CSC 561: Neural Networks and Deep Learning

Preliminaries

Marco Alvarez

Department of Computer Science and Statistics
University of Rhode Island

Spring 2024

Linear Algebra

Subset of most relevant concepts to Deep Learning

Credit: Deep Learning by Ian Goodfellow, Yoshua Bengio, Aaron Courville

Scalars and vectors

- Scalars
  - integers, real numbers, rational numbers, etc.
  - usually denoted by lowercase letters
- Vectors
  - 1-D array of elements (scalars)

$$\mathbf{x} \in \mathbb{R}^n \qquad \boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

Matrices and tensors

- Matrices
  - 2-D array of elements

$$\begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} \quad A \in \mathbb{R}^{m \times n}$$

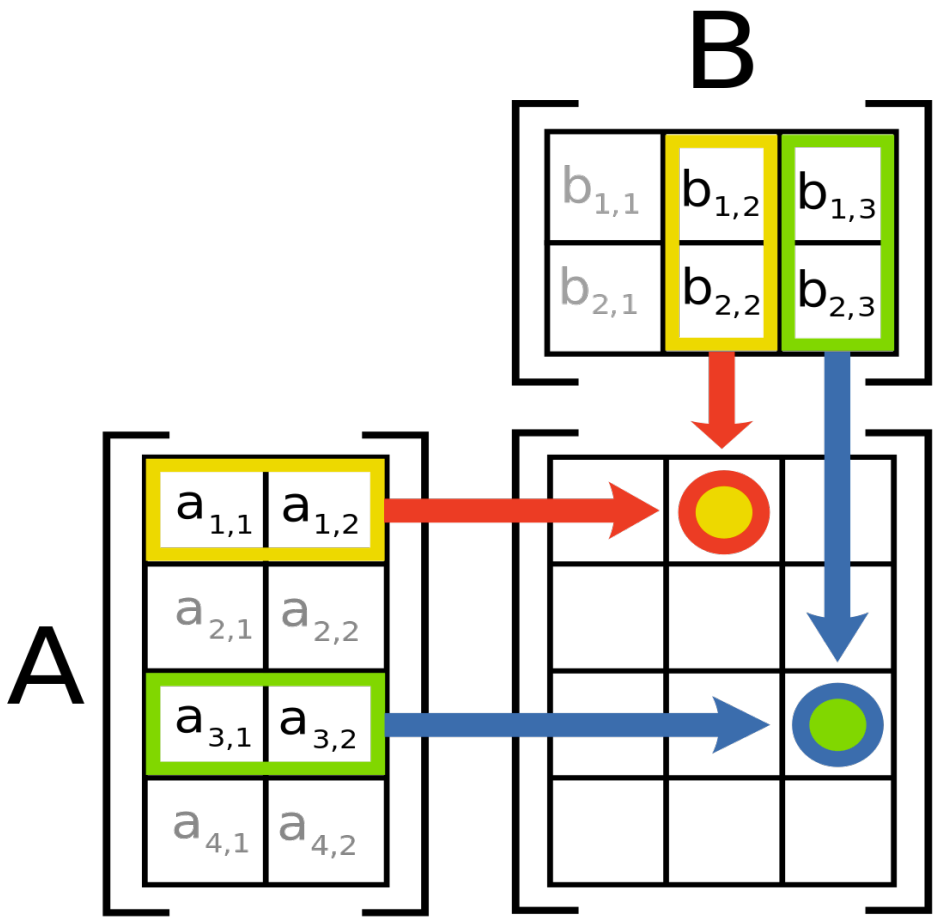
- Tensors
  - homogeneous arrays that may have zero (scalar) or more dimensions

Basics

- Matrix Multiplication

$$C = AB$$

$$C_{i,j} = \sum_{k} A_{i,k} B_{k,j}$$



number of columns in A must be equal to the number of rows in B

https://en.wikipedia.org/wiki/Matrix_multiplication

Dot product

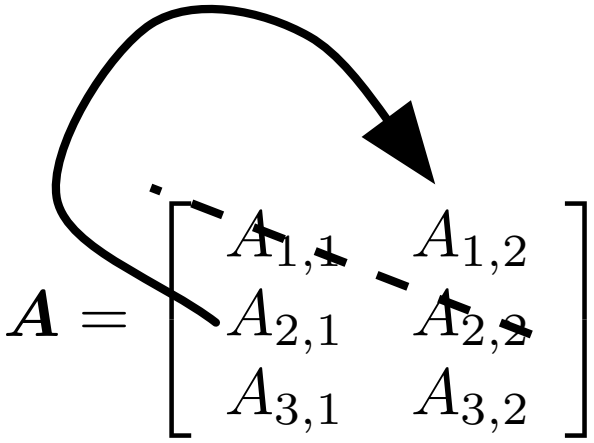
$$[x_1 \quad x_2 \quad \cdots \quad x_n] \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

$$\mathbf{x}^T\mathbf{y} = \sum_{i=1}^{n} x_i y_i$$

$$\mathbf{x}^T\mathbf{y} = \|\mathbf{x}\|\|\mathbf{y}\| \cos\theta$$

$$\mathbf{x}^T\mathbf{y} \in \mathbb{R}$$

Matrix transpose



$$\boldsymbol{A} = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \\ A_{3,1} & A_{3,2} \end{bmatrix} \Rightarrow \boldsymbol{A}^\top = \begin{bmatrix} A_{1,1} & A_{2,1} & A_{3,1} \\ A_{1,2} & A_{2,2} & A_{3,2} \end{bmatrix}$$

$$\left(A^T\right)_{i,j} = A_{j,i} \qquad (AB)^T = B^T A^T$$

Identity matrix

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\forall x \in \mathbb{R}^n, I_n x = x$$

$$A^{-1}A = I_n$$

Norms

- Functions that measure the magnitude ("length") of a vector
  - strictly positive, except for the zero vector
  - think about the distance between zero and the point represented by the vector

$$f(\mathbf{x}) = 0 \Rightarrow \mathbf{x} = \mathbf{0}$$

$$f(\mathbf{x} + \mathbf{y}) \leq f(\mathbf{x}) + f(\mathbf{y}) \text{ (triangle inequality)}$$

$$\forall \alpha \in \mathbb{R}, f(\alpha \mathbf{x}) = |\alpha| f(\mathbf{x})$$

Norms

ℓ_1-norm: $\|\mathbf{x}\|_1 = \sum_{i=1}^{n} |x_i|$

ℓ_2-norm: $\|\mathbf{x}\|_2 = \left(\sum_{i=1}^{n} x_i^2 \right)^{\frac{1}{2}}$

max norm: $\|\mathbf{x}\|_\infty = \max_i |\mathbf{x}_i|$

Special matrices and vectors

- Unit vector

$$\|\mathbf{x}\|_2 = 1$$

- Symmetric matrix

$$A = A^T$$

- Orthogonal matrix

$$A^T A = A A^T = I$$

$$A^{-1} = A^T$$

Programming with Tensors

Non-negotiable (Python)

