

CSCI 2410 Systems Programming

Fall 2026

Instructor: Jie Zhou

**Department of Computer Science
George Washington University**



Slides materials are mostly credited to Gabriel Parmer and Sibin Mohan at GWU.

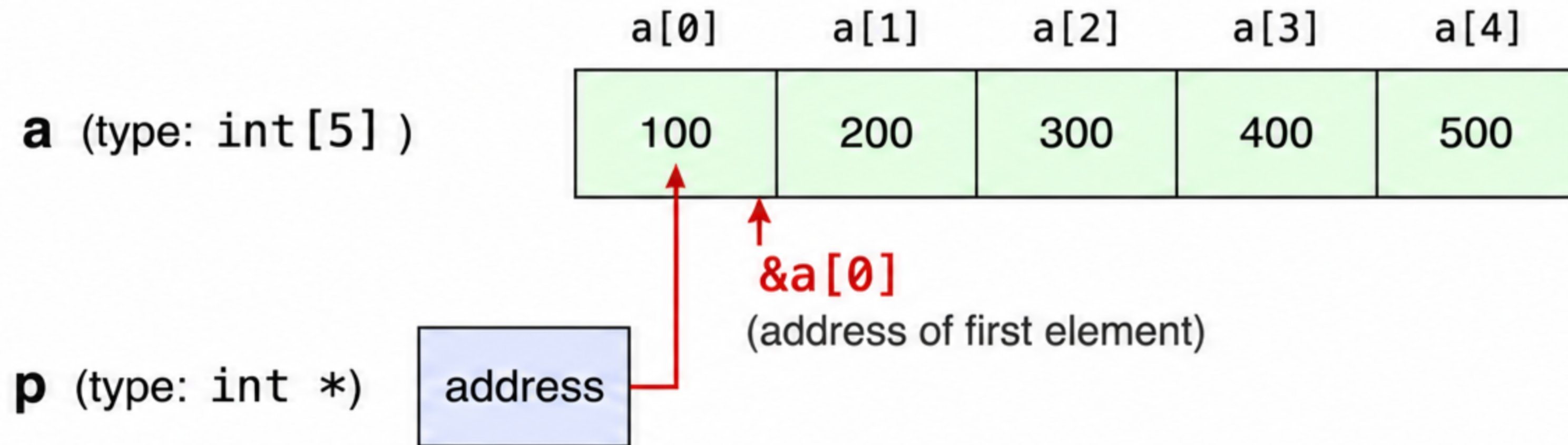
Pass by Value vs. Pass by Reference

- Functions may need to access (read/write) the same memory object.
- How to “pass” shared memory between function calls?
 - Pass by value: copying the entire shared memory object
 - Pass by reference: passing a pointer to the object

Array vs. Pointer

- An array is *not* a pointer.
- But an array can decay into a pointer to its first element.

```
int a[5] = {100, 200, 300, 400, 500};  
int *p = a;
```



Array vs. Pointer

- The variable name of an array is *not* a pointers.
- But an array's variable can decay into a pointer.

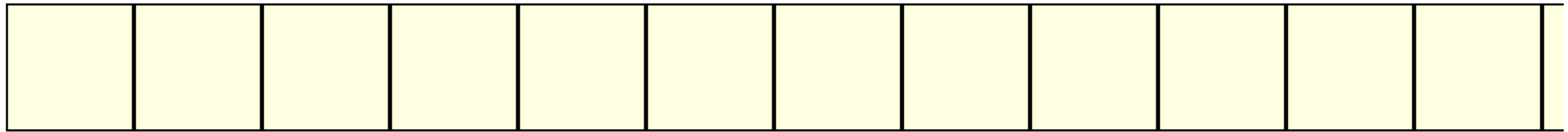
```
int a[5] = {100, 200, 300, 400, 500};  
int *p = a;
```

- When an array item is accessed, e.g., `a[2]`, C is basically doing pointer arithmetic. So, `a[2] → *(a+2)`

<code>a</code>	Start address of the array
<code>a + 2</code>	Address of the third item in the address
<code>*(a + 2)</code>	dereference address (<code>a+2</code>), i.e., accessing the third item in the array

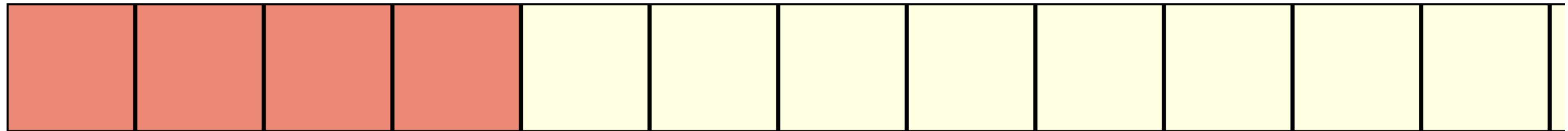
Pointer Arithmetic

- Consider an integer array, `int a[5];`
- 5 ints, each of 4 bytes



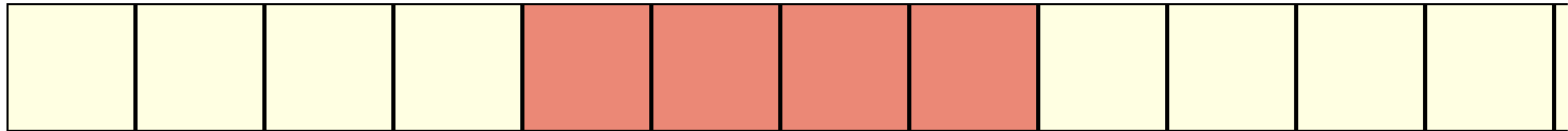
Pointer Arithmetic

- Consider an integer array, `int a[5];`
- 5 ints, each of 4 bytes
- `a[0]` is:



