Application of memory management in C: Cache-efficient algorithms

Sujoy Sinha Roy
School of Computer Science
University of Birmingham

Locality example

Both functions compute the sum of the elements of an input 2D matrix. Which one has better locality?

```
int sum_2d_array1(int a[N][M]){
        int i, j, sum=0;
        for(i=0; i<N; i++)
                for(j=0; j<M; j++)
                        sum =   sum + a[i][j];
        return sum;
}
```

Computes the sum over a row.
a[0][0]+a[0][1]+…
+a[1][0]+a[1][1]+…

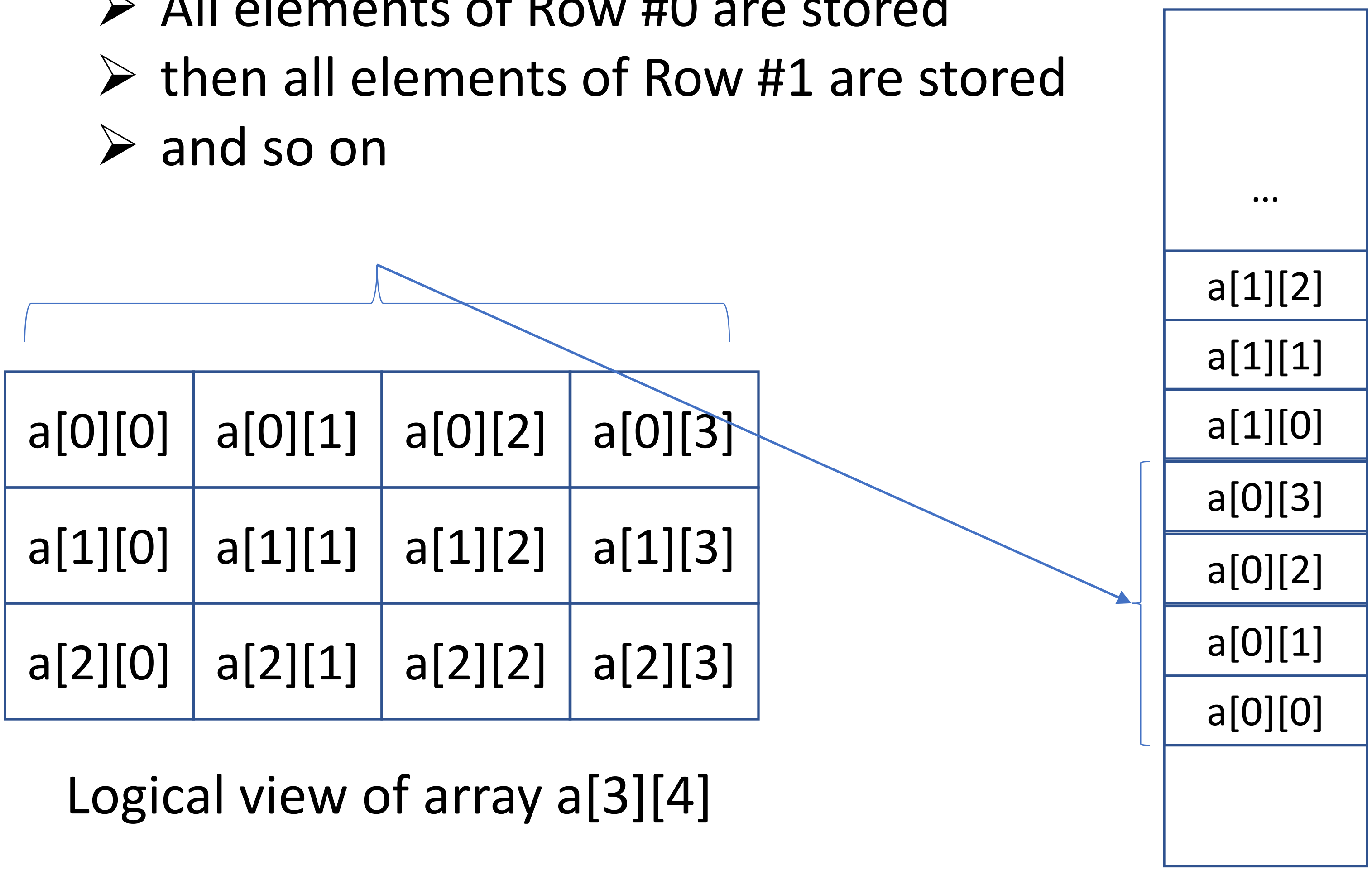
```
int sum_2d_array2(int a[N][M]){
        int i, j, sum=0;
        for(i=0; i<M; i++)
                for(j=0; j<N; j++)
                        sum =   sum + a[j][i];
        return sum;
}
```

