Relative Factors in Performance Analysis of Java Virtual Machines

Dayong Gu Clark Verbrugge
School of Computer Science, McGill University
Montréal, Canada
{dgu1,clump}@cs.mcgill.ca

Etienne M. Gagnon
Département d'informatique, Université du Québec à Montréal
Montréal, Canada
egagnon@sablevm.org

June 15, 2006

Outline

Have you ever ...

As an application developer

- Rewritten the code for better performance
- But, obtained no speedup as expected?

Application

Compiler

VM

GC

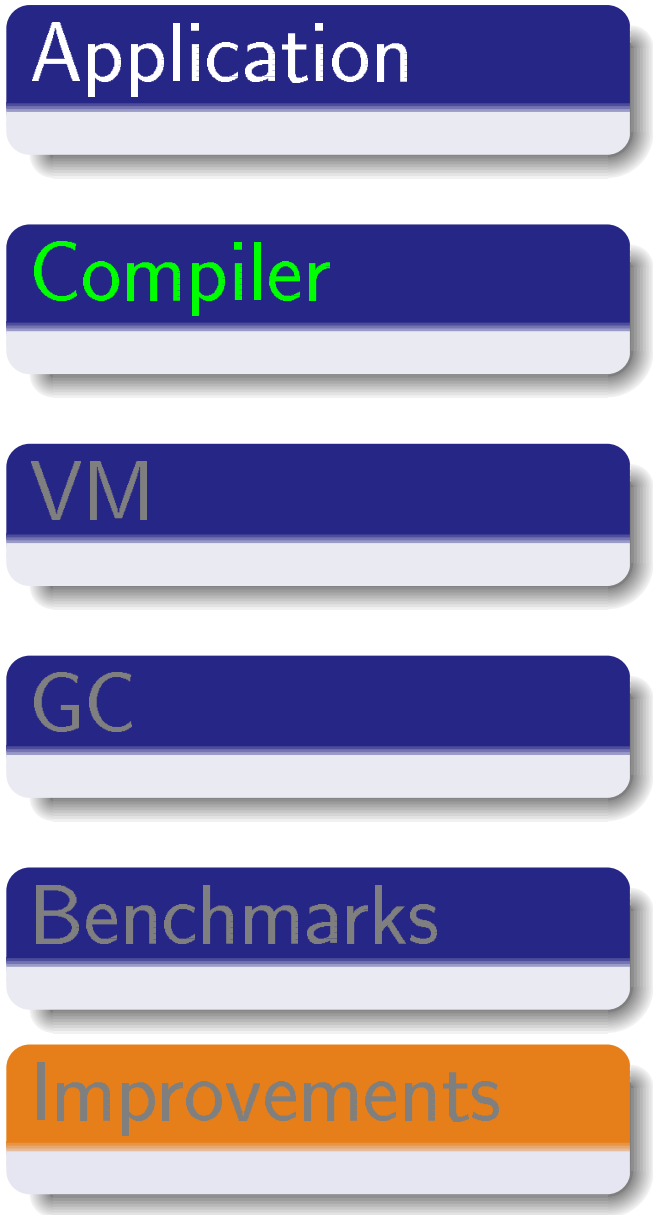
Benchmarks

Improvements

Have you ever ...

As a compiler developer

- Built a new optimization based on a neat analysis
- But, got no improvement or even slow down benchmark programs?



Have you ever ...

As a VM developer

- Applied a new technique inside a VM
- But, achieved no positive result, or only had a positive result by chance?

Application

Compiler

VM

GC

Benchmarks

Improvements

Have you ever ...

As a garbage collection (GC) developer

- Built a theoretically efficient GC algorithm
- But, the collector refused to run any faster or gave random results?

Application

Compiler

VM

GC

Benchmarks

Improvements

Have you ever ...

When measuring a set of benchmarks

- Found naughty benchmarks
- Disagreed with others, gave very strange or random results?

Application

Compiler

VM

GC

Benchmarks

Improvements

Have you ever ...

Improvements?

- Failed in reproducing the improvement of a published work
- Got a 10% improvement on a platform, but it disappeared after you got a new machine

Application

Compiler

VM

GC

Benchmarks

Improvements

Motivation

Goal

- Lots of us may have such kind of experiences
- How to understand these situations?
- Performance measurement analysis

Application

Compiler

VM

GC

Benchmarks

Improvements

Motivation

Performance measurement is difficult

- Computers are getting increasingly complex
- Many factors can affect measurement results, which are important?
- Virtual execution environments bring an extra layer and become even more challenging
- Study the relative factors in performance analysis

Motivating Example

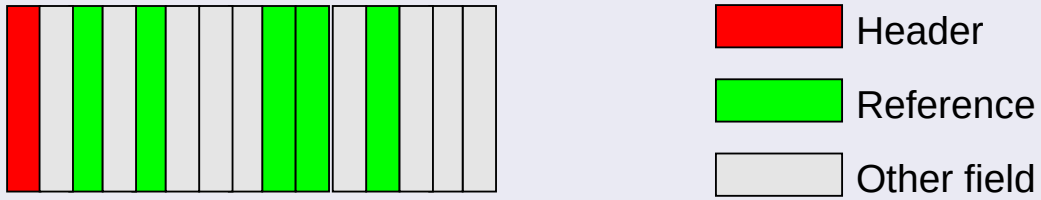
We developed a GC-related technique and observed surprising behaviors

Questions:

- What factors can impact performance?
- How large can the impact be?
- How can we explain the observed behaviors?

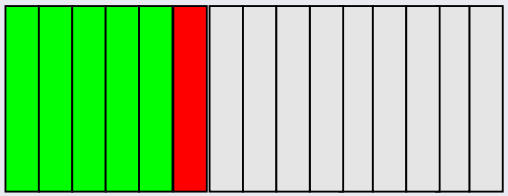
Motivating Example

Original Object Layout



References are located separately

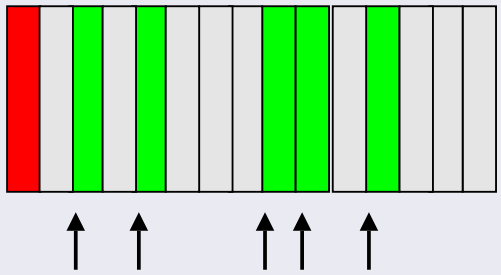
New: Reference Section (RS)

