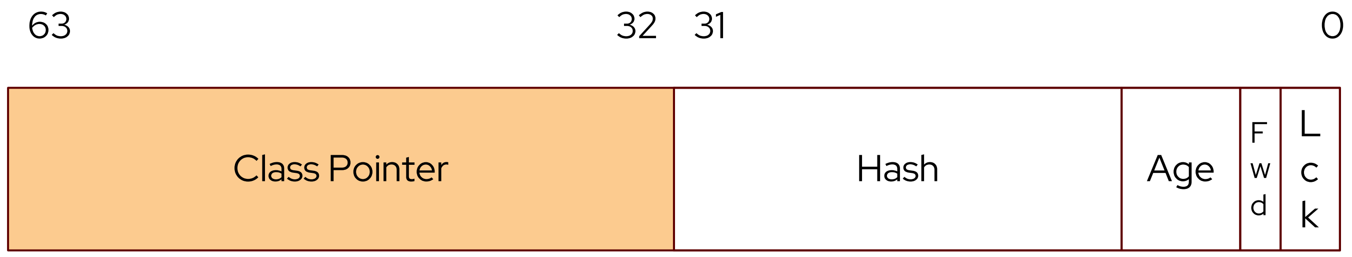
Lilliput: Tiny Classpointers

A 10-minute speed run

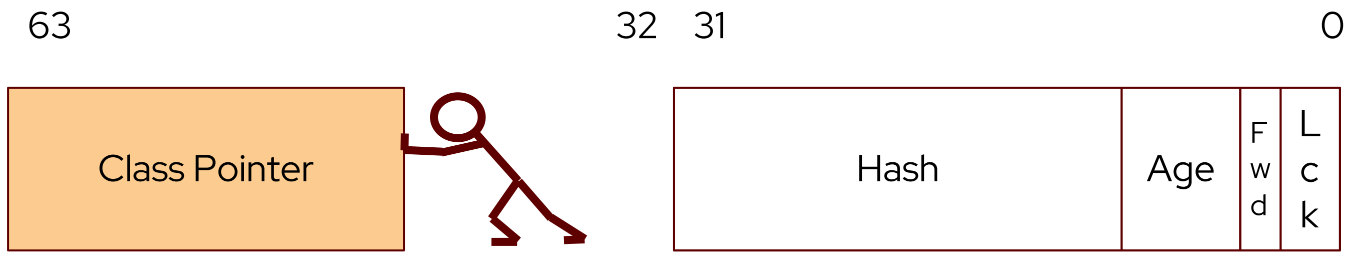
Thomas Stüfe
Principal Engineer
tstuefe@redhat.com

