
进程基础

宿船长 B站专用

目录

01 什么是进程

03 进程组织与控制

02 进程的状态与转换

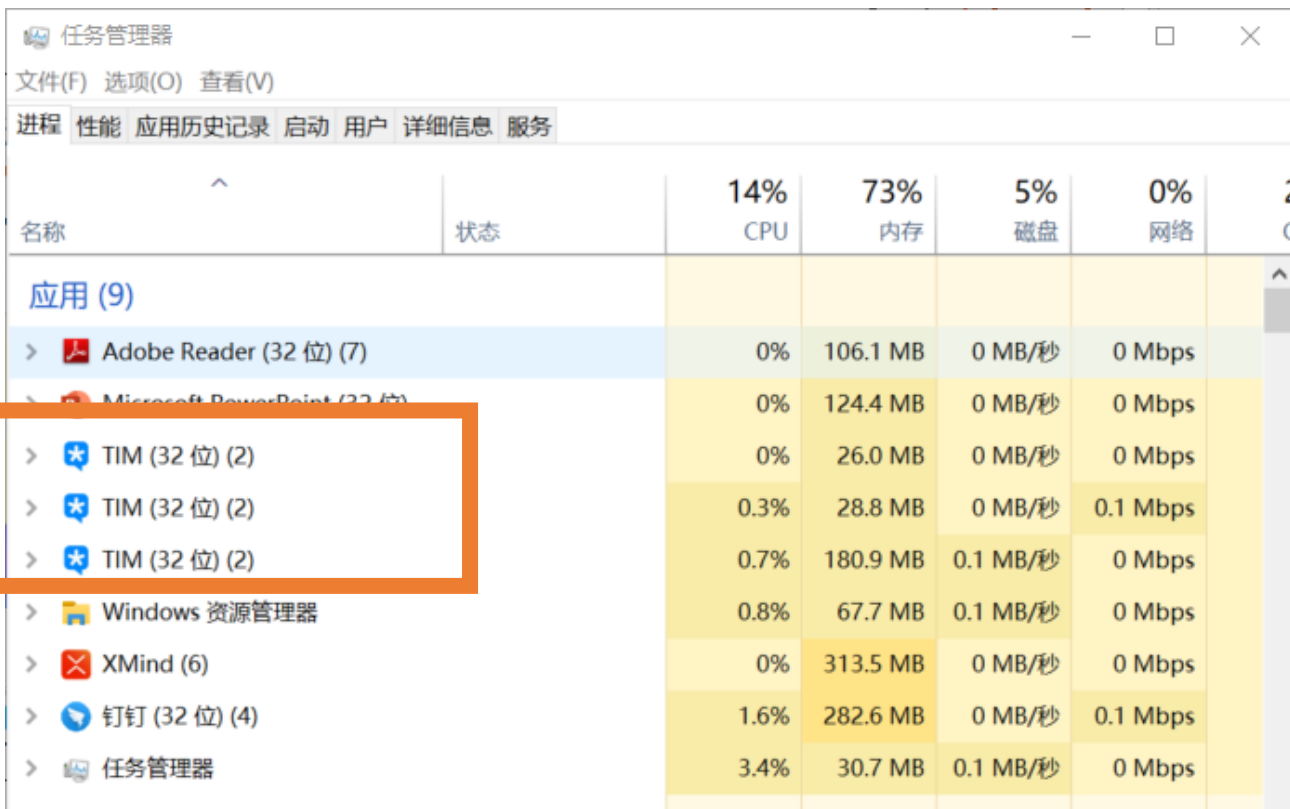
04 进程调度

宿船长 B站专用

进程基础



进程的概念



名称		状态	14% CPU	73% 内存	5% 磁盘	0% 网络
应用 (9)						
> Adobe Reader (32 位) (7)						
> Microsoft PowerPoint (32 位)						
> TIM (32 位) (2)						
> TIM (32 位) (2)						
> TIM (32 位) (2)						
> Windows 资源管理器						
> XMind (6)						
> 钉钉 (32 位) (4)						
> 任务管理器						

程序：是**静态的**，就是个存放在磁盘里的可执行文件，如：Tim.exe。

进程：是**动态的**，是程序的一次执行过程，如：可同时启动多次Tim程序

同一个程序多次执行会对应多个进程



进程的组成——PCB

当进程被创建时，操作系统会为该进程分配一个**唯一的、不重复的**“身份证号”——**PID**（Process ID，进程ID）

基本的进程描述信息，可以让操作系统区分各个进程

操作系统要记录PID、进程所属用户ID（UID）

可用于实现操作系统对资源的管理

还要记录给进程分配了哪些资源（如：分配了多少内存、正在使用哪些I/O设备、正在使用哪些文件）

还要记录进程的运行情况（如：CPU使用时间、磁盘使用情况、网络流量使用情况等）

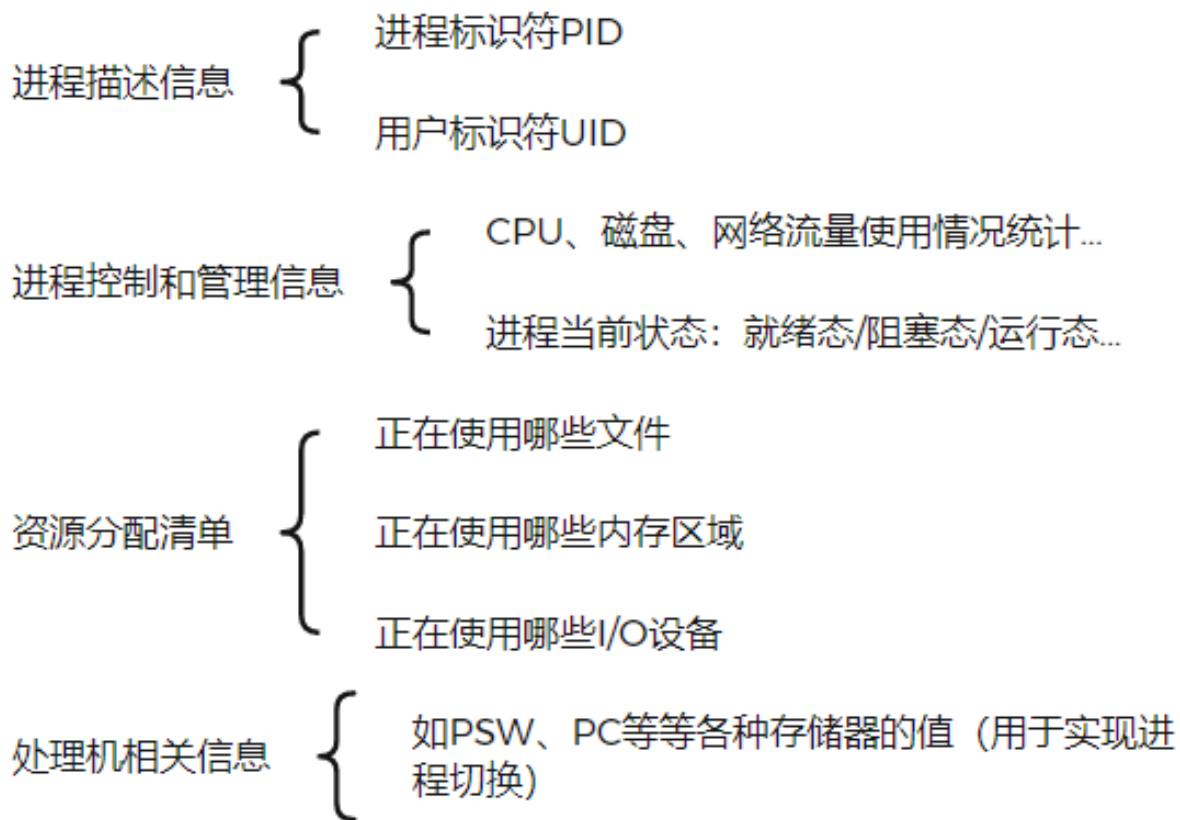
可用于实现操作系统对进程的控制、调度

这些信息都被保存在一个数据结构**PCB**（Process Control Block）中，即**进程控制块**。操作系统需要对各个并发运行的进程进行管理，**但凡管理时所需要的信息，都会被放在PCB中**

进程的组成——PCB

进程控制块 (PCB)

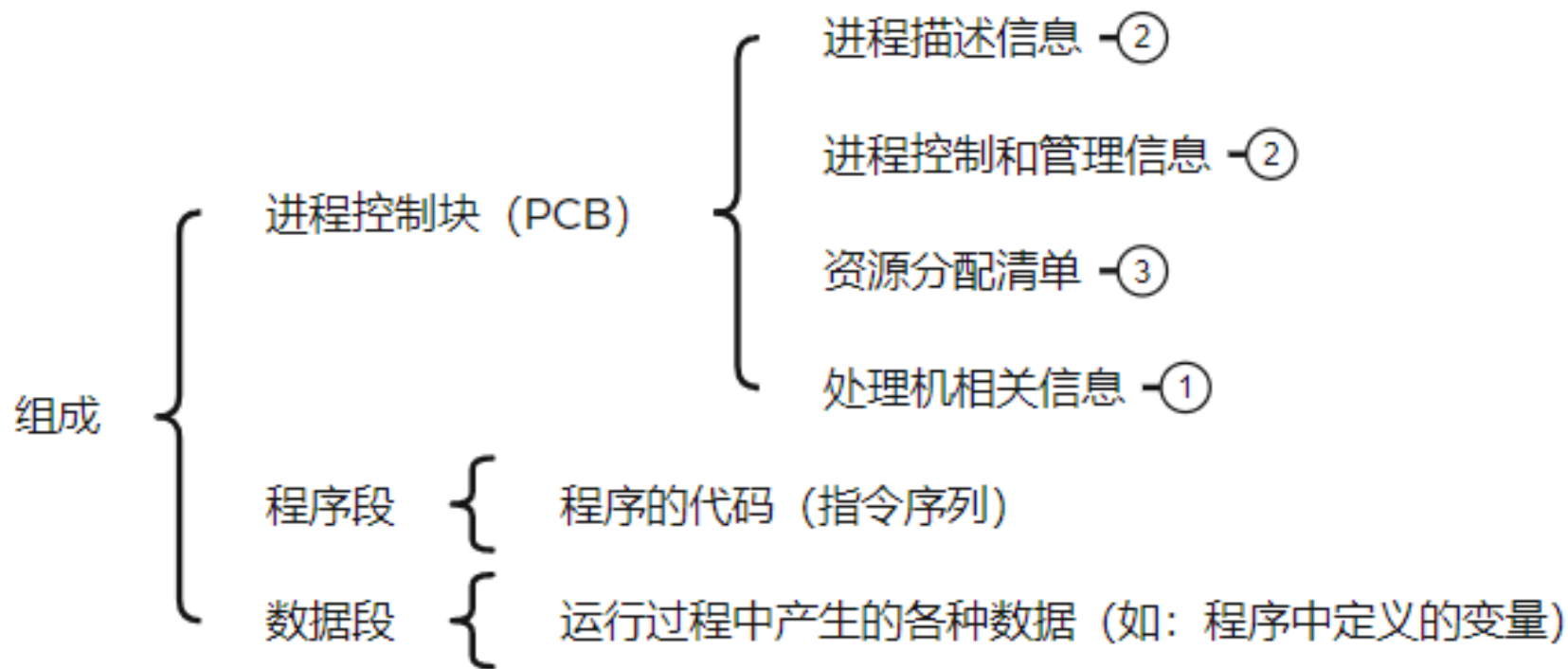
PCB是进程存在的唯一标志，当进程被创建时，操作系统为其创建PCB，当进程结束时，会回收其PCB。



操作系统对进程进行管理工作所需的信息都存在PCB中



进程的组成——程序段、数据段



PCB是给操作系统用的。

程序段、数据段是给进程自己用的。

宿船长 B站专用

进程的组成

组成

PCB是给操作系统用的

进程控制块 (PCB)

- 进程描述信息 ②
- 进程控制和管理信息 ②
- 资源分配清单 ③
- 处理机相关信息 ①

同时挂三个QQ号，会对应三个QQ进程，它们的PCB、数据段各不相同，但程序段的内容都是相同的（都是运行着相同的QQ程序）

程序段

程序的代码（指令序列）

数据段

运行过程中产生的各种数据（如：程序中定义的变量）

程序段、数据段是给进程自己用的，与进程自身的运行逻辑有关

更确切的说，应该是“进程实体(进程映像)的组成”

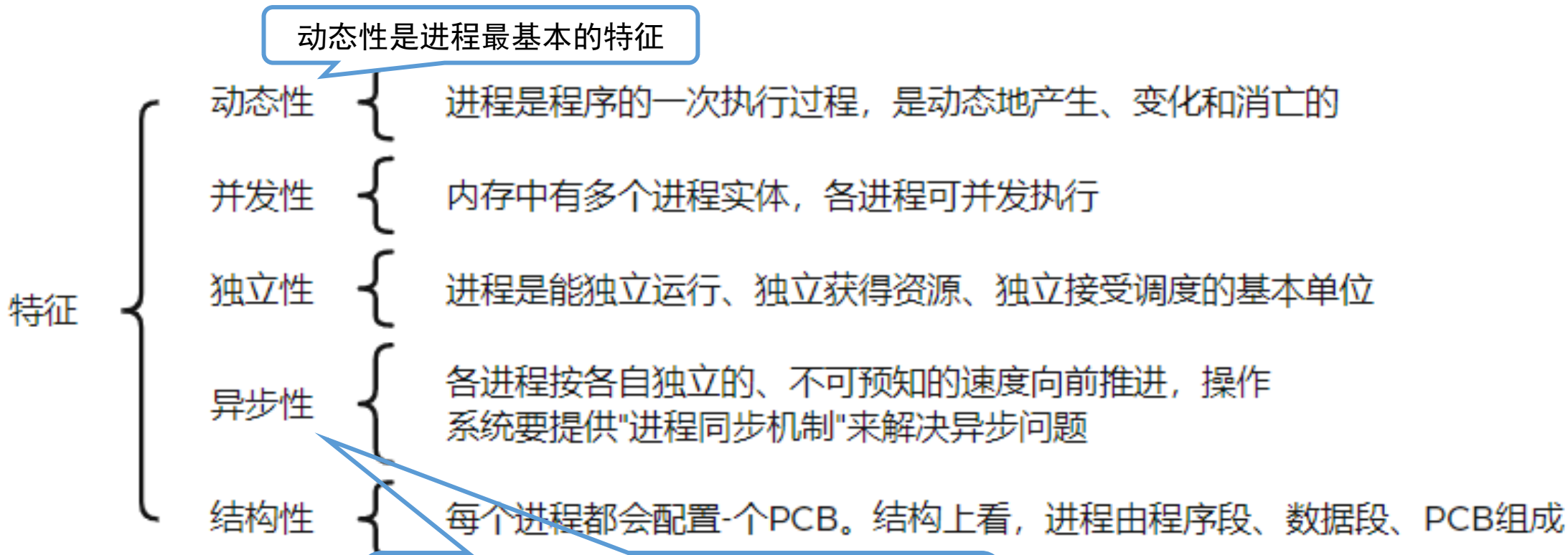
程序段、数据段、PCB三部分组成了进程实体（进程映像）
引入进程实体的概念后，可把进程定义为：
进程是进程实体的运行过程，是系统进行资源分配和调度的一个独立单位。

