Uses of the Java Virtual Machine

- The Java byte code is used for several languages other than Java:
  - Scala
  - Ruby
  - Python
  - Lisp
  - Scheme

HotSpot

- The HotSpot virtual machine originates from the Strongtalk virtual machine for the Smalltalk language.
- It was used by Sun research for the Self language.
- The first release as a Java virtual machine was in 1999.
- It is the default virtual machine from Sun/Oracle since Java 1.3.
- Hotspot is written in C++ some assembler, and consist of 250,000 lines.
- Due to HotSpot is partly written in assembler it has triggered the IcedTea project based on HotSpot but without assembler code. Available for example for Power and ARM processors and others.

The Java Byte Code Machine Model

- The JVM is a stack machine.
- This means a byte code instruction pops operands from a stack and pushes the result back to the stack.
- At about the same time as the JVM was designed Bell Labs also designed a virtual machine (for their Inferno operating system) which instead is a register-based virtual machine.
- Register-based virtual machines are easier to produce faster code for, and therefore HotSpot translates the byte code to that.

HotSpot JVM Execution

- Execution of a method starts by interpreting the byte code and after the execution count of the method has reached a limit, optimization is used.
- The whole method is optimized.
- Different optimization levels are used depending on whether the JVM is for desktops (clients) or servers.
- Servers are expected to run for longer time and enables more time-consuming optimizations.
- In addition to the method invocation counter, there are loop iteration counters which also can trigger optimization.

Deoptimization

- The optimization can make guesses and perform better optimizations as long as the guesses are correct.
- For this, runtime checks are inserted to validate the guesses.
- If a guess was wrong, the method is deoptimized and interpreted again, but can be optimized later.
- Deoptimization can also be needed after a new class has been loaded.

Client Optimization

- First the control flow graph of a method is constructed by inspecting the byte codes.
- Then the instructions of a basic block are created by simulating the the JVM execution stack.
- The stack-based execution model of the JVM is thus replaced with the SSA representation.
- This is called the HIR representation, or the high-level intermediate representation.
- Client JVM optimizations on SSA Form include
  - Constant folding
  - Value numbering
  - Inlining

Low-level intermediate representation

- Not SSA Form
- Essentially symbolic assembler code, as in Bell Labs' Inferno
- Unlimited number of machine registers before register allocation

Server HotSpot JVM Execution

- The server JVM also uses SSA Form.
- In addition to the control flow graph, control and data dependencies are analyzed.
- Additional optimizations include:
  - Constant propagation
  - Dead code elimination
  - Instruction scheduling
  - Graph coloring register allocation
  - Loop unrolling
  - Loop invariant code motion

EDA230 Optimizing Compilers in September 2013

- If you are interested in optimizing compilers, there is the course EDA230 in September.
- It is focused on SSA Form and you will start with a subset C compiler which first compiles and then simulates the input C program.
- There you will implement:
  - Lengauer-Tarjan dominance analysis
  - Translation to/from SSA Form
  - Constant propagation on SSA Form
  - Dead code elimination on SSA Form