

Python for High Performance Data Analytics

—— (1) Computation ——

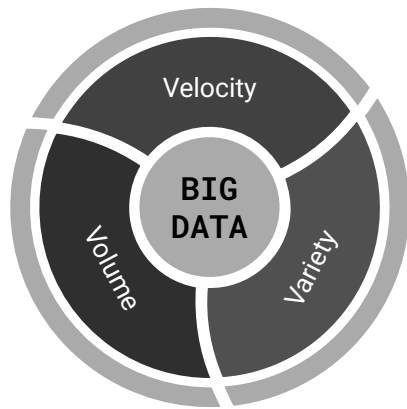
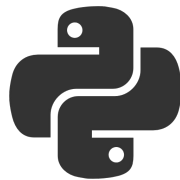
Qiyang Hu

UCLA Office of Advanced Research Computing

Feb 21, 2025

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














<https://github.com/huqy/HighPerfDataSciPython>

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

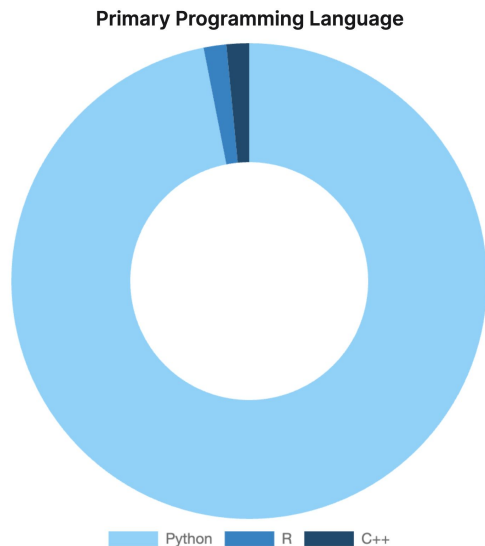
Python is *very* popular.

Ranking by counting hits of the most popular search engines

Feb 2025	Feb 2024	Change	Programming Language		Ratings	Change
1	1			Python	23.88%	+8.72%
2	3	▲		C++	11.37%	+0.84%
3	4	▲		Java	10.66%	+1.79%
4	2	▼		C	9.84%	-1.14%
5	5			C#	4.12%	-3.41%
6	6			JavaScript	3.78%	+0.61%
7	7			SQL	2.87%	+1.04%
8	8			Go	2.26%	+0.53%
9	12	▲		Delphi/Object Pascal	2.18%	+0.78%
10	9	▼		Visual Basic	2.04%	+0.52%
11	11			Fortran	1.75%	+0.35%

Latest [TIOBE Index](#)

Choice by data-sci practitioners



[ML_Contests Report 2023](#)

Python is *very* 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 still need to learn?

Learning it can help us

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



Better & Efficient Prompts



**Beyond the
Zero/Few Shots**



**Performance tuning is a
dangerous 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
<ul style="list-style-type: none">● From an end users' perspective	<ul style="list-style-type: none">● From a package developers' perspective
<ul style="list-style-type: none">● A <i>BIGGER</i>-picture review on the selected 3rd-party python libraries	<ul style="list-style-type: none">● Native Python tricks (e.g. container, lazy eval, mem)● Line-by-line explanations on these library interfaces
<ul style="list-style-type: none">● Demos on specific example problems	<ul style="list-style-type: none">● 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 FRAMES

