CS 262 Lecture 8: Pointers

GEORGE MASON UNIVERSITY

Overview of Lecture 8

Pointers

Basic pointer operations

Working with pointers – Examples

How is memory (RAM) allocated?

Null pointers, Segmentation faults

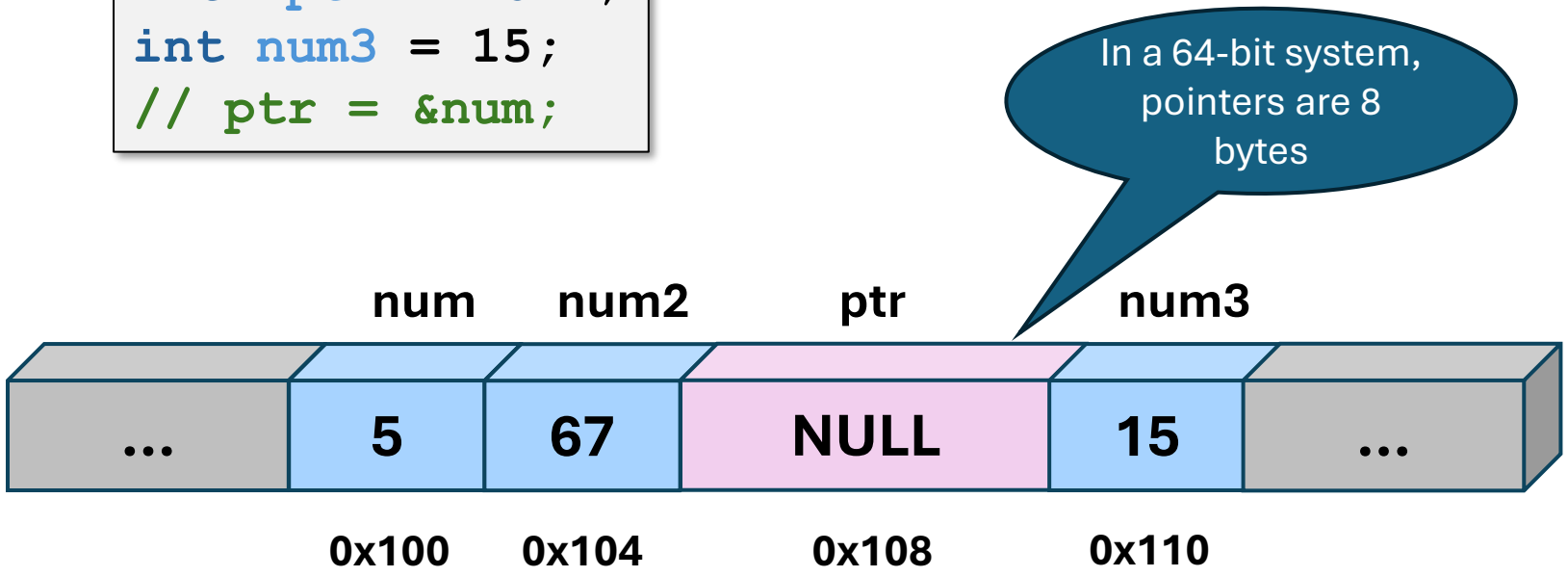
Array/pointer duality (kind of)

Functions - Call by reference

Pointer Basics - Definition

A **pointer** is a variable containing the address of another variable

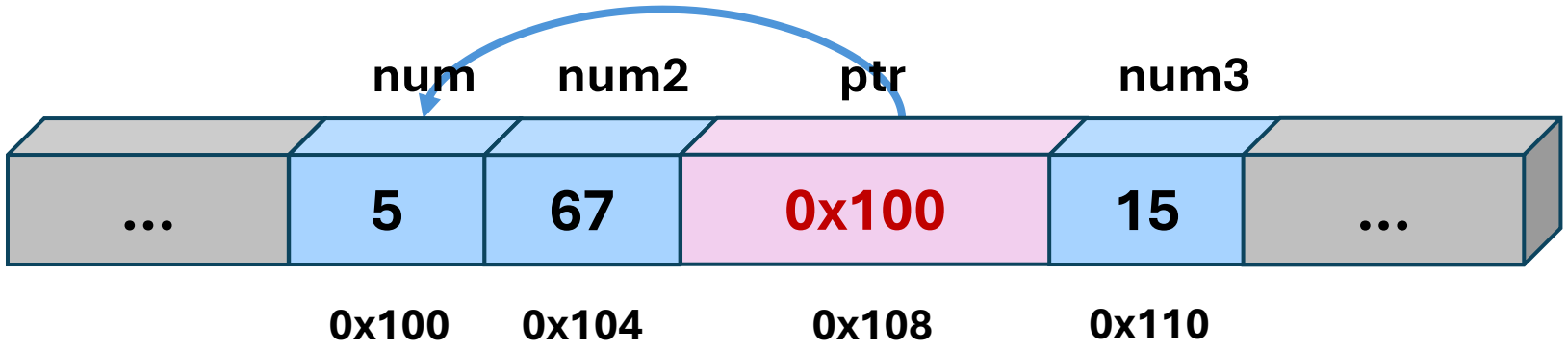
```
int num = 5;
int num2 = 67;
int *ptr = NULL;
int num3 = 15;
// ptr = &num;
```



Pointer Basics – A Pointer Pointing

A **pointer** is a variable containing the address of another variable

```
int num = 5;
int num2 = 67;
int *ptr = NULL;
int num3 = 15;
ptr = &num;
```



Pointer Basics – Some Important Details

Pointers

Just like other variables, uninitialized pointers contain garbage

Because of this, initialize pointers to an address or to NULL

Pointers have a type, corresponding to the type of variable they point to

```
int *ptr = NULL; // the type int * is a pointer to an int
char *ptr2 = NULL; // and char * is a pointer to a char
```

The type matters because of pointer arithmetic

Pointer Basics - Dereferencing

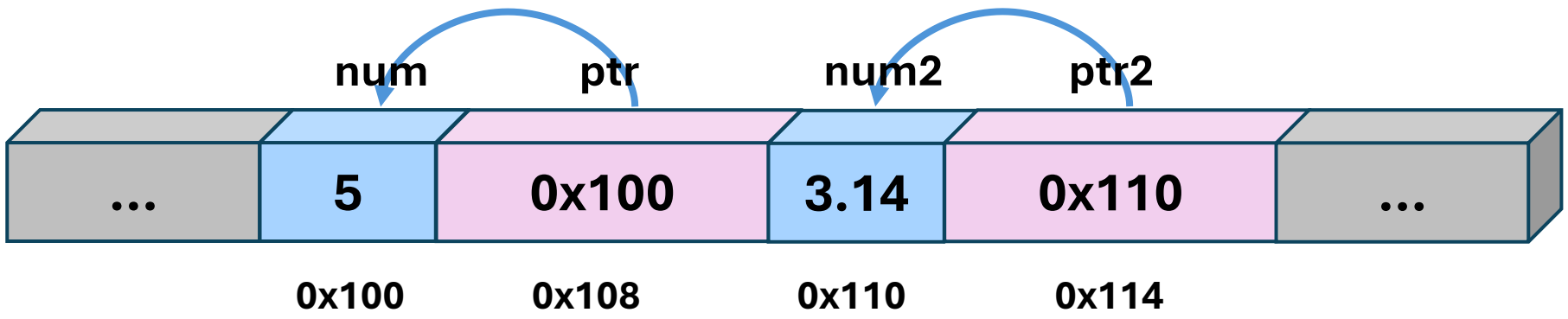
Dereferencing

Done by using the operator *

This lets us access the variable being pointed to by the pointer

```
int num = 5;
int *ptr = &num;
float num2 = 3.14;
float *ptr2 = &num2;
printf("%d", *ptr);
printf("%.2f", *ptr2);
```

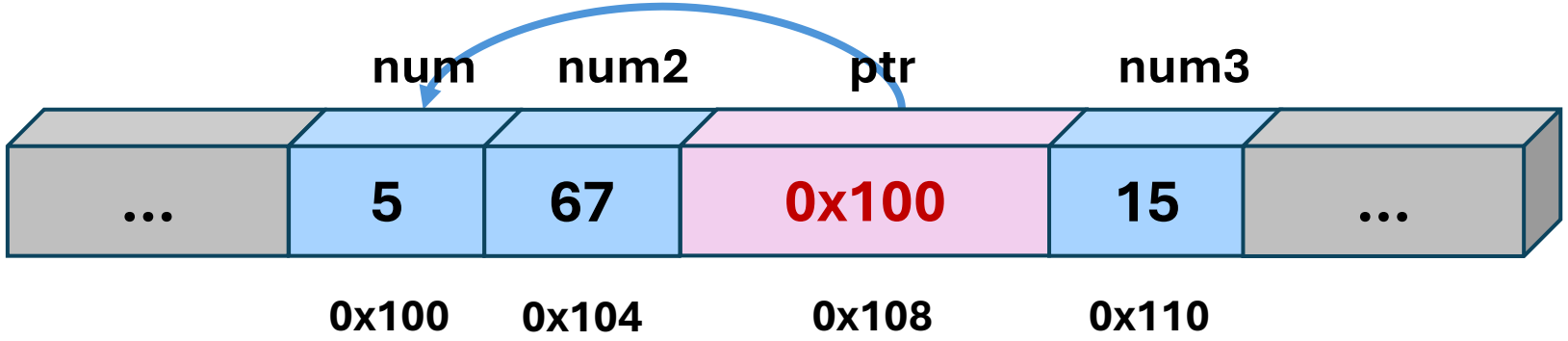
This prints the value at the memory address ptr is referencing



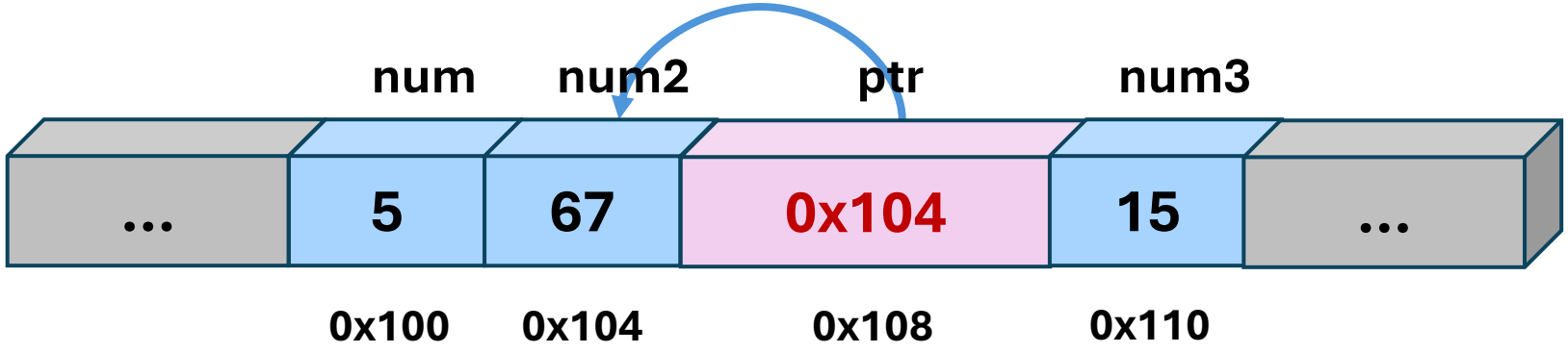
Intro to Pointer Arithmetic

Incrementing or decrementing the pointer changes the memory address it references, by the amount corresponding to the size of variable type

```
ptr = &num;
```




```
ptr++;
```



NULL Pointers

To check if a pointer is NULL

```
if(ptr == NULL) { // This tells us if a pointer is unassigned
...
```

Dereferencing a NULL pointer

This will result in a **segmentation fault**

Dereferencing a pointer referencing memory your program doesn't have access to

Also results in a **segmentation fault**

How is RAM Allocated?

What does it mean for your program to not have access to memory?

Memory is allocated to your program in **Segments**

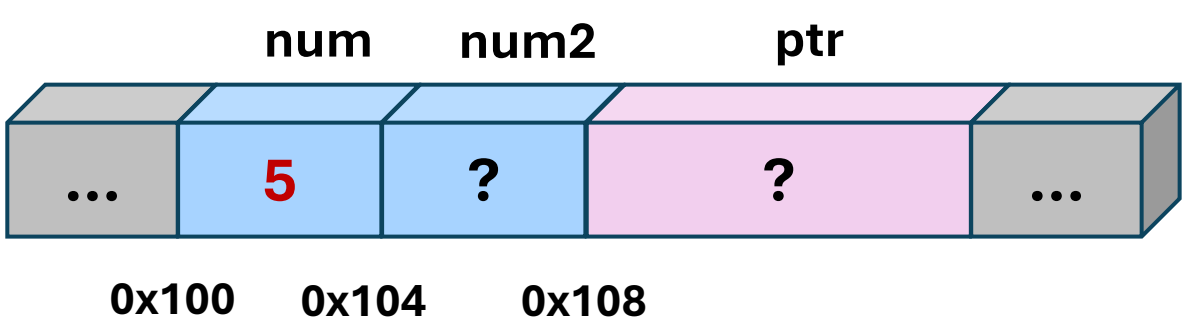
Segment	Contains
Stack	All local variables
Heap	Dynamic memory (we talk about this later)
Text/Code	Source Code
Data	Global Variables

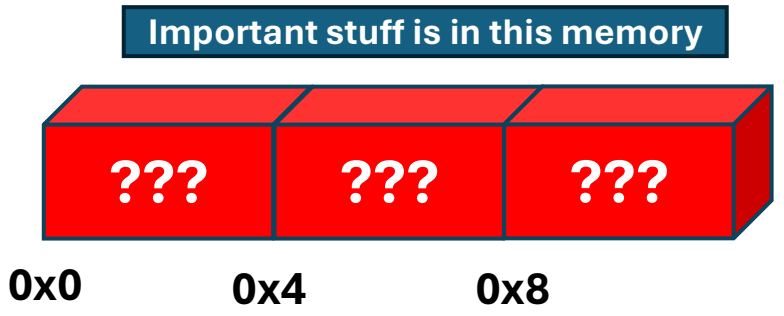
Dereferencing a pointer pointing to an address outside of these segments is what causes **segmentation faults**

What Happens If We Aren't Careful?

