Trace Compilation

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

September 2009

Department of Computer Science
University of California, Irvine

Background

Institute for System Software
Johannes Kepler University Linz, Austria

Java HotSpot™ VM

- Linear Scan Register Allocation
- Automatic Object Inlining
- Array Bounds Check Elimination
- Optimization of Strings
- Tail Calls

Department of Computer Science
University of California, Irvine

Maxine VM

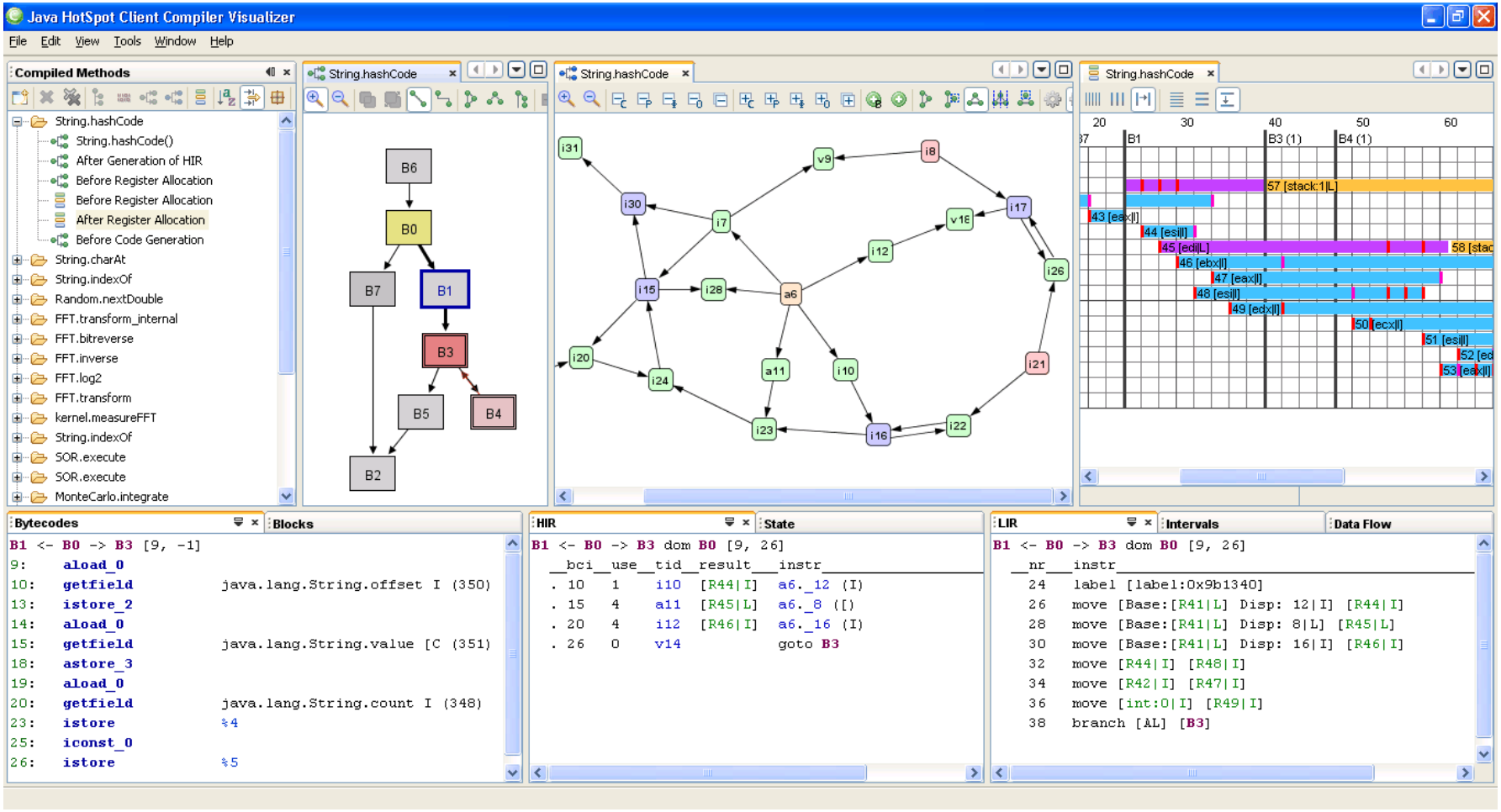
- Trace Based Compilation
- Phase Change Detection
- Hierarchical Layering of VMs

Feedback-Directed Optimistic Optimizations for Java

Commercial 1 – Tail Call Optimization

- Credit: Arnold Schwaighofer, JKU Linz
- Hard tail calls
  - Prefix for invoke bytecodes (reuse "wide" prefix)
  - Semantics checked by bytecode verifier
- Prototype implementation for Java HotSpot VM
  - Interpreter, Client Compiler, Server Compiler
  - Problem: Fixed size of compiled frames, parameters part of caller
    - Not enough space for parameter of tail calls
    - Non-sibling tail calls require adapter frames
- Protection domains
  - Security information is attached to stack frames
  - Problem: Tail calls across protection domains are security hole
    - Solution 1: Don't allow, but throw exception
    - Solution 2: Perform normal call and compress stack later if needed

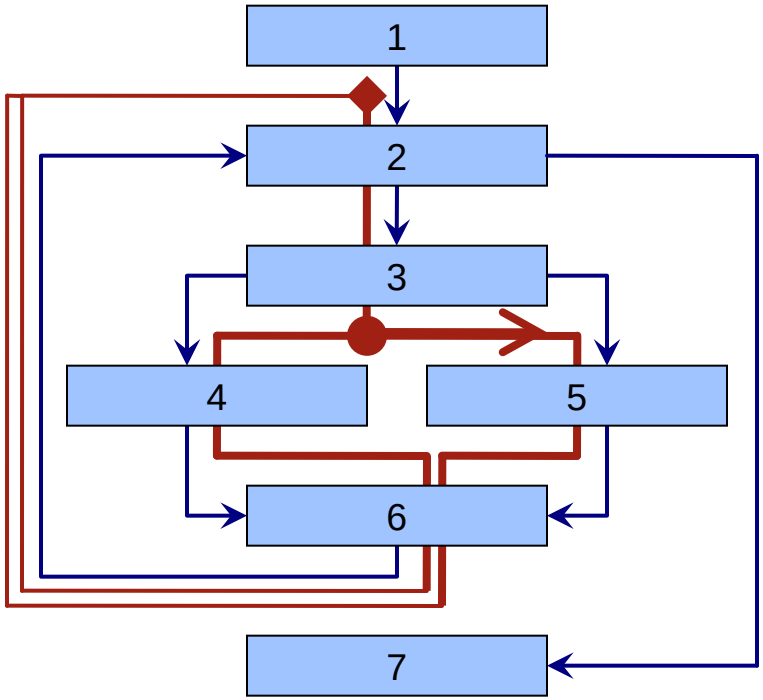
Commercial 2 – Client Compiler Visualizer



