



FOSDEM 2007 Java HotSpot™ Virtual Machine

Peter B. Kessler

Senior Staff Engineer

Sun Microsystems, Inc.

What Does the Virtual Machine Do?

- Bytecode execution
 - > Interpreter (also does profiling)
 - > 2 runtime compilers: `-client` and `-server`
 - > On-stack replacement (OSR)
- Storage allocation and garbage collector
 - > Fast inline allocation; concurrent, parallel collectors
- Runtimes
 - > Start up, shut down, class loading, threads, synchronization, safepoints, interactions with operating system, *etc.*

Interactions with the Platform

- Few API's
 - > jni.h
 - > JVM/TI
- “Intrinsification” of hot, or easy, or critical methods
 - > java.lang.Math.sin
 - > java.lang.String.indexOf
 - > sun.misc.Unsafe.compareAndSwapInt
- Mostly implementation
 - > Can be on a different release cycle than the JRE

Some Things to Know (1)

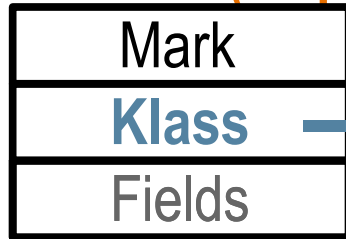
Mark word in every object

| Bitfields | | | Tag | State |
|--------------------------|-----|---|-----|---------------------|
| Hashcode | Age | 0 | 01 | Unlocked |
| Lock record address | | | 00 | Light-weight locked |
| Monitor address | | | 10 | Heavy-weight locked |
| Forwarding address, etc. | | | 11 | Marked for GC |
| Thread ID | Age | 1 | 01 | Biased / biasable |

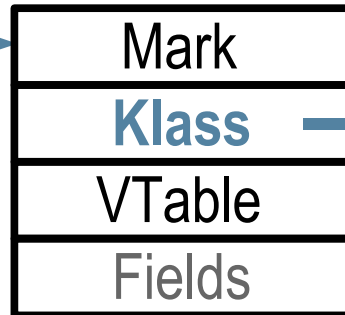
Some Things to Know (2)

Class in every object

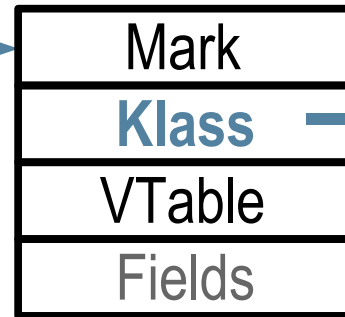
instance(OopDesc)



