
ORACLE®

JavaOne™

ORACLE®

Keep Learning with Oracle University

ORACLE UNIVERSITY

Classroom Training

Learning Subscription

Live Virtual Class

Training On Demand

Cloud

Technology

Applications

Industries

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

JavaOne
ORACLE

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Program Agenda

1. Introduction
2. Java Locking Review
3. HotSpot's Implementation
4. Profiling & Tuning
5. Everything Else

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

Mutual Exclusion

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

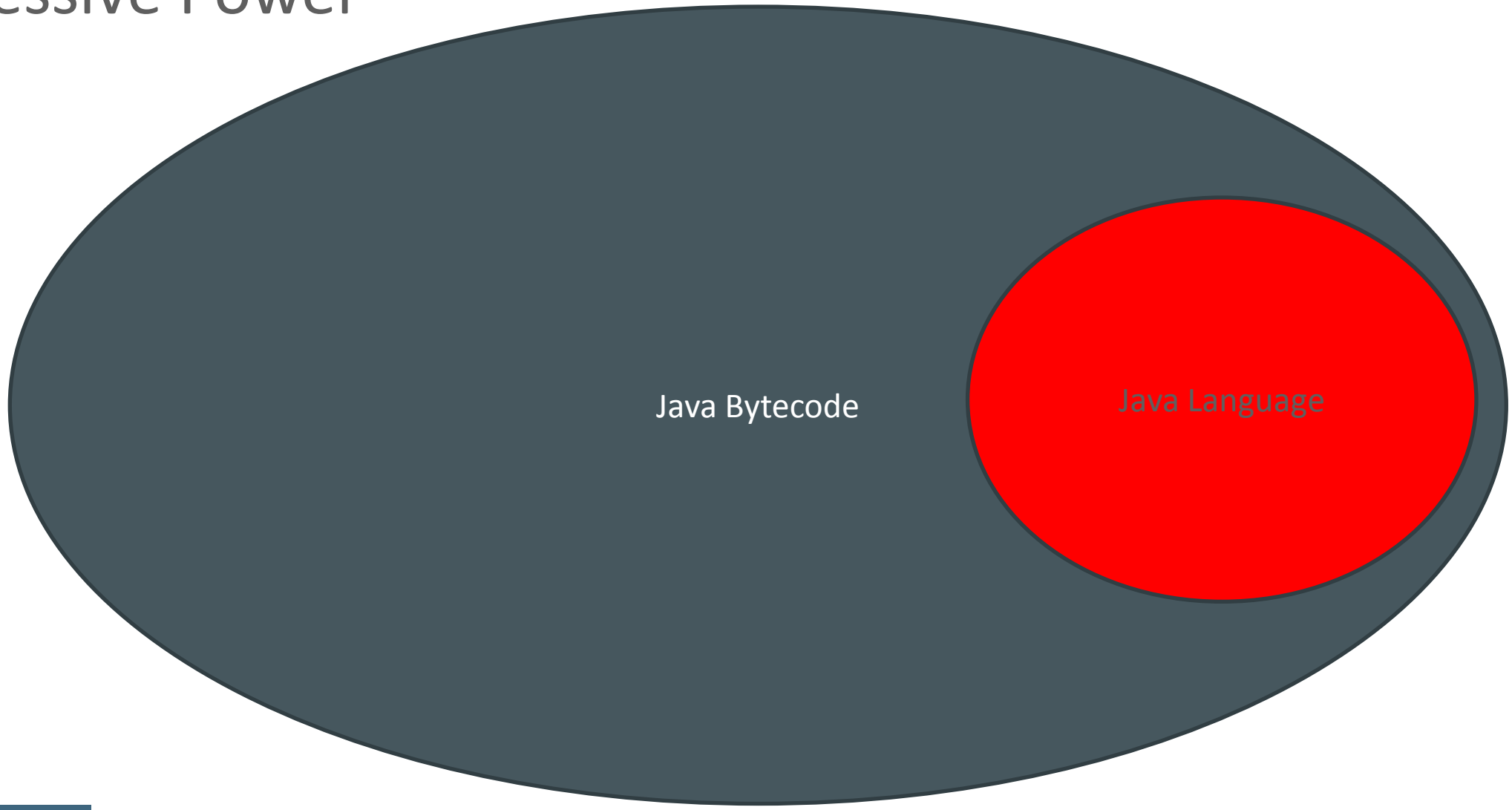
```
public synchronized void increment();
  flags: ACC_PUBLIC, ACC_SYNCHRONIZED
  Code:
  stack=3, locals=1, args_size=1
  0: aload_0
  1: dup
  2: getfield #2 // Field c:I
  5: iconst_1
  6: iadd
  7: putfield #2 // Field c:I
  10: return
```

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

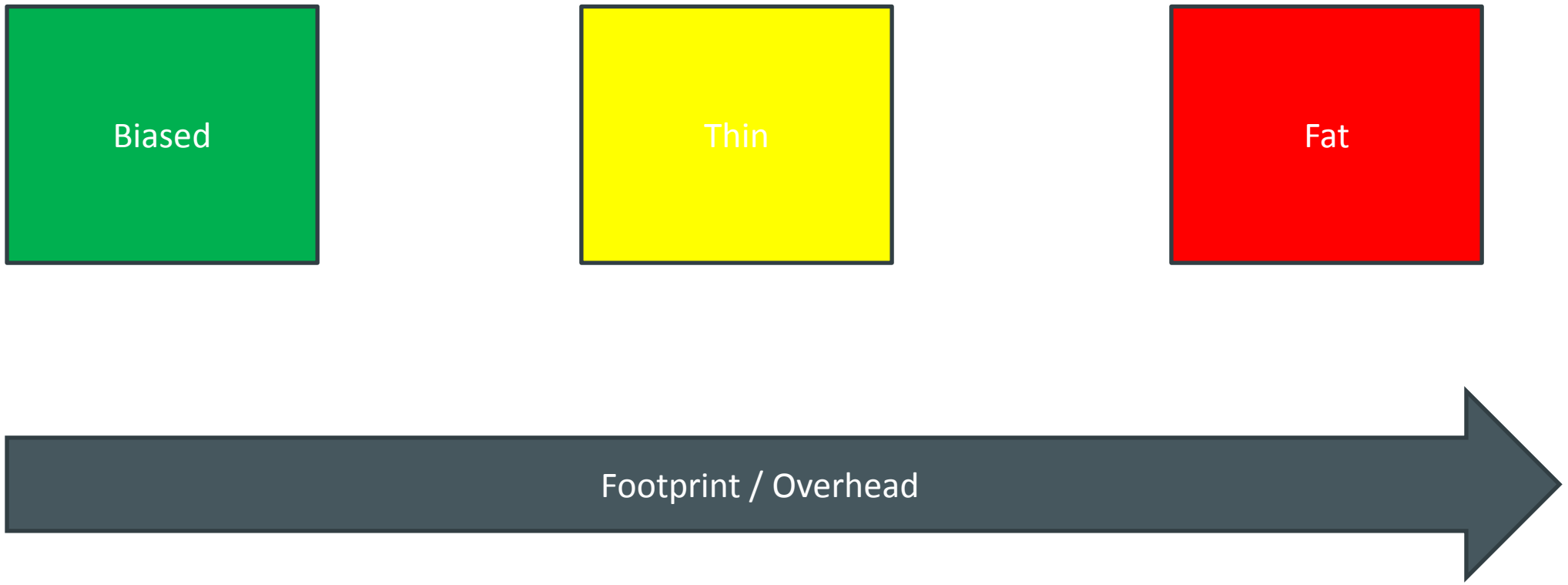
- Every object may be used as a monitor
- But most objects never are
- Those that are locked, are usually not used by multiple threads

