Introduction to 8086 Assembly

Lecture 17

2D and N-D Arrays

2D Arrays

0	0	0	0	0	0
0	1	2	3	4	5
0	2	4	6	8	10
0	3	6	9	12	15
0	4	8	12	16	20

- tabular data
- rows and columns

2D Arrays

0	0	0	0	0	0
0	1	2	3	4	5
0	2	4	6	8	10
0	3	6	9	12	15
0	4	8	12	16	20
0	5	10	15	20	25

$$A = \begin{pmatrix} 3 & -5 & 4 \\ 9 & 8 & -7 \\ -6 & 4 & 2 \end{pmatrix}, B = \begin{pmatrix} -2 & -1 & 1 \\ 5 & -7 & 6 \\ 9 & 3 & 2 \end{pmatrix}$$

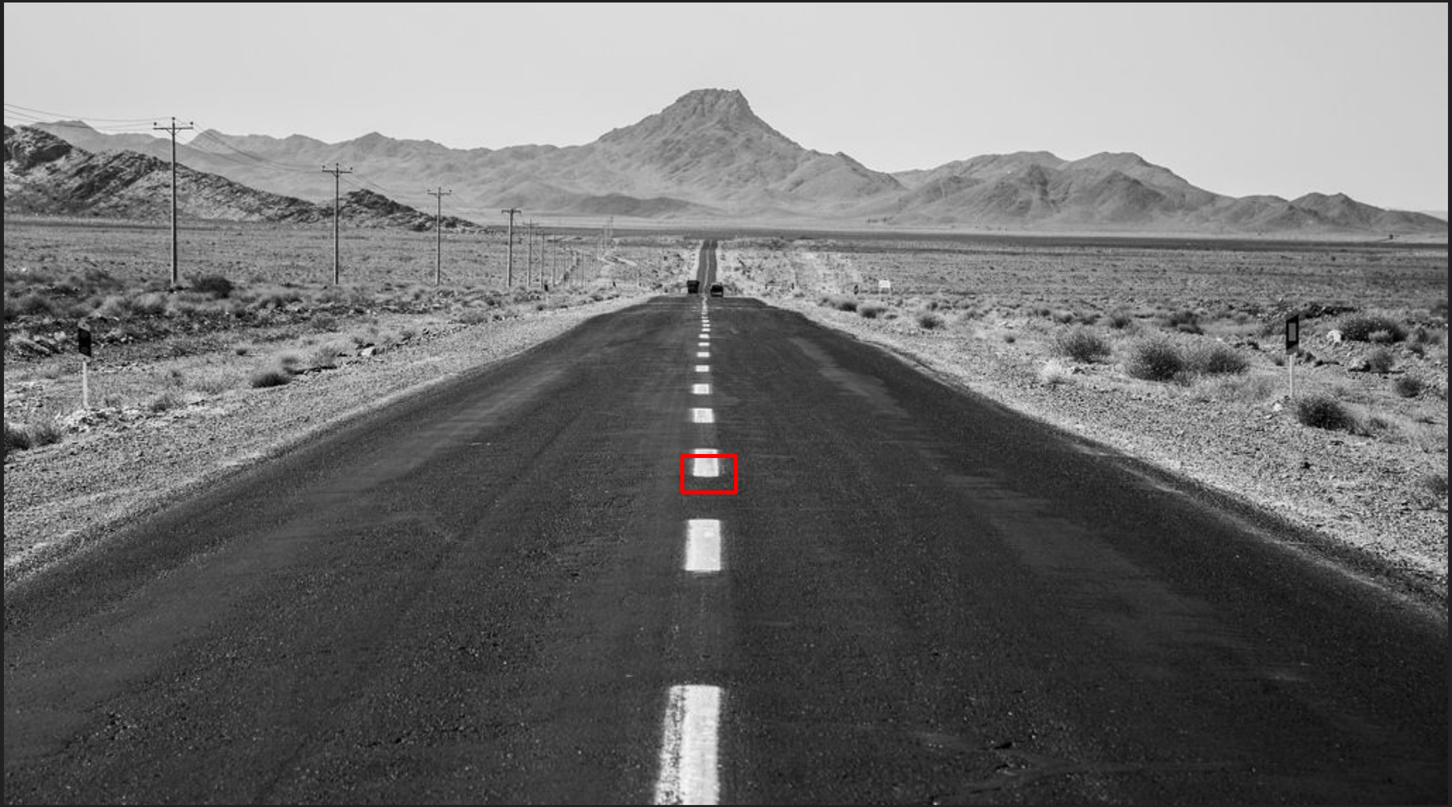
https://advancedmathclubsk.weebly.com/matrices.html

2D Arrays

K. N. Toosi
University of Technology

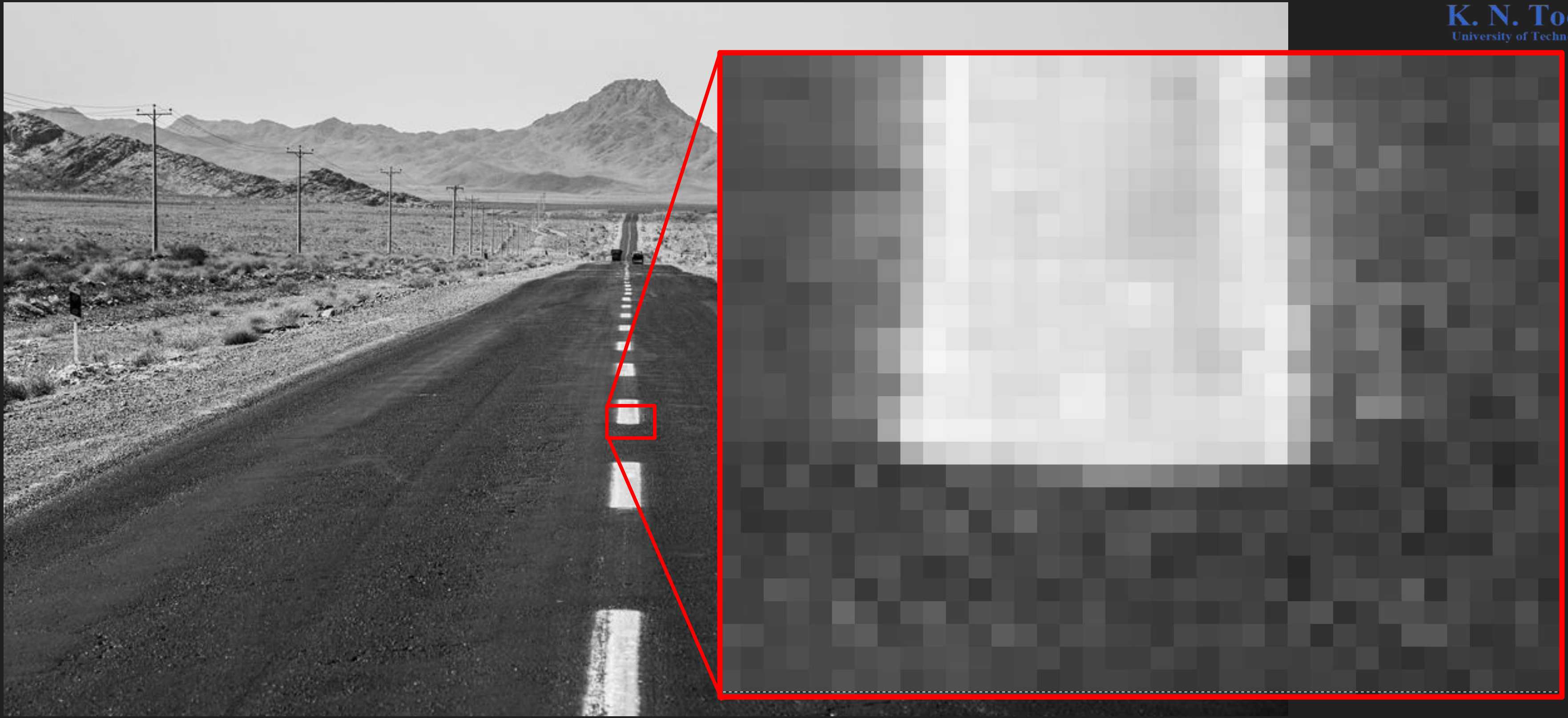
https://hiveminer.com/Tags/desert,isfahan/Recent

2D Arrays



https://hiveminer.com/Tags/desert,isfahan/Recent

2D Arrays



https://hiveminer.com/Tags/desert,isfahan/Recent

2D Arrays

K. N. Toosi
University of Technology

```
int a[3][4] = {{1, 3, 5, 7},
               {2, 4, 6, 8},
               {4,11,-1,7}};
```

```
printf("%d\n", a[1][2]);
```

	0	1	2	3
0	1	3	5	7
1	2	4	6	8
2	4	11	-1	7

2D Arrays

K. N. Toosi
University of Technology

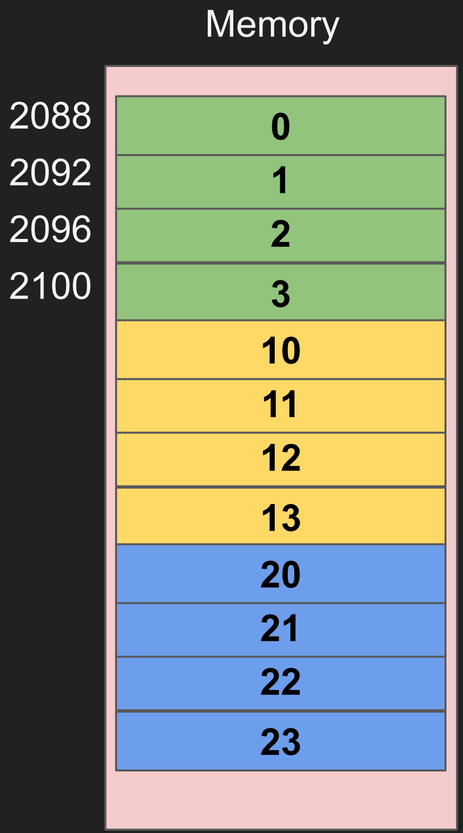
```
int a[3][4] = {{1, 3, 5, 7},
               {2, 4, 6, 8},
               {4,11,-1,7}};
```

```
printf("%d\n", a[i][j]);
```

	0	1	2	3
0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

How to implement 2D arrays?

	0	1	2	3
0	0	1	2	3
1	10	11	12	13
2	20	21	22	23

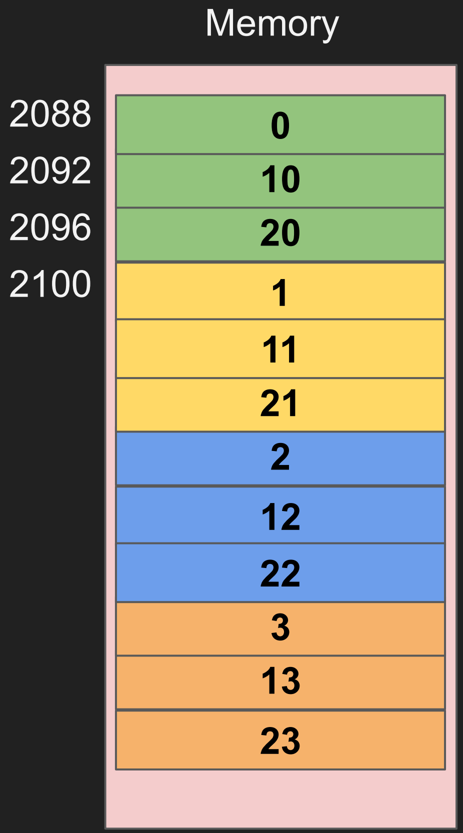


Memory

2088	0
2092	1
2096	2
2100	3
	10
	11
	12
	13
	20
	21
	22
	23

row by row

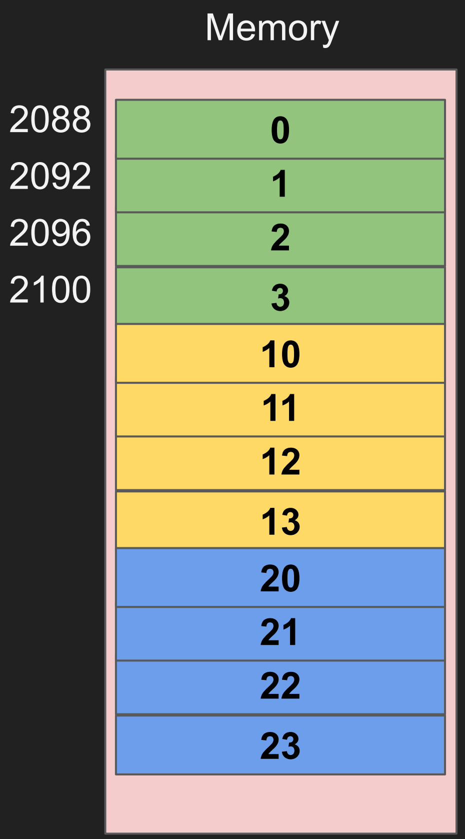
	0	1	2	3
0	0	1	2	3
1	10	11	12	13
2	20	21	22	23



