Lecture 16: more `malloc()` under the hood

CSE 29: Systems Programming and Software Tools

Olivia Weng

Announcements

- Problem set 4 released

- Sign up for Exam 3 on prairietest.com

Implementation Issues

- How to know how much memory is being `free()`'d when we're only given a pointer (and no length)? **Use headers**
- How to keep track of free memory blocks? **Implicit list + is-allocated bit**
- How to pick which free memory chunks to use for allocation?
  - Many viable options
- What to do with extra space when allocating a block that is smaller than the free block it is placed in?

Implicit List: Finding a Free Block

- First fit:
  - Search list from beginning, choose *first* free block that fits

Implicit List: Finding a Free Block

- Next fit:
  - Like first fit, but start searching from *where previous search finished*
  - Often is *faster* than first fit: avoids re-scanning unhelpful blocks

Implicit List: Finding a Free Block

- Best fit:
    - Search the list, choosing the best free block: fits with fewest bytes leftover
    - Will typically run slower than first fit

Implicit List: Finding a Free Block

- First fit:
  - Search list from beginning, choose *first* free block that fits
- Next fit:
  - Like first fit, but start searching from *where previous search finished*
  - Often is *faster* than first fit: avoids re-scanning unhelpful blocks
- Best fit:
  - Search the list, choosing the best free block: fits with fewest bytes leftover
  - Will typically run slower than first fit

- Which one is better?

Implementation Issues

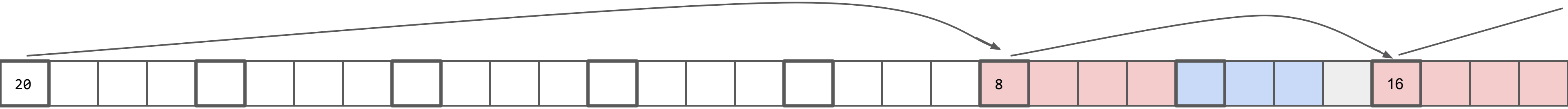
- How to know how much memory is being `free()`'d when we're only given a pointer (and no length)? **Use headers**
- How to keep track of free memory blocks? **Implicit list + is-allocated bit**
- How to pick which free memory chunks to use for allocation?
  - Many viable options **First fit, Next fit, Best fit**
- What to do with extra space when allocating a block that is smaller than the free block it is placed in?

Implementation Issues

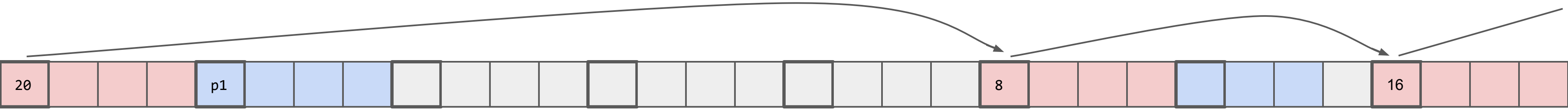
- How to know how much memory is being `free()`'d when we're only given a pointer (and no length)? **Use headers**
- How to keep track of free memory blocks? **Implicit list + is-allocated bit**
- How to pick which free memory chunks to use for allocation?
  - Many viable options **First fit, Next fit, Best fit**
- What to do with extra space when allocating a block that is smaller than the free block it is placed in?

Implicit List: Allocating in a Free Block

- 🟥 Header byte
- 🟦 Allocated byte
- ▨ Padding byte
- □ Free byte
- ▣ Byte whose address `% 4 == 0`



```
p1 = malloc(4)
```



