A Comparison of the Influence of Different Multi-Core Processors on the Runtime Overhead for Application-Level Monitoring

Multicore Software Engineering, Performance, and Tools 2012

Jan Waller and Wilhelm Hasselbring — 31.05.12

Christian-Albrechts-University Kiel
Software Engineering Group
se.informatik.uni-kiel.de

research.spec.org

kieker-monitoring.net

Motivation

Multicore

- as a commodity
- adding more cores (Moore's Law)
- hard to exploit

Monitoring

- continuously running software systems
- minimize runtime overhead

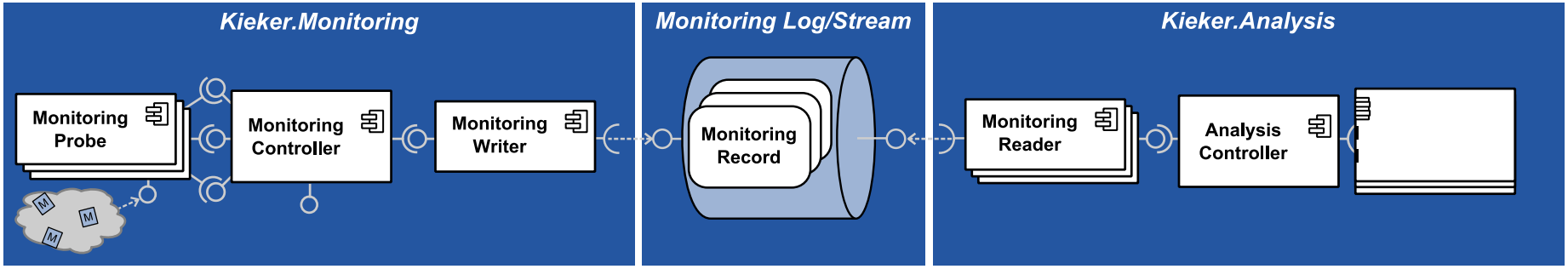
Exploit underutilized processing units to minimize the overhead.

Outline

1. Kieker Monitoring Framework
2. Portions of Monitoring Overhead
3. Micro-Benchmarks
4. Results of Benchmarks
5. Future Work and Outlook

Kieker Monitoring Framework

Christian-Albrechts-Universität zu Kiel



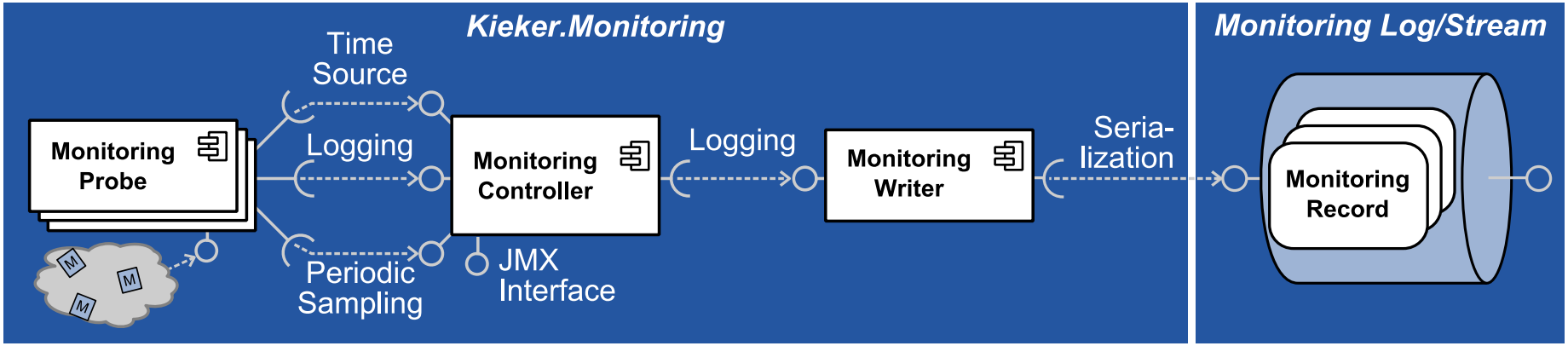
Extensible framework

- for Monitoring and Analyzing

  runtime behavior of concurrent or distributed software systems at the application level