注意：PCB是进程存在的唯一标志！
一个进程被“调度”，就是指操作系统决定让这个进程上CPU运行



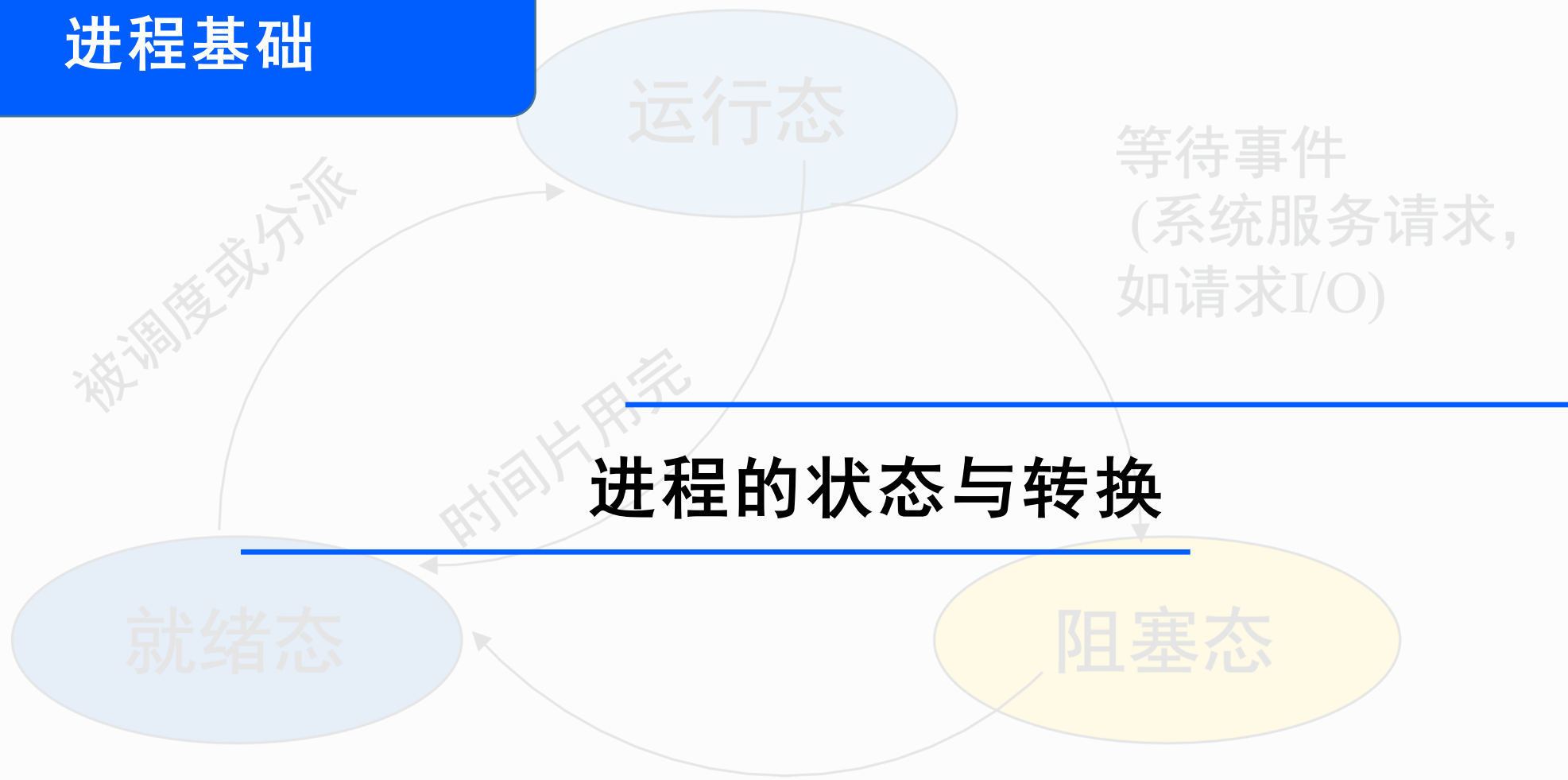
进程的特征

程序是静态的，进程是动态的，相比于程序，进程拥有以下特征：



异步性会导致并发程序执行结果的不确定性。
具体会在“进程同步”相关小节进行学习

进程基础

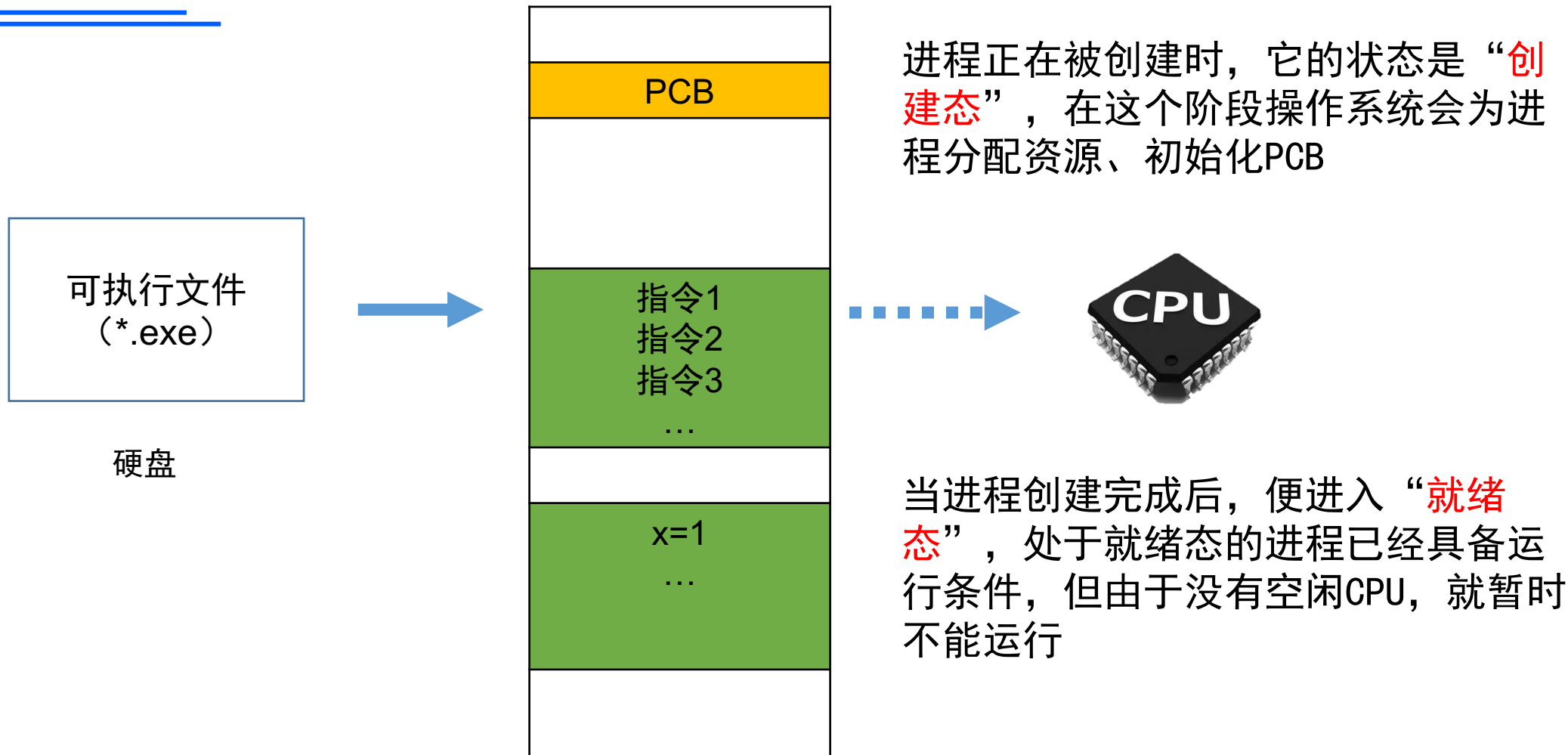


进程的状态与转换

宿船长 B站专用

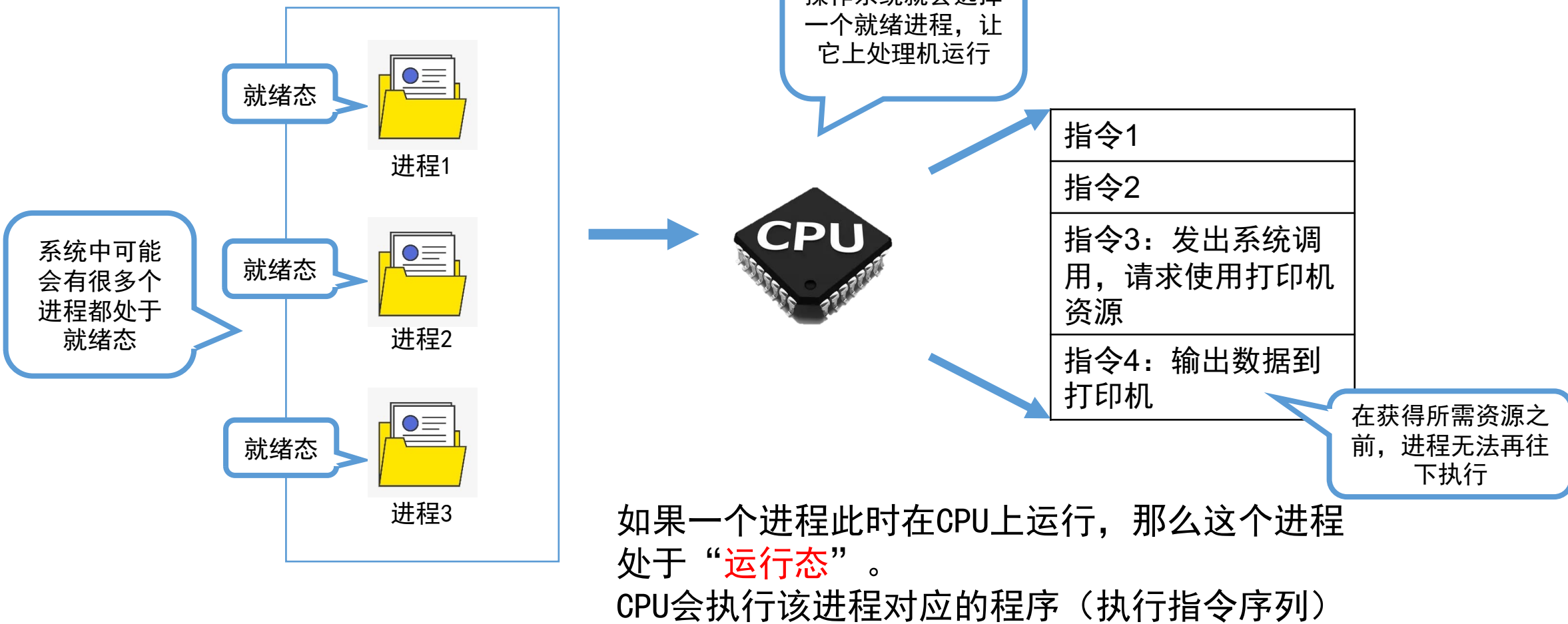


进程的状态——创建态、就绪态

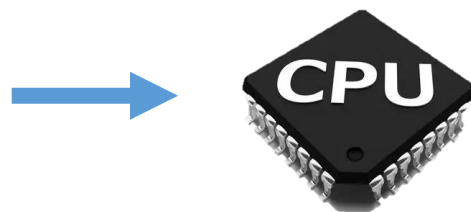
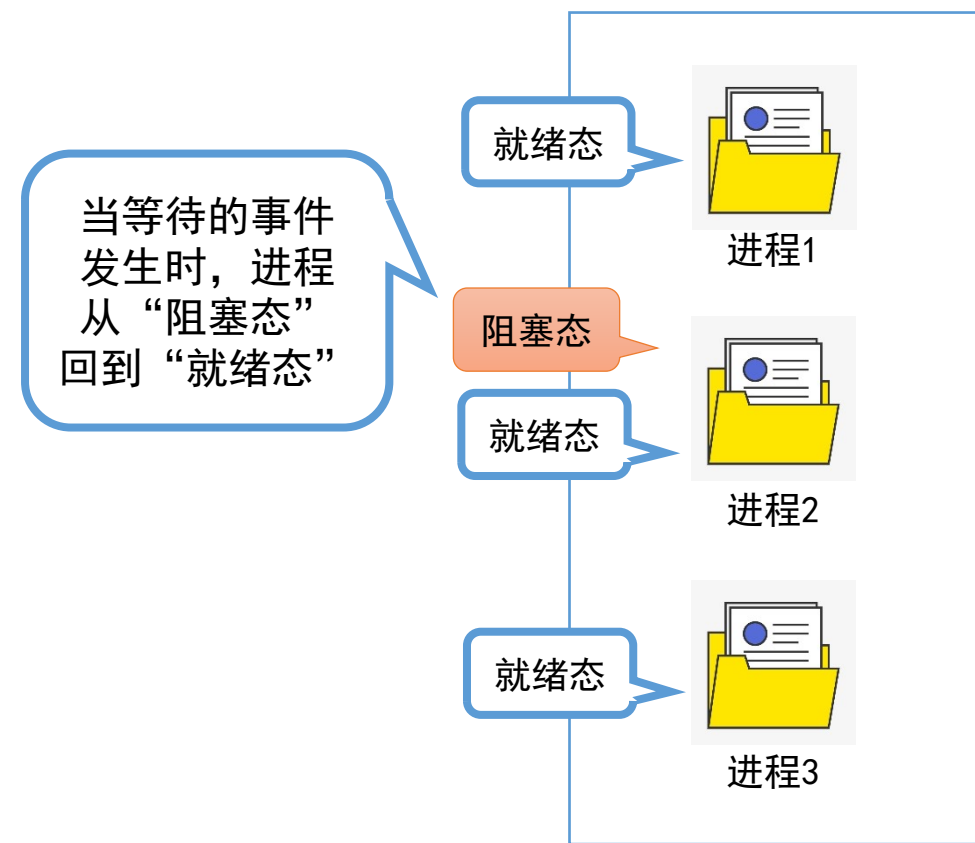