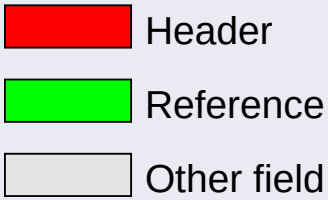


Group all references in a contiguous section

Motivating Example

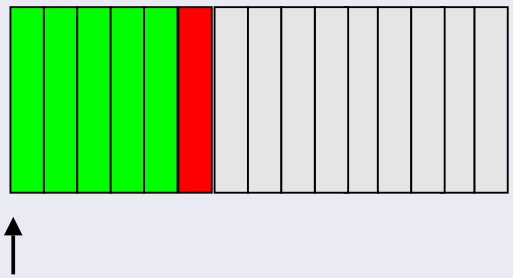
Original Reference Tracing





Get the the addresses of references one by one

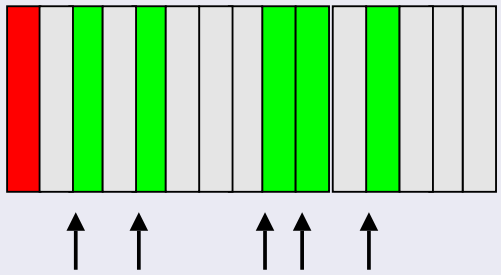
RS Reference Tracing

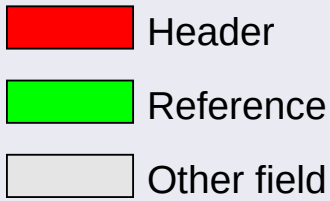


Get the first reference's address

Motivating Example

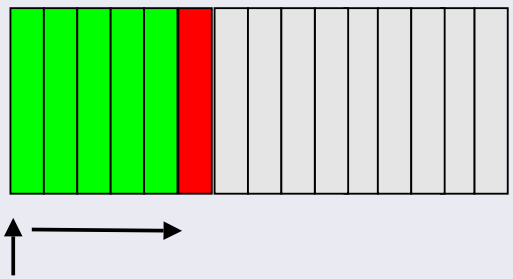
Original Reference Tracing





Get the addresses of references one by one

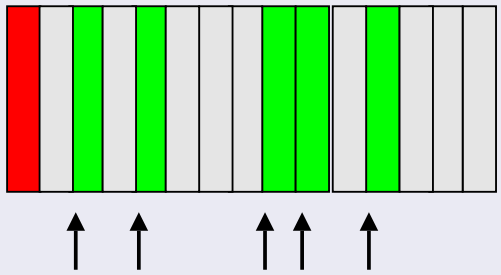
RS Reference Tracing

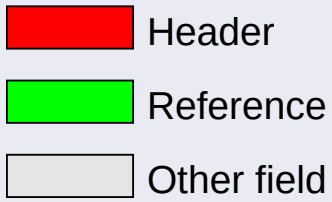


Scan the whole section immediately

Motivating Example

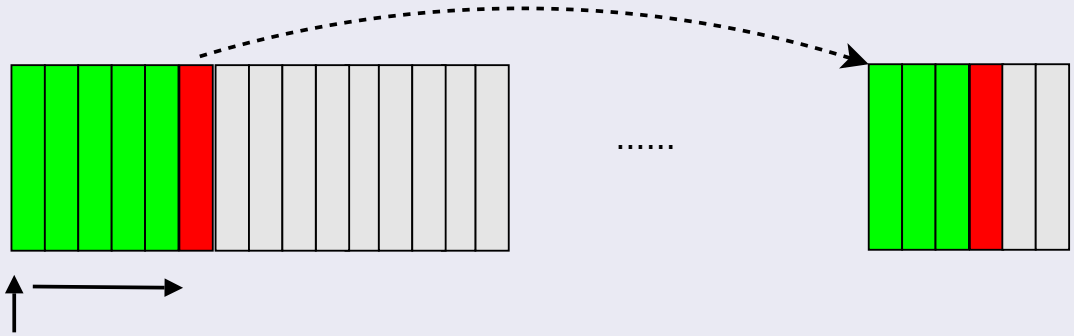
Original Reference Tracing





Get the addresses of references one by one

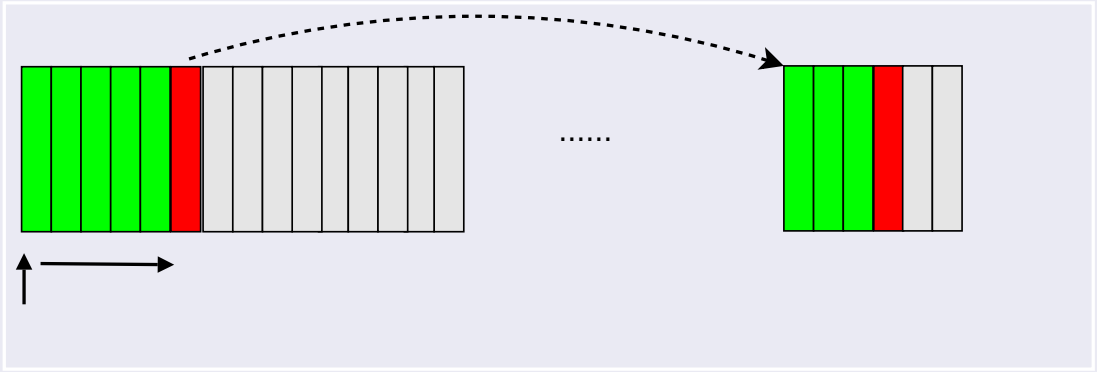
RS Reference Tracing



Jump to the next section

Implementation

RS Tracing Technique



- Useful for all tracing GCs
- Implemented in two JVMs

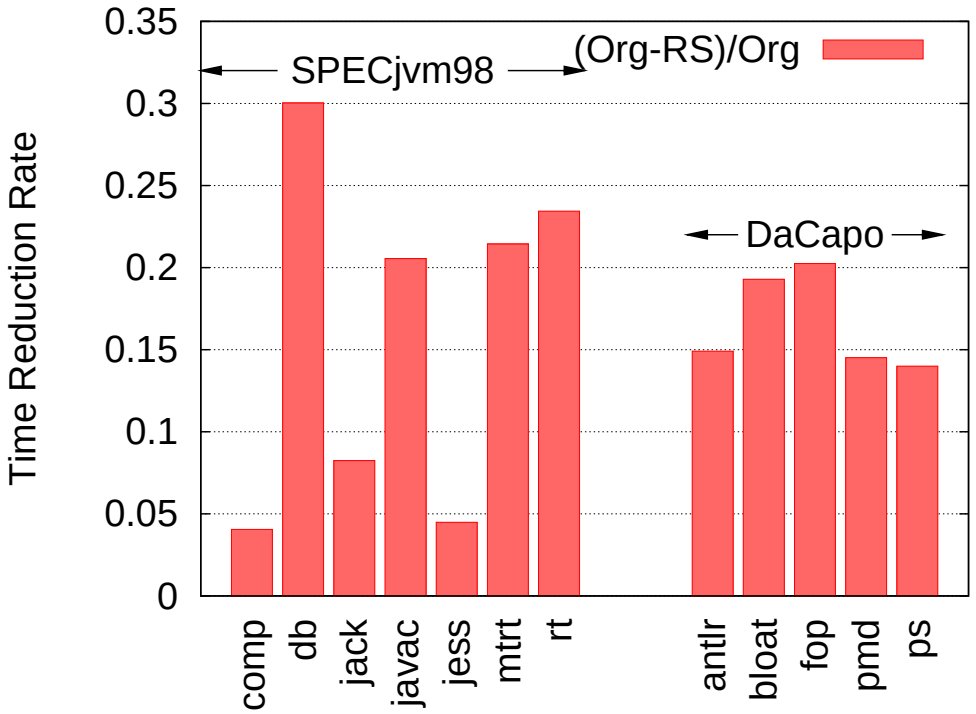
SableVM

An interpreter

Jikes RVM

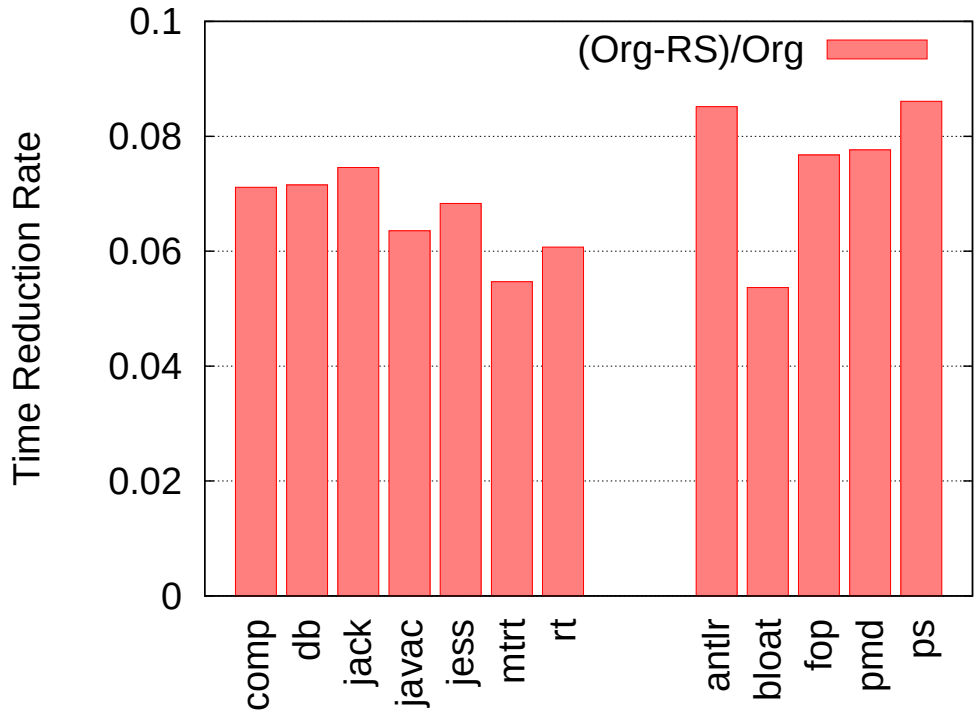
A compiler-based VM

GC Time Reduction: Semi-space



SableVM

Average saving 16.3%

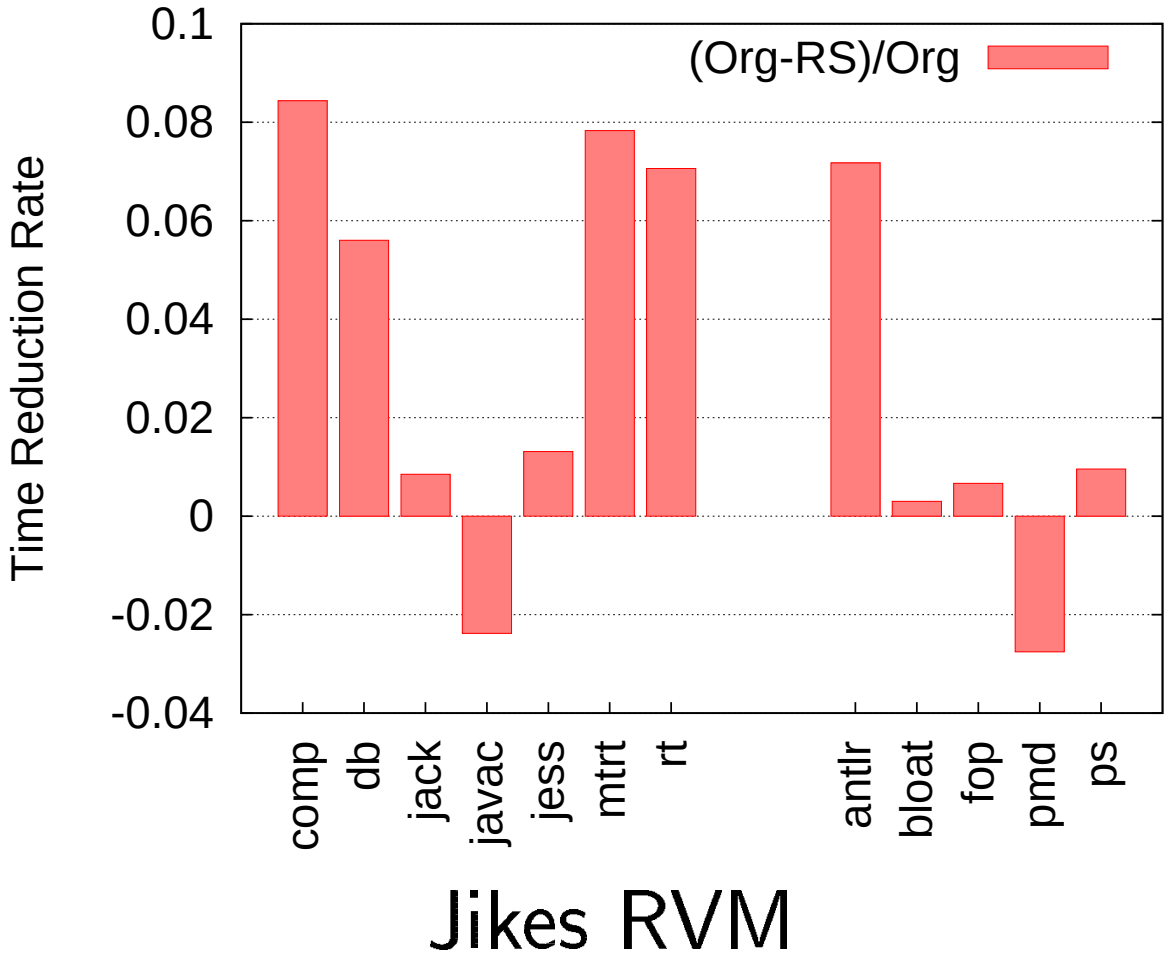


Jikes RVM

Average saving 7.1%

- Both JVMS obviously benefited on all benchmarks

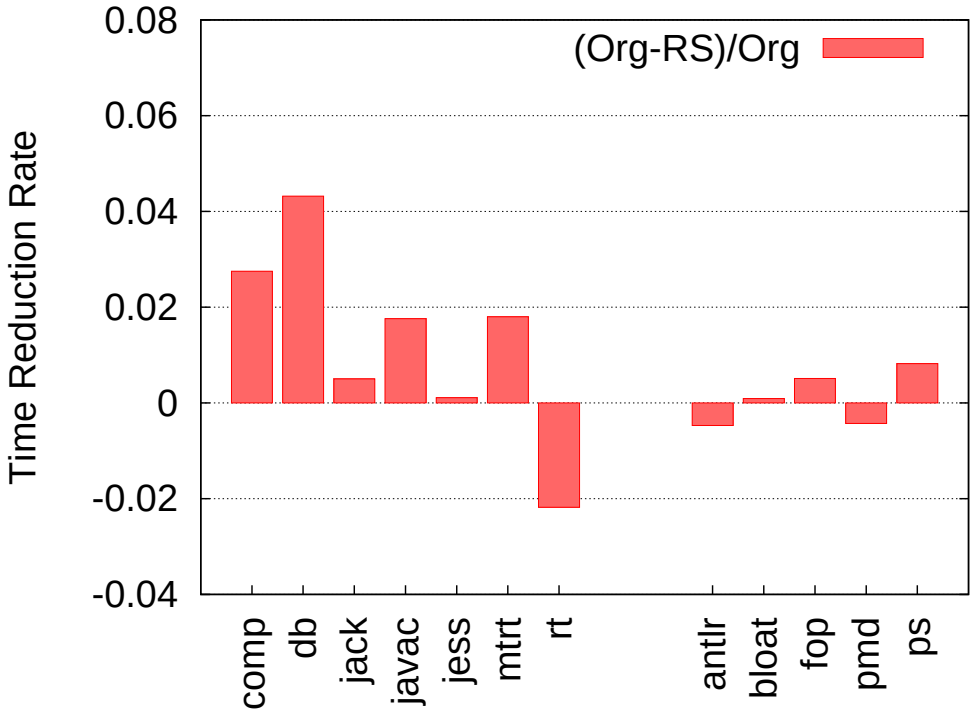
GC Time Reduction: GenMS



- **GenMS:** Generational-copying and mark-sweep
- **Less regular** results, some are **unexpected**

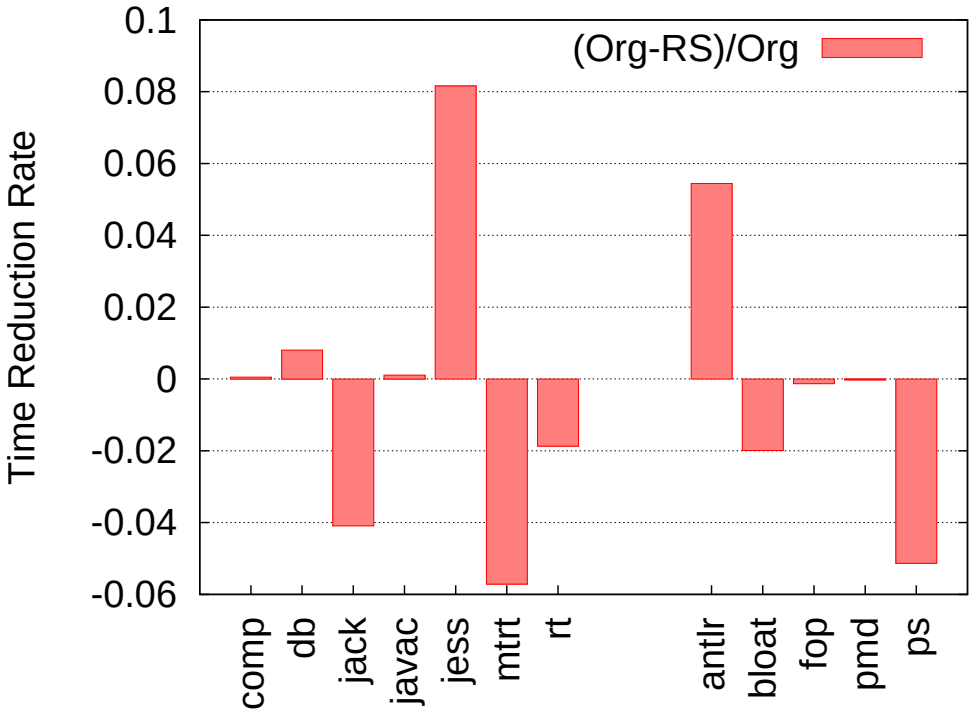
Mutator Time Reduction

Mutator Time = Whole Execution Time - GC Time



SableVM

Mysterious result

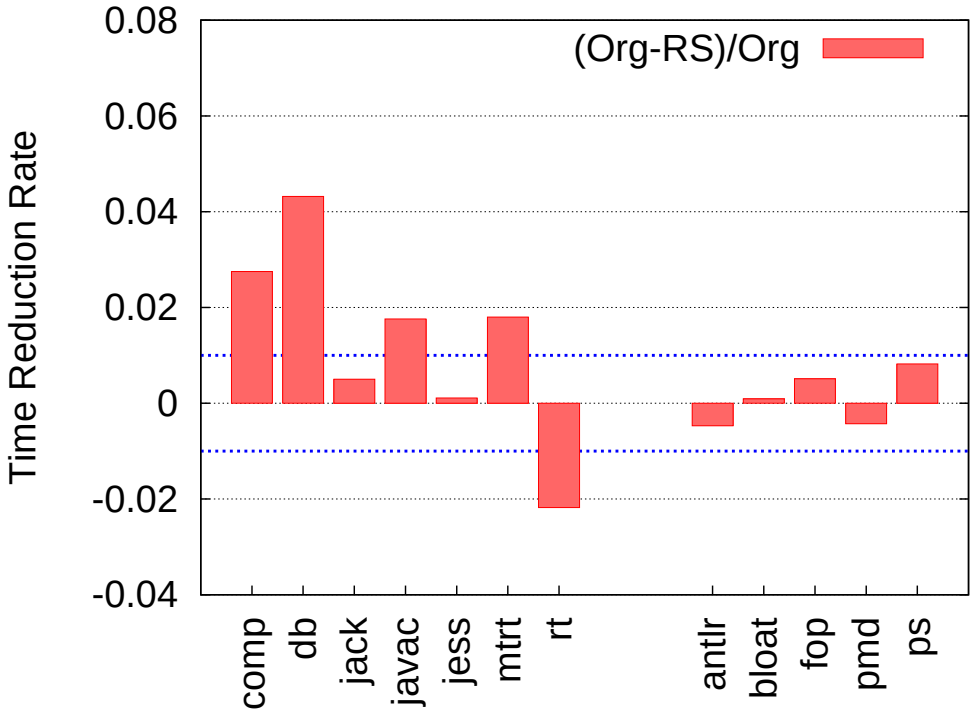


Jikes RVM

Quite **random** results

Mutator Time Reduction

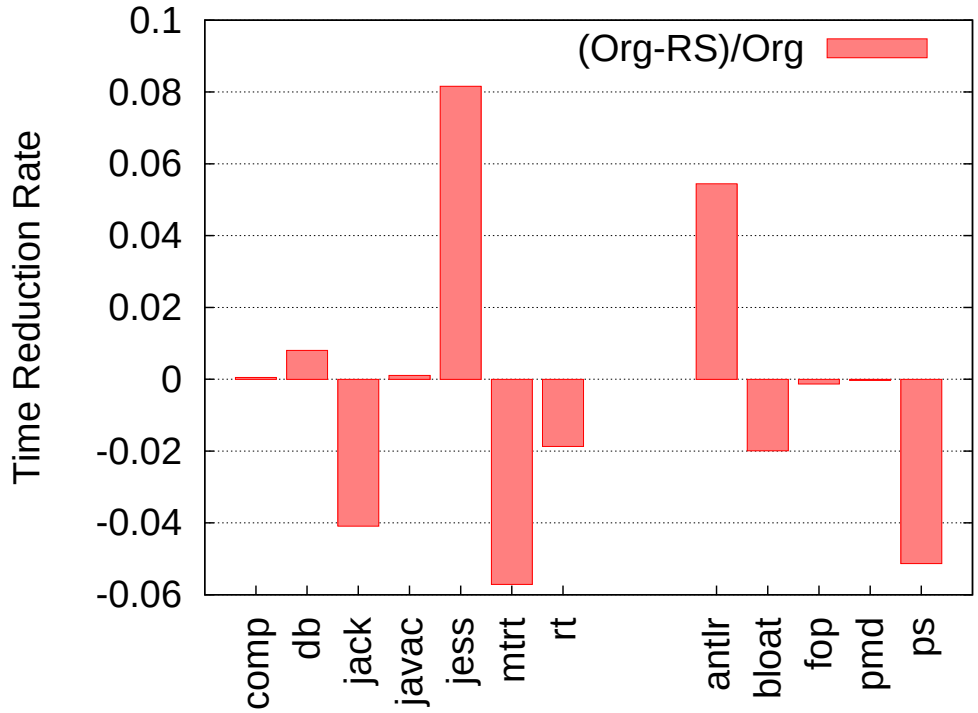
Mutator Time = Whole Execution Time - GC Time



SableVM

Mysterious result

Variation > GC time?!

