

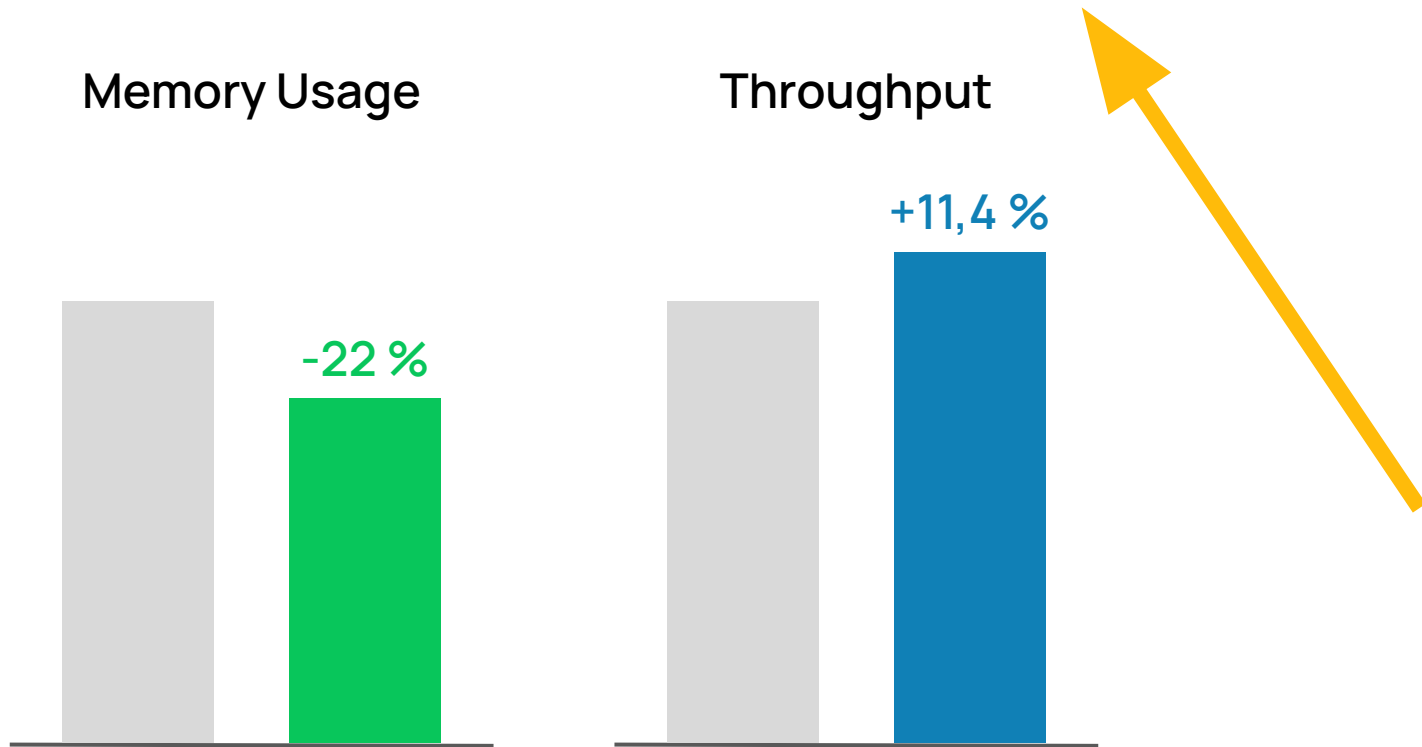


Von Compressed OOPs bis Compact Headers: JVM-Interna verständlich erklärt

Sven Woltmann

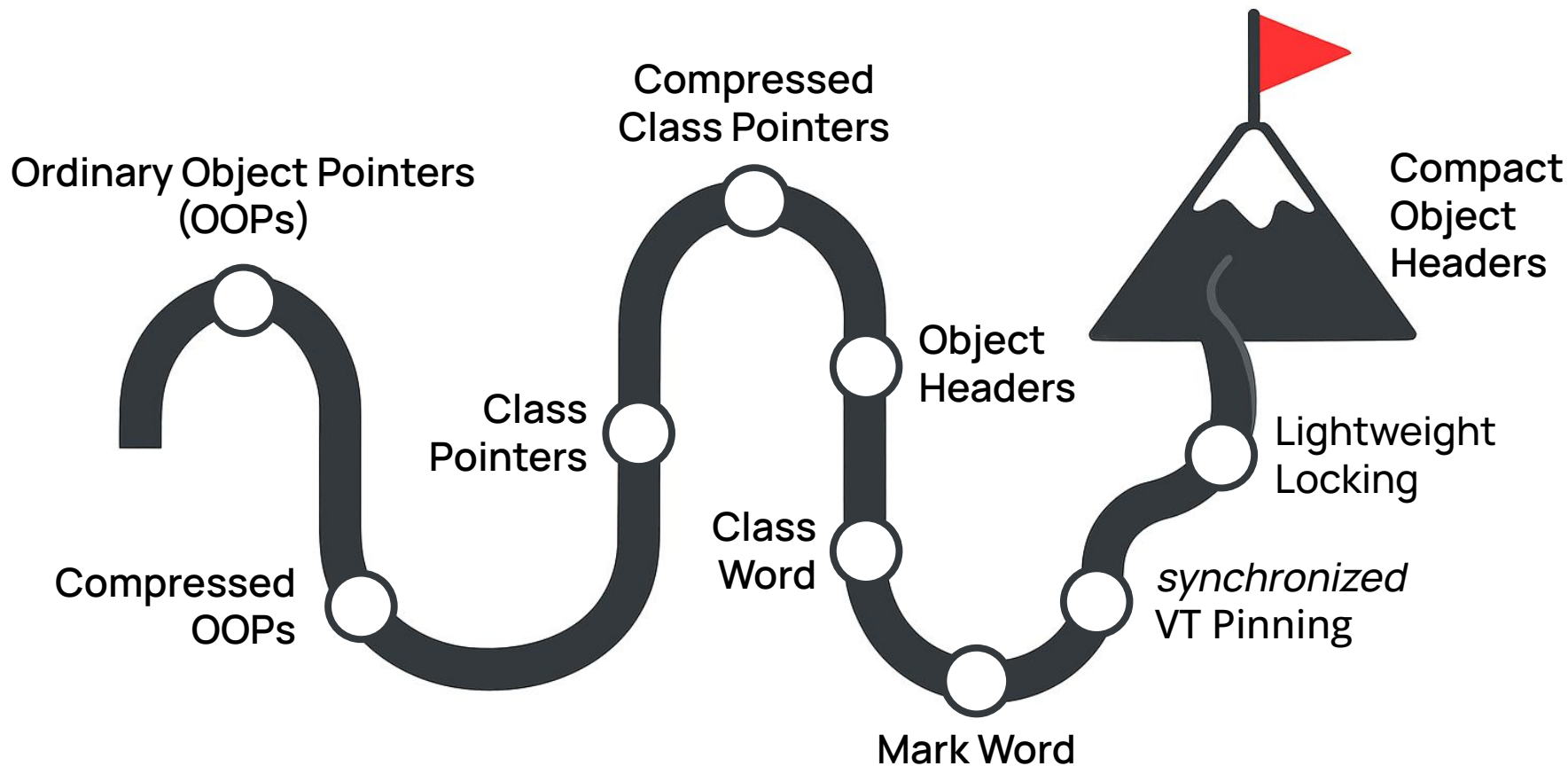
Java Author, Speaker, Trainer

Java 25 with -XX:+UseCompactObjectHeaders

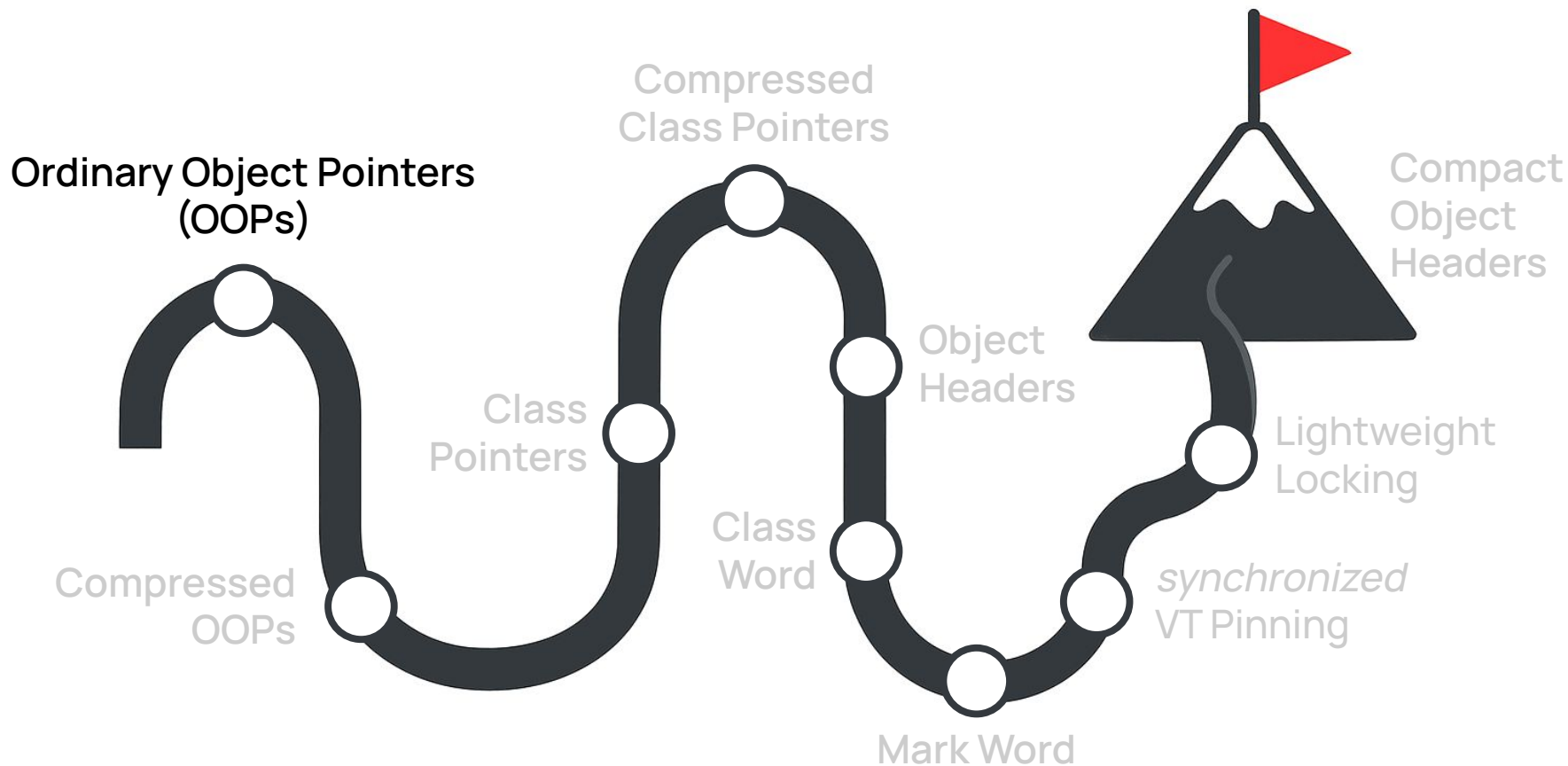


Source: <https://github.com/rkennke/talks/blob/master/Lilliput-FOSDEM-2025.pdf>

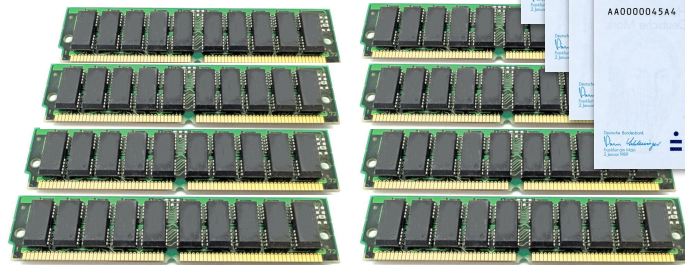
Journey into the JVM



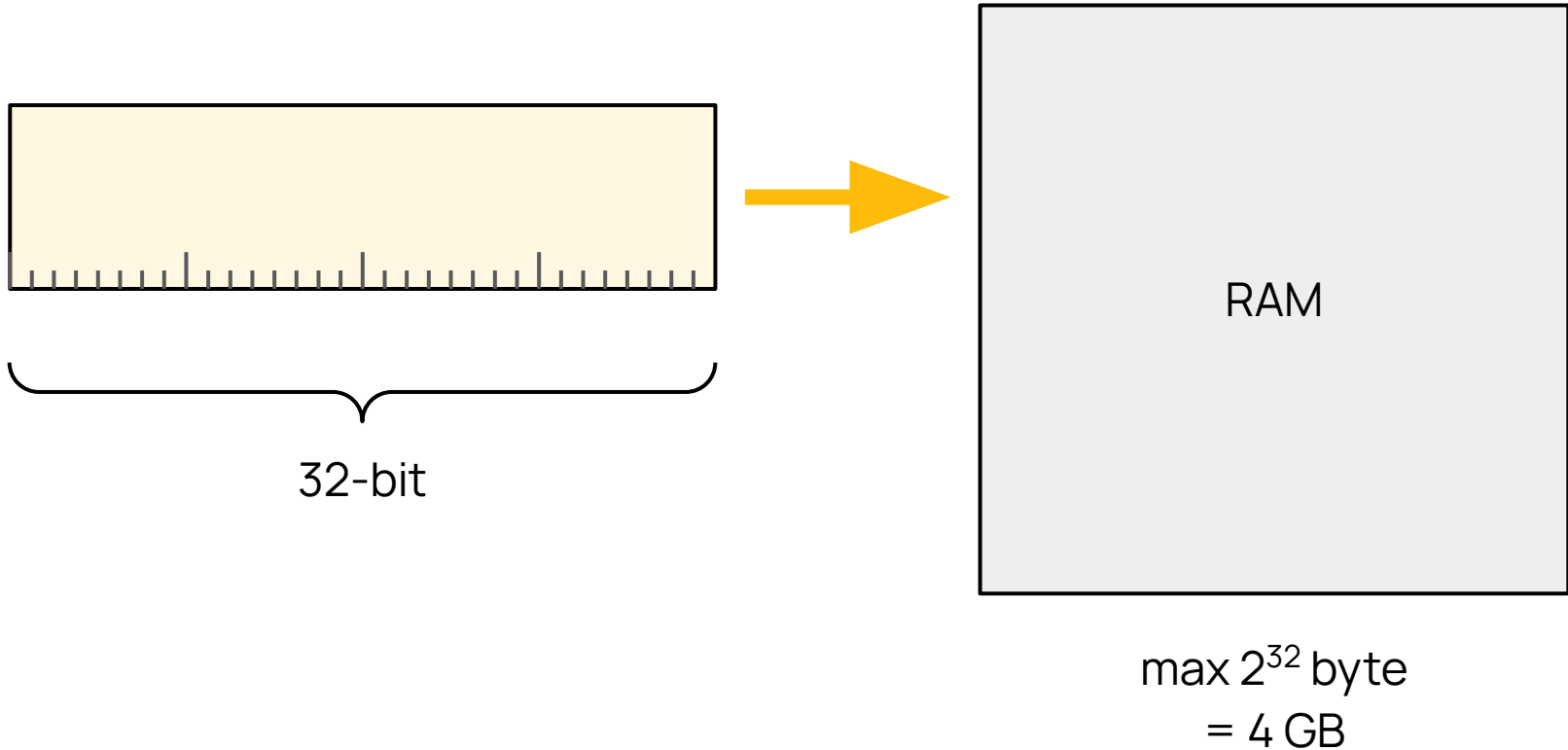
Journey into the JVM



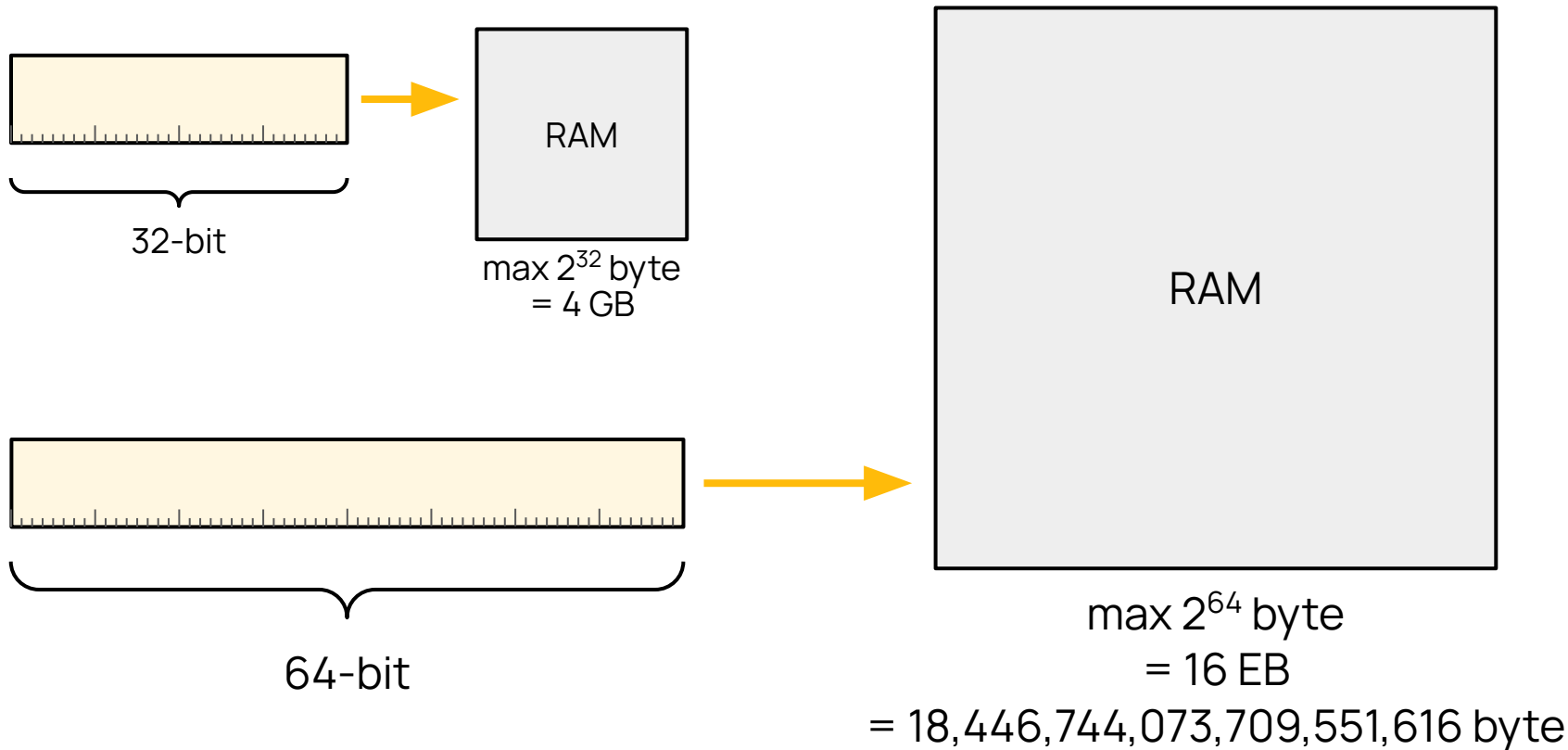
Ordinary Object Pointers



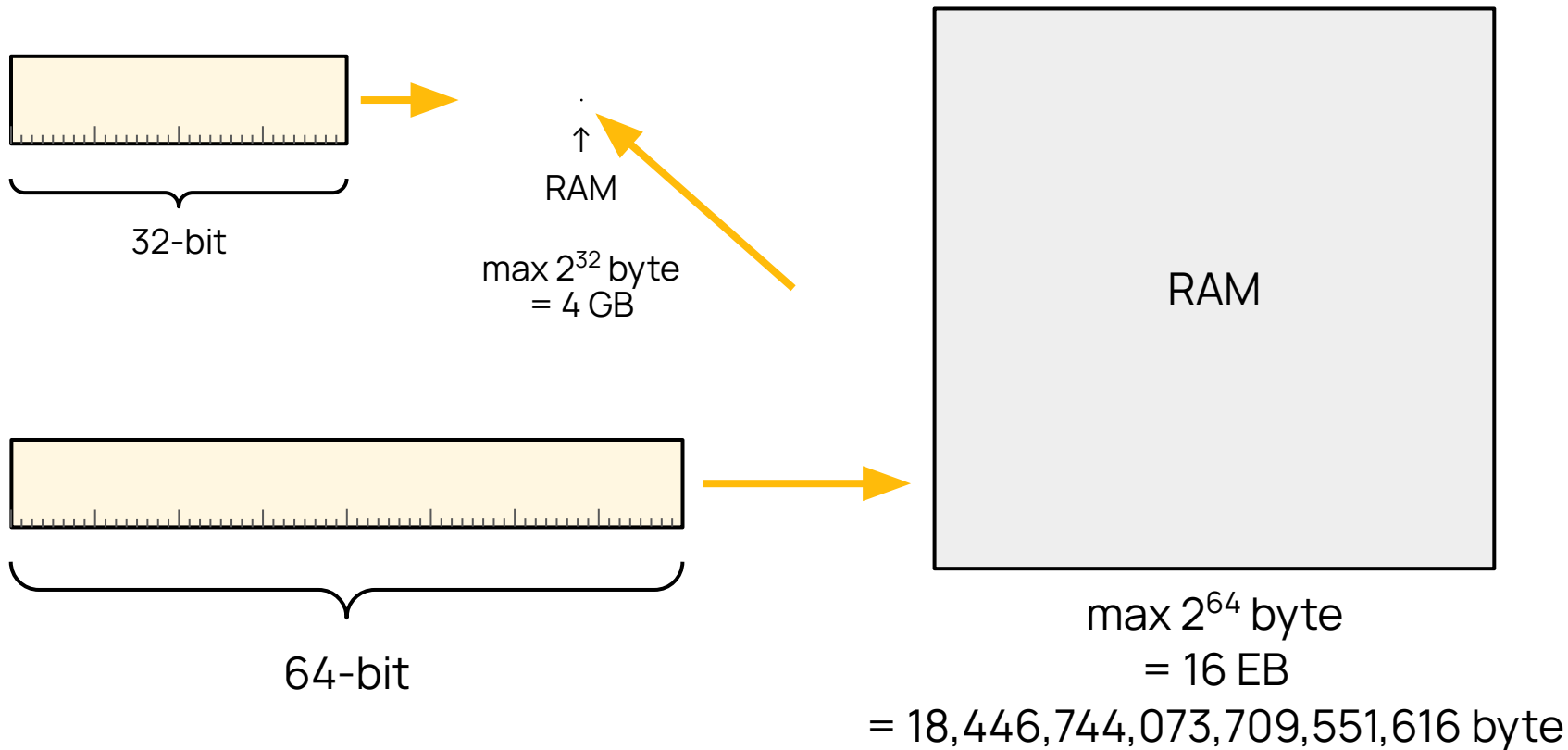
Ordinary Object Pointers



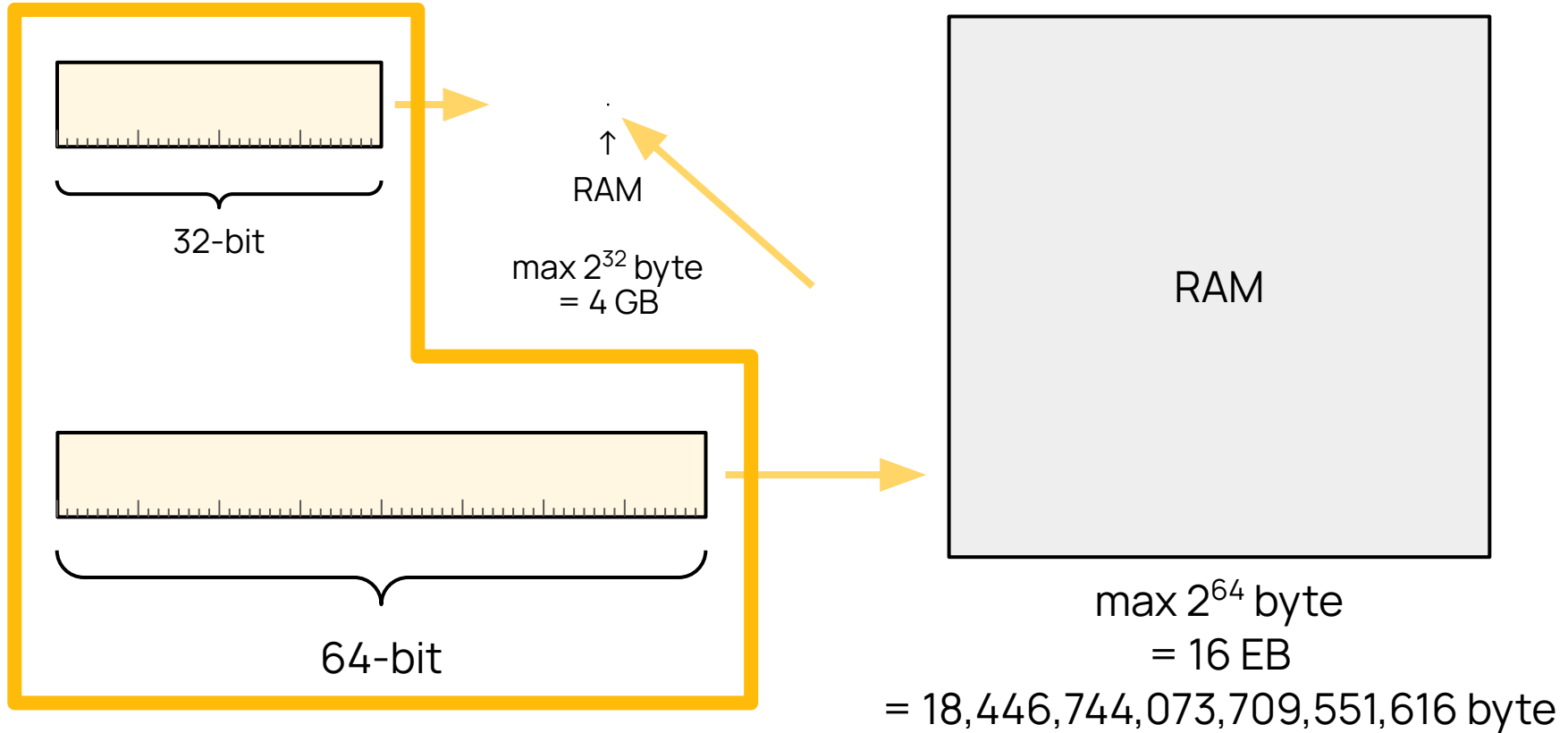
Ordinary Object Pointers



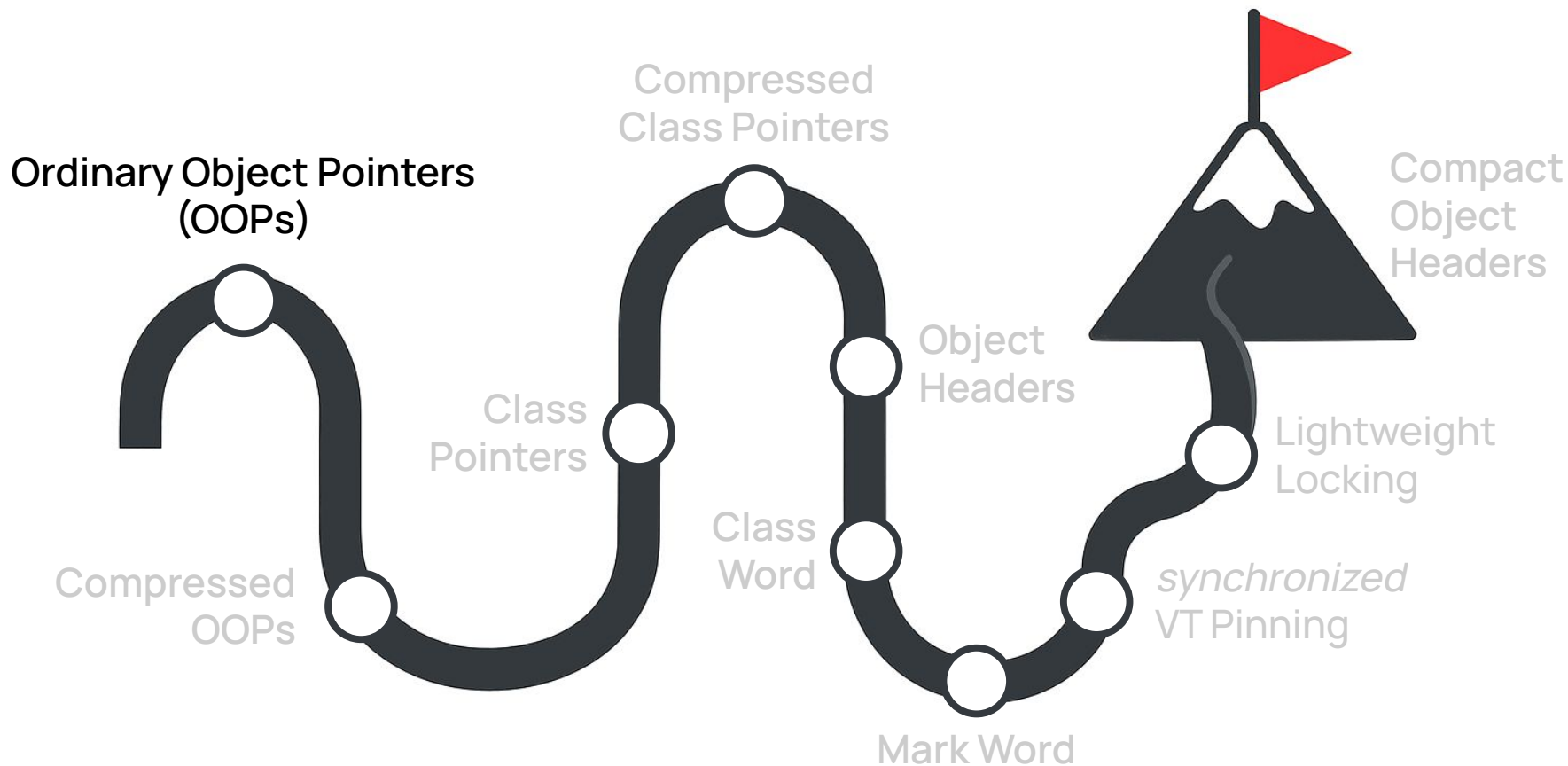
Ordinary Object Pointers



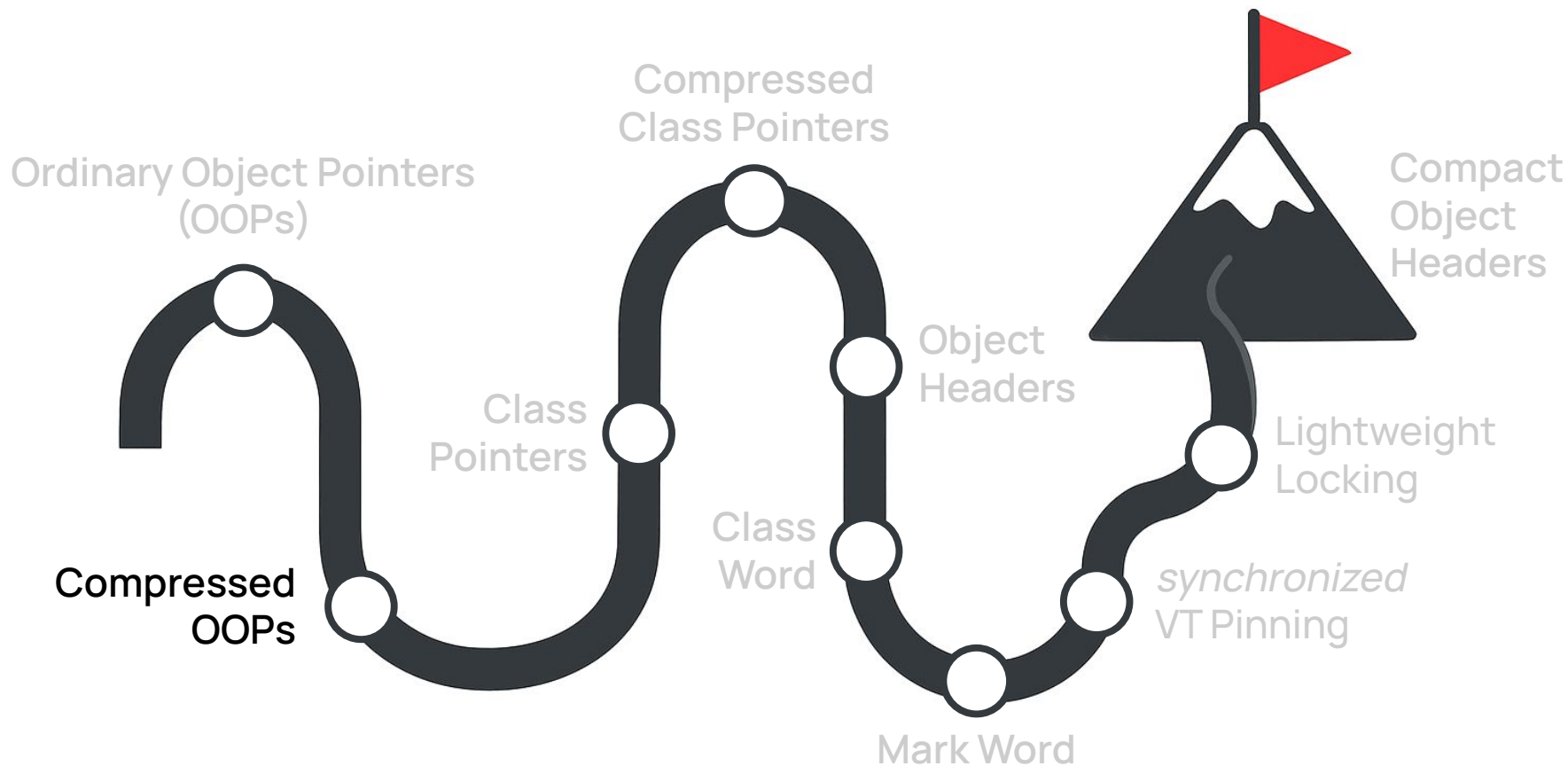
Ordinary Object Pointers



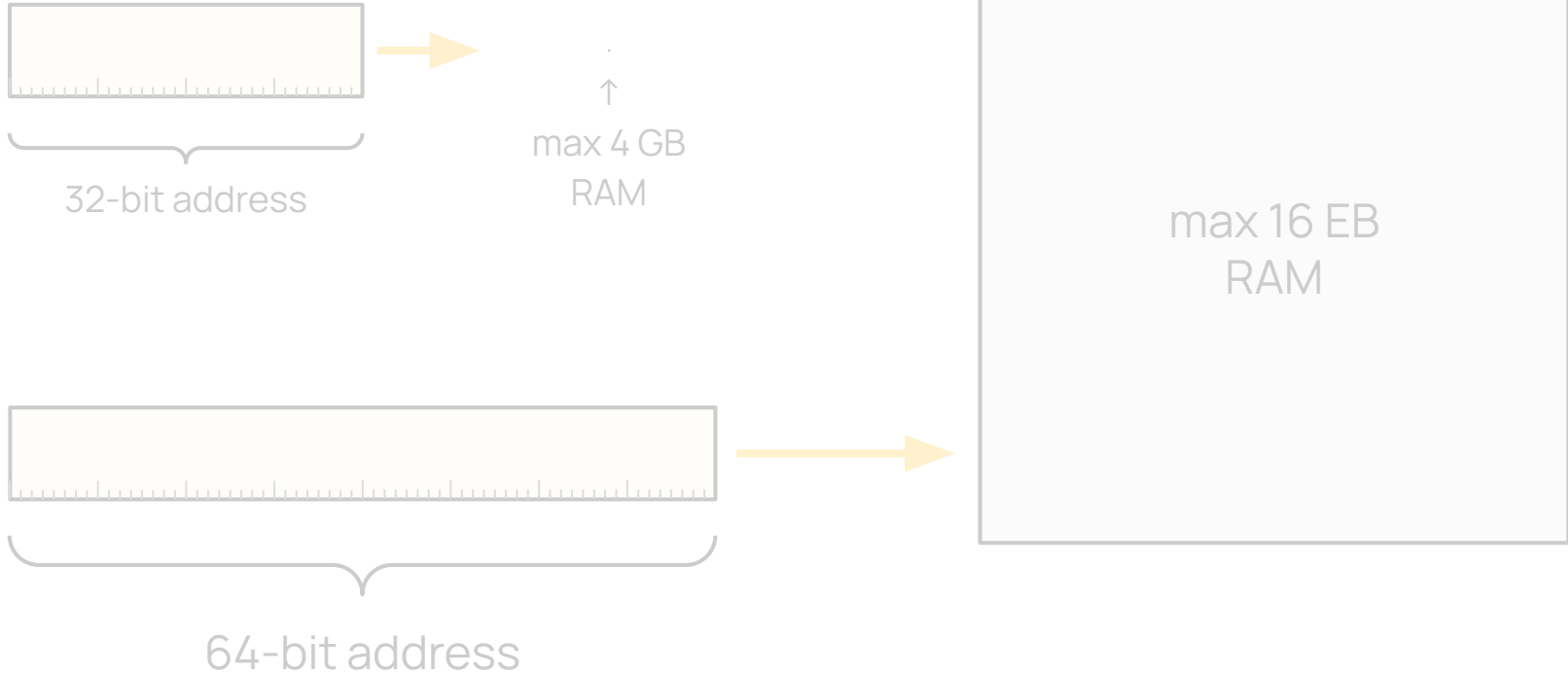
Journey into the JVM



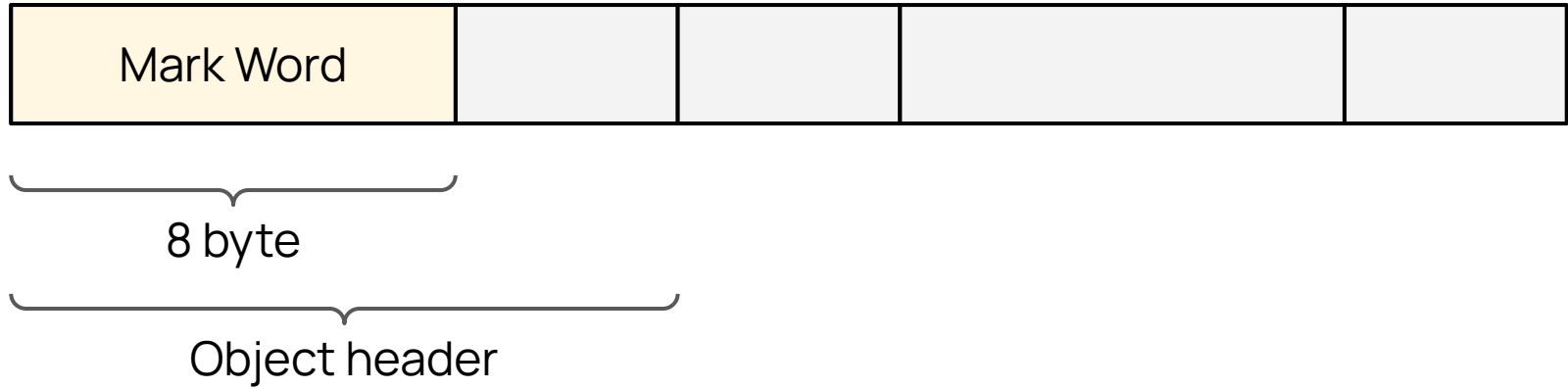
Journey into the JVM



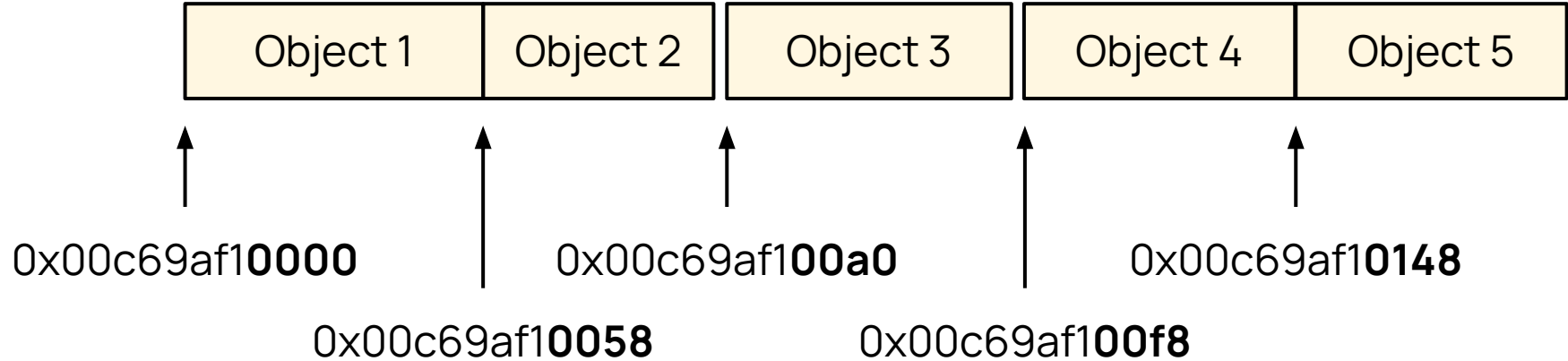
Compressed OOPs



Compressed OOPs



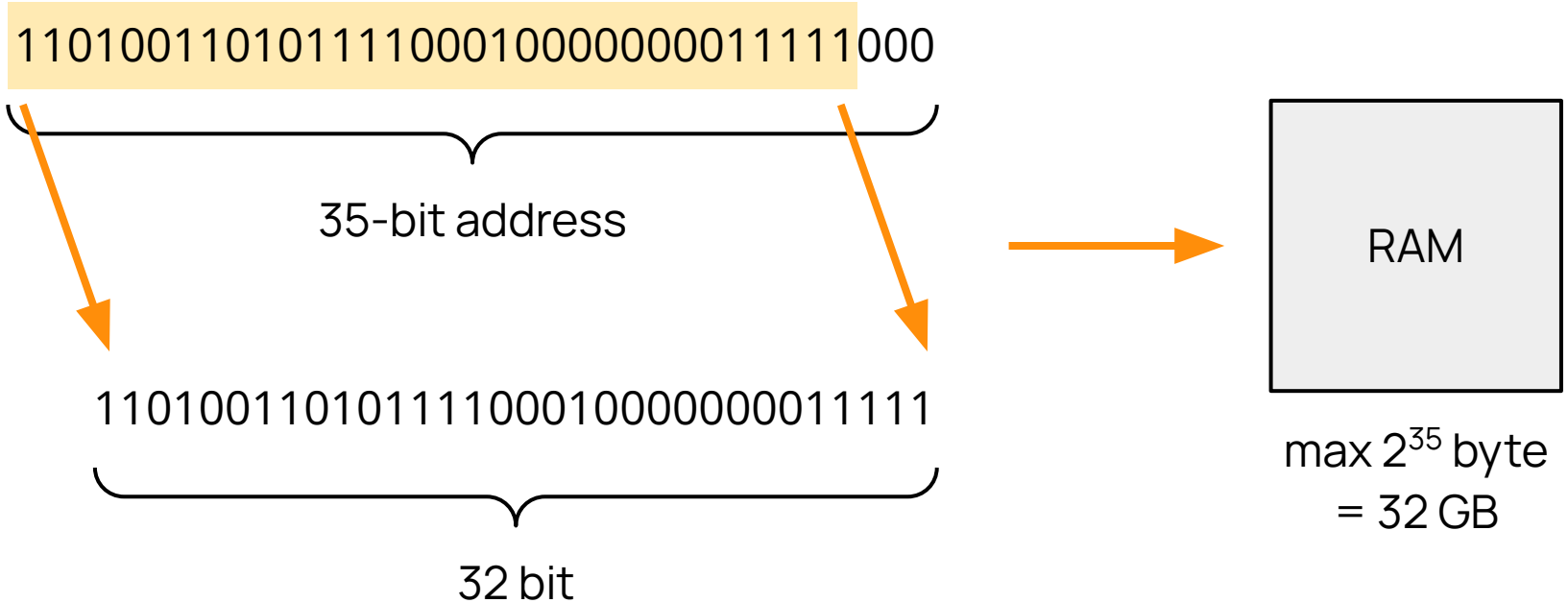
Compressed OOPs



Compressed OOPs

0x00c69af1 0000	0b00000000110001101001101011110001 00000000000000000000
0x00c69af1 0058	0b00000000110001101001101011110001 00000000010110000000
0x00c69af1 00a0	0b00000000110001101001101011110001 00000000101000000000
0x00c69af1 00f8	0b00000000110001101001101011110001 00000000111100000000
0x00c69af1 0148	0b00000000110001101001101011110001 00000001010000000000

Compressed OOPs



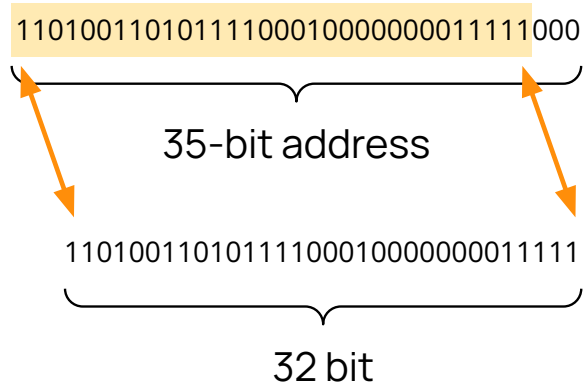
Compressed OOPs

Disable Compressed OOPs and use 64-bit pointers:

