南京大学
NANJING UNIVERSITY

计算机操作系统

2 处理器管理 – 2.4 多线程技术

2.4.1 多线程环境概述

掌握单线程结构进程

掌握多线程结构进程

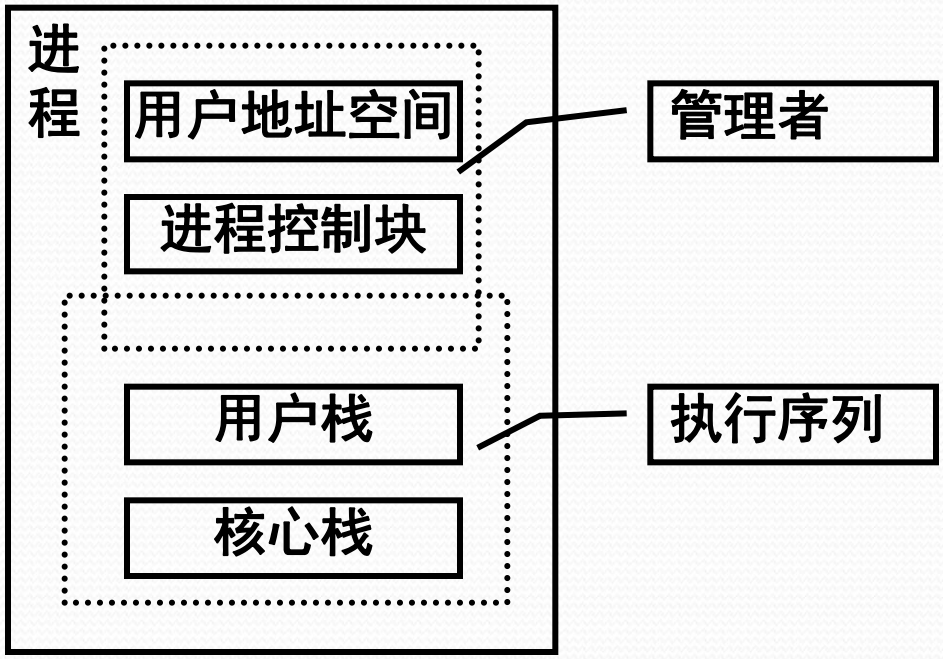
掌握多线程环境下进程与线程的概念

理解多线程环境下线程的状态与调度

理解多线程并发程序设计的优势与应用

单线程结构进程

- 传统进程是单线程结构进程



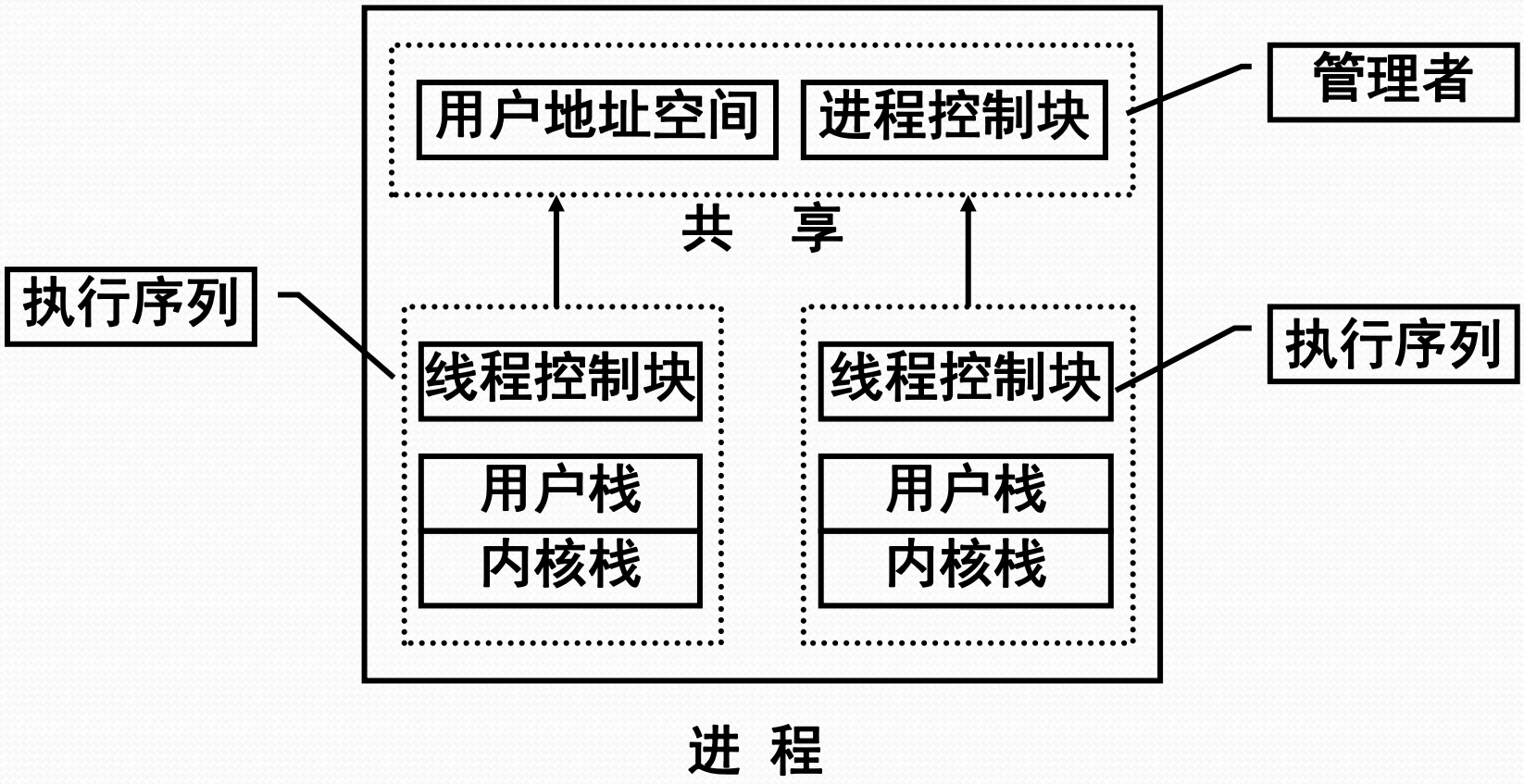
单线程结构进程的问题

- 单线程结构进程在并发程序设计上存在的问题
  - 进程切换开销大
  - 进程通信开销大
  - 限制了进程并发的粒度
  - 降低了并行计算的效率

解决问题的思路

- 把进程的两项功能，即“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；
- 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
- 线程的出现会减少进程并发执行所付出的时空开销，使得并发粒度更细、并发性更好

多线程结构进程



多线程环境下进程的概念

- 在多线程环境中，进程是操作系统中进行保护和资源分配的独立单位。具有：
  - 用来容纳进程映像的虚拟地址空间
  - 对进程、文件和设备的存取保护机制

多线程环境下线程的概念

- 线程是进程的一条执行路径，是调度的基本单位，同一个进程中的所有线程共享进程获得的主存空间和资源。它具有：
  - 线程执行状态
  - 受保护的线程上下文，当线程不运行时，用于存储现场信息
  - 独立的程序指令计数器
  - 执行堆栈
  - 容纳局部变量的静态存储器

多线程环境下线程的状态与调度

- 线程状态有运行、就绪和睡眠，无挂起
- 与线程状态变化有关的线程操作有：
  - 孵化、封锁、活化、剥夺、指派、结束
- OS感知线程环境下：
  - 处理器调度对象是线程
  - 进程没有三状态（或者说只有挂起状态）
- OS不感知线程环境下：
  - 处理器调度对象仍是进程
  - 用户空间中的用户调度程序调度线程

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
- 并行程度提高
- 节省内存空间

多线程技术的应用

- 前台和后台工作
- C/S应用模式
- 加快执行速度
- 设计用户接口

南京大学
NANJING UNIVERSITY

计算机操作系统

2 处理器管理 – 2.4 多线程技术

2.4.2 多线程的实现技术

掌握内核级多线程KLT

掌握用户级多线程ULT

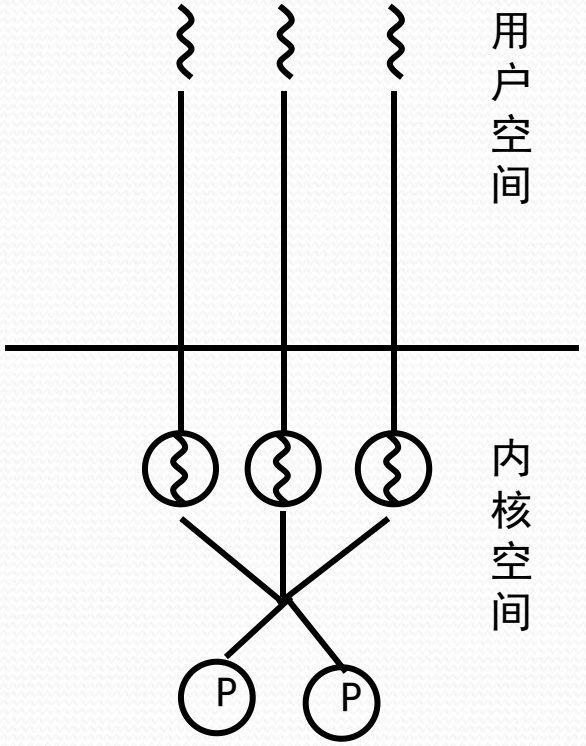
理解KLT与ULT的区别

掌握多线程实现的混合式策略

了解ULT与KLT的状态模型

内核级线程KLT, Kernel-Level Threads

- 线程管理的所有工作由OS内核来做
- OS提供了一个应用程序设计接口API，供开发者使用KLT
- OS直接调度KLT

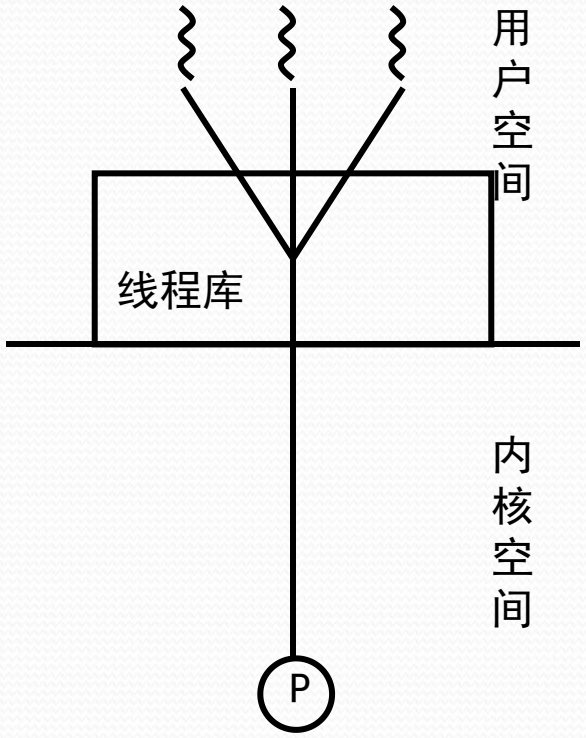


内核级线程的特点

- 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
- 多处理器环境中，内核能同时调度同一进程中多个线程并行执行
- 内核自身也可用多线程技术实现，能提高操作系统的执行速度和效率
- 应用程序线程在用户态运行，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模式切换，系统开销较大

用户级线程ULT, User-Level Threads

- 用户空间运行的线程库，提供多线程应用程序的开发和运行支撑环境
- 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
- 线程管理的所有工作都由应用程序完成，内核没有意识到线程的存在



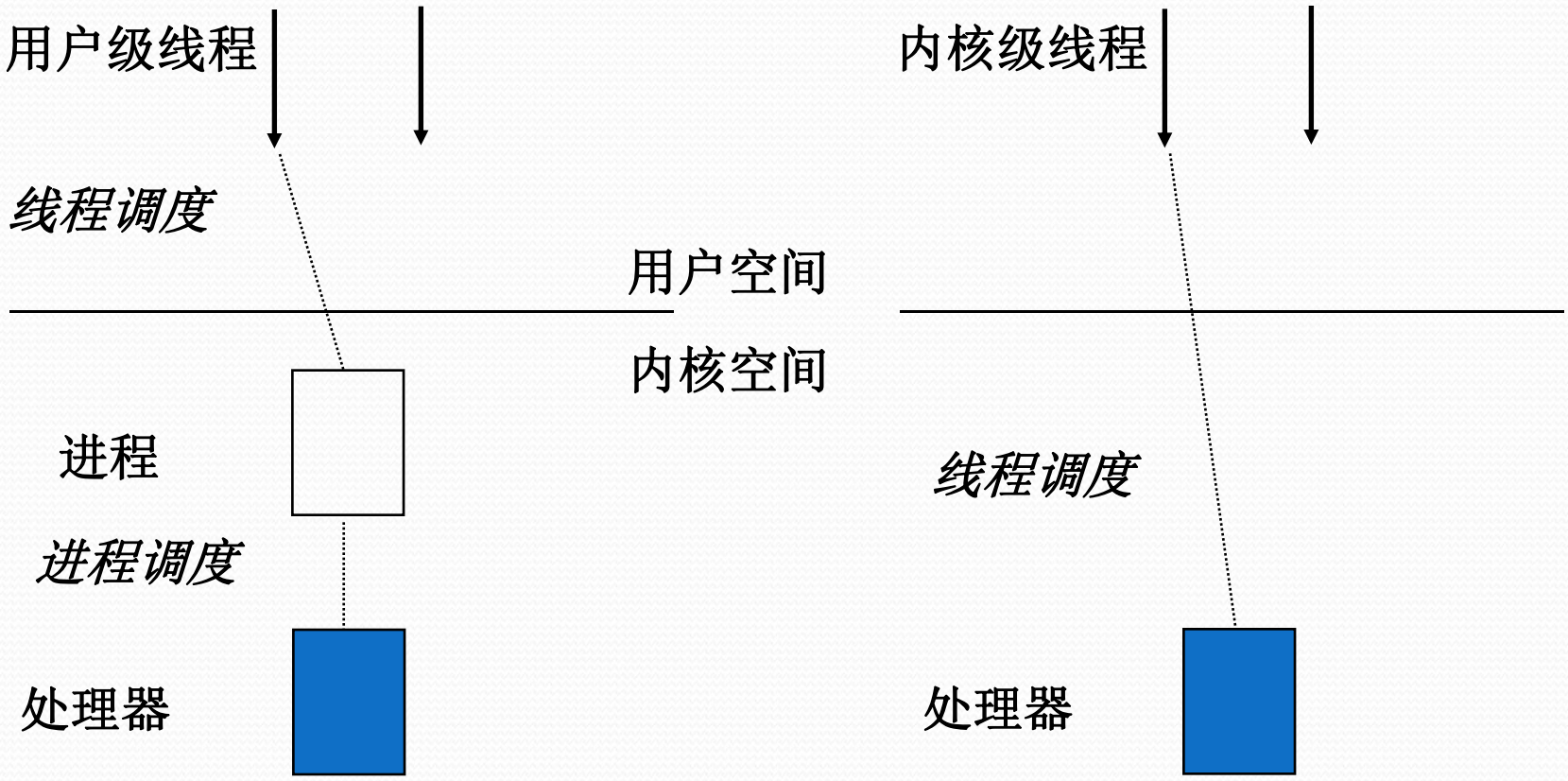
用户级线程的特点

- 所有线程管理数据结构均在进程的用户空间中，线程切换不需要内核模式，能节省模式切换开销和内核的宝贵资源
- 允许进程按应用特定需要选择调度算法，甚至根据应用需求裁剪调度算法
- 能运行在任何OS上，内核在支持ULT方面不需要做任何工作
- 不能利用多处理器的优点，OS调度进程，仅有一个ULT能执行
- 一个ULT的阻塞，将引起整个进程的阻塞

Jacketing技术

- 把阻塞式系统调用改造成非阻塞式的
- 当线程陷入系统调用时，执行jacketing程序
- 由jacketing 程序来检查资源使用情况，以决定是否执行进程切换或传递控制权给另一个线程

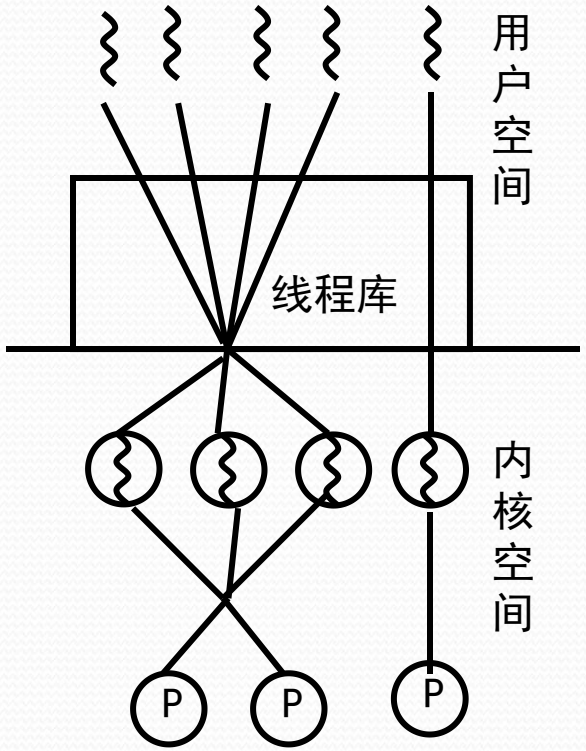
用户级线程 vs. 内核级线程



- ULT适用于解决逻辑并行性问题
- KLT适用于解决物理并行性问题

多线程实现的混合式策略

- 线程创建是完全在用户空间做的
- 单应用的多个**ULT**可以映射成一些**KLT**，通过调整**KLT**数目，可以达到较好的并行效果

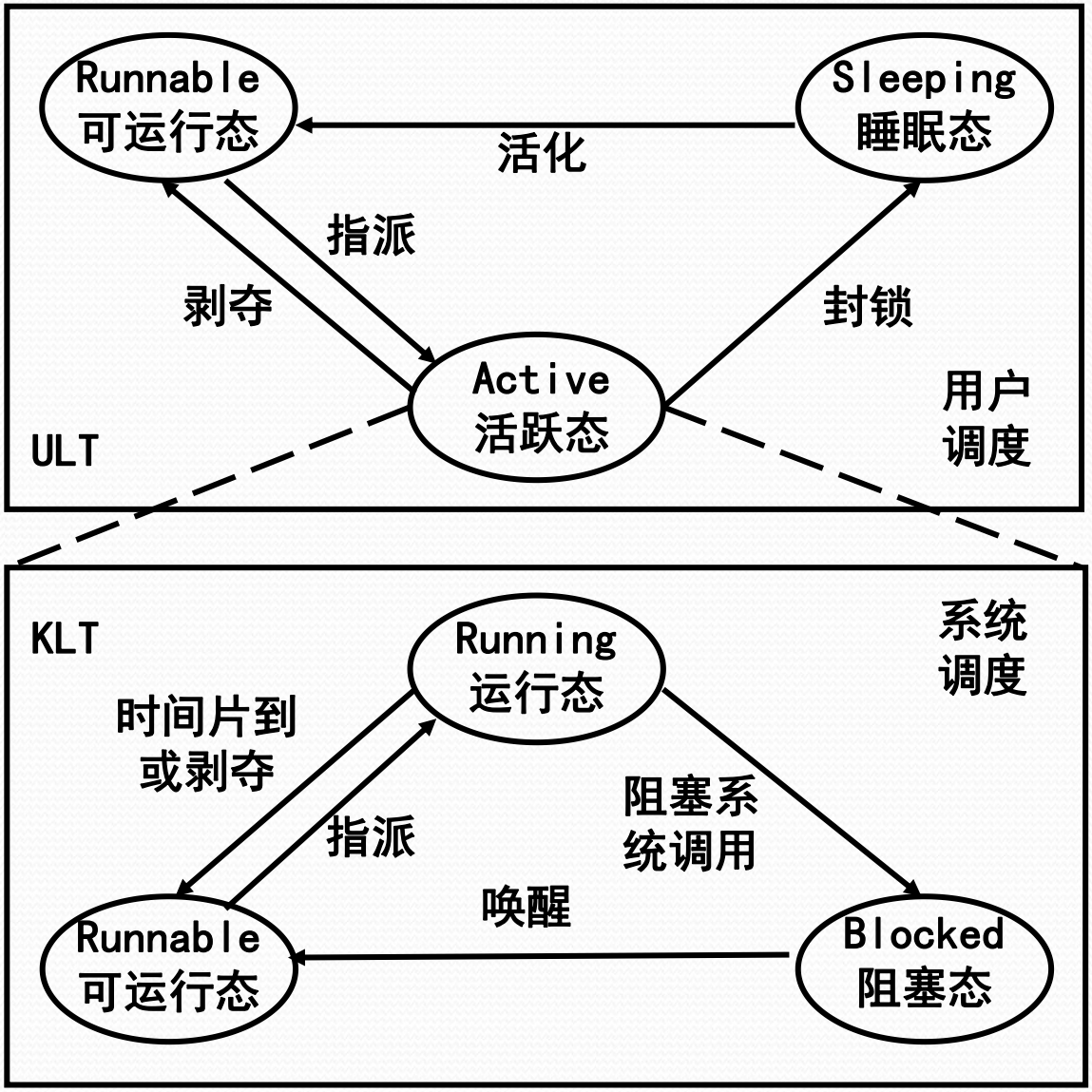


多线程实现混合式策略的特点

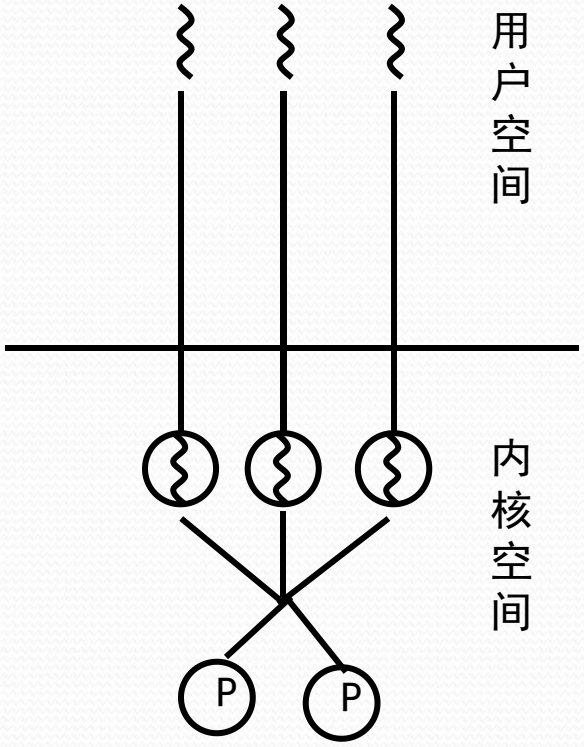
- 组合用户级线程/内核级线程设施
- 线程创建完全在用户空间中完成，线程的调度和同步也在应用程序中进行
- 一个应用中的多个用户级线程被映射到一些(小于等于用户级线程数目)内核级线程上
- 程序员可以针对特定应用和机器调节内核级线程的数目，以达到整体最佳结果
- 该方法将会结合纯粹用户级线程方法和内核级线程方法的优点，同时减少它们的缺点