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



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)

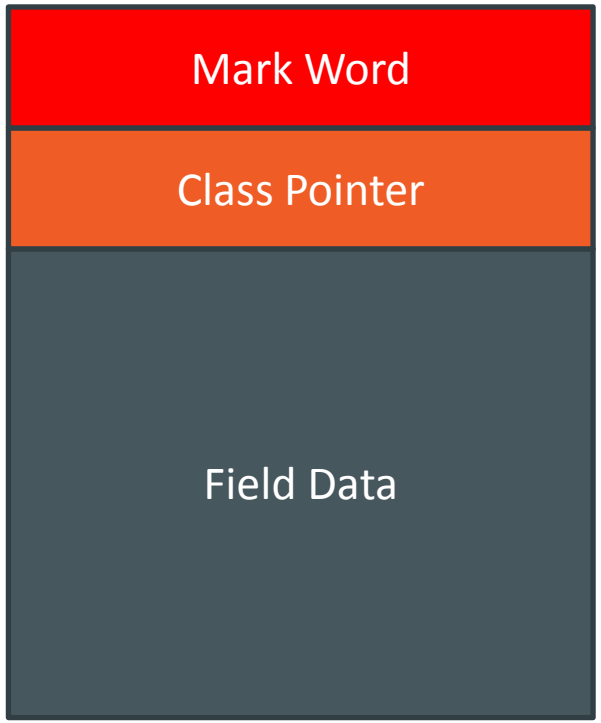
BiasedLock Banning

- Object Level
- Class Level
- Booting Phase

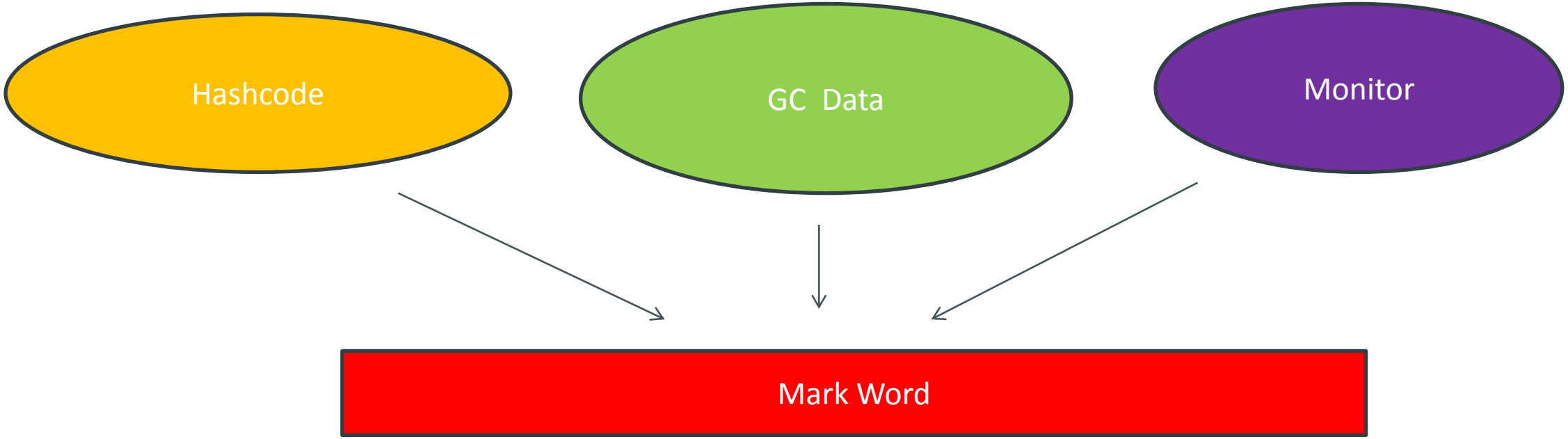
Per-Object Data

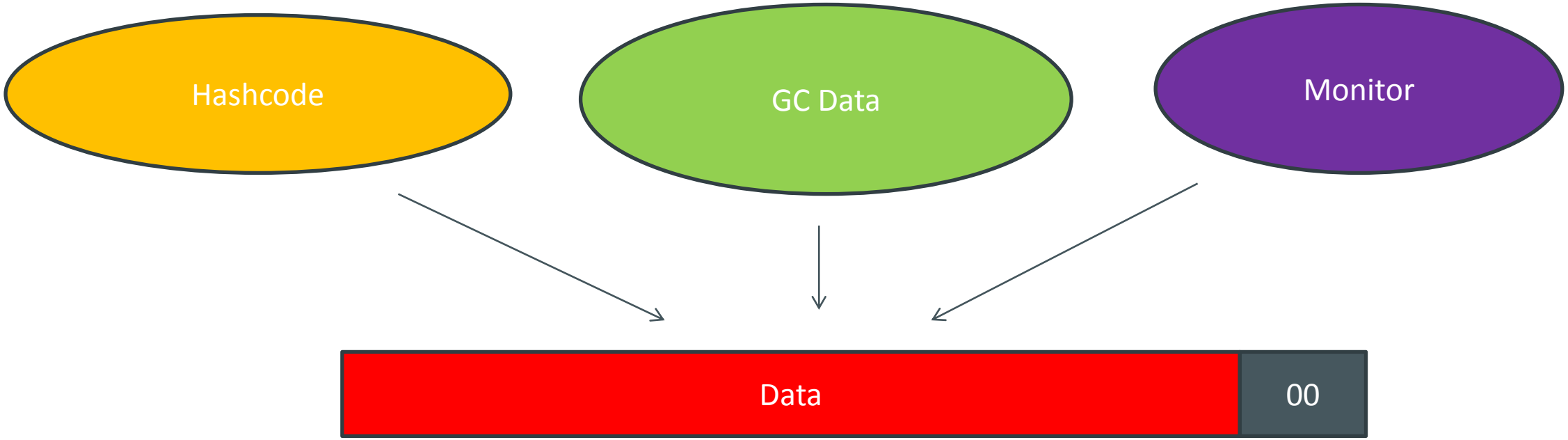
- Field data
- Metadata
  - Monitor condition
  - GC bookkeeping (e.g. age)
  - Hash code