Motivation



Class Pointers take a lot of space…

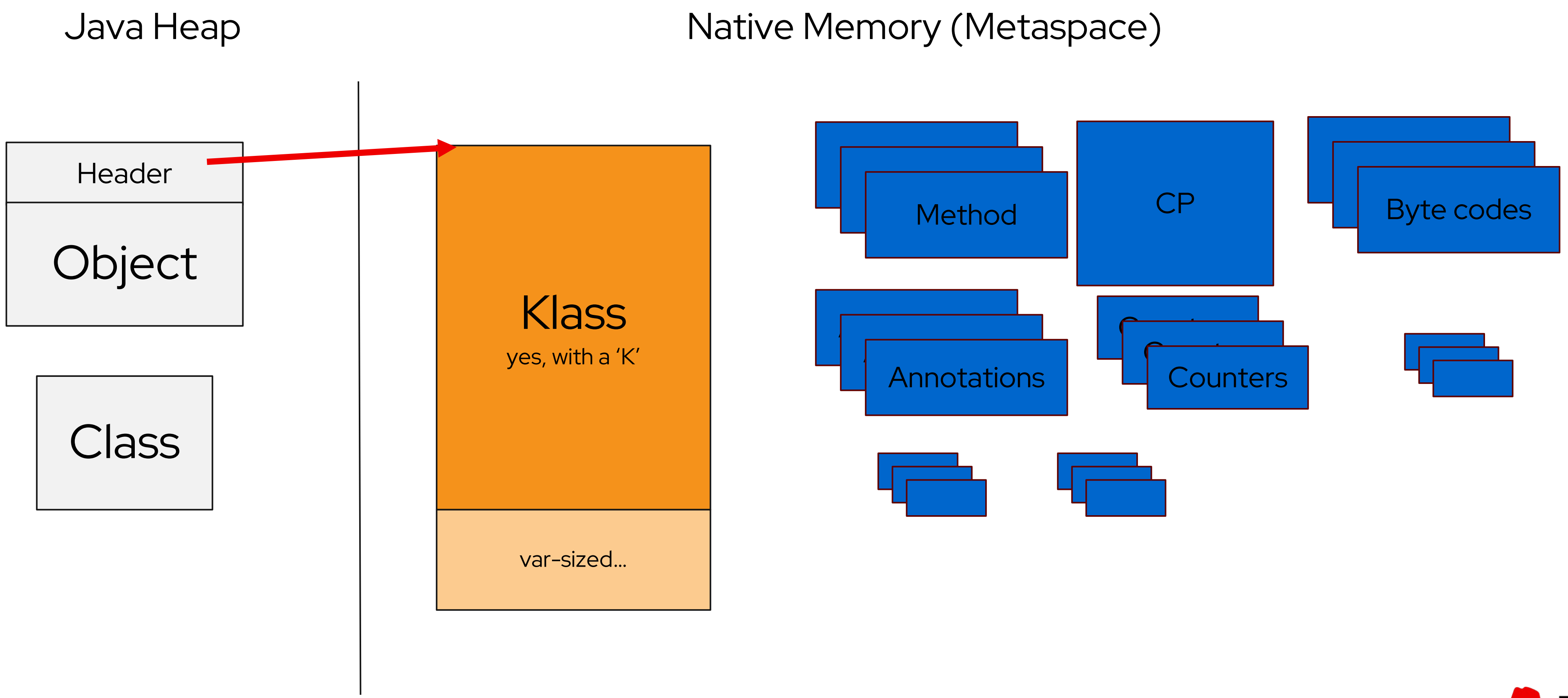
Motivation



...we need to make them smaller

What is a Class Pointer ?

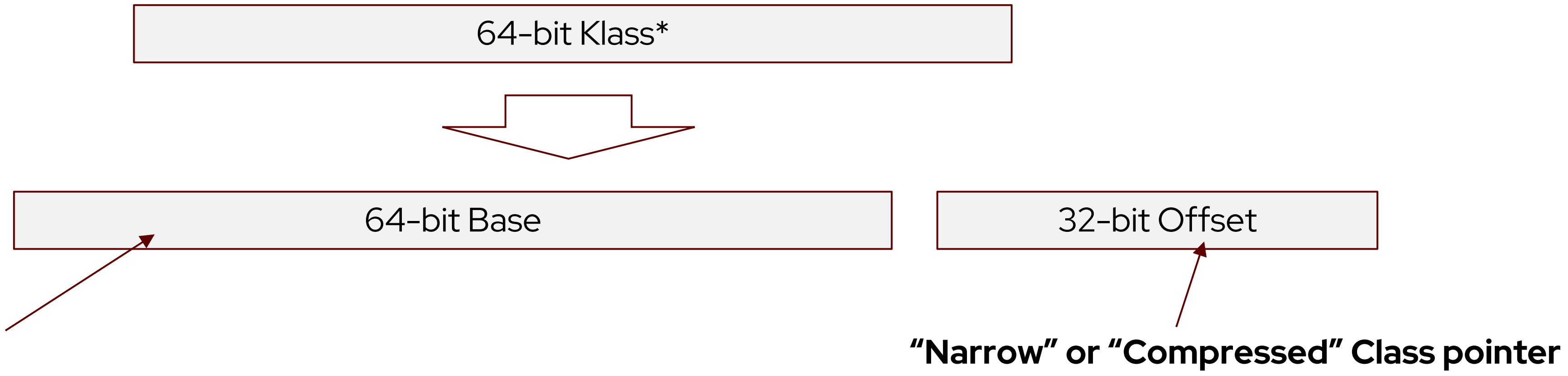
Class and Class Metadata



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.



