FACULTEIT INGENIEURSWETENSCHAPPEN

Using HPM-Sampling to Drive Dynamic Compilation

Dries Buytaert - Ghent University
Andy Georges - Ghent University
Michael Hind - IBM T.J. Watson
Matthew Arnold - IBM T.J. Watson
Lieven Eeckhout - Ghent University
Koen De Bosschere - Ghent University

OOPSLA 2007

To obtain high performance Java Virtual Machines have two modes of execution

Unoptimized execution

Optimized execution

To obtain high performance Java Virtual Machines have two modes of execution

Unoptimized execution

Optimized execution

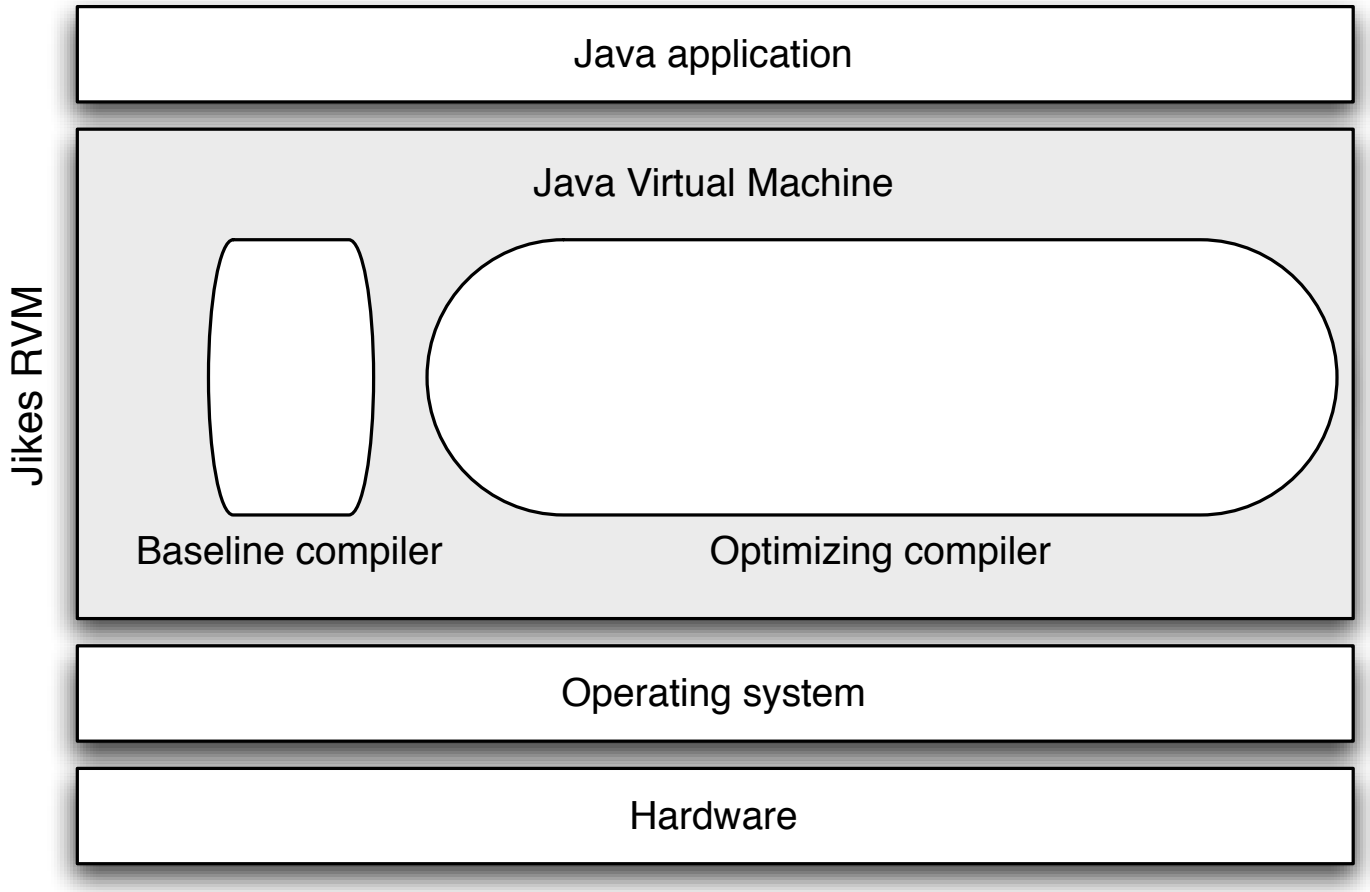
Unoptimized execution

	Interpreter	Quick compiler
Self-93		x
Sun Hotspot	x	
IBM DK	x	
IBM J9	x	
BEA JRocket		x
Jikes RVM		x
Intel ORP		x

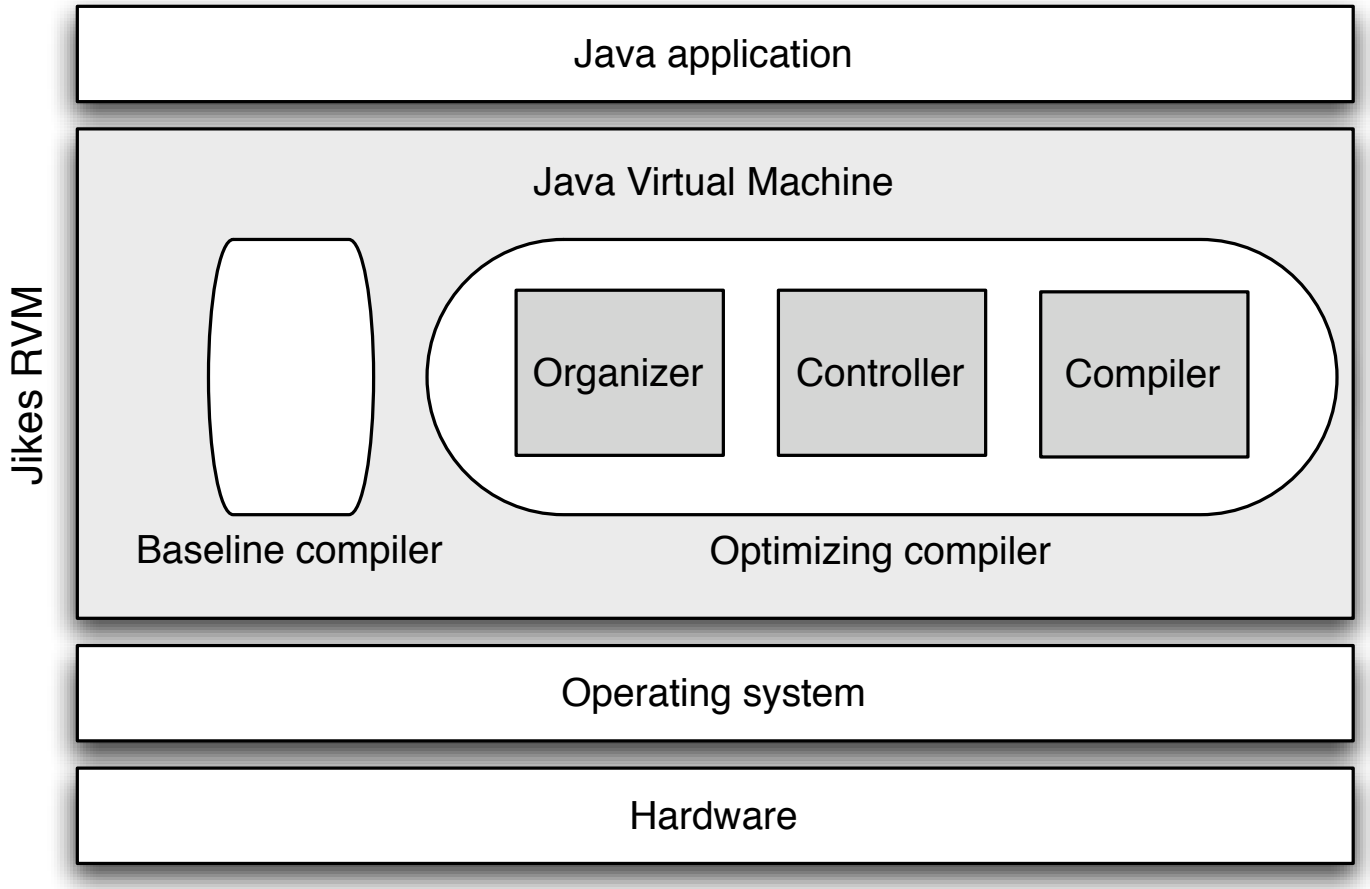
Unoptimized execution

VM	Interpreter	Quick compiler
Self-93		x
Sun Hotspot	x	
IBM DK	x	
IBM J9	x	
BEA JRockit		x
Jikes RVM		x
Intel ORP		x

