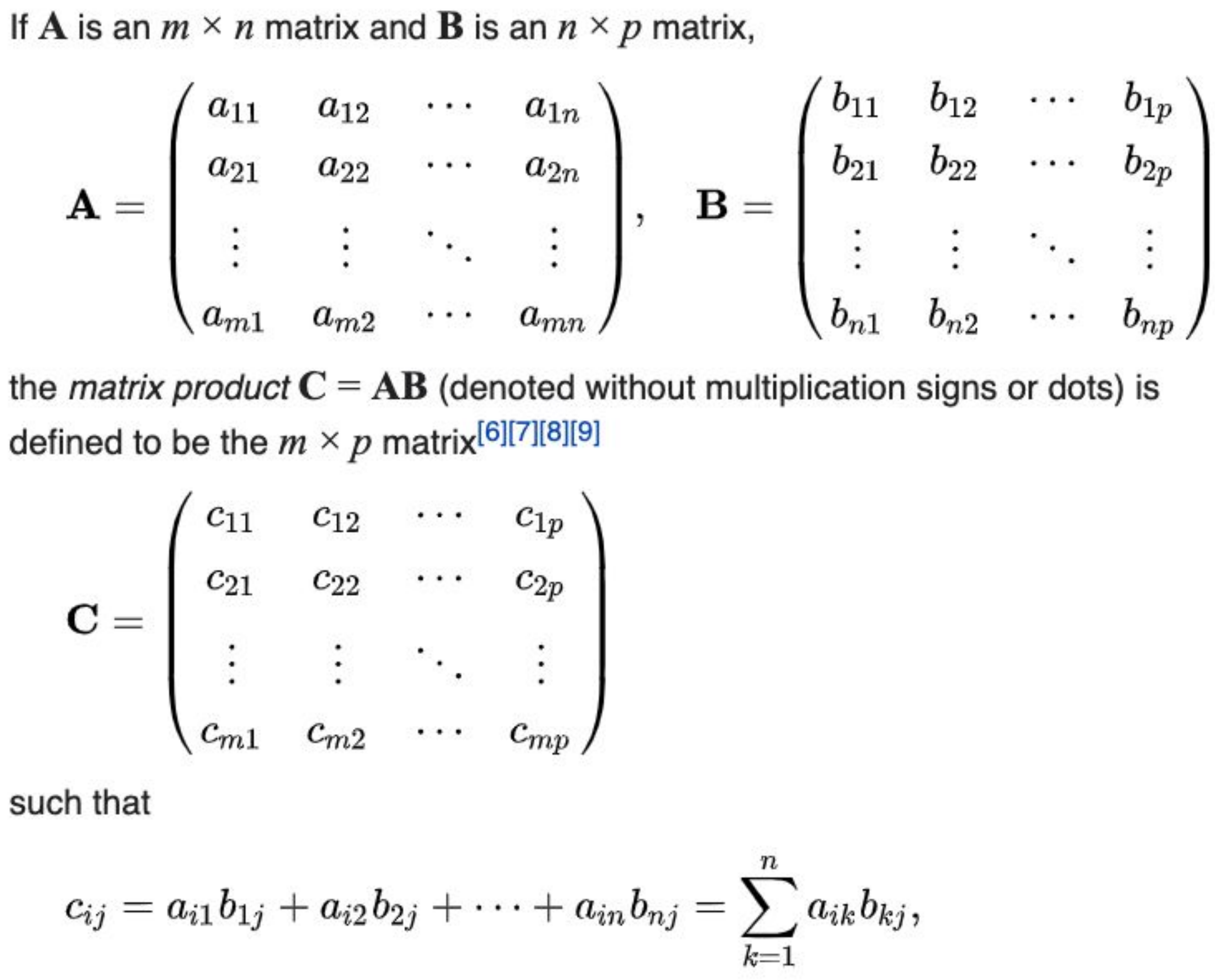
Speeding up data processing

- Fast libraries vs. pure Python
- Throughput vs. latency

Fast libraries

- Linear Algebra is the computational engine of data analysis, Neural Networks.
- Can be computed in pure python
- But: about 1000 times slower than using numpy
- Numpy Speed similar to matlab speed.
- Both use similar highly optimized libraries
- Example: LAPACK, library released in 1992,
  - written in FORTRAN!

Matlab and Numpy: fast linear algebra



If $\mathbf{A}$ is an $m \times n$ matrix and $\mathbf{B}$ is an $n \times p$ matrix,

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{np} \end{pmatrix}$$

the *matrix product* $\mathbf{C} = \mathbf{AB}$ (denoted without multiplication signs or dots) is defined to be the $m \times p$ matrix[6][7][8][9]

