计算机操作系统

Operating Systems

田卫东

March, 2014

第2章 进程管理

2.1 进程的基本概念

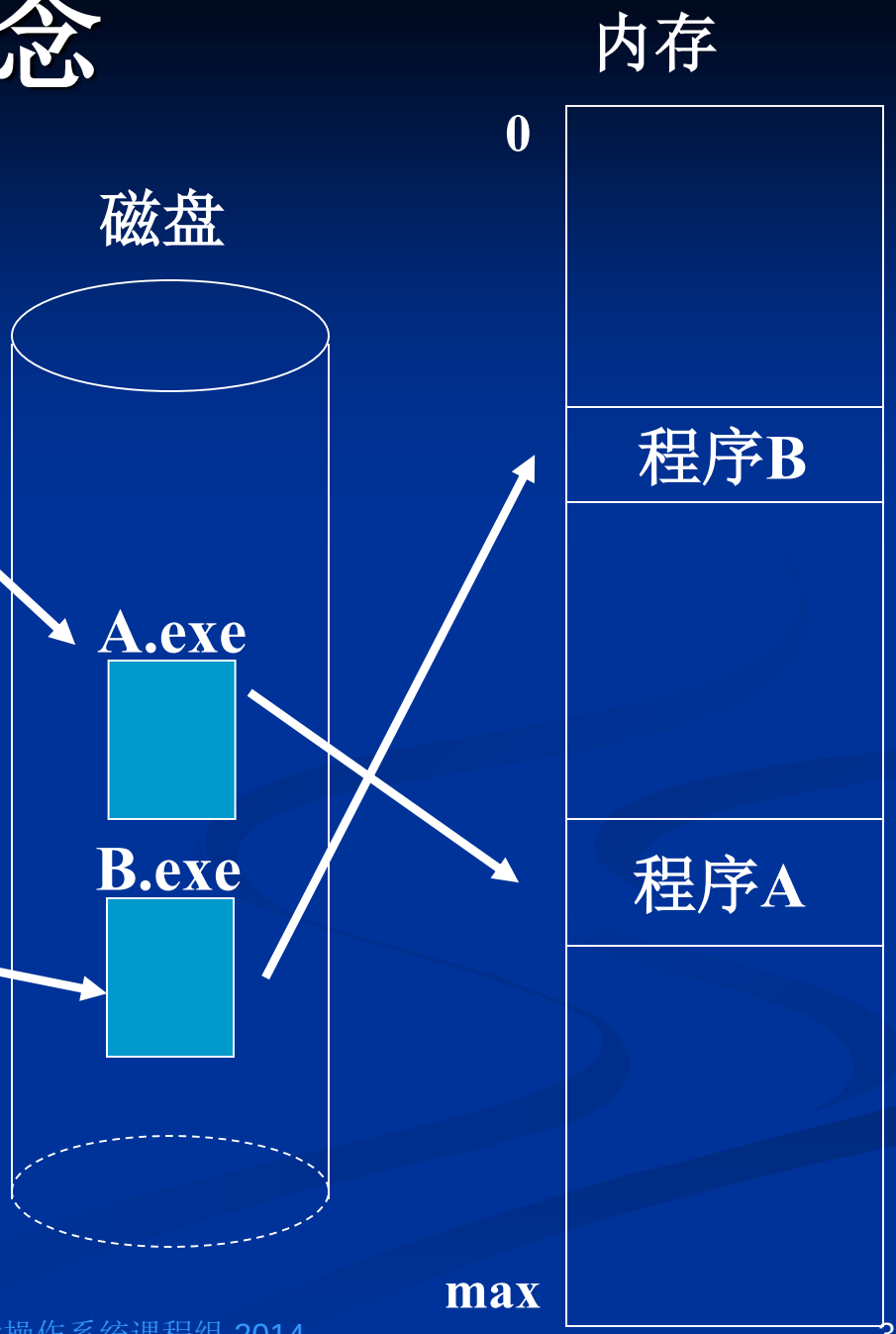
问题：OS如何组织程序运行？

A.c

```
#include <stdio.h>
int main()
{
    printf("hello world");
    return 0;
}
```

B.c

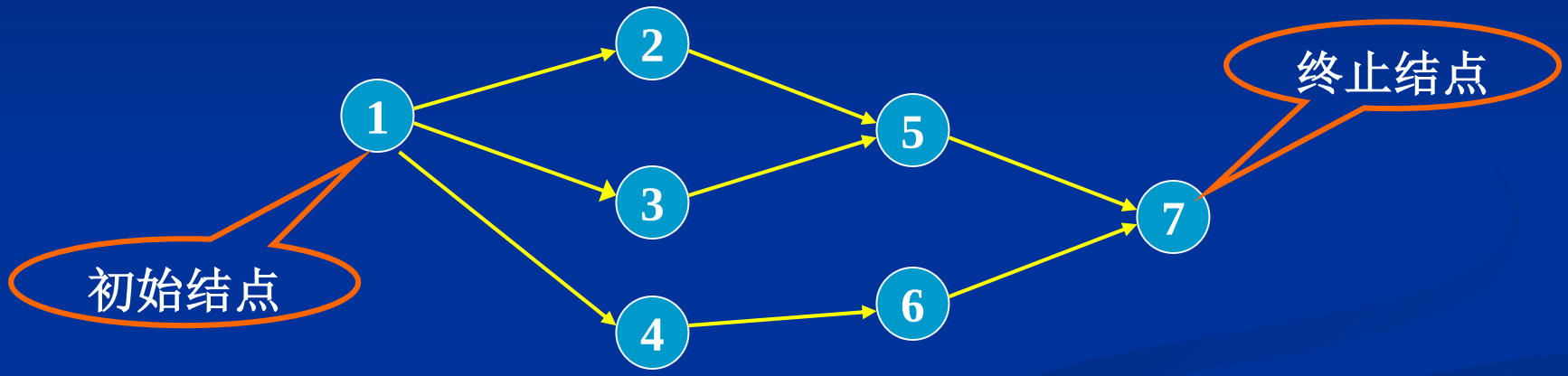
```
#include <stdio.h>
int main()
{
    scanf("%d", &r );
    double a = r*r*3.14;
    printf("area=%f", a );
    return 0;
}
```



