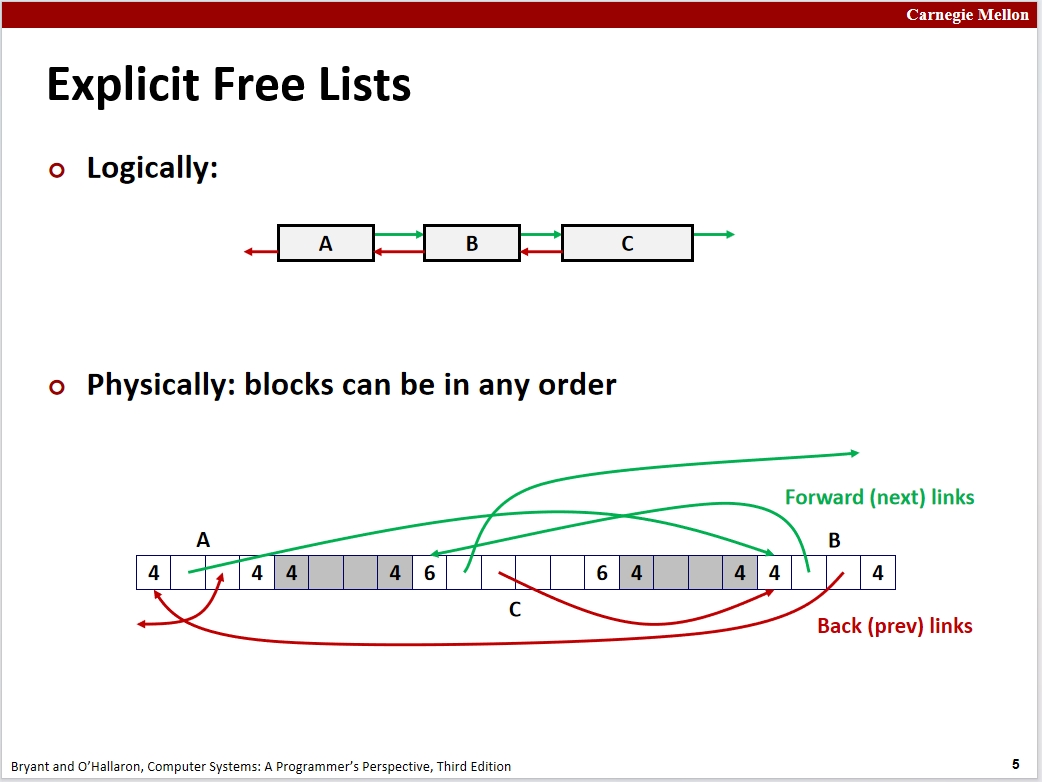


前面我们已经通过隐式空闲列表的方式来储存当前分配器的控制结构了。但是我们不难看到，对于空闲块中，一般来说都还有一块区域并不会被使用，或者说，这一块区域对于程序来说没有价值。正如前面我们使用一个特殊的位来利用信息使得本来同样的数据空间能够实现更多信息的储存一样。我们这里可以利用这些空闲的堆程序没有价值的数据空间来优化我们的程序性能。

就比如这里的**显式空闲双向链表**。通过在空闲块中插入我们的前驱和后继指针使得程序能够更快的查找对应的空闲块，且由于这些空间本来就不会被使用，所以这里提高了内存的利用率，在这同时还将搜索一个空闲块的时间复杂度优化到了与空闲块数量绑定的线性时间复杂度。



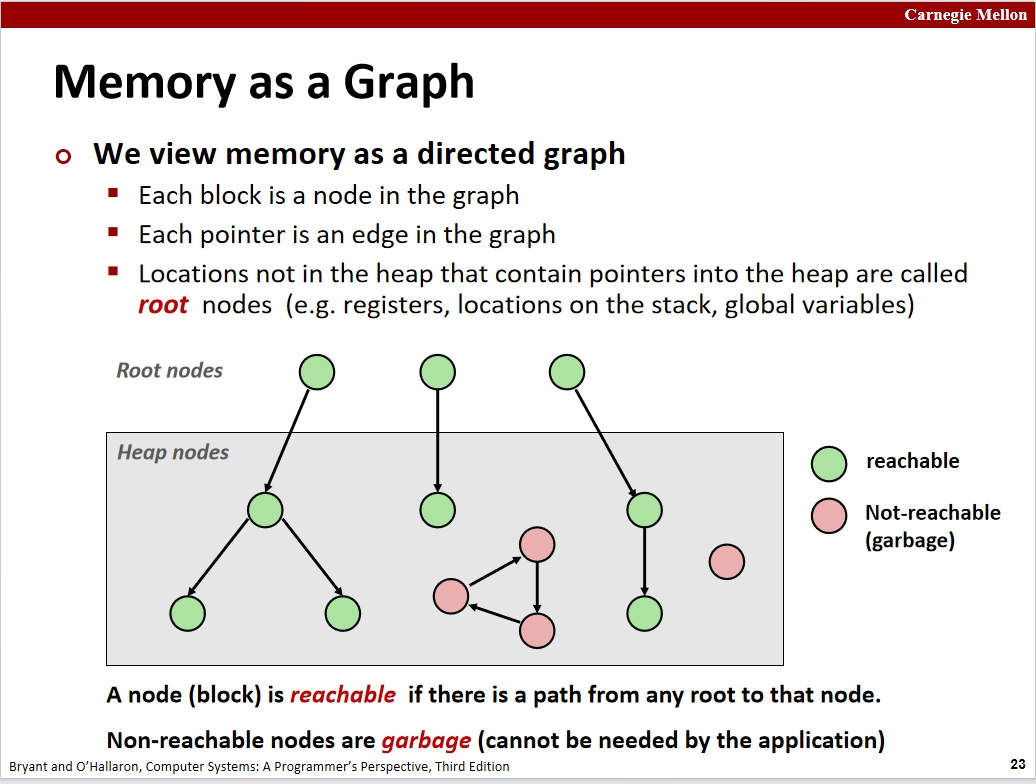
但同时需要注意的是，不要被一般的双向链表所局限了，我们这里的双向链表所能指向的地方是很多的，就比如上图中的情况。我们可以通过这个双向链表来扩展出多种不同的架构，以此来实现不同的功能。而且事实上要保持上图的有序逻辑结构在实际操作也是很麻烦。

