Large Scale Analysis of Bioimages Using Python

LUIS PEDRO COELHO

coelho@embl.de

On twitter: @luispedrocoelho

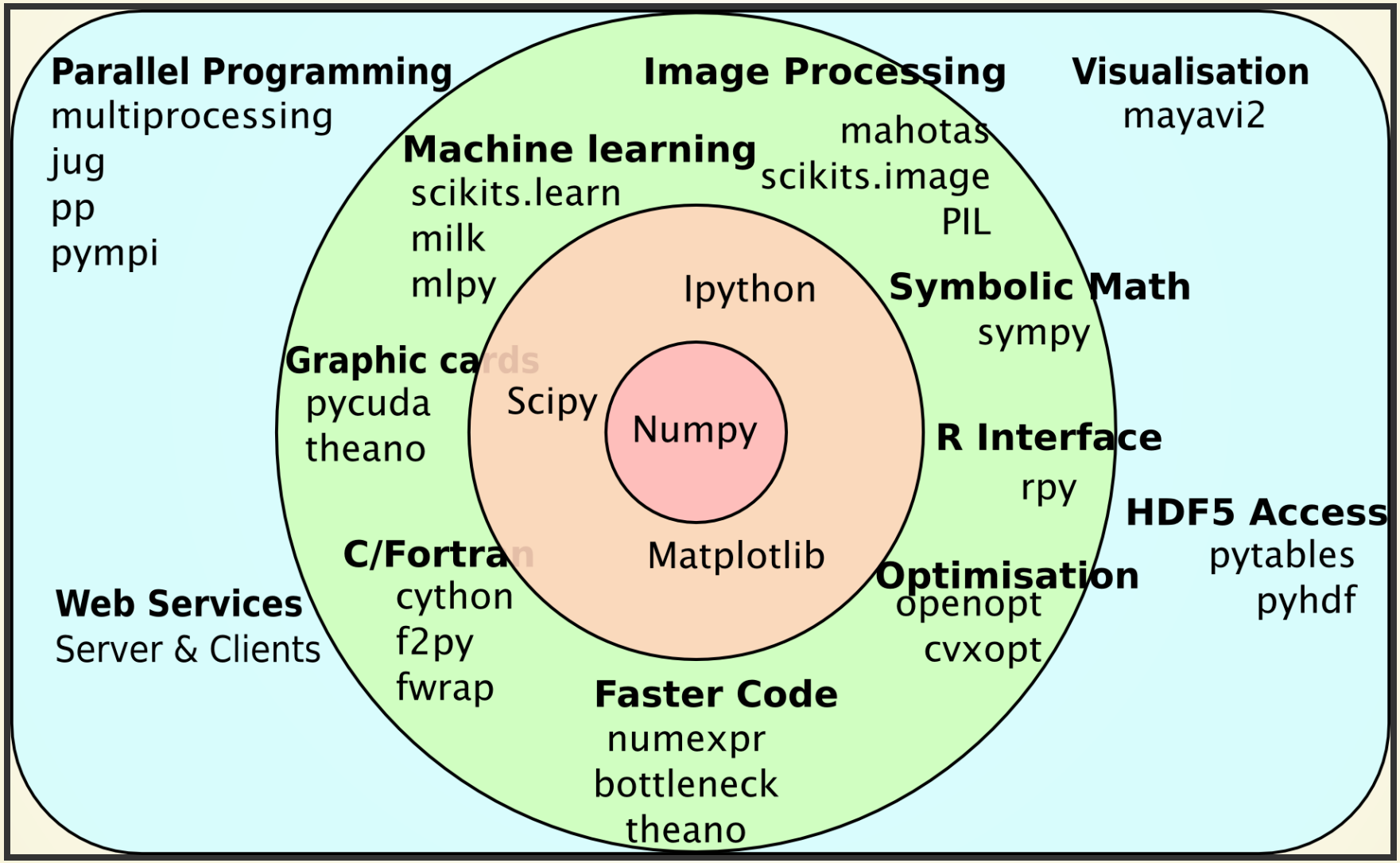
PYTHON HAS A GOOD ECOSYSTEM FOR DATA ANALYSIS

- NumPy
- Matplotlib
- IPython
- Scikit-learn
- Mahotas

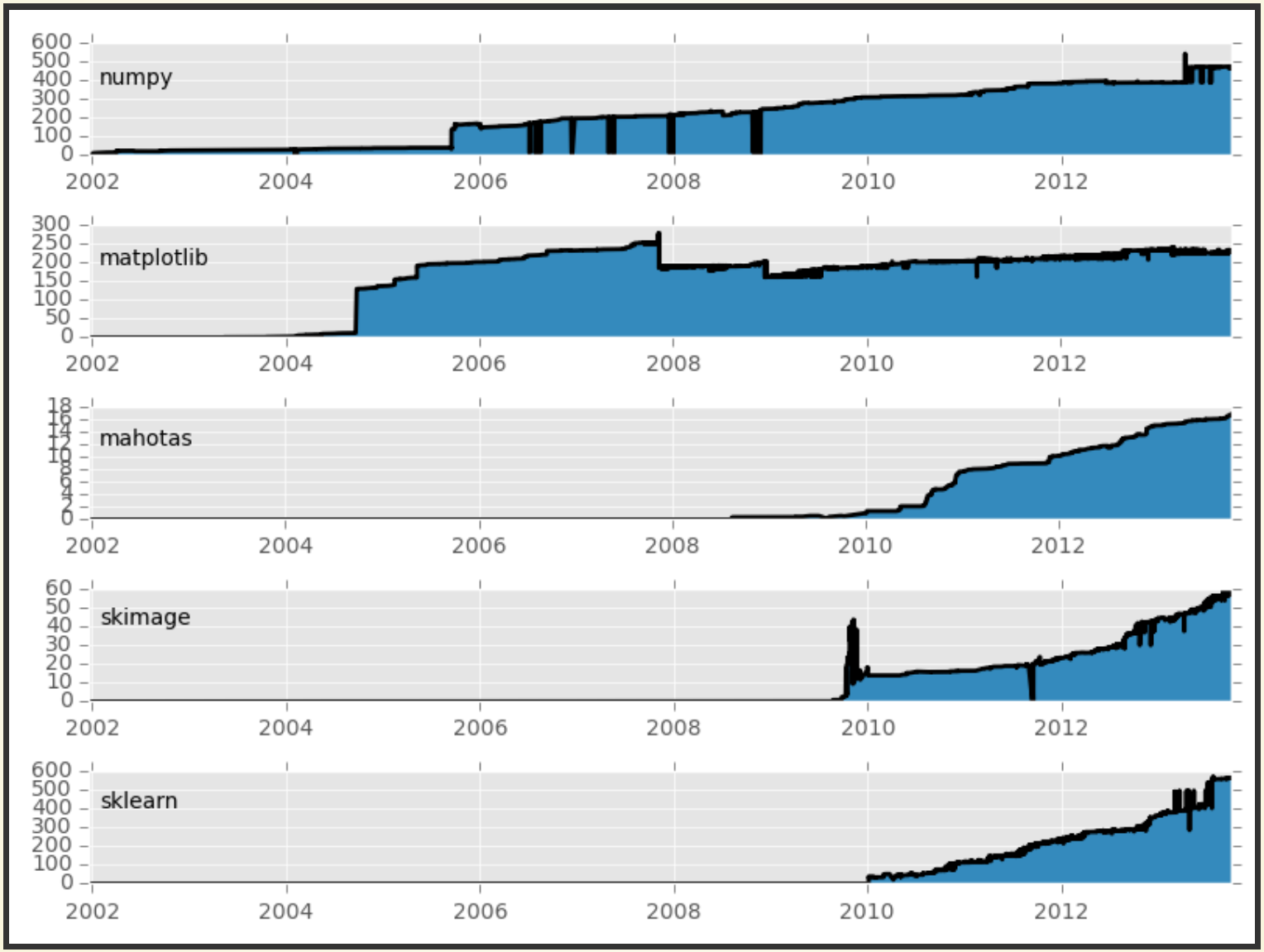
PYTHON HAS A GROWING ECOSYSTEM OF SCIENTIFIC PACKAGES AROUND NUMPY

Numpy provides basic data types (arrays, matrices).
Packages provide intelligence.

THE WIDER ECOSYSTEM



THE WIDER ECOSYSTEM



MULTIPLE PACKAGES ACT TOGETHER

Mahotas can rely on pre-existing functionality

1. An image type (numpy array).
2. Types to hold computed data (numpy array again).
3. Plotting & displaying (matplotlib).
4. Machine learning (sklearn or milk).

MODULARITY IS GOOD SOFTWARE ENGINEERING

- Improvements to one package benefit all.
- Separation of concerns.

CONSISTENCY ALSO HELPS HUMAN USERS

- Single type for many uses.
- Many simple operations can be done in numpy.
- Same basic conventions.
- No copying/conversion of data between packages.

MAHOTAS: COMPUTER VISION IN PYTHON

- Works with NumPy arrays
- Basic blocks necessary for image processing
- Sister package: mahotas-imread for parsing image files
- Modern open-source: github, automated tests, up-to-date documentation...

DEMONSTRATION

1. Load an image
2. Basic smoothing & thresholding

CODE

```
import mahotas as mh
from matplotlib import pyplot as plt

