CS294 Bibliography

This bibliography will be extended as the course progresses.

1 Introduction, overview, terminology

Much of the introductory terminology is taken from the book *Virtual Machines* by James E. Smith and Ravi Nair, Morgan Kaufman, 2005. This book is an excellent introduction to the field, with a superb historical and scholarly overview (but is now 10 years old and could use a refresh).

The *Java Virtual Machine Specification* is available as a download from http://docs.oracle.com/javase/specs/. We will be looking at the Java SE 8 JVM Specification in great detail in a coming lecture and you should download the spec. for reference.

VirtualBox is available from virtualbox.org. There is documentation and source code, but I am unaware of any technical overview of the internals.

Technical information on *Rosetta* is hard to come by; Google and Wikipedia are good starting points.

Dynamo: a transparent dynamic optimization system
Vasanth Bala, Evelyn Duesterwald, Sanjeev Banerjia, POPL 2001
http://dx.doi.org/10.1145/358438.349303
A widely-cited trace-driven binary reoptimizer. Wikipedia (Tracing_just-in-time_compilation) claims it was the first but I believe that honor should go to *Wiggins-Redstone*, done at DEC (it is hard to find descriptions in the literature; there is a presentation at HotChips 11 [1999]). *Shade* (Cmelik and Keppel, 1993), cited in the *Dynamo* paper, incorporated many of the same ideas but for a different purpose, and that paper cites many earlier systems with elements of tracing; the core idea is very old.

The Transmeta Code Morphing™ Software: using speculation, recovery, and adaptive retranslation to address real-life challenges
James C. Dehnert, Brian K. Grant, John P. Banning, Richard Johnson, Thomas Kistler, Alexander Klaiber, Jim Mattson, CGO '03, pp.15—24.
The best overview of the Transmeta binary translator.

Transmeta Breaks x86 Low-power Barrier
Tom R. Halfhill, Microprocessor Report, Feb. 14, 2000

2. Execution mechanisms, Part I: Interpretation

The origins of AST interpretation seem to be lost in the mists of time — I cannot find anything that is plausibly an original reference.

Slim binaries
Michael Franz, Thomas Kistler, CACM 40(12), Dec 1997
http://dx.doi.org/10.1145/265563.265576
A distribution format for programs in the form of compressed ASTs.

3 Anatomy of a Virtual Machine

See section 1 for a reference to the JVM Spec.

Another excellent description of a virtual machine can be found in *Smalltalk-80: The Language and its Implementation*, by Adele Goldberg and Dave Robson, Addison-Wesley, 1983. A free downloadable version is available from the ACM Classic Books web site, http://dl.acm.org/classics.cfm. The last section of the book has an executable specification of the Smalltalk-80 VM, written in Smalltalk.

Java Class Viewer is available from http://www.codeproject.com/Articles/35915/Java-Class-Viewer. I found that when I unzipped the archive, I ended up with filenames with DOS-like embedded paths. After editing those out of the names, I ran the utility like this (all on one line):

```
java -cp JavaClassViewer.jar:CommonLib.jar:FormatCLASS.jar  \
org.freeinternals.javaclassviewer.Main
```

YMMV.

The JNI spec is available at https://docs.oracle.com/javase/8/docs/technotes/guides/jni/spec/jniTOC.html.

4 Bytecode interpretation

An overview of the combined area of abstract and virtual machines can be found in:
Abstract machines for programming language implementation
Stephan Diehl, Pieter Hartel, Peter Sestoft, FGCS 16 (2000)
http://www.inf.ed.ac.uk/teaching/courses/lsi/diehl_abstract_machines.pdf

Branch Prediction and the Performance of Interpreters - Don't Trust Folklore
Erven Rohou, Bharath Narasimha Swamy, André Seznec, CGO 2015
https://hal.inria.fr/hal-01100647
A recent paper measuring interpreter branch prediction performance on modern hardware.

