Designing GPU-accelerated Image Data Flow Graphs for CLIJ2 and clEsperanto and deployment to ImageJ, Fiji, Icy, Matlab, QuPath, Python, napari and C++

Robert Haase DFG Cluster of Excellence Physics of Life, TU Dresden

Stéphane Rigaud Image Analysis Hub, C2RT, Institut Pasteur Paris

Introduction

Stéphane Rigaud
Image Analysis Hub, C2RT,
Institut Pasteur Paris

Robert Haase
Bio-image Analysis Technology Development group
DFG Cluster of Excellence “Physics of Life”, TU Dresden

Tutorial material download: https://github.com/clesperanto/i2k2020_tutorial_clij_clesperanto/

Acknowledements


Akanksha Jain (Treutlein lab) @jain_akanksha_


Dani Vorkel (Myers lab) @happifocus


Theresa Suckert (Krause lab, Oncoray)


Pradeep Rajasekhar (Poole lab, Monash U) @pr4deepr


Talley J. Lambert (HMS) @TalleyJLambert


Juan Nunez-Iglesias (Monash) @jnuneziglesias


Pavel Tomancak (CSBD / MPI-CBG) @PavelTomancak


Gene Myers (CSBD / MPI-CBG) @TheGeneMyers


Jean-Yves Tinevez (Institut Pasteur) @jytinevez


Florian Jug (CSBD / MPI-CBG) @florianjug


Szabolcs Horvát (Modes lab) @szhorvat


Johannes Girstmair (Tomancak lab) @jogirstmair


Brian Northon @truenorth_ia


Matthias Arzt (Myers/Jug lab)

Alex Herbert (University of Sussex)
Alexandr Dibrov (MPI CBG)
Bertrand Vernay (IGBMC, Strasbourg)
Bert Nitzsche (PoL TU Dresden)
Bradley Lowekamp (NIAID Washington)
Bram van den Broek (Netherlands Cancer Institute)
Brenton Cavanagh (RCSI)
Bruno C. Vellutini (MPI CBG)
Carl D. Modes (CSBD/MPI-CBG)
Curtis Rueden (UW-Madison LOCI)
Damir Krunic (DKFZ)
Daniel J. White (GE)
Deborah Schmidt (CSBD/MPI CBG)
Douglas P. Shepard (Univ Arizona)
Eduardo Conde-Sousa (University of Porto)
Erick Ratamero (The Jackson Laboratory)
Gaby G. Martins (IGC)
Gayathri Nadar (MPI CBG)
Guillaume Witz (Bern University)
Giovanni Cardone (MPI Biochem)
Irene Seijo Barandiaran (MPI CBG Dresden)
Jan Brocher (Biovoxxel)
Jean-Yves Tinevez (Institut Pasteur)
Johannes Girstmair (MPI CBG)
Juergen Gluch (Fraunhofer IKTS)
Kisha Sivanathan (Harvard Medical School)
Kota Miura
Laurent Thomas (Acquifer)
Lior Pytowski (University of Oxford)
Loic A. Royer (CZ Biohub)
Marion Louveaux (Institut Pasteur Paris)
Martin Weigert (EPFL Lausanne)
Matthew Foley (University of Sydney)
Nico Stuurman (UCSF)
Nicola Maghelli (MPI CBG)
Nicolas Brouilly (IBDM Marseille)
Nik Cordes (Los Alamos National Laboratory)
Noreen Walker (MPI CBG Dresden)
Ofra Golani (Weizmann Institute of Science)
Patrick Dummer
Peter Haub
Pete Bankhead (University of Edinburgh)
Peter Steinbach (HZDR Dresden)
Pit Kludig
Rita Fernandes (University of Porto)
Romain Guiet (BIOP-EPFL)
Ruth Whelan-Jeans
Sebastian Munck (VIB Leuven)
Siân Culley (MRC LMCB)
Stein Rørvik
Stéphane Dallongeville (Institut Pasteur)
Tanner Fadero (UNC-Chapel Hill)
Thomas Irmer (Zeiss)
Tobias Pietzsch (MPI-CBG)
Uwe Schmidt (MPI CBG)
Wilson Adams (VU Biophotonics)

https://fiji.sc

https://napari.dev

https://image.sc

http://eubias.org/NEUBIAS/

MPI CBG Core Facilities

- Advanced Imaging Facility
- Light Microscopy Facility
- Scientific Computing
- IT Department

Funding:

R.H. was supported by the German Federal Ministry of Research and Education (BMBF) under the code 031L0044 (Sysbio II) and by the Deutsche Forschungsgemeinschaft under Germany's Excellence Strategy – EXC2068 - Cluster of Excellence Physics of Life of TU Dresden.

