Java内存模型的历史变迁

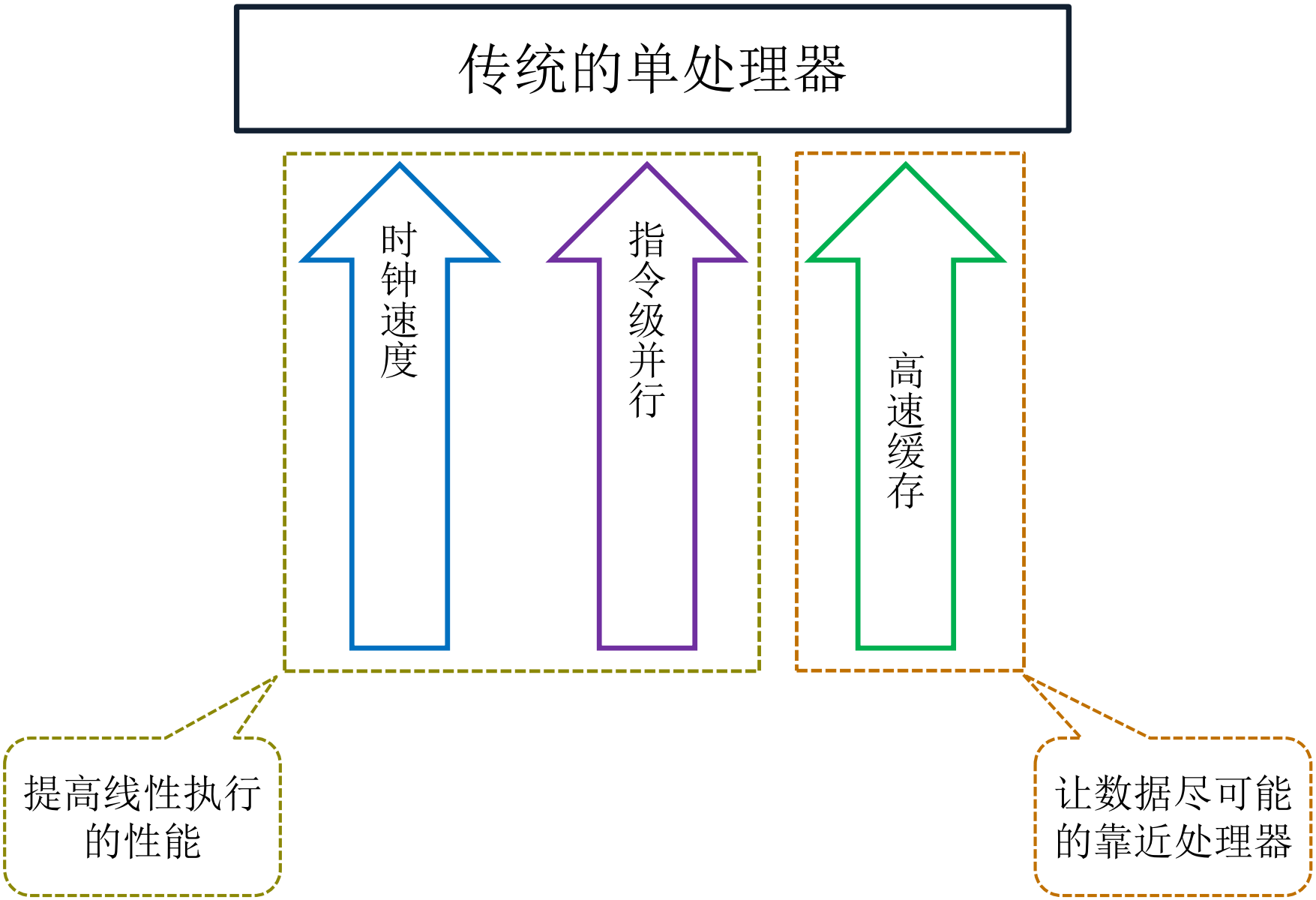
程晓明

2015年05月24日

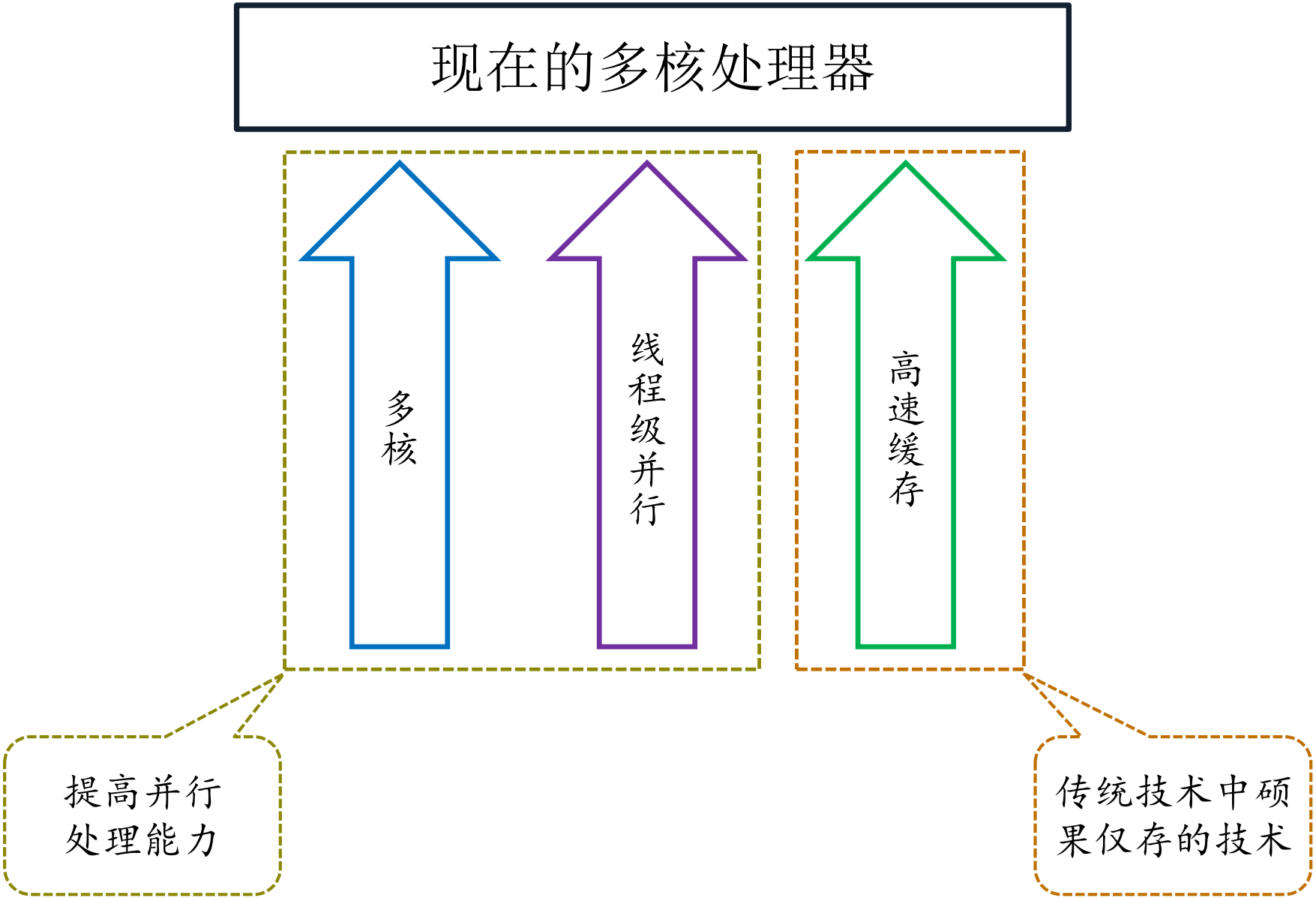
简介

- 背景知识
- 史前时代
  - 顺序一致性内存模型
- 解放前
  - Java旧的内存模型
- 解放后
  - Java新的内存模型

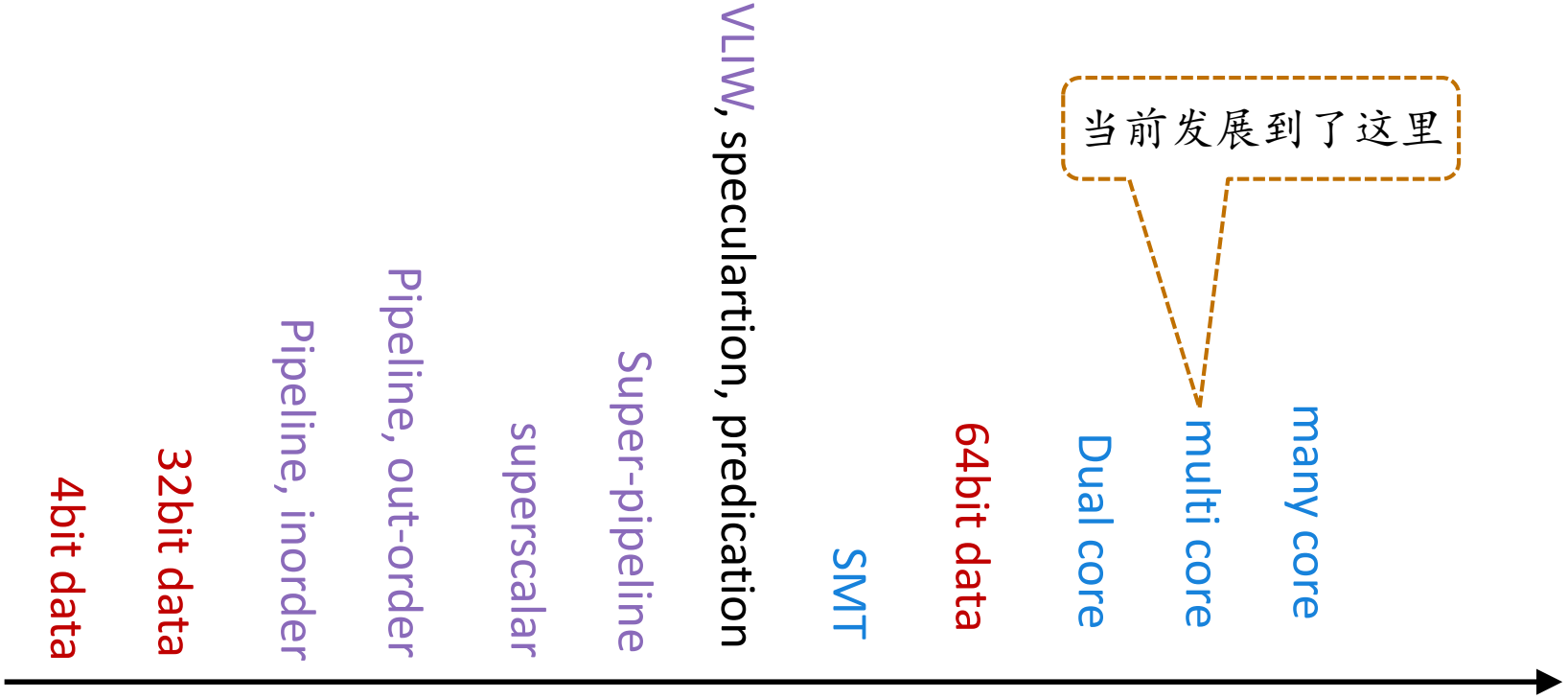
传统的CPU--提高线性执行性能



现代的CPU--提高并行执行性能

