Review

- Pointer
  - Motivation
  - Declaration
  - Swap

Arrays – Basics

Lecture 30-1

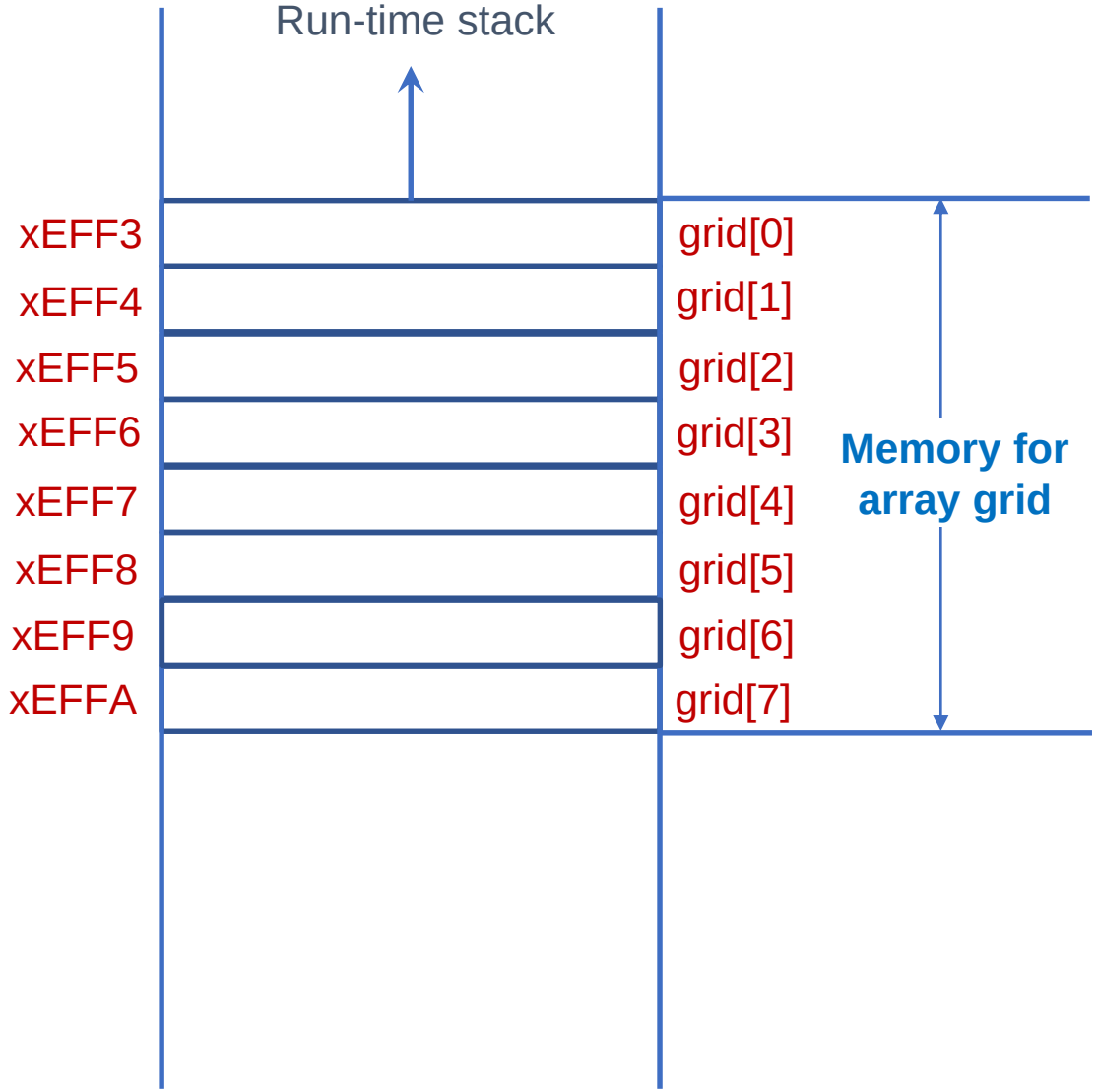
Hyung-Sin Kim

SNU Graduate School of Data Science

Array

- An array is a collection of **similar data items** that are stored **sequentially in memory** and accessible through a **single name** or identifier

