# EECS 280
# Discussion #11

Week of March 24

# Outline

- **Administrivia**
- Dynamic Memory Management
- Dynamic Memory Mistakes

# Administrivia

- Project 4
  - Due March 25 @ 11:59 PM
- Project 5
  - Out this week
  - Due last week of classes

# Outline

- Administrivia
- **Dynamic Memory Management**
  - Motivation
  - How To (`new` and `delete`)
  - Dynamic Memory Illustrated
- Dynamic Memory Mistakes

# Dynamic Memory - Motivation

# Dynamic Memory - Motivation

- Till now we've been trying to predict the amount of memory we would need to perform a task
  - Think `cards` and `squares` in Monopoly

# Dynamic Memory - Motivation

- Till now we've been trying to predict the amount of memory we would need to perform a task
  - Think `cards` and `squares` in Monopoly
- Problems
  - Often wasteful - 5 square Monopoly board
  - "Growing" data - strings, lists, trees, recursive calls

# Dynamic Memory - Motivation

- Till now we've been trying to predict the amount of memory we would need to perform a task
  - Think `cards` and `squares` in Monopoly
- Problems
  - Often wasteful - 5 square Monopoly board
  - "Growing" data - strings, lists, trees, recursive calls
- Solution: get memory as we need it!

# Using Dynamic Memory

- Basic Process
  - Declare a structure to track your memory
  - Ask for the memory (note address)
  - Use the memory (indirectly via pointers)
  - Give the memory back when done

# 1. Declare a Pointer

```
// single variable allocation
int *my_integer;

// array allocation
int *my_array;
```

# 2. Ask for Memory

- Requests for memory are made using the `new` operator
  - By default, `new` throws an exception on failure

```
// single variable allocation
int *my_integer = new int;

// array allocation
int *my_array = new int[10];

// dynamic array allocation
int z = 5;
int *my_dyn_array = new int[z];
```

# 3. Use the Memory

- Remember to dereference the pointer as appropriate

```
// single variable allocation
int *my_integer = new int;
(*my_integer) = 5;
cout << (*my_integer);

// array allocation
int *my_array = new int[10];
my_array[2] = 7;
cout << *(my_array + 2);
```

# 3. Use the Memory

- Remember to dereference the pointer as appropriate

```
// single variable allocation
int *my_integer = new int;
(*my_integer) = 5;
cout << (*my_integer);

// array allocation
int *my_array = new int[10];
my_array[2] = 7;
cout << *(my_array + 2);
```

5

# 3. Use the Memory

- Remember to dereference the pointer as appropriate

```
// single variable allocation
int *my_integer = new int;
(*my_integer) = 5;
cout << (*my_integer);

// array allocation
int *my_array = new int[10];
my_array[2] = 7;
cout << *(my_array + 2);
```

5

7

# 4. Release the Memory

- Release of dynamically allocated memory is achieved using the `delete` operator

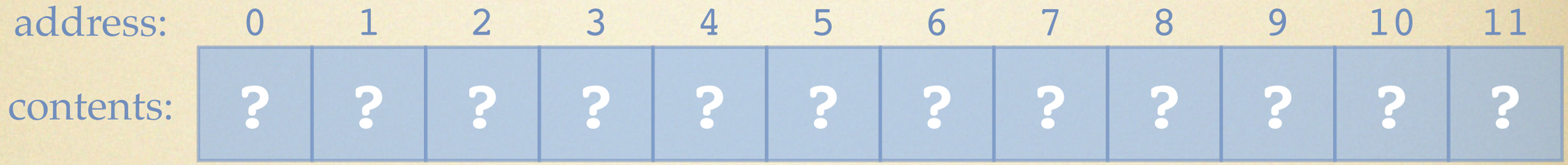
```
// single variable allocation
int *my_integer = new int;
(*my_integer) = 5;
cout << (*my_integer);
delete my_integer;

// array allocation
int *my_array = new int[10];
my_array[2] = 7;
cout << *(my_array + 2);
delete [] my_array;
```

