

# **PHAS0102: Techniques of High-Performance Computing**

# Moodle

- You should now all have access to Moodle.
  - If not, email me

# Tutorials

- (Group 1) Mondays 10-11, Euston Road (222) G01
  - Very full
- (Group 2) Mondays 11-12, Chadwick Building 2.18
  - 9 spaces

# Virtual drop-in hour

- Wednesdays 11:30-12:30, link on Moodle
- **No drop-in on 12 October**

**PHAS0102 Part 0:  
What is High-Performance  
Computing?**

# HPC Programming languages

## Compiled languages

C / C++

Rust

Fortran

Go, Java (usually), COBOL,  
Haskell, Pascal...

## Interpreted languages

Python

Matlab

Ruby

Julia

Javascript, Lua, Maple,  
Mathematica...

**PHAS0102 Part 1:  
High-Performance Computing  
with Python**

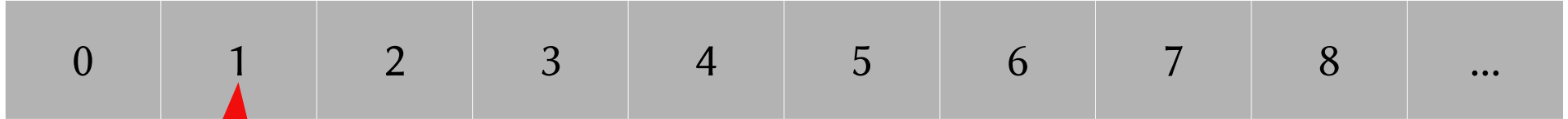
# Numpy and Scipy

- Numpy
  - Fast data types for vectors and matrices
  - Linear algebra operations
- Scipy
  - Matrix algorithms, optimisation algorithms, etc



# Memory layout

1 byte = 8 bits



Byte  
address

- It is faster to access a set of nearby bytes than a set of bytes that are spread out.
- Lower level languages like C and Fortran allow you to control exactly how memory is used, so can be much faster.

# Storing a matrix

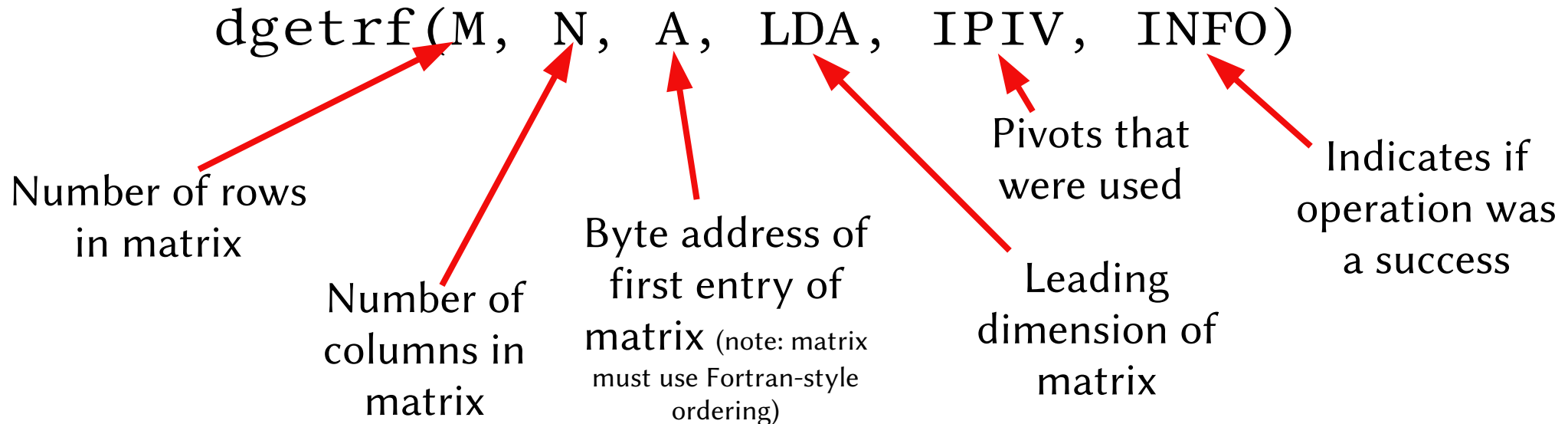
0	1	2	3	4	5	6	7	8	0
---	---	---	---	---	---	---	---	---	---

$$\begin{pmatrix} 5 & -3 \\ 2 & 1 \end{pmatrix} \quad \begin{matrix} 5 & -3 & 2 & 1 & \leftarrow \text{C-style ordering} \\ 5 & 2 & -3 & 1 & \leftarrow \text{Fortran-style ordering} \end{matrix}$$

- By default, Numpy uses C-style ordering, but you can tell it to use Fortran-style instead.

