操作系统
第七讲

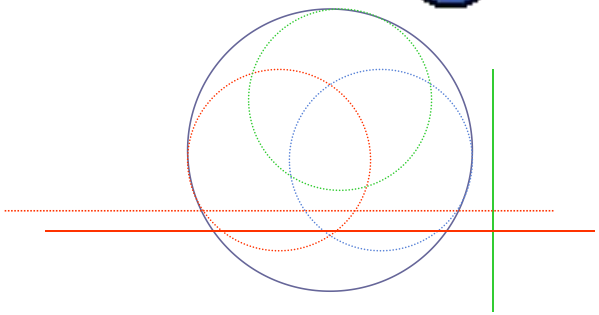
张涛

软件与微电子学院

Review

实时调度

多处理机调度



3.5 线程及其管理

线程的引入

进程和线程的比较

操作系统对线程的实现

线程举例

3.5.1 线程的引入

- 进程：资源分配单位（存储器、文件）和CPU调度（分派）单位。
  - 进程是拥有自己资源的单元体。
  - 进程是被调度分派在处理器上运行的单元体。
- 缺点：时间空间开销大，限制并发度的提高
- 线程：作为CPU调度单位，而进程只作为其他资源分配单位。
  - 只拥有必不可少的资源，如：线程状态、寄存器上下文和栈
  - 同样具有就绪、阻塞和执行三种基本状态

线程的概念

线程的概念

- 线程是进程内一个相对独立的、可调度的执行单元。
- 进程中的一个运行实体，是一个CPU调度单位
- 资源的拥有者还是进程

将原来进程的两个属性分开处理

多线程机制

- 一个进程可以有多个线程，这些线程共享进程资源，驻留在相同的地址空间，共享数据和文件。
- 一个线程修改了一个数据项，其他线程可以读取和使用此结果数据。一个线程打开并读一个文件时同一进程中的其他线程也可以同时读此文件。
- 这些线程运行在同一进程的相同的地址空间内。

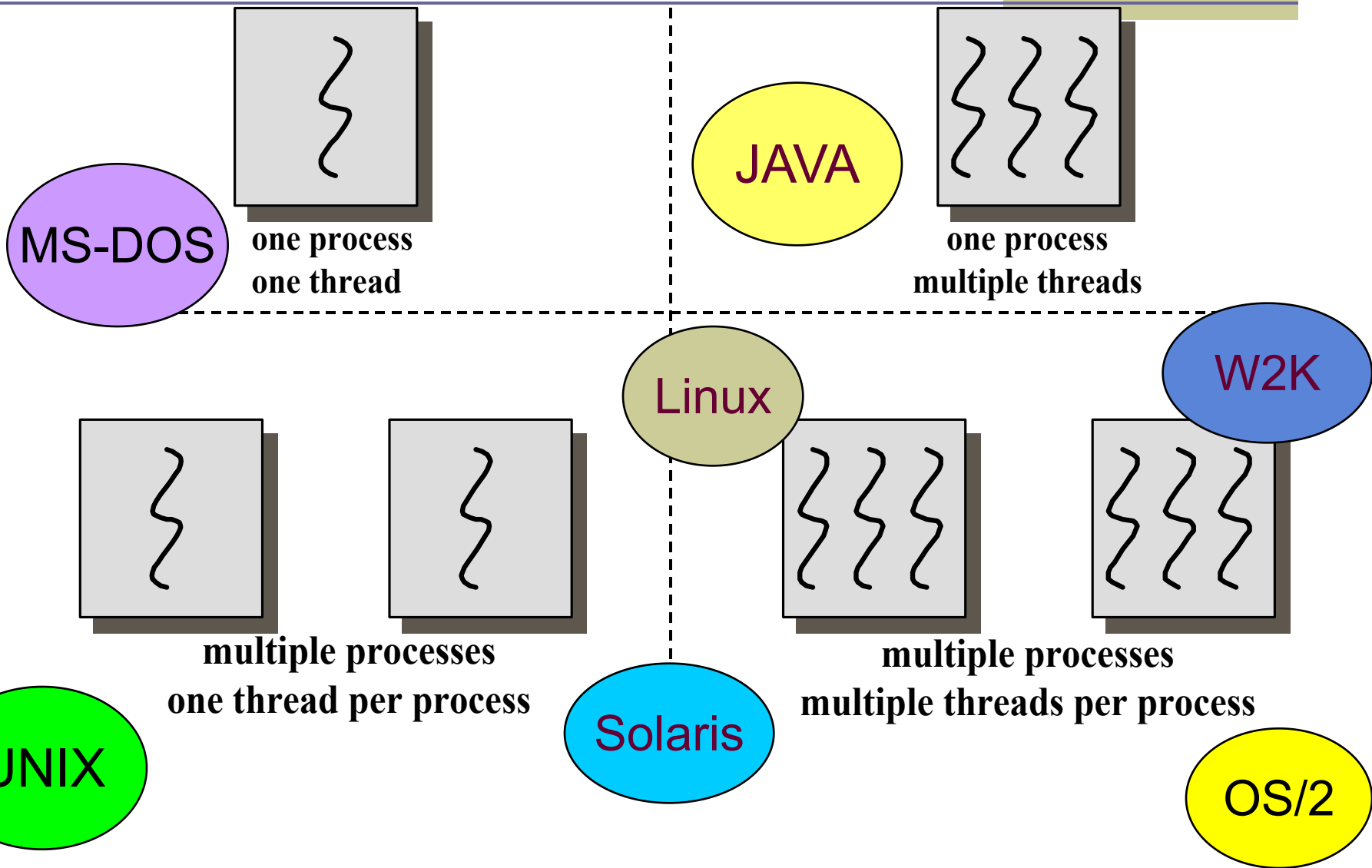
引入线程的好处

- 创建一个新线程花费时间少（结束亦如此）
- 两个线程的切换花费时间少（如果机器设有“存储[恢复]所有寄存器”指令，则整个切换过程用几条指令即可完成）
- 因为同一进程内的线程共享内存和文件，因此它们之间相互通信无须调用内核
- 适合多处理机系统