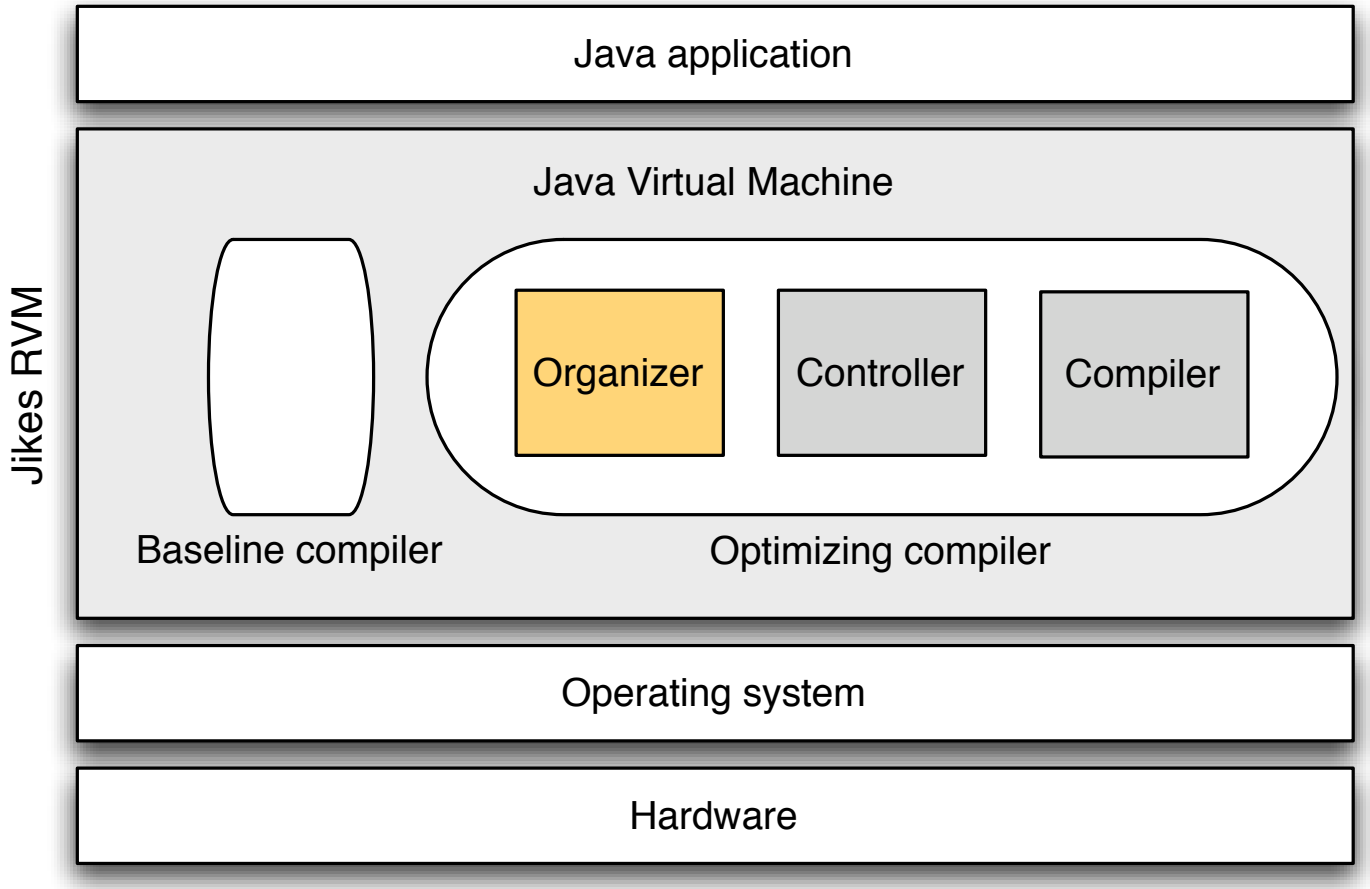
Jikes RVM


Java application
Java Virtual Machine
Jikes RVM
Baseline compiler
Optimizing compiler
Operating system
Hardware

The optimizing compiler has 3 sub-components



This work focuses mainly on the organizer



There are two approaches to find optimization candidates

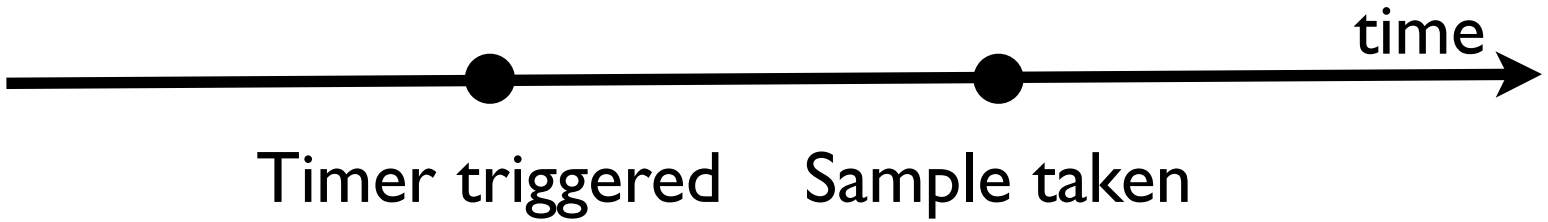
Counter-based sampling

Count the number of method invocations, and optionally loop iterations

Timer-based sampling

Sample the currently executing method at regular intervals using an operating system timer

Timer-based sampling



The size of the gap depends on
(i) how the timer is triggered and
(ii) how the sample is taken

The trigger mechanism initiates the sample-taking process

- *Nanosleep:* thread + while-loop + sleep
- *Setitimer*: interrupt handler
- Hardware performance monitors

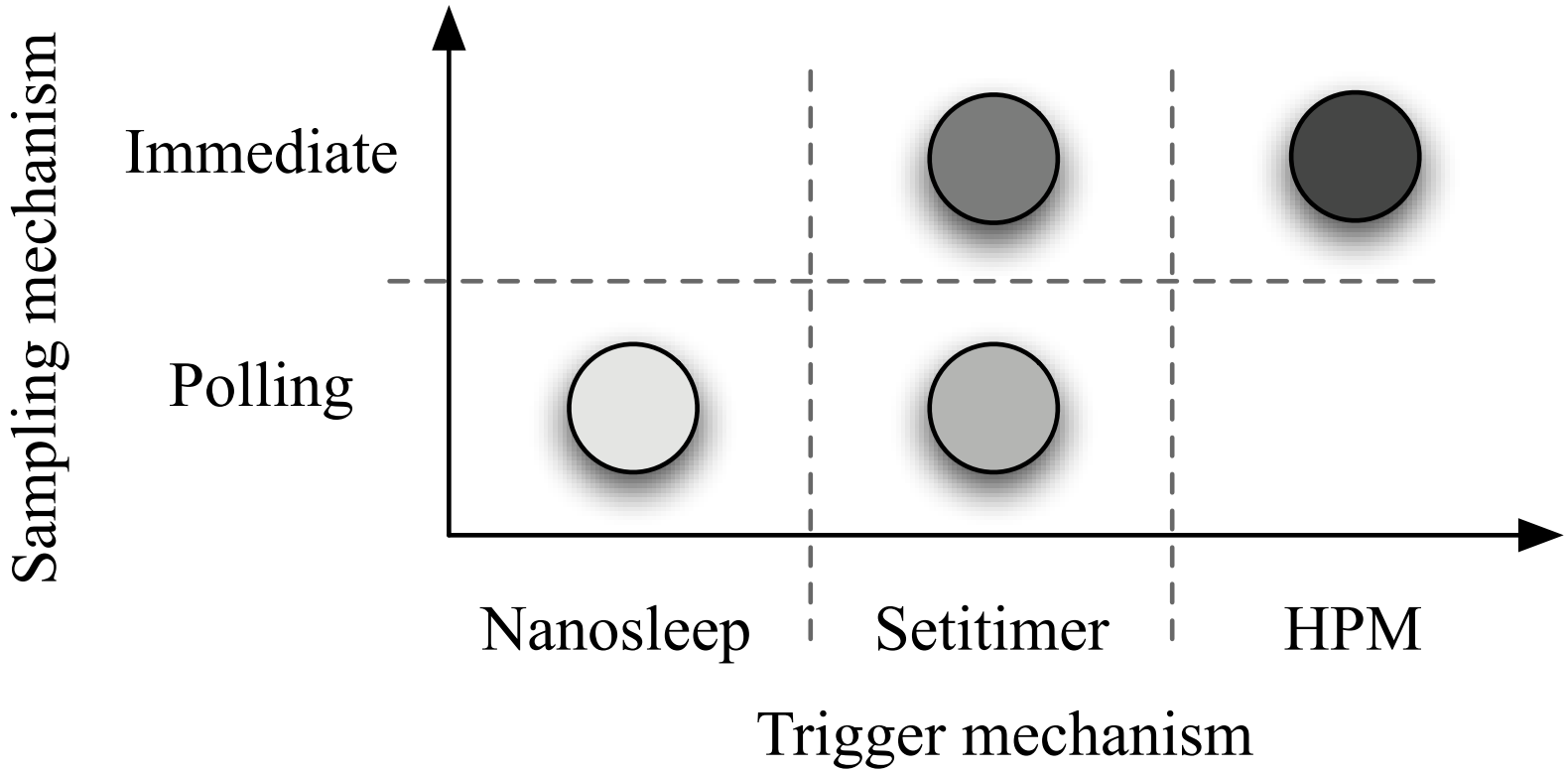
Using hardware performance monitors to initiates the sample-taking process

- HPMs are supported by most processors
- No general OS support yet for HPMs
- Count elapsed clock cycles
- Generate an interrupt when a counter overflows