进程的状态——运行态



进程的状态——阻塞态



空闲



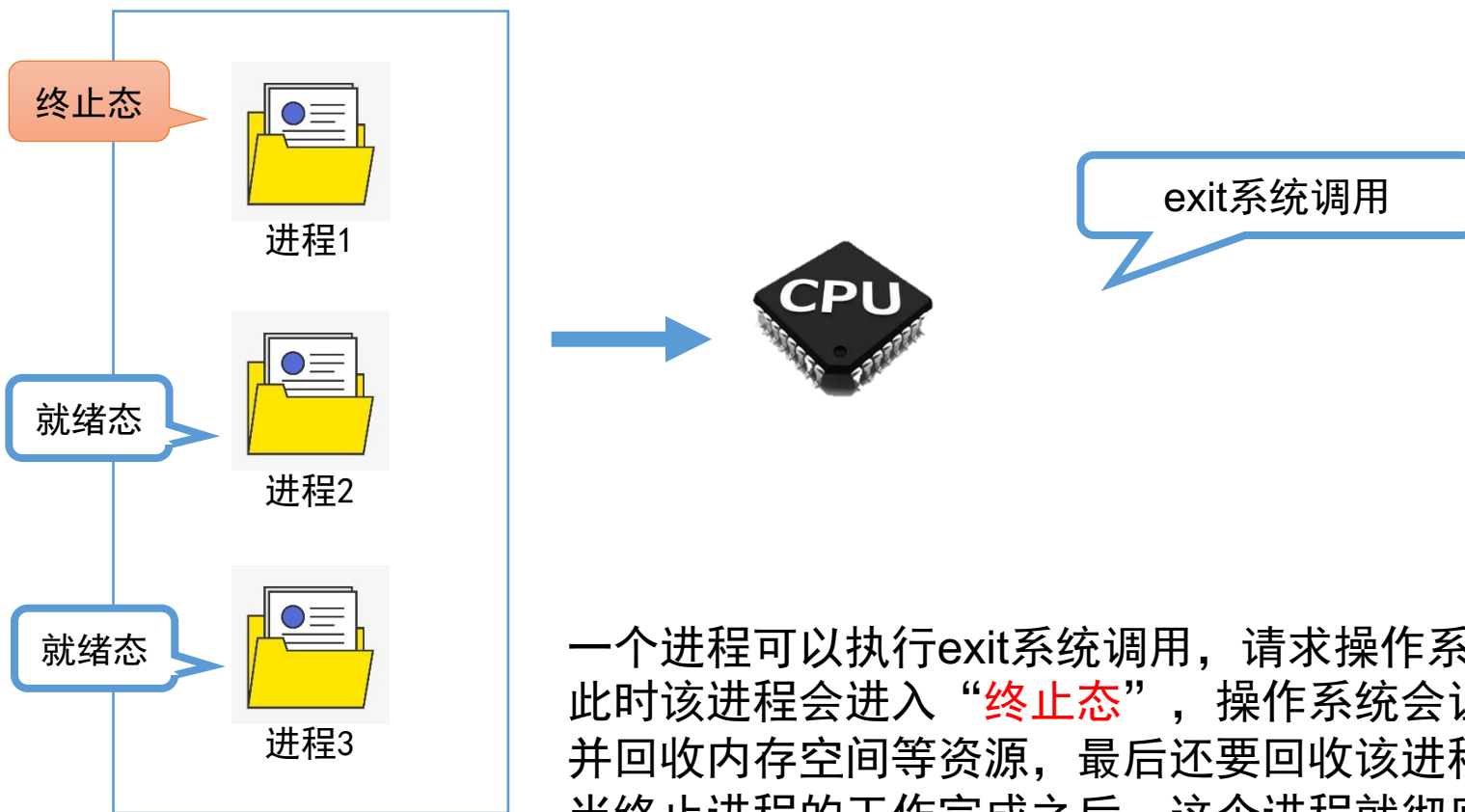
被进程1占用



在进程运行的过程中，可能会请求等待某个事件的发生（如等待某种系统资源的分配，或者等待其他进程的响应）。在这个事件发生之前，进程无法继续往下执行，此时操作系统会让这个进程下CPU，并让它进入“阻塞态”。当CPU空闲时，又会选择另一个“就绪态”进程上CPU运行。



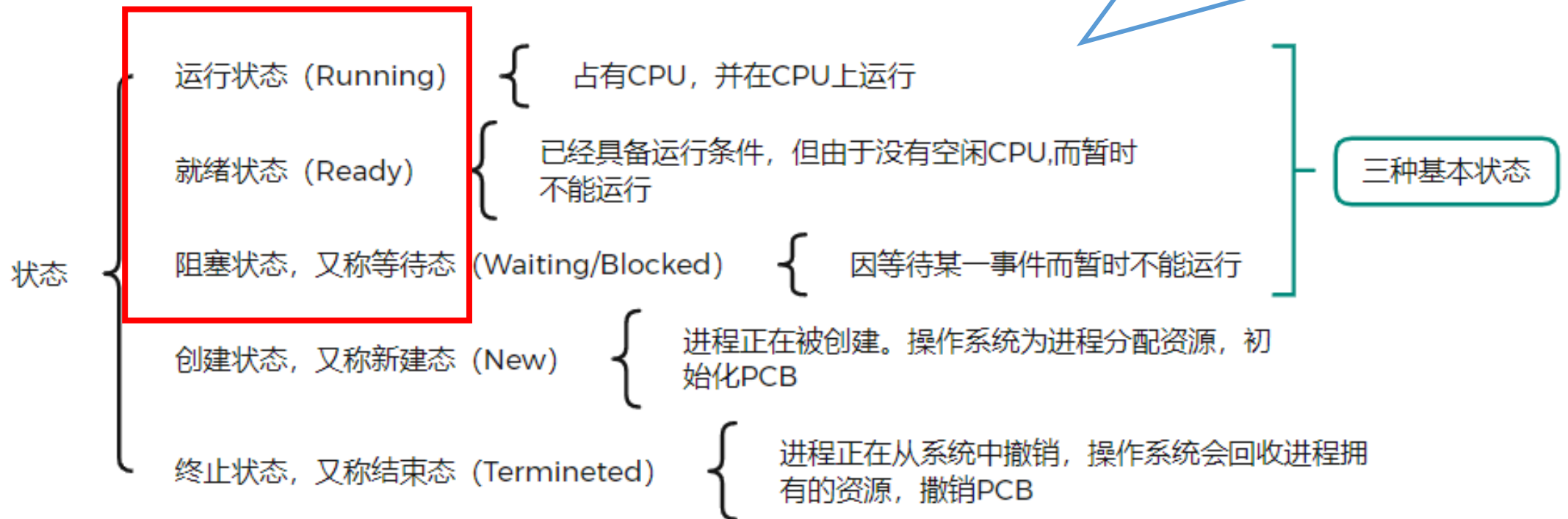
进程的状态——终止态



一个进程可以执行exit系统调用，请求操作系统终止该进程。此时该进程会进入“**终止态**”，操作系统会让该进程下CPU，并回收内存空间等资源，最后还要回收该进程的PCB。当终止进程的工作完成之后，这个进程就彻底消失了。

进程的状态

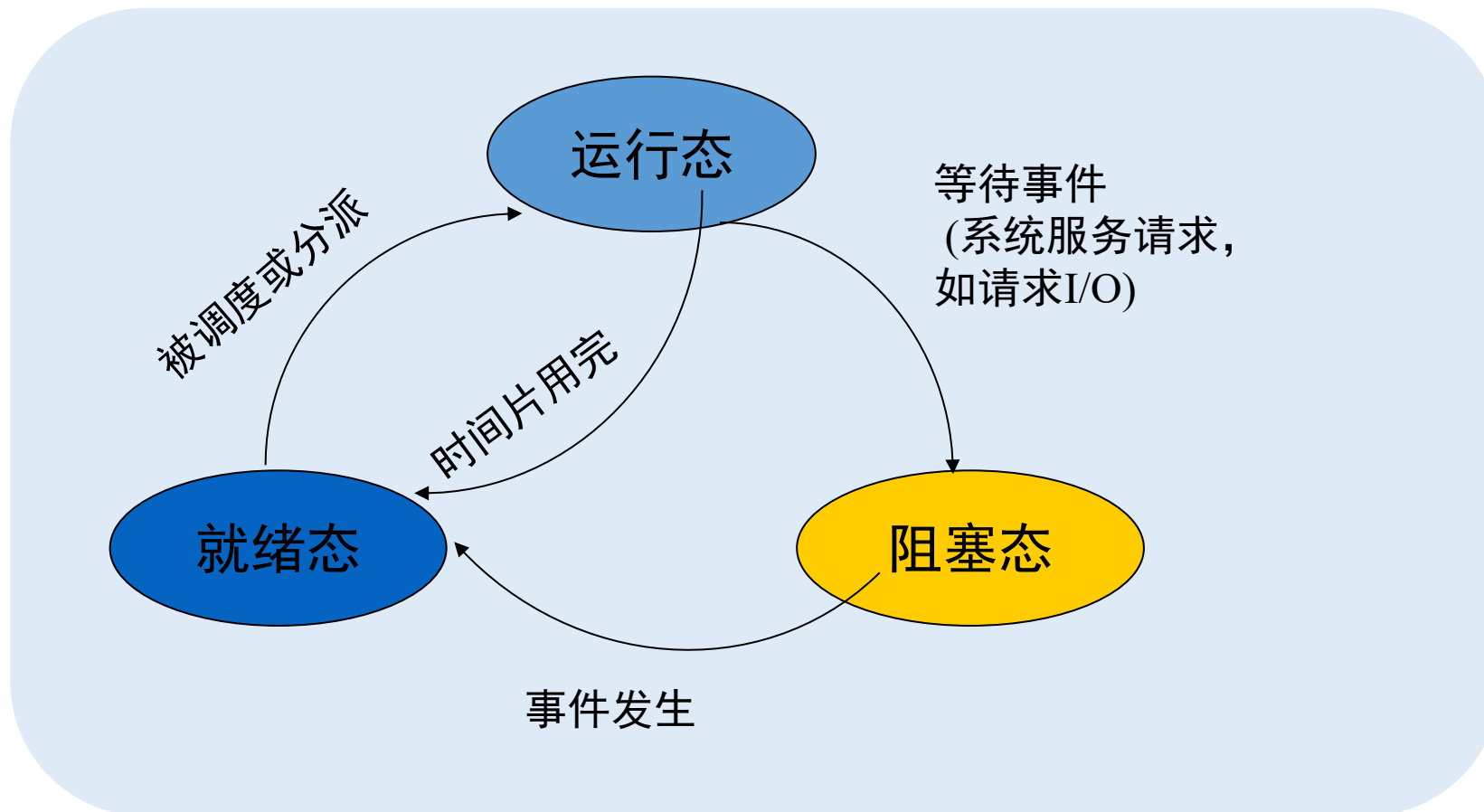
进程的整个生命周期中，大部分时间都处于三种基本状态



进程PCB中，会有一个变量state来表示进程的当前状态。如：1表示创建态、2表示就绪态、3表示运行态…
为了对同一个状态下的各个进程进行统一的管理，操作系统会将各个进程的PCB组织起来。