2.1 进程的基本概念

2.1.1 前趋图

前趋图：用于描述实体(进程、程序段)之间执行次序的DAG。



结点　：代表一条语句，一段程序或一个进程

有向边：两结点之间存在的先后次序关系→

→ ={ (Pi,Pj) | Pi must complete before Pj may start }

实例(上图)：

P = { P1, P2, P3, P4, P5 ,P6,P7}

→ = { (P1,P2), (P1,P3), (P1,P4), (P2,P5), (P3,P5), (P4,P6), (P5,P7), (P6,P7) }

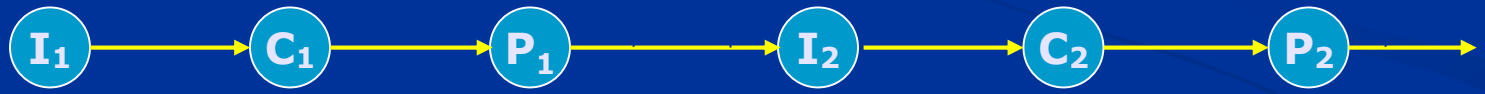
2.1 进程的基本概念
2.1.2 程序的顺序执行

(1) 顺序执行的基本概念

■**程序段级顺序执行**

- 输入程序段：I
- 计算程序段：C
- 打印程序段：P

■**程序级顺序执行**



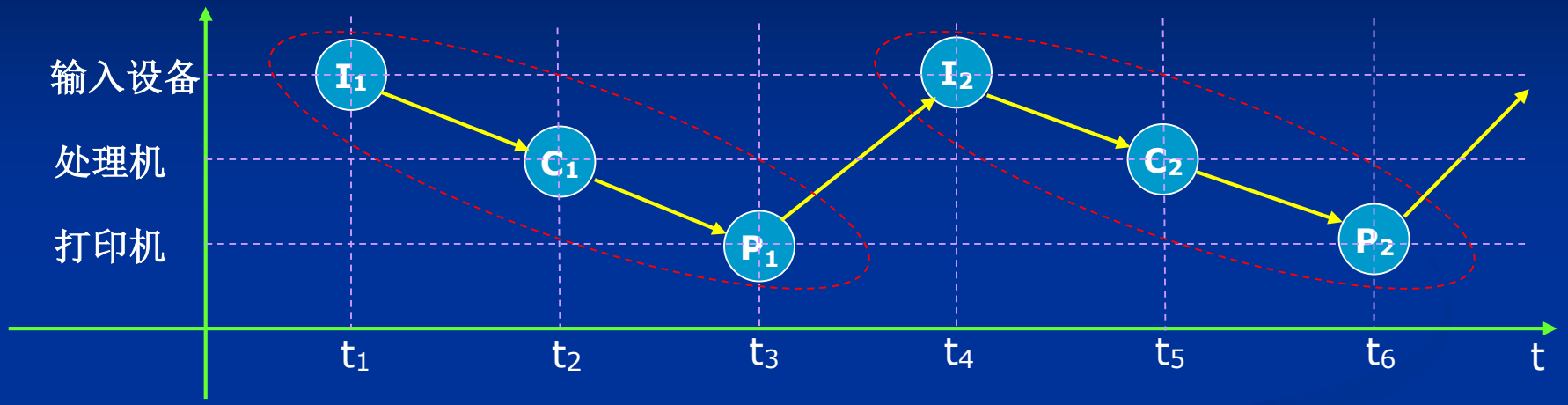
■**语句级顺序执行**

S1: a := x + y
S2: b := a – 5
S3: c := b + 1

(2) 顺序执行的特征

- **顺序性**
  处理机严格按照程序规定的顺序执行程序
- **封闭性**
  程序独占全机资源，不受其他程序的影响
- **可再现性**
  程序执行结果 = 初始条件 + 执行时的环境

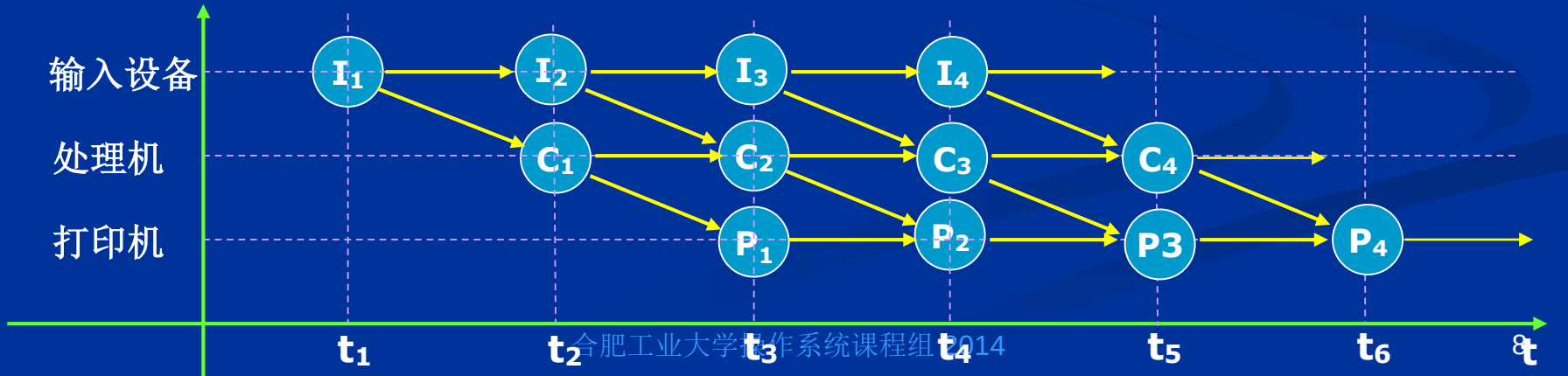
(3) 顺序执行存在的问题



- **存在的问题**
  - 整体资源利用率低
  - 单位时间内算题量少

2.1 进程的基本概念
2.1.3 程序的并发执行

(1) 并发执行的基本概念：多道程序设计技术



(2) 并发执行的问题

```
VAR  N : integer := 3 ;
Program A :
Begin
      while ( TRUE )
        N : = N+1 ;
      end while
end
Program B :
Begin
      while ( TRUE )
        print(N) ;
        N := 0 ;
      end while
end
```

程序A

程序B

语句执行顺序：　　程序B打印结果

```
N := N +1 ;
print(N);
N:=0;
```
→ 4

```
print(N);
N := 0;
N := N + 1
```
→ 3

```
print(N);
N:=0;
print(N);
```
→ 3 0

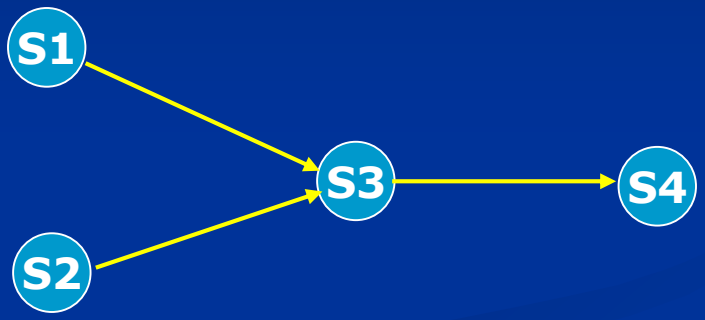
```
N := N +1
N := N +1
N := N +1
…
print(N);
N:=0
```
→ ?