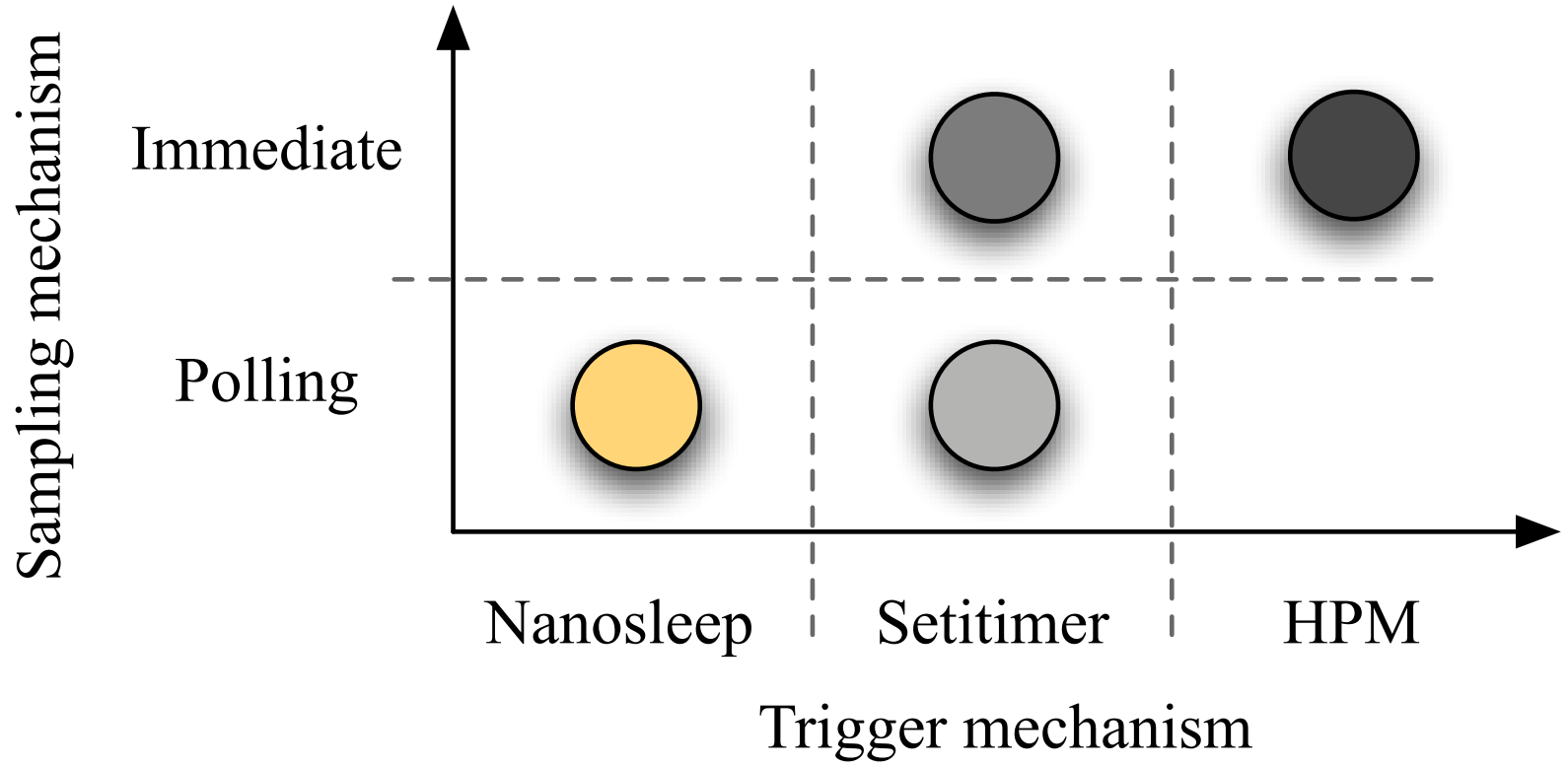
The sampling mechanism determines which method was executing

- *Polling points*: set a flag, grap the method ID when execution reaches a yieldpoint
- *Immediate*: no flag, no yieldpoint, just take a grab the PC from the stack when triggered, and use the PC to lookup the method ID

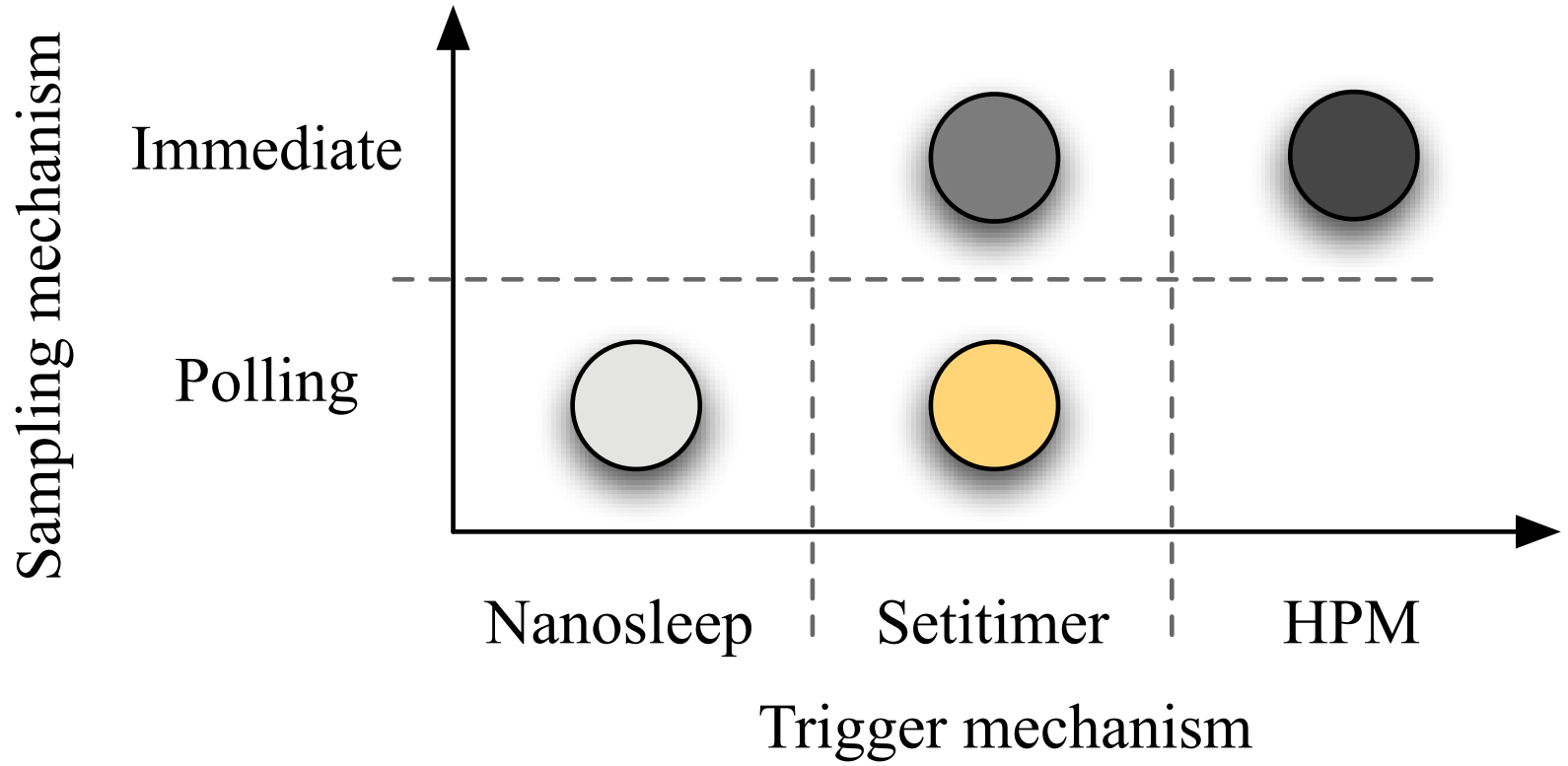
Timer-based profilers exist in many flavours