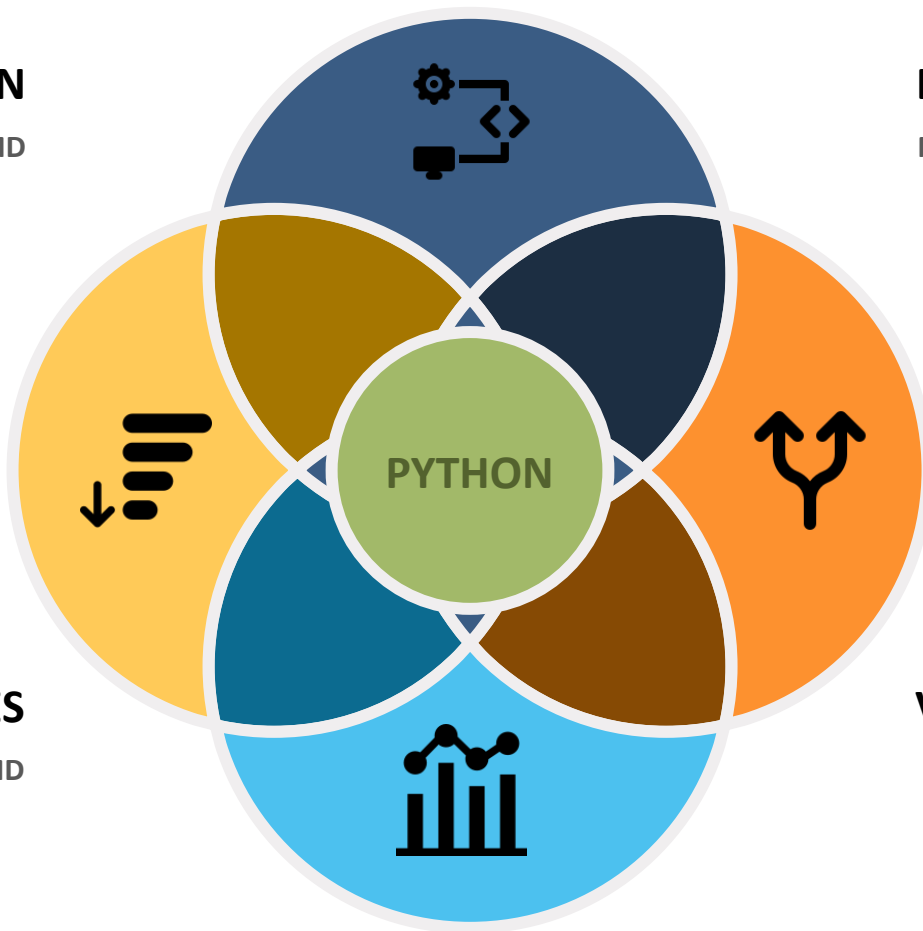
Single Node/GPU, SIMD

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



VISUALIZATION

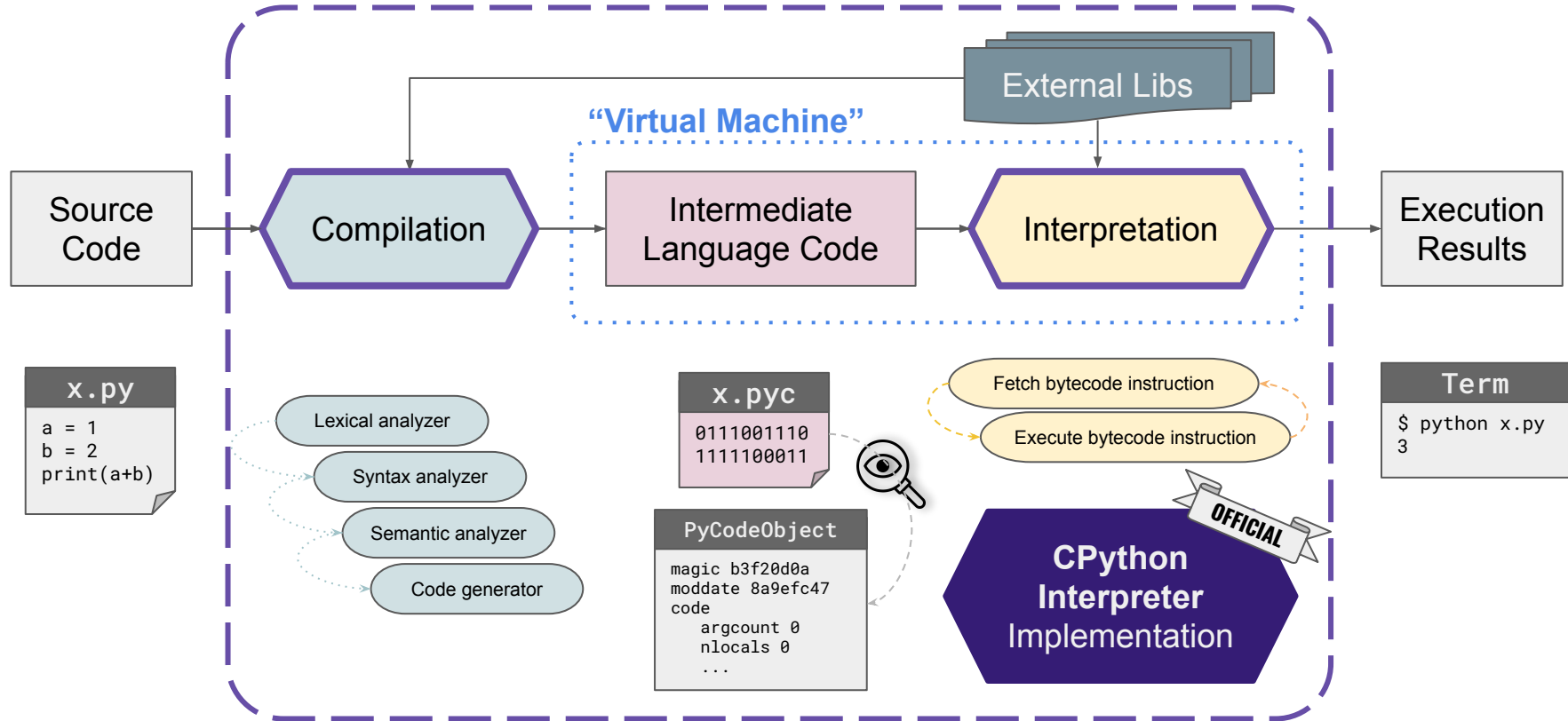
- Viz process for big data
- Matplotlib, Bokeh, Plotly
- Holoview and Datashader
- Traited VTK, Mayavi, Paraview



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?



Why Python is slow?

Python is Dynamically Typed rather than Statically Typed.

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

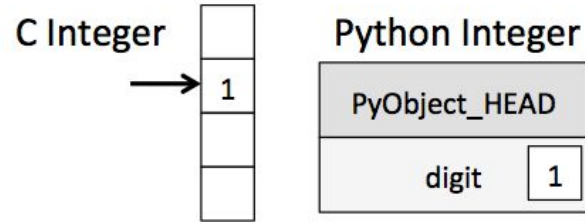
C Addition

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

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`

[Source](#)

GIL: Guilty or Gilly?

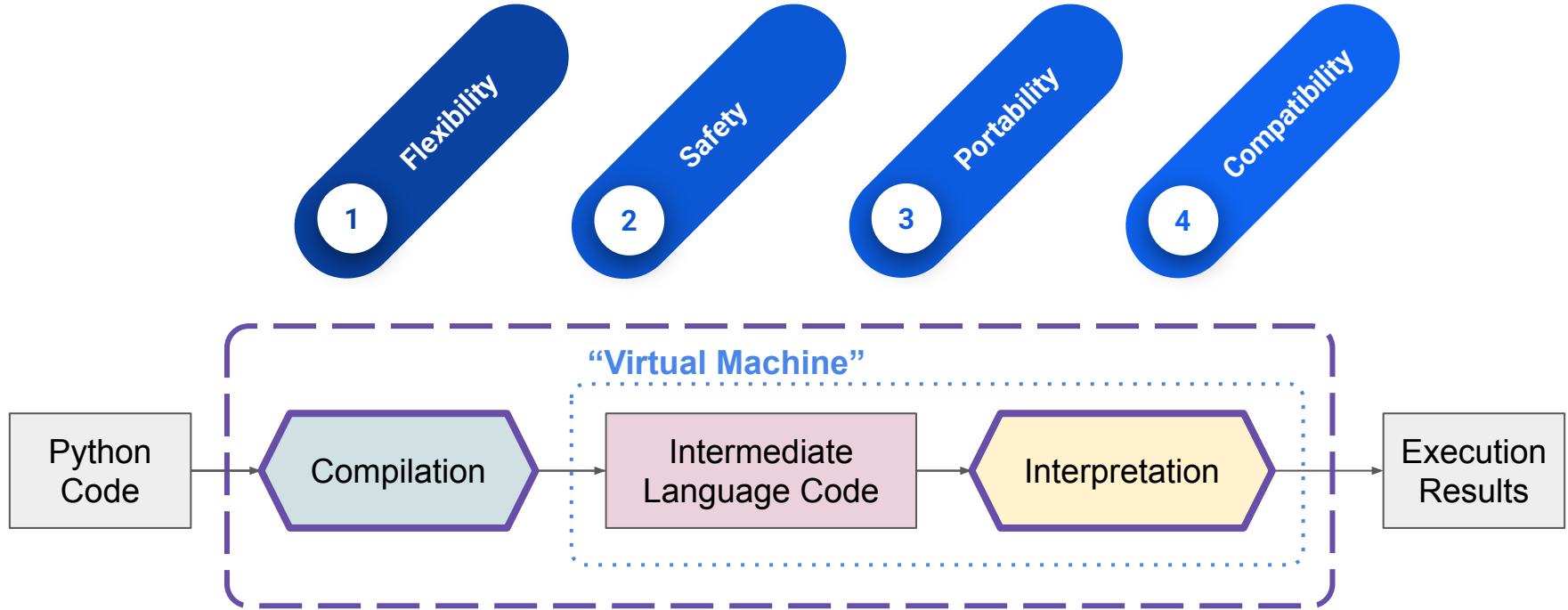


- GIL (Global Interpreter Lock)
 - A mutex (or a lock) that allows only one *thread* to hold the control of the Python interpreter.
- Why Python uses it?
 - GILs is added to the ref count variables to be kept protected from race conditions
 - GIL has performance benefits of GIL in single-threaded situation.
 - 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

