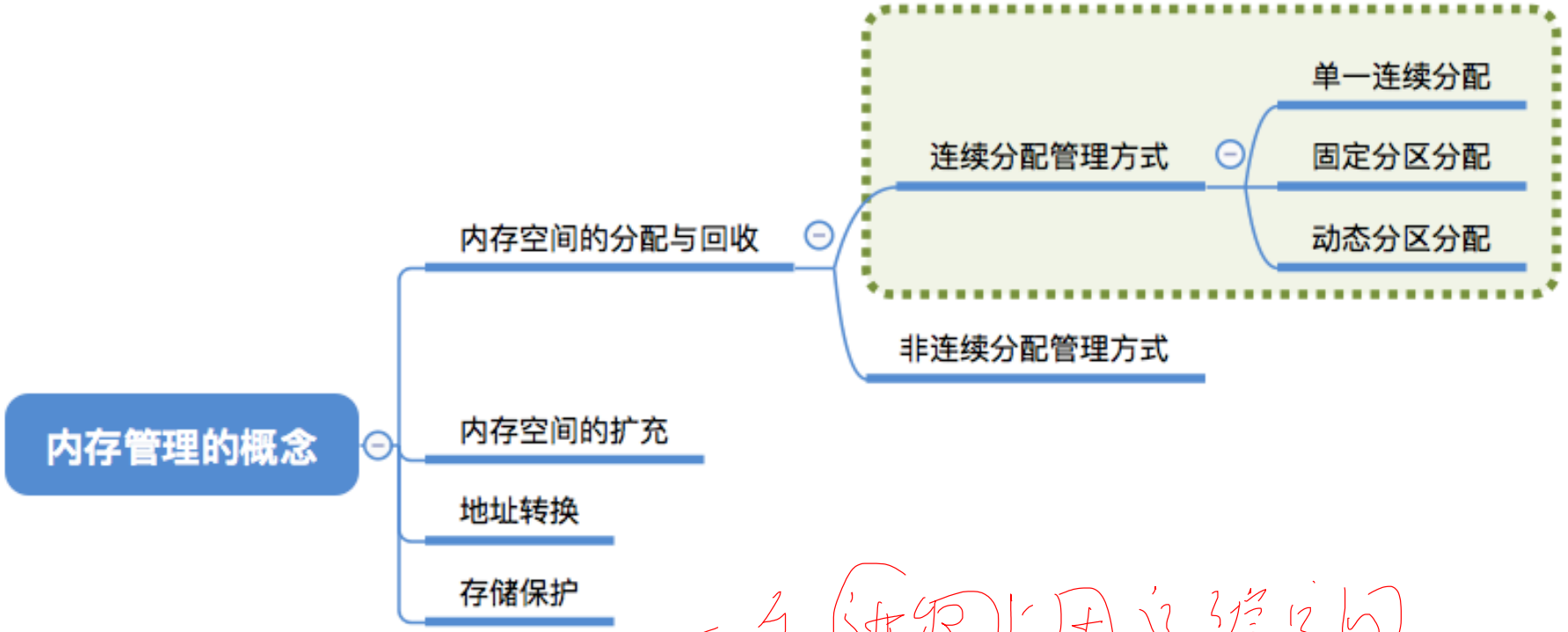
本节内容

连续分配管理方式

知识总览



一个进程占用连续空间

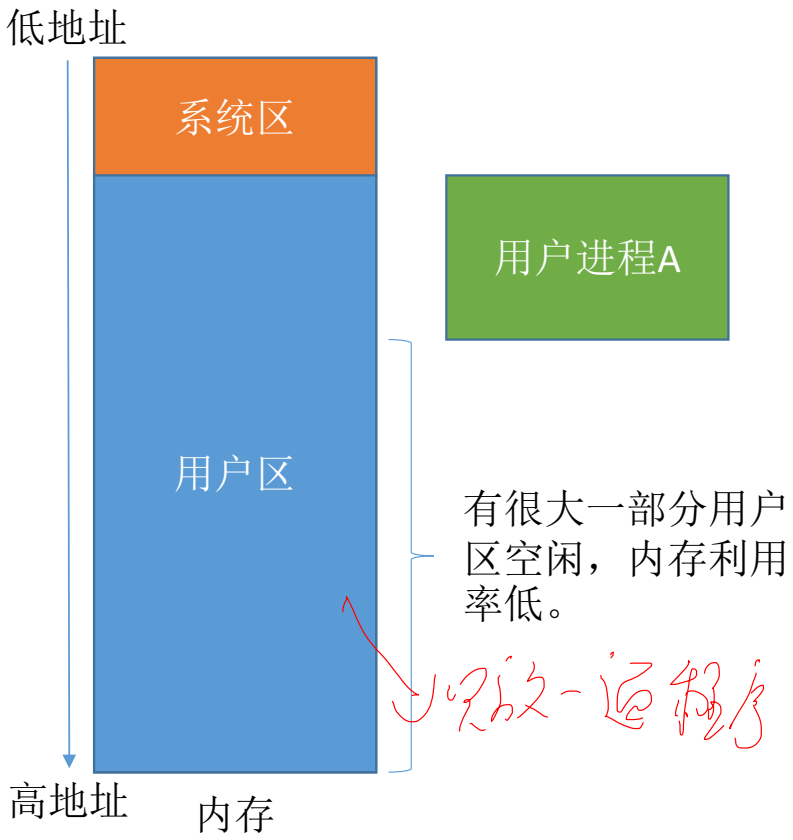
连续分配：指为用户进程分配的必须是一个连续的内存空间。

单一连续分配

在单一连续分配方式中，内存被分为系统区和用户区。
系统区通常位于内存的低地址部分，用于存放操作系统相关数据；用户区用于存放用户进程相关数据。
内存中只能有一道用户程序，用户程序独占整个用户区空间。

优点：实现简单；无外部碎片；可以采用覆盖技术扩充内存；不一定需要采取内存保护（eg：早期的 PC 操作系统 MS-DOS）。
缺点：只能用于单用户、单任务的操作系统中；有内部碎片；存储器利用率极低。

分配给某进程的内存区域中，如果有些部分没有用上，就是“内部碎片”

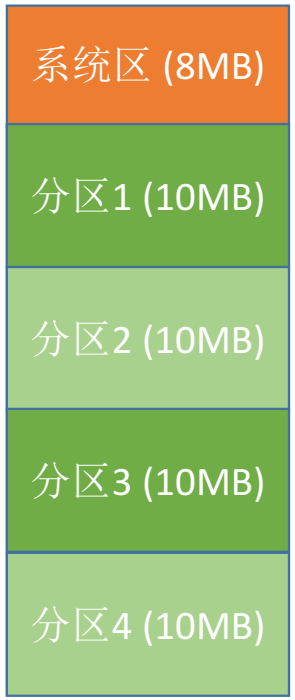


固定分区分配

20世纪60年代出现了支持多道程序的系统，为了能在内存中装入多道程序，且这些程序之间又不会相互干扰，于是将整个用户空间划分为若干个固定大小的分区，在每个分区中只装入一道作业，这样就形成了最早的、最简单的一种可运行多道程序的内存管理方式。

分区大小相等：缺乏灵活性，但是很适合用于用一台计算机控制多个相同对象的场合（比如：钢铁厂有n个相同的炼钢炉，就可把内存分为n个大小相等的区域存放n个炼钢炉控制程序）

分区大小不等：增加了灵活性，可以满足不同大小的进程需求。根据常在系统中运行的作业大小情况进行划分（比如：划分多个小分区、适量中等分区、少量大分区）



内存（分区大小相等）

内存（分区大小不等）

固定分区分配

操作系统需要建立一个数据结构——分区说明表，来实现各个分区的分配与回收。每个表项对应一个分区，通常按分区大小排列。每个表项包括对应分区的大小、起始地址、状态（是否已分配）。

