A Case for Vertical Profiling

Matthias Hauswirth
Amer Diwan

University of Colorado
at Boulder

Peter Sweeney
Michael Hind

IBM Thomas J. Watson
Research Center

Finding Causes of Performance Phenomena

C Program

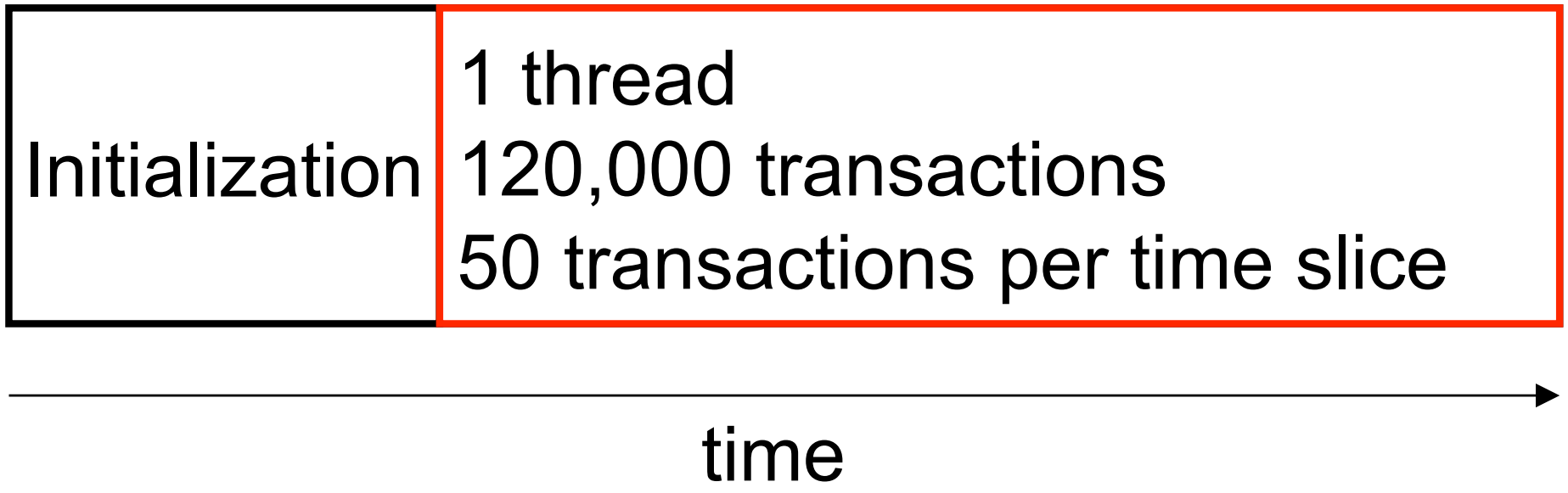
Application
Operating System
Hardware

Java / **.net** Program

Application
Framework
Java Library
Virtual Machine
Native Library
Operating System
Hardware

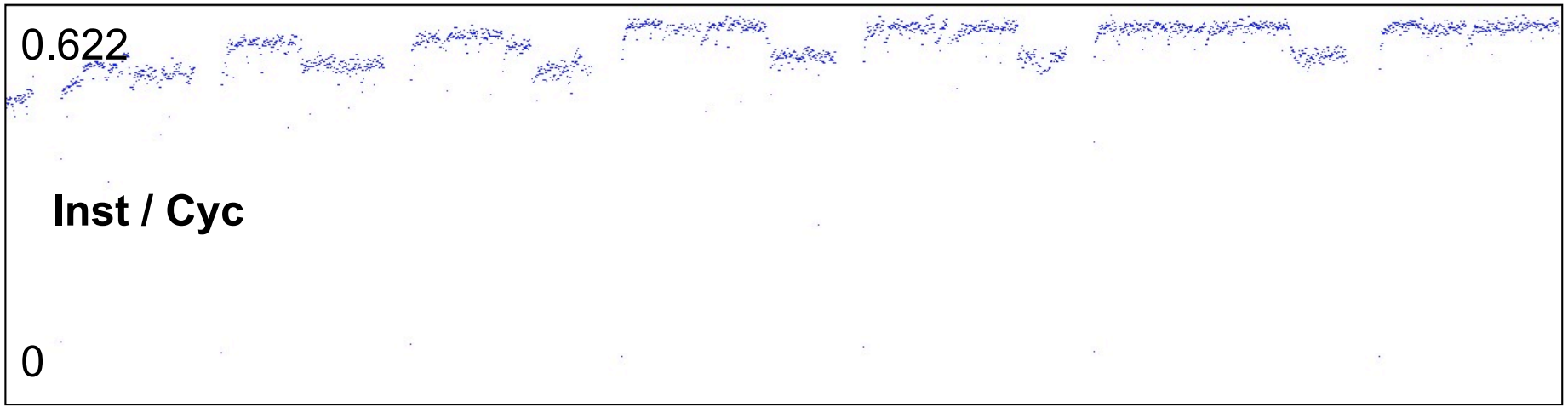
Methodology

- Benchmark: SPECjbb2000
- Virtual machine: JikesRVM



Expected Performance of Warehouse Thread

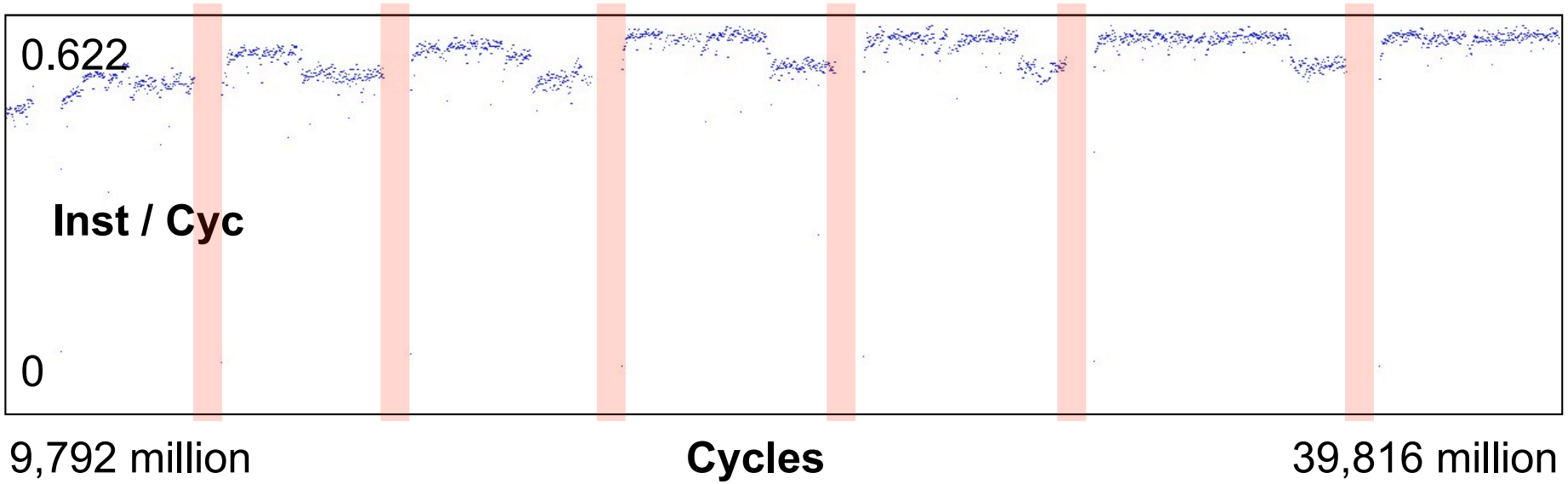
Observed Performance of Warehouse Thread



