CSO-Recitation 11

CSCI-UA 0201-007

R11: Assessment 09 & Dynamic memory allocation

Today's Topics

- Assessment 09
- Dynamic memory allocation
  - implement your malloc & free

Assessment 09

Q1 Linker

Which of the following statements are true about the C linker?

Linker takes .o as input

A. The linker takes as input ~~C source code~~ and outputs a binary executable file.
B. The linker has knowledge of the types of all variables declared or accessed in an object file.
C. The linker performs symbol resolution and relocation to replace each symbol reference to the symbol's address.
D. If x is a non-static global variable defined in source file obj.c, then x appears in the symbol table of the corresponding object file obj.o
E. If x is a non-static local variable defined in source file obj.c, then x appears in the symbol table of the corresponding object file obj.o
F. If x is a function defined in source file obj.c, then x appears in the symbol table of the corresponding object file obj.o

What symbol table contains

Symbol table contains:

1. Global symbols

    Non-static global variables & functions

2. Local symbols

    Static functions & global variables

3. External symbols

    External functions & variables defined in other ".o" files

```c
int a = 0;
int foo1()
{
    int b;
    ...
}
static int c;
static int foo2();
extern int d;
extern int foo3();
```

Does not contain local variables

Q1 Linker

Which of the following statements are true about the C linker?

A. The linker takes as input C source code and outputs a binary executable file.

B. The linker has knowledge of the types of all variables declared or accessed in an object file.

It has no knowledge of local variables

C. The linker performs symbol resolution and relocation to replace each symbol reference to the symbol's address.

D. If x is a non-static global variable defined in source file obj.c, then x appears in the symbol table of the corresponding object file obj.o

E. If x is a non-static local variable defined in source file obj.c, then x appears in the symbol table of the corresponding object file obj.o

F. If x is a function defined in source file obj.c, then x appears in the symbol table of the corresponding object file obj.o

Non-static globals & function definition vs declar

Abnormally, C supports using a function without definition/declaration (Nasty! Banned by C++)

- Normally need to define / declare before use
  - Define in one file; declare in other files
  - All resolve to the only definition

Defining multiple times => name collision

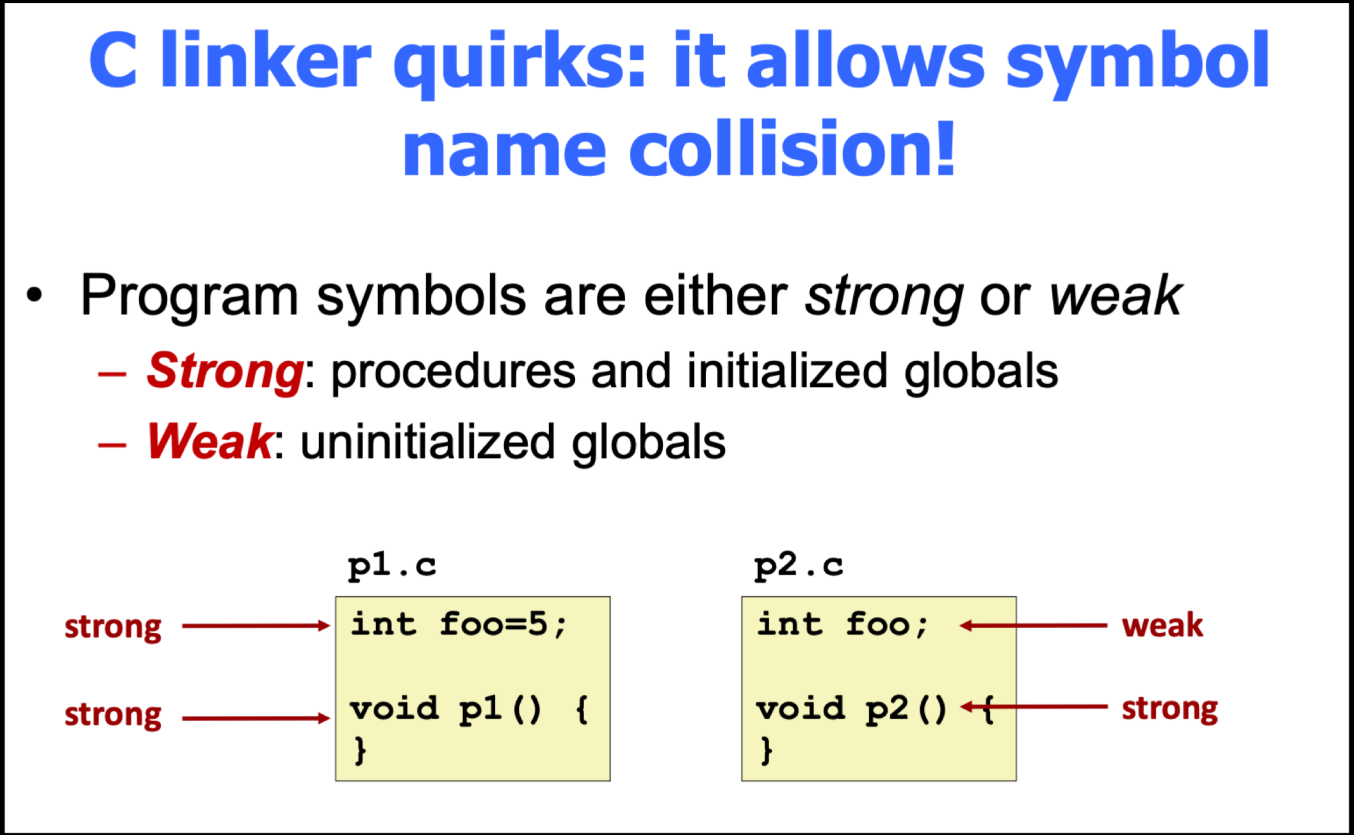
- Global variables:

extern int a;	declaration
int a;	definition
int a=xxx;	definition

- Functions:

extern int func();	declaration
int func();	declaration
int func() { … }	definition

C’s name collision resolution rule



# strong symbols	name collision resolution
>=2	link error
1	resolve to strong symbol
0	resolve to arbitrary one

Q2 Linking vs. compile error

```c
// foo.c
int x = 0;
void multiply_by_two() {
  x = x * 2;
}
```

```c
// main.c
#include <stdio.h>

int main()
{
  x = 1;
  multiply_by_two();
  printf("x=%d\n", x);
}
```

x not declared before use

To generate the executable file, the following 3 steps are performed:

step-1: gcc -c foo.c

step-2: gcc -c main.c

step-3: gcc foo.o main.o

Which of the following statements are true:

A. There is a compilation error when performing step-1
B. There is a compilation error when performing step-2
C. There is a linking error when performing step-3

Select C or not, we give full mark

D. All 3 steps can be performed successfully. When running ./a.out, the output is 2
E. All 3 steps can be performed successfully. When running ./a.out, the output is 1

Q3 Linking vs. compile error

```c
// foo.c
int x = 0;
void multiply_by_two() {
  x = x * 2;
}
```

```c
// main.c Q3
#include <stdio.h>
extern int x;

int main()
{
  x = 1;
  multiply_by_two();
  printf("x=%d\n", x);
}
```

resolved to the only definition of x in foo.c

This question is the same as Q2, except the main.c file has been changed to

Which of the following statements are true:

A. There is a compilation error when performing step-1

B. There is a compilation error when performing step-2

C. There is a linking error when performing step-3

D. All 3 steps can be performed successfully. When running ./a.out, the output is 2

E. All 3 steps can be performed successfully. When running ./a.out, the output is 1

Q4 Linking vs. compile error

```c
// foo.c
int x = 0;
void multiply_by_two() {
  x = x * 2;
```

```c
// main.c Q4
#include <stdio.h>
int x = 0;

int main()
{
  x = 1;
  multiply_by_two();
  printf("x=%d\n", x);
}
```

2 non-static global definitions =>
Name collision!

This question is the same as Q2, except the main.c file has been changed to

Which of the following statements are true:

A. There is a compilation error when performing step-1

B. There is a compilation error when performing step-2

C. There is a linking error when performing step-3

D. All 3 steps can be performed successfully. When running ./a.out, the output is 2

E. All 3 steps can be performed successfully. When running ./a.out, the output is 1

Q4 Linking vs. compile error

```c
// foo.c
int x = 0;
void multiply_by_two() {
  x = x * 2;
```

```c
// main.c Q4
#include <stdio.h>
int x = 0;

int main()
{
  x = 1;
  multiply_by_two();
  printf("x=%d\n", x);
}
```

2 strong symbols
=> link error

This question is the same as Q2, except the main.c file has been changed to

Which of the following statements are true:

A. There is a compilation error when performing step-1

B. There is a compilation error when performing step-2

C. There is a linking error when performing step-3

D. All 3 steps can be performed successfully. When running ./a.out, the output is 2

E. All 3 steps can be performed successfully. When running ./a.out, the output is 1

Q5 Linking vs. compile error

```c
// foo.c
int x = 0;
void multiply_by_two() {
  x = x * 2;
}
```

```c
// main.c Q5
#include <stdio.h>
static int x;

int main()
{
  x = 1;
  multiply_by_two();
  printf("x=%d\n", x);
}
```

static => 2 different "x" => no name collision

This question is the same as Q2, except the main.c file has been changed to

Which of the following statements are true:

A. There is a compilation error when performing step-1

B. There is a compilation error when performing step-2

C. There is a linking error when performing step-3

D. All 3 steps can be performed successfully. When running ./a.out, the output is 2

E. All 3 steps can be performed successfully. When running ./a.out, the output is 1

Q6 Linking vs. compile error

```c
// foo.c
int x = 0;
void multiply_by_two() {
  x = x * 2;
```

```c
// main.c Q6
#include <stdio.h>
int x;

int main()
{
  x = 1;
  multiply_by_two();
  printf("x=%d\n", x);
}
```

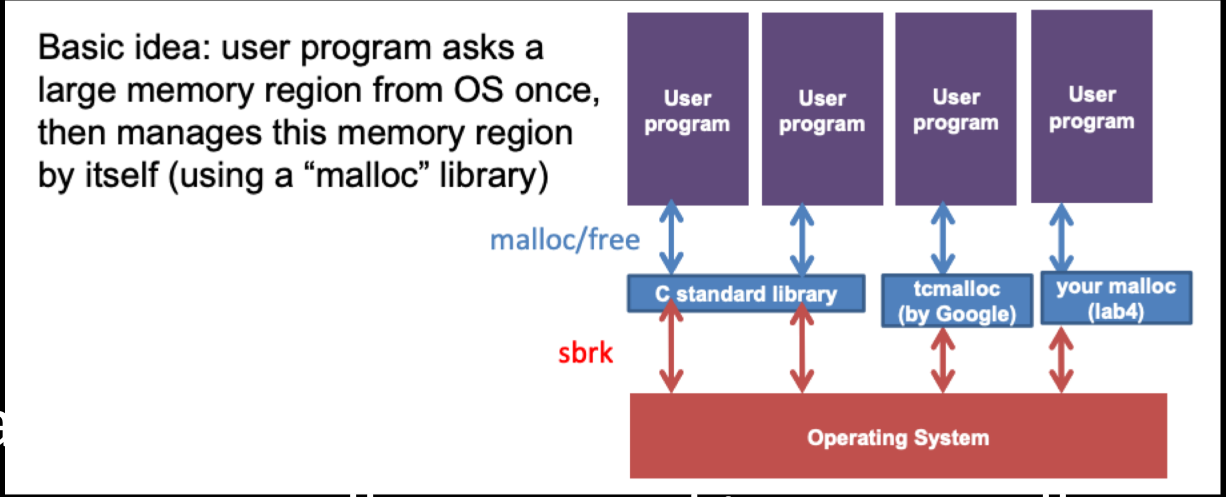
Uninitialized => Weak symbol
=> resolved to x in foo.c

This question is the same as Q2, except the main.c file has been changed to

Which of the following statements are true:

A. There is a compilation error when performing step-1
B. There is a compilation error when performing step-2
C. There is a linking error when performing step-3
D. All 3 steps can be performed successfully. When running ./a.out, the output is 2
E. All 3 steps can be performed successfully. When running ./a.out, the output is 1

Q7 Basic malloc



Which of the following statements a

A. Every call to malloc results in the memory allocator making a syscall (e.g. sbrk) to request memory from OS.
B. There is a special x86 instruction to handle malloc.
C. One must use the malloc/free functions provided by C stdlib and cannot not use any other malloc library.
D. malloc allocates space on the heap memory region of the running program.
E. malloc allocates space on the stack memory region of the running program.

Assumptions on application behavior:

- Use APIs correctly
  - Argument of free must be the return value of a previous malloc
  - No double free
- Use APIs freely
  - Can issue an arbitrary sequence of malloc/free

Restrictions on the allocator:

- Once allocated, space cannot be moved around

Q8 Malloc design

Which of the following are true w.r.t ign?

