第四部分　调度

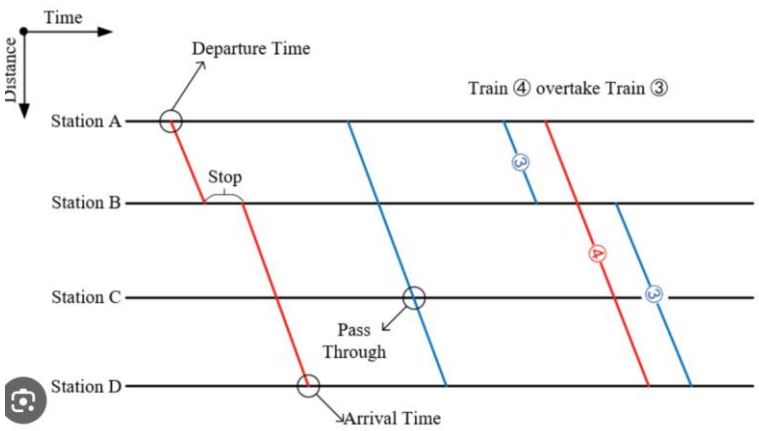
处理器调度的类型

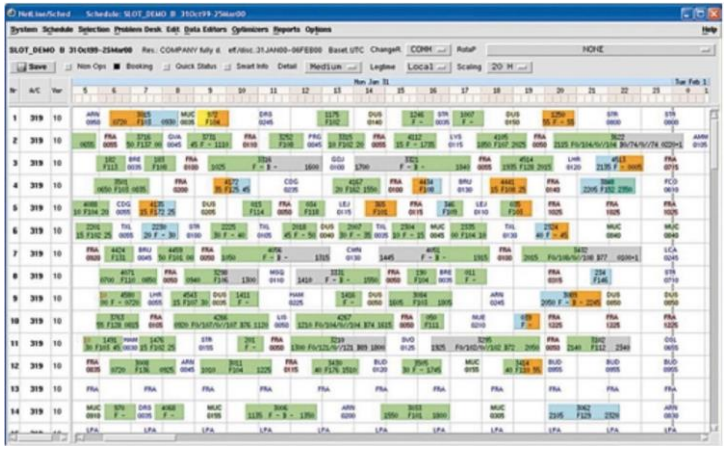
单处理器调度算法

多处理器调度

实时调度

生活中的调度





火车调度

飞机调度

9.1 处理器调度

处理器要执行哪些进程？

- 长程调度(Long-term scheduling)
  - 决定哪些新建进程可进入系统准备执行
  - 控制多道程序系统的并发程度
  - 进程越多则各进程对CPU的使用百分比越小
- 中程调度(Medium-term scheduling)
  - 决定交换哪些主存—辅存（内存-外存）进程
  - 基于多道程序设计的管理需要

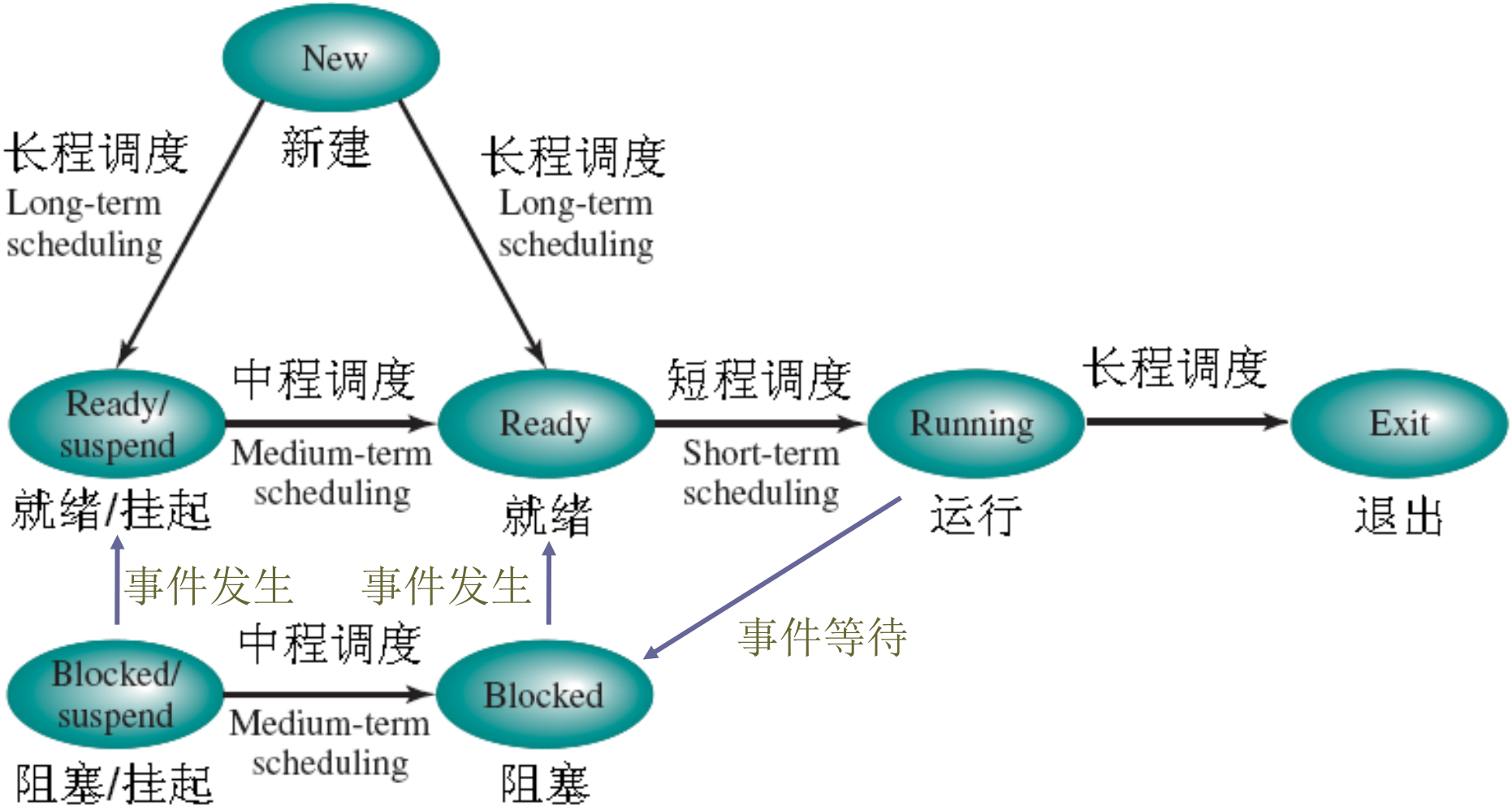
9.1 处理器调度

- 长程调度(Long-term scheduling)
- 中程调度(Medium-term scheduling)
- 短程调度(Short-term scheduling)
  - 决定下一个使用CPU的进程（dispatcher，分派程序）

