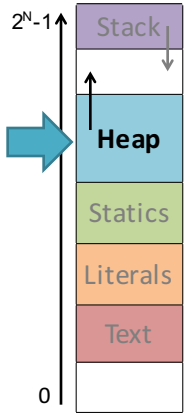
Dynamic Memory Allocation in the Heap

(malloc and free)

The Heap

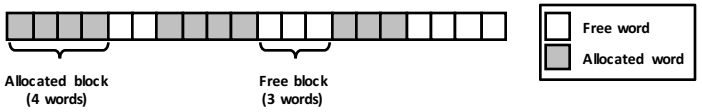


Perm	Contents	Managed by	Initialized
RW	Procedure context	Compiler	Run-time
RW	**Dynamic data structures**	Programmer, **malloc/free,** new/GC	**Run-time**
RW	Global variables/ static data structures	Compiler/ Assembler/Linker	Startup
R	String literals	Compiler/ Assembler/Linker	Startup
X	Instructions	Compiler/ Assembler/Linker	Startup

Allocator Basics

Pages too coarse-grained for allocating individual objects.

Instead: **flexible-sized, word-aligned blocks.**



pointer to newly allocated block of at least that size

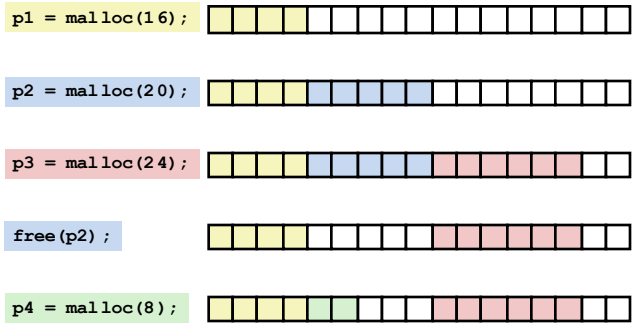
number of contiguous bytes required

```
void*  malloc(size_t  size);
```

pointer to allocated block to free

```
void  free(void*  ptr);
```


Example (32-bit words)



Beware **fragmentation**: unused memory that cannot be allocated.

Allocator Goals: malloc/free

1. **Programmer does not decide locations of distinct objects.**
   Just what size, when needed, and when no longer needed

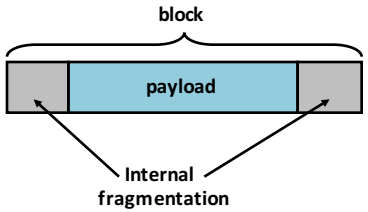
2. **Fast allocation.** mallocs/second

3. **High memory utilization.**
   
   Most of heap contains necessary program data.
   Little wasted space.

   Enemy: **fragmentation**.

Internal Fragmentation

payload smaller than block



Causes

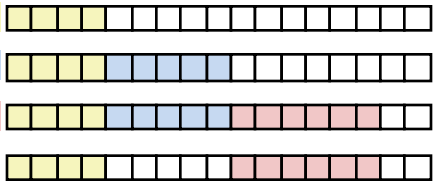
block metadata
padding for alignment
explicit policy decisions

6

External Fragmentation (32-bit)

Total free space large enough,
but no single free block large enough

```
p1 = malloc(16);
p2 = malloc(20);
p3 = malloc(24);
free(p2);
p4 = malloc(24);
```



Depends on the pattern of future requests.

7

Implementation Issues

Determine **how much to free** given just a pointer.

Keep track of free blocks.

Pick a block to allocate.

Choose what do with **extra space when allocating** a structure that is **smaller than the free block used**.

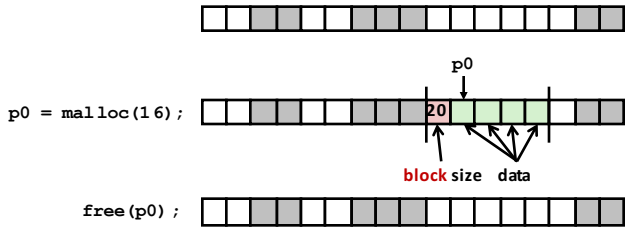
Make a freed block available for future reuse.

8

Knowing How Much to Free

Keep length of block in ***header*** word preceding block

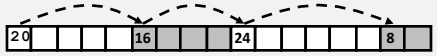
Takes extra space!



