Profile Caching for the Java Virtual Machine

Marcel Mohler, ETH Zurich

Bachelor Thesis

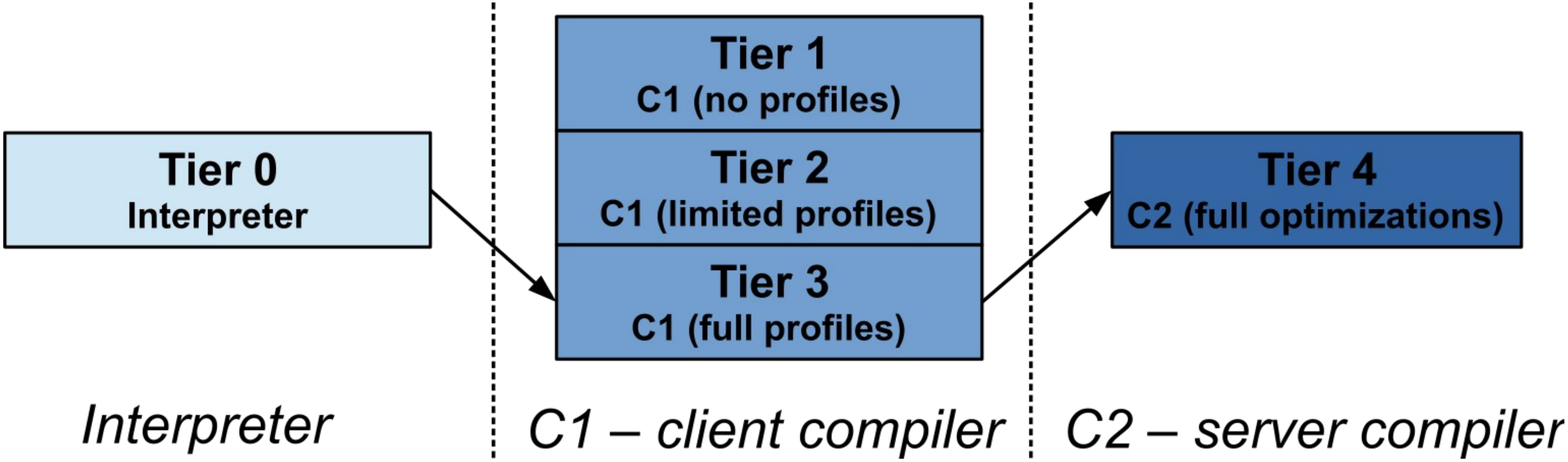
Supervisors: Zoltan Majo, Oracle
Tobias Hartmann, Oracle