- Basic data types
  - booleans, integers, floating point values, strings
- Control flow
- Built-in data structures
  - lists, tuples, dictionaries, sets
  - iterators
- Functions
  - functions can be assigned, passed, returned, and stored
  - lambda functions (inline and anonymous)
- Classes

Numpy

- Library for scientific computing
  - provides a high-performance multidimensional array object and routines for fast operations on these arrays
- The `ndarray` object encapsulates n-dimensional arrays of homogeneous data types
  - many operations performed in compiled code for higher performance
  - have a fixed size at creation
    - changing the size of an ndarray will create a new array and delete the original

```
n = 100000000
array1 = np.random.rand(n)
array2 = np.random.rand(n)
```

```
def dot_python(x, y):
    sum = 0
    for i in range(len(x)):
        sum += x[i] * y[i]
    return sum

def dot_numpy(x, y):
    return np.dot(x, y)
```

```
time_taken = timeit.timeit(lambda: dot_numpy(array1, array2), number=1)
print(f'Numpy Time: {time_taken} seconds')
```

```
Numpy Time: 0.05994947799996453 seconds
```

```
array1 = array1.tolist()
array2 = array2.tolist()
time_taken = timeit.timeit(lambda: dot_python(array1, array2), number=1)
print(f'Python Time: {time_taken} seconds')
```

```
Python Time: 8.325287125999978 seconds
```

The following figures are from:

"*NumPy Illustrated: The Visual Guide to NumPy*" by Lev Maximov

https://betterprogramming.pub/numpy-illustrated-the-visual-guide-to-numpy-3b1d4976de1d

Elegant code

In [3]:
```
a = [1, 2, 3]
[q*2 for q in a]
```

Out[3]: [2, 4, 6]

In [4]:
```
a = np.array([1, 2, 3])
a * 2
```

Out[4]: array([2, 4, 6])

In [1]:
```
a = [1, 2, 3]
b = [4, 5, 6]
[q+r for q, r in zip(a, b)]
```

Out[1]: [5, 7, 9]

In [2]:
```
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
a + b
```

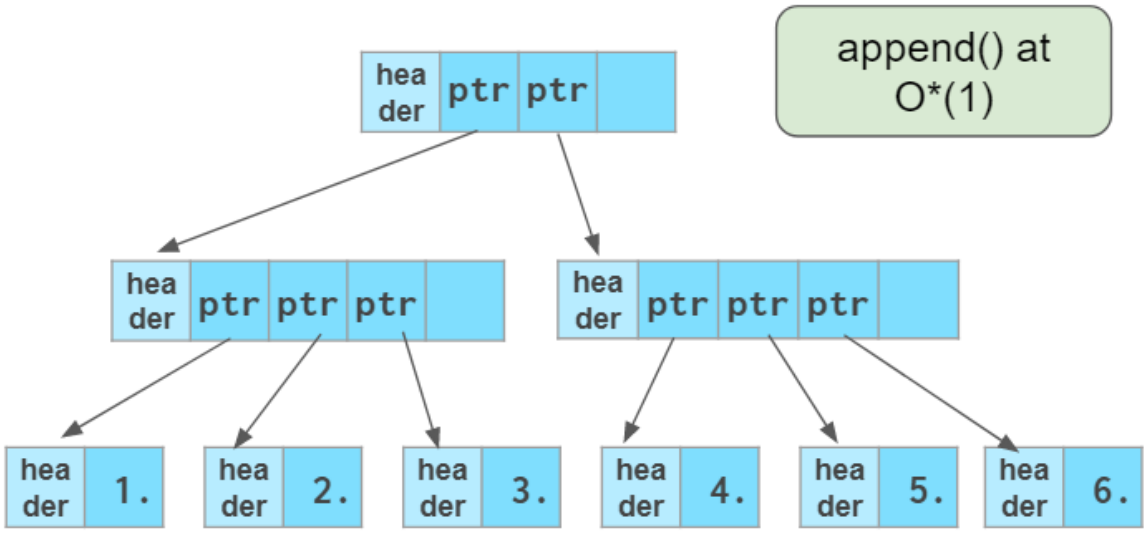
Out[2]: array([5, 7, 9])

Homogeneous and fixed-length

python list **vs** **numpy array**

1.	2.	3.
4.	5.	6.

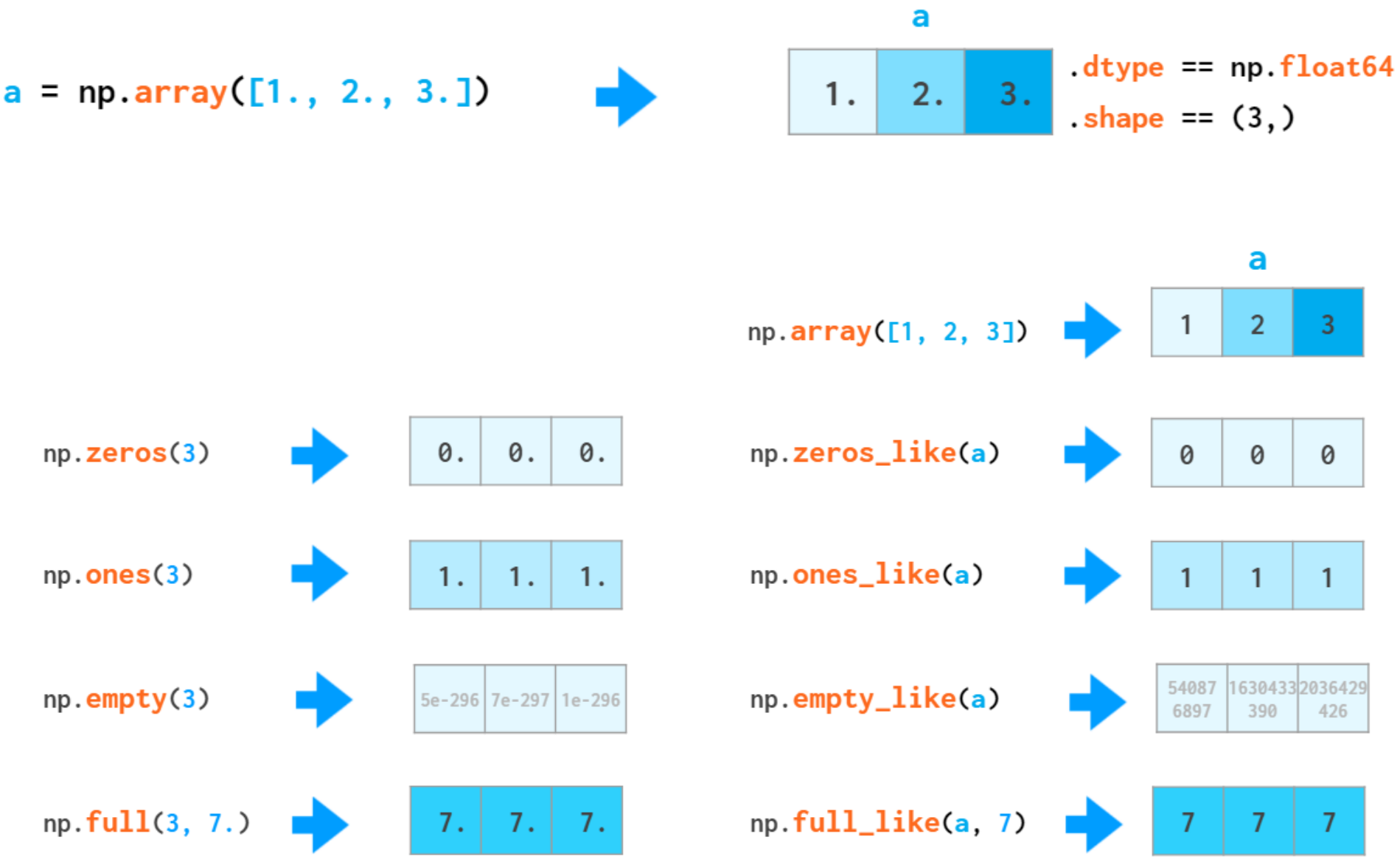
1.	2.	3.
4.	5.	6.



