1、

分配器allocators是幕后英雄，可以了解但不建议去使用它。

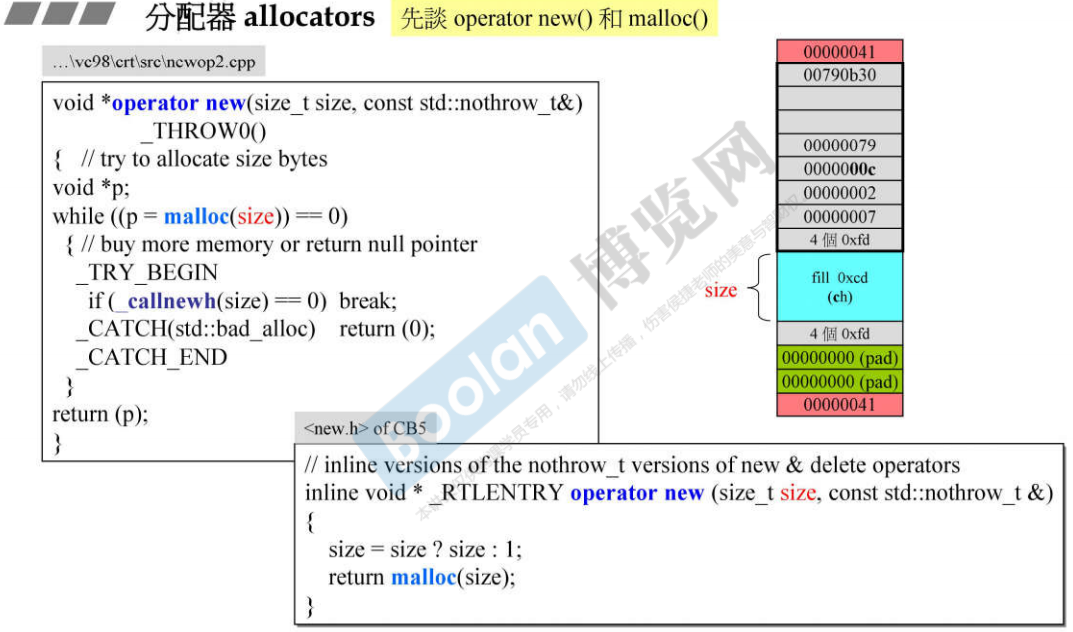
分配器内存的分配效率是影响容器效率的主要因素，例如时间效率和空间效率

2、

无论是什么分配器，最终要开辟内存仍然要回归到malloc()函数，malloc是与调用操作系统开辟内存API直接相连的函数

3、

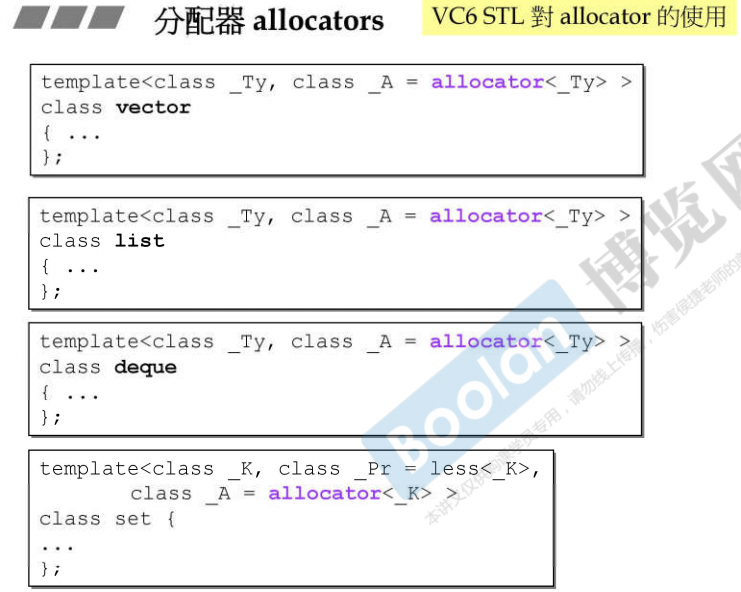
在C++中分配内存，由new()回归到operator new()再回归到malloc()（如第二张代码）



在内存图中可以看出，malloc除了我们的所需要的size以外还有其他内存也一起分配来了，这些附加的内存基本是固定的。因此主要看我们需要的size。

一头一尾红色部分叫cookie，它记录了malloc实际分配内存块整块的大小。想想在我们平时使用malloc()函数时之后用free返回的只是指针，那么大小呢？大小已经记录在cookie里面了，free能找到cookie。

