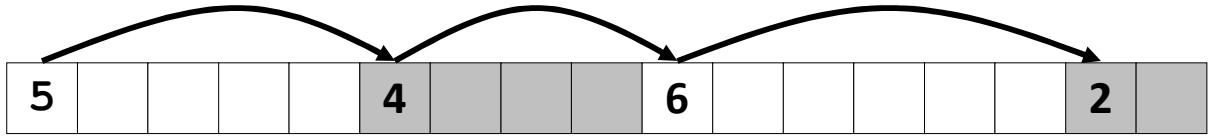
动态内存分配：进阶概念
(Dynamic Memory Allocation)

主要内容

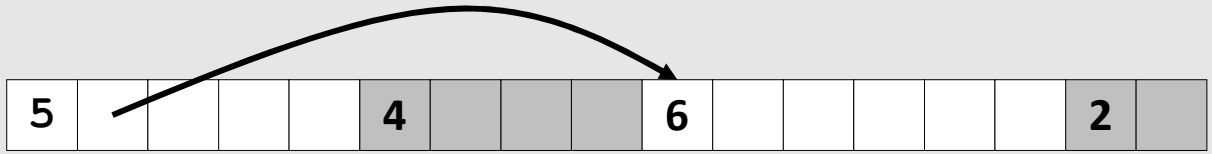
- 显式空闲列表
- 分离空闲列表
- 垃圾回收
- 内存相关的风险和陷阱

追踪空闲块的方法

- 方法 1：隐式链表（Implicit list）使用块的长度信息，把所有块串联起来



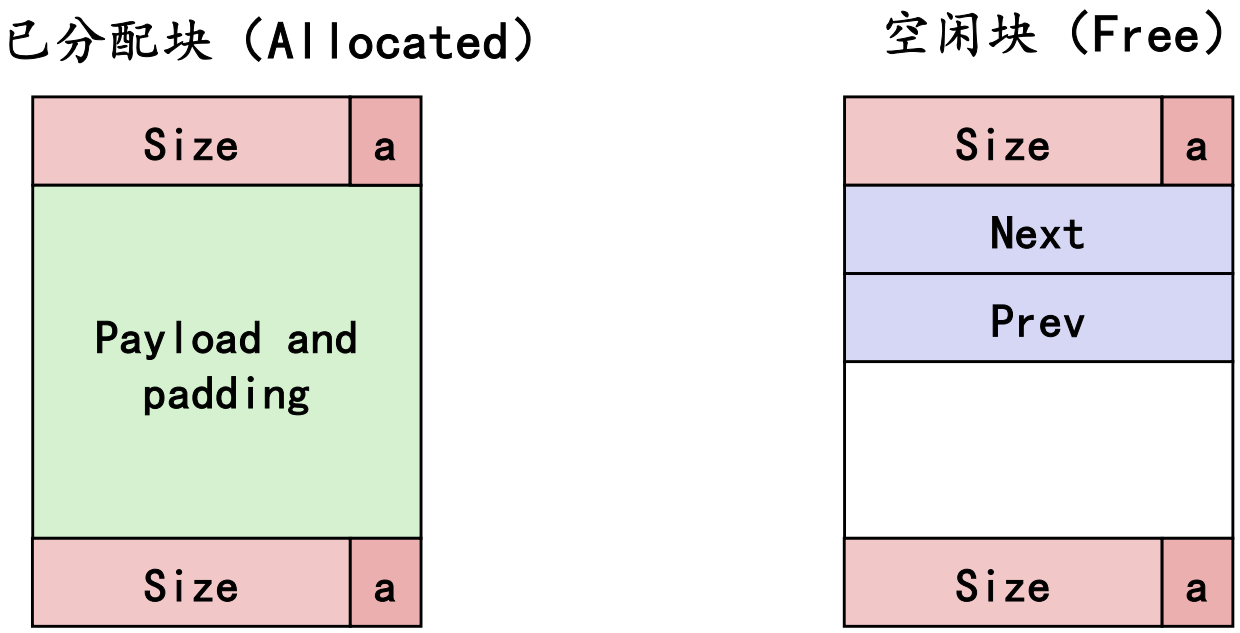
- 方法 2：显式链表（Explicit list）仅在空闲块之间使用指针建立链表



- 方法 3：分离空闲链表（Segregated free list）为不同大小类别维护不同的空闲链表

- 方法 4：按大小排序的块（Blocks sorted by size）
  - 使用平衡树（如红黑树），在每个空闲块里保存指针，并以块的长度作为关键字进行组织