- provides
  - software instrumentation
  - collection of information
  - logging of collected data
  - analysis/visualization of data

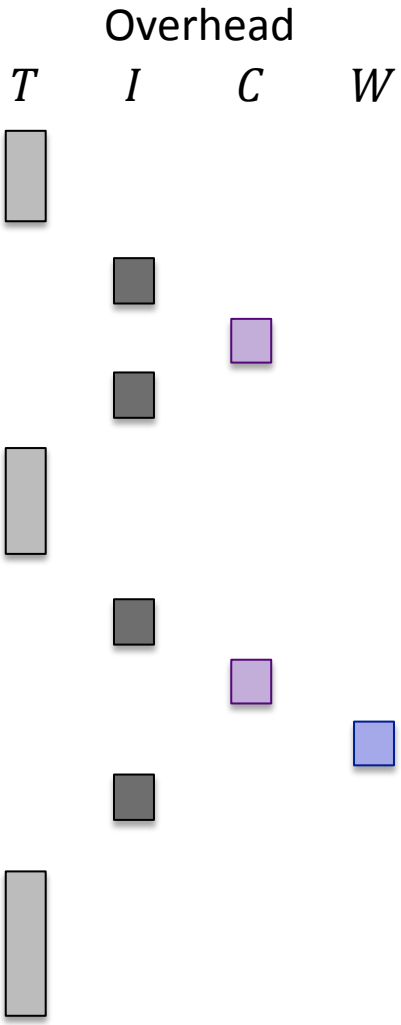
Kieker Monitoring Framework



Focus on the *Kieker.Monitoring* component:

1. Instrumentation
2. Data Collection
3. Writing

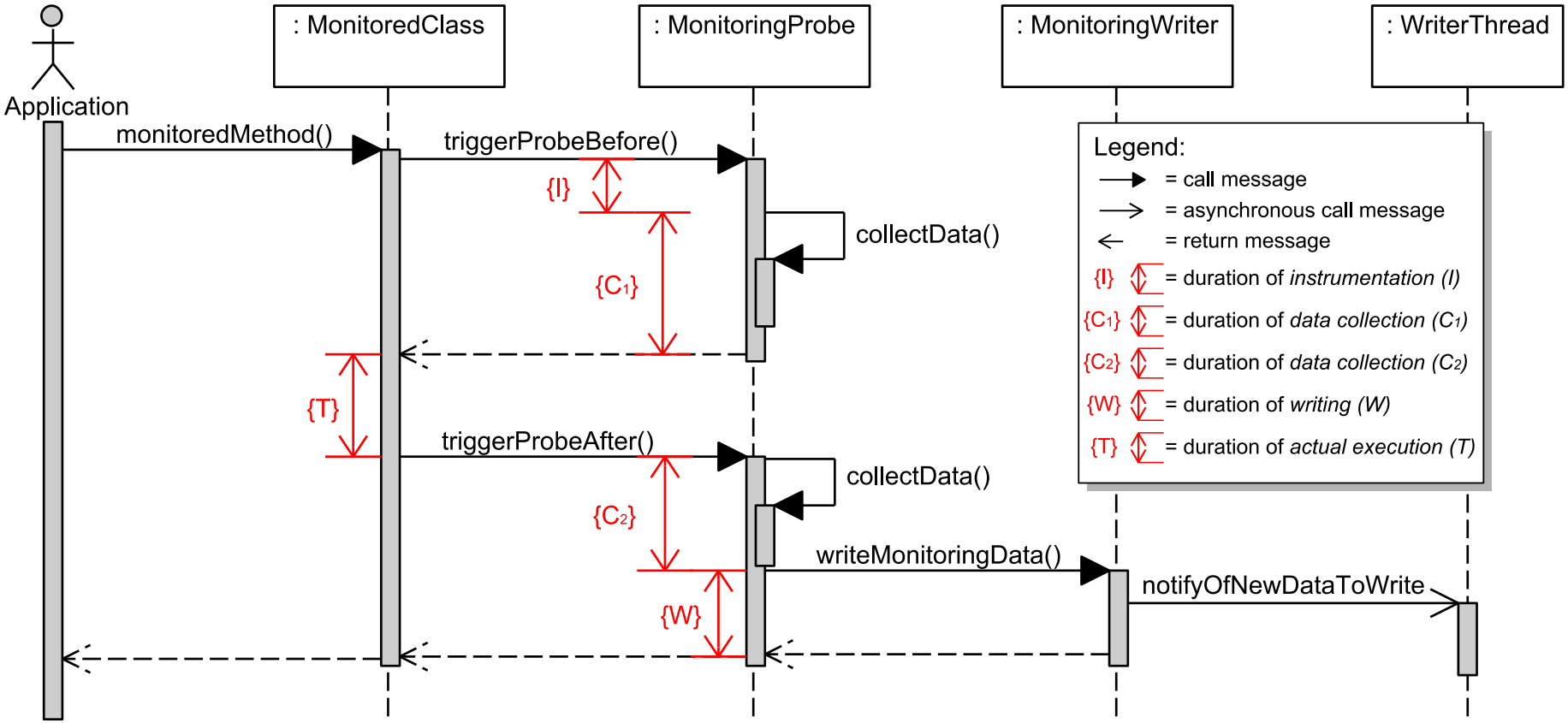
Monitoring Overhead



(Simplified) source code of a monitored method

```
public boolean monitoredMethod() {

  if (!isMonitoringEnabled()) {
    collectDataBefore();
  }

  // normal method code

  if (!isMonitoringEnabled()) {
    collectDataAfter();
    writeMonitoringData();
  }

  return retVal;
}
```

Monitoring Overhead



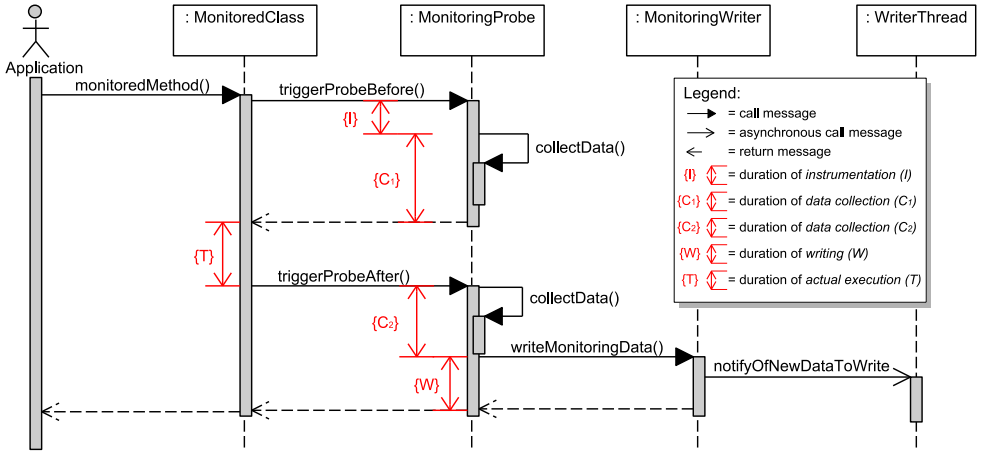
Execution time: T

Overhead: $I + C + W$ with $C = C_1 + C_2$

Monitoring Overhead

3 Portions of Overhead

$$I + C + W$$



Determine each portion (one at a time):

1. Determine T in the benchmark system
2. Add instrumentation I
3. Add data collection C
4. Add writing W

Outline

1. Kieker Monitoring Framework
2. Portions of Monitoring Overhead
3. Micro-Benchmarks
4. Results of Benchmarks
5. Future Work and Outlook