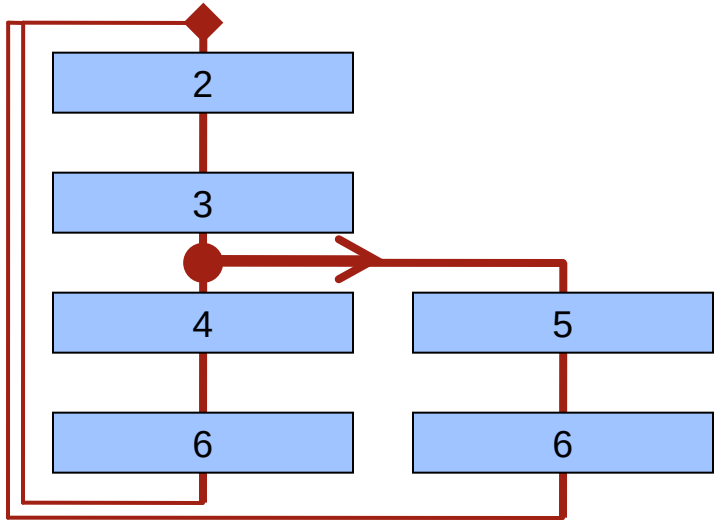
https://c1visualizer.dev.java.net/

Trace Compilation

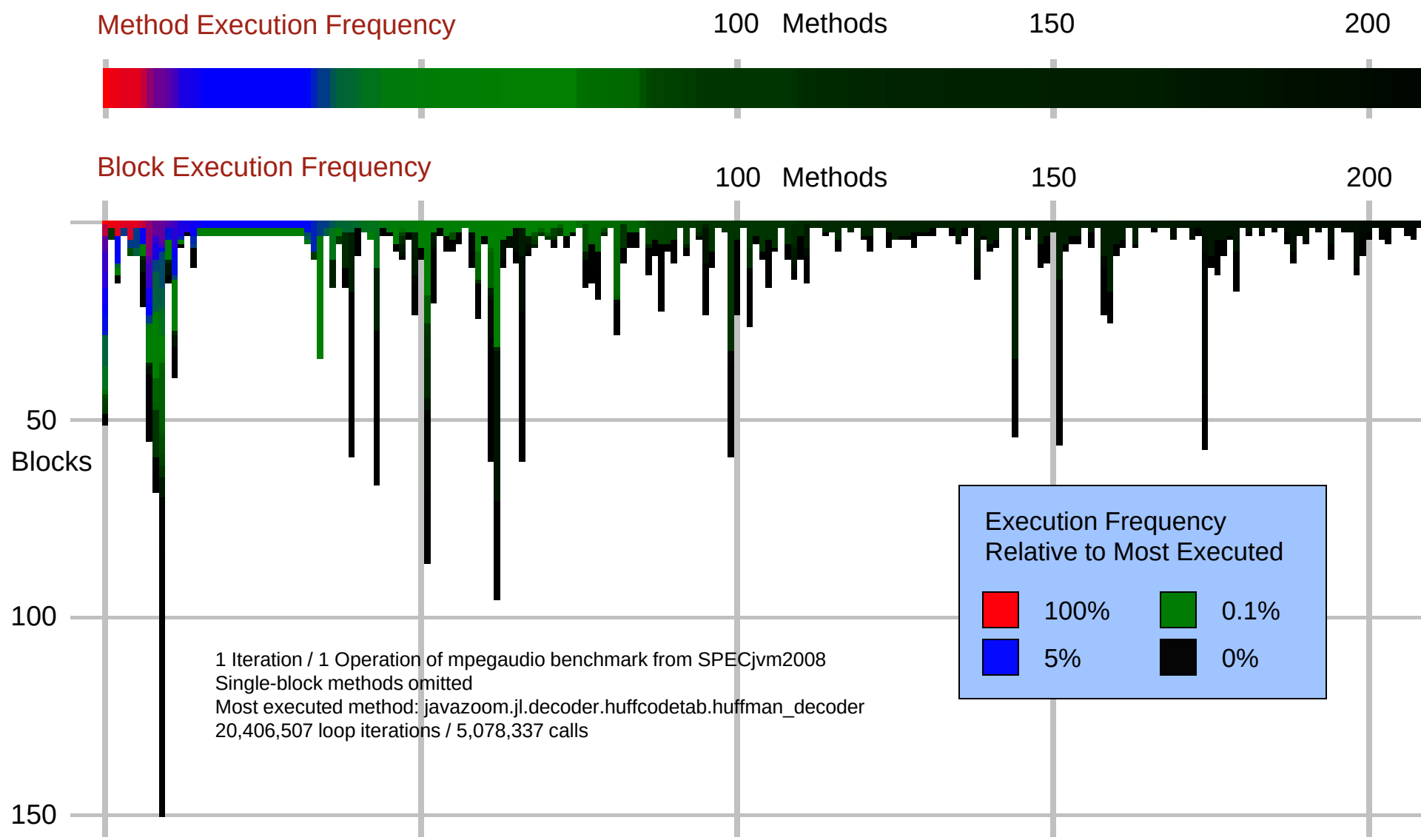
Control Flow Graph



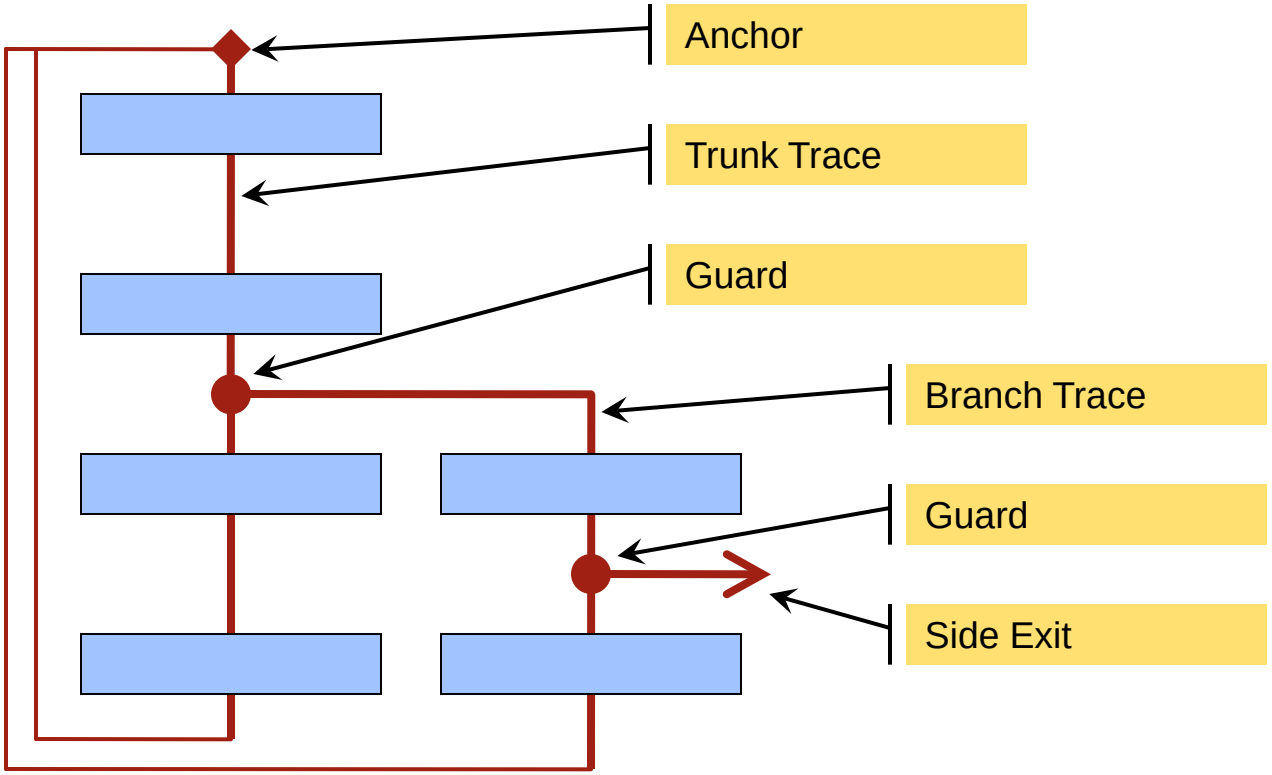
Trace Tree



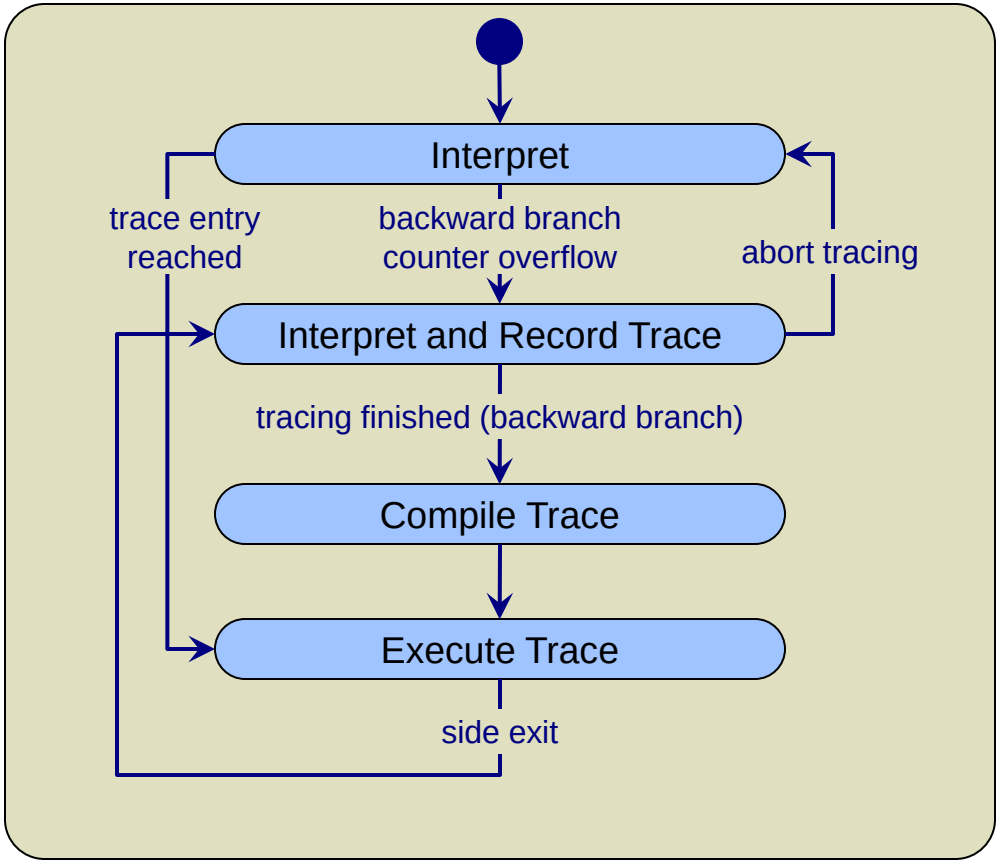
Why Trace Compilation?



