Analyzing and Optimizing Task Granularity on the JVM

Andrea Rosà, Eduardo Rosales, Walter Binder

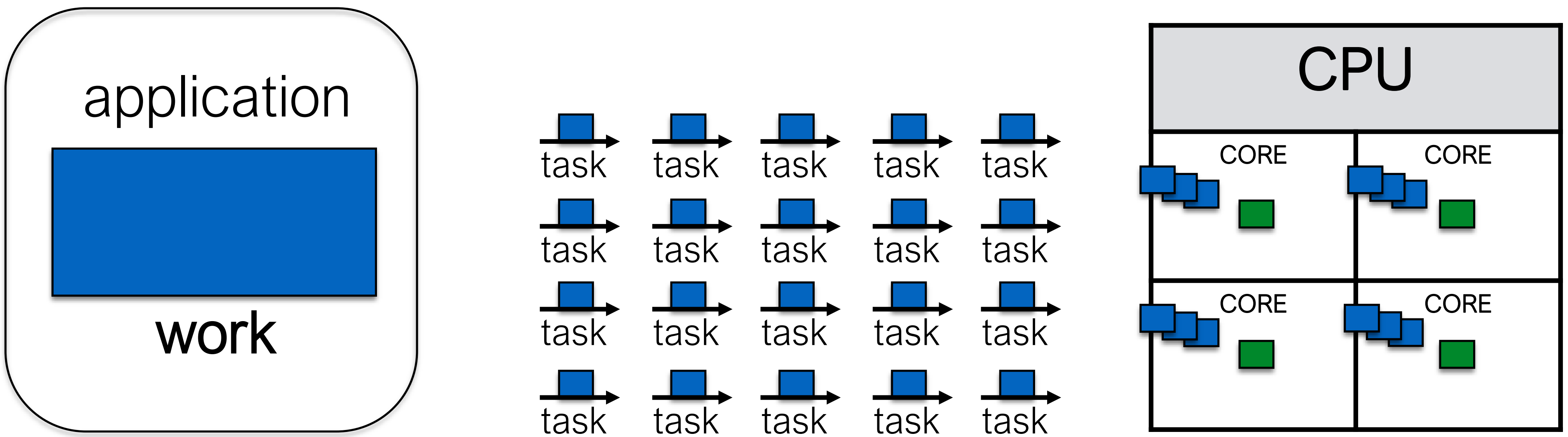
Università della Svizzera italiana (USI), Faculty of Informatics, Lugano, Switzerland

CGO 2018
February 26th, 2018
Vienna, Austria

Task Granularity

- The amount of work to be performed by parallel tasks

Fine-grained tasks



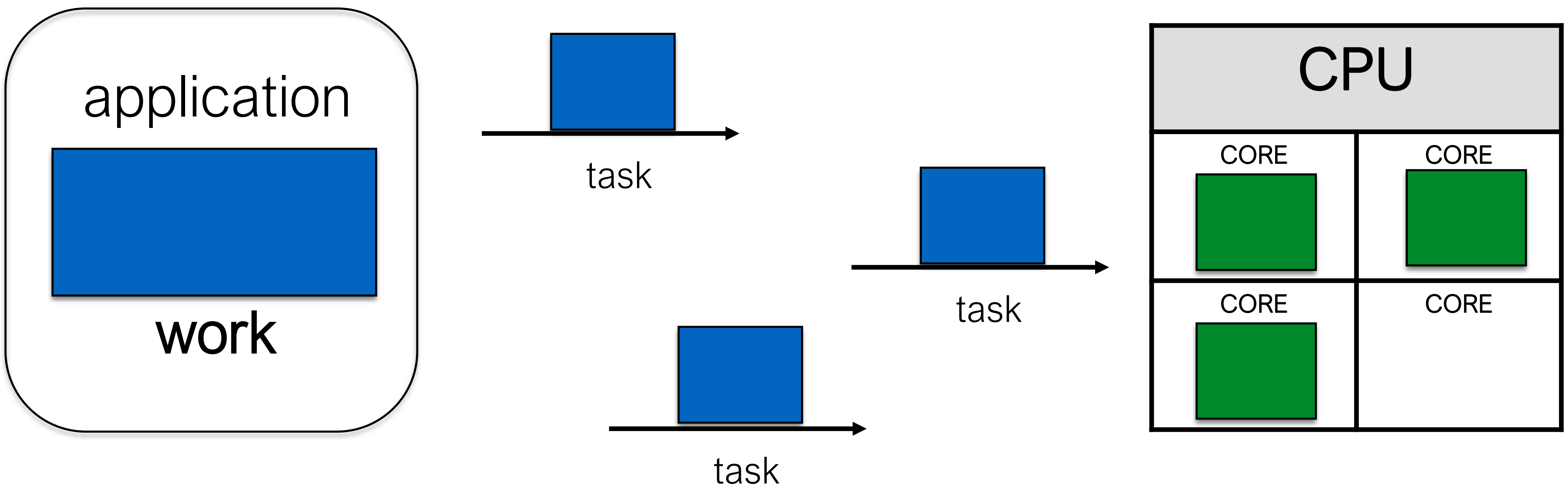
Parallelization overheads due to:

- Inter-thread communication
- Synchronization
- Task scheduling
- Task creation

Task Granularity

- The amount of work to be performed by parallel tasks

Coarse-grained tasks



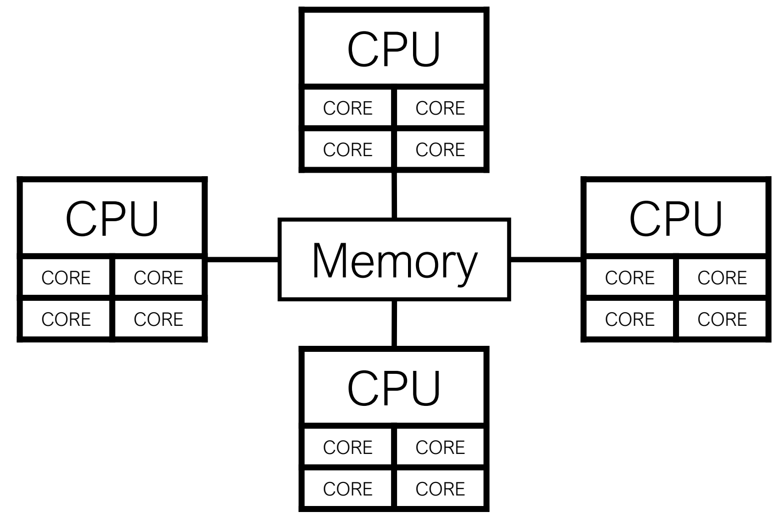
Missed parallelization opportunities:

- Low CPU utilization
- Load imbalance

Università
della
Svizzera
italiana

Task Granularity

- Our scope:



Multi-threaded, task-parallel applications executing on a single JVM

A single shared-memory multicore machine

- Task granularity little analyzed in the literature

Università della Svizzera italiana

Our Work

- **Goal: provide a better understanding of task granularity**
- Contribution:
  - `tgp`: task-granularity profiler for the JVM
  - Task-granularity analysis on DaCapo [1] and ScalaBench [2]
  - Task-granularity optimization
- Challenges:
  - Recognize every task spawned
  - Accurately measure granularity for each task
  - Collect metrics with low perturbation

[1] Blackburn et al. *The DaCapo Benchmarks: Java Benchmarking Development and Analysis*. OOPSLA'06.
[2] Sewe et al. *DaCapo con Scala: Design and Analysis of a Scala Benchmark Suite for the JVM* . OOPSLA'11.