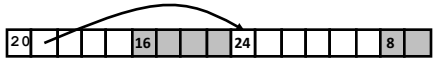
9

Keeping Track of Free Blocks

Method 1: ***Implicit list*** of all blocks using length



Method 2: ***Explicit list*** of free blocks using pointers



Method 3: *Seglist*

Different free lists for different size brackets

More methods that we will skip…

10

Implicit Free Lists

For each block we need: size, is-allocated?

Could store this information in two words: wasteful!

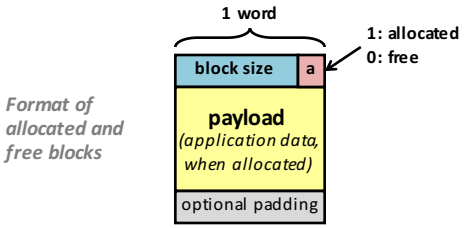
Standard trick

Steal low-order bit for allocated/free flag.

Mask off flag when using size.

8-byte aligned sizes have 3 zeroes in low-order bits

00000**000**
00001**000**
00010**000**
00011**000**
…

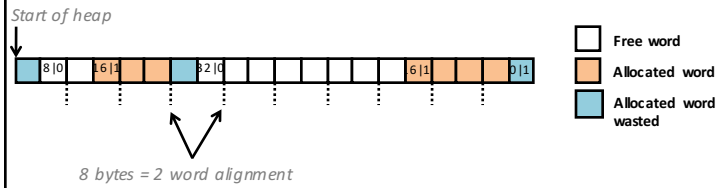


Format of allocated and free blocks

11

Implicit Free List Example (32-bit)

Sequence of blocks in heap (size|allocated): 8|0, 16|1, 32|0, 16|1



8-byte alignment

May require initial unused word

Causes some internal fragmentation

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

zero size is distinguishable from all real sizes

12

Implicit List: **Finding a Free Block**

First fit:

Search list from beginning, choose ***first*** free block that fits

Next fit:

Do first-fit starting where previous search finished

Best fit:

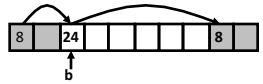
Search the list, choose the ***best*** free block: fits, with fewest bytes left over

13

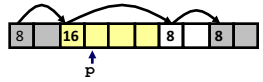
Implicit List: **Allocating in Free Block**

Allocating in a free block: *splitting*

Allocated space may be smaller than free space. Use it all? Split it up?




```
p = malloc(12);
```



14

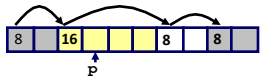
Implicit List: **Freeing a Block**

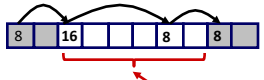
Simplest implementation:

Clear "allocated" flag.

Leads to "false fragmentation"

```
free(p);
```






```
malloc(20);
```

Enough contiguous space, but not one block!

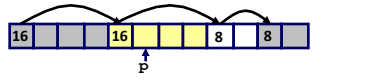
15

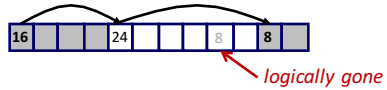
Coalescing

Join *(coalesce)* with adjacent free blocks.

Coalesce with following (free) block:

```
free(p)
```





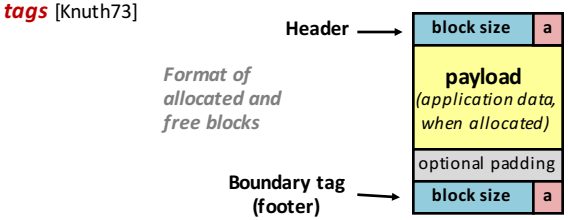
logically gone

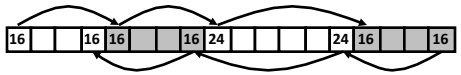
Coalesce with the *previous* block?

16

Bidirectional Coalescing

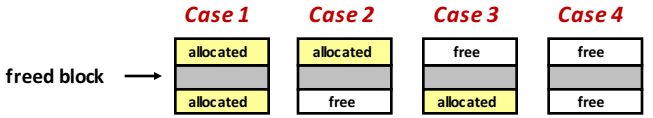
Boundary tags [Knuth73]