- In contrast to lists in Python, an array in C
  - Can store only a **single** data type
  - Has a **fixed size**
- Declaration - int grid[8];
  - **grid** is an array of eight integer variables
  - The first element (grid[0]) is allocated in the lowest memory address
  - The last element (grid[7]) is allocated in the highest memory address



Array – Example

```
#include <stdio.h>
#define NUM_STUDENTS 5

int main(void) {
    int midterm[NUM_STUDENTS];
    int final[NUM_STUDENTS];
    int total[NUM_STUDENTS];

   // Input exam scores
   for (int i=0; i < NUM_STUDENTS; i++) {
        printf("Input midterm score for student %d: ", i);
        scanf("%d", &midterm[i]);
        printf("Input final score for student %d: ", i);
        scanf("%d", &final[i]);
    }

    // Calculate total scores
    for (int i=0; i < NUM_STUDENTS; i++) {
        total[i] = midterm[i] + final[i];
    }

    // Output the total scores
    for (int i=0; i < NUM_STUDENTS; i++) {
        printf("Total score for Student %d is %d\n", i, total[i]);
    }

    return 0;
}
```

Array – String (Array of Characters)

- Strings in C are simply arrays of character type
  - char word[10];
  - To store “winter” (6 characters) in word, we need to mark where the string ends
    - word[0] = ‘w’; word[1] = ‘i’; word[2] = ‘n’; word[3] = ‘t’; word[4] = ‘e’; word[5] = ‘r’; word[7] = **‘\0’**;
  - **\0** is the special character sequence that indicates the null character whose ASCII value is 0
    - Serves as a **sentinel** that identifies the end of a string
    - We must reserve one element for the null character, and
    - Thus, **word** can store a string comprising up to 9 characters
  - printf(“%s”, word); // should print **winter, %s** is the format specification for string
- Strings can also be initialized within their declarations
  - char word[10] = “winter”; printf(“%s”, word);
  - Single quotes ‘ ’ for one character, double quotes “ ” for a string
  - The null character \0 is automatically added to the end of the string

Arrays – One Step Further

Lecture 30-2

Hyung-Sin Kim

SNU Graduate School of Data Science

Array – Relationship with Pointer

- Example
  - int values[10]; // Without any index, **values** itself is the same as &values[0]
  - int *valPtr;
  - valPtr = values;
- valPtr and values are very similar as shown below:
  - One difference is that **valPtr** can be reassigned but **values** cannot be reassigned
  - values = newArray[xx]; will cause a compiler error

	Using a Pointer	Using Name of Array	Using Array Notation
Address of array	valPtr	values	&values[0]
0-th element	*valPtr	*values	values[0]
Address of n-th element	(valPtr + n)	(values + n)	&values[n]
n-th element	*(valPtr + n)	*(values + n)	values[n]

Array – Passing by Reference

- Averaging function

```
#include <stdio.h>
#define MAX_NUMS 5
int Print(int inputValues[]);

int main(void) {
    int mean;
    int nums[MAX_NUMS];

    printf("Enter %d nums,\n", MAX_NUMS);
    for (int i =0; index < MAX_NUMS; index++) {
        printf("Input num %d: ", i);
        scanf("%d", &nums[i]);
    }
    mean = Average(nums);
    printf("The average of these nums is %d\n", mean);

    return 0;
}
```

