華中科技大學
计算机科学与技术学院
School of Computer Science & Technology, HUST

操作系统原理

第6章 处理机调度

华中科技大学计算机学院 谢美意

目 录

CONTENT

華中科技大學
计算机科学与技术学院
School of Computer Science & Technology, HUST

处理机的多级调度

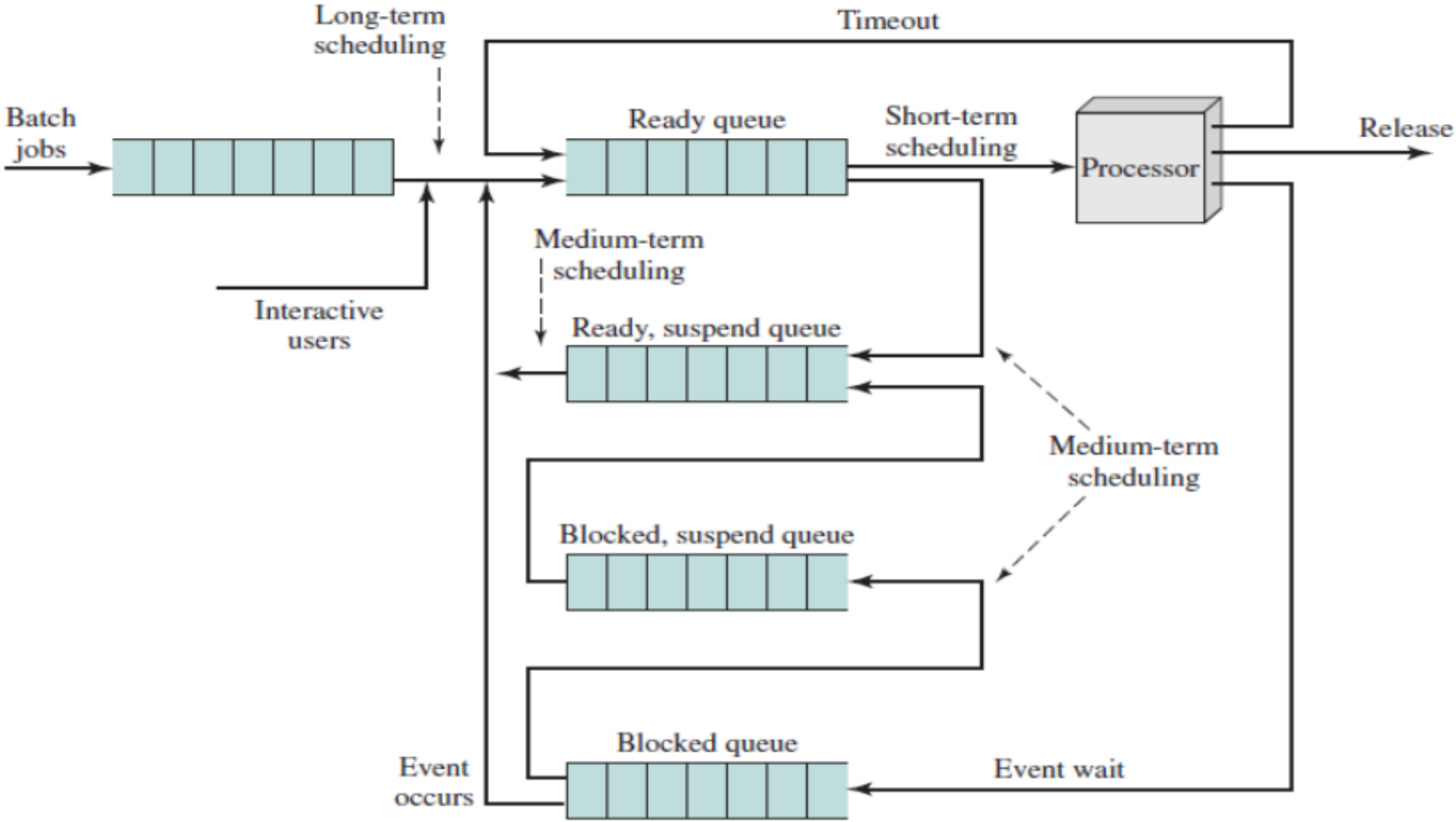
处理机的多级调度

宏观调度

- **长期调度**：决定是否为作业创建新进程，并将其加入将待执行的进程集合中。 **（作业调度）**

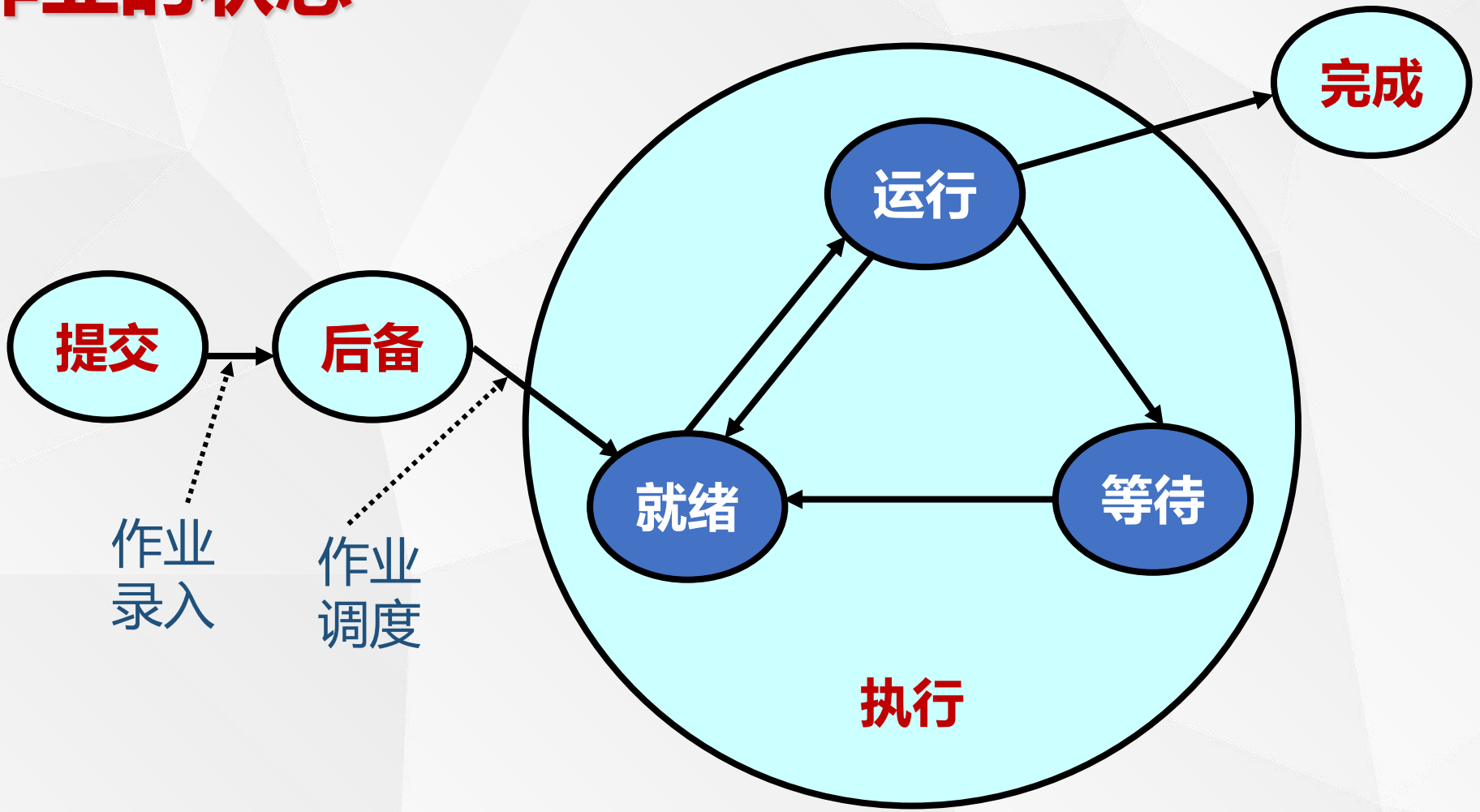
- **中期调度**：决定是否将一个挂起进程加载进内存。（换入调度）
- **短期调度**：决定哪个就绪进程/线程将被处理器执行。 **（进程调度）**

