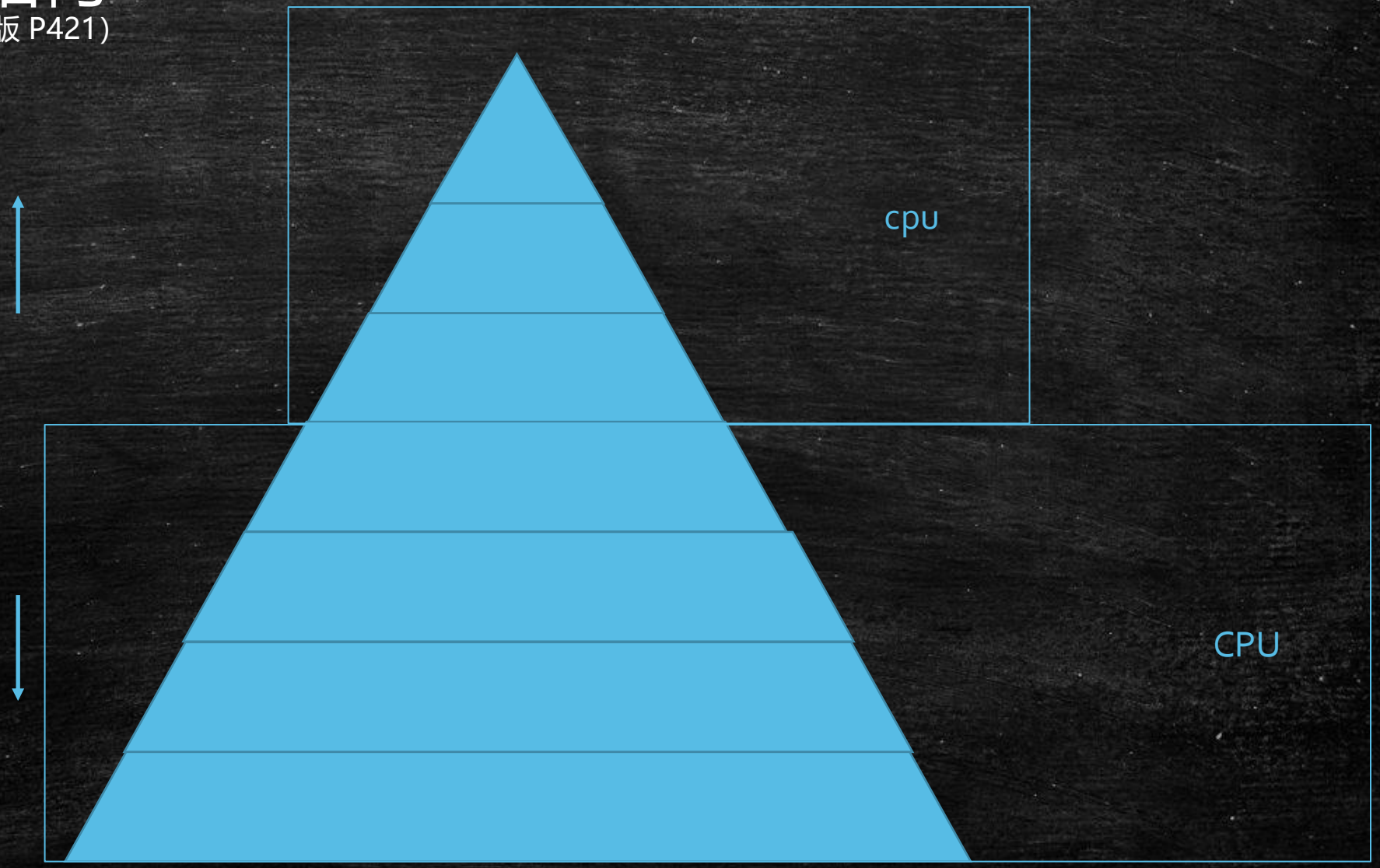
JMM

马士兵

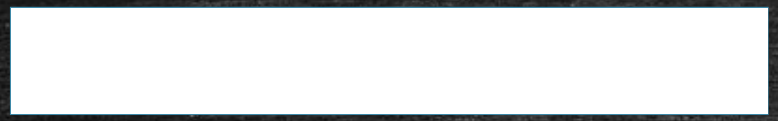
存储器的层次结构

（深入理解计算机系统 原书第三版 P421）



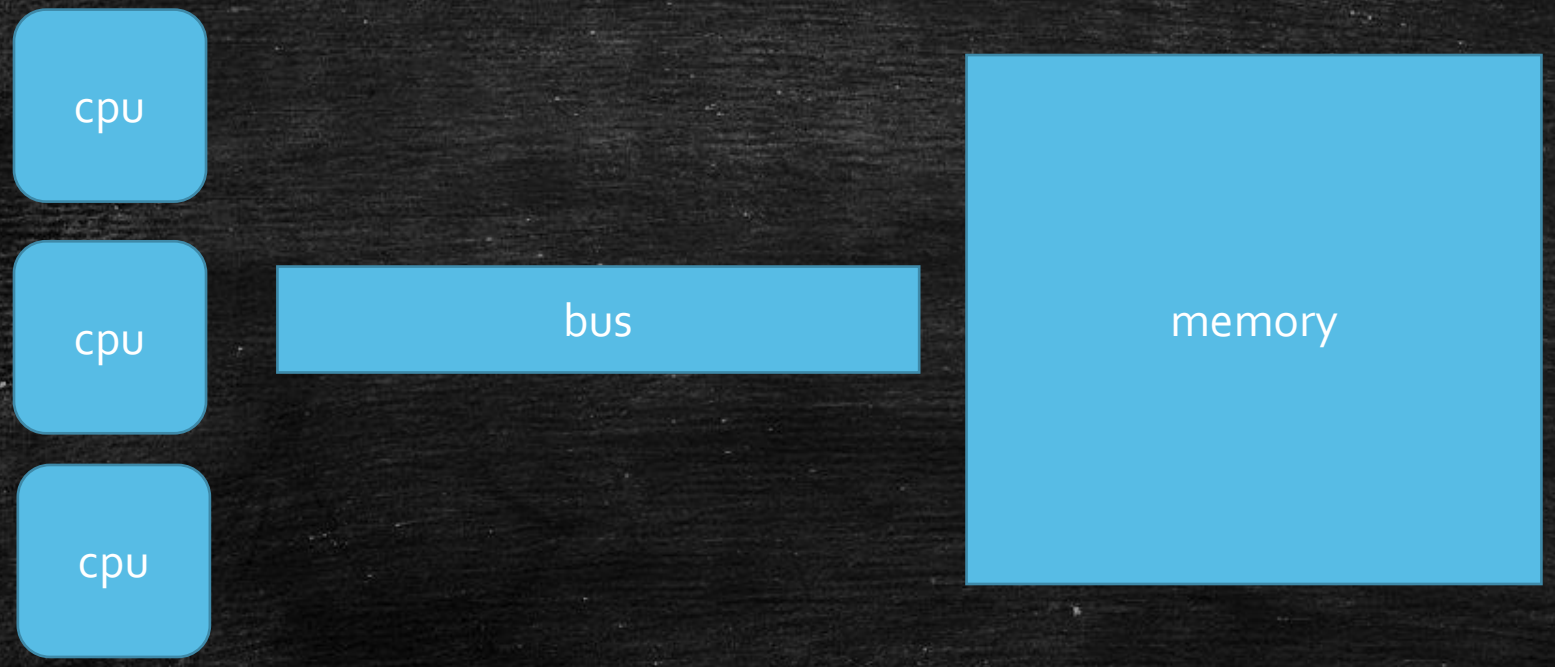
从CPU到	大约需要的 CPU 周期	大约需要的时间
主存		约60-80纳秒
QPI 总线传输 (between sockets, not drawn)		约20ns
L3 cache	约40-45 cycles,	约15ns
L2 cache	约10 cycles,	约3ns
L1 cache	约3-4 cycles,	约1ns
寄存器	1 cycle	