Prof. Thomas Gross, Laboratory for Software Technology

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

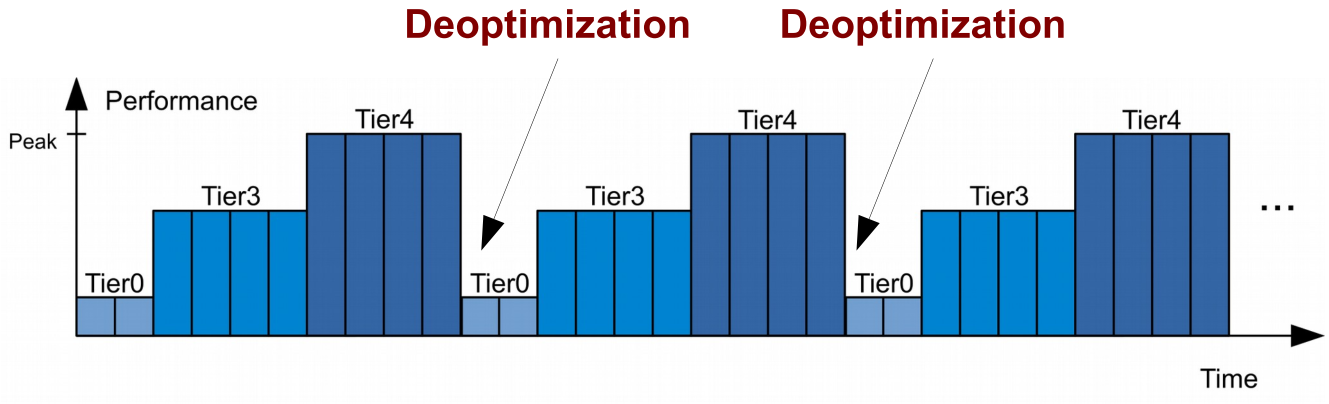
LST
Laboratory for Software Technology

Hotspot™: Tiered Compilation



- JVM gatheres profiles
- Uses these profiles for code optimizations

Problem

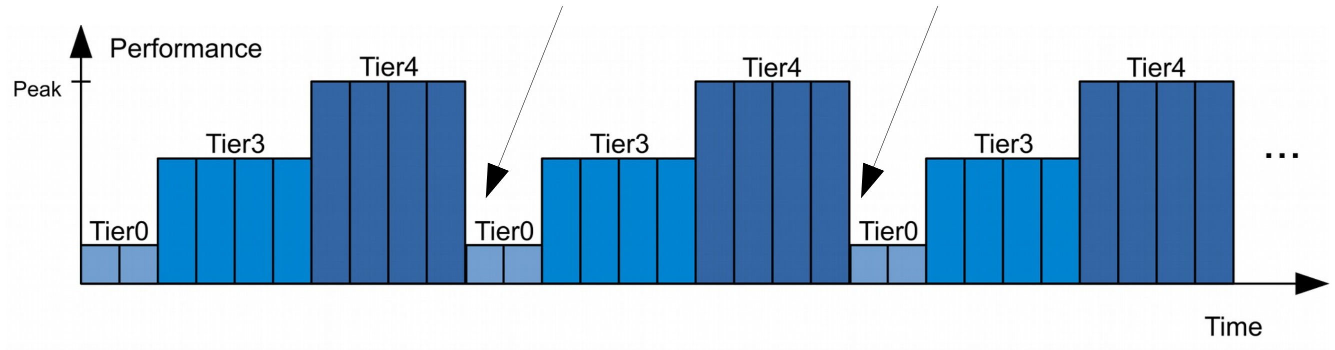


(a) default tiered compilation

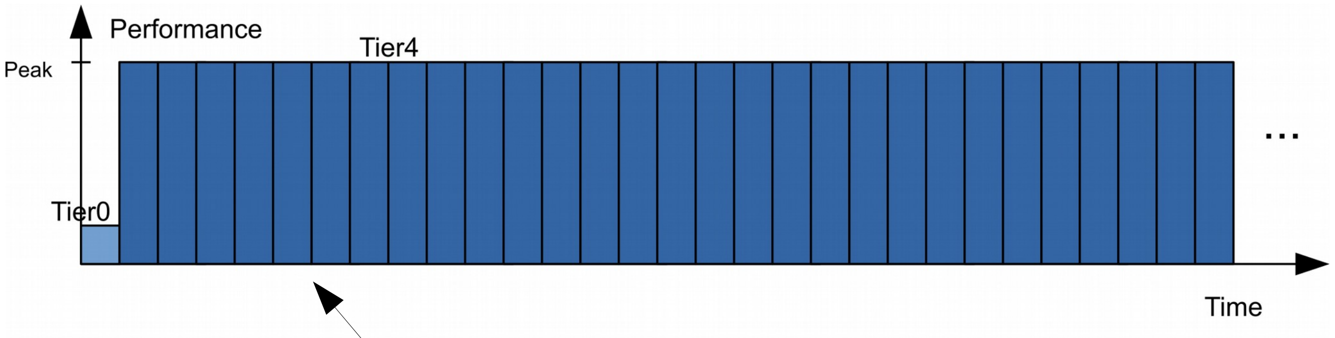
Idea

Deoptimization

Deoptimization



(a) default tiered compilation



(b) with cached profiles

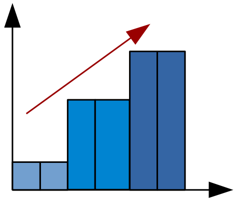
No deoptimization

Goals

Decrease performance fluctuations

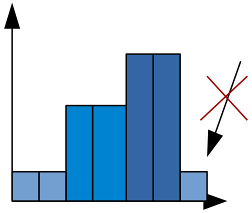
- Faster method performance warmup

  → reach peak performance quicker



- Less deoptimizations

  → stay on peak performance

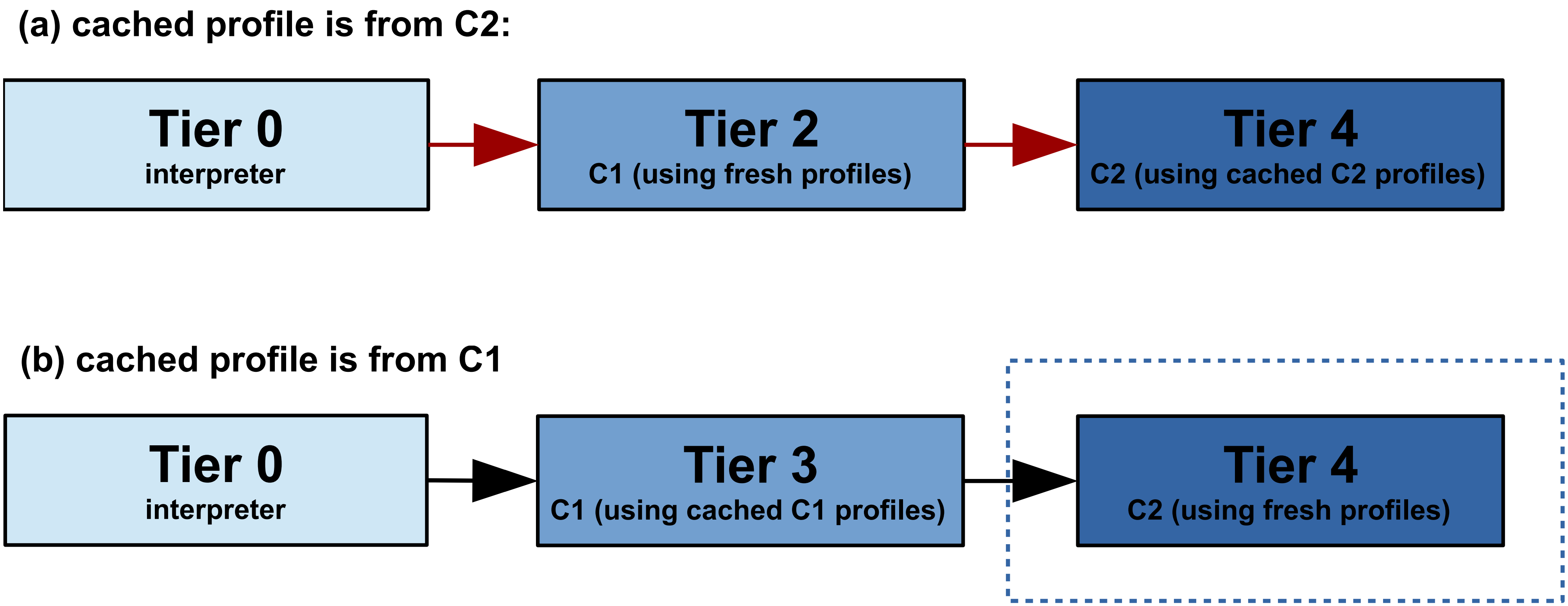


Design: dump profiles

- 1 run of JVM where profiles get dumped to disk

- Store method metadata, profiles and compile information of C3 & C4 compilations

Design: use profiles



(a) cached profile is from C2:

Tier 0
interpreter

Tier 2
C1 (using fresh profiles)

Tier 4
C2 (using cached C2 profiles)

(b) cached profile is from C1

Tier 0
interpreter

Tier 3
C1 (using cached C1 profiles)

Tier 4
C2 (using fresh profiles)

Implementation

- 1846 lines of code
- 24 files affected
- 2 new classes

  - ciCacheProfiles

  - ciCacheProfilesBroker

Evaluation

- ETH Data Center Observatory
- Focus on warmup, not overall performance
- 2 benchmark suites

