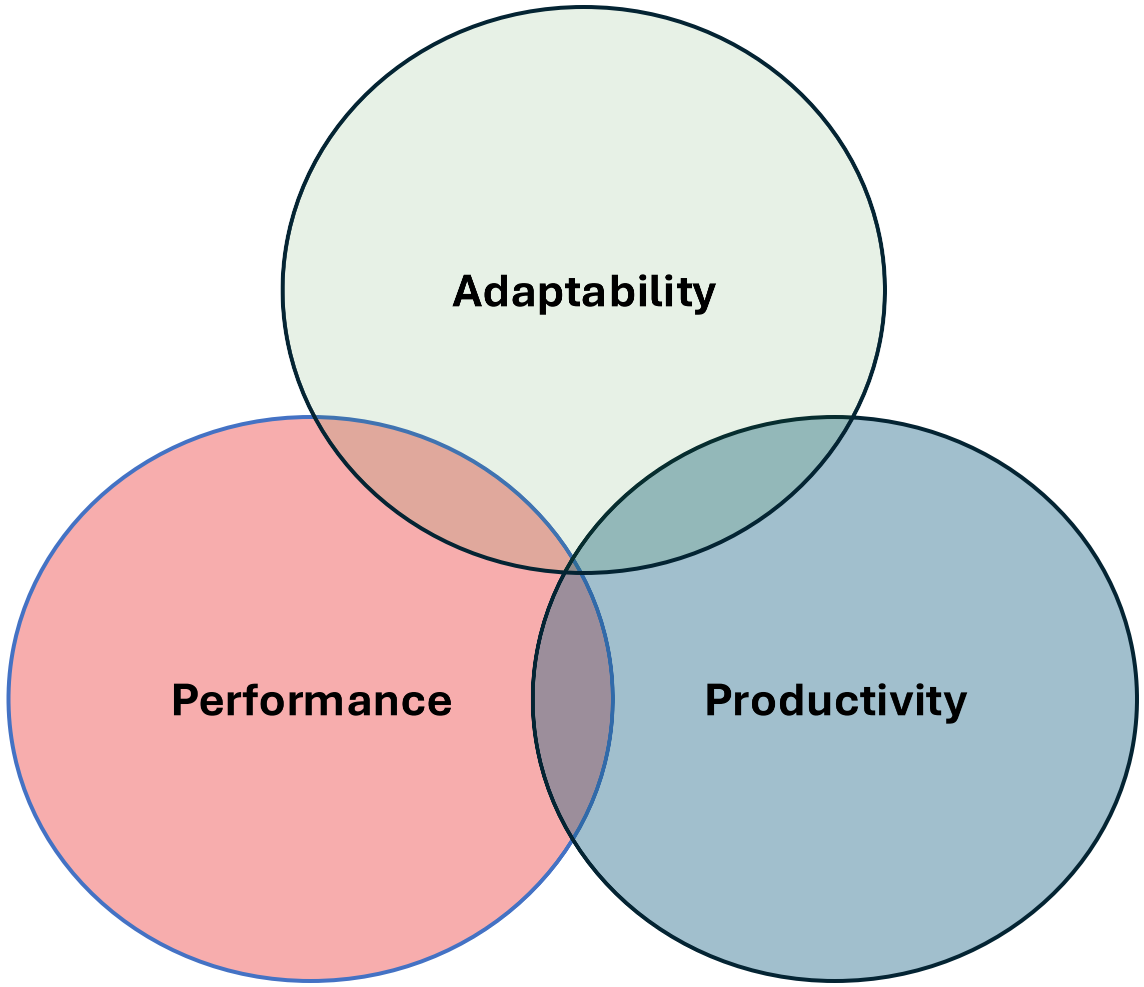
The HPC Balancing Act

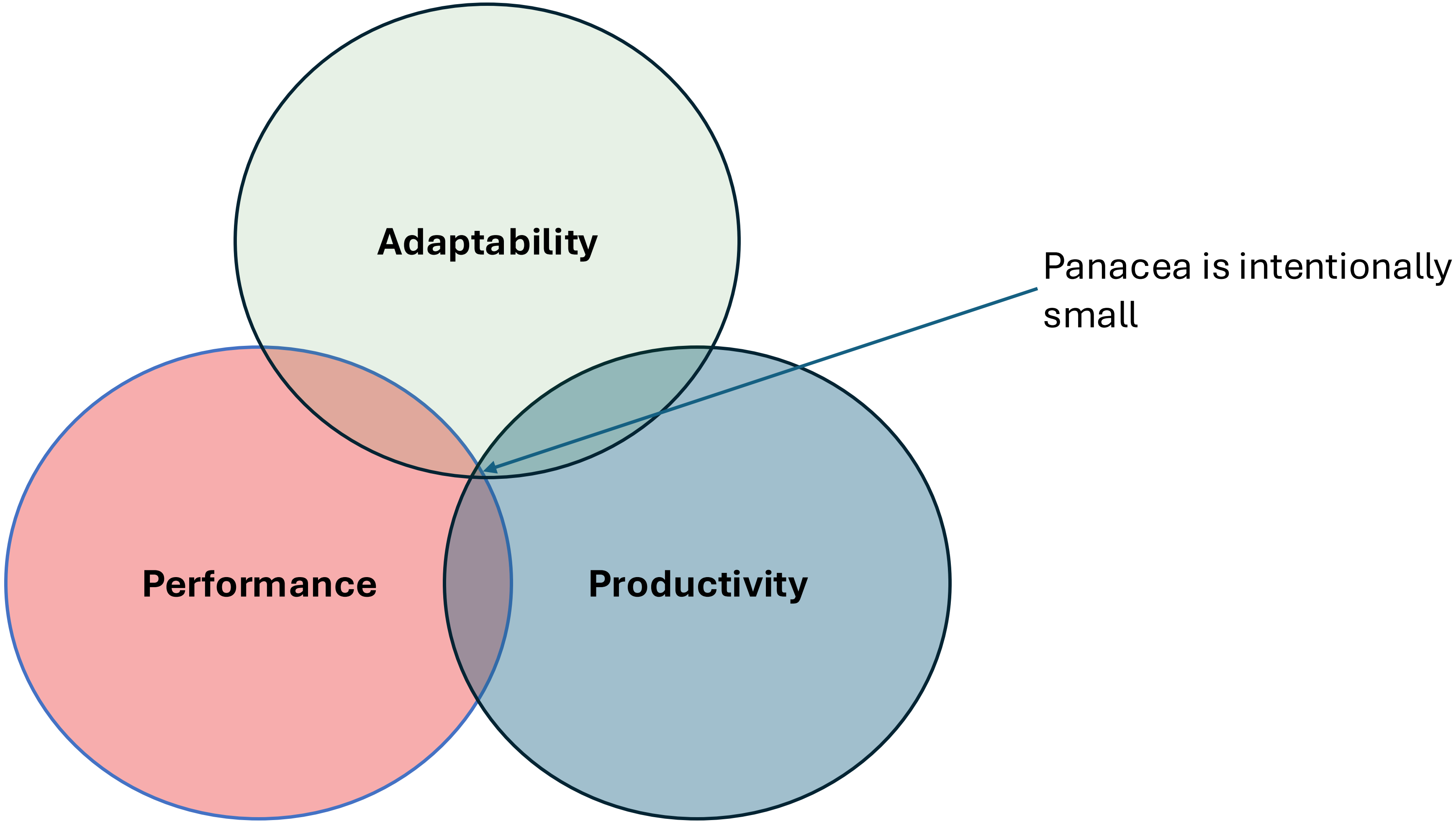
About me

- I am a PhD student at Virginia Tech
  - My primary field is HPC
  - I have no background in Physics, much less QP, HEP, NP
- My work is primarily developing a performant theory module
  - fitpack_cpp – Low latency, differentiable theory* module, currently in production running results for the Argonne Paper
  - GPU version of theory
- I have also played around with
  - A PDF level experimental module
  - Using generative models for parameterization of distributions

