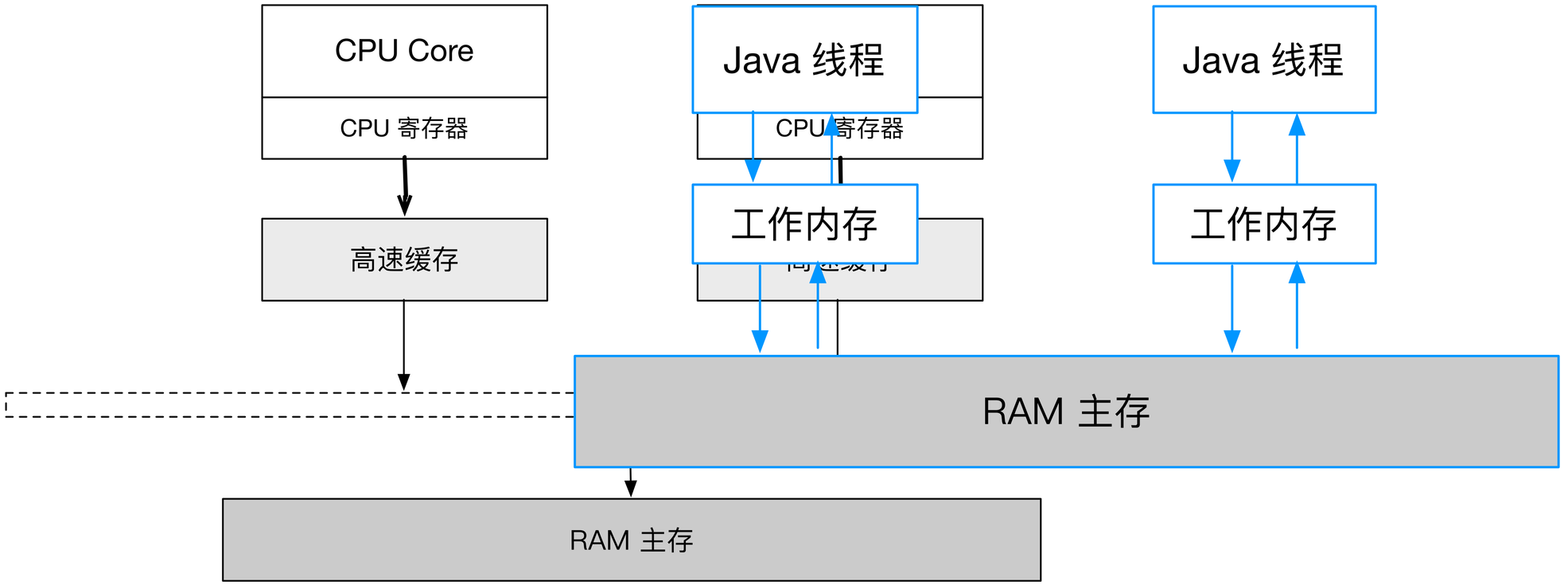