微观调度


Long-term scheduling
Timeout
Batch jobs
Ready queue
Short-term scheduling
Processor
Release
Medium-term scheduling
Interactive users
Ready, suspend queue
Medium-term scheduling
Blocked, suspend queue
Blocked queue
Event occurs
Event wait

作业调度

作业的状态



作业控制块

- **作业名**
- **资源要求**
  - 估计执行时间
  - 最迟完成时间
  - 要求的主存量
  - 要求外设的类型及台数
  - 要求文件量和输出量
- **类型**
  - 控制方式（联机/脱机）
  - 作业类型（计算型/IO型）
- **资源使用情况**
  - 进入系统时间
  - 开始执行时间
  - 已执行时间
  - 主存地址
  - 外设台号
- **优先级**
- **状态**

作业调度的目标

调度实质上是一个策略问题，设定的目标往往是相互冲突的：

- 单位时间内运行尽可能多的作业
- 使处理机尽可能保持“忙碌”
- 使各种I/O设备得以充分利用
- 对所有的作业都是公平合理的

作业调度算法的考虑因素

- 调度算法应与系统设计目标保持一致
- 注意系统资源均衡使用
- 保证提交的作业在截止时间内完成
- 设法缩短作业平均周转时间

作业调度算法性能的衡量指标

1. 周转时间

(1) 定义：一个作业提交给计算机系统到该作业的结果返回给用户所需要的时间。

$t_i = tc_i - ts_i$

t_i—作业i的周转时间

ts_i—作业i的提交时间

tc_i—作业i的完成时间

(2) 意义：说明作业i在系统中停留时间的长短。

(3) 平均周转时间：$t = \frac{1}{n}\sum_{i=1}^{n} ti$

作业调度算法性能的衡量指标

2. 带权周转时间

(1) 定义

一个作业的周转时间与其运行时间的比值。

$w_i = \frac{ti}{tri}$ ，tri为作业的实际运行时间

(2) 意义：说明作业i在系统中相对等待时间。

(3) 平均带权周转时间

$$w = \frac{1}{n}\sum_{i=1}^{n} wi$$

作业调度算法

1. 先来先服务调度算法（FCFS）

(1) 策略：按作业来到的先后次序进行调度。

(2) 特点：

- 优先考虑系统中等待时间最长的作业，不管运行特性。
- 简单，易实现。
- 性能指标不稳定，波动大。

先来先服务调度算法实例

作业	Tsi	Tri	开始时间	Tci	Ti	Wi
1	8.00	2.00	**8.00**	**10.00**	**2.00**	**1**
2	8.50	0.50	**10.00**	**10.50**	**2.00**	**4**
3	9.00	0.10	**10.50**	**10.60**	**1.60**	**16**
4	9.50	0.20	**10.60**	**10.80**	**1.30**	**6.5**

平均周转时间　　t = 1.725

平均带权周转时间　w = 6.875

作业调度算法（续）

2. 短作业优先调度算法（SJF：Shortest Job First)

(1）策略：按照作业请求运行的时间长短进行调度，优先选择短作业。

(2) 特点： 易实现，系统吞吐量高。

只照顾短作业，而没有考虑长作业的利益；需要预知作业的运行时间（how？）。

短作业优先调度算法实例

作业	Tsi	Tri	开始时间	Tci	Ti	Wi
1	8.00	2.00	8.00	10.00	2.00	1
2	8.50	0.50	10.30	10.80	2.30	4.6
3	9.00	0.10	10.00	10.10	1.10	11
4	9.50	0.20	10.10	10.30	0.80	4

平均周转时间　　t = 1.55

平均带权周转时间　w = 5.15

作业调度算法（续）

3. 响应比高者优先调度算法（HRN：Highest Response Ratio Next)

响应比 = 响应时间/运行时间

(1) 策略：优先选择响应比高者作业运行。

(2) 特点： 1、2算法的一种折衷，既照顾了短作业，又不使长作业等待时间过长。

响应比高者优先调度算法实例

作业	Tsi	Tri	开始时间	Tci	Ti	Wi
1	8.00	2.00	8	10.00	2.00	1
2	8.50	0.50	10.10	10.60	2.10	4.2
3	9.00	0.10	10.00	10.10	1.10	11
4	9.50	0.20	10.60	10.80	1.30	6.5

