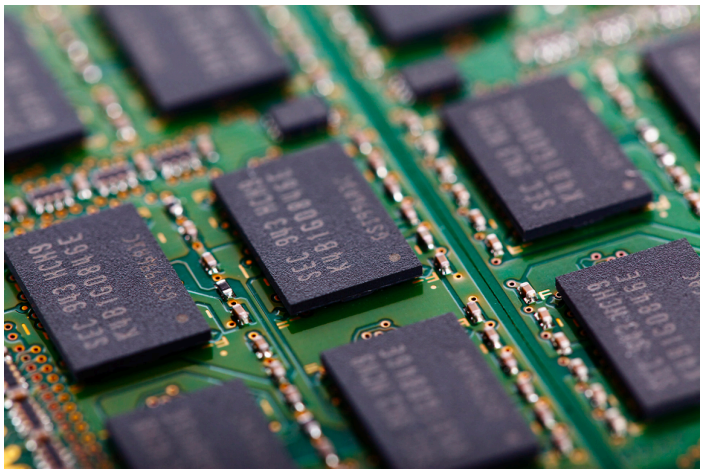
Memory



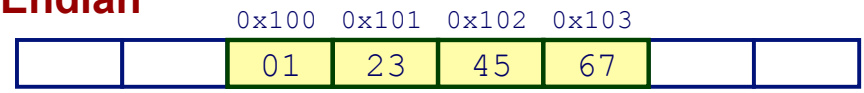
Memory Addresses

Content:	FF	00	57	92	B3	8A	... ...	10	46	DC
Address:	000 000 000	000 000 001	000 000 002	000 000 003	000 000 004	000 000 005	... ...	134 217 725	134 217 726	134 217 727

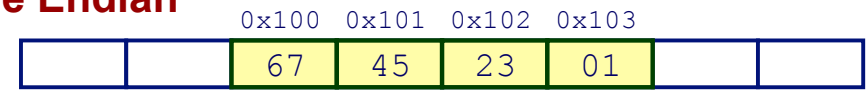
Byte Ordering (Endianness)

```
int x = 0x01234567; // 32-bit num at address 0x100
```

Big Endian



Little Endian



Sample Memory

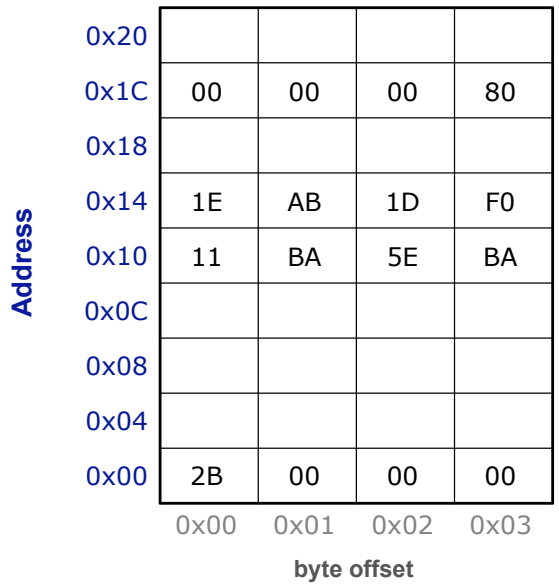
36 bytes (addresses 0x00 - 0x23)

Byte	2B	00	00	00								
	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0A	0x0B
	Address											

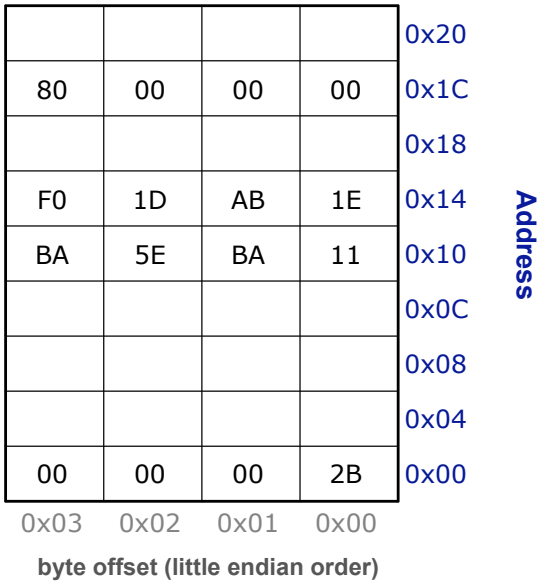
					11	BA	5E	BA	1E	AB	1D	F0
	0x0C	0x0D	0x0E	0x0F	0x10	0x11	0x12	0x13	0x14	0x15	0x16	0x17