  - SPECjvm 2008

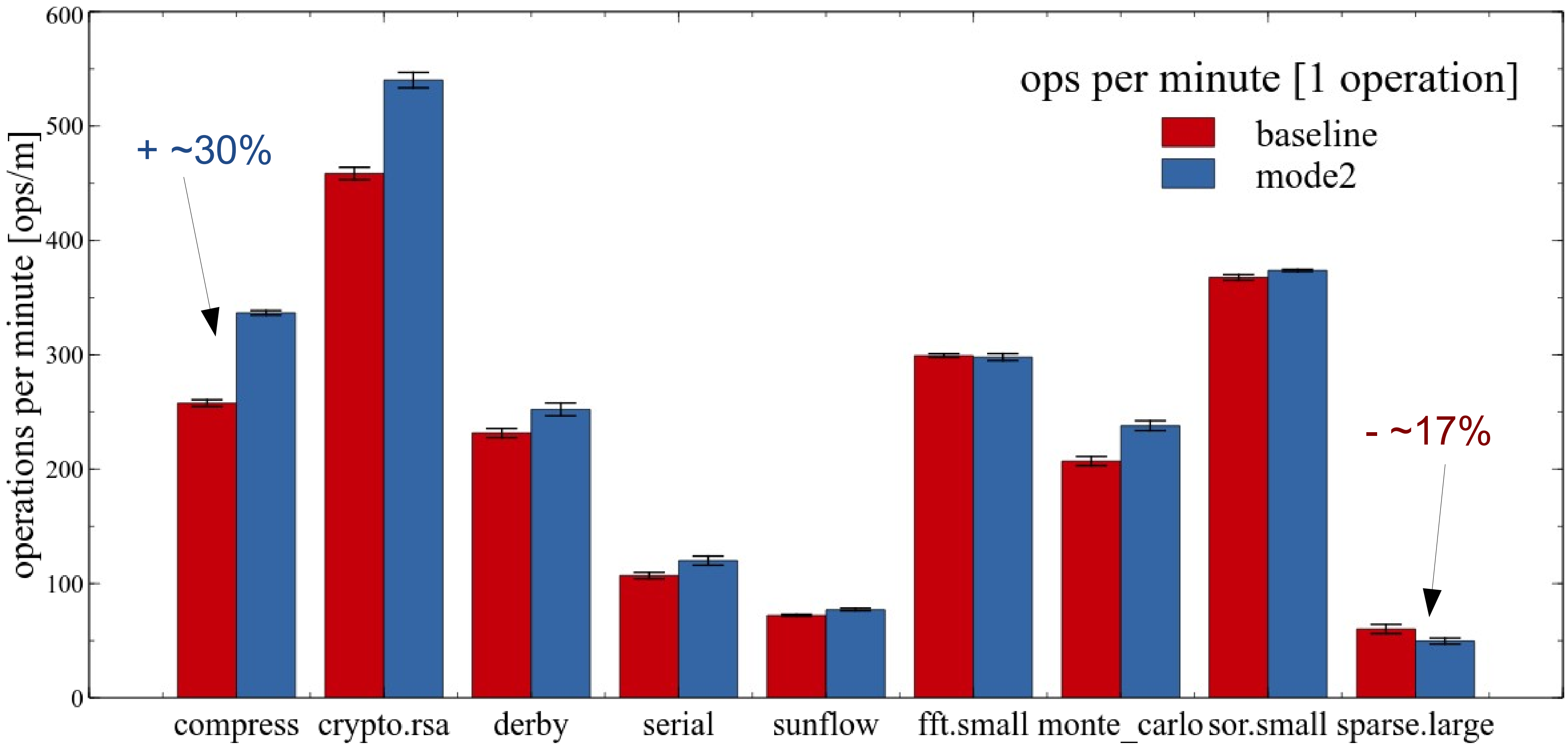
    17 individual benchmarks

  - Google Octane (using Nashorn),

    16 individual benchmarks

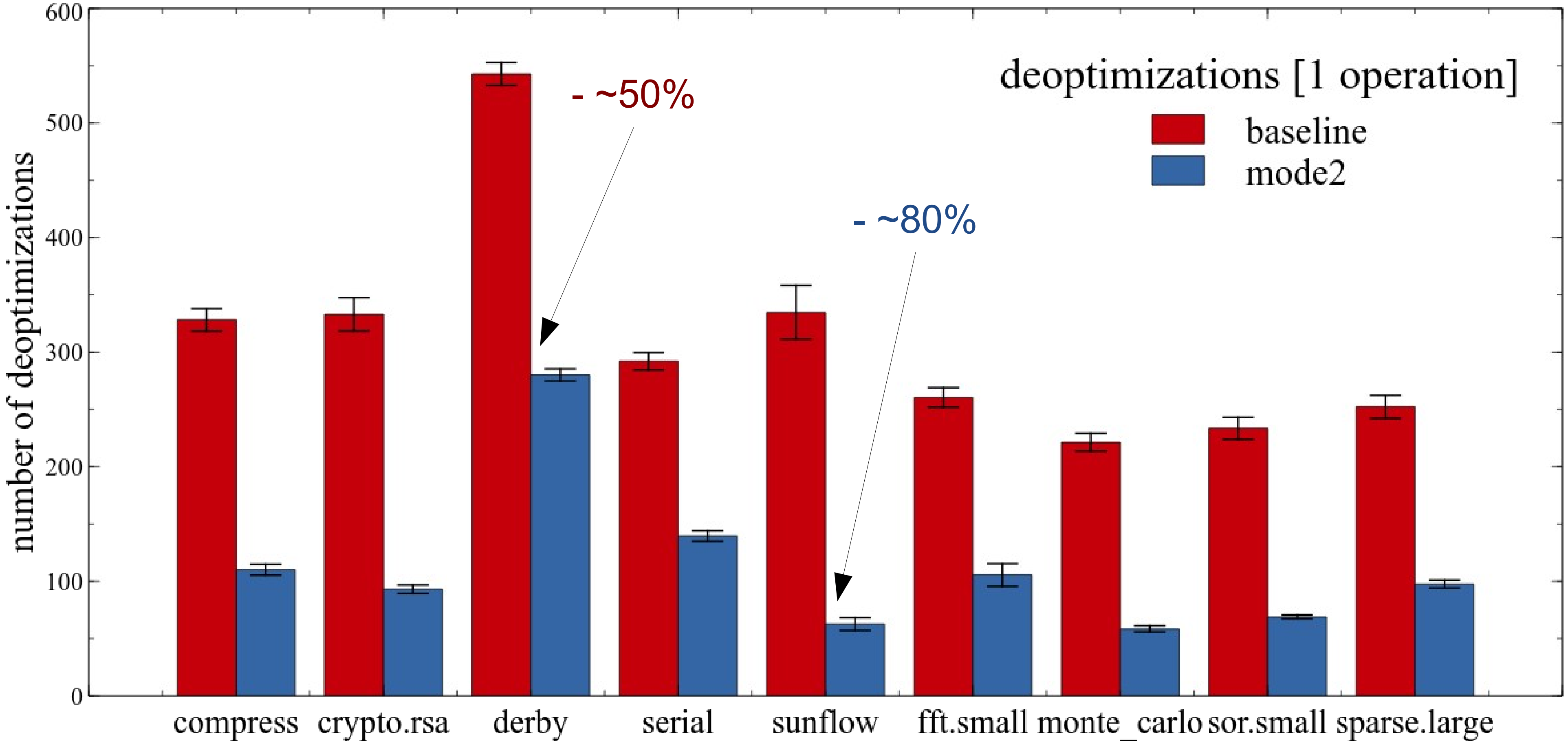
Performance evaluation

- Performance (higher is better)



Performance evaluation

- Deoptimizations (lower is better)