Memory

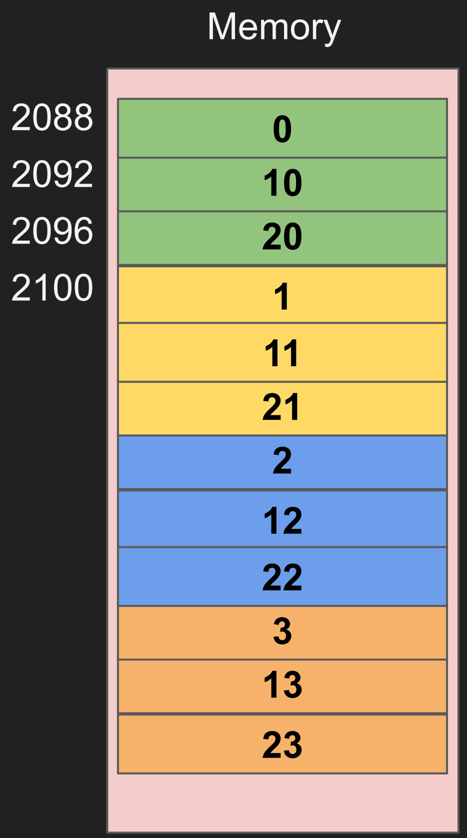
2088	0
2092	10
2096	20
2100	1
	11
	21
	2
	12
	22
	3
	13
	23

column by column

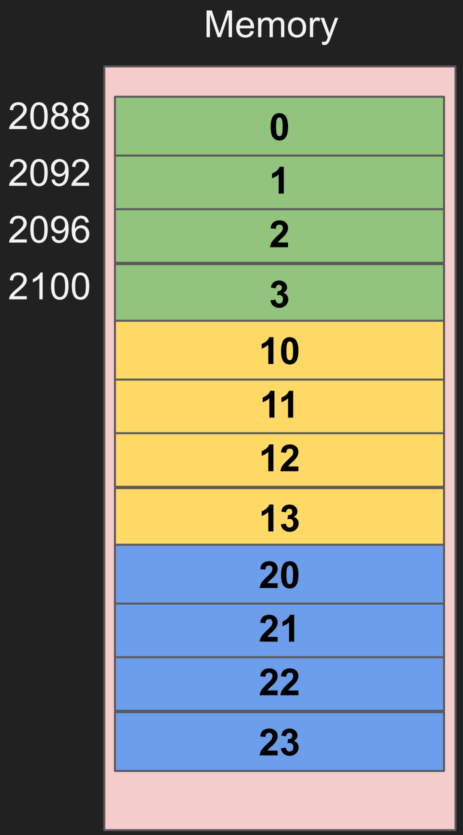


Row Major

	0	1	2	3
0	0	1	2	3
1	10	11	12	13
2	20	21	22	23

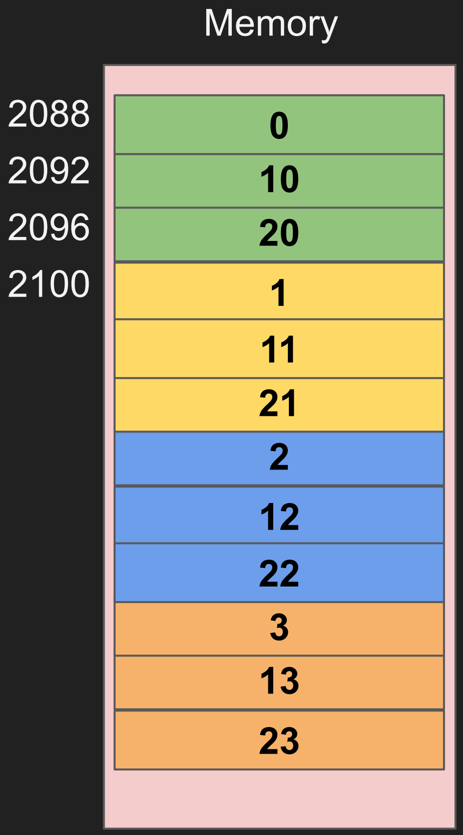


Column Major

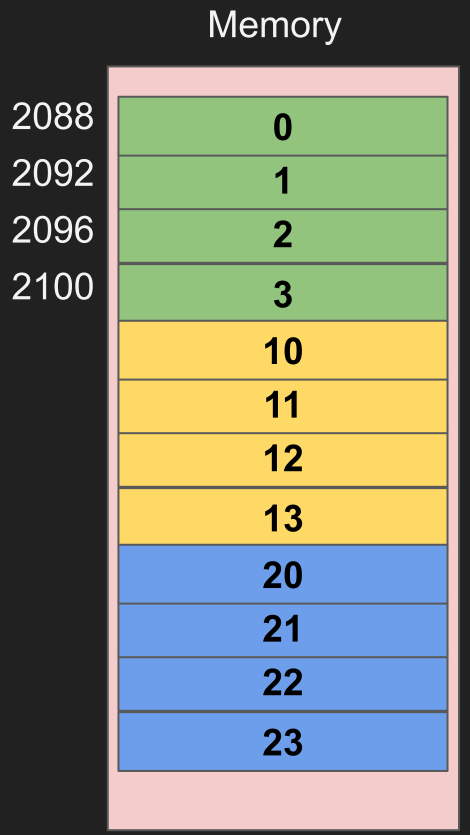


Row Major
(C,C++,Pascal, Python-numpy)

	0	1	2	3
0	0	1	2	3
1	10	11	12	13
2	20	21	22	23



Column Major
(Fortran, Matlab, R, …)



Memory

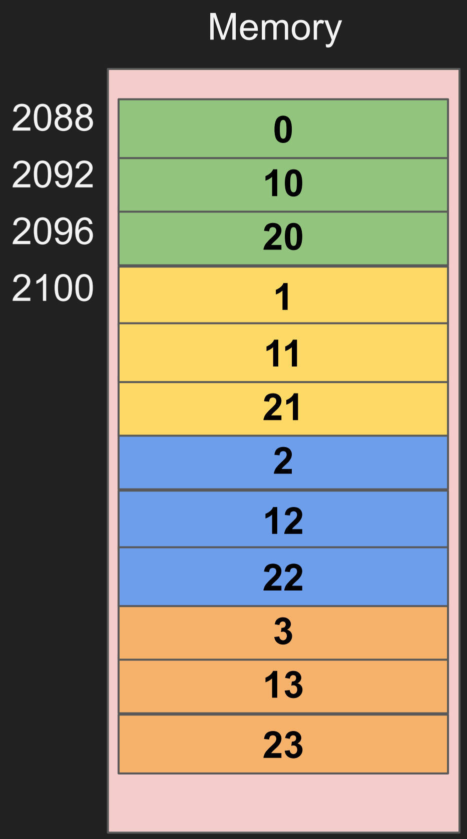
2088
2092
2096
2100

0
1
2
3
10
11
12
13
20
21
22
23

Row Major
(C,C++,Pascal, Python-numpy)

	0	1	2	3
0	0	1	2	3
1	10	11	12	13
2	20	21	22	23

Assembly?

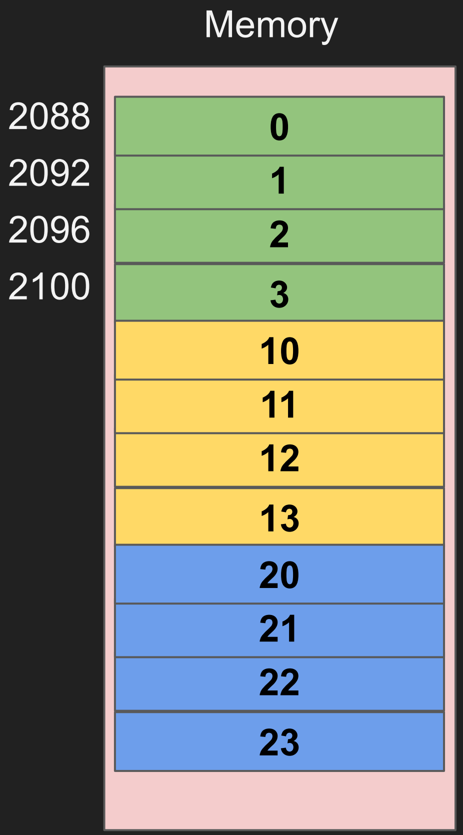


Memory

2088
2092
2096
2100

0
10
20
1
11
21
2
12
22
3
13
23

Column Major
(Fortran, Matlab, R, …)

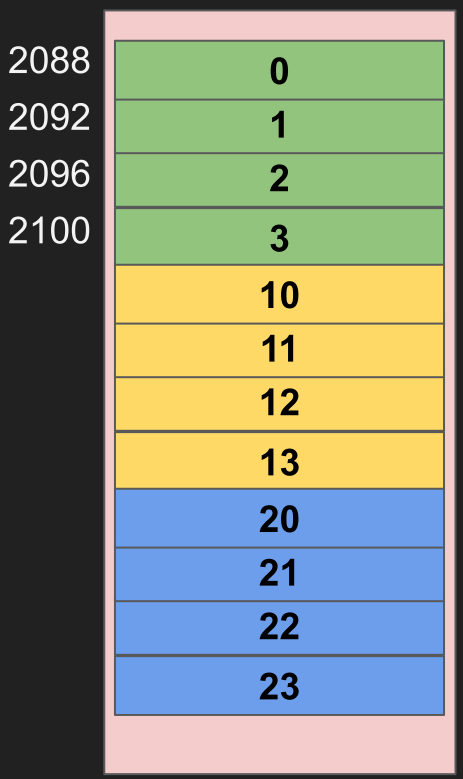


Row Major
(C,C++,Pascal, numpy)

	0	1	2	3
0	0	1	2	3
1	10	11	12	13
2	20	21	22	23

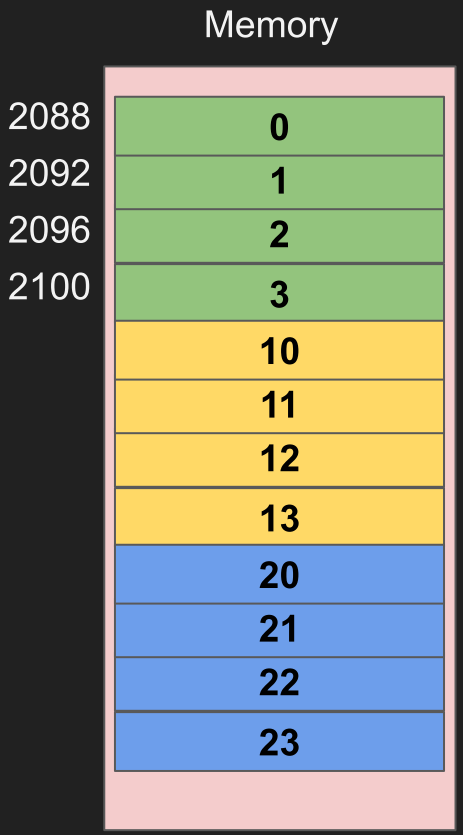
a[i][0] : ?

Memory



Row Major
(C,C++,Pascal, numpy)

	0	1	2	3
0	0	1	2	3
1	10	11	12	13
2	20	21	22	23



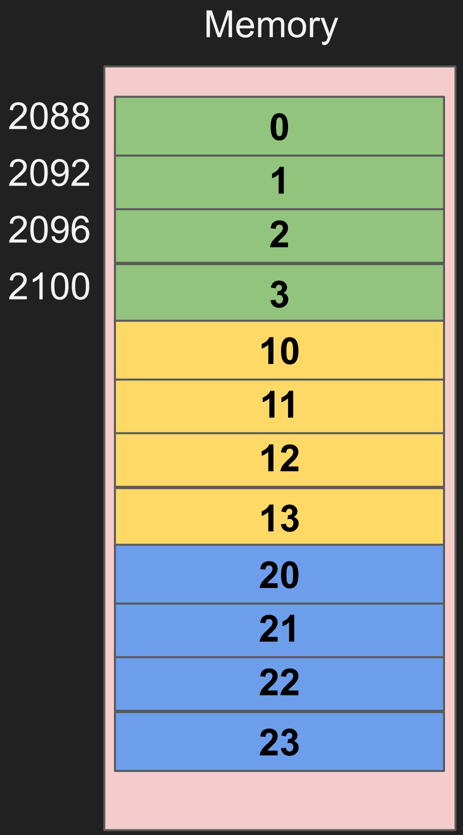
Memory

2088	0
2092	1
2096	2
2100	3
	10
	11
	12
	13
	20
	21
	22
	23

Row Major
(C,C++,Pascal, numpy)

```
a[i][0] : 4*i
```

	0	1	2	3
0	0	1	2	3
1	10	11	12	13
2	20	21	22	23

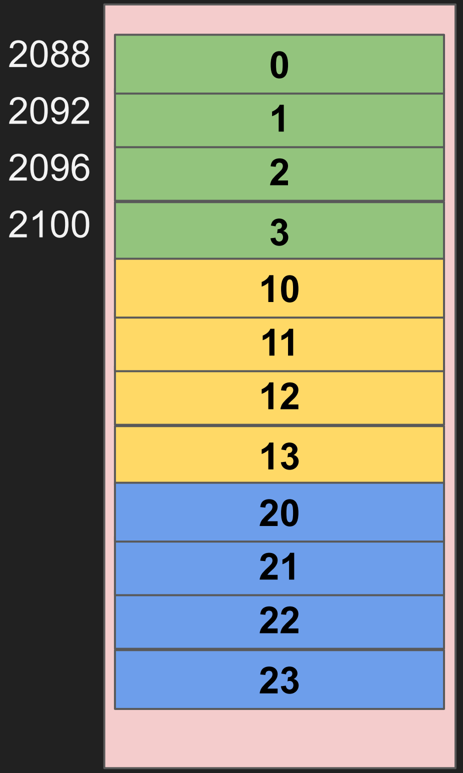


Row Major
(C,C++,Pascal, numpy)