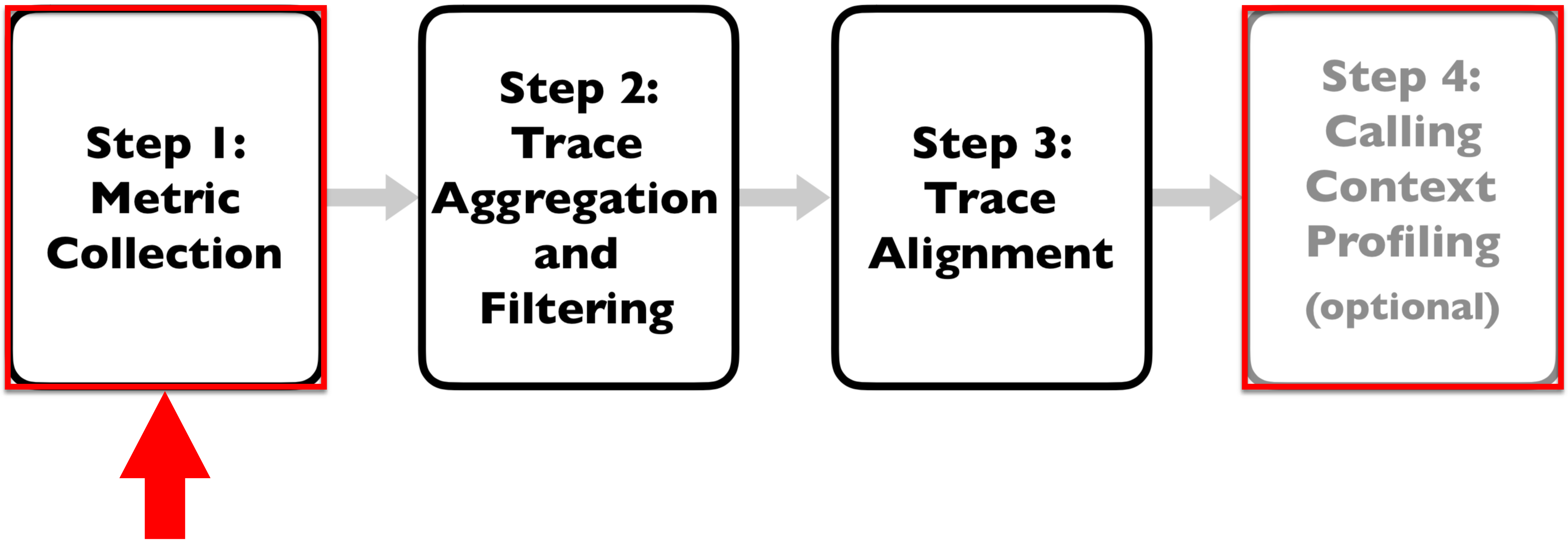
tgp

- **tgp**: a **T**ask-**G**ranularity **P**rofiler for multi-threaded, task-parallel applications executing on the JVM
- Features:
  - Profile accurate metrics on task granularity
  - Show the impact of task granularity on application and system performance
  - Actionable profiles
    - Users optimize code portions suggested by `tgp`

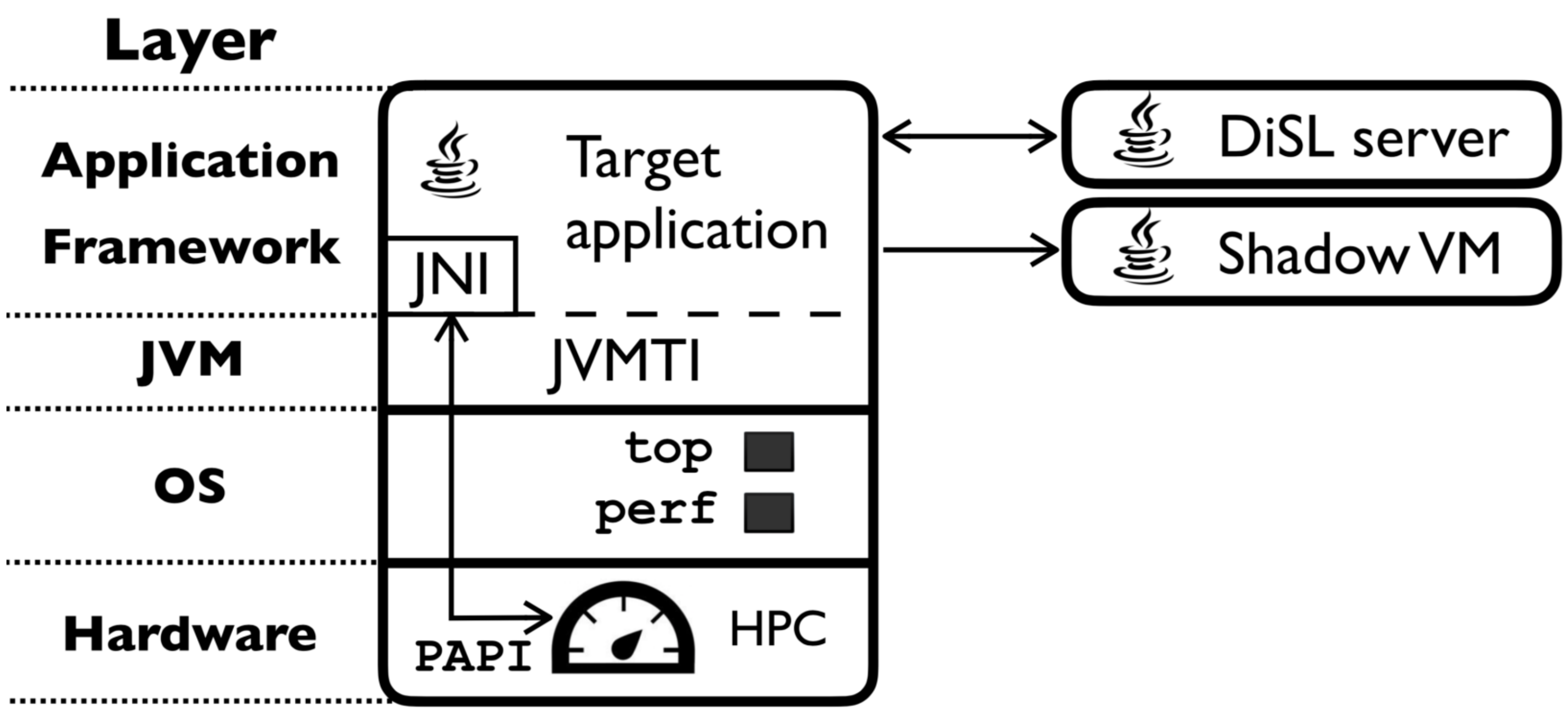
Task Model

- **Task**: every instance of
  - `java.lang.Runnable`
  - `java.util.concurrent.Callable`
  - `java.util.concurrent.ForkJoinTask`
- **Work**: code executed in the dynamic extent of
  - `Runnable.run()`
  - `Callable.call()`
  - `ForkJoinTask.exec()`

