

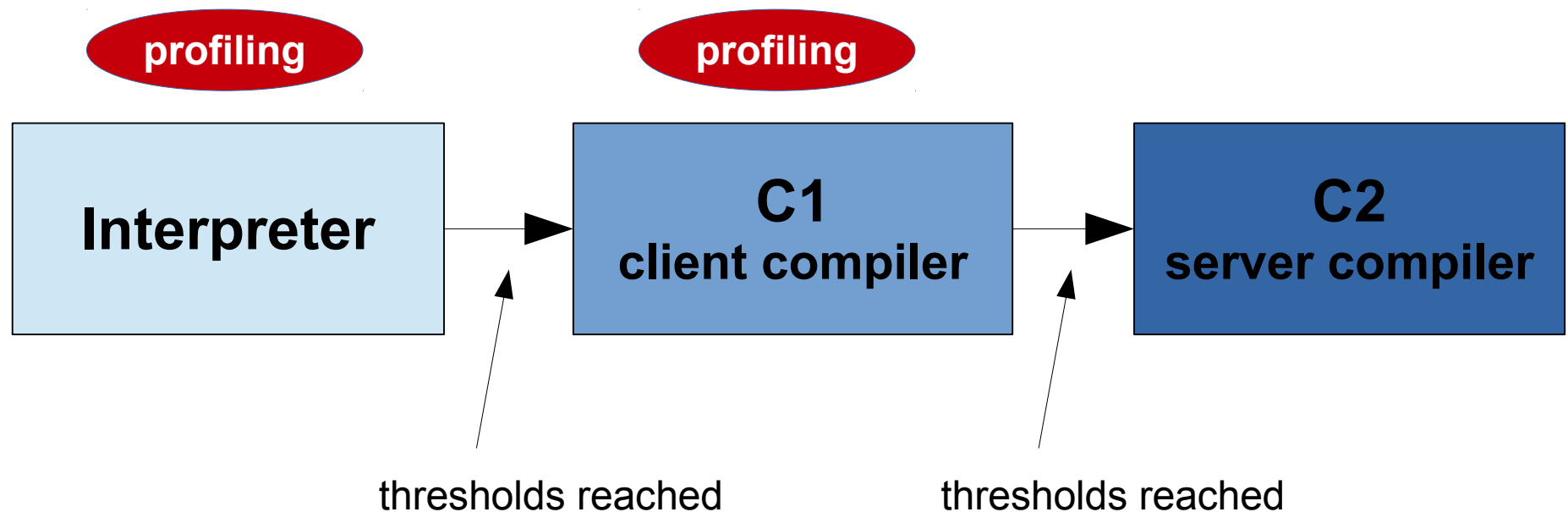
# Profile Caching for the Java Virtual Machine

Marcel Mohler, ETH Zurich  
Bachelor Thesis

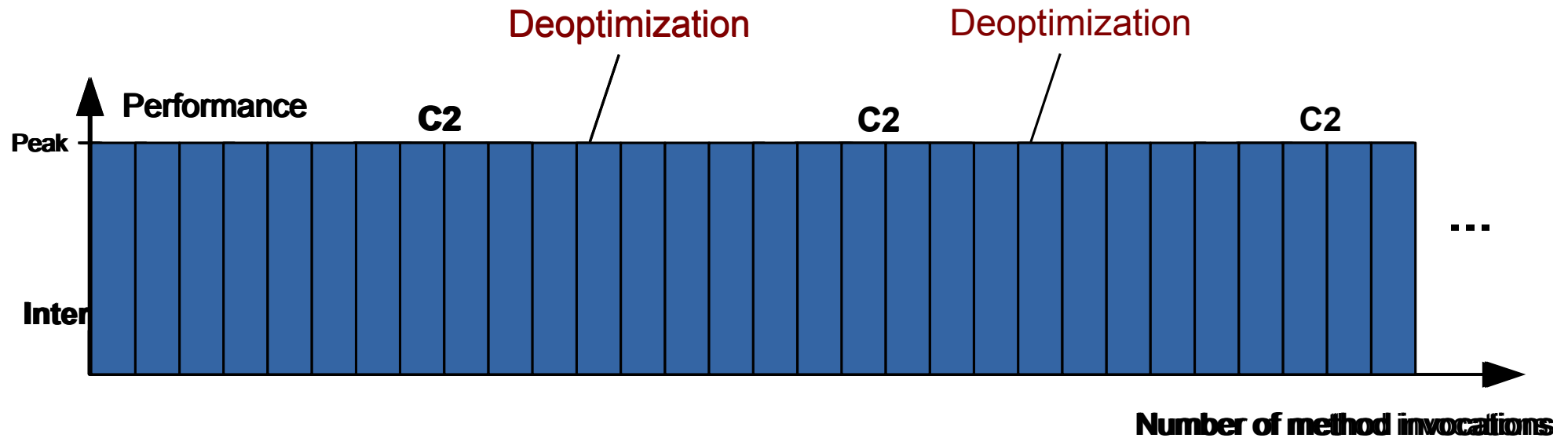
*Supervisors: Zoltan Majo, Oracle  
Tobias Hartmann, Oracle*