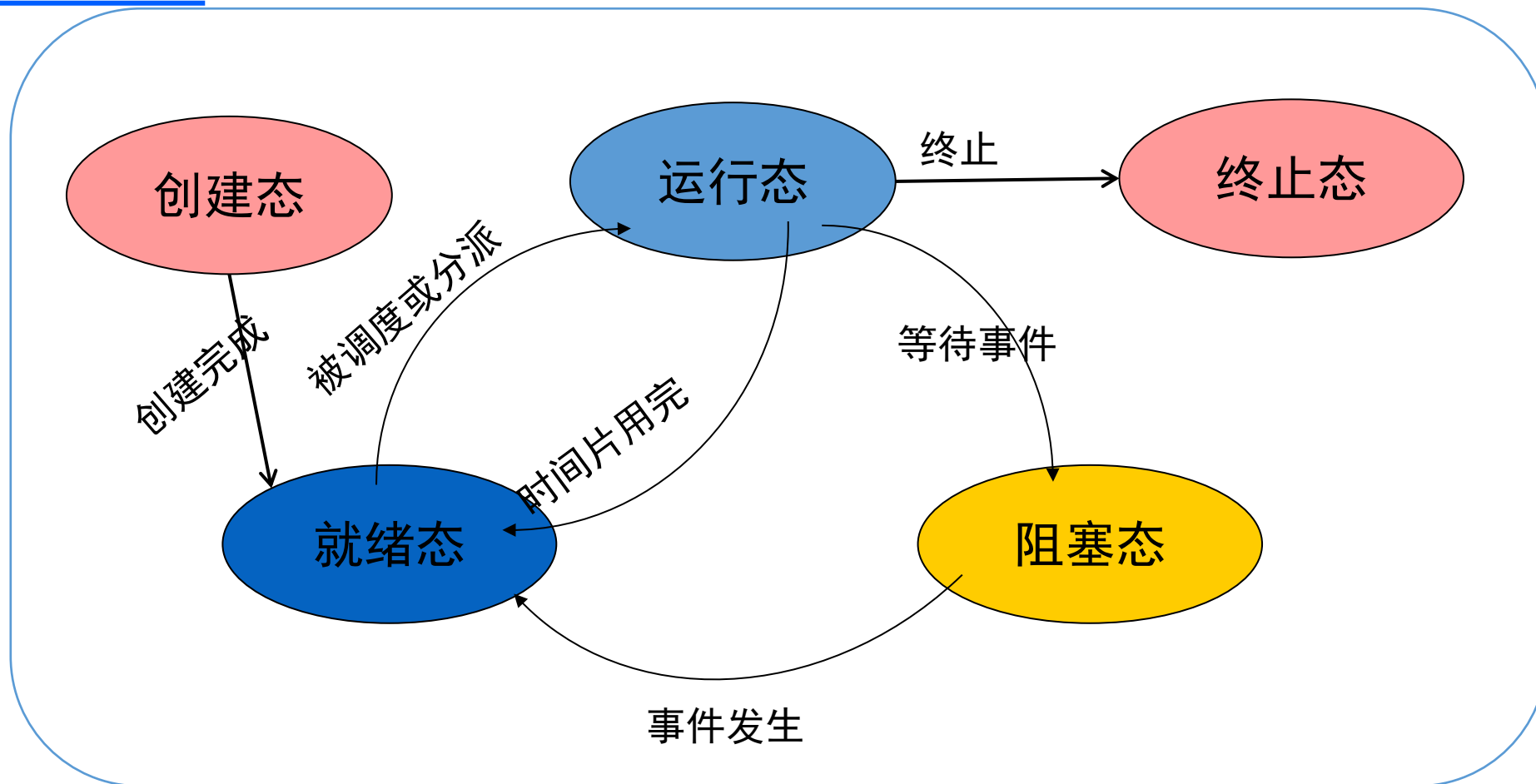
进程的状态——进程状态的转换



宿船长 B站专用



进程的状态——进程状态的转换



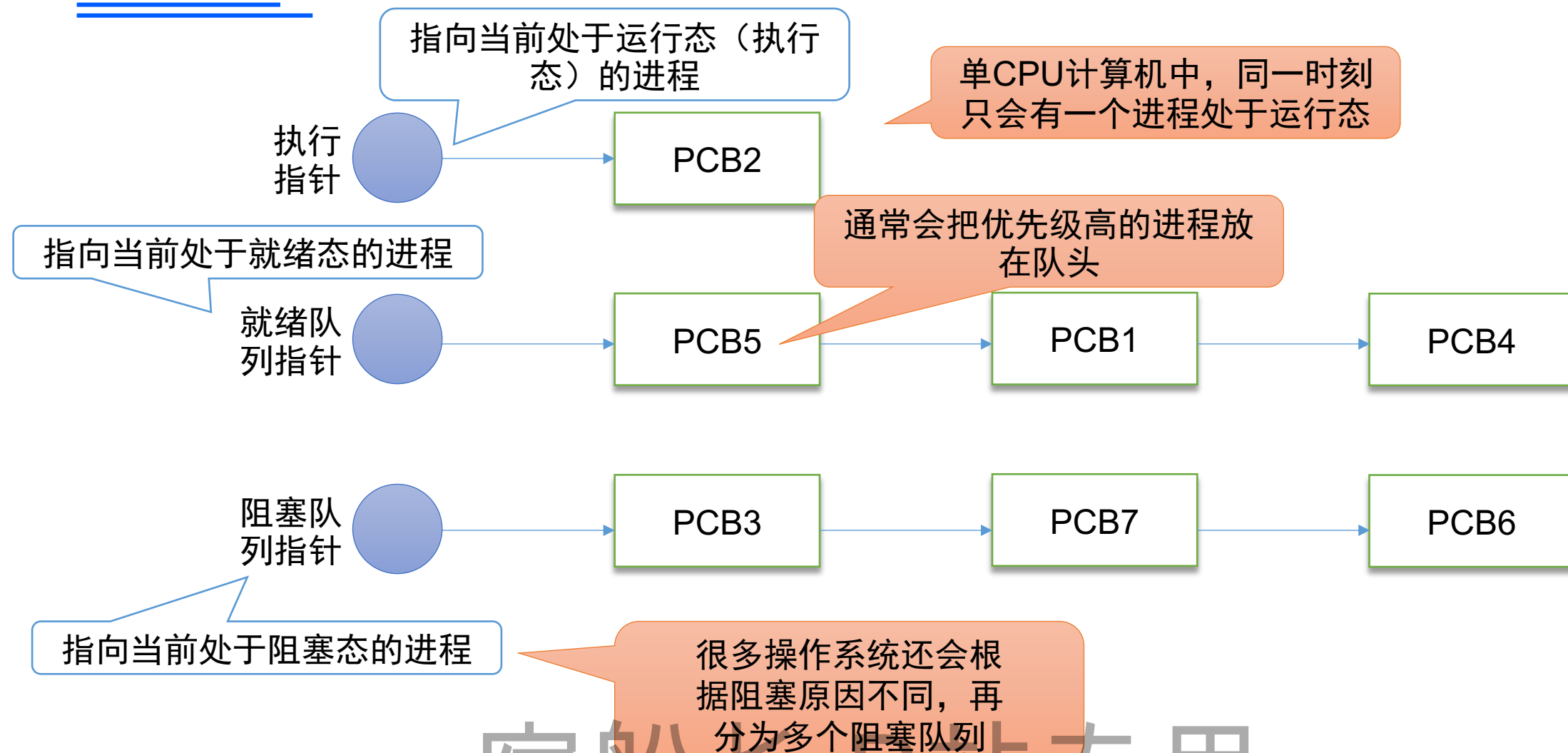
宿船长 B站专用

进程基础

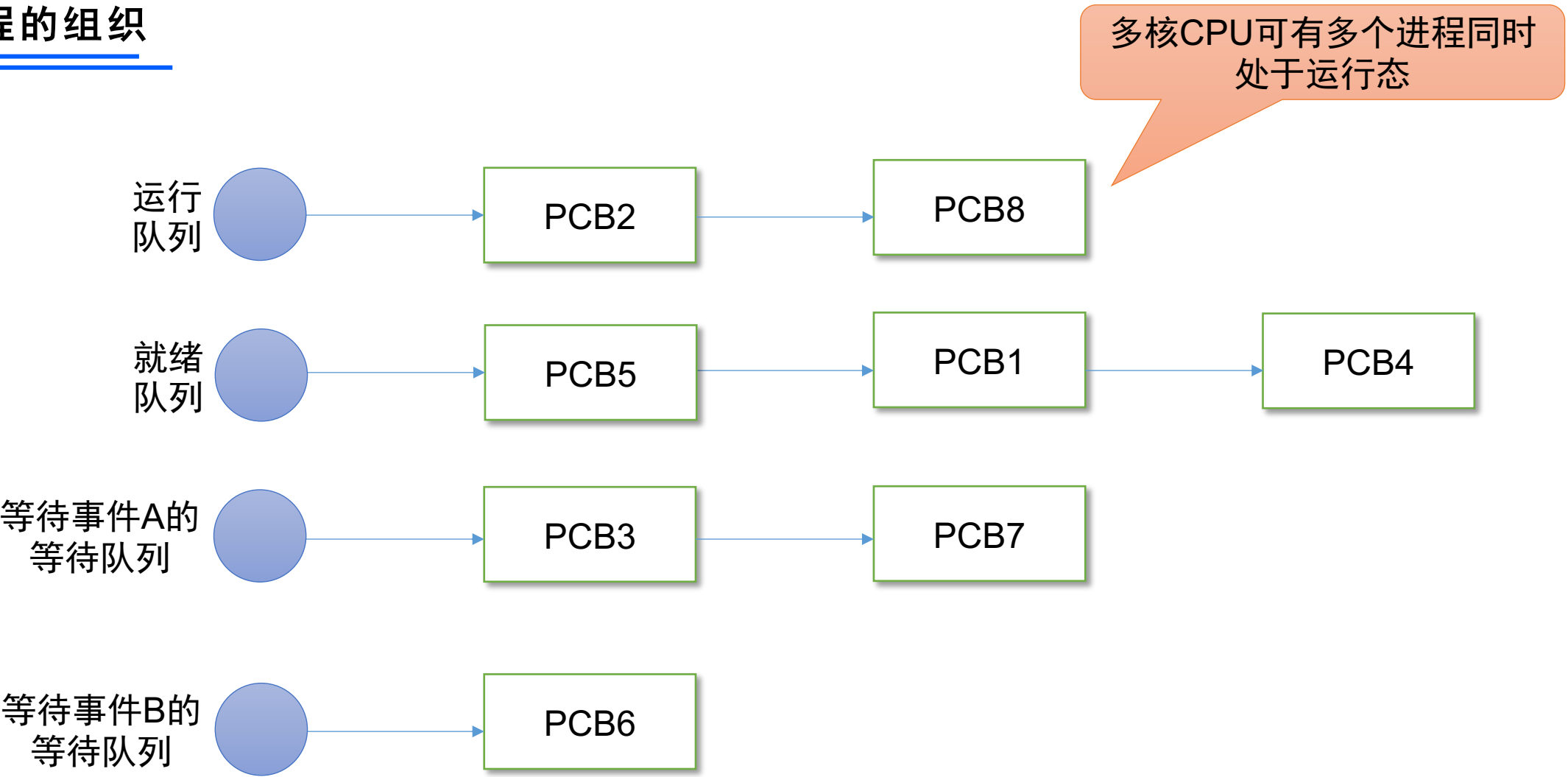
进程组织

宿船长 B站专用

进程的组织——链接方式

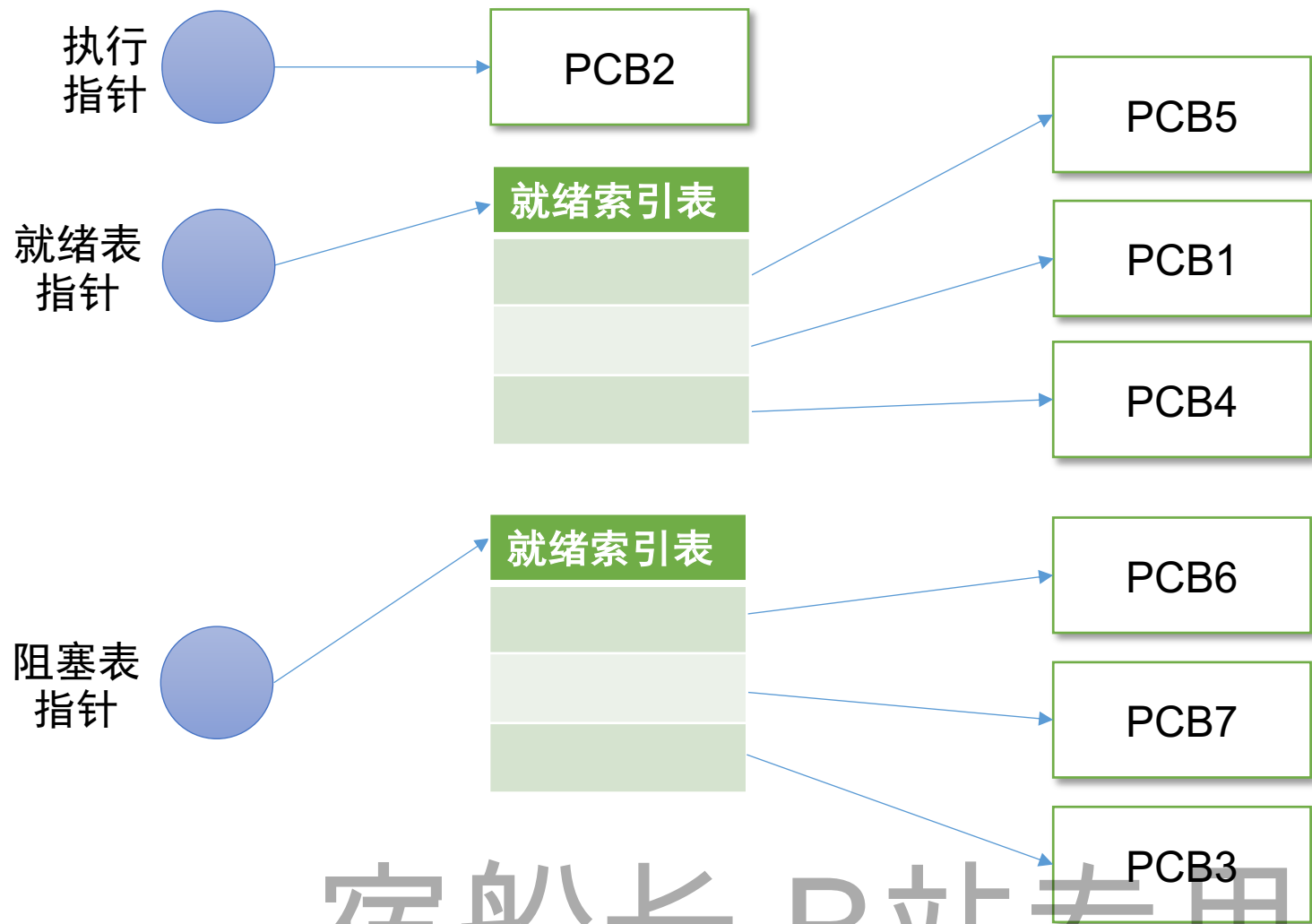


进程的组织



宿船长 B站专用

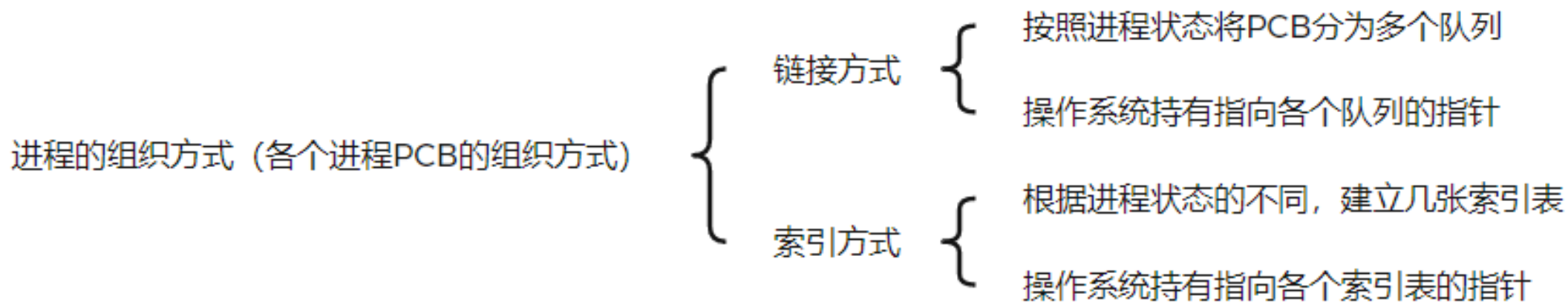
进程的组织——索引方式



宿船长 B站专用



进程的组织



进程基础

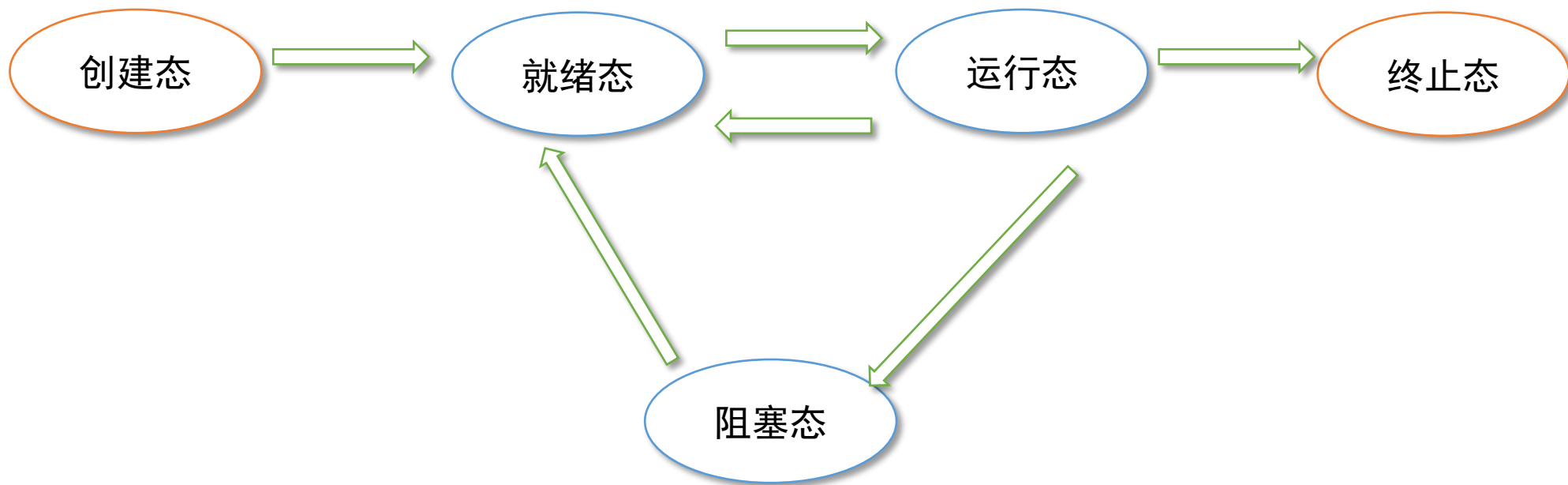
进程控制

宿船长 B站专用

进程控制——什么是进程控制？

进程控制的主要功能是对系统中的所有进程实施有效的管理，它具有创建新进程、撤销已有进程、实现进程状态转换等功能。

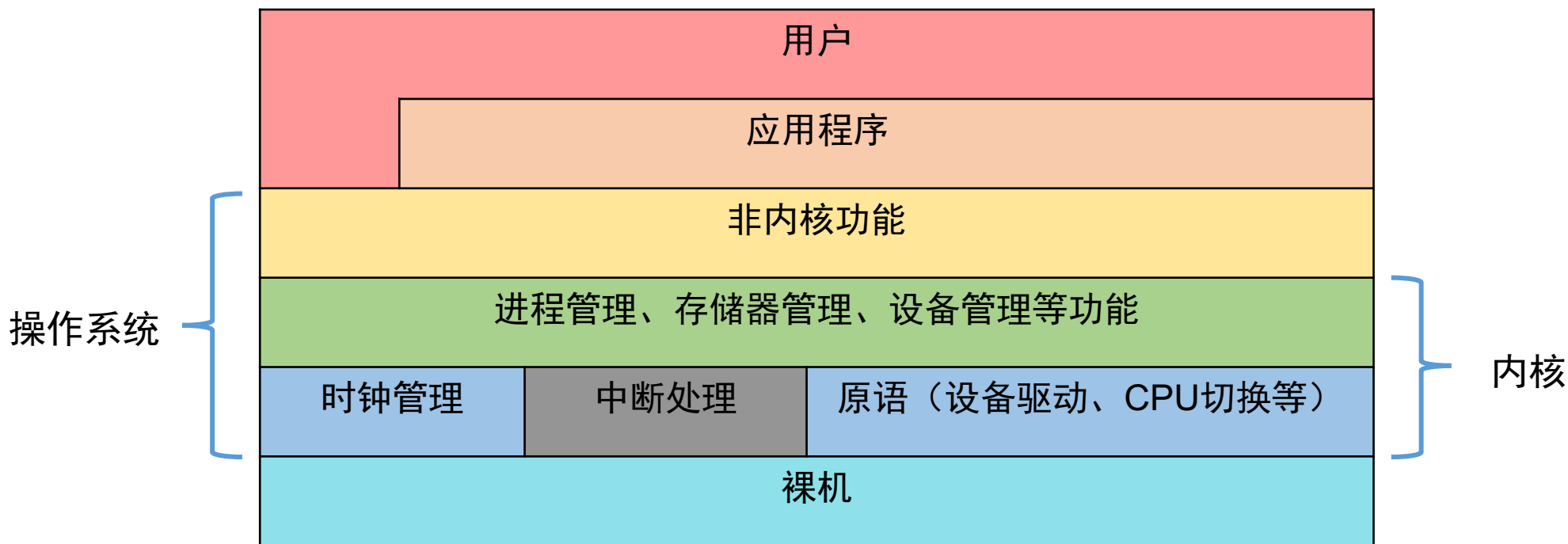
简化理解：反正进程控制就是要实现进程状态转换





进程控制——如何实现进程控制？

用“原语”实现



计算机系统的层次结构

宿船长 B站专用



进程控制——如何实现进程控制？

用“原语”实现

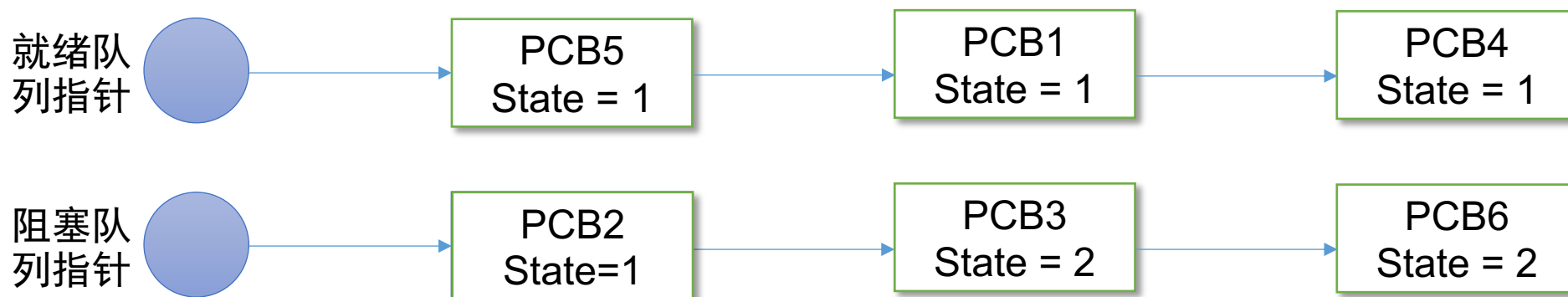
Q1:为什么要用“原语”实现？

答：原语的执行具有“原子性”，一气呵成

Q2:为何进程控制（状态转换）的过程要“一气呵成”？

