马士兵教育
www.mashibing.com

多线程与高并发（第二版）

马士兵

http://mashibing.com

版本

- 1.0 常见线程问题
  更新于腾讯课堂
- 2.0 梳理了顺序和思路，循序渐进，从基础开始
  增加很多1.0中没有讲过的内容
  更新于自有APP

这里是线程进阶

- 复习讲的内容
  - 建立线程
  - 互斥锁synchronized
- 进阶内容
  - 基础课程汇总未讲的内容
  - 底层剖析
  - 源码解析
  - 容易混淆的概念
  - 大厂面试解析
  - 应用场景解析
  - ...

- 前置知识
  - 0基础线程部分
  - lamda表达式 stream API

多线程内容的特点及学习方法

- 面试重灾区
- 琐碎
- 学习方法：
  - 时间不充裕 – 背
  - 时间充裕 – 慢慢自己做实验 + 背

从最基本的线程谈起

- 线程是什么？
  - 形象化：
  - 从底层角度：
- 面试题：什么是进程？什么是线程？什么是纤程/协程？
- 是不是线程数量越多，效率就越高？
- 线程在底层的执行（乱序 as-if-serial）

从一则招聘谈起

- 某大厂的一则招聘（P6-P7）

1. Java基础扎实，熟悉JVM、多线程、集合等基础，熟悉分布式、缓存、消息、搜索等机制
2. 三年以上Java开发经验，熟悉Spring、MyBatis等框架
3. 对于压榨CPU性能有浓厚兴趣！
4. 具有一定项目规划和决策能力，善于捕捉业务需求、系统架构设计中存在的问题，并给出有效的解决方案
5. 具有高度领域设计能力和业务分析能力，能独立分析和解决问题

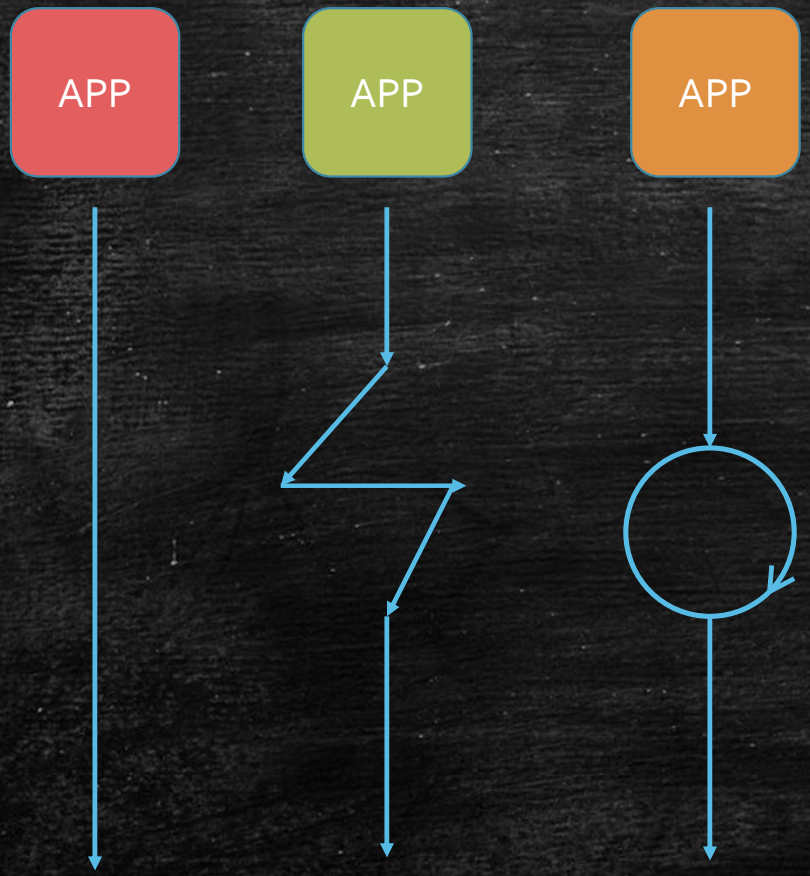
线程的历史 – 一部对于CPU性能压榨的历史

- 单进程人工切换
  - 纸带机
- 多进程批处理
  - 多个任务批量执行
- 多进程并行处理
  - 把程序写在不同的内存位置上来回切换
- 多线程
  - 一个程序内部不同任务的来回切换
  - selector - epoll
- 纤程/协程
  - 绿色线程，用户管理的（而不是OS管理的）线程

面试题：

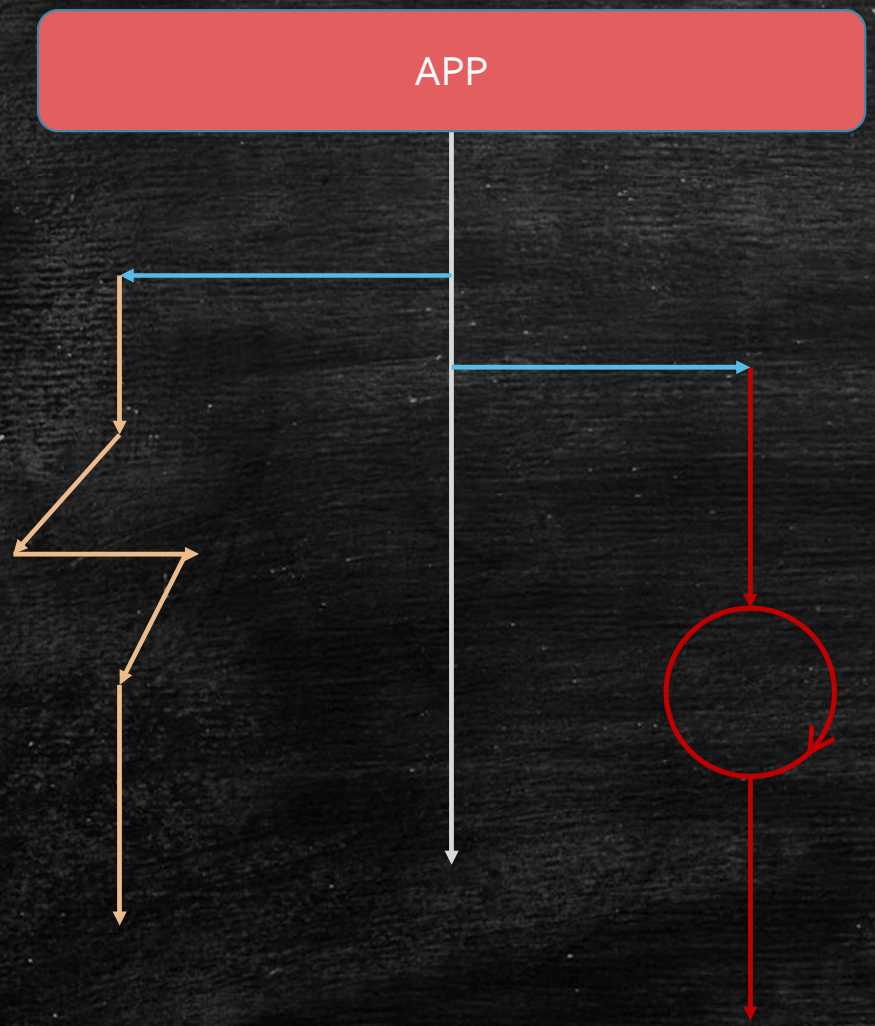
- 什么是进程?
- 什么是线程?
- 什么是纤程/协程?
- 什么是程序?

从通俗的角度理解



单线程

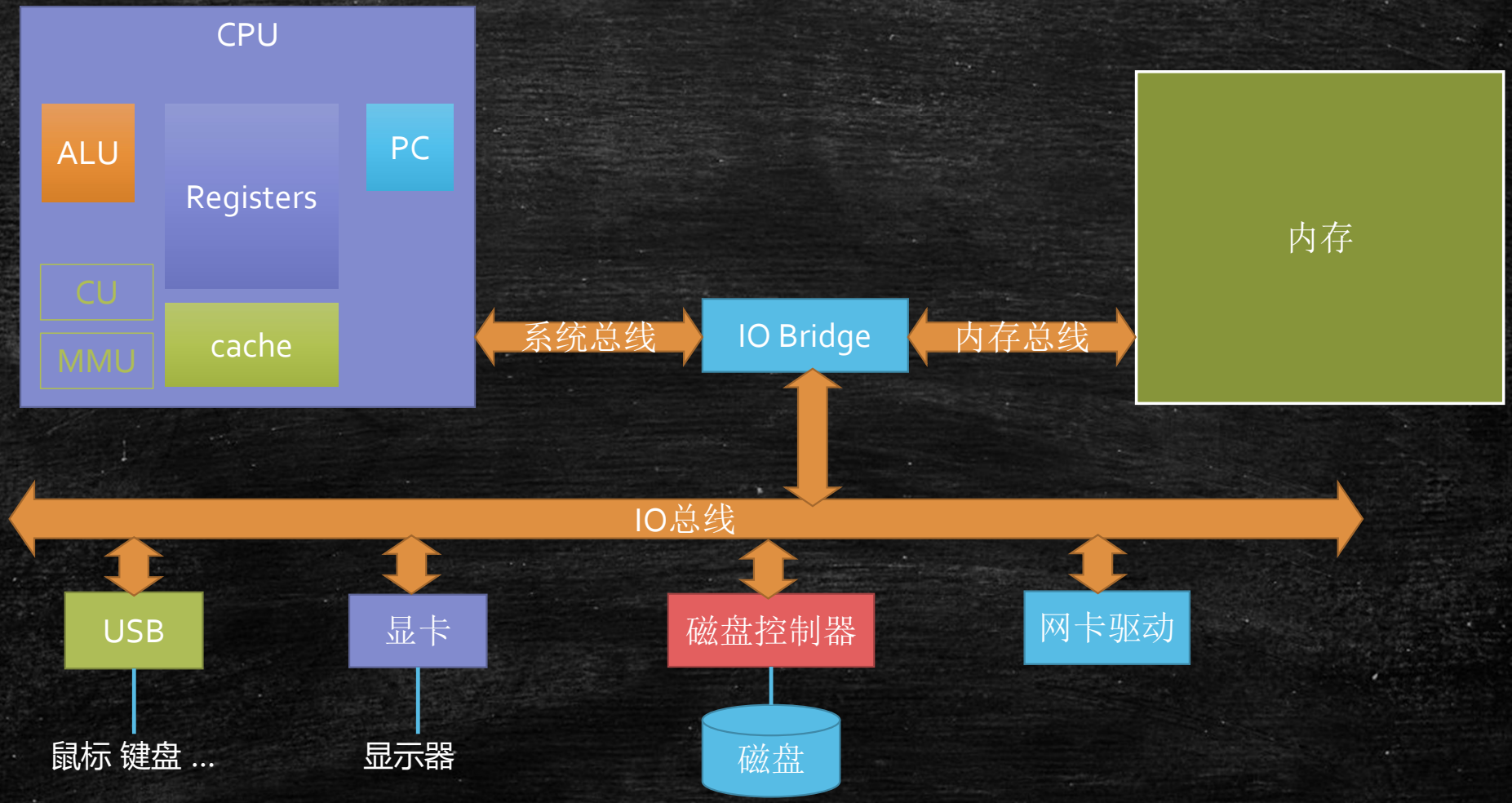
不同的执行路径



多线程

从专业的角度理解

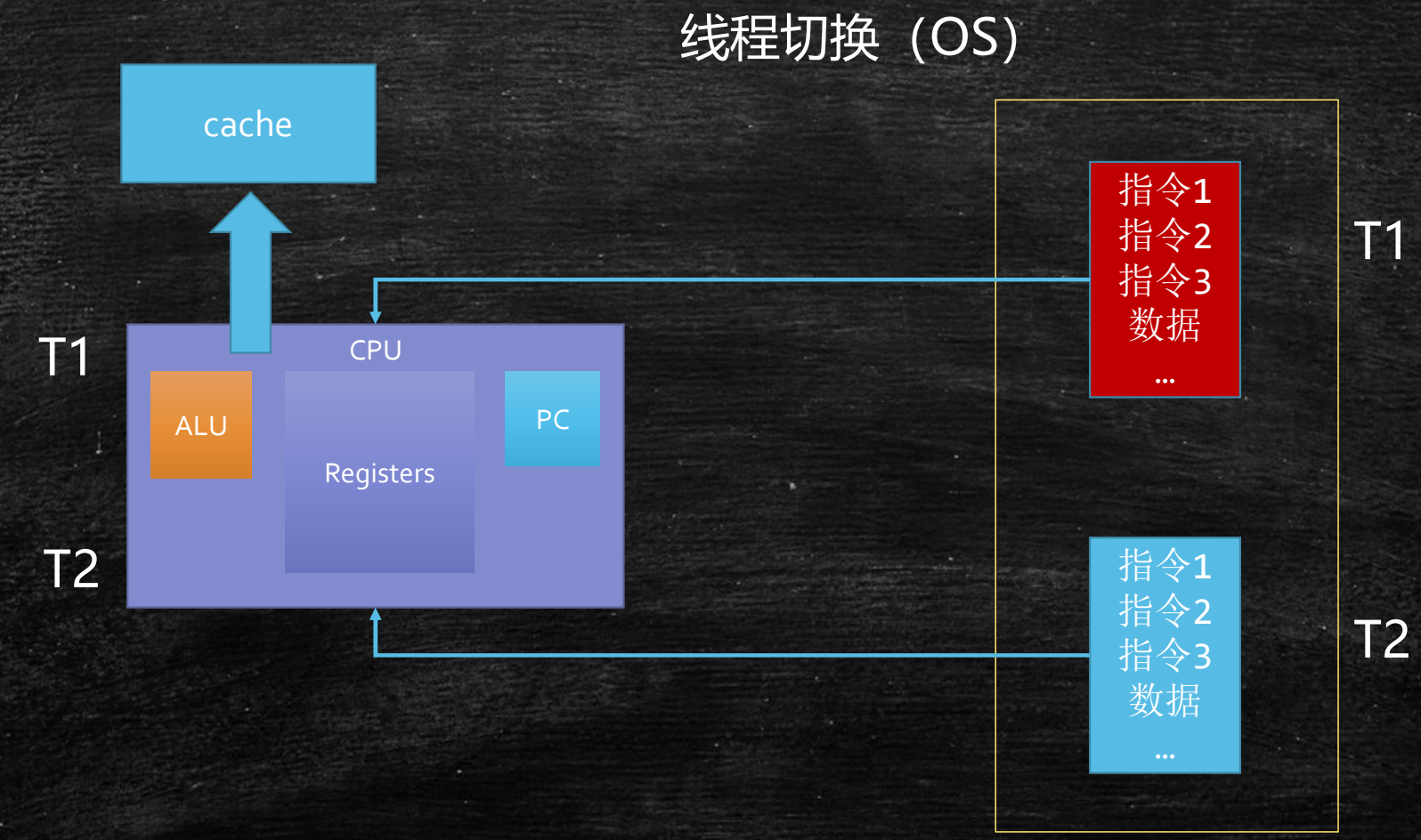
计算机的组成



http://mashibing.com

从底层角度：

从底层角度：


线程切换（OS）
cache
T1
CPU
ALU
Registers
PC
T2
指令1
指令2
指令3
数据
...
T1
指令1
指令2
指令3
数据
...
T2

1. 单核CPU设定多线程是否有意义？
2. 工作线程数是不是设置的越大越好？
3. 工作线程数（线程池中线程数量）设多少合适？

$N_{threads} = N_{CPU} * U_{CPU} * (1 + W/C)$

其中：

- N_{CPU}是处理器的核的数目，可以通过`Runtime.getRuntime().availableProcessors()`得到
- U_{CPU}是期望的CPU利用率（该值应该介于0和1之间）
- W/C是等待时间与计算时间的比率

线程撕裂者

线程的“打断”