3.5.2 进程和线程的比较

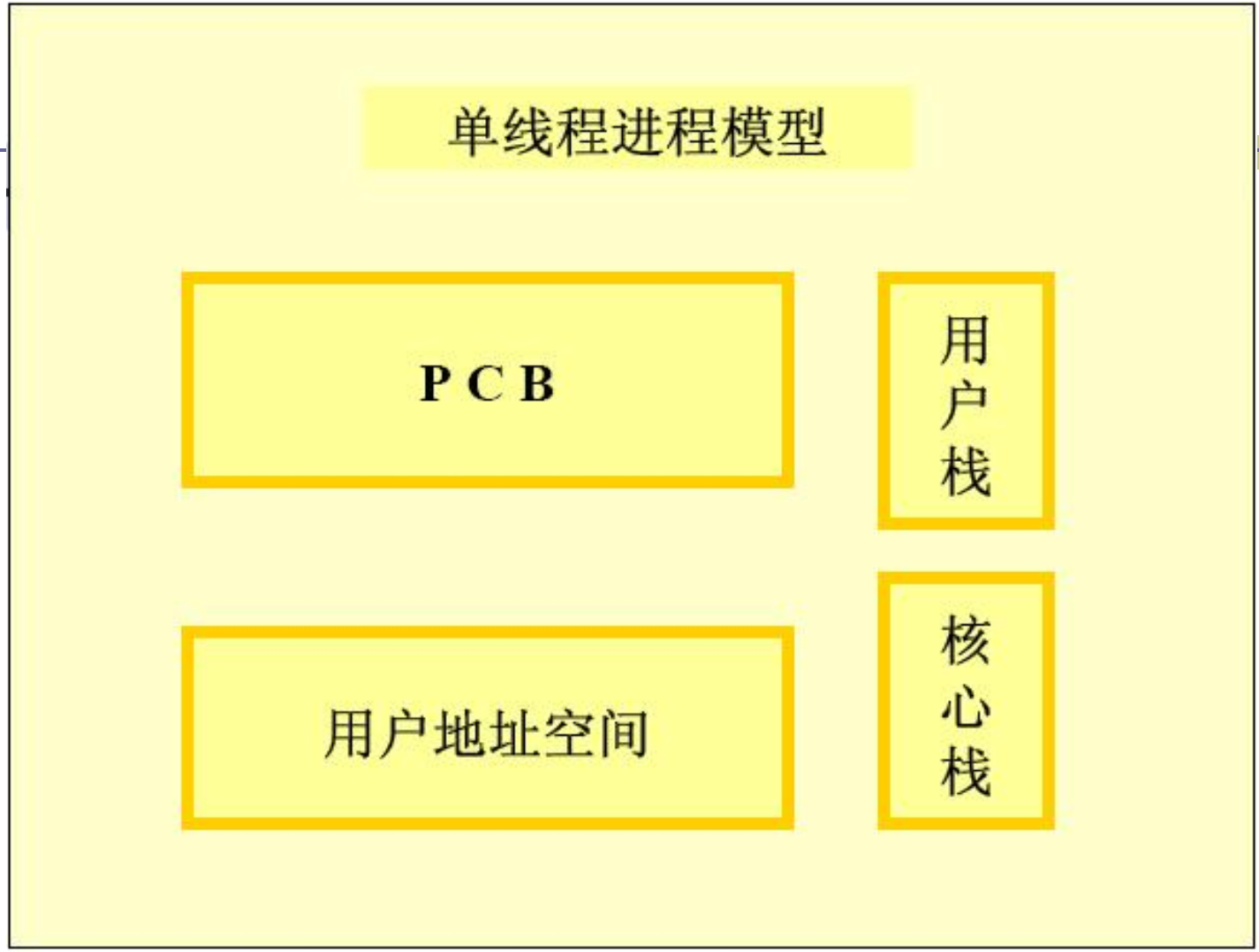
- 调度：线程上下文切换比进程上下文切换要快得多；
  - 线程的创建时间比进程短；终止时间比进程短；
  - 同进程内的线程切换时间比进程短；
- 拥有资源：进程间相互独立，同一进程的各线程间资源共享——某进程内的线程在其他进程不可见。
  - 由于同进程内线程间共享内存和文件资源，可直接进行不通过内核的通信；
- 系统开销：线程减小并发执行的时间和空间开销
- 并发性：在系统中建立更多的线程来提高并发程度。

SMP & 提高进程执行效率



多线程OS中的进程

- 多线程OS中的进程有以下属性：
  - 作为系统资源分配的单位。
  - 可包括多个线程。
  - 进程不是一个可执行的实体。
- 通信：进程间通信IPC，线程间可以直接读写进程数据段（如全局变量）来进行通信——需要同步和互斥手段的辅助以保证数据的一致性