	0	1	2	3
0	0	1	2	3
1	10	11	12	13
2	20	21	22	23

```
a[i][0] : 4*i
a[i][1] : ?
```

Memory



2088 0

2092 1

2096 2

2100 3

10

11

12

13

20

21

22

23

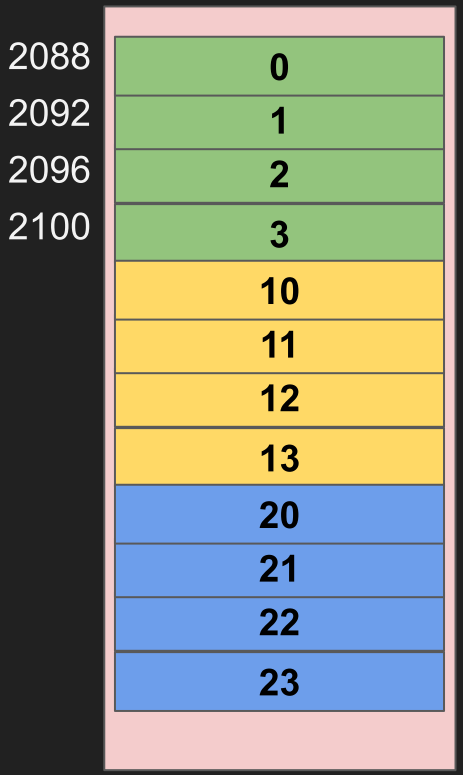
Row Major
(C,C++,Pascal, numpy)

	0	1	2	3
0	0	1	2	3
1	10	11	12	13
2	20	21	22	23

```
a[i][0] : 4*i
a[i][1] : 4*i+1
```

K. N. Toosi
University of Technology

Memory

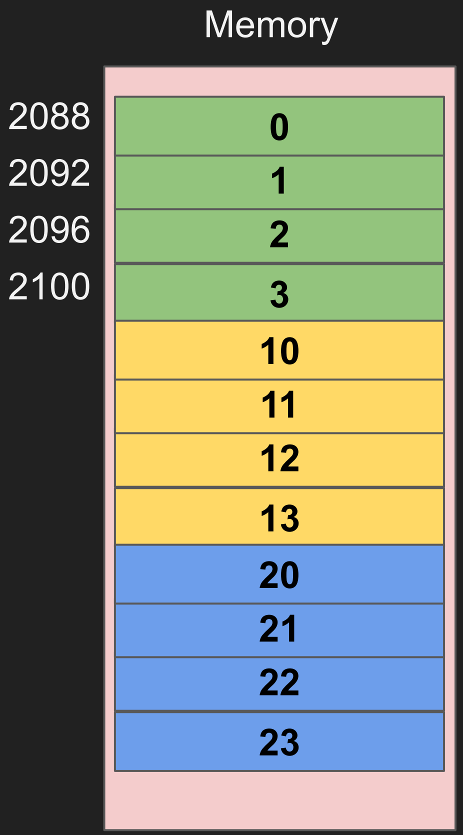


2088	0
2092	1
2096	2
2100	3
	10
	11
	12
	13
	20
	21
	22
	23

Row Major
(C,C++,Pascal, numpy)

	0	1	2	3
0	0	1	2	3
1	10	11	12	13
2	20	21	22	23

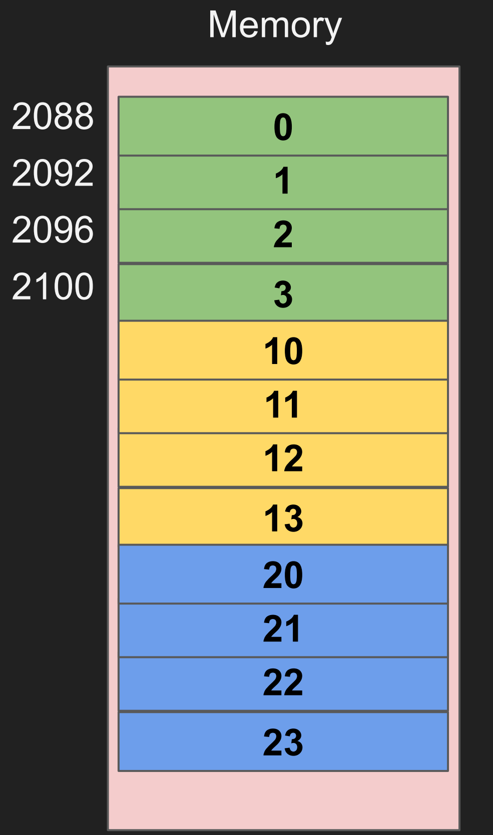
```
a[i][0] : 4*i
a[i][j] : ?
```



**Row Major
(C,C++,Pascal, numpy)**

	0	1	2	3
0	0	1	2	3
1	10	11	12	13
2	20	21	22	23

```
a[i][0] : 4*i
a[i][j] : 4*i+j
```

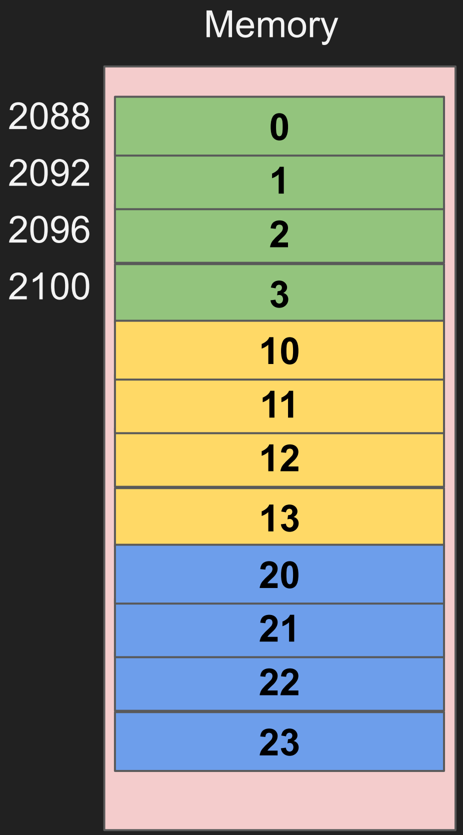


Row Major
(C,C++,Pascal, numpy)

	0	1		n-1
0	0	1	...	9
1	10	11	...	19
	⋮	⋮		⋮
m-1	80	81	...	89

```
int a[m][n];
```

```
a[i][j] : n*i + j
```

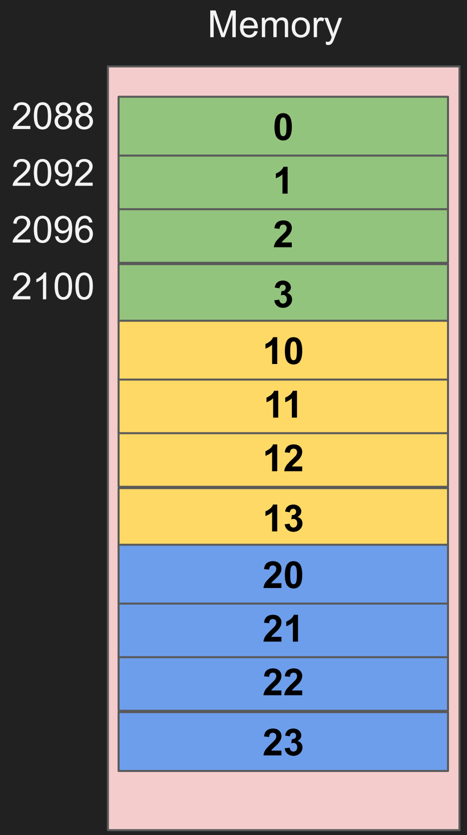


Row Major
(C,C++,Pascal, numpy)

	0	1		n-1
0	0	1	...	9
1	10	11	...	19
	⋮	⋮		⋮
m-1	80	81	...	89

```
int a[m][n];
```

```
a[i][j]
offset = (n*i+j)*sizeof(int)
```



Row Major
(C,C++,Pascal, numpy)

	0	1		n-1
0	0	1	...	9
1	10	11	...	19
	⋮	⋮		⋮
m-1	80	81	...	89

```
int a[m][n];
```

```
a[i][j] : n*i + j
```

to find `a[i][j]` we need to know

- element size?
- m (no. of rows of a)?
- n (no. of columns of a)?

Other standards?

- Column major?
    - `a[i][j] : ?`

	1	2	3	4
1	0	1	2	3
2	10	11	12	13
3	20	21	22	23

Other standards?

- Column major?
  - `a[i][j] : i+3*j`

	1	2	3	4
1	0	1	2	3
2	10	11	12	13
3	20	21	22	23

Other standards?

- Column major?
    - `a[i][j] : i+3*j`
- Index starting at 1
    - `a[i][j] : ?`

	1	2	3	4
1	0	1	2	3
2	10	11	12	13
3	20	21	22	23

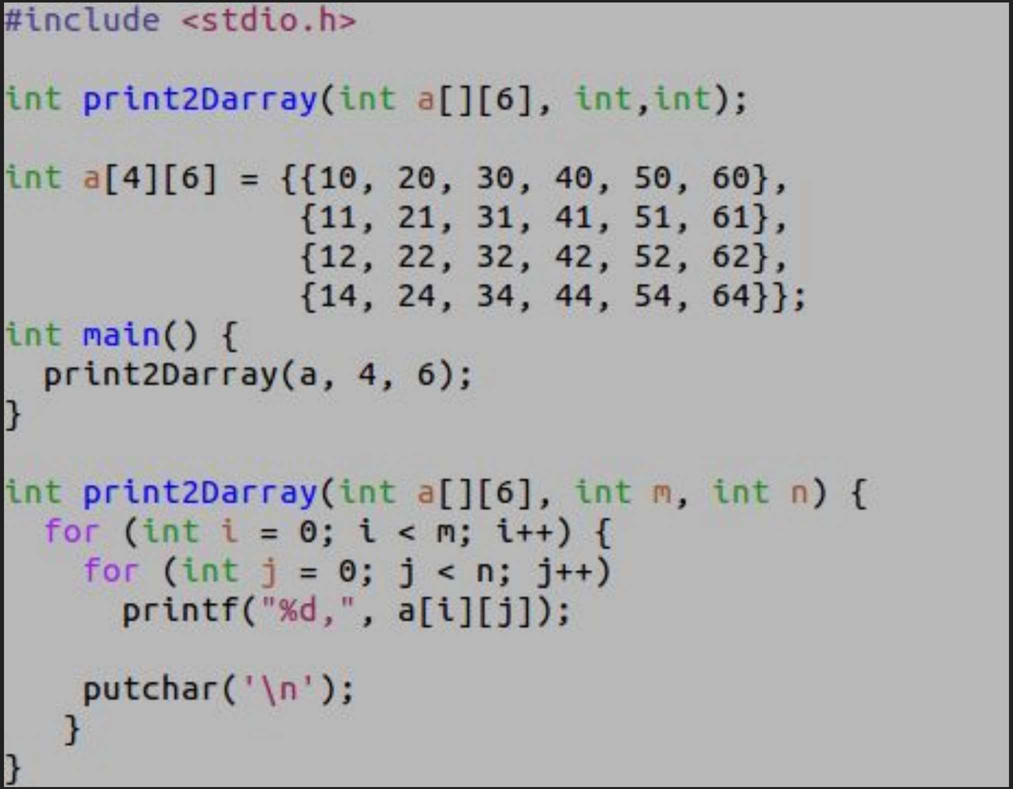
Other standards?

- Column major?
  - `a[i][j] : i+3*j`

- Index starting at 1
  - `a[i][j] : 4*(i-1)+j-1`
  - `a[i][j] : 4*(i-1)+j`

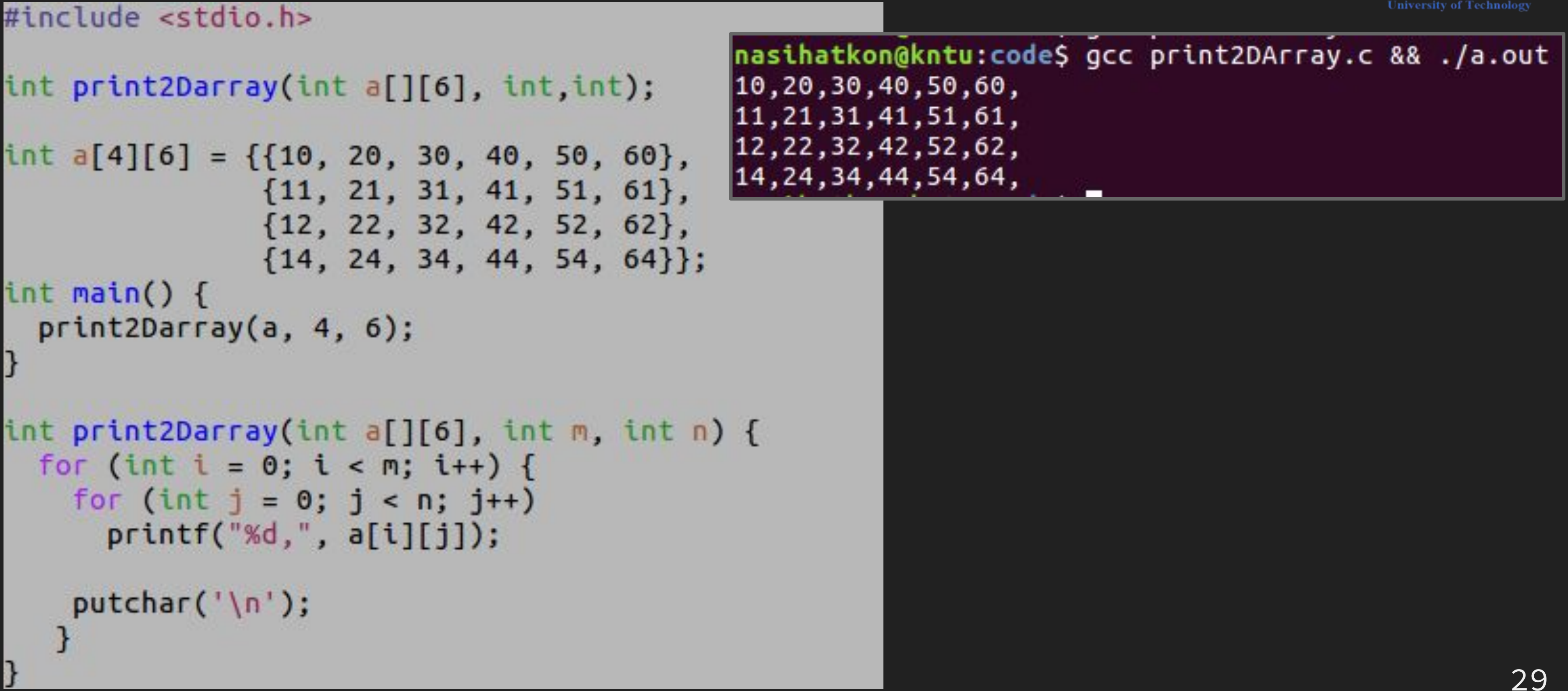
	1	2	3	4
1	0	1	2	3
2	10	11	12	13
3	20	21	22	23