I/O调度（第11章）

决定可用的I/O设备处理哪个进程挂起的I/O请求

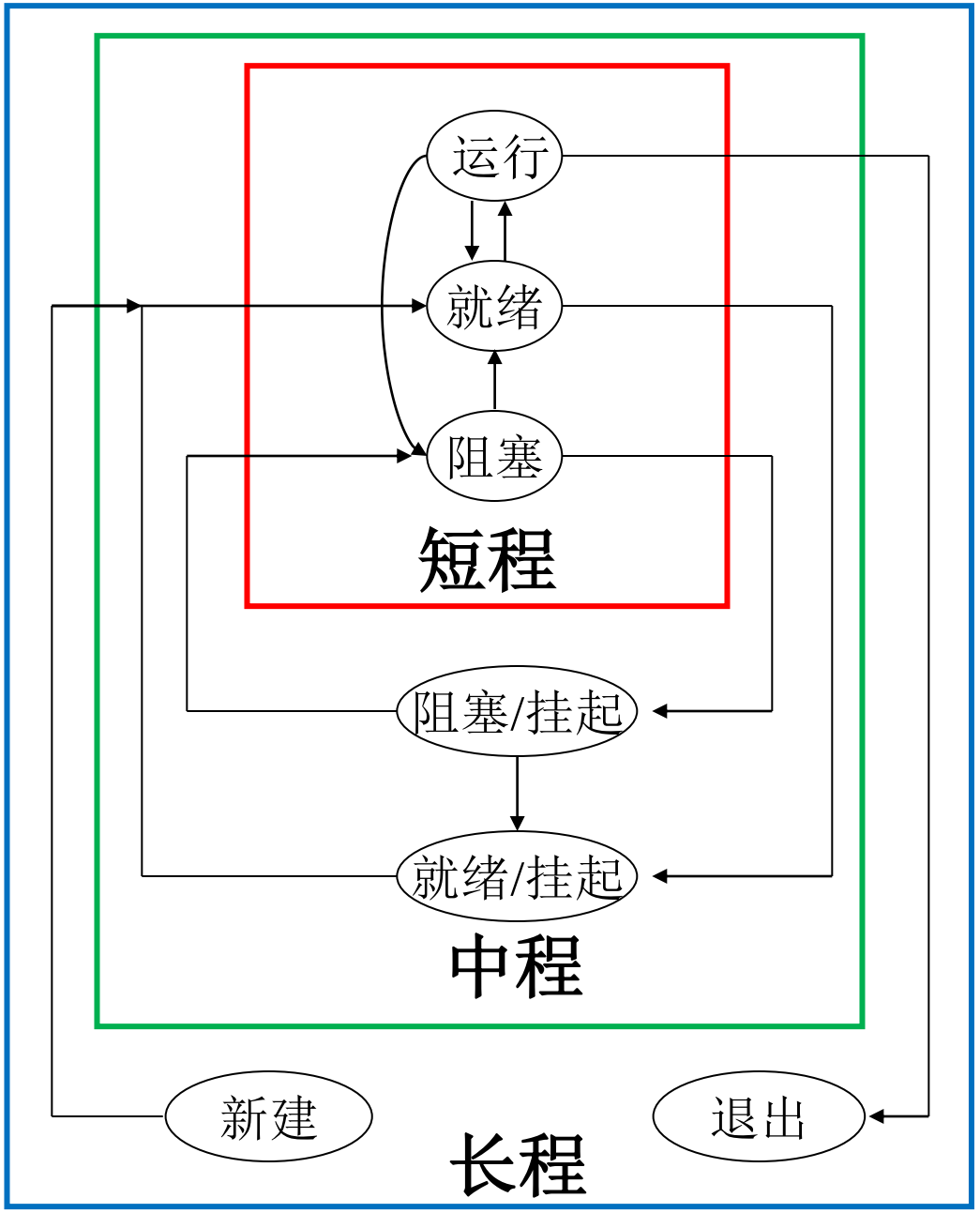
调度与进程状态转换



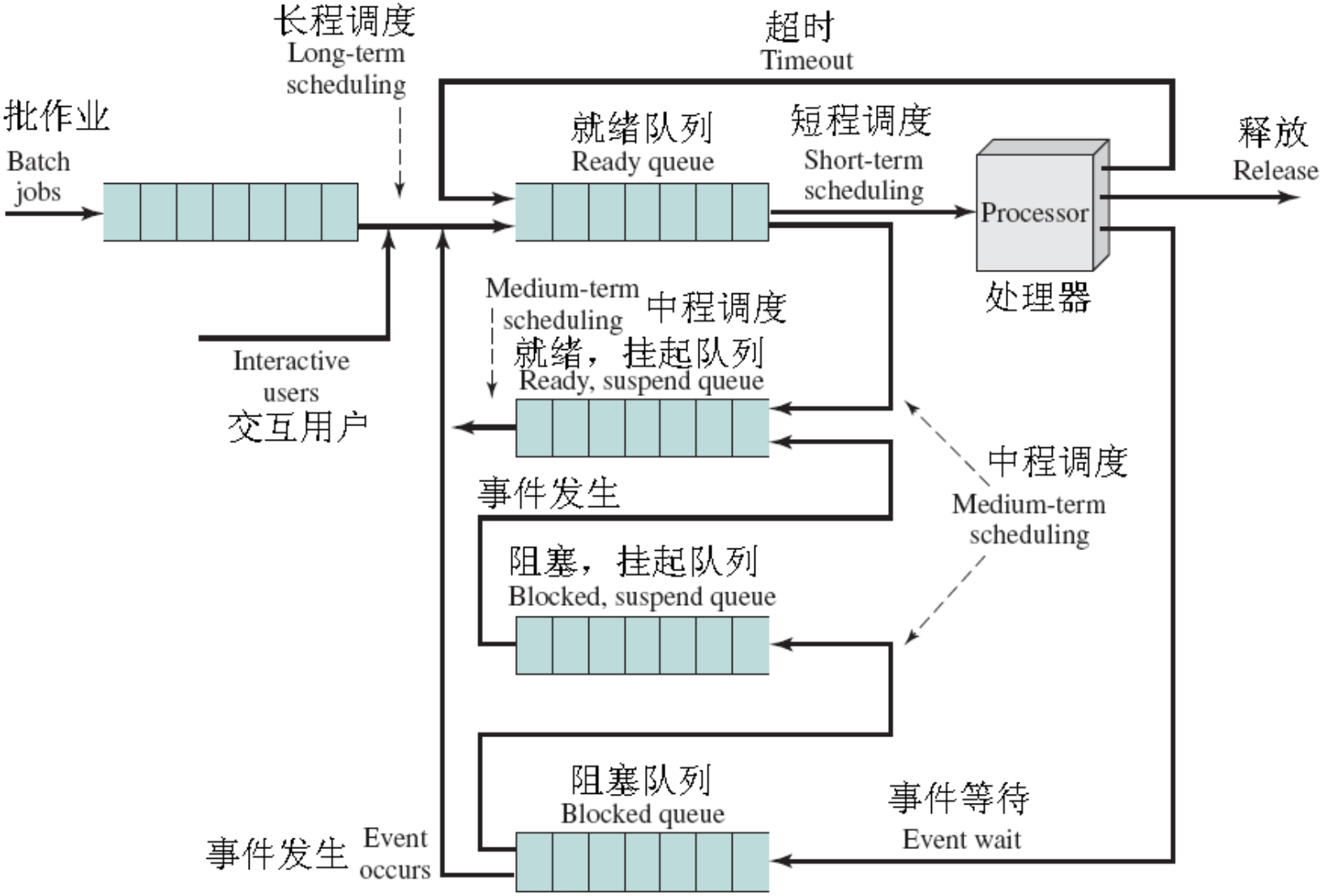
Scheduling and Process State Transitions

调度与进程状态转换

调度的层次



调度队列



Queueing Diagram for Scheduling

调度的队列图

短程调度时机

- 当前进程正常或异常终止（通过中断实现）
- 时钟或I/O中断
- 系统调用（通过软中断实现）
- 信号量操作（通过软中断实现）

短程调度模式

- 非剥夺式（nonpreemptive）
  - 让进程运行直到结束或阻塞的调度方式
  - 容易实现
  - 适合专用系统，不适合通用系统

短程调度模式

- 非剥夺式（nonpreemptive）
- 剥夺式（preemptive）
  - 允许将逻辑上可继续运行的进程在运行过程中暂停的调度方式
  - 可防止单一进程长时间独占CPU
  - 系统开销大（降低途径：硬件实现进程切换，或扩充主存以贮存大部分程序）

短程调度过程

- 进程上下文切换过程

1 保存现场

2 根据某种调度算法选择下一个运行的进程

  - 没有就绪进程，系统执行空闲进程 (idle)
  - 没有其他进程时该进程一直运行
  - 在执行过程中可接收中断

3 恢复现场

短程调度目标

- 面向用户的目标与面向系统的目标
- 定量目标与定性目标

短程调度目标

- 公平——确保每个进程都获得合理的CPU份额
- 效率——使CPU及其他系统资源尽量忙碌
- 响应时间（从提交到开始输出结果）——尽可能短
  - 在交互式系统中尤为重要
- 周转时间T_r（turnaround time：从提交到结束）（也叫驻留时间，residence time）——尽可能短
- 吞吐量（单位时间内完成的进程数）——尽可能大
- 实时性——可以指定进程完成的最后期限

9.2 进程调度算法

- 先来先服务(First Come First Served，FCFS)
- 最短进程优先(Shortest Process Next，SPN)
- 最短剩余优先(Shortest Remaining Time，SRT)
- 最高响应比优先(Highest Response Ratio Next，HRRN)

9.2 进程调度算法

- 时间片（time slicing）轮转(Round Robin，RR)
- 最高优先级优先(Highest Priority First，HPF)
- 多级队列反馈(Multilevel Feedback，MF/FB)

各调度策略的特点

类别 缩写	先来先服务 FCFS	轮转 RR	最短进程优先 SPN	最短剩余优先 SRT	最高响应比优先 HRRN	反馈 MF/FB
选择函数	max(w)	常数	min(s)	min(s-e)	max((w+s)/s)	e, 优先级
决策模式	非抢占	抢占 (时间片)	非抢占	抢占 (到达时)	非抢占	抢占 (时间片)
吞吐量	不强调	时间片太小时会低	高	高	高	不强调
响应时间	可能大	短进程小	短进程小	小	小	不强调
开销	最小	最小	可能高	可能高	可能高	可能高
对进程的影响	对短进程和I/O密集进程不利	公平对待	对长进程不利	对长进程不利	很好的平衡	可能对I/O密集进程有利
饥饿	无	无	可能	可能	无	可能