5 JVM bytecodes

Java intermediate bytecodes
James Gosling, ACM SIGPLAN IR'95 workshop on intermediate representations, 1995
http://dx.doi.org/10.1145/202530.202541

6 Dynamic language VMs

Smalltalk

Efficient Implementation of the Smalltalk-80 System
L Peter Deutsch, Allan M Schiffman
POPL 1984, pp 297—302.
http://dx.doi.org/10.1145/800017.800542

Some background on Smalltalk-80:

The August 1981 issue of *Byte* magazine was a special issue on the Smalltalk-80 system. Within can be found many articles about the language, environment and implementation.
https://archive.org/details/byte-magazine-1981-08

The language, the VM and the interactive system are described in two books:

Smalltalk-80: The language and its implementation
Adele Goldberg and David Robson, Addison-Wesley, 1983.
(aka “the Blue Book”)
Available free from at www.acm.org/classics

Smalltalk-80: The Interactive Programming Environment
Adele Goldberg, Addison-Wesley, 1983
(aka “the Orange Book”)
Available from http://www.world.st/learn/books

An additional book collects together implementor’s early experiences:

Smalltalk-80: Bits of history, words of advice
Glenn Krasner (ed), Addison-Wesley, 1983.
(aka “the Green Book”)
Available from http://www.world.st/learn/books

The Design and Evaluation of a High Performance Smalltalk System
David Michael Ungar, MIT Press, 1986
Ungar’s UCB thesis presented and evaluated the hardware and software design of a Smalltalk system on a RISC (SOAR). Of its many contributions, the one with most lasting impact has been the Generation Scavenging automatic storage reclamation technique, still widely used (with some enhancements). Various spin-out papers from the SOAR project described aspects of the work:
Compiling Smalltalk-80 to a RISC http://dx.doi.org/10.1145/36206.36192
SOAR: Smalltalk without bytecodes http://dx.doi.org/10.1145/28697.28708
Architecture of SOAR: Smalltalk on a RISC http://dx.doi.org/10.1145/800015.808182
What Price Smalltalk? http://dx.doi.org/10.1109/MC.1987.1663359
Generation Scavenging: A non-disruptive high performance storage reclamation algorithm http://dx.doi.org/10.1145/800020.808261 (this will be covered in the section on GC)

Self Language

Self: The Power of Simplicity
David Ungar and Randall B. Smith, OOPSLA 1987
http://dx.doi.org/10.1145/38765.38828
A gentle introduction to the language and the abstract model of computation.

The “Self videos” are about 1h10m and 20m long and also (somewhat immodestly) recommended:
http://www.selflanguage.org/

Self VM internal organization

Object storage and inheritance for Self, a prototype-based object-oriented programming language
Elgin Lee, Stanford Engineer’s Thesis, 1988.
The best description of the Self object storage system. I will upload a copy of this to the resources section; it’s hard to get otherwise.

An efficient implementation of SELF, a dynamically-typed object-oriented language based on prototypes
Craig Chambers, David Ungar, Elgin Lee, 1991
http://dx.doi.org/10.1145/74878.74884
The second best reference on the Self object storage system, but much less detailed than Lee’s thesis. Also includes detail in the early Self dynamic compiler.

7 Memory management

There are two excellent textbooks covering garbage collection:

Garbage Collection
Richard Jones with Rafael Lins, Wiley 1999 (2nd printing).

The Garbage Collection Handbook
Richard Jones, Antony Hosking, Eliot Moss, CRC Press 2012.

The latter is an updated and extended version of the former, and is preferable, if you have to pick one. The former has more detail on some techniques of historical interest.
In the GC Handbook, Chapter 1 covers mark-sweep, Chapter 3 covers mark-compact, Chapter 4 covers copying collection, Chapter 5 covers reference counting and Chapter 7 cover allocation. Richard Jones maintains a web site about memory management with a comprehensive bibliography: http://www.cs.kent.ac.uk/people/staff/rej/gc.html

The assigned reading is *Generation Scavenging: A non-disruptive high performance storage reclamation algorithm,* David Ungar, 1984.
http://dx.doi.org/10.1145/800020.808261

The pauseless GC algorithm
Cliff Click, Gil Tene and Michael Wolf, VEE '05
http://dx.doi.org/10.1145/1064979.1064988

MMU was introduced in:
A parallel, real-time garbage collector
Perry Cheng and Guy Blelloch, PLDI 2001
http://dx.doi.org/10.1145/381694.378823

BMU was introduced in:
MC2: High-Performance Garbage Collection for Memory-Constrained Environments
Narendran Sachindran J. Eliot B. Moss Emery D. Berger, OOPSLA 2004
http://dx.doi.org/10.1145/1035292.1028984

Here are a couple of recent papers on the energy impact of GC:
Impact of GC design on power and performance for Android
Ahmed Hussein, Mathias Payer, Antony Hosking and Christopher A Vick, SYSTOR '15
http://dx.doi.org/10.1145/2757667.2757674