Implicit List: Allocating in a Free Block

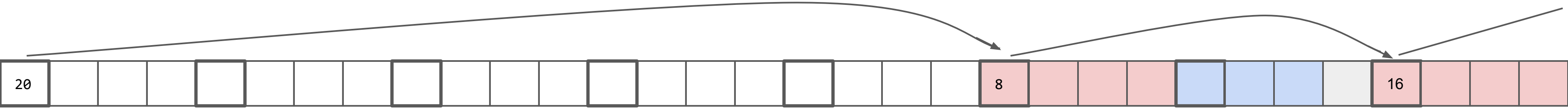
🟥 Header byte

🟦 Allocated byte

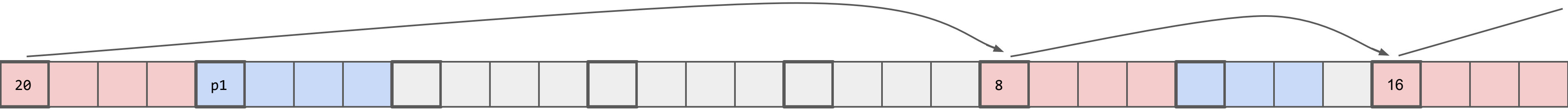
▨ Padding byte

□ Free byte

▢ Byte whose address `% 4 == 0`



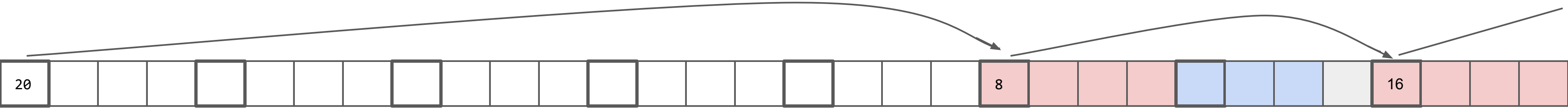
```
p1 = malloc(4)
```



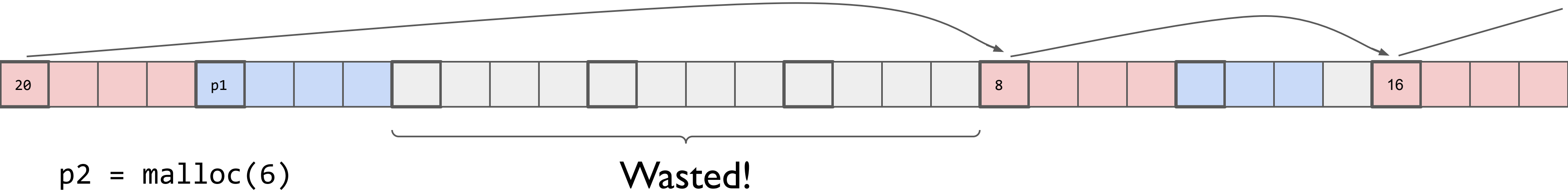
```
p2 = malloc(6)
```

Implicit List: Allocating in a Free Block

- 🟥 Header byte
- 🟦 Allocated byte
- ⬜ Padding byte
- □ Free byte
- ▣ Byte whose address `% 4 == 0`



```
p1 = malloc(4)
```

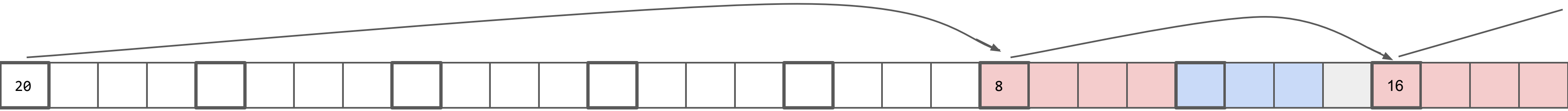


```
p2 = malloc(6)
```

Implicit List: Allocating in a Free Block

- 🟥 Header byte
- 🟦 Allocated byte
- ▨ Padding byte
- □ Free byte
- ▢ Byte whose address `% 4 == 0`

- Splitting a free block
  - Since object might be smaller than free space, we might want to split the free block

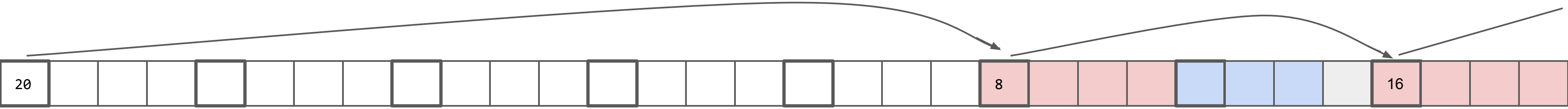


```
p1 = malloc(4)
```

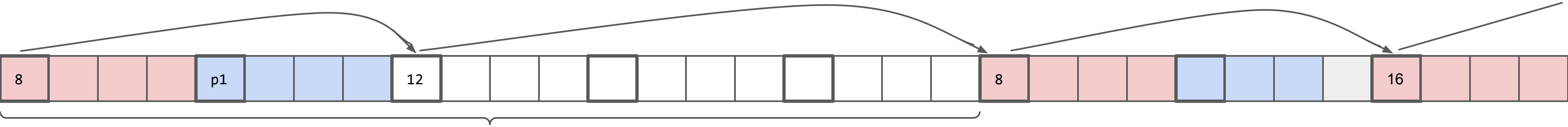
Implicit List: Allocating in a Free Block

