南京大学
NANJING UNIVERSITY

计算机与操作系统

第六讲 处理器调度

南京大学软件学院

本主题教学目标

1. 掌握调度的层次
2. 掌握进程的挂起态
3. 掌握低级调度及其策略
4. 掌握作业调度及其策略

南京大学
NANJING UNIVERSITY

第六讲 处理器调度

6.1 处理器调度的层次

6.2 进程的挂起态

6.3 调度算法

6.4 批处理作业的调度

6.1 处理器调度的层次

南京大学
NANJING UNIVERSITY

调度的目标

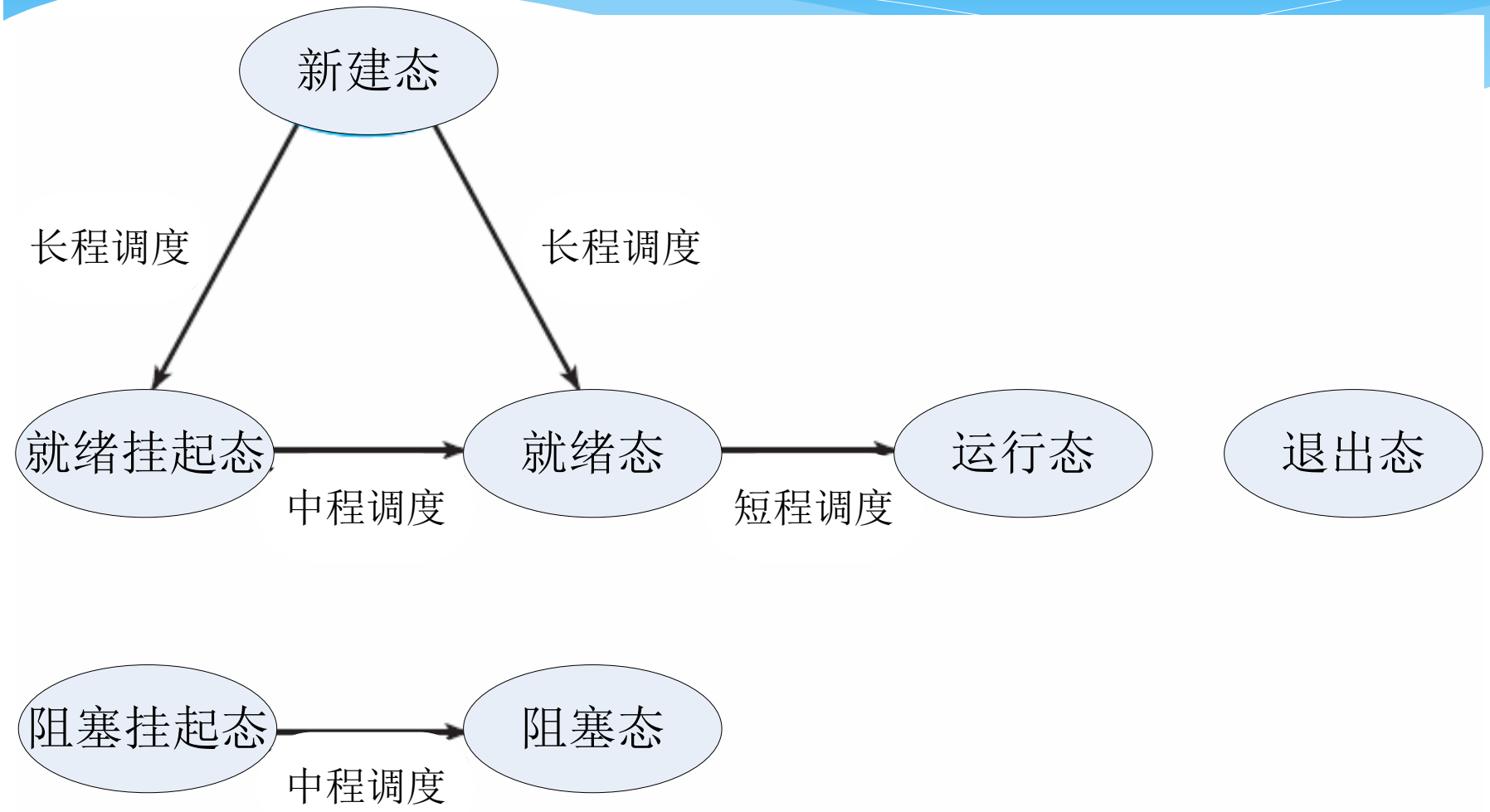
* 响应时间
* 吞吐量
* 处理器利用率

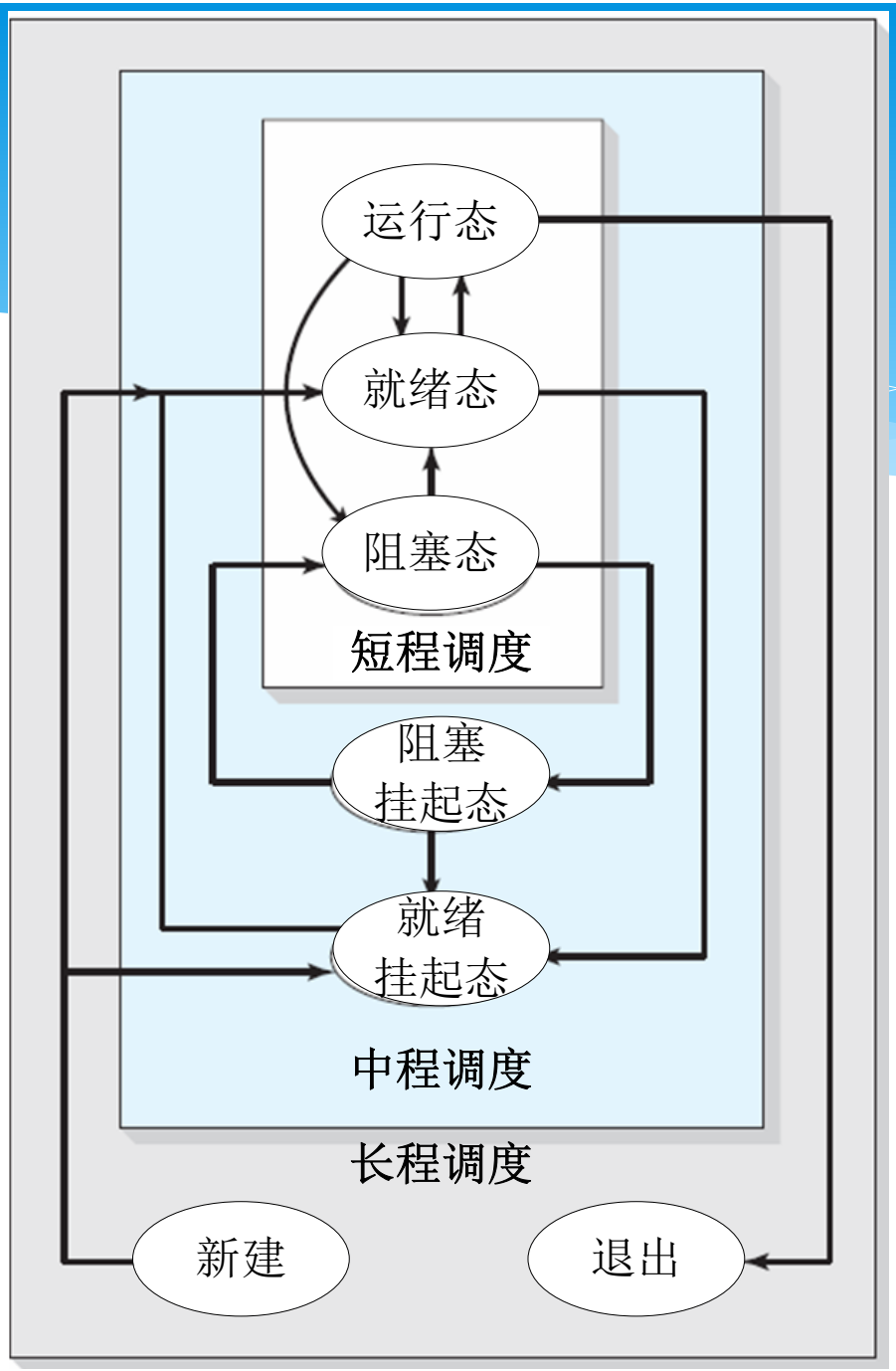
南京大学
NANJING UNIVERSITY

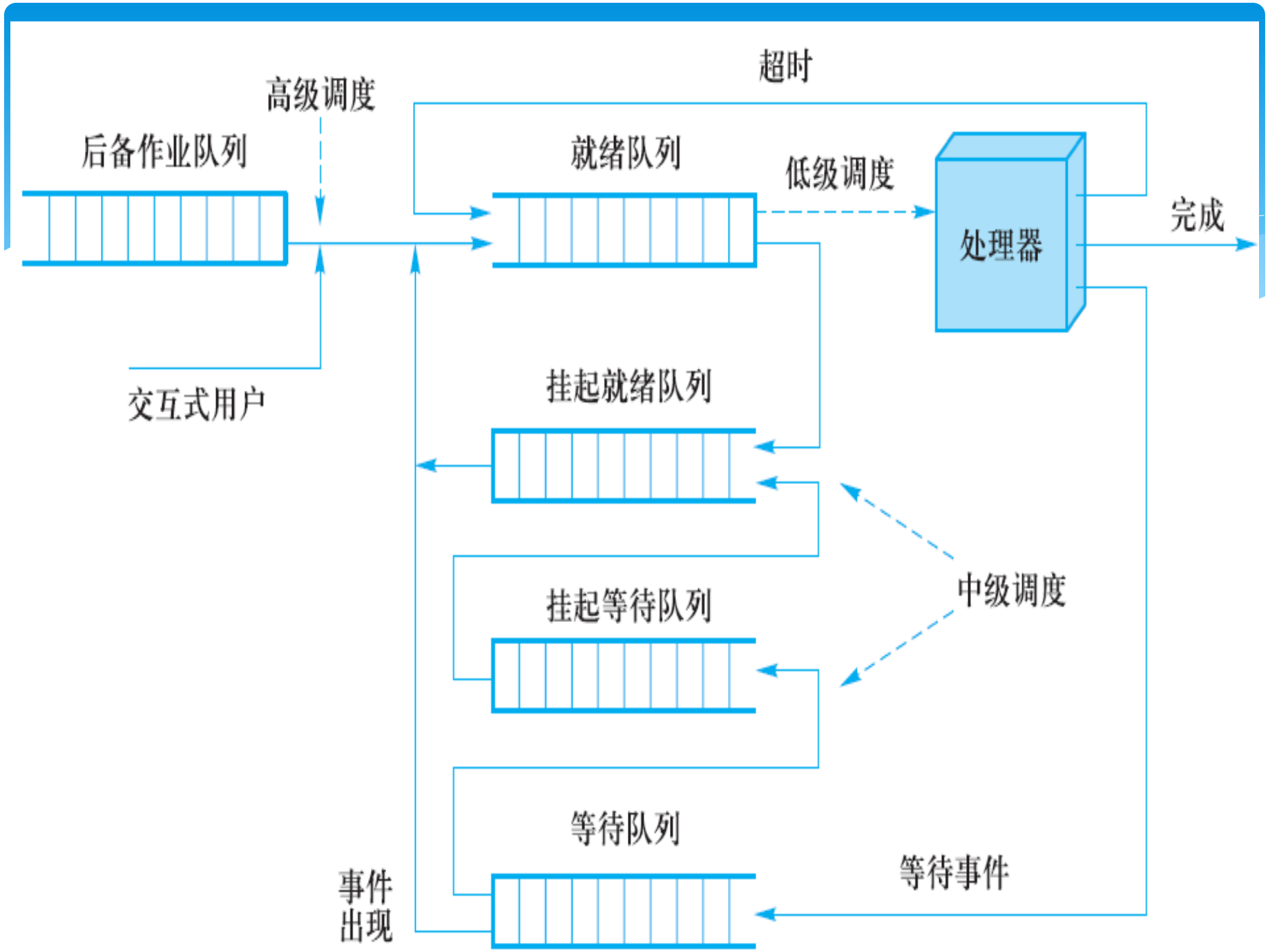
处理器调度的层次

* 长程调度(高级调度)
  * 决定加入到执行的进程池中
* 中程调度(中级调度)
  * 决定加入到部分或者全部在主存中的进程集合中
* 短程调度(低级调度)
  * 决定哪个可用进程将占用处理器执行
* I/O调度
  * 决定哪个进程挂起的I/O请求将被可用的I/O设备处理

处理器调度的层次




运行态
就绪态
阻塞态
短程调度
阻塞
挂起态
就绪
挂起态
中程调度
长程调度
新建
退出


超时
高级调度
后备作业队列
就绪队列
低级调度
处理器
完成
交互式用户
挂起就绪队列
中级调度
挂起等待队列
等待队列
等待事件
事件
出现

南京大学
NANJING UNIVERSITY

长程调度

* 决定哪个程序可以进入系统中被处理
* 控制多道程序设计的道数
* 创建的进程越多，每个进程可以执行的时间百分比就越少

中程调度

* 是交换(swapping)功能的一部分
* 根据需要管理多道程序设计的道数

南京大学
NANJING UNIVERSITY

短程调度

* 有时又称为分派器 (dispatcher)
* 执行得很频繁，精确地决定下一次执行哪个进程
* 引起短程调度的事件包括
  * 时钟中断
  * I/O中断
  * 操作系统调用
  * 信号

6.2 进程的挂起态

进程的挂起态

* 随着不断创建新的进程，系统的资源已经不能满足进程运行的要求，这个时候就必须把某些进程挂起(suspend)，对换到磁盘镜像区中，暂时不参与进程调度，起到平滑系统操作负荷的目的

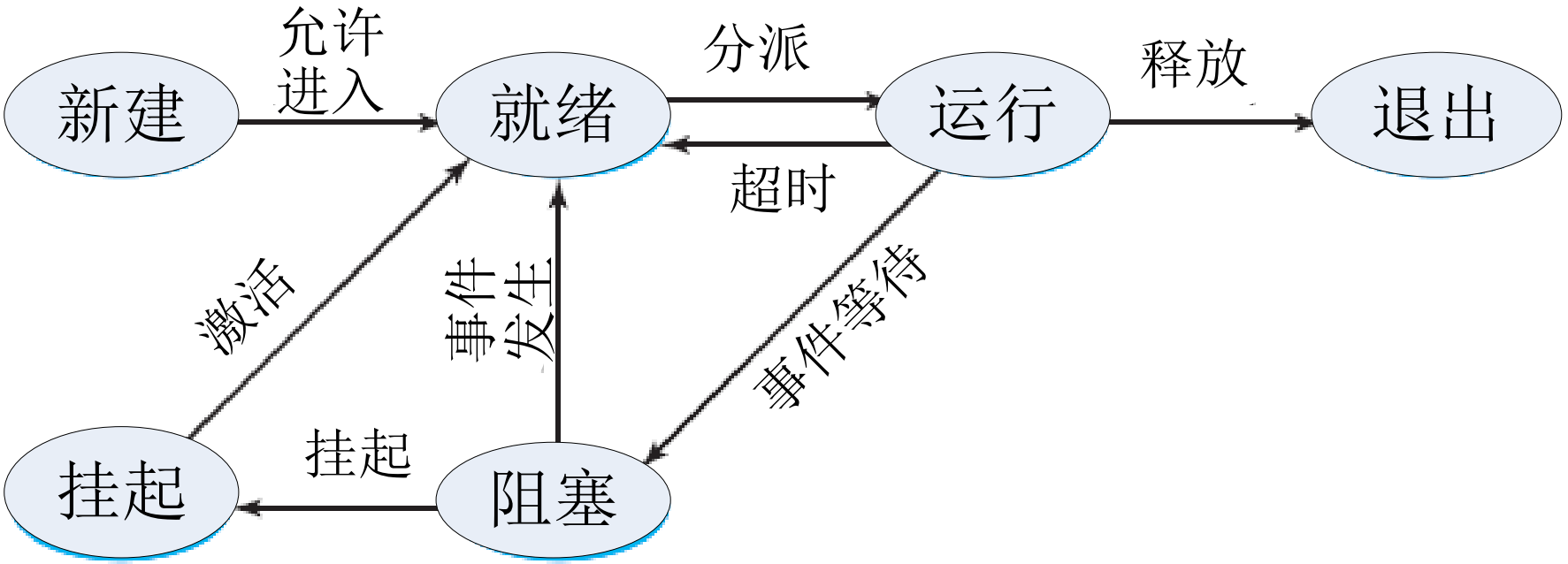
进程的挂起态

* 处理器比I/O快还能多，所以进程经常需要等待
* 将这些等待I/O的进程对换到磁盘，以空出更多主存
* 阻塞态进程对换到磁盘将变为挂起态
* 两种挂起态
  * 阻塞挂起态
  * 就绪挂起态