# A Technical Aside

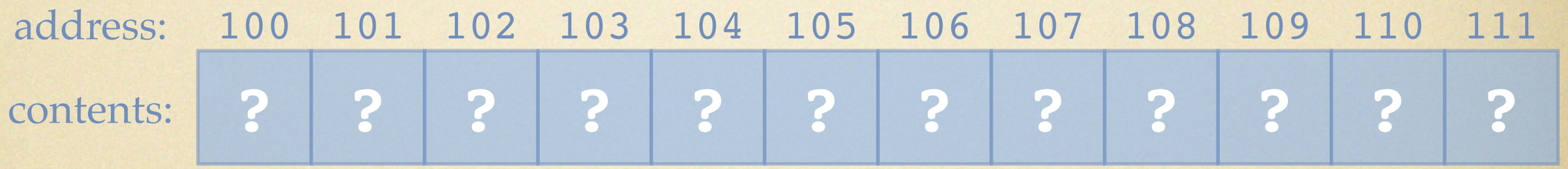
- Static variables and dynamic variables are stored in different parts of memory, in very different data structures
  - Static local variables are allocated on the system stack (with their associated functions)
  - Dynamic variables are allocated from a separate pool of memory, known as a heap

# Dynamic Memory Illustrated

**Stack**

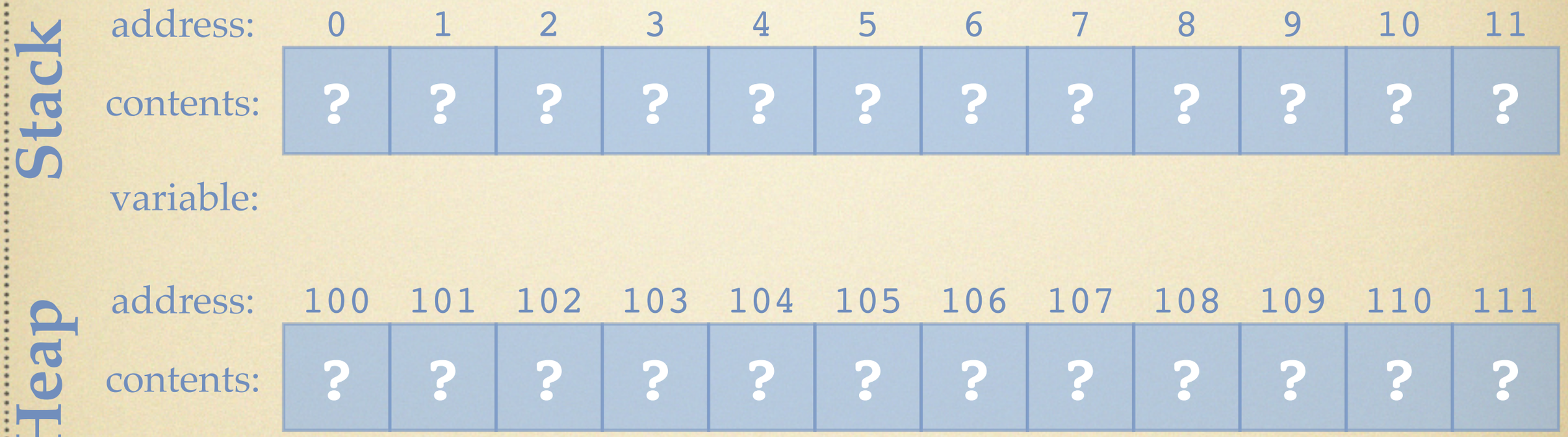



**Heap**




- For purposes of this illustration, we ignore the organizational differences between a stack and a heap