显式空闲链表

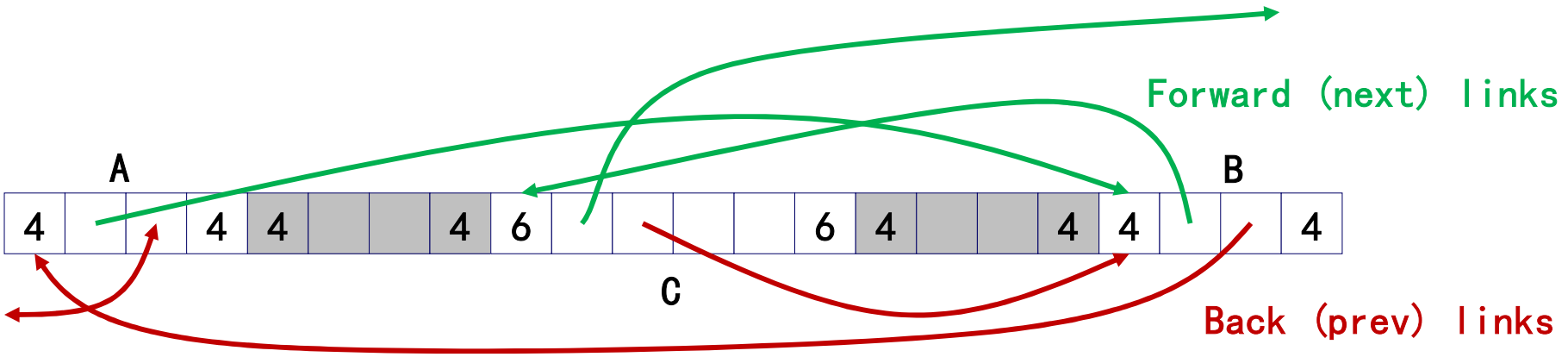


- 我们维护的是空闲块的链表，而不是所有块的链表。
  - “下一个”空闲块的位置可能在堆的任何地方，因此我们必须在块中保存前向和后向指针，而不仅仅是块的大小。
  - 仍然需要使用边界标记（boundary tags），以便在释放时可以合并相邻空闲块。
  - 幸运的是，我们只需要在空闲块里维护这些额外指针，因此可以利用空闲块的有效负载区来存放这些指针，而不会浪费额外的空间。