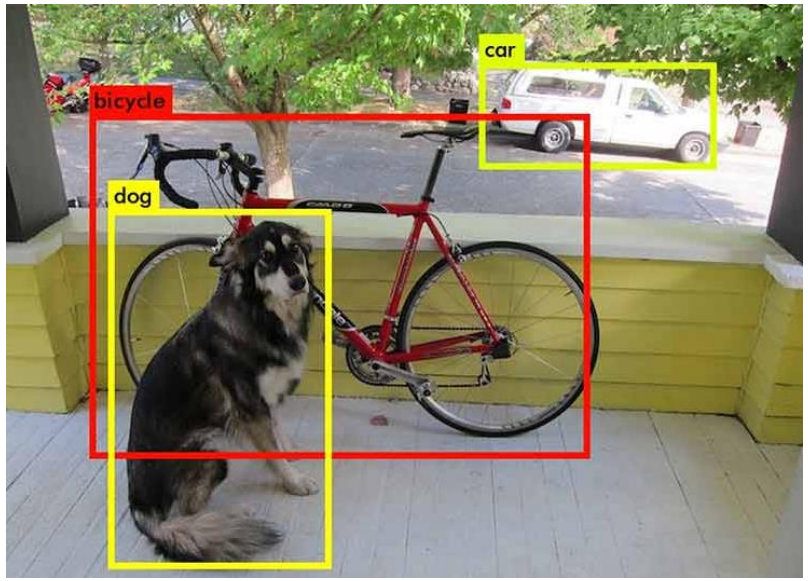
```
int Average(int inputValues[]) {
    int sum = 0;

    for (int i=0; i < MAX_NUMS; i++) {
        sum += inputValues[i];
    }

    return (sum / MAX_NUMS);
}
```

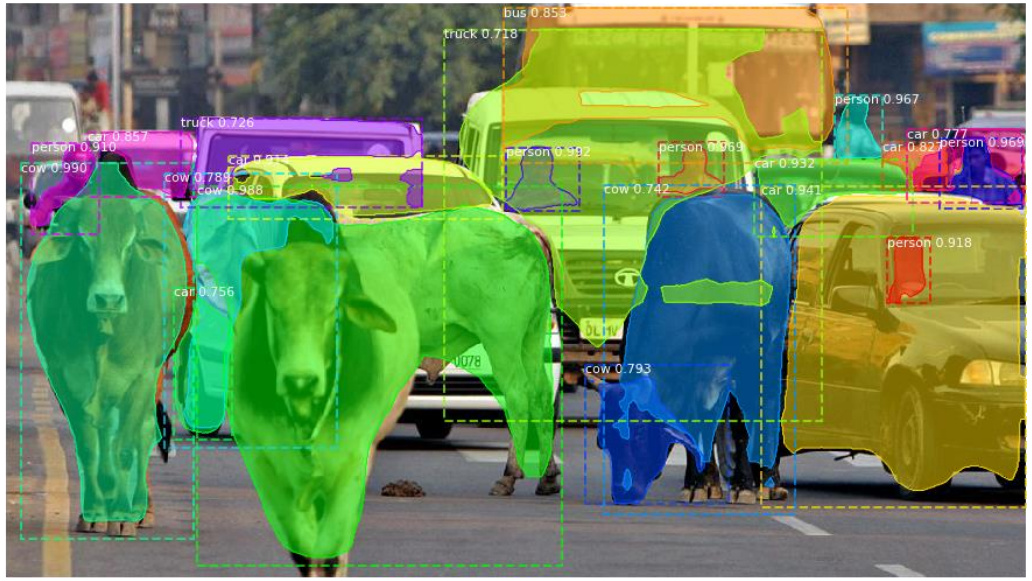
InputValues becomes nums (== &nums[0])