进程挂起的原因

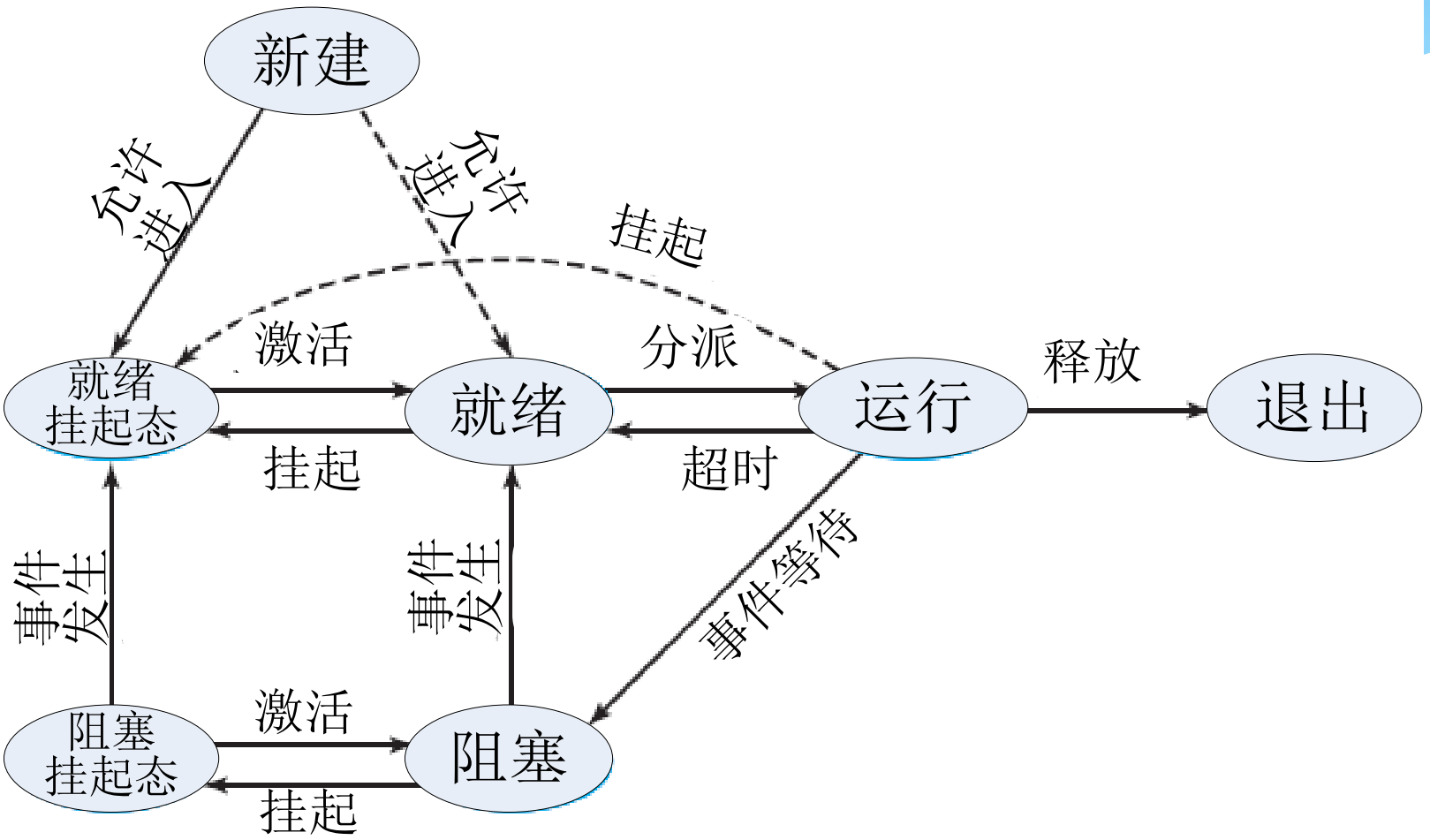
事件	说明
交换	操作系统需要释放足够的主存空间，以调入并执行处于就绪态的进程
其他OS原因	操作系统可能挂起后台进程或工具程序进程，或者被怀疑导致问题的进程
交互式用户请求	用户可能希望挂起一个程序的执行，目的是为了调试或者为了使用某个资源而建立连接
定时	一个进程可能会周期性地执行(例如审计或系统监视进程)，而且可能在等待下一个时间间隔时被挂起
父进程请求	父进程可能会希望挂起后代进程的执行，以检查或修改挂起的进程，或者协调不同后代进程之间的行为

含一个挂起态



南京大学
NANJING UNIVERSITY

含两个挂起态



南京大学
NANJING UNIVERSITY

进程的挂起态

* 一个挂起进程等同于不在主存的进程，它将不参与进程调度直到它们被对换进主存。它具有如下特征:
  * 该进程不能立即被执行
  * 挂起进程可能会等待一个事件，但所等待的事件是独立于挂起条件的，事件结束并不能导致进程具备执行条件
  * 进程进入挂起状态是由于操作系统、父进程或进程本身阻止它的运行
  * 结束进程挂起状态的命令只能通过操作系统或父进程发出

进程的挂起态

* 等待态→挂起等待态：如果当前不存在就绪进程，那么至少有一个等待态进程将被对换出去成为挂起等待态；操作系统根据当前资源状况和性能要求，也可以决定把等待态进程将被对换出去成为挂起等待态
* 挂起等待态→挂起就绪态：引起进程等待的事件发生之后，相应的挂起等待态进程将转换为挂起就绪态
* 就绪态→挂起就绪态：操作系统根据当前资源状况和性能要求，也可以决定把就绪态进程将被对换出去成为挂起就绪态

进程的挂起态

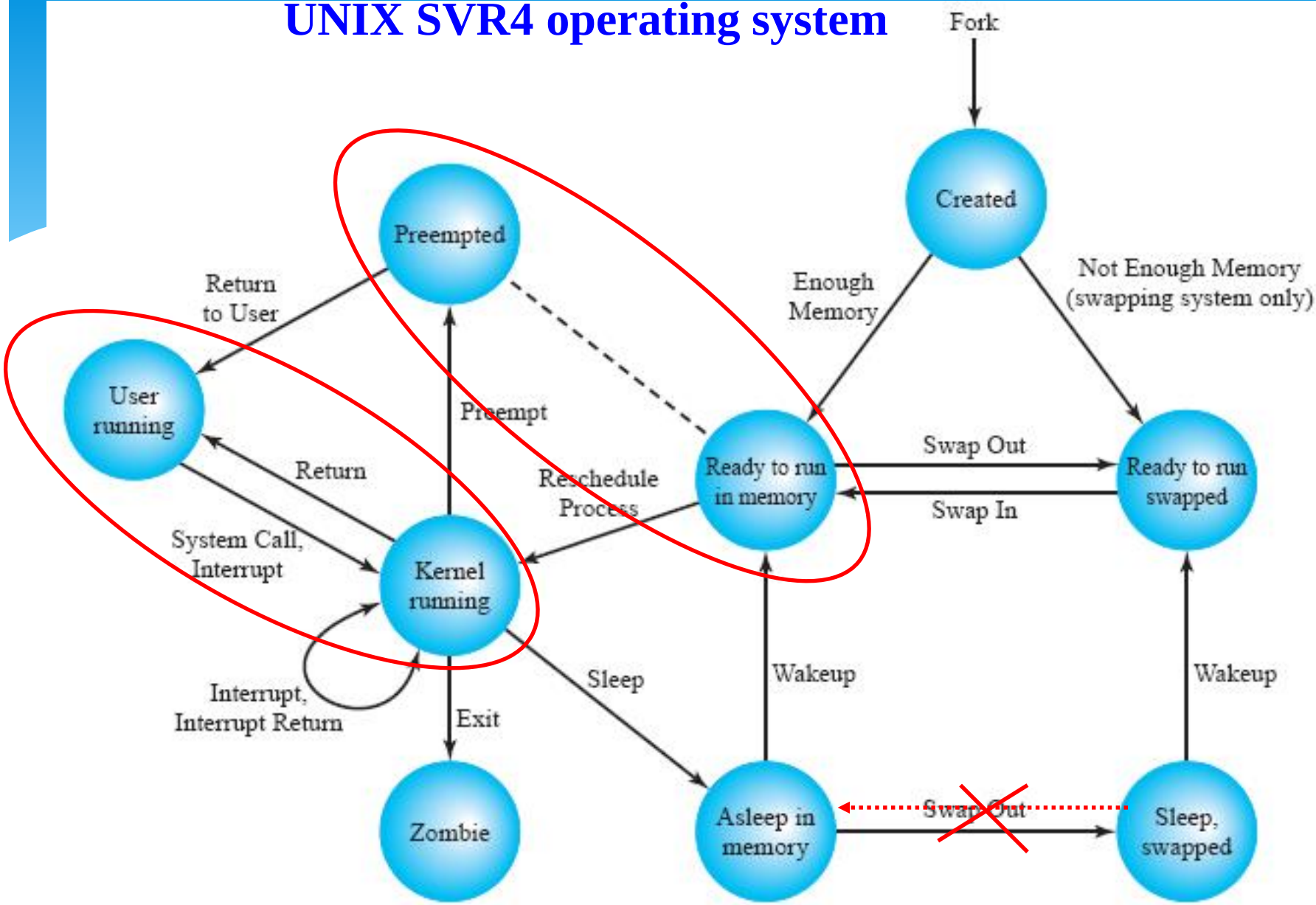
* 运行态→挂起就绪态：当一个具有较高优先级的挂起等待态进程的等待事件结束后，它需要抢占了CPU，而此时主存空间有不够，从而可能导致正在运行的进程转化为挂起就绪态。另外处于运行态的进程也可以自己挂起自己
* 新建态→挂起就绪态：考虑到系统当前资源状况和性能要求，可以决定新建的进程将被对换出去成为挂起就绪态

南京大学
NANJING UNIVERSITY

进程的挂起态

* 挂起就绪态→就绪态：当内存中没有就绪态进程，或者挂起就绪态进程具有比就绪态进程更高的优先级，系统将把挂起就绪态进程转换成就绪态
* 挂起等待态→等待态：当一个进程等待一个事件时，原则上时不需要把它调入内存的。但是在下面一种情况下，这一状态变化是可能的。当一个进程退出后，主存已经有了一大块自由空间，而某个挂起等待态进程具有较高的优先级并且操作系统已经得知导致它阻塞的事件即将结束，此时便发生了这一状态变化