显式空闲链表（Explicit Free Lists）

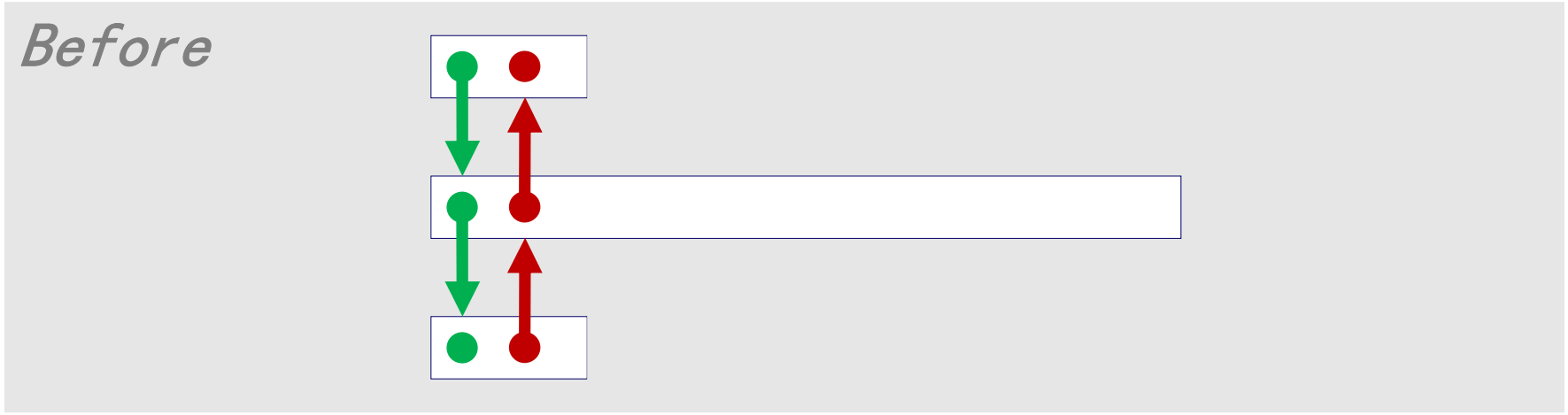
- 逻辑上：

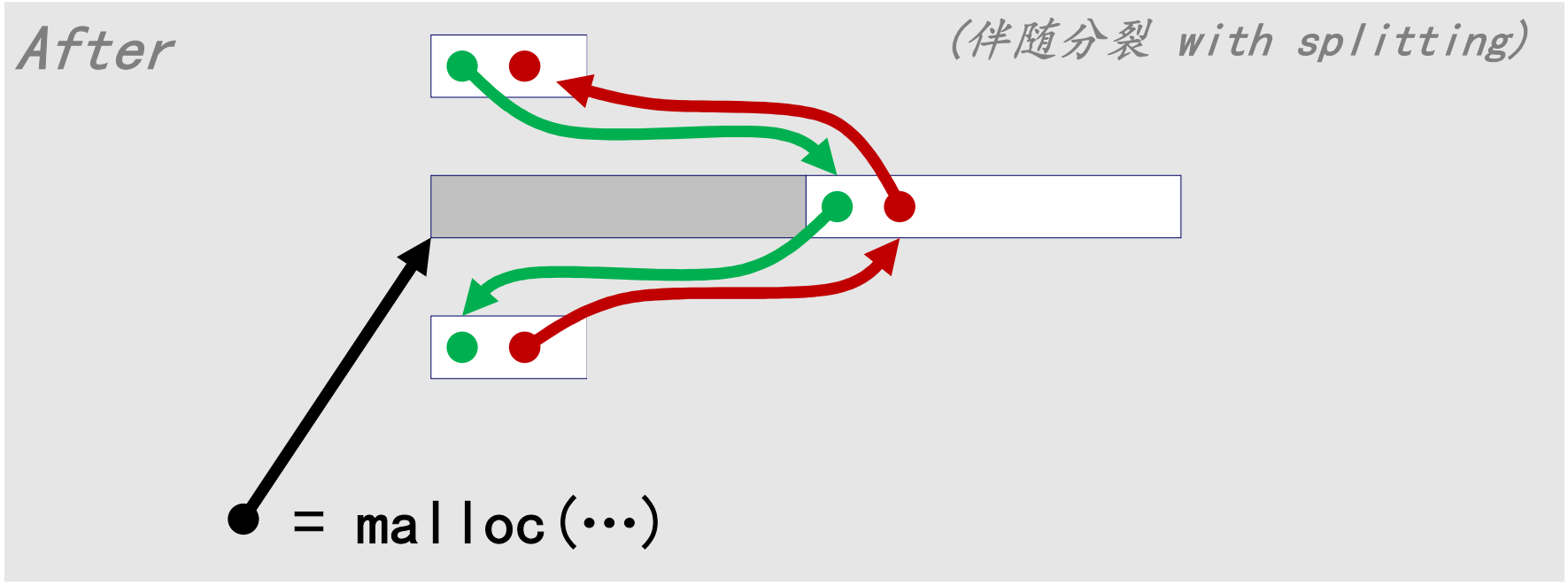
- 物理上：空闲块在堆中的位置可以是任意的，不需要相邻。
  - 前向指针（Forward, next links）用来指向下一个空闲块
  - 后向指针（Back, prev links）用来指向前一个空闲块。



