Lecture 17: Dynamic Memory (cont'd)

CS 105 March 27, 2019

Dynamic Memory Allocation Goals

- Provide memory (in heap) to a running program
- Recycle memory when necessary
- High throughput
- Good memory usage
  - Avoid fragmentation

Dynamic Memory Allocation Basics

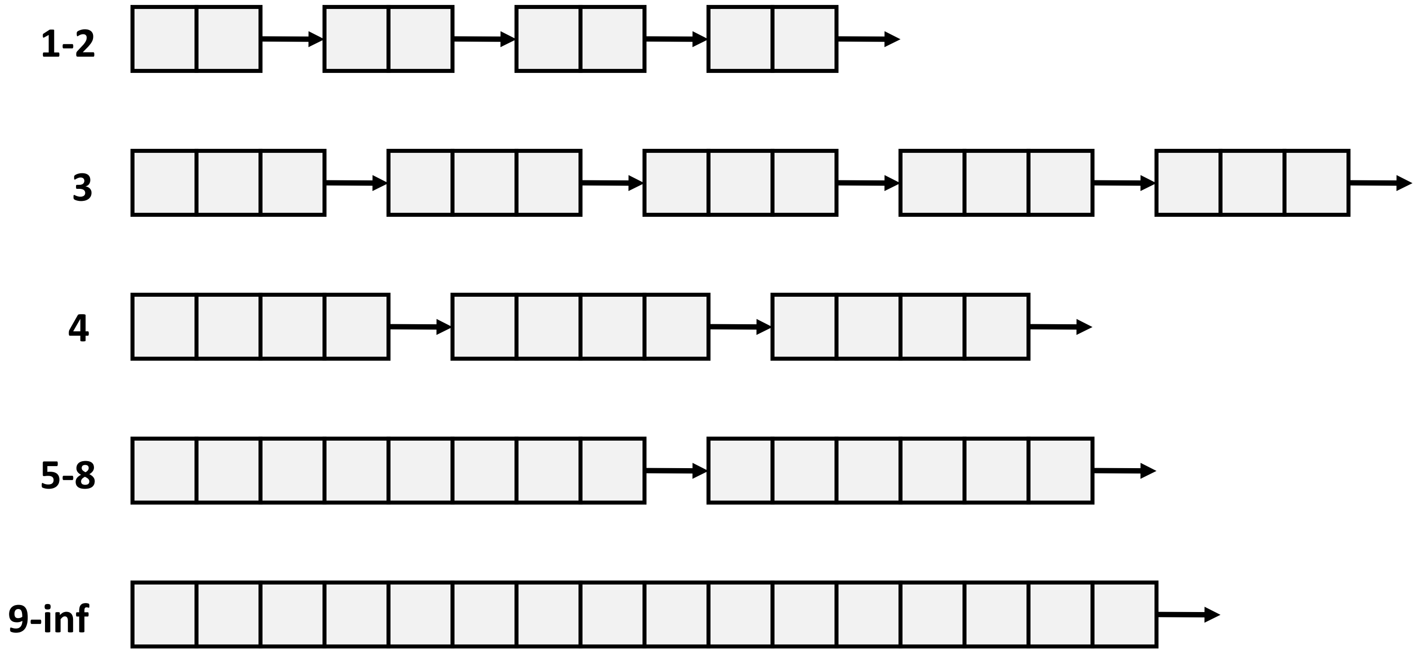
- Maintaining free blocks
  - Implicit lists, with boundary tags (covered last time)
  - Explicit lists, exclude free blocks (faster, but more overhead)
  - Segregated lists (different lists for different sized blocks)
  - Fancy data structures (red-black trees, for example)
- Allocation strategy
  - First-fit, Next-fit, Best-fit
- Coalescing free blocks

Memory-Related Perils and Pitfalls

- Dereferencing bad pointers (Correctness)
- Referencing non-existent variables (Correctness)
- Reading uninitialized memory (Correctness)
- Overreading memory (Security)
- Overwriting memory (Security)
- Referencing freed blocks (Security)
- Freeing blocks multiple times (Security)
- Failing to free blocks (Performance)