-XX:-UseCompressedOops

Compressed OOPs

Max 32 GB Heap



> 32 GB Heap
or

`-XX:-UseCompressedOops`

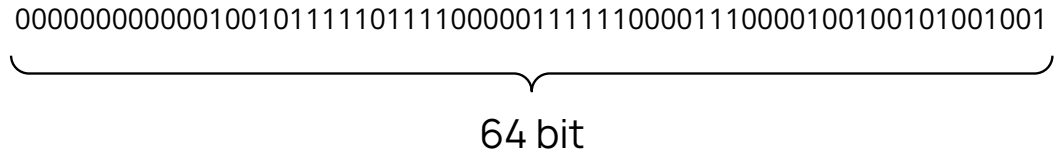


Compressed OOPs

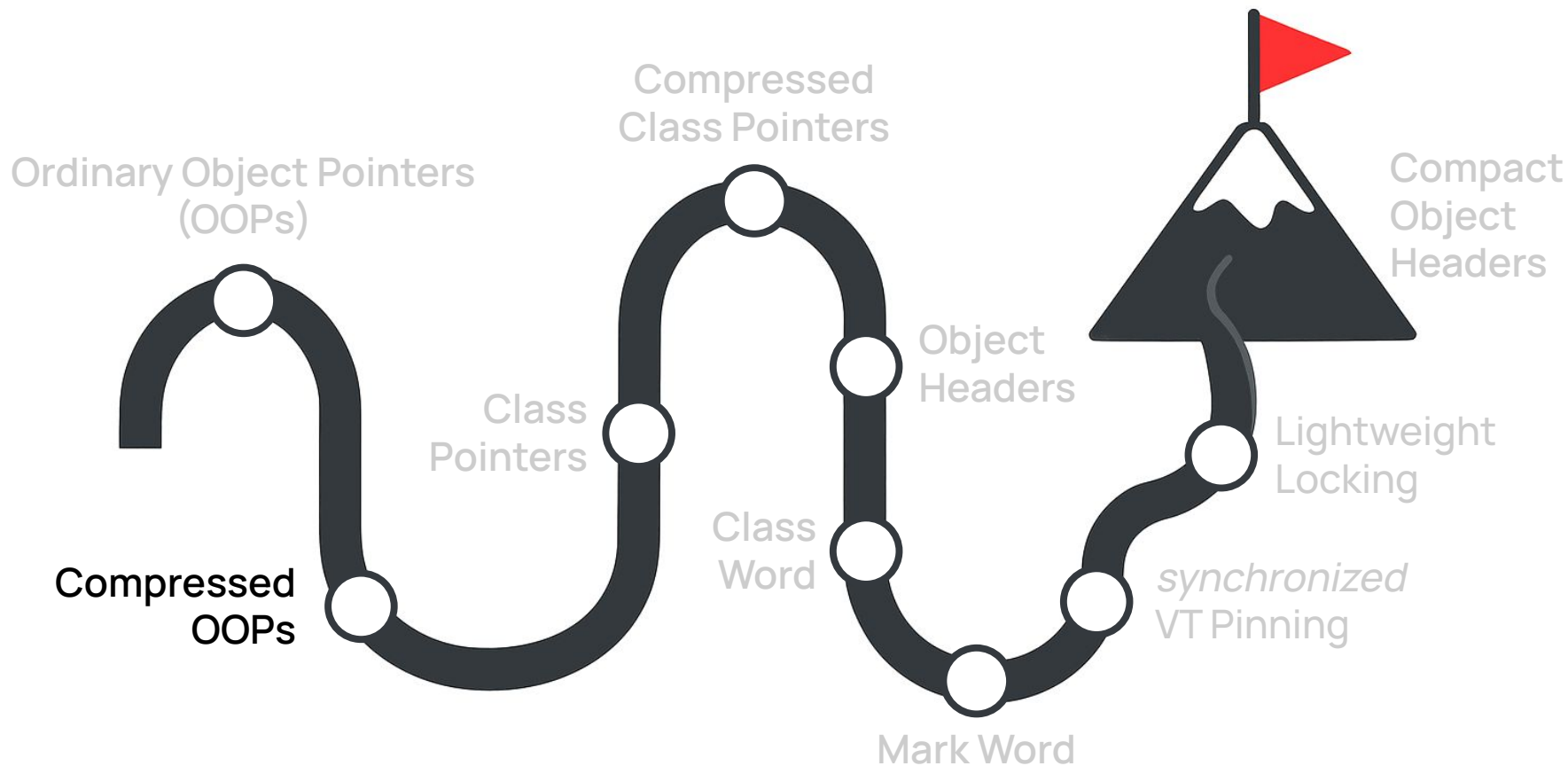
Max 32 GB Heap

> 32 GB Heap
or

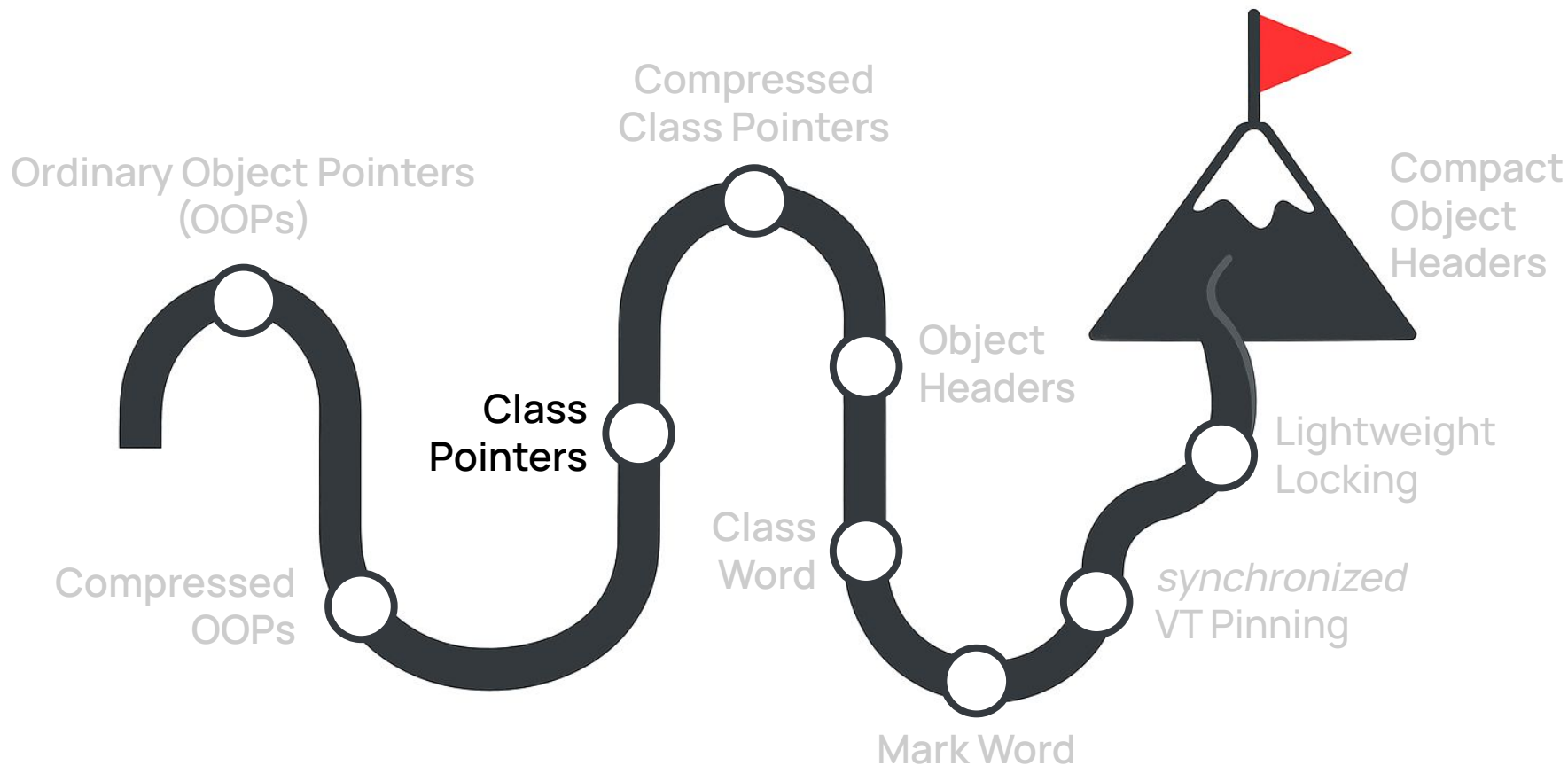
-XX:-UseCompressedOops



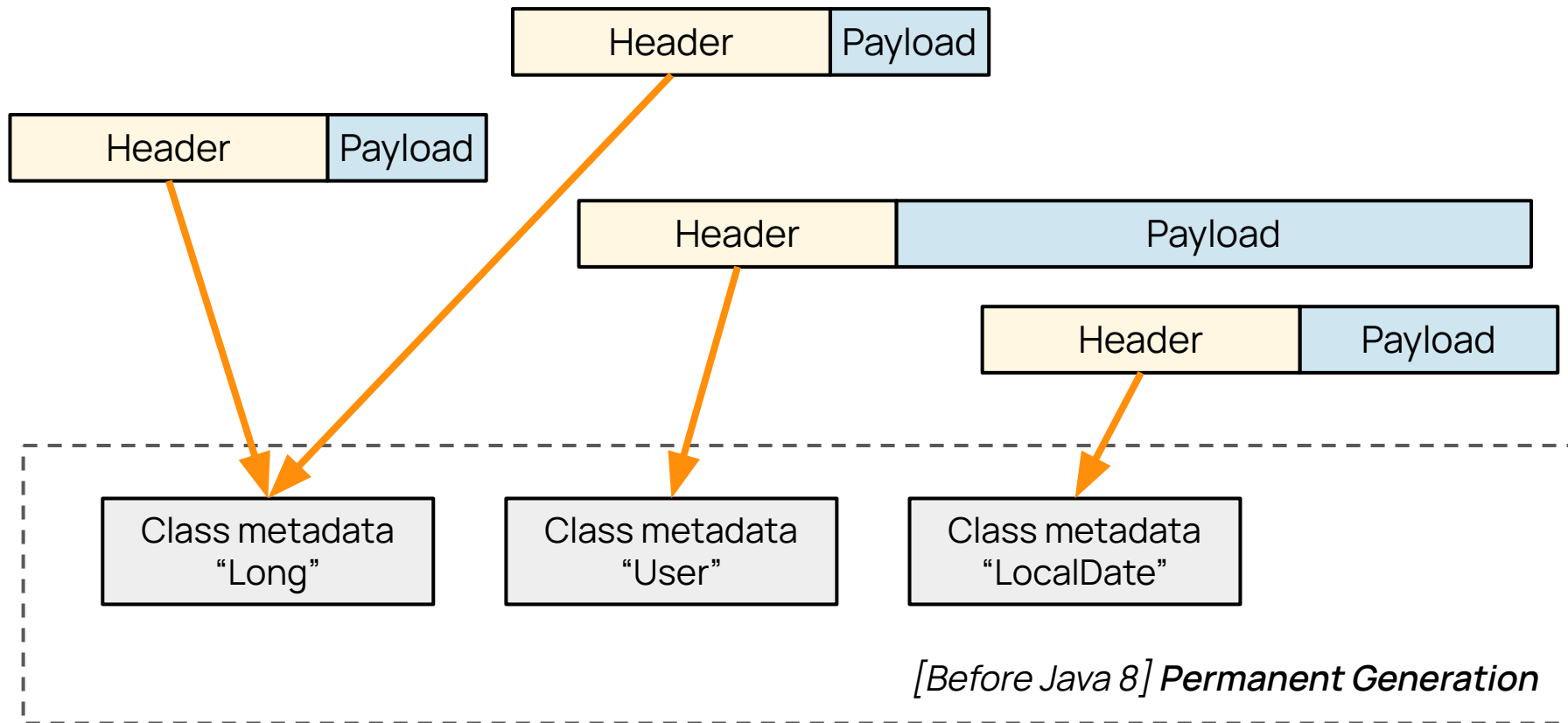
Journey into the JVM



Journey into the JVM



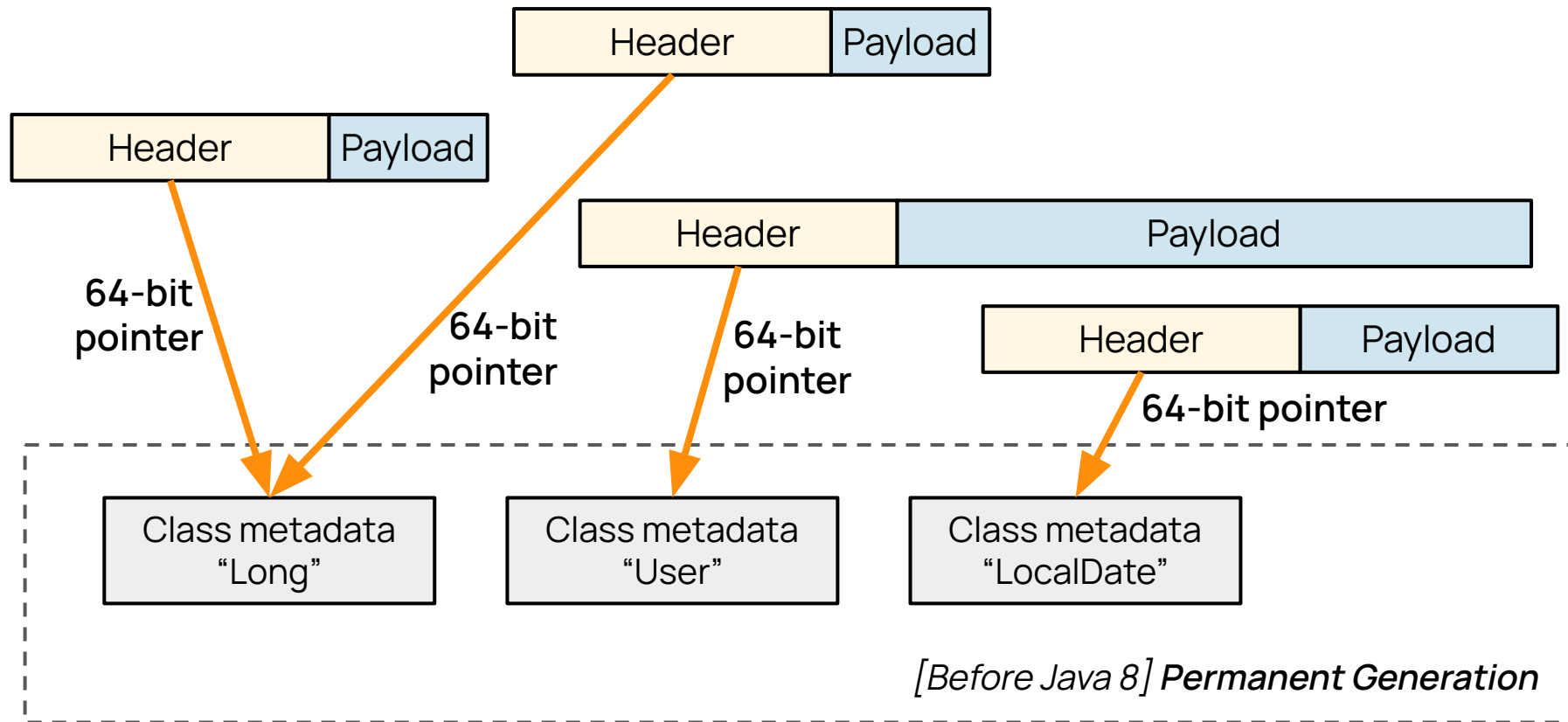
Class Pointers



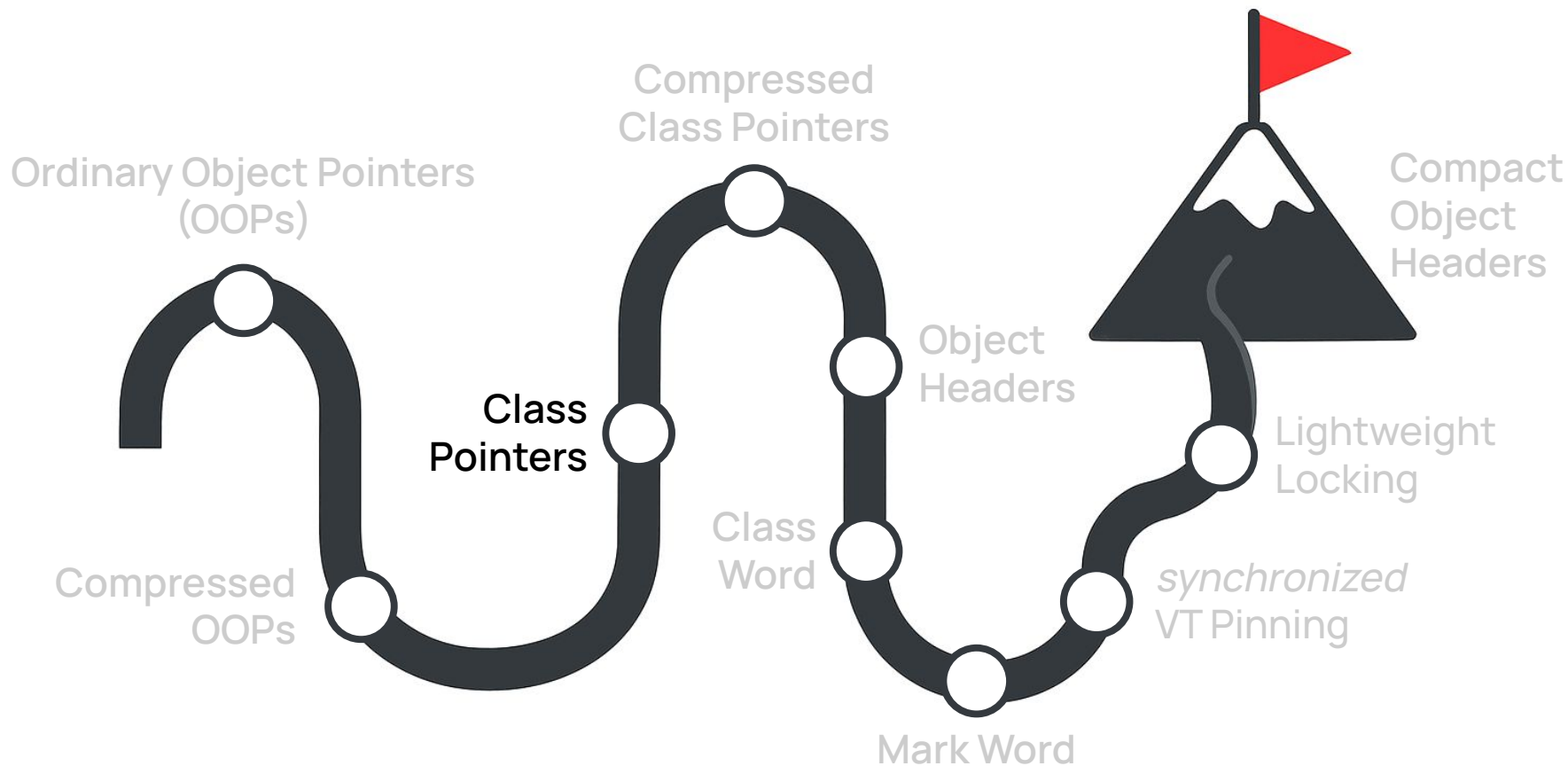
Class Pointers

OutOfMemoryError:
PermGen space

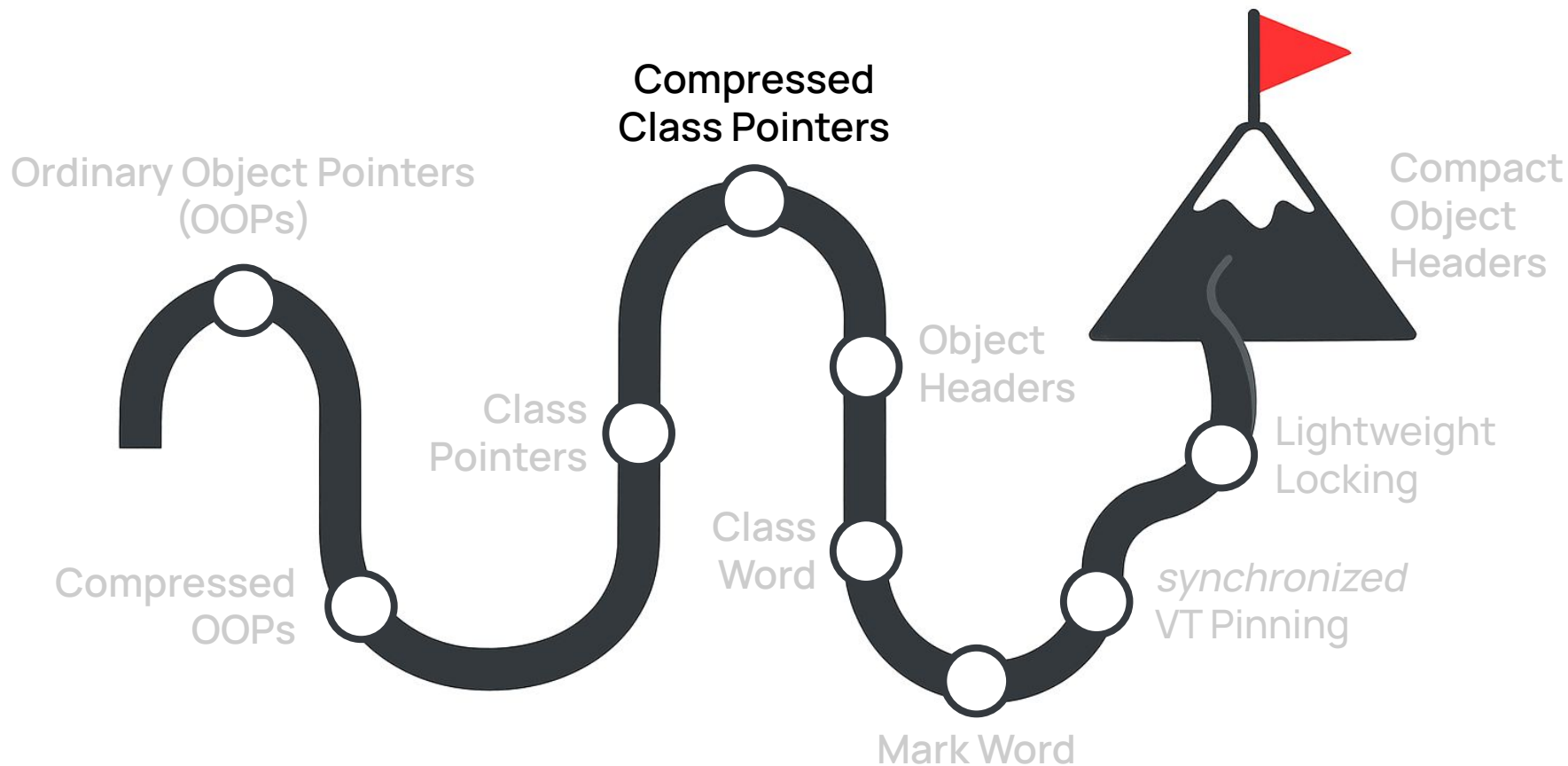
Class Pointers



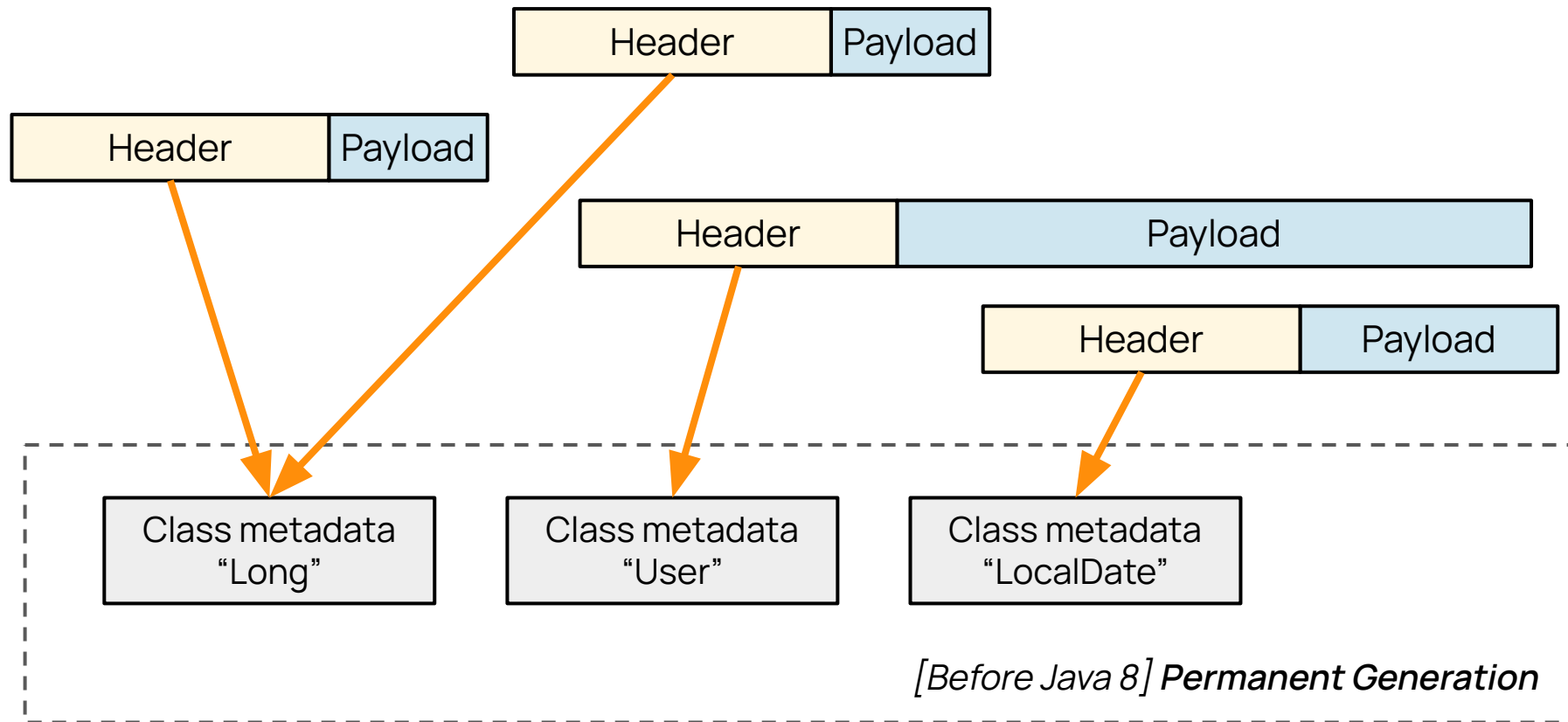
Journey into the JVM



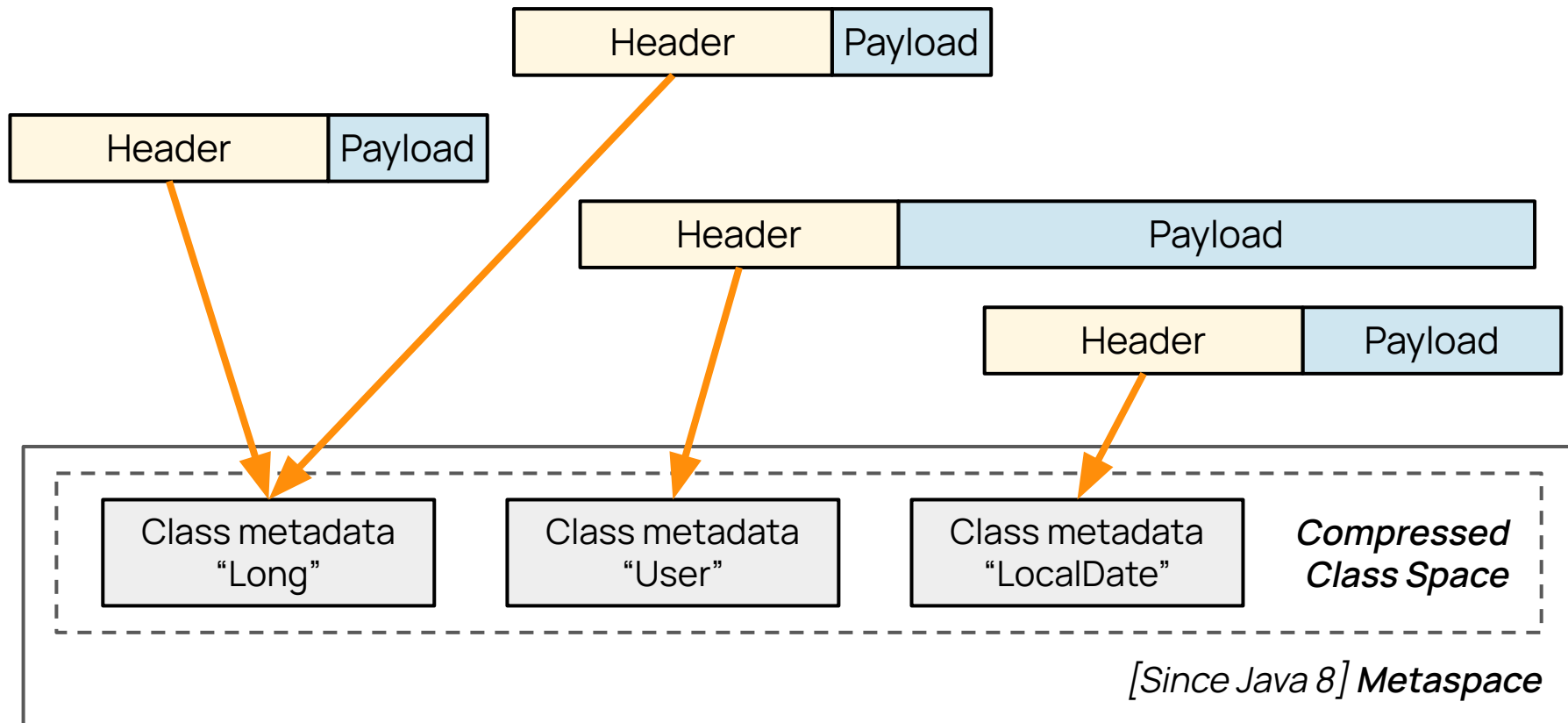
Journey into the JVM



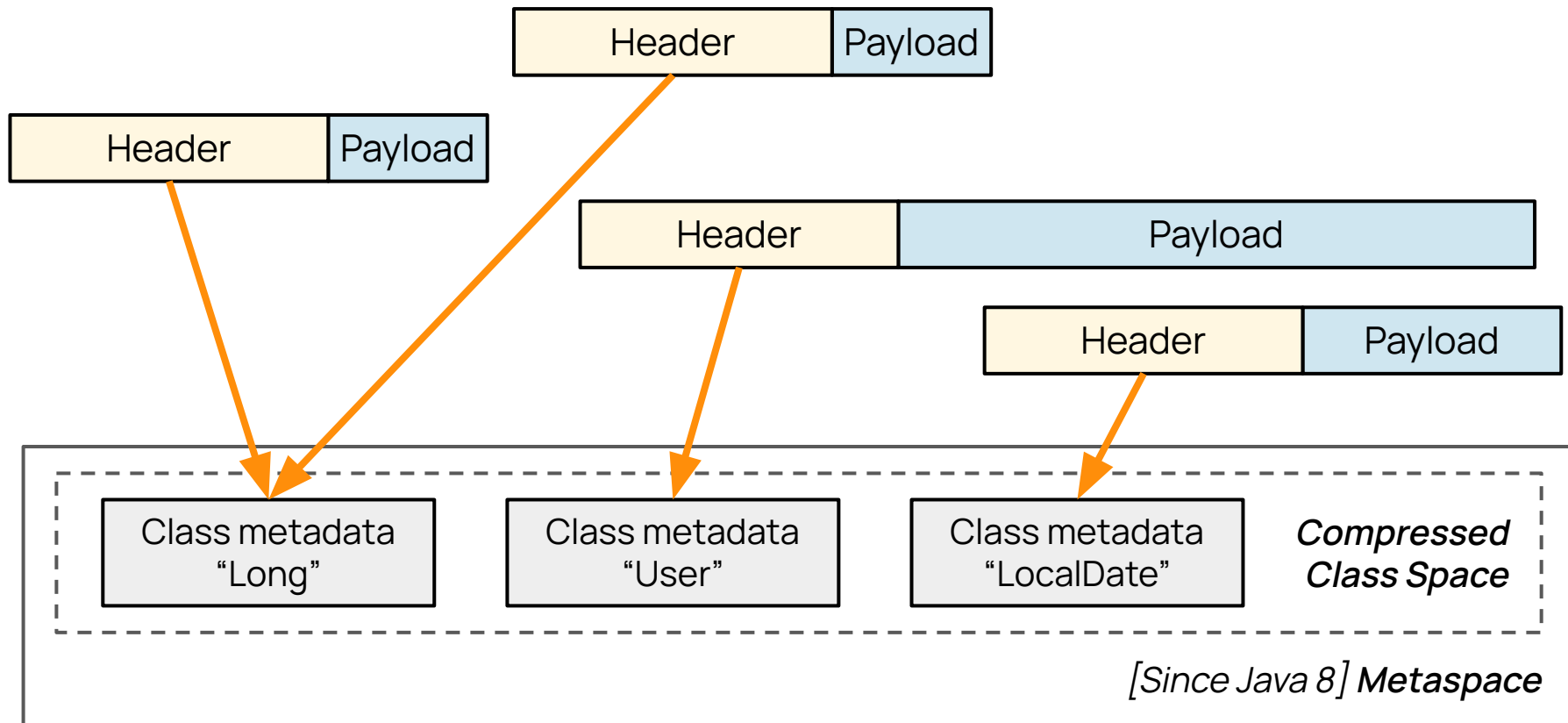
Compressed Class Pointers



Compressed Class Pointers



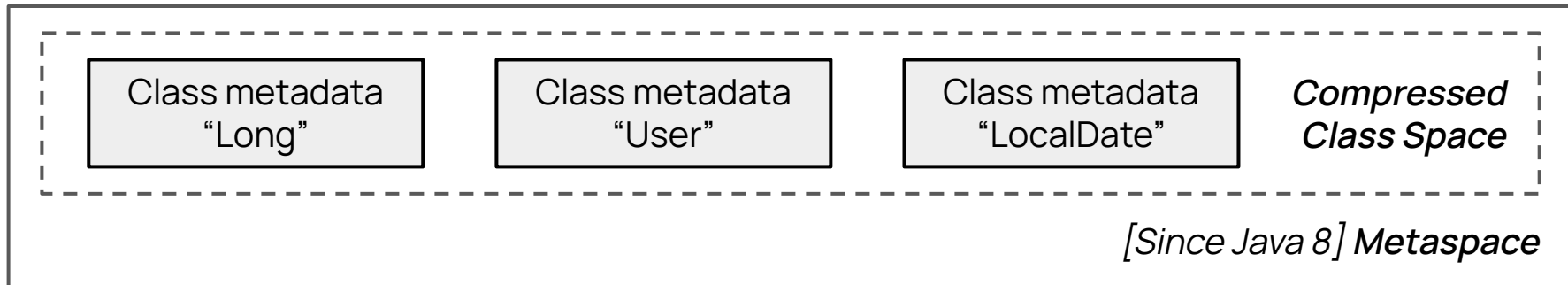
Compressed Class Pointers



Compressed Class Pointers

Class metadata size: 512~1024 bytes

Addressable with 64 bit: 2^{64} bytes / 768 bytes
 $\approx 24,000,000,000,000,000$

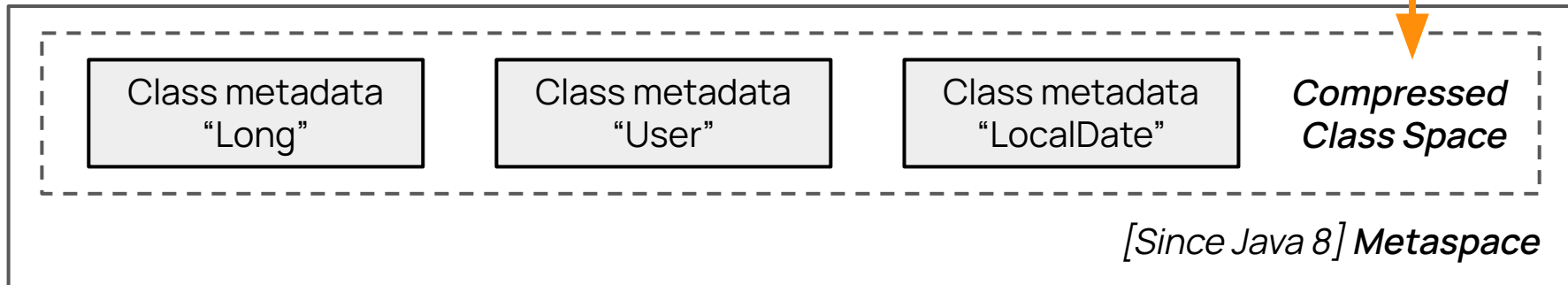


Compressed Class Pointers

Class metadata size: 512~1024 bytes

Addressable with 64 bit: 2^{64} bytes / 768 bytes
 $\approx 24,000,000,000,000,000$

`-XX:CompressedClassSpaceSize` → 1 GB by default