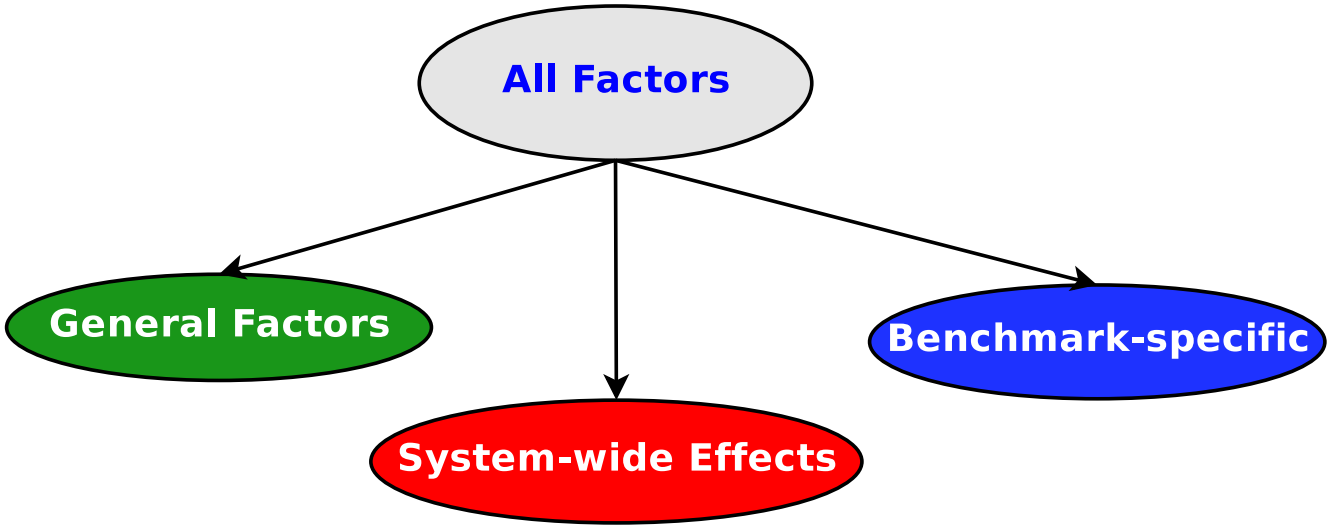


Jikes RVM

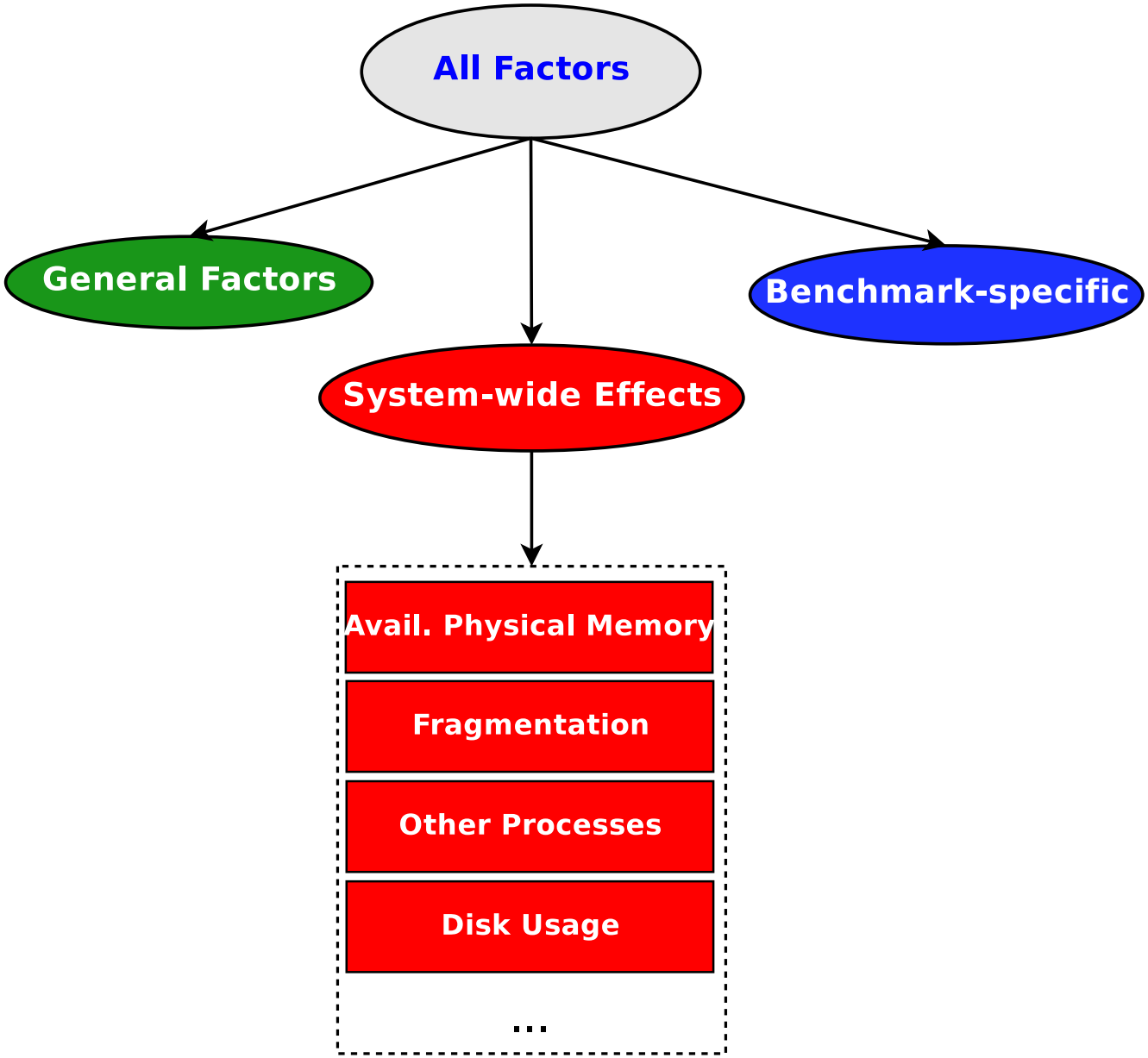
Quite **random** results

Outline

Relative Factors



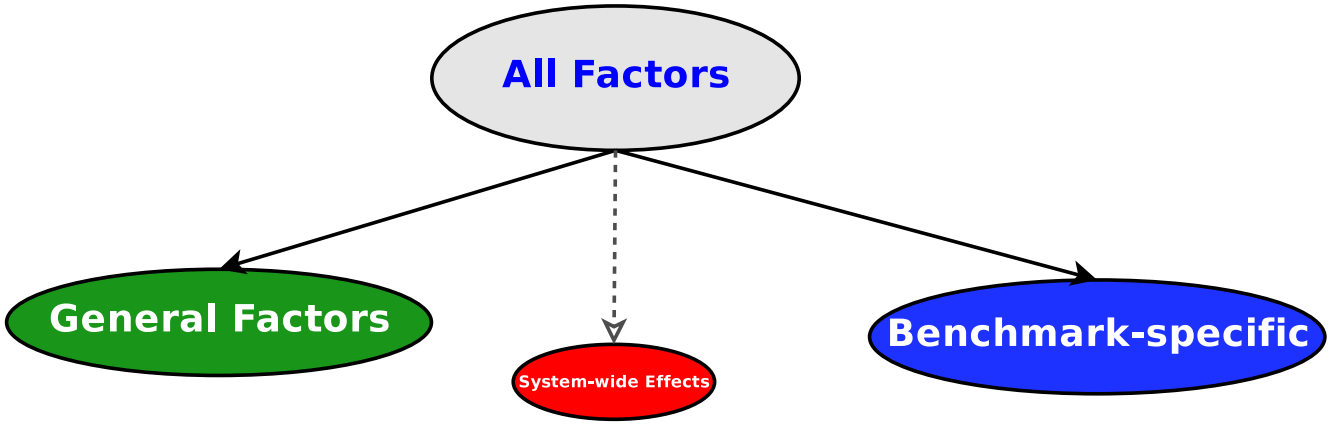
Relative Factors



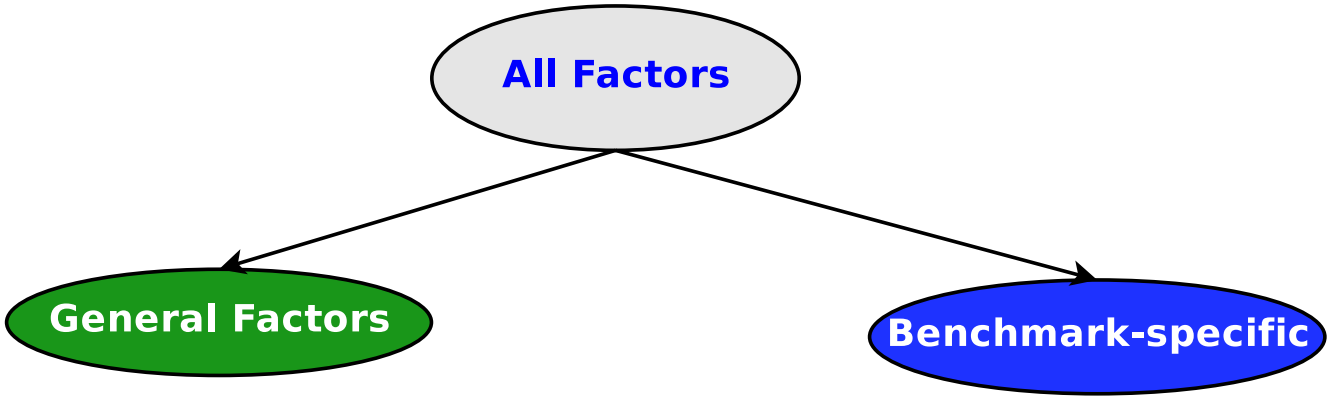
Reduce **System-wide Effect**

- Goal: Figure out the important reasons
- Reduce unnecessary noise
- Test on a **newly restarted, isolated, minimized workload** system, as most people do
- Make the **System-wide Effect** as small as possible