im = mh.imread('image_stretched.jpeg')
sigma = 2.3
imf = mh.gaussian_filter(im.mean(2), sigma)
binary = (imf > imf.mean())
labeled, _ = mh.label(binary)
plt.imshow(labeled)
```

IMPLEMENTATION IS IN C++

- Fast C++ code with a Python interface
- C++ templates allow for specialization
- Hand-written interface code

```
template<typename T>
void bbox(const numpy::aligned_array<T> array,
                    numpy::index_type* extrema) {
    gil_release nogil;
    const int N = array.size();
    typename numpy::aligned_array<T>::const_iterator pos = array.begin();
    for (int i = 0; i != N; ++i, ++pos) {
        if (*pos) {
            numpy::position where = pos.position();
            for (int j = 0; j != array.ndims(); ++j) {
                extrema[2*j] = std::min<numpy::index_type>(
                                        extrema[2*j], where[j]);
                extrema[2*j+1] = std::max<numpy::index_type>(
                                        extrema[2*j+1], where[j]+1);
            } } } }
```

RESULT IS FAST, TYPE-SAFE & FLEXIBLE

- Fast as it is compiled without runtime type checks
- Type-safe due to template wrappers
- Flexible as the interface is in Python
- Compilation in debug mode inserts many run-time checks

INTERFACES ARE HAND-WRITTEN

- Automated generators exist (swig)
- They work very well, but give poor error messages
- I find good error messages important