4 GB maximum

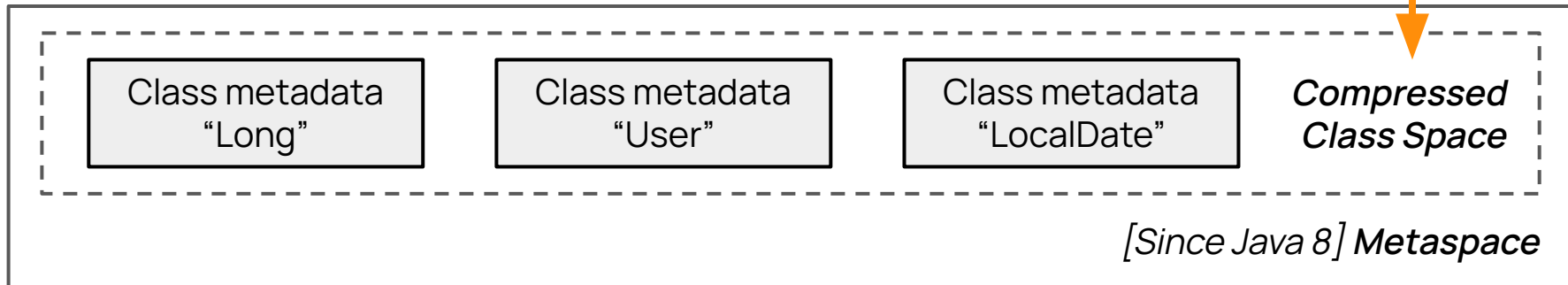


Compressed Class Pointers

Class metadata size: 512~1024 bytes

Addressable with 64 bit: 2^{64} bytes / 768 bytes
 $\approx 24,000,000,000,000,000$

`-XX:CompressedClassSpaceSize`  1 GB by default
4 GB maximum

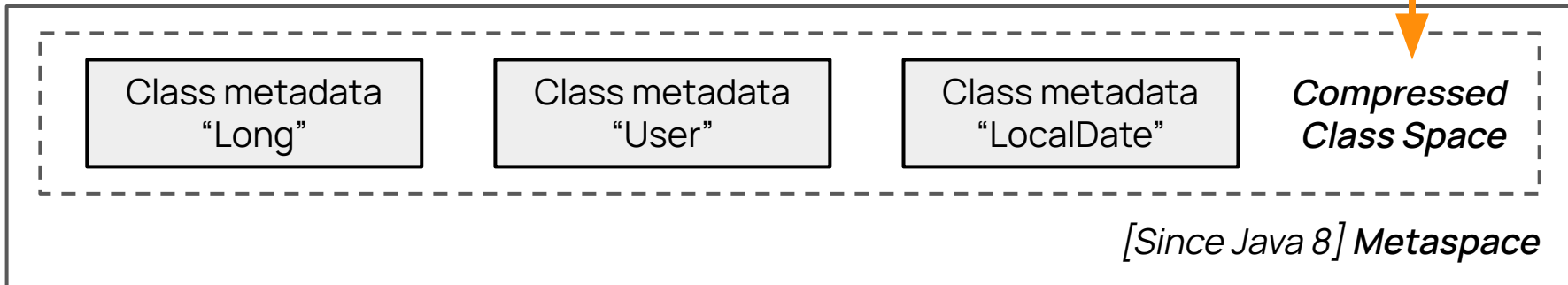


Compressed Class Pointers

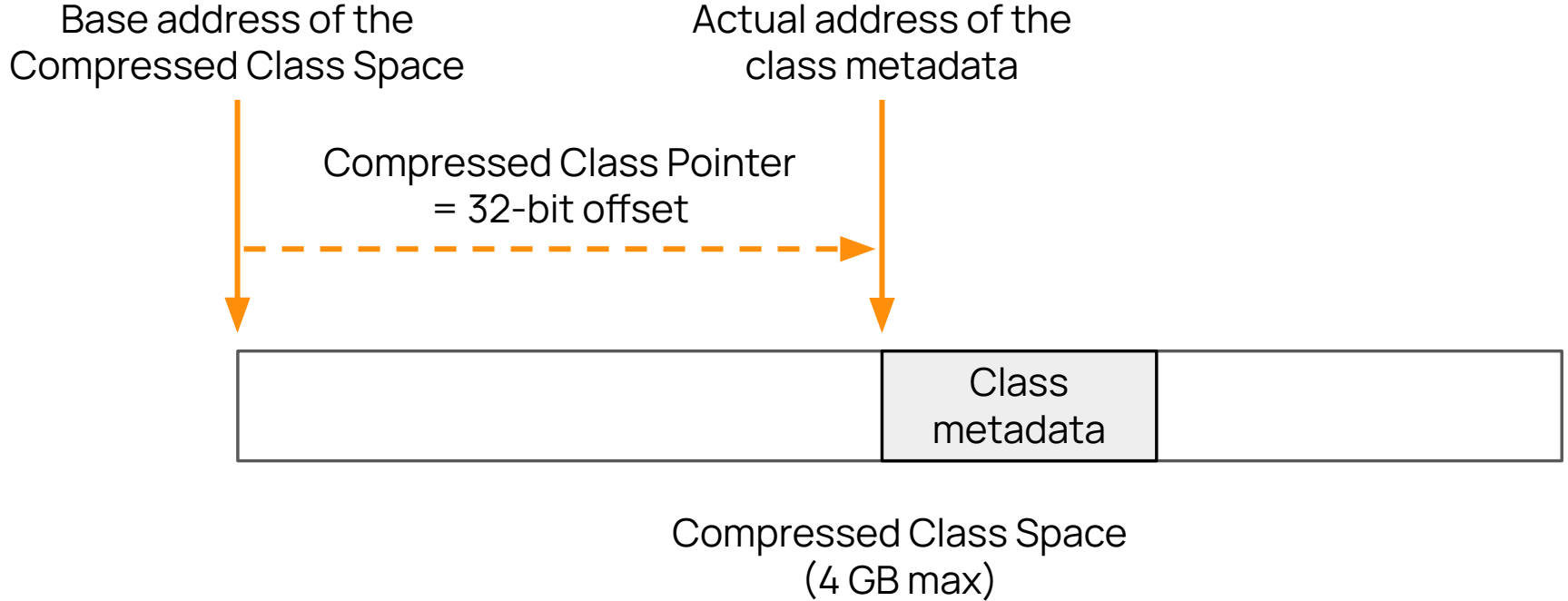
Class metadata size: 512~1024 bytes

Addressable with 32 bit: 2^{32} bytes = 4 GB

`-XX:CompressedClassSpaceSize` → 1 GB by default
→ 4 GB maximum



Compressed Class Pointers



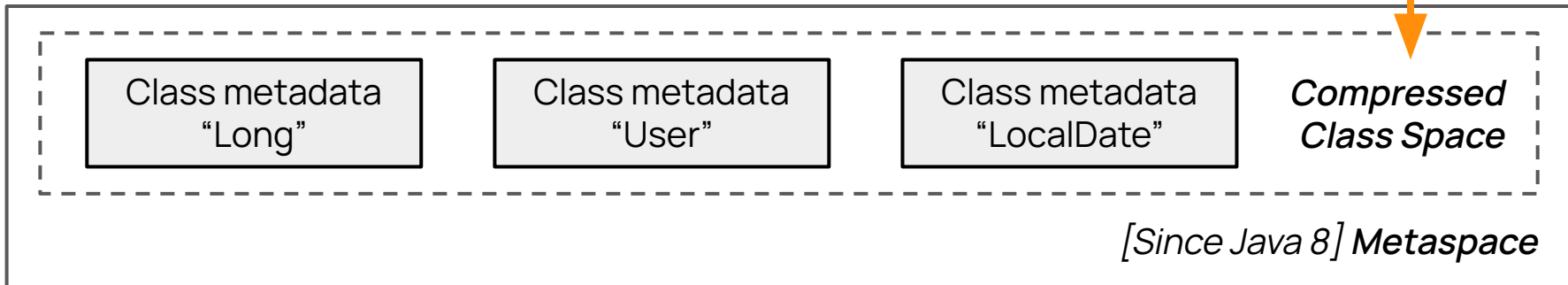
Compressed Class Pointers

Class metadata size: 512~1024 bytes

Addressable with 32 bit: 4 GB / 768 bytes
≈ 5,600,000



4 GB maximum



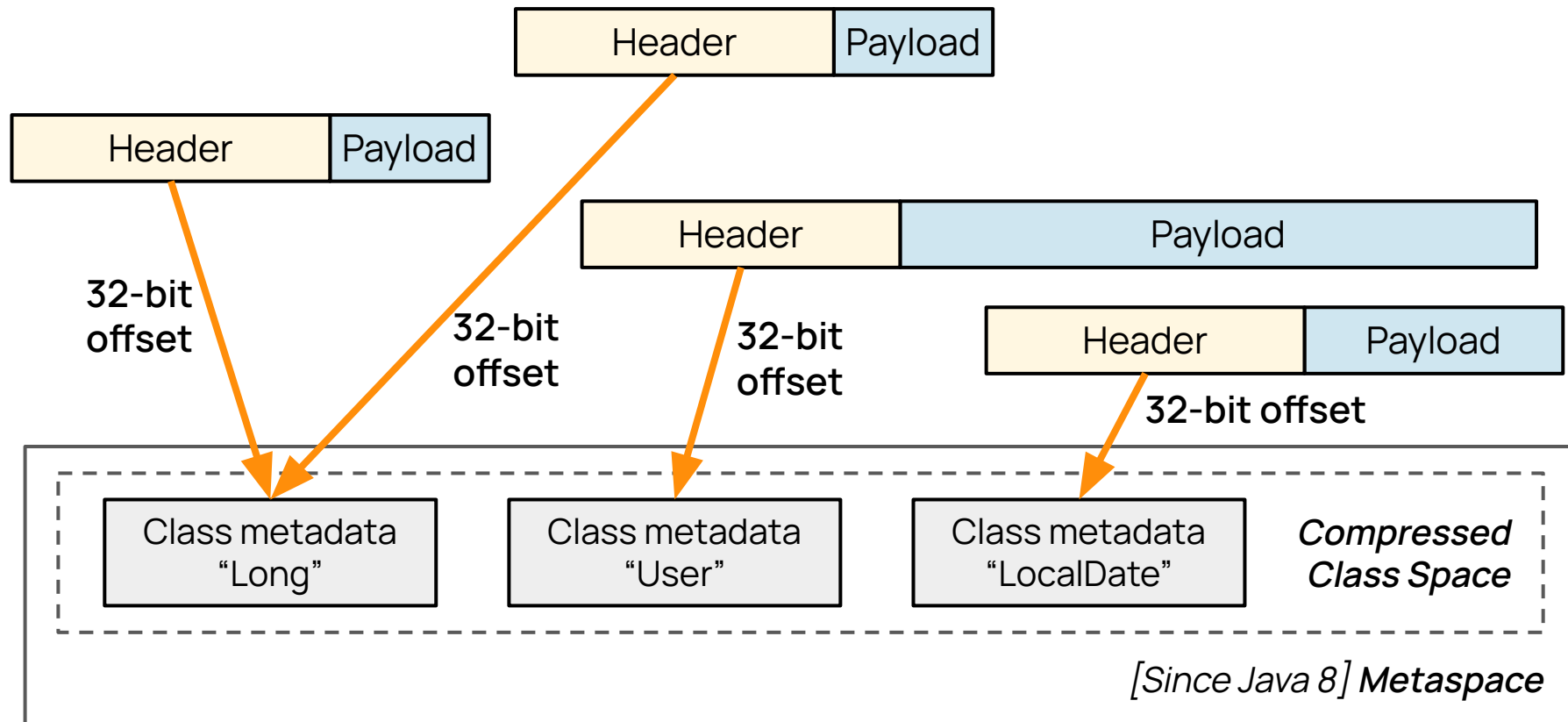
Compressed Class Pointers

Disable Compressed Class Pointers:

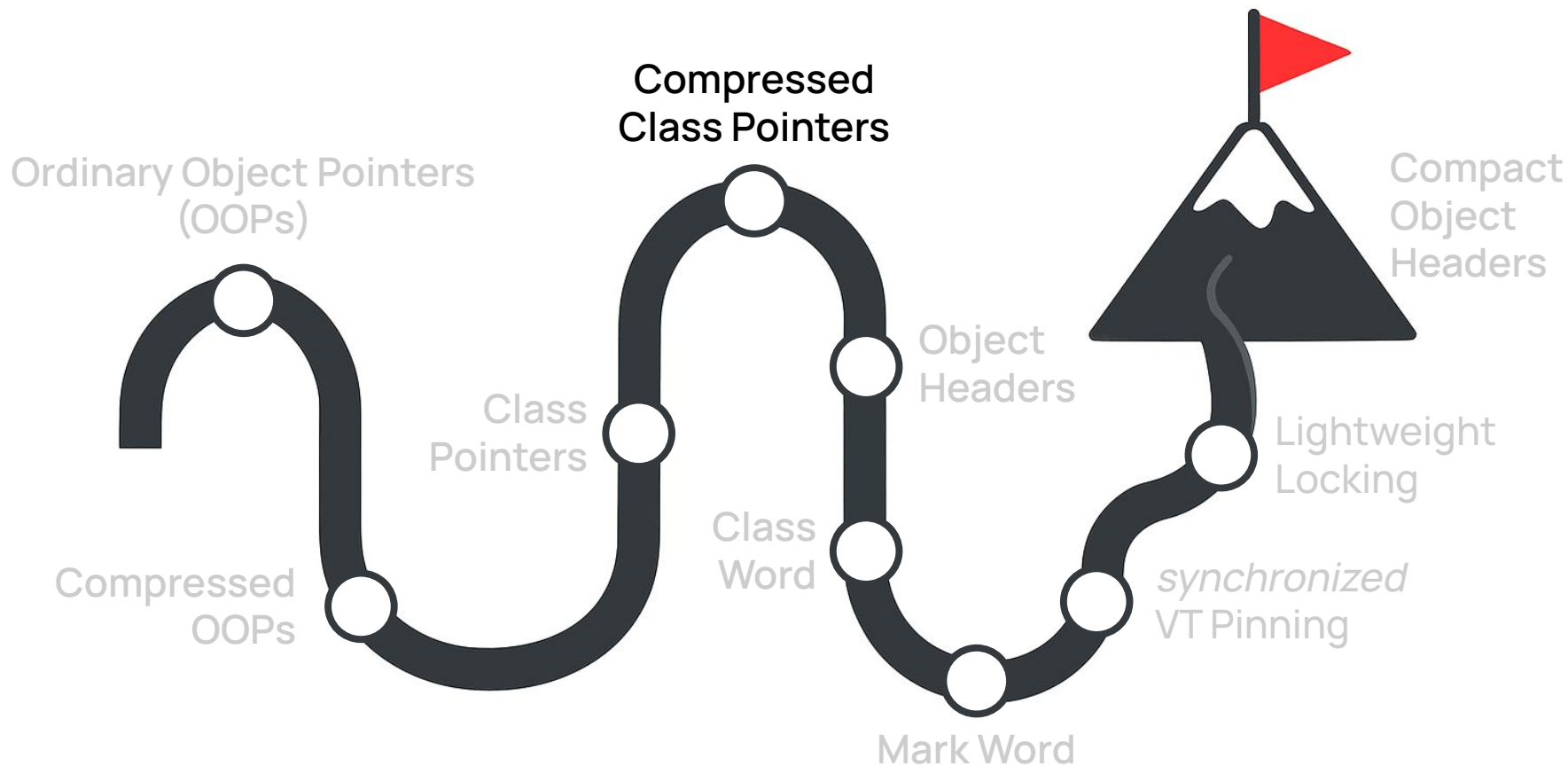
-XX:-UseCompressedClassPointers

since Java 25:
Deprecated

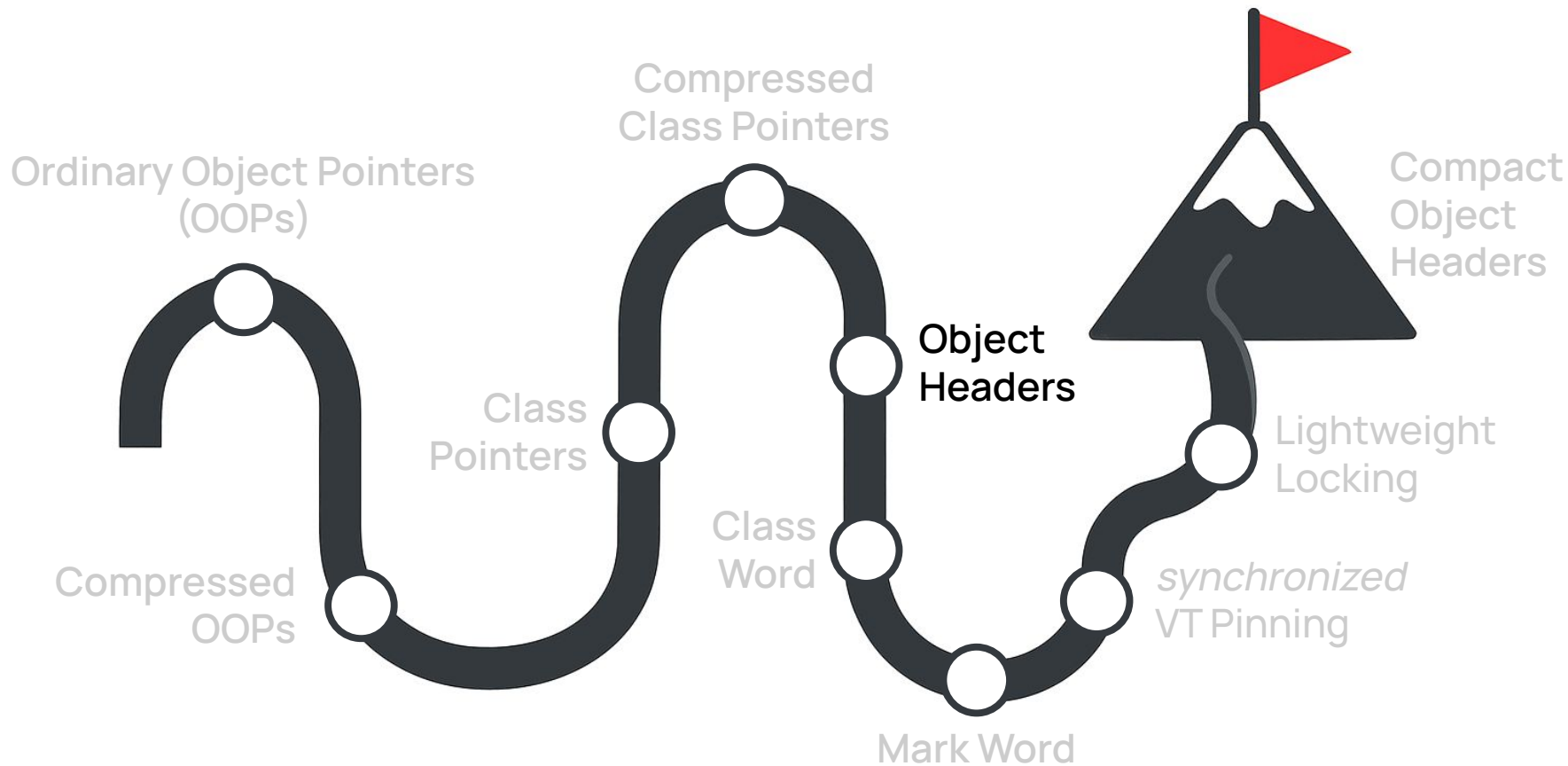
Compressed Class Pointers



Journey into the JVM

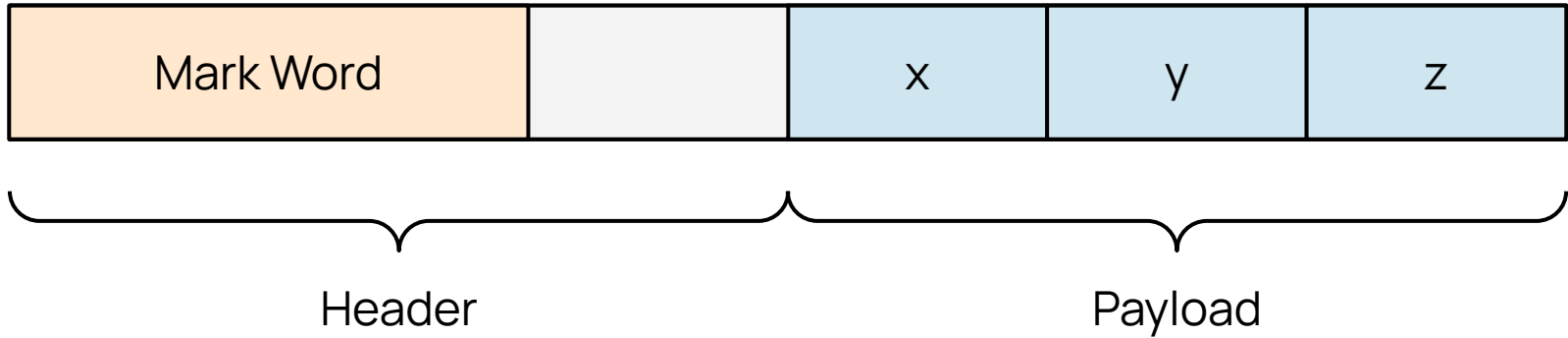


Journey into the JVM



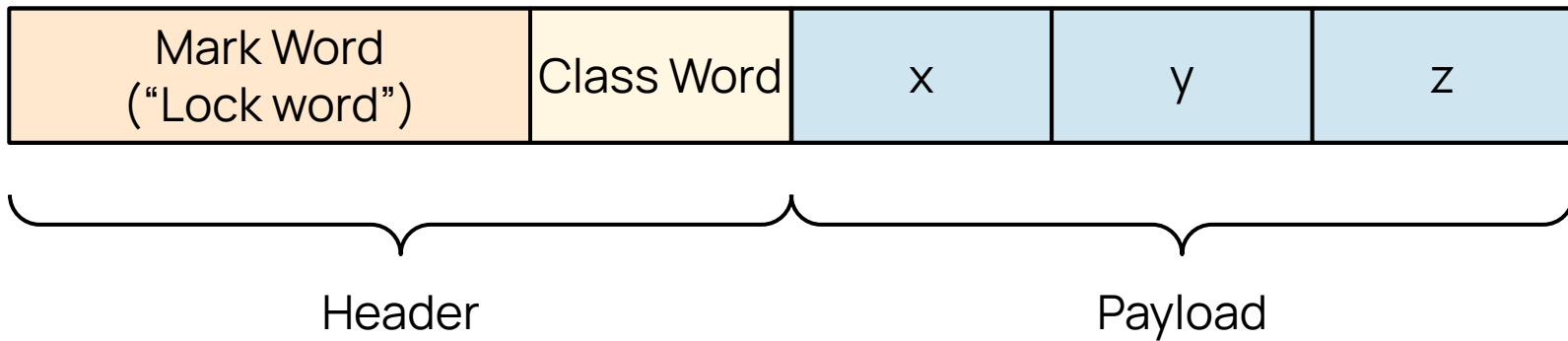
Object Headers

A Java object:

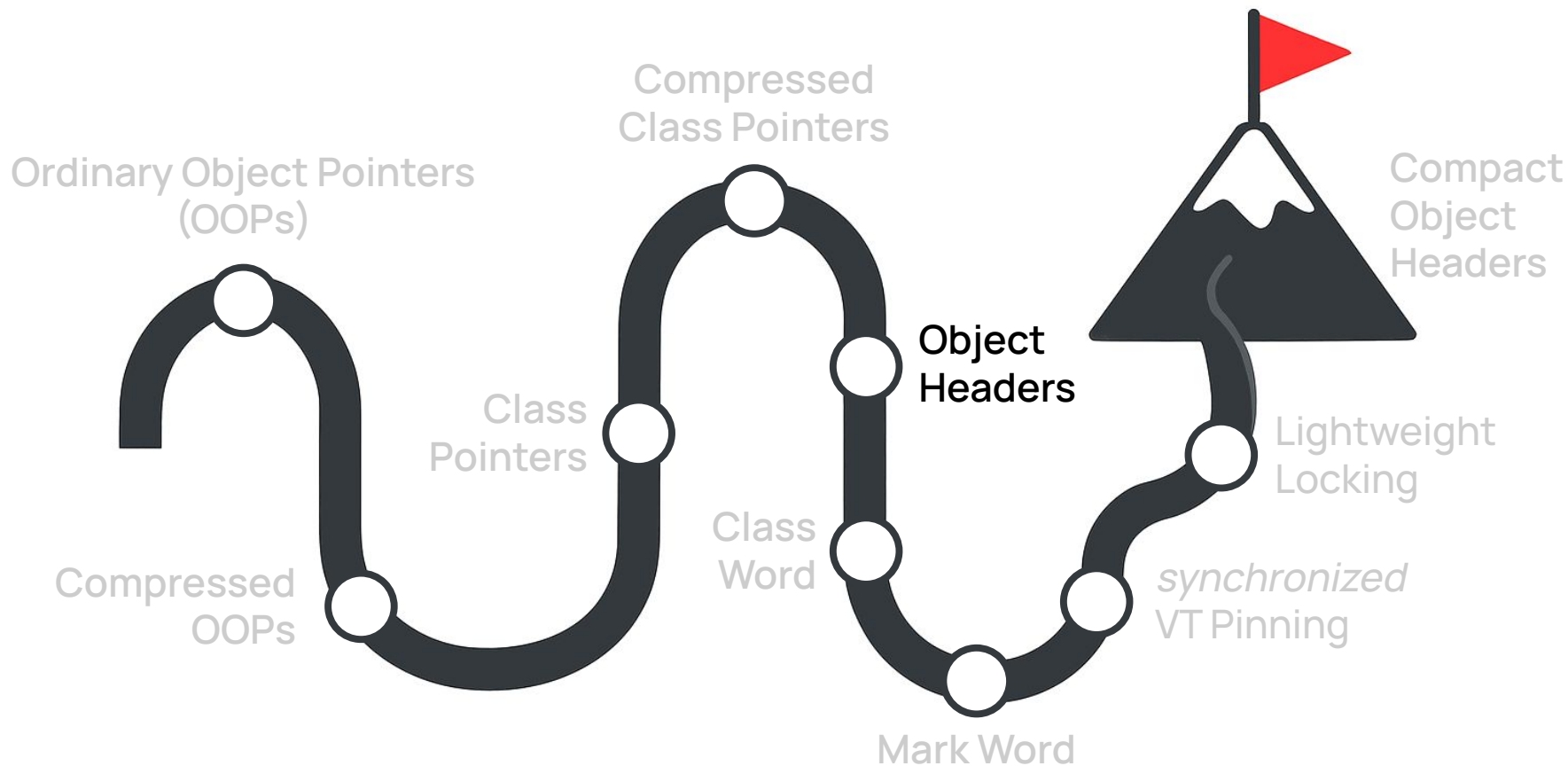


Object Headers

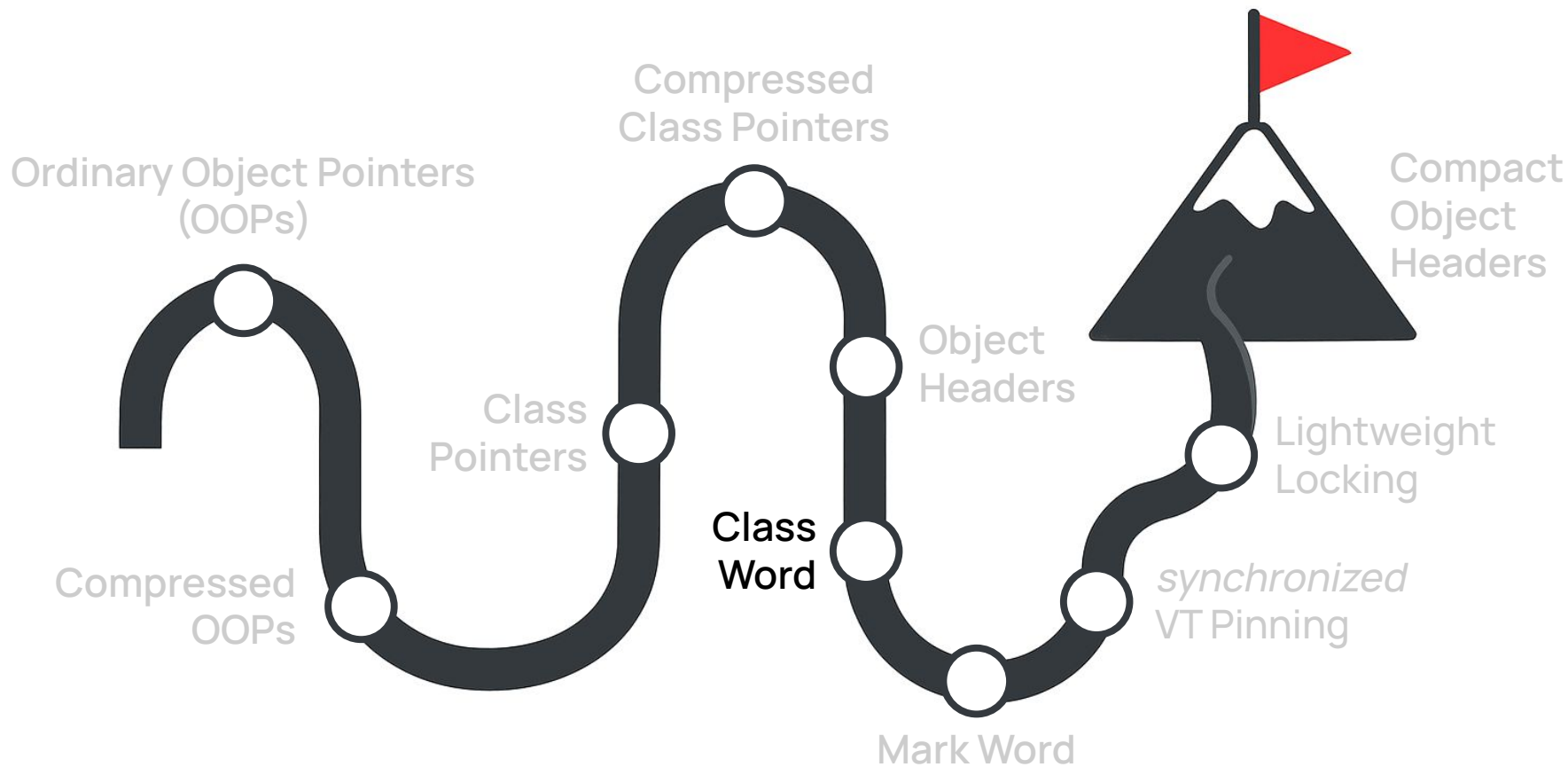
A Java object:



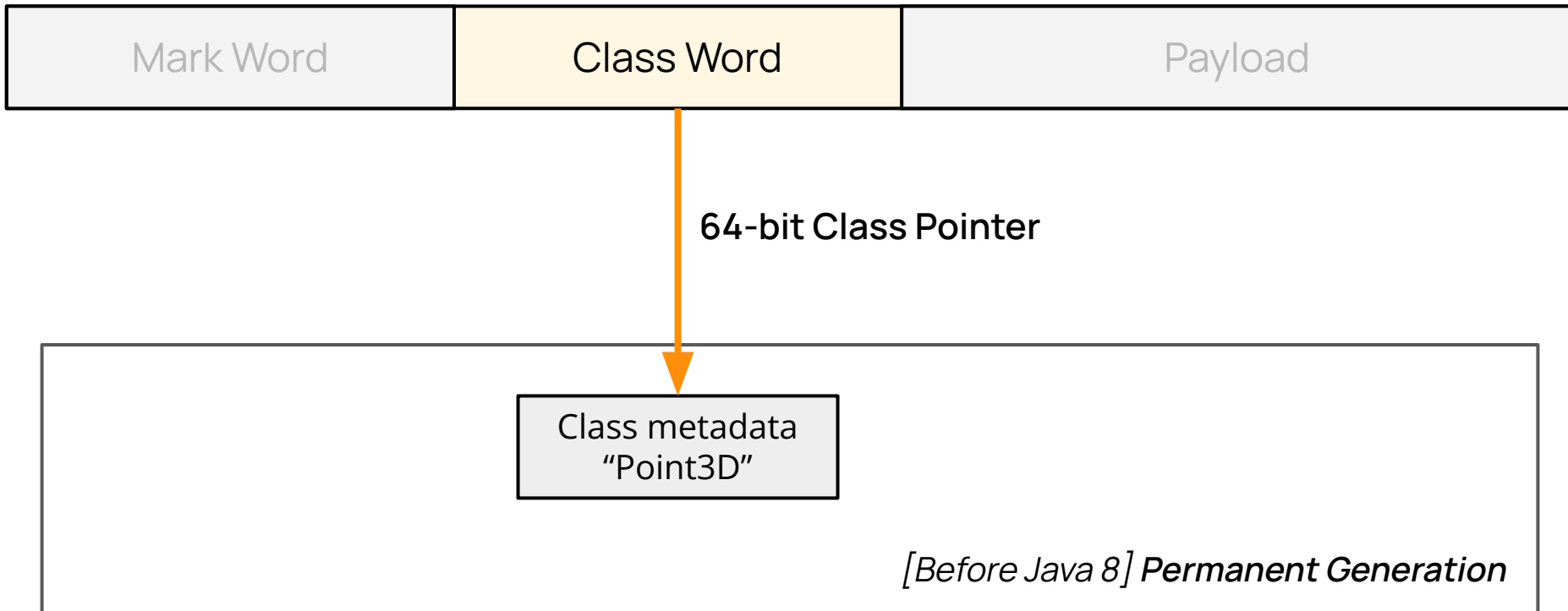
Journey into the JVM



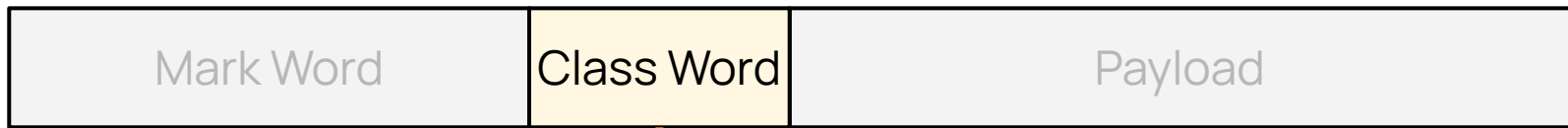
Journey into the JVM



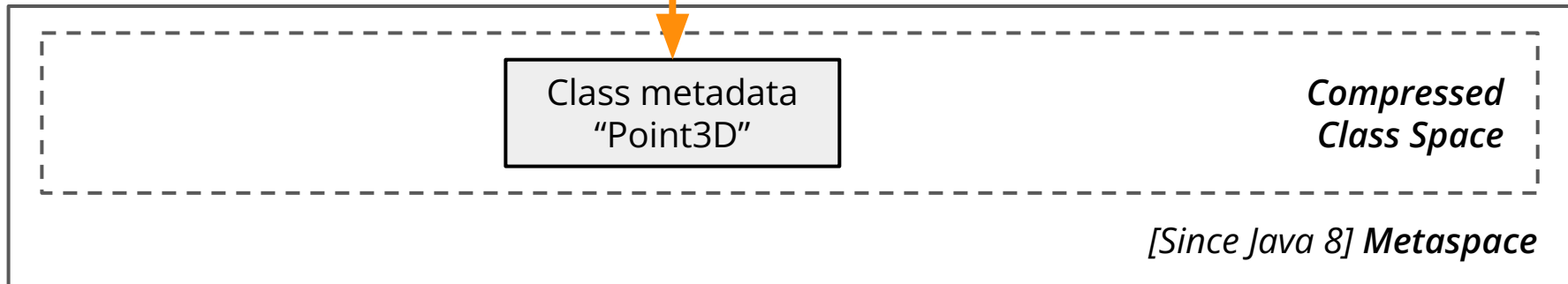
Class Word



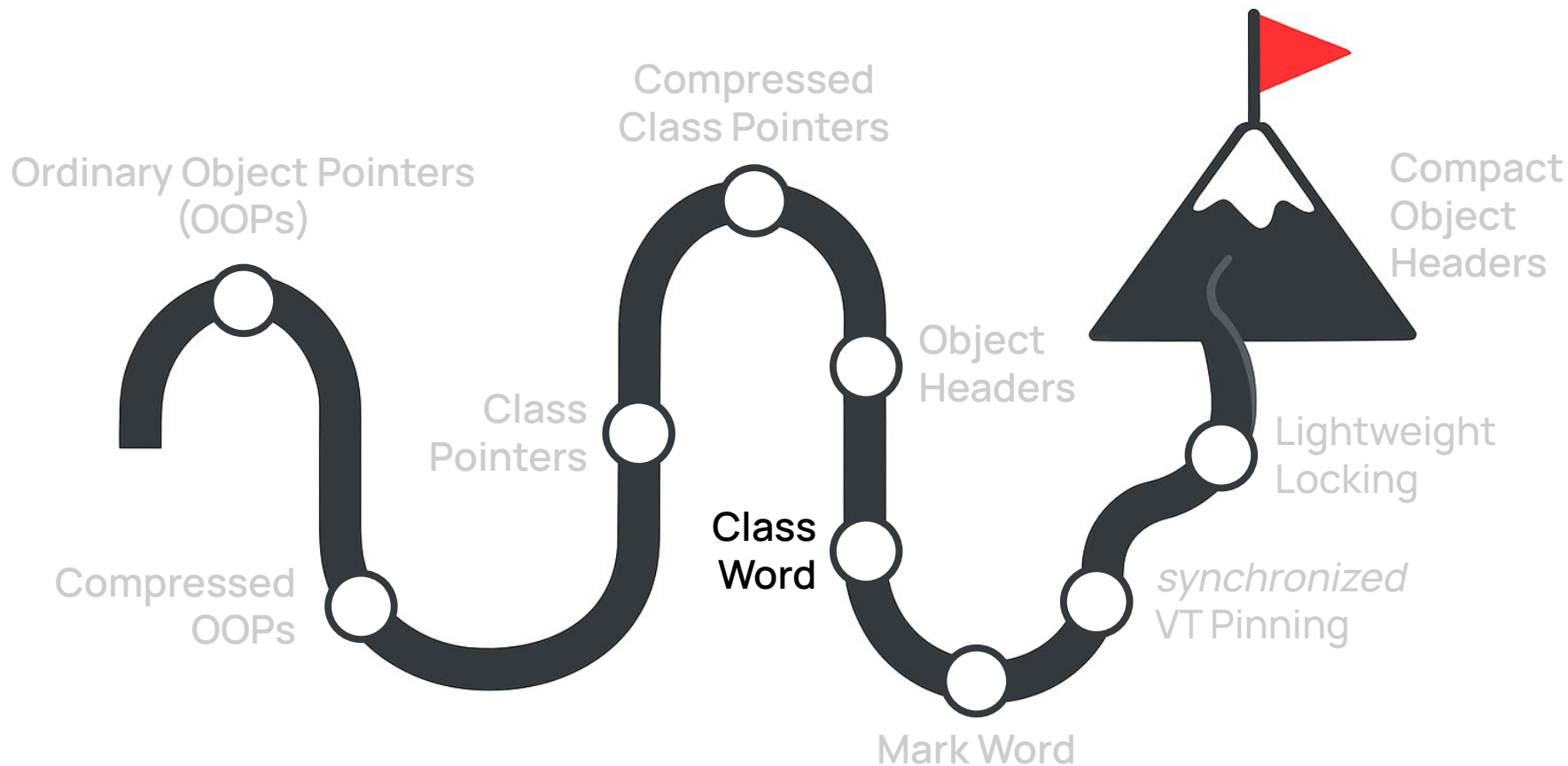
Class Word



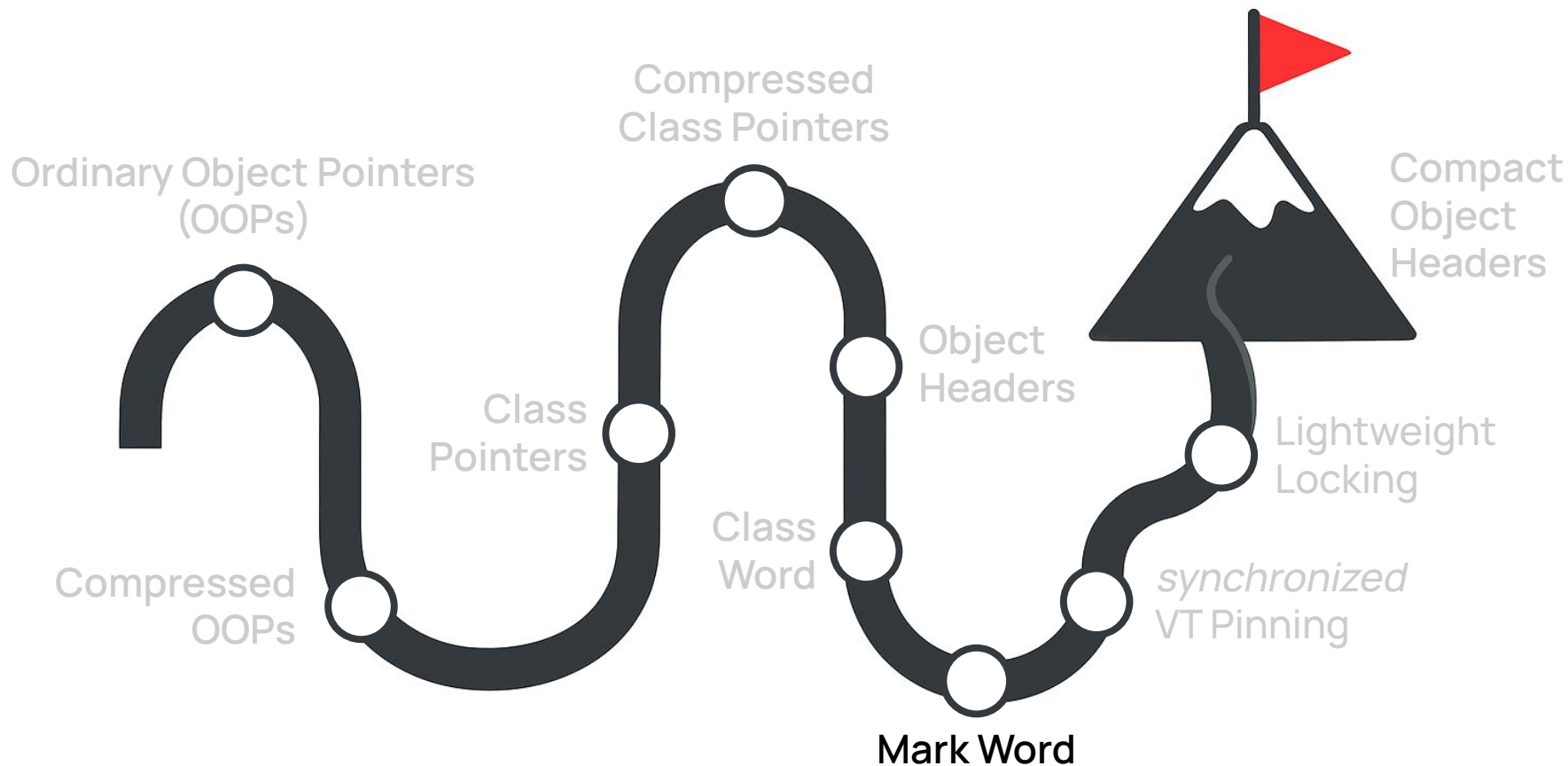
32-bit Compressed Class Pointer (= 32-bit offset)



Journey into the JVM



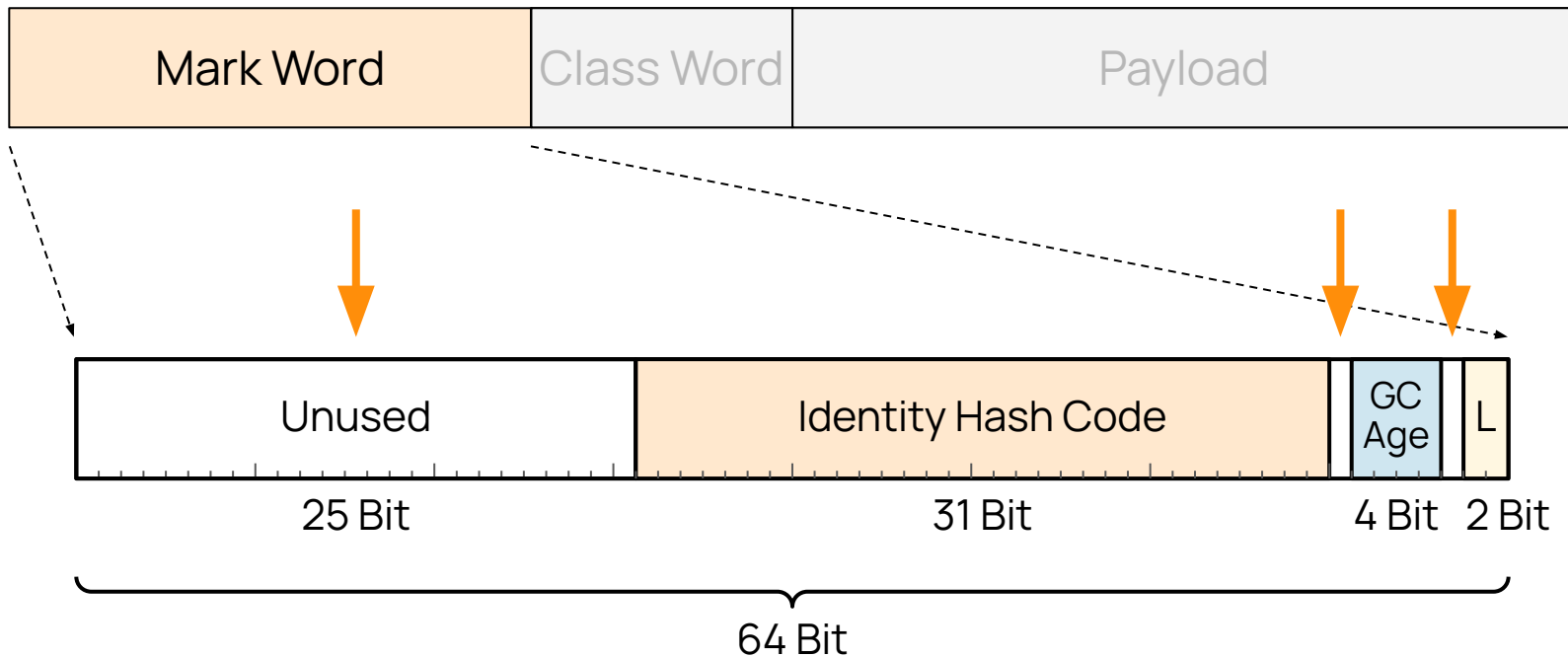
Journey into the JVM



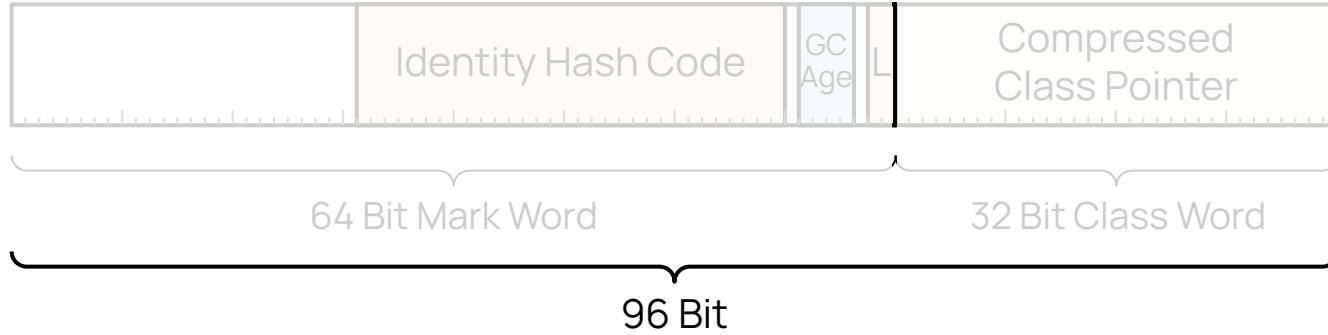
Mark Word



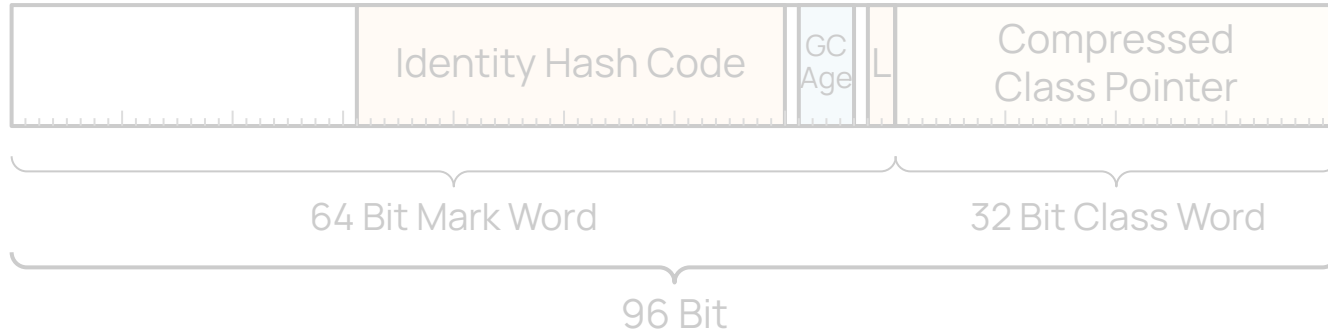
Mark Word



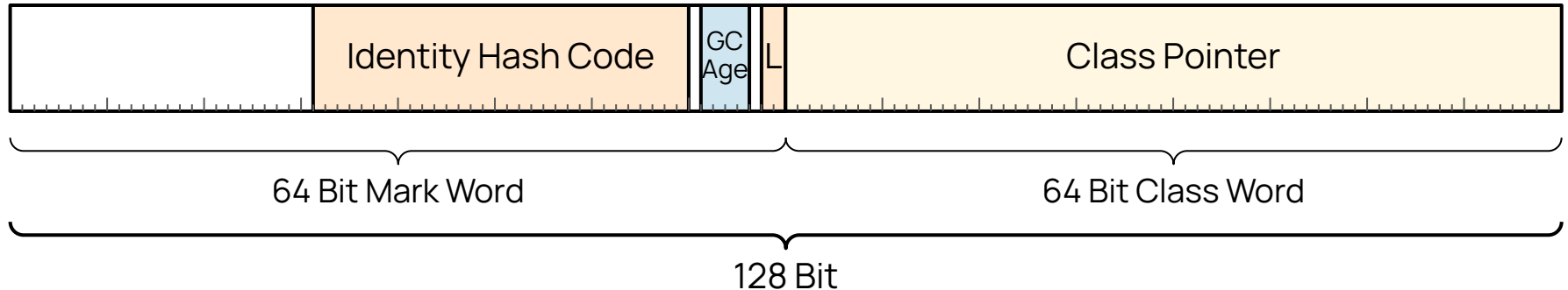
Object Header



Object Header

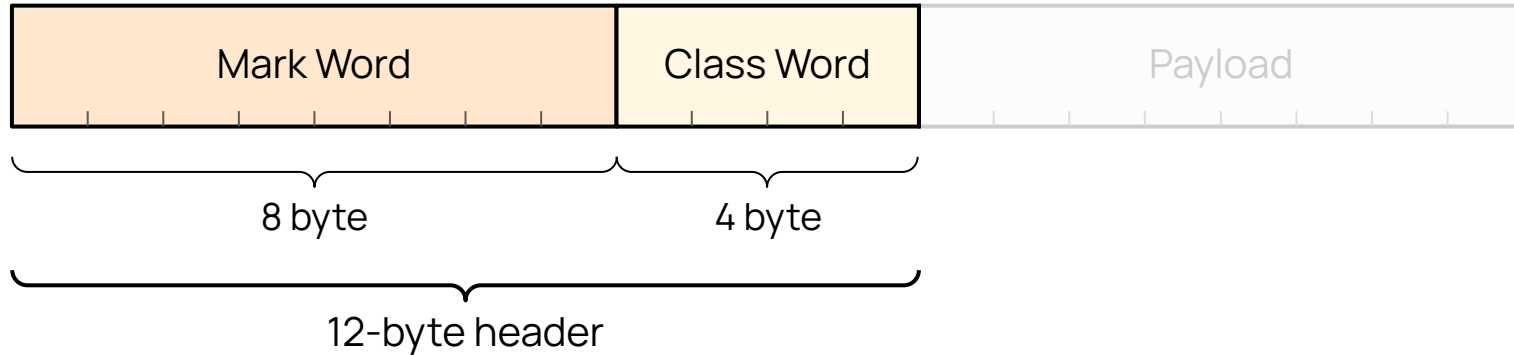


-XX:-UseCompressedClassPointers (deprecated since Java 25):



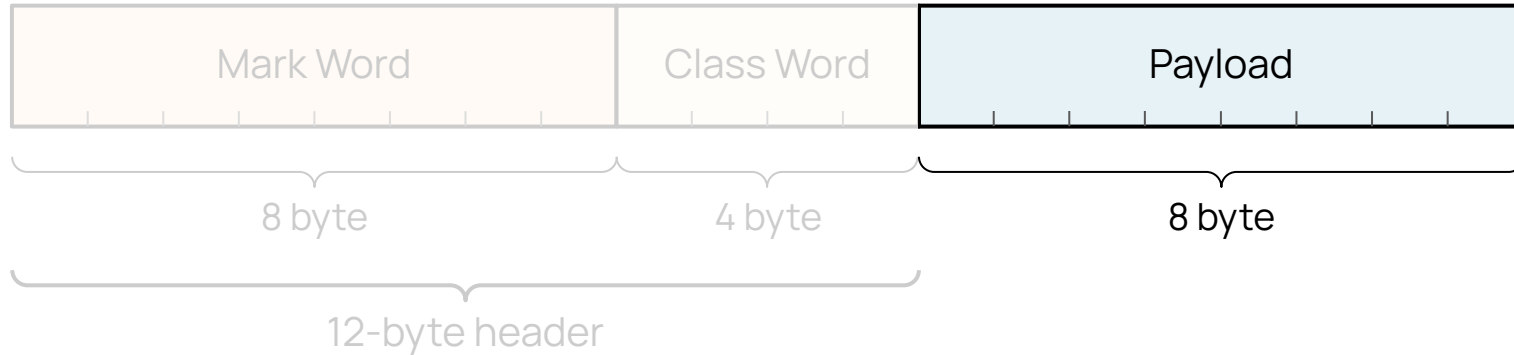
Java Object with Header and Payload

A `java.lang.Long` object:



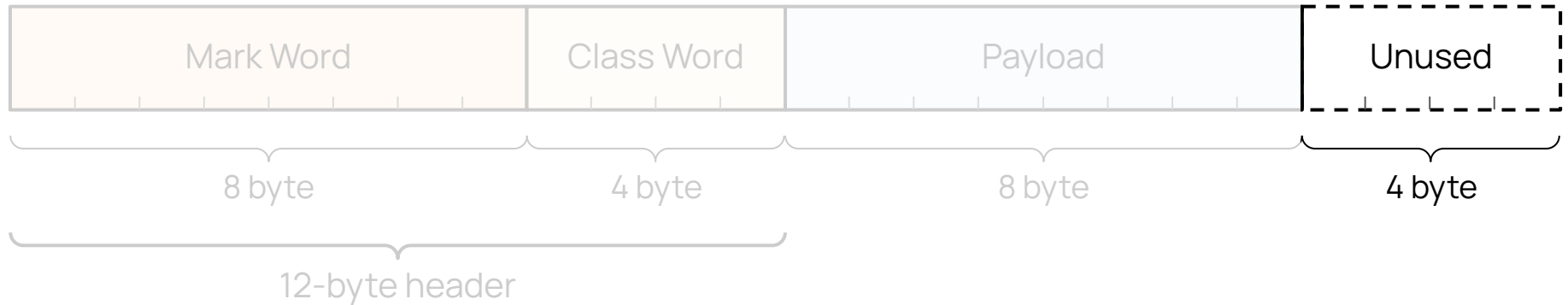
Java Object with Header and Payload

A `java.lang.Long` object:



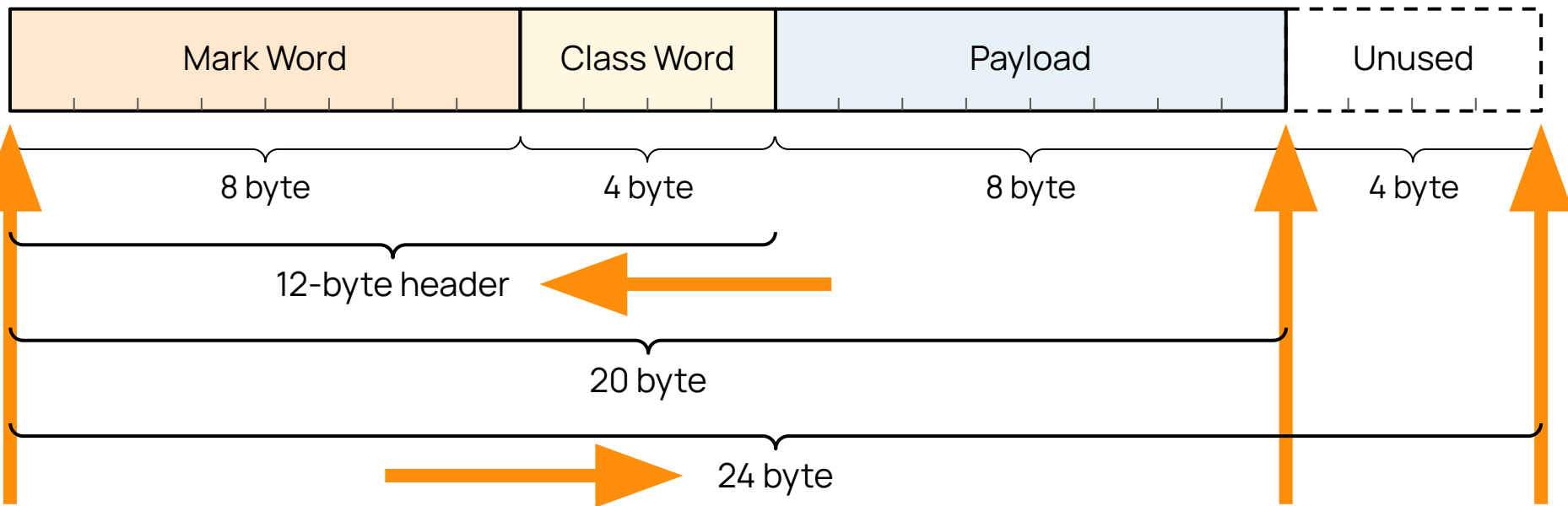
Java Object with Header and Payload

A `java.lang.Long` object:



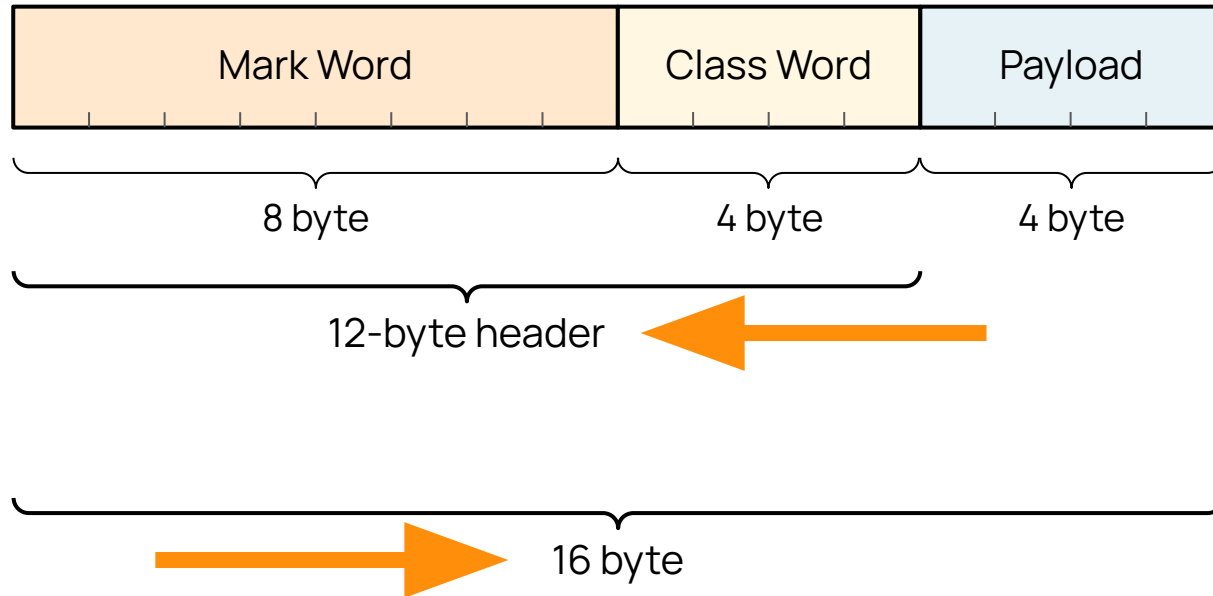
Java Object with Header and Payload

A `java.lang.Long` object:



Java Object with Header and Payload

A `java.lang.Integer` object:



Java Object Size

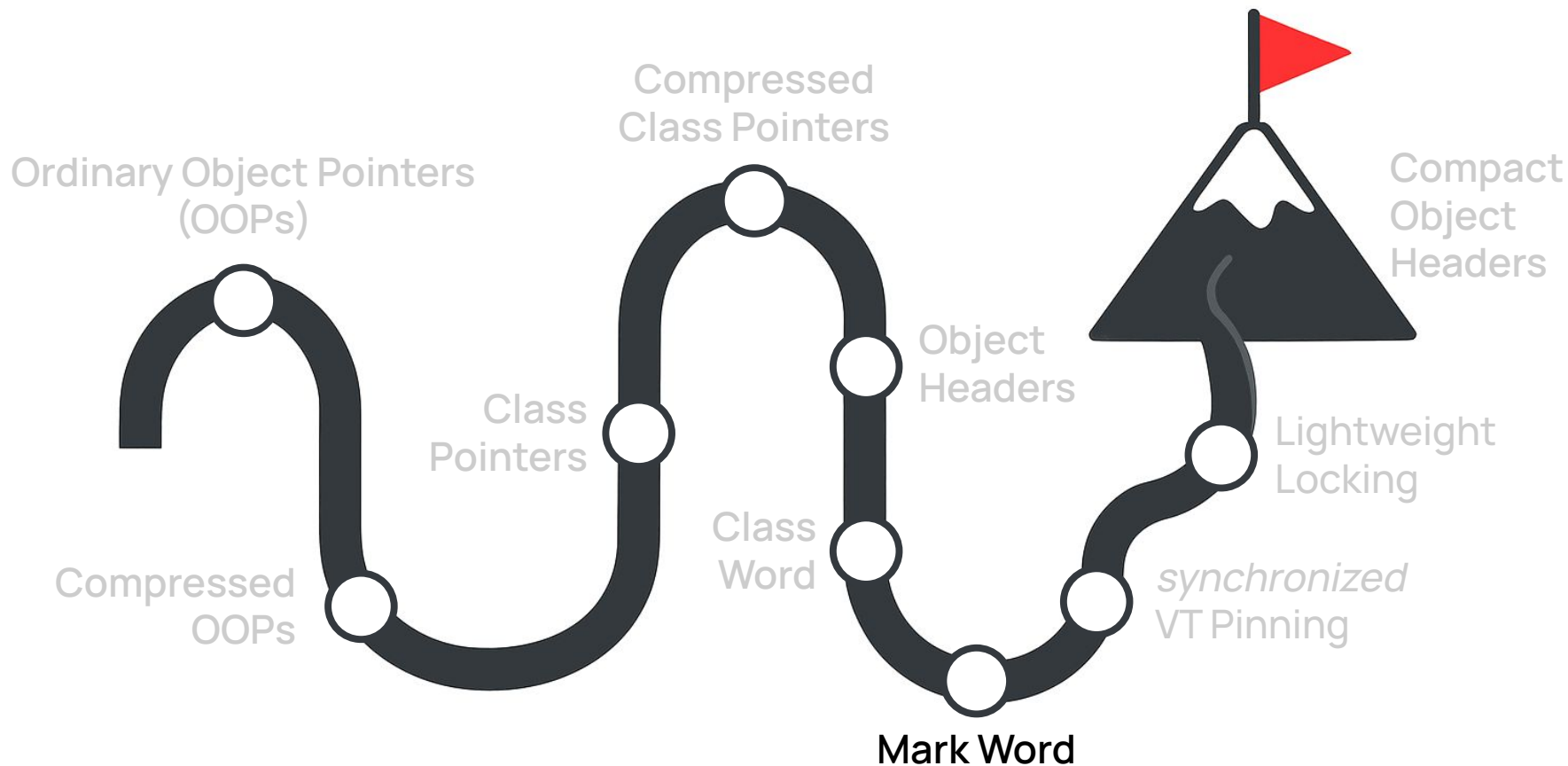
Typical object size: 32~64 bytes

Typical object header overhead: ~ 20 %

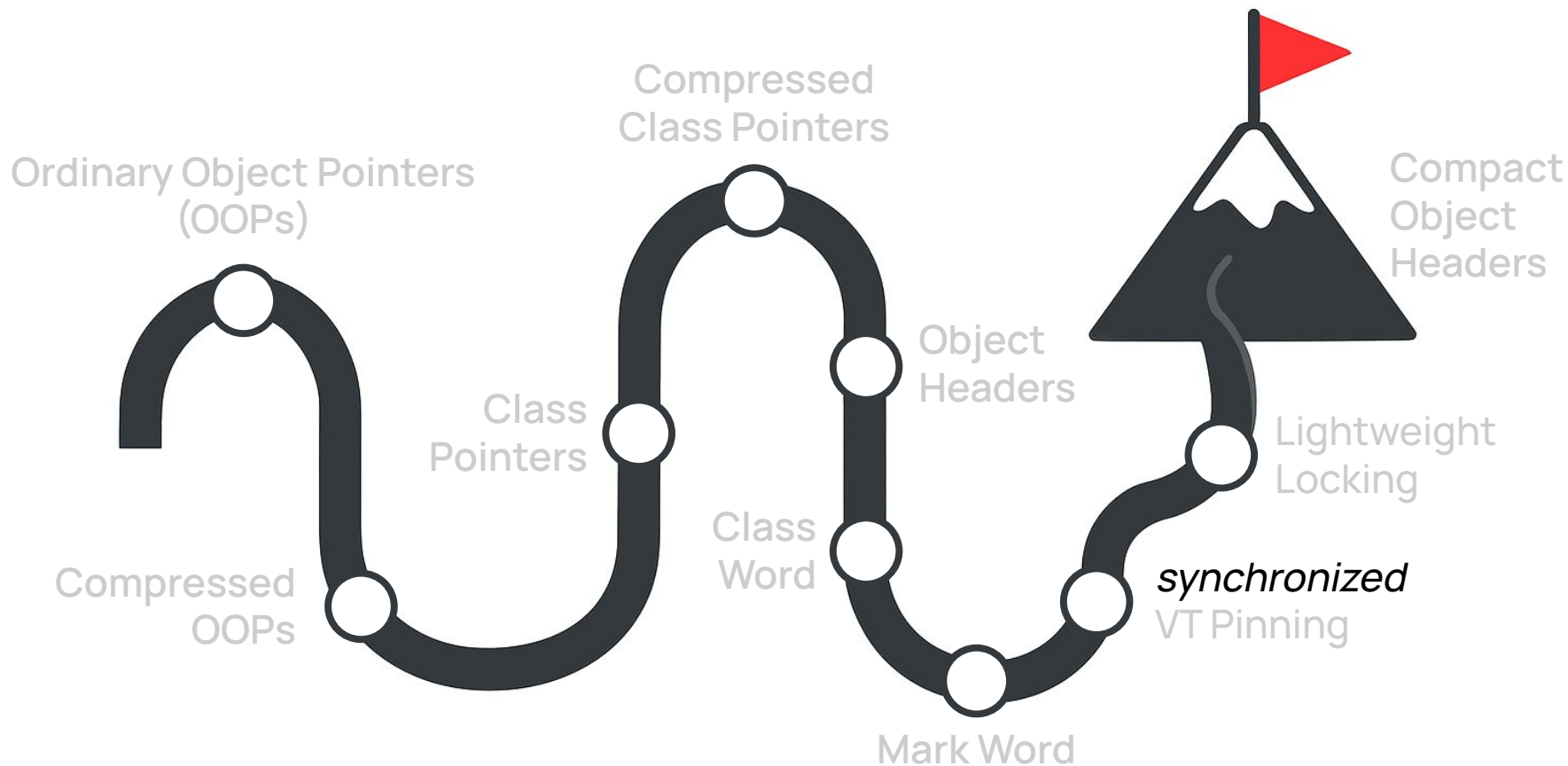
→ Project “Lilliput”

“The goal of this Project is to explore techniques to downsize Java object headers in the Hotspot JVM from 128 bits to 64 bits or less, reducing Java’s memory footprint.”