We Are Here

```
int num = 5;
int num2 = 67;
int *ptr = &num; // Initialize ptr to the address of num
ptr = &num2; // Set ptr to point to num2
printf("%d", *ptr); // This prints out the value of num2
ptr = NULL; // set ptr to NULL
printf("%d", *ptr); // What happens when we try this?
```

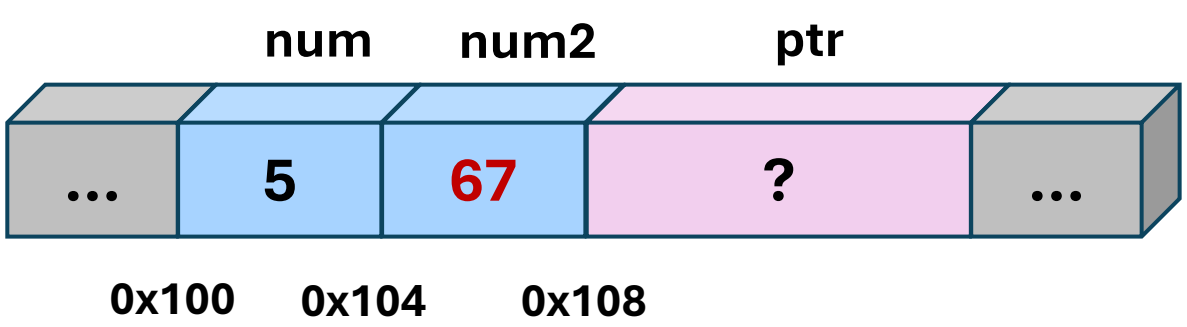


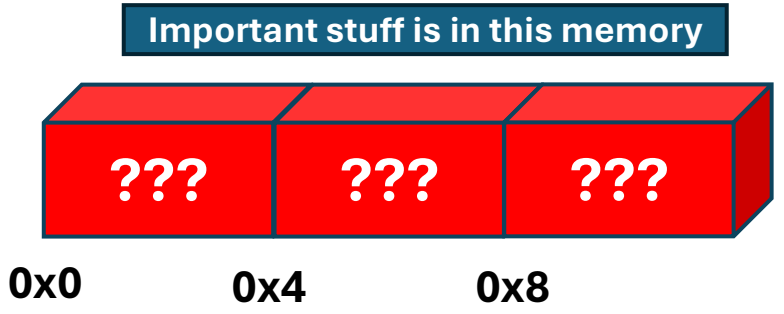


What Happens If We Aren't Careful?

We Are Here

```
int num = 5;
int num2 = 67;
int *ptr = &num; // Initialize ptr to the address of num
ptr = &num2; // Set ptr to point to num2
printf("%d", *ptr); // This prints out the value of num2
ptr = NULL; // set ptr to NULL
printf("%d", *ptr); // What happens when we try this?
```

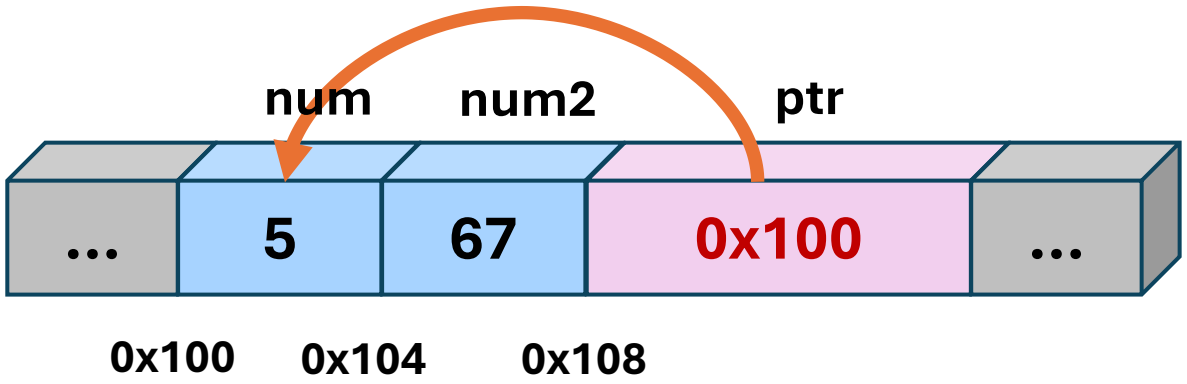


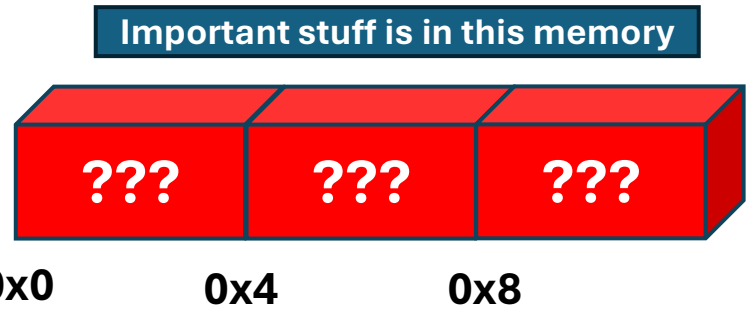


What Happens If We Aren't Careful?

We Are Here

```
int num = 5;
int num2 = 67;
int *ptr = &num; // Initialize ptr to the address of num
ptr = &num2; // Set ptr to point to num2
printf("%d", *ptr); // This prints out the value of num2
ptr = NULL; // set ptr to NULL
printf("%d", *ptr); // What happens when we try this?
```

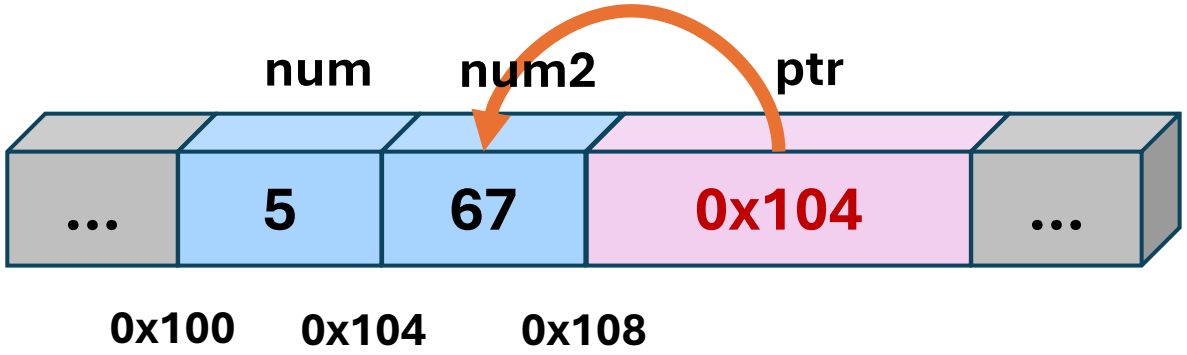


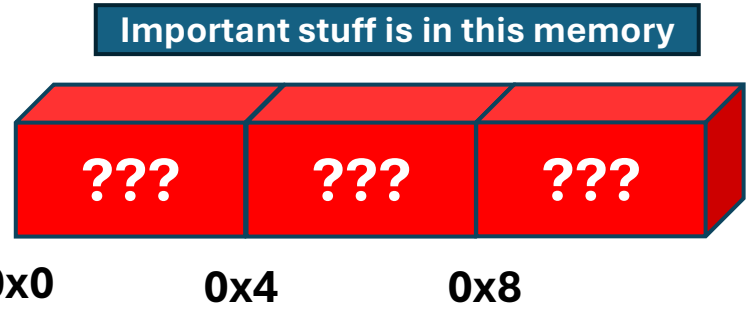


What Happens If We Aren't Careful?

We Are Here

```
int num = 5;
int num2 = 67;
int *ptr = &num; // Initialize ptr to the address of num
ptr = &num2; // Set ptr to point to num2
printf("%d", *ptr); // This prints out the value of num2
ptr = NULL; // set ptr to NULL
printf("%d", *ptr); // What happens when we try this?
```





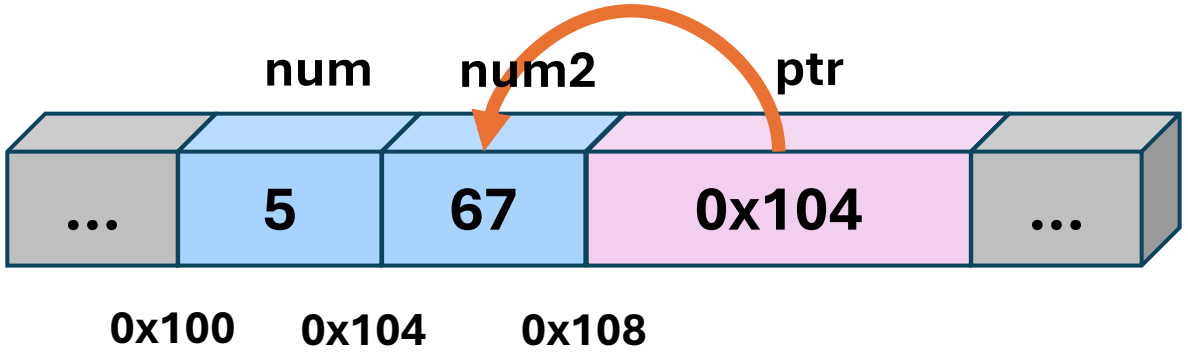
What Happens If We Aren't Careful?

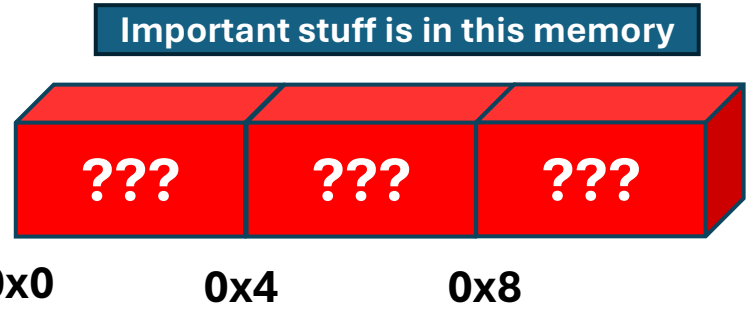
We Are Here

```c
int num = 5;
int num2 = 67;
int *ptr = &num; // Initialize ptr to the address of num
ptr = &num2; // Set ptr to point to num2
printf("%d", *ptr); // This prints out the value of num2
ptr = NULL; // set ptr to NULL
printf("%d", *ptr); // What happens when we try this?
```

Terminal

```
67
```

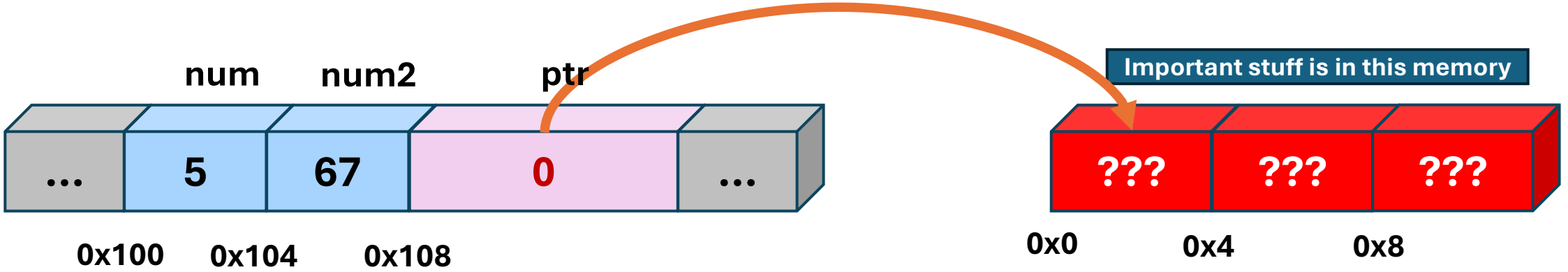




What Happens If We Aren't Careful?