Segregated Lists

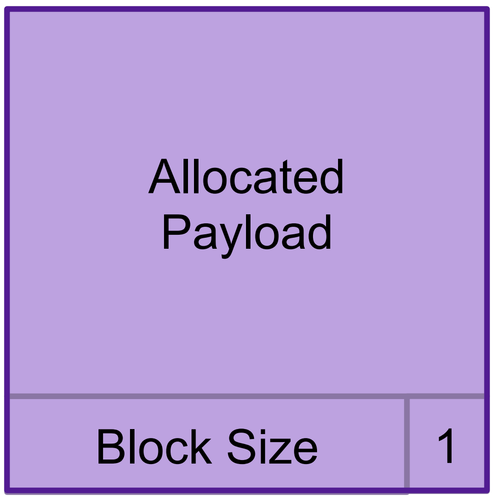
- Each *size class* of blocks has its own free list



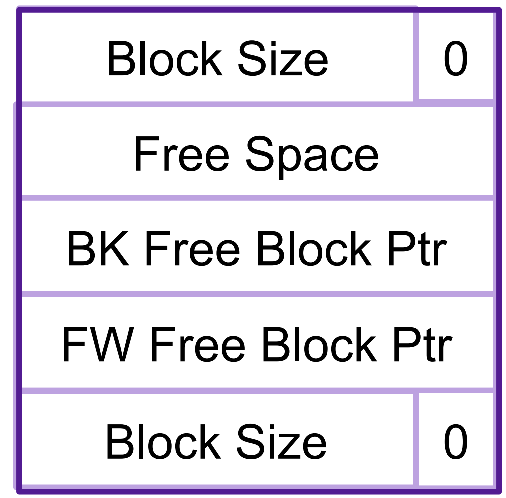
- Often have separate classes for each small size
- For larger sizes: One class for each two-power size

Segregated List Blocks

Allocated Blocks



Free Blocks



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

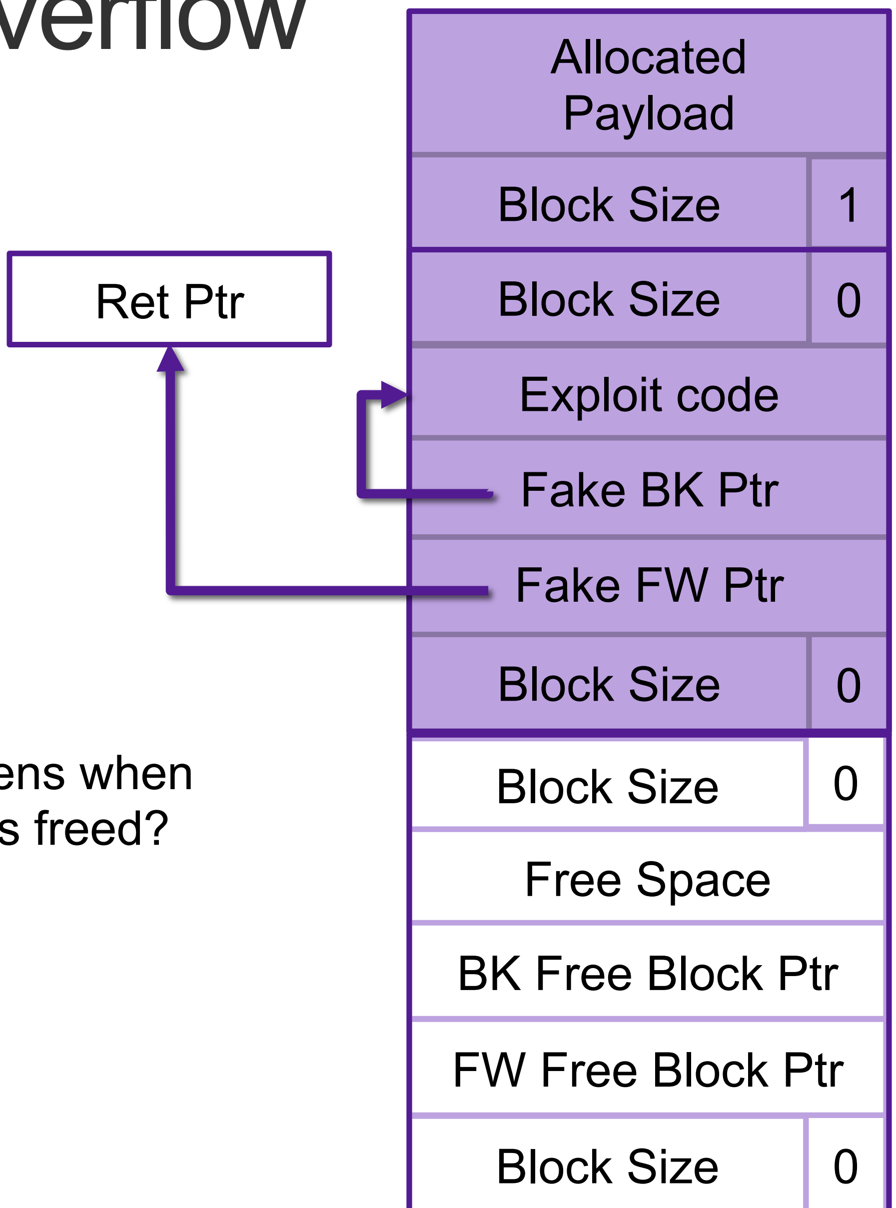
- If no block is found:
  - Request additional heap memory from OS (using **`sbrk()`**)
  - Allocate block of *n* bytes from this new memory
  - Place remainder as a single free block in largest size class.

