TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

A hands-on introduction to Scientific Python

Markus Wallerberger

Department of solid state physics, TU Wien
Co-organised by the Fachschaft Doktorat

What about speed? (philosophical remarks)

What about speed?

- Python needs an interpreter
- executes slower than compiled code
- loops are usually the bottleneck

What about speed?

- The most important time is *your* time.

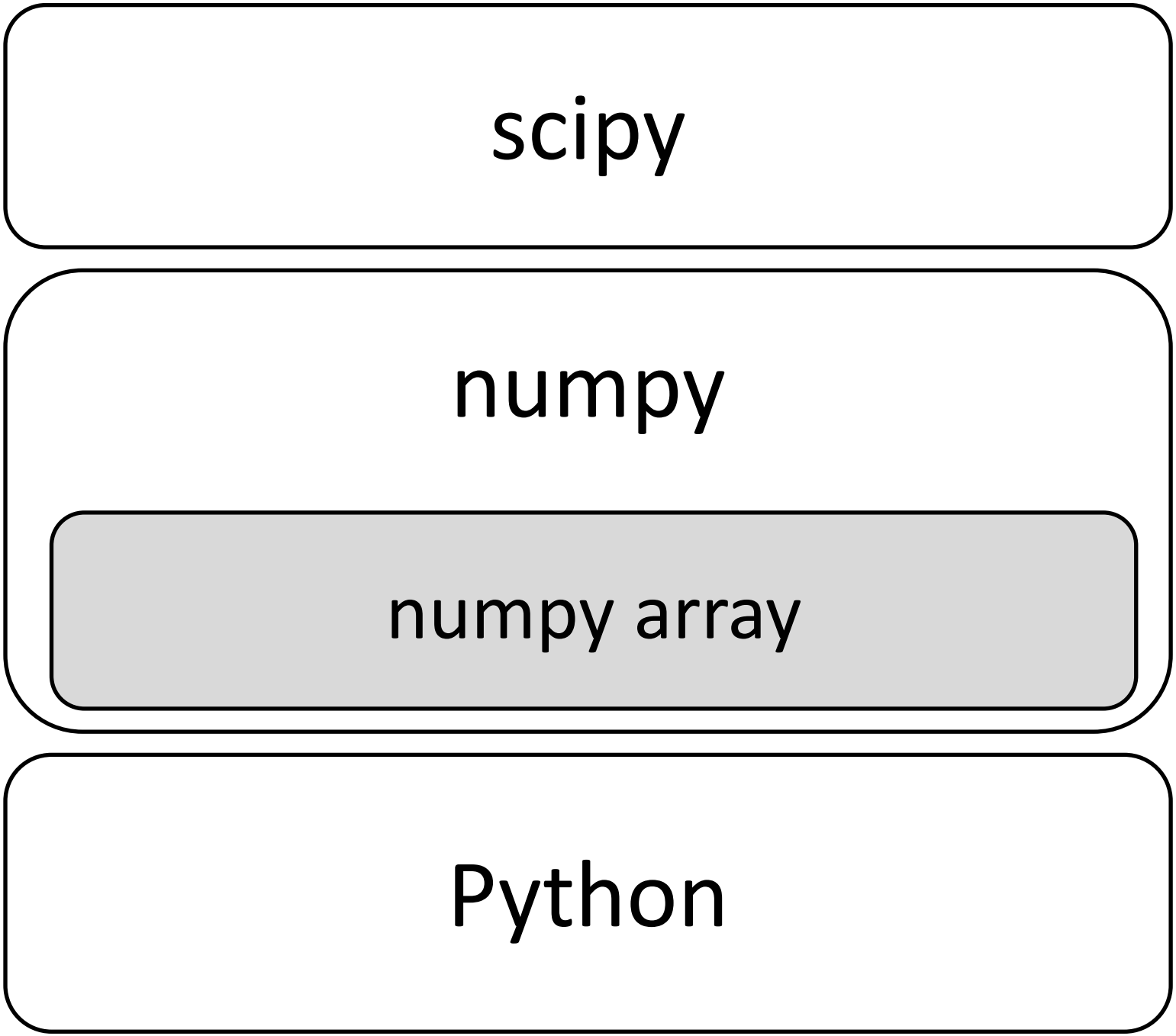
- The second most important time is *your* time, six months from now.

What about speed?

- Python needs an interpreter
- executes slower than compiled code
- **loops are usually the bottleneck**

NUMERICS?