分区号	大小（MB）	起始地址（M）	状态
1	2	8	未分配
2	2	10	未分配
3	4	12	已分配
……	……	……	……

用数据结构的数组（或链表）即可表示这个表

内存（分区大小不等）

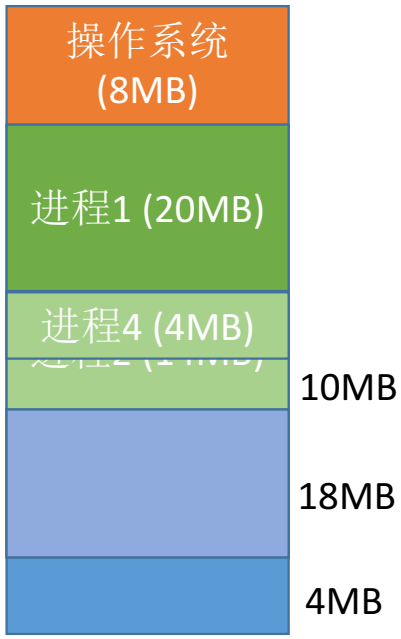
当某用户程序要装入内存时，由操作系统内核程序根据用户程序大小检索该表，从中找到一个能满足大小的、未分配的分区，将之分配给该程序，然后修改状态为“已分配”。

优点：实现简单，无外部碎片。

缺点：a. 当用户程序太大时，可能所有的分区都不能满足需求，此时不得不采用覆盖技术来解决，但这又会降低性能；b. 会产生内部碎片，内存利用率低。

动态分区分配

动态分区分配又称为可变分区分配。这种分配方式不会预先划分内存分区，而是在进程装入内存时，根据进程的大小动态地建立分区，并使分区的大小正好适合进程的需要。因此系统分区的大小和数目是可变的。（eg：假设某计算机内存大小为 64MB，系统区 8MB，用户区共 56 MB…）



内存

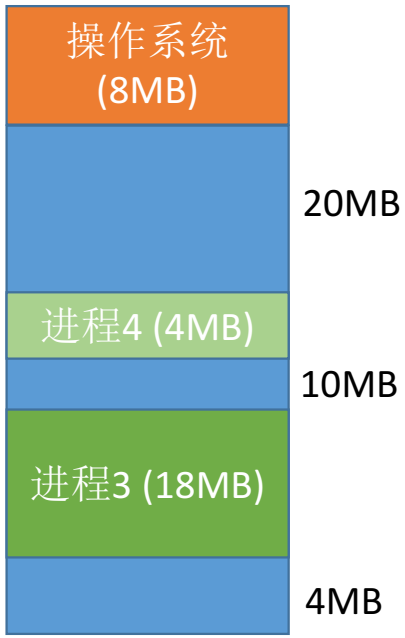
1. 系统要用什么样的数据结构记录内存的使用情况？
2. 当很多个空闲分区都能满足需求时，应该选择哪个分区进行分配？
3. 如何进行分区的分配与回收操作？

动态分区分配

1. 系统要用什么样的数据结构记录内存的使用情况？

两种常用的数据结构

- 空闲分区表
- 空闲分区链



分区号	分区大小（MB）	起始地址（M）	状态
1	20	8	空闲
2	10	32	空闲
3	4	60	空闲

空闲分区表：每个空闲分区对应一个表项。表项中包含分区号、分区大小、分区起始地址等信息

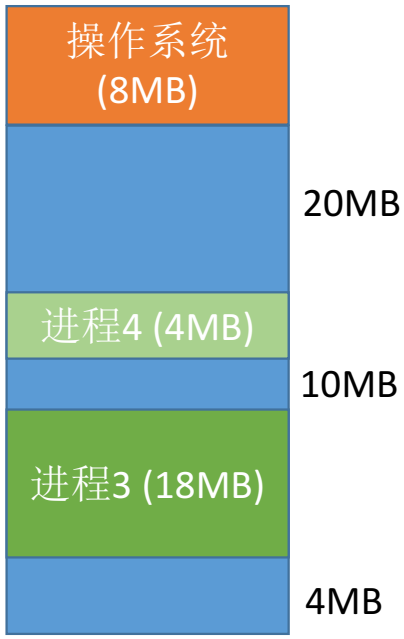
空闲分区链：每个分区的起始部分和末尾部分分别设置前向指针和后向指针。起始部分处还可记录分区大小等信息