A. The design can move previously allocated space to a different location to reduce fragmentation.

B. The design can assume that users strictly alternate calls to malloc and free.

C. The design can assume that the argument of free is the return value of some previous malloc calls.

D. The design can invoke arbitrary <stdlib.h> functions including standard library's malloc/free library calls.

E. None of the above.

Q9 Alignment

In order to ensure that the payload address is 16-byte aligned, we enforce the rule that the payload size allocated must be a multiple of 16 bytes.

Given a requested allocation of sz bytes in size (aka malloc(unsigned long sz)), which of the following C statement can round sz to the nearest multiples of 16?

- Result >= input sz
- Result % 16 == 0

Return result nearest to input sz

A. sz = sz / 16; result < input sz

B. sz = sz + sz % 16; sz=1, result=2 not multiple of 16

C. sz = sz + (sz % 16); same

D. sz = sz + 16 - (sz % 16); sz=16, result=32, but 16 is nearer

E. sz = ((size + 0xf) & ~0xf);

Q9 Alignment

In order to ensure that the payload address is 16-byte aligned, we enforce the rule that the payload size allocated must be a multiple of 16 bytes.

Given a requested allocation of sz bytes in size (aka malloc(unsigned long sz)), which of the following C statement can round sz to the nearest multiples of 16?

R = size + 15 – (size + 15) % 16

A. sz = sz / 16;

B. sz = sz + sz % 16;

C. sz = sz + (sz % 16);

D. sz = sz + 16 - (sz % 16);

E. sz = ((size + 0xf) & ~0xf);

size + 15

Clear the lowest 4 bits:
- (size + 15) % 16

Q9 Alignment

In order to ensure that the payload address is 16-byte aligned, we enforce the rule that the payload size allocated must be a multiple of 16 bytes.

Given a requested allocation of sz bytes in size (aka malloc(unsigned long sz)), which of the following C statement can round sz to the nearest multiples of 16?

A. sz = sz / 16;

B. sz = sz + sz % 16;

C. sz = sz + (sz % 16);

D. sz = sz + 16 - (sz % 16);

E. sz = ((size + 0xf) & ~0xf);

size + 15

Clear the lowest 4 bits:
- (size + 15) % 16

R = size + 15 – (size + 15) % 16

1. size % 16 == 0: R = size + 15 – 15 = size; R % 16 == 0
2. size % 16 != 0:

Q9 Alignment

In order to ensure that the payload address is 16-byte aligned, we enforce the rule that the payload size allocated must be a multiple of 16 bytes.

Given a requested allocation of sz bytes in size (aka malloc(unsigned long sz)), which of the following C statement can round sz to the nearest multiples of 16?

A. sz = sz / 16;

B. sz = sz + sz % 16;

C. sz = sz + (sz % 16);

D. sz = sz + 16 - (sz % 16);

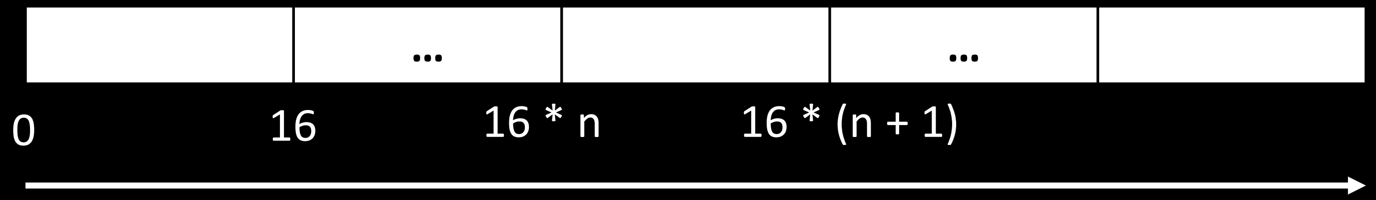
E. sz = ((size + 0xf) & ~0xf);

size + 15

Clear the lowest 4 bits:
- (size + 15) % 16

R = size + 15 – (size + 15) % 16

1. size % 16 == 0: R = size + 15 – 15 = size; R % 16 == 0
2. size % 16 != 0:



Q9 Alignment

In order to ensure that the payload address is 16-byte aligned, we enforce the rule that the payload size allocated must be a multiple of 16 bytes.

Given a requested allocation of sz bytes in size (aka malloc(unsigned long sz)), which of the following C statement can round sz to the nearest multiples of 16?

A. sz = sz / 16;

B. sz = sz + sz % 16;

C. sz = sz + (sz % 16);

D. sz = sz + 16 - (sz % 16);

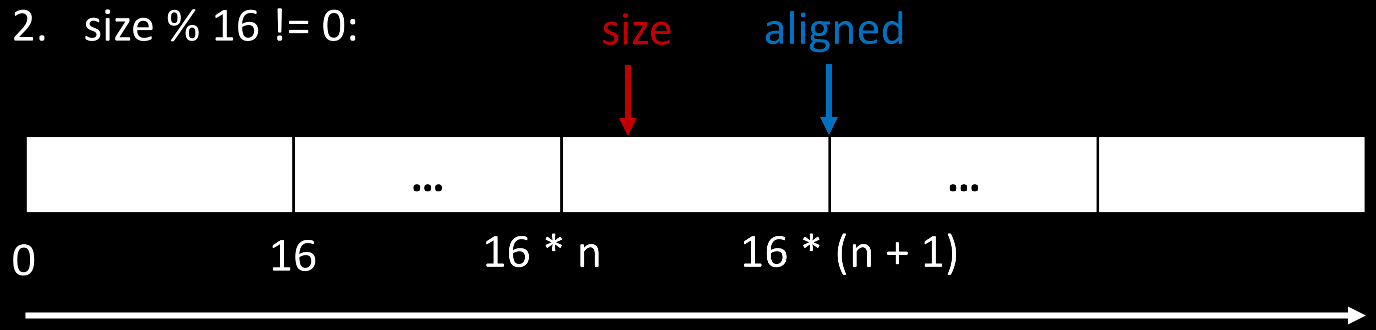
E. sz = ((size + 0xf) & ~0xf);

size + 15

Clear the lowest 4 bits:
- (size + 15) % 16

R = size + 15 – (size + 15) % 16

1. size % 16 == 0: R = size + 15 – 15 = size; R % 16 == 0
2. size % 16 != 0:



Q9 Alignment

In order to ensure that the payload address is 16-byte aligned, we enforce the rule that the payload size allocated must be a multiple of 16 bytes.

Given a requested allocation of sz bytes in size (aka malloc(unsigned long sz)), which of the following C statement can round sz to the nearest multiples of 16?

