南京大学
NANJING UNIVERSITY

计算机与操作系统

第五讲 多线程技术

南京大学软件学院

15:58

南京大学
NANJING UNIVERSITY

本主题教学目标

1. 掌握多线程环境下进程和线程的概念
2. 掌握线程的三种实现模型
3. 了解Windows的进程和线程管理
4. 掌握Solaris的进程和线程管理模型

南京大学
NANJING UNIVERSITY

第五讲 多线程技术

5.1 引入多线程的动机

5.2 多线程结构的进程

5.3 内核级线程的实现

5.4 用户级线程的实现

5.5 线程实现的混合策略

南京大学
NANJING UNIVERSITY

5.1 引入多线程的动机

单线程结构进程给并发程序设计效率带来问题

* 进程切换开销大
* 进程通信开销大
* 限制了进程并发的粒度
* 不适合并行计算的要求

线程的概念(1)

解决问题的基本思路：

* 把进程的两项功能－－“独立分配资源”与“被调度分派执行”分离开来，
* 进程作为系统资源分配和保护的独立单位，不需要频繁地切换和保护资源；
* 线程作为系统调度和分派的基本单位，能轻装运行，会被频繁地调度和切换，在这种指导思想下，产生了线程的概念。

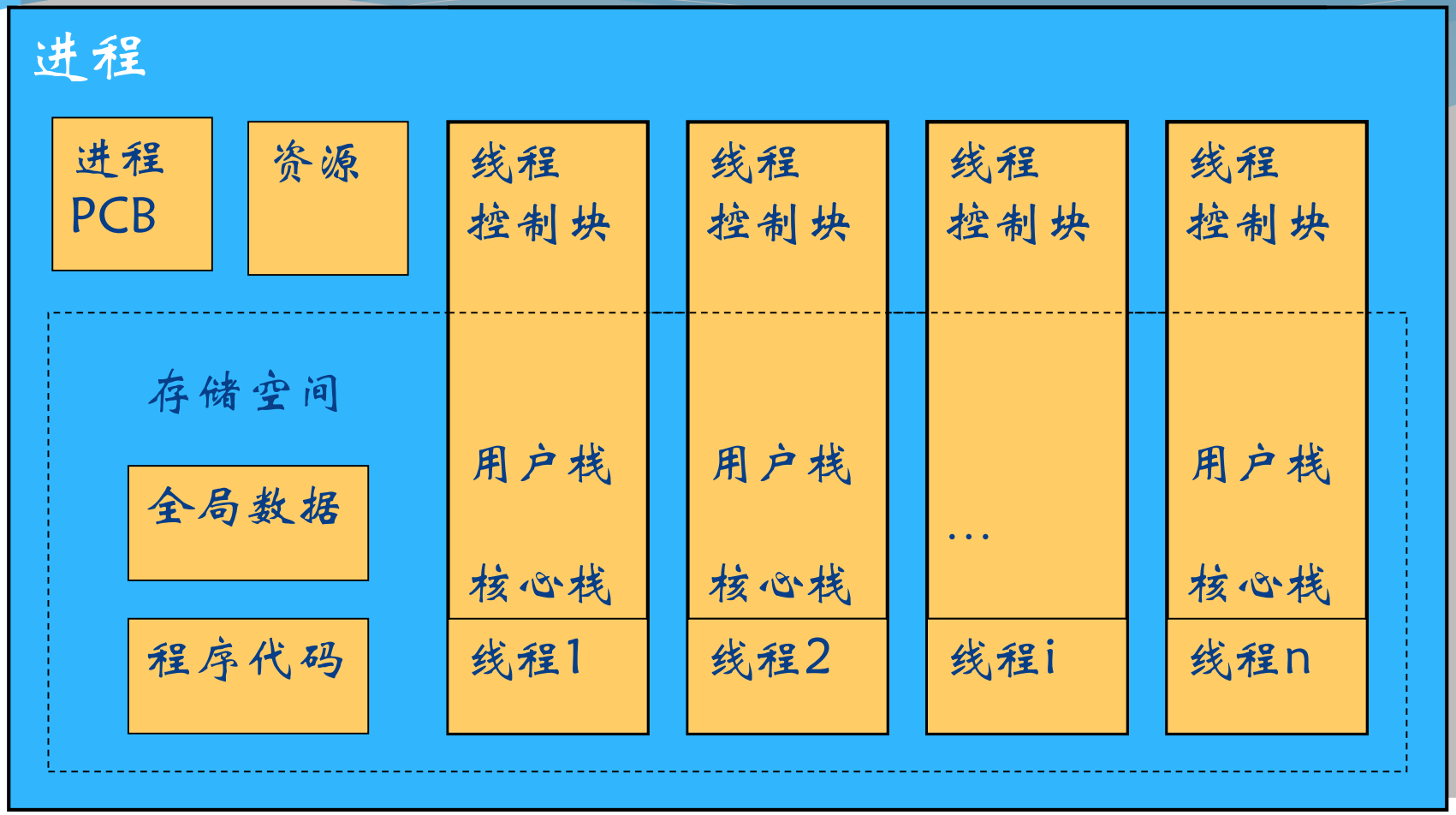
线程的概念(2)

* 操作系统中引入进程的目的是为了使多个程序并发执行，以改善资源使用率和提高系统效率，
* 操作系统中再引入线程，则是为了减少程序并发执行时所付出的时空开销，使得并发粒度更细、并发性更好。

南京大学
NANJING UNIVERSITY

5.2 多线程结构的进程

5.2 多线程结构的进程


进程
进程
PCB
资源
线程
控制块
线程
控制块
线程
控制块
线程
控制块
存储空间
全局数据
程序代码
用户栈
核心栈
用户栈
核心栈
...
用户栈
核心栈
线程1
线程2
线程i
线程n

南京大学
NANJING UNIVERSITY

多线程结构的进程

* 线程是进程的组成部分，每个进程内允许包含多个并发执行的实体(控制流)
* 线程作为处理器调度和分派的一级单位
* 线程的状态(运行态、就绪态、阻塞态)