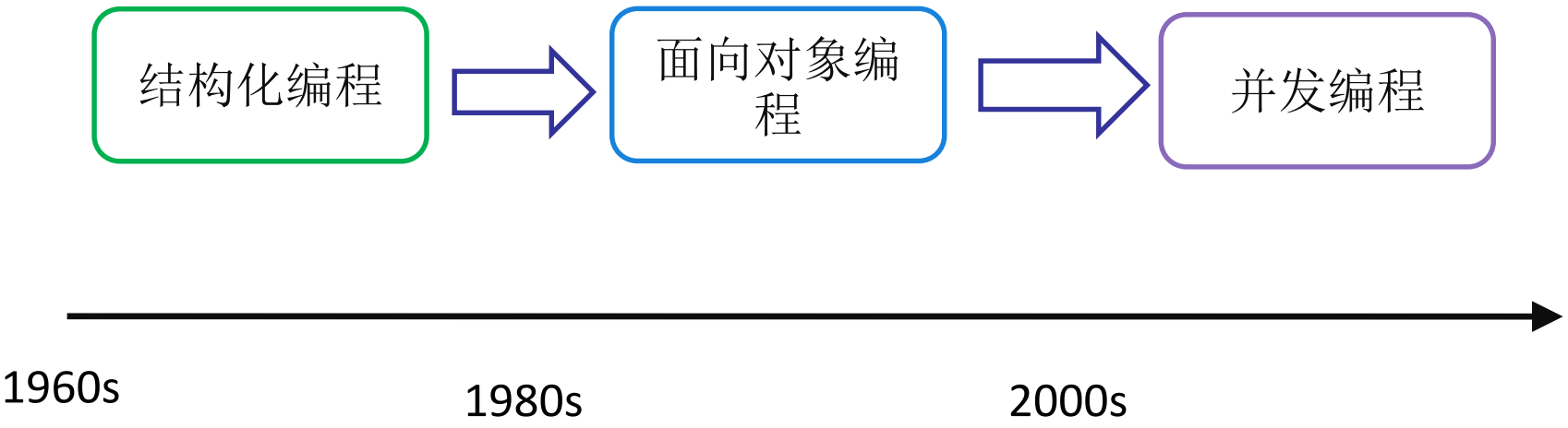


CPU微架构的演化历史



- Bit-Level Parallelism
- Instruction-Level Parallelism (ILP)
- Thread-Level Parallelism (TLP)

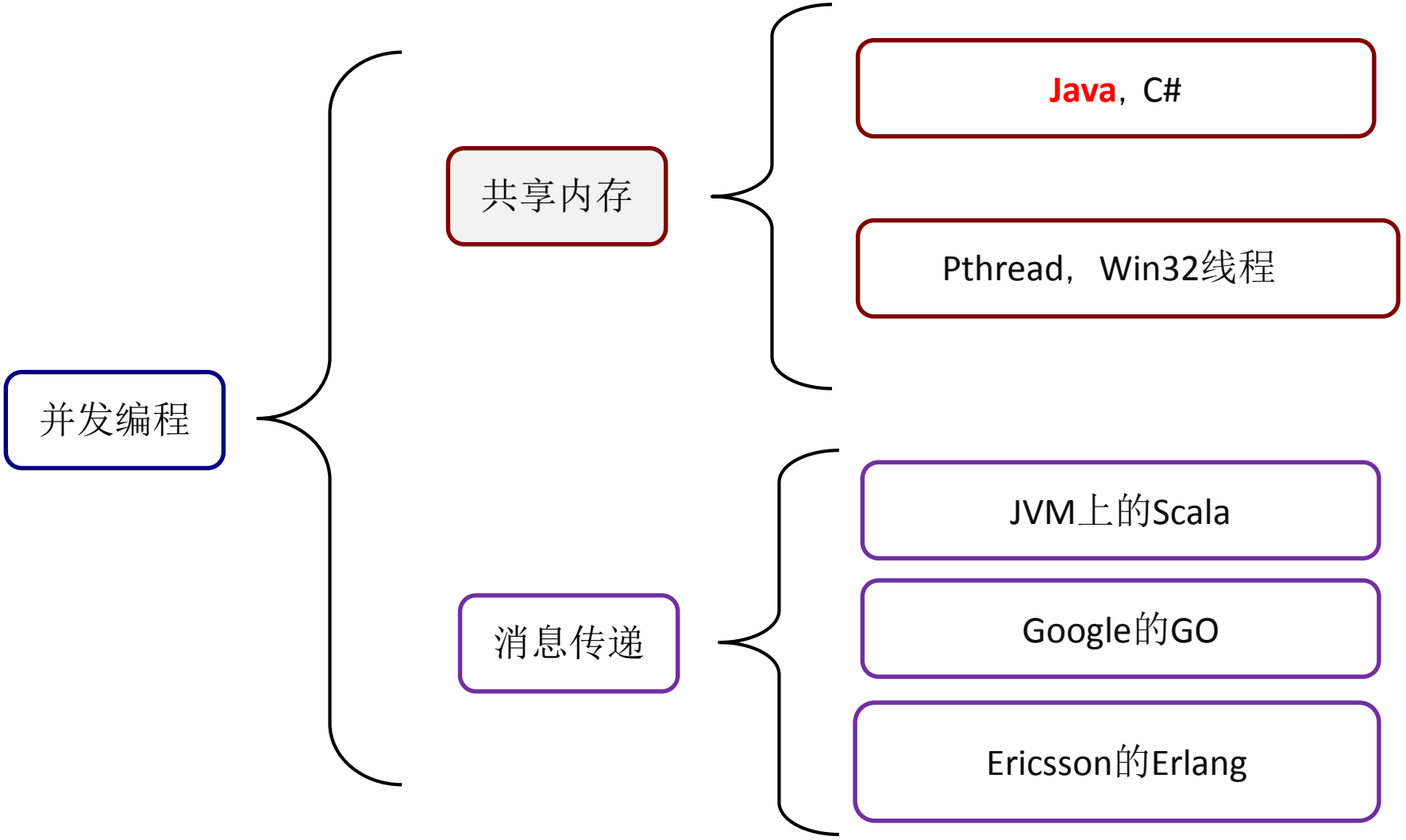
软件开发领域的重大变革



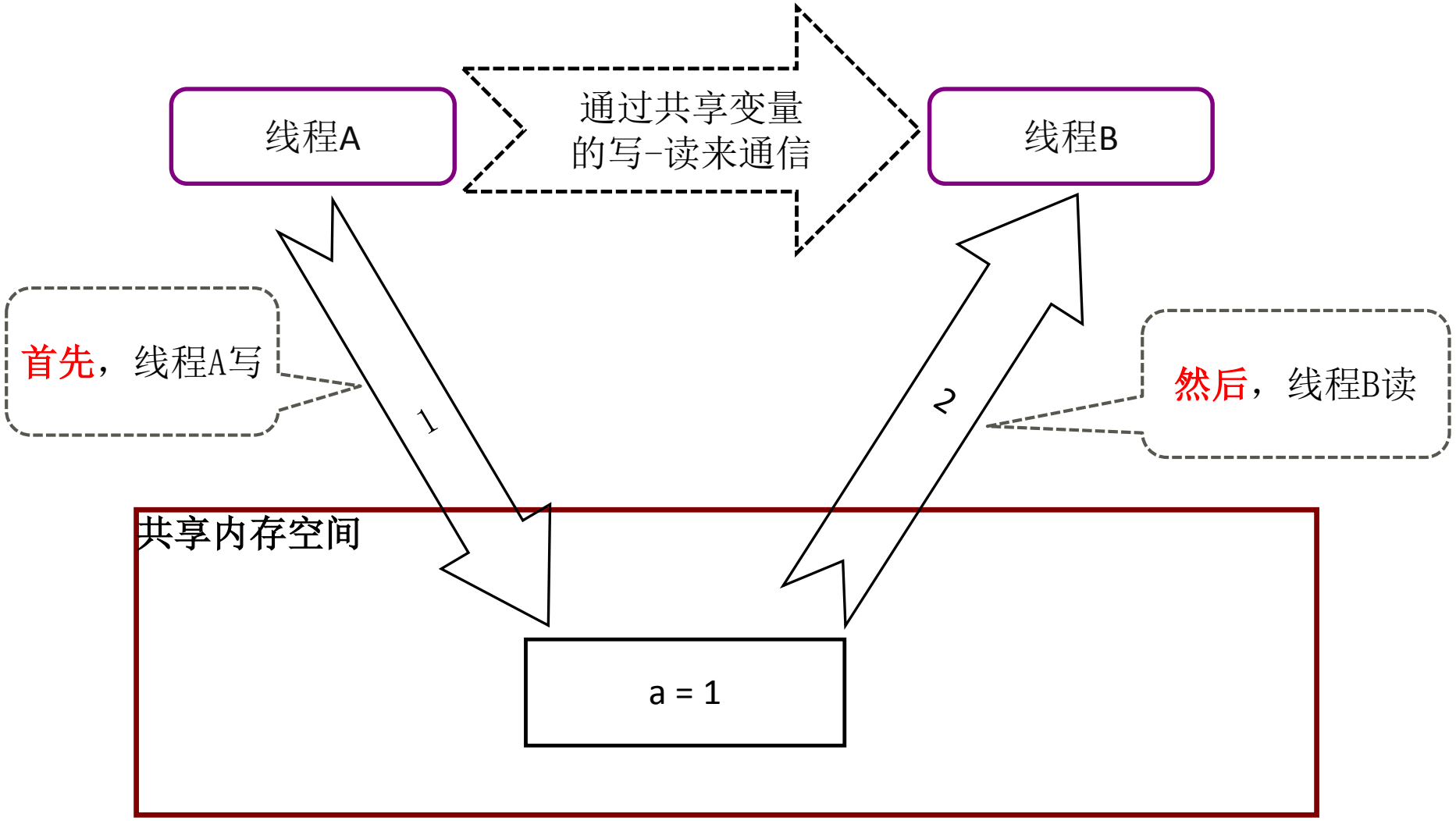
至少到2020之前，程序员必须理解并行处理器和并发编程，才能够使程序在并行计算机上有效的运行。

