Lilliput: Tiny Classpointers

A 10-minute speed run

Thomas Stüfe
Principal Engineer
tstuefe@redhat.com
## Motivation

Class Pointers take a lot of space...

<table>
<thead>
<tr>
<th>63</th>
<th>32</th>
<th>31</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class Pointer</td>
<td>Hash</td>
<td>Age Fw d L c k</td>
<td></td>
</tr>
</tbody>
</table>

---

**Note:** The table and diagram illustrate the layout of class pointers, hash, and age fields in a memory structure.
Motivation

...we need to make them smaller
What is a Class Pointer ?
Class and Class Metadata

Java Heap

- Header
- Object
- Class

Native Memory (Metaspace)

- Klass
  - yes, with a 'K'
  - var-sized...
- CP
- Method
- Annotations
- Counters
- Byte codes
We already compress Class Pointers (since JDK 8)

Klass* is 64-bit - too much.

We split Klass* into 64-bit base and 32-bit offset. We only store the offset in the object headers.

"Encoding Base"
Runtime-constant, determined at VM start

"Narrow" or "Compressed" Class pointer
32-bit offset?

⇒ all Klass must be confined to a 4GB(*) range.

⇒ **class space** : an enclosure for Klass structures

(*) Yes, I am ignoring the encoding shift
... and CDS

Same goes for CDS.

We place CDS archived metadata close to the class space.
Decoding

Raw Klass Pointer = **Encoding Base** + Offset (narrow Klass Pointer) (*)

- C++ : Base is a runtime value
- JIT: Base is a constant (64-bit immediate)

Many optimizations exists per CPU that depend on a “good” Base.

(*) still ignoring encoding shift
CPU-specific encoding bases

- RiscV: bits set only in \([12-32]\) (for \textit{lui}) or \([32-44]\) (\textit{addiw+slli})
- Arm64: Either a logical immediate aligned to 4GB (\textit{eor}) or bits in the third quadrant only (\textit{movk})
- S390: Prefer <4GB addresses (\textit{algfi}) or bits restricted to a single quadrant
- x64: Prefer < 4GB for the short form of \textit{mov} immediate
- PPC: Restrict bits to as few quadrants as possible
Optimization Example: unscaled encoding

If base is zero, we can omit the load immediate altogether.

JVM tries really hard to reserve class space in low address regions (even harder in JDK 22+).
Lilliput: 22-bit
Side Goals

- Address “enough” classes
- Contain invasiveness of patch:
  - Lilliput will need to coexist with legacy JVM for some time
  - ⇒ Keep Klass layout (for now)
  - ⇒ Keep using CDS + Metaspace
How many classes can we address today?

~5 million classes (*)
- 3GB class space
- Average Klass size ~6xx bytes

Using 3 GB class space would cost ~30 GB of Non-Class Metaspace!

(* without CDS)
How many classes do we need to address?

Normal case: x*100 .. x*1000, very large applications: x*100_000.
But we need to cater to weird corner cases too (generator cases).
Anything in the multi-million range is fine.

⇒ don’t reduce (for now) Klass encoding range size. Keep it at 4GB.
Increase Alignment

We can increase Klass* alignment and re-purpose the alignment shadow bits:

```
xxxxxxx-xxxxxxx-xxxxx00-00000000
```
10-bit alignment

Why 10 bit (1 KB)?

On average:

> 80% of Klass between 512 byte and 1K;

> 95% of Klass smaller than 1K.
22-bit Class Pointers

22 bits let us address **3 million** classes (*)

⇒ Klass needs 1 KB on average

⇒ Class space capped at 3 GB

(* without CDS)
Class Space morphs into a Table

Class Space

<table>
<thead>
<tr>
<th>Klass</th>
<th>Klass</th>
<th>K...</th>
<th>Klass</th>
<th>Klass</th>
<th>K...</th>
</tr>
</thead>
</table>

CDS

<table>
<thead>
<tr>
<th>Klass</th>
<th>Klass</th>
<th>K...</th>
</tr>
</thead>
</table>

nKlass

1  2  3  4  ...

1K  1K  1K  1K  1K  1K
22-bit narrow Klass Pointers

... but fragmentation hurts

Class Space

<table>
<thead>
<tr>
<th>Klass</th>
<th>Klass</th>
<th>K..</th>
<th>Klass</th>
<th>Klass</th>
</tr>
</thead>
</table>

CDS

<table>
<thead>
<tr>
<th>Klass</th>
<th>Klass</th>
<th>K.</th>
</tr>
</thead>
</table>

nKlass

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>...</th>
</tr>
</thead>
</table>

Klass

| 1K | 1K | 1K | 1K | 1K |

CDS

| 1K | 1K | 1K | 1K |

Red Hat
Make Metaspace alignment-aware

22-bit narrow Klass Pointers

It works beautifully: (almost) zero footprint degradation.
Statistics

Klass: **Few (relatively), coarse-grained**

Non-Klass: **Numerous, fine-grained**
New Markword Layout (for now...)

<table>
<thead>
<tr>
<th>Before:</th>
<th>Now (for now):</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>63</td>
</tr>
<tr>
<td>32-bit Class Pointer</td>
<td>22-bit Class Pointer</td>
</tr>
<tr>
<td>31</td>
<td>42 41</td>
</tr>
<tr>
<td>25-bit Hash</td>
<td>31-bit Hash</td>
</tr>
<tr>
<td>Age</td>
<td>free</td>
</tr>
<tr>
<td>Fwd</td>
<td>Age</td>
</tr>
<tr>
<td>Lock</td>
<td>Fwd</td>
</tr>
<tr>
<td></td>
<td>Lock</td>
</tr>
</tbody>
</table>

22-bit narrow Klass Pointers
To Do Next

- Analyze cache effects of hyper-aligning
  - Split up Klass?
  - Vary cadence by cache line size?
- 32-bit
  - Not technically difficult, just messy and onerous
Lilliput: 16-bit ?
16-bit Classpointers are possible

- First 65k classes: objects use 16-bit nKlass in mark word
- Later-class-objects: append nKlass (or, Klass*) to mark word

⇒ Variable-sized header

<table>
<thead>
<tr>
<th></th>
<th>Markword</th>
<th>Object fields...</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1..0xFFFF</td>
<td>0xFFFE</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Markword</th>
<th>0x10000 .. x</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xFFFF</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Summary
Result

- 10 bits free
- Restored ihash to 31-bit, 4 spare bits
- nKlass Pointer \(\Rightarrow\) nKlass ID
- Costs:
  - Addressable classes \(\sim 5 \rightarrow \sim 3\) mio
  - Slightly more complex decoding
Result (2)

Side benefits for Stock JVM (JDK 22+)

- Improved class space setup, e.g. much higher chance for unscaled or zero-based encoding, with ASLR
- Optimized klass decoding for RiscV and (to a lesser extent) Arm64 and X64
Thank you!

linkedin.com/company/red-hat

youtube.com/user/RedHatVideos

facebook.com/redhatinc