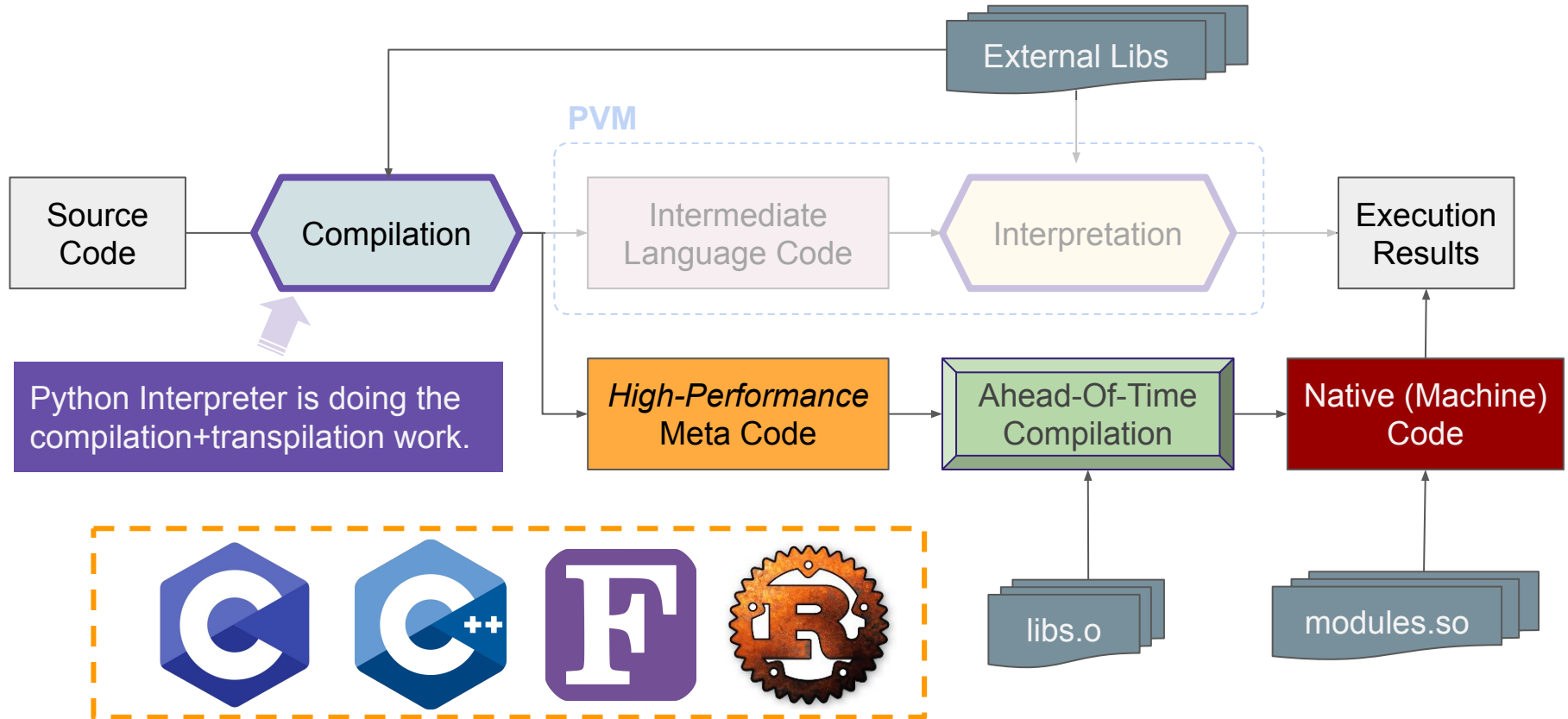
To speed up Python, it always means compromising



