CSE 351: The Hw/Sw Interface

Tom Bergan, TA
Week #1

Why take 351?

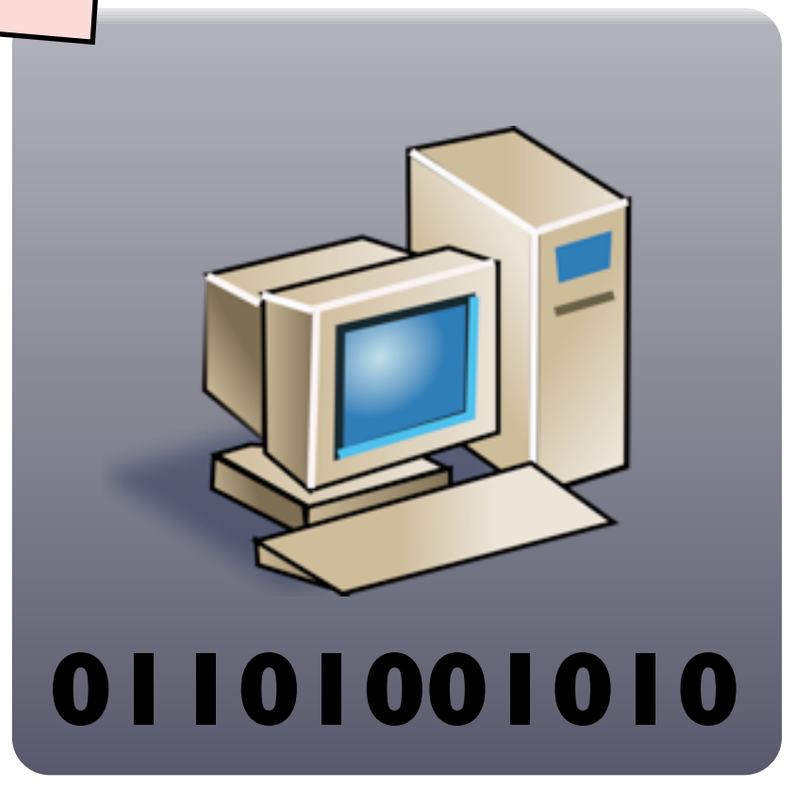
- It's required ☺

- My pitch:
  - This will (hopefully!) be an eye-opening look "under the hood"

Java

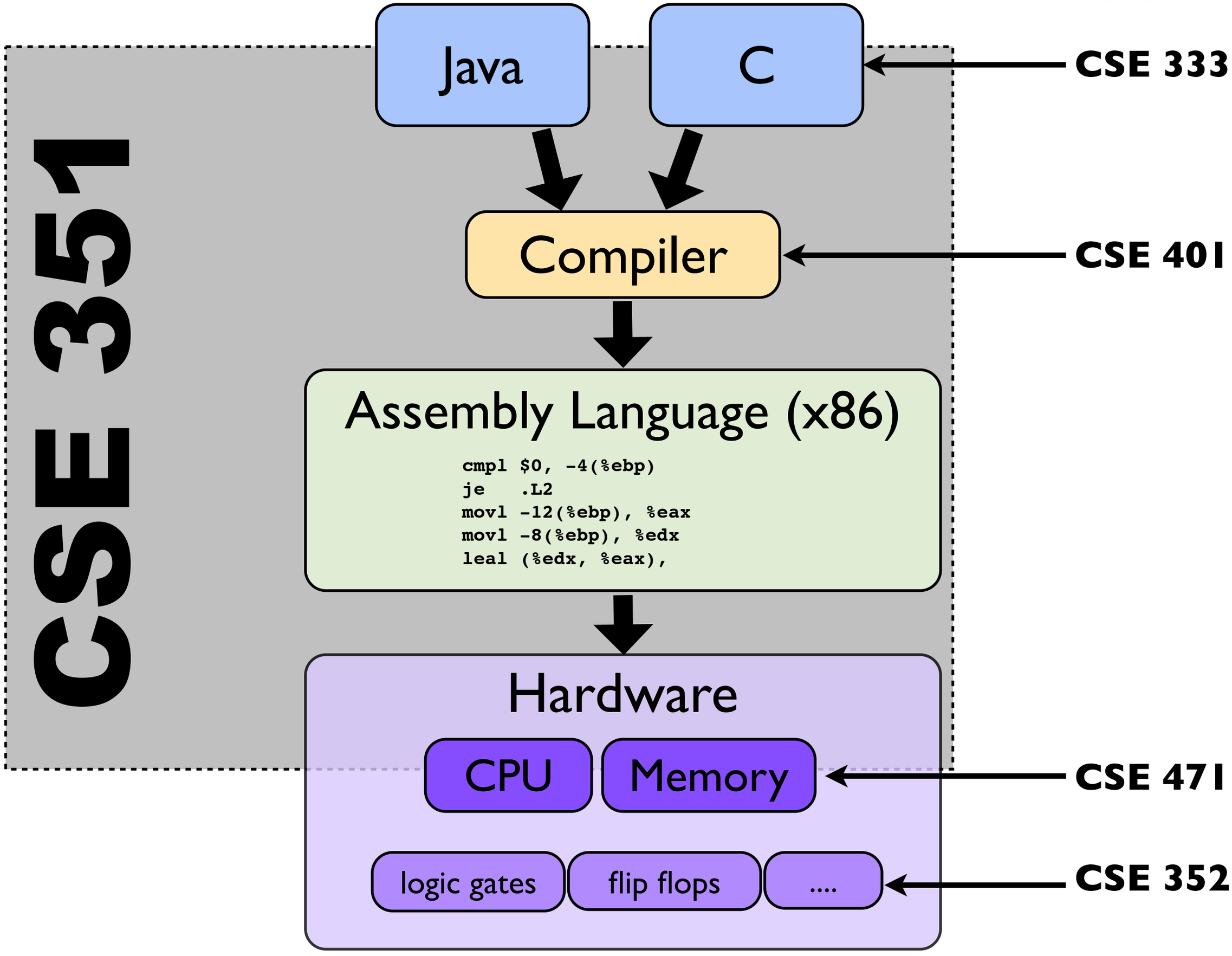
```
void MatrixMultiply(int[][] A, int[][] B)
{
  int[][] Result = new double[8][8];
  for (int i = 0; i < 8; i++)
    for (int j = 0; j < 8; j++)
      for (int k = 0; k < 8; k++)
        Result[i][j] += A[i][k] * B[k][j];
}
```

?



- How are numbers represented?
- How are data structures represented?
- How is memory allocated?
- What happens on a function call?
- ...

For more details:



What is this section for?

- Labs
- Questions!
  - please bring questions!
- Some extensions of the lectures / textbook
- Other resources:
  - discussion board (see course webpage)
  - office hours

Today

- ~~Introduction~~
- C overview
- Lab 1 quickstart
  - how to get started
  - how to compile and debug C code

Why learn C?

- For this class:
  - assignments are in C
  - C is very close to assembly language
- For yourself:
  - C code is everywhere

Hello, world!

Java

```
class HelloWorld {
    public static void main(String[] args) {
      System.out.println(“Hello, world!”);
    }
}
```

C

```
#include <stdio.h>

int main(int argc, char *argv[]) {
  printf(“Hello, world!\n”);
  return 0;
}
```

C vs Java

Java		C
`import java.xyz;`	Packages	`#include “xyz.h”`
`class Point {`	Classes	Structs - all members public `struct Point {`
`public int x;`		`int x;`
`public int y;`		`int y;`
		`};`
`public int foo(int a) {`	Methods	Functions `int foo(int a) {`
`while (x < y)`		`while (x < y)`
`...`		`...`
`return 42;`		`return 42;`
`}`		`}`

Similar syntax:
`if/then/else, while/for, switch/case, return`

C: three common confusions

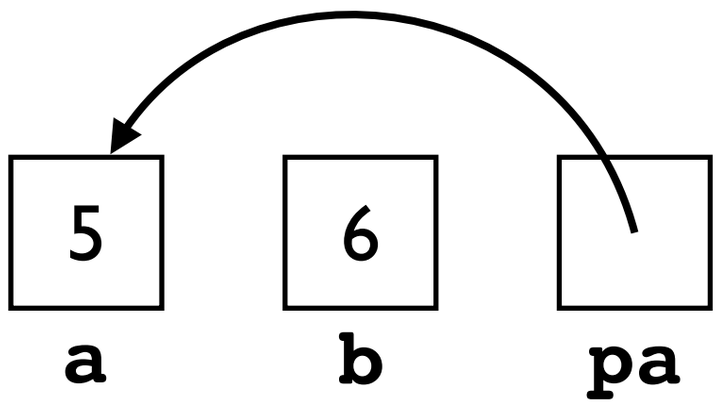
- Pointers
- Arrays
- The syntax for types (it can be weird...)