—— David A. Patterson & John L. Hennessy

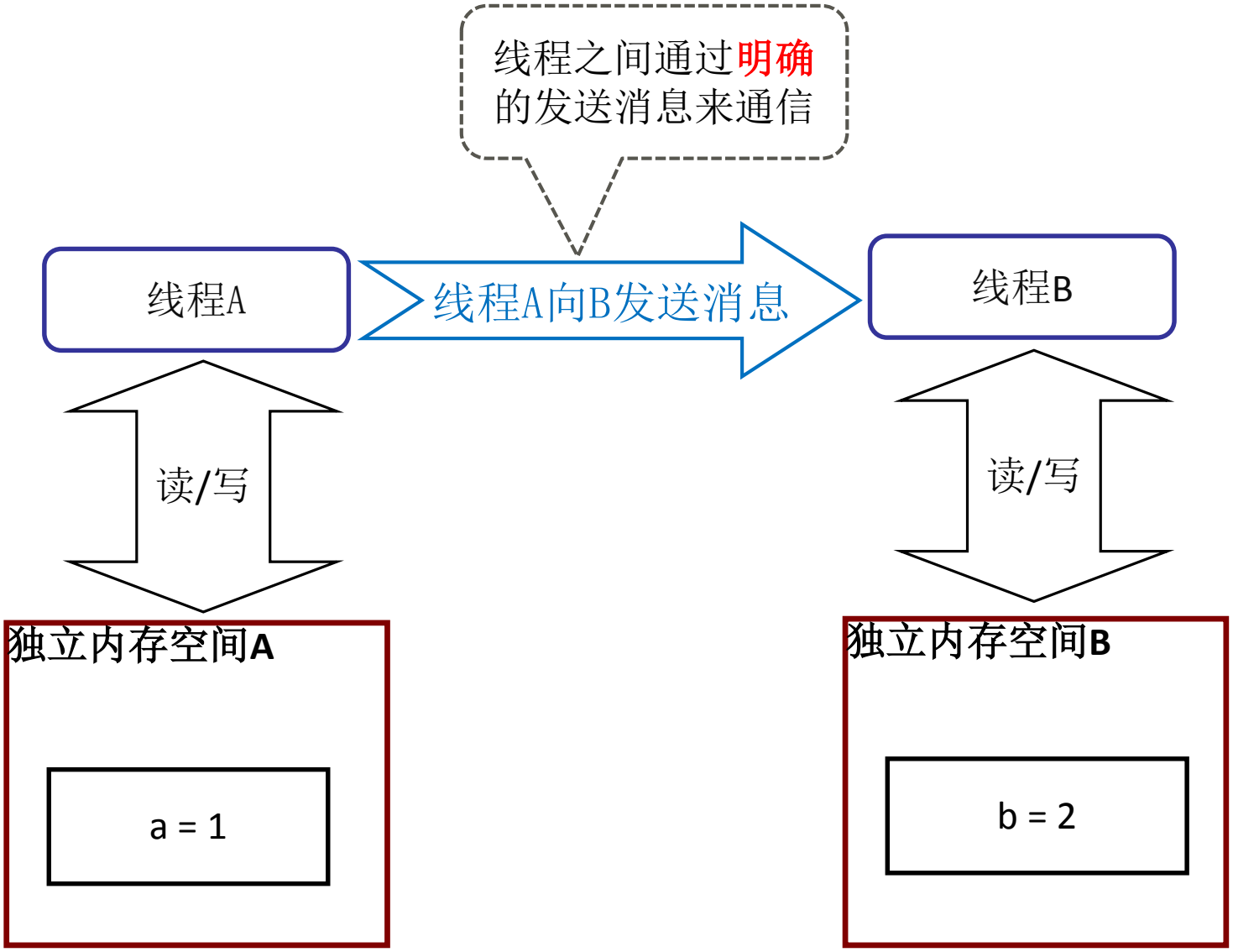
并发编程模型的分类


并发编程
共享内存
Java，C#
Pthread，Win32线程
消息传递
JVM上的Scala
Google的GO
Ericsson的Erlang

共享内存模型的线程间通信



消息传递模型的线程间通信



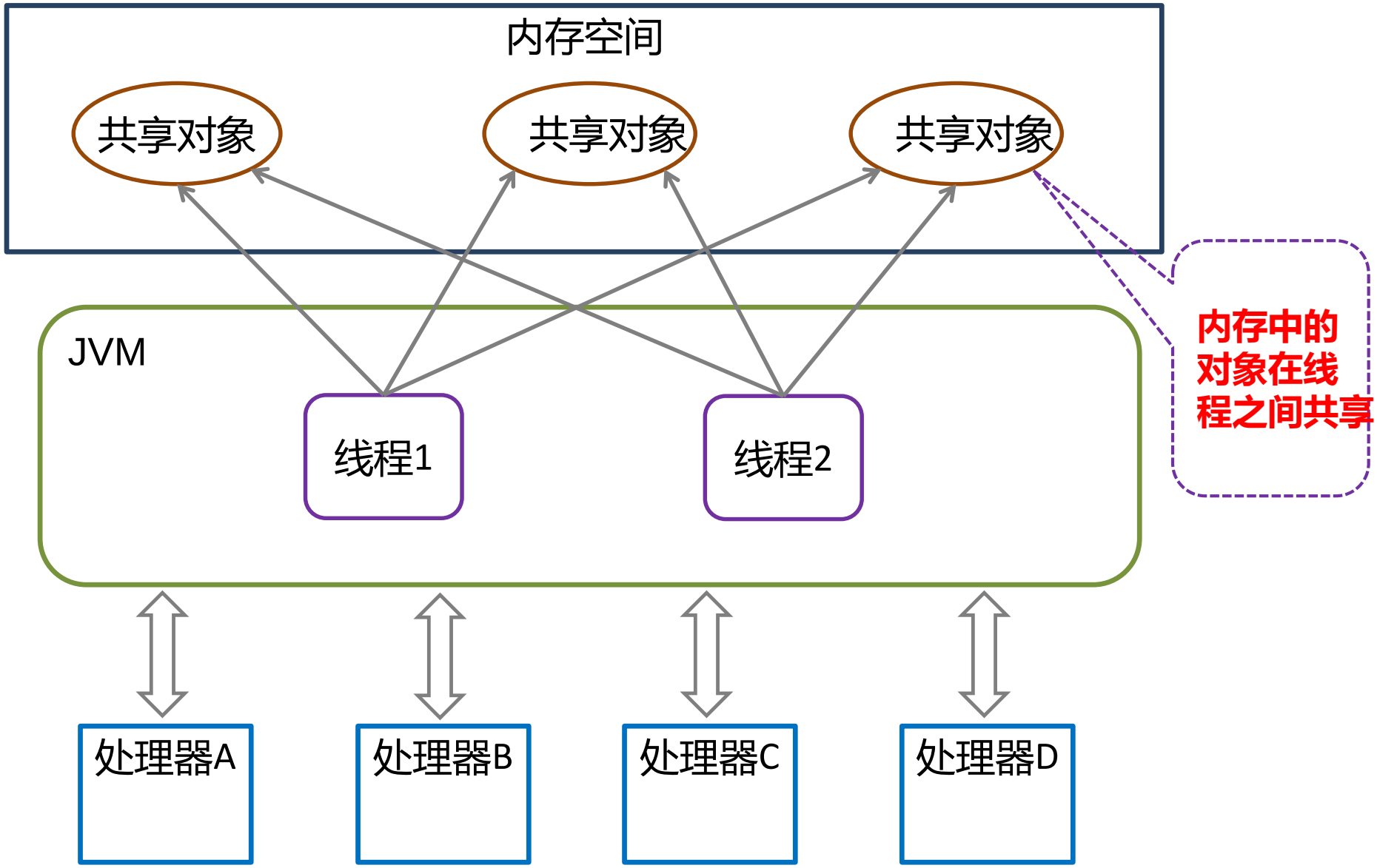
它们就是想对着干！

- 共享内存模型：显式同步，隐式通信
- 消息传递模型：显式通信，隐式同步