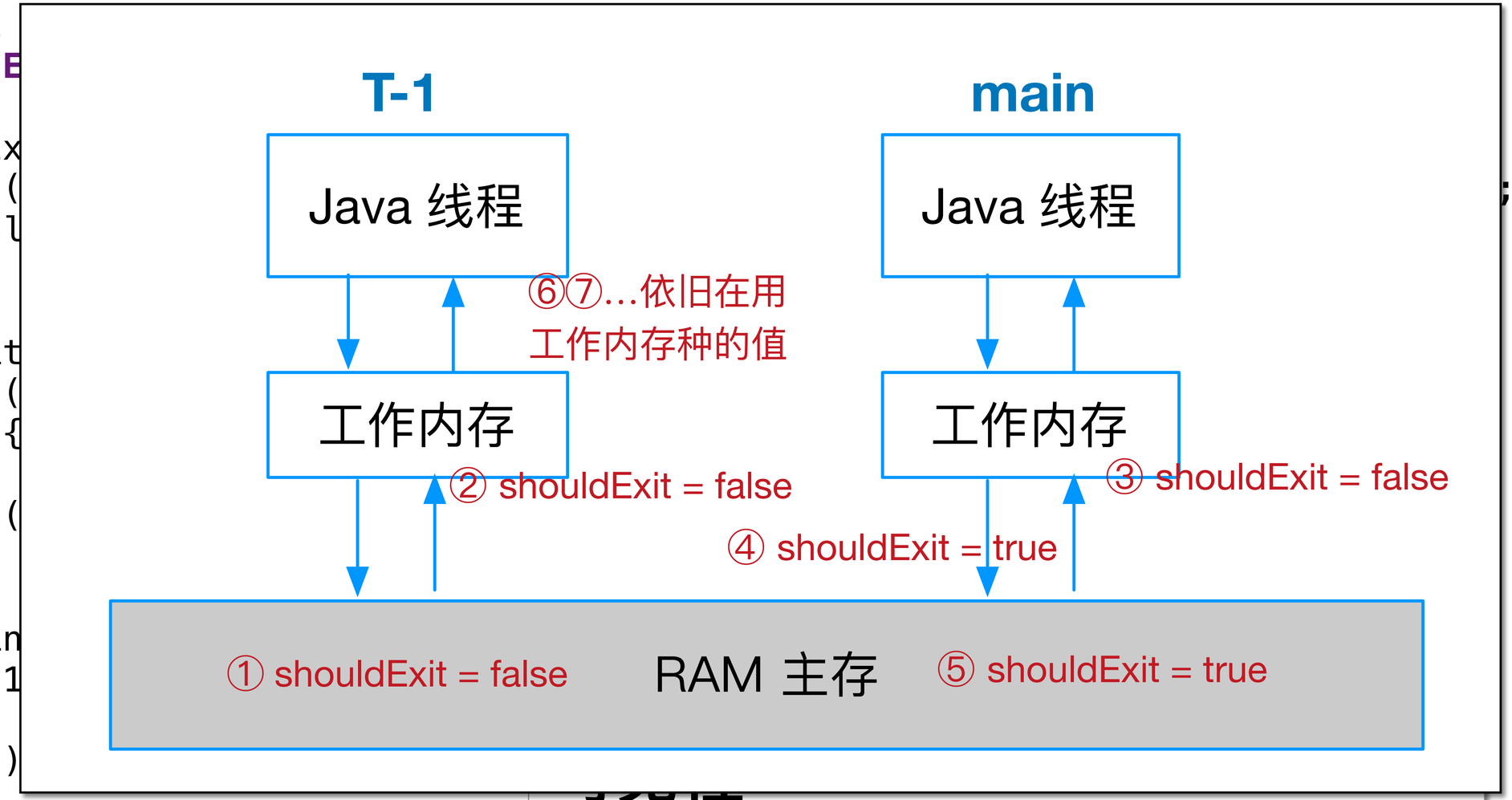
Java 内存模型

