ORACLE®

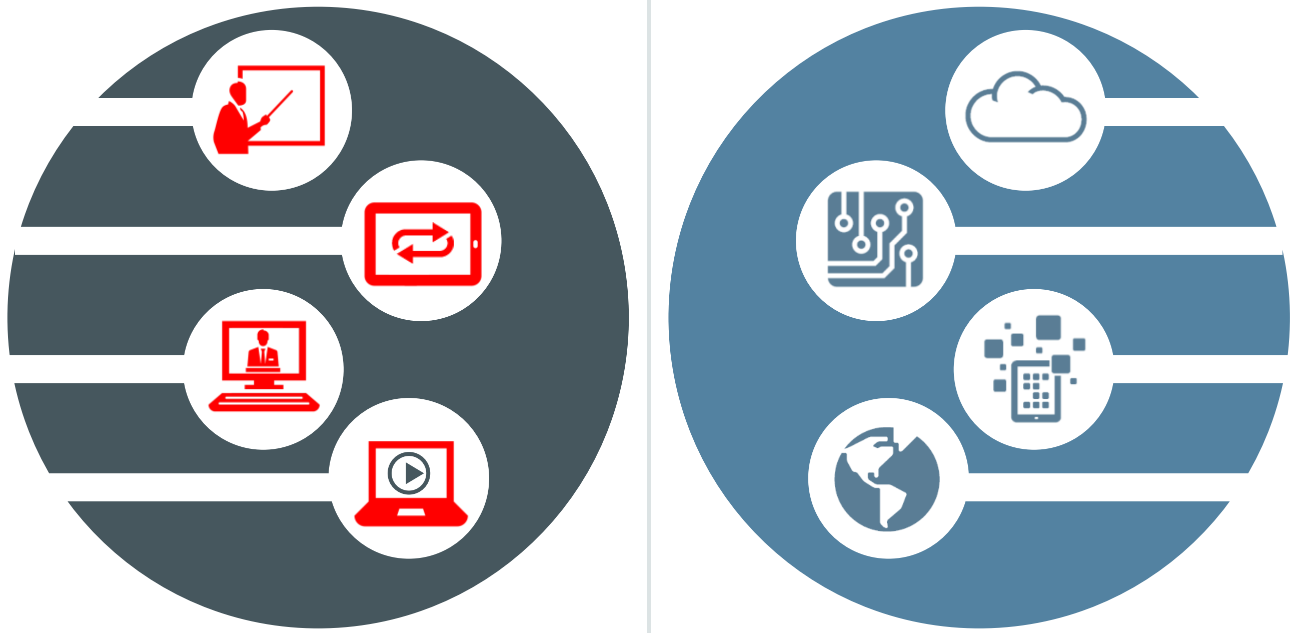
JavaOne™

ORACLE®

Keep Learning with Oracle University

ORACLE®

UNIVERSITY



education.oracle.com

Session Surveys

Help us help you!!

- Oracle would like to invite you to take a moment to give us your session feedback. Your feedback will help us to improve your conference.
- Please be sure to add your feedback for your attended sessions by using the Mobile Survey or in Schedule Builder.

HotSpot Synchronization

A Peek Under the Hood

David Buck
Principal Member of Technical Staff
Java SE
October 26, 2015

Safe Harbor Statement

Program Agenda

Howdy!

David Buck

- Java SE Sustaining Engineering
- I Fix JVM Bugs
- Hobbies:

  (non-Java) programming

Introduction

What We'll Cover

```
synchronized(this)   {
        c++;
}
```

What We'll Cover

```
synchronized(this)   {
        c++;
}
```

What We'll Cover

```
3: monitorenter

4: aload_0

5: dup
6: getfield #2 // Field c:I
9: iconst_1
10: iadd
11: putfield #2 // Field c:I
14: aload_1
15: monitorexit
```

What We'll Cover

```
3: monitorenter

4: aload_0

5: dup
6: getfield #2 // Field c:I
9: iconst_1
10: iadd
11: putfield #2 // Field c:I
14: aload_1
15: monitorexit
```

What We Won't Cover

What We Won’t Cover

- How to use locks

What We Won’t Cover

- How to use locks
- java.util.concurrent (JSR-166)

What We Won’t Cover

- How to use locks
- java.util.concurrent (JSR-166)
- Java’s memory model

Motivation

Motivation

- Avoiding premature optimization

Motivation

- Avoiding premature optimization
- Improve Profiling, Design, and Tuning

Motivation

- Avoiding premature optimization
- Improve Profiling, Design, and Tuning
- Fun!

Java Locking Review

Multithreading as Part of the Language

- So, what exactly **is** a Monitor?

Mutual Exclusion

Mutual Exclusion

Mut ex

$$\begin{tikzcd}
\text{Mutual} \arrow[dr] & & \text{Exclusion} \arrow[dl] \\
& \text{Mutex} &
\end{tikzcd}$$

condition variable

"Puffin crossing, London, UK" by secretlondon is licensed under CC BY-SA 3.0

Monitor = Mutex + Condition Variable

Monitor = Mutex + Condition Variable

- Mutex
  - synchronized keyword

Monitor = Mutex + Condition Variable

- Mutex
  - synchronized keyword
- Condition Variable
  - Object.wait()
  - Object.notify()
  - Object.notifyAll()

Java Locks are Recursive!

```
public synchronized void increment() {
        c++;
        printValue();
}

public synchronized void printValue() {
        System.out.println("My value is: " + c);
}
```

Memory Model

Establish a "happens-before" relationship

```
public class NoSync {
  private int c = 0;
  public void increment() {
          c++;
  }
}
```

```
public void increment();
  flags: ACC_PUBLIC
  Code:
  stack=3, locals=1, args_size=1
  0: aload_0
  1: dup
  2: getfield #2 // Field c:I
  5: iconst_1
  6: iadd
  7: putfield #2 // Field c:I
  10: return
```

Block Example

```
public void increment() {
        synchronized(this) {
                c++;
        }
}
```

Block Example

```
public void increment();
  flags: ACC_PUBLIC
  Code:
  stack=3, locals=3, args_size=1
  0: aload_0
  1: dup
  2: astore_1
  3: monitorenter
  4: aload_0
  5: dup
  6: getfield #2 // Field c:I
  9: iconst_1
  10: iadd
  11: putfield #2 // Field c:I
  14: aload_1
  15: monitorexit
  16: goto 24
  19: astore_2
  20: aload_1
  21: monitorexit
  22: aload_2
  23: athrow
  24: return
  Exception table:
  from to target type
          4 16 19 any
          19 22 19 any
```

Block Example

```
public void increment();
  flags: ACC_PUBLIC
  Code:
  stack=3, locals=3, args_size=1
  0: aload_0
  1: dup
  2: astore_1
  3: monitorenter
  4: aload_0
  5: dup
  6: getfield #2 // Field c:I
  9: iconst_1
  10: iadd
11: putfield #2 // Field c:I
14: aload_1
15: monitorexit
16: goto 24
19: astore_2
20: aload_1
21: monitorexit
22: aload_2
23: athrow
24: return
Exception table:
from to target type
        4 16 19 any
        19 22 19 any
```

Method Example

```
public synchronized void increment() {
        c++;
}
```

Method Example

```
public synchronized void increment();
  flags: ACC_PUBLIC, ACC_SYNCHRONIZED
  Code:
  stack=3, locals=1, args_size=1
  0: aload_0
  1: dup
  2: getfield #2 // Field c:I
  5: iconst_1
  6: iadd
  7: putfield #2 // Field c:I
  10: return
```