南京大学
NANJING UNIVERSITY

线程组成

* 线程唯一标识符及线程状态信息
* 未运行时保存的线程上下文，可把线程看成是进程中一个独立的程序计数器在操作
* 核心栈，核心态下工作时，保存参数
* 用于存放线程局部变量及用户栈的私有存储区

南京大学
NANJING UNIVERSITY

并发多线程程序设计的优点

- 快速线程切换
- 减少(系统)管理开销
- (线程)通信易于实现
- 便于共享资源
- 并行程度提高

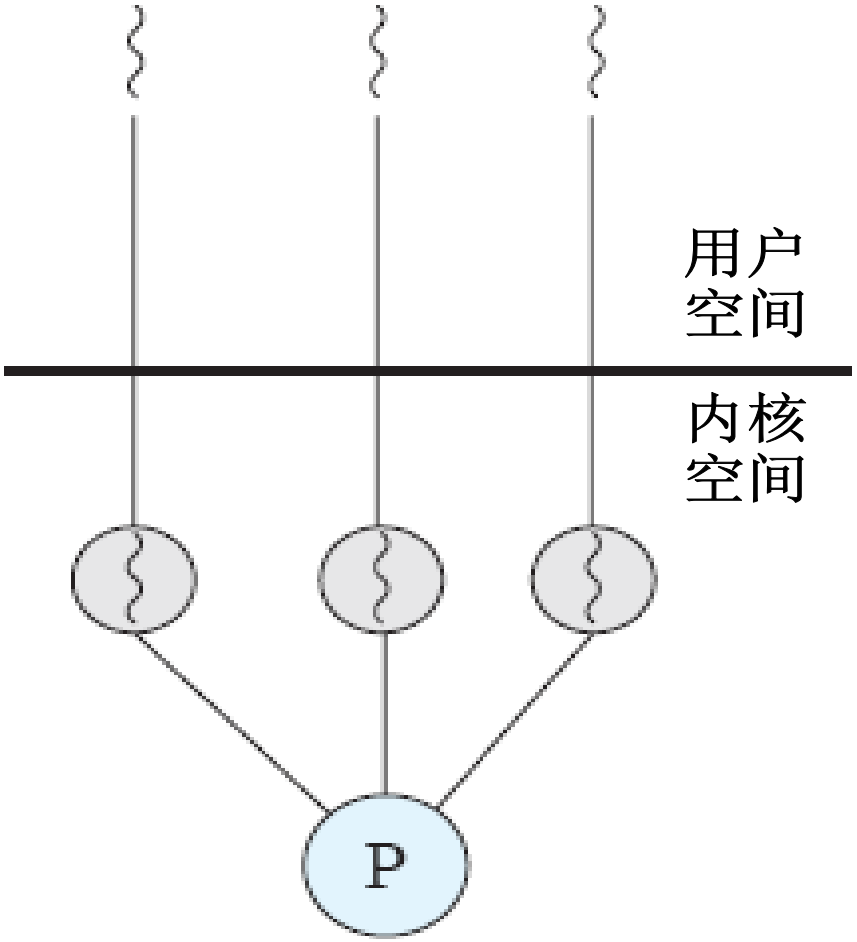
南京大学
NANJING UNIVERSITY

5.3 内核级线程的实现

5.3 内核级线程的实现
内核中运行的线程库

- 在内核中运行的线程库，是通过内核来管理线程库的
- 内核既要保存进程的数据结构，也要建立和维护线程的数据结构
- 由内核专门提供一组应用程序编程接口(API)，供开发者开发多线程应用程序
- Windows NT 和 OS/2都是采用这种方法的例子

内核级的线程
Kernel-Level Threads (KLT)



南京大学
NANJING UNIVERSITY

内核级的线程
Kernel-Level Threads (KLT)

优点

* 多处理器上，内核能同时调度同一进程中多个线程并行执行
* 进程中的一个线程被阻塞了，内核能调度同一进程的其它线程占有处理器运行
* 内核线程数据结构和堆栈很小，KLT切换快,内核自身也可用多线程技术实现，能提高系统的执行速度和效率