BEA's JRockit



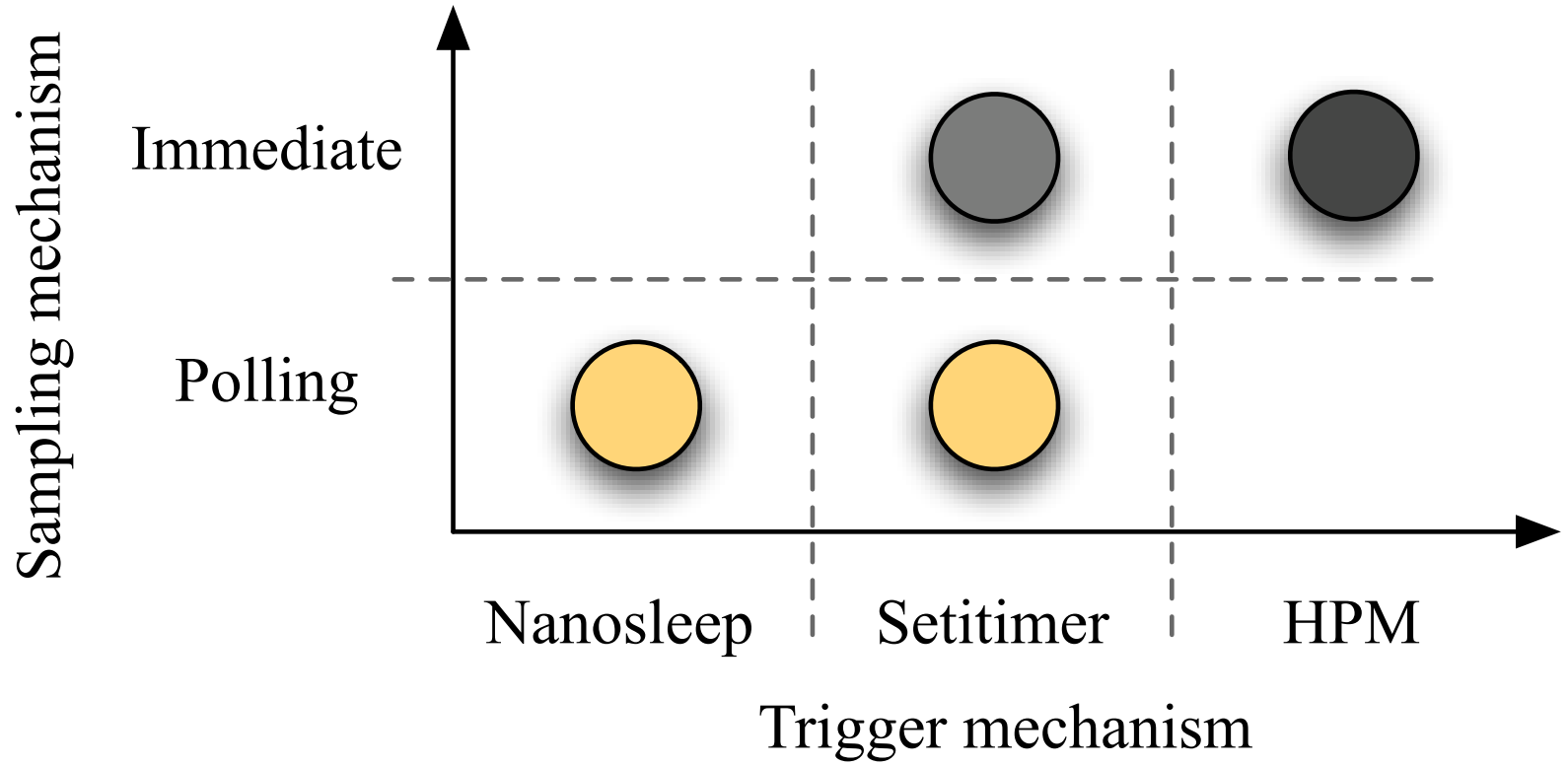
IBM's J9 VM


Sampling mechanism
Immediate
Polling
Nanosleep
Setitimer
HPM
Trigger mechanism

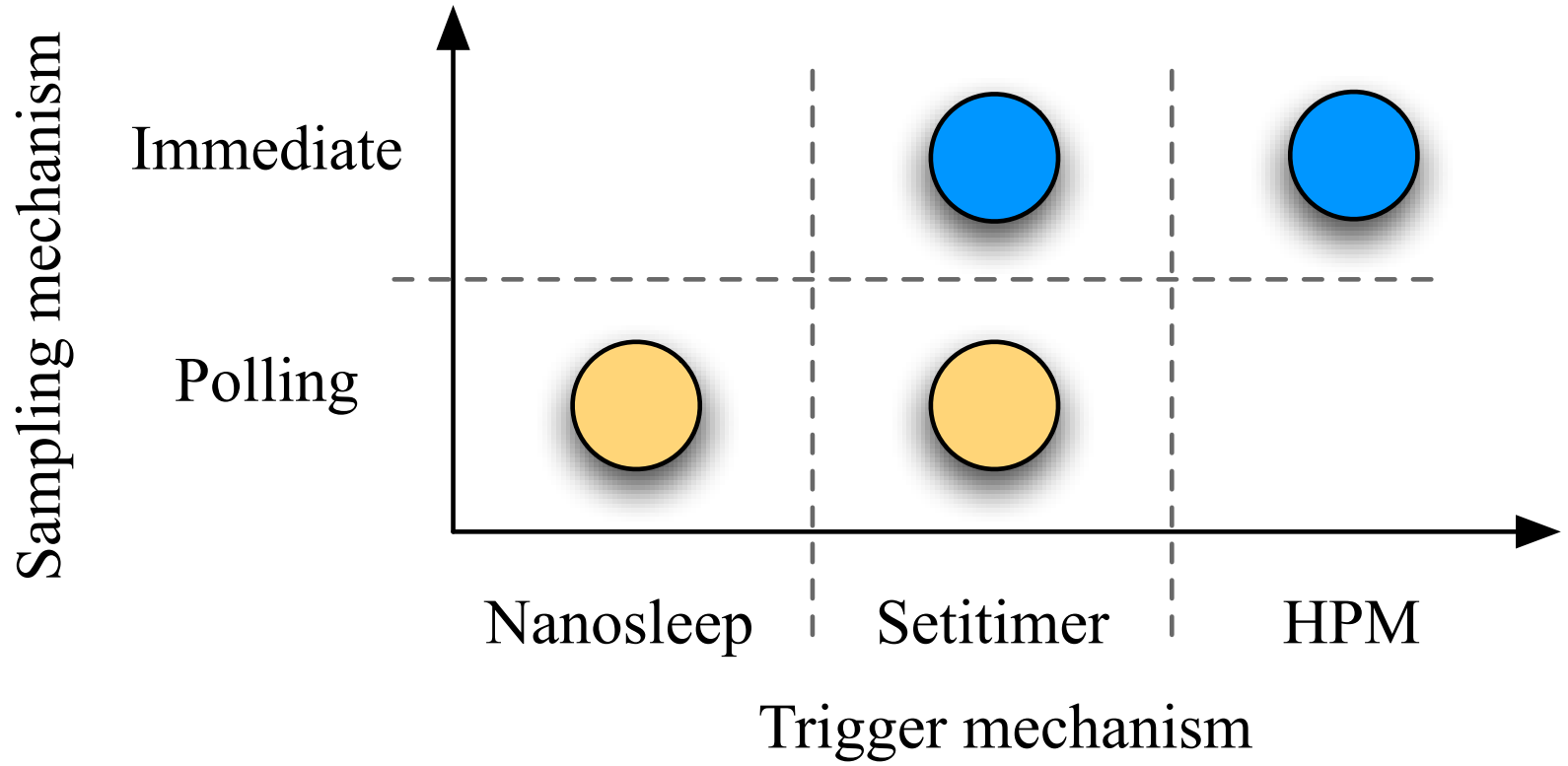
Sun's HotSpot VM

- No information available
- Uses only one level of optimization
- Suspect that it uses a counter-based solution rather than a timer-based solution

IBM's Jikes RVM
(both implemented)



We added those to IBM's Jikes RVM



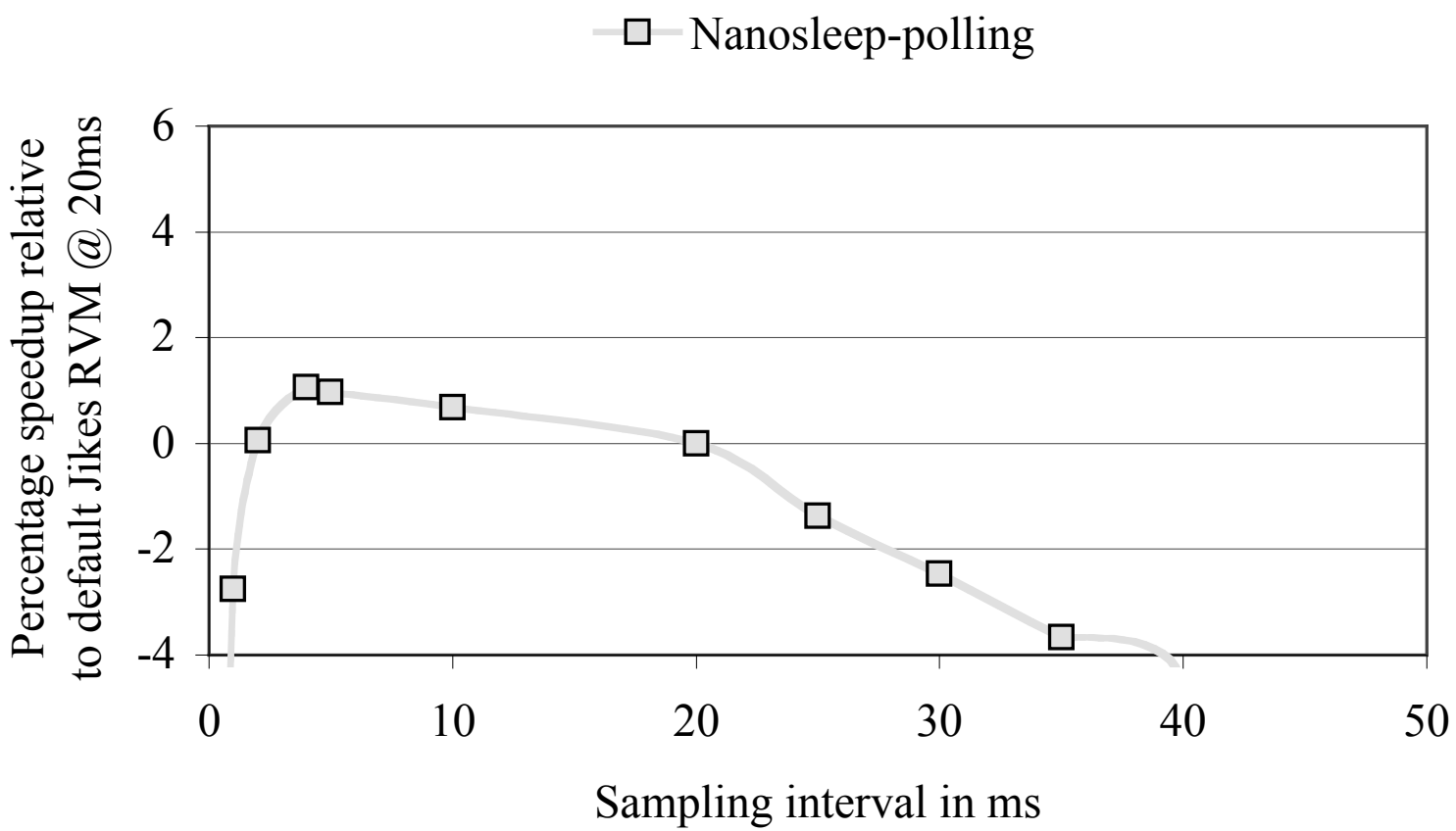
Experimental design

- Three benchmarks suites:
  - SpecJVM98
  - DaCapo
  - PseudoJBB
- Two hardware platforms:
  - AMD XP 3000
  - AMD XP 1500