单线程进程模型
PCB
用户栈
用户地址空间
核心栈

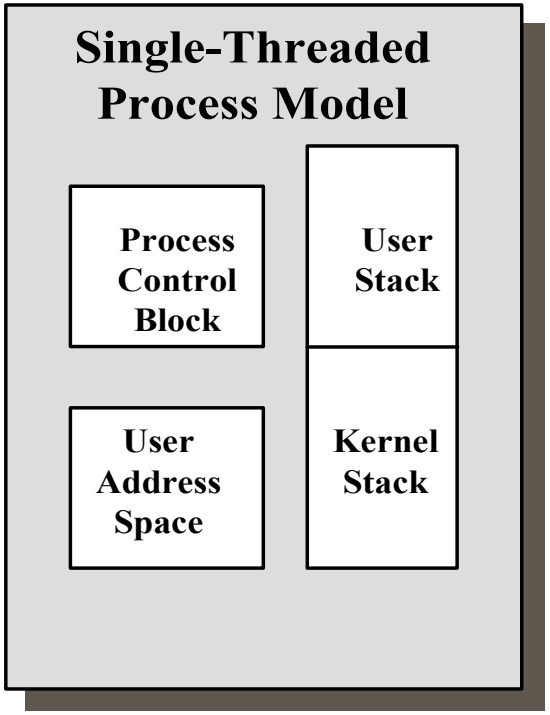
多线程进程模型

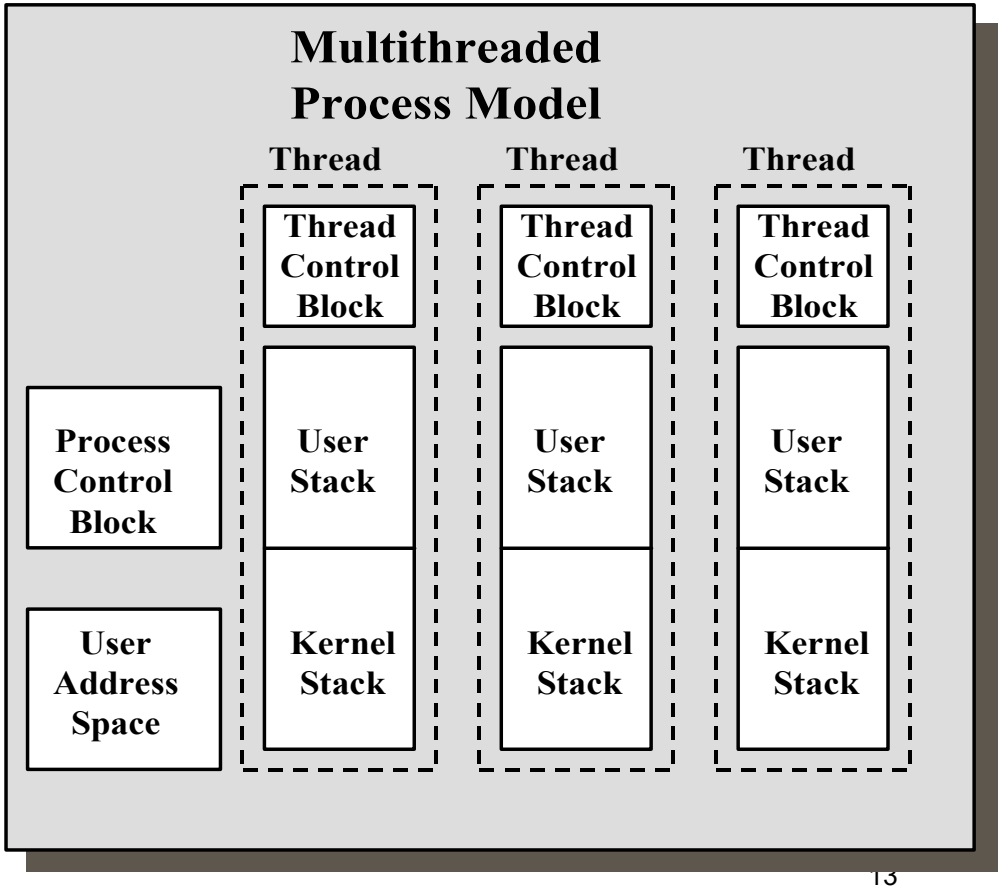
进程和线程

- 资源所有权（Process）：组织资源的实体单位
- 调度的实体（Thread）：调度、使用CPU的实体单位

进程和线程

- 资源所有权（Process）
- 调度的实体（Thread）





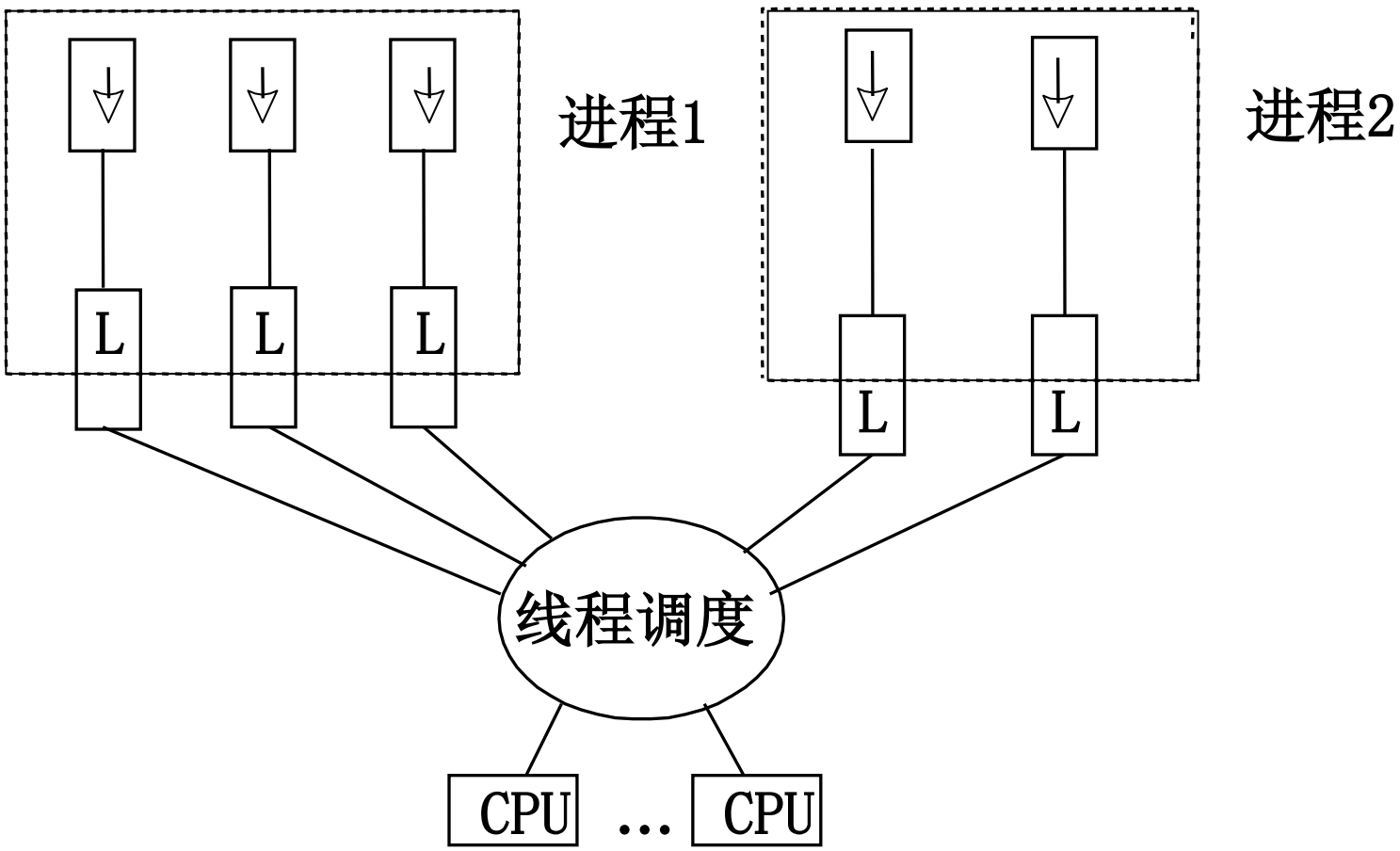
3.5.3 OS对线程的实现方式

- 核心级线程（内核线程，kernel-level thread）
  - 由操作系统内核进行管理。操作系统内核给应用程序提供相应的系统调用和应用程序接口API，以使用户程序可以创建、执行、撤消线程。
- 用户级线程（用户线程，User-level thread）
  - 管理过程全部由用户程序完成，操作系统内核心只对进程进行管理。

