5 Steps to Speed Up Your Data-Analysis on a Single Core

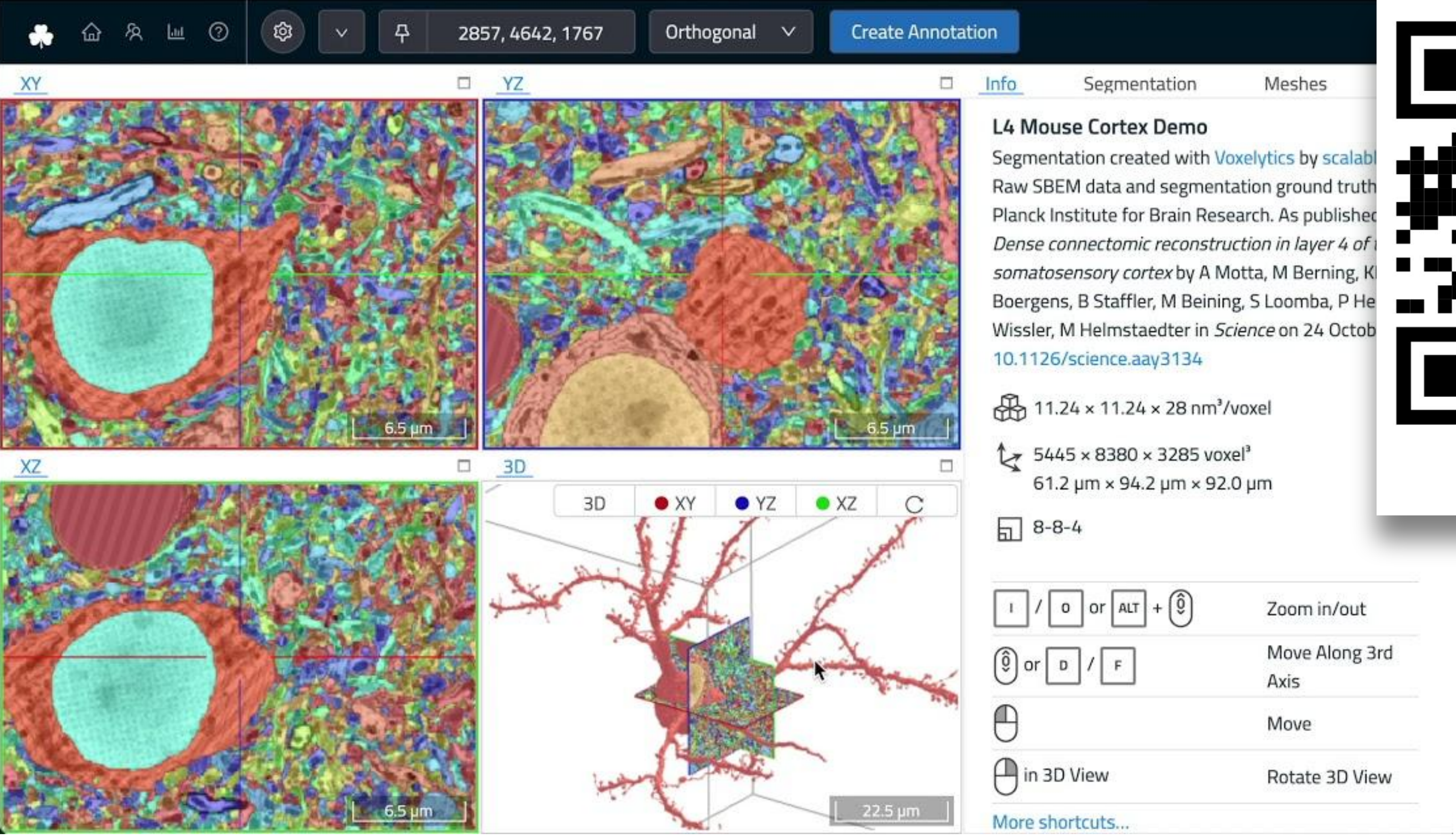
Jonathan Striebel

Hi, I’m **Jonathan Striebel**.

