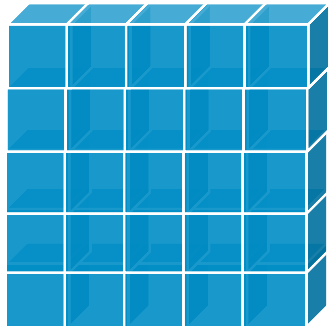
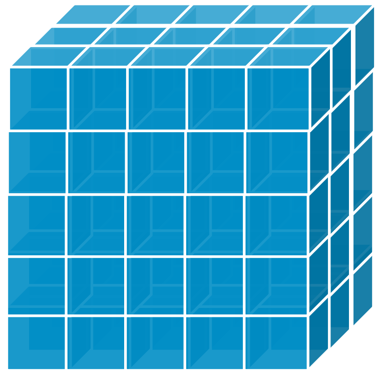
NumPy: Multi-Dimensional Arrays

Understanding Society

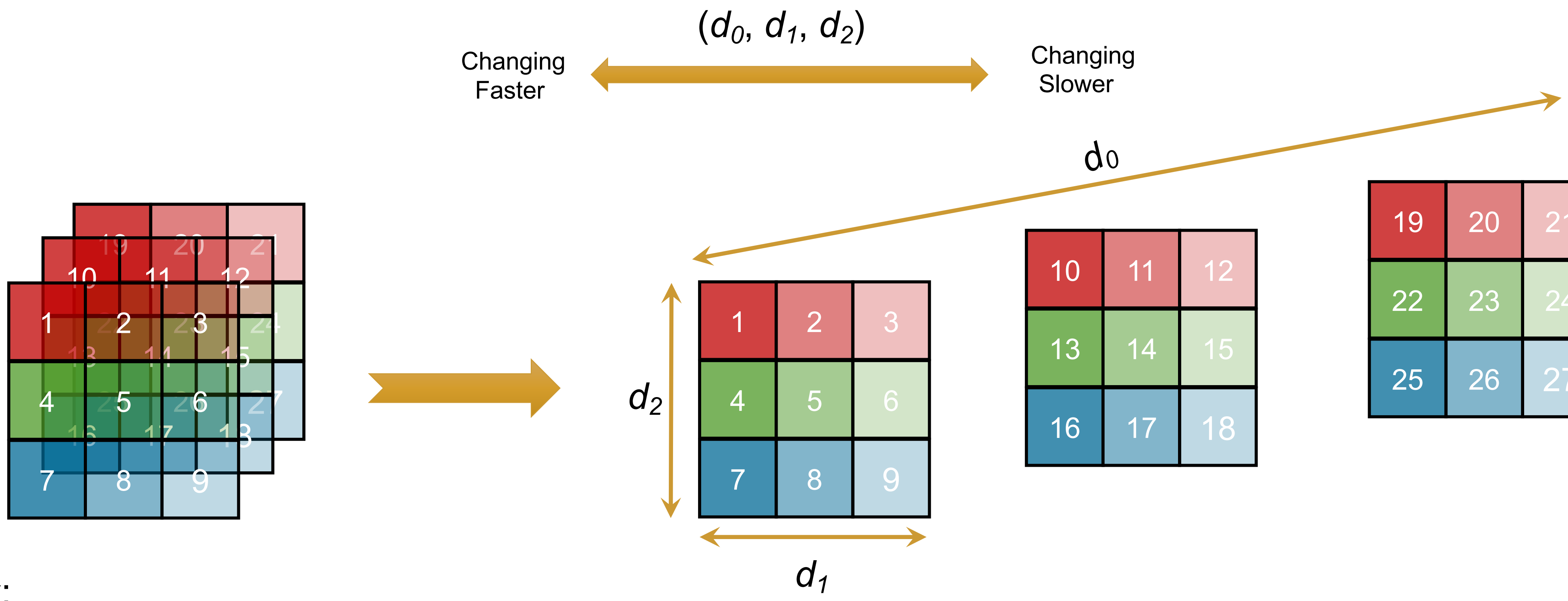
Multi-Dimensional Arrays in NumPy

- Multi-dimensional arrays (ndarrays) are basic data structures in NumPy:
  - 1-dimensional array (vector): e.g., stock price
  - 2-dimensional array (matrix): e.g., a table of students' grades
  - N-dimensional array (tensor): e.g., a color movie