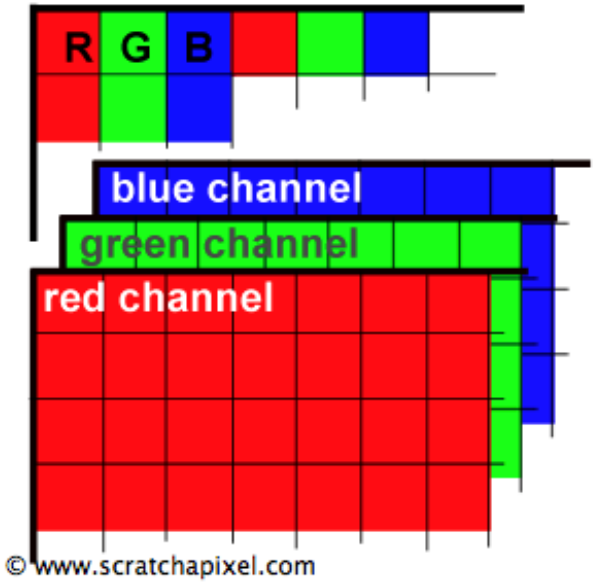
All elements of **nums** can be accessed by using **InputValues**

Array – Multi-dimensional Array

- Computer vision is a very popular field of AI
  - Object detection, object segmentation
  - An image is a multi-dimensional array