Method Example

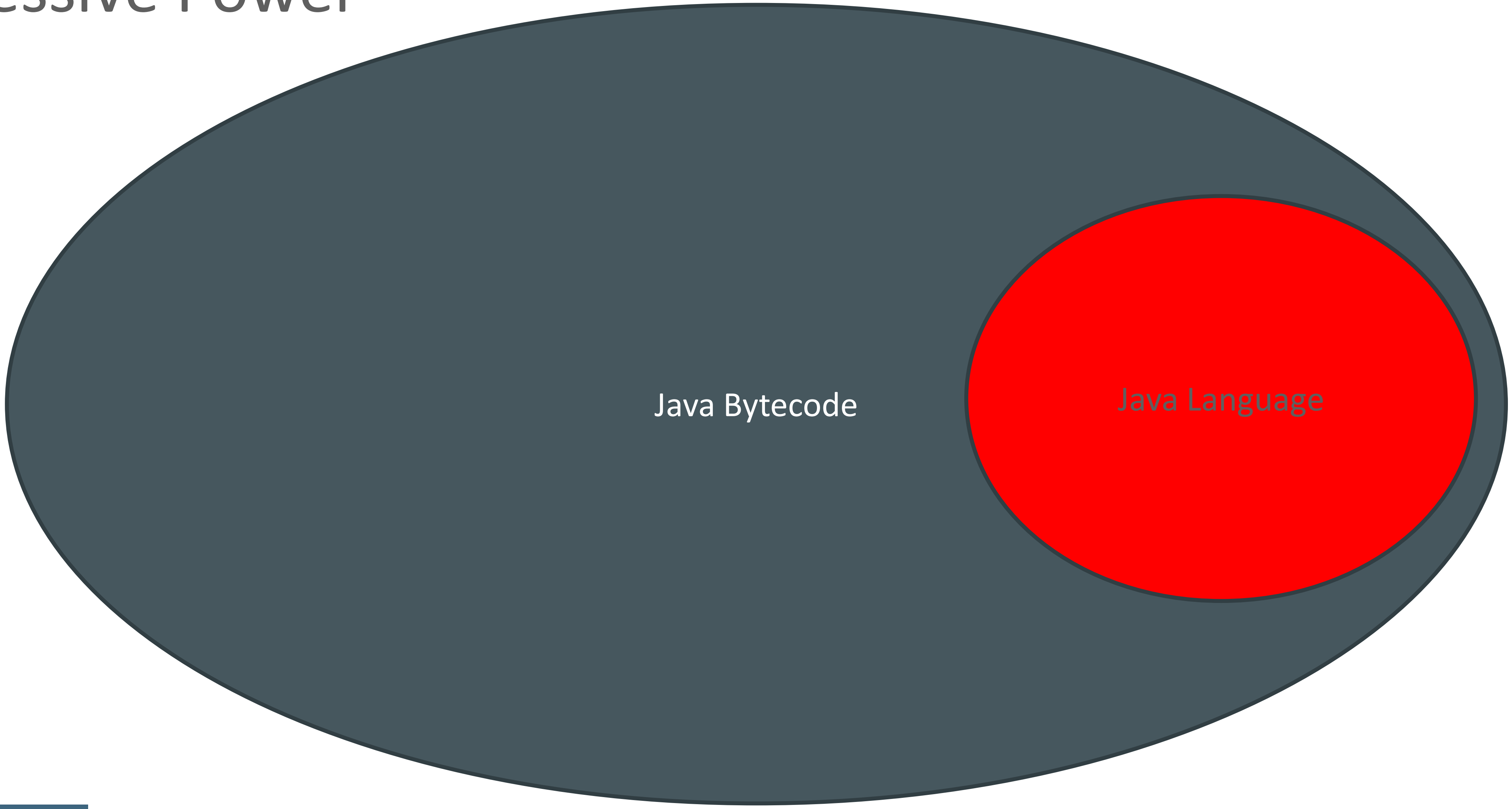
```
public synchronized void increment();
  flags: ACC_PUBLIC, ACC_SYNCHRONIZED
  Code:
  stack=3, locals=1, args_size=1
  0: aload_0
  1: dup
  2: getfield #2 // Field c:I
  5: iconst_1
  6: iadd
  7: putfield #2 // Field c:I
  10: return
```

Impossible in Java Language!

```
public void lockMe();
  flags: ACC_PUBLIC
  Code:
  stack=1, locals=1, args_size=1
  0: aload_0
  1: monitorenter
  2: return
```

```
public void unlockMe();
  flags: ACC_PUBLIC
  Code:
  stack=1, locals=1, args_size=1
  0: aload_0
  1: monitorexit
  2: return
```

Expressive Power



Java Monitor Limitations

Java Monitor Limitations

- No way to check status of a lock

Java Monitor Limitations

- No way to check status of a lock
- No timeout

Java Monitor Limitations

- No way to check status of a lock
- No timeout
- No way to cancel

Java Monitor Limitations

- No way to check status of a lock
- No timeout
- No way to cancel
- Must be recursive

Java Monitor Limitations

- No way to check status of a lock
- No timeout
- No way to cancel
- Must be recursive
- No reader / writer locking

Java Monitor Limitations

- No way to check status of a lock
- No timeout
- No way to cancel
- Must be recursive
- No reader / writer locking
- Security Issues

java.util.concurrent

HotSpot’s Implementation

Design Goals

Design Goals

- Every object may be used as a monitor

Design Goals

- Every object may be used as a monitor
- But most objects never are

Design Goals

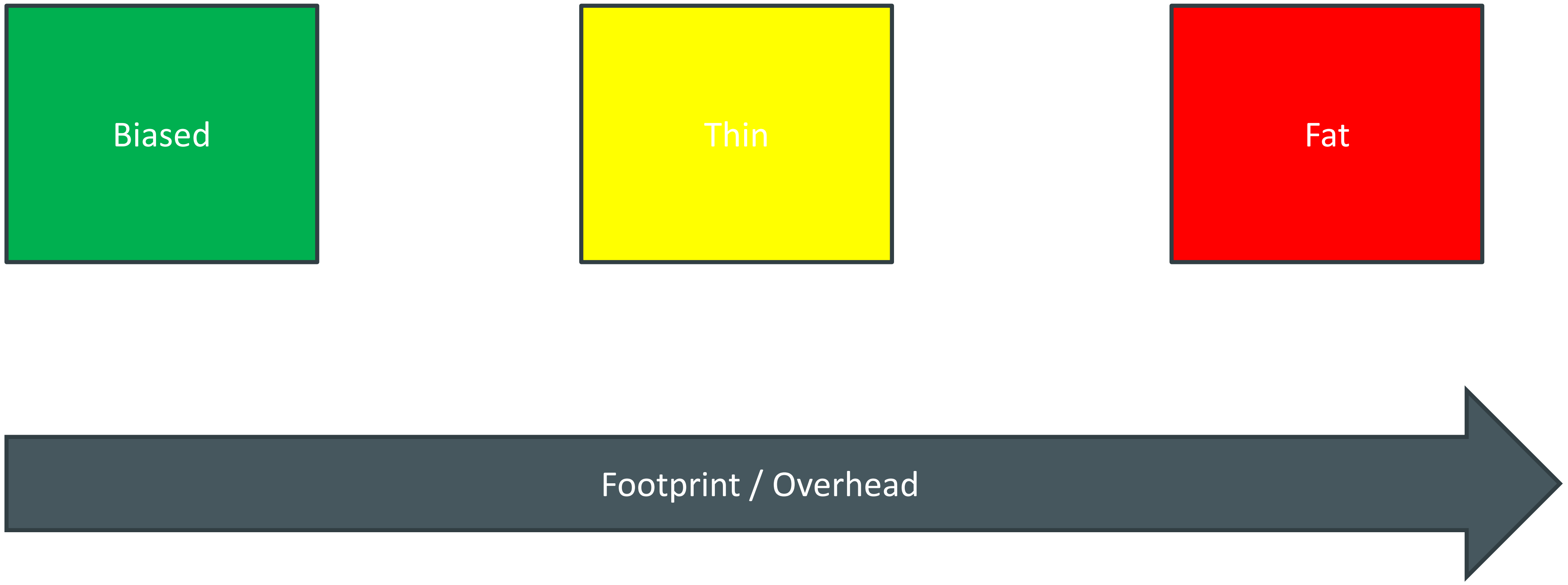
- Every object may be used as a monitor
- But most objects never are
- Those that are locked, are usually not used by multiple threads

Design Goals

- Every object may be used as a monitor
- But most objects never are
- Those that are locked, are usually not used by multiple threads
- Those that are used by multiple threads, are usually not contended

Lock Types

- Fat
- Thin
- Biased


Biased
Thin
Fat
Footprint / Overhead

Fat Lock

- Rely on OS scheduler
- Best use case: long wait times

- AKA: Heavyweight lock, inflated lock

Thin Lock

- Spin until lock is available
- Best use case: short pause times

- AKA: spin lock, stack lock (HS), lightweight lock

Biased Lock

- Only a single thread repeatedly locks object
- If other thread needs lock, bias needs to be revoked

BiasedLock Revocation

- Stop thread that currently holds bias (STW)
- Check if thread “really” holds lock (stack walk)

