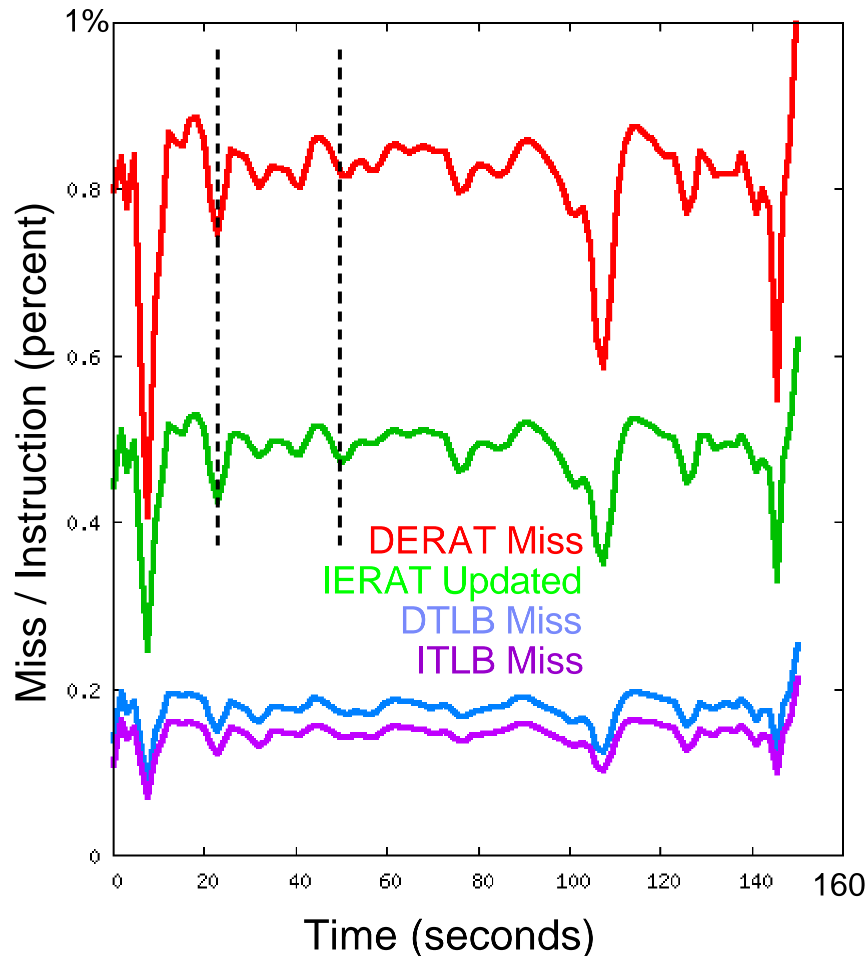
Branch Rate: WAS/apps (A – C) > small Java (D), Native code (E)

Branch Misprediction (CR: conditional): WAS/apps (A – C) > small Java (D), Native code (E), 2:1

Branch Misprediction (TA: target addr): WAS/apps (A – C) >> small Java (D), 4:1; Native (E), 3:1

© 2003 IBM Corporation

Address Translation



- **Tolerable frequency of TLB & ERAT misses**
  - 2 - 3 orders of magnitude fewer TLB misses during GC
    - Graph fitted using Bezier smoothing -- spikes actually correspond to events that take 0.2 - 0.3s. -- the time of a GC
  - ~500 instructions / DTLB miss
  - ~25% of DERAT misses result in a TLB Miss → can be expensive
- **Large Pages help!**
  - DTLB Miss Rate improved by 25%
  - ITLB Miss Rate improved by 15%

Much of working set is maintained in ERAT and TLB

© 2003 IBM Corporation

Outline

- End-to-End Optimization Project
  - Workload and Server Configurations
- Methodology
- **Performance Characteristics**
  - **Synchronizations**
- Summary

© 2003 IBM Corporation

Synchronization Overhead

- Mem-sync instructions can occur frequently in multi-threaded server code.
- Cost is relatively high

	Power4	Power5
isync	**30**	**10**
lwsync	**110**	**25**
sync	**140**	**50**
lwarx/stwcx	**80**	**75**

Cycle times for memory barriers and atomic read/write

© 2003 IBM Corporation

Locking Frequence

benchmark	*freq [ops/ms]*
jvm98_mtrt	120
jgf_mol	<< 1
jgf_monte	1650
jgf_ray	<<1
hedc	30
jigsaw	45
jbb (4wh)	1530
jbb (16 wh)	2320
trade6	630

 © 2003 IBM Corporation

Overhead in single-threaded benchmarks

- **IBM's commercial VM on 4-way Power5 1.6 GHz**
- **Removed sync-operations in the JIT code generator**

	speedup
jvm98_db	1.15
jvm98_jack	1.04
jvm98_javac	1.09
jgf_monte_A	1.02
jgf_monte_B	1.04
jbb (1 wh)	1.04