Numpy Multi-Dimensional Arrays in Memory

❖ The ‘C’ memory strategy for a 3-dimensional array:

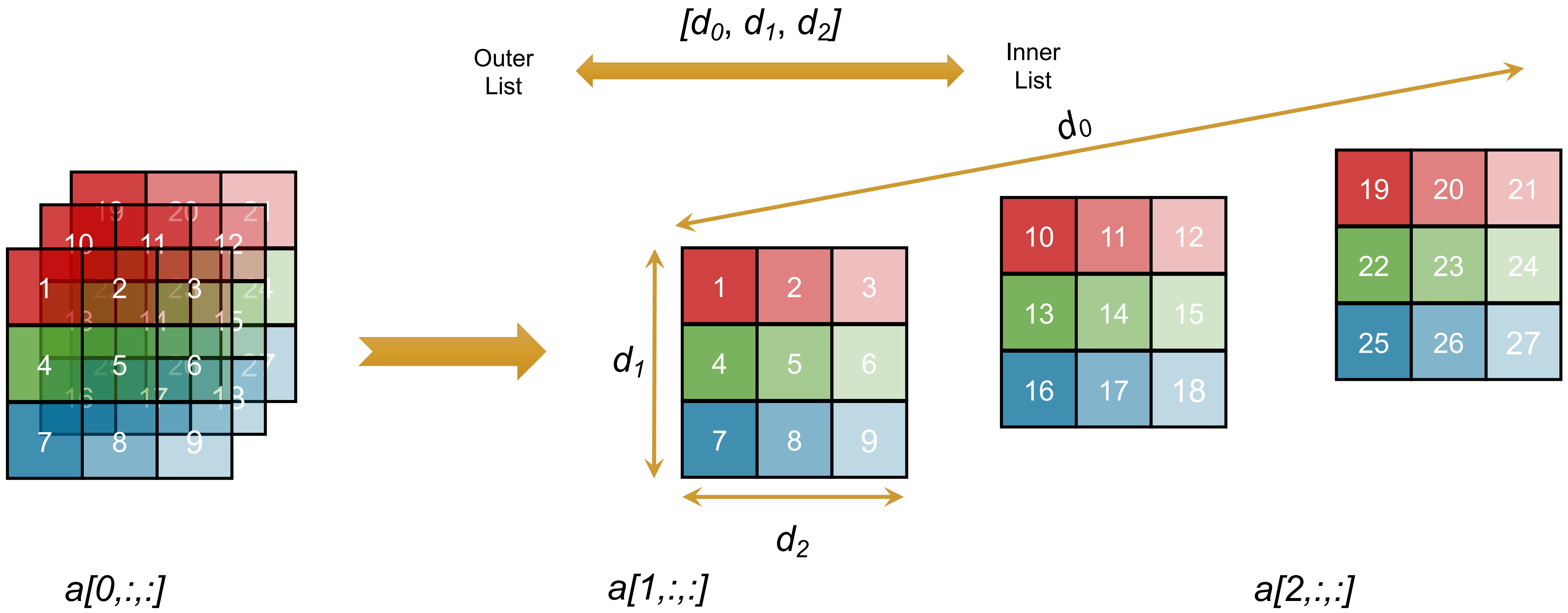


In memory:

1	10	19	2	11	20	3	12	21	4	13	22	5	14	23	6	15	24	7	16	25	8	17	26	9	18

Numpy Multi-Dimensional Arrays in Python

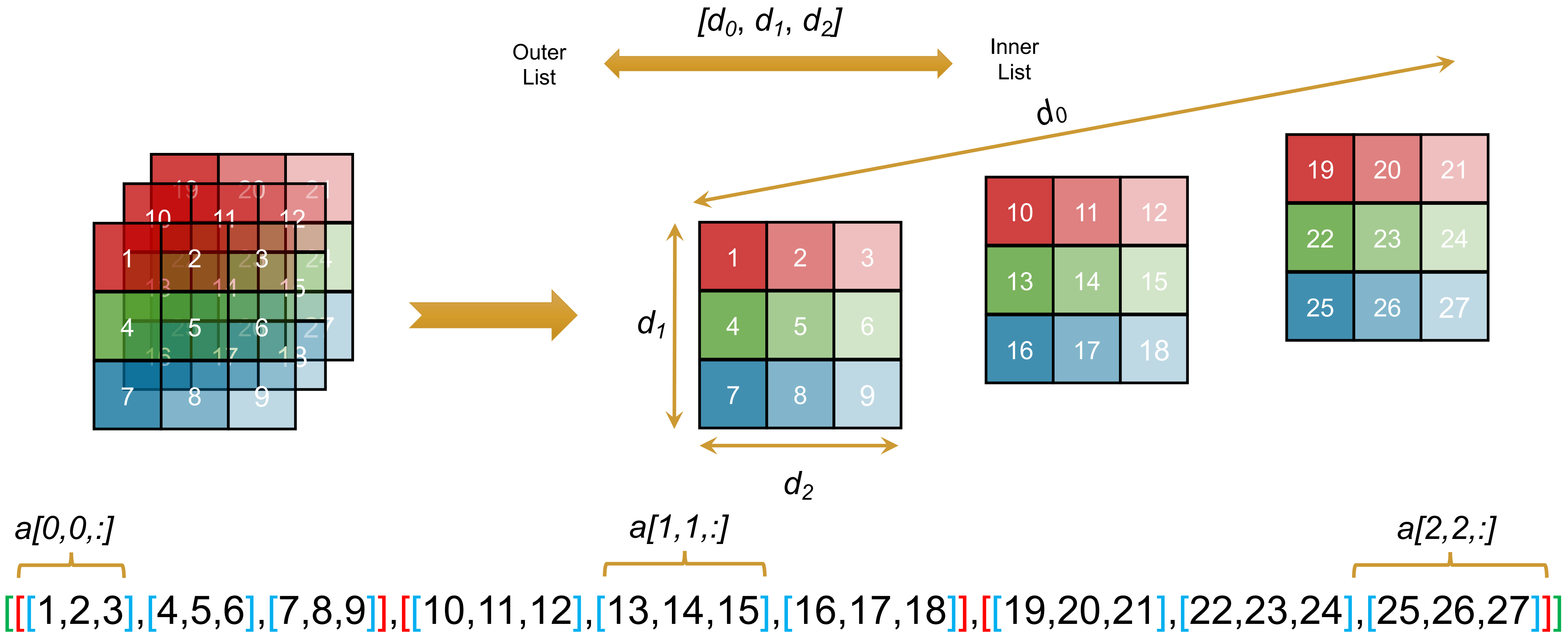
❖ The Numpy multi-dimensional arrays are represented as nested-lists in Python.



[[[1,2,3],[4,5,6],[7,8,9]],[[10,11,12],[13,14,15],[16,17,18]],[[19,20,21],[22,23,24],[25,26,27]]]

Numpy Multi-Dimensional Arrays in Python

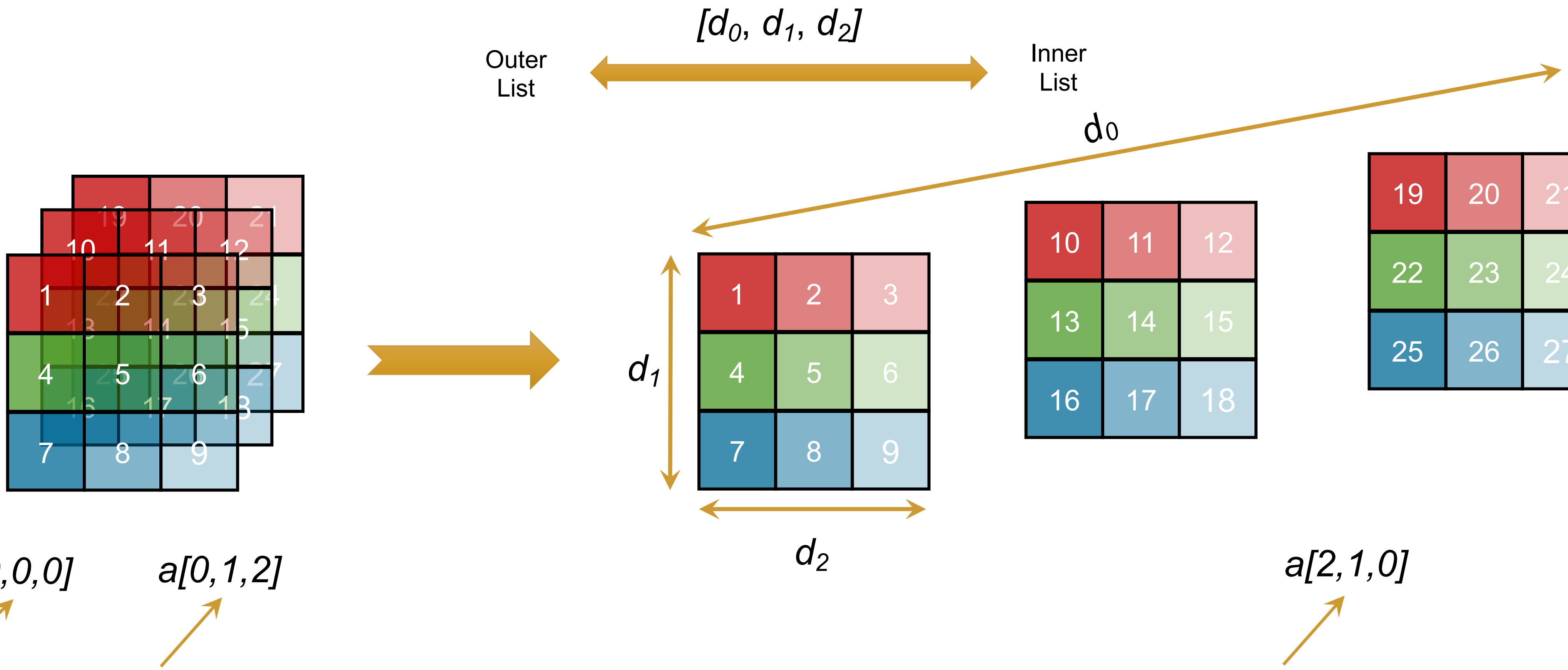
❖ The Numpy multi-dimensional arrays are represented as nested-lists in Python.



