Dynamic Memory Allocation: Advanced Concepts

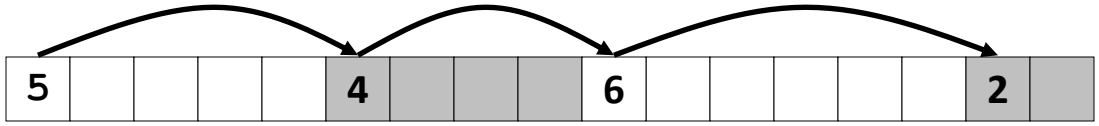
These slides adapted from materials provided by the textbook authors.

Dynamic Memory Allocation

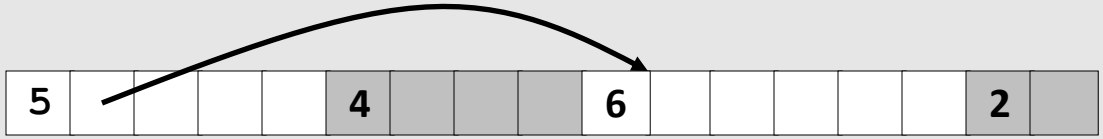
- **Explicit free lists**
- Segregated free lists
- Garbage collection
- Memory-related perils and pitfalls

Keeping Track of Free Blocks

- **Method 1: *Implicit free list* using length—links all blocks**

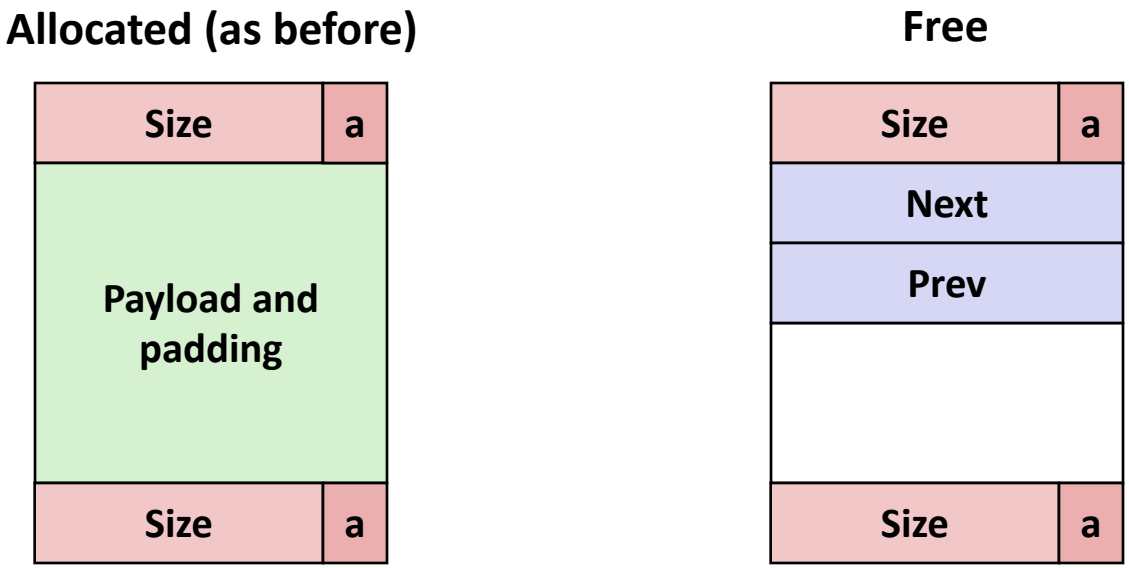


- **Method 2: *Explicit free list* among the free blocks using pointers**



- **Method 3: *Segregated free list***
  - Different free lists for different size classes
- **Method 4: *Blocks sorted by size***
  - Can use a balanced tree (e.g. Red-Black tree) with pointers within each free block, and the length used as a key