- interrupt()
  - 打断某个线程（设置标志位）
- isInterrupted()
  - 查询某线程是否被打断过（查询标志位）
- static interrupted()
  - 查询当前线程是否被打断过，并重置打断标志

线程的“结束”

面试题：如何优雅的结束一个线程？

e.g. 上传一个大文件，正在处理费时的计算
如何优雅的结束这个线程？

什么是单线程的乱序执行和as-if serial？

as – if serial？看上去像是序列化执行
不影响单线程的最终一致性

并发编程三大特性

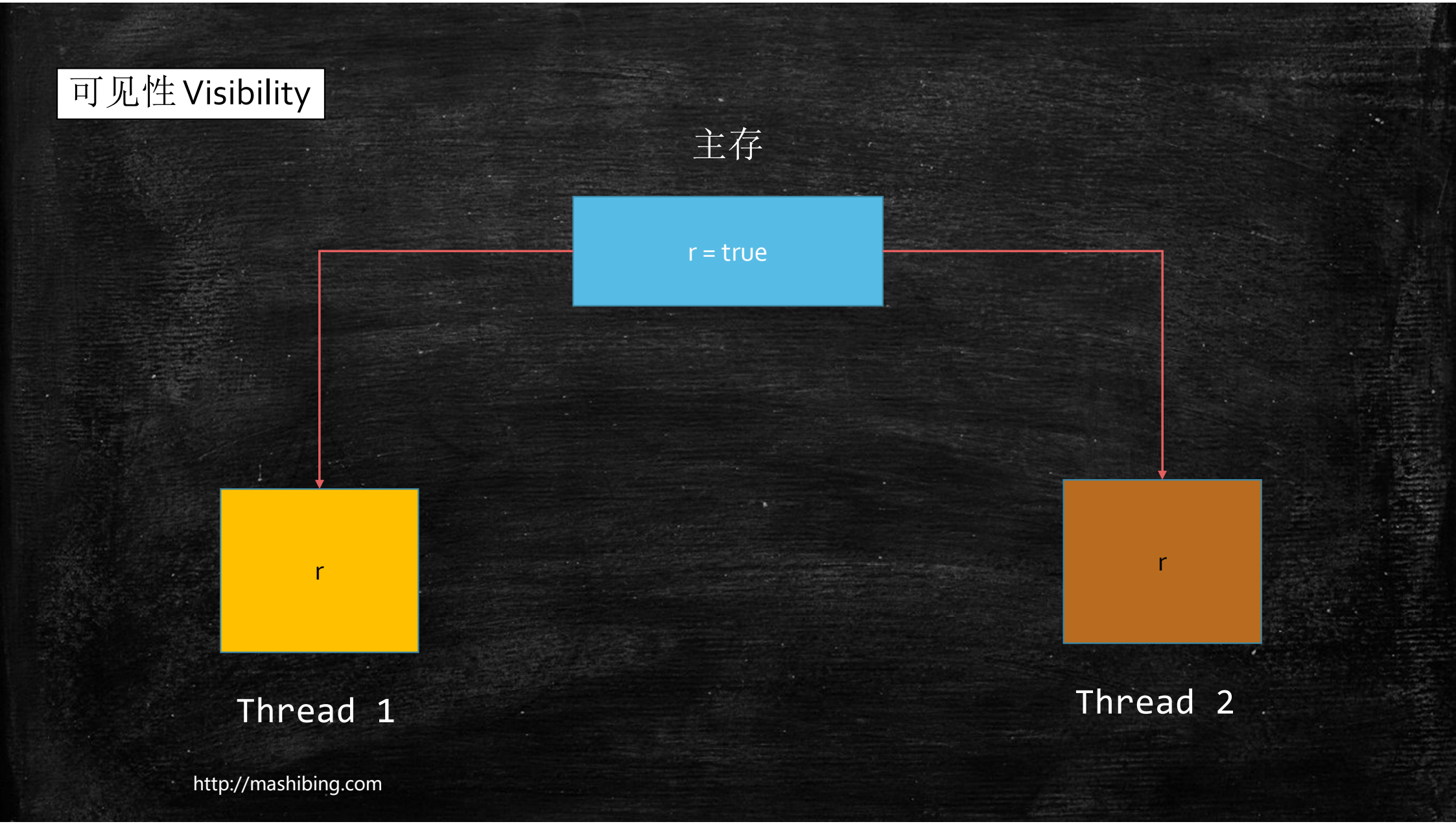
- 可见性（visibility)
- 有序性（ordering)
- 原子性（atomicity)

面试重中之重

务必牢牢掌握

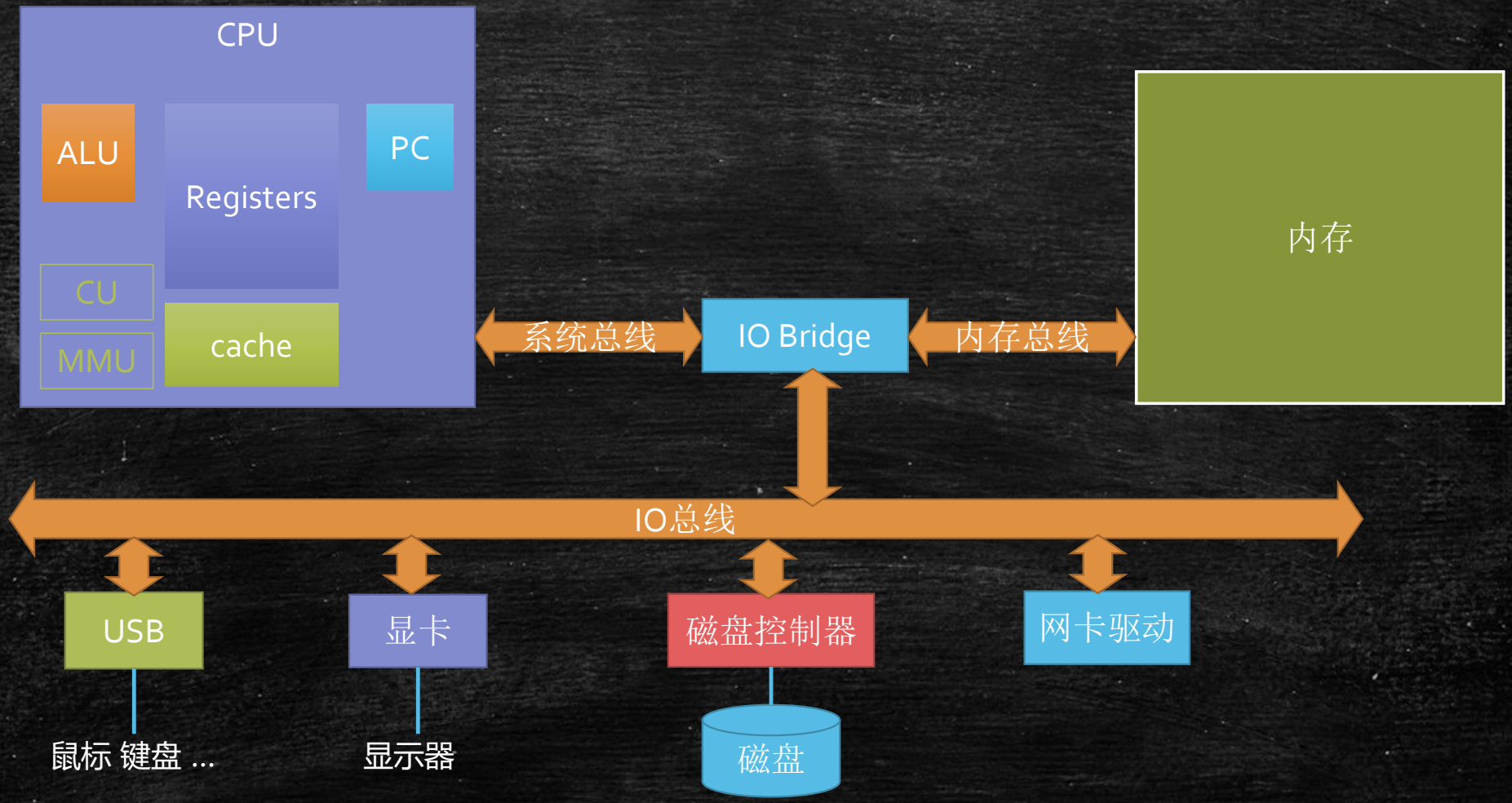
http://mashibing.com

可见性 Visibility



http://mashibing.com

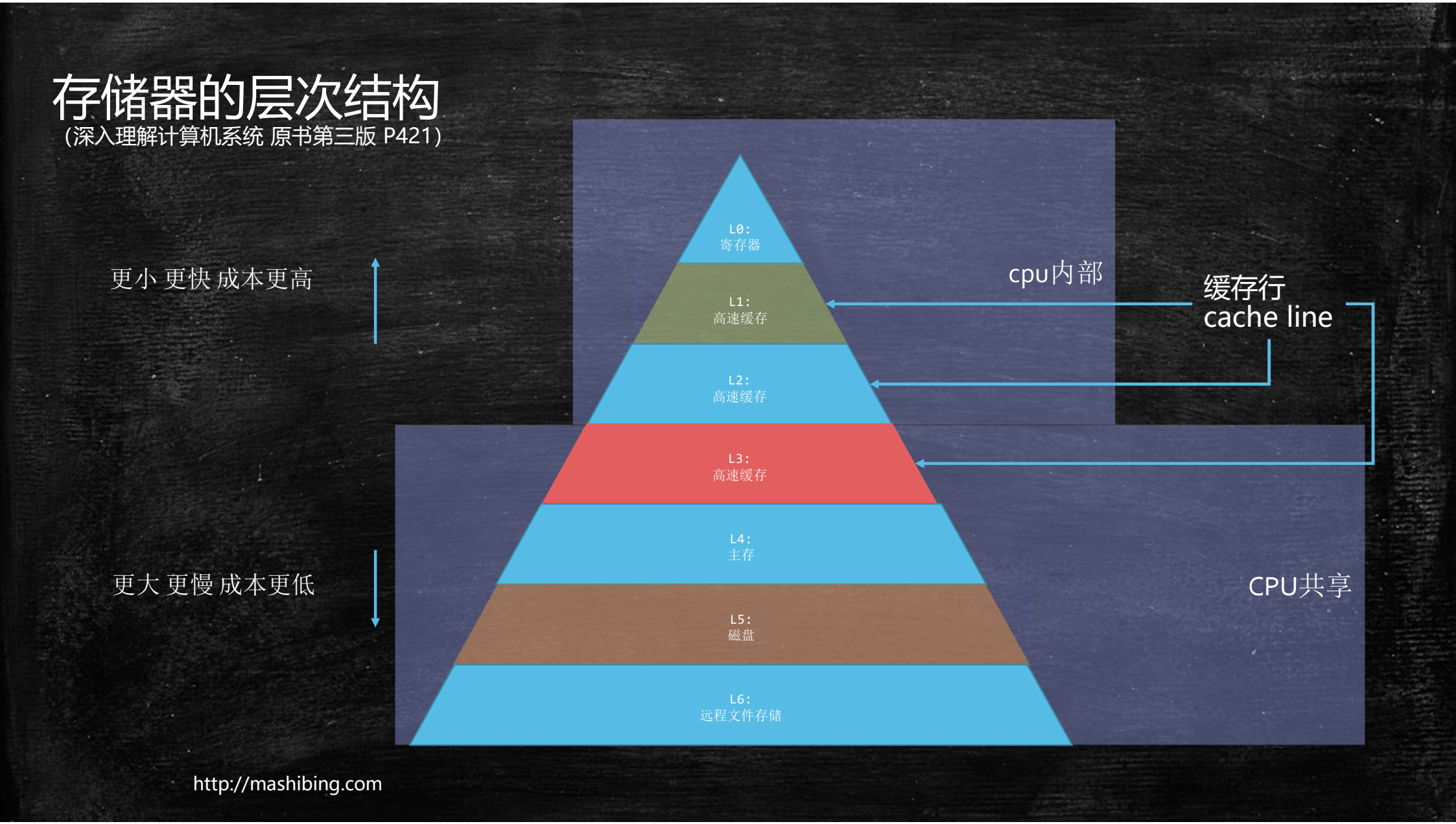
计算机的组成



http://mashibing.com

存储器的层次结构

(深入理解计算机系统 原书第三版 P421)



http://mashibing.com

从CPU的计算单元（ALU）到:

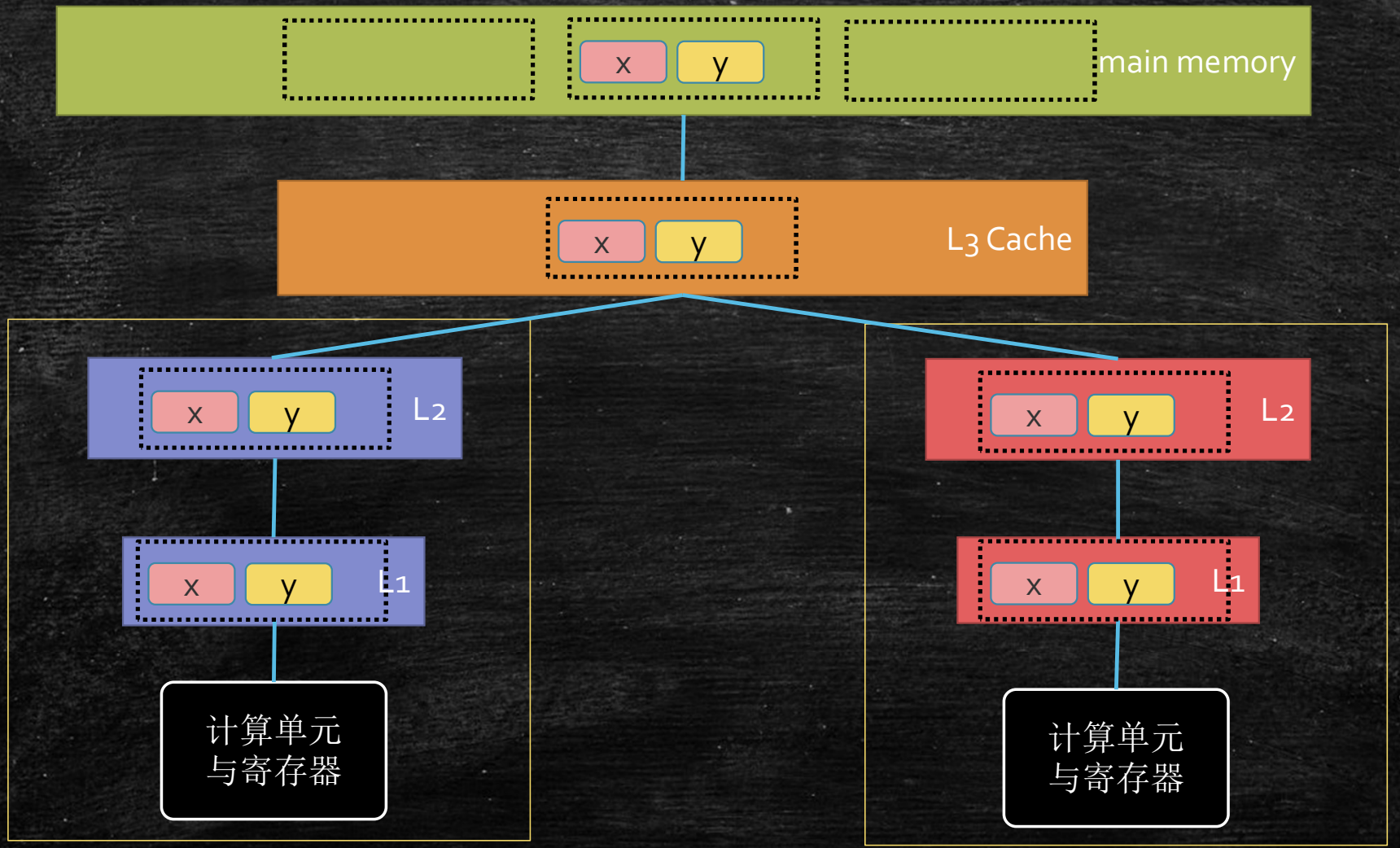
Registers	< 1ns
L1 cache	约1ns
L2 cache	约3ns
L3 cache	约15ns
main memory	约80ns