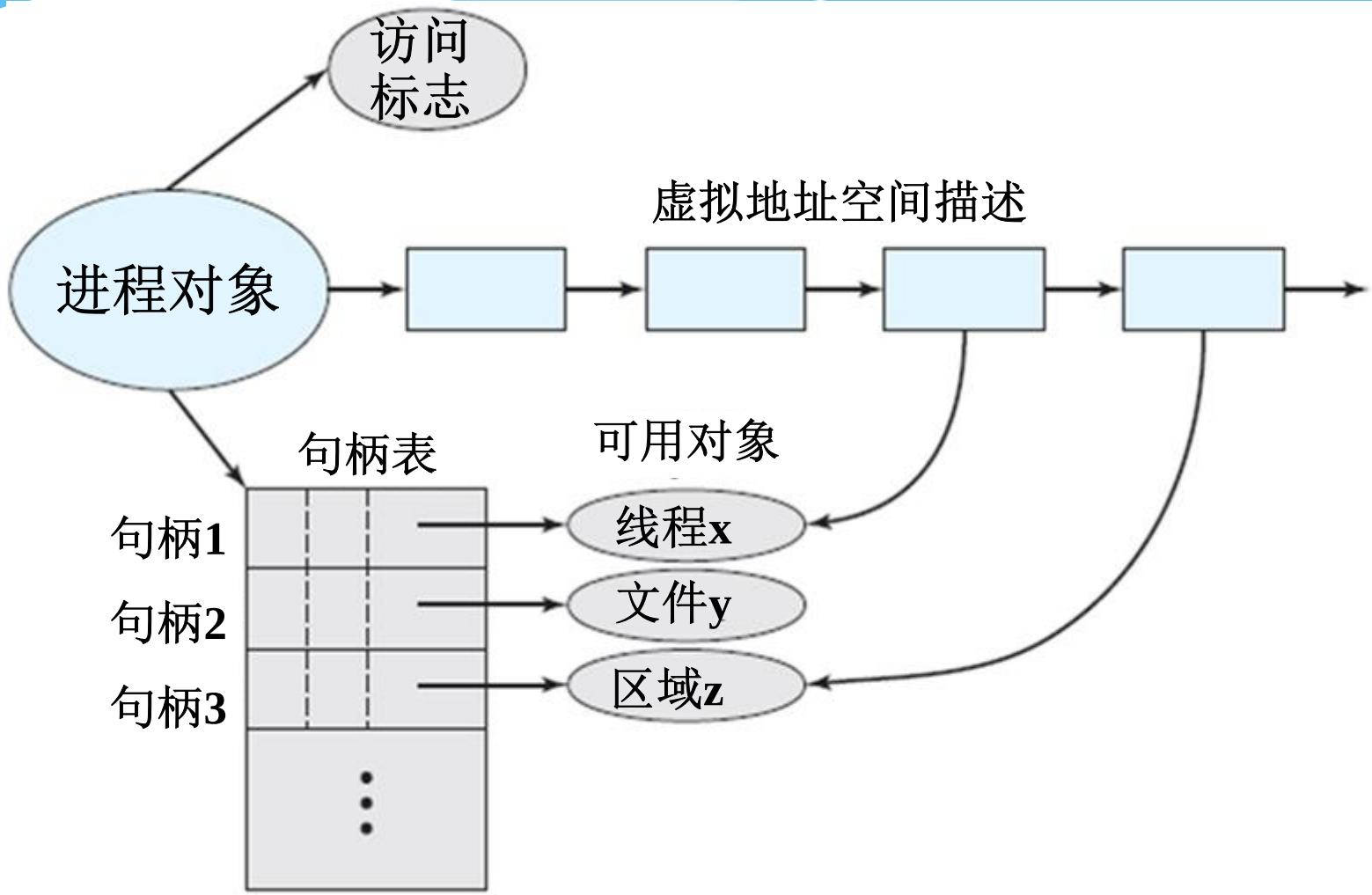
南京大学
NANJING UNIVERSITY

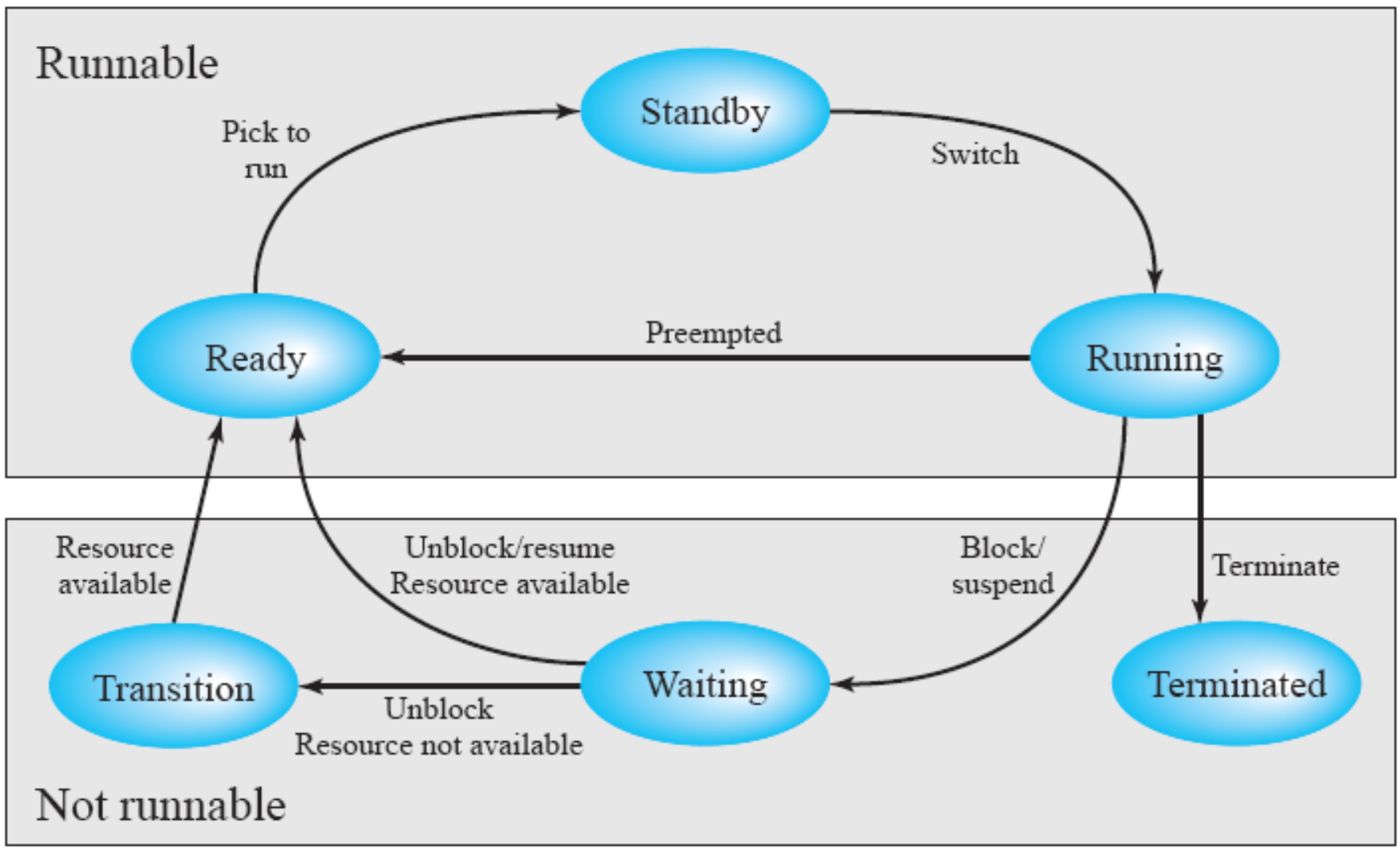
内核级的线程
Kernel-Level Threads (KLT)

缺点

* 应用程序线程在用户态运行，而线程调度和管理在内核实现,在同一进程中，控制权从一个线程传送到另一个线程时需要用户态-内核态-用户态的模式切换，系统开销较大

Windows进程与线程(例)




Runnable
Standby
Pick to run
Switch
Ready
Preempted
Running
Resource available
Unblock/resume Resource available
Block/ suspend
Terminate
Transition
Waiting
Terminated
Unblock
Resource not available
Not runnable