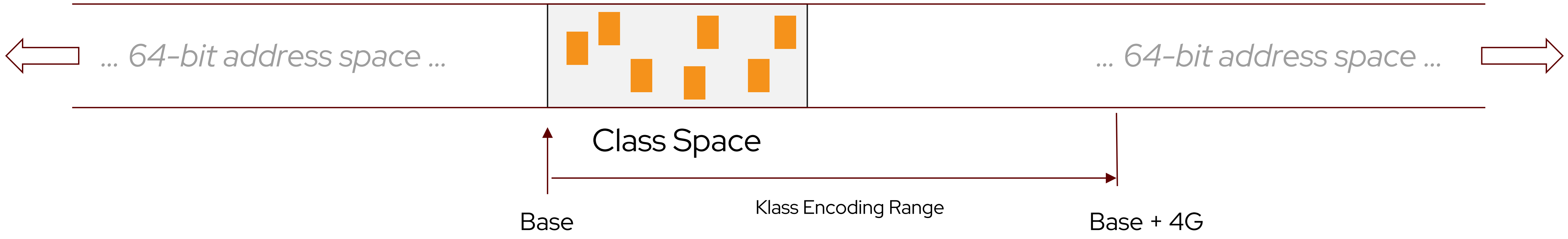
“Encoding Base”
Runtime-constant, determined at VM start

Class Space

32-bit offset?