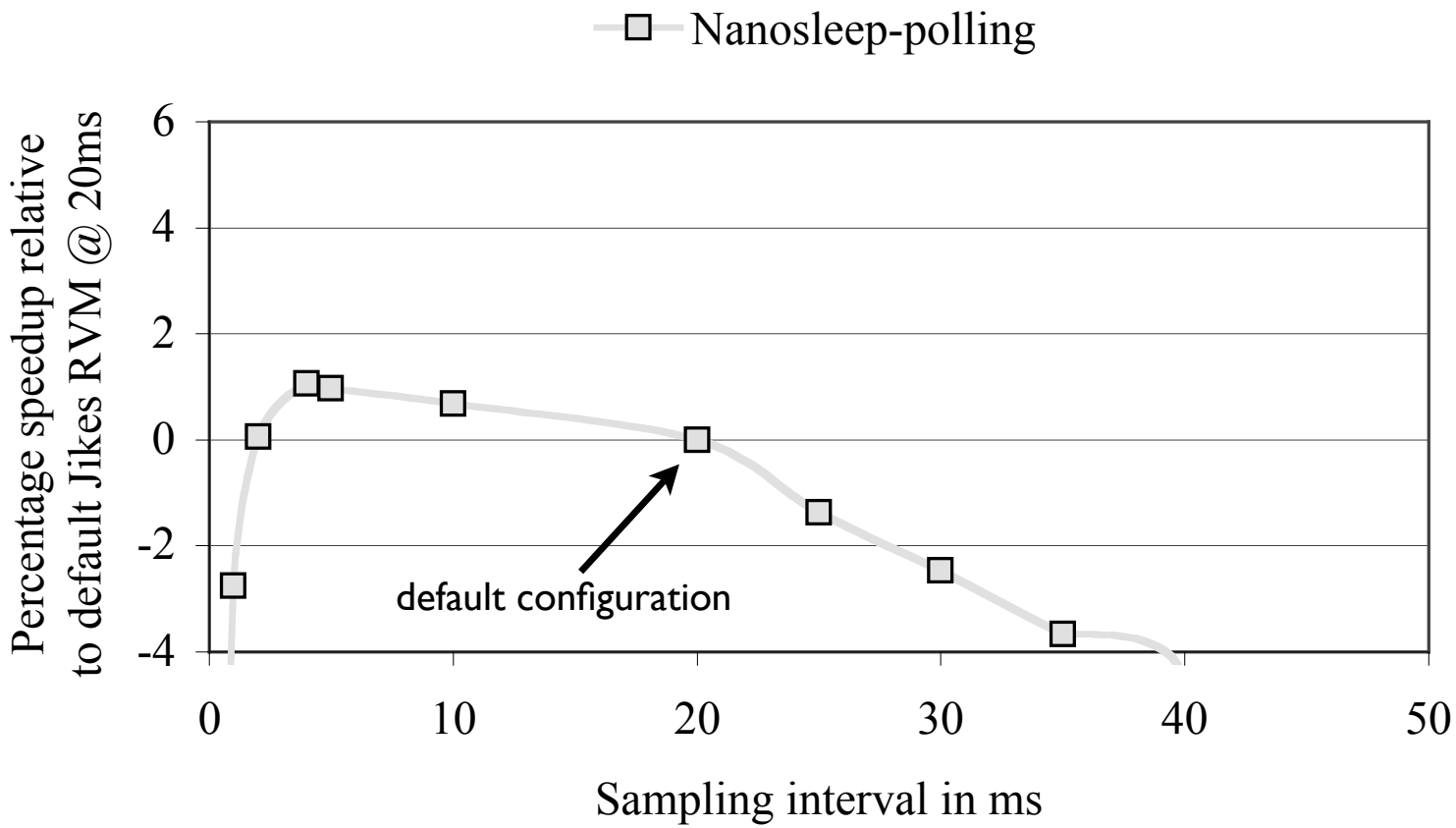
Data analysis

- 11 runs, removed first run, averaged remaining 10 runs
- For validating speedup we used a one-sided student t-test with a 95% confidence level