					00	00	00	80				
	0x18	0x19	0x1A	0x1B	0x1C	0x1D	0x1E	0x1F	0x20	0x21	0x22	0x23

Sample Memory

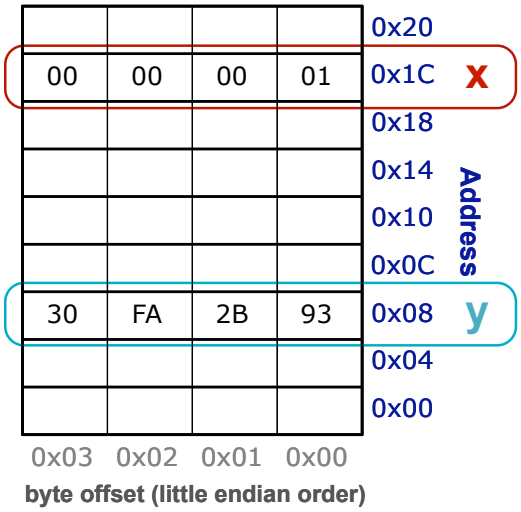


Sample Memory



Variables as Addresses

```
int x; // x at 0x1C
int y; // y at 0x08

x = 1;
y = 0x30FA2B93;
```



Java References

```
class Blob {
  // ... instance variables ...
}


public void doStuff() {

  Blob b1 = new Blob();
  Blob b2 = new Blob();

}
```

Java References

```
class Blob {
  // ... instance variables ...
}


public void doStuff() {

  Blob b1 = new Blob();
  Blob b2 = b1;

}
```


Pointers

address = index of a byte in memory

pointer = piece of data storing an address

`T* p;`	declare a pointer `p` that will point to something of type `T`
`&x`	**address of** `x` (get a pointer to `x`)
`*p`	**contents at** address given by pointer `p` (aka "**dereference** `p`" – follow the pointer)

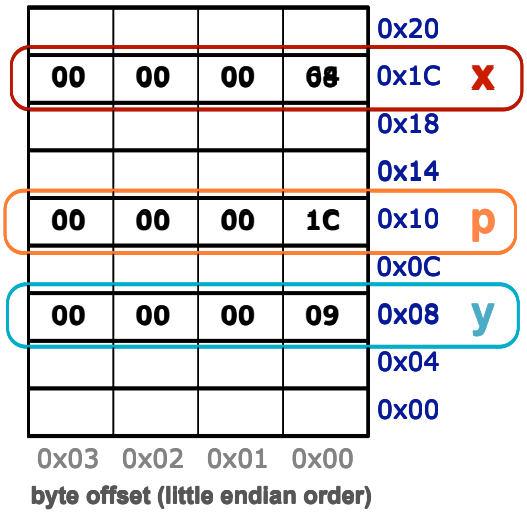
Pointer Example

```
int* p; // p: 0x10

int x = 8; // x: 0x1C
int y = 3; // y: 0x08
```

```
p = &x;
```

```
y = 1 + *p;
```

```
*p = 100;
```

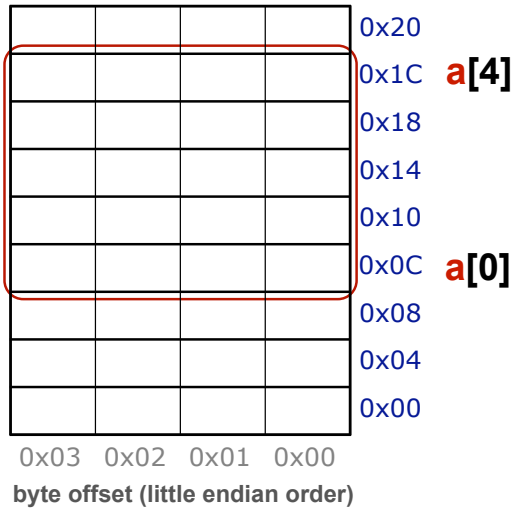


Pointer Example

```
void swap(int* p1, int* p2) {
    int temp = *p1;
    *p1 = *p2;
    *p2 = temp;
}

void main() {
    int x = 5;
    int y = 3;

    // x is 5, y is 3

    swap(&x, &y);

    // x is 3, y is 5
}
```