Pointers

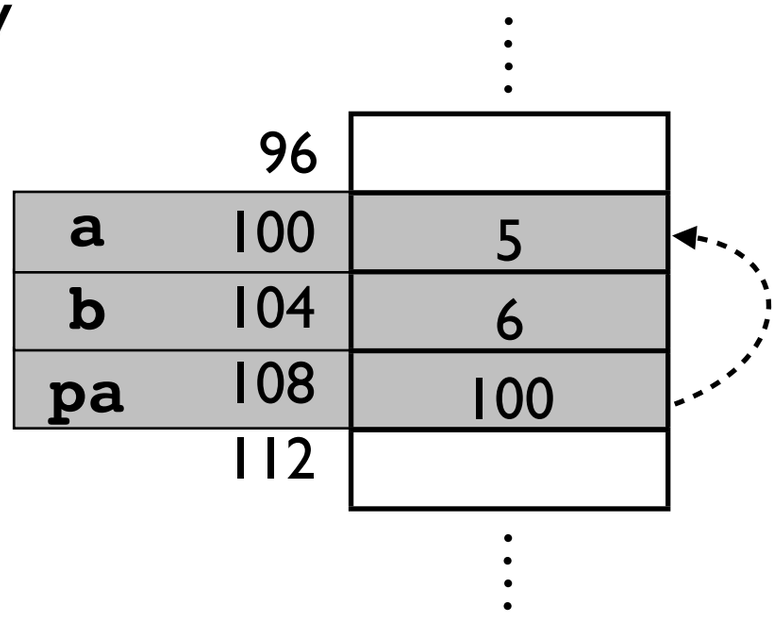
```
int a = 5;
int b = 6;
int *pa = &a; // declares a pointer to a
              // with value as the
              // address of a
```

"address of" operator

As a box diagram



In memory

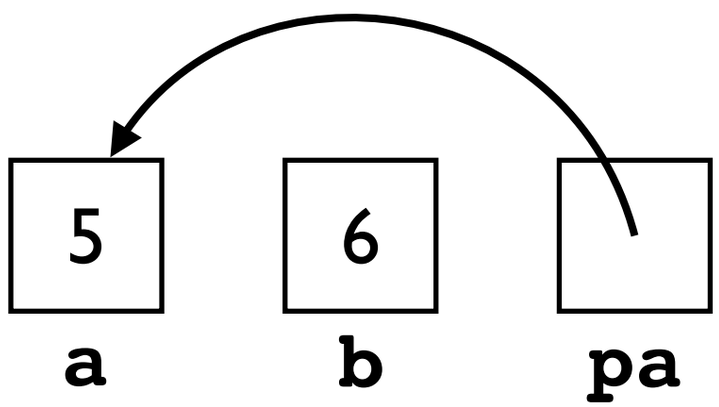


Pointers

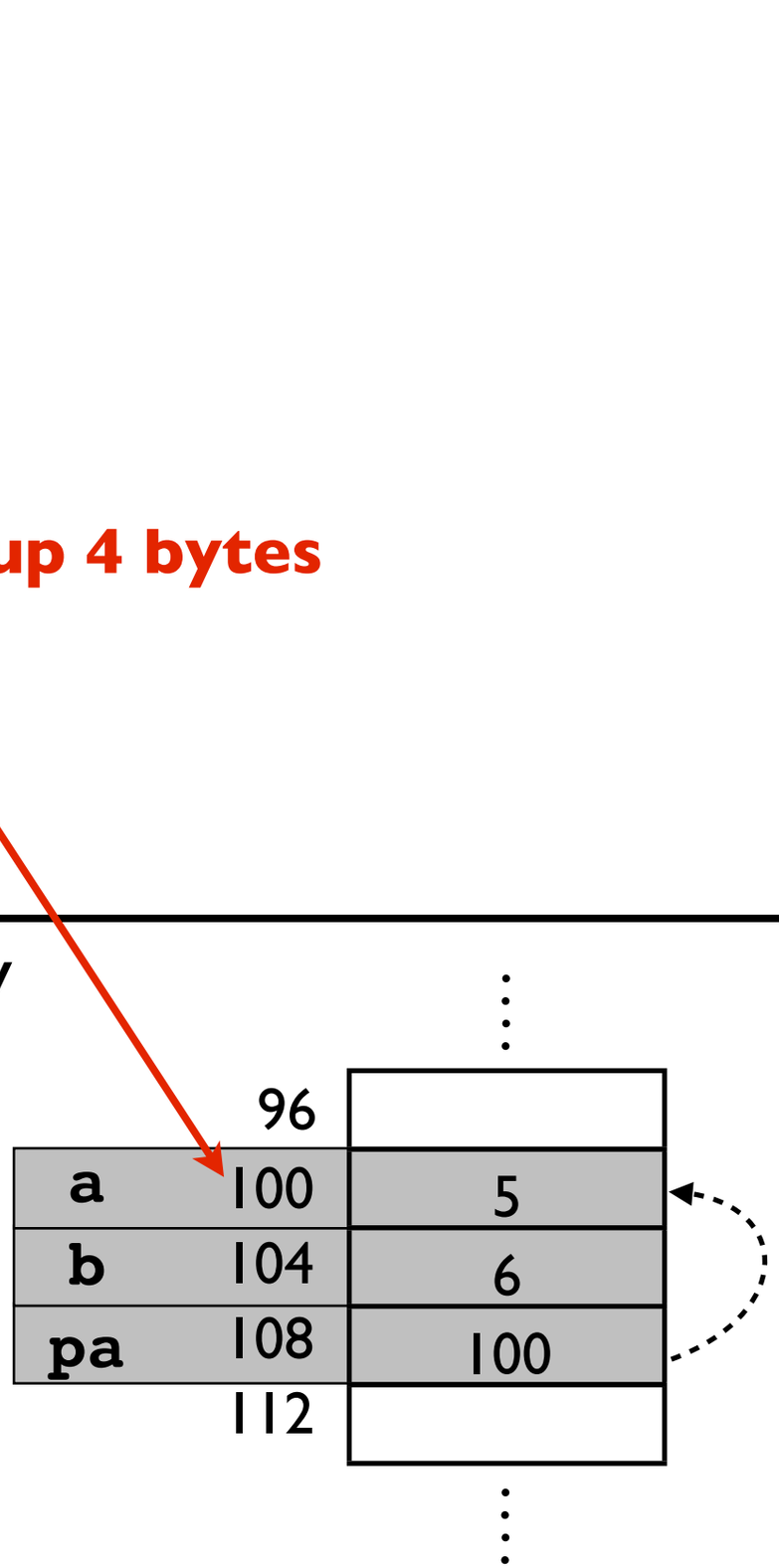
```
int a = 5;
int b = 6;
int *pa = &a;
```

Each `int` takes up 4 bytes

As a box diagram



In memory

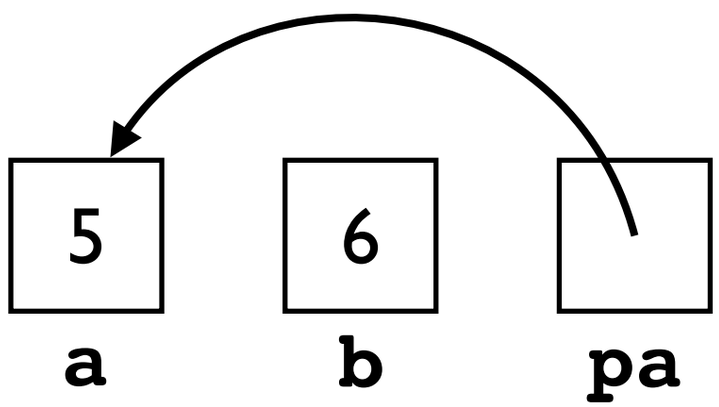


Pointers

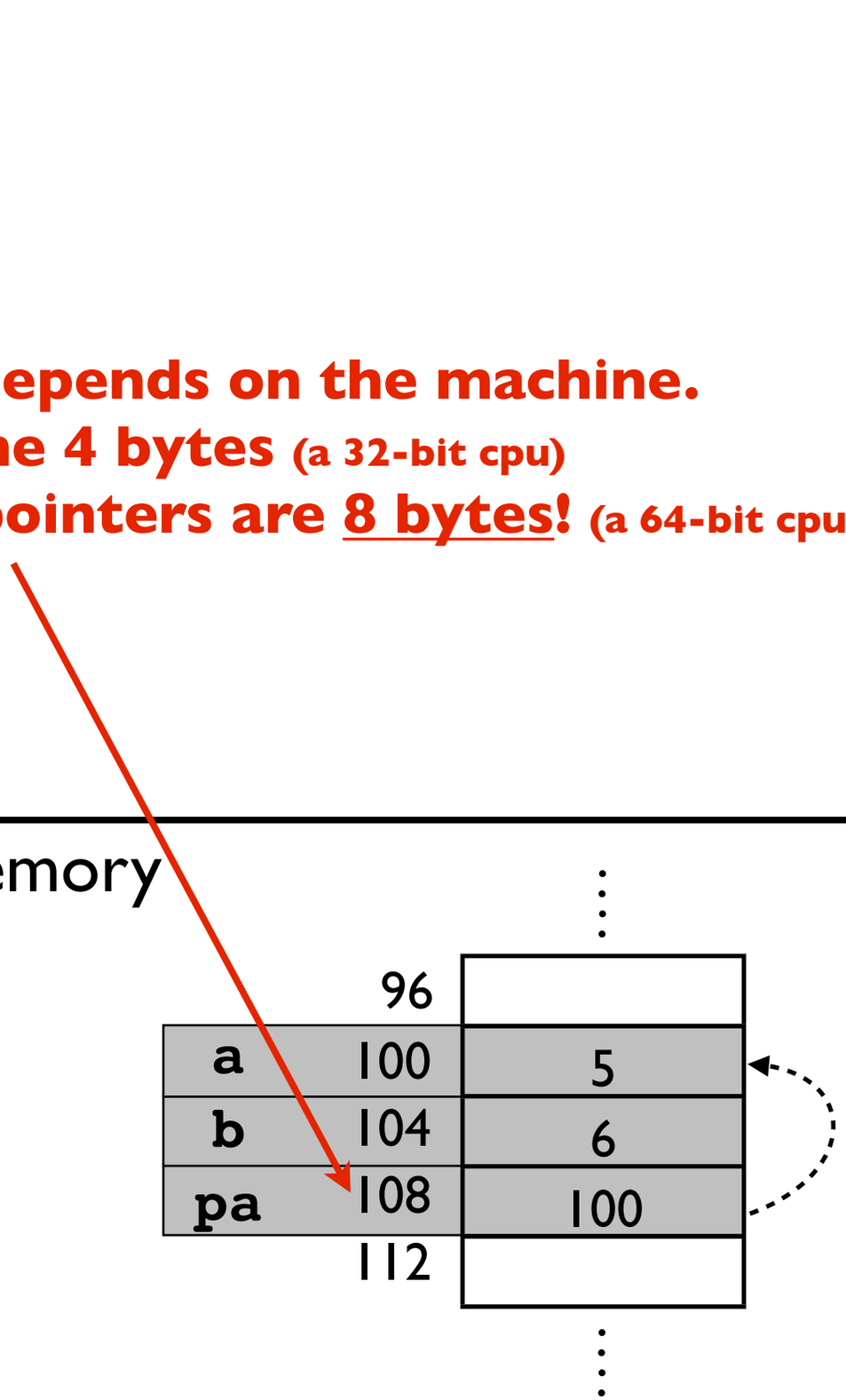
```
int a = 5;
int b = 6;
int *pa = &a;
```

