A Block-Based Bytecode Format

to Simplify and Improve Just-in-Time Compilation

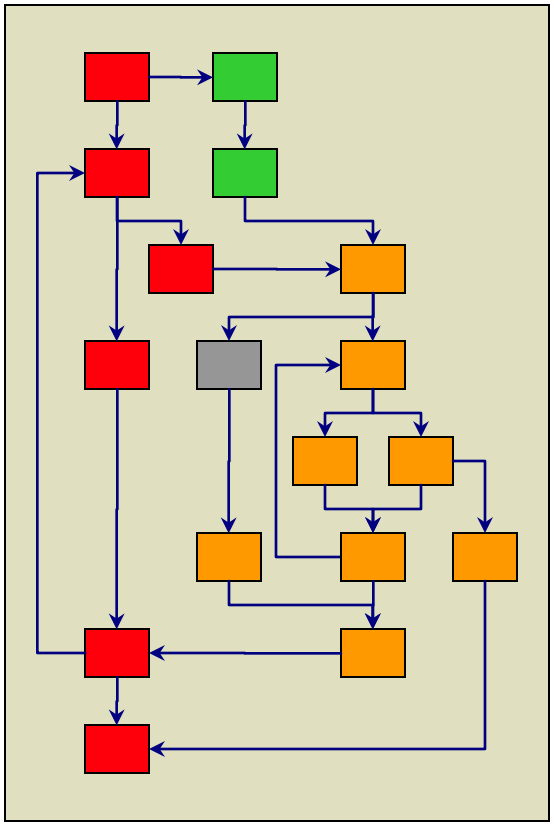
Christian Wimmer
cwimmer@uci.edu
www.christianwimmer.at

June 2009

Department of Computer Science
University of California, Irvine

Method Granularity

Software Engineering

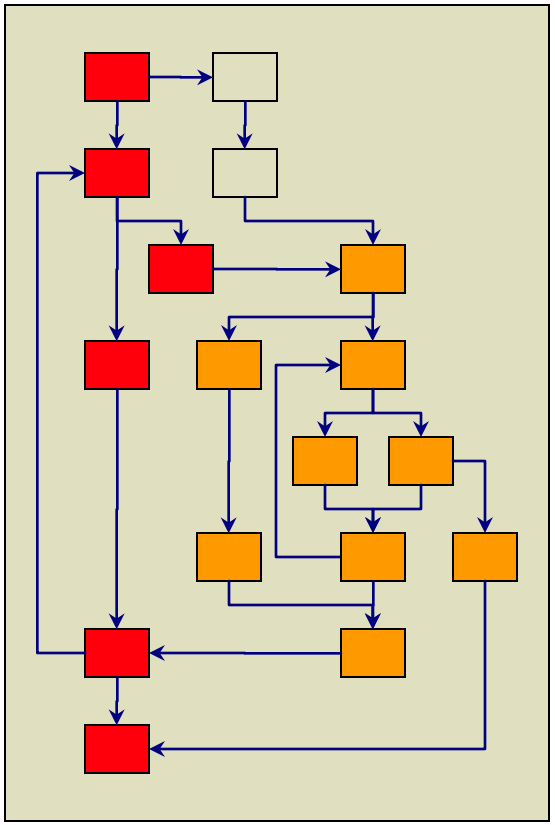


Unstructured Bytecodes
Method Granularity

JIT Compiler Design derived from Static Compilers

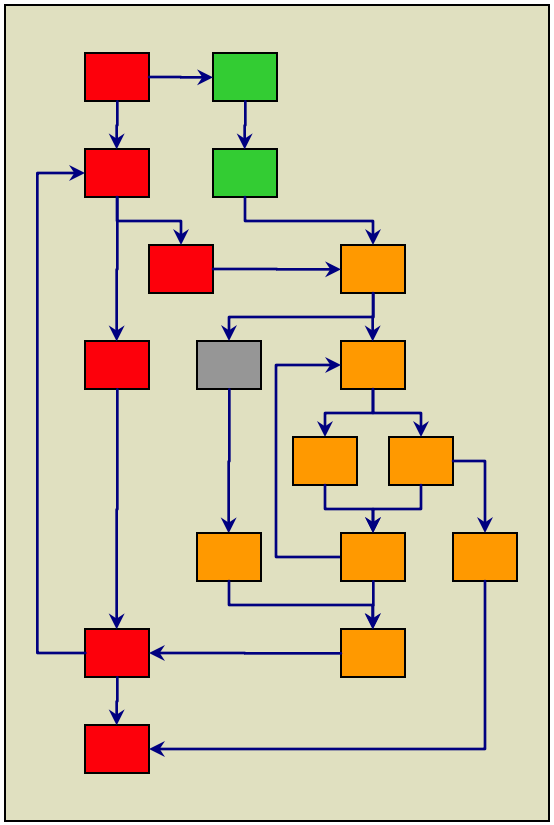
Method Structure used in Compiler because "it's there"

