CSE 351: Week 9

Tom Bergan, TA

Today

- Lab 5
- Reference counting

Lab 5: Explicit free list allocator

Your goals

Implement: `void* mm_malloc(size_t bytes);`

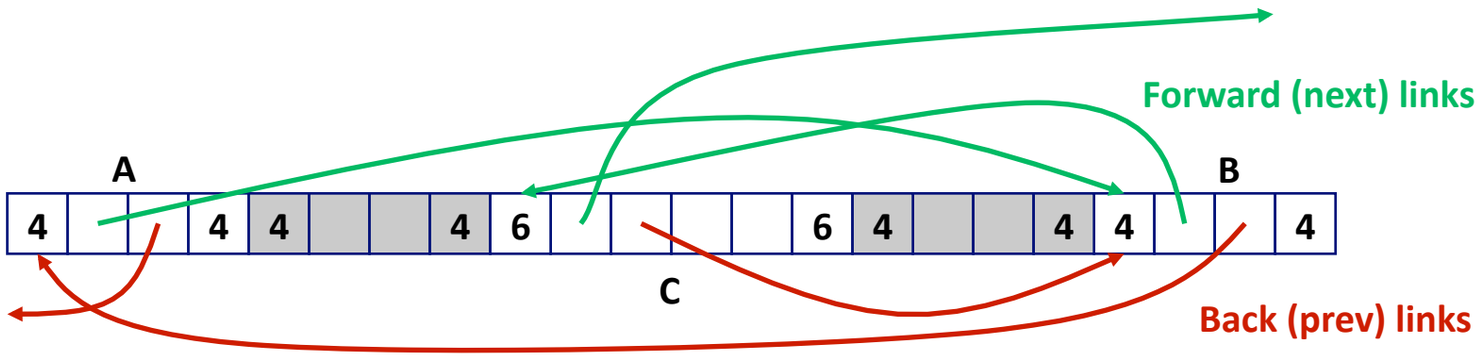
Implement: `void mm_free(void *ptr);`

We give you a starting point

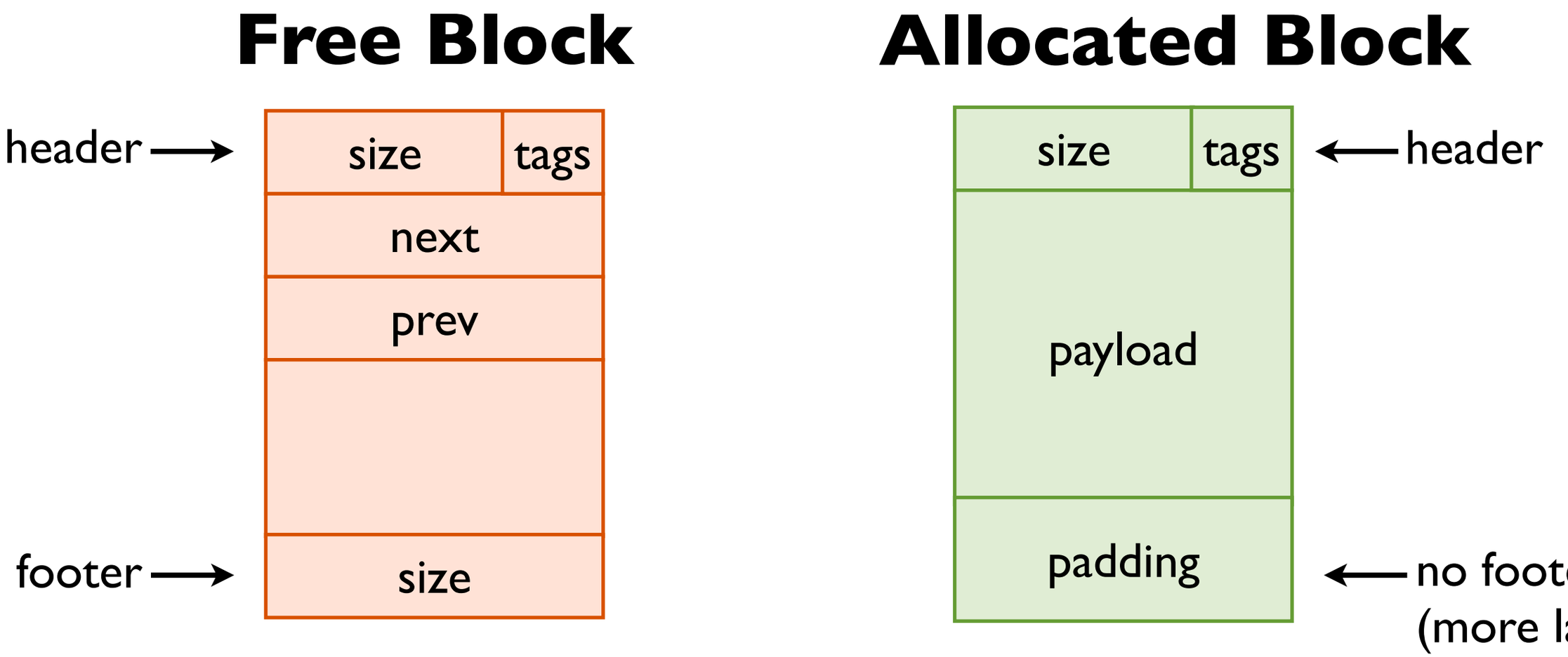
Lab 5: Explicit free list allocator

The free-block list is a doubly-linked list

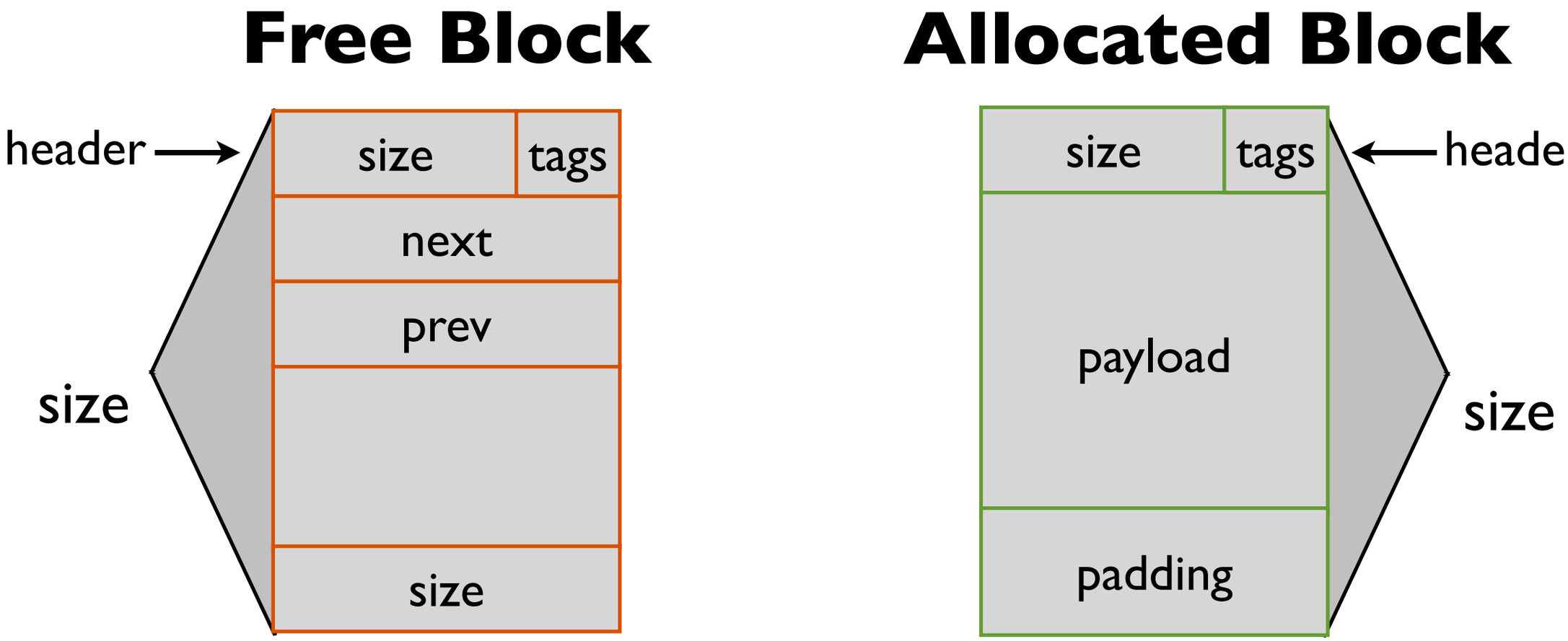
Physical view (blocks can be in any order)