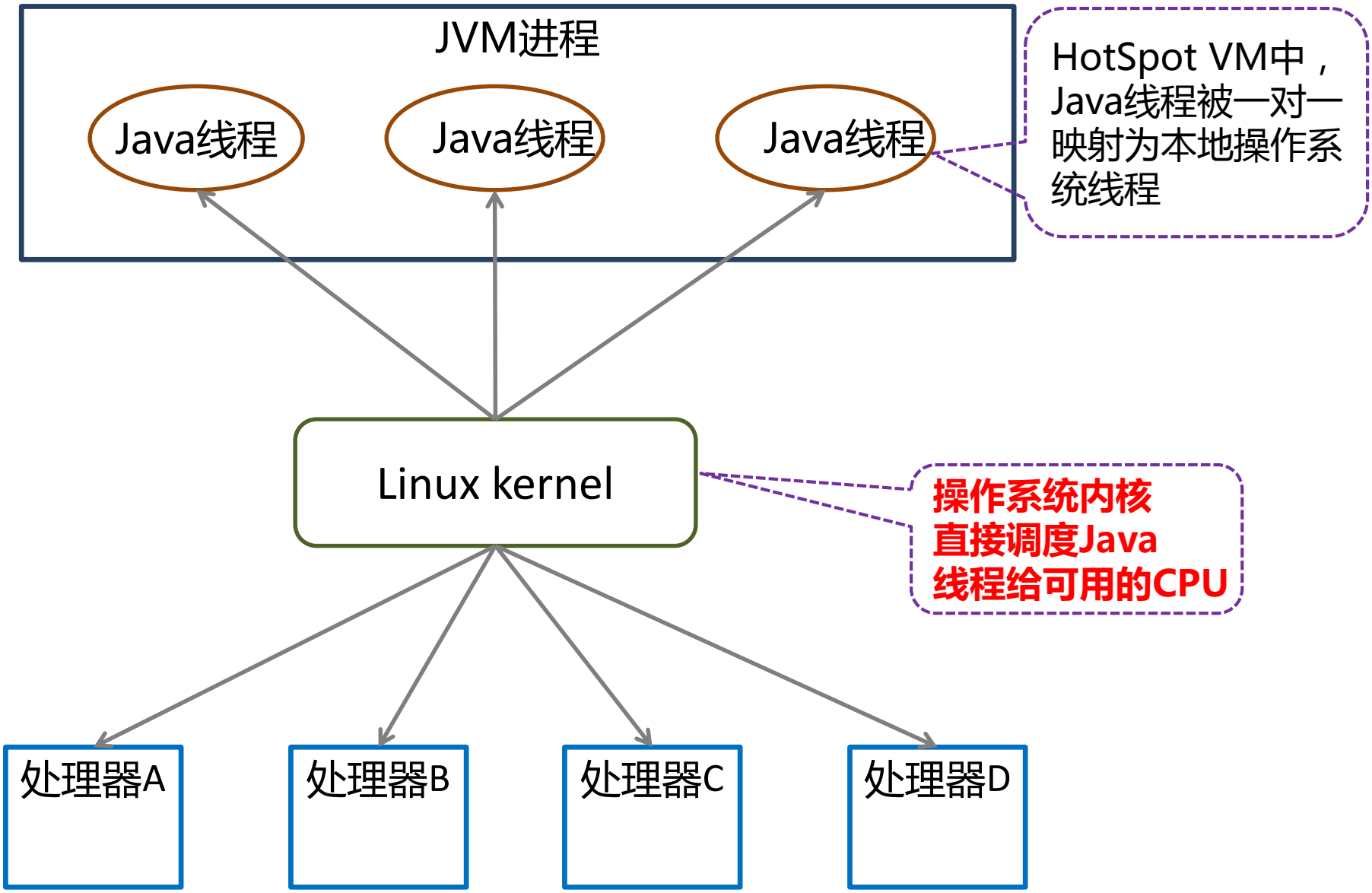
刚好相反！

- Java采用共享内存的并发模型
- Java线程之间的通信总是隐式进行

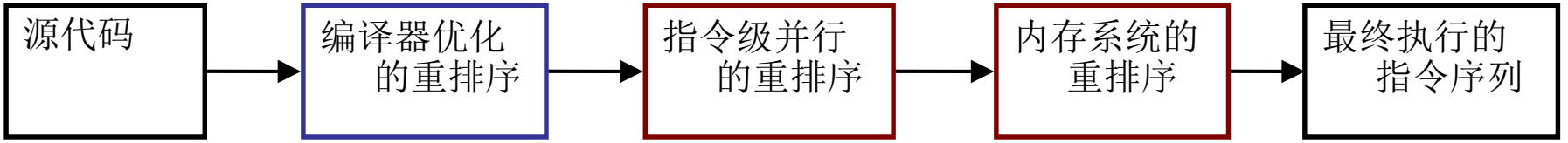
Java并发模型—硬件视图

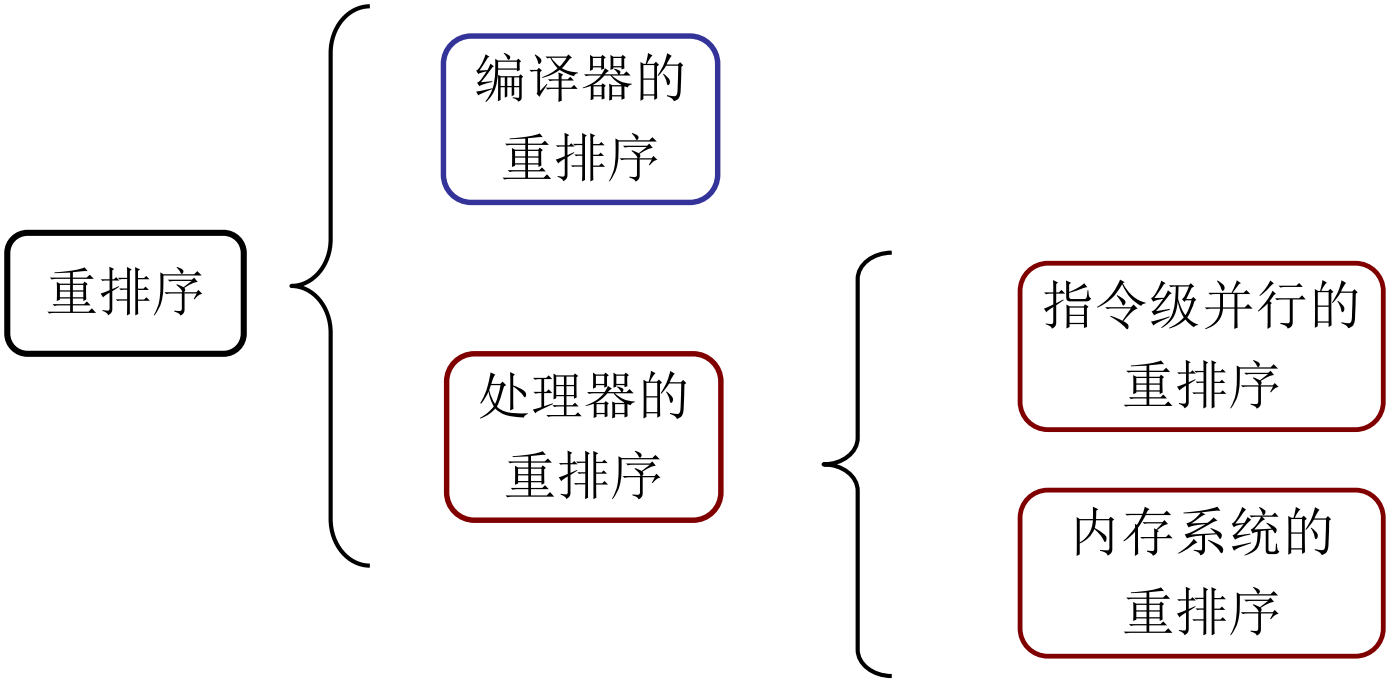


Java并发模型—操作系统视图

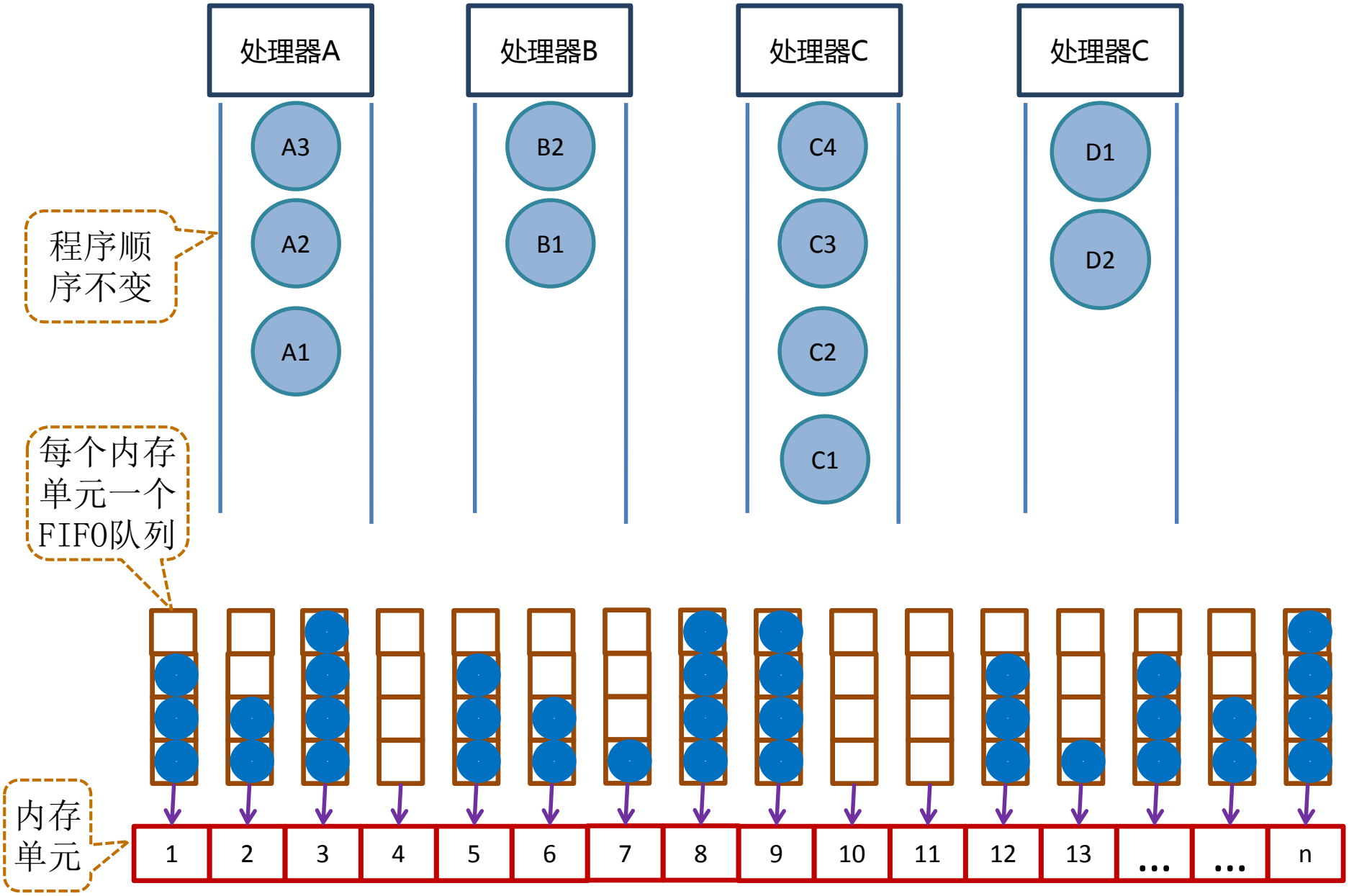


编译器和处理器喜欢不择手段的冒险

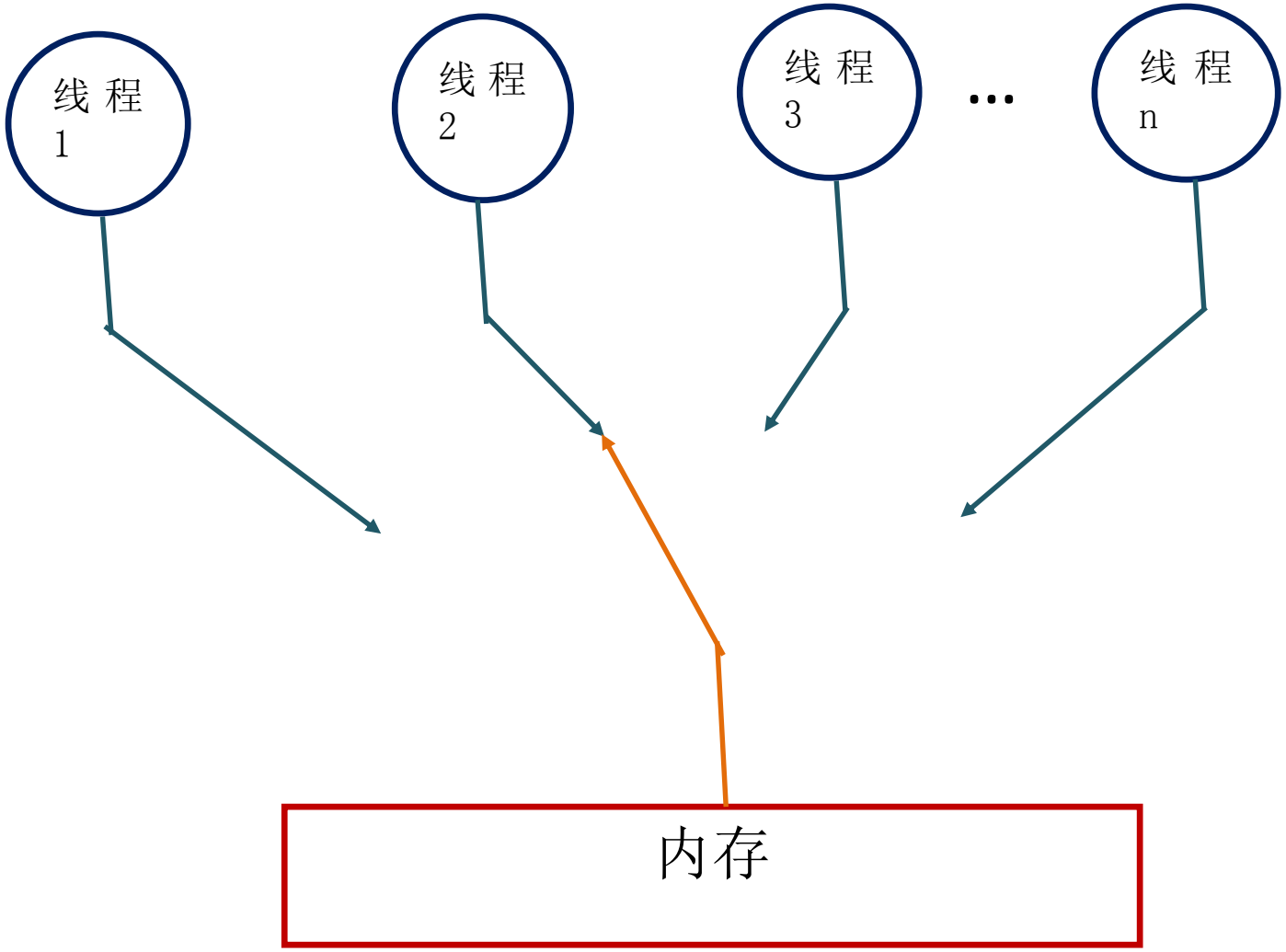




