Which computer language?

- Prototype using Python
- Scientific languages: Fortran, C, C++, Python
- I recommend Python & C++
- C++ because Oop (object-oriented programming)
- Writing large codes are easier in C++. Clear abstraction of tasks!
- Useful features like function overloading, virtual function, classes, template, etc.
- Python programming is fast. Portable to GPUs

- C++: The standard float operations are slower compared to Fortran.
- However, fast libraries are available: blitz++, Armadillo, Boost, Eigen
- OpenACC: std::vector

Python vs. C++

- Python, interpreter language, is *typically* slower than C++.
- However, using C libraries and clever programming can make Python code as fast as C++.
- Coding in Python is fast. That’s why we prototype in Python flrst.
- We could write a parallel code in Python.
- We can call eThcient C, C++, and Fortran codes in Python code.

Compilers

- GNU ("GNU's Not Unix!")—Free software: gcc
- Clang, LLVM
- Intel compilers
- PGI compilers
- Nvidia compiler
- AMD AOCC for Rome processor

MPI&OpenMP

- OpenMP part of language now: gcc, Intel compilers…
- MPI—free ones: OpenMPI, MPICH3
- Other MPI’s: Intel, Cray, …
- Please install MPICH3 & mpi4pi in your laptop/desktop.

Testing

- Test thoroughly!
- Test against exact results (e.g., energy conservation)!
- Test against the limiting cases (viscosity = 0)
- Keep testing frequently.

Versioning

- Management of difterent version of a code.
- Github: free for academic use

Building codes

- A package has 50 to 100 or even larger number of files.
- Use tools to build with dependencies
- CMake, Makefile

Open or closed?

- Opensource or
- Commercial code
- Academic code: often open source.
- Which license? GPL, BSD, ...
- I recommend BSD.

Patience & perseverance

- TARANG as an example
- Developers: MKV, Anando, Manthan, Soumyadeep, Abhishek, Shashwat
- Versions: C++, Python, CUDA
- A general code for fluid, MHD, convection, …

- Make the code better and better..
- Tested thoroughly, organised workshop
- Hunt for hardware: EKA, PARAM YUVA, SHAHEEN
- Contributors: Many students and users
- At present, our code is one of the best spectral codes in the world

Dr. A. P. J. Abdul Kalam Cray HPC award to TARANG

http://mahendra-verma.blogspot.com/search?q=cray

Teamwork

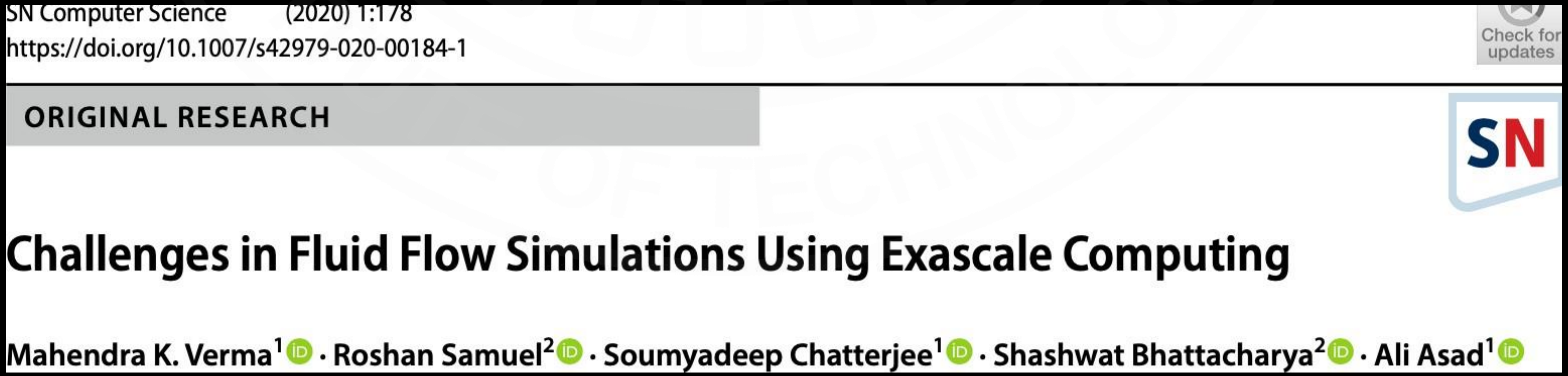
- Core team
- Part-time developers
- Testers
- Users
- Documenters

Advertise!

- Make a good manual!
- Good website!
- Use social media!
- Conduct developer workshop!
- Conduct user workshop!
- Spread word around!

Exascale challenge

- Applications that scale well on exascale systems.
- Well-designed program
- Optimisation
- Great hardware
- Requires good knowledge of hardware, software



SN Computer Science (2020) 1:178
https://doi.org/10.1007/s42979-020-00184-1

Check for updates
SN

ORIGINAL RESEARCH

Challenges in Fluid Flow Simulations Using Exascale Computing

Mahendra K. Verma[1] · Roshan Samuel[2] · Soumyadeep Chatterjee[1] · Shashwat Bhattacharya[2] · Ali Asad[1]

Classes and Inheritance

- Python objects MIT lectures slides link: Objects
- Python classes and inheritance MIT lecture slides link: Classes
- Python modules lectures slides link: Modules

Python Pitfalls

Contrasting C and Python

Python’s Numpy Array

```
import numpy as np

x = np.array([1,2,3])

print(x, id(x))


x = x**2

print(x, id(x))
```

Output:

```
[1 2 3] 13502195596

[1 4 9] 13502195596494449
```

$x = 2 * x$

```
import numpy as np

x = np.array([1,2,3])

print(x, id(x))


x *= x

print(x, id(x))


x = x**2

print(x, id(x))
```

Output:

```
[1 2 3] 139304922347056


[1 4 9] 139304922347056


[ 1 16 81] 139304922347248
```

X x = 2 *x
x *= 2

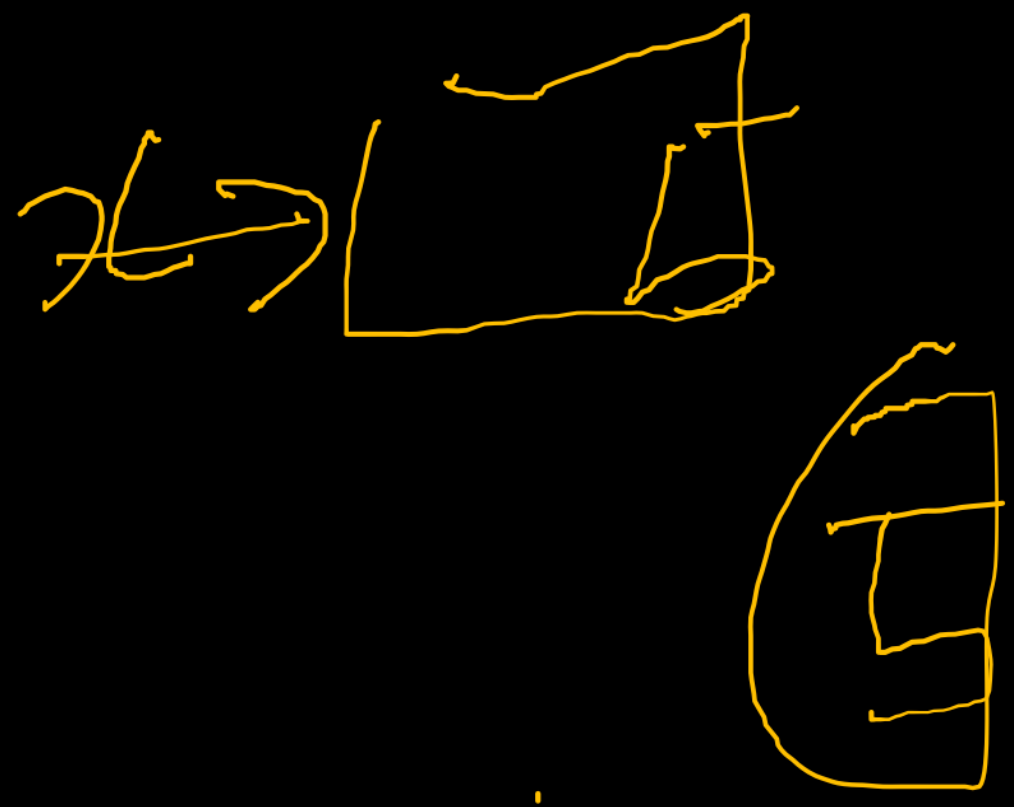
Reference type Variables

Objects

- Variables store the address of an object in memory.
- Objects of C++ are *reference* type variable.
- Python variables are objects.
- Hence, Python variables are of *reference type*.
- On the contrary, integer/float vars of C are of *value type*.