- 🟥 Header byte
- 🟦 Allocated byte
- ▨ Padding byte
- □ Free byte
- ▣ Byte whose address `% 4 == 0`

- Splitting a free block
  - Since object might be smaller than free space, we might want to split the free block



```
p1 = malloc(4)
```

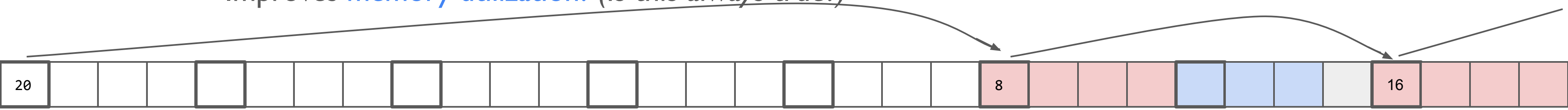


Free block split into allocated block and free block

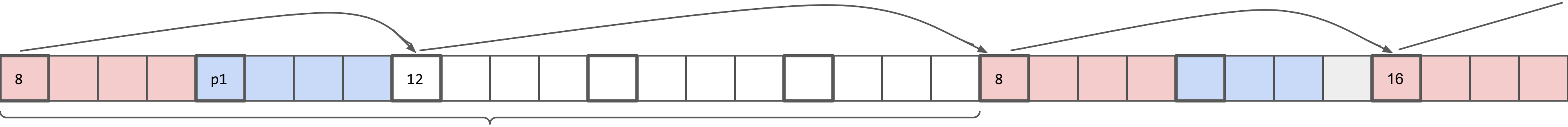
Implicit List: Allocating in a Free Block

- 🟥 Header byte
- 🟦 Allocated byte
- ▒ Padding byte
- □ Free byte
- ▣ Byte whose address `% 4 == 0`

- Splitting a free block
  - Since object might be smaller than free space, we might want to split the free block
  - Improves memory utilization! *(Is this always true?)*



```
p1 = malloc(4)
```



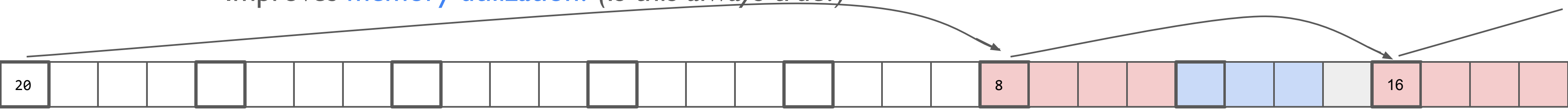
Free block split into allocated block and free block

```
p2 = malloc(6)
```
Now possible!

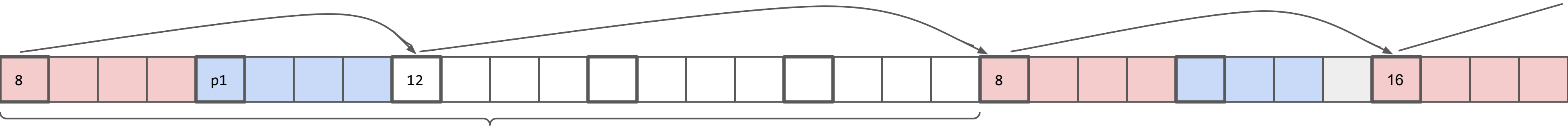
Implicit List: Allocating in a Free Block

- 🟥 Header byte
- 🟦 Allocated byte
- ▨ Padding byte
- □ Free byte
- ▢ Byte whose address `% 4 == 0`

- Splitting a free block
    - Since object might be smaller than free space, we might want to split the free block
    - Improves memory utilization! *(Is this always true?)*



`p1 = malloc(4)`



Free block split into allocated block and free block

`p2 = malloc(6)` Now possible!