Lab 5: Block format

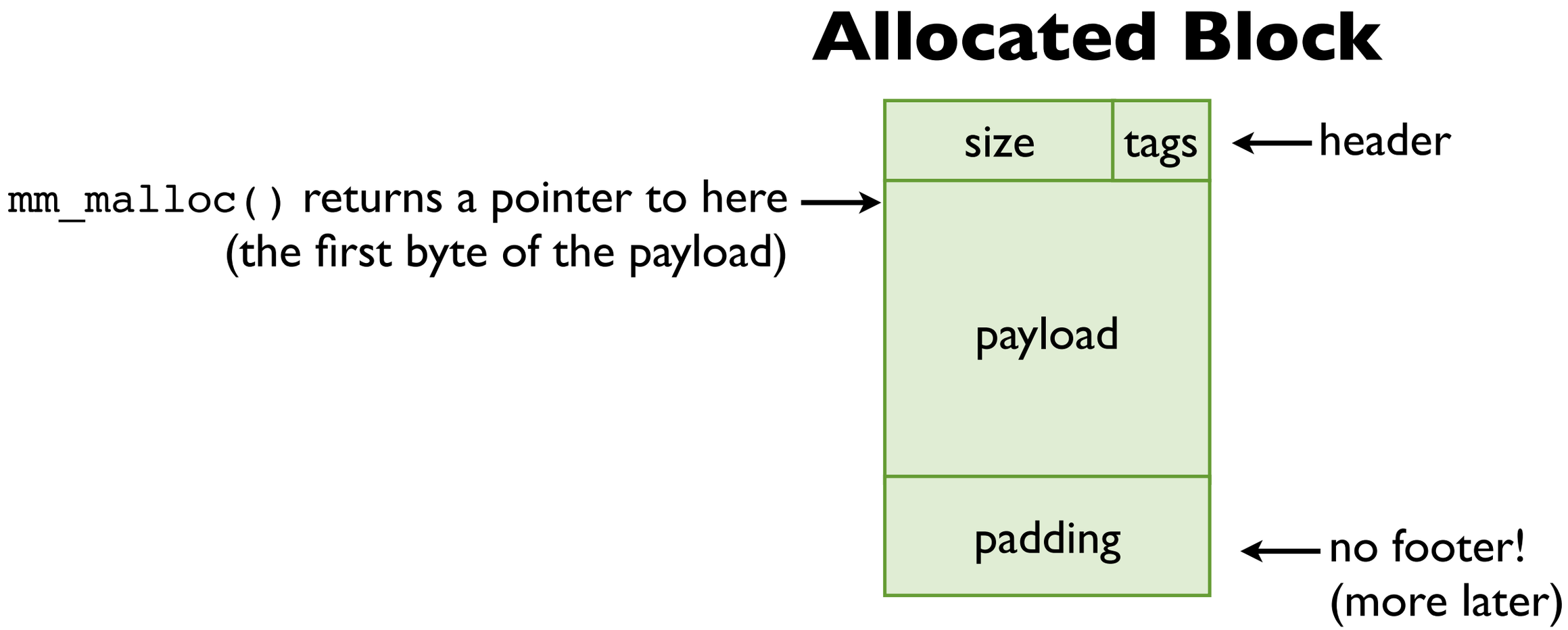


Lab 5: Block format



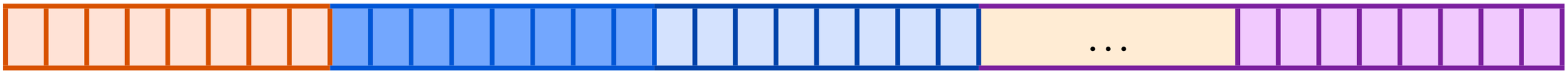
The size includes the <u>whole</u> block

Lab 5: Block format



Lab 5: Block format

Free Block



header
8 bytes

next
8 bytes

prev
8 bytes

(empty space)

footer
8 bytes

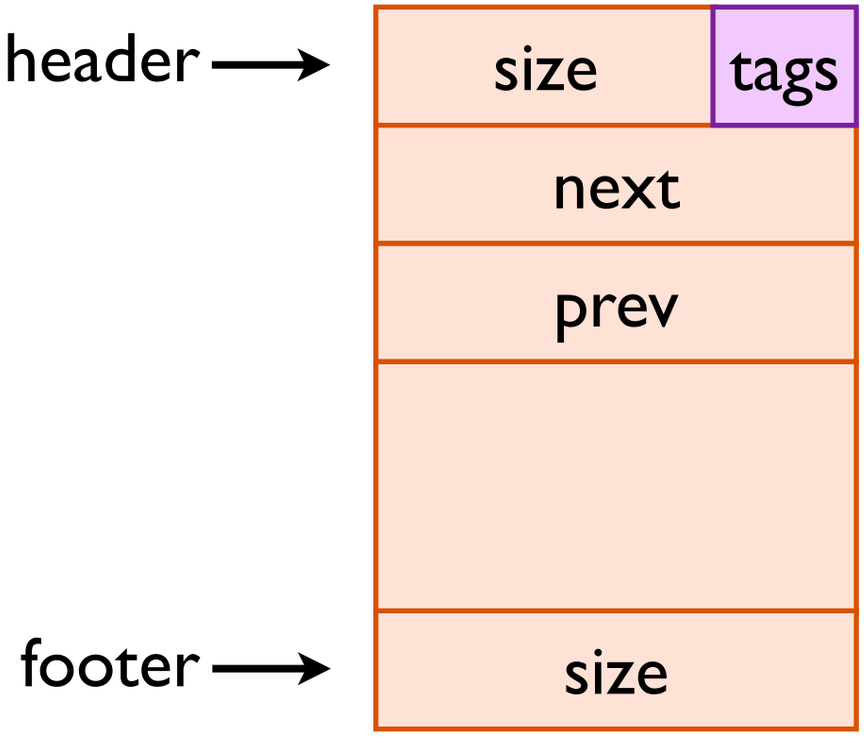
Allocated Block

header
8 bytes

`mm_malloc()` returns a pointer to here
(the first byte of the payload)

Lab 5: Block tags

Free Block

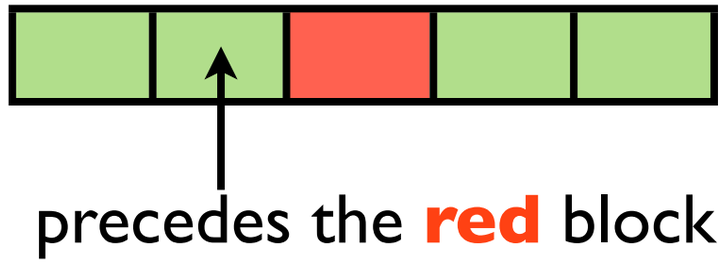


Size **Tag**

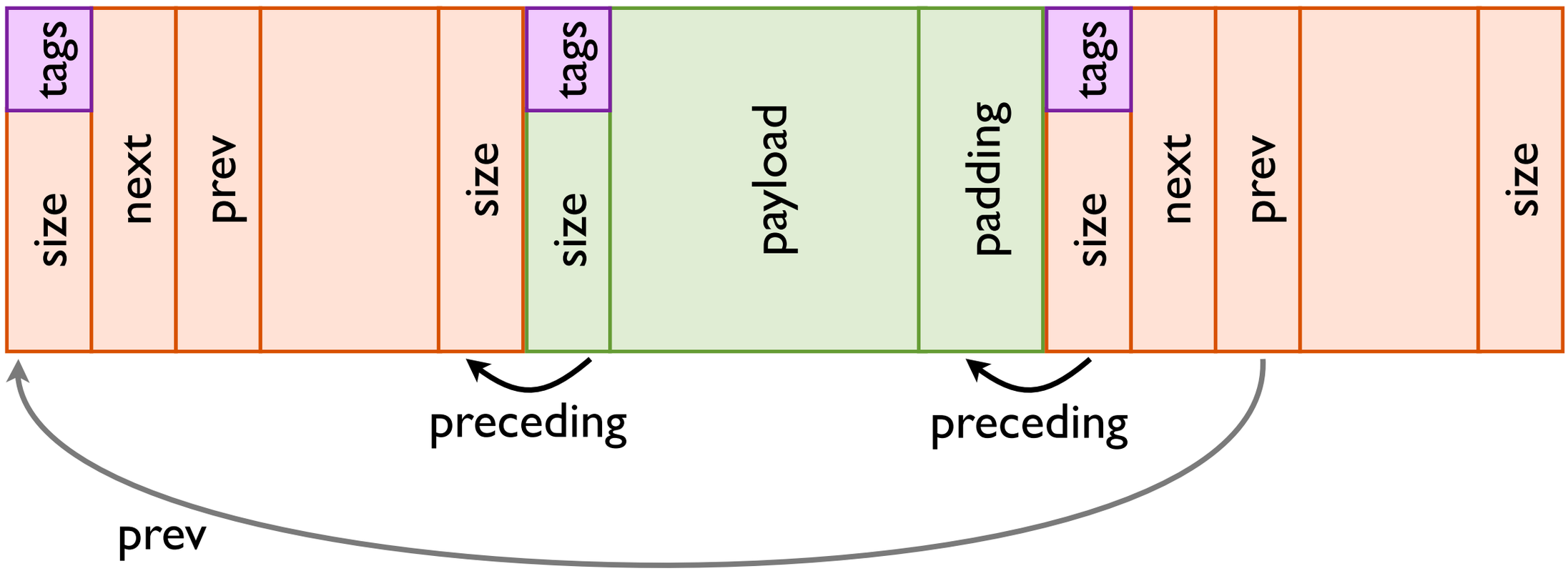
Size: aligned to $2^3 = 8$

Tag bit 0: this block allocated?
Tag bit 1: preceding block allocated?
Tag bit 2: unused