线程混合式策略下的线程状态

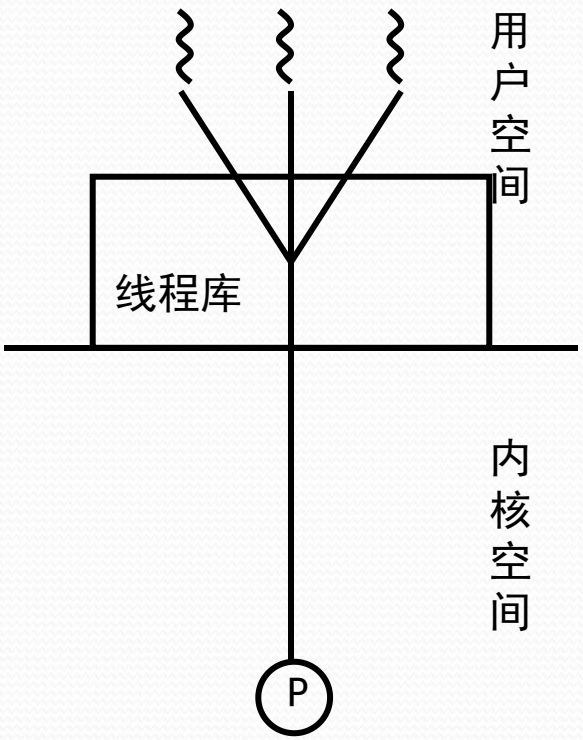
- KLT三态，系统调度负责
- ULT三态，用户调度负责
- 活跃态ULT代表绑定KLT的三态
- 活跃态ULT运行时可激活用户调度
- 非阻塞系统调用可使用Jacketing启动用户调度，调整活跃态ULT



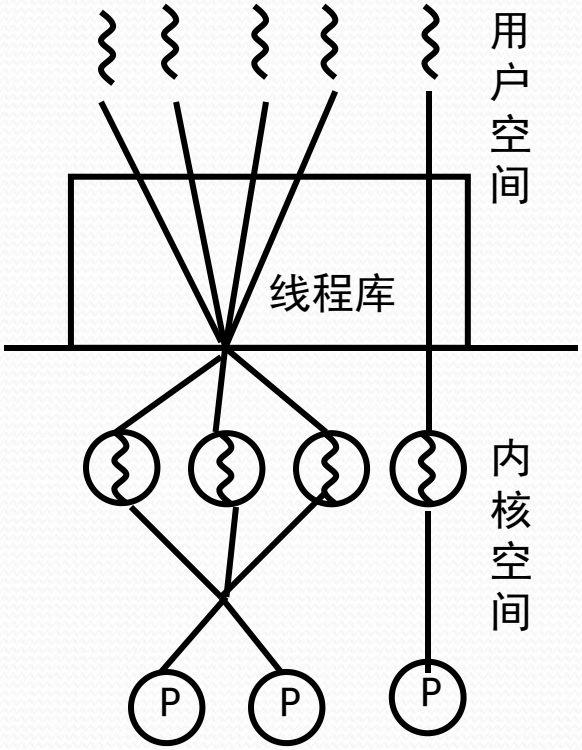
多线程实现的各种策略总结



1）内核级线程



2）用户级线程



3）混合式线程

Solaris多线程技术(补充)

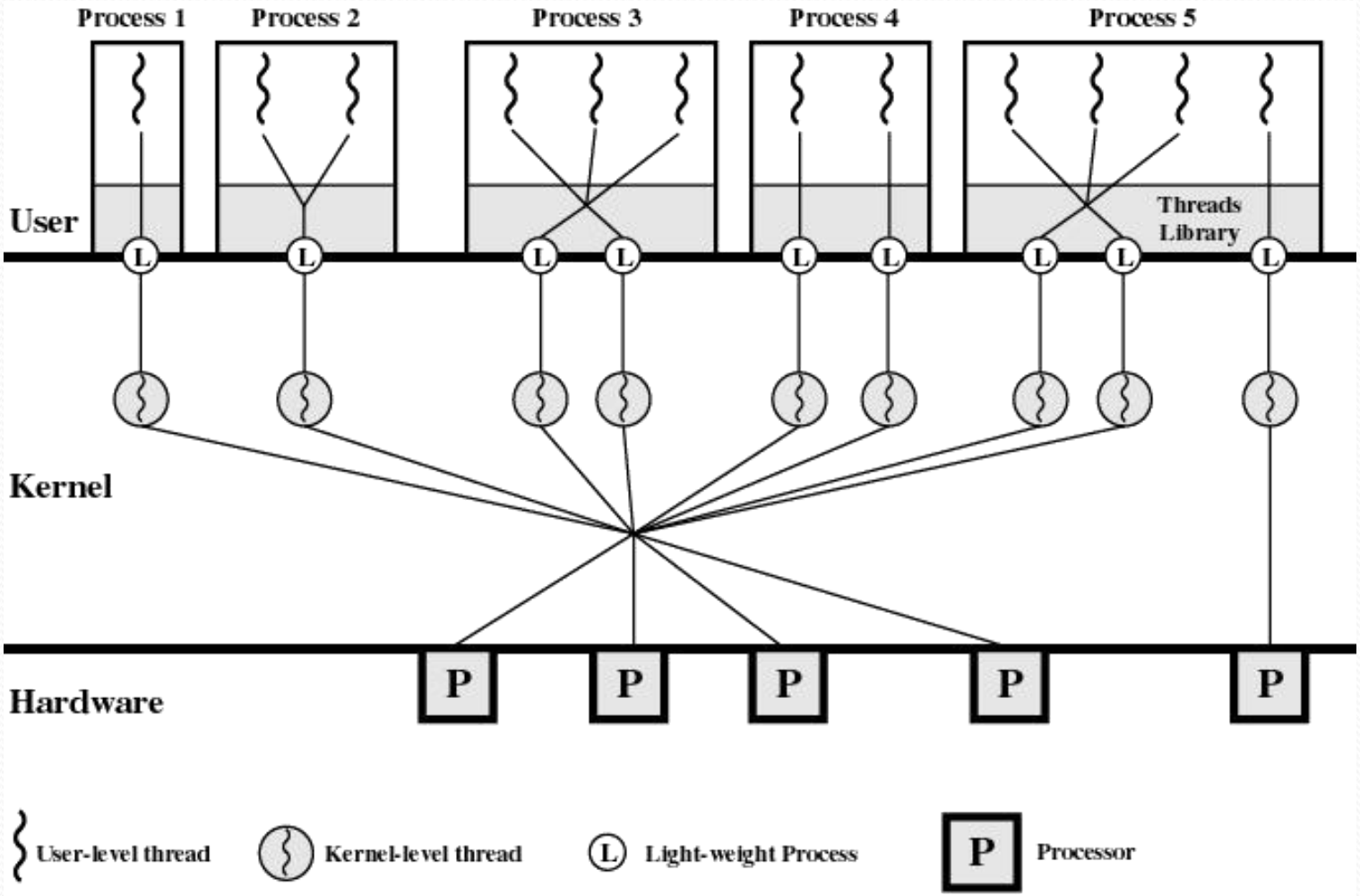


Figure 4.15 Solaris Multithreaded Architecture Example

Solaris

多线程技术

(补充)

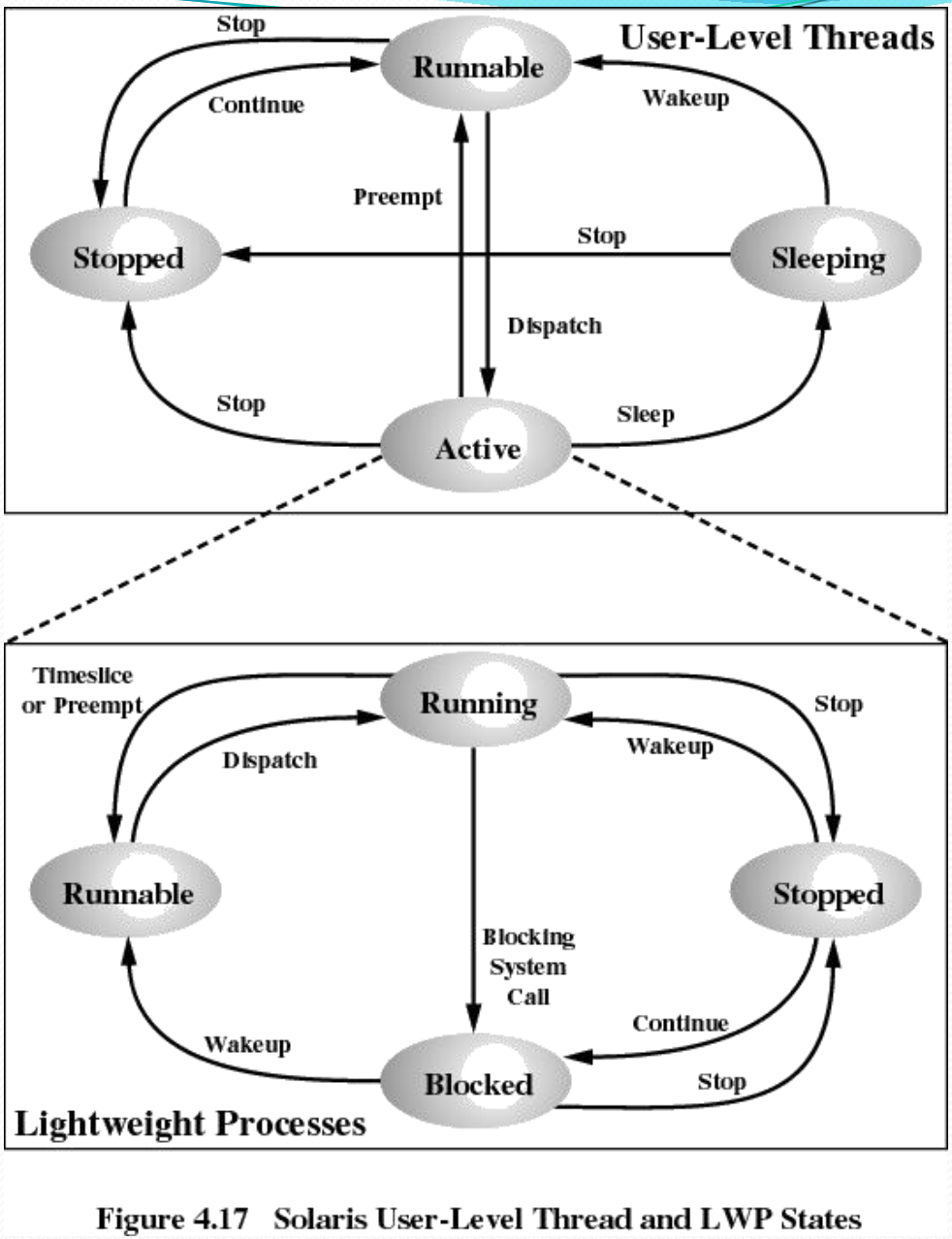


Figure 4.17 Solaris User-Level Thread and LWP States

南京大学
NANJING UNIVERSITY

计算机操作系统

2 处理器管理 – 2.5 处理器调度

2.5.1 处理器调度的层次

掌握处理器调度的层次

掌握高级调度

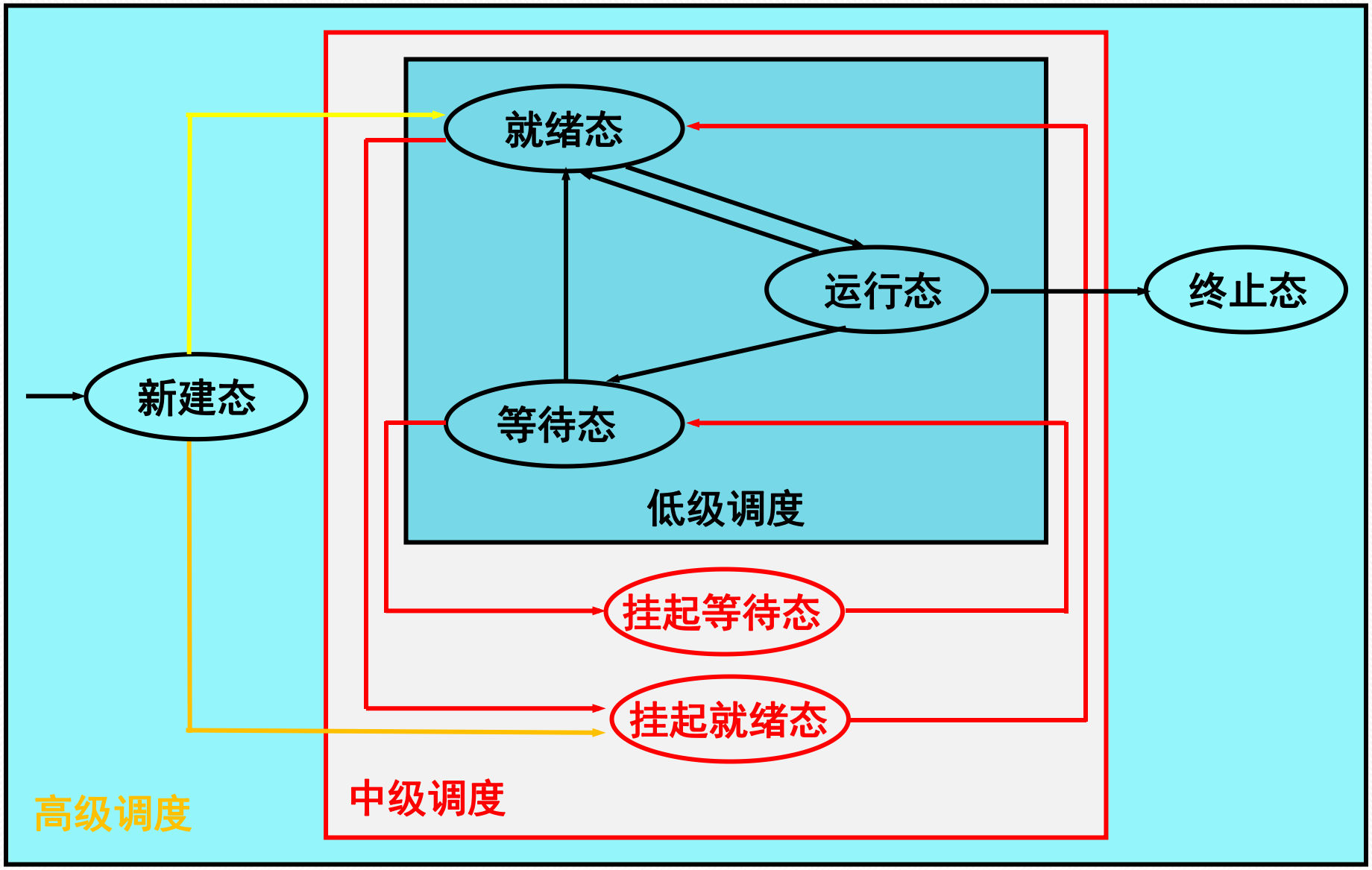
掌握中级调度

掌握低级调度

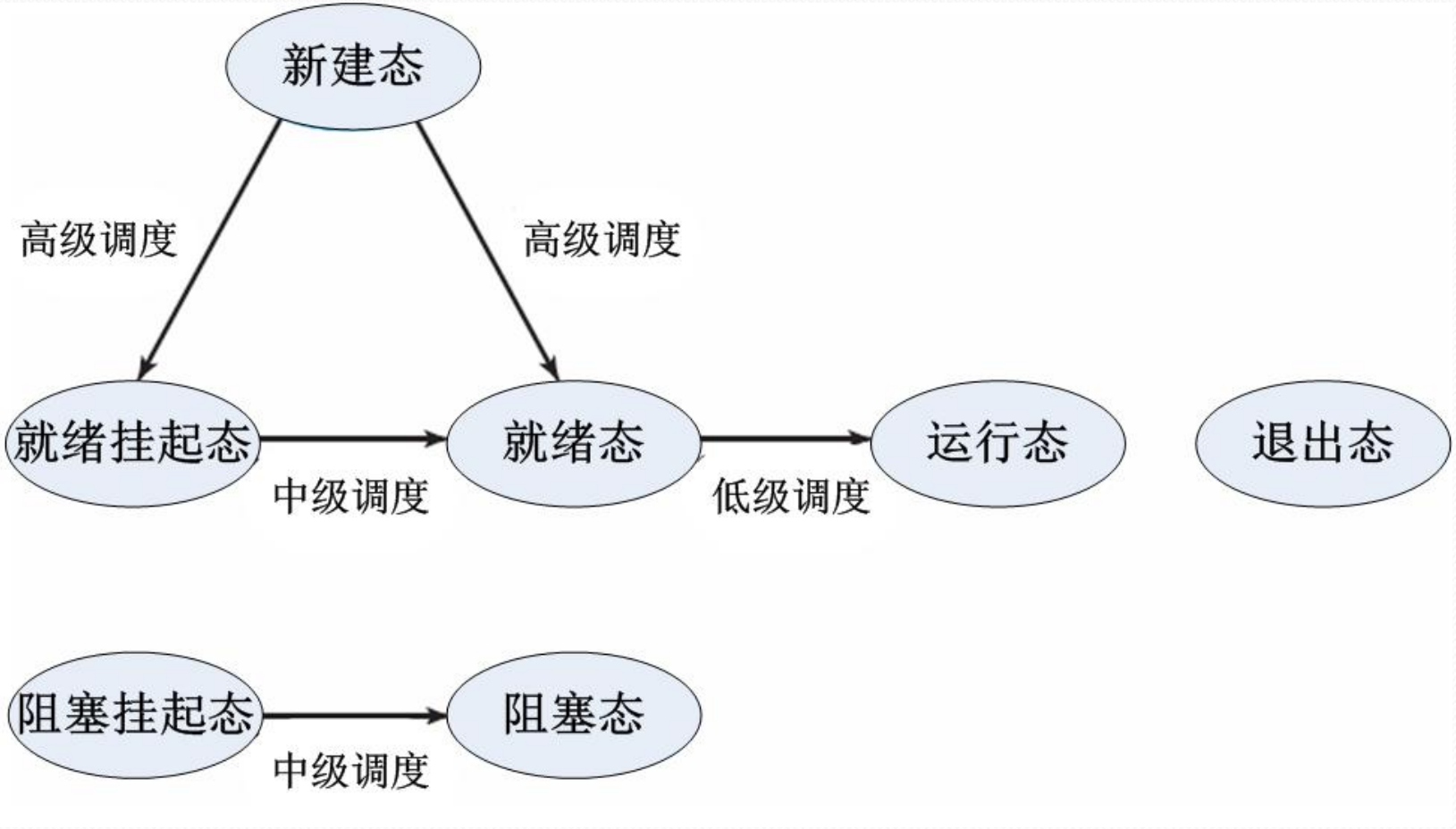
处理器调度的层次

- 高级调度：又称长程调度，作业调度
  - 决定能否加入到执行的进程池中
- 中级调度，又称平衡负载调度
  - 决定主存中的可用进程集合
- 低级调度：又称短程调度，进程调度
  - 决定哪个可用进程占用处理器执行

