NLP 南京大学 自然语言处理研究组 Natural Language Processing Group

南京大学 NANJING UNIVERSITY

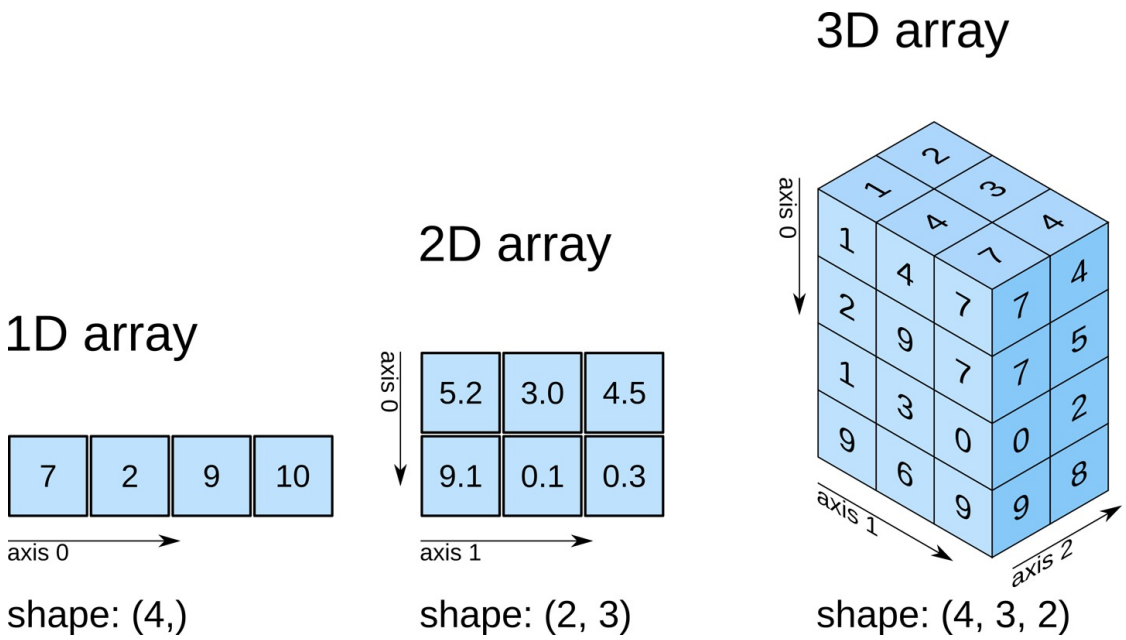
Numpy及其应用

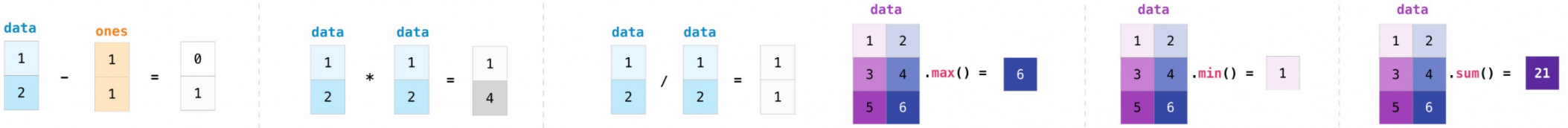
黄书剑

NumPy

- **NumPy is the fundamental package for scientific computing in Python.**
  - **a multidimensional array object**
  - various derived objects (such as masked arrays and matrices)
  - an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

multi-dimensional array





https://numpy.org/doc/stable/user/absolute_beginners.html

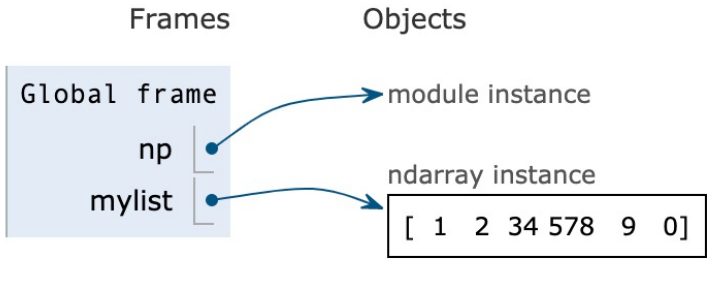
NumPy ndarray v.s. Python List

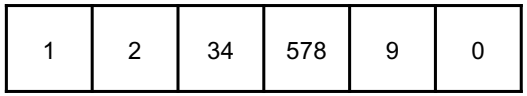
- **目标：更高的计算效率**

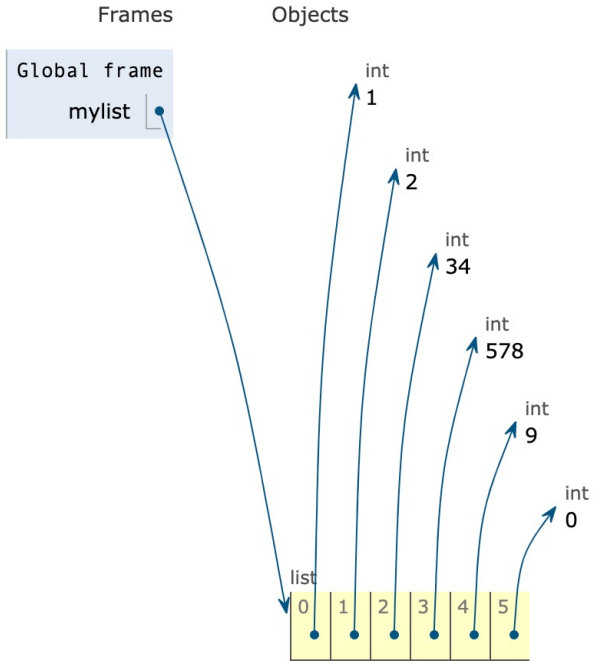
- **更高效的数据组织**
  - 同类型元素 + 固定长度
- **更高效的计算方式**
  - 向量化编程 v.s. 循环
- **实现和底层支持**
  - 基于C的实现
  - 利用Basic Linear Algebra Subprograms, BLAS
    - Intel MTK、 Open BLAS、 CUDA等