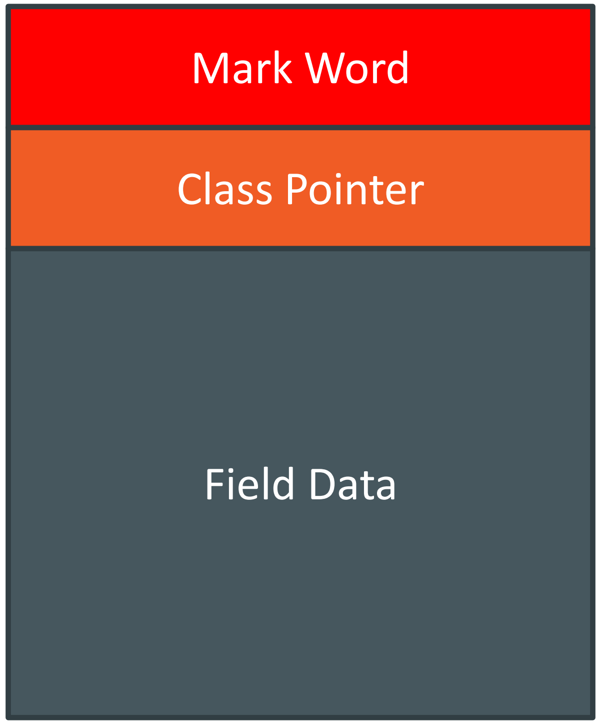
BiasedLock Banning

- Object Level
- Class Level
- Booting Phase

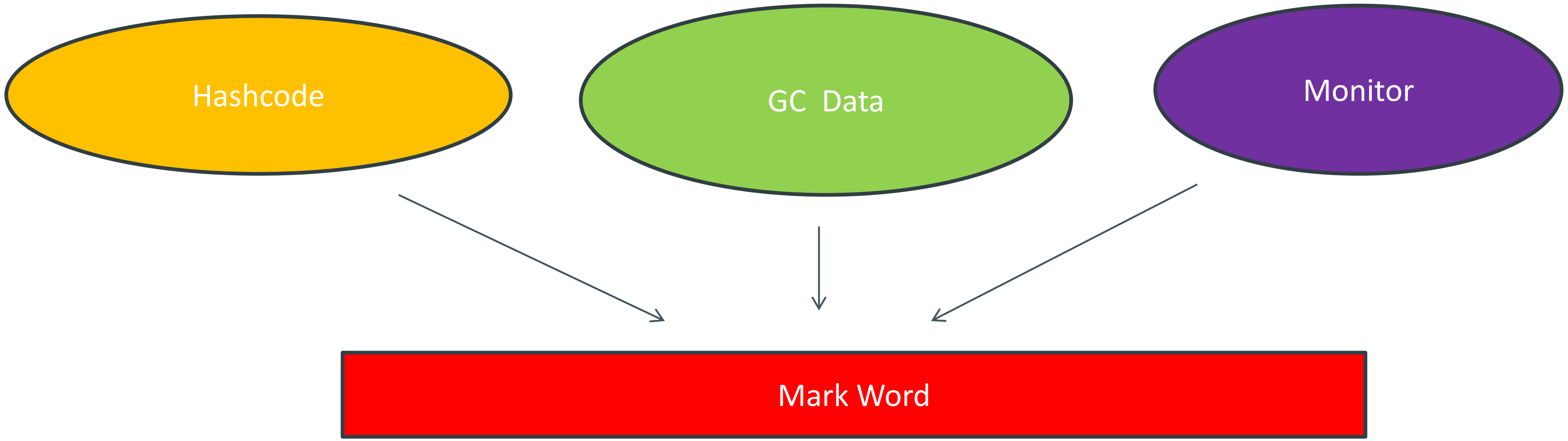
Per-Object Data

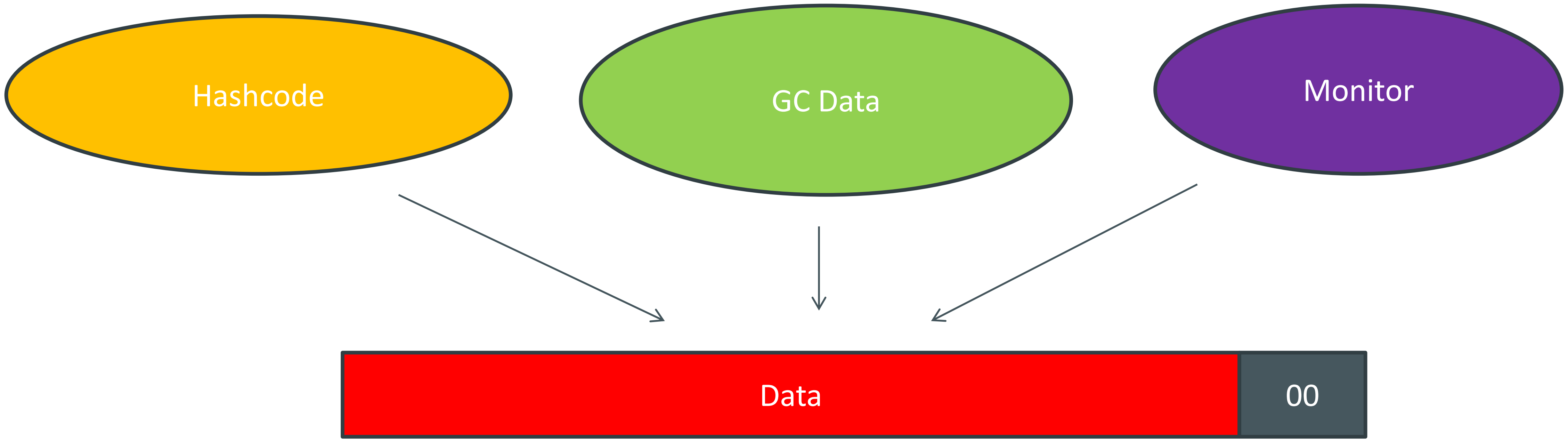
- Field data
- Metadata
  - Monitor condition
  - GC bookkeeping (e.g. age)
  - Hash code