w：已等待时间、e：已执行时间

s：进程所需总服务时间

进程调度例

进程	到达时间	服务时间
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

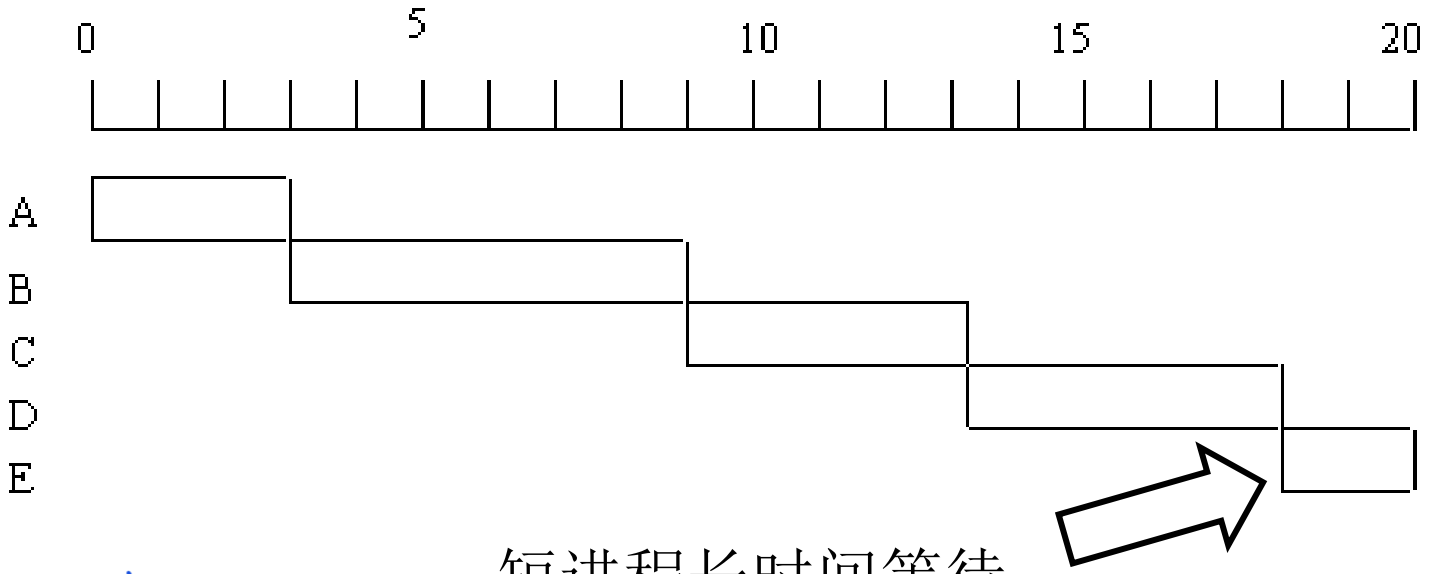
先来先服务（FCFS）

进程	到达时间	服务时间
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

- 当前进程结束后，选择最早到达

就绪队列的进程（非剥夺式）

- 短进程等待执行的时间可能较长
- I/O密集进程必须等待CPU密集进程结束

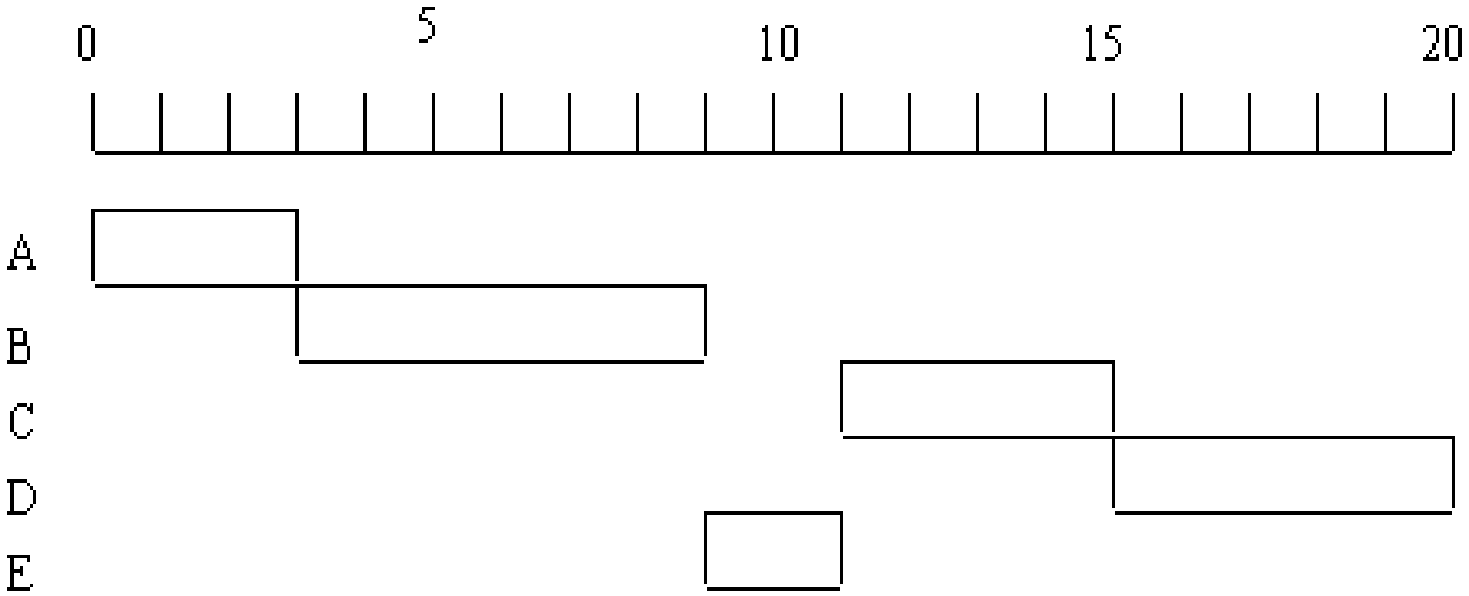


短进程长时间等待

最短进程优先（SPN）

进程	到达时间	服务时间
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

- 选取估计运行时间最短的进程
- 不可剥夺式
- “长”进程可能饿死



（不适合分时和事务处理系统）

最短进程优先的证明

假设有n个任务顺序完成，完成这n个任务的时间为$t_1, t_2, \ldots, t_n$，则完成这n个任务的总时间为

$$T = nt_1 + (n-1)t_2 + \ldots + t_n.$$

设上述式子中t_i的系数为a_i。

现在证明一个命题：对于上式，若存在$t_j \leq t_i$，$i < j$，则交换t_i, t_j后得到的T'不大于T。

证明：显然，$a_j < a_i$，且有

$$T' = nt_1 + (n-1)t_2 + \ldots + a_i t_j + \ldots + a_j t_i + \ldots + t_n.$$

则

$$T - T' = (a_i - a_j)(t_i - t_j) \geq 0.$$

当且仅当$t_i = t_j$时，上述不等式等号成立。

证毕。

由上述命题很容易地得到，当且仅当$t_1 \leq t_2 \leq \ldots \leq t_n$时，$T$最小。

最短进程优先（SPN）

- 难点：预知或估计进程的执行时间
  - 生产环境：统计
  - 交互式环境：估计

最短进程优先（SPN）

- 进程执行时间的估计：“指数平滑”技术

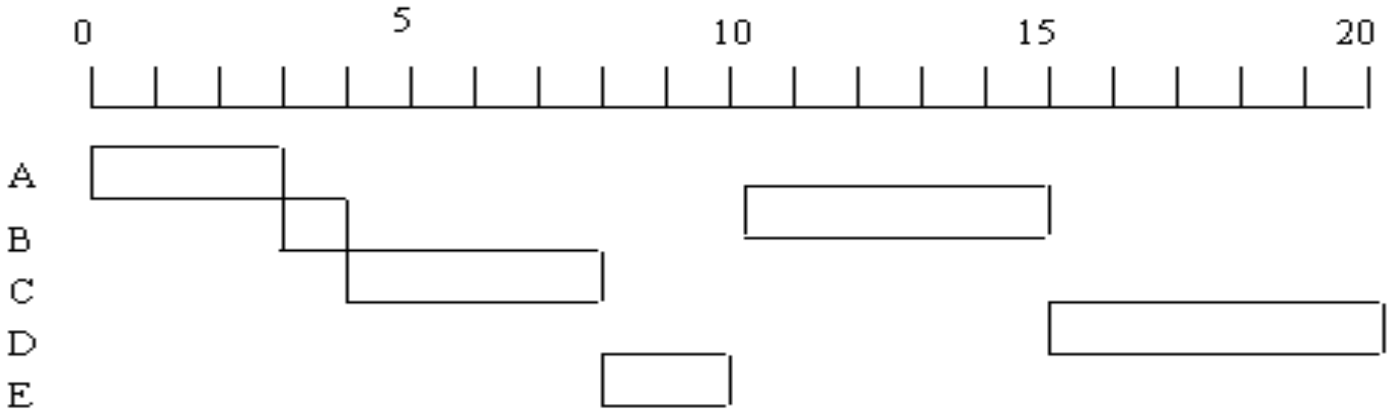
$$S_{n+1} = aT_n + (1-a)S_n$$

  - S_n：第 n 个实例执行时间的估计值
  - T_n：第 n 个实例执行时间的测量值
  - a：加权系数， a 值越大，对变化反应越快(对老的运行时间忘记得越快)($0 < a < 1$)