Explicit Free Lists

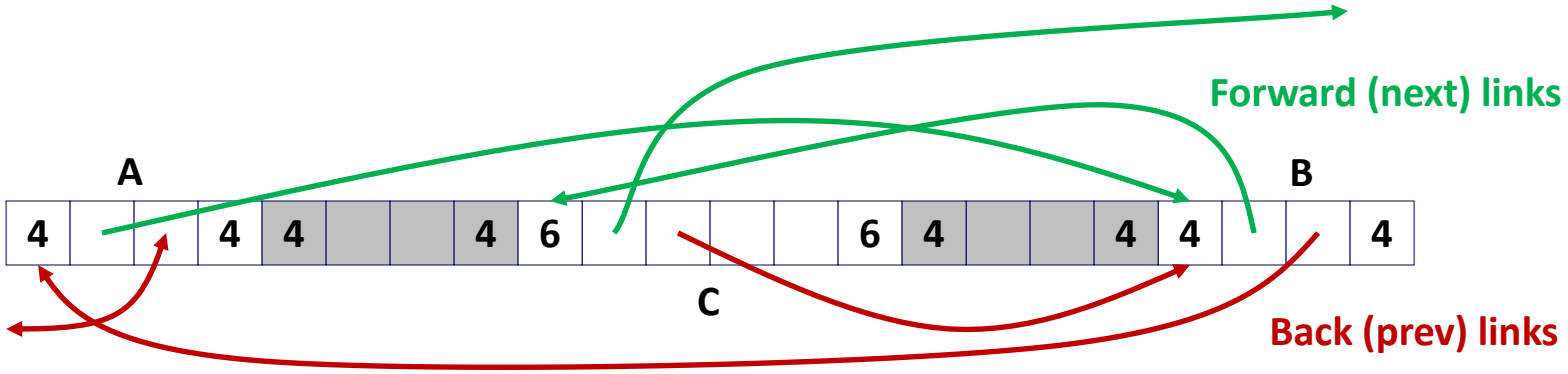


- **Maintain list(s) of *free* blocks, not *all* blocks**
  - The "next" free block could be anywhere
    - So we need to store forward/back pointers, not just sizes
  - Still need boundary tags for coalescing
  - Luckily we track only free blocks, so we can use payload area

Explicit Free Lists

- Logically:

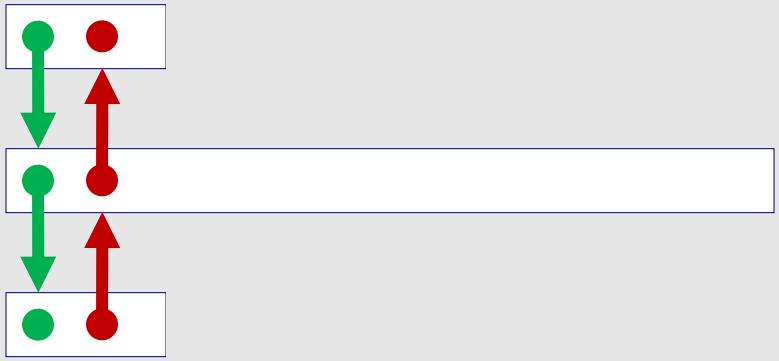
- Physically: blocks can be in any order