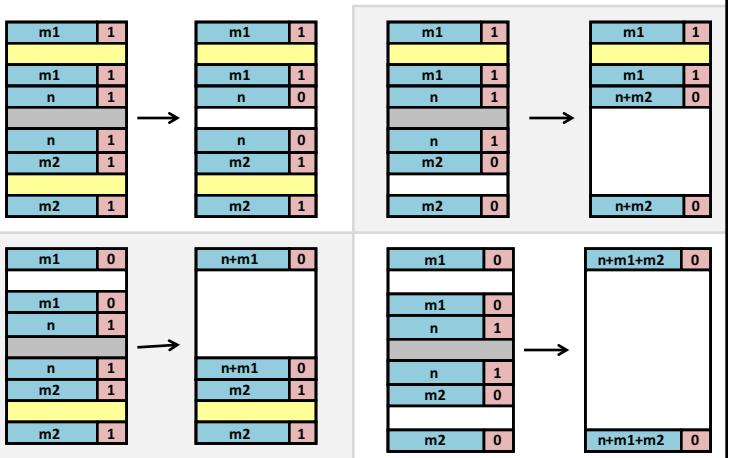
17

Constant-Time Coalescing: 4 cases



18

Constant-Time Coalescing: 4 cases



19

Summary: Implicit Free Lists

Implementation: very simple

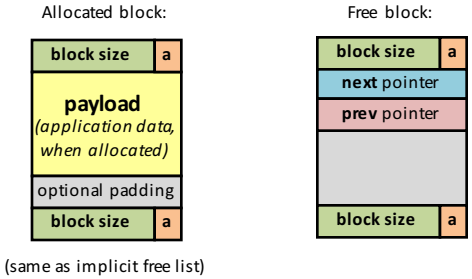
Allocate: O(blocks in heap)
Free: O(1)
Memory utilization: depends on placement policy

Not widely used in practice
some special purpose applications

Splitting, boundary tags, coalescing are general to ***all*** allocators.

20

Explicit Free Lists



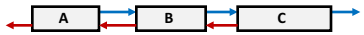
(same as implicit free list)

Maintain list of *free* blocks rather than implicit list of *all* blocks.

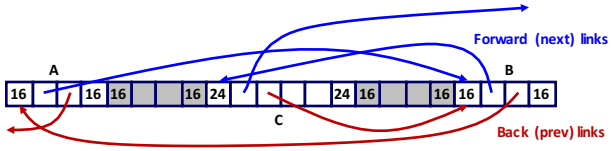
21

Explicit Free Lists

Logically (doubly-linked lists):

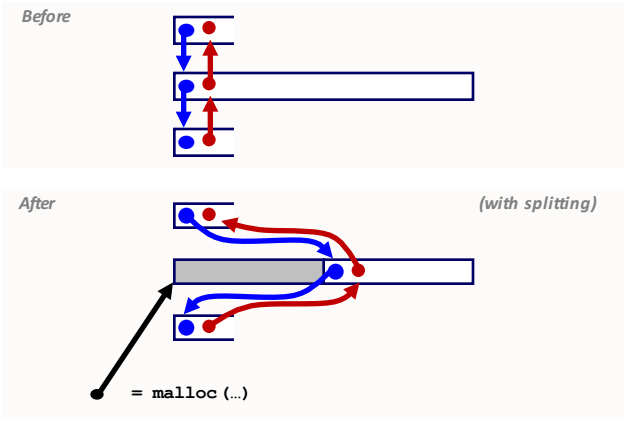


Physically: blocks can be in any order



23

Allocating From Explicit Free Lists



24

Freeing with Explicit Free Lists

Insertion policy: Where in the free list do you put a freed block?

LIFO (last-in-first-out) policy

- ***Pro:*** simple and constant time
- ***Con:*** studies suggest fragmentation is worse than address ordered

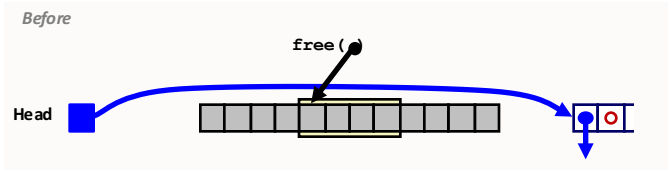
Address-ordered policy

- ***Con:*** linear-time search to insert freed blocks
- ***Pro:*** studies suggest fragmentation is lower than LIFO

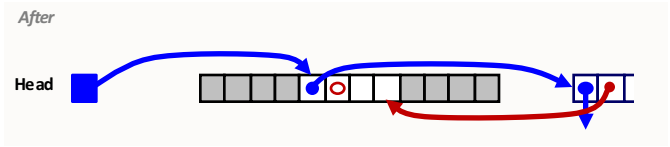
Cache effects?

