ICHEC
Irish Centre for High-end Computing

PYTHON IN HIGH-PERFORMANCE COMPUTING

Presenters:

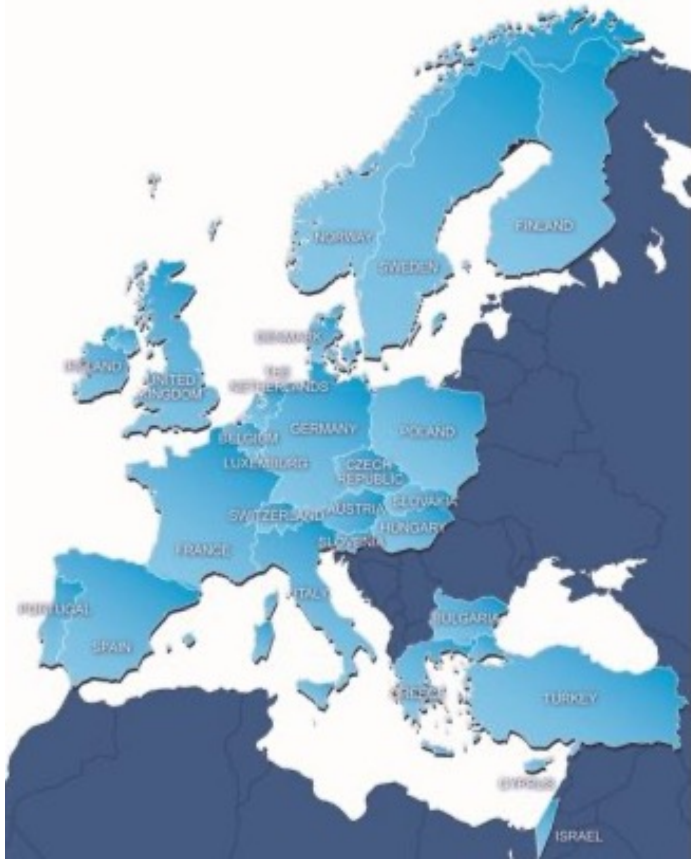
Chris Werner
Fionnuala Solomon
Adam Ralph

ICHEC
Irish Centre for High-end Computing

PRACE – Partnership for Advanced Computing in Europe

PRACE

- ICHEC is a member of PRACE, which has 20+ member countries
- Provides a range of organisations from academia to industry with access to Europe’s supercomputers via host members
- PRACE’s training helps scientists and engineers make optimal use of the machines in their quest for new discoveries



To start: 5 mins

- Ensure you can log into Kay

```
ssh courseXX@kay.ichec.ie
```

- If you can, sit back and relax!

- If you have not set up an ssh key, YOU MUST DO IT NOW!

```
ssh-keygen –t ed25519
```

  - Copy the FULL KEY generated in id_ed25519.pub and send to either fionnuala.solomon@ichec.ie or adam.ralph@ichec.ie

- If you do not know your course number and the log in password, check your emails & junk folder for a message titled “Python in HPC @ ICHEC” sent on Monday

ICHEC
Irish Centre for High-end Computing

Schedule – Day 1

Time	Topic
13:00	Introduction, Welcome and Setup
13:45	Fundamentals
14:45	**BREAK**
15:00	Numba
16:00	*Session End*

ICHEC
Irish Centre for High-end Computing

Schedule – Day 2

Time	Topic
13:00	Cython
14:00	**BREAK**
14:10	Interfacing with C libraries using cffi
15:00	**BREAK**
15:10	MPI – Part 1
16:15	*Session End*

Schedule – Day 3

ICHEC
Irish Centre for High-end Computing

Time	Topic
13:00	MPI – Part 2
14:30	**BREAK**
14:40	Dask Array
15:30	Dask MPI
15:55	**BREAK**
16:00	Dask GPU
16:45	*Session End*

