Python for High Performance Data Analytics

—— (1) Computation ——

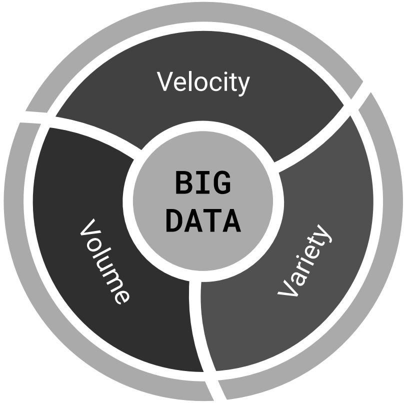
Qiyang Hu

UCLA Office of Advanced Research Computing

May 17, 2024

Outline for today

- Do we need to learn?
- Why Python is slow?
- How to speed Python up?
    - By AOT bindings
    - By JIT
    - By new interpreters
- Demos

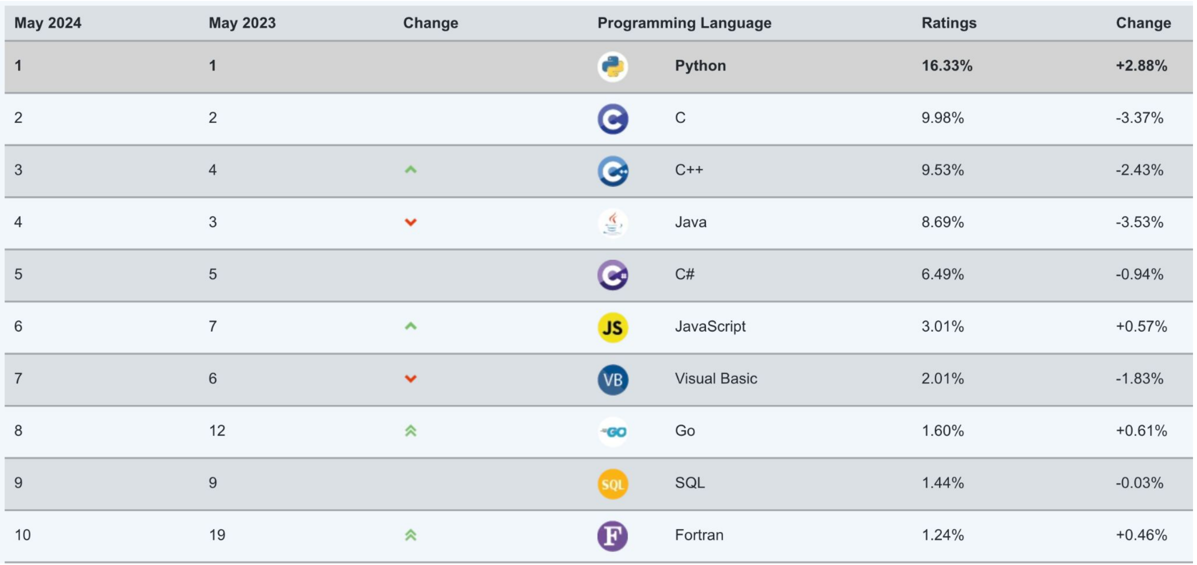


Outline for today

- Do we need to learn?
- Why Python is slow?
- How to speed Python up?
  - By AOT bindings
  - By JIT
  - By new interpreters
- Demos

Python is popular.

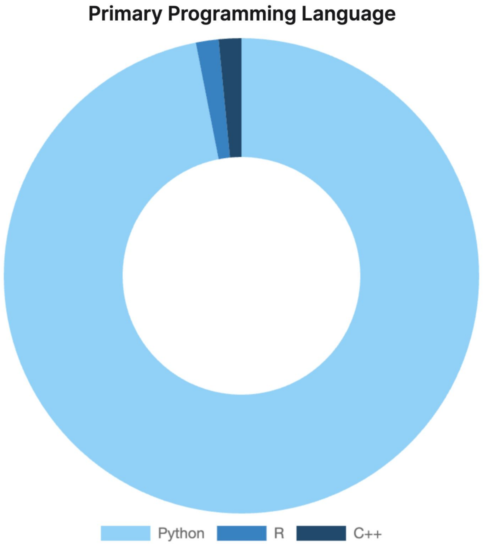
Ranking by counting hits of the most popular search engines



May 2024	May 2023	Change	Programming Language	Ratings	Change
1	**1**		**Python**	**16.33%**	**+2.88%**
2	2		C	9.98%	-3.37%
3	4	↑	C++	9.53%	-2.43%
4	3	↓	Java	8.69%	-3.53%
5	5		C#	6.49%	-0.94%
6	7	↑	JavaScript	3.01%	+0.57%
7	6	↓	Visual Basic	2.01%	-1.83%
8	12	⇈	Go	1.60%	+0.61%
9	9		SQL	1.44%	-0.03%
10	19	⇈	Fortran	1.24%	+0.46%

Latest TIOBE Index

Choice by competition winners



ML_Contests Report 2023

Python is slow.

LANGUAGES	**TIME (S) ***	**SPEEDUP VS PYTHON**
Python 3.10.9	970 s	1x
NUMPY	171 s	6x
SCALAR C++	0.11 s	9000x

According to Chris Lattner, using Mandelbrot Algorithm on AWS instance h3-standard-88 with Intel Xeon (https://www.modular.com/max/mojo)

Do we need to learn in the ChatGPT age?

Learning can help us

- Understanding CS/HPC Concepts
- Providing chains of thoughts
- Nailing down the key problem quickly

Beyond the Zero/Few Shots

Better & Efficient Prompts

Performance tuning is a <u>dangerous</u> zone for GPT users.

- Needs the expertise and knowledge to make judgements to correct the hallucination.
- Needs to be able to cross-reference
- Needs to distill the domain knowledge

About this series

- The lectures will focus on
  - *High-level* and conceptual overviews.
  - Introducing libraries that require *minimal* efforts to boost performance.
  - Short Jupyter Notebook demos

- What can/can’t expected in the series?

✓ **CAN**	✗ **CAN’T**
● From an end users’ perspective	● From a package developers’ perspective
● A ***BIGGER***-picture review on the selected 3rd-party python libraries	● Native Python tricks (e.g. container, lazy eval, mem) ● Line-by-line explanations on these library interfaces
● Demos on specific example problems	● Discussion on the performance of various algorithms

Two Big **Do-Not**’s

Don’t optimize prematurely.

> *“The real problem is that programmers have spent far too much time worrying about efficiency in the wrong places and at the wrong times…”*
>
> -- Donald Knuth in “TAOCP”

- Easiest to understand and explain
- Quickest to write
- Easiest to test and maintain
- Most portable to migrate

Don’t trust benchmarks.

All benchmark numbers are “wrong”.

- Specific hardware/OS/libraries
- In-situ running environments
- Different nature of datasets
- Sometimes very version-sensitive

- Understand the mechanisms
- Focus on the qualitative comparisons
- Need to do your own experiments.

COMPUTATION

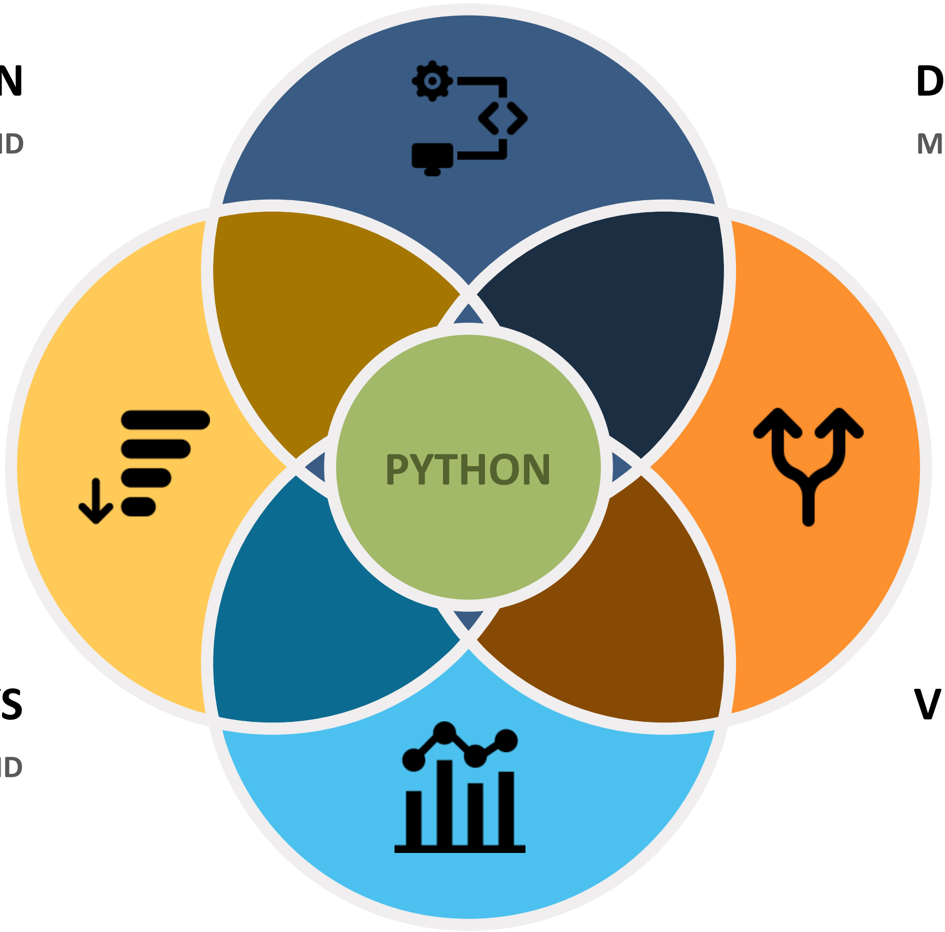
Single Node/GPU, SIMD

- Pypy, Numba, NumExpr
- Pythran, Cython
- F2py, ctypes

DISTRIBUTED

Multiple Nodes/Machines

- MapReduce-based: PySpark, PyFlink
- MPI-based: mpi4py, Horovod
- Joblib, Dask, Ray