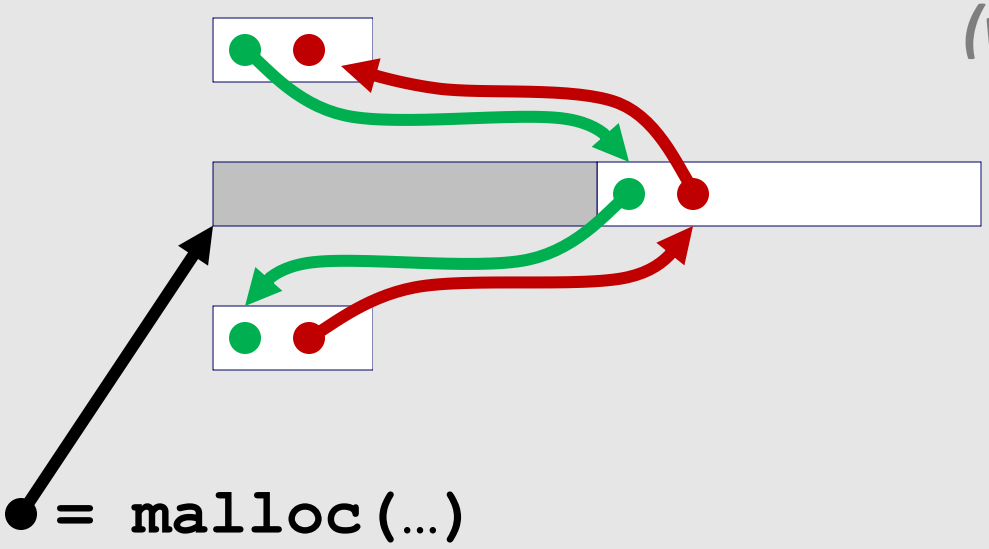
Allocating From Explicit Free Lists

conceptual graphic

Before



After *(with splitting)*



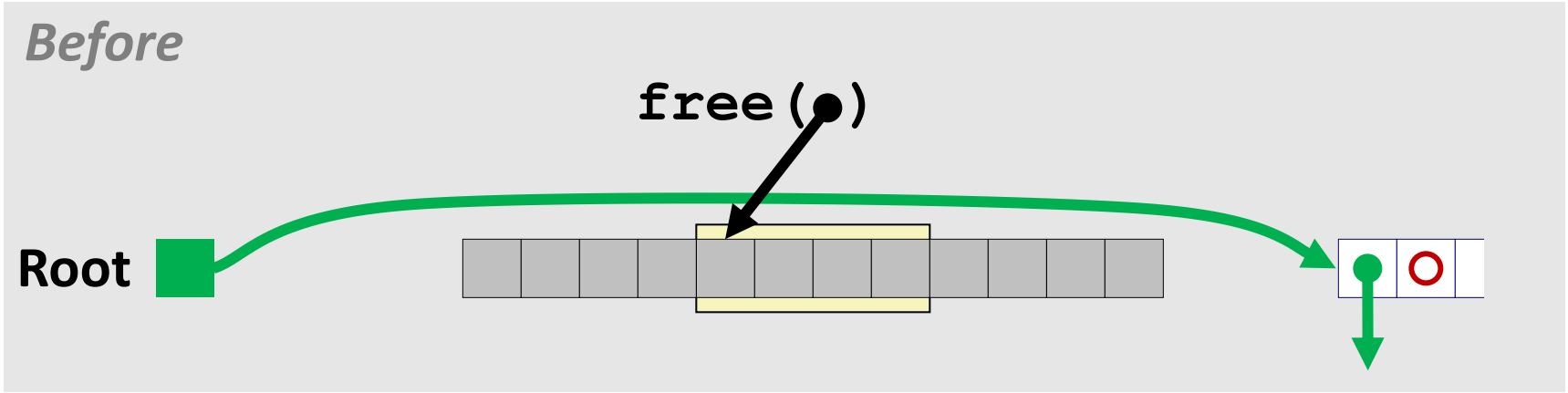
Freeing With Explicit Free Lists

- ***Insertion policy*: Where in the free list do you put a newly freed block?**
- **LIFO (last-in-first-out) policy**
  - Insert freed block at the beginning of the free list
  - ***Pro:*** simple and constant time
  - ***Con:*** studies suggest fragmentation is worse than address ordered

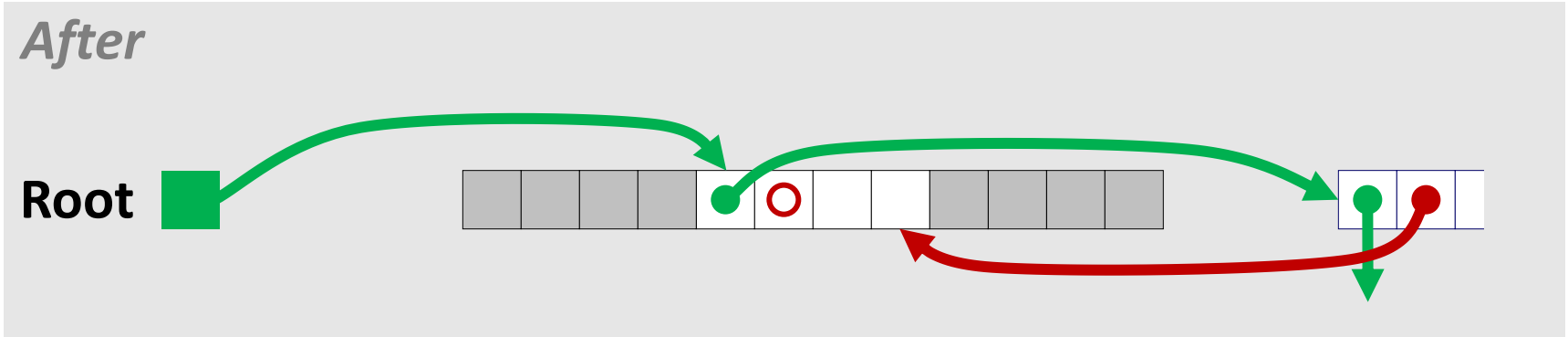
- **Address-ordered policy**
  - Insert freed blocks so that free list blocks are always in address order:
    addr(prev) < addr(curr) < addr(next)
  - ***Con:*** requires search
  - ***Pro:*** studies suggest fragmentation is lower than LIFO

