CS354: Machine Organization and Programming

Lecture 35
Monday the November 23rd 2015

Section 2
Instructor: Leo Arulraj

Class Announcements

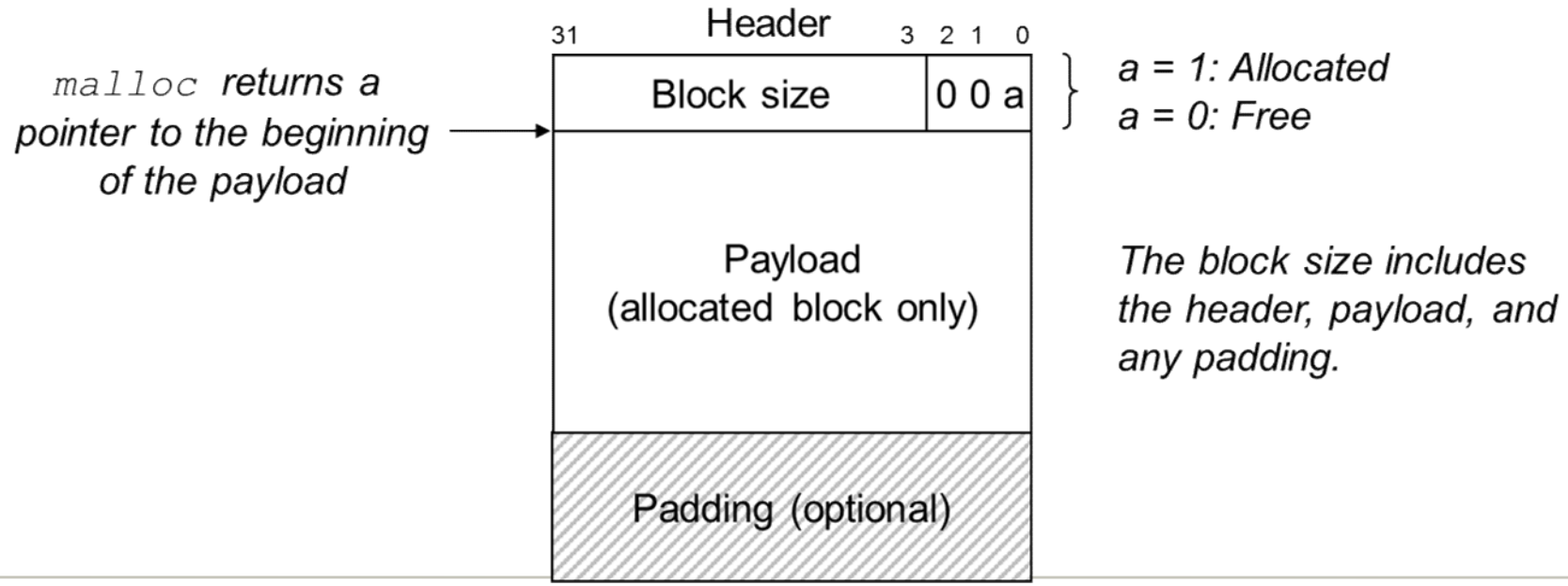
1. Programming Assignment 4 is due day after tomorrow – Wednesday (11/25) by 9 AM.

Lecture Overview

1. Implicit free lists
2. Placing allocated blocks
3. Splitting free blocks
4. Getting additional heap memory
5. Coalescing

Implicit free list node structure

Make the free list node embedded into the free space it is used to manage. Called an implicit free list. Format of a free heap block is as below:



Implicit free list node structure

Header: block size and allocated/free bit

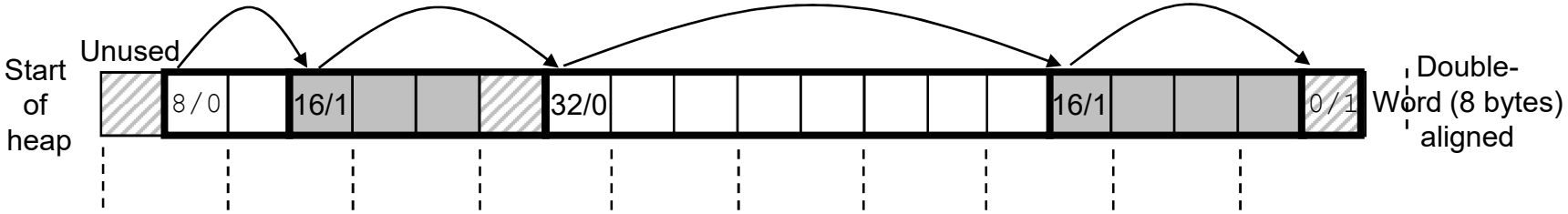
E.g. Header for a block of size 24 (0x18) which is allocated will be 0x19

Payload: this is space where application can store its own data.