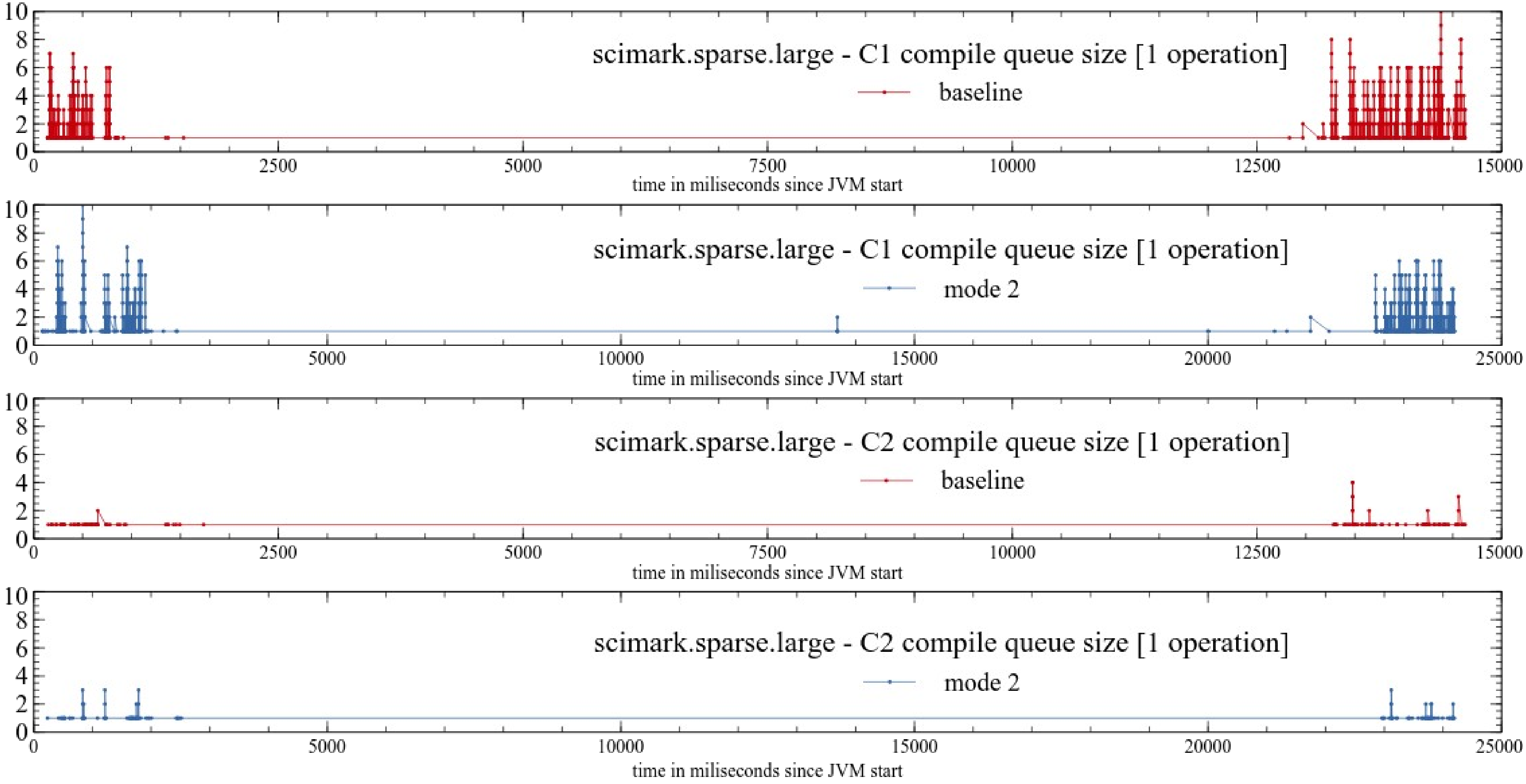


- ~50%

- ~80%

Performance evaluation

- Compilation queue



Other benchmark results

- Disabling intrinsics does not influence performance
- Benefit mainly from C2 compilations. Disabling C1 profiles does not affect performance significantly
- Around 70% of the compilations use profiles

Other approaches

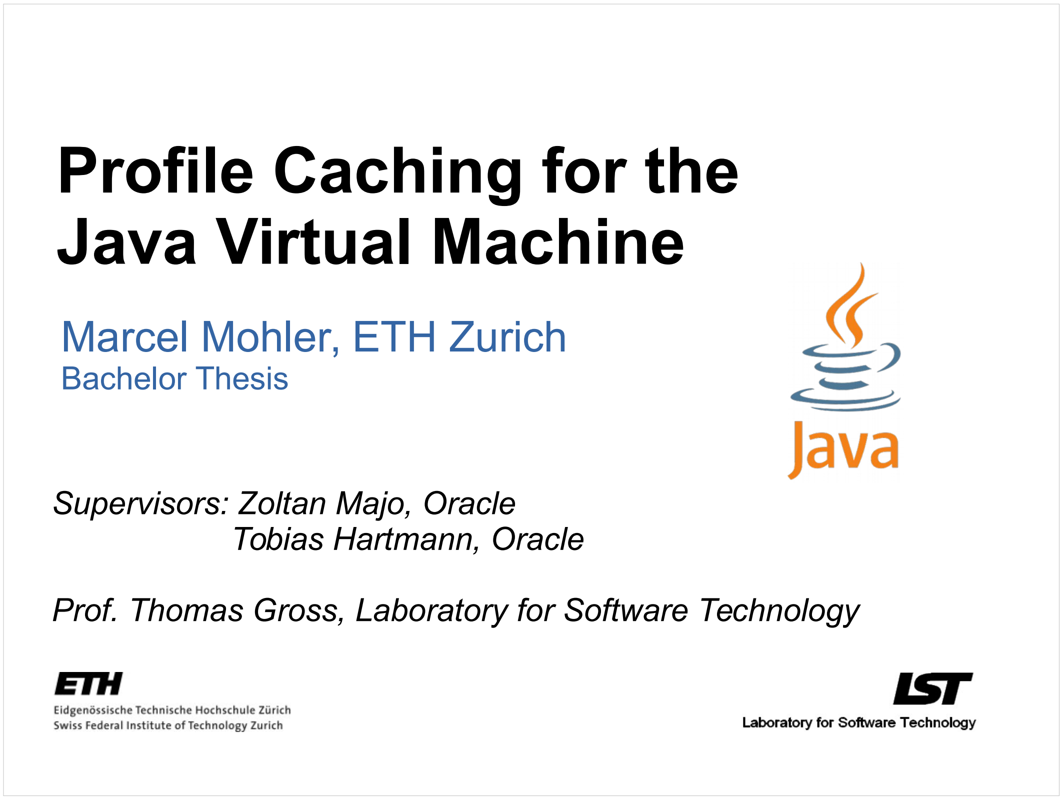
- Presented: Mode2

- Mode 0: skip C1 & lower compilation thresholds
- Mode 1: skip C1 & keep original compilation thresholds

Conclusion

- Complex system
- Reasons for performance influence difficult to measure
- Cached profiles *can* greatly improve warmup performance if used properly
- System requires manual configuration

Thank you for listening



Profile Caching for the Java Virtual Machine

Marcel Mohler, ETH Zurich
Bachelor Thesis

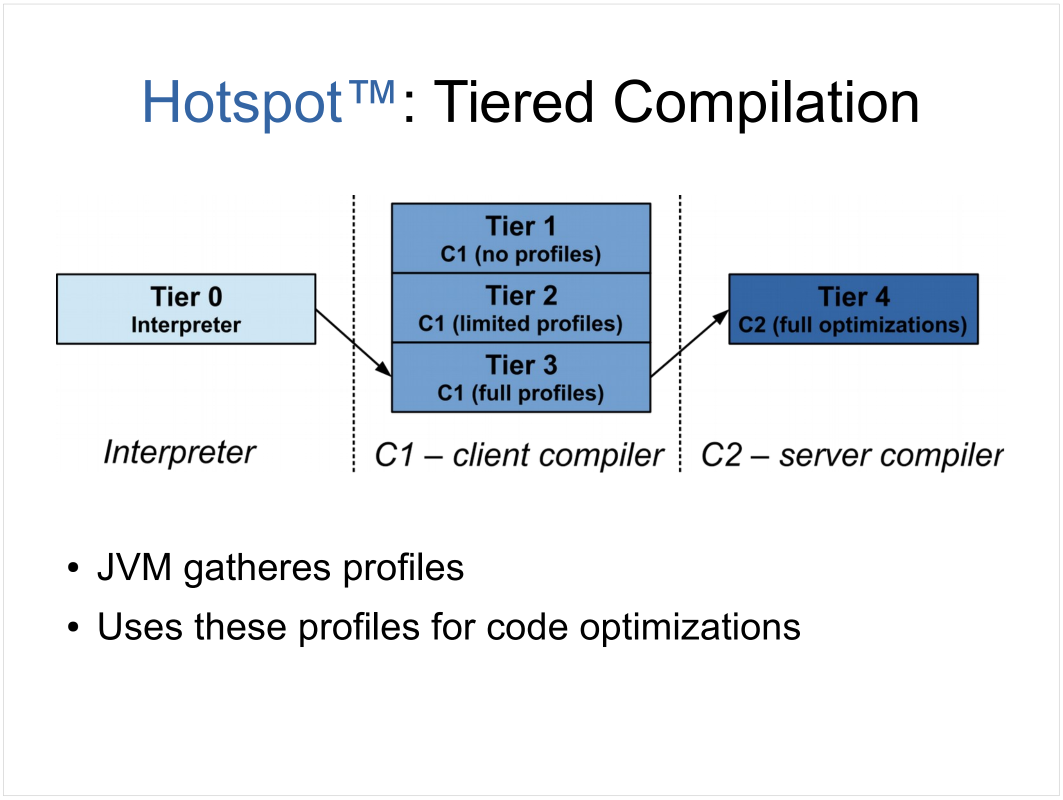
Supervisors: Zoltan Majo, Oracle
Tobias Hartmann, Oracle