Freeing With a LIFO Policy (Case 1)

conceptual graphic

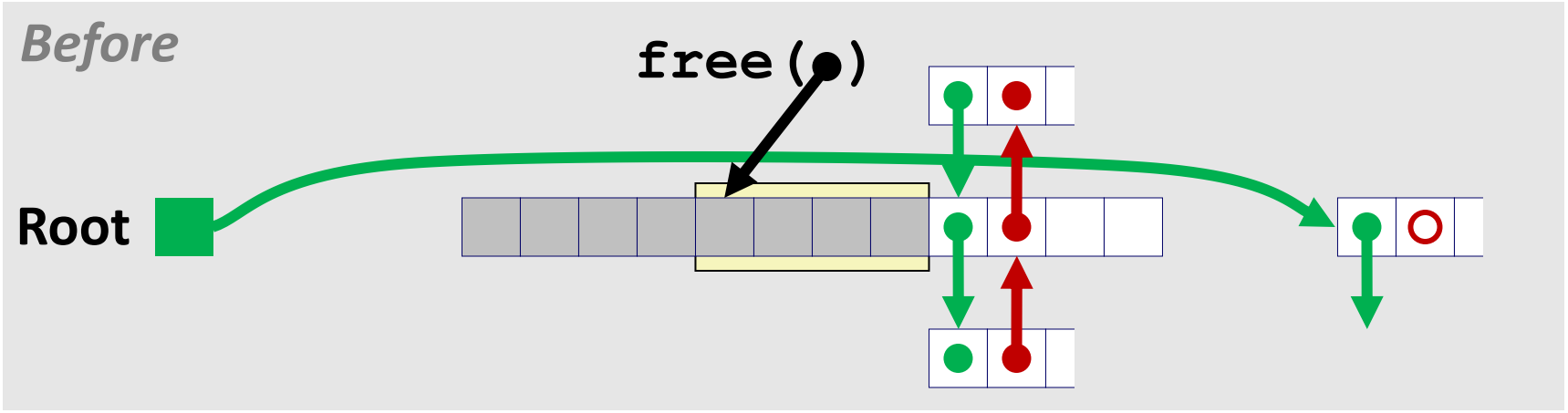


- Insert the freed block at the root of the list

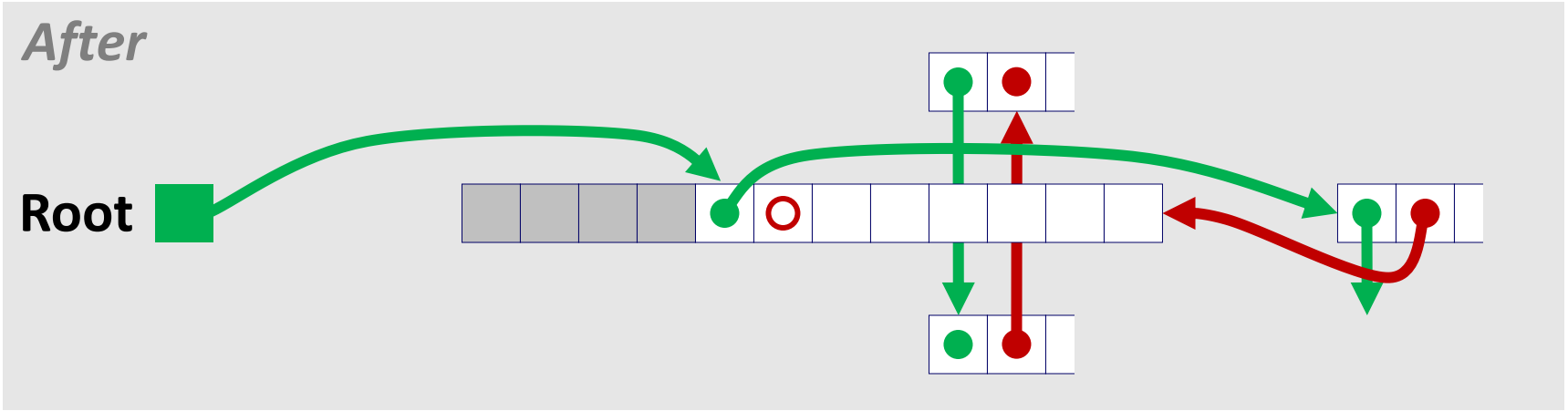