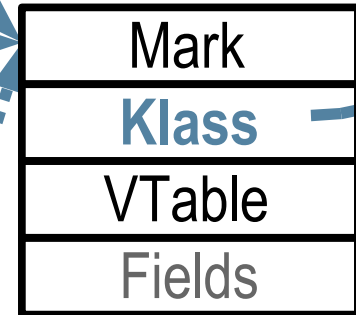
instanceKlass



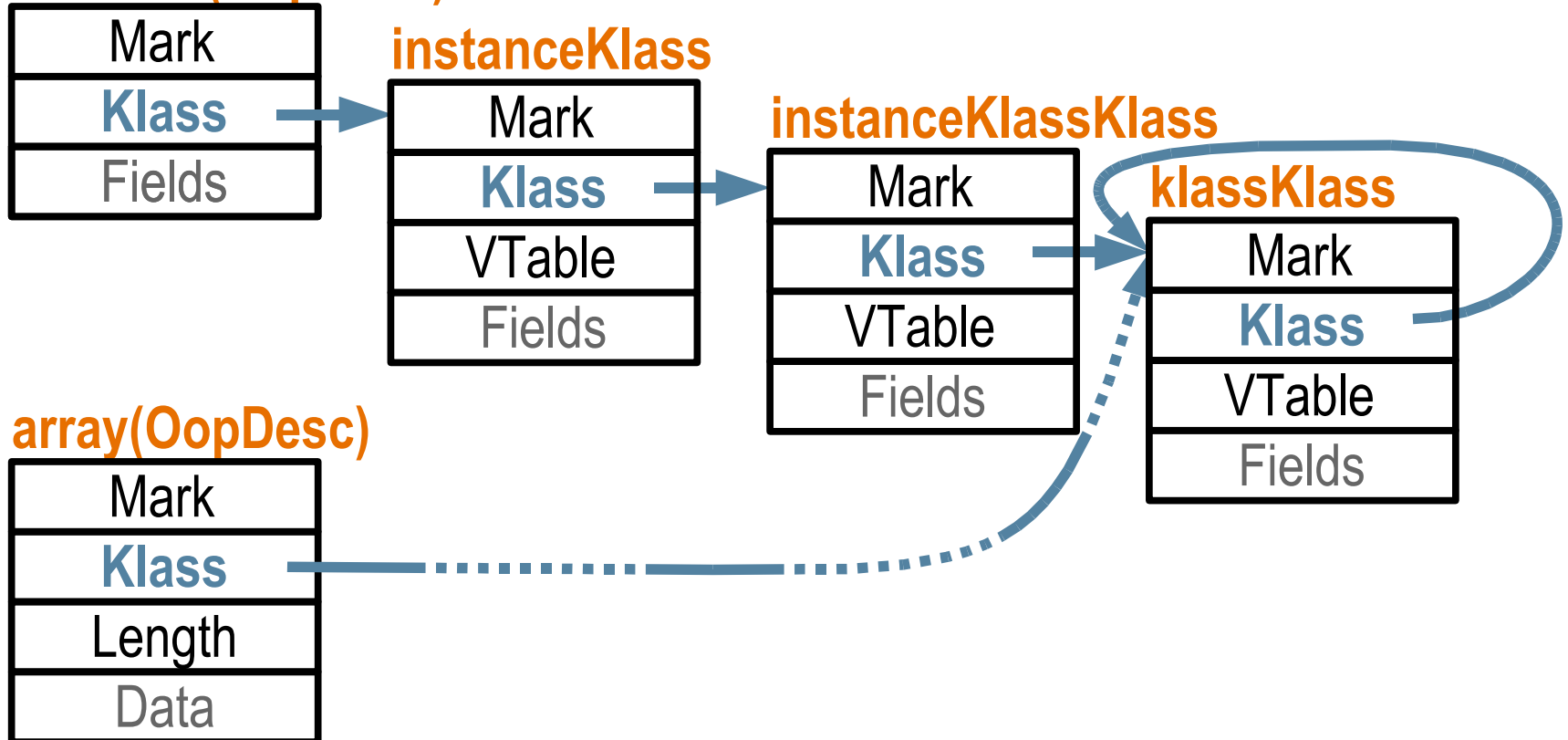
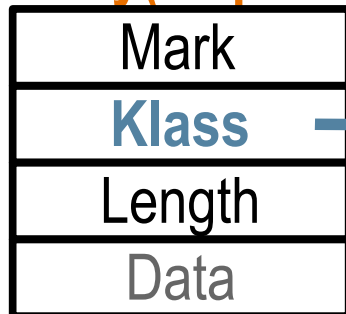
instanceKlassKlass



klassKlass



array(OopDesc)



Process

- Plan features, or get bug reports
- Discuss with group, *etc.*
- Write code
- Test for correctness, performance impact
- Code review by 2 other engineers
- Nightly testing in group workspace
- Integration testing in VM workspace
- “Big Apps” tests, cross-platform performance tests
- Backport (if necessary)

Actual Code!

- Coming up on 10 years of coding
- 188K lines of code
- 136 contributors + sponsored changes
 - > *You could be next!*
- The virtual machine is already out under GPLv2
- Competition drives innovation
 - > So do real applications and customers
- What's fun about it?
 - > Complex multithreaded program that has to work perfectly!