4、



这里容器vector\list\deque\set所使用的分配器是一个模板类allocator<>

5、

分配器allocator最主要的任务是allocate和deallocate

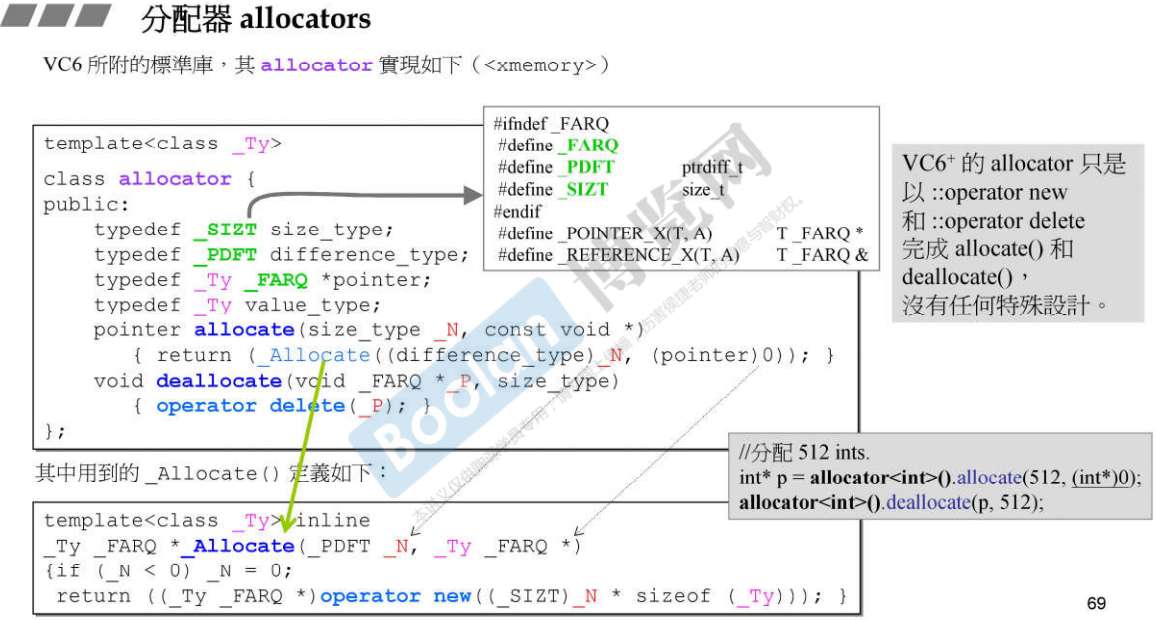
可以看到allocate调用\_Allocate函数，而\_Allocate调用operator new()，而operator new()调用malloc()函数。结论是allocate函数最终使用malloc来分配内存，会有一些overhead开销。（这里的内存管理部分在其他课程）

可以看到deallocate调用的是operator delete()，而operator delete()调用C的free函数

一个重要的语法是：一个typename后直接加小括号，就是这个类的临时对象没有名称。

例如：（临时对象）allocator<int>()

