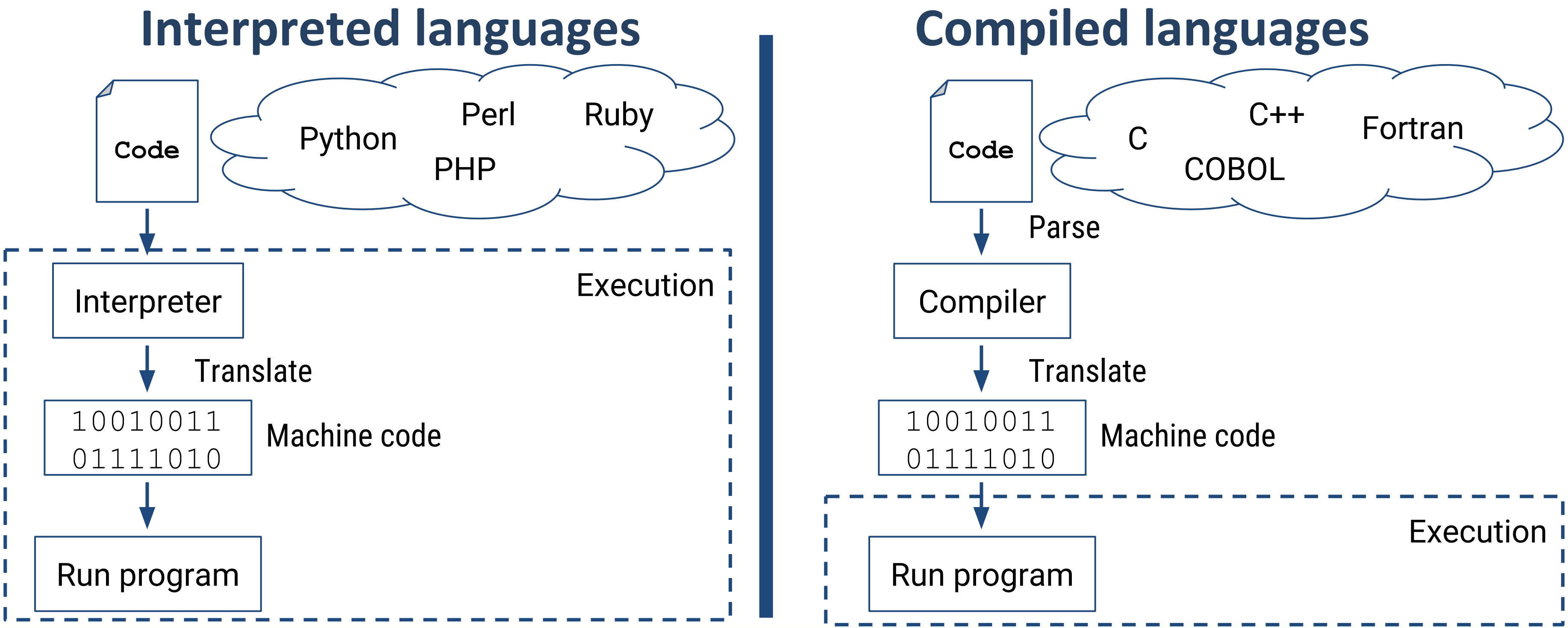
XSEDE
Extreme Science and Engineering
Discovery Environment

Programming for performance in Python

Presented by:

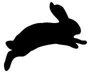
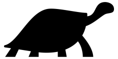
- Steve Lantz - steve.lantz@cornell.edu
- Roberto Camacho - rcamachobarranco@utep.edu

Programming for performance in Python

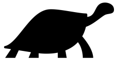
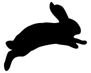