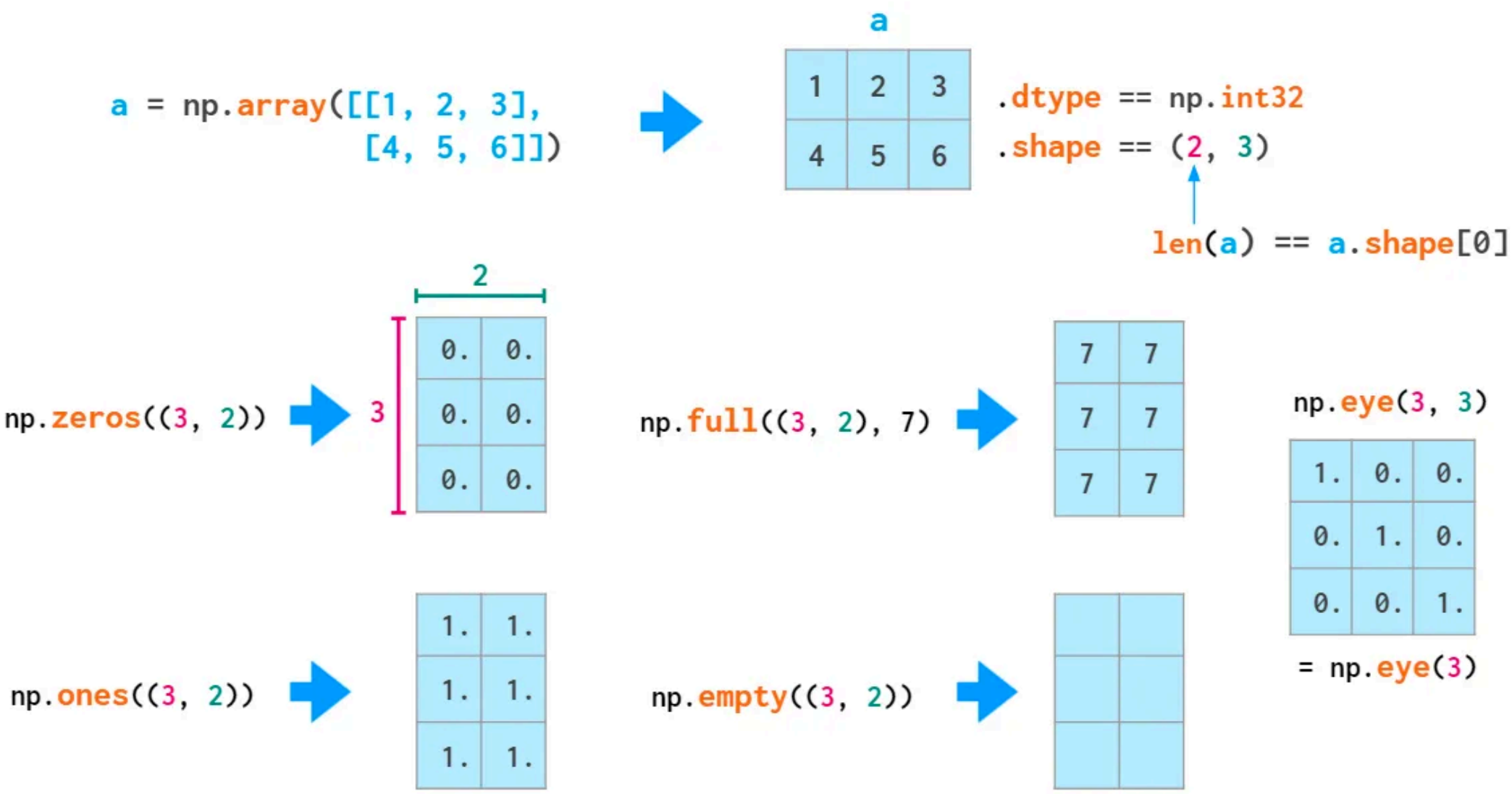
append() at O*(1)

append() at O(N)