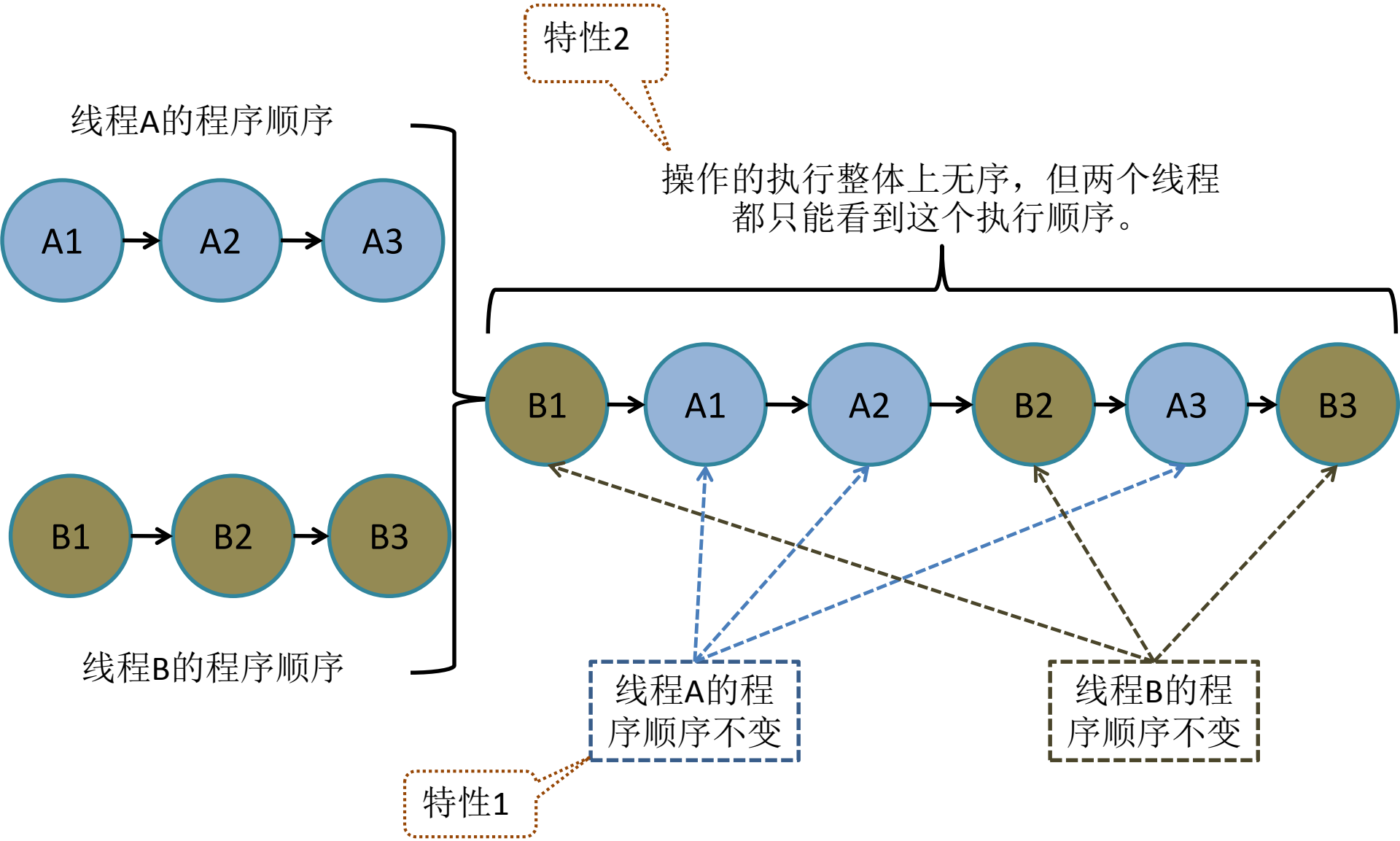
顺序一致性内存模型的原型结构



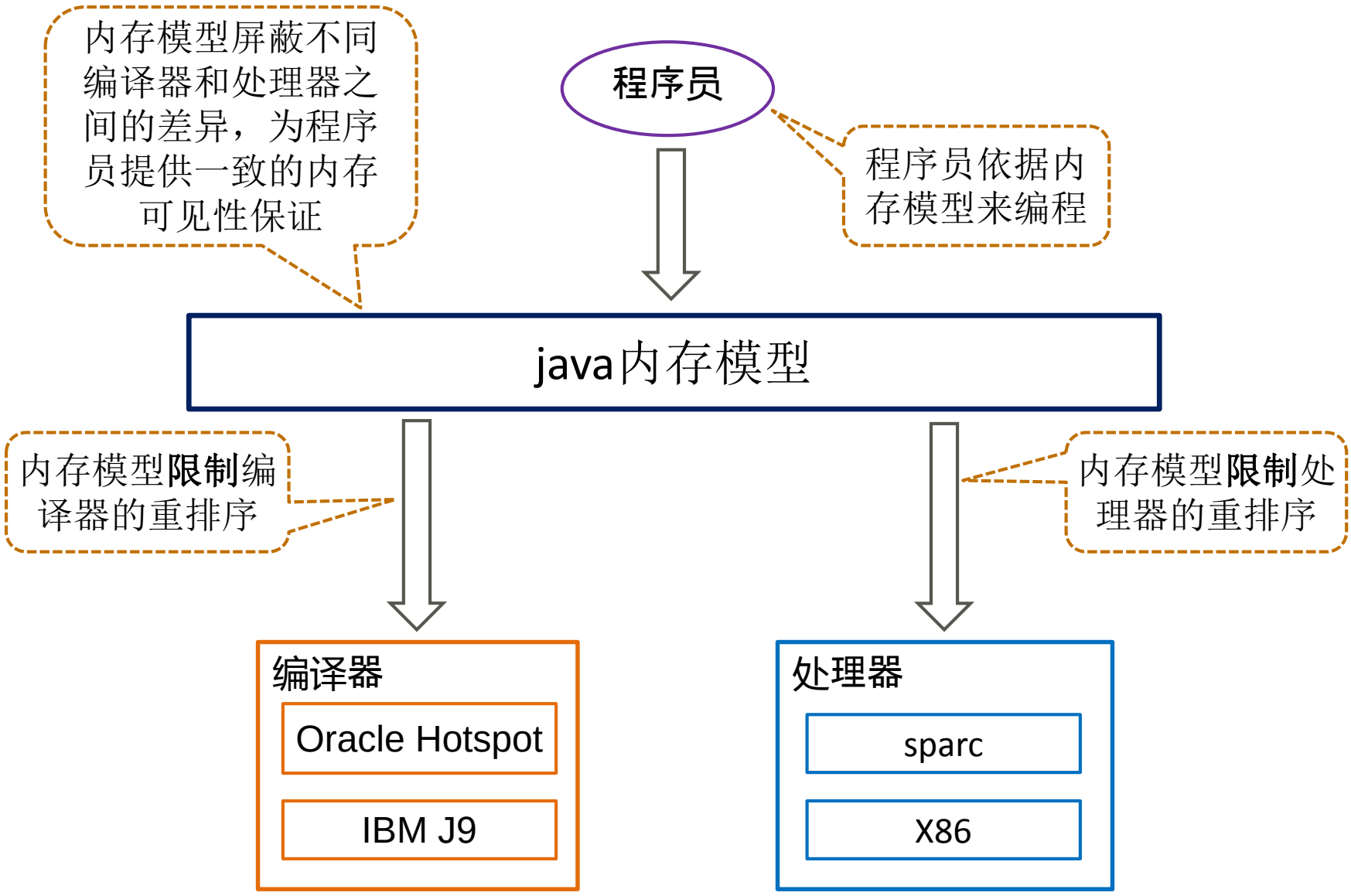
顺序一致性内存模型的程序员视图



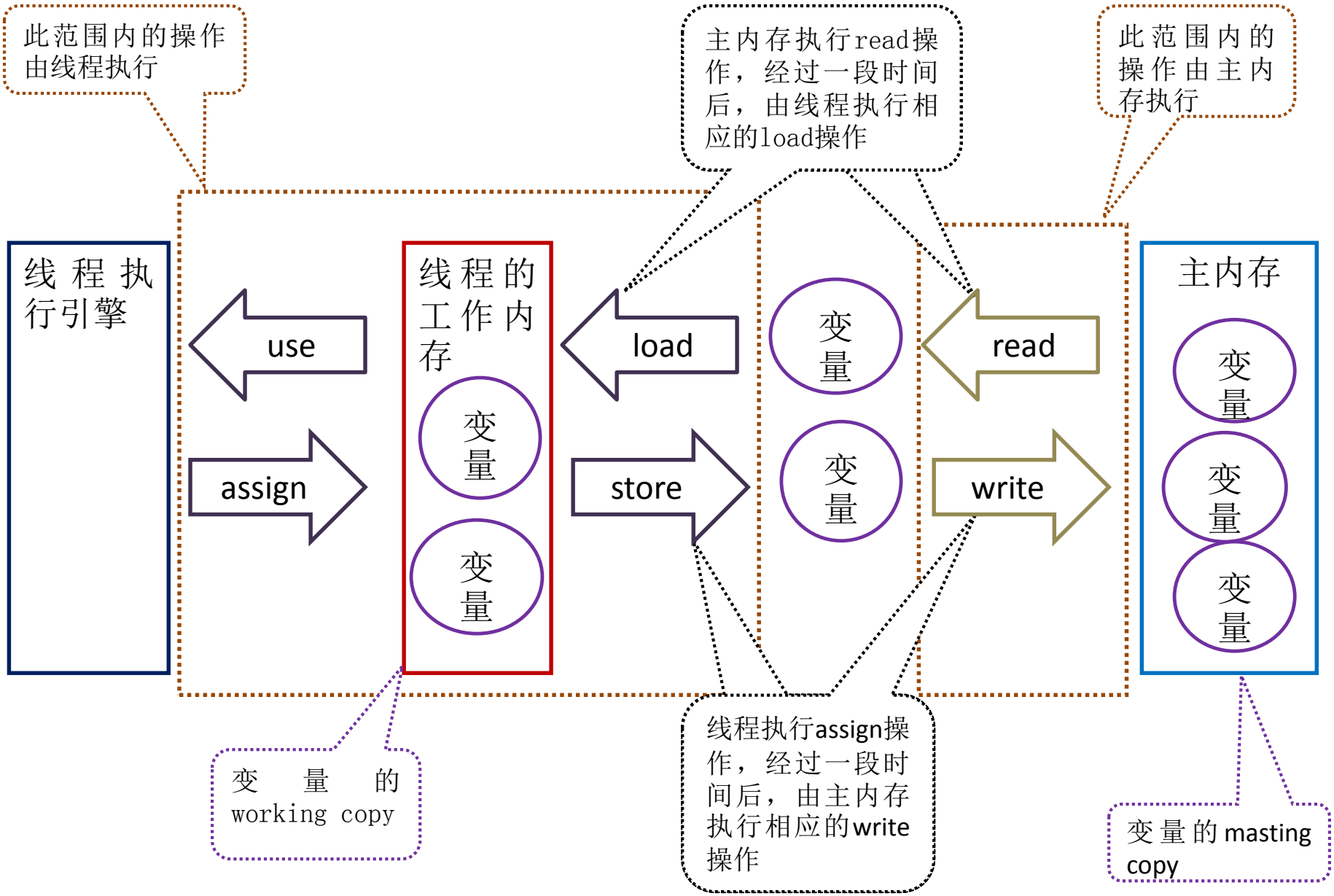
顺序一致性内存模型的2大特性



Java内存模型是一张契约



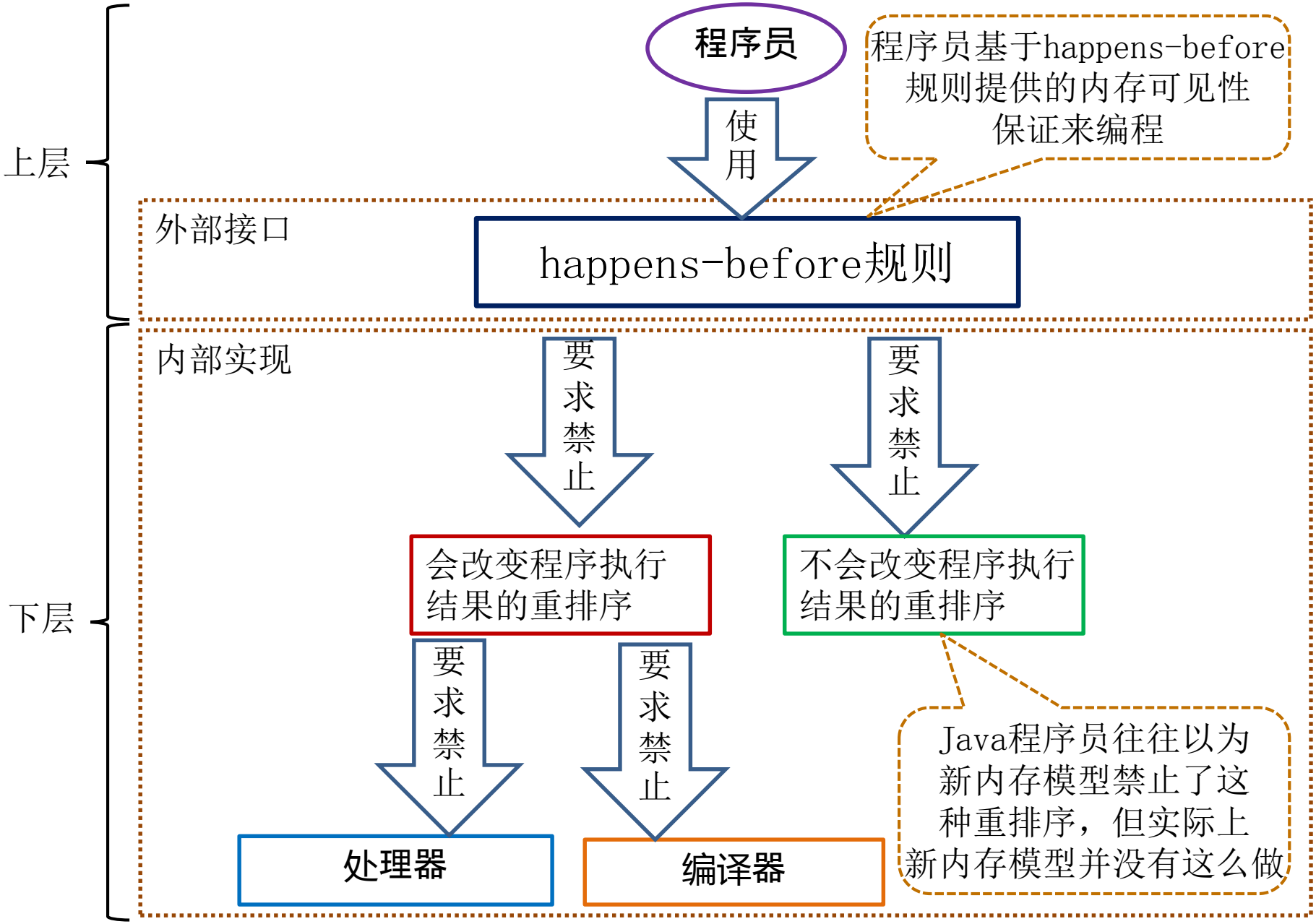
旧内存模型的结构


此范围内的操作由线程执行
主内存执行read操作，经过一段时间后，由线程执行相应的load操作
此范围内的操作由主内存执行
线程执行引擎
use