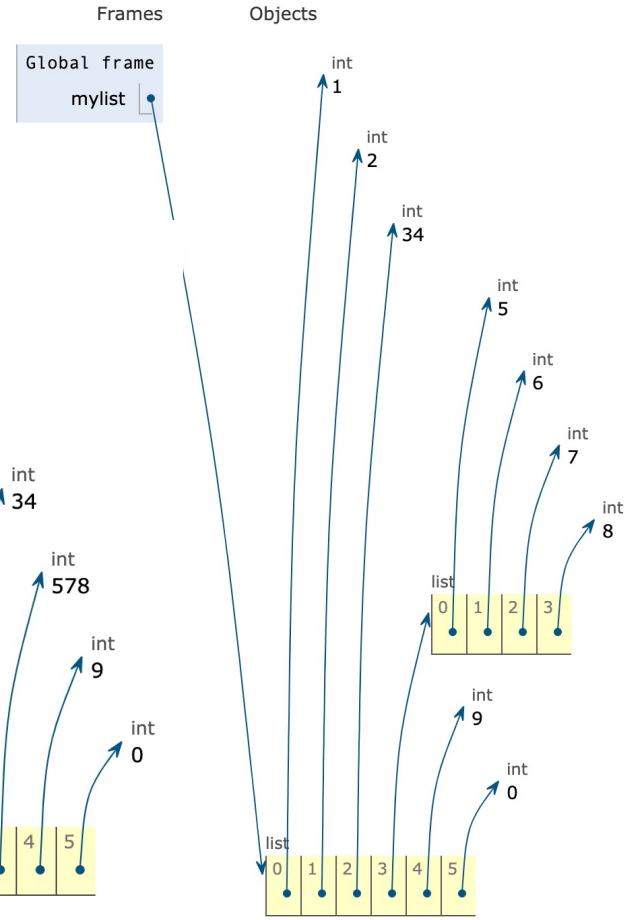
NumPy Array v.s. Python List

- **数据组织上的差异**
  - 更贴近C的数组实现（连续的内存空间）
  - 同质结构（元素类型相同）







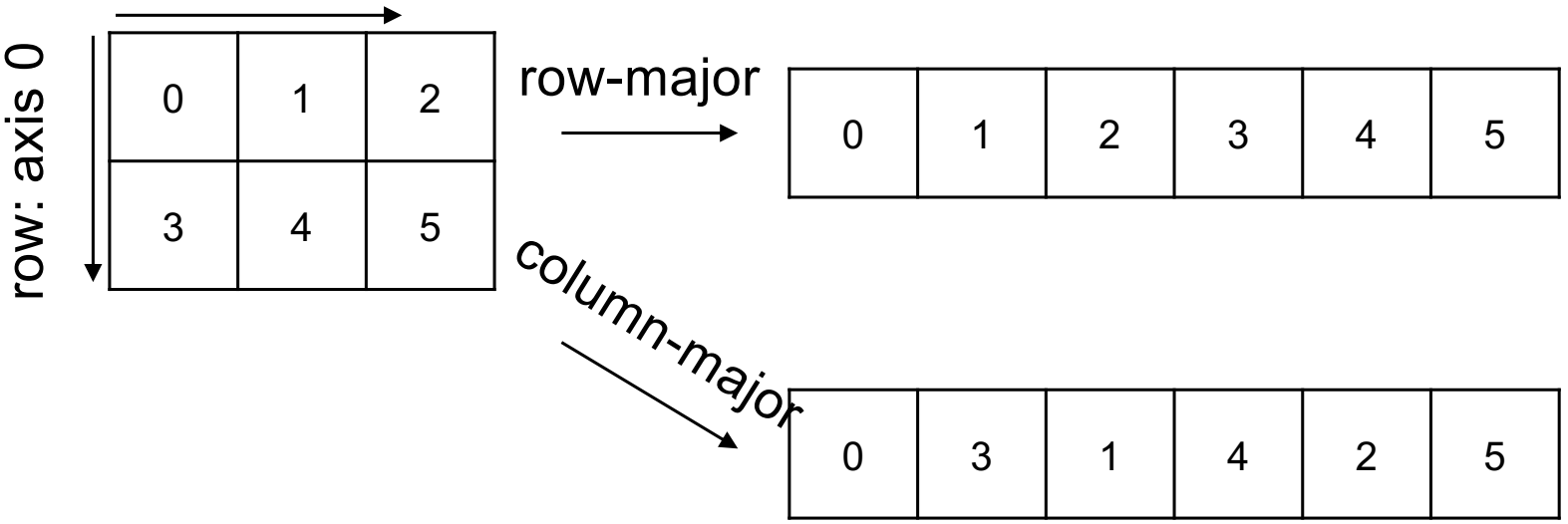



```
mylist = np.array([1, 2, 34, 578, 9 ,0])
```

数组的存储表示

- **行主序 v.s. 列主序**

C++:
int matrix[2][3]