A. sz = sz / 16;

B. sz = sz + sz % 16;

C. sz = sz + (sz % 16);

D. sz = sz + 16 - (sz % 16);

E. sz = ((size + 0xf) & ~0xf);

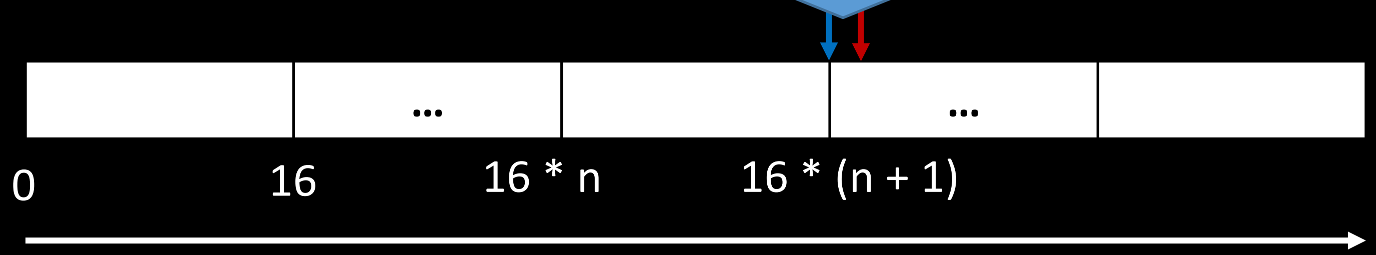
size + 15

Clear the lowest 4 bits:
- (size + 15) % 16

R = size + 15 – (size + 15) % 16

1. size % 16 == 0: R = size + 15 – 1
2. size % 16 != 0:

gap = (size + 15) % 16



Q9 Alignment

In order to ensure that the payload address is 16-byte aligned, we enforce the rule that the payload size allocated must be a multiple of 16 bytes.

Given a requested allocation of sz bytes in size (aka malloc(unsigned long sz)), which of the following C statement can round sz to the nearest multiples of 16?

A. sz = sz / 16;

B. sz = sz + sz % 16;

C. sz = sz + (sz % 16);

D. sz = sz + 16 - (sz % 16);

E. sz = ((size + 0xf) & ~0xf);

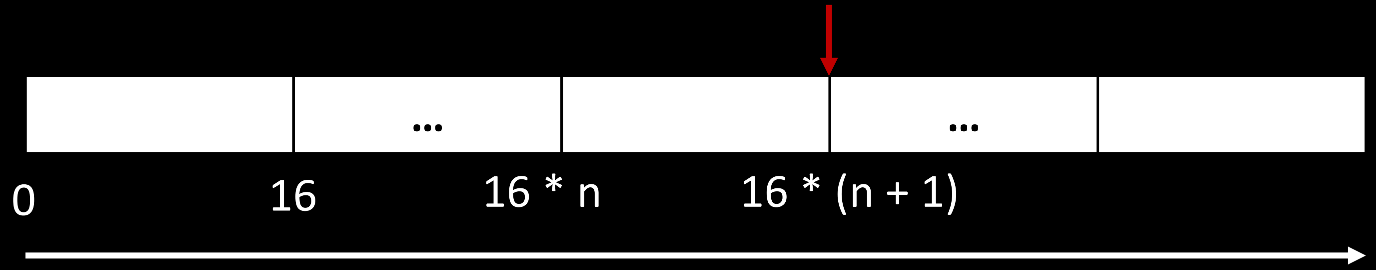
size + 15

Clear the lowest 4 bits:
- (size + 15) % 16

R = size + 15 – (size + 15) % 16

1. size % 16 == 0: R = size + 15 – 15 = size; R % 16 == 0
2. size % 16 != 0:

size + 15 – (size + 15) % 16



Q10 Set block status

```
typedef struct {
  unsigned long size_and_status;
  unsigned long padding;
} header;
void set_status(header *h, bool status) {
  ...
}
```

Suppose in the implicit list design, the block header is defined as (Lecture slides 22 and 27). Please write a function to set the status of the chunk while leaving its size unchanged? What's the body of the set_status function?

A. h->size_and_status |= (unsigned long)status;

B. h->size_and_status &= (unsigned long)status;

C. h->size_and_status = (h->size_and_status & ~0x1) | (unsigned long)status;

D. h->size_and_status = ((h->size_and_status >> 1) << 1) | (unsigned long)status;

E. h->size_and_status ^= (unsigned long)status;

Q10 Set block status

```
typedef struct {
  unsigned long size_and_status;
  unsigned long padding;
} header;

void set_status(header *h, bool status) {
 ...
}
```

Keep the highest bits.

Suppose in the implicit list design, the block heade cture slides 22 and 27). Please write a function to set th nk while leaving its size unchanged? W

size	status (1bit)

A. h->size_and_status |= (unsigned long)status;

B. h->size_and_status &= (unsigned long)status;

C. h->size_and_status = (h->size_and_status & ~0x1) | (unsigned long)status;

D. h->size_and_status = ((h->size_and_status >> 1) << 1) | (unsigned long)status;

E. h->size_and_status ^= (unsigned long)status;

Q10 Set block status

```
typedef struct {
  unsigned long size_and_status;
  unsigned long padding;
} header;
void set_status(header *h, bool status) {
  ...
}
```

Keep the highest bits.

Suppose in the implicit list design, the block heade cture slides 22 and 27). Please write a function to set th nk while leaving its size unchanged? W