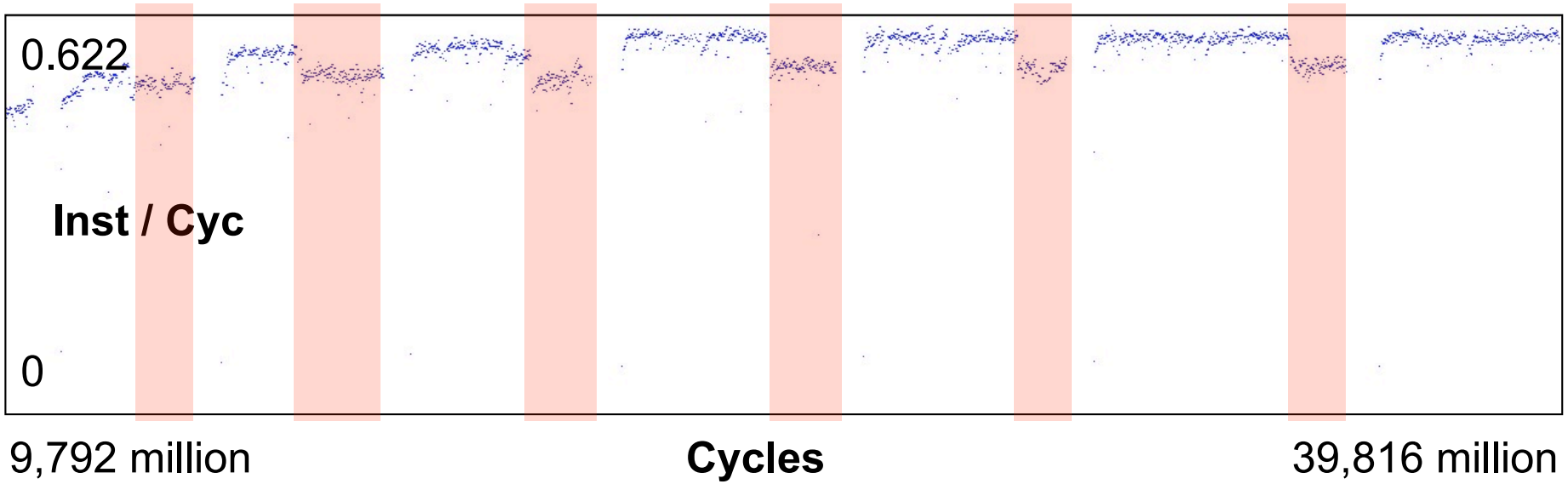
Investigation: Why this Difference?

- Correlate IPC with more than 100 other hardware performance metrics
  - No significant overall correlation

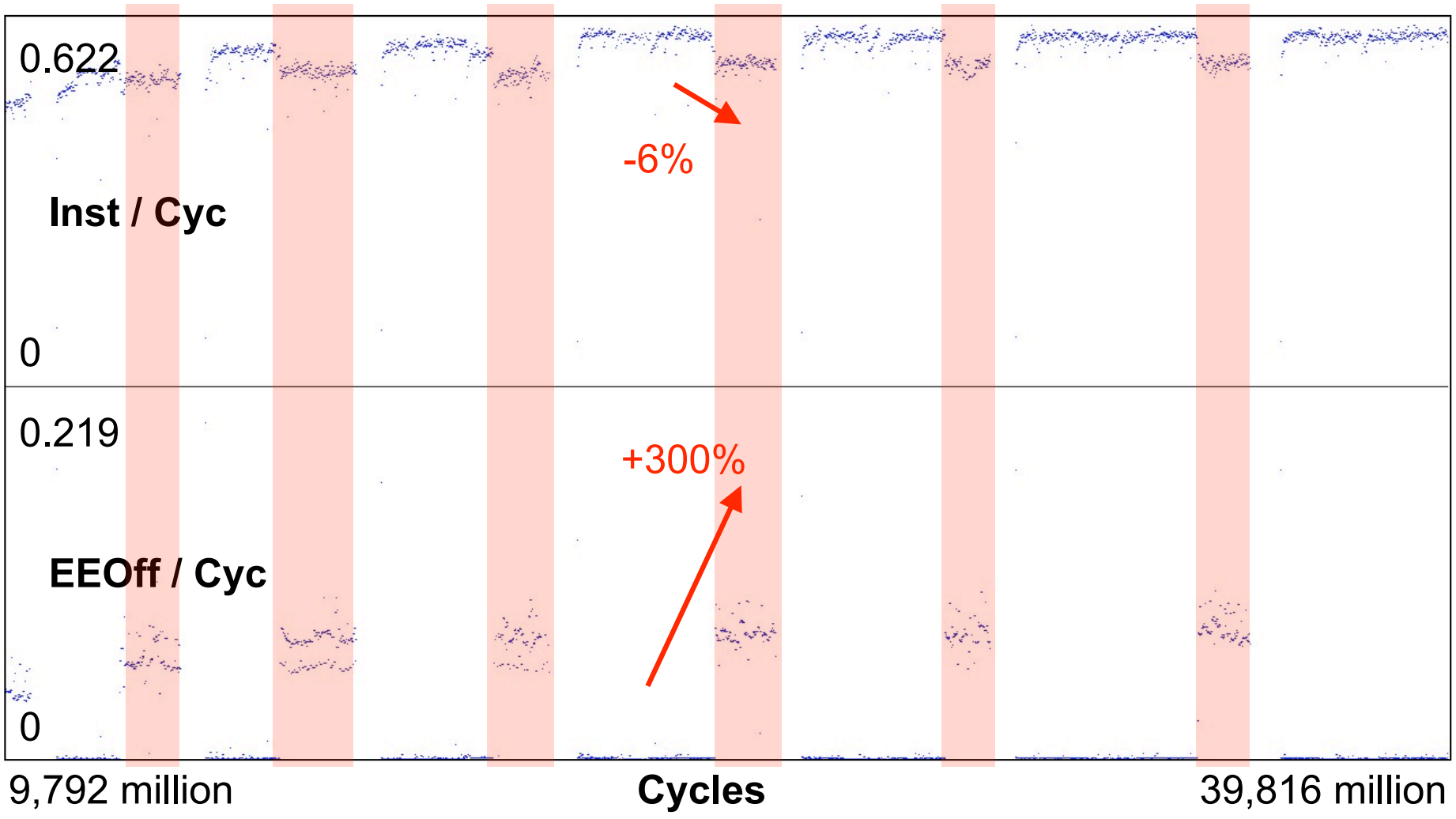
Investigation:
Correlate with GC Activity



