CMPT 295

Unit - Machine-Level Programming

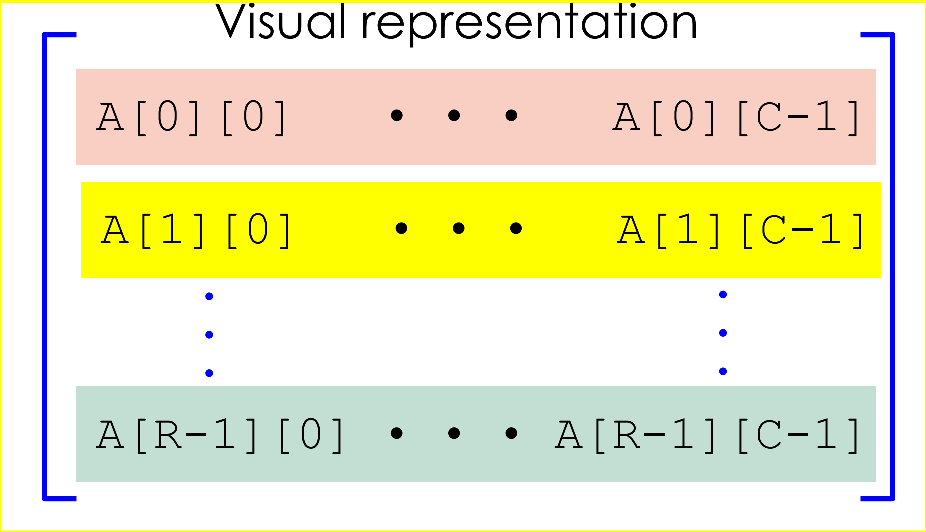
Lecture 21 – Assembly language – Array – 2D

Last lecture

- Recursion
  - Handled without special instruction
    - Stack frames
    - x86-64 Function call and Register saving conventions
- Manipulation of arrays – in x86-64
  - From x86-64's perspective, an array is a contiguously allocated region of **n** * **L** bytes in memory where **L = sizeof(** ***T*** **)** and ***T*** -> data type of elements stored in array
  - Compute memory address of each array element
    - **A[i] = A + i * L**

Today's Menu

- Introduction
  - C program -> assembly code -> machine level code
- Assembly language basics: data, `move` operation
  - Memory addressing modes
- Operation `leaq` and Arithmetic & logical operations
- Conditional Statement – Condition Code + `cmovX`
- Loops
- Function call – Stack
  - Overview of Function Call
  - Memory Layout and Stack - x86-64 instructions and registers
  - Passing control
  - Passing data – Calling Conventions
  - Managing local data
  - Recursion
- Array (cont'd)
- Buffer Overflow
- Floating-point operations



2D Array

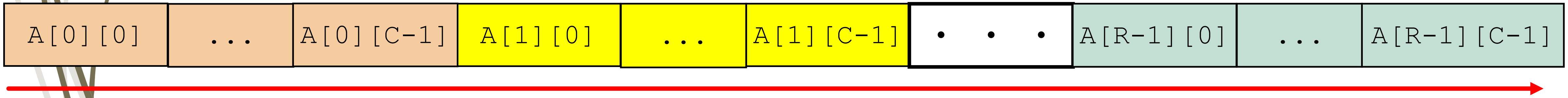
T A[R][C];

=> in C

- … is 2D array of data type ***T***
  R rows, **C** columns

=> in x86-64

- … is a contiguously allocated region of ***R*** * **C** * **L** bytes in memory where **L = sizeof(*T*)**
- Array layout in memory
  - Row-Major ordering –> Example using **int A[R][C];**