Elements of a Trace Tree



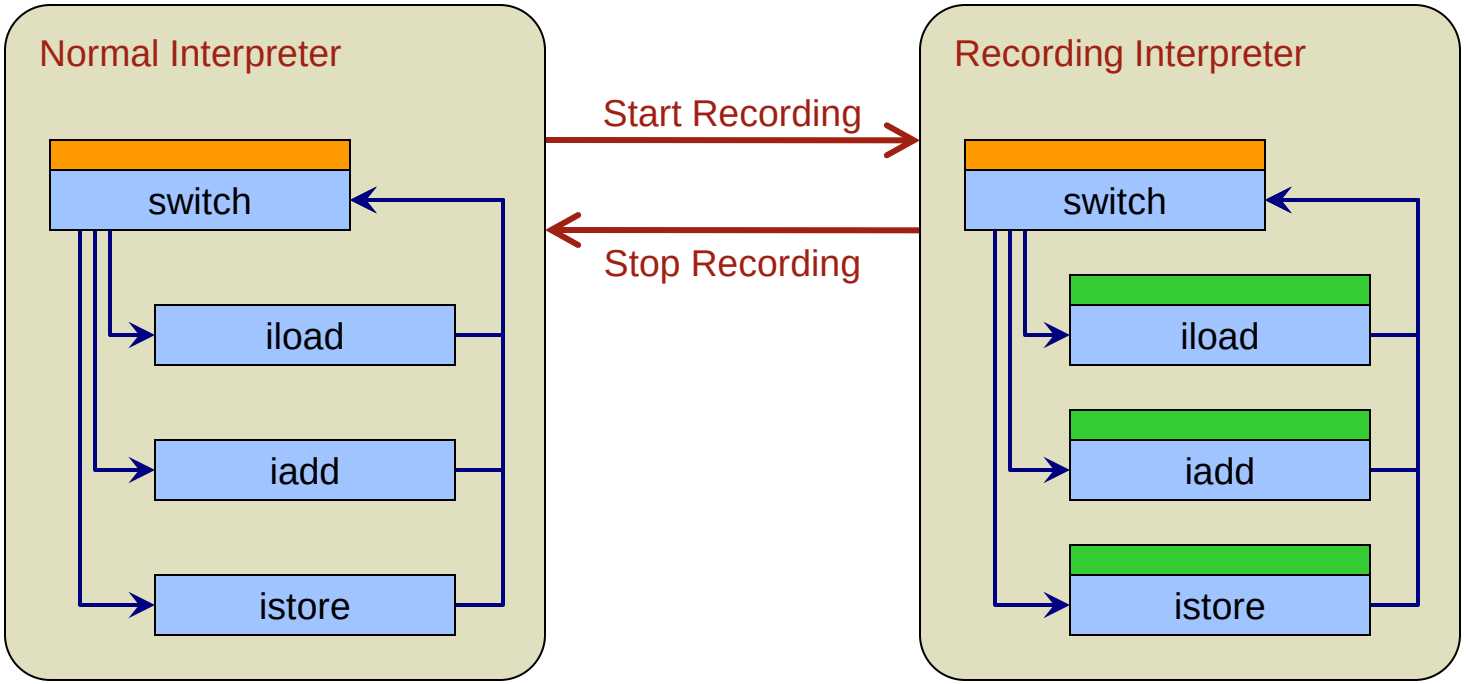
Trace Compilation



Flavors of Trace Compilers

- Dynamic Languages
  - TraceMonkey: Mozilla's JavaScript engine for Firefox
  - Elimination of dynamic type checks
  - Low compilation threshold
  - Fast compilation, no expensive optimizations (yet)
- Java – Maxine VM
  - Evolved from a research prototype (Hotpath VM)
  - Maxine has no interpreter – where to integrate trace recording?
  - Traces currently cover only loops, no merge points
- Java – HotSpot VM
  - Research funding requested, but still pending
  - Stay tuned…

