Python for Big Data

•••

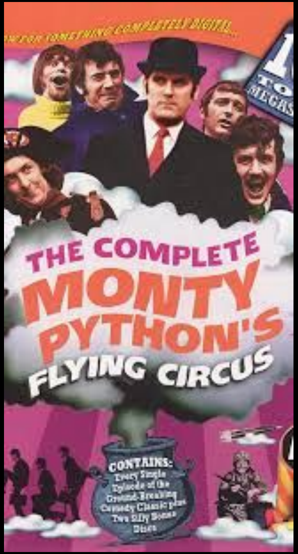
Bala Desinghu
Senior Scientist
Office of Advanced Research Computing (OARC)
Rutgers University

Python

Developed by Guido van Rossum in 1990s

Named after BBC TV show Monty Python

Popular scripting language



Why Python?

Readability

Simplicity

High level language

Concise

Enforces good practices

Rapid development

Covers broad spectrum of application areas

Python Limitations

Speed - Much slower than c, C++, fortran

Memory - not so good for memory intensive computations

Solutions

Speed = numpy, numba, Dask, PySpark, Jython, Cython, mpi4py,....

Memory = Dask, PySpark, Vaex,....

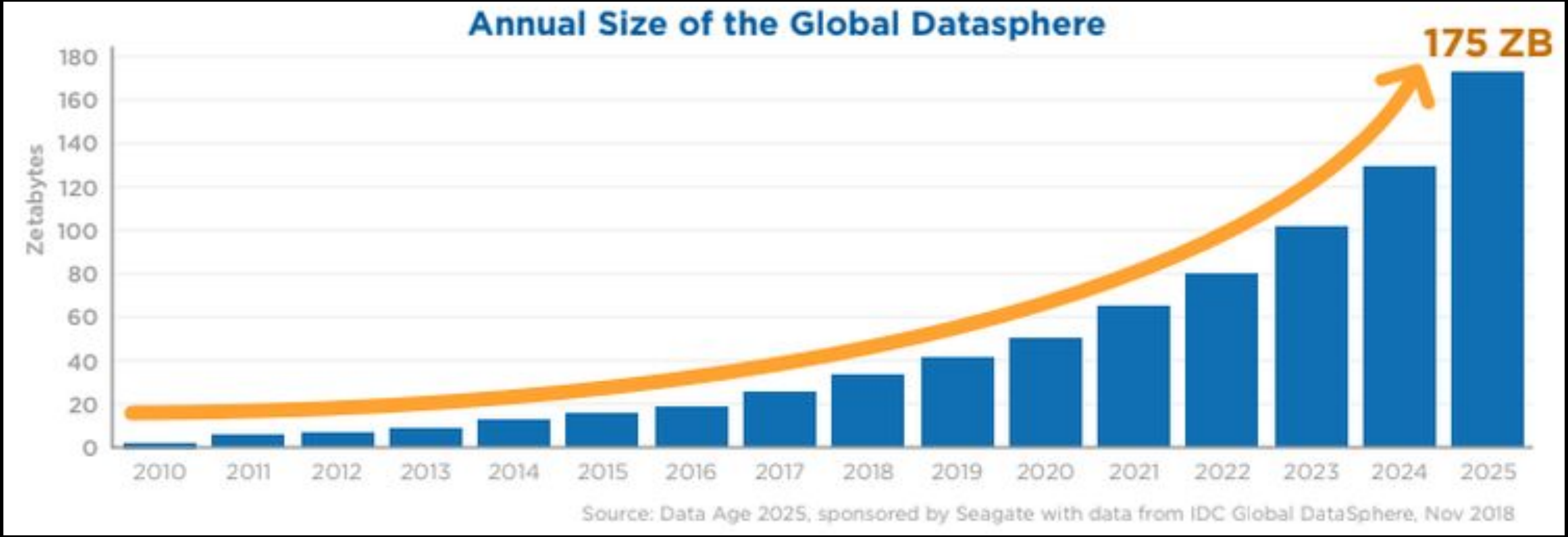
Big Data

Volume

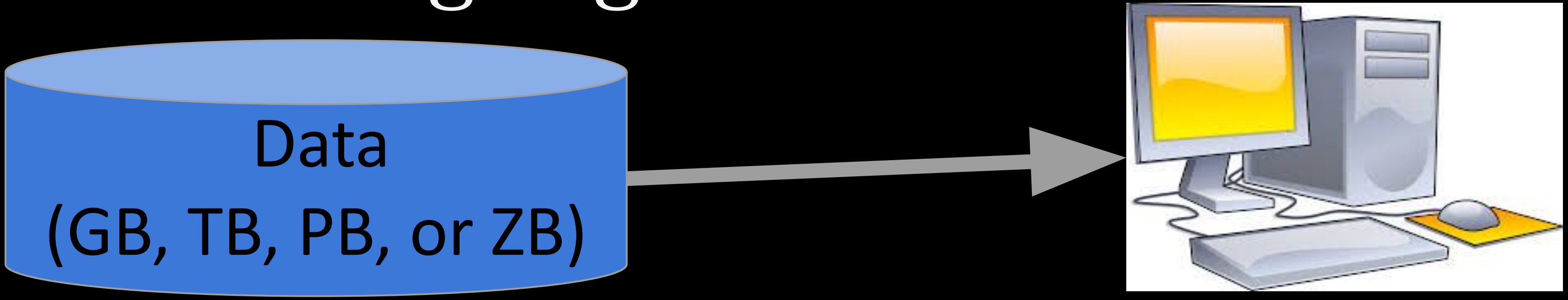
Velocity

Variety

Veracity


Annual Size of the Global Datasphere
175 ZB
Zetabytes
180
160
140
120
100
80
60
40
20
0
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
Source: Data Age 2025, sponsored by Seagate with data from IDC Global DataSphere, Nov 2018

Processing Big Data



- Data becomes big when the data won’t fit into the memory for processing.
    - Data > 8 GB - > big for my laptop
    - Data > 200 GB -> big for a typical compute node on a campus cluster
    - Data > 12 TB -> big for Bridges-2 at PSC
    - HPE built a machine with 160 TB in 2017
- For practical purposes, let us take 1 TB as the Big Data.

Processing big data:

Compute resources, Storage, Memory capacity, Data analysis tools, Viz tools, etc.

Python is suitable for rapid development and comes with a lot of analysis and viz tools. How to scale Python for CPU and Memory intensive computations?

Approaches addressing the CPU and Memory intensive computations with Python

- Data structures - Dense arrays in Numpy and Pandas dataframes
- Vector operations - Numpy, Pandas,
- Just-in-time Compilation - Numba
- Distributed task executions - Dask (API to Numpy, Scikit-Learn, Tensorflow, Keras)
    - Collections - arrays, bags, dataframes
    - Delayed Functions
    - Distributed task executions
    - Distributed Machine Learning

Demos