DATA ARRAYS

Single Node/GPU, SIMD

- Numpy
- Pandas, Polars
- Modin, Pandarallel, Swifter
- Dask DataFrame, Vaex

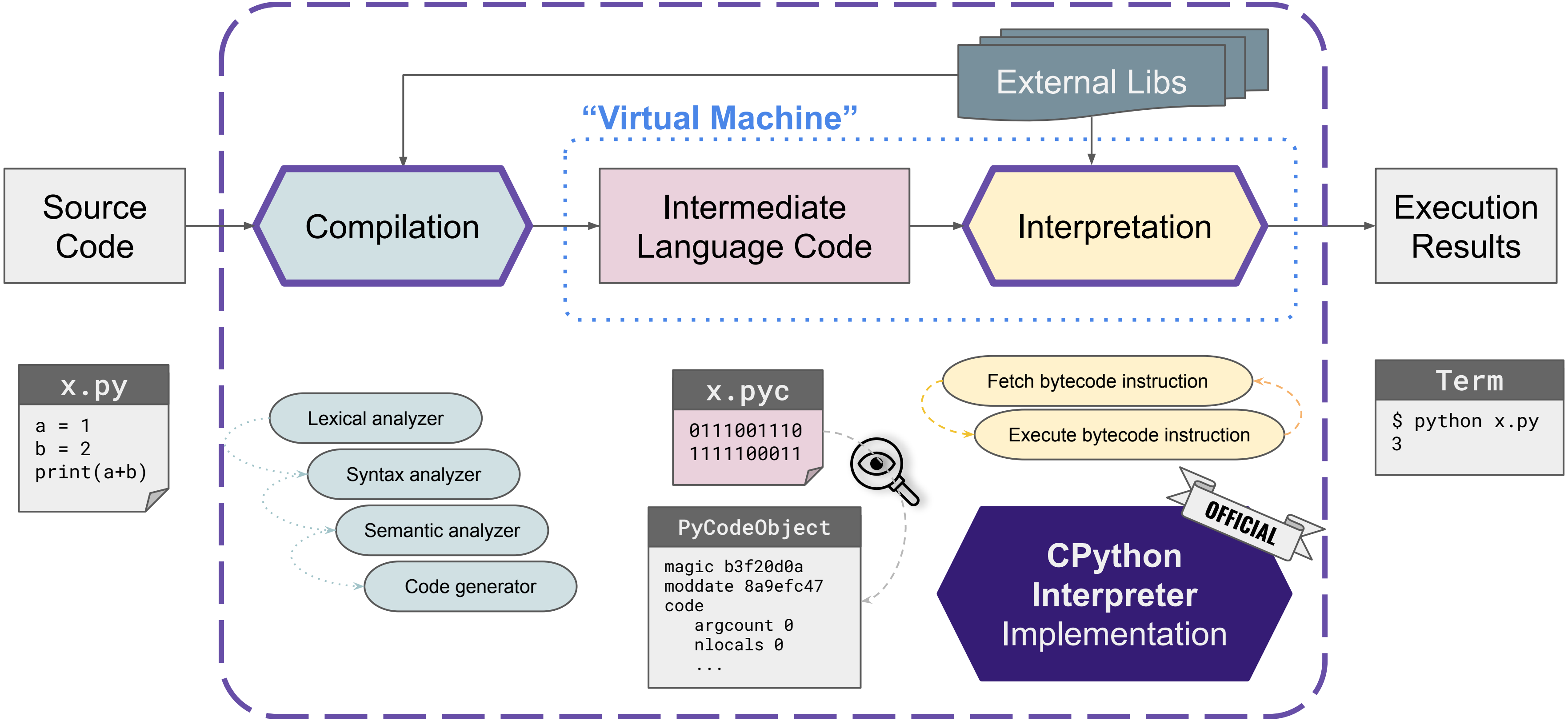
VISUALIZATION

- Viz process for big data
- Matplotlib, Bokeh, Plotly
- Holoview and Datashader
- Traited VTK, Mayavi, Paraview

Outline for today

- Do we need to learn?
- Why Python is slow?
- How to speed Python up?
  - By AOT bindings
  - By JIT
  - By new interpreters
- Demos

Seriously, what is Python?



x.py

```
a = 1
b = 2
print(a+b)
```

x.pyc

```
0111001110
1111100011
```

PyCodeObject

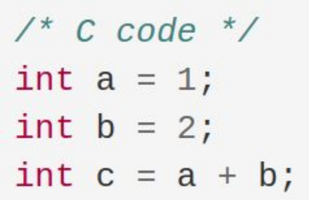
```
magic b3f20d0a
moddate 8a9efc47
code
   argcount 0
   nlocals 0
   ...
```

Term

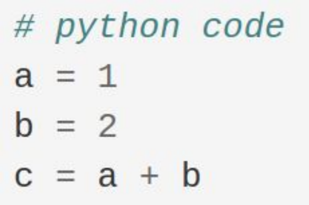
```
$ python x.py
3
```

Why Python is slow?

Python is Dynamically Typed rather than Statically Typed.

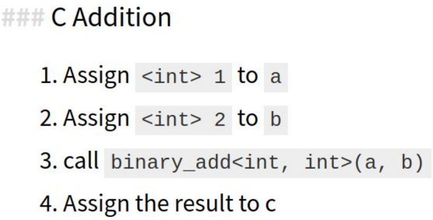


```
/* C code */
int a = 1;
int b = 2;
int c = a + b;
```



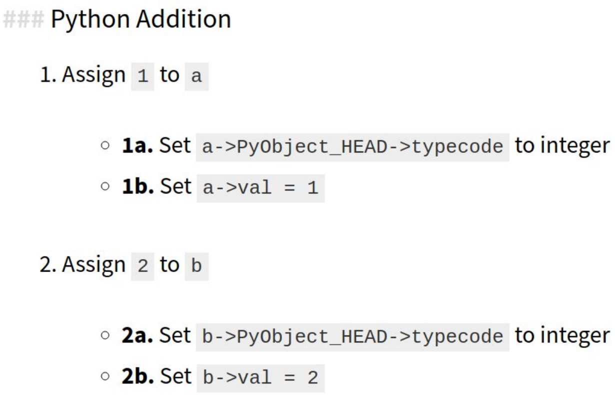
```
# python code
a = 1
b = 2
c = a + b
```

Source



C Addition

1. Assign `<int> 1` to `a`
2. Assign `<int> 2` to `b`
3. call `binary_add<int, int>(a, b)`
4. Assign the result to c



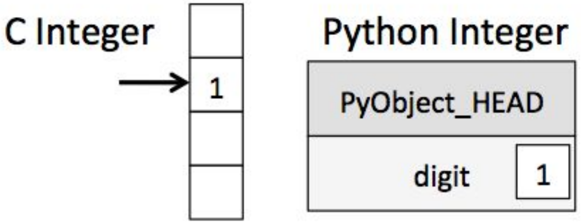
Python Addition

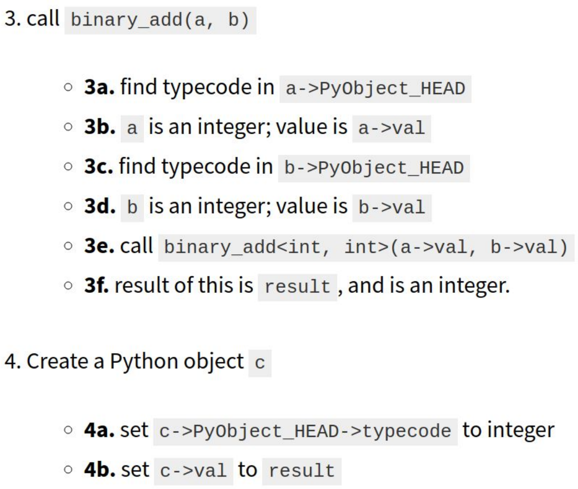
1. Assign `1` to `a`

   - **1a.** Set `a->PyObject_HEAD->typecode` to integer
   - **1b.** Set `a->val = 1`

2. Assign `2` to `b`

   - **2a.** Set `b->PyObject_HEAD->typecode` to integer
   - **2b.** Set `b->val = 2`





3. call `binary_add(a, b)`

   - **3a.** find typecode in `a->PyObject_HEAD`
   - **3b.** `a` is an integer; value is `a->val`
   - **3c.** find typecode in `b->PyObject_HEAD`
   - **3d.** `b` is an integer; value is `b->val`
   - **3e.** call `binary_add<int, int>(a->val, b->val)`
   - **3f.** result of this is `result`, and is an integer.

4. Create a Python object `c`

   - **4a.** set `c->PyObject_HEAD->typecode` to integer
   - **4b.** set `c->val` to `result`

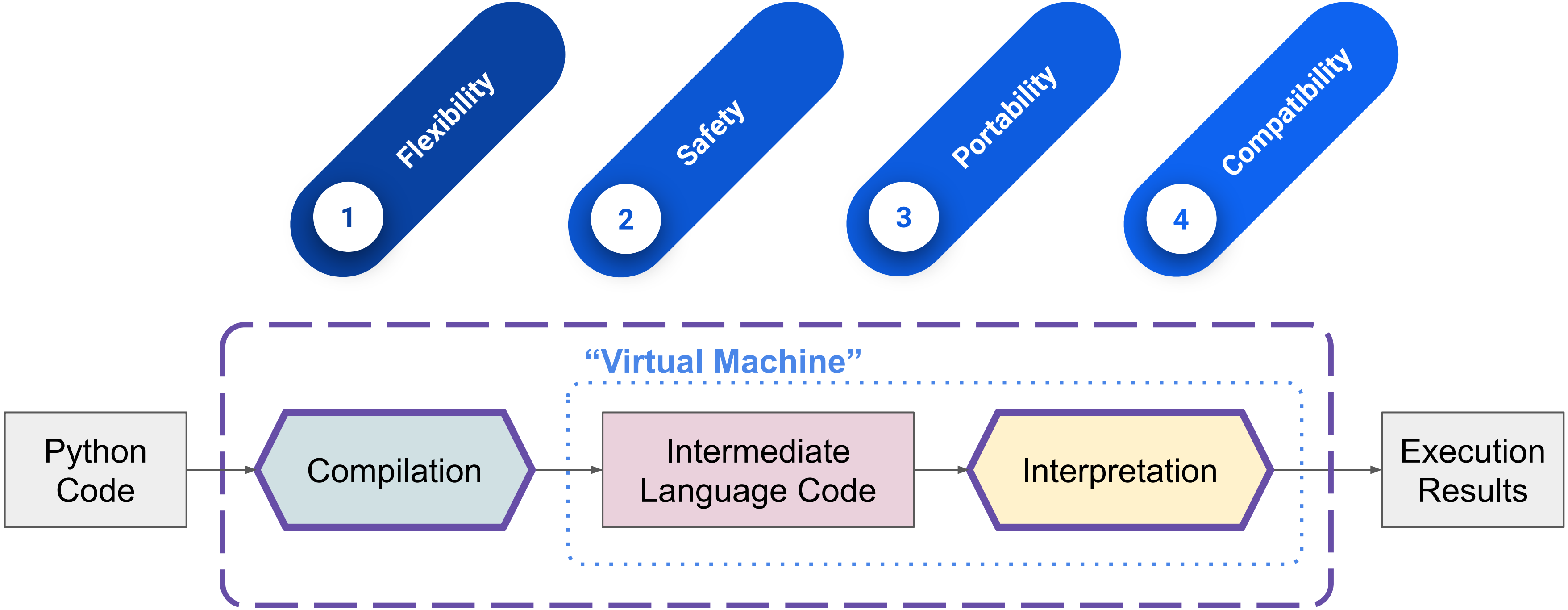
GIL: Guilty or Gilly?