Example: Print a 2D Array



```
#include <stdio.h>

int print2Darray(int a[][6], int,int);

int a[4][6] = {{10, 20, 30, 40, 50, 60},
               {11, 21, 31, 41, 51, 61},
               {12, 22, 32, 42, 52, 62},
               {14, 24, 34, 44, 54, 64}};
int main() {
  print2Darray(a, 4, 6);
}

int print2Darray(int a[][6], int m, int n) {
  for (int i = 0; i < m; i++) {
    for (int j = 0; j < n; j++)
      printf("%d,", a[i][j]);

    putchar('\n');
   }
}
```

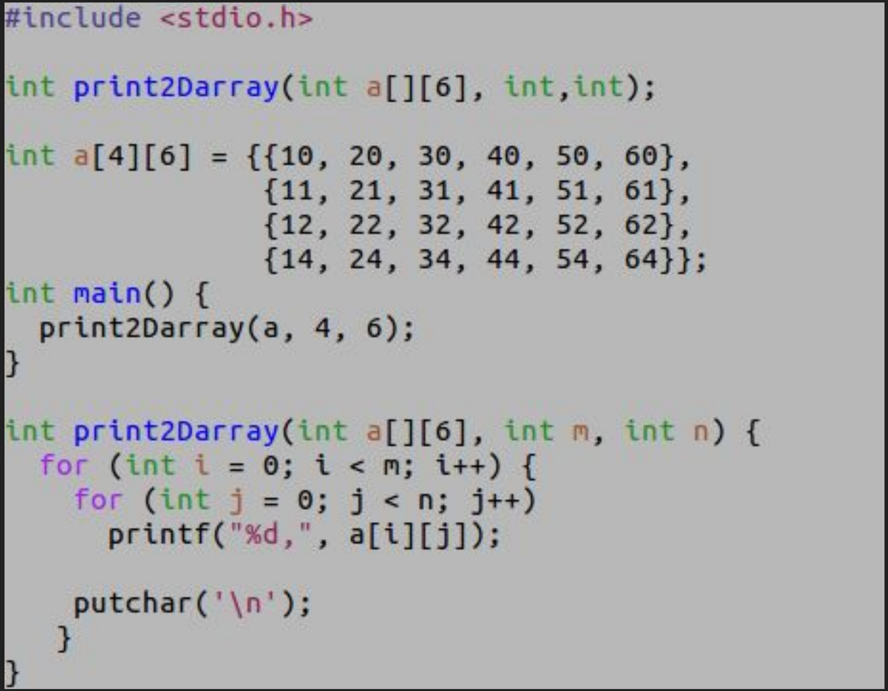
Example: Print a 2D Array



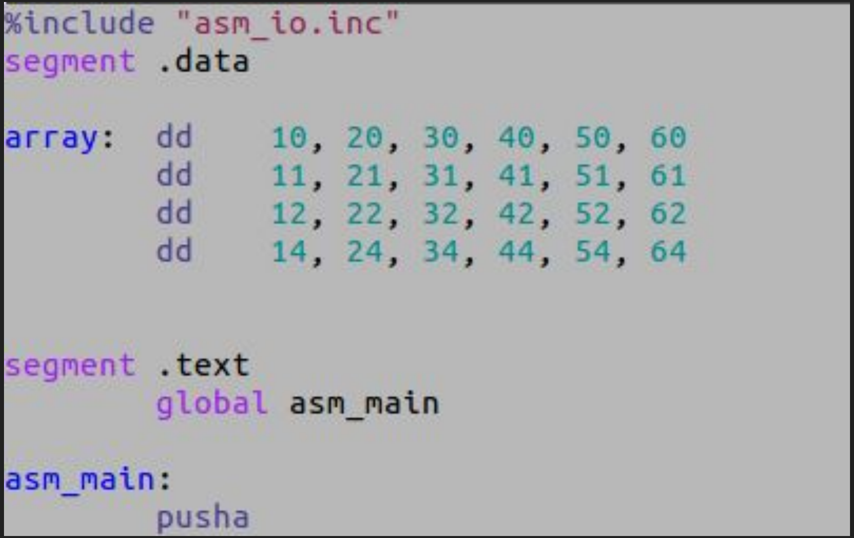
```c
#include <stdio.h>

int print2Darray(int a[][6], int,int);

int a[4][6] = {{10, 20, 30, 40, 50, 60},
               {11, 21, 31, 41, 51, 61},
               {12, 22, 32, 42, 52, 62},
               {14, 24, 34, 44, 54, 64}};
int main() {
  print2Darray(a, 4, 6);
}

int print2Darray(int a[][6], int m, int n) {
  for (int i = 0; i < m; i++) {
    for (int j = 0; j < n; j++)
      printf("%d,", a[i][j]);

    putchar('\n');
   }
}
```

```
nasihatkon@kntu:code$ gcc print2DArray.c && ./a.out
10,20,30,40,50,60,
11,21,31,41,51,61,
12,22,32,42,52,62,
14,24,34,44,54,64,
```

Example: Print a 2D Array

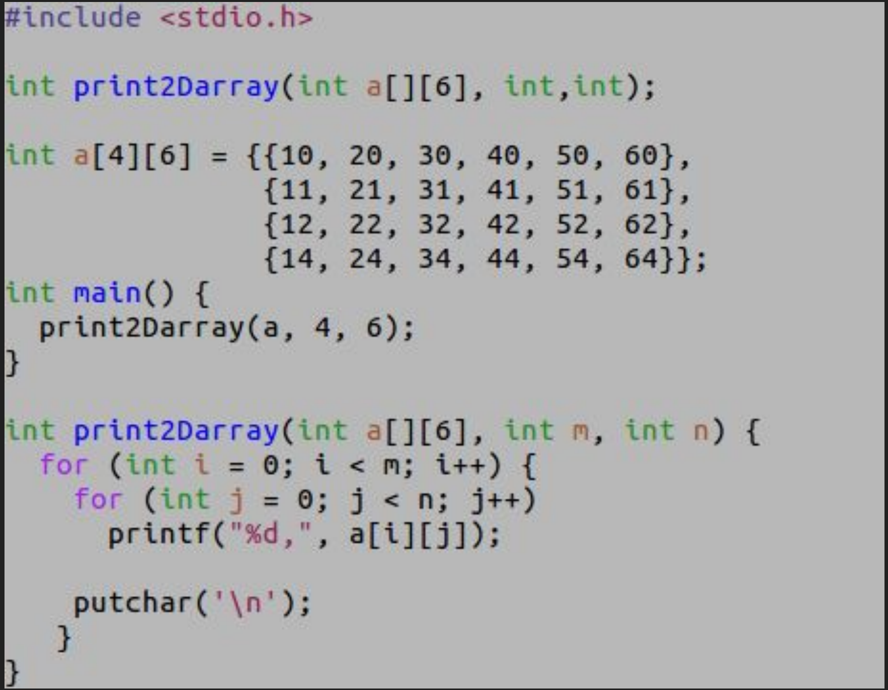


```c
#include <stdio.h>

int print2Darray(int a[][6], int,int);

int a[4][6] = {{10, 20, 30, 40, 50, 60},
               {11, 21, 31, 41, 51, 61},
               {12, 22, 32, 42, 52, 62},
               {14, 24, 34, 44, 54, 64}};
int main() {
  print2Darray(a, 4, 6);
}

int print2Darray(int a[][6], int m, int n) {
  for (int i = 0; i < m; i++) {
    for (int j = 0; j < n; j++)
      printf("%d,", a[i][j]);

    putchar('\n');
   }
}
```



```nasm
%include "asm_io.inc"
segment .data

array:  dd     10, 20, 30, 40, 50, 60
        dd     11, 21, 31, 41, 51, 61
        dd     12, 22, 32, 42, 52, 62
        dd     14, 24, 34, 44, 54, 64


segment .text
        global asm_main

asm_main:
        pusha
```

Example: Print a 2D Array

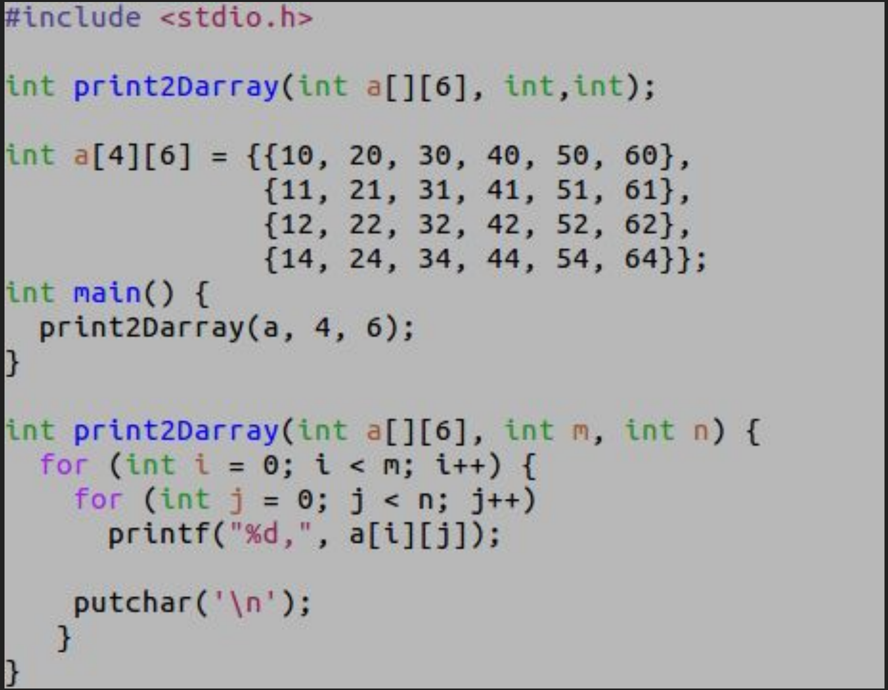


```c
#include <stdio.h>

int print2Darray(int a[][6], int,int);

int a[4][6] = {{10, 20, 30, 40, 50, 60},
               {11, 21, 31, 41, 51, 61},
               {12, 22, 32, 42, 52, 62},
               {14, 24, 34, 44, 54, 64}};
int main() {
  print2Darray(a, 4, 6);
}

int print2Darray(int a[][6], int m, int n) {
  for (int i = 0; i < m; i++) {
    for (int j = 0; j < n; j++)
      printf("%d,", a[i][j]);

    putchar('\n');
   }
}
```

print2DArray1.asm

```
segment .data

array :  dd    10 , 20 , 30 , 40 , 50 , 60
         dd    11 , 21 , 31 , 41 , 51 , 61
         dd    12 , 22 , 32 , 42 , 52 , 62
         dd    14 , 24 , 34 , 44 , 54 , 64

segment .text
         ⋮
    ; print2DArray (array, m, n)
    push 6          ; no of columns
    push 4          ; no of rows
    push array      ; address of array
    call print2DArray
    add  esp , 12
```

Example: Print a 2D Array



```c
#include <stdio.h>

int print2Darray(int a[][6], int,int);

int a[4][6] = {{10, 20, 30, 40, 50, 60},
               {11, 21, 31, 41, 51, 61},
               {12, 22, 32, 42, 52, 62},
               {14, 24, 34, 44, 54, 64}};
int main() {
  print2Darray(a, 4, 6);
}

int print2Darray(int a[][6], int m, int n) {
  for (int i = 0; i < m; i++) {
    for (int j = 0; j < n; j++)
      printf("%d,", a[i][j]);

    putchar('\n');
   }
}
```

print2DArray3.asm

```
segment .data

array :  dd   10 , 20 , 30 , 40 , 50 , 60
         dd   11 , 21 , 31 , 41 , 51 , 61
         dd   12 , 22 , 32 , 42 , 52 , 62
         dd   14 , 24 , 34 , 44 , 54 , 64

segment .text
            ⋮
    ; print2DArray (array, m, n)
    push 6          ; no of columns
    push 4          ; no of rows
    push array      ; address of array
    call print2DArray
    add  esp , 12
```