从显式空闲链表中分配内存

概念图



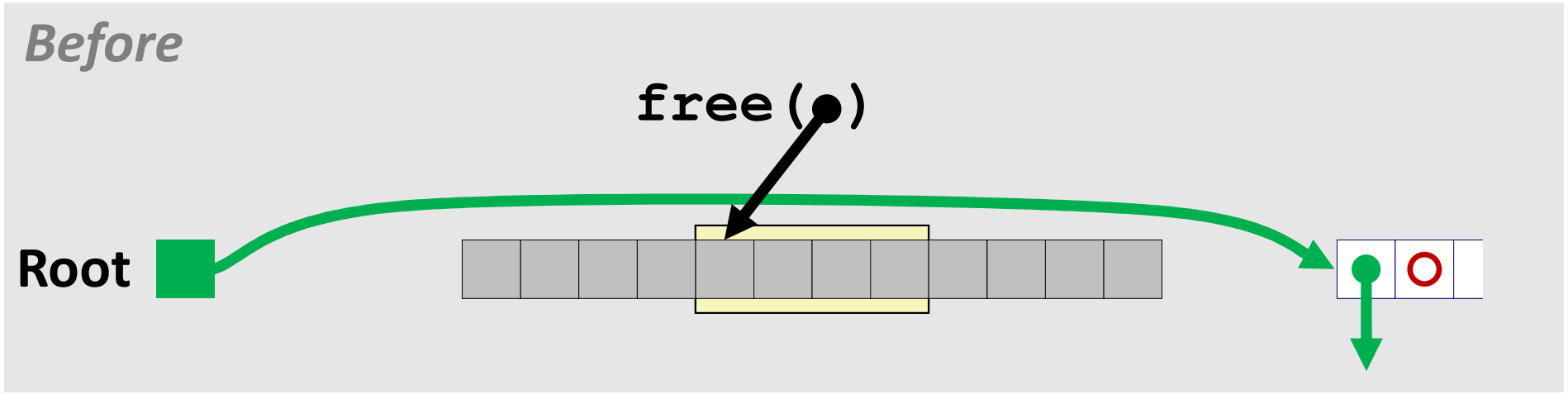


使用显式空闲链表释放内存

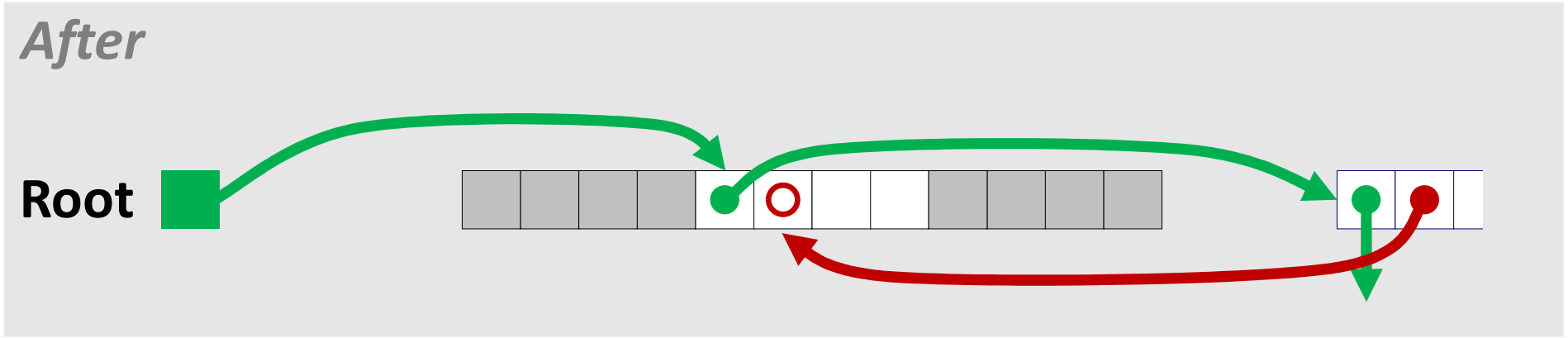
- 插入策略（Insertion policy）：在空闲链表中，新的空闲块应该放在哪里？
- LIFO（后进先出）策略
  - 将释放的块插入到空闲链表的开头
  - 优点（Pro）：简单，耗时为常数时间
  - 缺点（Con）：研究表明，这种方式的碎片化情况通常比按地址顺序更严重
- 按地址顺序策略（Address-ordered policy）
  - 将释放的块按地址顺序插入空闲链表，始终满足：
    addr(prev) < addr(curr) < addr(next)
  - 缺点（Con）：需要进行查找操作
  - 优点（Pro）：研究表明，这种方式的碎片化程度比 LIFO 更低