最短剩余优先（SRT）

进程	到达时间	服务时间
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

- 剥夺式的SPN
  - 新进程进入就绪队列时引发重新调度
  - 选取估计剩余时间最短的那个进程
- 需要估计进程剩余执行时间
- “长”进程可能饿死



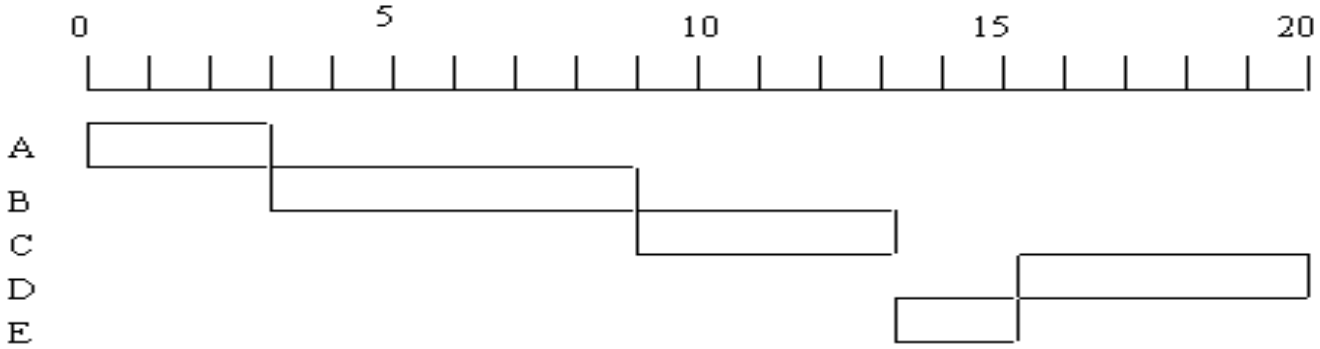
最高响应比优先（HRRN）

进程	到达时间	服务时间
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

- 非剥夺式
- 选取响应比R最大的进程
  - w：等待CPU的时间
  - s：估计执行时间

$$RR = \frac{w + s}{s}$$

- 利短进程，但长进程也不会饿死
- 同SPN、SRT，要估计执行时间



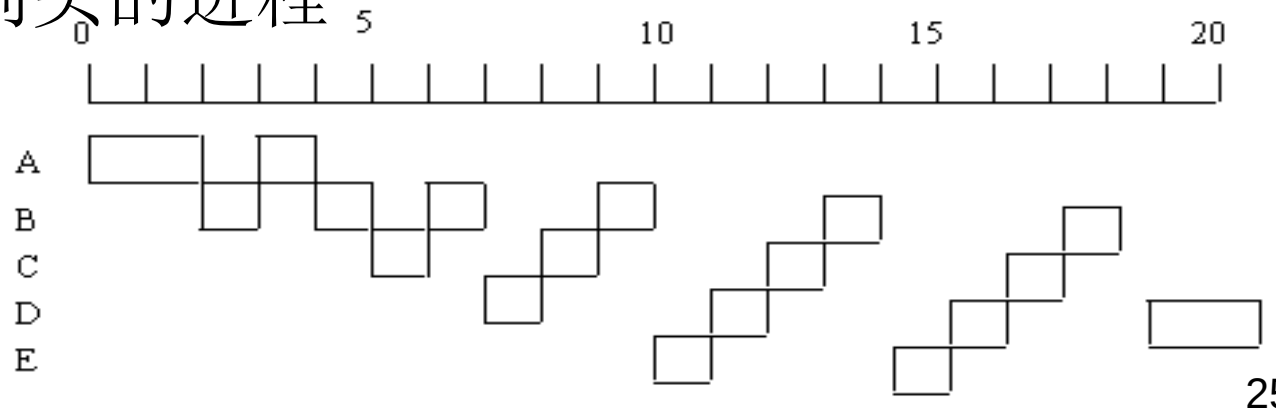
时间片轮转（RR）

进程	到达时间	服务时间
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

- 剥夺式的FCFS
  - 把CPU划分成若干时间片，

按顺序分配给就绪列队中的各个进程

  - 时间片用完时，系统剥夺该进程的CPU，将该进程排列到就绪队列的末尾
  - 运行就绪队列头的进程

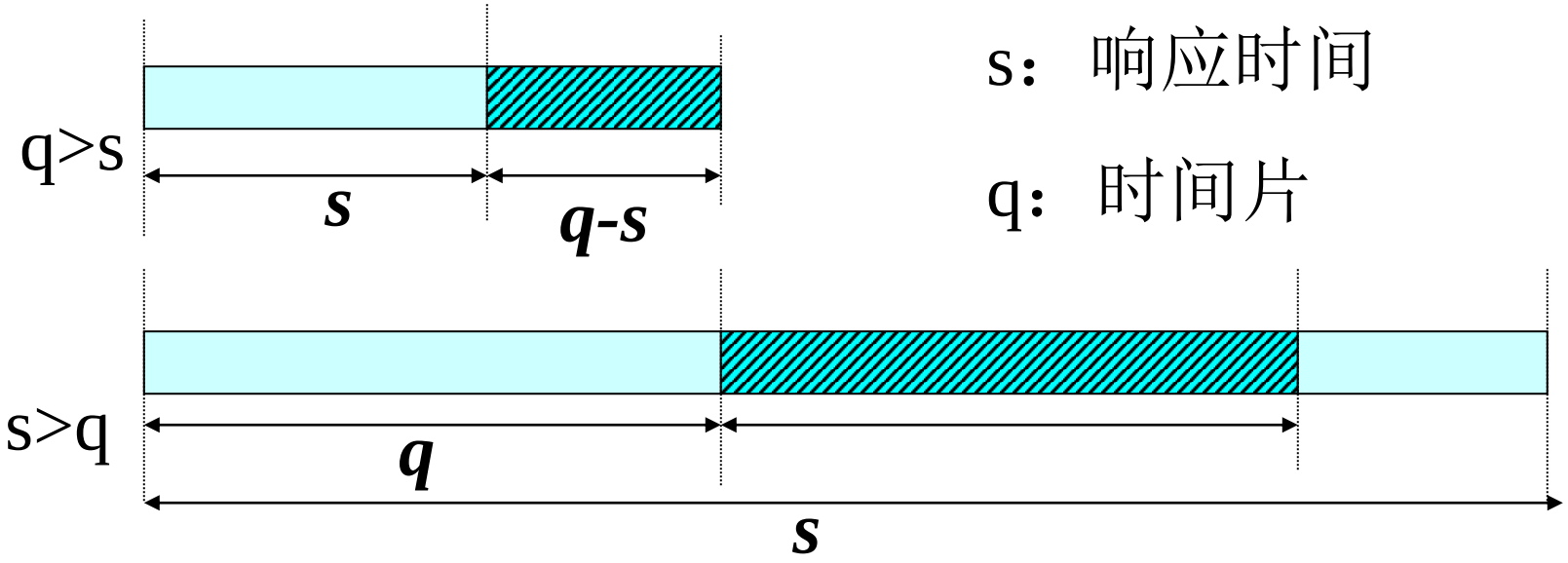


时间片轮转（RR）（续）

- 时间片（time slice/quantum）长度的选择问题：
  - 太短：切换开销大
  - 太长：响应时间变长
  - 一般比典型的一次交互过程时间略长
- 分时系统和事务处理系统中常用时间片轮转法
- 有利于CPU密集进程，不利于I/O密集进程