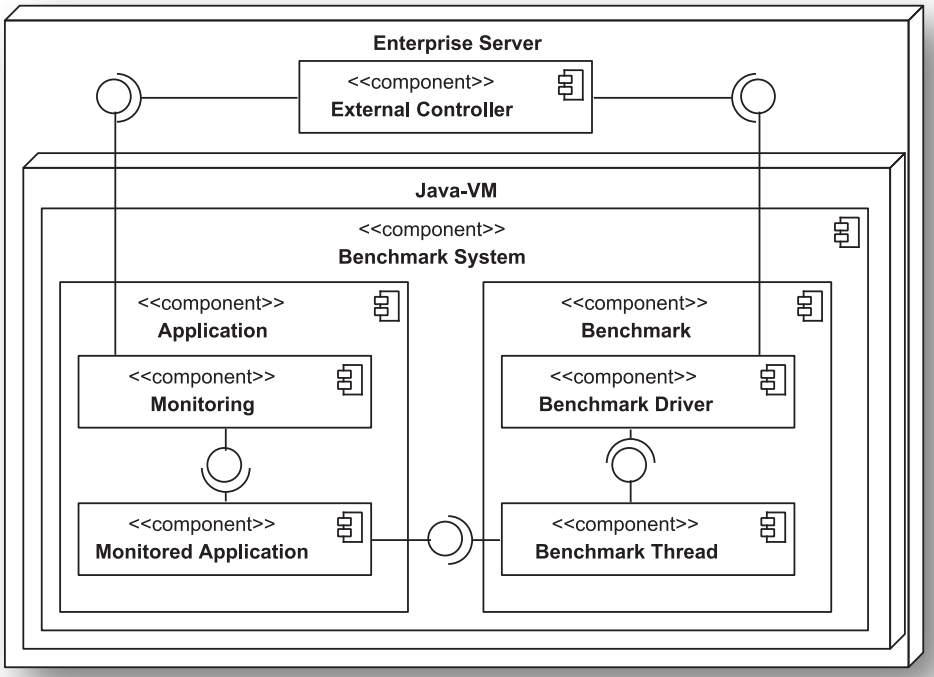
Micro-Benchmarks

Benchmark designed to measure individual portions

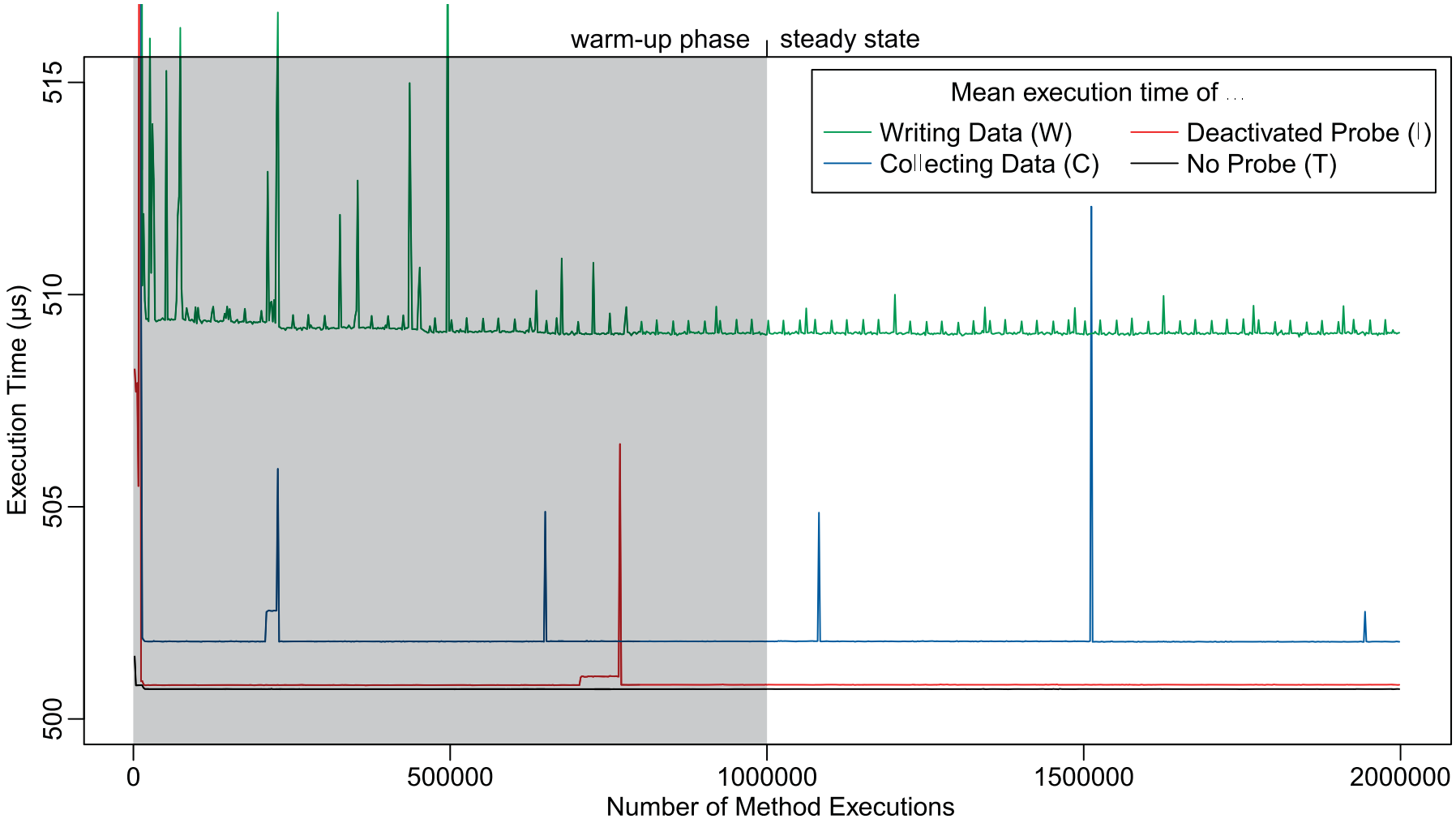
- *External Controller* configures *Monitoring* and *Driver*
- *Monitored Application* provides fixed T
- *Benchmark Threads*