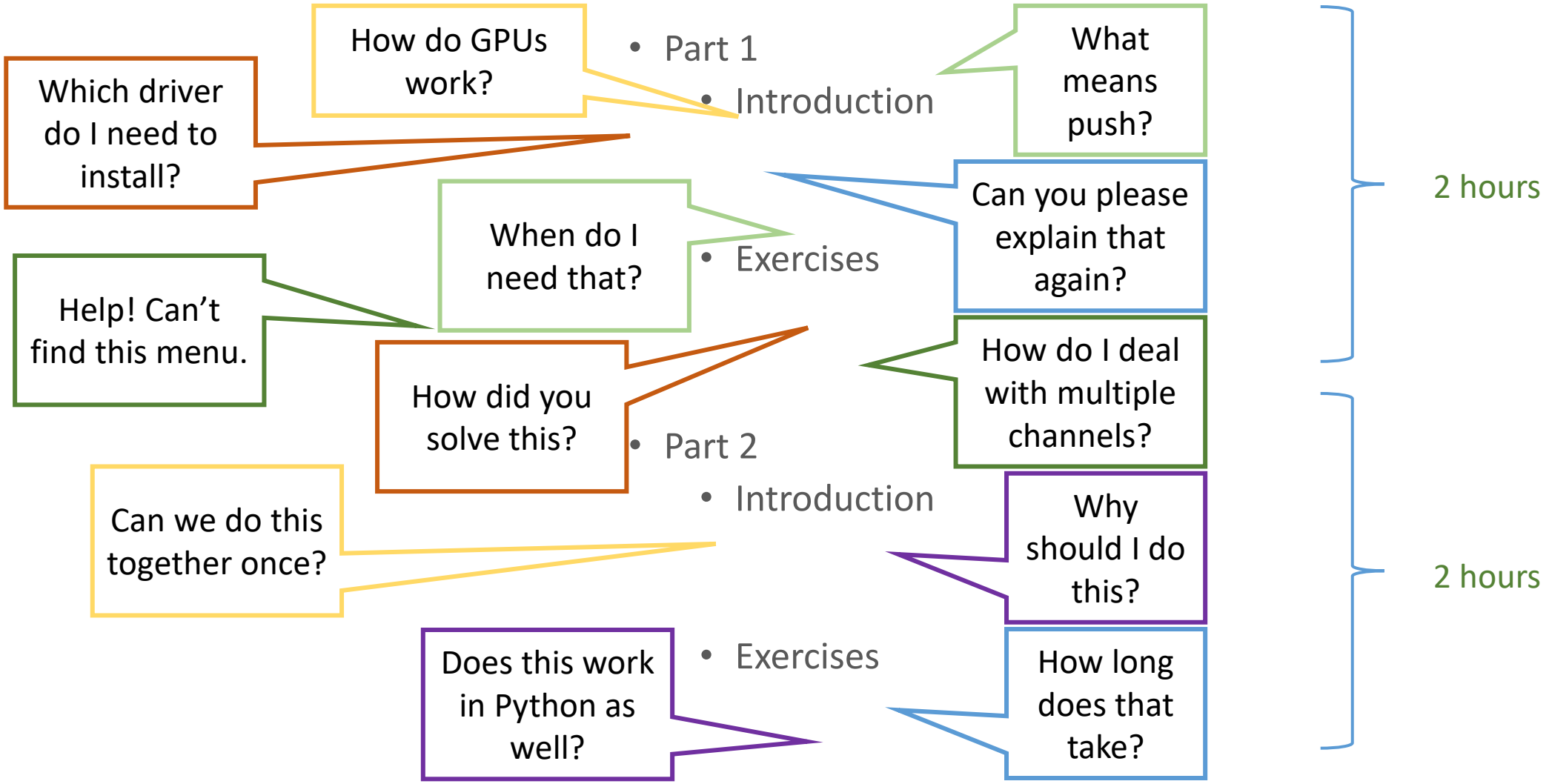
@haesleinhuepf @StRigaud

https://clij.github.io/

#I2K2020, virtually in Janelia

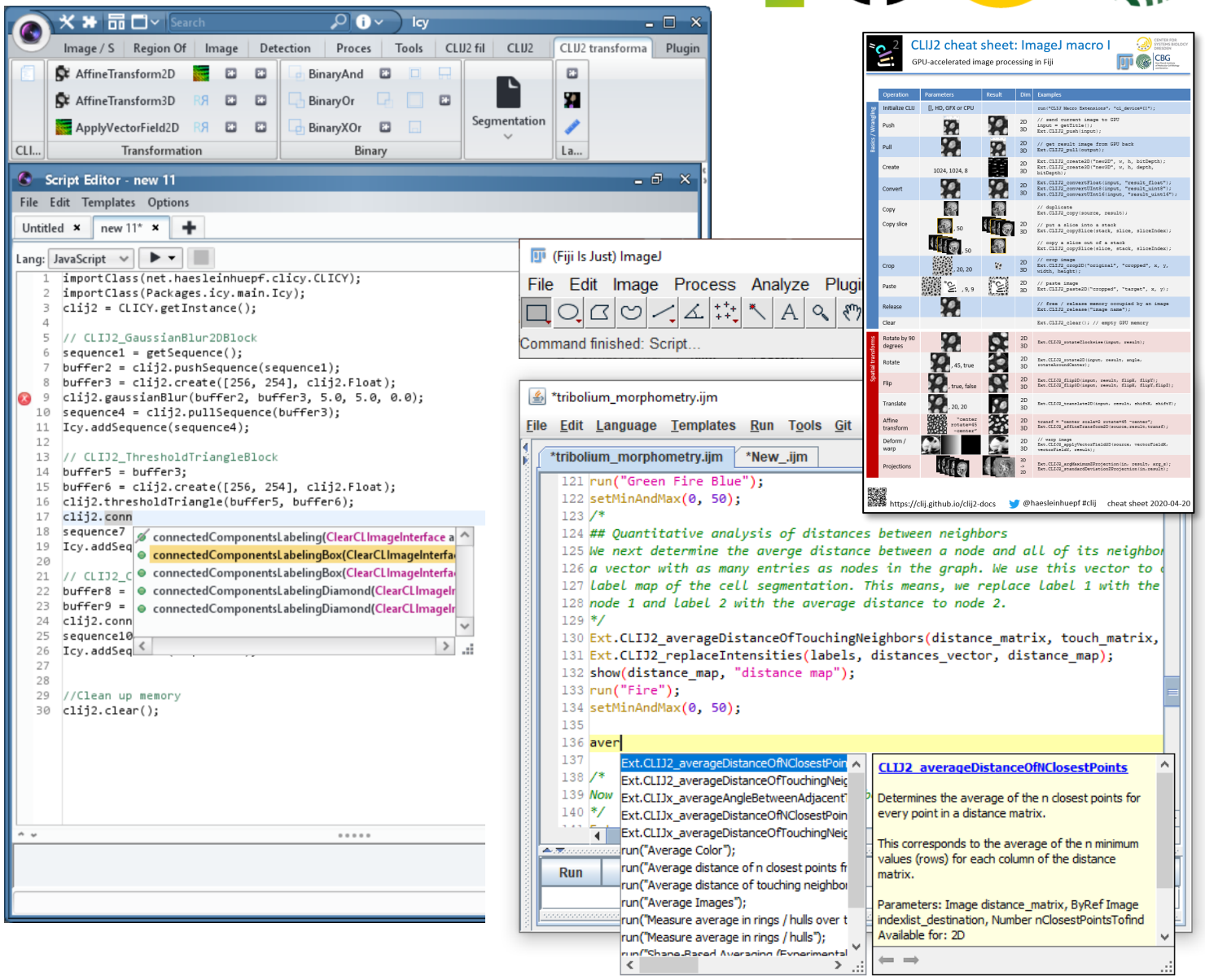
This tutorial

Please introduce yourself before asking a question!



Part I: Programming workflows