平均周转时间　　t = 1.625

平均带权周转时间　w = 5.675

进程调度

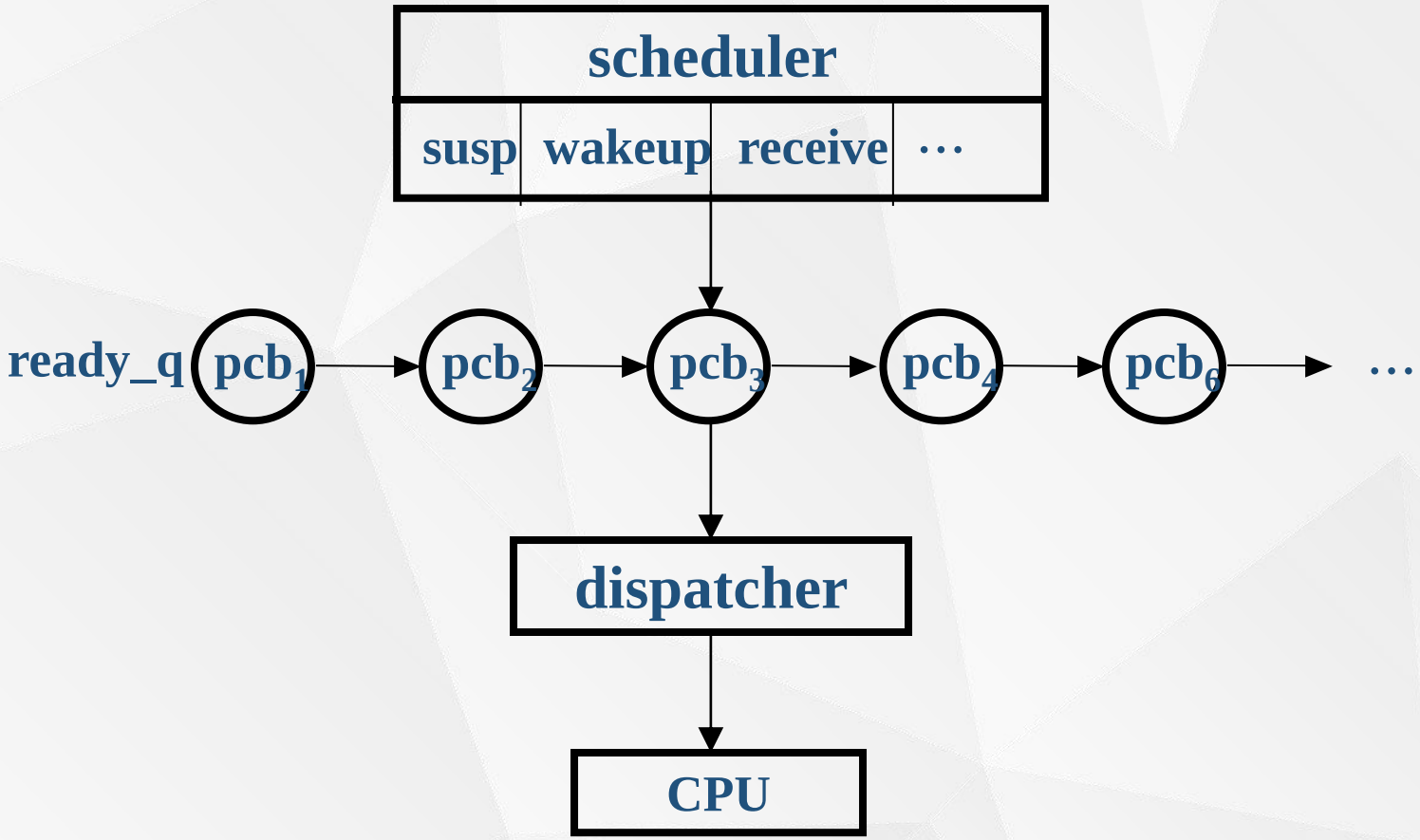
进程调度/分派结构

1. 调度

在众多处于就绪状态的进程中，按一定的原则选择一个进程。

2. 分派

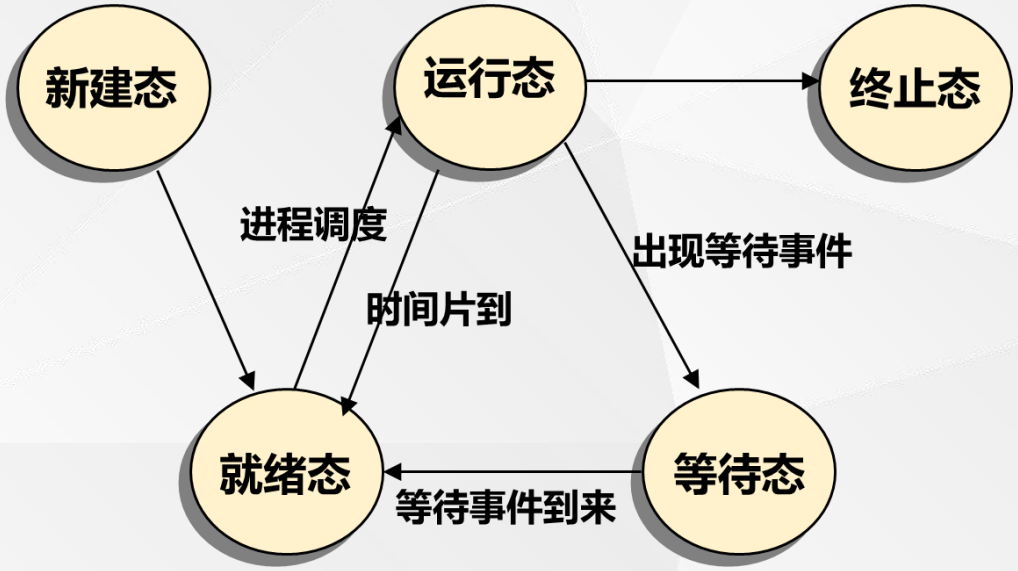
当处理机空闲时，移出就绪队列中第一个进程，并赋予它使用处理机的权利。



进程调度的功能

1. 记录进程的有关情况和状态特征
2. 决定调度策略
   - 优先调度原则 —— 进程就绪队列按进程优先级高低排序
   - 先来先服务原则 —— 进程就绪队列按进程来到的先后次序排序
3. 实施处理机的分配和回收

CPU调度时机



- 当一个进程从运行态切换成等待态时
- 当一个进程从运行态切换成就绪态时
- 当一个进程从等待态切换成就绪态时
- 当一个进程终止时

进程调度方式

当一进程正在处理机上执行时，若有某个更为“重要而紧迫”的进程需要进行运行，系统如何分配处理机。

- **非剥夺方式**——正在执行的进程继续执行，直到该进程完成或发生某事件而进入“完成”或“阻塞”状态时，才把处理机分配给“重要而紧迫”的进程。
- **剥夺方式**——当“重要而紧迫”的进程一到，便暂停正在执行的进程，立即把处理机分配给优先级更高的进程。

进程调度算法

1. 优先数调度算法

- 预先确定各进程的优先数，系统把处理机的使用权赋予就绪队列中具备最高优先权（优先数和一定的优先级相对应）的就绪进程。
- 优先数的分类及确定
  - 静态优先数
  - 动态优先数

静态优先数

- 在进程被创建时确定，且一经确定后在整个进程运行期间不再改变。
- 静态优先数的确定
  - 优先数根据进程所需使用的资源来计算
  - 优先数基于程序运行时间的估计
  - 优先数基于进程的类型

动态优先数

- 进程优先数在进程运行期间可以改变。
- 动态优先数的确定
  - 进程使用CPU超过一定数值时，降低优先数；
  - 进程进行I/O操作后，增加优先数；
  - 进程等待时间超过一定数值时，提高优先数。

2. 循环轮转调度算法

当CPU空闲时，选取就绪队列首元素，赋予一个时间片，当时间片用完时，该进程转为就绪态并进入就绪队列末端。

该队列排序的原则是什么？

讨论：时间片如何取值?

- 若时间片取值太小，多数进程不能在一个时间片内运行完毕，切换就会频繁，开销显著增大，从系统效率来看，时间片取大一点好。
- 若时间片取值太大，随着就绪队列里进程数目增加，轮转一次的总时间增大，对进程的响应速度变慢。
- 为了满足响应时间要求，要么限制就绪进程数量，要么采用动态时间片法，根据负载状况，及时调整时间片的大小。