结论：程序B因为受到程序A的影响，运行结果出现了不可再现性，即结果不定。

(3) 什么条件下，允许程序的并发执行？

- 请研究下面典型程序段的并发执行：

```
S1:    a := x + 2
S2:    b := y + 1
S3:    c := a + b
S4:    d := c + 6
```



(4) 并发执行的特征

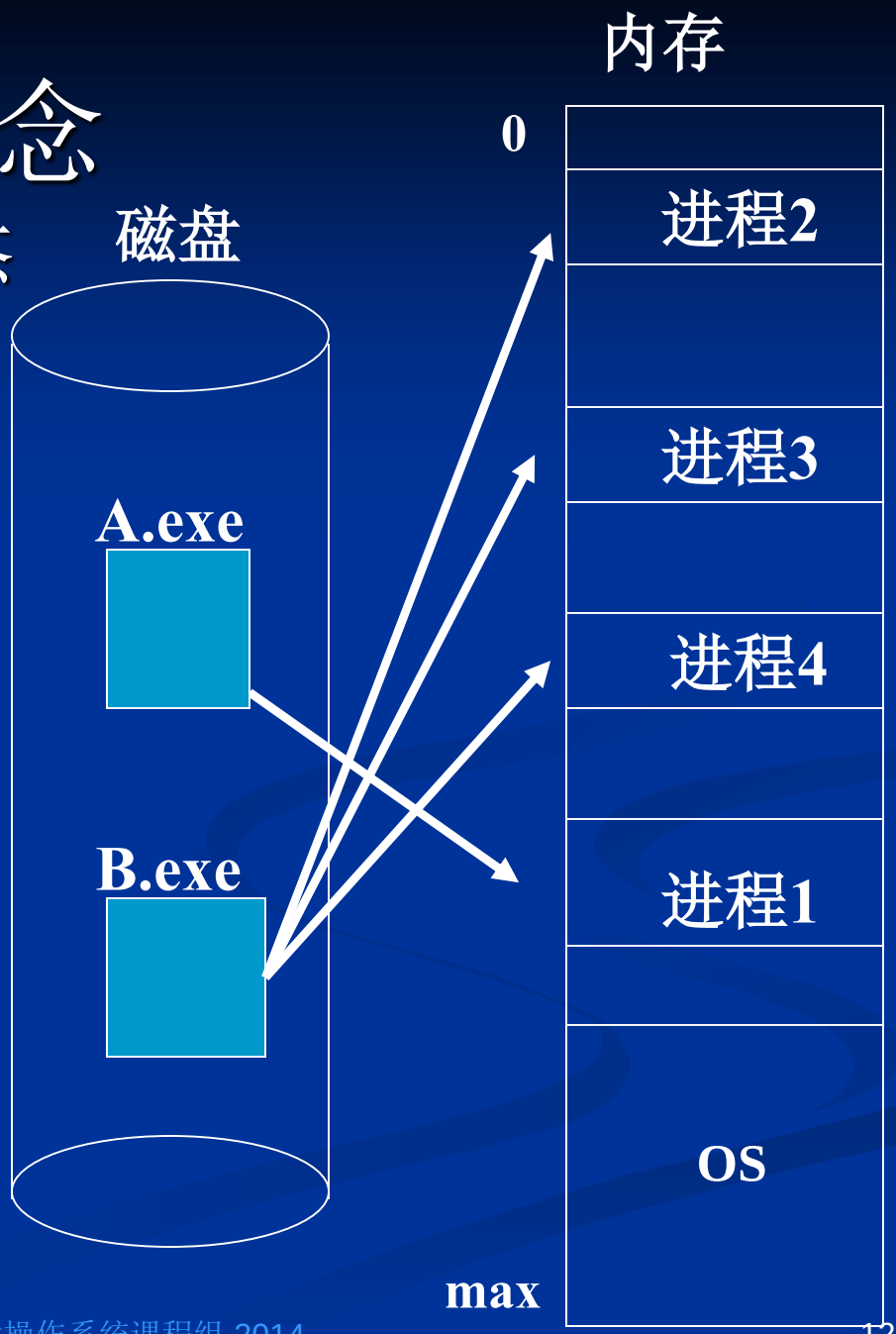
- **间断性**
  由于资源的共享，程序执行具有"执行-暂停-执行"的特点
- **失去封闭性**
  程序不再独占全机资源，运行受到其他程序的影响
- **不可再现性**
  程序执行结果受其他程序的影响，结果不定

2.1 进程的基本概念

2.1.4 进程的特征与状态

(1) 进程概念的产生

- 在多道程序环境下，程序的并发执行代替了程序的顺序执行，程序的活动不再处于封闭系统中，从而出现了许多新特征。为此人们引入了新概念—进程



(2) 进程的特征

- 历史上出现过的对进程的一些描述
  - 程序在处理机上的一次执行过程；
  - 可以和别的计算并行执行的计算；
  - 进程是程序在一个数据集合上运行的过程，是系统进行资源分配和调度的一个独立单位；
  - 是一个具有一定功能的程序，是关于某个数据集合的一次运行活动。