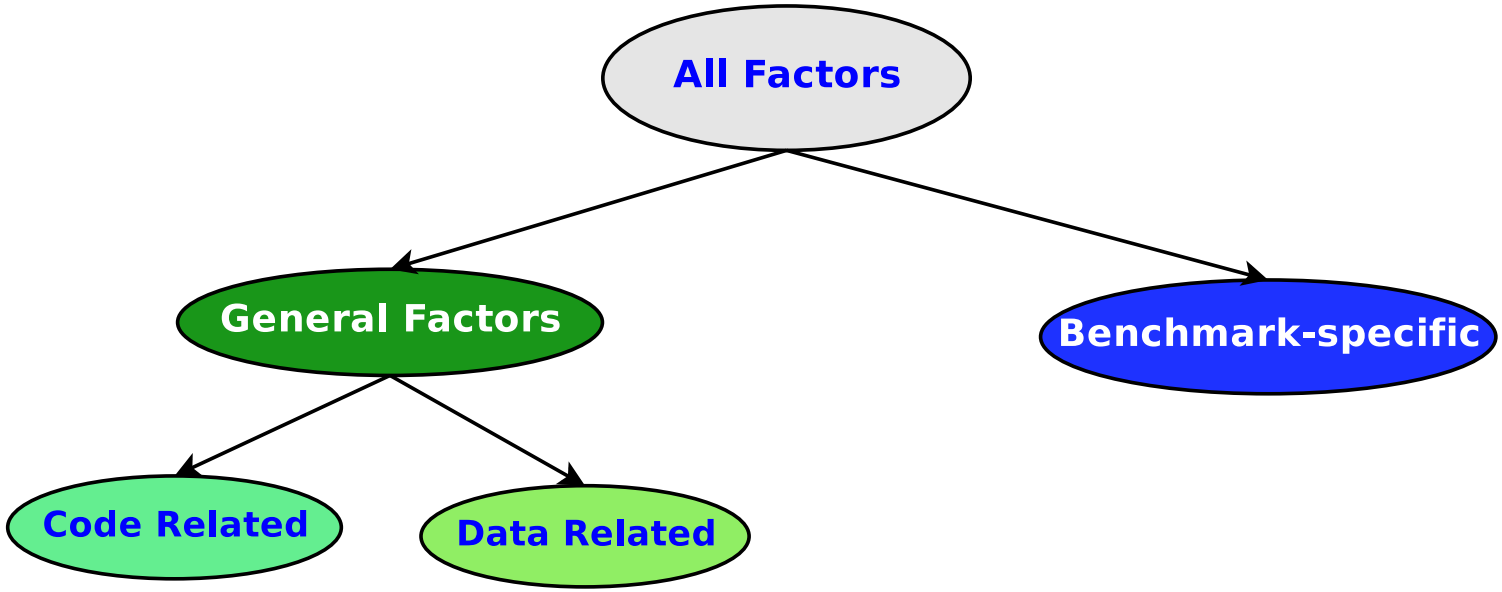
Make System-wide Effect Small



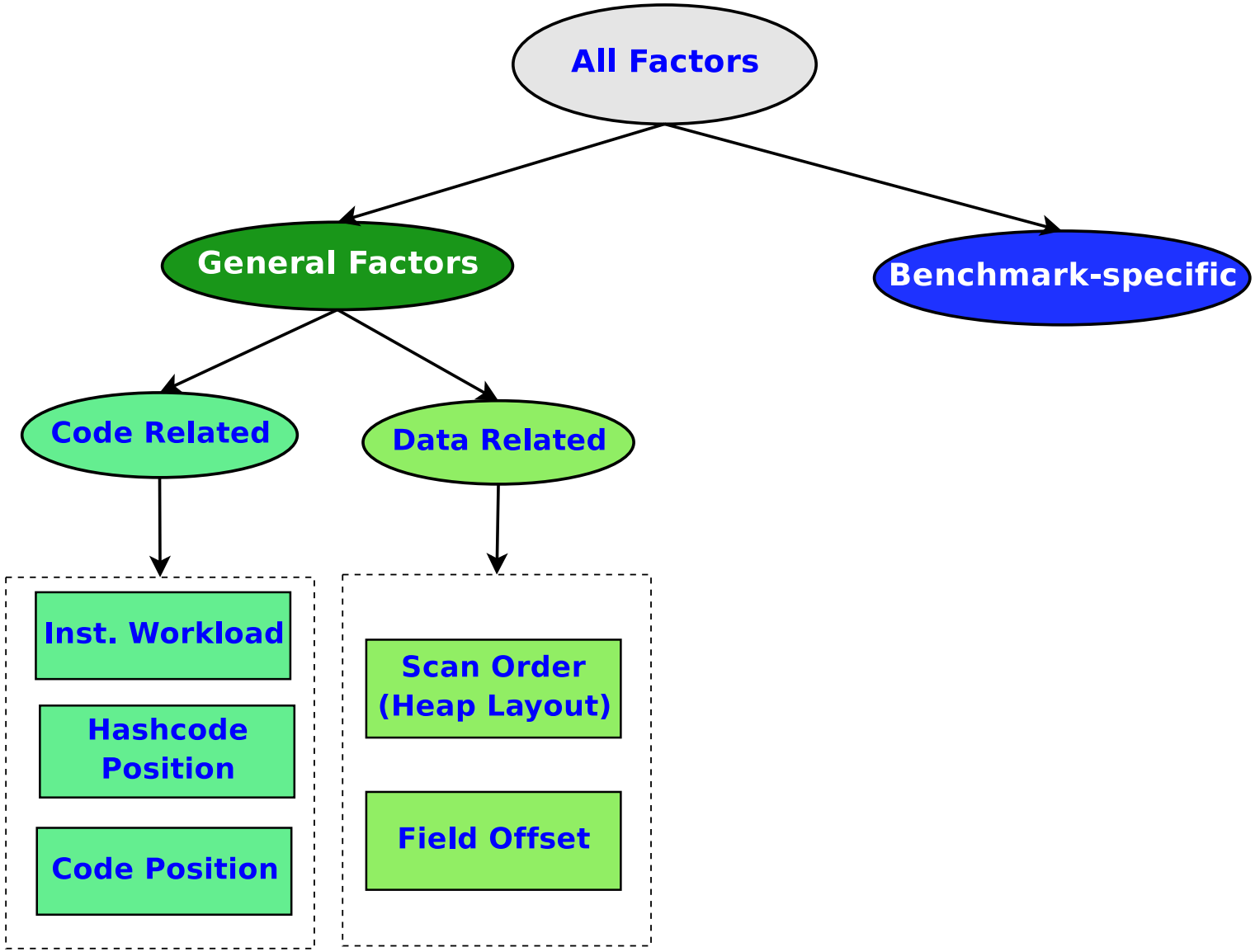
Relative Factors



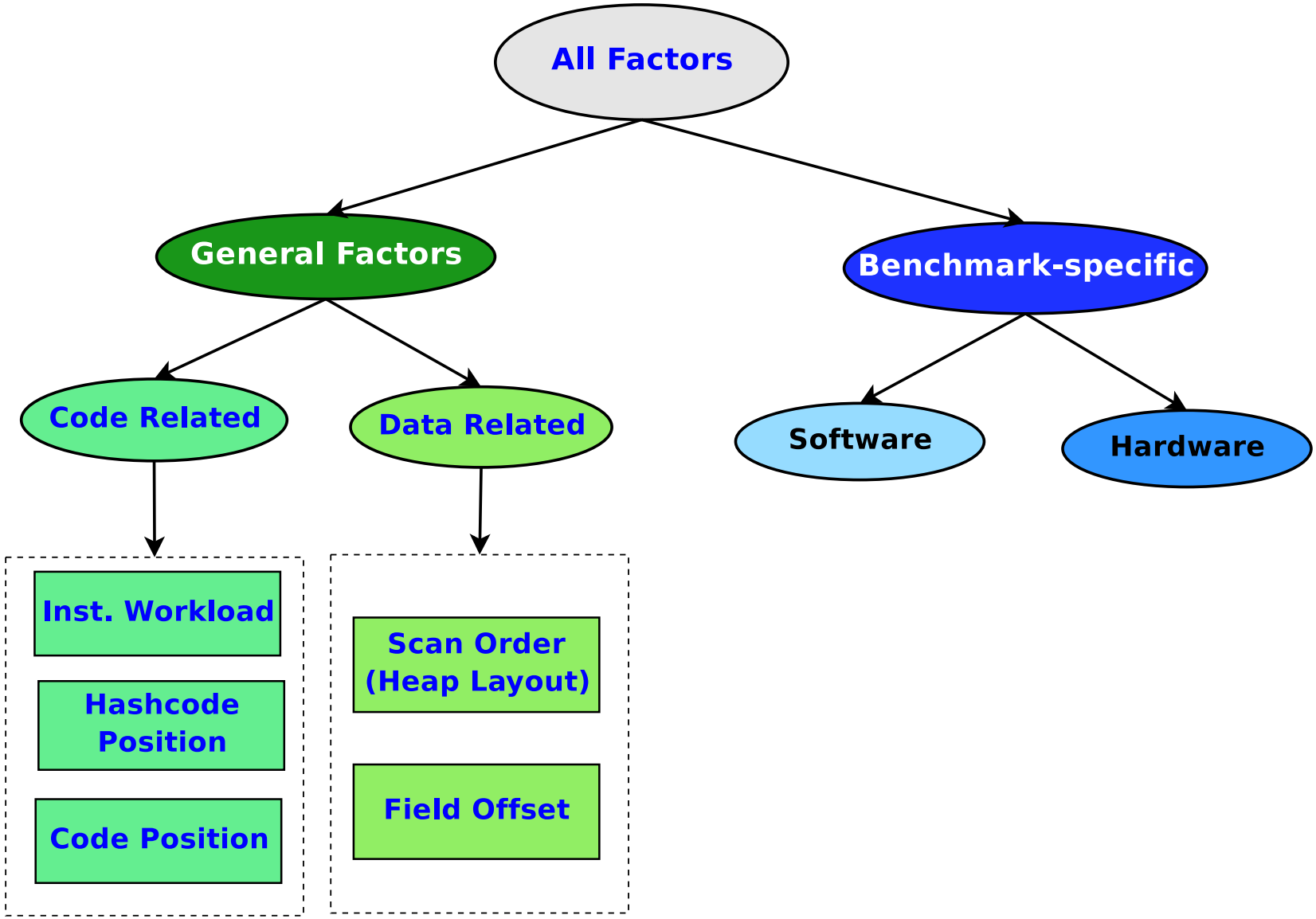
Relative Factors



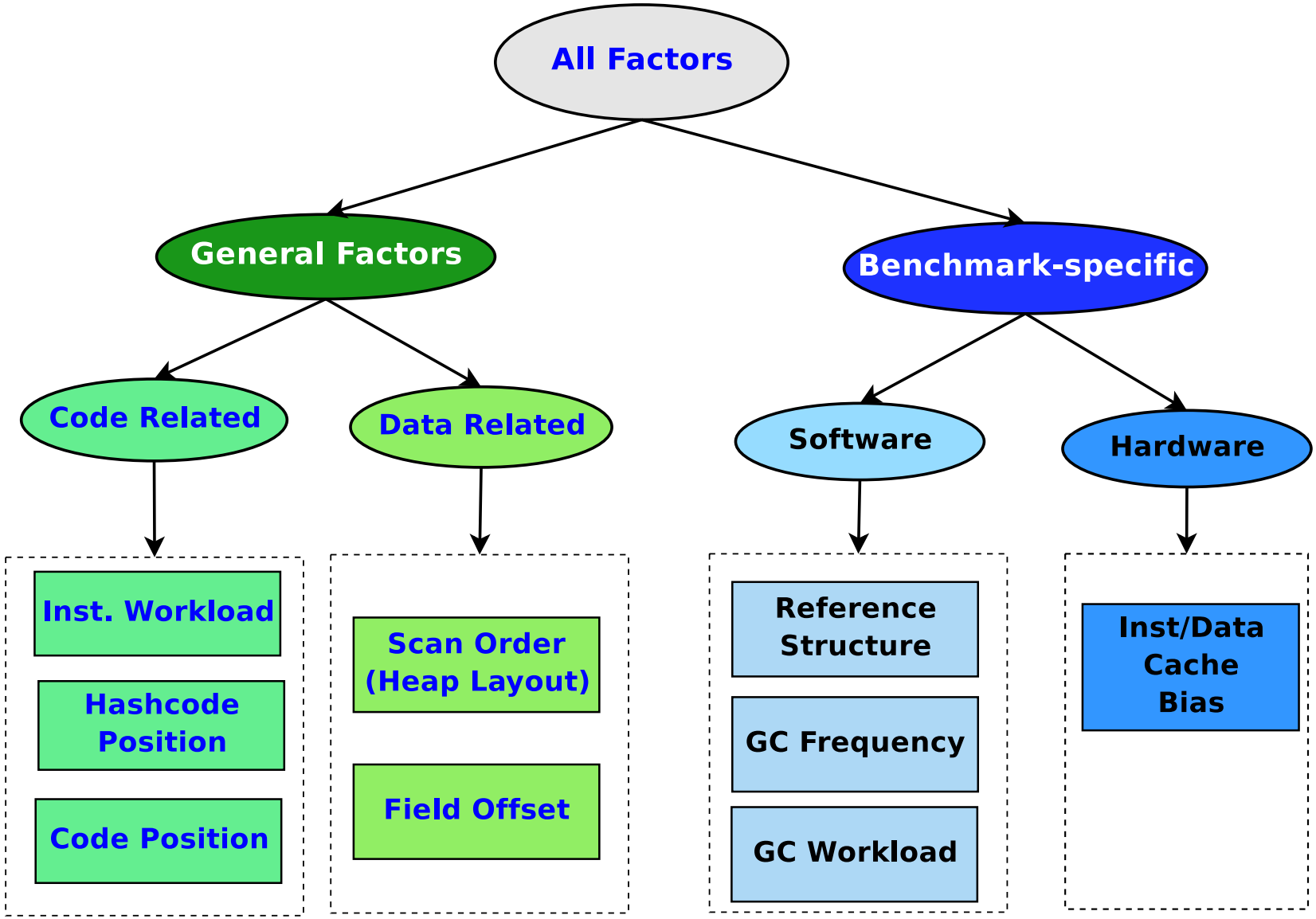
Relative Factors



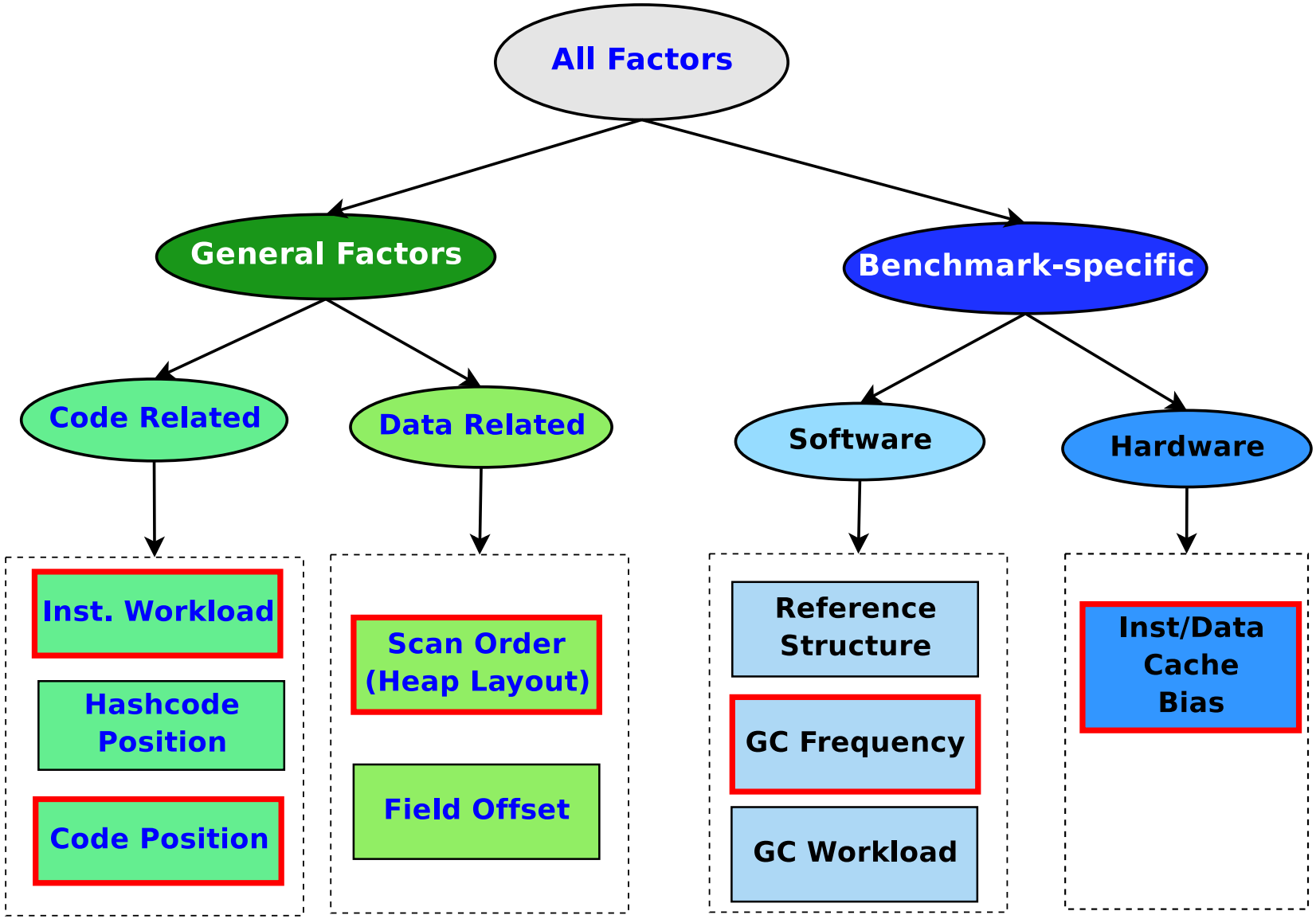
Relative Factors



Relative Factors



Relative Factors



Selected Factors

General

- Instruction
- Code Position
- Scan Order

Benchmark-specific

- GC Frequency
- Cache Bias

Outline

Factors

General

- Instruction
- Code Position
- Scan Order

Benchmark Spec.