Computes the sum over a column.
a[0][0]+a[1][0]+…
+a[0][1]+a[1][1]+…

Recap: Memory layout of two-dimensional array

C compiler stores 2D array in **row-major** order

- All elements of Row #0 are stored
- then all elements of Row #1 are stored
- and so on



Logical view of array a[3][4]

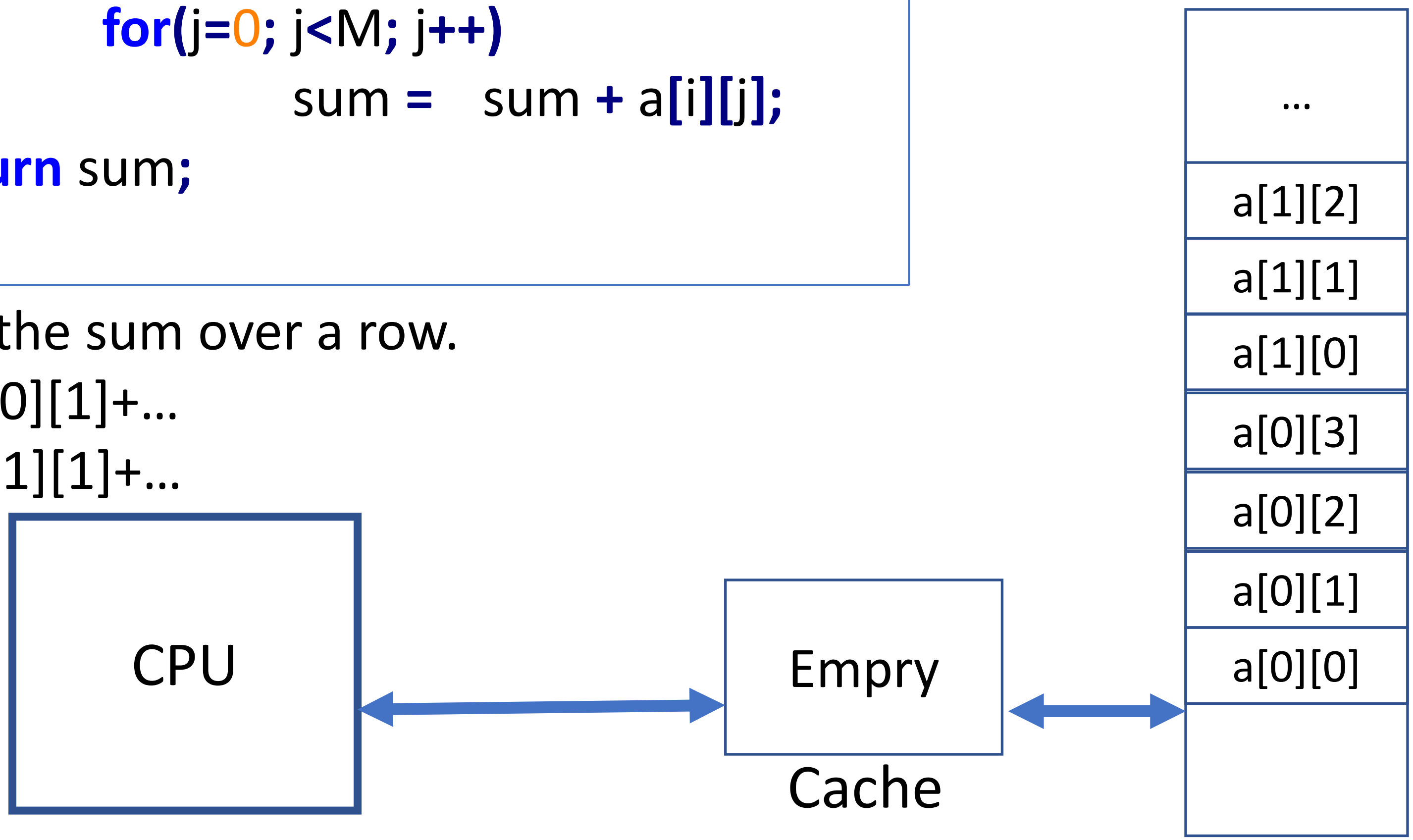
Memory layout

Locality example: the first case

```
int sum_2d_array1(int a[N][M]){
        int i, j, sum=0;
        for(i=0; i<N; i++)
                for(j=0; j<M; j++)
                        sum =   sum + a[i][j];
        return sum;
}
```

Array elements are initially in the Main Memory.