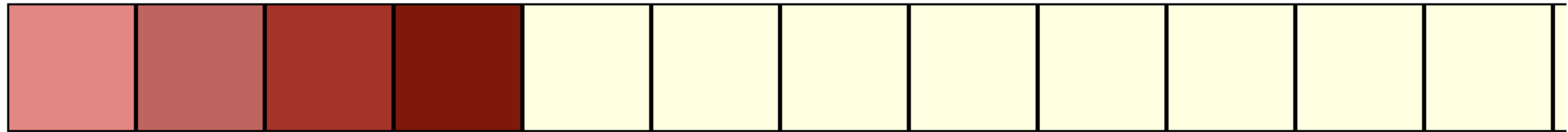
Pointer Arithmetic

- Consider an integer array, `int a[5];`
- 5 ints, each of 4 bytes
- `a[1]` (or `*(a + 1)`) is:



Pointer Arithmetic

- Consider an integer array, `int a[5];`
- 5 ints, each of 4 bytes
- We cannot access the individual bytes:



- What if we want to access the individual bytes?
- We can do pointer casting.

malloc

```
void *malloc(size_t size);
```

- Allocate raw memory of `size` bytes
- No initialization
 - “data” is whatever is in memory.
- It is a good practice to use `memset()` to set memory to 0.
- Returns a pointer to newly allocated memory

```
char* str = malloc(128 * sizeof(char));
```

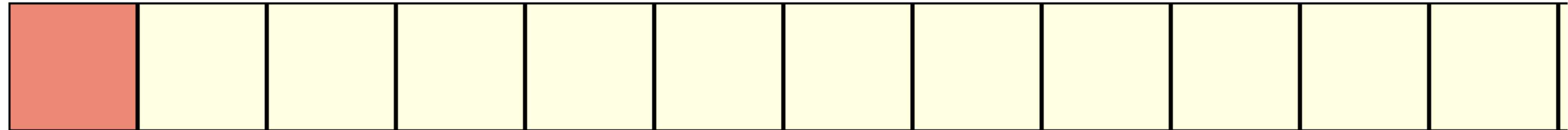
- Allocate 128 bytes of memory
- `str` points to the beginning of this chunk of memory.

Pointer Cast

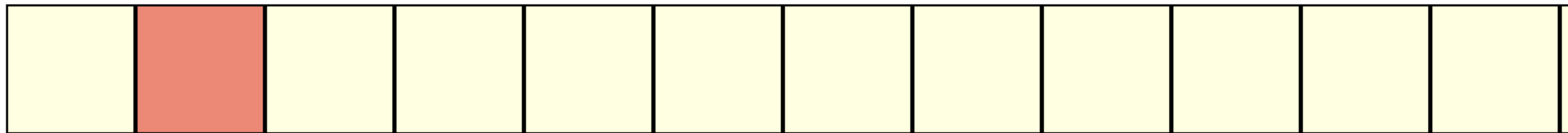
- Can cast from one pointer type to another
- Between any two pointers
- A pointer itself is always the same size, e.g., 8 bytes on a 64-bit system
- Does not change the memory underneath
 - but, changes pointer arithmetic.

Pointer Cast

- Consider an integer array, `int a[5];`
- now we do `char *pc = (char *) a;`
- `pc` points to the same memory region as `a`.
- But now, we can treat the memory object as characters.
- `pc[0]` is:



- `pc[1]` is:



Example

```
#include <assert.h>
#include <stdio.h>

int main()
{
    int* a = (int*)malloc( sizeof(int) ) ;
    assert(a != NULL) ; // SAME as assert (a)

    *a = 1145258561 ;

    printf( "a = %d\n", *a ) ;

    char* ppc = (char*) a ;
    for(unsigned int i = 0 ; i < sizeof(*a); ++i )
        printf( "%c ", *ppc++ ) ;

    printf("\n") ;
    return 0 ;
}
```

Example

```
struct Animal {
    int age;
};

struct Dog {
    struct Animal animal;    // "base" object first
    int bark_volume;
};

void f(void) {
    struct Animal *a = malloc(sizeof(struct Dog));

    struct Dog *d = (struct Dog *)a;

    a->age = 5;
    d->bark_volume = 10;

    free(a);
}
```

Cast between Regular Pointers and void *

- Can cast any pointer to void *
- Can also cast void * to any pointer
- Cannot dereference a void *

```
void *pv = malloc(32);  
int *pi = (int*) pv; // the (int *) before pv is optional  
double *pd = (double*) pv; // (double *) before pv is also optional.  
char *pc = pi; // (char *) before pi is needed; there would be a warning otherwise.
```