CSCI2467: Systems Programming Concepts

Pointers

Course Instructors:

Matthew Toups
Caitlin Boyce

Course Assistants:

Saroj Duwal
David McDonald

Spring 2020

THE UNIVERSITY *of*
NEW ORLEANS

DEPARTMENT OF
COMPUTER SCIENCE

Pointers in C

C Code

```
int x;
int *p;

x = 99;    //holds a value
p = &x;    //holds an address of a value
```

Operator	Function
&	“address of”

Source: wchapman.net

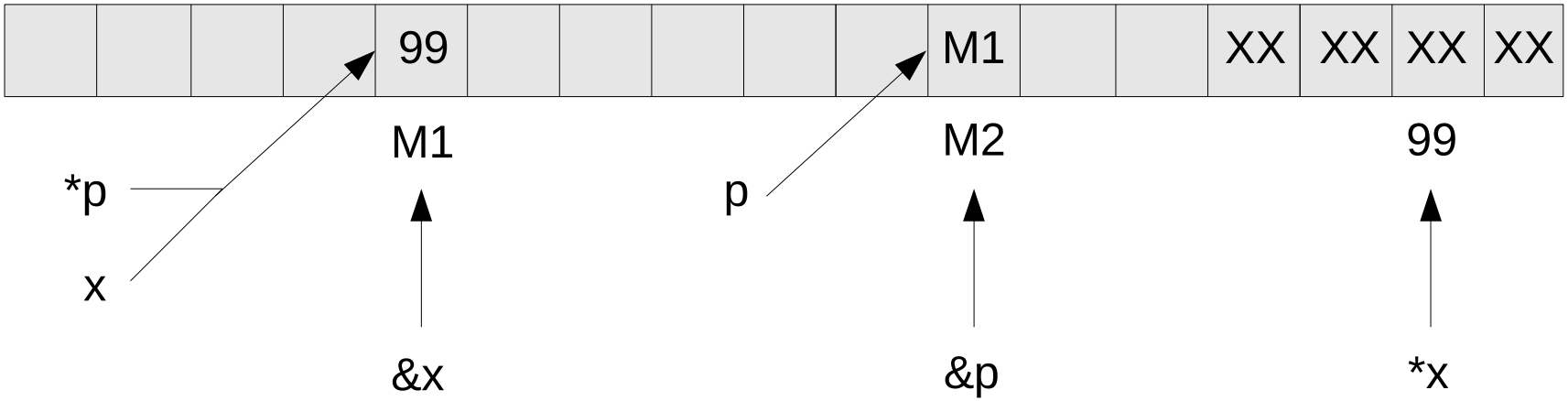
Pointers in C

C Code

```
int x;
int *p;

x = 99;    //holds a value
p = &x;    //holds an address of a value
```