Object Header

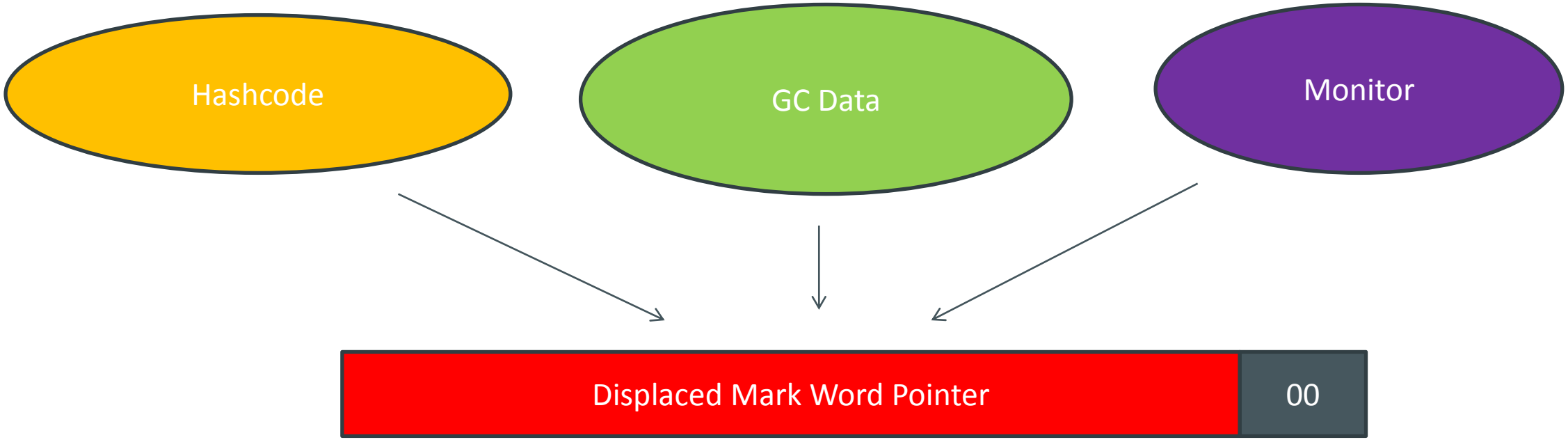


Busy Mark Word is Busy

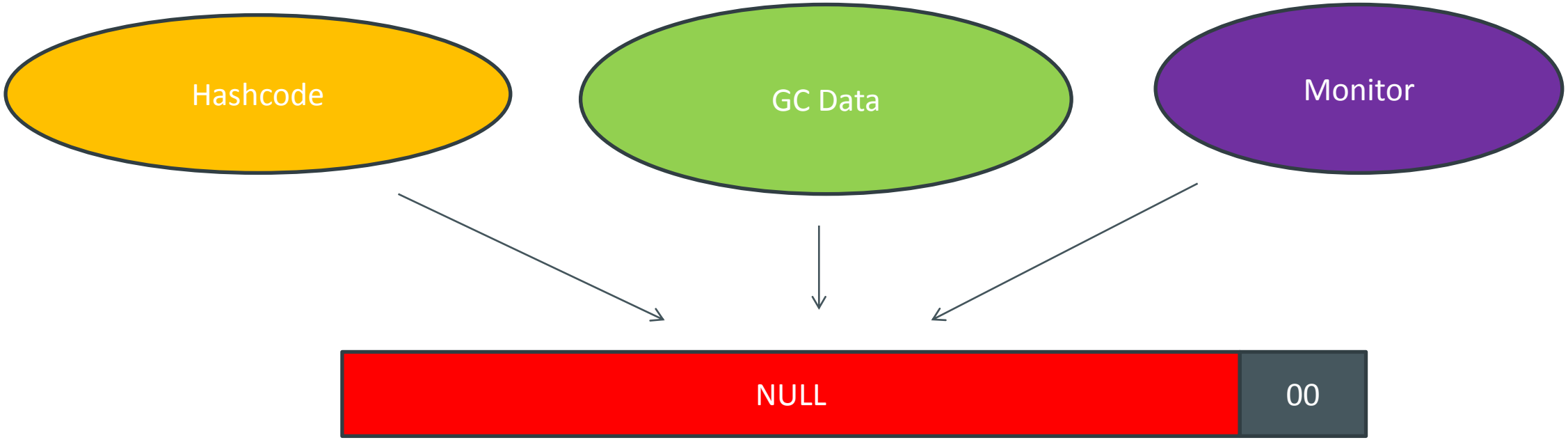




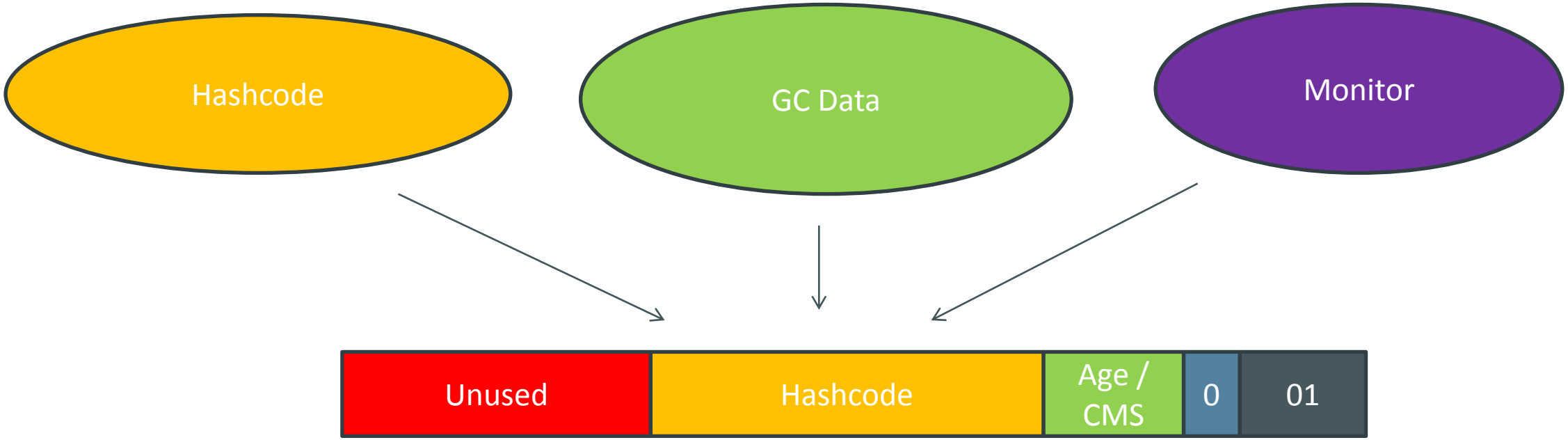
Thin Locked



