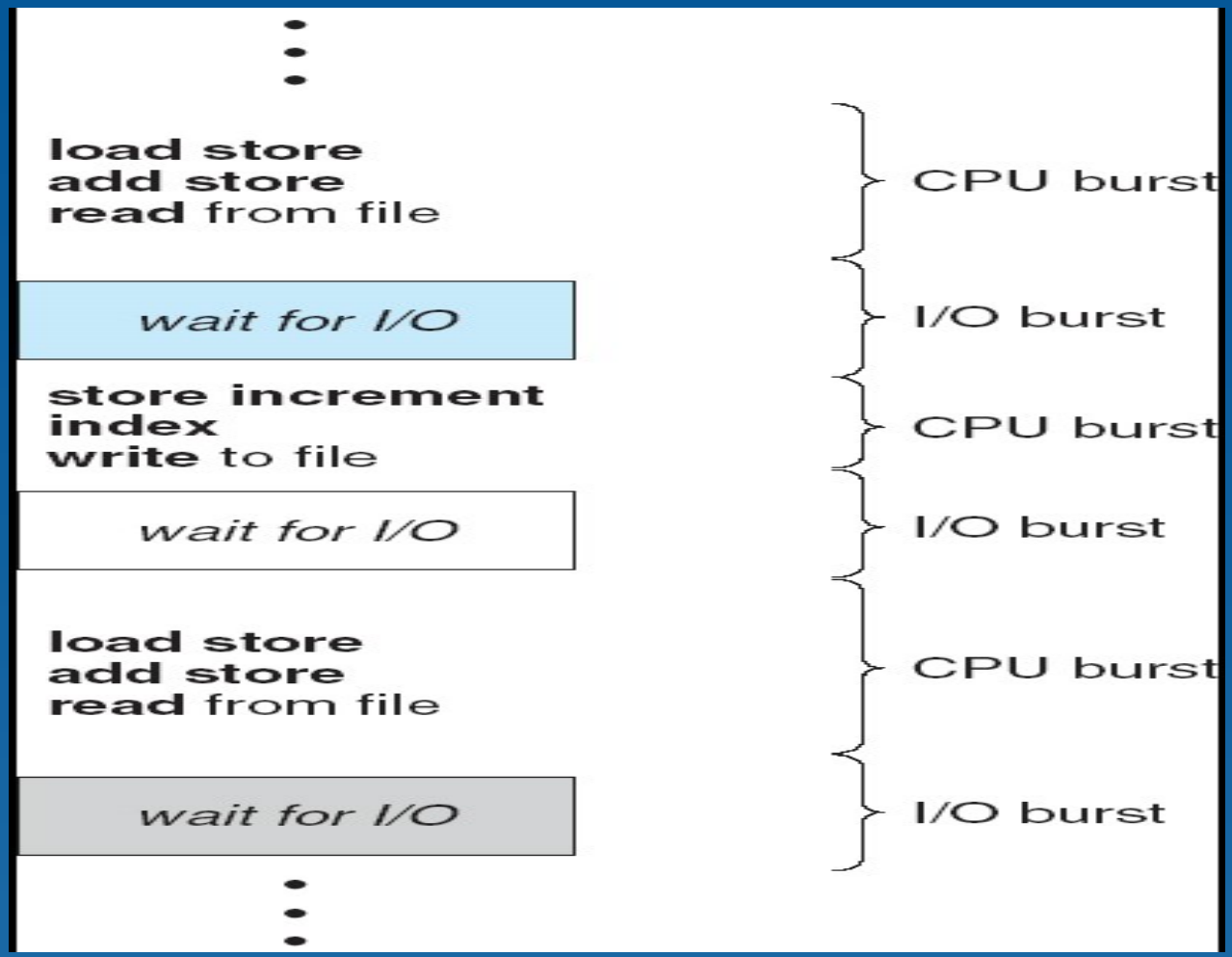
Cpu 调度 (Scheduling)

相关基本概念

- 引入多程序设计，目的是提高计算机资源利用率，尤其是 CPU 利用率 (CPU utilization)
- CPU 密集 – I/O 密集的循环
- 进程的执行，呈现出 CPU 运行和 I/O 等待的交替循环
- CPU 密集型， I/O 密集型