动态分区分配

2. 当很多个空闲分区都能满足需求时，应该选择哪个分区进行分配？

进程5 (4MB)

应该用最大的分区进行分配？还是用最小的分区进行分配？又或是用地址最低的部分进行分配？



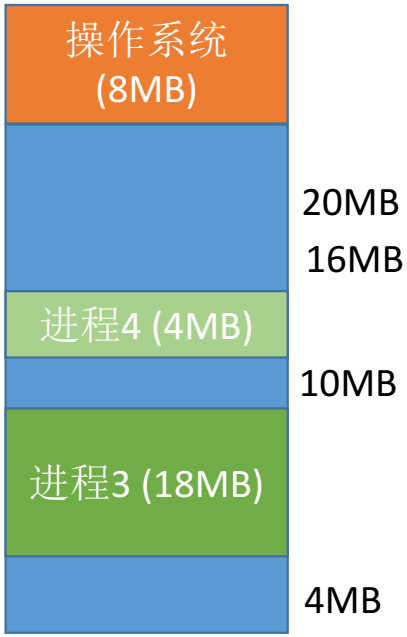
内存

把一个新作业装入内存时，须按照一定的动态分区分配算法，从空闲分区表（或空闲分区链）中选出一个分区分配给该作业。由于分配算法算法对系统性能有很大的影响，因此人们对它进行了广泛的研究。

下个小节会介绍四种动态分区分配算法...

动态分区分配

3. 如何进行分区的分配与回收操作？
假设系统采用的数据结构是“空闲分区表”...
如何分配？



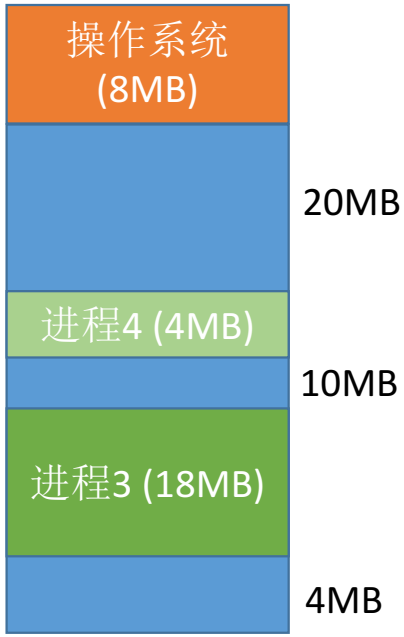
分区号	分区大小（MB）	起始地址（M）	状态
1	20	8	空闲
2	10	32	空闲
3	4	60	空闲