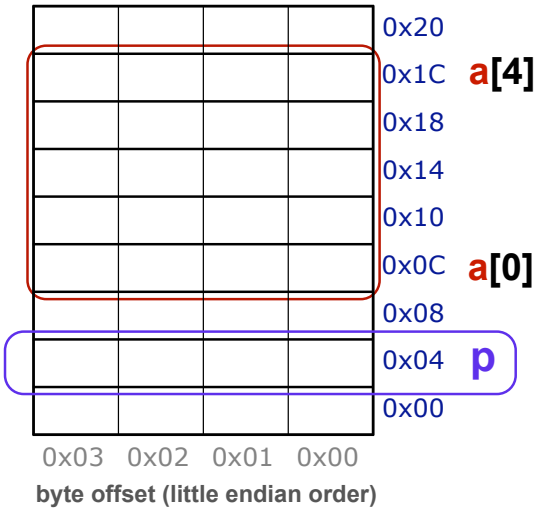
Arrays in C

```
int a[5]; // creation

a[0] = 0xFF; // indexing
a[3] = a[0];

~~a.length~~; // nope!

a[5] = 0xBAD; // uh oh
a[-1] = 0xBAD; // x2
```



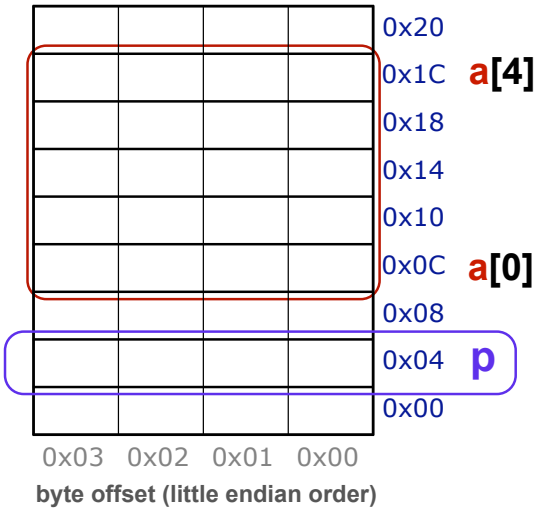
Arrays as Pointers

```
int a[5];

int* p;
```



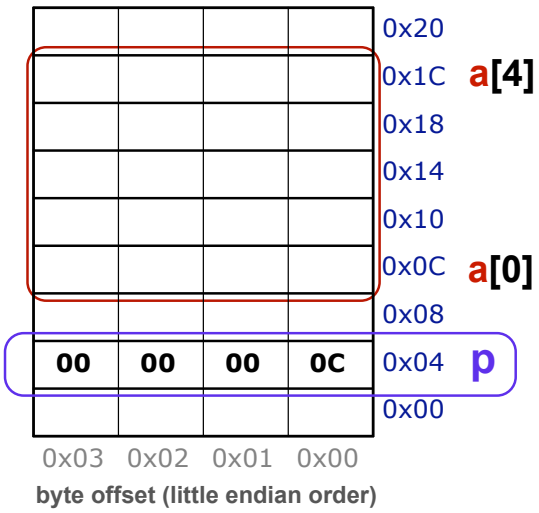
Arrays as Pointers

```
int a[5];

int* p;

p = &a[0];
```



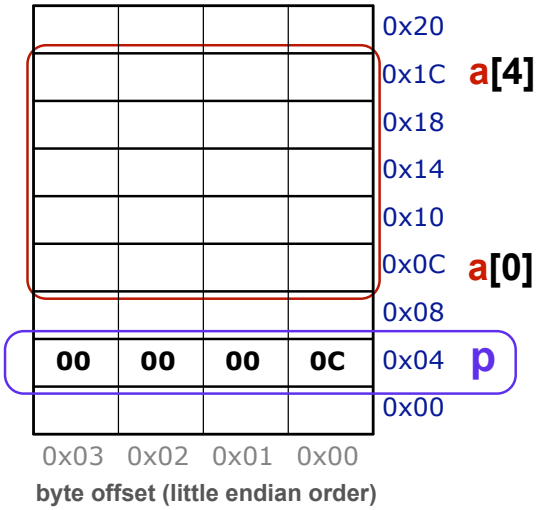
Arrays as Pointers

```
int a[5];

int* p;

p = &a[0];
```