Memory



Source: wchapman.net

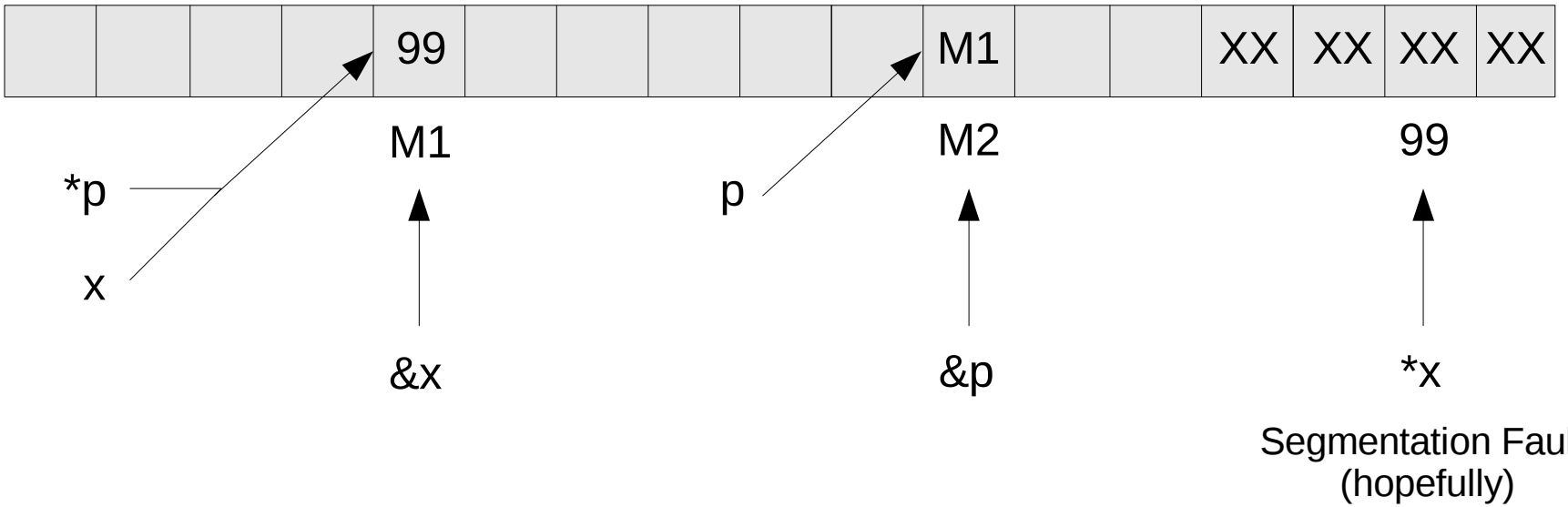
Pointers in C

C Code

```
int x;
int *p;

x = 99;    //holds a value
p = &x;    //holds an address of a value
```

Memory



C pointer syntax

Operator	Function
*	pointer / dereference
&	"address of"

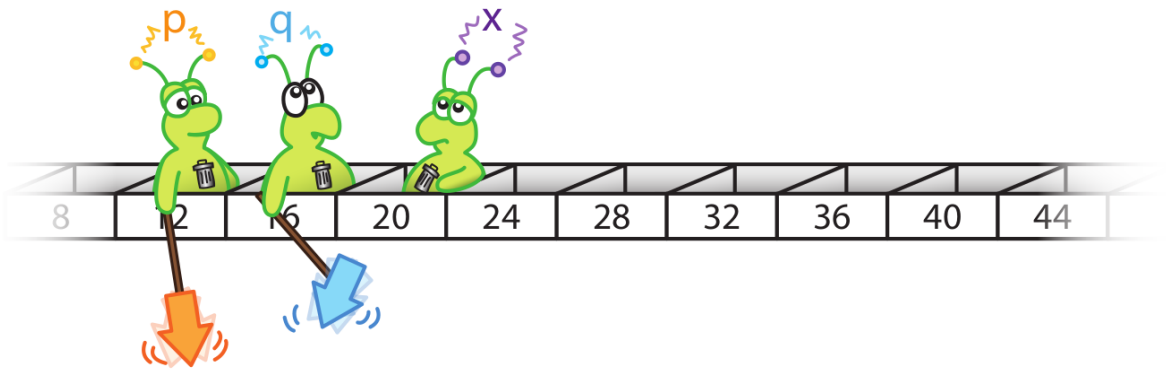
```
int x = 1, y = 2, z[10];
int *ip; /* ip is a pointer to int */

ip = &x; /* ip now points to x */
y = *ip; /* y is now 1 */
*ip = 0; /* x is now 0 */
ip = &z[0]; /* ip now points to z[0] */
```