多核CPU

http://mashibing.com

按块读取
程序局部性原理，可以提高效率
充分发挥总线 CPU针脚等一次性读取更多数据的能力

cache line的概念 缓存行对齐 伪共享

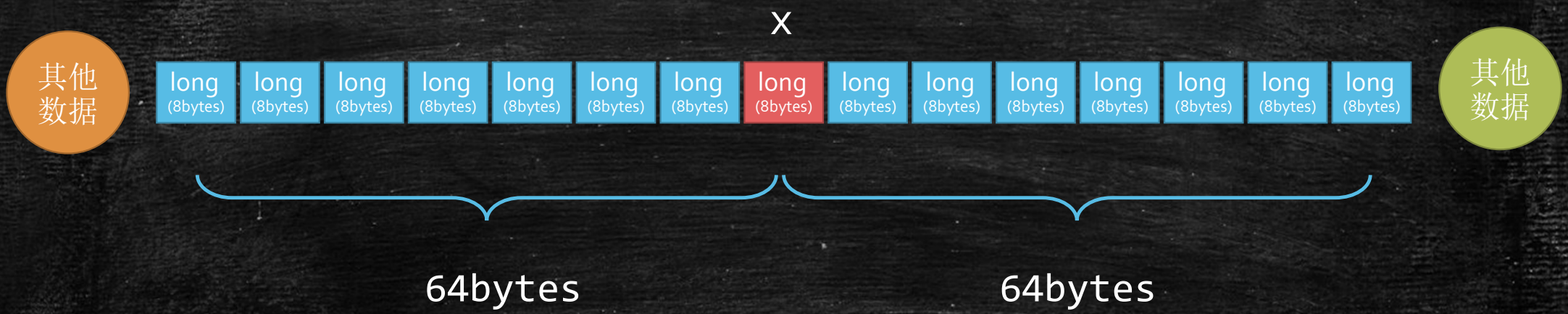


http://mashibing.com

缓存行填充的编程技巧

disruptor RingBuffer的实现

JDK中 LinkedBlockingQueue 1.7



任意其他有效数据都不可能和X位于同一行！
减少了缓存一致性的开销！

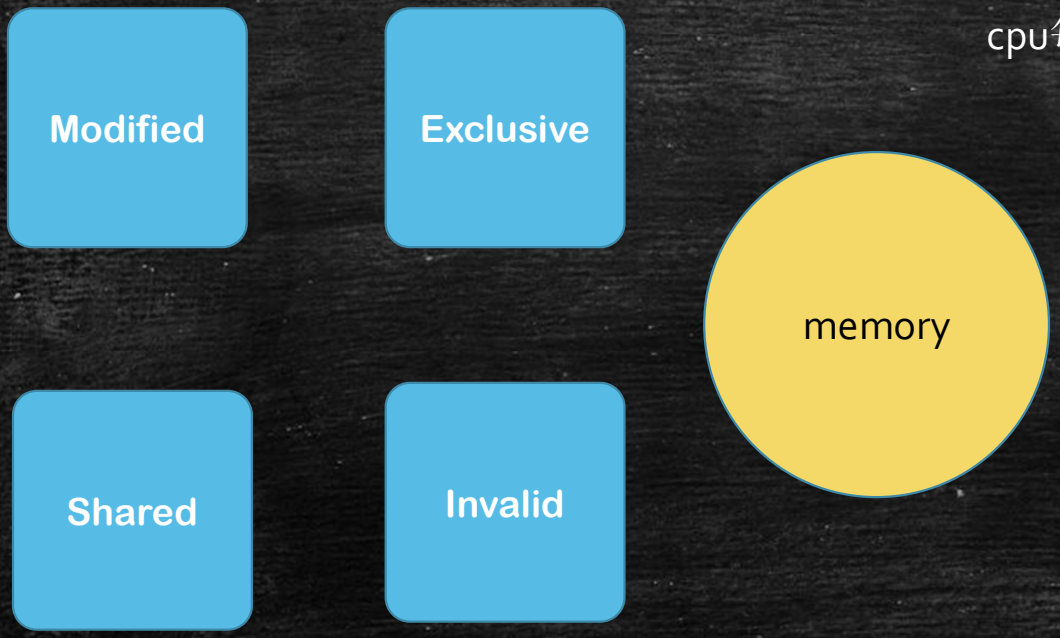
http://mashibing.com

缓存一致性协议

和volatile无关

MSI MESI MOSI Synapse Firefly DragonProtocol ...

MESI Cache一致性协议



cpu每个cache line标记四种状态（额外两位）

缓存锁实现之一
有些无法被缓存的数据
或者跨越多个缓存行的数据
依然必须使用总线锁/缓存锁

缓存行：
缓存行越大，局部性空间效率越高，但读取时间慢
缓存行越小，局部性空间效率越低，但读取时间快
取一个折中值，目前多用：
64字节

阶段小结

- volatile保障线程可见性
- 缓存行
- 缓存一致性协议

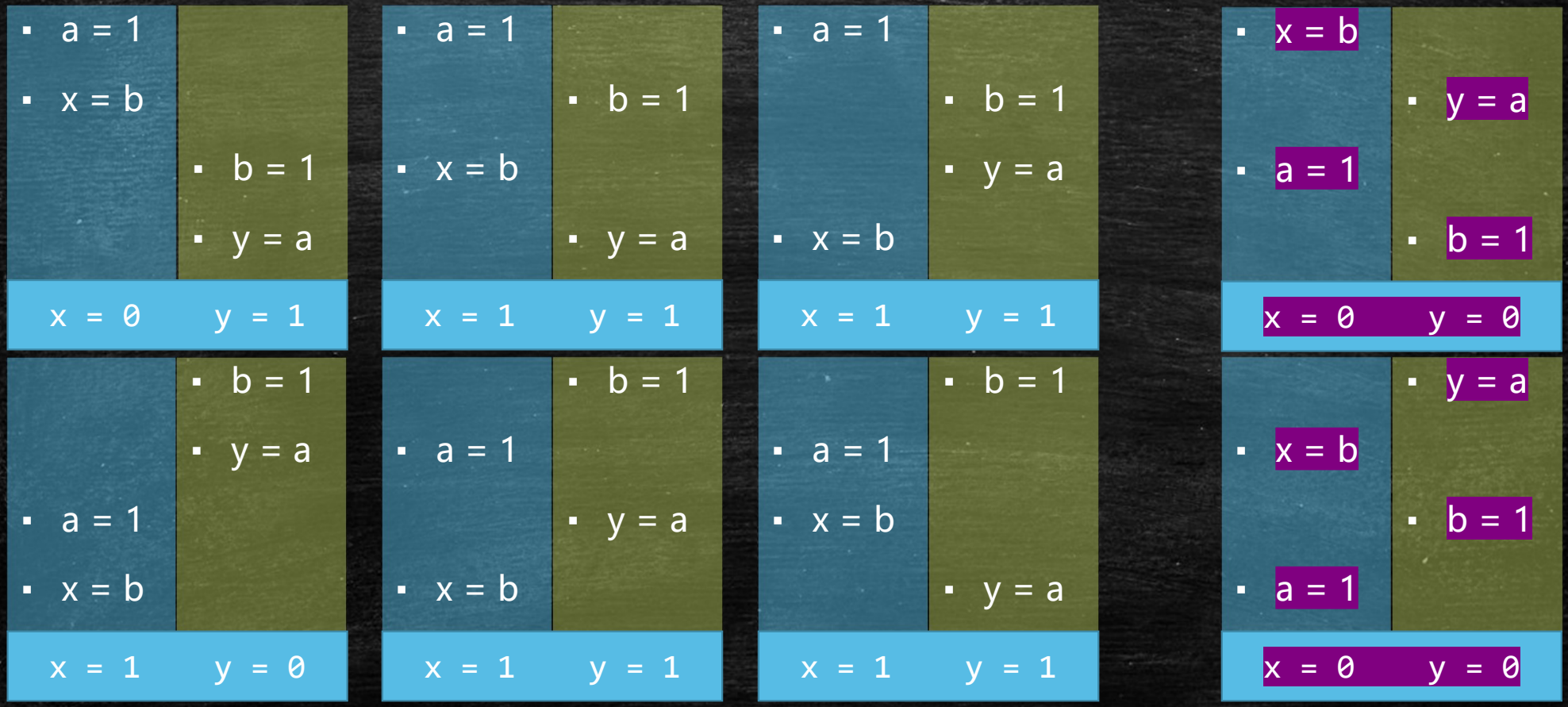
程序真的是按“顺序”执行的吗?

- x = 1
- x ++

- x = 1
- y = 1

- 为了提高执行效率，CPU指令可能会乱序执行
- 乱序执行不得影响单线程的最终一致性
  as – if – serial： 单线程程序看上去象序列化执行
- 乱序在多线程的情况下可能会产生难于察觉的错误

Ordering程序的 排列组合



http://mashibing.com

为何乱序?

- 简单说，为了提高效率

http://mashibing.com

乱序存在的条件

- as – if – serial
- 不影响单线程的最终一致性

JVM内存屏障

LoadLoad屏障：

对于这样的语句Load1; LoadLoad; Load2，

在Load2及后续读取操作要读取的数据被访问前，保证Load1要读取的数据被读取完毕。

StoreStore屏障：

对于这样的语句Store1; StoreStore; Store2，

在Store2及后续写入操作执行前，保证Store1的写入操作对其它处理器可见。

LoadStore屏障：

对于这样的语句Load1; LoadStore; Store2，

在Store2及后续写入操作被刷出前，保证Load1要读取的数据被读取完毕。

StoreLoad屏障：对于这样的语句Store1; StoreLoad; Load2，

在Load2及后续所有读取操作执行前，保证Store1的写入对所有处理器可见。

volatile的实现细节
二：JVM层面

---- StoreStoreBarrier ----

volatile 写

---- StoreLoadBarrier ----

volatile 读

---- LoadLoadBarrier ----
---- LoadStoreBarrier ----

如何阻止乱序?