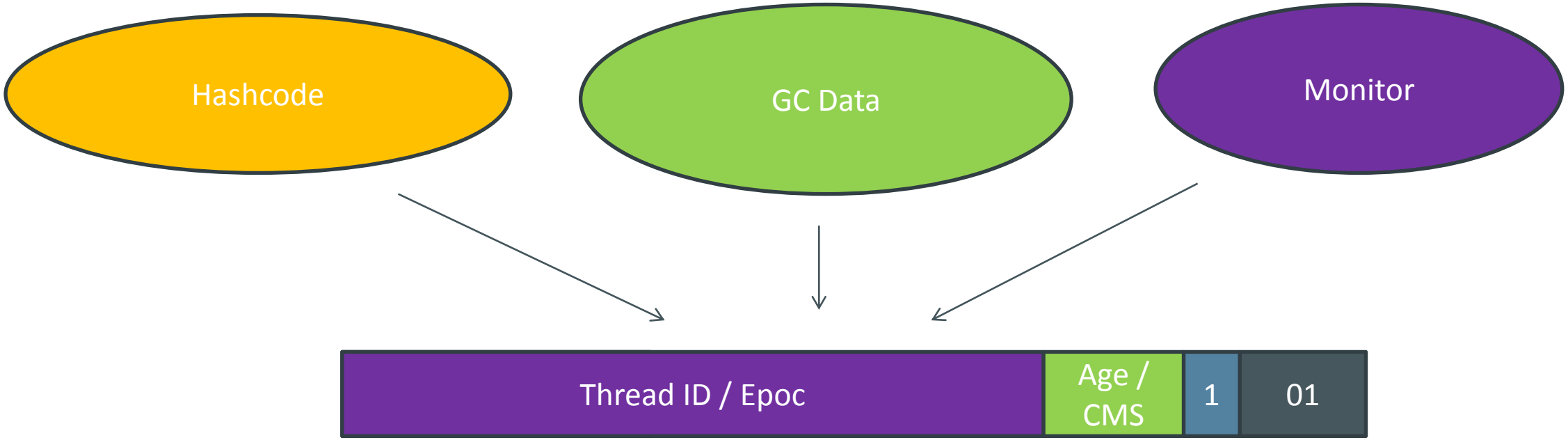
Inflating



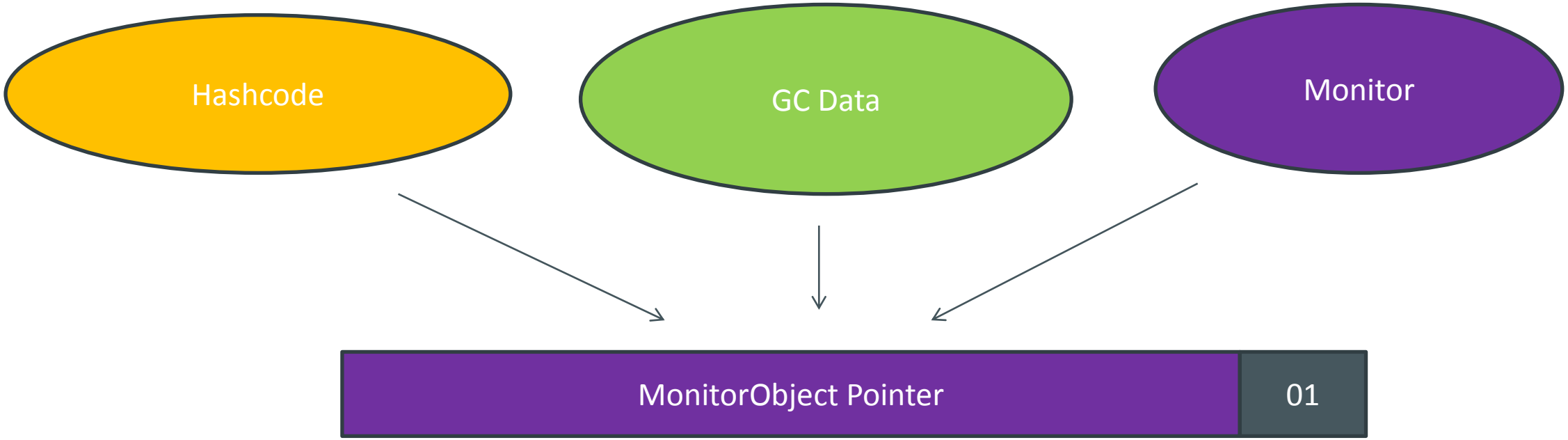
Unlocked
Banned for Biased Locking



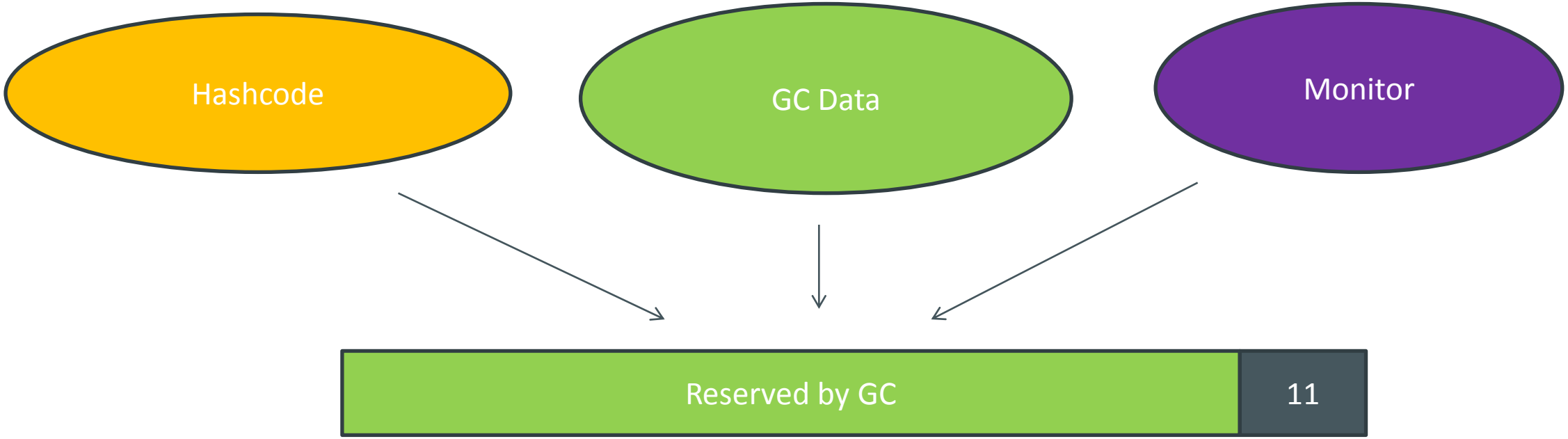