内核线程KLT

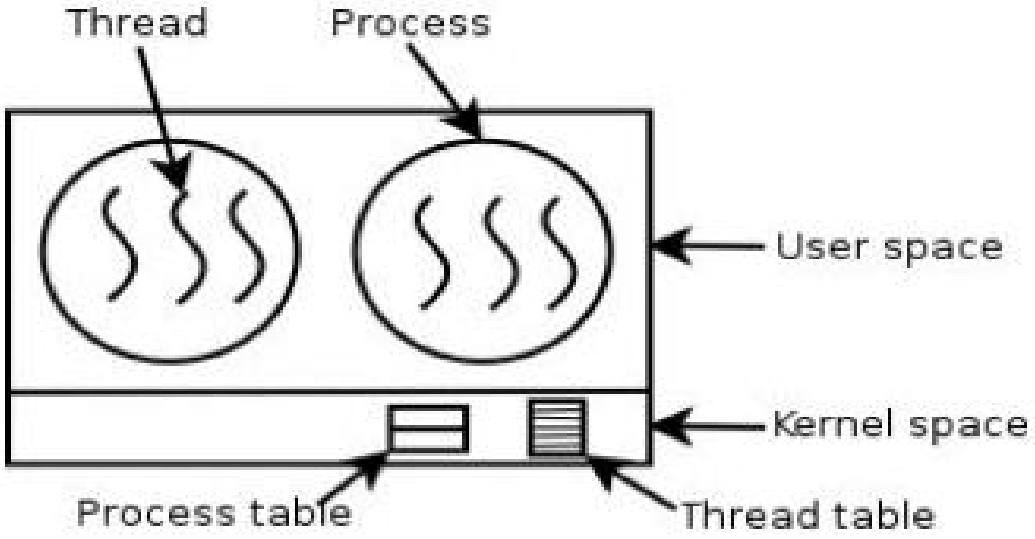
- 依赖于OS核心，由内核的内部需求进行创建和撤销，用来执行一个指定的函数。
  - 内核维护进程和线程的上下文信息；
  - 线程切换由内核完成；
  - 一个线程发起系统调用而阻塞，不会影响其他线程的运行。
  - 时间片分配给线程，所以多线程的进程获得更多CPU时间。
  - Windows NT和OS/2支持内核线程；

内核级线程



内核线程优点和缺点

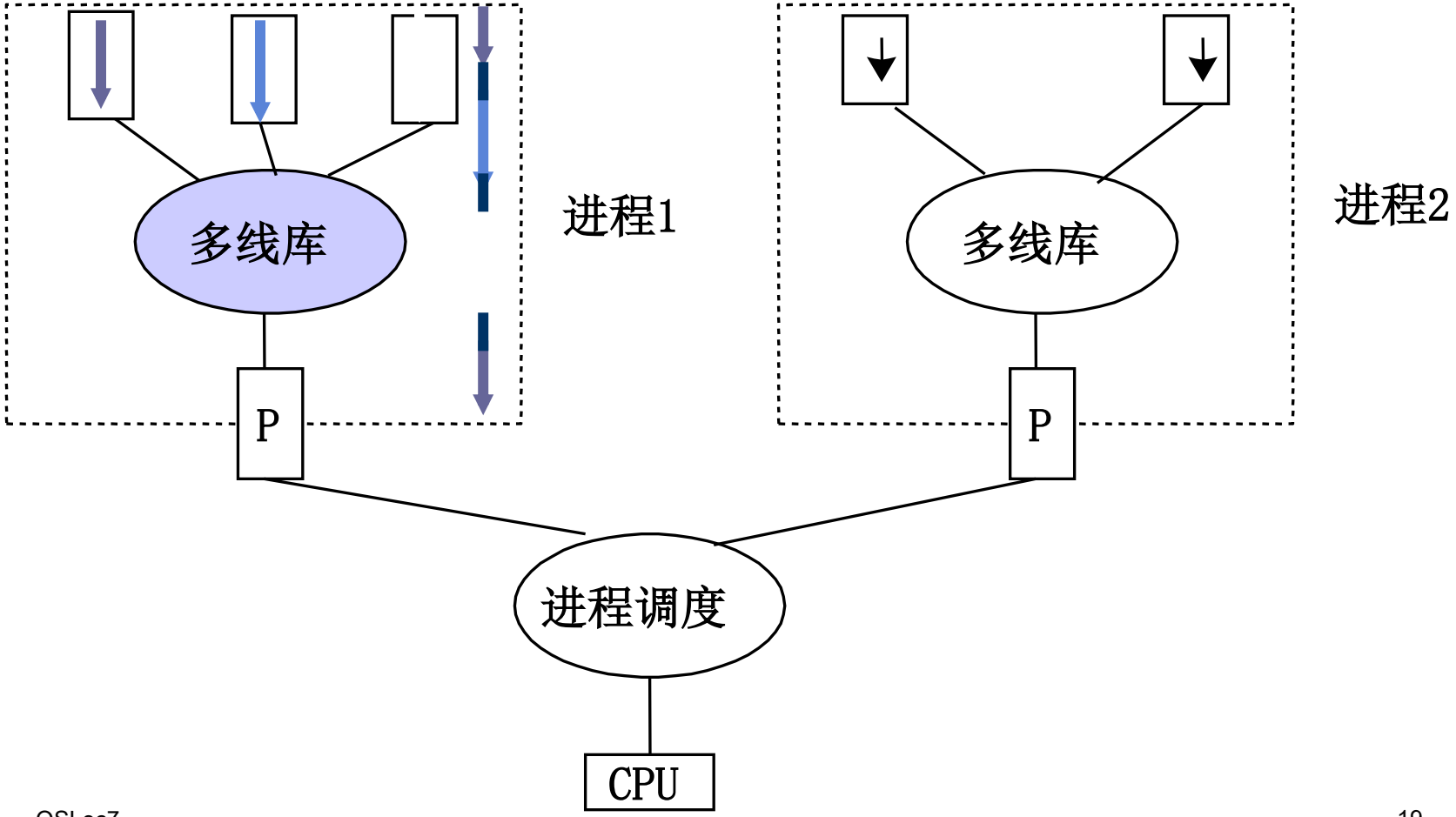
- 优点：
  - 对多处理器，核心可以同时调度同一进程的多个线程
  - 阻塞是在线程一级完成
  - 核心例程是多线程的
- 缺点：
  - 在同一进程内的线程切换调用内核，导致速度下降



用户线程ULT

- 用户级线程仅存在于用户空间中，线程的创建、撤消、线程之间的同步与通信等功能，都无须内核来实现。
  - 用户线程的维护由应用进程完成（通过线程库，用户空间，一组管理线程的过程）；
  - 内核不了解用户线程的存在；
  - 用户线程切换不需要内核特权；
  - 用户线程调度算法可针对应用优化；
- 线程库
  - 创建、撤消线程；在线程之间传递消息和数据；调度线程执行；保护和恢复线程上下文