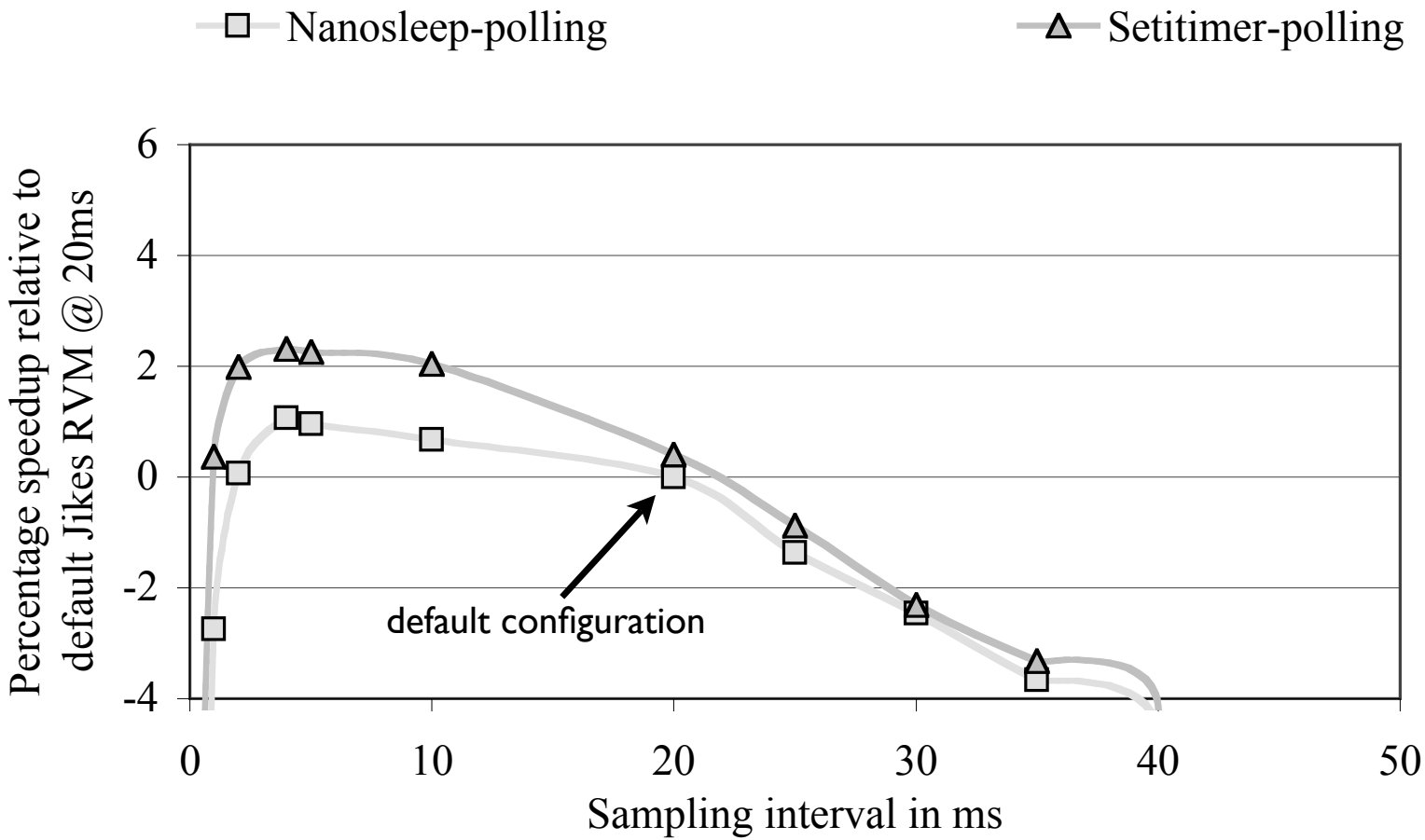
Average percentage speedup



Average percentage speedup



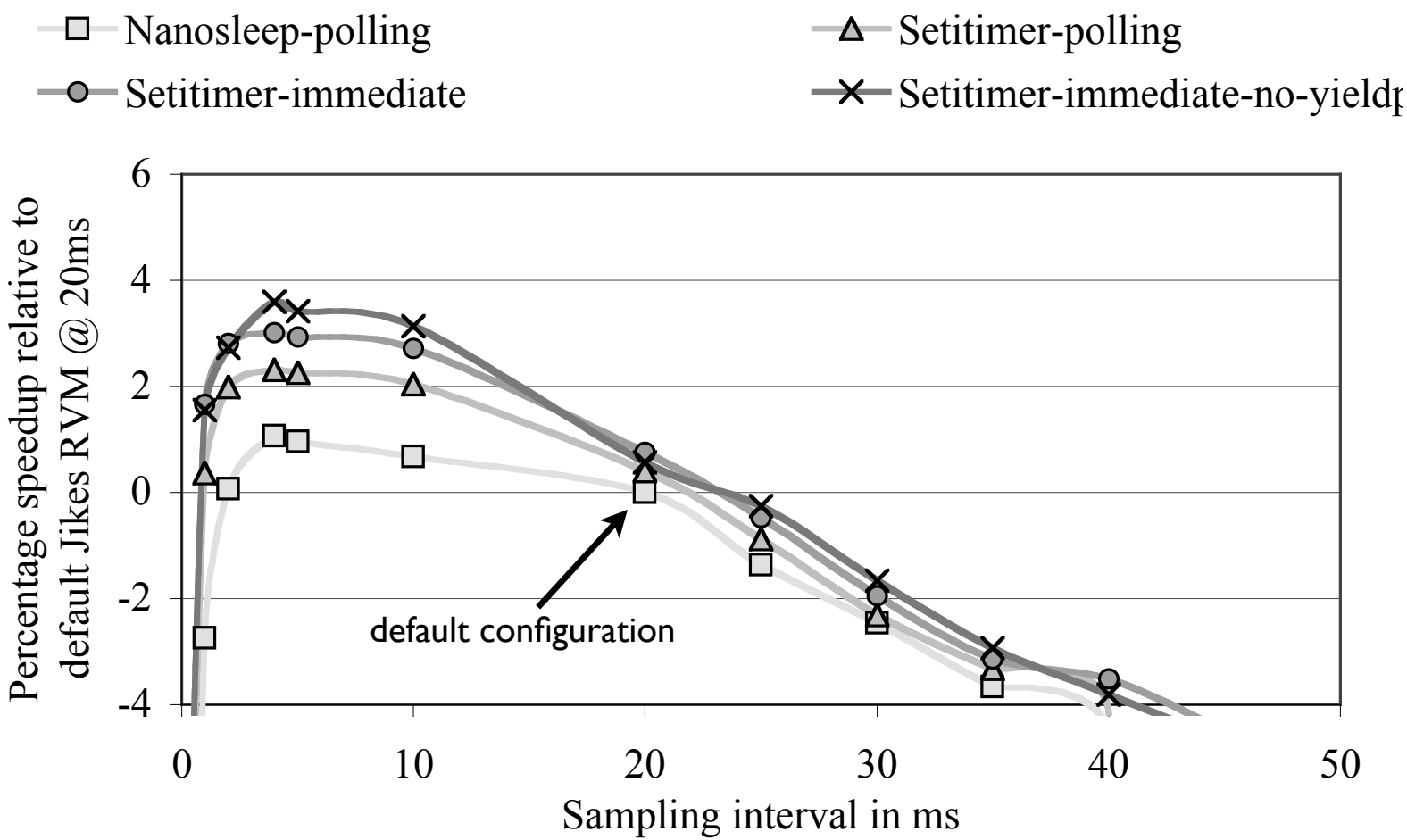
Average percentage speedup



We implemented a "setitimer-immediate" approach in Jikes RVM

- Removed polling points code
- Can sample all methods

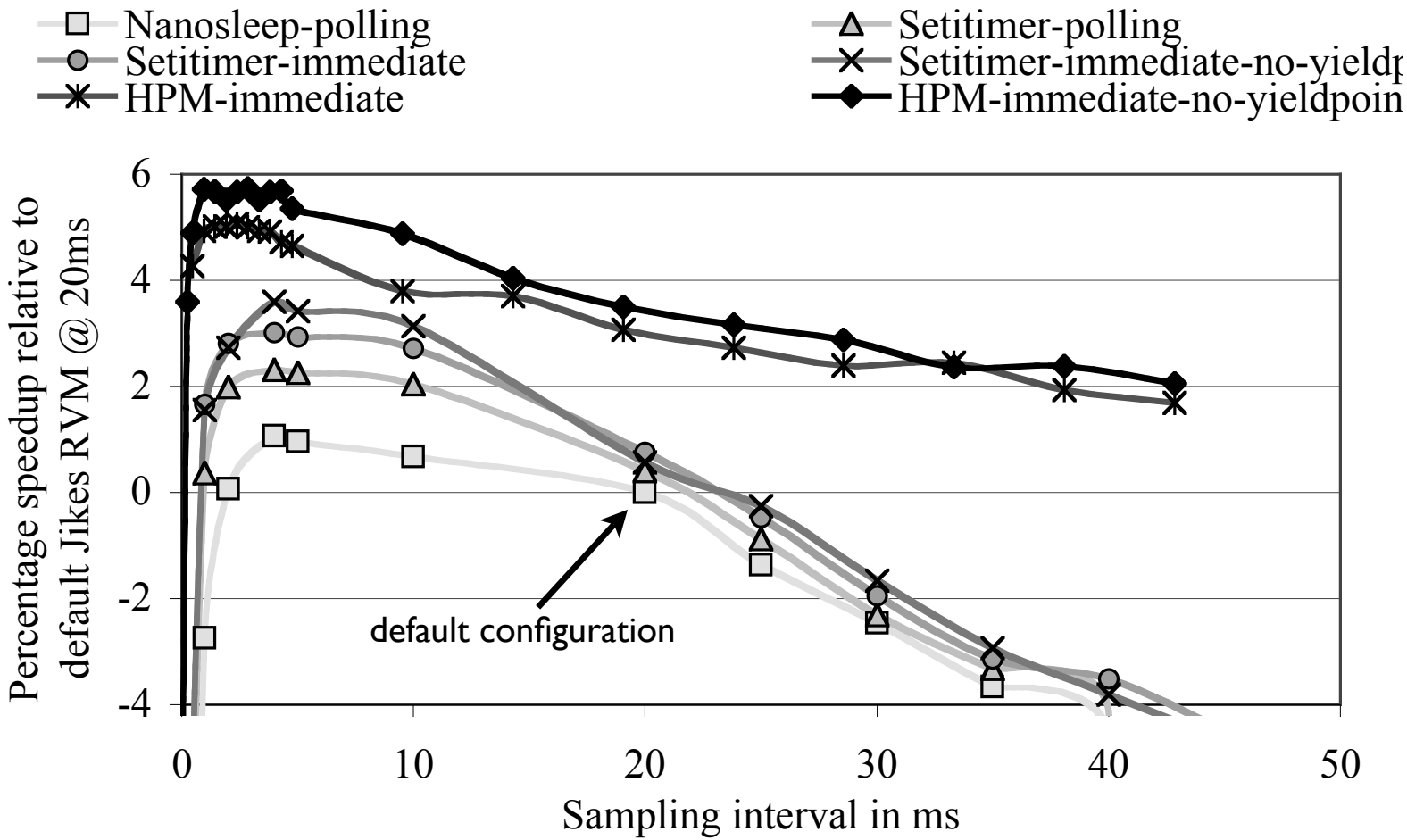
Average percentage speedup



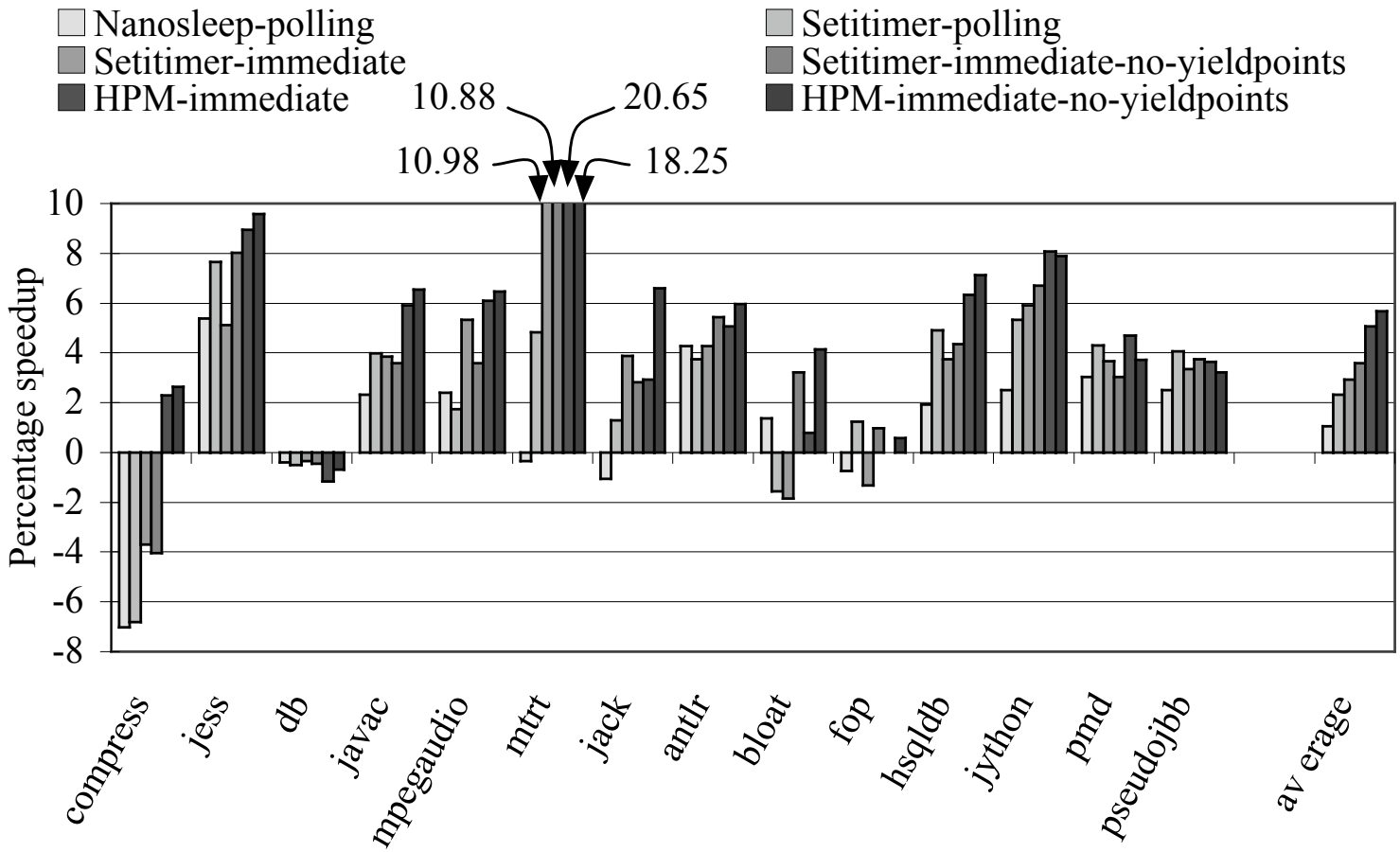
We implemented a "HPM-immediate" approach in Jikes RVM

- Removed polling points from code
- Can sample all methods
- No OS limitations
- No delay between expiration of the timer and scheduling of the thread
- Fixed timer tick intervals become relatively less frequent as processors get faster