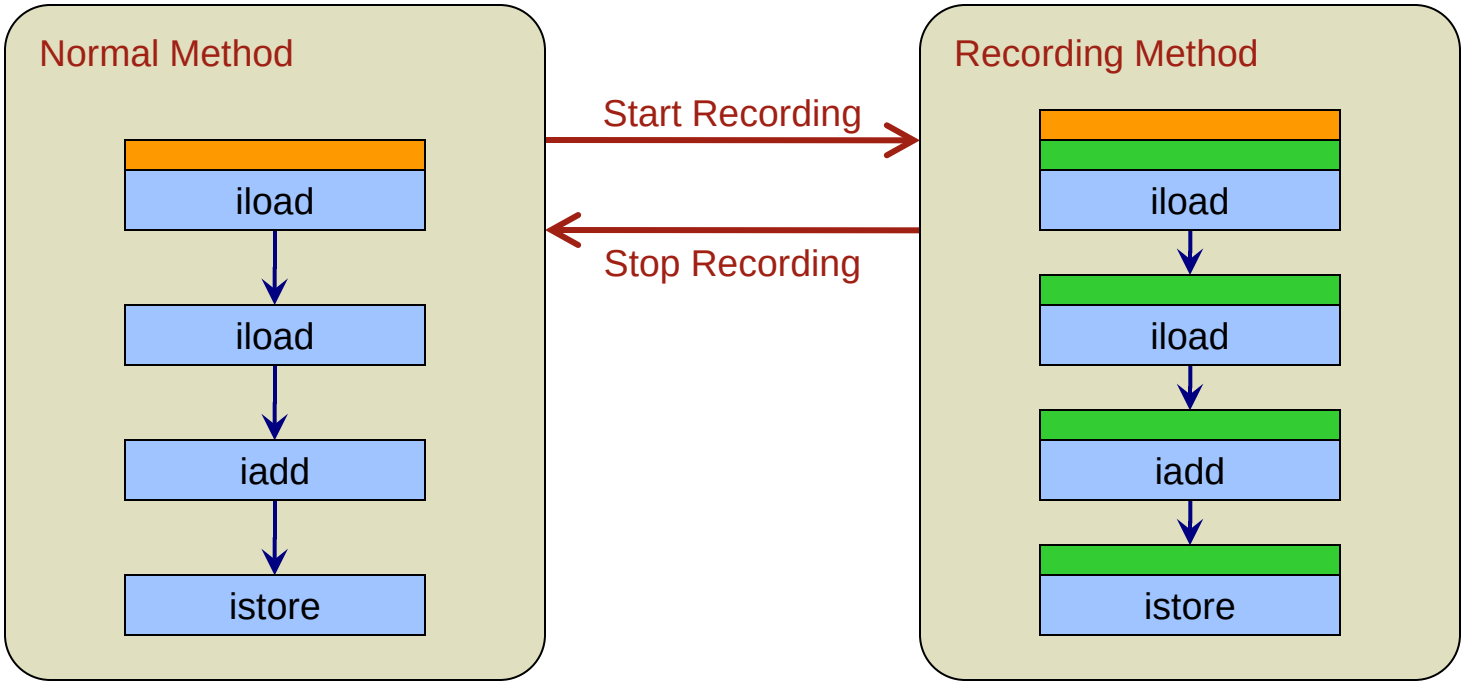
Trace Recording in Interpreter



Profiling Instrumentation

Recording Instrumentation

Trace Recording in Template JIT



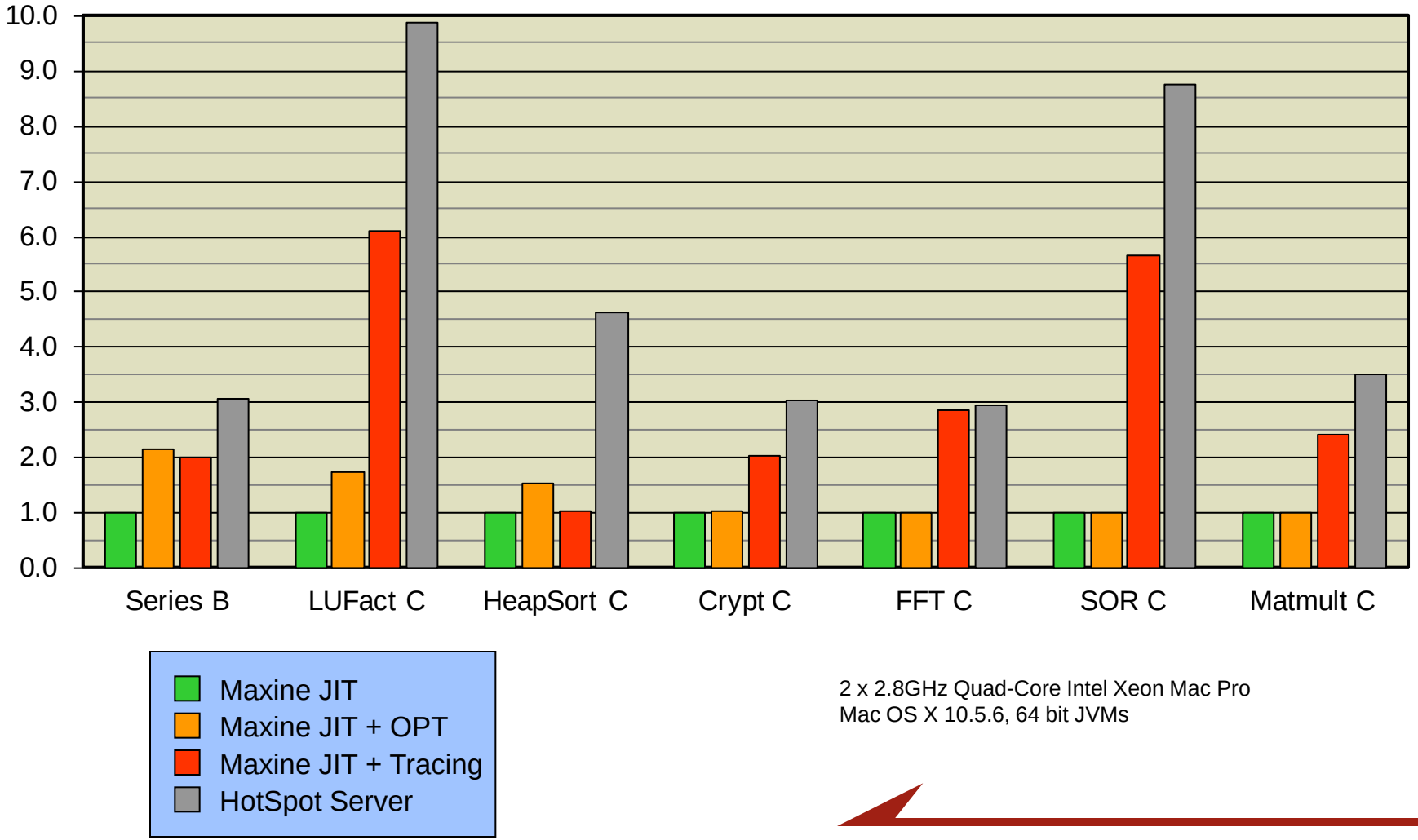
Profiling Instrumentation

Recording Instrumentation

Evaluation Trace Compilation

Speedup relative to Maxine baseline compilation

Preliminary Numbers!