Object Header

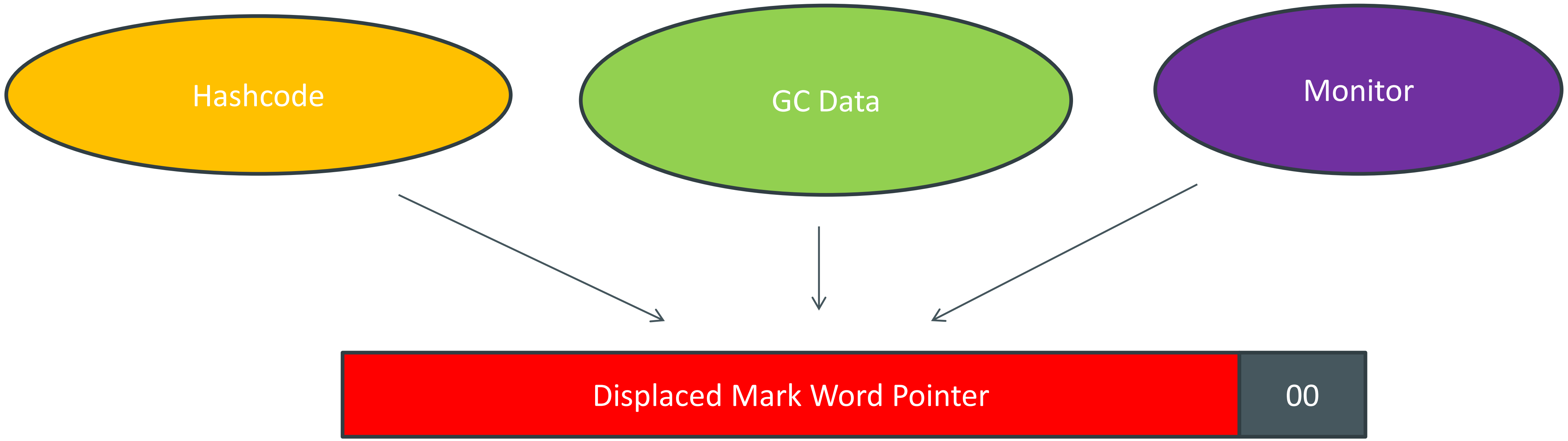


Busy Mark Word is Busy

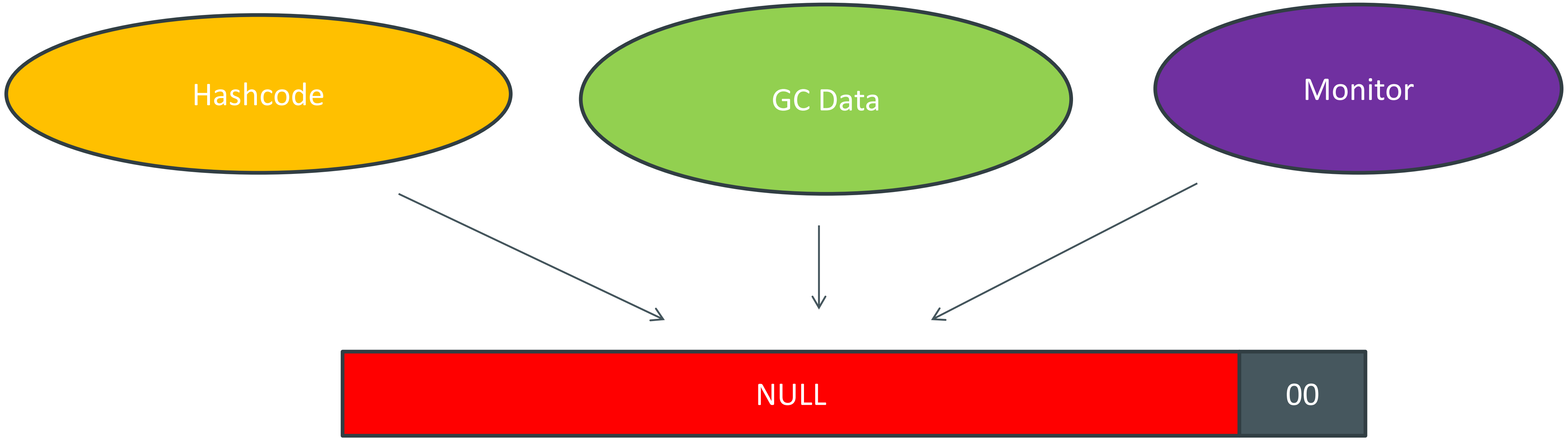



Hashcode
GC Data
Monitor
Data
00

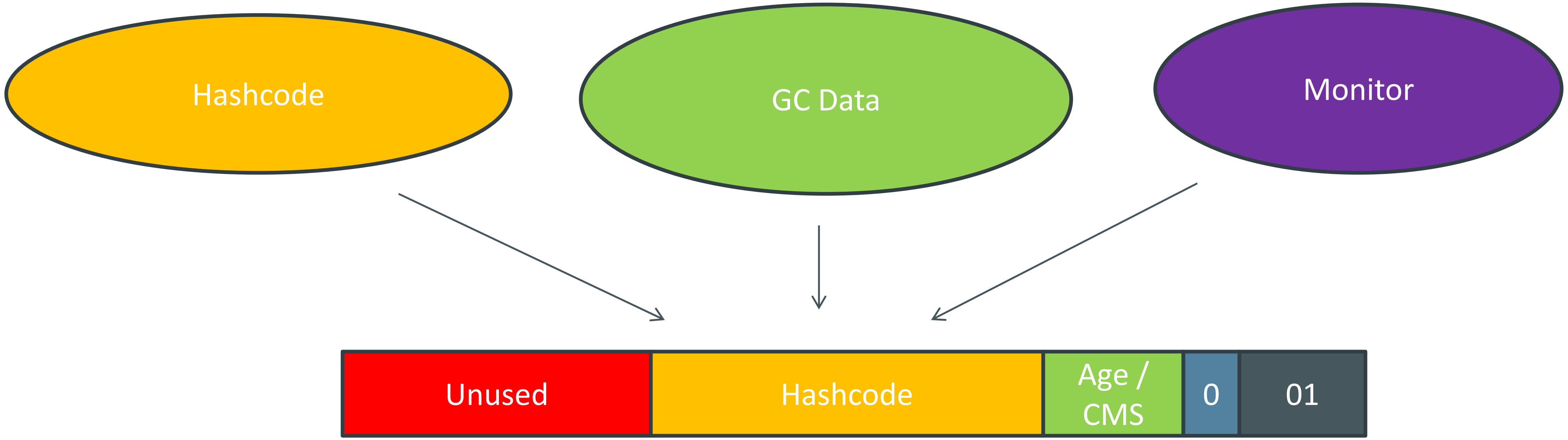
Thin Locked



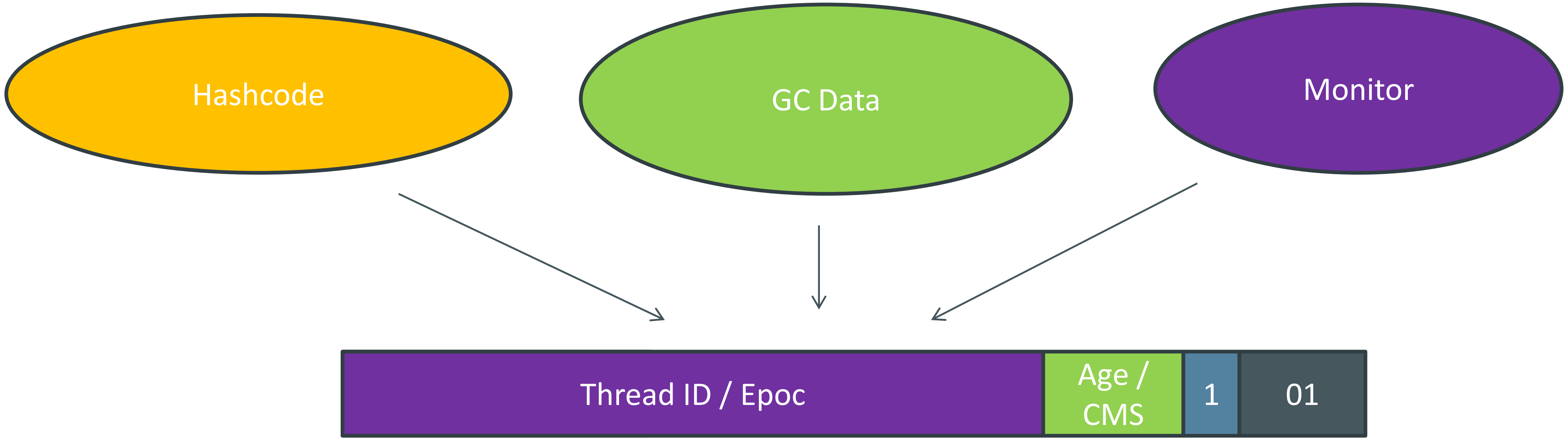