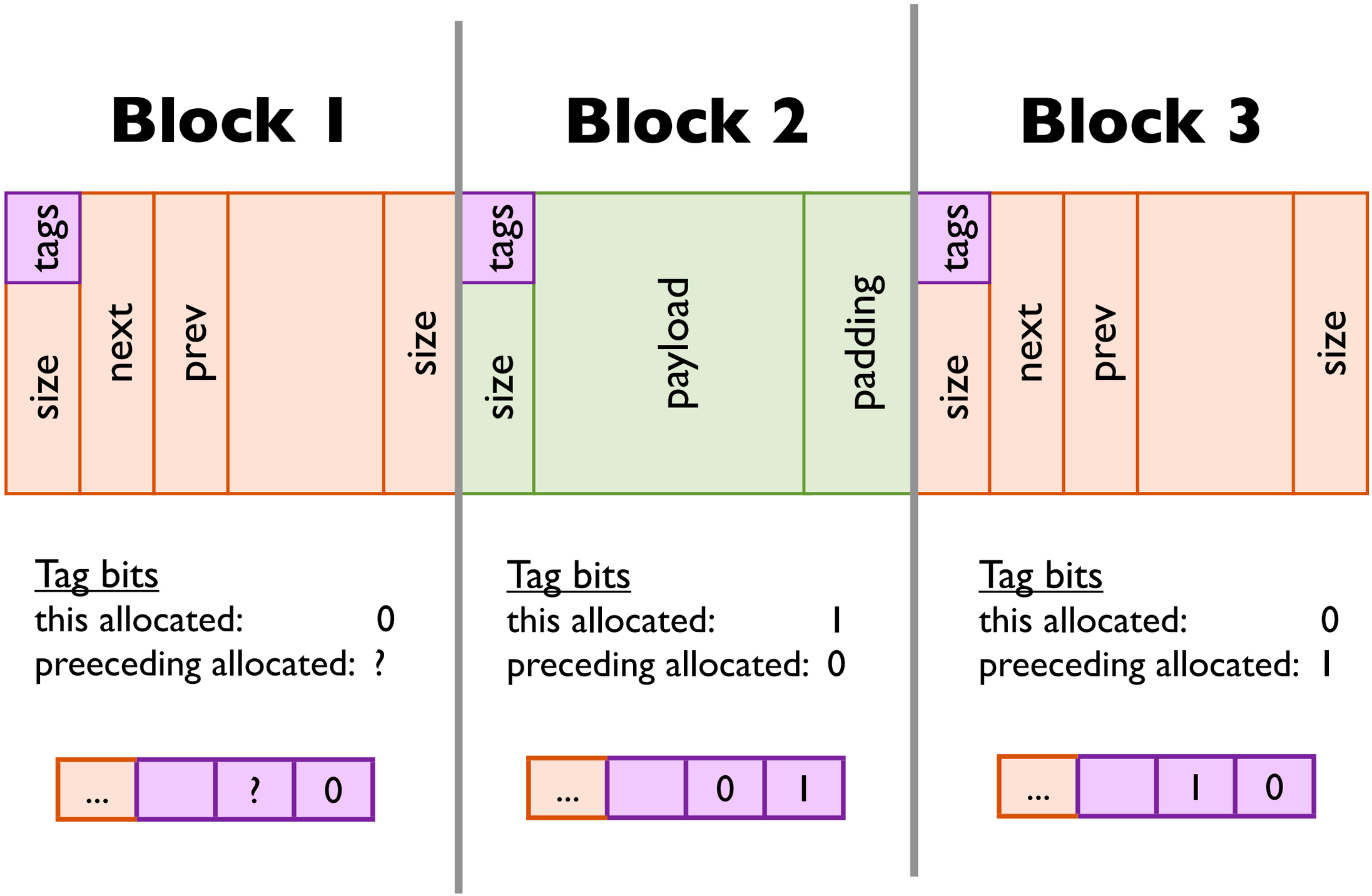
Adjacent blocks in memory



Lab 5: Adjacent blocks



Lab 5: Adjacent blocks

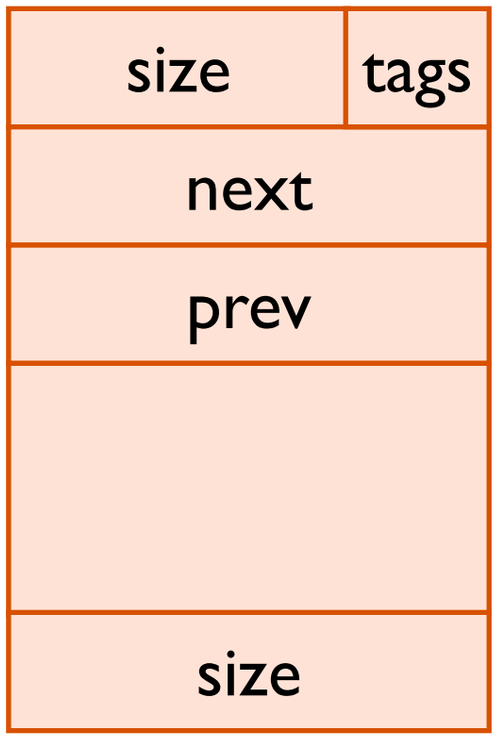


Lab 5: Block data structure

mm.c

```
struct BlockInfo {
  size_t sizeAndTags;
  struct BlockInfo *next;
  struct BlockInfo *prev;
};
```

Free Block



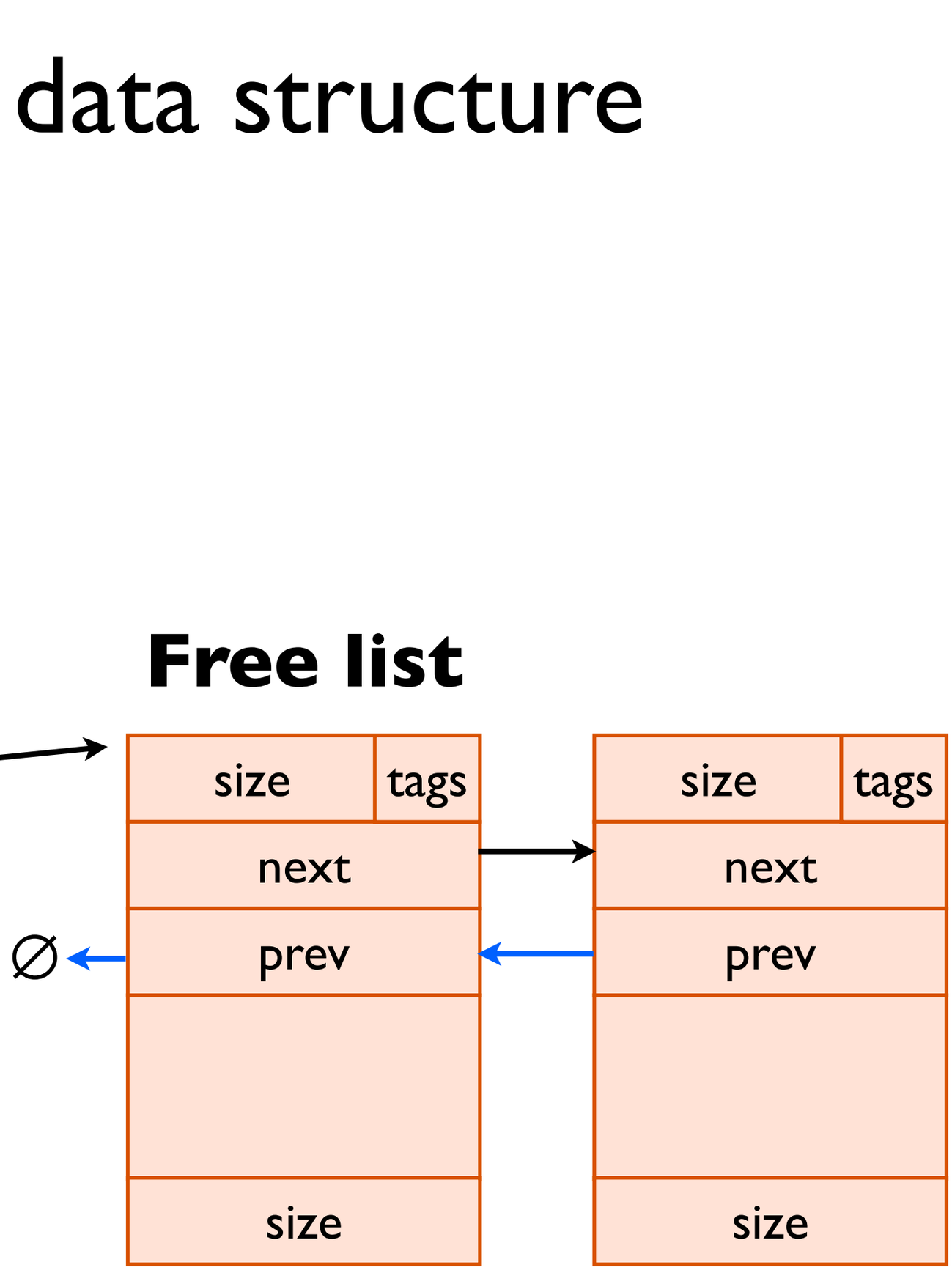
Lab 5: Block data structure

mm.c

```
struct BlockInfo {
  size_t sizeAndTags;
  struct BlockInfo *next;
  struct BlockInfo *prev;
};

// global variable
FREE_LIST_HEAD
```