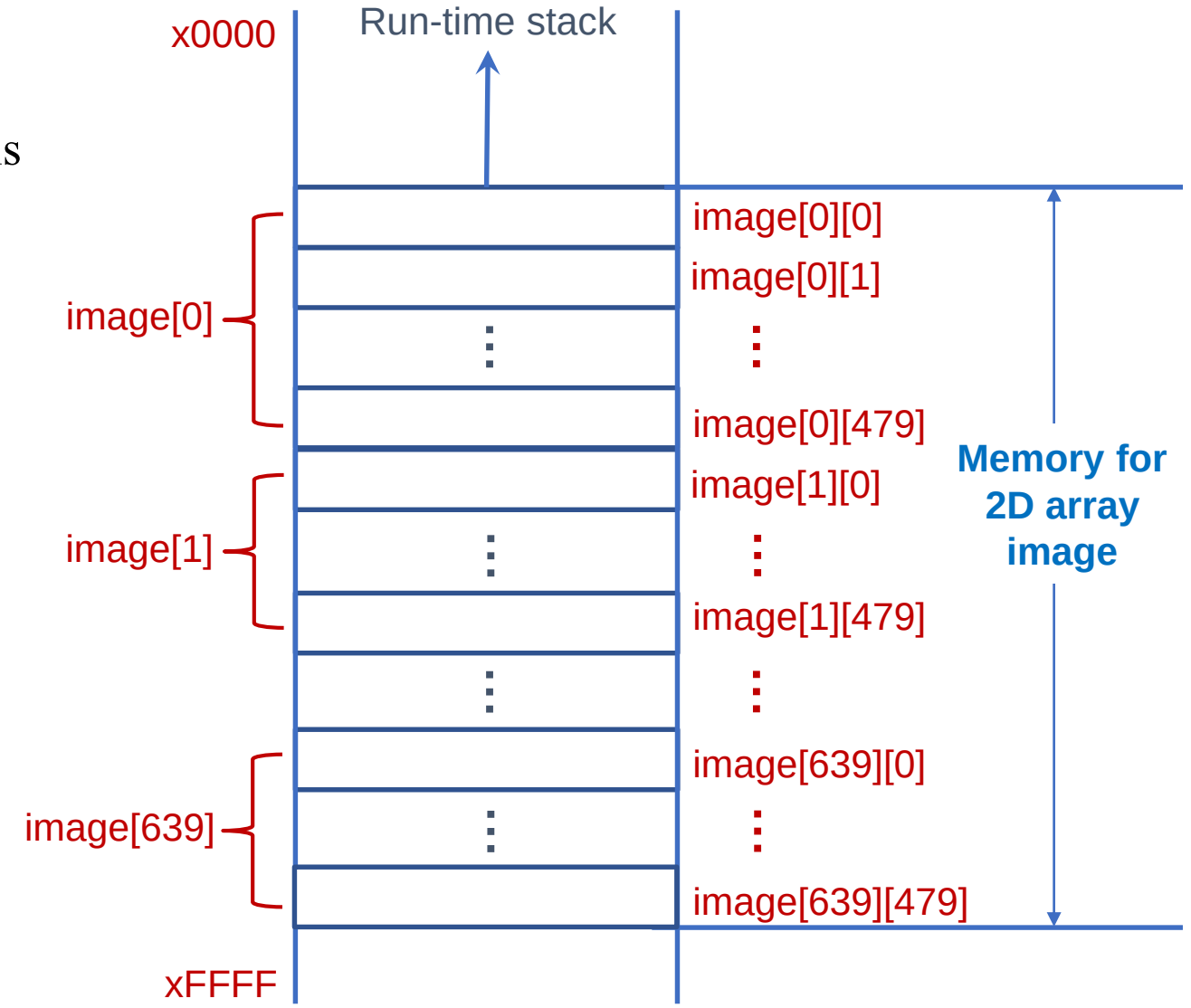




Array – Multi-dimensional Array

- 2D array – int values[ROWS][COLS];
  - Useful for processing an image (e.g., pixels in a 640x480 image)
  - All columns in a row are grouped and allocated in memory like an array

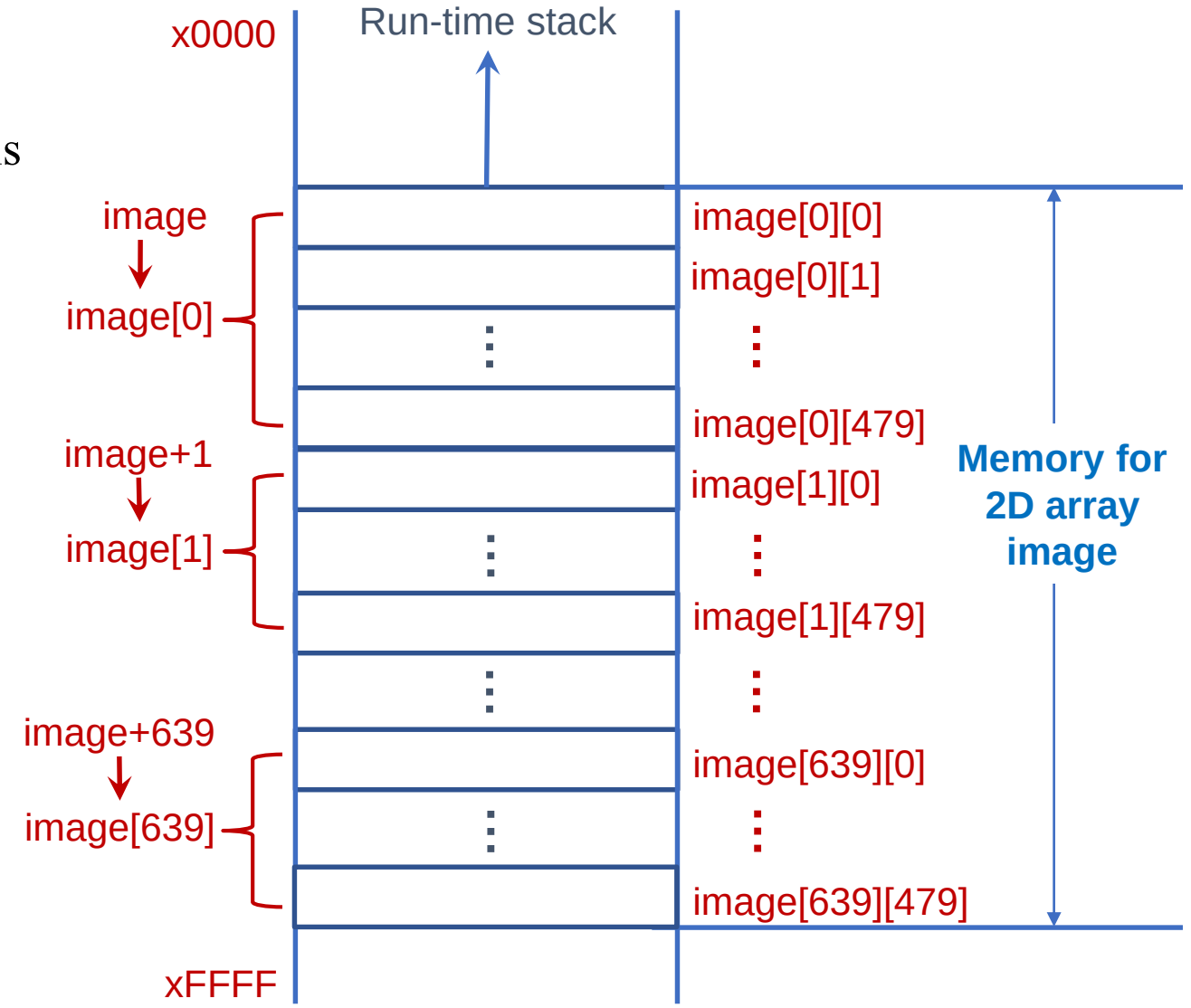
- Example
  - int image[640][480];
  - image[n] is a 1D array of 480 elements