UNIX SVR4 operating system



6.3 调度算法

6.3 调度算法

* 6.3.1 短程调度准则
* 6.3.2 优先级调度
* 6.3.3 调度的模式
* 6.3.4 调度算法

南京大学
NANJING UNIVERSITY

6.3.1 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

6.3.1 短程调度准则

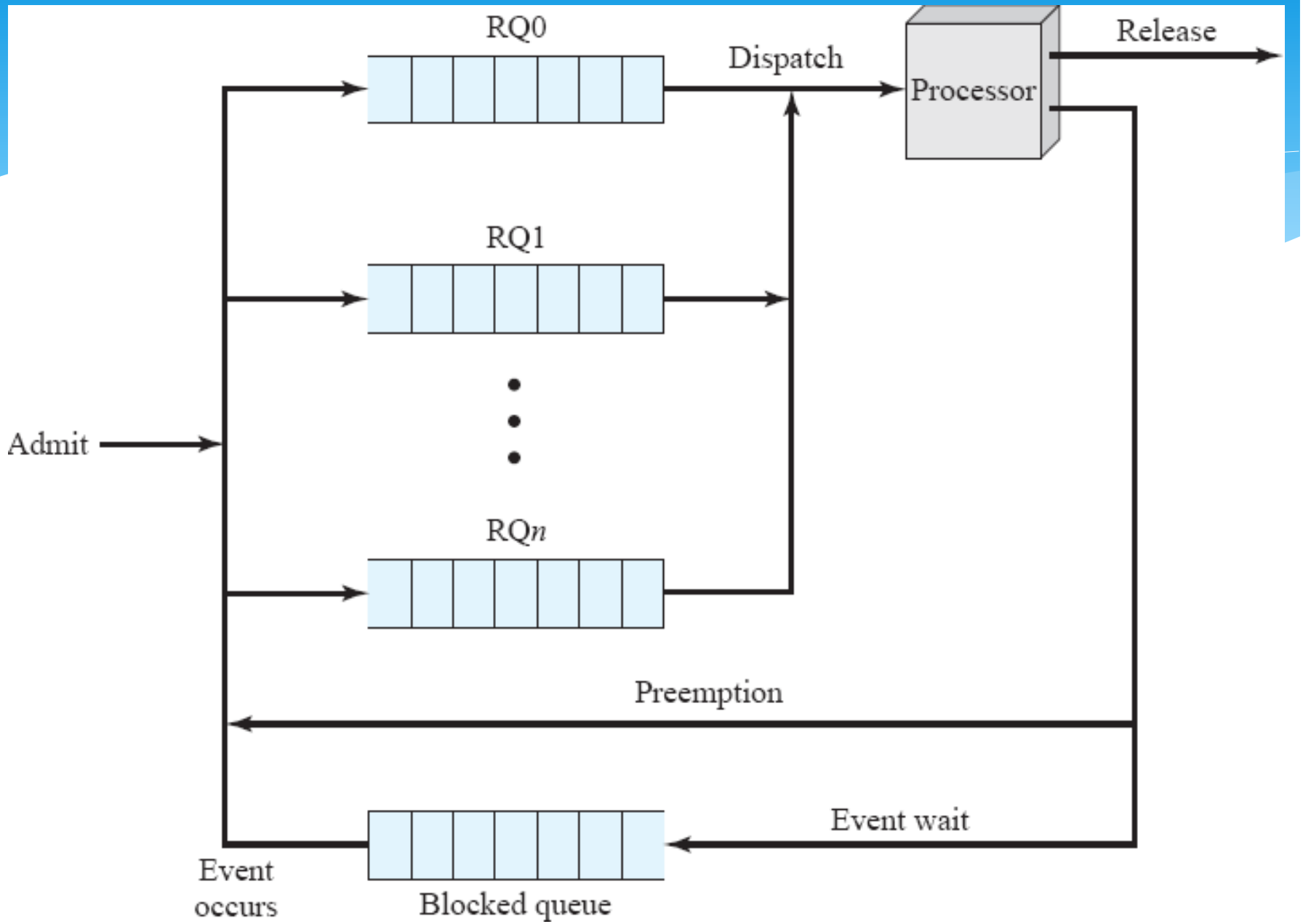
与性能无关

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

南京大学
NANJING UNIVERSITY

6.3.2 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

6.3.3 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

6.3.3 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

6.3.4 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

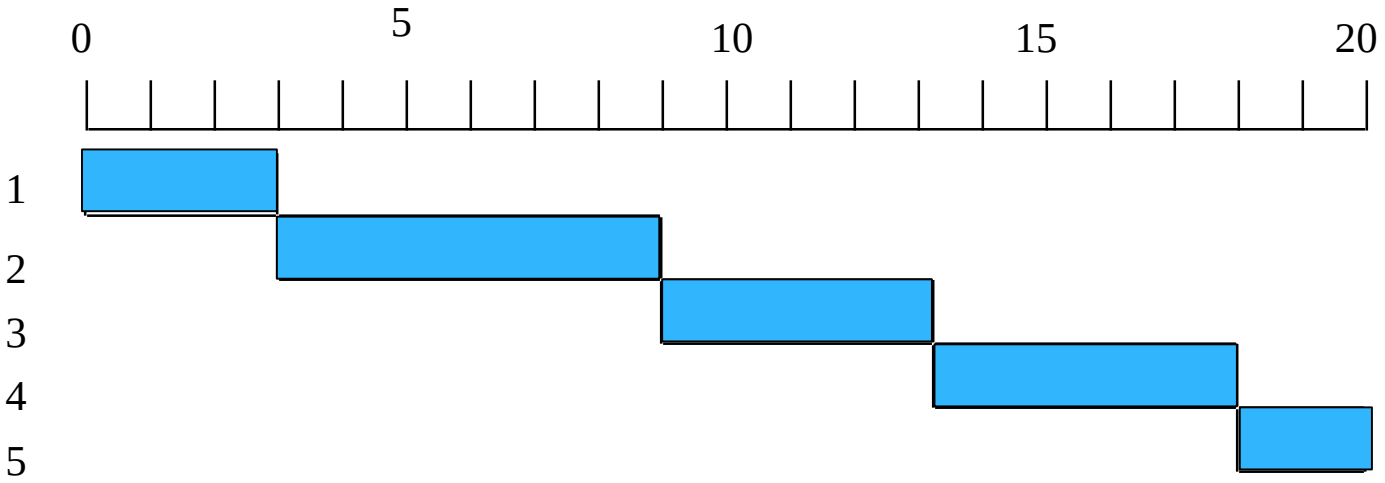
南京大学
NANJING UNIVERSITY

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

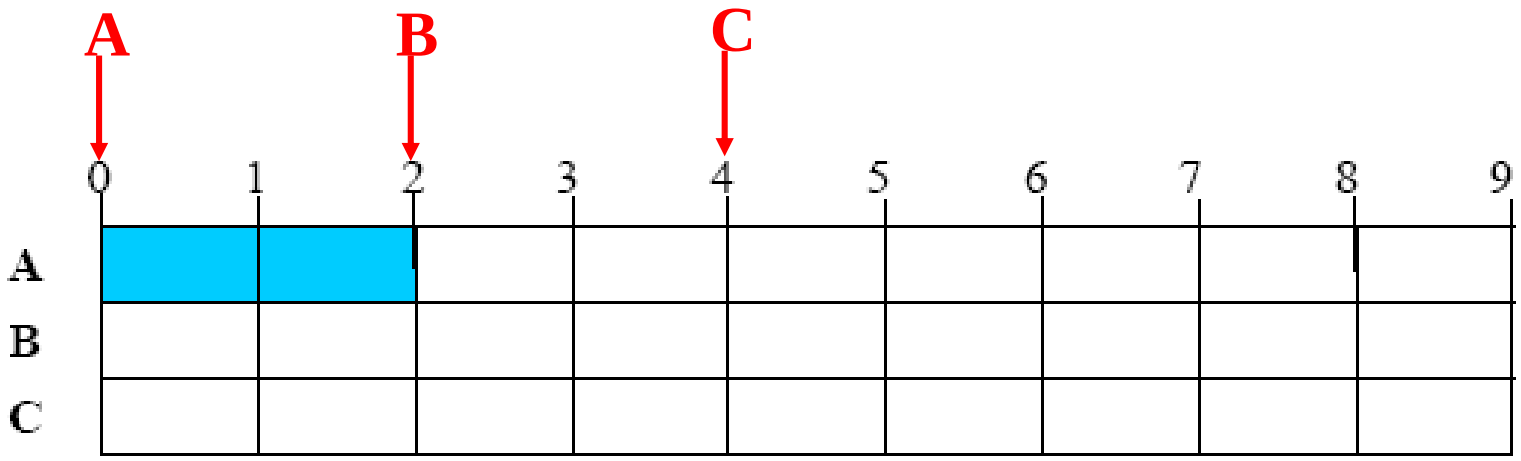
Time=0~2

Q= 空

队列

Running=A

B在2ms时刻到达



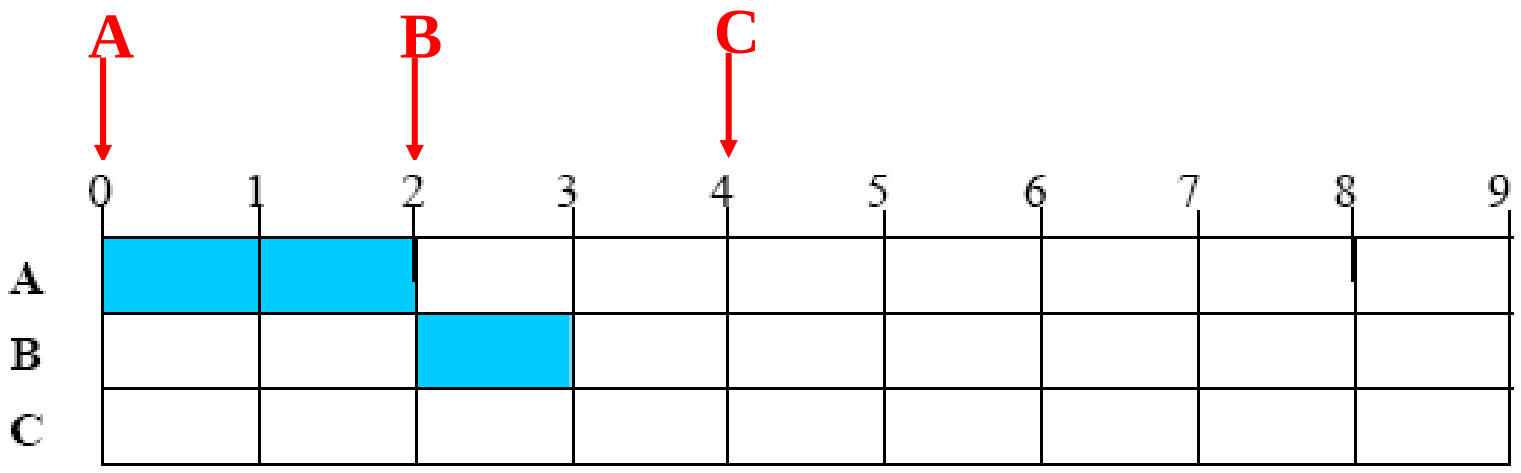
RR（时间片轮转）

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

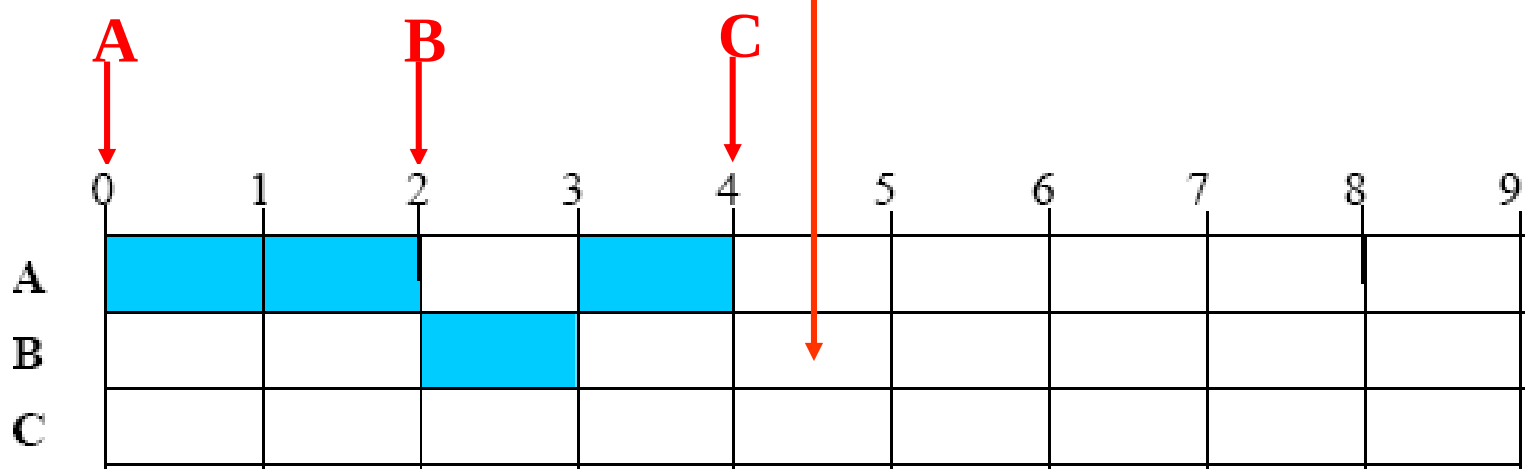
Time=3~4

Q=B

Time=4

Running=A and finished

Q=BC



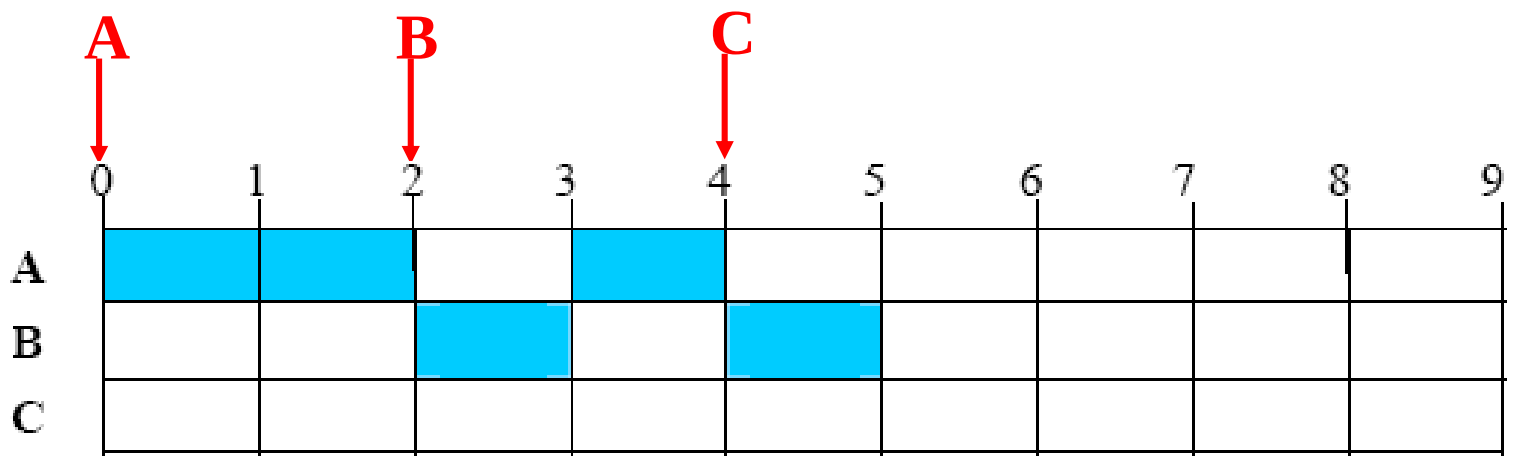
RR（时间片轮转）

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

Time=4~5

Q=C

Running=B



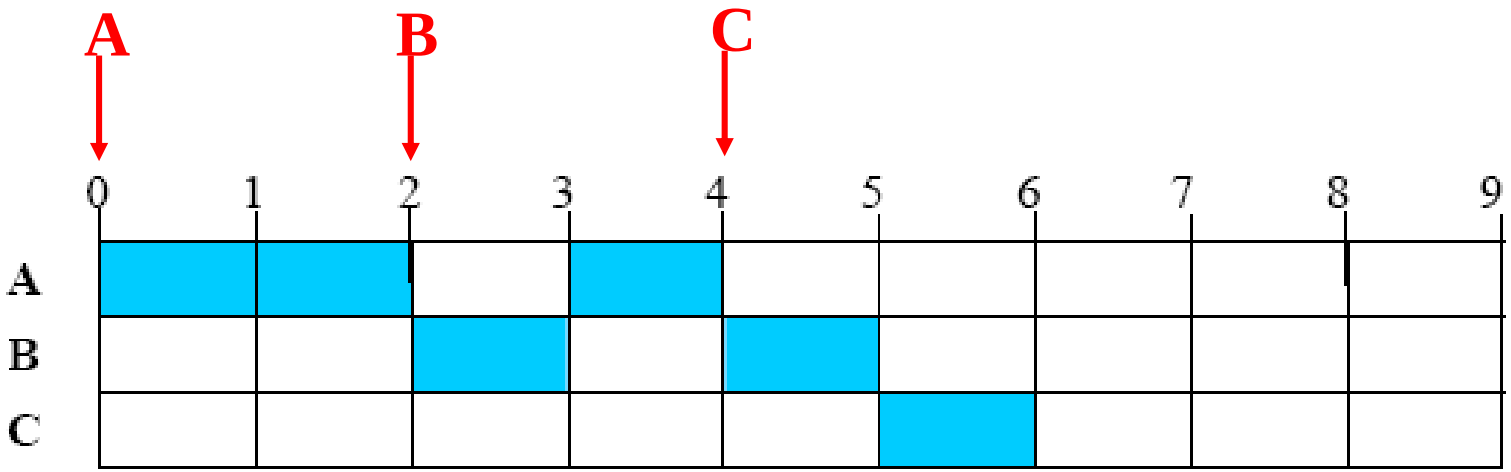
RR（时间片轮转）

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

Time = 5~6

Q=B

Running=C



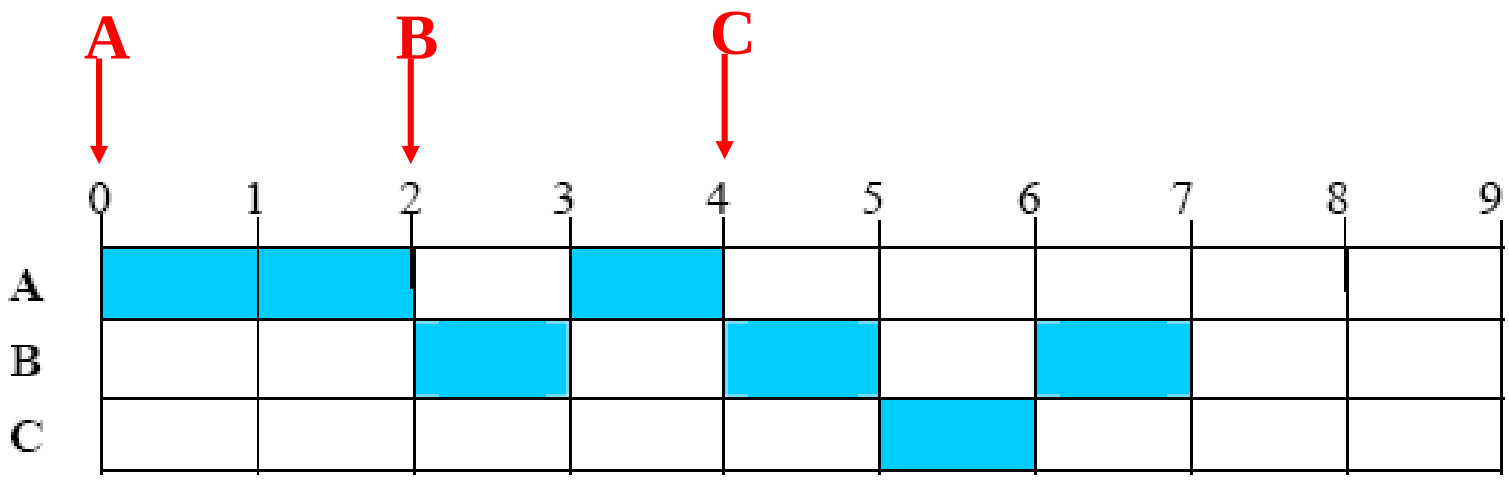
RR (时间片轮转)