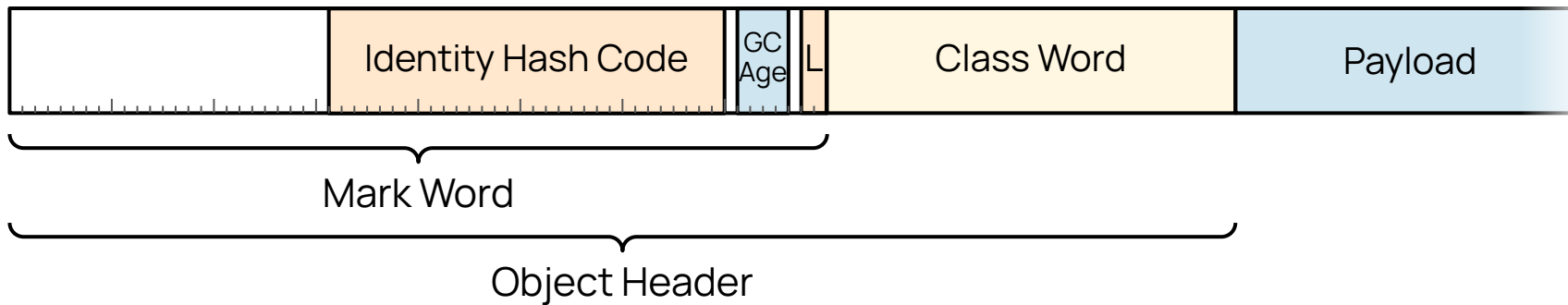
Journey into the JVM



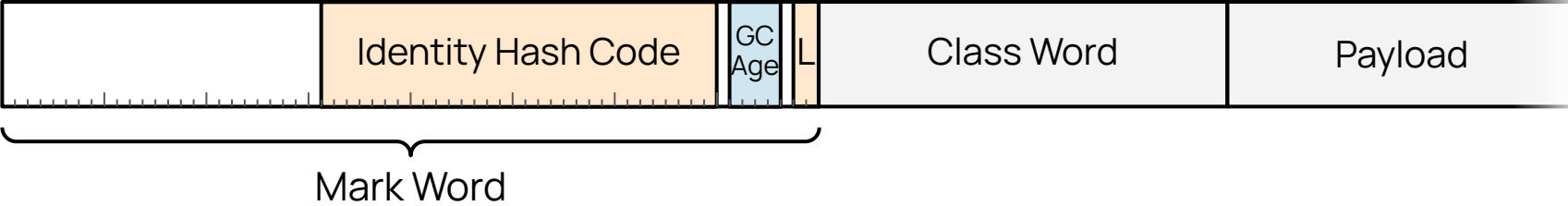
Journey into the JVM



Java Heap



Java Heap

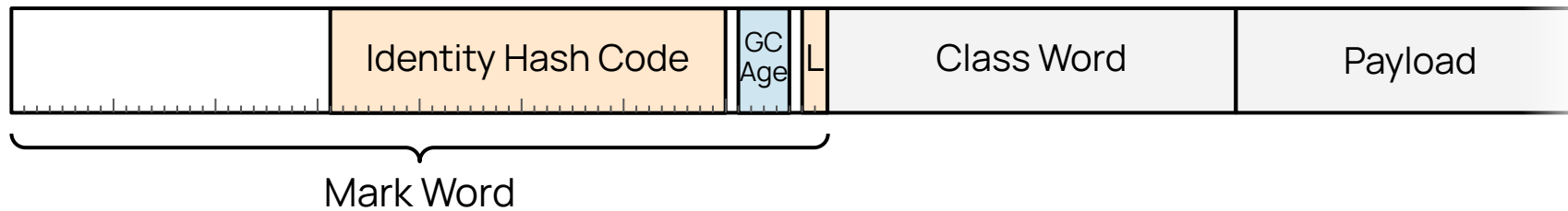


Thread Stack

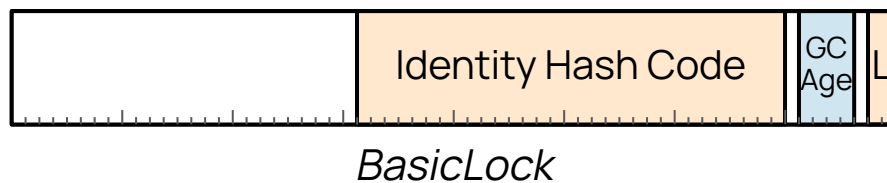


BasicLock

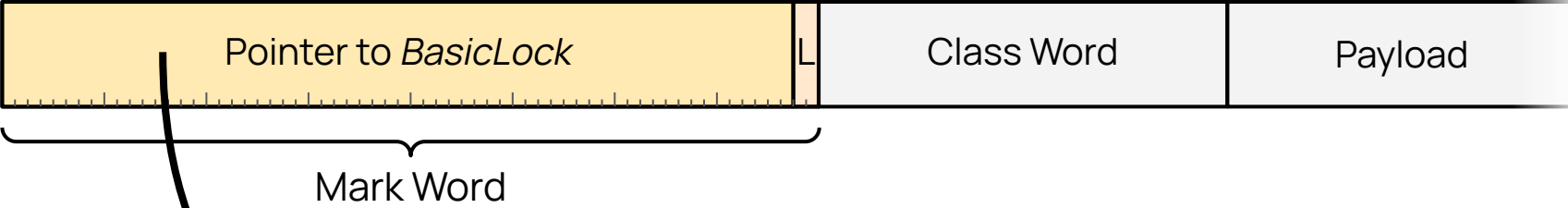
Java Heap



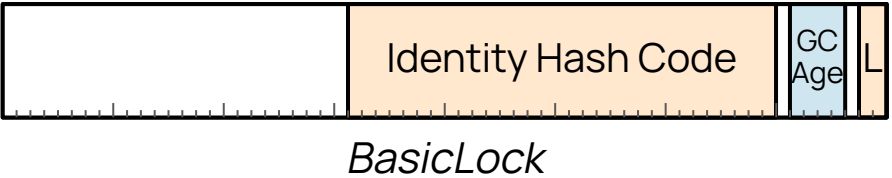
Thread Stack



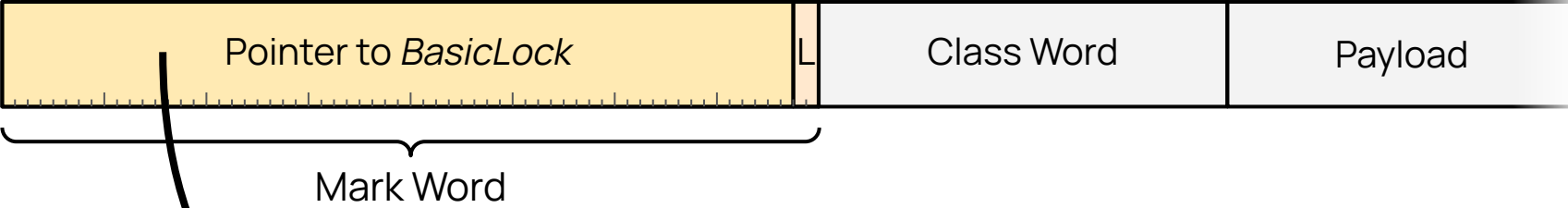
Java Heap



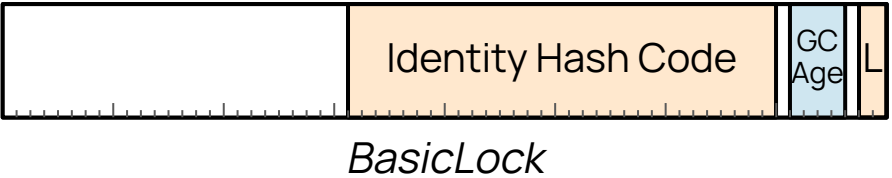
Thread Stack



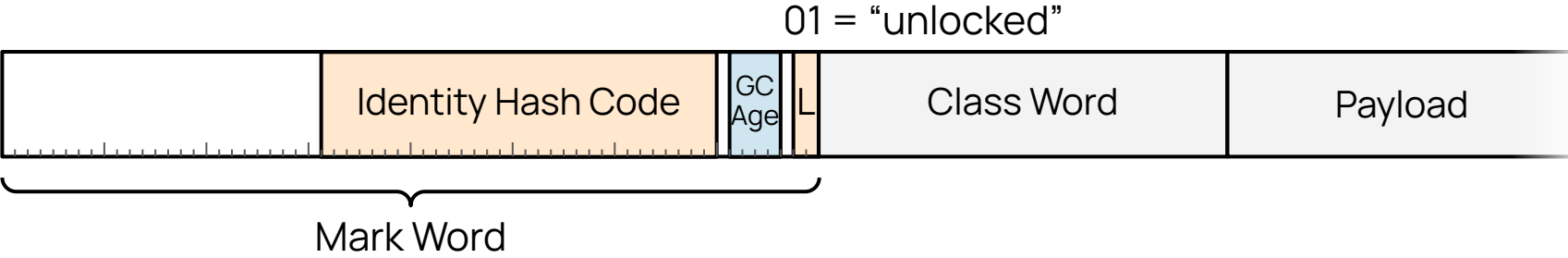
Java Heap



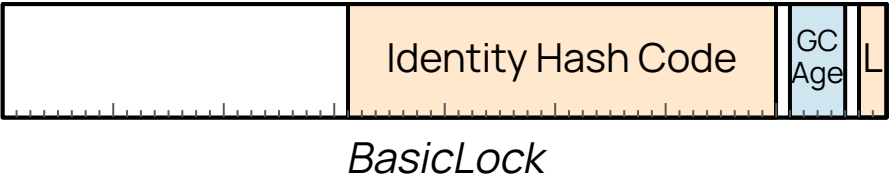
Thread Stack



Java Heap



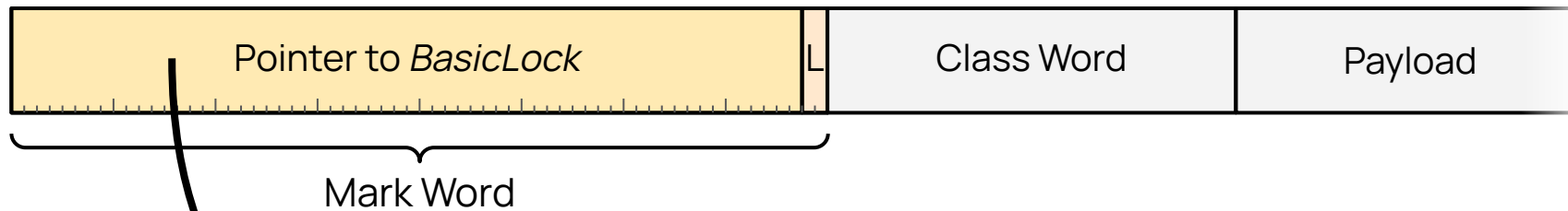
Thread Stack



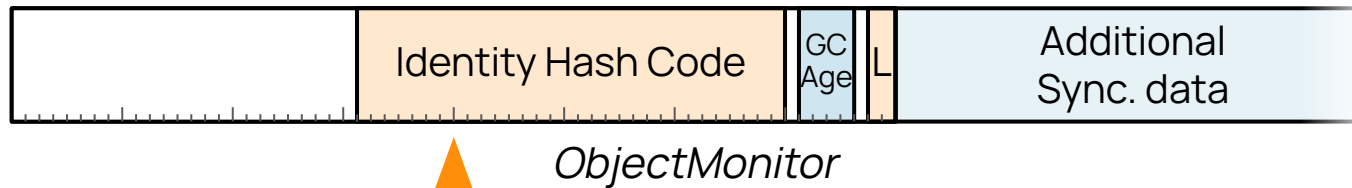
Java Heap



00 = "locked"



JVM
Memory

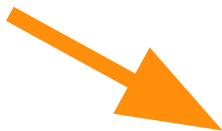


Thread Stack

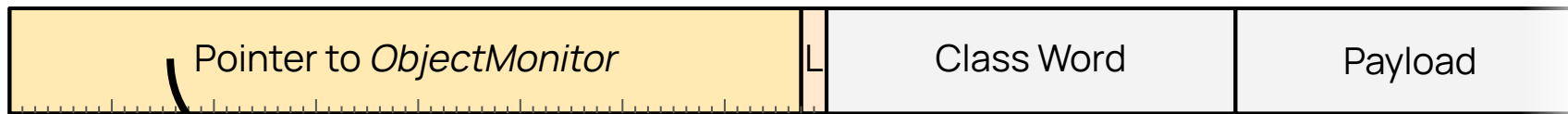


BasicLock

Java Heap

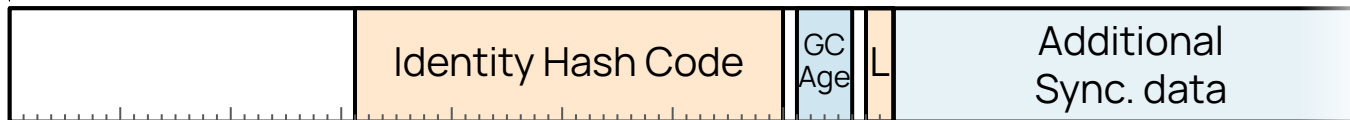


00 = "locked"



Mark Word

JVM
Memory

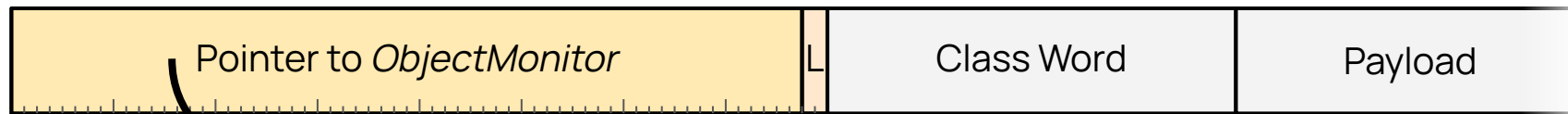


ObjectMonitor

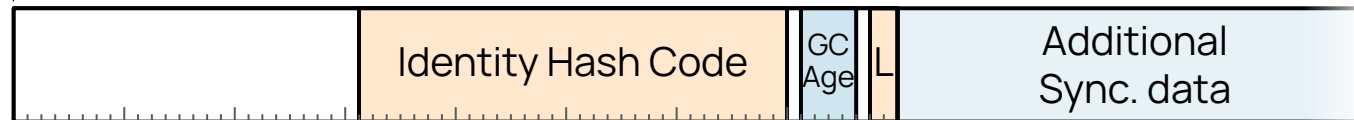
Thread Stack

Java Heap

10 = "inflated"



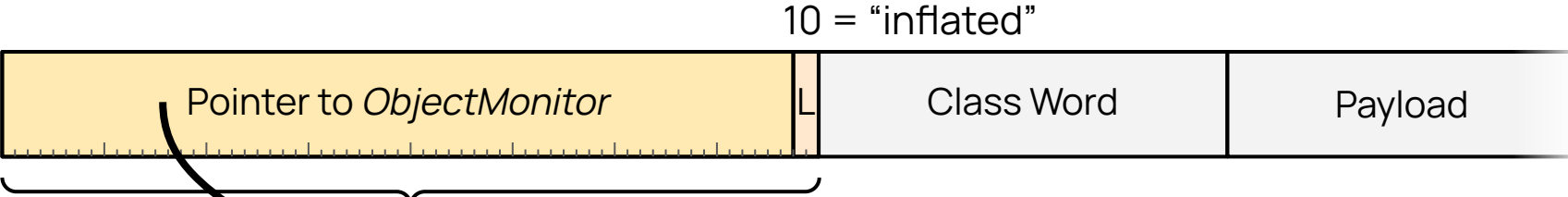
JVM
Memory



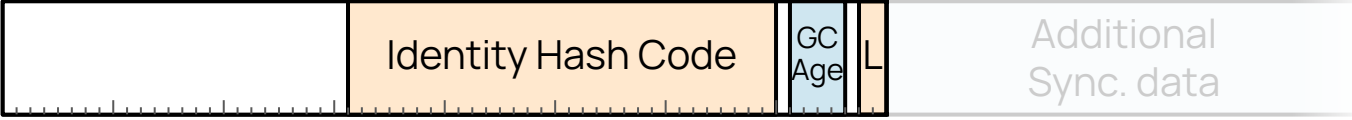
ObjectMonitor

Thread Stack

Java Heap



JVM
Memory

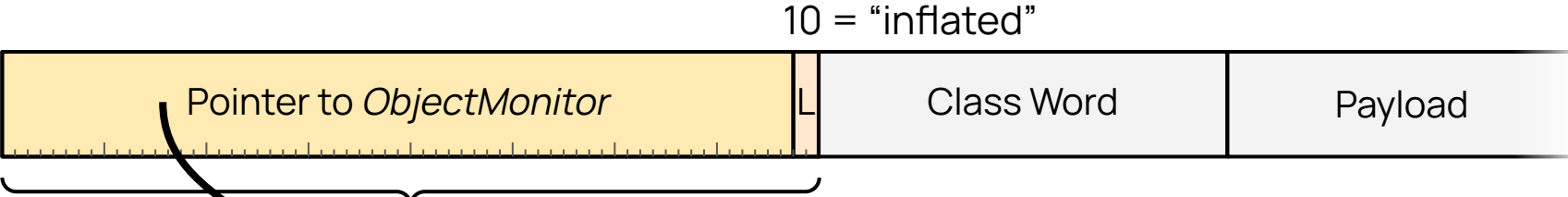


Thread Stack



ObjectMonitor

Java Heap



JVM Memory



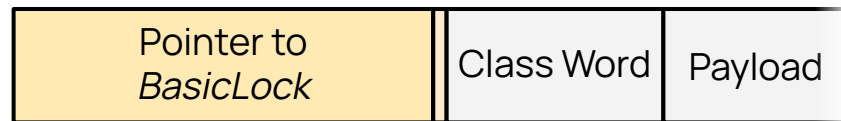
ObjectMonitor

Thread Stack



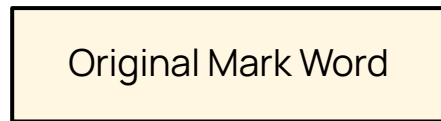
Java Heap

00 = "locked"



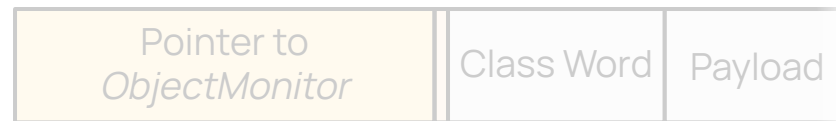
JVM
Memory

Thread Stack



BasicLock

10 = "inflated"

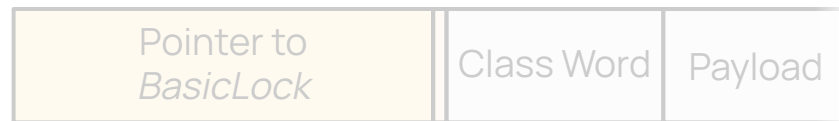


ObjectMonitor



Java Heap

00 = "locked"



JVM
Memory

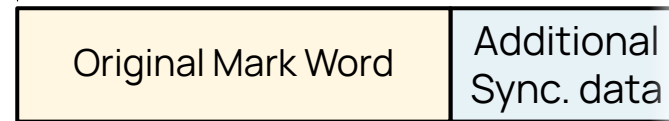
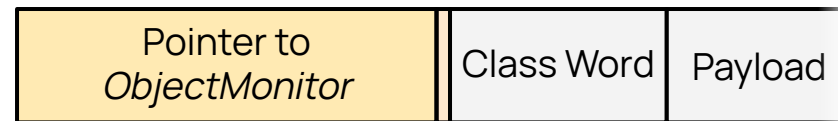
Thread Stack



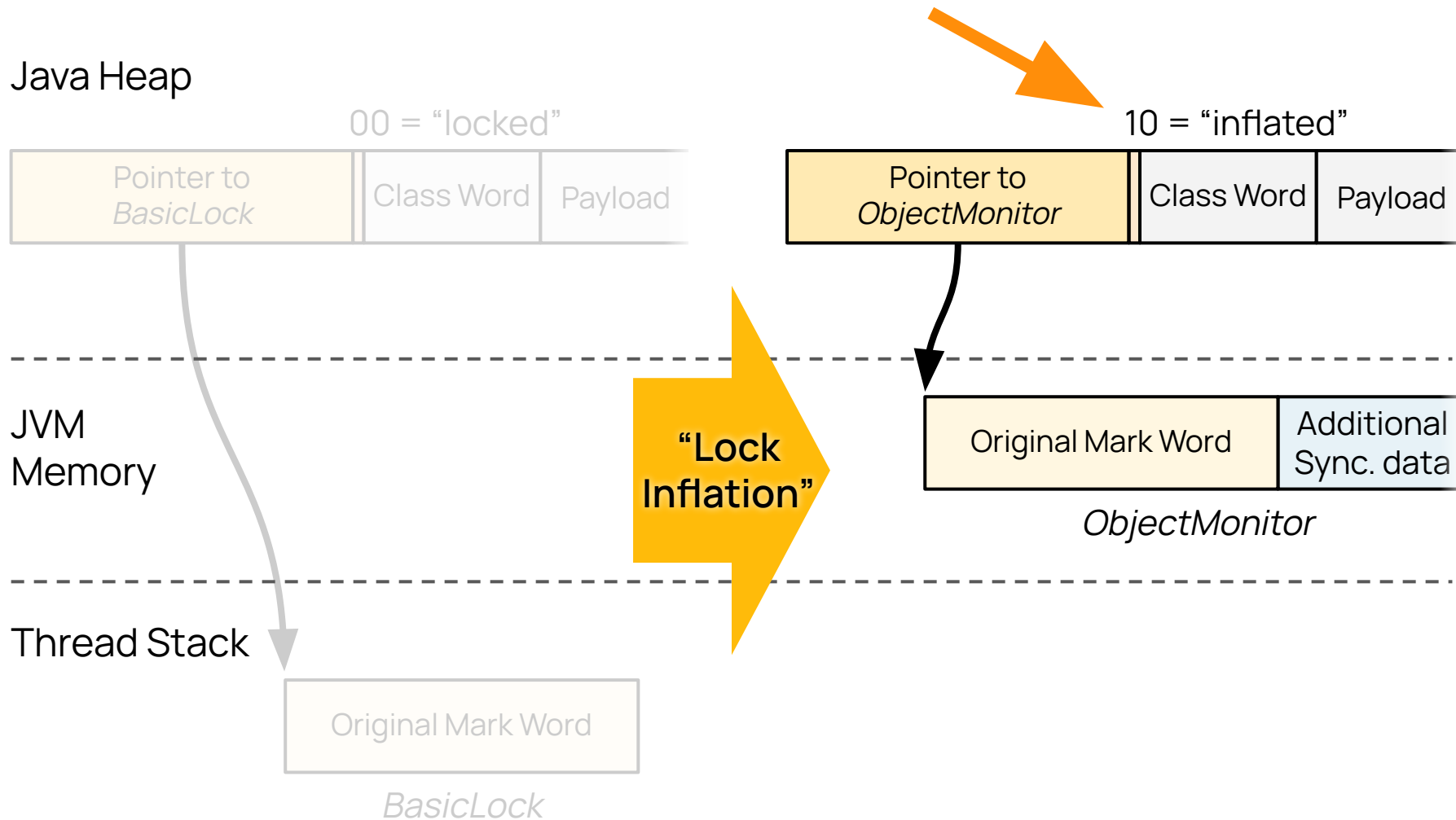
BasicLock

"Lock
Inflation"

10 = "inflated"

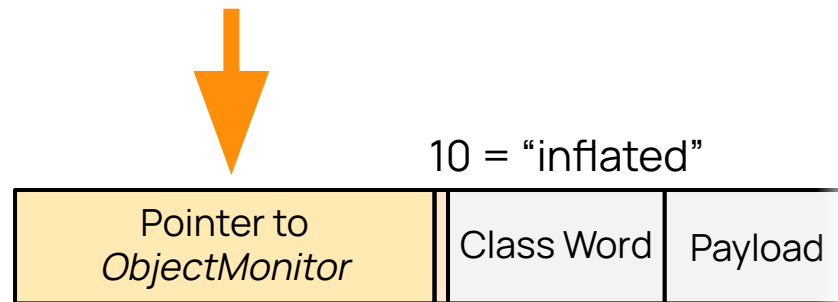
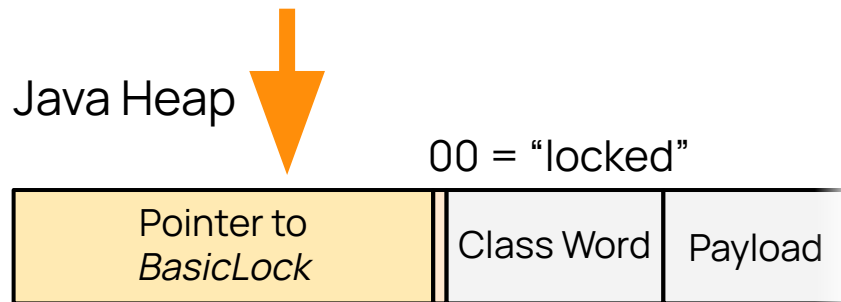


ObjectMonitor

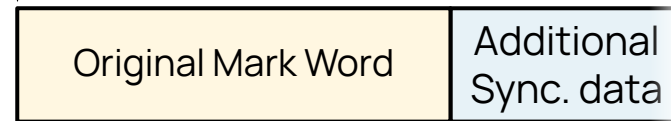


“Stack Locking”

“Legacy Stack Locking”

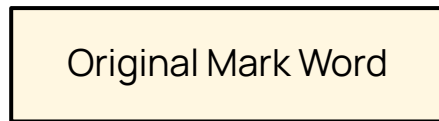


JVM
Memory

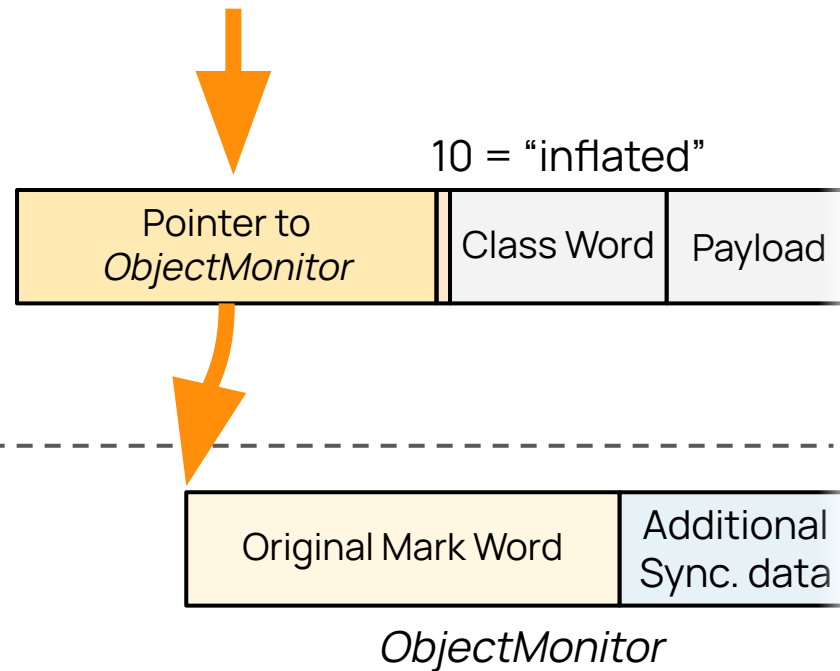
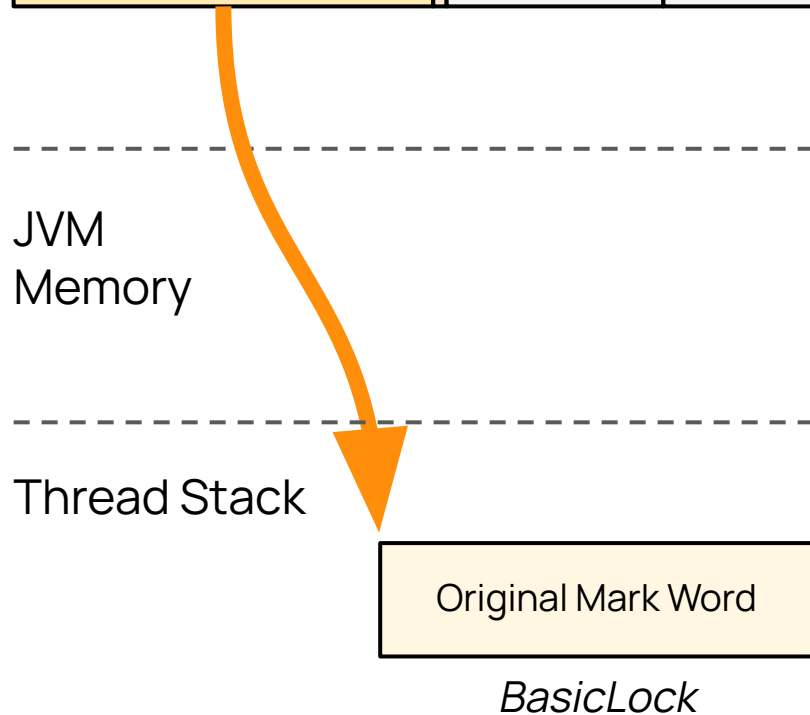
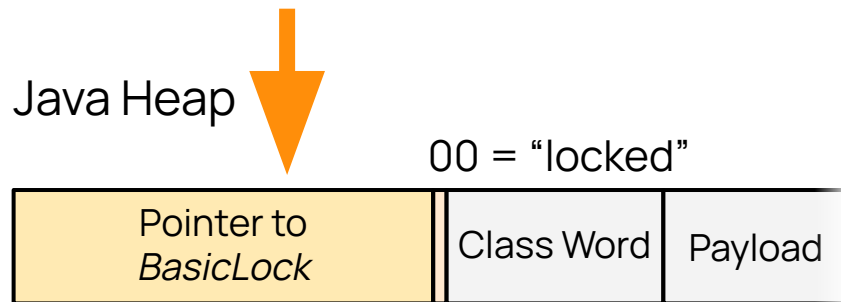


ObjectMonitor

Thread Stack

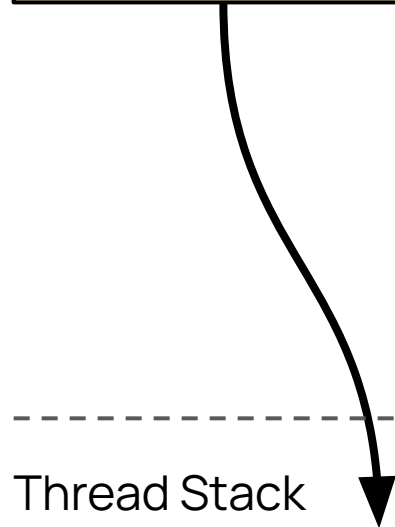
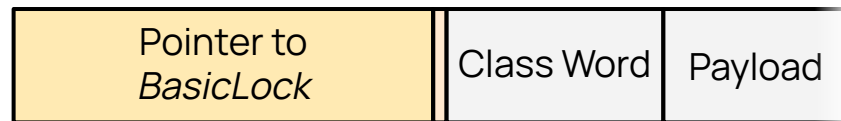


BasicLock



Java Heap

00 = "locked"

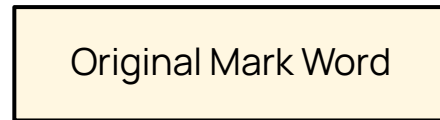


Thread Stack
Carrier Tread 1



BasicLock

Thread Stack
Carrier Tread 2



BasicLock

Java Heap

00 = "locked"



Thread Stack
Carrier Tread 1



Thread Stack
Carrier Tread 2

Original Mark Word

BasicLock

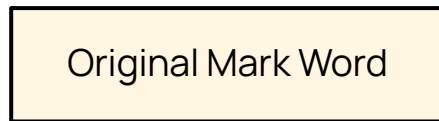
Java Heap

00 = "locked"