2. 循环轮转度算法（续）

简单循环轮转调度

就绪队列中的所有进程以等速度向前进展

$$q = t/n$$

t 为响应时间，n为进入系统的进程数目。

由于该算法简单易于实现，且系统开销较小，早期的分时操作系统和目前一些应用系统中广泛采用了这种调度算法。

循环轮转调度算法的发展

可变时间片轮转调度：时间片的大小是可变的，系统可根据系统中当前的进程数来确定时间片的大小。

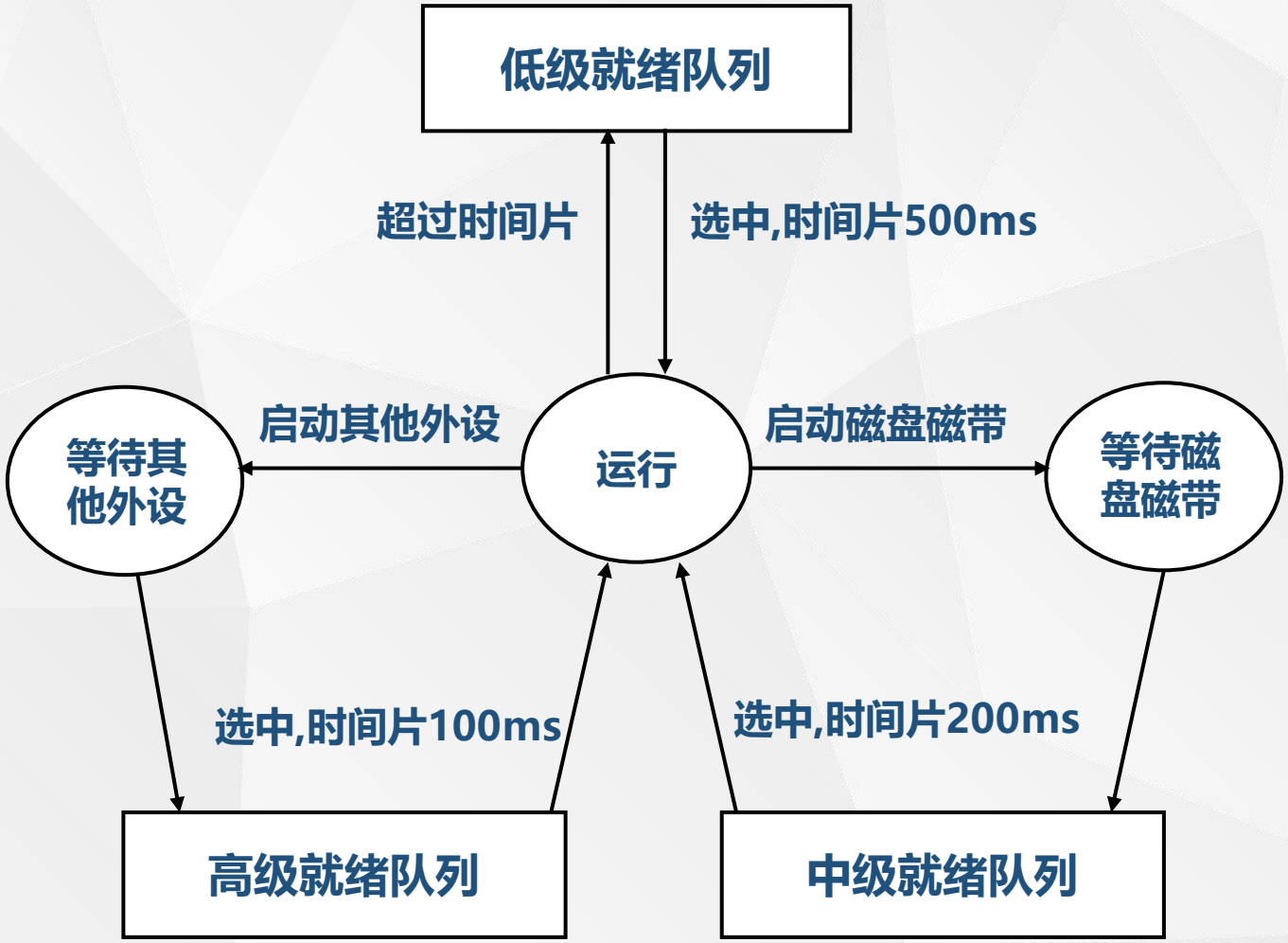
这种算法从理论上克服了系统中进程数很少时系统开销大的缺点，但修改时间片的大小、统计系统进程的数量也需要消耗系统时间。另外，调整时间片大小的周期若太大，等于是固定时间片；若太小，则系统开销很大，得不偿失。

循环轮转调度算法的发展（续）

多重时间片循环调度：又称**反馈循环队列**或**多队列策略**。主要思想是将就绪进程分为两级或多级，系统相应建立两个或多个就绪进程队列，较高优先级的队列一般分配给较短的时间片。

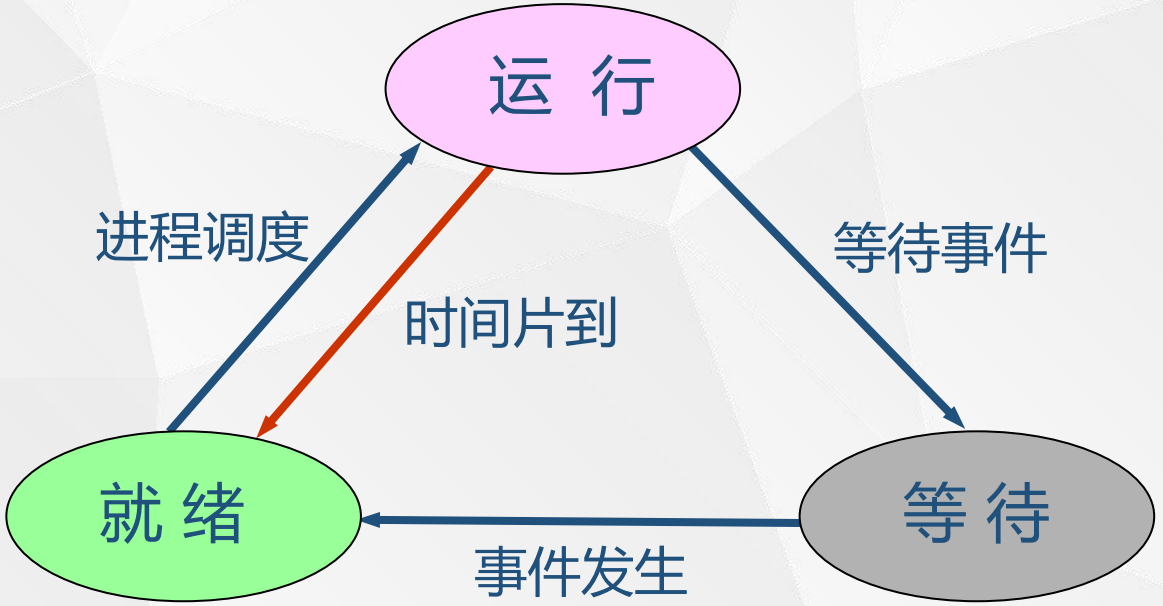
处理器调度先从高级就绪进程队列中选取可占有处理器的进程，只有在其为空时，才从较低级的就绪进程队列中选取。

三级反馈队列示例

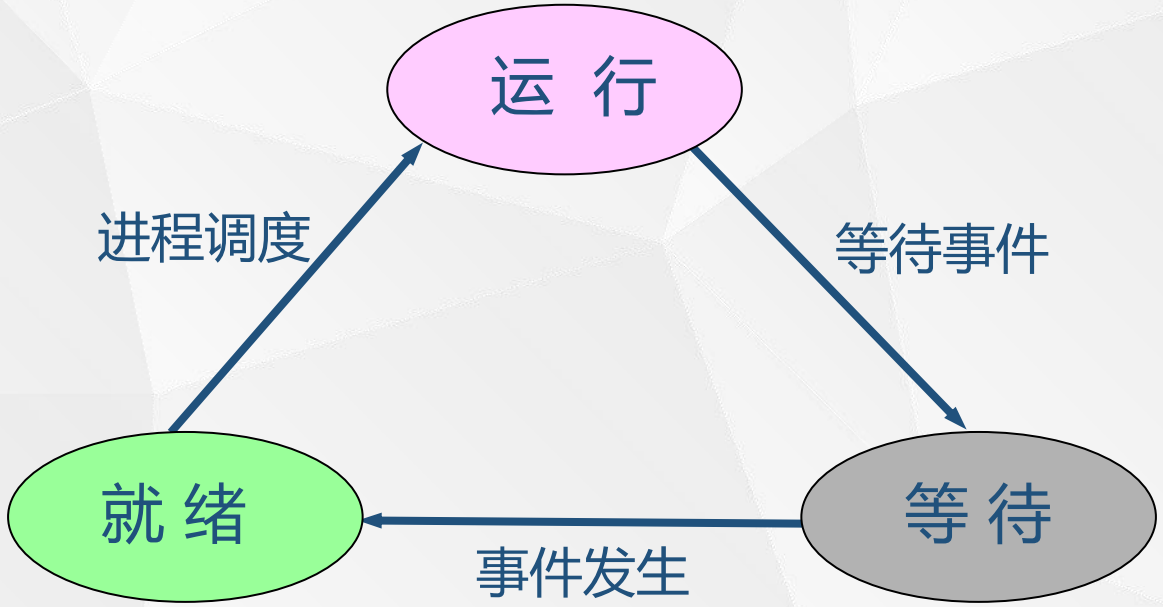

低级就绪队列
超过时间片
选中,时间片500ms
等待其他外设
启动其他外设
运行
启动磁盘磁带
等待磁盘磁带
选中,时间片100ms
选中,时间片200ms
高级就绪队列
中级就绪队列