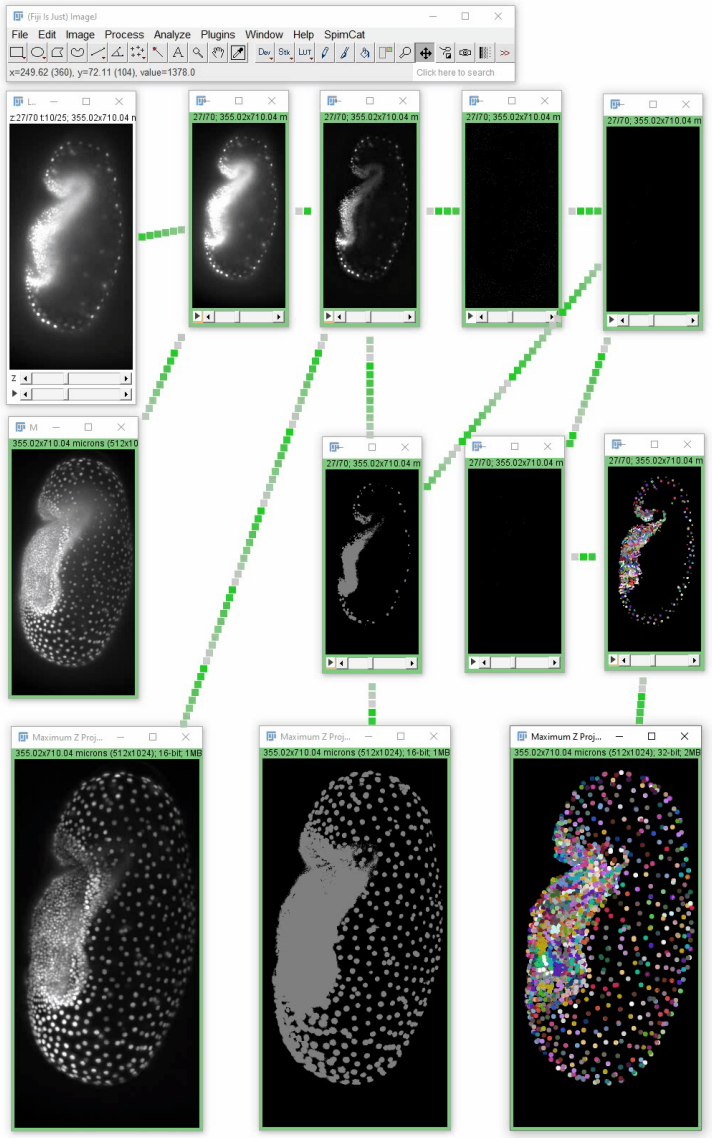
- Basics
  - Push / pull: Data transfer
  - Image processing operations
  - Memory management
- Code recording
  - Icy Bioimaging
  - ImageJ/Fiji
- Auto-completion
- Languages
  - ImageJ Macro
  - Fiji Jython
  - Matlab
  - Icy JavaScript
- Exercises