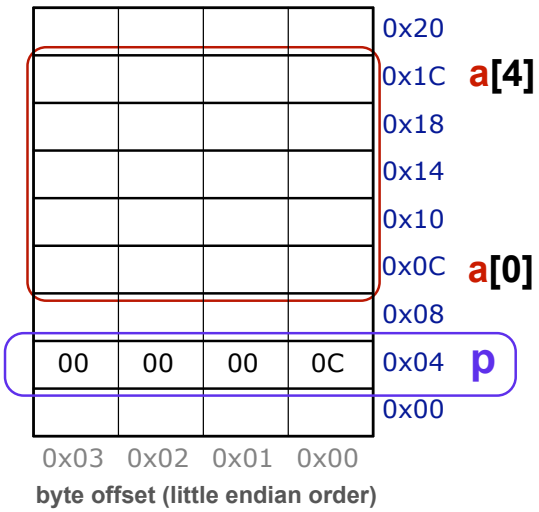
Arrays as Pointers

```
int a[5];

int* p;

p = &a[0]; // or p = a;
```



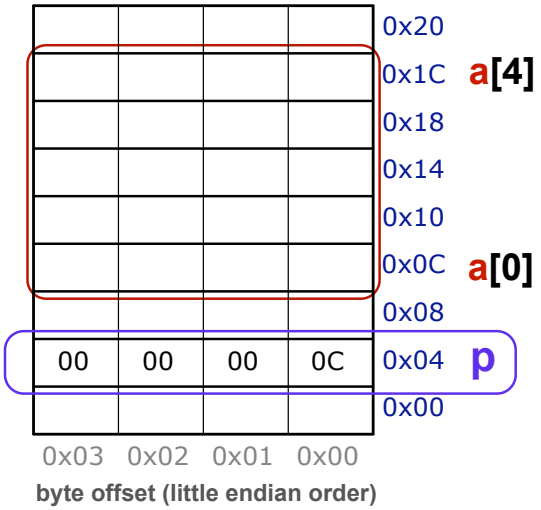
Arrays as Pointers

```
int a[5];

int* p;

p = a; // &a[0]
```



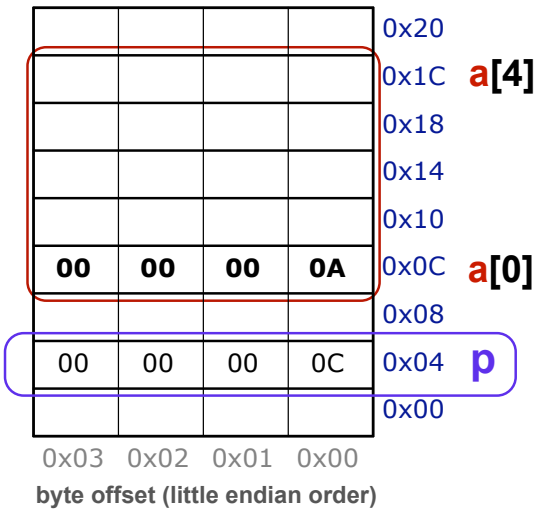
Arrays as Pointers

```
int a[5];

int* p;

p = a; // &a[0]

*p = 0xA;
```



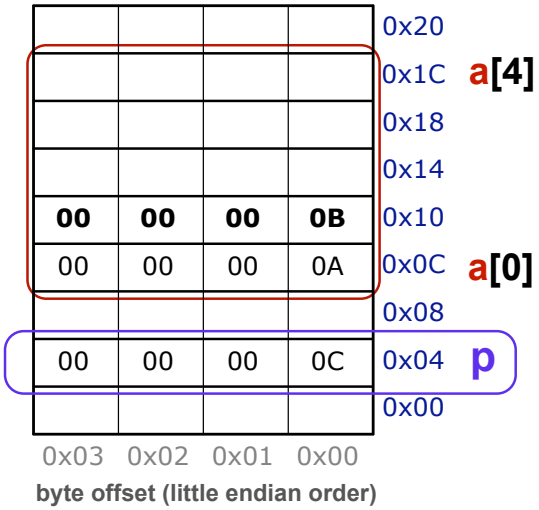
Arrays as Pointers

```
int a[5];

int* p;

p = a; // &a[0]

*p = 0xA;
```