SCIKIT-IMAGE IS ANOTHER GOOD ALTERNATIVE

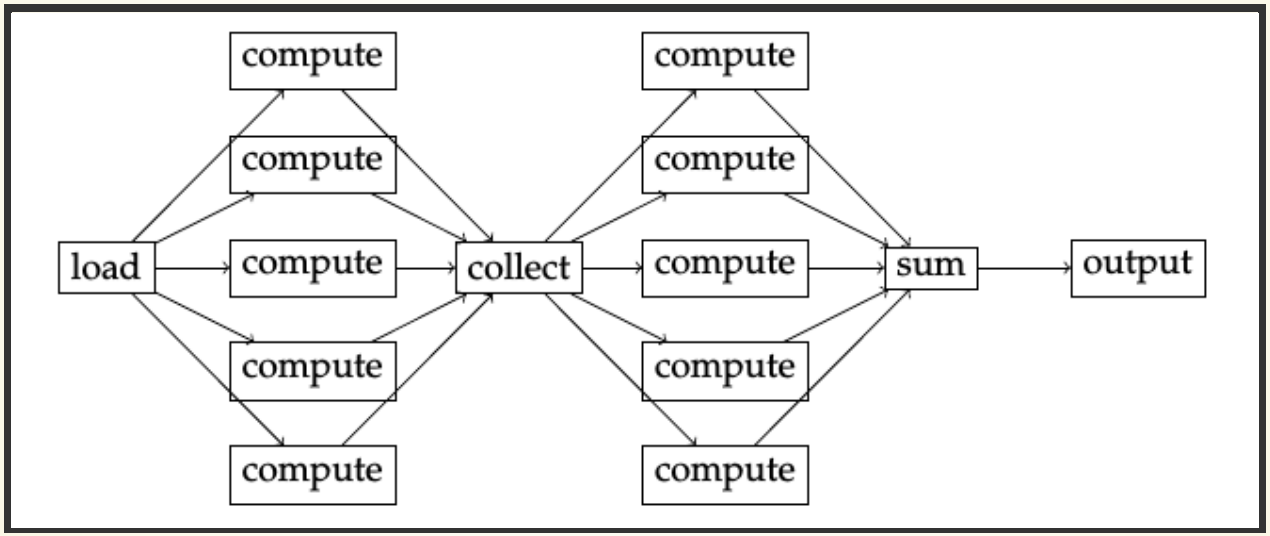
- Uses Cython instead of C++
- More heavily focused on natural images
  (Mahotas is heavily influenced by scientific images)

JUG FOR LARGE SCALE ANALYSIS

- Reproducibility
- Parallelism
- Memoization

JUG USE CASES

- Parameter sweeps
- Preprocessing of large data
- Embarassingly parallel problems
- Coarse grained parallelism



JUG TASKS

- A *Task* is a Python function & its arguments
- Arguments can be either values or results of other Tasks
- Thus, Tasks implicitly define a *dependency graph*
- A Task is identified by its hash
- Hash of function name + arguments (recursively)

DESIGN DECISIONS

- Code is not taken into account for hash.
- This means that changing the code will not trigger recomputation
- Explicit **invalidate** operation triggers recomputation *of all dependent tasks*

JUG EXECUTION LOOP

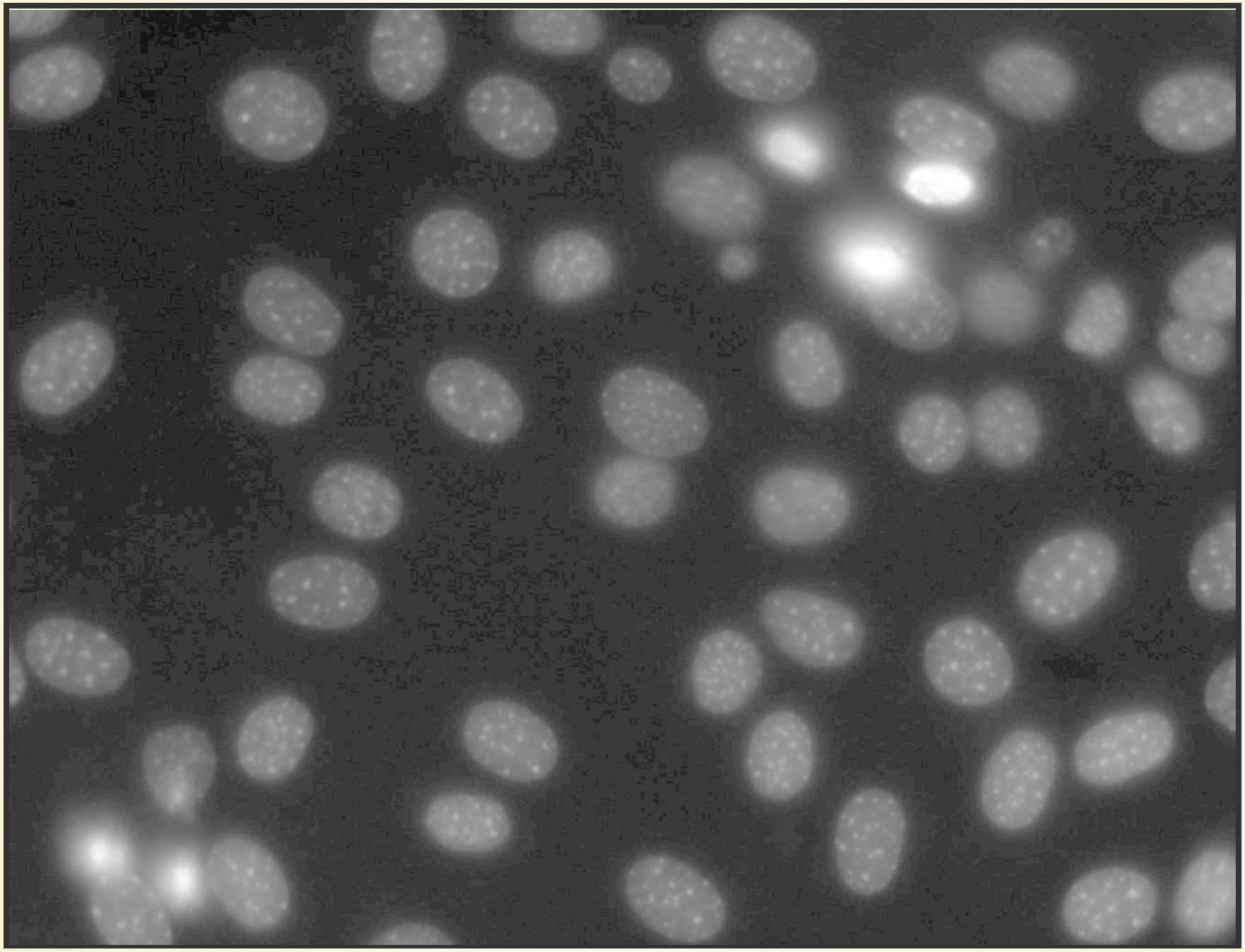
```
for task in alltasks:
    if backend.has_data(task):
        print("Task is already done")
    elif backend.has_data(task.dependencies()):
        if backend.lock(task):
            r = task.execute()
            backend.write(task.hash(), r)
```