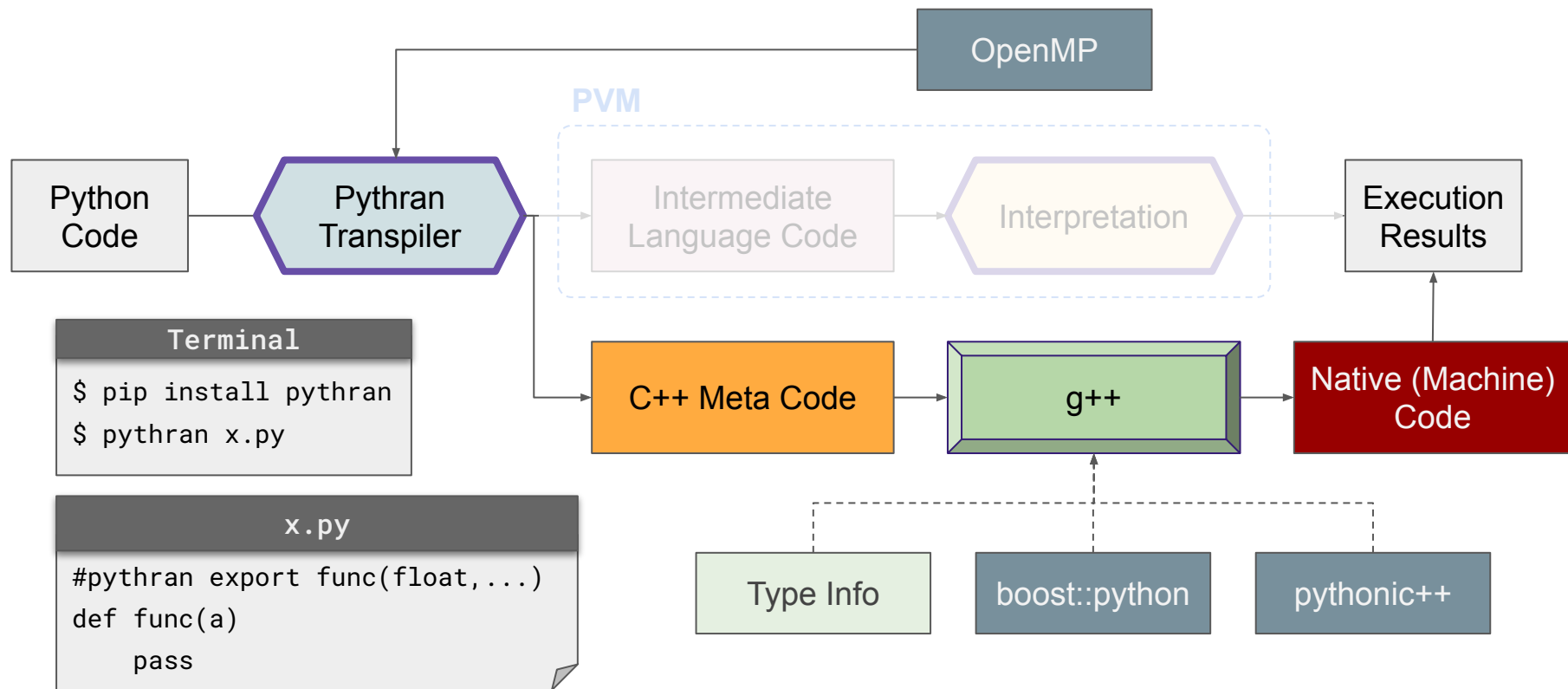
Outline for today

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

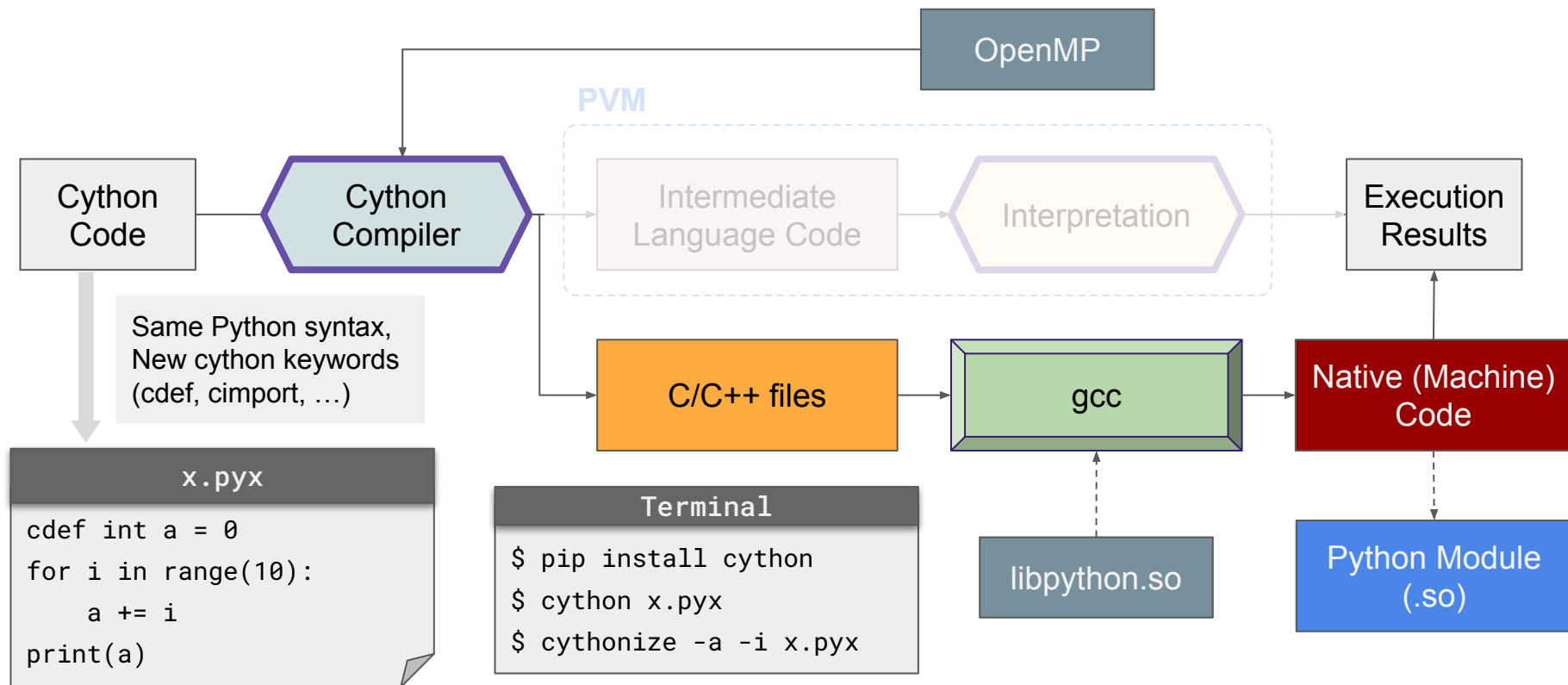
Boosting the speed by **AOT** Compiler



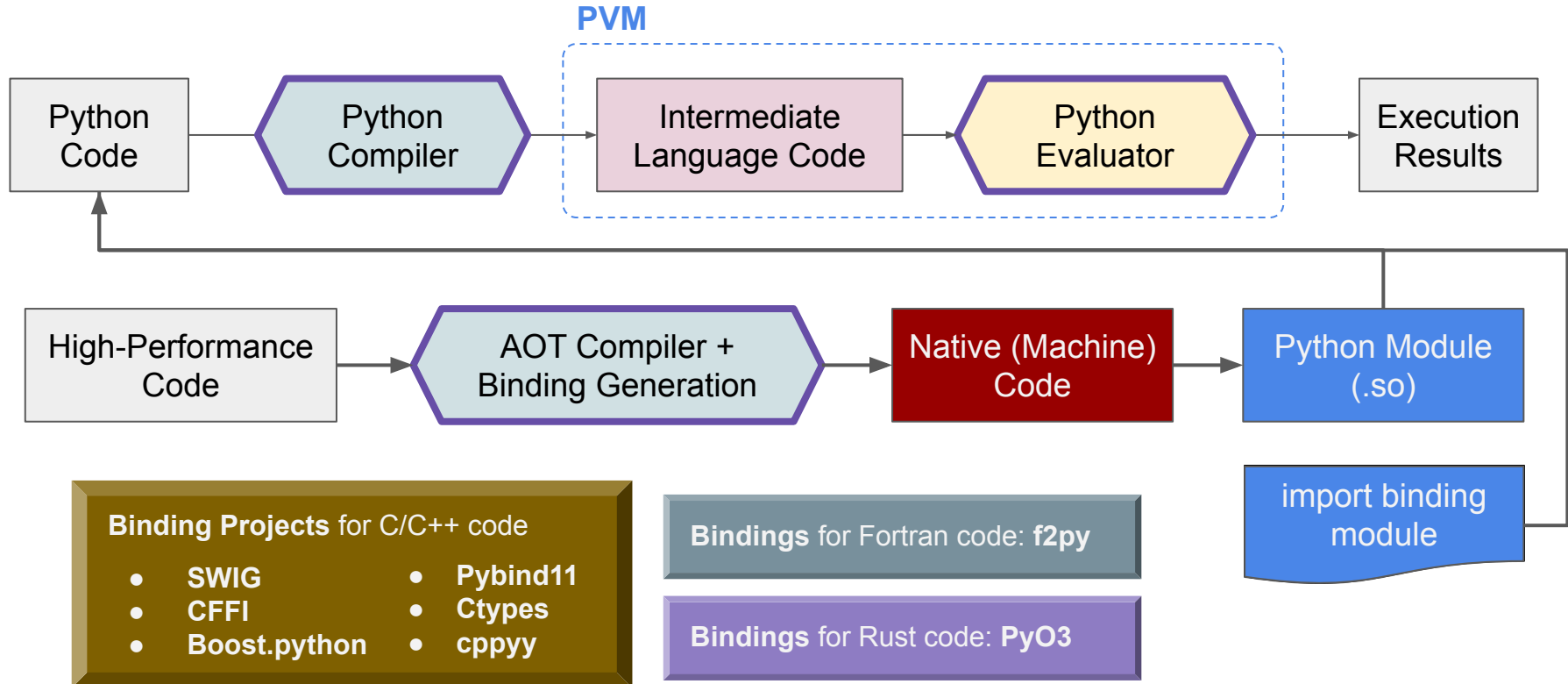
Pythran: an AOT compiler for a subset of the Python



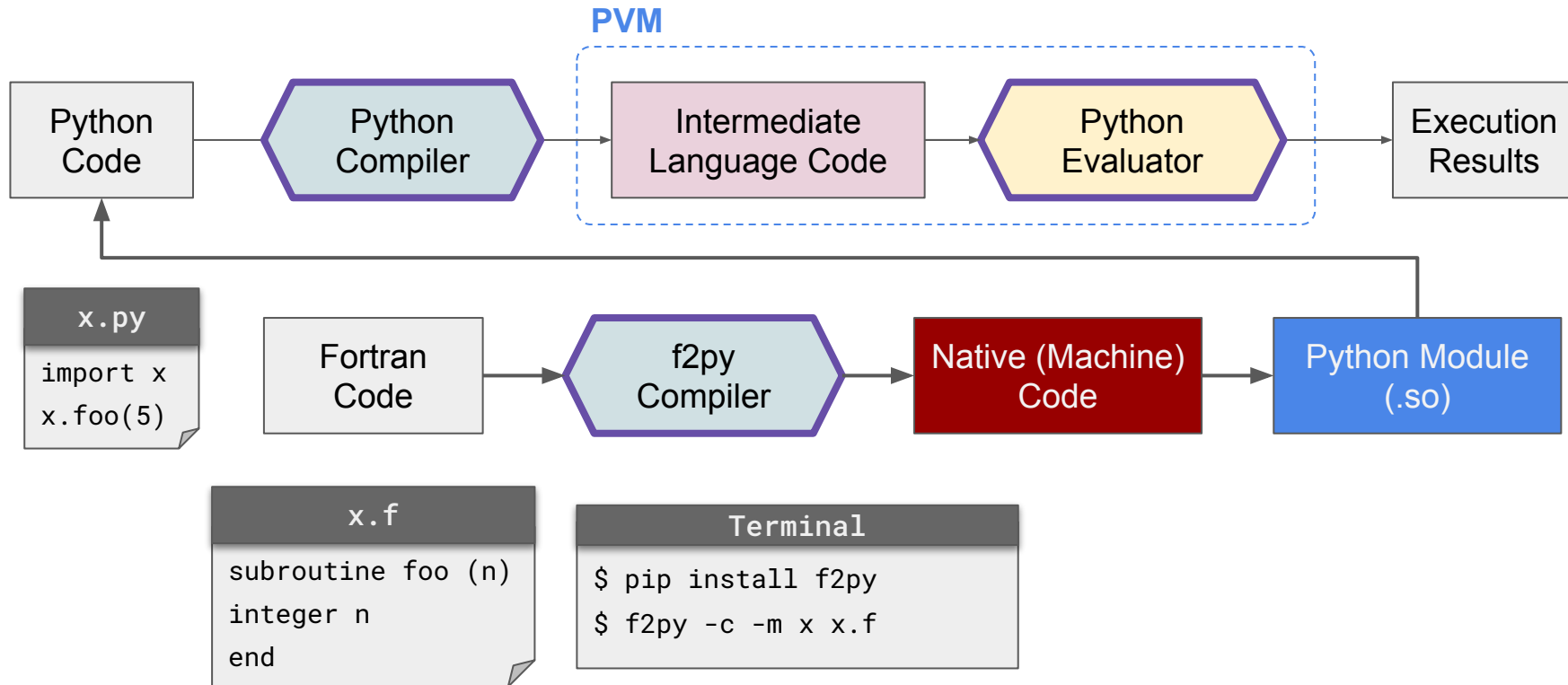
Cython: Compiler to write C extensions for Python