处理器调度层次与进程状态转换



处理器调度层次与关键状态转换



高级调度

- 分时OS中，高级调度决定：
  - 是否接受一个终端用户的连接
  - 命令能否被系统接纳并构成进程
  - 新建态进程是否加入就绪进程队列
- 批处理OS中，高级调度又称为作业调度，功能是按照某种原则从后备作业队列中选取作业进入主存，并为作业做好运行前的准备工作和完成后的善后工作

中级调度

- 引进中级调度是为了提高内存利用率和作业吞吐量
- 中级调度决定那些进程被允许驻留在主存中参与竞争处理器及其他资源，起到短期调整系统负荷的作用
- 中级调度把一些进程换出主存，从而使之进入“挂起”状态，不参与进程调度，以平顺系统的负载

低级调度

- 低级调度：又称处理器调度、进程调度、短程调度，按照某种原则把处理器分配给就绪态进程或内核级线程
- 进程调度程序：又称分派程序，操作系统中实现处理器调度的程序，是操作系统的最核心部分
- 处理器调度策略的优劣直接影响到整个系统的性能

低级调度的主要功能

- 记住进程或内核级线程的状态
- 决定某个进程或内核级线程什么时候获得处理器，以及占用多长时间
- 把处理器分配给进程或内核级线程
- 收回处理器

南京大学
NANJING UNIVERSITY

计算机操作系统

2 处理器管理 – 2.5 处理器调度

2.5.2 处理器调度算法

理解处理器调度算法的选择原则

理解先来先服务调度算法

掌握时间片轮转调度算法

掌握优先数调度算法

掌握分级调度算法

掌握彩票调度算法

选择处理器调度算法的原则

- 资源利用率：使得CPU或其他资源的使用率尽可能高且能够并行工作
- 响应时间：使交互式用户的响应时间尽可能小，或尽快处理实时任务
- 周转时间：提交给系统开始到执行完成获得结果为止的这段时间间隔称周转时间，应该使周转时间或平均周转时间尽可能短
- 吞吐量：单位时间处理的进程数尽可能多
- 公平性：确保每个用户每个进程获得合理的CPU份额或其他资源份额

优先数调度算法

- 根据分配给进程的优先数决定运行进程
  - 抢占式优先数调度算法
  - 非抢占式优先数调度算法
- 优先数的确定准则
  - 进程负担任务的紧迫程度
  - 进程的交互性
  - 进程使用外设的频度
  - 进程进入系统的时间长短

与进入系统时间相关的优先数

- 计算时间短(作业/进程)优先
- 剩余计算时间短进程优先
- 响应比高者(作业/进程)优先

  响应比 = 等待时间/进入时间
- 先来先服务：先进队先被选择
  - 多用于高级调度；低级调度中，以计算为主的进程过于优越

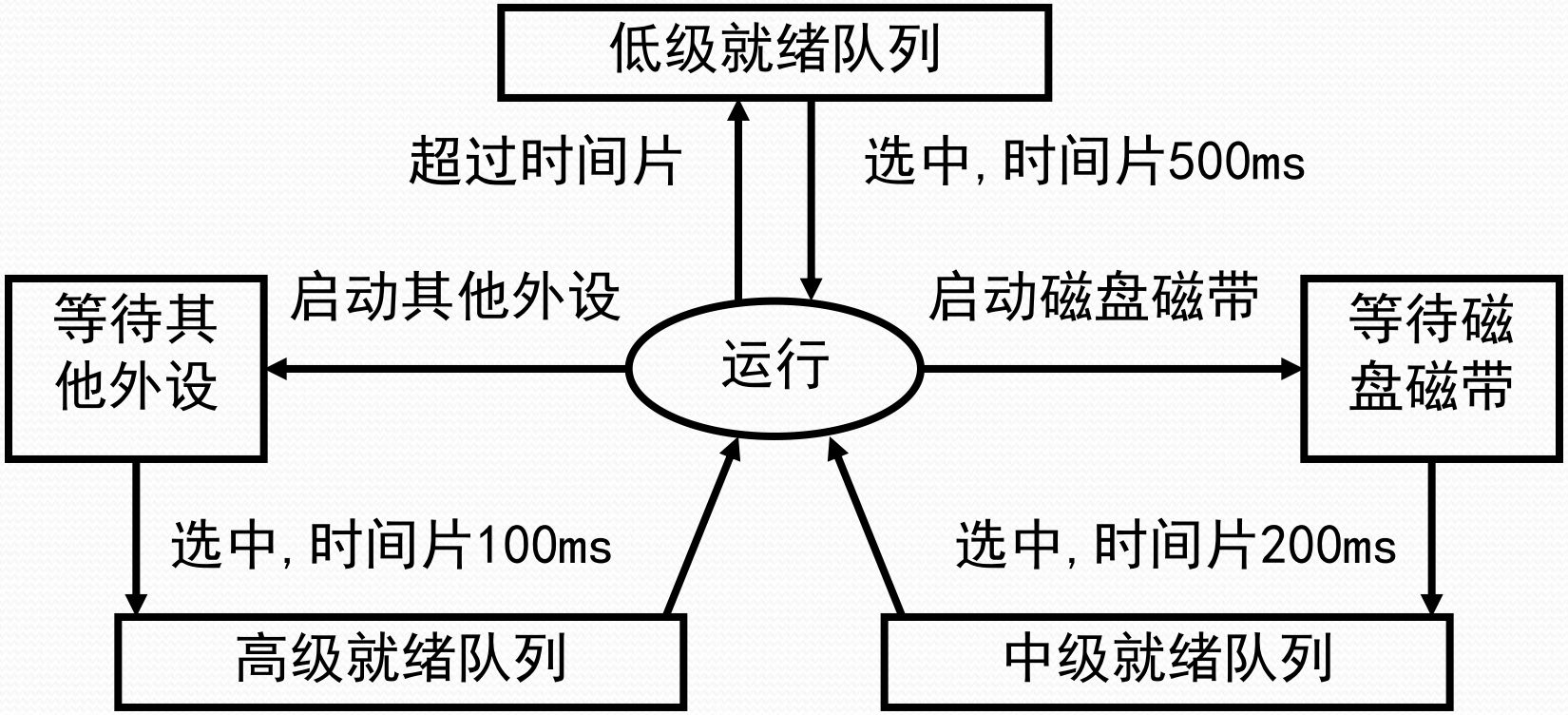
时间片轮转调度算法

- 根据各个进程进入就绪队列的时间先后轮流占有CPU一个时间片
- 时间片中断
- 时间片的确定：选择长短合适的时间片，过长则退化为先来先服务算法，过短则调度开销大
- 单时间片，多时间片和动态时间片

分级调度算法

- 又称多队列策略，反馈循环队列
- 基本思想
  - 建立多个不同优先级的就绪进程队列
  - 多个就绪进程队列间按照优先数调度
  - 高优先级就绪进程，分配的时间片短
  - 单个就绪进程队列中进程的优先数和时间片相同

分级调度算法的例


低级就绪队列
超过时间片
选中，时间片500ms
等待其他外设
启动其他外设
运行
启动磁盘磁带
等待磁盘磁带
选中，时间片100ms
选中，时间片200ms
高级就绪队列
中级就绪队列

分级调度算法的分级原则

- 一般分级原则
  - 外设访问，交互性，时间紧迫程度，系统效率，用户立场，…
- 现代操作系统的实现模型
  - 多个高优先级的实时进程队列，如：硬实时、网络、软实时
  - 多个分时任务的进程队列，根据基准优先数和执行行为调整
  - 队列数可能多达32-128个

彩票调度算法

- 基本思想：为进程发放针对系统各种资源（如CPU时间）的彩票；当调度程序需要做出决策时，随机选择一张彩票，持有该彩票的进程将获得系统资源
- 合作进程之间的彩票交换

2.5 处理器调度(补充内容)

调度算法

调度算法

- 1 短程调度准则
- 2 优先级调度
- 3 调度的模式
- 4 调度算法

1 短程调度准则

与性能相关

- 面向用户
  - 周转时间
  - 响应时间
  - 最后期限
- 面向系统
  - 吞吐量
  - 处理器利用率

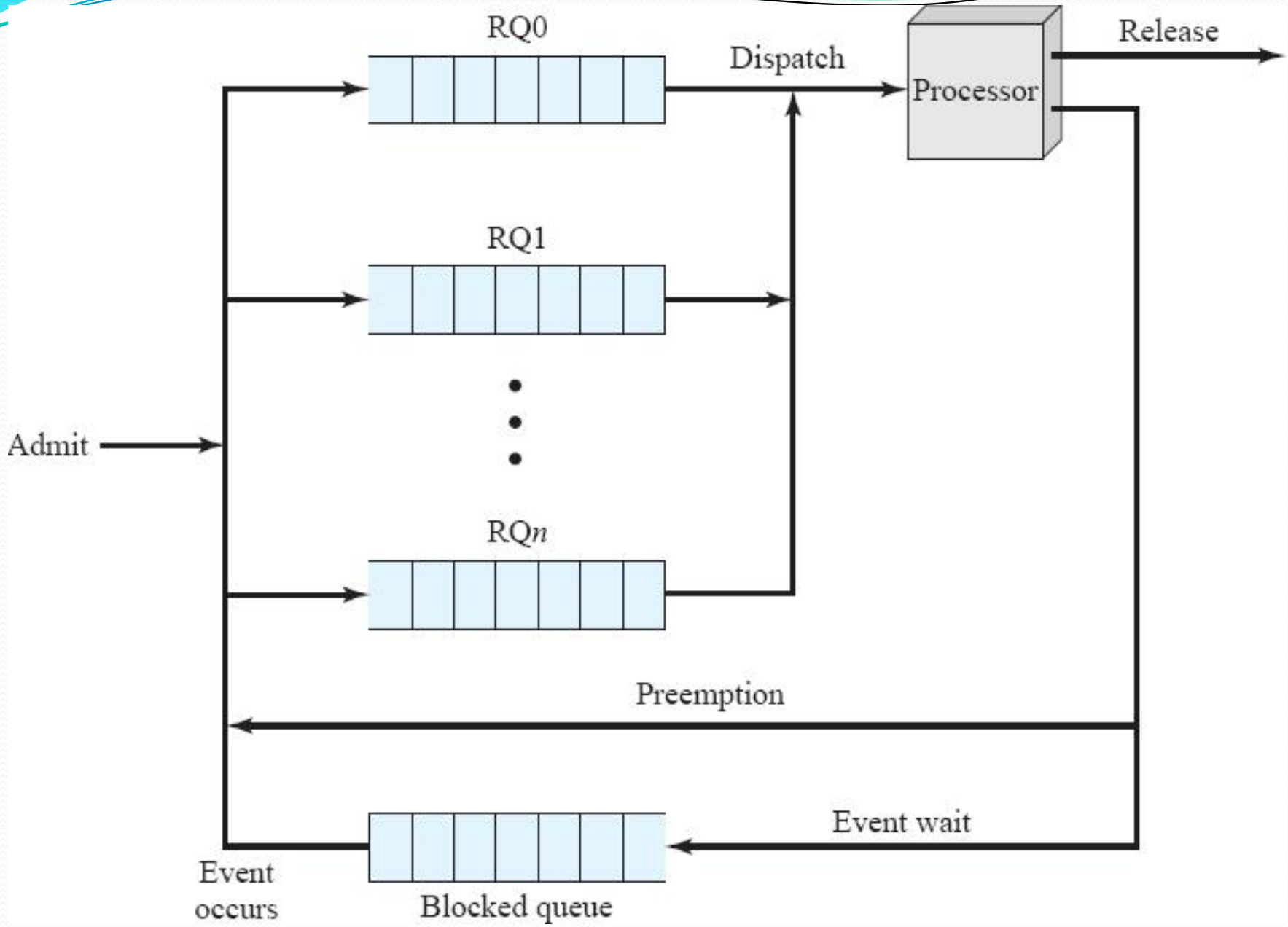
1 短程调度准则

与性能无关

- 面向用户
  - 可预测性
- 面向系统
  - 公平
  - 强制优先级
  - 平衡资源

2 优先级调度

- 调度器总是选择优先级较高的进程
- 提供多个就绪队列 (一组就绪队列) 代表各个级别的优先级
- 问题：低优先级有可能饥饿
  - 一个进程的优先级应该随着它的时间或执行的历史而变化


RQ0
Dispatch
Processor
Release
RQ1
Admit
RQn
Preemption
Event wait
Event
occurs
Blocked queue

3 调度的模式

- 非抢占式
  - 一个进程一旦处于运行态，它就不断执行直到终止，或者为等待I/O或请求某些操作系统服务而阻塞自己

3 调度的模式

- 抢占式
  - 当前正在运行的进程可能被操作系统中断，并转移到就绪态，关于抢占的决策可能是在一个进程到达时，或者在一个中断发生后把一个被阻塞的进程置为就绪态时，或者基于周期性的时间中断
  - 与非抢占式相比，抢占式可能会导致较大的开销，但是可能对所有进程提供更好的服务，可以避免任何一个进程独占处理器太长时间

4 调度算法

- FCFS (先来先服务)
- RR (时间片轮转)
- SPN (最短进程优先)
- SRT (最短剩余时间优先)
- HRRF (最高响应比优先)
- Feedback (多级反馈调度)

进程调度(例)

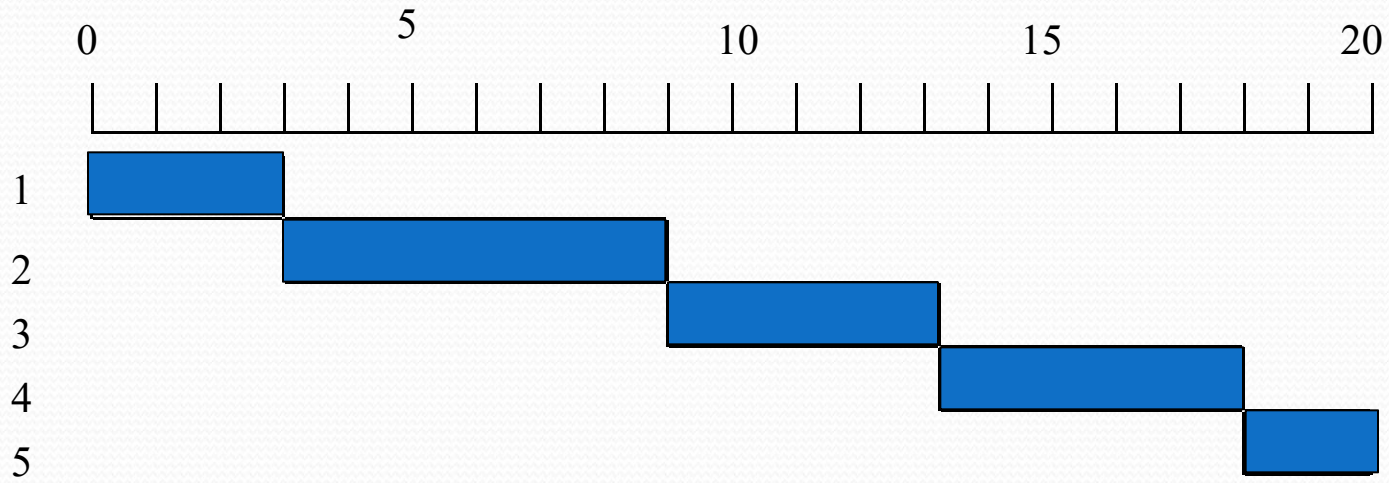
Process	Arrival Time	Service Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

FCFS (先来先服务)

- 当某个进程就绪时，都加入就绪队列(ready queue)
- 当前正在运行的进程停止执行时，选择在就绪队列到存在时间最长的进程运行

FCFS (先来先服务)

Process	Arrival Time	Service Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2



FCFS (先来先服务)

- 一个短进程可能不得不等待很长时间才能获得执行
- 偏袒计算为主的进程
  - I/O多的进程不得不等待计算为主的进程做完

RR (时间片轮转)

- 基于时钟做抢占式调度
- 以一个周期性间隔产生时钟中断，当中断发生时，当前正在运行的进程被置于就绪队列中，然后基于FCFS策略选择下一个就绪进程运行

RR (时间片轮转)

Process	Arrival Time	Service Time
A	0	3
B	2	4
C	4	2

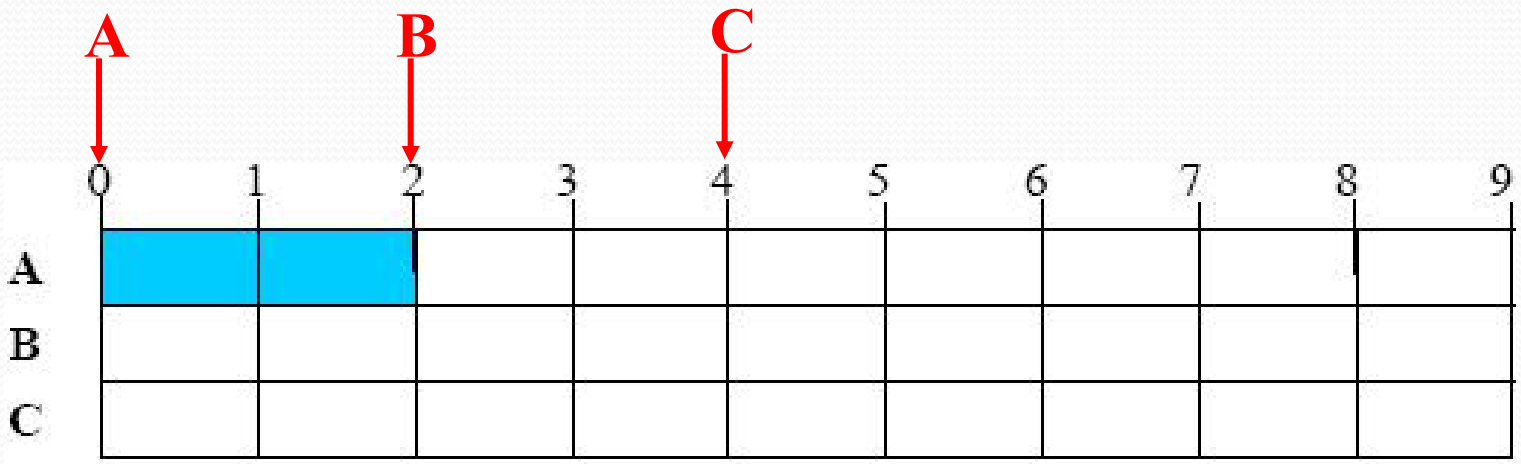
Time=0~2

Q= 空

队列

Running=A

B在2ms时刻到达



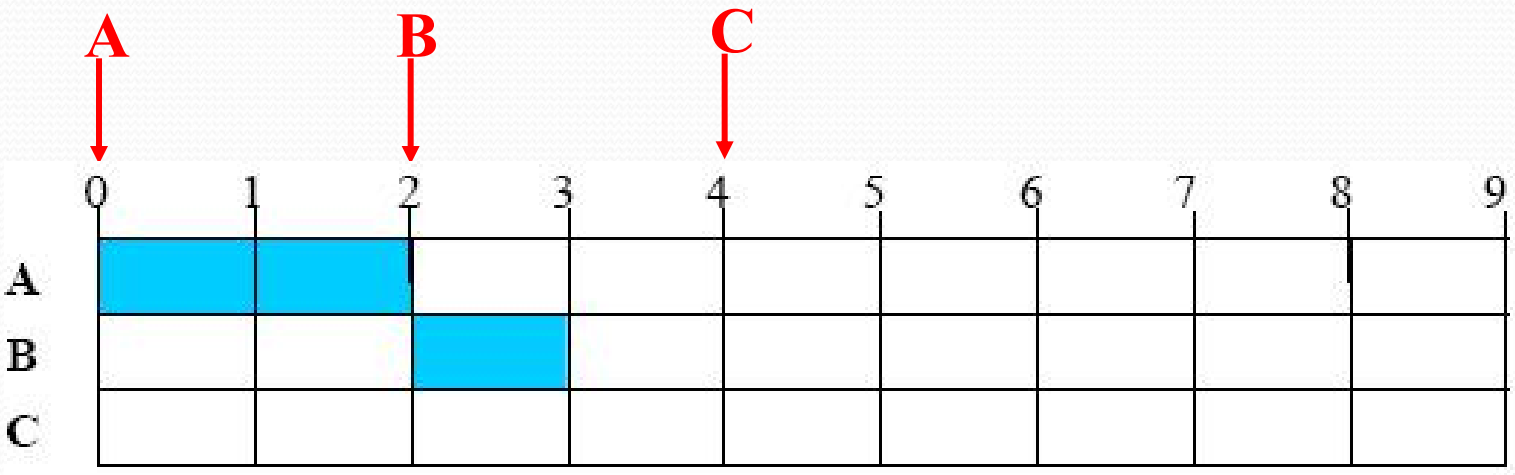
RR (时间片轮转)