[[[1,2,3],[4,5,6],[7,8,9]],[[10,11,12],[13,14,15],[16,17,18]],[[19,20,21],[22,23,24],[25,26,27]]]

Numpy Multi-Dimensional Arrays in Python

❖ The Numpy multi-dimensional arrays are represented as nested-lists in Python.

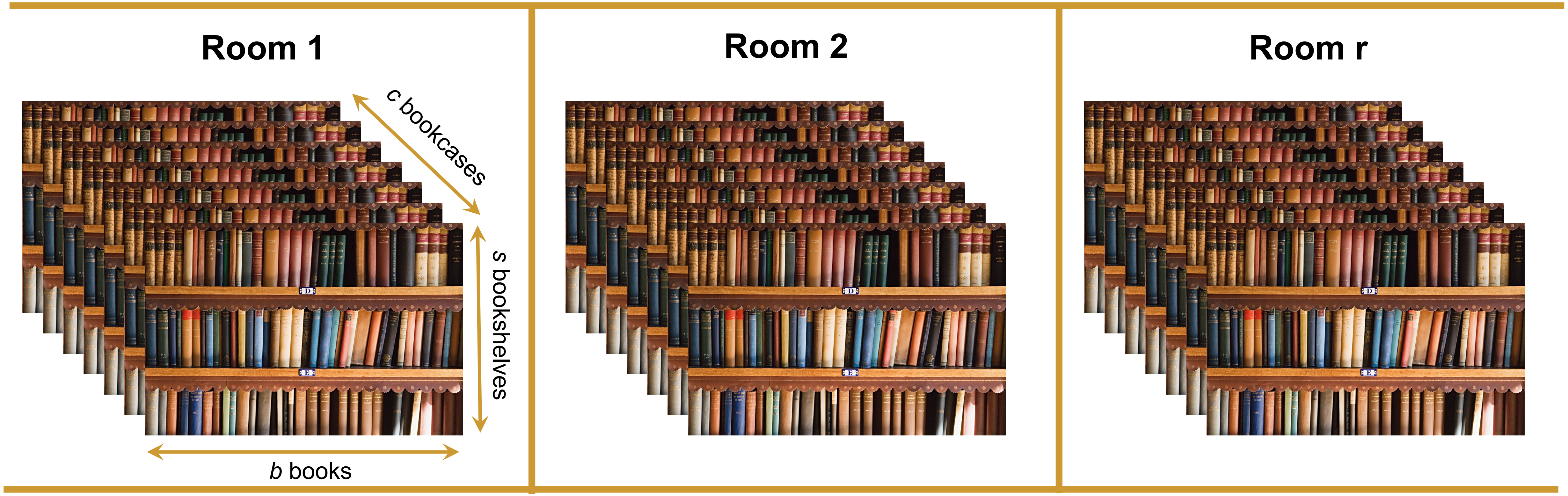


[[[1,2,3],[4,5,6],[7,8,9]],[[10,11,12],[13,14,15],[16,17,18]],[[19,20,21],[22,23,24],[25,26,27]]]

Questions?

