Implementation-level efficiency of python scientific software

Both Algorithm and Implementation level difference can leads to >1000x performance difference. So you have to pay attention to both.

Matrix multiplication of two 4096x4096 matrices

Pure Python

55432s (extrapolated)

Numpy Default BLAS/ATLAS (multi-core)

2.79s

Numpy Intel MKL (multi-core)

265 ms.

CUDA (CUBLAS) with PyTorch

21.7ms

Use fast linear algebra routine whenever possible

What if no existing implementation available?

(Ranked from easy to hard, lower to higher performance improvement potential if used correctly)

Numba JIT

Write Cython code which compiles to C

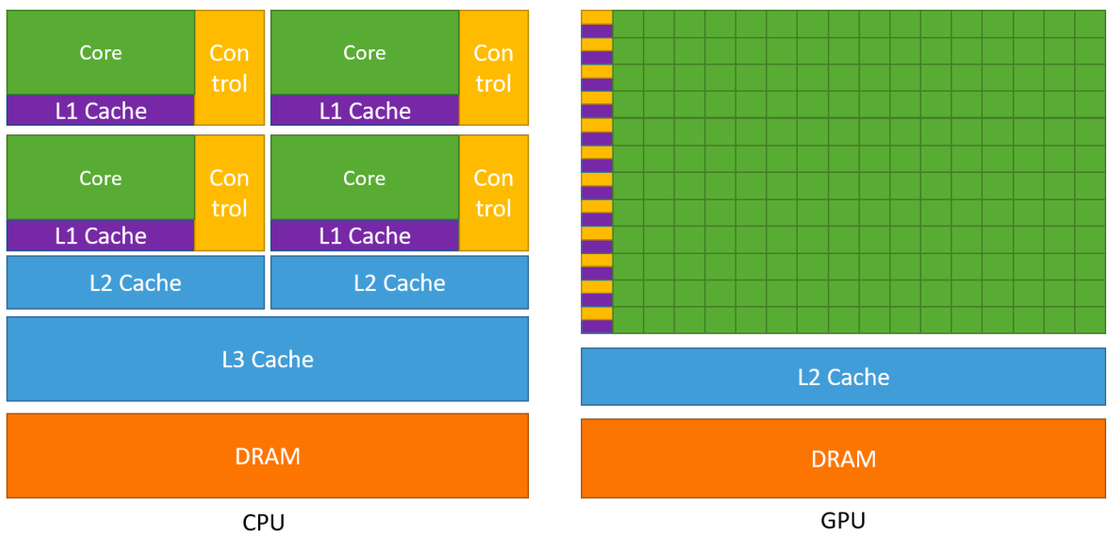
Implement with C/C++ and create pybind11 binding

Low-level optimizations for speed

Built-in parallelism in modern CPU: Single Instruction Multiple Data (SIMD)

Microarchitecture	ISA	throughput (per clock)	SIMD width	max flops/cycle
Nehalem	SSE	1 add + 1 mul	4	8
Sandy Bridge /Ivy Bridge	AVX	1 add + 1 mul	8	16
Haswell	AVX2	2 fmadds	8	32
Knights Corner	AVX-512F	1 fmadd(∗)	16	32
Knights Landing	AVX-512F	2 fmadds	16	32

Cache efficiency: communication bandwidth



Cache	Latency	CPU cycles	Size	
	L1 access	~1.2 ns	~4	Between 32 KB and 512 KB
	L2 access	~3 ns	~10	Between 128 KB and 24 MB
	L3 access	~12 ns	~40	Between 2 MB and 32 MB

The order of accessing data matters (improving spatially locality reduces cache miss)

Communication cost for multicore, multi-CPU, or multi-node computation

https://www.eidos.ic.i.u-tokyo.ac.jp/~tau/lecture/parallel_distributed/2016/slides/pdf/simd.pdf
https://teivah.medium.com/go-and-cpu-caches-af5d32cc5592
https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html

Fast algorithm design!