Free list



Lab 5: Source code overview

We give you

```
void* searchFreeList(int reqSize)

// Manipulate the free list (assumes freeBlock is on the free list)
void insertFreeBlock(BlockInfo *freeBlock)
void removeFreeBlock(BlockInfo *freeBlock)
void coalesceFreeBlock(BlockInfo *freeBlock)

// Get more memory from the OS and add it to the free list
void requestMoreSpace(size_t reqSize)
```

You implement

```
void* mm_malloc(size_t size)
void mm_free(void *ptr)
```

Lab 5: Hints

- You will have to do some pointer arithmetic
  ... use: `POINTER_ADD(), POINTER_SUB()`

- You don't need any more global variables
- You don't need to call any functions in memlib.h
- We did a lot of the tedious linked-list manipulation for you
  ... use it!
- Do the simple thing first
  ... our solution is ~75 lines total, including comments

- Debug with gdb
  ... use command: `print *(BlockInfo*)p`
  ... this allows you to view pointer p through the "lens"
  of a BlockInfo, i.e., it pretends that p points to a BlockInfo

Today

- ~~Lab 5~~
- Reference counting

Reference Counting

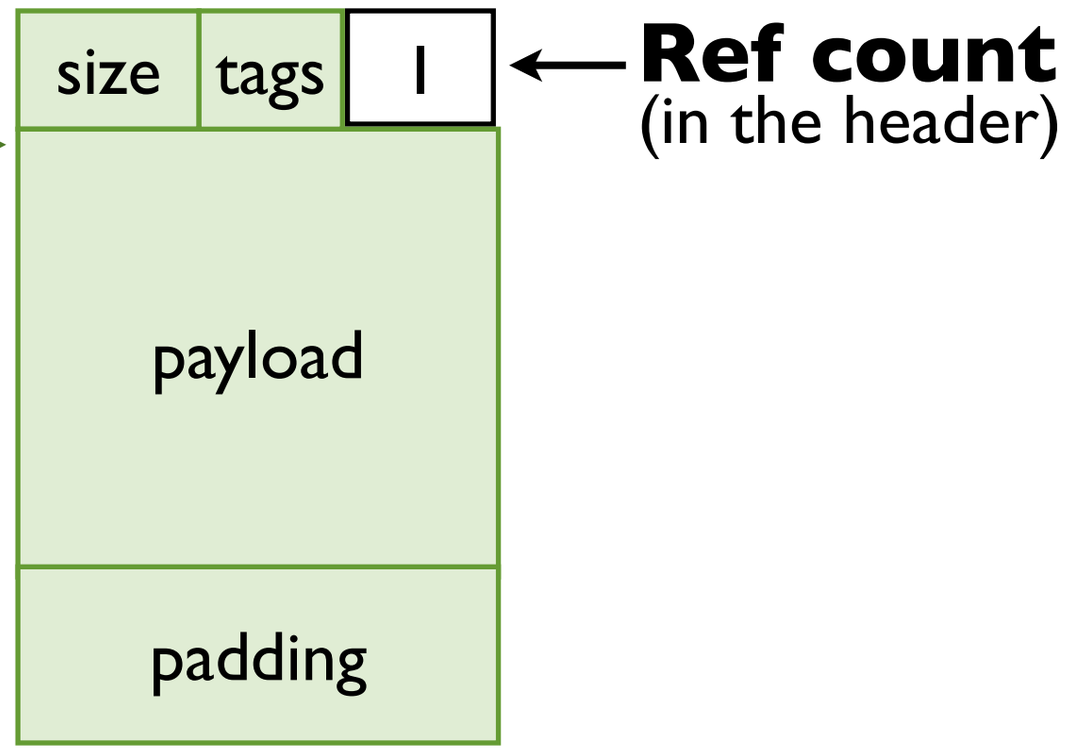
Basic idea: count the # of pointers to each object

Example

```
void foo() {
   int *x = malloc(4);

}
```

Heap



Reference Counting

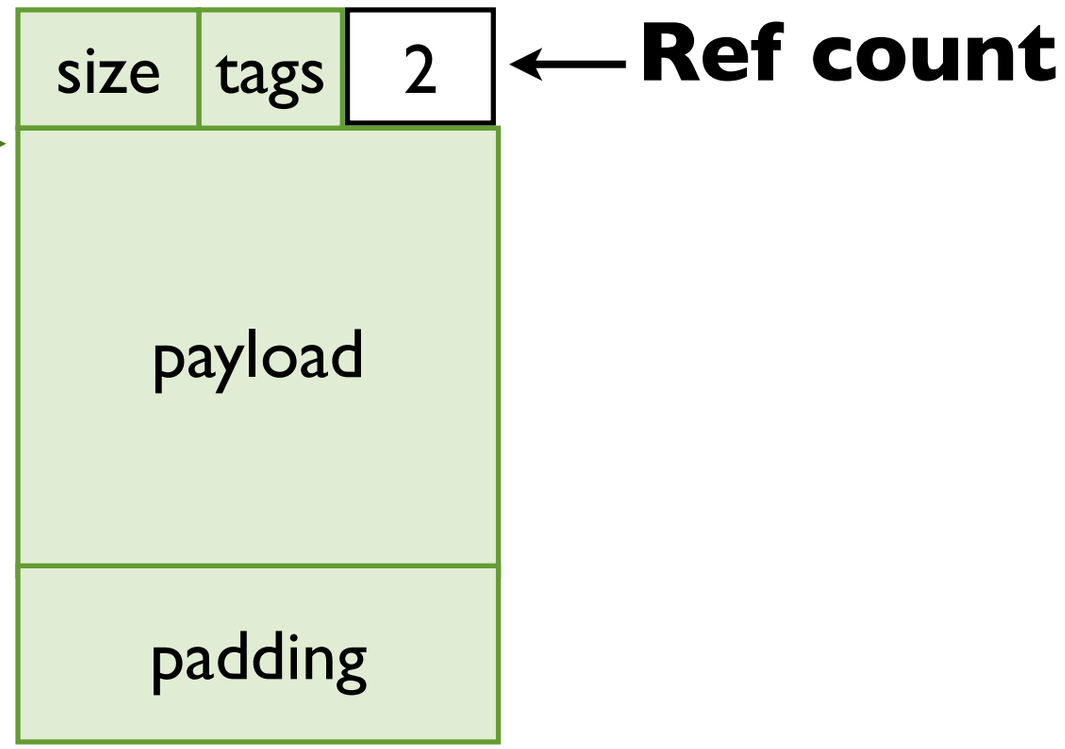
Basic idea: count the # of pointers to each object

Example

```
void foo() {
   int *x = malloc(4);
   int *p = x;
   ...

}
```

Heap



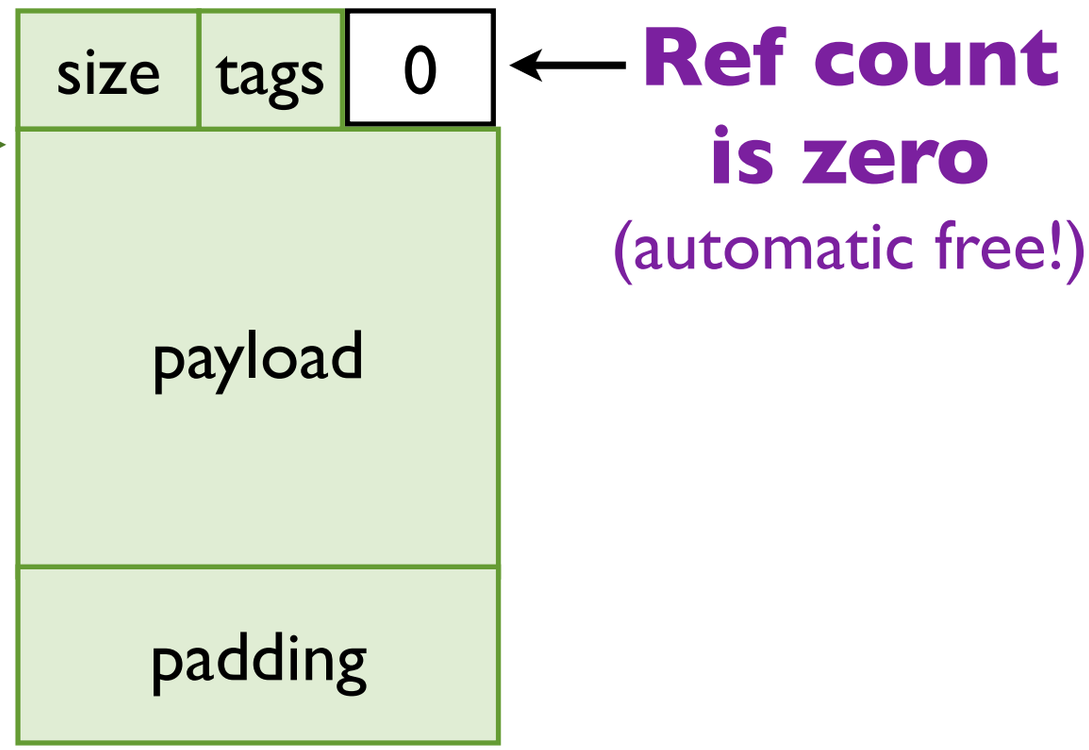
Reference Counting

When the count=0, free the object

Example

```
void foo() {
   int *x = malloc(4);
   int *p = x;
   ...
   // x and p go out-of-scope
}
```