Size of a pointer depends on the machine.
- **Here, we assume 4 bytes (a 32-bit cpu)**
- **For your labs, pointers are 8 bytes! (a 64-bit cpu)**

As a box diagram



In memory



Useful tip ...

```
// This program will print the size of
// various data types.  Try it!

#include <stdio.h>

int main(int argc, char *argv[]) {
  printf("sizeof char:  %d\n", sizeof(char));
  printf("sizeof int:   %d\n", sizeof(int));
  printf("sizeof int*:  %d\n", sizeof(int*));
  printf("sizeof char*: %d\n", sizeof(char*));
  return 0;
}
```

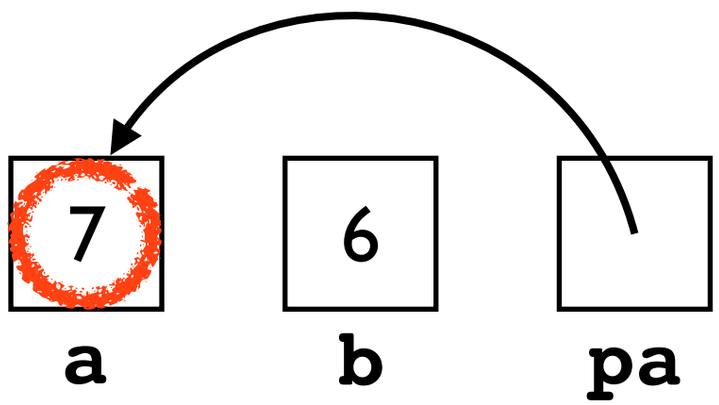
Output on a
64-bit machine

```
sizeof char:  1
sizeof int:   4
sizeof int*:  8
sizeof char*: 8
```

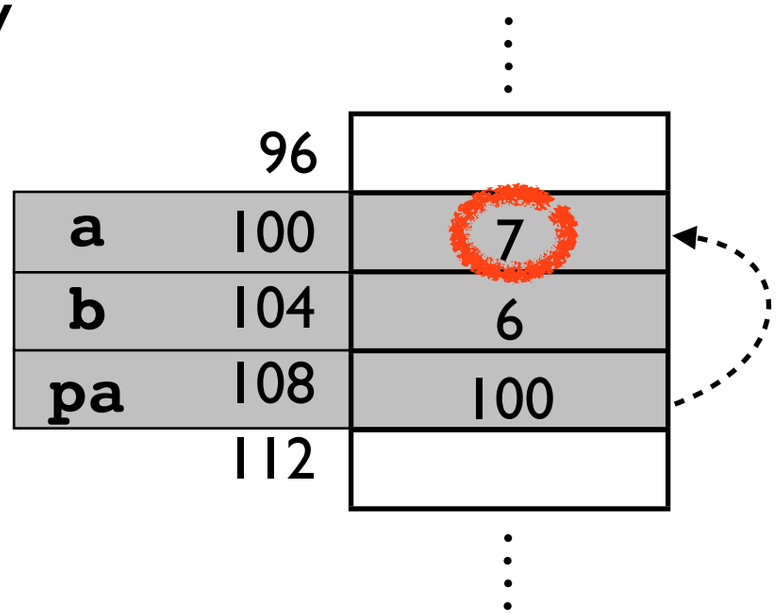
Pointers

```
int a = 5;
int b = 6;
int *pa = &a;

*pa = 7;          // changes value of a to 7
                  // (a == 7)
```

As a box diagram

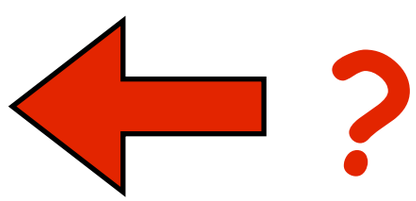


In memory



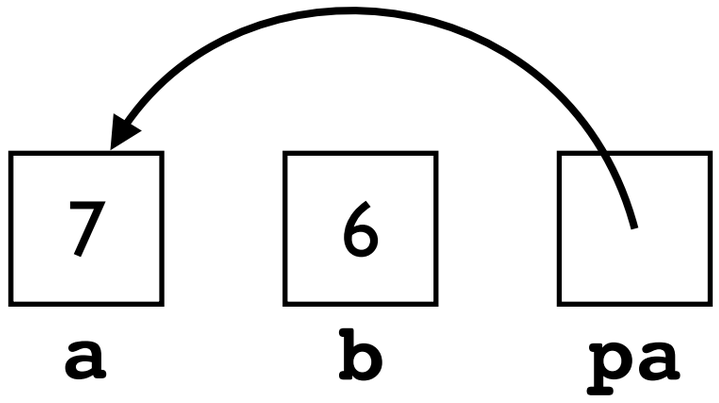
Pointers

```
int a = 5;
int b = 6;
int *pa = &a;

*pa = 7;
b = *pa;
```



?

As a box diagram



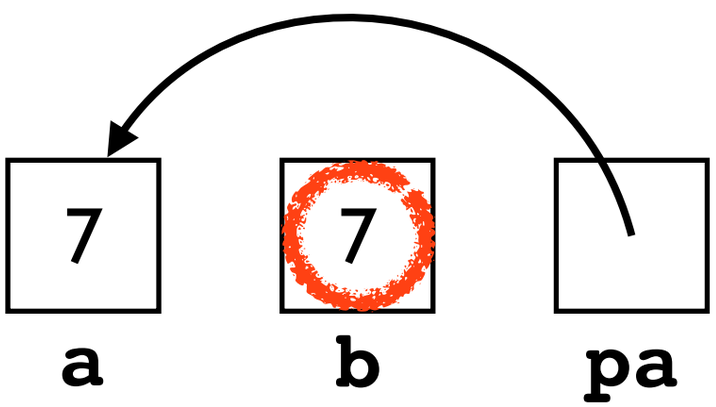
In memory

Pointers

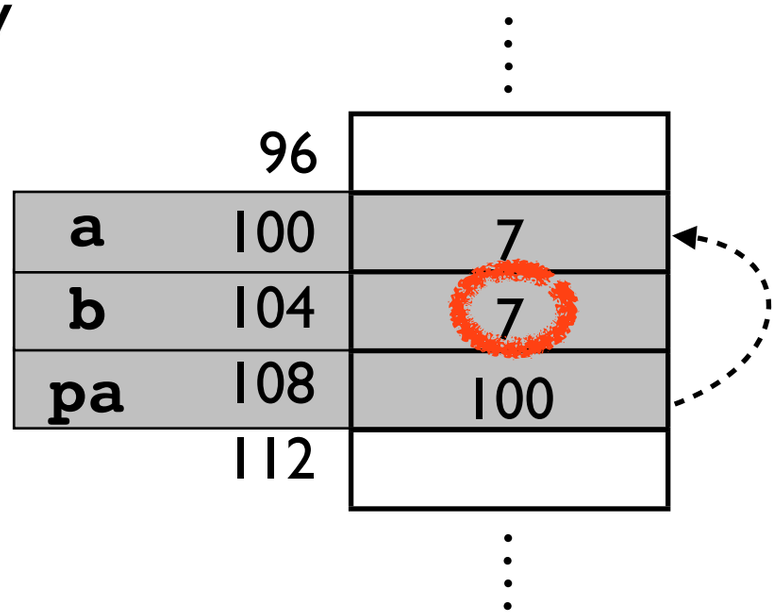
```
int a = 5;
int b = 6;
int *pa = &a;

*pa = 7;
b = *pa;        // changes value of b to 7
```

a "*dereference*"