答：如果不能“一气呵成”，就有可能导致操作系统中的某些关键数据结构信息不统一的情况，这会影响操作系统进行别的管理工作

Eg: 假设PCB中的变量state表示进程当前所处状态，1表示就绪态，2表示阻塞态...



假设此时进程2等待的事件发生，则操作系统中，负责进程控制的内核程序至少需要做这样两件事：

- ①将PCB2的state设为1
- ②将PCB2从阻塞队列放到就绪队列

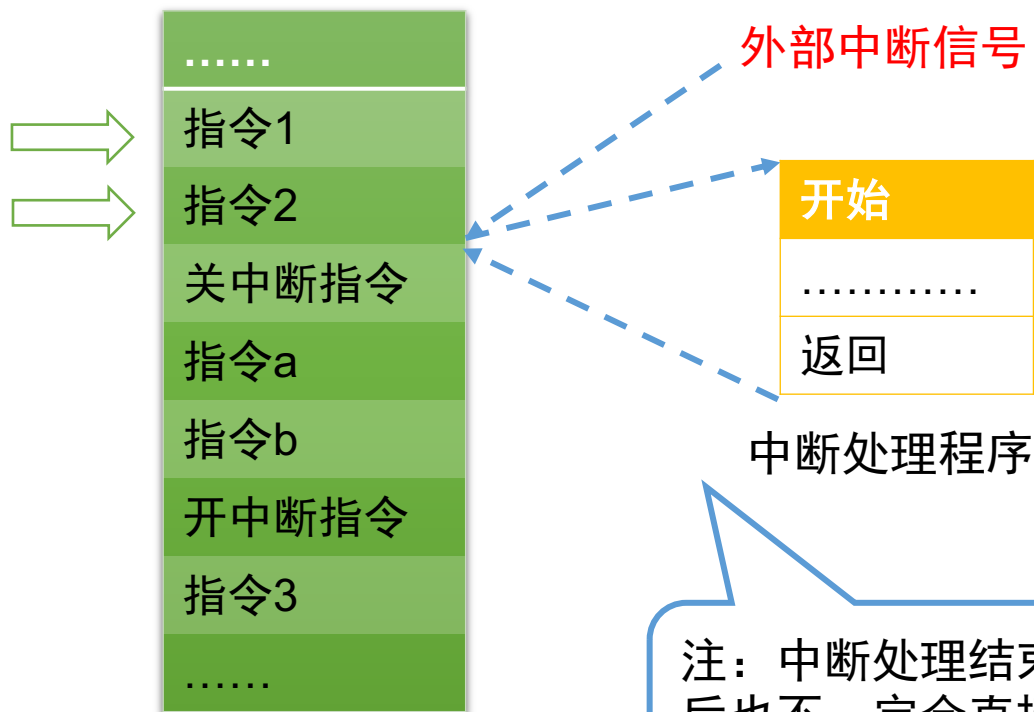
完成了第一步后收到中断信号，那么PCB2的state=1，但是它却被放在阻塞队列里

借船长B站专用



进程控制——如何实现原语的“原子性”？

原语的执行具有原子性，即执行过程只能一气呵成，期间不允许被中断。
可以用“关中断指令”和“开中断指令”这两个特权指令实现原子性



正常情况：CPU每执行完一条指令都会例行检查是否有中断信号需要处理，如果有，则暂停运行当前这段程序，转而执行相应的中断处理程序。

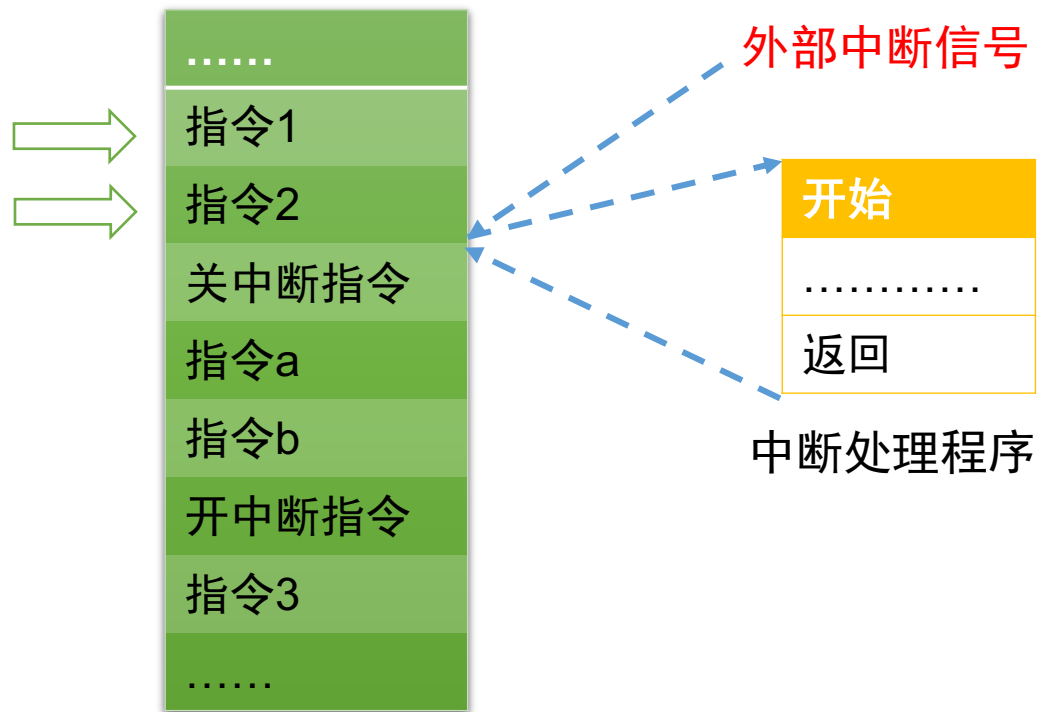
(内核程序，运行在核心态)

注：中断处理结束之后也不一定会直接回到原进程执行

进程控制——如何实现原语的“原子性”？

原语的执行具有原子性，即执行过程只能一气呵成，期间不允许被中断。

可以用“关中断指令”和“开中断指令”这两个特权指令实现原子性



CPU执行了关中断指令之后，就不再例行检查中断信号，直到执行开中断指令之后才会恢复检查。

这样，关中断、开中断之间的这些指令序列就是不可被中断的，这就实现了“原子性”

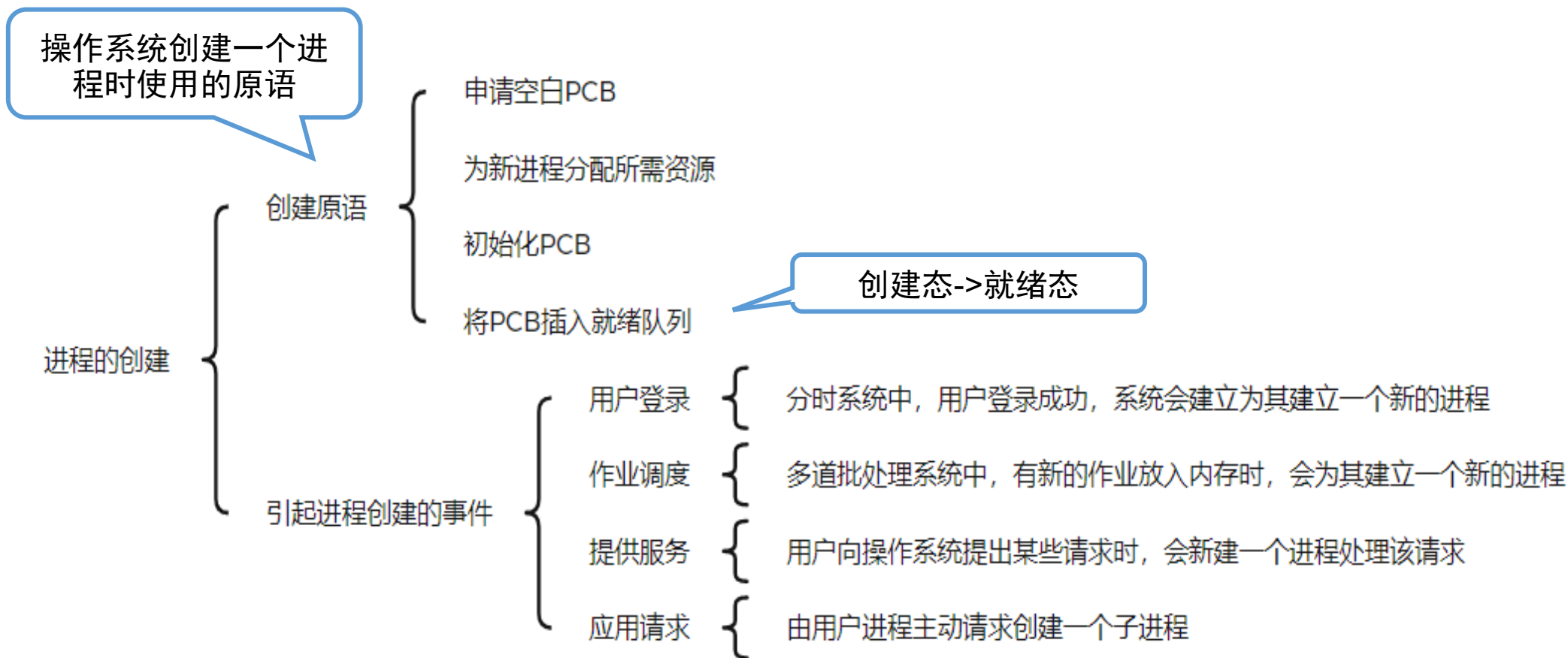
思考：如果这两个特权指令允许用户程序使用的话，会发生什么情况？

(内核程序，运行在核心态)