Methodology



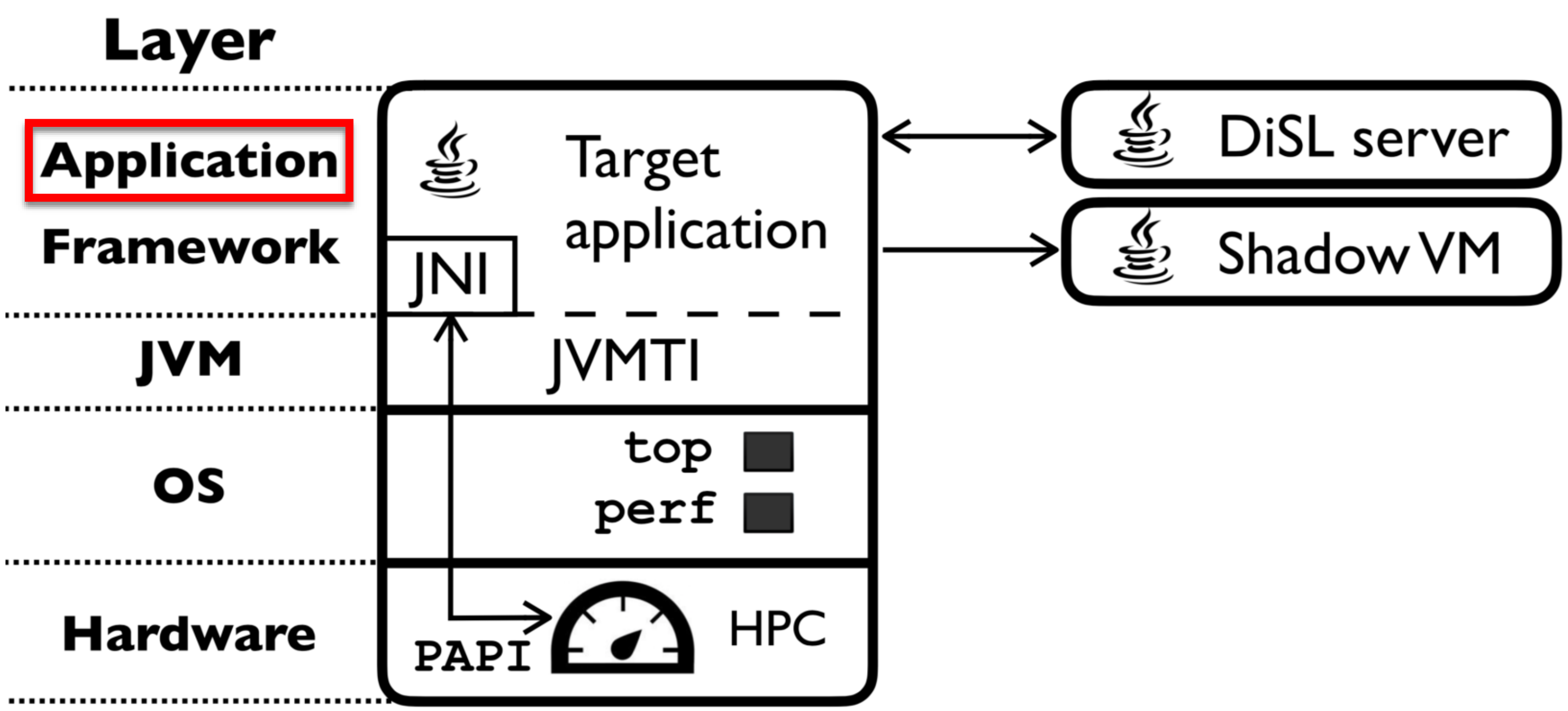
Metric Collection



- Vertical profiler [1]

[1] Hauswirth et al. *Vertical Profiling: Understanding the Behavior of Object-oriented Applications*. OOPSLA'04.

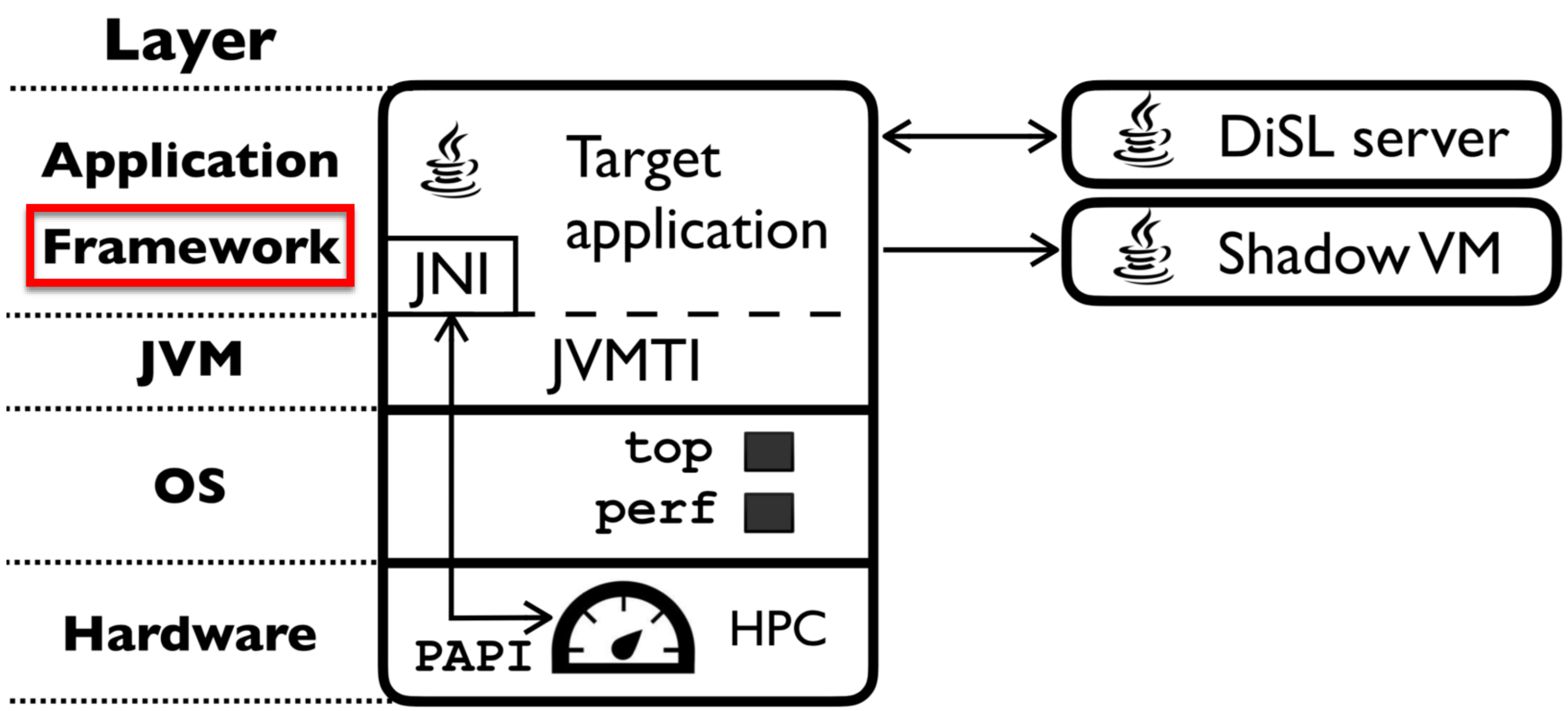
Metric Collection



- Creator/executor thread
- Start/end execution timestamp
  - Correlate task execution with OS-level metrics
- Task type