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.

Buffer Overflow



Then what happens when the top block gets freed?

Tools for Dealing With Memory Bugs

- Debugger: **`gdb`**
  - Good for finding bad pointer dereferences
  - Hard to detect the other memory bugs
- Heap consistency checker (e.g., **`mcheck`**)
  - Usually run silently, printing message only on error
  - Can be used as a probe to find an error
- glibc malloc contains checking code
  - **`setenv MALLOC_CHECK_ 3`**
- Binary translator: **`valgrind`**
  - Powerful debugging and analysis technique
  - Rewrites text section of executable object file
  - Checks each individual reference at runtime
    - Bad pointers, overwrites, refs outside of allocated block

Garbage Collection (Implicit Allocator)

- *Garbage collection:* automatic reclamation of heap-allocated storage—application never has to free

```
void foo() {
   int *p = malloc(128);
   return; /* p block is now garbage */
}
```

- Common in many dynamic languages:
  - Python, Java, Ruby, Perl, ML, Lisp, Mathematica

- Variants ("conservative" garbage collectors) exist for C and C++
  - However, cannot necessarily collect all garbage

Garbage Collection

- How does the memory manager know when memory can be freed?
  - In general we cannot know what is going to be used in the future since it depends on conditionals
  - But we can tell that certain blocks cannot be used if there are no pointers to them