Biased



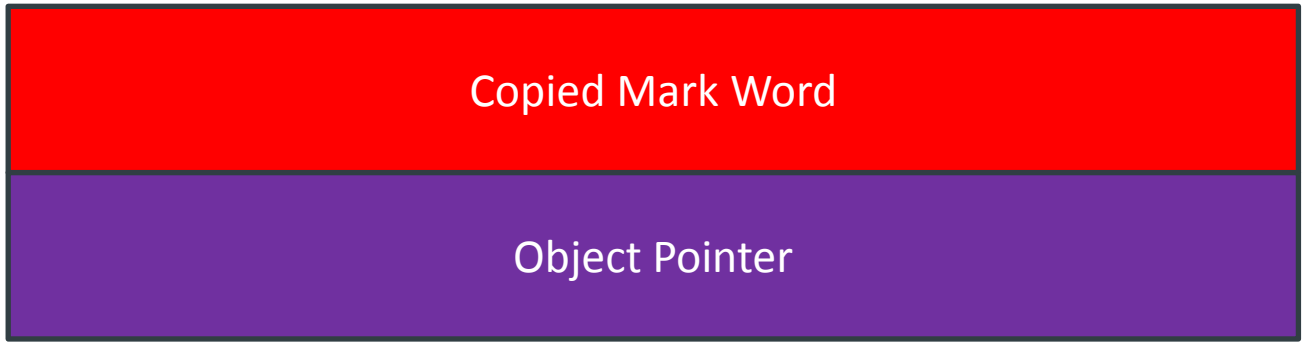
Fat Locked



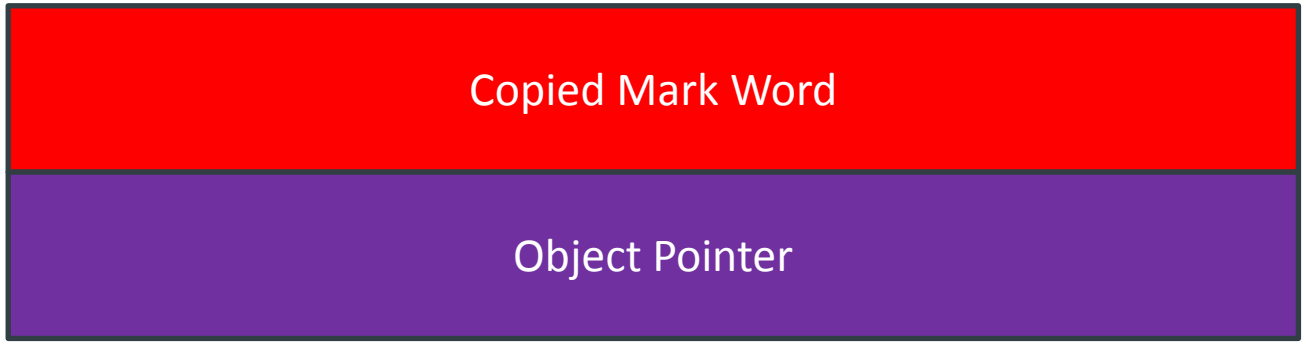
GC Running（STW）

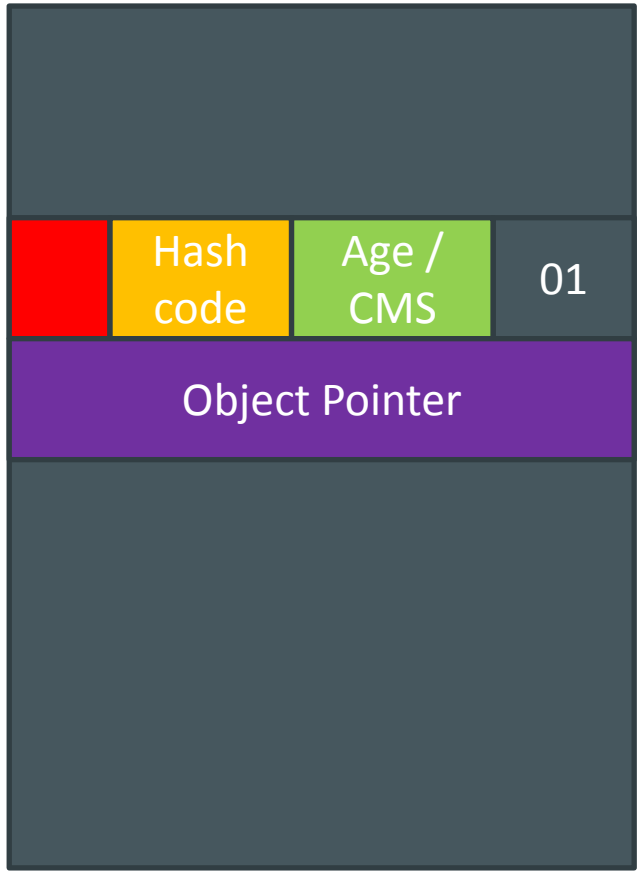


Lock Record