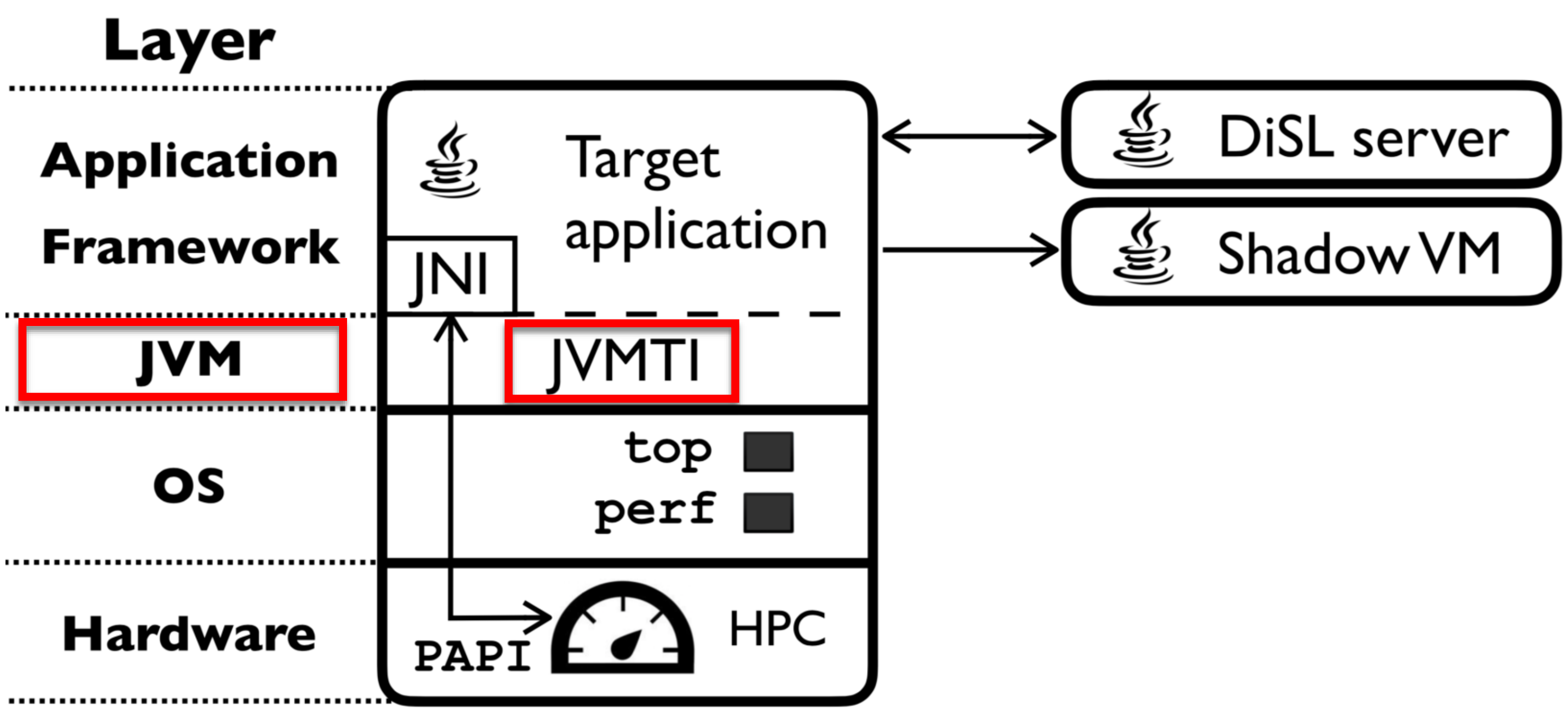
Università
della
Svizzera
italiana

Metric Collection



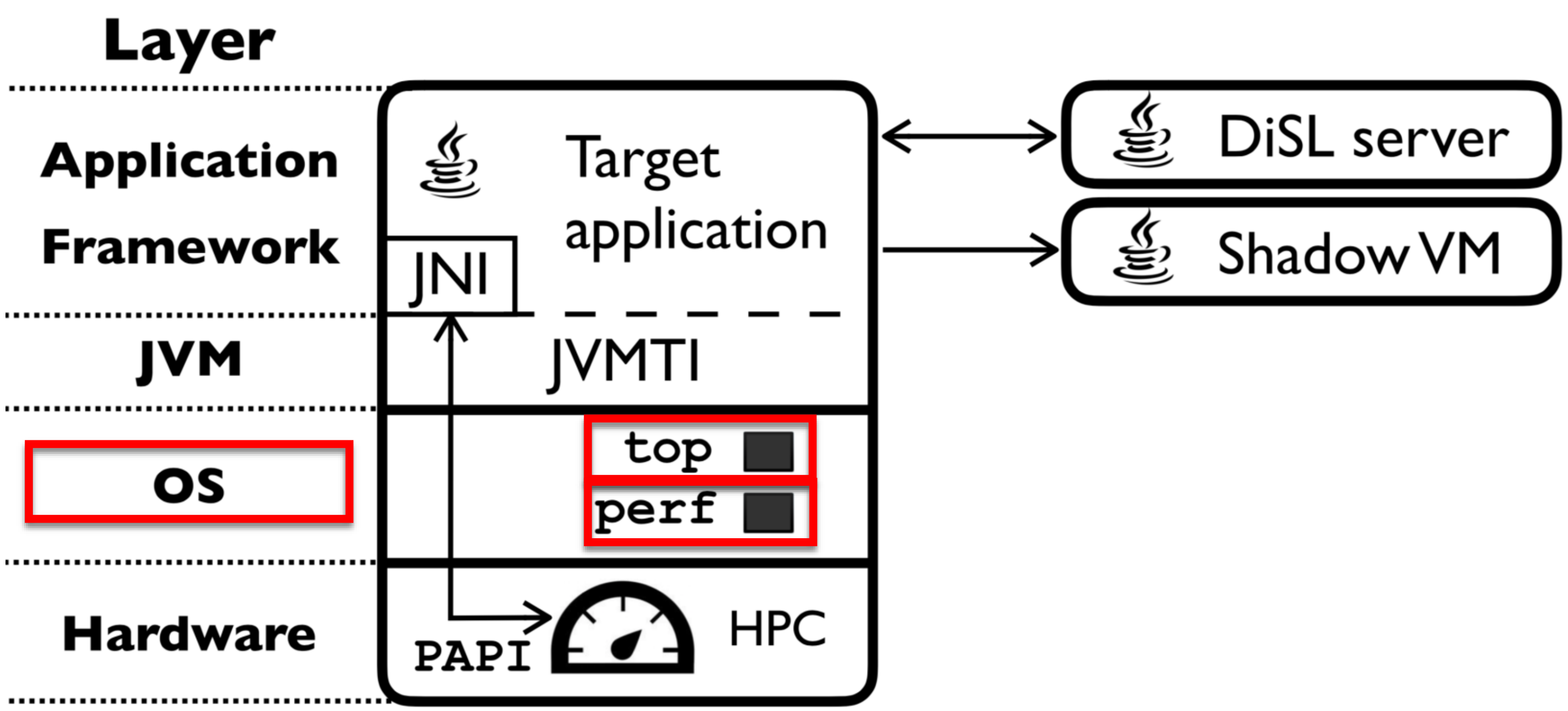
- Submissions to task executor frameworks
  - Instances of `java.util.concurrent.Executor`

Metric Collection



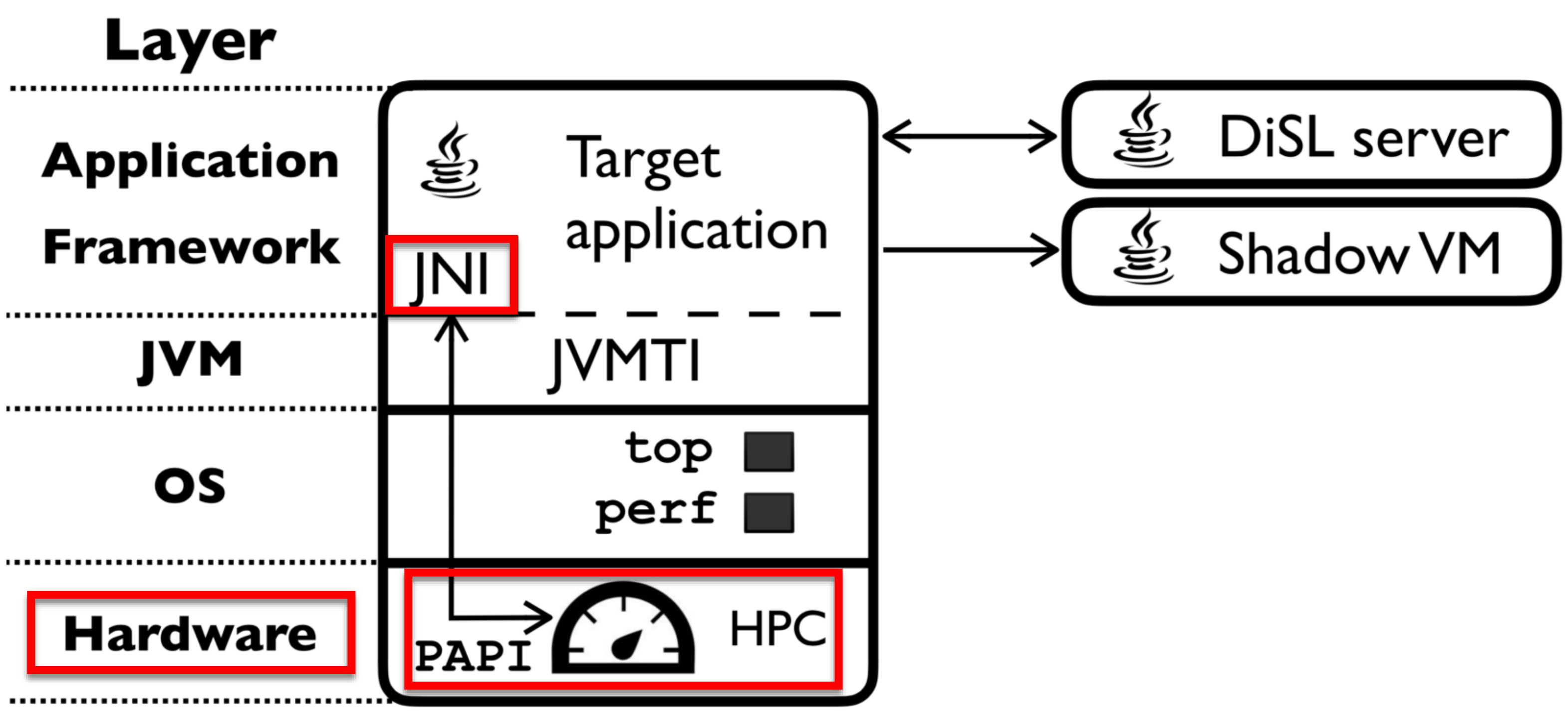
- Garbage collection activities
  - Correlate unexpected metric fluctuations with GC