- GIL (Global Interpreter Lock)
  - A mutex (or a lock) that allows only one *thread* to hold the control of the Python interpreter.
- Why Python uses it?
  - GILs is added to the ref count variables to be kept protected from race conditions
  - GIL has performance benefits of GIL in single-threaded situation.
  - Historically Python has been around when OS did not have a concept of threads.
- Correct way to use it:
  - Multi-processing vs multi-threading:
    - Multi-threading: good for IO-intensive code, bad for CPU-intensive code
    - use multiple processes with “multiprocessing” module instead of threads
    - Consider to use Intel Distribution of Python
  - Attempts from Python community to remove the GIL from CPython:
    - Gilectomy (abandoned)
    - A new compiler flag: nogil (expected in Python 3.1x)
  - Alternative Python interpreters, as GIL only with CPython
    - multiple interpreter implementations

Outline for today

- Do we need to learn?
- Why Python is slow?
- How to speed Python up?
  - By AOT bindings
  - By JIT
  - By new interpreters
- Demos

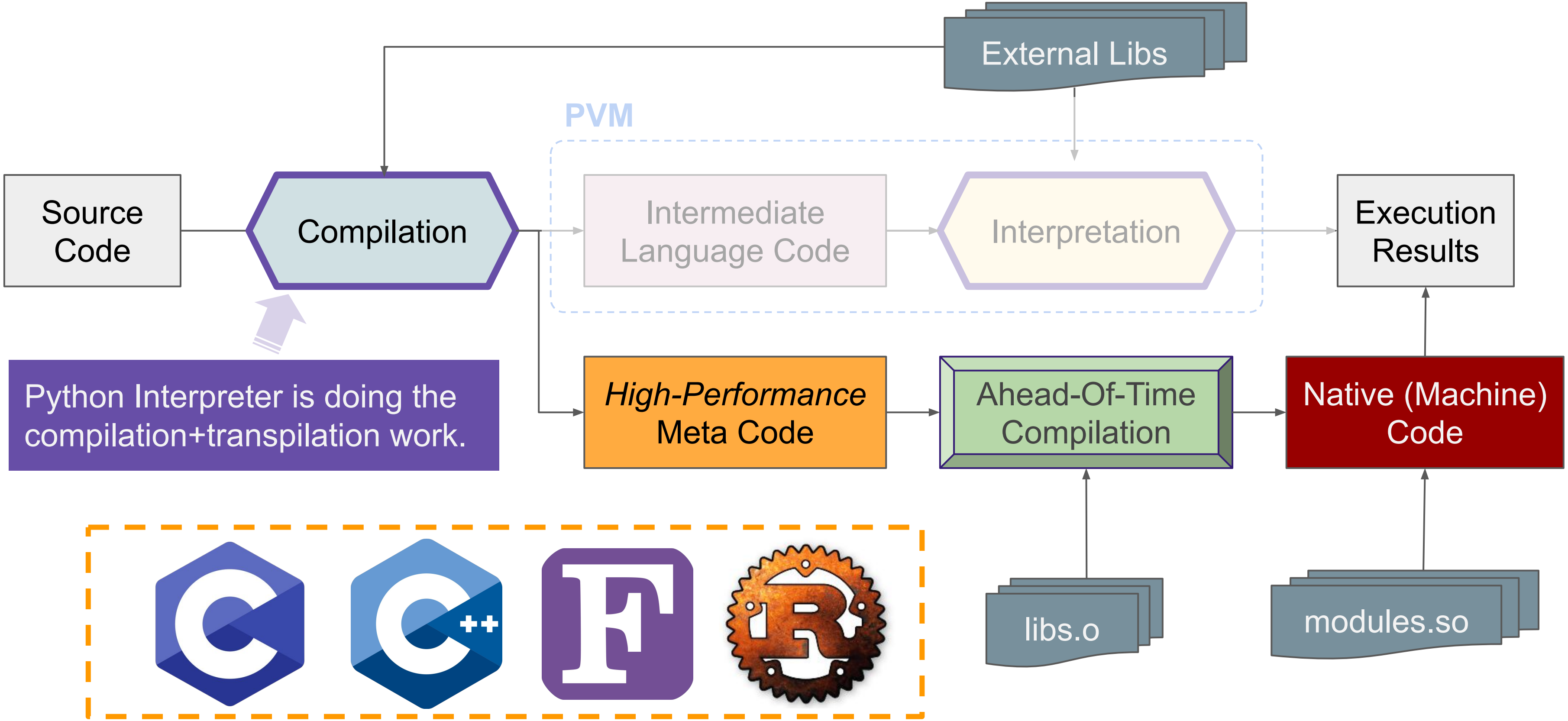
To speed up Python, it always means to compromise



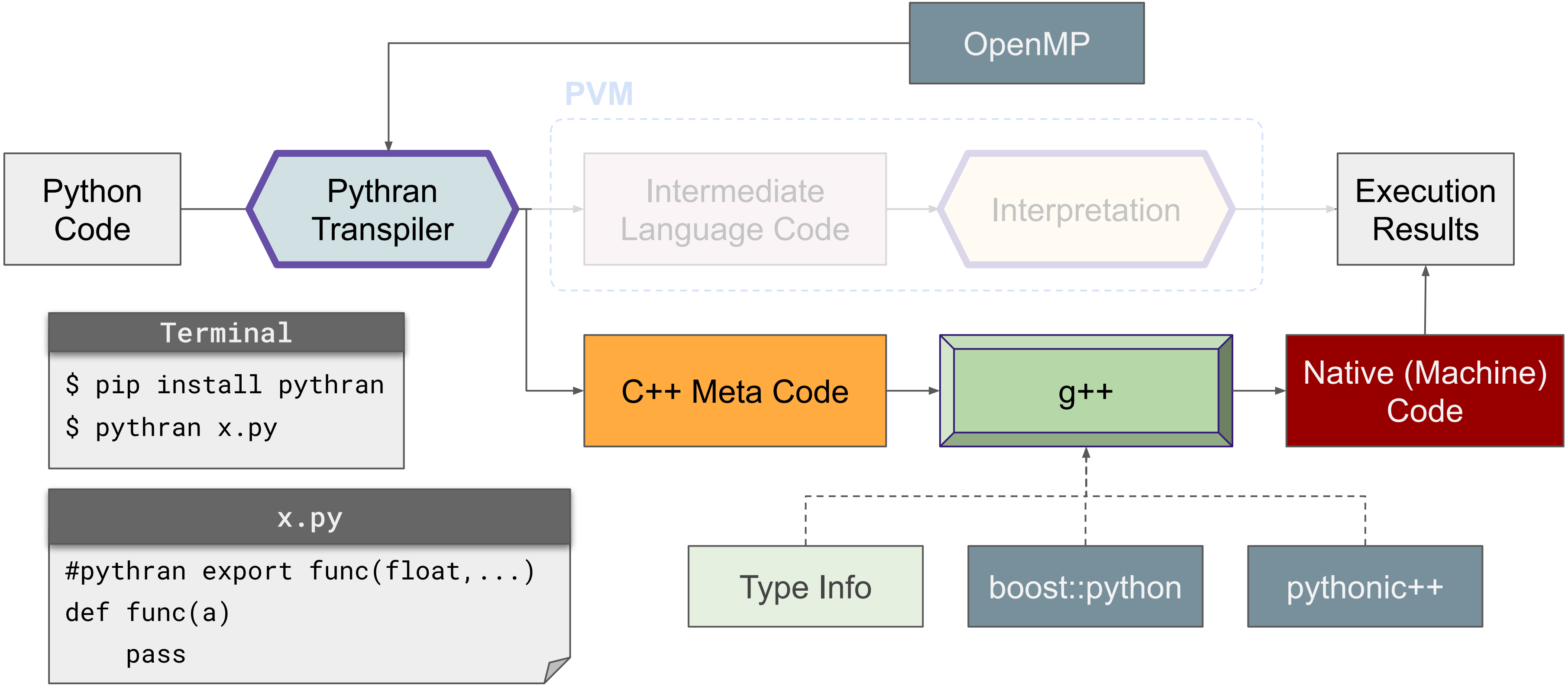
Outline for today

- Do we need to learn?
- Why Python is slow?
- How to speed Python up?
  - By AOT bindings
  - By JIT
  - By new interpreters
- Demos

Boosting the speed by AOT Compiler


External Libs
PVM
Source Code
Compilation
Intermediate Language Code
Interpretation
Execution Results
Python Interpreter is doing the compilation+transpilation work.
High-Performance Meta Code
Ahead-Of-Time Compilation
Native (Machine) Code
libs.o
modules.so