Process	Arrival Time	Service Time
A	0	3
B	2	4
C	4	2

Time=2~3

Q=A

Running=B



RR (时间片轮转)

Process	Arrival Time	Service Time
A	0	3
B	2	4
C	4	2

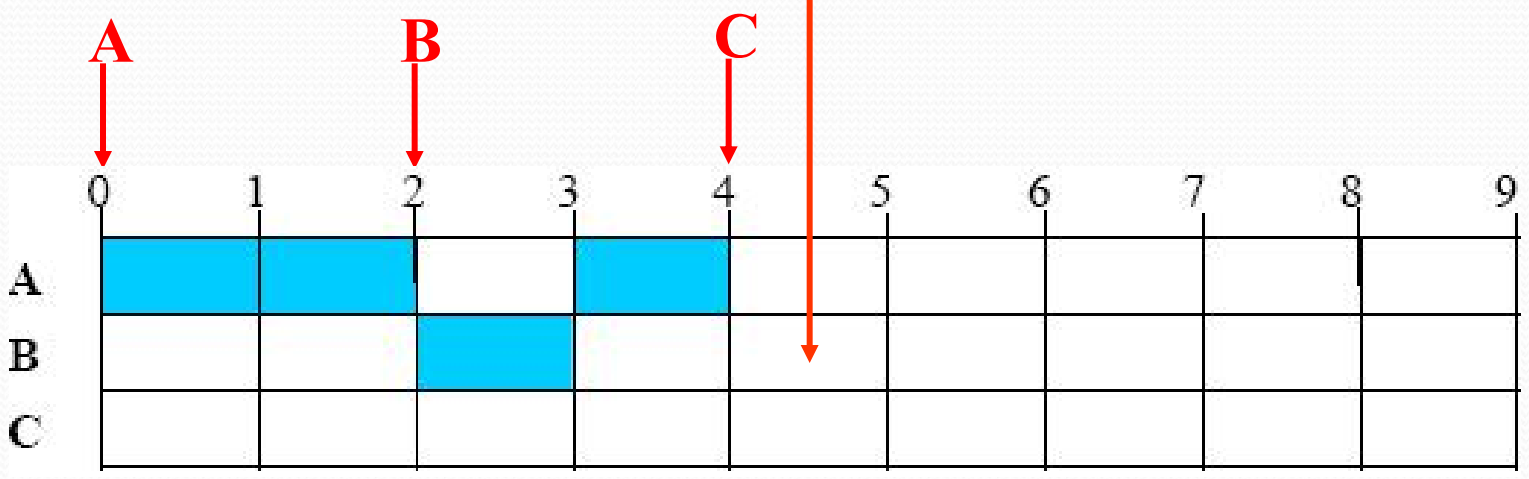
Time=3~4

Q=B

Time=4

Running=A and finished

Q=BC



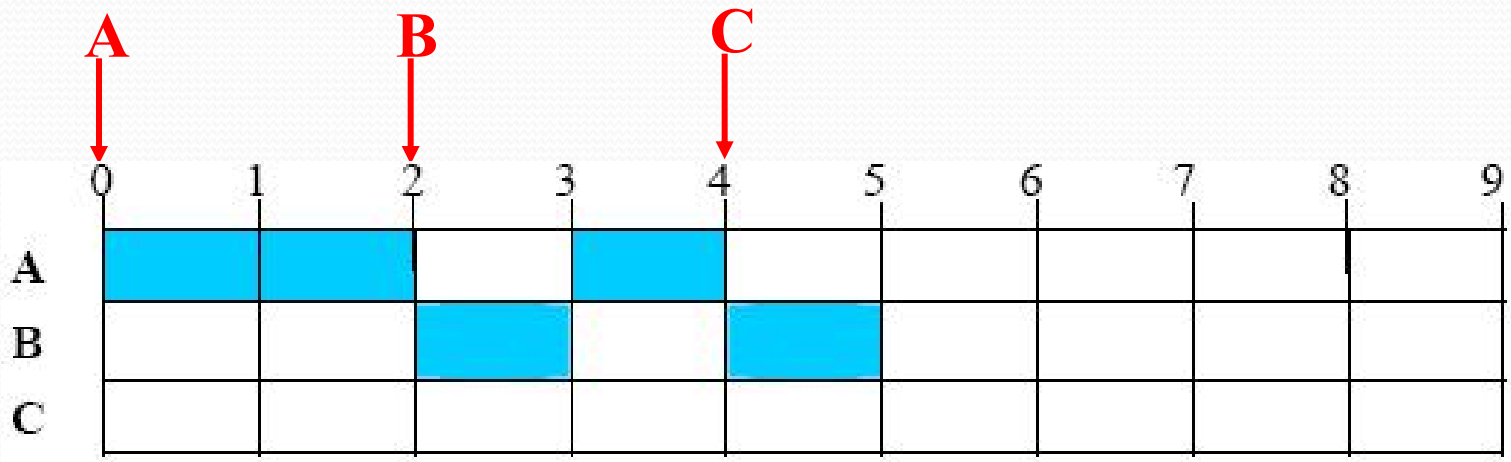
RR (时间片轮转)

Process	Arrival Time	Service Time
A	0	3
B	2	4
C	4	2

Time=4~5

Q=C

Running=B



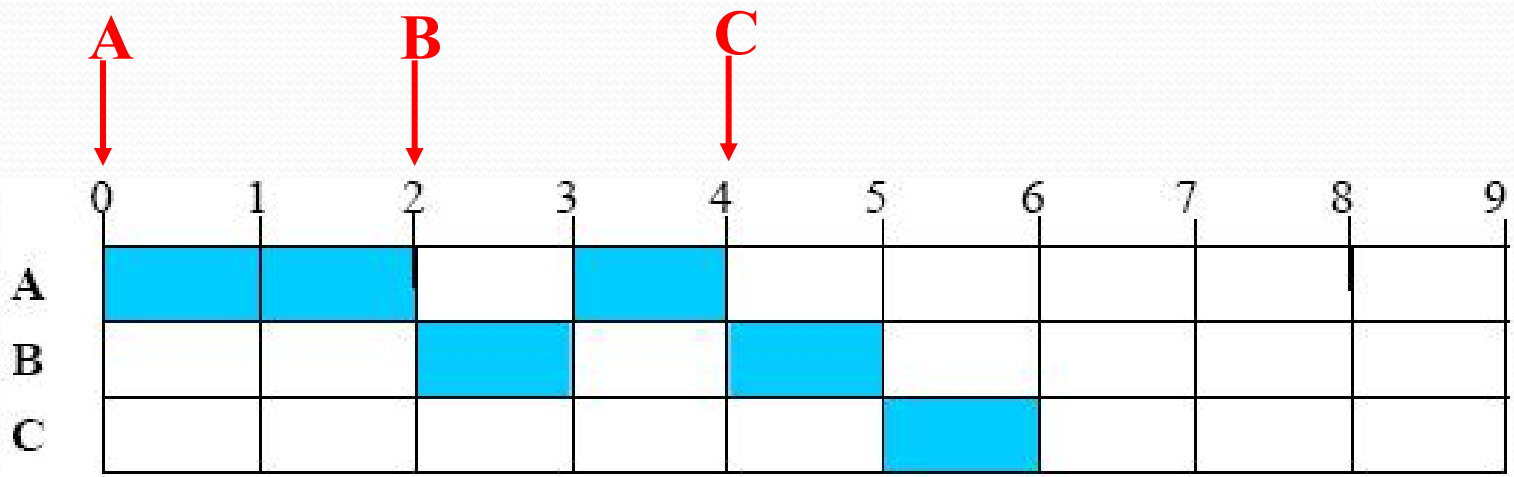
RR (时间片轮转)

Process	Arrival Time	Service Time
A	0	3
B	2	4
C	4	2

Time = 5~6

Q=B

Running=C



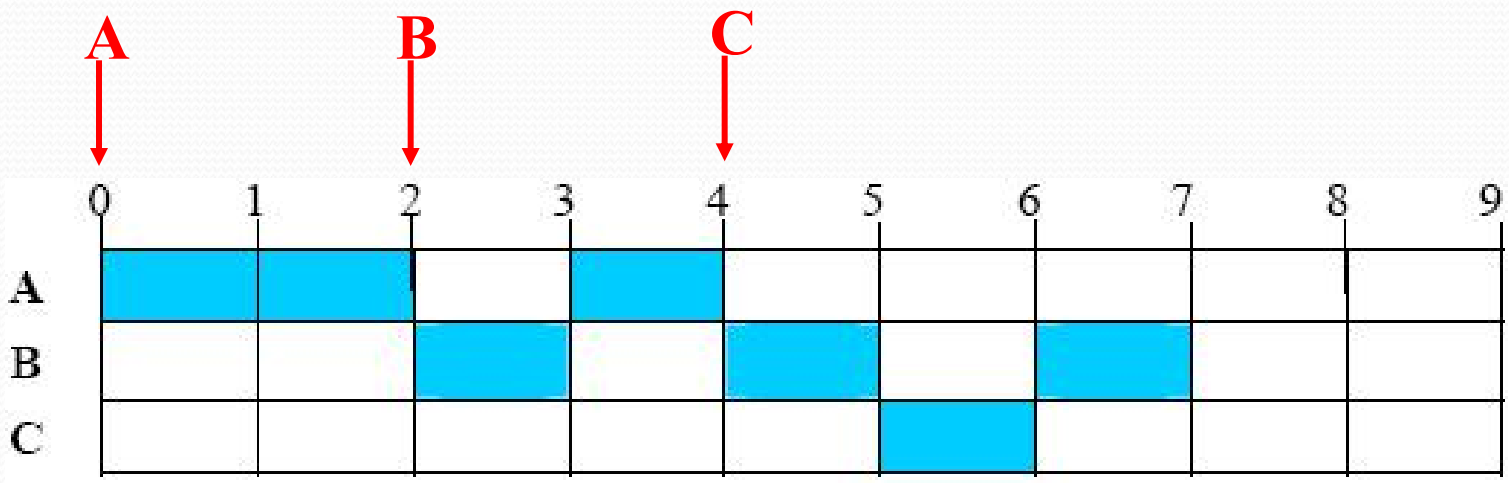
RR (时间片轮转)

Process	Arrival Time	Service Time
A	0	3
B	2	4
C	4	2

Time = 6~7

Q=C

Running=B



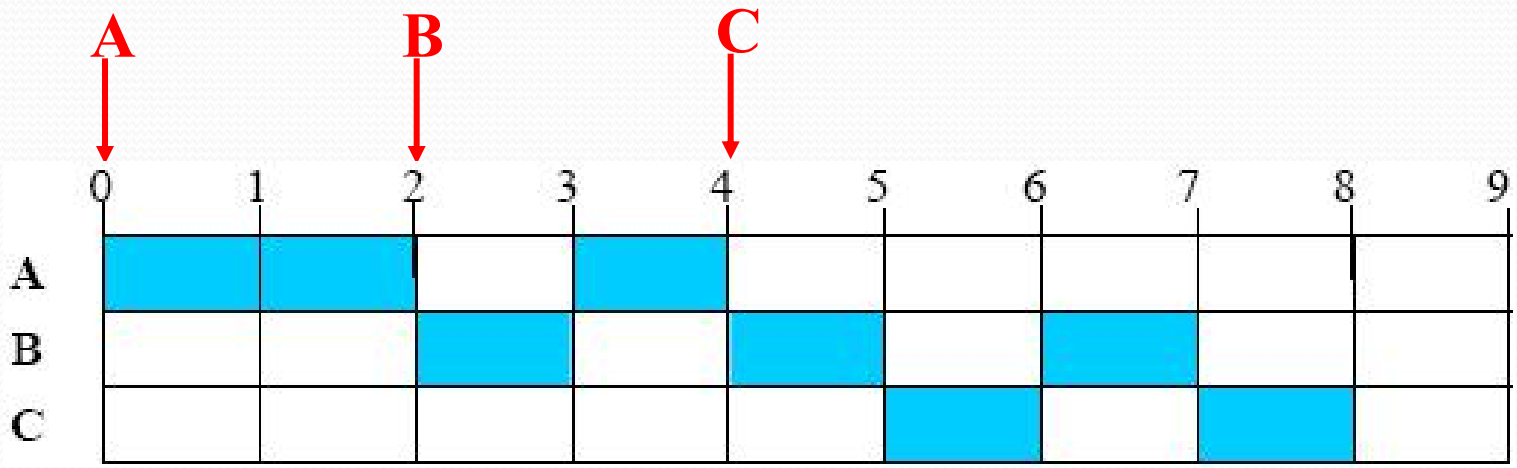
RR (时间片轮转)

Process	Arrival Time	Service Time
A	0	3
B	2	4
C	4	2

Time = 7~8

Q=B

Running=C and finished



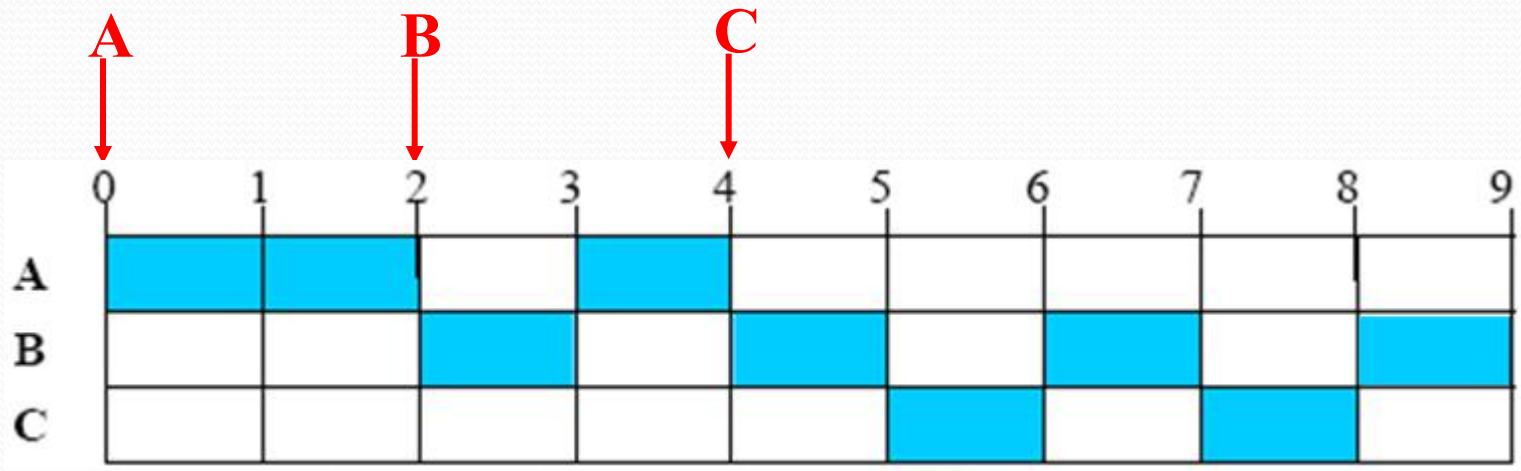
RR (时间片轮转)

Process	Arrival Time	Service Time
A	0	3
B	2	4
C	4	2

Time = 7~8

Q=B

Running=C and finished

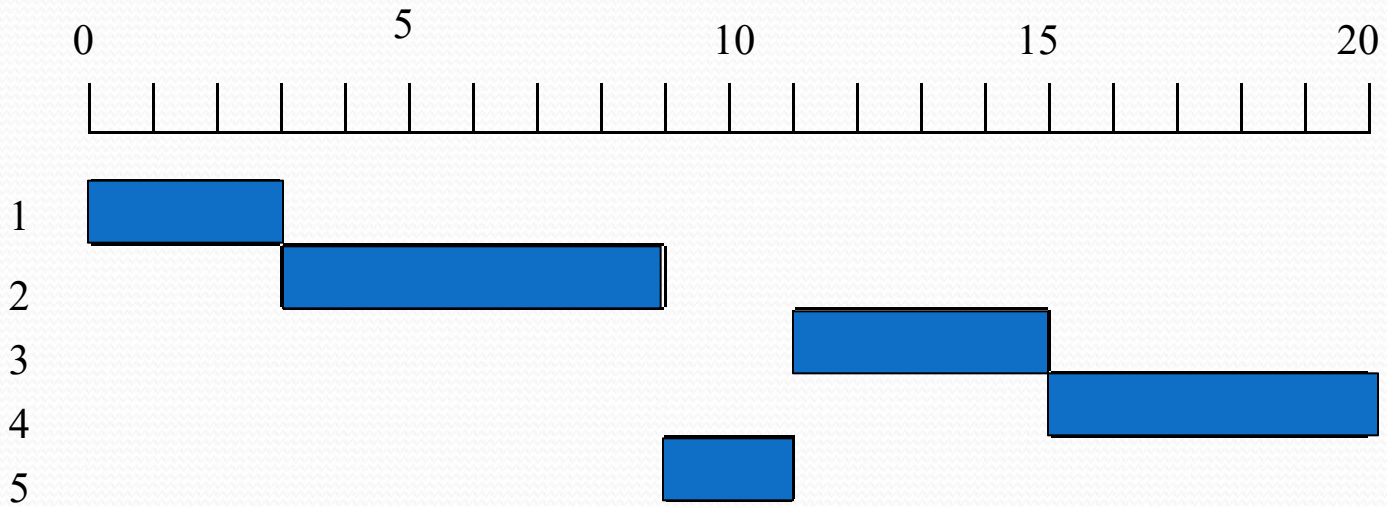


SPN (最短进程优先)

- 非抢占式调度
- 选择所需处理时间最短的进程
- 短进程将会越过长进程，优先获得调度

SPN (最短进程优先)

Process	Arrival Time	Service Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2



SPN (最短进程优先)

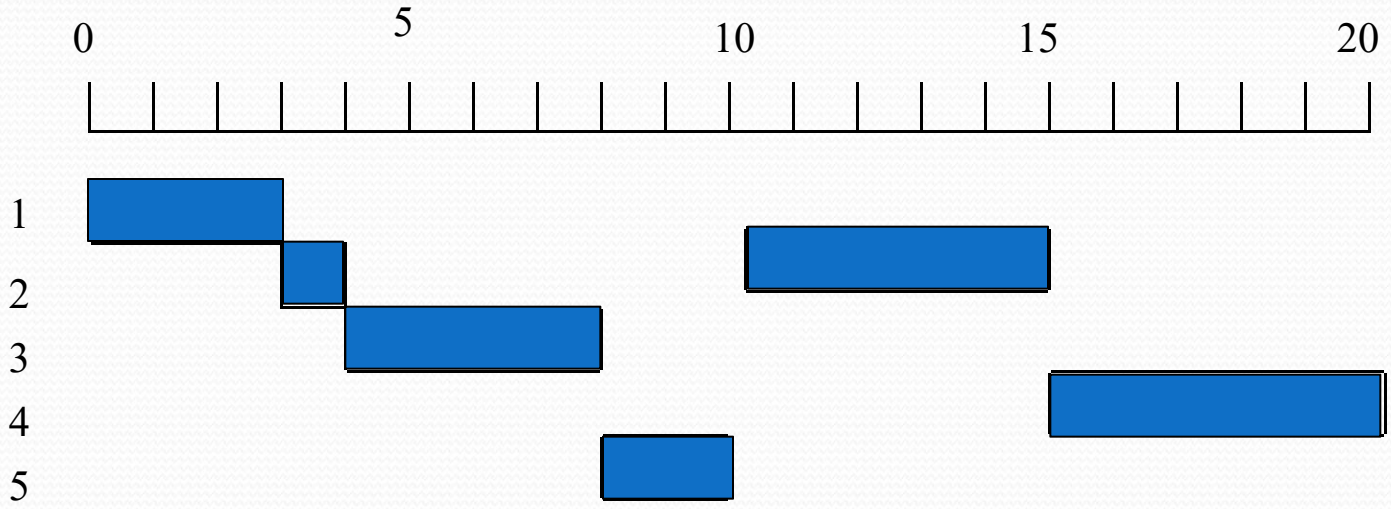
- 问题
  - 只要持续不断地提供更短的进程，长进程就有可能饿死。

SRT（最短剩余时间优先）
Shortest Remaining Time

- 抢占式调度
- 调度器总是选择预期剩余时间更短的进程
- 当一个新进程加入就绪队列，他可能比当前运行的进程具有更短的剩余时间，只要该新进就绪，调度器就可能抢占当前正在运行的进程