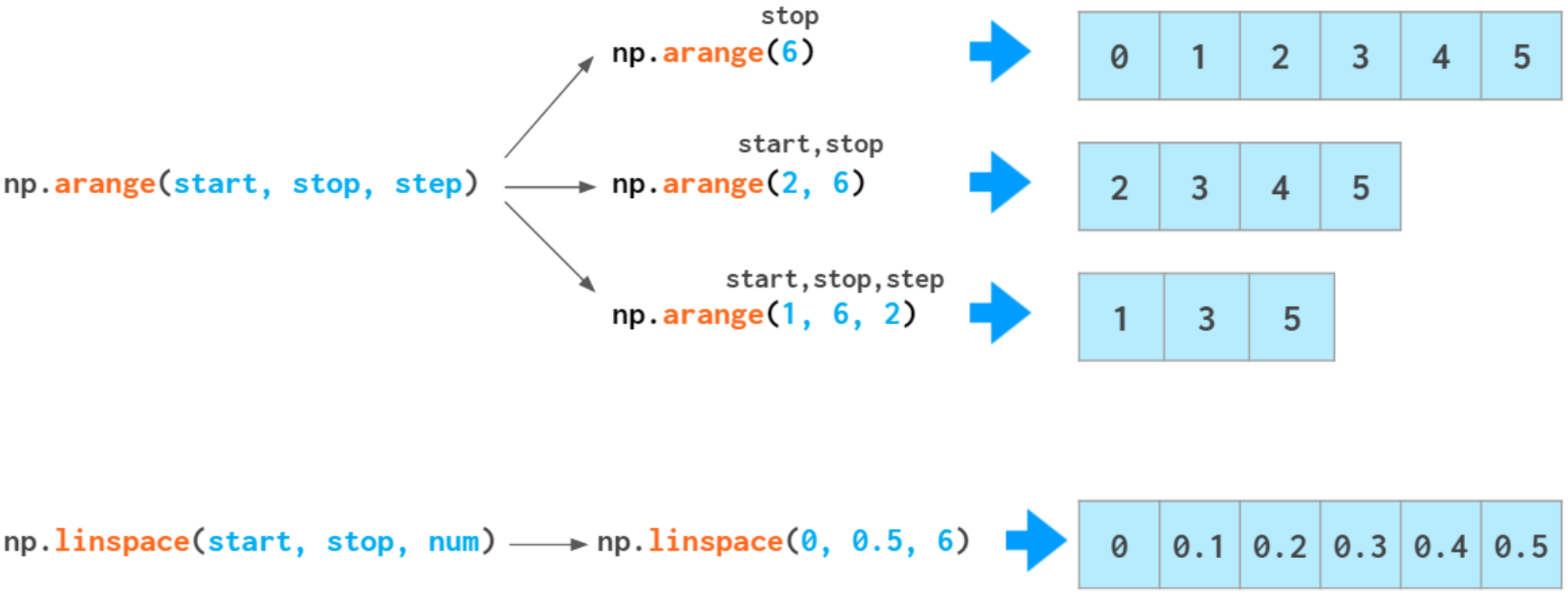
Creating numpy arrays



Creating numpy arrays



Creating numpy arrays



Creating numpy arrays

`np.random.randint(0, 10, 3)`

uniform, x ∈ [0, 10)

Careful!

`np.random.randint(0, 10)` is [0, 10), but

`random.randint(0, 10)` is [0, 10]

`np.random.rand(3)`

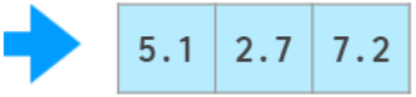
uniform, x ∈ [0, 1)

`np.random.randn(3)`

normal, μ=0, σ=1

`np.random.uniform(1, 10, 3)`

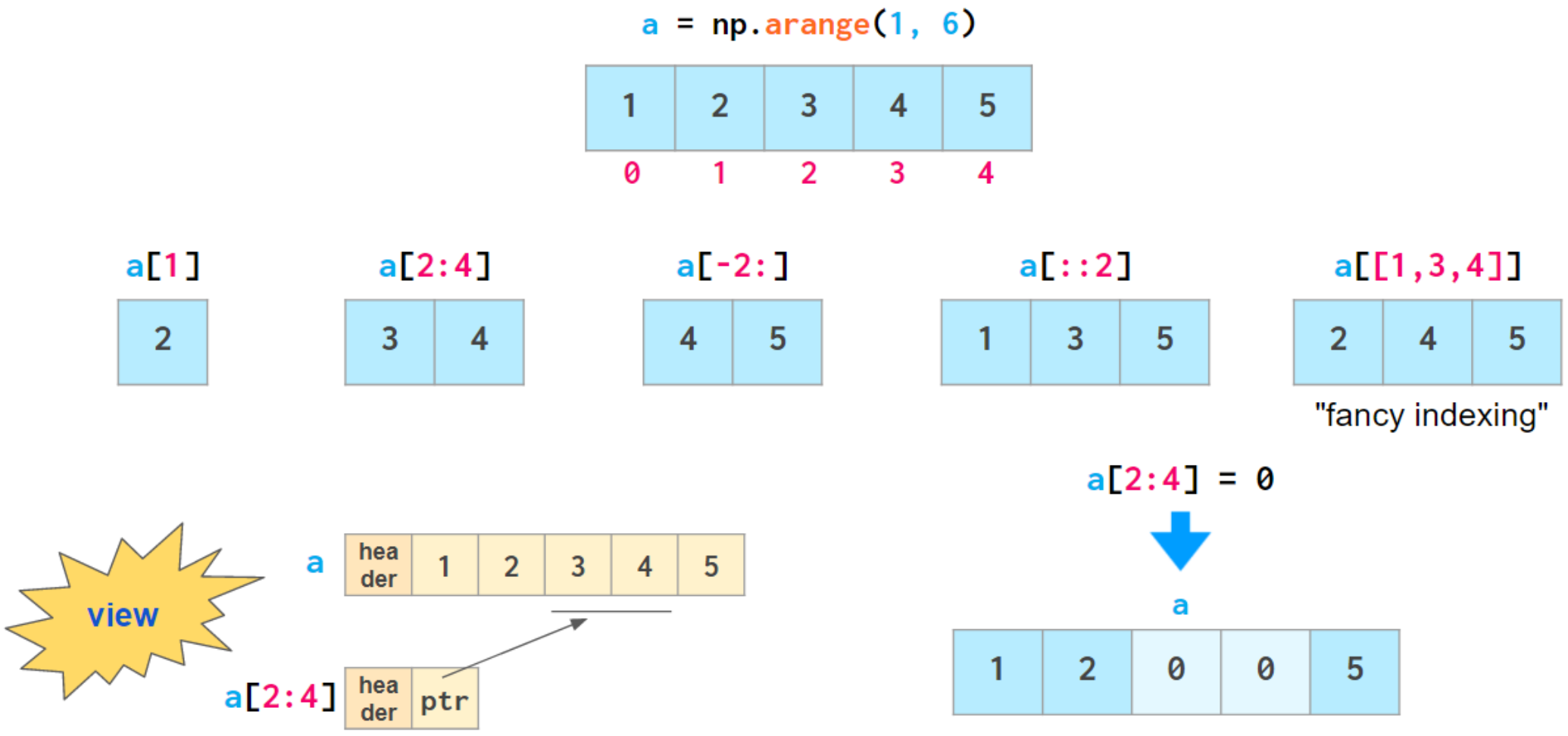
uniform, x ∈ [1, 10)



`np.random.normal(5, 2, 3)`

normal, μ=5, σ=2

Indexing


a = np.arange(1, 6)
1 2 3 4 5
0 1 2 3 4
a[1]
2
a[2:4]
3 4
a[-2:]
4 5
a[::2]
1 3 5
a[[1,3,4]]
2 4 5
"fancy indexing"
view
a
hea der
1 2 3 4 5
a[2:4]
hea der
ptr
a[2:4] = 0
a
1 2 0 0 5