Example: Print a 2D Array

```c
int print2Darray(int a[][6], int m, int n) {
  for (int i = 0; i < m; i++) {
   for (int j = 0; j < n; j++)
     printf("%d,", a[i][j]);

   putchar('\n');
  }
}
```

```nasm
delim: db ",  ", 0

%define ARRAY  [ebp+8]
%define M      [ebp+12]
%define N      [ebp+16]

; print2DArray(ARRAY, M, N)
print2DArray:
        push ebp
        mov  ebp, esp

        mov ebx, ARRAY
        mov esi, 0
loop1:
        cmp esi, M
        jge endloop1
```

```nasm
        inc esi
        jmp loop1
endloop1:
        mov esp, ebp
        pop ebp
        ret
```

print2DArray3.asm

Example: Print a 2D Array

```c
int print2Darray(int a[][6], int m, int n) {
  for (int i = 0; i < m; i++) {
   for (int j = 0; j < n; j++)
     printf("%d,", a[i][j]);

   putchar('\n');
  }
}
```

```nasm
delim: db  ",   ", 0

%define ARRAY  [ebp+8]
%define M      [ebp+12]
%define N      [ebp+16]

; print2DArray(ARRAY, M, N)
print2DArray:
        push ebp
        mov  ebp, esp

        mov ebx, ARRAY
        mov esi, 0
loop1:
        cmp esi, M
        jge endloop1

        mov edi, 0
loop2:
        cmp edi, N
        jge endloop2
```

```nasm









        inc edi
        jmp loop2
endloop2:




        inc esi
        jmp loop1
endloop1:
        mov esp, ebp
        pop ebp
        ret
```

print2DArray3.asm

Example: Print a 2D Array

```c
int print2Darray(int a[][6], int m, int n) {
  for (int i = 0; i < m; i++) {
   for (int j = 0; j < n; j++)
     printf("%d,", a[i][j]);

   putchar('\n');
  }
}
```

```nasm
delim: db  ",   ", 0

%define ARRAY  [ebp+8]
%define M      [ebp+12]
%define N      [ebp+16]

; print2DArray(ARRAY, M, N)
print2DArray:
        push ebp
        mov  ebp, esp

        mov ebx, ARRAY
        mov esi, 0
loop1:
        cmp esi, M
        jge endloop1

        mov edi, 0
loop2:
        cmp edi, N
        jge endloop2
```

```nasm
        inc edi
        jmp loop2
endloop2:
        mov al, 10
        call print_char

        inc esi
        jmp loop1
endloop1:
        mov esp, ebp
        pop ebp
        ret
```

print2DArray3.asm

Example: Print a 2D Array

```c
int print2Darray(int a[][6], int m, int n) {
  for (int i = 0; i < m; i++) {
   for (int j = 0; j < n; j++)
     printf("%d,", a[i][j]);

   putchar('\n');
  }
}
```

```nasm
delim: db ", ", 0

%define ARRAY  [ebp+8]
%define M      [ebp+12]
%define N      [ebp+16]

; print2DArray(ARRAY, M, N)
print2DArray:
        push ebp
        mov  ebp, esp

        mov ebx, ARRAY
        mov esi, 0
loop1:
        cmp esi, M
        jge endloop1

        mov edi, 0
loop2:
        cmp edi, N
        jge endloop2
              ⋮
```

```nasm
        ; index = esi*N+edi
        mov eax, N
        mul esi
        add eax, edi




        inc edi
        jmp loop2
endloop2:
        mov al, 10
        call print_char

        inc esi
        jmp loop1
endloop1:
        mov esp, ebp
        pop ebp
        ret
```

print2DArray3.asm

Example: Print a 2D Array

```c
int print2Darray(int a[][6], int m, int n) {
  for (int i = 0; i < m; i++) {
   for (int j = 0; j < n; j++)
     printf("%d,", a[i][j]);

   putchar('\n');
  }
}
```

```nasm
delim: db  ",   ", 0

%define ARRAY  [ebp+8]
%define M      [ebp+12]
%define N      [ebp+16]

; print2DArray(ARRAY, M, N)
print2DArray:
        push ebp
        mov  ebp, esp

        mov ebx, ARRAY
        mov esi, 0
loop1:
        cmp esi, M
        jge endloop1

        mov edi, 0
loop2:
        cmp edi, N
        jge endloop2
              ⋮
```

```nasm
        ; index = esi*N+edi
        mov eax, N
        mul esi
        add eax, edi

        mov eax, [ebx+4*eax]
        call print_int
        mov eax, delim
        call print_string

        inc edi
        jmp loop2
endloop2:
        mov al, 10
        call print_char

        inc esi
        jmp loop1
endloop1:
        mov esp, ebp
        pop ebp
        ret
```

print2DArray3.asm

Example: Print a 2D Array

```c
int print2Darray(int a[][6], int m, int n) {
  for (int i = 0; i < m; i++) {
   for (int j = 0; j < n; j++)
     printf("%d,", a[i][j]);

   putchar('\n');
  }
}
```

```nasm
delim: db  ",   ", 0

%define ARRAY  [ebp+8]
%define M      [ebp+12]
%define N      [ebp+16]

; print2DArray(ARRAY, M, N)
print2DArray:
        push ebp
        mov  ebp, esp

        mov ebx, ARRAY
        mov esi, 0
loop1:
        cmp esi, M
        jge endloop1

        mov edi, 0
loop2:
        cmp edi, N
        jge endloop2
                ⋮
```

```nasm
        ; index = esi*N+edi
        mov eax, N
        mul esi
        add eax, edi

        mov eax, [ebx+4*eax]
        call print_int
        mov eax, delim
        call print_string

        inc edi
        jmp loop2
endloop2:
        mov al, 10
        call print_char

        inc esi
        jmp loop1
endloop1:
        mov esp, ebp
        pop ebp
        ret
```

print2DArray3.asm

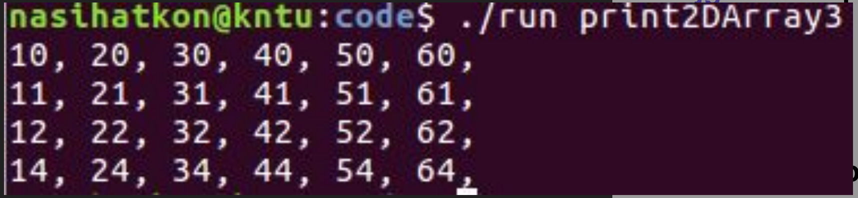
Example: Print a 2D Array

```c
int print2Darray(int a[][6], int m, int n) {
  for (int i = 0; i < m; i++) {
   for (int j = 0; j < n; j++)
     printf("%d,", a[i][j]);

   putchar('\n');
  }
}
```

```nasm
delim: db ",  ", 0

%define ARRAY  [ebp+8]
%define M      [ebp+12]
%define N      [ebp+16]

; print2DArray(ARRAY, M, N)
print2DArray:
        push ebp
        mov  ebp, esp

        mov ebx, ARRAY
        mov esi, 0
loop1:
        cmp esi, M
        jge endloop1
                    ⋮
```

```nasm
      ; index = esi*N+edi
        mov eax, N
        mul esi
        add eax, edi

        mov eax, [ebx+4*eax]
        call print_int
        mov eax, delim
        call print_string

        inc edi
        jmp loop2
endloop2:
        mov al, 10
        call print_char

        inc esi
        jmp loop1
endloop1:
        mov esp, ebp
        pop ebp
        ret
```

print2DArray3.asm



```
nasihatkon@kntu:code$ ./run print2DArray3
10, 20, 30, 40, 50, 60,
11, 21, 31, 41, 51, 61,
12, 22, 32, 42, 52, 62,
14, 24, 34, 44, 54, 64,
```

Example: Print a 2D Array

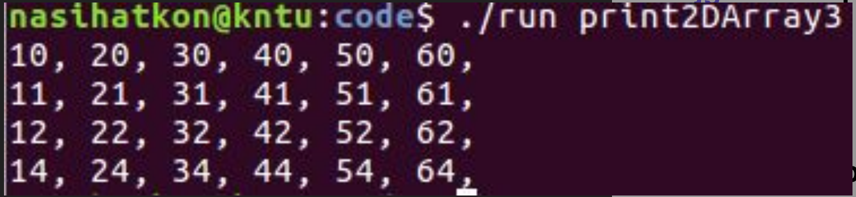
```c
int print2Darray(int a[][6], int m, int n) {
  for (int i = 0; i < m; i++) {
    for (int j = 0; j < n; j++)
      printf("%d,", a[i][j]);

    putchar('\n');
  }
}
```

```nasm
delim: db  ",   ", 0

%define ARRAY  [ebp+8]
%define M      [ebp+12]
%define N      [ebp+16]

; print2DArray(ARRAY, M, N)

        mov esi, 0
loop1:
        cmp esi, M
        jge endloop1

                                  o2
                  ⋮
```

```nasm
     ; index = esi*N+edi
        mov eax, N
        mul esi
        add eax, edi

        mov eax, [ebx+4*eax]
        call print_int
         ov eax, delim
         all print_string

         nc edi
        jmp loop2
endloop2:
        mov al, 10
        call print_char

        inc esi
        jmp loop1
endloop1:
        mov esp, ebp
        pop ebp
        ret
```

print2DArray3.asm

Make it faster?



```
nasihatkon@kntu:code$ ./run print2DArray3
10, 20, 30, 40, 50, 60,
11, 21, 31, 41, 51, 61,
12, 22, 32, 42, 52, 62,
14, 24, 34, 44, 54, 64,
```

Example: Print a 2D Array

print2DArray3.asm

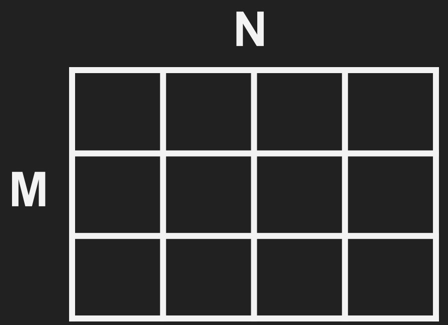
```
        mov ebx, ARRAY
        mov esi, 0
loop1:
        cmp esi, M
        jge endloop1

        mov edi, 0
loop2:
        cmp edi, N
        jge endloop2

      ; index = esi*N+edi
        mov eax, N
        mul esi
        add eax, edi

        mov eax, [ebx+4*eax]
        call print_int
        mov eax, delim
        call print_string

                  ⋮
```

print2DArray4.asm

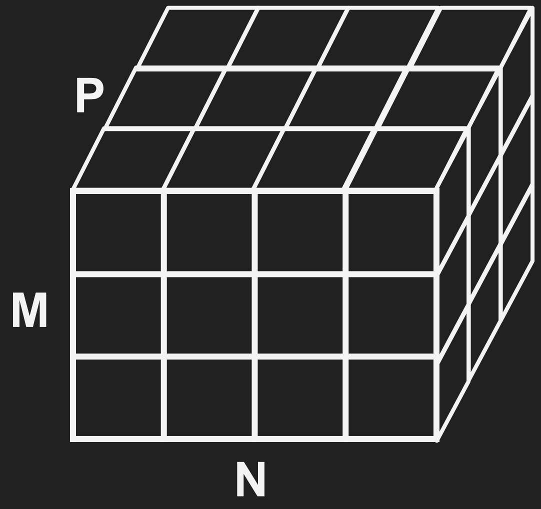
```
        mov ebx, ARRAY
        mov esi, 0
loop1:
        cmp esi, M
        jge endloop1

        mov edi, 0
loop2:
        cmp edi, N
        jge endloop2


        mov eax, [ebx]
        call print_int
        mov eax, delim
        call print_string


        add ebx, 4

              ⋮
```

3D Arrays

- 2D array
  - M ❌ N

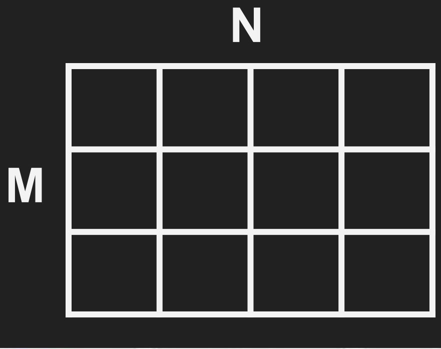


- 3D array
  - M ❌ N ❌ P

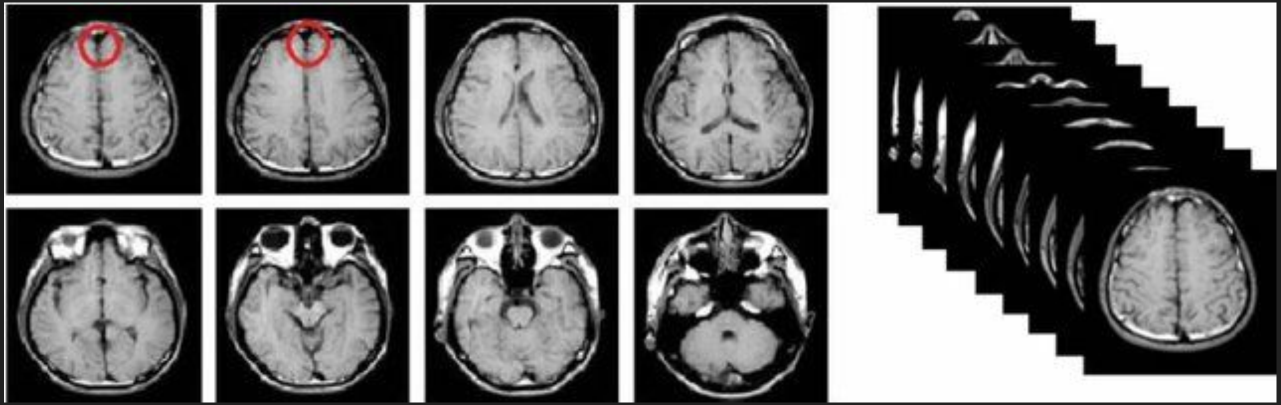


3D Arrays

- 2D array
  - M ❌ N



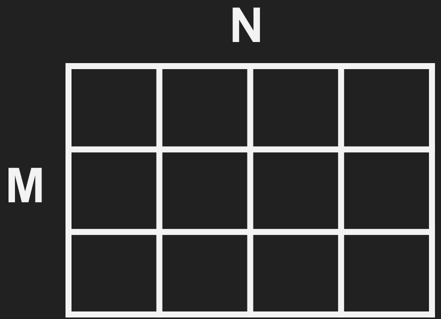
- 3D array
  - M ❌ N ❌ P



Cui, Lu-Bin, et al. "An eigenvalue problem for even order tensors with its applications."