SRT (最短剩余时间优先)

Process	Arrival Time	Service Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2



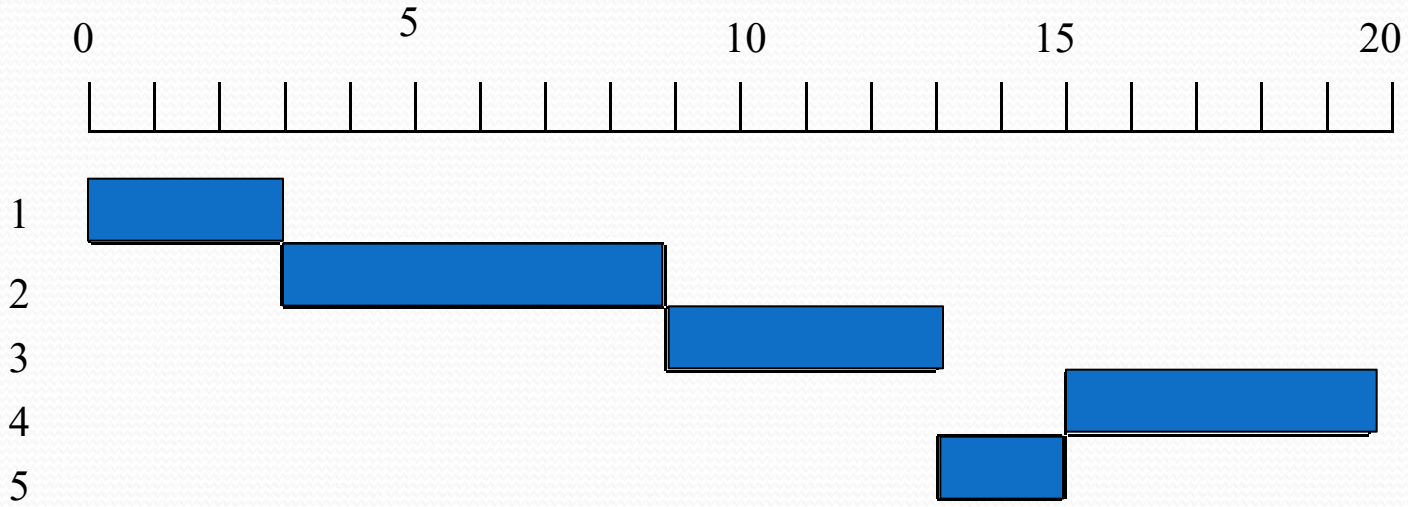
HRRN (最高响应比优先)
Highest Response Ratio Next

- 选择响应比最高的

$$\frac{\text{等待时间} + \text{期待服务时间}}{\text{期待服务时间}}$$

HRRN (最高响应比优先)
Highest Response Ratio Next

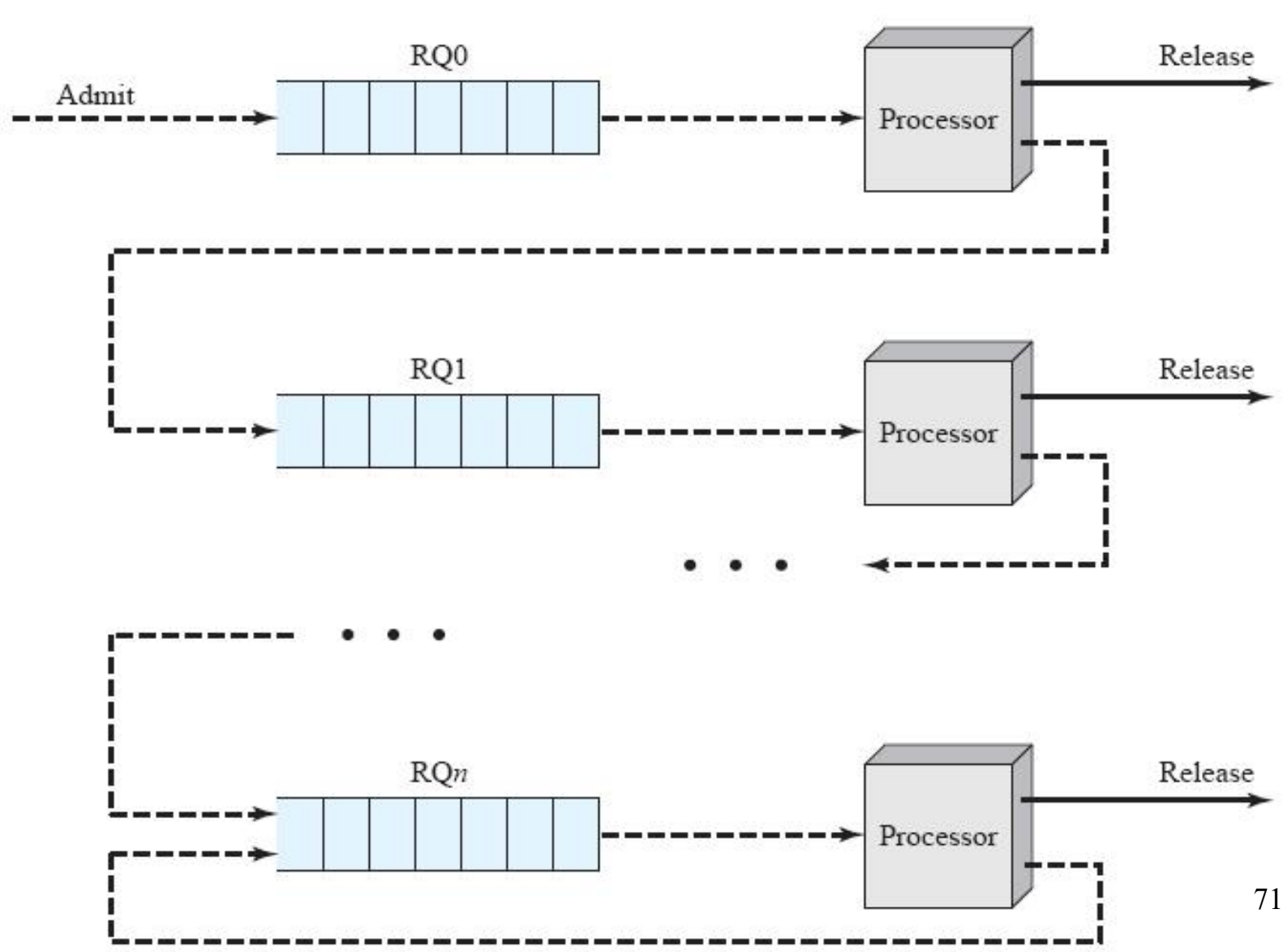
Process	Arrival Time	Service Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2



多级反馈调度 Feedback

- 基本思想
  - 建立多个不同优先级的就绪进程队列
  - 多个就绪进程队列之间按照优先数调度
  - 高优先级的就绪进程，分配的时间片短
  - 单个就绪进程队列中的进程的优先数和时间片相同，按照先来先服务算法调度
- 分级原则：外设访问，交互性，时间紧迫程度，系统效率，用户立场，...

Feedback算法


RQ0
Admit
Processor
Release
RQ1
Processor
Release
RQn
Processor
Release

Feedback (q=1)

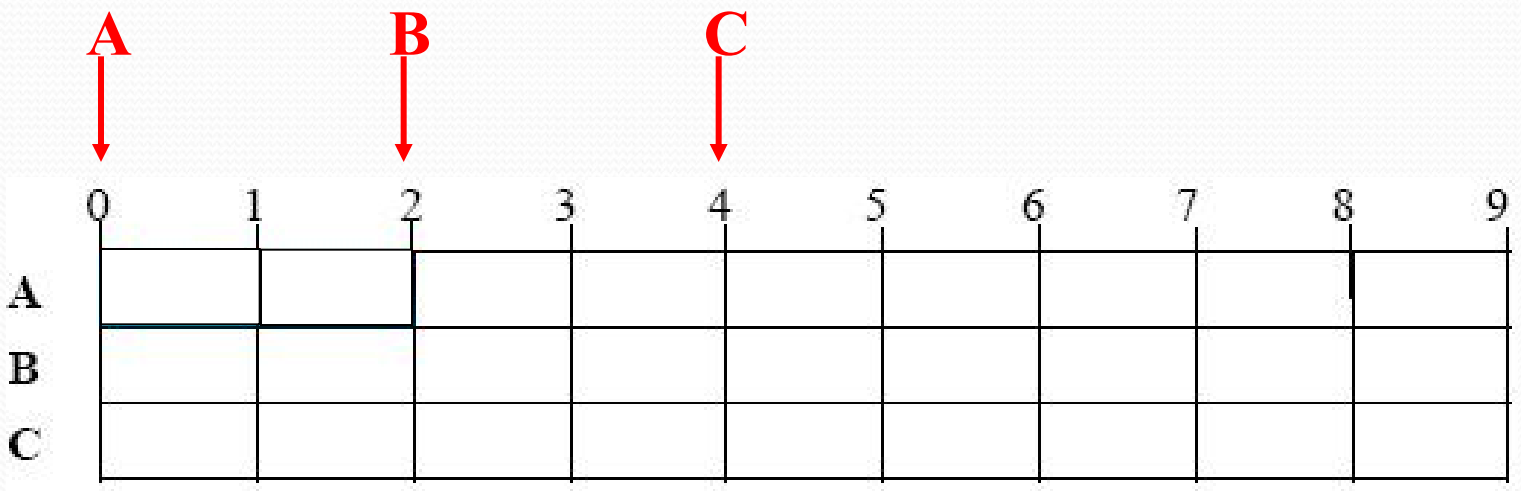
Time=0

RQ0= A

RQ1=

RQ2=

Process	Arrival Time	Service Time
A	0	3
B	2	4
C	4	2



- 当一个进程第一次进入系统时，它被放置在RQ0，当它第一次被抢占后并返回就绪状态时，它被放置在RQ1。在随后的时间里，每当它被抢占时，它被降级到下一个低优先级队列中。一个短进程很快会执行完，不会在就绪队列中降很多级。

Feedback (q=1)

Time=0~2

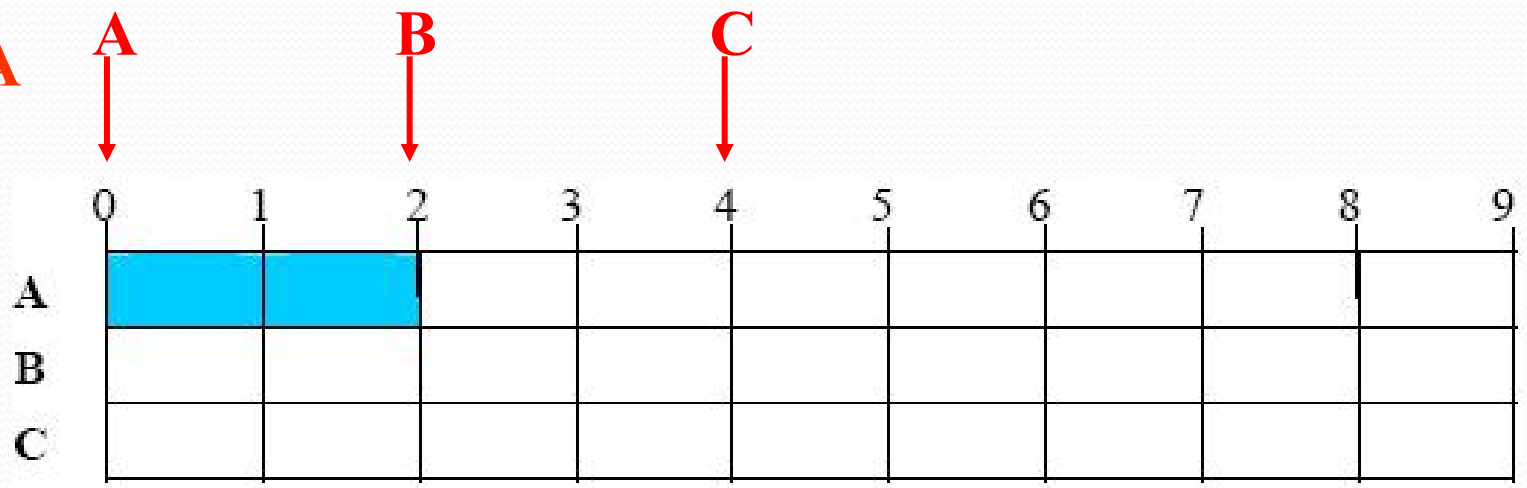
RQ0=

RQ1=

RQ2=

Process	Arrival Time	Service Time
A	0	3
B	2	4
C	4	2

Running=A



Feedback (q=1)

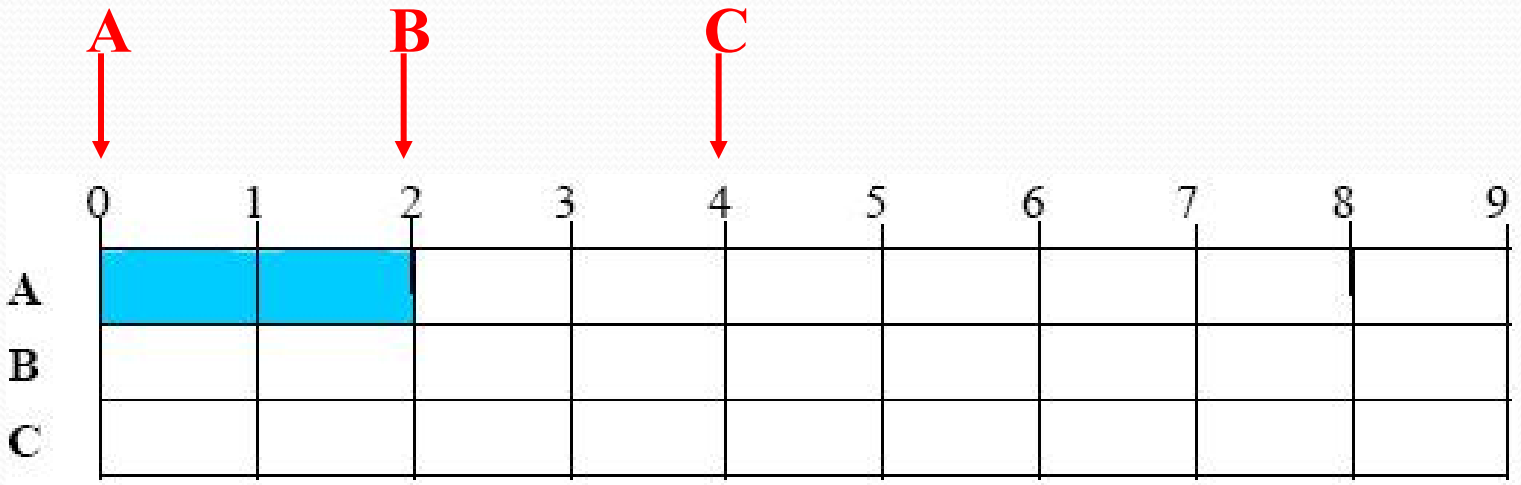
Time=2

RQ0= B

RQ1=

RQ2=

Process	Arrival Time	Service Time
A	0	3
B	2	4
C	4	2



Feedback (q=1)

Time=2~3

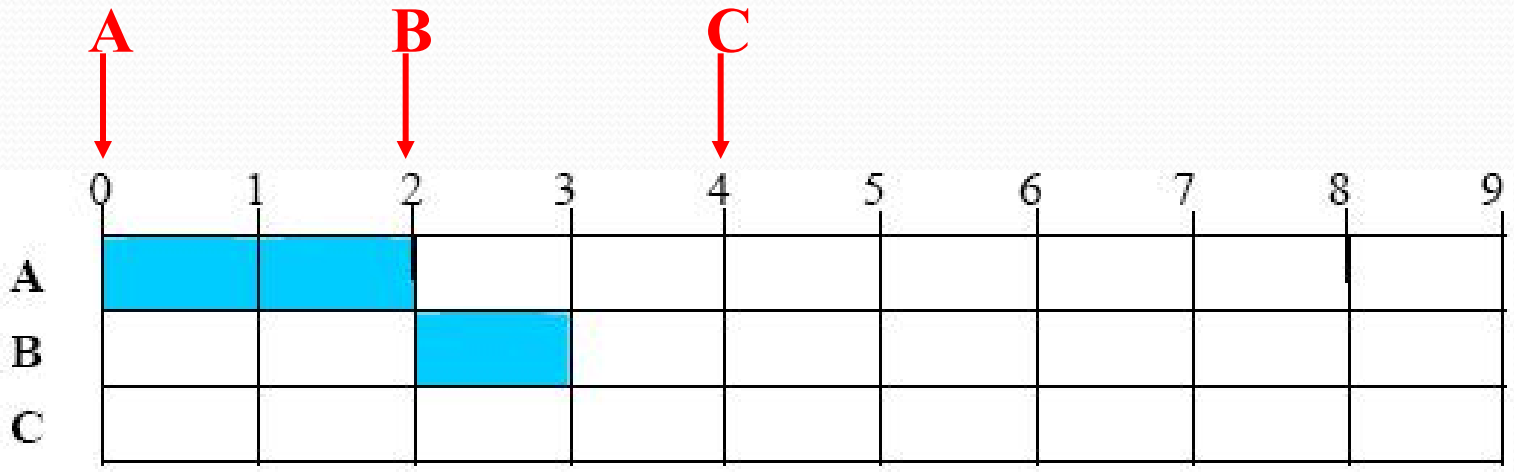
RQ0=

RQ1= A

RQ2=

Running=B

Process	Arrival Time	Service Time
A	0	3
B	2	4
C	4	2



Feedback (q=1)

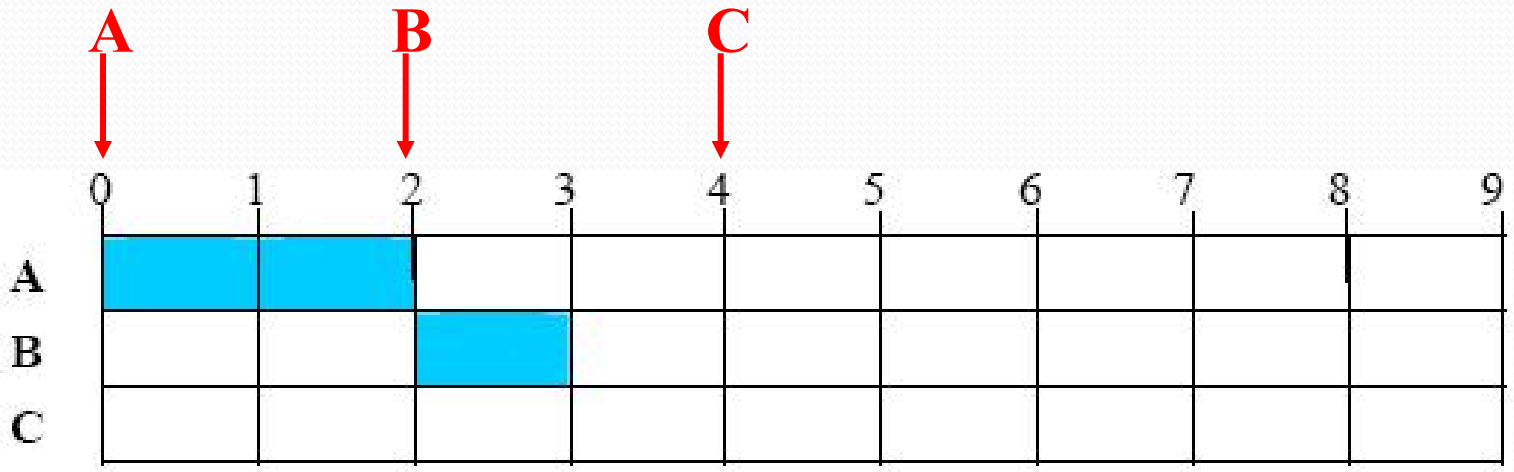
Time=3

RQ0=

RQ1= A

RQ2=

Process	Arrival Time	Service Time
A	0	3
B	2	4
C	4	2



Feedback (q=1)