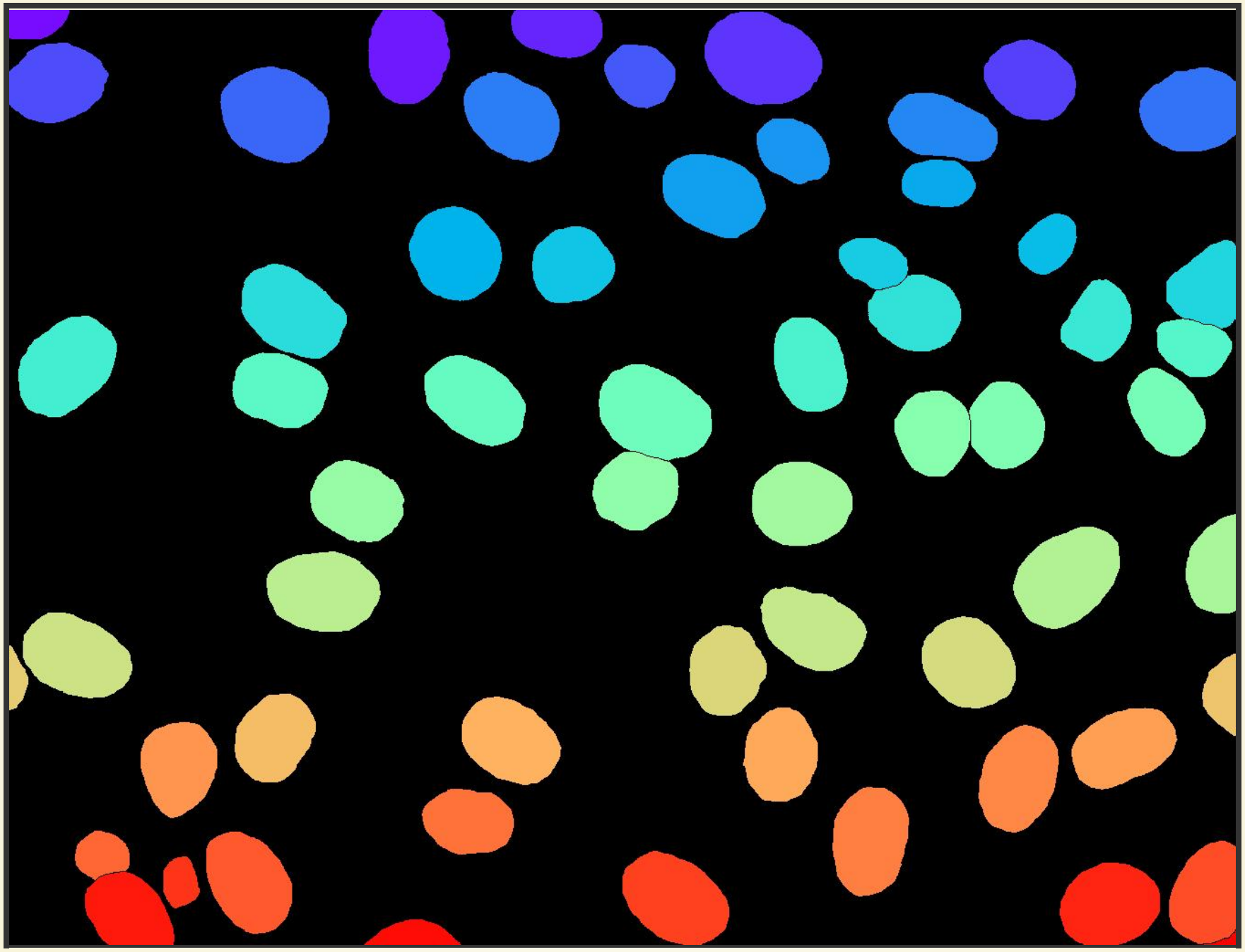
PROCESSES COMMUNICATE THROUGH BACKEND

- All communication between processes is through the backend
- User must start processes manually.
- Backend handles all locking issues
  (jobs are separate processes).
- Jobs may start at different times.
- Backend is very simple: key/value store + locking
- Two backends available: filesystem & Redis

EXAMPLE (DEMO)

- Some images that you want to segment.
- Compare to a gold standard (hand-segmentation).





```
import mahotas as mh
def method1(image, sigma):
    image = mh.imread(image)[:,:,0]
    image  = mh.gaussian_filter(image, sigma)
    binimage = (image > image.mean())
    labeled, _ = mh.label(binimage)
    return labeled
```

NOW, WE DO A LIVE DEMO...

```
import mahotas as mh
from jug import TaskGenerator
from glob import glob
@TaskGenerator
def method1(image, sigma):
    ...
    return labeled

@TaskGenerator
def print_results(...):
    ...

inputs = glob('images/*.jpg')
results = []
for im in inputs:
    m1 = method1(im, 2)
    m2 = method2(im, 4)
```

JUGFILE IS LIKE PYTHON EXCEPT FOR TASKGENERATOR

Decorator magic, but without decorators it is still simple:

```
from jug import TaskGenerator
@TaskGenerator
def method1(image, sigma):
    ...
    return labeled

m1 = method1(im, 2)
```

is equivalent to

```
from jug import Task
def method1(image, sigma):
    ...
    return labeled

m1 = Task(method1, im, 2)
```