调度用的进程状态变迁图

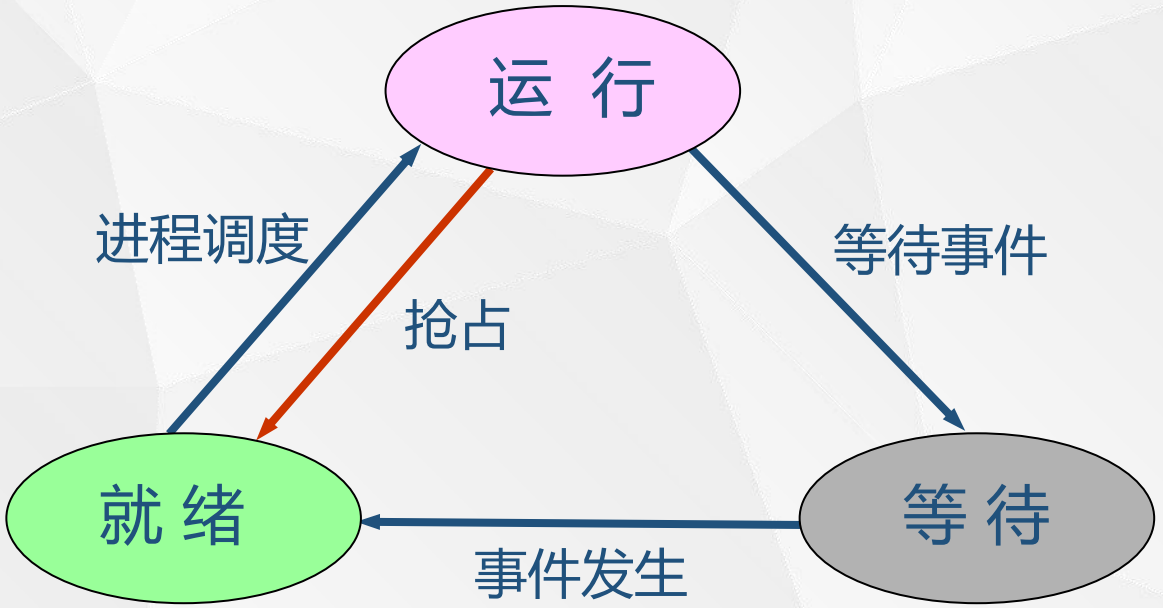
简单时间片轮转算法



不可抢占的优先数调度算法



简单可抢占的优先数调度算法



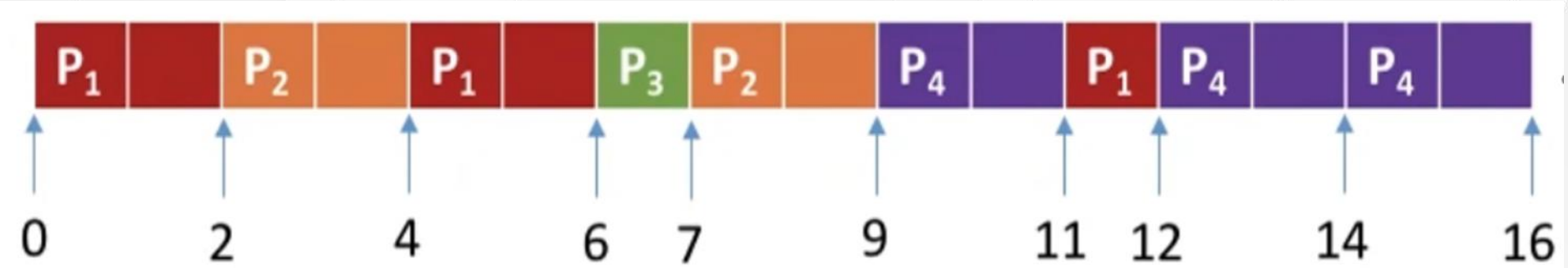
例1：各进程到达就绪队列的时间、需要的运行时间如下表所示。若系统采用**时间片轮转调度算法**，且时间片大小为2，画出进程运行时序图。

进程	到达时间	运行时间
P1	0	5
P2	2	4
P3	4	1
P4	5	6

调度过程

- 0时刻（P1运行）：只有P1到达就绪对列，让P1运行一个时间片2。
- 2时刻（P2运行）：P2到达就绪对列，P1被剥夺CPU，放到队尾，P2运行。//p1
- 4时刻（P1运行）：P3到达，先插到就绪队列队尾，紧接着，P2也插到队尾。//p3:p2
- 5时刻（P1运行）：P4到达，插到就绪队尾。//p3:p2:p4
- 6时刻（P3运行）：P1时间片用完,回到就绪队尾。//p2:p4:p1
- 7时刻（P2运行）：P3主动放弃CPU。//p4:p1
- 9时刻（P4运行）：P2时间片用完，并刚好运行完。//p1
- 11时刻（P1运行）：P4时间片用完，回到就绪队尾。//p4
- 12时刻（P4运行）：P1运行完，主动放弃CPU，就绪队列只剩P4。
- 14时刻（P4运行）:P4运行完。

时序图



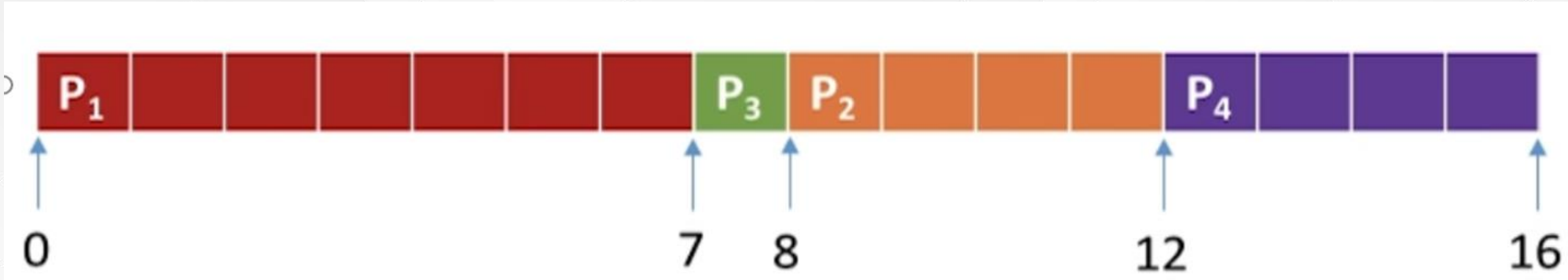
例2：各进程到达就绪队列的时间、需要的运行时间、进程优先级如下表所示。若系统采用**非抢占式的优先级调度算法**，画出进程运行时序图。

进程	到达时间	运行时间	优先级
P1	0	7	1
P2	2	4	2
P3	4	1	3
P4	5	4	2

调度过程

- 0时刻（P1运行）：只有P1到达。
- 7时刻（P3运行）：P1运行完主动放弃处理机，其余进程都已到达，P3优先级最高。
- 8时刻（P2运行）：P3完成，P2、P4优先级相同，由于P2先到达，P2优先。
- 12时刻（P4）：P2完成，只剩P4。
- 16时刻：所有进程结束。