  call *monitored method*
  - #totalCalls
  - #recodedCalls
- Run 4 times to measure

complete source code available at:
http://kieker-monitoring.net

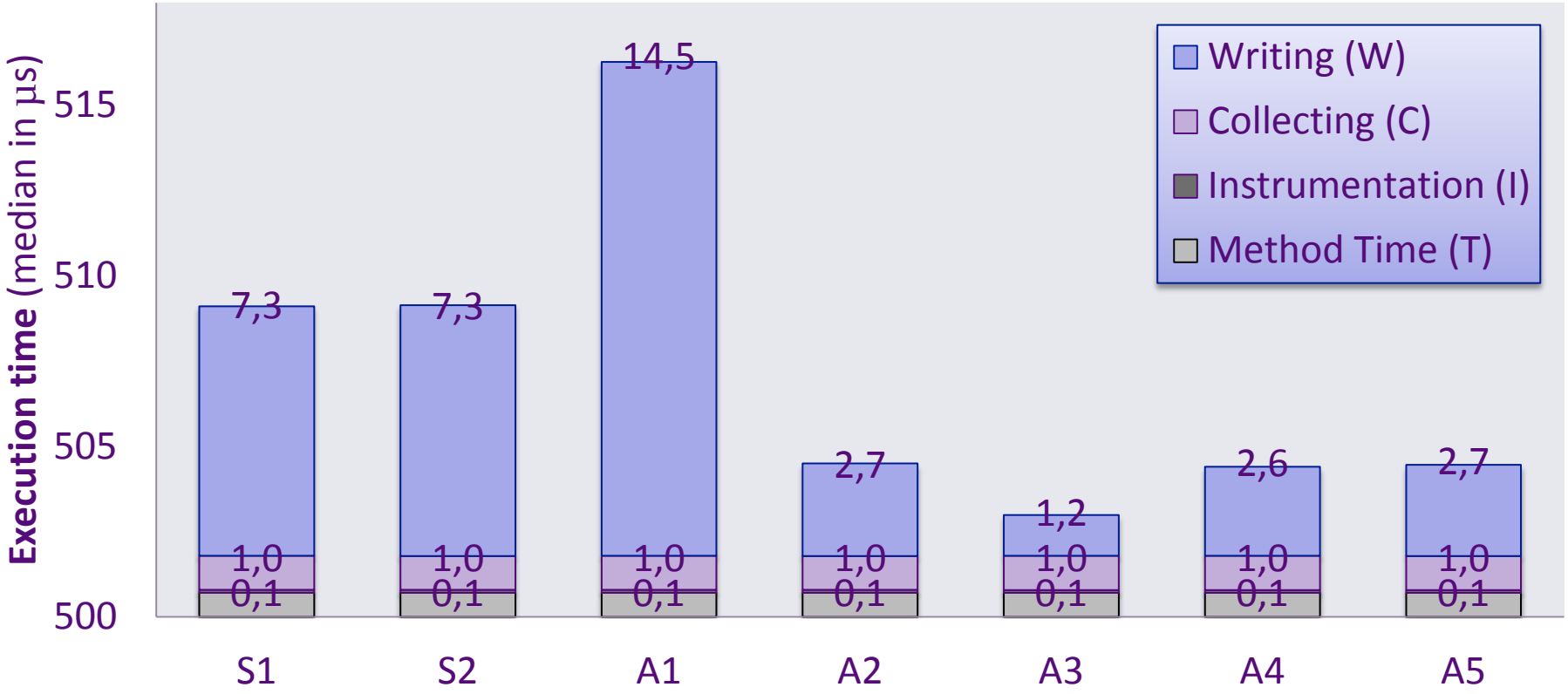


Benchmark – Warm-up Phase



Single Threaded Benchmark

Christian-Albrechts-Universität zu Kiel

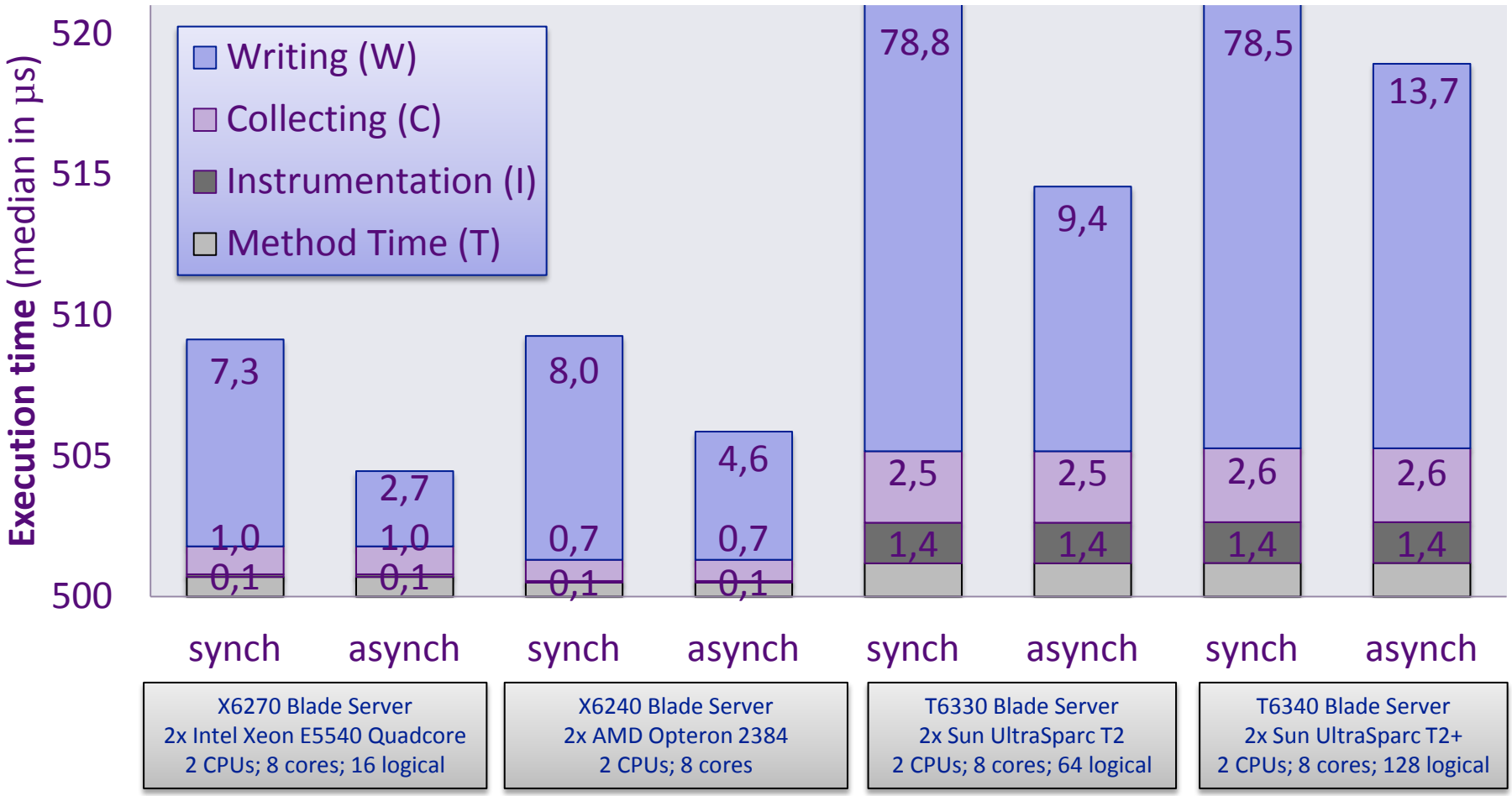


Exp	Writer	Cores	Notes
S1	SyncFS	1	single physical core
S2	SyncFS	2	two logical cores on the same physical core

Exp	Writer	Cores	Notes
A1	AsyncFS	1	single physical core
A2	AsyncFS	2	two logical cores on the same physical core
A3	AsyncFS	2	two physical cores on the same processor
A4	AsyncFS	2	two physical cores on different processors
A5	AsyncFS	16	whole system is available

X6270 Blade Server
2x Intel Xeon 2.53 GHz E5540 Quadcore / 24 GB RAM
Solaris 10 / Oracle Java x64 Server VM 1.6.0_26 (1 GB heap)