Source: K&R Chapter 5

Pointers illustrated

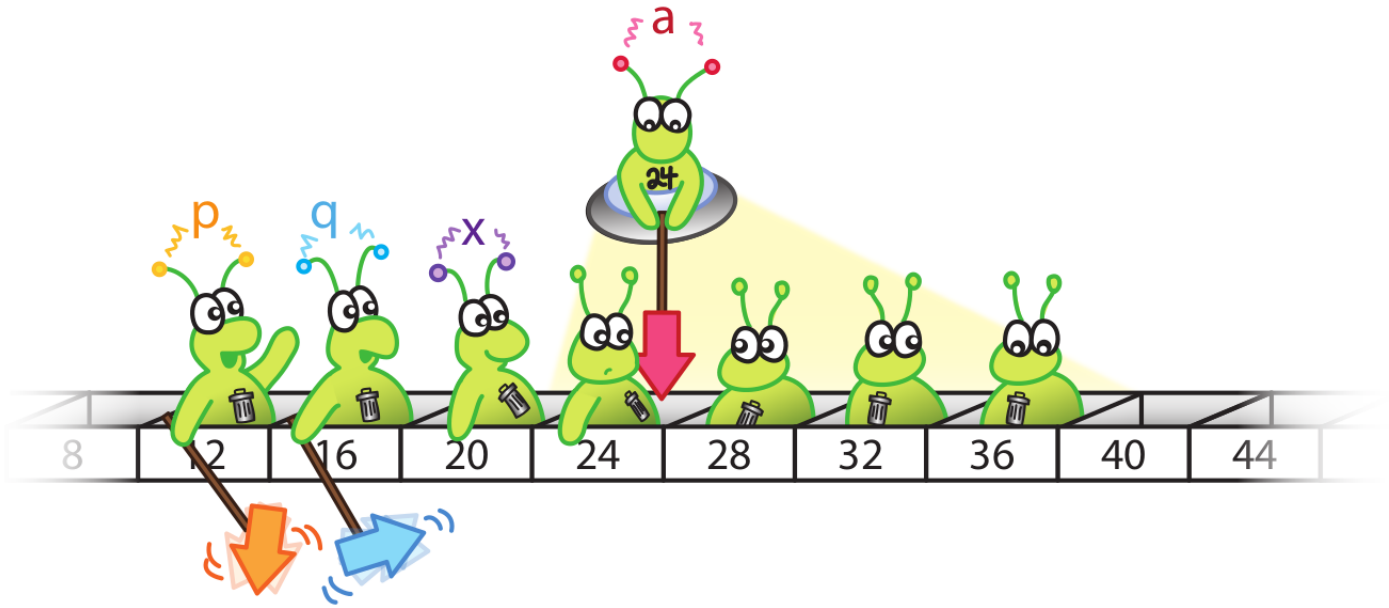
Imagine memory as a long block of boxes that store data. Each box is labeled with an **address**. A **pointer** is a variable that holds a particular address. An **array** is a group of contiguous boxes that can be accessed by their index values.




```
int *p, *q, x;
```

Here we declare p and q as pointers that will hold the *addresses* of `int` variables, and `x` as an ordinary `int` variable.