3D Arrays

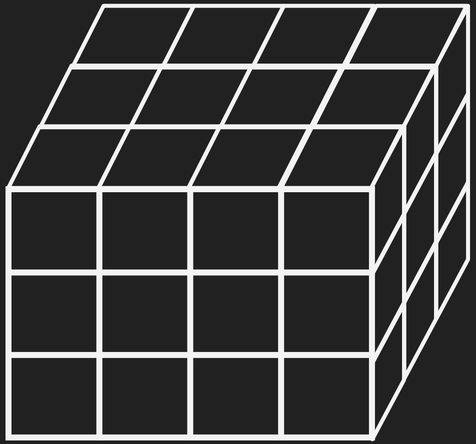
- 2D array
  - M ❌ N



- 3D array
  - M ❌ N ❌ P

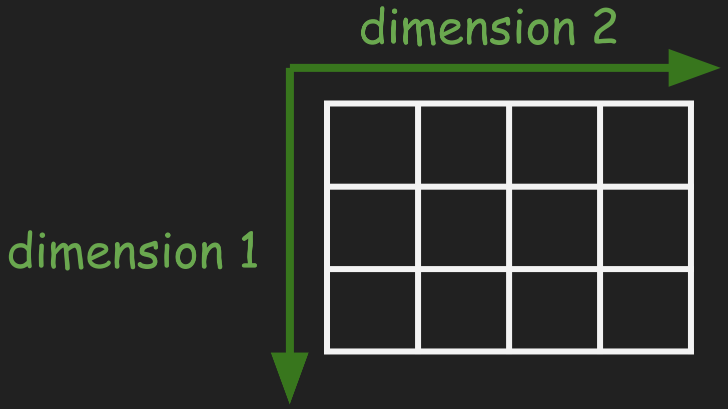
ND Arrays: Row-major vs. Column-major

- What does row-major and column-major mean?
  - Matlab vs Numpy ND-arrays



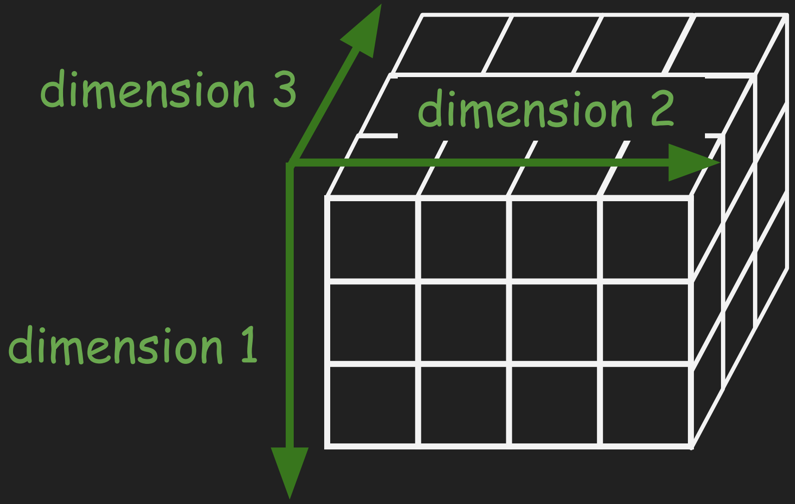
ND Arrays: Row-major vs. Column-major

- What does row-major and column-major mean?
  - Matlab vs Numpy ND-arrays



ND Arrays: Row-major vs. Column-major

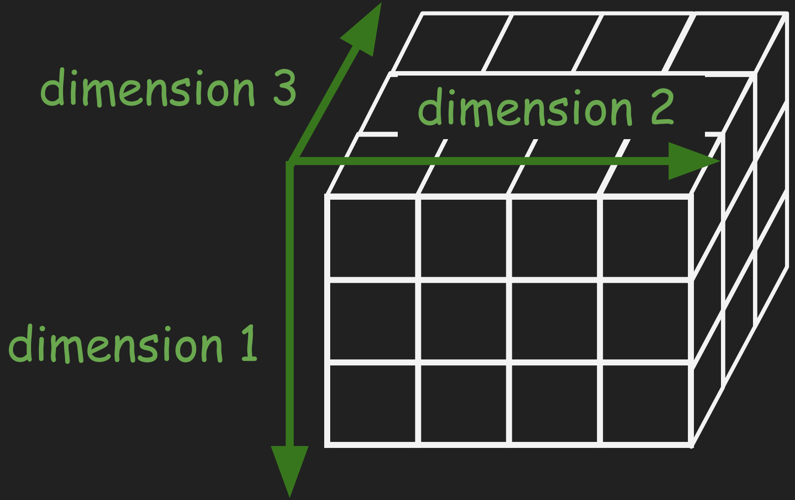
- What does row-major and column-major mean?
    - Matlab vs Numpy ND-arrays



ND Arrays: Row-major vs. Column-major

- What does row-major and column-major mean?
  - Matlab vs Numpy ND-arrays

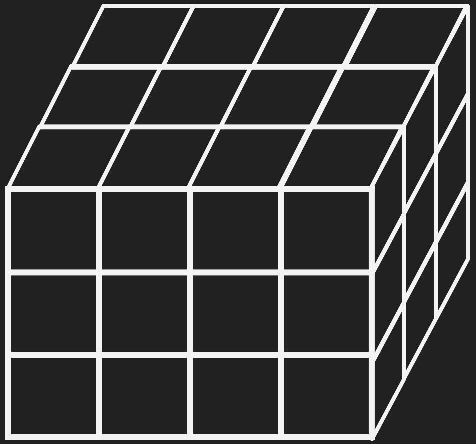
- Row-major: last index moves fastest
- Column-major: first index moves fastest



ND Arrays: Row-major vs. Column-major

- What does row-major and column-major mean?
  - Matlab vs Numpy ND-arrays

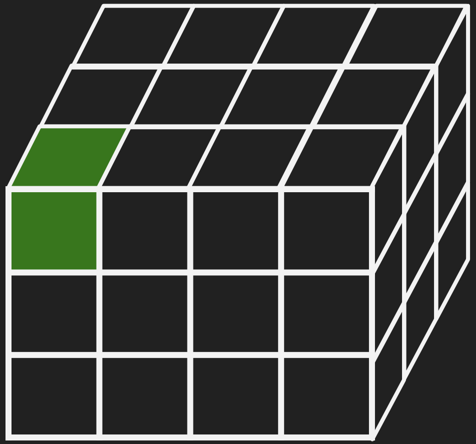
- **Row-major: last index moves fastest**



ND Arrays: Row-major vs. Column-major

- What does row-major and column-major mean?
  - Matlab vs Numpy ND-arrays

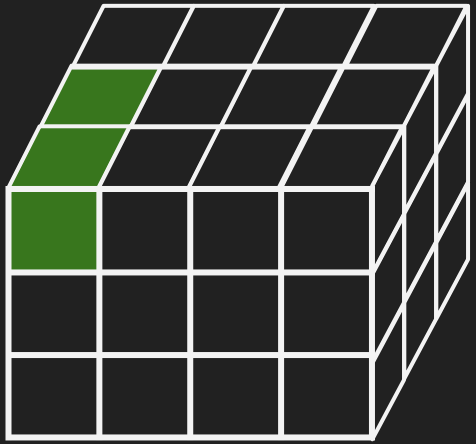
- **Row-major: last index moves fastest**



ND Arrays: Row-major vs. Column-major

- What does row-major and column-major mean?
  - Matlab vs Numpy ND-arrays

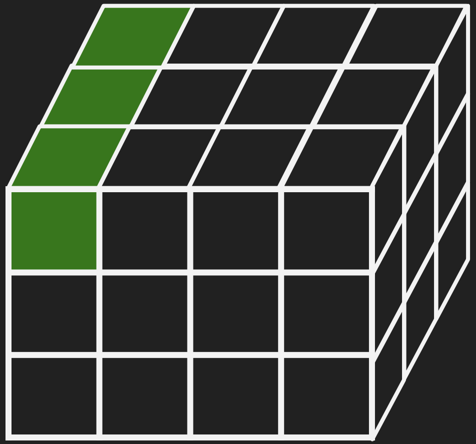
- **Row-major: last index moves fastest**



ND Arrays: Row-major vs. Column-major

- What does row-major and column-major mean?
    - Matlab vs Numpy ND-arrays

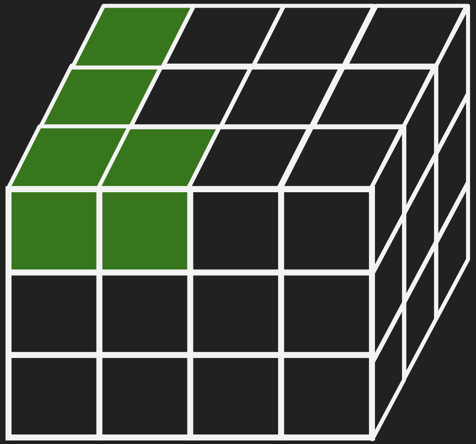
- **Row-major: last index moves fastest**



ND Arrays: Row-major vs. Column-major

- What does row-major and column-major mean?
  - Matlab vs Numpy ND-arrays

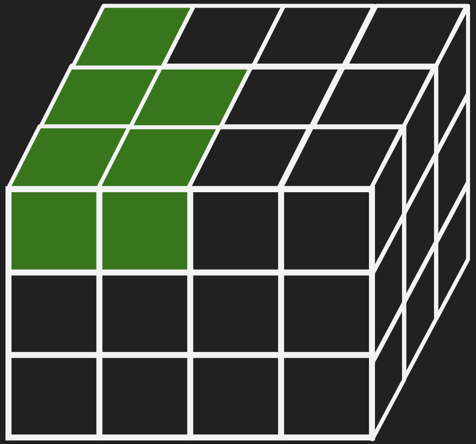
- **Row-major: last index moves fastest**



ND Arrays: Row-major vs. Column-major

- What does row-major and column-major mean?
  - Matlab vs Numpy ND-arrays

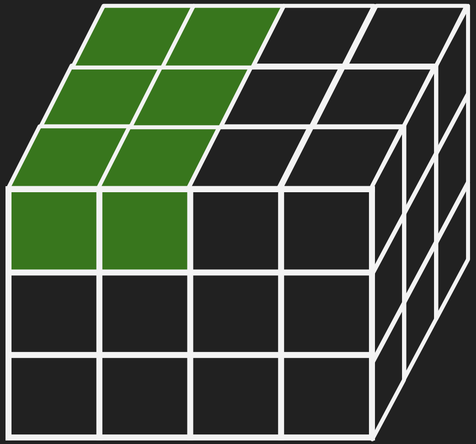
- **Row-major: last index moves fastest**



ND Arrays: Row-major vs. Column-major

- What does row-major and column-major mean?
  - Matlab vs Numpy ND-arrays

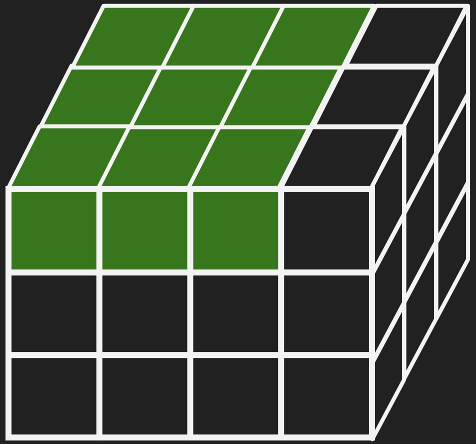
- **Row-major: last index moves fastest**



ND Arrays: Row-major vs. Column-major

- What does row-major and column-major mean?
  - Matlab vs Numpy ND-arrays

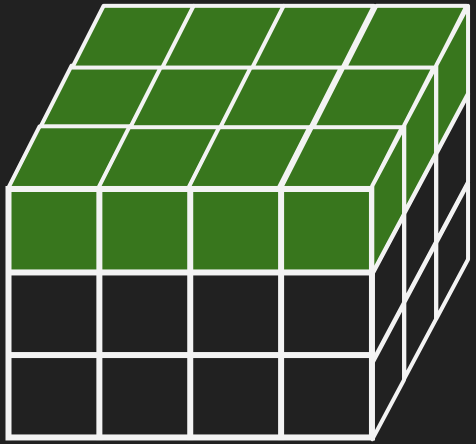
- **Row-major: last index moves fastest**



ND Arrays: Row-major vs. Column-major

- What does row-major and column-major mean?
  - Matlab vs Numpy ND-arrays

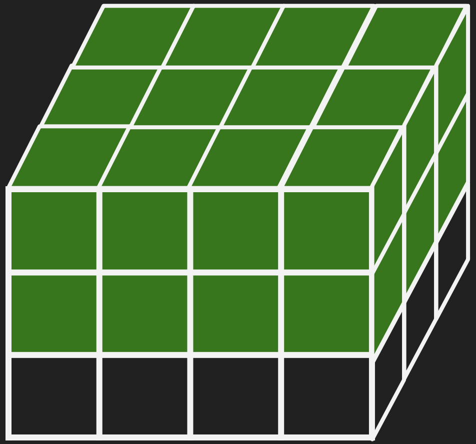
- **Row-major: last index moves fastest**