Phenomenon ❶ Pre-GC Dip



Phenomenon ❶ Pre-GC Dip
Correlate with OS-Level Metric

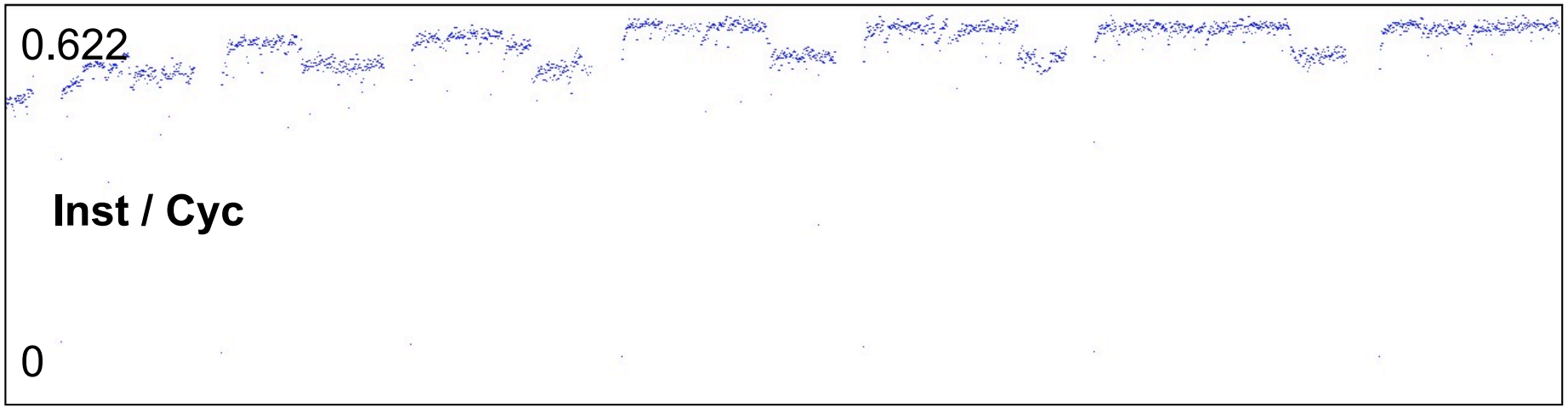


Phenomenon ❶ Pre-GC Dip
Next Steps

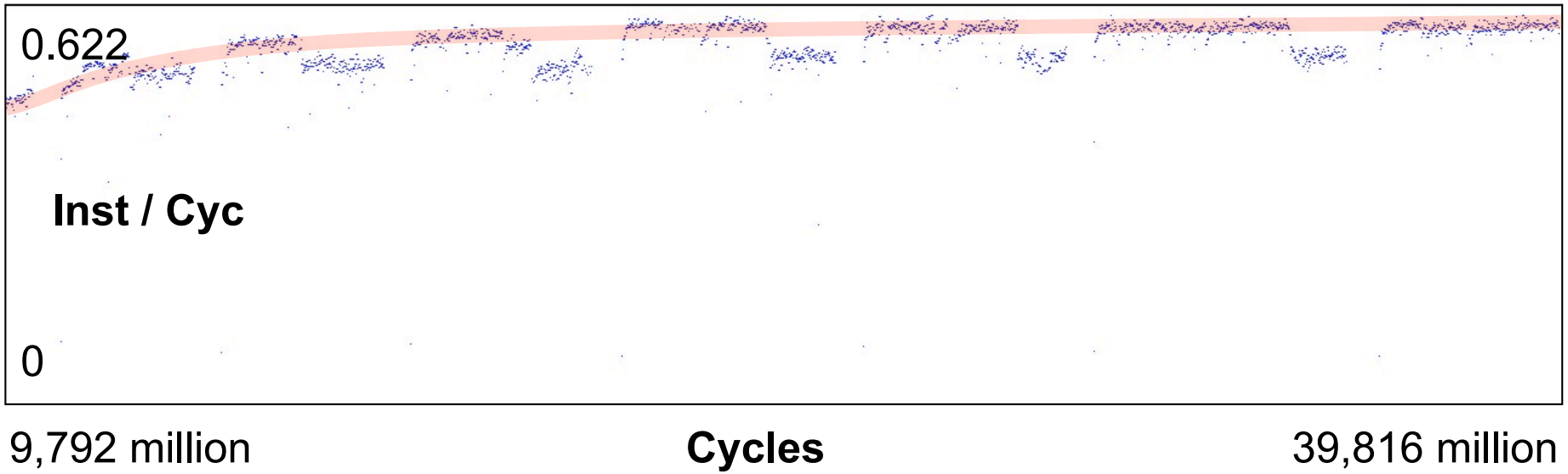
- We have not found the root cause yet…

- Need metrics from different levels:
  - Allocation
  - Synchronization
  - System calls
  - Interrupts