Fast algorithms:

Barnes-Hut t-SNE

Fast interpolation-based t-SNE

Randomized SVD

Basic ideas:

FFT

Nearest neighbors

Tree

Random projection

Coarse graining

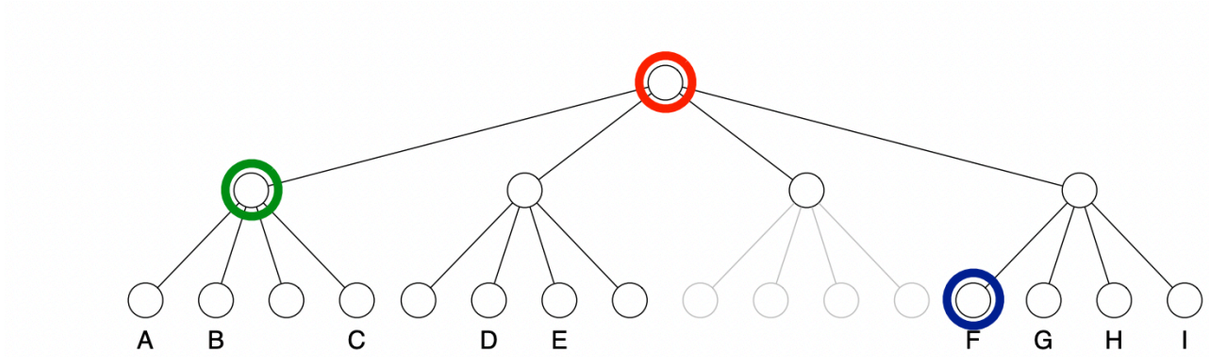
Barnes-Hut t-SNE

O(N log N).

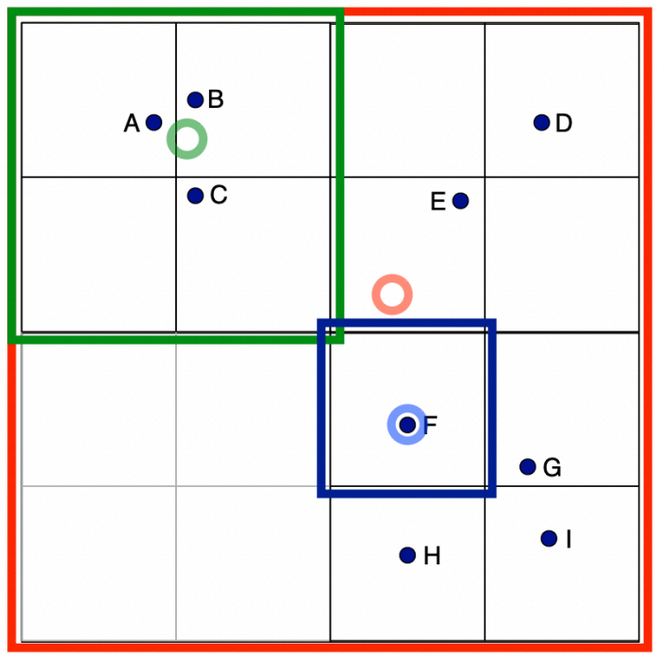
The objective can be viewed as pairwise attractive and repulsive forces between all cells

$$\frac{\partial C}{\partial \mathbf{y}_i} = 4(F_{attr} + F_{rep}) = 4\left(\sum_{j \neq i} p_{ij} q_{ij} Z(\mathbf{y}_i - \mathbf{y}_j) - \sum_{j \neq i} q_{ij}^2 Z(\mathbf{y}_i - \mathbf{y}_j)\right)$$

Borrow ideas from N-body simulation!