进程5 (4

分区号	分区大小（MB）	起始地址（M）	状态
1	16	12	空闲
2	10	32	空闲
3	4	60	空闲

动态分区分配

3. 如何进行分区的分配与回收操作？
假设系统采用的数据结构是“空闲分区表”...
如何分配？

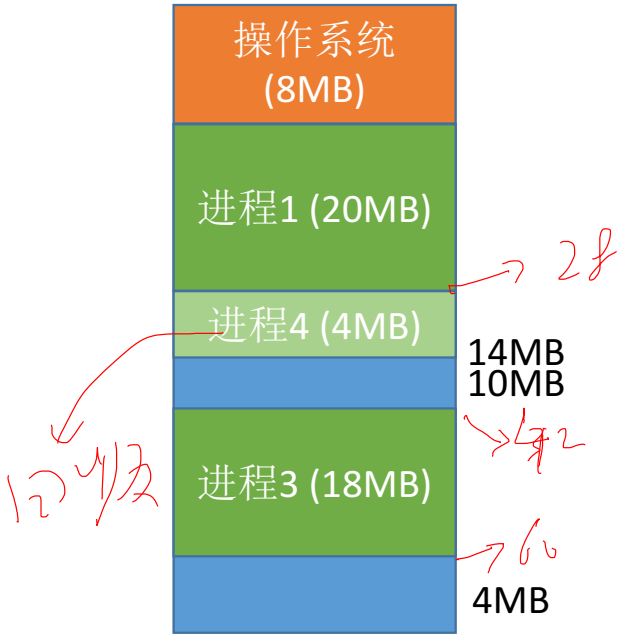


分区号	分区大小（MB）	起始地址（M）	状态
1	20	8	空闲
2	10	32	空闲
3	4	60	空闲

分区号	分区大小（MB）	起始地址（M）	状态
1	20	8	空闲
2	10	32	空闲

动态分区分配

3. 如何进行分区的分配与回收操作？
假设系统采用的数据结构是“空闲分区表”...
如何回收？



内存

情况一：回收区的后面有一个相邻的空闲分区

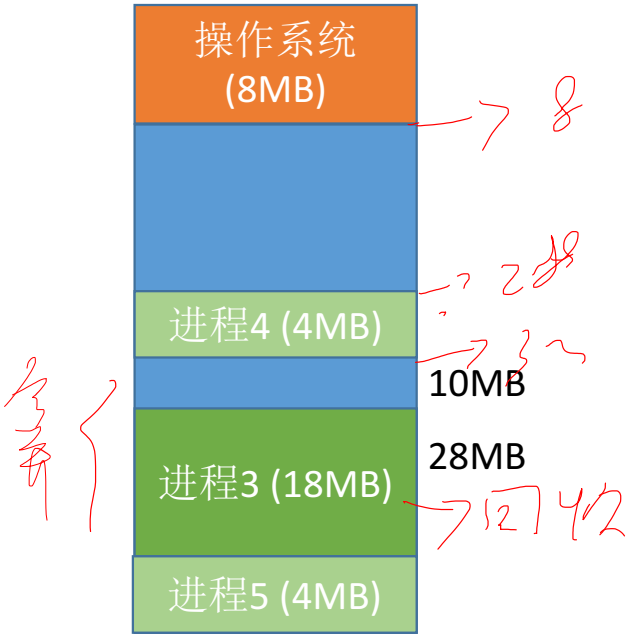
分区号	分区大小（MB）	起始地址（M）	状态
1	10	32	空闲
2	4	60	空闲

分区号	分区大小（MB）	起始地址（M）	状态
1	14	28	空闲
2	4	60	空闲

两个相邻的空闲分区合并为一个

动态分区分配

3. 如何进行分区的分配与回收操作？
假设系统采用的数据结构是“空闲分区表”...
如何回收？



内存

情况二：回收区的前面有一个相邻的空闲分区

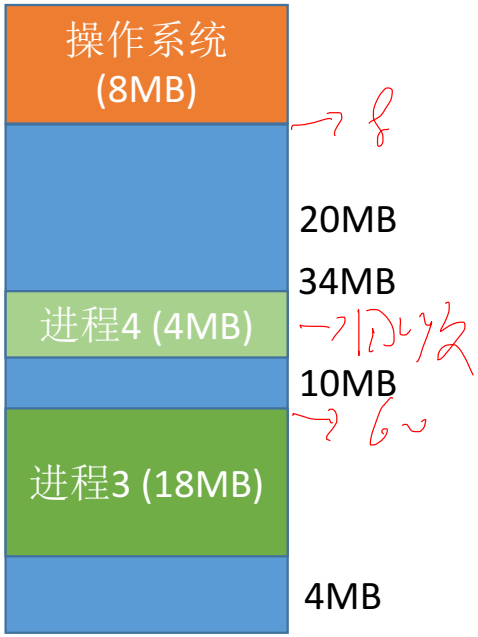
分区号	分区大小（MB）	起始地址（M）	状态
1	20	8	空闲
2	10	32	空闲