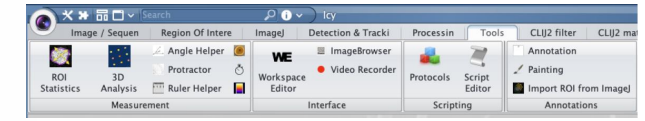


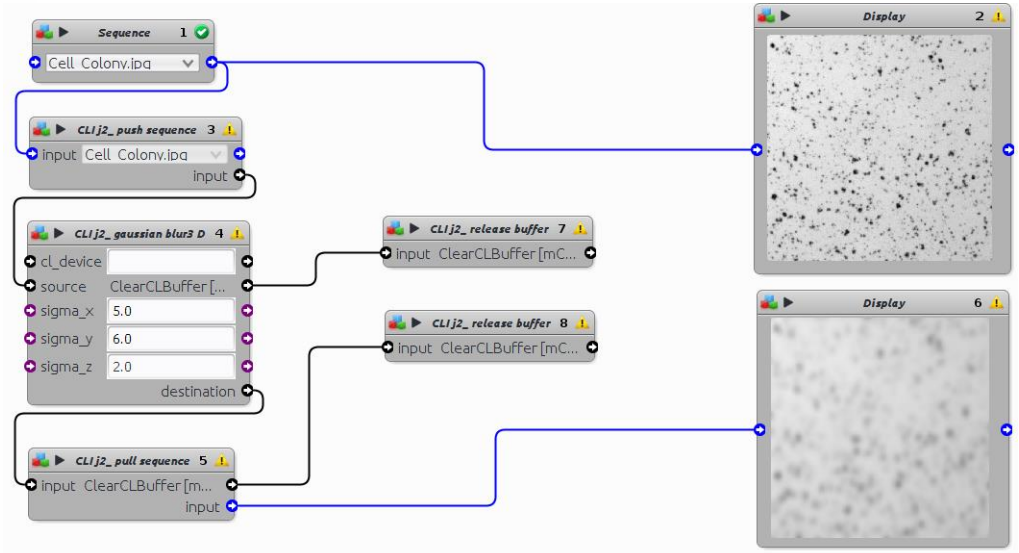
Part II: Interactive workflow design

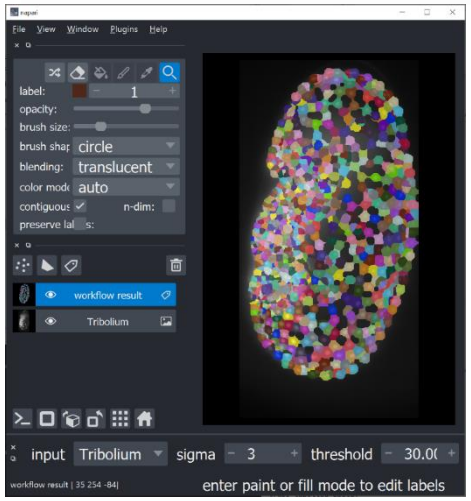
- Designing Image data flow graphs
- Expert system
- Code-generation
  - ImageJ Macro
  - Python
  - C++
  - QuPath Groovy
  - Matlab
  - Java/Maven
- Exercises

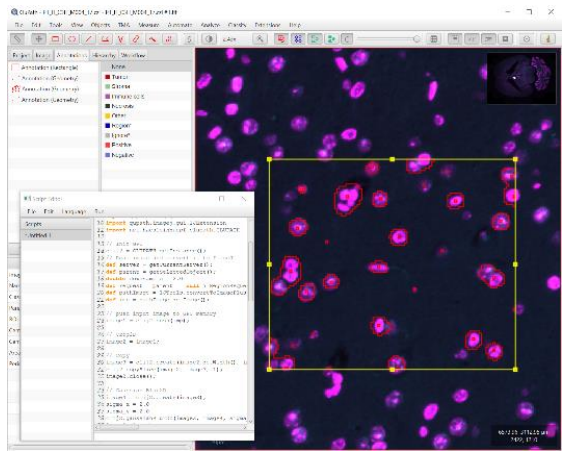
- Work on your own data











PoL Physics of Life TU Dresden
INSTITUT PASTEUR
CENTER FOR SYSTEMS BIOLOGY DRESDEN
CBG Max Planck Institute of Molecular Cell Biology and Genetics