Binding ideas for adopting high-performance languages



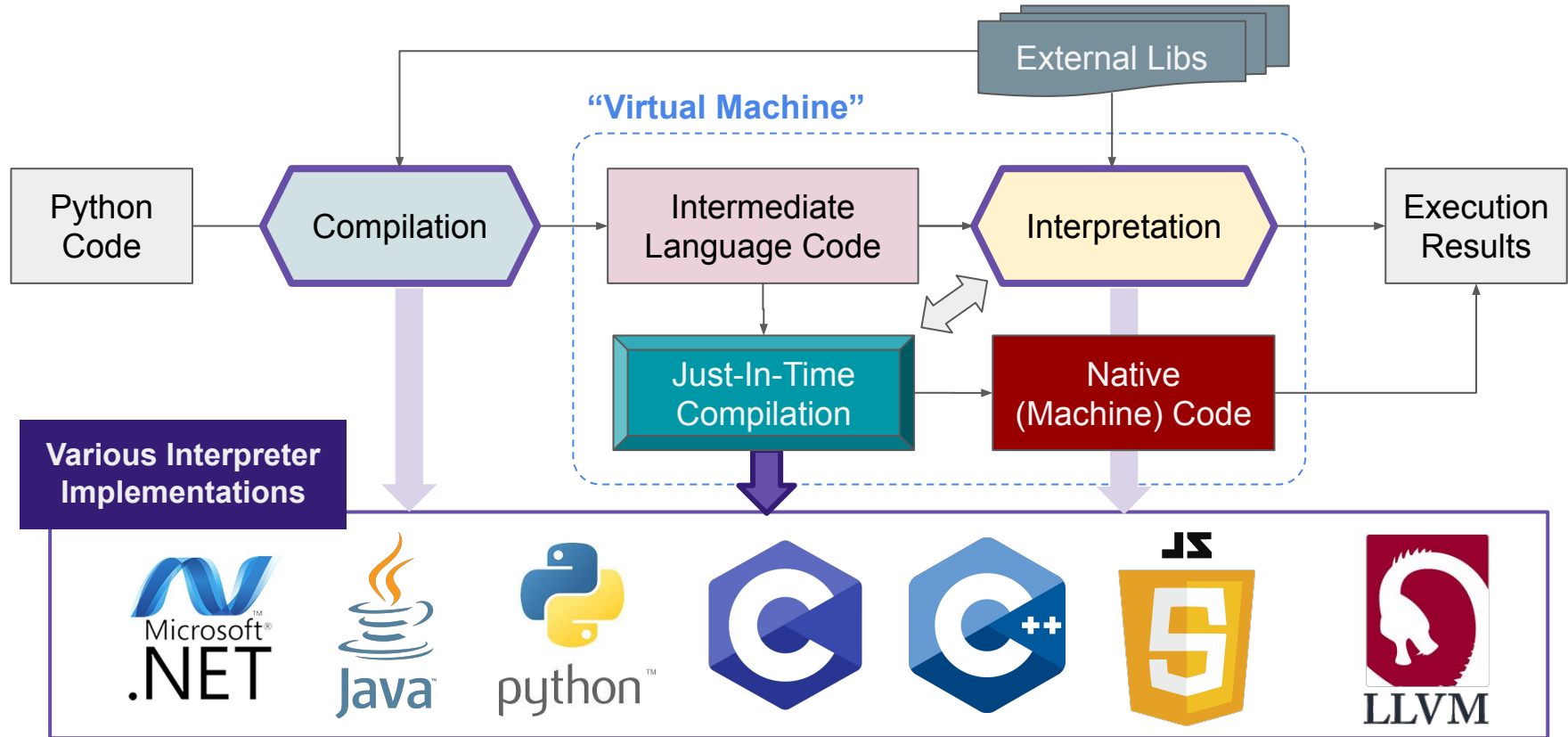
f2py: wrap/bind fortran code for use in Python