Array – Multi-dimensional Array

- 2D array – int values[ROWS][COLS];
  - Useful for processing an image (e.g., pixels in a 640x480 image)
  - All columns in a row are grouped and allocated in memory like an array

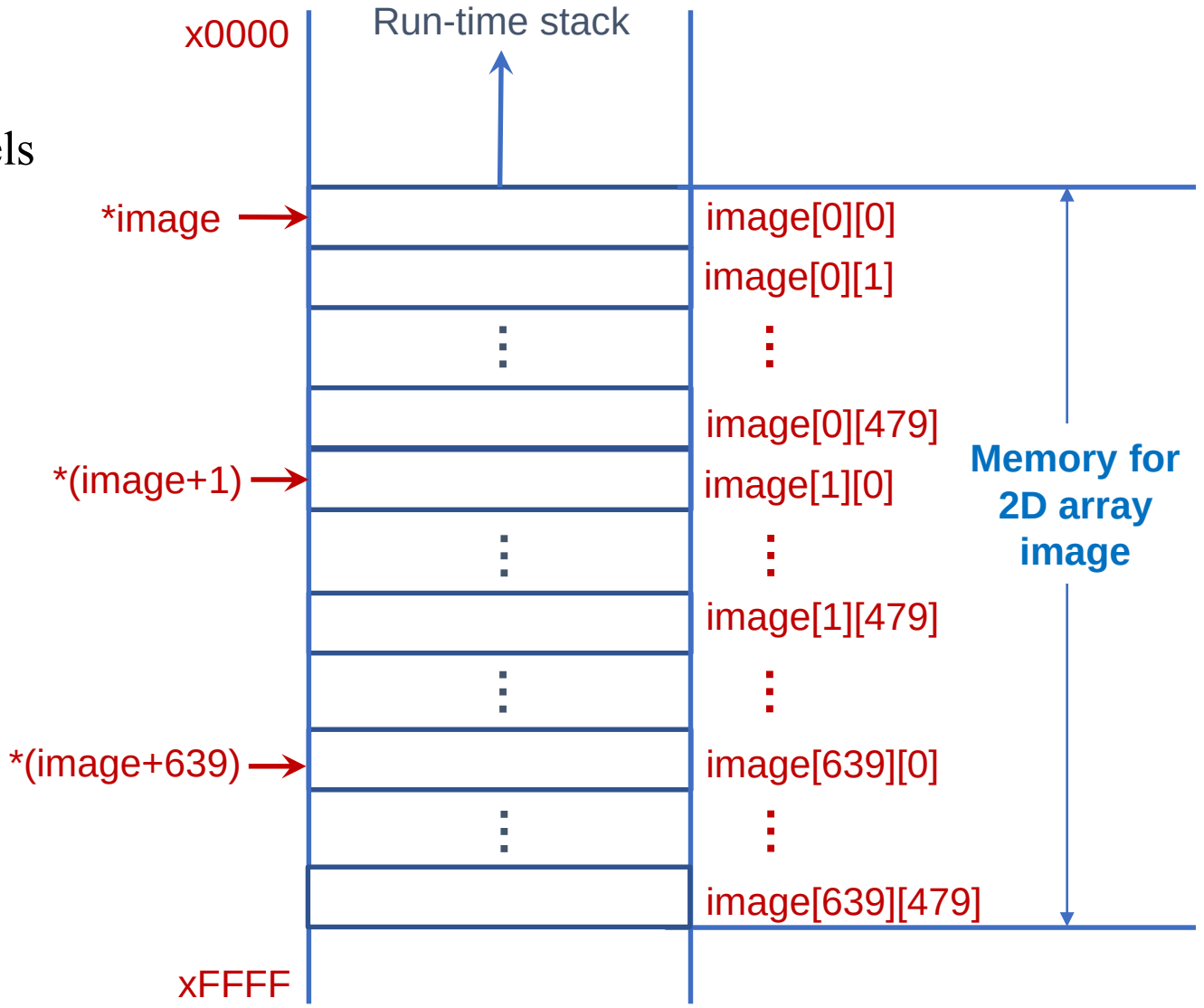
- Example
  - int image[640][480];
  - image[n] is a 1D array of 480 elements
  - (image+n) points to n-th 1D array