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

Time = 6~7

Q=C

Running=B



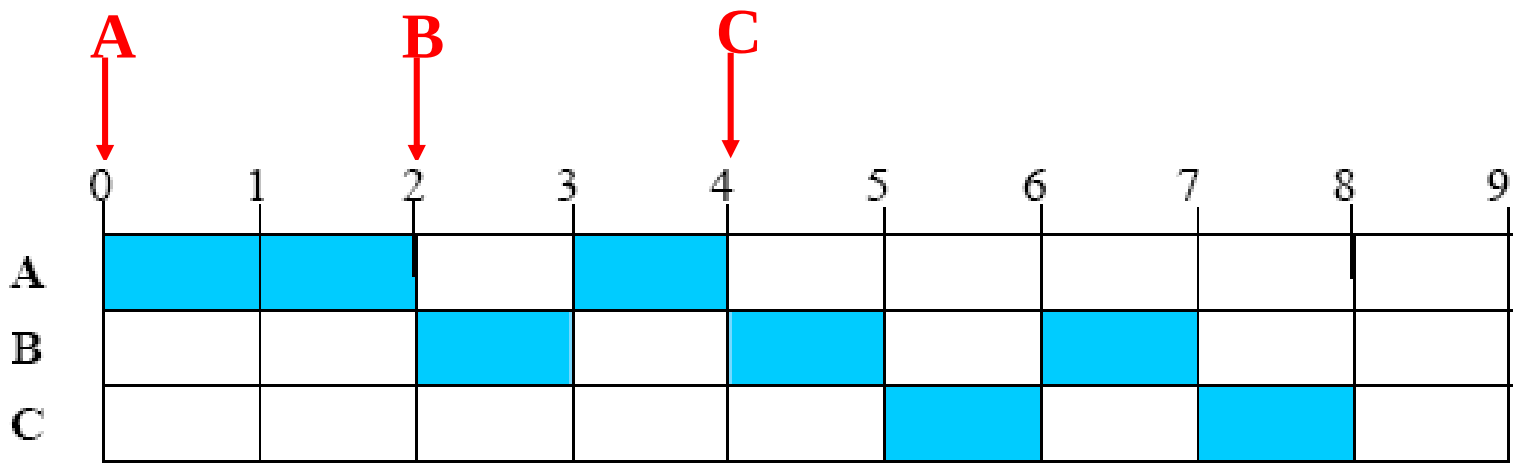
RR (时间片轮转)

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

Time = 7~8

Q=B

Running=C and finished

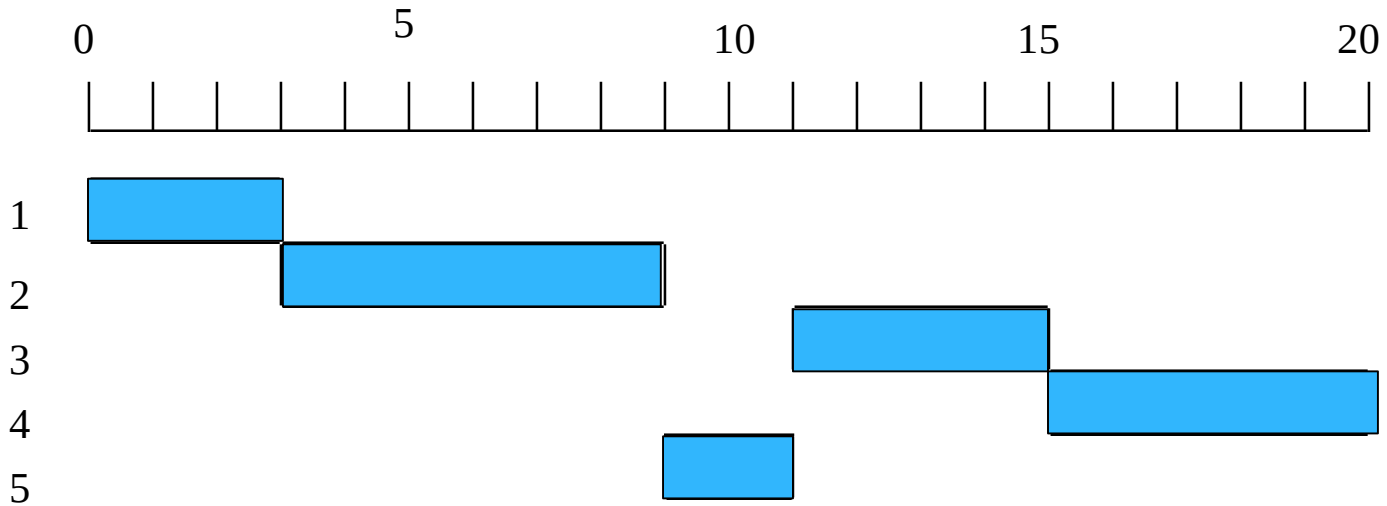


SPN (最短进程优先)

* 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)

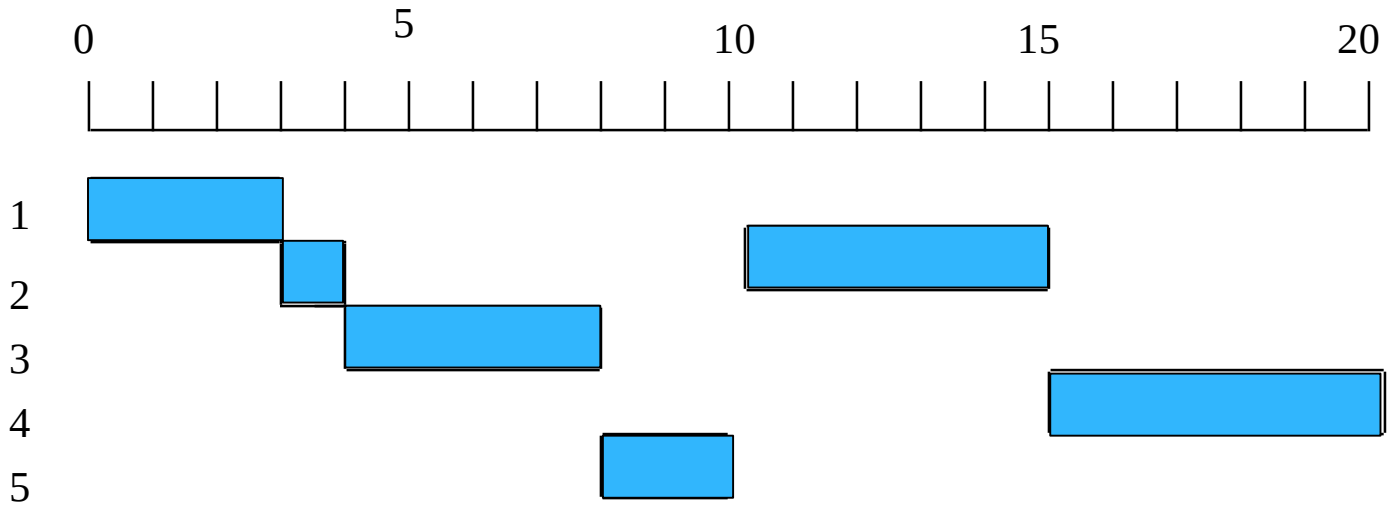
* 问题
  * 只要持续不断地提供更短的进程，长进程就有可能饿死。

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



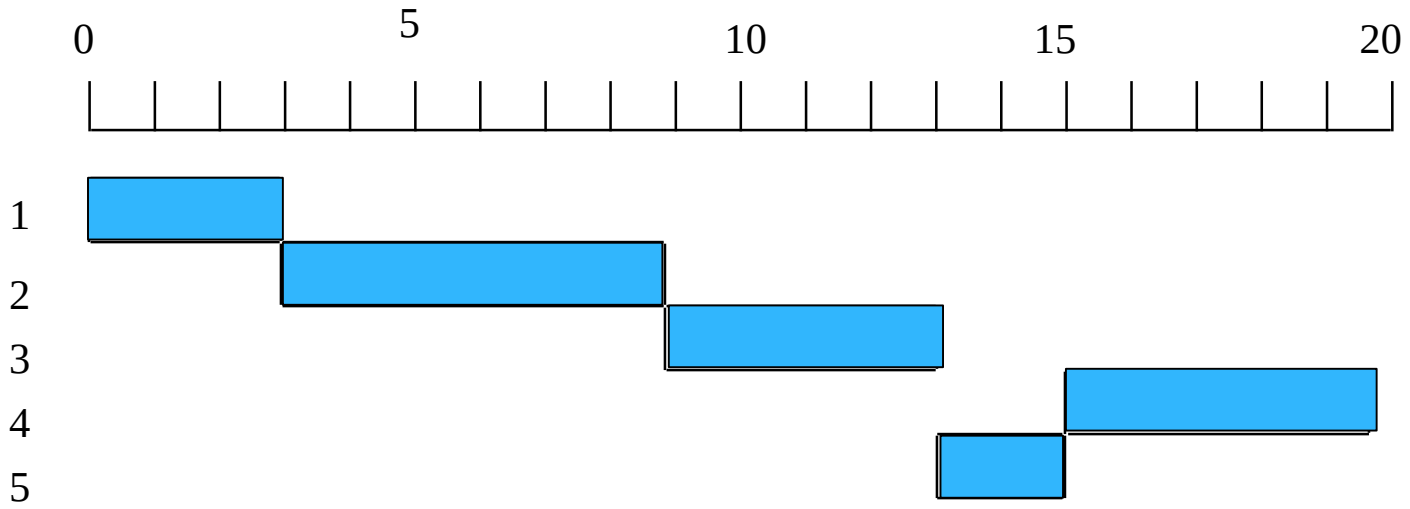
HRRN (最高响应比优先)
Highest Response Ratio Next

* 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

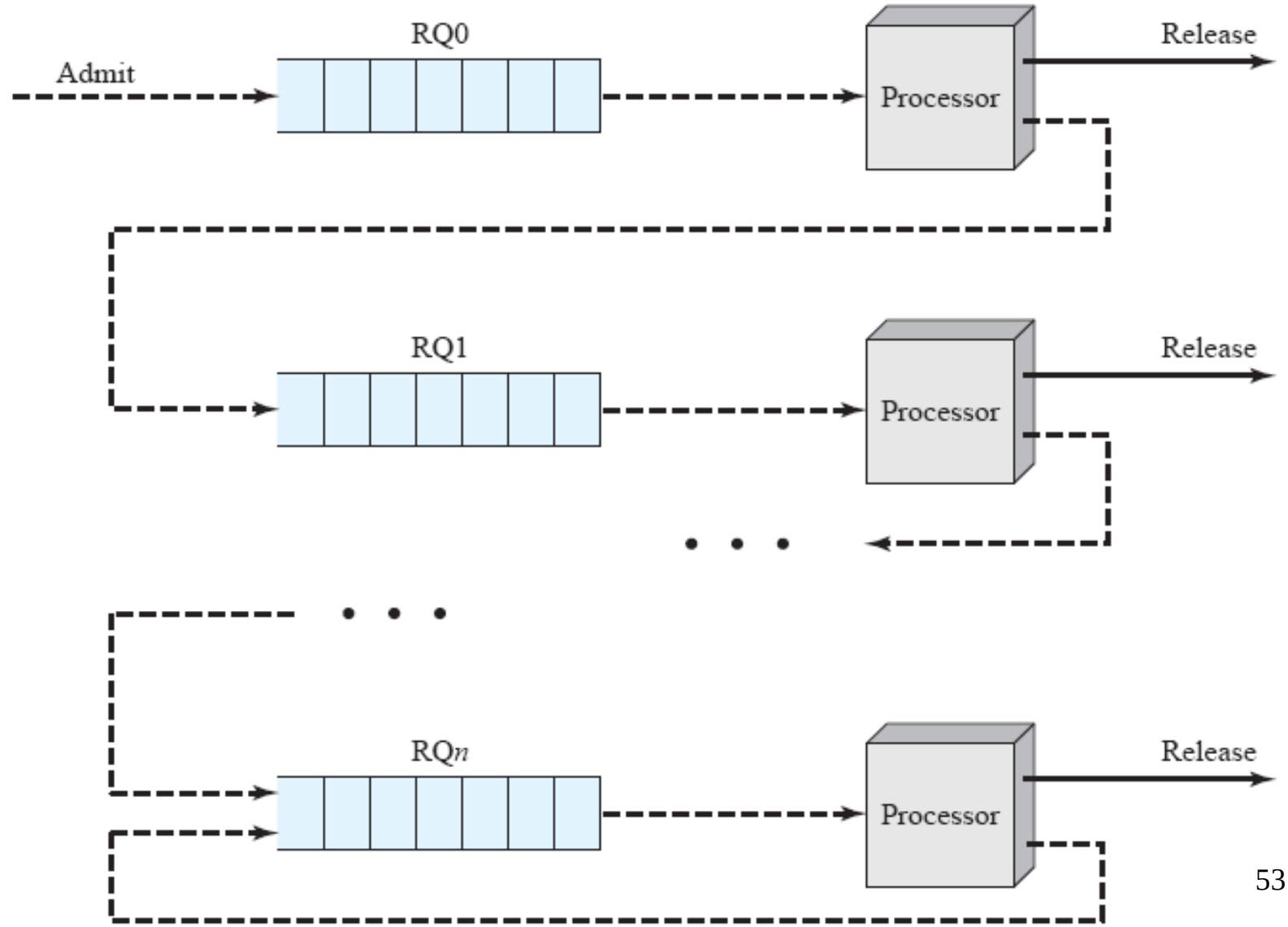


南京大学
NANJING UNIVERSITY

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)