Freeing With a LIFO Policy (Case 2)

conceptual graphic

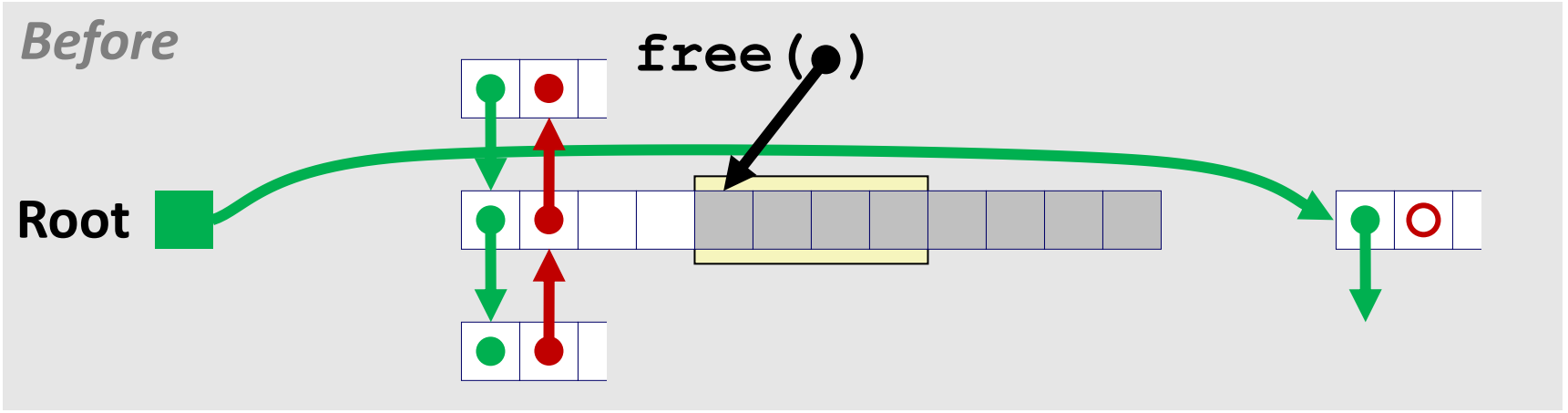


- Splice out successor block, coalesce both memory blocks and insert the new block at the root of the list