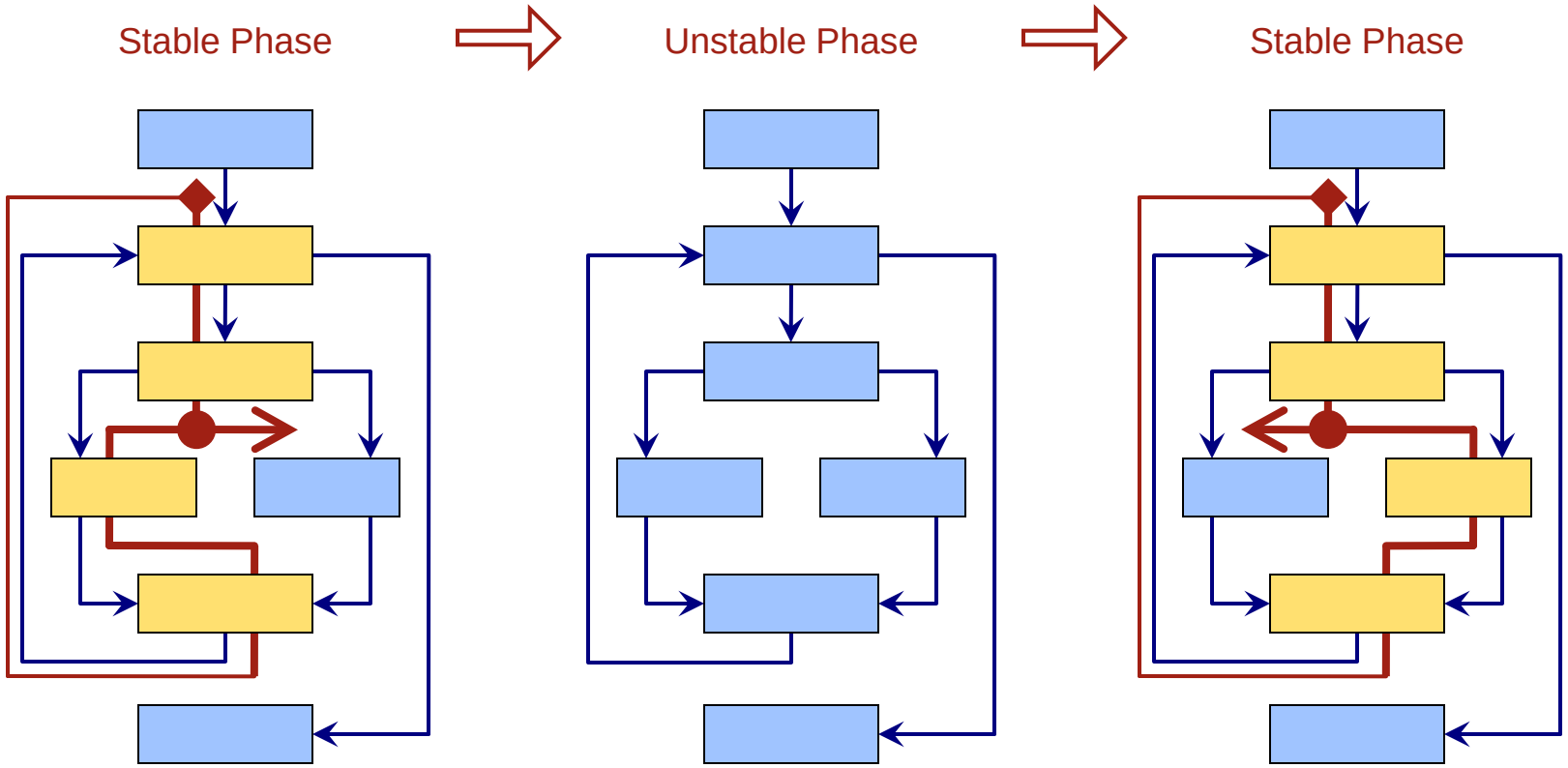


2 x 2.8GHz Quad-Core Intel Xeon Mac Pro
Mac OS X 10.5.6, 64 bit JVMs

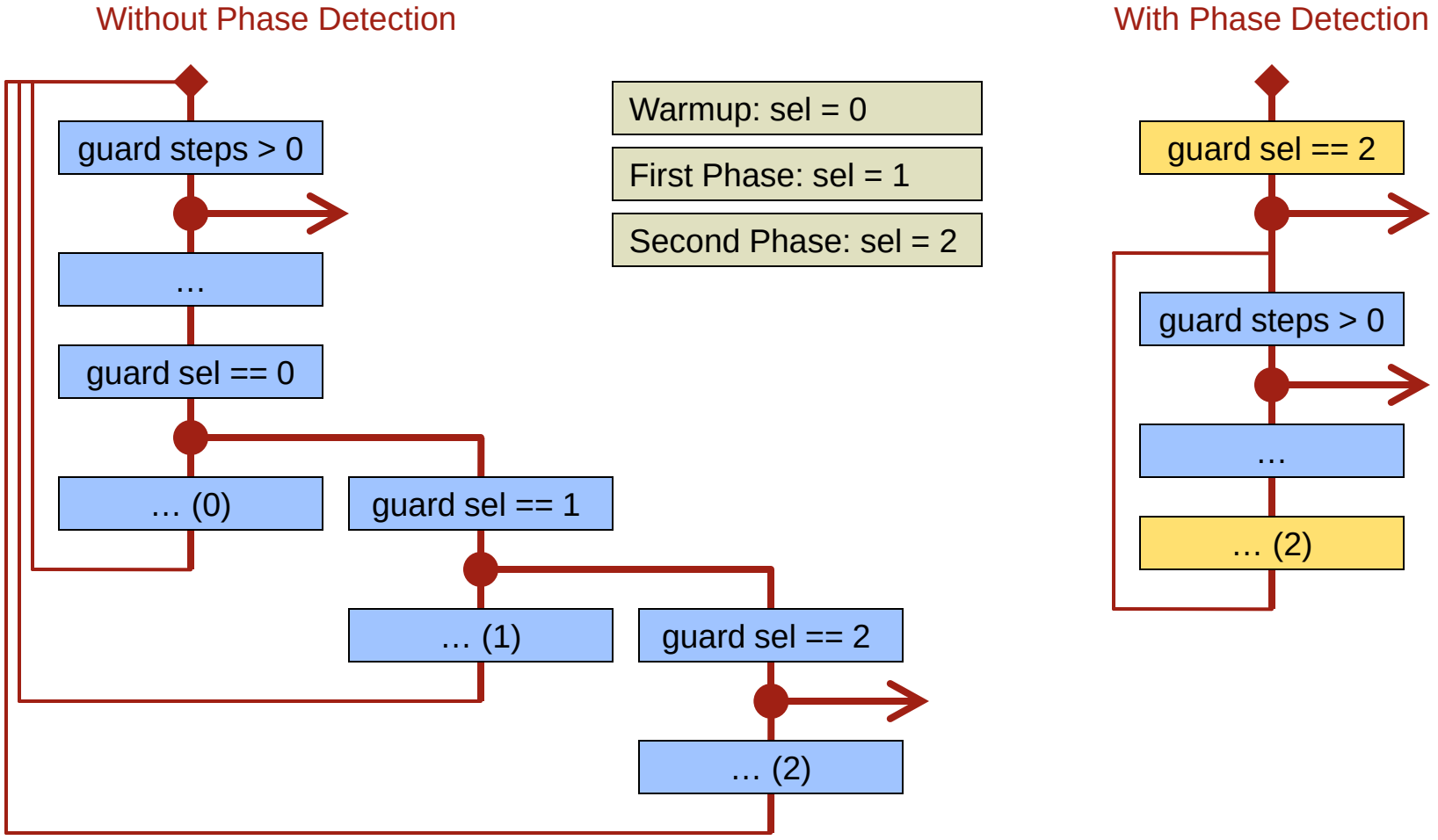
Problems and Future Work

- Loop-centric approach
  - Trace and compile only loops
  - Good for numerical applications
  - Loops are not the first-order element of object-oriented applications
  - Need to optimize also on per-method basis if necessary
- Tail duplication