ND Arrays: Row-major vs. Column-major

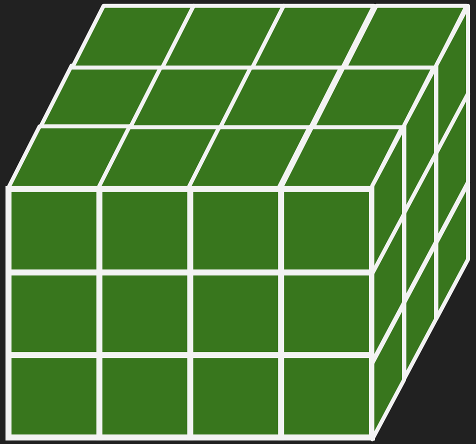
- What does row-major and column-major mean?
    - Matlab vs Numpy ND-arrays

- **Row-major: last index moves fastest**



ND Arrays: Row-major vs. Column-major

- What does row-major and column-major mean?
  - Matlab vs Numpy ND-arrays

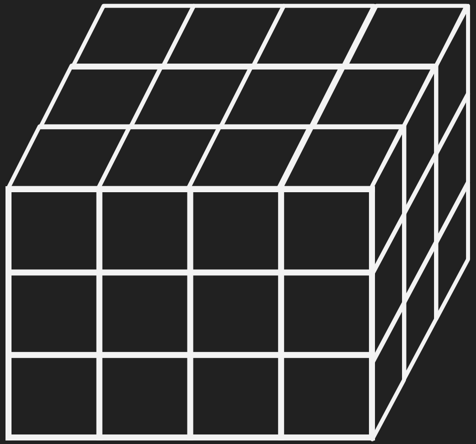


- **Row-major: last index moves fastest**

ND Arrays: Row-major vs. Column-major

- What does row-major and column-major mean?
  - Matlab vs Numpy ND-arrays

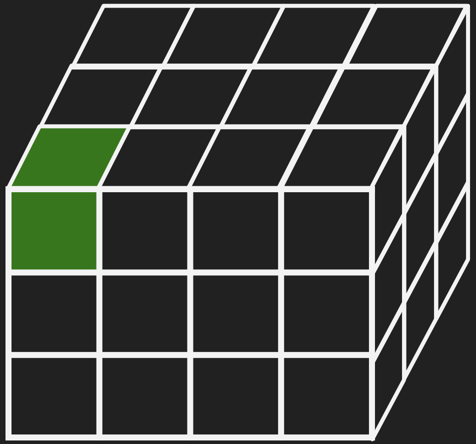
- **Column-major: first index moves fastest**



ND Arrays: Row-major vs. Column-major

- What does row-major and column-major mean?
  - Matlab vs Numpy ND-arrays

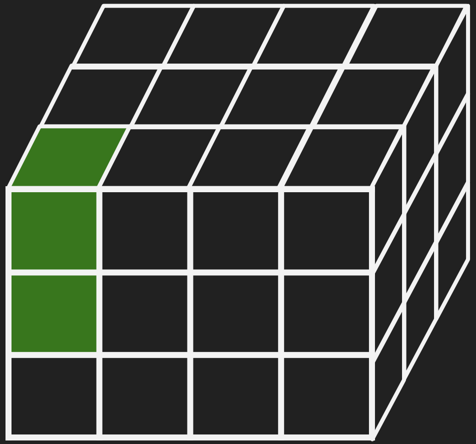
- **Column-major: first index moves fastest**



ND Arrays: Row-major vs. Column-major

- What does row-major and column-major mean?
  - Matlab vs Numpy ND-arrays

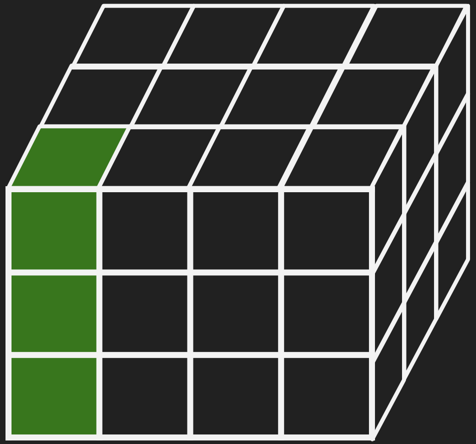
- **Column-major: first index moves fastest**



ND Arrays: Row-major vs. Column-major

- What does row-major and column-major mean?
    - Matlab vs Numpy ND-arrays

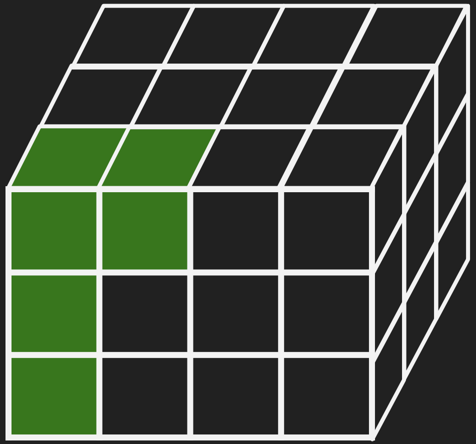
- **Column-major: first index moves fastest**



ND Arrays: Row-major vs. Column-major

- What does row-major and column-major mean?
    - Matlab vs Numpy ND-arrays

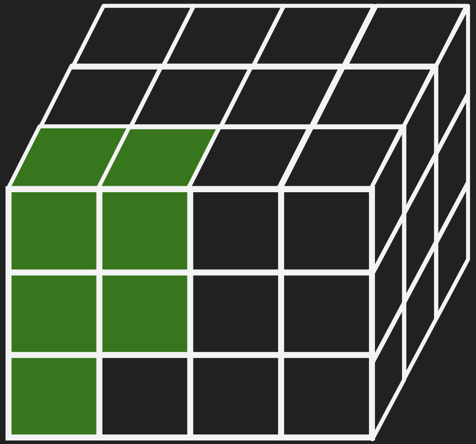
- **Column-major: first index moves fastest**



ND Arrays: Row-major vs. Column-major

- What does row-major and column-major mean?
  - Matlab vs Numpy ND-arrays

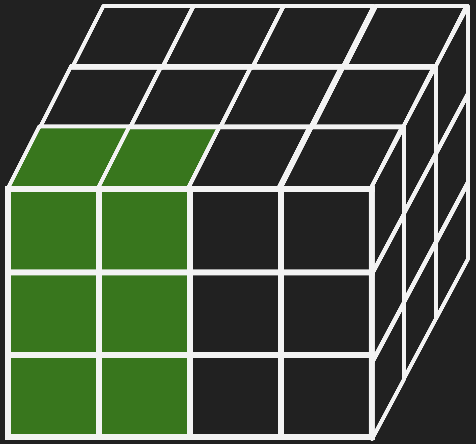
- **Column-major: first index moves fastest**



ND Arrays: Row-major vs. Column-major

- What does row-major and column-major mean?
    - Matlab vs Numpy ND-arrays

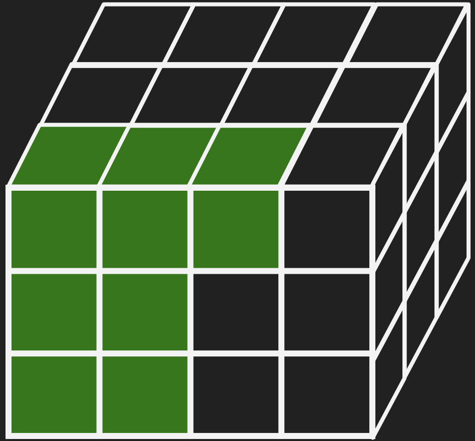
- **Column-major: first index moves fastest**



ND Arrays: Row-major vs. Column-major

- What does row-major and column-major mean?
  - Matlab vs Numpy ND-arrays

- **Column-major: first index moves fastest**

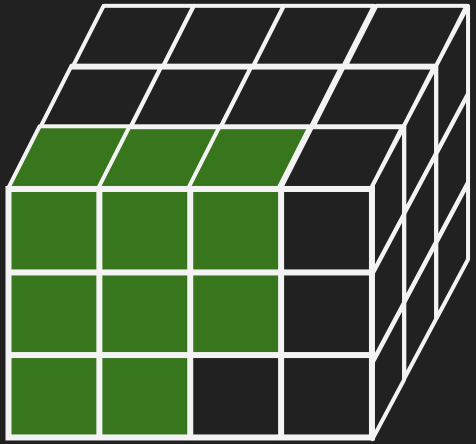


ND Arrays: Row-major vs. Column-major

K. N. Toosi
University of Technology

- What does row-major and column-major mean?
  - Matlab vs Numpy ND-arrays

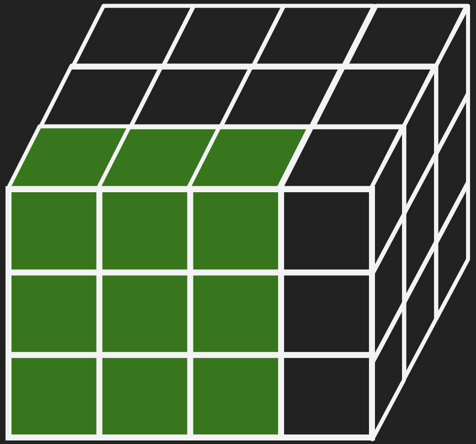
- **Column-major: first index moves fastest**



ND Arrays: Row-major vs. Column-major

- What does row-major and column-major mean?
  - Matlab vs Numpy ND-arrays

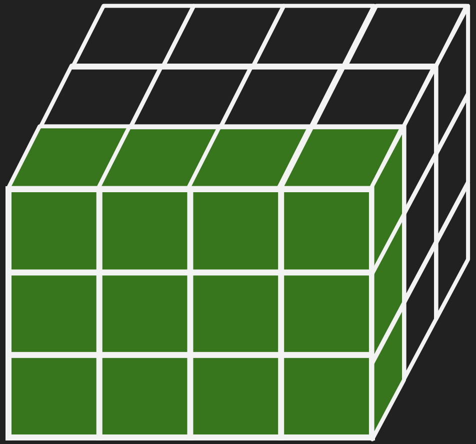
- **Column-major: first index moves fastest**



ND Arrays: Row-major vs. Column-major

- What does row-major and column-major mean?
    - Matlab vs Numpy ND-arrays

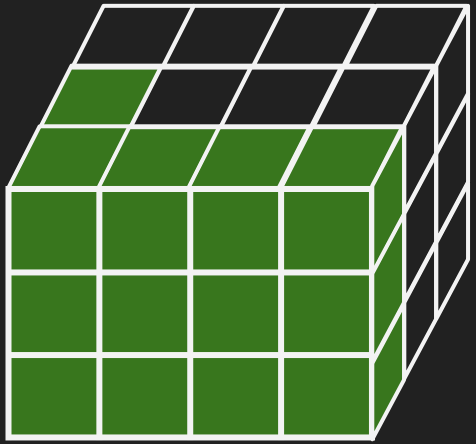
- **Column-major: first index moves fastest**



ND Arrays: Row-major vs. Column-major

- What does row-major and column-major mean?
    - Matlab vs Numpy ND-arrays

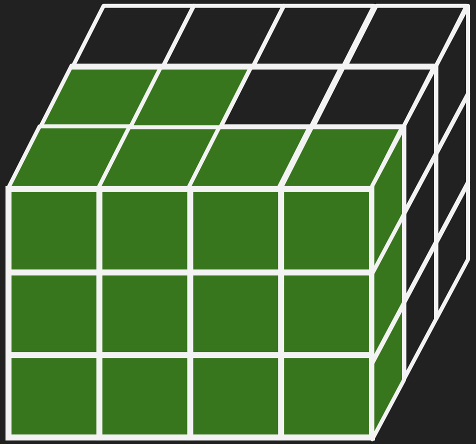
- **Column-major: first index moves fastest**



ND Arrays: Row-major vs. Column-major

- What does row-major and column-major mean?
  - Matlab vs Numpy ND-arrays

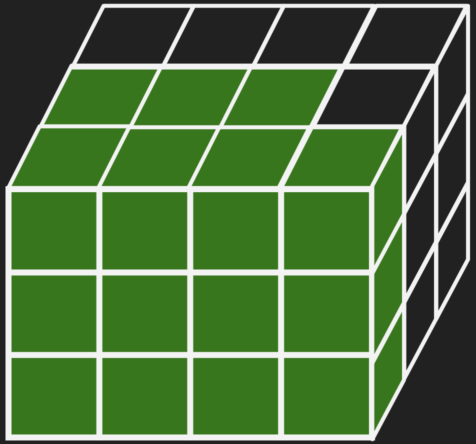
- **Column-major: first index moves fastest**



ND Arrays: Row-major vs. Column-major

- What does row-major and column-major mean?
  - Matlab vs Numpy ND-arrays

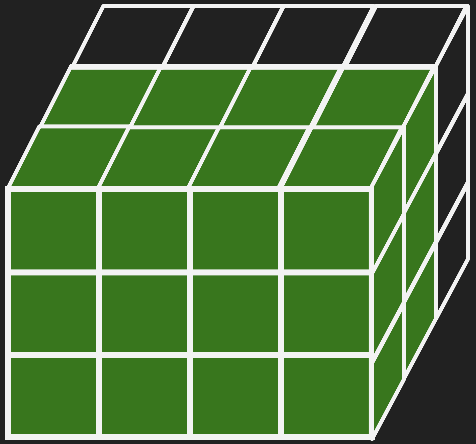
- **Column-major: first index moves fastest**