cache line



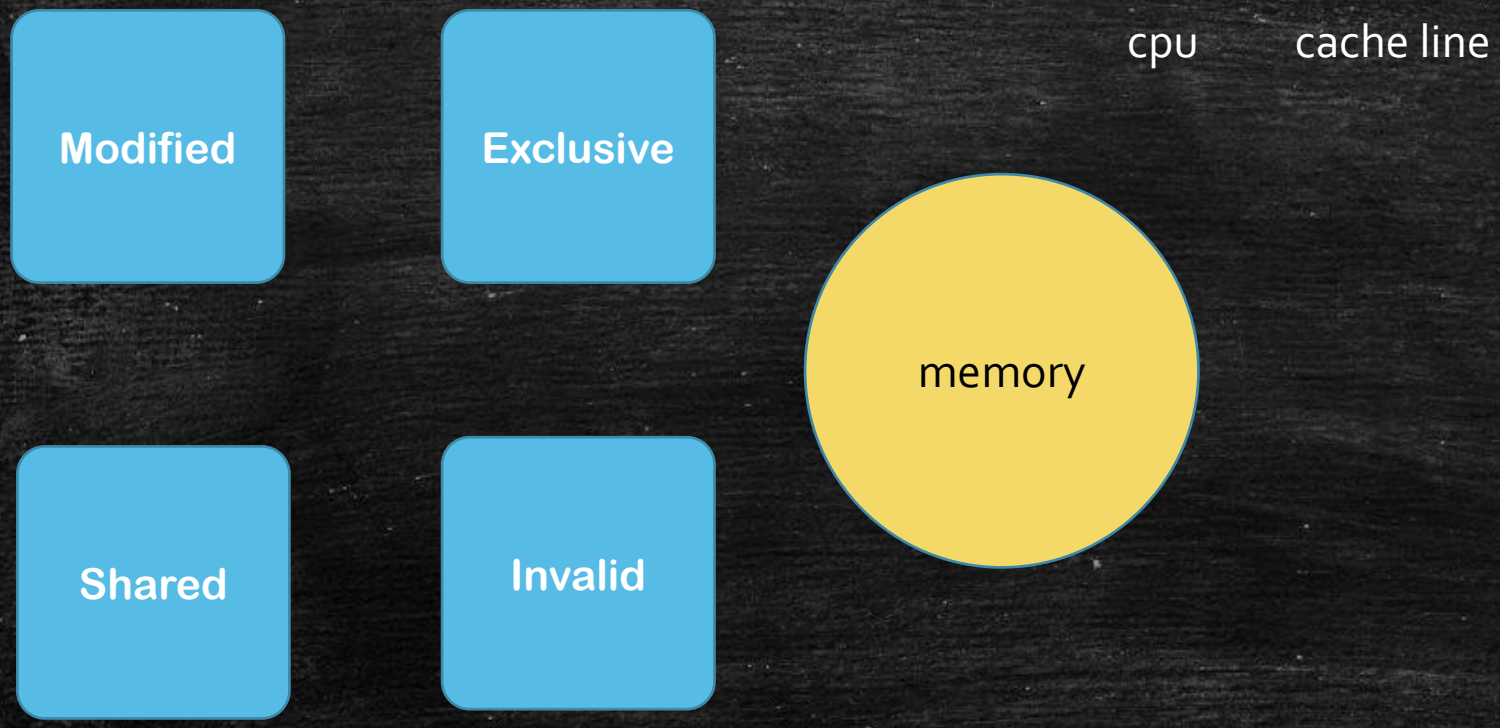
disruptor

```
public long p1, p2, p3, p4, p5, p6, p7; // cache line padding
    private volatile long cursor = INITIAL_CURSOR_VALUE;
    public long p8, p9, p10, p11, p12, p13, p14; // cache line padding
```