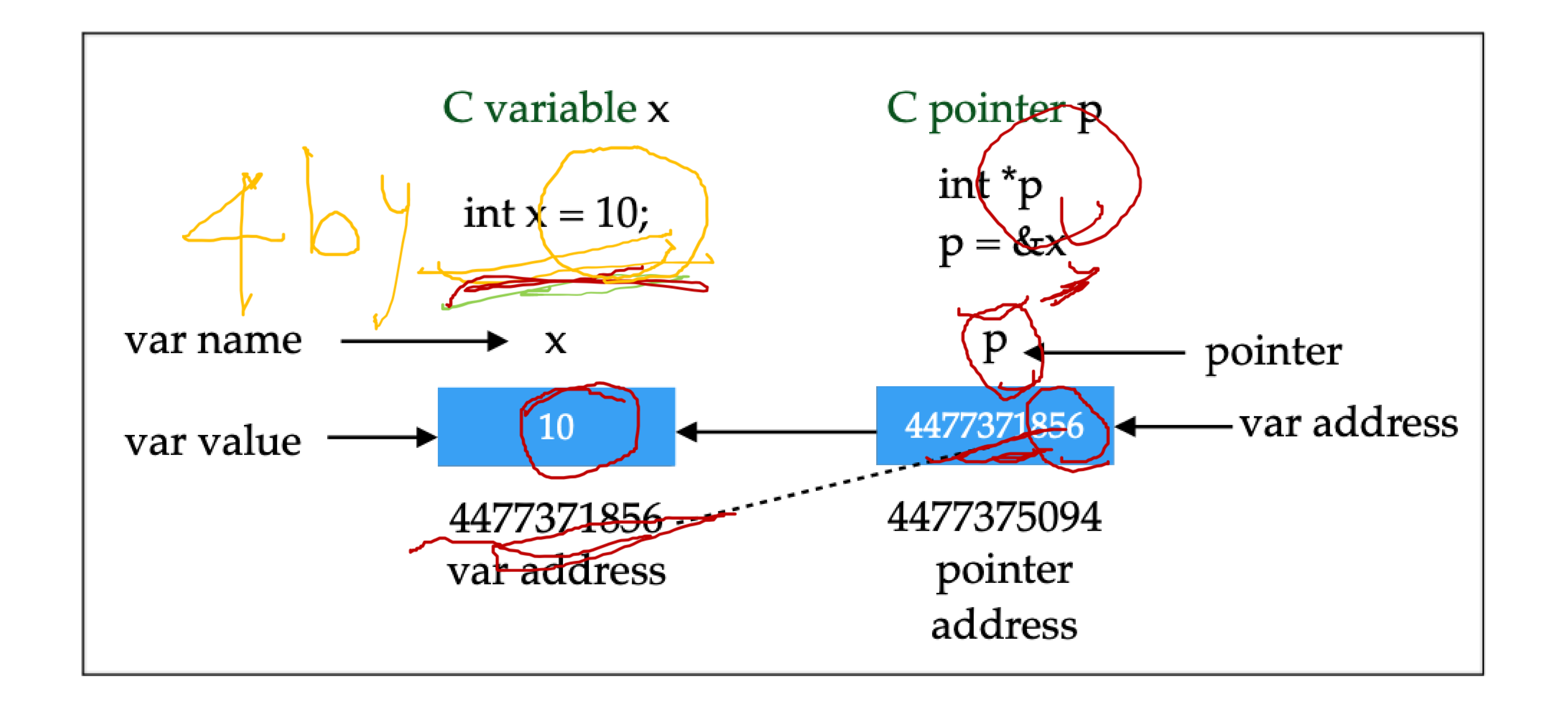
C

int x = 5



Value type Variable

C integer



C variable x

```
int x = 10;
```

C pointer p

```
int *p
p = &x
```

var name → x

p ← pointer

var value → 10 ← 4477371856 ← var address

4477371856 var address

4477375094 pointer address

Python x → 5

Reference type Variable

C++ Object

```
class Student {
    public:
    string name;
    int rollNo;
};
```

```
int main()
{
Student s;
s.name = "Alice";
s.rollNo = 1;

cout << "Address of s using & operator: "
     << &s << endl;

cout << "Address of s using std::address of
     function: " << addressof(s) << endl;
return 0;
```

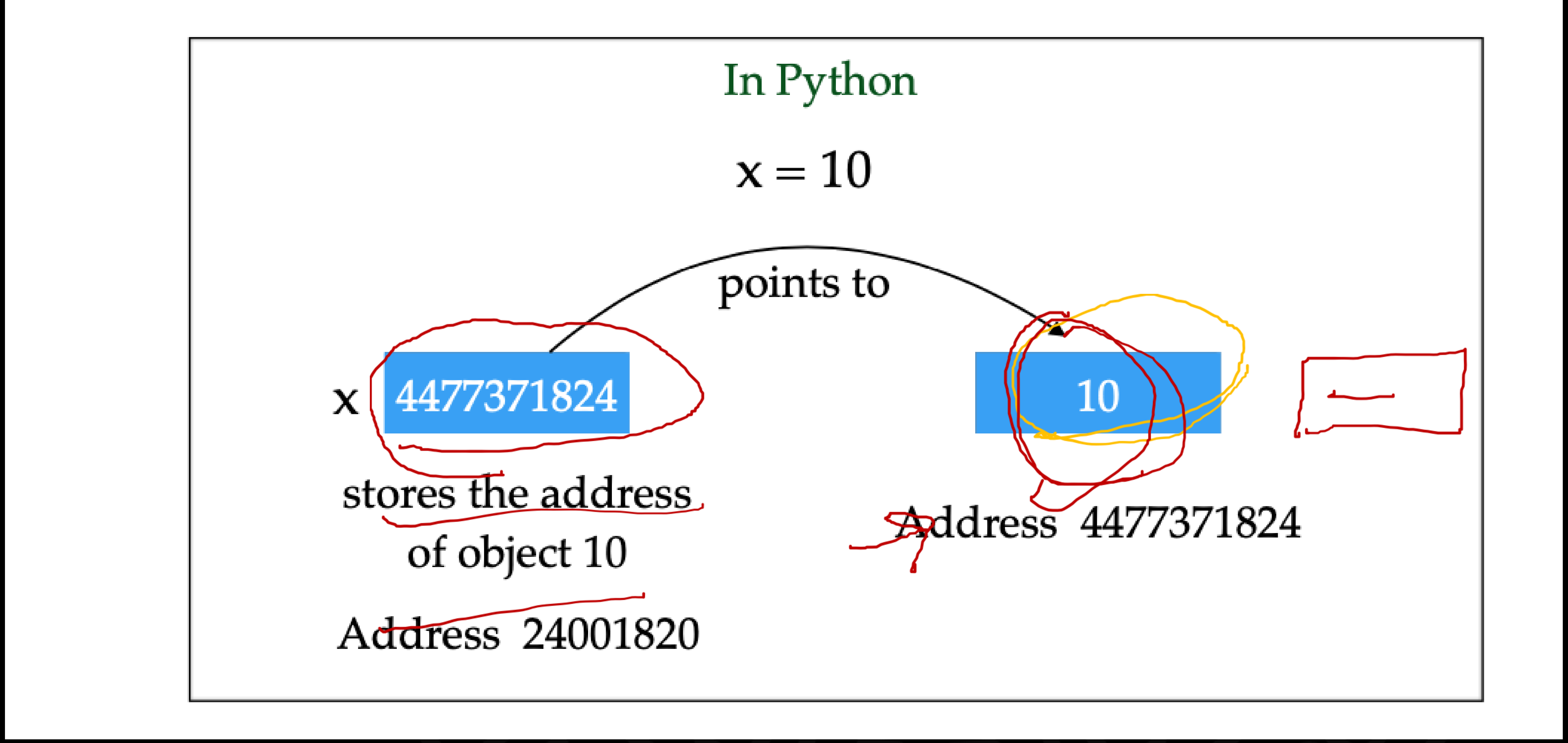
```
Address of s using & operator: 0x7fffe0d944a0
Address of s using std::addressof function: 0x7fffe0d944a0
```

Python Variables are Objects

- Python variables are objects. Hence, they are *vars by reference*.

```
x = 5

print("type of x: ", type(x))



y = "Hello"

print("type of y: ", type(y))
```

```
type of x:  <class ‘int'>



type of y:  <class 'str'>
```



In Python

x = 10

stores the address of object 10

Address 24001820

Address 4477371824

Immutable Objects

5

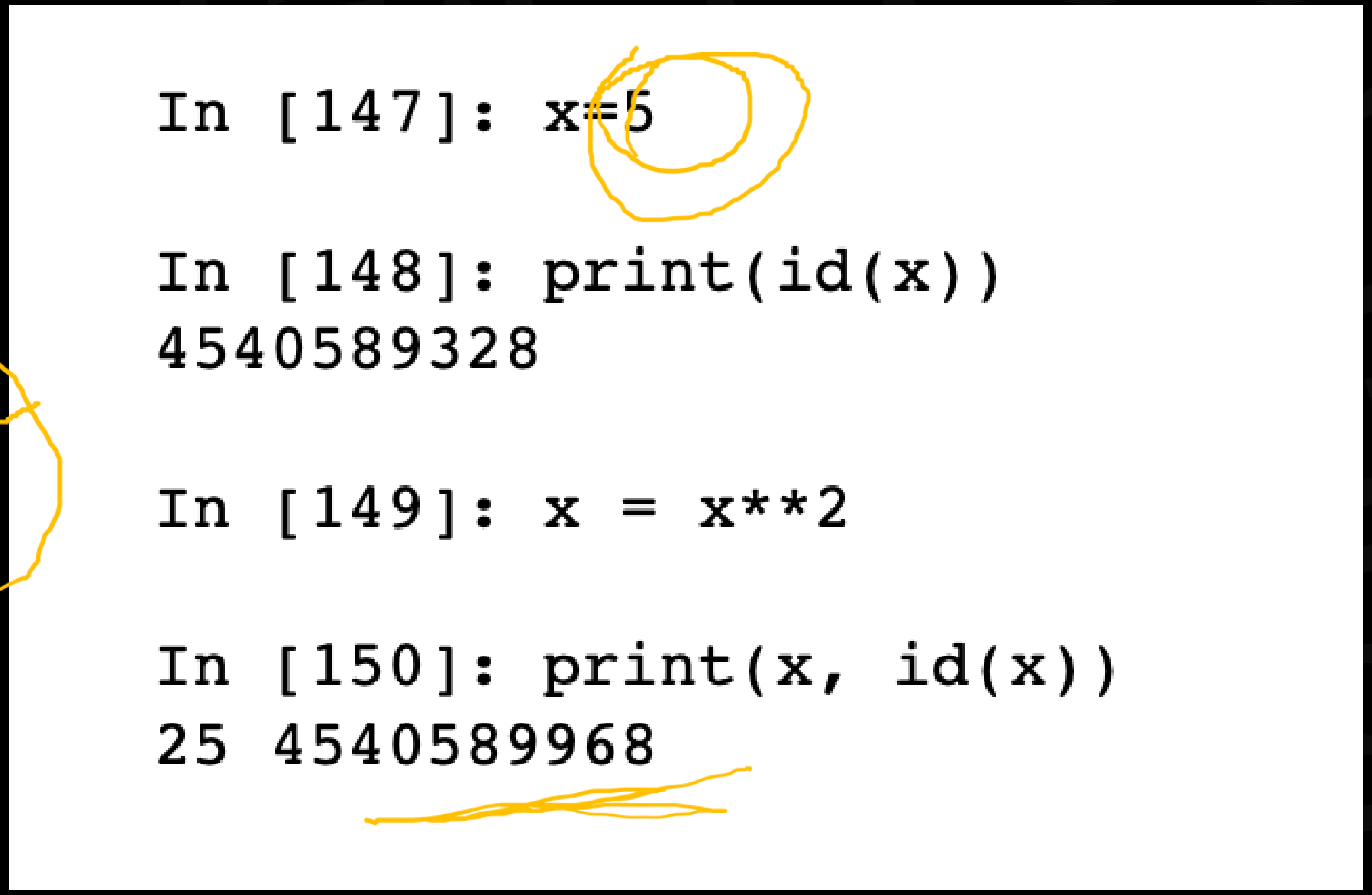
Python data types, integer, float, complex, string, and tuples are immutable.