# 1. Declare a Pointer




```
// single variable allocation
int *p;

// array allocation
int *a;
```

# 1. Declare a Pointer

**Stack**

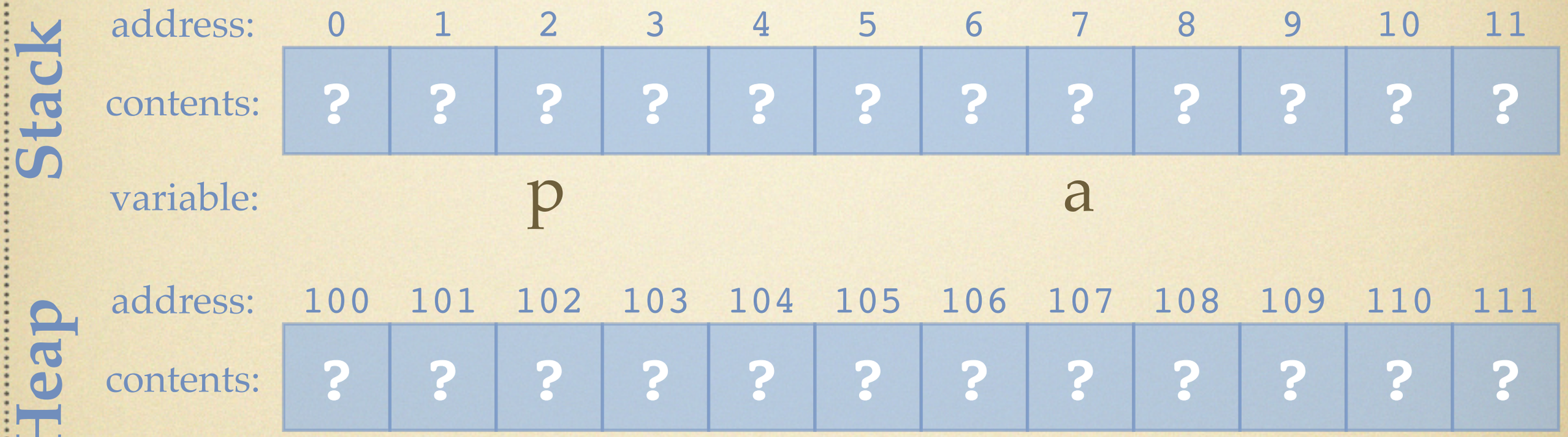
| address: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| contents: | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? |
| variable: | | | p | | | | | | | | | |

**Heap**

| address: | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 110 | 111 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| contents: | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? |
| allocated: | | | | | | | | | | | | |

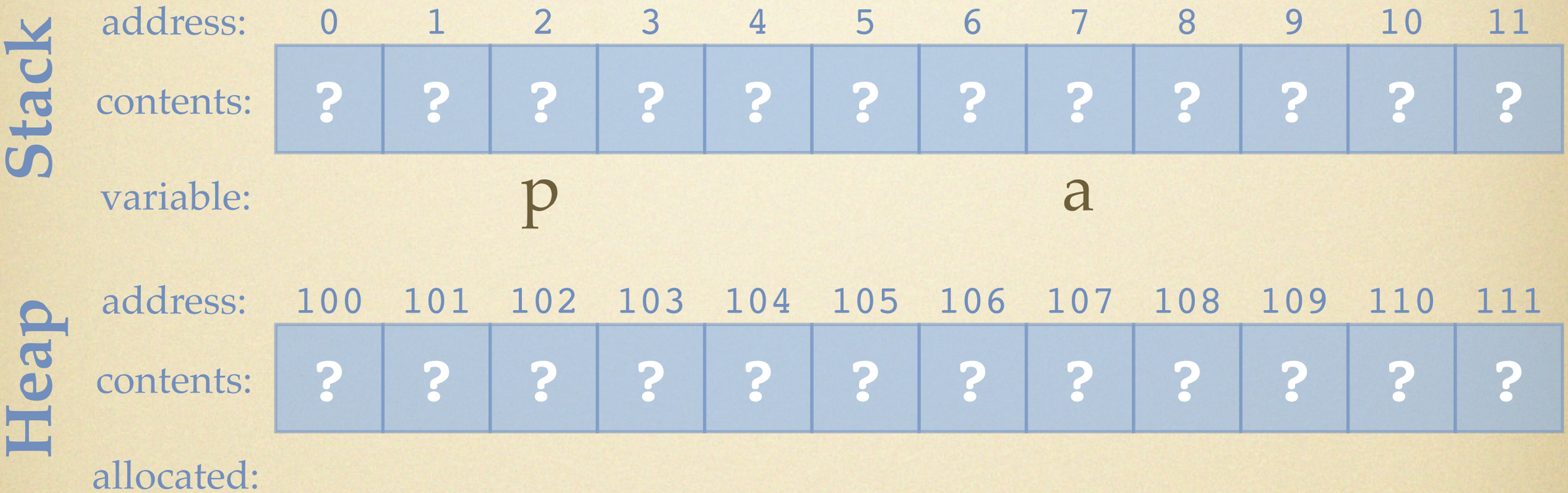
```
// single variable allocation
int *p;

// array allocation
int *a;
```