We Are Here

```
int num = 5;
int num2 = 67;
int *ptr = &num; // Initialize ptr to the address of num
ptr = &num2; // Set ptr to point to num2
printf("%d", *ptr); // This prints out the value of num2
ptr = NULL; // set ptr to NULL
printf("%d", *ptr); // What happens when we try this?
```



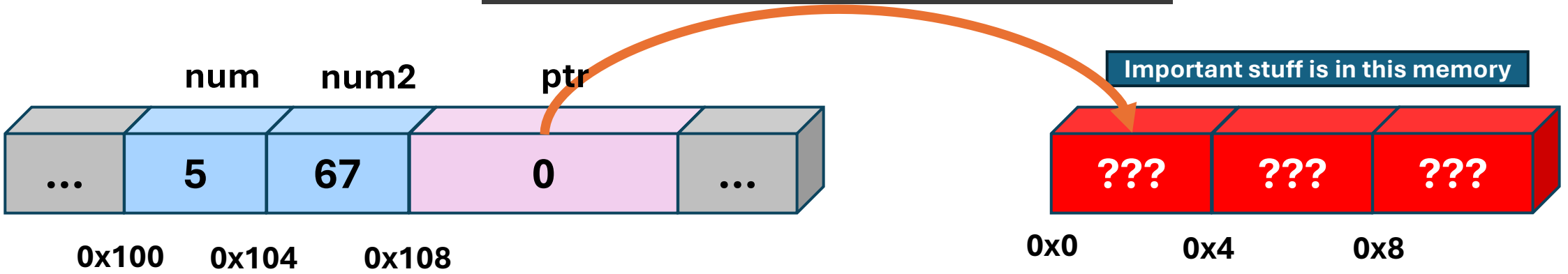
Let's See A Segmentation Fault

We Are Here

```c
int num = 5;
int num2 = 67;
int *ptr = &num; // Initialize ptr to the address of num
ptr = &num2; // Set ptr to point to num2
printf("%d", *ptr); // This prints out the value of num2
ptr = NULL; // set ptr to NULL
printf("%d", *ptr); // What happens when we try this?
```

Terminal

```
Segmentation fault (core dumped)
```



Working With Arrays and Pointer Arithmetic

```c
int main() {
    int arr[] = {2, 4, 8, 16, 32};
    int size = sizeof(arr) / sizeof(arr[0]);
    print_array(arr, size);
    return 0;
}

void print_array(int *ptr, int size) {
    for(int i=0; i<size; i++) {
        printf("%d ", *(ptr + i));
    }
}
```

And why can we pass in `arr` as a parameter

When the function wants a pointer?

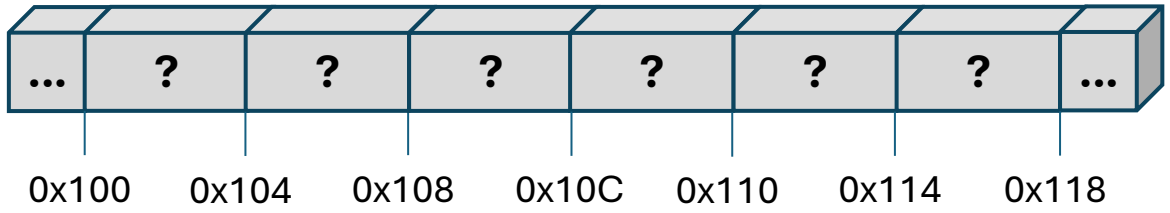
And what's going on here?

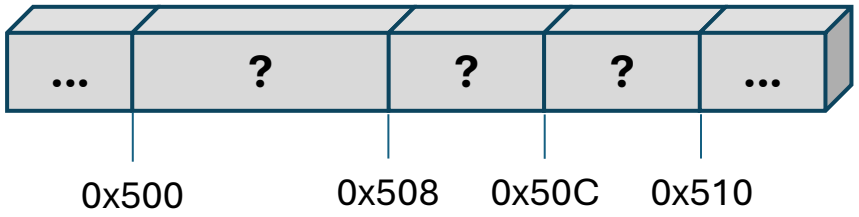
Working With Arrays and Pointer Arithmetic

Here

```c
int main() {
    int arr[] = {2, 4, 8, 16, 32};
    // This computes the size of the array
    int size = sizeof(arr) / sizeof(arr[0]);
    print_array(arr, size)
    return 0;
}

void print_array(int *ptr, int size) {
    for(int i=0; i<size; i++) {
        // We use pointer arithmetic here
        printf("%d ", *(ptr + i));
    }
}
```



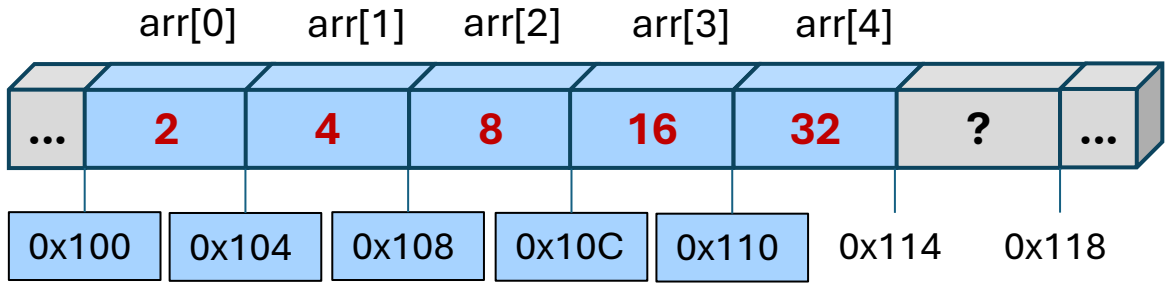


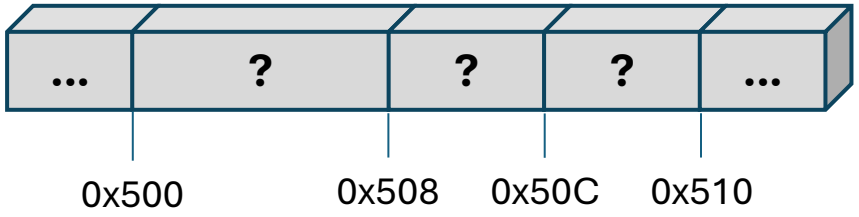
Working With Arrays and Pointer Arithmetic

We Are Here

```c
int main() {
    int arr[] = {2, 4, 8, 16, 32};
    // This computes the size of the array
    int size = sizeof(arr) / sizeof(arr[0]);
    print_array(arr, size)
    return 0;
}

void print_array(int *ptr, int size) {
    for(int i=0; i<size; i++) {
        // We use pointer arithmetic here
        printf("%d ", *(ptr + i));
    }
}
```



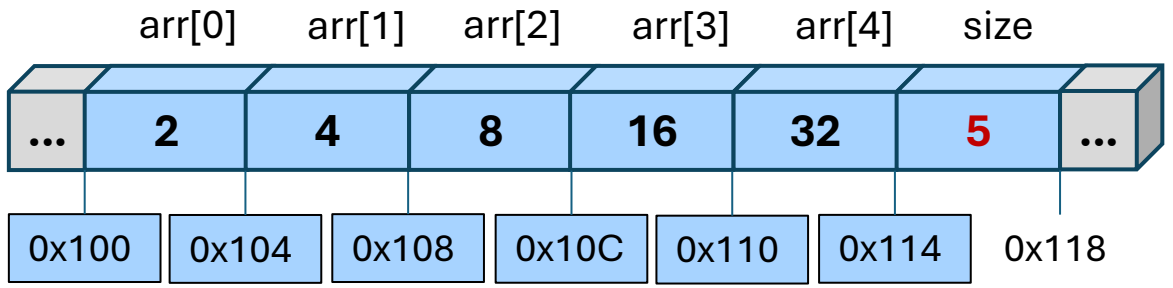


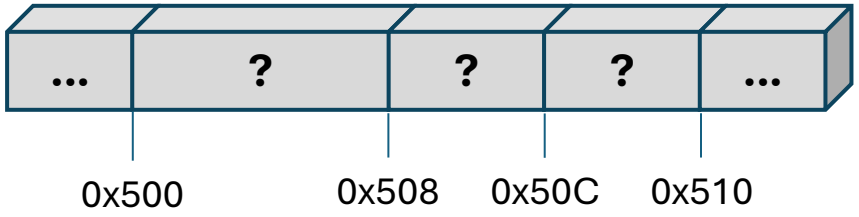
Working With Arrays and Pointer Arithmetic

We Are Here

```
int main() {
    int arr[] = {2, 4, 8, 16, 32};
    // This computes the size of the array
    int size = sizeof(arr) / sizeof(arr[0]);
    print_array(arr, size)
    return 0;
}

void print_array(int *ptr, int size) {
    for(int i=0; i<size; i++) {
        // We use pointer arithmetic here
        printf("%d ", *(ptr + i));
    }
}
```





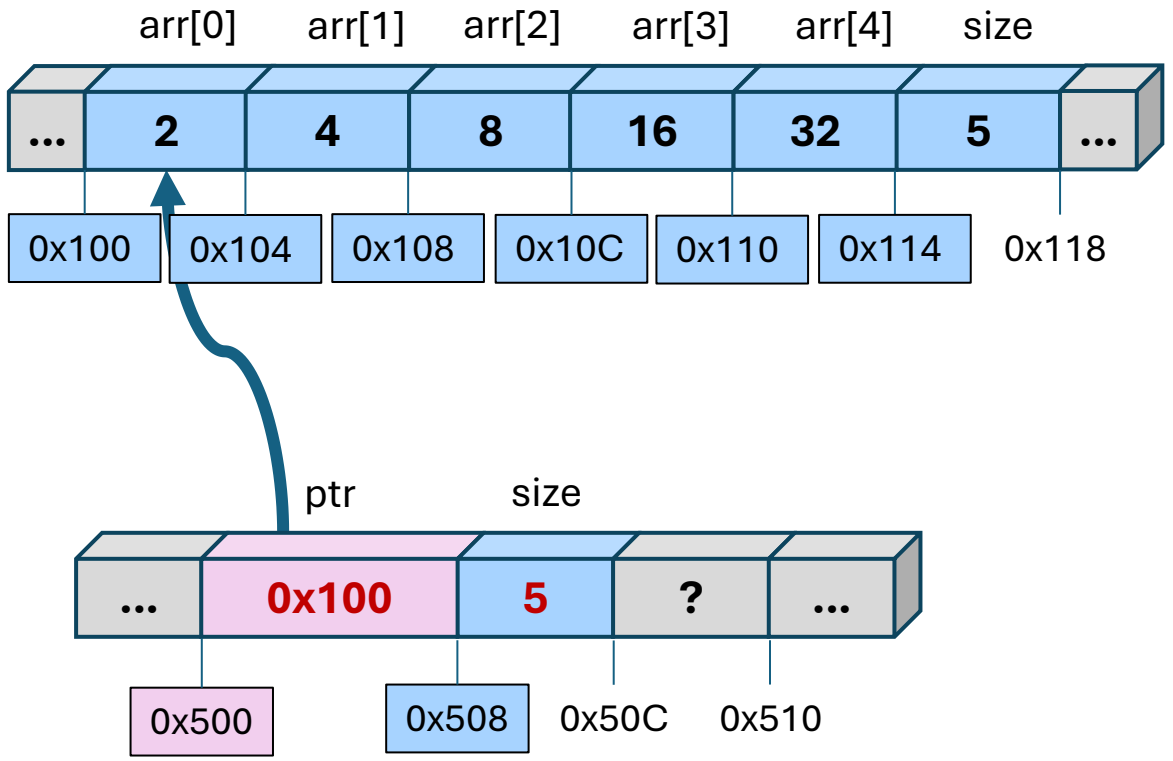
Working With Arrays and Pointer Arithmetic

```
int main() {
    int arr[] = {2, 4, 8, 16, 32};
    // This computes the size of the array
    int size = sizeof(arr) / sizeof(arr[0]);
    print_array(arr, size)
    return 0;
}

void print_array(int *ptr, int size) {
    for(int i=0; i<size; i++) {
        // We use pointer arithmetic here
        printf("%d ", *(ptr + i));
    }
}
```