© 2003 IBM Corporation

Thread-Local Locks

Fraction of lock operations on thread-local locks [%]:

Benchmark	Fraction [%]
jvm98_mtrt	99.3
jgf_mol	82.7
jgf_monte	99.6
jgf_ray	81.3
hedc	89.6
jigsaw	45.5
jbb (4wh)	33.5
jbb (16 wh)	18.9
trade6	30.3

© 2003 IBM Corporation

Lock locality

scheduling dependence

implies thread-local

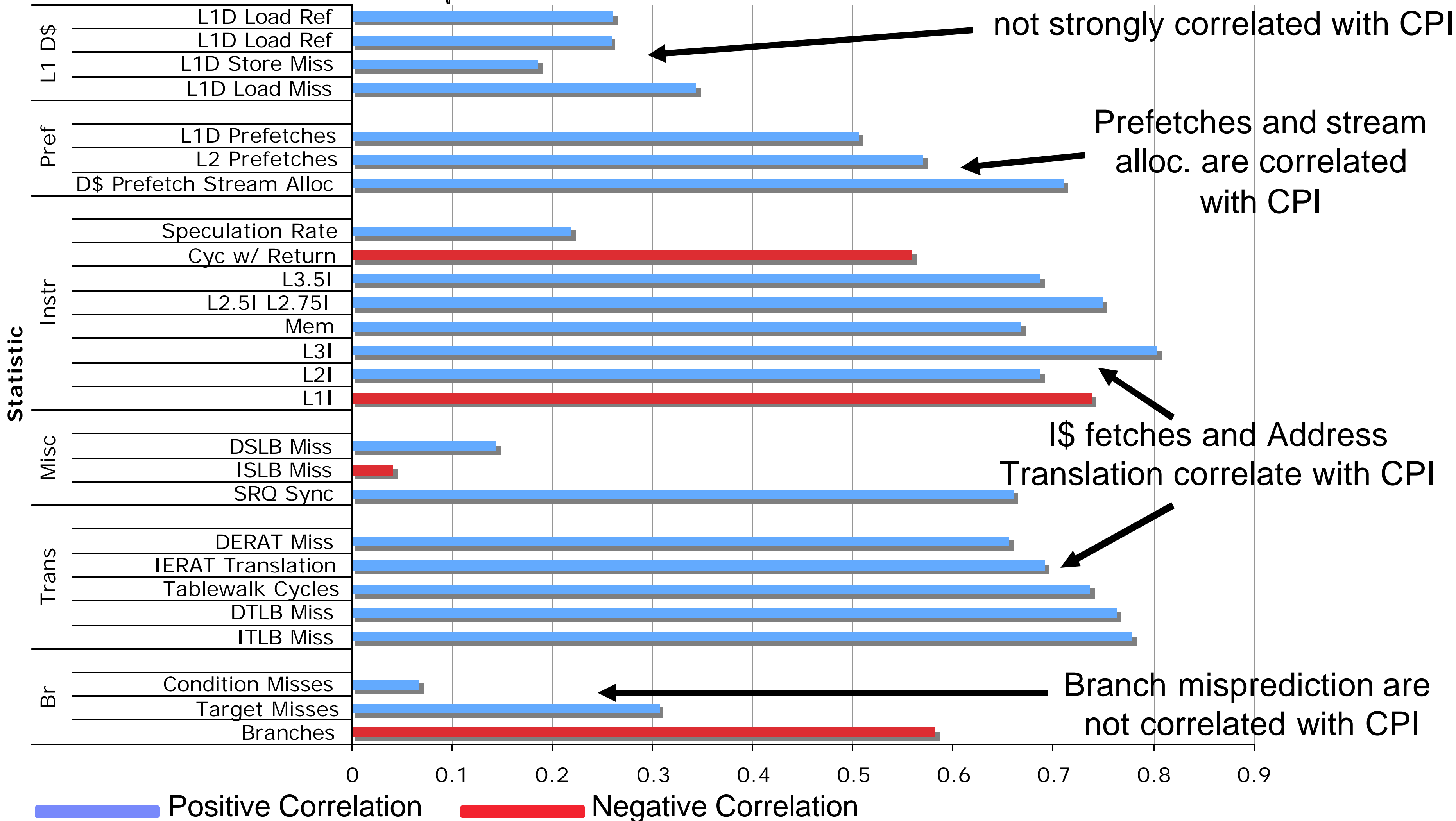
Dynamic fracti… …s that have locality to <...>:

	thr [%]	*proc [%]*	*thr or proc [%]*
jvm98_mtrt	**99.9**	**99.9**	**99.9**
jgf_mol	**98.4**	**98.2**	**98.5**
jgf_monte	**99.8**	**98.8**	**99.8**
jgf_ray	**99.0**	**99.0**	**99.2**
hedc	**97.2**	**97.6**	**98.1**
jigsaw	**84.8**	**91.8**	**91.8**
jbb (4wh)	**99.7**	**94.9**	**99.7**
jbb (16 wh)	**99.7**	**91.7**	**99.7**
trade6	**74.5**	**64.5**	**76.7**