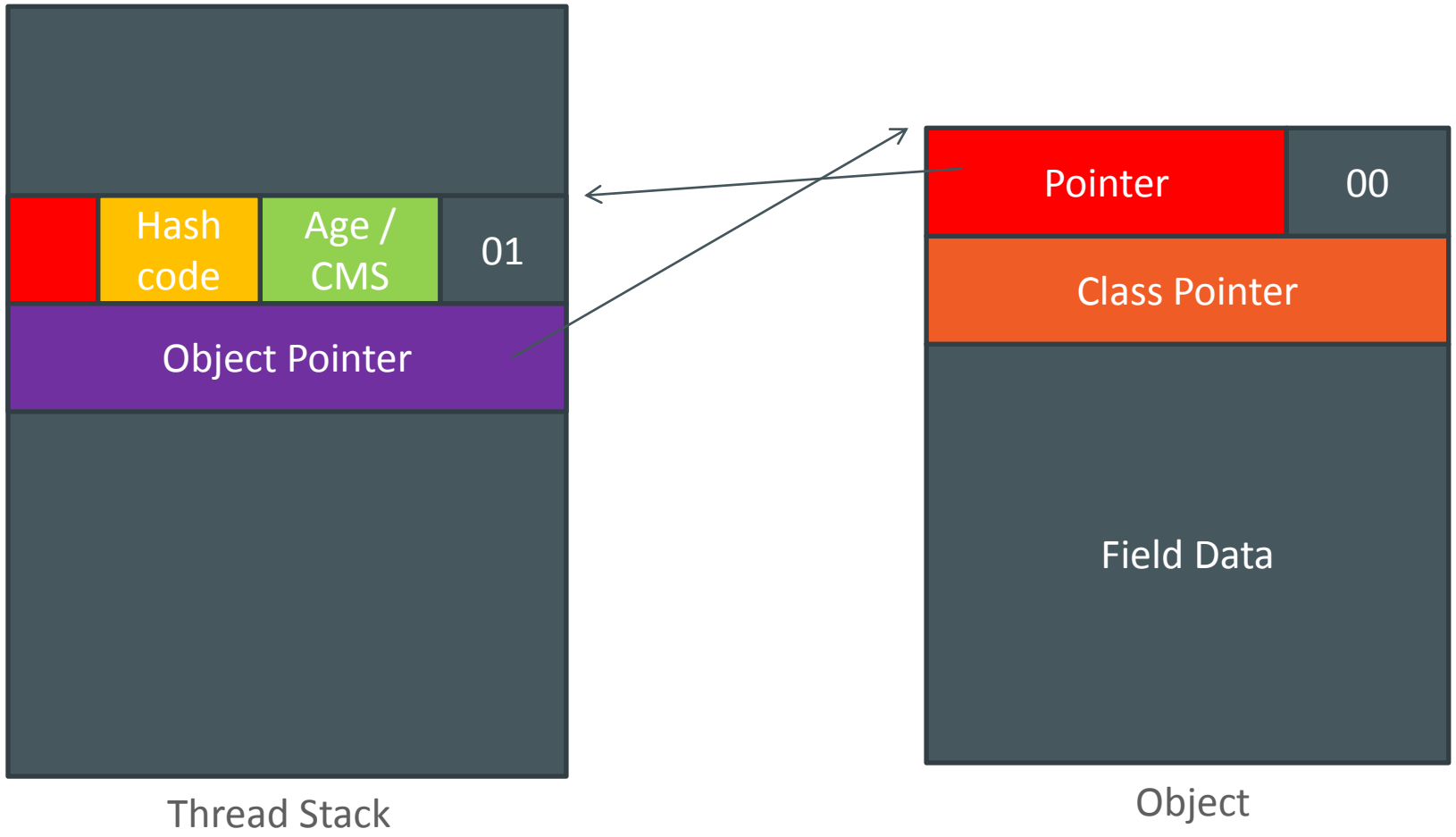
Lock Record



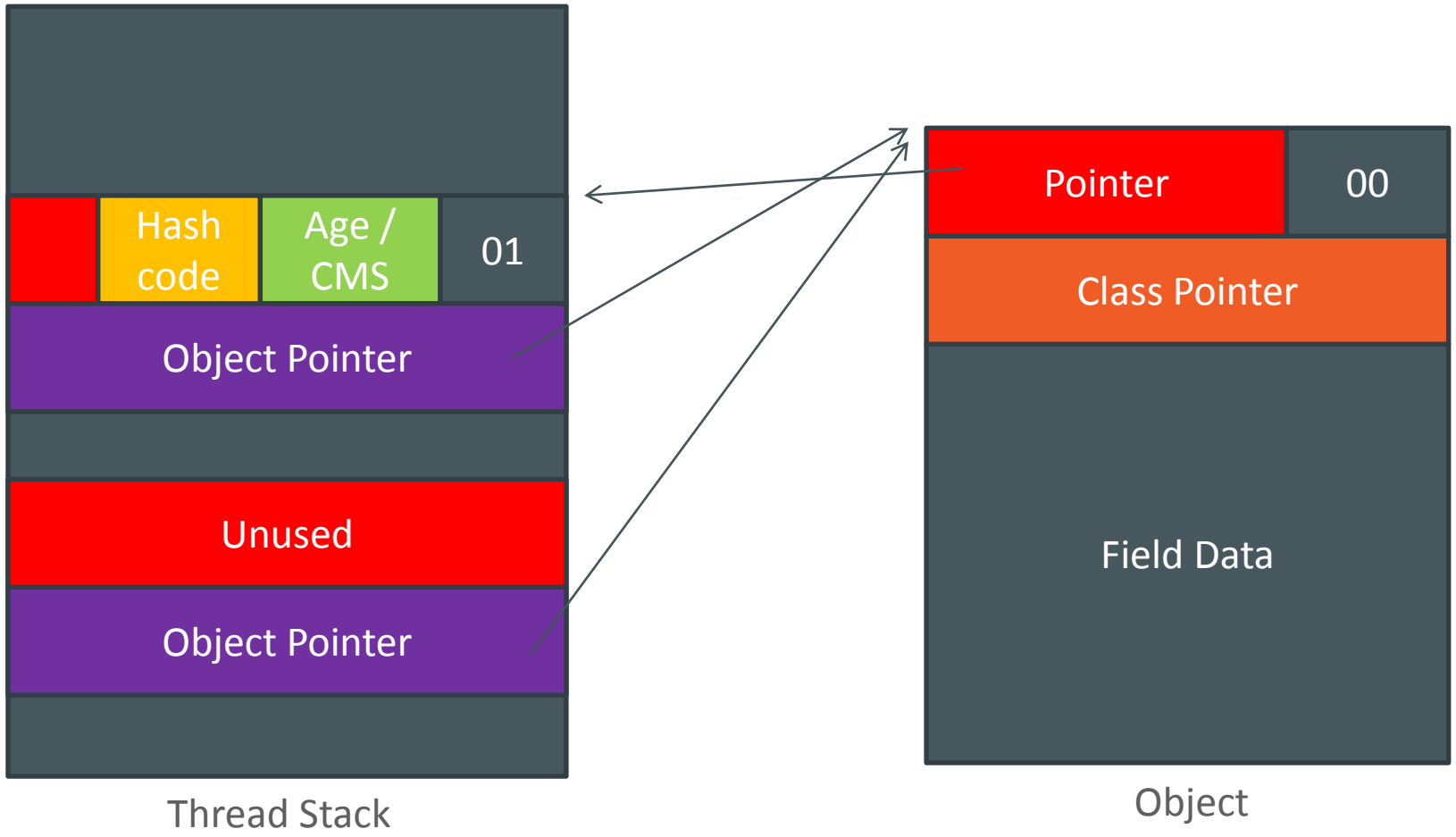


Thread Stack

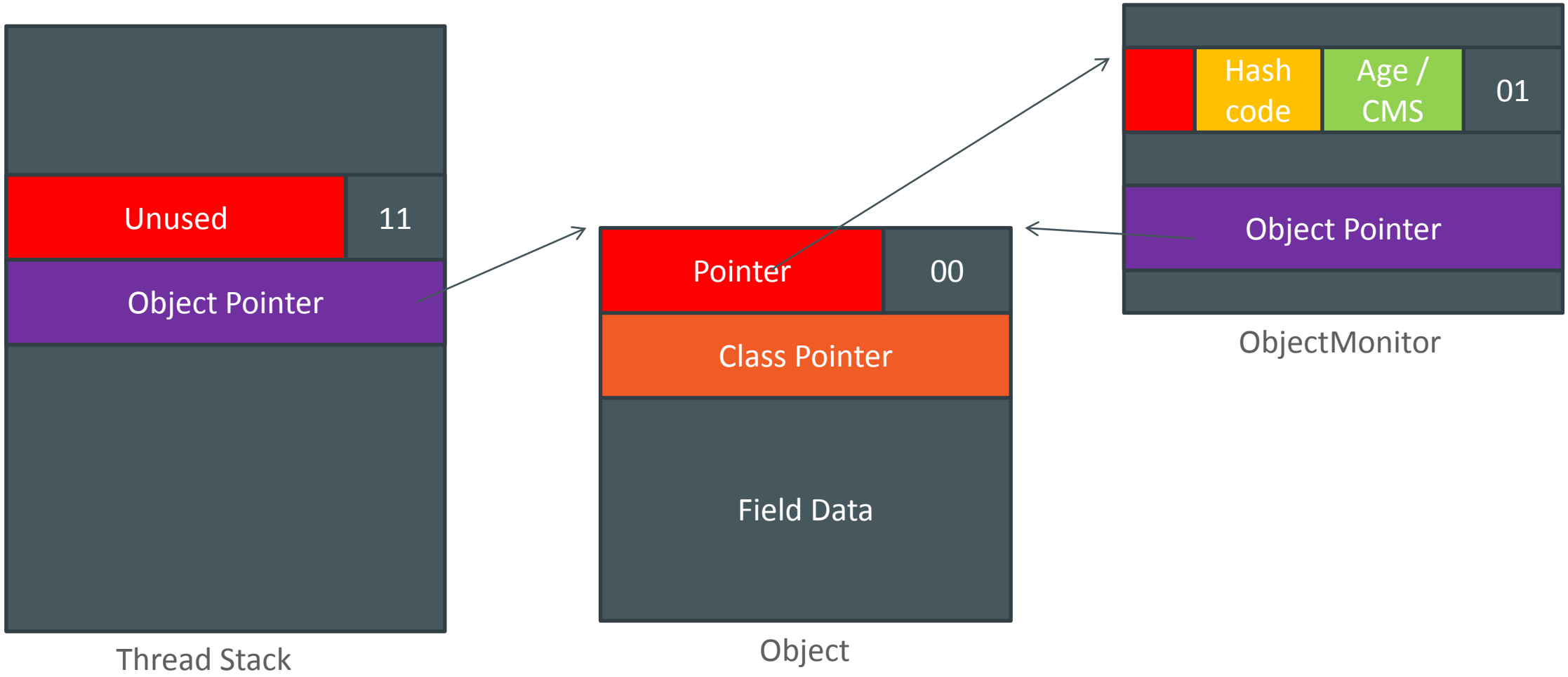
Thin Lock