时序图



例3：各进程到达就绪队列的时间、需要的运行时间、进程优先级如下表所示。若系统采用**抢占式的优先级调度算法**，画出进程运行时序图。

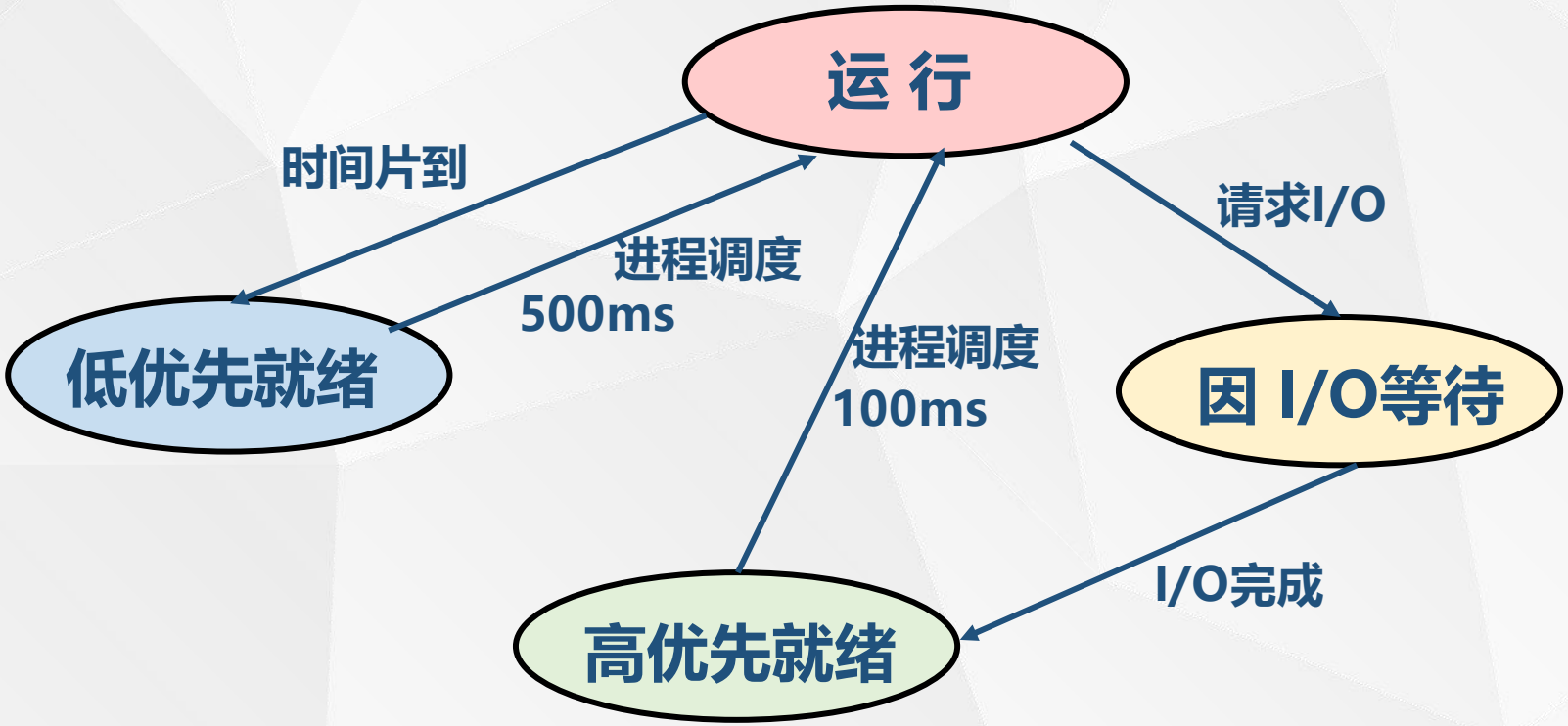
进程	到达时间	运行时间	优先级
P1	0	7	1
P2	2	4	2
P3	4	1	3
P4	5	4	2

调度过程

- 0时刻（P1运行）：P1到达。
- 2时刻（P2运行）：P2到达，优先级更高。
- 4时刻（P3运行）：P3到达，优先级更高。
- 5时刻（P2运行）：P3完成，主动释放，P4到达，由于P2更先进入就绪队列，P2上。
- 7时刻（P4运行）：只剩P1、P4，P4优先级高。
- 11时刻（P1运行）：P4完成，P1上。

时序图

时间片与优先数混合调度



队列结构

- **I/O等待队列** —— 一个进程如果请求I/O，则进入I/O等待队列；
- **低优先就绪队** —— 一个进程如果在运行中超过了它的时间片，就进入低优先就绪队列；
- **高优先就绪队列** —— 当进程从等待状态变为就绪状态时，则进入高优先就绪队列。

进程调度算法

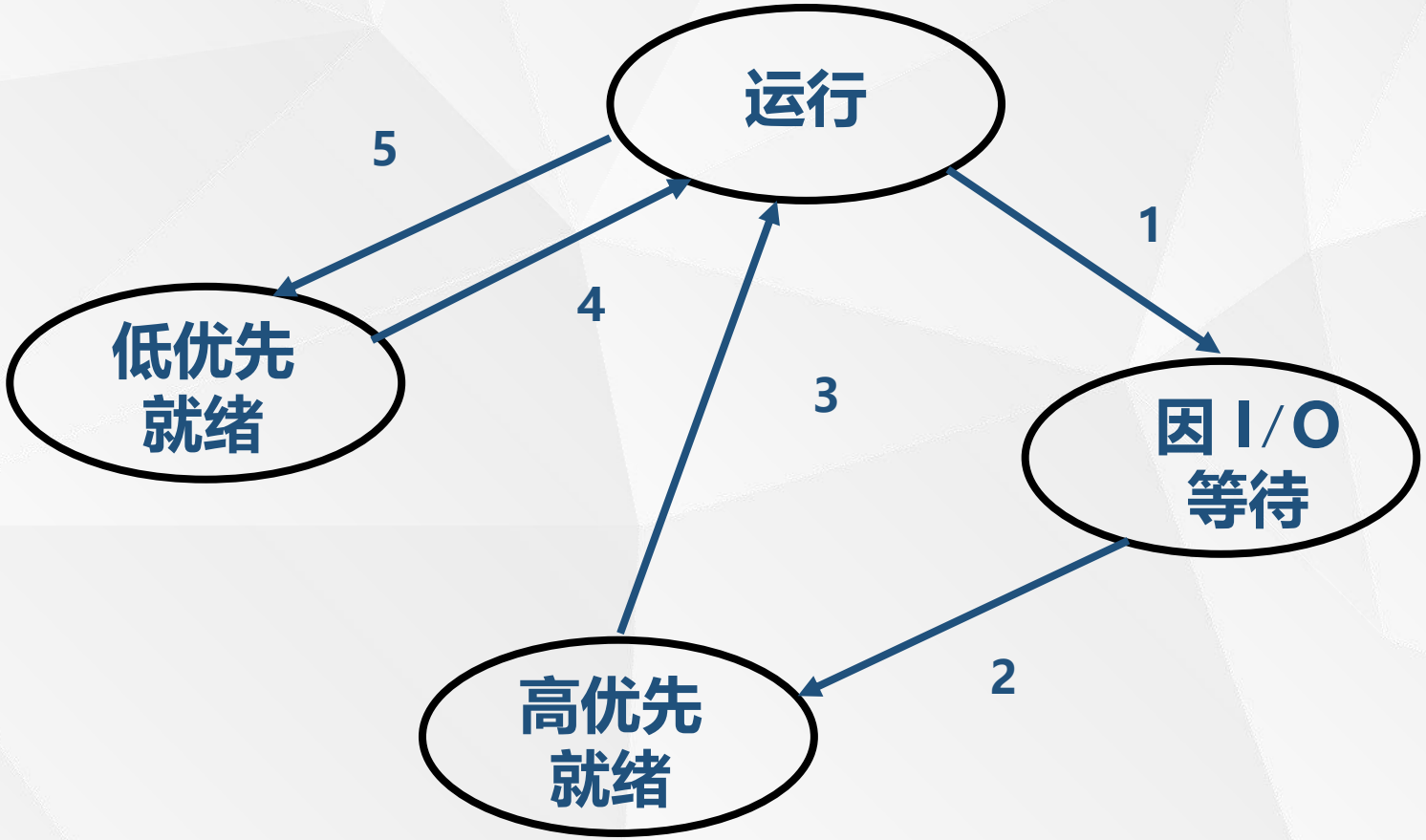
采用优先调度与时间片调度相结合的调度策略：

(1) 当CPU空闲时，若高优先就绪队列非空，则从高优先就绪队列中选择一个进程运行，分配时间片为100ms。

(2) 当CPU空闲时，若高优先就绪队列为空，则从低优先就绪队列中选择一个进程运行，分配时间片为500ms。

调度效果：优先照顾了I/O量大的进程；适当照顾了计算量大的进程。

Q：以下因果变迁有没有可能出现？什么情况下出现？



变迁1 → 变迁3?