*Prof. Thomas Gross, Laboratory for Software Technology, ETH*

# Tiered Compilation in HotSpot JVM



# Problem: performance fluctuations



**Goal: Decrease performance fluctuations**

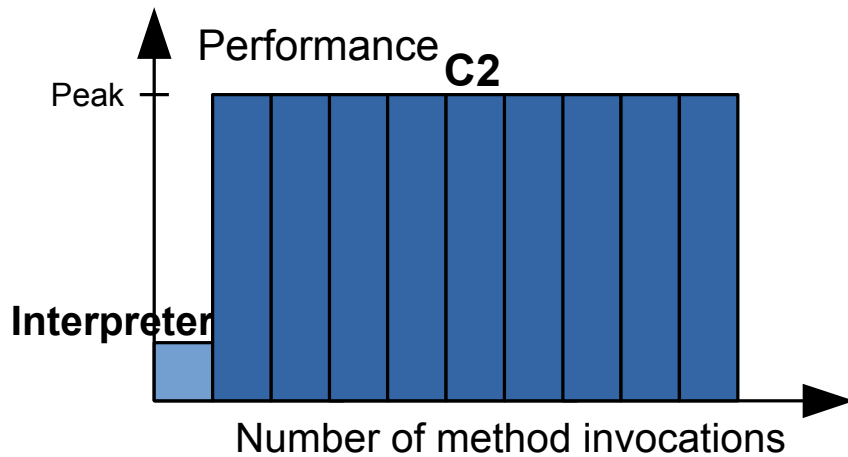
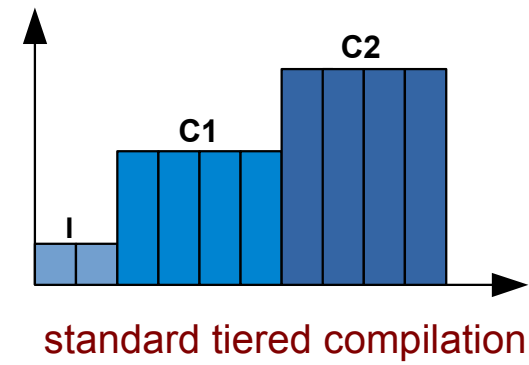
# Observation

- Profiles need to be gathered each time the JVM starts
  - Most frequently used methods often do not change
- Idea: cache and reuse the profiles!