SOME DETAILS

- Anything that can be serialized by Python (pickle) will work fine.
- Special support for NumPy arrays (speed/disk usage)
- Works with filesystem (including working well on NFS).
- Alternatively, use Redis

TO RUN ON A CLUSTER, USE THE CLUSTER INTERFACE

```
#!/bin/bash

# SOME SETUP CODE MIGHT GO HERE

jug execute
```

Then use your cluster interface:

```
qsub run-jug.sh
```

SOME MORE OPERATIONS

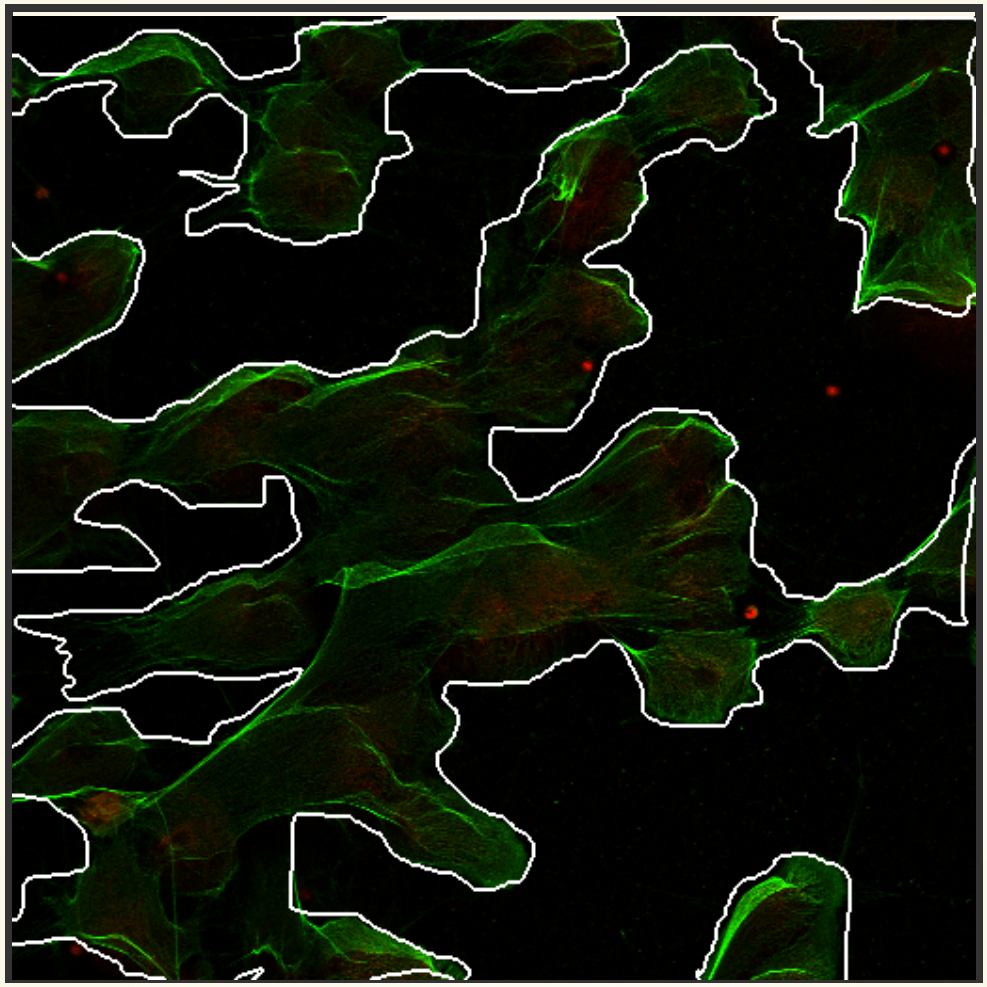
- jug cleanup: run garbage collection on the backend
- jug sleep-until: wait until all tasks are finished

EXAMPLE APPLICATION

- Quantification of Neutrophil Extracellular Traps (NETs)
- Neutrophils physically ensnare bacteria by exploding and using their DNA fibers to build a net