CPU

https://www.cnblogs.com/z00377750/p/9180644.html



MSI MESI MOSI Synapse Firefly Dragon

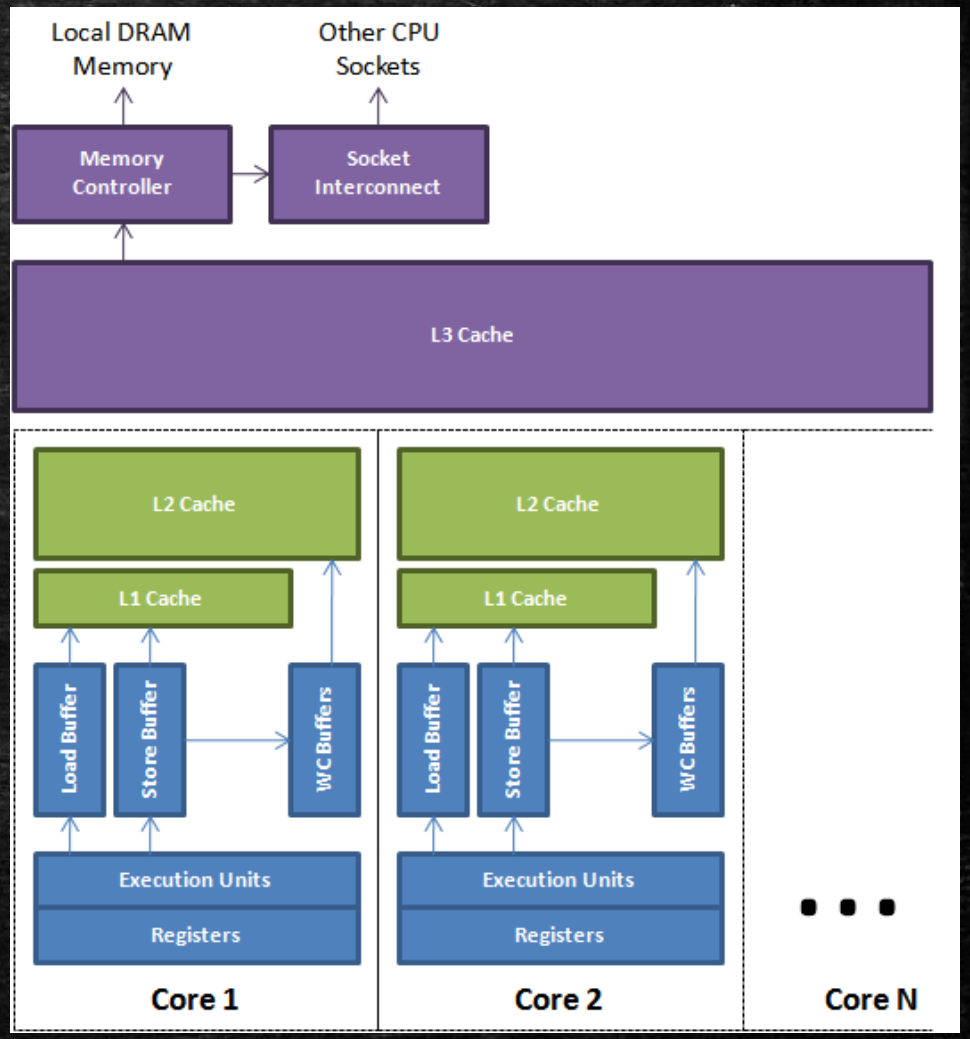
cpu
cpu
https://www.cnblogs.com/liushaodong/p/4777308.html