Pointers illustrated

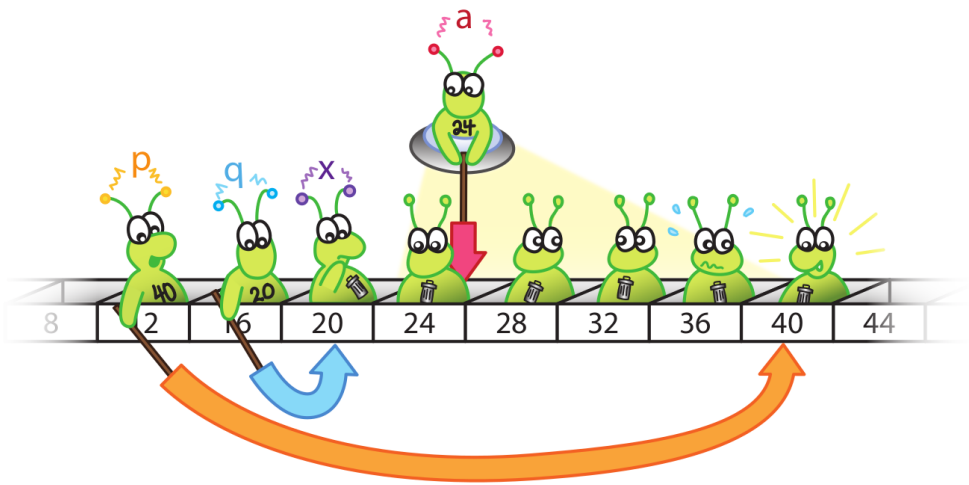



```
int a[4];
```

This line defines an array that can store four `int` values.
Now, a points to the first index of this array.

(None of the variables have been assigned values yet, so they contain "garbage" – whatever had been stored in these blocks of memory before)

Pointers illustrated



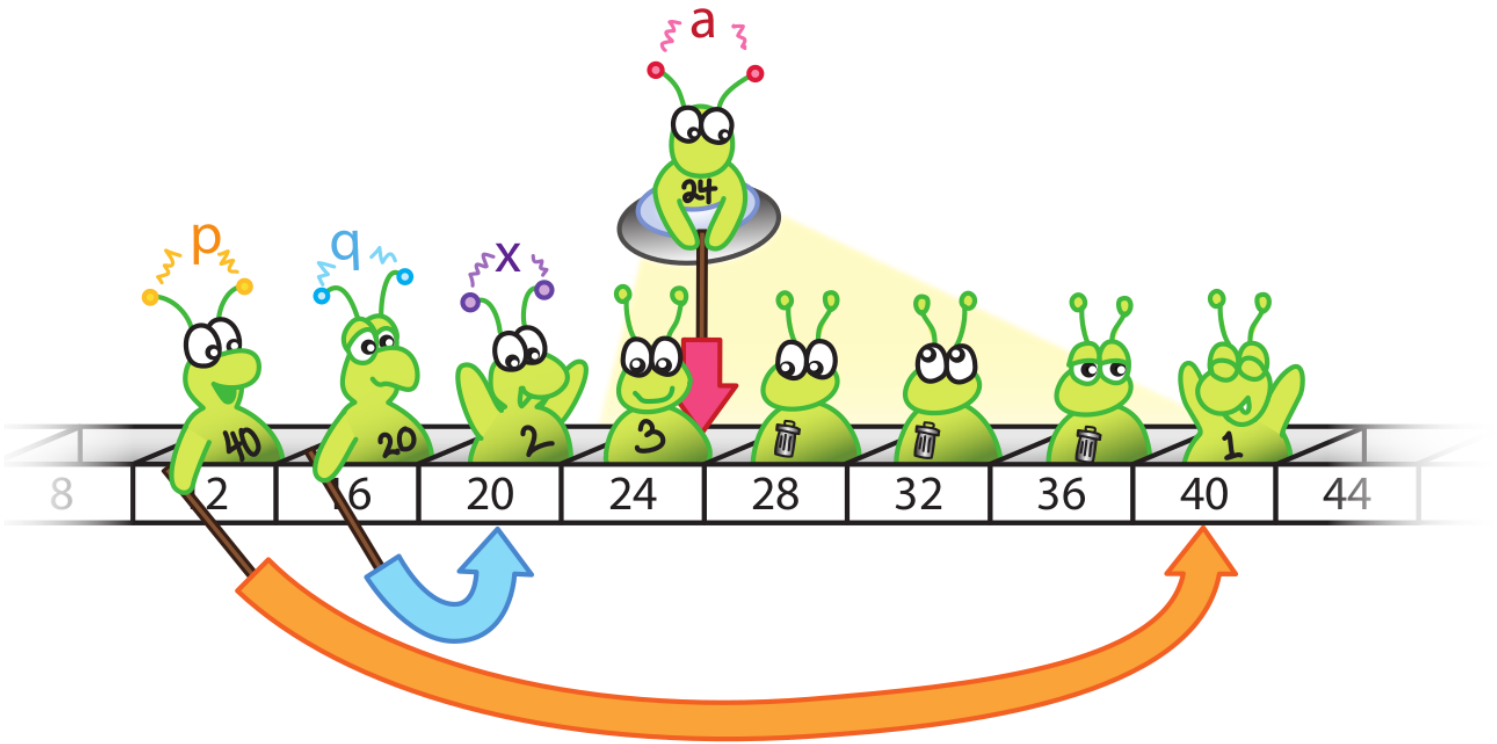

```
p = (int*) malloc(sizeof(int));
q = &x;
```