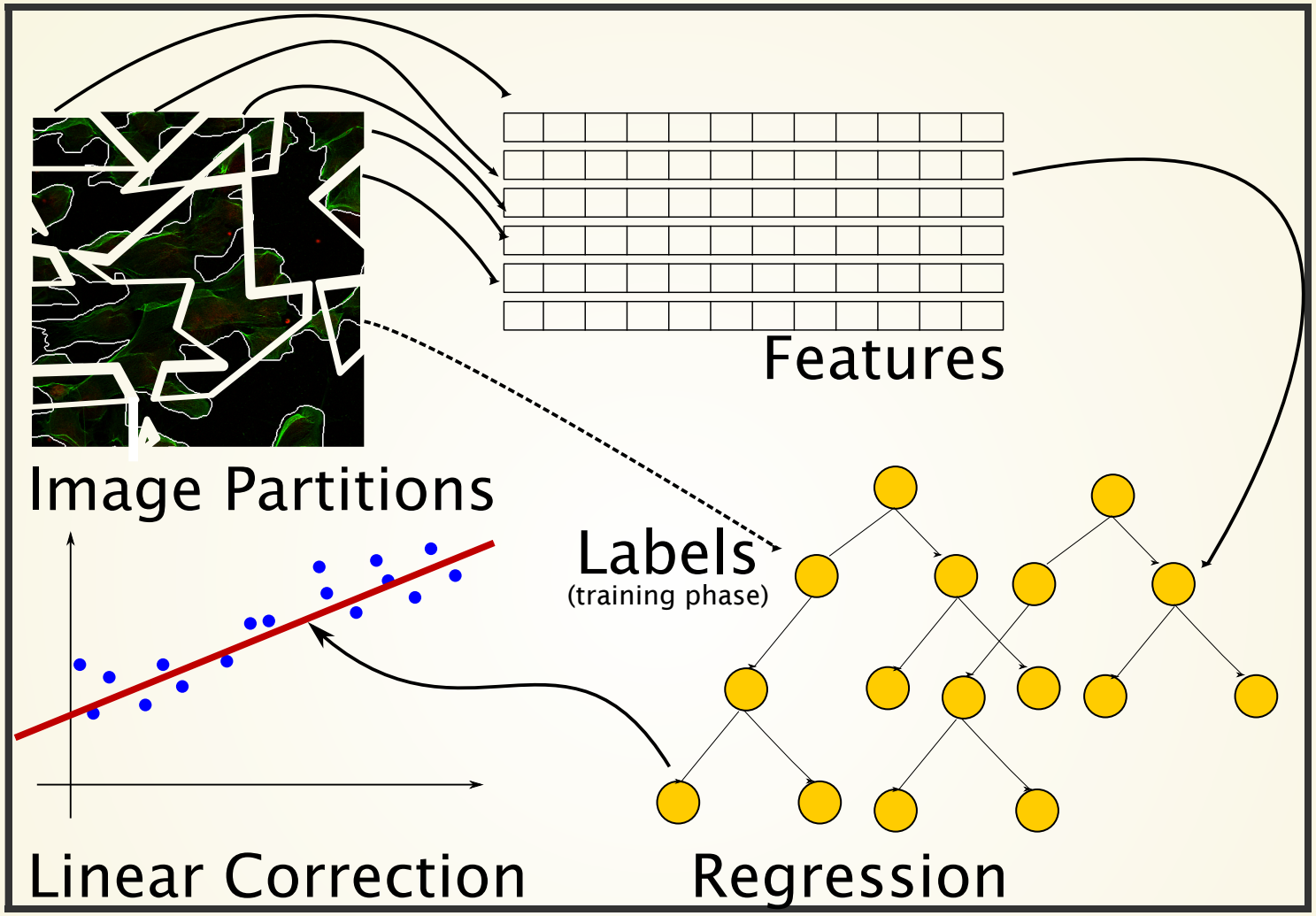
(This is work currently under review.)

CAN WE USE A SUPERVISED APPROACH?



INITIAL REMARKS

- We do not need perfect segmentation (similar to *Learning to Count* framework)
- We want to try several methods


Features
Image Partitions
Labels
(training phase)
Linear Correction
Regression

DIFFERENT SUBMODULES LEAD TO DIFFERENT METHODS

- How to break up image?
- Which features to compute?
- Which regression module to use? (we always used random forests).