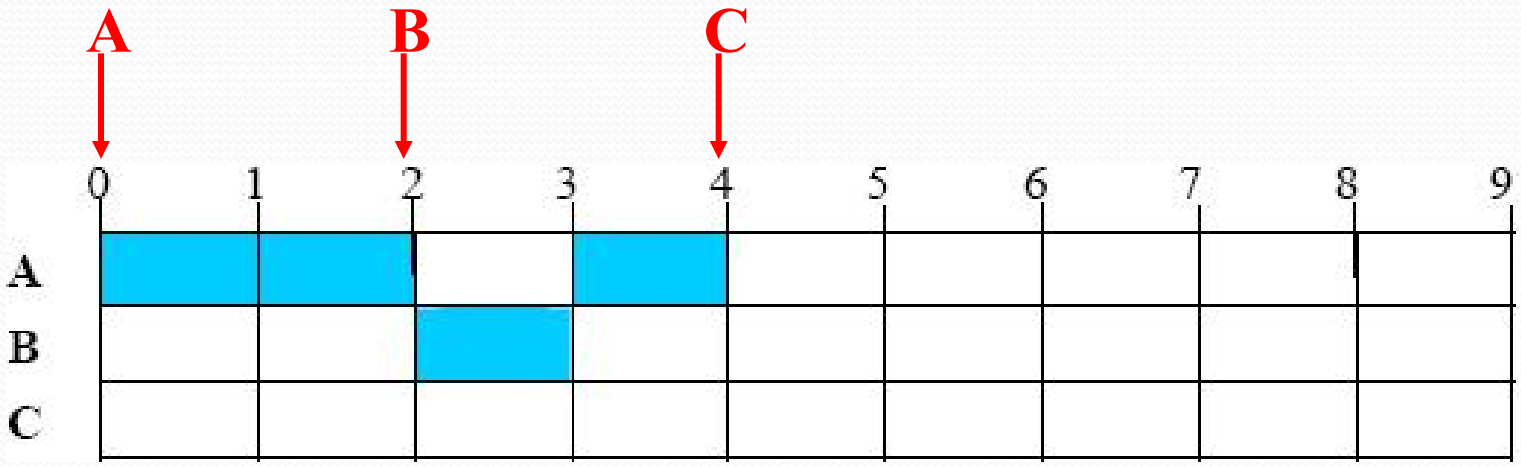
Time=3~4

RQ0=

RQ1= B

RQ2=

Process	Arrival Time	Service Time
A	0	3
B	2	4
C	4	2



Running=A and finished

Feedback (q=1)

Time=4

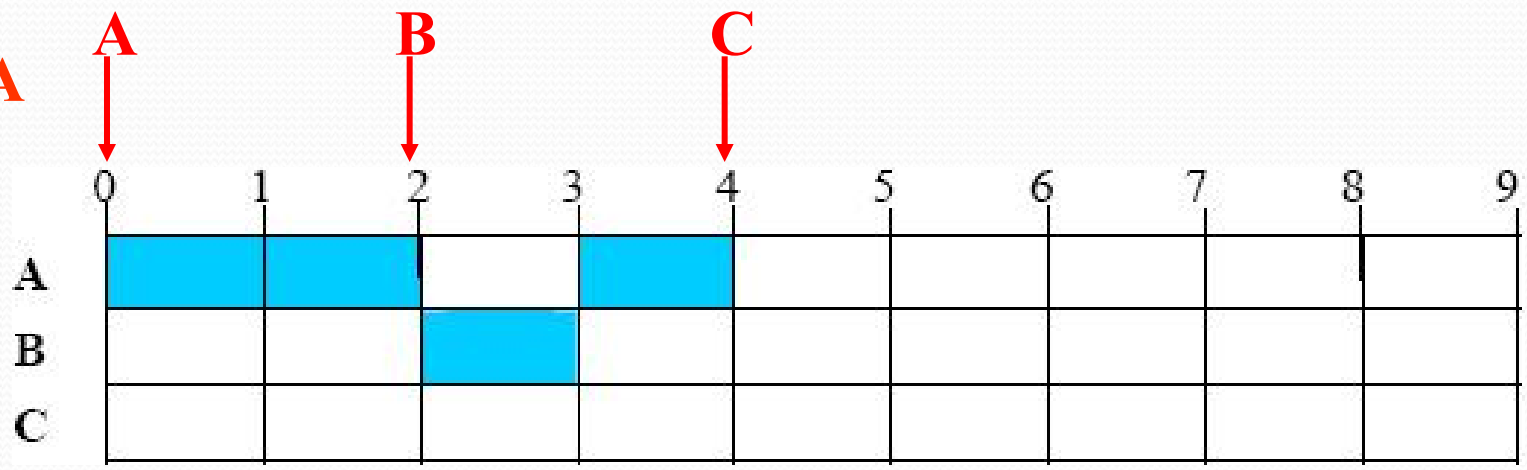
RQ0= C

RQ1= B

RQ2=

Process	Arrival Time	Service Time
A	0	3
B	2	4
C	4	2

Running=A



Feedback (q=1)

Time=4~5

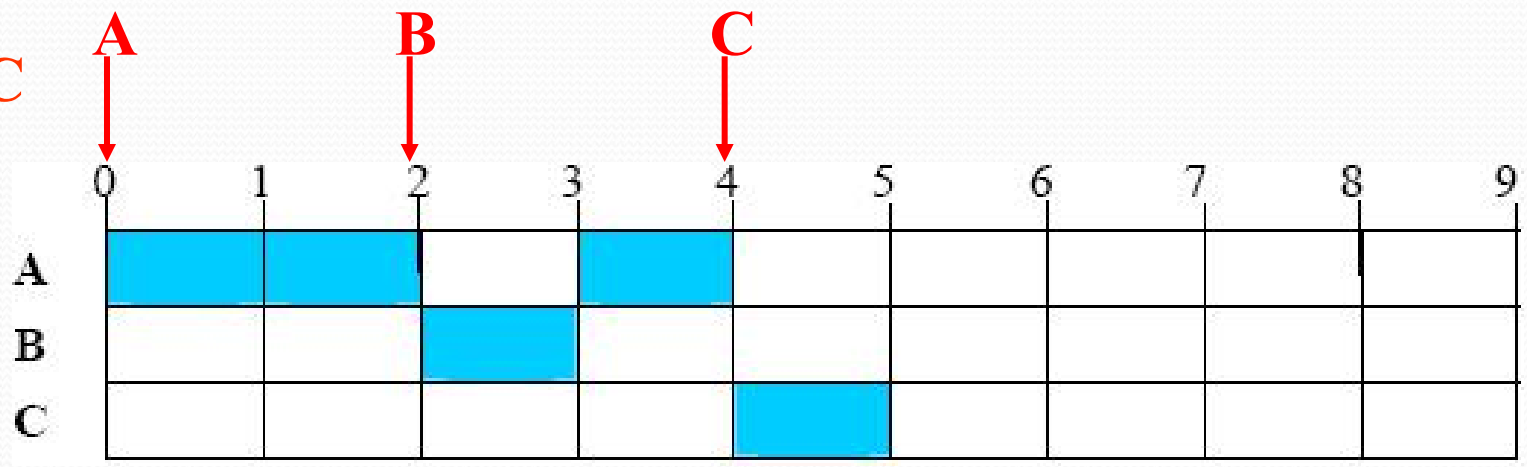
RQ0=

RQ1= B

RQ2=

Process	Arrival Time	Service Time
A	0	3
B	2	4
C	4	2

Running=C



Feedback (q=1)

Time=5~6

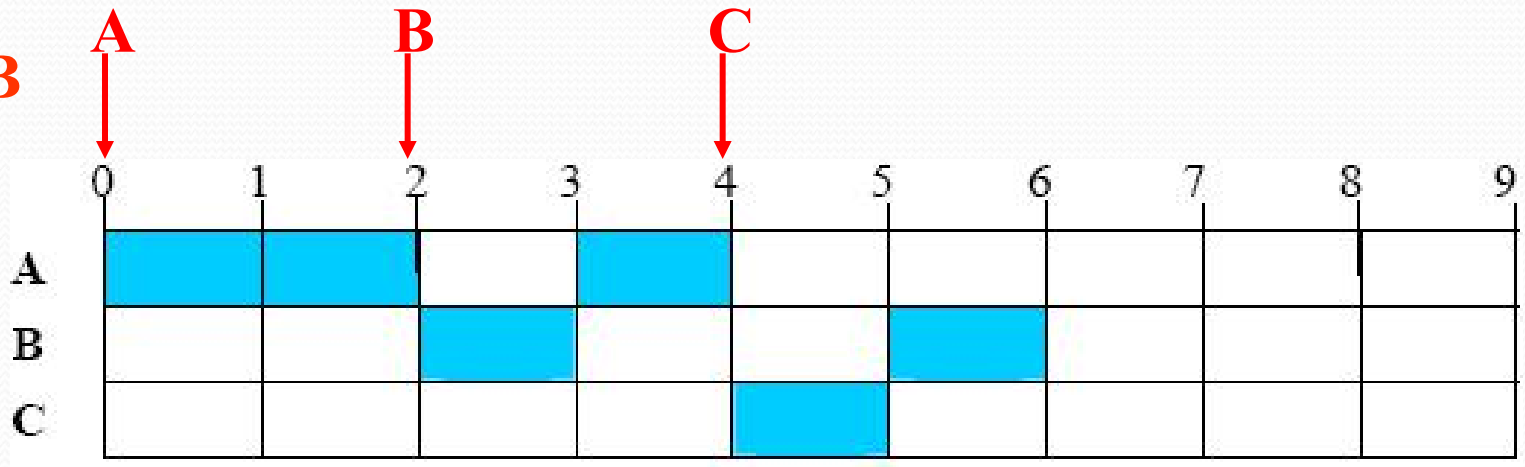
RQ0=

RQ1= C

RQ2=

Process	Arrival Time	Service Time
A	0	3
B	2	4
C	4	2

Running=B



Feedback (q=1)

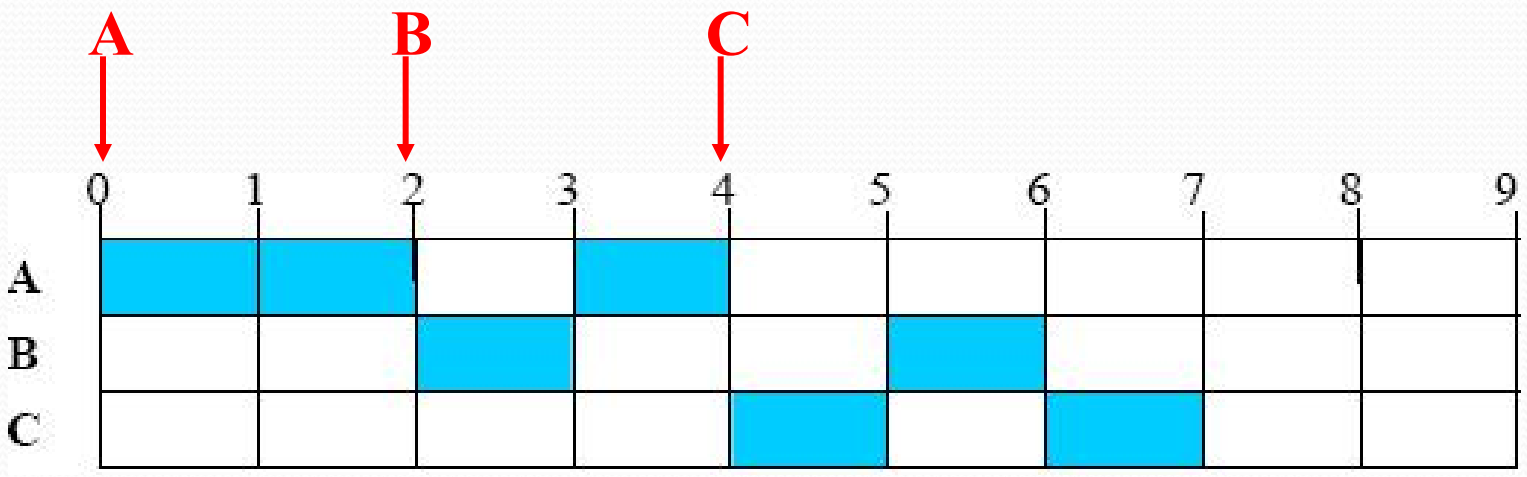
Time=6~7

RQ0=

RQ1=

RQ2= B

Process	Arrival Time	Service Time
A	0	3
B	2	4
C	4	2



Running=C and finished

Feedback (q=1)

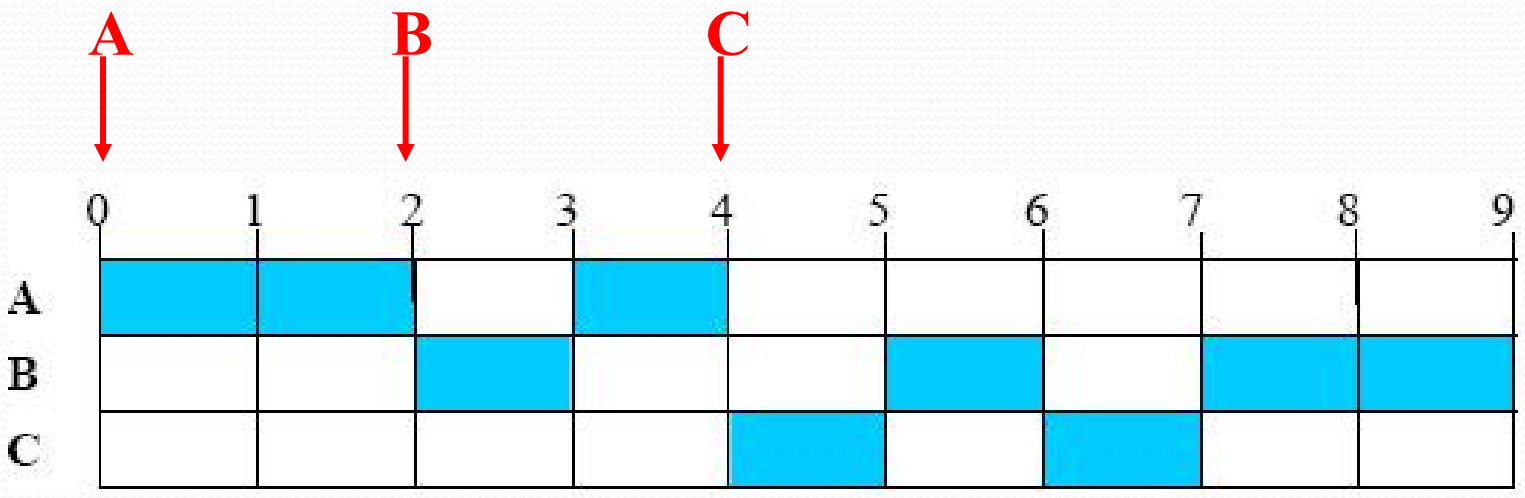
Time=7~9

RQ0=

RQ1=

RQ2=

Process	Arrival Time	Service Time
A	0	3
B	2	4
C	4	2



Running=B and finished

Feedback ($q=2^i$)

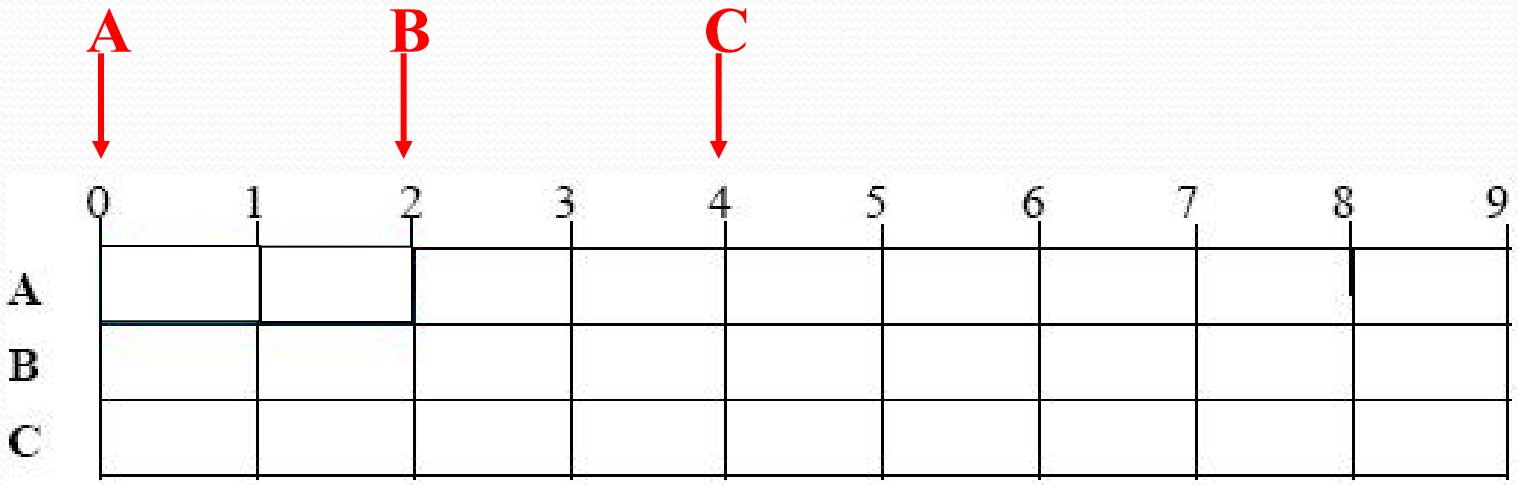
Time=0

RQ0= A

RQ1=

RQ2=

Process	Arrival Time	Service Time
A	0	3
B	2	4
C	4	2



- 当一个进程第一次进入系统时，它被放置在RQ0，当它第一次被抢占后并返回就绪状态时，它被放置在RQ1。在随后的时间里，每当它被抢占时，它被降级到下一个低优先级队列中。一个短进程很快会执行完，不会在就绪队列中降很多级。

Feedback ($q=2^i$)

Time=0~2

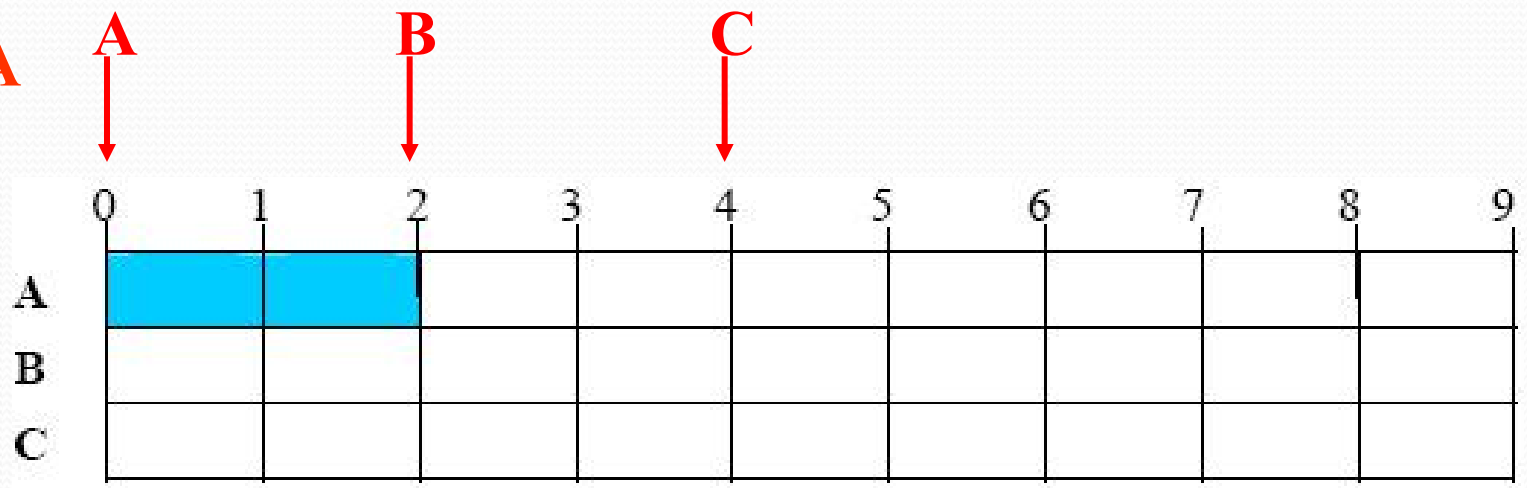
RQ0=

RQ1=

RQ2=

Process	Arrival Time	Service Time
A	0	3
B	2	4
C	4	2

Running=A



Feedback ($q=2^i$)