Pythran: an AOT compiler for a subset of the Python



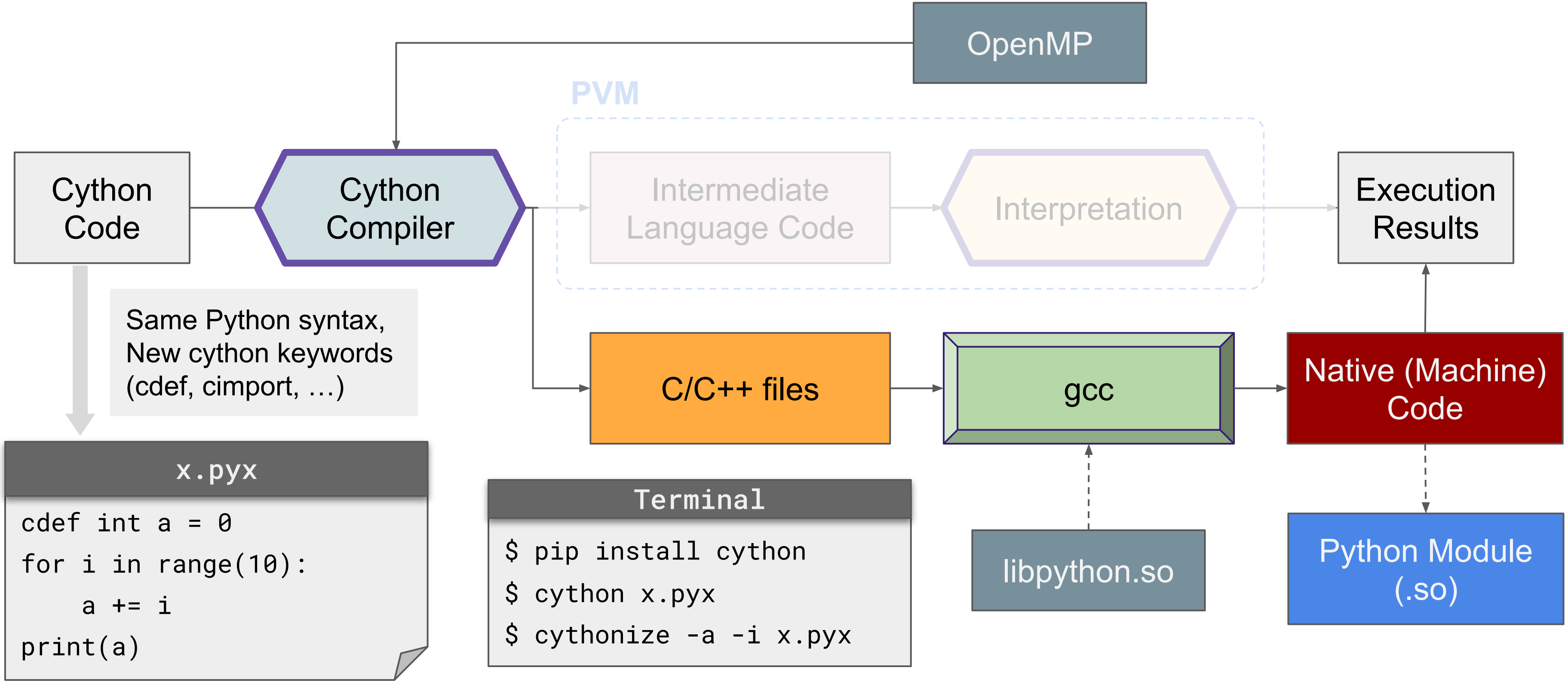
Terminal

```
$ pip install pythran
$ pythran x.py
```

x.py

```
#pythran export func(float,...)
def func(a)
    pass
```

Cython: Compiler to write C extensions for Python



Same Python syntax, New cython keywords (cdef, cimport, …)

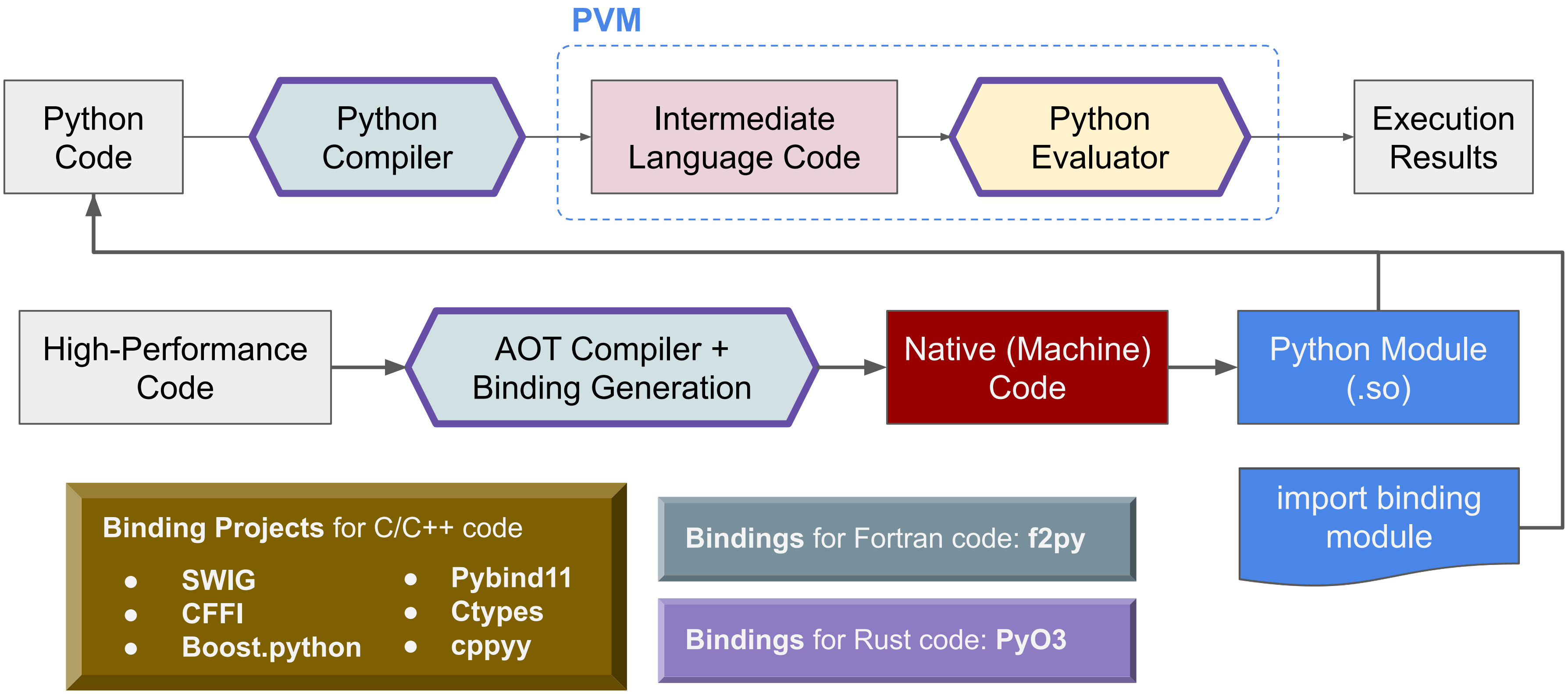
x.pyx

```
cdef int a = 0
for i in range(10):
    a += i
print(a)
```

Terminal

```
$ pip install cython
$ cython x.pyx
$ cythonize -a -i x.pyx
```

Binding ideas for adopting high-performance languages



PVM

Python Code

Python Compiler

Intermediate Language Code

Python Evaluator

Execution Results

High-Performance Code

AOT Compiler + Binding Generation

Native (Machine) Code

Python Module (.so)