Metric Collection



- CPU utilization (kernel and user)
  - Determine if CPU is well utilized (especially in the case of coarse-grained tasks)
- Context switches
  - Measure contention and synchronization among tasks

Metric Collection



- Reference cycles
  - Task-granularity measure

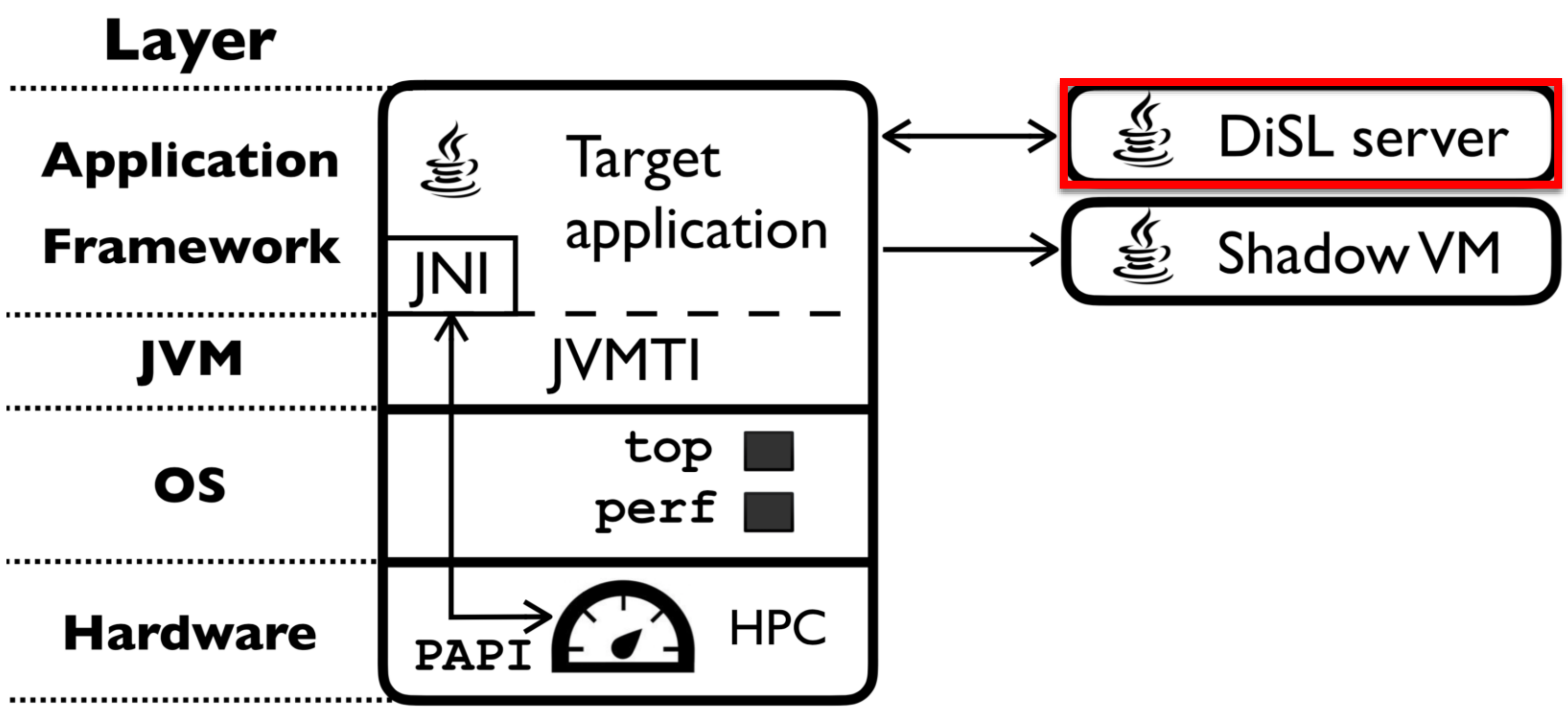
✓ Ensure consistent profiling in case of frequency scaling

✓ Represent instruction complexity

✓ Account for latencies (e.g., cache misses, misalignments)