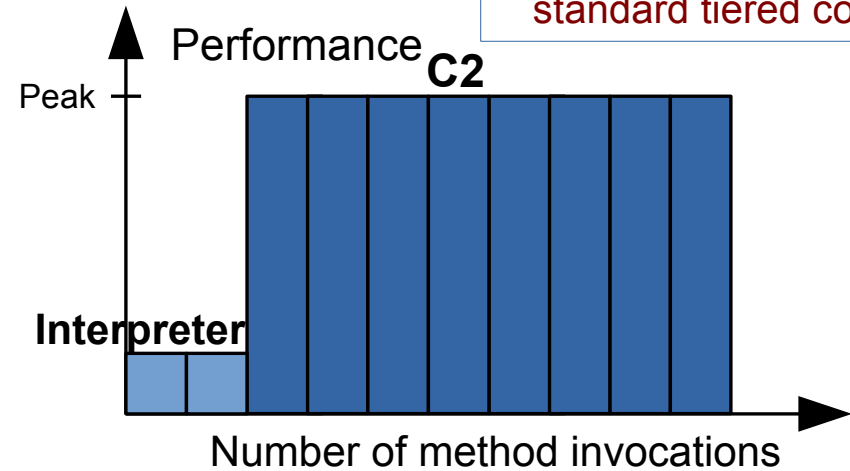
# Outline

1. Design
2. Implementation
3. Performance evaluation
4. Conclusion

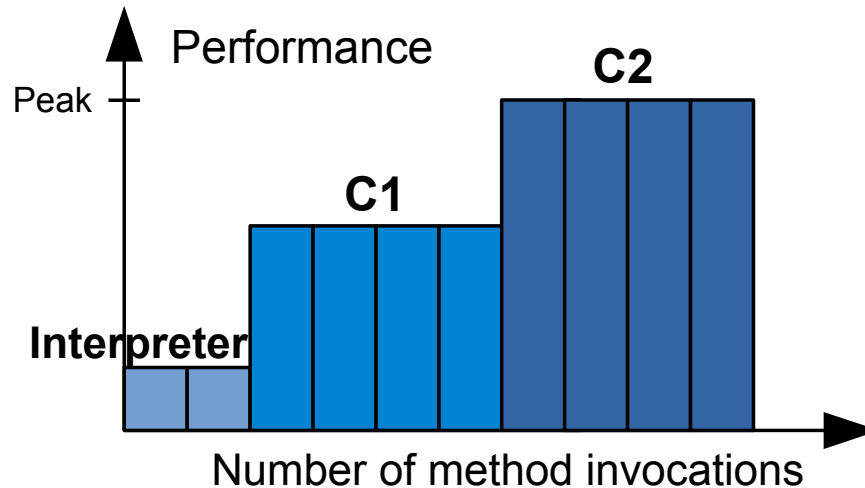
# Design: 3 modes



Mode 0

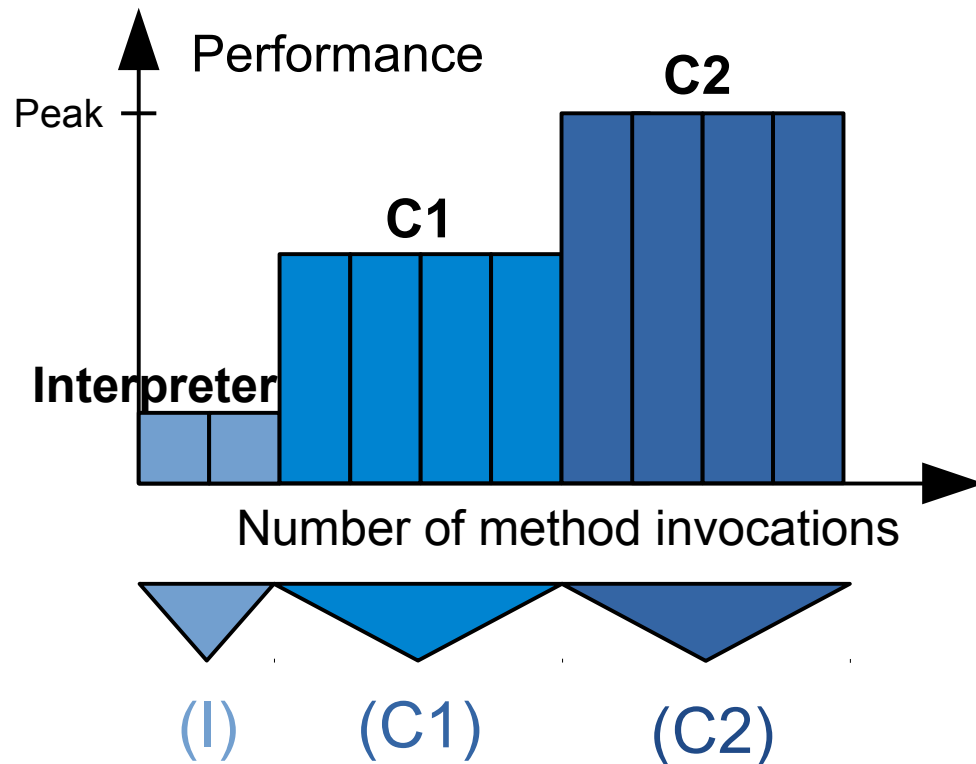


Mode 1



Mode 2

# Design: Mode 2



(I) unmodified

(C1) remove profiling code  
→ ~30% speedup

(C2) use cached profile  
→ better code quality

# Implementation

- Based on **existing** compilation replay functionality
- **Configurable** by JVM flags
- Select **all** or **single** methods

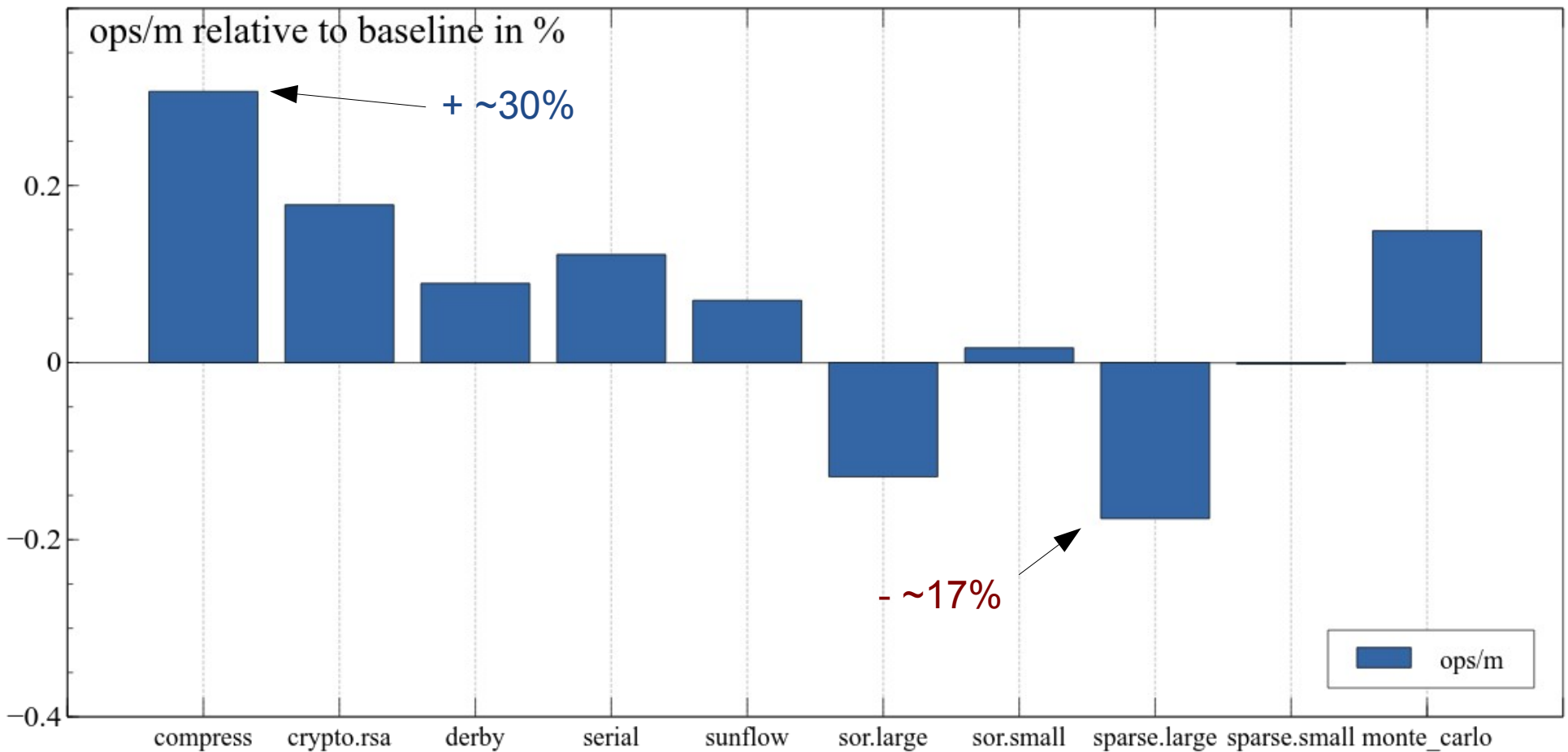


# Evaluation

- ETH Data Center Observatory
- Focus on **warmup**
- 2 **benchmark suites**
  - SPECjvm 2008  
17/21 individual benchmarks used
  - Google Octane (using Nashorn)  
16/17 individual benchmarks used