Inflating



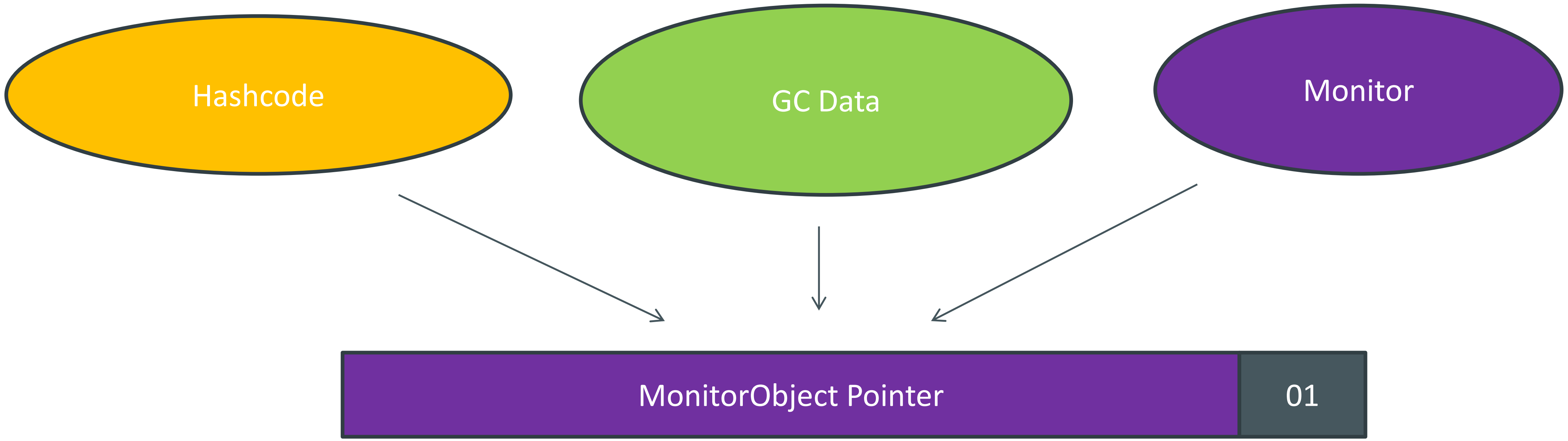
Unlocked
Banned for Biased Locking



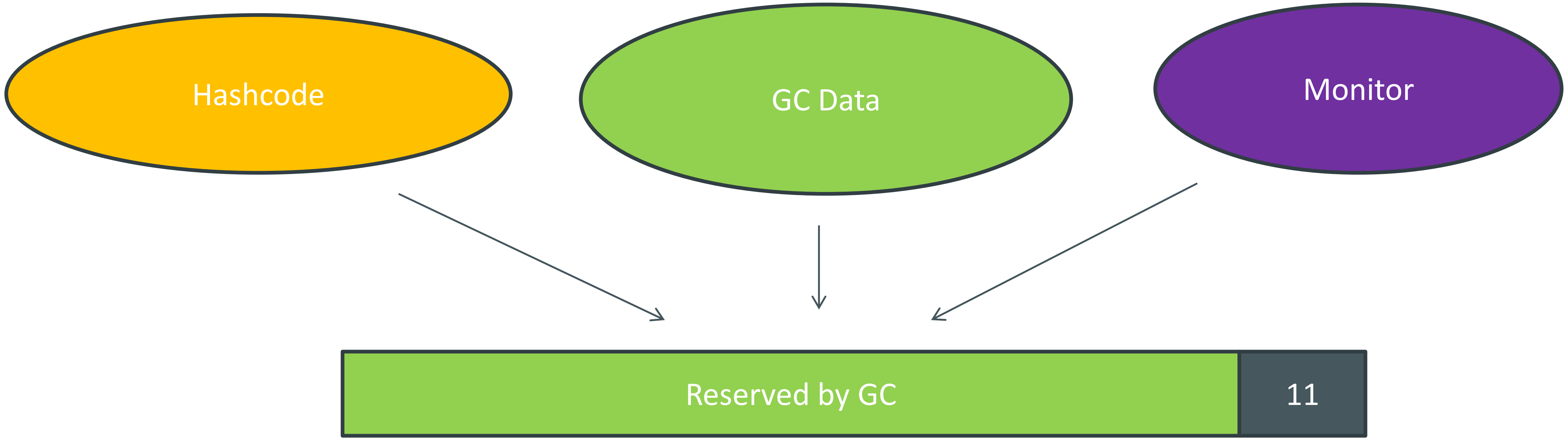
Biased



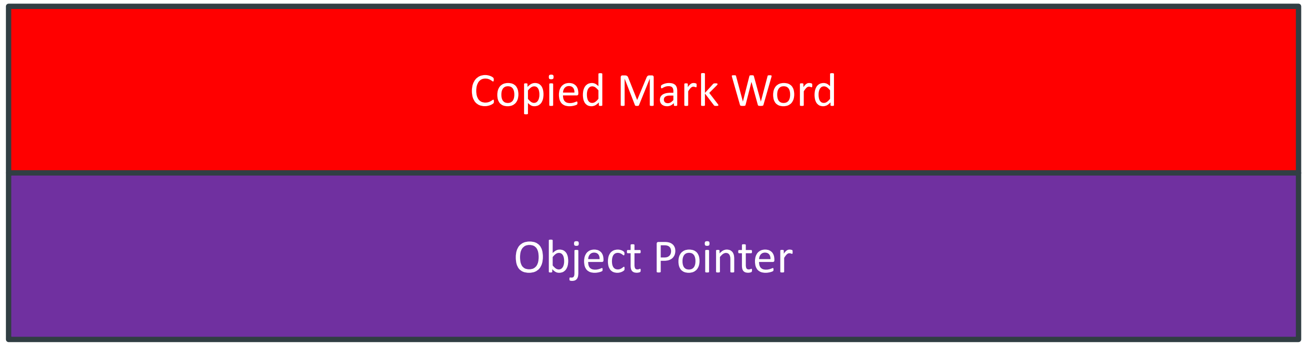
Fat Locked



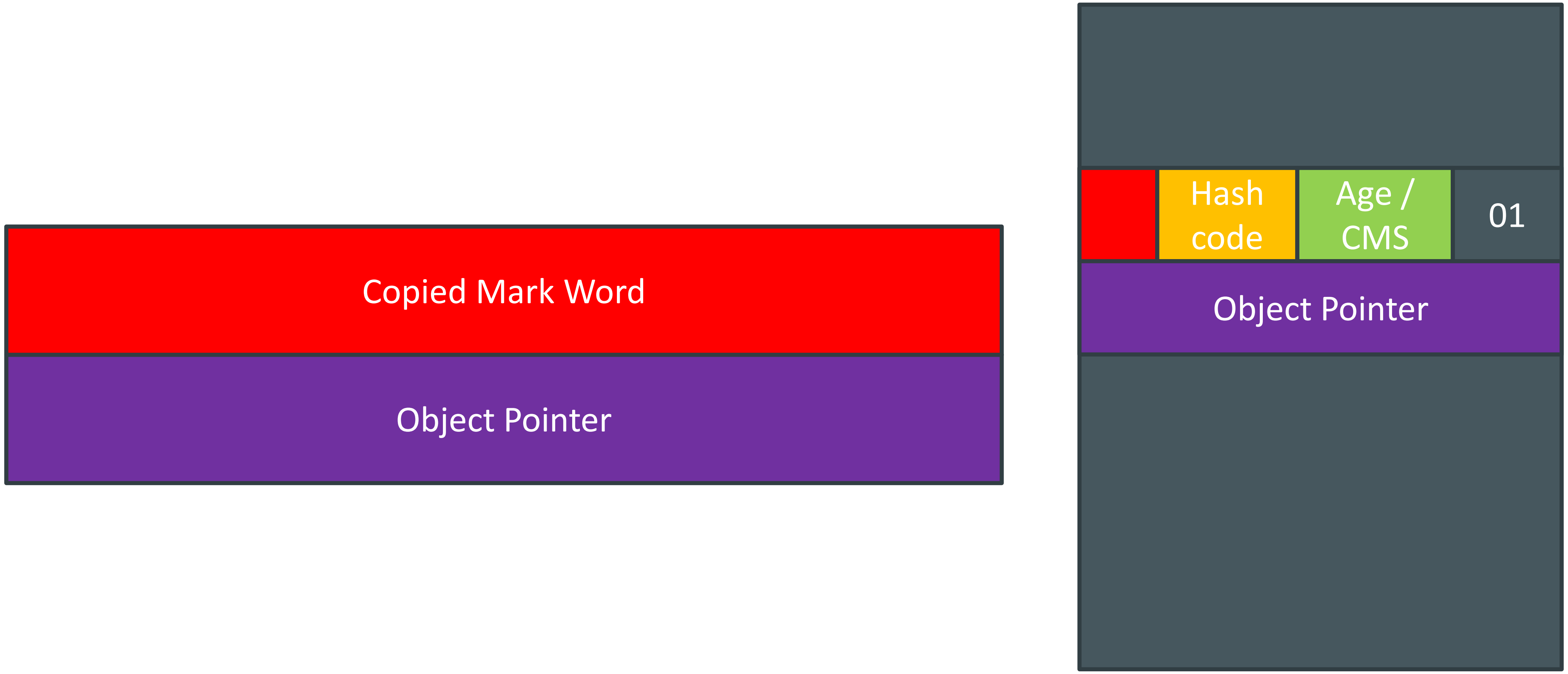
GC Running（STW）



Lock Record