Careful when making copies

python list

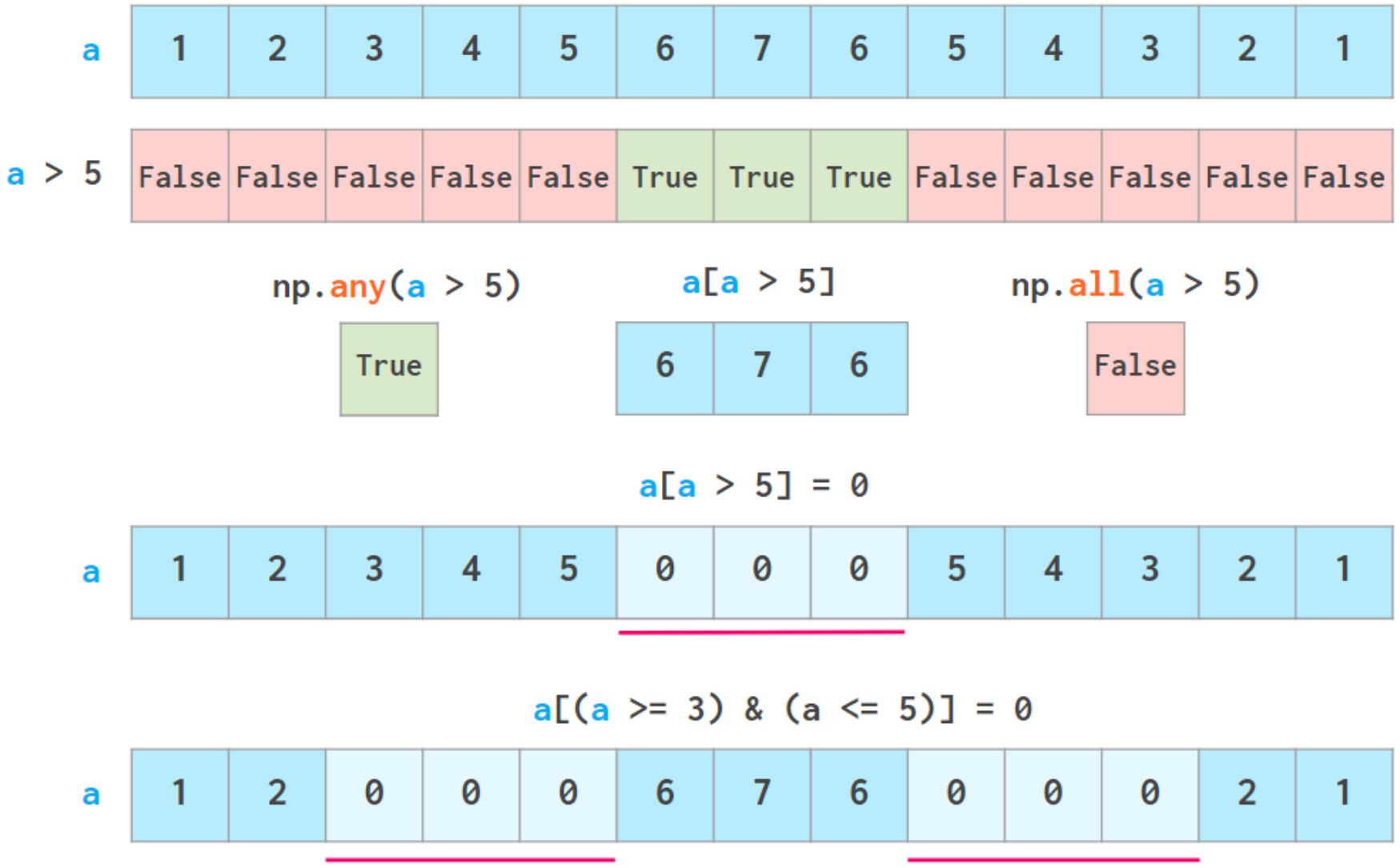
```
a = [1, 2, 3]
b = a          # no copy
c = a[:]       # copy
d = a.copy()   # copy
```

vs

numpy array

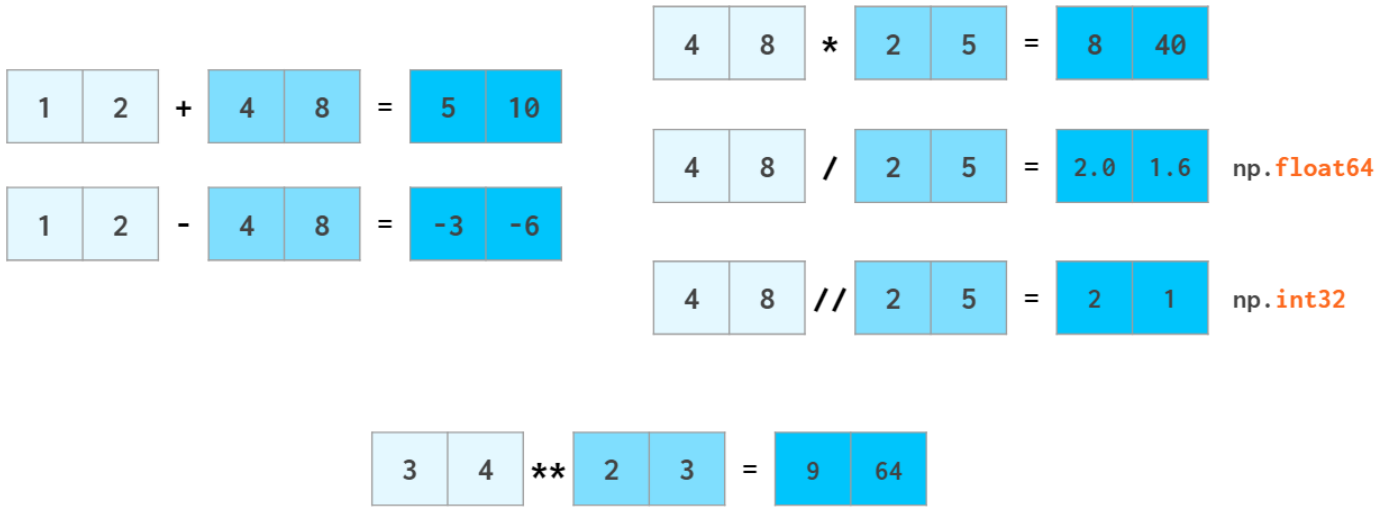
```
a = np.array([1, 2, 3])
b = a          # no copy
c = a[:]       # no copy!!!
d = a.copy()   # copy
```