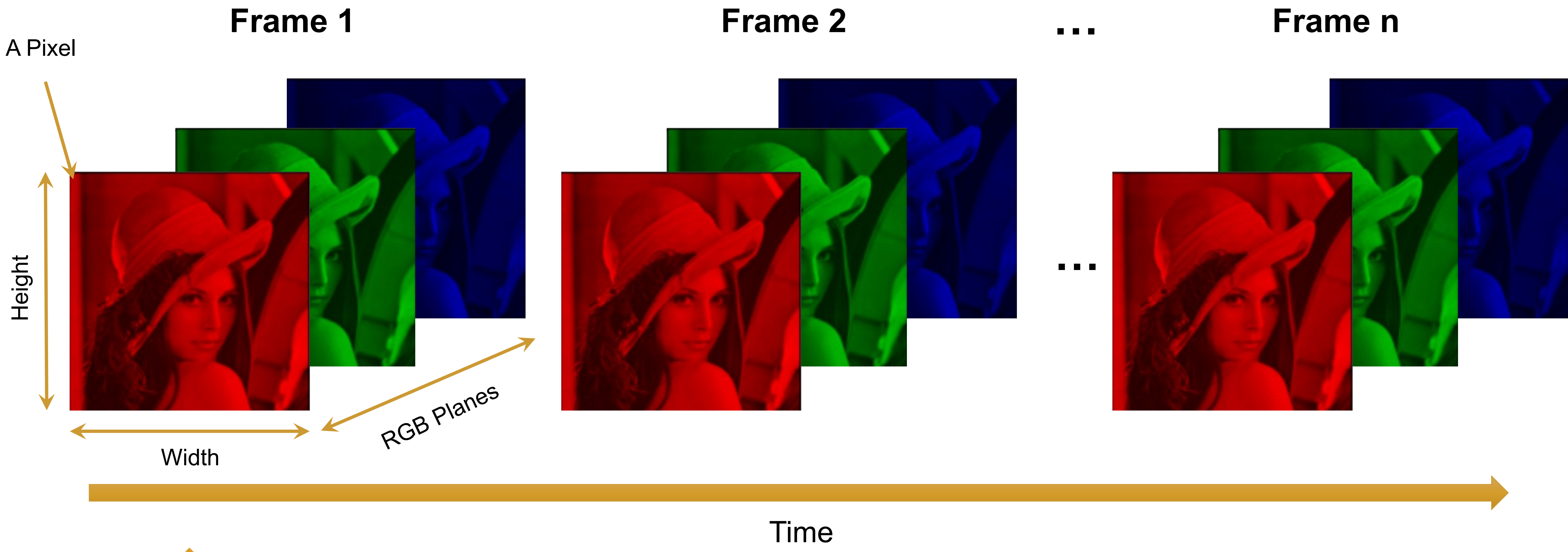
Numpy Multi-Dimensional Arrays: Human Understanding

As a human user, we intend to assign a meaning to each dimension. For example, a 4-dimentional array can be seen as *r* rooms of a library each of which with *c* bookcases, each of which with *s* bookshelves, and each of which with *b* books.



Multi-Dimensional Arrays: A Real Example

❖ A digital color movie is a 4D array:



Questions?

Creating Multidimensional Arrays

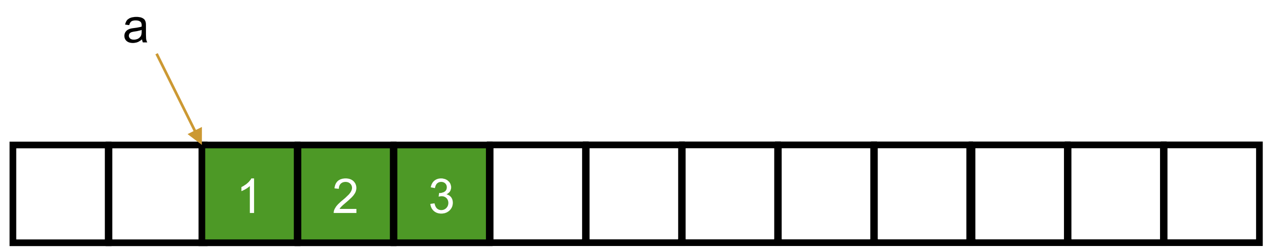
1) From a Python lists or tuples:
   - `numpy.array`
2) From a file:
   - Text file for up to two dimensions: `numpy.savetxt` and `numpy.loadtxt`
   - Binary file for any dimensions: `numpy.save` and `numpy.load`
3) Using intrinsic NumPy array creation functions:
   - `numpy.empty`
   - `numpy.zeros`
   - `numpy.ones`
   - `numpy.eye`
4) Creating arrays of random values
   - `numpy.random.random`
   - `numpy.random.randint`
   - `numpy.random.uniform, numpy.random.normal`