As a box diagram



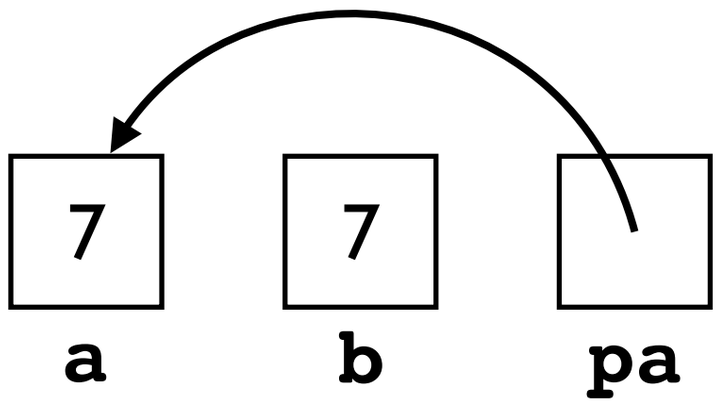
In memory



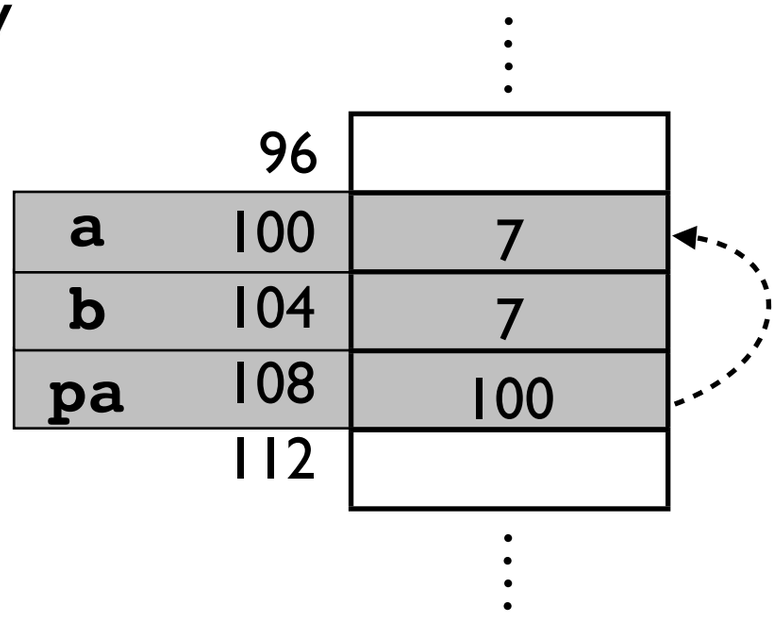
Pointers

```
int a = 5;
int b = 6;
int *pa = &a;

*pa = 7;
b = *pa;
pa = &b;
```

As a box diagram



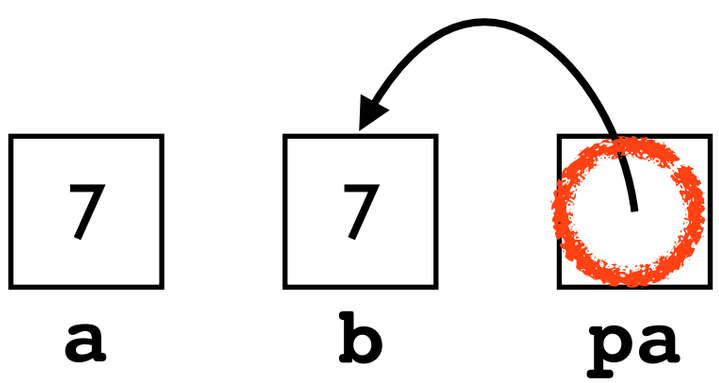
In memory



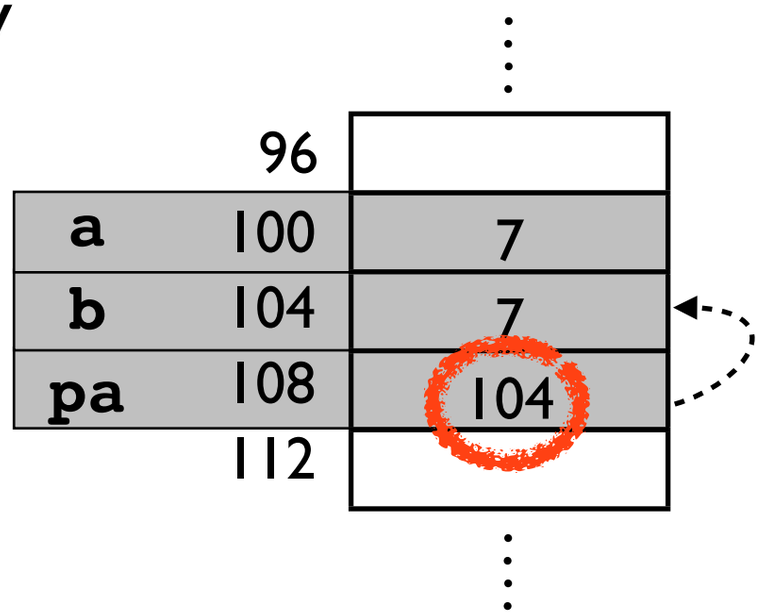
Pointers

```
int a = 5;
int b = 6;
int *pa = &a;

*pa = 7;
b = *pa;
pa = &b;          // changes pa to point at b
```

As a box diagram

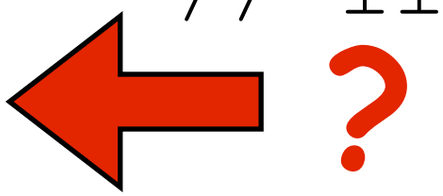


In memory



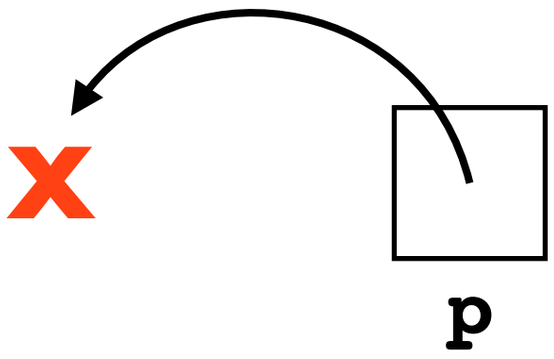
Null pointers

```
int *p = NULL;   // points at address 0
                 // like null in Java
int x = *p;
```

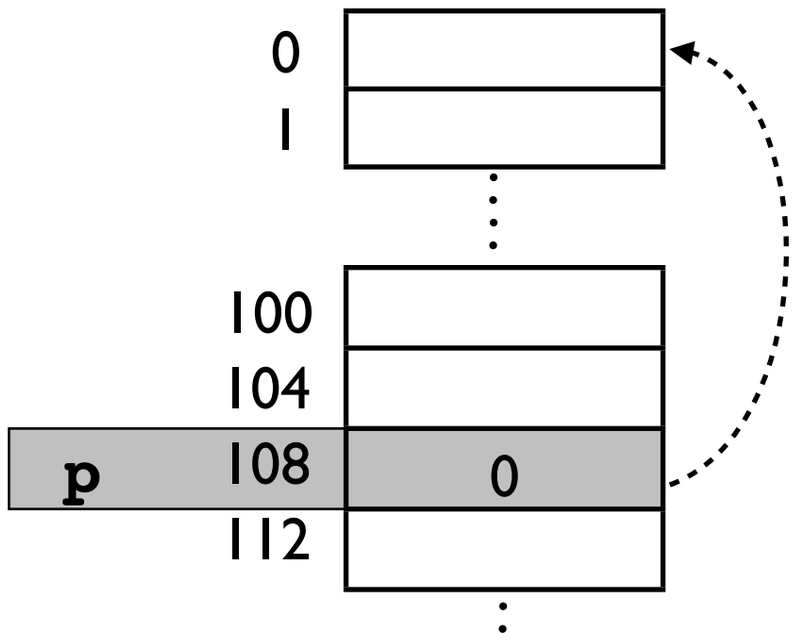


?