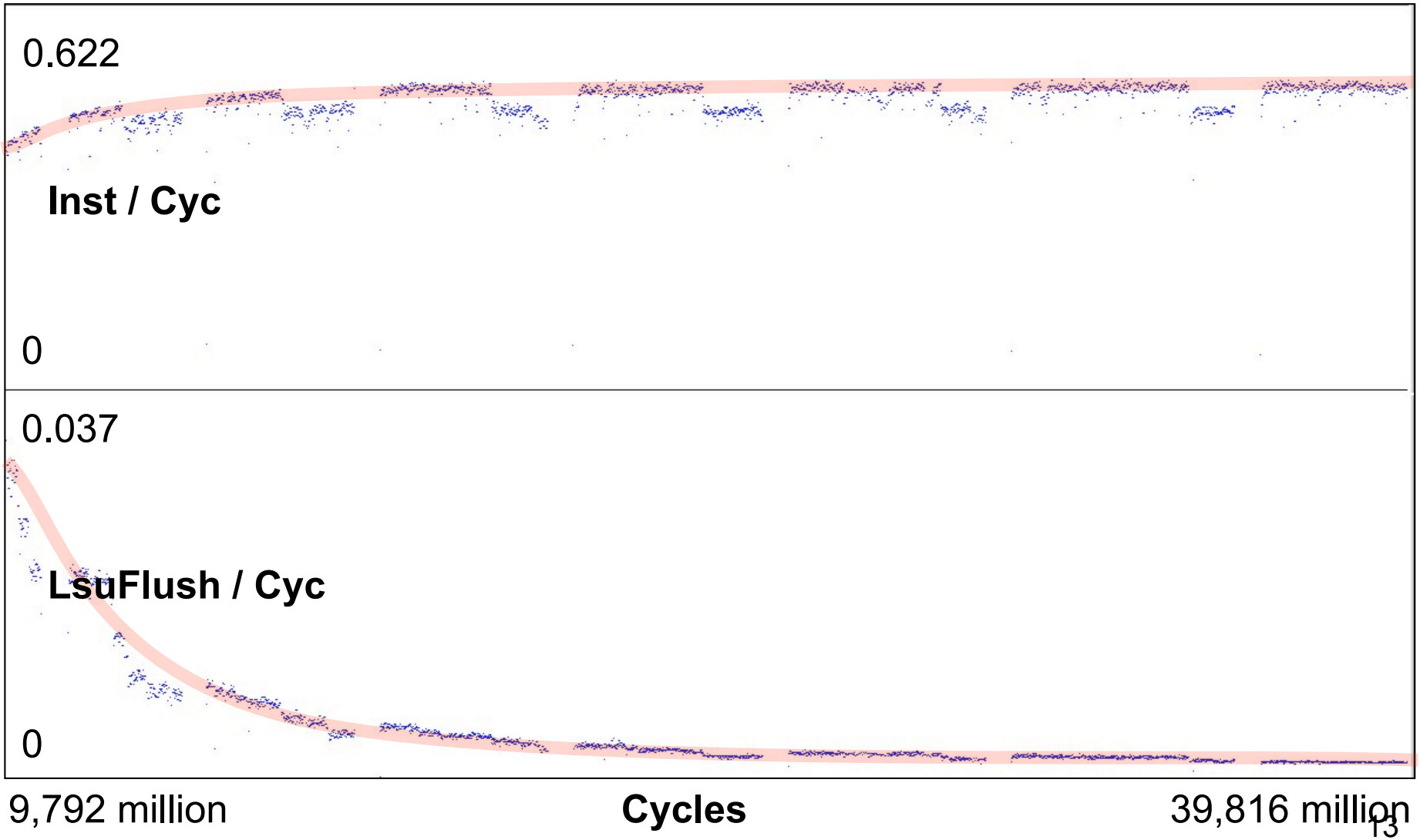
Observed Performance



Phenomenon ❷ Continuous increase



Phenomenon ❷ Continuous increase
Correlate with HW-Level Metric



Phenomenon ❷ Continuous increase Correlate with VM-Level

Metric	Non-Opt	AOS		Opt
		Start	End	
IPC	0.3479	0.4091	0.4890	0.5082
LsuFlush/Cyc	0.0533	0.0250	0.0017	0.0007

Phenomenon ❷ Continuous increase

Next Steps

- We have not verified the root cause yet…
- Need metrics from different levels:
  - Recompilation activity
  - Time spent executing non-optimized vs. optimized code

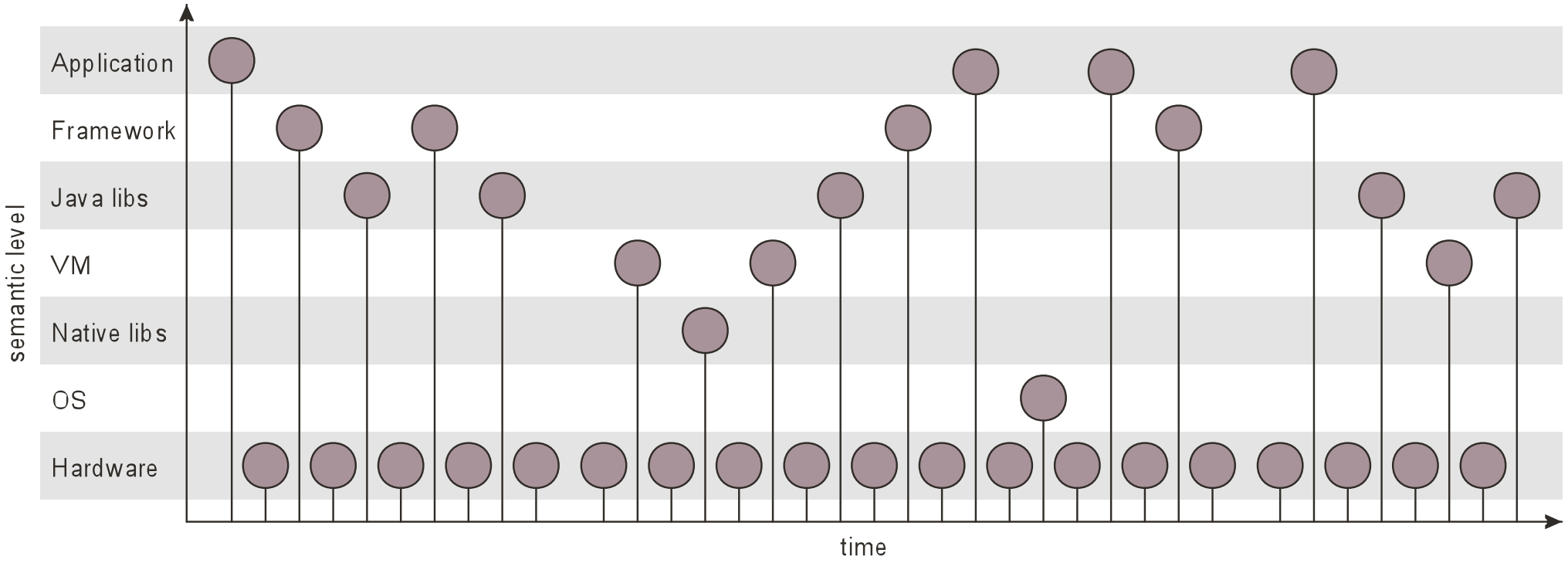
Vertical Profiling

- Gather data about multiple levels
  - Application
  - Framework
  - Java Library
  - ❶❷ Virtual Machine
  - ❶ Native Library
  - ❶ Operating System
  - ❶❷ Hardware