Array – Multi-dimensional Array

- 2D array – int values[ROWS][COLS];
  - Useful for processing an image (e.g., pixels in a 640x480 image)
  - All columns in a row are grouped and allocated in memory like an array

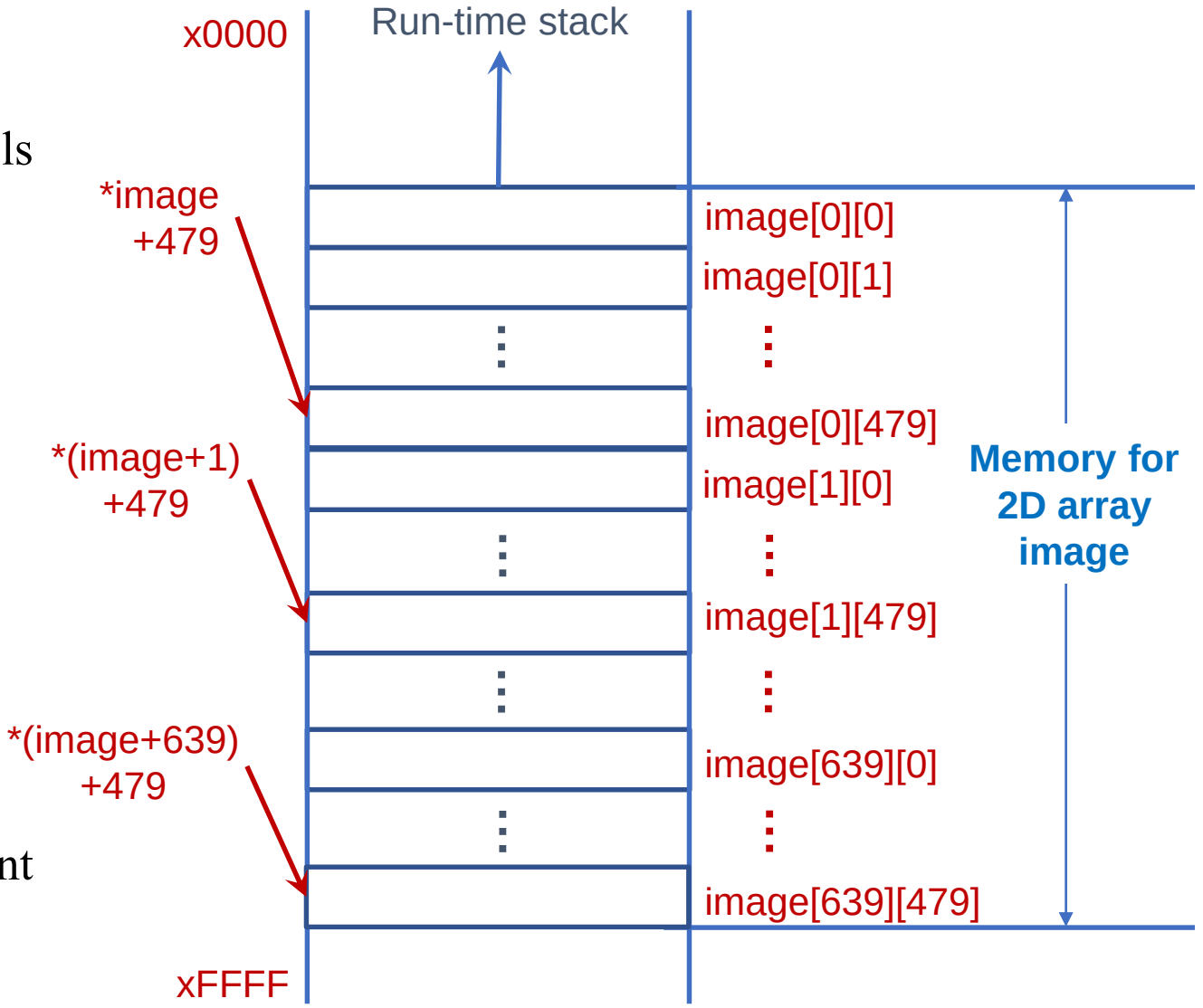
- Example
  - int image[640][480];
  - image[n] is a 1D array of 480 elements
  - (image+n) points to n-th 1D array
  - *(image+n) points to the 0-th element of image[n]: image[n][0]