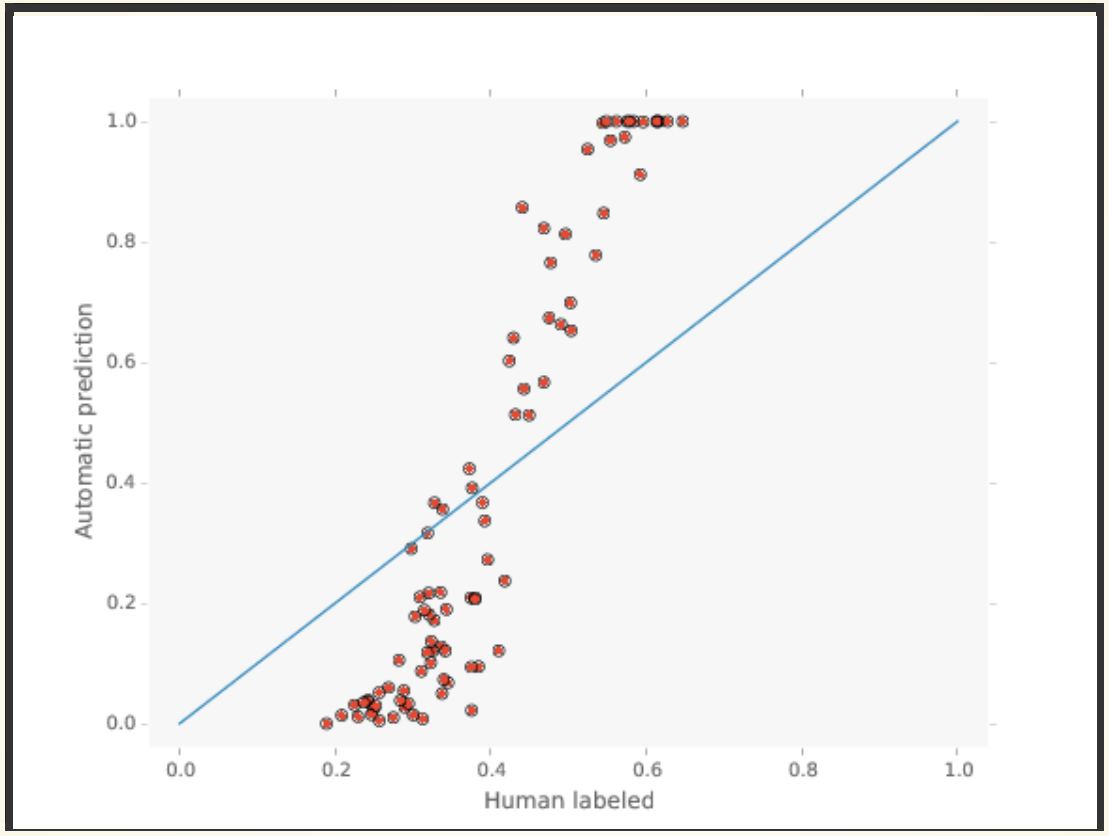
DIFFERENT WAYS TO BREAK UP IMAGE

1. Oversegmentation
2. Regular grid
3. Interest point detection

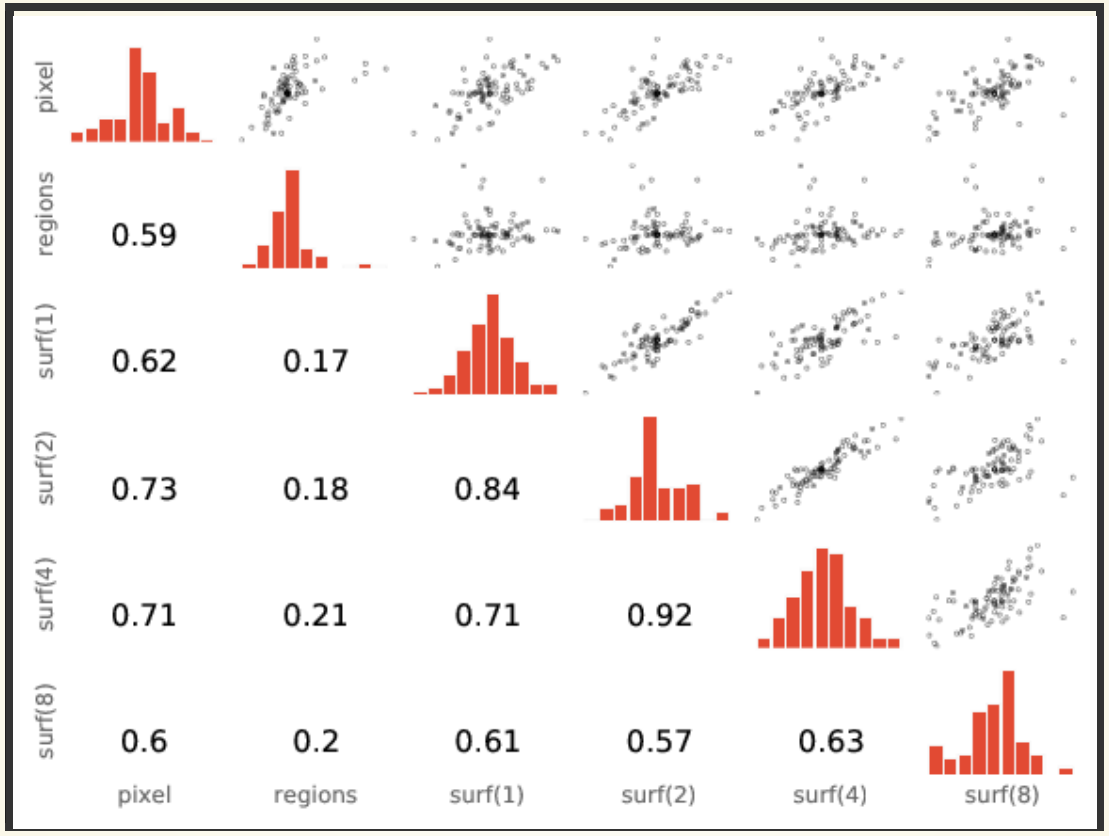
DIFFERENT WAYS TO COMPUTE FEATURES

- Texture features
- Image filterings
- SURF features (different scales)

INITIAL (RAW) RESULTS

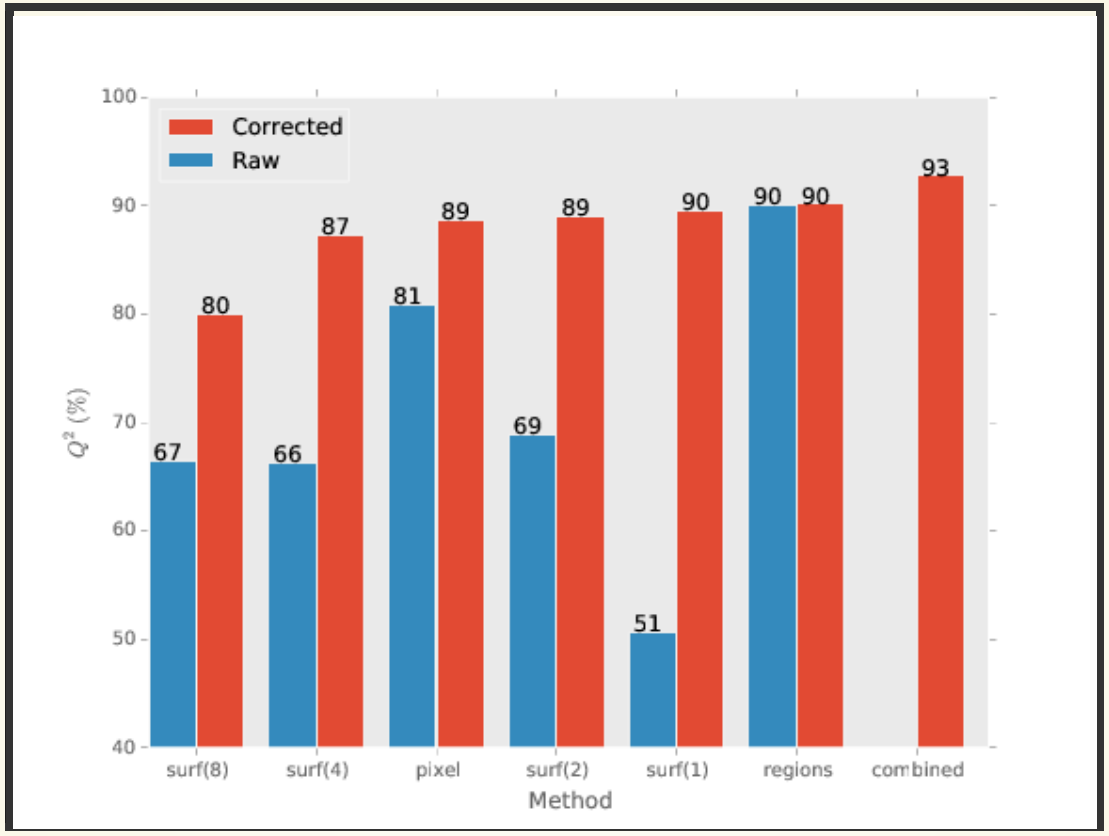


CAN WE COMBINE THE METHODS?