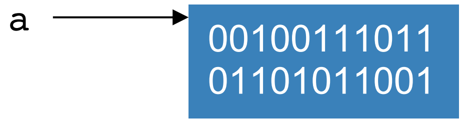
Zen of Python

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Flat is better than nested.
- Sparse is better than dense.
- Readability counts.
- Special cases aren't special enough to break the rules.
- Errors should never pass silently, unless explicitly silenced.
- In the face of ambiguity, refuse the temptation to guess.
- There should be one—and preferably only one—obvious way to do it.
- Although that way may not be obvious at first unless you're Dutch.
- Now is better than never. Although never is often better than *right* now.
- If the implementation is hard to explain, it's a bad idea.
- Namespaces are one honking great idea – lets do more of those

Python Overview

- Python is an interpreted, high-level and general-purpose programming language
  - Emphasis on code readability with use of significant whitespace
  - Object oriented and language constructs approach promote clear and logical code writing
- Objects are dynamic – can create objects that are non-static
- Operations can be overloaded
- Highly flexible – as close to ‘English’ as coding can get

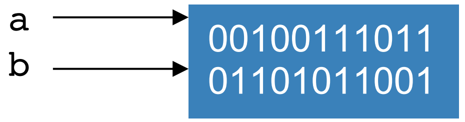
Reference counting & garbage collection



`(ref = 2)`

```
a = np.random.random(x)
```

Reference counting & garbage collection

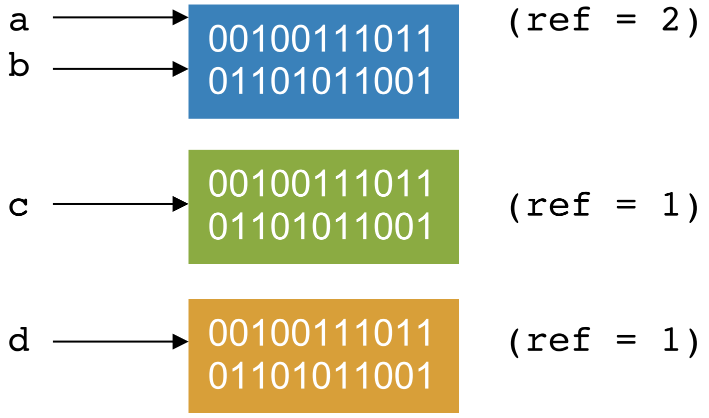


```
(ref = 2)
```

`a = np.random.random(x)`

`b = a.T (` *increases ref count* `)`

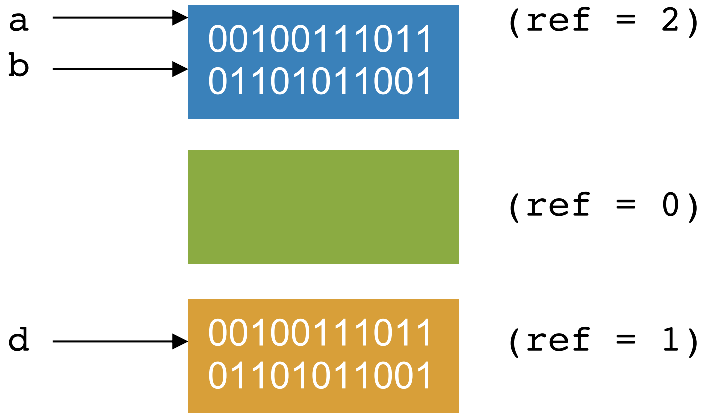
Reference counting & garbage collection



```
a = np.random.random(x)
b = a.T ( increases ref count )
```

```
c = np.random.random(x)
d = np.random.random(x)
```

Reference counting & garbage collection



```
a = np.random.random(x)
b = a.T
```
(*increases ref count*)