Computes the sum over a row.
a[0][0]+a[0][1]+…
+a[1][0]+a[1][1]+…



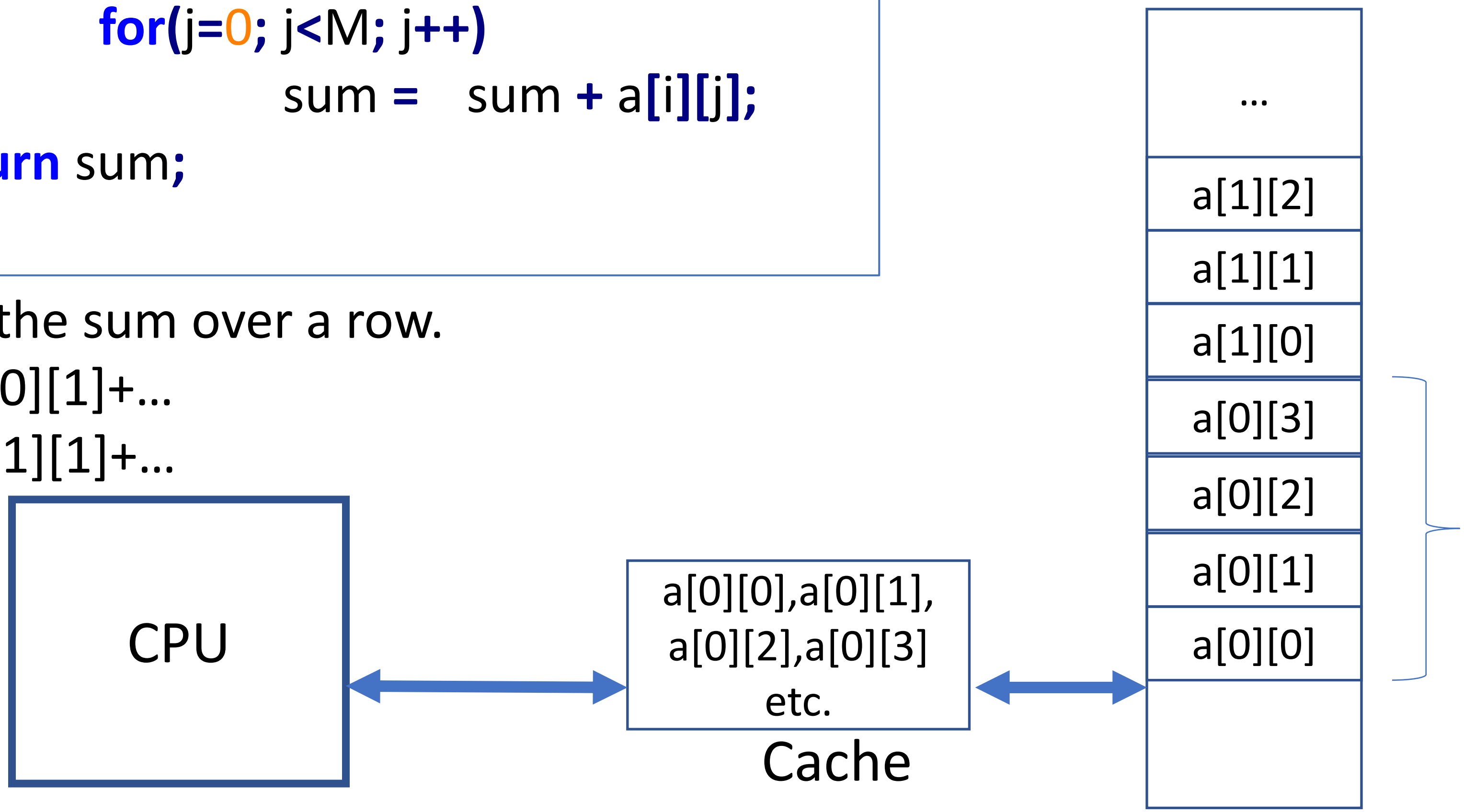
1. CPU requires a[0][0] to compute sum = sum + a[0][0]

Locality example: the first case

```
int sum_2d_array1(int a[N][M]){
        int i, j, sum=0;
        for(i=0; i<N; i++)
                for(j=0; j<M; j++)
                        sum =   sum + a[i][j];
        return sum;
}
```

Array elements are initially in the Main Memory.