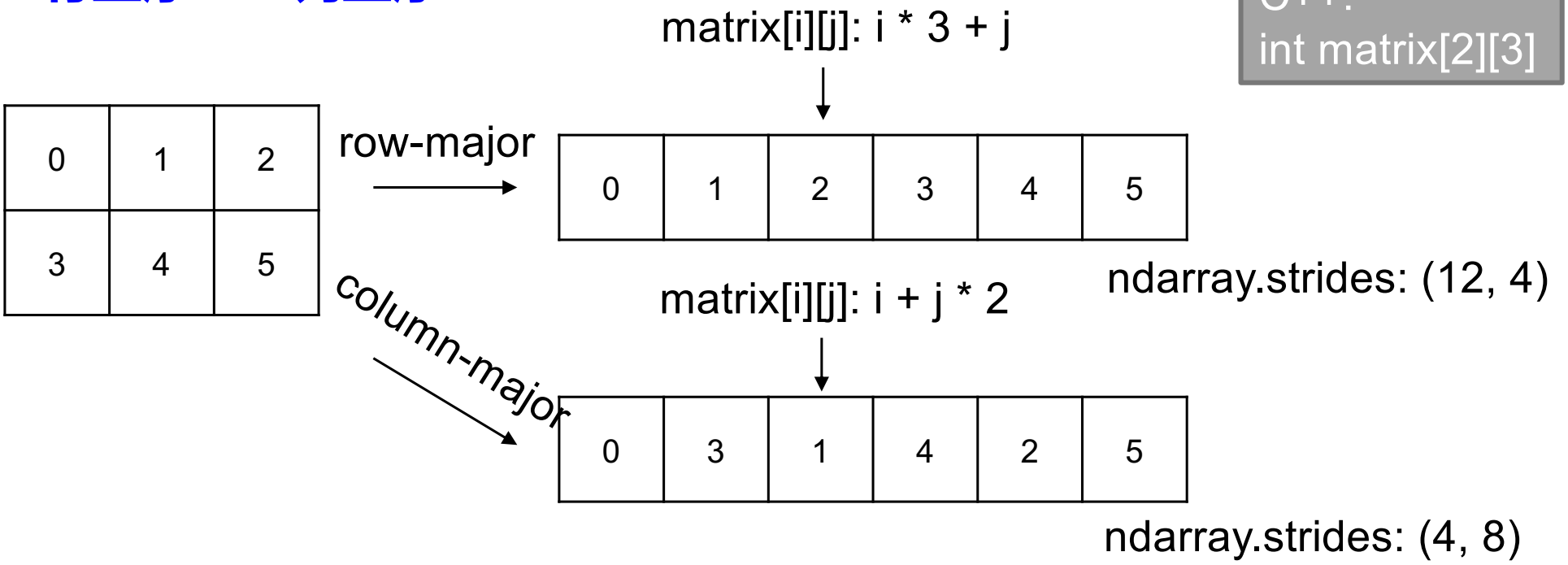


行主序是C等语言的格式，列主序是Fortran等语言的格式

数组的存储表示

- **行主序 v.s. 列主序**

C++:
int matrix[2][3]

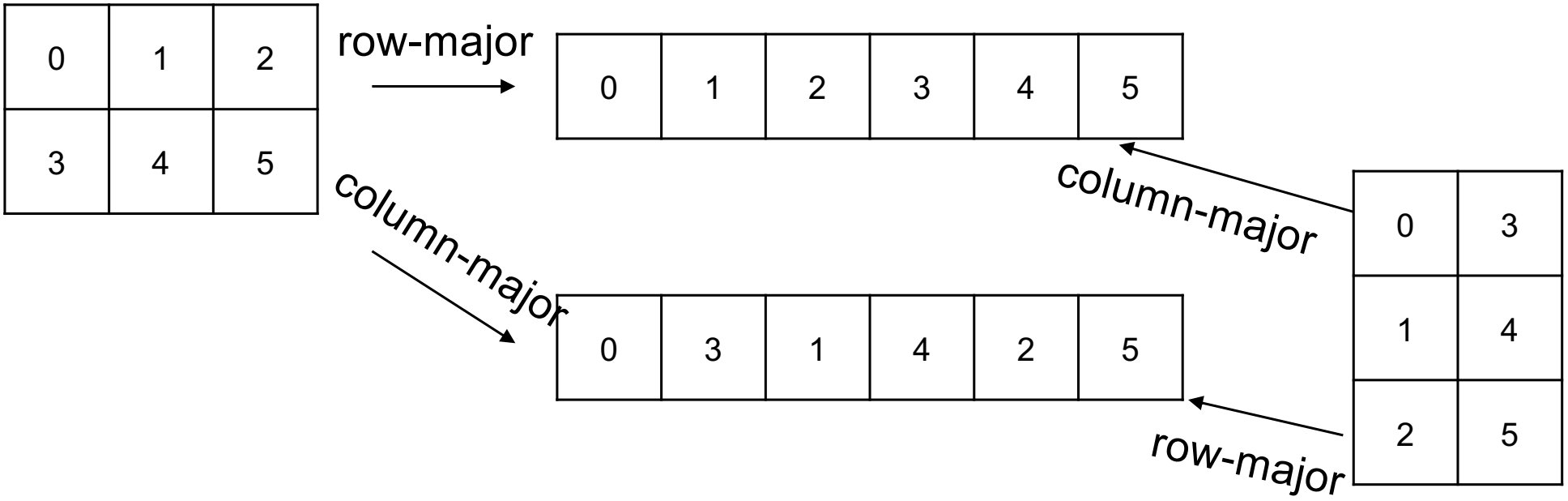


strides: 表示高维数组每一维对应的下标每次偏移时，在一维数组的存储中发生的偏移量(此处假设元素为int32，4个字节)。

数组的存储表示

- **行主序 v.s. 列主序**

C++:
int matrix[2][3]