Numerics


scipy
numpy
numpy array
Python

numpy arrays

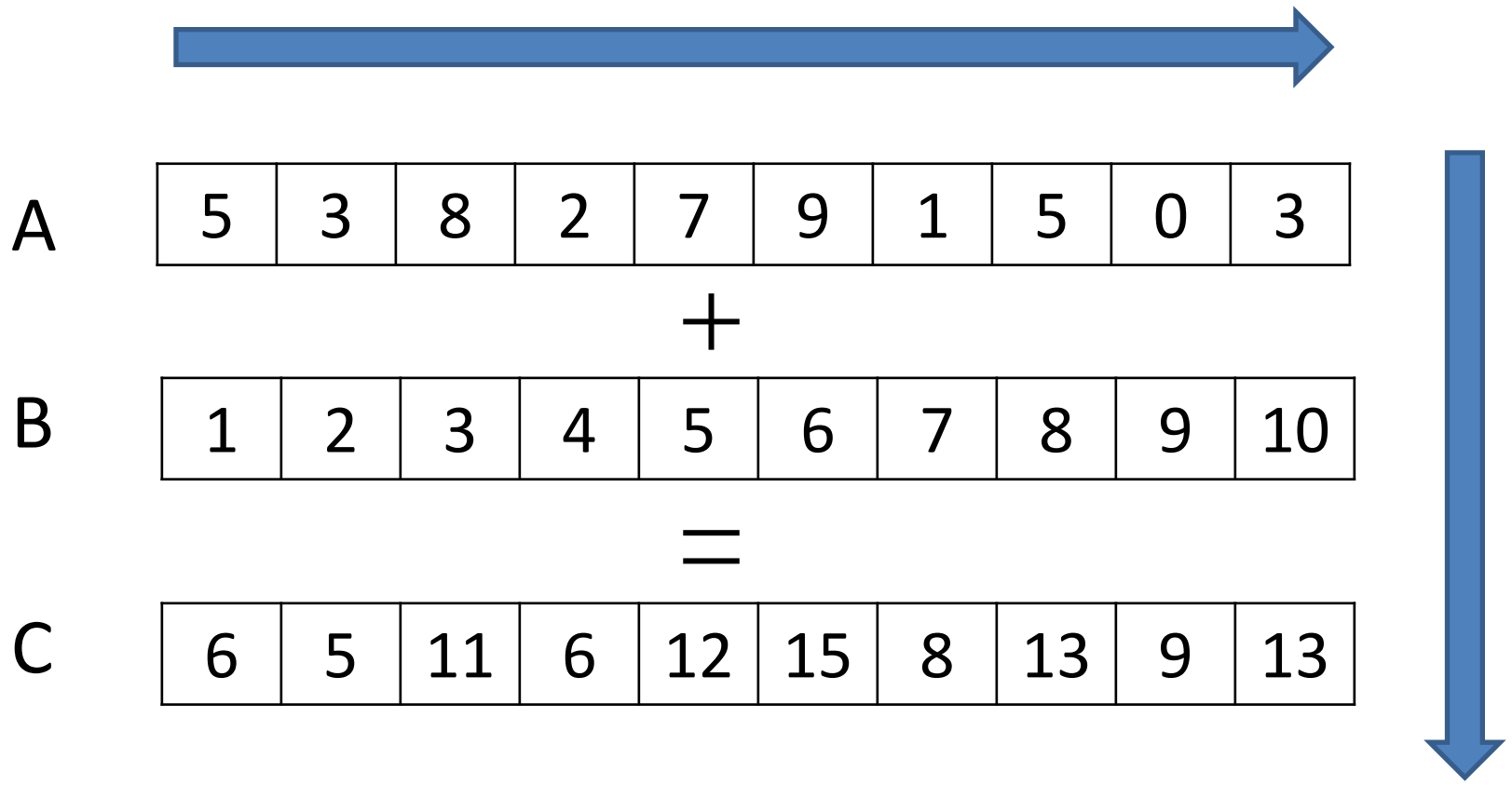
- *n*-dimensional arrays of one data type

```
import numpy as np
A = np.array([5, 2, 8, 2, 7, 9, 1, 5, 0, 3])
```

- manipulation (like Matlab)
  - vectorisation
  - slicing
  - reduction

construction

- np.zeros(10) → array of 10 zeros
  np.ones(10) → array of 10 ones
- np.arange(2, 15) → 2, 3, ..., 14
- Convert from and to a list:
  my_list = list(my_array)
  my_array = np.array(my_list)
- Read from text file:
  data = np.fromfile("data.txt", sep=" ")


A
5 3 8 2 7 9 1 5 0 3
+
B
1 2 3 4 5 6 7 8 9 10
=
C
6 5 11 6 12 15 8 13 9 13

"vectorisation"

- C = A + B
- C = A * B
- C = 2 * A
- C = sin(A)
- C += 1
- C[...] = 3
- ~~C = 3~~

for i in **range**(9):