Prof. Thomas Gross, Laboratory for Software Technology

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

LST
Laboratory for Software Technology

Welcome



Hotspot™: Tiered Compilation

Tier 0
Interpreter

Tier 1
C1 (no profiles)

Tier 2
C1 (limited profiles)

Tier 3
C1 (full profiles)

Tier 4
C2 (full optimizations)

Interpreter

C1 – client compiler

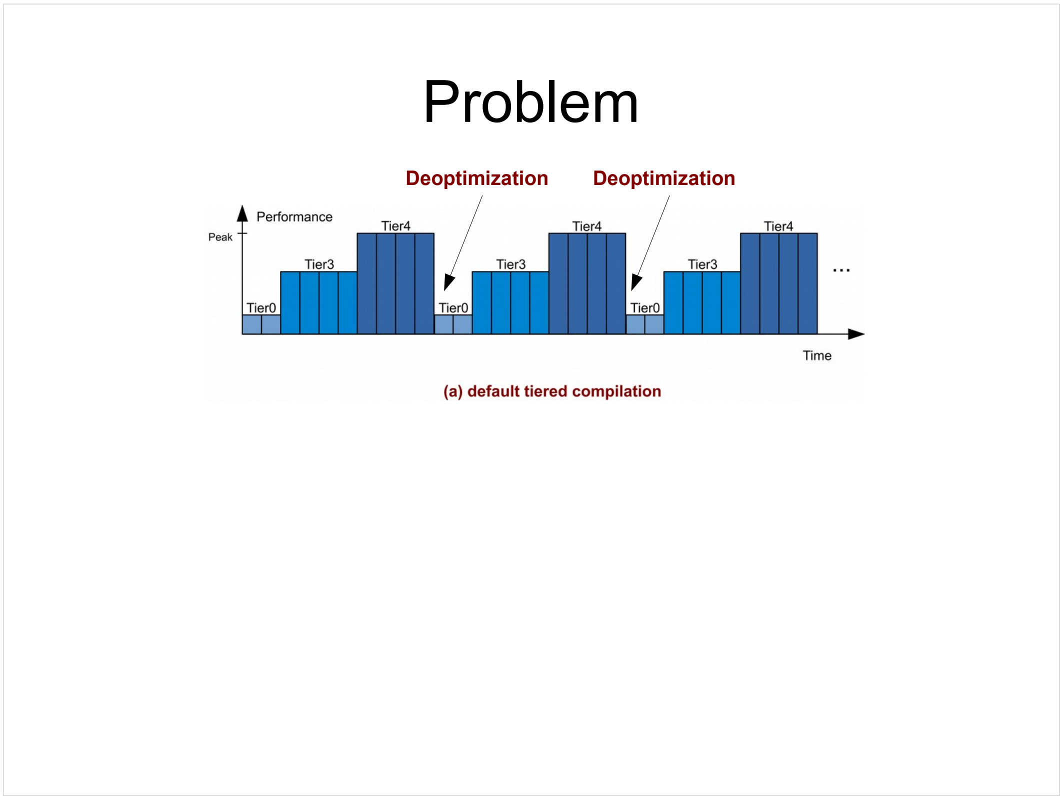
C2 – server compiler

- JVM gatheres profiles
- Uses these profiles for code optimizations

First off, a few things about the Just-in-time compiler of Hotspot

Hotspot is a JVM maintained by Oracle and heavily used all around the world in millions of devices

Profiles crucial to the performance of the code



Problem

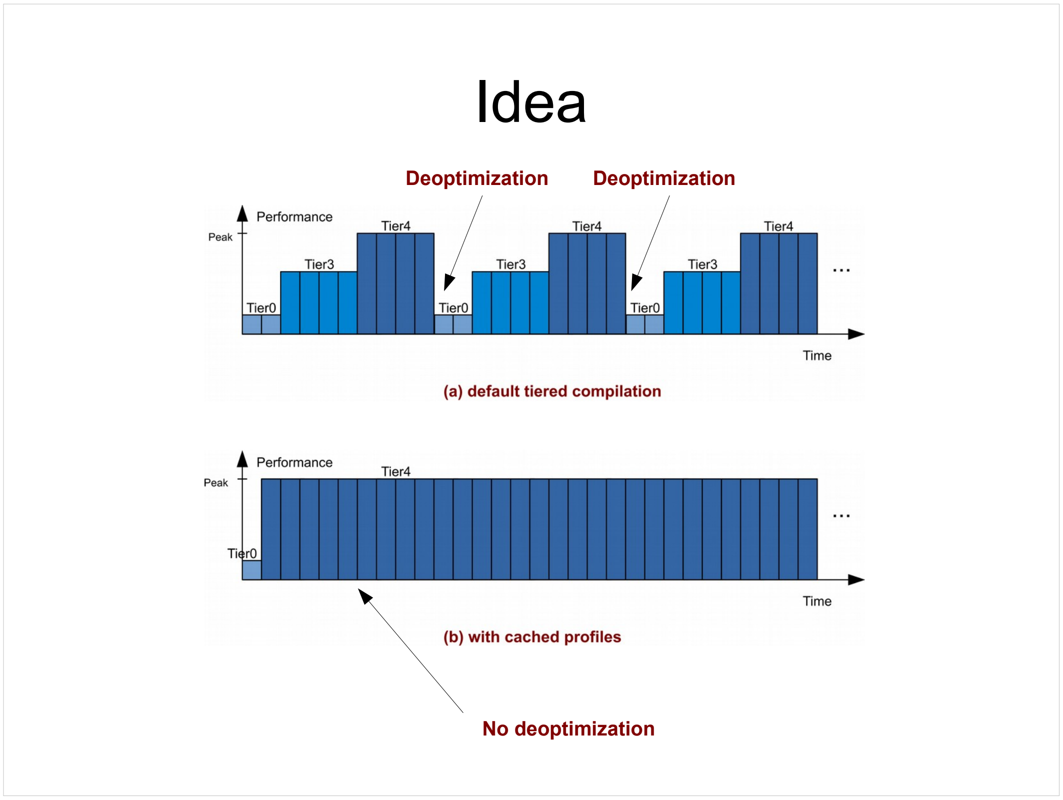
(a) default tiered compilation

- What is the current problem?

- Methods start compiling at Tier0 and have to go through multiple tiers until reaching peak performance

- In case of deoptimizations we start from the beginning again

- Ideally we want this



Idea

Deoptimization

Deoptimization

(a) default tiered compilation

(b) with cached profiles

No deoptimization

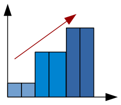
- Don't rely on the freshly gathered profiles
- Use profiles created in a previous run of the same program
- Allows the jvm to reach peak performance more quickly