We Are Here

Here



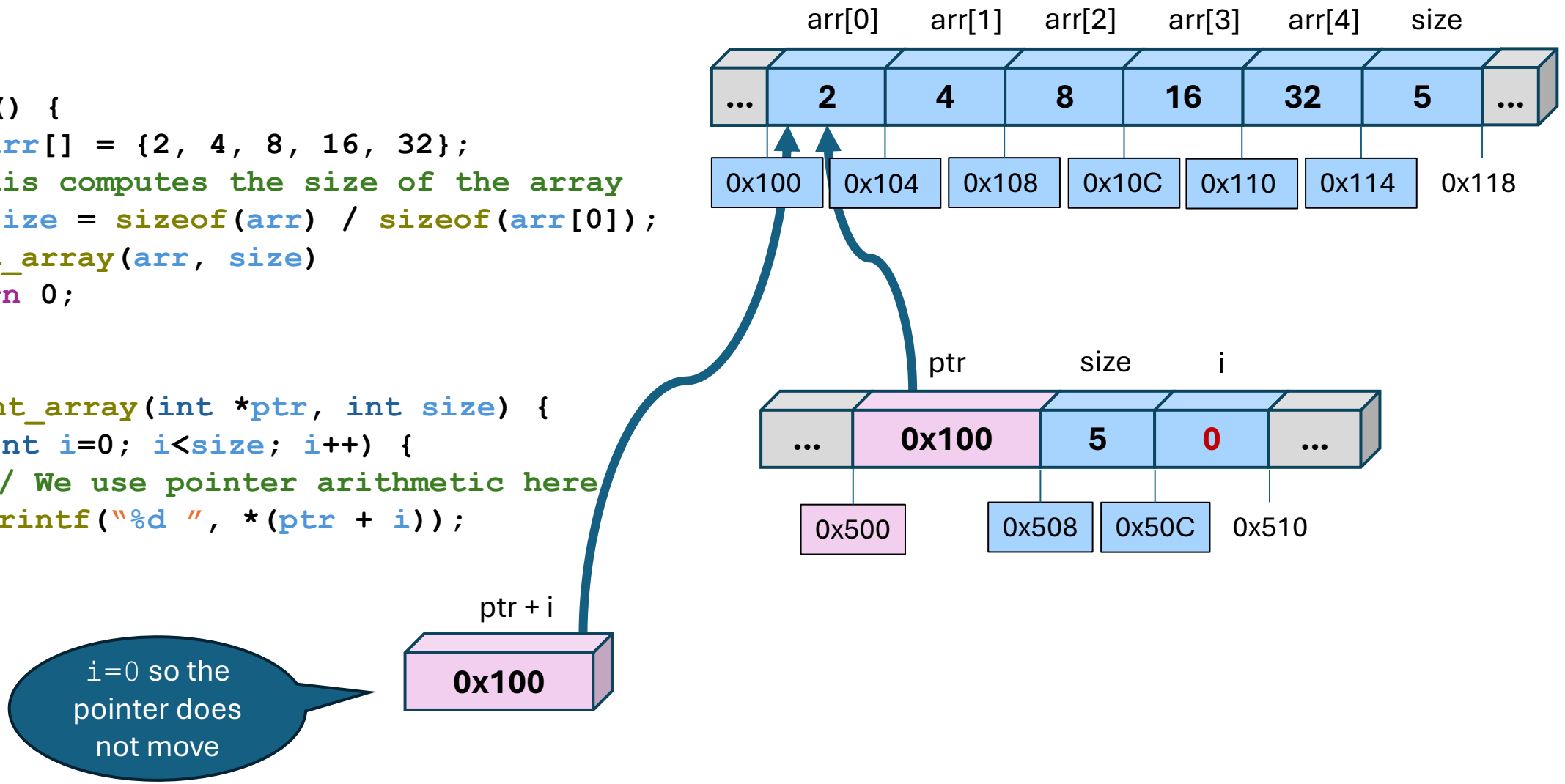
Working With Arrays and Pointer Arithmetic

```c
int main() {
    int arr[] = {2, 4, 8, 16, 32};
    // This computes the size of the array
    int size = sizeof(arr) / sizeof(arr[0]);
    print_array(arr, size)
    return 0;
}

void print_array(int *ptr, int size) {
    for(int i=0; i<size; i++) {
        // We use pointer arithmetic here
        printf("%d ", *(ptr + i));
    }
}
```

We Are Here

We Are Here



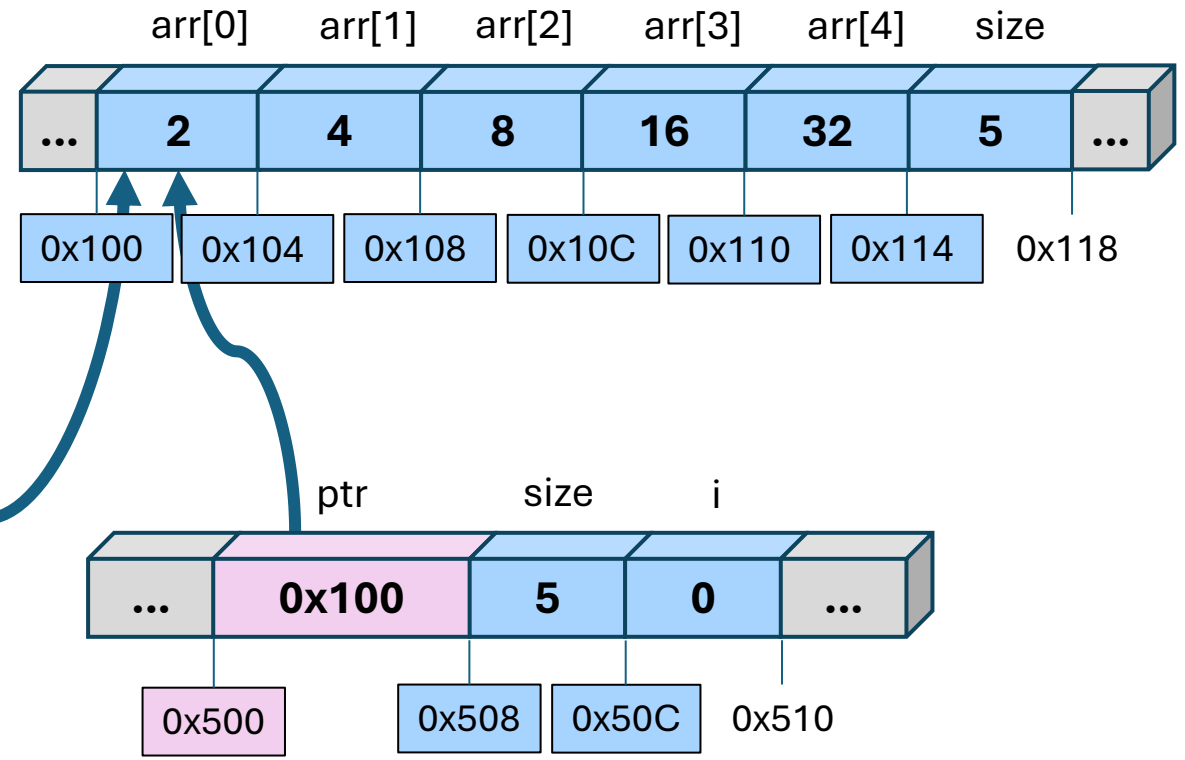
Working With Arrays and Pointer Arithmetic

```c
int main() {
    int arr[] = {2, 4, 8, 16, 32};
    // This computes the size of the array
    int size = sizeof(arr) / sizeof(arr[0]);
    print_array(arr, size)
    return 0;
}

void print_array(int *ptr, int size) {
    for(int i=0; i<size; i++) {
        // We use pointer arithmetic here
        printf("%d ", *(ptr + i));
    }
}
```

We Are Here

We Are Here



i=0 so the pointer does not move

Terminal

```
2
```

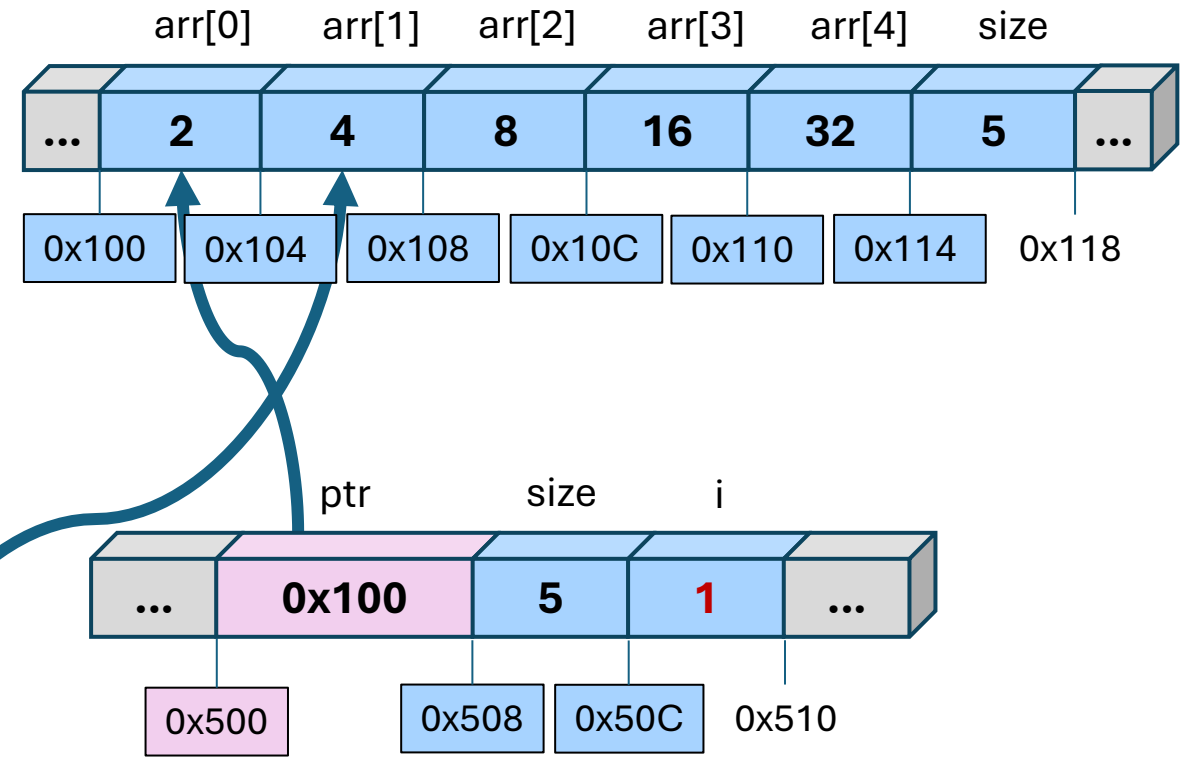
Working With Arrays and Pointer Arithmetic

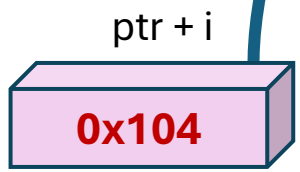
```c
int main() {
    int arr[] = {2, 4, 8, 16, 32};
    // This computes the size of the array
    int size = sizeof(arr) / sizeof(arr[0]);
    print_array(arr, size)
    return 0;
}

void print_array(int *ptr, int size) {
    for(int i=0; i<size; i++) {
        // We use pointer arithmetic here
        printf("%d ", *(ptr + i));
    }
}
```

We Are Here

We Are Here





i=1 so the pointer moves 4 bytes

Terminal

```
2
```

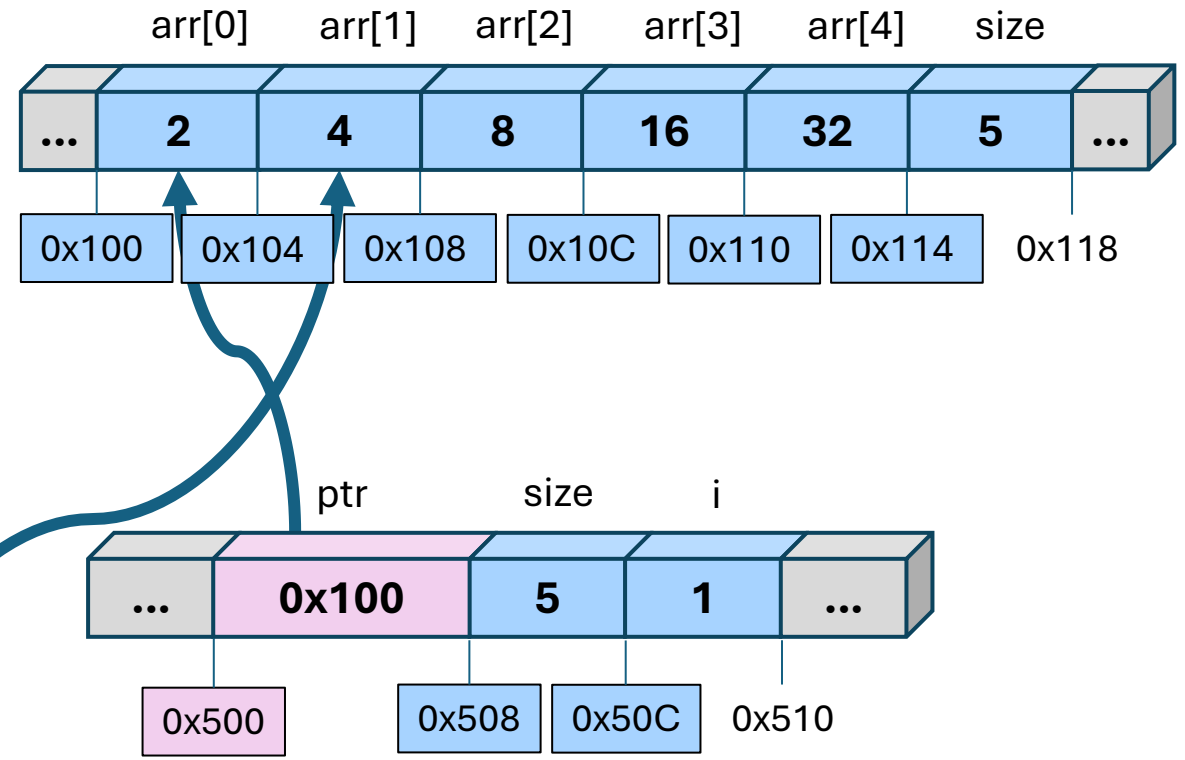
Working With Arrays and Pointer Arithmetic