# 1. Declare a Pointer



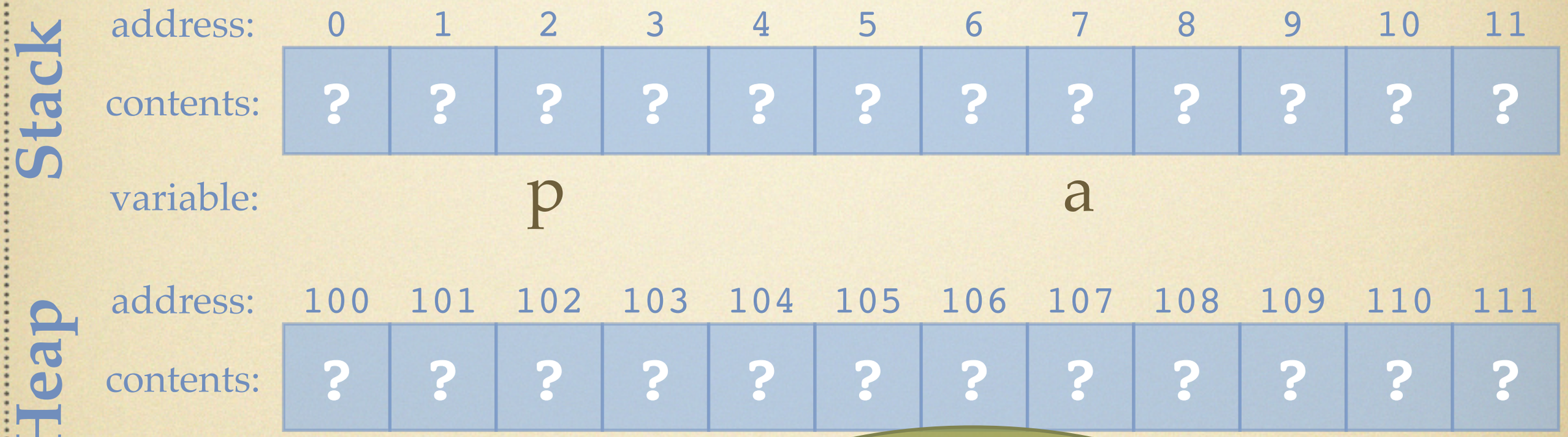

```
// single variable allocation
int *p;

// array allocation
int *a;
```

# 2. Ask for Memory




```
// single variable allocation
int *p = new int;

// array allocation
int *a = new int[10];
```

# 2. Ask for Memory




You can have 100

```
// single variable allocation
int *p = new int;

// array allocation
int *a = new int[10];
```

# 2. Ask for Memory

**Stack**

| address: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| contents: | ? | ? | 100 | ? | ? | ? | ? | ? | ? | ? | ? | ? |
| variable: | | | p | | | | | a | | | | |

**Heap**

| address: | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 110 | 111 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| contents: | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? |
| allocated: | x | | | | | | | | | | | |

You can have 100

```
// single variable allocation
int *p = new int;

// array allocation
int *a = new int[10];
```

# 2. Ask for Memory

**Stack**

| address: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| contents: | ? | ? | 100 | ? | ? | ? | ? | ? | ? | ? | ? | ? |
| variable: | | | p | | | | | a | | | | |

**Heap**

| address: | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 110 | 111 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| contents: | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? |
| allocated: | x | | | | | | | | | | | |

```
// single variable allocation
int *p = new int;

// array allocation
int *a = new int[10];
```