我累了，直接看隐式内存回收进入网络。

隐式内存回收，顾名思义，我们不需要对内存进行显示的回收，系统会自动回收我们在程序运行阶段所分配的内存。这里简单讨论一种**垃圾回收**机制。一般是将那些永远不能再被引用的块视为垃圾，并将这些块进行回收。

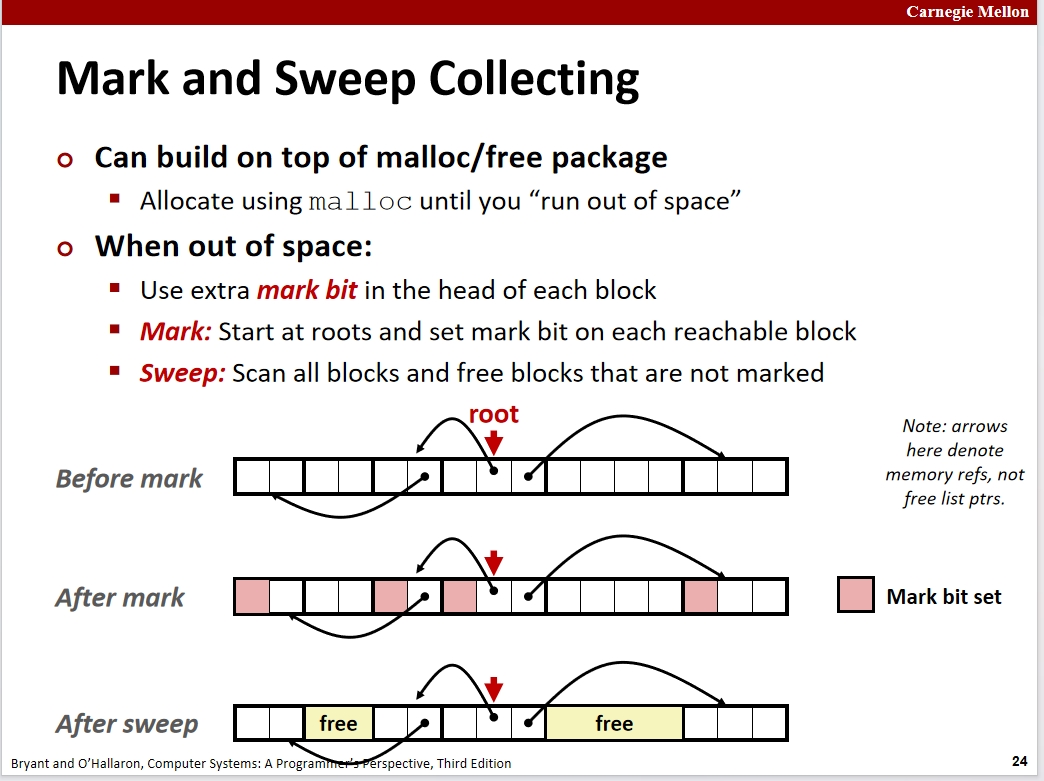
但现在的第一个问题是我们怎么知道一个数据块时垃圾。以一种朴素的视角来看的话，如果我们在之后程序的整个运行阶段都不会用到这段内存，那自然是垃圾，但这个对于系统来说是未知的，我们需要别的东西来进行界定。

就比如说如果一个块没有指向它的指针，我们可以视为这个块时一个垃圾块。程序通过扫描整个程序当前的指针查看指向的块，并推断当前是否存在着额外的没被引用的块，这些块就会被视为垃圾。



首先来看这个有趣的图吧。在内存系统中，我们将每个块抽象为一个点，将每个节点连接起来构成一个有向可达图。在这个图中，在这个有向图中，对于那些可以从堆外节点一直访问的堆内节点，标记为绿色，或者说标记为可达，这些节点是有效节点。对于那些堆内不可达的节点，标记为红色，视为垃圾。

在c/c++中，这种隐式的内存回收机制一般只是在内存分配器的下一层封装中的。来看一下这个隐式内存回收机制需要注意的东西。



首先，这种内存机制一般只有我们在遭遇内存不足时才会进行使用，频繁使用会导致效率的下降。当我们消耗完当前的可用内存或者说没有找到适配的内存块时，程序会触发隐式内存回收机制。

要建立这个机制，我们需要先给每个块附加上一个信息。同之前有效位一样，我们可以使用一个不会影响数据的位来进行这个信息的储存。而这个信息指的是当前块是否有指针指向它。

有了这个信息的支持，我们才能够进行之后的垃圾回收机制的俩个阶段。首先，**Mark**，系统会从(堆)根节点出发，仅仅只是去遍历一遍所有的块并设置经过的块的上面提到的有效位。

所以这个阶段就是标记所有非垃圾的节点。再接下来我们就可以去释放那些未被标记的块了，也就是说，这种架构下的垃圾回收需要对堆内存进行俩次遍历。

需要注意的是，这里的起始指针会是整个进程中使用到的指针，包括但不限于当前活跃寄存器中的指针，当前栈帧的指针，程序可读区域的指针等。

**开摆！！！！！**

嘻嘻