ND Arrays: Row-major vs. Column-major

- What does row-major and column-major mean?
  - Matlab vs Numpy ND-arrays

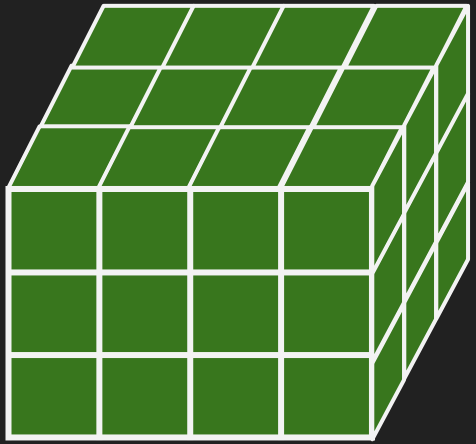
- **Column-major: first index moves fastest**



ND Arrays: Row-major vs. Column-major

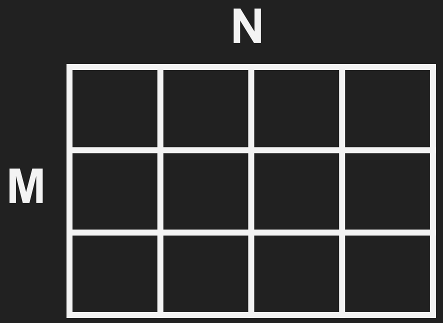
- What does row-major and column-major mean?
    - Matlab vs Numpy ND-arrays

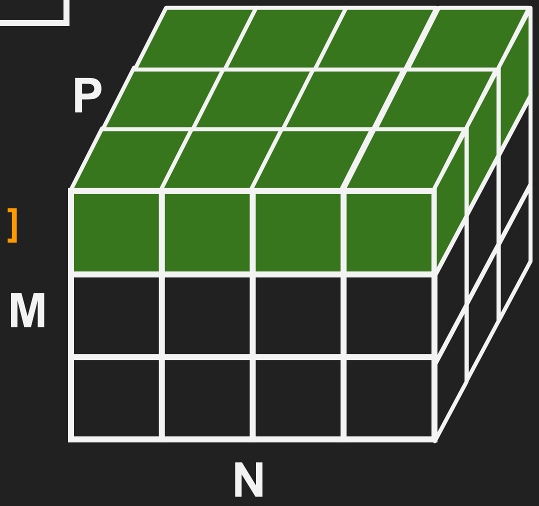
- **Column-major: first index moves fastest**



3D Arrays

- 2D array (row-major)
  - M ❌ N
  - M 1D arrays (rows) of size N
  - `Array[i, j] = Array[N*i + j]`
- 3D array (row-major)
  - M ❌ N ❌ P
  - M 2D arrays of dimension N ❌ P
    - Array[i, j, k] = Array[i, j*P + k] = Array[i *N*P + j * P + k]

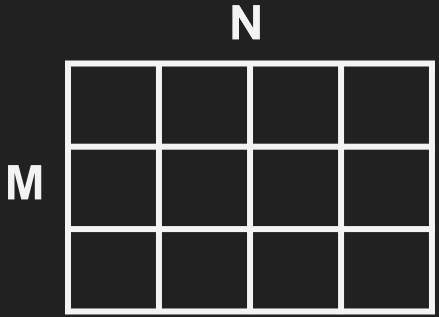


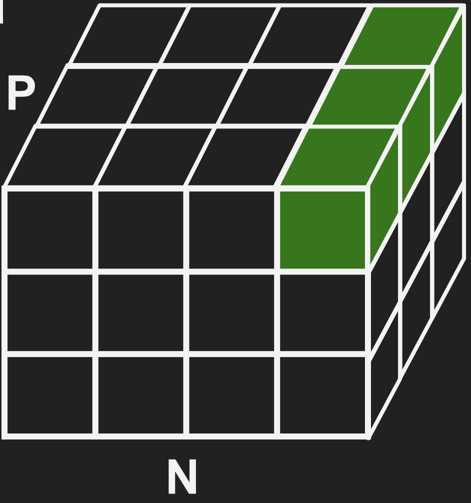


3D Arrays

- 2D array (row-major)
  - M ❌ N
  - M 1D arrays (rows) of size N
  - `Array[i, j] = Array[N*i + j]`
- 3D array (row-major)
  - M ❌ N ❌ P
  - M 2D arrays of dimension N ❌ P
    - ■ Array[i, j, k] = Array[i, j*P + k] = Array[i *N*P + j * P + k]
  - An M ❌ N 2D array of 1D arrays of size P
    - ■ Array[i, j, k] = Array[i, j][k]

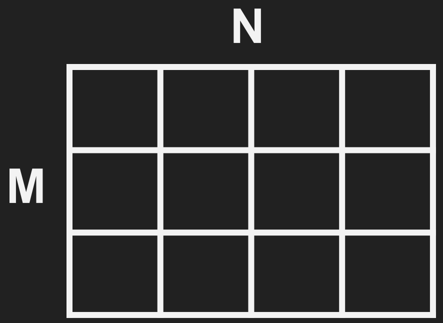
      = Array[i*N + j][k] = Array[(i * N + j) P + k]

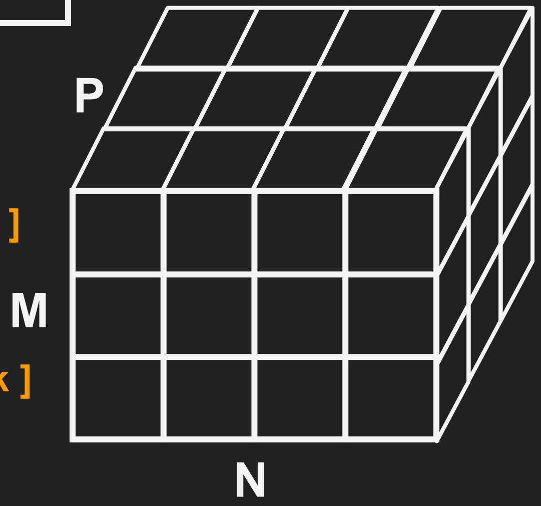




3D Arrays

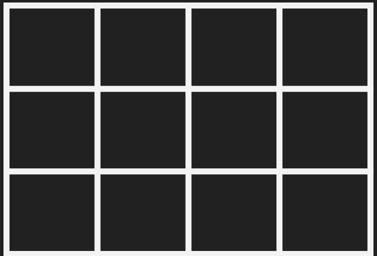
- 2D array (row-major)
  - M ❌ N
  - M 1D arrays (rows) of size N
  - `Array[i, j] = Array[N*i + j]`
- 3D array (row-major)
  - M ❌ N ❌ P
  - M 2D arrays of dimension N ❌ P
    - Array[i, j, k] = Array[i, j*P + k] = Array[i *N*P + j * P + k]
  - An M ❌ N 2D array of 1D arrays of size P
    - Array[i, j, k] = Array[i, j][k]
      = Array[i*N + j][k] = Array[(i * N + j) P + k]



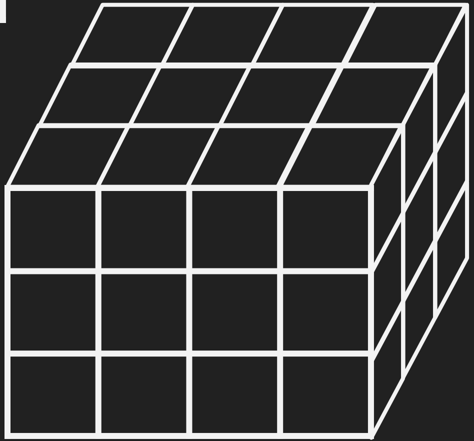


number of + and * operations?

ND Arrays



- 2D array (row-major)
  - M ❌ N
  - M 1D arrays (rows) of size N
  - `Array[i, j] = Array[N*i + j]`
- 3D array (row-major)
  - M ❌ N ❌ P
  - M 2D arrays of dimension N ❌ P
    - Array[i, j, k] = Array[i, j*P + k] = Array[i *N*P + j * P + k]
  - An M ❌ N 2D array of 1D arrays of size P
    - Array[i, j, k] = Array[i*N + j , k] = Array[(i * N + j) P + k]
- ND array (row-major)
  - M_1 ❌ M_2 ❌ … ❌ M_n
  - Array[i_1, i_2, …, i_n] = Array[$M_2 * \dots * M_n * i_1 + M_3 * \dots * M_n * i_2 + \dots + M_n * i_{n-1} + i_n$]



ND Arrays

- ND array (row-major)
  - M_1 ❌ M_2 ❌ … ❌ M_n
  - $\text{Array}[i_1, i_2, \ldots, i_n] = \text{Array}[\ M_2 * \ldots * M_n * i_1 + M_3 * \ldots * M_n * i_2 + \ldots + M_n * i_{n-1} + i_n\ ]$

$$M_2 M_3 M_4 M_5\ i_1 + M_3\ M_4\ M_5\ i_2 + M_4\ \ M_5\ i_3 + M_5\ i_4 + i_5$$

ND Arrays

- ND array (row-major)
    - M_1 ❌ M_2 ❌ … ❌ M_n
    - $\text{Array}[i_1, i_2, \ldots, i_n] = \text{Array}[\ M_2 * \ldots * M_n * i_1 + M_3 * \ldots * M_n * i_2 + \ldots + M_n * i_{n-1} + i_n\ ]$

$$M_2\, M_3\, M_4\, M_5\ i_1 + M_3\, M_4\, M_5\ i_2 + M_4\ M_5\ i_3 + M_5\ i_4 + i_5$$

ND Arrays

- ND array (row-major)
  - M_1 ❌ M_2 ❌ … ❌ M_n
  - $Array[i_1, i_2, \ldots, i_n] = Array[\, M_2 * \ldots * M_n * i_1 + M_3 * \ldots * M_n * i_2 + \ldots + M_n * i_{n-1} + i_n \,]$

$$M_2 M_3 M_4 M_5 \; i_1 + M_3 M_4 M_5 \; i_2 + M_4 \; M_5 \; i_3 + M_5 \; i_4 + i_5$$

$$M_5 \, (\, M_2 M_3 M_4 \; i_1 + M_3 \; M_4 \; i_2 + M_4 \; i_3 + \; i_4 \,) + i_5$$

ND Arrays

- ND array (row-major)
  - ○ M_1 ❌ M_2 ❌ … ❌ M_n
  - ○ $\text{Array}[i_1, i_2, \ldots, i_n] = \text{Array}[\, M_2 * \ldots * M_n * i_1 + M_3 * \ldots * M_n * i_2 + \ldots + M_n * i_{n-1} + i_n \,]$

$$M_2 M_3 M_4 M_5 \; i_1 + M_3 M_4 M_5 \; i_2 + M_4 M_5 \; i_3 + M_5 \; i_4 + i_5$$

$$M_5 \left(M_2 M_3 M_4 \; i_1 + M_3 M_4 \; i_2 + M_4 \; i_3 + i_4 \right) + i_5$$

$$M_5 \left(M_4 \left(M_2 M_3 \; i_1 + M_3 \; i_2 + i_3 \right) + i_4 \right) + i_5$$

ND Arrays

- ND array (row-major)
  - $M_1 \times M_2 \times \dots \times M_n$
  - $\text{Array}[i_1, i_2, \dots, i_n] = \text{Array}[\, M_2 * \dots * M_n * i_1 + M_3 * \dots * M_n * i_2 + \dots + M_n * i_{n-1} + i_n \,]$

$$M_2 M_3 M_4 M_5\ i_1 + M_3 M_4 M_5\ i_2 + M_4\ M_5\ i_3 + M_5\ i_4 + i_5$$

$$M_5 (\ M_2 M_3 M_4\ i_1 + M_3 M_4\ i_2 + M_4\ i_3 + i_4\ ) + i_5$$

$$M_5 (\ M_4 (M_2 M_3\ i_1 + M_3\ i_2 + i_3) + i_4\ ) + i_5$$

$$M_5 (\ M_4 (M_3 (M_2\ i_1 + i_2) + i_3) + i_4\ ) + i_5$$

ND Arrays

$$M_2 M_3 M_4 M_5 \; i_1 + M_3 M_4 M_5 \; i_2 + M_4 \; M_5 \; i_3 + M_5 \; i_4 + i_5$$

$$M_5 (M_4 (M_3 (M_2 \; i_1 + i_2) + i_3) + i_4) + i_5$$

- $j_1 = i_1$
- $j_2 = M_2 \; j_1 + i_2$
- $j_3 = M_3 \; j_2 + i_3$
- $j_4 = M_4 \; j_3 + i_4$
- $j_5 = M_5 \; j_4 + i_5$
- $\text{Array}[i_1, i_2, i_3, i_4, i_5] = \text{Array}[j_5]$

ND Arrays

- ND array (row-major)
  - M_1 ❌ M_2 ❌ … ❌ M_n
  - $Array[i_1, i_2, …, i_n] = Array[\ M_2 * … * M_n * i_1 + M_3 * … * M_n * i_2 + … + M_n * i_{n-1} + i_n\ ]$

- $j_1 = i_1$
- $j_2 = M_2 * j_1 + i_2$
- $j_3 = M_3 * j_2 + i_3$
- $\vdots$
- $j_n = M_n * j_{n-1} + i_n$
- $Array[i_1, i_2, …, i_n] = Array[j_n]$

number of + and * operations?