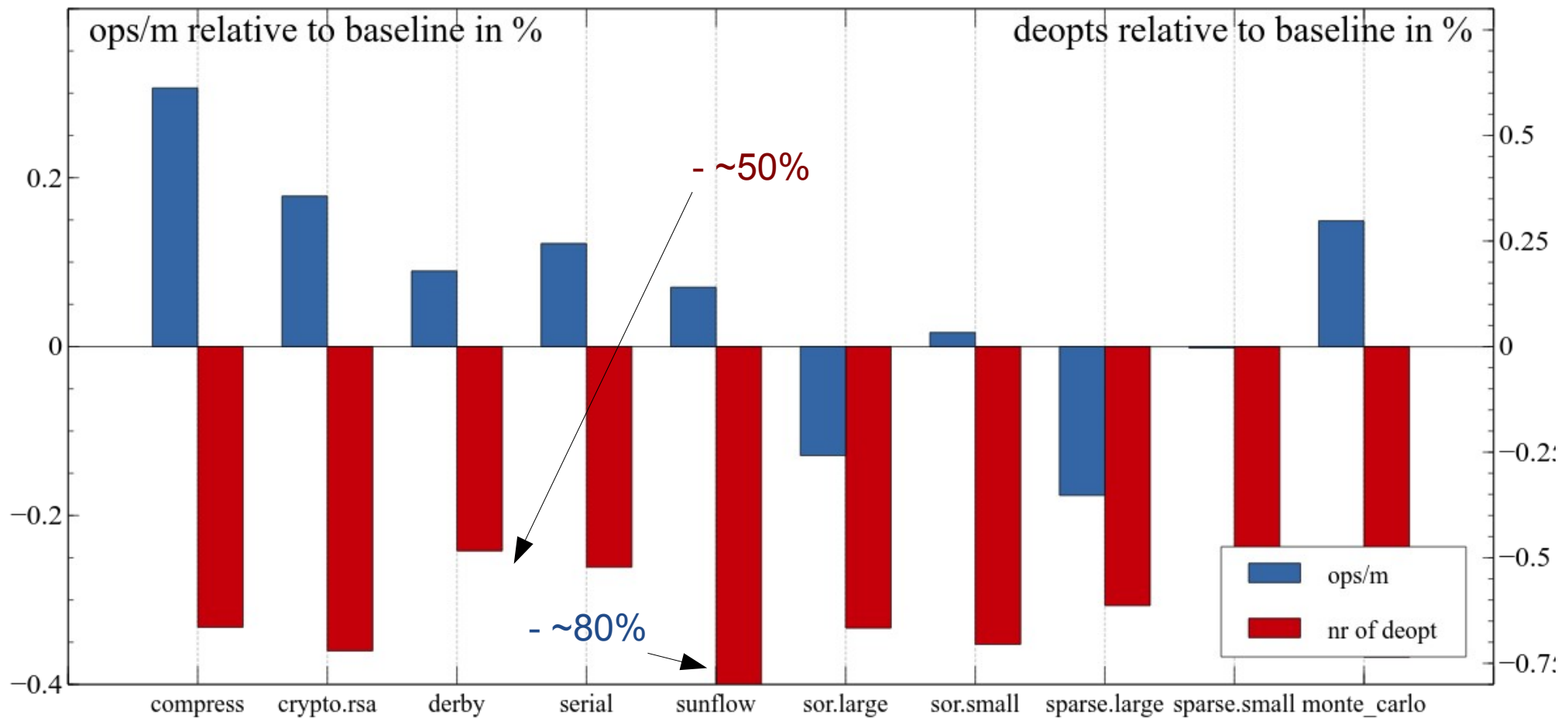
# Performance evaluation

- Performance (higher is better)



# Performance evaluation

- **Deoptimizations** (lower is better)

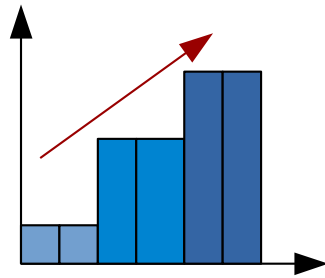


# Other benchmark results

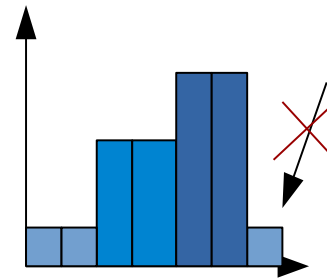
- Benefit mainly from **C2 compilations**. Disabling interpreter profiles rarely affects performance
- Around 70% of the compilations **use cached profiles**
- No association between performance and load on **compile queue**

# Conclusion

- Cached profiles can **improve** warmup performance
- System allows **fine tuning**
- **Main benefits:**