在使用deallocate时，要返回分配的指针，和内存的大小。（内存大小是硬伤，没有人会去记住当初分配的多少内存）

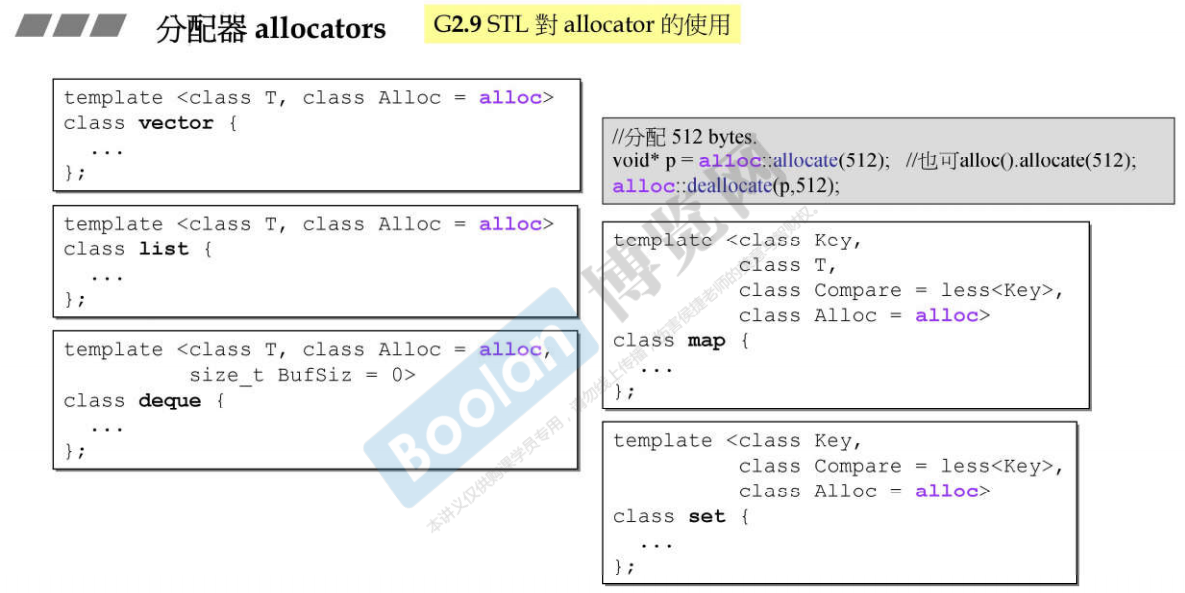


6、

在使用malloc分配内存时总会有内存开销，如果所需内存的size较小，那么开销的比例较大，不能忍受，而如果所需内存的size较大，那么开销的比例较小，可以忍受。

然而在现实中，开销的比例都会较大。在很多编译器中都是。

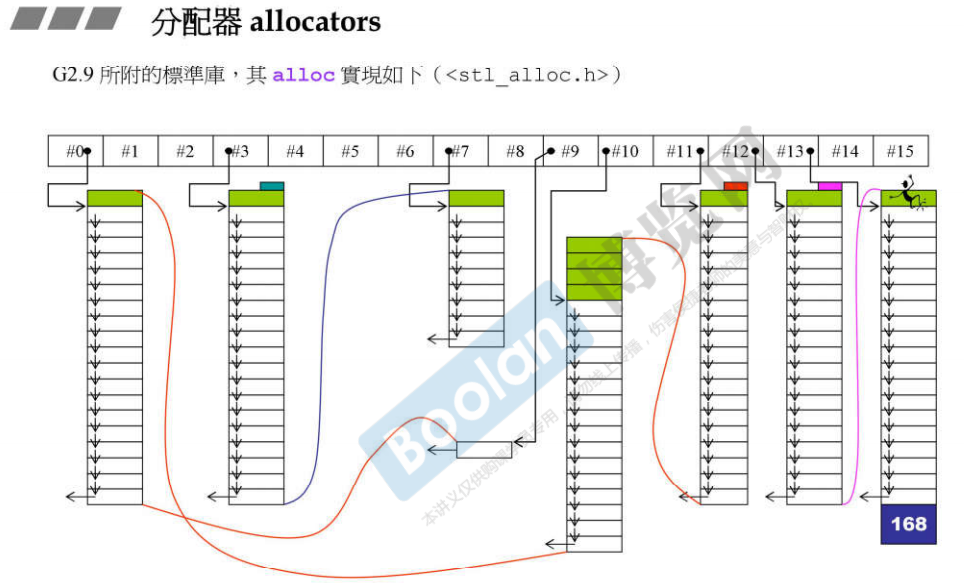
7、



G2.9使用的是alloc，而不是allocator这个类。

8、

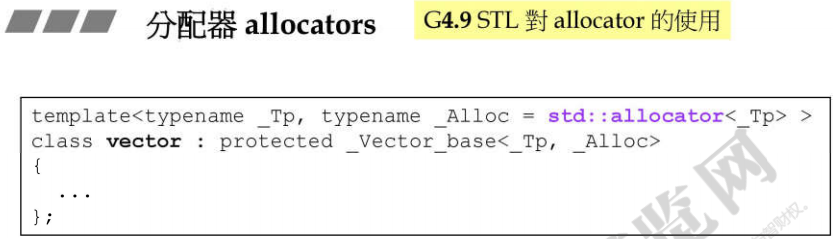
无论如何，所有分配内存的最后动作就是malloc，而malloc会在分配内存时带来额外开销，因此上面所说alloc的使用诉求是减少malloc的使用次数。