A. h->size_and_status |= (unsigned long)status;

B. h->size_and_status &= (unsigned long)status;

C. h->size_and_status = (h->size_and_status & ~0x1) | (unsigned long)status;

D. h->size_and_status = ((h->size_and_status >> 1) << 1) | (unsigned long)status;

E. h->size_and_status ^= (unsigned long)status;

Dynamic Memory Allocation

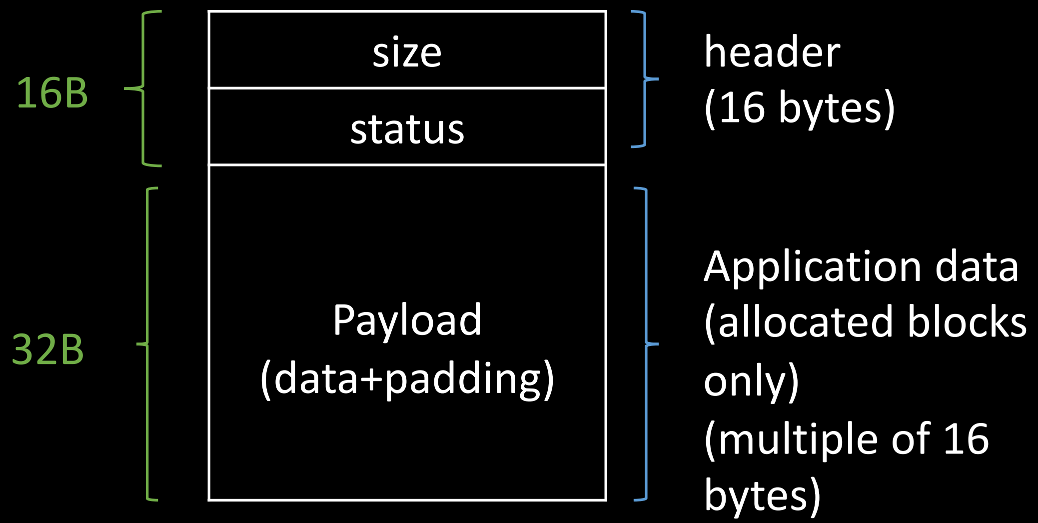
For when static memory isn’t enough

Malloc using Implicit list

1. Structure of implicit list
2. Malloc
    1. Where to place an allocation?
    2. Splitting a free block
3. Free
    1. Coalescing a free block
4. Realloc

Malloc using Implicit list (lab4)

- Structure of implicit list
  - Implicit list means that it does not use pointers explicitly, but it can find the next chunk just like a linked list.
- A chunk:



e.g. p=malloc(20);

Malloc using Implicit list

- Malloc:
  - Find a large enough free chunk
    - Ask_os_for_chunk if not found
  - Place it
    - Split
    - Set status & size

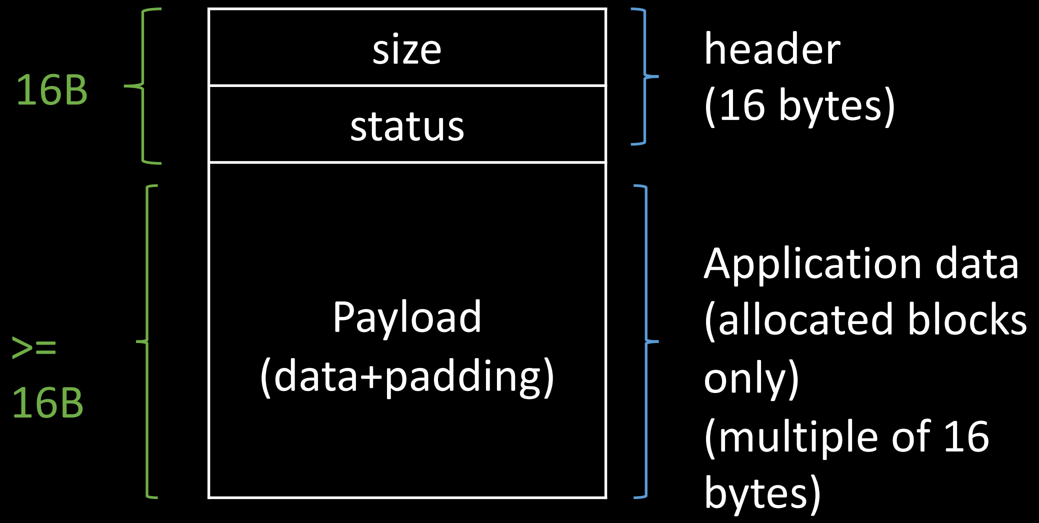
Malloc using Implicit list – find the chunk

- Where to place an allocation?
- Different algorithms:
  - First fit → easy & fast; cause fragmentation at beginning of the heap
  - Best fit → good for utilization; slower
  - Next fit → faster than first fit; even worse fragmentation

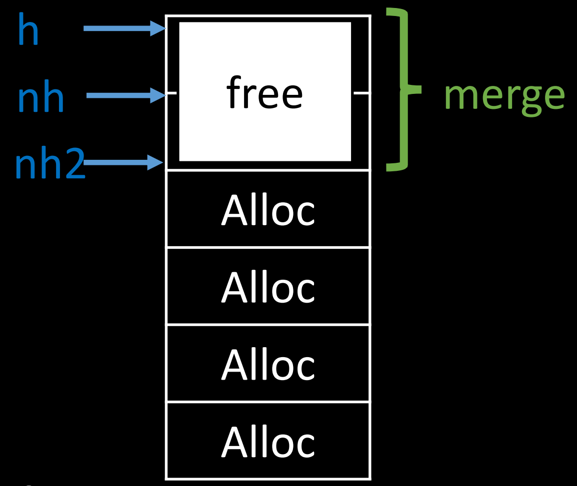
Malloc using Implicit list – place it

- Splitting a free block
  - Compute the remaining size
  - If < MIN_CHUNK_SZ
    - return // don't split
  - else:
    - Split into 2 chunks, and set their size & status

How large is it?

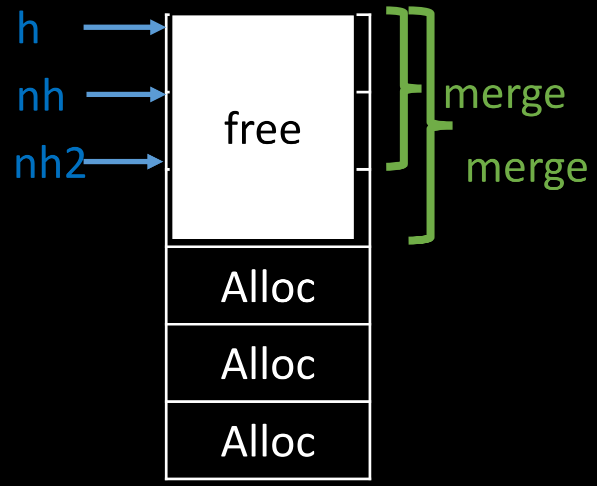


Malloc using Implicit list – free



free(void *p):

- h=payload2header(p) // get chunk pointer
- Set status of the chunk h
- Coalescing a free block: Merge h with its next free neighbor
  - If next_chunk(h) is free {
    - Increase h’s size by next chunk’s size
  - }
  - Any problem with the impl.? Can we do better?
    - Use while instead of if



Why will there be multiple consecutive free chunks?