Padding (optional): could be for alignment or due to an allocator strategy to avoid external fragmentation.

Implicit free list example

Shaded blocks are allocated ones and unshaded ones are free ones.



A special end node with size zero and its allocated bit set is used to denote the end of the free list.

Advantage of implicit free list is their simplicity.

Disadvantage is the cost of traversing the entire list during allocation.

Implicit free list

System's alignment requirement and allocator's choice of block format impose a minimum block size on the allocator.

So, even if application requested just one byte of memory, allocator would have to create a two word block.

Placing Allocated Blocks

Manner in which the free list is searched for a free block during an allocation request is determined by the *placement policy*.

First Fit

Next Fit

Best Fit

First Fit

Search the free list from the beginning and choose the first free block that fits.

Advantage: Retains large free blocks at the end of the free list.

Disadvantage: Leaves "splinters" of small free blocks toward the beginning of the free list increasing the search time for larger blocks.

Next Fit

Similar to First Fit but instead start the search at the place where the previous search left off.

Advantage: runs significantly faster than first fit especially when the front of the free list becomes littered with small splinters.

Disadvantage: Some studies suggest it might have worse memory utilization compared to first fit.

Best Fit

Examines every free block and chooses the free block with the smallest size that fits.

Advantage: Has the best memory utilization

Disadvantage: Requires exhaustive search of the heap

Splitting a free block

After locating a free block that fits the requested memory, how much of the free block to allocate?

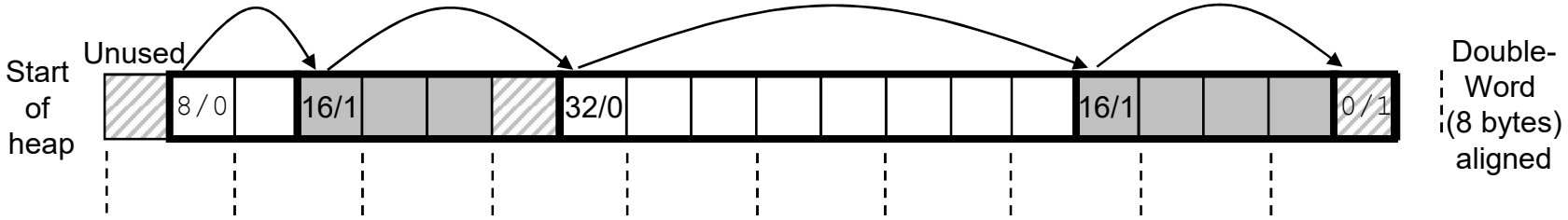
Options:

1) Use the entire free block: Simple and fast but leads to internal fragmentation.
2) Split the free block into two parts: First part is the allocated part and the second part will be a new free block.

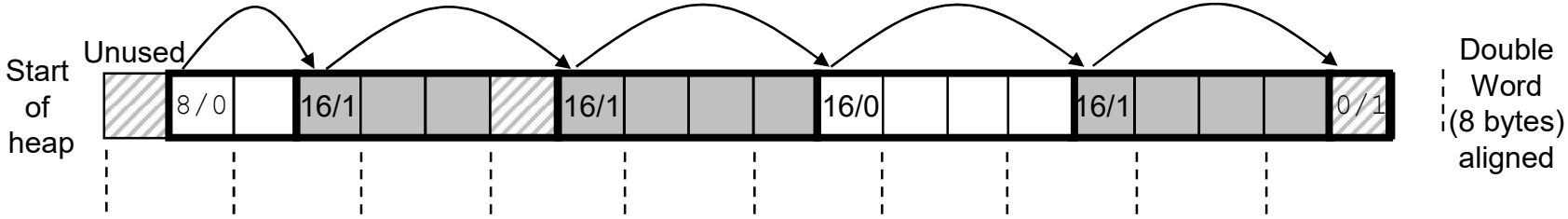
Implicit free list before and after splitting

Application calls: int *q = alloc(3*sizeof(int))