@jostriebel

jonathan@scalableminds.com

scalableminds



webknossos.org

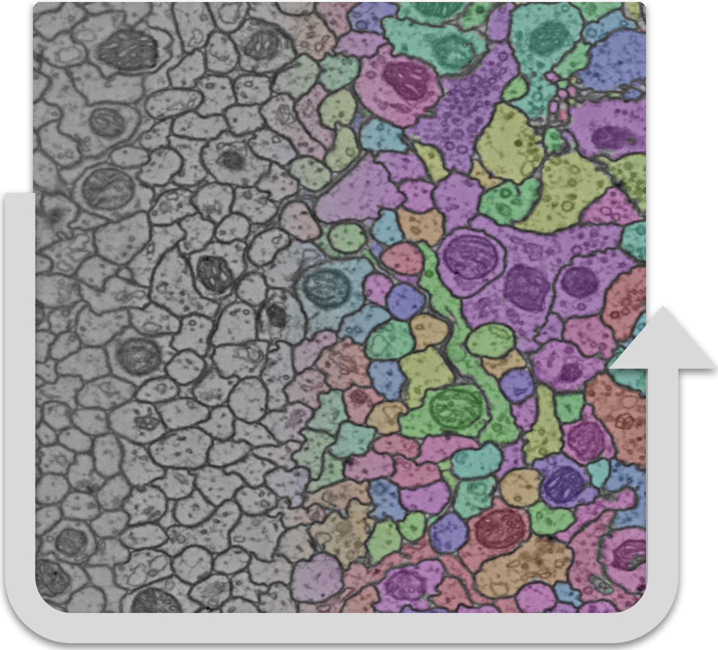
Python library: `pip install webknossos`

see docs.webknossos.org/webknossos-py

Large Scale Data-Analysis Experiments

PB-scale image data

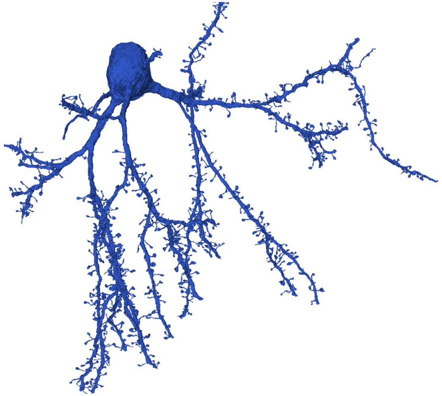
Machine Learning Systems



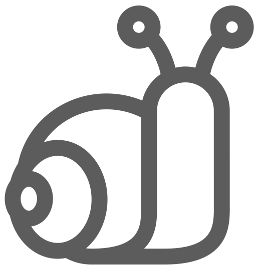
Segmentation & Agglomeration Algorithms

Running weeks in HPC Clusters

Neuron Reconstructions for Biological Analysis

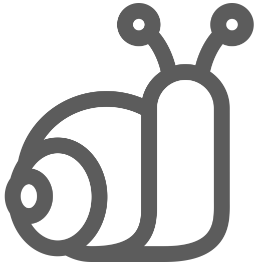


Why to Speed Up



It’s too slow

Why to Speed Up: Example

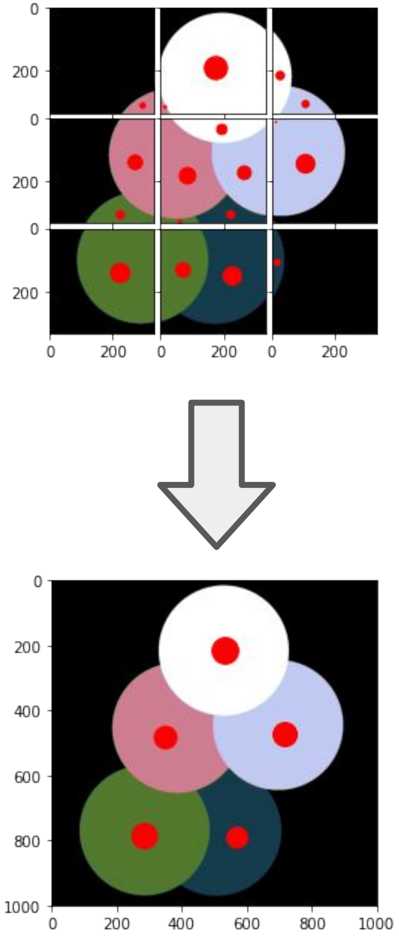


Combining statistics of billions of segments across thousands of chunks

1. ∞ (crashes with OOM after 2 days, 1TB memory)
2. 18h (3-4 iterations per week)
3. 7h (5-8 iterations per week)

500+% faster IRL

Toy Example: ~200% faster



Why to Speed Up on a Single Core

Don’t parallelize (yet):

- Single core improvements **pay off, also when parallelized** later
- Parallelization **needs resources**

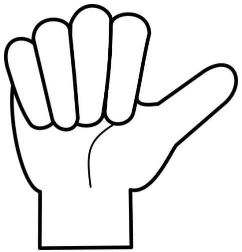
  (cores, memory, cluster-nodes, money, time)
- Code may be **hard** / **impossible** to parallelize