Outside Projects

- University of Linz, Austria
 - > “Escape Analysis in the Context of Dynamic Compilation and Deoptimization”, Thomas Kotzmann, Ph.D.
 - > “Linear Scan Register Allocation for the Java HotSpot? Client Compiler”, Christian Wimmer, M.S., 2004
 - > Others papers with Hanspeter Mössenböck
- Doug Lea, SUNY Oswego
 - > Changes for JSR-133: memory model and threads

Further Reading

- http://java.sun.com/javase/technologies/hotspot/gc/memorymanagement_whitepaper.pdf
- T. Printezis and D. L. Detlefs. “A Generational Mostly-Concurrent Garbage Collector”. In A. L. Hosking, editor, *Proceedings of the 2000 International Symposium on Memory Management (ISMM 2000)*, pages 134-154, Minneapolis, MN, USA, October 2000. ACM Press.

Blogs

- <http://blogs.sun.com/jonthecollector>
- <http://blogs.sun.com/dagastine>
- <http://blogs.sun.com/tony>



FOSDEM 2007 Java HotSpot™ Virtual Machine

Peter B. Kessler
Senior Staff Engineer
Sun Microsystems, Inc.

What Does the Virtual Machine Do?

- Bytecode execution
 - > Interpreter (also does profiling)
 - > 2 runtime compilers: `-client` and `-server`
 - > On-stack replacement (OSR)
- Storage allocation and garbage collector
 - > Fast inline allocation; concurrent, parallel collectors
- Runtimes
 - > Start up, shut down, class loading, threads, synchronization, safepoints, interactions with operating system, *etc.*

2

Start out with interpreter:

- **Get running quickly**
- **Interpreter gathers profiling to guide compilation**

Client compiler:

- **Inlining based on CHA**
- **CFG + per-basic-block SSA**
- **Linear scan register allocator**

Server compiler:

- **Inlining based on CHA plus profiling**
- **SSA-like “sea of nodes”**
- **Global code motion, loop opts, scheduling**

**Optimistic optimizations and back-out and recompile
(Further reading for GC)**

Lots of fiddly details in runtimes:

- **signals**
- **thread stacks**

Interactions with the Platform

- Few API's
 - > jni.h
 - > JVM/TI
- “Intrinsification” of hot, or easy, or critical methods
 - > java.lang.Math.sin
 - > java.lang.String.indexOf
 - > sun.misc.Unsafe.compareAndSwapInt
- Mostly implementation
 - > Can be on a different release cycle than the JRE

3

The VM has few API's to it.

- **The obvious ones for jni and tools**
- **The less obvious ones**
where we know the semantics (but check!)

**So we change change the implementations
more frequently than the platform rev's.**

Some Things to Know (1)

Mark word in every object

| Bitfields | | | Tag | State |
|--------------------------|-----|---|-----|---------------------|
| Hashcode | Age | 0 | 01 | Unlocked |
| Lock record address | | | 00 | Light-weight locked |
| Monitor address | | | 10 | Heavy-weight locked |
| Forwarding address, etc. | | | 11 | Marked for GC |
| Thread ID | Age | 1 | 01 | Biased / biasable |

4

**Most objects start off in “01”
- unlocked, unbiased**

**Move to “11” during GC
when forwarding pointer is installed**

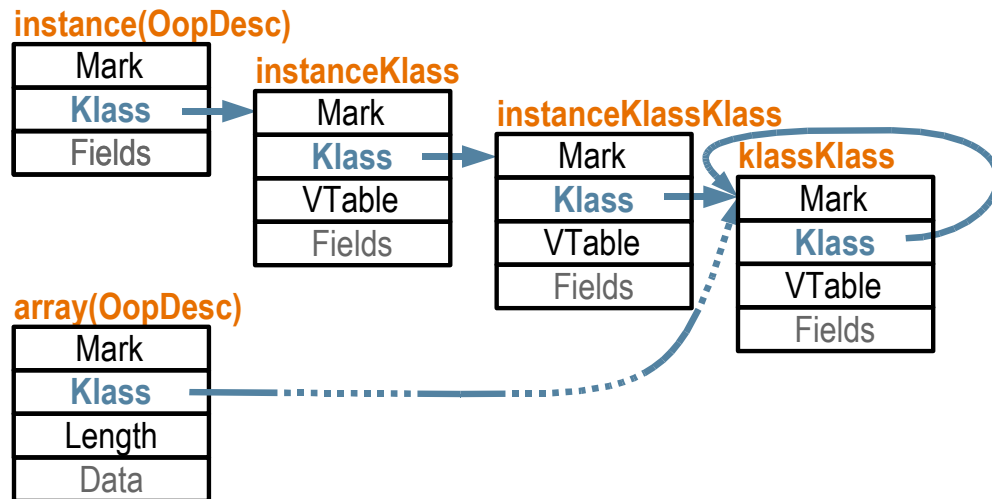
**Lightweight lock (typical!) uses “00”
with pointer to thread stack slot
for displaced header**