用户级线程

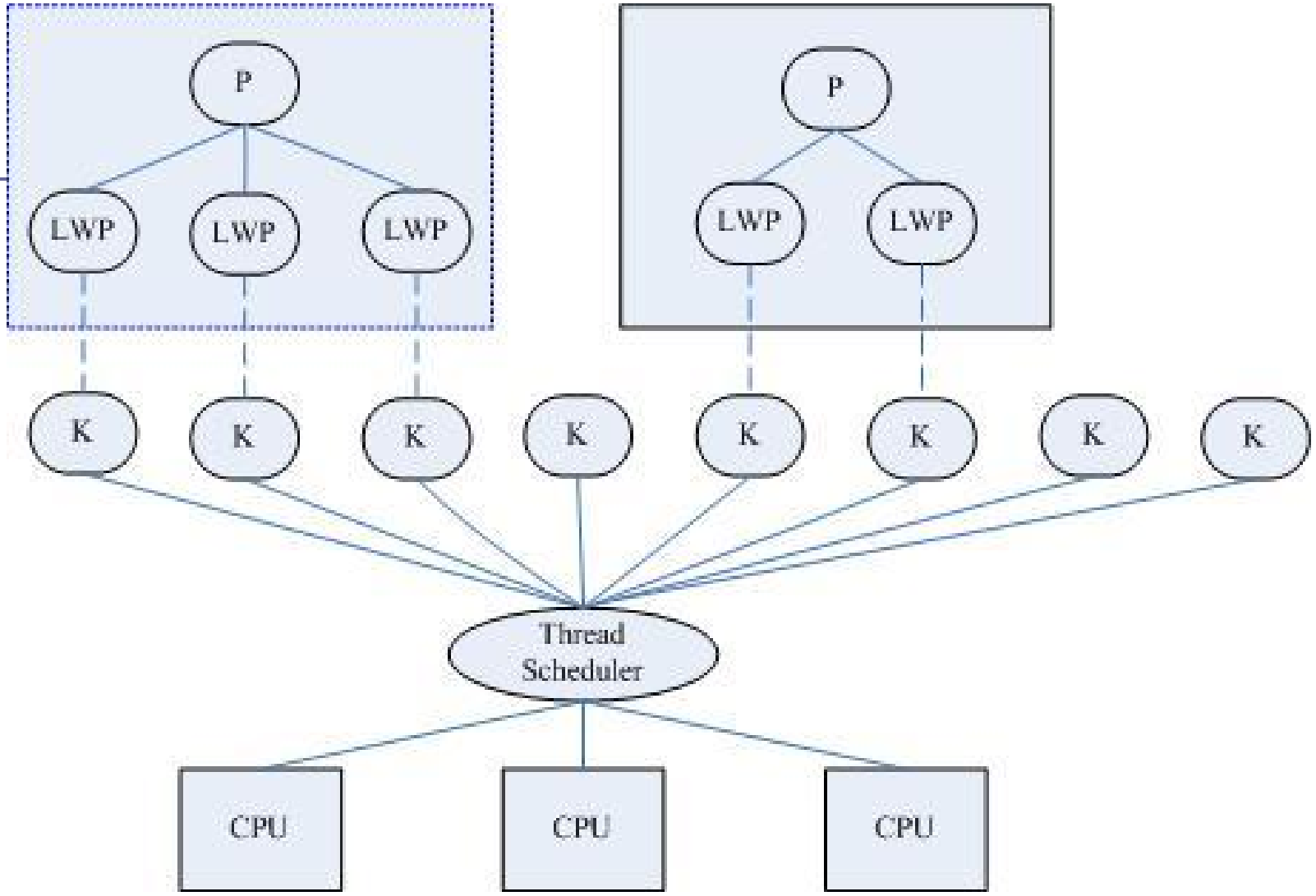


用户线程优点和缺点

- 优点：
  - 线程切换不调用核心
  - 调度是应用程序特定的：可以选择适合的算法
  - ULT可运行在任何操作系统上（只需线程库）
- 缺点：
  - 大多数系统调用是阻塞的，因此核心阻塞进程，故进程中所有线程将被阻塞
  - 核心只将处理器分配给进程，同一进程中的两个线程不能同时运行于两个处理器上

用户级线程的实现

- 运行时系统(Runtime System)
  - 用于管理和控制线程的函数(过程)的集合， 其中包括用于创建和撤消线程的函数、 线程同步和通信的函数以及实现线程调度的函数等。
- 内核控制线程， 又称为轻量级进程LWP(Light Weight Process)
  - 每一个进程都可拥有多个LWP，可通过系统调用来获得内核提供的服务，这样，当一个用户级线程运行时，只要将它连接到一个LWP上，此时它便具有了内核支持线程的所有属性。



利用轻型进程作为中间系统

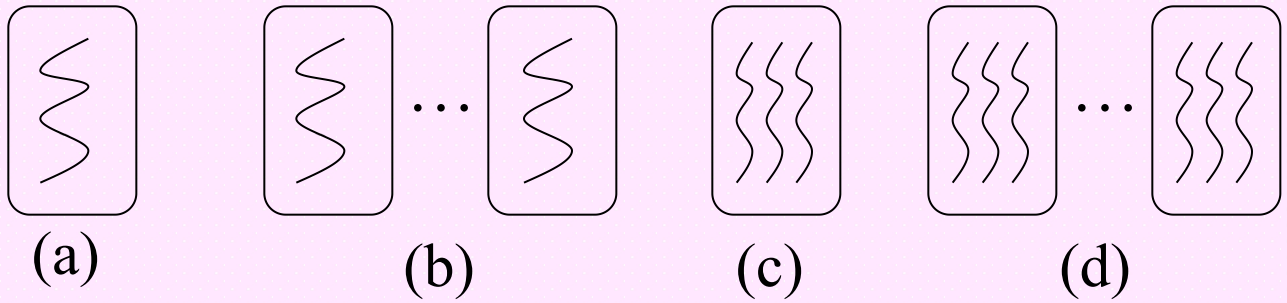
线程的状态

- 状态参数： 在OS中的每一个线程都可以利用线程标识符和一组状态参数进行描述。
  - ① 寄存器状态② 堆栈③ 线程运行状态④ 优先级⑤ 线程专有存储器⑥ 信号屏蔽
- 线程运行状态：各线程之间也存在着共享资源和相互合作的制约关系，致使线程在运行时也具有间断性。
  - ① 执行状态② 就绪状态③ 阻塞状态