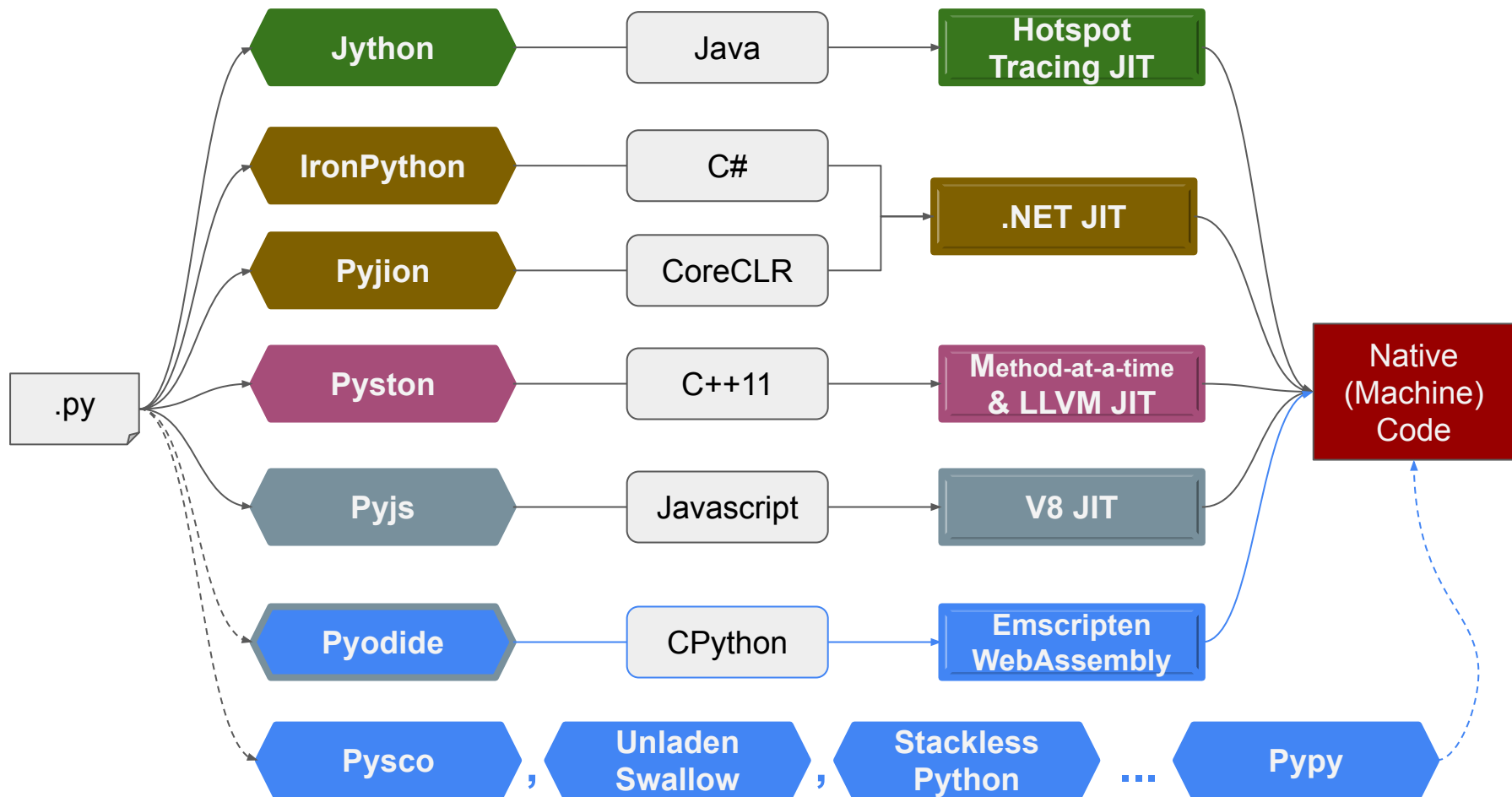


Outline for today

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

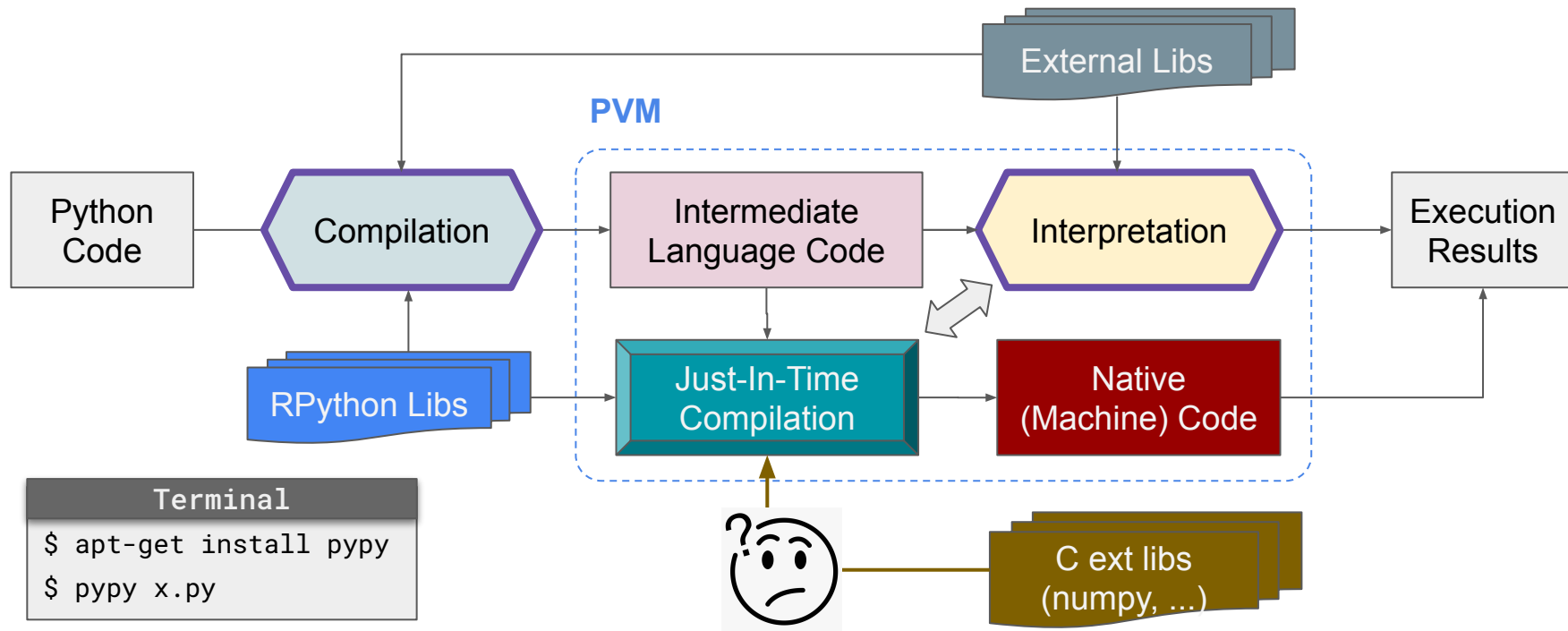
Boosting the speed by **JIT**



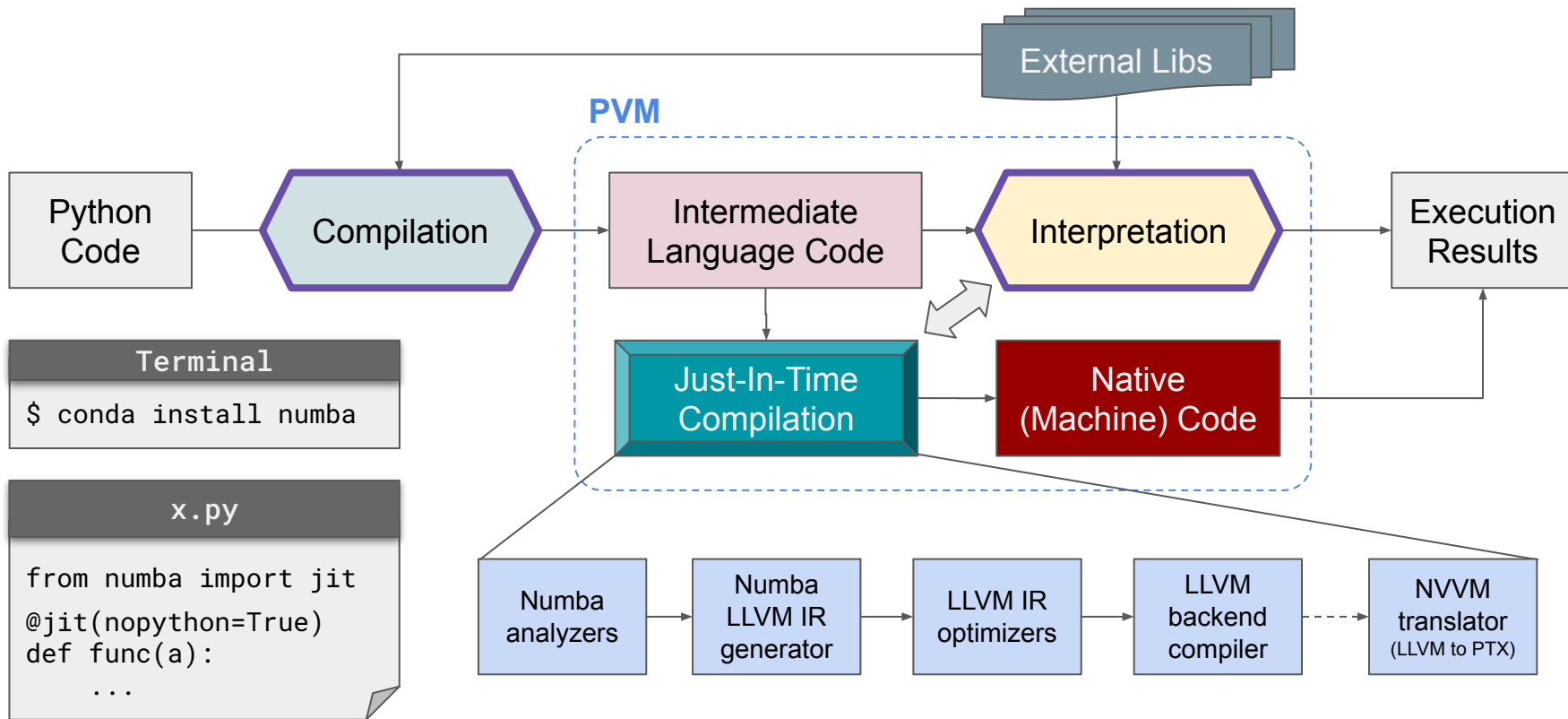