import binding module

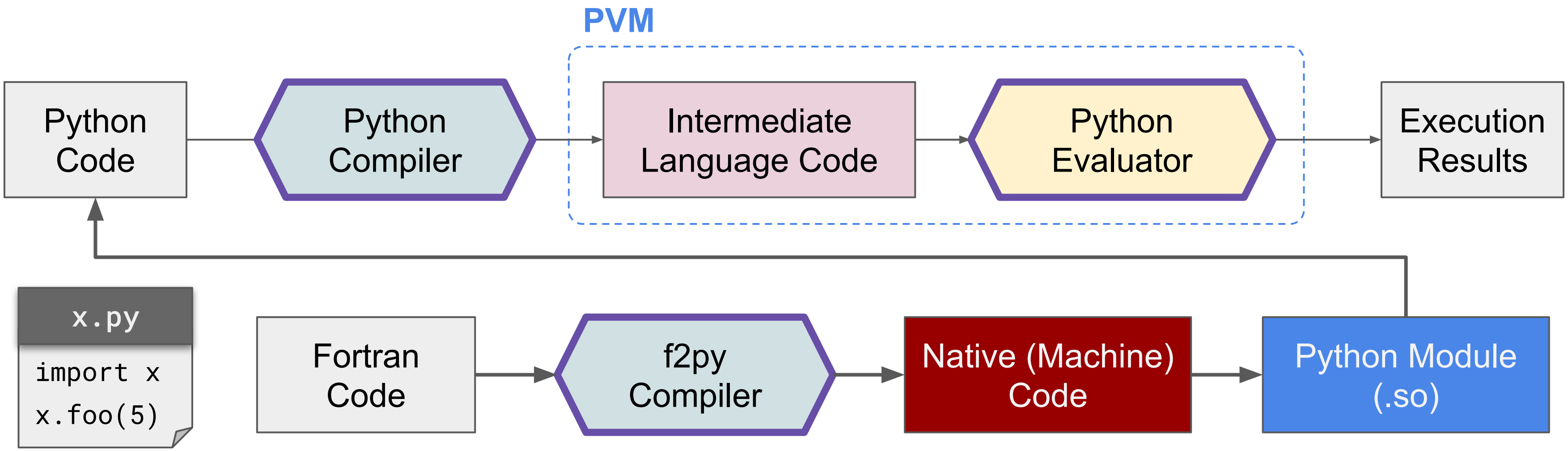
Binding Projects for C/C++ code

- **SWIG**
- **CFFI**
- **Boost.python**
- **Pybind11**
- **Ctypes**
- **cppyy**

Bindings for Fortran code: **f2py**

Bindings for Rust code: **PyO3**

f2py: wrap/bind fortran code for use in Python



x.py

```
import x
x.foo(5)
```

x.f

```
subroutine foo (n)
integer n
end
```

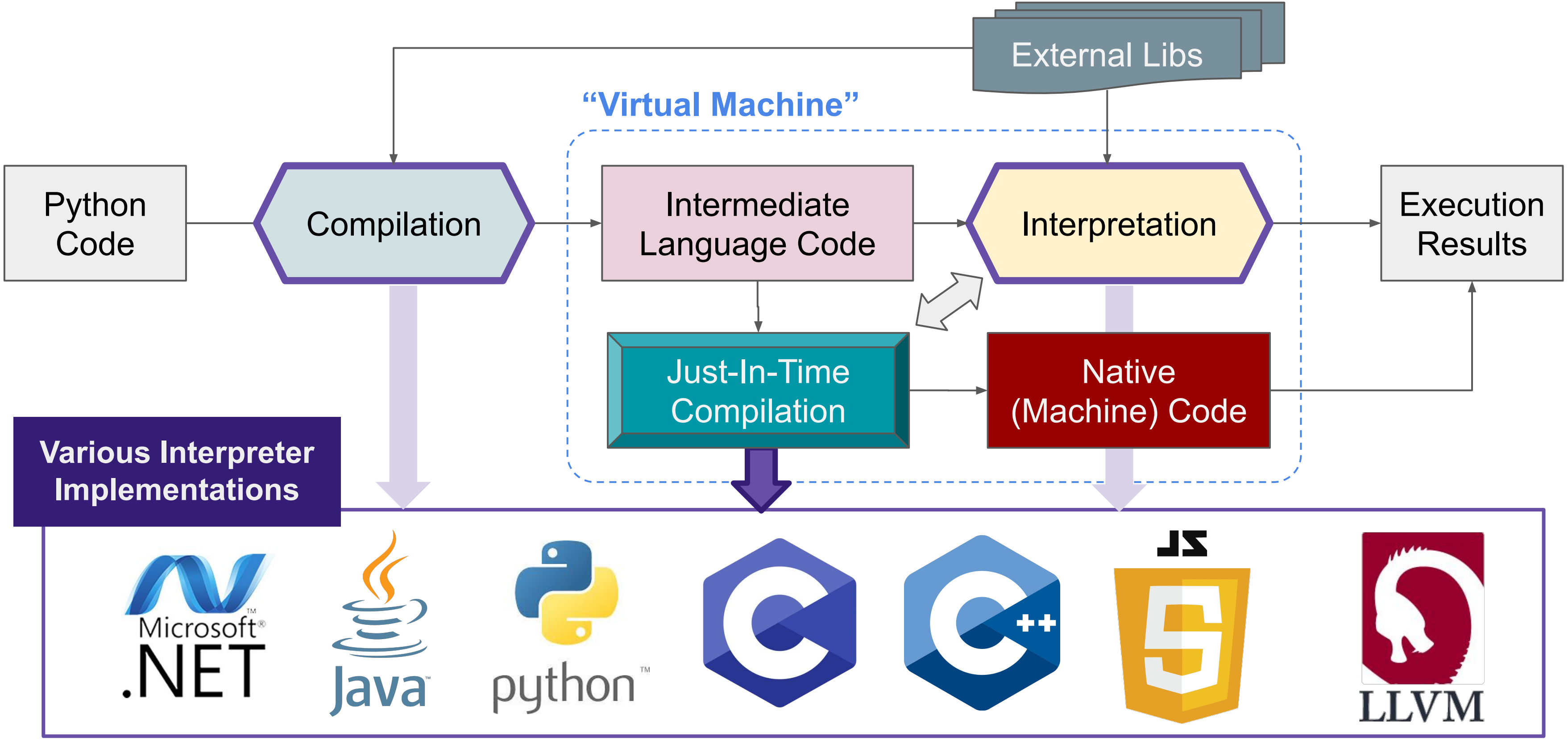
Terminal

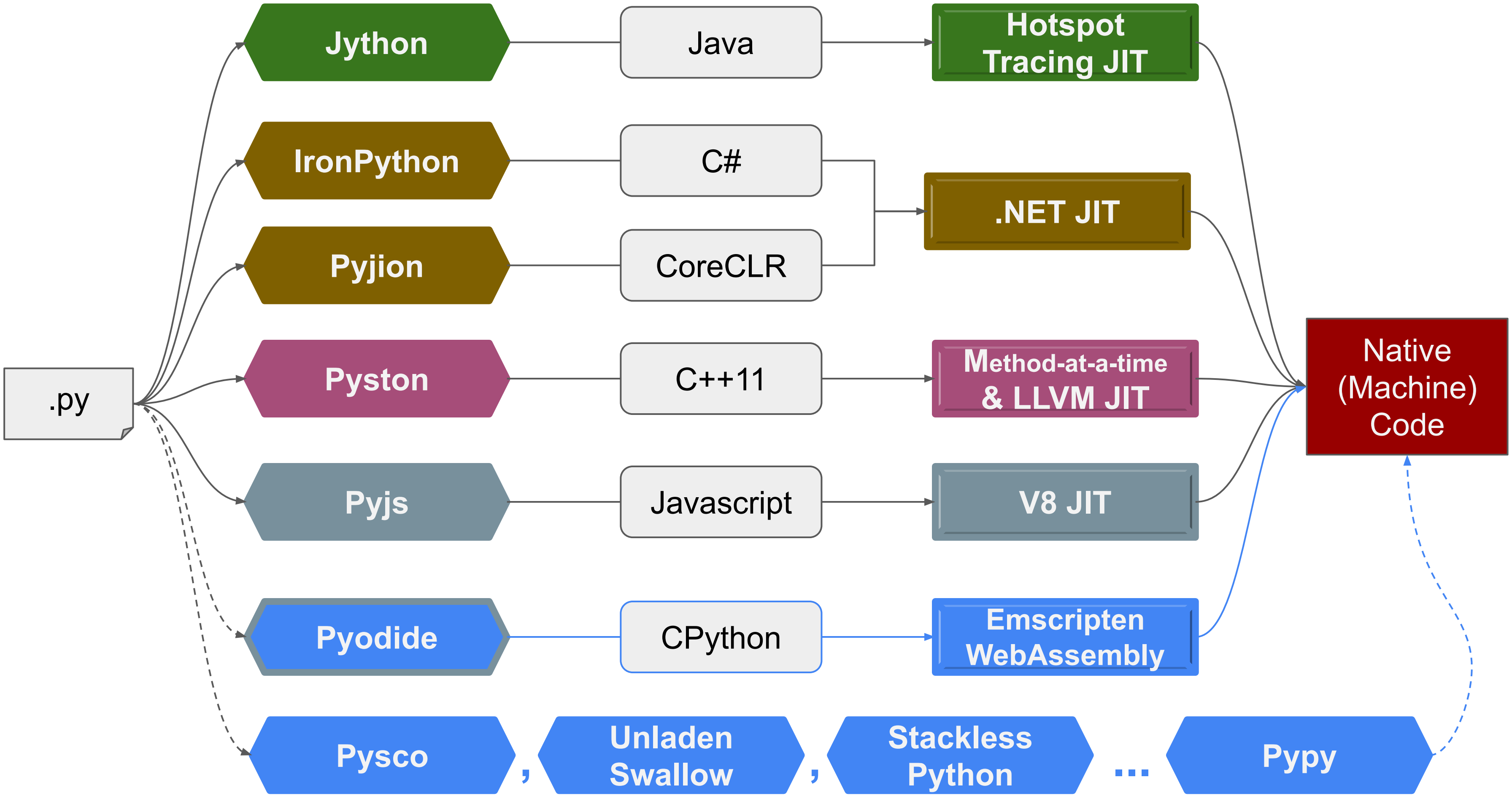
```
$ pip install f2py
$ f2py -c -m x x.f
```

Outline for today

- Do we need to learn?
- Why Python is slow?
- How to speed Python up?
  - By AOT bindings
  - By JIT
  - By new interpreters
- Demos

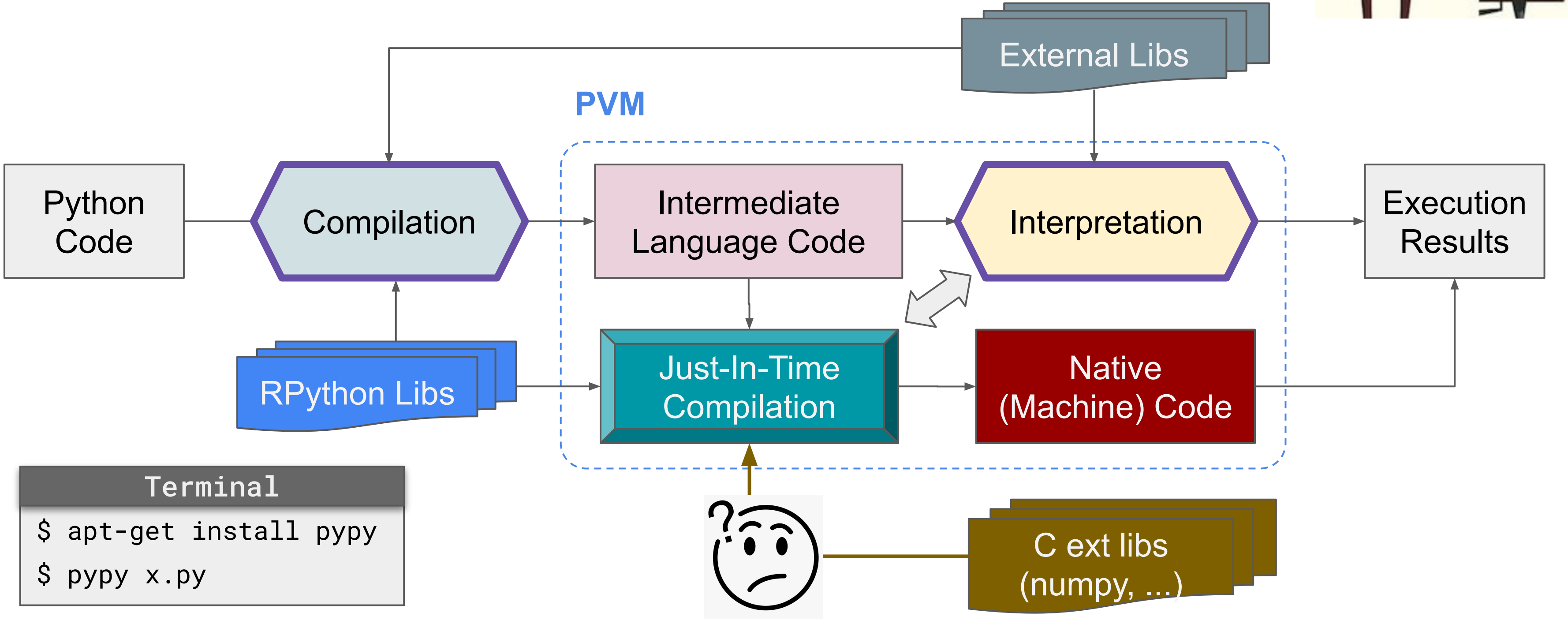
Boosting the speed by JIT




.py
Jython
Java
Hotspot Tracing JIT
IronPython
C#
.NET JIT
Pyjion
CoreCLR
Pyston
C++11
Method-at-a-time & LLVM JIT
Native (Machine) Code
Pyjs
Javascript
V8 JIT
Pyodide
CPython
Emscripten WebAssembly
Pysco
,
Unladen Swallow
,
Stackless Python
...
Pypy

