Compiling Code, Just in Time

IITB CSE Research Symposium

Manas Thakur

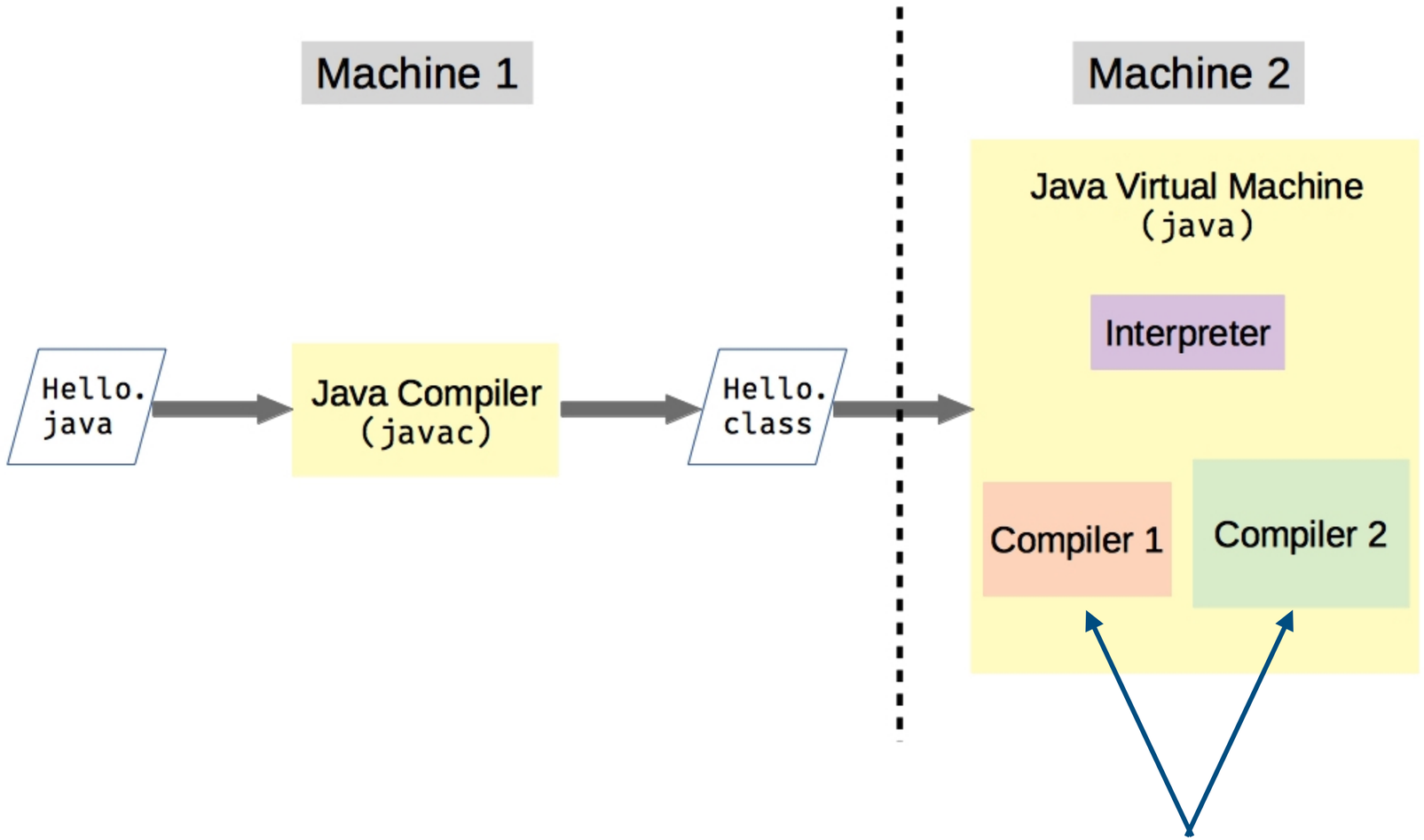
March 26th, 2023

Superfast Java

- Wait! What? Isn't Java supposed to be (super) slow?

- Are Java programs interpreted or compiled?
- Java bytecodes are compiled *just in time*, making modern Java programs run (very) fast.

The Java Compilation+Execution Model



Just-In-Time (JIT) Compilers

Tiered Interpretation and Compilation

- 0: Interpreter with some profiling
- 1: Pure C1
- 2: C1 with invocation and backedge counting
- 3: C1 with full profiling
- 4: C2

JIT Compilation in the HotSpot VM

- Starts off with interpretation of bytecodes
- **Hot spots** get identified by *profiling*:
  - Method invocation counts
  - Backedge counts
- Identified code regions are inserted into a *compilation queue*
- Compiler threads compile methods in the background, while bytecode interpretation continues
- Entry points of methods changed dynamically
- Hot loops are replaced *on-the-stack*
  - called On-Stack Replacement (OSR)
  - **FFT:** What all might OSR involve?