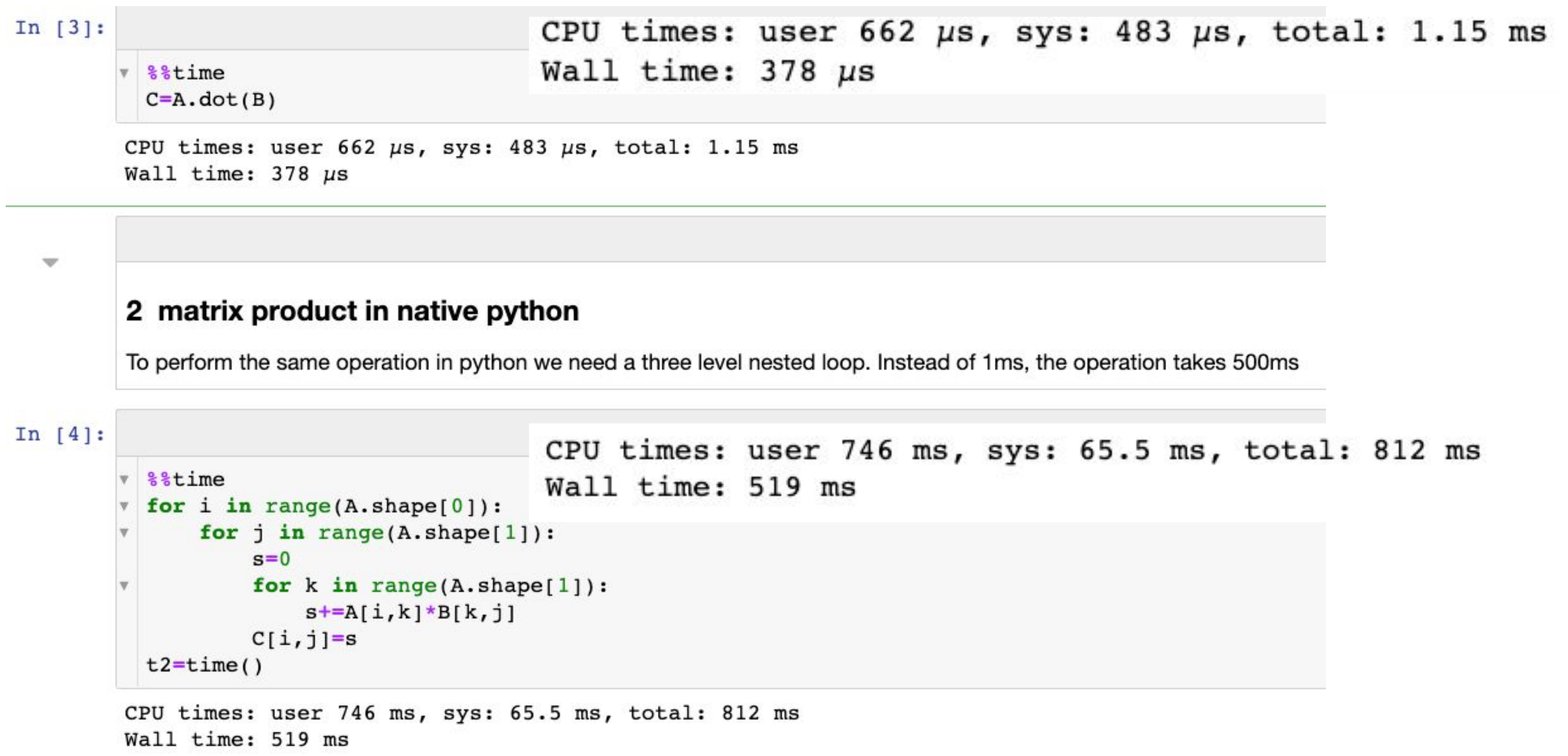
$$\mathbf{C} = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1p} \\ c_{21} & c_{22} & \cdots & c_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mp} \end{pmatrix}$$

such that

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{in}b_{nj} = \sum_{k=1}^{n} a_{ik}b_{kj},$$

Numpy vs pure python

From: 4_NumpyVsPurePython.ipynb, A,B are 100X100



In [3]:

```
%%time
C=A.dot(B)
```

```
CPU times: user 662 µs, sys: 483 µs, total: 1.15 ms
Wall time: 378 µs
```

```
CPU times: user 662 µs, sys: 483 µs, total: 1.15 ms
Wall time: 378 µs
```

2 matrix product in native python

To perform the same operation in python we need a three level nested loop. Instead of 1ms, the operation takes 500ms

In [4]:

```
%%time
for i in range(A.shape[0]):
    for j in range(A.shape[1]):
        s=0
        for k in range(A.shape[1]):
            s+=A[i,k]*B[k,j]
        C[i,j]=s
t2=time()
```

```
CPU times: user 746 ms, sys: 65.5 ms, total: 812 ms
Wall time: 519 ms
```

```
CPU times: user 746 ms, sys: 65.5 ms, total: 812 ms
Wall time: 519 ms
```

Big data starts when your data cannot fit in the memory of one computer.

Throughput vs latency

- **Throughput:** the number of bytes processed per second
  - Can benefit from parallelism
  - Can benefit from processing blocks of data.
  - Important for data science
- **Latency:** The time it takes to process a single byte from input to output.
  - Does not benefit from parallelism
  - Does not benefit from processing blocks of data.
  - Important for gaming, robotics, self-driving cars.

Queues: an example of Latency vs. throughput

- **Latency:** time between joining the line and leaving the store.
- **Throughput:** Number of people coming out of the store per minute.
- **Minimal latency**: Cashier work
- The customer cares about latency.
- The store cares about throughput (dollars per minute) and to some degree latency (customer happier).

Latency and throughput for big data

- 1TB on disk.
- 32 bit floating point numbers. $x_1, x_2, \ldots$
- Compute $\sum_i x_i$, $\sum_i x_i^2$
- Bottleneck: reading from disk: 200MB/sec -> GB / 5sec
- Reading 1TB takes 5000 seconds ~ 1.4 hours
- If we partition data among 100 machines, we can finish in 50 seconds!
- Map-Reduce = a way to organize this type of computation.

Summary / Increasing throughput for Big Data

- The bottleneck is usually moving of data, not computation.
- Hardware: better to have many cheap/slow/unreliable computers than a few expensive/fast/reliable computing.
- Map-Reduce / Hadoop / Spark = Methods for organizing computation on large, unreliable computer clusters.