线程的创建和终止

- 创建：
  - 应用程序在启动时，通常仅有一个线程在执行——初始化线程
  - 初始化线程可根据需要再去创建若干个线程
  - 线程创建函数执行完后，将返回一个线程标识符供以后使用
- 终止
  - 线程完成了自己的工作后自愿退出
  - 线程在运行中出现错误或由于某种原因而被其它线程强行终止

基于线程的观点OS分为四类



- 单进程和单线程系统
- 多进程和单线程系统
- 单进程和多线程系统
- 多进程和多线程系统

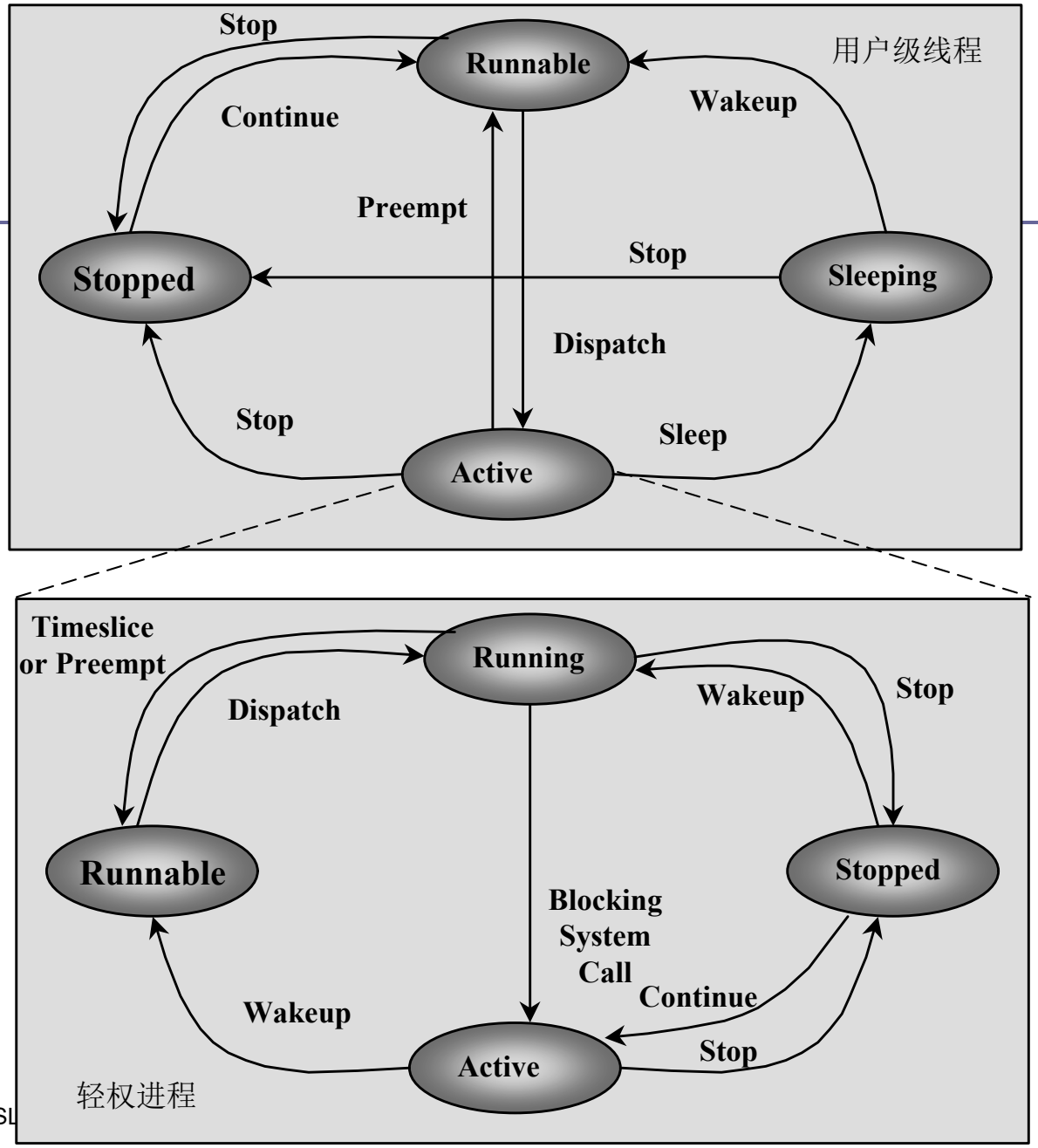
线程操作在SPARC上执行时间的比较(ms)