Before splitting:

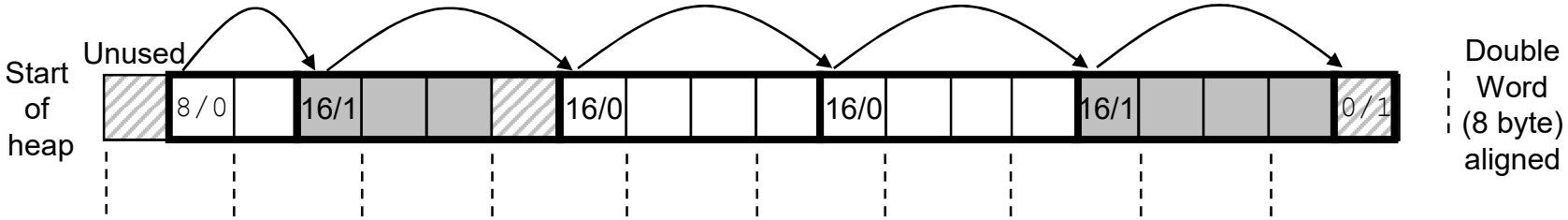


After splitting:



Coalescing Adjacent Free Blocks

Suppose application now calls: free(q)



What if application calls:

int *r = malloc(4*sizeof(int)) ?

Fails because there is no single free block that can hold more than 3 integers.

Coalescing Adjacent Free Blocks

False fragmentation: Lot of free space is available but is chopped up into small, unusable free blocks.

Solution: coalesce or merge adjacent free blocks.

Immediate Coalescing: Merge any adjacent free blocks each time a block is freed. Fast but can lead to a form of thrashing where a block is repeatedly split and then coalesced.

Deferred Coalescing: Waiting to coalesce free blocks at some later time.

Getting additional Heap Memory

What if there is no more free space in any free block?

Allocator calls sbrk() system call to allocate more heap space and then it adds a new free node encompassing this newly allocated heap space into the free list.

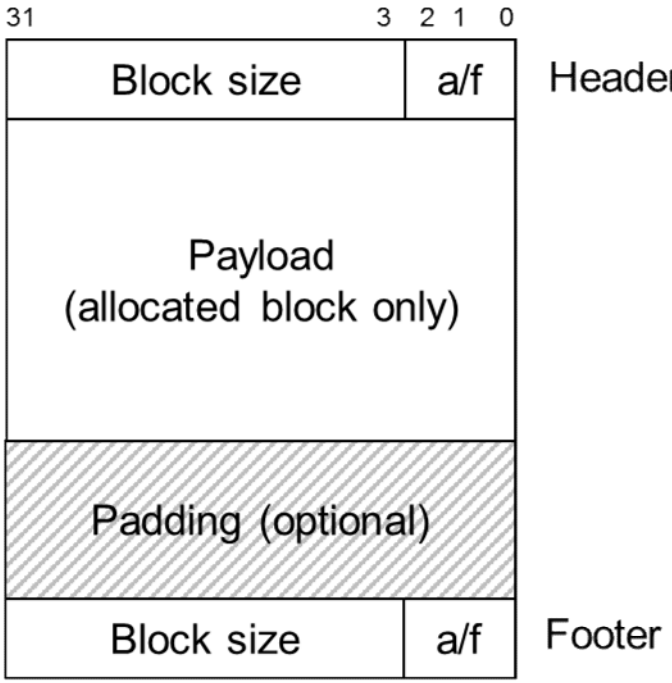
How to Coalesce with Previous Block?

1. Coalescing with the next free block is straightforward because of header of current block points to the header of the next block.

2. With current format of implicit free list, coalescing with a previous free block can only be done by a full free list scan until we reach the free block before the block just being freed.

A modified implicit free list

Adding Boundary tags to help coalesce with previous free block in constant time.



a = 001: Allocated
f = 000: Free

Idea is: Add a footer (aka boundary tag) which is a replica of the header.

A modified implicit free list

With boundary tags, allocator just needs to check the footer of the previous block to find its size and allocation status.

Disadvantage: Requiring each block to contain both a header and a footer can be a memory overhead if application deals with small memory allocations.

A modified implicit free list

A Solution to the space issue of boundary tags:

Size info store in the footer is looked up only for free blocks.

If alloc/free status in the previous block's footer is instead stored in the excess bits of the current block's header, then allocated blocks would not need any footer at all.