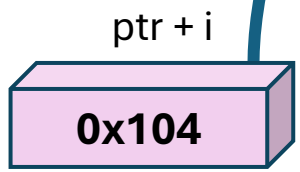
```c
int main() {
    int arr[] = {2, 4, 8, 16, 32};
    // This computes the size of the array
    int size = sizeof(arr) / sizeof(arr[0]);
    print_array(arr, size)
    return 0;
}

void print_array(int *ptr, int size) {
    for(int i=0; i<size; i++) {
        // We use pointer arithmetic here
        printf("%d ", *(ptr + i));
    }
}
```

We Are Here

We Are Here





i=1 so the pointer moves 4 bytes

Terminal

```
2 4
```

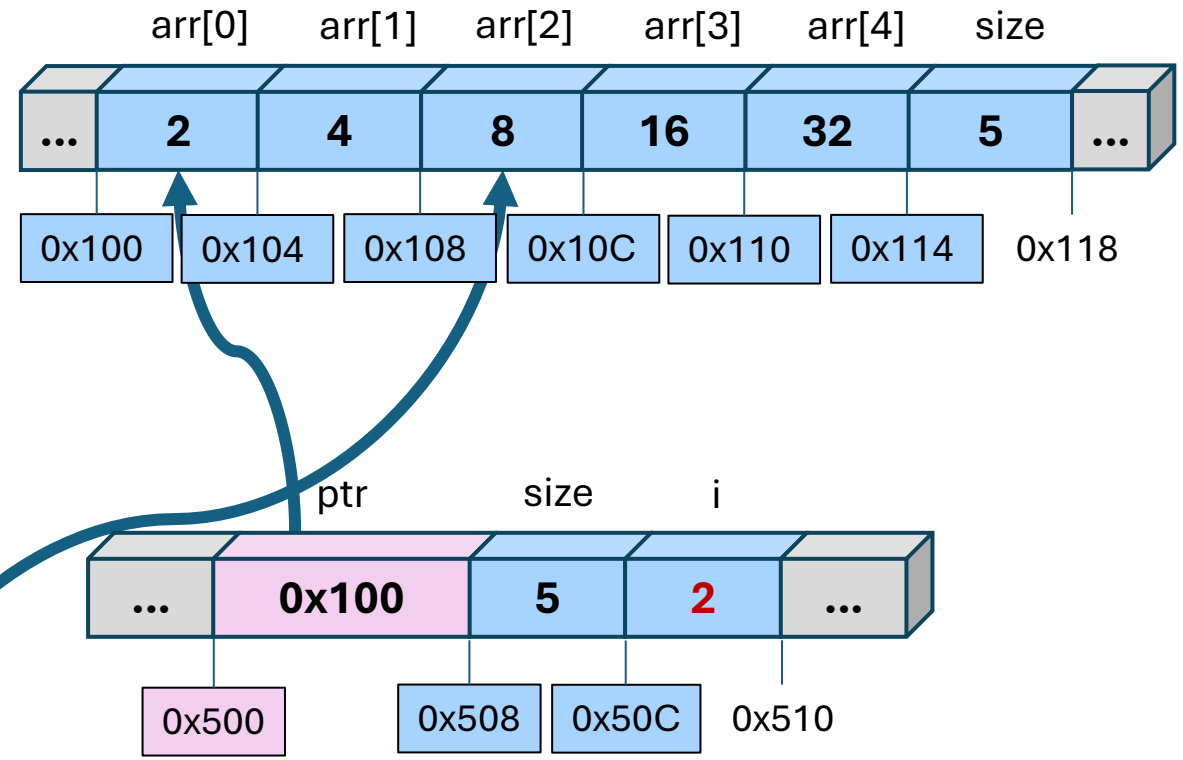
Working With Arrays and Pointer Arithmetic

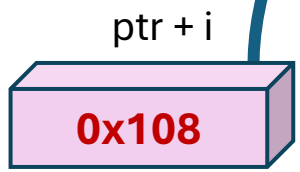
```c
int main() {
    int arr[] = {2, 4, 8, 16, 32};
    // This computes the size of the array
    int size = sizeof(arr) / sizeof(arr[0]);
    print_array(arr, size)
    return 0;
}

void print_array(int *ptr, int size) {
    for(int i=0; i<size; i++) {
        // We use pointer arithmetic here
        printf("%d ", *(ptr + i));
    }
}
```

We Are Here

We Are Here





i=2 so the pointer moves 8 bytes

Terminal

```
2 4
```

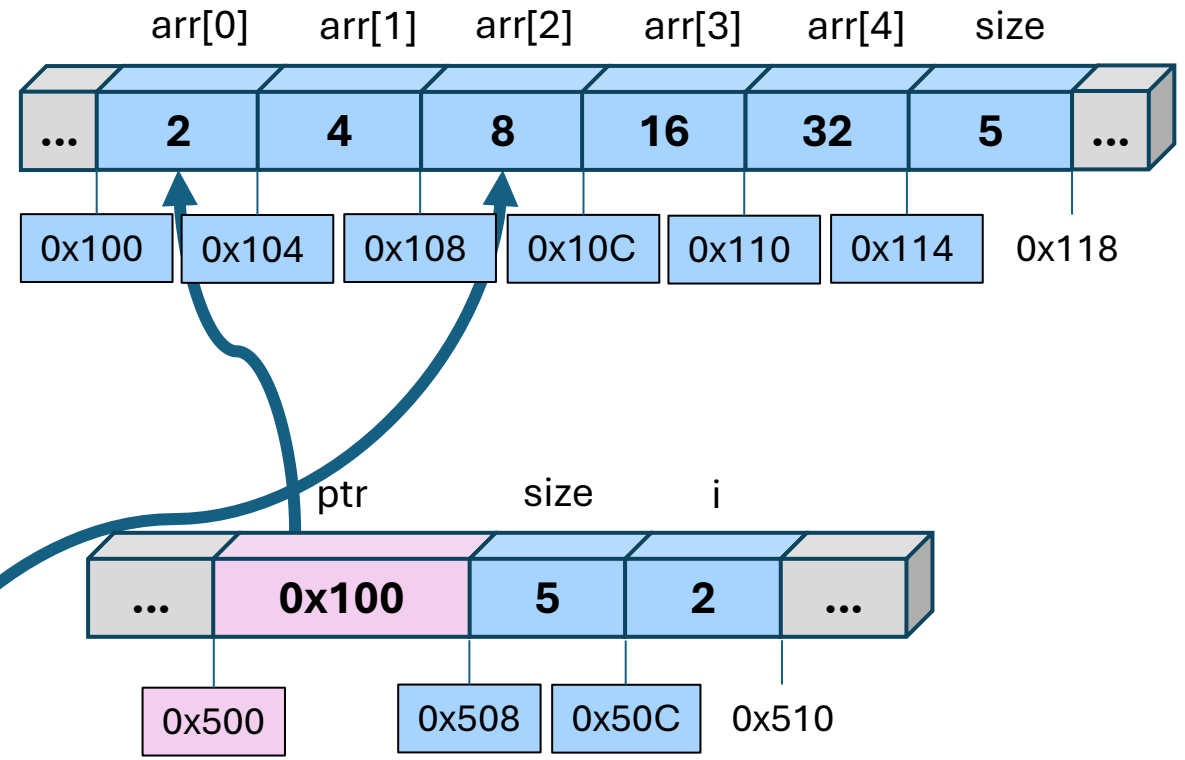
Working With Arrays and Pointer Arithmetic

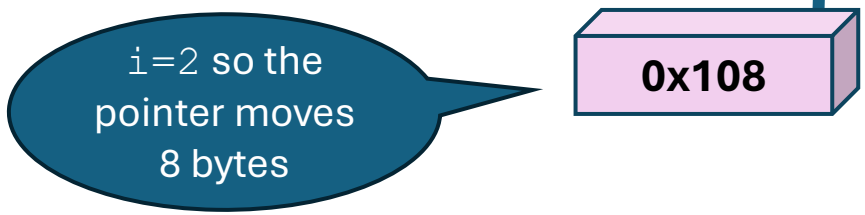
```c
int main() {
    int arr[] = {2, 4, 8, 16, 32};
    // This computes the size of the array
    int size = sizeof(arr) / sizeof(arr[0]);
    print_array(arr, size)
    return 0;
}

void print_array(int *ptr, int size) {
    for(int i=0; i<size; i++) {
        // We use pointer arithmetic here
        printf("%d ", *(ptr + i));
    }
}
```

We Are Here

We Are Here





Terminal

```
2 4 8
```

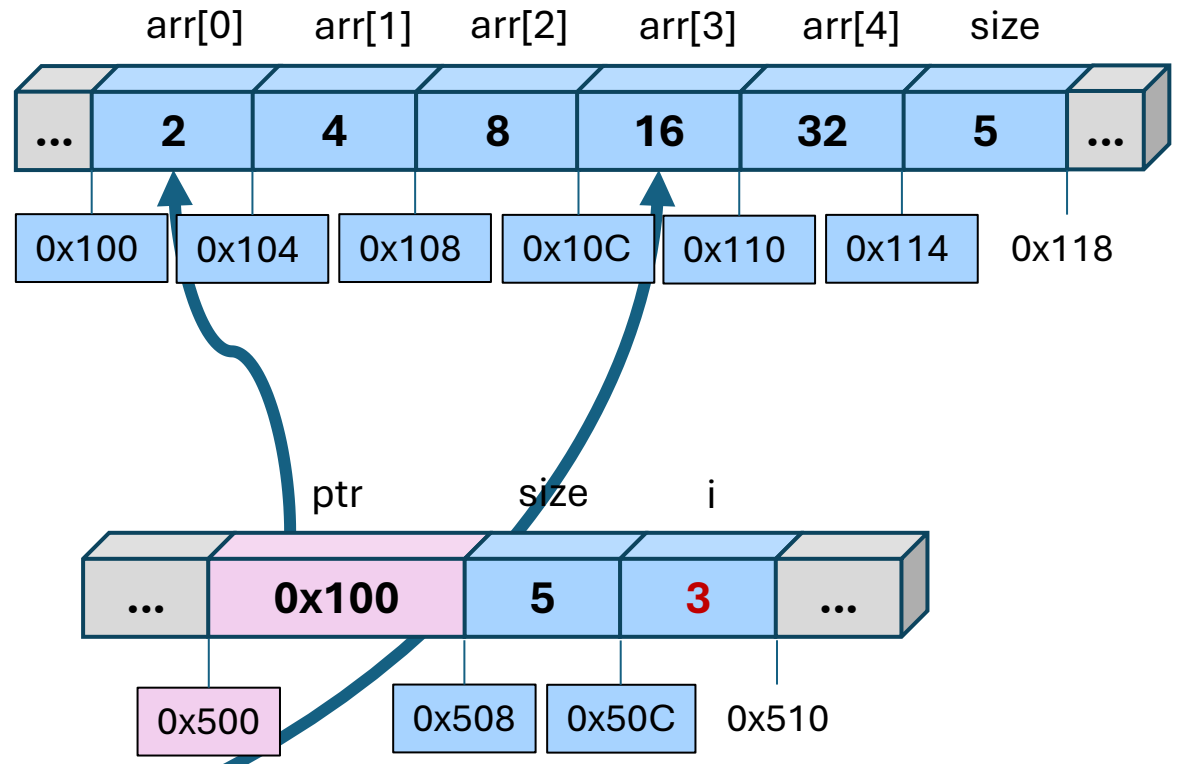
Working With Arrays and Pointer Arithmetic

```c
int main() {
    int arr[] = {2, 4, 8, 16, 32};
    // This computes the size of the array
    int size = sizeof(arr) / sizeof(arr[0]);
    print_array(arr, size)
    return 0;
}

void print_array(int *ptr, int size) {
    for(int i=0; i<size; i++) {
        // We use pointer arithmetic here
        printf("%d ", *(ptr + i));
    }
}
```