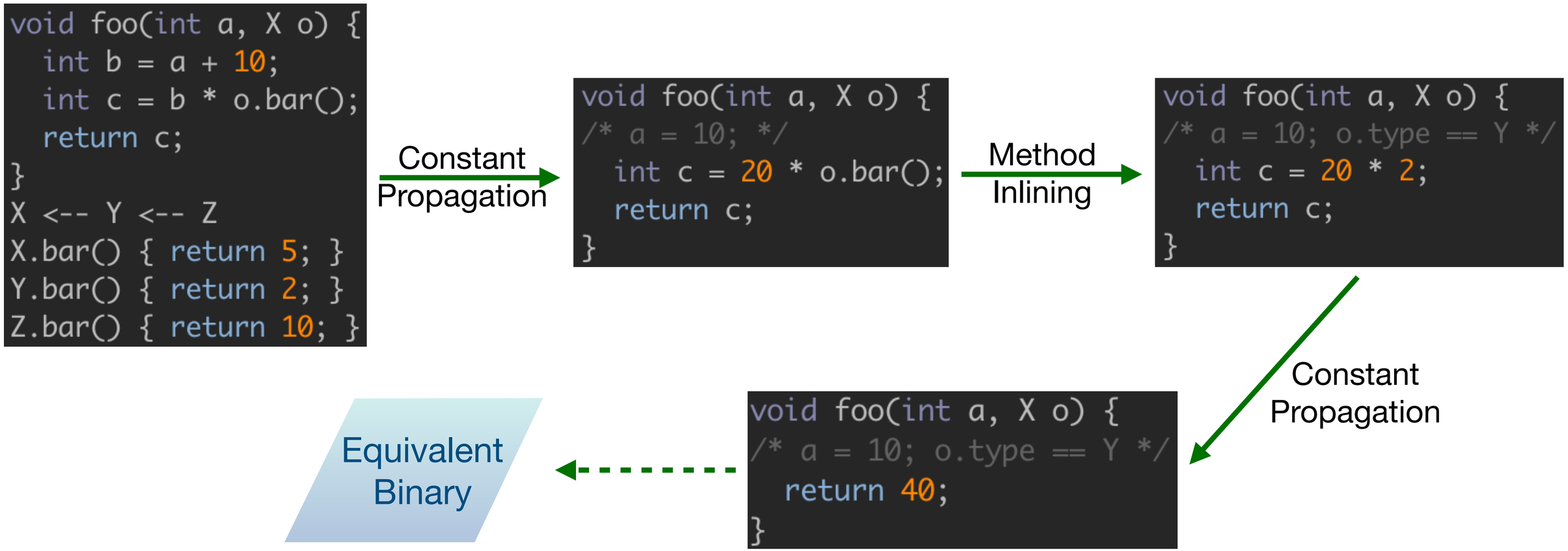
JIT in HotSpot Revisited

- 0: Interpreter with some profiling
- 1: Pure C1
- 2: C1 with invocation and backedge counting
- 3: C1 with full profiling
- 4: C2

Speculative Optimizations

- Why not profile more than just methods calls and backedges?
- **Speculation** based on *live* runtime profile
- Examples:
  - Branch prediction
  - Implicit null checks
  - Monomorphization
  - Method inlining
  - Even newer optimizations!

Specialization using Speculation



- We compile and create a specialized binary under certain assumptions; what if the assumptions fail in a subsequent run?

Speculation and **Deoptimization**

- When a profile-guided assumption fails, the compiled method is invalidated, and the execution falls back to a safe path.
- Which one?
  - Interpretation!
- Compiled method states:
  - `in use, not entrant, zombie, unloaded`

JIT Research Directions

- JIT compilers are heavily resource constrained; how can we make them obtain precise *program analysis* results without affecting the compilation time?

- How can we reduce
  - the cost of deoptimization?
  - the frequency of deoptimization?

- How can we save compilation effort if the program behavior doesn't change (much) within and even across VM instances?

Question of the Symposium

- Which programs are the fastest?
  - Python
  - C
  - C++
  - Java

It's the compiler that makes programs fast.

Research @ **CompL, IITB**

- Precise yet efficient program analysis for languages like Java with staged compilation
- Performing more aggressive optimizations in JITs using static analysis
- Discovering new optimizations for upcoming features such as value types
- Saving compilation effort by recording dynamism in languages like R and Python

➤**Join us!**

https://www.cse.iitb.ac.in/~manas

- Implementation in industry compilers, in collaboration with flagship companies
- Publications at top venues (TOPLAS, OOPSLA, SAS, ECOOP, CC, et cetera)

- **Do you want to be the next COMPLER improving real COMPILERS?**