- Free h2
- Free h3
- Free h1
- Root cause:
  - Cannot coalesce with previous chunk

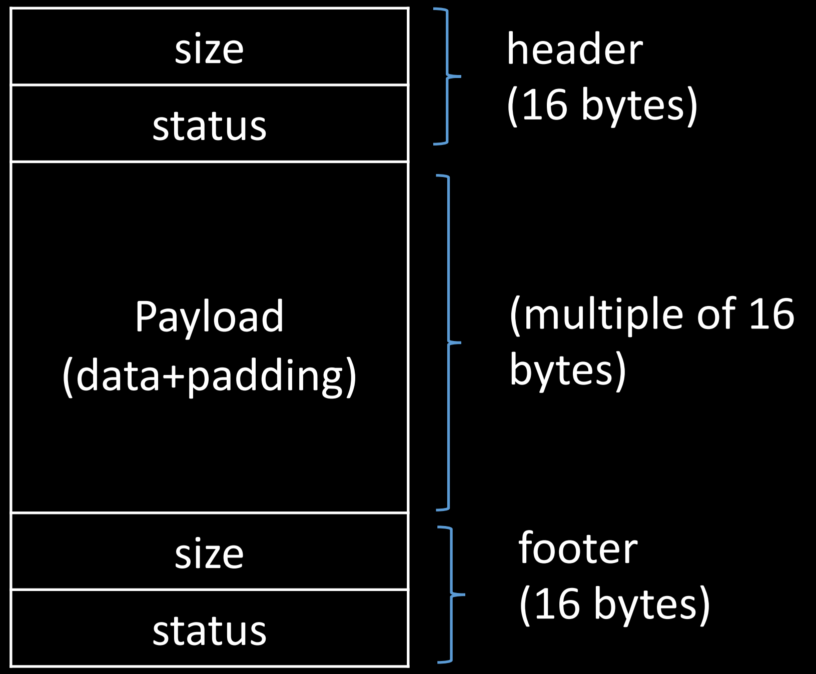
Why will there be multiple consecutive free chunks?

- Free h2
- Free h3
- Root ca
  - Cann
    with previous chunk

Cannot check the status of the previous chunk.

Solution
(not required in lab4

when implementing implicit list)

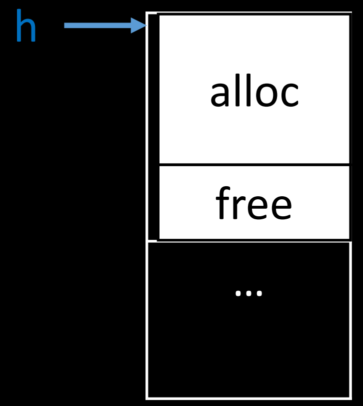


Malloc using Implicit list – realloc

- Realloc(void *p, size_t size): here only discuss p != NULL
  - Resize p’s memory region to the given size
- Example use case: p points to an array;
  - array too small for inserting new element => increase the array size
  - Elements deleted => shrink the array size
- Brute force implementation:
  - q=Malloc(size);
  - Copy p to q;
  - Free(p);
- Problem?
  - Inefficient. Consider shrinking case.

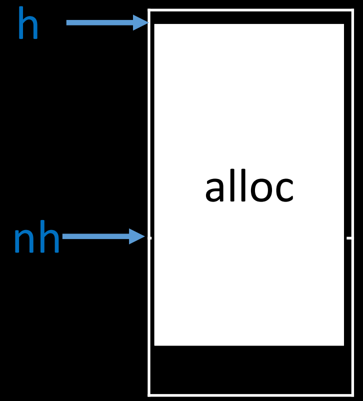
Malloc using Implicit list – realloc

- h=payload2header(p); // get current chunk pointer
- Case 1. Shrinking
  - Split



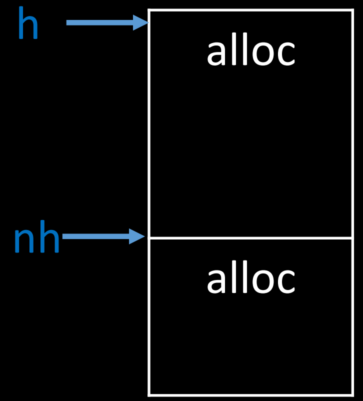
Malloc using Implicit list – realloc

- h=payload2header(p); // get current chunk pointer
- Case 2. Expanding
  - Case 2.1. there is enough space in the next chunk (nh) to accommodate the increased size
    - Utilize the space in the next chunk.



Malloc using Implicit list – realloc

- h=payload2header(p); // get current chunk pointer
- Case 2. Expanding
  - Case 2.2. next chunk is allocated or there is no enough space
    - Fall back to the brute force approach



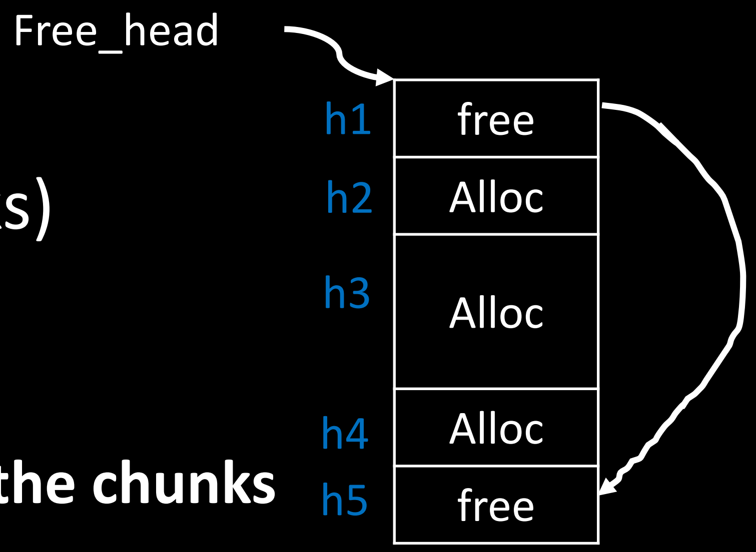
Malloc using Implicit list

- Performance tip:
  - set debug to false
  - wrap any sanity check (e.g., assertions) you wrote with if (debug). E.g.,

```
if (debug) {
        mm_checkheap(true);
}
```

Malloc using Explicit free list

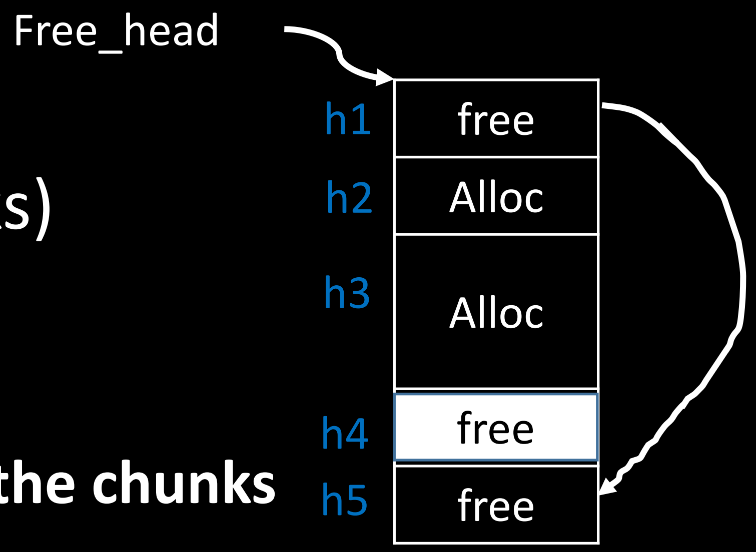
- Implicit list is slow: each malloc can be O(#chunks)
- Explicit free list: O(#free chunks)
  - Chain the free chunks only into a list
  - **Important: list not necessarily in the same order as the chunks**