Don't race the memory bus: taming the GC leadfoot
Ahmed Hussein, Antony Hosking, Mathias Payer and Christopher A Vick, ISMM 2015
http://dx.doi.org/10.1145/2754169.2754182

8 Advanced interpretation

Threaded code
James R Bell, CACM 1973 16(6)
http://dx.doi.org/10.1145/362248.362270

Indirect threaded code
Robert Dewar, CACM 1975 18(6)
http://dx.doi.org/10.1145/360825.360849

Stack caching for interpreters
M Anton Ertl, PLDI 95
http://dx.doi.org/10.1145/207110.207165

Combining stack caching with dynamic super instructions
M Anton Ertl and David Gregg, IVME '04
http://dx.doi.org/10.1145/1059579.1059583

The Structure and Performance of Efficient Interpreters
M Anton Ertl and David Gregg
http://www.complang.tuwien.ac.at/anton/tmp/interpreter-arch.ps

Optimizing direct threaded code by selective inlining
Ian Piumarta and Fabio Riccardi, PLDI 98
http://dx.doi.org/10.1145/277652.277743

vmgen — A Generator of Efficient Virtual Machine Interpreters
M Anton Ertl, David Gregg, Andreas Krall, Bernd Paysan, SP&E 2002 32(3)
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.98.873&rep=rep1&type=pdf

9 Dynamic compilation

A Brief History of Just-In-Time
John Aycock, Computing Survey, 2003
http://dx.doi.org/10.1145/857076.857077
A survey of the dynamic compilation literature.

Efficient Implementation of the Smalltalk-80 System
L Peter Deutsch, Allan M Schiffman
POPL 1984, pp 297—302.
http://dx.doi.org/10.1145/800017.800542
Described the first JIT compiler and inline caches.

Self VM — compilation-related

Optimizing Dynamically-Typed Object-Oriented Languages With Polymorphic Inline Caches
Urs Hölzle, Craig Chambers and David Ungar, ECOOP 91
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.36.6379&rep=rep1&type=pdf
A generalization of inline caching, paving the way for adaptive optimization.

Customization: optimizing compiler technology for SELF, a dynamically-typed object-oriented programming language
Craig Chambers and David Ungar, PLDI '89
http://dx.doi.org/10.1145/74818.74831
http://dx.doi.org/10.1145/989393.989425 (including a retrospective)

Debugging optimized code with dynamic deoptimization
Urs Hölzle, Craig Chambers and David Ungar, PLDI '92
http://dx.doi.org/10.1145/143103.143114
The canonical reference on deoptimization.

Adaptive optimization for Self: Reconciling High Performance with Exploratory Programming
Urs Hölzle, Stanford Ph.D. thesis and Sun Labs technical report, 1994
http://bibliography.selflanguage.org/urs-thesis.html
The best complete overview of the last generation of the Self VM. It is summarized in:
A Third-Generation Self Implementation: Reconciling Responsiveness with Performance
Urs Hölzle and David Ungar, OOPSLA 94
http://dx.doi.org/10.1145/191081.191116

A complete list of Self publications is at bibliography.selflanguage.org.

HotSpot JVM

A simple graph-based intermediate representation
Cliff Click and Michael Paleczny
http://dx.doi.org/10.1145/202530.202534

The Java HotSpot Server Compiler
Michael Paleczny, Chris Vick and Cliff Click, JVM'01
https://www.usenix.org/legacy/events/jvm01/full_papers/paleczny/paleczny.pdf

Fast subtype checking in the HotSpot JVM
Cliff Click and John Rose, Java Grande 2002
http://dx.doi.org/10.1145/583810.583821

Design of the Java HotSpot™ client compiler for Java 6
Kotzmann et al, TACO 5(1) 2008
http://dx.doi.org/10.1145/1369396.1370017

Miscellaneous compilation and optimization techniques

Optimization of Object-Oriented Programs Using Static Class Hierarchy Analysis
Jeffrey Dean, David Grove, and Craig Chambers, ECOOP 95
http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.117.2420

Fast static analysis of C++ virtual function calls
David F Bacon and Peter F Sweeney, OOPSLA 96
http://dx.doi.org/10.1145/236337.236371
Introduces Rapid Type Analysis, another technique for devirtualizing calls.