To avoid this:
VT is "**pinned**" to its Carrier Thread

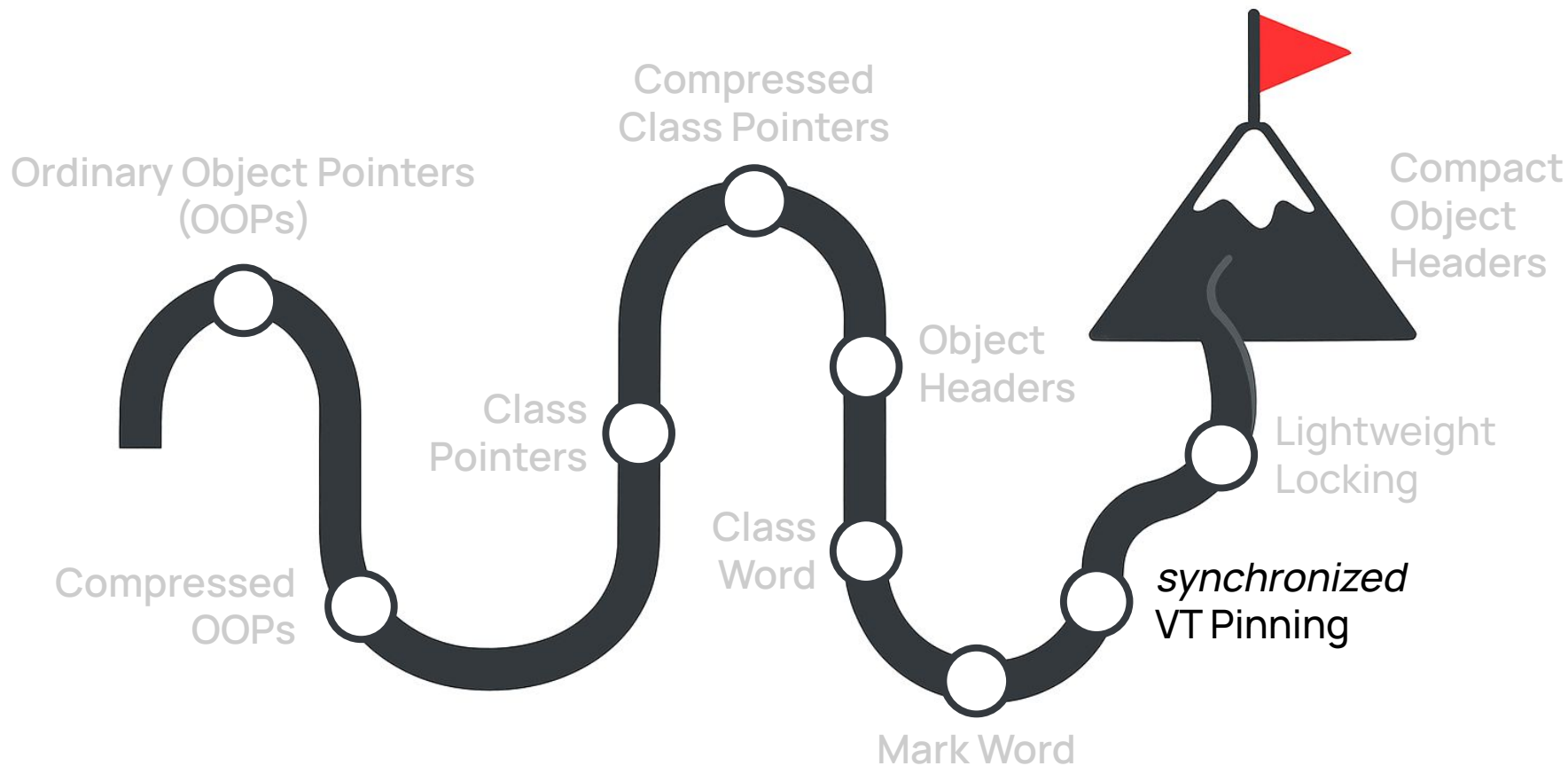
Thread Stack
Carrier Tread 1



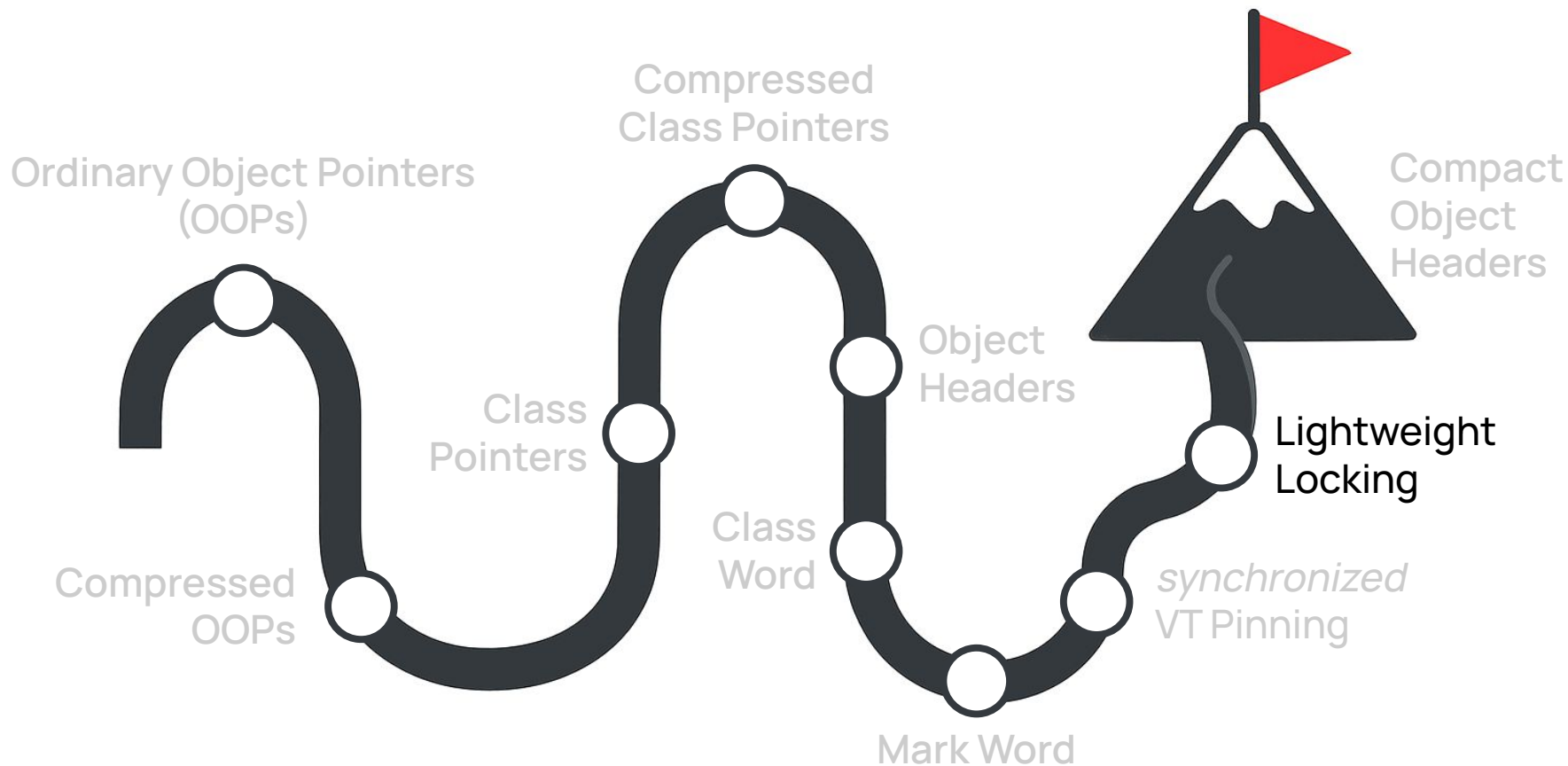
BasicLock

Thread Stack
Carrier Tread 2

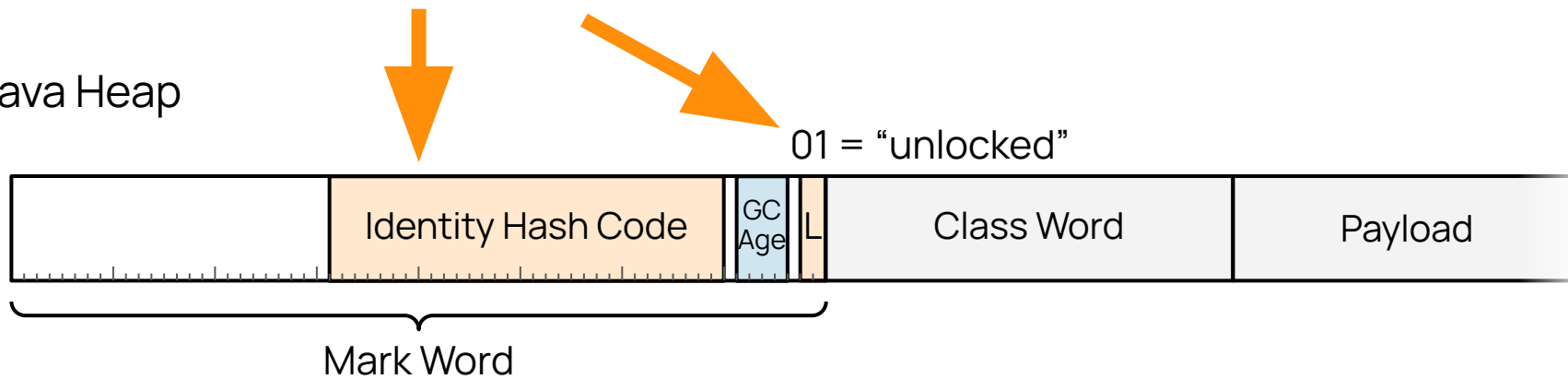
Journey into the JVM



Journey into the JVM

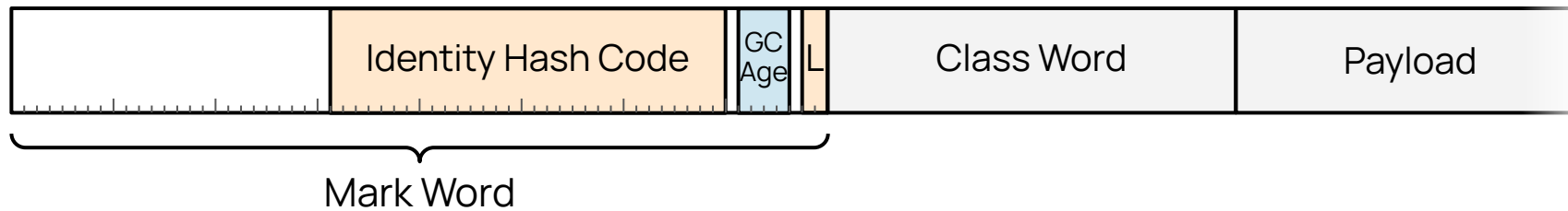


Java Heap



Thread Stack

Java Heap

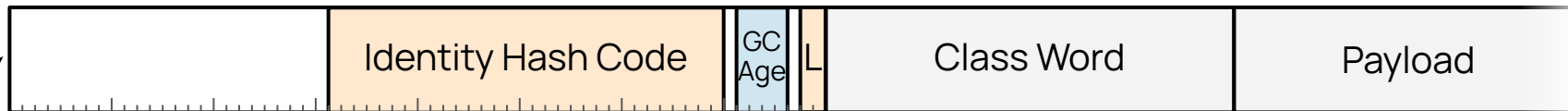


Thread Stack

Java Heap

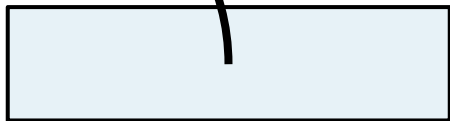


00 = "locked"

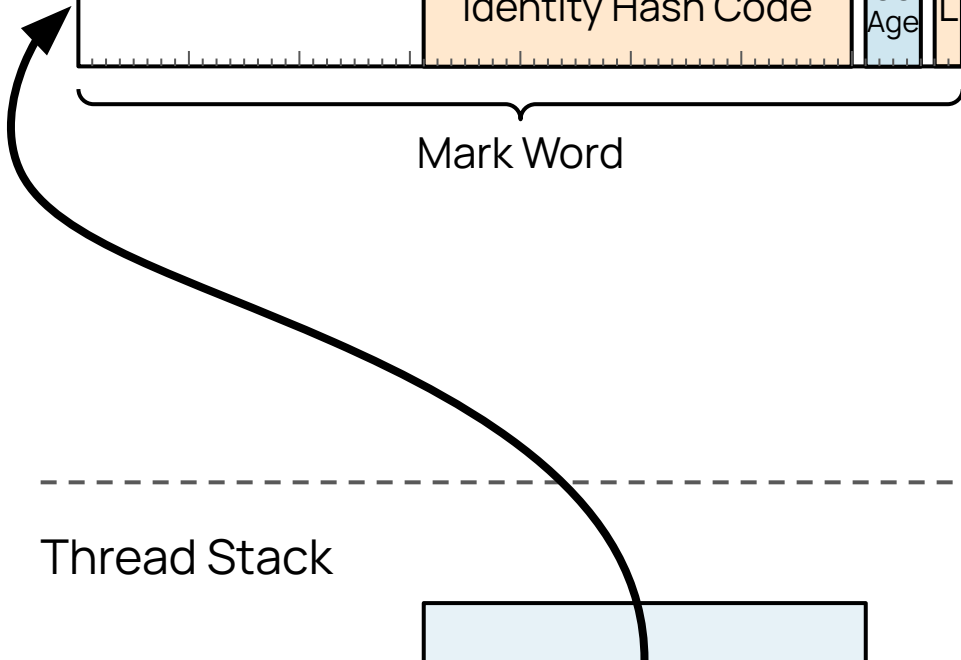


Mark Word

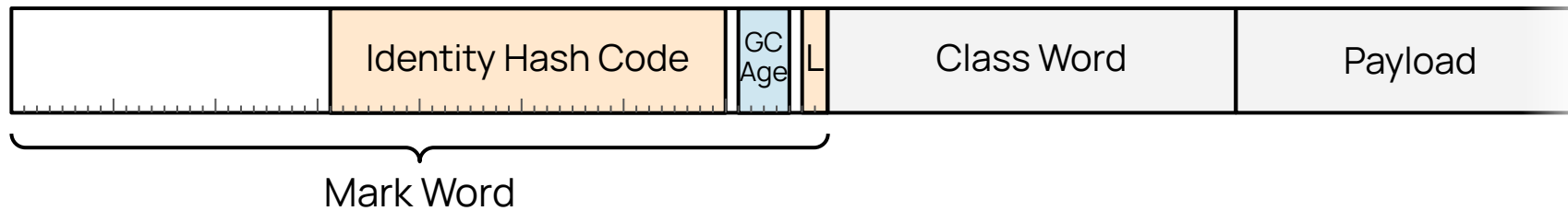
Thread Stack



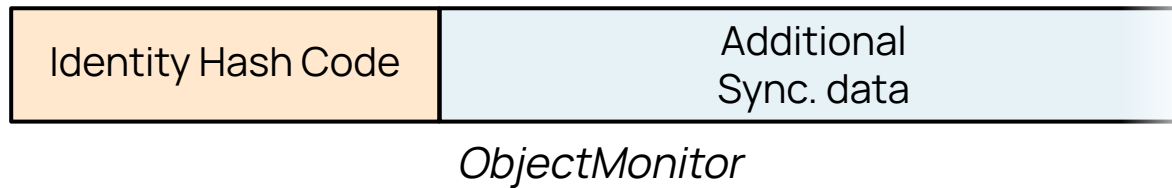
LockStack



Java Heap

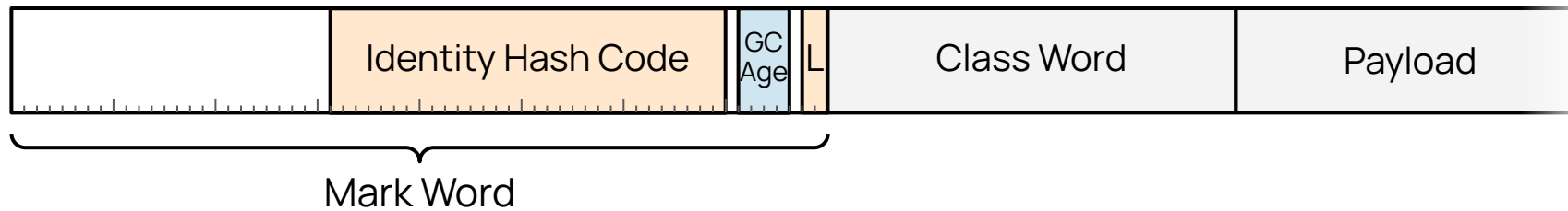


JVM
Memory

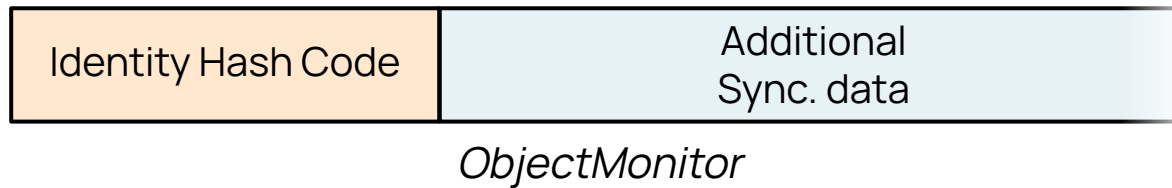


Thread Stack

Java Heap

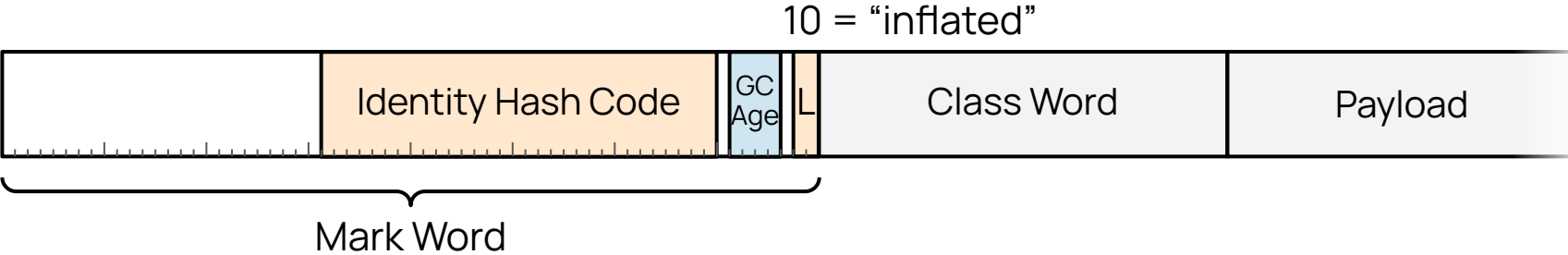


JVM
Memory



Thread Stack

Java Heap



JVM
Memory



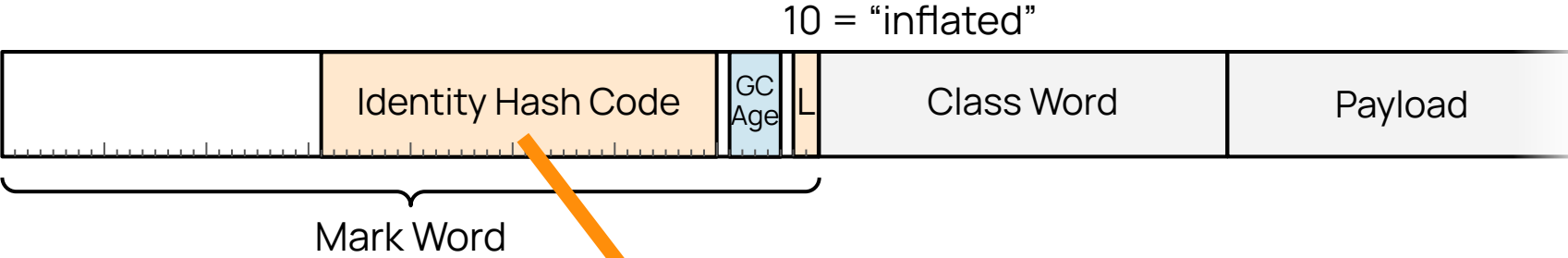
ObjectMonitor

Thread Stack

Key	Value

ObjectMonitorTable

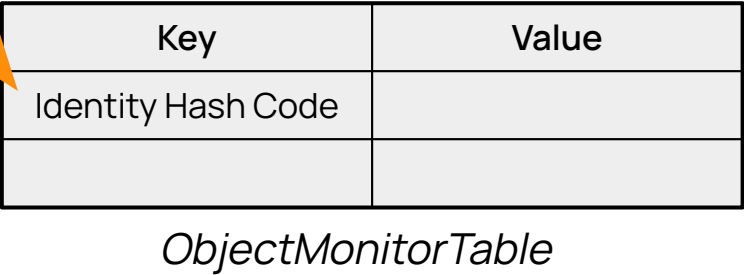
Java Heap



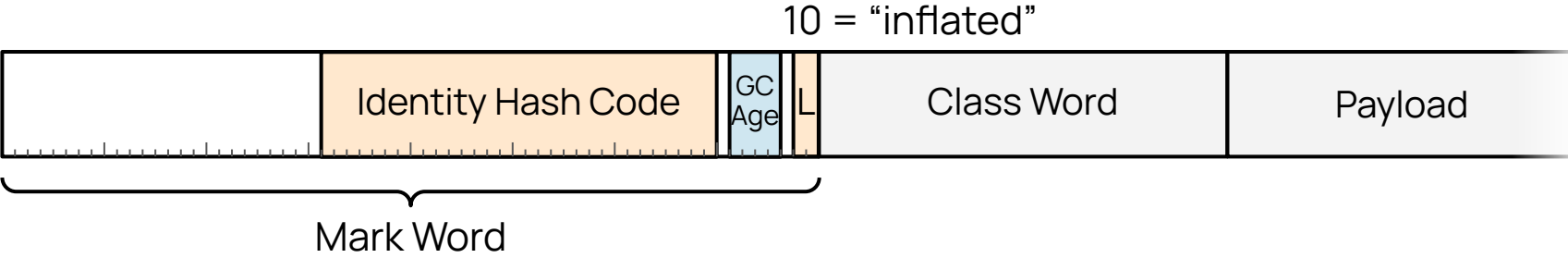
JVM Memory



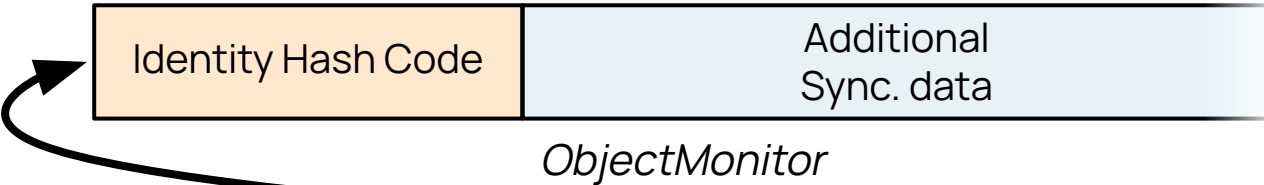
Thread Stack



Java Heap



JVM Memory



Thread Stack

Key	Value
Identity Hash Code	<i>ObjectMonitor</i> Pointer

ObjectMonitorTable

Java Heap

00 = "locked"

Identity Hash Code

GC
Age

L

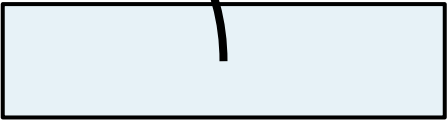
Class Word

Payload

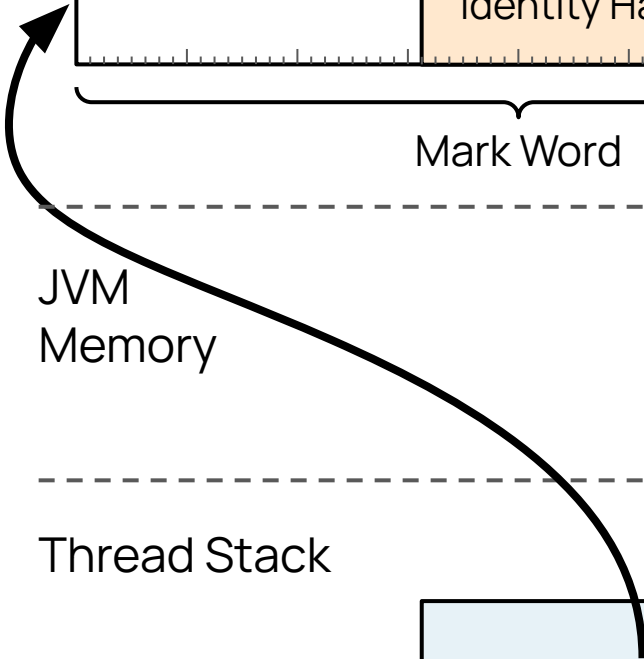
Mark Word

JVM
Memory

Thread Stack



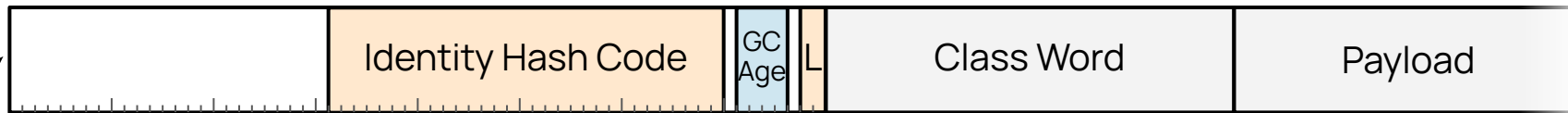
LockStack



Java Heap



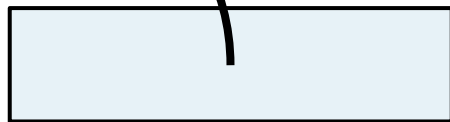
00 = "locked"



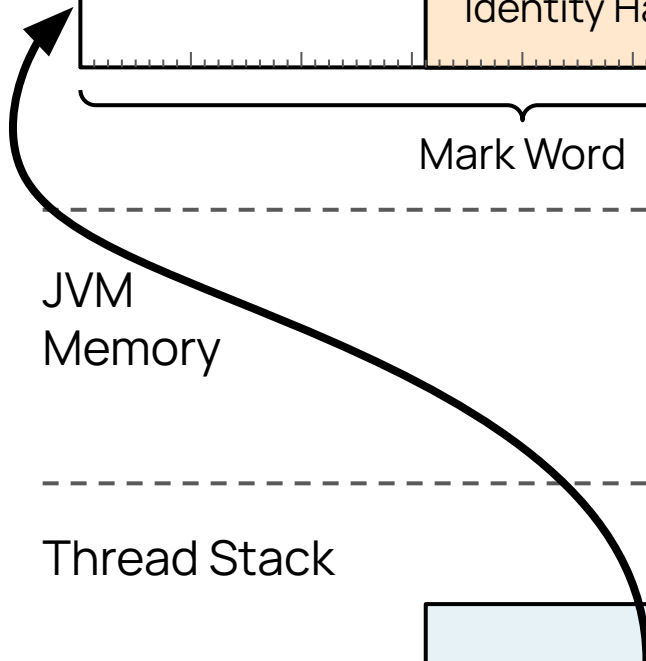
Mark Word

JVM
Memory

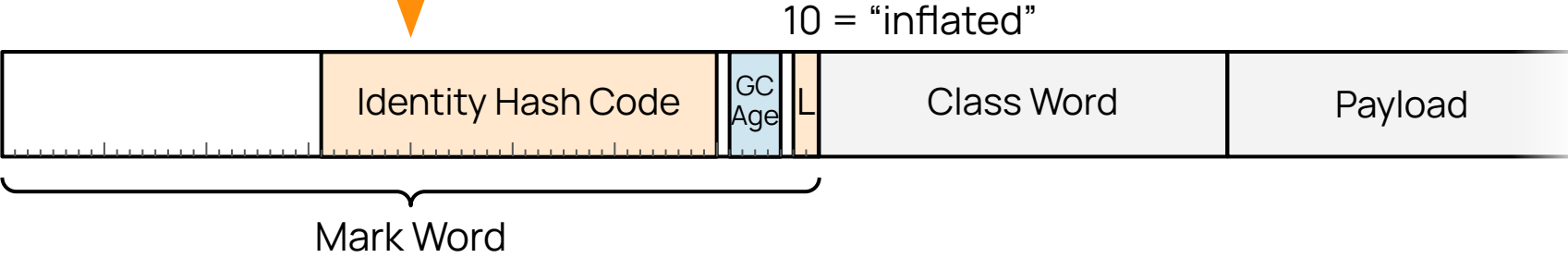
Thread Stack



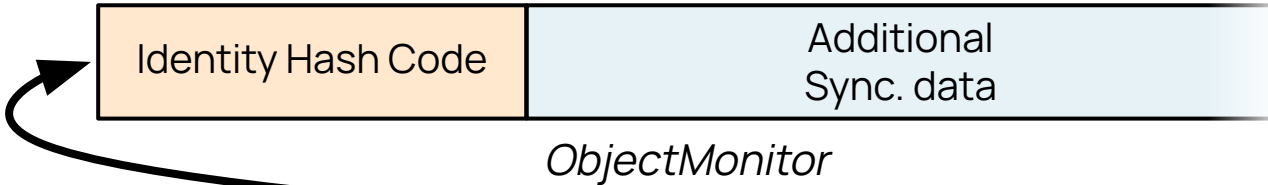
LockStack



Java Heap



JVM
Memory



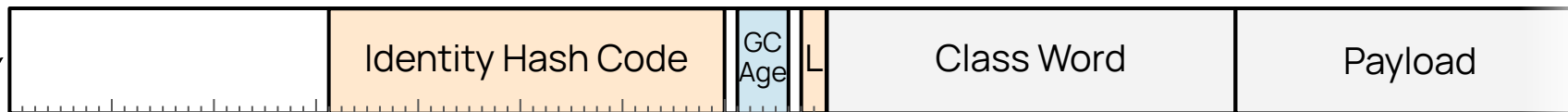
Thread Stack

Key	Value
Identity Hash Code	<i>ObjectMonitor</i> Pointer

ObjectMonitorTable

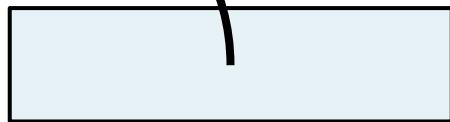
Java Heap

00 = "locked"



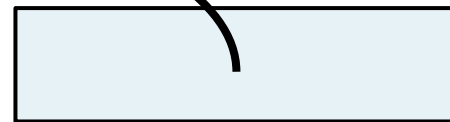
Mark Word

Thread Stack
Carrier Tread 1

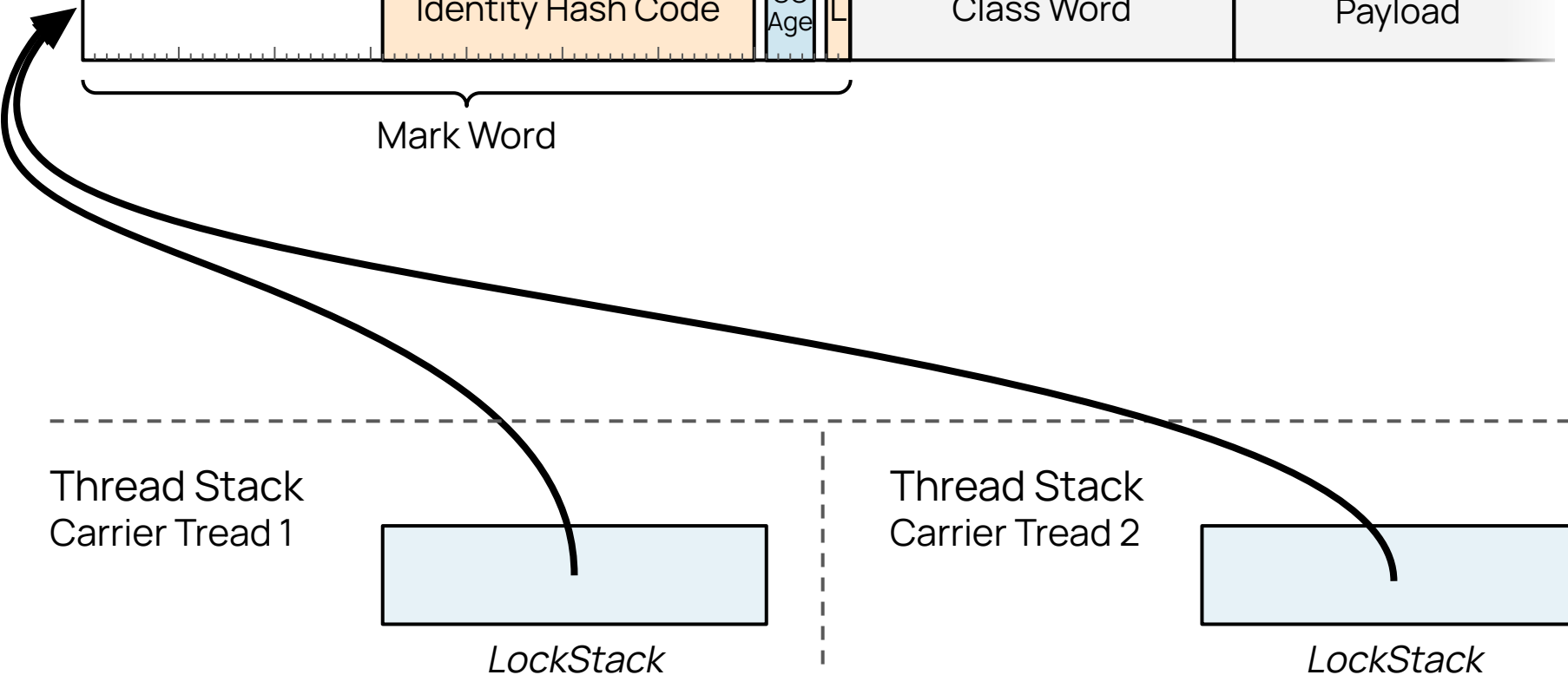


LockStack

Thread Stack
Carrier Tread 2



LockStack



Lightweight Locking

Enable Lightweight Locking in Java 21 + 22:

`-XX:LockingMode=2`

Enabled by default in Java 23.

Lightweight Locking

Switch back to Legacy Stack Locking since Java 23:

-XX:LockingMode=1

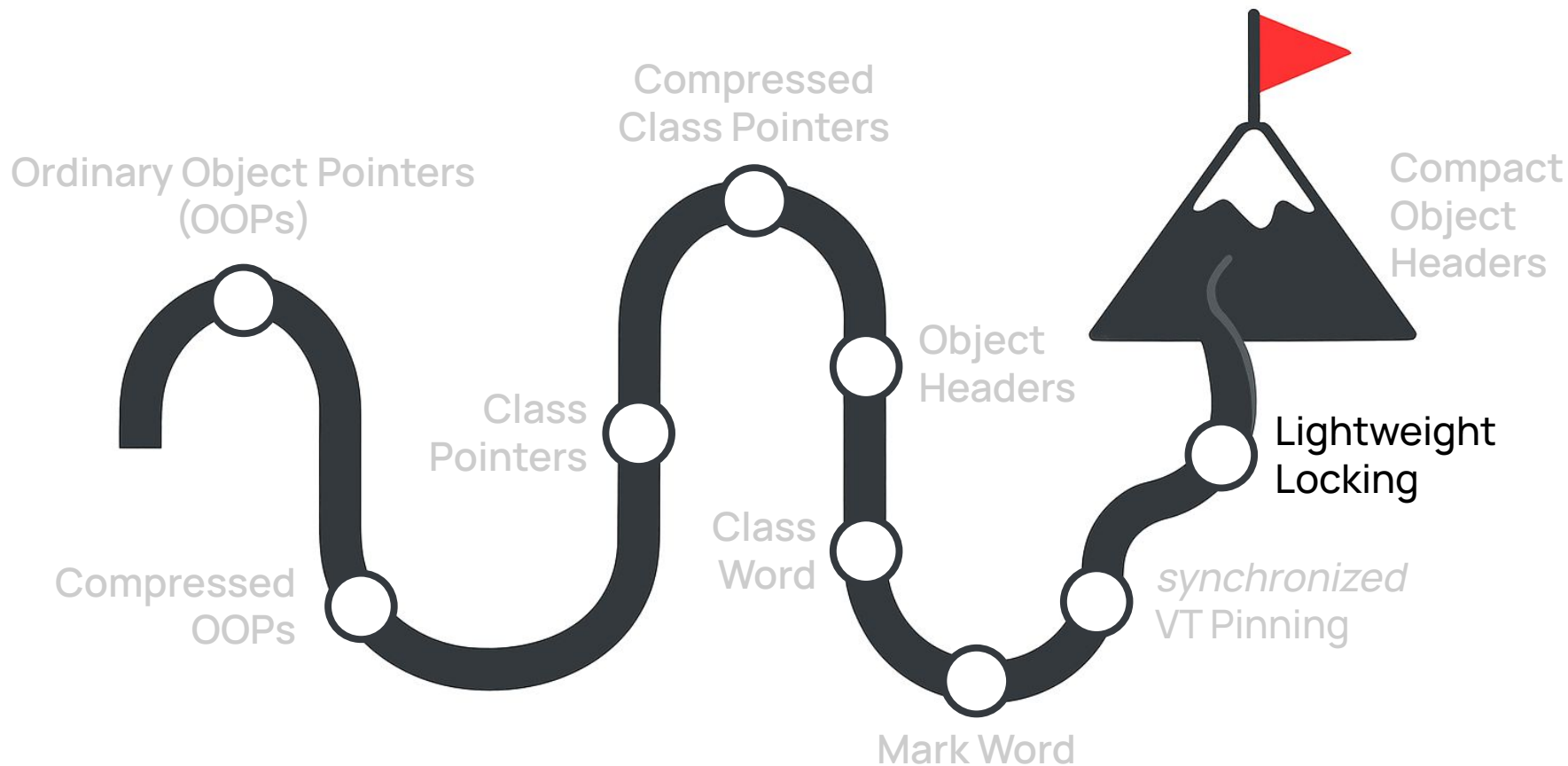
Java 24:
Deprecated

Java 26:
Deactivated

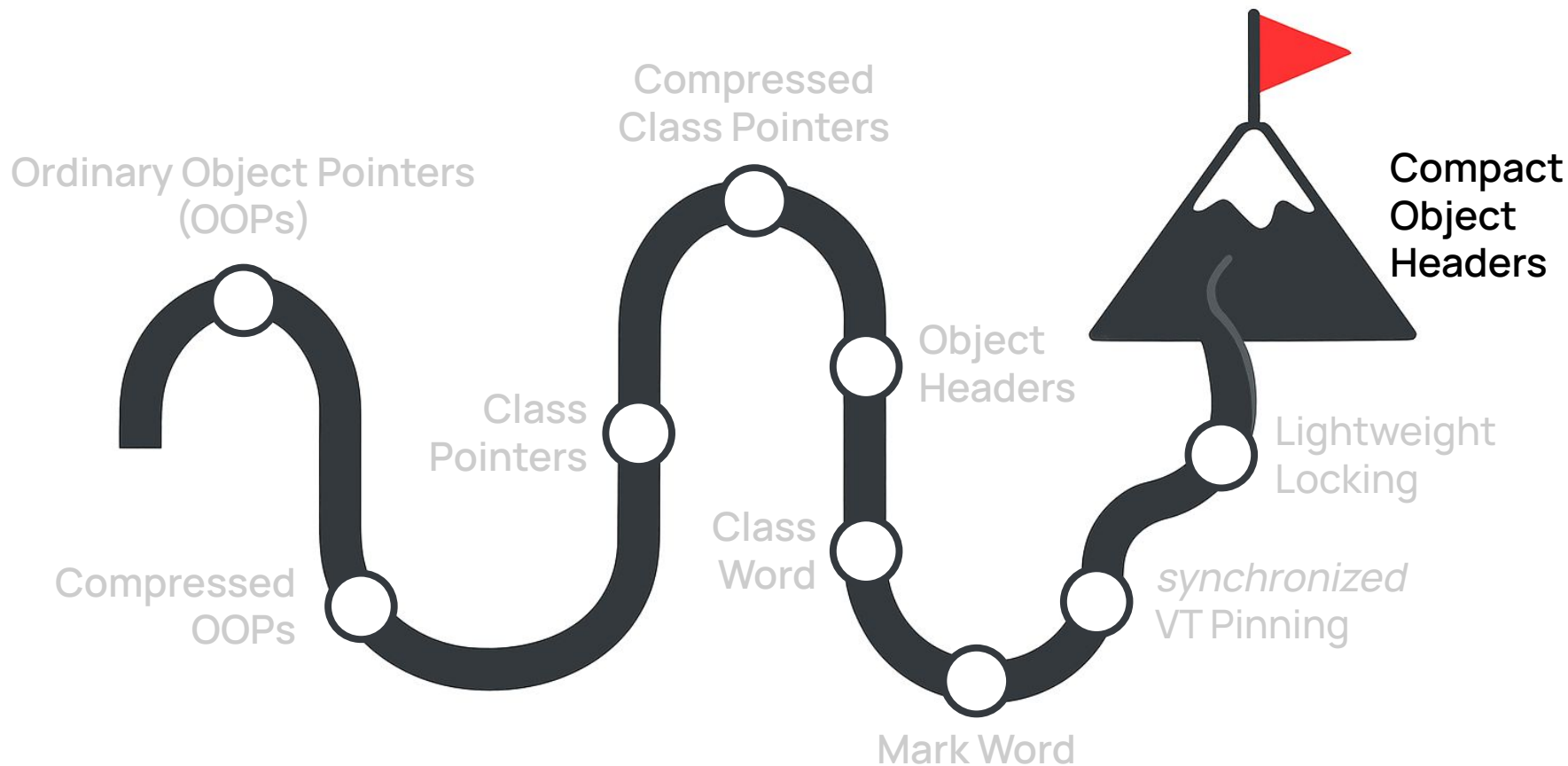
Java 27:
Removed

→ Virtual threads will be pinned again in a **synchronized** block.