vertical

❶ Pre-GC Dip ❷ Continuous increase

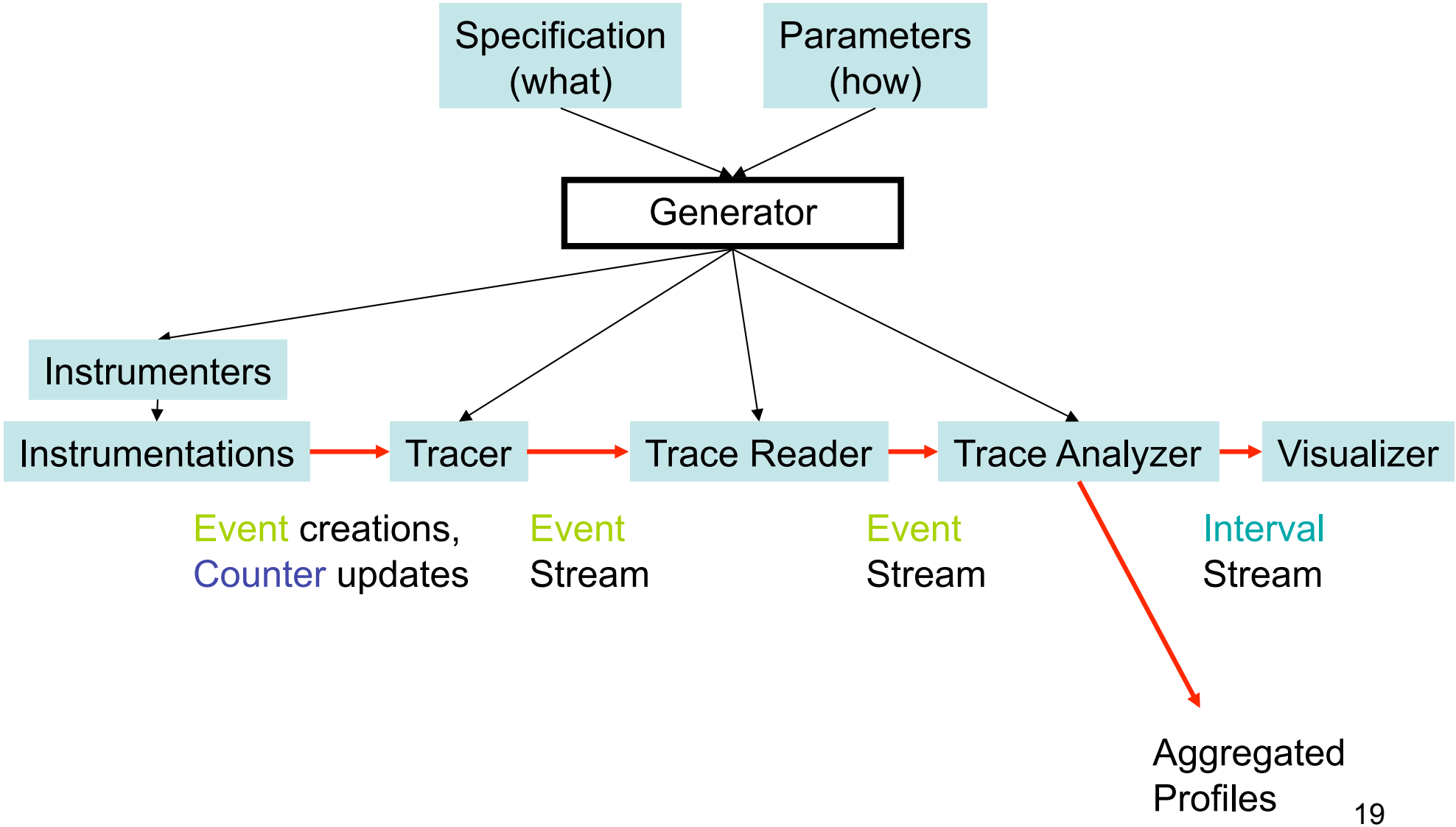
Vertical Event Trace



Challenges & Possible Approaches

- Huge difference in event frequencies
  - E.g. 7 GCs, but 20 billion instructions completed
  - Idea: Count high-frequency events, trace low-frequency events
- Large number of possible metrics
  - Trace everything: impossible to anticipate, too expensive
  - Write many specialized profilers: error prone, large effort
  - Idea: Generate profilers from specification
- Overhead
  - E.g. tracing every memory access is very expensive
  - Idea: Provide tunable profiling parameters for least overhead
- Perturbation
  - E.g. instrumenting every memory access perturbs HPMs
  - Idea: Use separate runs for interfering metrics
- Separate Traces
  - E.g. handling non-determinism
  - Idea: Combine traces using intervals to summarize

Architecture



Vertical Profiling Specification: **What** to Profile

```
specification IPC_And_BytesAllocated {
  hardware counter long   Cyc;
  hardware counter long   Inst;                                   Counters
  software counter long   BytesAllocated;

  event ThreadSwitch {
    int     fromThread;
    int     toThread;                                             Events
    long    cyc = Cyc;
    long    inst = Inst;
    long    bytesAllocated = BytesAllocated;
  }
  interval TimeSlice {
    starts with ThreadSwitch;
    ends with ThreadSwitch where end.fromThread == start.toThread;
    double          ipc = (end.inst-start.inst) / (end.cyc-start.cyc);   Intervals
    long            bytesAllocated = end.bytesAllocated – start.bytesAllocated;
  }
}
```

Event Attributes

Interval Metrics

Status

- Profiling
  - Hardware Performance Monitors [VM'04]
  - Software Performance Monitors
  - Specification-driven (early prototype)
- Visualization & Analysis
  - IBM Performance Explorer

Future Work

- Evaluate utility
  - Find root causes of phenomena
- Evaluate perturbation
  - Intra-level perturbation
    (e.g. HPM → HPM)
  - Inter-level perturbation
    (e.g. lock tracing → HPM)
- Semi-automate investigative process
  - Statistics / Machine learning

Related Work

- Trace Analyzer
  - [Perl 92] Performance Assertion Checking
  - [Perl et al. 98] Continuous Monitoring
- Software Performance Counters
  - [Microsoft] Windows Management Instrumentation
- HPM and JikesRVM
  - [Sweeney et al. 04] Using Hardware Performance Monitors to Understand the Behavior of Java Applications

Questions?

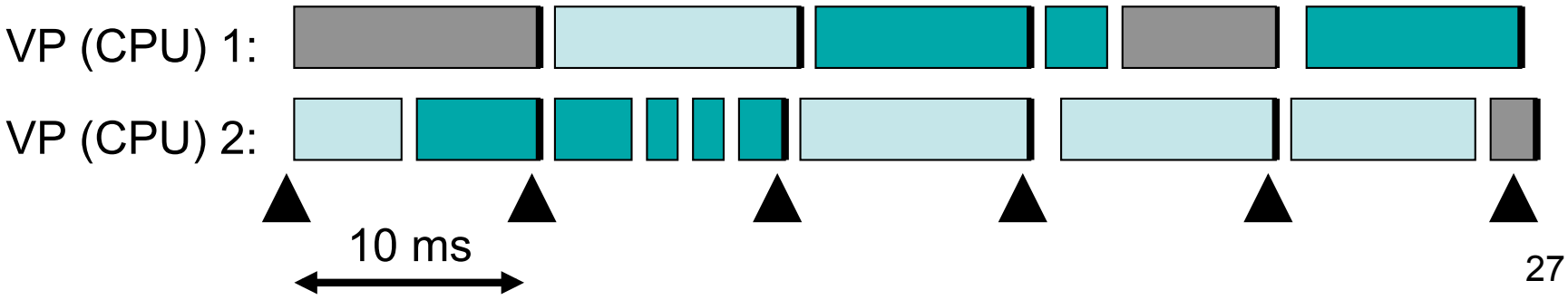
EXTRAS

Profiling HPMs: Infrastructure

VM	JikesRVM 2.3.0.1+ HPM Facility
C Library	AIX 5.x pmapi Library
OS	AIX 5.x pmsvc Kernel Extension
Hardware	Power4 Performance Monitors

Profiling HPMs: Samples

- A sample represents a time slice
  - Start and end time (in time-base or “decrementer” ticks)
  - 8 event counts
  - Processor id
  - Java thread id
  - Preempted or yielding
  - Java method ending the sample