Pypy: using Python to interpret Python

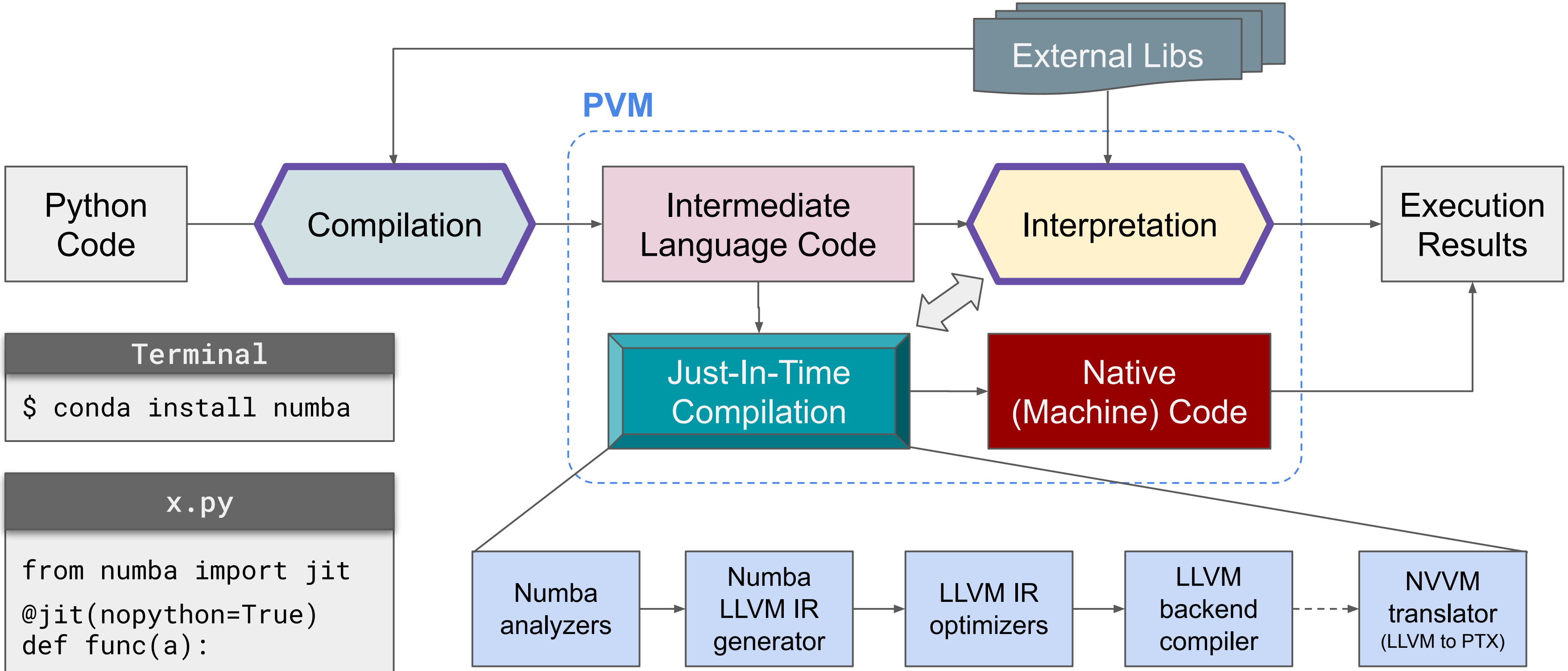
- RPython = Restricted/Reduced Python



Terminal

```
$ apt-get install pypy
$ pypy x.py
```

Numba: a high-performance python JIT compiler



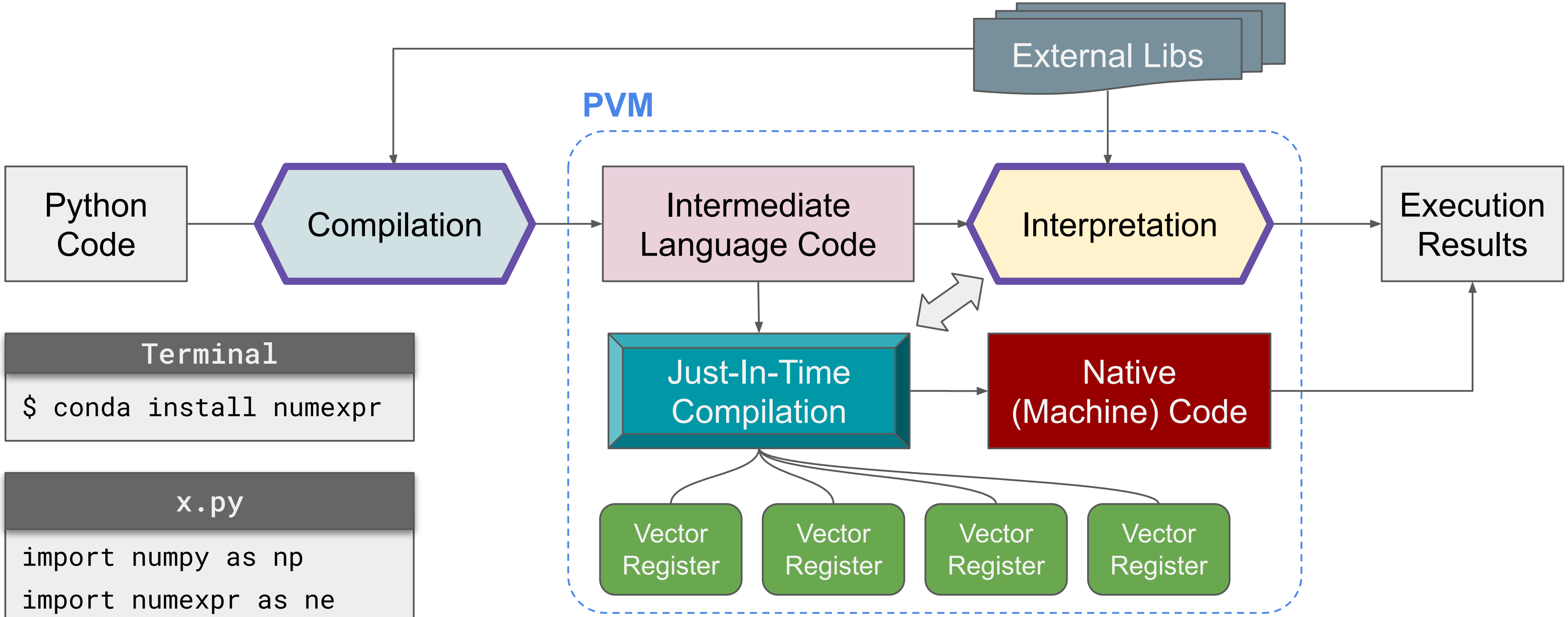
Terminal

```
$ conda install numba
```

x.py

```
from numba import jit
@jit(nopython=True)
def func(a):
    ...
```

NumExpr: C-based JIT booster for numpy large arrays



Terminal

```
$ conda install numexpr
```

x.py

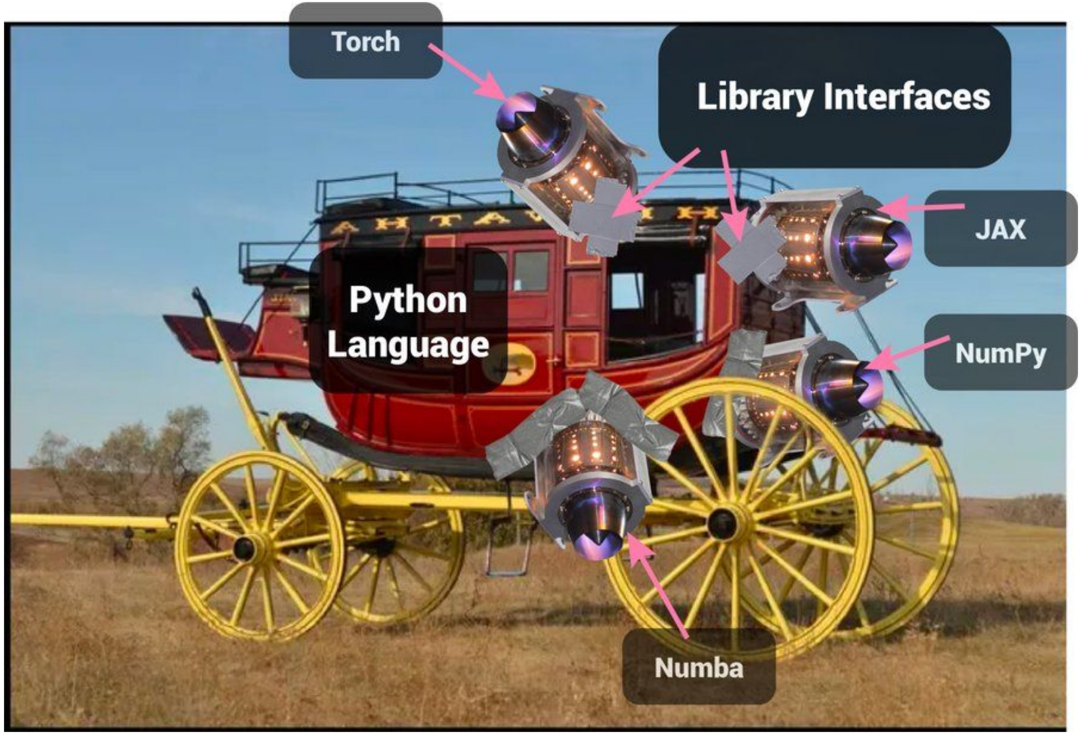
```
import numpy as np
import numexpr as ne
a = np.arange(1e6)
ne.evaluate("a + 1")
```

No temp arrays, Multi-threading, Intel MKL supported

Outline for today

- Do we need to learn?
- Why Python is slow?
- How to speed Python up?
  - By AOT bindings
  - By JIT
  - By new interpreters
- Demos

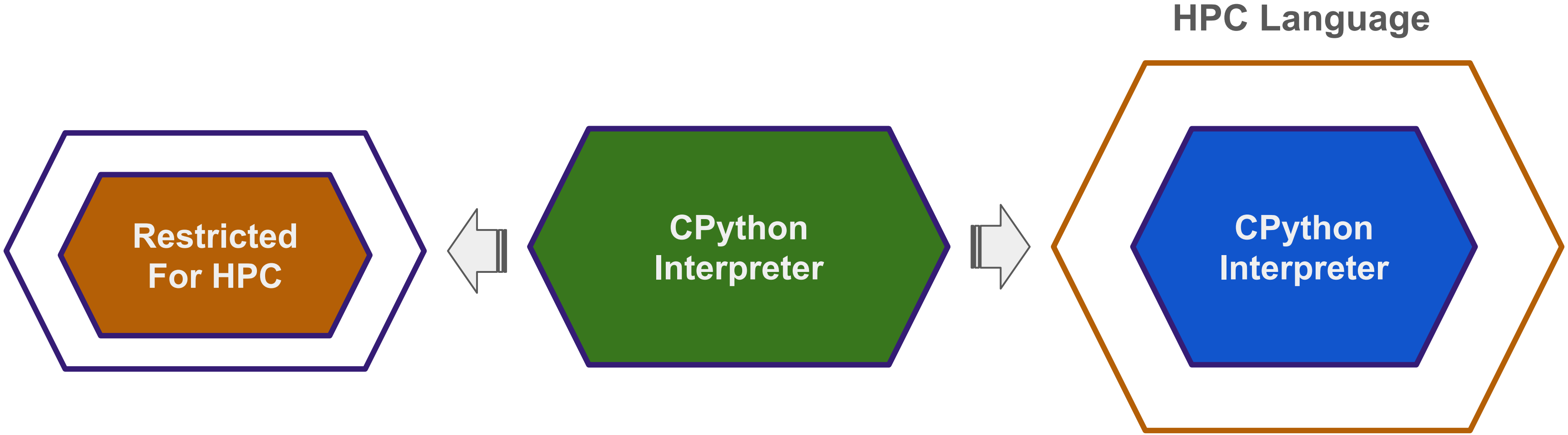
This is what we are having right now 🤭



Downsides in the “Glue-and-Patch” Approach

- Difficult in learning the implementation even for experienced devs
- Difficult in debugging, profiling or resolving performance problems
- No effective parallel processing way in Python
- Special challenges in AI ages:
    - Fundamental limitations of sophisticated compilation backend to create high performance implementation of Python code, even in Pytorch’s compile()
    - Unavoidable performance bottlenecks when calling a bunch of compiled functions
    - A faster implementation for deployment has no guarantee to run identically to its python version (e.g. using ONNX or Torchscript)