(2) 进程的特征

- 动态性

  是程序的一次执行过程，因而是动态的。
- 并发性

  引入进程就是为了和其他程序并发执行，提高资源利用率。
- 独立性

  能独立运行的基本单位，也是资源分配和调度的基本单位。
- 异步性

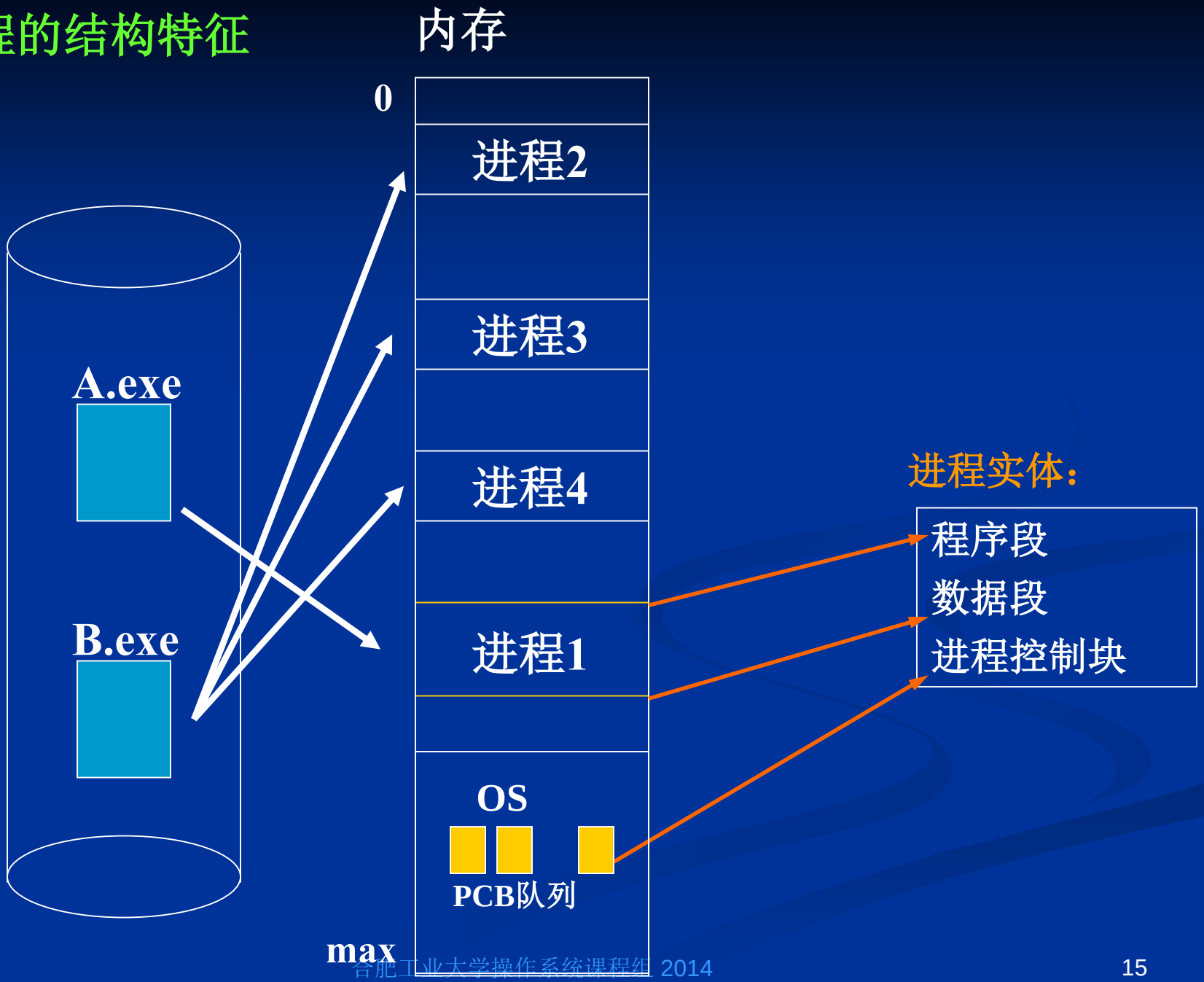
  进程以各自独立的、不可预知的速度向前推进。
- 结构特征

  程序段

  数据段

  进程控制块（PCB）

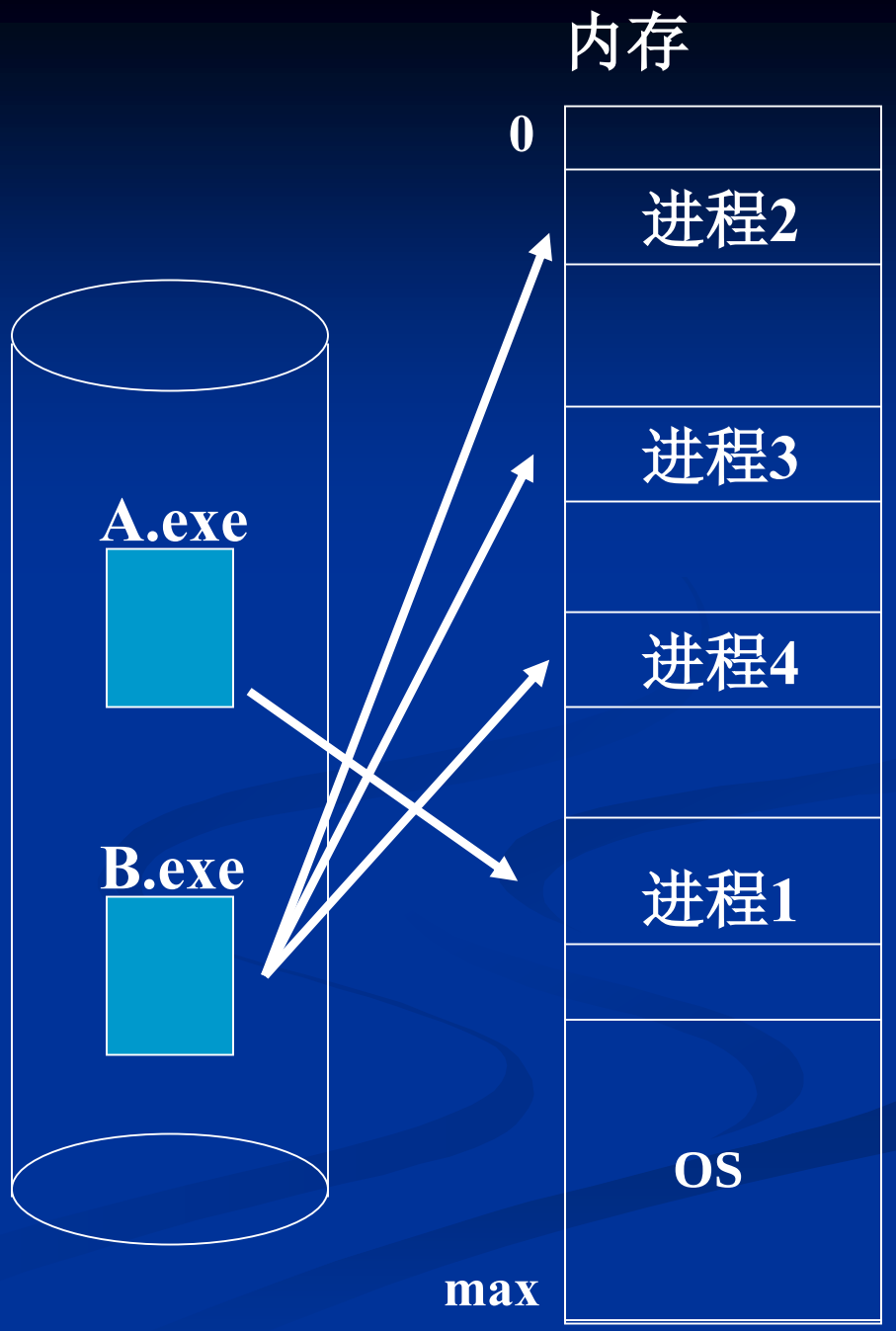
进程的结构特征



(3) 进程的定义

进程(Process)的定义：

进程实体的运行过程，是系统进行资源分配和调度的独立单位。



(4) 进程的三种基本状态

- 进程基本状态

  1) 就绪(Ready)状态

  进程已经获得除处理机之外的所有资源，一旦获得处理机就可以立即执行；

  2) 执行(Running)状态

  一个进程已获得了必要的资源，正在处理机上运行；

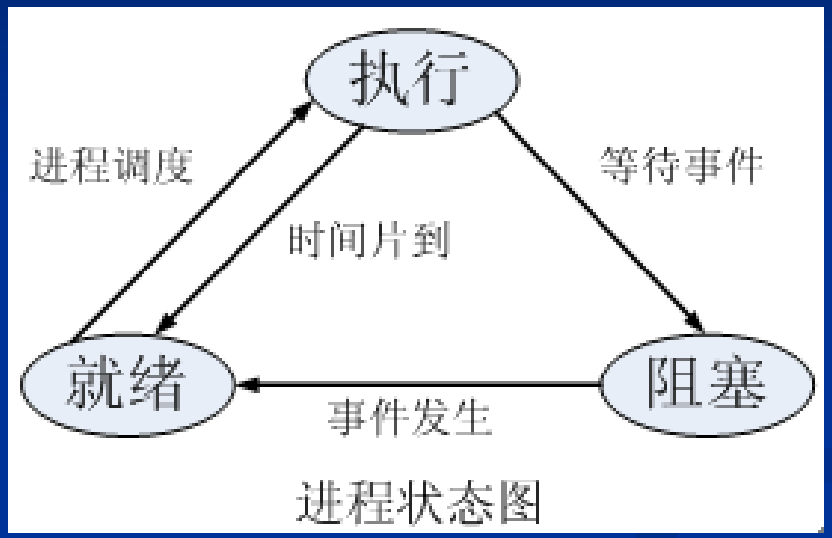
  3) 阻塞(Block)状态

  正在执行的进程，由于发生某事件而暂停无法继续执行，（如等待I/O、数据到达）

- 进程并非固定处于某种状态，其状态会随着自身的运行和和外界条件的变化而变化。

(4) 进程的三种基本状态

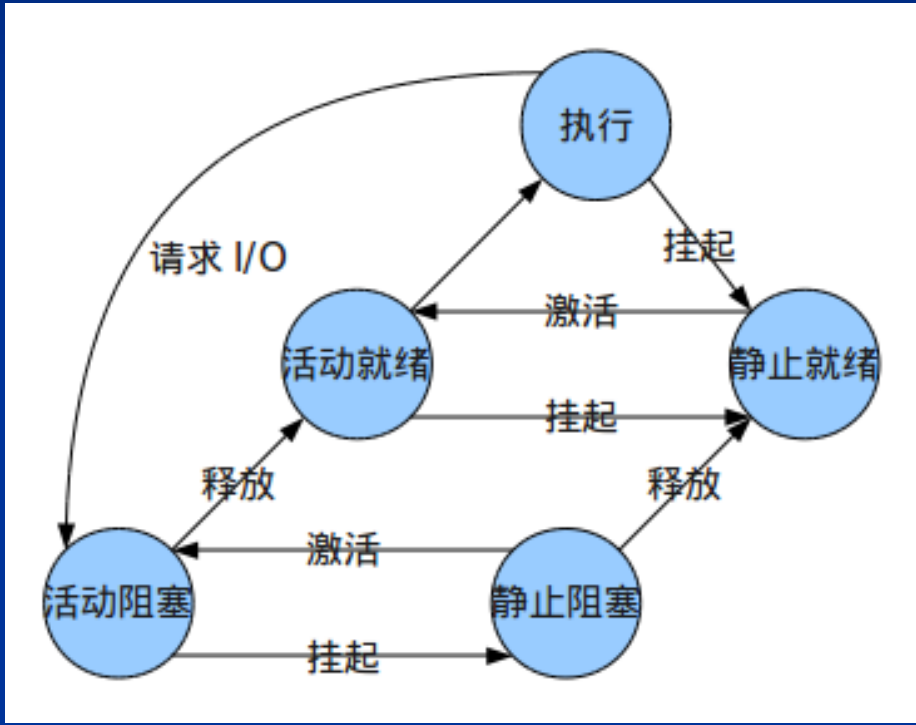
- 进程基本状态之间的转化



进程状态图

(4) 进程的三种基本状态

- 引入挂起状态的进程状态转化

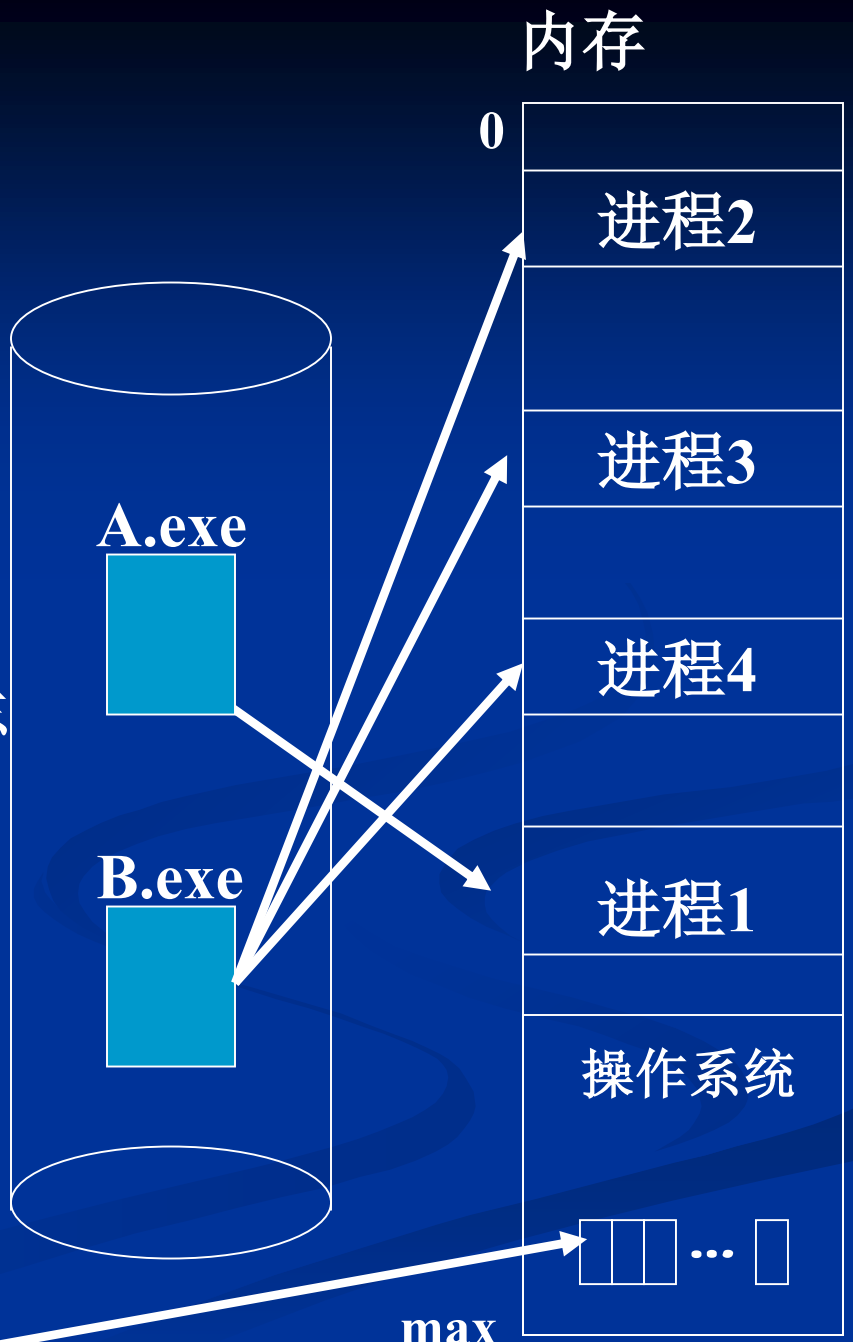


2.1 进程的基本概念

2.1.5 进程控制块

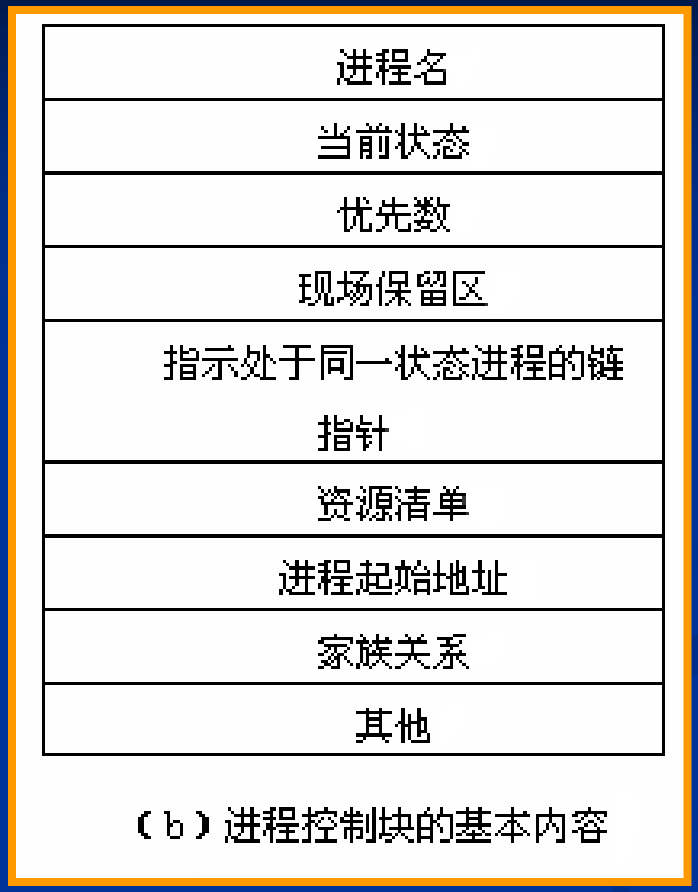
(1) 进程控制块基本概念

- Process Control Block, PCB
- OS管理进程的数据结构
- 进程控制块是进程存在的唯一标志
- 进程控制块位置：操作系统内核