assign
线程的工作内存
变量
变量
load
store
变量
变量
read
write
主内存
变量
变量
变量
变量的working copy
线程执行assign操作，经过一段时间后，由主内存执行相应的write操作
变量的masting copy

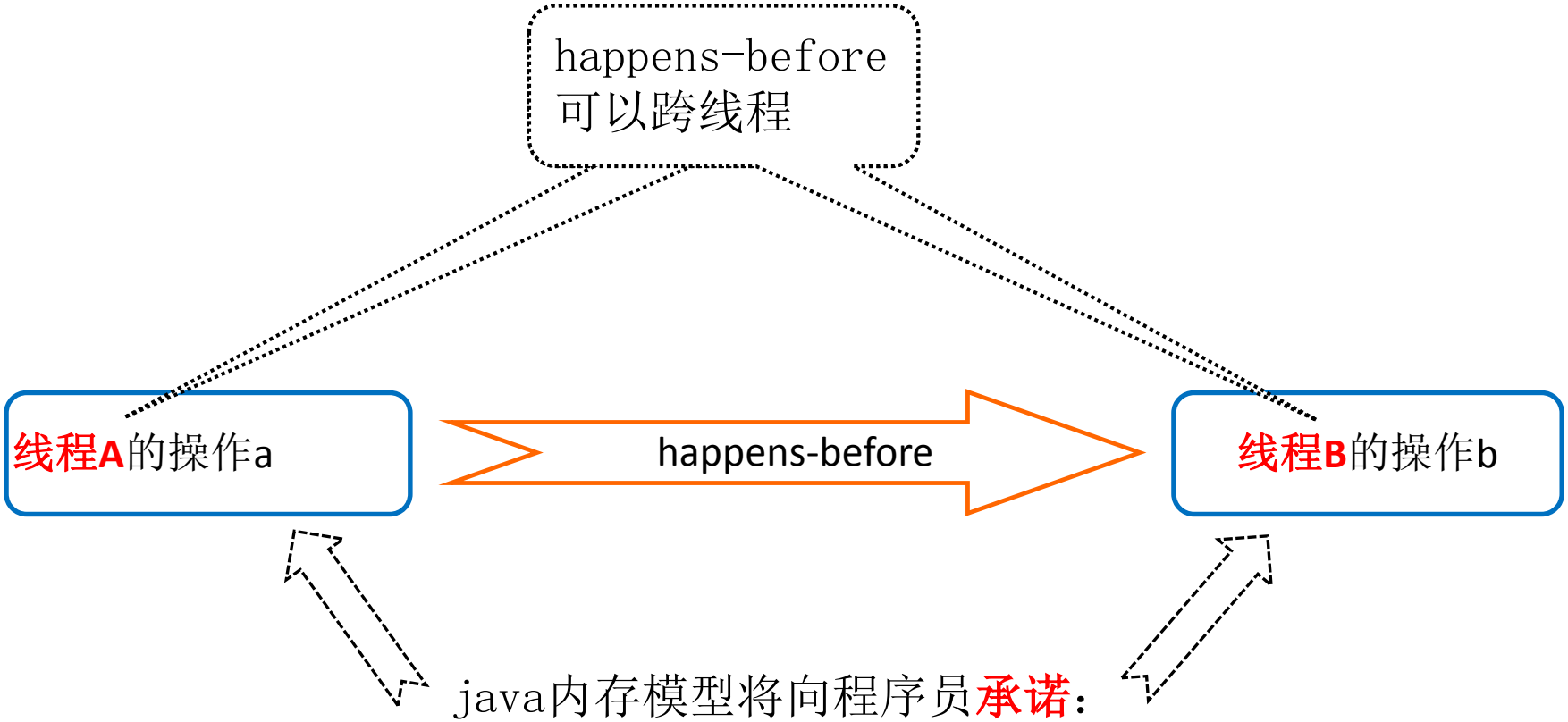
旧内存模型的规则

- 执行顺序和一致性：规则1，规则2，…, 规则n
- 变量的规则：规则1，规则2，…, 规则n
- 锁规则：规则1，规则2，…, 规则n
- 锁和变量的交互作用：规则1，规则2，…, 规则n
- volatile规则：规则1，规则2，…, 规则n