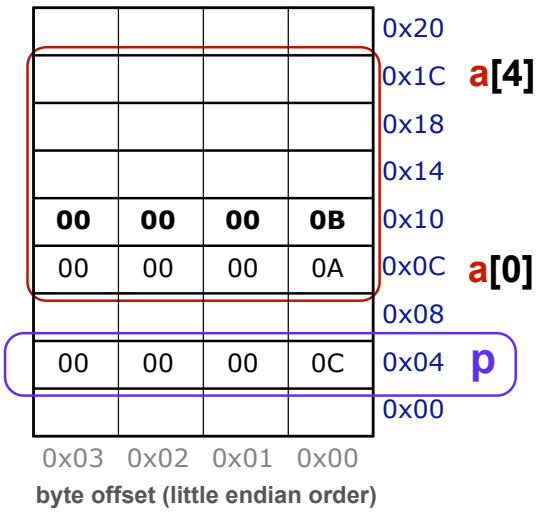
Arrays as Pointers

```
int a[5];

int* p;

p = a; // &a[0]

*p = 0xA;
a[1] = 0xB; // using a
```



Arrays as Pointers

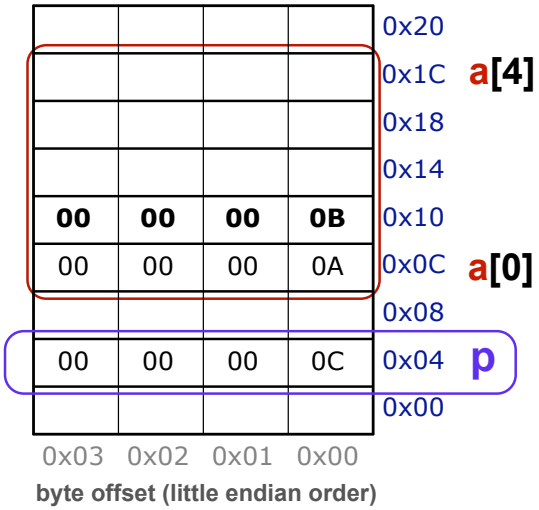
```
int a[5];

int* p;

p = a; // &a[0]

*p = 0xA;
p[1] = 0xB; // using p
```



Arrays as Pointers

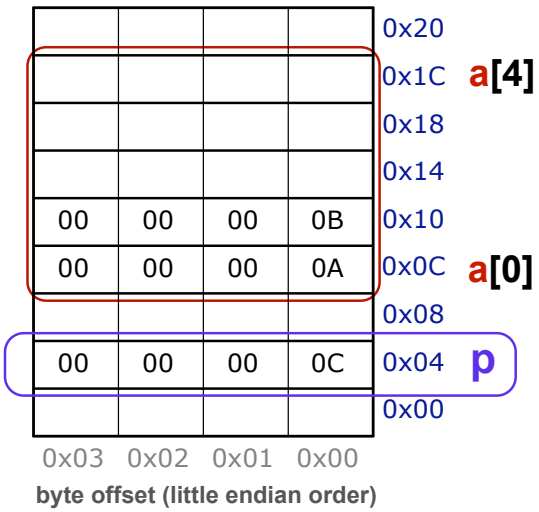
```
int a[5];

int* p;

p = a; // &a[0]

*p = 0xA;
*(p + 1) = 0xB;
```

Pointer Arithmetic!



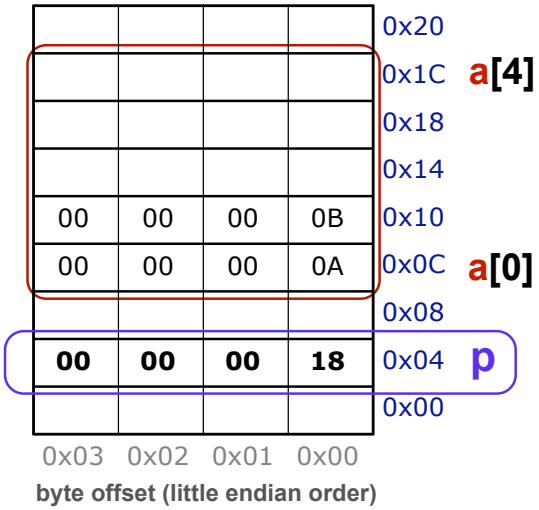
Arrays as Pointers

```
int a[5];

int* p;

p = a; // &a[0]

*p = 0xA;
*(p + 1) = 0xB;

p = p + 3;
```