(2) 进程控制块的内容

- 进程标志符
- 处理机状态
- 进程调度信息
- 进程控制信息

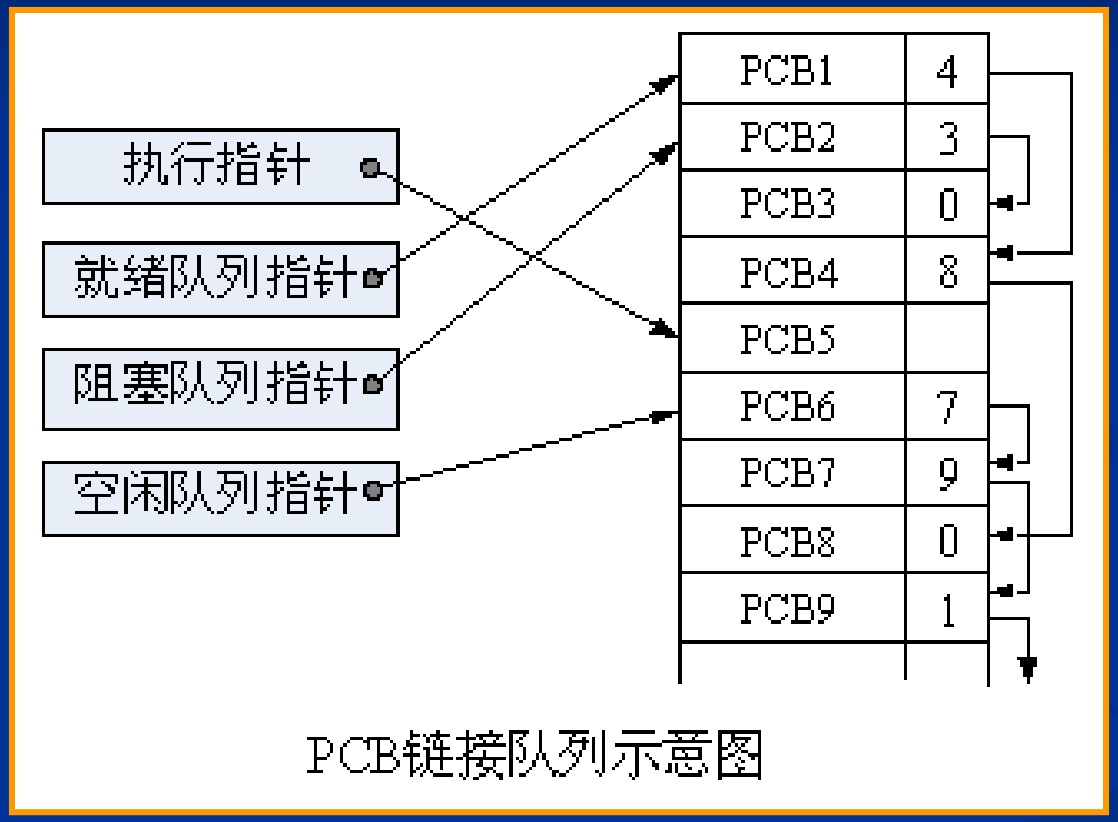


（b）进程控制块的基本内容

```
/*
*********************************************************************************************************
*                                              TASK CONTROL BLOCK
*********************************************************************************************************
*/

typedef struct os_tcb {
    OS_STK        *OSTCBStkPtr;        /* Pointer to current top of stack                          */

#if OS_TASK_CREATE_EXT_EN
    void          *OSTCBExtPtr;        /* Pointer to user definable data for TCB extension         */
    OS_STK        *OSTCBStkBottom;     /* Pointer to bottom of stack                               */
    INT32U         OSTCBStkSize;       /* Size of task stack (in bytes)                            */
    INT16U         OSTCBOpt;           /* Task options as passed by OSTaskCreateExt()              */