Line 1 requests 4 bytes (enough for one `int`) of memory using `malloc()`, and stores the location of that memory in p (cast to `int *` to indicate the type of the data being pointed to).

Line 2 looks up the *address of* `x` and stores it in q.

Source: CS Illustrated

Pointers illustrated




```
*p = 1;
*q = 2;
*a = 3;
```

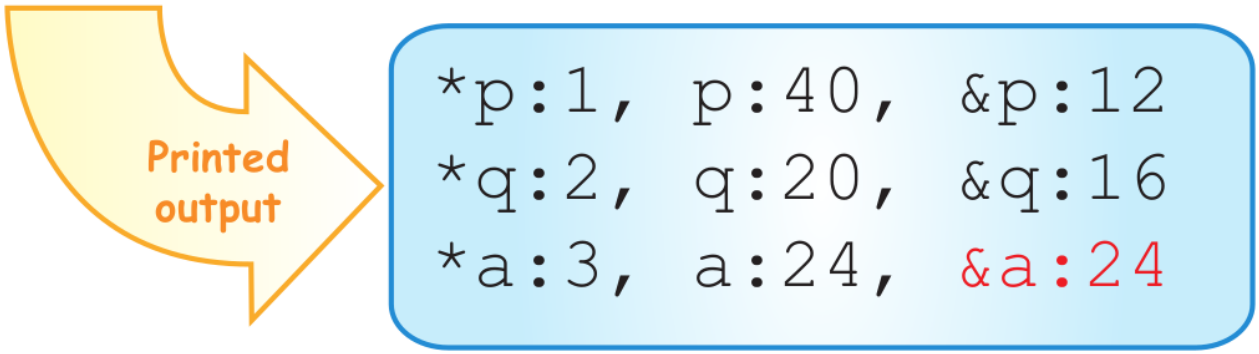
We can access the data *referenced* by a pointer by **dereferencing** it using the $*$. Dereferencing looks inside the memory (box) at the location (address) stored by the pointer.

Here we put values 1, 2, and 3 into boxes pointed to by p, q, and a.

Source: CS Illustrated

Pointers illustrated

```
printf("*p:%u, p:%u, &p:%u\n", *p, p, &p);
printf("*q:%u, q:%u, &q:%u\n", *q, q, &q);
printf("*a:%u, a:%u, &a:%u\n", *a, a, &a);
```



Note: this illustration assumes 32-bit (4-byte) pointers

Source: CS Illustrated

C pointer syntax

Operator	Function
*	pointer / dereference
&	"address of"

```
int x = 1, y = 2, z[10];
int *ip; /* ip is a pointer to int */

ip = &x; /* ip now points to x */
y = *ip; /* y is now 1 */
*ip = 0; /* x is now 0 */
ip = &z[0]; /* ip now points to z[0] */
```

Source: K&R Chapter 5

C pointer syntax

C uses "call-by-value" semantics for function calls

```
void swap(int x, int y)
{
    int temp;

    temp = x;
    x = y;
    y = temp;
}
```

```
int a=123,b=456;

swap(a, b);
```

This function won't swap `a` and `b`, only *copies* of the values.