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

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

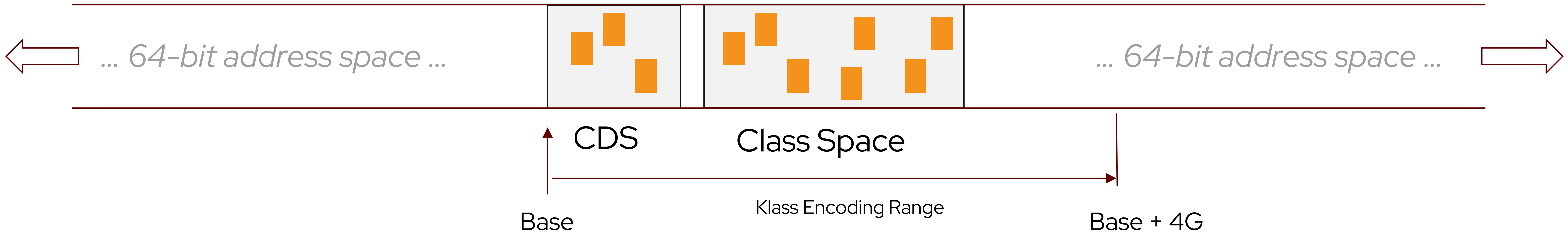


(*) Yes, I am ignoring the encoding shift

… 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

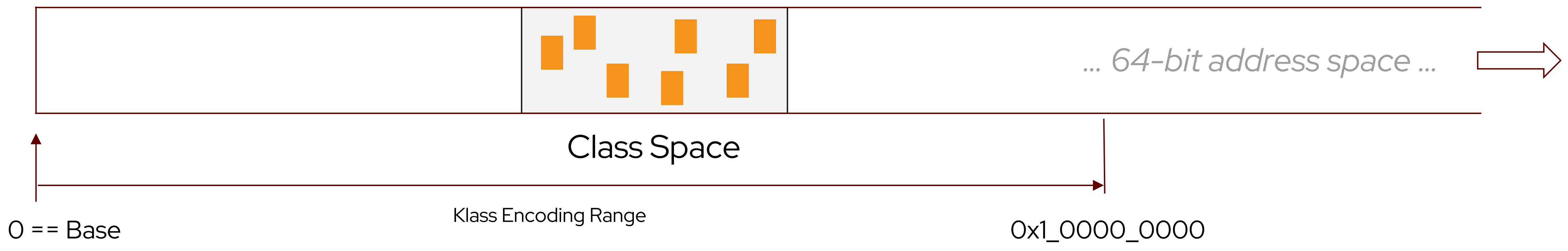
CPU-specific encoding bases

- RiscV: bits set only in [12-32) (for *lui*) or [32-44) (*addiw+slli*)
- Arm64: Either a logical immediate aligned to 4GB (*eor*) or bits in the third quadrant only (*movk)*
- S390: Prefer <4GB addresses (*algfi*) or bits restricted to a single quadrant
- x64: Prefer < 4GB for the short form of 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:

```
31                16   15     10 9         0
xxxxxxxx-xxxxxxxx-xxxxxx00-00000000
```

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

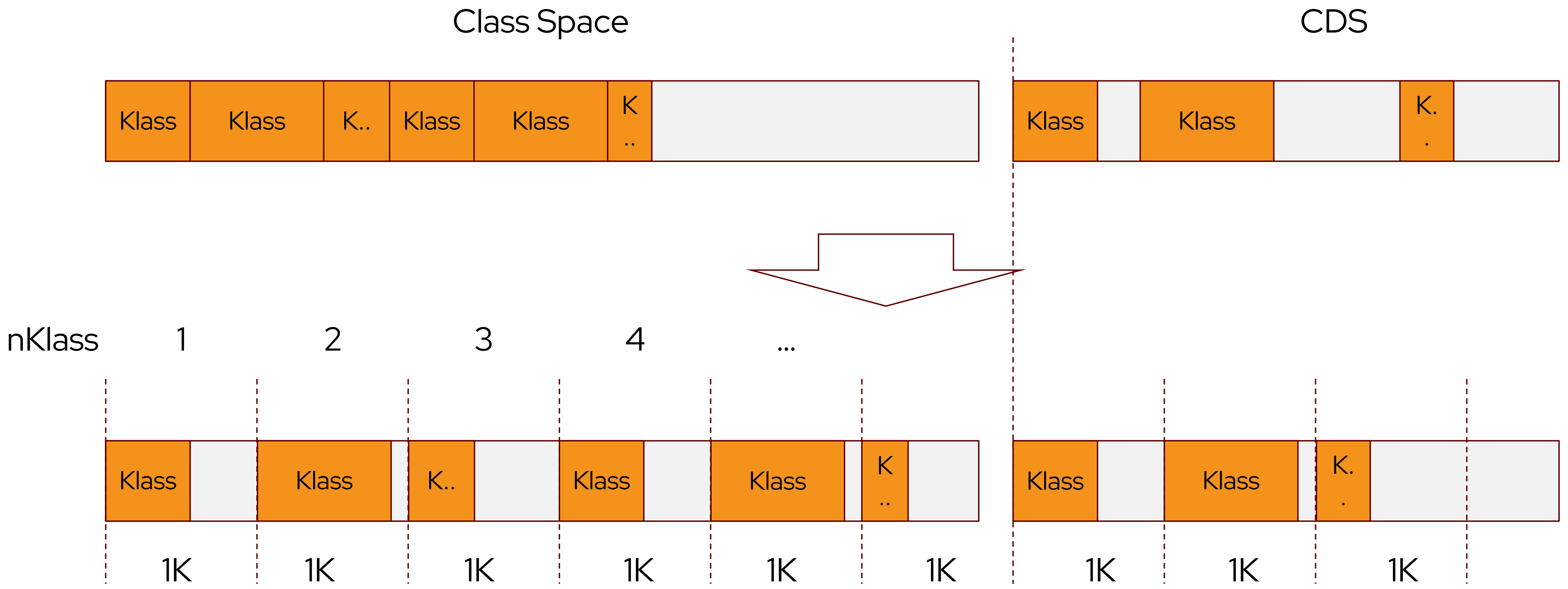
22 bits let us address **3 million** classes (*)