使用 LIFO 策略释放（案例 1）

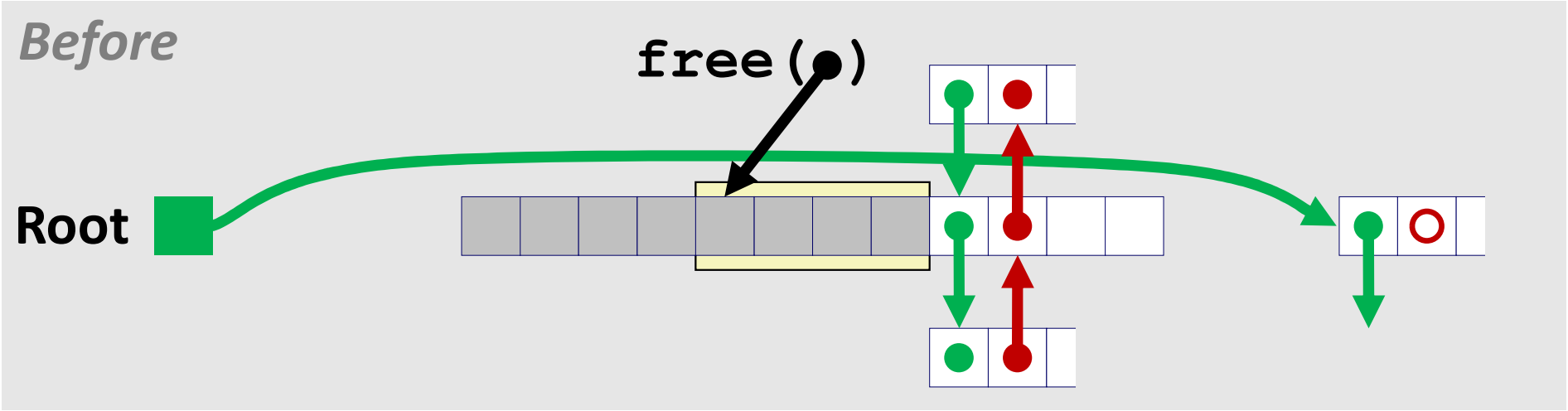
概念图



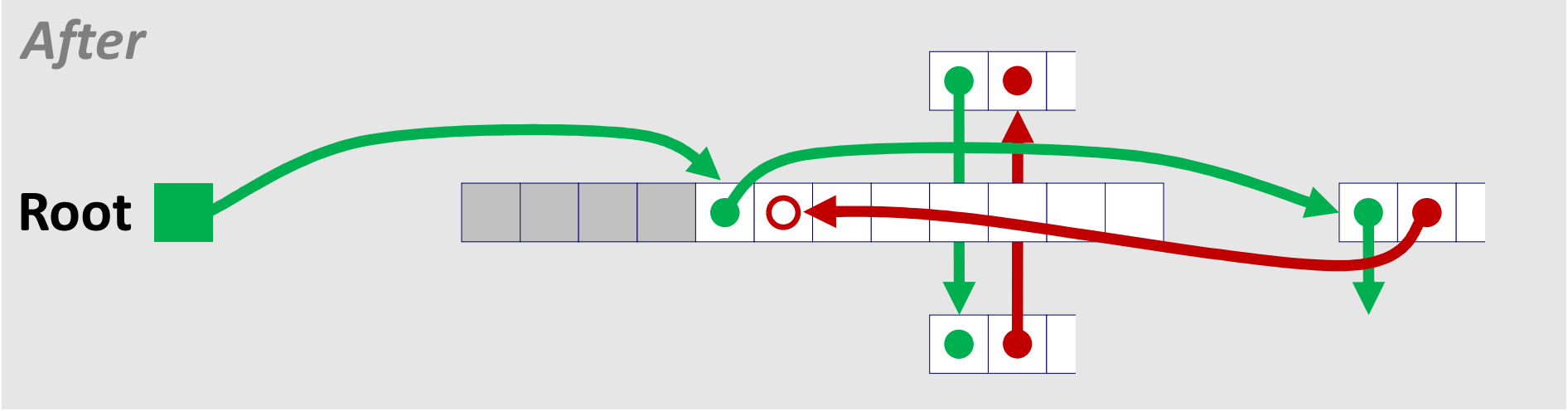
- 将释放的块插入到列表的根部



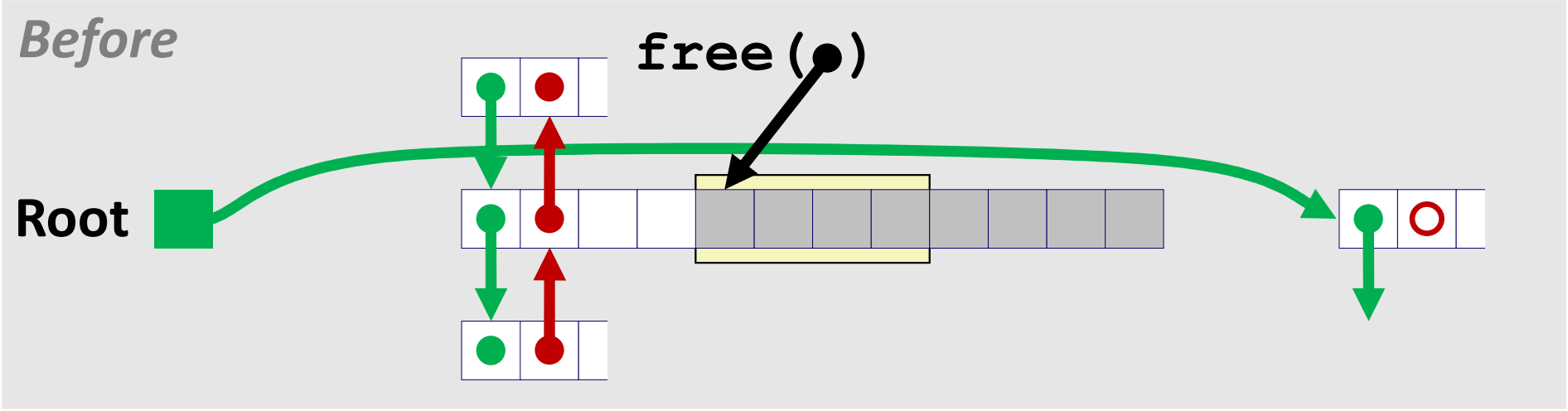
使用 LIFO 策略释放（案例 2）



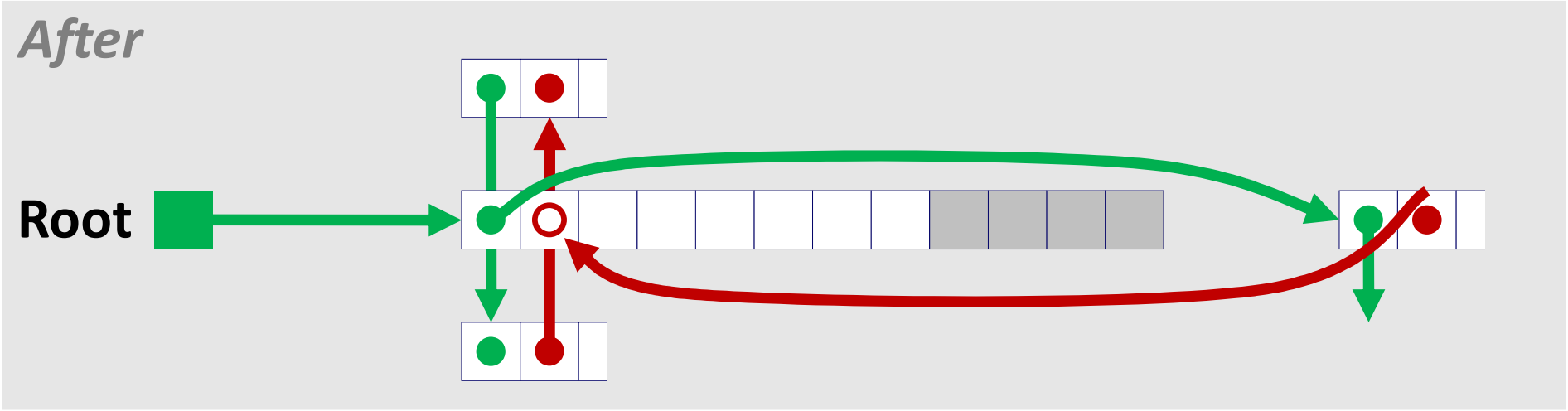
- 拼接后继块，合并两个内存块，并将新块插入列表的根部