Source: K&R Section 5.2

Addressing example

```
void swap (long *xp,
           long *yp)
{
  long t0 = *xp;
  long t1 = *yp;
  *xp = t1;
  *yp = t0;
}
```

Called with:

```
long a=123,b=456;

swap(&a, &b);
```

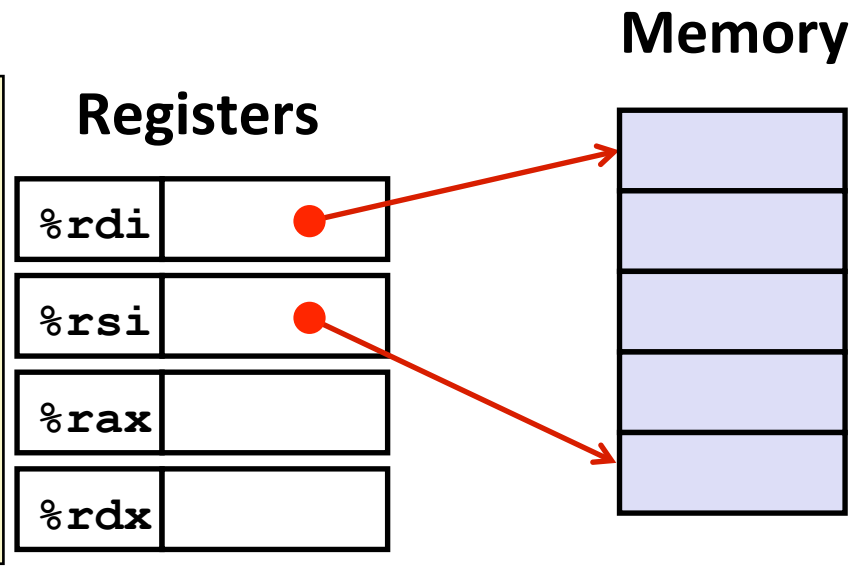
```
swap:
  mov   rax, QWORD PTR [rdi]
  mov   rdx, QWORD PTR [rsi]
  mov   QWORD PTR [rdi], rdx
  mov   QWORD PTR [rsi], rax
```

(or in the other "flavor" asm)

```
swap:
  movq    (%rdi), %rax
  movq    (%rsi), %rdx
  movq    %rdx, (%rdi)
  movq    %rax, (%rsi)
  ret
```

Understanding swap()

```
void swap
    (long *xp, long *yp)
{
  long t0 = *xp;
  long t1 = *yp;
  *xp = t1;
  *yp = t0;
}
```



Register	Value
rdi	xp
rsi	yp
rax	t0
rdx	t1

```
mov   rax, QWORD PTR [rdi]
mov   rdx, QWORD PTR [rsi]
mov   QWORD PTR [rdi], rdx
mov   QWORD PTR [rsi], rax
```

Understanding swap()