Università
della
Svizzera
italiana

Metric Collection

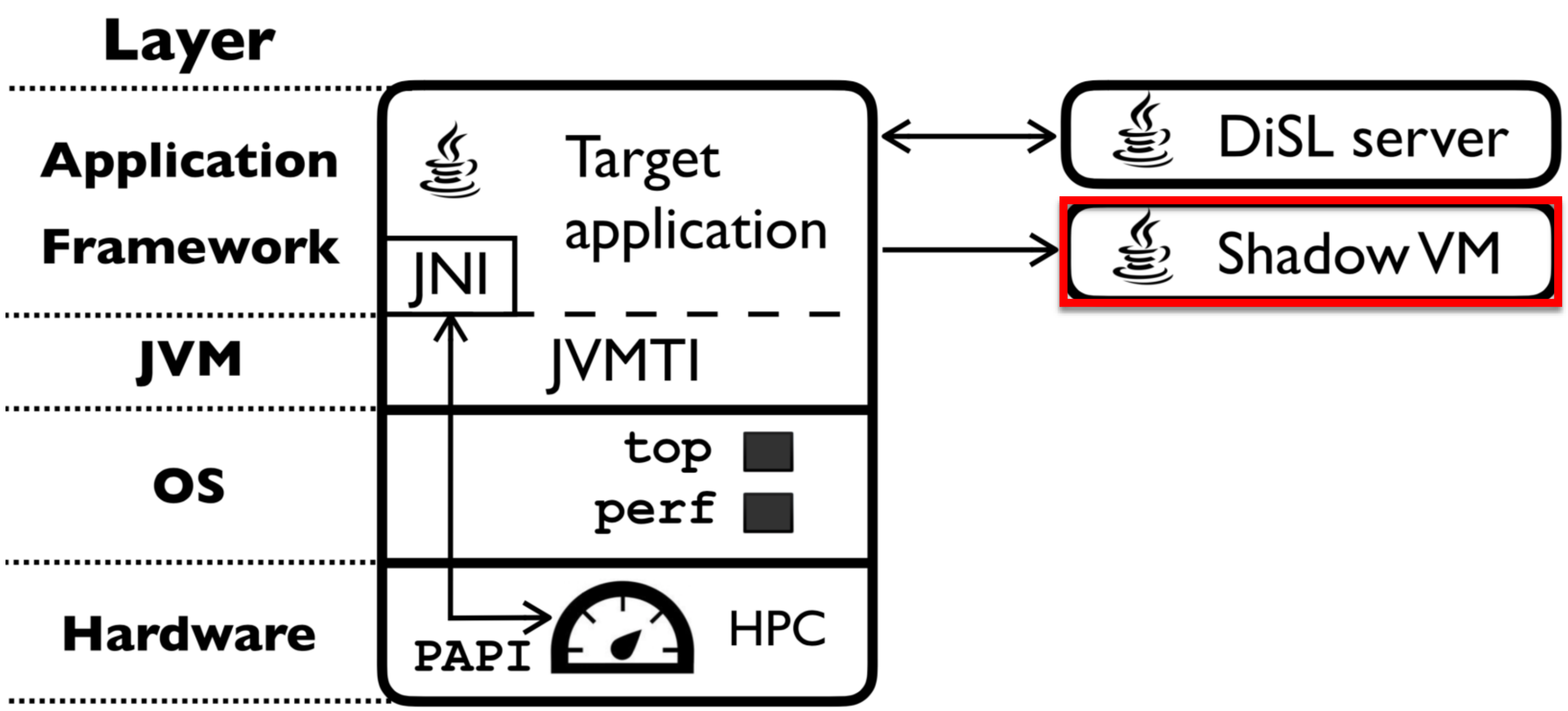


- DiSL server [1]: performs load-time instrumentation
- DiSL offers full bytecode coverage
  - All tasks detected
- Low instrumentation overhead
  - Minimized and efficient instrumentation code
  - No heap allocations

[1] Marek et al. *DiSL: A Domain-specific Language for Bytecode Instrumentation*. AOSD’12.

Università
della
Svizzera
italiana

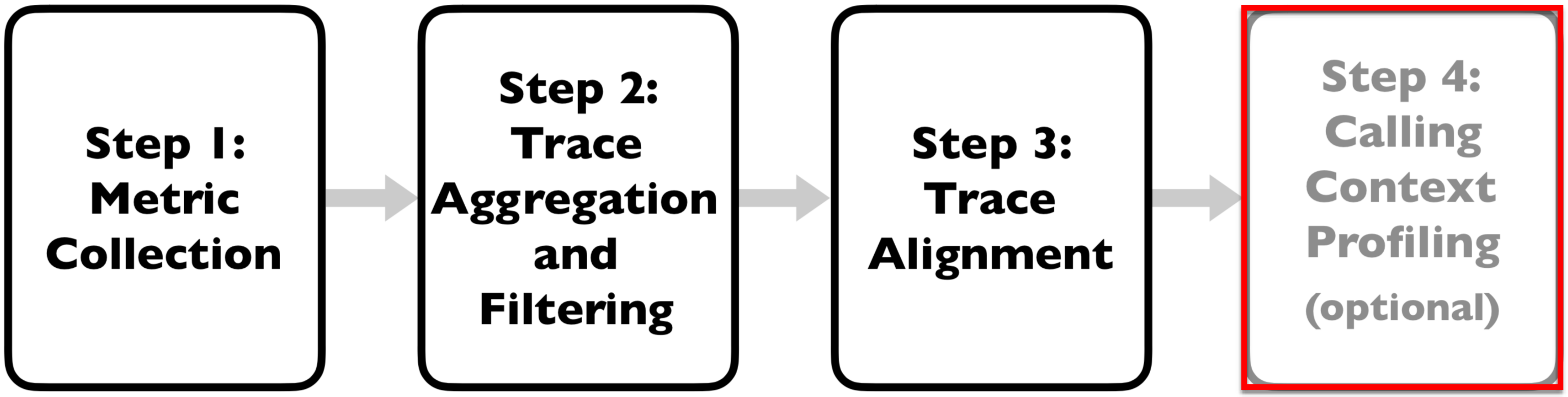
Metric Collection



- ShadowVM [1]: executes analysis code in isolation
  - ✓ No shared states between application and analysis VM
  - ✓ All thread lifecycle events intercepted (including shutdown)
  - ✓ Reduces application slowdown
    - Contains most of the profiling logic

[1] Marek et al. *ShadowVM: Robust and Comprehensive Dynamic Program Analysis for the Java Platform*. GPCE'13.

Calling Context Profiling



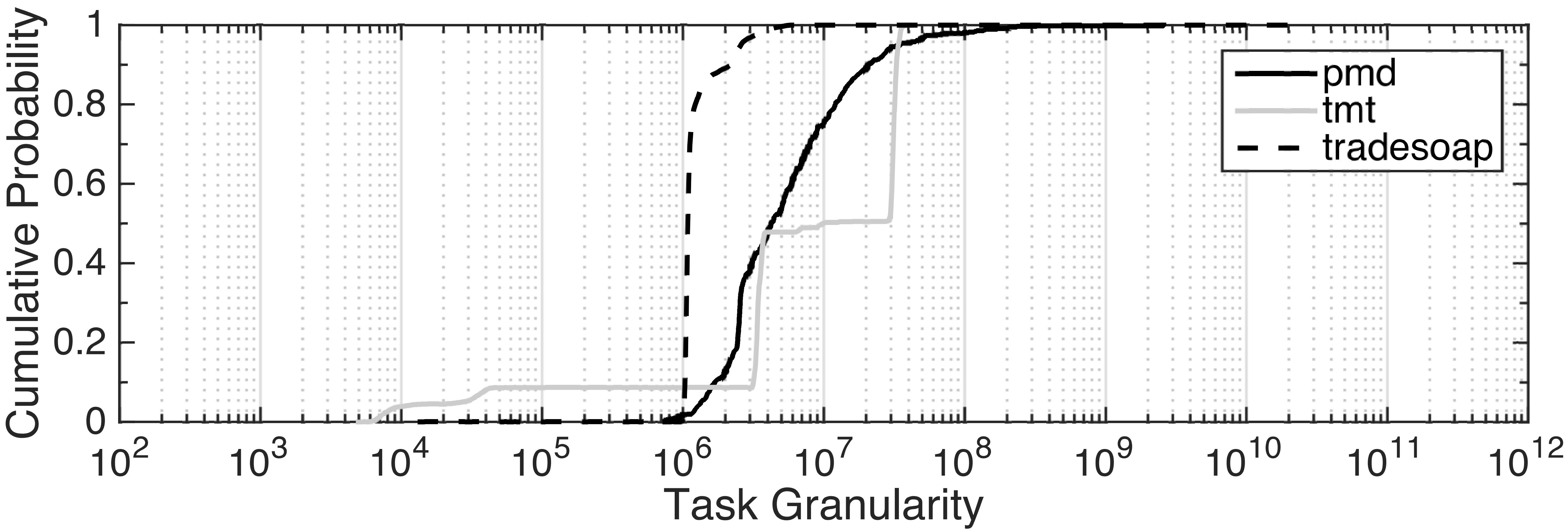
- Calling contexts: methods open in the call stack
  - Optional pass
  - Collected on a subset of problematic tasks
  - Collected at task creation, submission and execution
    - Often need modifications to optimize task granularity
  - Provide actionable profiles

Task Granularity Analysis

- Target: DaCapo and ScalaBench applications
- Input size: largest possible
- Focus only on steady-state [1]