CPU 运行和 I/O 等待的交替循环

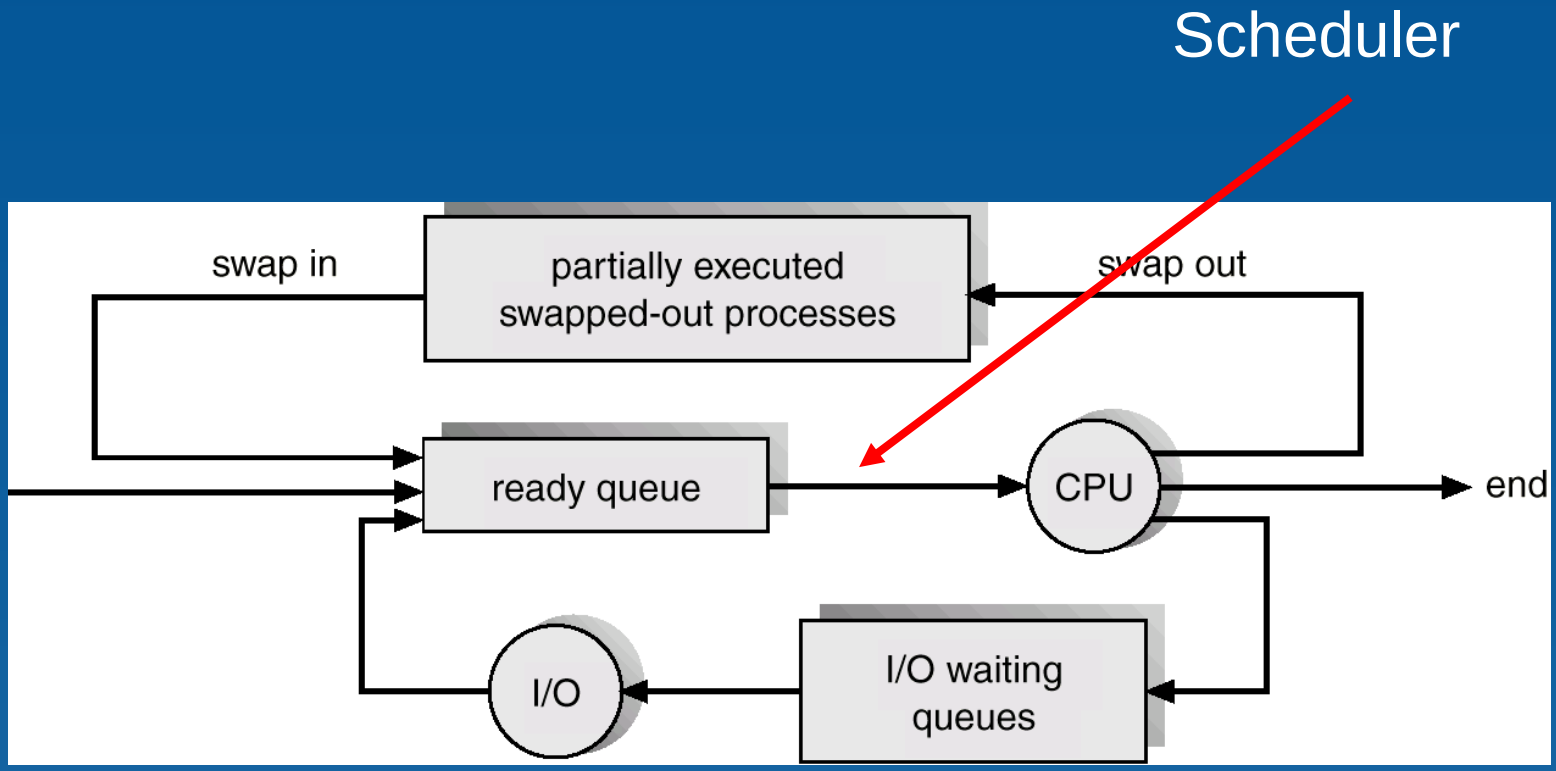


CPU 调度器 (Scheduler)

- CPU 调度器的使命
- 从内存中一堆准备就绪的进程中（就绪队列中的就绪进程），选取一个进程；
- 将 CPU 分配给该进程。

- 后者也可以由 dispatcher 完成，见后面讨论

CPU 调度器的操作对象



CPU 调度器的操作时机

- 调用 CPU 调度器的时机，通常发生在:
  - 1. 某一进程从执行状态转为等待状态
  - 2. 某一进程从执行状态转为就绪状态
  - 3. 某一进程从等待状态转为就绪状态
  - 4. 某一进程终止
- 注意，调度时机不限于此 4 种情况。例如?
- 第 1 种情形和第 4 种情形称作 “非抢占式” (*nonpreemptive*) 调度
- 第 1 种情形和第 4 种情形称作 “抢占式” (*preemptive*) 调度

CPU 分配器 (Dispatcher)

- CPU 调度器决定了将 CPU 分配给谁。后续操作就是， CPU 分配器将 CPU 控制权移交给该进程。操作内容通常包括:
  - 上下文切换 (switching context)
  - 从内核态 (kernel mode) 转移至用户态 (user mode)
  - 跳转至用户程序中 PC 寄存器所指示的位置
- *分配延迟 (Dispatch latency)* – CPU 分配器暂停前一进程，启动后一进程所经历的时间

CPU 调度器追求指标

- CPU 利用率 (CPU utilization)
- 吞吐率 (Throughput) – 单位时间内完成执行的进程数
- 周转时间 (Turnaround time) – 执行某一进程所耗用的 CPU 累积时间
- 等待时间 (Waiting time) – 某一进程等待在就绪队列里面的累积时间
- 响应时间 (Response time) – 某一进程从发出调度请求，到其得到 CPU 调度器响应，其间所经历的时间

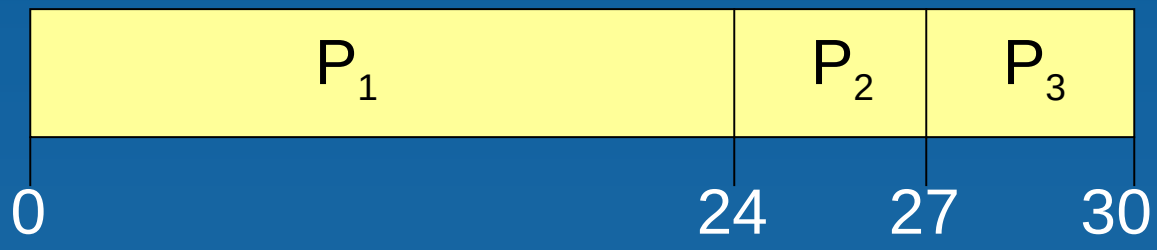
优异的指标，当然是

- Maximize CPU utilization
- Maximize throughput
- Minimize turnaround time
- Minimize waiting time
- Minimize response time