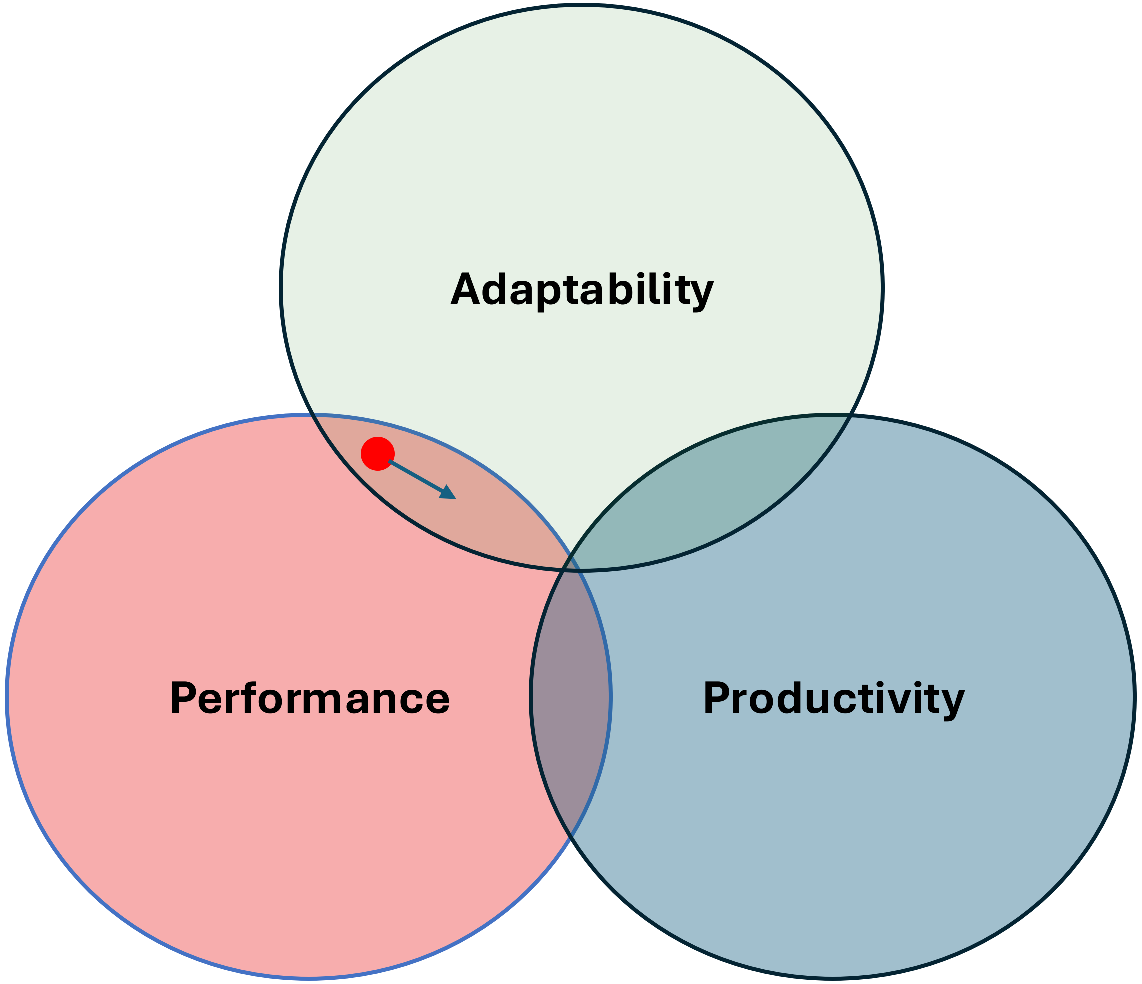
The HPC Balancing Act



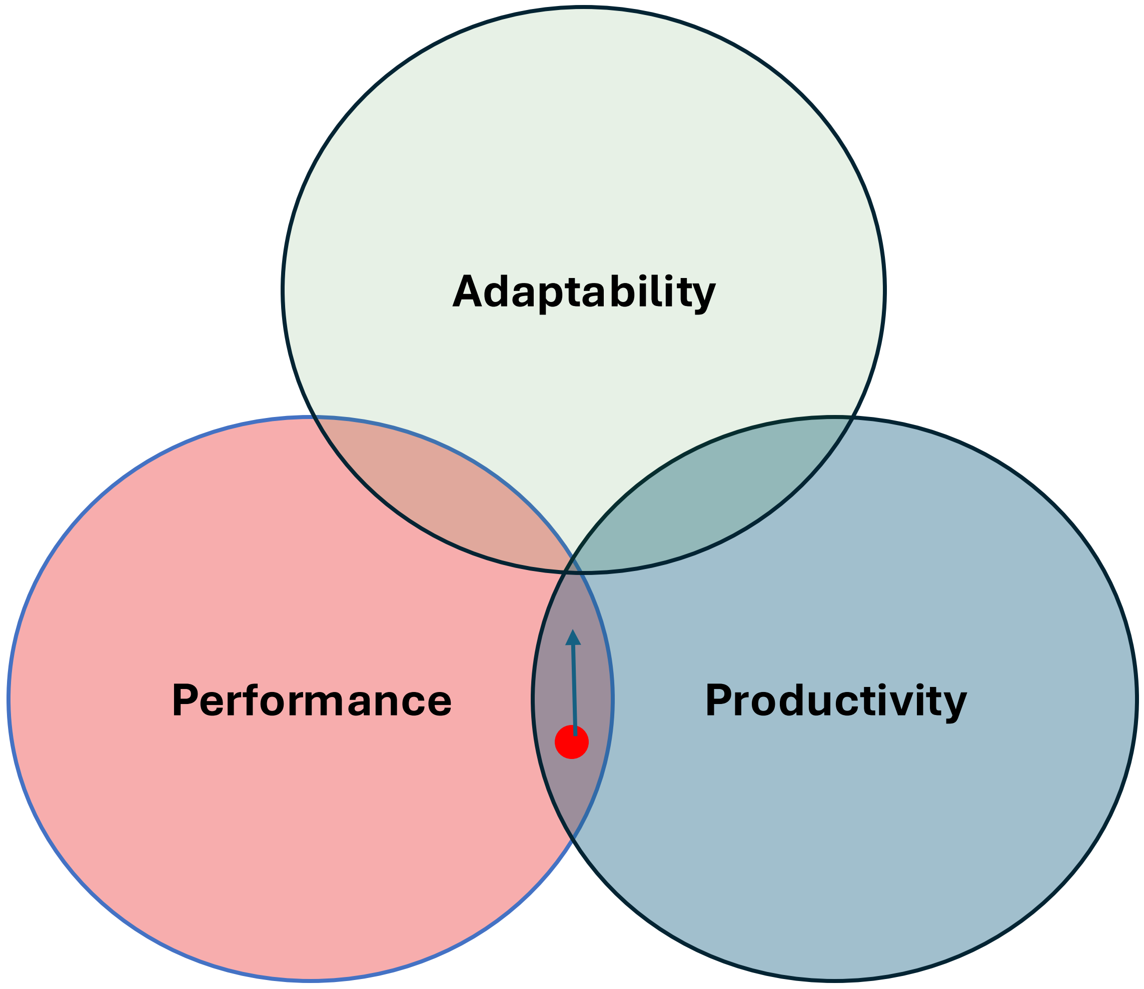
The HPC Balancing Act



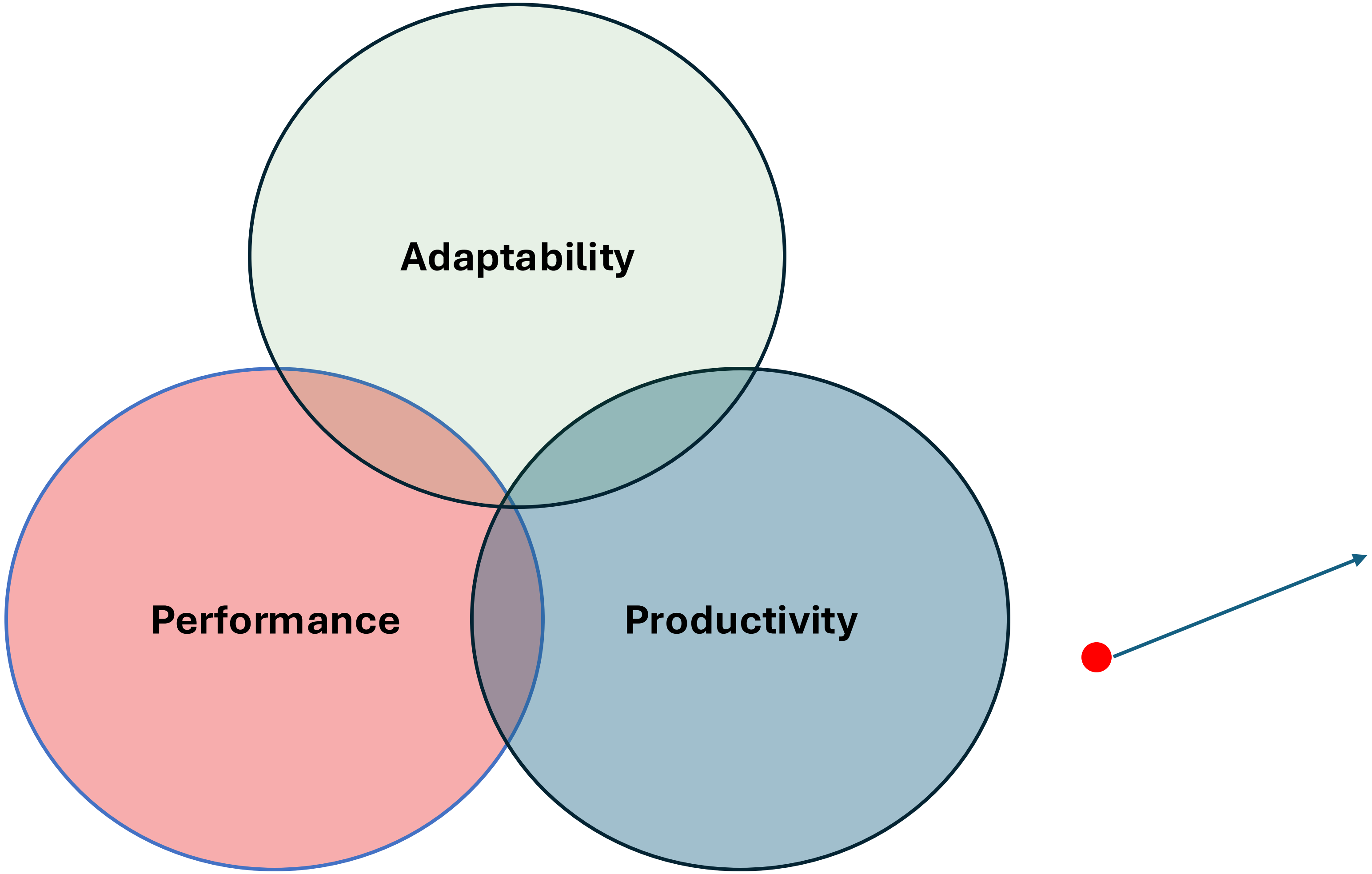
The HPC Balancing Act



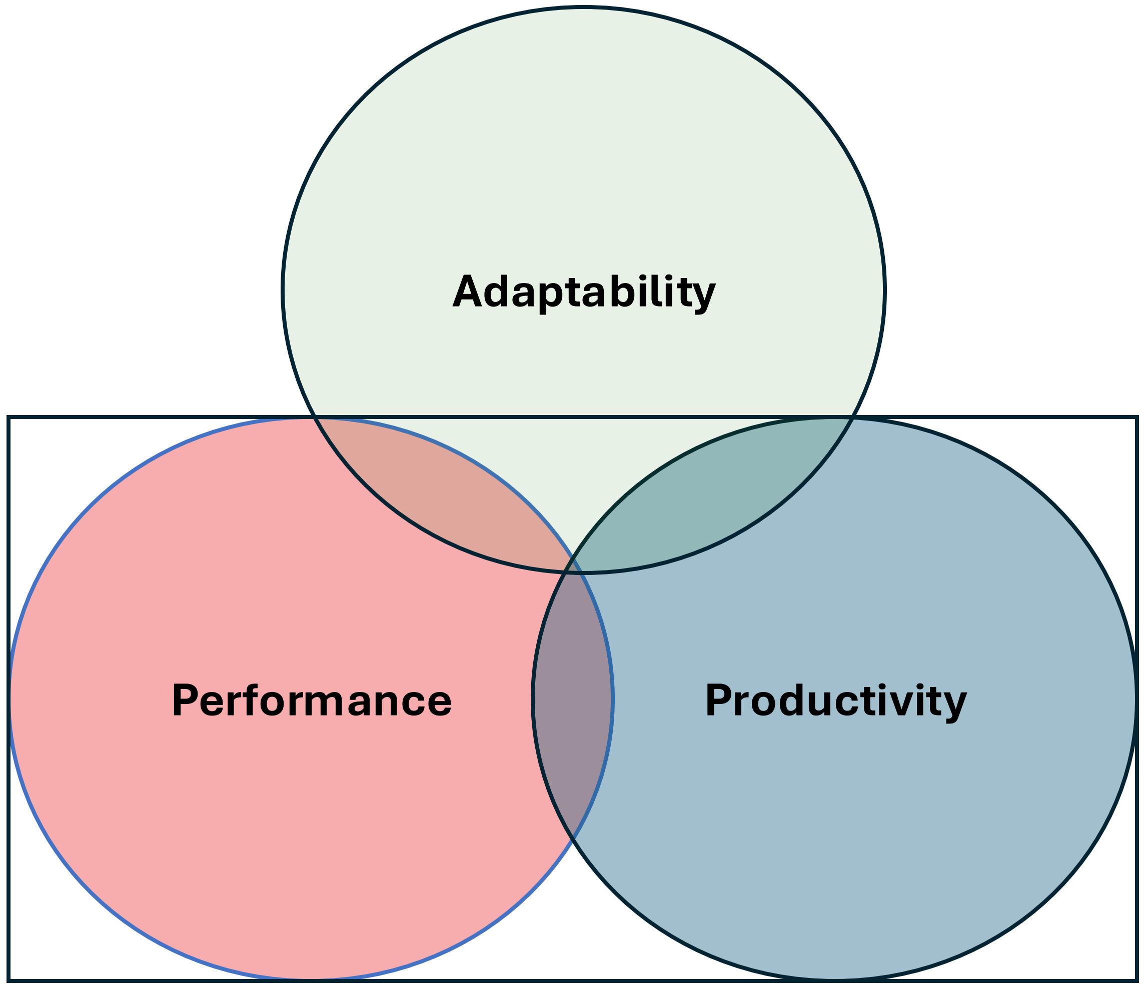
The HPC Balancing Act



The HPC Balancing Act



The HPC Balancing Act



The Battle of the Languages

Pytorch isn't Magic

I see this get thrown around a lot

"We are using Pytorch because we want [Something] to be differentiable"