宿船长 B站专用

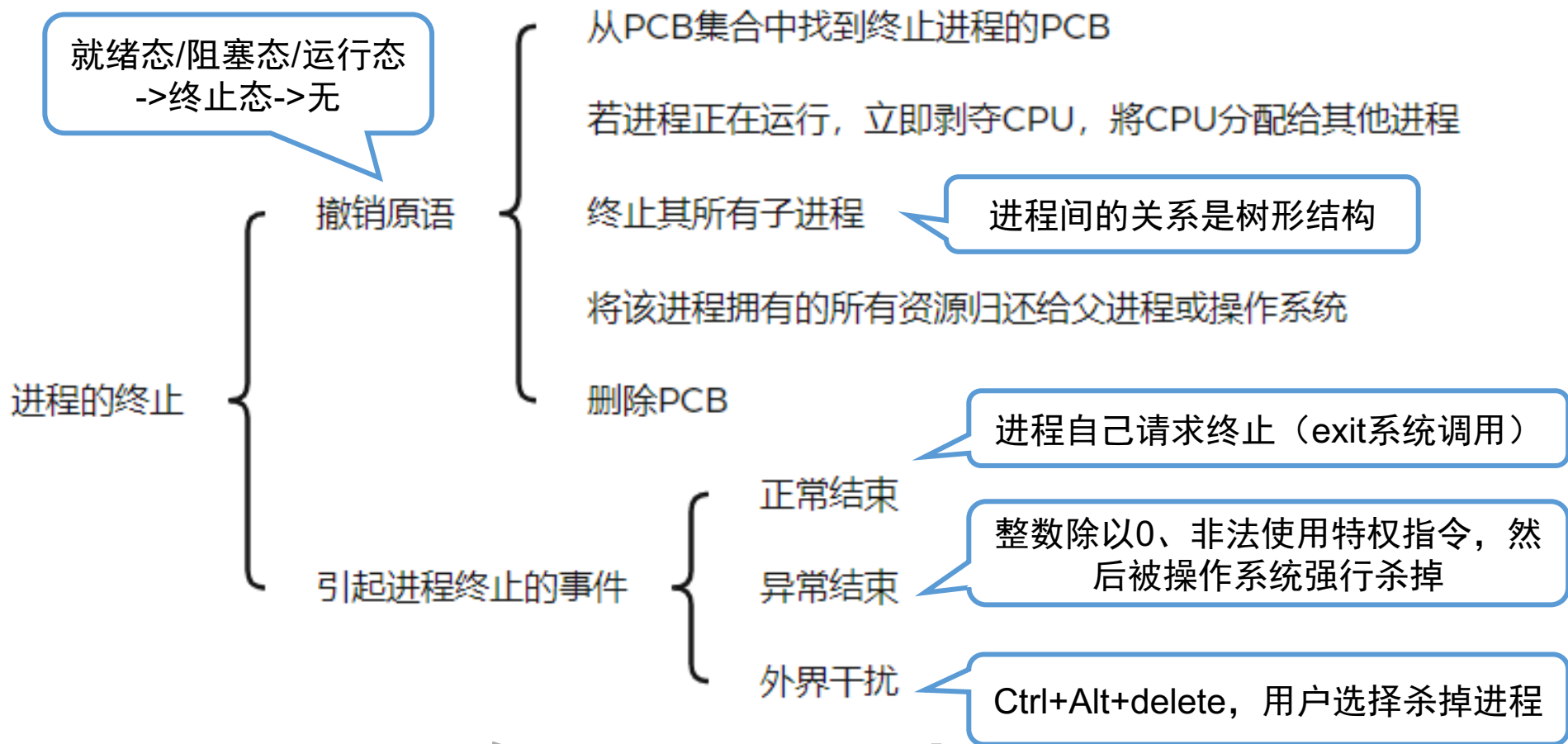


进程控制——进程控制的原语





进程控制——进程控制的原语



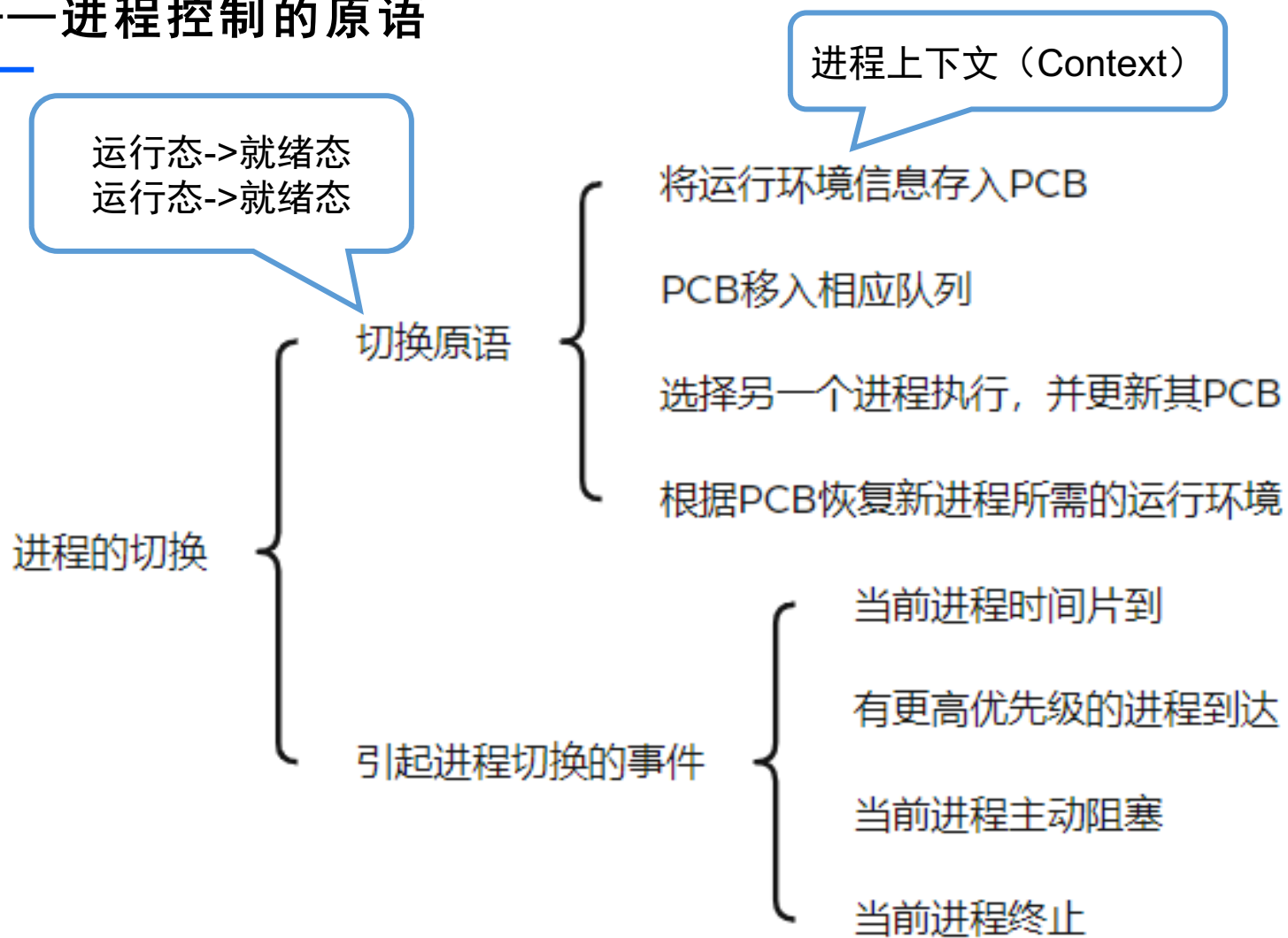


进程控制——进程控制的原语





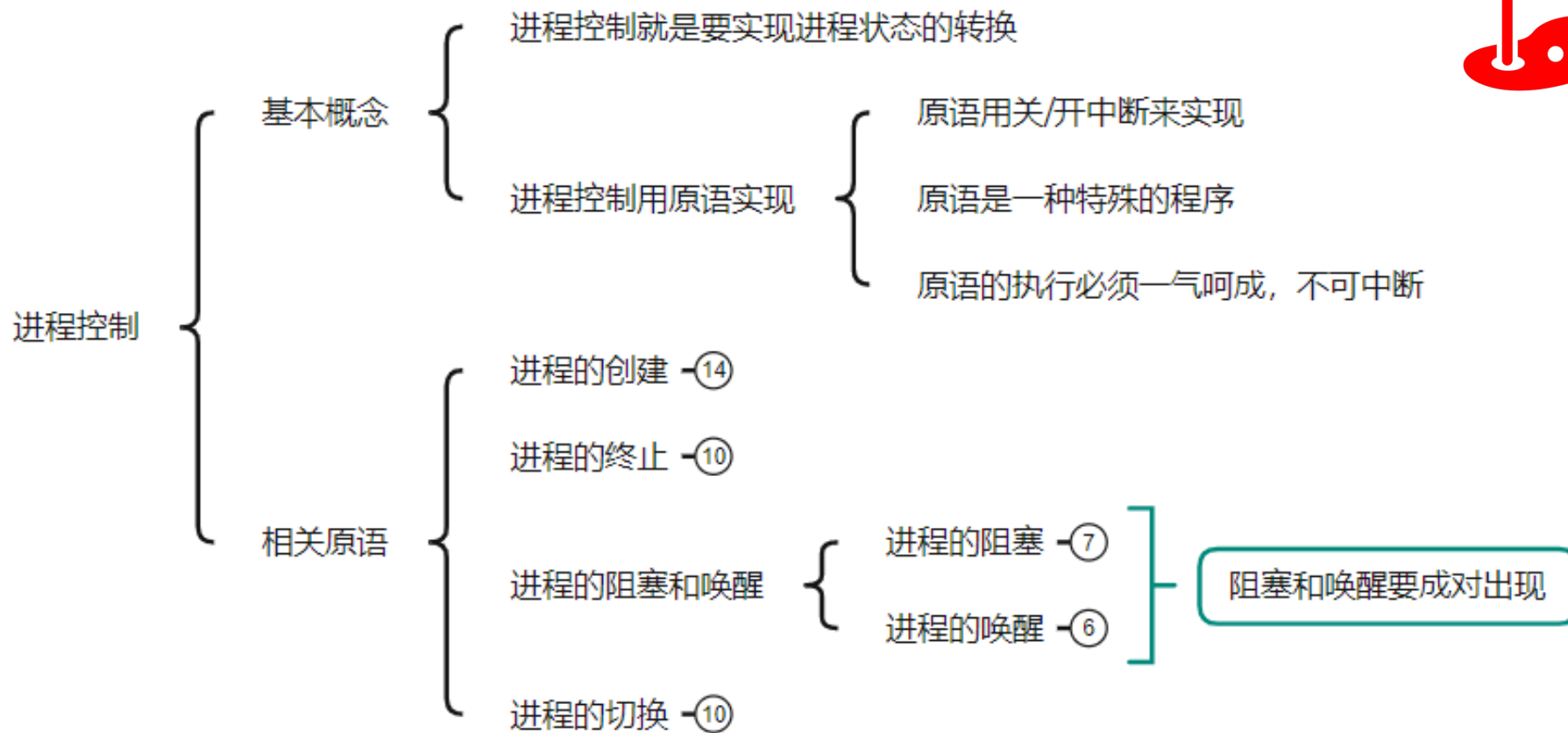
进程控制——进程控制的原语



宿船长 B站专用



进程控制



无论哪个进程控制原语，要做的无非三类事情：

1. 更新PCB中的信息
2. 将PCB插入合适的队列
3. 分配/回收资源

修改进程状态 (state)
保存/恢复运行环境

宿船长 B站专用

进程控制——进程控制的相关原语

学习技巧：进程控制会导致进程状态的转换。无论哪个进程控制原语，要做的无非三类事情：

1. 更新PCB中的信息
 - a. 所有的进程控制原语一定都会修改进程状态标志
 - b. 剥夺当前运行进程的CPU使用权必然需要保存其运行环境
 - c. 某进程开始运行前必然要恢复期运行环境
2. 将PCB插入合适的队列
3. 分配/回收资源

进程基础

进程调度

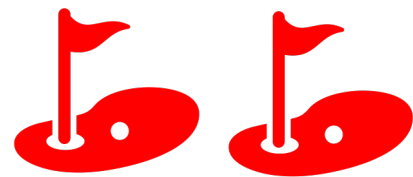
宿船长 B站专用

处理机调度——调度的基本概念

举个调度的例子：

银行处理用户服务时，普通用户先来先服务；VIP用户可优先被服务

当有一堆任务要处理，但由于资源有限，这些事情没法同时处理。这就需要确定**某种规则**来**决定**处理这些任务的**顺序**，这就是“调度”研究的问题。



进程调度的时机

进程调度（低级调度），就是按照某种算法从就绪队列中选择一个进程为其分配处理机。

需要进行进程调度与切换的情况

当前运行的进程**主动放弃**处理机

进程正常终止
运行过程中发生异常而终止
进程主动请求阻塞（如等待I/O）

当前运行的进程**被动放弃**处理机

分给进程的时间片用完
有更紧急的事需要处理（如I/O中断）
有更高优先级的进程进入就绪队列

不能进行进程调度与切换的情况

1. 在**处理中断的过程中**。中断处理过程复杂，与硬件密切相关，很难做到在中断处理过程中进行进程切换。
2. 进程在**操作系统内核程序临界区**中。
3. 在**原子操作过程中**（原语）。原子操作不可中断，要一气呵成（如之前讲过的修改PCB中进程状态标志，并把PCB放到相应队列）

宿船长 B站专用

进程调度的时机

进程在**操作系统内核程序临界区**中**不能**进行调度与切换



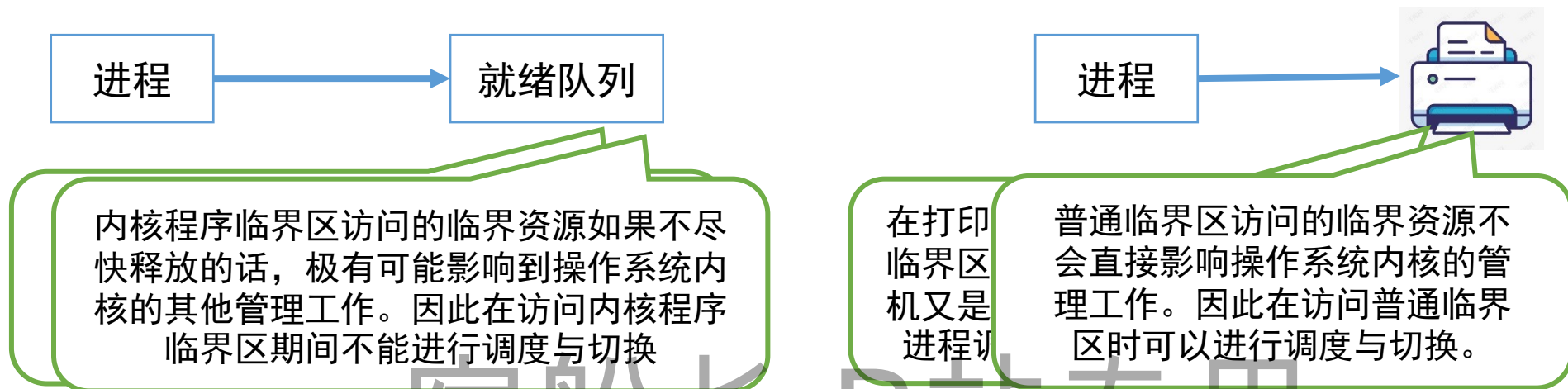
(2012年联考真题) 进程处于**临界区**时**不能**进行处理机调度



临界资源：一个时间段内只允许一个进程使用的资源。各进程需要**互斥地**访问临界资源。

临界区：访问临界资源的那段代码。

内核程序临界区一般是用来访问**某种内核数据结构的**，比如进程的就绪队列（由各就绪进程的PCB组成）



进程调度的时机

进程调度（低级调度），就是按照某种算法从就绪队列中选择一个进程为其分配处理机。

有的系统中，只允许进程主动放弃处理机

需要进行进程调度与切换的情况

当前运行的进程**主动放弃**处理机

进程正常终止
运行过程中发生异常而终止
进程主动请求阻塞（如等待I/O）

当前运行的进程**被动放弃**处理机

分给进程的时间片用完
有更紧急的事需要处理（如I/O中断）
有更高优先级的进程进入就绪队列

有的系统中，进程可以主动放弃处理机，当有更紧急的任务需要处理时，也会强行剥夺处理机（被动放弃）

不能进行进程调度与切换的情况

1. 在**处理中断的过程中**。中断处理过程复杂，与硬件密切相关，很难做到在中断处理过程中进行进程切换。
2. 进程在**操作系统内核程序临界区**中。
3. 在**原子操作过程中**（原语）。原子操作不可中断，要一气呵成（如之前讲过的修改PCB中进程状态标志，并把PCB放到相应队列）

