MINI VALGRIND

TO LEARN

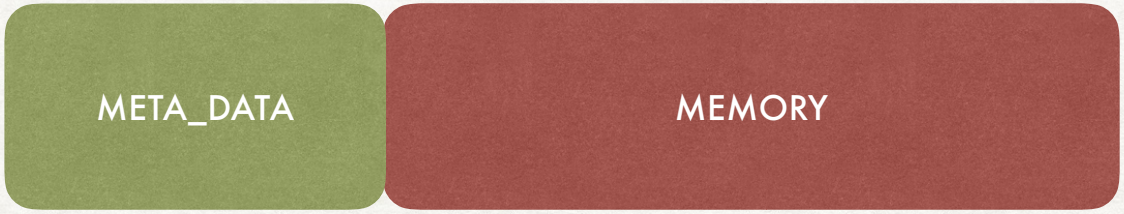
- Learn how Valgrind works
- Learn memory layout and how malloc/free works
  - How does free know how many bytes it has to free?

MALLOC/FREE & MEMORY LAYOUT

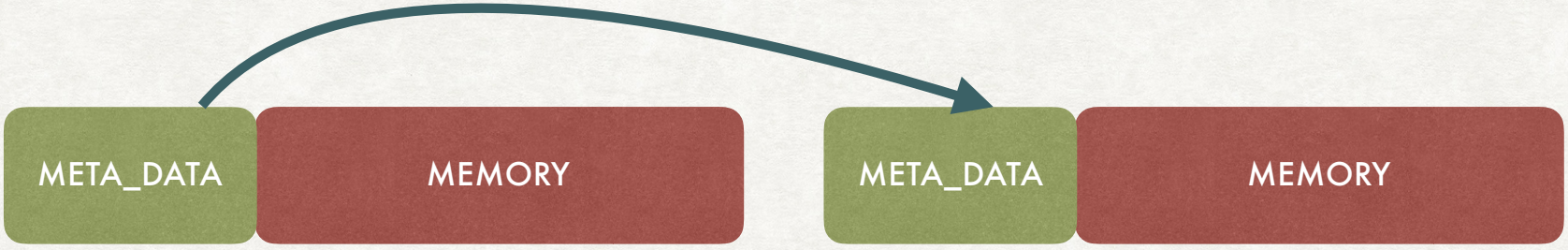
- How does free know how many bytes to free?
- How does those ‘free block’ be reused?
  - You need some meta data to tell you the size of a memory block
  - Put them into a list when you free memory blocks
- When malloc
  - Find a block in the list that is big enough
  - Grow heap otherwise