- volatile（JVM级别）
- 内存屏障

数据一致性

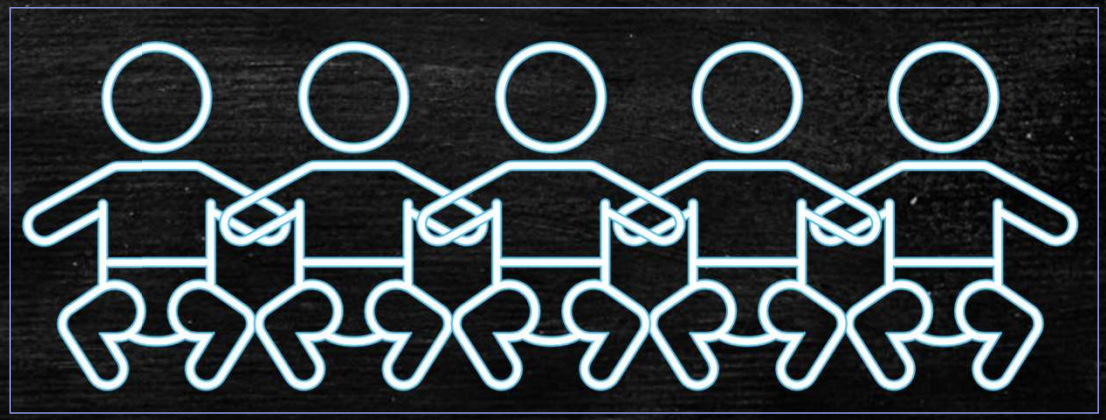
锁的概念

http://mashibing.com

锁的概念

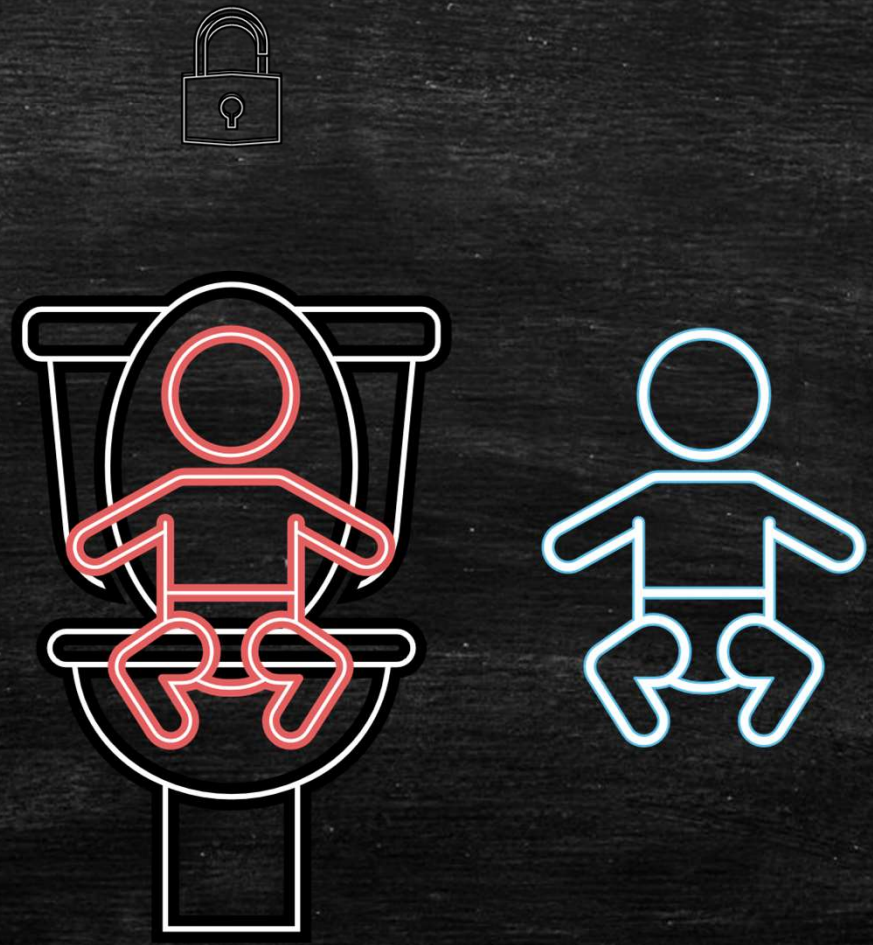
不持有锁的线程咋办？
忙等待
进队列等待

等待队列



http://mashibing.com

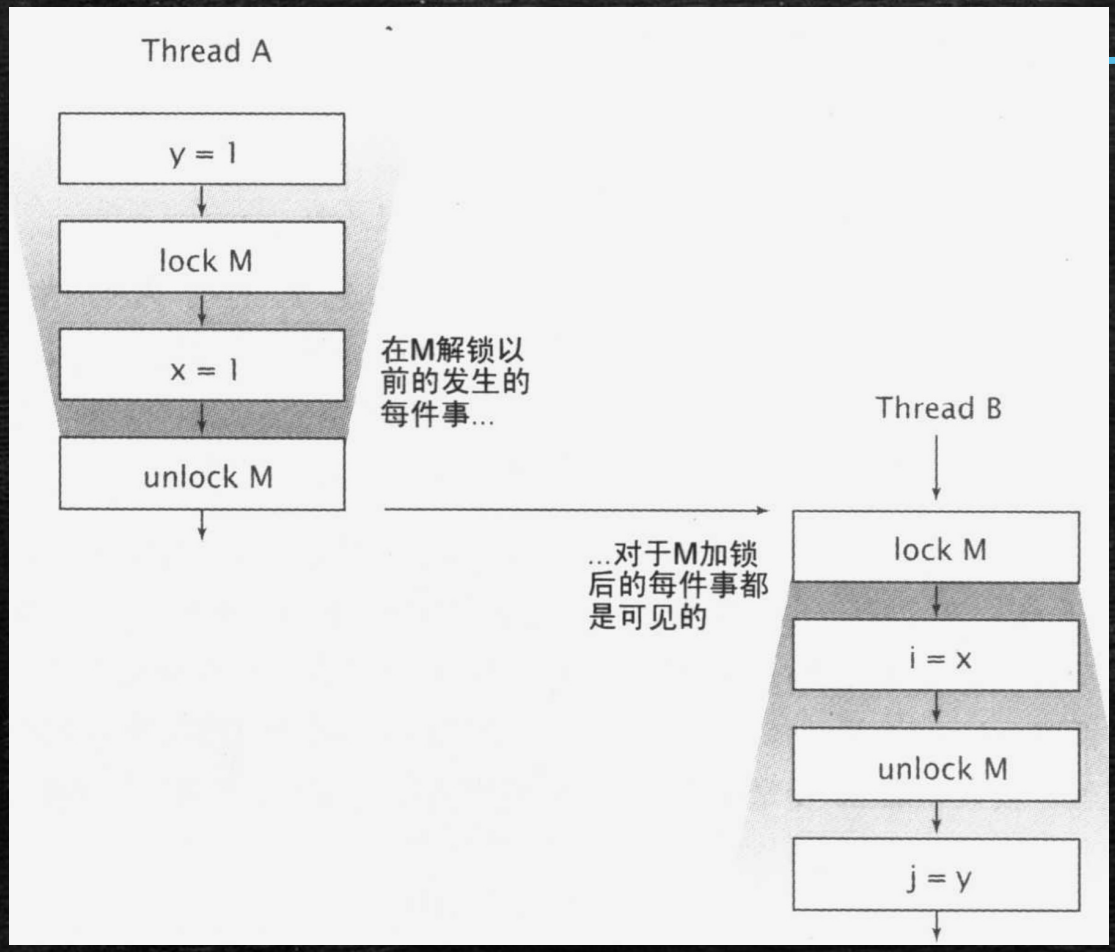
锁的概念



不持有锁的线程咋办？
忙等待
进队列等待

http://mashibing.com

synchronized可见性



锁不仅仅是关于同步与互斥的，也是关于内存可见的。为了保证所有线程都能够看到共享的、可变变量的最新值，读取和写入线程必须使用公共的锁进行同步。

contents – 基础

1. 什么是线程
2. 线程实现
3. 常用方法
4. 线程状态
5. 线程同步
6. synchronized细节（锁升级、同步方法与非同步方法、可重入）
7. 异常与锁
8. volatile关键字
9. AtomicXXX原子类
10. wait notify线程通信（高频面试）

contents – 进阶（1）- JUC常见同步工具

1. CAS自旋原理
2. ReentrantLock 可重入锁
3. Condition条件等待
4. CountDownLatch 门闩
5. CyclicBarrier 线程栅栏
6. Semaphore 信号量
7. Semaphore与Lock的区别（高频面试）
8. ThreadLocal 线程本地变量
9. 同步容器类的演变

contents – 进阶（2）- 同步容器

1. Map/Set从无锁到同步
2. 队列
    1. ArrayList
    2. LinkedList
    3. Collections.synchronizedXXX
    4. CopyOnWriteList
    5. Queue
        1. ConcurrentLinkedQueue / ConcurrentArrayQueue
        2. BlockingQueque
            - LinkedBQ – ArrayBQ – TransferQ – SynchronousQ
    6. DelayQueue执行定时任务

contents – 进阶（3）- 线程池

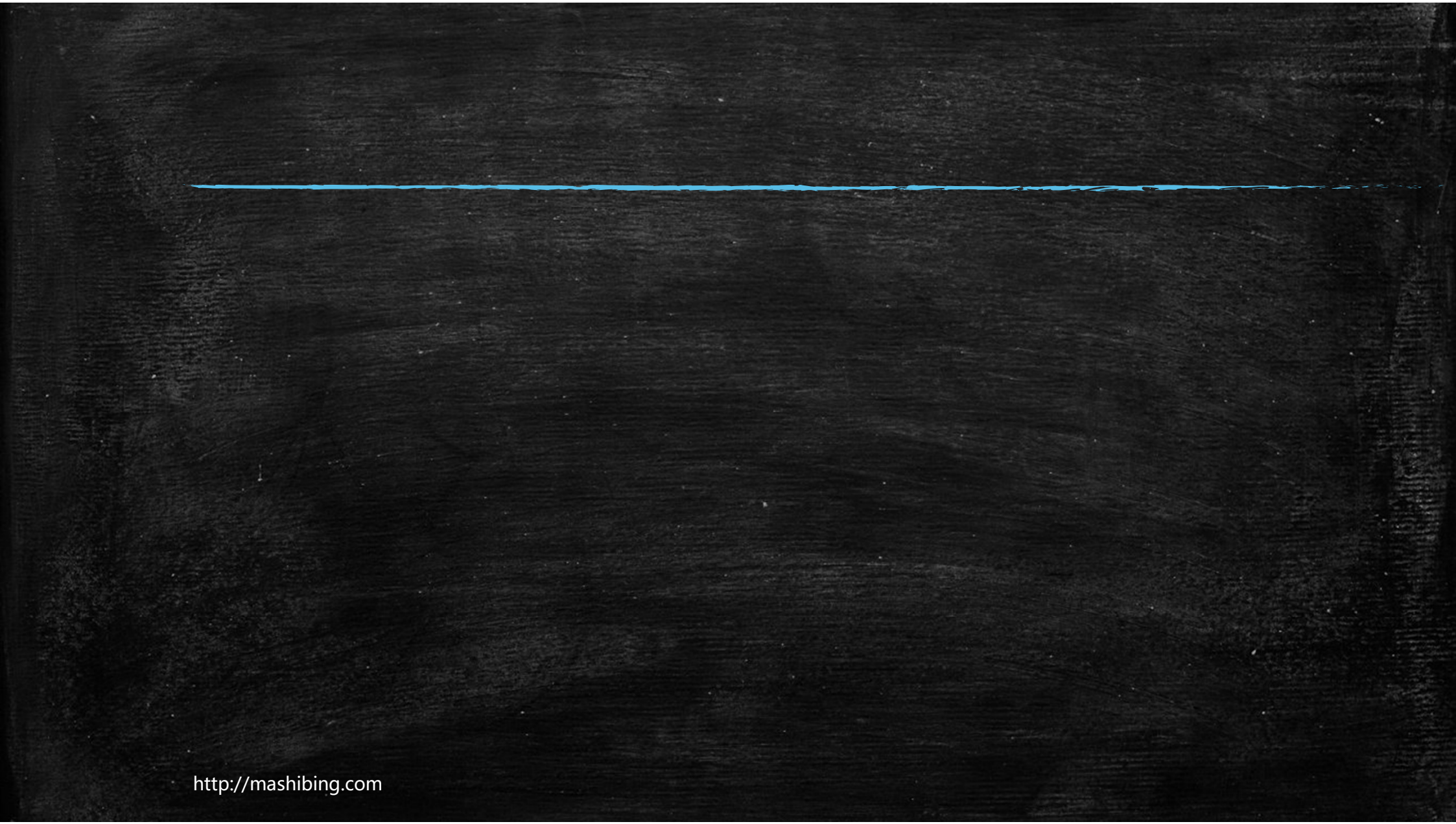
1. 线程池ThreadPool与线程执行器Executor
2. Executors and ExecutorService
3. Callable 带返回值的 Runnable
4. Future 线程异步调用
5. 常用线程池实现
   - fixed cached single scheduled workstealing forkjoin
6. ParallelStreamAPI – JDK8流的并行处理、

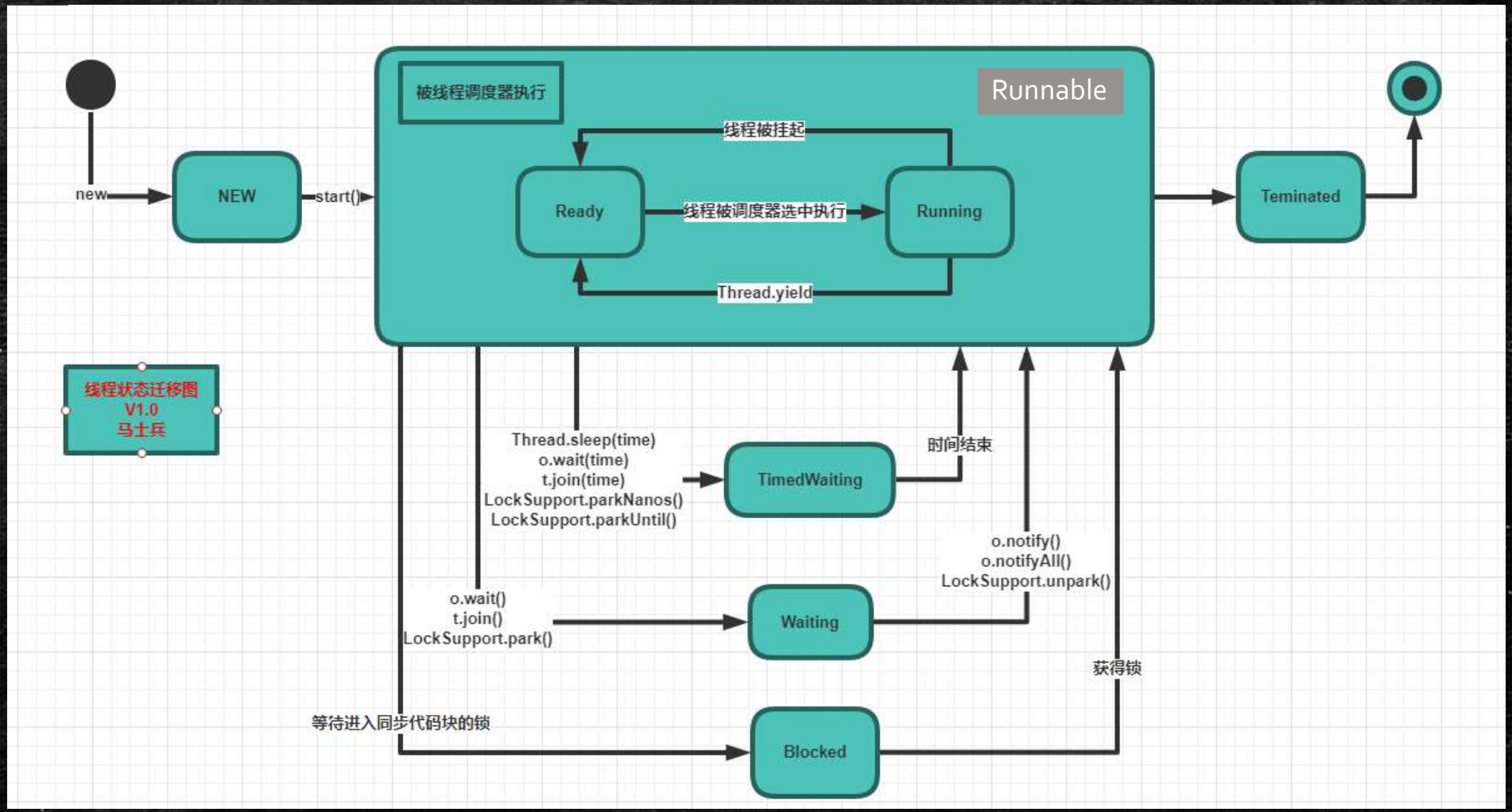
contents – 进阶（4）- 高级内容

1. 线程顺序执行控制（面试高频）
2. 纤程（面试加分项）
3. Disruptor - 目前性能最高的MQ微框架

学习

- 上天
  - 锻炼解决问题技能
  - 高并发 缓存 大流量 大数据量
- 入地
  - 面试
  - JVM OS 算法 线程 IO

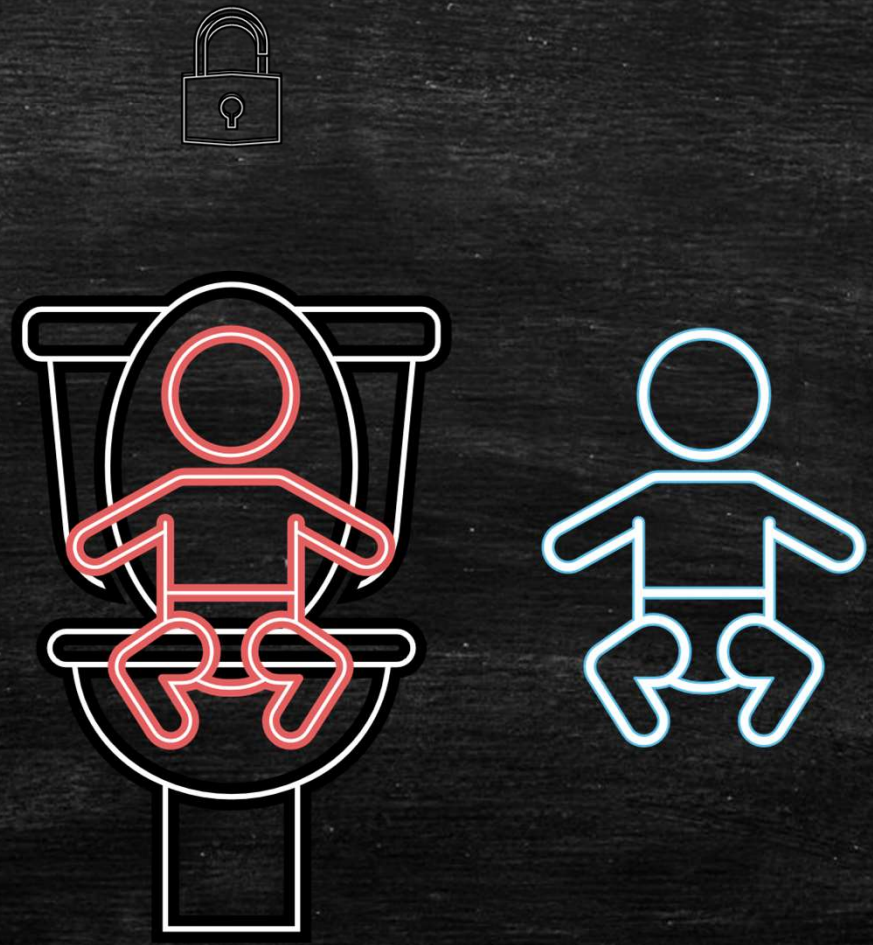

http://mashibing.com


new
NEW
start()
被线程调度器执行
Runnable
线程被挂起
Ready
线程被调度器选中执行
Running
Thread.yield
Teminated
线程状态迁移图
V1.0
马士兵
Thread.sleep(time)
o.wait(time)
t.join(time)
LockSupport.parkNanos()
LockSupport.parkUntil()
TimedWaiting
时间结束
o.notify()
o.notifyAll()
LockSupport.unpark()
o.wait()
t.join()
LockSupport.park()
Waiting
获得锁
等待进入同步代码块的锁
Blocked