- GC Frequency
- Cache Bias

Instruction Workload

- The number of machine instruction executed
- A fundamental factor of execution time
- Measured using hardware counters

Instruction Workload results

In GC

- Reduced up to 12 %
- $\Rightarrow$ GC speedup

In Mutator

- The variation is very small (*on average* 0.03%)
- **Instruction workload** did not cause the performance changes in mutator

Factors

General

- Instruction
- Code Position
- Scan Order

Benchmark Spec.

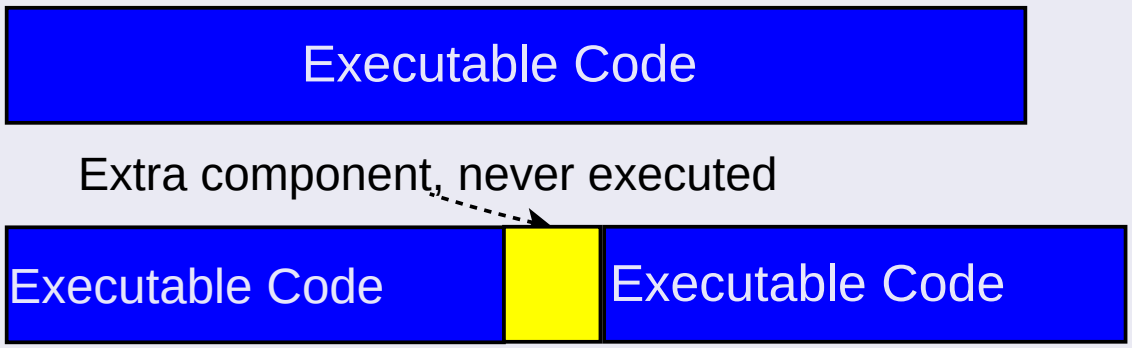
- GC Frequency
- Cache Bias

Code Position

- How code is arranged in memory
- Arrangement can affect cache performance
- How large can the impact be?
- Test by modifying the code position

Modify **Code Position**

Jikes RVM: Add an Extra Component, Never Executed

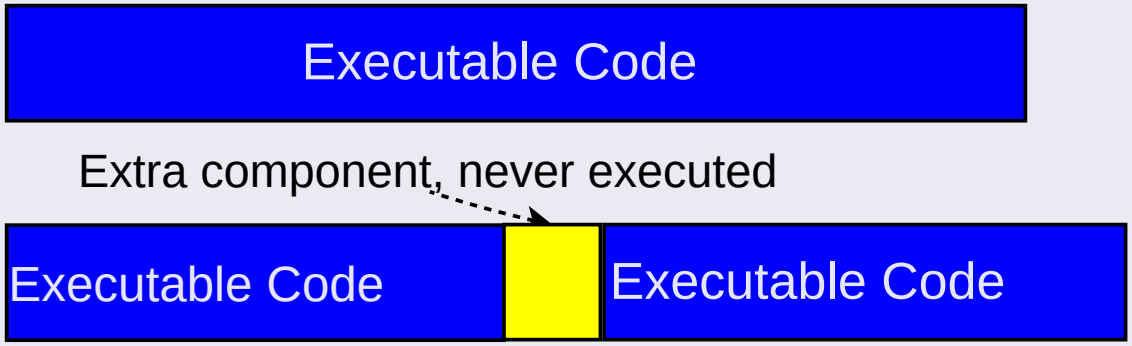


- Compare T_{org} vs T_{with_extra}
- Try different configurations

SableVM: Code Shifting

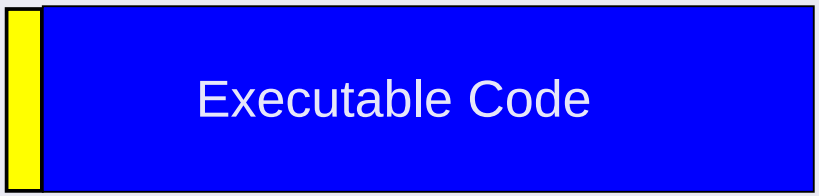
Modify **Code Position**

Jikes RVM: Add Extra Component, Never Executed



- Compare T_{org} and T_{with_extra}
- Try different configurations

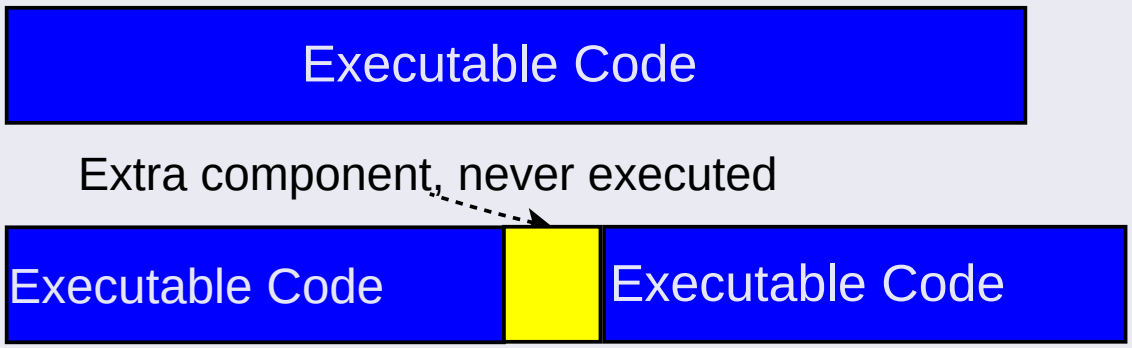
SableVM: Code Shifting



- Add empty space before the executable code
- Shift the code

Modify **Code Position**

Jikes RVM: Add Extra Component, Never Executed



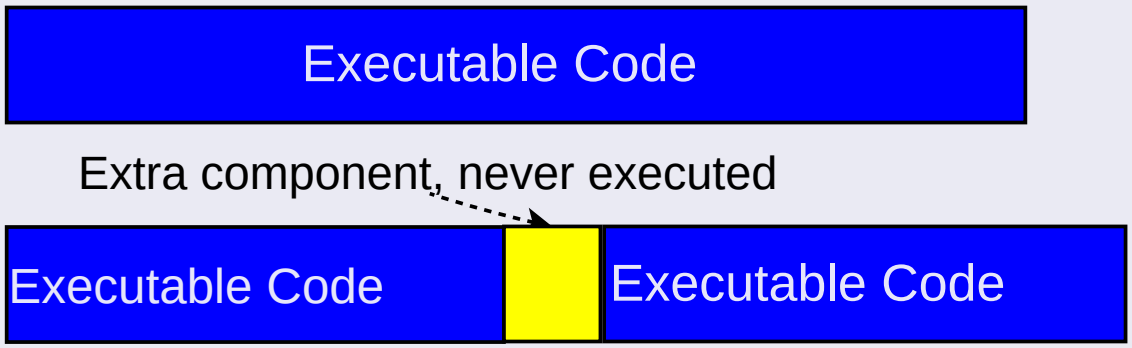
- Compare T_{org} and T_{with_extra}
- Try different configurations

SableVM: Code Shifting

- Increase the space by 4 bytes in each step

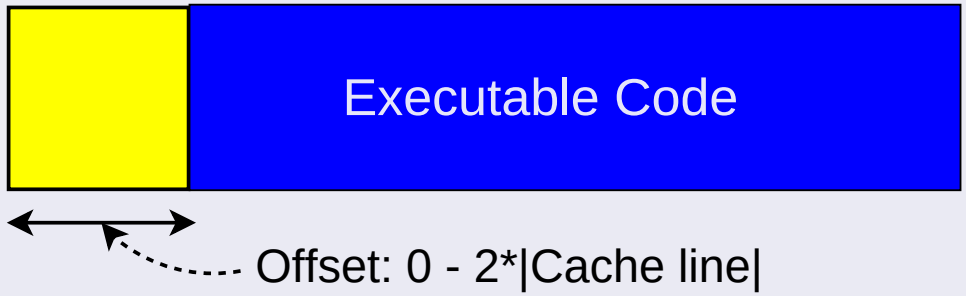
Modify **Code Position**

Jikes RVM: Add Extra Component, Never Executed



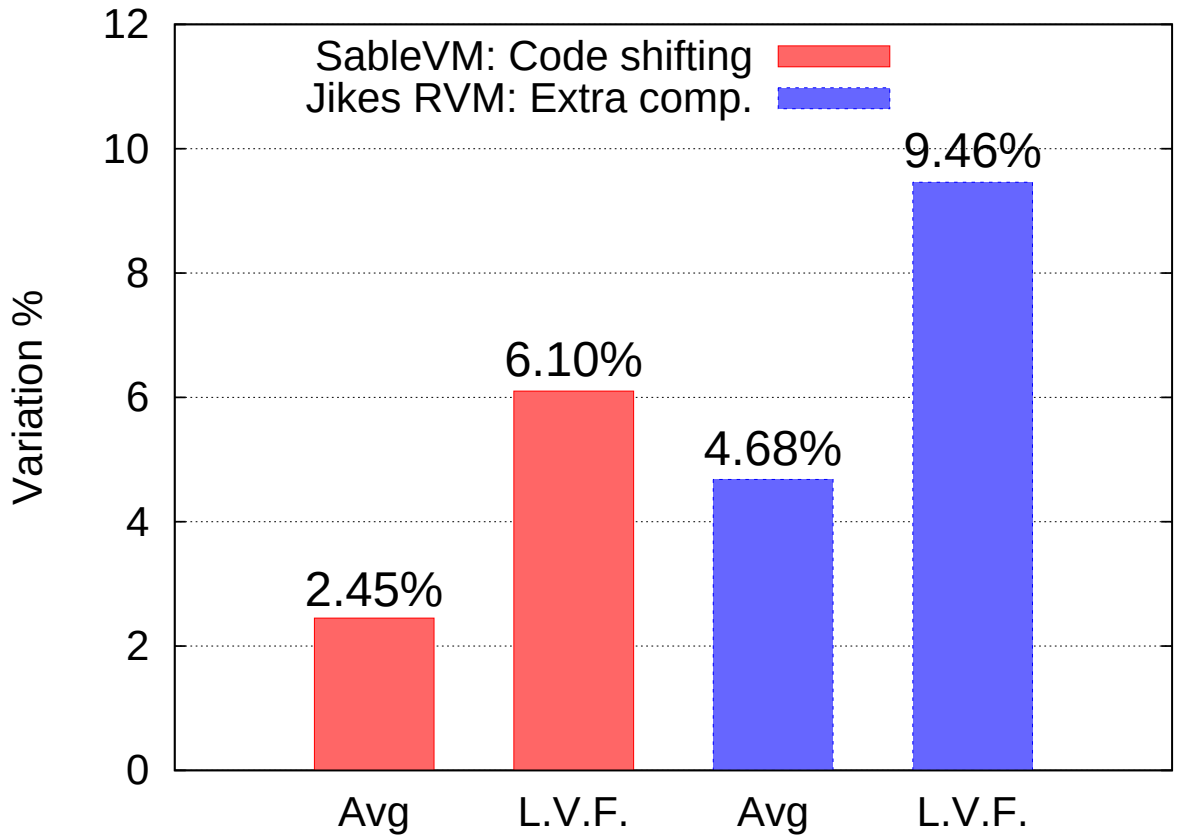
- Compare T_{org} and T_{with_extra}
- Try different configurations

SableVM: **Code Shifting**



- Obtain a set of shifted versions

Code Position Results



- **L.V.F.** for **L**argest **V**ariation **F**ound
- As large as 9.46%
- Could be even larger

- The impact of **Code position** can be **nearly 10%**
- Unexpectedly significant

Factors

General

- Instruction
- Code Position
- Scan Order

Benchmark Spec.