Goals

Decreace performance fluctuations

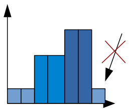
- Faster method performance warmup

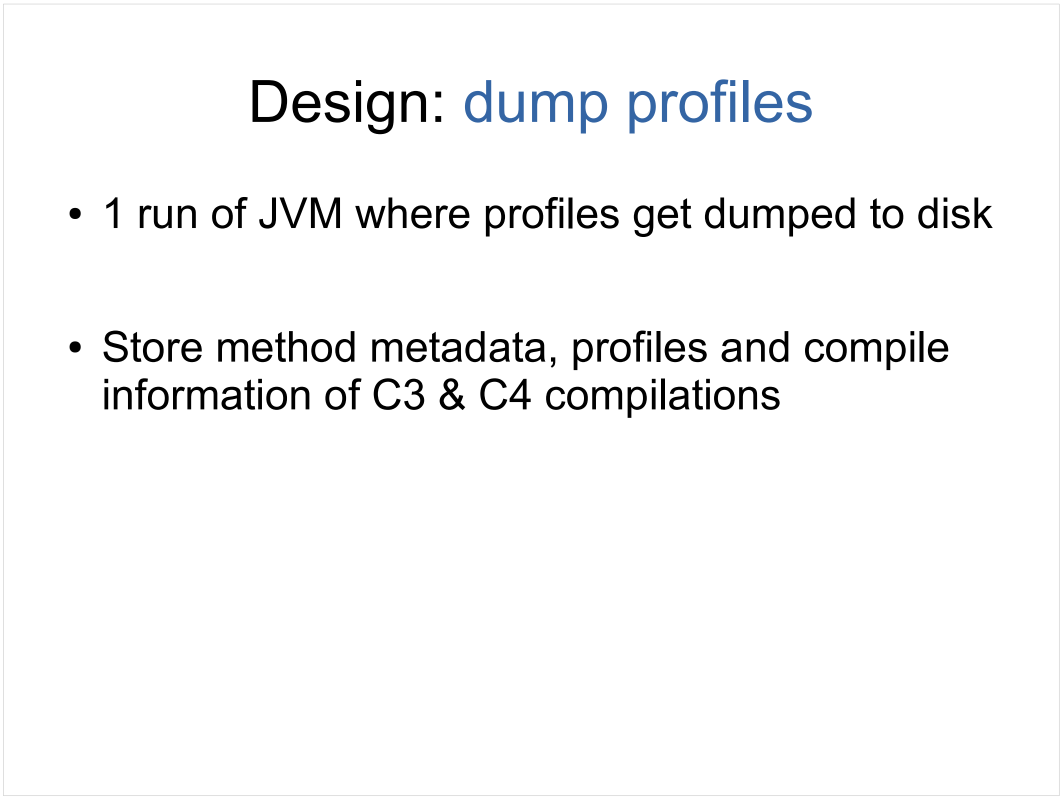
  → reach peak performance quicker



- Less deoptimizations

  → stay on peak performance



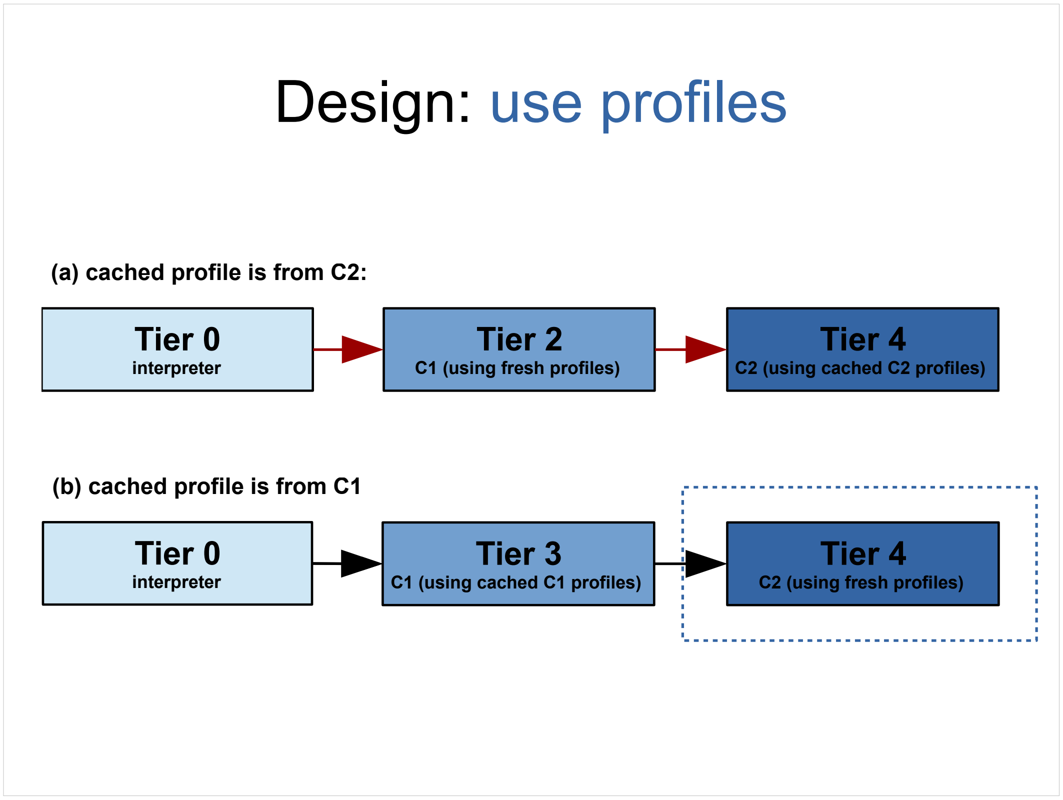


Design: dump profiles

- 1 run of JVM where profiles get dumped to disk
- Store method metadata, profiles and compile information of C3 & C4 compilations

Takes two steps

First dump the profiles



Design: use profiles

(a) cached profile is from C2:

Tier 0
interpreter

Tier 2
C1 (using fresh profiles)

Tier 4
C2 (using cached C2 profiles)

(b) cached profile is from C1

Tier 0
interpreter

Tier 3
C1 (using cached C1 profiles)

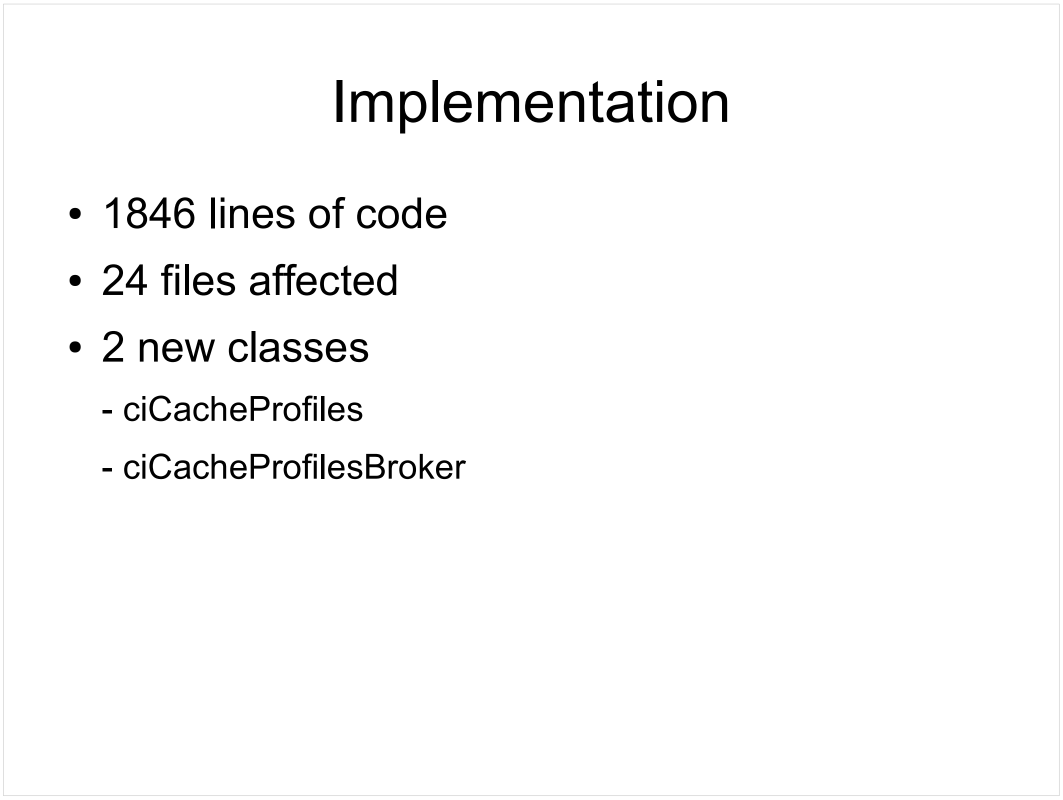
Tier 4
C2 (using fresh profiles)