We Are Here

We Are Here



ptr + i

0x10C

i=3 so the pointer moves 12 bytes

Terminal

```
2 4 8
```

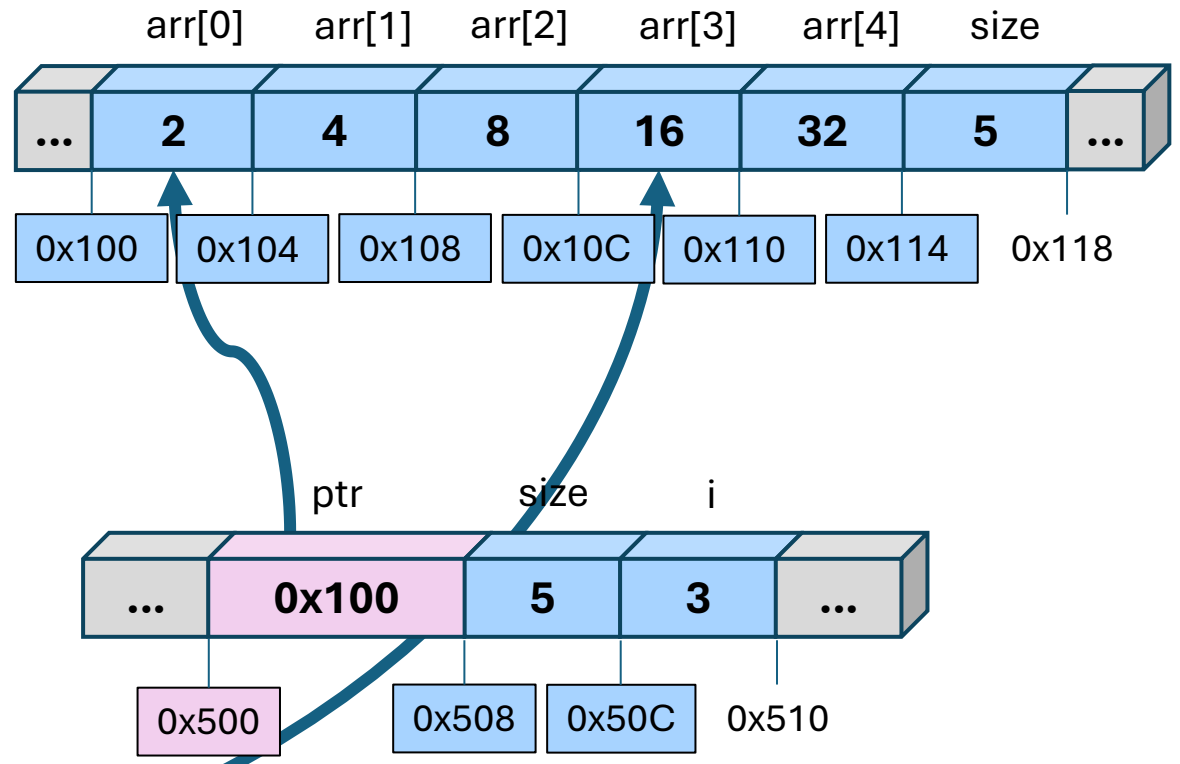
Working With Arrays and Pointer Arithmetic

```c
int main() {
    int arr[] = {2, 4, 8, 16, 32};
    // This computes the size of the array
    int size = sizeof(arr) / sizeof(arr[0]);
    print_array(arr, size)
    return 0;
}

void print_array(int *ptr, int size) {
    for(int i=0; i<size; i++) {
        // We use pointer arithmetic here
        printf("%d ", *(ptr + i));
    }
}
```

We Are Here

We Are Here



i=3 so the pointer moves 12 bytes

Terminal

```
2 4 8 16
```

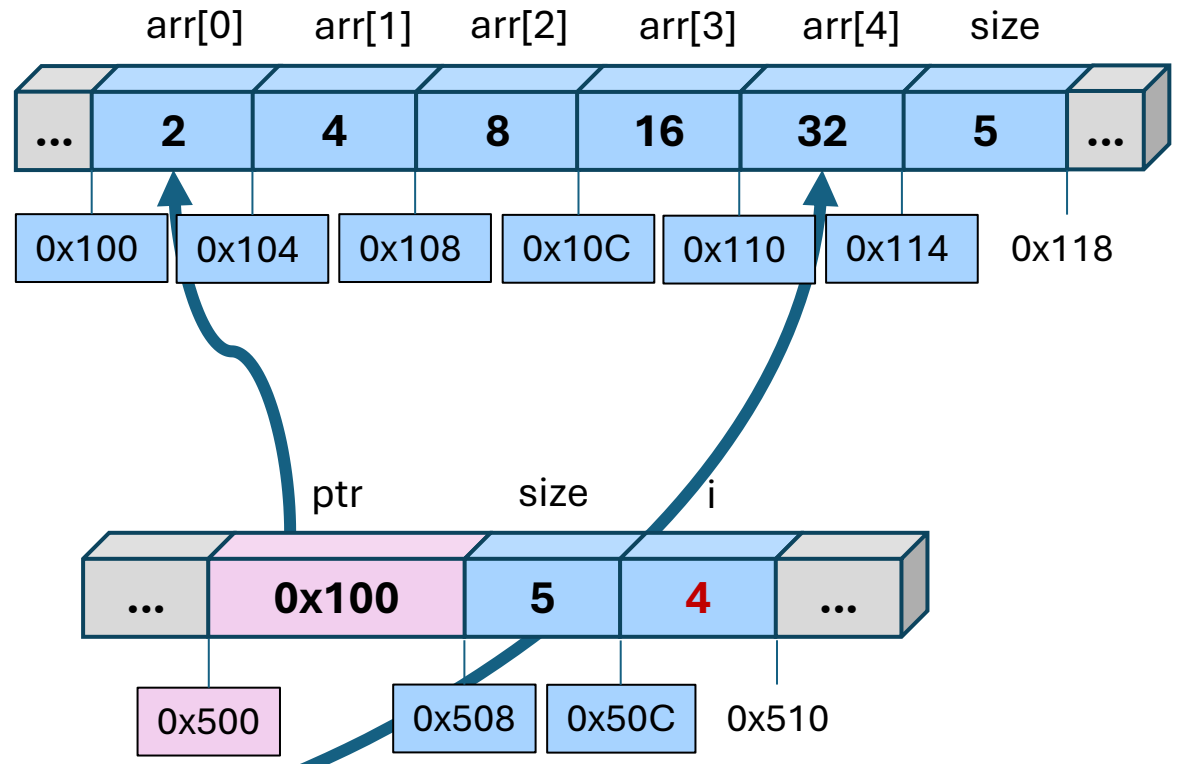
Working With Arrays and Pointer Arithmetic

```c
int main() {
    int arr[] = {2, 4, 8, 16, 32};
    // This computes the size of the array
    int size = sizeof(arr) / sizeof(arr[0]);
    print_array(arr, size)
    return 0;
}

void print_array(int *ptr, int size) {
    for(int i=0; i<size; i++) {
        // We use pointer arithmetic here
        printf("%d ", *(ptr + i));
    }
}
```

We Are Here

We Are Here



i=4 so the pointer moves 16 bytes

Terminal

```
2 4 8 16
```

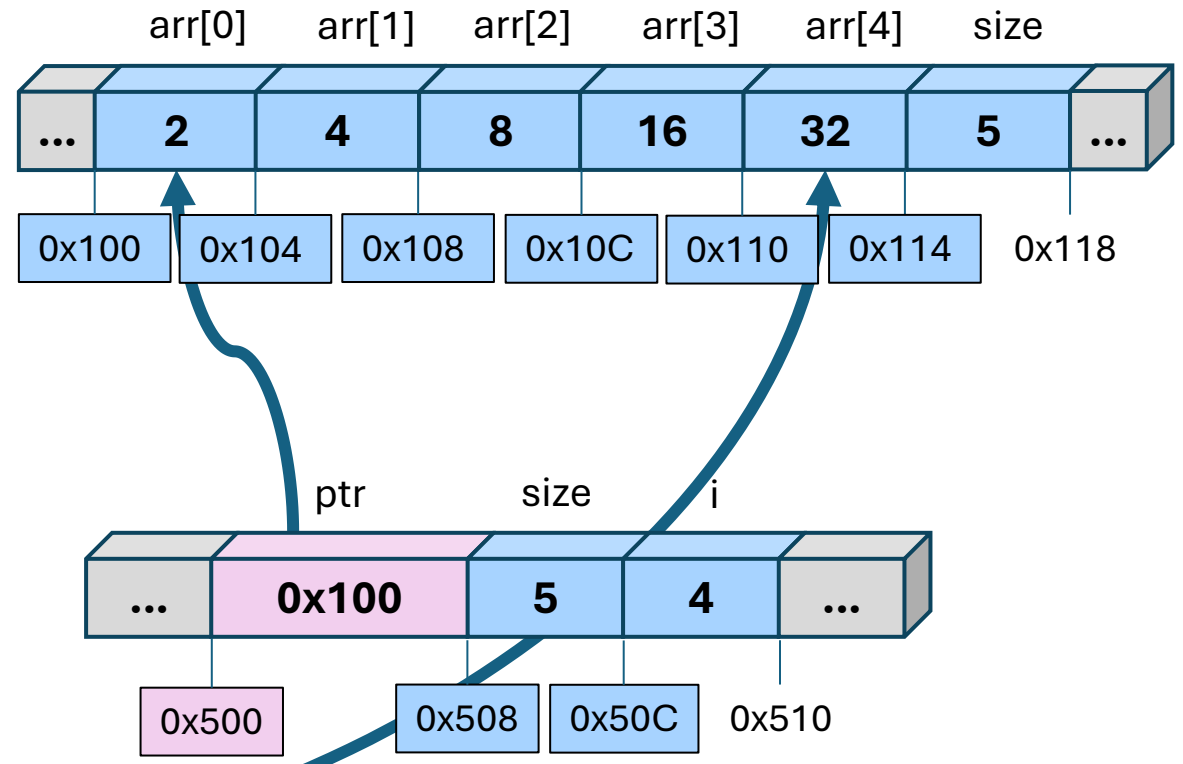
Working With Arrays and Pointer Arithmetic

```c
int main() {
    int arr[] = {2, 4, 8, 16, 32};
    // This computes the size of the array
    int size = sizeof(arr) / sizeof(arr[0]);
    print_array(arr, size)
    return 0;
}

void print_array(int *ptr, int size) {
    for(int i=0; i<size; i++) {
        // We use pointer arithmetic here
        printf("%d ", *(ptr + i));
    }
}
```

We Are Here

We Are Here



ptr + i

0x110

i=4 so the pointer moves 16 bytes

Terminal

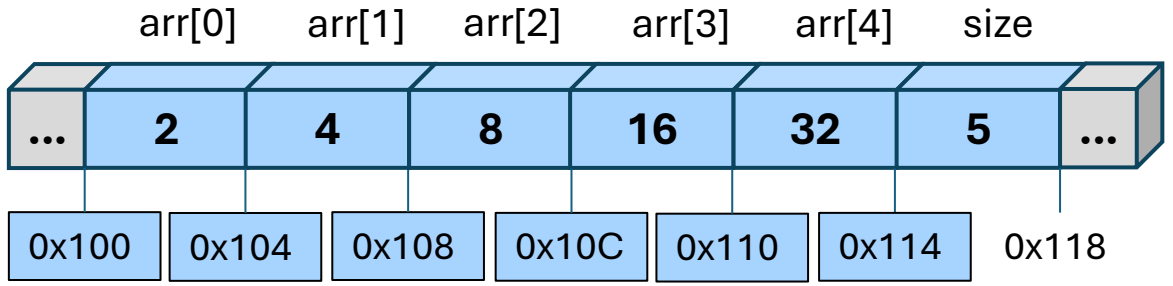
```
2 4 8 16 32
```

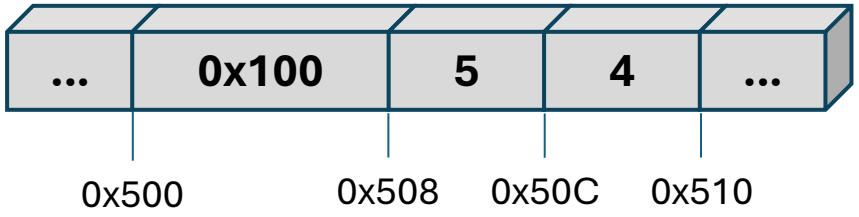
Working With Arrays and Pointer Arithmetic

```
int main() {
    int arr[] = {2, 4, 8, 16, 32};
    // This computes the size of the array
    int size = sizeof(arr) / sizeof(arr[0]);
    print_array(arr, size)
    return 0;
}

void print_array(int *ptr, int size) {
    for(int i=0; i<size; i++) {
        // We use pointer arithmetic here
        printf("%d ", *(ptr + i));
    }
}
```

We Are Here





Terminal

```
2 4 8 16 32
```

Functions - Call By Reference

Call By Reference

Rather than passing copies of the variables to a function, call by reference passes the **memory address** of a variable

Guess what – we've already done this with sscanf

```
sscanf(buffer, "%d", &val);
```

`&val` is the memory address of the variable val

Functions - Call By Reference

Call By Reference

The parameters are received as references to the actual arguments, allowing the function to directly modify the original values

```
int main() {
    int num = 5;
    modify(&num);
    printf("%d\n", num);
    return 0;
}
void modify(int *x) {
    (*x)++;
}
```

`&num` is the memory address of num

`(*x)++` increments the value stored at the address by 1

Swapping Two Variables By Reference

```c
int main() {
    int a = 5;
    int b = 67;
    printf("a = %d, b = %d\n", a, b);
    // Prints a = 5, b = 20

    swap(&a, &b);

    printf("a = %d, b = %d\n", a, b);
    // Prints a = 20, b = 5
    return 0;
}
```

```c
void swap(int *x, int *y) {
    int temp = *x;
    *x = *y;
    *y = temp;
    return;
}
```