As a box diagram



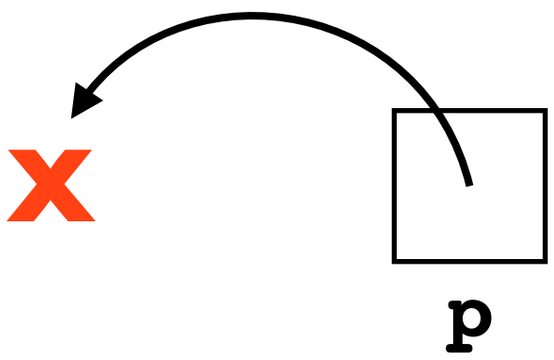
In memory



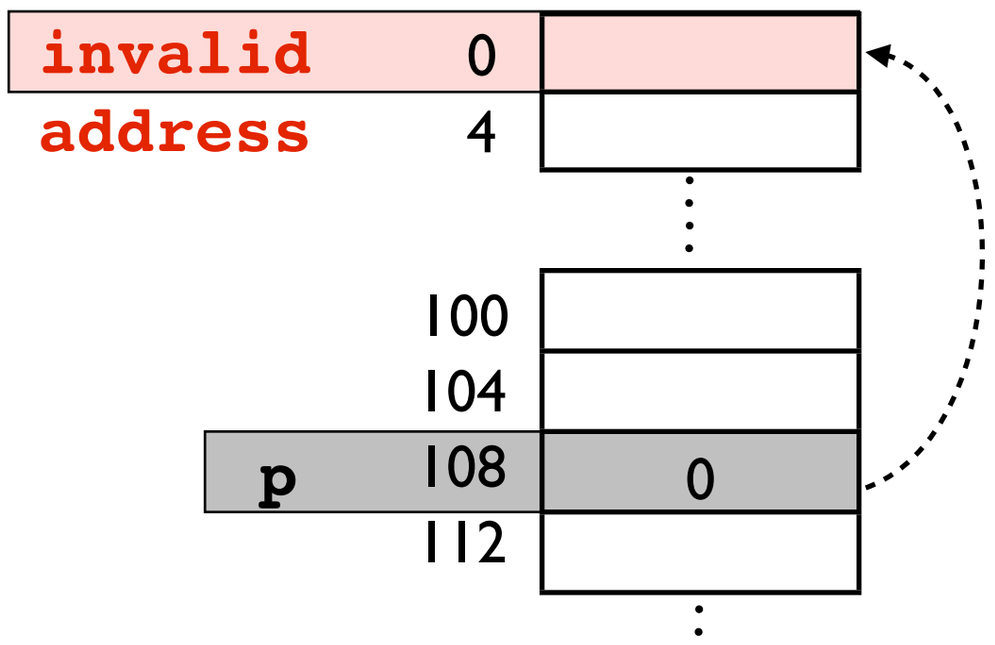
Null pointers

```
int *p = NULL;

int x = *p; // this will crash your program!
            // 0 is an invalid address
            // you get a "segmentation
            //   fault", aka SIGSEGV
```

As a box diagram



In memory

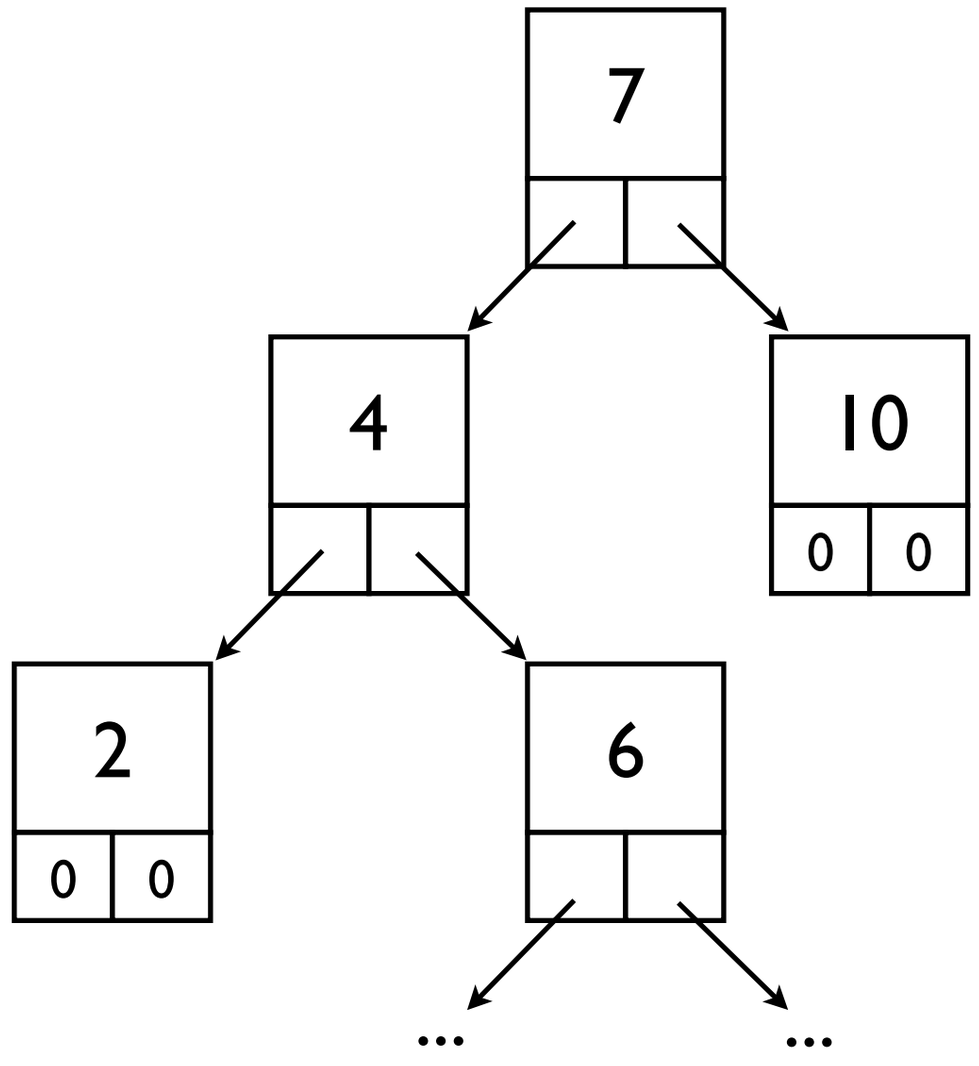


What are pointers good for?

- Data structures!

  Here's a binary tree:

```
struct Tree {
    int x;
    struct Tree *left;
    struct Tree *right;
};
```



Pass-by-value vs. Pass-by-pointer

```
int foo(int x) {
  return x + 1;
}
```

```
void bar(int* x) {
  *x += 1;
}
```

```
void main() {
  int x = 5;
  int y = foo(x);
        // x==5
        // y==6
  bar(&x);
        // x==6
        // y==6
}
```

by-value

by-pointer

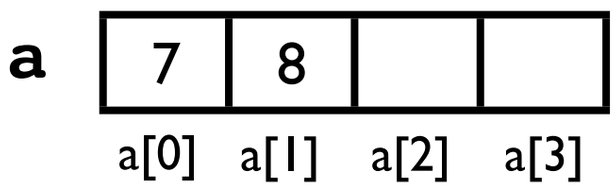
C: three common confusions

- ~~Pointers~~
- Arrays
- The syntax for types (it can be weird...)

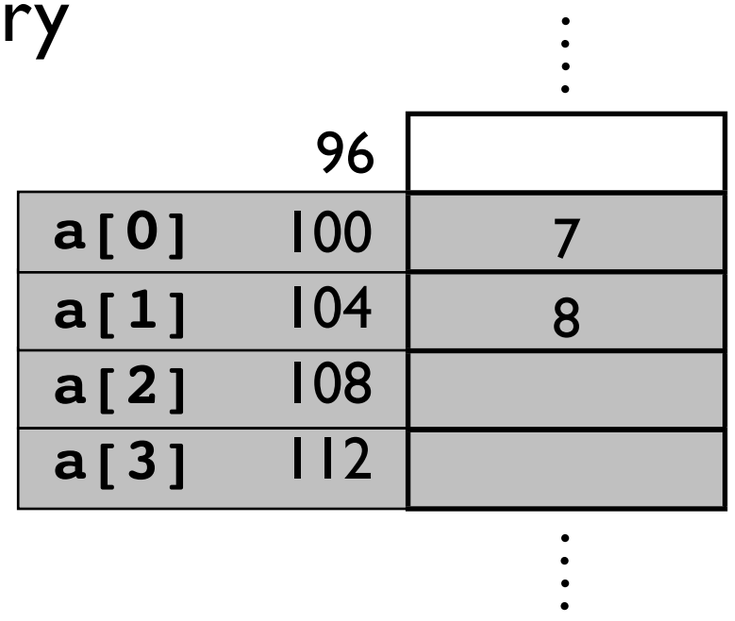
Arrays

```
int a[4];  // declares an array of 4 ints

a[0] = 7;  // assigns to the first element
a[1] = 8;  // assigns to the second element
           // this is just like Java
```