Computes the sum over a row.
a[0][0]+a[0][1]+…
+a[1][0]+a[1][1]+…



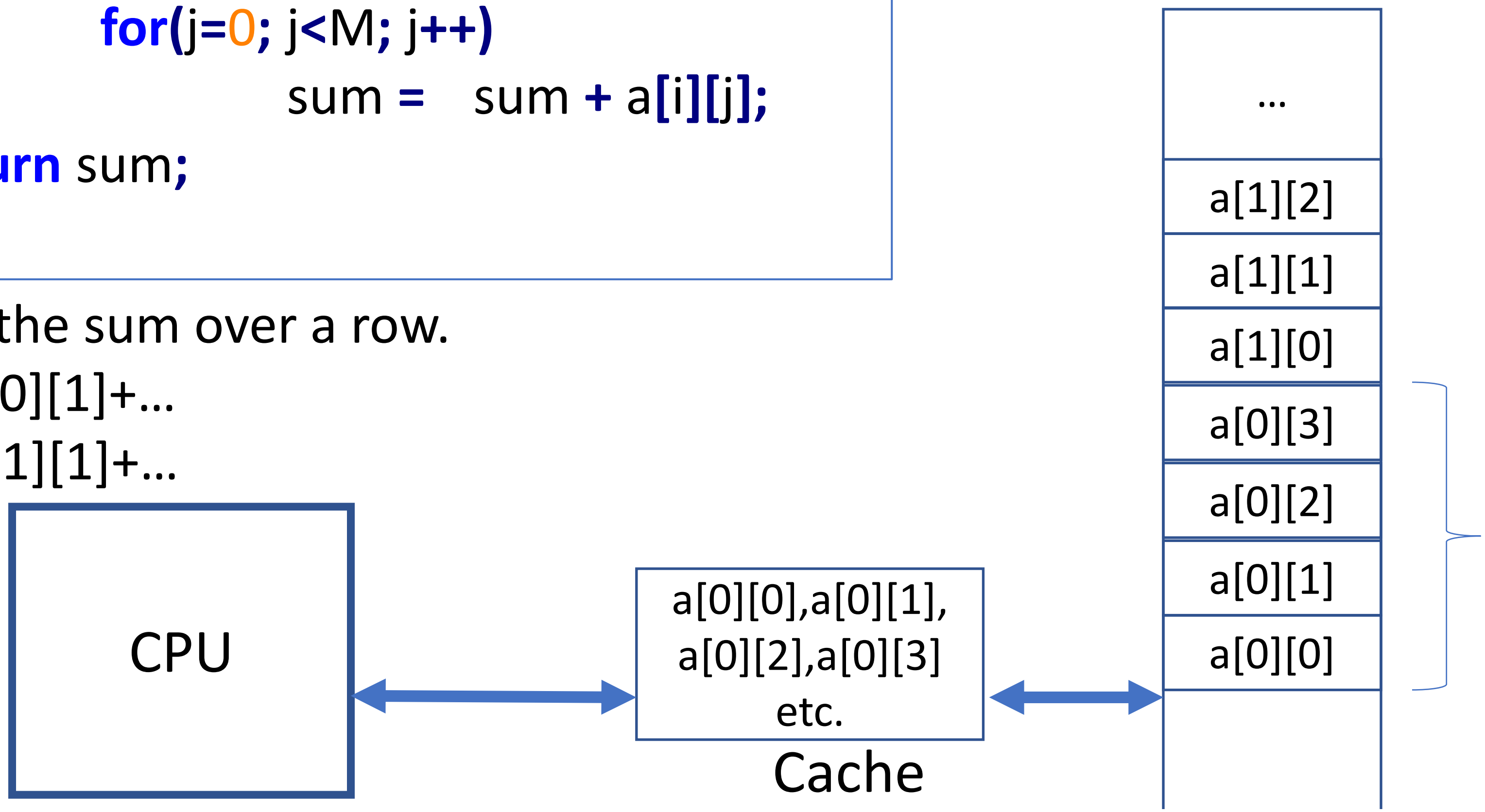
1. CPU requires a[0][0] to compute sum = sum + a[0][0]
2. Due to ‘spatial locality’, say a[0][0], a[0][1], a[0][2] etc. are loaded from Main Memory to Cache. [100 cycles are spent to access Main Memory]

Locality example: the first case

```c
int sum_2d_array1(int a[N][M]){
        int i, j, sum=0;
        for(i=0; i<N; i++)
                for(j=0; j<M; j++)
                        sum =   sum + a[i][j];
        return sum;
}
```

Array elements are initially in the Main Memory.