Questions?

Shallow versus Deep Copy

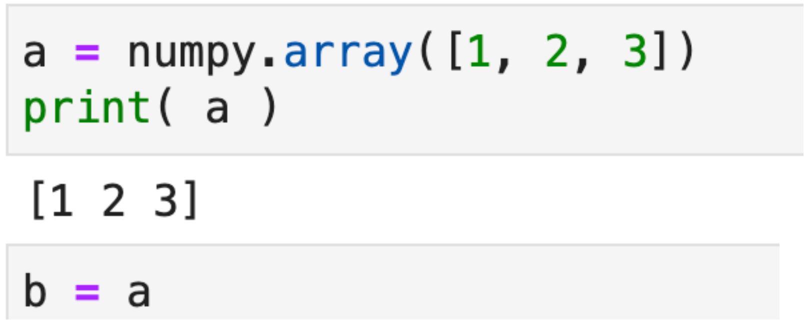
```
a = numpy.array([1, 2, 3])
print( a )
```

```
[1 2 3]
```



Shallow versus Deep Copy

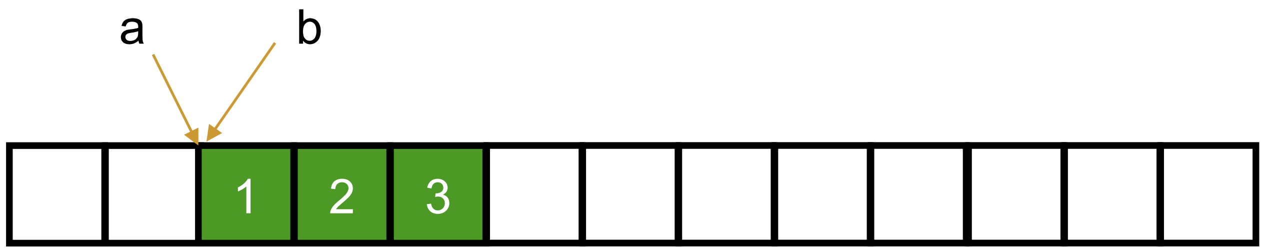
Assignment statements in Python do not copy objects, they create bindings between a variable name and a memory address of an object.



```
a = numpy.array([1, 2, 3])
print( a )
```

```
[1 2 3]
```

```
b = a
```



Shallow versus Deep Copy

Assignment statements in Python do not copy objects, they create bindings between a variable name and a memory address of an object.

```
a = numpy.array([1, 2, 3])
print( a )
```

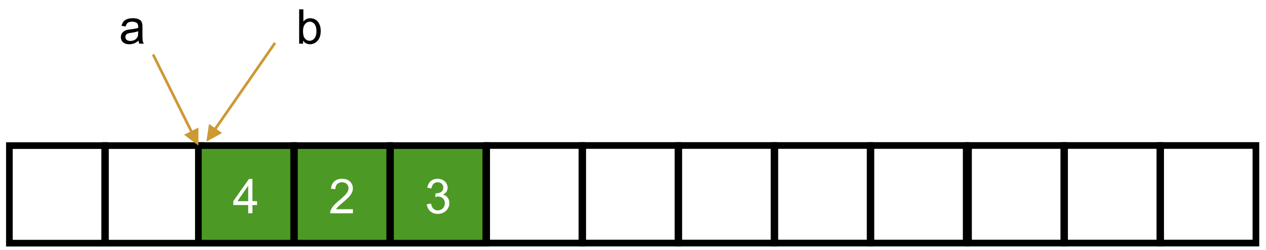
```
[1 2 3]
```

```
b = a
b[0] = 4
print(b)
```

```
[4 2 3]
```