Pytorch isn't Magic

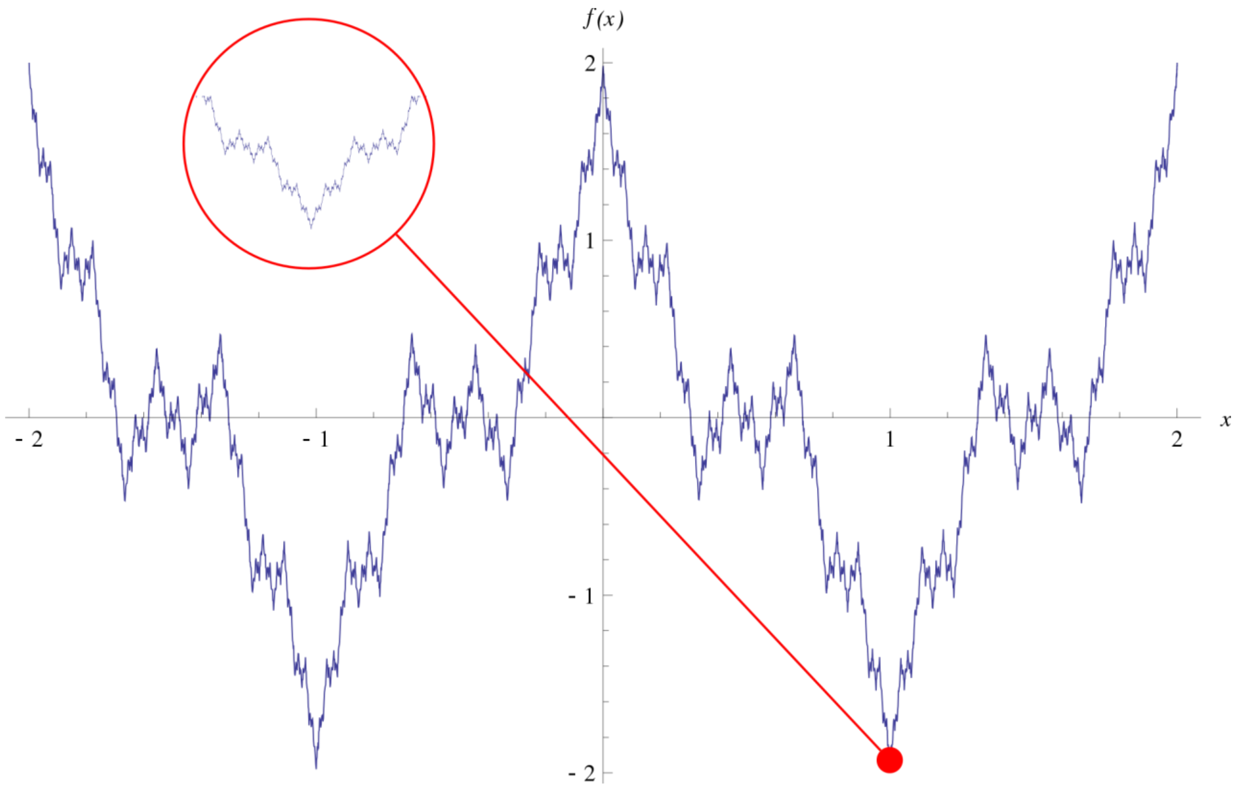
$$\text{Pytorch}\left(\sum_{n=1}^{\infty} \frac{\sin(n^2 x)}{n^2} \right)$$

Pytorch isn't Magic

$$\text{Pytorch}\left(\sum_{n=1}^{\infty} \frac{\sin(n^2 x)}{n^2} \right)$$

Pytorch isn't Magic

$$\text{Pytorch}\left(\sum_{n=1}^{\infty} \frac{\sin(n^2 x)}{n^2} \right)$$



Pytorch isn't Magic

- A derivative of a function at a point exists or doesn't

Pytorch isn't Magic

- A derivative of a function at a point exists or doesn't
- A library can't change that
  - A library can make it **simpler** to find out the derivative

Pytorch isn't Magic

- A derivative of a function at a point exists or doesn't
- A library can't change that
  - A library can make it **simpler** to find out the derivative
- We can also reformulate our problem so that we can extract derivatives for values of interest e.g. DistroSA, Normalizing Flows

Pytorch isn't Magic

- A derivative of a function at a point exists or doesn't
- A library can't change that
  - A library can make it **simpler** to find out the derivative
- We can also reformulate our problem so that we can extract derivatives for values of interest e.g. DistroSA, Normalizing Flows
  - Pytorch/Tensorflow cannot automatically do this for us (to my knowledge)

The Battle of the Languages (Contd)

- I keep seeing calls for us to choose a side
  - Our project has components that would benefit from ML/DL libraries, and other that wouldn't

The Battle of the Languages (Contd)

- I keep seeing calls for us to choose a side
- But!!

The Battle of the Languages (Contd)

- I keep seeing calls for us to choose a side
- But!!

The Battle of the Languages (Contd)

- I keep seeing calls for us to choose a side
  - Our project has components that would benefit from ML/DL, and other that wouldn't
- But most of these libraries rely on high performance languages for their performance-critical components
  - Specifically Matrix multiplication!

How much faster is matrix multiplication?

	A100 40GB PCIe	A100 80GB PCIe	A100 40GB SXM	A100 80GB SXM
FP64	9.7 TFLOPS			
FP64 Tensor Core	19.5 TFLOPS			
FP32	19.5 TFLOPS			
Tensor Float 32 (TF32)	156 TFLOPS | 312 TFLOPS*			
BFLOAT16 Tensor Core	312 TFLOPS | 624 TFLOPS*			
FP16 Tensor Core	312 TFLOPS | 624 TFLOPS*			
INT8 Tensor Core	624 TOPS | 1248 TOPS*			

Just 2X Faster

How much faster is matrix multiplication?

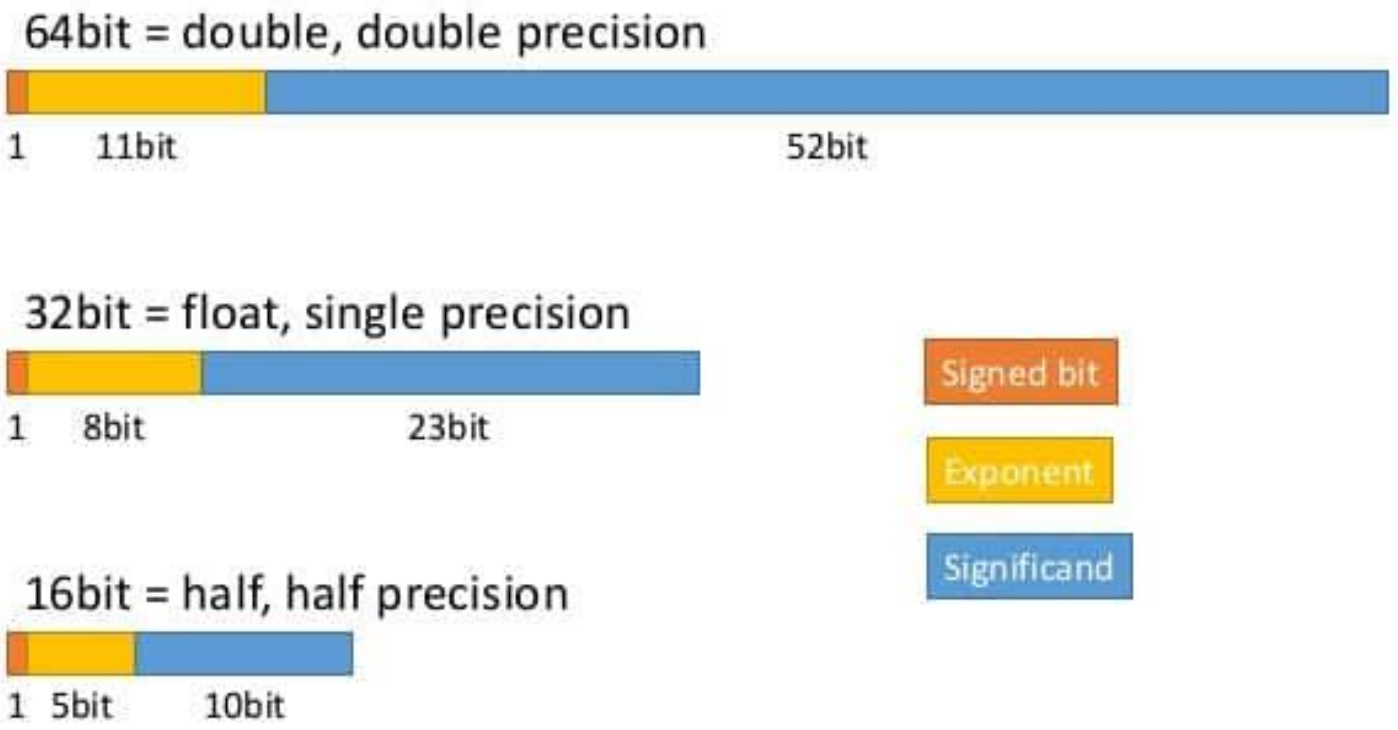
	A100 40GB PCIe	A100 80GB PCIe	A100 40GB SXM	A100 80GB SXM
FP64	9.7 TFLOPS			
FP64 Tensor Core	19.5 TFLOPS			
FP32	19.5 TFLOPS			
Tensor Float 32 (TF32)	156 TFLOPS | 312 TFLOPS*			
BFLOAT16 Tensor Core	312 TFLOPS | 624 TFLOPS*			
FP16 Tensor Core	312 TFLOPS | 624 TFLOPS*			
INT8 Tensor Core	624 TOPS | 1248 TOPS*			