From another perspective



Subsets to Python

- Pypy
- Pythran
- JAX

New CPython

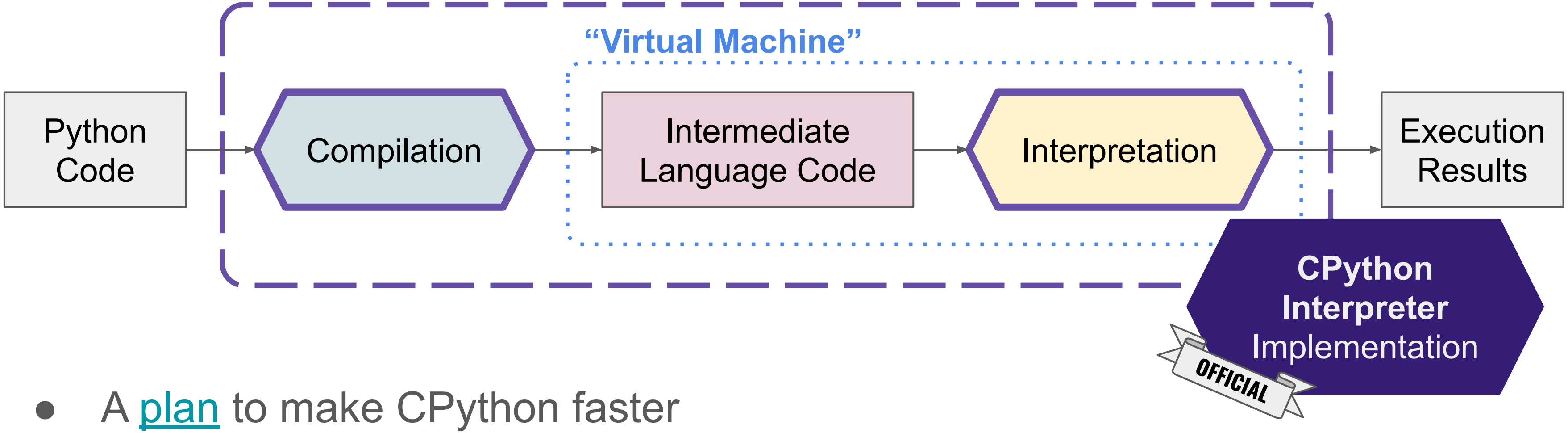
❖ Fully optimization

❖ Rewritten, maybe?

Supersets to Python

- Cython
- Mojo

The “Shannon Plan”



“Virtual Machine”

Python Code

Compilation

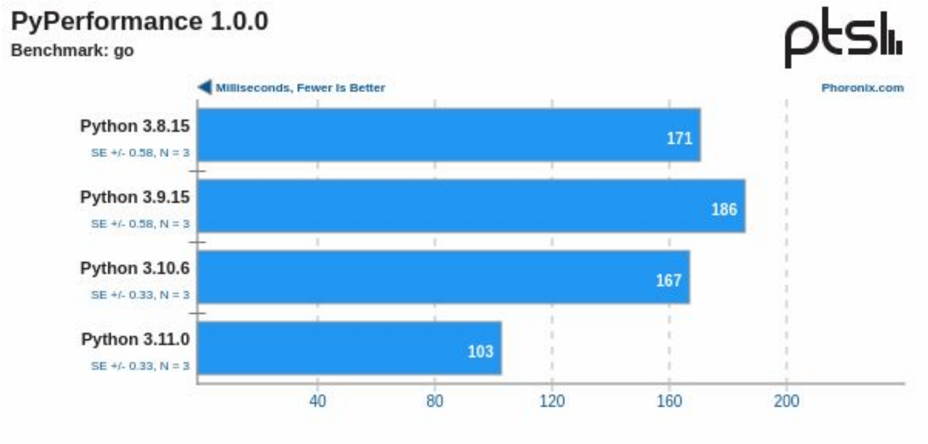
Intermediate Language Code

Interpretation

Execution Results

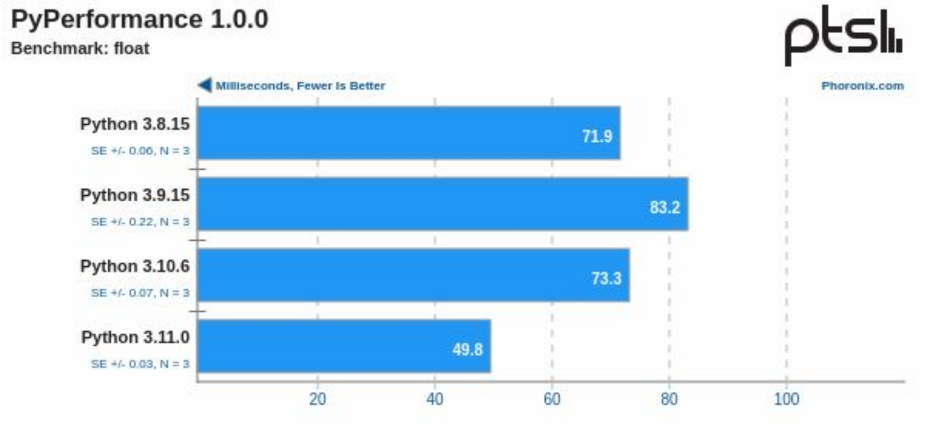
CPython Interpreter Implementation

- A plan to make CPython faster
  - Originally proposed by Eric Snow, and Mark Shannon in 2020
  - Guido van Rossum joined and gave a talk in Python Language Summit (May 2021)
  - Based on the experience with “HotPy” and “HoyPy 2”
  - Promising 5x in 4 years, 1.5x per year



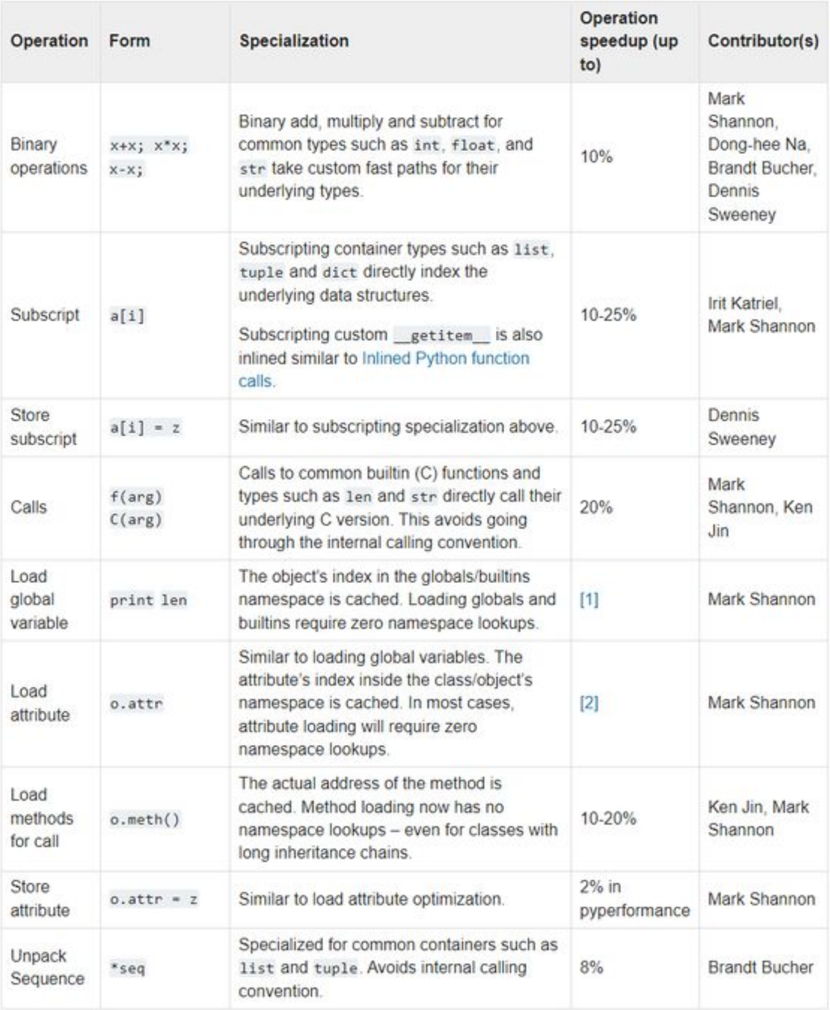
PyPerformance 1.0.0

Benchmark: go



PyPerformance 1.0.0

Benchmark: float



Operation	Form	Specialization	Operation speedup (up to)	Contributor(s)
Binary operations	`x+x; x*x; x-x;`	Binary add, multiply and subtract for common types such as `int`, `float`, and `str` take custom fast paths for their underlying types.	10%	Mark Shannon, Dong-hee Na, Brandt Bucher, Dennis Sweeney
Subscript	`a[i]`	Subscripting container types such as `list`, `tuple` and `dict` directly index the underlying data structures. Subscripting custom `__getitem__` is also inlined similar to Inlined Python function calls.	10-25%	Irit Katriel, Mark Shannon
Store subscript	`a[i] = z`	Similar to subscripting specialization above.	10-25%	Dennis Sweeney

Calls	`f(arg)`
	`C(arg)`
		Calls to common builtin (C) functions and types such as `len` and `str` directly call their underlying C version. This avoids going through the internal calling convention.	20%	Mark Shannon, Ken Jin