First-Come, First-Served (FCFS) Scheduling

Process	Burst Time
P_1	24
P_2	3
P_3	3

- 假设进程到达就绪队列的顺序： P_1 , P_2 , P_3
  FCFS 调度算法的调度结果如甘特图 (Gantt Chart) :



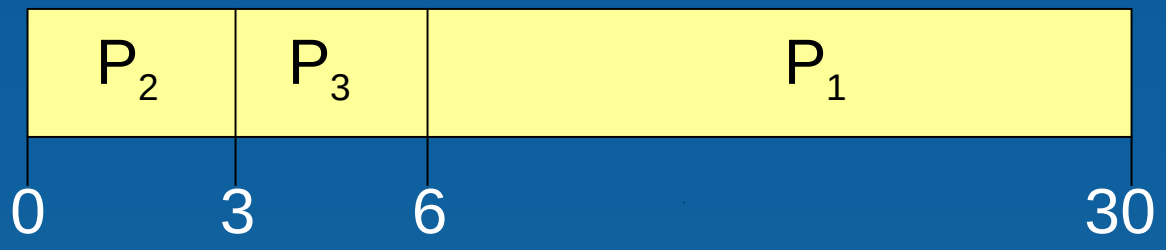
- 等待时间 (Waiting time) P_1 = 0; P_2 = 24; P_3 = 27
- 平均等待时间： (0 + 24 + 27)/3 = 17

FCFS 调度算法（续）

假设进程到达就绪队列的顺序：

$$P_2, P_3, P_1$$

- FCFS 调度算法的调度结果有显著变化，如甘特图：



等待时间： $P_1 = 6; P_2 = 0, P_3 = 3$

平均等待时间： (6 + 0 + 3)/3 = 3 ，改善非常多！

- 启示：短进程先于长进程，会得到意外效果

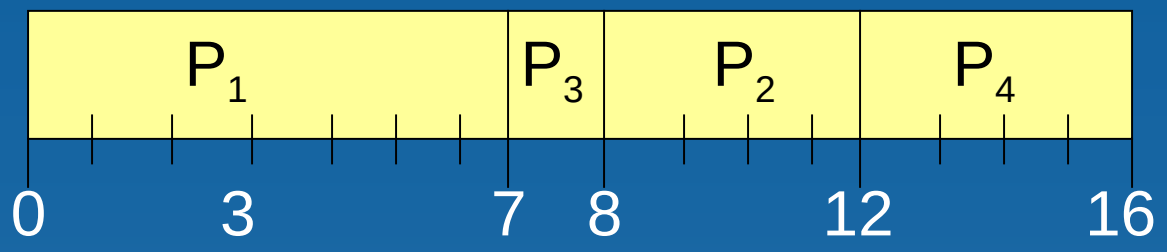
Shortest-Job-First (SJF) 调度算法

- 算法要求:
- 进入就绪队列的进程预告需要多长 CPU 时间才能完成本次执行
- 算法思想:
- 选取就绪队列中，“需要 CPU 时间”最短的进程

举例：非抢占式 (Non-Preemptive) SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (非抢占式)

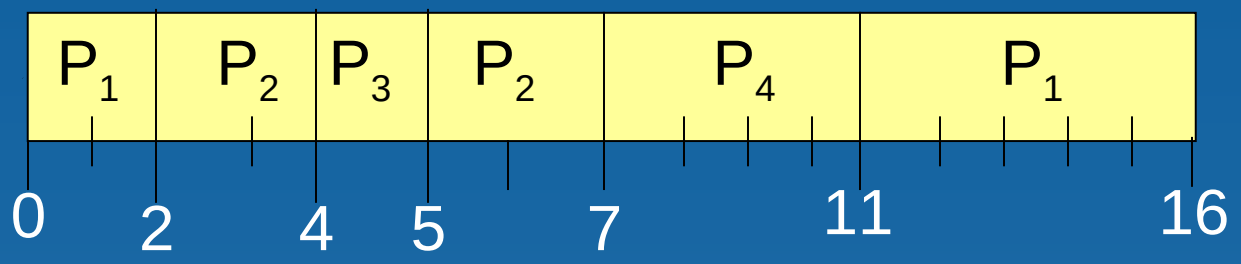


- 平均等待时间 = (0 + 6 + 3 + 7)/4 = 4

举例：抢占式 (Preemptive) SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (抢占式)



- 平均等待时间 = (9 + 1 + 0 +2)/4 = 3

两种 SJF 策略比较

- 非抢占式 (Non-Preemptive) – 一旦 CPU 分配给了某个进程，就不能“抢过来”，除非该进程主动放弃 CPU （CPU burst cycle 结束，或者进程转去做 I/O 操作）
- 抢占式 (Preemptive) – 上述描述的“非”
- 当一个进程进入就绪队列，如果它的 CPU 时间小于当前拥有 CPU 的进程的剩余“预估”时间，前者抢占后者的 CPU 。此算法称作 Shortest-Remaining-Time-First (SRTF)