分区号	分区大小（MB）	起始地址（M）	状态
1	20	8	空闲
2	28	32	空闲

两个相邻的空闲分区合并为一个

动态分区分配

3. 如何进行分区的分配与回收操作？
假设系统采用的数据结构是“空闲分区表”...
如何回收？



内存

情况三：回收区的前、后各有一个相邻的空闲分区

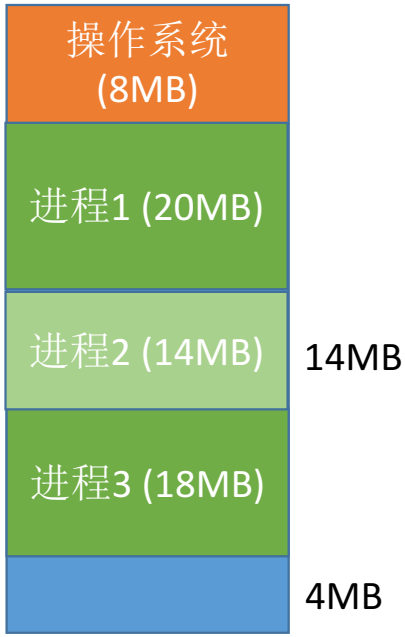
分区号	分区大小（MB）	起始地址（M）	状态
1	20	8	空闲
2	10	32	空闲
3	4	60	空闲

分区号	分区大小（MB）	起始地址（M）	状态
1	34	8	空闲
2	4	60	空闲

三个相邻的空闲分区合并为一个

动态分区分配

3. 如何进行分区的分配与回收操作？
假设系统采用的数据结构是“空闲分区表”...
如何回收？



情况四：回收区的前、后都没有相邻的空闲分区

分区号	分区大小（MB）	起始地址（M）	状态
1	4	60	空闲

分区号	分区大小（MB）	起始地址（M）	状态
1	14	28	空闲
2	4	60	空闲

新增一个表项

注：各表项的顺序不一定按照地址递增顺序排列，具体的排列方式需要依据动态分区分配算法来确定。

动态分区分配

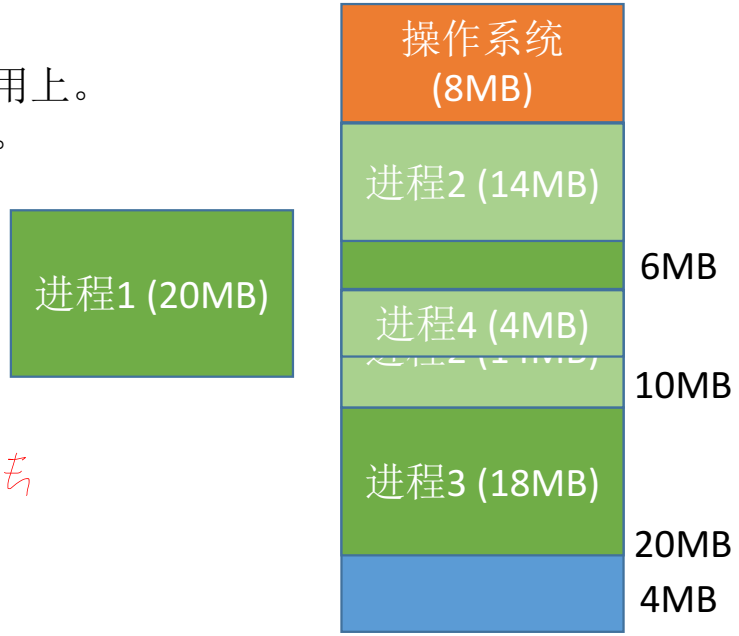
动态分区分配又称为可变分区分配。这种分配方式不会预先划分内存分区，而是在进程装入内存时，根据进程的大小动态地建立分区，并使分区的大小正好适合进程的需要。因此系统分区的大小和数目是可变的。