MINI VALGRIND

- Learn how to create meta data and combine them with memory asked by users



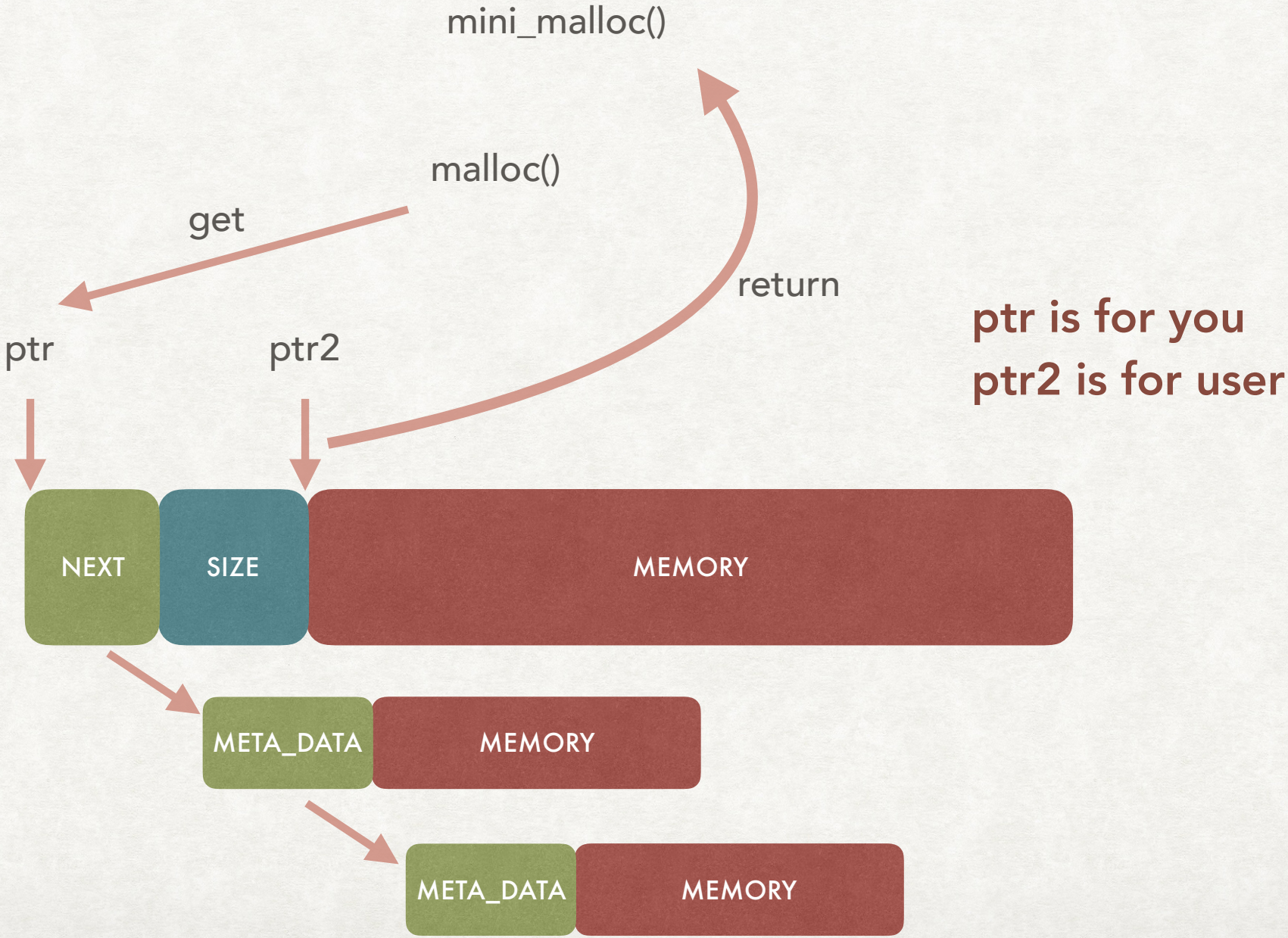
- Learn how to use linked-list to link memory blocks



MINI VALGRIND

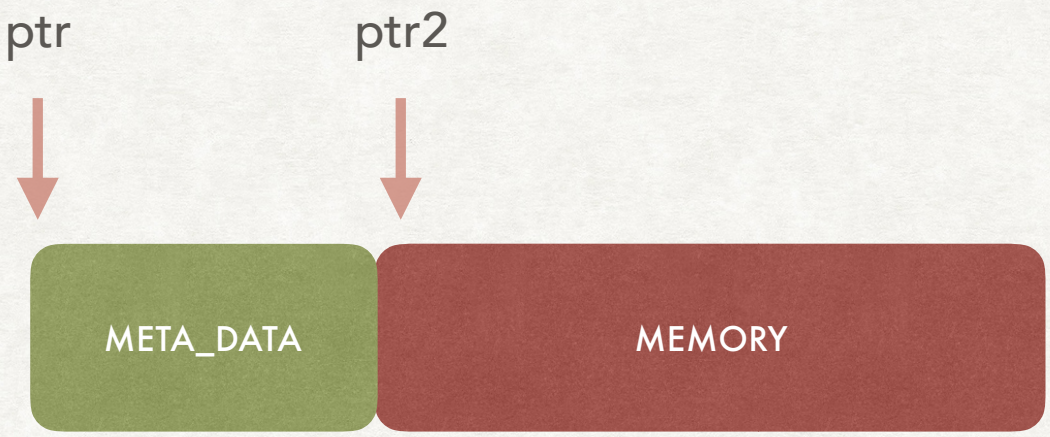
- mini_malloc(size_t size, const char * file, size_t line)
  - use system malloc to get memory
  - create meta data
  - put the memory in a list
  - update total_usage
- mini_free(void * ptr)
  - remove from list
  - use system free to free memory
  - update total_free