- C[i] = A[i] + B[i]
- C[i] = A[i] * B[i]
- C[i] = 2*A[i]
- C[i] = sin(A[i])
- C[i] = C[i] + 1
- C[i] = 3

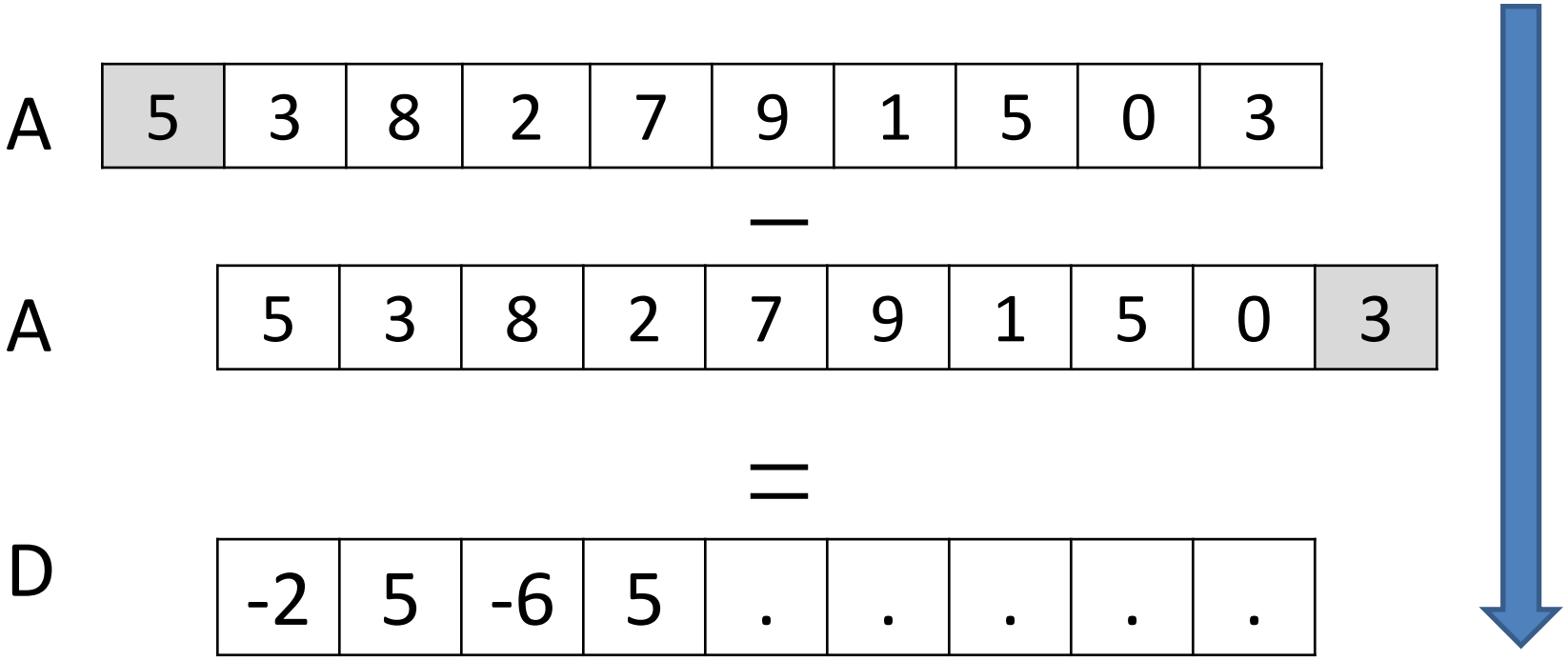
slicing

- A[2:5] = 1
- A[::-1] = C

- for i in range(2,5):
  A[i] = 1
- for i in range(9):
  A[9-i] = C[i]

How would you implement forward differences?

$$\Delta f_n \sim= f(x_{n+1}) - f(x_n)$$



D = A[1:] – A[:-1]