新内存模型的设计


上层
下层
程序员
使用
程序员基于happens-before规则提供的内存可见性保证来编程
外部接口
happens-before规则
内部实现
要求禁止
要求禁止
会改变程序执行结果的重排序
不会改变程序执行结果的重排序
要求禁止
要求禁止
处理器
编译器
Java程序员往往以为新内存模型禁止了这种重排序，但实际上新内存模型并没有这么做

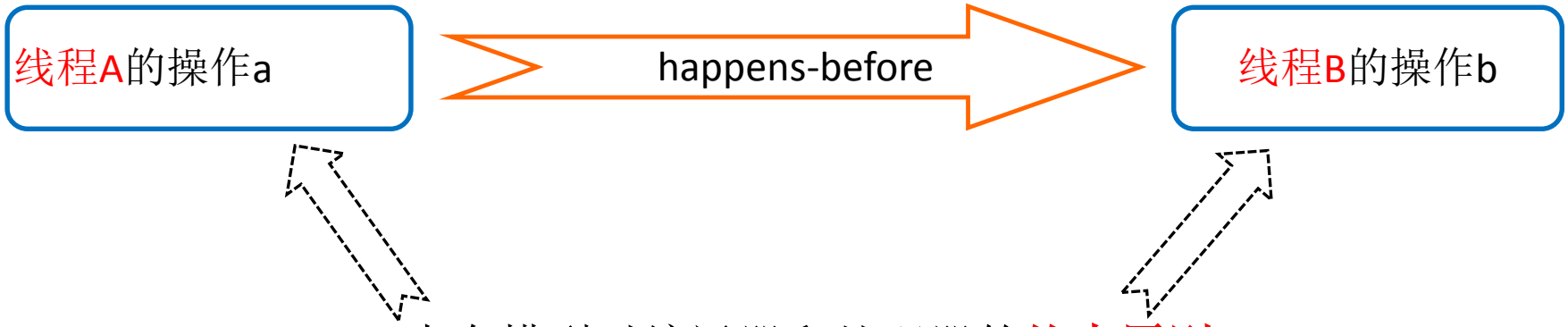
happens-before对程序员的承诺



java内存模型将向程序员**承诺**：

a操作的结果将对b可见，且a的执行顺序排在b之前。

happens-before对编译器和处理器的约束



java内存模型对编译器和处理器的**约束原则**：

如果不会改变程序的执行结果

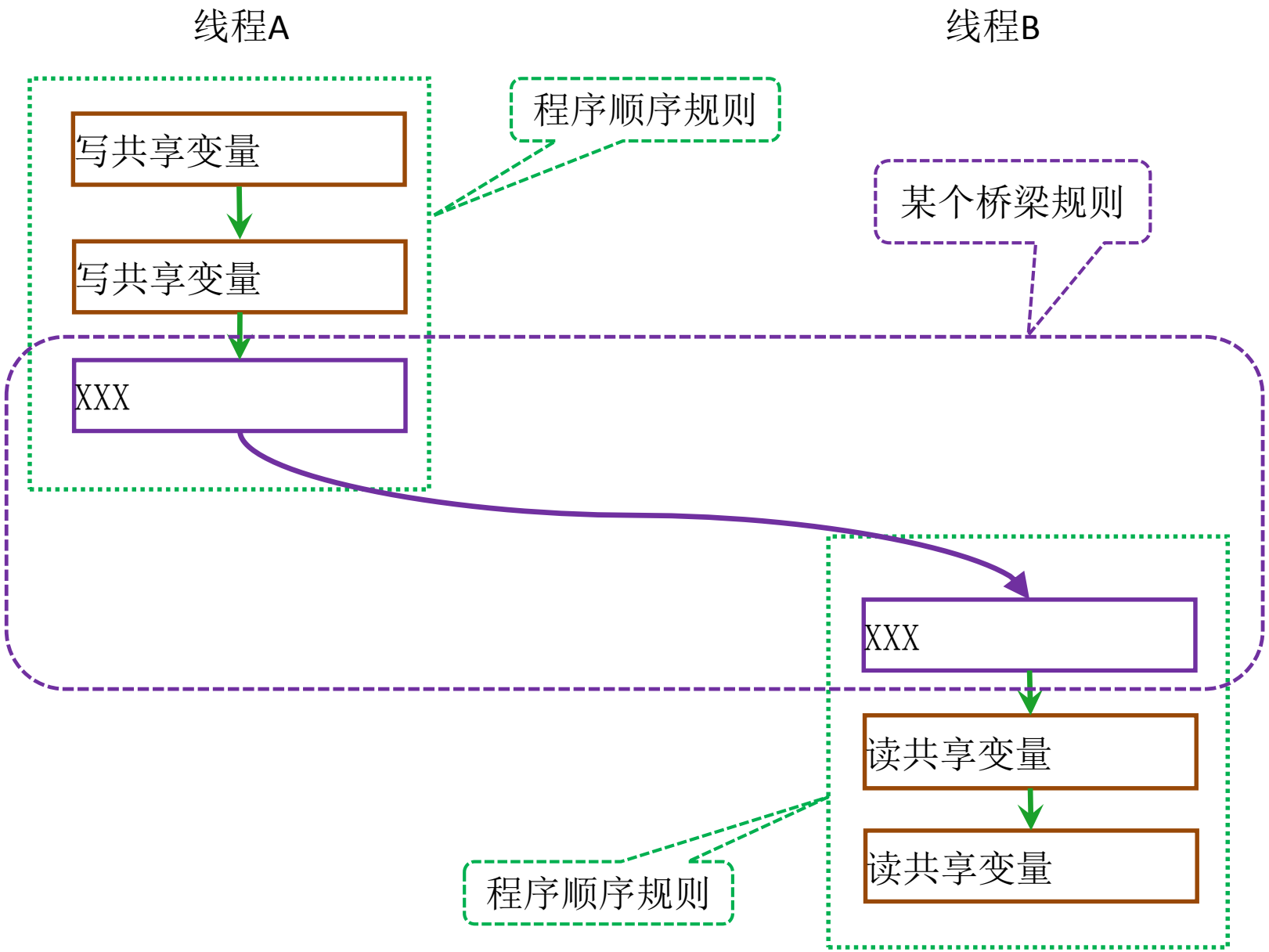
（指的是单线程程序和正确同步的多线程程序），

编译器和处理器怎么优化都行。

happens-before规则

- **程序顺序规则：**一个线程中的每个操作，happens- before 于该线程中的任意后续操作。
- **锁规则：**对一个锁的解锁，happens- before 于后续对这个锁的加锁。
- **volatile规则：**对一个volatile变量的写，happens- before 于后续对这个volatile变量的读。
- **start()规则：**如果线程A执行操作ThreadB.start()（启动线程B），那么A线程的ThreadB.start()操作happens- before于线程B中的任意操作。
- **join()规则：**如果线程A执行操作ThreadB. join()并成功返回，那么线程B中的任意操作happens-before于线程A从ThreadB. join()操作成功返回。
- **传递性：**如果A happens- before B，且B happens- before C，那么A happens- before C。