The value of immutable object is unchangeable once it is created.

$x = 5$

$x = x^2$

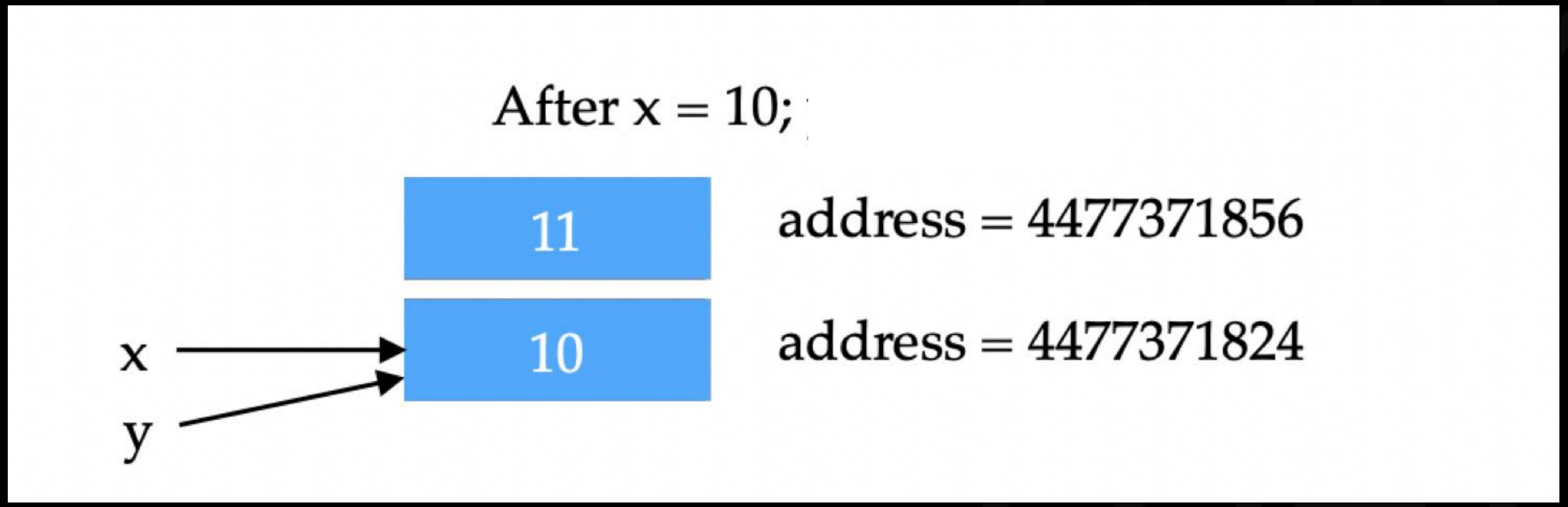
$x = 25$



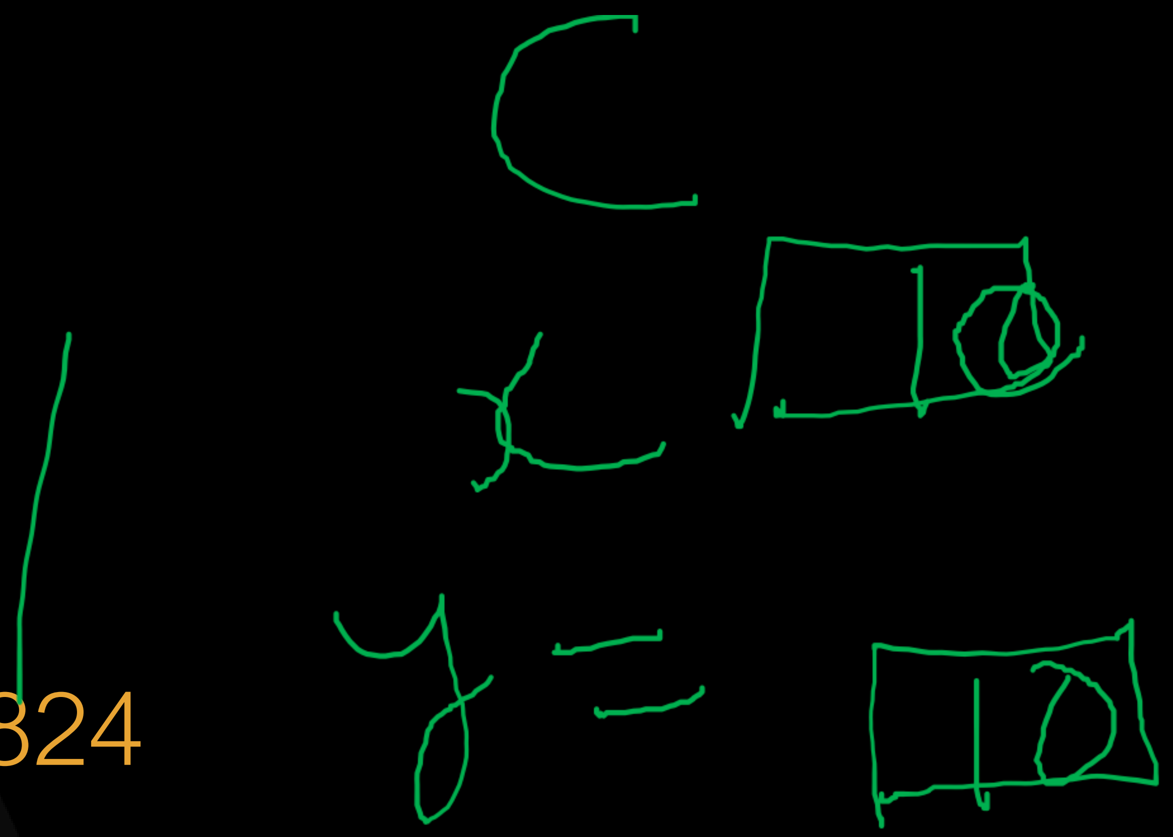
```
In [147]: x=5

In [148]: print(id(x))
4540589328

In [149]: x = x**2

In [150]: print(x, id(x))
25 4540589968
```

$x \rightarrow 5$

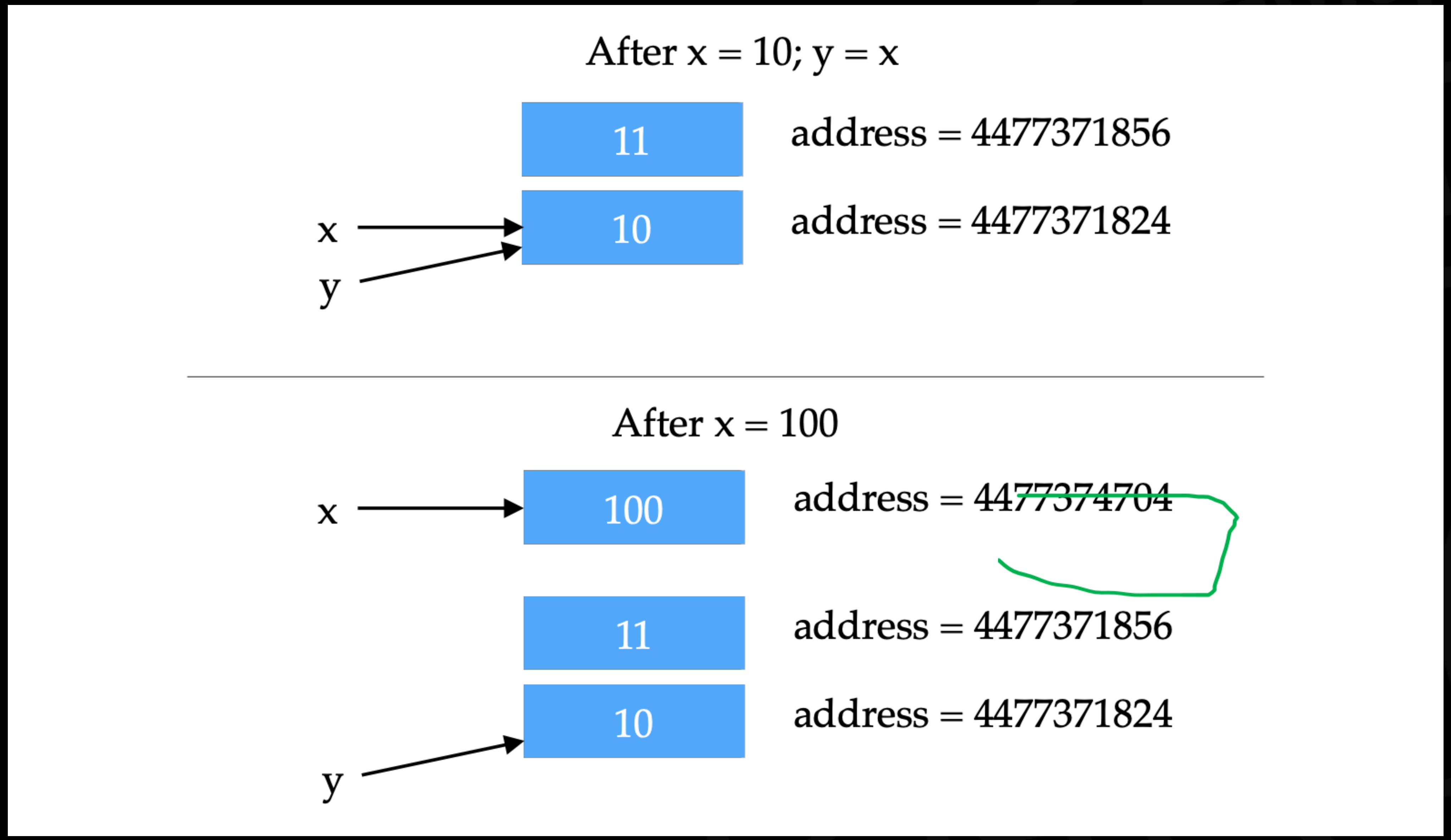
$x \rightarrow 25$



```
In [32]: y = x

In [33]: id(10)

Out[33]: 4477371824

In [34]: id(x)

Out[34]: 4477371824

In [35]: id(11)

Out[35]: 4477371856

In [36]: sys.getsizeof(10)
```



bytes



Objects 10, 100 remain unchanged

```
In [38]: id(y)

Out[38]: 4477371824

In [39]: x = 100

In [40]: id(x)

Out[40]: 4477374704

In [52]: id(100)

Out[52]: 4477374704
```