南京大学
NANJING UNIVERSITY

5.4 用户级线程的实现

5.4 用户级线程的实现
用户级空间的线程库

- 在用户空间运行的线程库，由于它完全在用户空间中运行，操作系统内核对线程库不可见，而仅仅知道管理的是一般的单线程进程
- 线程库是多线程应用程序的开发和运行支撑环境

南京大学
NANJING UNIVERSITY

用户级的线程
User-Level Threads (ULT)

- 在一个纯粹的用户级线程中，有关线程管理的所有工作都由应用程序完成，内核没有意识到有线程的存在
- 任何应用程序均需通过线程库进行程序设计，再与线程库连接后运行来实现多线程
- 线程库是一个ULT管理的例行程序包，实质上线程库是线程的运行支撑环境

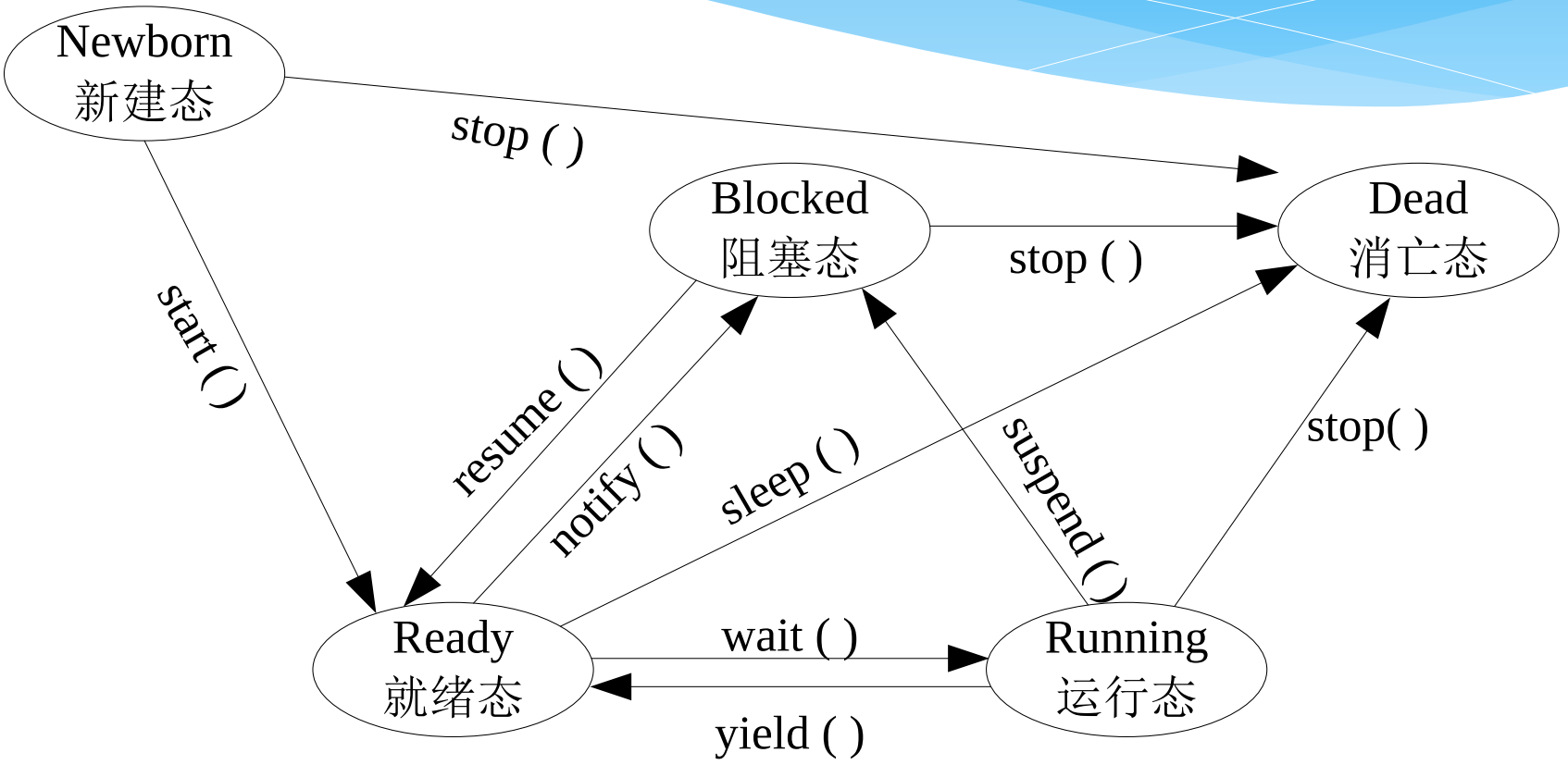
南京大学
NANJING UNIVERSITY

用户级的线程
User-Level Threads (ULT)

基本线程控制原语

* 孵化(Spaw)：新建线程
* 阻塞(Block)：阻塞线程
* 解除阻塞(Unblock)：当阻塞一个线程的事件发生时，该线程被转移到就绪队列中
* 结束(Finish)