不用改变数据表示，仅需要改变访问方式，即可完成转置！

numpy中的转置

```
import numpy as np #默认引用和别名，后续大多省略
```

```
def print_array(arr):
    print(arr.dtype)
    print(arr)
    print(arr.strides)
    print()
myarray = np.array([[1,2,3],[4,5,6]])
print_array(myarray)
```

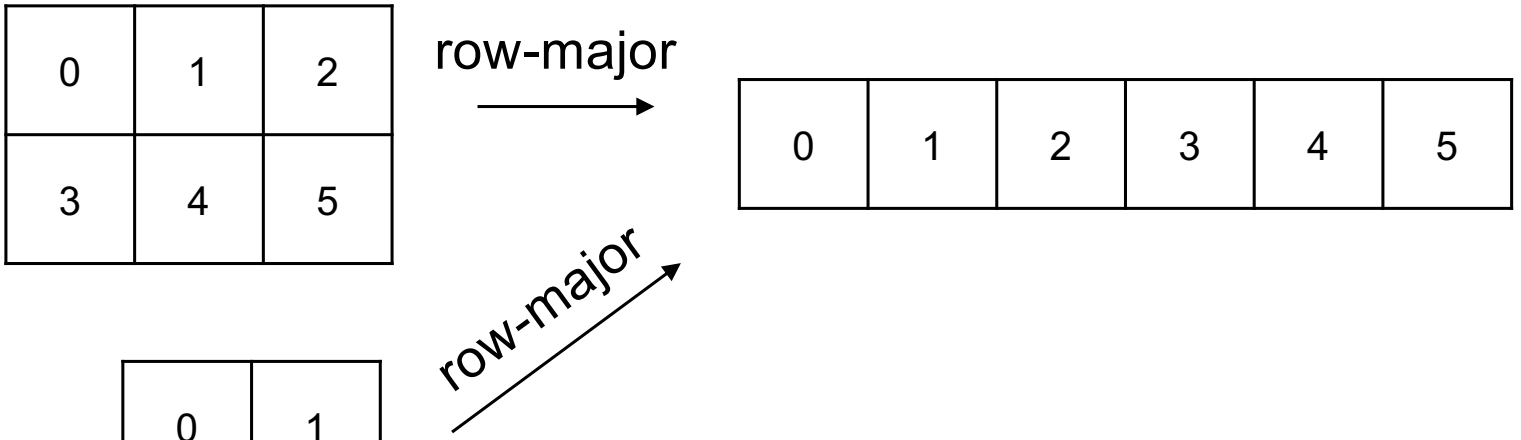
```
myarrayT = myarray.T
print_array(myarrayT)
```

```
myarray2 = np.array([[1,4],[2,5],[3,6]])
print_array(myarray2)
```

同样的数组，不一样的内部表示！

数组的存储表示

C++:
int matrix[2][3]



同样的内部表示，不一样的外部形式！

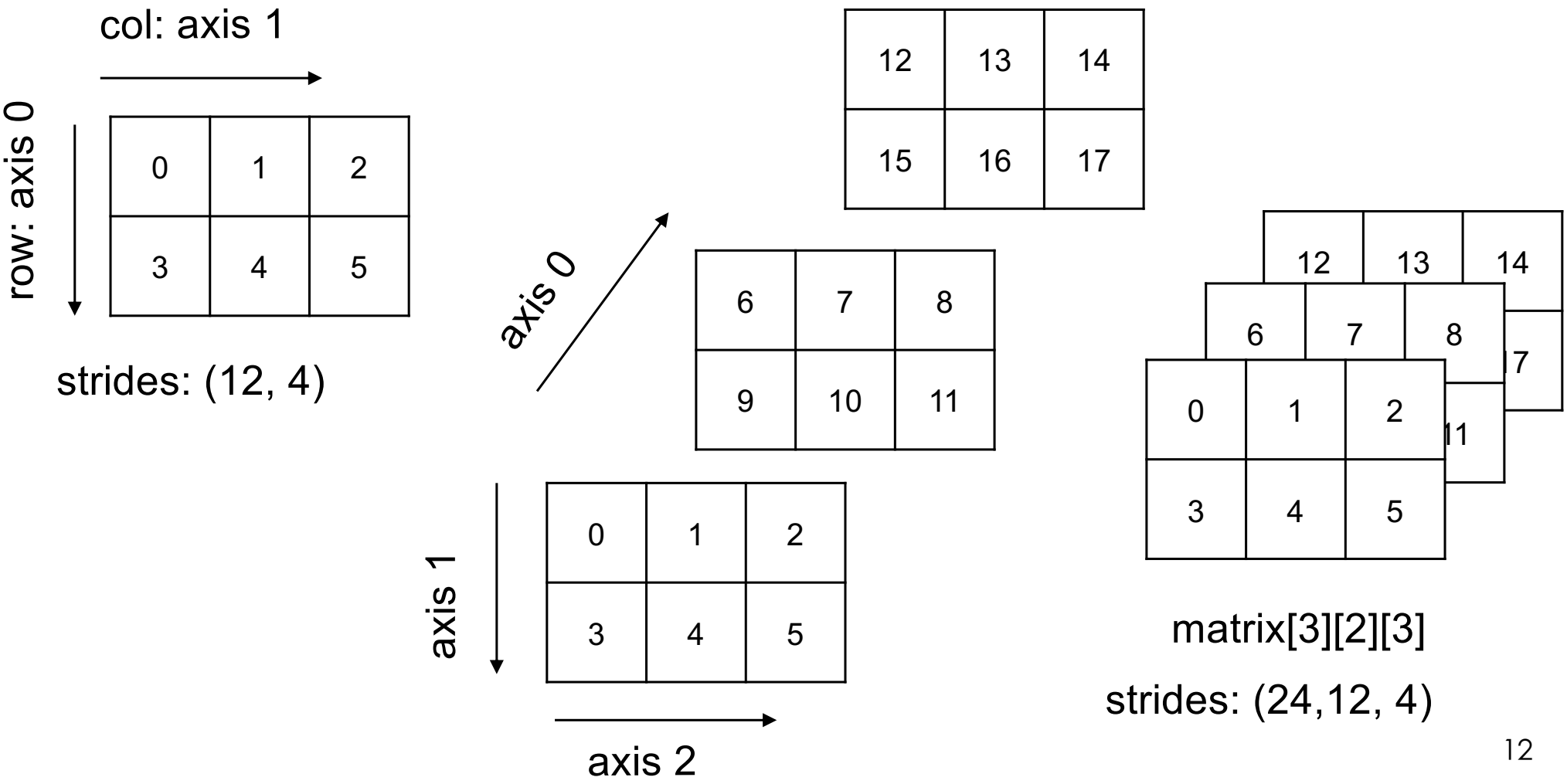
numpy中改变数组的形状

```
data = np.arange(1, 7)
print_array(data)

data1 = np.arange(1, 7).reshape((3,2))
print_array(data1)

data2 = np.arange(1, 7).reshape((2,3))
print_array(data2)
```