WCBuffer

CPU

Memory Barrier

volatile windows lock



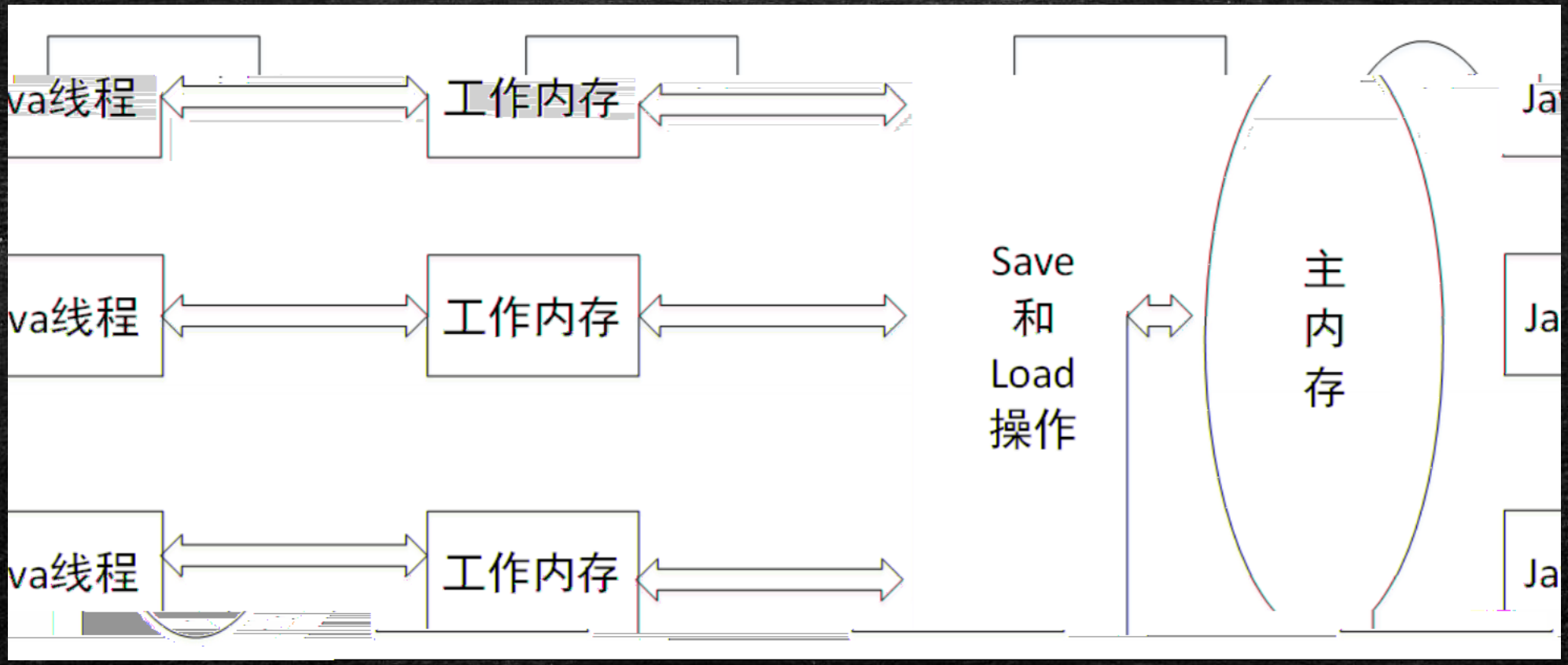
jmm/Disorder.java

https://preshing.com/20120515/memory-reordering-caught-in-the-act/

```
"C:\Program Files\Java\jdk1.8.0_181\bin\java.exe" ...
第2728842次 (0,0)

Process finished with exit code 0
```