100

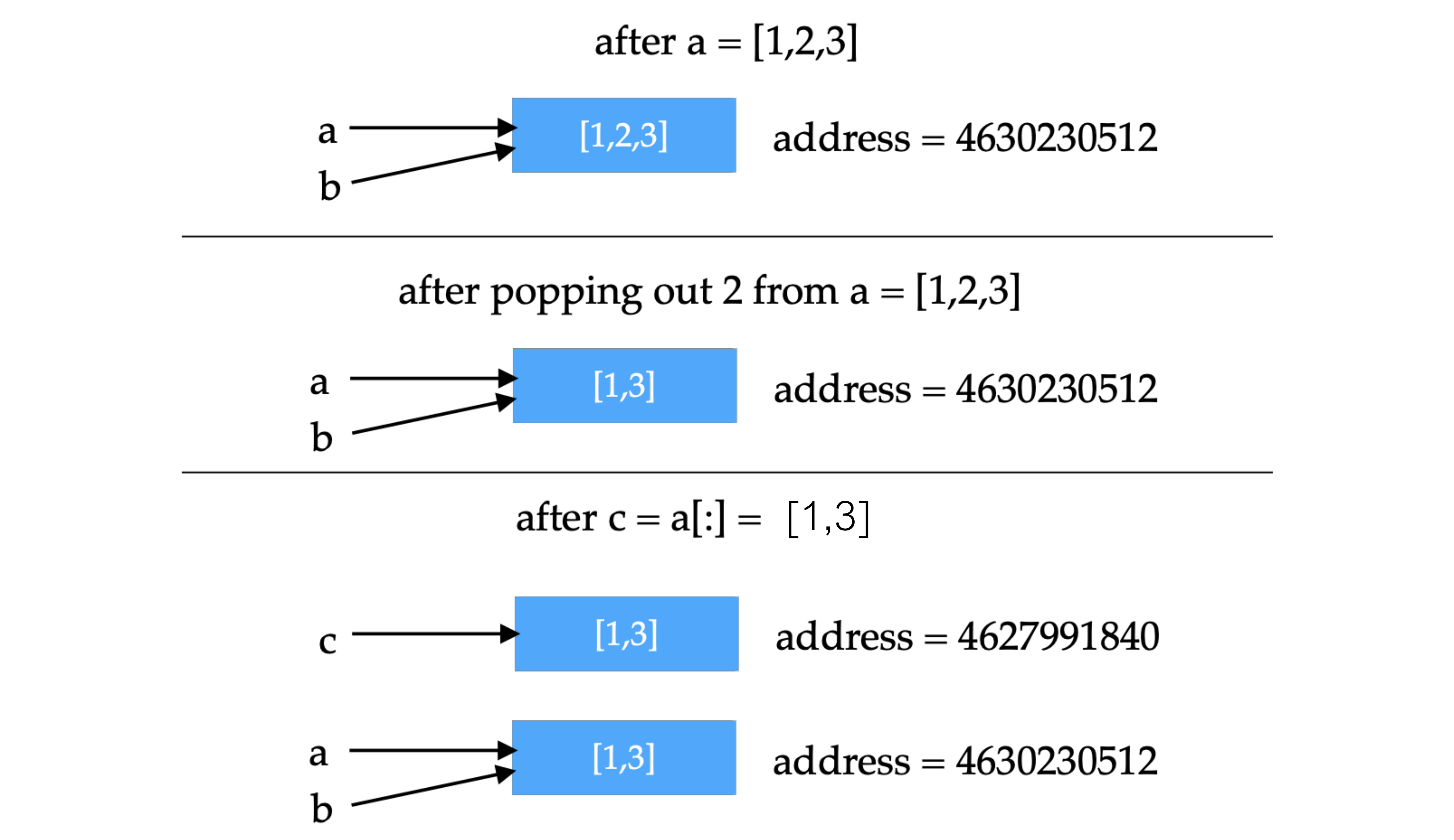
~~10~~

Integers and Floats are immutable objects.

```
In [53]: id(y)
```

Mutable objects

Lists & Arrays



after a = [1,2,3]

a

b

[1,2,3]

address = 4630230512

after popping out 2 from a = [1,2,3]

a

b

[1,3]

address = 4630230512

after c = a[:] = [1,3]

c

[1,3]

address = 4627991840

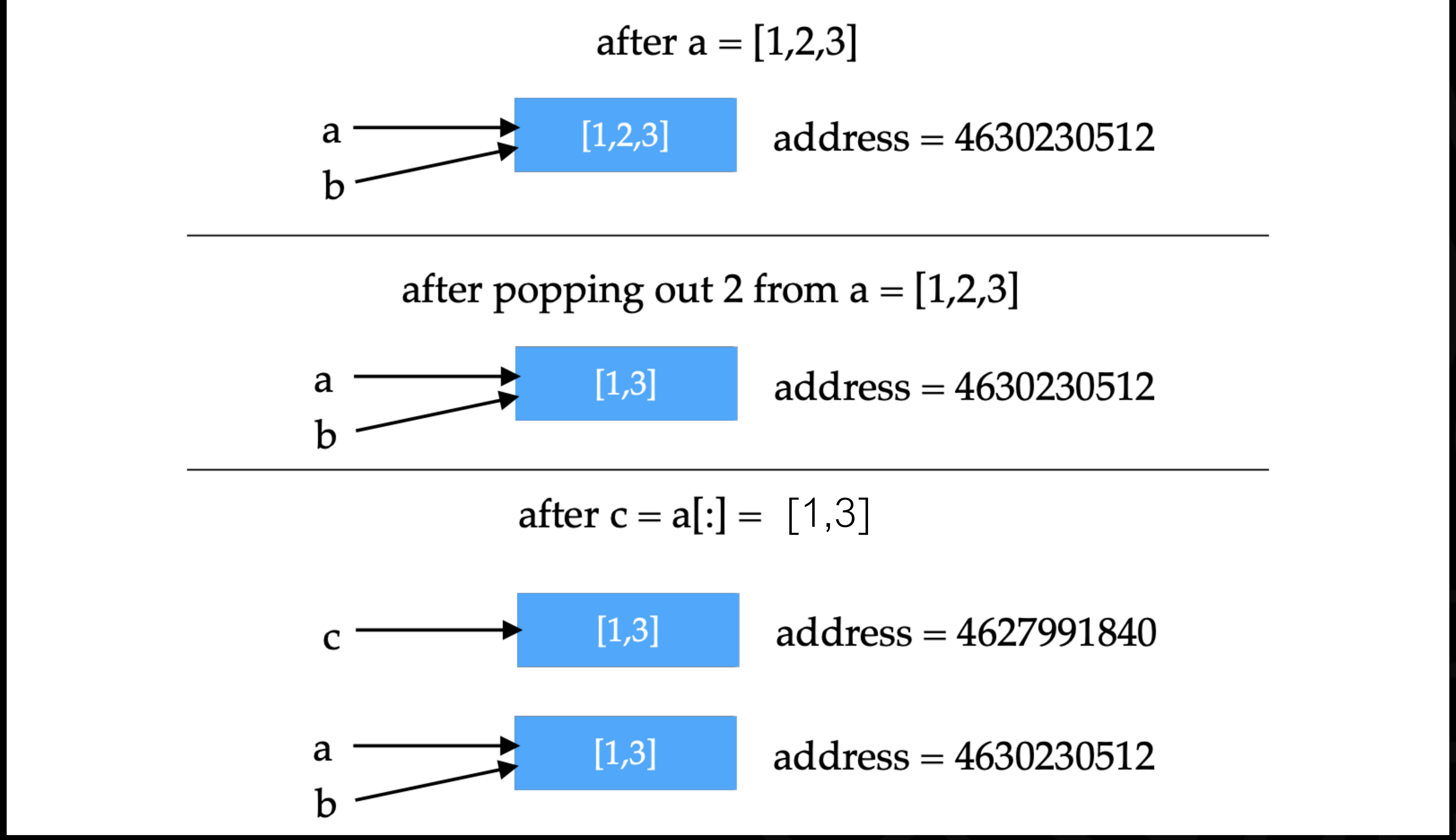
a

b

[1,3]

address = 4630230512

```
In [1]: a = [1,2,3]

In [2]: b=a

In [3]: print(id(a), id(b))

4630230512 4630230512

In [4]: a.pop(1)

Out[4]: 2

In [6]: print(a, b)

[1, 3] [1, 3]
```



after a = [1,2,3]

address = 4630230512

after popping out 2 from a = [1,2,3]

address = 4630230512

after c = a[:] = [1,3]

address = 4627991840

address = 4630230512

```
In [8]: c = a[:]
```

c = a

```
In [9]: print(c, id(a), id(c))
```

```
[1, 3] 4630230512
4627991840
```

Difference between Python and C arrays

```
In [166]: x = array([1,2,3])


In [167]: print(id(x))

140562475313552


In [168]: x = x**2


In [169]: print(x, id(x))

[1 4 9] 140562590651216
```

New object

How to fix it?

```
In [166]: x = array([1,2,3])

In [167]: print(id(x))

140562475313552

In [168]: x *= x

In [169]: print(x, id(x))

[1 4 9] 140562475313552
```

```
x = 5

print(x, id(x))

x *= x

print(x, id(x))

x = x**2

print(x, id(x))

x = None

print(x, id(x))
```

```
5 135022451966320

25 135022451966960

625 135022173479600

None 101171791987680
```

np array()

ipython

pylab

Same array object

- Same array if
  - x += 2

    x = x + 2
  - x -= 5
  - x //= 10

Comparison with C/C++

C

C = a

Python x = new

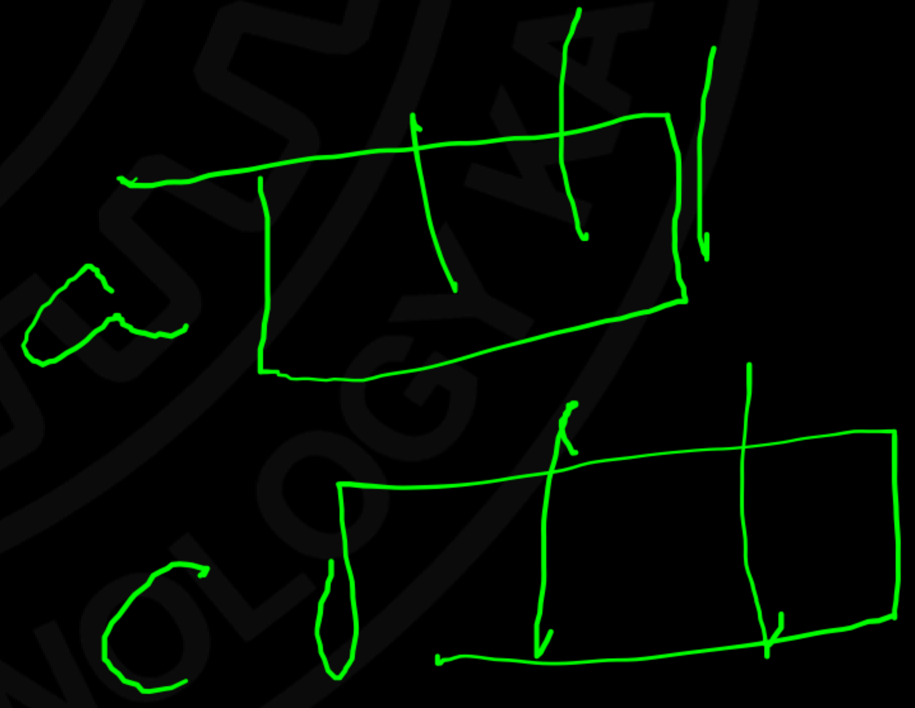
C = a

a = 2 + a

Copying a Python Array

c

c = a[:]