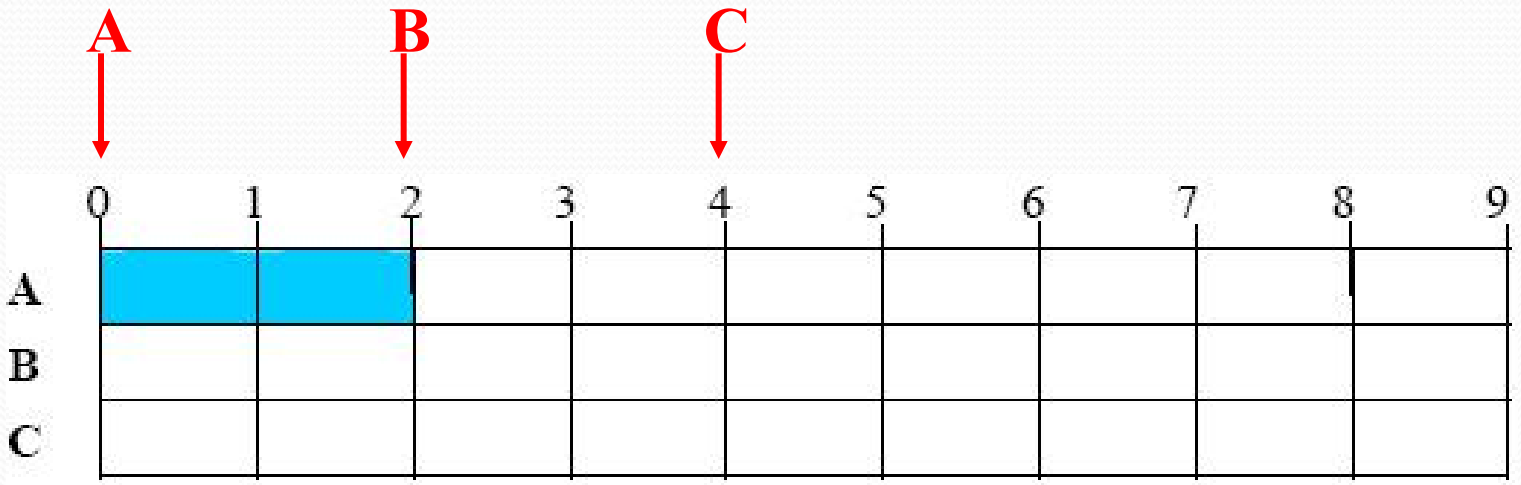
Time=2

RQ0= B

RQ1=

RQ2=

Process	Arrival Time	Service Time
A	0	3
B	2	4
C	4	2



Feedback ($q=2^i$)

Time=2~3

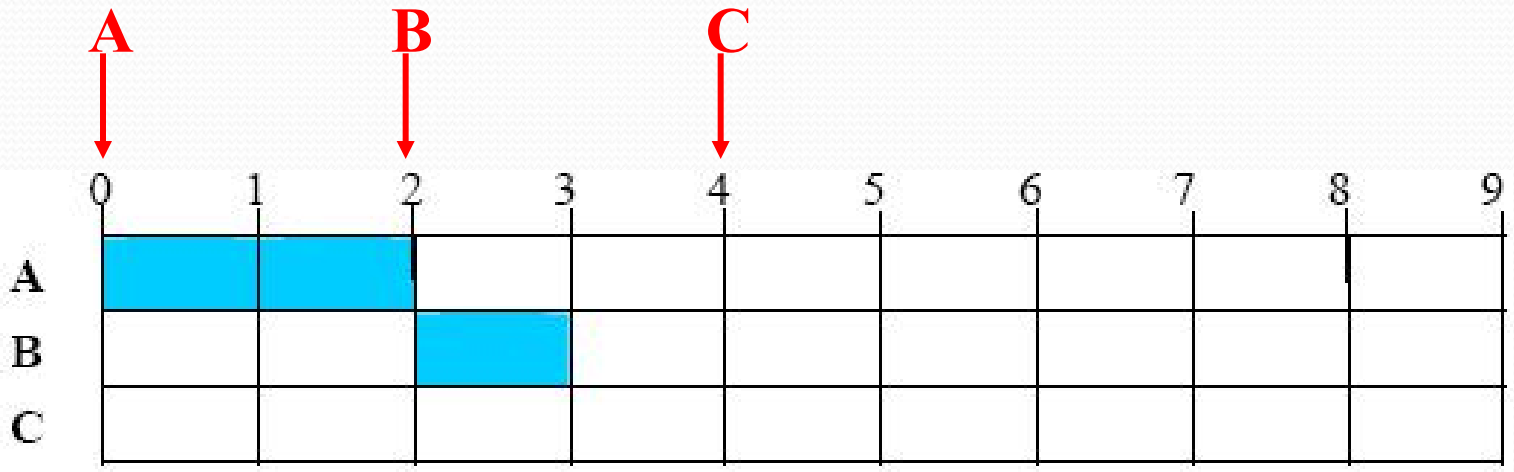
RQ0=

RQ1= A

RQ2=

Running=B

Process	Arrival Time	Service Time
A	0	3
B	2	4
C	4	2



Feedback ($q=2^i$)

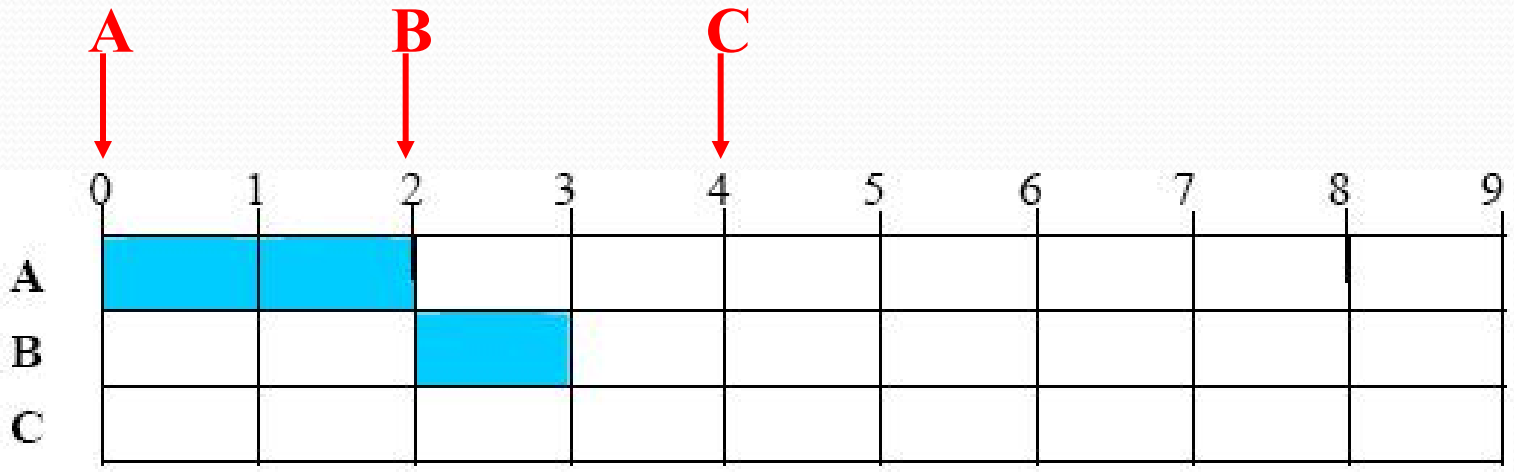
Time=3

RQ0=

RQ1= A

RQ2=

Process	Arrival Time	Service Time
A	0	3
B	2	4
C	4	2



Feedback ($q=2^i$)

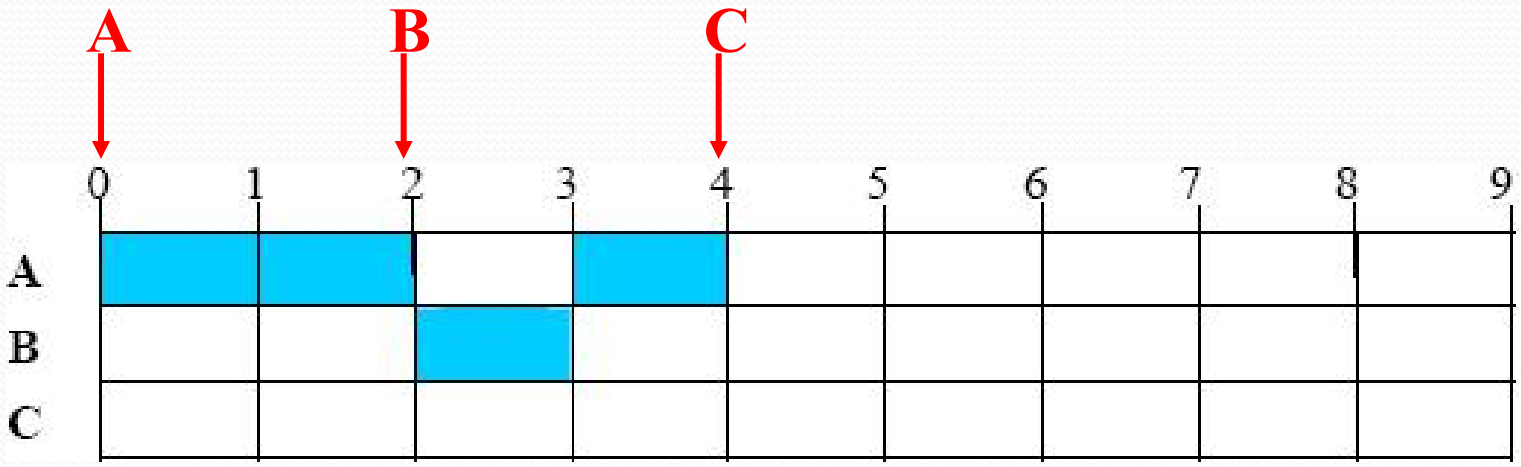
Time=3~4

RQ0=

RQ1= B

RQ2=

Process	Arrival Time	Service Time
A	0	3
B	2	4
C	4	2



Running=A and finished

Feedback ($q=2^i$)

Time=4

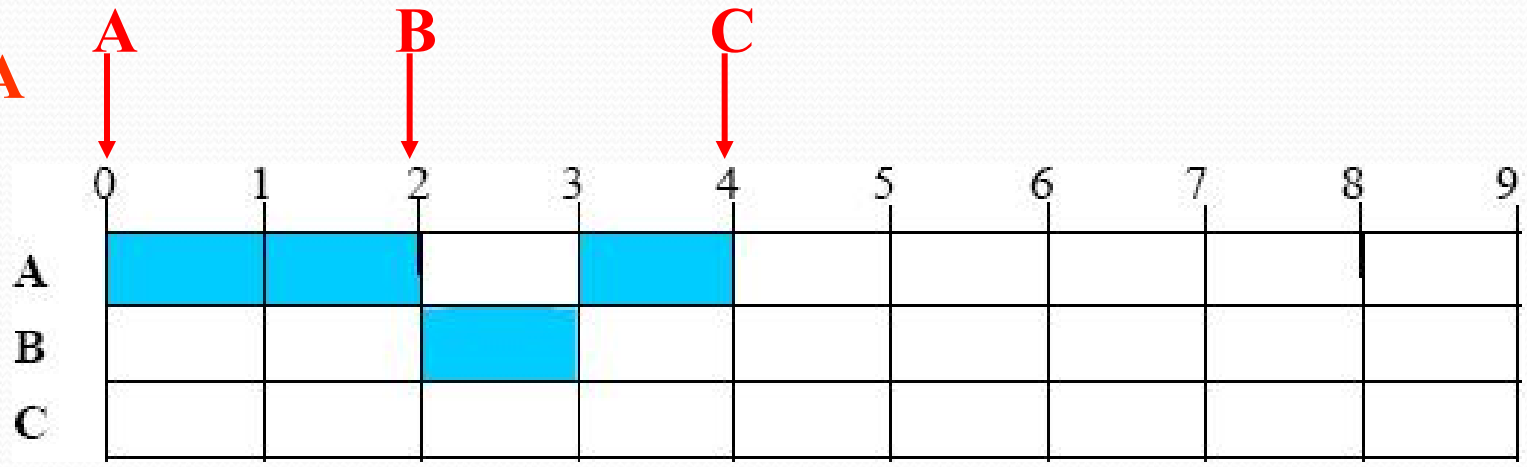
RQ0= C

RQ1= B

RQ2=

Process	Arrival Time	Service Time
A	0	3
B	2	4
C	4	2

Running=A



Feedback ($q=2^i$)

Time=4~5

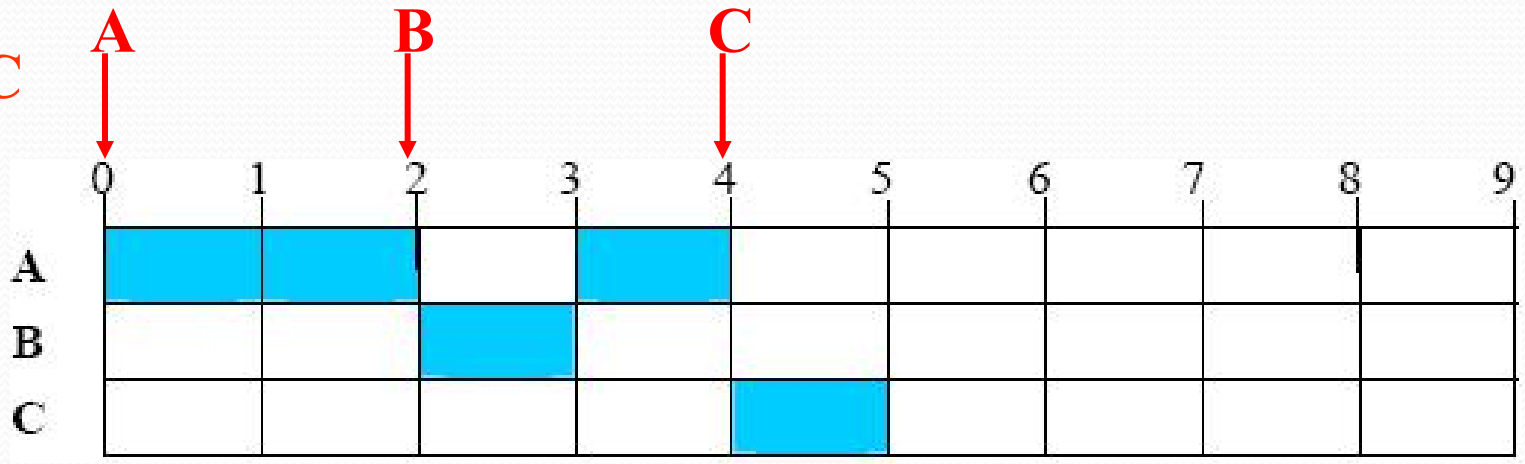
RQ0=

RQ1= B

RQ2=

Process	Arrival Time	Service Time
A	0	3
B	2	4
C	4	2

Running=C



Feedback ($q=2^i$)

Time=5~6

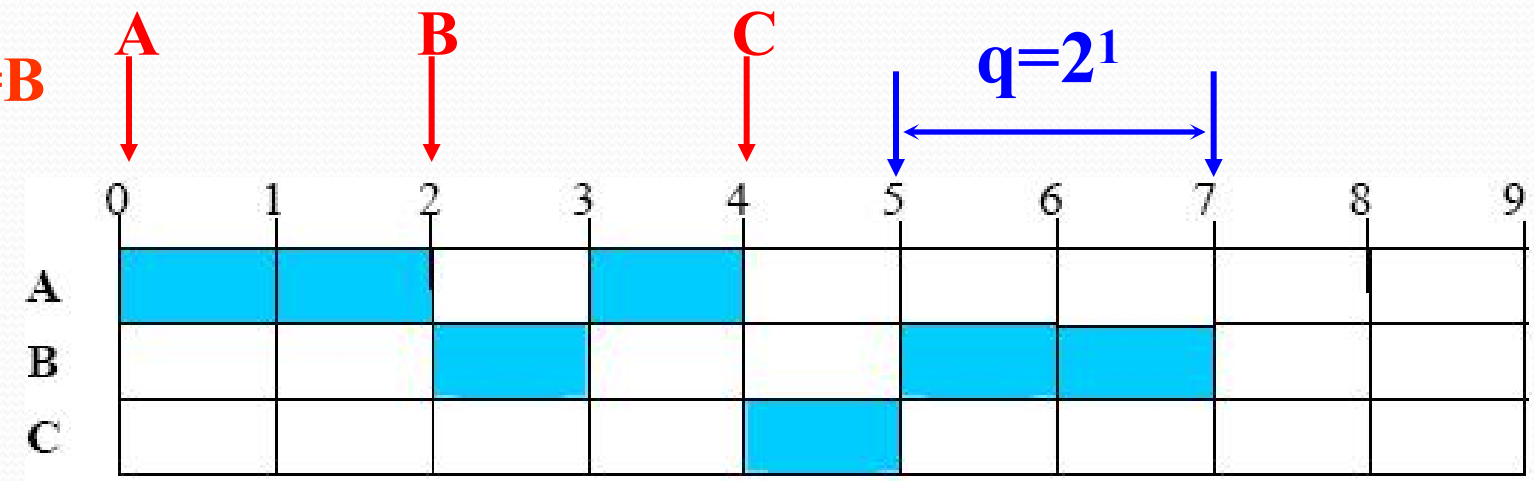
RQ0=

RQ1= C

RQ2=

Process	Arrival Time	Service Time
A	0	3
B	2	4
C	4	2

Running=B



Feedback ($q=2^i$)

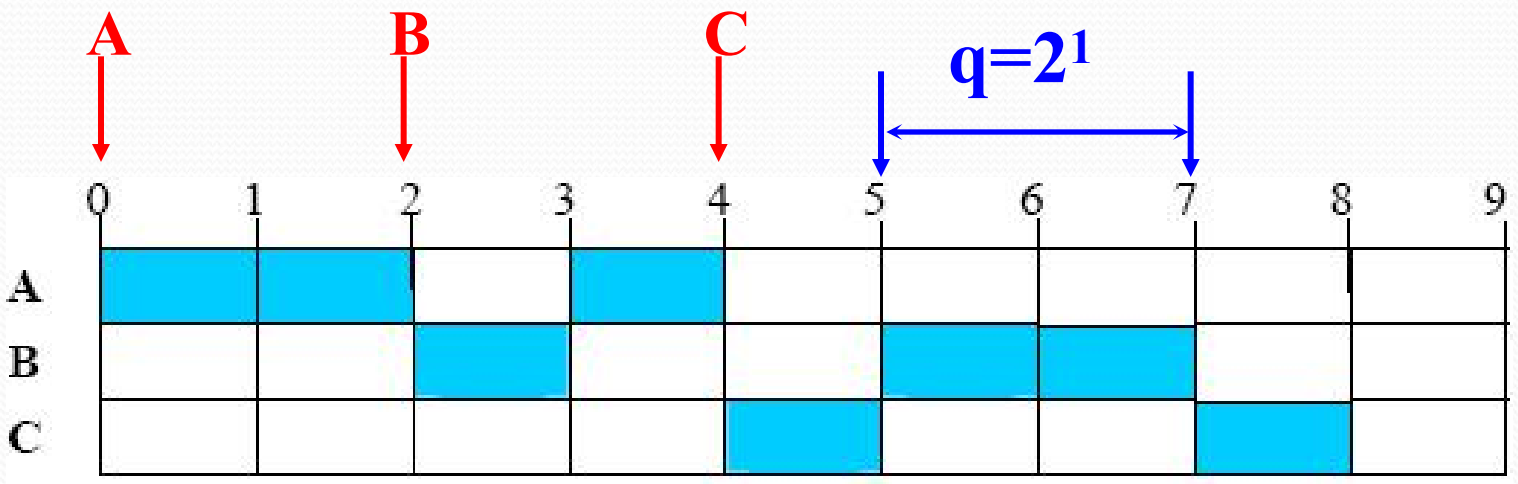
Time=6~7

RQ0=

RQ1=

RQ2= B

Process	Arrival Time	Service Time
A	0	3
B	2	4
C	4	2



Running=C and finished

Feedback ($q=2^i$)