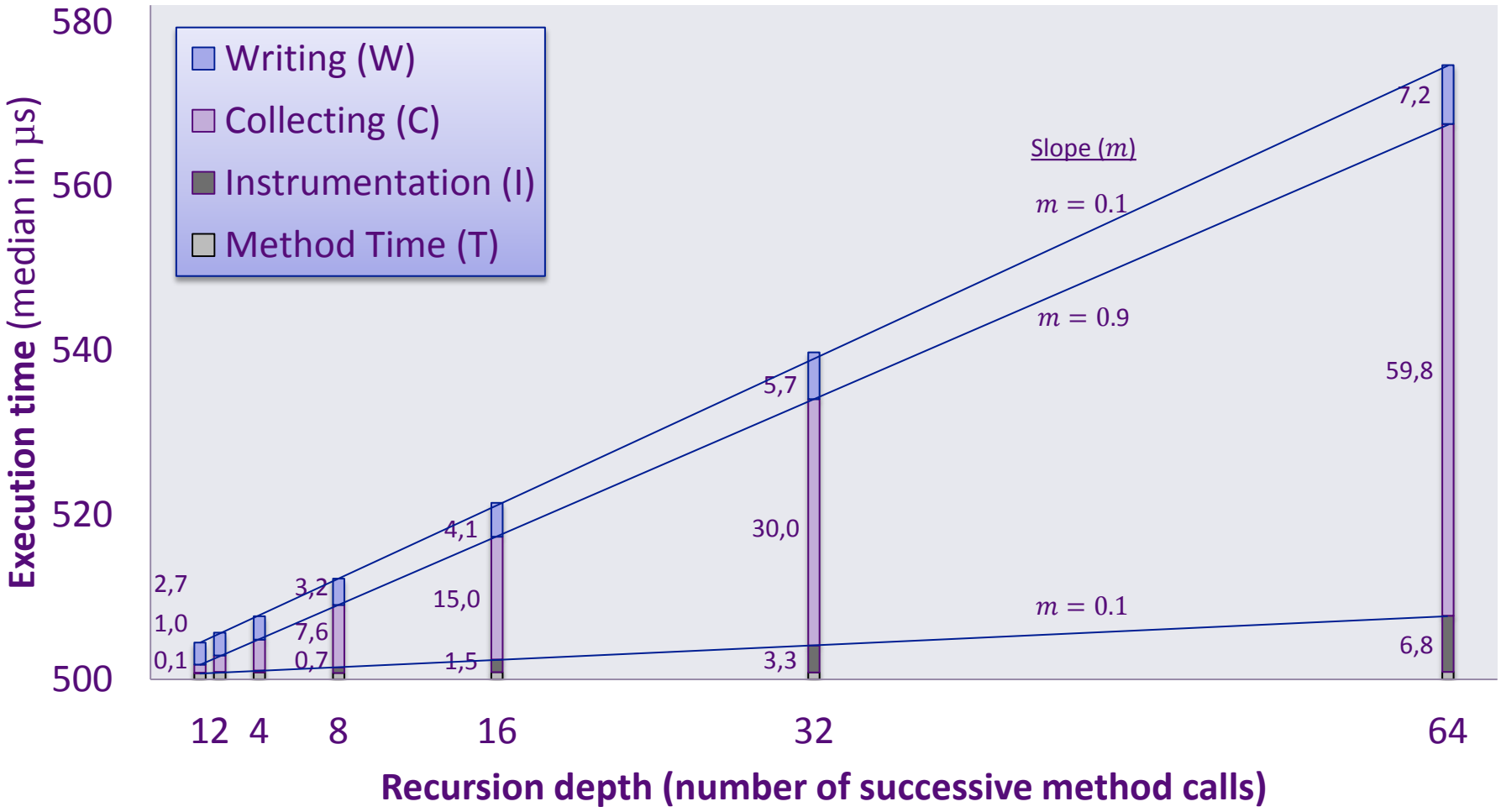
Multi-Core Architectures



Experiment similar to:

S2	SyncFS writer	all cores	whole system is available
A5	AsyncFS writer	all cores	whole system is available

Linear Rise of Overhead



Experiment similar to:

A5	AsyncFS writer	16 cores	whole system is available

Outline

1. Kieker Monitoring Framework
2. Portions of Monitoring Overhead
3. Micro-Benchmarks
4. Results of Benchmarks
5. Future Work and Outlook

Future Work

- Publish results on multi-threaded benchmarks
  - #threads < cores
  - #threads = cores – 1
  - #threads = cores
  - #threads > cores
- Perform macro-benchmark evaluations
  - e.g., SPECjEnterprise2010, …
- Compare results of other monitoring frameworks
- Establish a benchmark suite in the community

Performance breakpoints
#threads = physical cores – 1
#threads = logical cores – 1

Thank you for your attention!

Christian-Albrechts-Universität zu Kiel

http://kieker-monitoring.net

Kieker is distributed as open-source software (Apache License, V. 2.0)

Kieker is distributed as part of SPEC® RG's repository of peer-reviewed tools for quantitative system evaluation and analysis.
http://research.spec.org/projects/tools.html

Further Reading