锁的概念

http://mashibing.com

锁的概念



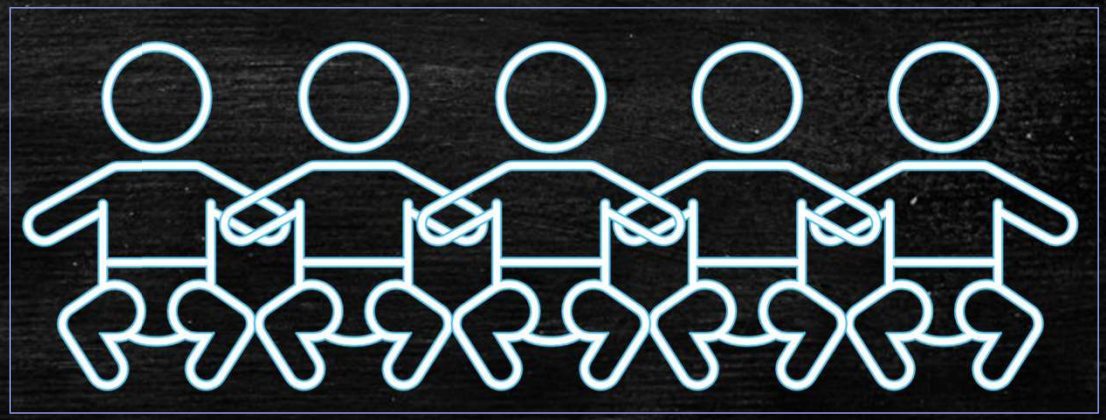
不持有锁的线程咋办？
忙等待
进队列等待

http://mashibing.com

锁的概念

不持有锁的线程咋办？
忙等待
进队列等待

等待队列



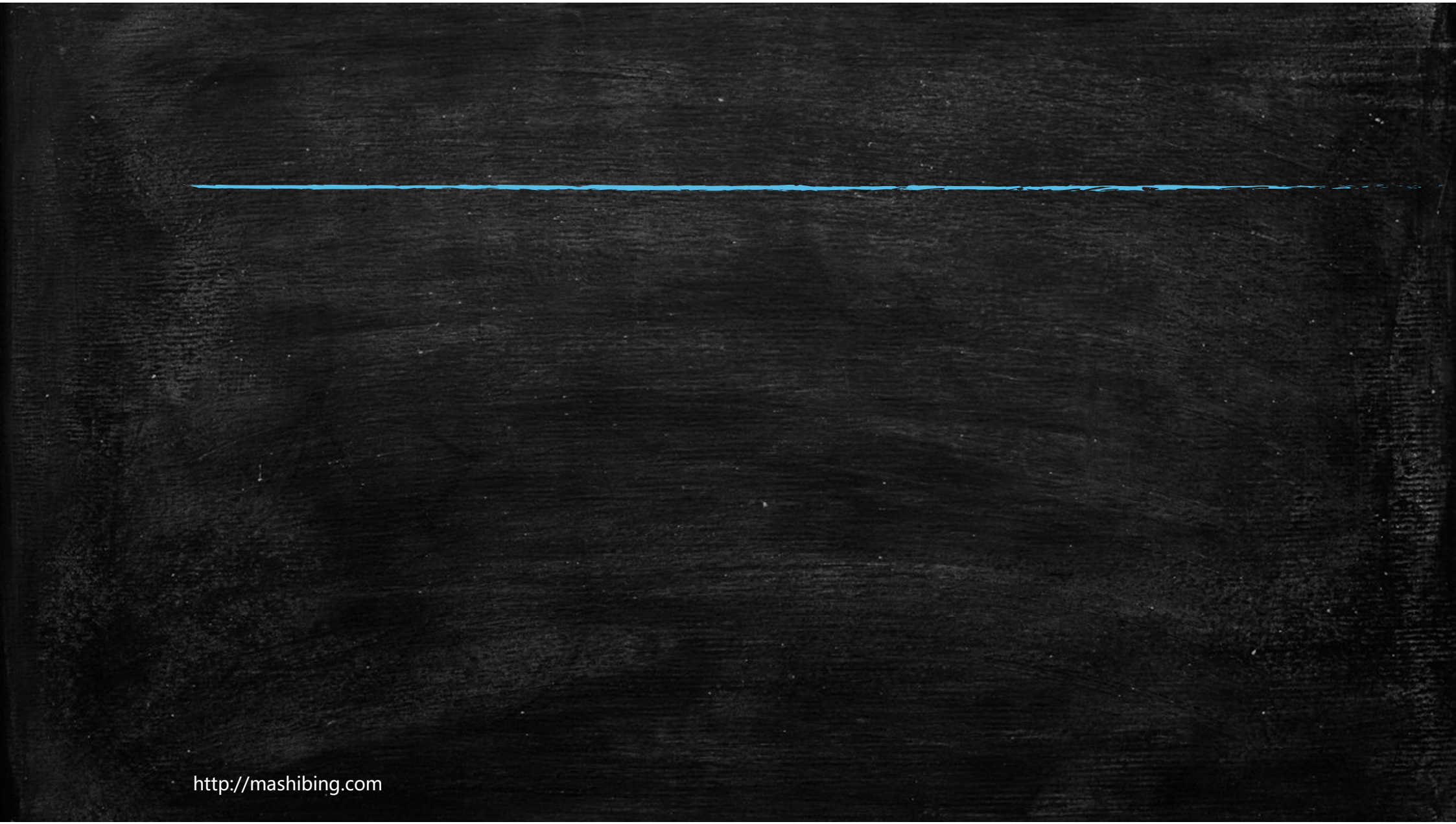
http://mashibing.com

sync VS lock

- 两个线程
  - sync lock
- 线程数非常多
  - sync lock

淘宝面试

实现一个容器，提供两个方法，add，size
写两个线程，线程1添加10个元素到容器中，线程2实现监控元素的个数，
当个数到5个时，线程2给出提示并结束


http://mashibing.com

三年java面腾讯

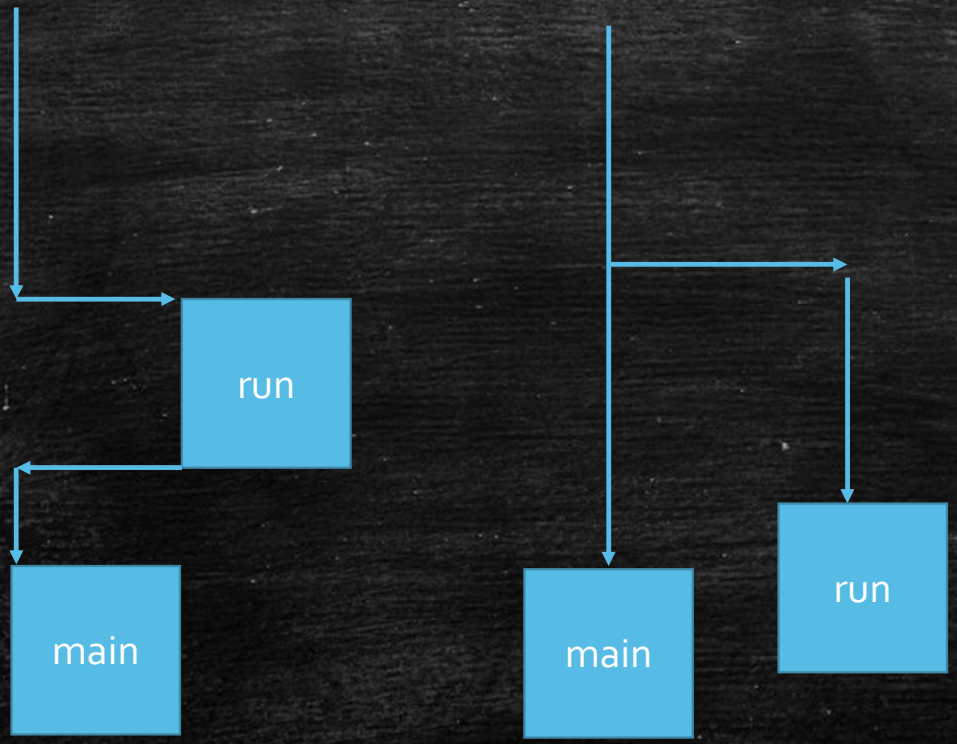
- 算法 JVM
- Redis
- 追问式：
  - 线程？
  - 纤程？quasar？协程？
  - 更轻量级的线程
    - java instrument

基本概念

- 进程 线程 协程/纤程（quasar）
- program app -> QQ.exe
- QQ running -> 进程
- QQ running -> 进程
- 线程 -> 一个进程里面的不同的执行路径
- 纤程 -> CPU – Ring0 – 1 2 - Ring3
  - Ring0 -> 内核态 Ring3 -> 用户态
  - 内核调用/系统调用 – 线程的操作
  - 用户态启动线程
    - 进入到内核态 – 保存用户态的线程
    - 用户态 不经过内核态的线程 – 纤程 golang的go程

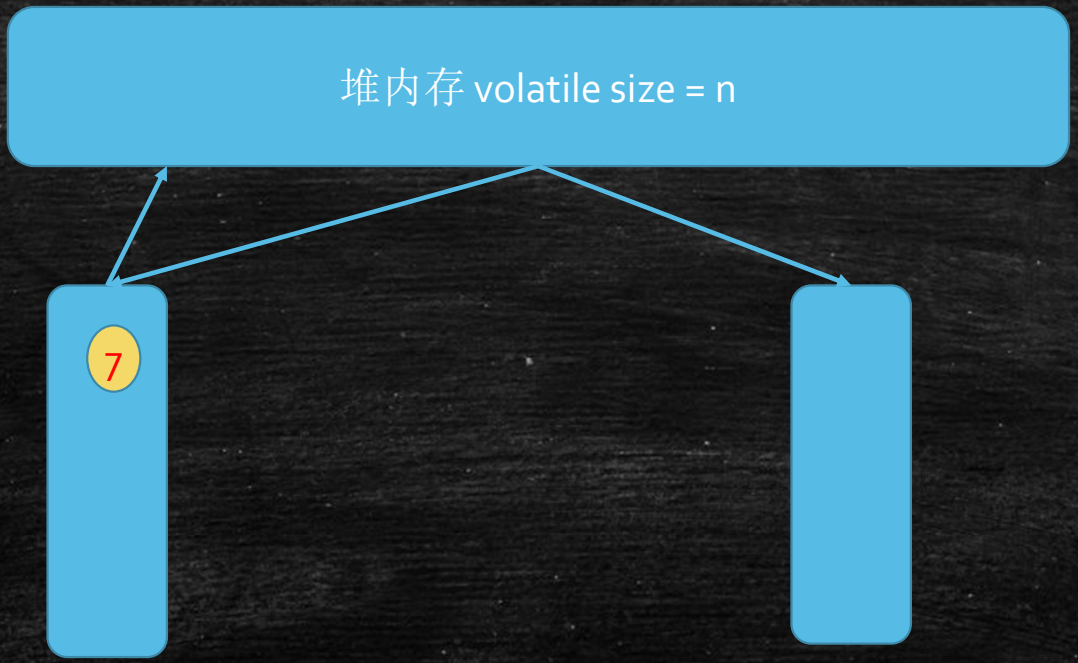
- 用户态 – 内核态
  - int 0x80 – 128
  - sysenter cpu支持
  - 保存用户态现场
    - 寄存器压栈
    - 进行syscall
    - 内核态返回 eax
    - 恢复用户态现场
    - 用户程序继续执行

线程的基本概念



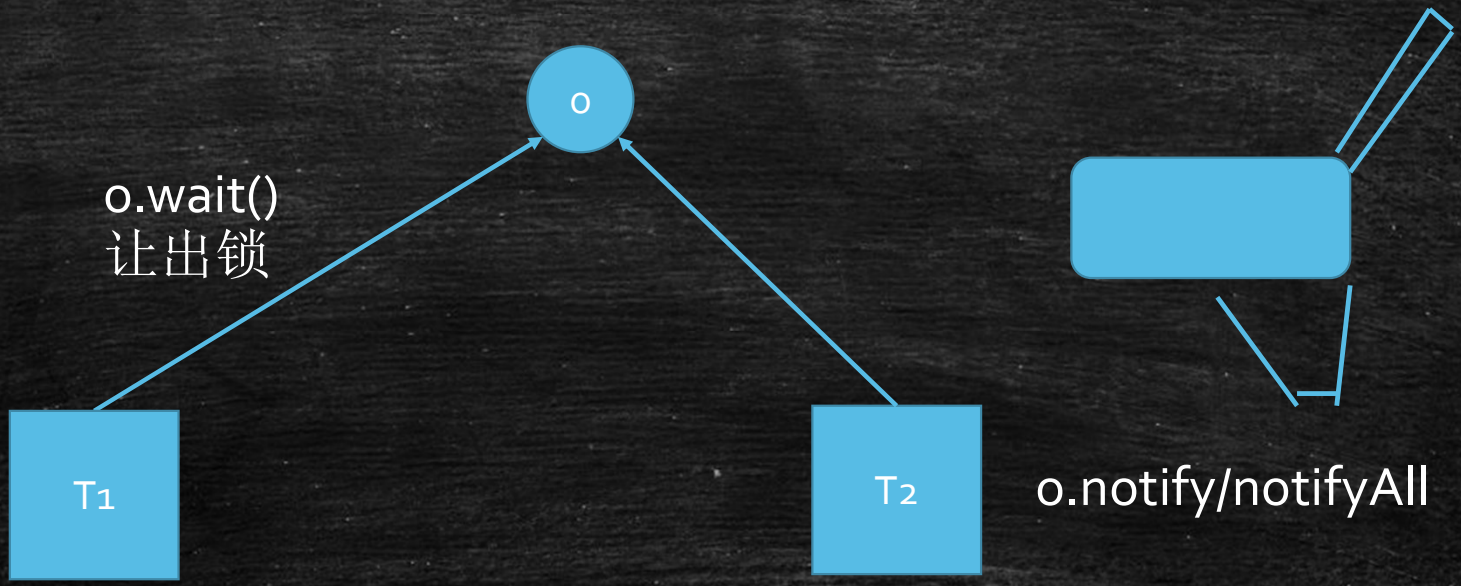
```
long start = System.currentTimeMillis();
```