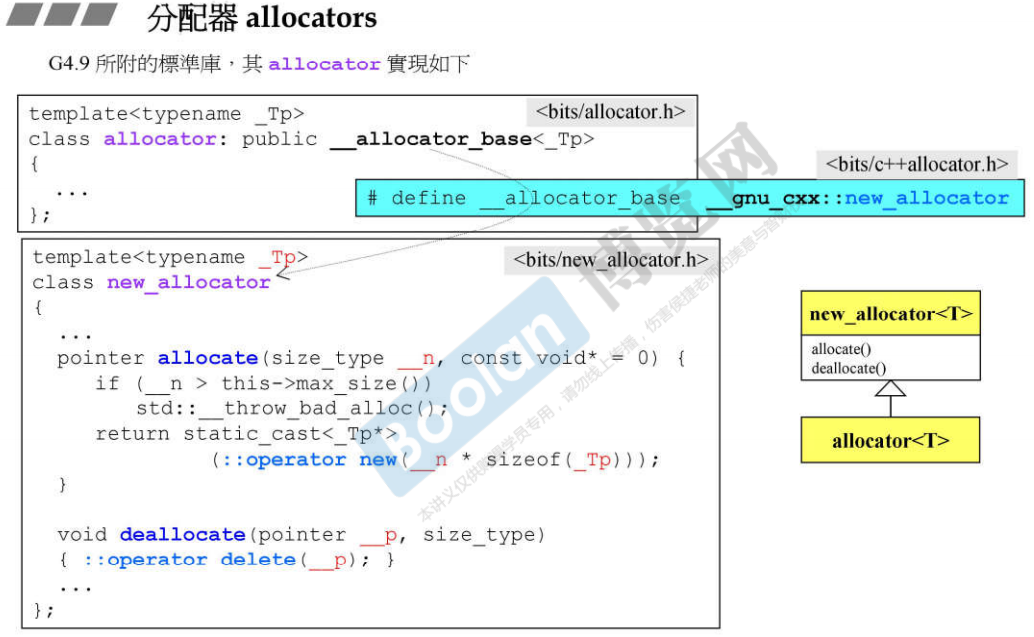


第一个链表负责8个字节，第二个链表负责16个字节，第三个链表负责24个字节…以此类推。这里最后负责链表结点负责128字节，不是168

这个分配器会将容器元素的大小调整到8字节的倍数。并且省去了每个元素的额外内存开销。举例子，如果原来的分配器，放入100w个元素，每个元素的额外开销是8个字节（消耗在cookie？）那么就会消耗掉800w个字节开销。而如果使用alloc这个分配器，这800w个字节就可省略去。（直观表达，不严谨，意思到位就行，实际还是有一些开销的，因为始终都要用到malloc，不过真的cookie会少很多很多）

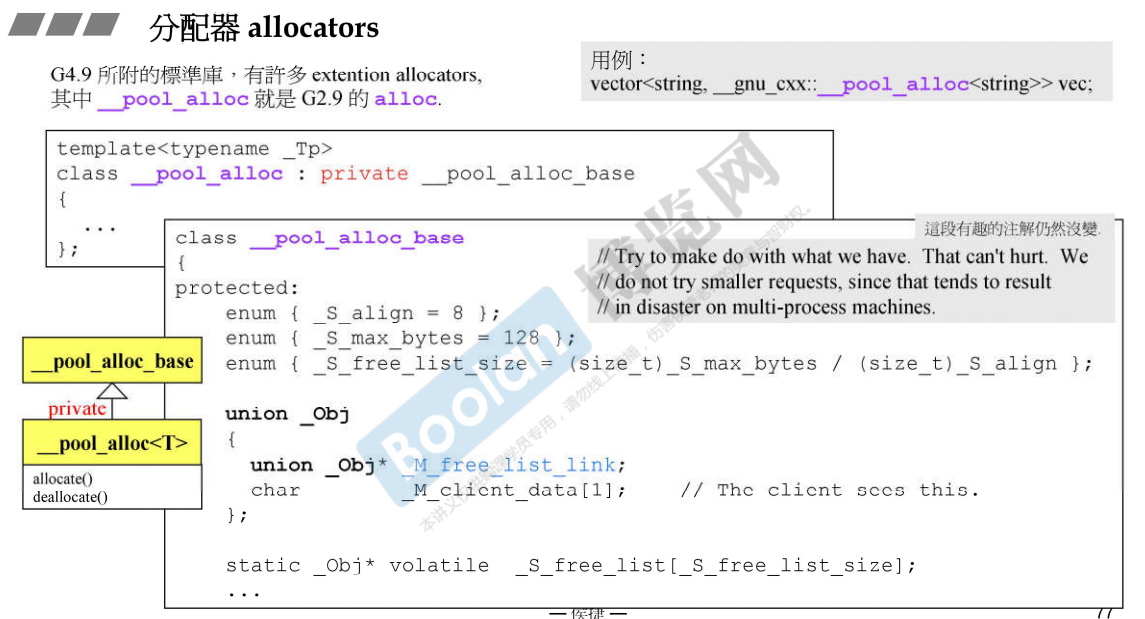
9、





到了G4.9就不用alloc了呢？？

看了容器源代码，它们得分配器还是用回了allocate，原因老师说自己也不知道，不过他说alloc有一些缺点。Alloc变成了一个扩充型的分配器，变成了pool alloc



想要用这个分配器可以使用扩展文件夹，按照图片用例来使用。