关于 SJF 算法的结论

- SJF 是最优算法 – why
- SJF 算法有致命缺陷 – To Be Cont

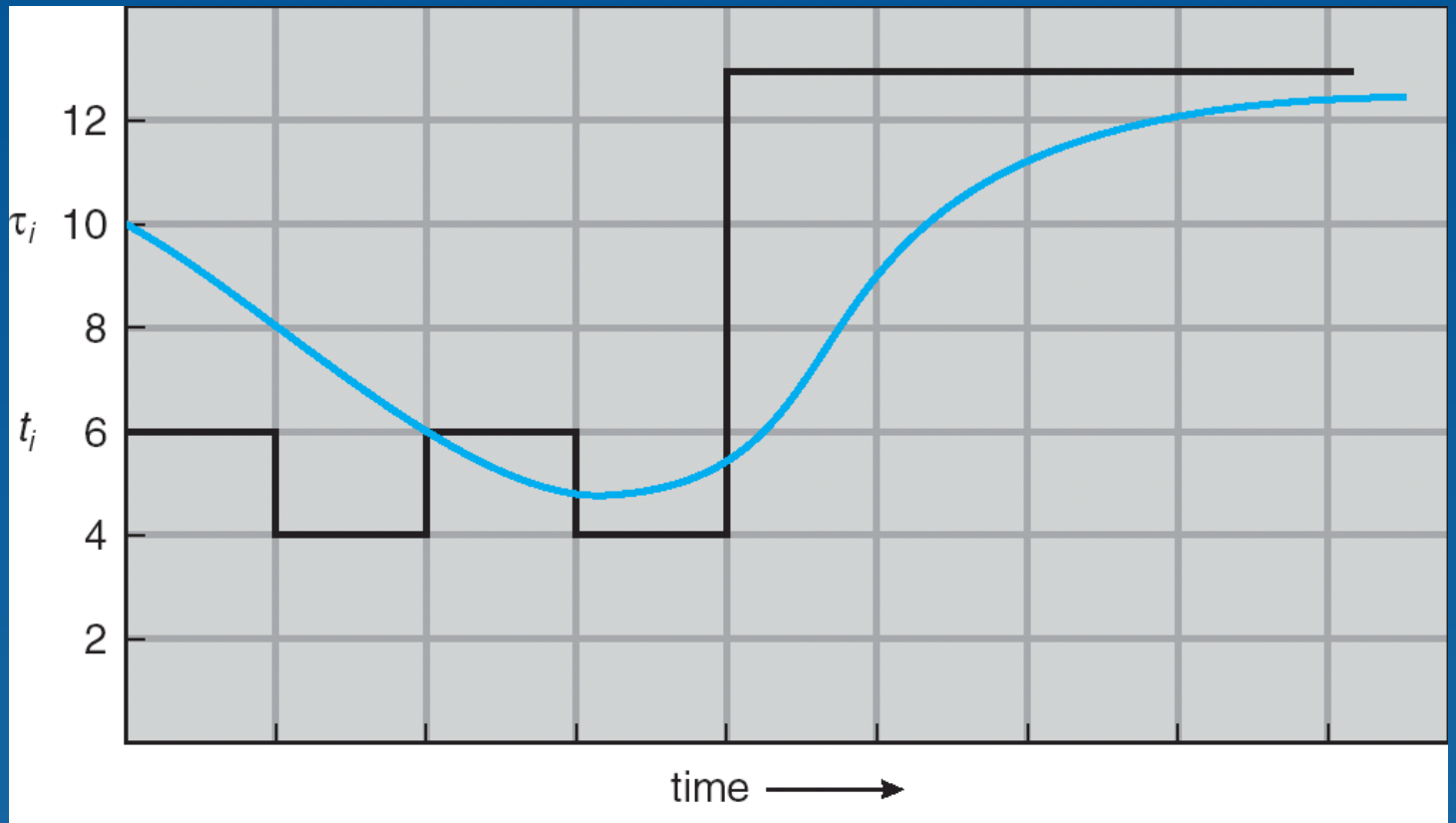
进入就绪队列的进程怎么“预估”CPU时间？

- 不可能准确地预测，为什么？ (e.g. 等待输入数据)
- 只能根据过去的 CPU burst cycles 拟合
- 例如，指数平均 Exponential average 思想

1. t_n = actual lenght of n^{th} CPU burst
2. τ_{n+1} = predicted value for the next CPU burst
3. $\alpha, 0 \le \alpha \le 1$
4. Define :

$$\tau_{n=1} = \alpha\, t_n + (1-\alpha)\tau_n .$$

图，“指数平均”求进程的下一个 CPU Burst Cycle



CPU burst (t_i)	6	4	6	4	13	13	13	...
"guess" (τ_i)	10	8	6	6	5	9	11	12	...

假如设计一个新算法：HRN (Highest response Ratio Next)

- HRN = (W + T) / T

W 代表等待时间， T 代表预估 CPU 时间。

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

优先权法 (Priority Scheduling)

- 每个进程都有一个优先数 (priority number)，通常是个整型数
- 选取就绪队列中，优先权最高的进程
- （最小优先数 ≡ 最高优先权）
  - Preemptive
  - Nonpreemptive (p163)
- 当优先权定义为进程“需要的 CPU 时间”时，SJF 算法就是优先权法