Link list: h1 <--> h5

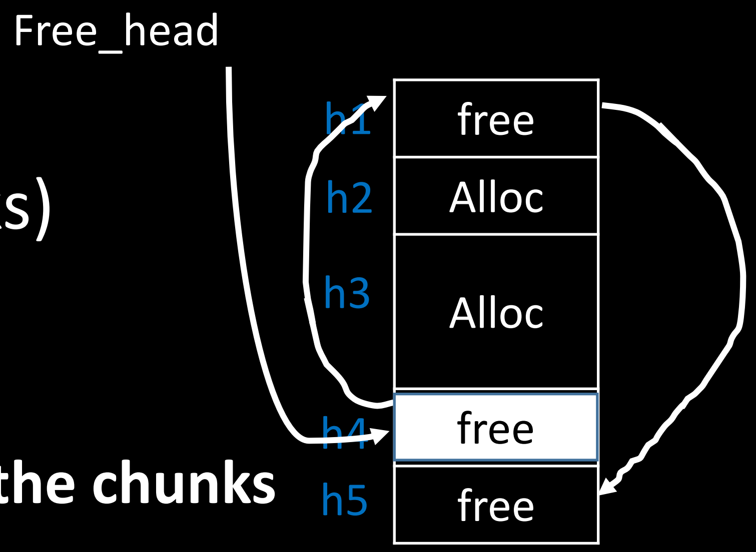
Malloc using Explicit free list

- Implicit list is slow: each malloc can be O(#chunks)
- Explicit free list: O(#free chunks)
  - Chain the free chunks only into a list
  - **Important: list not necessarily in the same order as the chunks**



Malloc using Explicit free list

- Implicit list is slow: each malloc can be O(#chunks)
- Explicit free list: O(#free chunks)
  - Chain the free chunks only into a list
  - **Important: list not necessarily in the same order as the chunks**

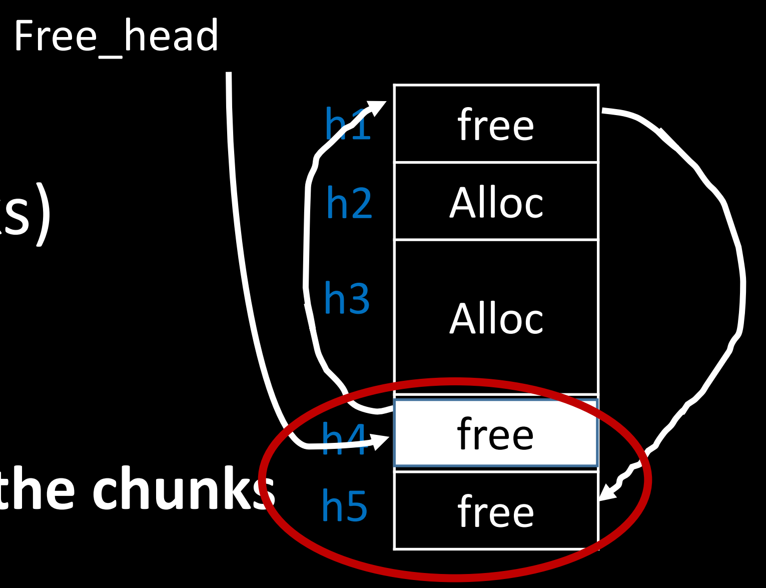


Link list: h4 <--> h1 <--> h5
When free a new block:

- Update the head
- Always insert the new free block at the head

Malloc using Explicit free list

- Implicit list is slow: each malloc can be O(#chunks)
- Explicit free list: O(#free chunks)
  - Chain the free chunks only into a list
  - **Important: list not necessarily in the same order as the chunks**



Challenge: how to coalesce the consecutive free chunks?

- h4 and h5 are not adjacent in the explicit link list
- Traverse the link list?
- No! Borrow the idea from implicit list.
  - Using header and footer

Link list: h4 <--> h1 <--> h5

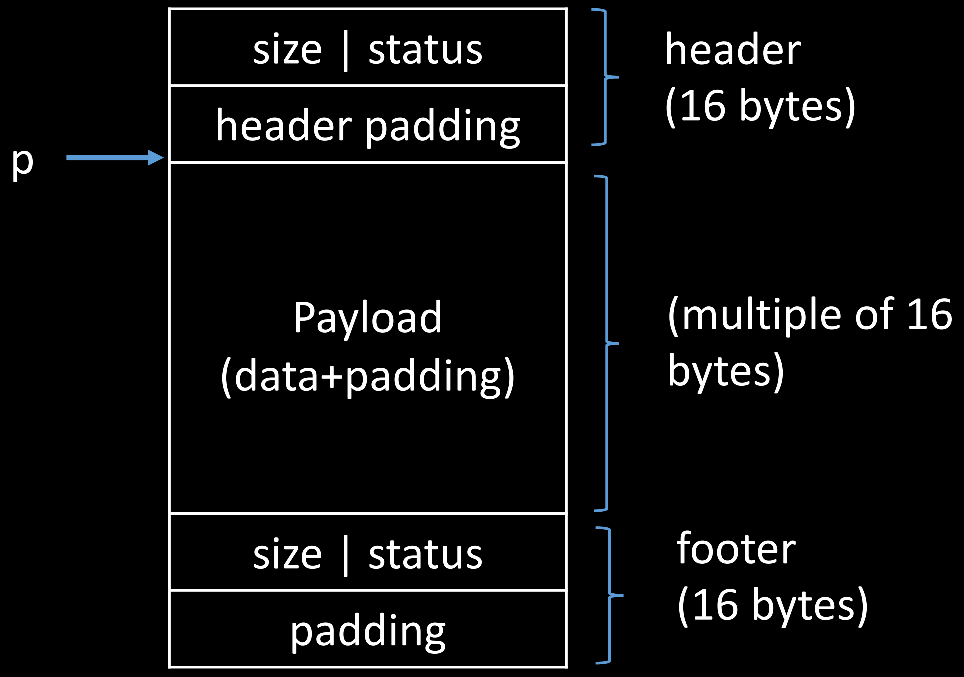
When free a new block:

- Update the head
- Always insert the new free chunk at the head

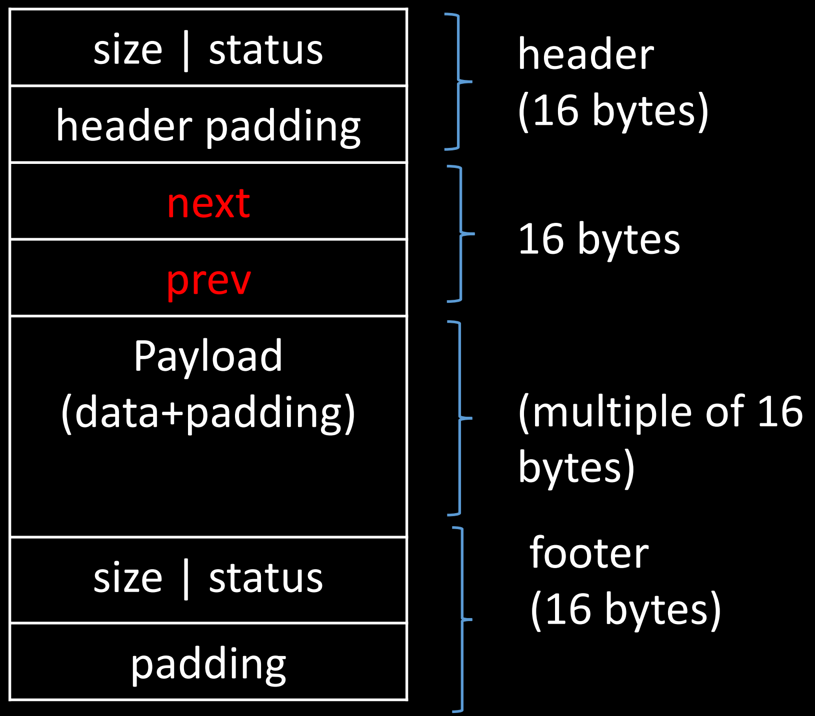
Malloc using Explicit free list

- The structure

Allocated chunk:

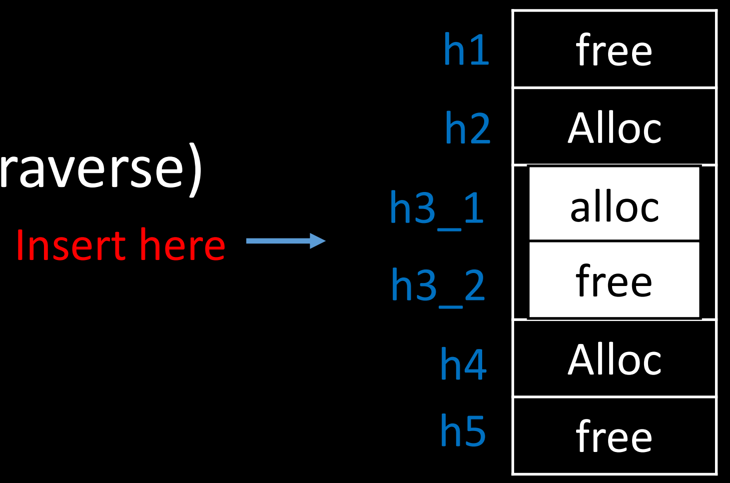


Free chunk:



Implement Malloc using Explicit free list

- Malloc <malloc(size)>
  - find a free chunk (in your linked list – linked list traverse)
  - delete this chunk from the linked list
    - ask the OS for chunk if not found
  - Split chunk
  - If split succeeds
    - set 1st chunk status to allocated
    - insert the 2nd chunk to the linked list
  - return pointer to the payload



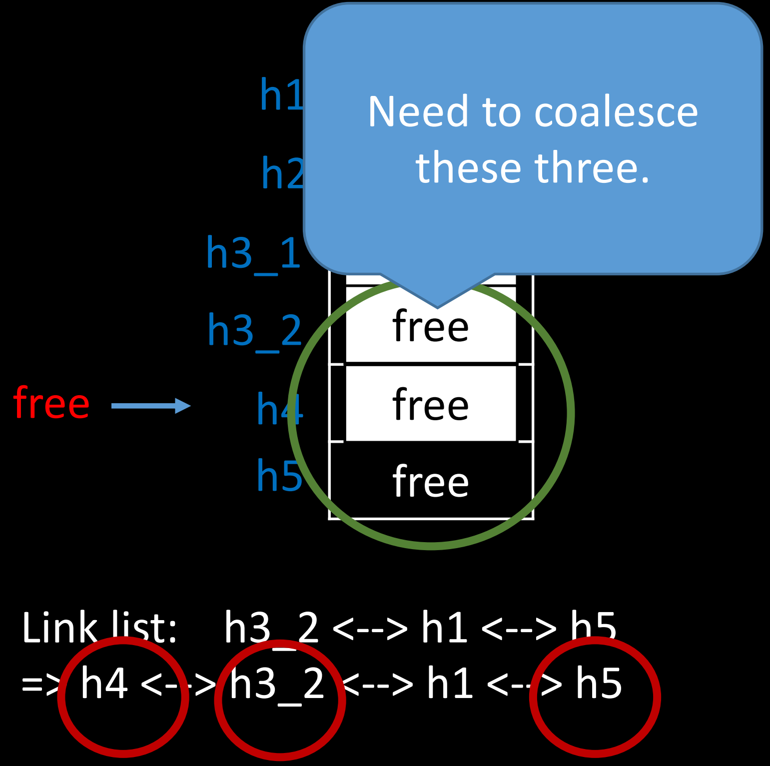
Link list: h1 <---> h5 <---> h3

Link list: h1 <---> h5

Link list: h3_2 <--> h1 <---> h5

Implement Malloc using Explicit free list

- Free <free(p)>
  - go to the header from payload
  - free the chunk
    - set this chunk status to be free
    - initialize the next & prev pointer
  - Coalesce free chunks
    - Use footer to find if consecutive chunks are free
    - If so, delete it from the linked list and merge
  - insert this new free block into the linked list



How to merge the nodes which are not adjacent to each other in the list?

Implement Malloc using Explicit free list

How to merge the nodes which are not adjacent to each other in the list?
=> Keep 1 node, and delete others

Link list: h3_2 <--> h1 <--> h5 <--> h'
⇒ h… <--> h3_2 <--> h1 <--> … <--> h'

⇒ h3_2 (with updated size) <--> h1 <--> h'