You can have 101-110

# 2. Ask for Memory


```
// single variable allocation
int *p = new int;

// array allocation
int *a = new int[10];
```

You can have 101-110

# 3. Use the Memory


```
// single variable allocation
...
(*p) = 5;

// array allocation
...
a[2] = 7;
```

# 3. Use the Memory


```
// single variable allocation
...
(*p) = 5;

// array allocation
...
a[2] = 7;
```

# 3. Use the Memory

**Stack**

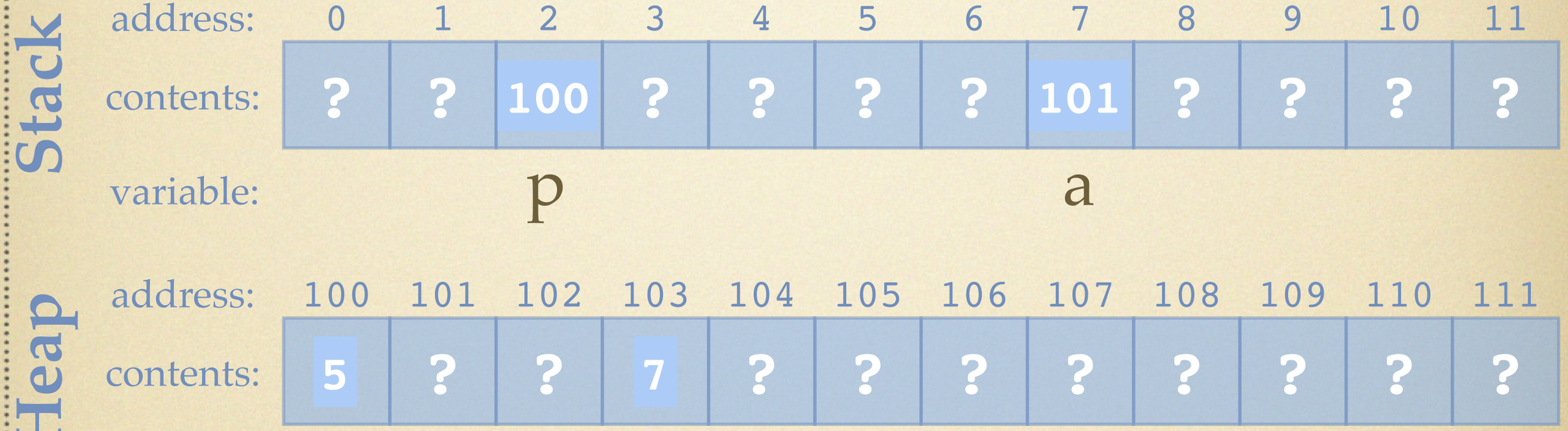
| address: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| contents: | ? | ? | 100 | ? | ? | ? | ? | 101 | ? | ? | ? | ? |
| variable: | | | p | | | | | a | | | | |

**Heap**

| address: | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 110 | 111 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| contents: | 5 | ? | ? | 7 | ? | ? | ? | ? | ? | ? | ? | ? |
| allocated: | x | x | x | x | x | x | x | x | x | x | x | |

```
// single variable allocation
...
(*p) = 5;

// array allocation
...
a[2] = 7;
```

# 4. Release the Memory




```
// single variable allocation
...
delete p;

// array allocation
...
delete [] a;
```

# 4. Release the Memory

**Stack**

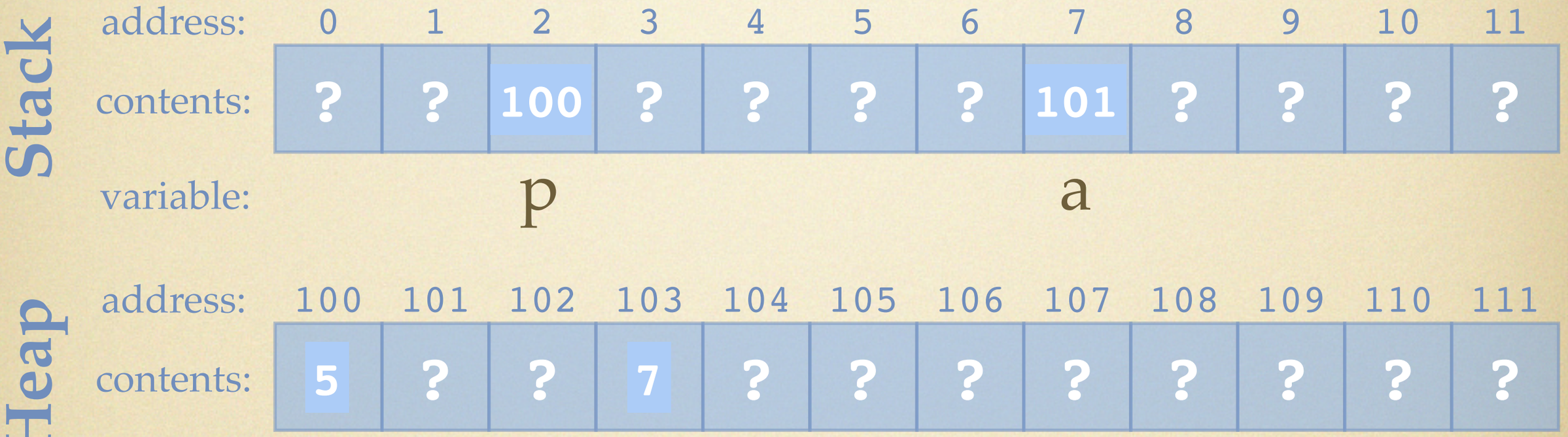
| address: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| contents: | ? | ? | 100 | ? | ? | ? | ? | 101 | ? | ? | ? | ? |
| variable: | | | p | | | | | a | | | | |