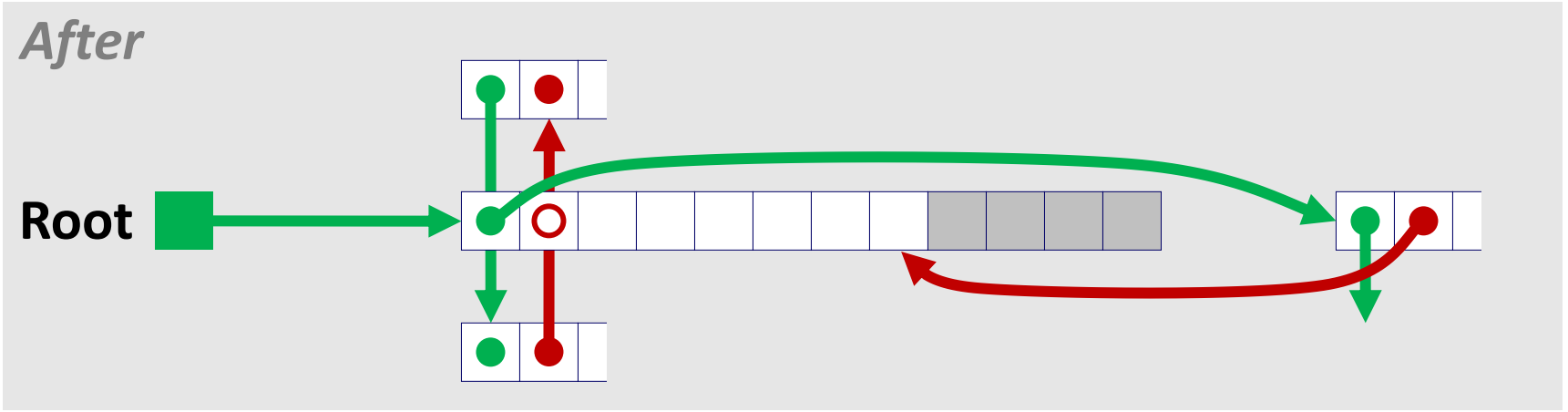


Freeing With a LIFO Policy (Case 3)

conceptual graphic

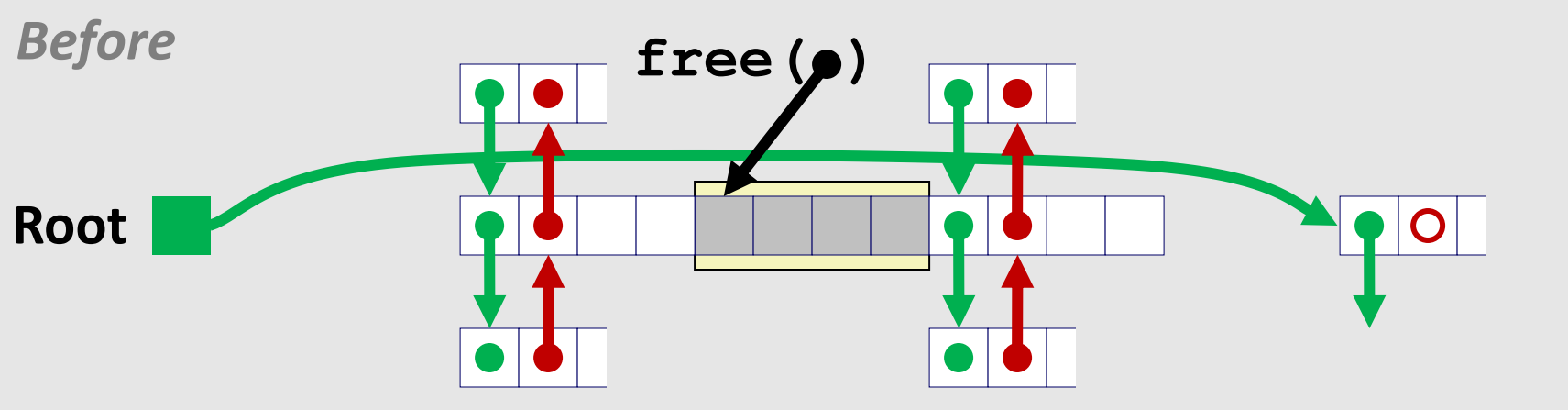


- Splice out predecessor block, coalesce both memory blocks, and insert the new block at the root of the list