    INT16U         OSTCBId;            /* Task ID (0..65535)                                       */
#endif

    struct os_tcb *OSTCBNext;          /* Pointer to next     TCB in the TCB list                  */
    struct os_tcb *OSTCBPrev;          /* Pointer to previous TCB in the TCB list                  */

#if (OS_Q_EN && (OS_MAX_QS >= 2)) || OS_MBOX_EN || OS_SEM_EN
    OS_EVENT      *OSTCBEventPtr;      /* Pointer to event control block                           */
#endif

#if (OS_Q_EN && (OS_MAX_QS >= 2)) || OS_MBOX_EN
    void          *OSTCBMsg;           /* Message received from OSMboxPost() or OSQPost()          */
#endif

    INT16U         OSTCBDly;           /* Nbr ticks to delay task or, timeout waiting for event    */
    INT8U          OSTCBStat;          /* Task status                                              */
    INT8U          OSTCBPrio;          /* Task priority (0 == highest, 63 == lowest)               */

    INT8U          OSTCBX;             /* Bit position in group  corresponding to task priority (0..7) */
    INT8U          OSTCBY;             /* Index into ready table corresponding to task priority    */
    INT8U          OSTCBBitX;          /* Bit mask to access bit position in ready table           */
    INT8U          OSTCBBitY;          /* Bit mask to access bit position in ready group           */

#if OS_TASK_DEL_EN
    BOOLEAN        OSTCBDelReq;        /* Indicates whether a task needs to delete itself          */
#endif
} OS_TCB;

/*$PAGE*/
/*
*********************************************************************************************************
*                                              GLOBAL VARIABLES
*********************************************************************************************************
```