  - Trace trees lead to duplication of join blocks
  - Block count can explode for complex control flow
  - Need to support join blocks in traces
- Trace calling convention
  - Switch to and from traces with low overhead

Phase Detection using Trace Compilation

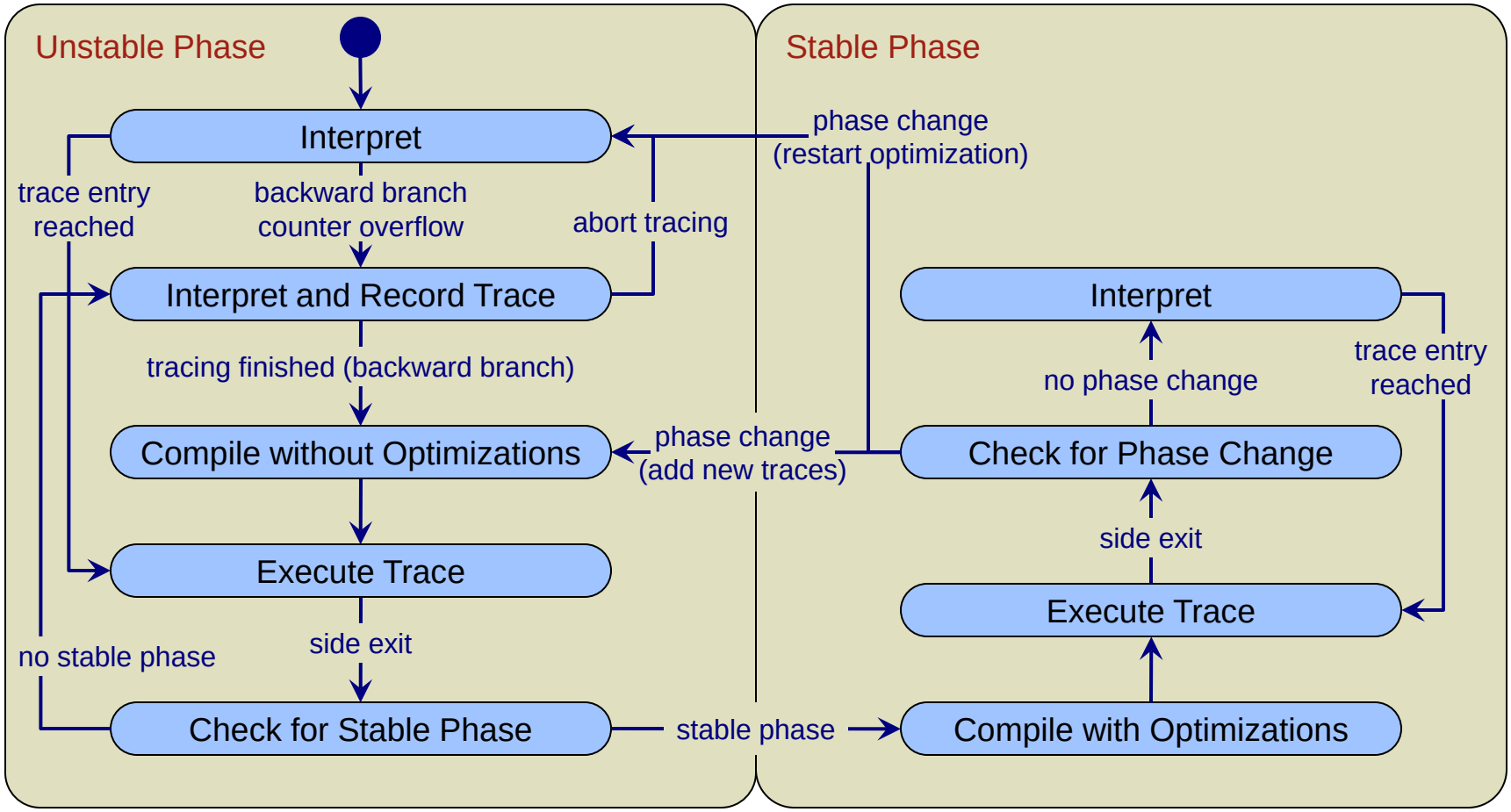


Phases Based on Control Flow

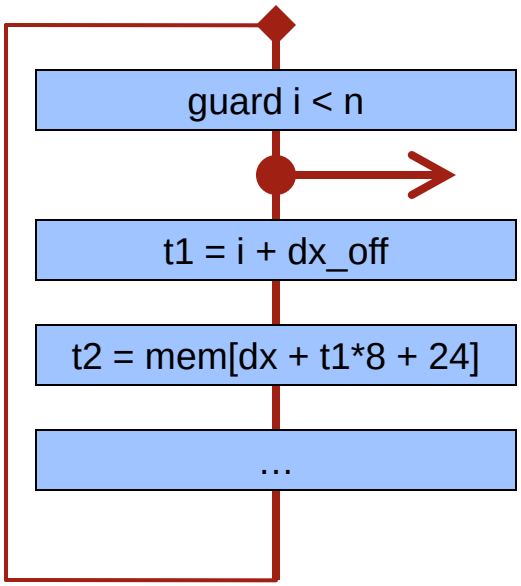


Example from Java Grande Series benchmark (simplified and adapted)