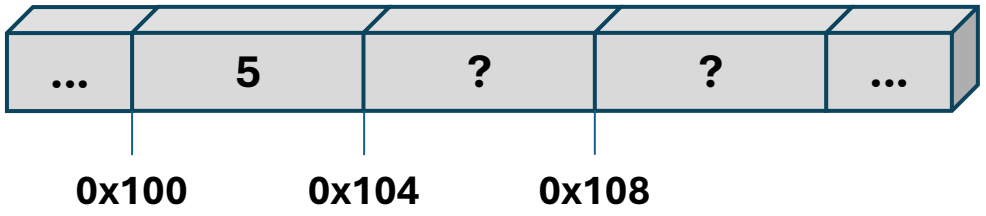
Swapping Two Variables By Reference

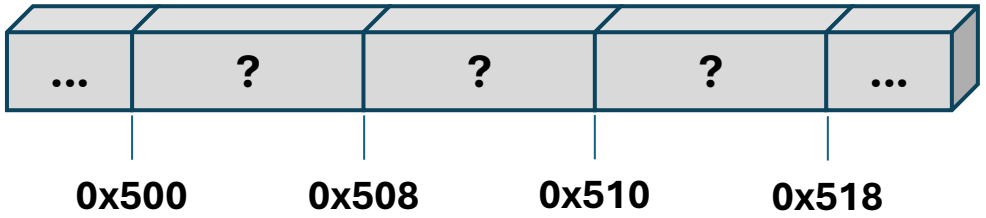
Here

```c
int main() {
    int a = 5;
    int b = 67;
    printf("a = %d, b = %d\n", a, b);
    // Prints a = 5, b = 67

    swap(&a, &b);

    printf("a = %d, b = %d\n", a, b);
    // Prints a = 67, b = 5
    return 0;
}
void swap(int *x, int *y) {
    int temp = *x;
    *x = *y;
    *y = temp;
    return;
}
```





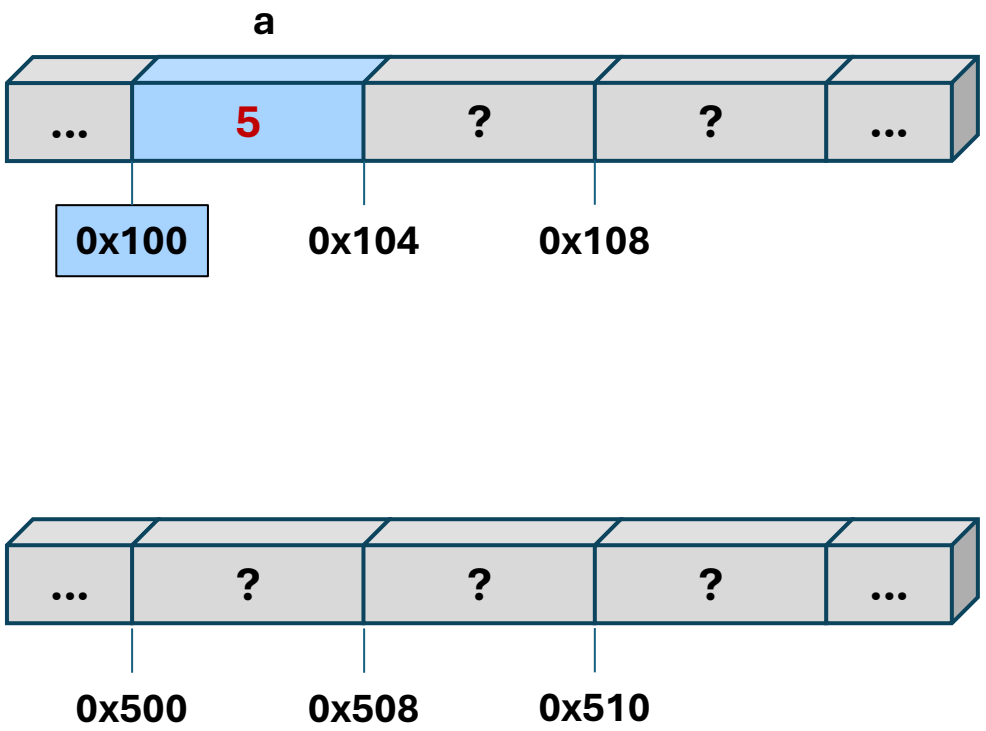
Swapping Two Variables By Reference

We Are Here

```
int main() {
    int a = 5;
    int b = 67;
    printf("a = %d, b = %d\n", a, b);
    // Prints a = 5, b = 67

    swap(&a, &b);

    printf("a = %d, b = %d\n", a, b);
    // Prints a = 67, b = 5
    return 0;
}
void swap(int *x, int *y) {
    int temp = *x;
    *x = *y;
    *y = temp;
    return;
}
```



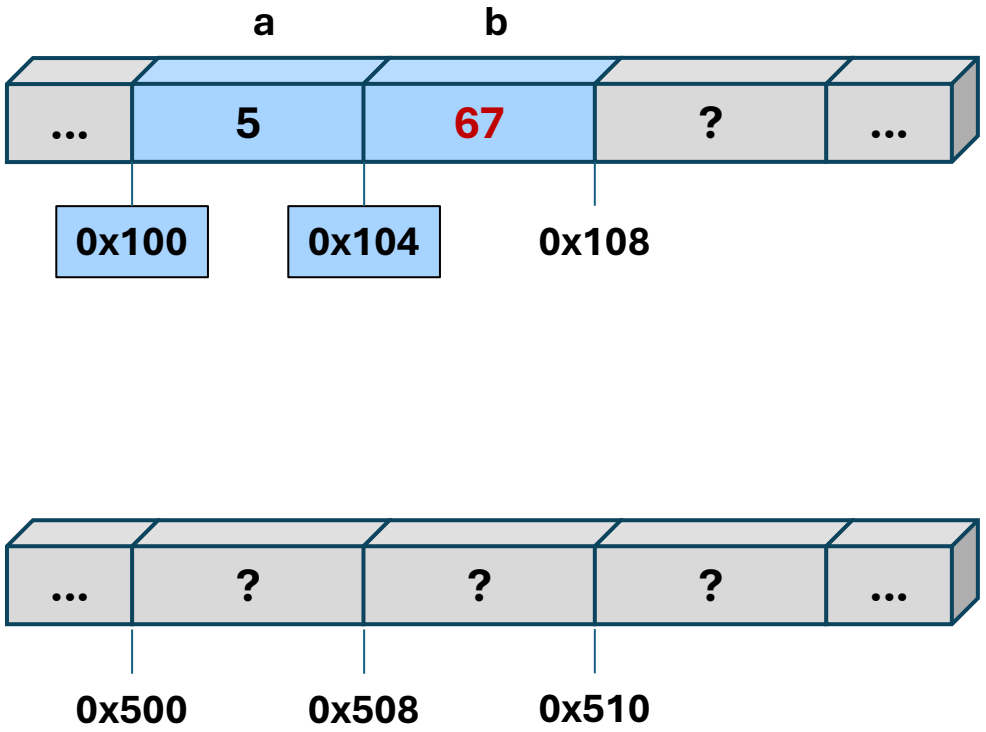
Swapping Two Variables By Reference

We Are Here

```
int main() {
    int a = 5;
    int b = 67;
    printf("a = %d, b = %d\n", a, b);
    // Prints a = 5, b = 67

    swap(&a, &b);

    printf("a = %d, b = %d\n", a, b);
    // Prints a = 67, b = 5
    return 0;
}
void swap(int *x, int *y) {
    int temp = *x;
    *x = *y;
    *y = temp;
    return;
}
```



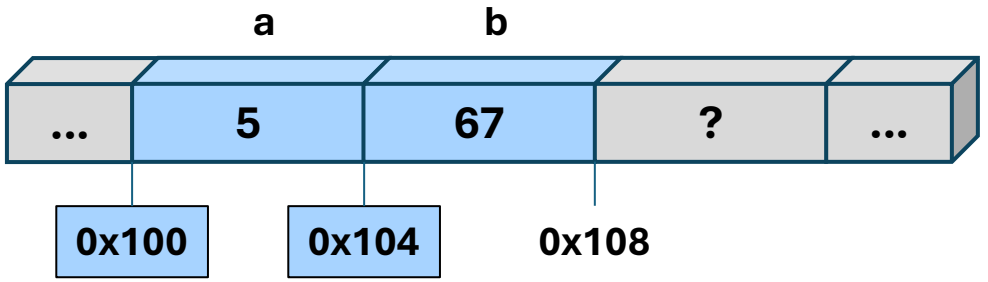
Swapping Two Variables By Reference

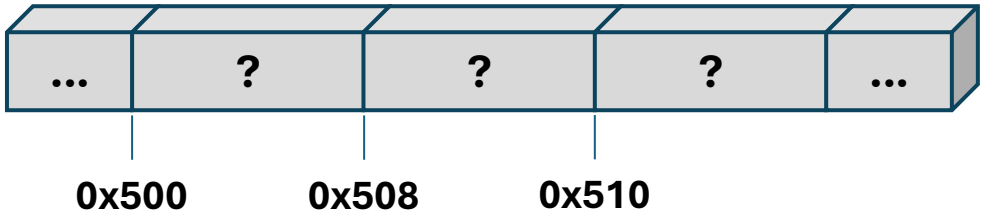
```c
int main() {
    int a = 5;
    int b = 67;
    printf("a = %d, b = %d\n", a, b);
    // Prints a = 5, b = 67

    swap(&a, &b);

    printf("a = %d, b = %d\n", a, b);
    // Prints a = 67, b = 5
    return 0;
}
void swap(int *x, int *y) {
    int temp = *x;
    *x = *y;
    *y = temp;
    return;
}
```

We Are Here





Terminal

```
a = 5, b = 67
```

Swapping Two Variables By Reference

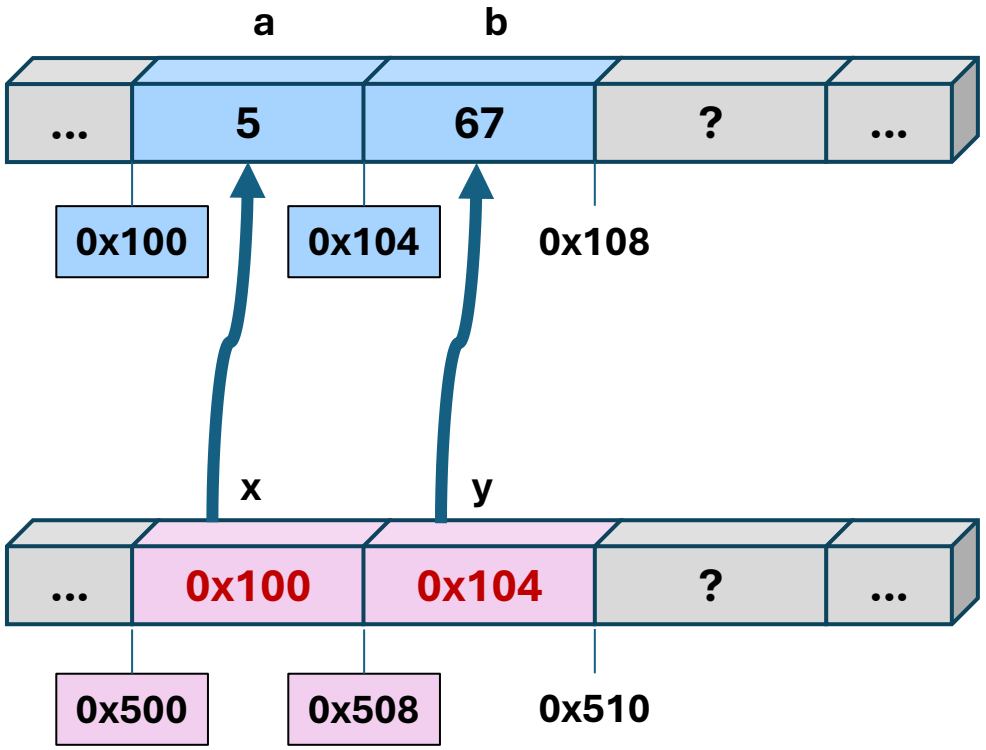
```
int main() {
    int a = 5;
    int b = 67;
    printf("a = %d, b = %d\n", a, b);
    // Prints a = 5, b = 67

    swap(&a, &b);

    printf("a = %d, b = %d\n", a, b);
    // Prints a = 67, b = 5
    return 0;
}
void swap(int *x, int *y) {
    int temp = *x;
    *x = *y;
    *y = temp;
    return;
}
```