Computes the sum over a row.
a[0][0]+a[0][1]+…
+a[1][0]+a[1][1]+…



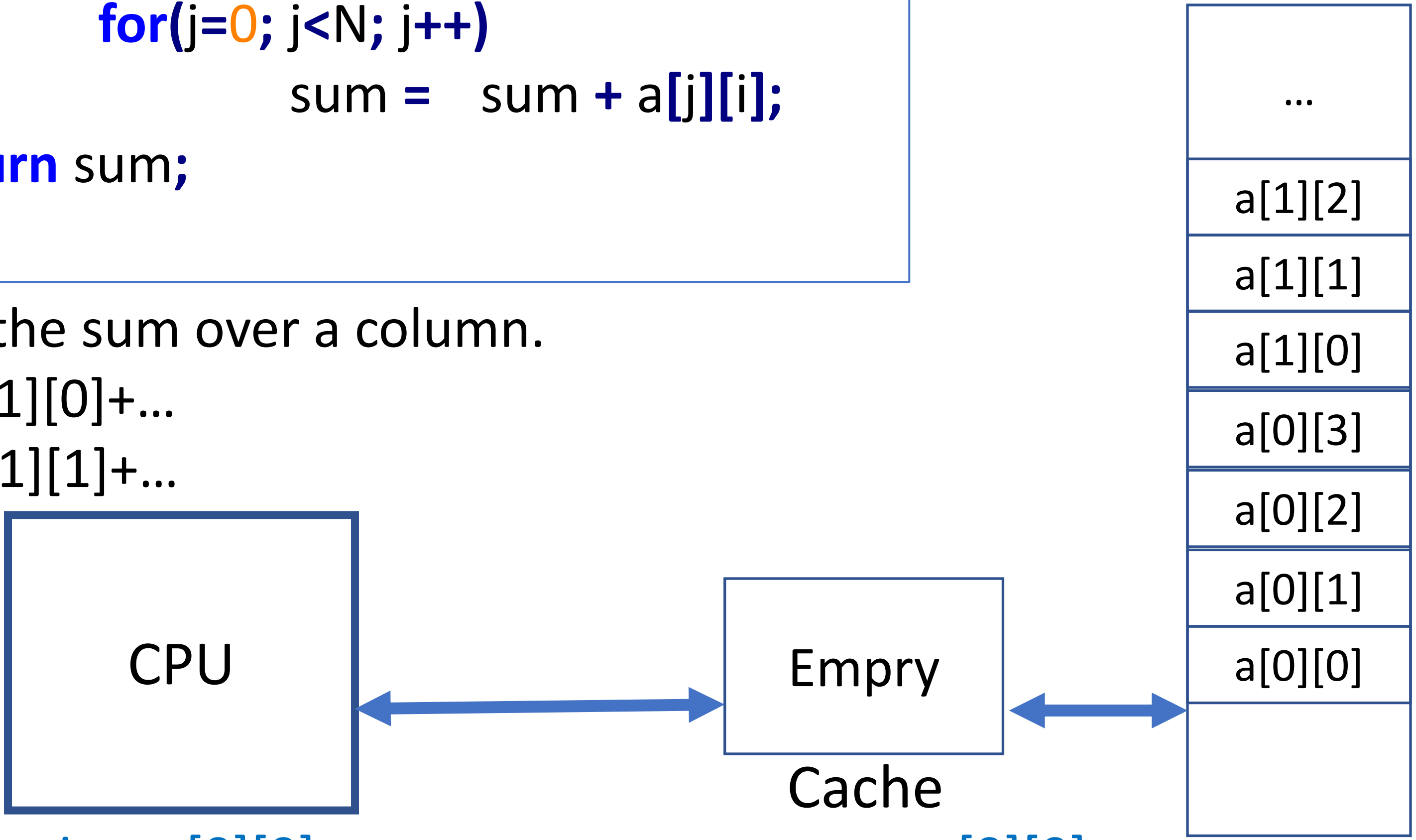
3. CPU computes sum=a[0][0]+a[0][1]+a[0][2] … by reading the elements from Cache. [Each access takes 4 cycles]
4. **Advantage: Cache hit happens most of the times.**