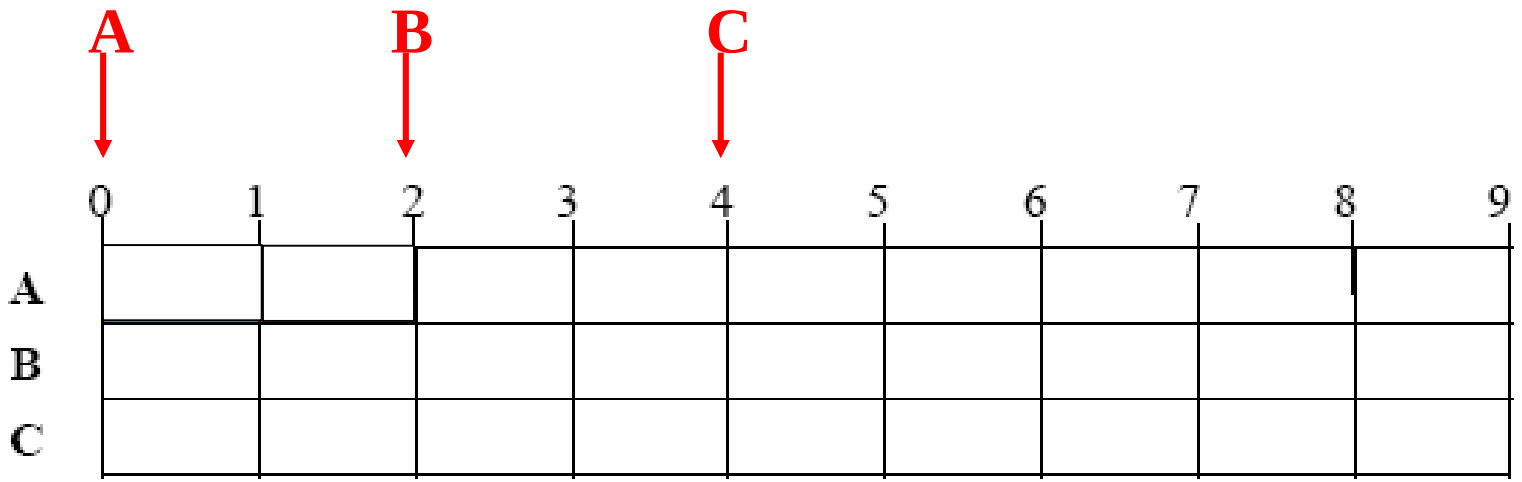
Time=0

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

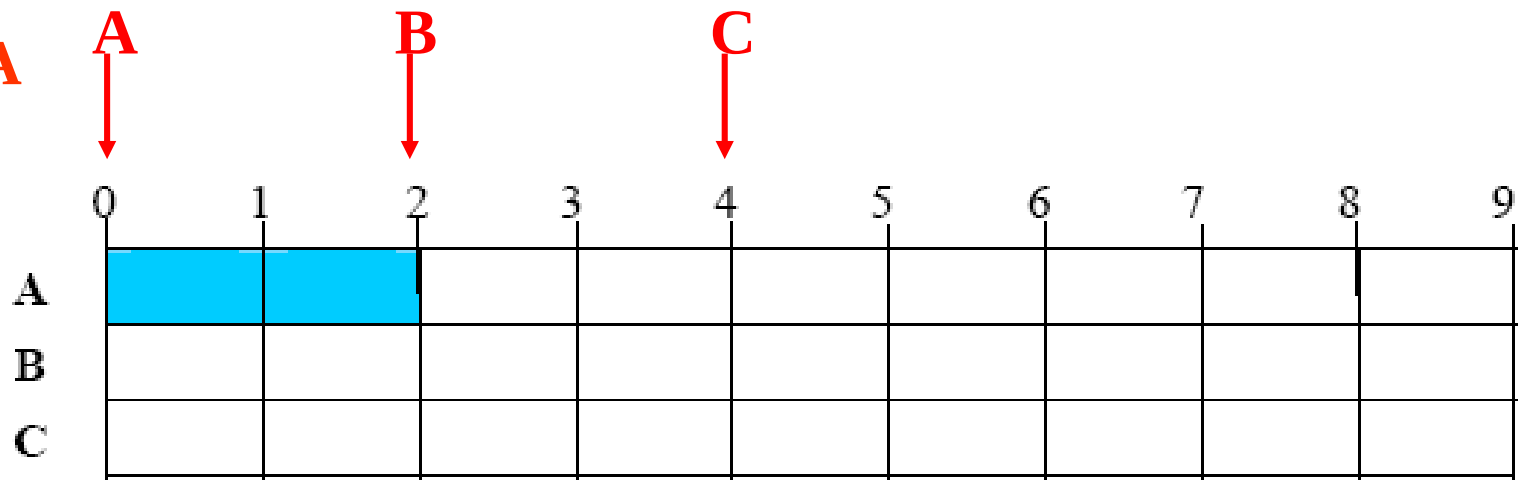
RQ0=

RQ1=

RQ2=

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

Running=A



Feedback (q=1)

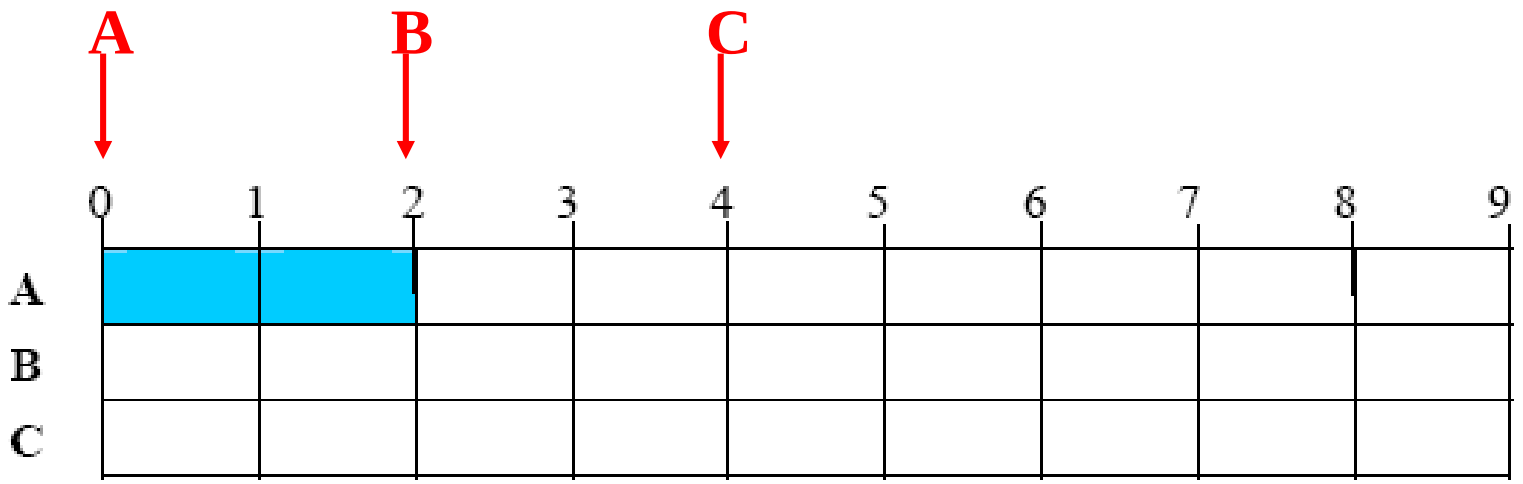
Time=2

RQ0= B

RQ1=

RQ2=

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



Feedback (q=1)

Time=2~3

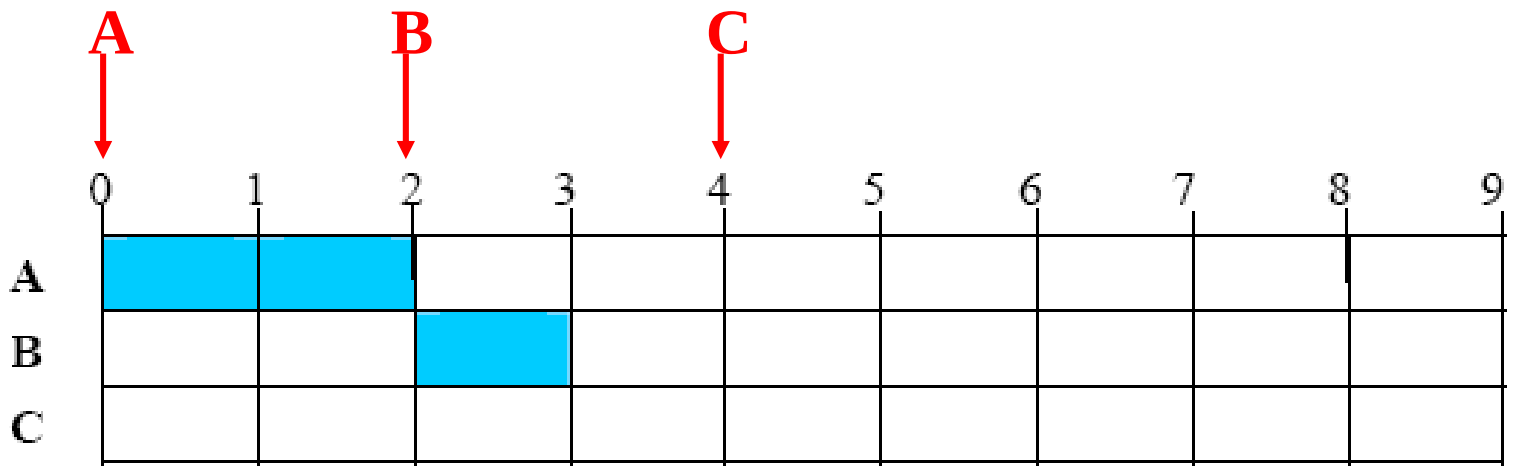
RQ0=

RQ1= A

RQ2=

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

Running=B



Feedback (q=1)

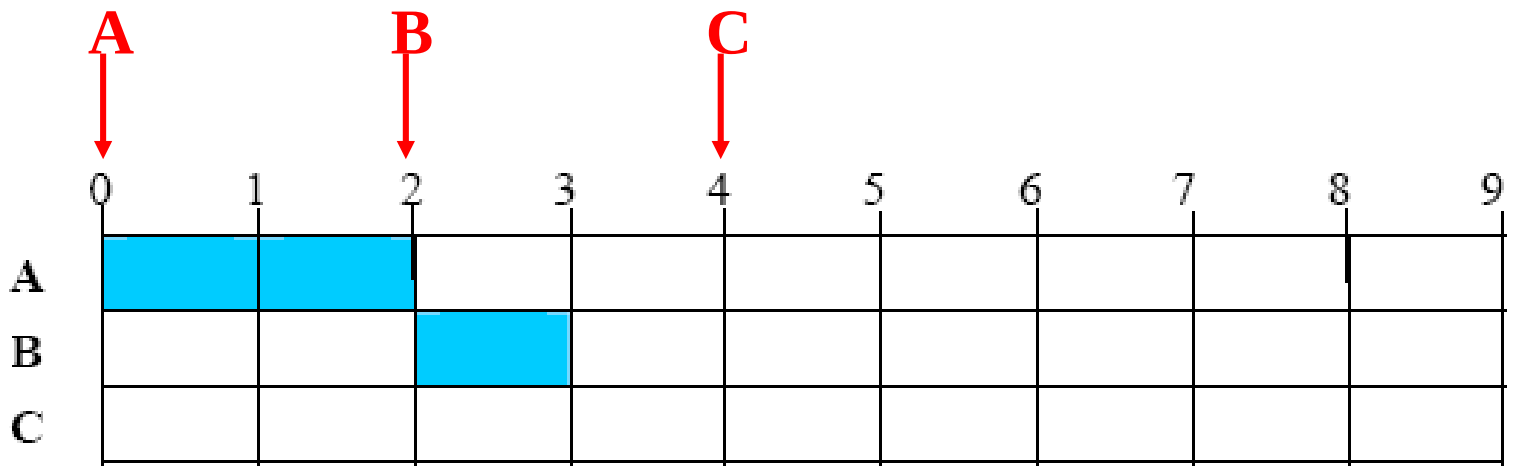
Time=3

RQ0=

RQ1= A

RQ2=

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



Feedback (q=1)