动态分区分配没有内部碎片，但是有外部碎片。
内部碎片，分配给某进程的内存区域中，如果有些部分没有用上。
外部碎片，是指内存中的某些空闲分区由于太小而难以利用。

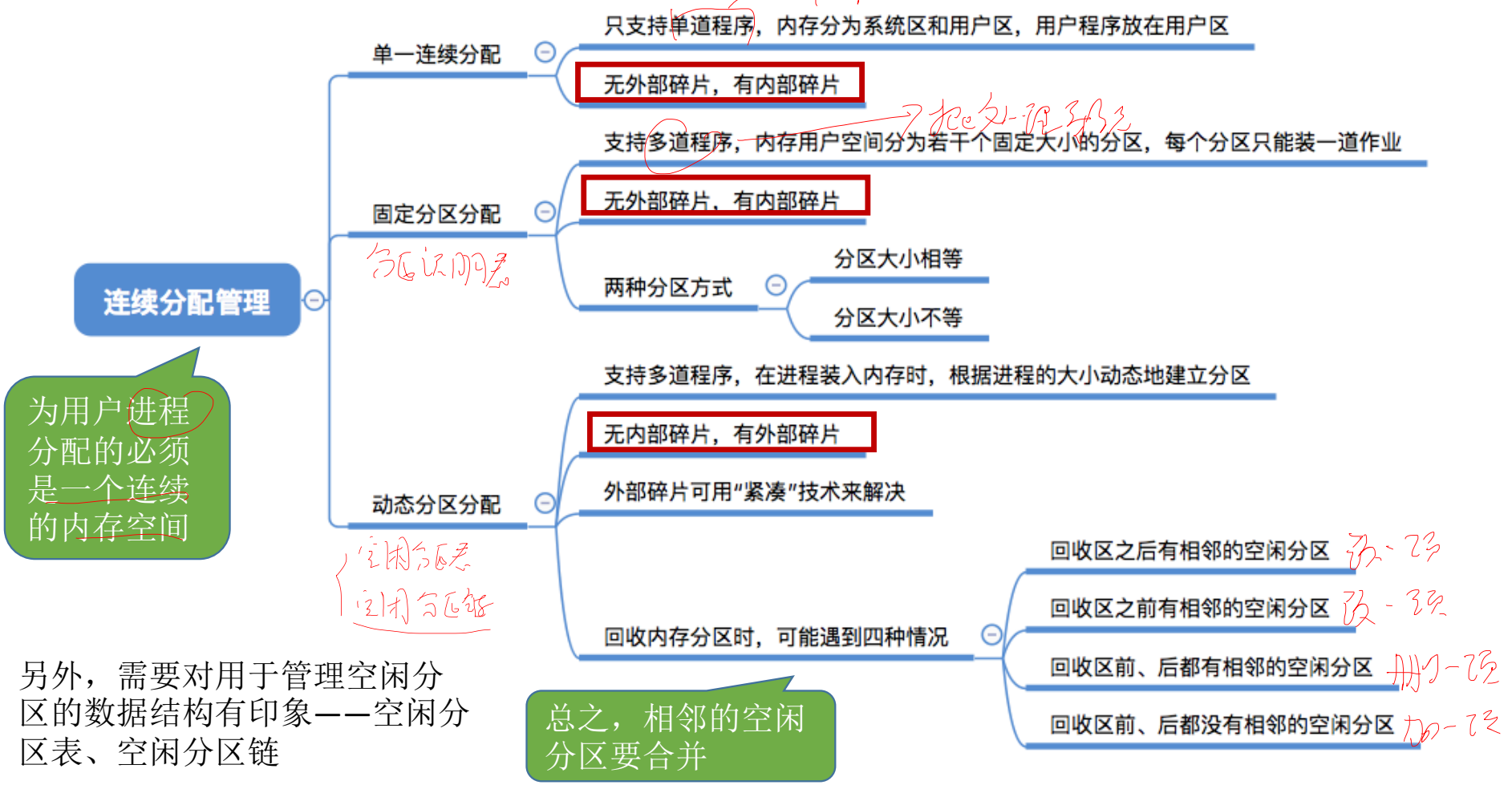
如果内存中空闲空间的总和本来可以满足某进程的要求，但由于进程需要的是一整块连续的内存空间，因此这些“碎片”不能满足进程的需求。
可以通过紧凑（拼凑，Compaction）技术来解决外部碎片。

1. 回忆交换技术，什么是换入/换出？什么是中级调度（内存调度）？

2. 思考动态分区分配应使用哪种装入方式？“紧凑”之后需要做什么处理？



知识回顾与重要考点



另外，需要对用于管理空闲分区的数据结构有印象——空闲分区表、空闲分区链