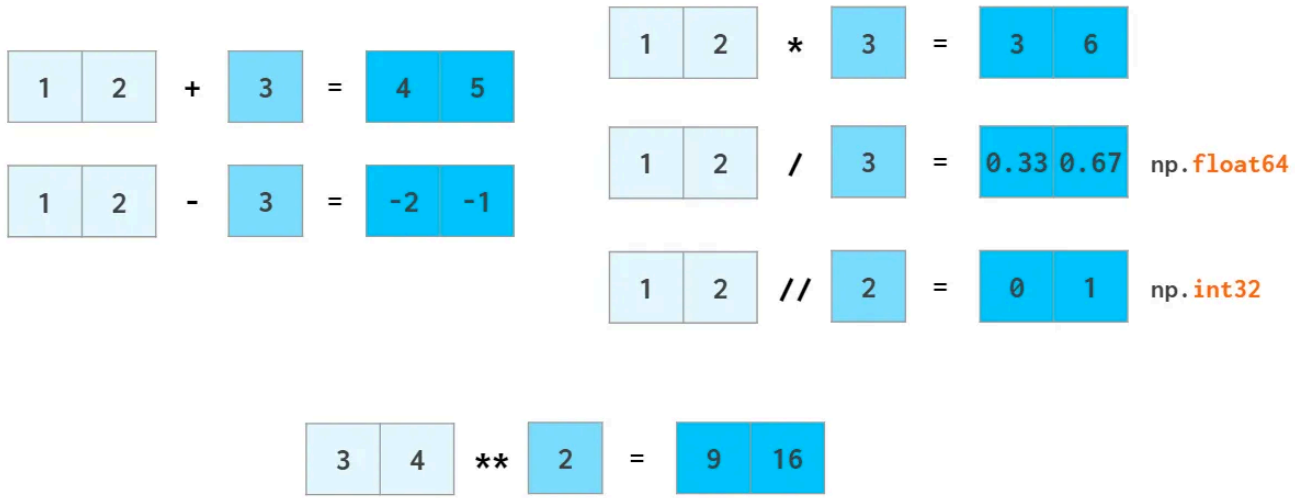
Boolean indexing



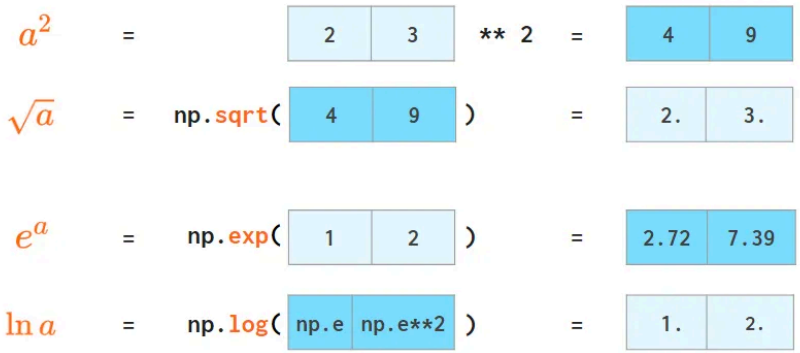
&	and
|	or
^	xor
~	not

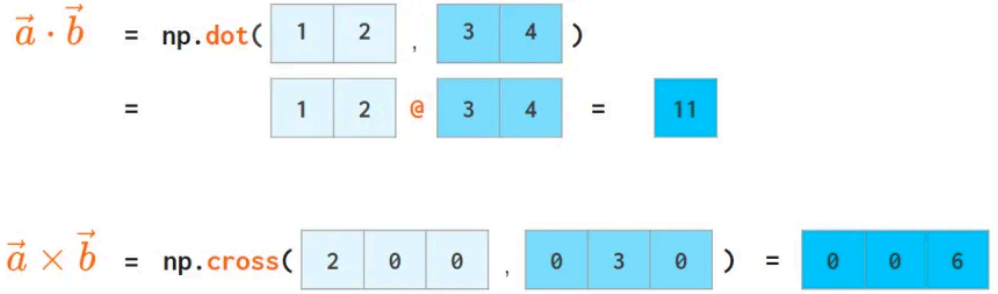
Vector operations

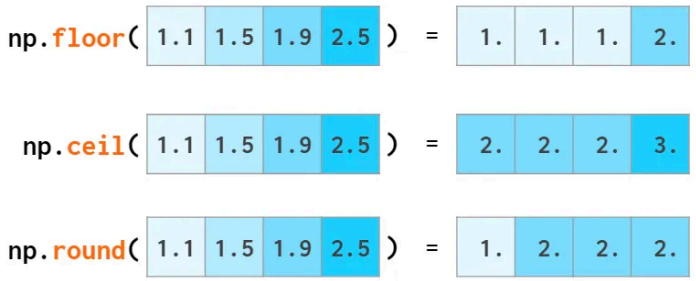


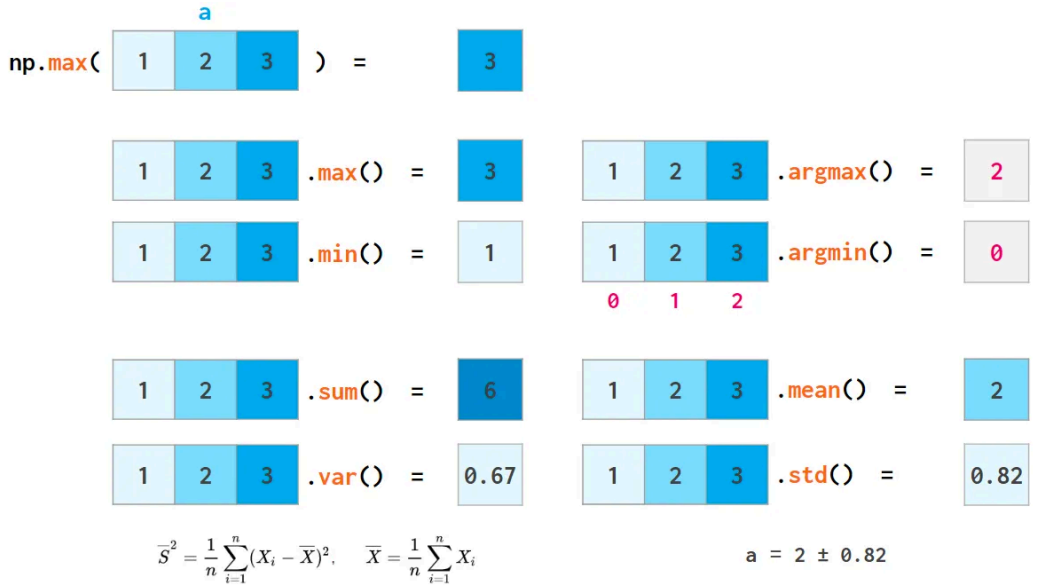


Math functions

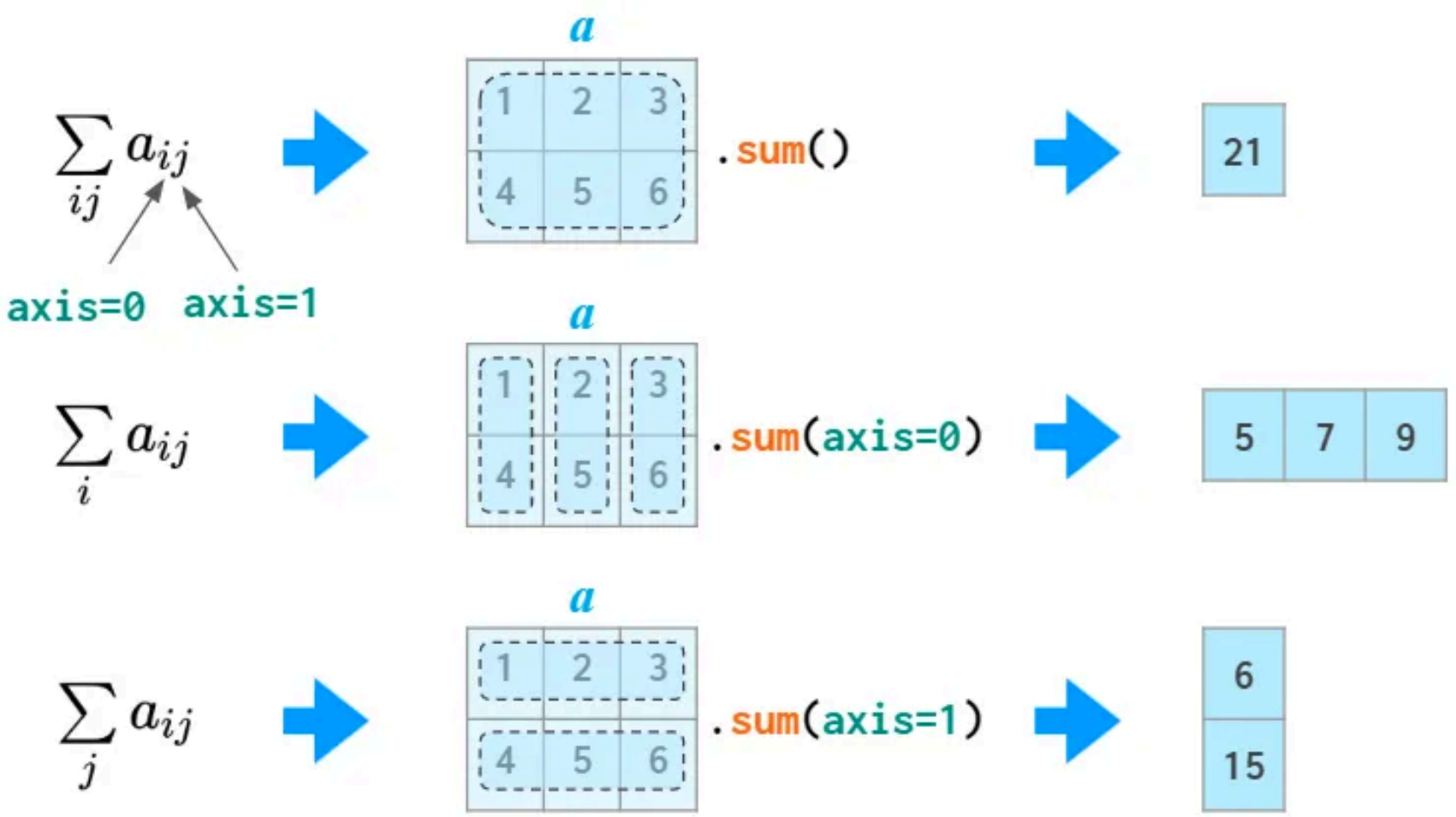




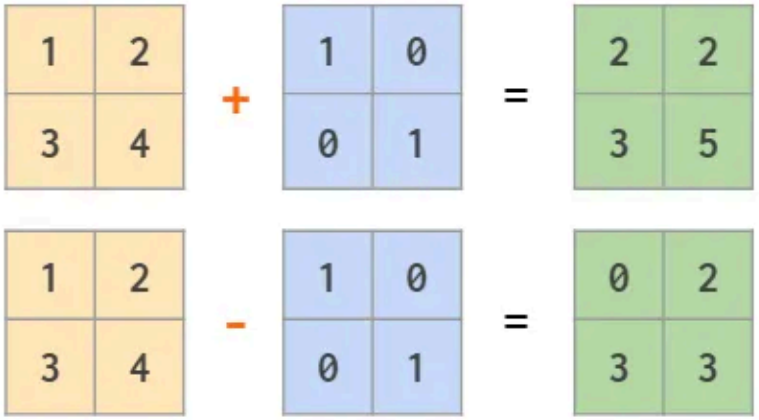


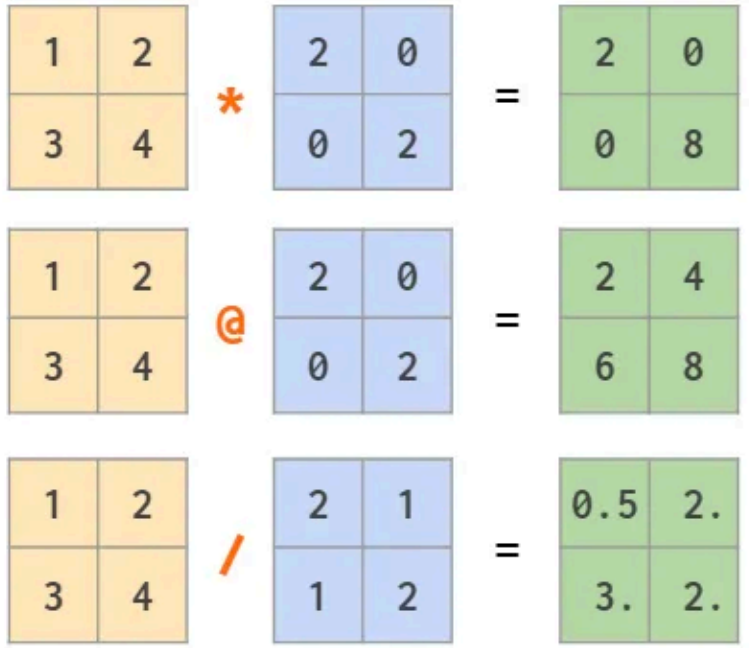


The axis

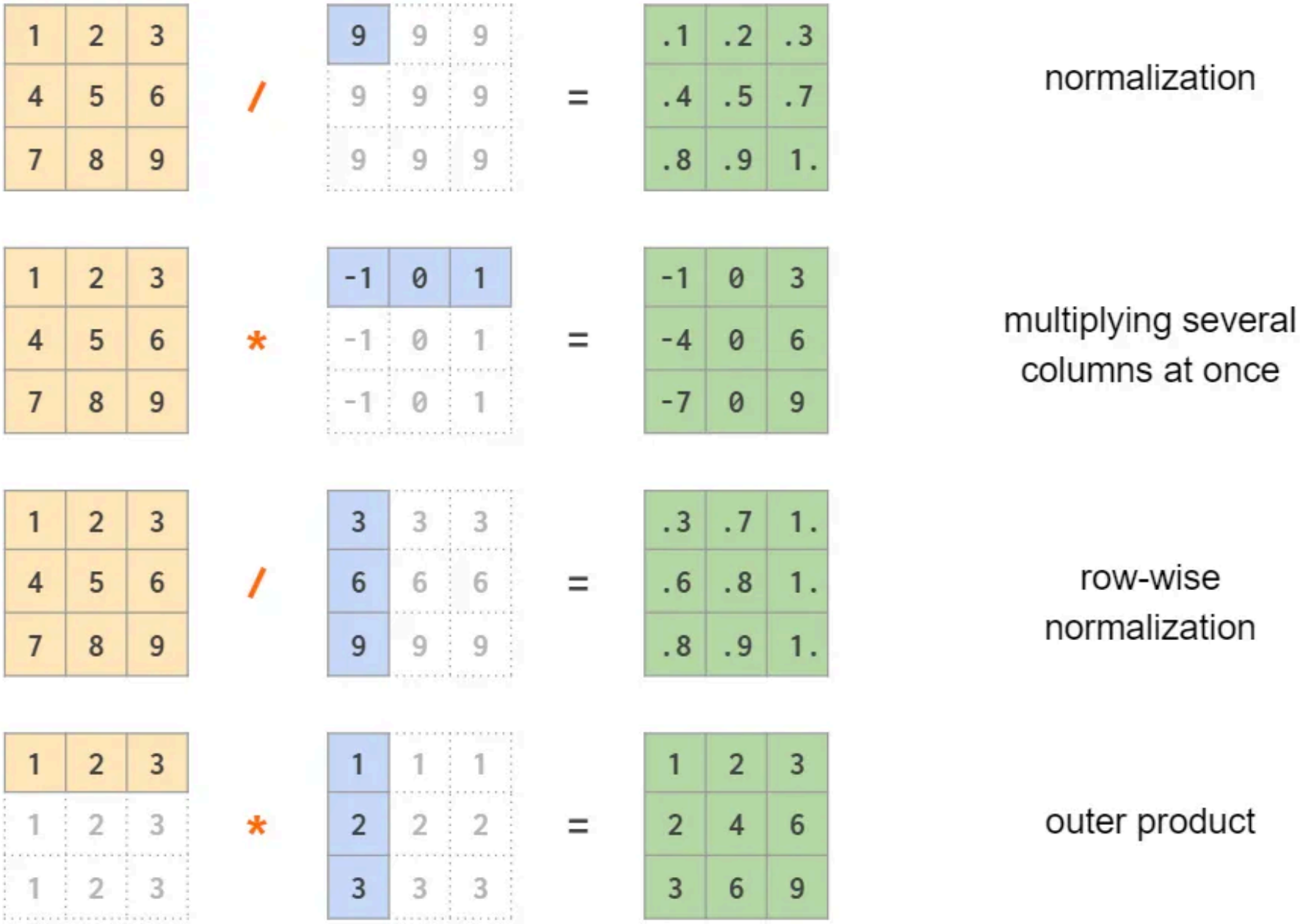

$\sum_{ij} a_{ij}$
axis=0 axis=1
a
1 2 3
4 5 6
.sum()
21
$\sum_{i} a_{ij}$
a
1 2 3
4 5 6
.sum(axis=0)
5 7 9
$\sum_{j} a_{ij}$
a
1 2 3
4 5 6
.sum(axis=1)
6
15

Matrix operations





Broadcasting



General broadcasting rules