同样的数组，不一样的内部表示！
同样的内部表示，不一样的外部形式！

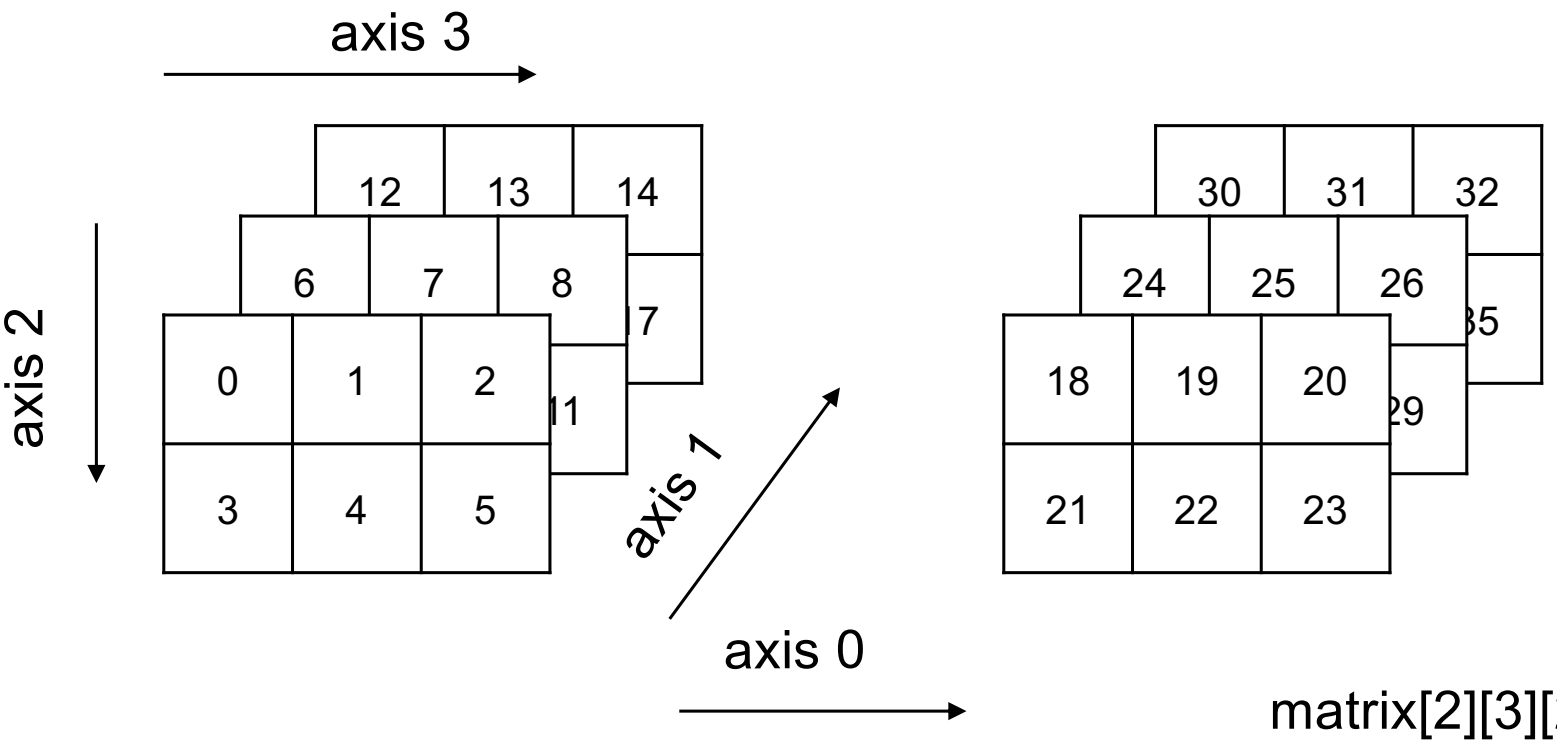
抽象和封装：数据内部表示 v.s. 数据的外部使用

高维数组


col: axis 1
row: axis 0
0 1 2
3 4 5
strides: (12, 4)
axis 0
12 13 14
15 16 17
6 7 8
9 10 11
axis 1
0 1 2
3 4 5
axis 2
12 13 14
6 7 8
0 1 2
3 4 5
matrix[3][2][3]
strides: (24,12, 4)