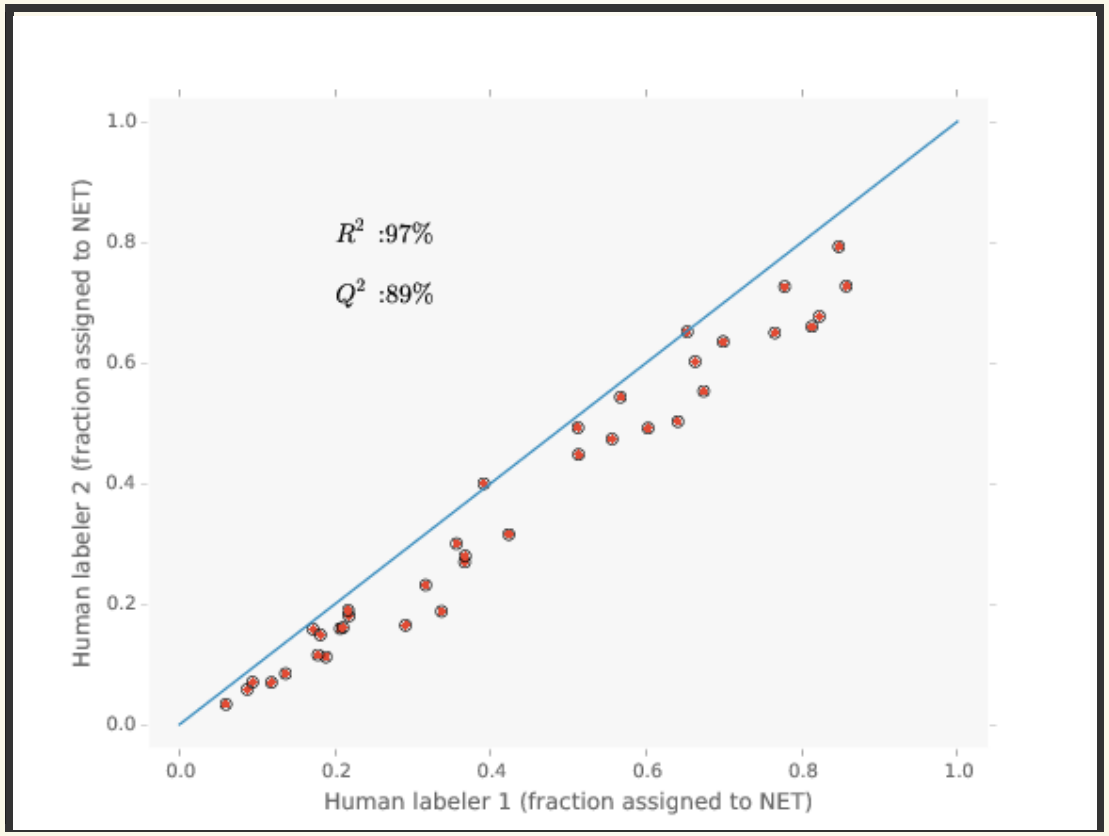
Above diagonal, we show a scatter plot of *residuals*.
Below diagonal, we show Pearson correlation of residuals.

LINEAR REGRESSION FOR COMBINATION

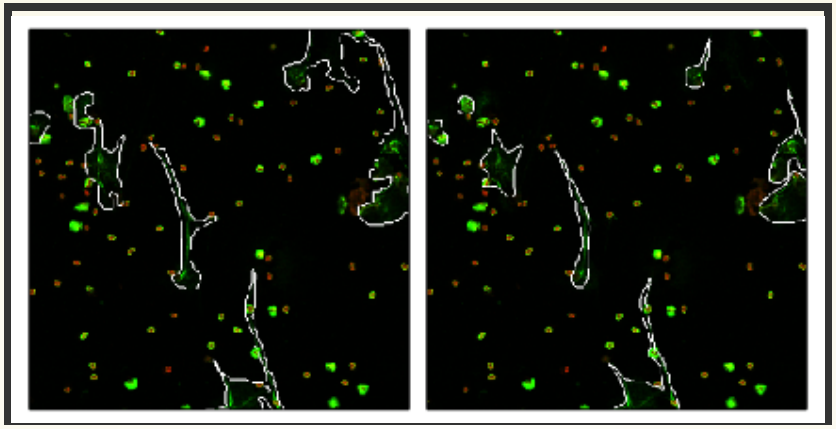


HOW GOOD IS 93 PERCENT?

Well, how good are humans (the baseline method)?



COMPARISON EXAMPLE



LABEL IT TWICE

- If your method uses human labeled data, *label it twice*
- Many cases, humans are not that great
- In our case, operator bias is major issue but bias seems consistent

CROSS-VALIDATE BY EXPERIMENT, NOT BY IMAGE!

- Each experiment resulted in several images from the same slide.
- Using images from the same experiment in training/testing results in a 3-6% Q^2 boost!
- Other work in related problems shows analogous effects.

WHOLE COMPUTATION IS MANAGED WITH JUG

- Prototype to publication to reproduction without breaks
- Easy to reproduce
- Smallish problem (100 CPU hours)
- We will make the code available when paper is accepted

Live demo time again...

CONCLUSIONS

- Python is great for data analysis
- Partially because it makes it easy to use non-Python
- Partially because of the ecosystem of packages
- Jug makes it easier to manage computations
- IPython is great for communication
- When using human-labeled data, label it twice
- Beware the cross-validation schedule

ACKNOWLEDGEMENTS

- The people who contributed patches, reports to my projects
- Catarina Pato
- Ana Friães
- Ariane Neumann
- Mário Ramirez
- Marien von Köckritz-Blickwede
- João Carriço

Thank You