MEMORY LAYOUT



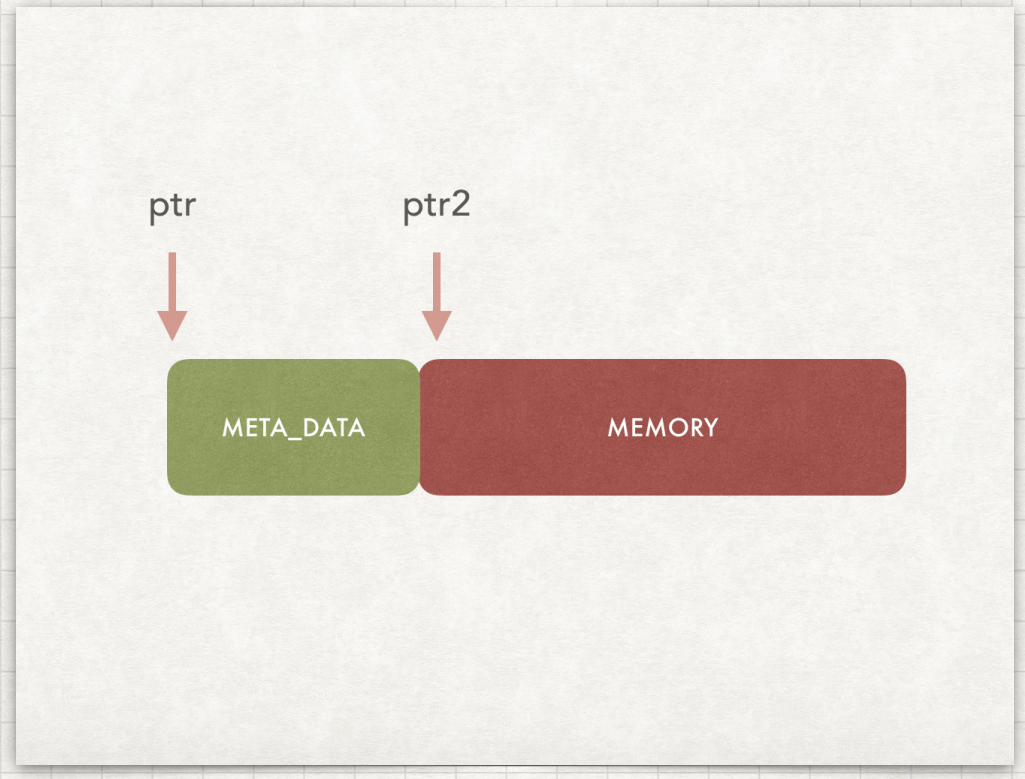
ptr is for you
ptr2 is for user

MEMORY LAYOUT



MEMORY LAYOUT

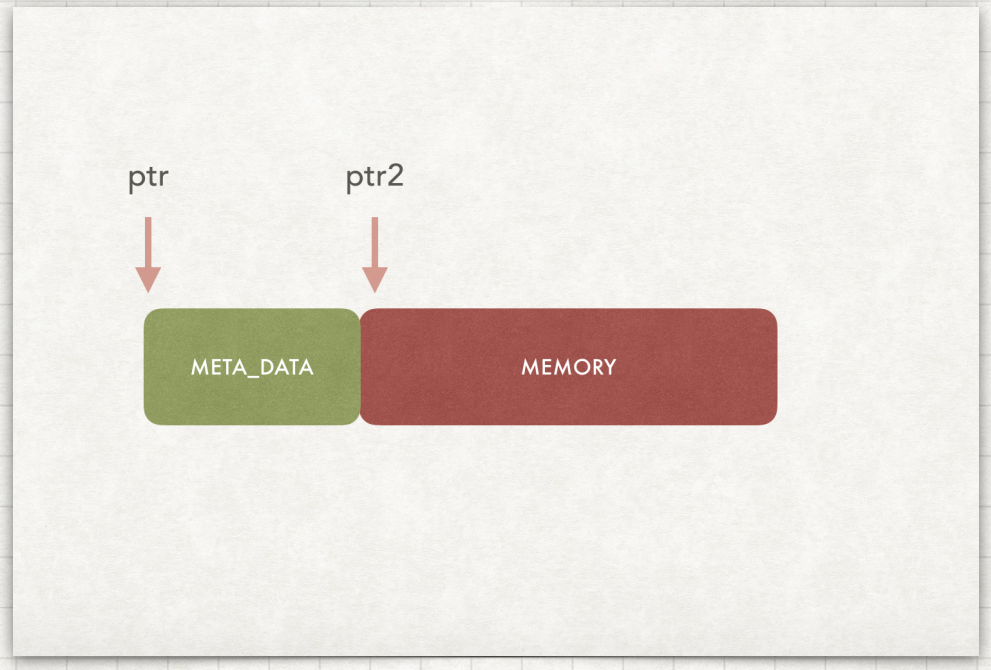
- Malloc
  - Get ptr return ptr2
- Free
  - Get ptr2 return ptr
- How to calculate ptr/ptr2 from ptr2/ptr?



MEMORY LAYOUT

SMALL TRICK

- Zero-Length Array
- struct my_wrap{
  - M
- };



HINT

- Global variables in mini_valgrind.c
  - meta_data * head : head for linked-list
  - total_usage: keep track of bytes used
  - total_free: keep track of bytes freed
- Split linked-list logic to separate functions.
- Insert has to be constant time.
- Have to catch bad calls of free(like double free).

DEMO

```
  char *ptr1 = (char *)malloc(1000);
  char *ptr4 = (char *)malloc(10);
  int invalid;

  free(ptr1);
  ptr1 = NULL;
  free(ptr1);
  free(ptr4);
free(ptr4);
  ptr4 = NULL;
  free(&invalid);
```

```
Heap report:


Program made 2 bad call(s) to free.

Heap summary:

  Total memory usage: 1010 bytes, Total memory free: 1010 bytes
  No leak, all memory are freed, Congratulations!
```

DEMO

```
char *ptr1 = (char *)malloc(1000);
char *ptr4 = (char *)malloc(10);
int invalid;

free(ptr1);
ptr1 = NULL;
free(ptr1);
free(ptr4);
free(ptr4);
ptr4 = NULL;
free(&invalid);
```

```
Heap report:


Program made 2 bad call(s) to free.

Heap summary:

  Total memory usage: 1010 bytes, Total memory free: 1010 bytes
  No leak, all memory are freed, Congratulations!
```

DOUBLE FREE

DEMO

```
 9      char *ptr1 = (char *)malloc(1000);
10      char *ptr4 = (char *)malloc(10);
11      int invalid;
12
13      free(ptr1);
14      ptr1 = NULL;
15      free(&invalid);
16      ptr4 = NULL;
17
18
```

```
Heap report:

  Leak file test.c : line 10
  Leak size : 10 bytes
  Leak memory address : 0x7fafe2403280


Program made 1 bad call(s) to free.

Heap summary:

  Total memory usage: 1010 bytes, Total memory free: 1000 bytes
  Total leak: 10 bytes
```

memory leak: ptr4