masking

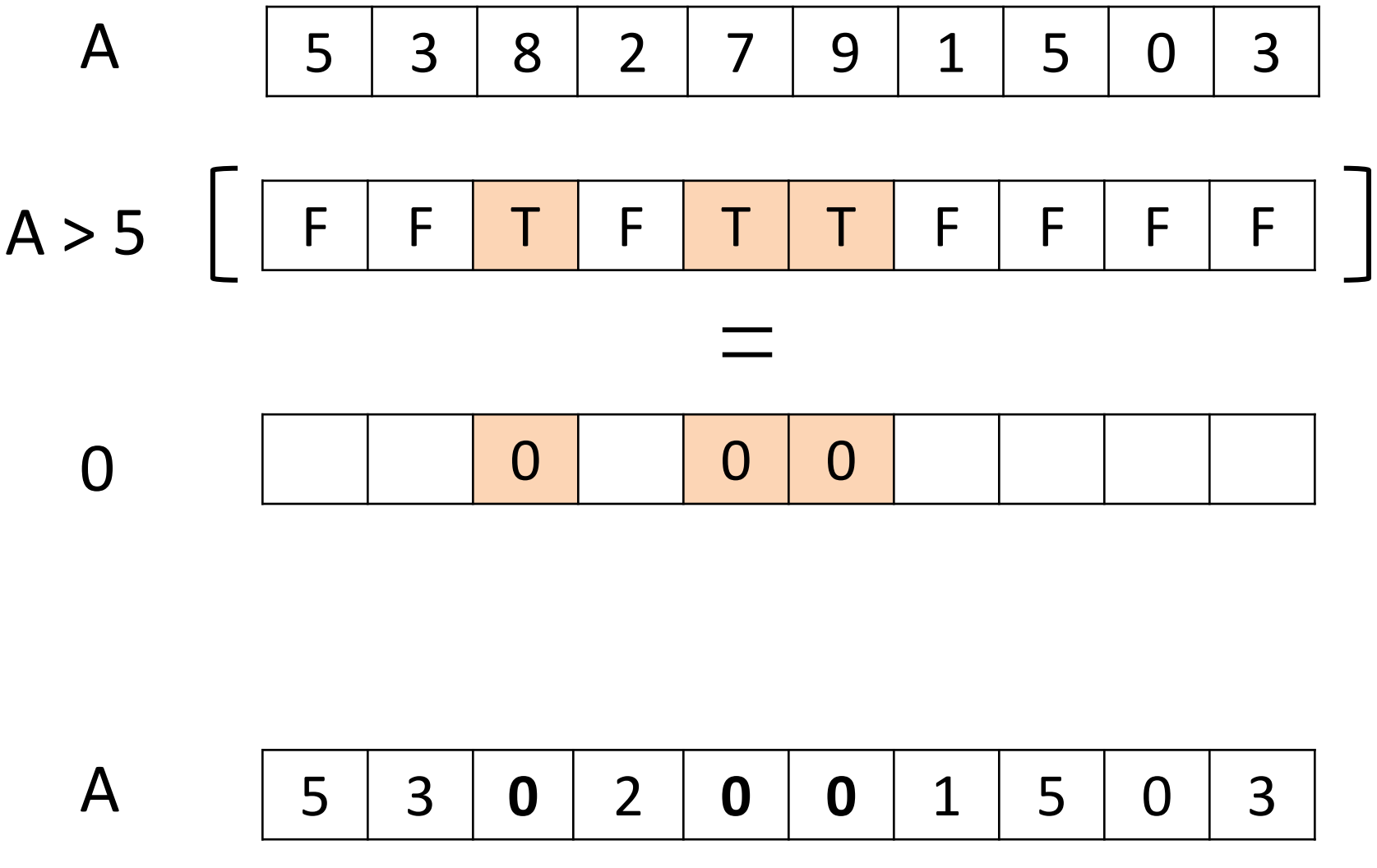
- vectorisation also for conditions
  A > 5 → array of True and False

- can be combined with slicing
  A[A > 5] = 0

- **for** i **in** range(9):
  if A[i] > 5:
  A[i] = 0

- np.where(cond, then_expr, else_expr)

A[A > 5] = 0



Cherry-picking

- using one array as indices for another array

```
indices = np.array([4, 2, 5])
A = np.array([1., 2., 3., 5., 8., 0., 13., 21.])
A[indices]
```

- useful: some functions return indices:
- np.nonzero(A) → indices of non-zero elements
- np.argsort(A) → indices that give a sorted A

reduction

- np.sum(A)
- np.prod(A)
- np.mean(A)
- np.amax(A) → value of the maximum
  np.argmax(A) → index of the maximum
- np.count_nonzero(A) → number of Trues

- **import** numpy **as** np
- np.*[press TAB]*
- np.arange?
- x = np.arange(10)
- x + 1
- x.*[press TAB]*
- x/2
- x[1:3]
- x[x%2 == 0]
- y = 4.5 * np.ones(10)
- np.sum(y)
- x * y
- np.dot(x,y)
- np.sin(x/np.pi)

play around a bit:
Get a "feel" for the numpy array

Generate an array *x* from 100 values between 0 and 2π. Then generate an array *y* of the same size, with the values 2sin(*x*)cos(*x*).

implement the Trapezoid integration rule for some function in arrays x and y = f(x)

For an array of random numbers in the range [0, 1), give the number of elements > 0.5. Then multiply all numbers < 0.5 with two.

given a permutation p (an array of indices 0 .. *n,* but shuffled), compute the permutation q such that p[q] gives the trivial permutation.