变迁1 → 变迁4?

变迁5 → 变迁3?

变迁2 → 变迁5?

Linux系统的进程调度

Linux进程调度目标和特点

(1) 调度策略

① 基于动态优先级和可变时间片的调度

② 调度方式为可抢占式调度

(2) 调度目标

① 实现算法复杂度为O(1)级的调度

② 提高交互性能

③ 保证公平

④ 实现SMP可扩展性

(3) Linux进程调度的特点（多级反馈循环调度）

① 基于进程过去行为的启发式算法；

② 选择优先级高的进程先运行，相同优先级的进程按循环方式调度；

③ 动态优先级依进程占有CPU的情况、休眠时间的长短来增、减；

④ 系统根据进程优先级调整分配给它的时间片；

⑤ 实施可抢占调度方式

可变优先级

(1) 基于优先级的调度

优先级高的进程先运行，相同优先级的进程按轮转方式进行调度。

(2) 静态优先级

① **静态优先级的确定**——在进程创建时，新创建的进程继承父进程的静态优先级

② **静态优先级的取值范围**——0-99对应实时进程，100-139对应普通进程（取值越小，优先级越高）；

③ **静态优先级的改变**——用户可以通过系统调用改变nice值，从而改变自己拥有的静态优先级。

(3) 动态优先级

① 每个进程有一个动态优先级，它是进程调度程序选择可运行进程所使用的参数，其取值范围是100 (最高优先级) ~ 139 (最低优先级)

（实时进程优先级不变）

② **动态优先级的计算**

动态优先级 = max(100，min(静态优先级 - bonus + 5，139))

- bonus的取值范围是 0～10
- bonus的值小于5，表示降低动态优先级，以示惩罚
- bonus的值大于5，表示增加动态优先级以示奖励

③ 进程休眠时间与bonus值的关系

平均休眠时间	bonus值
大于或等于0，小于 100ms	0
大于或等于100，小于 200ms	1
大于或等于200，小于 300ms	2
大于或等于300，小于 400ms	3
大于或等于400，小于 500ms	4
大于或等于500，小于 600ms	5
大于或等于600，小于 700ms	6
大于或等于700，小于 800ms	7
大于或等于800，小于 900ms	8
大于或等于900，小于 1000ms	9
大于1s	10

可变时间片

(1) Linux系统的进程调度的目标

优先照顾交互式进程，为其提供较长的时间片。

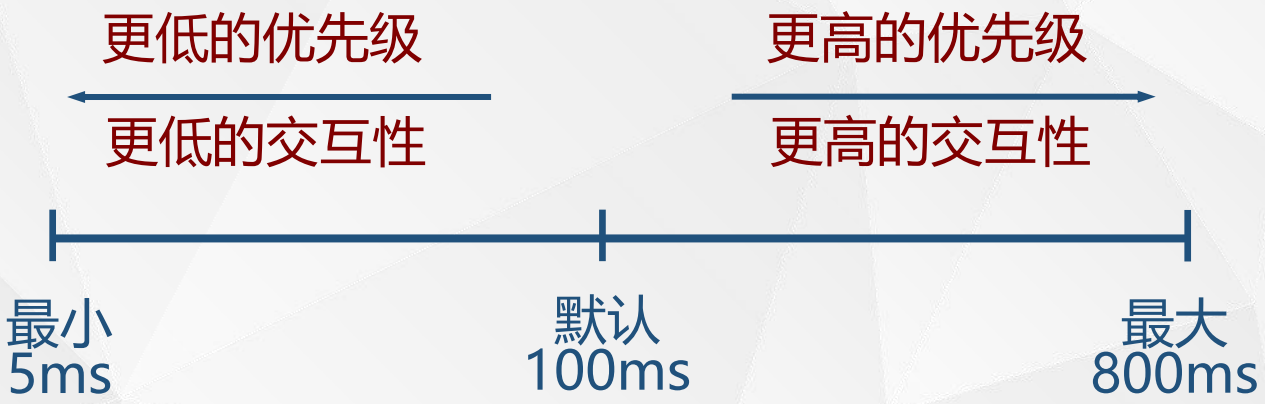
(2) 时间片的计算

① 基本时间片

静态优先级本质上决定了进程的基本时间片。

$$\text{基本时间片} = \begin{cases} (140 - \text{静态优先级}) \times 20 & \text{若静态优先级} < 120 \\ (140 - \text{静态优先级}) \times 5 & \text{若静态优先级} \geq 120 \end{cases}$$

静态优先级越高（值越小），基本时间片越长。



说明	静态优先级	nice值	基本时间片
最高静态优先级	100	-20	800ms
高静态优先级	110	-10	600ms
缺省静态优先级	120	0	100ms
低静态优先级	130	+10	50ms
最低静态优先级	139	+19	5ms
初始创建的进程	父进程的值	父进程的值	父进程的一半

(3) 时间片处理的时机

① 创建新进程时的处理

新创建的子进程和父进程均分父进程剩余的时间片。

② 进程用完时间片时的处理

根据任务的静态优先级重新计算时间片。

(4) 时间片的使用

① 一个进程拥有的时间片可分多次使用，放弃CPU时进入活动队列；

② 当一个进程的时间片耗尽时，认为是过期进程，进入过期队列。

(5) 活动队列和过期队列

每个处理器维护两个优先级数组：**活动数组** 和 **过期数组**。

① 活动数组上的可执行队列中的进程都有剩余时间片

② 过期数组上的可执行队列中的进程都已耗尽时间片

当一个进程的时间片耗尽时，被移至过期队列中；当活动数组上的可执行队列中的所有进程都已耗尽时间片，在活动数组和过期数组之间切换指针。

思考：

- 新创建的进程应该进哪个数组?
- 如果不断有新进程到达怎么办?
- 实时进程时间片耗尽了怎么办?

Linux进程调度算法

(1) 可执行队列（runqueue 结构）

数据类型	名称	说明
spinlock_t	lock	保护进程链表的自旋锁
⋮	⋮	⋮
prio_array_t	*active	指向活动进程链表的指针
prio_array_t	*expired	指向过期进程链表的指针
prio_array_t	arrays[2]	活动进程和过期进程的两个集合
⋮	⋮	⋮