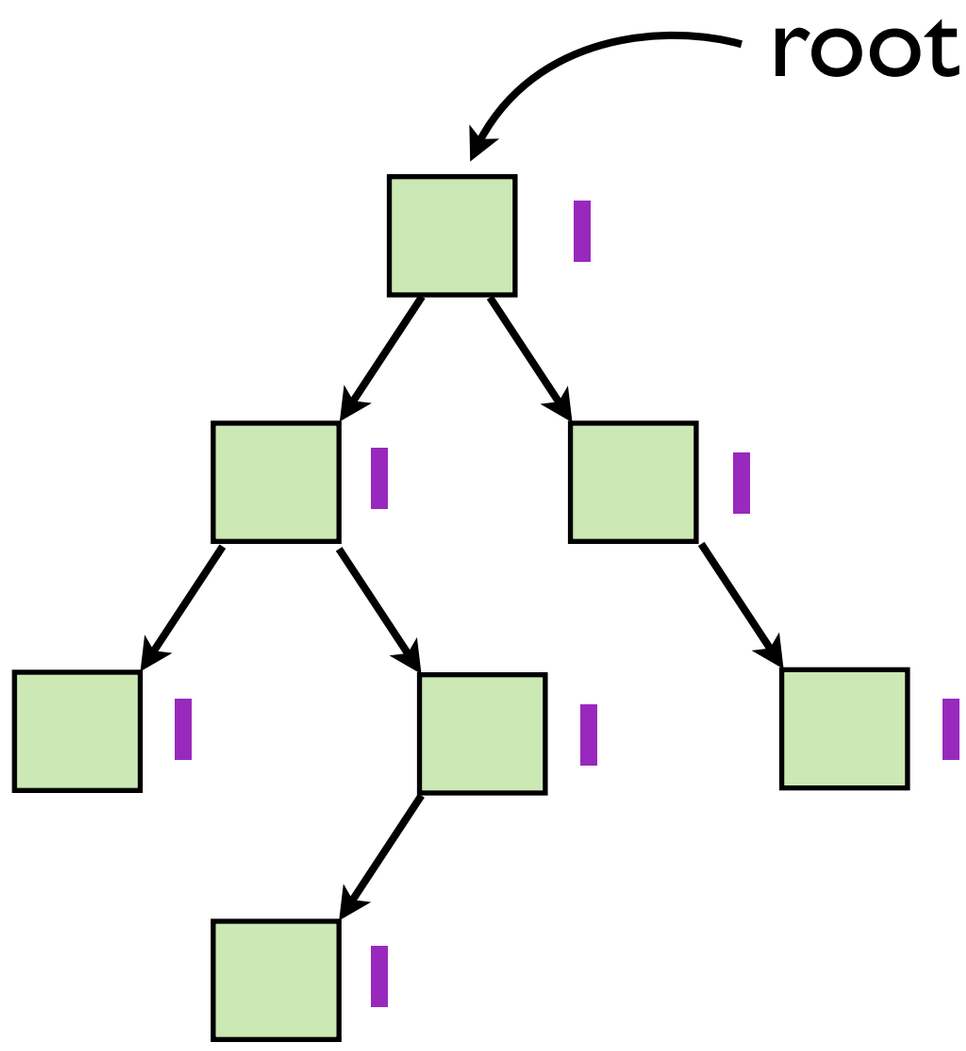
Heap



Ref count is zero
(automatic free!)

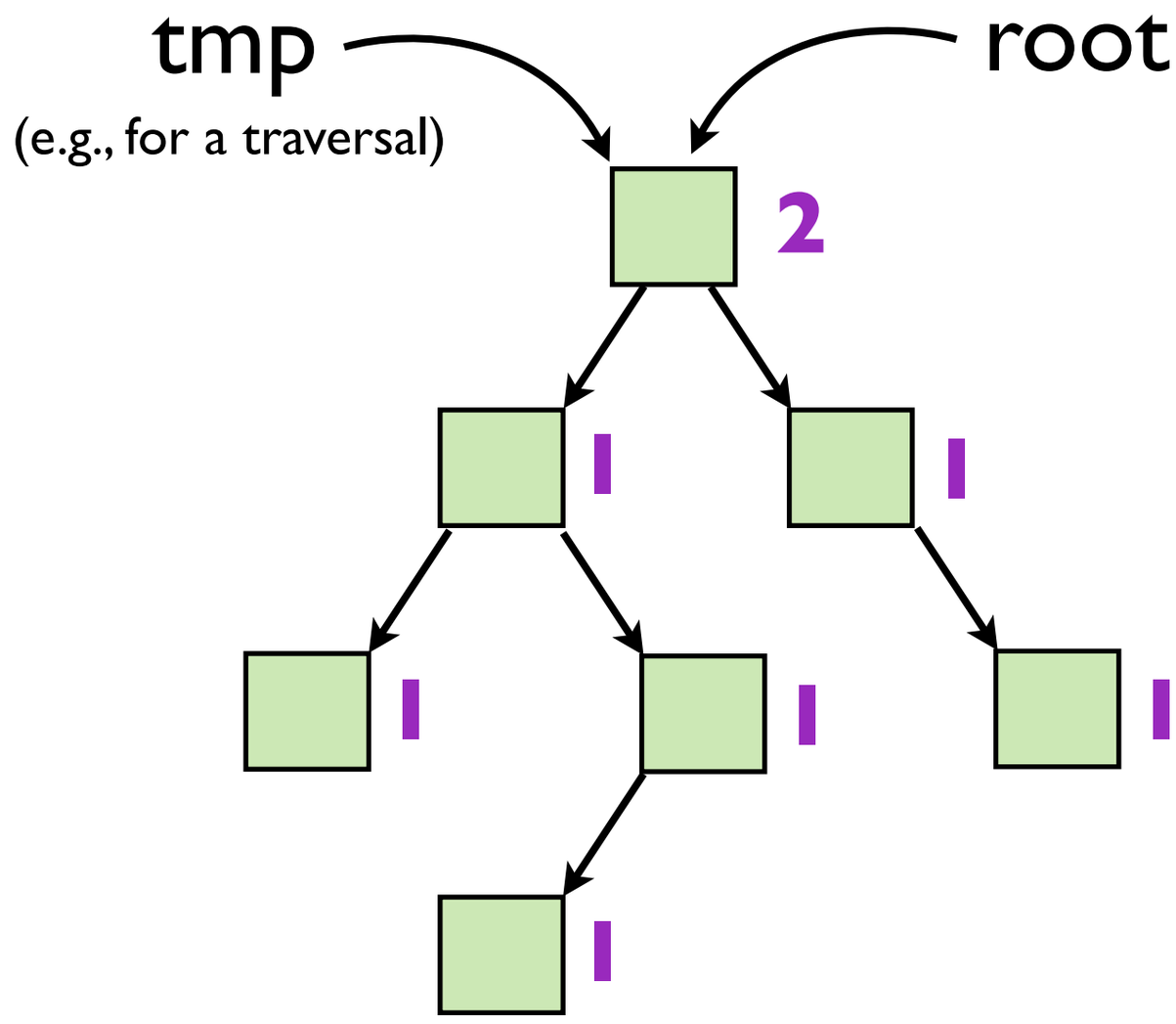
Reference Counting

Example: a tree
(ref counts in purple)