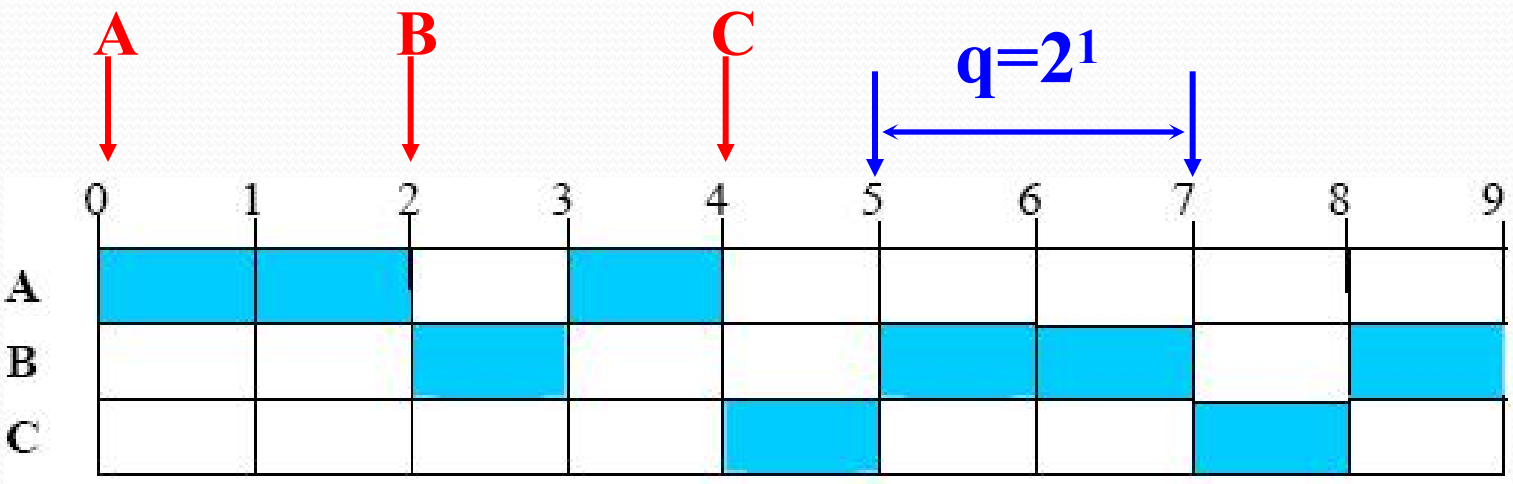
Time=7~9

RQ0=

RQ1=

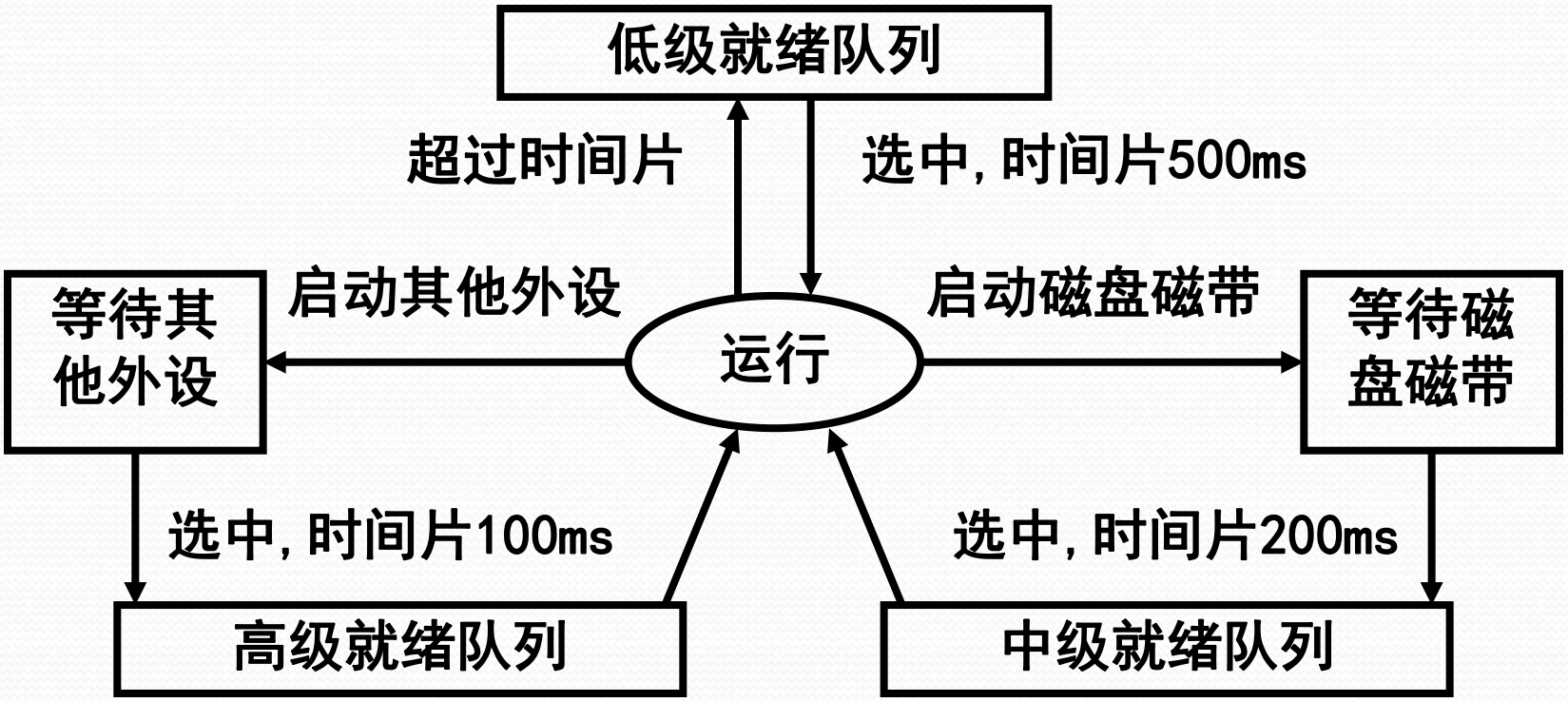
RQ2=

Process	Arrival Time	Service Time
A	0	3
B	2	4
C	4	2



Running=B and finished

Feedback


低级就绪队列
超过时间片
选中，时间片500ms
等待其他外设
启动其他外设
运行
启动磁盘磁带
等待磁盘磁带
选中，时间片100ms
选中，时间片200ms
高级就绪队列
中级就绪队列

传统Unix系统的调度(例)

- 多级反馈队列，每个优先级队列使用时间片轮转
- 每秒重新计算每个进程的优先级
- 给每个进程赋予基本优先级的目的是把所有进程划分成固定的优先级区
- 可控调节因子

Unix SVR4调度算法(例)

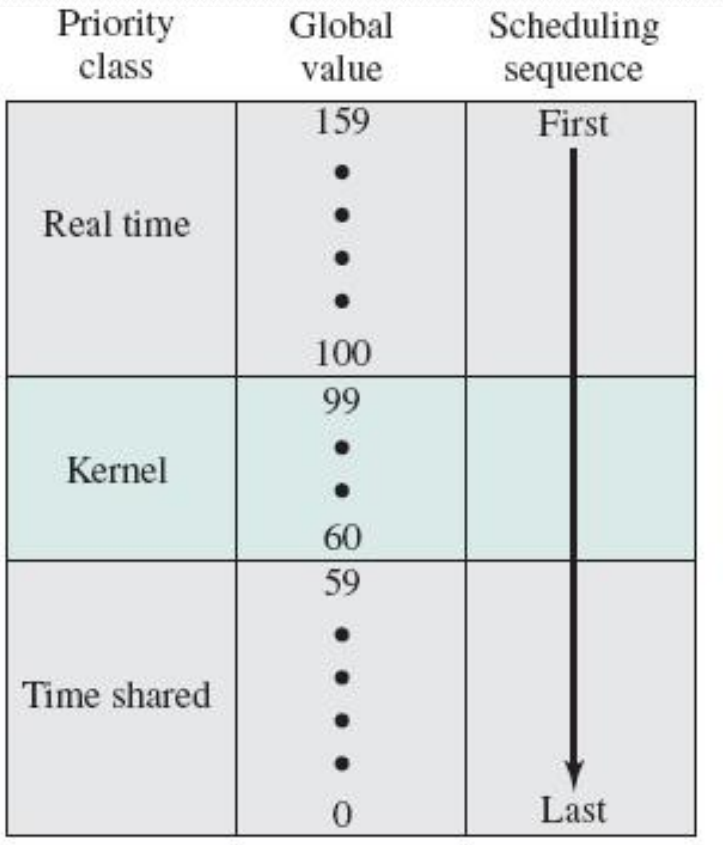
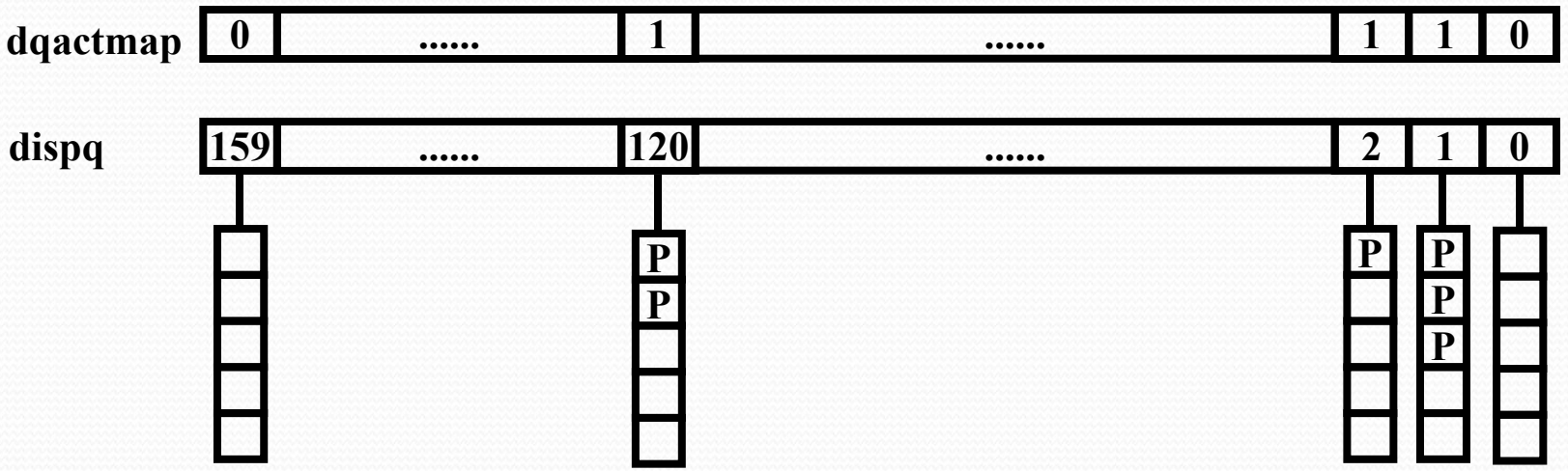


Figure 10.12 SVR4 Priority Classses

Unix SVR4调度算法(例)

- 多级反馈队列，每一个优先数都对应于一个就绪进程队列
- 实时优先级层次：优先数和时间片都是固定的，在抢占点执行抢占
- 分时优先级层次：优先数和时间片是可变的，从0优先数的100ms到59优先数的10ms



Bands

- 优先级递减
  - 对换
  - 块I/O设备控制
  - 文件操作
  - 字符I/O设备控制
  - 用户进程

Windows调度算法(例)

- 主要设计目标：基于内核级线程的可抢占式调度，向单个用户提供交互式的计算环境，并支持各种服务器程序
- 优先级和优先数
  - 实时优先级层次(优先数为31-16)：用于通信任务和实时任务，优先数不可变
  - 可变优先级层次(优先数为15-0)：用于用户提交的交互式任务，优先数可动态调整
- 多级反馈队列，每一个优先数都对应于一个就绪进程队列

Windows调度算法(例)

- 优先数可动态调整原则
  - 线程所属的进程对象有一个进程基本优先数，取值范围从0到15
  - 线程对象有一个线程基本优先数，取值范围从-2到2
  - 线程的初始优先数为进程基本优先数加上线程基本优先数，但必须在0到15的范围内
  - 线程的动态优先数必须在初始优先数到15的范围内
- 当存在N个处理器时，N-1个处理器上将运行N-1个最高优先级的线程，其他线程将共享剩下的一个处理器

彩票调度算法

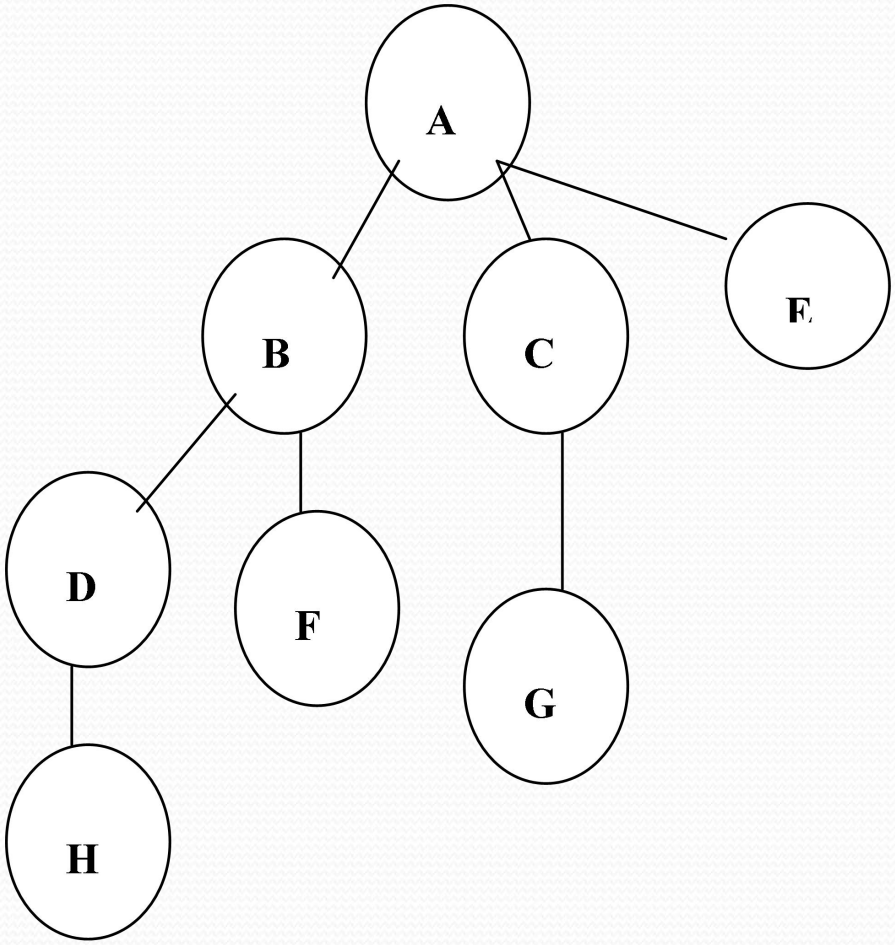
- 基本思想：为进程发放针对系统各种资源(如CPU时间)的彩票；当调度程序需要做出决策时，随机选择一张彩票，持有该彩票的进程将获得系统资源
- 合作进程之间的彩票交换

习题(进程管理的fork系统调用)

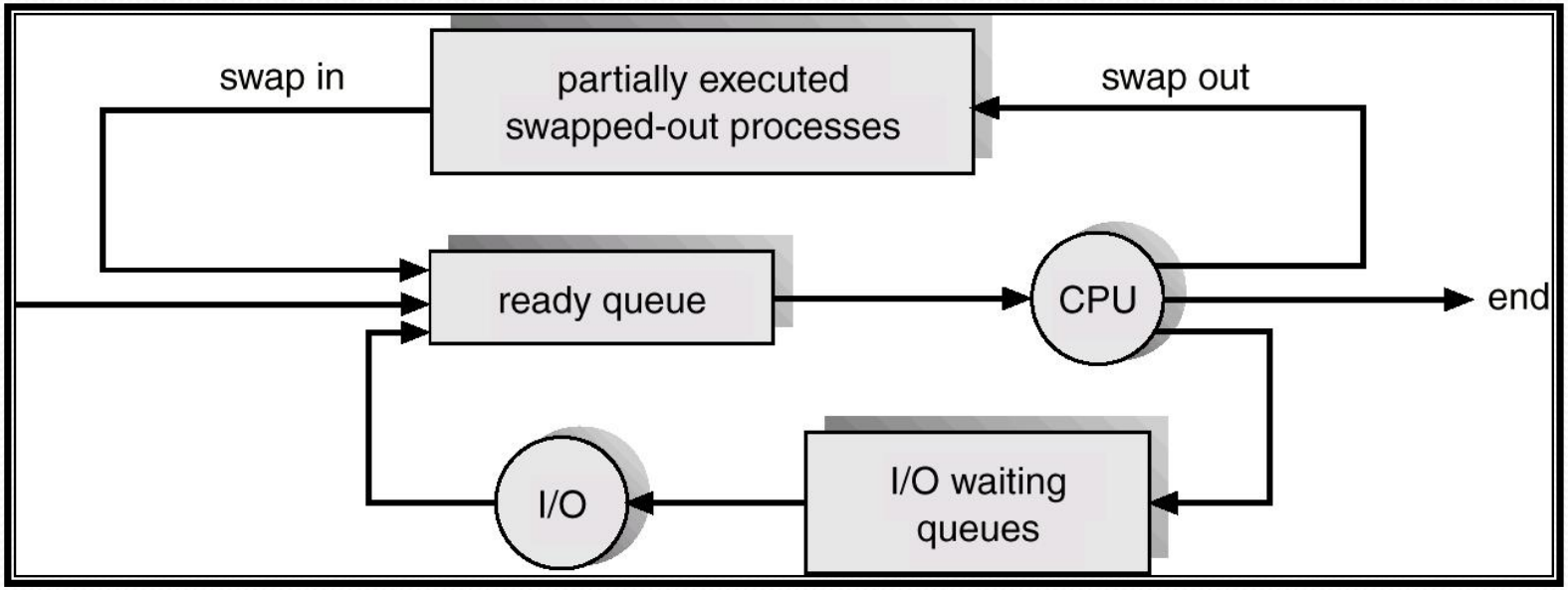
33. 在UNIX系统中运行以下程序，最多可产生出多少进程?画出进程家属树。
(《操作系统教程》第四版第2章)

```
main( ) {
  fork( );  /*←pc(程序计数器)，进程A*/
  fork( );
  fork( );
}
```

进程管理的fork系统调用



Addition of Medium Term Scheduling [补充]



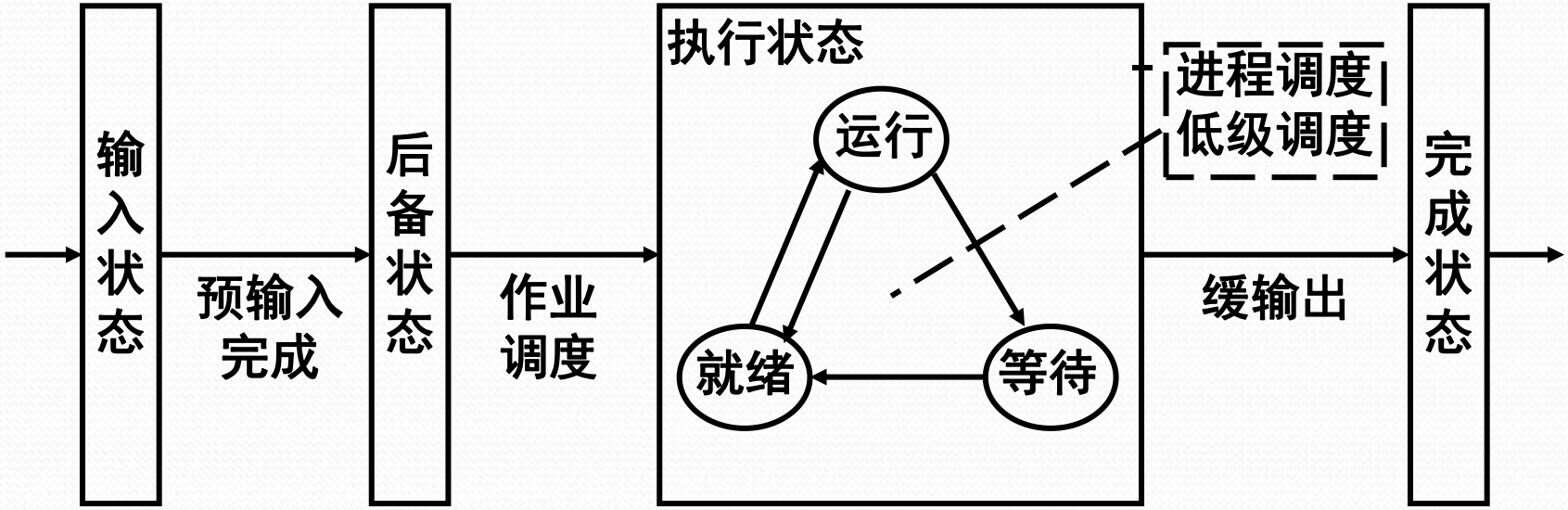
批处理作业的调度

批处理作业的管理

- 作业说明语言和作业说明书
- 脱机控制方式(批处理控制方式)
- 作业控制块JCB
- 作业状态
  - 输入状态：作业正在从输入设备上预输入信息
  - 后备状态：作业预输入结束但尚未被选中执行
  - 执行状态：作业已经被选中并构成进程去竞争处理器资源以获得运行
  - 完成状态：作业运行结束，正在等待缓输出

批处理作业的状态

作业调度与进程调度



批处理作业的调度

- 作业调度：按一定的策略选取若干个作业让它们进入内存、构成进程去竞争处理器以获得运行机会
- 用户立场：自己作业的周转时间尽可能的小
- 系统立场：希望进入系统的作业的平均周转时间尽可能的小
- 适当的作业调度算法必须既考虑用户的要求又有利于系统效率的提高