⇒ Klass needs 1 KB on average

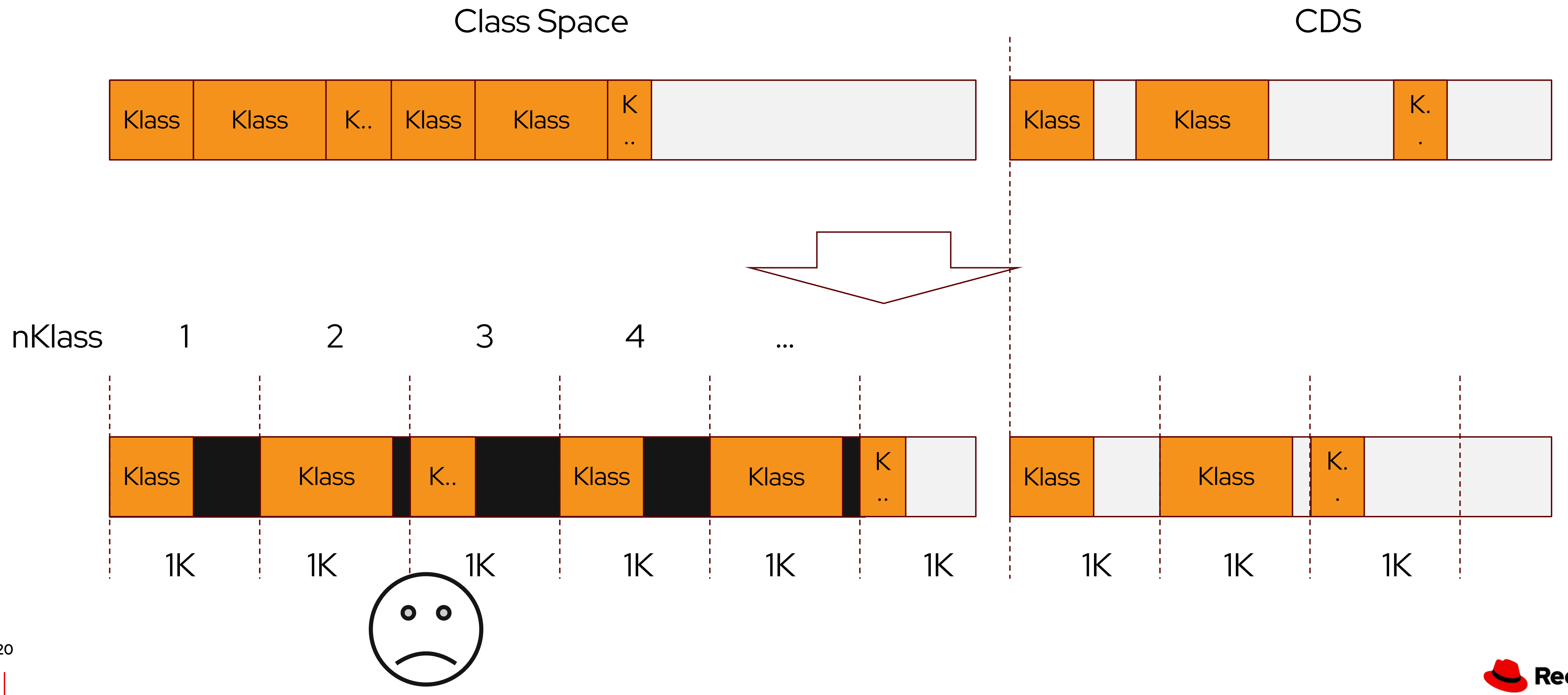
⇒ Class space capped at 3 GB

(* without CDS)