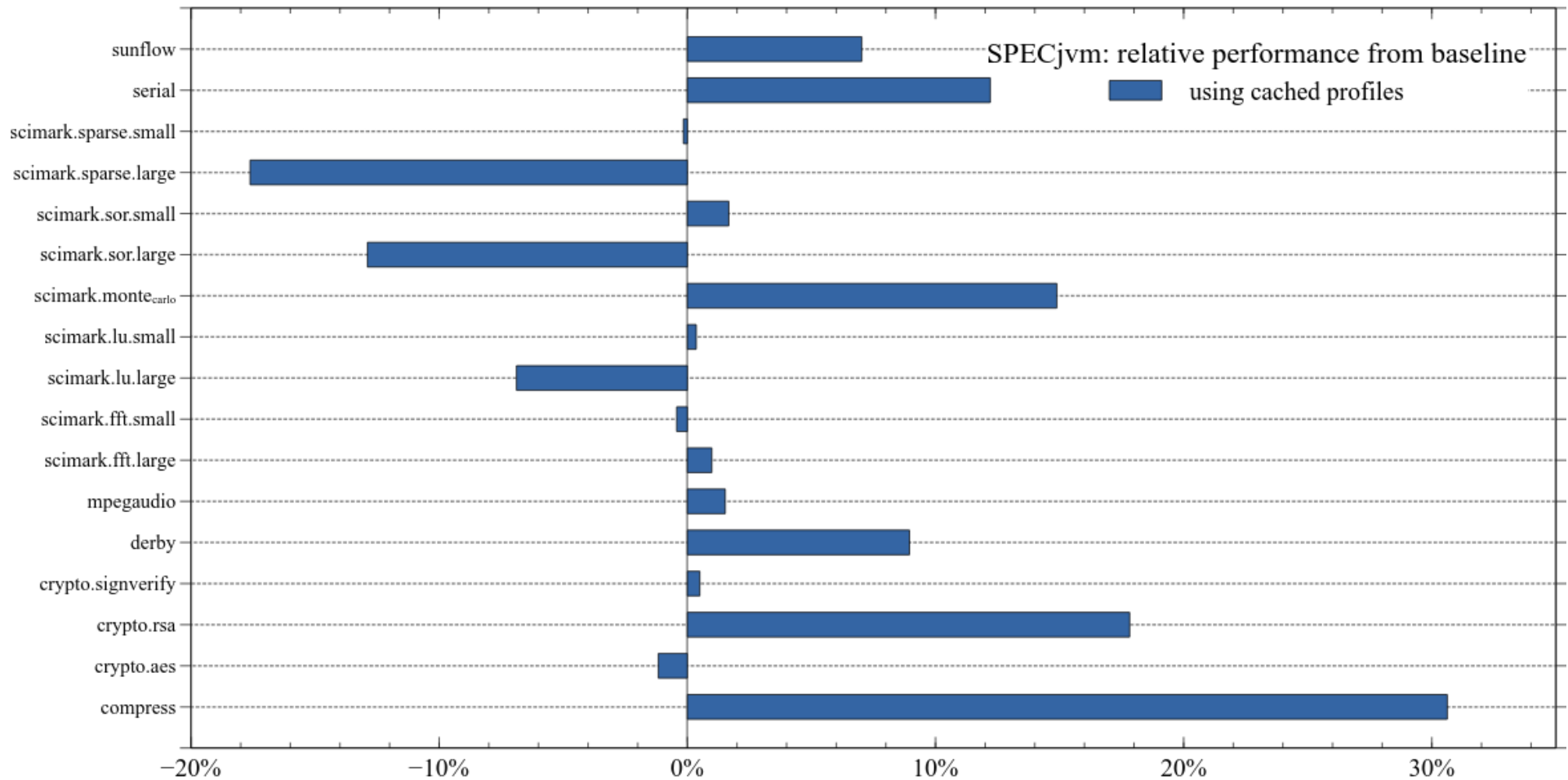
faster warmup



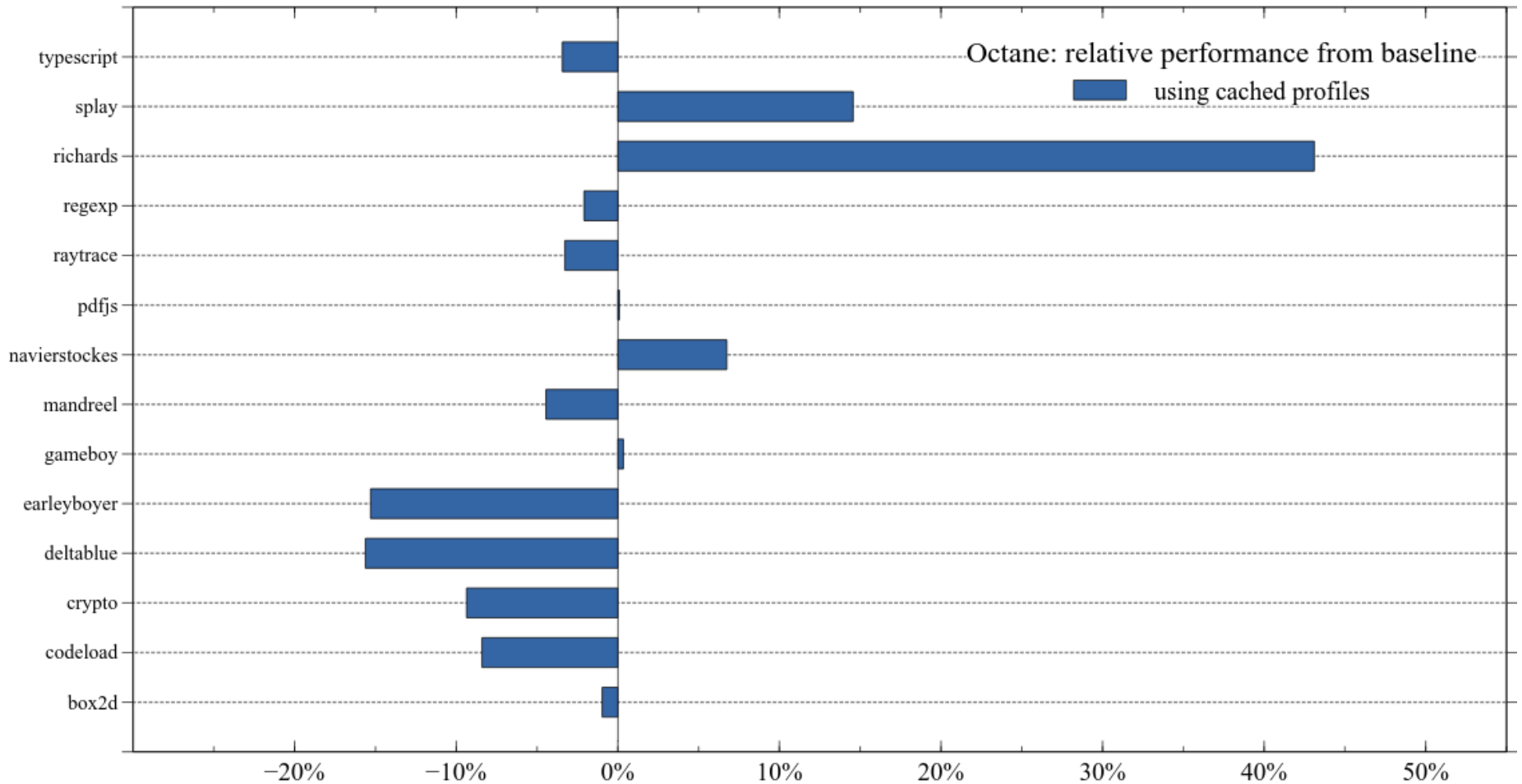
fewer deoptimizations

- Room for future work

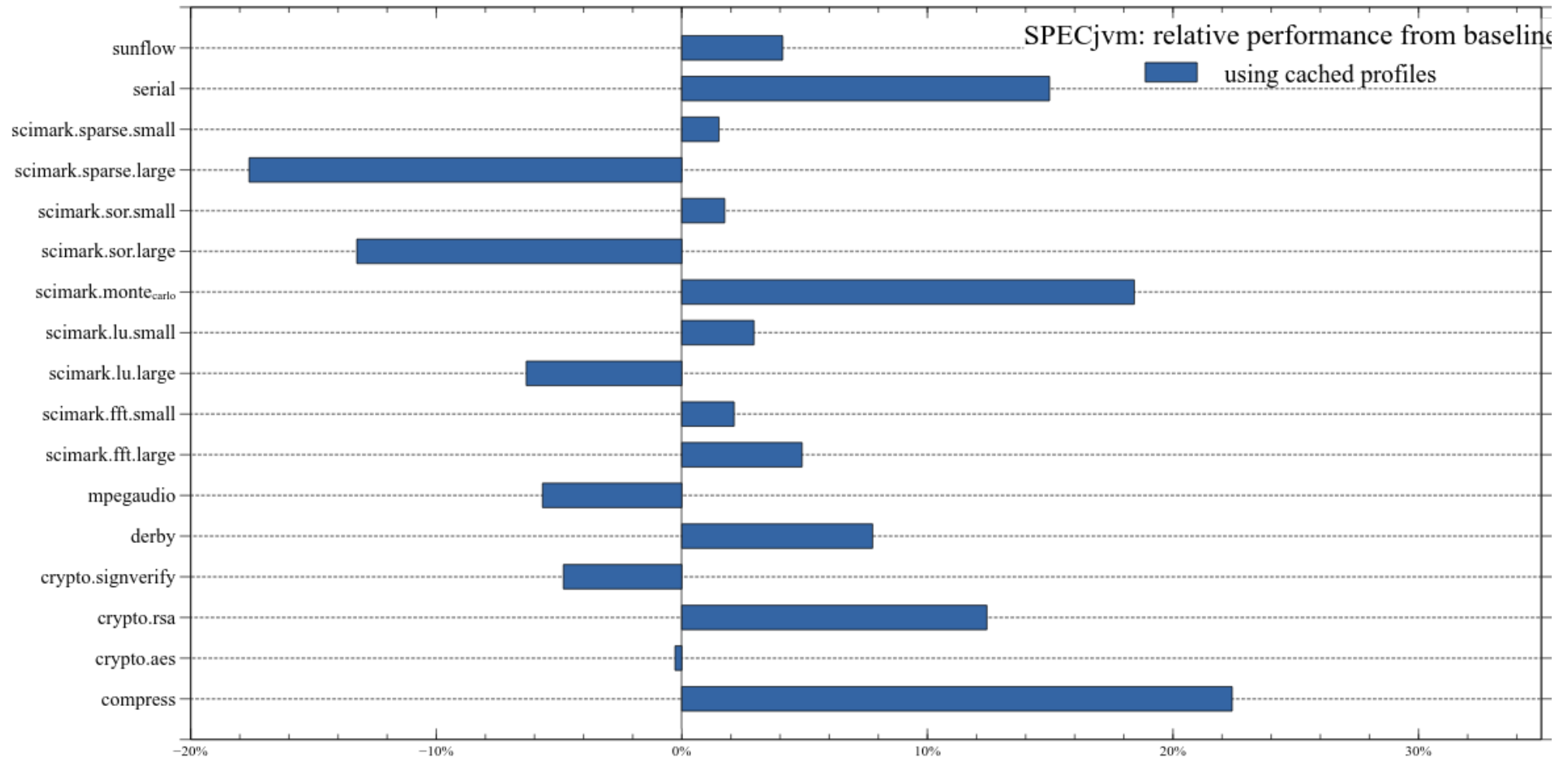
# SPECjvm: relative performance



# Octane: relative performance

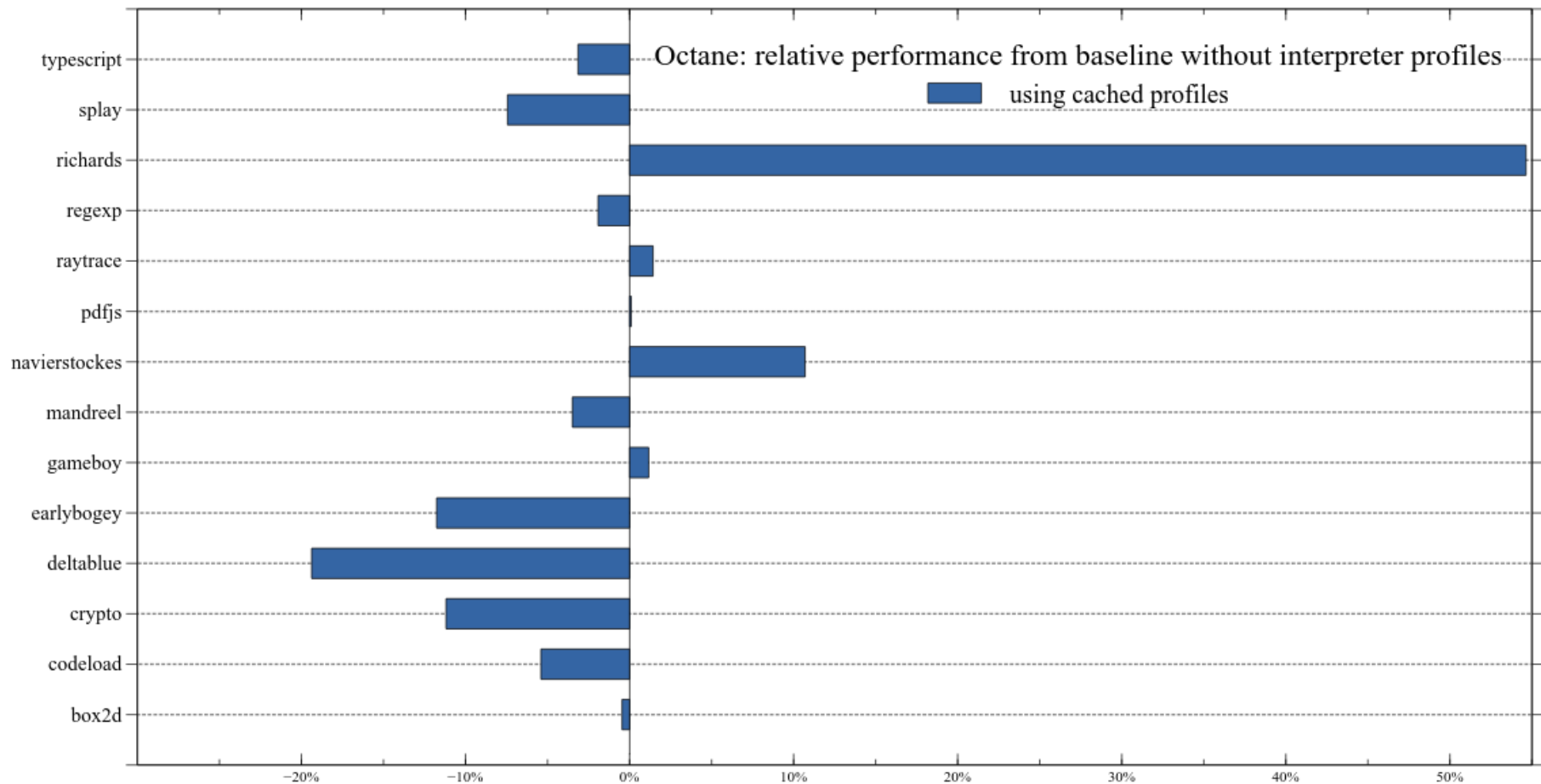


# SPECjvm: relative performance without interpreter profiles



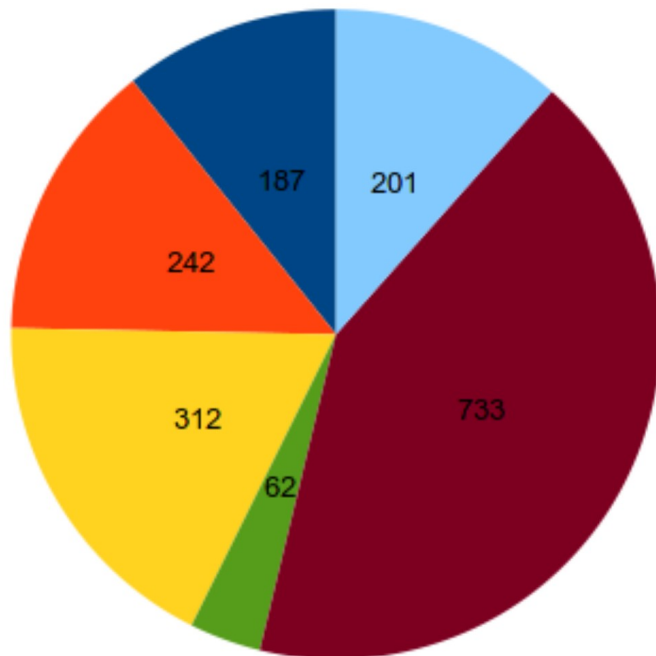


# Octane: relative performance without interpreter profiles