Registers

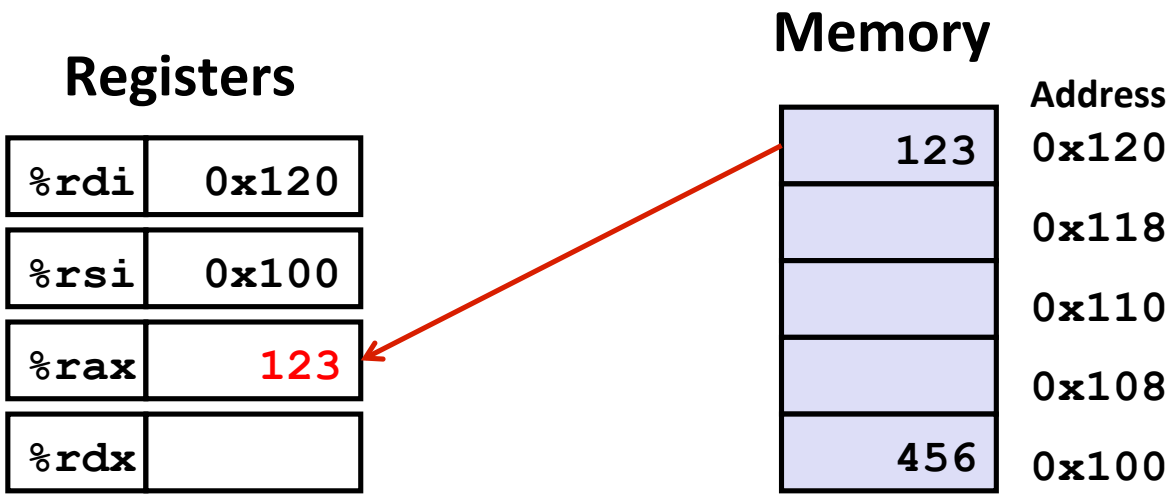
%rdi	0x120
%rsi	0x100
%rax	
%rdx	

Memory

	Address
123	0x120
	0x118
	0x110
	0x108
456	0x100

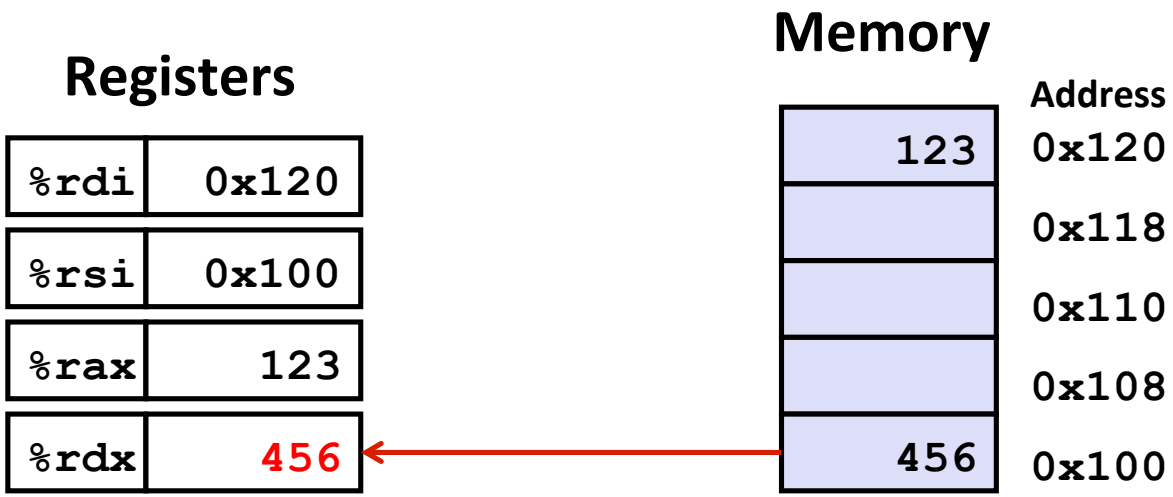
```
swap:
   mov   rax, QWORD PTR [rdi]
   mov   rdx, QWORD PTR [rsi]
   mov   QWORD PTR [rdi], rdx
   mov   QWORD PTR [rsi], rax
```

Understanding swap()