Locality example: the second case

```
int sum_2d_array2(int a[N][M]){
        int i, j, sum=0;
        for(i=0; i<M; i++)
                for(j=0; j<N; j++)
                        sum =   sum + a[j][i];
        return sum;
}
```

Computes the sum over a column.
a[0][0]+a[1][0]+…
+a[0][1]+a[1][1]+…

Array elements are initially in the Main Memory.



1. CPU requires a[0][0] to compute sum = sum + a[0][0]

Locality example: the second case

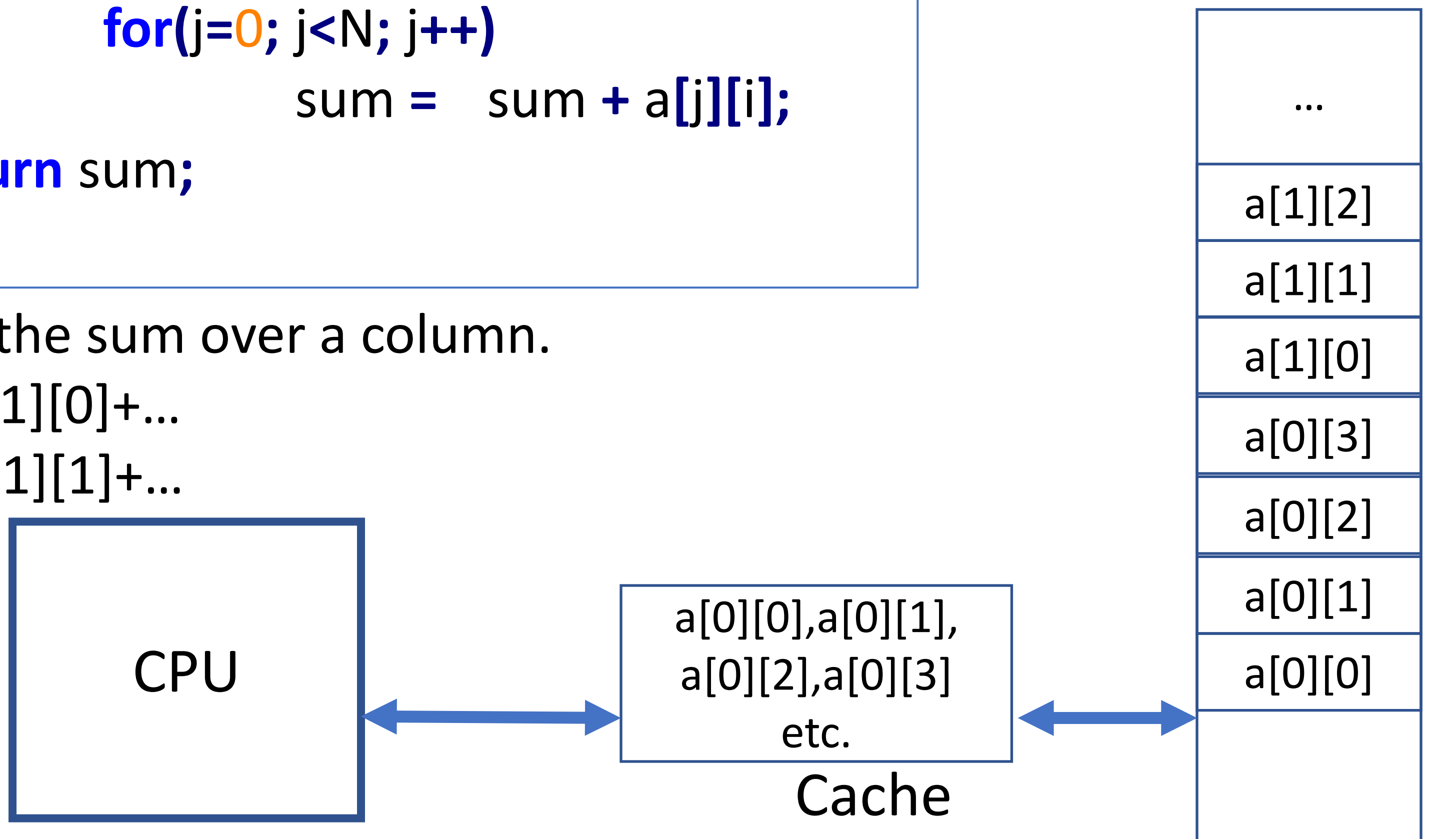
```
int sum_2d_array2(int a[N][M]){
        int i, j, sum=0;
        for(i=0; i<M; i++)
                for(j=0; j<N; j++)
                        sum =   sum + a[j][i];
        return sum;
}
```

Array elements are initially in the Main Memory.