操　　作	用户级线程	核心级线程	进　程
创建	52	350	1700
使用信号量同步	66	390	200

3.5.4 线程举例

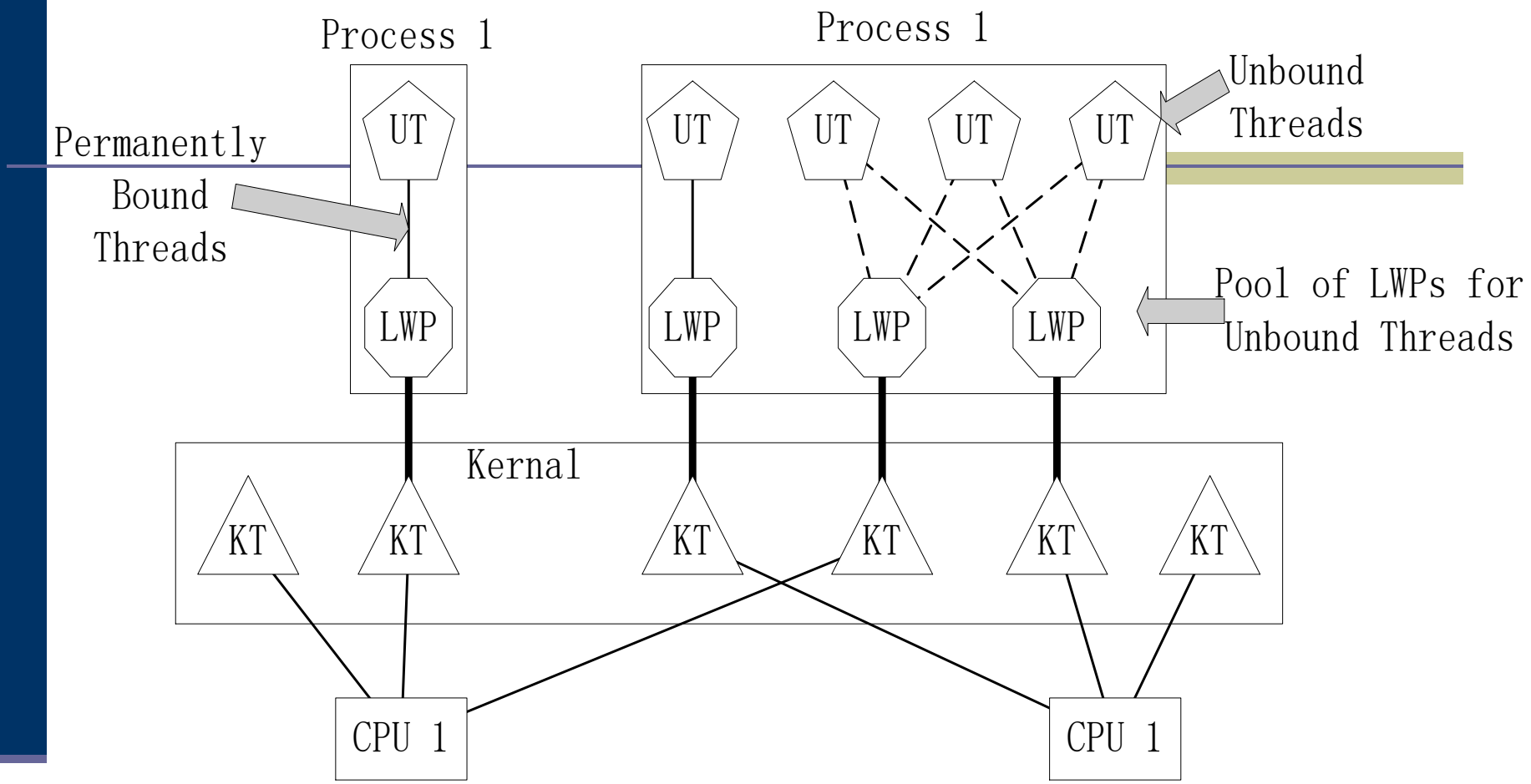
- SUN Solaris
  - Solaris支持内核线程(Kernel threads)、轻权进程(Lightweight Processes)和用户线程(User Level Threads)。一个进程可有大量用户线程；大量用户线程复用少量的轻权进程，不同的轻权进程分别对应不同的内核线程。

Solaris用户线程和轻权进程



SUN Solaris

- 用户级线程在使用系统调用时（如文件读写），需要“捆绑(bound)”在一个LWP上。
  - 永久捆绑：一个LWP固定被一个用户级线程占用，该LWP移到LWP池之外
  - 临时捆绑：从LWP池中临时分配一个未被占用的LWP
- 在使用系统调用时，如果所有LWP已被其他用户级线程所占用（捆绑），则该线程阻塞直到有可用的LWP——例如6个用户级线程，而LWP池中有4个LWP
- 如果LWP执行系统调用时阻塞（如read()调用），则当前捆绑在LWP上的用户级线程也阻塞。



用户线程、轻权进程和核心线程的关系

有关的C库函数

```
/* 创建用户级线程                                          */
 int thr_create(void *stack_base, size_t stack_size,
     void *(*start_routine)(void *), void *arg, long flags,
     thread_t *new_thread_id);
```

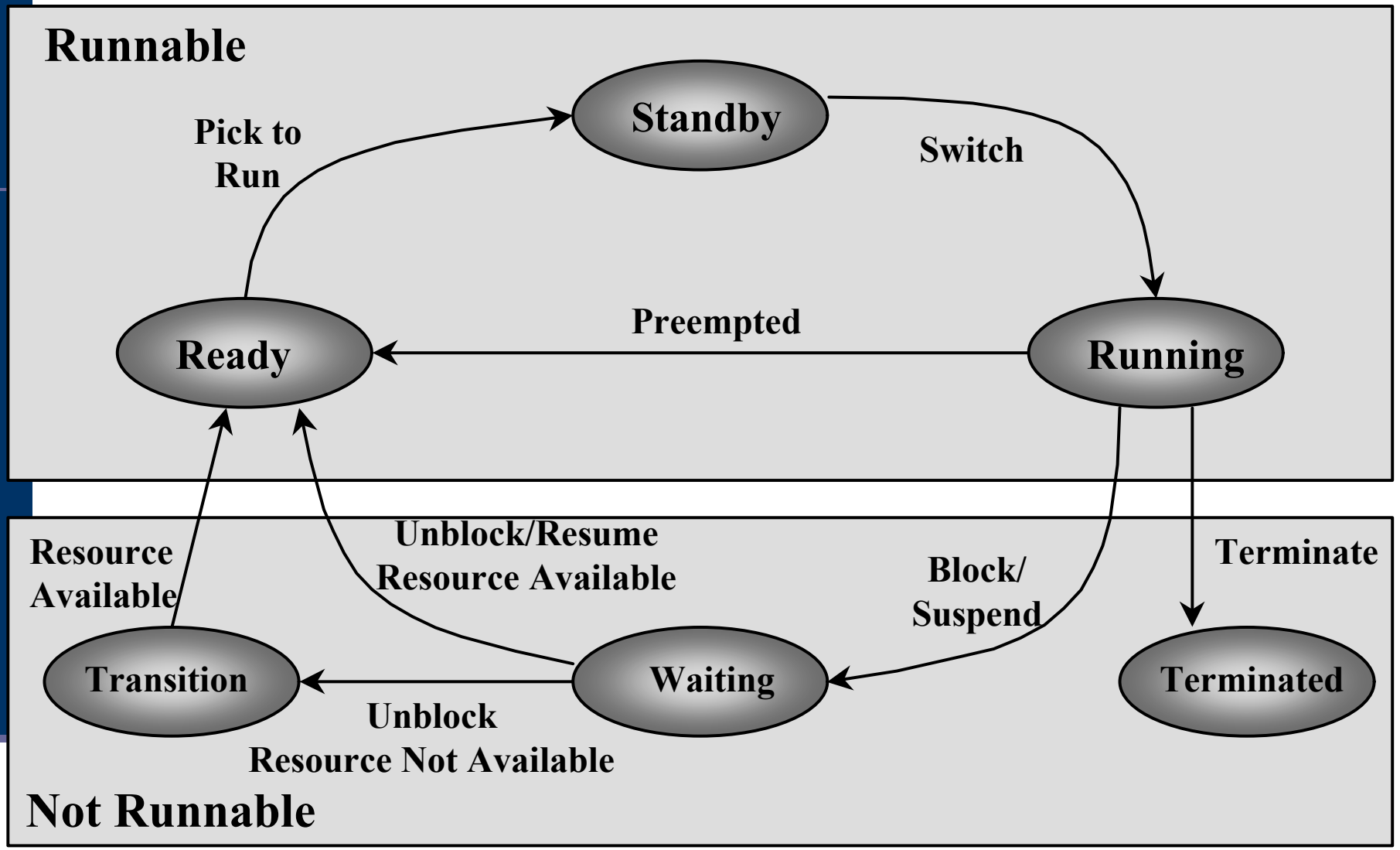
其中flags包括：THR_BOUND（永久捆绑）, THR_NEW_LWP（创建新LWP放入LWP池），若两者同时指定则创建两个新LWP，一个永久捆绑而另一个放入LWP池