Is it always worth it to split your free blocks?

Implementation Issues

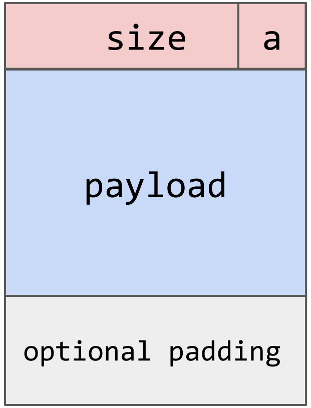
- How to know how much memory is being `free()`'d when we're only given a pointer (and no length)? **Use headers**
- How to keep track of free memory blocks? **Implicit list + is-allocated bit**
- How to pick which free memory chunks to use for allocation?
  - Many viable options **First fit, Next fit, Best fit**
- What to do with extra space when allocating a block that is smaller than the free block it is placed in? **Splitting**
- How to reinsert freed memory into the heap?

Implementation Issues

- How to know how much memory is being `free()`'d when we're only given a pointer (and no length)? **Use headers**
- How to keep track of free memory blocks? **Implicit list + is-allocated bit**
- How to pick which free memory chunks to use for allocation?
  - Many viable options **First fit, Next fit, Best fit**
- What to do with extra space when allocating a block that is smaller than the free block it is placed in? **Splitting**
- How to reinsert freed memory into the heap?

Implicit List: Freeing a Block

- Simplest implementation:
  - Need only set the “allocated” flag in header to 0



`size` `a`

`payload`

`optional padding`

`a = 1`: allocated block

`a = 0`: free block

`size` = block size

`payload`: object data (allocated blocks only)

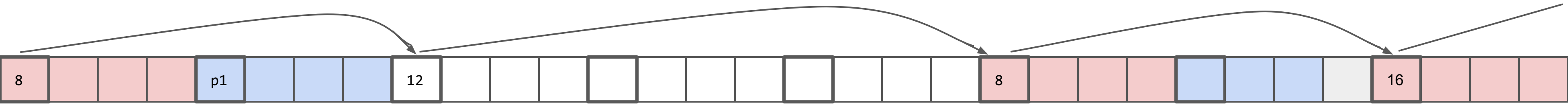
How to set the is-allocated bit to 0?

- Header: 0x000000C1
  - Assume 4 byte unsigned int

- 🟥 Header byte
- 🟦 Allocated byte
- ⬜ Padding byte
- □ Free byte
- ▢ Byte whose `address % 4 == 0`

Implicit List: Freeing a Block

- Simplest implementation:
    - Need only set the “allocated” flag in header to 0



```
free(p1)
```