但是进程在普通临界区中是可以进行调度、切换的。



进程调度的方式

非剥夺调度方式，又称**非抢占方式**。即，只允许进程主动放弃处理机。在运行过程中即便有更紧迫的任务到达，当前进程依然会继续使用处理机，直到该进程终止或主动要求进入阻塞态。

实现简单，系统开销小但是无法及时处理紧急任务，适合于早期的批处理系统

剥夺调度方式，又称**抢占方式**。当一个进程正在处理机上执行时，如果有一个更重要或更紧迫的进程需要使用处理机，则立即暂停正在执行的进程，将处理机分配给更重要紧迫的那个进程。

可以优先处理更紧急的进程，也可实现让各进程按时间片轮流执行的功能（通过时钟中断）。适合于分时操作系统、实时操作系统



进程的切换与过程

“狭义的进程调度”与“进程切换”的区别：

狭义的进程调度指的是从就绪队列中**选中一个要运行的进程**。（这个进程可以是刚刚被暂停执行的进程，也可能是**另一个进程**，后一种情况就需要**进程切换**）

进程切换是指一个进程让出处理机，由另一个进程占用处理机的过程。

广义的进程调度包含了选择一个进程和进程切换两个步骤。

进程切换的过程主要完成了：

1. 对原来运行进程各种数据的保存
2. 对新的进程各种数据的恢复

（如：程序计数器、程序状态字、各种数据寄存器等处理机现场信息，这些信息一般保存在进程控制块）

注意：**进程切换是有代价的**，因此如果**过于频繁的**进行进程**调度、切换**，必然会使整个**系统的效率降低**，使系统大部分时间都花在了进程切换上，而真正用于执行进程的时间减少。



调度算法的评价指标——CPU利用率

由于早期的CPU造价极其昂贵，因此人们会希望让CPU尽可能多地工作

CPU利用率：指CPU“忙碌”的时间占总时间的比例。

$$\text{利用率} = \frac{\text{忙碌的时间}}{\text{总时间}}$$

eg. 某计算机只支持单道程序，某个作业刚开始需要在CPU上运行5秒，再用打印机打印输出5秒，之后再执行5秒，才能结束。在此过程中，CPU利用率、打印机利用率分别是多少？

$$\text{CPU利用率} = \frac{5 + 5}{5 + 5 + 5} = 66.66\%$$

$$\text{打印机利用率} = \frac{5}{15} = 33.33\%$$



调度算法的评价指标——系统吞吐量

对于计算机来说，希望能用尽可能少的时间处理完尽可能多的作业

系统吞吐量：单位时间内完成作业的数量

$$\text{系统吞吐量} = \frac{\text{总共完成了多少道作业}}{\text{总共花了多少时间}}$$

eg. 某计算机系统处理完10道作业，共花费100秒，则系统吞吐量为？

$$10/100 = 0.1 \text{道/秒}$$



调度算法的评价指标——周转时间

对于计算机的用户来说，他很关心自己的作业从提交到完成花了多少时间。

周转时间，是指从**作业被提交给系统开始**，到**作业完成为止**的这段时间间隔。

它包括四个部分：作业在外存后备队列上等待作业调度（高级调度）的时间、进程在就绪队列上等待进程调度（低级调度）的时间、进程在CPU上执行的时间、进程等待I/O操作完成的时间。后三项在一个作业的整个处理过程中，可能发生多次。

(作业)**平均周转时间** = 作业完成时间 - 作业提交时间

$$\text{平均周转时间} = \frac{\text{各作业周转时间之和}}{\text{作业数}}$$

对于用户来说，更关心自己的单个作业的周转时间

对于操作系统来说，更关心系统的整体表现，因此更关心所有作业周转时间的平均值

调度算法的评价指标——带权周转时间



带权周转时间必然 ≥ 1

带权周转时间与周转时间都是
越小越好

$$\text{带权周转时间} = \frac{\text{作业周转时间}}{\text{作业实际运行的时间}} = \frac{\text{作业完成时间} - \text{作业提交时间}}{\text{作业实际运行的时间}}$$

$$\text{平均带权周转时间} = \frac{\text{各作业周转时间之和}}{\text{作业数}}$$

对于周转时间相同的两个作业，实际运行时间长的作业在相同时间内被服务的时间更多，带权周转时间更小，用户满意度更高。

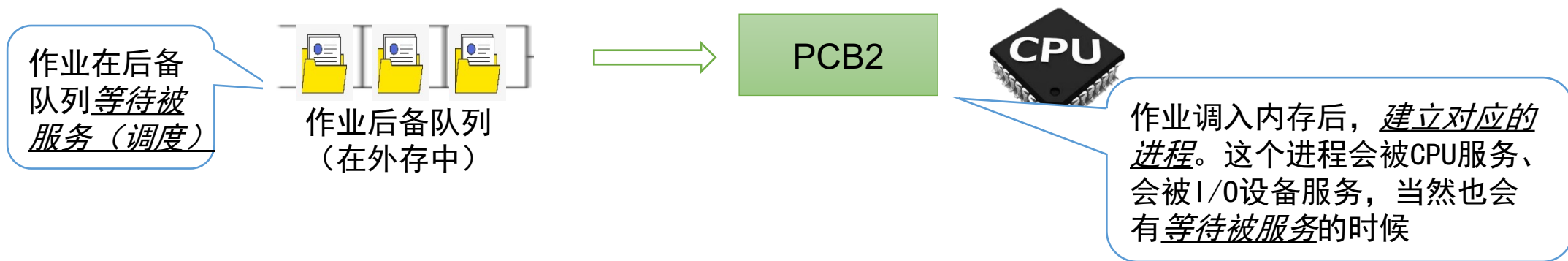
对于实际运行时间相同的两个作业，周转时间短的带权周转时间更小，用户满意度更高。



调度算法的评价指标——等待时间

计算机的用户希望自己的作业尽可能少的等待处理机

等待时间，指进程/作业处于等待处理机状态时间之和，等待时间越长，用户满意度越低。



对于**进程**来说，等待时间就是指进程建立后**等待被服务的时间之和**，在等待I/O完成的期间其实进程也是在被服务的，所以不计入等待时间。

对于**作业**来说，不仅要考虑**建立进程后的等待时间**，还要加上作业在外存后备队列中等待的时间。

一个作业总共需要被CPU服务多久，被I/O设备服务多久一般是确定不变的，因此调度算法其实只会影响作业/进程的等待时间。当然，与前面指标类似，也有“**平均等待时间**”来评价整体性能。



调度算法的评价指标——响应时间

对于计算机用户来说，会希望自己的提交的请求（比如通过键盘输入了一个调试命令）尽早地开始被系统服务、回应。

响应时间，指从用户**提交请求**到**首次产生响应**所用的时间。



调度算法——先来先服务 (FCFS, First Come First Serve)

作业名	提交时间	要求执行时间	开始执行时间	完成时间	周转时间	带权周转时间
A	1.0	2.0	1.0	3.0	2.0	1.0
B	1.2	3.0	3.0	6.0	4.8	1.6
C	1.4	1.2	6.0	7.2	5.8	4.8
D	1.5	0.3	7.2	7.5	6.0	20.0
平均周转时间： $T = (2.0 + 4.8 + 5.8 + 6.0) / 4 = 4.65$				平均带权周转时间： $W = (1 + 1.6 + 4.8 + 20) / 4 = 6.85$		

- 优点：实现简单；
- 缺点：对长作业有利，对短作业不利；
 - 平均周转时间可能较长；没有考虑任务的紧迫性

调度算法——先来先服务 (FCFS, First Come First Serve)

FCFS

算法思想

主要从“公平”的角度考虑（类似于我们生活中排队买东西的例子）

算法规则

按照作业/进程到达的先后顺序进行服务

用于作业/进程调度

用于作业调度时，考虑的是哪个作业先到达后备队列；
用于进程调度时，考虑的是哪个进程先到达就绪队列

是否可抢占？

非抢占式的算法

优缺点

优点：公平、算法实现简单

缺点：排在长作业（进程）后面的短作业需要等待很长时间，带权周转时间很大，对短作业来说用户体验不好。即，FCFS算法对长作业有利，对短作业不利（eg：排队买奶茶…）

某进程/作业长期得不到服务

是否会导致饥饿？

不会

宿船长 B站专用



调度算法——短作业优先 (SJF, Shortest Job First)

算法分类： 非抢占式调度算法： 短进程优先调度算法 (SPF)

抢占式调度算法： 最短剩余时间优先算法 (SRTN)

宿船长 B站专用



调度算法——短作业优先 (SJF, Shortest Job First)

非抢占式调度算法：短进程优先调度算法 (SPF)

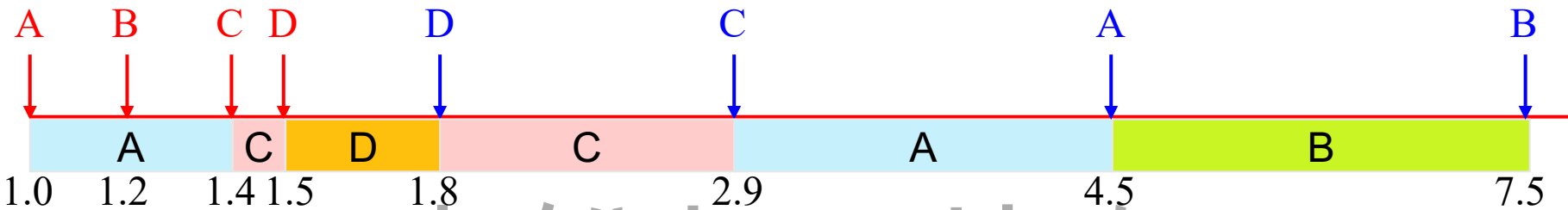
作业名	提交时间	要求执行时间	开始时间	完成时间	周转时间	带权周转时间
A	1.0	2.0	1.0	3.0	2.0	1.0
D	1.5	0.3	3.0	3.3	1.8	6.0
C	1.4	1.2	3.3	4.5	3.1	2.6
B	1.2	3.0	4.5	7.5	6.3	2.1
平均周转时间： $T = (2.0 + 1.8 + 3.1 + 6.3) / 4 = 3.3$				平均带权周转时间： $W = (1 + 6 + 2.6 + 2.1) / 4 = 2.93$		



调度算法——短作业优先 (SJF, Shortest Job First)

抢占式调度算法：最短剩余时间优先算法 (SRTN)

作业名	提交时间	要求执行时间	开始时间	完成时间	周转时间	带权周转时间
A	1.0	2.0	1.0	4.5	3.5	1.75
B	1.2	3.0	4.5	7.5	6.3	2.1
C	1.4	1.2	1.4	2.9	1.5	1.25
D	1.5	0.3	1.5	1.8	0.3	1.0
平均周转时间： $T = (3.5 + 6.3 + 1.5 + 0.3) / 4 = 2.9$				平均带权周转时间： $W = (1.75 + 2.1 + 1.25 + 1) / 4 = 1.53$		



宿船长 B站专用



调度算法——短作业优先 (SJF, Shortest Job First)

注意几个小细节:

1. 如果题目中**未特别说明**, 所提到的“短作业/进程优先算法”默认是**非抢占式**的
2. 很多书上都会说“SJF调度算法的平均等待时间、平均周转时间最少”

严格来说, 这个表述是错误的, 不严谨的。之前的例子表明, 最短剩余时间优先算法得到的平均等待时间、平均周转时间还要更少

应该加上一个条件“在**所有进程同时可运行**时, 采用SJF调度算法的平均等待时间、平均周转时间最少”;

或者说“在**所有进程都几乎同时到达**时, 采用SJF调度算法的平均等待时间、平均周转时间最少”; 如果不加上述前提条件, 则应该说“**抢占式**的短作业/进程优先调度算法 (**最短剩余时间优先**, **SRNT**算法) 的平均等待时间、平均周转时间最少”

3. 虽然严格来说, SJF的平均等待时间、平均周转时间并不一定最少, 但相比于其他算法 (如FCFS), SJF依然可以获得较少的平均等待时间、平均周转时间
4. 如果选择题中遇到“SJF算法的平均等待时间、平均周转时间最少”的选项, 那最好判断其他选项是不是有很明显的错误, 如果没有更合适的选项, 那也应该选择该选项

调度算法——短作业优先 (SJF, Shortest Job First)

SJF

- 算法思想 追求最少的平均等待时间, 最少的平均周转时间、最少的平均平均带权周转时间
- 算法规则 最短的作业/进程优先得到服务 (所谓“最短”, 是指要求服务时间最短)
- 用于作业/进程调度 即可用于作业调度, 也可用于进程调度。用于进程调度时称为“短进程优先 (SPF, Shortest Process First) 算法”
- 是否可抢占? SJF和SPF是**非抢占式**的算法。但是**也有抢占式的版本——最短剩余时间优先算法 (SRTN, Shortest Remaining Time Next)**
- 优缺点
 - 优点: “最短的” 平均等待时间、平均周转时间
 - 缺点: 不公平。**对短作业有利, 对长作业不利**。可能产生**饥饿现象**。另外, 作业/进程的运行时间是由用户提供的, 并不一定真实, 不一定能做到真正的短作业优先
- 是否会导致饥饿? 会。如果源源不断地有短作业/进程到来, 可能使长作业/进程长时间得不到服务, 产生“**饥饿**”现象。如果一直得不到服务, 则称为“**饿死**”

调度算法——高响应比优先（HRRN）

$$\text{响应比} = \frac{\text{要求服务时间} + \text{等待时间}}{\text{要求服务时间}} = 1 + \frac{\text{等待时间}}{\text{要求服务时间}}$$

按高响应比优先调度算法进行调度，给出下述作业的调度顺序，并计算平均周转时间。要求写出计算过程。



作业名	提交时间	要求执行时间
A	1.0	2.0
B	1.2	3.0
C	1.4	1.2
D	1.5	0.3

顺序：A,D,C,B

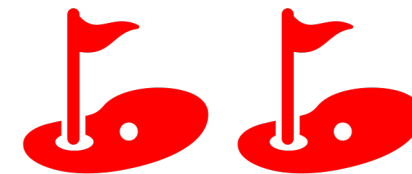
调度算法——高响应比优先 (HRRN, Highest Response Ratio Next)