Just 2X Faster

AMD 7702 is 2 TFLOPS

Traditional Data types

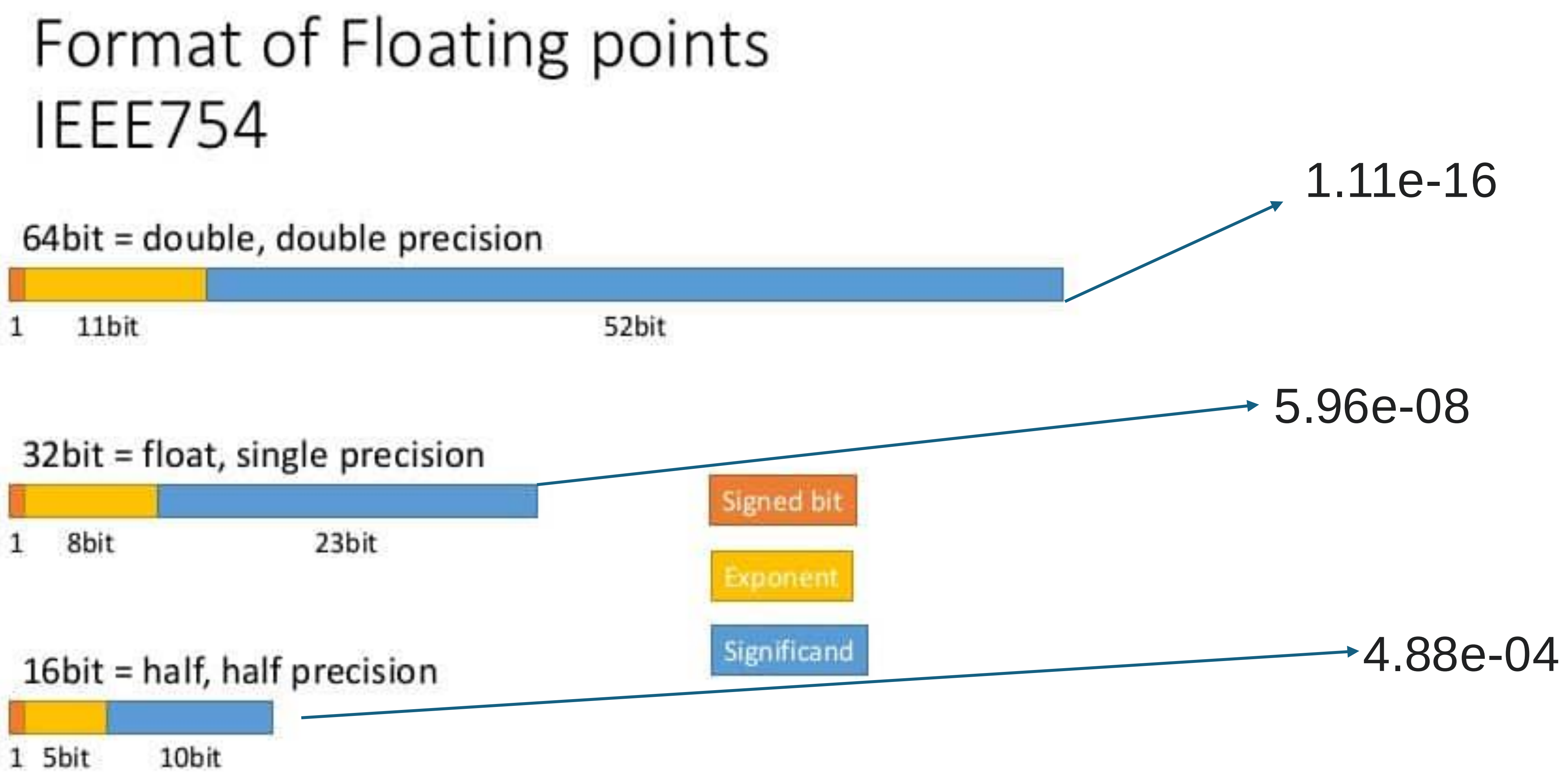
Format of Floating points
IEEE754



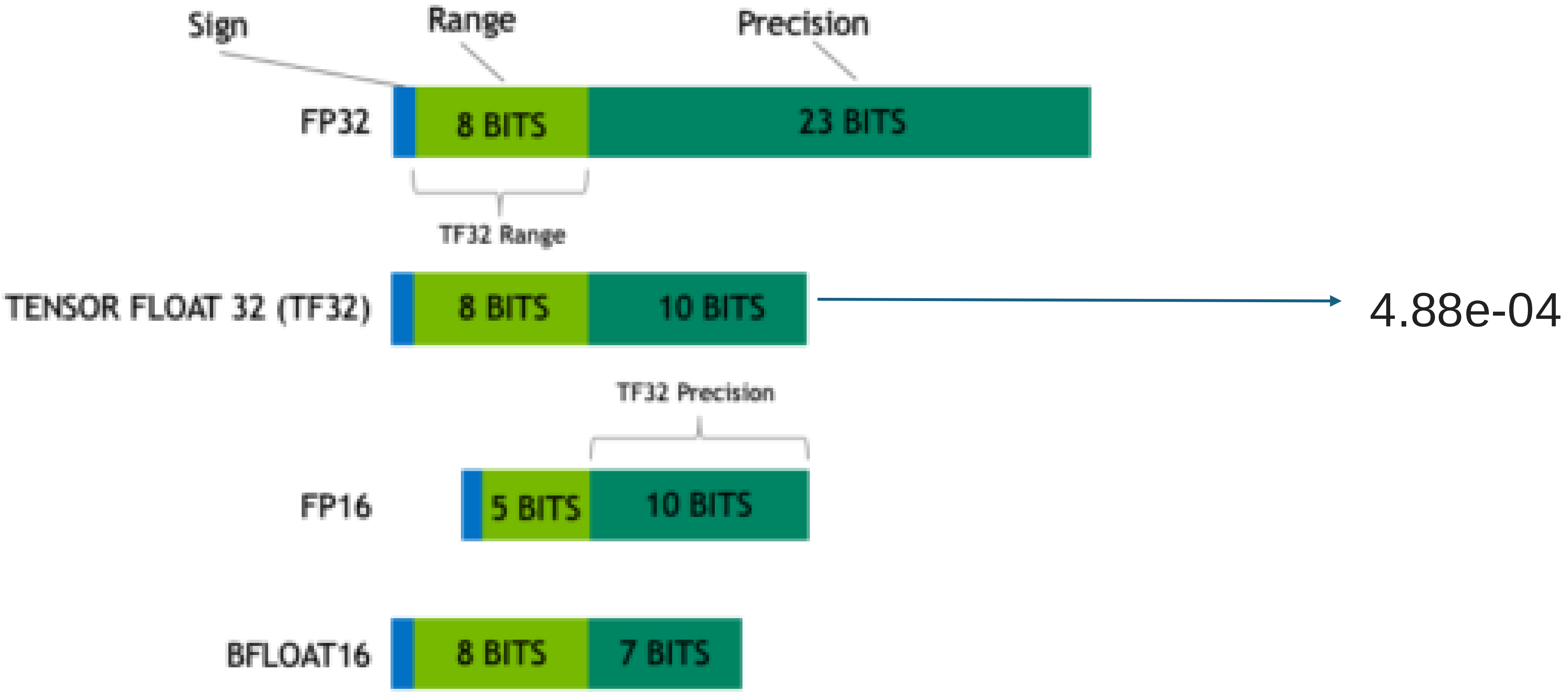
Machine Epsilon

It is the smallest positive floating-point number that, when added to 1.0, yields a result different from 1.0.