volatile

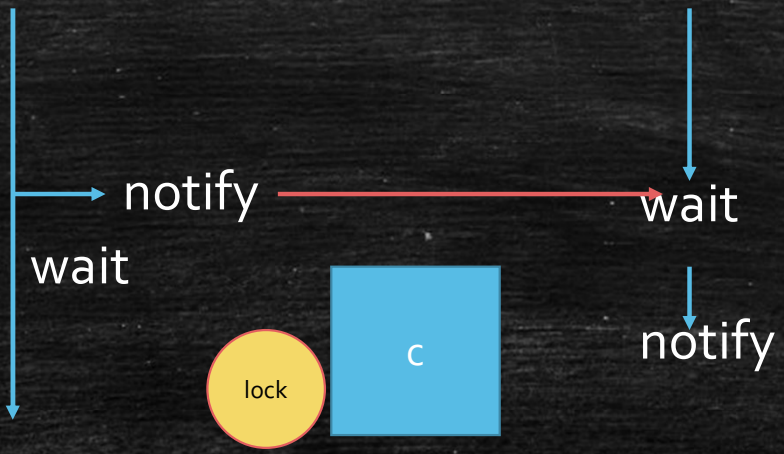


volatile

wait notify



wait notify

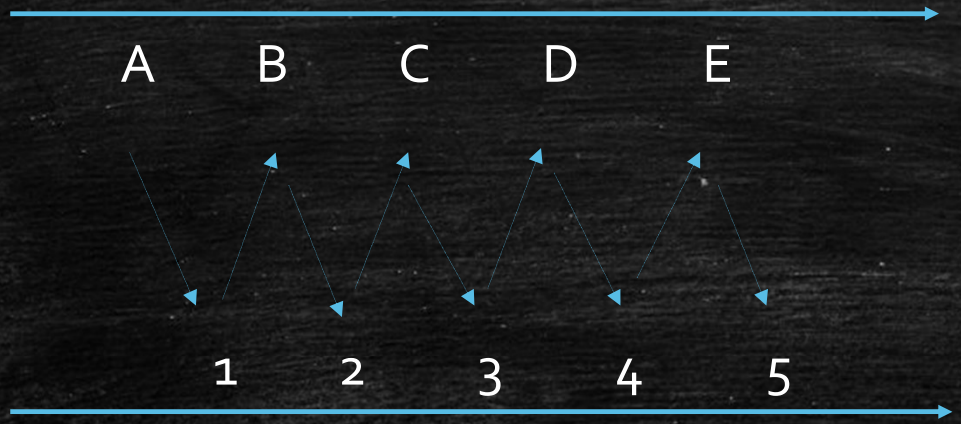


sync 锁升级

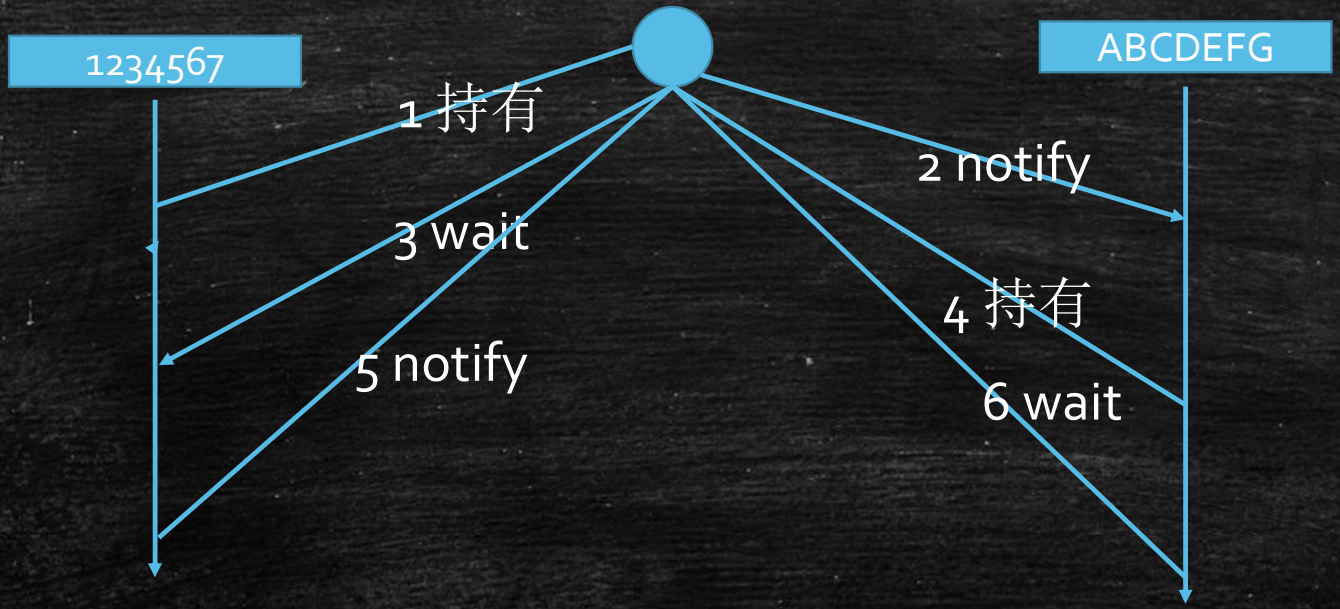
- 偏向锁
- 自旋锁
- 重量级锁

面试题：

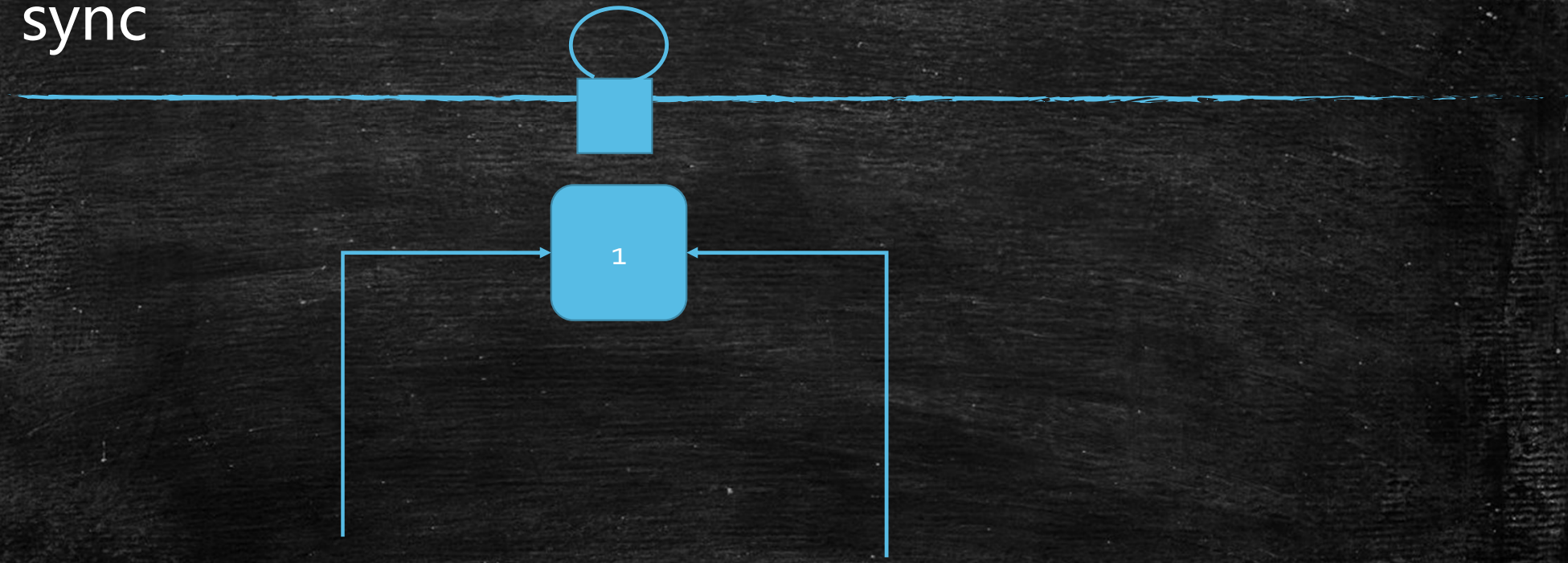
- 用两个线程，一个输出字母，一个输出数字，交替输出1A2B3C4D...26Z

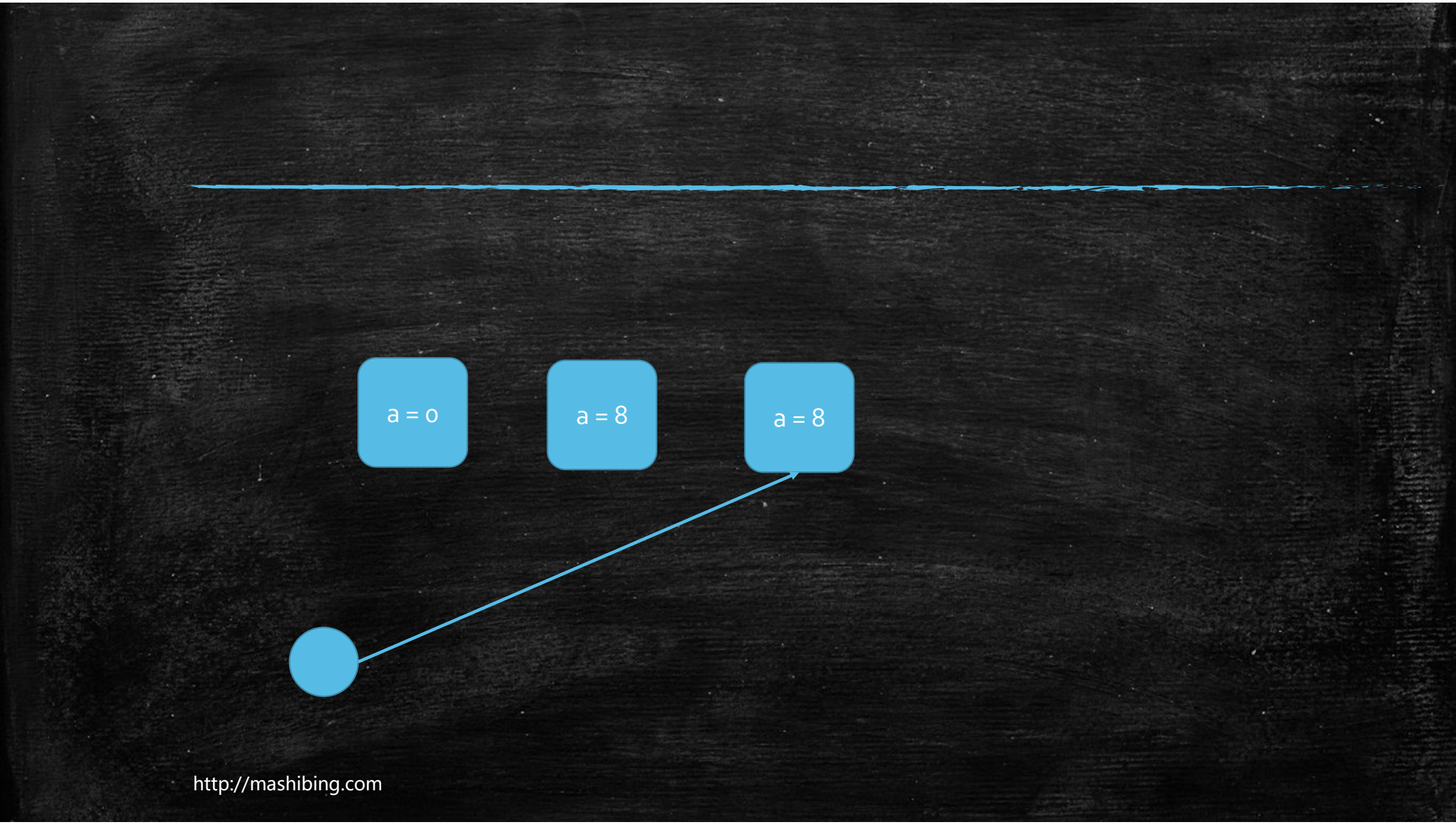


synchronized wait notify



sync




a = 0
a = 8
a = 8
http://mashibing.com

第一天内容回顾

- 线程的概念、启动方式、常用方法
- synchronized (Object)
  - 不能用String常量 Integer Long
  - “object”
- 线程同步
  - synchronized
    - 锁的是对象不是代码
    - this XX.class
    - 锁定方法 非锁定方法 同时执行
    - 锁升级
      - 偏向锁 自旋锁 重量级锁
      - 线程数少 – 自旋 多 – 重量级锁
      - 操作消耗时间长 重量级锁

volatile

- 保证线程可见性
  - MESI
  - 缓存一致性协议
- 禁止指令重排序(CPU)
  - DCL单例
  - Double Check Lock
  - Mgr06.java
    - loadfence原语指令
    - storefence原语指令

CAS（无锁优化 自旋 乐观锁）

- Compare And Set/Swap
- cas(V, Expected, NewValue)
  - if V == E
    V = New
    otherwise try again or fail
  - CPU原语支持
- ABA问题
  - 加version
  - A 1.0
  - B 2.0
  - A 3.0
  - cas(version)
  - 如果基础类型，无所谓 – 引用类型 你的女朋友跟你复合，中间经历了别的女人

Unsafe = c c++的指针

- 直接操作内存
  - allocateMemory putXX freeMemory pageSize
- 直接生成类实例
  - allocateInstance
- 直接操作类或实例变量
  - objectFieldOffset
  - getInt
  - getObject
- CAS相关操作
  - weakCompareAndSetObject Int Long
- c -> malloc free c++ -> new delete

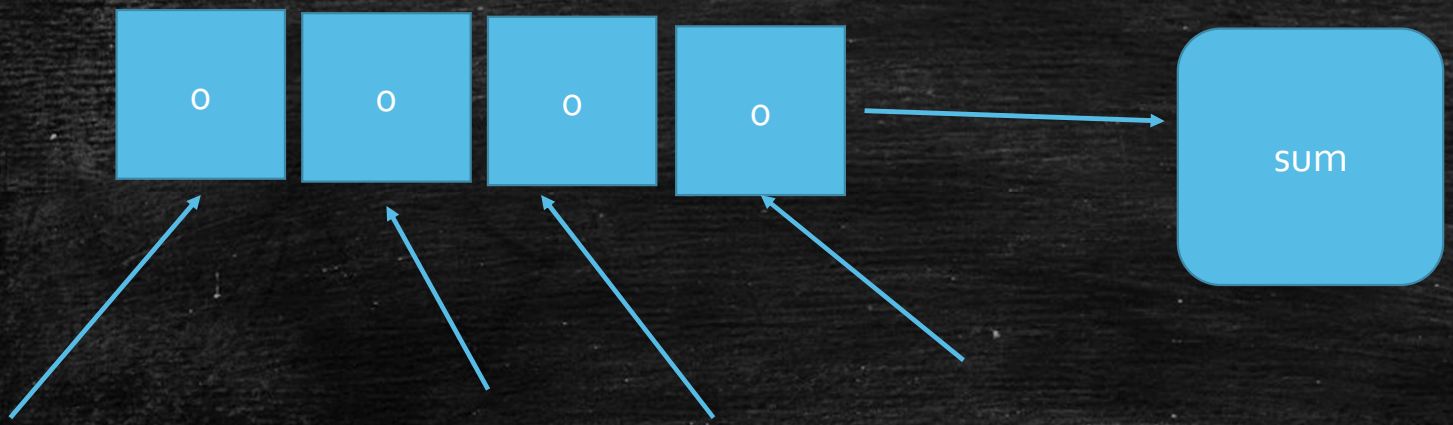
CAS

cas(期望值, 更新值)

m=0
m++

expected = read m;
cas(0, 1) {
　　for(;;) 如果当前m值==0 m=1
}

LongAdder



间歇性复习

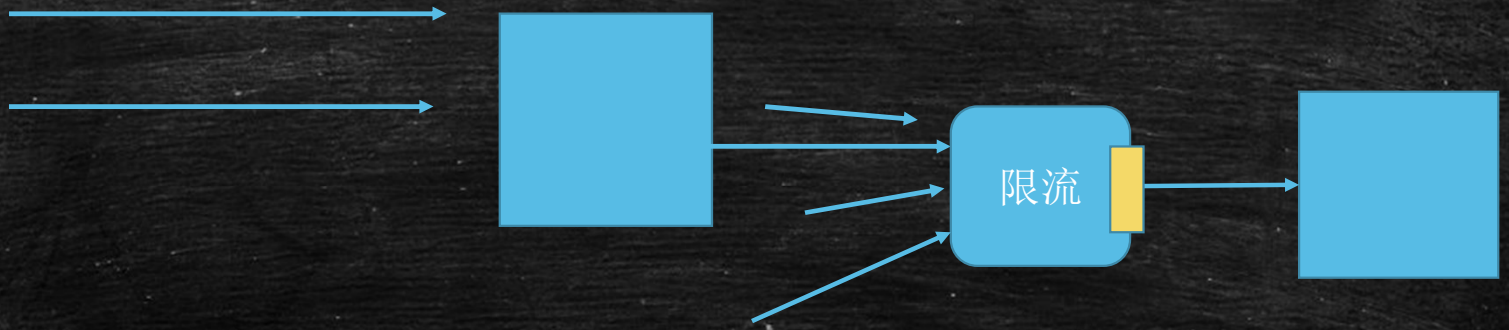
- synchronized
- volatile
- atomicXXX -> CAS
- increment -> sync atomicXXX LongAdder