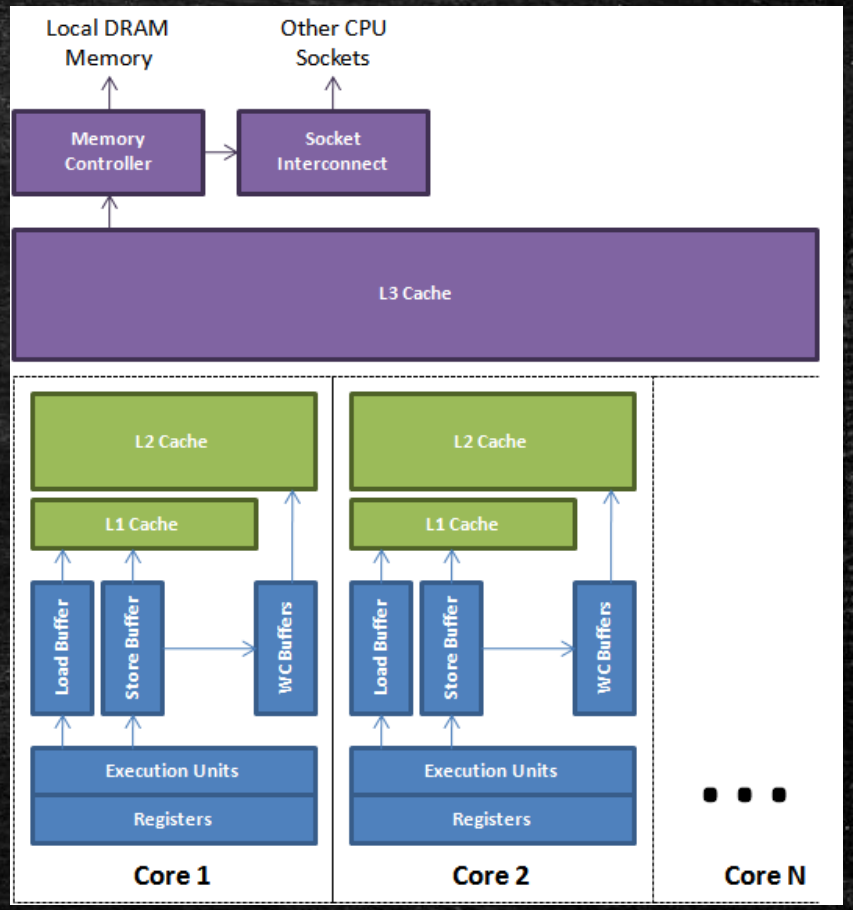
```
"C:\Program Files\Java\jdk1.8.0_181\bin\java.exe" ...
第113299次 (0,0)

Process finished with exit code 0
```

java



cpu

https://www.cnblogs.com/liushaodong/p/4777308.html



JUC/028_WriteCombining

X86 CPU

sfence: sfence sfence
lfence lfence lfence
mfence mfence mfence

intel lock

x86 lock …” Full

Barrier

CPU Software Locks

JSR

LoadLoad

Load1; LoadLoad; Load2

Load2 Load1

StoreStore

Store1; StoreStore; Store2

Store2 Store1

LoadStore

Load1; LoadStore; Store2

Store2 Load1

StoreLoad Store1; StoreLoad; Load2

Load2 Store1

volatile

jmm/TestVolatile.java

volatile int j

Name:	cp_info #6	<j>
Descriptor:	cp_info #5	<I>
Access flags:	0x0040 [volatile]	

volatile
JVM

StoreStoreBarrier
volatile
StoreLoadBarrier

LoadLoadBarrier
volatile
LoadStoreBarrier

volatile

hsdis

lock xxx xxx

https://blog.csdn.net/qq_26222859/article/details/52235930

synchronized

monitor enter

monitor exit

synchronized
JVM

C C++

synchronized
CPU

https://blog.csdn.net/21aspnet/article/details/88571740
lock comxchg

java8

JSR-133 JMM

Java P364

lock
unlock
read
load read
use
assign
store write
write

volatile volatile

as if serial

1.
2.
   - JavaAgent_AboutObject.md
3. 体
   - JavaAgent_AboutObject.md
4.
   - https://blog.csdn.net/clover_lily/article/details/80095580
5.
   - GC
6. Object o = new Object
   - JavaAgent_AboutObject.md

1.

1. class loading
2. class linking (verification, preparation, resolution)
3. class initializing
4.
5.
6. <init>
   1.
   2.