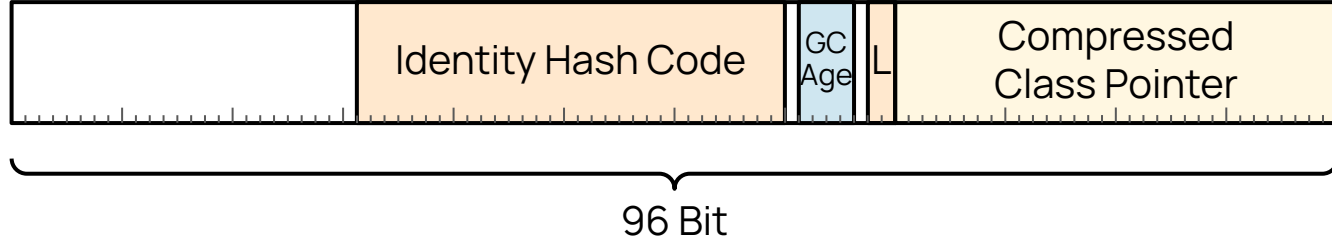
Journey into the JVM



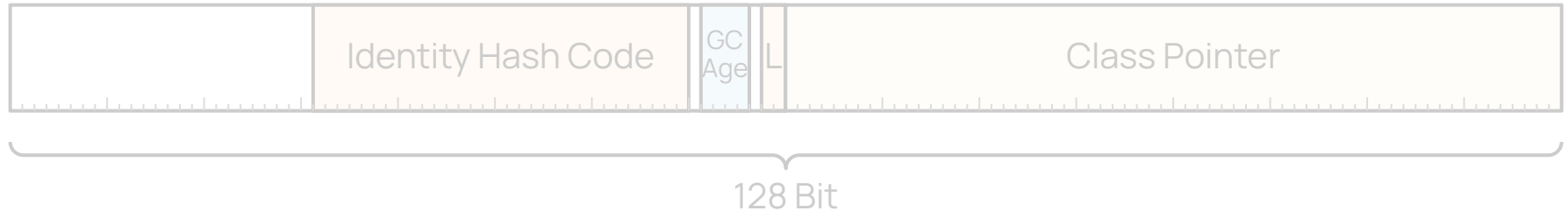
Journey into the JVM



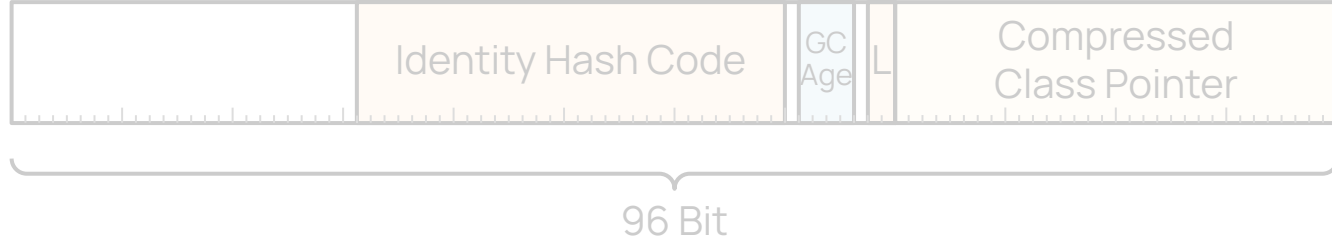
Compact Object Headers



-XX:-UseCompressedClassPointers:



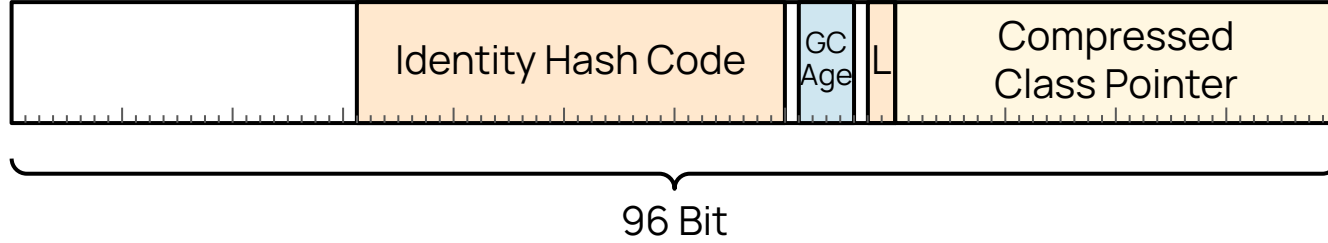
Compact Object Headers



`-XX:-UseCompressedClassPointers:`



Compact Object Headers



JEP 450: Compact Object Headers (Experimental)

<i>Owner</i>	Roman Kennke
<i>Type</i>	Feature
<i>Scope</i>	Implementation
<i>Status</i>	Closed / Delivered
<i>Release</i>	24
<i>Component</i>	hotspot / runtime
<i>Discussion</i>	hotspot dash dev at openjdk dot org
<i>Effort</i>	L
<i>Duration</i>	L
<i>Relates to</i>	JEP 519: Compact Object Headers
<i>Reviewed by</i>	Aleksey Shipilev, Erik Österlund, John Rose, Stefan Karlsson, Thomas Stuefe
<i>Endorsed by</i>	Vladimir Kozlov
<i>Created</i>	2022/10/07 19:27
<i>Updated</i>	2025/04/15 14:03
<i>Issue</i>	8294992

Summary

Reduce the size of object headers in the HotSpot JVM from between 96 and 128 bits down to 64 bits on 64-bit architectures. This will reduce heap size, improve deployment density, and increase data locality.

Goals

When enabled, this feature

- Must reduce the object header size to 64 bits (8 bytes) on the target 64-bit platforms (x64 and AArch64),
- Should reduce object sizes and memory footprint on realistic workloads,
- Should not introduce more than 5% throughput or latency overheads on the target 64-bit platforms, and only in infrequent cases, and
- Should not introduce measurable throughput or latency overheads on non-target 64-bit platforms.

JEP 519: Compact Object Headers

<i>Owner</i>	Roman Kennke
<i>Type</i>	Feature
<i>Scope</i>	Implementation
<i>Status</i>	Closed / Delivered
<i>Release</i>	25
<i>Component</i>	hotspot/runtime
<i>Discussion</i>	hotspot dash dev at openjdk dot org
<i>Effort</i>	S
<i>Duration</i>	XS
<i>Relates to</i>	JEP 450: Compact Object Headers (Experimental)
<i>Reviewed by</i>	Coleen Phillimore, Stefan Karlsson, Vladimir Kozlov
<i>Endorsed by</i>	Vladimir Kozlov
<i>Created</i>	2025/04/15 14:01
<i>Updated</i>	2025/08/24 18:03
<i>Issue</i>	8354672

Summary

Change compact object headers from an experimental feature to a product feature.

Non-Goals

It is not a goal to make compact object headers be the default object-header layout.

Motivation

Compact object headers were introduced as an alternative object-header layout by [JEP 450](#) in JDK 24. Features of this size are best introduced carefully and gradually, so we introduced it as an experimental feature.

Since JDK 24, compact object headers have proven their stability and performance.

They have been tested at Oracle by running the full JDK test suite. They have also been tested at Amazon by hundreds of services in production, most of them using backports of the feature to JDK 21 and JDK 17.

JEP 534: Compact Object Headers by Default

<i>Owner</i>	Roman Kennke
<i>Type</i>	Feature
<i>Scope</i>	JDK
<i>Status</i>	Integrated
<i>Release</i>	27
<i>Component</i>	hotspot/runtime
<i>Discussion</i>	hotspot dash runtime dash dev at openjdk dot org
<i>Effort</i>	XS
<i>Duration</i>	XS
<i>Relates to</i>	JEP 519: Compact Object Headers
<i>Reviewed by</i>	Vladimir Kozlov
<i>Endorsed by</i>	Vladimir Kozlov
<i>Created</i>	2025/07/01 13:36
<i>Updated</i>	2026/05/28 12:07
<i>Issue</i>	8361187

Summary

Make compact object headers the default object header layout in the HotSpot JVM. Compact object headers reduce object headers from 96 bits down to 64 bits on 64-bit architectures, thereby reducing heap size, improving deployment density, and increasing data locality.

Non-Goals

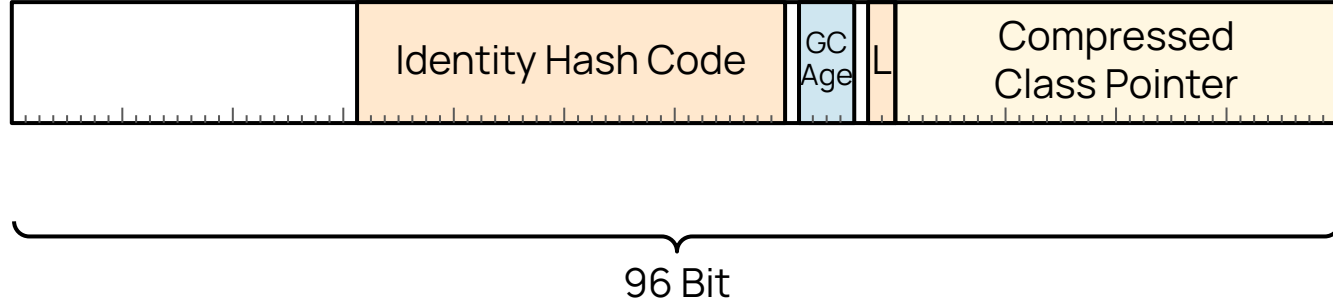
- It is not a goal to remove the old 96-bit object header layout at this time.

Motivation

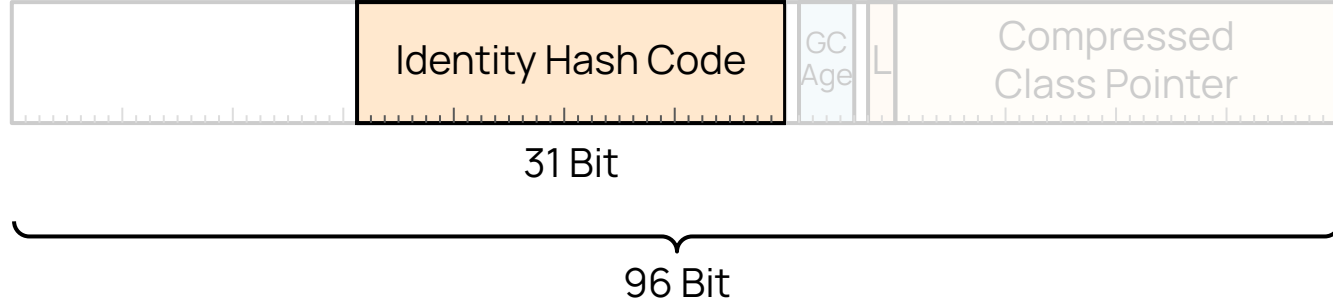
Compact object headers were introduced as an alternative object-header layout via [JEP 450](#) (JDK 24), and turned into a full product feature via [JEP 519](#) (JDK 25). Features of this size are best introduced carefully and gradually, so we have not turned it on by default, yet.

Since JDK 24, compact object headers have proven their stability and performance.

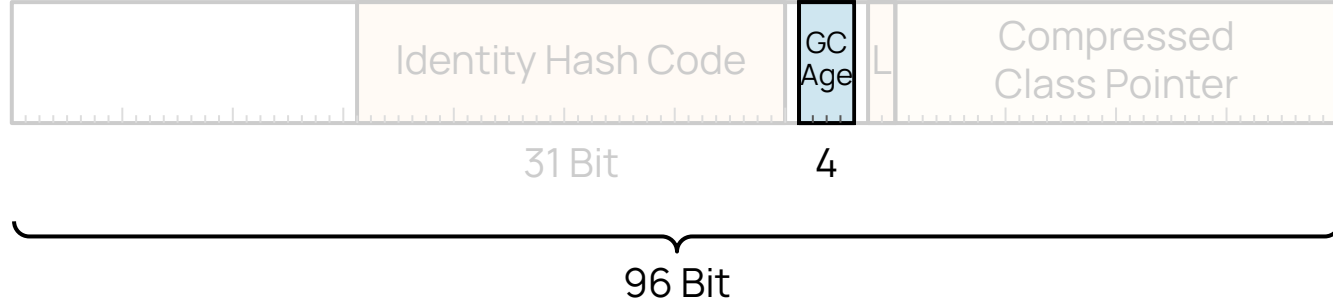
Compact Object Headers



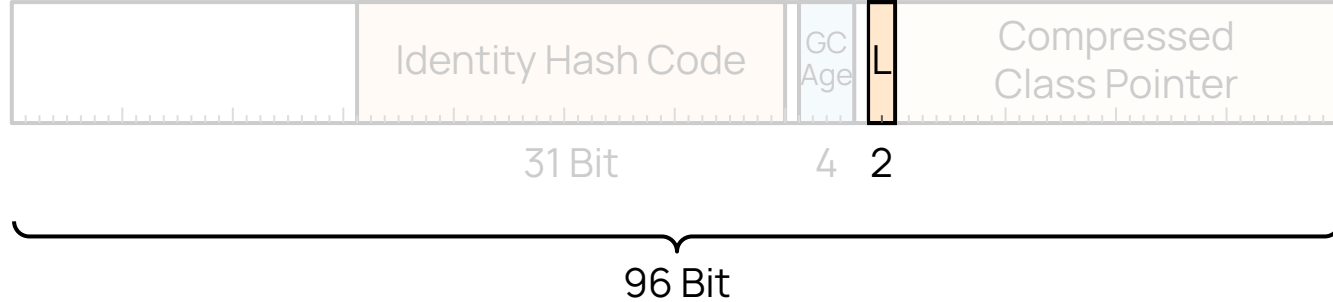
Compact Object Headers



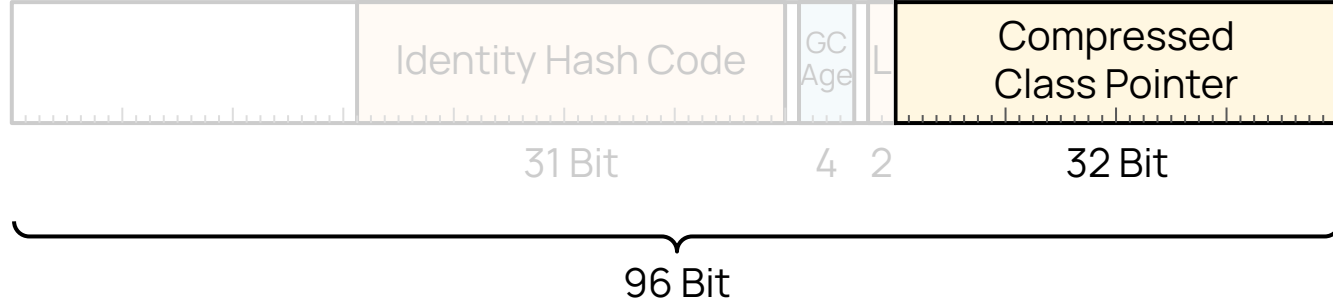
Compact Object Headers



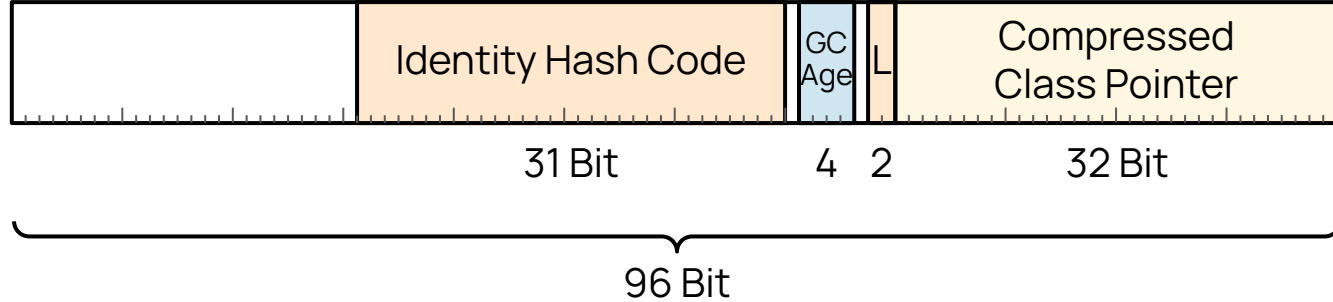
Compact Object Headers



Compact Object Headers

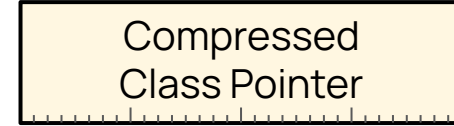


Compact Object Headers



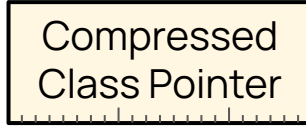
$$\begin{array}{r} 31 \text{ Bit} \\ + 4 \text{ Bit} \\ + 2 \text{ Bit} \\ + 32 \text{ Bit} \\ \hline = 69 \text{ Bit} \end{array}$$

Compact Object Headers



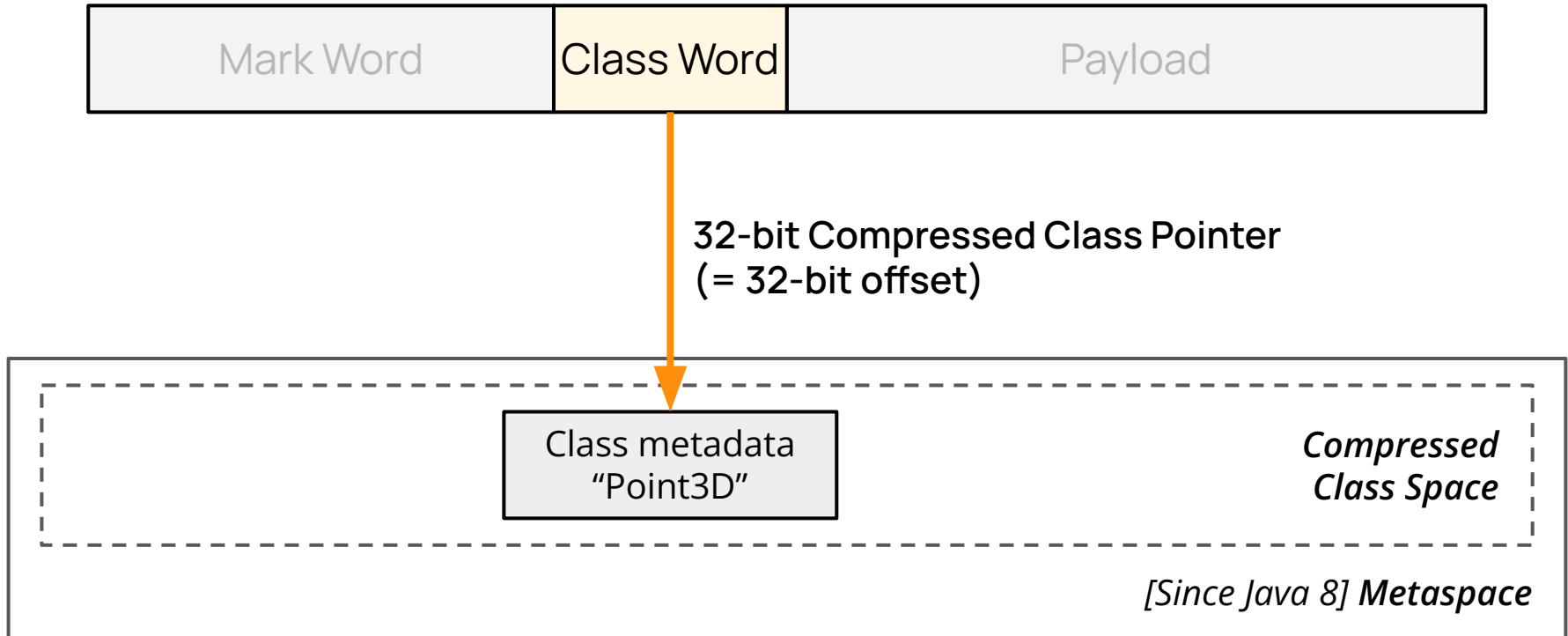
32 Bit

Compact Object Headers

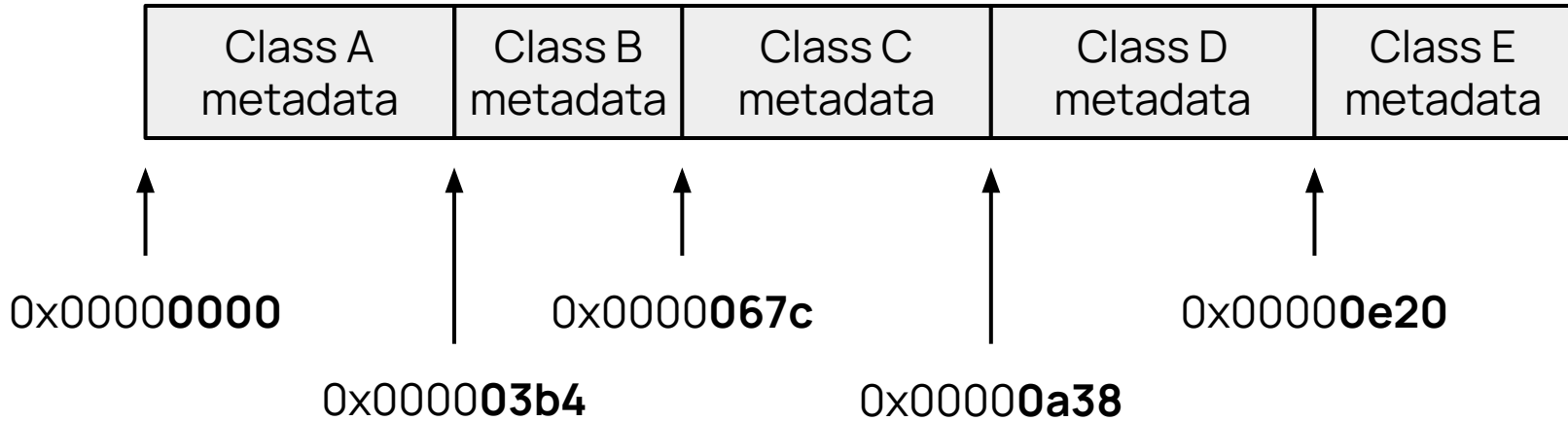


22 Bit

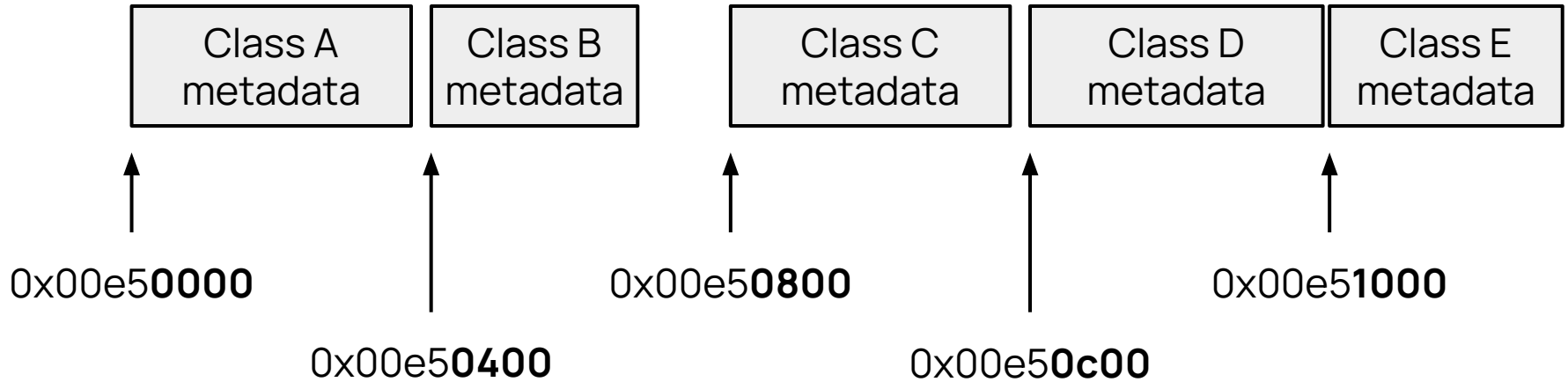
Compact Object Headers



Compact Object Headers



Compact Object Headers



Compact Object Headers

0x00e5**0000**

0b0000000011100101**00000000000000000000**

0x00e5**0400**

0b0000000011100101**00000100000000000000**

0x00e5**0800**

0b0000000011100101**00001000000000000000**

0x00e5**0c00**

0b0000000011100101**00001100000000000000**

0x00e5**1000**

0b0000000011100101**00010000000000000000**

Trailing 10 bits

Compact Object Headers

0x00e5**0000**

0x00e5**0400**

0x00e5**0800**

0x00e5**0c00**

0x00e5**1000**

0b00000000011100101**00000000000000000000**

0b00000000011100101**00000100000000000000**

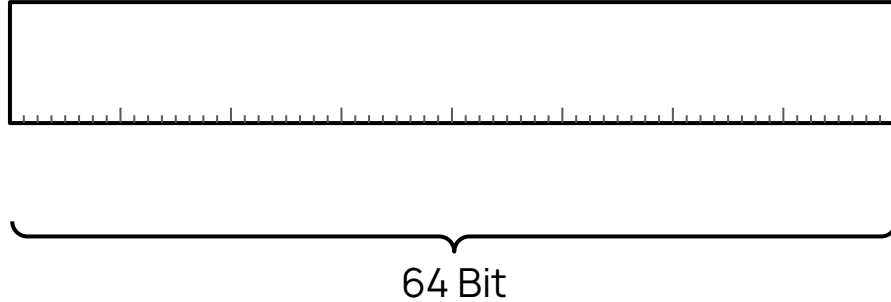
0b00000000011100101**00001000000000000000**

0b00000000011100101**00001100000000000000**

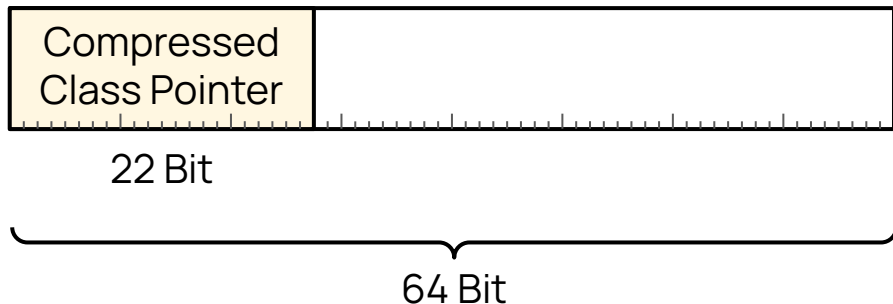
0b00000000011100101**00010000000000000000**

Leading 22 bits

Compact Object Headers



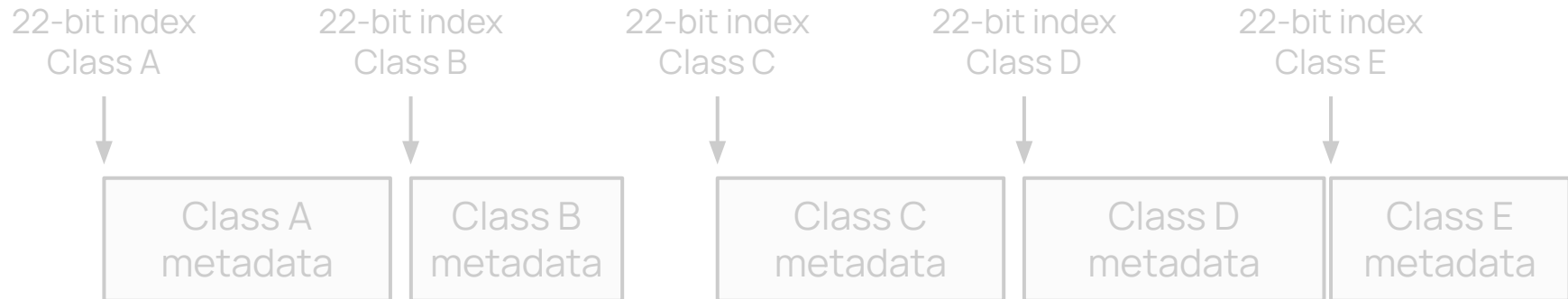
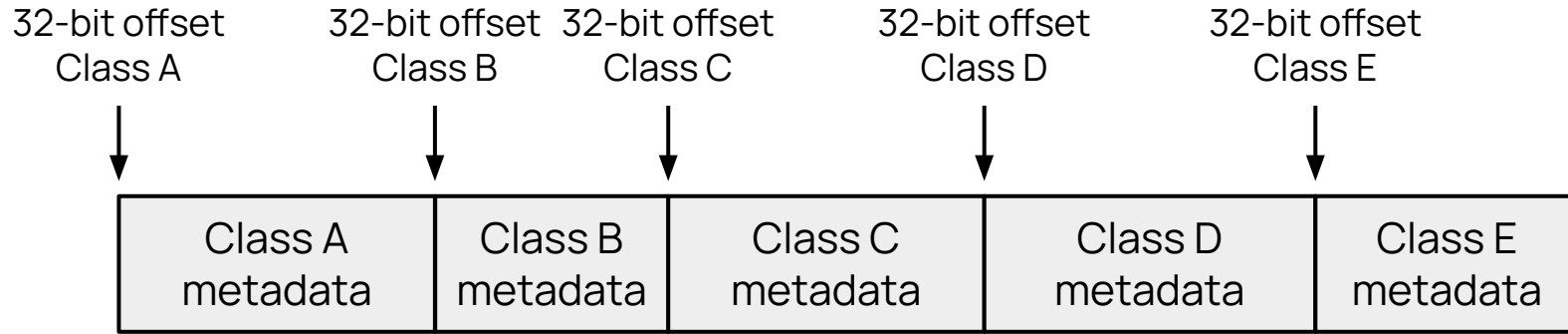
Compact Object Headers



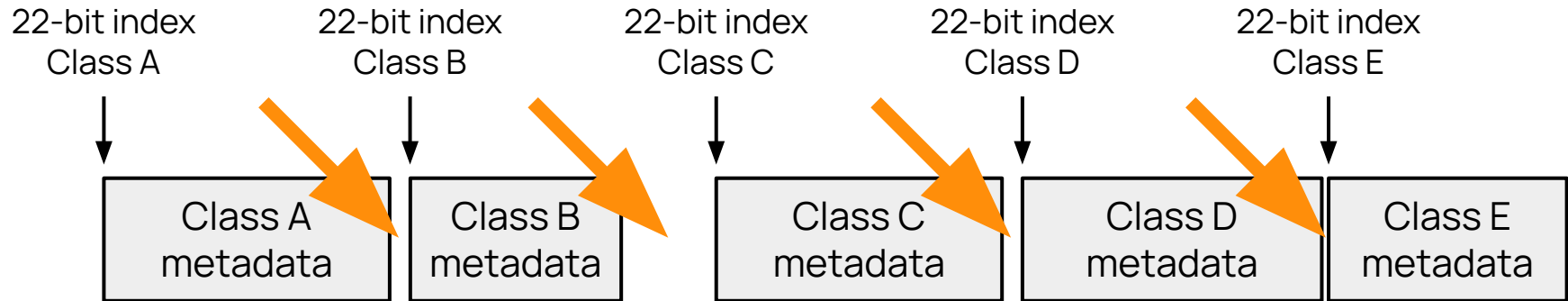
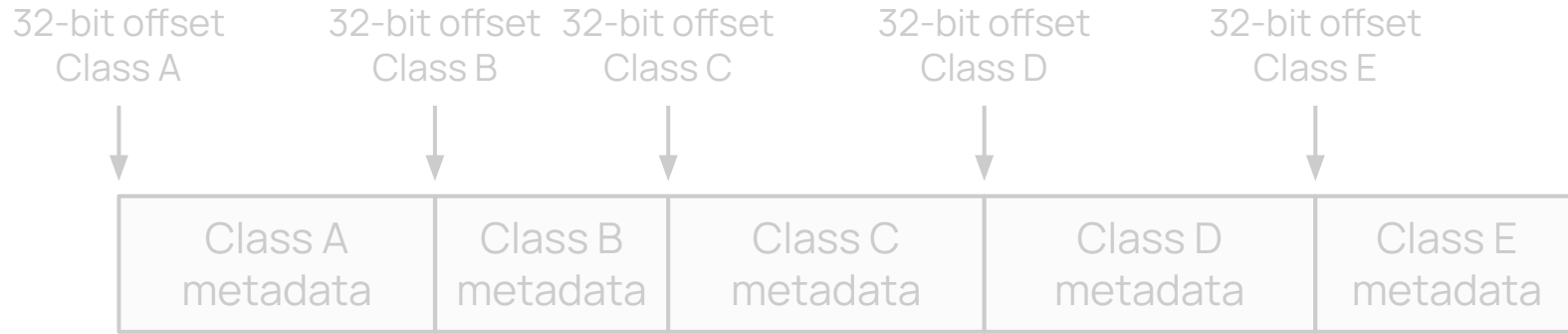
Addressable with 22 bit: $2^{22} = 4,194,304$ classes

Addressable with 32 bit: 2^{32} bytes / (768 bytes / class)
 $\approx 5,600,000$ classes

