Lecture 4: C/C++ Arrays and Structures

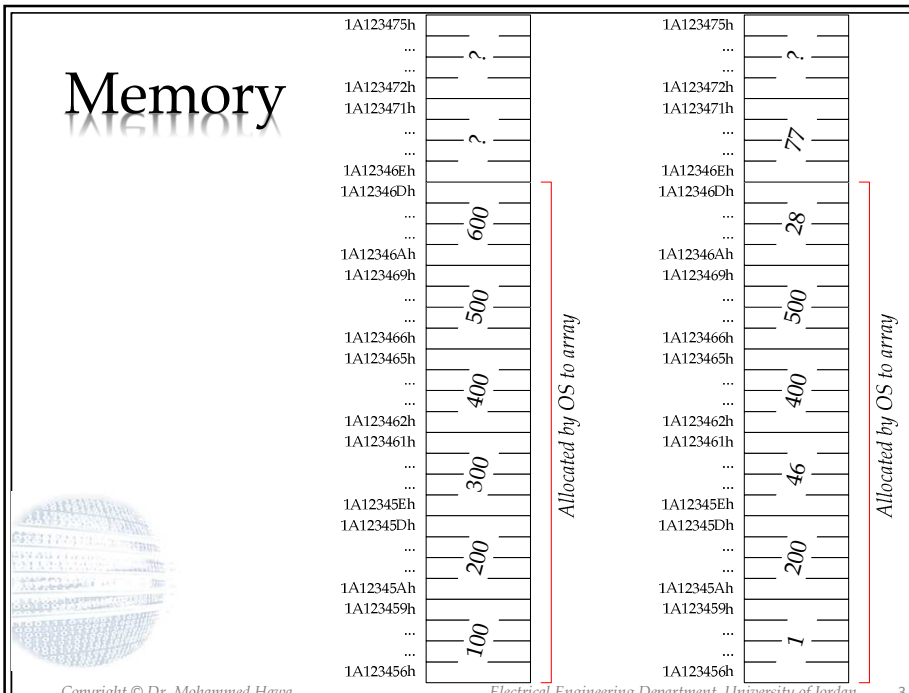
Dr. Mohammed Hawa
Electrical Engineering Department
University of Jordan

Arrays

```
int array[6] = {100, 200, 300, 400, 500, 600};

array[0] = 1;  // first element
array[2] = 46; // third element
array[5] = 28; // last element

array[6] = 77;  // valid code but bad at runtime (just a warning)
array[-4] = 55; // valid code but bad at runtime
```

- No runtime check on array bounds is done by the CPU (better performance if the CPU does not check, but can cause segmentation fault).

Memory



Rules of Thumb

```
float array[6] = {1.0, 2.0, 3.0, 4.0, 5.0, 6.0};
int k;

// know this
array          ==     &(array[0]);
array + k      ==     &(array[k]); // pointer arithmetic
*array         ==     array[0];
*(array + k)   ==     array[k];     // pointer arithmetic

// exampple
array[2]       = 4.6; // third element
*(array+2)     = 4.6; // third element (2 steps of 4)
```


Homework

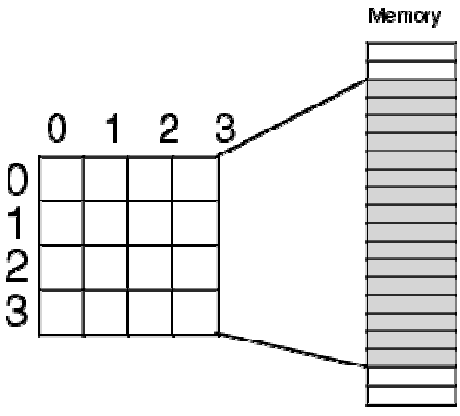
- Do you understand this?
- Verify the answer using Visual Studio.

```
int a[5];

a[3] = 128;
((short*) a)[7] = 2;
std::cout << a[3] << std::endl;
// answer is 131200!!
```


Multi-Dimensional Arrays

- C/C++ supports two-dimensional, three-dimensional, etc arrays.
- char array[3][3];
- int array[7][7][15][2];

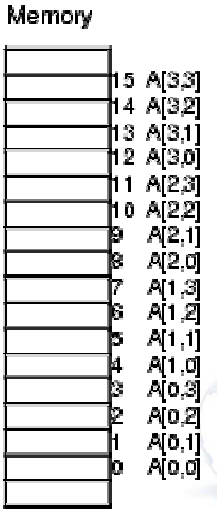
- Be aware of spatial locality of reference.