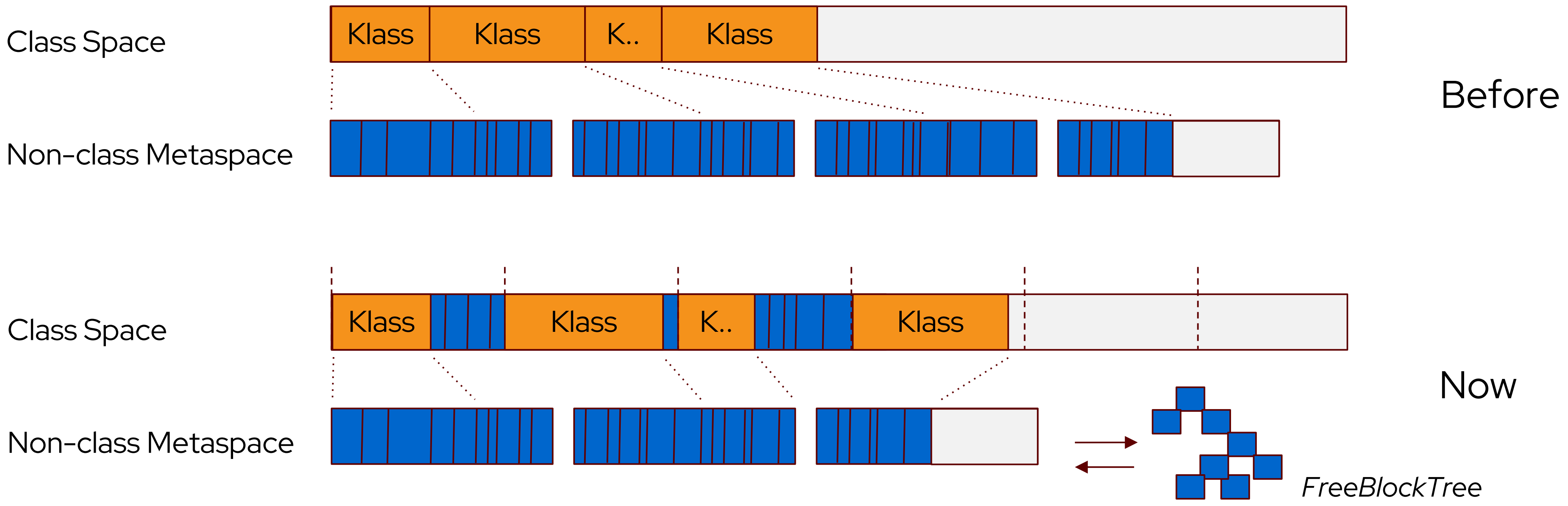
Class Space morphs into a Table



… but fragmentation hurts