Python Assignment Operation

C = a

- Assignment statements do not copy objects.
- the = operator: It only creates a new variable that shares the reference of the original object.

```
import numpy as np

x = np.array([1,2,3])

print(x, id(x))



y = x

print(y, id(y))
```

x

y

Output:

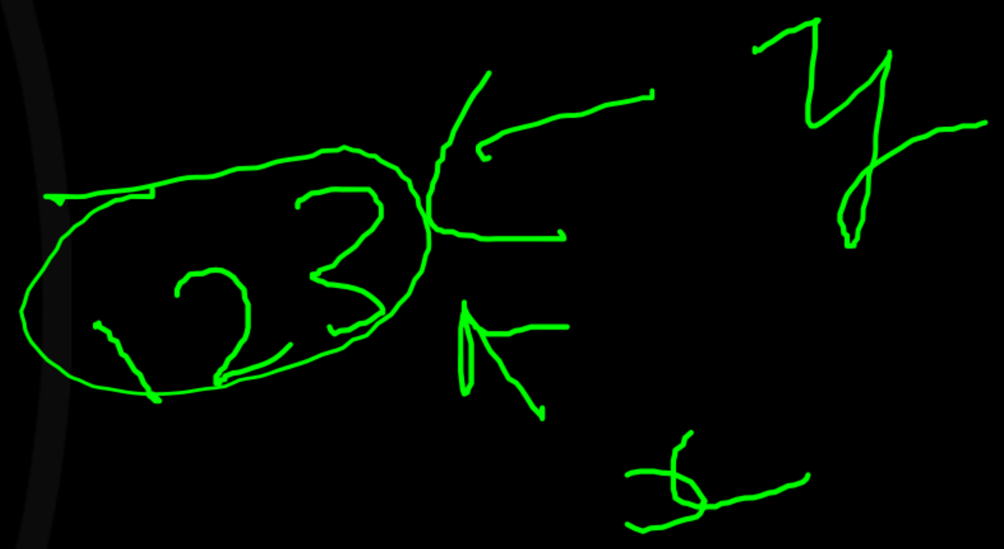
```
[1 2 3] 136812351023184



[1 2 3] 136812351023184
```

In C++/C, y will be a copy of x!

asarray(x)

- asarray(x): rename x or convert an object to an array

```
import numpy as np

x = np.array([1,2,3])

y = np.asarray(x)

print(x, y, id(x), id(y))



z = [2,3]

w = np.asarray(z)

print(z, w, id(z), id(w))
```



z = 2*z

[2 3 2 3]

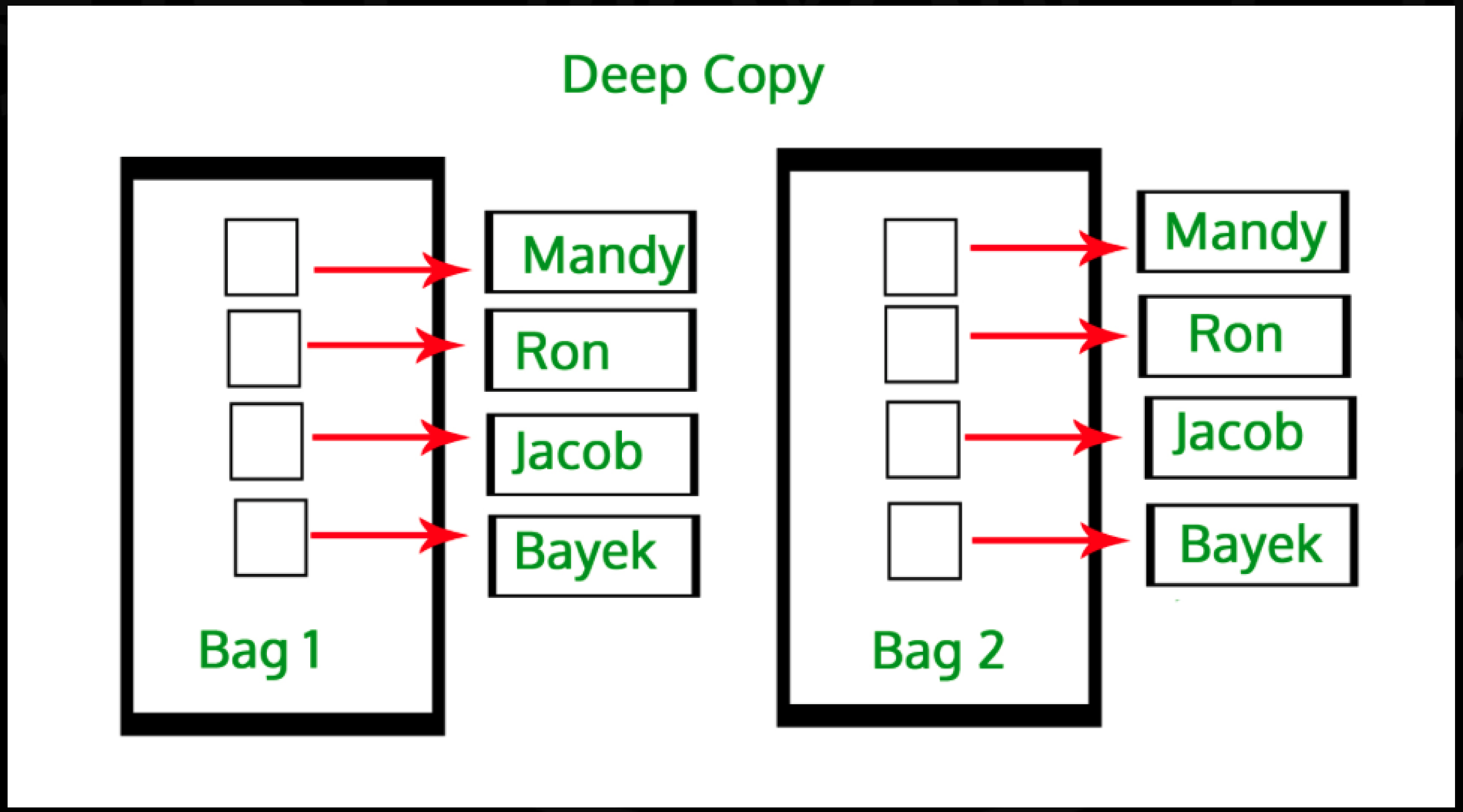
Output:

```
[1 2 3] [1 2 3] 136812185242672 136812185242672

[2, 3] [2 3] 136812183843520 136812185243920
```

copy.deepcopy()

Copy.copy()   c = a[:]

- copy.deepcopy(x) makes a deep copy
- All the objects of the structure copied recursively.



Deep Copy

Bag 1

Mandy
Ron
Jacob
Bayek

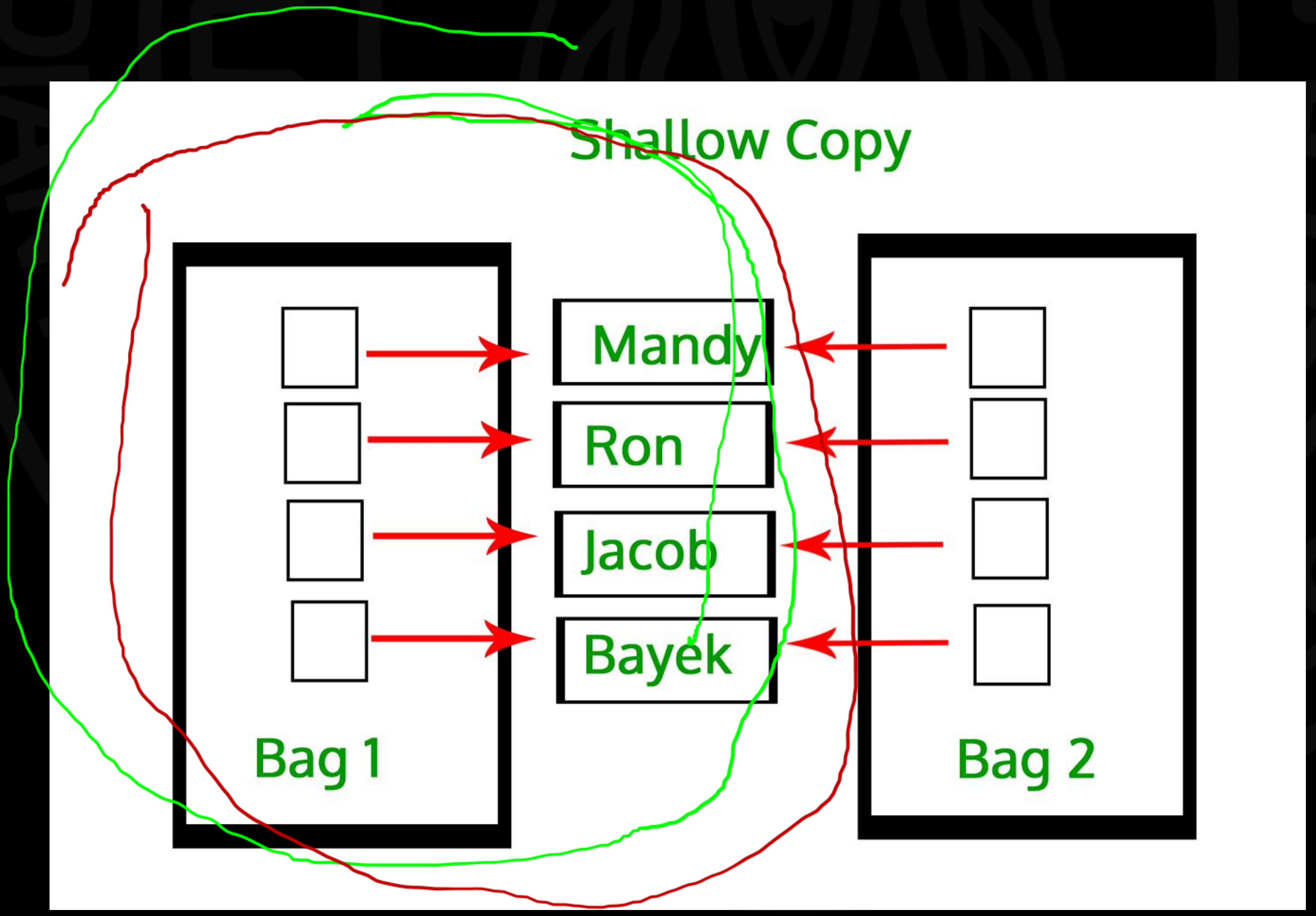
Bag 2

Mandy
Ron
Jacob
Bayek

https://www.geeksforgeeks.org/copy-python-deep-copy-shallow-copy/

copy.copy(x)