优先权算法的一个缺陷

- Issue ≡ 进程饥饿 (Starvation) –

  优先权较低的就绪进程可能永远得不到CPU

- Solution ≡ Aging 思想 –

  就绪进程等在就绪队列里的时间，折算叠加到进程优先权。因此，等待在就绪队列里的进程，其优先权单调递增

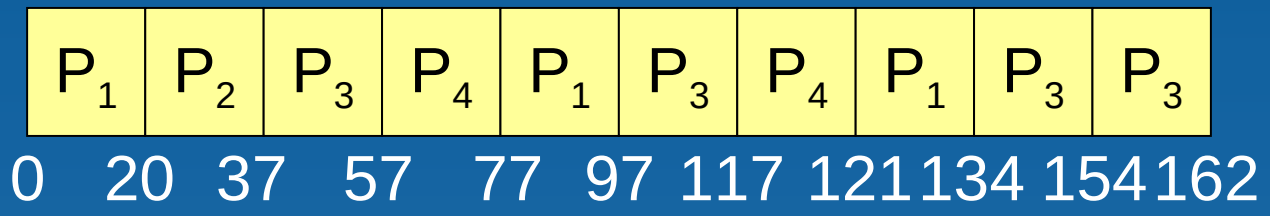
轮转法 (Round Robin， RR)

- 每个就绪进程获得一小段 CPU 时间 (时间片，*time quantum*)，通常 10ms -100ms
- 时间片用毕，这个进程被迫交出 CPU，重新挂回到就绪队列
- 当然，进程在时间片用毕之前其 Burst Cycle 结束，也（主动）交出 CPU
- 假设 n 个就绪进程，时间片 *q*，每个就绪进程得到 1/*n* 的 CPU 时间。任何就绪进程最多等待 (*n*-1)*q* 单位时间