- When operating on two arrays, NumPy compares their shapes element-wise
  - starts with the rightmost dimension and works backwards
  - two dimensions are compatible when:
    - they are equal, or
    - one of them is 1
  - otherwise a ValueError: operands could not be broadcast together exception is thrown
- Input arrays do not need to have the same number of dimensions
  - resulting array will have the same number of dimensions as the input array with the greatest number of dimensions
  - the size of each dimension is the largest size of the corresponding dimension among the input arrays
  - missing dimensions are assumed to have size one

```
A      (4d array):  8 x 1 x 6 x 1
B      (3d array):      7 x 1 x 5
Result (4d array):  8 x 7 x 6 x 5

A      (2d array):  5 x 4
B      (1d array):      1
Result (2d array):  5 x 4

A      (2d array):  5 x 4
B      (1d array):      4
Result (2d array):  5 x 4

A      (3d array):  15 x 3 x 5
B      (3d array):  15 x 1 x 5
Result (3d array):  15 x 3 x 5
```

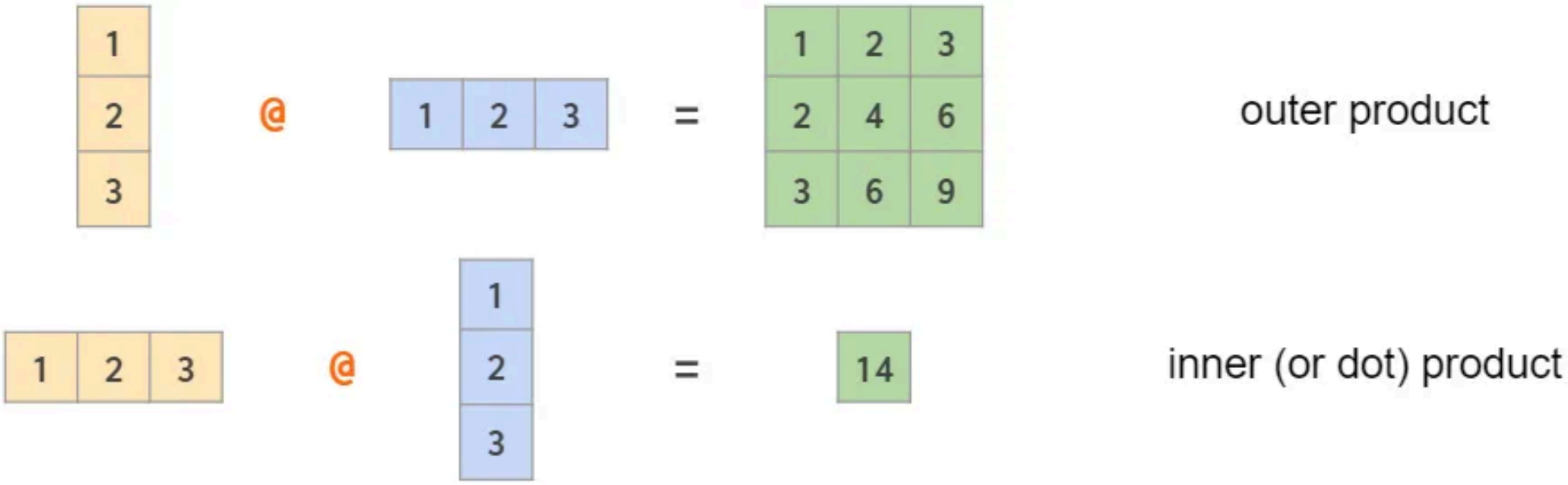
```
A      (3d array):  15 x 3 x 5
B      (2d array):       3 x 5
Result (3d array):  15 x 3 x 5

A      (3d array):  15 x 3 x 5
B      (2d array):       3 x 1
Result (3d array):  15 x 3 x 5

A      (1d array):  3
B      (1d array):  4 # mismatch

A      (2d array):      2 x 1
B      (3d array):  8 x 4 x 3 # mismatch
```

Must know your shapes …



Concatenation