Implicit List: Freeing a Block

- Simplest implementation:
  - Need only set the “allocated” flag in header to 0

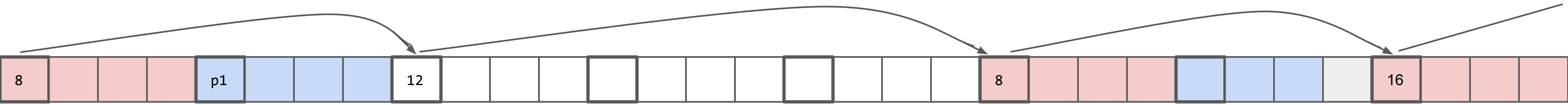
🟥 Header byte

🟦 Allocated byte

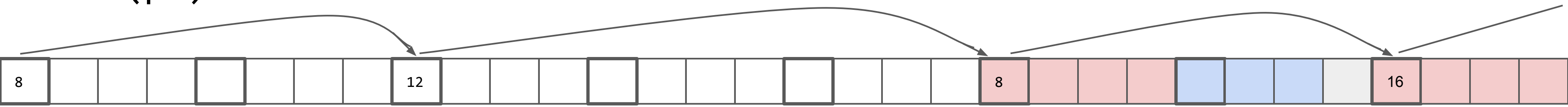
▒ Padding byte

□ Free byte

▣ Byte whose address `% 4 == 0`



```
free(p1)
```



Implicit List: Freeing a Block

- Simplest implementation:
  - Need only set the “allocated” flag in header to 0

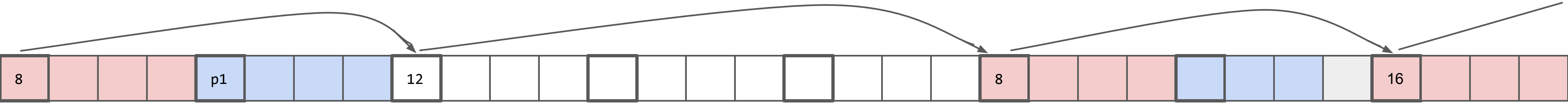
🟥 Header byte

🟦 Allocated byte

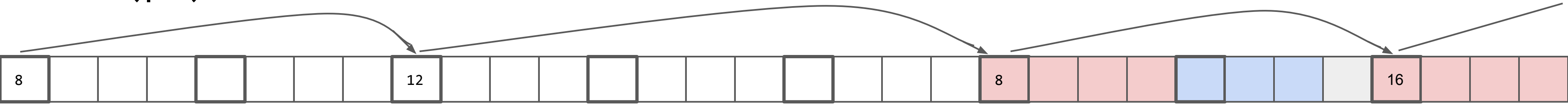
▨ Padding byte

□ Free byte

▢ Byte whose address `% 4 == 0`



```
free(p1)
```

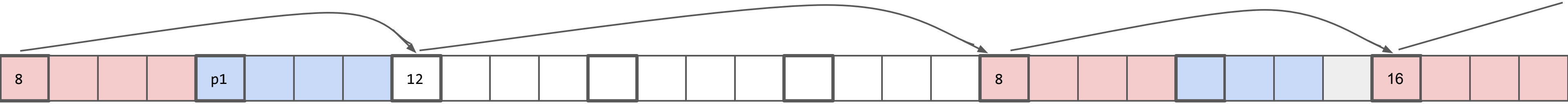


```
malloc(10)
```

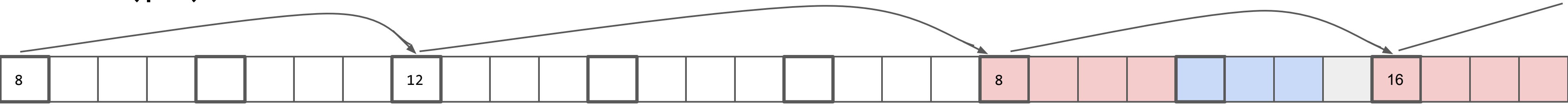
Implicit List: Freeing a Block

- Simplest implementation:
  - Need only set the “allocated” flag in header to 0
  - But leads to *poor memory utilization*

- 🟥 Header byte
- 🟦 Allocated byte
- ▨ Padding byte
- □ Free byte
- ▣ Byte whose address `% 4 == 0`



`free(p1)`



`malloc(10)` Oops!

There is enough free space, but the allocator won’t be able to find it!

Implicit List: Coalescing

- When freeing a block, join (coalesce) it with next/previous blocks, if they are free

🟥 Header byte

🟦 Allocated byte

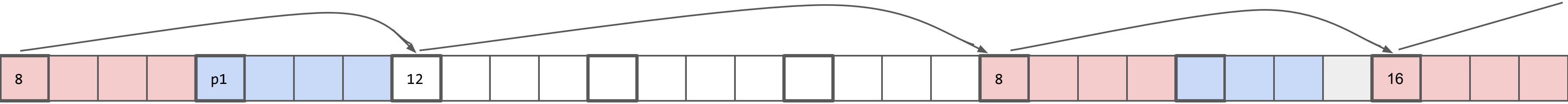
▨ Padding byte

□ Free byte

▣ Byte whose address `% 4 == 0`

Implicit List: Coalescing

- When freeing a block, join (coalesce) it with next/previous blocks, if they are free



```
free(p1)
```

Implicit List: Coalescing

- When freeing a block, join (coalesce) it with next/previous blocks, if they are free