© 2003 IBM Corporation

CPI Correlation

$$\frac{\sum(x-\bar{x})(y-\bar{y})}{\sqrt{\sum(x-\bar{x})^2\sum(y-\bar{y})^2}}$$



No single parameter is perfectly correlated with CPI – a balanced system

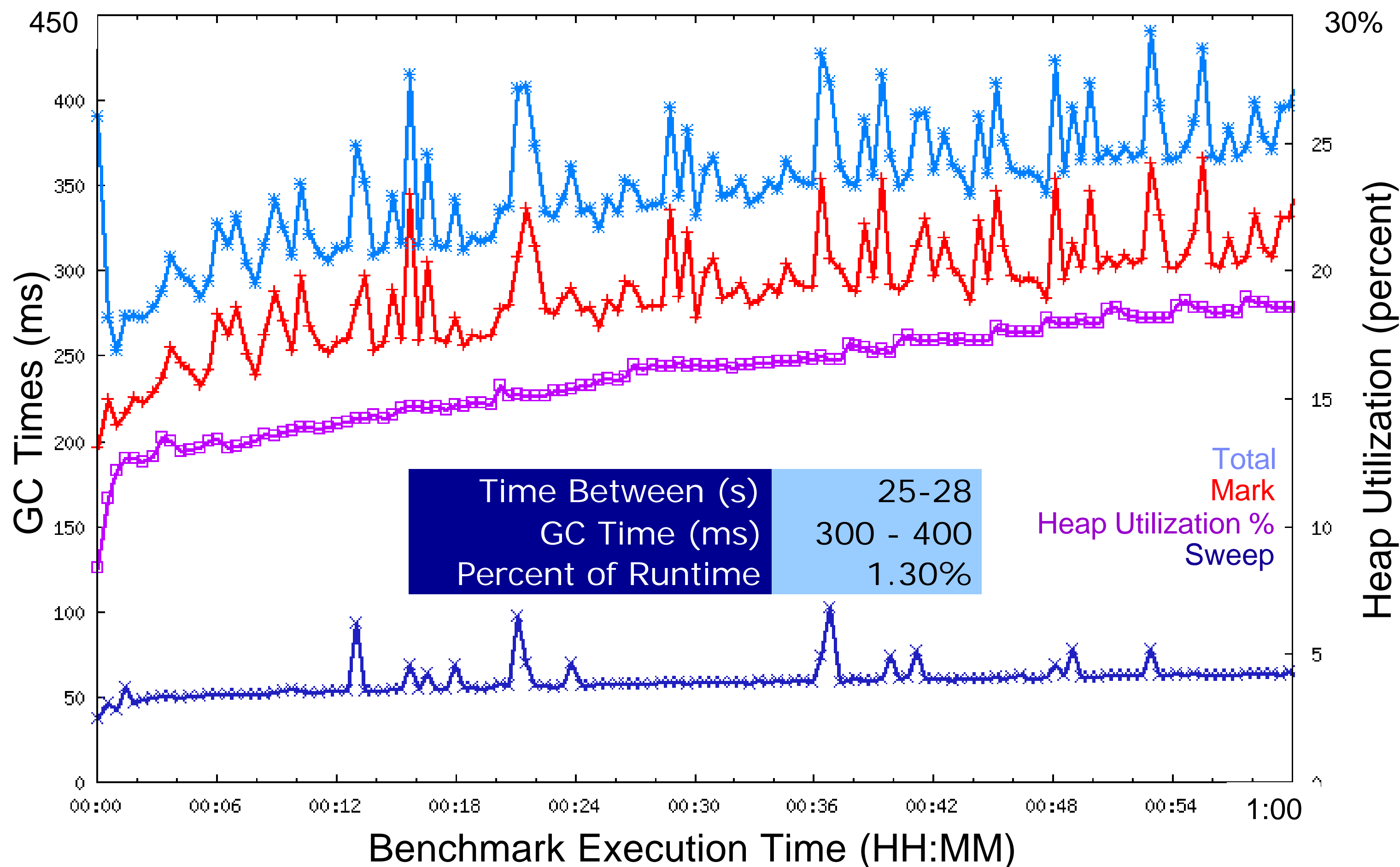
© 2003 IBM Corporation

Summary

- We have presented performance characteristics of Java server workloads
- Unlike a desktop system, GC is not a big issue
- They have higher branch-target misprediction rate
- Data cache miss rates are high
  - Java meta data cache miss is high
  - Mostly capacity misses, and low communication misses
- About 60% of CPU time is in Java, and ½ of that is in the jited code
- Method profile is flat
- Quite a large number of redundant memory sync operations
- No single performance metric has a very high correlation with CPI.

© 2003 IBM Corporation

Garbage Collection



GC not as significant as past characterization papers have shown

© 2003 IBM Corporation