Comparing interpreted and compiled languages

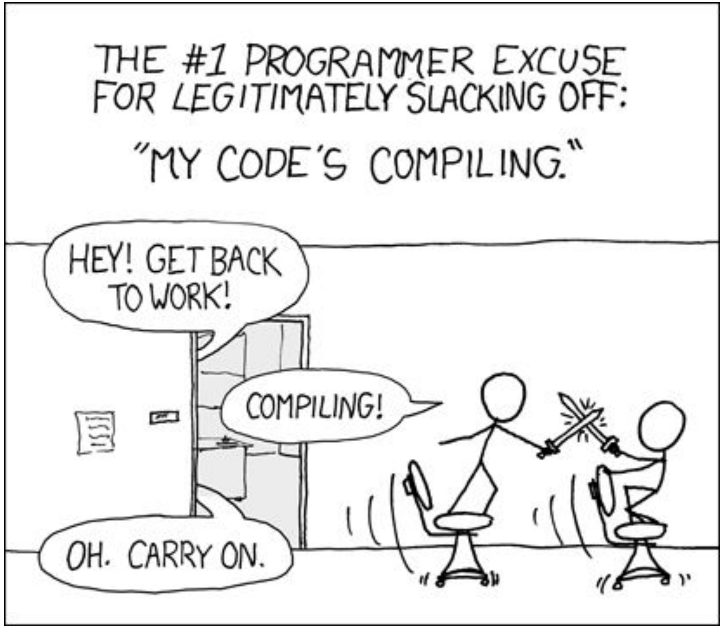
Interpreted code

- Very flexible (adaptive code)
- Development: 
- Application performance: 
- Great for prototyping
- High overhead during execution
- Portable
- Interpreter required for running

Compiled code

- Static
- Development: 
- Application performance: 
- Great for production code
- High overhead during dev stage
- Device-specific
- Compiler required for dev stage

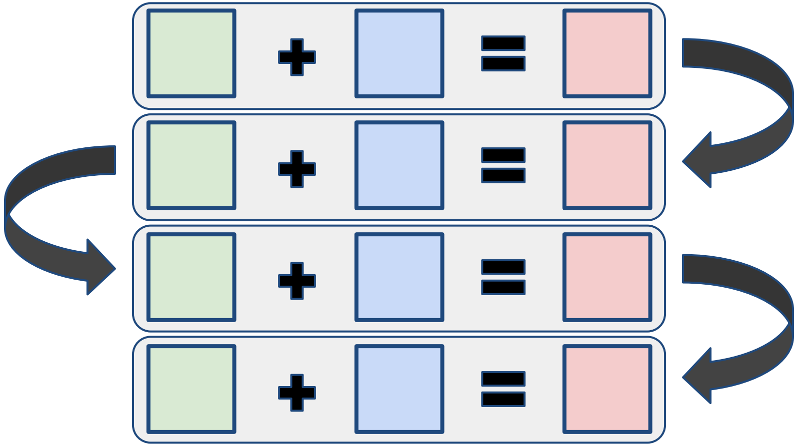
It all depends on how you define performance...



XSEDE

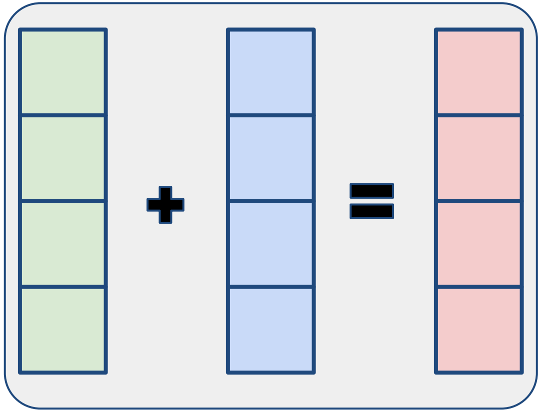
A peek at how microprocessors work

Sequential processing (SISD)