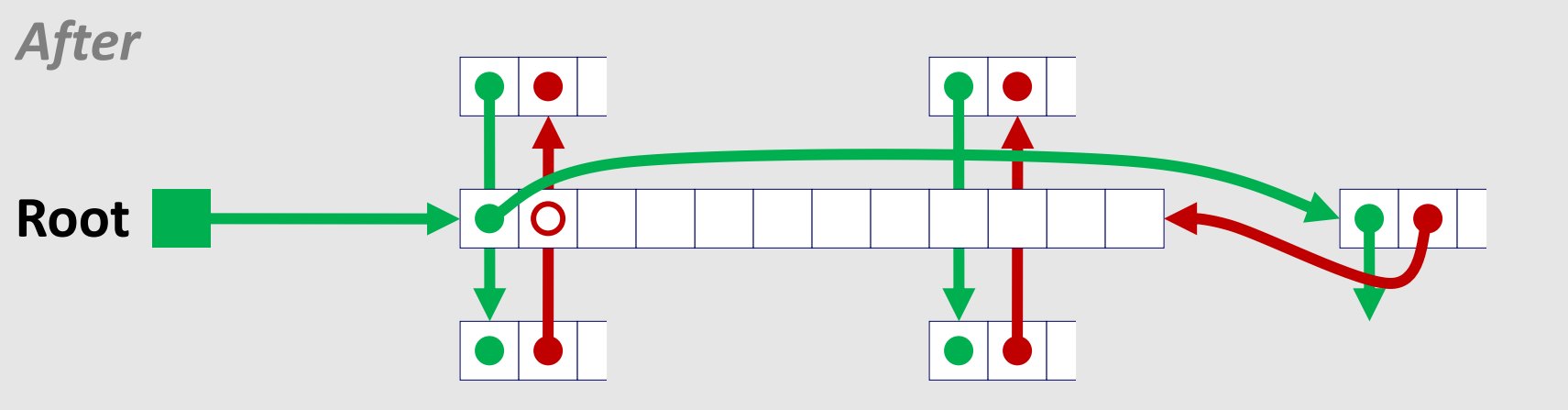


Freeing With a LIFO Policy (Case 4)

conceptual graphic



- Splice out predecessor and successor blocks, coalesce all 3 memory blocks and insert the new block at the root of the list



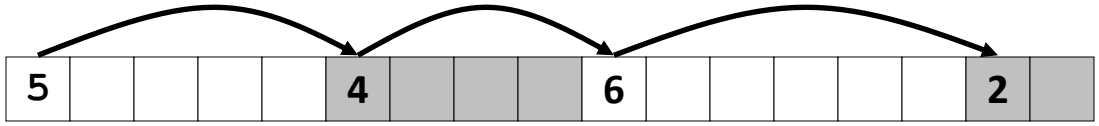
Explicit List Summary

- **Comparison to implicit list:**
  - Allocate is linear time in number of ***free*** blocks instead of ***all*** blocks
    - ***Much faster*** when most of the memory is full
  - Slightly more complicated allocate and free since needs to splice blocks in and out of the list
  - Some extra space for the links (2 extra words needed for each block)
    - Does this increase internal fragmentation?

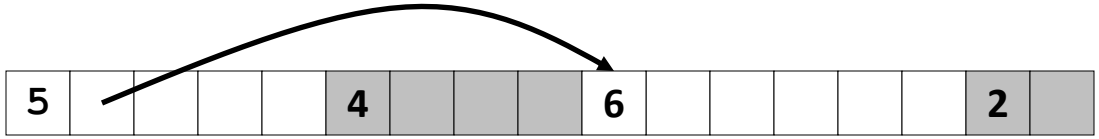
- **Most common use of linked lists is in conjunction with segregated free lists**
  - Keep multiple linked lists of different size classes, or possibly for different types of objects

Keeping Track of Free Blocks

- **Method 1: *Implicit list* using length—links all blocks**



- **Method 2: *Explicit list* among the free blocks using pointers**



- **Method 3: *Segregated free list***
  - Different free lists for different size classes