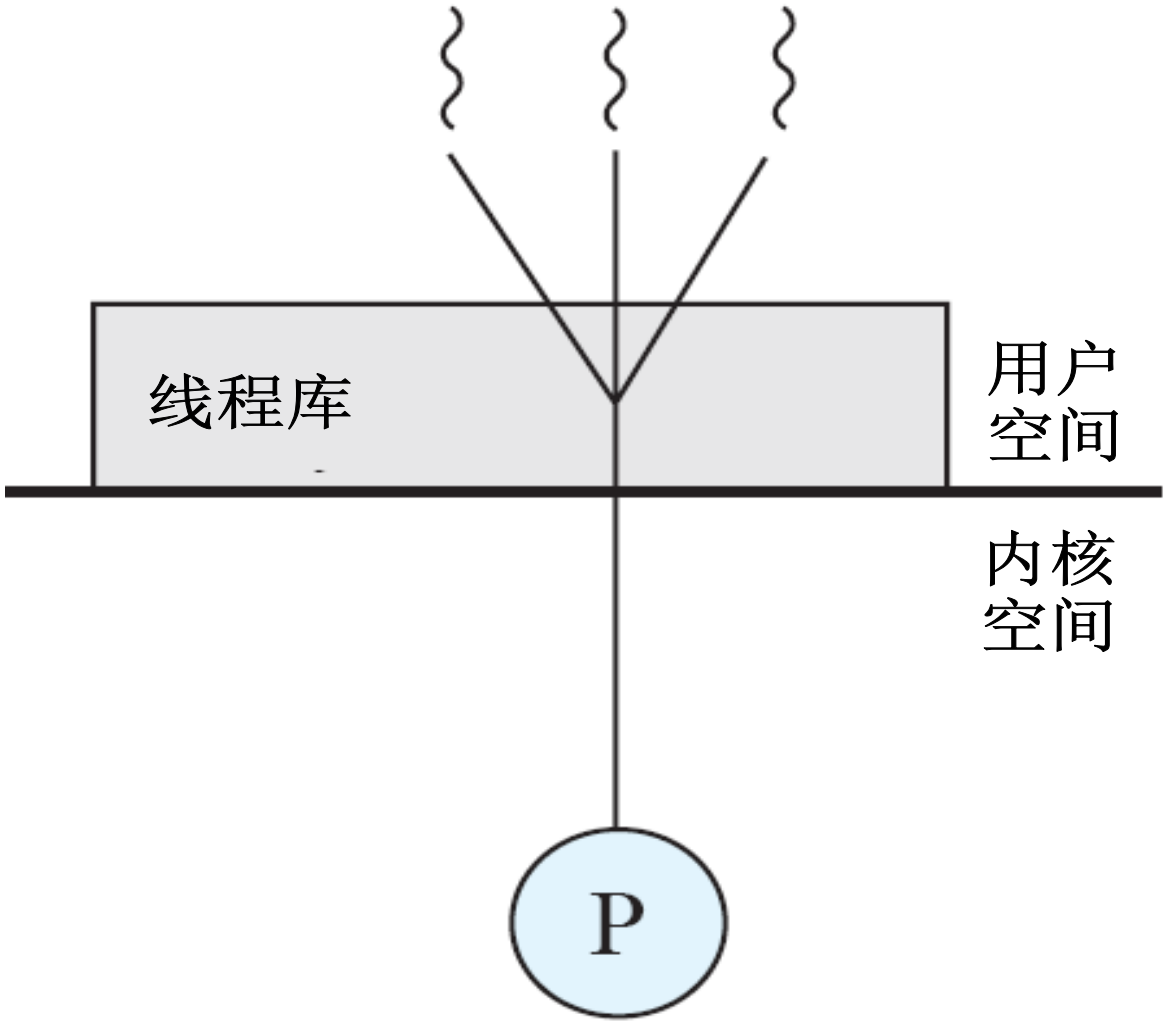
用户级的线程
User-Level Threads (ULT)



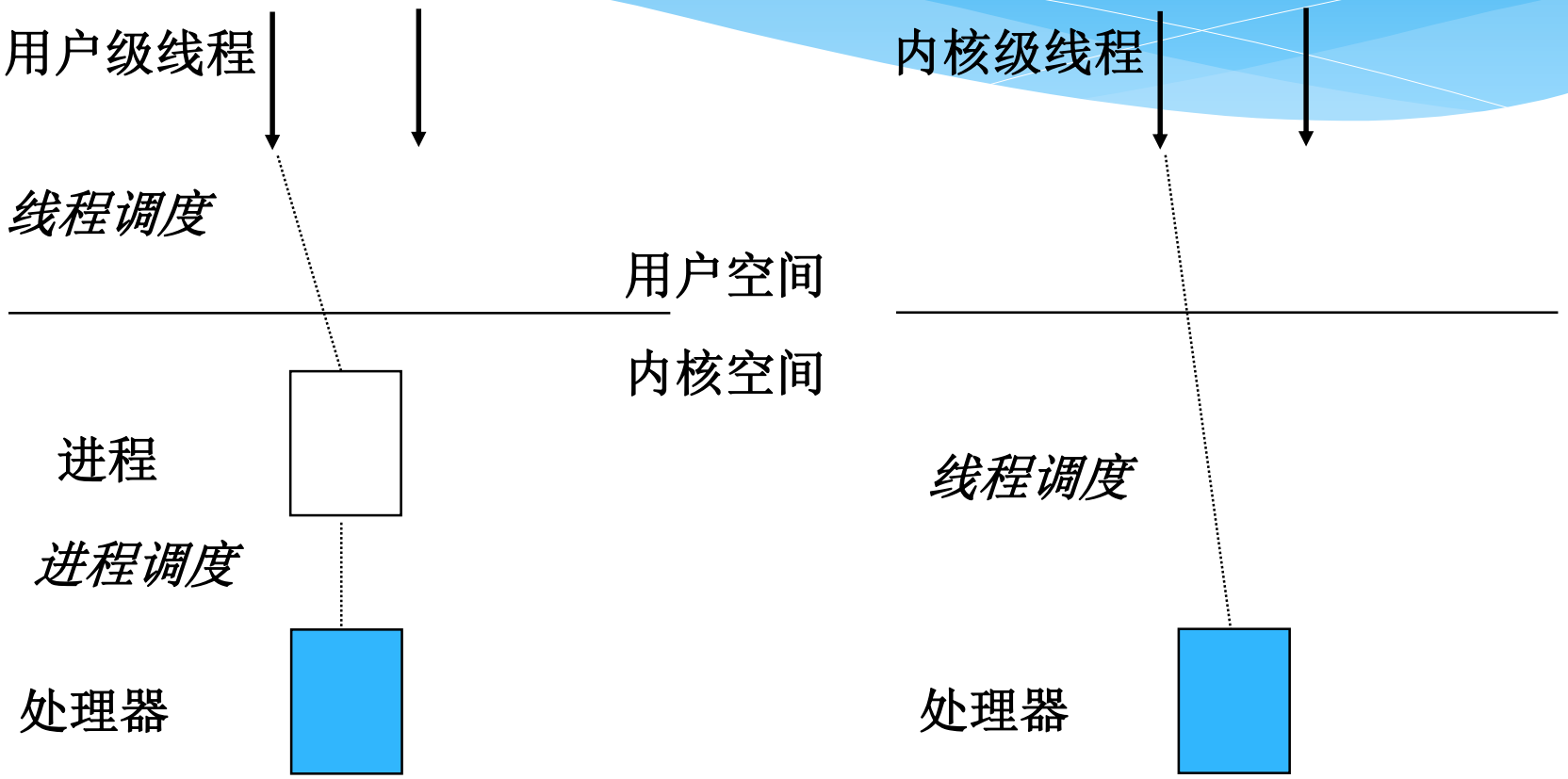
Java的线程状态

用户级的线程

User-Level Threads (ULT)