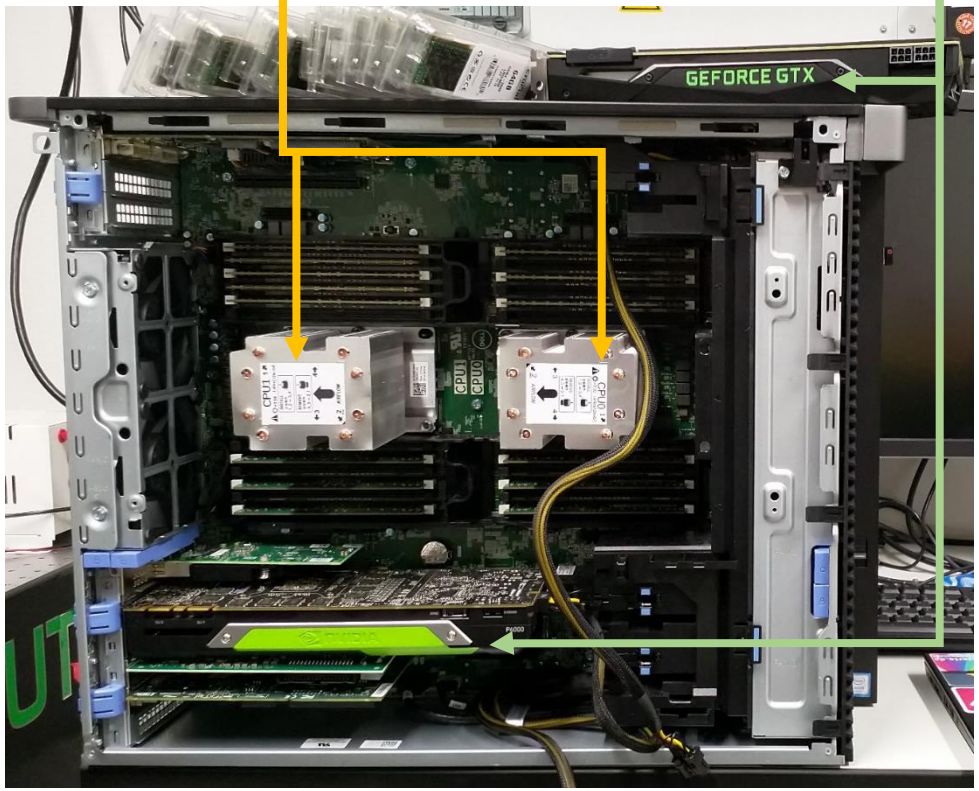
Introduction: [Image] processing on Graphics Processing Units

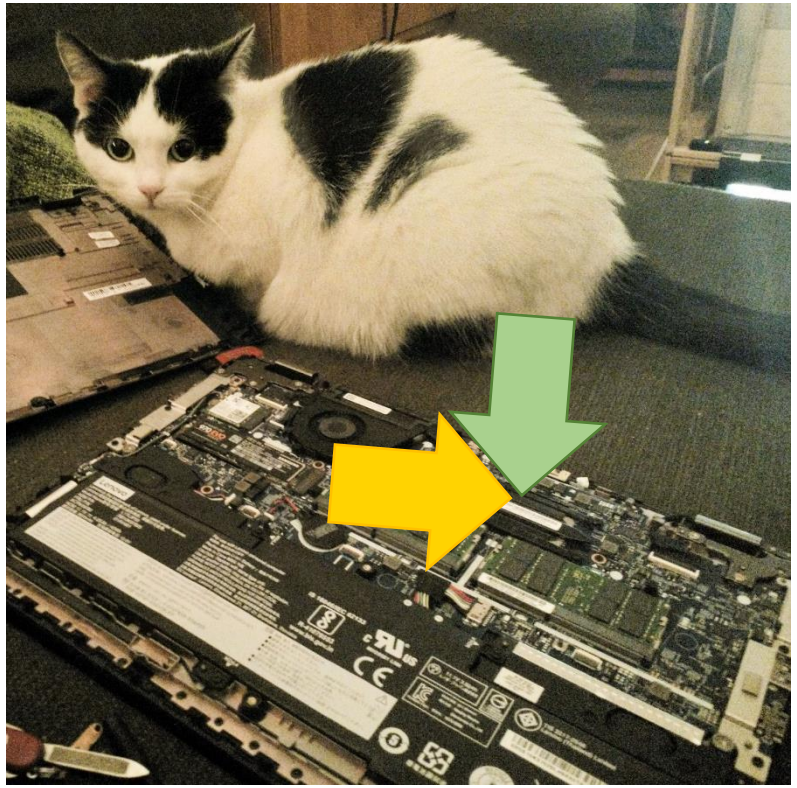
GPU-accelerated image processing

- Typical computers contain Graphics Processing Units

Central Processing Unit (CPU)