高维数组

Tensor



matrix[2][3][2][3]

strides: (72, 24,12, 4)

```
data = np.arange(1, 19)
print_array(data)

data1 = np.arange(1, 19).reshape((3,2,3))
print_array(data1)

data2 = np.arange(1, 19).reshape((2,3,3))
print_array(data2)
```

试着自己实现一个n维矩阵类，创建任意维度的矩阵，提供元素访问操作、转置操作、reshape操作等功能。

ndarray的基本属性

- **dtype：元素类型**
- **ndim：维数**
- **shape：数组形状**
  - 整数构成的元组，如：(2, 3), (1, 4)等
  - 向量的形状是元素仅有1个的元组，须加逗号区分，如: (3,)

- **size：所有元素个数**
- **itemsize：单个元素大小（bytes）**
- **strides：访问规则**
- **……**

```
Attributes
----------
T : ndarray
    Transpose of the array.
data : buffer
    The array's elements, in memory.
dtype : dtype object
    Describes the format of the elements in the array.
size : int
    Number of elements in the array.
itemsize : int
    The memory use of each array element in bytes.
ndim : int
    The array's number of dimensions.
shape : tuple of ints
    Shape of the array.
strides : tuple of ints
    The step-size required to move from one element to the next in
    memory. For example, a contiguous ``(3, 4)`` array of type
    ``int16`` in C-order has strides ``(8, 2)``.  This implies that
    to move from element to element in memory requires jumps of 2 bytes.
    To move from row-to-row, one needs to jump 8 bytes at a time
    (``2 * 4``).
```

```
help(np.ndarray)
np.ndarray?
```

南京大學
NANJING UNIVERSITY

NDARRAY的创建

ndarray的创建

- **从已有列表创建**
  - 使用np.array函数

查阅 np.array?

```
array(object, dtype=None, copy=True, order='K', subok=False, ndmin=0)
```

```
In [62]: np.array([1, 2, 3, 4])
Out[62]: array([1, 2, 3, 4])

In [63]: a1 = np.array([1, 2, 3, 4])
Out[63]: a2 = np.array(a1, order = 'F', ndmin = 2)
```

ndarray的创建

- **从已有列表创建**
- **数值自动填充**
  - zeros, ones, empty, full …
  - zeros_like, ones_like, empty_like, full_like …
  - identity, eye
  - diag, arrange
  - linspace, logspace, meshgrid
  - fromfunction, fromfile
  - np.random.rand

```
In [58]: np.zeros(2)
Out[58]: array([0., 0.])

In [59]: np.ones(3)
Out[59]: array([1., 1., 1.])

In [60]: np.diag(range(1,5))
Out[60]:
array([[1, 0, 0, 0],
       [0, 2, 0, 0],
       [0, 0, 3, 0],
       [0, 0, 0, 4]])

In [69]: np.arange(1,6,2)
Out[69]: array([1, 3, 5])

In [71]: np.linspace(0, 10, num=5)
Out[71]: array([ 0. ,  2.5,  5. ,  7.5, 10. ])
```

more examples on examples_numpy.ipynb

NumPy支持的数据类型

- **各种字长的5种基础数据类型**
  - booleans, integers, unsigned integers, floating point, complex
  - 例如: bool_ , int8, int16, int32, float32, uint64, complex64 等
- **更精确指定数据类型，以获取存储和计算上的性能提升**
- **NumPy也可以用于支持自定义类型（Structured arrays）**
  - https://numpy.org/doc/stable/user/basics.rec.html#structured-arrays

南京大學
NANJING UNIVERSITY

索引、切片、视图

从NDARRAY创建NDARRAY

索引和切片

- **定位单个元素**
  - 下标（正值、负值）
- **选择多个元素**
  - 切片（（start, end, step）序列开始、序列结尾）

- **对于高维数组，须从0-ndim指明每个需要切片的维度**
- **每个维度的索引和切片操作相互独立**
  - `myarray[-3:,-3:]`

列表切片（slice）

- 语法说明

```
newlist = listA[start : end : step]
```

开始位置　结束位置　步长

- 结束位置元素不在结果列表中
- 步长用于跳过部分元素
- start、end 可以省略，分别表示从列表开始、直到列表结束
- step可以省略，表示默认步长为1

```
In [7]: myarray = np.arange(100).reshape(10, 10)
In [8]: myarray
Out[8]:
array([[ 0,  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, 28, 29],
    ...


In [9]: myarray[: ,1]

In [10]: myarray[-3:,-3:]

In [11]: myarray[:,-3:]

In [12]: myarray[-3:]
```

切片和视图

- **切片操作得到的数据子集是原数组的一种展示方式（称为视图）**
  - 视图仍然具有正常数组的行为、效率得到提升
  - 视图中的改动将影响原数组数据

```
In [19]: newarray =
myarray[::5,-3:]
    ...: print_array(newarray)

Element Type:  int64
Shape:     (2, 3)
Strides:   (400, 8)
[[ 7  8  9]
 [57 58 59]]
```

```
In [20]: newarray2 =
np.array([[ 7, 8, 9], [57, 58,
59]], dtype="int32")
    ...:
print_array(newarray2)

Element Type:  int32
Shape:     (2, 3)
Strides:   (12, 4)
[[ 7  8  9]
[57 58 59]]
```