  (e.g. when reducing results from a map-reduce)

5 Steps

- Profiling
- Efficient IO
- Vectorization
- Memory & Precision Tradeoffs
- Jit-ting with numba

1. Profiling



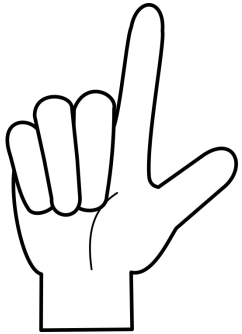
Speed

- **py-spy** (sampling based, flamegraphs, speedscopes)
- **yappy** (line-wise)
- cProfile (ships with Python)
- pyinstrument
- Palanteer

Memory

- memory-profiler
- Guppy3

2. Efficient IO



50-70% faster

Text-based

csv

json

yaml

Binary format

hdf5

npy

parquet

pickle

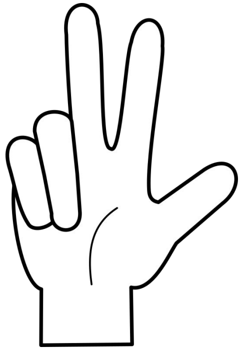
sqlite

zarr

...

see also
github.com/mverleg/array_storage_benchmark

3. Vectorization



170-180% faster

NumPy

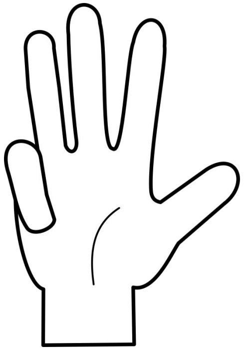
compact representation for numerical arrays
with optimized functions

Pandas

tabular data backed by numpy

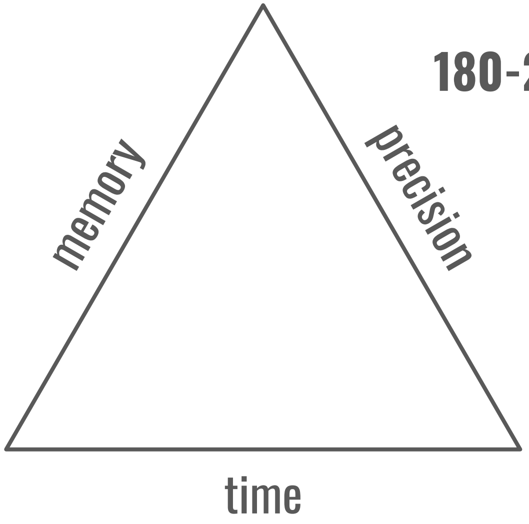
often semi-fast, consider alternatives:
Polars, datatable, PandasPy, modin (multi-threaded), vaex (out-of-core), dask-dataframe (parallel)

4. Memory & Precision Tradeoffs

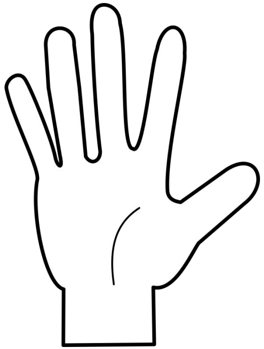


180-200% faster

- Data type
  - direct effect on memory & precision
  - time affected via read/write/operations on data
- Iterative methods (e.g. divide & conquer, reduce)
  - less data to keep in memory
  - iterations might lose numerical precision
- Lookup tables
  - time vs memory
- Compression
  - lossless: time vs memory
  - lossy: time vs memory vs precision



5. Jit-ting with numba



190-200% faster

1. Add `@numba.njit` decorator, enables jitting in nopython mode

2. Fix shape & dtype errors, as broadcasting is more strict in numba

3. 🚀

Further Steps

- Upgrade dependencies & Python
- Faster Python Runtimes
  PyPy, Pyjion, Cinder
- Optimize critical code-paths closer to the metal
  Cython, pybind11, cffi, PyO3, ONNX, ...
- Parallelization
  async, threading, multiprocessing, Spark/Dask/Ray, ...

Summary

- Profiling
- Efficient IO
- Vectorization
- Memory & Precision Tradeoffs
- Jit-ting with numba

Jonathan Striebel | jonathan@scalableminds.com @jostriebel

Thanks!

code @

github.com/jstriebel/data-analysis-speedup

Interested in working at scalable minds?

We’re looking for Python Engineers, Scala Backend Engineers

Jonathan Striebel | jonathan@scalableminds.com @jostriebel