happens-before的通用模式



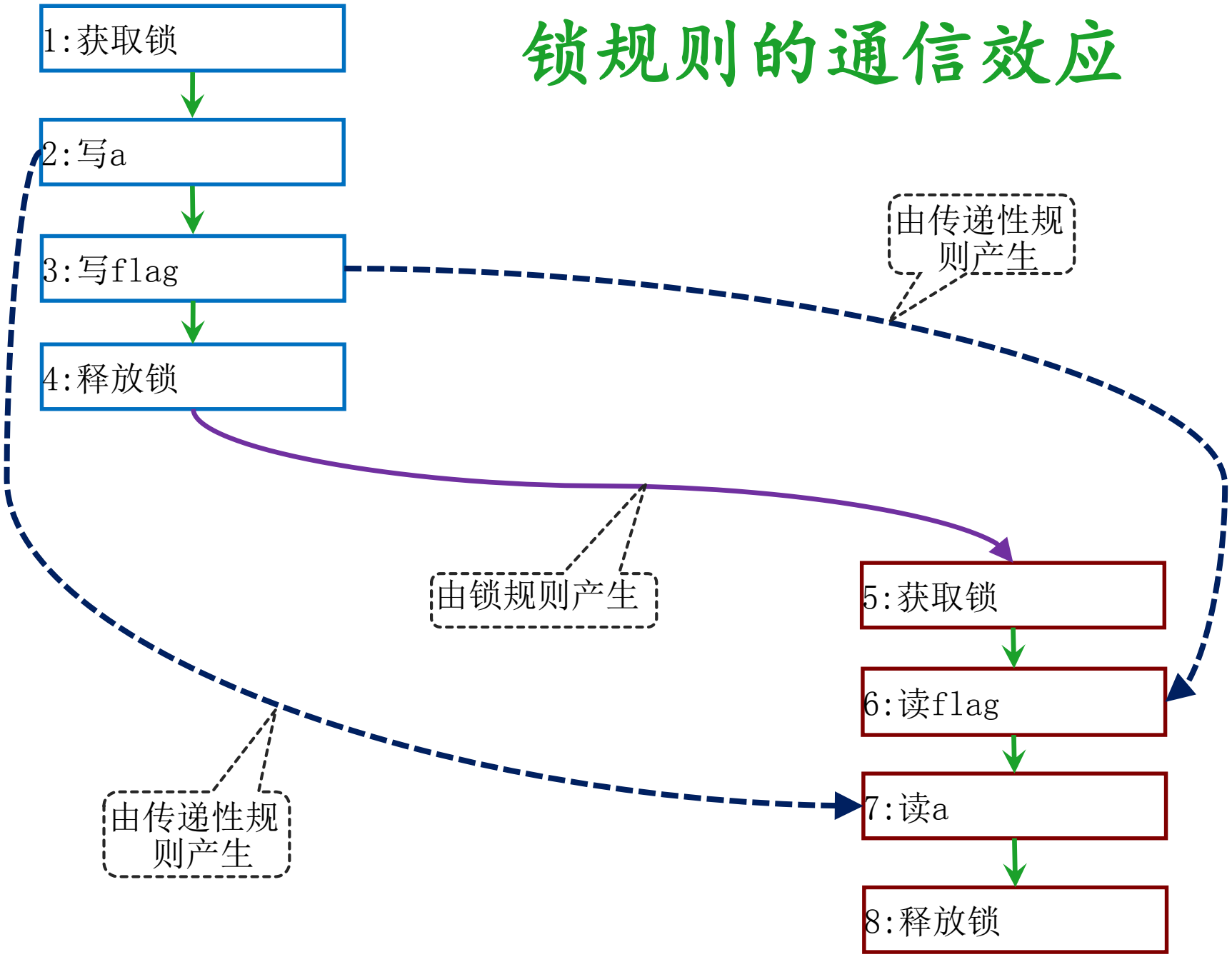
锁规则的案例

```
class SynchronizedExample {
    int a = 0;
    boolean flag = false;

    public synchronized void writer() { //1，写线程执行
        a = 1;                          //2
        flag = true;                    //3
    }                                   //4，释放锁

    public synchronized void reader() { //5，读线程执行，获取锁
        if (flag) {                     //6
            int i = a;                  //7
        }
    }                                   //8
}
```

写线程先执行

读线程后执行

锁规则的通信效应


1:获取锁
2:写a
3:写flag
4:释放锁
由传递性规则产生
由锁规则产生
5:获取锁
6:读flag
7:读a
8:释放锁
由传递性规则产生

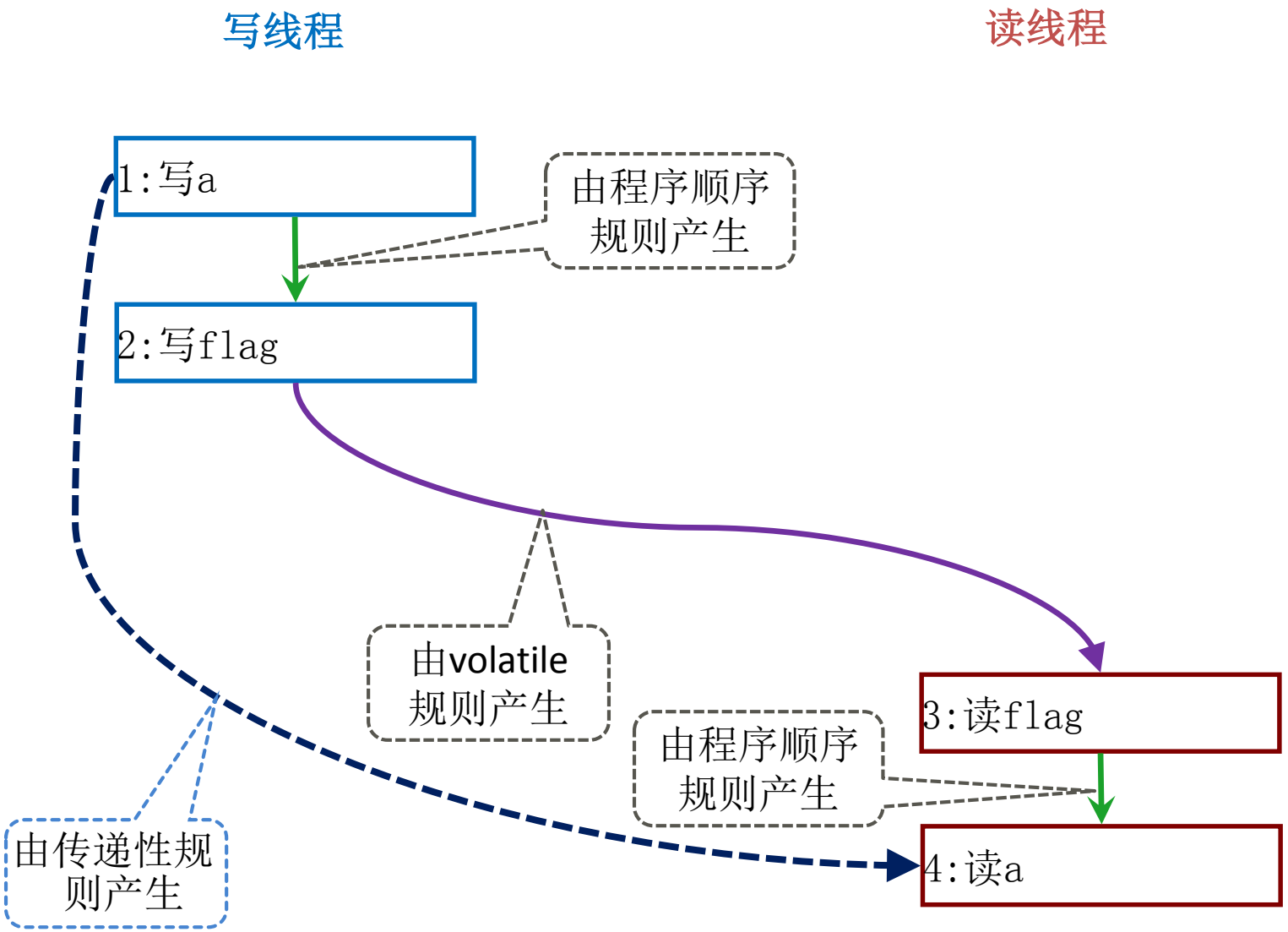
volatile规则的案例

```
class VolatileExample {
    int                     a = 0;
    volatile boolean flag = false;  // volatile变量

    public void writer() {     //写线程执行
        a = 1;                 //1
        flag = true;           //2，volatile写
    }

    public void reader() {     //读线程执行
        if (flag) {            //3，volatile读
            int i =  a;        //4
        }
    }
}
```

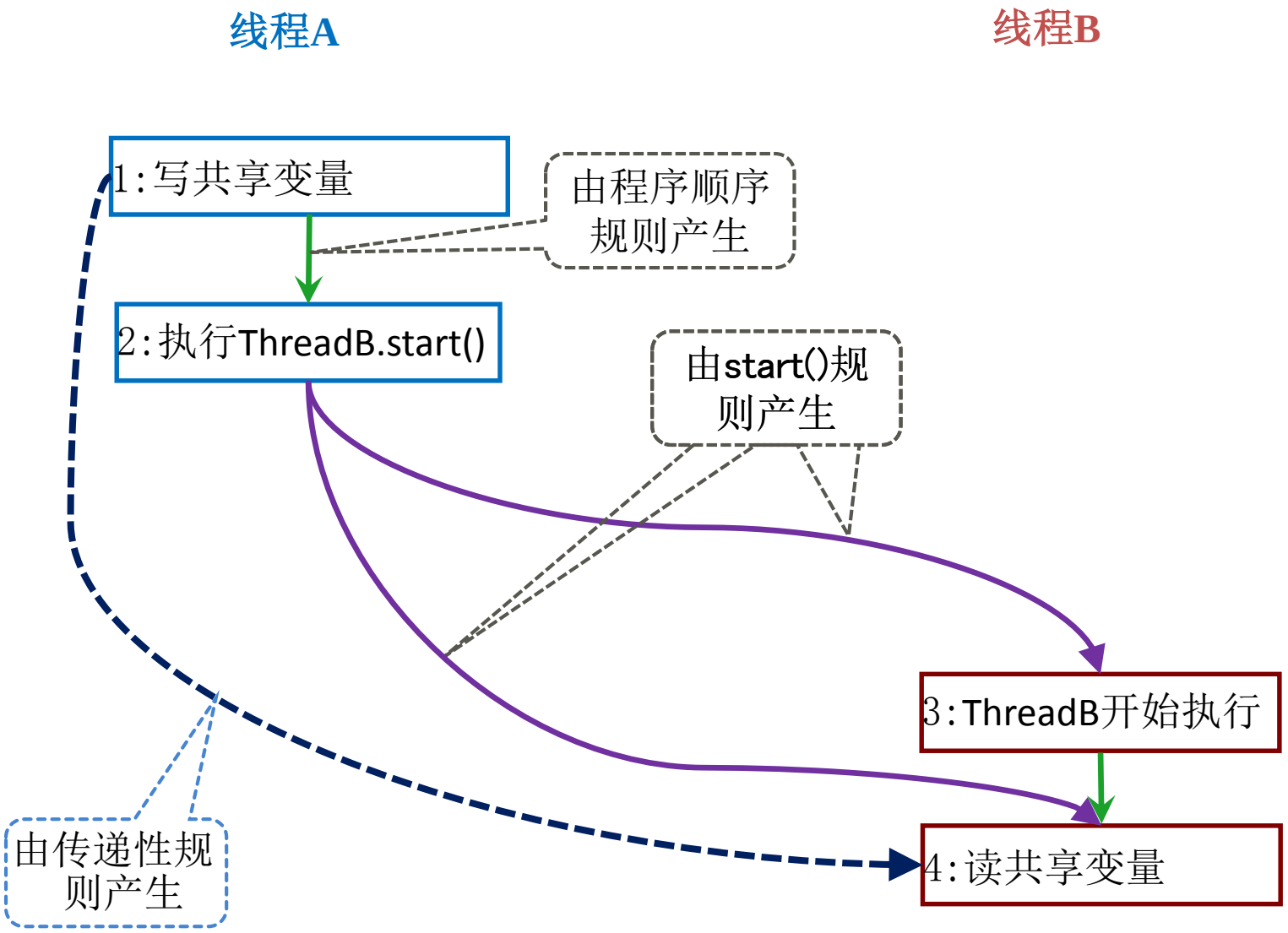
写线程先执行

读线程后执行

volatile规则的通信效应



start()规则的通信效应



join()规则的通信效应