ReentrantLock vs synchronized

- cas vs sync
- trylock
- lockinterupptibly
- 公平和非公平

Guava RateLimiter

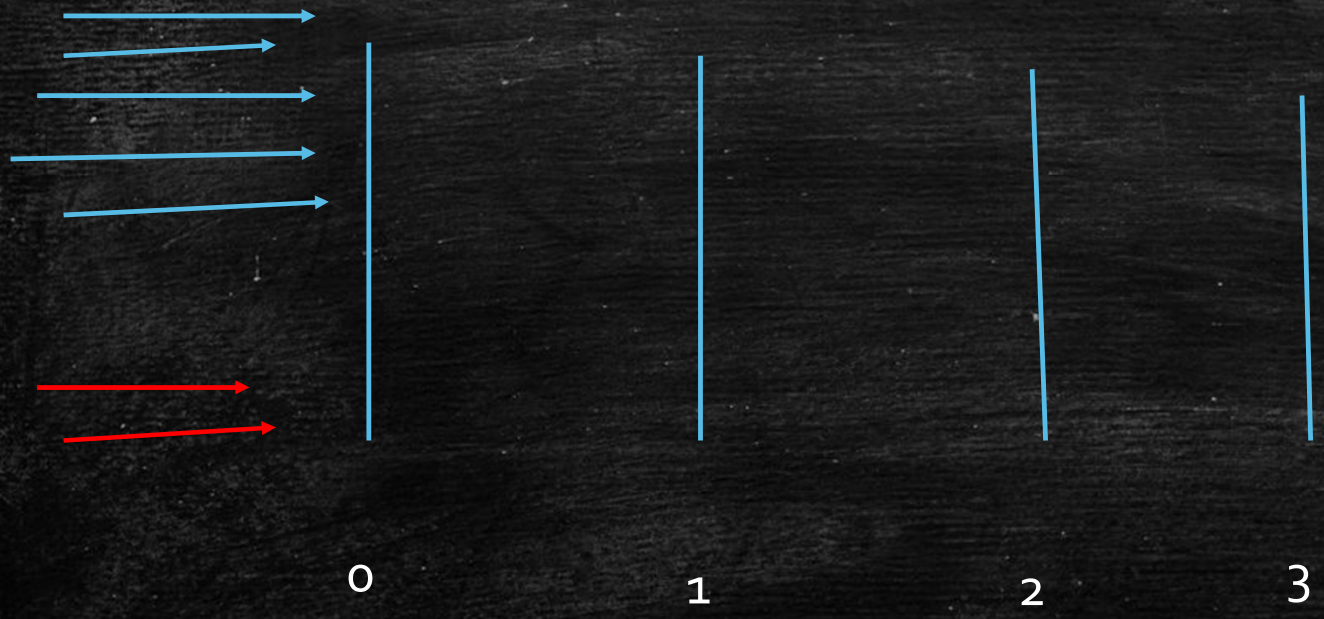
- 限流



CyclicBarrier

- 复杂操作
  1. 数据库
  2. 网络
  3. 文件
- 并发执行
  - 线程 – 操作
  - 线程 – 操作

phaser



http://mashibing.com

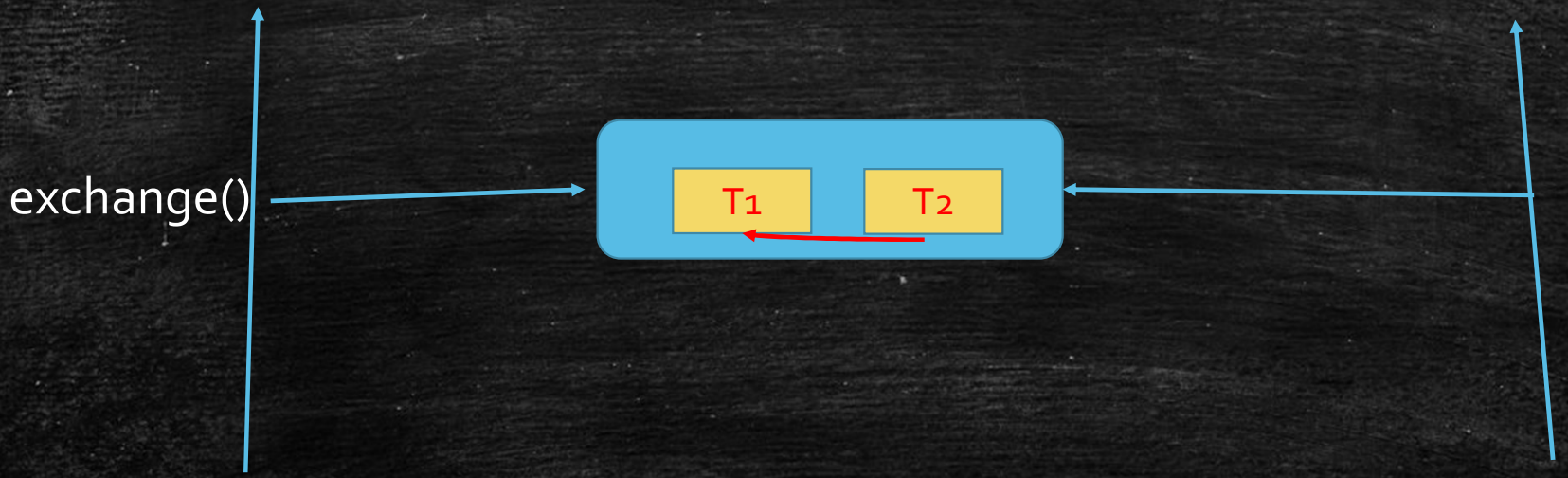
ReadWriteLock

- 共享锁
- 排他锁

semaphore

- 限流
- 车道和收费站

Exchanger



游戏中两个人交换装备?

复习：

- 线程的基本概念
- synchronized
- volatile
- AtomicXXX
- 各种JUC同步锁
  - ReentrantLock
  - CountDownLatch
  - CyclicBarrier
  - Phaser
  - ReadWriteLock – StampedLock
  - Semaphore
  - Exchanger
  - LockSupport

第四讲:

- TestLockSupport
- 淘宝面试题:
  - 实现一个容器，提供两个方法，add，size
    写两个线程，线程1添加10个元素到容器中，线程2实现监控元素的个数，当个数到5个时，线程2给出提示并结束
  - 面试题：写一个固定容量同步容器，拥有put和get方法，以及getCount方法，能够支持2个生产者线程以及10个消费者线程的阻塞调用
- 源码阅读技巧
- AQS源码解析

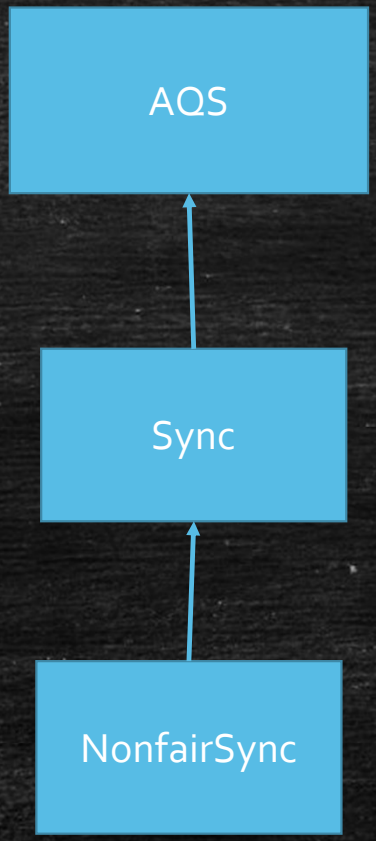
源码阅读原则

- 跑不起来不读
- 解决问题就好 – 目的性
- 一条线索到底
- 无关细节略过
- 一般不读静态
- 一般动态读法

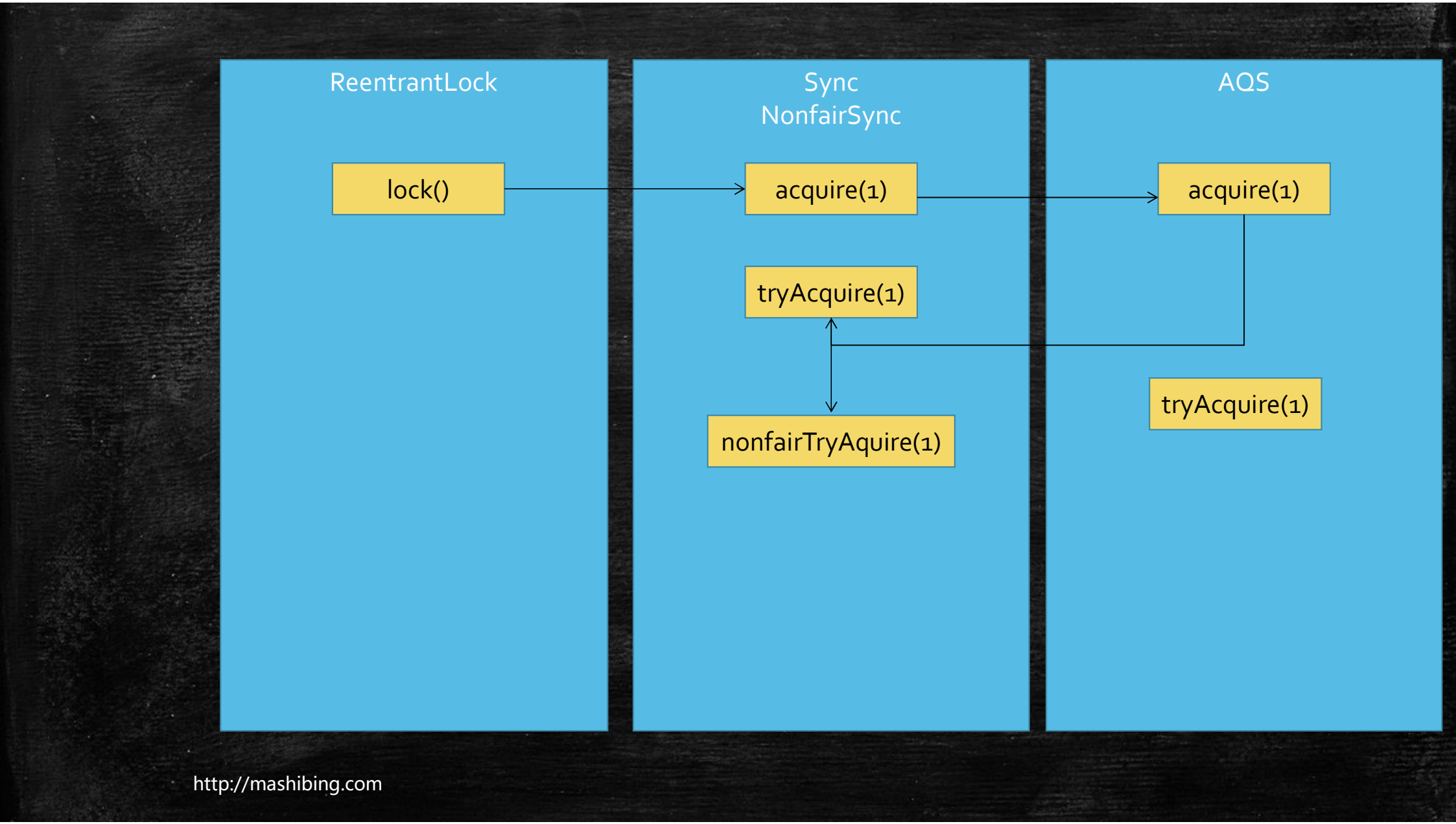
读源码很难！
理解别人的思路！
1：数据结构基础
2：设计模式

AQS：
Template Method
Callback Function
父类默认实现
子类具体实现

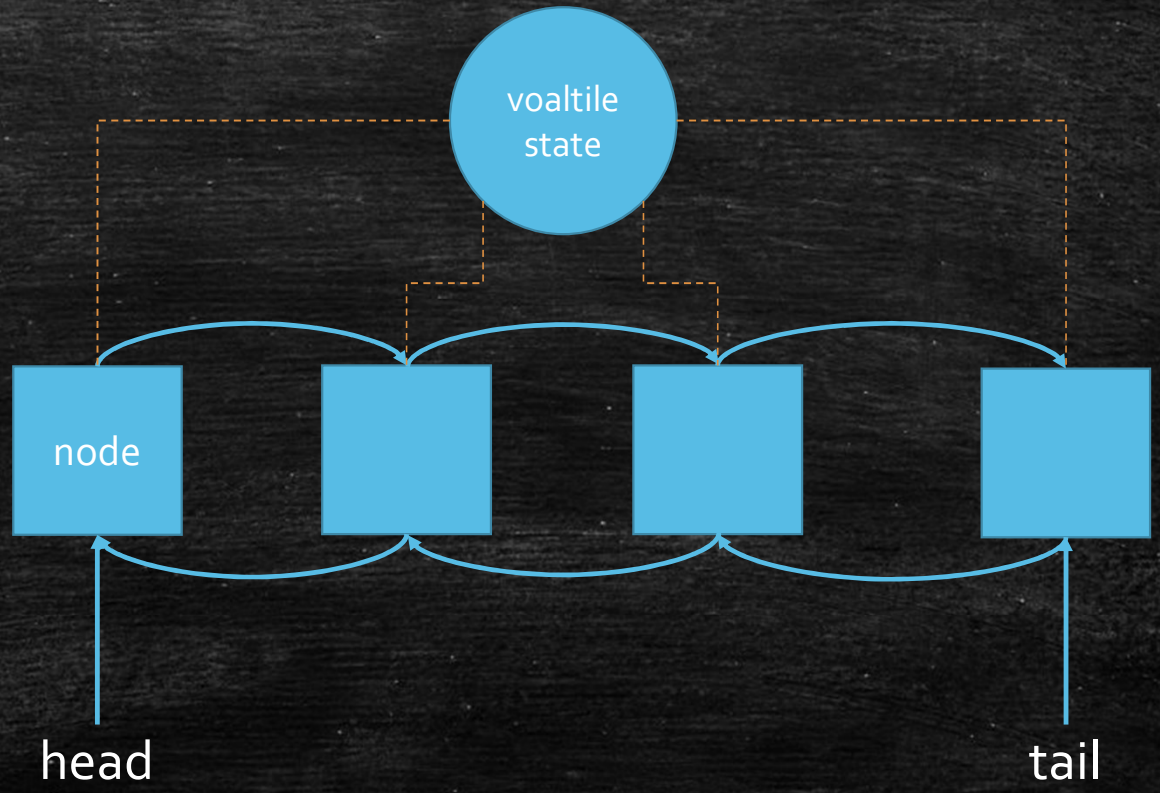
读源码先读骨架



http://mashibing.com


ReentrantLock
lock()
Sync
NonfairSync
acquire(1)
tryAcquire(1)
nonfairTryAquire(1)
AQS
acquire(1)
tryAcquire(1)
http://mashibing.com

AQS（CLH）



第五天

- AQS源码
  - VarHandle -> 1: 普通属性原子操作 2: 比反射快，直接操纵二进制码
- 强软弱虚
- ThreadLocal

ThreadLocal源码

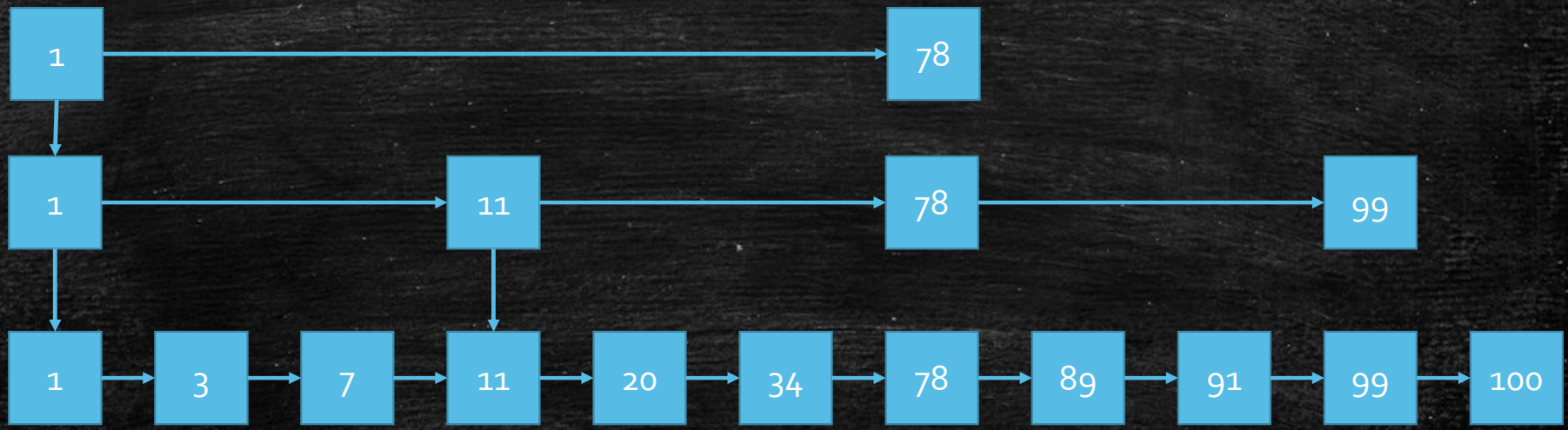
- set
  - Thread.currentThread. map(ThreadLocal，person)
  - 设到了当前线程的map中
- ThreadLocal用途
  - 声明式事务，保证同一个Connection