If points are sufficiently far away,
it is enough to approximate with average of a cell



Accelerating t-SNE using Tree-Based Algorithms, 2014

Fast interpolation-based t-SNE

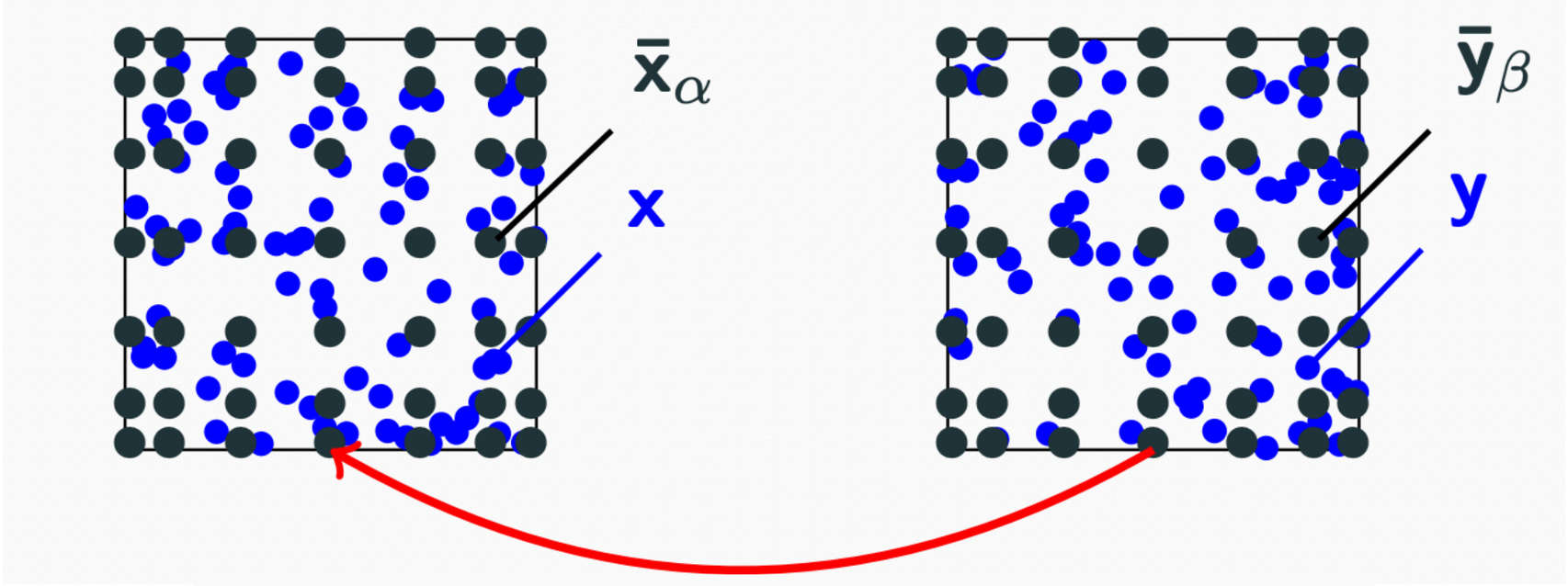
O(pN+p log p).

The objective can be viewed as pairwise attractive and repulsive forces between all cells

$$\frac{\partial C}{\partial \mathbf{y}_i} = 4(F_{attr} + F_{rep}) = 4\left(\sum_{j \neq i} p_{ij} q_{ij} Z(\mathbf{y}_i - \mathbf{y}_j) - \sum_{j \neq i} q_{ij}^2 Z(\mathbf{y}_i - \mathbf{y}_j)\right)$$

Borrow ideas from N-body simulation!

Approximate pairwise forces with interpolant with fixed spacing (fast FFT-based acceleration available)



$$\tilde{\mathbf{k}}(\mathbf{x}, \mathbf{y}) = \sum_{|\alpha| \leq p} S(\bar{\mathbf{x}}_\alpha, \mathbf{x}) \sum_{|\beta| \leq p} \mathbf{k}(\bar{\mathbf{x}}_\alpha, \bar{\mathbf{y}}_\beta) S(\bar{\mathbf{y}}_\beta, \mathbf{y})$$

O(p^2+p N).

Fast interpolation-based t-SNE for improved visualization of single-cell RNA-seq data

Fast interpolation-based t-SNE O(pN+p log p).