Average percentage speedup



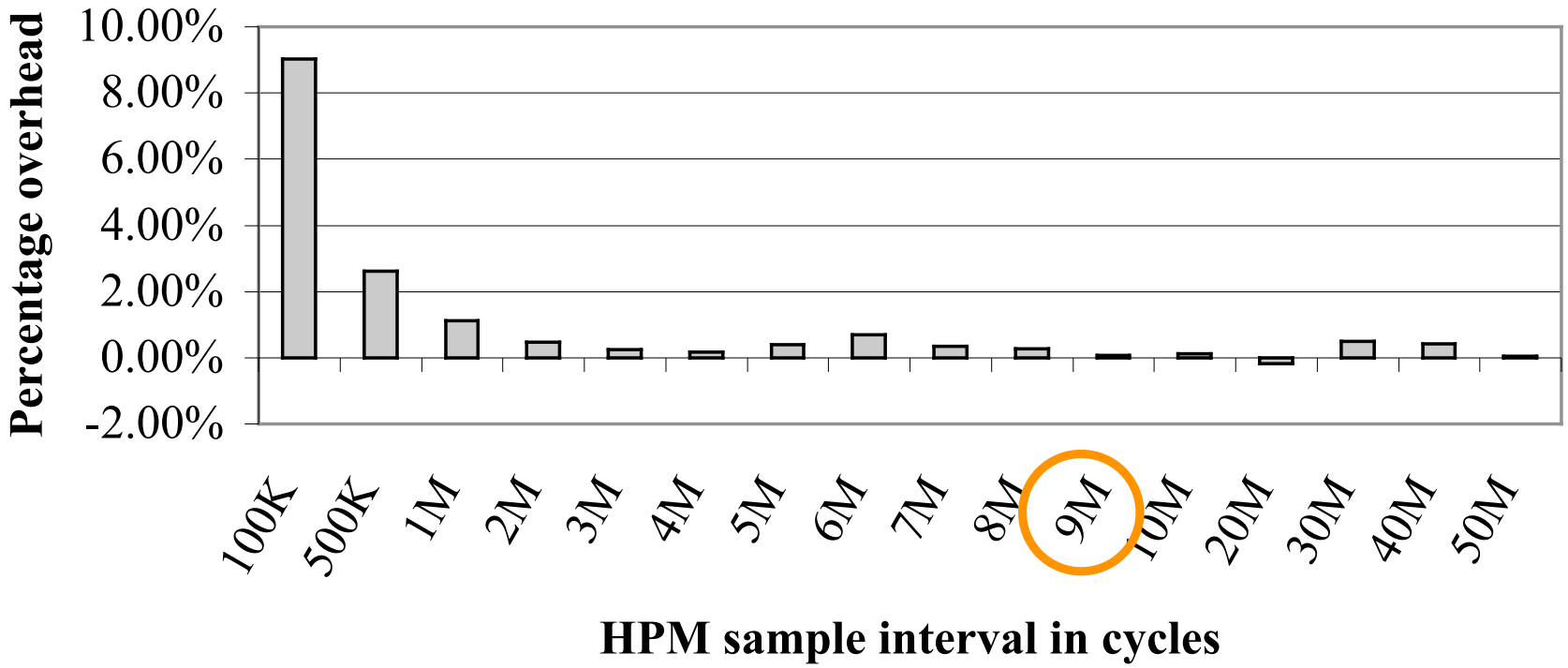
Average percentage speedup



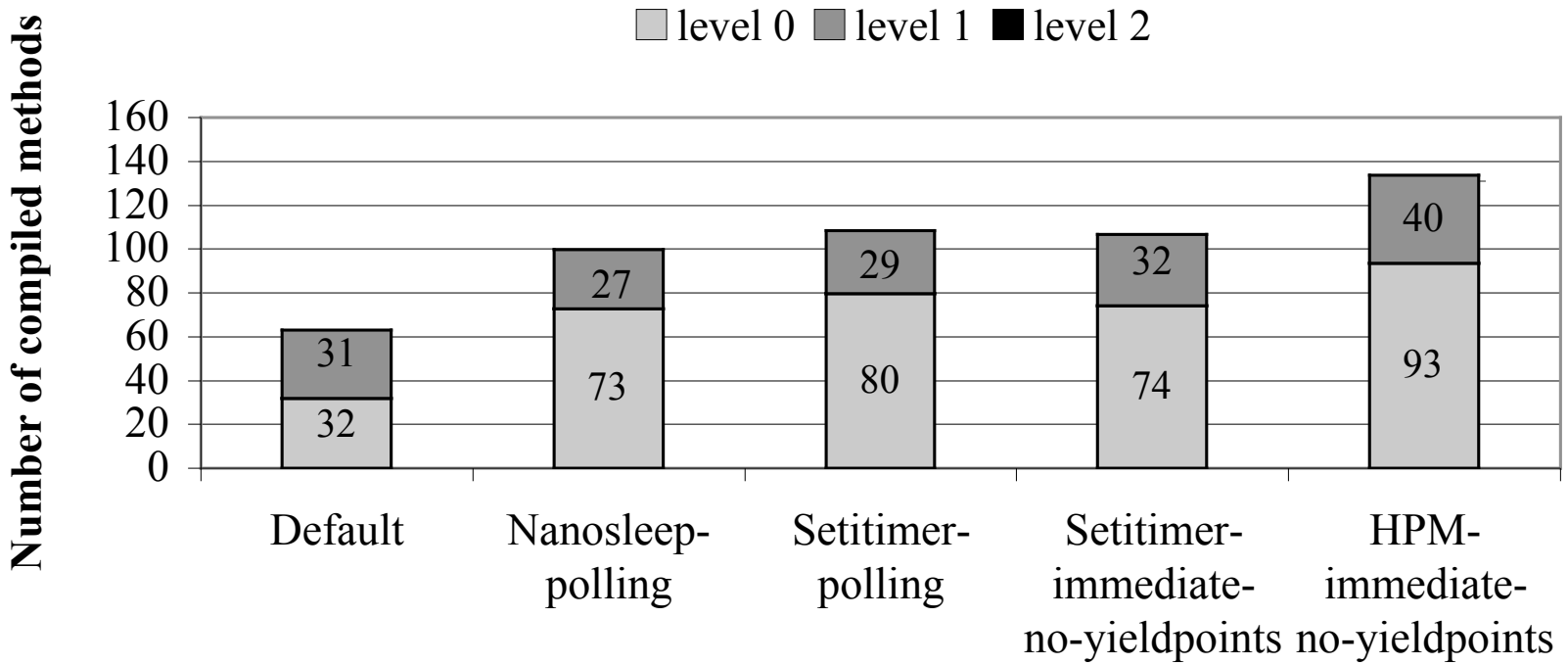
Average speedup of 5.7% over default Jikes RVM

- Immediate sampling: 3.0%
- HPM triggering: 2.1%
- Removing polling points: 0.6%

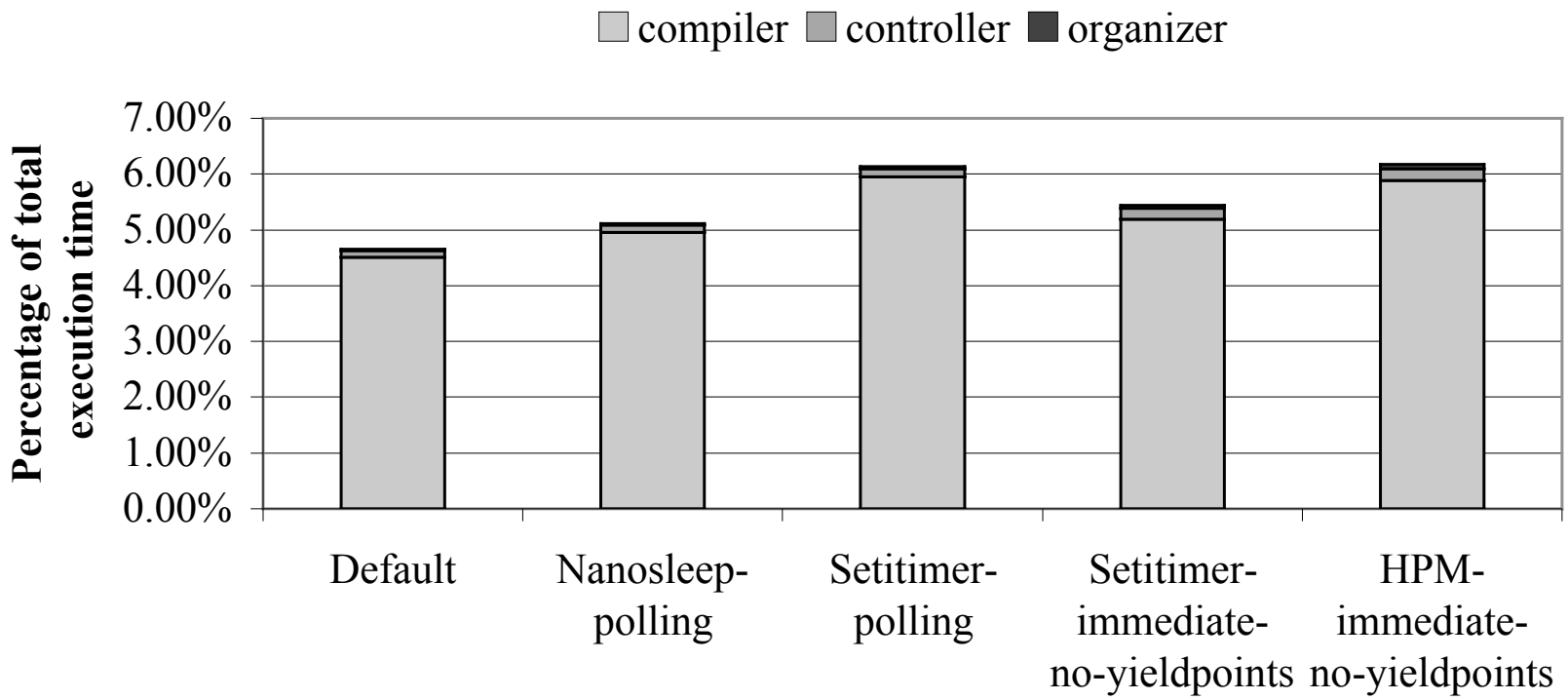
Producing HPM samples doesn't add much overhead



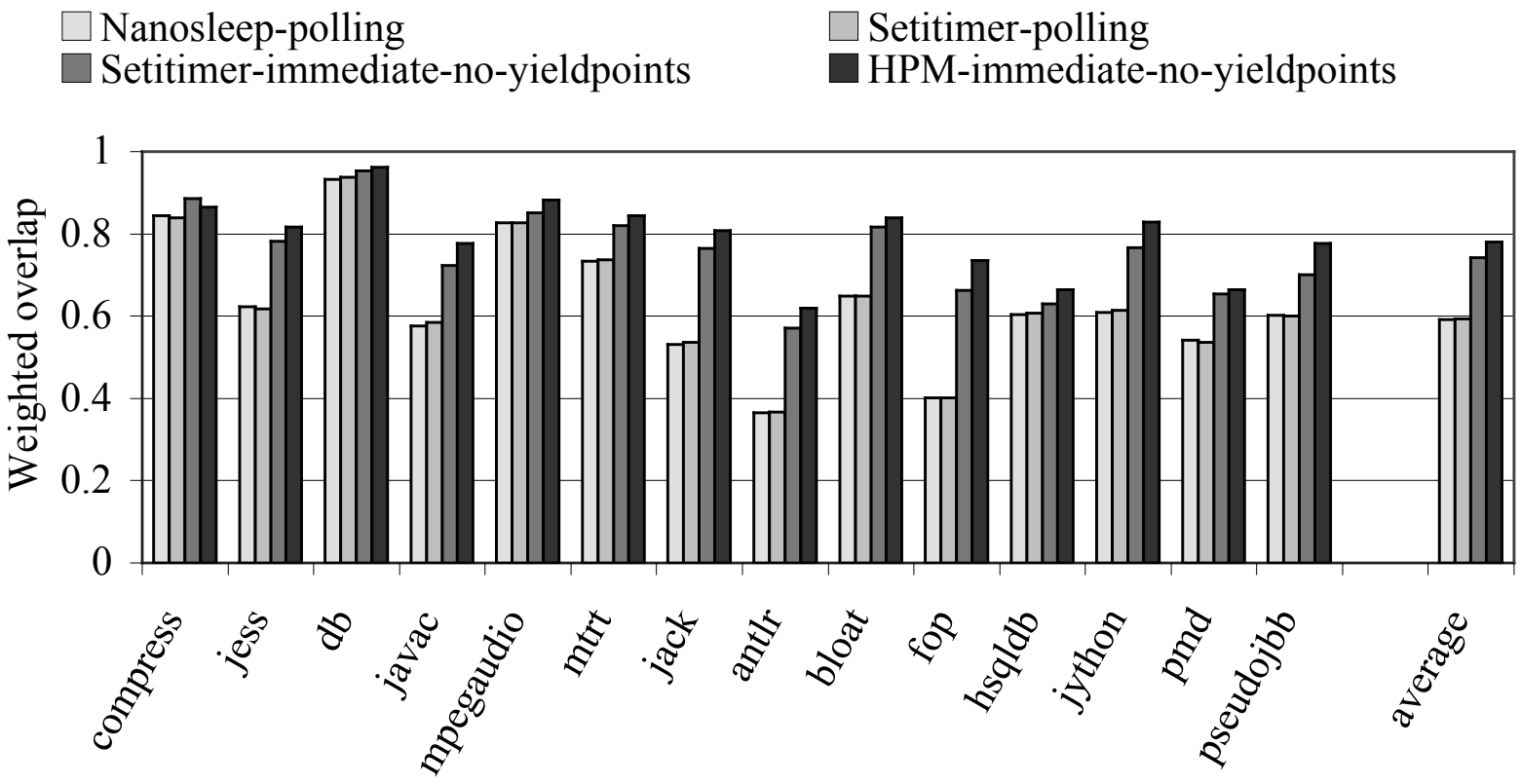
HPM-Immediate optimizes twice as many methods