抽象和封装：数据内部表示 v.s. 数据的外部使用

```
In [21]: newarray3 = newarray[:,-2:]
```

创建切片的切片

```
In [22]: newarray3.fill(0)
```

将切片数组置零

```
In [23]: print(newarray3)
    ...: print(newarray)
    ...: print(myarray)
```

```
[[0 0]
 [0 0]]
[[ 7  0  0]
 [57  0  0]]
[[ 0  1  2  3  4  5  6  7  0  0]
 [10 11 12 13 14 15 16 17 18 19]
 [20 21 22 23 24 25 26 27 28 29]
 [30 31 32 33 34 35 36 37 38 39]
 [40 41 42 43 44 45 46 47 48 49]
 [50 51 52 53 54 55 56 57  0  0]
 [60 61 62 63 64 65 66 67 68 69]
 [70 71 72 73 74 75 76 77 78 79]
 [80 81 82 83 84 85 86 87 88 89]
 [90 91 92 93 94 95 96 97 98 99]]
```

原数组也被置零，除非显式指明copy：

```
newarray3 = newarray[:,-2:].copy()
```

试着自己实现一个三维矩阵类，让它能进行多次slicing操作，操作结果是原矩阵的一个视图。

fancy indexing & boolean indexing

- **花式索引**
  - 使用一个索引序列从ndarray中选择元素，并创建新数组

```
In [24]: myarray = np.arange(0, 100, 10)
    ...: indices = [1, 5, -1]
    ...: newarray = myarray[indices]
```

- **布尔索引**
  - 使用一个bool序列进行元素选择，并创建新数组

```
In [25]: myarray = np.arange(8)
    ...: b = [False,  True,  False, False, False, False,  True, False]
    ...: myarray[b]
Out[25]: array([1, 6])
```

https://jakevdp.github.io/PythonDataScienceHandbook/02.07-fancy-indexing.html

南京大学
NANJING UNIVERSITY

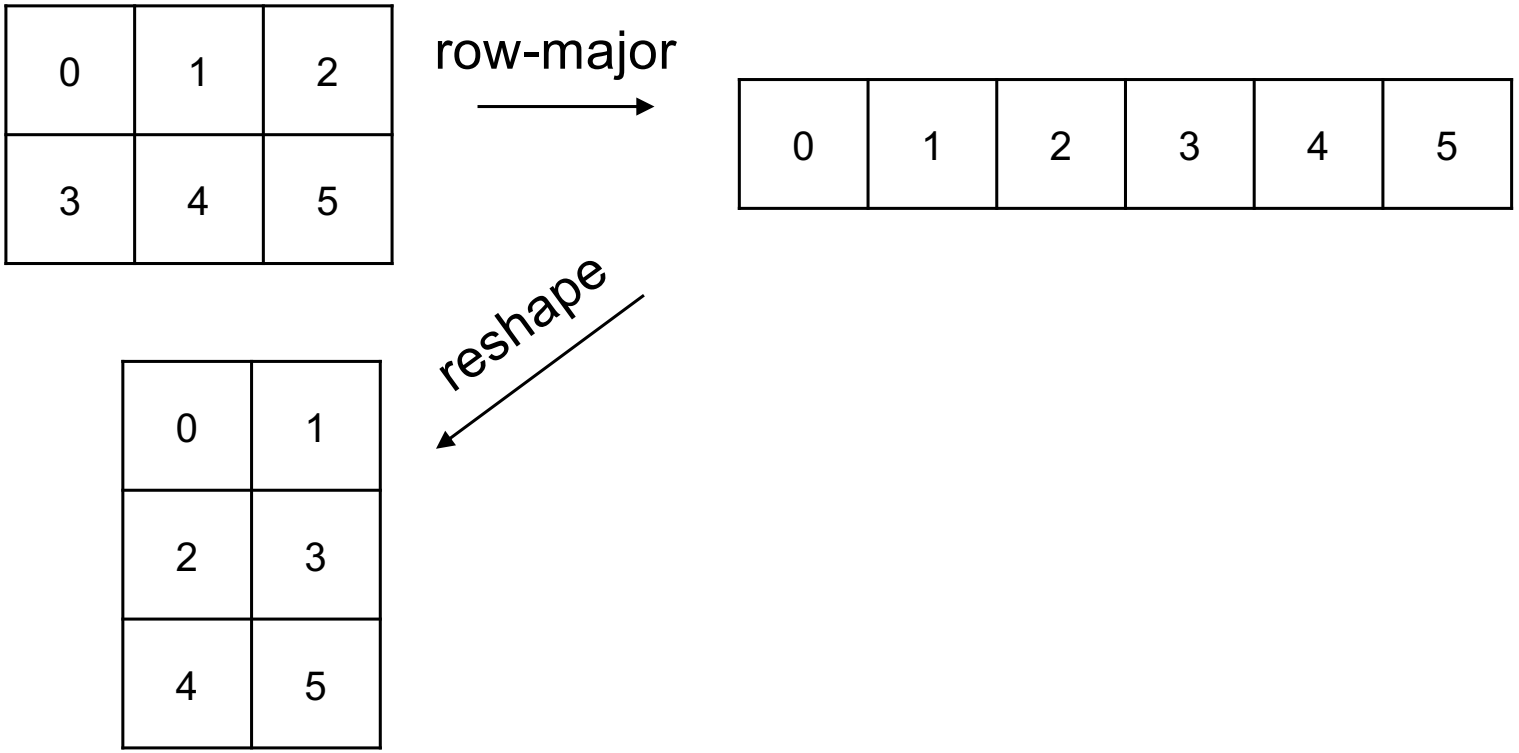
调整数组形状和值

改变数组维度

- **基于底层表示支持上层的高效实现（一般返回视图）**
  - np.reshape/np.ndarray.reshape
  - np.ravel/np.ndarry.ravel（折叠为一维数组）
  - np.squeeze（删除长度为1的维度）
  - np.expand_dims和np.newaxis（增加新的维度）
  - np.transpose/np.ndarray.transpose/np.ndarray.T