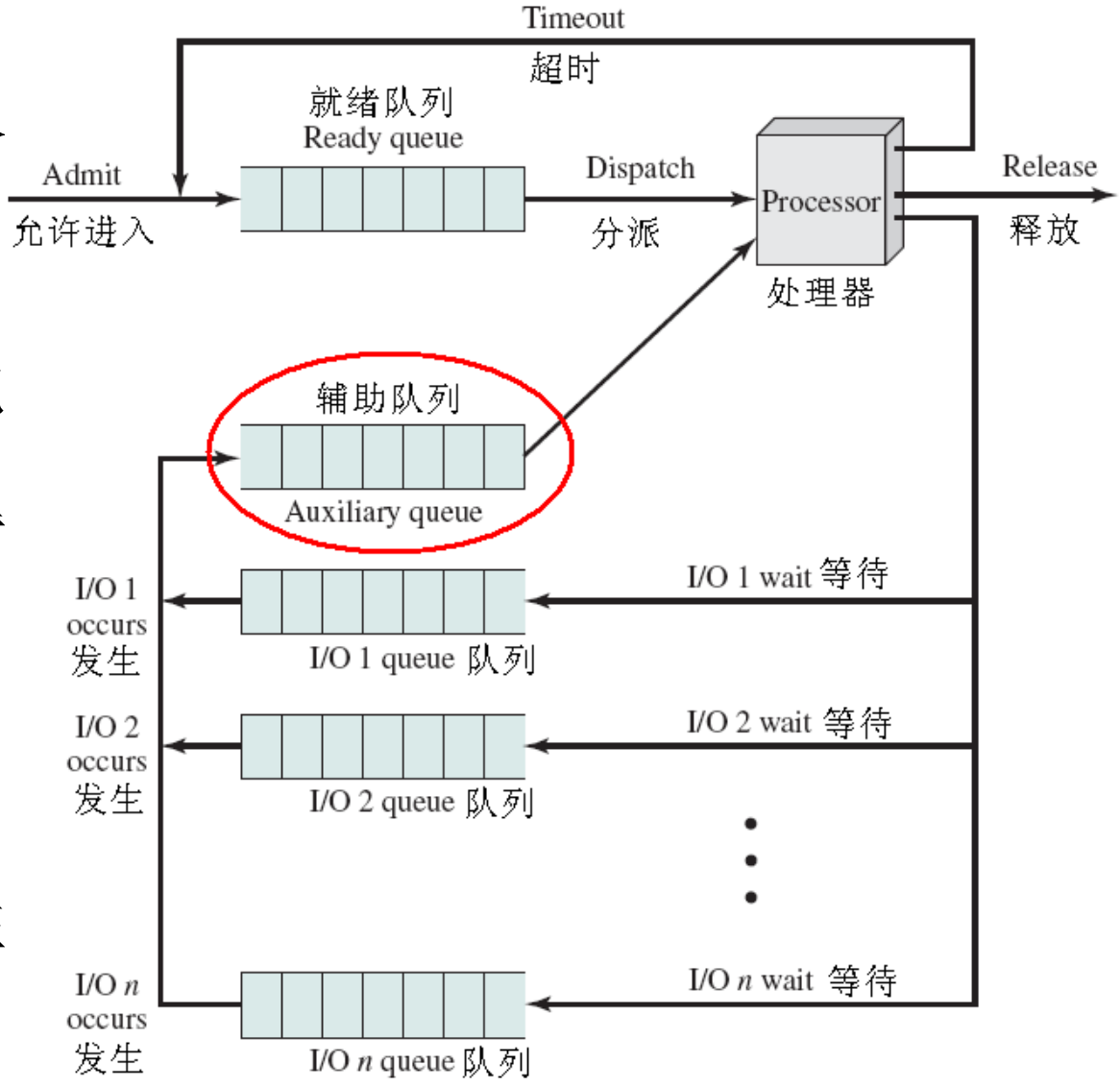
时间片轮转（RR）（续）

- 时间片（time slice/quantum）长度的选择
- 分时系统和事务处理系统中常用时间片轮转法
- 有利于CPU密集进程，不利于I/O密集进程



虚拟轮转VRR（Virtual Round Robin）

- 阻塞解除的进程进入辅助队列
- 辅助队列中的进程比就绪队列的优先获得处理器
- 可解决轮转对I/O密集型进程的不公平性问题



Queueing Diagram for Virtual Round-Robin Scheduler

虚拟轮转调度的队列图

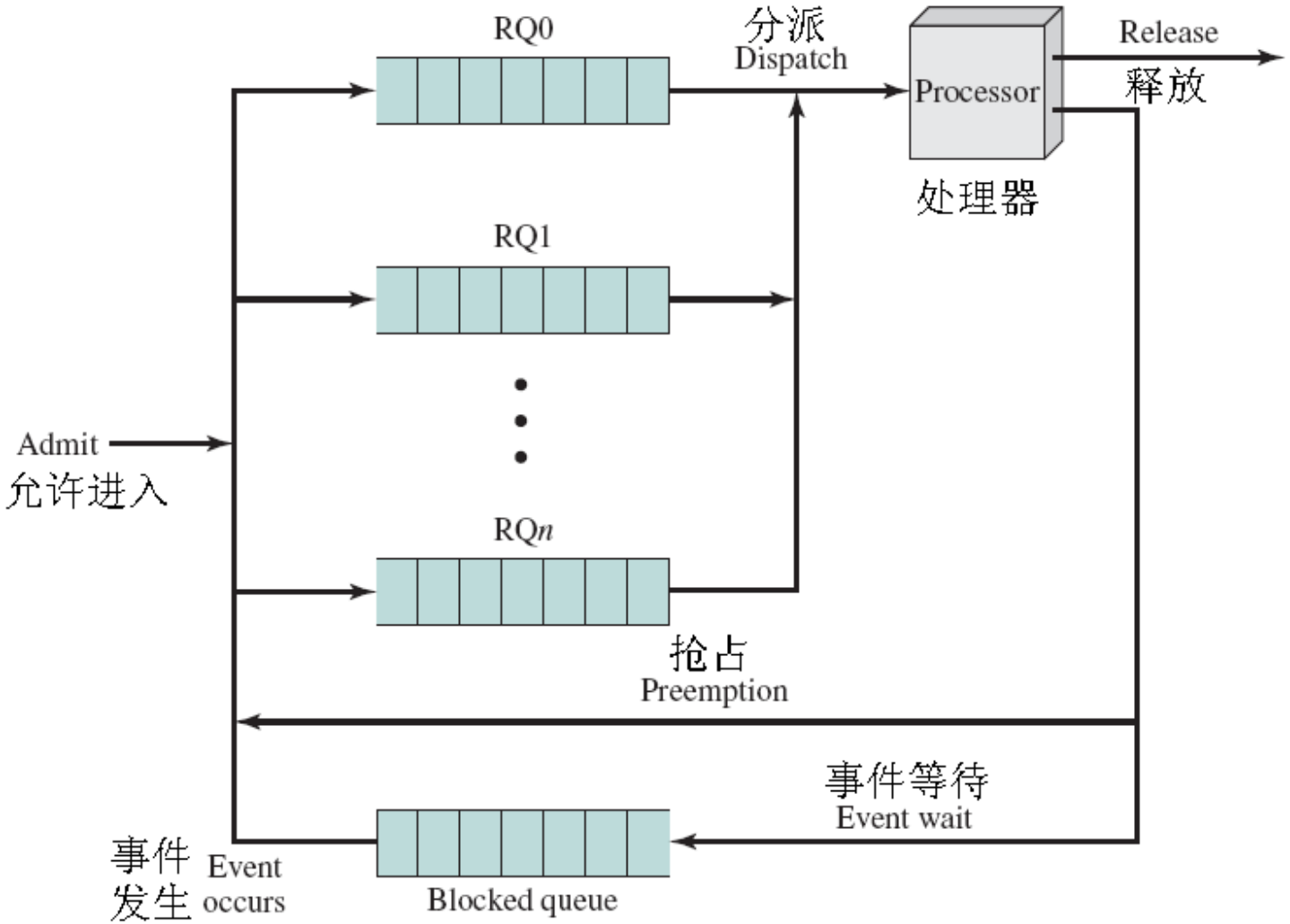
最高优先级优先（HPF）

- 每个进程被赋予一个优先级（priority）
- 每次调度选取就绪队列中优先级最高的进程

最高优先级优先（HPF）

- 优先级确定方法：
  - 静态：进程创建时指定优先级，在进程运行时优先级保持不变
  - 动态：在进程创建时指定一个优先级，但在其生命周期内优先级可以动态变化（如等待时间长的优先级可提高，时间片过后的优先级可降低 ）
- 实现时可对应不同优先级采用多个就绪队列