RR 算法举例，时间片设定 20 个单位

Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

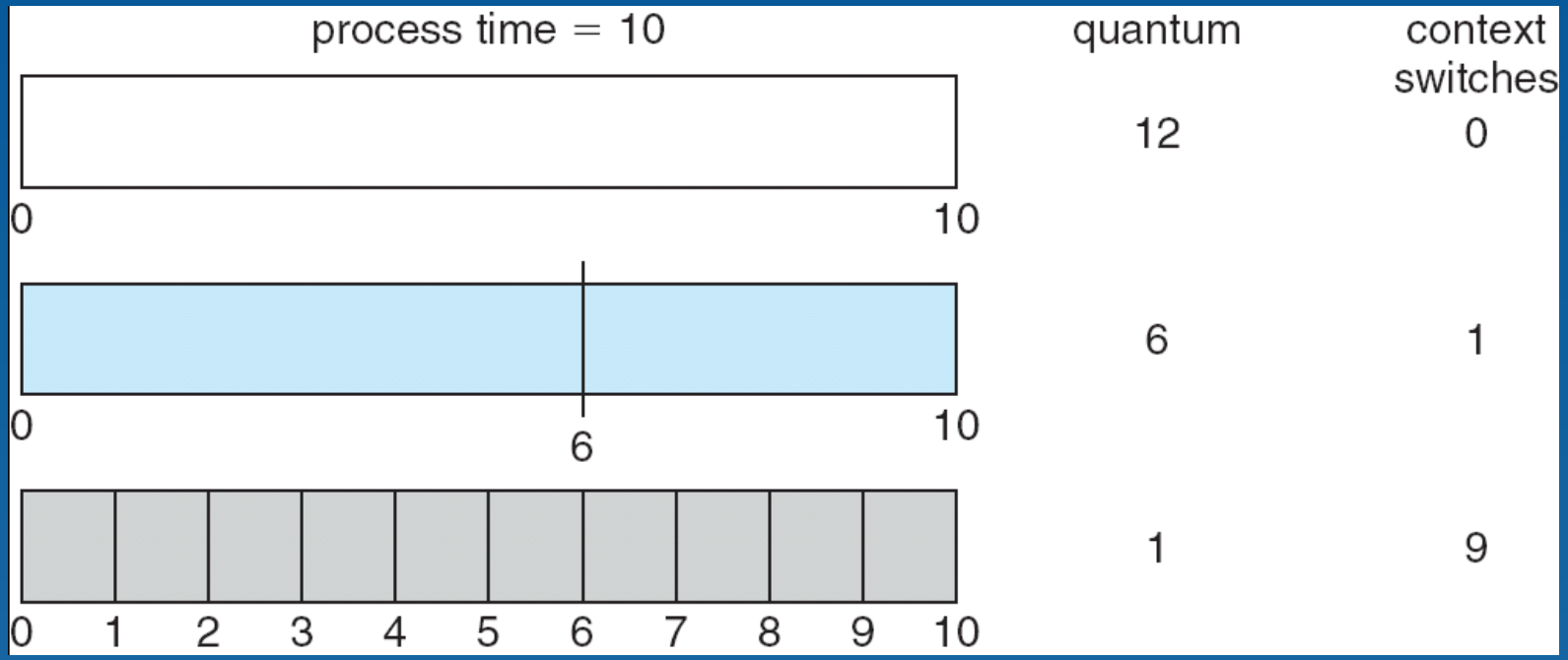
- 甘特图



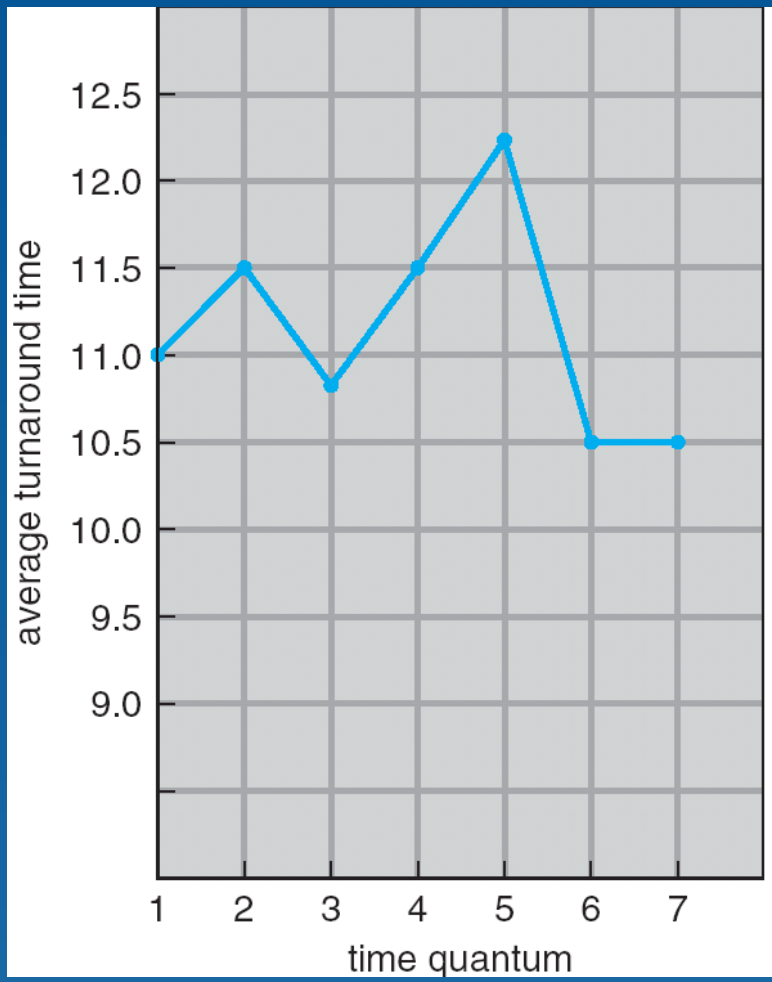
轮转法（续）

- 平均周转时间通常优于 SJF
- 响应时间一定优于 SJF
- 性能分析
  - *q* large ⇒ FIFO
  - *q* small ⇒ 上下文切换开销太大，*q* 必须远远大于上下文切换时间

时间片与上下文切换时间的关系



周转时间受时间片的影响

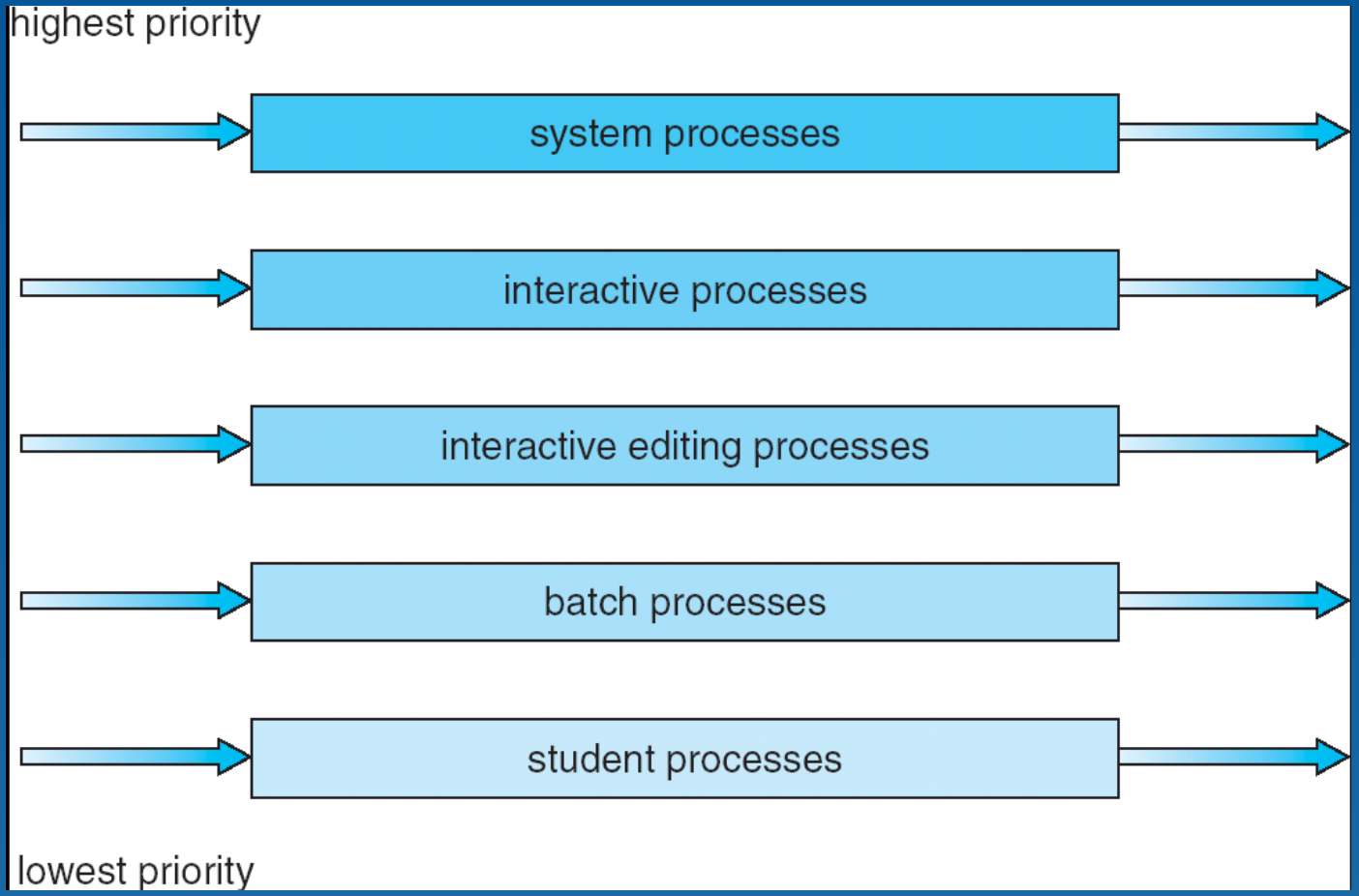


process	time
P_1	6
P_2	3
P_3	1
P_4	7

多层队列 (Multilevel Queue)