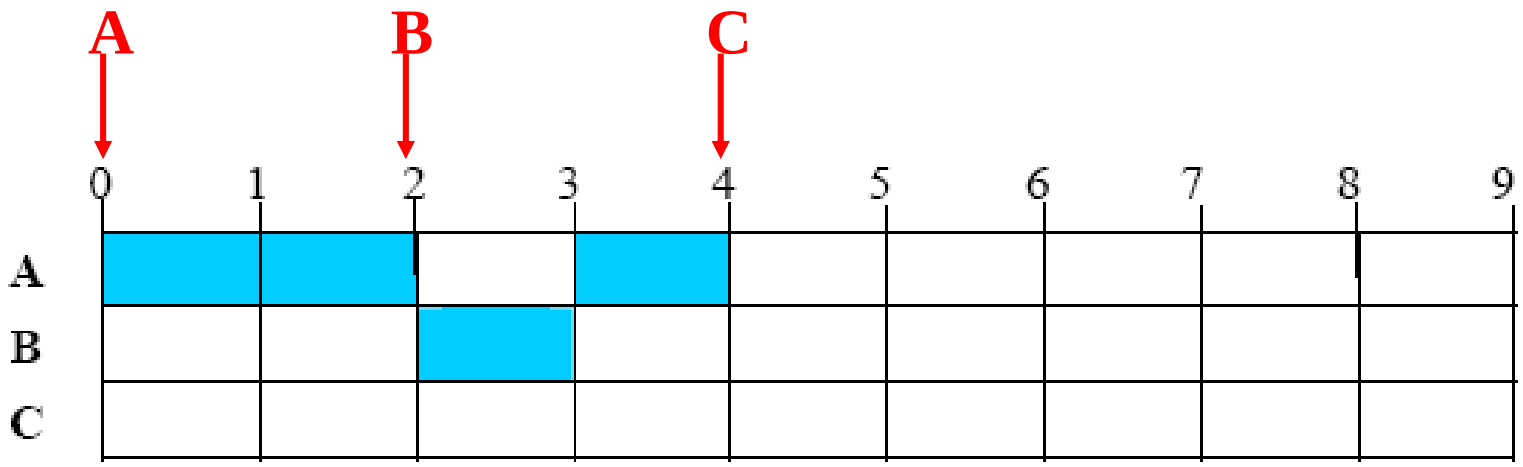
Time=3~4

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

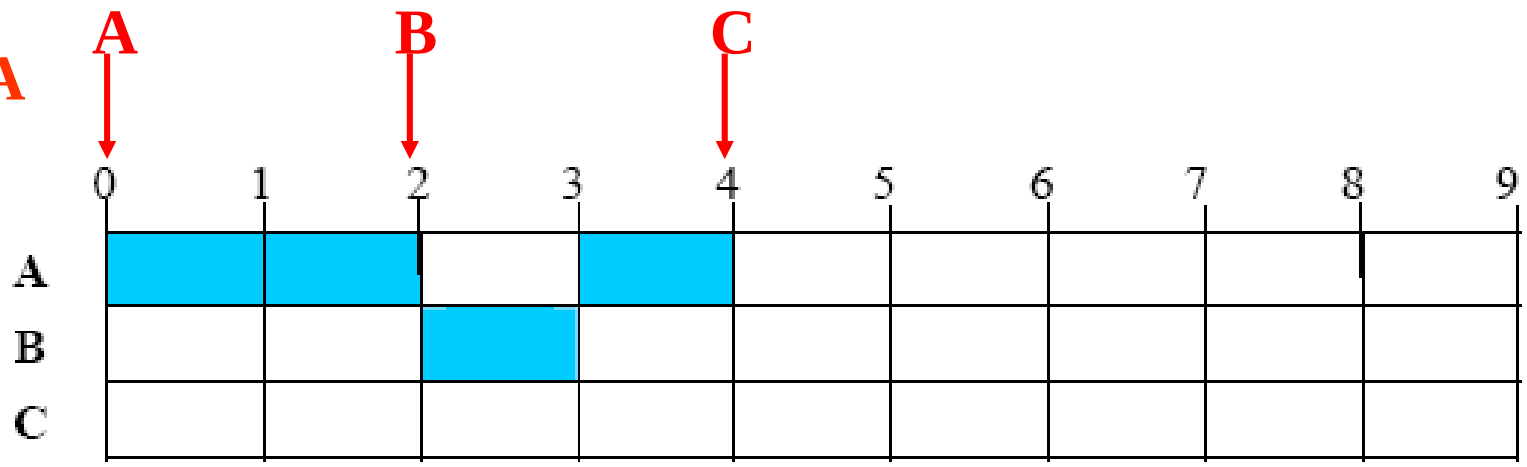
RQ0= C

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

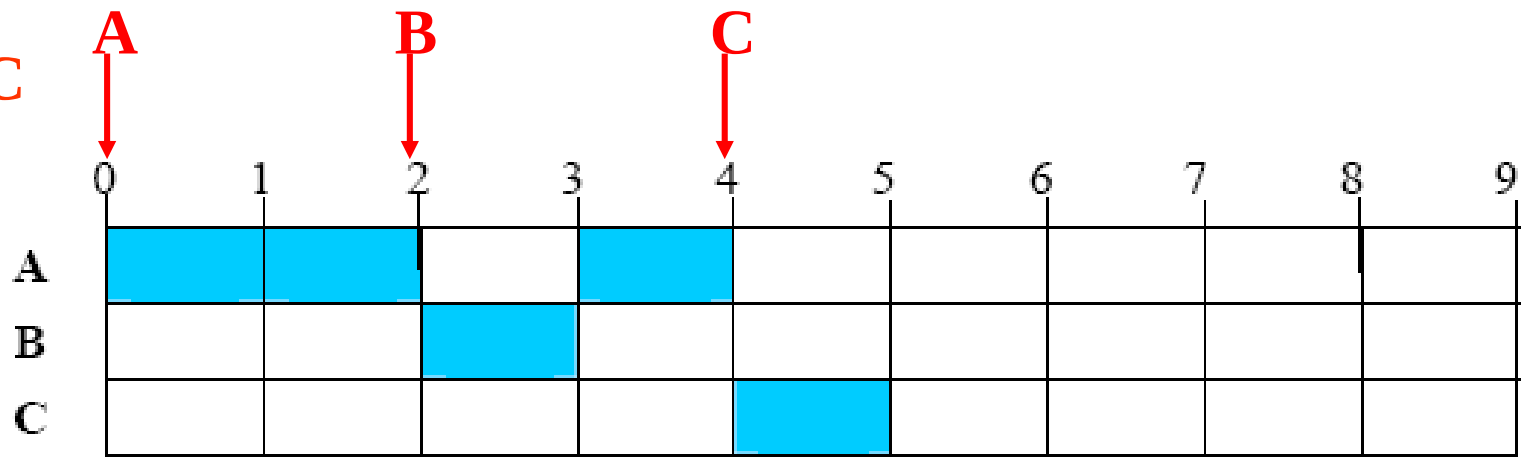
RQ0=

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

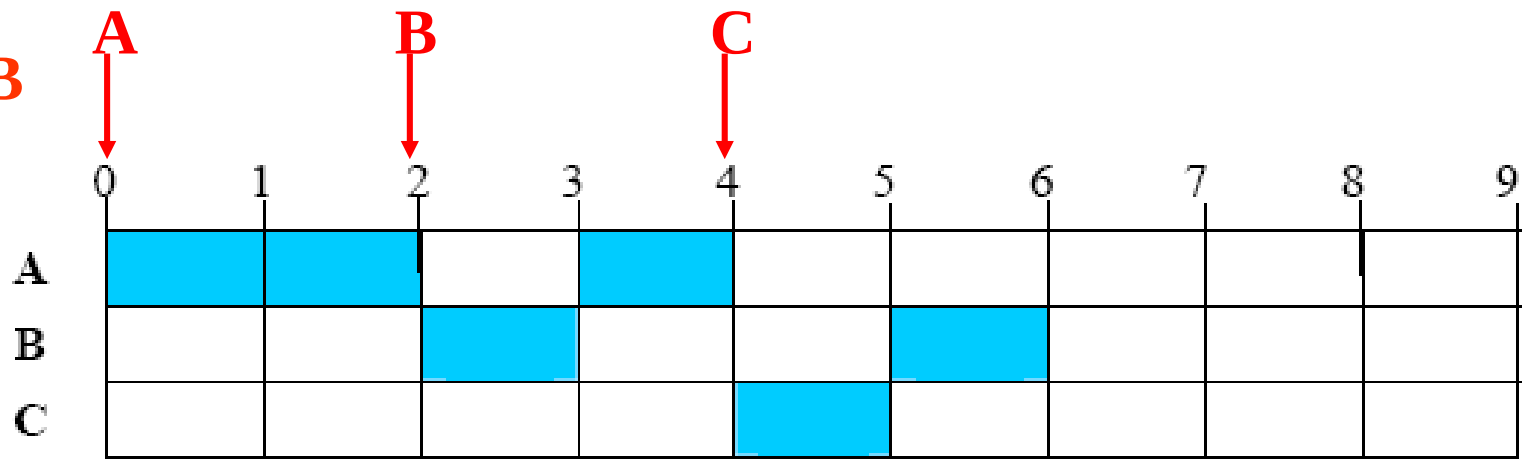
RQ0=

RQ1= C

RQ2=

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

Running=B



Feedback (q=1)

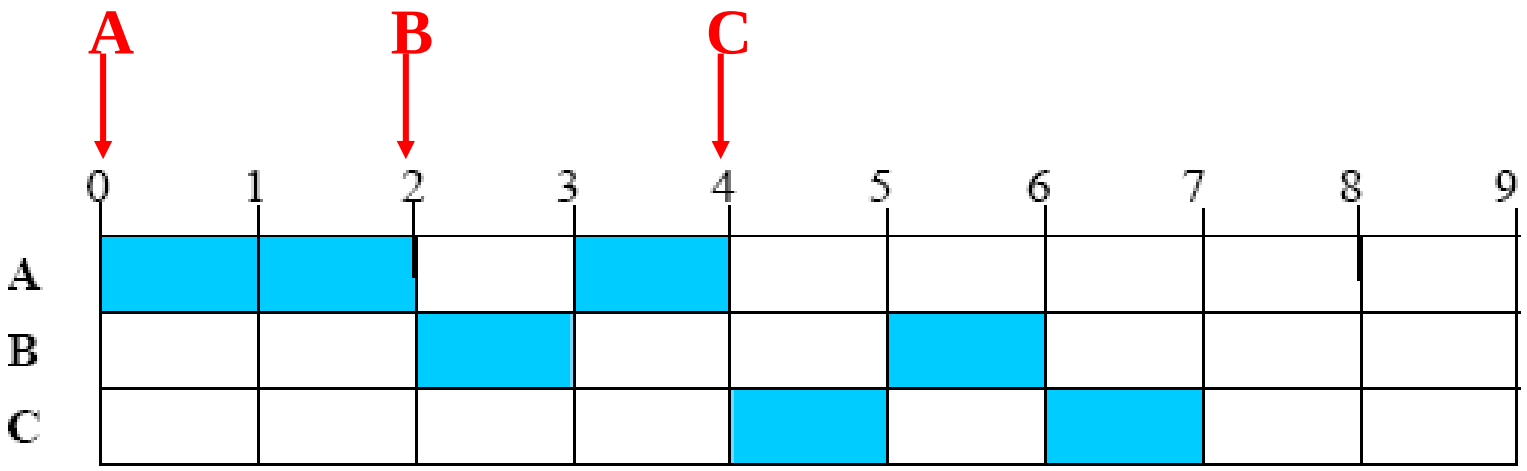
Time=6~7

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)

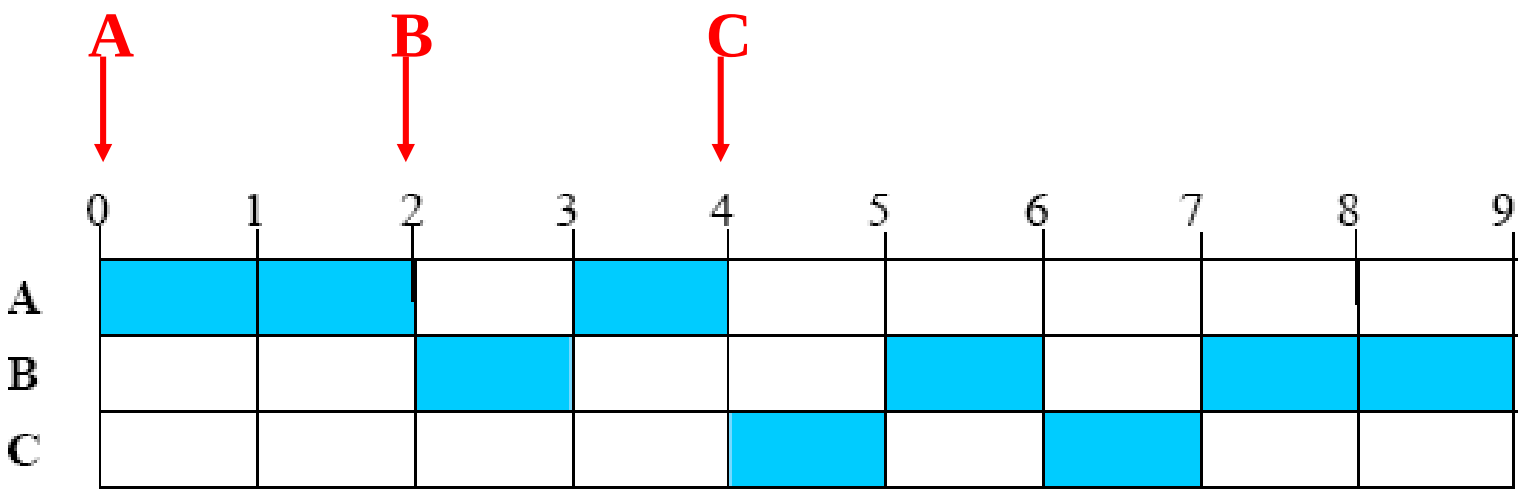
Time=7~9

RQ0=

RQ1=

RQ2=

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



Running=B and finished

Feedback ($q=2^i$)

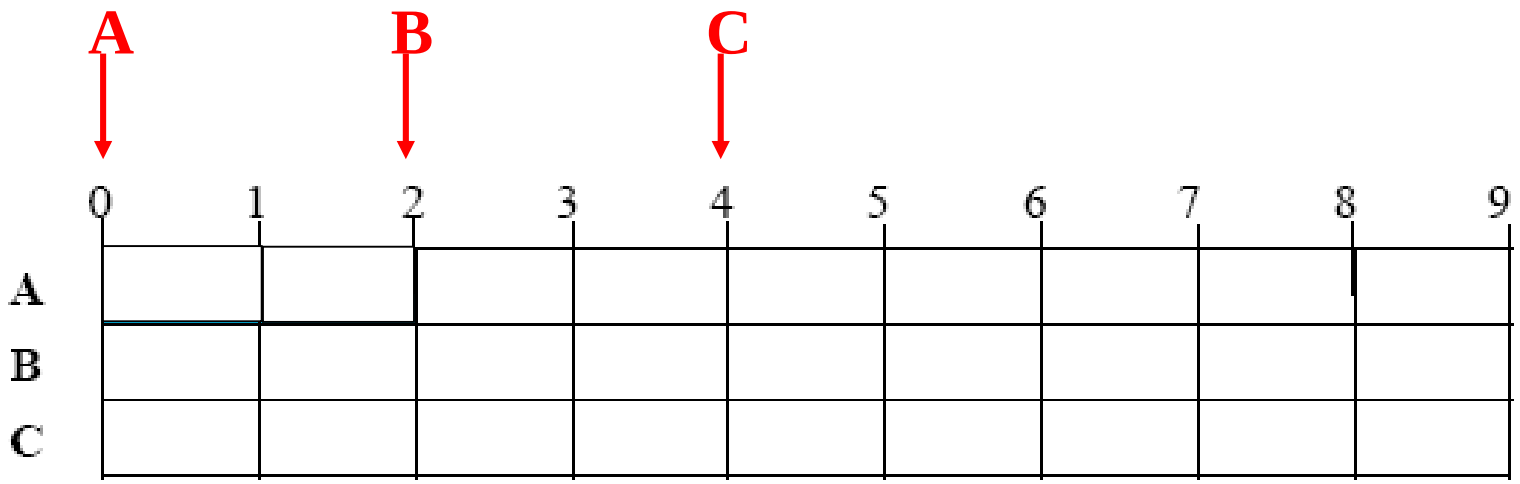
Time=0

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

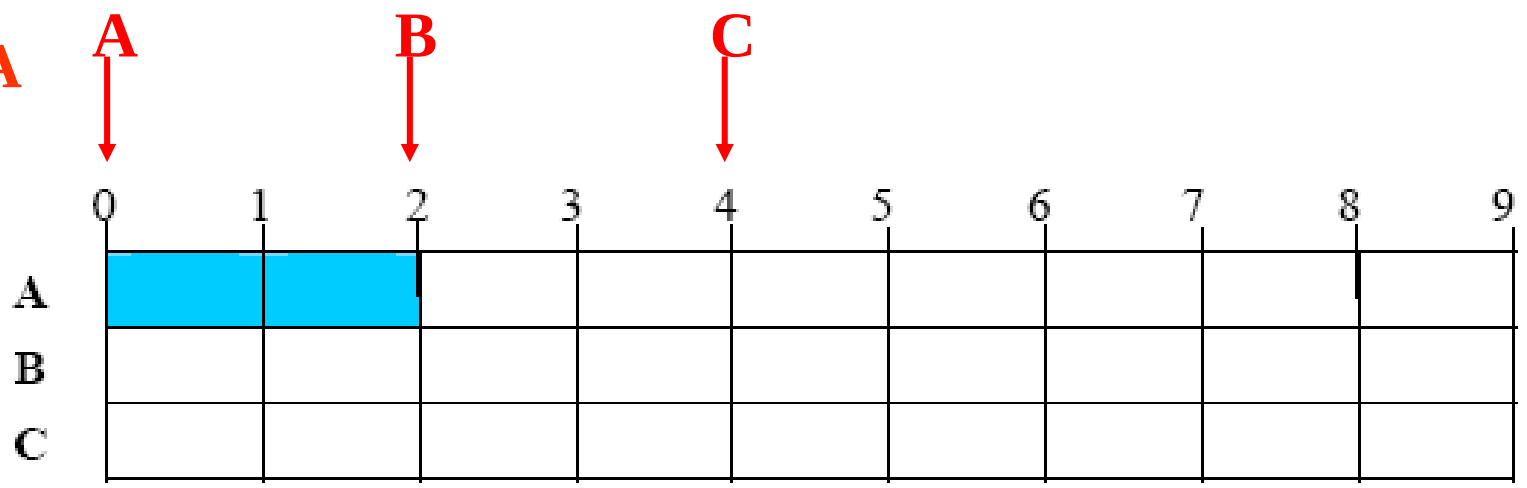
RQ0=

RQ1=

RQ2=

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

Running=A



Feedback ($q=2^i$)