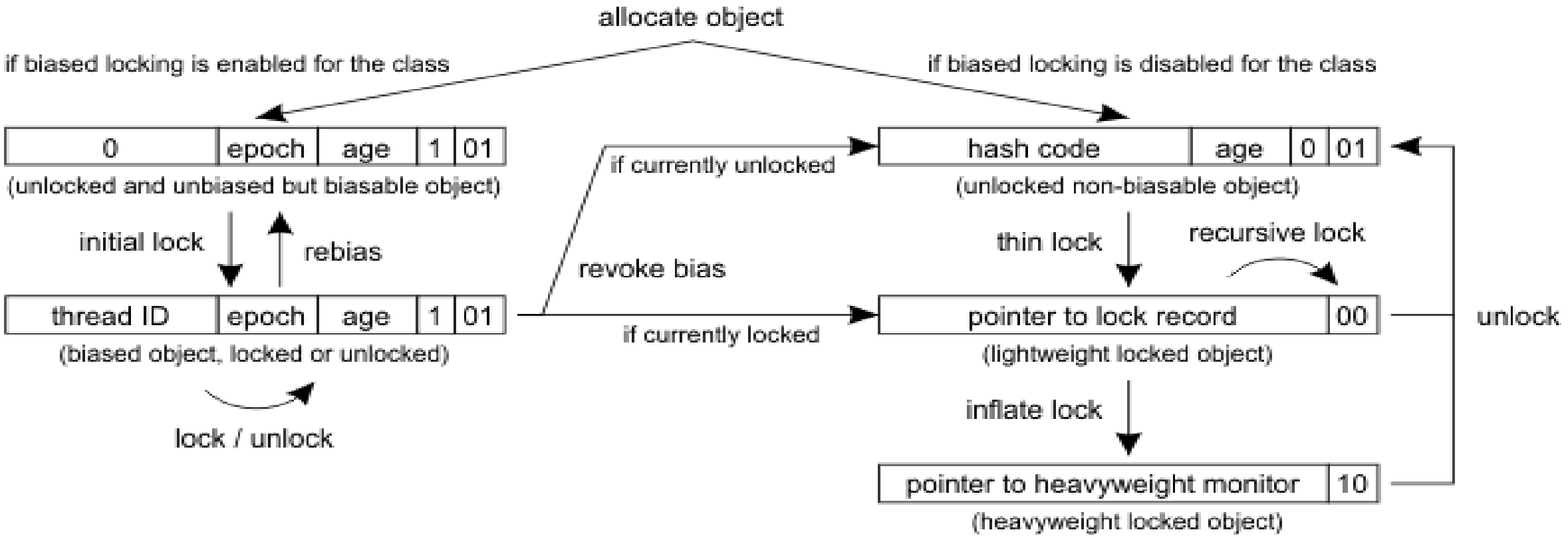
Thin Lock (Recursive)

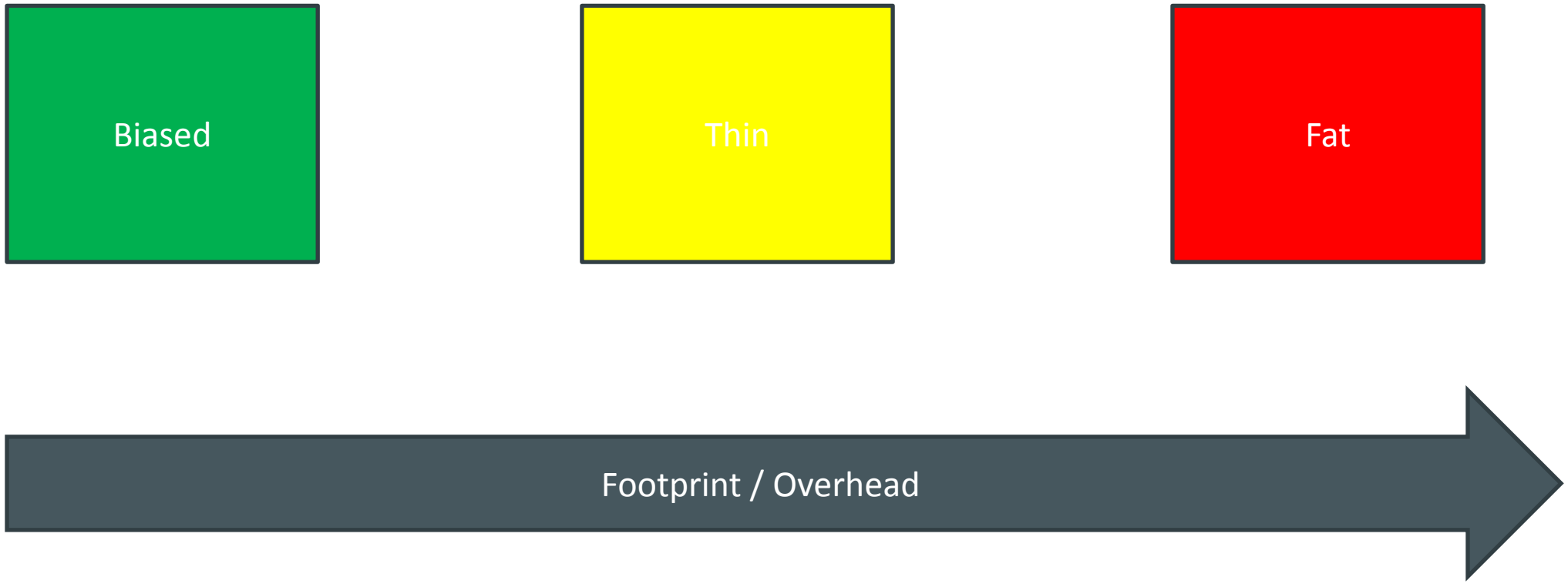


Fat Lock



Lock Transitions





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!