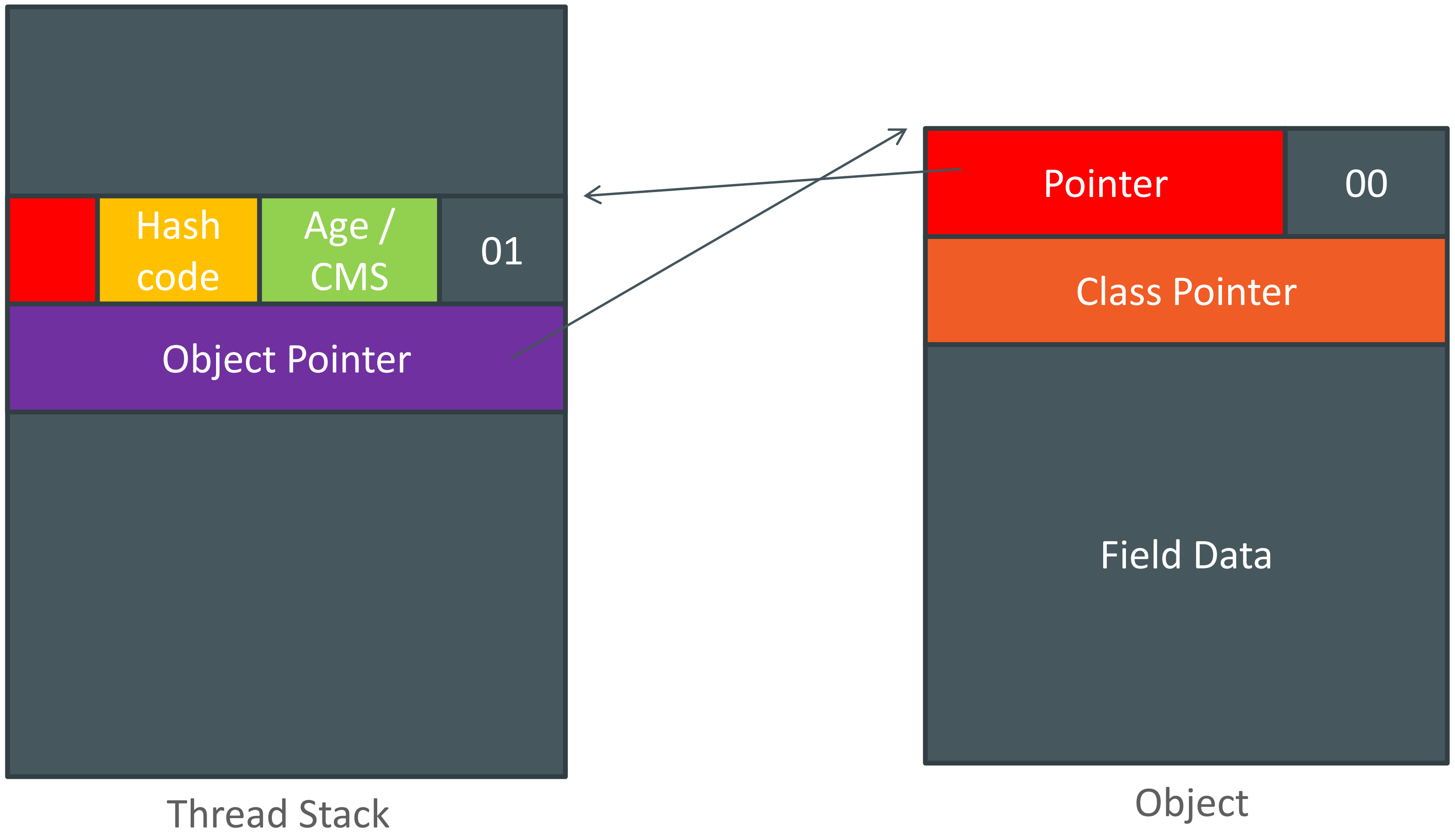


Lock Record

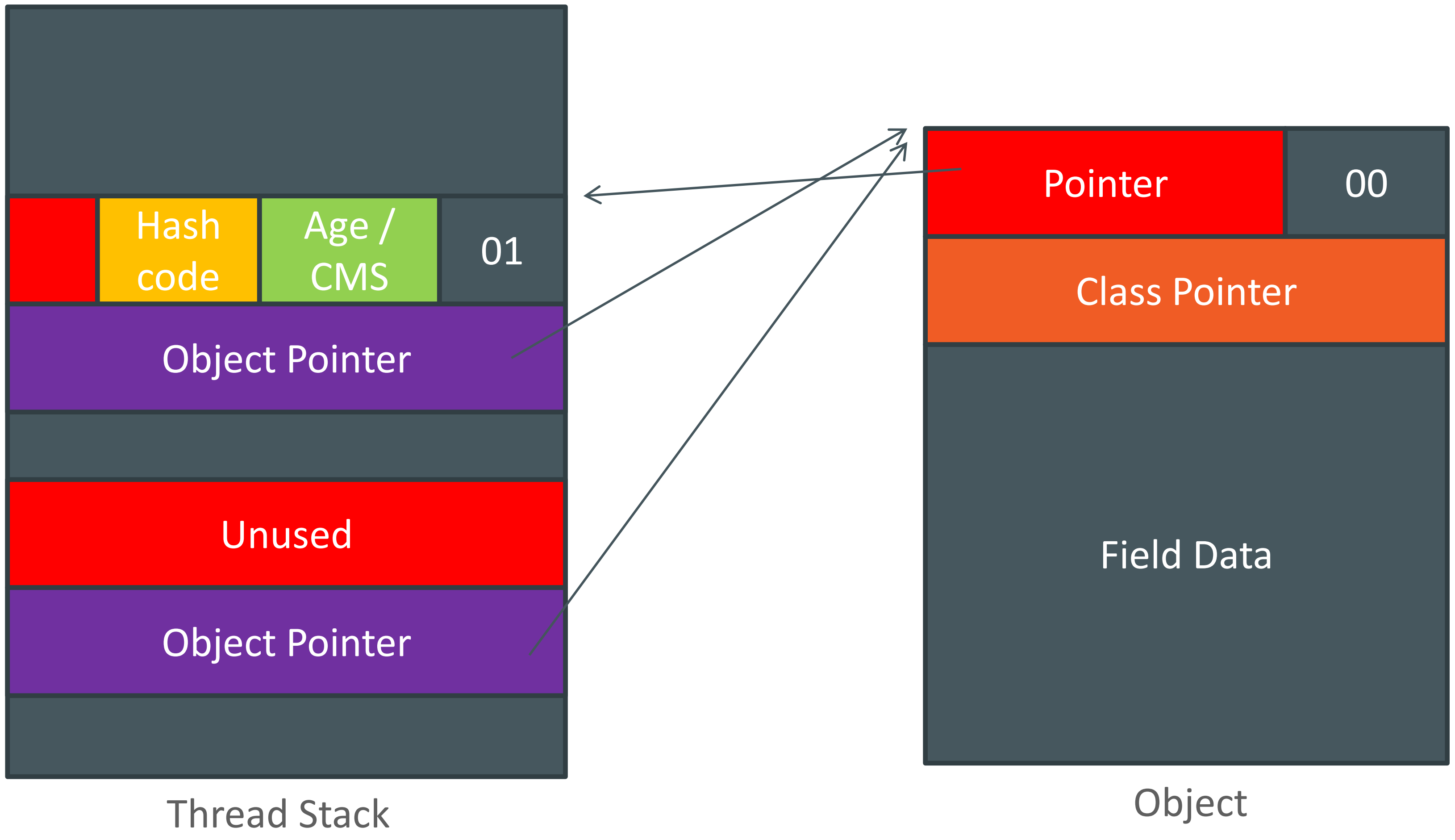


Thread Stack

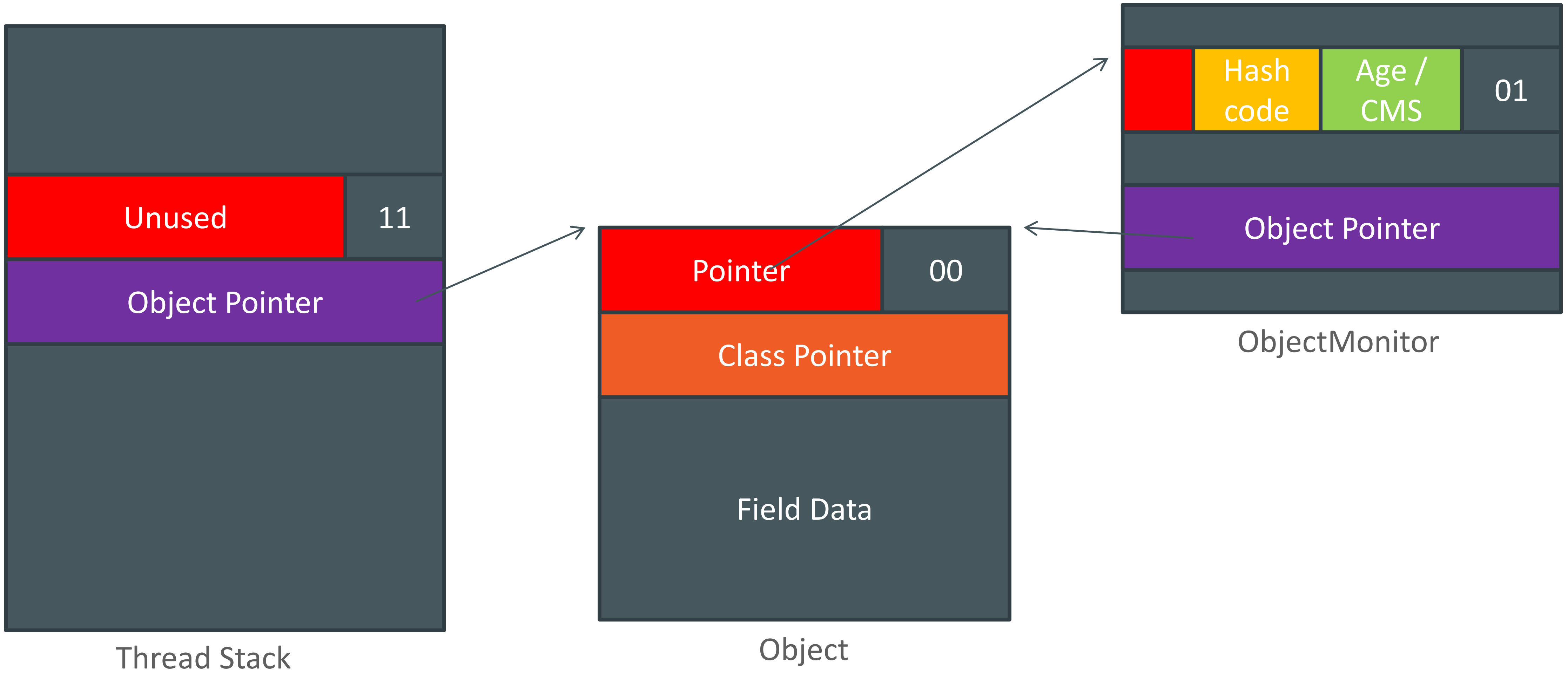
Thin Lock


Hash code
Age / CMS
01
Object Pointer
Pointer
00
Class Pointer
Field Data
Thread Stack
Object