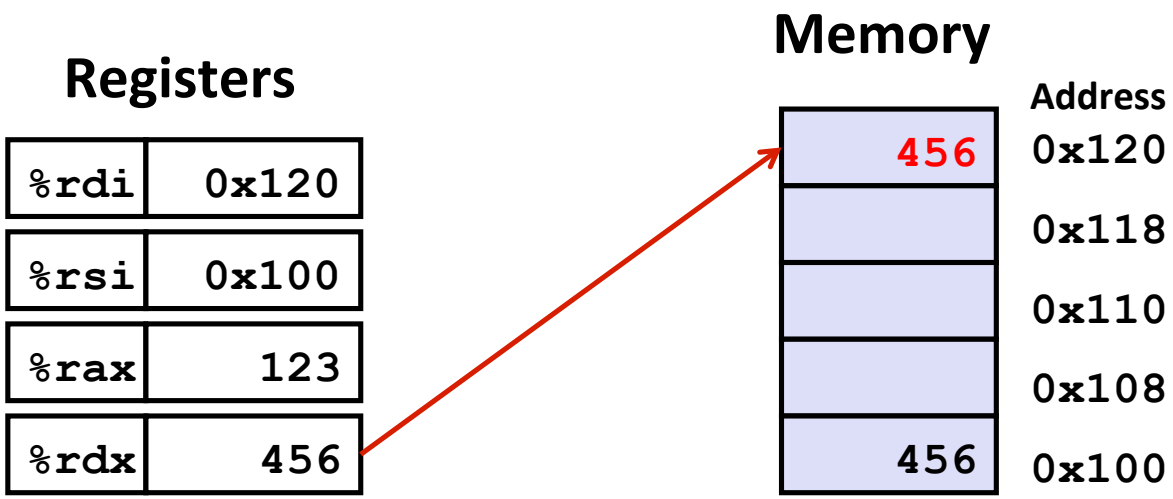

```
swap:
   mov   rax, QWORD PTR [rdi]
   mov   rdx, QWORD PTR [rsi]
   mov   QWORD PTR [rdi], rdx
   mov   QWORD PTR [rsi], rax
```

Understanding swap()



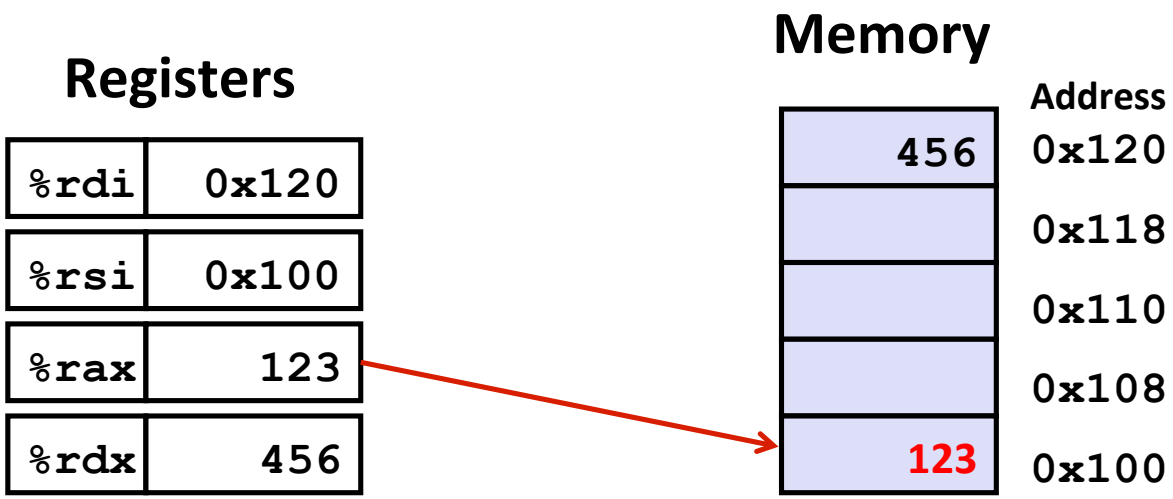

```
swap:
   mov   rax, QWORD PTR [rdi]
   mov   rdx, QWORD PTR [rsi]
   mov   QWORD PTR [rdi], rdx
   mov   QWORD PTR [rsi], rax
```

Understanding swap()




```
swap:
   mov   rax, QWORD PTR [rdi]
   mov   rdx, QWORD PTR [rsi]
   mov   QWORD PTR [rdi], rdx
   mov   QWORD PTR [rsi], rax
```

Understanding swap()




```
swap:
   mov   rax, QWORD PTR [rdi]
   mov   rdx, QWORD PTR [rsi]
   mov   QWORD PTR [rdi], rdx
   mov   QWORD PTR [rsi], rax
```