(3) 进程控制块的组织形式

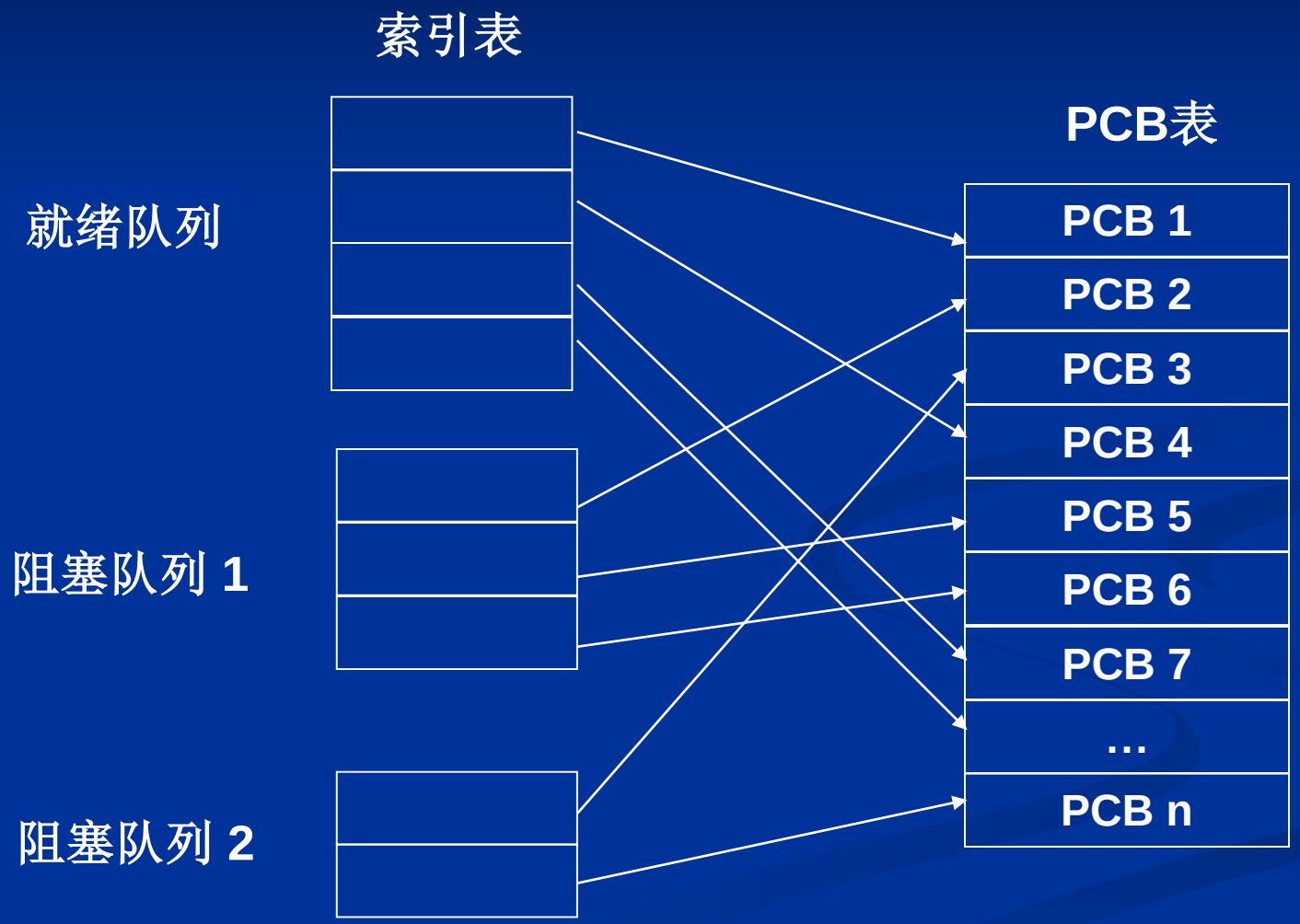
- 链接方式



PCB链接队列示意图

(3) 进程控制块的组织形式

- 索引方式



2.2 进程控制

2.2.1 进程的创建和撤消

(1) 进程的创建

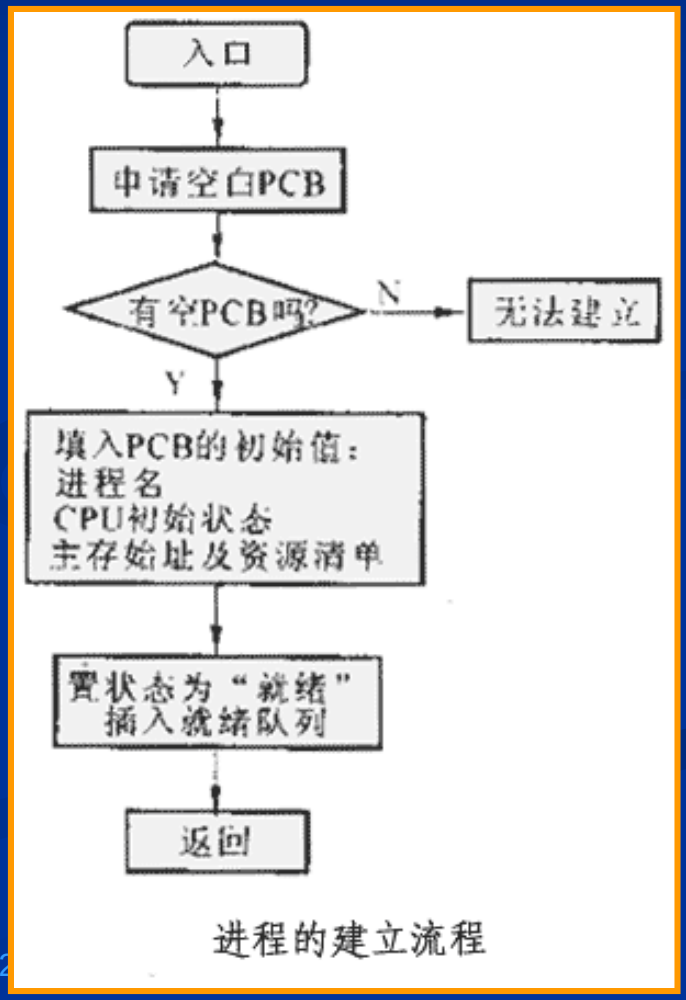
- 引起进程创建的事件
  - 用户登录
  - 作业调度
  - 提供服务

    例如：提供打印服务
  - 应用请求

    例如：程序自行创建
- 进程创建原语

  create()

- 创建进程的流程：



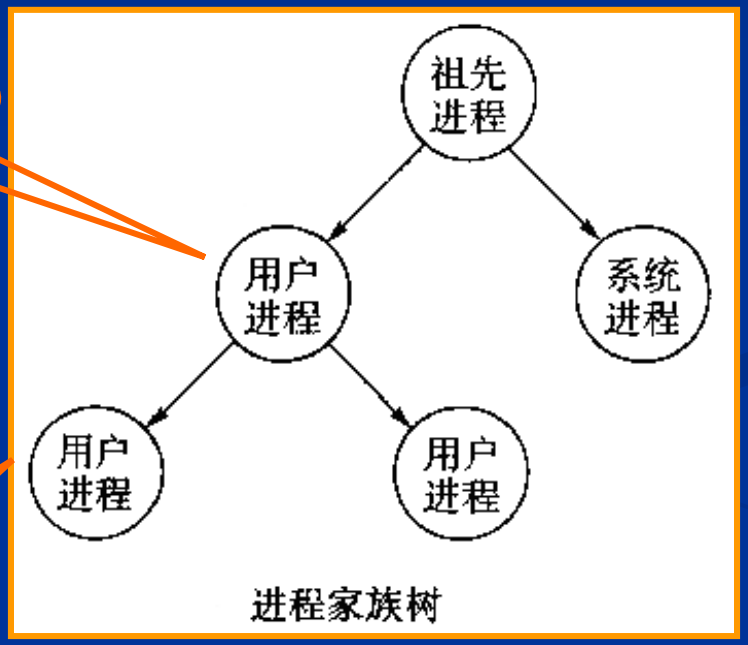
进程的建立流程

(1) 进程的创建

- 进程图(树）：

  体现进程的创建关系。

父进程



进程家族树

子进程

(2) 进程的撤消(终止)

- 引起进程撤消的事件
  - 正常结束
  - 异常结束
  - 外界干预

- 进程撤消原语

  exit()

- 撤消进程的流程:

① 根据被终止进程的标识符，从PCB集合中检索出该进程的PCB，从中读出该进程的状态。

② 若被终止进程正处于执行状态，应立即终止该进程的执行，并置调度标志为真，用于指示该进程被终止后应重新进行调度。

③ 若该进程还有子孙进程，还应将其所有子孙进程予以终止，以防他们成为不可控的进程。

④ 将被终止进程所拥有的全部资源，或者归还给其父进程，或者归还给系统。

⑤ 将被终止进程（它的PCB）从所在队列（或链表）中移出，等待其他程序来搜集信息。

2.2 进程控制

2.2.2 进程的阻塞与唤醒

(1) 进程的阻塞

- 引起进程阻塞的事件
  - 请求系统服务
  - 请求操作
  - 等待数据、资源
  - 等待工作任务

- 进程阻塞原语

  block()

◆进程自己阻塞自己

- 阻塞原理

(2) 进程的唤醒

- 引起进程唤醒的事件
  - 允许系统服务
  - 允许操作
  - 数据、资源到达
  - 工作任务到达
- 进程唤醒原语

  wakeup()

- 唤醒原理

◆被别的进程唤醒

2.2 进程控制
2.2.2 进程的挂起与激活

(1) 进程的挂起

- 引起进程挂起的事件
  - 调试系统
  - 暂停程序执行
- 进程挂起原语

  suspend()

(2) 进程的激活

- 引起进程激活的事件
  - 调试系统结束
  - 程序继续执行
- 进程激活原语

  activate()