Graphics Processing Unit (GPU)



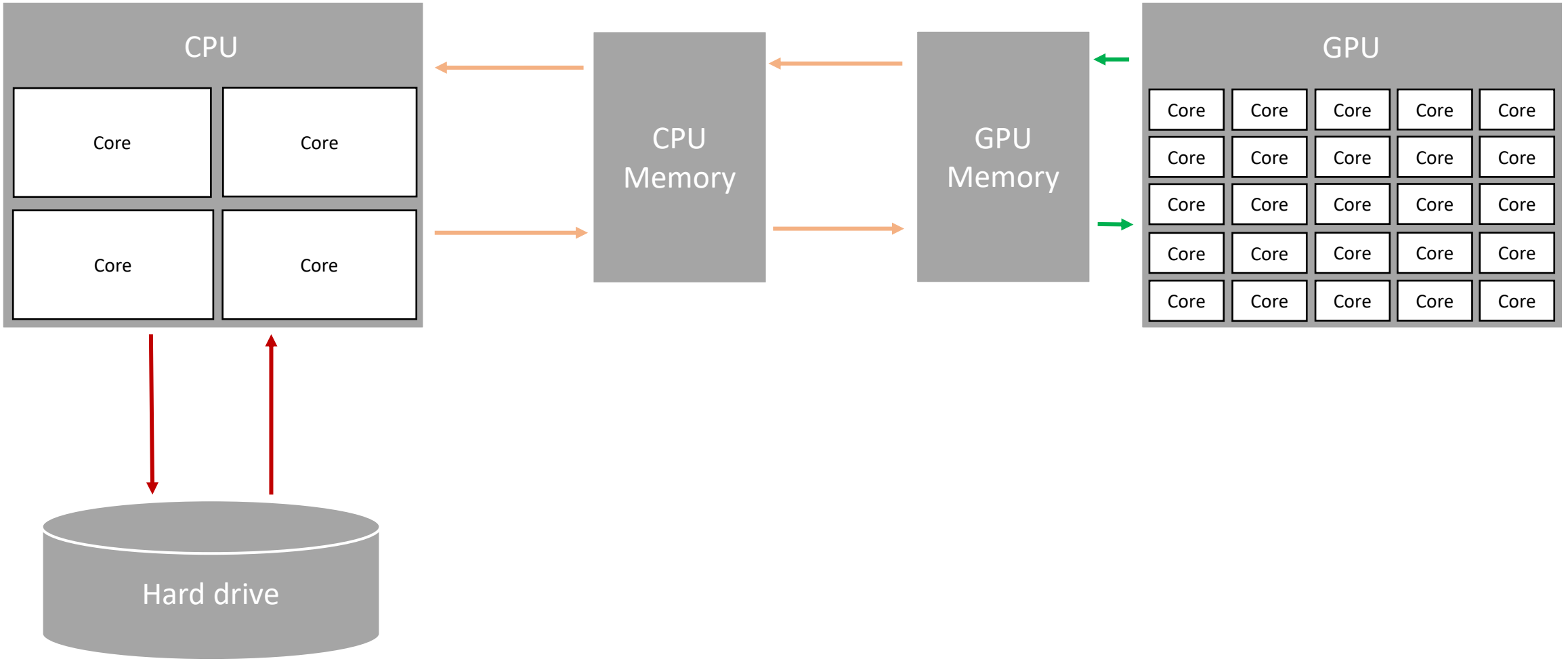


Most laptops contain *integrated* GPUs

Alternative: *external* GPUs

GPUs allow real-time image processing

- GPUs are specialised in processing, super fast thanks to many cores and fast memory access



Checklist

When does GPU-accelerated image processing make sense?

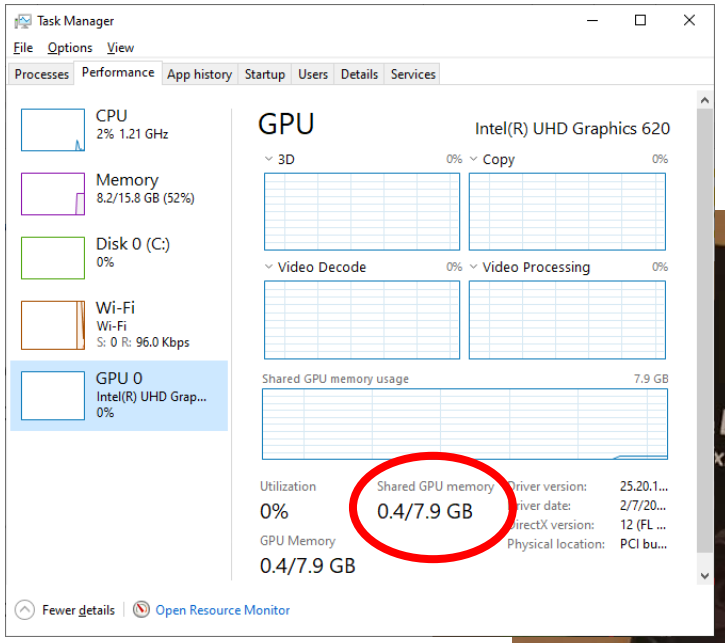
In order to accelerate your image analysis workflow

- you need an image analysis workflow in the first place,
- it should be slow; ideally:
  - (image processing time / image loading time) > 10 to 100,
- a single processing step (rule of thumb: image size in GB x4) should fit in your graphics card memory,
- you need to re-write your workflow at least partly and
- you should have experience with ImageJ macro.