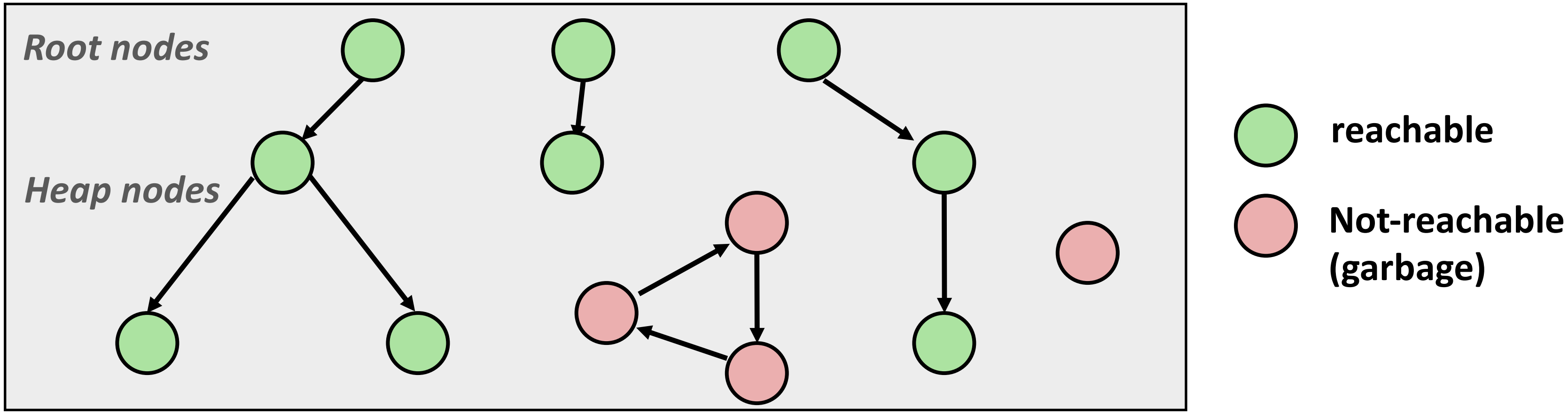
- Must make certain assumptions about pointers
  - Memory manager can distinguish pointers from non-pointers
  - All pointers point to the start of a block
  - Cannot hide pointers
    (e.g., by coercing them to an **`int`**, and then back again)

Classical GC Algorithms

- Mark-and-sweep collection (McCarthy, 1960)
  - Does not move blocks (unless you also "compact")
- Reference counting (Collins, 1960)
  - Does not move blocks (not discussed)
- Copying collection (Minsky, 1963)
  - Moves blocks (not discussed)
- Generational Collectors (Lieberman and Hewitt, 1983)
  - Collection based on lifetimes
    - Most allocations become garbage very soon
    - So focus reclamation work on zones of memory recently allocated

Memory as a Graph

- We view memory as a directed graph
  - Each block is a node in the graph
  - Each pointer is an edge in the graph
  - Locations not in the heap that contain pointers into the heap are called ***root*** nodes (e.g. registers, locations on the stack, global variables)