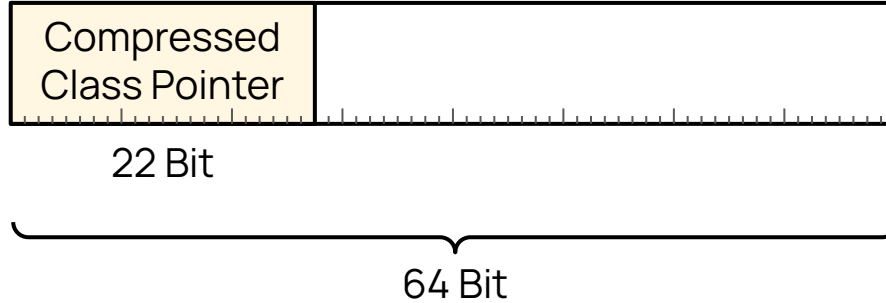
Compact Object Headers



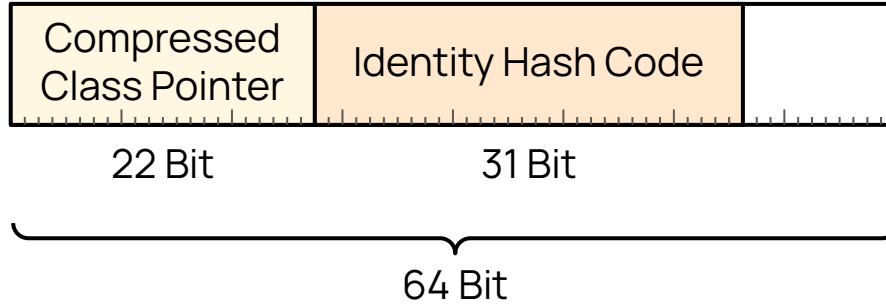
Compact Object Headers



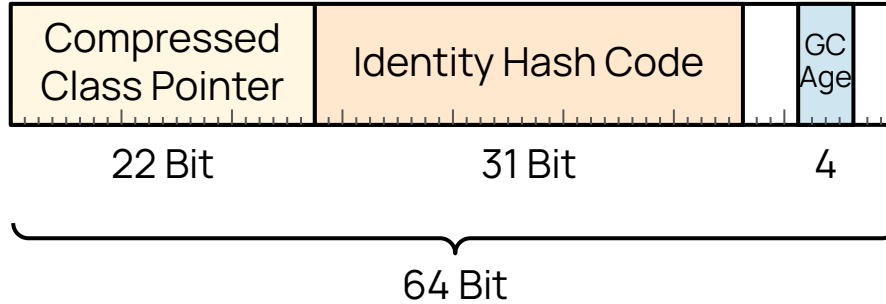
Compact Object Headers



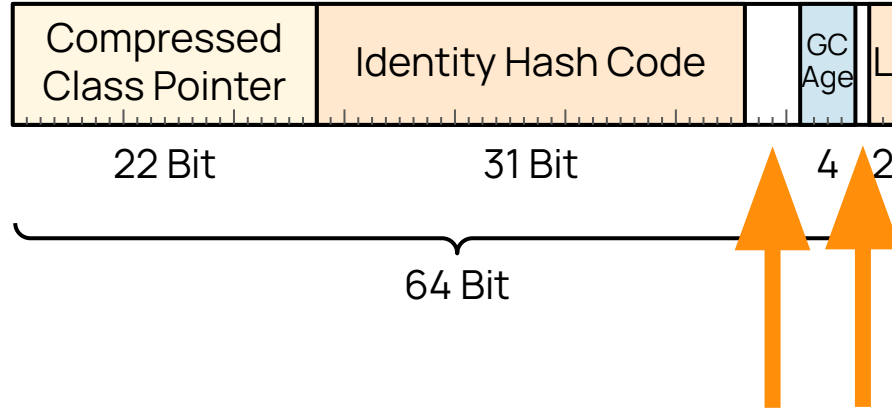
Compact Object Headers



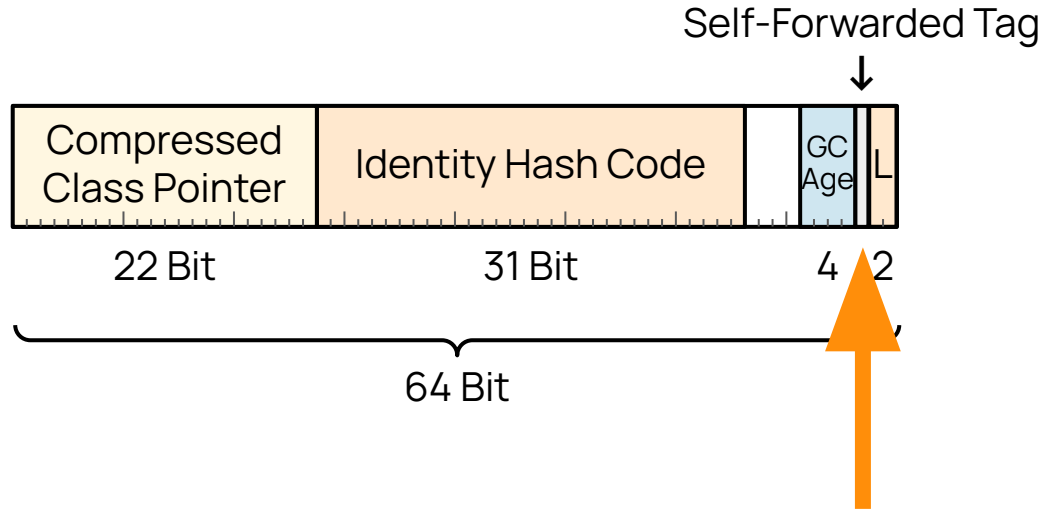
Compact Object Headers



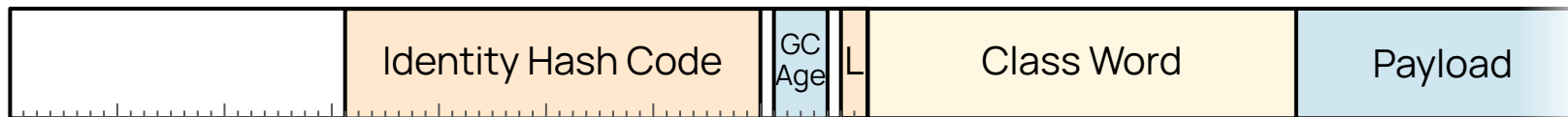
Compact Object Headers



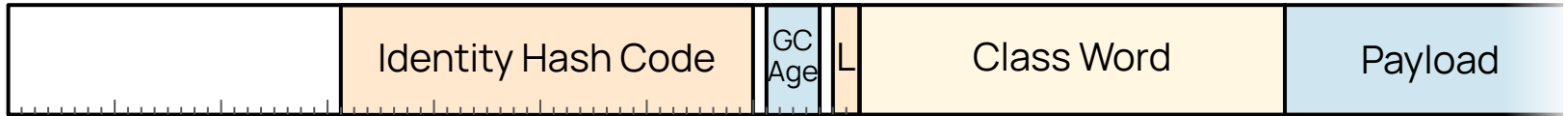
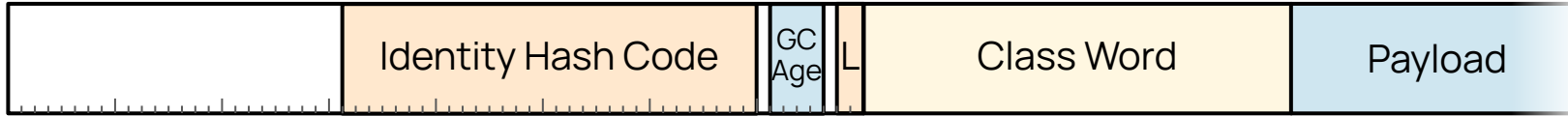
Compact Object Headers



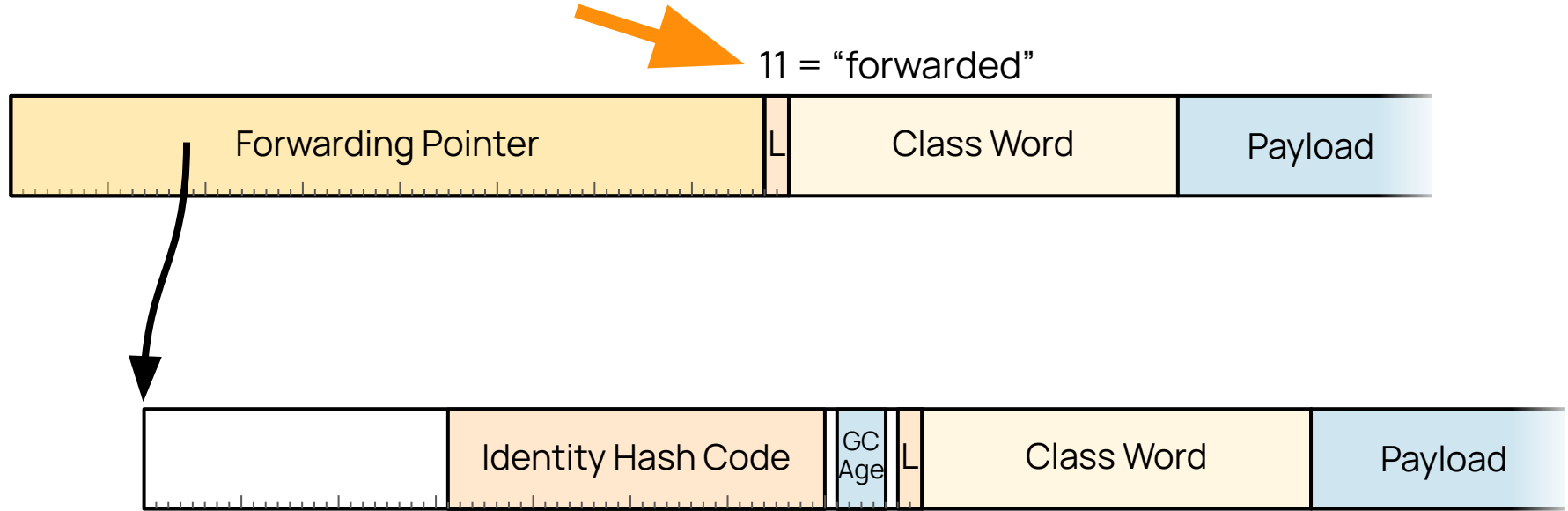
Compact Object Headers



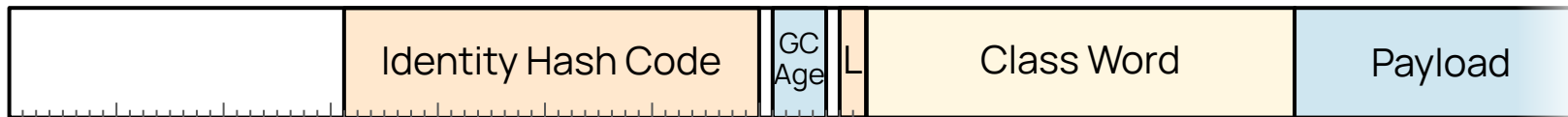
Compact Object Headers



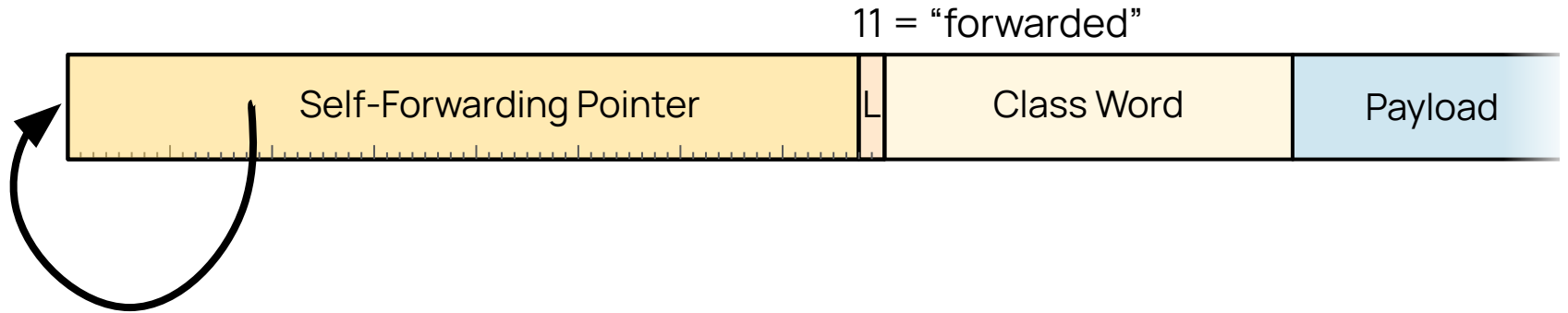
Compact Object Headers



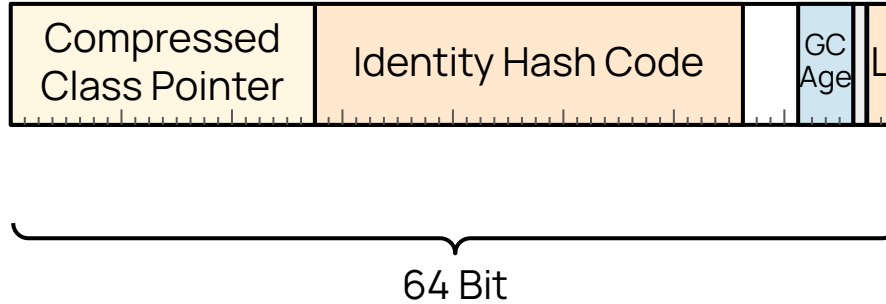
Compact Object Headers



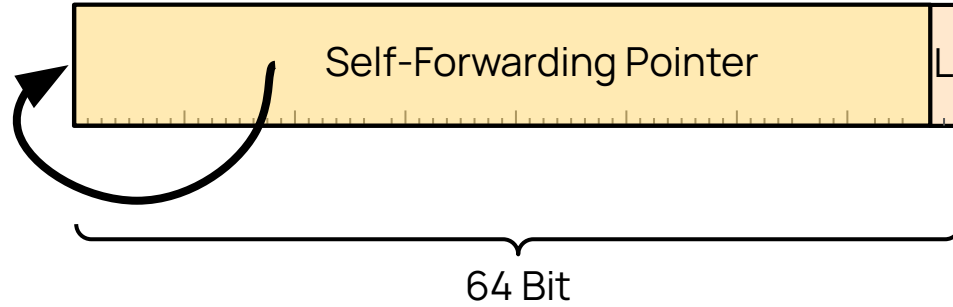
Compact Object Headers



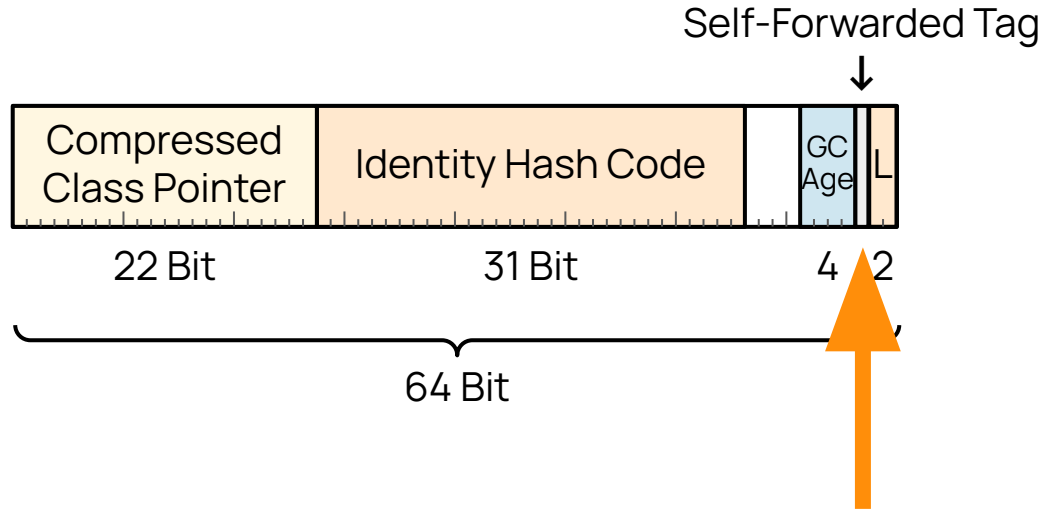
Compact Object Headers



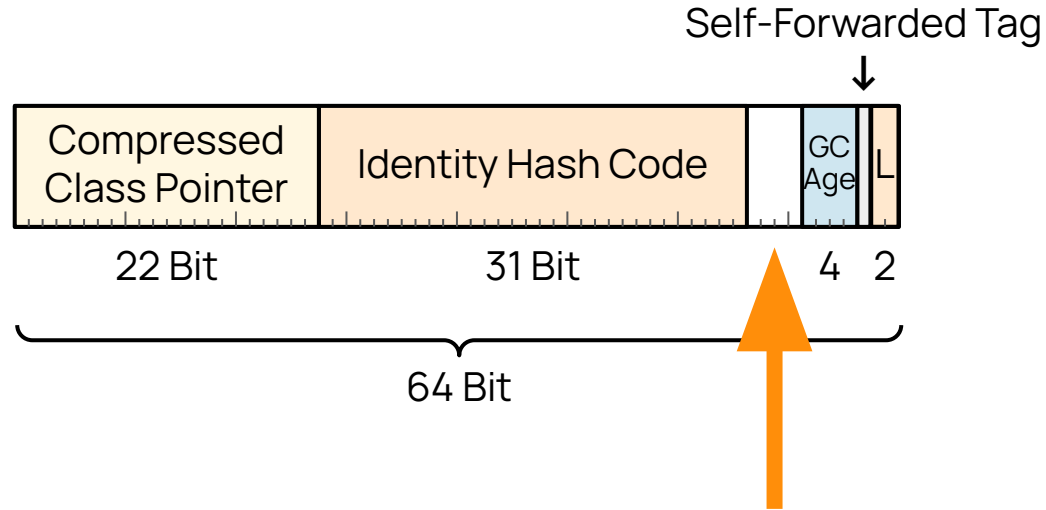
Compact Object Headers



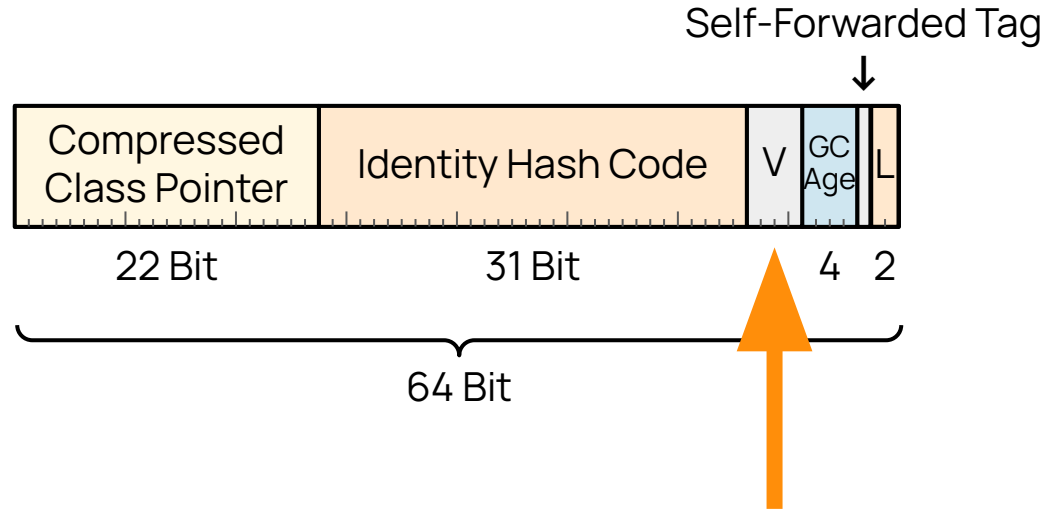
Compact Object Headers



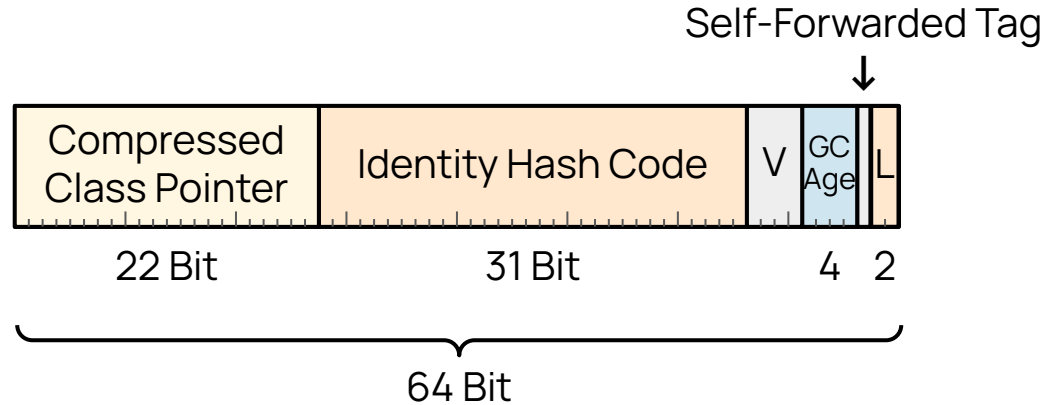
Compact Object Headers



Compact Object Headers



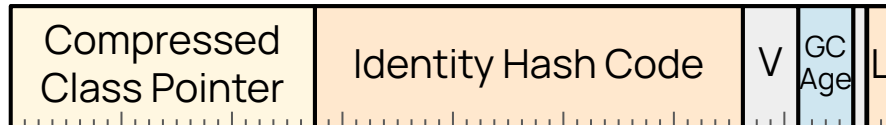
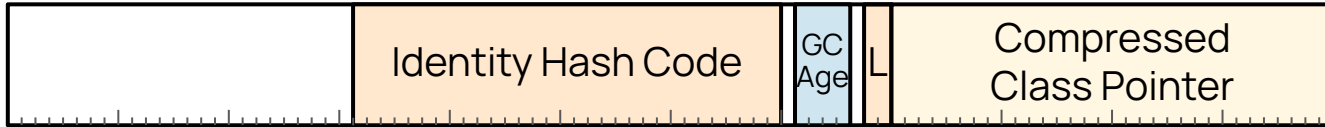
Compact Object Headers



Java 25: `-XX:+UseCompactObjectHeaders`

Java 27: enabled by default

Compact Object Headers



“Was it worth it?”

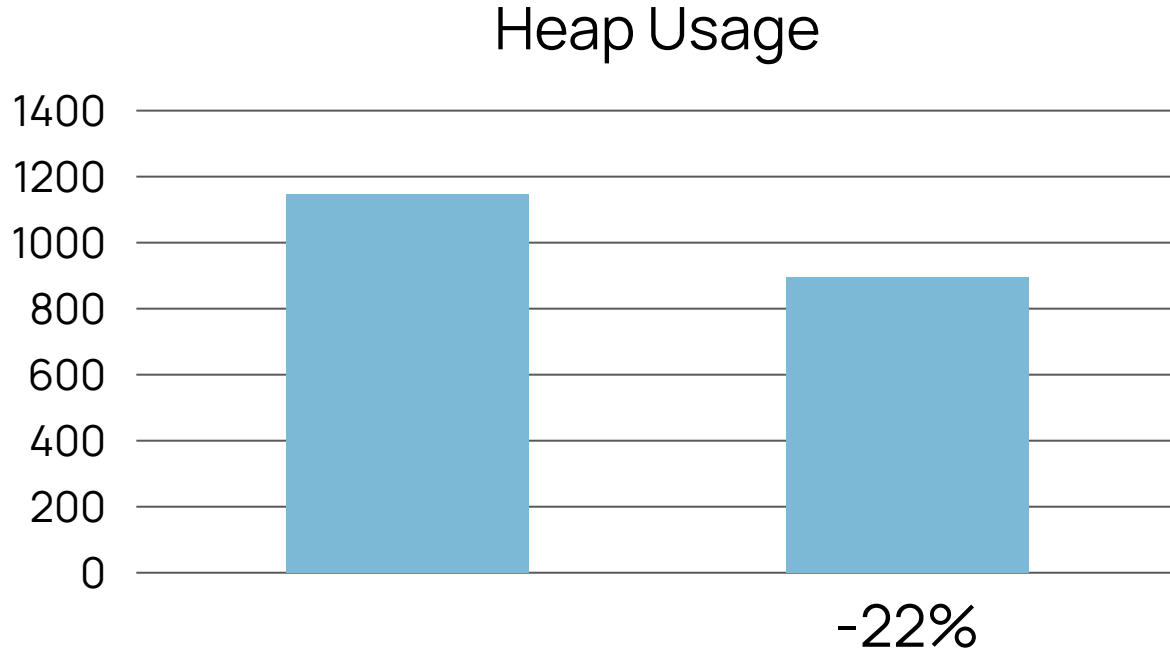
Compact Object Headers

“Early adopters of Project Lilliput who have tried it with real-world applications confirm that **live data is typically reduced by 10%–20%.**”

Source: <https://openjdk.org/jeps/450>



Compact Object Headers



Source: <https://github.com/rkennke/talks/blob/master/Lilliput-FOSDEM-2025.pdf>



Compact Object Headers

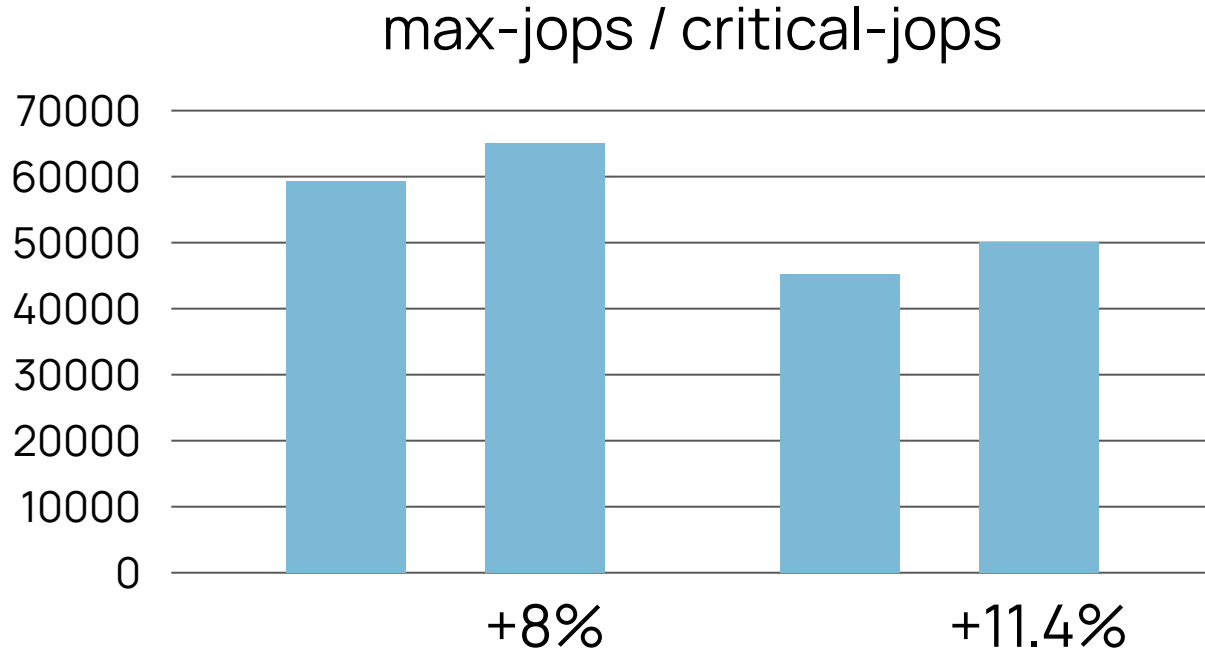
“Java object memory usage
is **reduced by about 5-10%.**”

Source:

https://www.alibabacloud.com/blog/compact-object-headers-in-dragonwell-jdk-reducing-costs-and-increasing-efficiency-for-java-applications_601065



Compact Object Headers



Source: <https://github.com/rkennke/talks/blob/master/Lilliput-FOSDEM-2025.pdf>



Compact Object Headers

“The exact gain varies depending on the workload (especially the number of small objects created),
but in many cases, it was about 10% faster!”

Source:

https://www.reddit.com/r/scala/comments/1jptiv3/xxusecompactobjectheaders_is_your_new_turbo/



plokhotnyuk
u/plokhotnyuk

Compact Object Headers

“Java’s newly allocated object memory usage
(allocation rate) and
GC frequency are reduced by about 5-10%.”

“Max(absolute throughput)
and critical (low latency required throughput)
are improved by 6.17% and 9.01% respectively.”

Source:

https://www.alibabacloud.com/blog/compact-object-headers-in-dragonwell-jdk-reducing-costs-and-increasing-efficiency-for-java-applications_601065



Compact Object Headers

To enable in Java 25: `-XX:+UseCompactObjectHeaders`

As of Java 27: Enabled by default

Limitations: ~~`-XX:-UseCompressedClassPointers`~~

~~`-XX:LockingMode=1` (Legacy Stack Locking)~~

Max. 8 TB Heap (ZGC: no limit)

Outlook: “Lilliput 2”

→ 32-bit headers

[About](#)
[Adoption](#)
[Amber](#)
[Build](#)
[Client Libraries](#)
[Code Tools](#)
[Coin](#)
[Compatibility & Specification Review](#)
[Compiler](#)
[CRaC](#)
[Device I/O](#)
[Duke](#)
[Graal](#)
[HotSpot](#)
[IDE Tooling & Support](#)
[JDK 8](#)
[JDK 8u](#)
[JDK Updates](#)
[Kulla](#)
[Lepi](#)

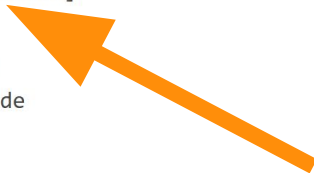
Lilliput 2 - 4-byte headers

Created by Roman Kennke on Apr 18, 2024

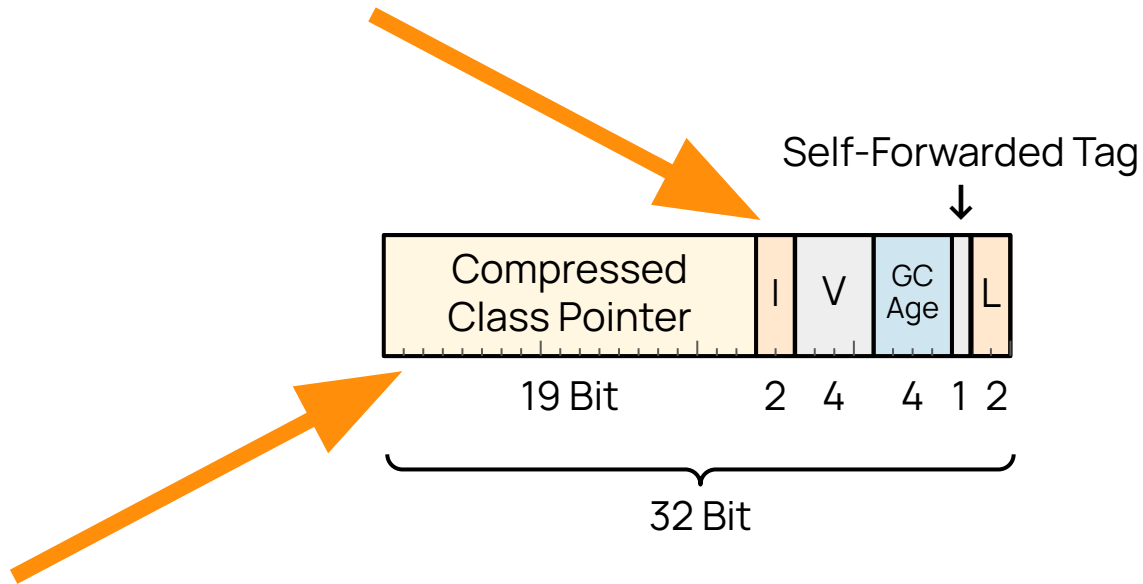
Header layout

32 13 6 2 0
[CCCCCCCCCCCCCCCCCHVVVAAASLL]

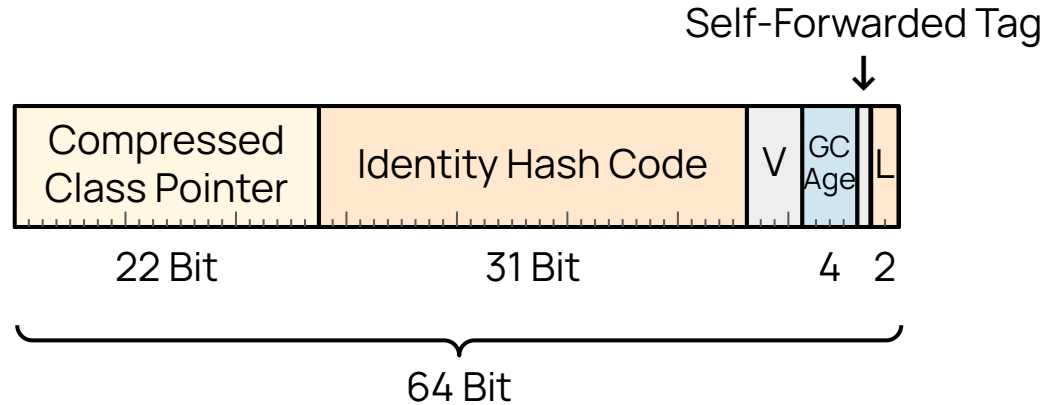
C - Compressed Class-Pointer
H - Compact Identity Hash-Code
V - Valhalla Bits
A - GC Age Bits
S - Self-Forwarding Bits
L - Lock-Bits



“Lilliput 2”

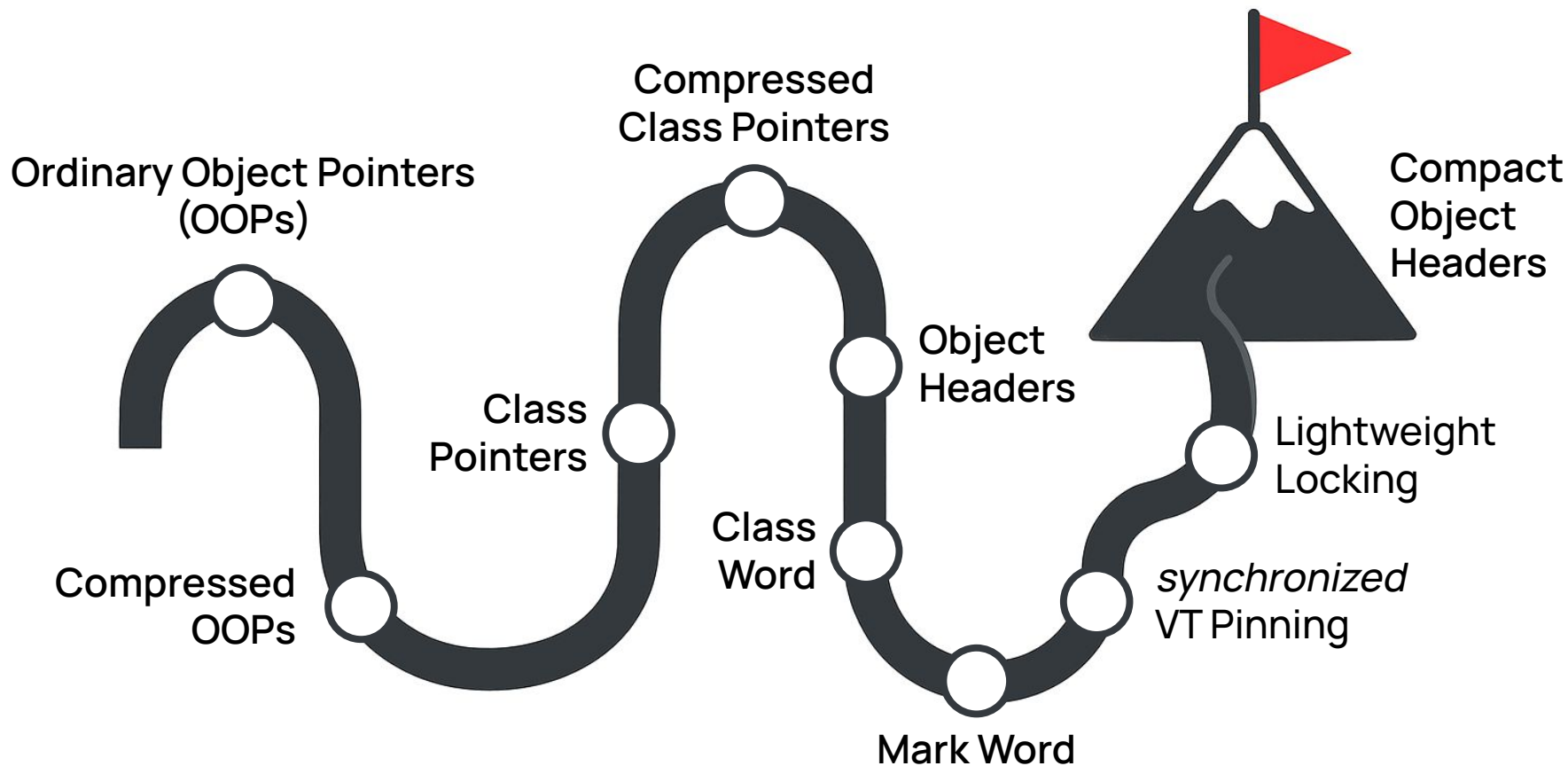


Compact Object Headers



-XX:+UseCompactObjectHeaders

Journey into the JVM



Links +
Slides



LinkedIn



Sven Woltmann

Java Author, Speaker, Trainer



sven@happycoders.eu



<https://www.happycoders.eu/>



<https://linkedin.com/in/sven-woltmann/>



<https://youtube.com/c/happycoders>