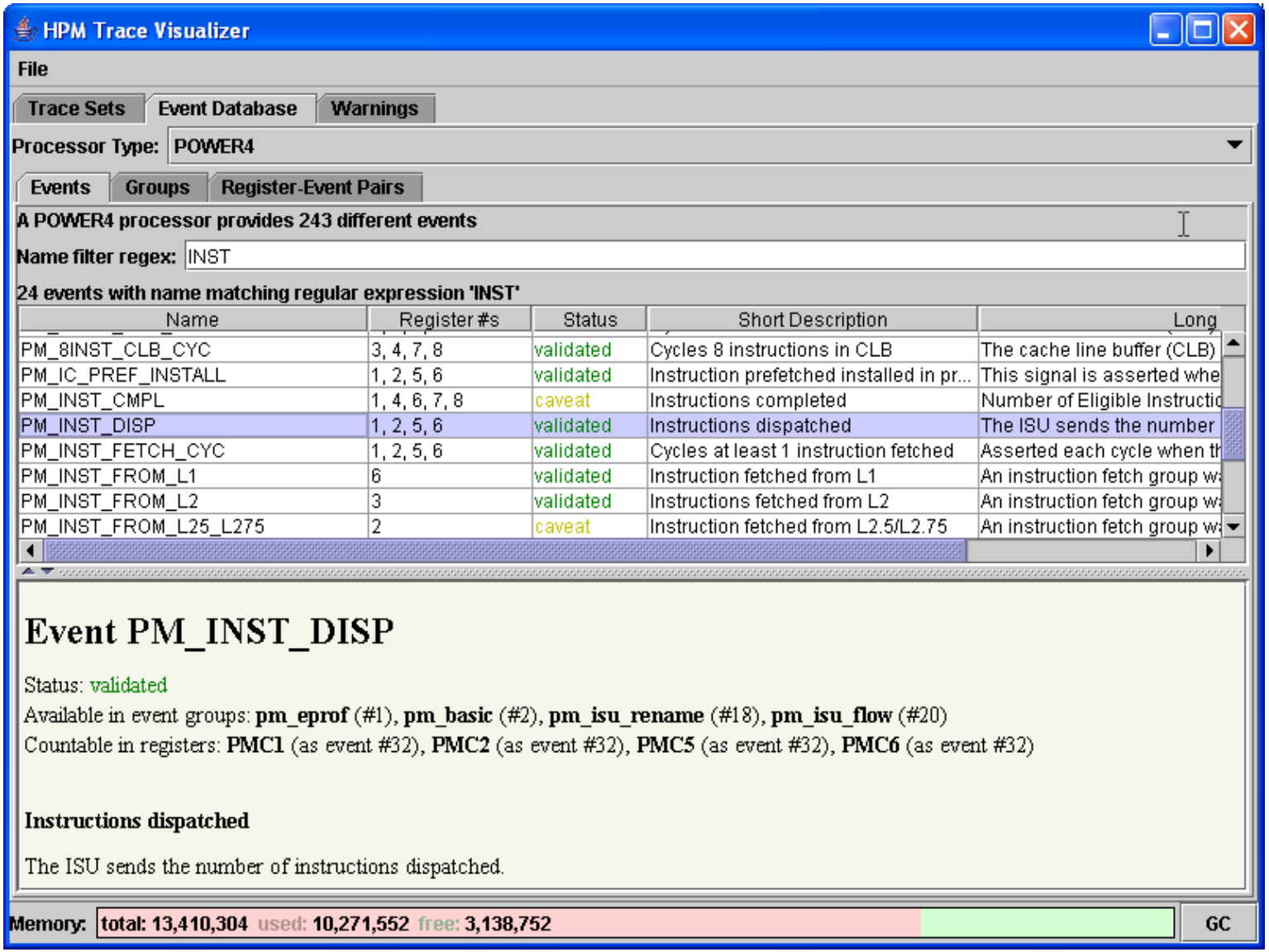
Profiling HPMs: Benchmark

- SPEC JBB
- Modified to execute a given number of transactions (120,000)
- Startup phase (ca. 8 sec)
  - 1 main thread
- Steady-state phase (ca. 24 sec)
  - *N* warehouse threads
- Configurations
  - {1,2,3,4} warehouses on {1,2,3,4} processors
- Steady-state behavior
  - Ca. 50 transactions per 10 ms time slice

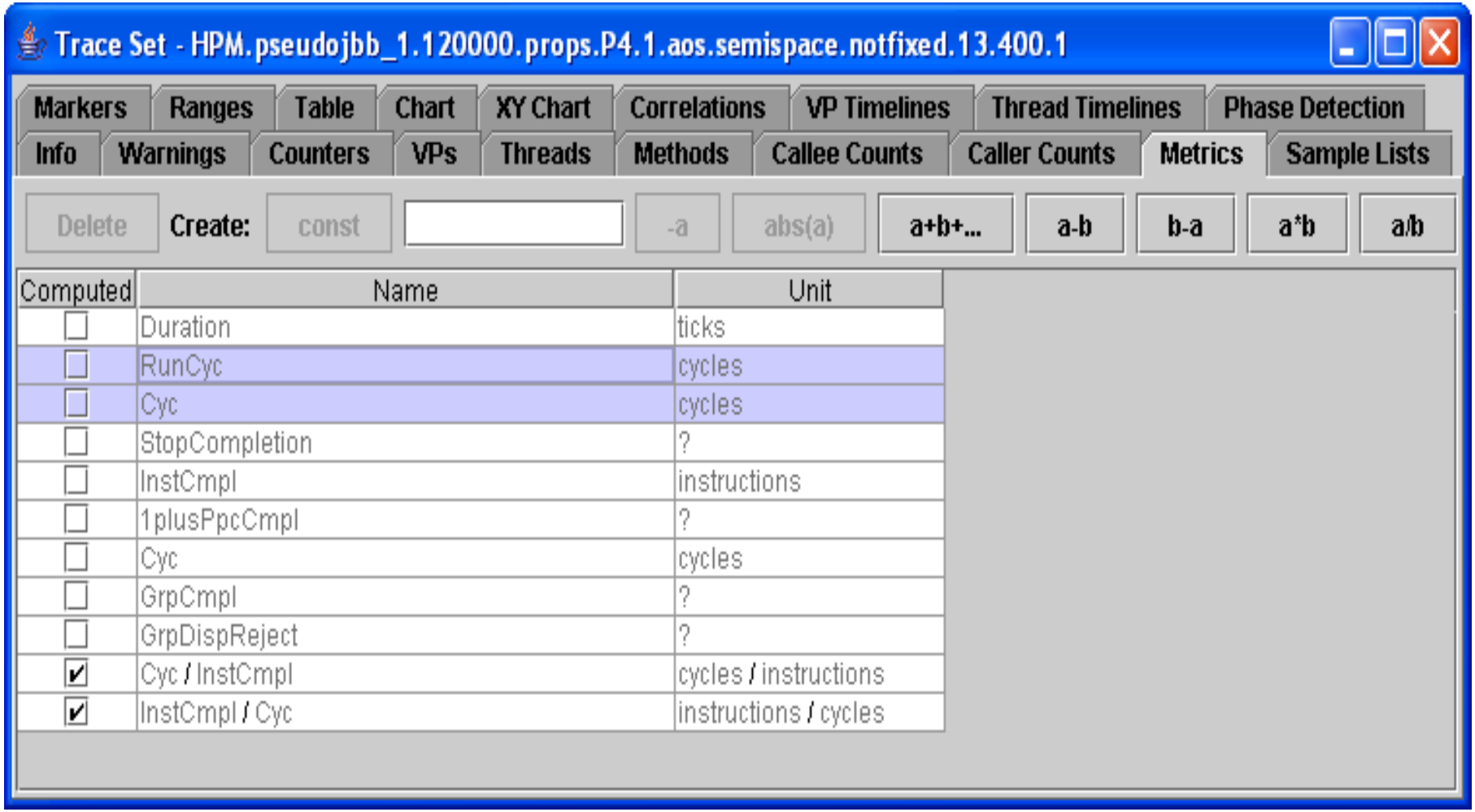
Performance Explorer

- Visualizer for JikesRVM hardware performance counter traces
- Built-in information about all Power4 performance events
- Support for creating computed metrics (e.g. *Inst*/*Cyc*, given *Cyc* and *Instr* counter values)
- Multiple visualizations, like time chart and scatter plot (for correlation of metrics)