- copy.copy makes a shallow copy
- shallow copy creates a new compound object.
- constructs a new collection object and then populates it with references to the child objects found in the original.



https://www.geeksforgeeks.org/copy-python-deep-copy-shallow-copy/

copy()

```
import numpy as np

import copy

x = np.array([[1,2],[3,4]])

y = copy.copy(x)



y[0][0] = 100

print("x = ", x , "\n")

print("y = ", y , "\n")
```

$x = \begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}$

Output:

```
x =  [[1 2] [3 4]]

 id(x) =  132397090744016



y =  [[100   2] [3   4]]

 id(y) =  132397090740944
```

```
x= [1, 2, [3,5], 4]

y = copy.copy(x)

y[0] = 100

y[2][0]=200



print("x = ", x , "\n")

print("y = ", y , "\n")
```

100 200

Output:

```
x =  [1, 2, [200, 5], 4]

 id(x) =  132397091993408



y =  [100, 2, [200, 5], 4]

 id(y) =  132397255454016
```

deepcopy()

```
import numpy as np

import copy

x = np.array([[1,2],[3,4]])

y = copy.deepcopy(x)



y[0][0] = 100

print("x = ", x , "\n")

print("y = ", y , "\n")
```

Output:

```
x =  [[1 2] [3 4]]

 id(x) =  132397090744016


y =  [[100   2] [3   4]]

 id(y) =  132397090740944
```

```
x = np.array([1, 2, [3,5], 4],

    dtype = object)

y = copy.deepcopy(x)


y[0] = 100

y[2][0]=200


print("x = ", x , "\n")

print("y = ", y , "\n")
```

```
x =  [1 2 list([3, 5]) 4]


y =  [100 2 list([200, 5]) 4]
```

How to Avoid creating a new array?

```
import numpy as np

x = np.array([1,2,3])

print(x, id(x))



x = x**2

print(x, id(x))
```

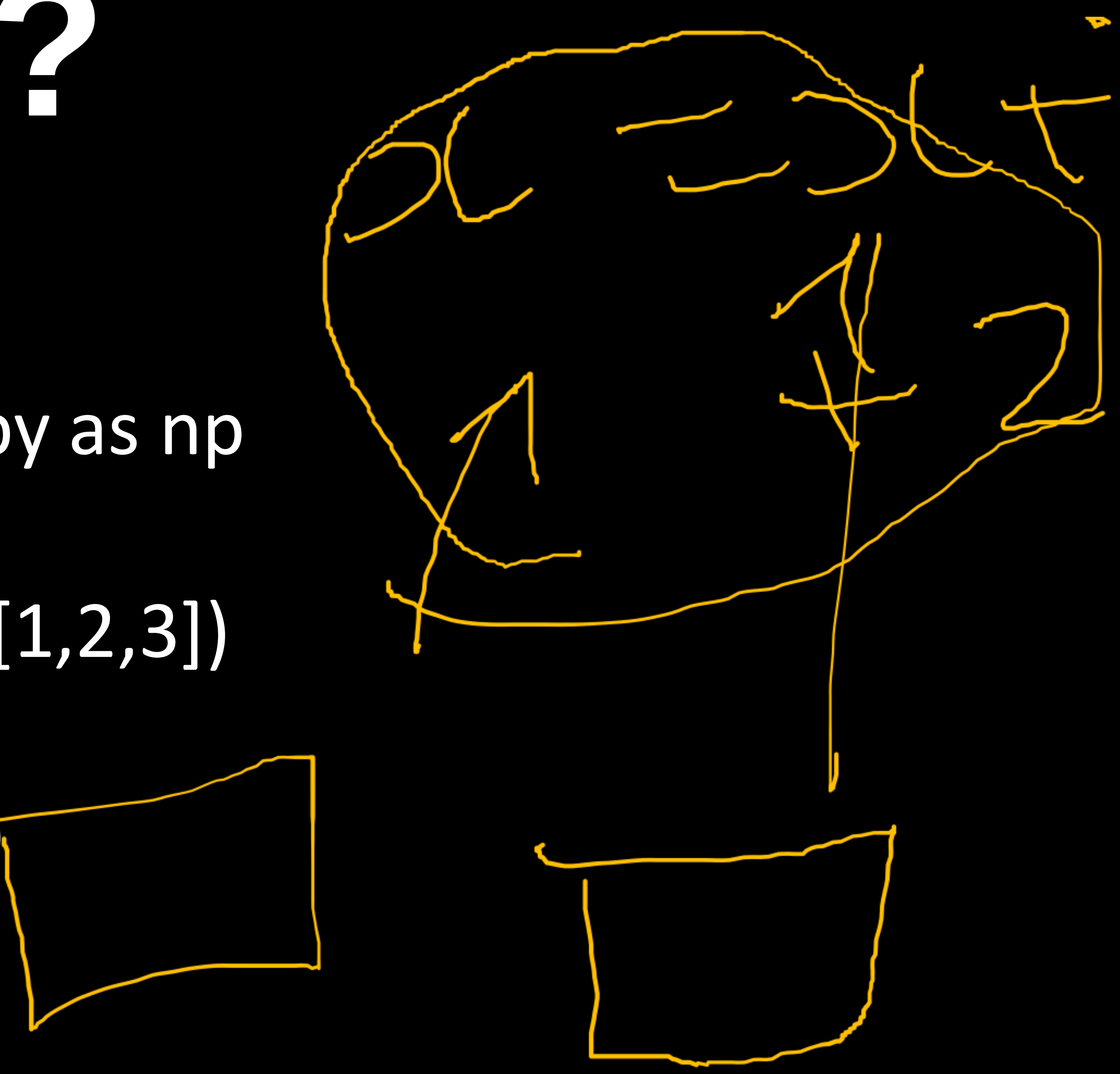
Output:

```
[1 2 3] 13502195596


[1 4 9] 1350219559649449
```

```
import numpy as np

x = np.array([1,2,3])

print(x, id(x))



x *= x

print(x, id(x))
```

Output:

```
[1 2 3] 132397255380016


[1 4 9] 132397255380016
```



How to Avoid creating a new array?

```
import numpy as np

x = np.array([1,2,3])

print(x, id(x))



x = x+2

print(x, id(x))
```

Output:

```
[1 2 3] 132397090744208


[3 4 5] 132397090740080
```

```
import numpy as np

x = np.array([1,2,3])

print(x, id(x))



x += 2

print(x, id(x))
```

Output:

```
[1 2 3] 132397255380016


[3 4 5] 132397255380016
```

How to Avoid creating a new array?

```
import numpy as np

x = np.array([1,2,3])

print(x, id(x))



~~x *= np.power(x,3~~)

print(x, id(x))
```

x^4

x

$x = x ** 4$

3

x^k

Output:

```
[1 2 3] 132397091032016



[ 1 16 81] 132397091032016
```

In C/C++

- Integer/float variables are *variables by value*.

In C/C++

- C++ objects are *variables by reference* or *variables by ~~reference~~* depending on the function call.

value

```
void test(type &arg);

// function declaration



test(myObject);

// function call
```

ref

```
void test(type *arg);

// function declaration
```

ref

```
void test(type arg);

// function declaration


type my object;

test(myObject);

// function call
```

value

In C/C++

- C++ objects are *variables by reference* or *variables by reference* depending on the function call.

```
void test(type &arg);

// function declaration



test(myObject);

// function call
```

```
Output:

x =  [[1 2] [3 4]]

  id(x) =  132397090744016



y =  [[100   2] [3   4]]

  id(y) =  132397090740944
```

Summary

- Python arrays and variables behave differently than C arrays.
- For example, in C/C++ with a = 2*a, the updated variable is same as before.
- But, not in Python.
- Avoid creating unnecessary Python arrays.