In a future run of the JVM we can then use these profiles

They get stored in a data-structure located in the C++ heap

And used whenever possible

I present mode 2 with focus on keeping the original tiered compilation steps (being conservative), there are 2 more modes



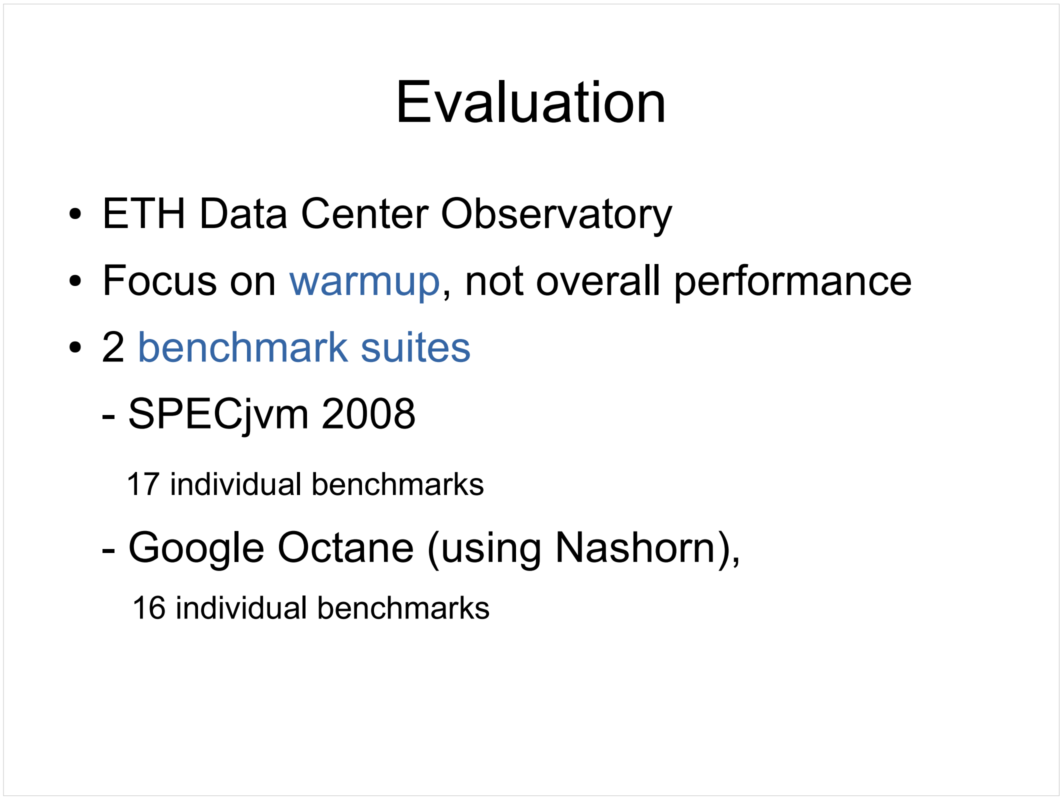
Implementation

- 1846 lines of code
- 24 files affected
- 2 new classes
  - ciCacheProfiles
  - ciCacheProfilesBroker

More than 1800 lines of code

Two dozens of files affected

Separated from the rest, turning feature of results in JVM being unaffected by the changes



Evaluation

- ETH Data Center Observatory
- Focus on warmup, not overall performance
- 2 benchmark suites

  - SPECjvm 2008

    17 individual benchmarks

  - Google Octane (using Nashorn),

    16 individual benchmarks

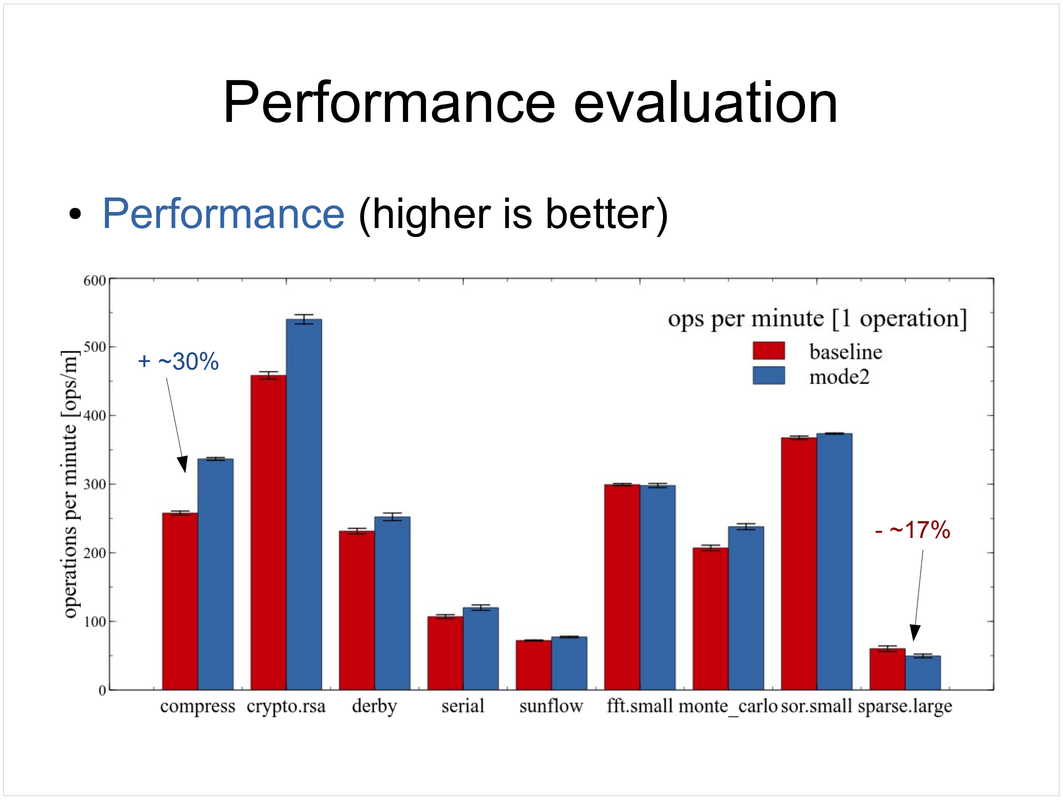
DCO provided by ETH

Had to focus on warmup since it doesn't really affect performance for a long running benchmark