优先级排队



Priority Queueing

优先级排队

多级队列反馈（MF/FB）

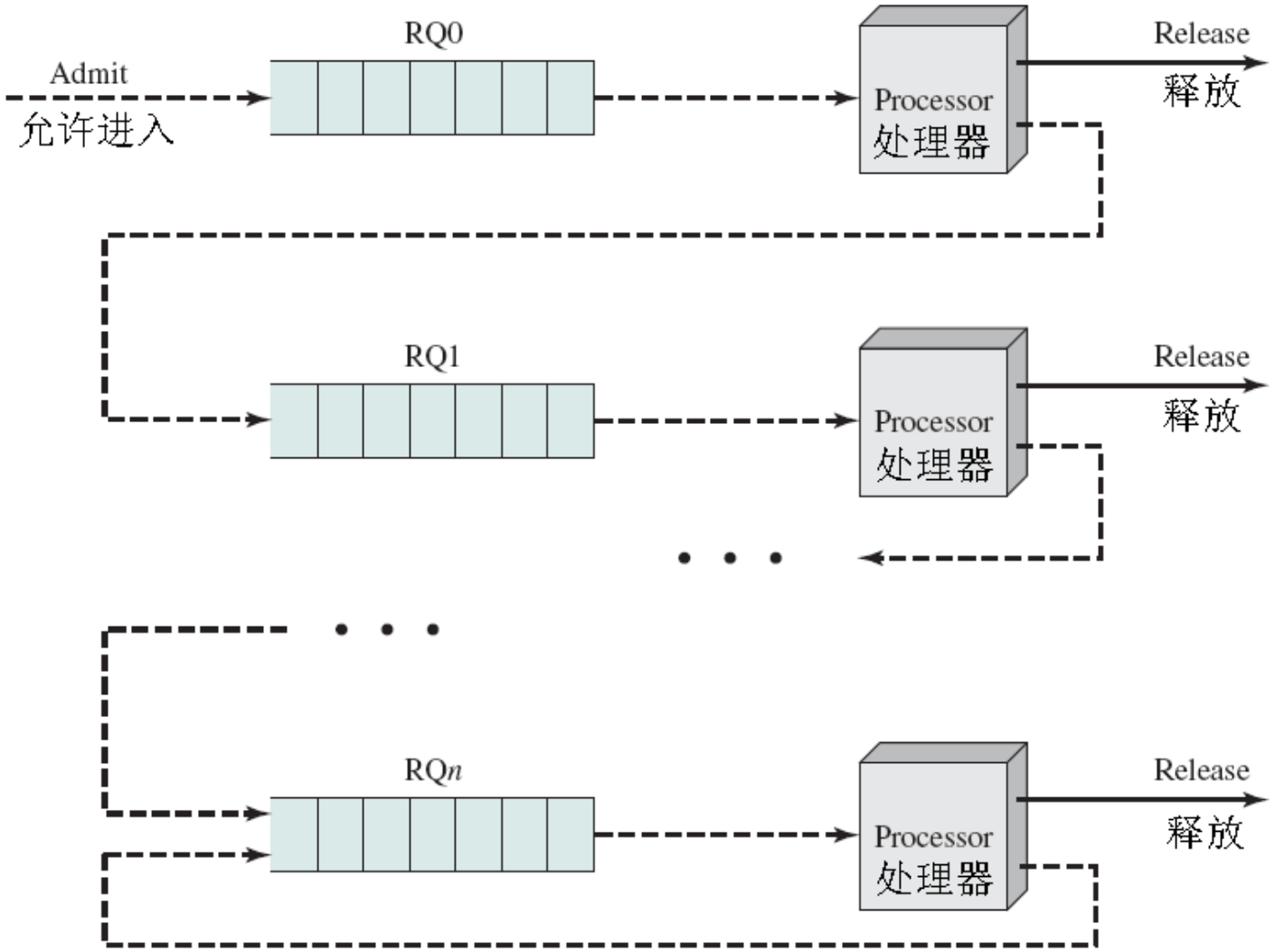
- 剥夺式、时间片
- 关注进程已执行的时间
- 不用估计（剩余）执行时间

多级队列反馈（MF/FB）

动态优先级

- 设立多个优先级就绪队列，各个队列运行时间片可能不同（优先级越高[i越小]时间片越小：2^i）
- 新的就绪进程进入最高优先级队列
- 进程由于时间片用完被抢占而放弃CPU，下降一个优先级队列（最高 -> 最低）
- 进程由于等待而放弃CPU后，进入等待队列，一旦等待的事件发生，则回到原来的就绪队列
- 在各优先级队列中采用FCFS（最低级队列中采用RR）