Traditional Data types



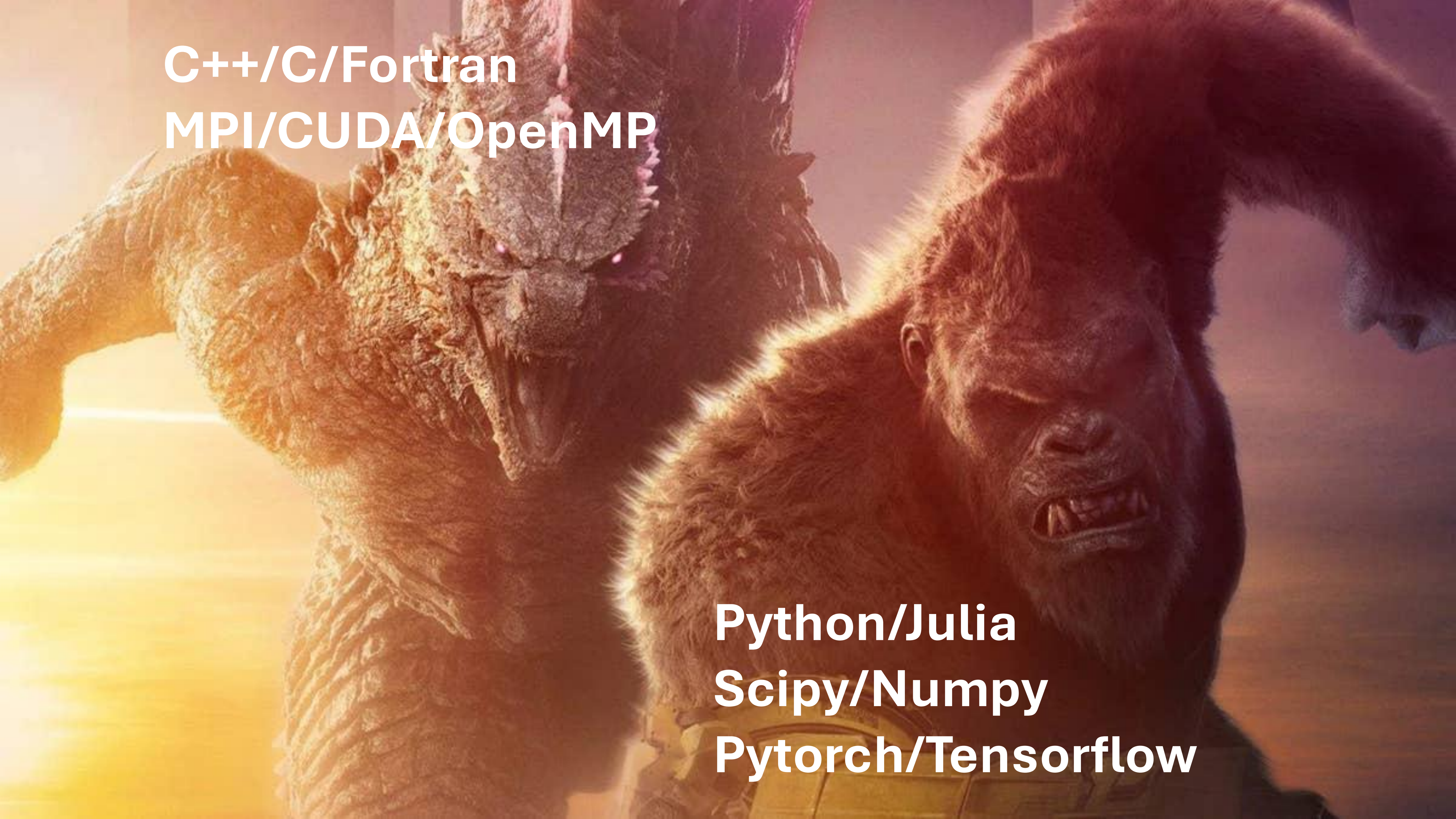
Tensor Data Types



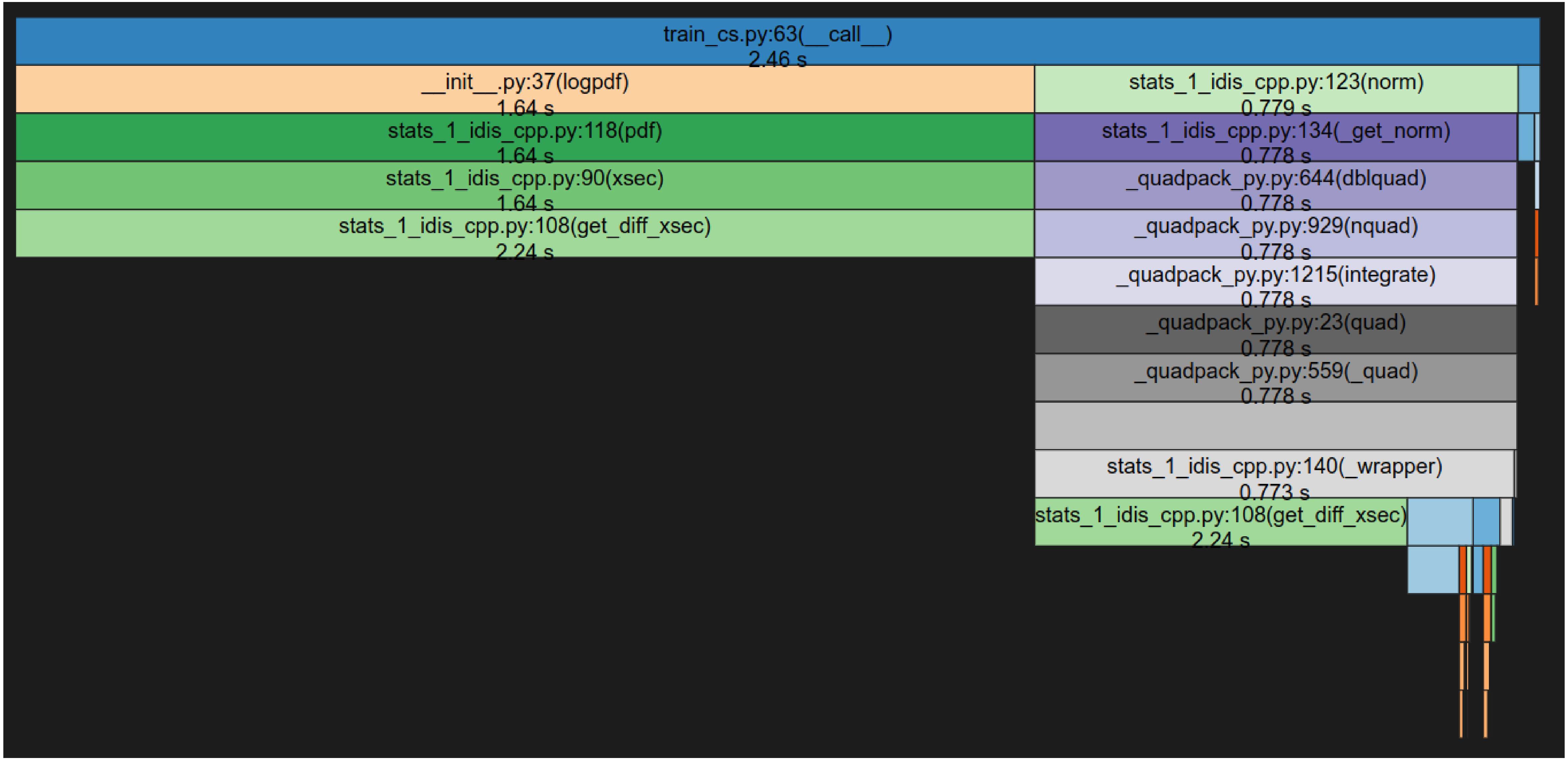
The Battle of the Languages (Contd)

- I keep seeing calls for us to choose a side
  - Our project has components that would benefit from ML/DL, and other that wouldn't
- But most of these libraries rely on high performance languages for their performance-critical components
  - Specifically Matrix multiplication!