Array – Multi-dimensional Array

- 2D array – int values[ROWS][COLS];
  - Useful for processing an image (e.g., pixels in a 640x480 image)
  - All columns in a row are grouped and allocated in memory like an array

- Example
  - int image[640][480];
  - image[n] is a 1D array of 480 elements
  - (image+n) points to n-th 1D array
  - *(image+n) points to the 0-th element of image[n]: image[n][0]
  - (*(image+n)+m) points to the m-th element of image[n]



Array – Multi-dimensional Array

- 2D array – int values[ROWS][COLS];
  - Useful for processing an image (e.g., pixels in a 640x480 image)
  - All columns in a row are grouped and allocated in memory like an array
- Example
  - int image[640][480];
  - image[n] is a 1D array of 480 elements
  - (image+n) points to n-th 1D array
  - *(image+n) points to the 0-th element of image[n]: image[n][0]
  - (*(image+n)+m) points to the m-th element of image[n]

	Using Name of Array	Using Array Notation
Address of n-th array	*(image+n)	image[n] or &image[n][0]
n-th array's 0-th element	**(image+n)	image[n][0]
Address of n-th array's m-th element	(*(image+n) + m)	&image[n][m]
n-th array's m-th element	*(*(image+n) + m)	image[n][m]

Array – Variable-length Arrays

- Array size can be a variable
  - int functionA(int len) {
  - int values[len];
  - …
  - }

- The size of values (len) is not known at compile time
  - In this case, C uses a different type of allocation scheme, which is out of scope of this course

- It is sometimes convenient to use variable-length arrays, which sacrifices performance due to the use of a more complex memory allocation scheme

Array – Warning

- C does not provide protection against exceeding the size of an array
  - No compile error from the following codes
    - int values[10];
    - values[13] = 10;
  - Memory objects outside of the array can be corrupted, resulting in unintended behaviors
    - One of the most common errors in C

- We often use a variable as an index for an array, such as values[i]
  - We must make sure if i is between 0 and values' size
  - Precious 6 months evaporated in my glorious grad-school years…

Summary

- Array
  - Relationship with Pointer
  - Passing by Reference
  - Multi-dimensional Array

Thanks!