Further acceleration with FFT (avoiding the p^2 computation):

$\mathbf{k}(\bar{\mathbf{x}}_\alpha, \bar{\mathbf{y}}_\beta)$ between equidistant points can be embedded into a Toeplitz matrix (many relative distances are repeated)

Example Toeplitz matrix

$$\begin{bmatrix} a_0 & a_{-1} & a_{-2} & a_{-3} \\ a_1 & a_0 & a_{-1} & a_{-2} \\ a_2 & a_1 & a_0 & a_{-1} \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \qquad \begin{bmatrix} 7 & 11 & 5 & 6 \\ 3 & 7 & 11 & 5 \\ 8 & 3 & 7 & 11 \\ 1 & 8 & 3 & 7 \end{bmatrix}$$

Toeplitz matrix can be embedded into a circulant matrix like this

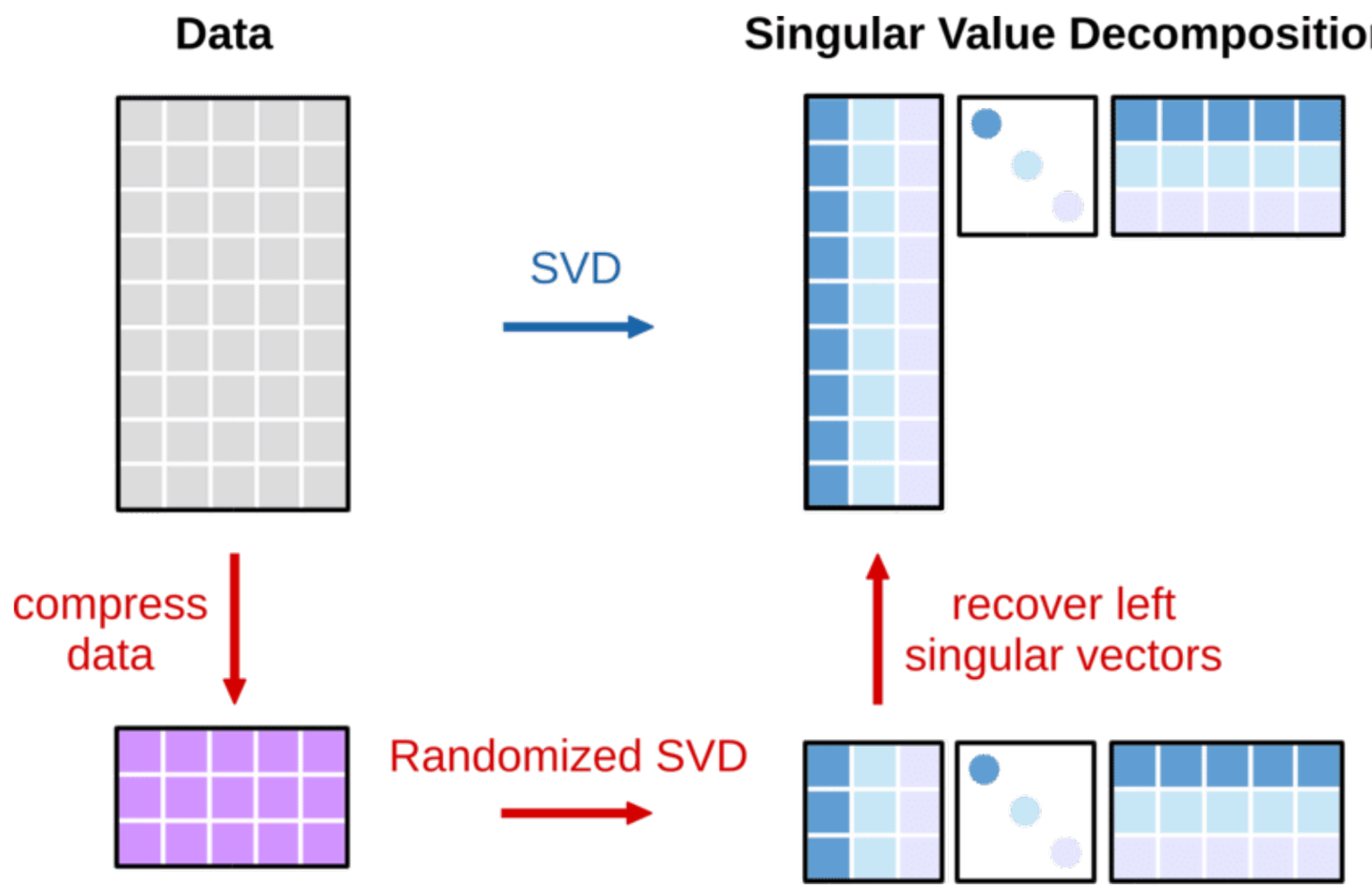
$$C_8 = \begin{bmatrix} a_0 & a_{-1} & a_{-2} & a_{-3} & \mathbf{a_0} & \mathbf{a_3} & \mathbf{a_2} & \mathbf{a_1} \\ a_1 & a_0 & a_{-1} & a_{-2} & \mathbf{a_{-3}} & \mathbf{a_0} & \mathbf{a_3} & \mathbf{a_2} \\ a_2 & a_1 & a_0 & a_{-1} & \mathbf{a_{-2}} & \mathbf{a_{-3}} & \mathbf{a_0} & \mathbf{a_3} \\ a_3 & a_2 & a_1 & a_0 & \mathbf{a_{-1}} & \mathbf{a_{-2}} & \mathbf{a_{-3}} & \mathbf{a_0} \\ \mathbf{a_0} & \mathbf{a_3} & \mathbf{a_2} & \mathbf{a_1} & a_0 & a_{-1} & a_{-2} & a_{-3} \\ \mathbf{a_{-3}} & \mathbf{a_0} & \mathbf{a_3} & \mathbf{a_2} & a_1 & a_0 & a_{-1} & a_{-2} \\ \mathbf{a_{-2}} & \mathbf{a_{-3}} & \mathbf{a_0} & \mathbf{a_3} & a_2 & a_1 & a_0 & a_{-1} \\ \mathbf{a_{-1}} & \mathbf{a_{-2}} & \mathbf{a_{-3}} & \mathbf{a_0} & a_3 & a_2 & a_1 & a_0 \end{bmatrix}$$