**Heap**

| address: | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 110 | 111 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| contents: | 5 | ? | ? | 7 | ? | ? | ? | ? | ? | ? | ? | ? |
| allocated: | | x | x | x | x | x | x | x | x | x | x | |

```
// single variable allocation
...
delete p;

// array allocation
...
delete [] a;
```

# 4. Release the Memory




```
// single variable allocation
...
delete p;

// array allocation
...
delete [] a;
```

# 4. Release the Memory

**Stack**

| address: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| contents: | ? | ? | 100 | ? | ? | ? | ? | 101 | ? | ? | ? | ? |
| variable: | | | p | | | | | a | | | | |

**Heap**

| address: | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 110 | 111 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| contents: | 5 | ? | ? | 7 | ? | ? | ? | ? | ? | ? | ? | ? |
| allocated: | | | | | | | | | | | | |

NOTE: The pointers retain their value AND the formerly allocated space retains its contents. However, you are no longer guaranteed ownership over the space in the heap.

# Outline

- Administrivia
- Dynamic Memory Management
- **Dynamic Memory Mistakes**
  - Mistakes
  - Consequences

# Dereferencing NULL

- 0 or NULL is a special address: it does not correspond to valid memory
  - Dereferencing NULL always causes a segmentation fault

```
int *p = NULL;
*p = 5;
```

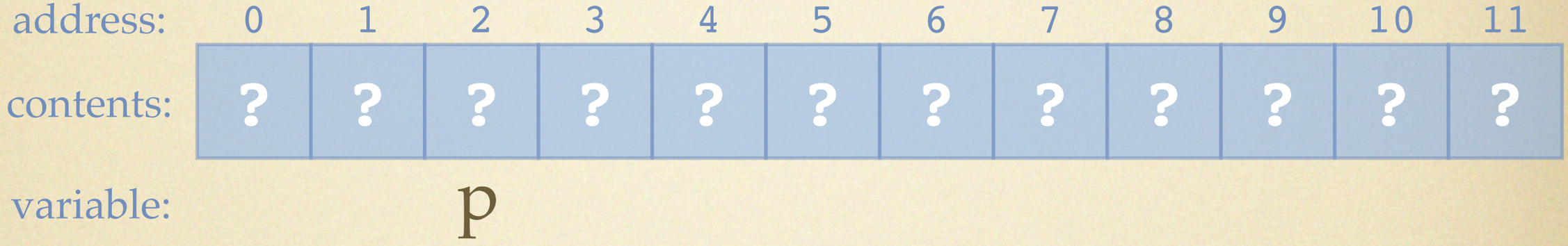
# Dereferencing NULL

- 0 or NULL is a special address: it does not correspond to valid memory
  - Dereferencing NULL always causes a segmentation fault

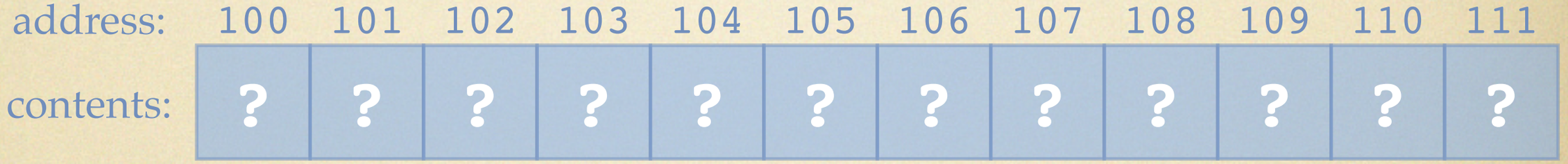
```
int *p = NULL;
*p = 5;
```

# Failing to Allocate Memory

**Stack**




**Heap**




```
int *p;
*p = 5;
```

# Failing to Allocate Memory

**Stack**

| address: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| contents: | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? |
| variable: | | | p | | | | | | | | | |