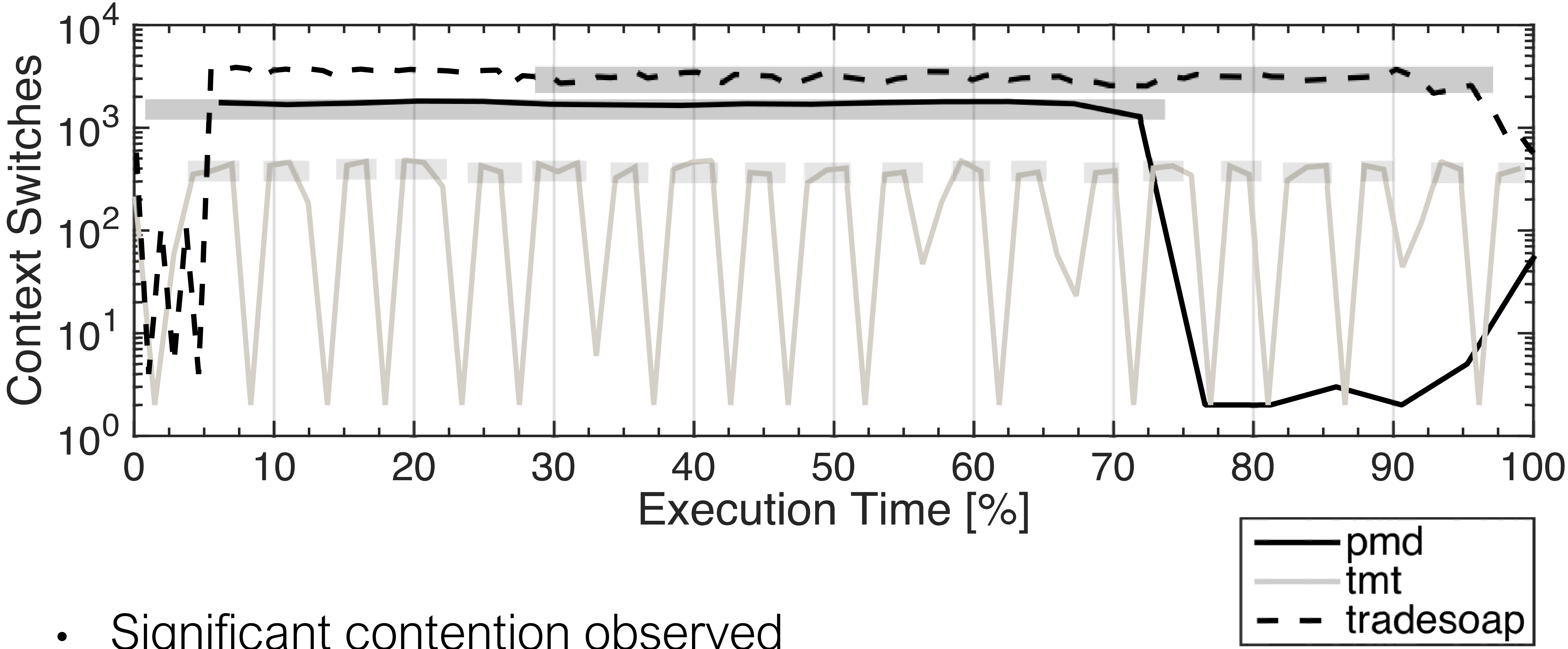
[1] Lengauer et al. *A Comprehensive Java Benchmark Study on Memory and Garbage Collection Behavior of DaCapo, DaCapo Scala, and SPECjvm2008*. ICPE'17.

Fine-Grained Tasks



- Large groups of tasks of same type and low granularity
  - `pmd`: 570 tasks
  - `tmt`: 16’184 tasks
  - `tradesoap`: 112’965 tasks

Fine-Grained Tasks

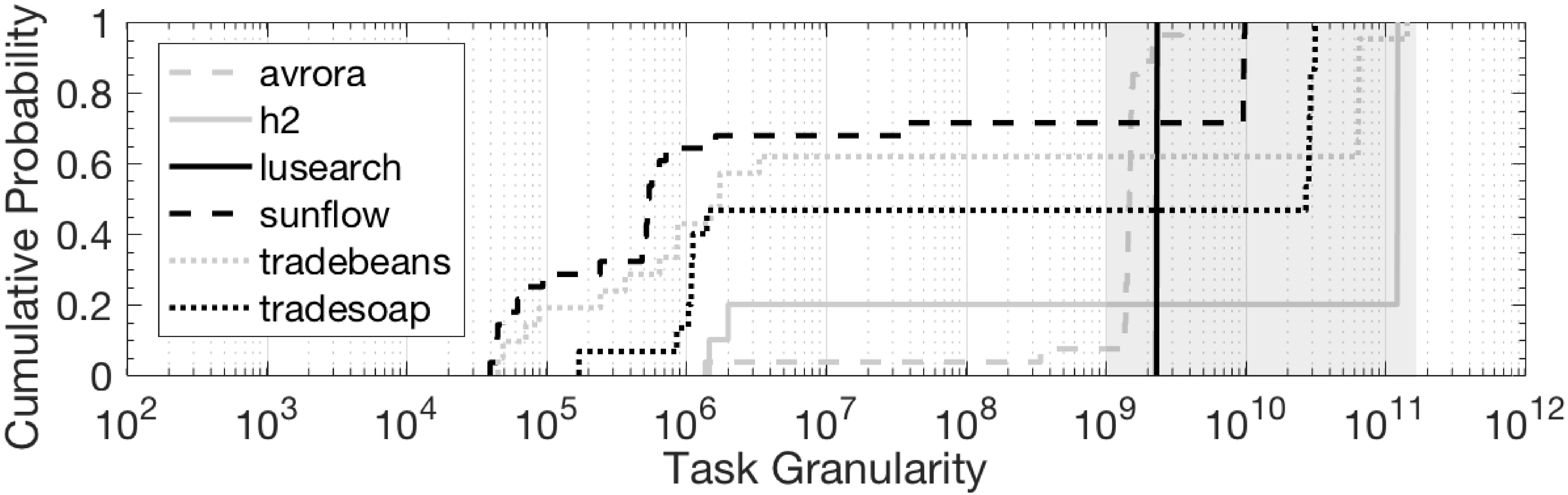


- Significant contention observed
  - `pmd` / `tmt`: only when fine-grained tasks are executed

Fine-Grained Tasks

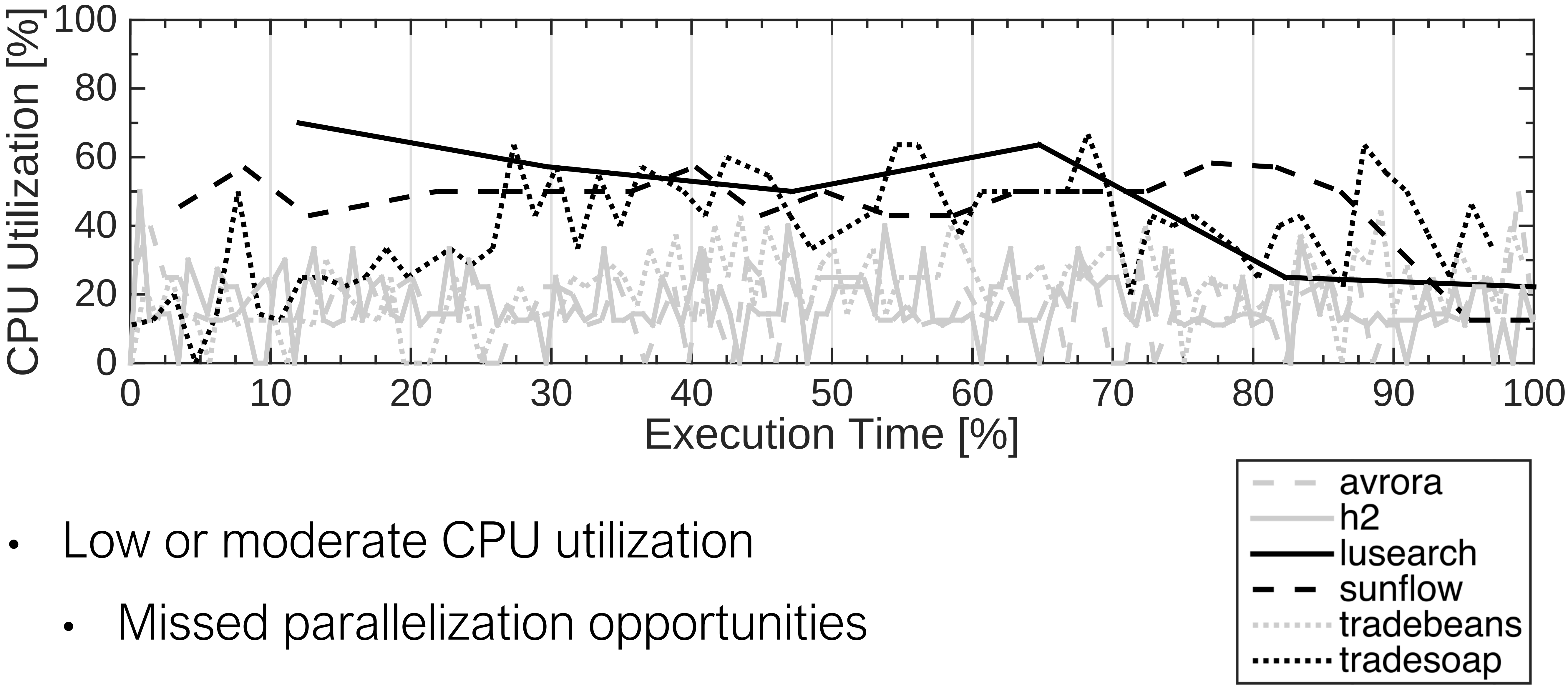
- `pmd` / `tmt`: presence of fine-grained tasks significantly interfering with each other
- Optimization: merge tasks
  - ✓ Reduce contention between tasks
  - ✓ Reduce creation and scheduling overheads

