**Memory Allocation**

- Multiple ways to store program data

- Static data: fixed size at compile-time, entire lifetime of the program, read-only

- Stack-allocated data: local/temporary variables, deallocated on return from function

- Dynamic (heap-allocated) data: size only known at runtime, lifetime known only at runtime

**Dynamic Memory Allocation**

- Dynamic memory allocators are used to acquire virtual memory at runtime

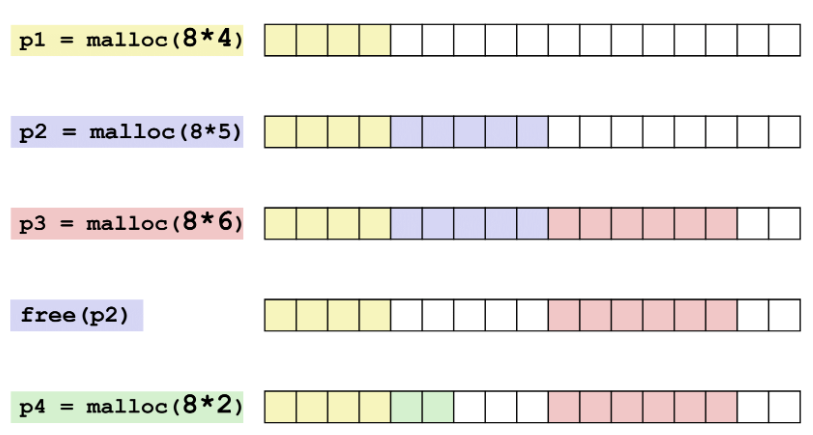
- Explicit allocator: programmer allocates and frees space (malloc and free in C)

- Implicit allocator: programmer only allocates space, doesn’t free (garbage collection in Java)

- Allocator organizes heap as collection of variable-sized blocks which are either allocated or free

- Top of heap is labeled by brk pointer

- Notation: draw memory divided into boxes, each box can hold 64 bits/8 bytes; allocations will be in sizes that are a multiple of boxes



- Allocator requirements: can’t control number or size of allocated blocks, must respond immediately to malloc, must allocate blocks from free memory, must align blocks so they satisfy all alignment requirements, can’t move allocated blocks

- Given some sequence of malloc and free requests, goal is to maximize **throughput** and **peak memory utilization** (often conflicting)

- Throughput: number of completed requests per unit time

- malloc(p) results in a block with a payload of p bytes, *aggregate payload* *Pk* is the sum of currently allocated payloads

- Heap size *Hk* is non-decreasing (can be increased using sbrk)

- Peak memory utilization is defined as *Uk = (max Pi)/Hk*

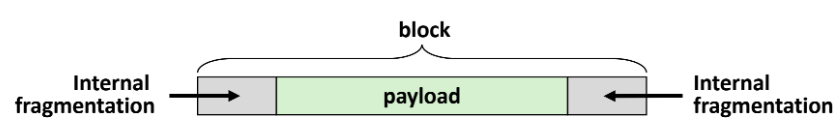
- Poor memory utilization is caused by fragmentation

- Internal fragmentation is wasted space in the heap inside blocks, and external fragmentation is wasted space in the heap between allocated blocks

- Internal fragmentation occurs if payload is smaller than the block in a given block

- Can be caused by padding for alignment purposes, overhead of maintaining heap data structures, explicit policy decisions (first-fit)

- Easy to measure because it only depends on past requests

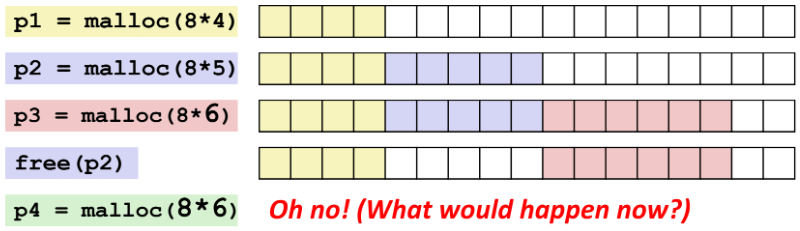


- External fragmentation occurs when allocation/free pattern leaves holes between blocks

- The aggregate payload is non-continuous

- Can cause situations where there is enough aggregate heap memory to satisfy request, but no single free block is large enough

- Don’t know what future requests will be, so it’s difficult-to-impossible to know if past placements will become problematic



- Know how much to free by keeping the length of a block in the box preceding the block

- This box is called the header field or header; requires an extra box for every allocated block

- Keep track of free blocks with one of following:

- Implicit free list using length – links all blocks using math; no actual pointers and must check each block to see if it’s allocated or free

- Explicit free list among only free blocks – using pointers to connect

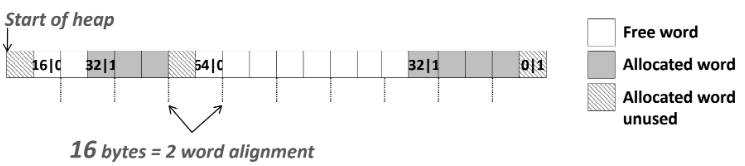
- Segregated free list – different free lists for different size “classes”

- Blocks sorted by size – can use a balanced binary tree (red-black tree) with pointers within each free block, and length used as a key

- Implicit Free Lists

- Use lowest bit as an allocated/free flag

- Special one-word marker (0|1) marks end of list



- Implicit List: finding a free block

- First fit searches list from beginning and chooses first free block that fits

- Can take linear time in total number of blocks; in practice can cause splinters at beginning of list

- Next fit is like first fit, but starts where previous search finished

- Should be faster than first fit, but research suggests fragmentation is worse

- Best fit searches the list and chooses the best free block

- Keeps fragments small, but usually worse throughput because you have to see the entire list

- Can do combinations of algorithms

- Implicit List: allocating in a free block

- Split the block if allocated space is smaller than free space

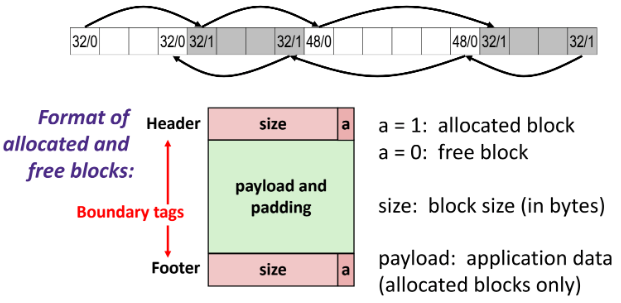
- Implicit List: freeing a block

- Simplest implementation clears allocated flag, but can lead to false fragmentation

- Join (coalesce) with next block if also free

- Can coalesce with previous block by replicating header at end of free blocks – boundary tags

- 4 different cases for coalescing



- Explicit free lists use list of free blocks, rather than implicit list of all blocks

- Need to store next/previous pointers, not just sizes

- Use “payload” space for pointers

- Still need boundary tags for coalescing

- Physically, blocks can be in any order

- Boundary tag needed only for free blocks

- Block allocation is linear time in free blocks instead of all blocks; much faster when most of the memory is full

- More complicated allocate and free; minimum block size is increased due to extra pointers being needed