多级队列反馈（MF）续

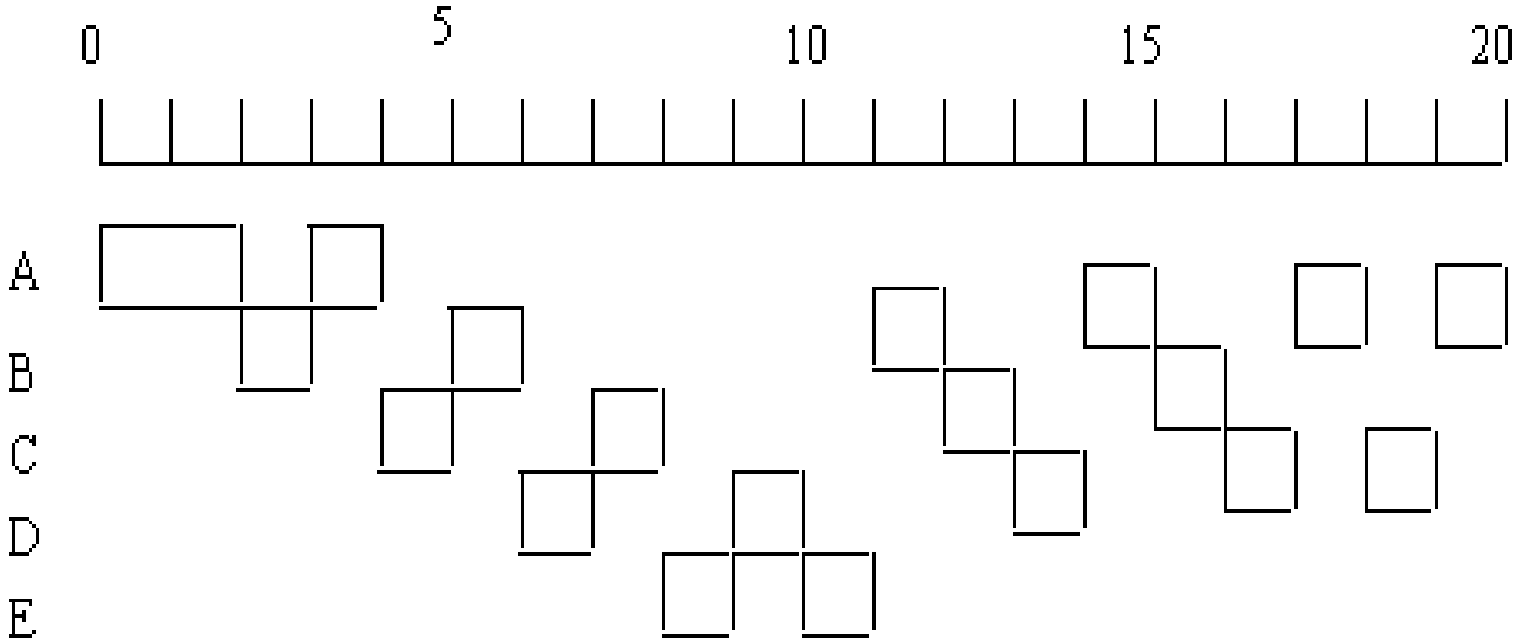


Feedback Scheduling

反馈调度

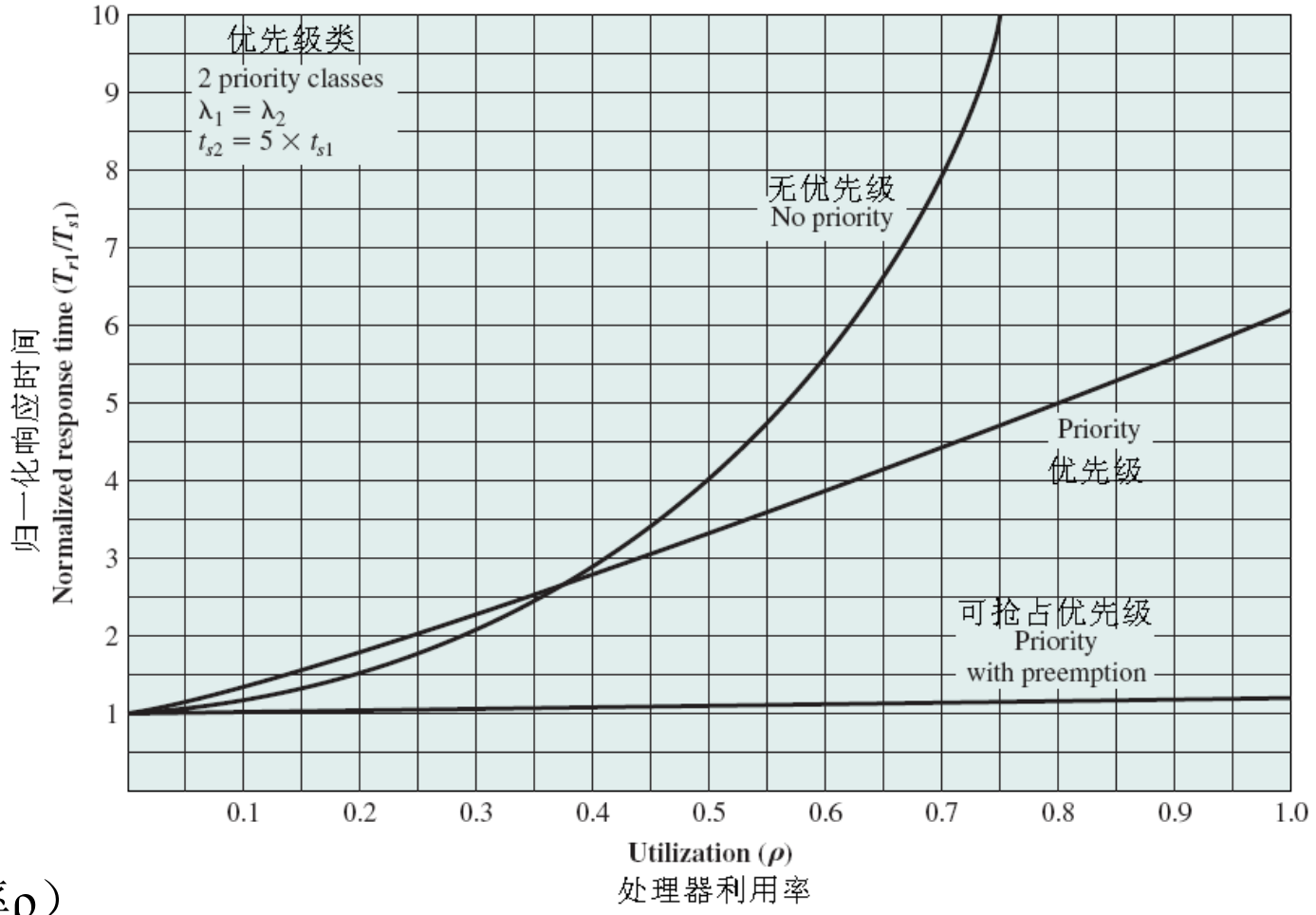
多级队列反馈（MF）续

- 调度：先按FCFS从最高优先级队列中选取，若最高优先级队列为空，按FCFS从次高优先级队列选取……



9.2.4 性能比较——响应时间

- 短进程(高优先级)的标准化响应时间



Normalized Response Time for Shorter Processes

短进程的归一化响应时间

归一化响应时间

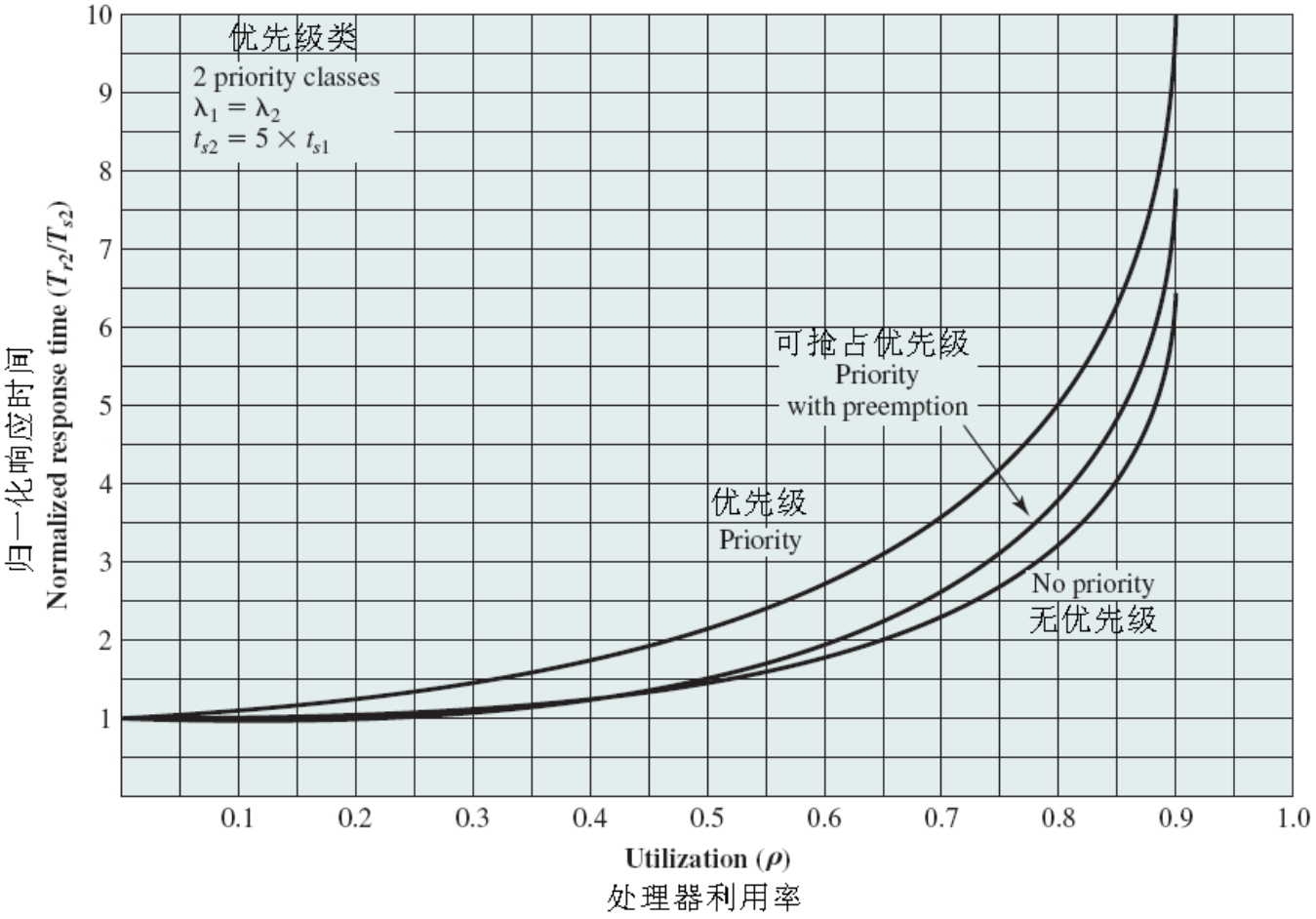
= 周转时间Tr

/平均服务时间Ts

= 1/（1-处理器利用率ρ）

性能比较——响应时间（续）

- 长进程(低优先级)的标准化响应时间



Normalized Response Time for Longer Processes

长进程的归一化响应时间

性能比较——周转时间

- 仿真建模—50,000个仿真进程，按服务时间分成100组，80%CPU利用率（所需时间的百分点~进程服务时间长度）

归一化周转时间

= Tr/Ts

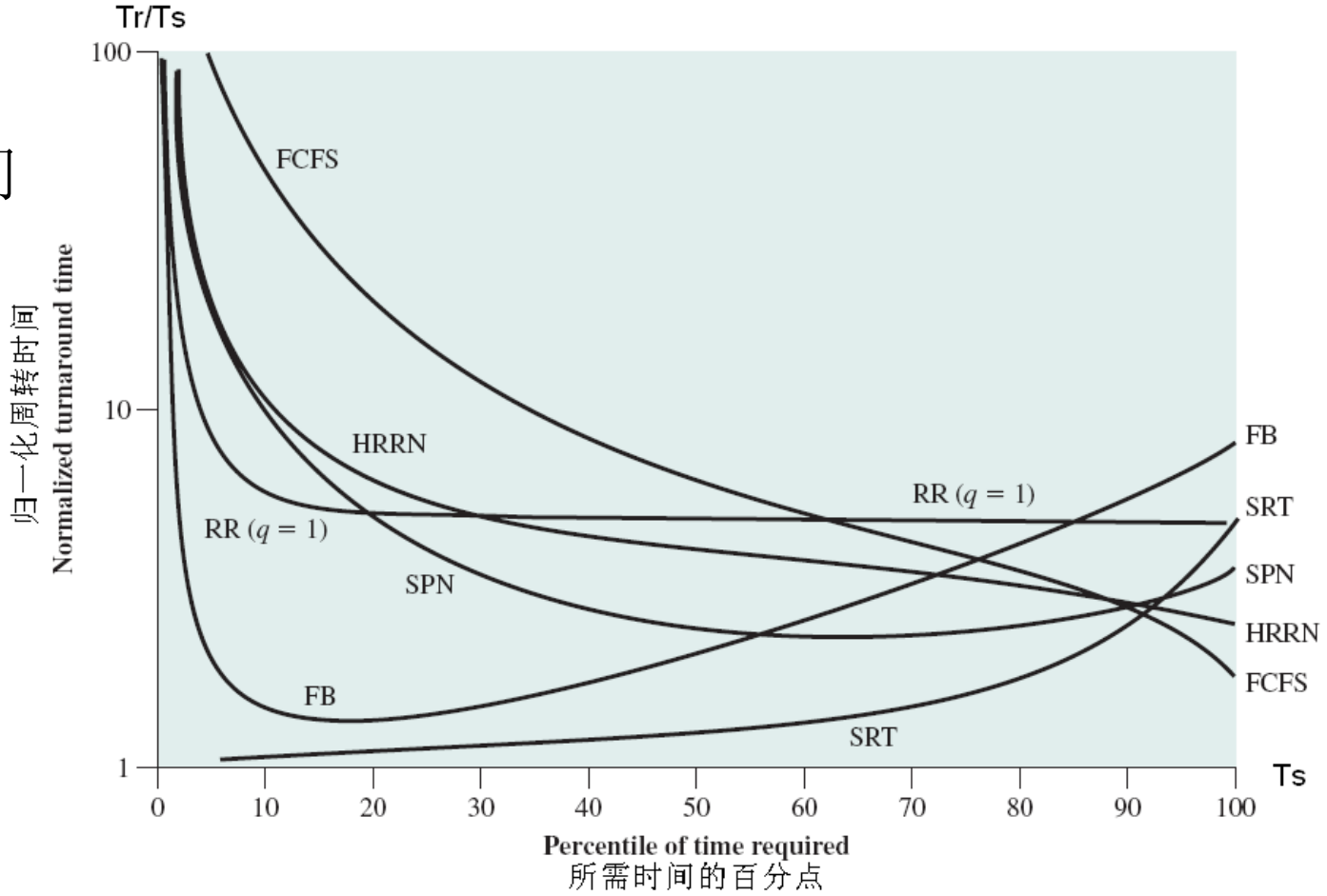
=(Ts+Tw)/Ts

其中：

Tr=周转时间

Ts=服务时间

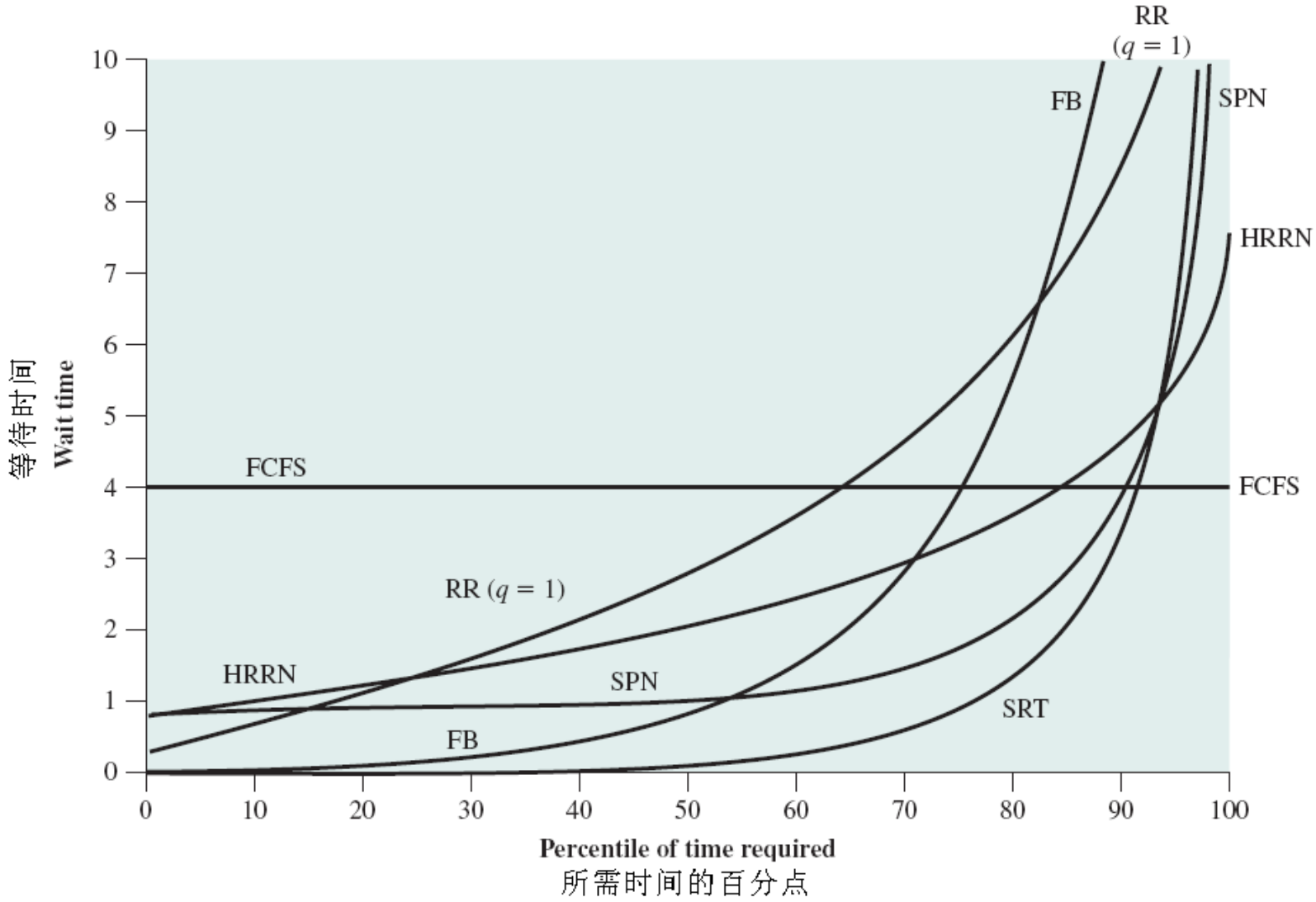
Tw=等待时间



Simulation Result for Normalized Turnaround Time

归一化周转时间的仿真结果

性能比较——等待时间



Simulation Result for Waiting Time

等待时间的仿真结果

9.2.5 公平共享调度

- 用户应用或作业可以是一个进程（或线程）集合
- 用户关心的是应用或作业的整体性能
- 应该基于进程集合进行调度决策

9.2.5 公平共享调度

- 一个用户也可看成是一个用户组的成员
- 同一用户组的用户应只影响本用户组的调度而不影响其他用户组的调度，即应基于用户组进行调度决策
- 公平共享调度（Fair-Share Scheduling，FSS）
  - 基于组调度，每个组公平共享处理器时间
  - 每个用户被指定某种类型的权值，以定义其使用共享资源的份额

公平共享调度示例

进程j在时间段i开始处的