**Heavy lock uses “10”
with pointer to lock structure**

**Biased locking (from IBM TRL paper) uses “101”
with thread id and “age” to avoid CAS
on uncontended lock enter/exit**

Some Things to Know (2)

Class in every object



5

Every collectable object (java or not) has 2 word header

First word is “mark” described above

Second word is like C++ vtable slot

- pointer to object that knows how to
- invoke virtual functions on object
- holds data for those operations

Meta-data in “permanent” generation

fooOopDesc and pointer to it as fooOop

fooOopDesc versus fooKlass

Ugly code: translate from classOop to Klass*

For example:

Instance holds instance fields

instanceClass holds static fields and oopMap for instances

instanceClassClass holds oopMap for instanceClass

classClass holds oopMap for instanceClassClass (and self!)

Arrays, etc., have their own hierarchy up to classClass

Process

- Plan features, or get bug reports
- Discuss with group, *etc.*
- Write code
- Test for correctness, performance impact
- Code review by 2 other engineers
- Nightly testing in group workspace
- Integration testing in VM workspace
- “Big Apps” tests, cross-platform performance tests
- Backport (if necessary)

6

Long planning process, short bug escalation cycle

Discussions with GC group, VM group, JAT, CCC, project team

Targeted tests, PRT

- **Don't break the integration workspace!**
- **Can't have all platforms on my desktop**
- **Labs with big, exotic, machines**
- **Get feedback early**

Longevity tests, tiered platform tests, manual tests

Not everything is backported

Actual Code!

- Coming up on 10 years of coding
- 188K lines of code
- 136 contributors + sponsored changes
 - > *You could be next!*
- The virtual machine is already out under GPLv2
- Competition drives innovation
 - > So do real applications and customers
- What's fun about it?
 - > Complex multithreaded program that has to work perfectly!

7

Outside Projects

- University of Linz, Austria
 - > “Escape Analysis in the Context of Dynamic Compilation and Deoptimization”, Thomas Kotzmann, Ph.D.
 - > “Linear Scan Register Allocation for the Java HotSpot? Client Compiler”, Christian Wimmer, M.S., 2004
 - > Others papers with Hanspeter Mössenböck
- Doug Lea, SUNY Oswego
 - > Changes for JSR-133: memory model and threads

8

**These were all done before we were open source
Changes don't have to be this major!**

**We've also had contributions of fixes to our
build process for new gcc releases, etc.**

Further Reading

- http://java.sun.com/javase/technologies/hotspot/gc/memorymanagement_whitepaper.pdf
- T. Printezis and D. L. Detlefs. "A Generational Mostly-Concurrent Garbage Collector". In A. L. Hosking, editor, *Proceedings of the 2000 International Symposium on Memory Management (ISMM 2000)*, pages 134-154, Minneapolis, MN, USA, October 2000. ACM Press.

Blogs

- <http://blogs.sun.com/jonthecollector>
- <http://blogs.sun.com/dagastine>
- <http://blogs.sun.com/tony>

9

Also:

D. L. Detlefs, C. H. Flood, S. Heller, and T. Printezis. "Garbage-First Garbage Collection". In A. Diwan, editor, *Proceedings of the 2004 International Symposium on Memory Management (ISMM 2004)*, pages 37-48, Vancouver, Canada, October 2004. ACM Press.

And the usual openjdk.dev.java.net mailing lists.