Divided benchmarks up in their parts and restarted JVM in between benchmarks

Also several microbenchmarks which are presented in the thesis

Going to present numbers from Mode2, there is a lot more in the thesis

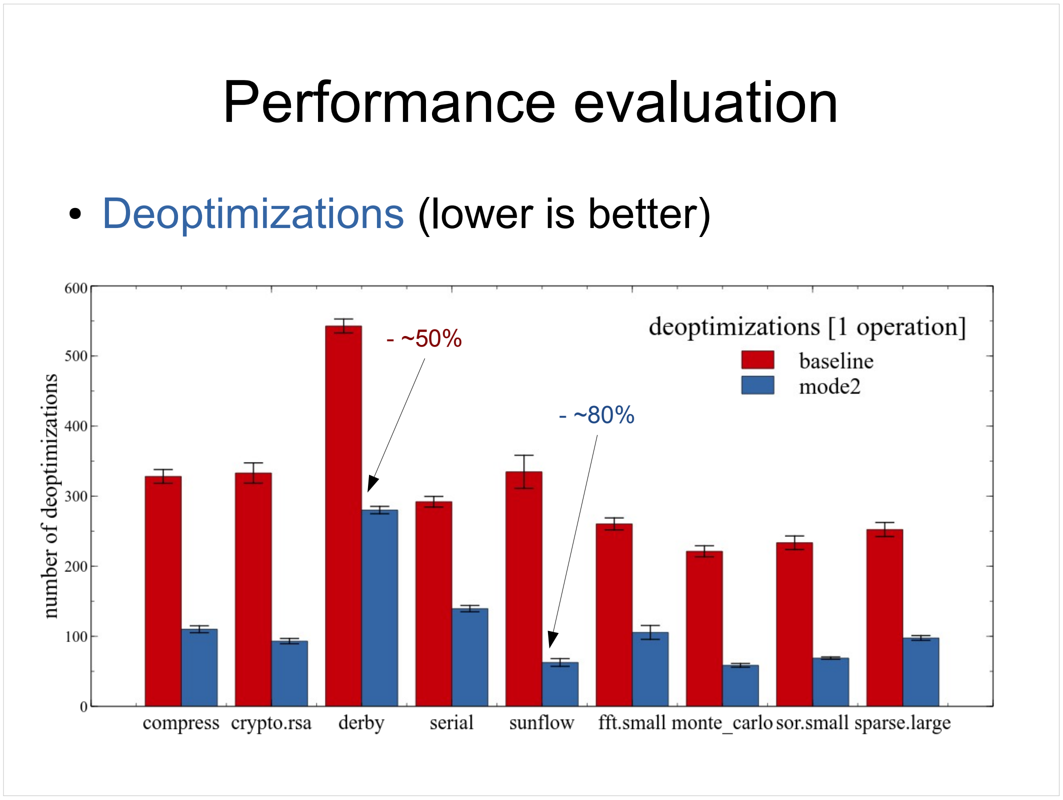


Performance evaluation

- Performance (higher is better)

Performance example of SpecJVM benchmark

1 run, around 5-30 seconds



Performance evaluation

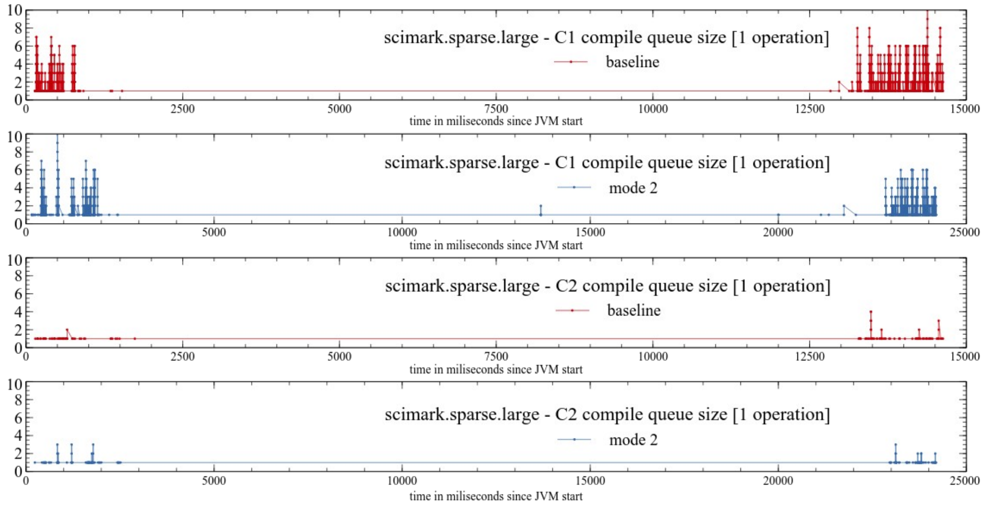
- Deoptimizations (lower is better)

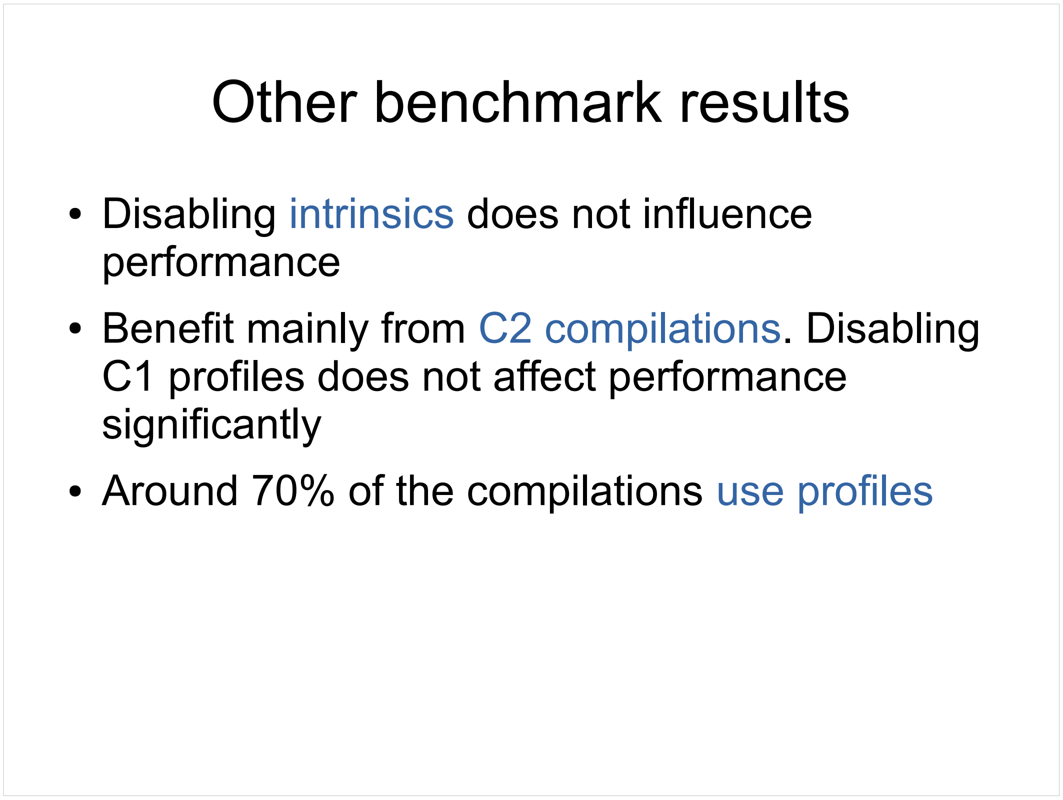
Deoptimizations lowered significantly, good indicator of improved code quality

Still difficult to measure how much influence a deopt has

Performance evaluation

- Compilation queue





Other benchmark results

- Disabling intrinsics does not influence performance
- Benefit mainly from C2 compilations. Disabling C1 profiles does not affect performance significantly
- Around 70% of the compilations use profiles

Rest 30% are mainly L1/L2 compilations and a few skipped ones (lambda expressions etc)

Other approaches

- Presented: Mode2

- Mode 0: skip C1 & lower compilation thresholds
- Mode 1: skip C1 & keep original compilation thresholds

Conclusion

- Complex system
- Reasons for performance influence difficult to measure
- Cached profiles *can* greatly improve warmup performance if used properly
- System requires manual configuration

Thank you for listening