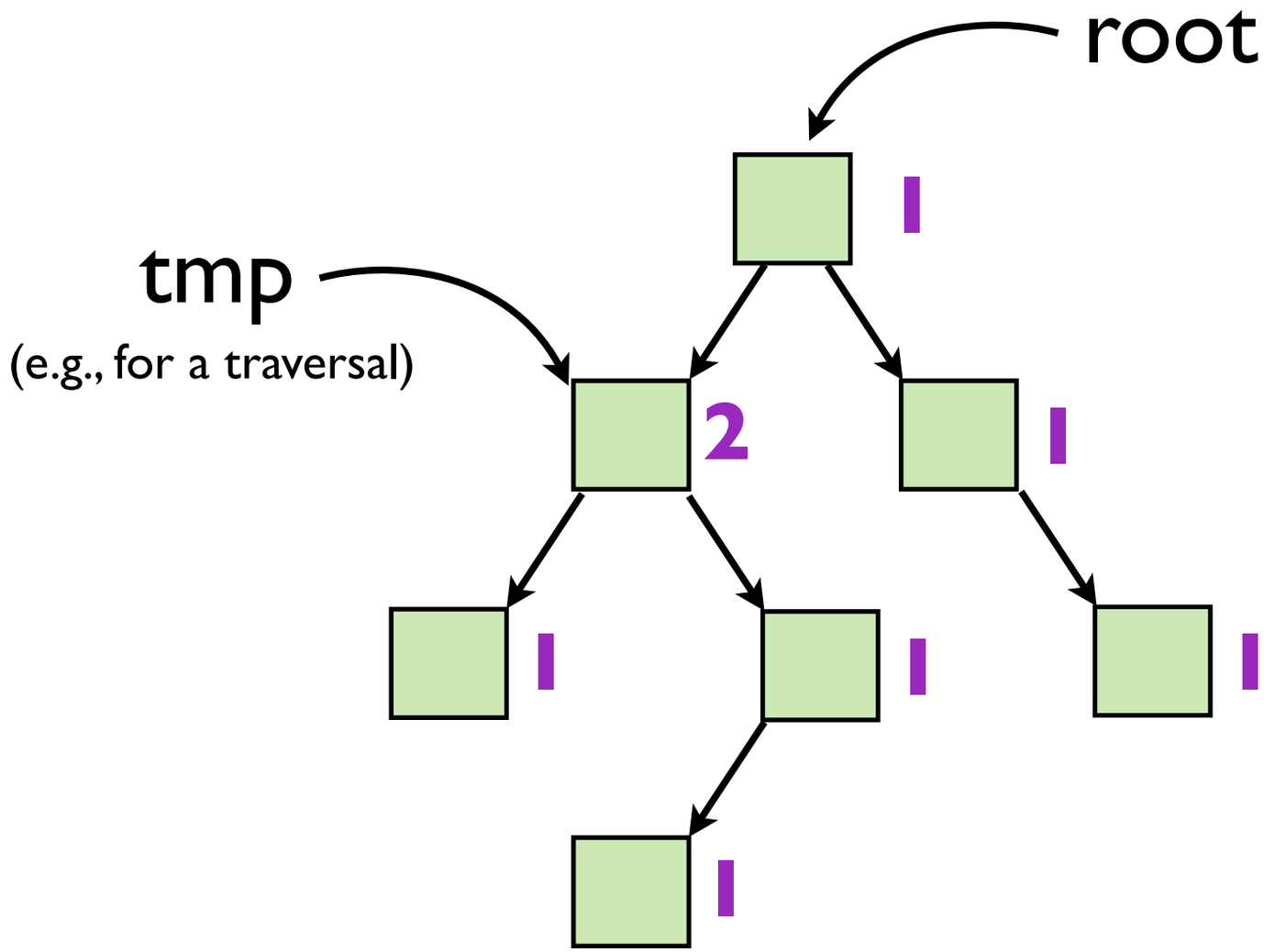
Reference Counting

Example: a tree
(ref counts in purple)



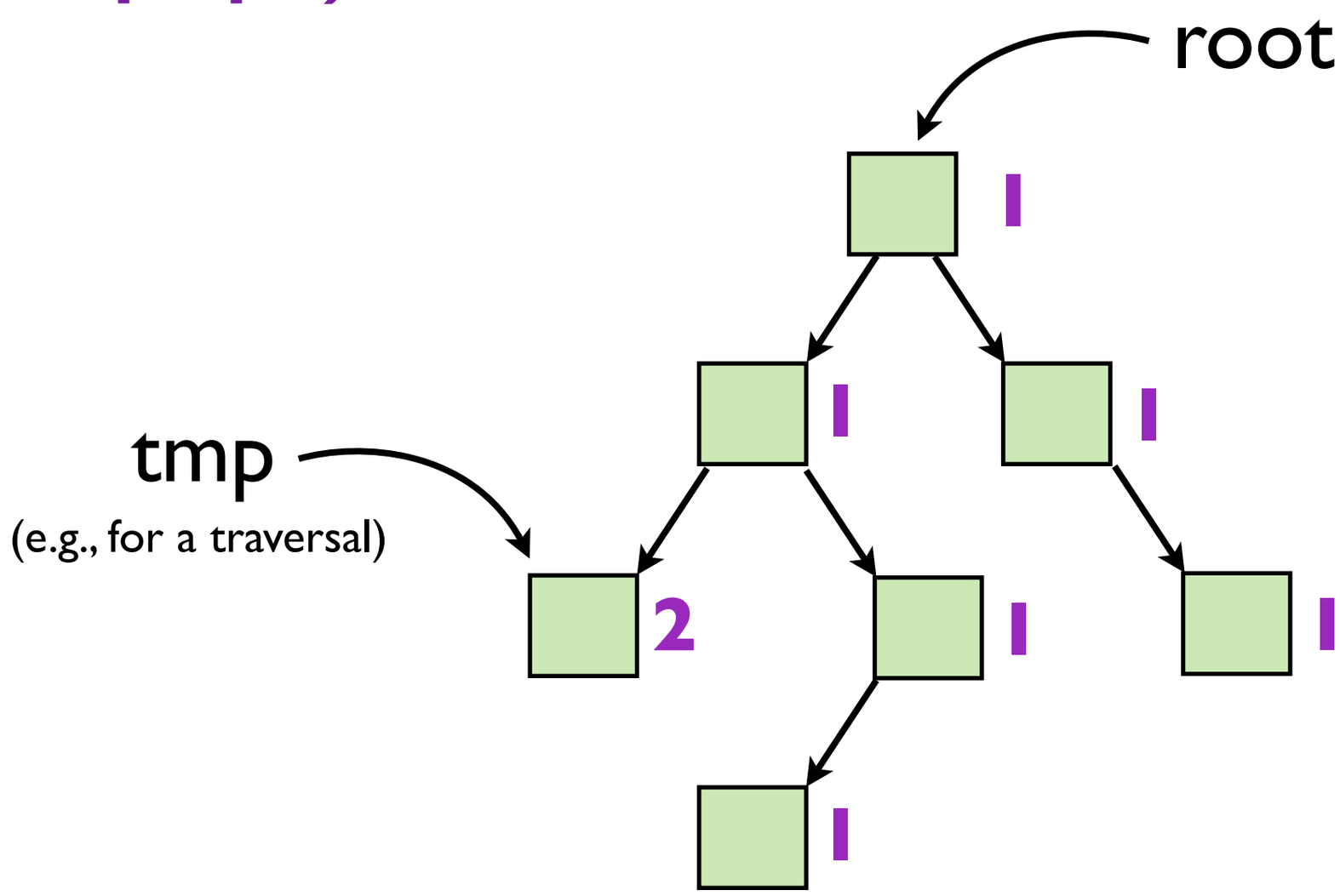
Reference Counting

Example: a tree
(ref counts in purple)



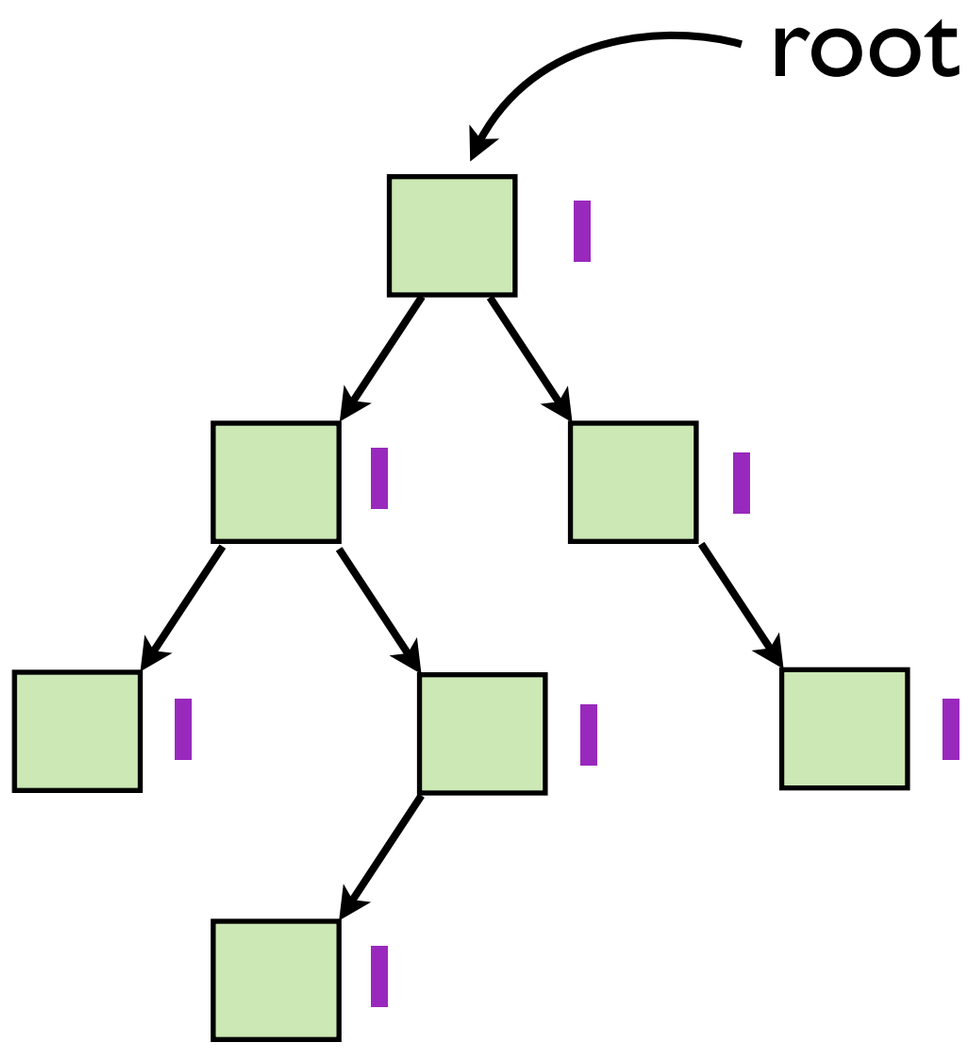
Reference Counting

Example: a tree
(ref counts in purple)