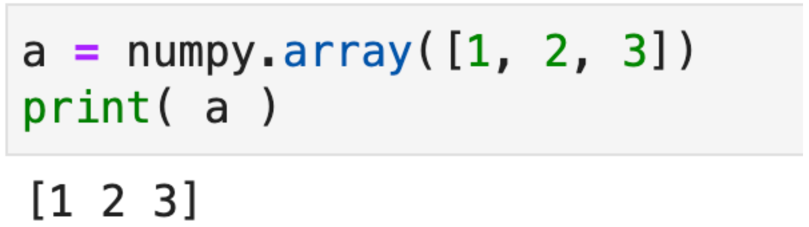
```
print(a)
```

```
[4 2 3]
```



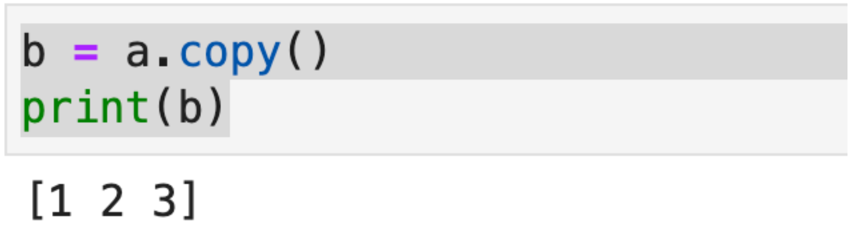
Assignment creates a shallow copy.

Shallow versus Deep Copy



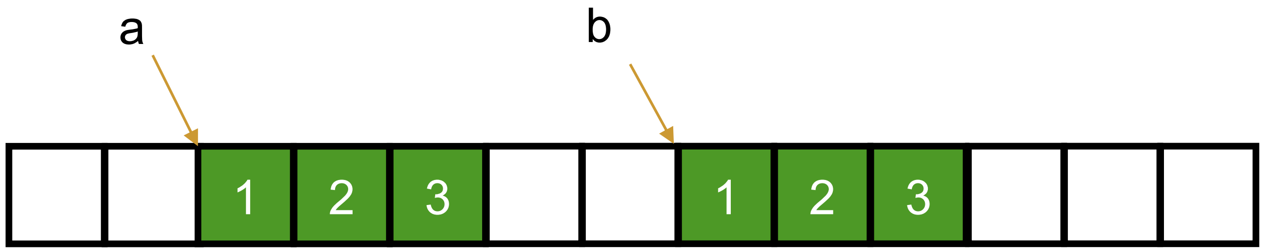
```
a = numpy.array([1, 2, 3])
print( a )
```

```
[1 2 3]
```



```
b = a.copy()
print(b)
```

```
[1 2 3]
```



Shallow versus Deep Copy

```
a = numpy.array([1, 2, 3])
print( a )
```

```
[1 2 3]
```

```
b = a.copy()
print(b)
```

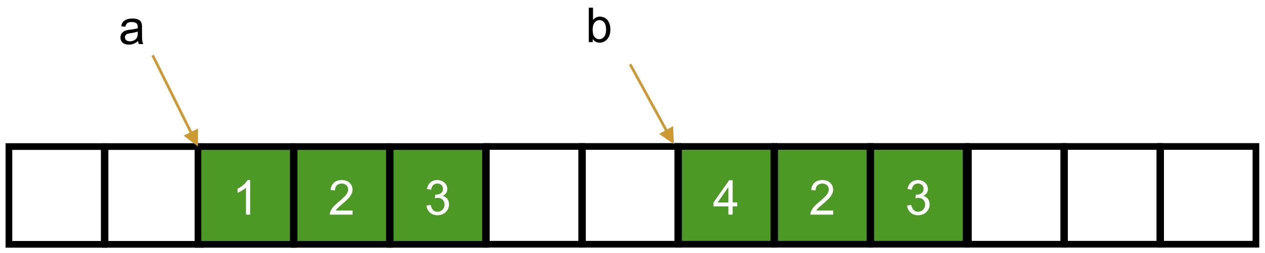
```
[1 2 3]
```

```
b[0] = 4
print(b)
```

```
[4 2 3]
```

```
print(a)
```

```
[1 2 3]
```



Use 'copy' function to create a deep copy of an array.

Questions?

Indexing Multi-Dimensional Arrays in Numpy

- Indexing single element
- Indexing by slicing
- Boolean Indexing
- Using `numpy.ix_` function

Thanks!