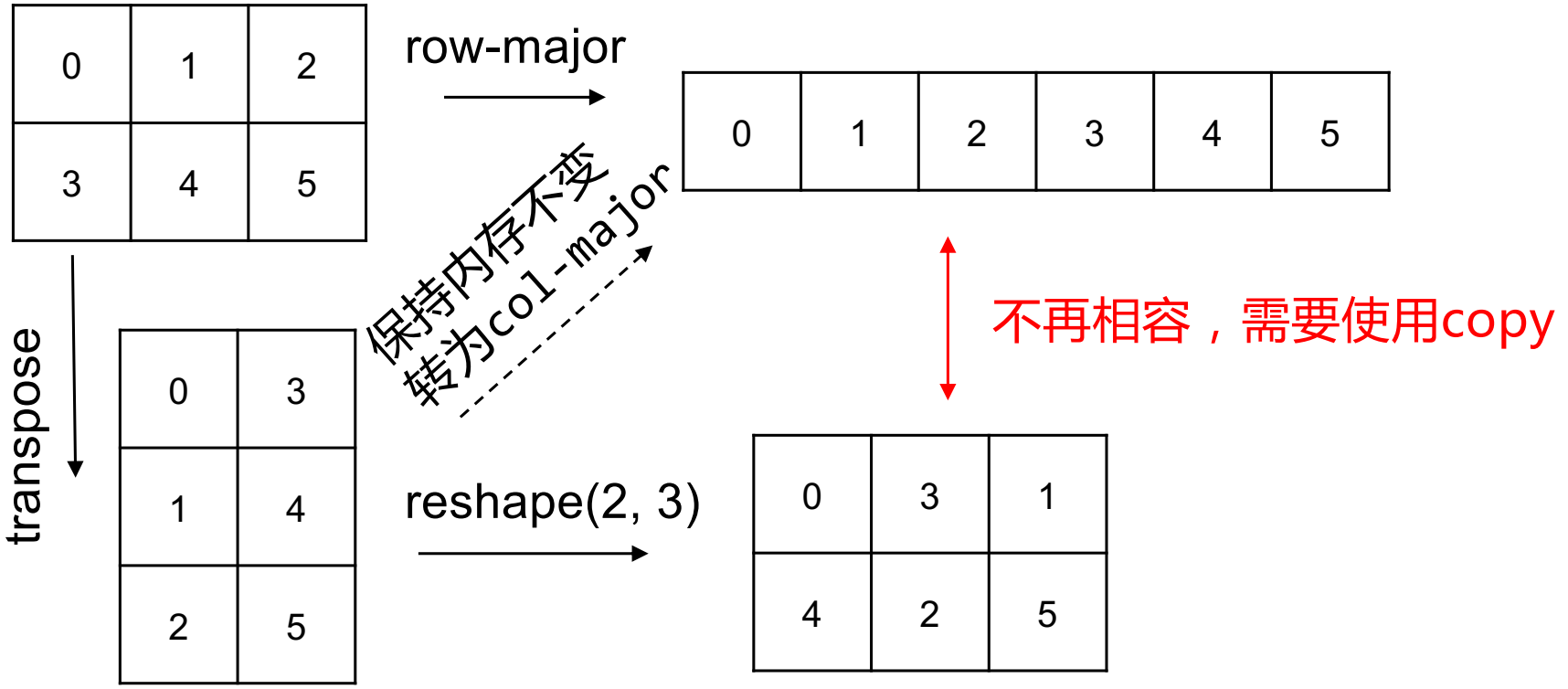
改变形状与内存布局

- **reshape默认按照行主序进行（C style）**
  - 对于以行主序存储的数组来说，不需要改变内存布局



改变形状与内存布局

- **reshape默认按照行主序进行（C style）**
  - 对于以行主序存储的数组来说，不需要改变内存布局
  - 如果转置后，再进行reshape，则难以使用原内存布局



改变形状与内存布局

- **reshape默认按照逻辑上的行主序进行（C style）**
  - 对于以行主序存储的数组来说，不需要改变内存布局
- **reshape时可以指定顺序（C style or F style or Automatic）**
  - A参数根据内存布局自动选择，此时reshape可能有不同结果

```
In [122]: x = np.arange(6).reshape(2,3)
array([[0, 1, 2],
       [3, 4, 5]])

In [124]: x.reshape(3, 2, order = 'F’)
Out[124]:
array([[0, 4],
       [3, 2],
       [1, 5]])
```

```
In [125]: x.reshape(3, 2, order = 'A')
Out[125]:
array([[0, 1],
       [2, 3],
       [4, 5]])

In [126]: x.copy(order = 'F').reshape(3, 2,
order = 'A’)
Out[126]:
array([[0, 4],
       [3, 2],
       [1, 5]])
```

改变数组大小、形状、内容

- **一般将会按照要求创建新的数组**
- **改变形状**
  - np.resize
  - np.ndarray.flatten
- **改变元素顺序**
  - np.rot90, np.fliplr, np.flipud, np.sort
- **堆叠**
  - np.hstack, np.vstack, np.dstack, np.concatenate
- **修改单个元素**
  - np.append, np.insert, np.delete