As a box diagram



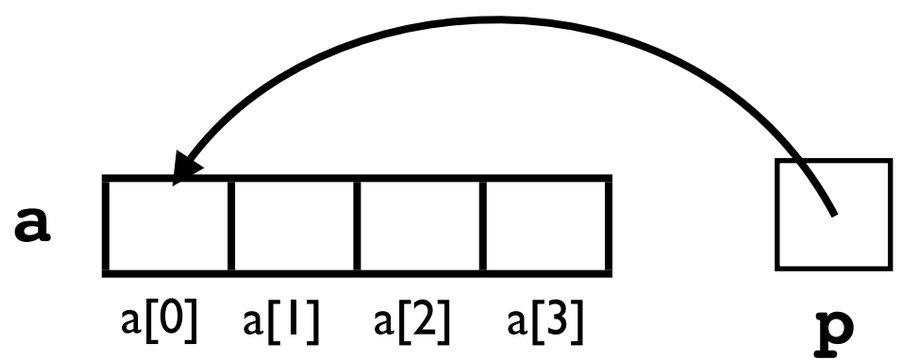
In memory



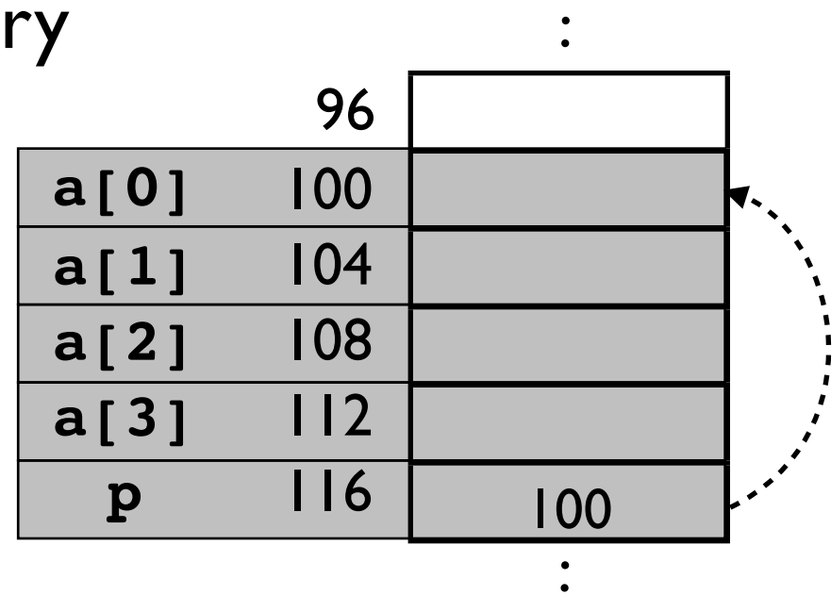
Arrays are pointers!

```
int a[4];
int *p = a; // pointer to the first element
            // of the array

int *p = &a[0]; // another way to write the
                // same declaration
```

As a box diagram

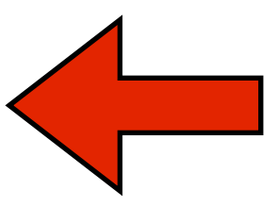


In memory



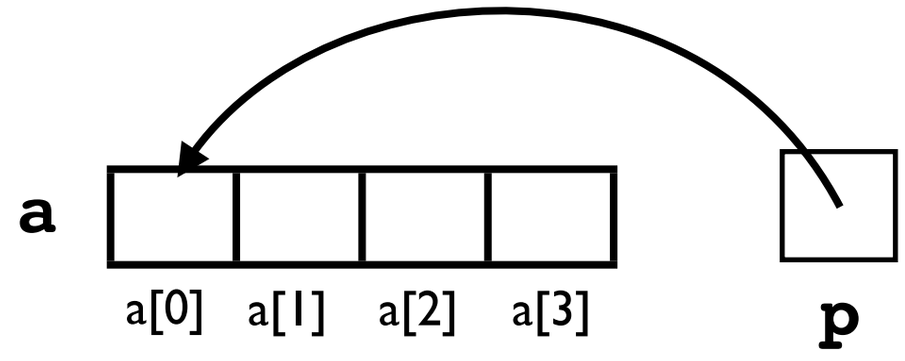
Arrays are pointers!

```
int a[4];
int *p = a; // pointer to the first element
            // of the array


*p = 7;
```

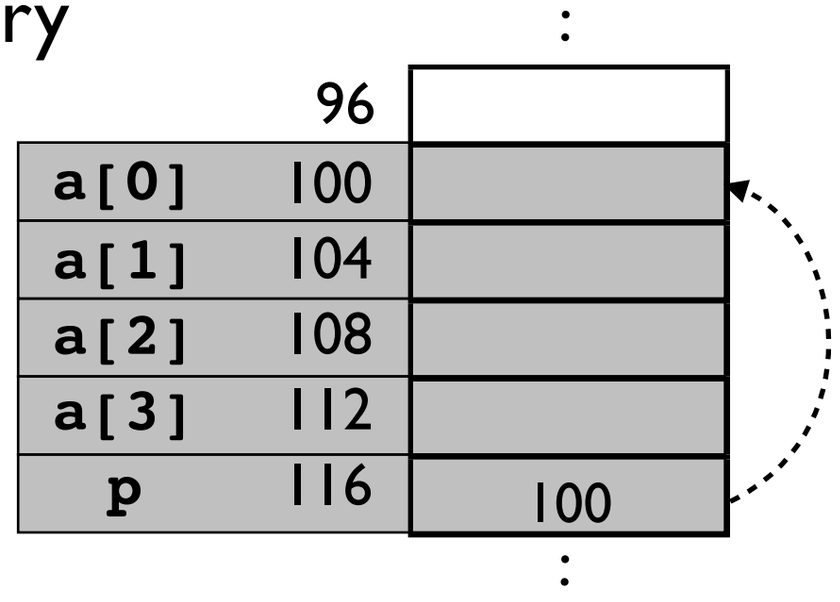


?

As a box diagram



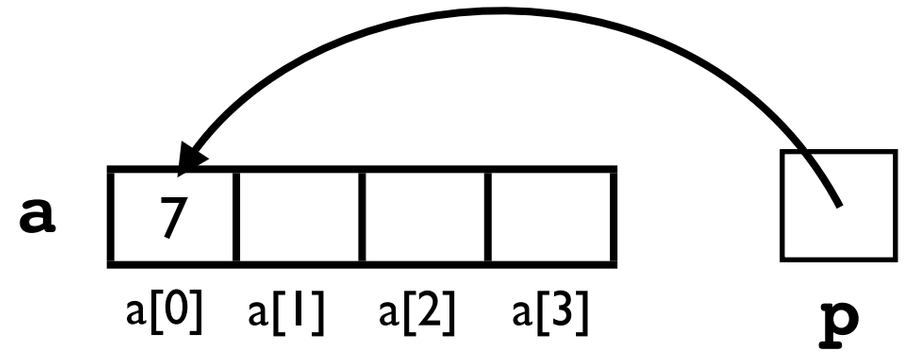
In memory



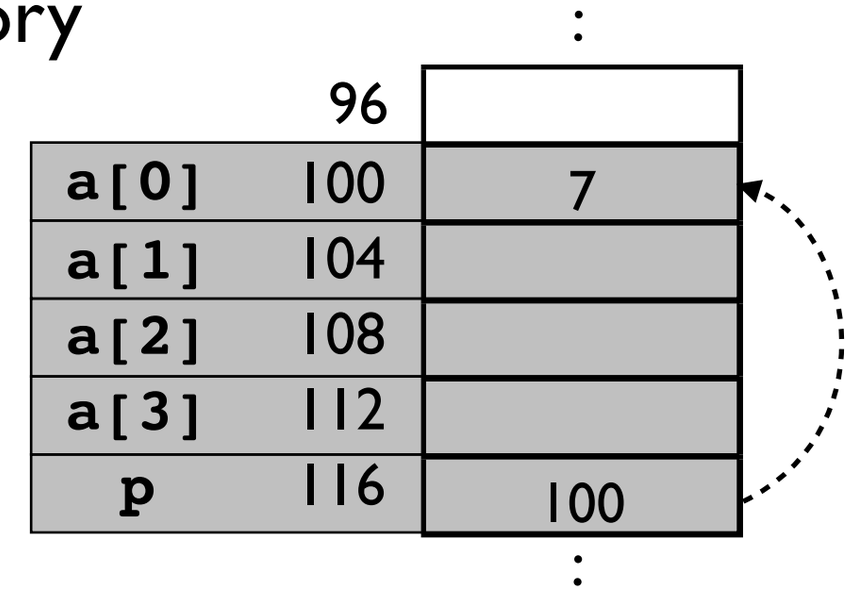
Arrays are pointers!

```
int a[4];
int *p = a; // pointer to the first element
            // of the array

a[0] = 7;   // these statements have the
*p = 7;     // same effect
```

As a box diagram

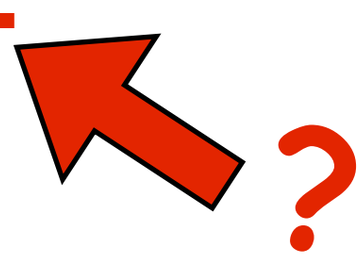


In memory

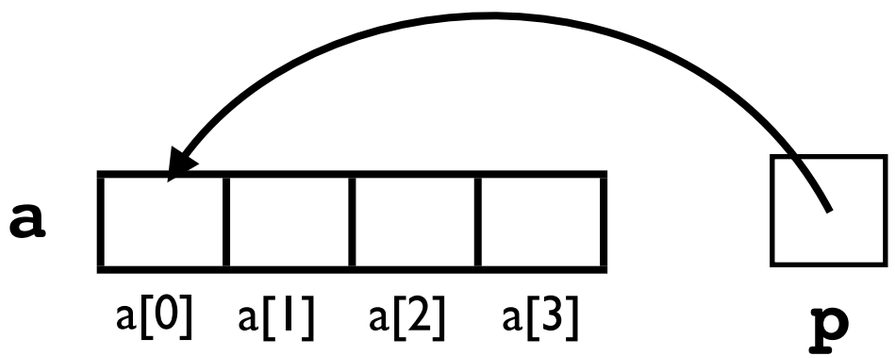


Pointer arithmetic

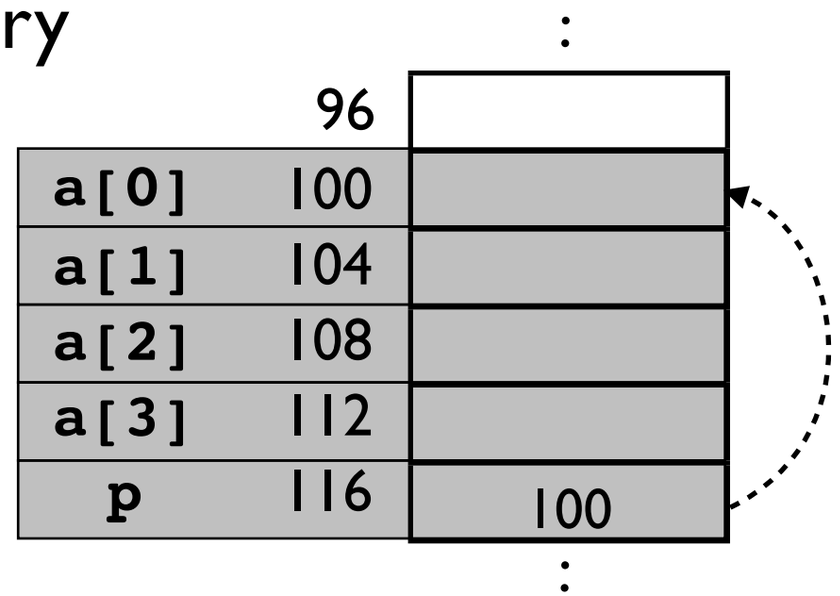
```
int a[4];
int *p = a;


*(p+2) = 9;
```