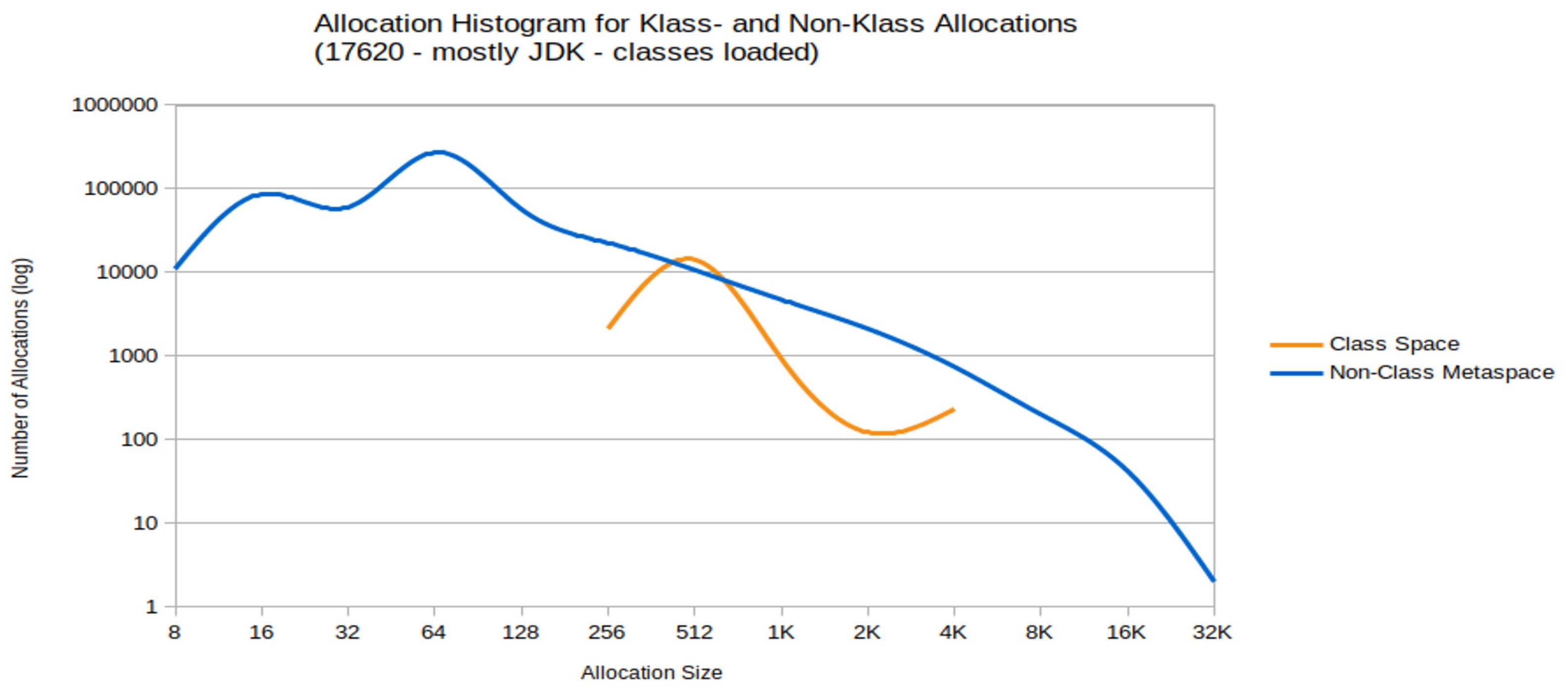


Make Metaspace alignment-aware



It works beautifully: (almost) zero footprint degradation.