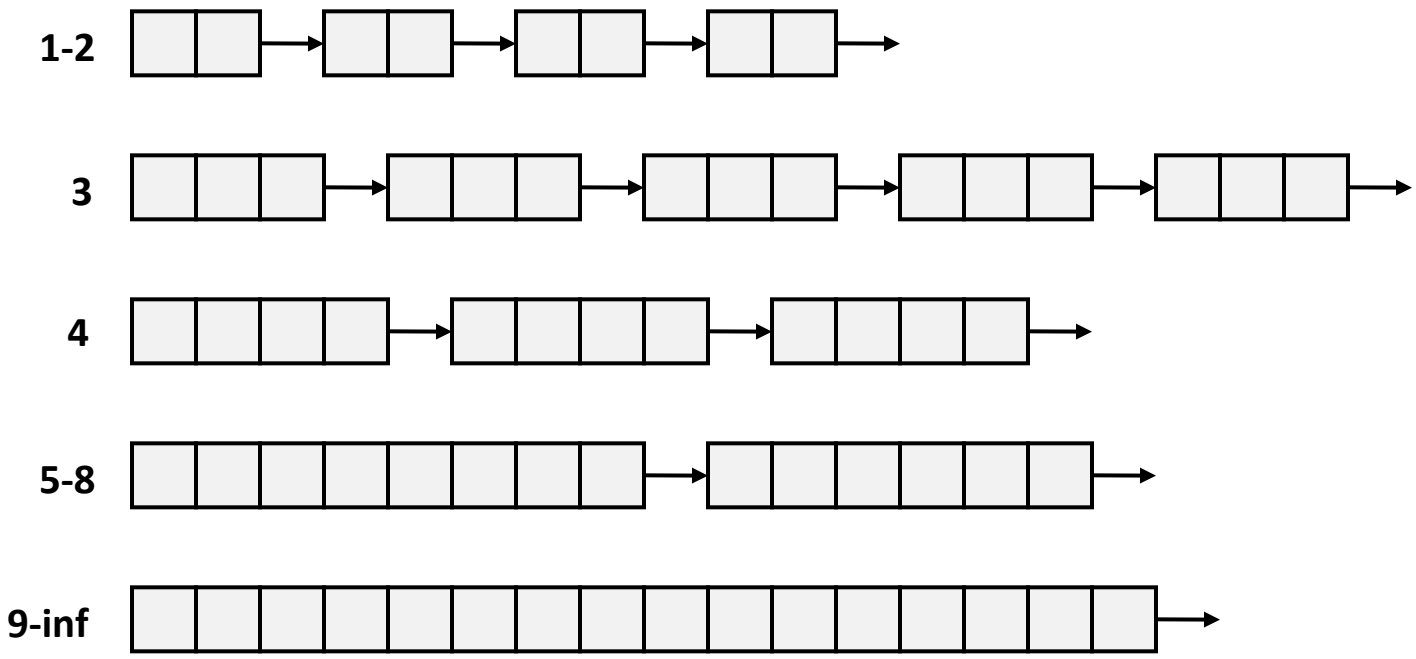
- **Method 4: *Blocks sorted by size***
  - Can use a balanced tree (e.g. Red-Black tree) with pointers within each free block, and the length used as a key

Dynamic Memory Allocation

- Explicit free lists
- **Segregated free lists**
- Garbage collection
- Memory-related perils and pitfalls

Segregated List (Seglist) Allocators

- Each *size class* of blocks has its own free list



- Often have separate classes for each small size
- For larger sizes: One class for each two-power size

Seglist Allocator

- **Given an array of free lists, each one for some size class**

- **To allocate a block of size *n*:**
  - Search appropriate free list for block of size *m* > *n*
  - If an appropriate block is found:
    - Split block and place fragment on appropriate list (optional)
  - If no block is found, try next larger class
  - Repeat until block is found

- **If no block is found:**
  - Request additional heap memory from OS (using `sbrk()`)
  - Allocate block of *n* bytes from this new memory
  - Place remainder as a single free block in largest size class.

Seglist Allocator (cont.)

- **To free a block:**
  - Coalesce and place on appropriate list

- **Advantages of seglist allocators**
  - Higher throughput
    - log time for power-of-two size classes
  - Better memory utilization
    - First-fit search of segregated free list approximates a best-fit search of entire heap.
    - Extreme case: Giving each block its own size class is equivalent to best-fit.

More Info on Allocators

- **D. Knuth, "*The Art of Computer Programming*", 2nd edition, Addison Wesley, 1973**
  - The classic reference on dynamic storage allocation

- **Wilson et al, "*Dynamic Storage Allocation: A Survey and Critical Review*", Proc. 1995 Int'l Workshop on Memory Management, Kinross, Scotland, Sept, 1995.**
  - Comprehensive survey
  - Available from CS:APP student site (csapp.cs.cmu.edu)