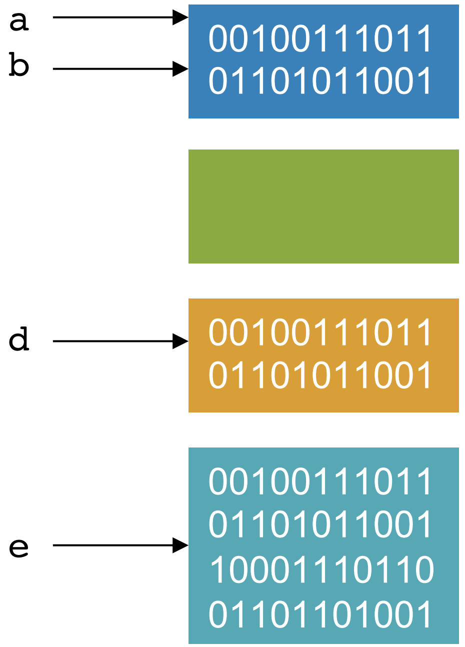
```
c = np.random.random(x)
d = np.random.random(x)
```

```
del c
```
(*sets ref count to 0*)

OR

```
c = 1
```

Reference counting & garbage collection



(ref = 2)

```
a = np.random.random(x)
b = a.T
```
(*increases ref count*)

(ref = 0)

```
c = np.random.random(x)
d = np.random.random(x)
```

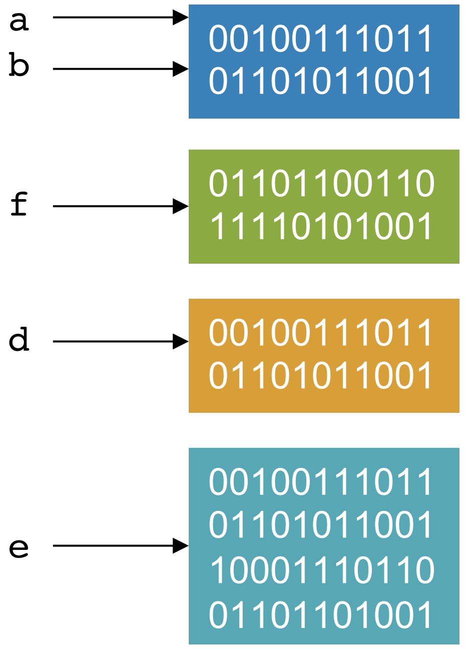
(ref = 1)

```
del c
```
(*sets ref count to 0*)

(ref = 1)

```
e = np.random.random(y)
```
(y > x)

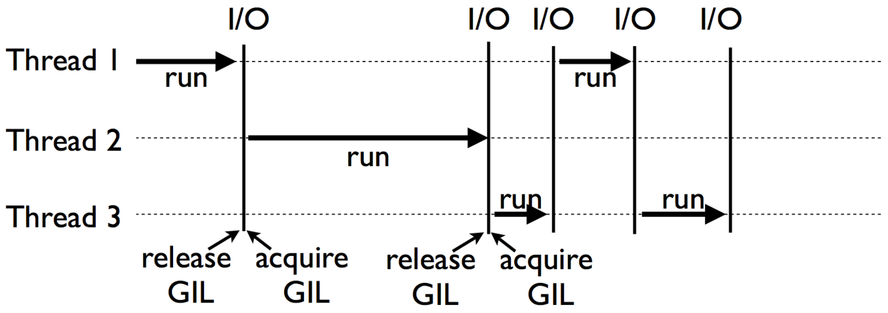
Reference counting & garbage collection



```
(ref = 2)    a = np.random.random(x)
             b = a.T ( increases ref count )

(ref = 1)    c = np.random.random(x)
             d = np.random.random(x)

(ref = 1)    del c ( sets ref count to 0 )

(ref = 1)    e = np.random.random(y)
             ( y > x )

             f = np.random.random(x)
```

Global Interpreter Lock (GIL) in CPython

- A lock is a mechanism of forcing limits on access to a resource in an environment where there are many threads of execution



- Two methods
  - `acquire()`
  - `release()`

Parallelisation Strategies for Python

- Array based communications using NumPy
- Using caching based techniques
- JIT (just in time) compilation with Numba
- Using extended Cython programming language
- Embed compiled code in a Python program
  - C, Fortran
- Utilise parallel programming
  - Multiprocessing, MPI
- Use of libraries such as Dask for simpler methodologies

Setting up

1. Log into Kay
2. Load modules
3. Create ssh tunnel
4. Open JupyterHub
5. Set up the environment