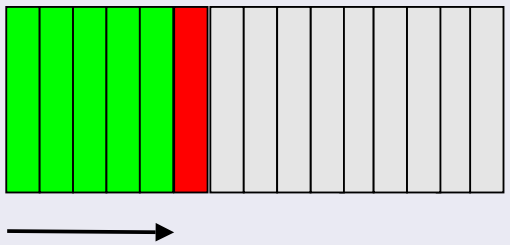
- GC Frequency
- Cache Bias

Scan Order

- Changing the scan order $\Rightarrow$ different heap layouts
- Impacts data cache performance
- We measured two scan orders

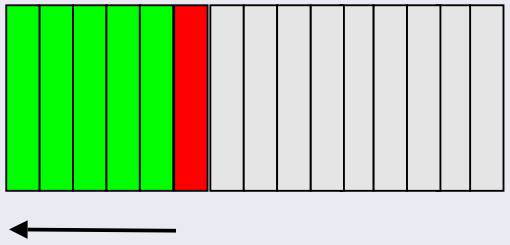
Two Scan Orders

Order 1



subclass → superclass

Order 2



superclass → subclass

Scan Order results

- Data cache performance changed
- No dominant winner

Scan Order results

Cyc/Miss	Mutator	GC
SPECjvm98	396	137
DaCapo	254	167
Average	337	150

- Low data cache misses density in mutator
- 10% in data cache miss $\not\Rightarrow$ 1% in whole execution time

- Data cache performance changed
- No dominant winner

Scan Order results

Cyc/Miss	Mutator	GC
SPECjvm98	396	137
DaCapo	254	167
Average	337	150

- Low data cache misses density in mutator
- 10% in data cache miss $\not\Rightarrow$ 1% in whole execution time

- Data cache performance changed
- No dominant winner
- Impact: **scan order** $<$ **code position**

Factors

General

- Instruction
- Code Position
- Scan Order

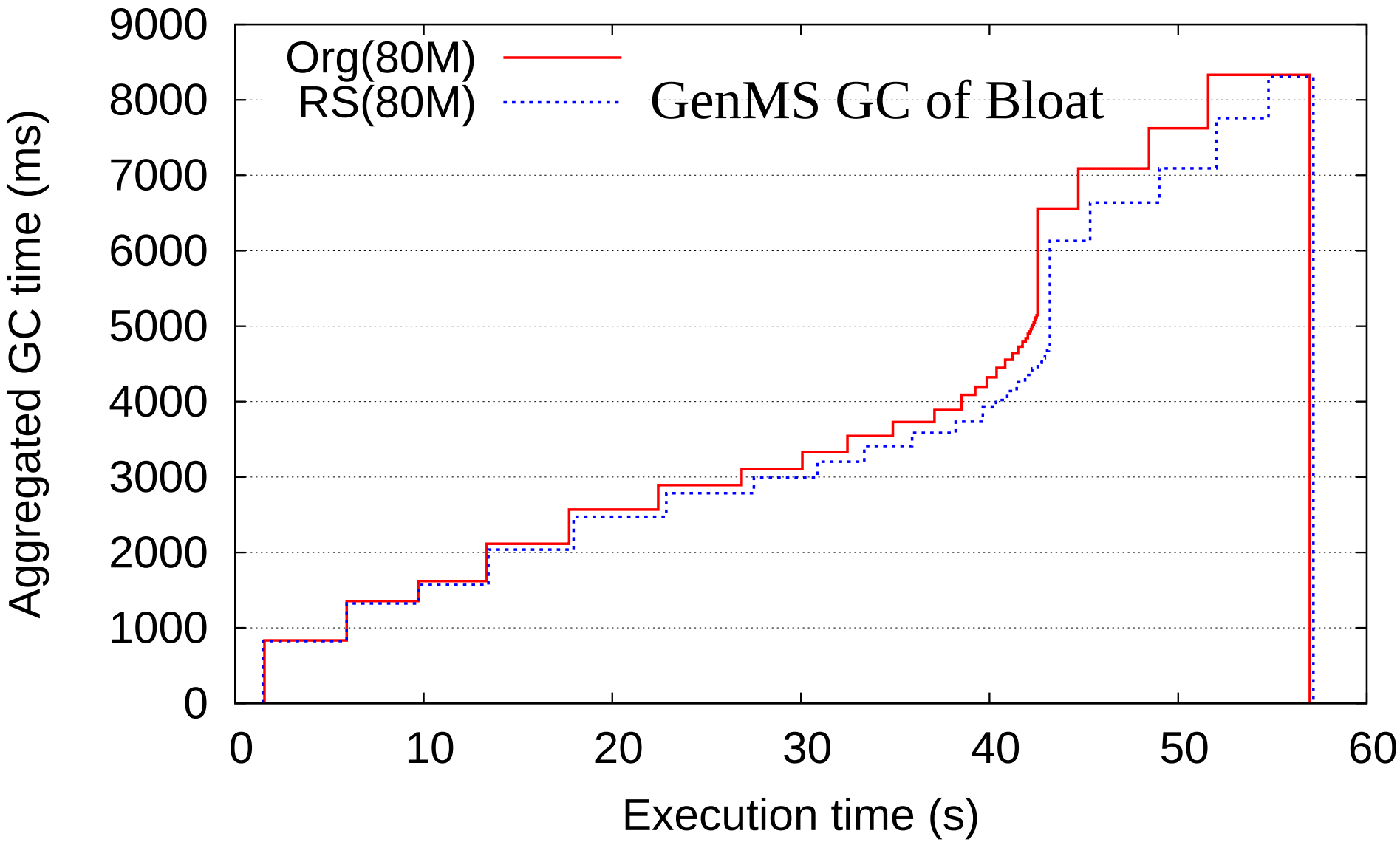
Benchmark Spec.

- GC Frequency
- Cache Bias

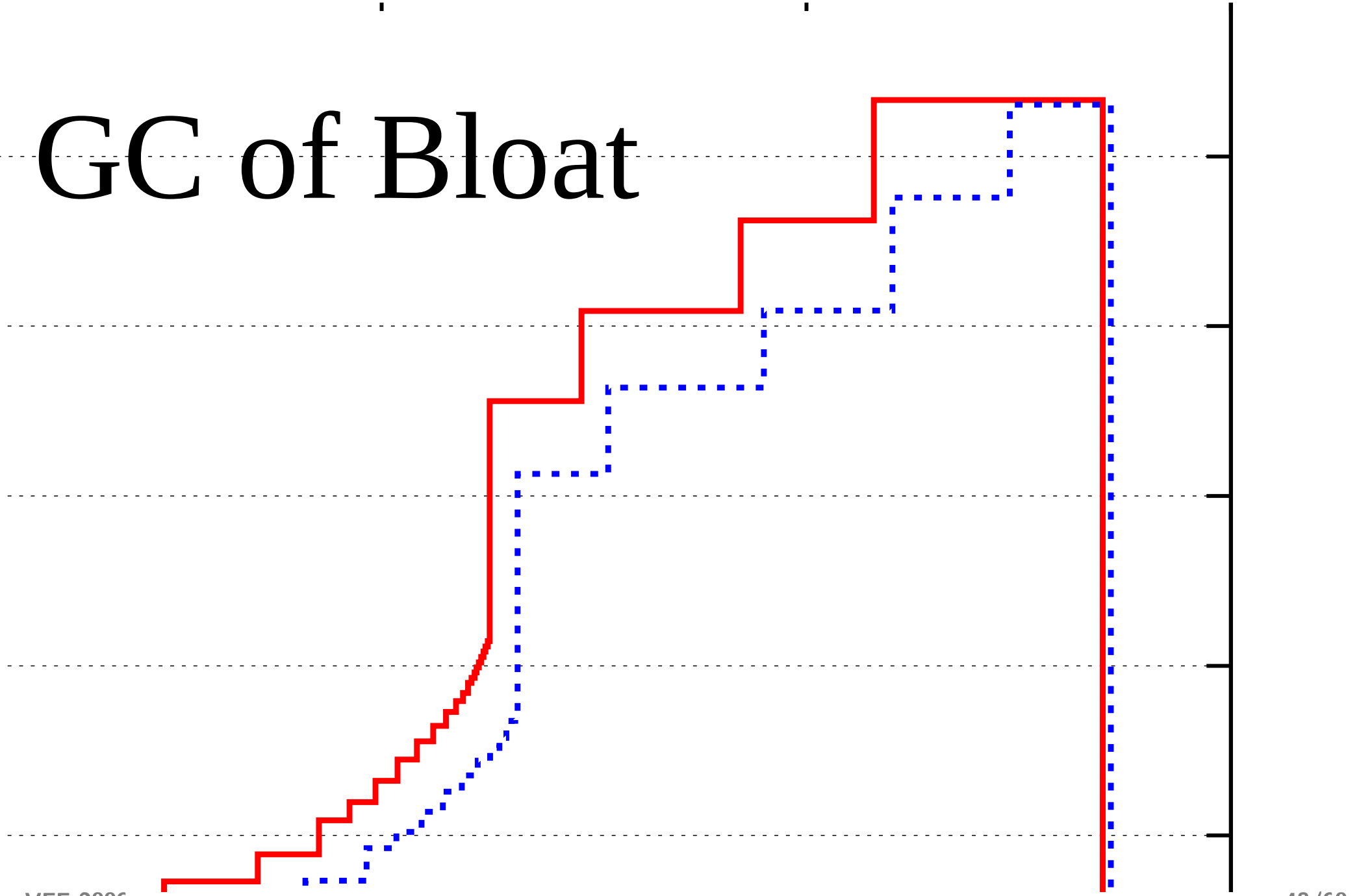
GC Frequency

- The number of GC cycles can be different
- **GenMS** GC results on benchmark BLOAT

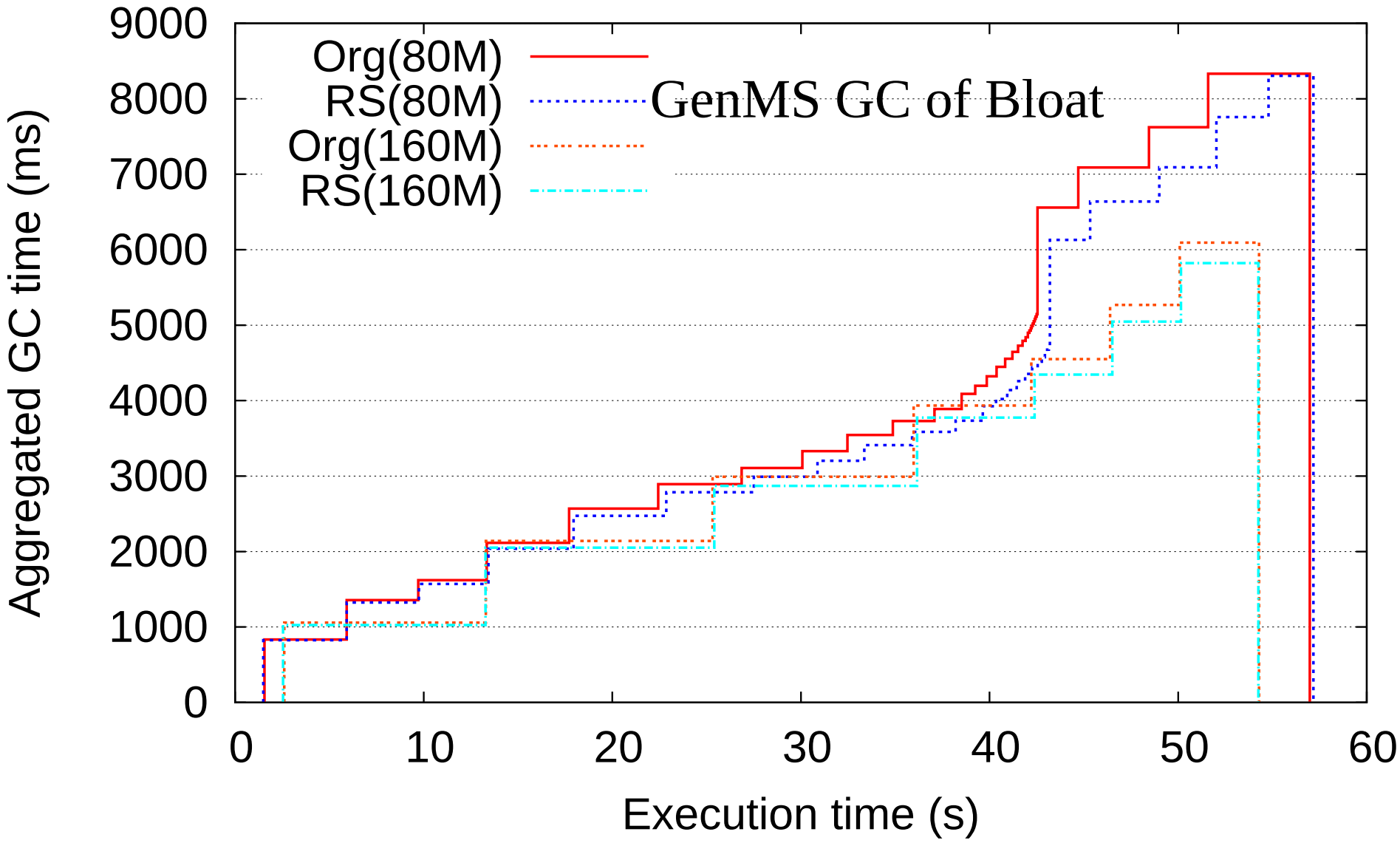
GC Frequency results



GC Frequency results



GC Frequency results



Factors

General

- Instruction
- Code Position
- Scan Order

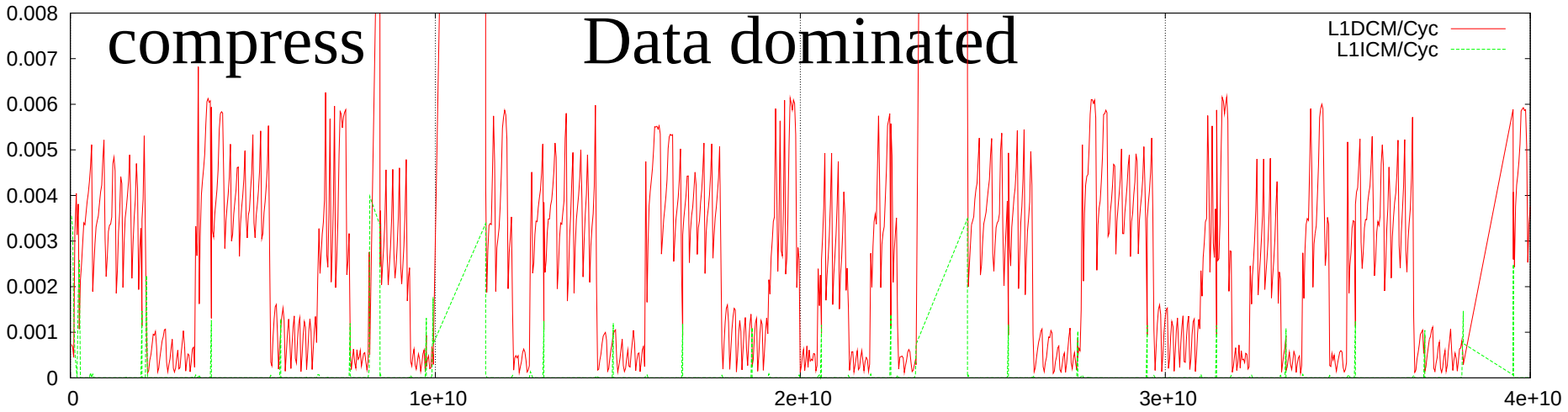
Benchmark Spec.