Row-Major Order: C/C++/C# etc

A:array [0..3,0..3] of char

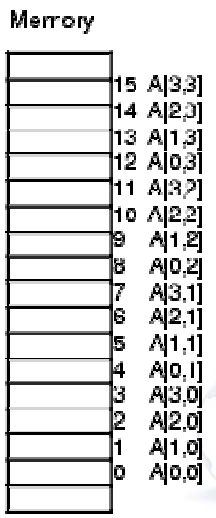
	0	1	2	3
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15



Column-Major Order: MATLAB, etc

A:array [0..3,0..3] of char

	0	1	2	3
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15

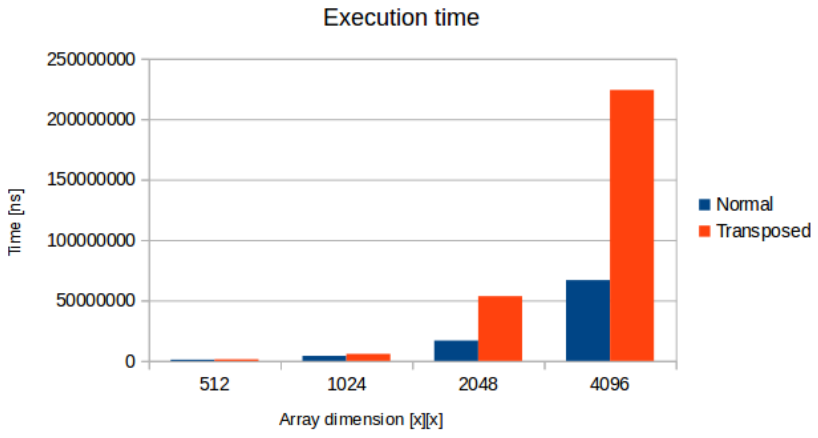


C/C++ Efficiency: Avoid Cache Miss

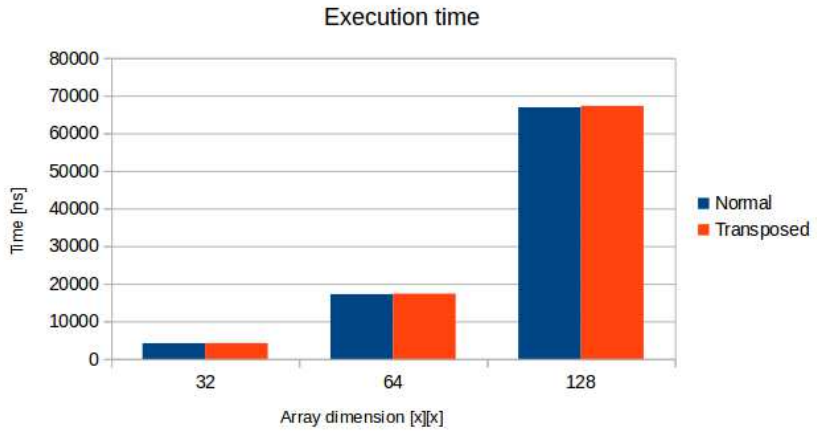
```
quint64 sumArrayRowByRow_Normal()
{
    quint64 sum = 0;

    for (size_t i = 0; i < HEIGHT; i++)
        for (size_t j = 0; j < WIDTH; j++)
            sum += array[i][j];

    return sum;
}
```

```
quint64 sumArrayColumnByColumn_Transposed()
{
    quint64 sum = 0;

    for (size_t i = 0; i < WIDTH; i++)
        for (size_t j = 0; j < HEIGHT; j++)
            sum += array[j][i];

    return sum;
}
```