25

Freeing With a LIFO Policy (Case 1)

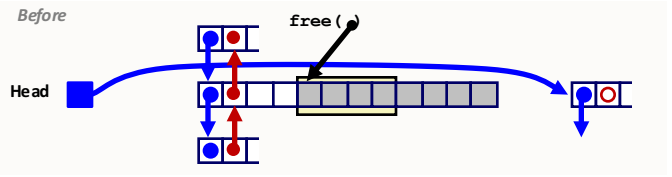


Insert the freed block at head of free list.

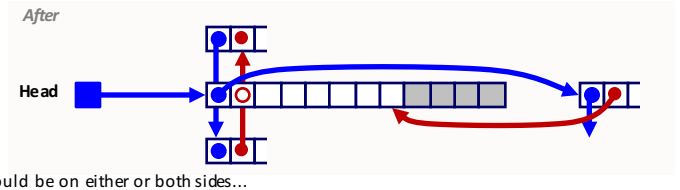


26

What if the freed block is adjacent to another?



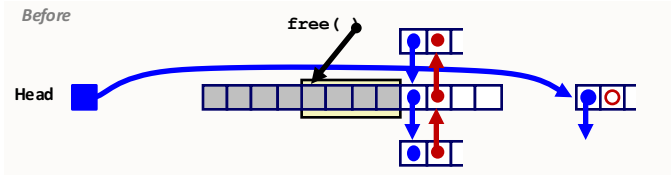
Splice out predecessor block, coalesce both memory blocks, and insert the new block at the head of the free list.



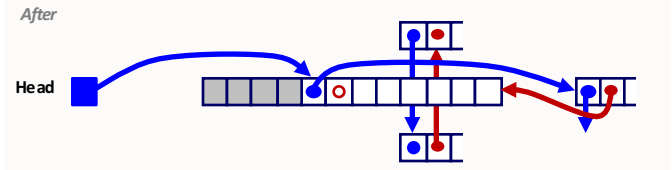
Could be on either or both sides...

27

Freeing With a LIFO Policy (Case 3)

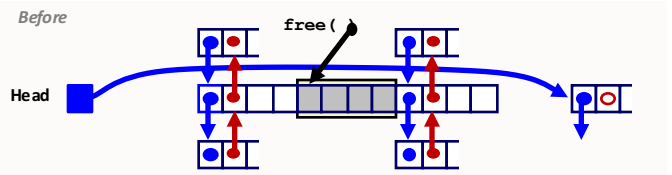


Splice out successor block, coalesce both memory blocks and insert the new block at the head of the free list.

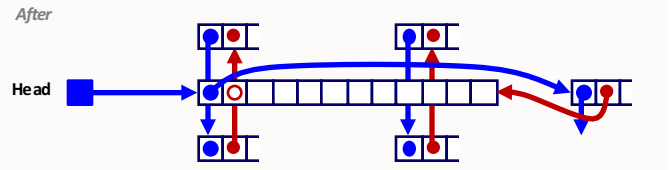


28

Freeing With a LIFO Policy (Case 4)

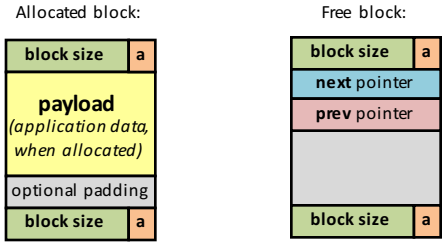


Splice out predecessor and successor blocks, coalesce all 3 memory blocks and insert the new block at the head of the list.



29

Do we always need the boundary tag?



34

Explicit Free Lists: Summary

Implementation: fairly simple

Allocate: O(*free* blocks) vs. O(*all* blocks)
Free: O(1)
Memory utilization:
depends on placement policy
larger minimum block size (next/prev)

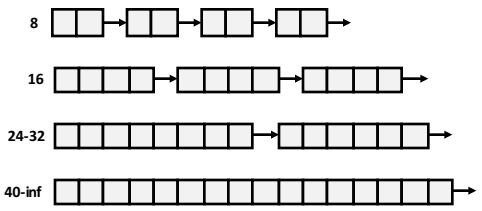
Used widely in practice.

Splitting, boundary tags, coalescing are general to ***all*** allocators.

36

Seglist Allocators

Each *size bracket* has its own free list



Faster best-fit allocation…

38

Summary of Key Allocator Policies

All policies offer trade-offs in fragmentation and throughput.

Placement policy:
First-fit, next-fit, best-fit, etc.
Seglists approximate best-fit in low time

Splitting policy
Always? Sometimes? Size bound?

Coalescing policy:
Immediate vs. deferred

41