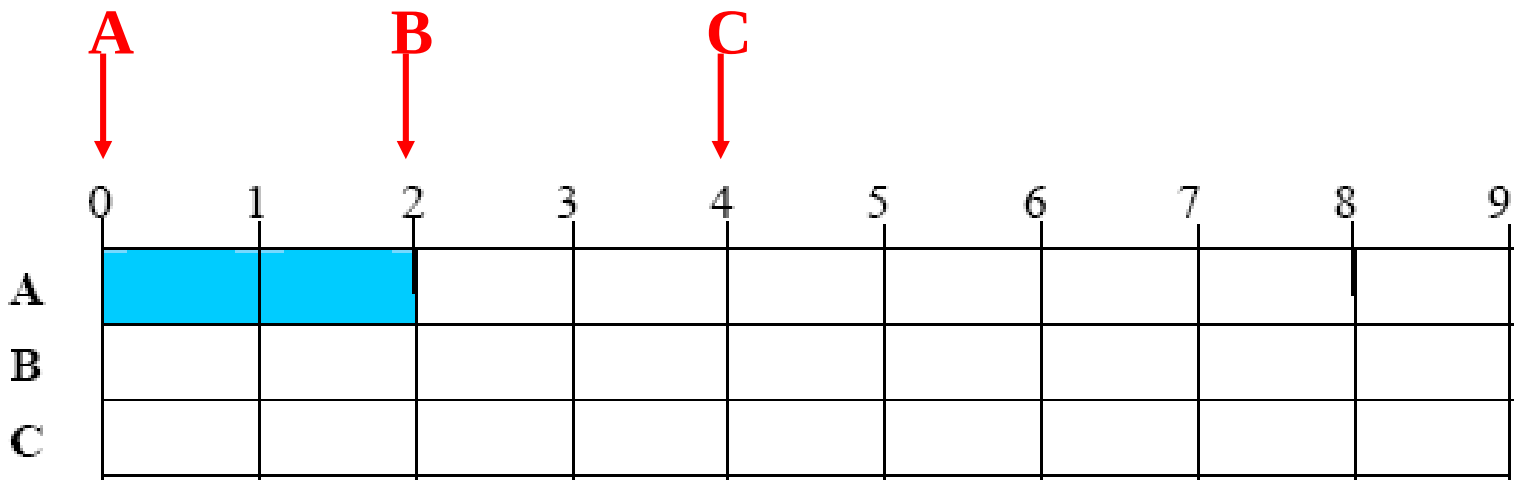
Time=2

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

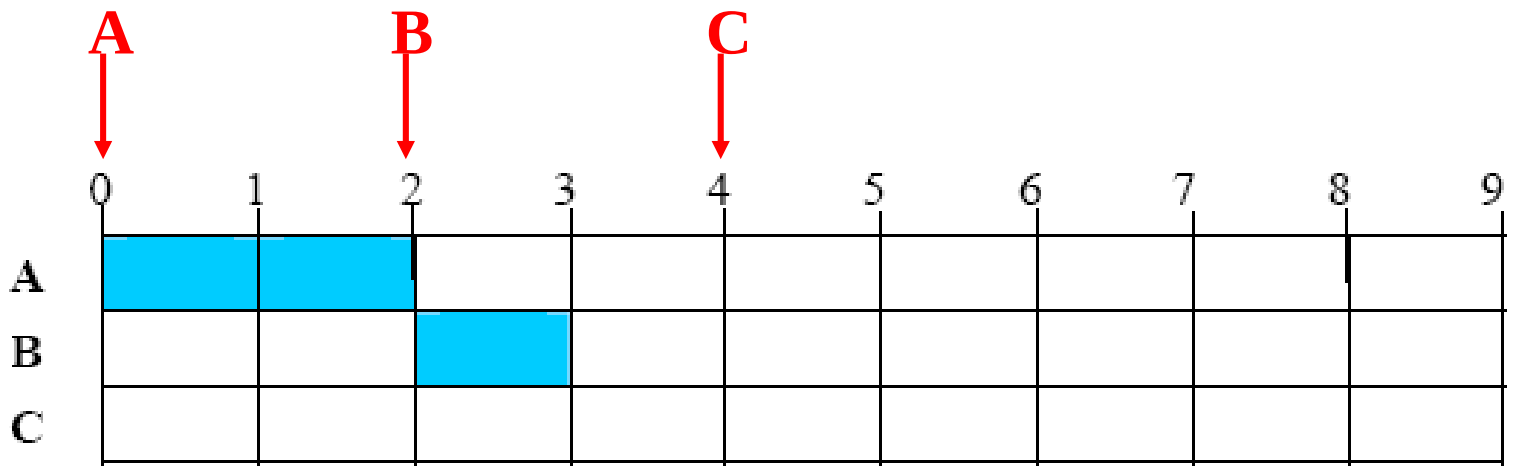
RQ0=

RQ1= A

RQ2=

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

Running=B



Feedback ($q=2^i$)

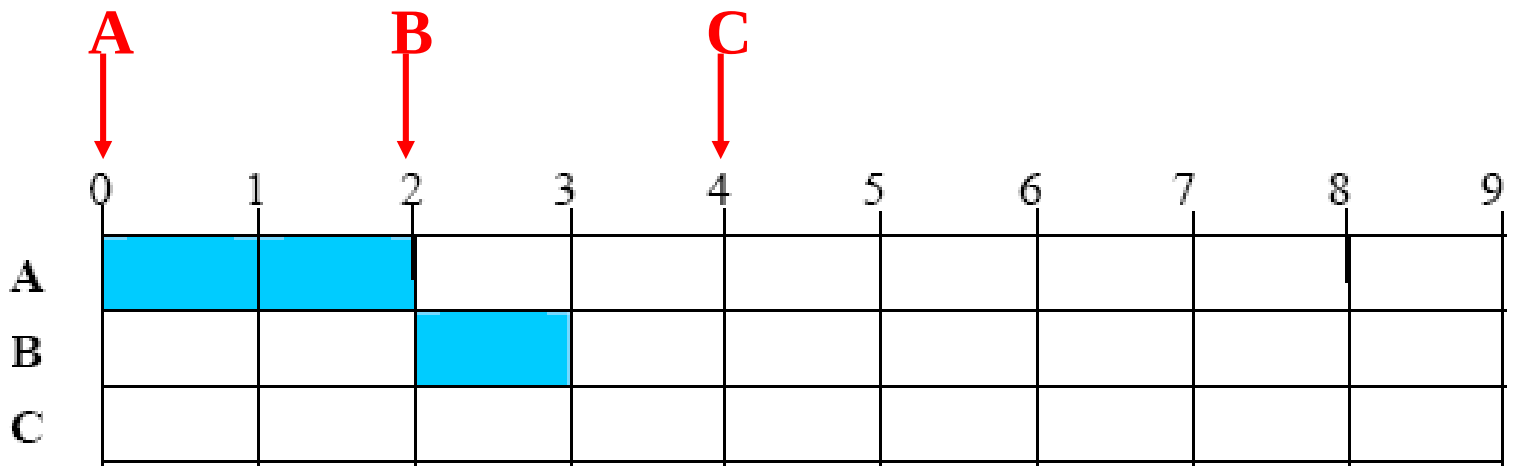
Time=3

RQ0=

RQ1= A

RQ2=

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



Feedback ($q=2^i$)

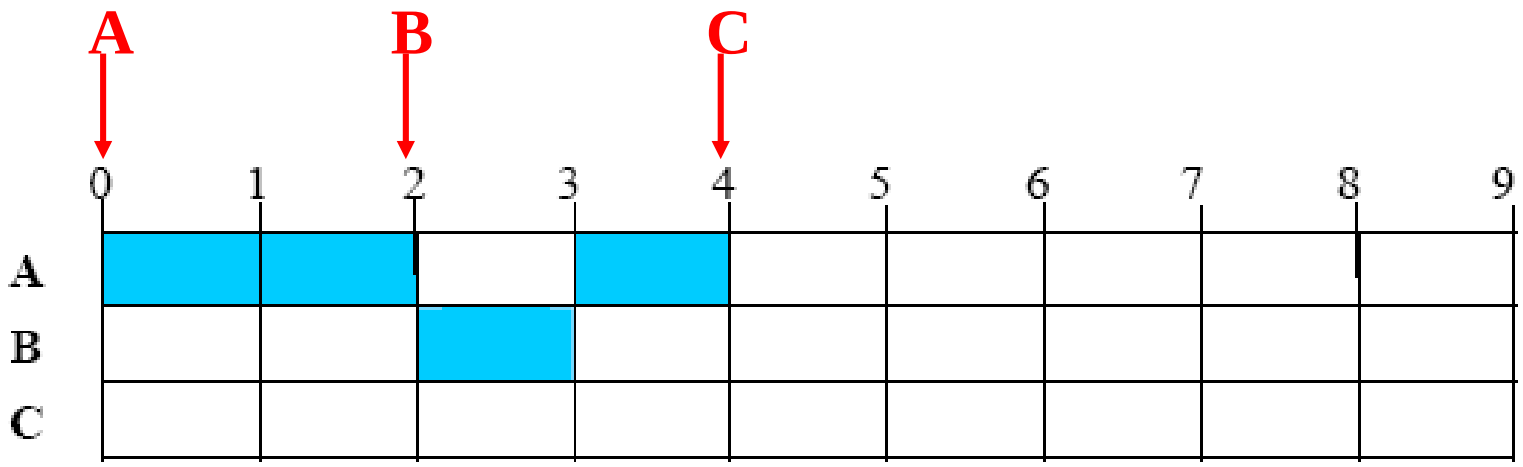
Time=3~4

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

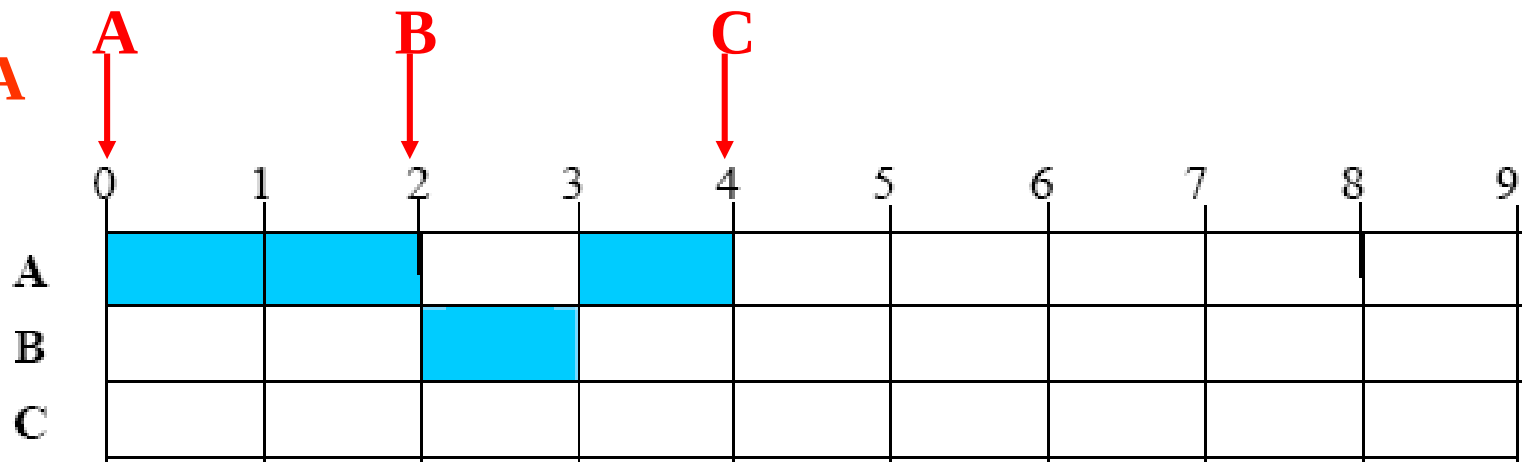
RQ0= C

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

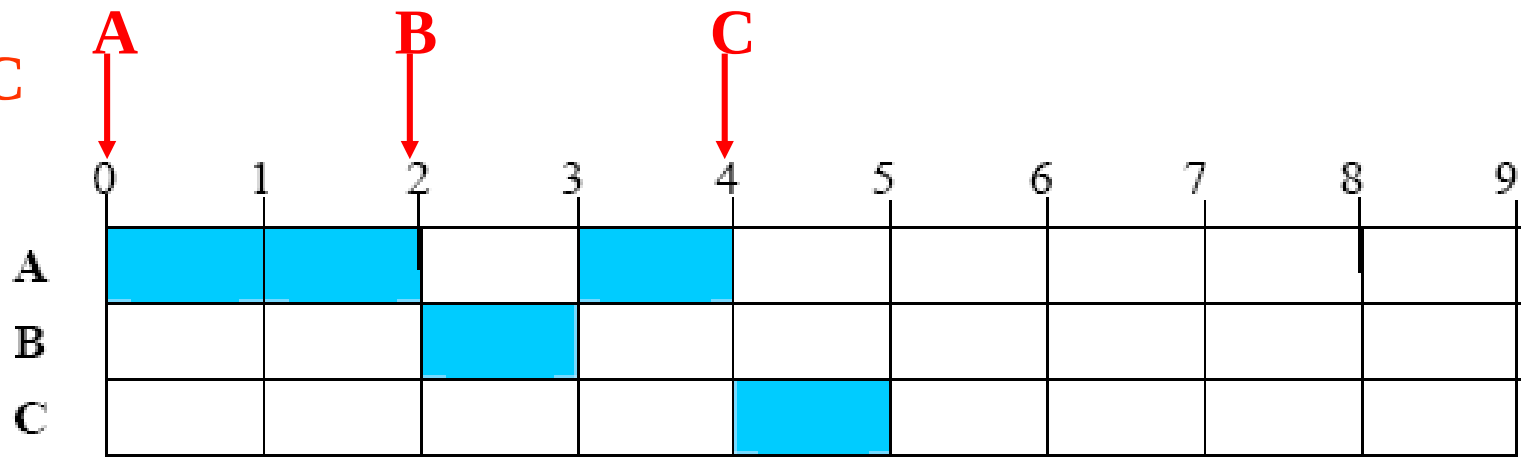
RQ0=

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

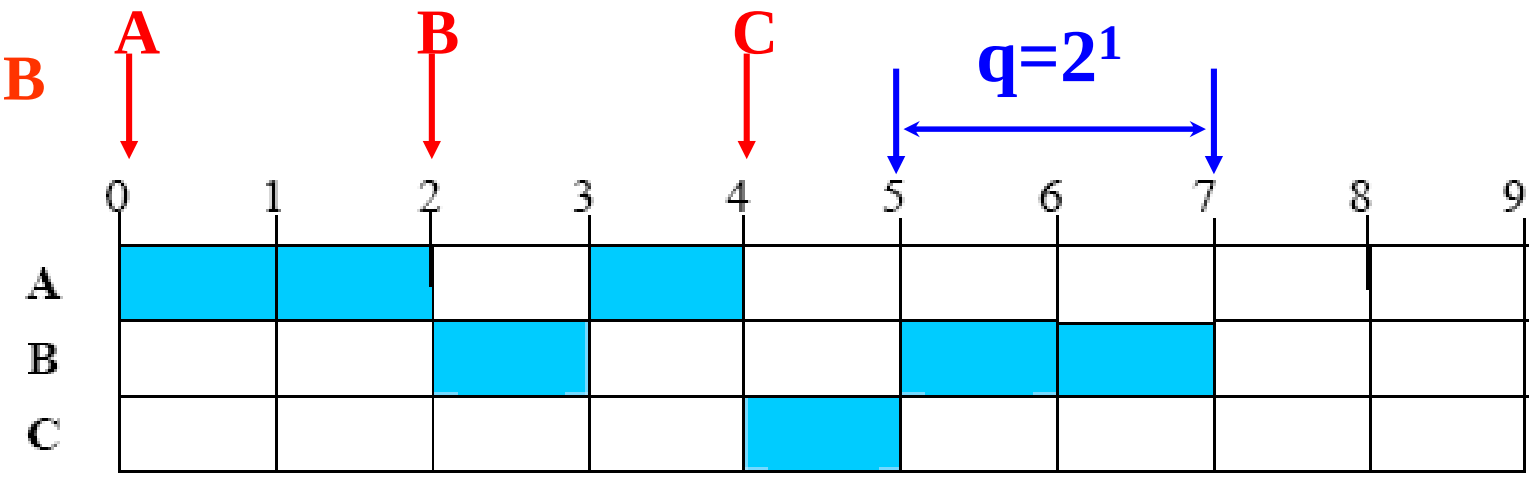
RQ0=

RQ1= C

RQ2=

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

Running=B



Feedback ($q=2^i$)