Reference Counting

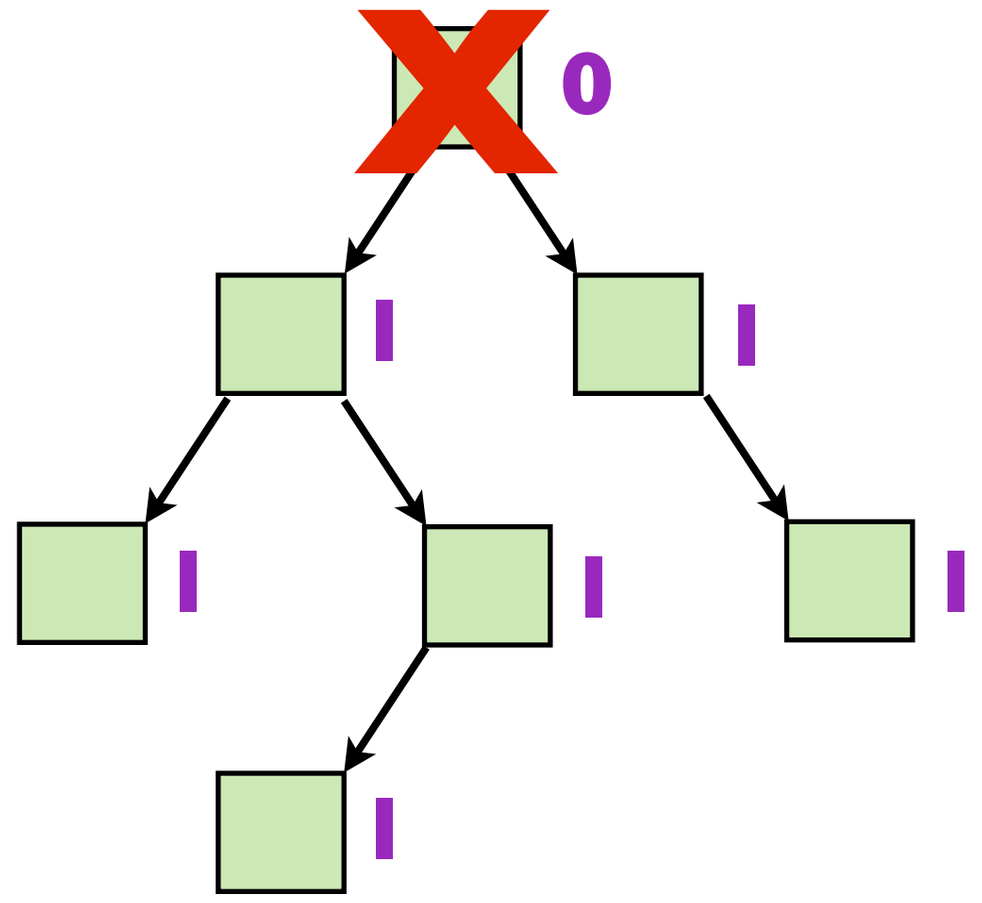
Example: a tree
(ref counts in purple)



Reference Counting

Example: a tree
(ref counts in purple)

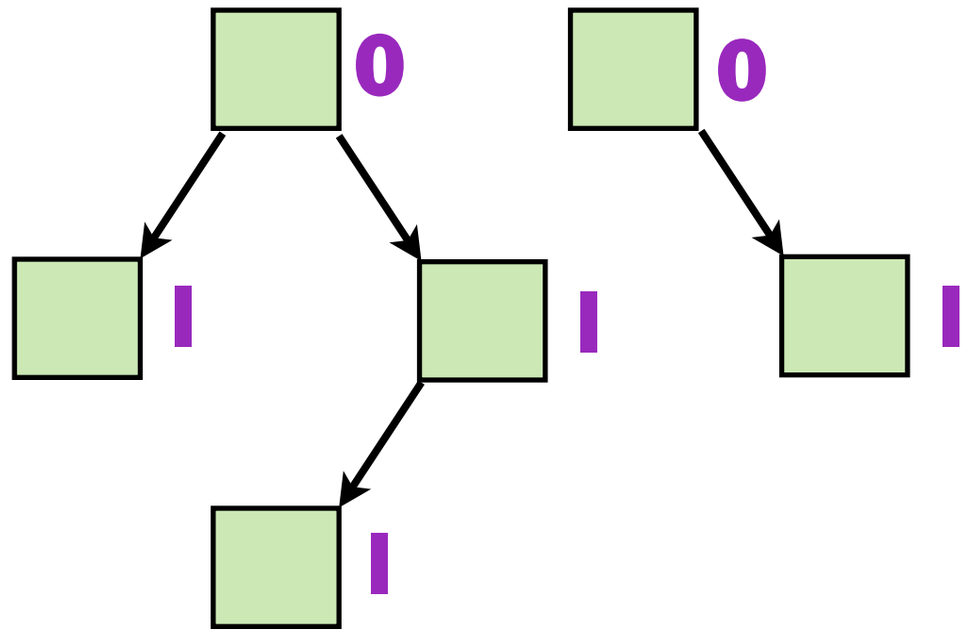
`root = NULL`



Reference Counting

Example: a tree
(ref counts in purple)

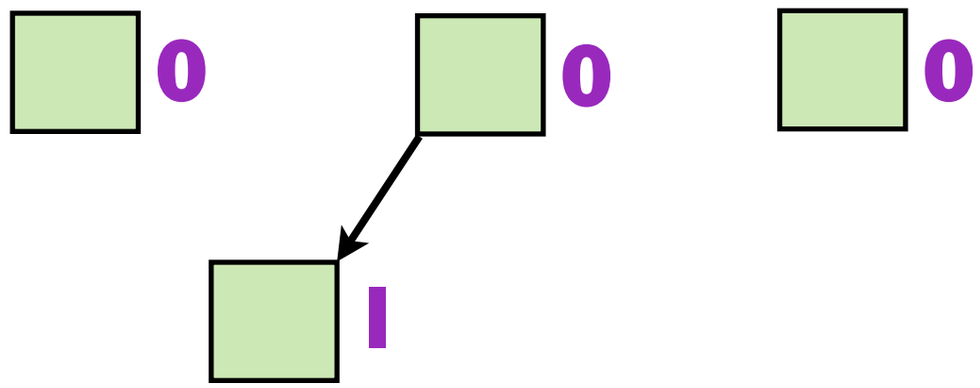
`root = NULL`



Reference Counting

Example: a tree
(ref counts in purple)

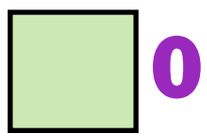
root = NULL



Reference Counting

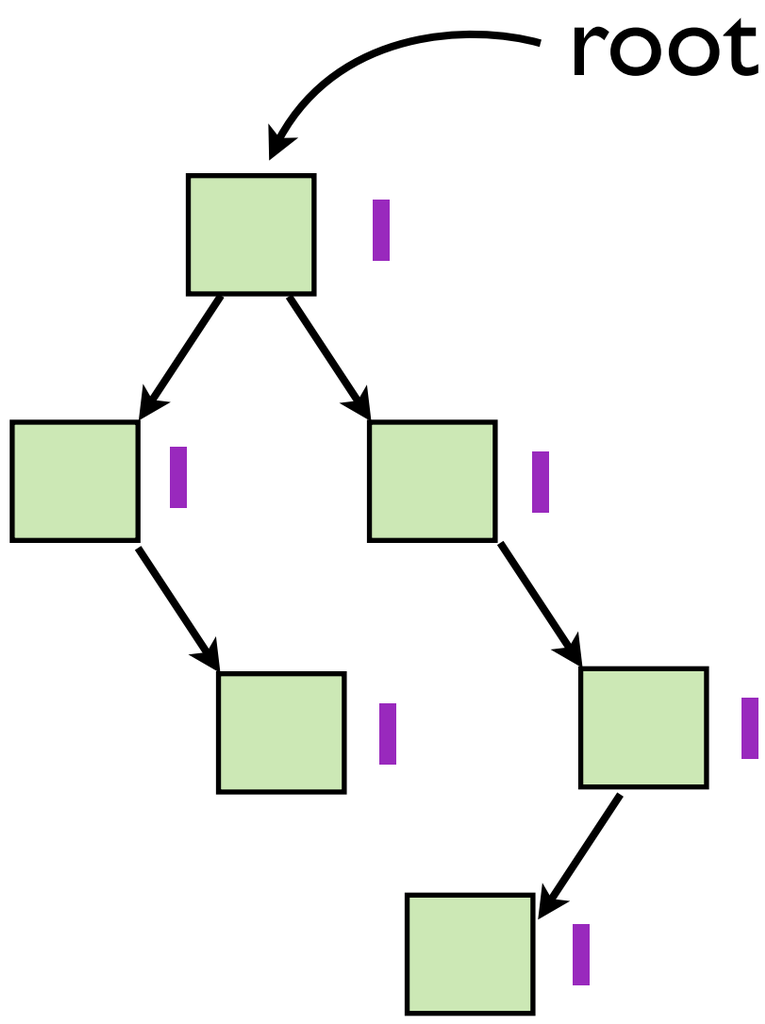
Example: a tree
(ref counts in purple)