Computes the sum over a column.

a[0][0]+a[1][0]+…

+a[0][1]+a[1][1]+…



1. CPU requires a[0][0] to compute sum = sum + a[0][0]
2. Due to 'spatial locality', say a[0][0], a[0][1], a[0][2] etc. are loaded from Main Memory to Cache. [100 cycles are spent to access Main Memory]

Locality example: the second case

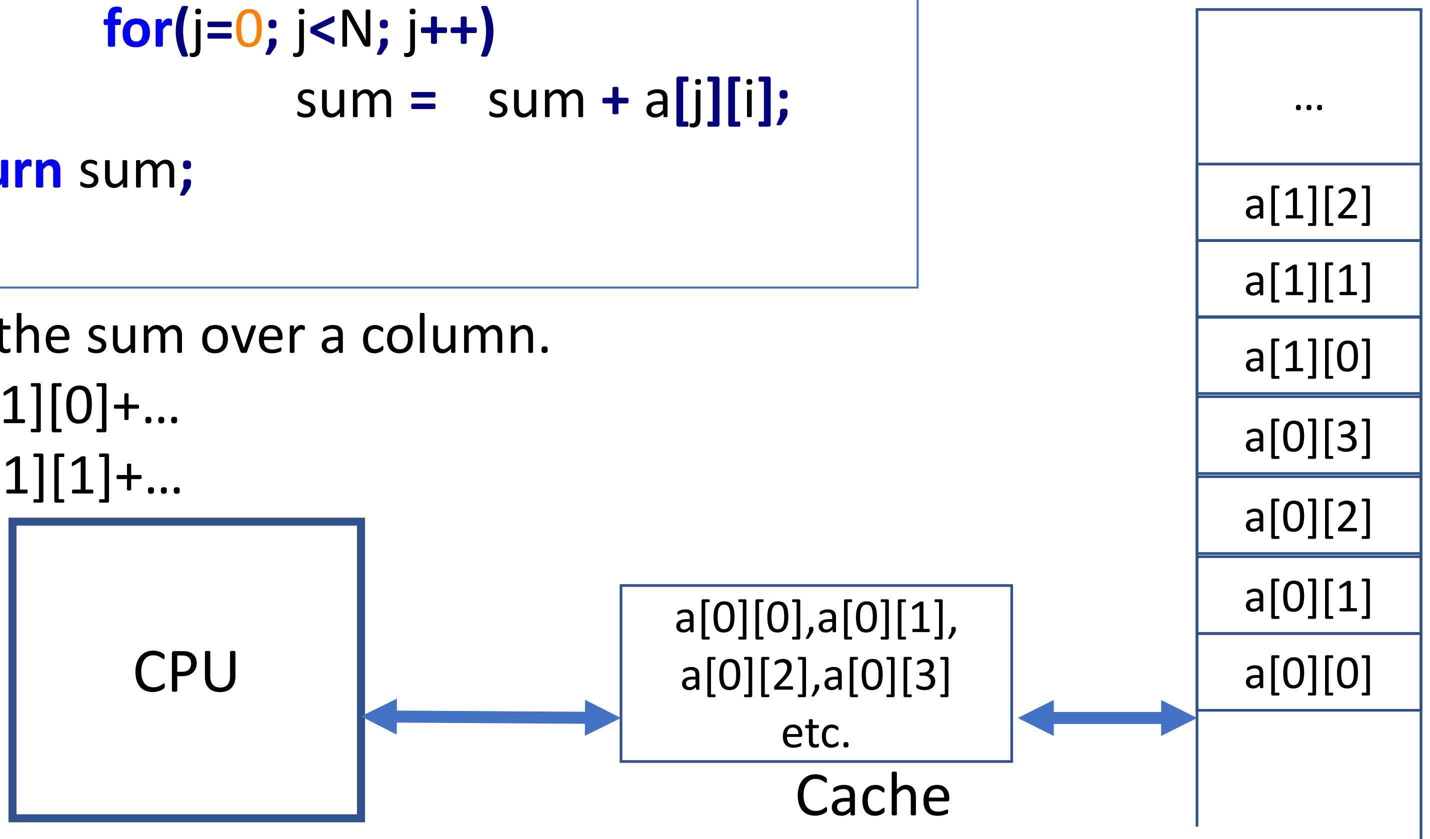
```c
int sum_2d_array2(int a[N][M]){
        int i, j, sum=0;
        for(i=0; i<M; i++)
                for(j=0; j<N; j++)
                        sum =   sum + a[j][i];
        return sum;
}
```