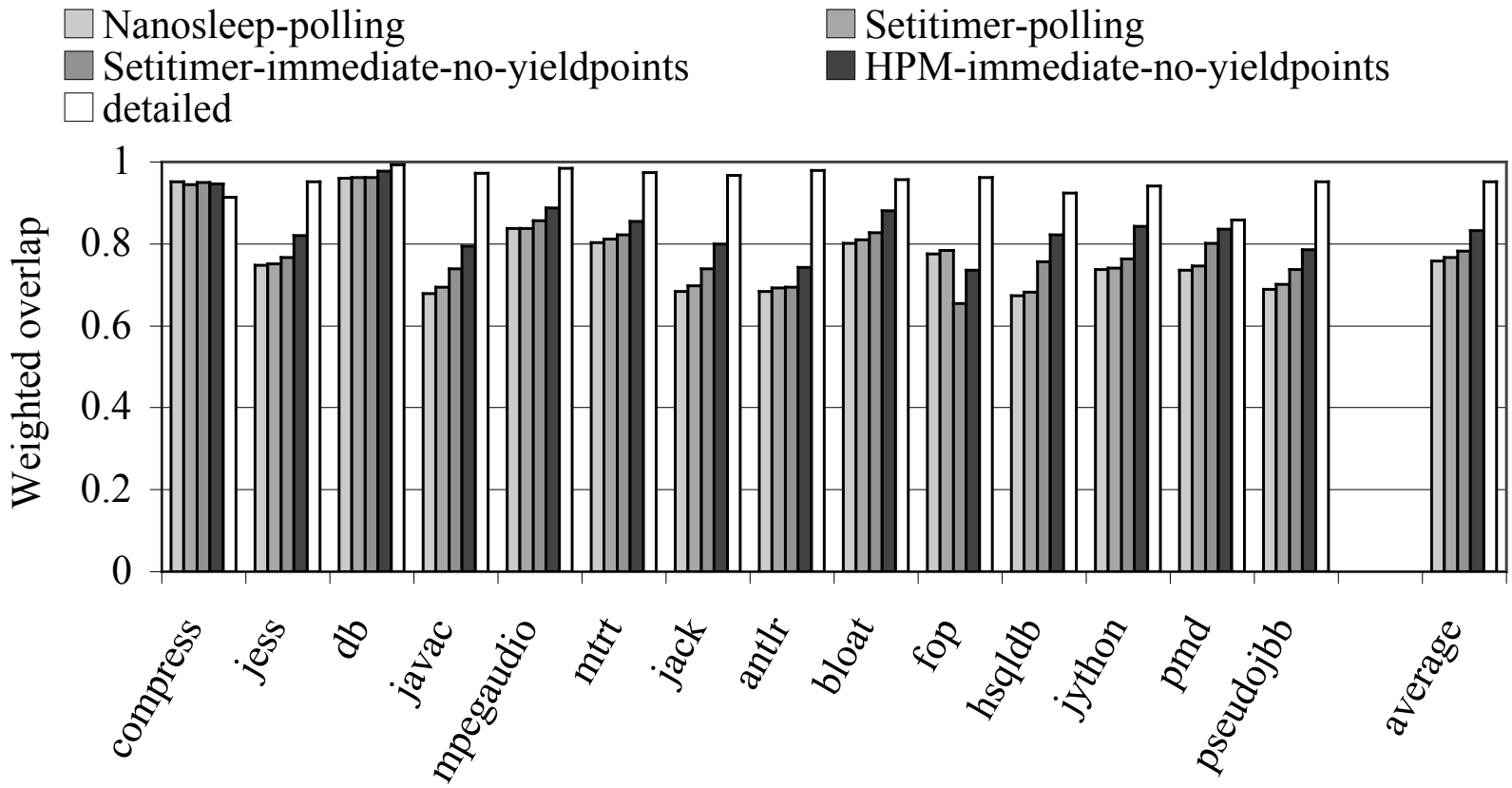
Consuming HMP-samples doesn't add much overhead either



HPM-Immediate is more accurate



HPM-Immediate is more stable



Conclusions

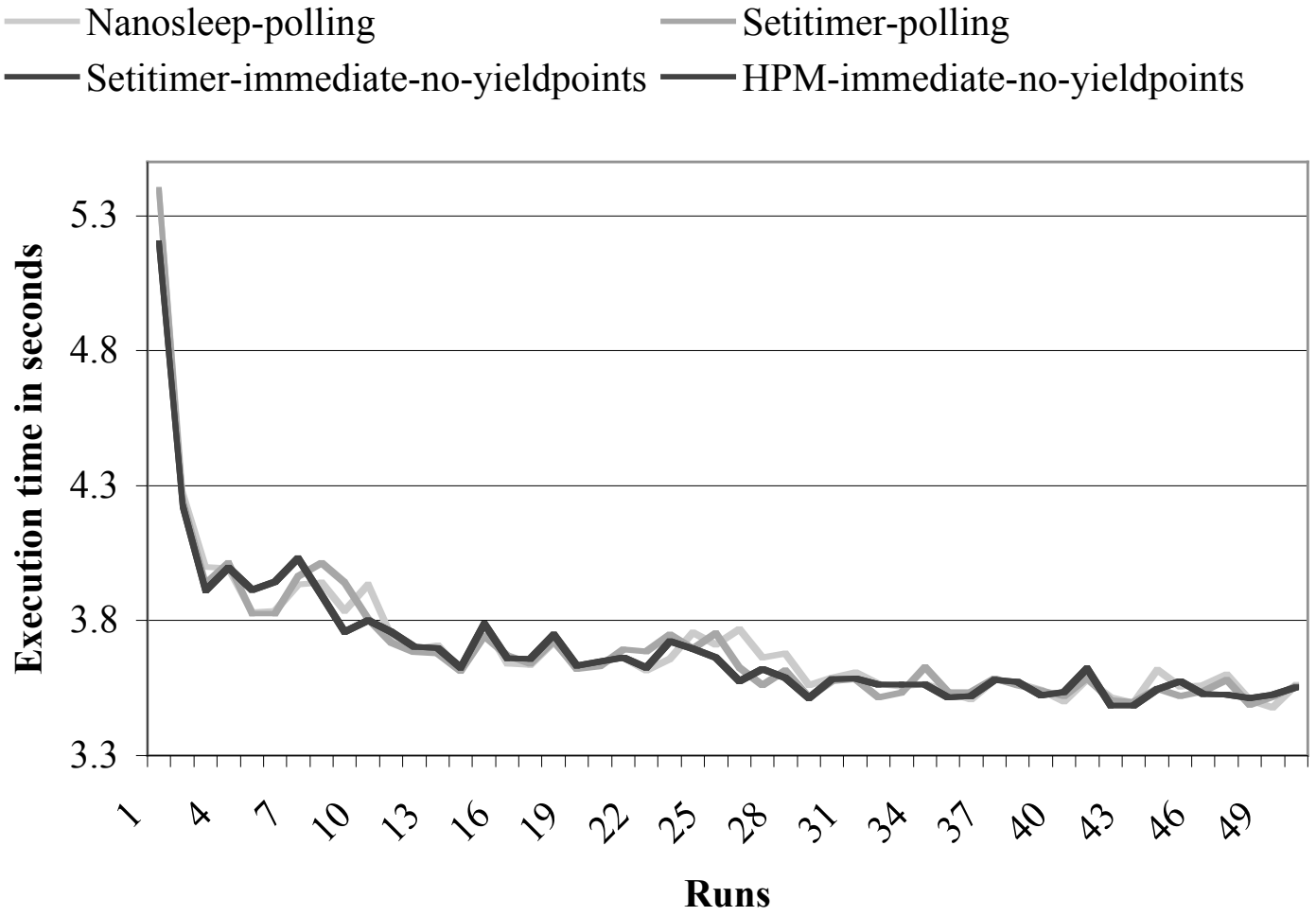
- Empirically evaluated the design space of several sample-based profilers.
- Described the design and implementation of HPM-sampling approach.
- Empirically evaluated the HPM-sampling in Jikes RVM, demonstrating that it improves performance by 5.7% on average, and up to 18.3%.
- Simple idea but clear win. Need for "black box" HPM functionality.

Future work

- Can we use non-cycle count events (i.e. cache miss count events)?
- Can we use it for other parts of the optimization system (i.e. method inlining)?

Thanks!

No real change for steady-state performance



Related work

- Lu et al.: for path profiling
- Adl-Tabatabai et al.: prefetching
- Lipasti et al.: speculative optimization
- Schneider et al.: locality improvements
- Merten et al.: branch behavior
- ...