Design, implementation and evaluation of adaptive recompilation with on-stack replacement
Stephen J Fink and Feng Qian, CGO '03
http://dl.acm.org/citation.cfm?id=776261.776288&coll=DL&dl=GUIDE&CFID=545999259&CFTOKEN=65787538

Partial Escape Analysis and Scalar Replacement for Java
Lukas Stadler, Thomas Würthinger and Hanspeter Mössenböck, CGO 14
http://dx.doi.org/10.1145/2544137.2544157

Automatic Construction of Inlining Heuristics using Machine Learning
Sameer Kulkarni, John Cavazos, Christian Wimmer and Douglas Simon, CGO 2013
http://dx.doi.org/10.1109/CGO.2013.6495004

Speculation without regret: reducing deoptimization meta-data in the Graal compiler
Gilles Duboscq, Thomas Würthinger and Hanspeter Mössenböck, PPPJ '14
http://dx.doi.org/10.1145/2647508.2647521

Tracing and meta-tracing

See the Dynamo paper (above, under *Introduction*) for an earlier description of trace compilation.

HotpathVM: an effective JIT compiler for resource-constrained devices
Andreas Gal, Christian Probst and Michael Franz, VEE ‘06
http://dx.doi.org/10.1145/1134760.1134780

PyPy's approach to virtual machine construction
Armin Rigo and Samuel Pedroni, OOPSLA ’06
http://dx.doi.org/10.1145/1176617.1176753

Tracing the meta-level: PyPy's tracing JIT compiler
Carl Friedrich Bolz, Antonio Cuni, Maciej Fijalkowski and Armin Rigo, ICOOOLPS '09
http://dx.doi.org/10.1145/1565824.1565827

Runtime feedback in a meta-tracing JIT for efficient dynamic languages
Carl Friedrich Bolz et al., ICOOOLPS 11
http://dx.doi.org/10.1145/2069172.2069181

10 Metacircular VMs

See *Smalltalk-80: The language and its implementation* (under the Smalltalk section, above) for a metacircular Smalltalk bytecode interpreter.

Implementing a Java Virtual Machine in the Java Programming Language
Antero Taivalsaari, Sun Microsystems Laboratories Technical Report TR-98- 64, March 1998.
http://dl.acm.org/citation.cfm?id=974968
A Java bytecode interpreter written in Java.

The Wikipedia pages referenced in the lectures mention dozens of language implementations stacked on top of the JVM and CLI. Here a couple of earlier such stacked languages, on top of Self:

self includes: Smalltalk
Mario Wolczko, in *Prototype-Based Programming*, Noble, Taivalsaari, Moore (eds), Springer, 1999.
http://merlintec.com/download/mario.pdf
Describes an implementation of Smalltalk-80 atop the Self VM.

Design and implementation of Pep, a Java just-in-time translator
Ole Agesen, Theory and Practice of Object Systems, 3(2), 1997
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.22.1089&rep=rep1&type=ps
An implementation of the JVM atop the Self VM.

Back to the future: the story of Squeak, a practical Smalltalk written in itself
Dan Ingalls, Ted Kaehler, John Maloney, Scott Wallace, Alan Kay, OOPSLA 97
http://dx.doi.org/10.1145/263700.263754

SqueakJS: A modern and practical Smalltalk that runs in any browser
Bert Freudenberg at al., DLS 14
http://dx.doi.org/10.1145/2775052.2661100
Squeak on JavaScript!

A Java virtual machine architecture for very small devices
Nik Shaylor, Douglas N. Simon, William R Bush, LCTES 03
http://dx.doi.org/10.1145/780731.780738
The Squawk JVM, a small JVM implemented in Java and translated to C, inspired by Squeak

The Jalapeño Dynamic Optimizing Compiler for Java
Burke et al., Java Grande 1999
http://dx.doi.org/10.1145/304065.304113
The first paper about the system which would eventually be named Jikes RVM.

Adaptive optimization in the Jalapeño JVM
Matthew Arnold, Stephen Fink, David Grove, Michael Hind and Peter Sweeney, OOPSLA 2000
http://dx.doi.org/10.1145/354222.353175

Demystifying Magic: High-level Low-level Programming
Frampton et al., VEE '09
http://dx.doi.org/10.1145/1508293.1508305
An overview of the techniques used to develop low-level code (VMs, GC) in a HLL (Java)

Oil and Water? High Performance Garbage Collection in Java with MMTk
Stephen M. Blackburn, Perry Cheng, Kathryn M. McKinley, ICSE '04
http://dl.acm.org/citation.cfm?
id=998675.999420&coll=DL&dl=GUIDE&CFID=545999259&CFTOKEN=65787538

Constructing a metacircular virtual machine in an exploratory programming environment
David Ungar, Adam Spitz and Alex Ausch, OOPSLA '05
http://dx.doi.org/10.1145/1094855.1094865
Describes the Klein VM.