**Heap**

| address: | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 110 | 111 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| contents: | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? |
| allocated: | | | | | | | | | | | | |

```
int *p;
*p = 5;
```

# Deallocating Static Memory

Stack

| address: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| contents: | ? | ? | 5 | ? | ? | ? | ? | 100 | ? | ? | ? | ? |
| variable: | | | y | | | | | p | | | | |

Heap

| address: | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 110 | 111 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| contents: | 5 | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? |
| allocated: | x | | | | | | | | | | | |

```
int y = 5;
int *p = new int;
(*p) = y;
delete y;
```

# Deallocating Static Memory

**Stack**

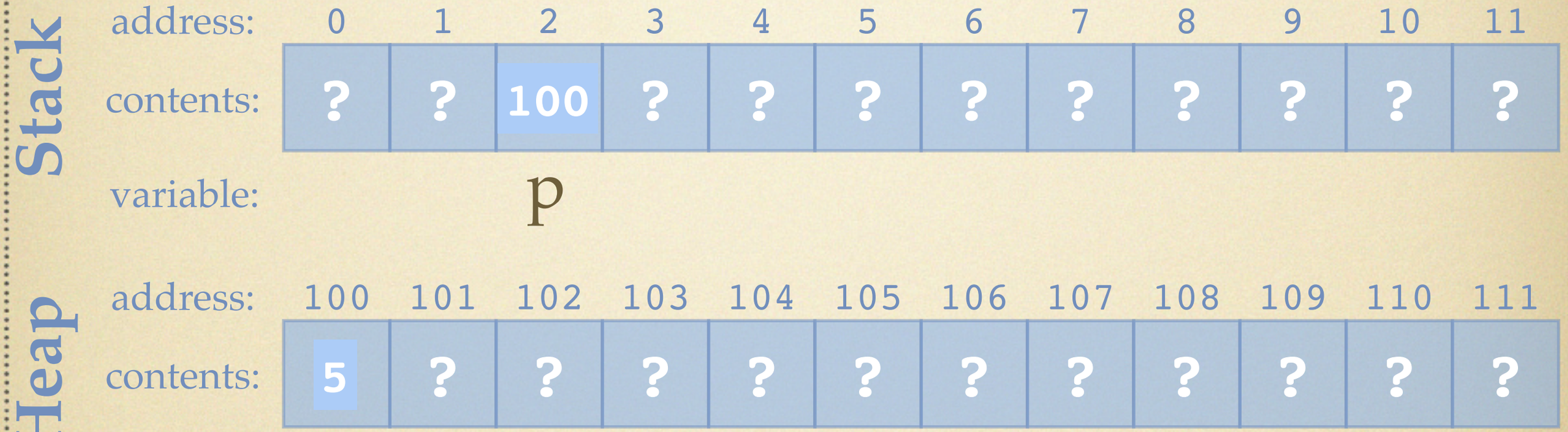
| address: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| contents: | ? | ? | 5 | ? | ? | ? | ? | 100 | ? | ? | ? | ? |
| variable: | | | y | | | | | p | | | | |

**Heap**

| address: | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 110 | 111 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| contents: | 5 | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? |
| allocated: | x | | | | | | | | | | | |

```
int y = 5;
int *p = new int;
(*p) = y;
delete y;
```

# Releasing Deallocated Memory




```
int *p = new int;
(*p) = 5;
delete p;
delete p;
```

# Releasing Deallocated Memory

**Stack**

| address: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| contents: | ? | ? | 100 | ? | ? | ? | ? | ? | ? | ? | ? | ? |
| variable: | | | p | | | | | | | | | |

**Heap**

| address: | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 110 | 111 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| contents: | 5 | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? |
| allocated: | | | | | | | | | | | | |

```
int *p = new int;
(*p) = 5;
delete p;
delete p;
```