Circulant matrix - vector multiplication can be accelerated with FFT in O(n log n) time instead of O(n^2)

$C_n x = \text{DFT}^{-1}(\text{DFT}(a_n) \odot \text{DFT}(x))$ Each DFT or inverse DFT step take n log n time

Fast algorithm design: Randomized algorithms

Singular value decomposition m-by-n matrix A, m>n $O(mn^2)$



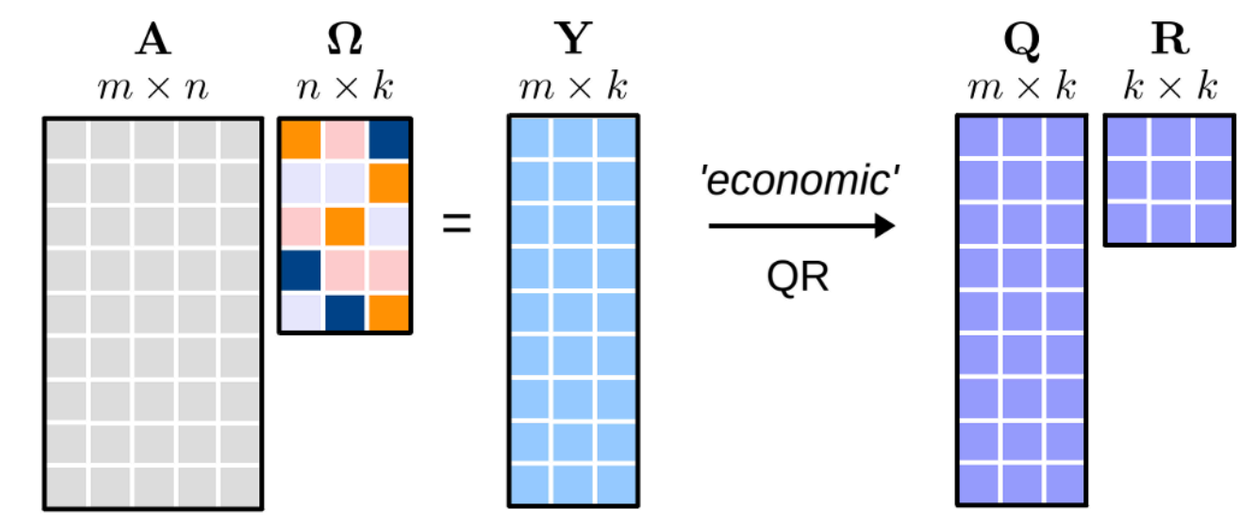
Fast algorithm design: Randomized algorithms