C Arrays

https://nyu-cds.github.io/python-numba/05-cuda/

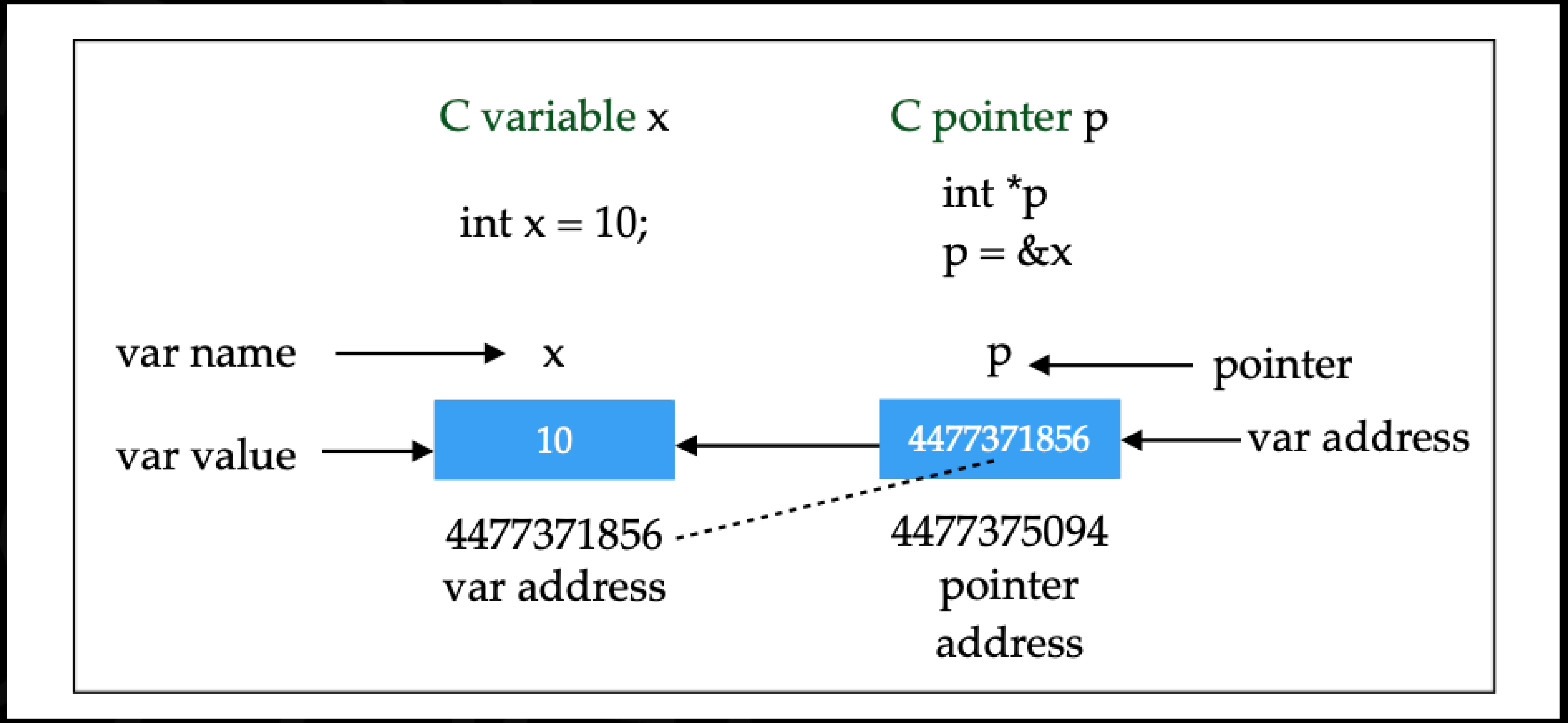
Mahendra Verma

C Pointers

- A pointer stores the address of another variable.
- p = &x
- & is the address getter.
- Indirection operator *
- x = *p

C Pointers

```
int main() {
  int x = 10;
  int* ptr;
  ptr = &x;
  // int* ptr = &x;

  // Output the value of x
  printf("x = %d %d \n", x, *ptr);

  // Output the memory address of x
  printf("ptr, &x = %p %p  \n", ptr, &x);

  return 0;
}
```



```
x = 10 10

ptr, &x = 4477371856, 4477371856
```

C Static Arrays (1D)

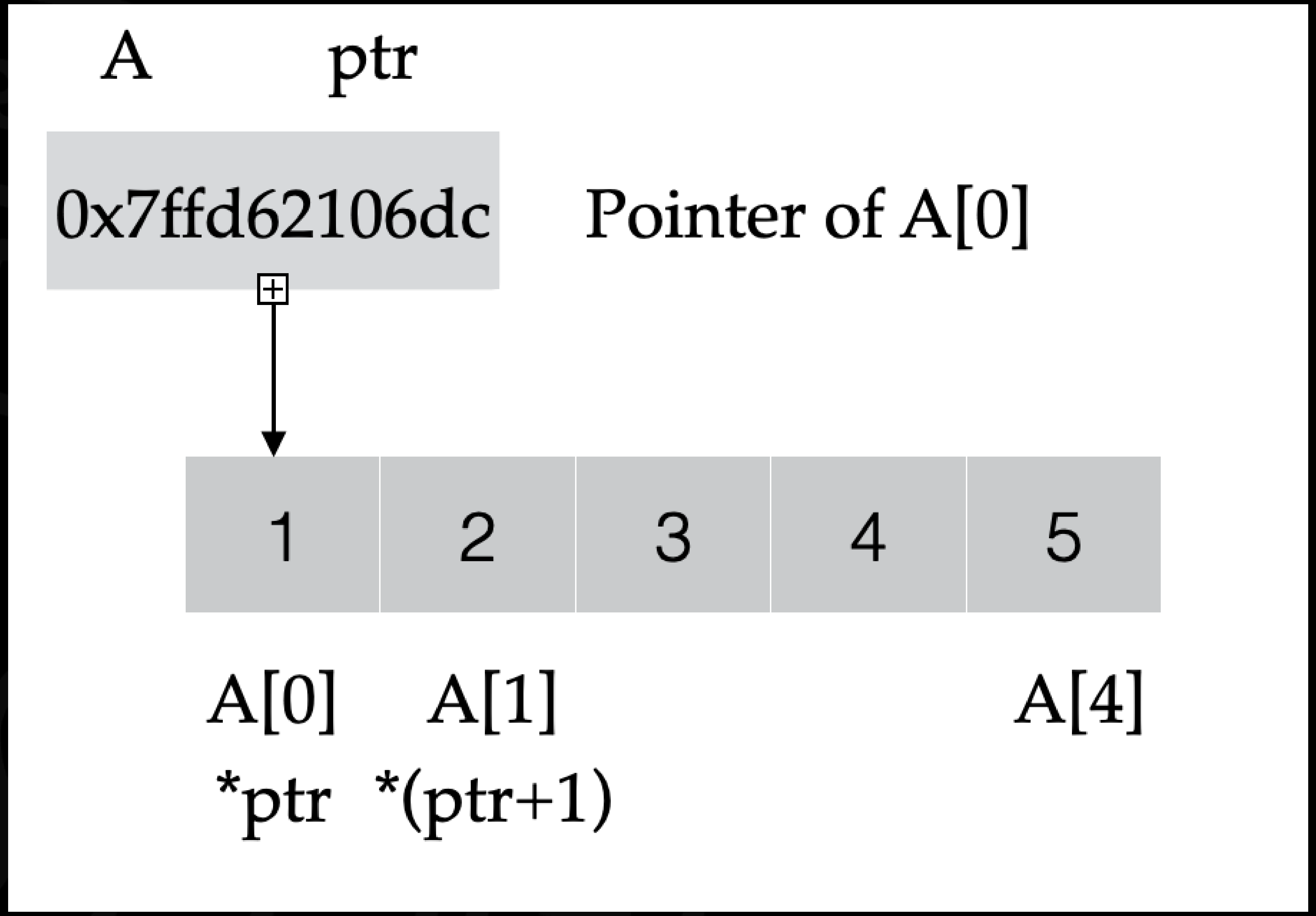
```
int main() {
  int N = 4;
  int a[N], b[N], c[N];

  for (int i = 0; i < N; i++) {
      a[i] = i; b[i] = 1;
  }

  add(a, b, c, N);
  add2(a, b, c, N);
}
```

```
void add(int a[], int b[], int c[], int N)
{
  for (int i=0; i<N; i++)
    c[i] = a[i] + b[i];
}
```

```
void add2(int *a, int *b, int *c, int N)
{
  for (int i=0; i<N; i++)
    *(c+i) = *(a+i) + *(b+i);
}
```

C Arrays & Pointers

```
int main()
{
    int A[5] = { 1, 2, 3, 4, 5 };
    int *ptr = A;

    printf("A, ptr, ptr of A[0] = %p %p %p \n", A,
ptr, &(A[0]));
    printf("A[0], *(ptr), *(ptr+1) = %d %d %d \n",
       A[0], *(ptr), *(ptr+1));
    return 0;
}
```



```
A, ptr, ptr of A[0] = 0x7ffe02540740
0x7ffe02540740 0x7ffe02540740

A[0], *(ptr), *(ptr+1) = 1 1 2
```

Dynamic Arrays & Malloc

```
!gcc c_array1D_dynamic.c

!./a.out 5
```

```
int main(int argc, char* argv[]) {
  // strlol = string to long integer
  int N = strtol(argv[1], NULL, 10);
  int *a;

  a = (int*)malloc(N * sizeof(int));

  for (int i = 0; i < N; i++) {
      a[i] = i;
      printf(" i, a[i] = %d %d \n", i,  a[i]);
  }
}
```

```
i, a[i] = 0 0

i, a[i] = 1 1

i, a[i] = 2 2

i, a[i] = 3 3

i, a[i] = 4 4
```

C Dynamic Arrays (1D)

Array as argument

```
int main(int argc, char* argv[]) {
  int N = strtol(argv[1], NULL, 10);
  int *a;

  a = (int*)malloc(N * sizeof(int));

  for (int i = 0; i < N; i++) {
      a[i] = i;
      printf(" i, a[i] = %d %d \n", i,  a[i]);
  }

  print_array(a, N);

  return 0;
}
```