用户级线程 VS. 内核级线程



用户级的线程 User-Level Threads (ULT)

优点

- 线程切换不需要内核特权方式
  - 线程切换不需要内核特权方式，所有线程管理数据结构均在进程的用户空间中，管理线程切换的线程库也在用户地址空间运行，能节省模式切换的开销和内核的宝贵资源
- 按应用特定需要来调度
  - 允许进程按应用特定需要选择调度算法，且线程库的线程调度算法与操作系统的低级调度算法无关

南京大学
NANJING UNIVERSITY

用户级的线程
User-Level Threads (ULT)

优点

- 用户级线程(ULT)能运行在任何OS上
  - 内核在支持ULT方面不需要做任何改变，线程库是可以被所有应用共享的应用级实用程序，许多当代操作系统和语言均提供了线程库，传统UNIX并不支持多线程，但已有了多个基于UNIX的用户线程库

南京大学
NANJING UNIVERSITY

用户级的线程
User-Level Threads (ULT)

缺点

- 由于大多数系统调用是阻塞型的，因此，一个ULT的阻塞，将引起整个进程的阻塞
- ULT不能利用多处理器的优点，内核分配进程到CPU上，仅有一个ULT能执行，因此，不可能得益于多线程的并发执行

内核没有意识到有线程的存在

南京大学 NANJING UNIVERSITY

用户级的线程 User-Level Threads (ULT)

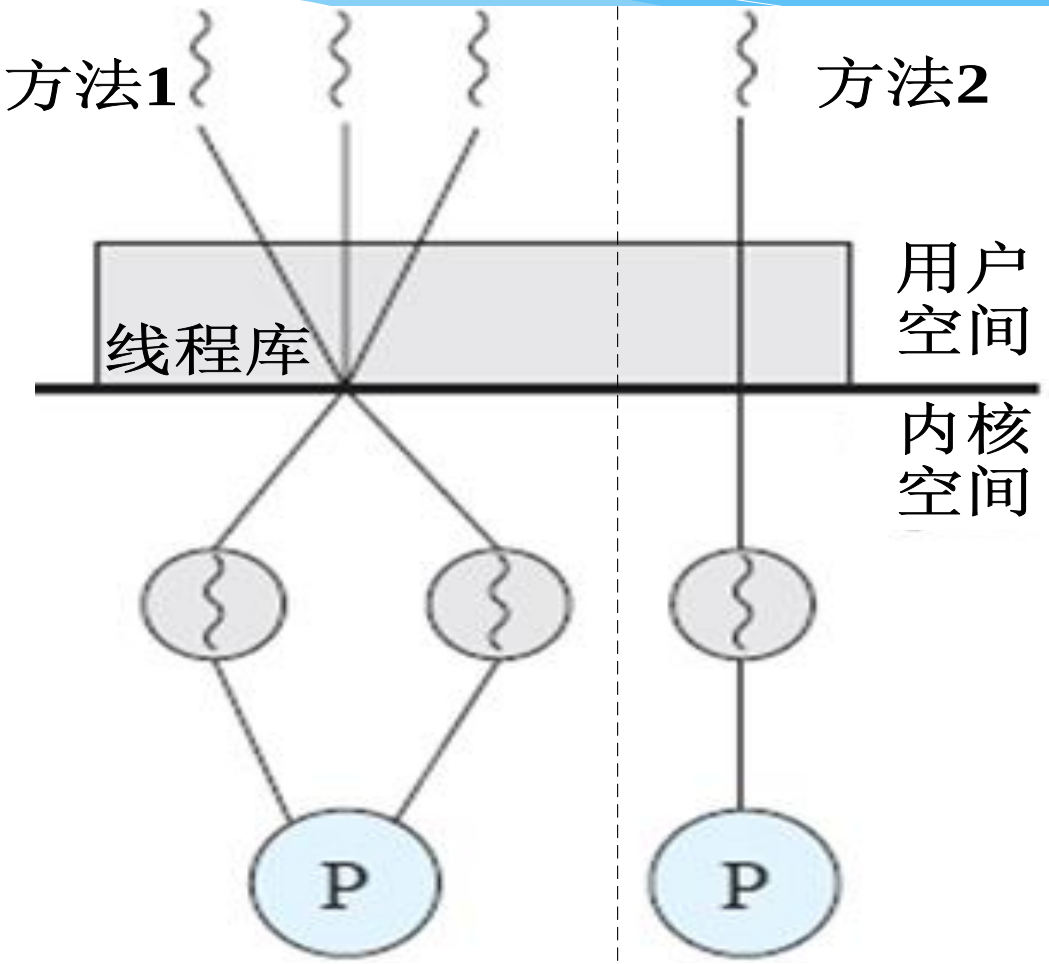
Jacketing技术

* 在纯用户级线程中，多线程应用不能利用多处理器的优点，可以采用jacketing 技术来解决阻塞线程的问题，其主要思想是
  * 把阻塞式的系统调用改造成非阻塞式的，当线程执行系统调用时，首先执行jacketing 实用程序，由jacketing 程序来检查资源使用情况，以决定是否执行系统调用或传递控制权给另一个线程