# Releasing an Array with `delete`

**Stack**

| address: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| contents: | ? | ? | 100 | ? | ? | ? | ? | ? | ? | ? | ? | ? |
| variable: | | | a | | | | | | | | | |

**Heap**

| address: | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 110 | 111 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| contents: | 5 | 7 | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? |
| allocated: | x | x | | | | | | | | | | |

```
int *a = new int[2];
(*a) = 5;
a[1] = 7;
delete a;
```

# Releasing an Array with `delete`

**Stack**

| address: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| contents: | ? | ? | 100 | ? | ? | ? | ? | ? | ? | ? | ? | ? |
| variable: | | | a | | | | | | | | | |

**Heap**

| address: | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 110 | 111 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| contents: | 5 | 7 | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? |
| allocated: | x | x | | | | | | | | | | |

```
int *a = new int[2];
(*a) = 5;
a[1] = 7;
delete a;
```

# Allocating Space Unnecessarily


```
int *p = new int;
(*p) = 5;
p = new int;
```

# Allocating Space Unnecessarily


```
int *p = new int;
(*p) = 5;
p = new int;
```

Memory Leak!

# Allocating Space Unnecessarily

**Stack**

| address: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| contents: | ? | ? | 100 | ? | ? | ? | ? | ? | ? | ? | ? | ? |
| variable: | | | p | | | | | | | | | |

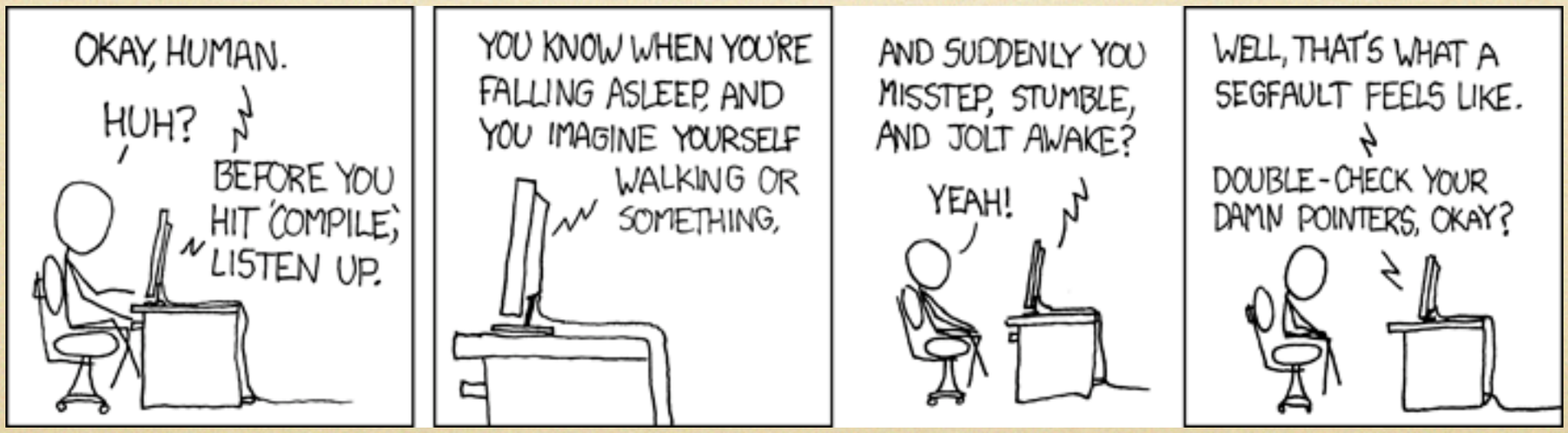
**Heap**

| address: | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 110 | 111 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| contents: | 5 | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? |
| allocated: | x | | | | | | | | | | | |

```
int *p = new int;
(*p) = 5;
p = new int;
```

# Consequences

- Usually a dynamic memory bug will result in the follow symptoms
  - Values changing mysteriously
  - Segmentation faults



# Final Thoughts

- We have discussed the idea of dynamic memory today
  - We addressed the issue of allocating exactly as much memory as needed (dynamic array)
  - How would you use dynamic memory allocation to accommodate a “growing” data structure?
- Lastly: support your GSI’s :)