- 把就绪队列拆分成几个队列
- 例如:
- 要求交互的进程，在前台队列
- 可以批处理的进程，在后台队列
- 每个队列有其自己的调度算法。例如，
  - 前台就绪队列 – RR
  - 后台就绪队列 – FCFS

多层队列调度示例


highest priority
system processes
interactive processes
interactive editing processes
batch processes
student processes
lowest priority

多层队列（续）

- 就绪进程进入就绪队列时，决定去哪儿？
- CPU 怎么在队列间分配?
  - 固定优先权法。例如，先前台队列，再后台队列。
  - 时间片办法。例如， 80% 的 CPU 时间给前台队列， 20% 的 CPU 时间给后台进程

多层反馈队列 (Multilevel Feedback Queue)

- 基本上类似于多层队列算法
- 另外考虑了，进程可以在就绪队列之间“漂移”

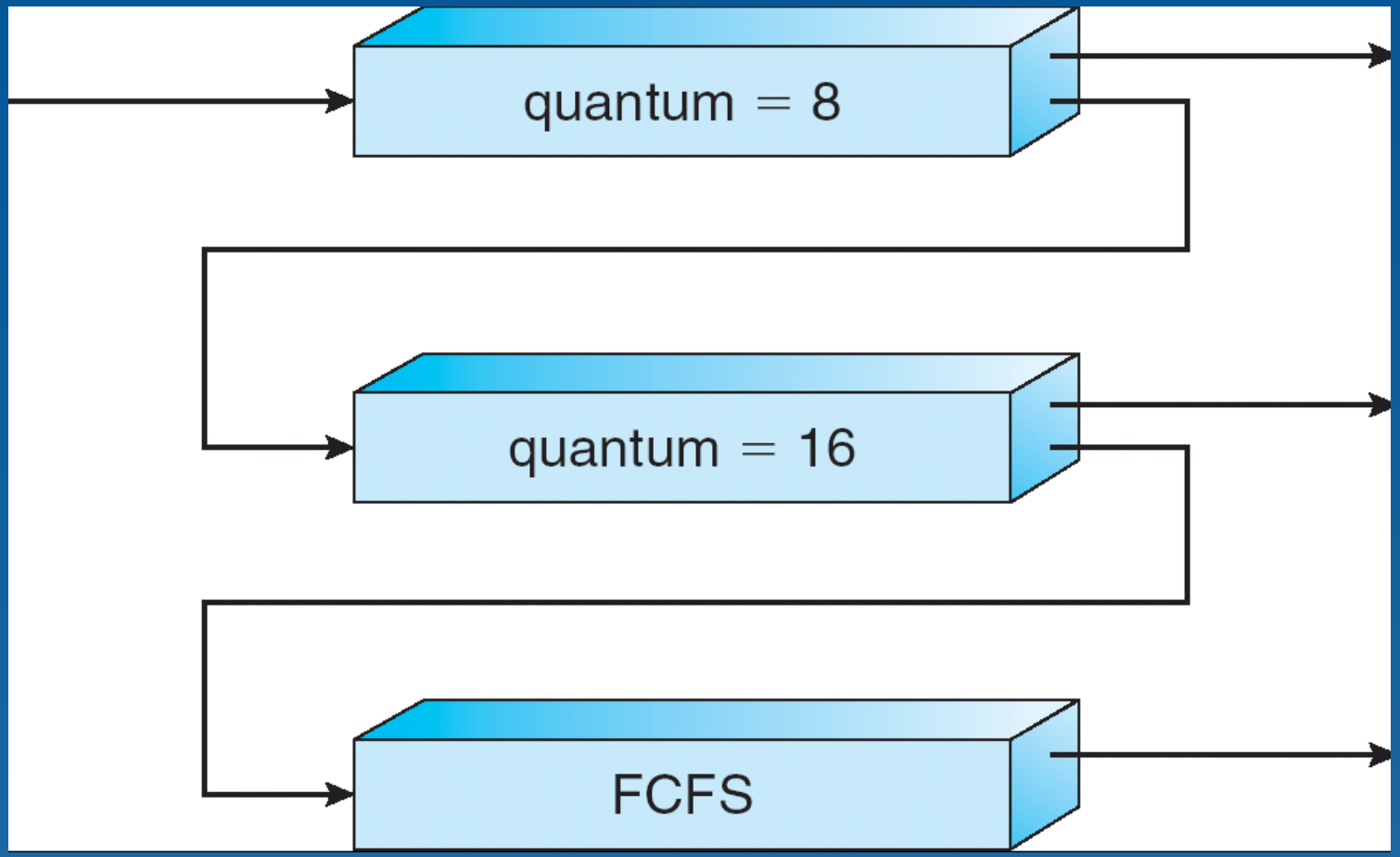
多层反馈队列（续）

- 设计“多层反馈队列”算法应定义如下要素
  - 队列个数
  - 每层队列它自己的调度算法
  - 一个算法，将就绪进程升级至高层次队列
  - 一个算法，将就绪进程降级至低层次队列
  - 一个算法，决定当一个就绪进程进入就绪队列时，去哪层