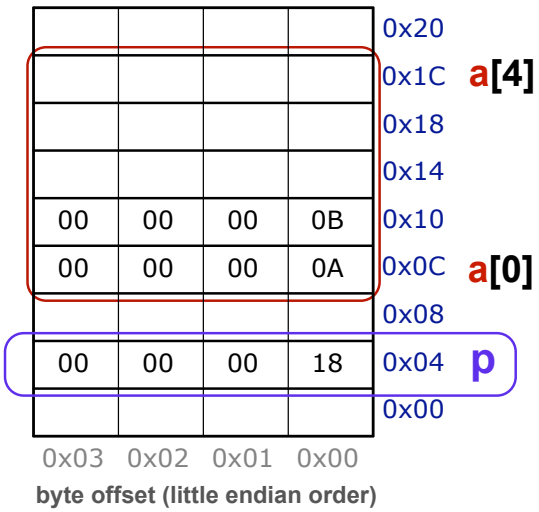
Arrays as Pointers

```
int a[5];

int* p;

p = a; // &a[0]

*p = 0xA;
*(p + 1) = 0xB;

p = p + 3;
```



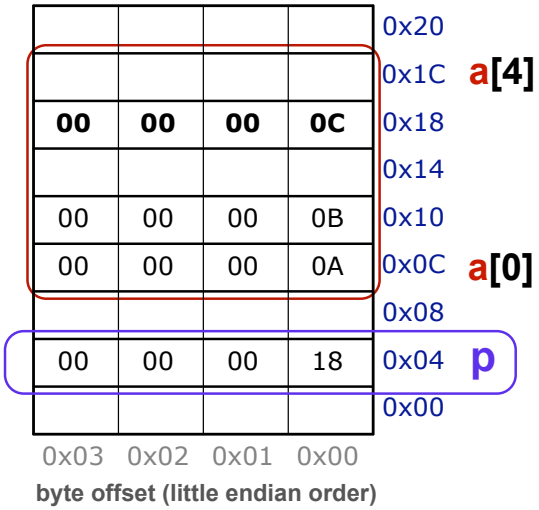
Arrays as Pointers

```
int a[5];

int* p;

p = a; // &a[0]

*p = 0xA;
*(p + 1) = 0xB;

p = p + 3;
*p = a[1] + 1;
```