We Are Here

Here



Swapping Two Variables By Reference

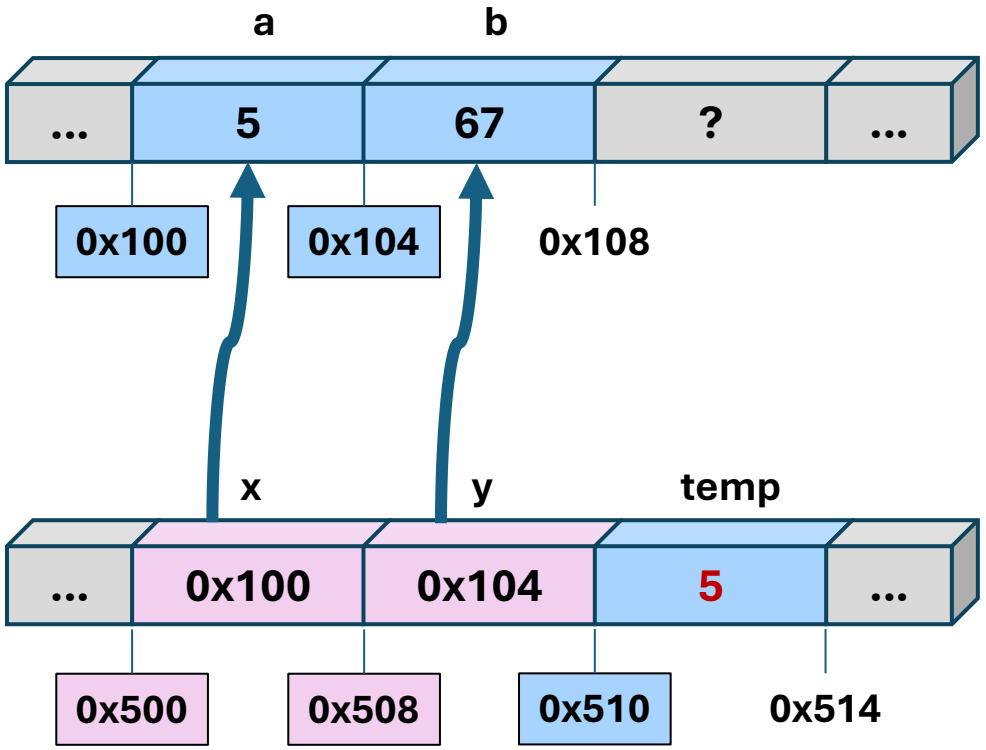
```
int main() {
    int a = 5;
    int b = 67;
    printf("a = %d, b = %d\n", a, b);
    // Prints a = 5, b = 67

    swap(&a, &b);

    printf("a = %d, b = %d\n", a, b);
    // Prints a = 67, b = 5
    return 0;
}
void swap(int *x, int *y) {
    int temp = *x;
    *x = *y;
    *y = temp;
    return;
}
```

We Are Here

We Are Here



Swapping Two Variables By Reference

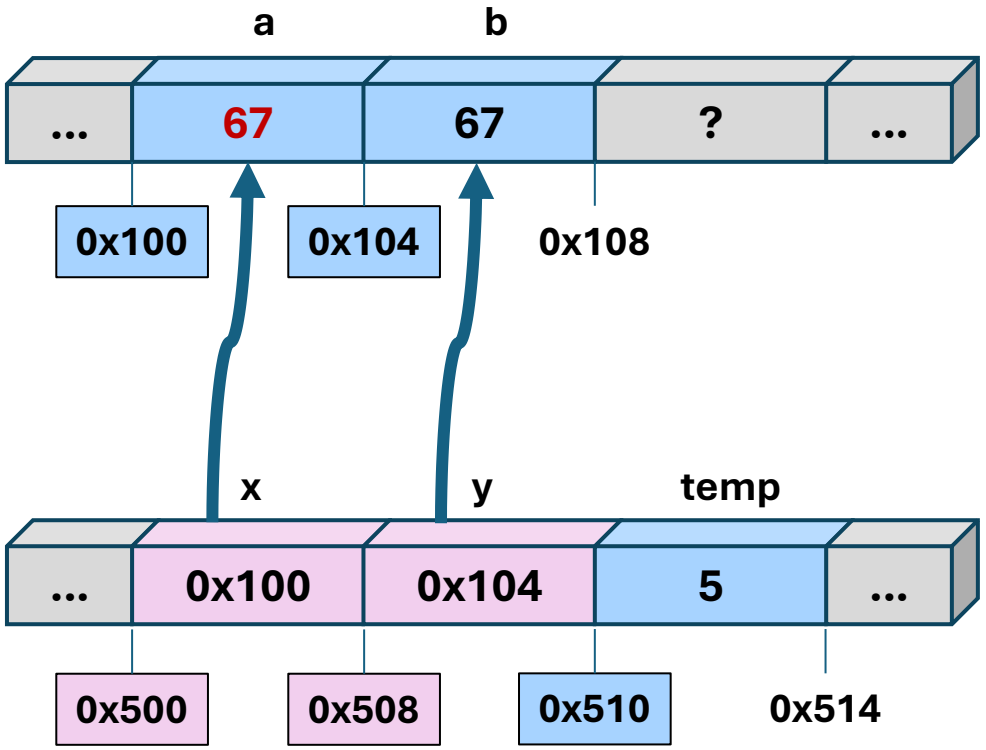
```
int main() {
    int a = 5;
    int b = 67;
    printf("a = %d, b = %d\n", a, b);
    // Prints a = 5, b = 67

    swap(&a, &b);

    printf("a = %d, b = %d\n", a, b);
    // Prints a = 67, b = 5
    return 0;
}
void swap(int *x, int *y) {
    int temp = *x;
    *x = *y;
    *y = temp;
    return;
}
```

We Are Here

We Are Here



Swapping Two Variables By Reference

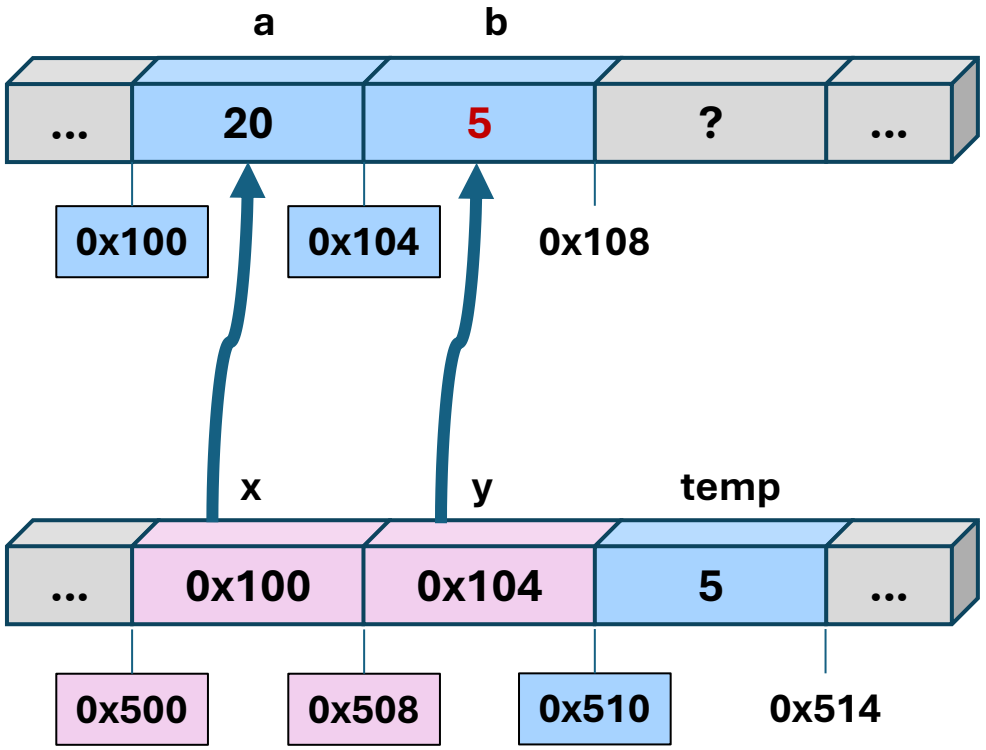
```
int main() {
    int a = 5;
    int b = 67;
    printf("a = %d, b = %d\n", a, b);
    // Prints a = 5, b = 67

    swap(&a, &b);

    printf("a = %d, b = %d\n", a, b);
    // Prints a = 67, b = 5
    return 0;
}
void swap(int *x, int *y) {
    int temp = *x;
    *x = *y;
    *y = temp;
    return;
}
```

We Are Here

We Are Here



Swapping Two Variables By Reference

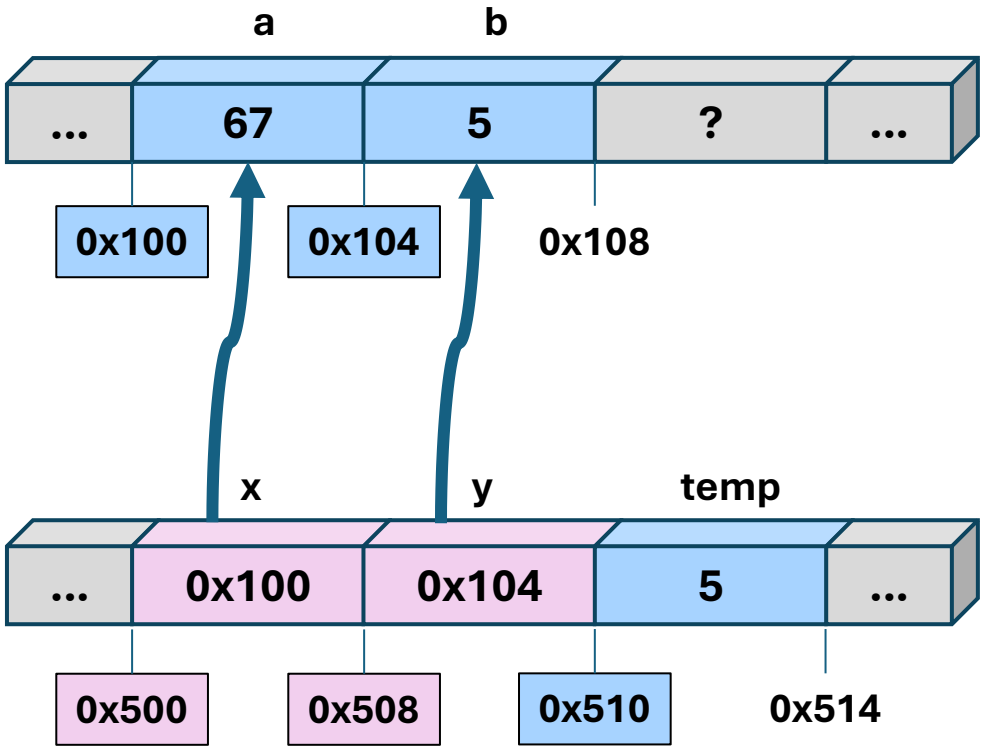
```c
int main() {
    int a = 5;
    int b = 67;
    printf("a = %d, b = %d\n", a, b);
    // Prints a = 5, b = 67

    swap(&a, &b);

    printf("a = %d, b = %d\n", a, b);
    // Prints a = 67, b = 5
    return 0;
}
void swap(int *x, int *y) {
    int temp = *x;
    *x = *y;
    *y = temp;
    return;
}
```

We Are Here

We Are Here



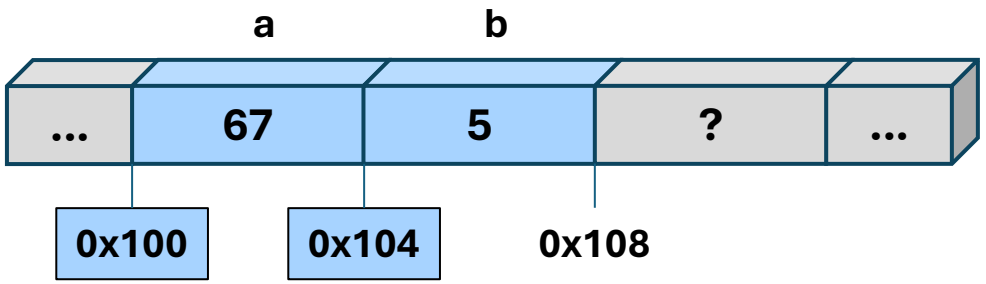
Swapping Two Variables By Reference

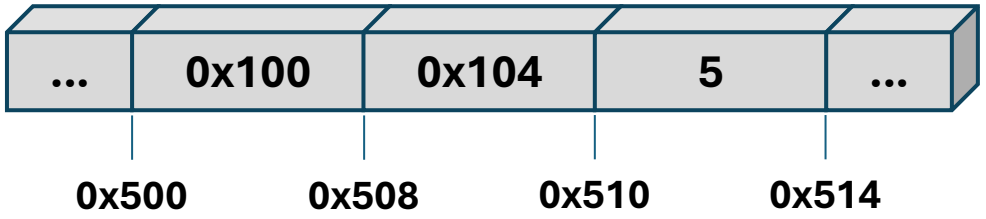
```c
int main() {
    int a = 5;
    int b = 67;
    printf("a = %d, b = %d\n", a, b);
    // Prints a = 5, b = 67

    swap(&a, &b);

    printf("a = %d, b = %d\n", a, b);
    // Prints a = 67, b = 5
    return 0;
}
void swap(int *x, int *y) {
    int temp = *x;
    *x = *y;
    *y = temp;
    return;
}
```

We Are Here





Terminal

```
a = 67, b = 5
```

Practice

Let's Practice

Pay close attention – These resemble what you will see on quizzes and exams

Pointers are 4 bytes on a 32-bit system and 8 bytes on a 64-bit system. Why do you think this is the case?