Phase Detection Structure



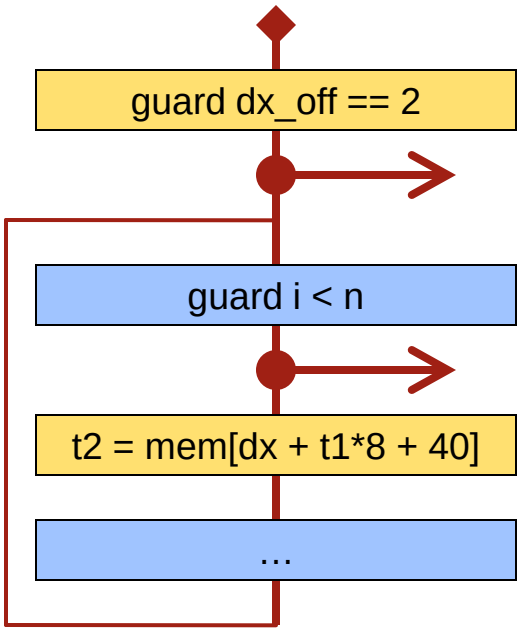
Phases Based on Data Flow



First Phase:
dx_off = 1

Second Phase:
dx_off = 2

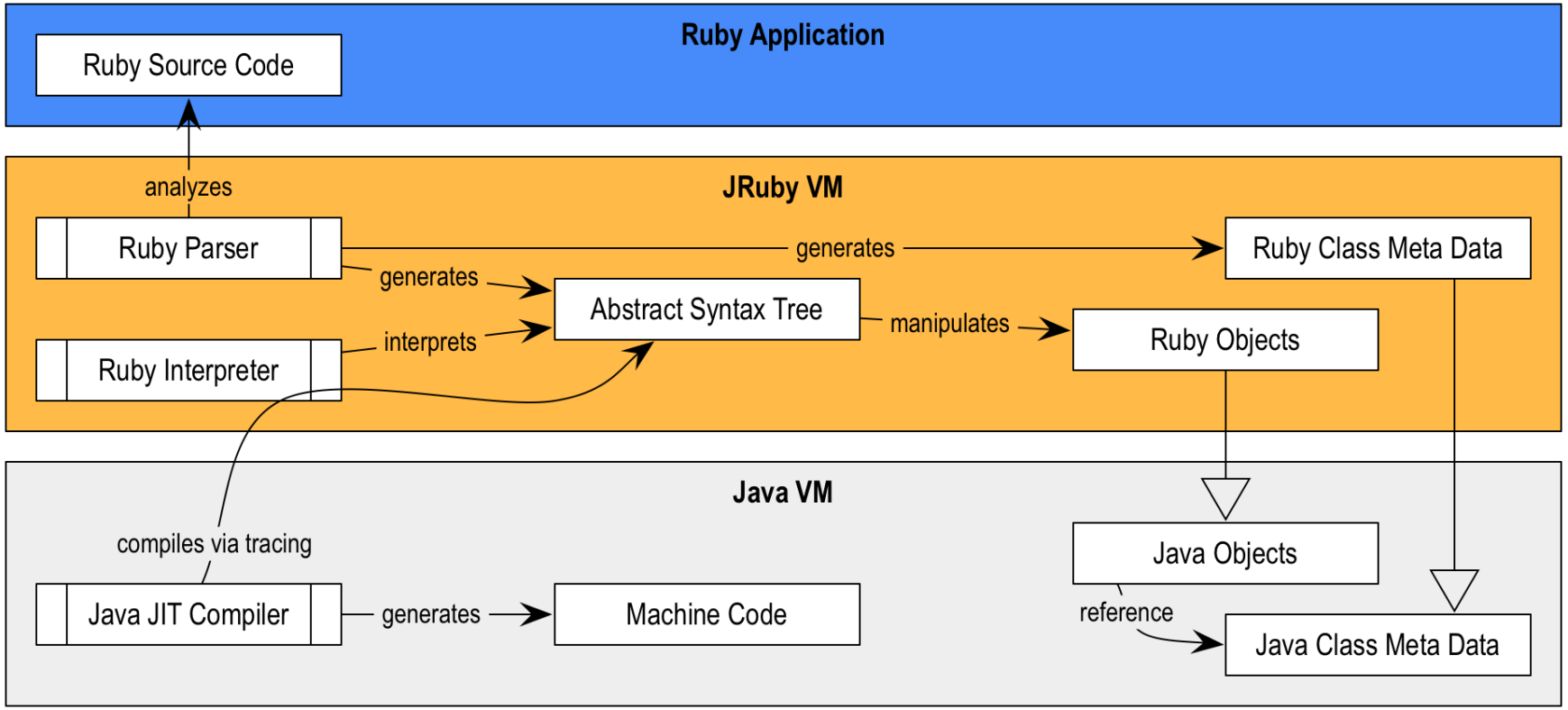
…



Example from Java Grande LUFact benchmark (simplified)