第六节课

- 容器 – Vector – Hashtable
- CopyOnWriteList
- ConcurrentHashMap
- ConcurrentSkipListMap
- BlockingQueue
- 目标：为ThreadPool做准备

- Vector Hashtable
  - 自带锁 – 基本不用
- Hashtable -> CHM
- Vector -> Queue
  - Queue List
  - 对线程友好的API offer peek poll
  - BlockingQueue
  - put take -> 阻塞
- DelayQueue SynchronousQ TransferQ

跳表



http://mashibing.com

线程池

- ThreadPoolExecutor
- ForkJoinPool
  - 分解汇总的任务
  - 用很少的线程可以执行很多的任务（子任务）TPE做不到先执行子任务
  - CPU密集型

调整线程池的大小

《Java并发编程实战》（http://mng.bz/979c）一书中，Brian Goetz和合著者们为线程池大小的优化提供了不少中肯的建议。这非常重要，如果线程池中线程的数量过多，最终它们会竞争稀缺的处理器和内存资源，浪费大量的时间在上下文切换上。反之，如果线程的数目过少，正如你的应用所面临的情况，处理器的一些核可能就无法充分利用。Brian Goetz建议，线程池大小与处理器的利用率之比可以使用下面的公式进行估算：

$$N_{threads} = N_{CPU} * U_{CPU} * (1 + W/C)$$

其中：

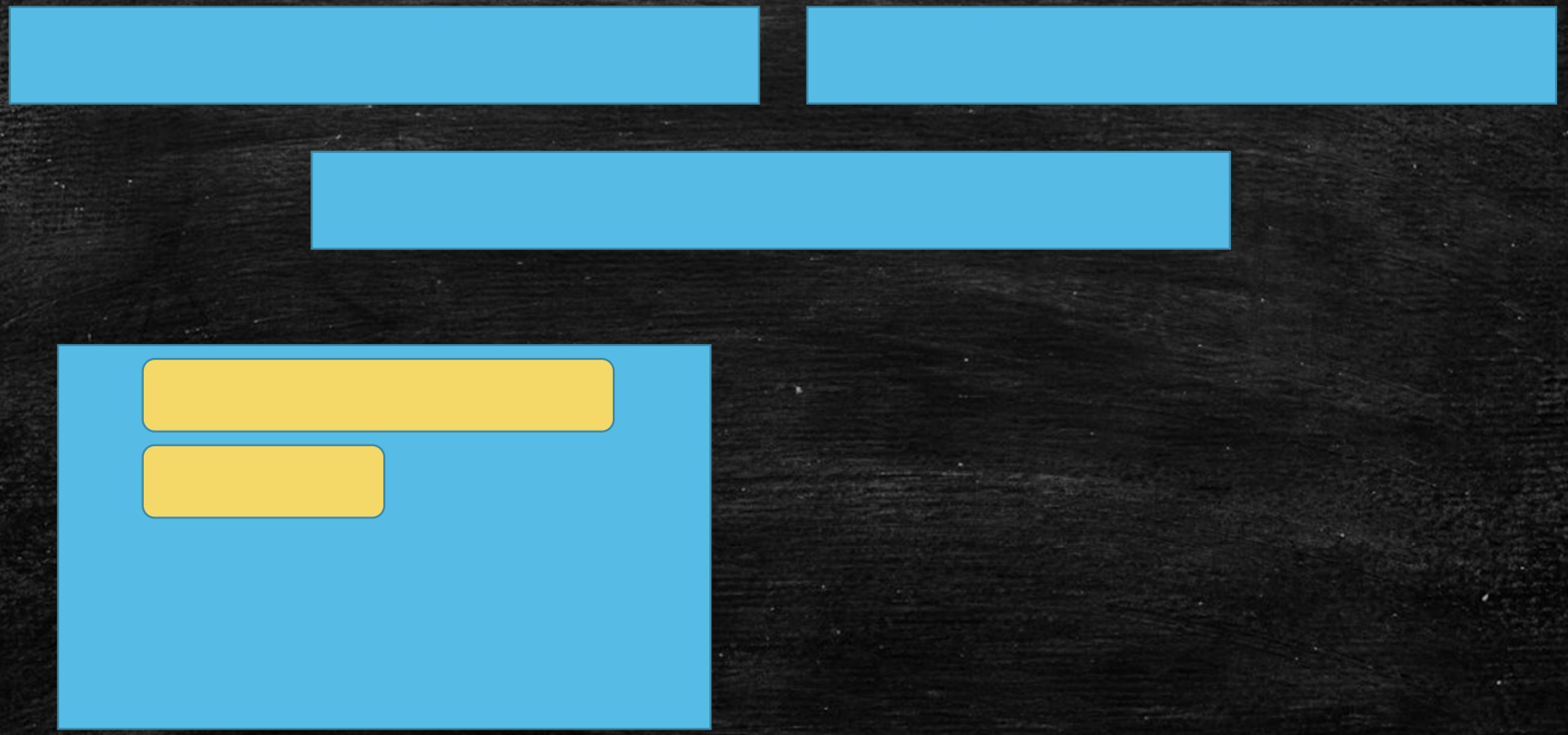
- N_{CPU}是处理器的核的数目，可以通过`Runtime.getRuntime().availableProcessors()`得到
- U_{CPU}是期望的CPU利用率（该值应该介于0和1之间）
- W/C是等待时间与计算时间的比率

Volatile MESI Lock MemoryBarrier 总线锁 缓存锁

http://mashibing.com

计算机的存储体系

NUMA
（zgc）



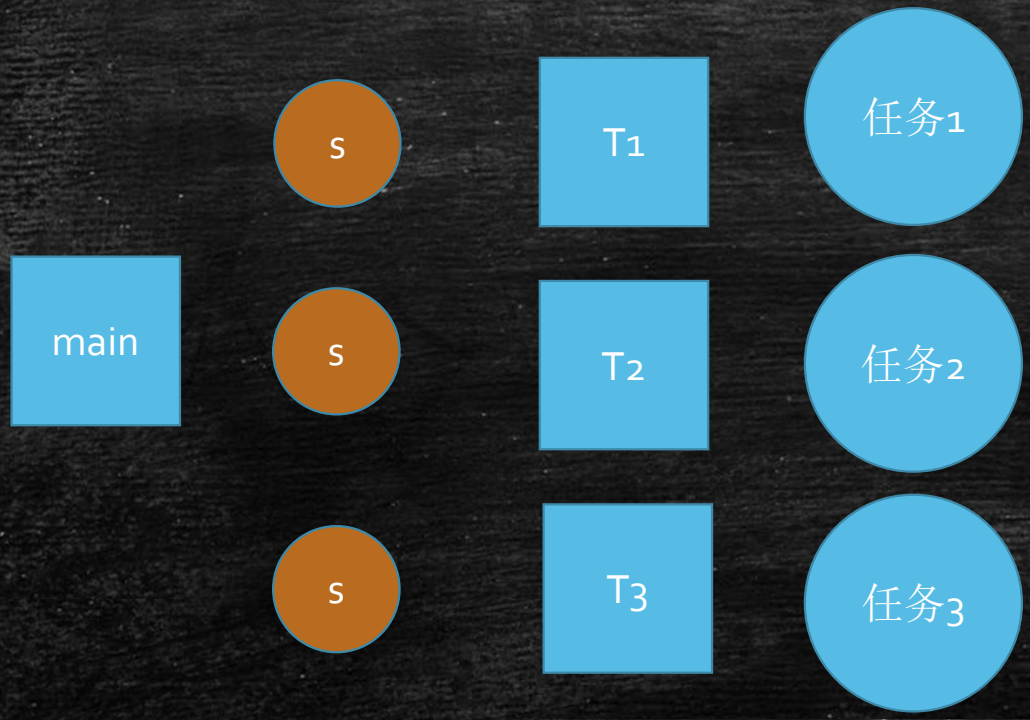
并行多任务取消问题（多线程问题）

最近遇到的比较多的问题：

一个大任务由多任务构成，要求其中某一个子任务出错后，
所有任务最短时间结束

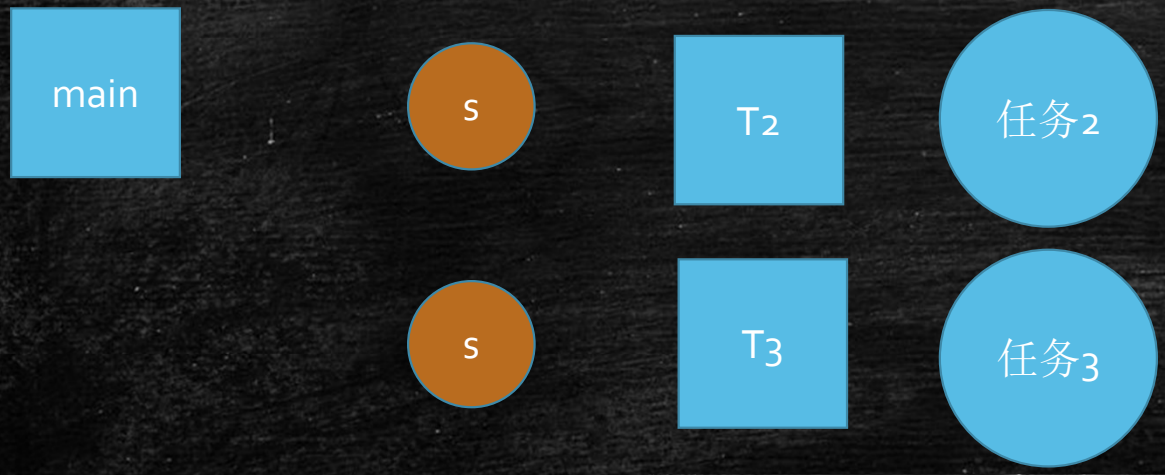
分布式事务模拟

Thread run() (Runnable)
ThreadPool
Future Callable (guava asyncCallable)
CompletableFuture
取消 日志 回滚 超时的处理



Thread run()

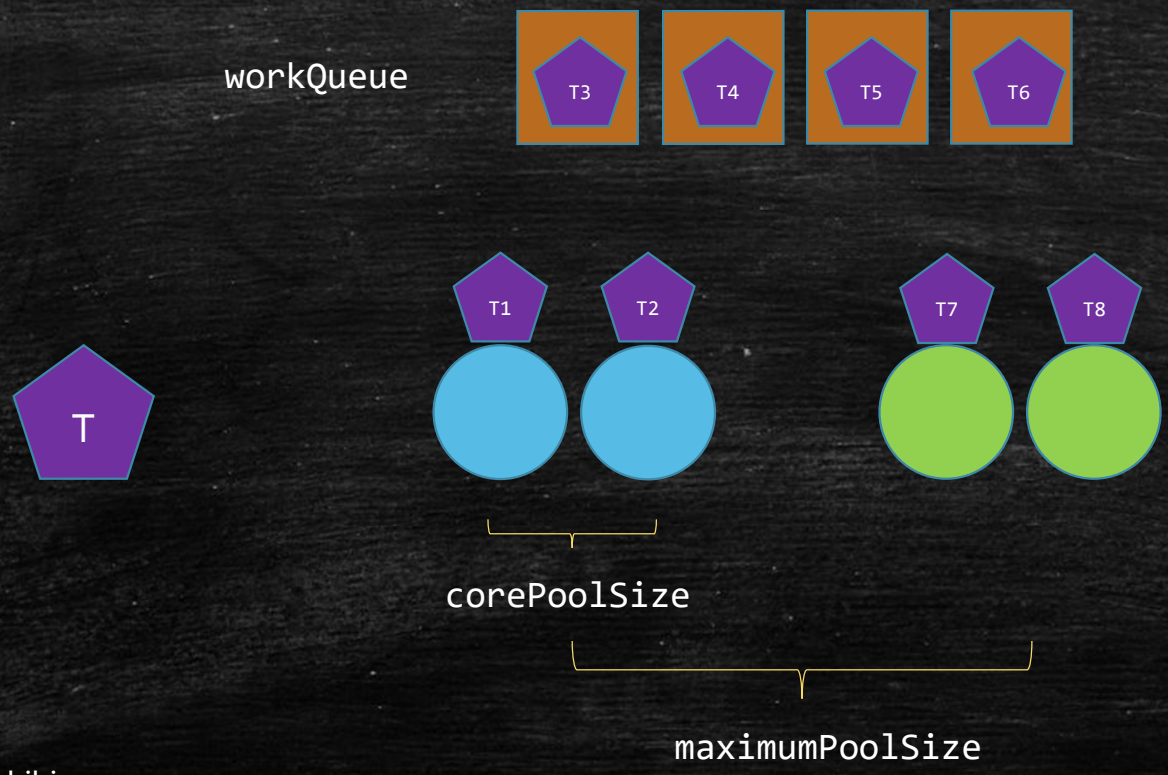
一个大任务由多任务构成，要求其中某一个子任务出错后，所有任务最短时间结束



Future Callable call

一个大任务由多任务构成，要求其中某一个子任务出错后，所有任务最短时间结束

线程池七大参数详解


workQueue
T3
T4
T5
T6
T
T1
T2
T7
T8
corePoolSize
maximumPoolSize

华为考题（生产者，消费者问题）

* 题目：有一个生产奶酪的厂家，每天需要生产100000份奶酪卖给超市，通过一辆货车发货，送货车每次送100份
* 厂家有一个容量为1000份的冷库，用于奶酪保鲜，生产的奶酪需要先存放在冷库，运输车辆从冷库取货
* 厂家有三条生产线，分别是牛奶供应生产线，发酵剂制作生产线，奶酪生产线。
* 生产每份奶酪需要2份牛奶和一份发酵剂
* 请设计生产系统

华为笔试

```
/**
 * 喝咖啡
 * 题目描述
 *
 * 2019.4.3华为笔试题目：喝咖啡
 *
 * 有m个咖啡机，n个人。一个清洗机。
 * 有n个人要喝咖啡，喝完之后要清洗杯子。m个咖啡机做咖啡的时间各不相同，分别为V1、V2...Vm。清洗机清洗杯子所用时间为x。
 * 此外，如果一个人喝完的杯子y时间内没清洗，里面的残存的咖啡会挥发掉，相当于清洗完毕。
 * 问所有人喝完咖啡+清洗完杯子需要的最短时间（忽略喝咖啡需要的时间）。
 *
 * 输入：
 * 第一行：一个数字T，表示T组测试数据
 * 第二行：第一组测试数据的  n m x y
 * 第三行：第一组测试数据m个咖啡机的工作效率   V1 V2 。。。Vm
 *
 * 第四行：第二组。。。。以此类推
 */
```