- A. van Hoorn, J. Waller, and W. Hasselbring. *Kieker: A Framework for Application Performance Monitoring and Dynamic Software Analysis.* Proc. 3rd ACM/SPEC Int. Conf. Perform. Eng. (ICPE '12), ACM, 2012
- A. van Hoorn, M. Rohr, W. Hasselbring, J. Waller, J. Ehlers, S. Frey, and D. Kieselhorst. *Continuous Monitoring of Software Services: Design and Application of the Kieker Framework.* Technical report TR-0921, Department of Computer Science, University of Kiel, Germany, 2009

Backup Slides

Micro-Benchmarks


```
public long monitoredMethod(final long methodTime) {
    final long exitTime = System.nanoTime() + methodTime;
    long currentTime = System.nanoTime();
    while (currentTime < exitTime) {
        currentTime = System.nanoTime();
    }
    return currentTime;
}

for (int i = 0; i < totalCalls; i++) {
  start_ns = System.nanoTime();
  mc.monitoredMethod(methodTime);
  stop_ns = System.nanoTime();
  timings[j] = stop_ns - start_ns;
  j = (j + 1) % recordedCalls;
}
```

(simplified) source code of the micro-benchmark
included in the Kieker releases (http://kieker-monitoring.net)

Monitoring Log Size

each Experiment:

- 20 minutes / configuration
- 2,000,000 monitored calls
- 362 MB Kieker monitoring log files

Recursion depth 64:

- 23 GB monitoring log files
- 19.3 MB/s written to disc