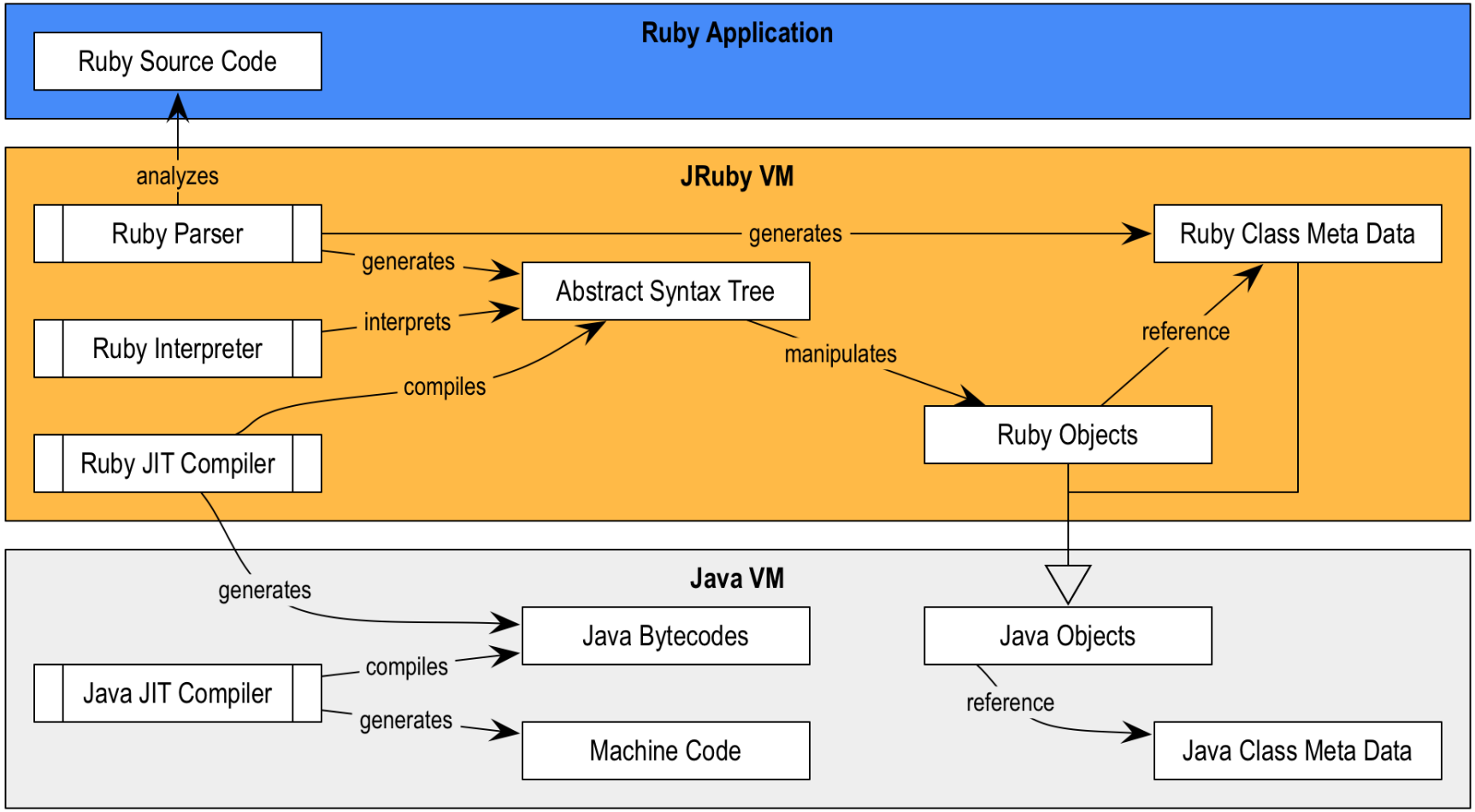
Possibilities of Trace Compilation

- Hierarchical layering of virtual machines

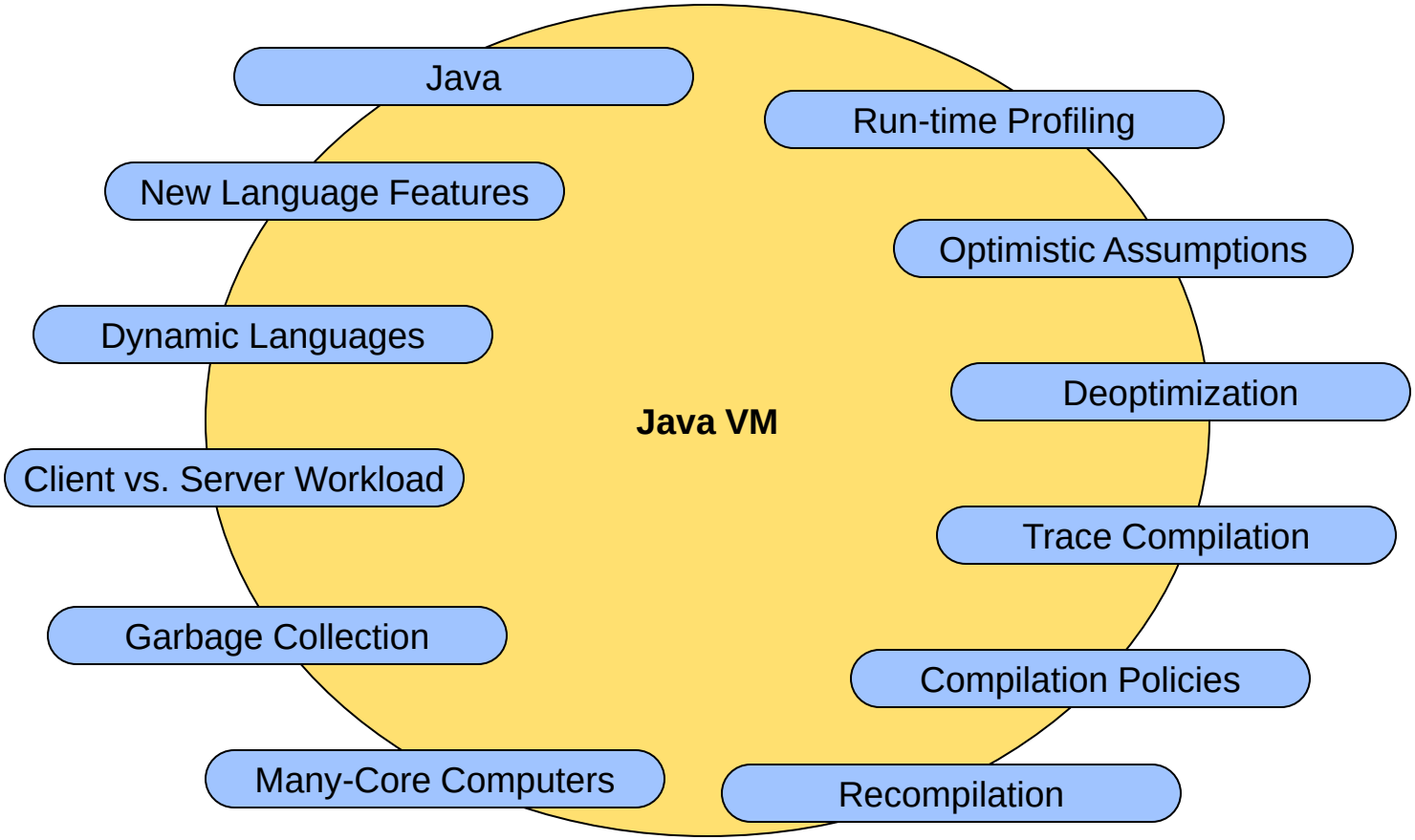


Possibilities of Trace Compilation

- Hierarchical layering of virtual machines – Current structure



Summary


Java
Run-time Profiling
New Language Features
Optimistic Assumptions
Dynamic Languages
Deoptimization
Java VM
Client vs. Server Workload
Trace Compilation
Garbage Collection
Compilation Policies
Many-Core Computers
Recompilation