Performance: Due to CPU Cache

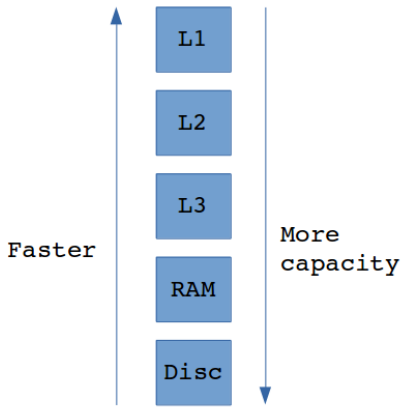


Depends on Cache Size



Know your hardware!

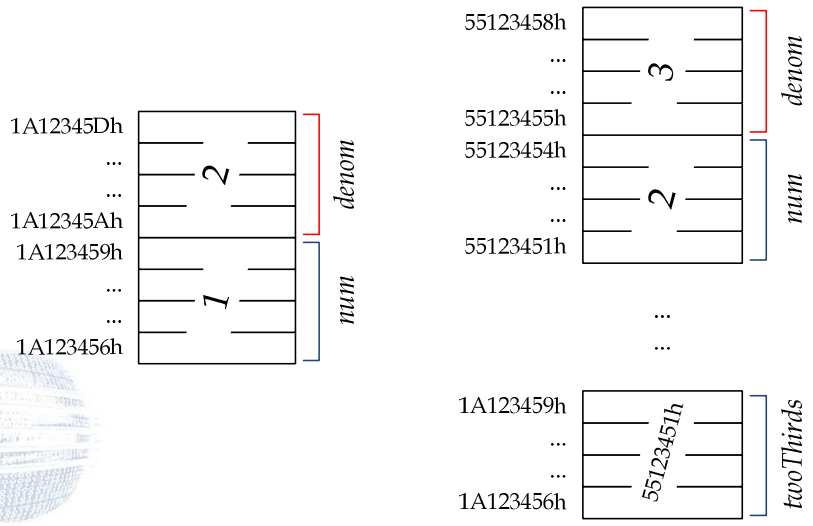
- L1 is the fastest.
  Hit latency ~4 cycles.
- L2 is bigger but slower.
  Hit latency: ~10 cycles.
- L3 is optional. Bigger and slower than L2.
  Hit latency: ~40 cycles.
- *Example:*
- L1 cache: 64K
- L2 cache: 256K
- L3 cache: 3072K



Structures

```
struct fraction
{
	int numerator;
	int denominator;
};

fraction half; // we can store 1/2

half.numerator = 1;
half.denominator = 2;

printf("Value: %d/%d\n",half.numerator, half.denominator);

fraction* twoThirds = new fraction; // C++ allocation

twoThirds->numerator = 2;
twoThirds->denominator = 3;

printf("Value: %d/%d\n", twoThirds->numerator, twoThirds->denominator);

delete twoThirds; // C++ memoery de-allocation
```


Memory



Tricky!

- What happens to memory if you execute the following code?

```
struct fraction
{
        int numerator;
        int denominator;
};

fraction secret;

((fraction*) &(secret.denominator))->numerator = 12;
((fraction*) &(secret.denominator))->denominator = 33;
```


C: malloc/free; C++: new/delete

```
typedef struct
{
        int    ID;
        char   name[20];
        double GPA;
} StudentRecord; // keep it in one place


StudentRecord *ahmed_ptr;

ahmed_ptr = (StudentRecord*) malloc(sizeof(StudentRecord));
// C memeory allocation
// malloc, free require #include <stdlib.h>

ahmed_ptr->ID = 11234;
strcpy(ahmed_ptr->name, "Ahmad"); // don't say ahmed_ptr->name = "Ahmad"
ahmed_ptr->GPA = 3.5;
// strcpy requires #include <string.h>

printf("Student Record: %d, %s, %f\n",ahmed_ptr->ID,
                        ahmed_ptr->name, (*ahmed_ptr).GPA);

free(ahmed_ptr);
```