Randomized SVD (sketch)

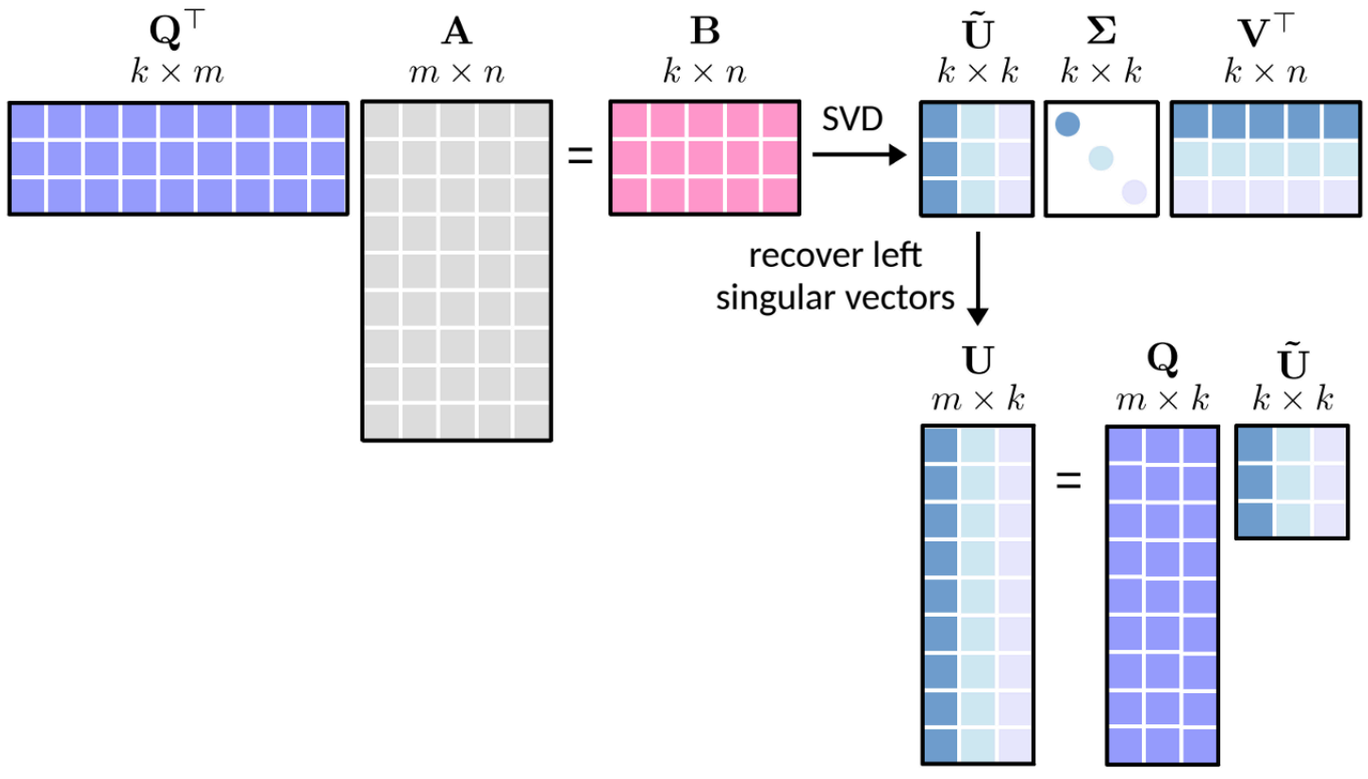
rank k decomposition for m-by-n matrix A, m>n

$$O(mnk + mk^2 + nk^2)$$

Step 1.