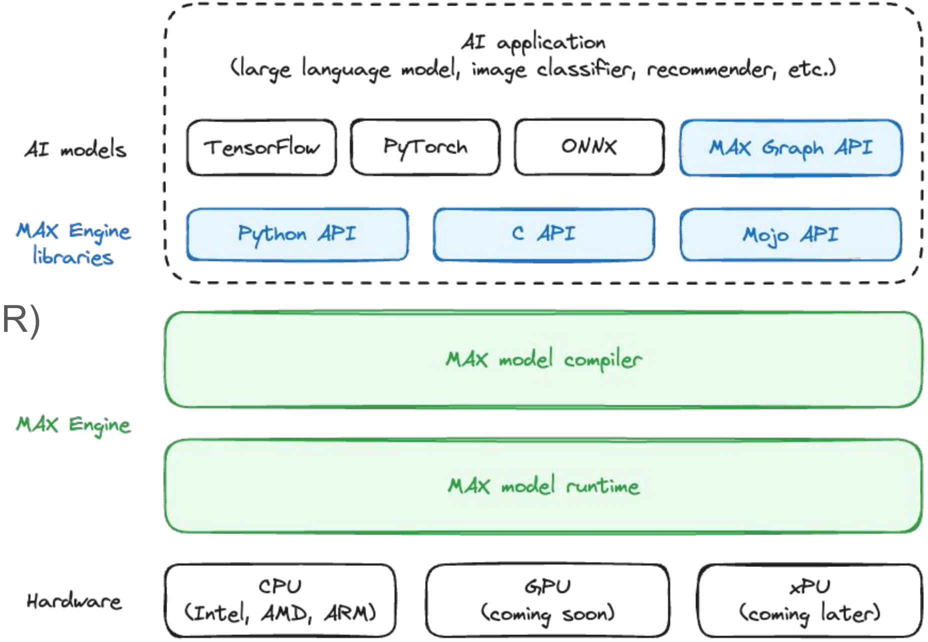
Load global variable	`print len`	The object's index in the globals/builtins namespace is cached. Loading globals and builtins require zero namespace lookups.	[1]	Mark Shannon
Load attribute	`o.attr`	Similar to loading global variables. The attribute's index inside the class/object's namespace is cached. In most cases, attribute loading will require zero namespace lookups.	[2]	Mark Shannon
Load methods for call	`o.meth()`	The actual address of the method is cached. Method loading now has no namespace lookups – even for classes with long inheritance chains.	10-20%	Ken Jin, Mark Shannon
Store attribute	`o.attr = z`	Similar to load attribute optimization.	2% in pyperformance	Mark Shannon
Unpack Sequence	`*seq`	Specialized for common containers such as `list` and `tuple`. Avoids internal calling convention.	8%	Brandt Bucher

Mojo == Python++ (?)

- A *new* language
    - Using Python as the syntax
    - High performanced
        - Little faster in plain python mode
        - Optional super-faster mode
            - Adding new syntax for devs
- Aiming to another challenge
    - Heterostructures in hardware
    - Based on “intermediate representation” (IR)
- As a block in a bigger picture
    - MAX engine framework
    - Possibly incremental adaptation

Mojo is promising, but its future is still not clear.

LANGUAGES	TIME (S) *	SPEEDUP VS PYTHON
Python 3.10.9	970 s	1x
SCALAR C++	0.11 s	9000x



Outline for today

- Do we need to learn?
- Why Python is slow?
- How to speed Python up?
  - By AOT bindings
  - By JIT
  - By new interpreters
- Demos

About Jupyter demo

bit.ly/hpdspy_01

- Code example will be running in Google Colab.
  - `IPython` (interpreter implementation) as Python kernel in Jupyter Notebook
    - Based on `CPython`, enhancing interactive features.
    - Shell prompted as `In [#]:`
    - Interacting with external files/modules by `%`*magic* commands
    - Some comparisons were not made in the same baseline.
    - An interesting project in web dev: **PyScript**
  - Colab comes with some installed libraries, but not all.
  - Performance benchmark was done based on array operations
    - Started with 1000 points in 3 dimensions
    - Calculate the pairwise 1000x1000 distances
    - Arrays (containers, dataframes) will be our *main* subject to discuss in the next lecture.

bit.ly/hpdspy_01

COMPUTATION

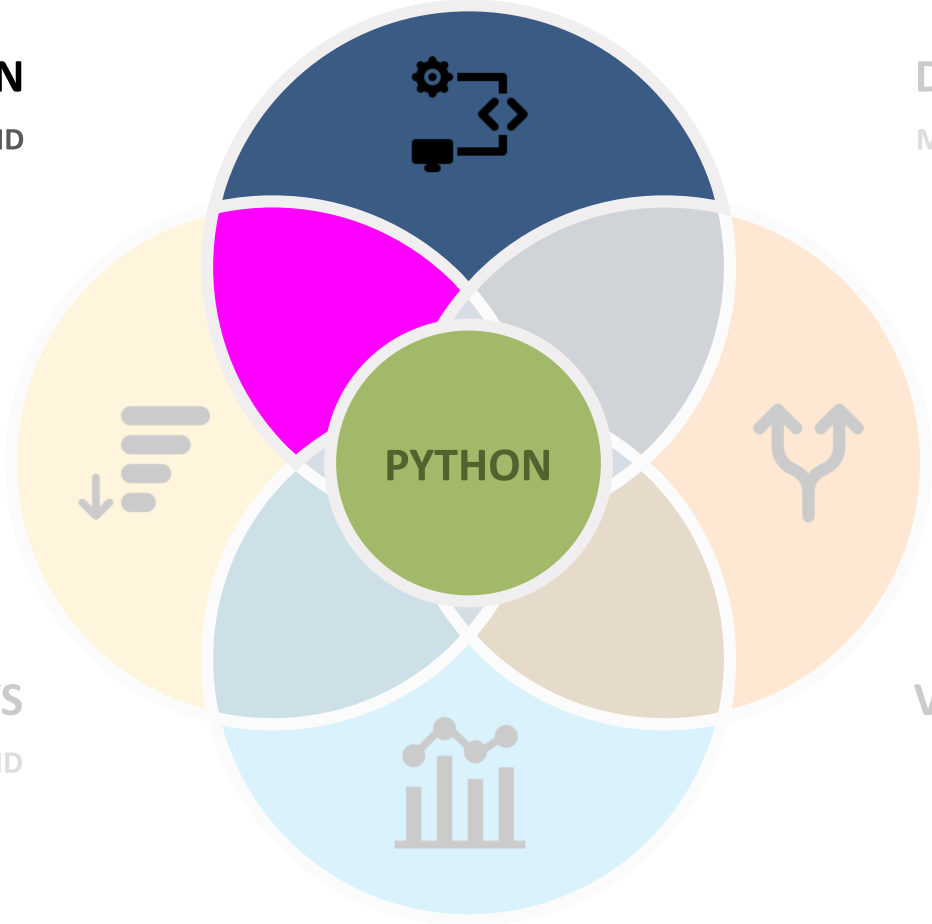
Single Node/GPU, SIMD

- Pypy, Numba, NumExpr
- Pythran, Cython
- F2py, ctypes

DISTRIBUTED

Multiple Nodes/Machines

- MapReduce-based: PySpark, PyFlink
- MPI-based: mpi4py, Horovod
- Joblib, Dask, Ray



DATA ARRAYS

Single Node/GPU, SIMD

- Numpy
- Pandas, Polars
- Modin, Pandarallel, Swifter
- Dask DataFrame, Vaex

VISUALIZATION

- Viz process for big data
- Matplotlib, Bokeh, Plotly
- Holoview and Datashader
- Traited VTK, Mayavi, Paraview

COMPUTATION

Single Node/GPU, SIMD

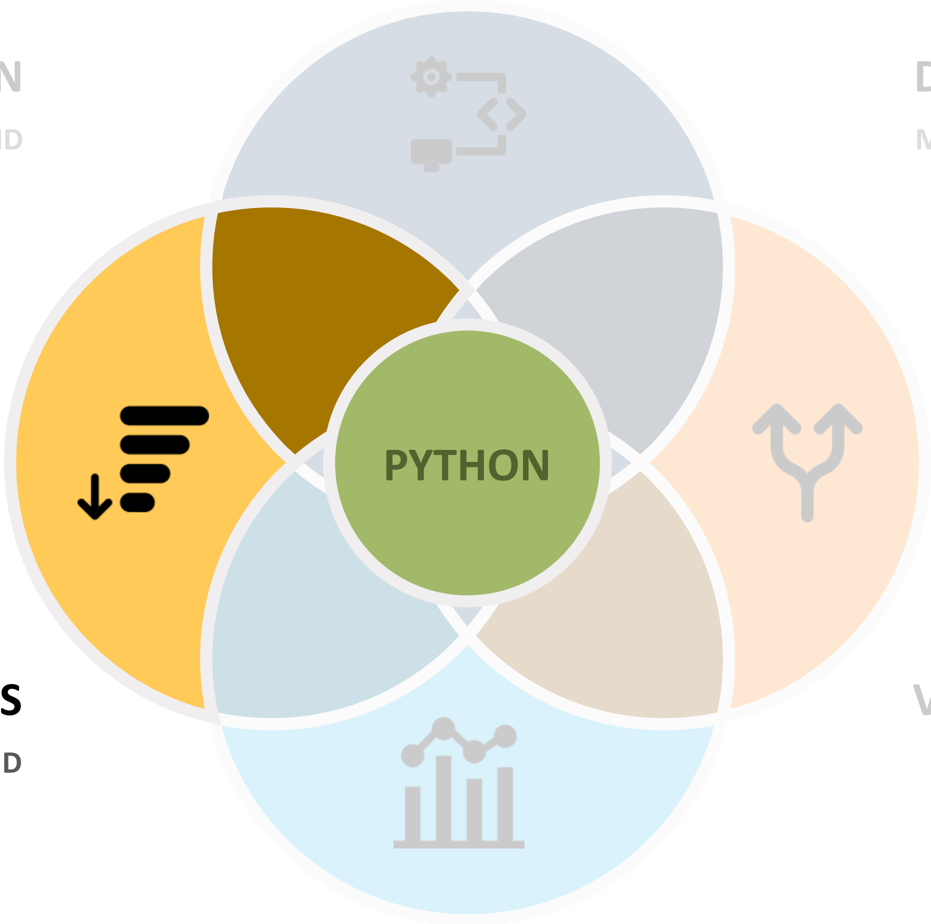
- Pypy, Numba, NumExpr
- Pythran, Cython
- F2py, ctypes

See you next week!

DATA ARRAYS

Single Node/GPU, SIMD

- Numpy
- Pandas, Polars
- Modin, Pandarallel, Swifter
- Dask DataFrame, Vaex



DISTRIBUTED

Multiple Nodes/Machines

- MapReduce-based: PySpark, PyFlink
- MPI-based: mpi4py, Horovod
- Joblib, Dask, Ray

VISUALIZATION

- Viz process for big data
- Matplotlib, Bokeh, Plotly
- Holoview and Datashader
- Traited VTK, Mayavi, Paraview