多层反馈队列示例

- 三层队列
  - Q_0 – 用 RR 算法，时间片 8 ms
  - Q_1 – 用 RR 算法，时间片 16 ms
  - Q_2 – 用 FCFS 算法

多层反馈队列示例（续）



多层反馈队列示例（续）

- 调度场景
  - 一个就绪进程进入 Q_0 层。当它分配到 CPU，可执行 8 ms。如果它 8 ms 后没有执行完毕，则迁移至 Q_1 层。否则，它离开就绪队列，该干嘛干嘛。
  - 在 Q_1 层，当它分配到 CPU，可执行 16 ms。如果它 16 ms 后没有执行完毕，则迁移至 Q_2 层。否则，它离开就绪队列，该干嘛干嘛。

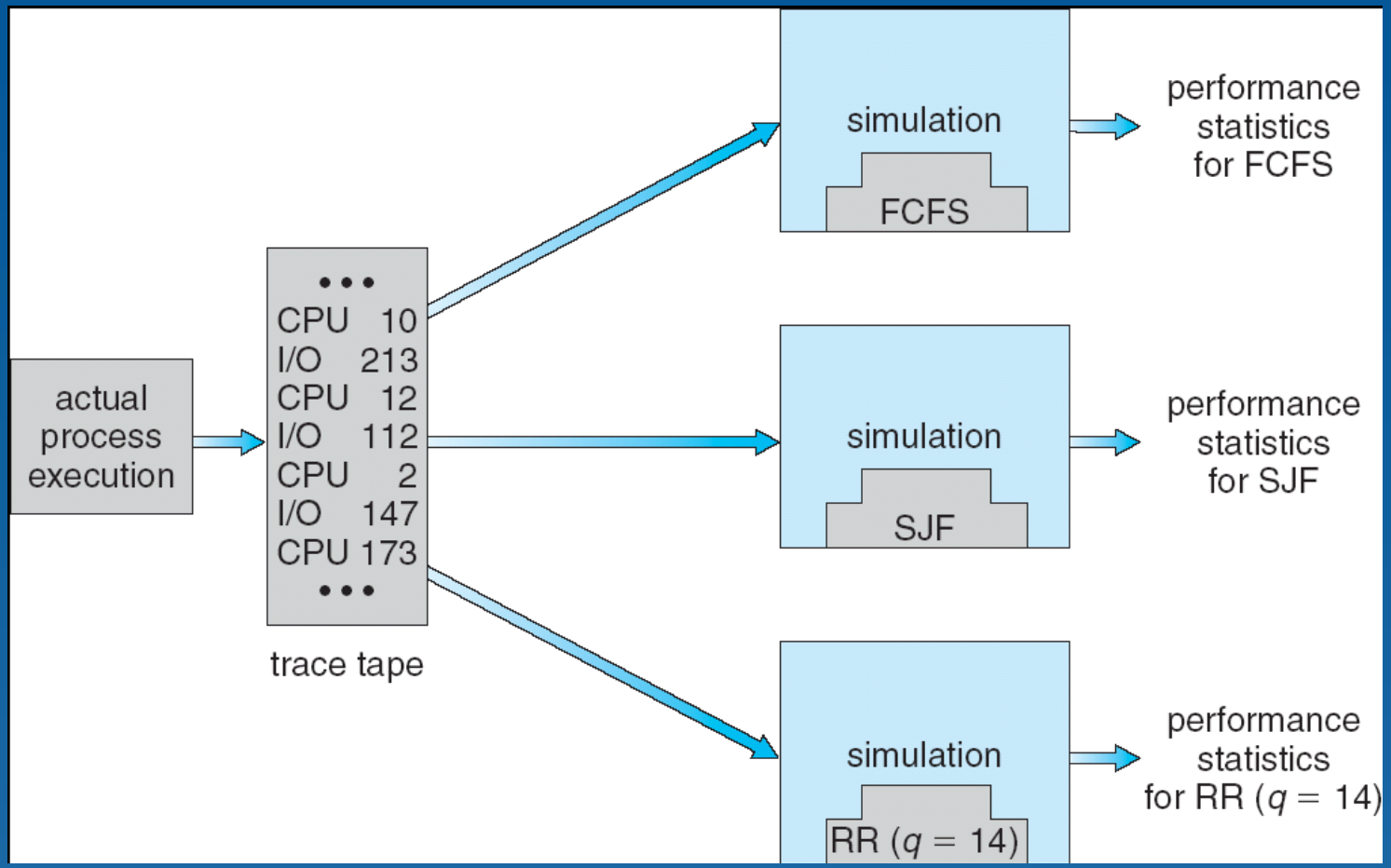
实时调度

- 硬实时系统 – 调度机制能够确保一个关键任务在给定的时间点前完成
- 软实时计算 – 调度机制尽量给予关键任务最高优先级，尽量在预定时间点前完成

调度算法评估

- 确定模型法 (Deterministic modeling) – 采用事先设定的特定负荷，计算在给定负荷下每个算法的性能
- 排队模型 (Queueing models)
- 编程实现该算法，观察其执行情况
- 仿真

仿真


actual process execution
CPU 10
I/O 213
CPU 12
I/O 112
CPU 2
I/O 147
CPU 173
trace tape
simulation
FCFS
performance statistics for FCFS
simulation
SJF
performance statistics for SJF
simulation
RR (q = 14)
performance statistics for RR (q = 14)


END