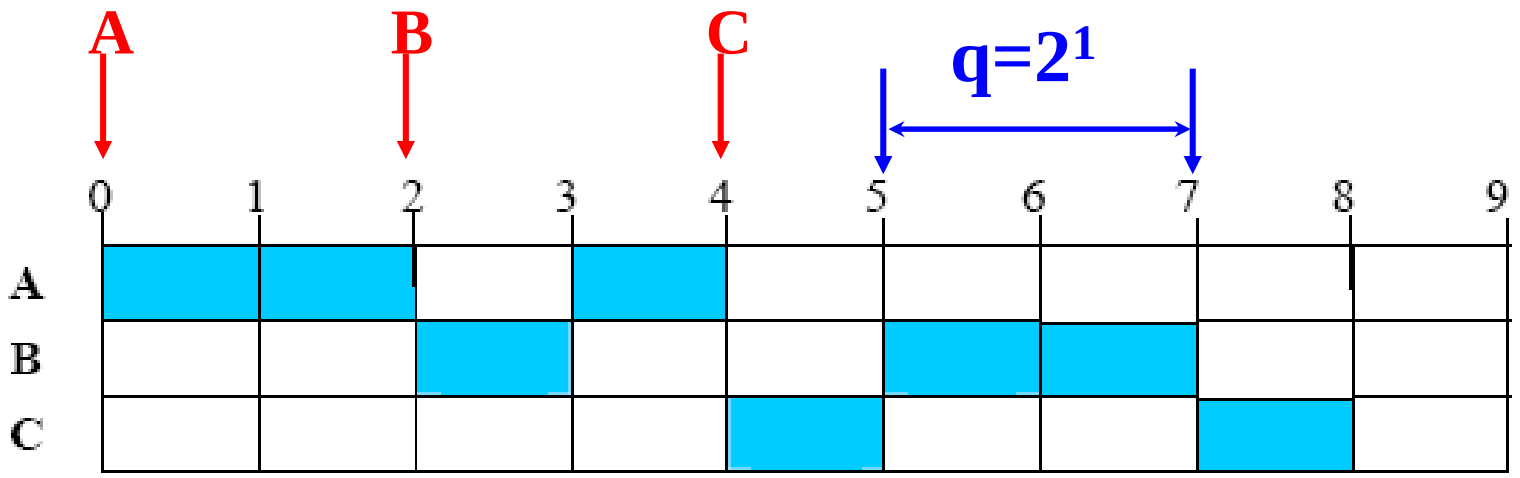
Time=6~7

RQ0=

RQ1=

RQ2= B

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



Running=C and finished

Feedback ($q=2^i$)

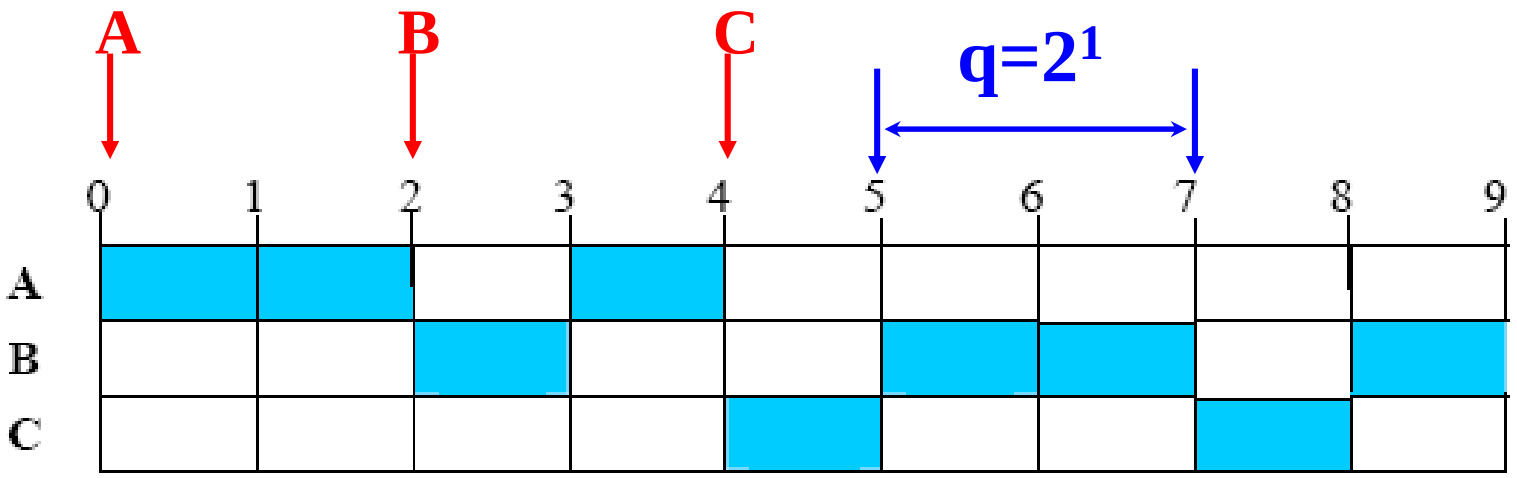
Time=7~9

RQ0=

RQ1=

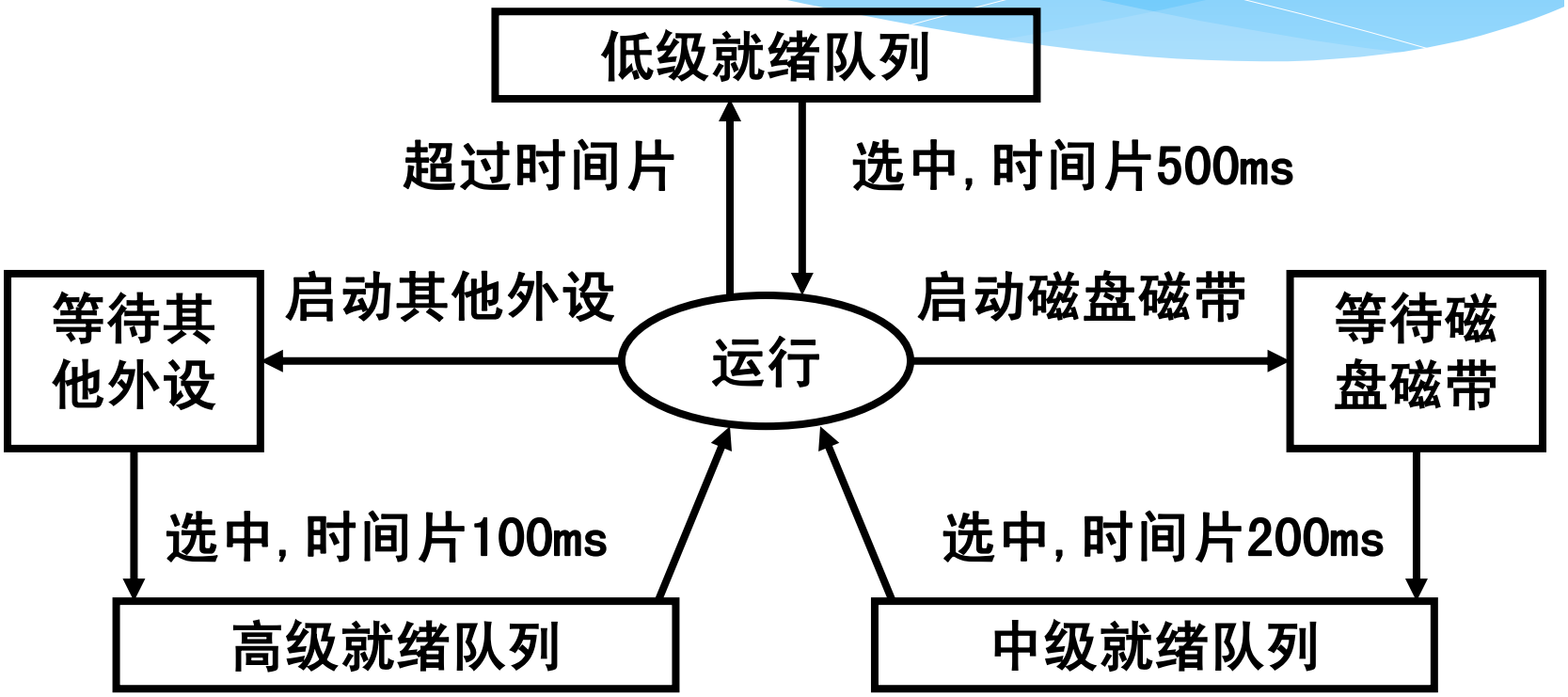
RQ2=

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

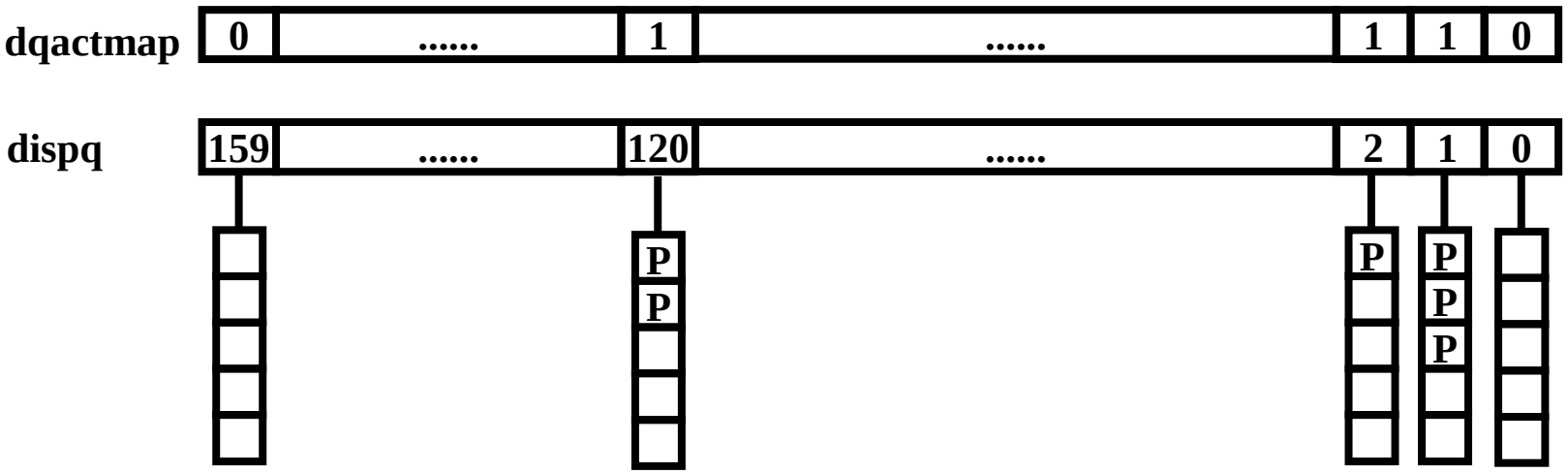
南京大学
NANJING UNIVERSITY

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)

* 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


南京大学
NANJING UNIVERSITY

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

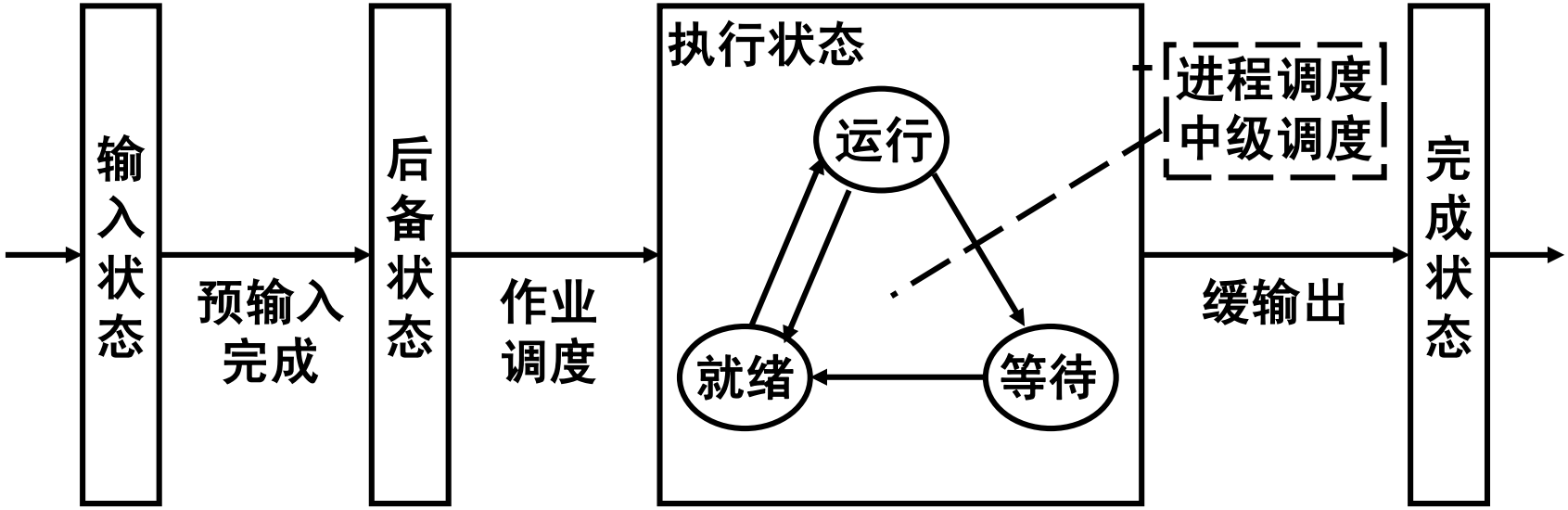
* 基本思想：为进程发放针对系统各种资源(如CPU时间)的彩票；当调度程序需要做出决策时，随机选择一张彩票，持有该彩票的进程将获得系统资源
* 合作进程之间的彩票交换

6.4 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



南京大学
NANJING UNIVERSITY

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

批处理作业的调度算法

* 先来先服务算法
* 最短作业优先算法
* 响应比最高者优先算法

  响应比＝已等待时间/估计计算时间
* 优先数法
* 分类调度算法
* 用磁带与不用磁带的作业搭配算法

本主题小结

1. 掌握调度的层次
2. 掌握进程的挂起态
3. 掌握低级调度及其策略
4. 掌握作业调度及其策略