Step 2.



Why?

$$A \approx QQ^T A = Q(\tilde{U}\Sigma V^T)$$
$$\approx U\Sigma V^T$$

Reference: Halko, N., Martinsson, P.-G., & Tropp, J. A. (2011). Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, *53*(2), 217–288.
Image source: N. Benjamin Erichson, Sergey Voronin, Steven L. Brunton, J. Nathan Kutz (2016). Randomized Matrix Decompositions Using R. arXiv:1608.02148

Anatomy of python scientific software

Code

Code

Command-line interface (CLI)

Docstring

Auto-generation

Supporting docs

Frontpage (Github README.md or webpage)
- Introduction
- Installation
- Quick Start Guide

Documentation
- API documentation
- Tutorial

LICENSE (Copyright and Usage)
- e.g. BSD 3-Clause Clear

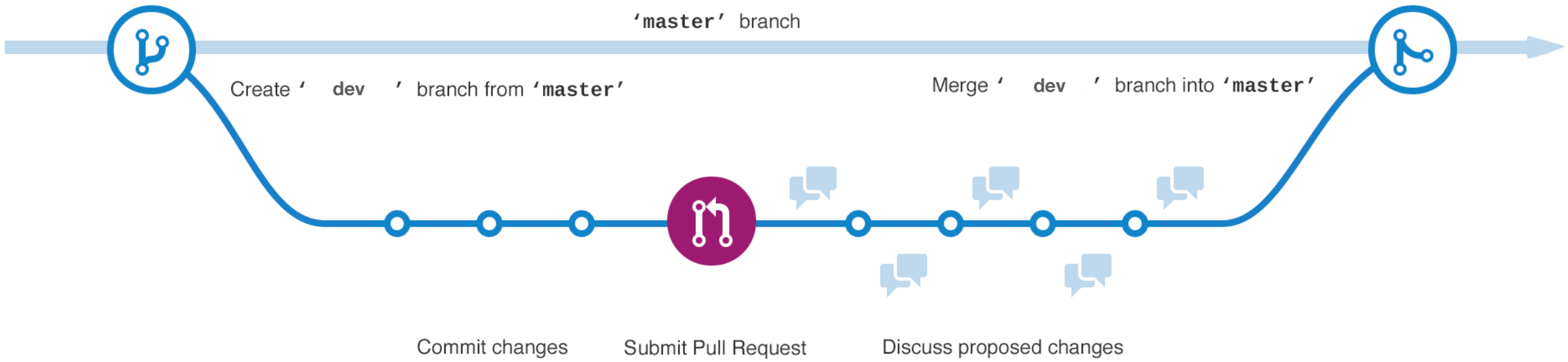
Create packages

pip (pypi)
conda (bioconda)

Deployment

Web server
for computational intensive applications (machine learning models) or data intensive (e.g. database) applications

Github workflow



Task for Day 2:

1. **Code review (set up Github, create dev branch and pull request)**
2. **Respond to code review (after approval by your reviewer, merge your code)**
3. **Write docstring. Optional: implement an CLI with capability to take input file and generate output file / plots; implement scikit-learn Estimator CLI**

Docopt : simplest way to make a CLI that parses command line input

```
"""Naval Fate.

Usage:
  naval_fate.py ship new <name>...
  naval_fate.py ship <name> move <x> <y> [--speed=<kn>]
  naval_fate.py ship shoot <x> <y>
  naval_fate.py mine (set|remove) <x> <y> [--moored | --drifting]
  naval_fate.py (-h | --help)
  naval_fate.py --version

Options:
  -h --help     Show this screen.
  --version     Show version.
  --speed=<kn>  Speed in knots [default: 10].
  --moored      Moored (anchored) mine.
  --drifting    Drifting mine.

"""
from docopt import docopt


if __name__ == '__main__':
    arguments = docopt(__doc__, version='Naval Fate 2.0')
    print(arguments)
```

https://github.com/jazzband/docopt-ng

http://try.docopt.org/