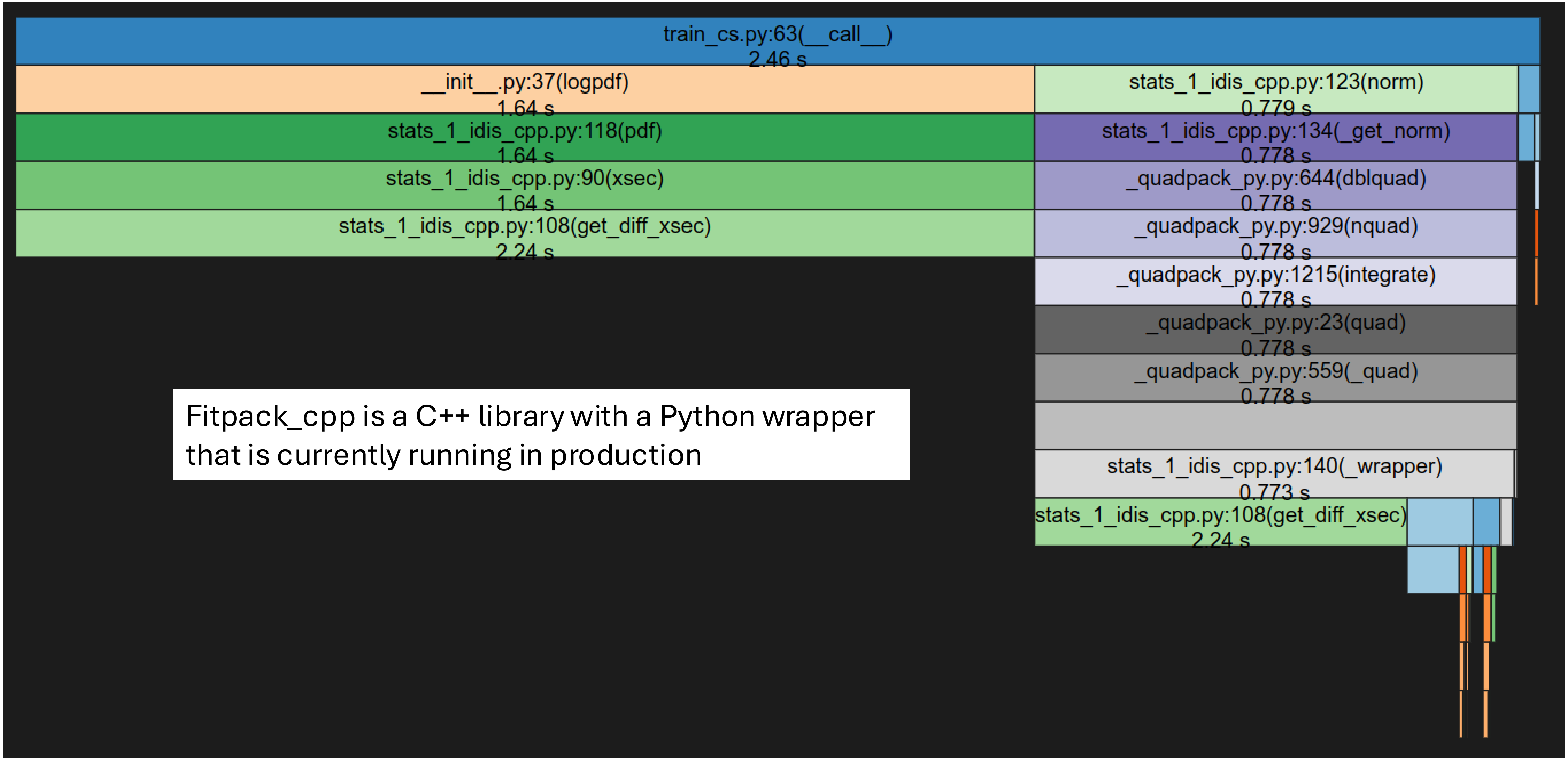
So why don't we ?