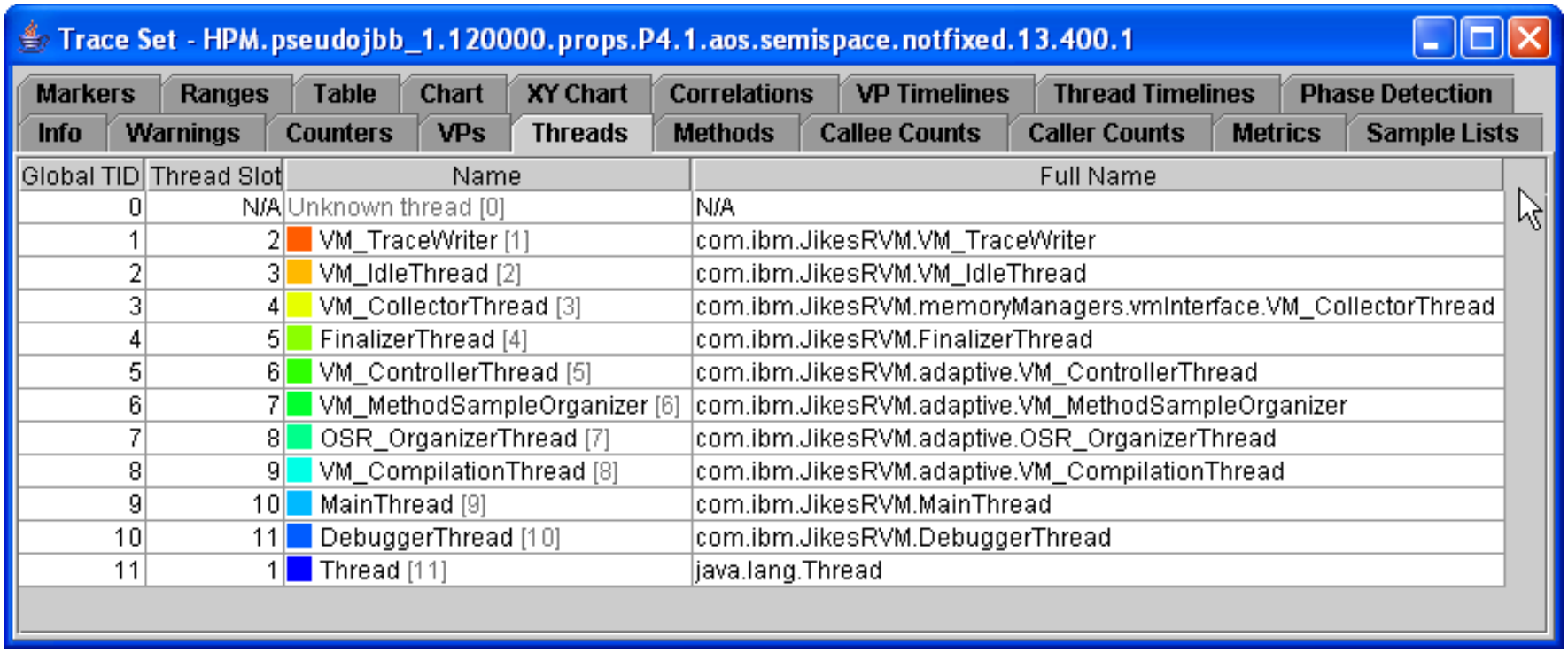
Performance Explorer: Power4 Event Information