有关的系统调用

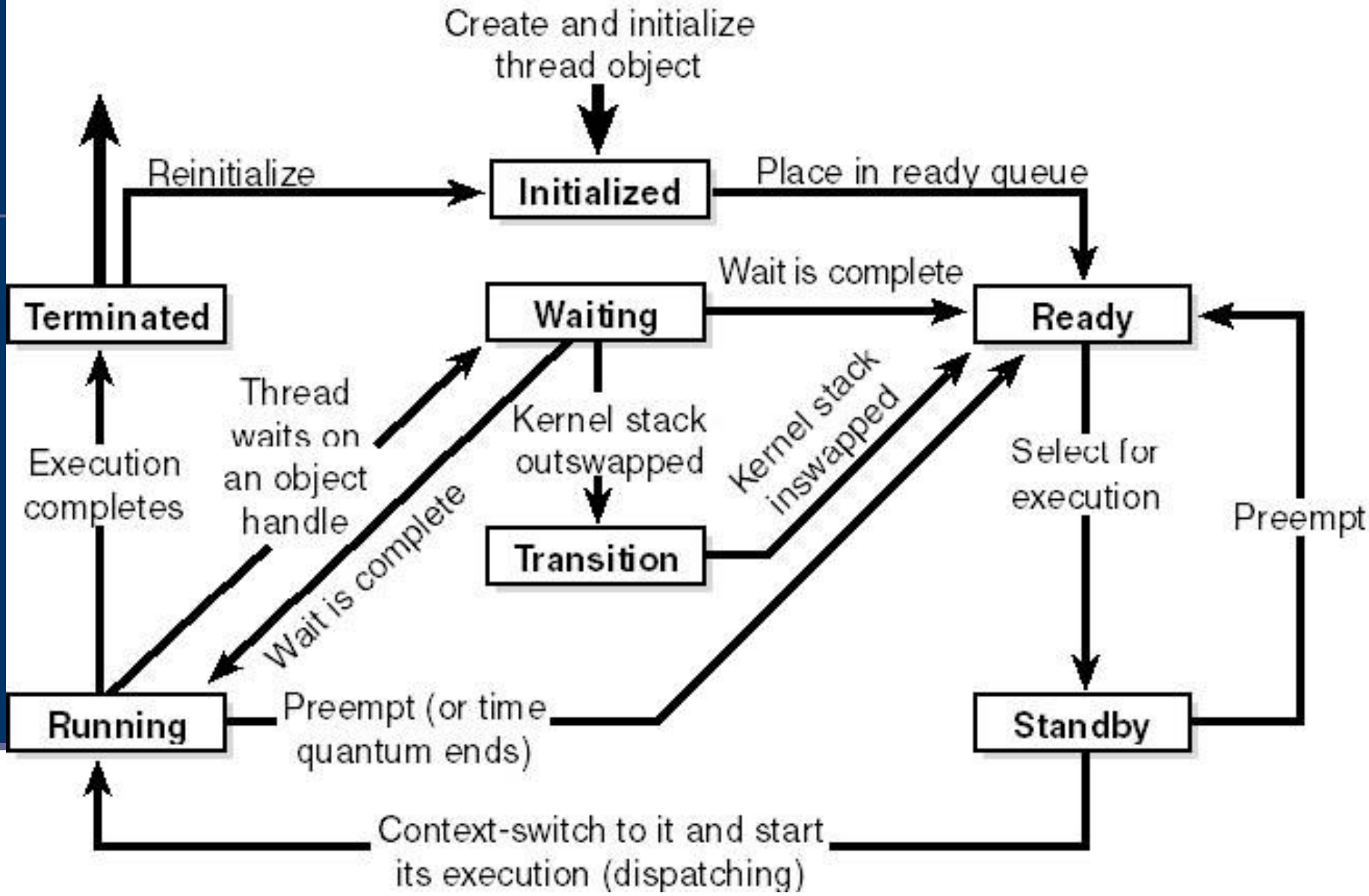
```
 /* 在当前进程中创建LWP                                    */
  int _lwp_create(ucontext_t *contextp, unsigned long flags,
      lwpid_t *new_lwp_id);
  /* 构造LWP上下文                                         */
  void _lwp_makecontext(ucontext_t *ucp,
          void (*start_routine)( void *), void *arg,
          void *private, caddr_t stack_base, size_t stack_size);
   /* 注意：没有进行"捆绑"操作的系统调用 */
```

Windows NT

- 就绪状态(Ready)：进程已获得除处理机外的所需资源，等待执行。
- 备用状态(Standby)：特定处理器的执行对象，系统中每个处理器上只能有一个处于备用状态的线程。
- 运行状态(Running)：完成描述表切换，线程进入运行状态，直到内核抢先、时间片用完、线程终止或进行等待状态。
- 等待状态(Waiting)：线程等待对象句柄，以同步它的执行。等待结束时，根据优先级进入运行、就绪状态。
- 转换状态(Transition)：线程在准备执行而其内核堆栈处于外存时，线程进入转换状态；当其内核堆栈调回内存，线程进入就绪状态。
- 终止状态(Terminated)：线程执行完就进入终止状态；如执行体有一指向线程对象的指针，可将线程对象重新初始化，并再次使用。
- 初始化状态(Initialized)：线程创建过程中的线程状态；



Windows NT的线程状态



Windows 2000线程状态

NT线程的有关API

- CreateThread()函数在调用进程的地址空间上创建一个线程，以执行指定的函数；返回值为所创建线程的句柄。
- ExitThread()函数用于结束本线程。
- SuspendThread()函数用于挂起指定的线程。
- ResumeThread()函数递减指定线程的挂起计数，挂起计数为0时，线程恢复执行。

作业、程序、进程和线程的比较

- 作业
  - 用户向计算机提交任务的任务实体
- 程序
  - 一组有序的指令集合
- 进程
  - 系统分配资源的基本单位
- 线程
  - 处理机调度的基本单位

What you need to do?

- 复习课本3.5节的内容
- 作业：在使用线程的系统中，若使用用户级线程，是每个线程一个堆栈还是每个进程一个堆栈？如果是使用内核级线程情况又如何？请给出解释。

See you next time!