(2) 优先级数组（prio_array_t 结构）

优先级数组是 prio_array 类型的结构体，该数组描述了可运行进程的集合，包括：

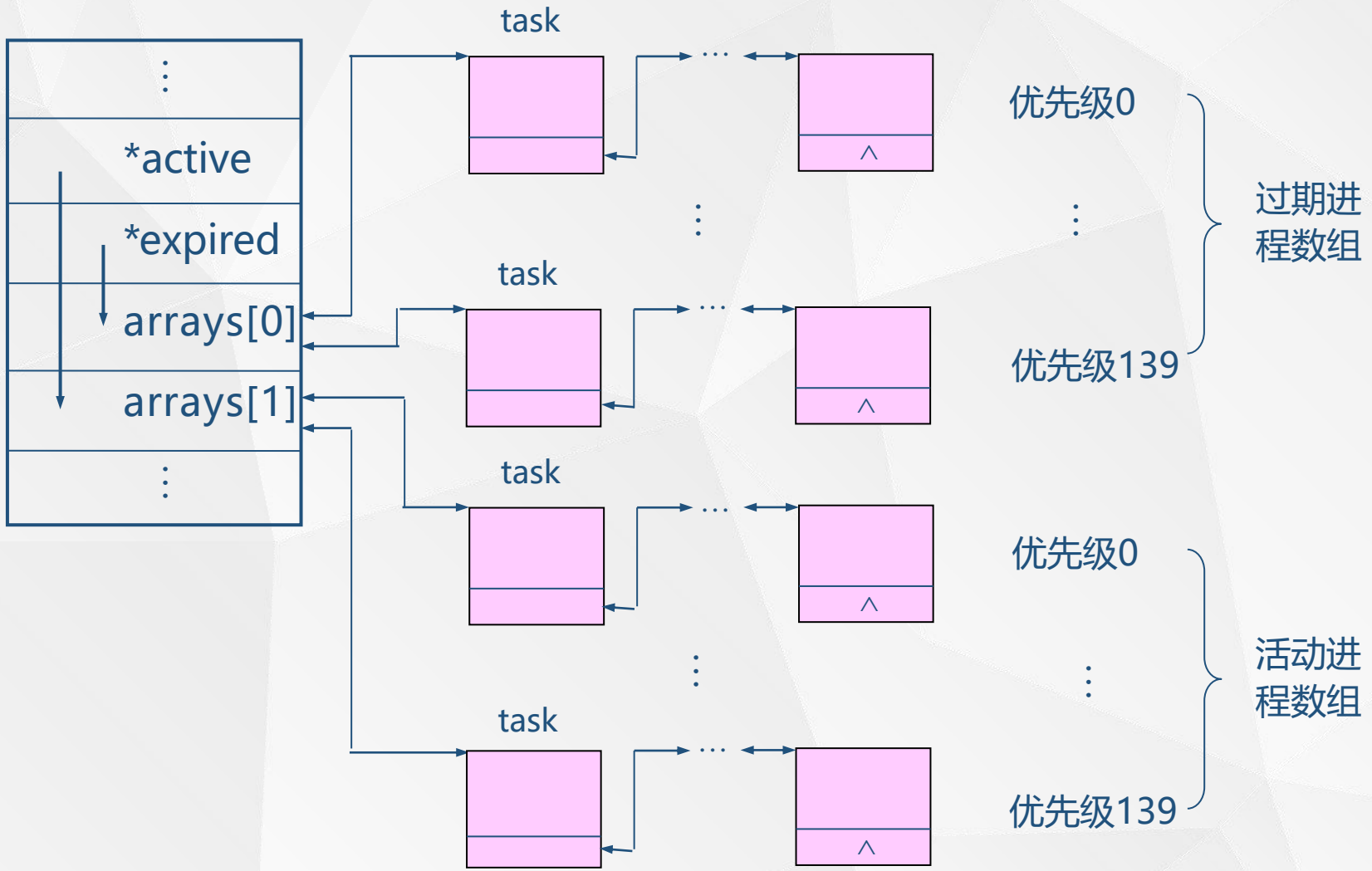
① 140个双向链表头（每个链表对应一个优先级队列）

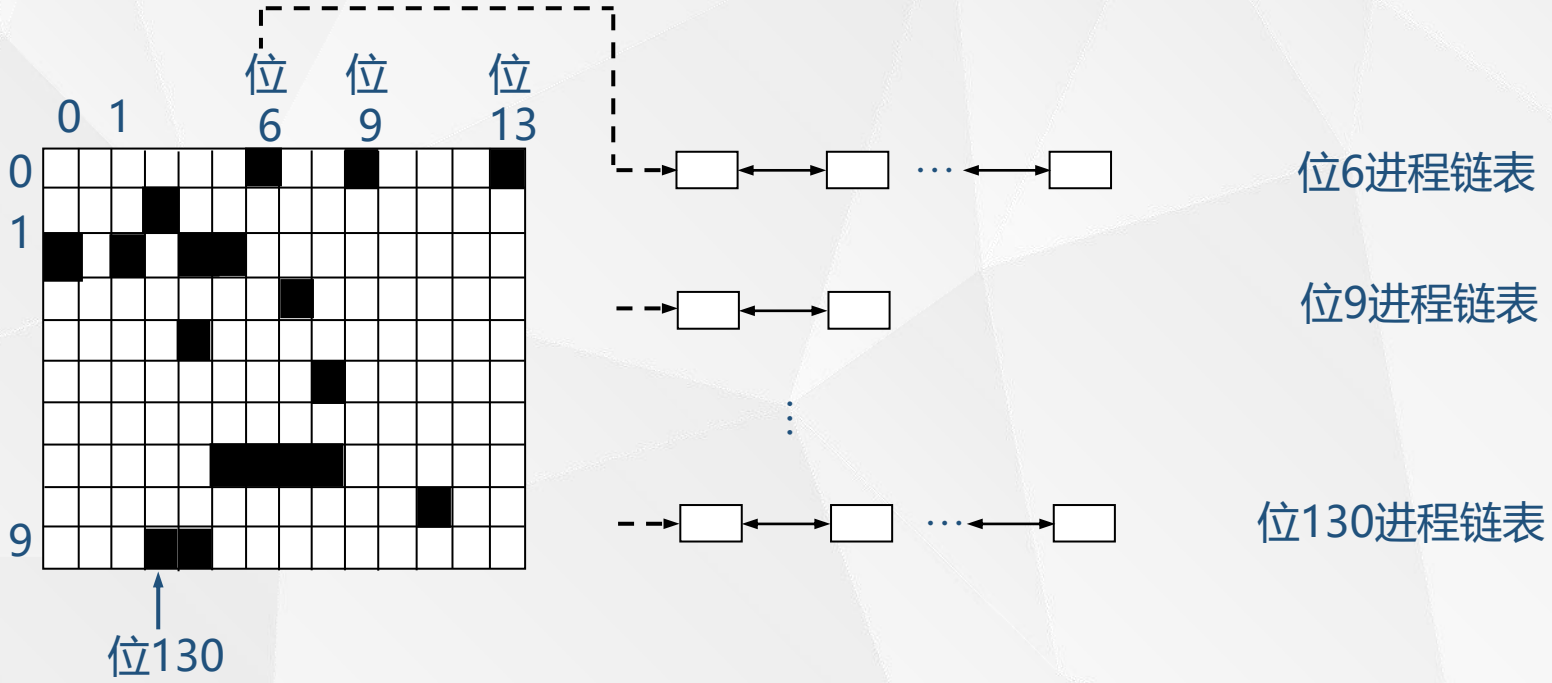
② 一个进程优先级位图

③ 该数组所包含的进程总数

```
struct prio_array {
        int                 nr_active;                  /* 任务数目 */
        unsigned            bitmap[BITMAP_SIZE];        /* 优先级位图 */
        struct list_head    queue[MAX_PRIO];            /* 优先级队列 */
 }
```

④ 优先级数组图示


*active
*expired
arrays[0]
arrays[1]
task
task
task
task
优先级0
优先级139
过期进程数组
优先级0
优先级139
活动进程数组



优先级位图

⑤ 优先级位图的处理

- 初始时，位图中所有位被置为0；
- 当一个具有某个特定优先级的进程就绪时，位图中相应位被置为1；
- 调度时，查找系统中优先级最高的进程，就转化为查找位图中取值为1的第一个位。
- 由于优先级个数是常量，所以查找时间恒定，不受系统中可执行进程数目的影响，使Linux系统的进程调度算法具有O(1) 的算法复杂度。

第6章 小结

一、处理机的多级调度

二、作业调度

1. 作业的四种状态

2. 作业控制块

3. 周转时间、带权周转时间：定义、物理意义

4. **常用的作业调度算法**：对一批作业，若采用不同的调度算法，能分析、计算调度性能的好坏。

第6章 小结（续）

三、进程调度

1. 调度方式：**非剥夺方式、剥夺方式**

2. 常用的进程调度算法：

优先数调度、循环轮转调度

3. 调度用的进程状态变迁图

通过调度用的进程状态变迁图，能分析系统采用的调度策略，调度性能的好坏，能分析因果变迁及条件。