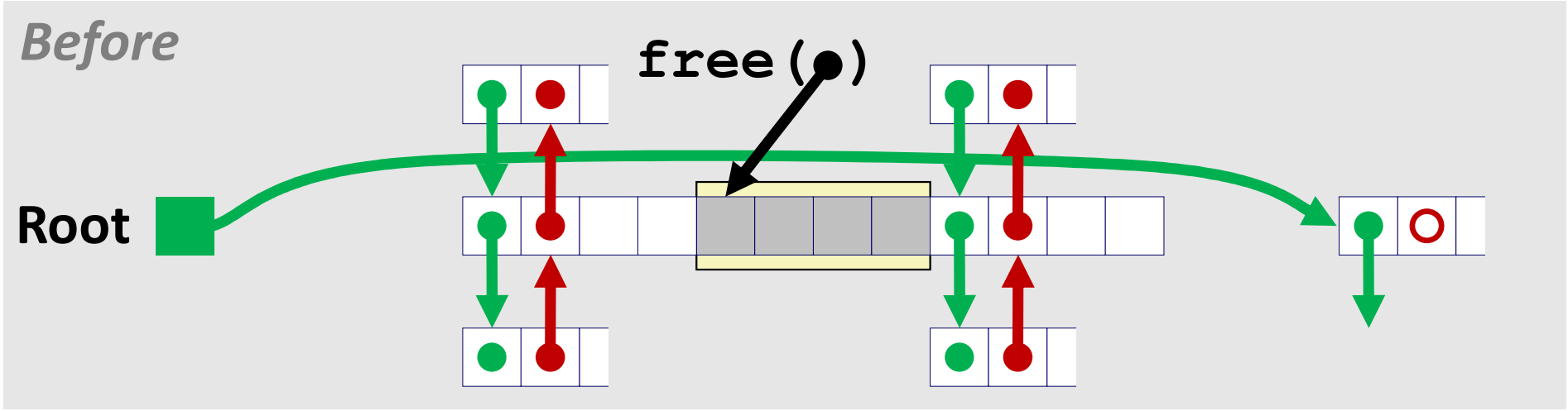
使用 LIFO 策略释放（案例 3）



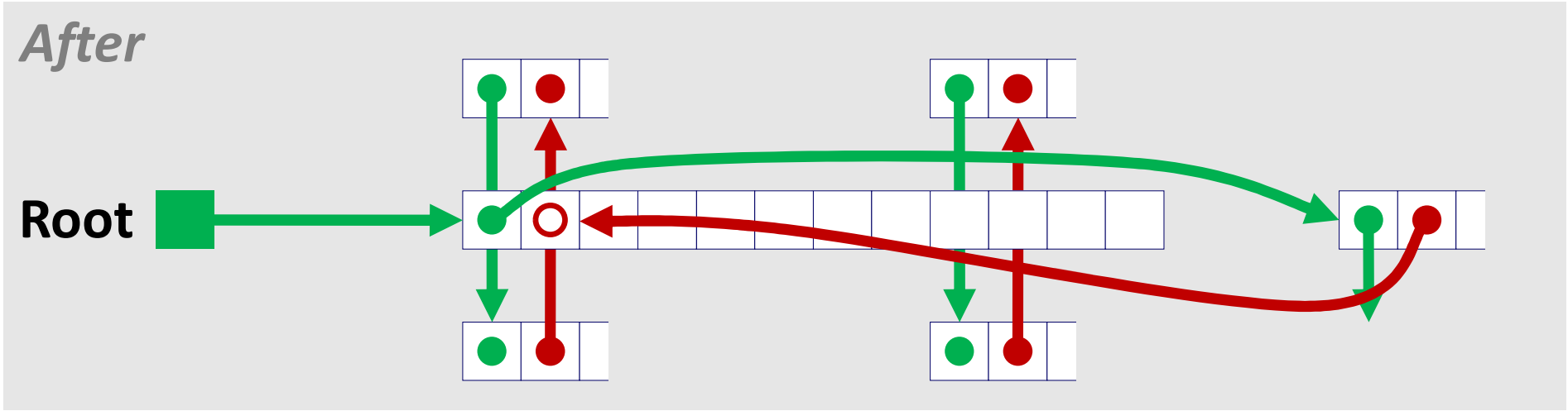
- 拼接前一个块，合并两个内存块，并将新块插入到列表的根部



使用 LIFO 策略释放（案例 4）



- 拼接前驱块和后继块，合并所有 3 个内存块，并将新块插入列表的根部



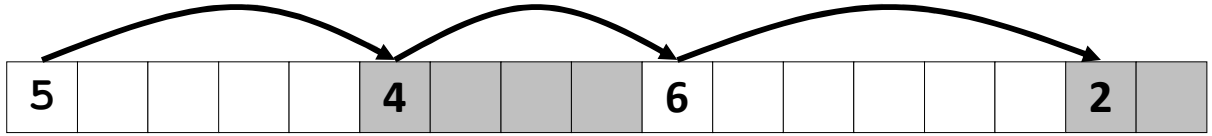
显式链表总结

- 与隐式链表的比较
  - 分配操作的时间复杂度与*空闲块*的数量成线性关系，而不是所有块。
    - 当内存大部分已被使用时，分配会快得多。
  - 分配和释放操作稍微复杂一些，因为需要把块插入或移出链表。
  - 需要额外的空间存储指针（每个块需要 2 个额外的字）。
    - 这会不会增加内部碎片？

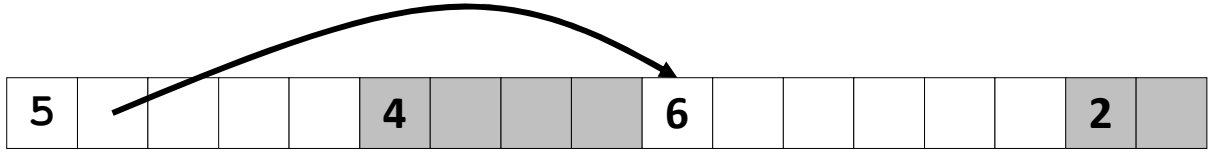
- 链表最常见的用法
  - 通常与分离空闲链表（segregated free lists）结合使用。为不同大小的块维护多个链表，或者为不同类型的对象维护多个链表。