可见性

计算机组成和Java内存模型(JMM)



```
public class VolatileTest1
    private boolean shouldE

    public void setShouldEx
        System.out.println(
        shouldExit = newVal
    }

    public void waitForExit
        System.out.println(
        while(!shouldExit){
        }
        System.out.println(
    }

    public static void main
        final VolatileTest1
        new Thread(() -> {
            vt.waitForExit()
        }, "T-1").start();

        vt.setShouldExit(true);
    }
}
```



当一个线程修改了线程共享变量的值，
其它线程能够立即得知这个修改

可见性

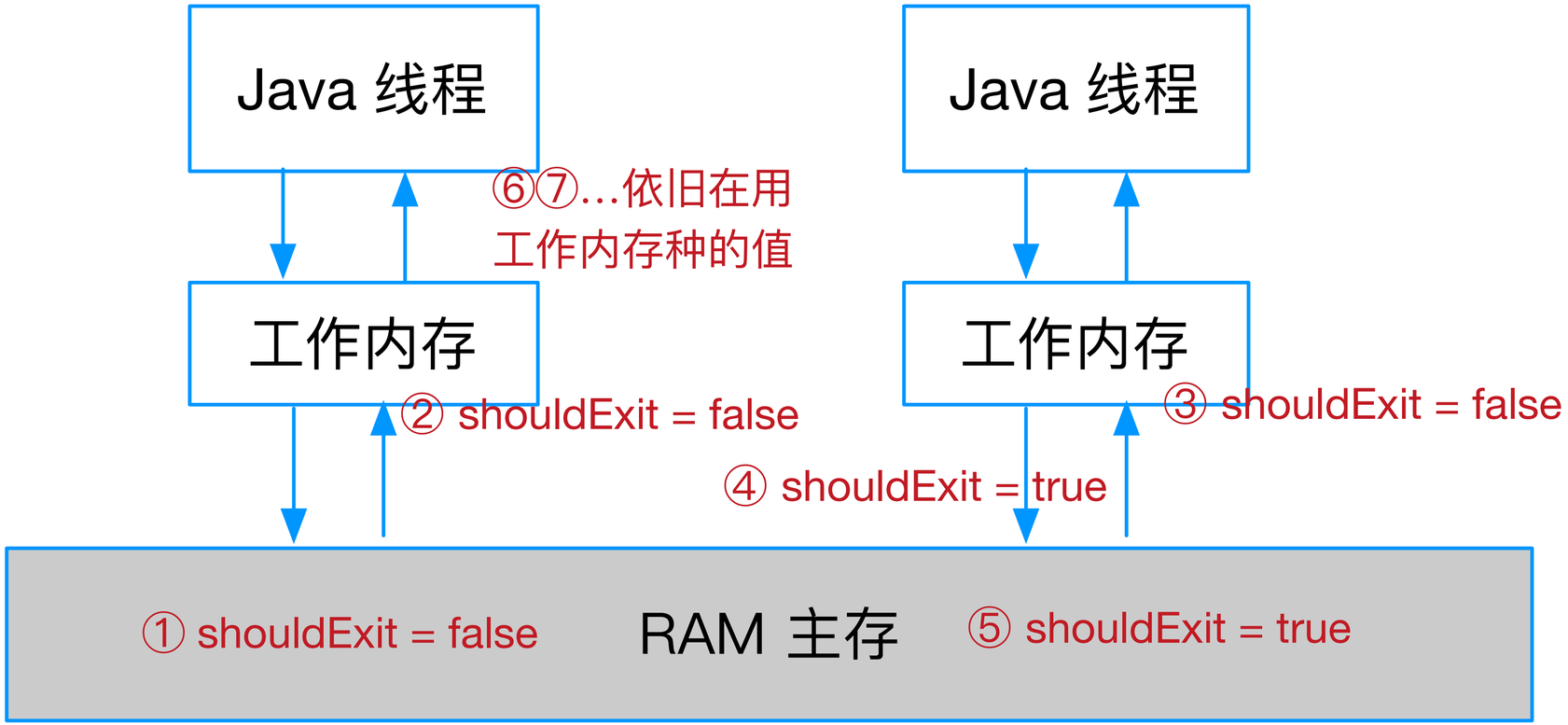
- 多个线程访问同一个变量时，一个线程修改了变量的值，其它线程能够立即看到修改后的值
- Java通过volatile关键字来保证可见性
  - volatile关键字会强制将修改的值立即写入主存
  - 线程读取volatile变量值时会去主存读取，以便获取到最新的值
- 通过sychronized / Lock也可实现可见性

```
public class VolatileTest1 {
    private volatile boolean shouldExit = false;

    public void setShouldExit(boolean newValue){
        System.out.println("[" + Thread.currentThread().getName() + "] setShouldExit “ + newValue);
        shouldExit = newValue;
    }

    public void waitForExit(){
        System.out.println("[" + T                    ne() + "] waitingForExit()");
        while(!shouldExit){
        }
        System.out.println("[" + T                    ne() + "] waitForExit end");
    }

    public static void main(String[] argvs) throws Exception{
        final VolatileTest1 vt = new VolatileTest1();
        for(int i=0; i<2; i++){
            new Thread(new Runnable(){
               @Override
                public void run() {
                    vt.waitForExit();
                }
            }, "T-" + i).start();
        }
        vt.setShouldExit(true);
    }
}
```

Thread.sleep(100);
Thread.yield();
System.out.println(“something”);

Java并发编程

- 原子性
  - 一个操作是不可中断的。即使是在多个线程一起执行的时候，一个操作一旦开始，就不会被其它线程干扰
- 可见性
  - 当多个线程访问同一个变量时，一个线程修改了这个变量的值，其他线程能够立即看得到修改的值
- 有序性
  - 程序执行的顺序按照代码的先后顺序执行


Java 线程
Java 线程
⑥⑦...依旧在用
工作内存种的值
工作内存
工作内存
② shouldExit = false
③ shouldExit = false
④ shouldExit = true
① shouldExit = false
RAM 主存
⑤ shouldExit = true