# BLAS and LAPACK

- BLAS = Basic Linear Algebra Subroutines
- LAPACK = Linear Algebra Package

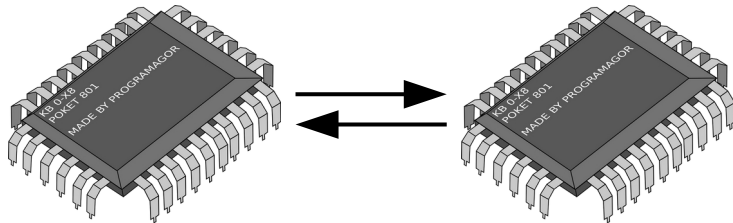
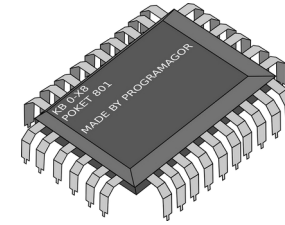
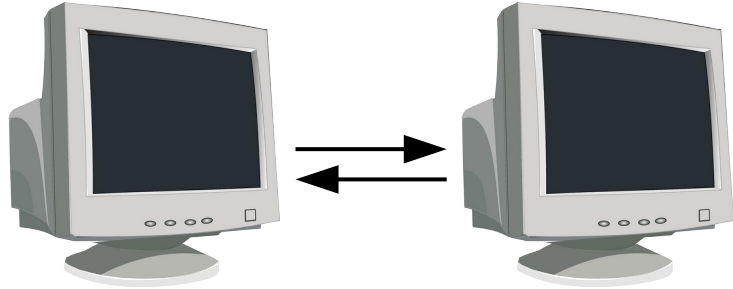


# Numpy

- Internally, Numpy uses BLAS and LAPACK routines.
- Numpy handles the memory layout, data types, etc for you.

[live Numpy demo]

# Parallelisation



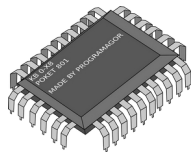
A screenshot of the Windows Task Manager window, showing the "Processes List" tab. The window title is "Windows Task Manager". The menu bar includes "File", "Refresh Rate", "Process", and "Help". The "Running Processes" list is as follows:

Process	Private Bytes	Working Set	Private Bytes	Working Set	Private Bytes	Working Set
kwm	11022 cs	0.00%	0:00	0 Sleep	5764K	381
sh	11020 cs	0.00%	0:00	0 Sleep	1576K	81
X	11017 cs	0.00%	0:01	0 Sleep	7612K	334
xinit	11016 cs	0.00%	0:00	0 Sleep	1964K	70
startx	11008 cs	0.00%	0:00	0 Sleep	1588K	84
ksnapshot	10625 cs	0.88%	0:00	0 Sleep	6508K	305
ktop	10568 cs	15.79%	0:26	0 Run	6096K	424
tcsh	8350 cs	0.00%	0:00	0 Sleep	1966K	125
mutt	8331 cs	0.00%	0:01	0 Sleep	1748K	130
kvt	8328 cs	0.00%	0:01	0 Sleep	6068K	407

At the bottom of the window, the status bar shows: "66 Processes", "Memory: 56 MB used, 5428 kB free", and "Swap: 2548 kB used, 63 M".

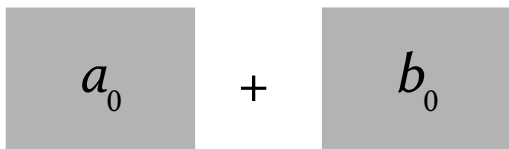


- MPI = Message Passing Interface
- Used to send commands and data between processors / computers
- `import mpi4py`



# SIMD

- SIMD = Single Instruction Multiple Data

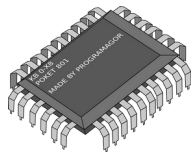


+



- AVX2 – 256 bits can be operators on in a single CPU cycle.



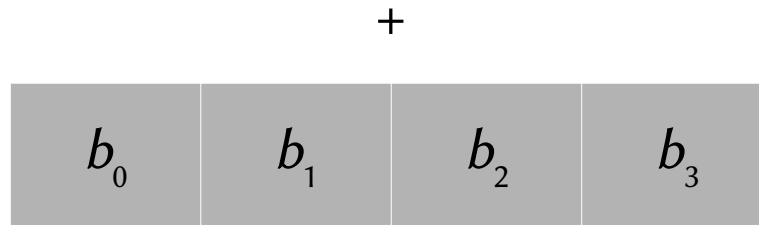


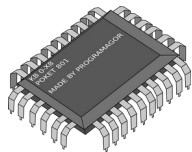
# SIMD

256 bits = 32 bytes

Single precision floating point number is 4 bytes  $\rightarrow$  8 singles at once  
Double precision floating point number is 8 bytes  $\rightarrow$  4 doubles at once

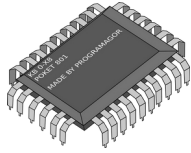
- AVX2 – 256 bits can be operators on in a single CPU cycle.





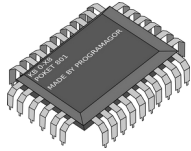
# SIMD

- Some CPUs support AVX-512 (512 bits in one operation) for some operations



# Multithreading

- Multithreading = running multiple threads / processes at once on the same CPU
- `import threading`
- Due to how Python's memory management works, only one thread will be active at a time.



# Multithreading

- Next Friday: Using Numba to do proper multithreading with Python
- Monday: Hands-on Numpy practice (tasks to work through in lecture notes or Moodle or [mscroggs.co.uk/PHAS0102](http://mscroggs.co.uk/PHAS0102))