追踪空闲块的方法

- 方法 1：隐式链表（Implicit list）使用块的长度信息，把所有块串联起来



- 方法 2：显式链表（Explicit list）仅在空闲块之间使用指针建立链表

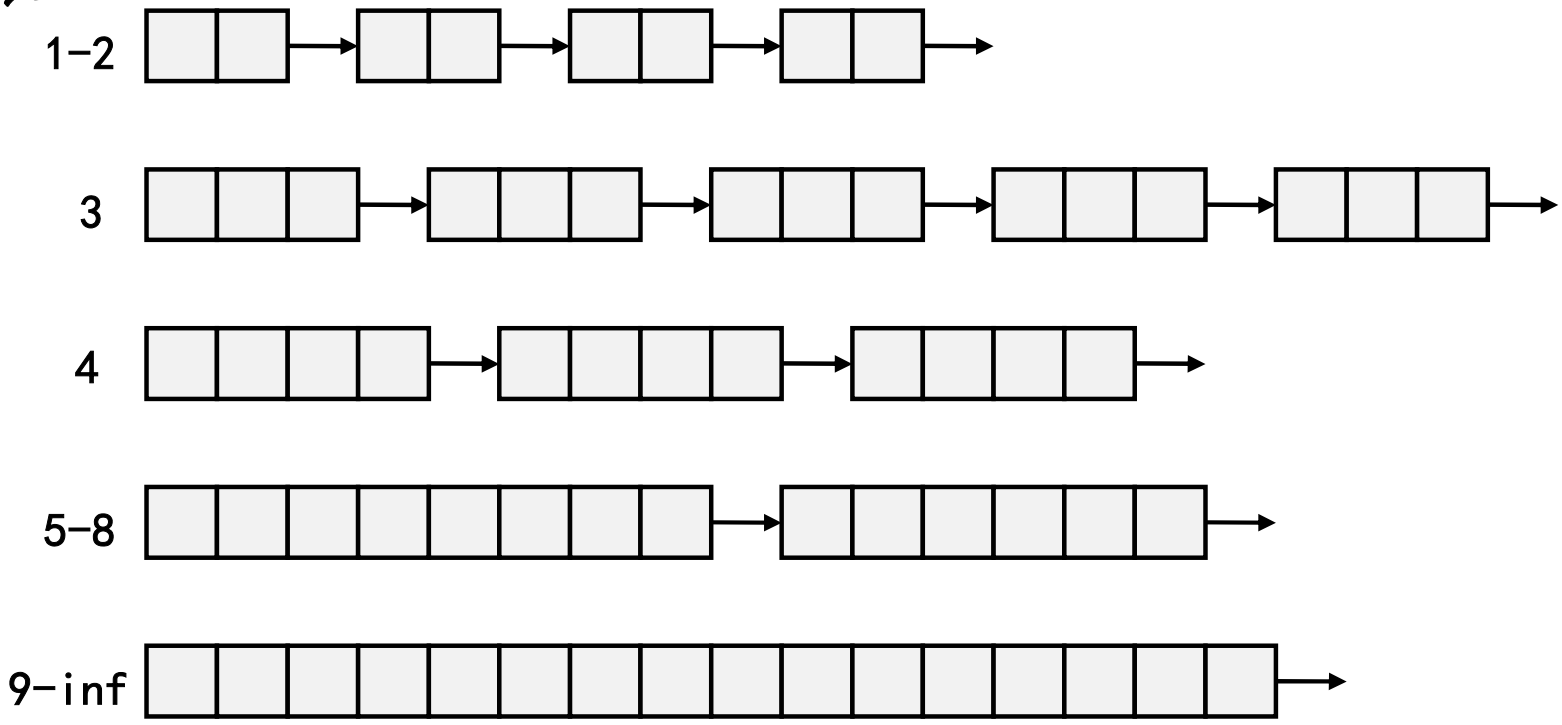


- 方法 3：分离空闲链表（Segregated free list）为不同大小类别维护不同的空闲链表

- 方法 4：按大小排序的块（Blocks sorted by size）
  - 使用平衡树（如红黑树），在每个空闲块里保存指针，并以块的长度作为关键字进行组织

分离链表（Seglist）分配器

- 每一个 大小类别（size class） 的块都有自己独立的空闲链表



- 通常会为每一个小的大小类别单独建立一个链表
- 对于更大的块大小：通常每个 2 的幂次大小对应一个链表