Array elements are initially in the Main Memory.

Computes the sum over a column.

a[0][0]+a[1][0]+…

+a[0][1]+a[1][1]+…



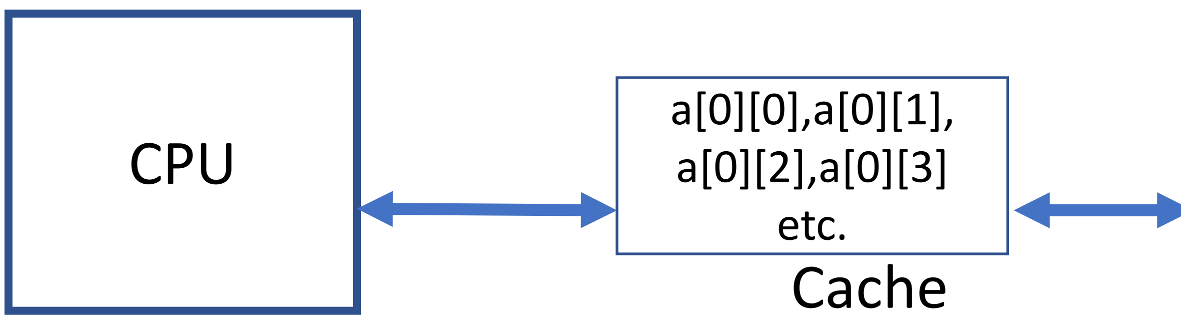
3. However, CPU to computes sum=sum+a[0][0]+a[1][0]+a[2][0]+…
4. Since, none of {a[1][0], a[2][0], …} are in the Cache. Hence, Main Memory is accessed for each of them. [100 cycles for every access]

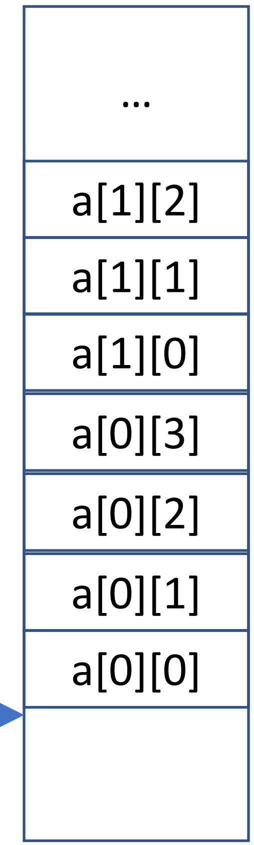
Locality example: the second case

```
int sum_2d_array2(int a[N][M]){
        int i, j, sum=0;
        for(i=0; i<M; i++)
                for(j=0; j<N; j++)
                        sum =   sum + a[j][i];
        return sum;
}
```

Array elements are initially in the Main Memory.

Computes the sum over a column.
a[0][0]+a[1][0]+…
+a[0][1]+a[1][1]+…





Disadvantage: Cache miss happens always

Locality example: conclusions

```
int sum_2d_array1(int a[N][M]){
        int i, j, sum=0;
        for(i=0; i<N; i++)
                for(j=0; j<M; j++)
                        sum =   sum + a[i][j];
        return sum;
}
```

High Cache hit rate! Hence, much faster execution ☺

```
int sum_2d_array2(int a[N][M]){
        int i, j, sum=0;
        for(i=0; i<M; i++)
                for(j=0; j<N; j++)
                        sum =   sum + a[j][i];
        return sum;
}
```

Always Cache miss. Order of magnitude slower execution ☹

This is where the difference between a Java programmer and a C programmer becomes apparent.

Theory vs practice

"In theory there is no difference between theory and practice. But in practice there is." - Yogi Berra

We also see 'theory vs practice' when we run algorithms.