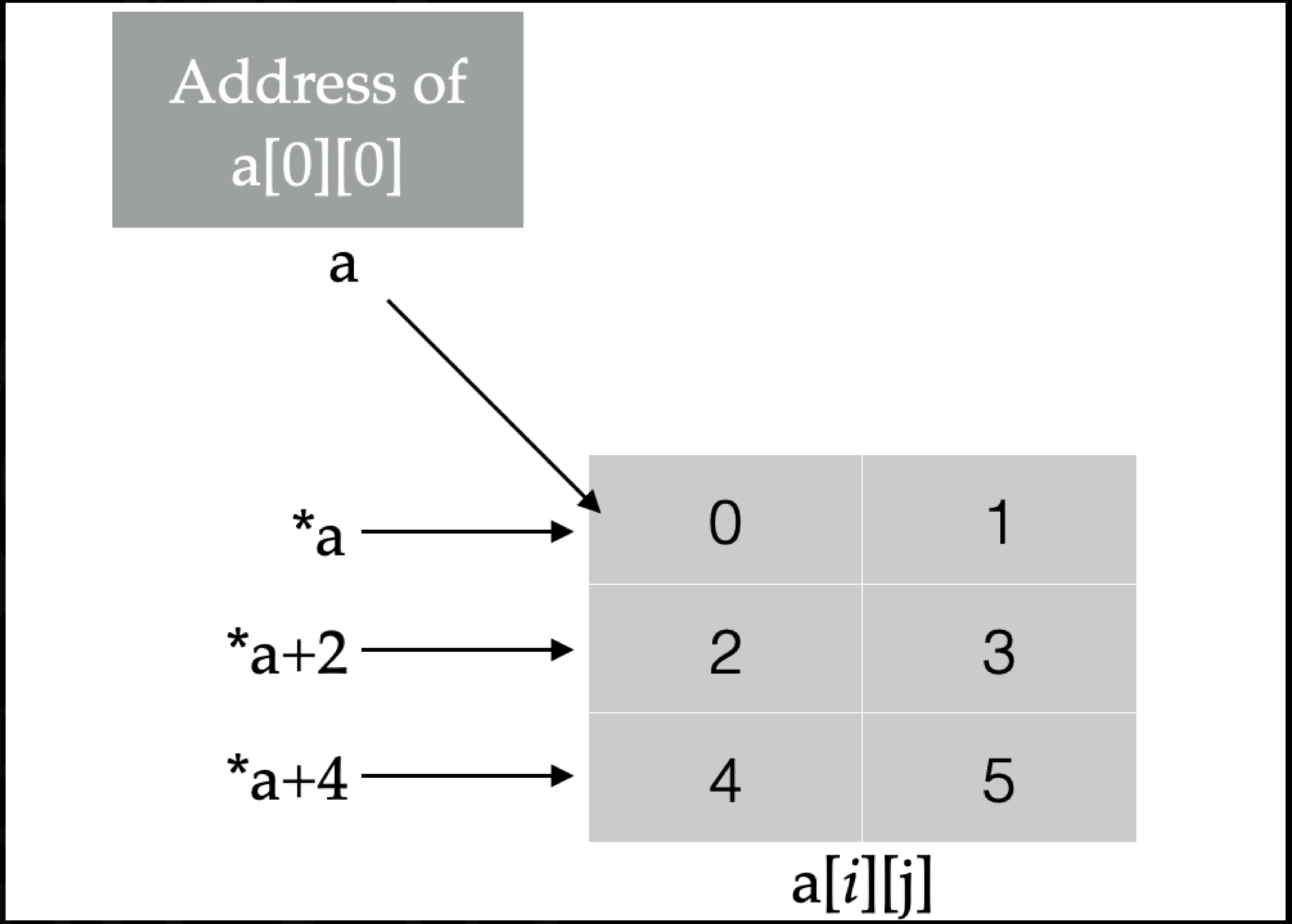
```
void print_array(int *a, int N)
{
  for (int i=0; i<N; i++)
    printf(" i, a[i] = %d %d \n", i,  *(a+i));
}
```

```
i, a[i] = 0 0

i, a[i] = 1 1

i, a[i] = 2 2

i, a[i] = 3 3

i, a[i] = 4 4
```

Higher-dimensional arrays

2D Array

```
int main(void)
{
    // an array with 3 rows and 2 columns.
    int a[3][2] = { { 0, 1 }, { 2, 3 }, { 4, 5 } };

    int* ptr = &a[0][0];
    printf("%p, %p, %p, %p \n", ptr, a,  *a, *a+1 );
    printf("%d %d %d \n", *(*a+1),*(*a+2),*((*a+3)  );
    return (0);
}
```



```
0x7ffc9bca06c0, 0x7ffc9bca06c0, 0x7ffc9bca06c0,
0x7ffc9bca06c4
1 2 3
```

C Static Arrays (2D)

```
void func1(int row, int col, int matrix[row][col]){
    int i, j;
    for(i=0; i<row; i++){
        for(j=0; j<col; j++){
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }
}
```

```
int main() {
    int row = 3;
    int col = 3;
    int a[row][col];
    for(int i=0; i<row; i++)
        for(int j=0; j<col; j++)
            a[i][j] = i+j;
}
```

```
void func2(int row, int col, int matrix[][col])
```

C Static Arrays (2D)

```
void func3(int row, int col, int *matrix){
    int i, j;
    for(i=0; i<row; i++){
        for(j=0; j<col; j++){
            printf("func3: %d ", matrix[i*col+j]);
        }
        printf("\n");
    }
}
```

C Dynamic Arrays (2D)

As 1D array

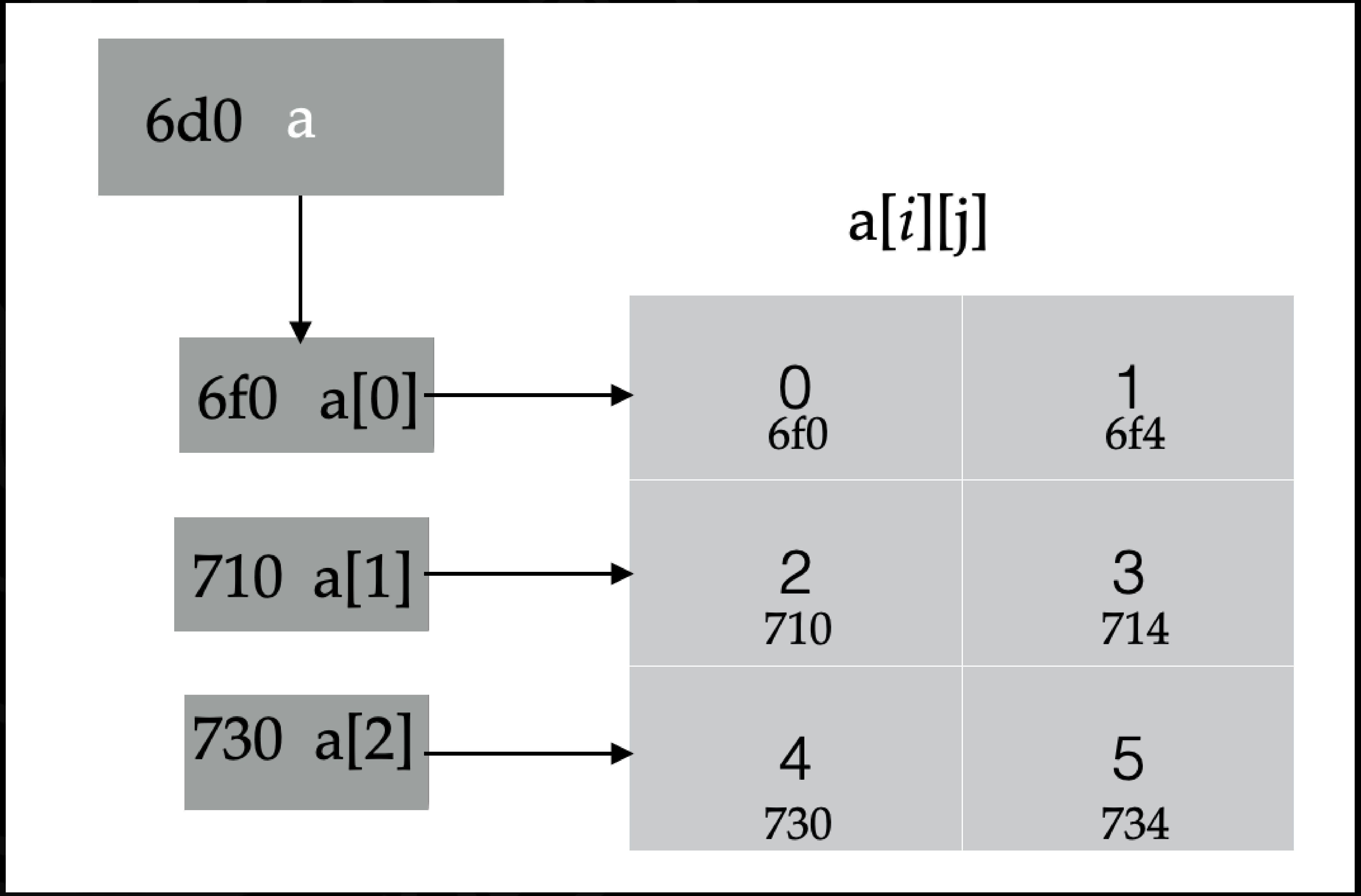
```
int main(int argc, char* argv[]) {

    int m = strtol(argv[1], NULL, 10);
    int n = strtol(argv[2], NULL, 10);
    int *a;

    a = (int*)malloc(m*n *
sizeof(int));

    for(int i=0; i<m; i++)
        for(int j=0; j<n; j++)
            *(a + i*n + j) = I+j;

}
```

```
void func1(int m, int n, int* matrix){
    int i, j;
    for(i=0; i<m; i++){
        for(j=0; j<n; j++){
            printf("%d  \n ", *(matrix + i*n + j));
        }
        printf("\n");
    }
}
```

C Dynamic Arrays (2D)

As array of arrays

```
int main(int argc, char* argv[]) {

 int **arr = (int **)malloc(m * sizeof(int *));

    for (int i = 0; i < m; i++) {
        arr[i] = (int *)malloc(n * sizeof(int));
        printf("arr = %p \n ", arr[i]);
        for (int j = 0; j < n; j++) {
            arr[i][j] = i+j;
        }
    }

    func2(m, n, arr);
}
```



C Dynamic Arrays (2D)

As array of arrays

```
void func2(int m, int n, int** matrix){
    int i, j;
    for(i=0; i<m; i++){
        for(j=0; j<n; j++)
            printf("func2: %d ", matrix[i][j]);

        printf("\n");
    }
}
```

C Static Arrays (3D)

```
int arr[2][2][1];

    for(int i=0; i<m; i++)
        for(int j=0; j<n; j++)
            for(int k=0; k<p; k++)
                arr[i][j][k] = i+j+k;
```

```
void func1(int m, int n, int p, int
matrix[][n][p]) {
    int i, j, k;
    for(i=0; i<m; i++){
        for(j=0; j<n; j++)
            for(k=0; k<p; k++)
                printf(" %d  ", matrix[i][j][k]);
        printf("\n");
    }
}
```

C Dynamic Arrays (3D)

As 1D Array

```
    int *a;

   a = (int*)malloc(m1*n1*p1 * sizeof(int));

    for(int i=0; i<m1; i++)
        for(int j=0; j<n1; j++)
            for(int k=0; k<p1; k++)
                *(a + i*n1*p1 + j*p1 + k) = i+j+k;

func2(m1, n1, p1, a);
```

```
void func2(int m, int n, int p, int* matrix){
    int i, j, k;
    for(i=0; i<m; i++) {
        for(j=0; j<n; j++)
            for(k=0; k<p; k++)
                printf("%d ", *(matrix + i*n*p + j*p + k));
        printf("\n");
    }
}
```

```
int main(int argc, char* argv[])
```

int main(int argc, char* argv[])

- Int argc: number of arguments
- Char *argv: charcter pointer string
- argv[]: Name of the program
- int threads_count = strtol(argv[1], NULL, 10);
- strtol: String to int with base 10.

```
!g++ hello_world.cc -fopenmp

!./a.out 3
```

- argc = 2
- argv[0] = a.out
- argv[1] = 3

भारतीय प्रौद्योगिकी संस्थान कानपुर
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

भारतीय प्रौद्योगिकी संस्थान कानपुर

INDIAN INSTITUTE OF TECHNOLOGY KANPUR