Pypy: using Python to interpret Python

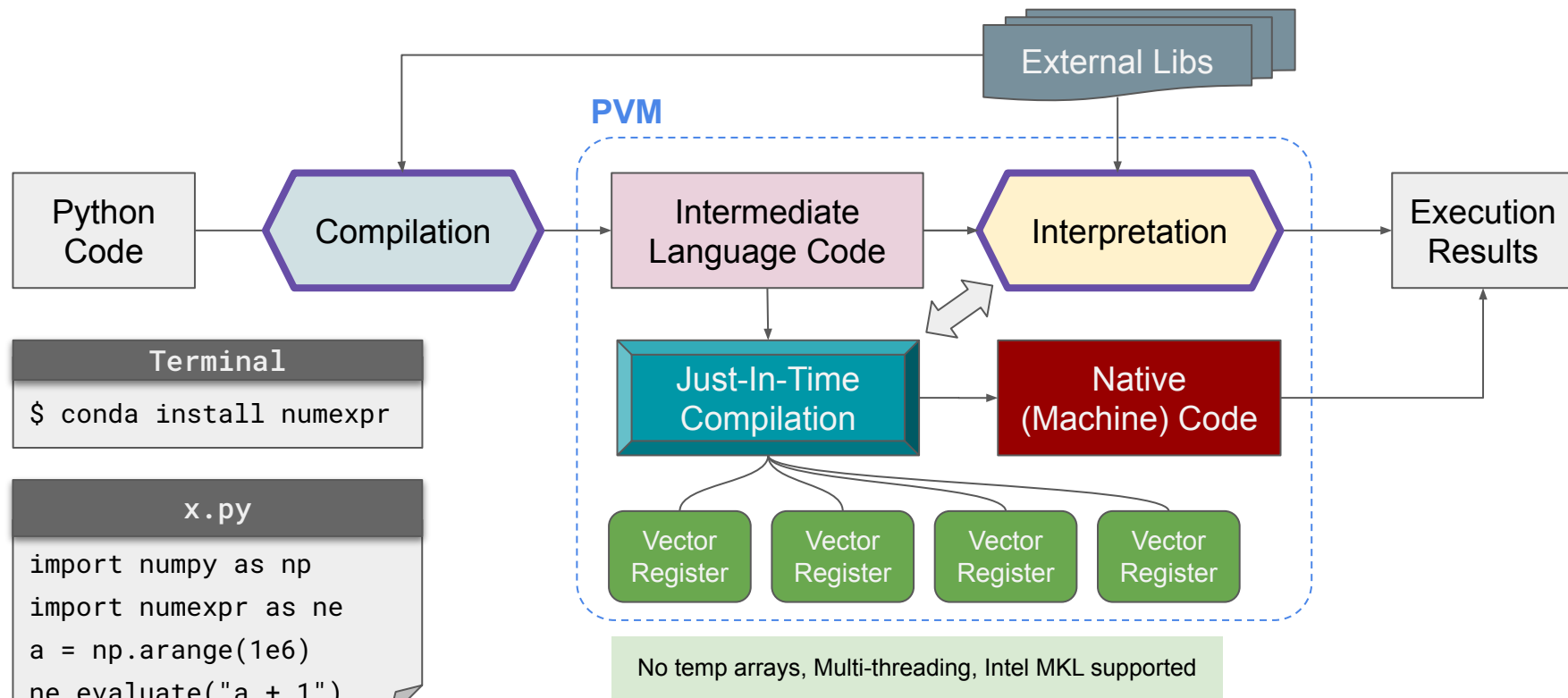
- RPython = Restricted/Reduced Python



Numba: a high-performance python JIT compiler



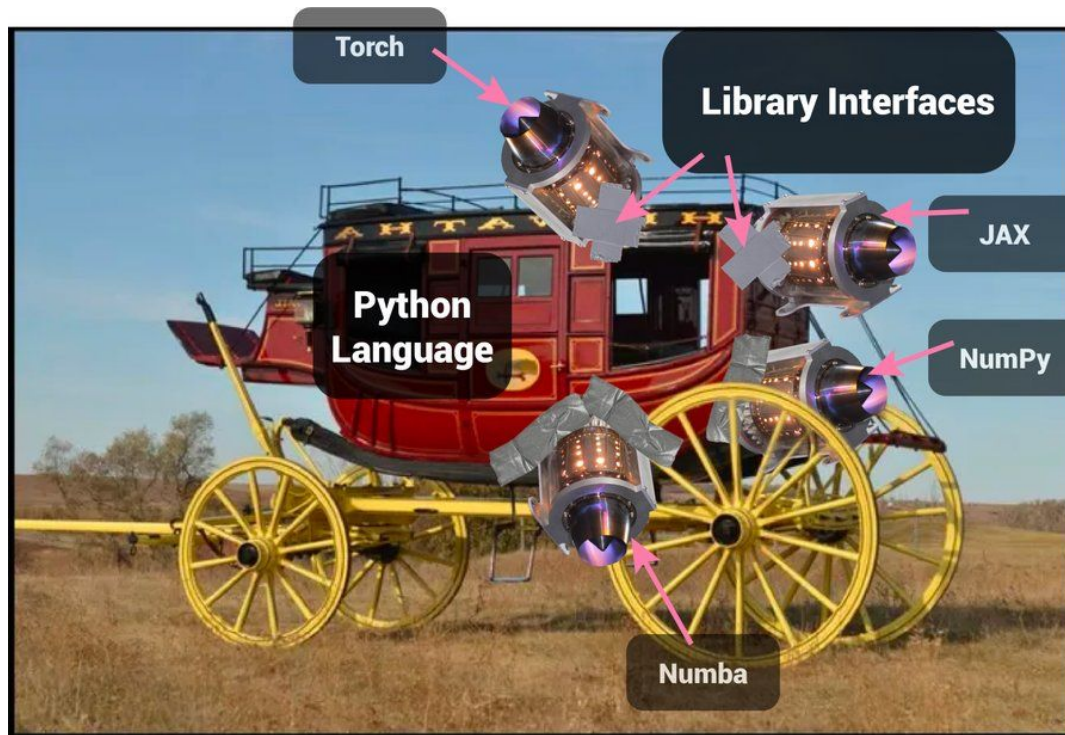
NumExpr: C-based JIT booster for numpy large arrays