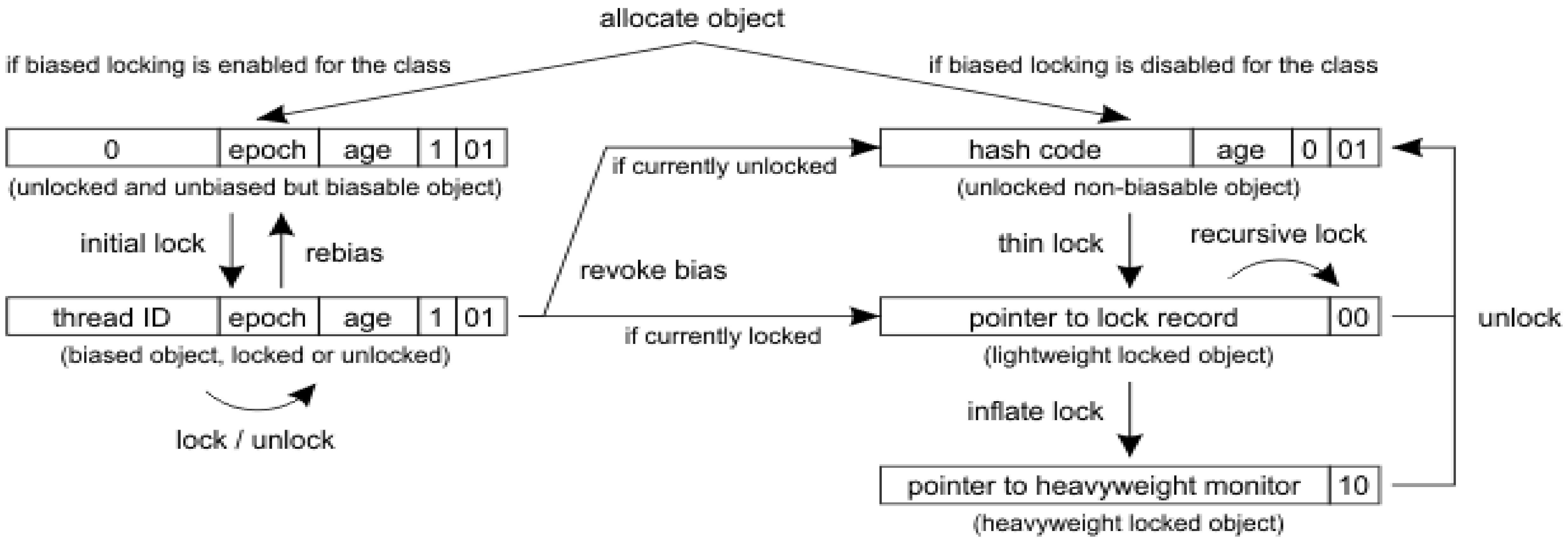
Thin Lock (Recursive)

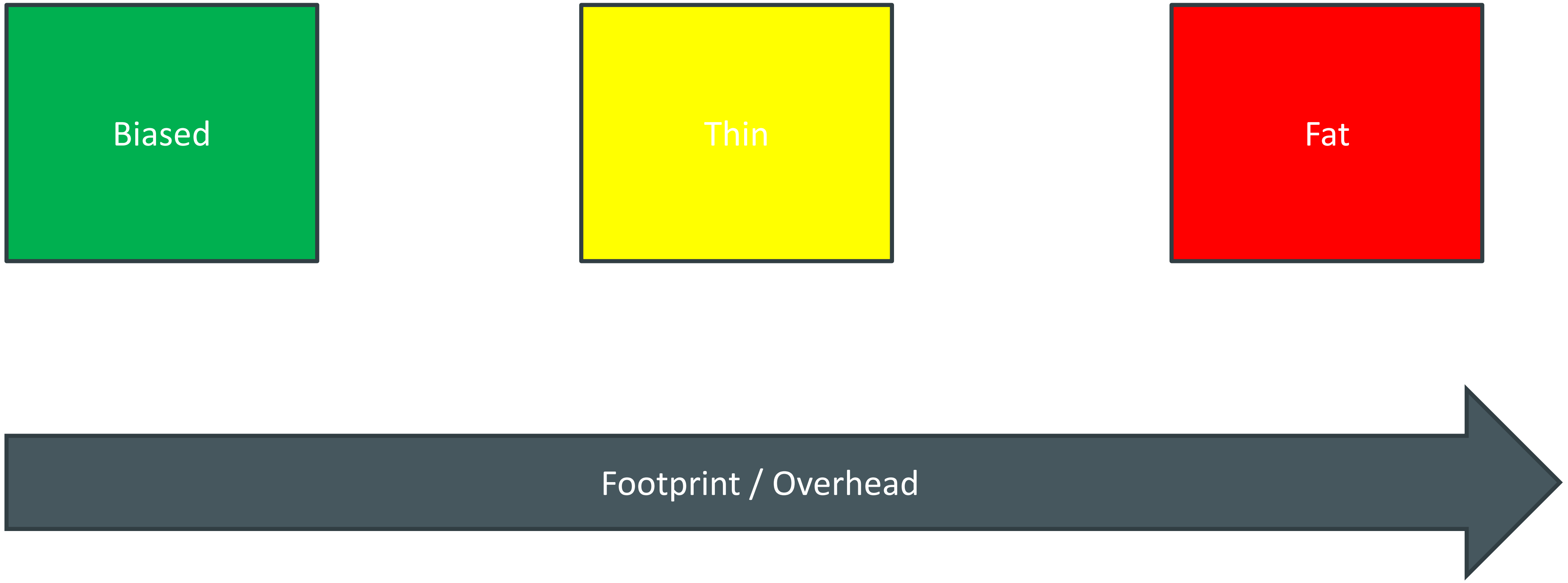


Fat Lock


Unused
11
Object Pointer
Thread Stack
Pointer
00
Class Pointer
Field Data
Object
Hash code
Age / CMS
01
Object Pointer
ObjectMonitor

Lock Transitions


allocate object
if biased locking is enabled for the class
if biased locking is disabled for the class
0
epoch
age
1
01
(unlocked and unbiased but biasable object)
hash code
age
0
01
(unlocked non-biasable object)
if currently unlocked
initial lock
rebias
thin lock
recursive lock
revoke bias
thread ID
epoch
age
1
01
(biased object, locked or unlocked)
pointer to lock record
00
(lightweight locked object)
if currently locked
unlock
lock / unlock
inflate lock
pointer to heavyweight monitor
10
(heavyweight locked object)


Biased
Thin
Fat
Footprint / Overhead

Profiling & Tuning

Profiling

DANGER:

Performance Impact Ahead!

Profiling

- Performance Counters
- DTrace
- Java Flight Recorder

Performance Counters

- No performance impact
- Not officially supported
- Intended for HotSpot troubleshooting
- Example:

```
jstat -snap -J-Djstat.showUnsupported=true <JVM_PID> |grep _sync
```

DTrace

- Most flexible
- Higher learning curve
- Supported platforms
  - Solaris
  - Oracle Linux
  - OSX
- Must use -XX:+**DTraceMonitorProbes**

DTrace

Mutex Probes

- monitor-contended-enter
- monitor-contended-entered
- monitor-contended-exit

DTrace

Condition Variable Probes

- monitor-wait
- monitor-waited
- monitor-notify
- monitor-notifyAll

Java Flight Recorder

- Free for development use
- Supported Platforms: all OracleJDK Java SE Platforms

Options

- PrintConcurrentLocks
- UseBiasedLocking
- DTraceMonitorProbes
- BiasedLockingStartupDelay
- PrintBiasedLockingStatistics
- TraceBiasedLocking
- TraceMonitorInflation
- MonitorInUseLists
- TraceMonitorMismatch
- UseHeavyMonitors
- BiasedLockingBulkRebiasThreshold
- BiasedLockingBulkRevokeThreshold
- BiasedLockingDecayTime
- SyncKnobs

Options

- PrintConcurrentLocks
- UseBiasedLocking
- DTraceMonitorProbes
- BiasedLockingStartupDelay
- PrintBiasedLockingStatistics
- TraceBiasedLocking
- TraceMonitorInflation
- MonitorInUseLists
- TraceMonitorMismatch
- UseHeavyMonitors
- BiasedLockingBulkRebiasThreshold
- BiasedLockingBulkRevokeThreshold
- BiasedLockingDecayTime
- SyncKnobs

PrintConcurrentLocks

- Displays java.util.concurrent locks in thread dumps just like normal locks!