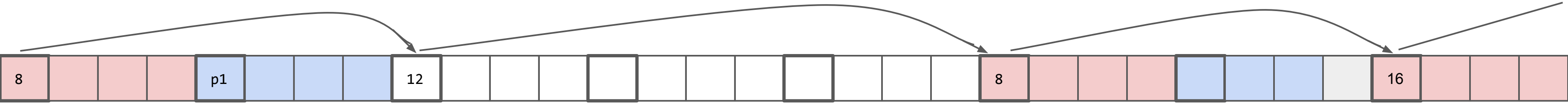
🟥 Header byte

🟦 Allocated byte

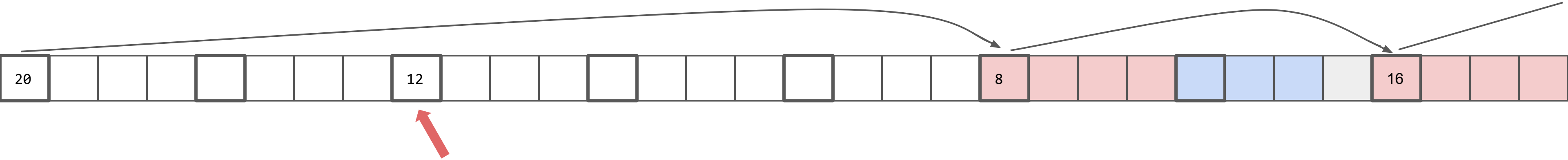
⬜ Padding byte

□ Free byte

□ Byte whose address `% 4 == 0`



`free(p1)`



Ignored

Four cases for coalescing

1. Previous and next blocks are both allocated

Four cases for coalescing

2. Next block is free

Four cases for coalescing

3. Previous block is free

Four cases for coalescing

4. Previous and next blocks are free

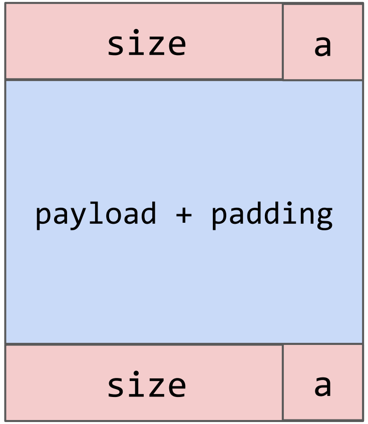
How to find the previous block?

- Search again from the start of the heap to find previous block

- Problem: Time-consuming
  - How can we save time?

How to find the previous block?

- Block footers!



size a

payload + padding

size a

Block header (4 bytes)

a = 1: allocated
a = 0: free

Block footer (4 bytes)

What tradeoffs do block footers introduce?

Problem with footers

- Problem: High memory overhead to have footers for ALL blocks

Problem with footers

- Problem: High memory overhead to have footers for ALL blocks

- Solution: Only free blocks have footers

How to check if previous block is free or not?

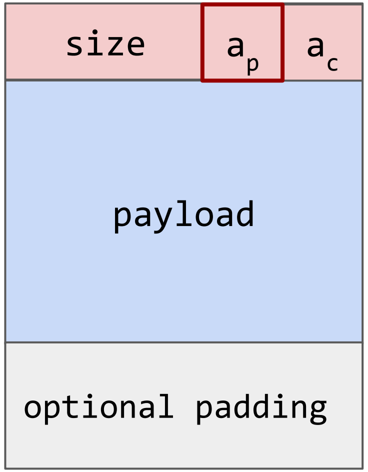
- Problem: pointer arithmetic insufficient!
    - If previous block is free: all good!

How to check if previous block is free or not?

- Problem: pointer arithmetic insufficient!
  - If previous block is free: all good!
  - If previous block is allocated: bad!

Storing more metadata in the header

- Structure of allocated block
  - Since memory is 4-byte aligned, the 2 lowest-order address bits are always 0
  - LSB = current block status
  - 2nd LSB = previous block status



a_c = 1: current block alloc'd
a_c = 0: current block free

a_p = 1: previous block alloc'd
a_p = 0: previous block free

What is the size, current status, and previous status?

- Header = 0x82 => 0x00000082
  - Assume 4-byte unsigned int

Coalescing with the previous block

- Steps:
  - check previous block status in my header
  - if free: look at footer and use info to update previous block's header (pointer arithmetic!)

When to coalesce?

- Immediate coalescing: coalesce each time `free()` is called
- Deferred coalescing: try to improve performance of `free()` by deferring coalescing until needed.
  - Ex:
    - Coalesce as you scan the heap for malloc()
    - Coalesce when the memory utilization reaches some threshold

- Which one is better?
  - Remember throughput vs. memory utilization tradeoff