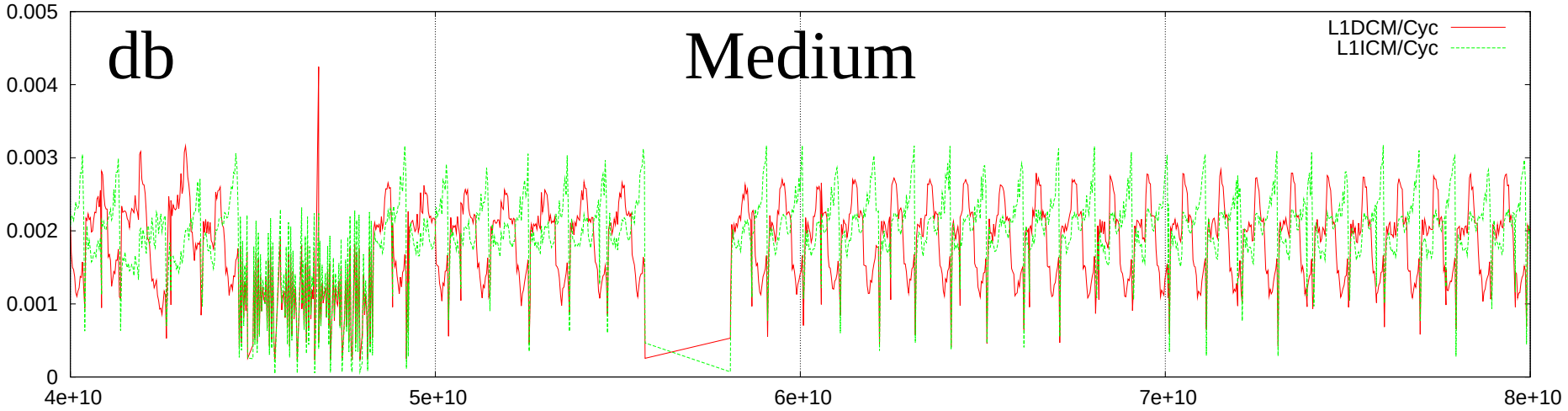
- GC Frequency
- Cache Bias

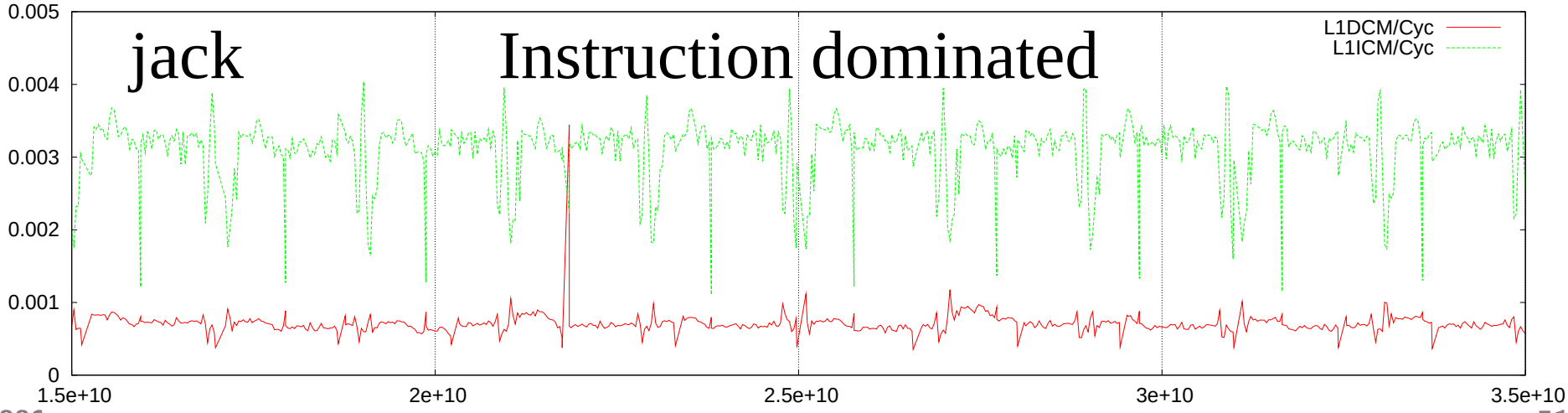
Benchmark Cache Bias

- More sensitive to the behavior of one type of cache than the other
- Cache performance graphs

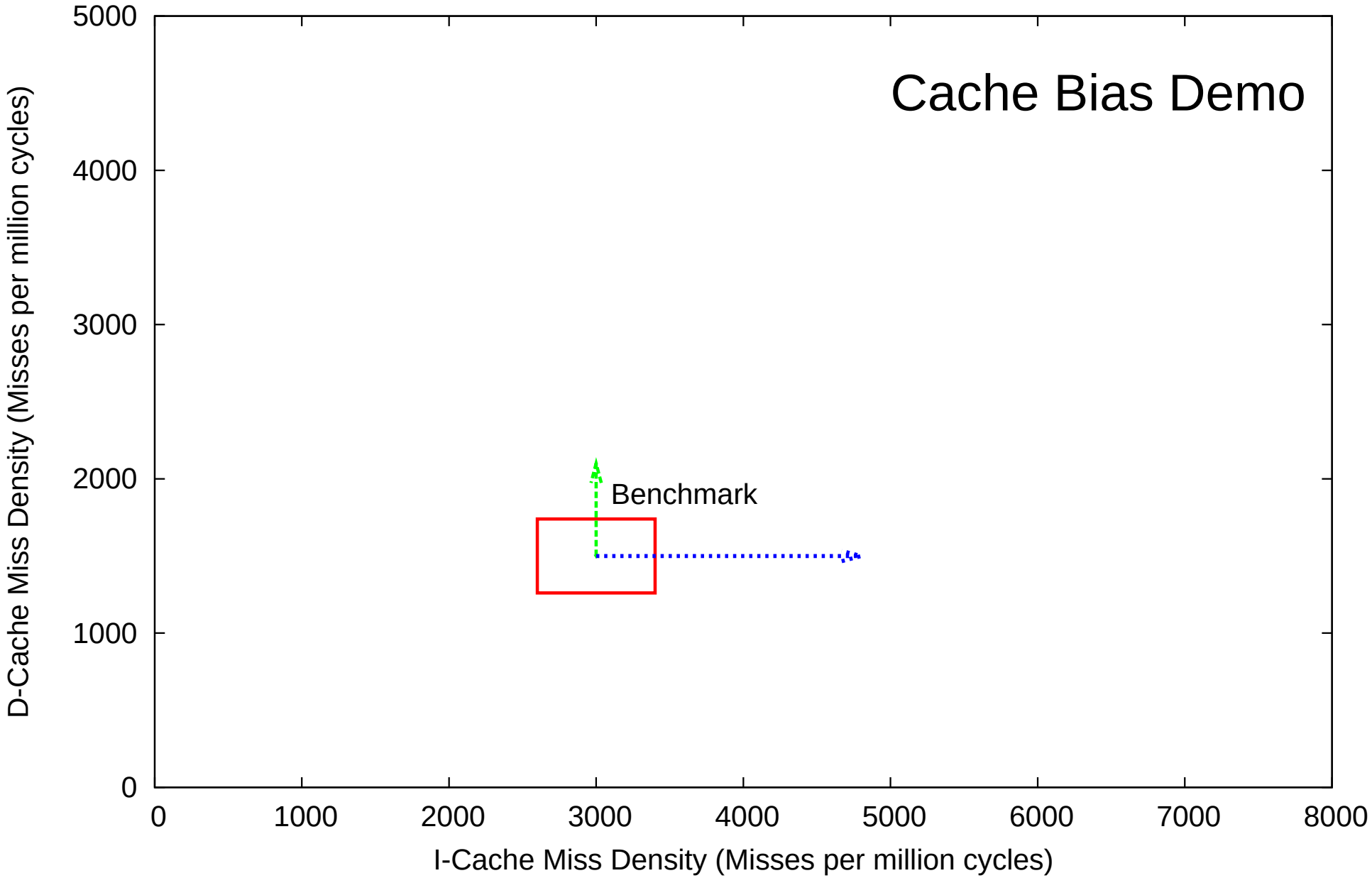
X-AXIS	Elapsed cycles
Y-AXIS	Cache misses density
Red curves	Data cache results
Green curves	Inst. cache results


compress
Data dominated
L1DCM/Cyc
L1ICM/Cyc
0.008
0.007
0.006
0.005
0.004
0.003
0.002
0.001
0
0
1e+10
2e+10
3e+10
4e+10

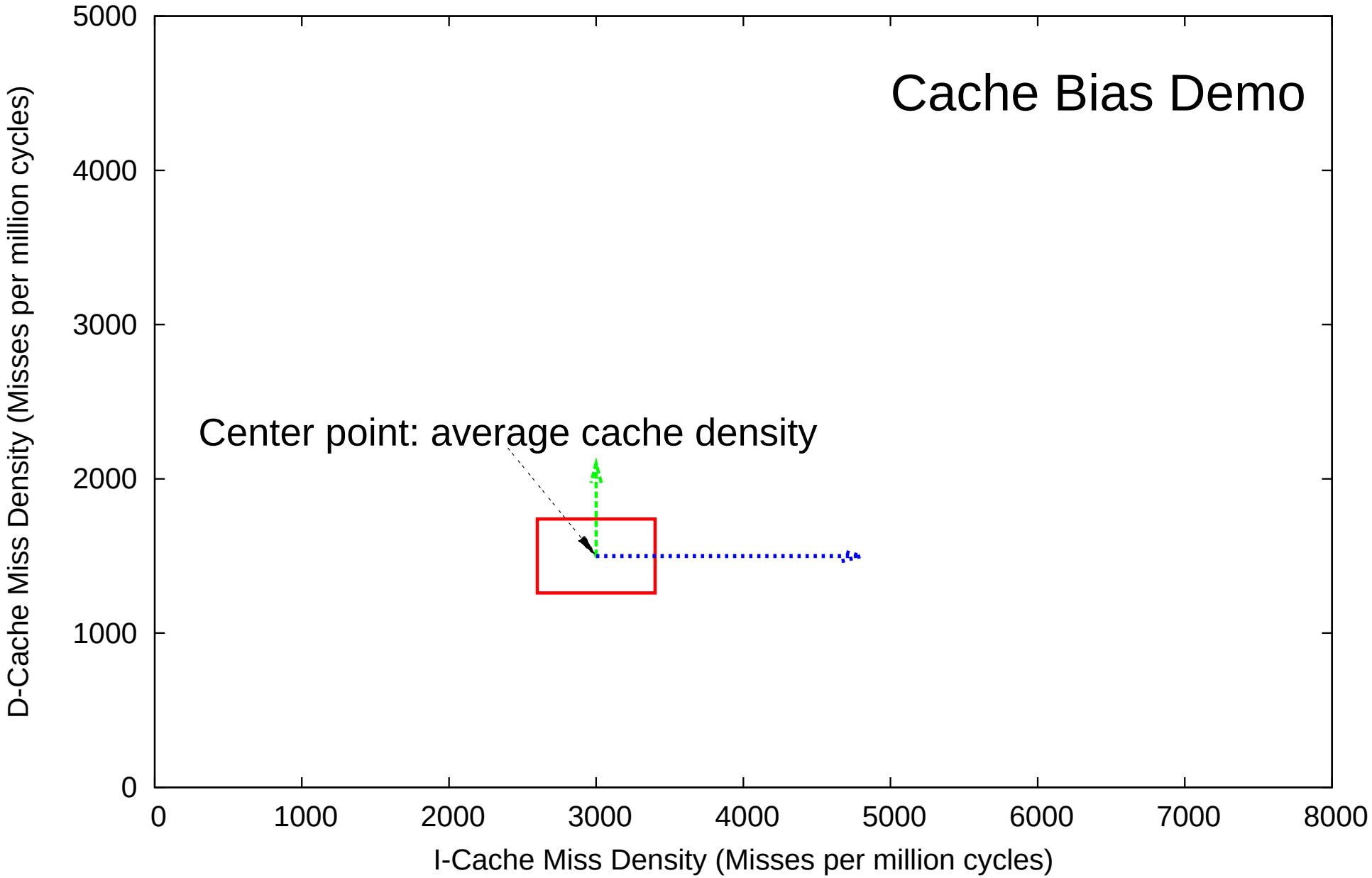

db
Medium
L1DCM/Cyc
L1ICM/Cyc
0.005
0.004
0.003
0.002
0.001
0
4e+10
5e+10
6e+10
7e+10
8e+10


jack
Instruction dominated
L1DCM/Cyc
L1ICM/Cyc
0.005
0.004
0.003
0.002
0.001
0
1.5e+10
2e+10
2.5e+10
3e+10
3.5e+10