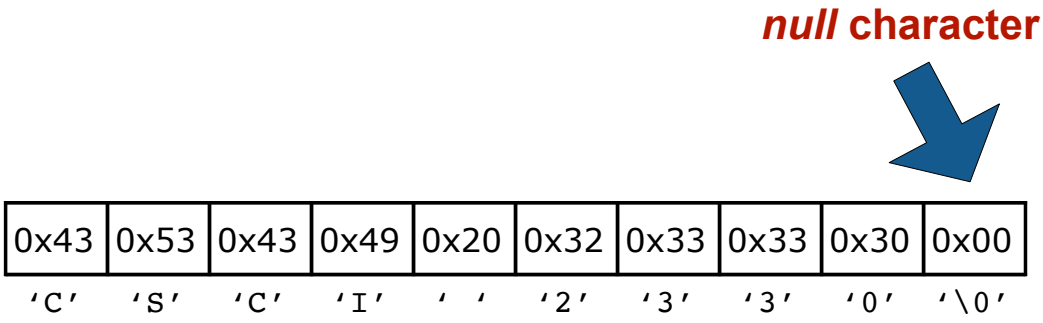


Arrays as Pointers

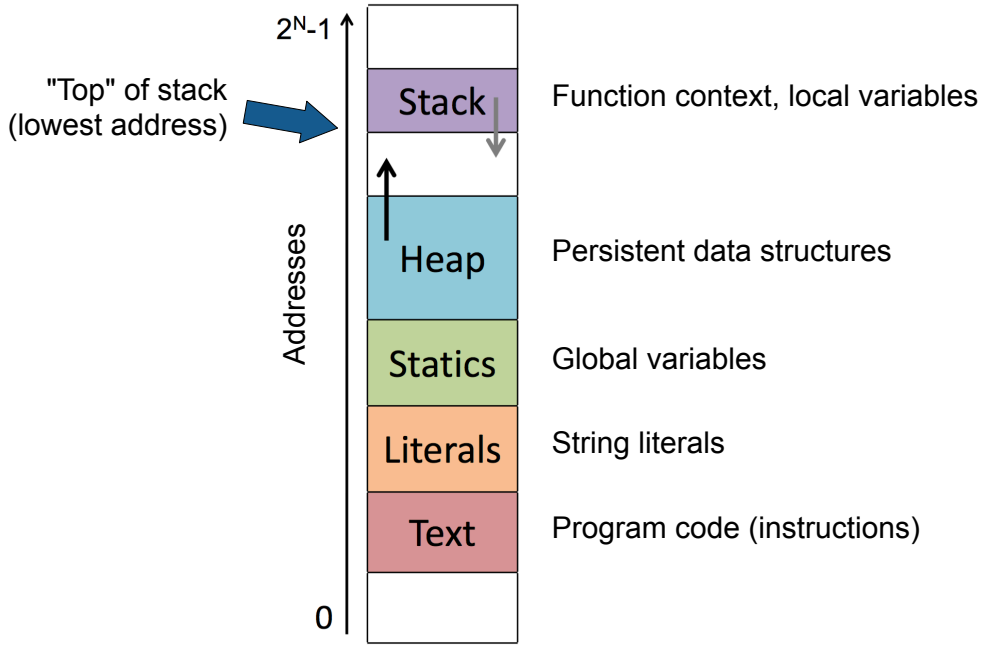
```
int a[5];

int* p;

p = a; // &a[0]

*p = 0xA;
*(p + 1) = 0xB;

p = p + 3;
*p = a[1] + 1;
```



Null-Terminated Strings

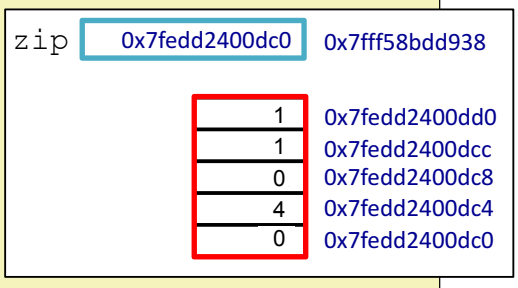


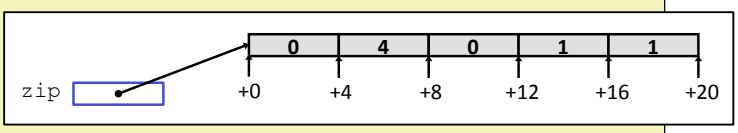
Memory Layout (Segments)



Dynamic Memory Allocation

```
#define ZIP_LENGTH 5

int* zip = malloc(sizeof(int) * ZIP_LENGTH);
if (zip == NULL) {
    perror("malloc failed");
    exit(0);
}

zip[0] = 0;
zip[1] = 4;
zip[2] = 0;
zip[3] = 1;
zip[4] = 1;

printf("zip is");
for (int i = 0; i < ZIP_LENGTH; i++) {
    printf(" %d", zip[i]);
}
printf("\n");

free(zip);
```

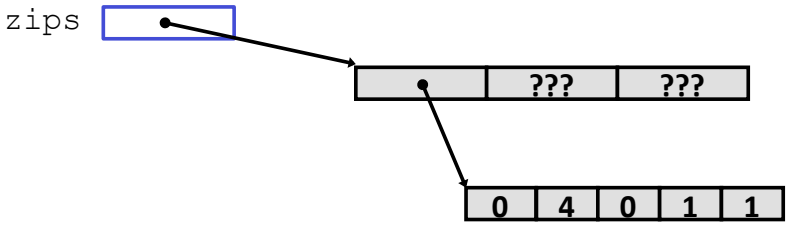




Pointers to Pointers to ...

```
#define NUM_ZIPS 3

int** zips = malloc(sizeof(int*) * NUM_ZIPS);
...
int* zip = malloc(sizeof(int) * ZIP_LENGTH);
zips[0] = zip;
...
int* first_zip = zips[0];
first_zip[0] = 0;
zips[0][1] = 4;
zips[0][2] = 0;
first_zip[3] = 1;
zips[0][4] = 1;
```

allocating *pointers*!



Memory Errors

(credit: xkcd.com)