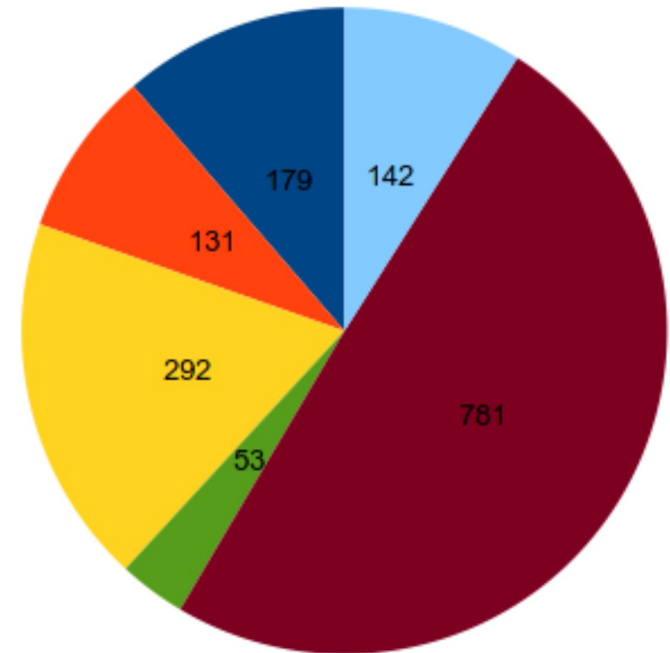


# Type of compilations

compress - mode 2



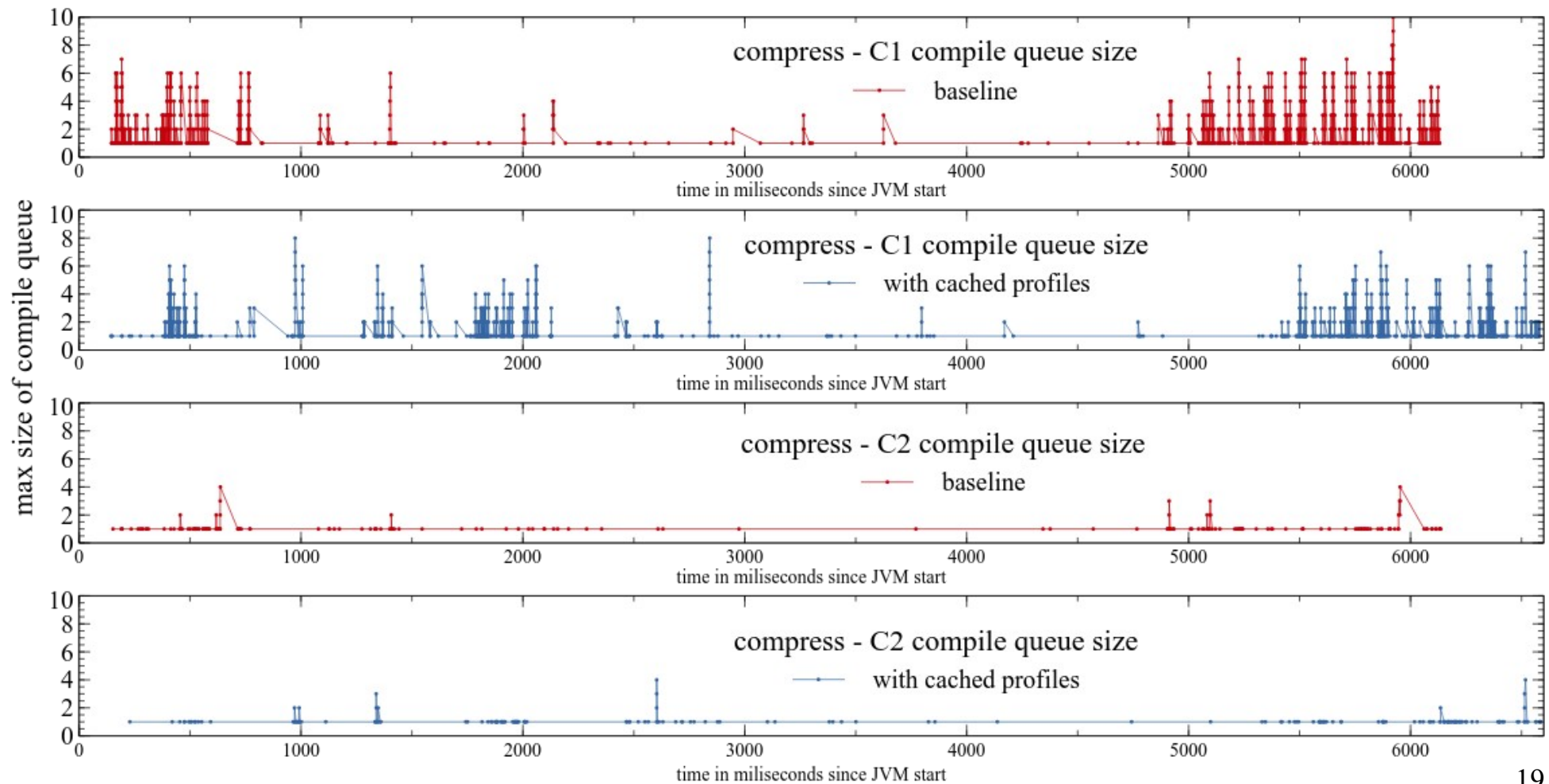
sparse.large - mode 2



- without cached profile – compile level 1
- without cached profile – compile level 2
- without cached profile – compile level 3
- without cached profile – compile level 4
- with cached profile – compile level 3
- with cached profile – compile level 4

# Compile queue

- compress



# Compile queue

- scimark.sparse.large