Compiler Optimization



Block Granularity

Software Engineering

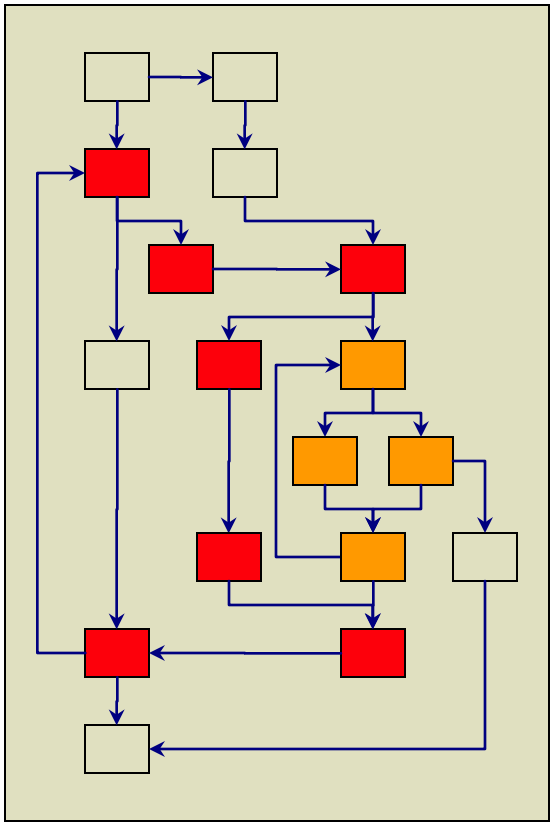


Block-Based Bytecodes with Structure Information

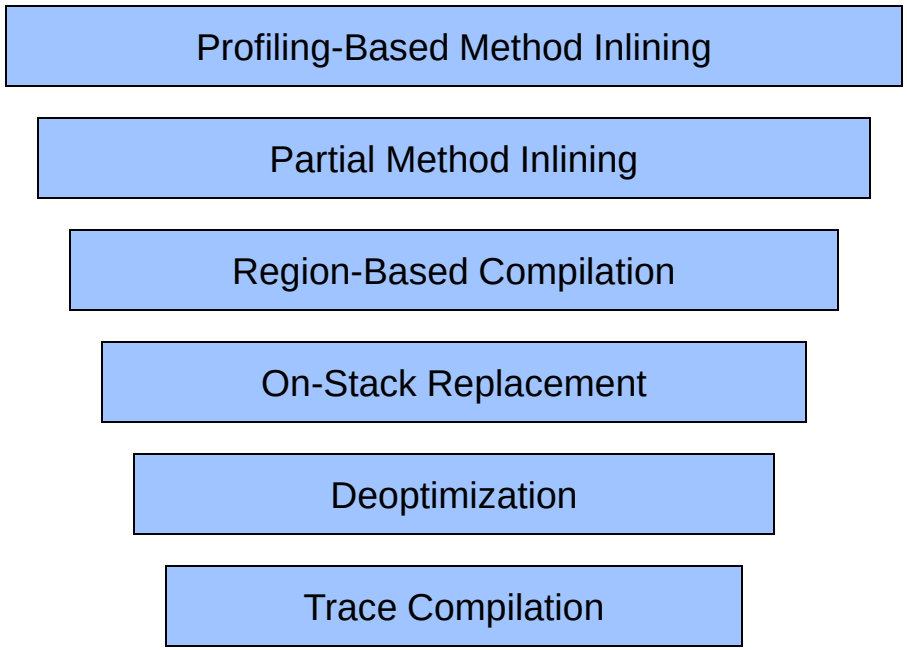
Novel JIT Compilation Strategies using Profiling Data

Compilation Units based on Execution Frequency

Compiler Optimization



Compiler Research



Main Idea and Challenges

Definition of a Block-Based Bytecode Format

Execution in Virtual Machine

- Efficient Interpretation
- Simple Compilation
- Profile Information
- Trace Compilation