C++/C/Fortran
MPI/CUDA/OpenMP
Python/Julia
Scipy/Numpy
Pytorch/Tensorflow

Interop

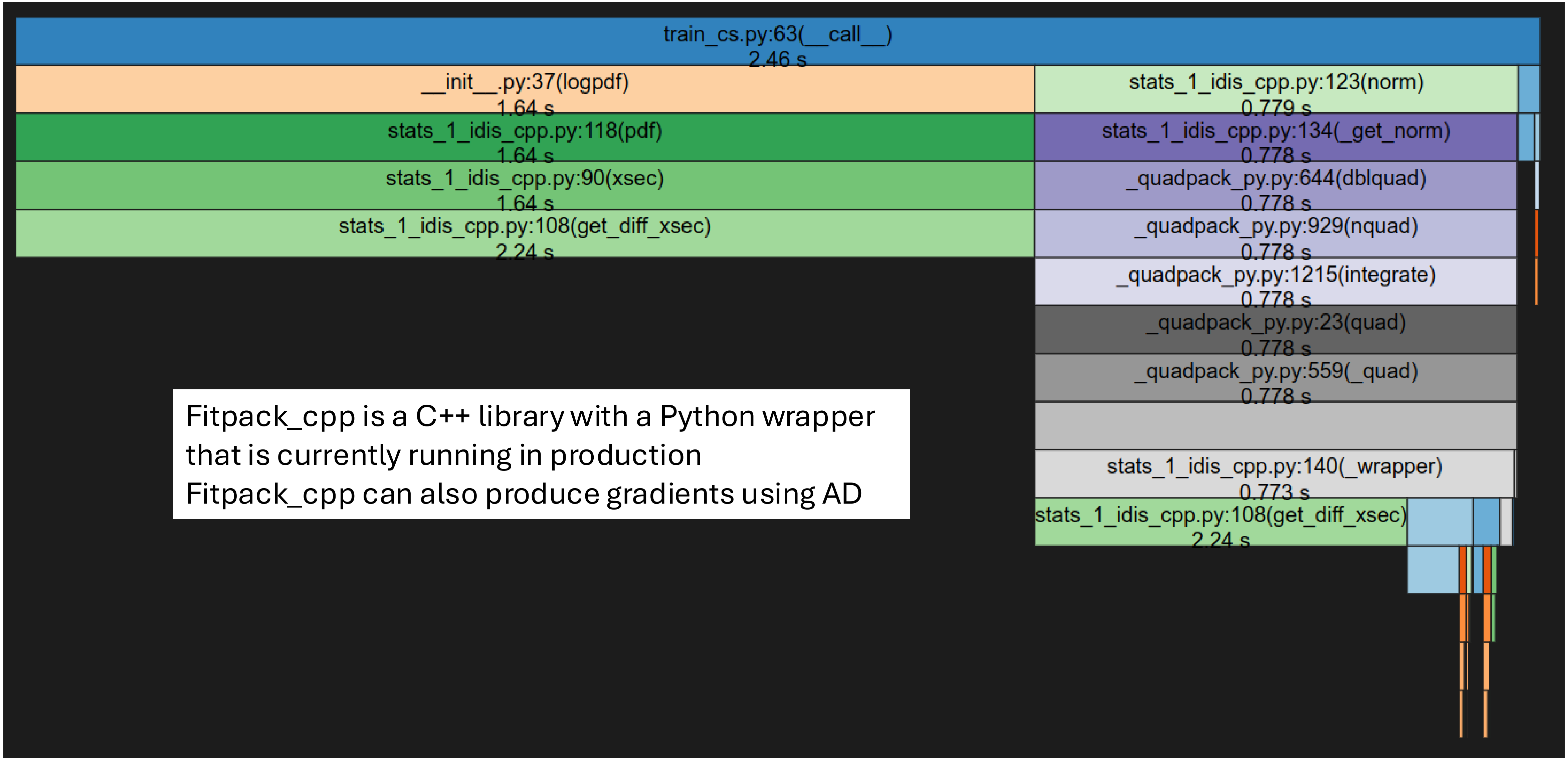

train_cs.py:63(__call__)
2.46 s
__init__.py:37(logpdf)
1.64 s
stats_1_idis_cpp.py:123(norm)
0.779 s
stats_1_idis_cpp.py:118(pdf)
1.64 s
stats_1_idis_cpp.py:134(_get_norm)
0.778 s
stats_1_idis_cpp.py:90(xsec)
1.64 s
_quadpack_py.py:644(dblquad)
0.778 s
stats_1_idis_cpp.py:108(get_diff_xsec)
2.24 s
_quadpack_py.py:929(nquad)
0.778 s
_quadpack_py.py:1215(integrate)
0.778 s
_quadpack_py.py:23(quad)
0.778 s
_quadpack_py.py:559(_quad)
0.778 s
stats_1_idis_cpp.py:140(_wrapper)
0.773 s
stats_1_idis_cpp.py:108(get_diff_xsec)
2.24 s

Interop



Fitpack_cpp is a C++ library with a Python wrapper that is currently running in production

Interop



Fitpack_cpp is a C++ library with a Python wrapper that is currently running in production
Fitpack_cpp can also produce gradients using AD

The Workflow shouldn't be a set of Black Boxes

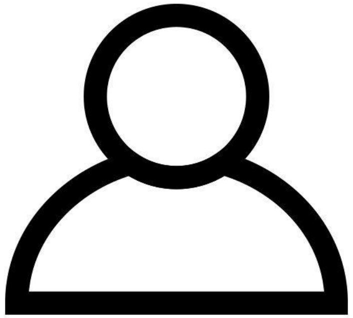
Scientist

Me

I developed this component in theory.
We need it to be fast

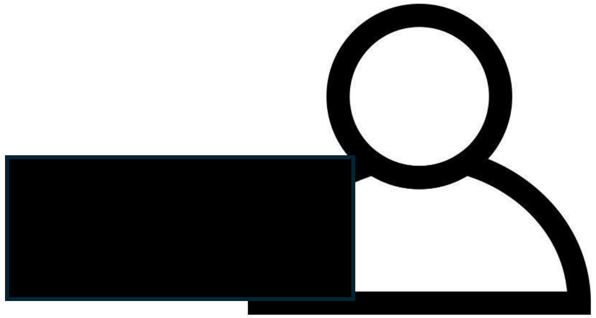
The Workflow shouldn't be a set of Black Boxes

Scientist



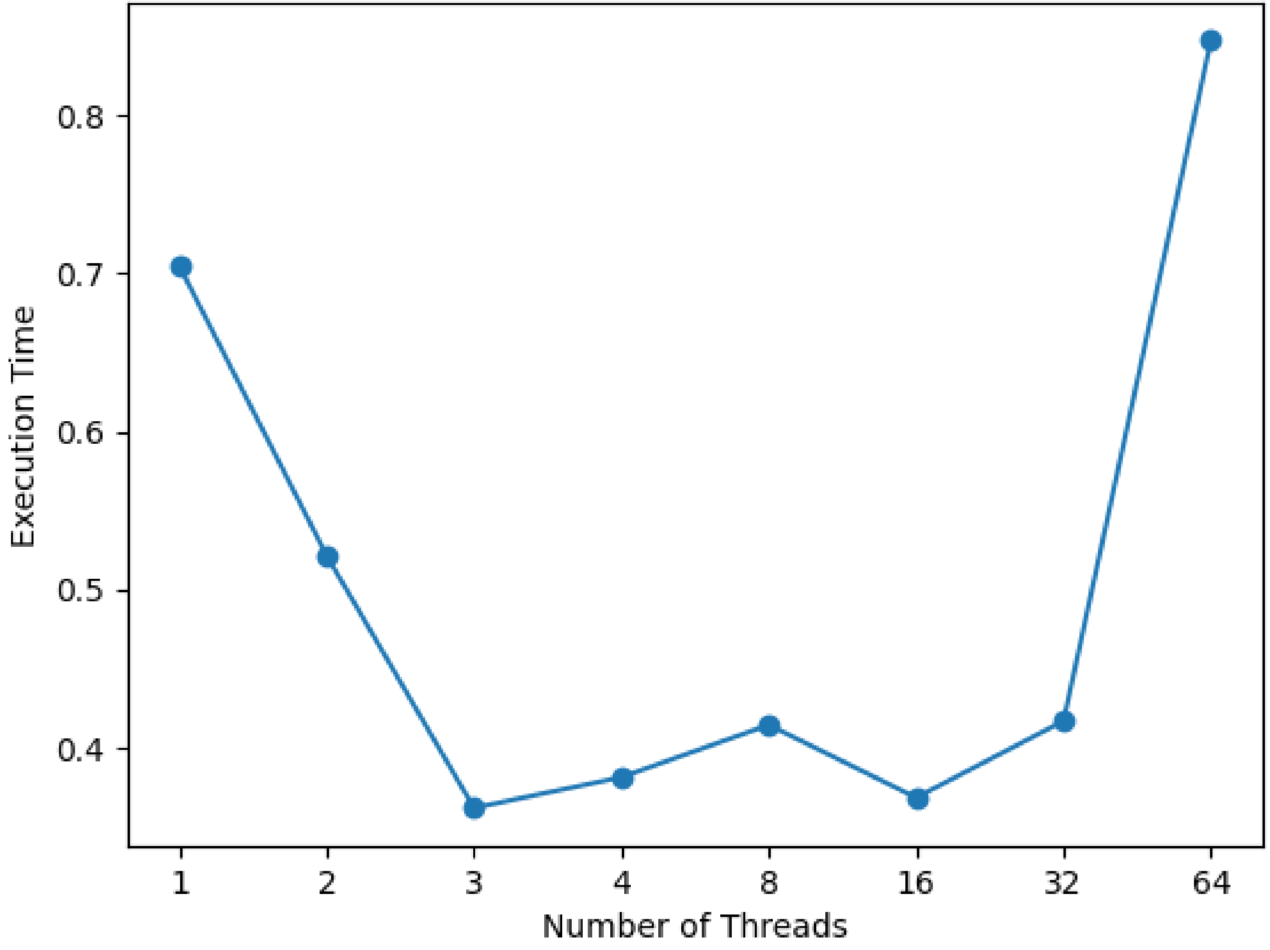
I developed this component in theory.
We need it to be fast

Me



But I can't parallelize this!

The Workflow shouldn't be a set of Black Boxes



Performance of fitpack_cpp plateaus with number of threads

- Design choices made due to insufficient information
- Large amount of work required to fix this

Theory shouldn't be a Black-Box

- Theory has a lot of moving pieces
- Algorithmic design choices should be made in collaboration

Theory shouldn't be a Black-Box

- Theory has a lot of moving pieces
- Algorithmic design choices should be made in collaboration
  - We want to avoid performance plateaus

Theory shouldn't be a Black-Box

- Theory has a lot of moving pieces
- Algorithmic design choices should be made in collaboration
    - We want to avoid performance plateaus
    - For example, we should be thinking of not evaluation the cross-section at one point, but 1 million.

Conclusion

- We need to find a middle ground, where we can all **collaborate**.
    - When designing algorithms, working together can help make choices that will reduce the work needed in the future to overcome issues.
    - Choice of language can be done based on quantitative metrics like % of total time executed, simplicity
- We need to collaborate so that we design scalable algorithms for this problem.
    - HPC should not be an afterthought