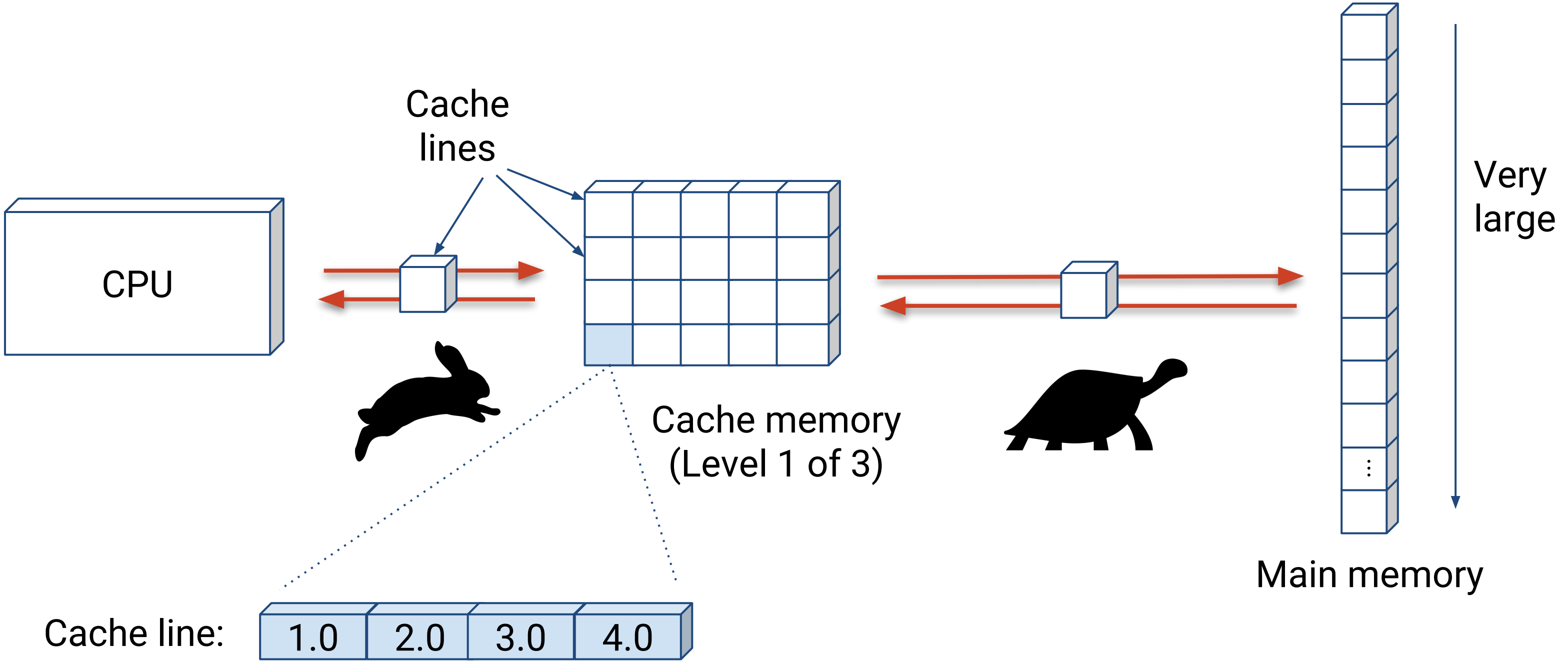
SISD: 4 instructions, 4 outputs

Vector processing (SIMD)



SIMD: 1 instruction, 4 outputs

CPU, cache, and main memory



XSEDE

Improving performance of Python code

- **Libraries with compiled code wrappers**
  - NumPy, Tensorflow, Keras
  - Transparent use of vector processing (SIMD)
  - Efficient use of cache
  - Underlying code is C or Fortran
- **Cython**
  - Converts Python code to C and compiles it
- **Custom distributions**
  - PyCUDA (for GPUs), Intel Python, PyPy

NumPy: looking under the hood

- NumPy arrays:
  - Metadata (length, data type, dimensions)
  - Stored in contiguous RAM memory
- Why is this important?
  - Wrappers around optimized C routines
  - Spatial locality ➡ efficient loading
  - Vectorized instructions

XSEDE

NumPy: looking under the hood

- **NumPy** methods are based on the **BLAS** and **LAPACK** routines
  - Low-level specifications of linear algebra operations
  - Choose the best implementation!
    - Default
    - ATLAS
    - Intel MKL
    - OpenBLAS