markword

markword的结构，定义在markOop.hpp文件：

```
  32 bits:
  --------
  hash:25 ------------>| age:4    biased_lock:1 lock:2 (normal object)
  JavaThread*:23 epoch:2 age:4    biased_lock:1 lock:2 (biased object)
  size:32 ------------------------------------------>| (CMS free block)
  PromotedObject*:29 ---------->| promo_bits:3 ----->| (CMS promoted object)

  64 bits:
  --------
  unused:25 hash:31 -->| unused:1   age:4    biased_lock:1 lock:2 (normal object)
  JavaThread*:54 epoch:2 unused:1   age:4    biased_lock:1 lock:2 (biased object)
  PromotedObject*:61 --------------------->| promo_bits:3 ----->| (CMS promoted object)
  size:64 ----------------------------------------------------->| (CMS free block)

  unused:25 hash:31 -->| cms_free:1 age:4    biased_lock:1 lock:2 (COOPs && normal object)
  JavaThread*:54 epoch:2 cms_free:1 age:4    biased_lock:1 lock:2 (COOPs && biased object)
  narrowOop:32 unused:24 cms_free:1 unused:4 promo_bits:3 ----->| (COOPs && CMS promoted object)
  unused:21 size:35 -->| cms_free:1 unused:7 ------------------>| (COOPs && CMS free block)
 [ptr             | 00]  locked             ptr points to real header on stack
object header
d lock (header is wapped out)
markSweep to mark an object
```

```
  [header      | 0 | 01]  unlocked          regular
  [ptr            | 10]  monitor           inflate
  [ptr            | 11]  marked            used by
```

markword 64

锁状态	25bit		4bit	1bit	2bit
	23bit	2bit		是否偏向锁	锁标志位
无锁态	对象的hashCode		分代年龄	0	01
轻量级锁	指向栈中锁记录的指针				00
重量级锁	指向互斥量(重量级锁)的指针				10
GC标记	空				11
偏向锁	线程ID	Epoch	分代年龄	1	01

如果对象没有重写hashcode方法，那么默认是调用os::random产生hashcode,可以通过System.identityHashCode获取;

os::random产生hashcode的规则为:next_rand = (16807*seed) mod (2*31-1),因此可以使用31位存储;另外一旦生成了hashcode,JVM会将其记录在markword中;

hashcode?　　hashcode()　　System.identityHashCode

GC　　15　　15

Banksteel 工具 Google 翻译 Git在Eclipse中的使用... Redis 设计与实现 (... ThreadLocal终极源码... Redn

当Java处在偏向锁、重量级锁状态时，hashcode值存储在哪？

RednaxelaFX

编译原理 JavaScript 编程 等 7 个话题的优秀回答者

68 人赞同了该回答

这是一个针对HotSpot VM的锁实现的问题。

简单答案是：

- 当一个对象已经计算过identity hash code，它就无法进入偏向锁状态；
- 当一个对象当前正处于偏向锁状态，并且需要计算其identity hash code的话，则它的偏向锁会被撤销，并且锁会膨胀为重量锁；
- 重量锁的实现中，ObjectMonitor类里有字段可以记录非加锁状态下的mark word，其中可以存储identity hash code的值。或者简单说就是重量锁可以存下identity hash code。

需要注意下，当调用锁对象的 `Object#hash` 或 `System.identityHashCode()` 方法会导致该对象的偏向锁或轻量级锁升级。这是因为在Java中一个对象的hashcode是在调用这两个方法时才生成的，如果是无锁状态则存放在 `mark word` 中，如果是重量级锁则存放在对应的monitor中，而偏向锁是没有地方能存放该信息的，所以必须升级。

https://cloud.tencent.com/developer/article/1480590
https://cloud.tencent.com/developer/article/1484167
https://cloud.tencent.com/developer/article/1485795
https://cloud.tencent.com/developer/article/1482500

Synchronized和ReentrantLock的区别

原理弄清楚了，顺便总结了几点Synchronized和ReentrantLock的区别：

1. Synchronized是JVM层次的锁实现，ReentrantLock是JDK层次的锁实现；
2. Synchronized的锁状态是无法在代码中直接判断的，但是ReentrantLock可以通过 `ReentrantLock#isLocked` 判断；
3. Synchronized是非公平锁，ReentrantLock是可以是公平也可以是非公平的；
4. Synchronized是不可以被中断的，而 `ReentrantLock#lockInterruptibly` 方法是可以被中断的；
5. 在发生异常时Synchronized会自动释放锁（由javac编译时自动实现），而ReentrantLock需要开发者在finally块中显示释放锁；
6. ReentrantLock获取锁的形式有多种：如立即返回是否成功的tryLock(),以及等待指定时长的获取，更加灵活；