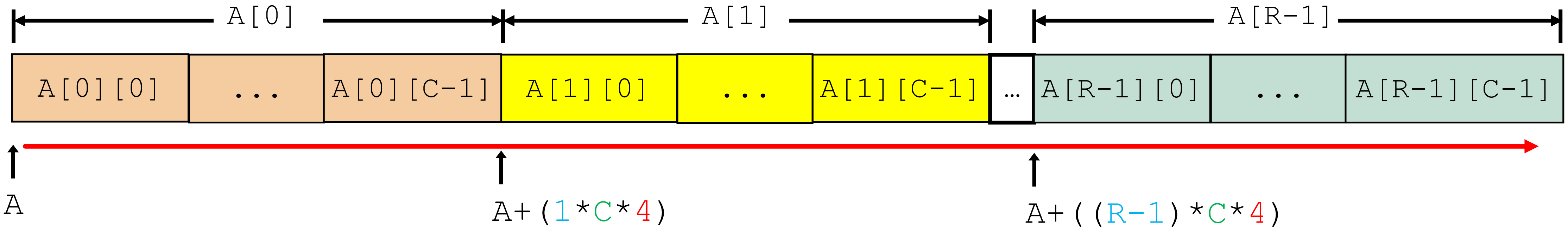


R*C*int Bytes

Accessing a row of 2D array

T A[R][C];

- **A[i]** is an array of **C** elements (row **i** of array **A**)
- Memory address of each row **A[i]**: **A + (i * C * L)**
  - where **A** is **base memory address**
- Can access other rows by incrementing **A** by **i * C * L**
- Example using **int A[R][C];**



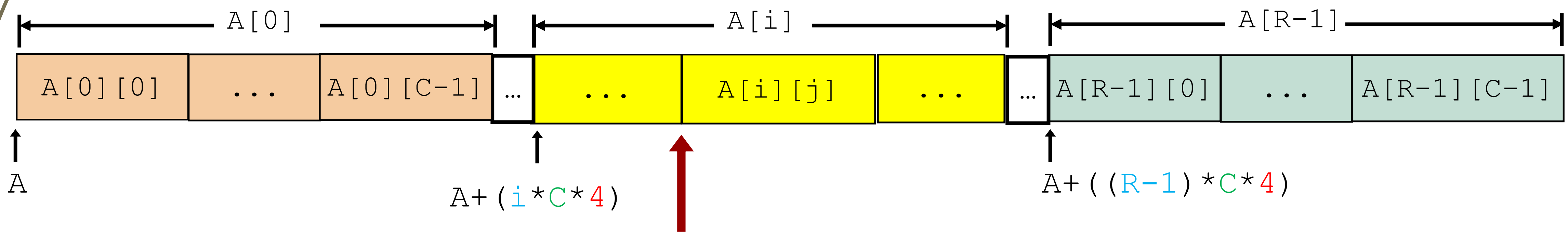
Accessing an element of 2D array

T A[R][C];

- `A[i][j]` is element of type *T*, which requires `L` bytes
- Memory address of each element `A[i][j]`:

  `A + (i * C * L) + (j * L) = A + (i * C + j) * L`

- Example using `int A[R][C];`



Homework

Example: `int A[3][5];`

- In memory:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

- Let's compute the memory address to access:
- `A[2]`:
- `A[2][3]`:

Demo - Accessing an element of 2D array

```c
#define N 4

char A[N][N] = {  1,  -2,   3,  -4,
                 -5,   6,  -7,   8,
                 -1,   2,  -3,   4,
                  5,  -6,   7,  -8};

char C[N][N];
```

```c
void main() {

    printf("Original matrix: \n");
    printMatrixByRow(A, N);

    printf("Copy: \n");
    copy(A, C, N);
    printMatrixByRow(C, N);

    return;
}

// Print N elements in a row
void printMatrixByRow(void *D, int n) {
```

+ **copy.s** on our course web site

Summary

- Manipulation of 2D arrays – in x86-64
  - From x86-64's perspective, a 2D array is a contiguously allocated region of *R* * C * `L` bytes in memory where `L = sizeof(T)` and *T* -> data type of elements stored in array
  - 2D Array layout in memory: Row-Major ordering
  - Memory address of each row `A[i]`: `A + (i * C * L)`
  - Memory address of each element `A[i][j]`:

`A + (i * C * L) + (j * L)`

`=> A + (i * C + j) * L`

Next Lecture

- Introduction
  - C program -> assembly code -> machine level code
- Assembly language basics: data, `move` operation
  - Memory addressing modes
- Operation `leaq` and Arithmetic & logical operations
- Conditional Statement – Condition Code + `cmovX`
- Loops
- Function call – Stack
  - Overview of Function Call
  - Memory Layout and Stack - x86-64 instructions and registers
  - Passing control
  - Passing data – Calling Conventions
  - Managing local data
  - Recursion
- Array
- Buffer Overflow
- Floating-point operations