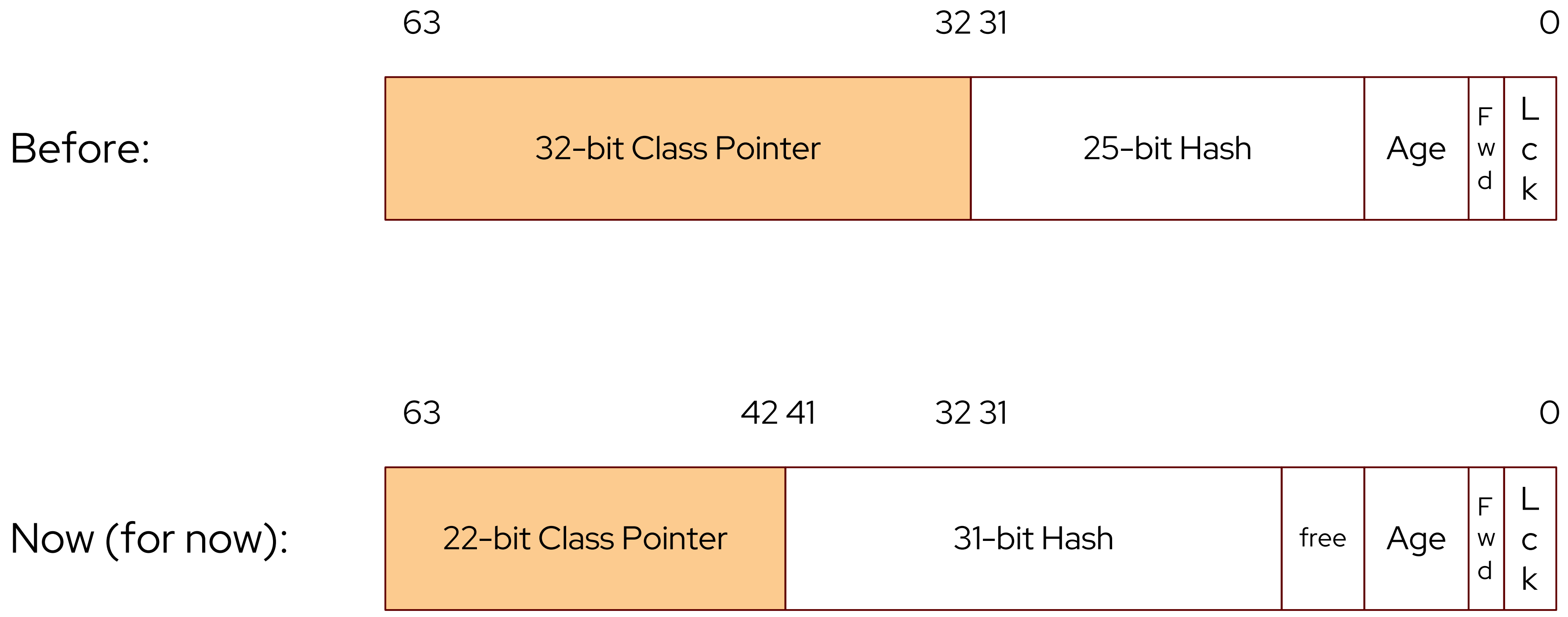
Statistics



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

Non-Klass: **Numerous, fine-grained**

New Markword Layout (for now...)



To Do Next

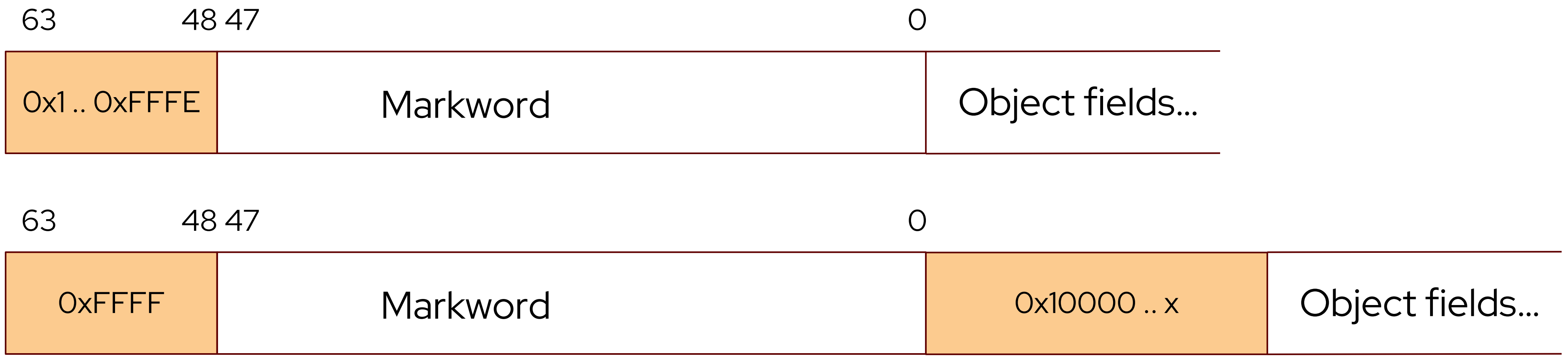
- Analyze cache effects of hyper-aligning
  - Split up Klass?
  - Vary cadence by cache line size?
- 32-bit
  - Not technically difficult, just messy and onerous

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



Summary

Result

- 10 bits free
- Restored ihash to 31-bit, 4 spare bits
- nKlass Pointer ⇒ nKlass ID
- Costs:
  - Addressable classes ~5 → ~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