南京大学
NANJING UNIVERSITY

5.5 线程实现的混合策略

线程实现的混合策略

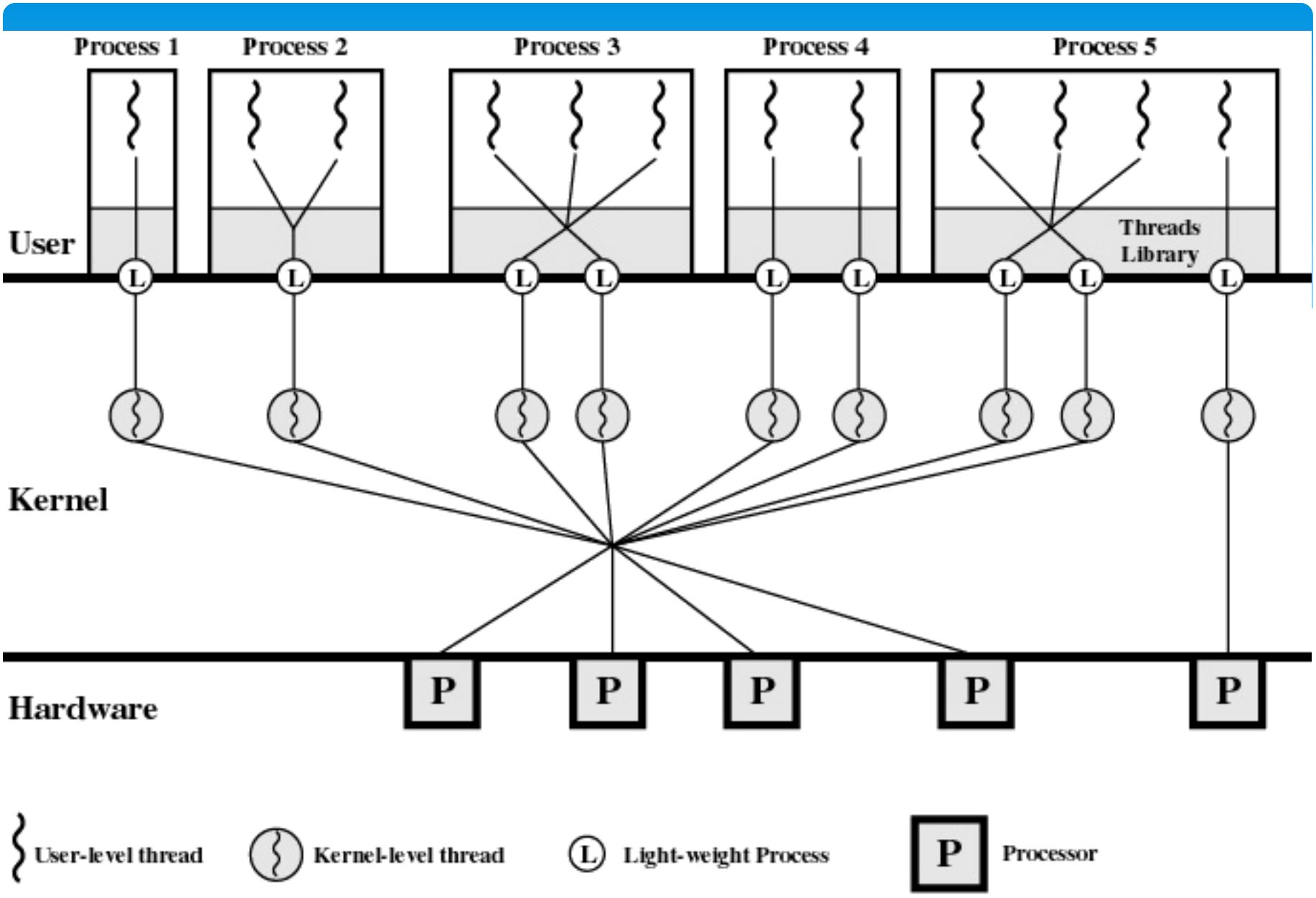


线程实现的混合策略

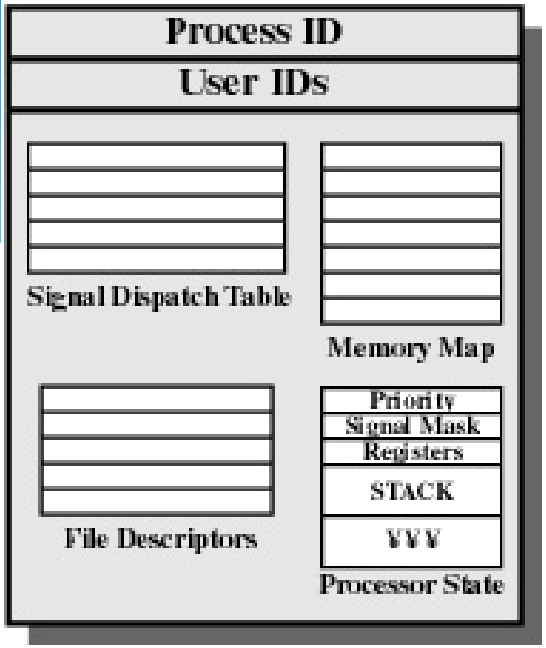
- 组合用户级线程/内核级线程设施。例如：Solaris
- 线程创建完全在用户空间中完成，线程的调度和同步也在应用程序中进行
- 一个应用程序中的多个用户级线程被映射到一些(小于或等于用户级线程的数目)内核级线程上
- 程序员可以为特定的应用程序和机器调节内核级线程的数目，以达到整体最佳结果
- 该方法将会结合纯粹用户级线程方法和内核级线程方法的优点，同时减少它们的缺点