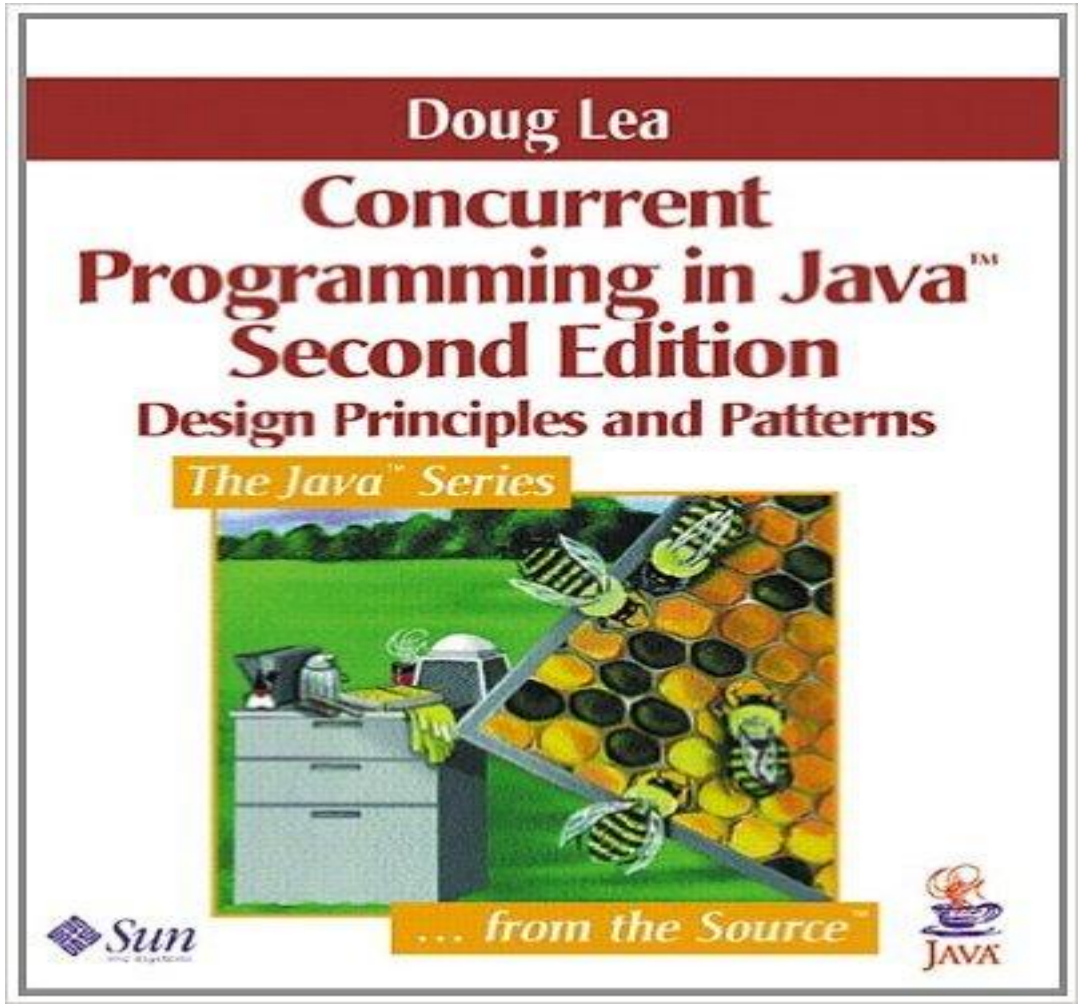
UseBiasedLocking

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

Everything Else

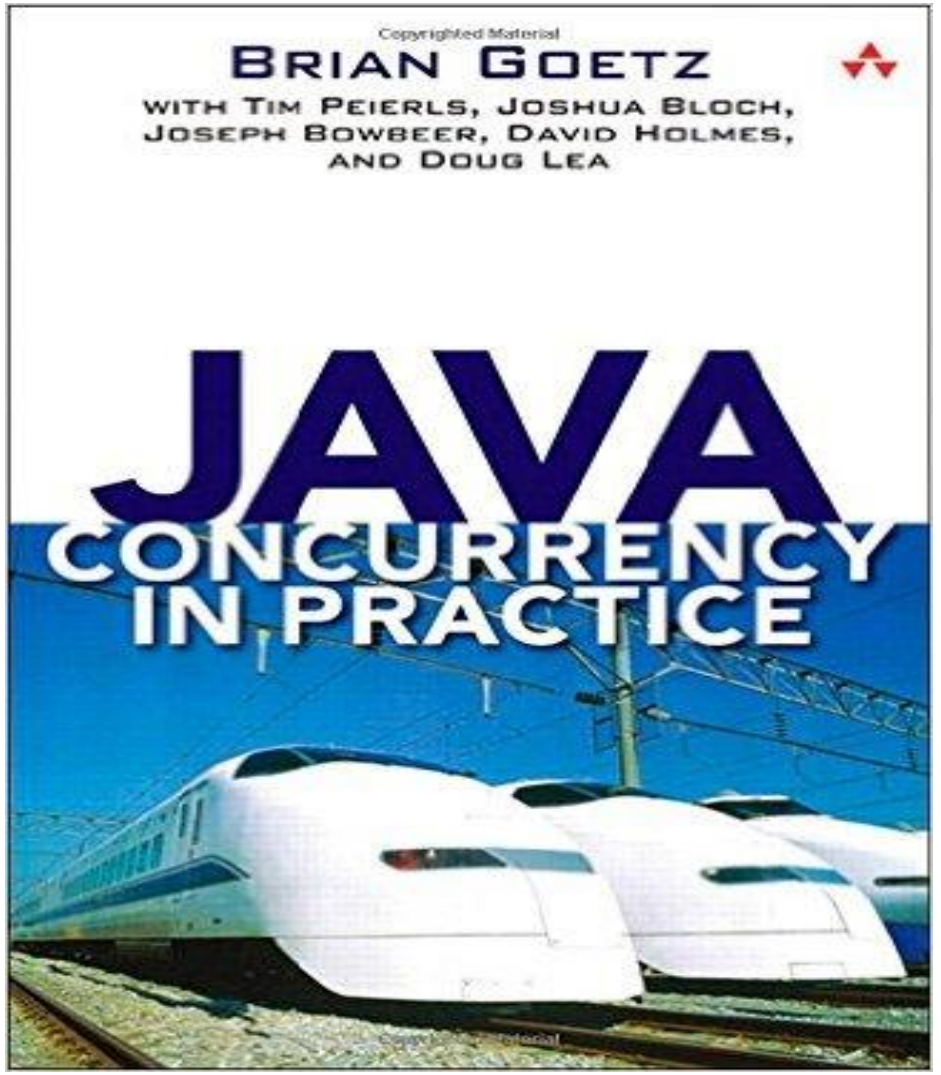
Concurrent Programming in Java™: Design Principles and Patterns

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



Java Concurrency in Practice

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



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®