As a box diagram



In memory

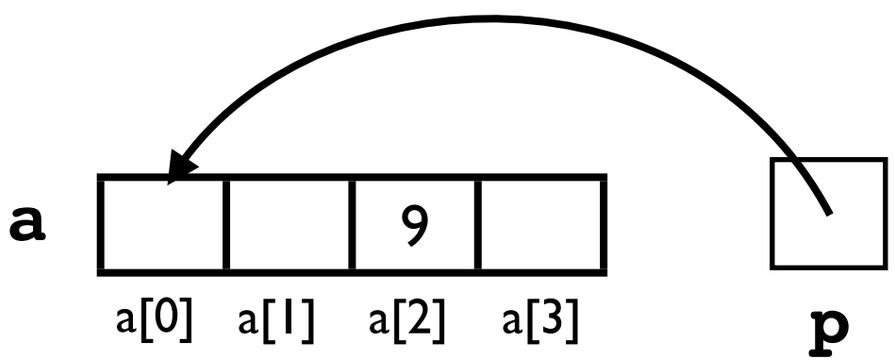


Pointer arithmetic

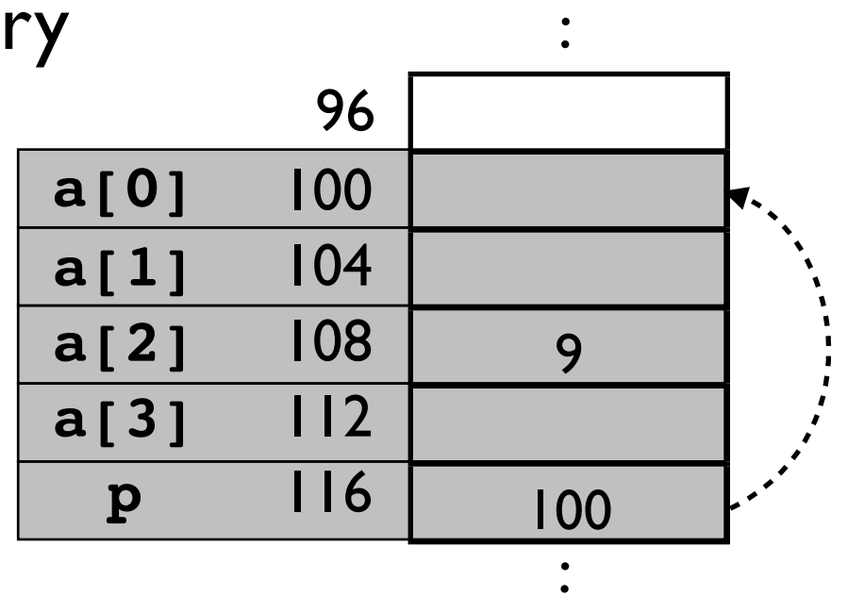
```
int a[4];
int *p = a;

a[2] = 9;     // these statements have the
*(p+2) = 9;   // same effect
```

pointer arithmetic
adds 2*sizeof(int) to the value of p

As a box diagram

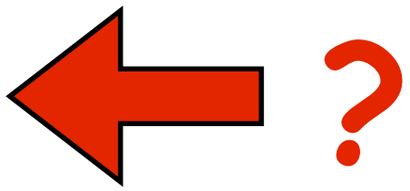


In memory

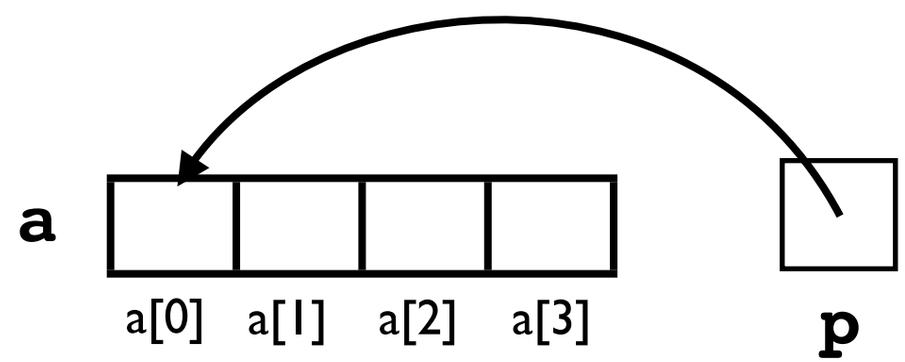


Pointer arithmetic

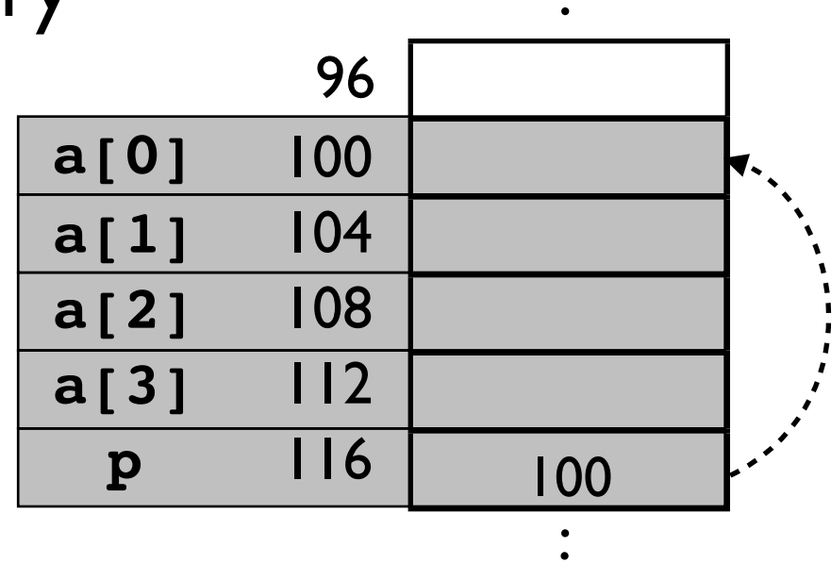
```
int a[4];
int *p = a;

a[4] = 0;
*(p+4) = 0;
```



As a box diagram



In memory

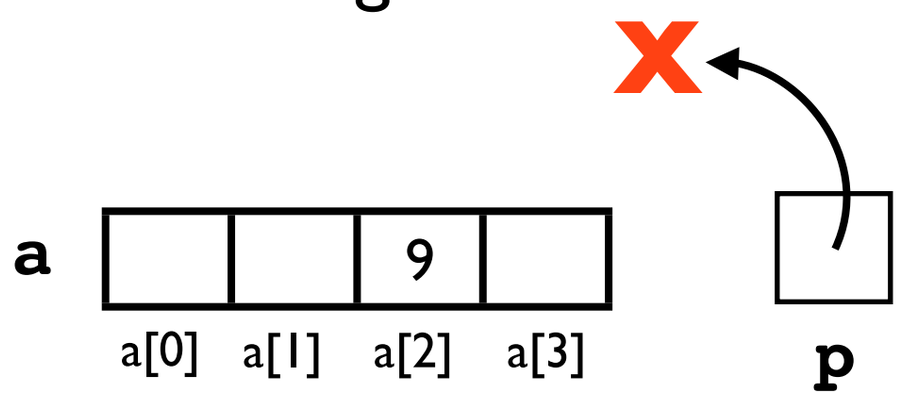


Pointer arithmetic

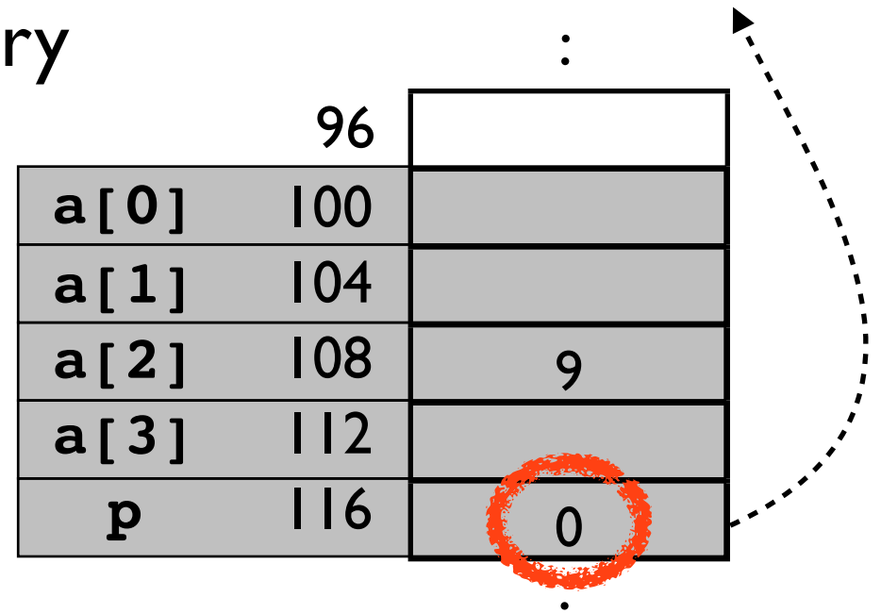
```
int a[4];
int *p = a;

a[4] = 0;     // this statements overwrites
              // p with 0!
```

```
*(p+4) = 42; // this will crash!
```