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

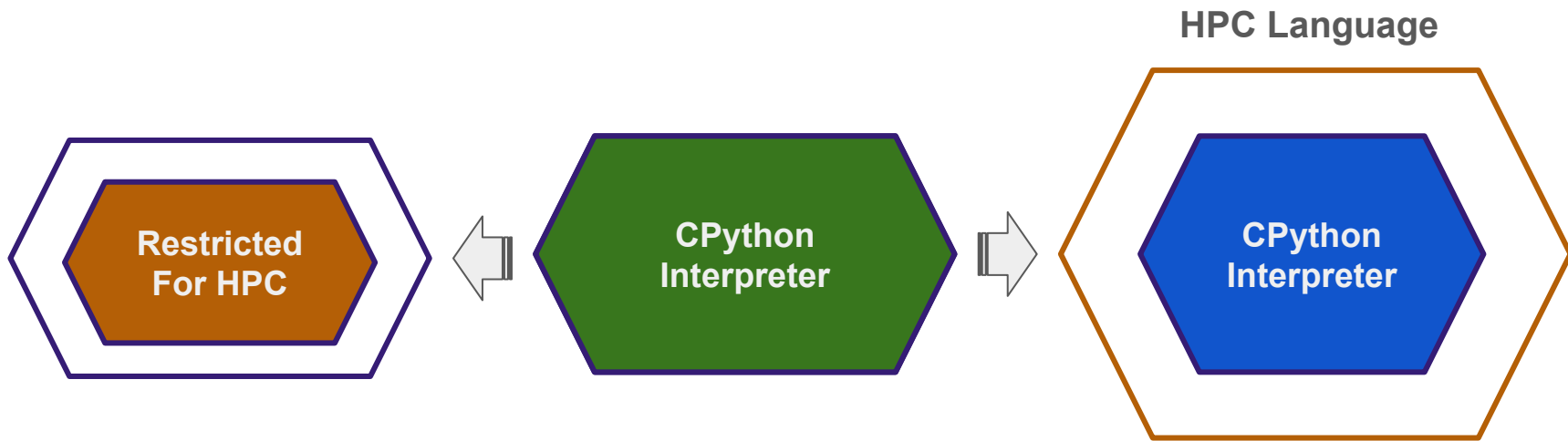
This is what we are having right now



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
- Numba
- JAX

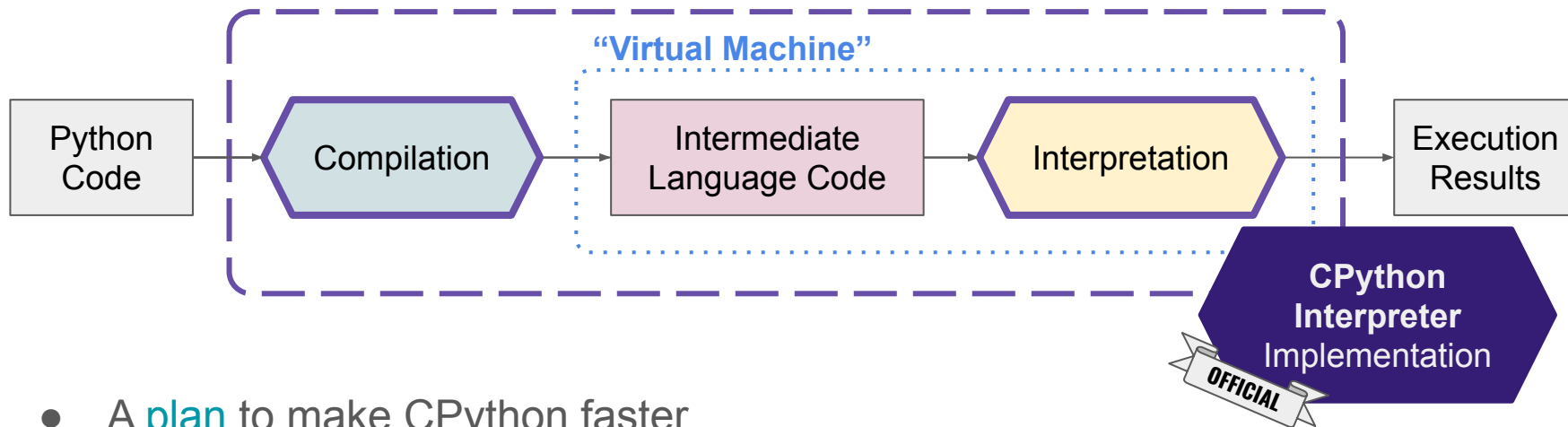
New CPython

- ❖ Fully optimization
- ❖ Rewritten, maybe?

Supersets to Python

- Cython
- Mojo

The “Shannon Plan”



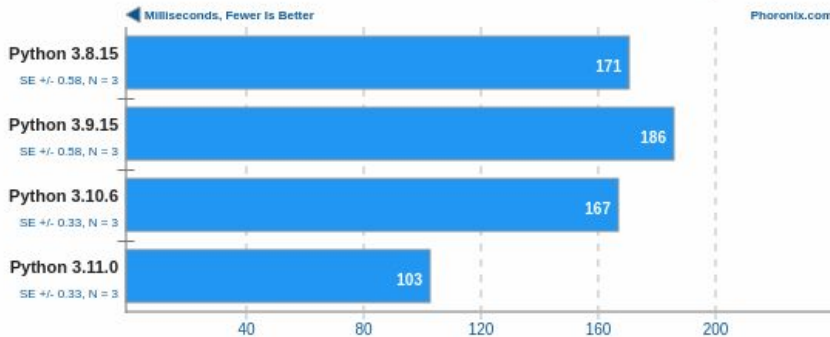
- A [plan](#) to make CPython faster
 - Originally proposed by Eric Snow, and Mark Shannon in 2020
 - Guido van Rossum joined and gave a talk in Python Language Summit (May 2021)
 - Based on the experience with “HotPy” and “HoyPy 2”
 - Promising 5x in 4 years, 1.5x per year

PyPerformance 1.0.0

Benchmark: go



Phoronix.com

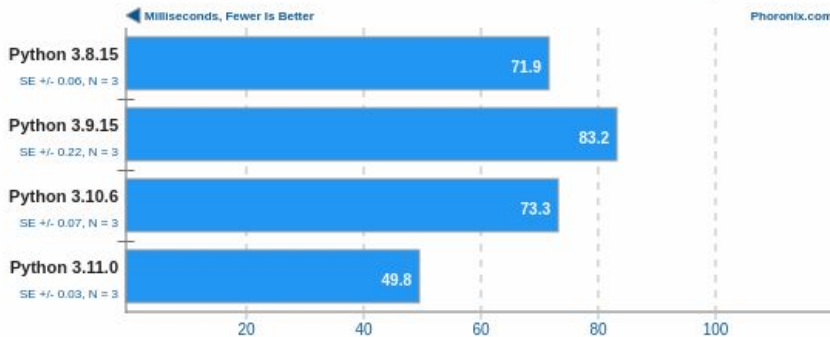


PyPerformance 1.0.0

Benchmark: float



Phoronix.com



Python 3.14 (α) will be using a new interpreter in CPython

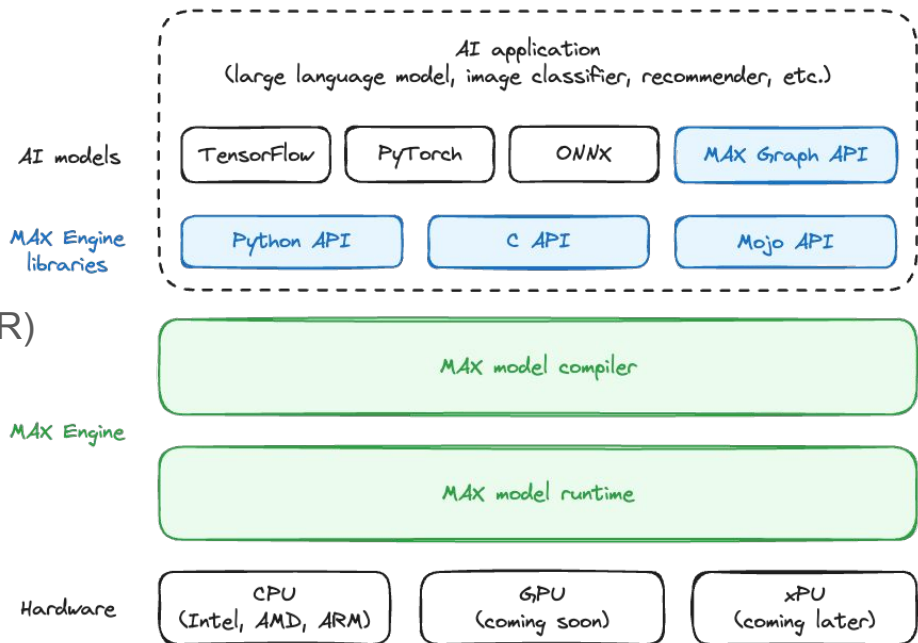
- Tail Call Optimization (TCO)
- **3~30% faster** python code
- **40%** speed-up in heavy-python benchmarks
- **Outperform JIT** compilers
- Max performed with Profile Guided Optimization
- Python 3.14.0 in Oct, 2025

Mojo == Python++ (?)

- A *new* language
 - Using Python as the syntax
 - High performance
 - 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.

SURVEY