Solaris的进程与线程

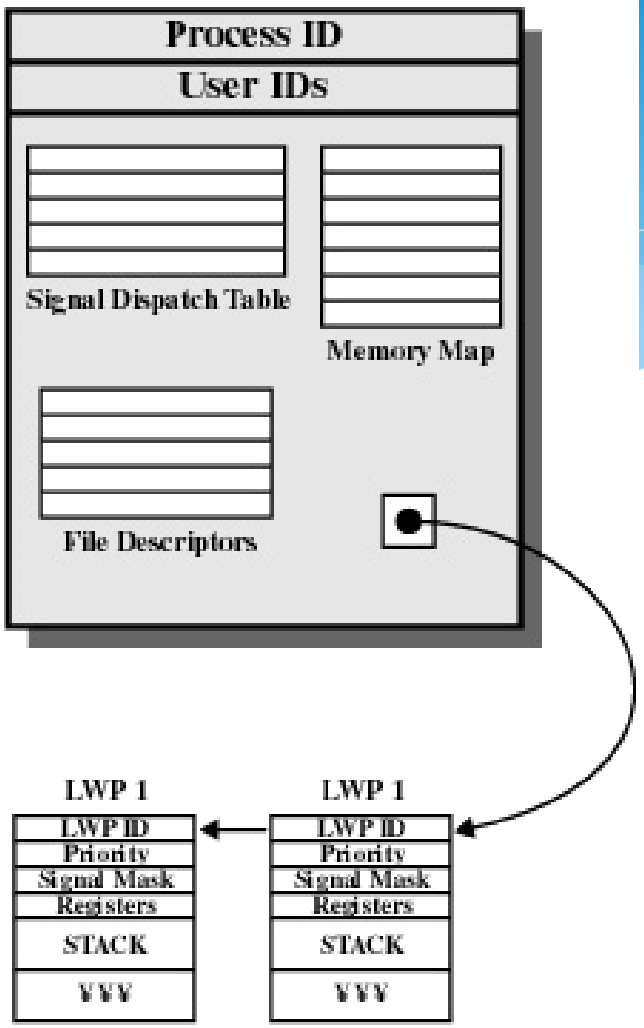
* 内核级线程：这是可以调度和分派到系统处理器上运行的基本实体
* 用户级线程：通过进程地址空间中的线程库实现，它们对操作系统是不可见的
* 轻量级进程：轻量级进程可以看做是用户级线程和内核级线程的映射，每个轻量级进程支持一个或多个用户级线程，并映射到一个内核级线程

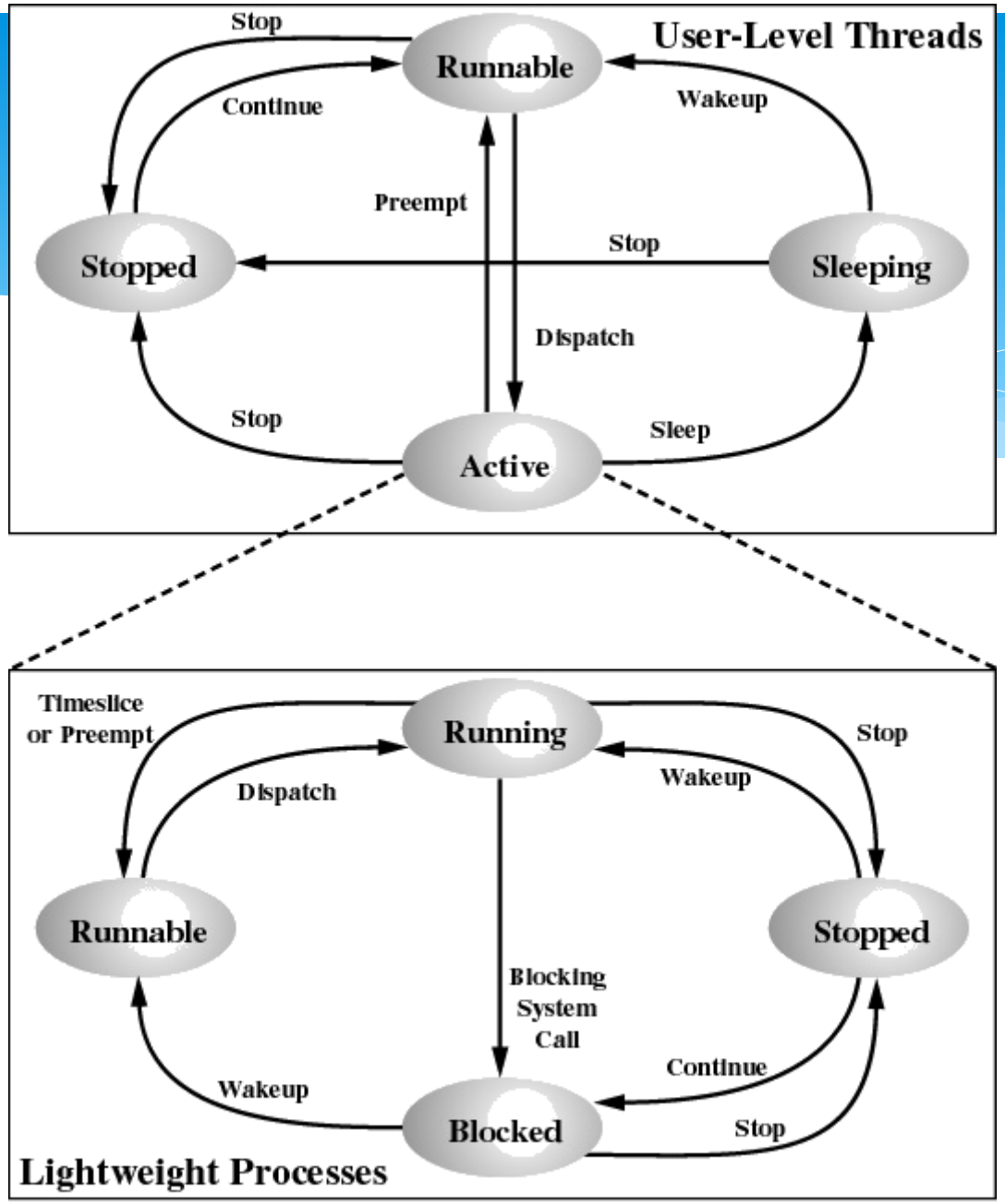

Process 1
Process 2
Process 3
Process 4
Process 5
User
Threads Library
L
Kernel
Hardware
P
User-level thread
Kernel-level thread
Light-weight Process
Processor

UNIX Process Structure



Solaris Process Structure




User-Level Threads
Runnable
Stop
Continue
Wakeup
Preempt
Stopped
Stop
Sleeping
Dispatch
Stop
Sleep
Active
Lightweight Processes
Timeslice
or Preempt
Running
Stop
Dispatch
Wakeup
Runnable
Stopped
Blocking
System
Call
Continue
Wakeup
Blocked
Stop

南京大学
NANJING UNIVERSITY

本主题小结

1. 掌握多线程环境下进程和线程的概念
2. 掌握线程的三种实现模型
3. 了解Windows的进程和线程管理
4. 掌握Solaris的进程和线程管理模型