As a box diagram

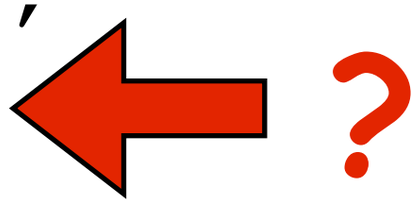


In memory



Crazy example ...

```
int *p = 103406;
*p = 42;
```

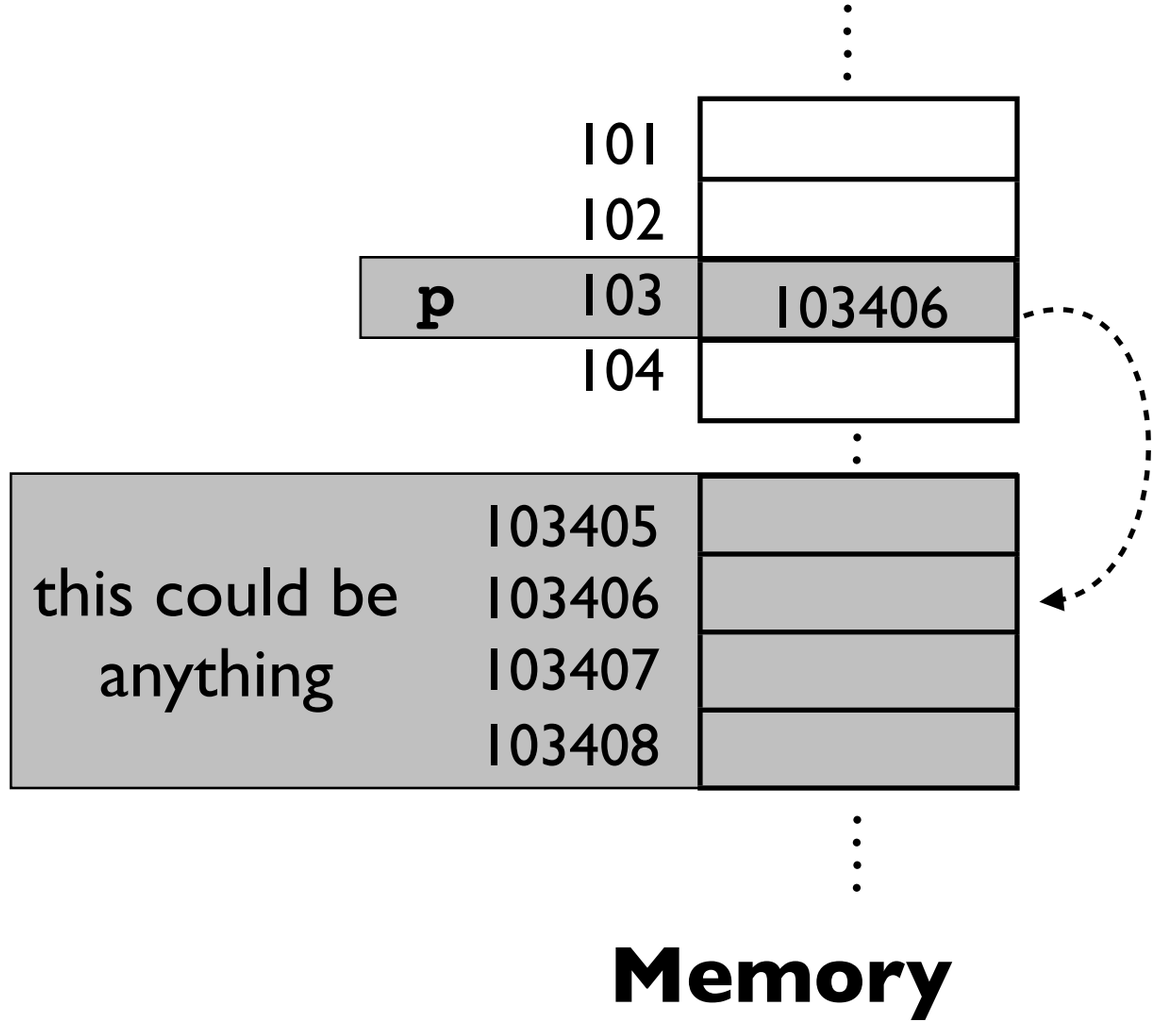


?

This code is evil.

It might overwrite a data structure!

It might even overwrite code!



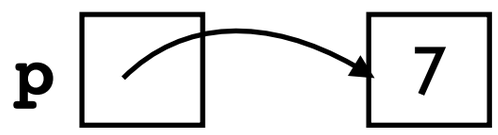
C: three common confusions

- ~~Pointers~~
- ~~Arrays~~
- The syntax for types (it can be weird...)

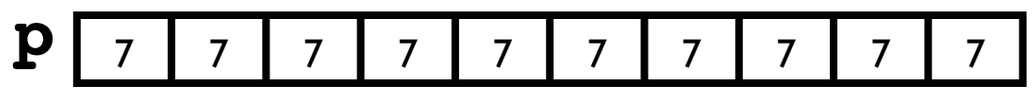
Some Pointer/Array Types

```
int *p
int* p
   // declares a pointer to an integer
```

[note: whitespace doesn't matter]

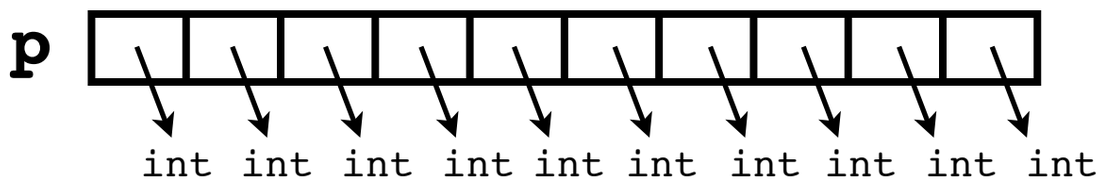



```
int p[10]
   // declares an array of 10 integers
```

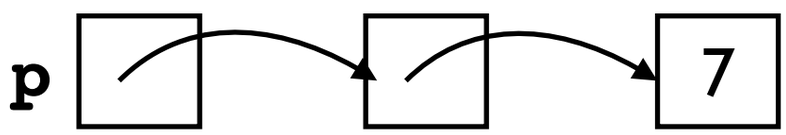


Some Pointer/Array Types

```
int *p[10]
  // declares an array of 10 pointers which
  // each point to an integer
```



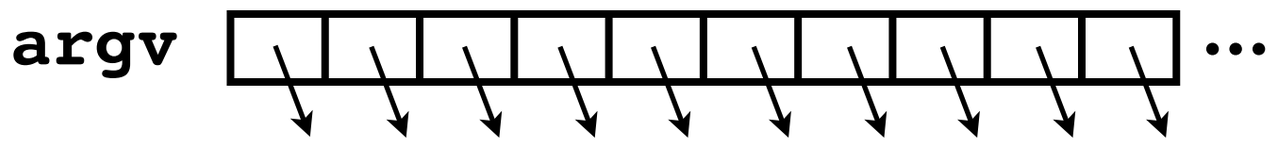

```
int **p
  // declares a pointer to a pointer to
  // an integer
```



Some Pointer/Array Types

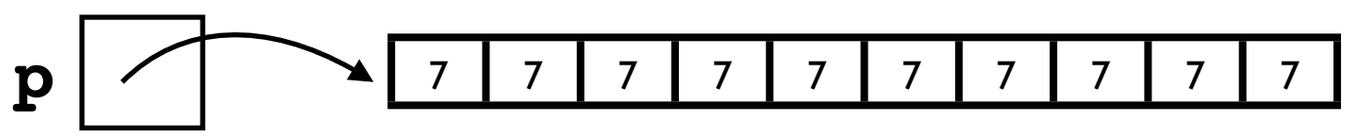
```
char *argv[]
  // declares an array of pointers-to-chars
  // the array has unknown length
```

[used in main()]




```
int (*p)[10]
  // declares a pointer to array of 10
  // integers
```

[you probably won't use this in this class]



Today

- ~~Introduction~~
- ~~C overview~~
- Lab 1 quickstart
  - how to get started
  - how to compile and debug C code

DEMO!