Performance Explorer: Creation of Computed Metrics

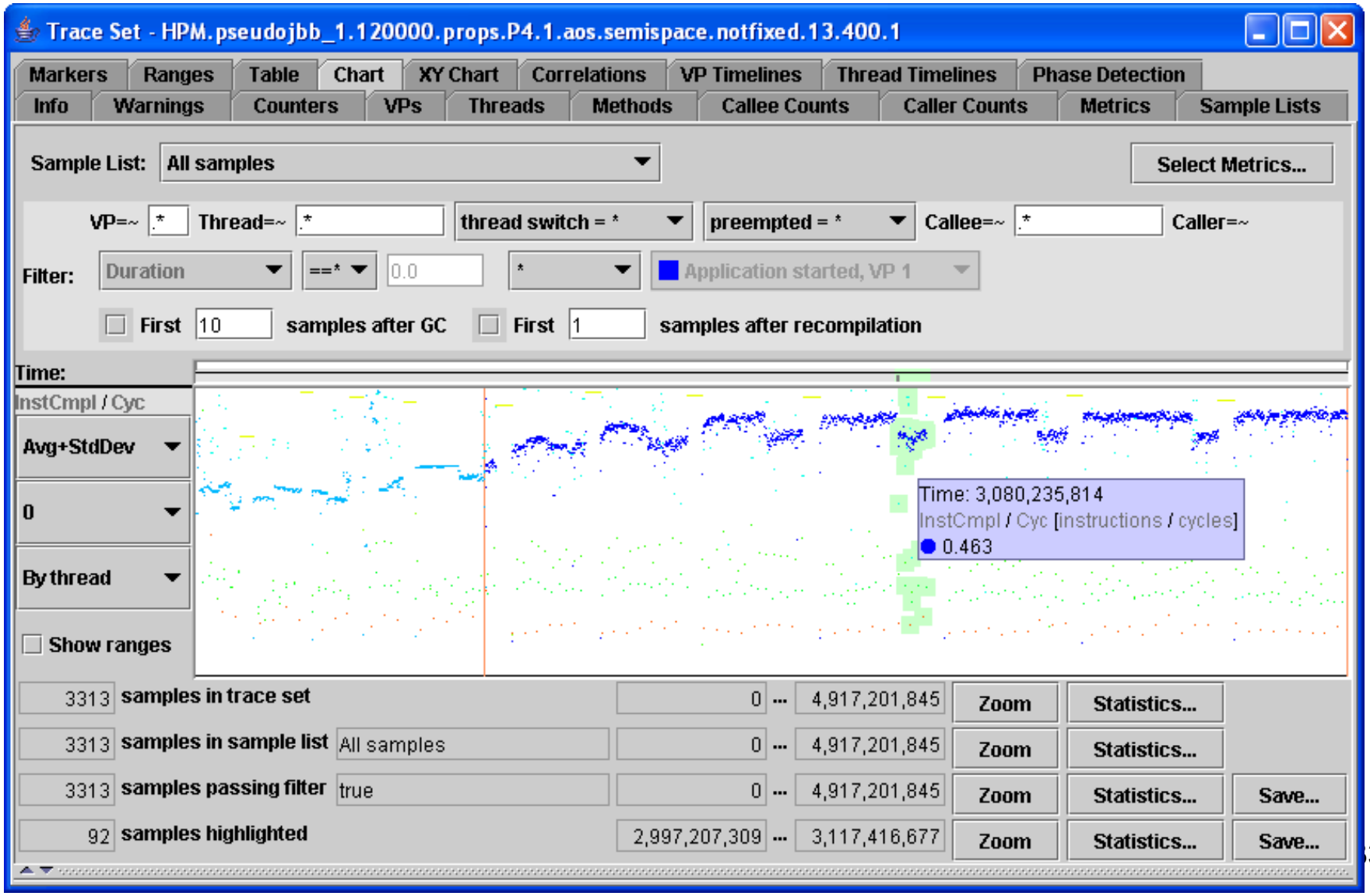


Performance Explorer: Overview of Java Threads

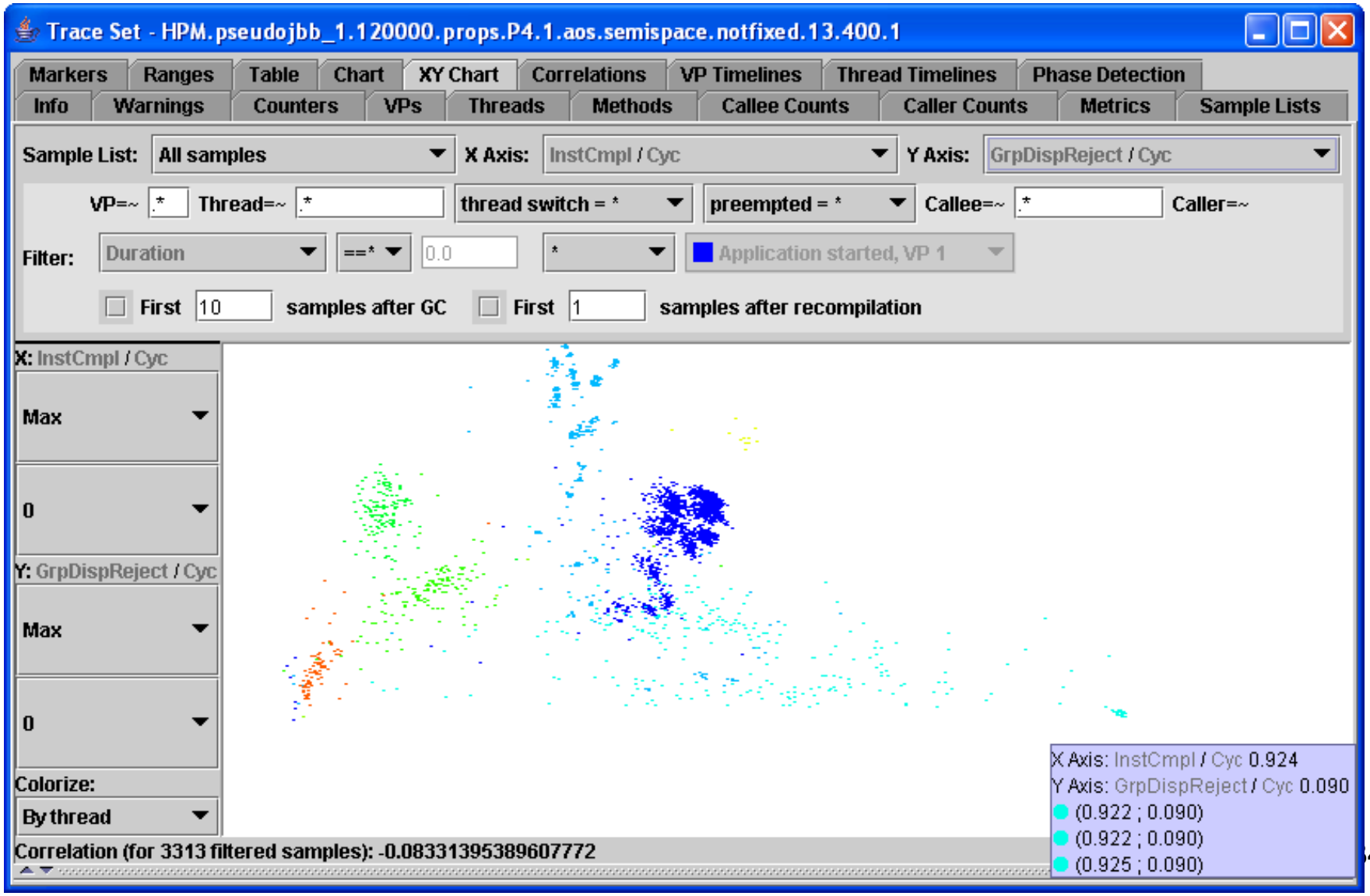


Global TID	Thread Slot	Name	Full Name
0	N/A	Unknown thread [0]	N/A
1	2	VM_TraceWriter [1]	com.ibm.JikesRVM.VM_TraceWriter
2	3	VM_IdleThread [2]	com.ibm.JikesRVM.VM_IdleThread
3	4	VM_CollectorThread [3]	com.ibm.JikesRVM.memoryManagers.vmInterface.VM_CollectorThread
4	5	FinalizerThread [4]	com.ibm.JikesRVM.FinalizerThread
5	6	VM_ControllerThread [5]	com.ibm.JikesRVM.adaptive.VM_ControllerThread
6	7	VM_MethodSampleOrganizer [6]	com.ibm.JikesRVM.adaptive.VM_MethodSampleOrganizer
7	8	OSR_OrganizerThread [7]	com.ibm.JikesRVM.adaptive.OSR_OrganizerThread
8	9	VM_CompilationThread [8]	com.ibm.JikesRVM.adaptive.VM_CompilationThread
9	10	MainThread [9]	com.ibm.JikesRVM.MainThread
10	11	DebuggerThread [10]	com.ibm.JikesRVM.DebuggerThread
11	1	Thread [11]	java.lang.Thread

Performance Explorer: Time Chart



Performance Explorer: Scatter Plot



Phenomenon ❶ Pre-GC Dip in IPC

Other Correlated Metrics

Metric	Normal	Dip	Increase
IPC	0.4924	0.46095	-6.4%
EeOff/Cyc	0.01965	0.0785	+300%
HvCyc/Cyc	0.02387	0.12489	+423%
GrpDispBlkSbCyc/Cyc	0.00595	0.02577	+333%
LsuSrqSyncCyc/Cyc	0.00612	0.017	+178%
StcxFail/StcxPassFail	0.00086	0.00395	+362%
LsuLrqFullCyc/Cyc	0.00077	0.00271	+250%

Vertical Profiling Matrix

Instrument: **Observe:**	Hardware	Machine code	Byte code	Source code
Hardware	◎	◎	◎	
OS	◎			◎
Native libs				◎
VM			◎	◎
Java libs				◎
Framework				◎
Application				◎

Vertical Profiling Matrix

- Two “vertical” dimensions
  - What we observe
  - What we instrument
- We may observe higher level behavior by instrumenting a lower level, or vice versa
  - Instrument HW, observe OS time
  - Instrument byte code, observe branch misses

Vertical profiling specification: **How** to profile

Parameter	Possible Values
Buffer size	**100000**, 1000000, 10000000, …
Buffer type	**Java byte[]**, Java int[], native
Buffer ownership	**Global**, Processor, Thread
Buffer access synchronization	**None**, Lock-free, Locked
Buffer access	**Java**, Magic
Buffer overflow handling	**Flush**, Disable, Ignore
Buffer flushing	**Explicit**, Seg fault, Each thread switch
Buffer flush target	**File**, Socket, C routine