root = NULL



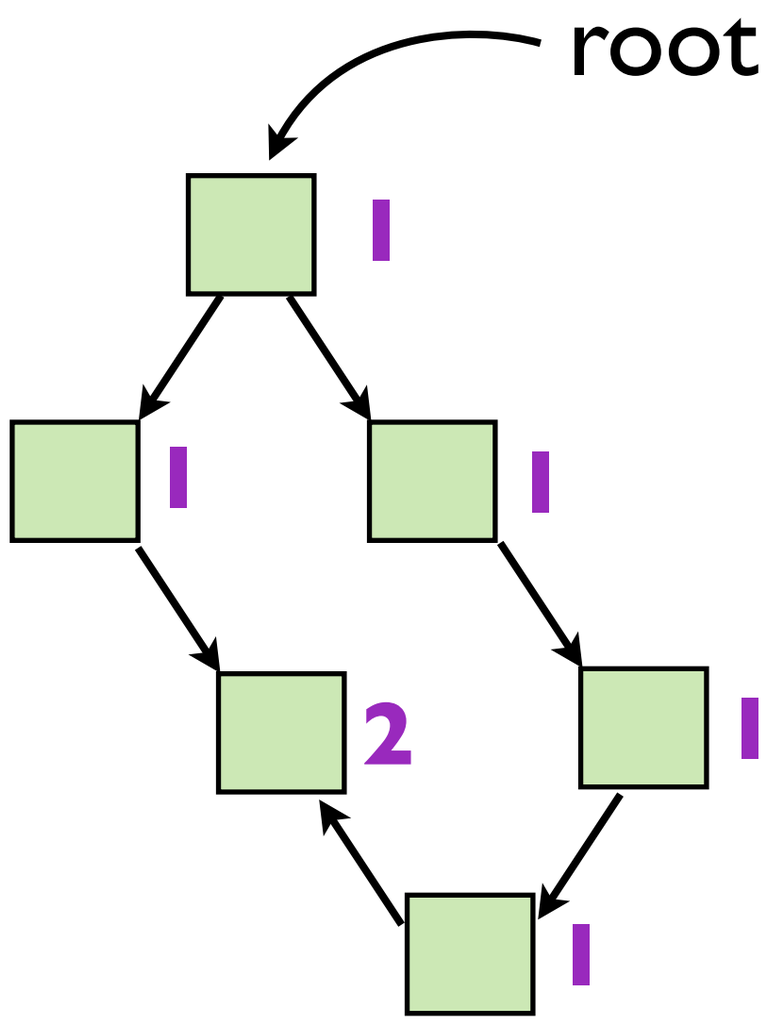
Reference Counting

Example: a directed graph
(ref counts in purple)



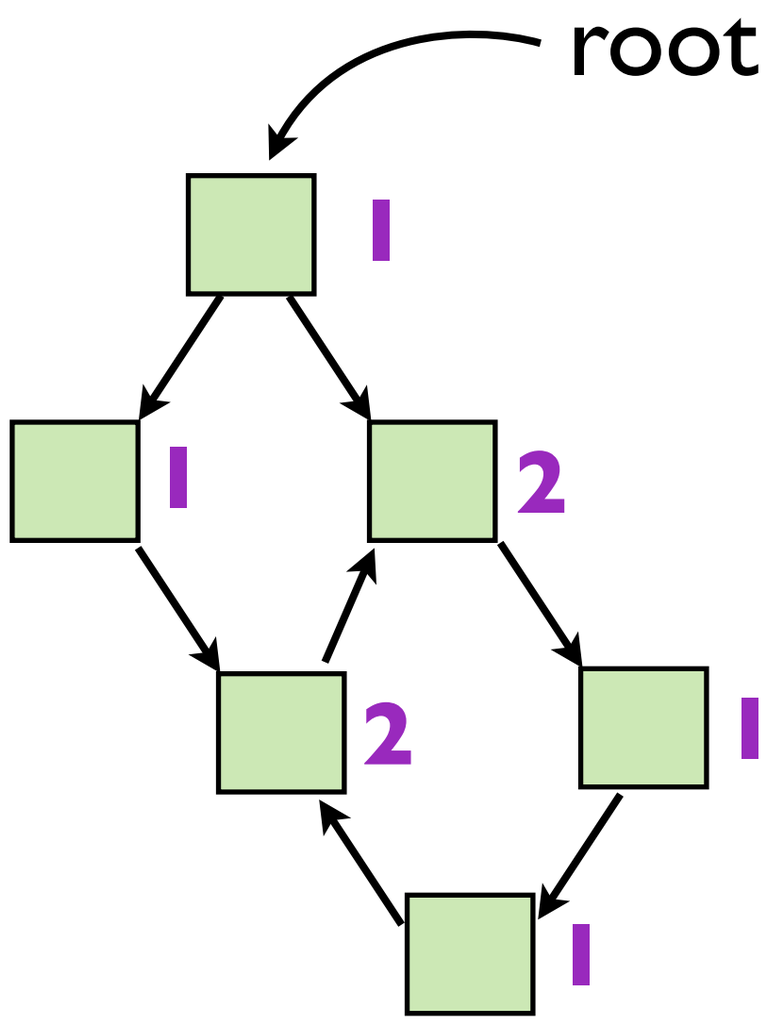
Reference Counting

Example: a directed graph
(ref counts in purple)



Reference Counting

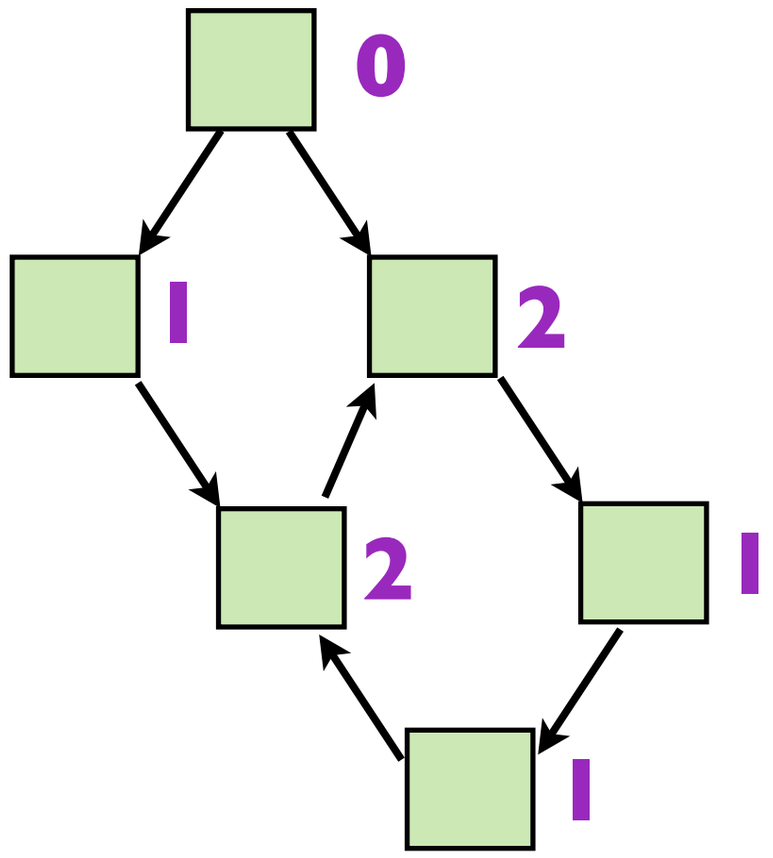
Example: a directed graph
(ref counts in purple)



Reference Counting

Example: a directed graph
(ref counts in purple)

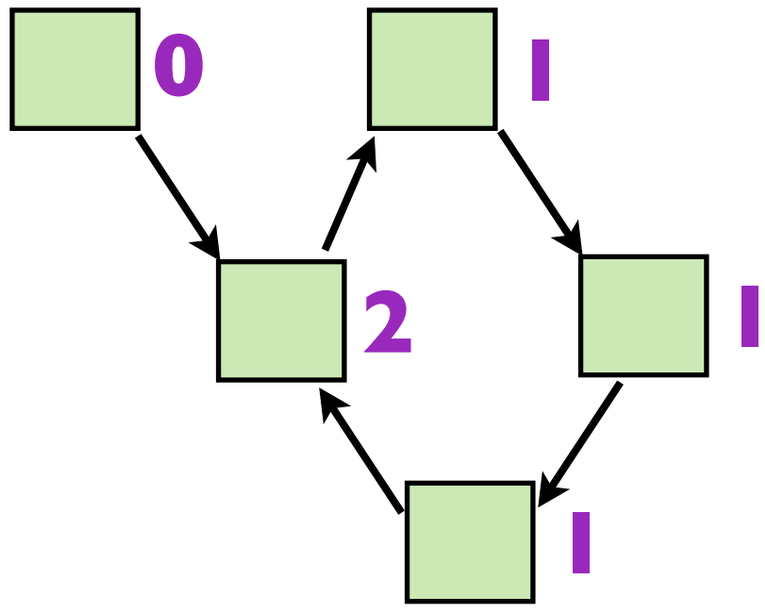
`root = NULL`



Reference Counting

Example: a directed graph
(ref counts in purple)

root = NULL



Reference Counting

Example: a directed graph
(ref counts in purple)

root = NULL

Reference counting cannot garbage collect cycles!