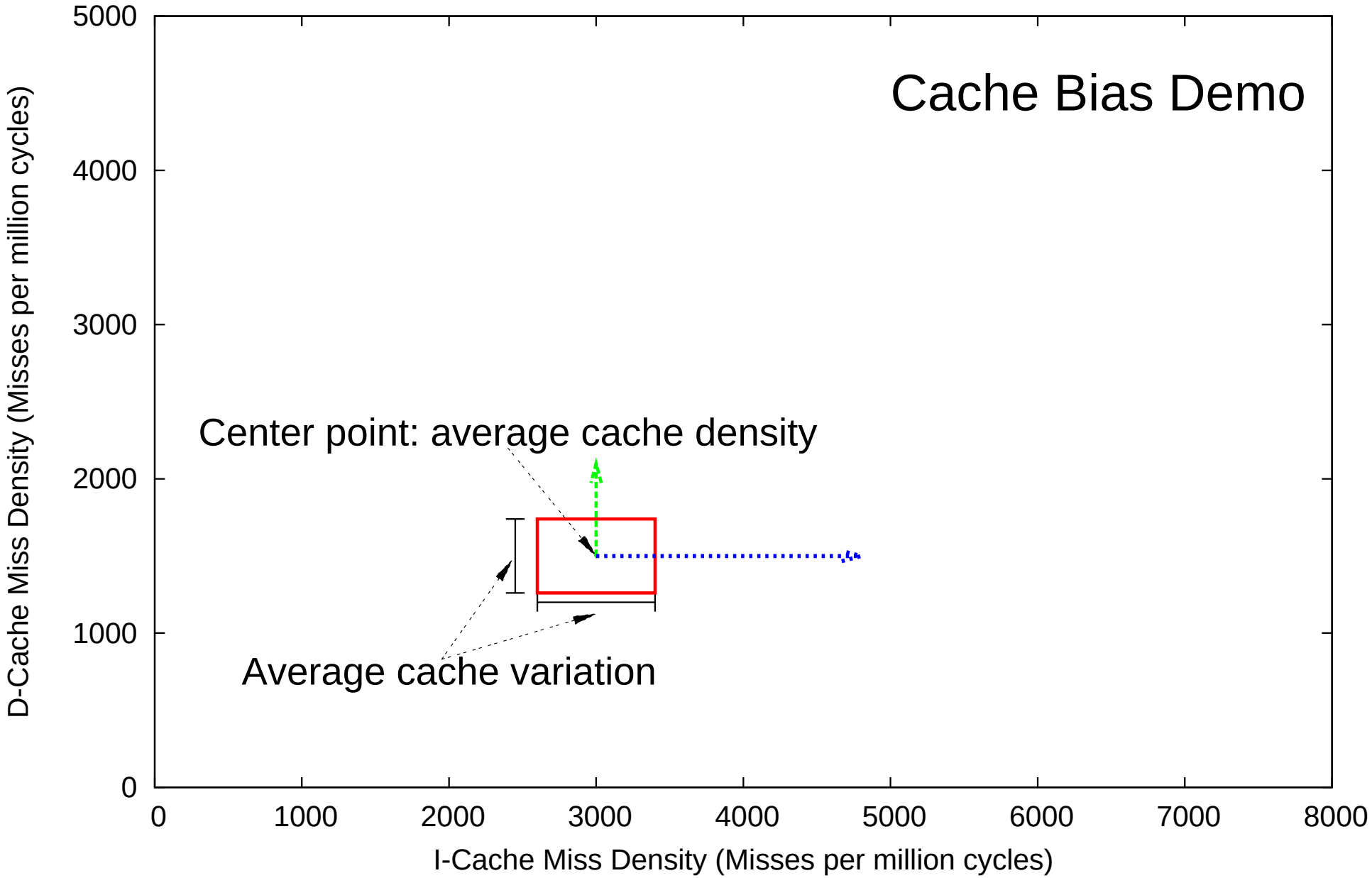
Cache Bias Graph Demo



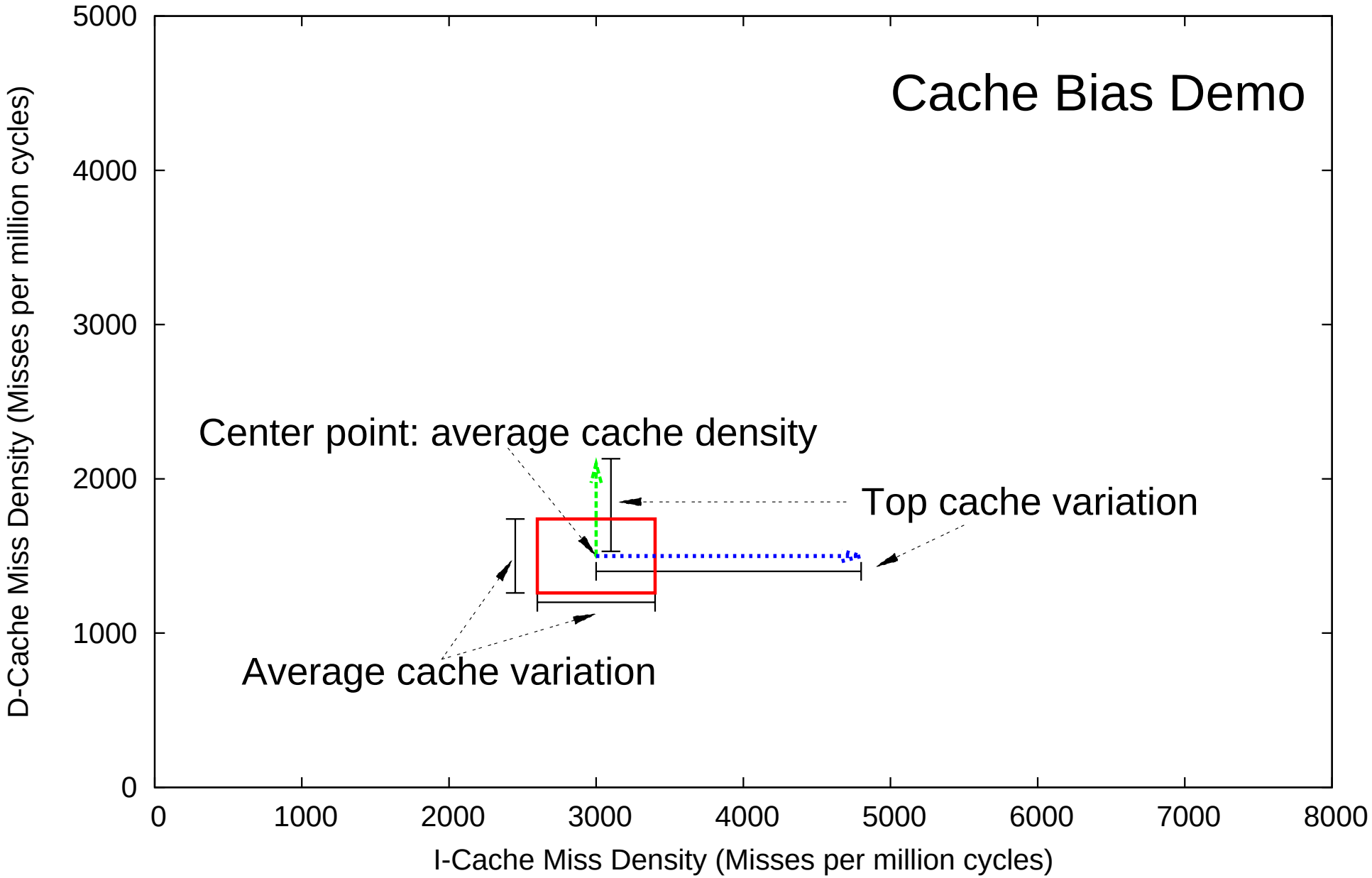
Cache Bias Graph Demo



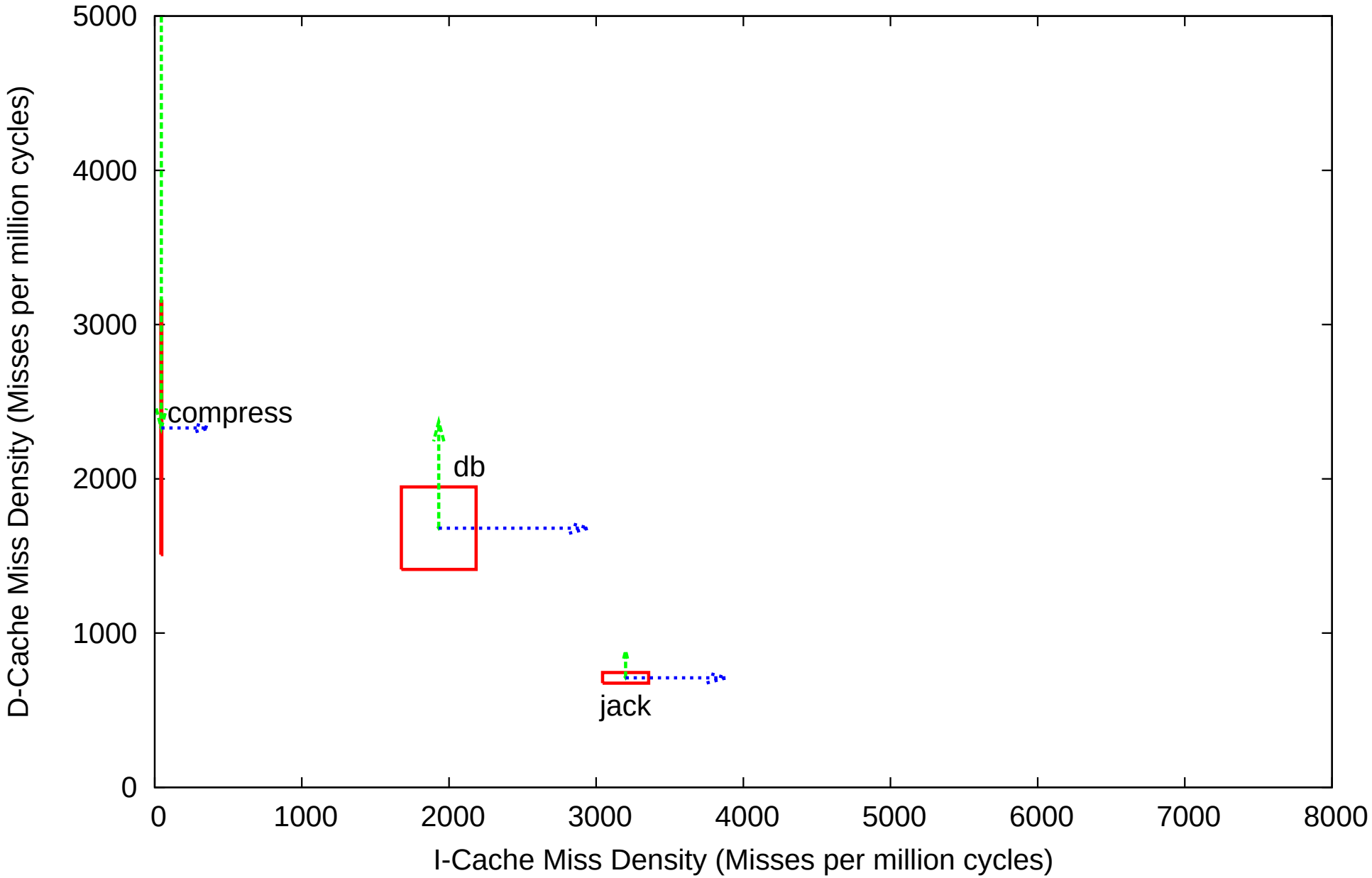
Cache Bias Graph Demo



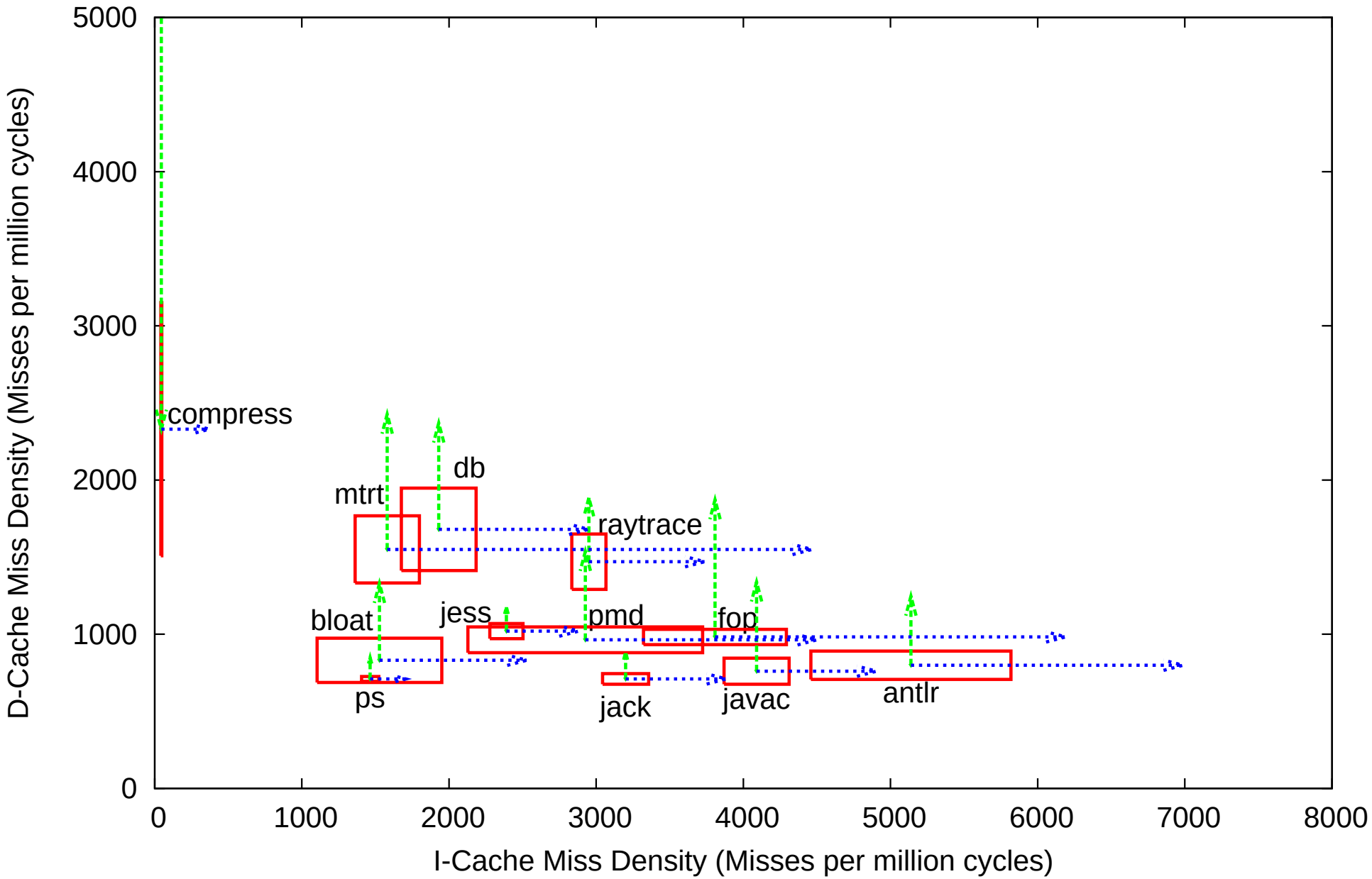
Cache Bias Graph Demo



Cache Bias


5000
4000
3000
2000
1000
0
D-Cache Miss Density (Misses per million cycles)
compress
db
jack
0
1000
2000
3000
4000
5000
6000
7000
8000
I-Cache Miss Density (Misses per million cycles)

Cache Bias


D-Cache Miss Density (Misses per million cycles)
I-Cache Miss Density (Misses per million cycles)
0
1000
2000
3000
4000
5000
6000
7000
8000
compress
mtrt
db
raytrace
bloat
jess
pmd
fop
ps
jack
javac
antlr

Conclusions

- Side-effects are significant enough to distort judgement on techniques
  - Measured influences of up to nearly 10%!
- It is necessary to do deep analysis on measurement results
- Many potential factors affect performance,
  - We estimated the importance of the factors
  - Case study: a GC optimization
- Present a general categorization of relative factors
  - Code/Data, Benchmark-specific, System-wide effects
  - Investigated relative impact of each factor

Future Work

- Code layout
  - Reduce code related "noise" $\Rightarrow$ more accurate performance measurement
  - Apply potential optimizations about code layout
- Further use of hardware data
  - Online and offline analysis on hardware data
  - Optimizations based on hardware data

Questions?

Thank you!