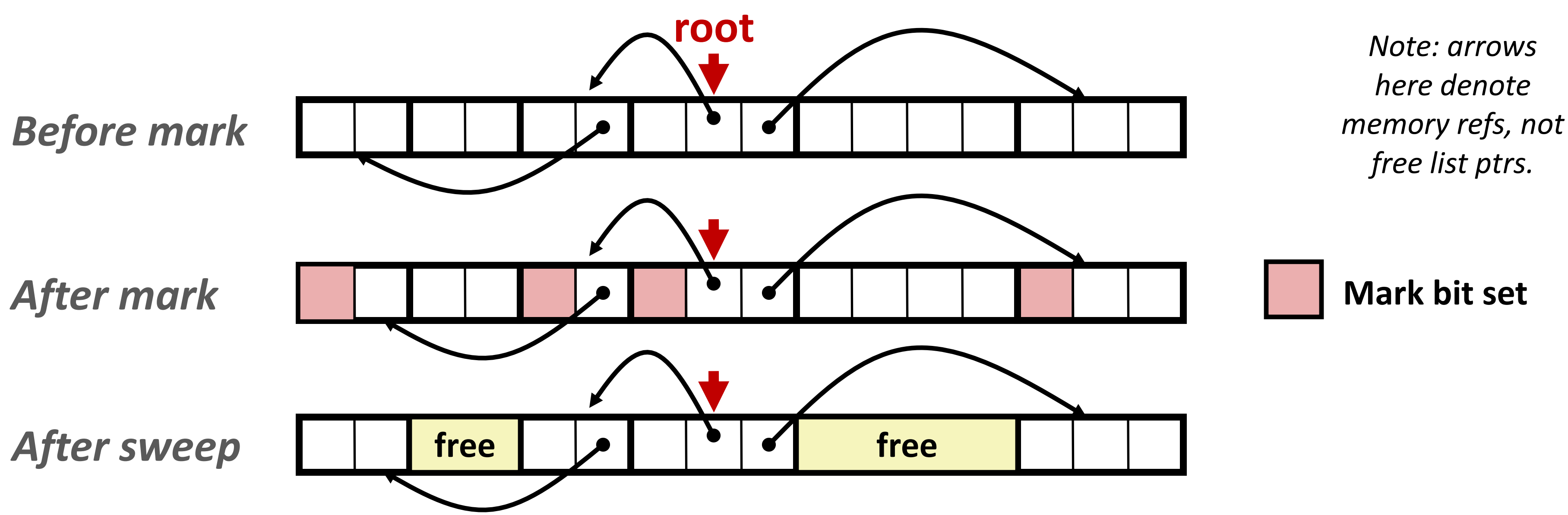


A node (block) is *reachable* if there is a path from any root to that node.

Non-reachable nodes are *garbage* (cannot be needed by the application)

Mark and Sweep Collecting

- Can build on top of malloc/free package
  - Allocate using `malloc` until you "run out of space"
- When out of space:
  - Use extra ***mark bit*** in the head of each block
  - ***Mark:*** Start at roots and set mark bit on each reachable block
  - ***Sweep:*** Scan all blocks and free blocks that are not marked



Mark and Sweep (cont.)

Mark using depth-first traversal of the memory graph

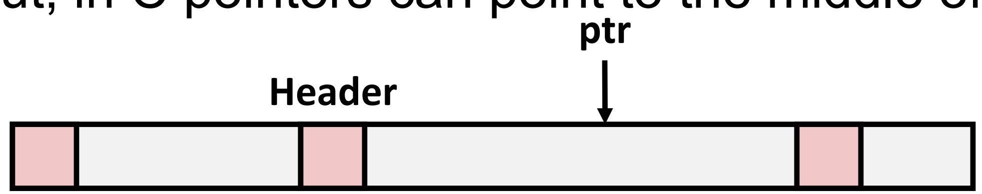
```
ptr mark(ptr p) {
   if (!is_ptr(p)) return;          // do nothing if not pointer
   if (markBitSet(p)) return;       // check if already marked
   setMarkBit(p);                   // set the mark bit
   for (i=0; i < length(p); i++)    // call mark on all words
     mark(p[i]);                    //   in the block
   return;
}
```

Sweep using lengths to find next block

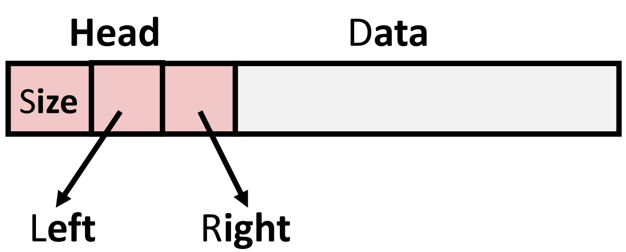
```
ptr sweep(ptr p, ptr end) {
   while (p < end) {
      if markBitSet(p)
         clearMarkBit();
      else if (allocateBitSet(p))
         free(p);
      p += length(p);
}
```

Conservative Mark & Sweep in C

- A "conservative garbage collector" for C programs
  - `is_ptr()` determines if a word is a pointer by checking if it points to an allocated block of memory
  - But, in C pointers can point to the middle of a block



- So how to find the beginning of the block?
  - Can use a balanced binary tree to keep track of all allocated blocks (key is start-of-block)
  - Balanced-tree pointers can be stored in header (use two additional words)



Left: smaller addresses
Right: larger addresses

Introduction to the Malloc Lab

Simulate a dynamic memory allocator by implementing four functions

```
int   mm_init(void);
void *mm_malloc(size_t size);
void  mm_free(void *ptr);
void *mm_realloc(void *ptr, size_t size);
```

Goals are

- Correctness
- Performance: space utilization and throughput
- Programming style