Source Languages

- For Static and Dynamic Languages
- Functional Languages
- Dynamic Method Calls
- Closures, Tail Calls, …

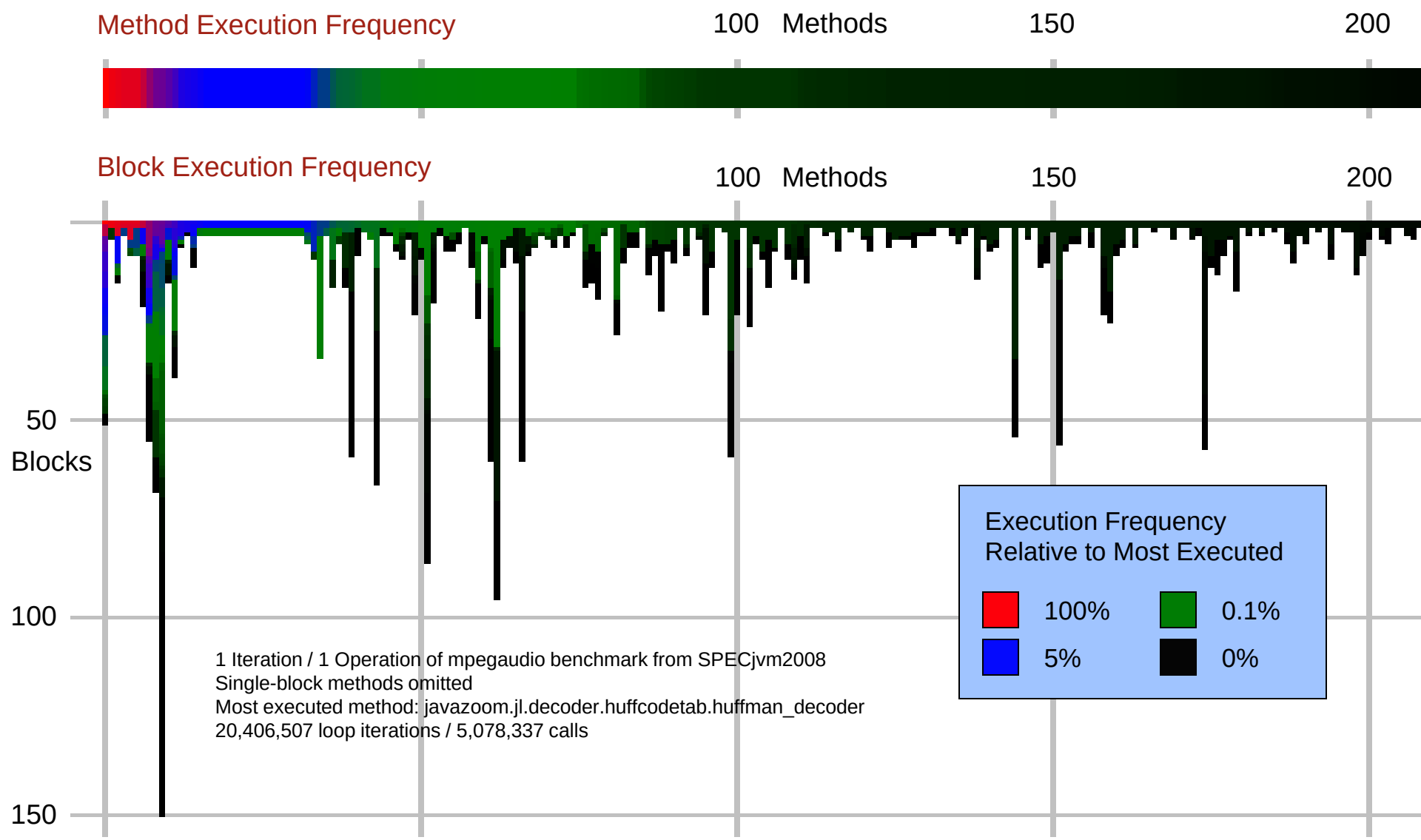
Design Decisions

- High-Level Structure, e.g., Loops
- “Calling Convention” of Blocks
- Stack Based vs. Register Based
- Static Single Assignment Form?

Safety and Security

- Easy Verification
- Inherently Safe Code Formats

SPECjvm2008 – mpegaudio



Summary

Software Engineering	Compiler Optimization
Method Structuring	Block Structuring
Readability	Execution Frequency
Extendibility	Feedback-Directed Optimization
Reusability	Profiling
…	…

Methods hide the real focus where compiler optimizations should be applied