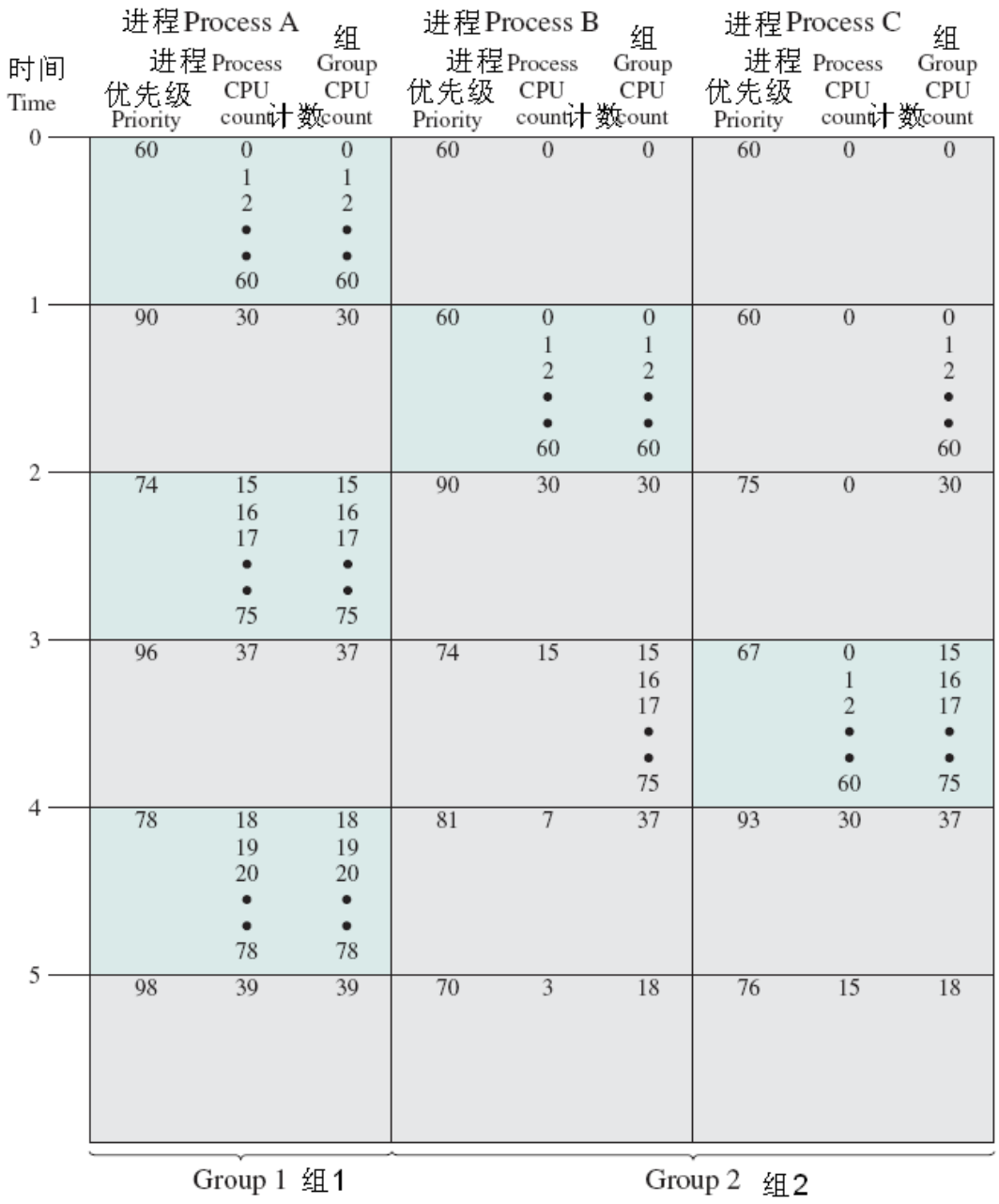
优先级：

$P_j(i) = Base_j + CPU_j(i)/2$

$+ GCPU_j(i)/4W_k$

其中：

- 进程j的基础优先级$Base_j$=60
- 进程j在时间段i中处理器使用计数$CPU_j(i)= CPU_j(i-1)/2$
- 组j在时间段i中处理器使用计数$GCPU_j(i)= GCPU_j(i-1)/2$
- 分配给组k的权值W_k=0.5



Colored rectangle represents executing process 彩色矩形表示正在执行的进程

Example of Fair-Share Scheduler—Three Processes, Two Groups

公平共享调度例——三个进程、两个组

问答

9.3 传统Unix调度

- Unix SVR3、4.3BSD Unix
- 分时交互系统
- 多级队列反馈，其中每个优先级队列采用RR
- 优先级一秒重新计算一次
- 优先级基于进程类型和执行历史

$$P_j(i) = Base_j + CPU_j(i)/2 + nice_j$$

$$CPU_j(i) = CPU_j(i-1)/2$$

  - 基本优先级Base取决于进程类型所属的优先级固定区间，以下区间优先级依次递减：交换程序、块 I/O设备控制、文件操作、字符I/O操作、用户进程
  - 调整因子nice用于保证进程优先级不超出其区间范围

传统Unix的进程调度

- 调度由0号进程完成：始终在核心态执行，与其他进程并不完全一样。采用时间片、多级反馈队列、核心优先级、用户优先级
- 时机：
  - 进程由核心态转入用户态时：在每次执行核心代码之后返回用户态之前，检查各就绪进程的优先级并进行调度。如中断——进程回到就绪队列
  - 进程主动放弃处理机时：进程申请系统资源而未得到满足（如read），或进行进程间同步而暂停（如wait或pause）
  - 进程退出（如exit）——进程进入阻塞队列或exit状态

用户优先级

- 进程在用户态和核心态的优先级是不同的，这里说的是用户态进程的优先级。它是基于执行时间的动态优先级，进程优先级可为0~127之间的任一整数（共128个）。优先数越大，优先级越低
  - 0~49之间的优先级（共50个）为系统内核保留
  - 用户态下的进程优先级为50~127之间（共78个）
- 在Unix System V中，进程优先数：

  P_pri = P_CPU / 2 + PUSER + P_nice + NZERO
- ……