Maxine: An approachable virtual machine for, and in, Java
Christian Wimmer et al., TACO 9(4) 2013
http://dx.doi.org/10.1145/2400682.2400689

Snippets: Taking the High Road to a Low Level
Doug Simon et al., TACO 12(2) 2015
http://dx.doi.org/10.1145/2764907

Truffle/Graal and related topics

Partial Evaluation of Computation Process – An Approach to a Compiler-Compiler
Yoshihiko Futamura, Higher-Order and Symbolic Computation 12, 381–391 (1999)
http://www.brics.dk/~hosc/local/HOSC-12-4-pp381-391.pdf
The original description of partial evaluation. Don't be fooled by the '99 publication date — the original (in Japanese) was from the 1970s. See Futamura's web site: http://fi.ftmr.info/

One VM to rule them all
Wuerthinger et al., Onward! 2013
http://dx.doi.org/10.1145/2509578.2509581

Truffle and Graal papers and presentations can be found at
https://wiki.openjdk.java.net/display/Graal/Publications+and+Presentations

Truffle tutorial video at SPLASH 2014, Christian Wimmer
Video (90mins, SimpleLanguage begins at 22:25) Slides

Truffle/Graal tutorial at DSLDI 2015, Thomas Würthinger, Christian Humer
Videos: Part1 Part2 (2x90mins)
The first hour of Part 2 is about Truffle. Part 1 is more about Graal; jump at 25mins — it will show the low-level detail of Graal code generation for SimpleLanguage, and includes an introduction to the Ideal Graph Visualizer.

Self-Optimizing AST Interpreters
Thomas Würthinger, Andreas Wöß, Lukas Stadler, Gilles Duboscq, Doug Simon, Christian Wimmer, DLS '12

A Domain-Specific Language for Building Self-Optimizing AST Interpreters
Christian Humer, Christian Wimmer, Christian Wirth, Andreas Wöß, Thomas Würthinger, GPCE'14
An overview of the Truffle DSL. The current version has been enhanced since the paper. For details, see http://mail.openjdk.java.net/pipermail/graal-dev/2015-February/002912.html. The change log is here.

Truffle tutorial videos at CGO 2014, Christian Wimmer
Videos: Part 1, Part 2 (2x90mins) Slides

An Object Storage Model for the Truffle Language Implementation Framework
Andreas Wöß, Christian Wirth, Daniele Bonetta, Chris Seaton, Christian Humer and Hanspeter Mössenböck, PPPJ '14, http://dx.doi.org/10.1145/2647508.2647517

Dynamically composing languages in a modular way: supporting C extensions for dynamic languages
Matthias Grimmer, Chris Seeton, Thomas Würthinger and Hanspeter Mössenböck, MODULARITY 2015, http://dx.doi.org/10.1145/2724525.2728790

Concurrency

Eliminating synchronization-related atomic operations with biased locking and bulk rebiasing
Kenneth Russell and David Detlefs, OOPSLA '06
http://dx.doi.org/10.1145/1167515.1167496

See Jones, Hosking and Moss (*GC Handbook*, above under Memory Management) for descriptions and references on concurrent and parallel GC.

Tools

Building Debuggers and Other Tools: We Can "Have it All": A Position Paper
Michael L. Van De Vanter, ICOOOLPS 2015
http://vandevanter.net/mlvdv/publications/2015-icooolps.pdf

Debugging at Full Speed
Chris Seaton, Michael L. Van De Vanter, and Michael Haupt, Dyla'14
http://dx.doi.org/10.1145/2617548.2617550

System VMs

Formal requirements for virtualizable third generation architectures
Gerald J. Popek and Robert P. Goldberg, CACM: Volume 17 Issue 7, July 1974
http://dx.doi.org/10.1145/361011.361073

The Evolution of an x86 Virtual Machine Monitor
Ole Agesen, Alex Garthwaite, Jeffrey Sheldon, Pratap Subrahmanyam, SIGOPS 44(4), 2010.
http://dx.doi.org/10.1145/1899928.1899930

See also the Smith and Nair book (first entry under Introduction, above).