HRRN

算法思想	要综合考虑作业/进程的等待时间和要求服务的时间
算法规则	在每次调度时先计算各个作业/进程的 响应比 ，选择 响应比最高的 作业/进程为其服务
用于作业/进程调度	即可用于作业调度，也可用于进程调度
是否可抢占?	非抢占式的算法。因此只有当前运行的作业/进程主动放弃处理机时，才需要调度，才需要计算响应比
优缺点	综合考虑了等待时间和运行时间（要求服务时间） 等待时间相同时，要求服务时间短的优先（SJF的优点） 要求服务时间相同时，等待时间长的优先（FCFS的优点） 对于长作业来说，随着等待时间越来越久，其响应比也会越来越大，从而避免了长作业饥饿的问题
是否会导致饥饿?	不会

响应比≥1

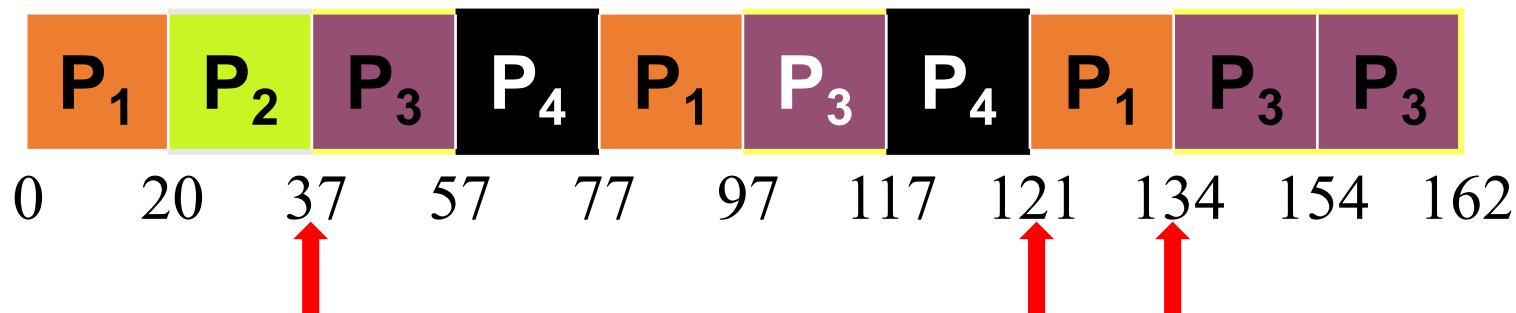
响应比 = $\frac{\text{要求服务时间} + \text{等待时间}}{\text{要求服务时间}}$:



调度算法——时间片轮转 (RR, Round-Robin)



假设一个系统采用时间片轮转调度算法，时间片长度为20ms，4个进程P₁，P₂，P₃，P₄依次进入系统，各进程所需要的运行时间为：53ms，17ms，68ms，24ms，计算其平均周转时间。



平均周转时间：(134 + 37 + 162 + 121) / 4 = 113.5ms

SJF的平均周转时间：(94 + 17 + 162 + 41) / 4 = 78.5ms

宿船长B站专用

调度算法——时间片轮转 (RR, Round-Robin)

时间片大小的确定:

N为就绪队列中进程数, T为系统响应时间, q为时间片

$$T=Nq$$

影响因素:

- 系统的相应时间
- 就绪进程的数量
- 进程调度及切换开销
- CPU的运行速度

调度算法——时间片轮转 (RR, Round-Robin)

RR

算法思想	公平地、轮流地为各个进程服务，让每个进程在一定时间间隔内都可以得到响应
算法规则	按照各进程到达就绪队列的顺序，轮流让各个进程执行一个时间片（如100ms）。若进程未在一个时间片内执行完，则剥夺处理机，将进程重新放到就绪队列队尾重新排队。
用于作业/进程调度	用于进程调度（只有作业放入内存建立了相应的进程后，才能被分配处理机时间片）
是否可抢占？	若进程未能在时间片内运行完，将被强行剥夺处理机使用权，因此时间片轮转调度算法属于抢占式的算法。由时钟装置发出时钟中断来通知CPU时间片已到
优缺点	优点：公平；响应快，适用于分时操作系统； 缺点：由于高频率的进程切换，因此有一定开销；不区分任务的紧急程度
是否会导致饥饿？	不会
补充	时间片太大或太小分别有什么影响？

宿船长 B站专用



调度算法——优先级调度算法

■ 优先级进程调度算法的类型：通常用一个整数表示优先级

- 非强占式优先级调度算法
- 强占式优先级调度算法

■ 优先级的设计方法：

- **静态优先级：**进程创建时确定其优先级，整个生命周期中不改变。
确定依据：进程类型；进程对资源的需求；进程的估计运行时间
- **动态优先级：**
进程创建时赋一个优先级初值，运行期间动态调整其权值。



调度算法——优先级调度算法

例： 有5个进程P1、P2、P3、P4、P5，它们同时依次进入就绪队列，其静态优先级和需要的处理机时间如下所示， 采用非抢占式的调度方式，给出调度顺序，并计算平均周转时间。

进程	处理机时间	优先级
P1	10	3
P2	1	1
P3	2	3
P4	1	4
P5	5	2

调度顺序： P2→P5→P1→P3→P4

调度算法——优先级调度算法

补充：

就绪队列未必只有一个，可以按照不同优先级来组织。另外，也可以把优先级高的进程排在更靠近队头的位置

根据优先级是否可以动态改变，可将优先级分为**静态优先级**和**动态优先级**两种。

静态优先级：创建进程时确定，之后一直不变。

动态优先级：创建进程时有一个初始值，之后会根据情况动态地调整优先级。

I/O设备和CPU可以并行工作。如果优先让I/O繁忙型进程优先运行的话，则越有可能让I/O设备尽早地投入工作，则资源利用率、系统吞吐量都会得到提升

Q1：如何合理地设置各类进程的优先级？

通常：系统进程优先级**高于**用户进程
前台进程优先级**高于**后台进程

操作系统更**偏好I/O型进程**（或称**I/O繁忙型进程**）

注：与I/O型进程相对的是**计算型进程**（或称**CPU繁忙型进程**）

Q2：如果采用的是动态优先级，什么时候应该调整？

可以从追求公平、提升资源利用率等角度考虑

如果某进程在就绪队列中等待了很长时间，则可以适当提升其优先级

如果某进程占用处理机运行了很长时间，则可适当降低其优先级

如果发现一个进程频繁地进行I/O操作，则可适当提升其优先级

调度算法——优先级调度算法

优先级调度

算法思想

随着计算机的发展，特别是实时操作系统的出现，越来越多的应用场景需要根据任务的紧急程度来决定处理顺序

算法规则

调度时选择优先级最高的作业/进程

用于作业/进程调度

既可用于作业调度，也可用于进程调度。甚至，还会用于在之后会学习的I/O调度中

是否可抢占？

抢占式、非抢占式都有。做题时的区别在于：非抢占式只需在进程主动放弃处理机时进行调度即可，而抢占式还需在就绪队列变化时，检查是否会发生抢占。

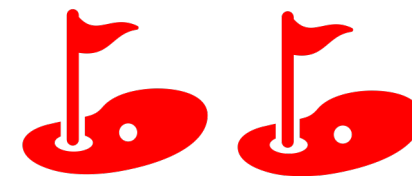
优缺点

优点：用优先级区分紧急程度、重要程度，适用于实时操作系统。可灵活地调整对各种作业/进程的偏好程度。

缺点：若源源不断地有高优先级进程到来，则可能导致饥饿

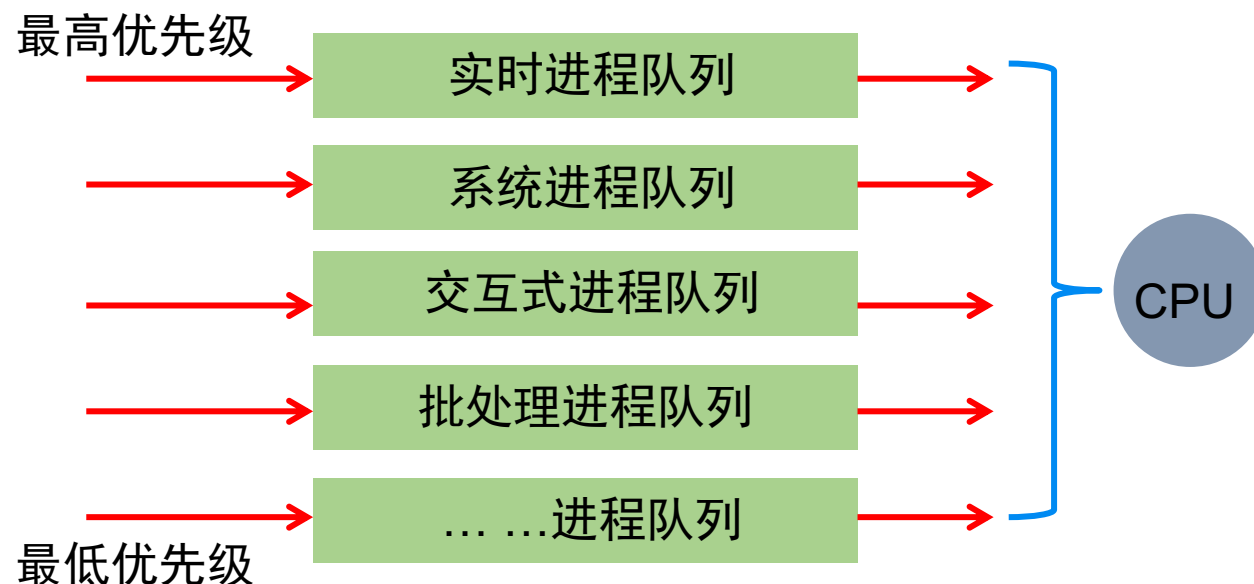
是否会导致饥饿？ 会

宿船长 B站专用

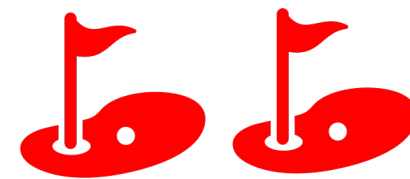


调度算法——多级反馈队列调度算法

系统中设置多个就绪队列，每个队列优先级不同；
每个队列有自己独立的进程调度算法；
一个进程依据其属性固定位于某个就绪队列中。



宿船长 B站专用



调度算法——多级反馈队列调度算法

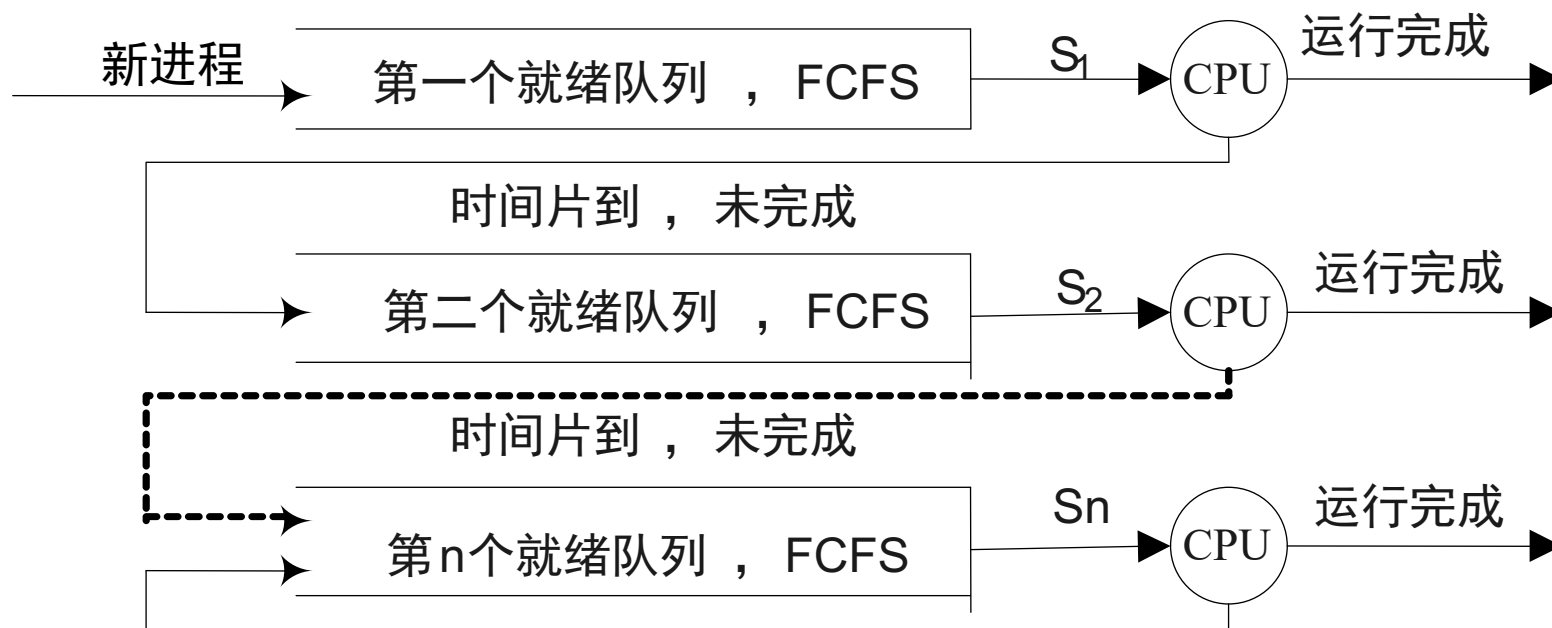
按调度级别（优先级）设置多个就绪进程队列

按优先级（或就绪队列）设置不同时间片

各级就绪队列按FCFS组织，按时间片调度，每个进程被调度后运行一个当前队列的时间片长度

最后一级按时间片轮转方式组织调度

各队列间按抢占式优先级算法调度



宿船长B站专用

调度算法——多级反馈队列调度算法

多级反馈队列

算法思想

对其他调度算法的折中权衡

算法规则

1. 设置多级就绪队列，各级队列优先级从高到低，时间片从小到大
2. 新进程到达时先进入第1级队列，按FCFS原则排队等待被分配时间片，若用完时间片进程还未结束，则进程进入下一级队列队尾。如果此时已经是在最下级的队列，则重新放回该队列队尾
3. 只有第k级队列为空时，才会为k+1级队头的进程分配时间片

用于作业/进程调度

用于进程调度

是否可抢占？

抢占式的算法。在k级队列的进程运行过程中，若更上级的队列（1~k-1级）中进入了一个新进程，则由于新进程处于优先级更高的队列中，因此新进程会抢占处理机，原来运行的进程放回k级队列队尾。

优缺点

对各类型进程相对公平（FCFS的优点）；每个新到达的进程都可以很快就得到响应（RR的优点）；短进程只用较少的时间就可完成（SPF的优点）；不必实现估计进程的运行时间（避免用户作假）；可灵活地调整对各类进程的偏好程度，比如CPU密集型进程、I/O密集型进程（拓展：可以将因I/O而阻塞的进程重新放回原队列，这样I/O型进程就可以保持较高优先级）

是否会导致饥饿？

会

宿船长 B站专用



拜拜

宿船长 B站专用