Coarse-Grained Tasks



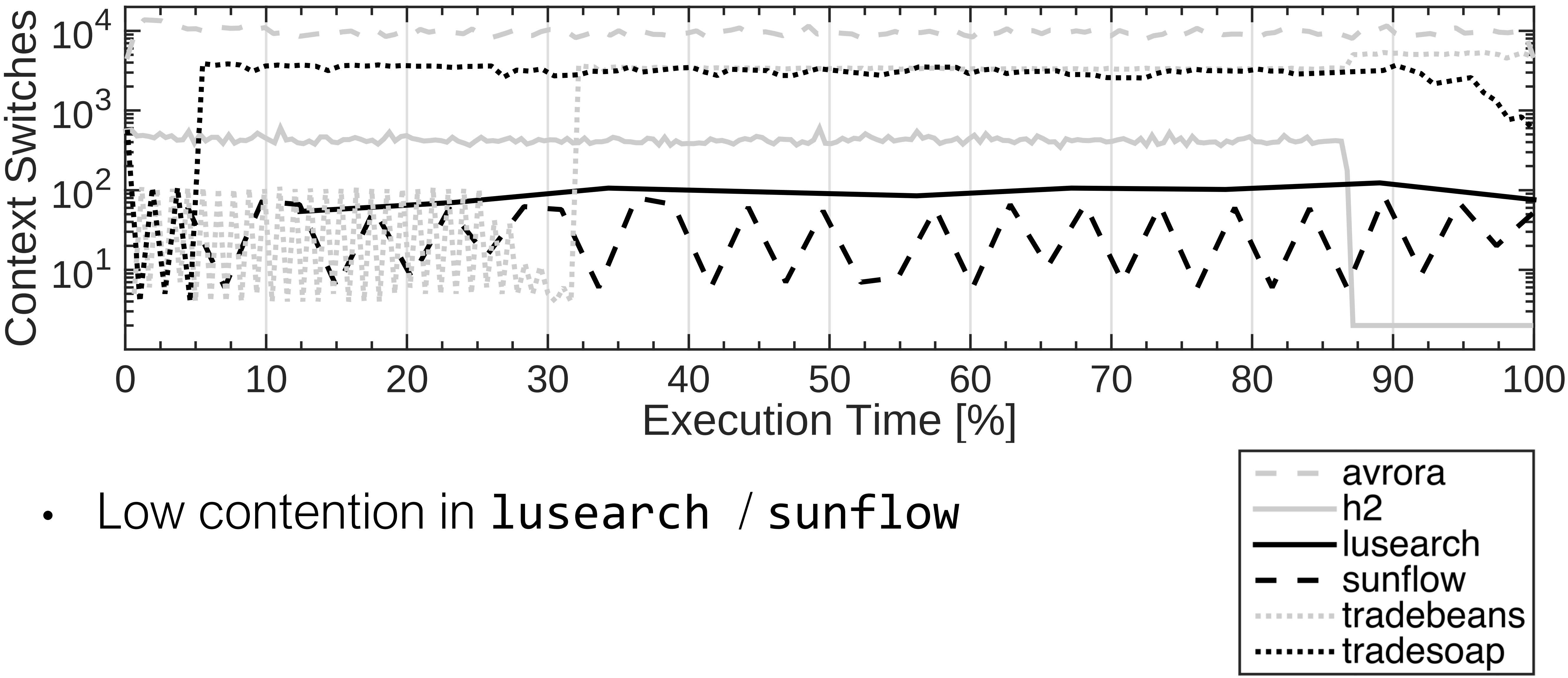
- Coarse-grained tasks in 6 benchmarks

Coarse-Grained Tasks



- Low or moderate CPU utilization
  - Missed parallelization opportunities

Coarse-Grained Tasks



- Low contention in `lusearch` / `sunflow`

Università
della
Svizzera
italiana

Coarse-Grained Tasks

- `lusearch` / `sunflow`: presence of optimizable coarse-grained tasks
    - Moderate CPU utilization and little interference
- Optimization: split tasks into smaller ones
    - ✓ Better utilize CPU

Università della Svizzera italiana

Task Granularity Optimization

- Target benchmarks: **`pmd`** and **`lusearch`**
  - Small modifications to task creation, submission and execution
  - Guided by actionable profiles

pmd (`PmdRunnable`)				**lusearch** (`QueryThread`)			
# tasks	**AVG task granularity**	**Speedup AVG**	**Speedup 95% confidence interval**	**# tasks**	**AVG task granularity**	**Speedup AVG**	**Speedup 95% confidence interval**
285	$3.1\cdot10^7$	1.2869	(1.2774, 1.2958)	16	$1.2\cdot10^9$	1.0992	(1.0823, 1.1170)
143	$6.3\cdot10^7$	1.3826	(1.3720, 1.3924)	32	$5.9\cdot10^8$	1.1031	(1.0849, 1.1219)
72	$1.2\cdot10^8$	1.4804	(1.4662, 1.4948)	64	$3.0\cdot10^8$	1.1270	(1.1074, 1.1468)
36	$2.4\cdot10^8$	1.5266	(1.5148, 1.5375)	128	$1.5\cdot10^8$	1.1190	(1.0976, 1.1399)
18	$4.9\cdot10^8$	1.5101	(1.4995, 1.5201)				
9	$9.9\cdot10^8$	1.4933	(1.4835, 1.5018)				

✓ Optimizing task granularity leads to significant performance improvements

Università
della
Svizzera
italiana

Overhead

Benchmark	Overhead	
	AVG	**95% confidence interval**
avrora	1.0155	(0.9959, 1.0374)
h2	1.0056	(0.9892, 1.0220)
lusearch	1.0033	(0.9883, 1.0184)
pmd	1.0646	(1.0439, 1.0847)
sunflow	1.0072	(0.9584, 1.0554)
tmt	1.0266	(0.9999, 1.0533)
tradebeans	1.0046	(0.9945, 1.0144)
tradesoap	1.0067	(1.0026, 1.0107)

- Limited profiling overhead
  - ✓ Low perturbation of the collected metrics

Conclusions

- We presented `tgp`, a task-granularity profiler for the JVM
- We analyzed task granularity in DaCapo and ScalaBench
- We revealed fine- and coarse-grained tasks causing performance drawbacks
- We optimized task granularity in `pmd` and `lusearch`
  - Speedups up to 1.53x (`pmd`) and 1.13x (`lusearch`)

Thank you for the attention

- Andrea Rosà

andrea.rosa@usi.ch
http://www.inf.usi.ch/phd/rosaa