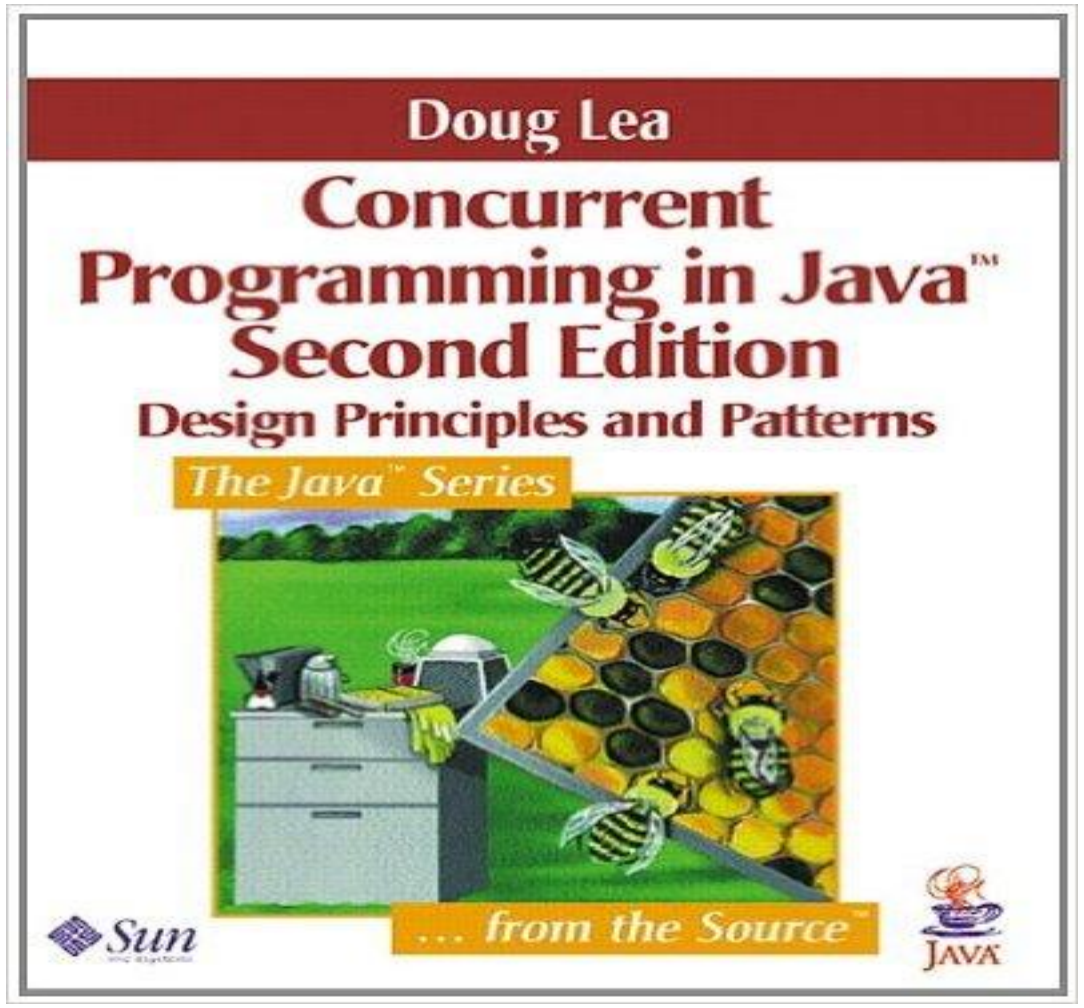
UseBiasedLocking

- Disables biased locking
- Worth trying (benchmarking) on systems with very high contention

Everything Else

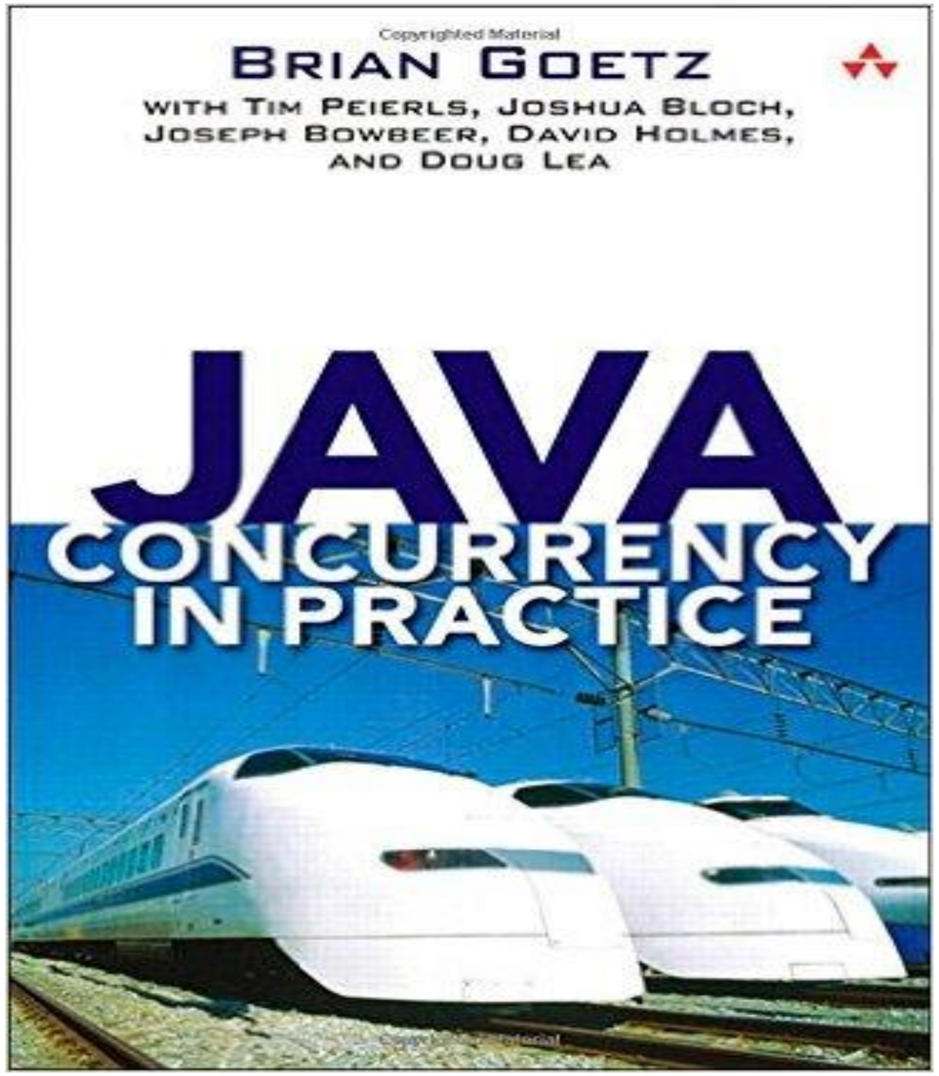
Concurrent Programming in Java™: Design Principles and Patterns

- JDK 1.2 Era
  - No modern memory model
  - The source of java.util.concurrent
- Focus on design
- A classic


Doug Lea
Concurrent Programming in Java™ Second Edition
Design Principles and Patterns
The Java™ Series
... from the Source™
Sun
JAVA

Java Concurrency in Practice

- JDK 1.6 Era
  - New Memory Model
  - java.util.concurrent
- If you read only **one** book on Java concurrency…


Copyrighted Material
BRIAN GOETZ
WITH TIM PEIERLS, JOSHUA BLOCH, JOSEPH BOWBEER, DAVID HOLMES, AND DOUG LEA
JAVA CONCURRENCY IN PRACTICE

Summary

- Leave optimization up to the JVM
- If simple monitors do not provide what you need, check out java.util.concurrent
- Profiling tools: JFR or DTrace
  - Watch out for performance impact
- Everyone really should read CPiJ and JCiP

Thank You!!!

References

- [jstat man page]

  https://docs.oracle.com/javase/8/docs/technotes/tools/unix/jstat.html

- [DTrace Probes in HotSpot VM]

  http://docs.oracle.com/javase/8/docs/technotes/guides/vm/dtrace.html

- [JMC Tutorial]

  http://hirt.se/blog/?p=611

- [David Dice's Weblog]

  https://blogs.oracle.com/dave/

- [HotSpot Internals]

  https://wiki.openjdk.java.net/display/HotSpot/Main

JavaOne™

ORACLE®