If you really want to get the most out of it, you should

- own a graphics card with GDDR6 memory (memory bandwidth > 400 Gb/s).

Disclosure: I don't receive any money or anything from any GPU vendor.



PoL Physics of Life TU Dresden
INSTITUT PASTEUR
CENTER FOR SYSTEMS BIOLOGY DRESDEN
CBG Max Planck Institute of Molecular Cell Biology and Genetics

What can we gain from GPU-accelerated image processing?

GPU-accelerated image processing

- … depends on operation, image size, parameters, hardware, ….

vs.

Workstation CPU

2x Intel Xeon Silver 4110

Workstation GPU

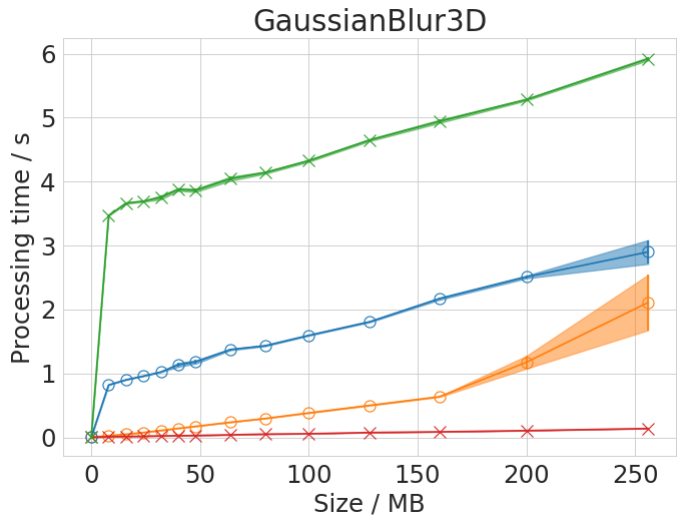
Nvidia Quadro P6000

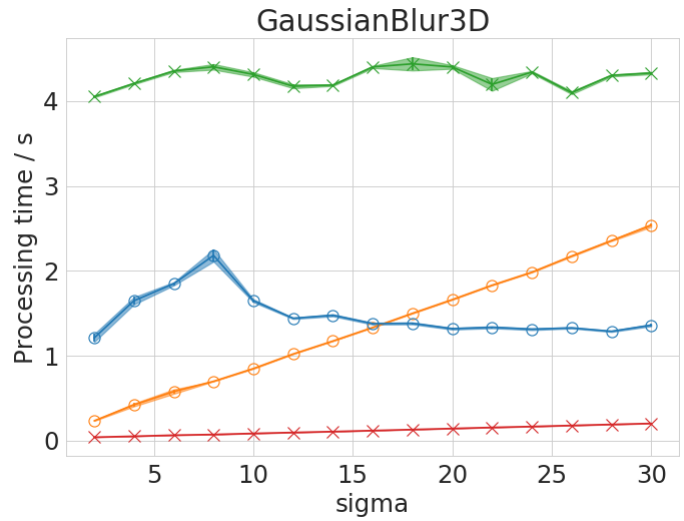
Laptop CPU

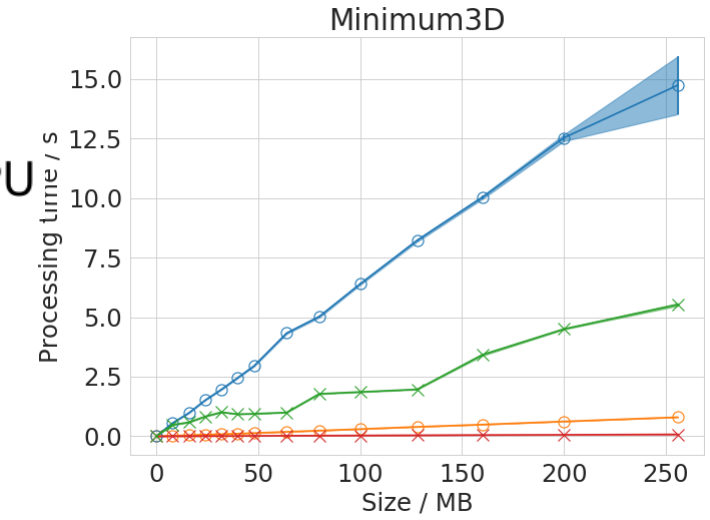
Intel Core i7-8650U

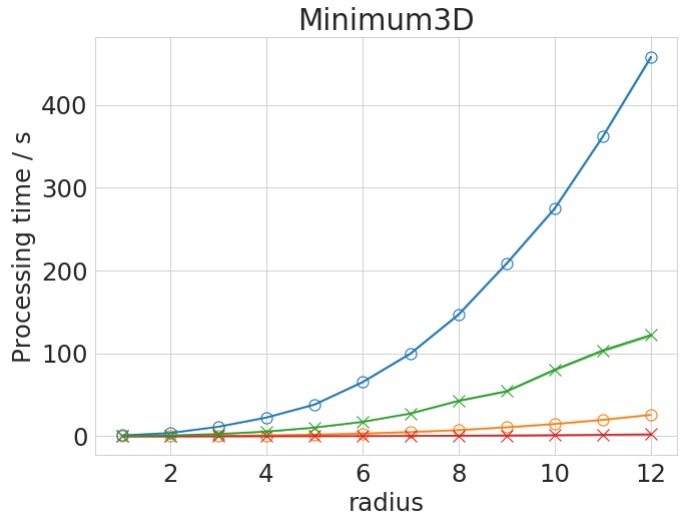
Laptop GPU

Intel UHD 620 GPU



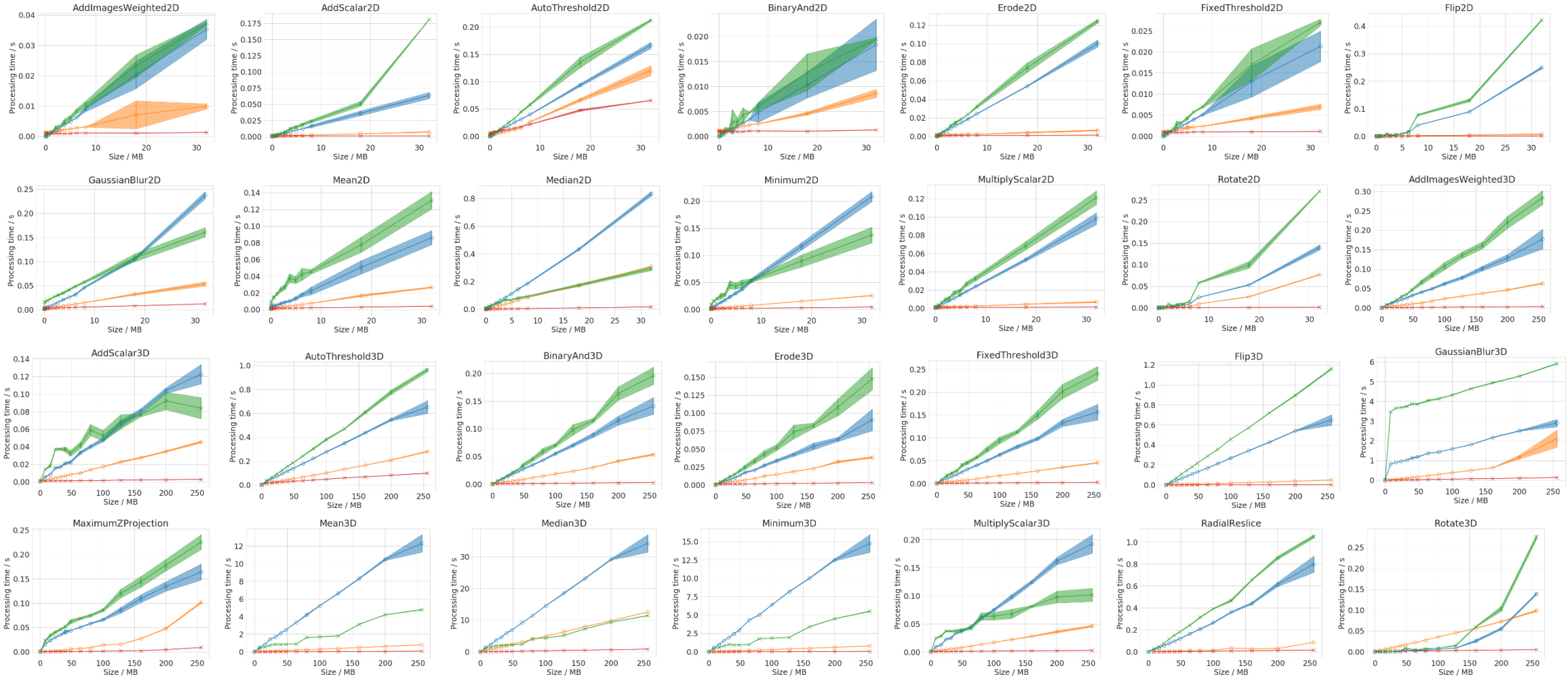






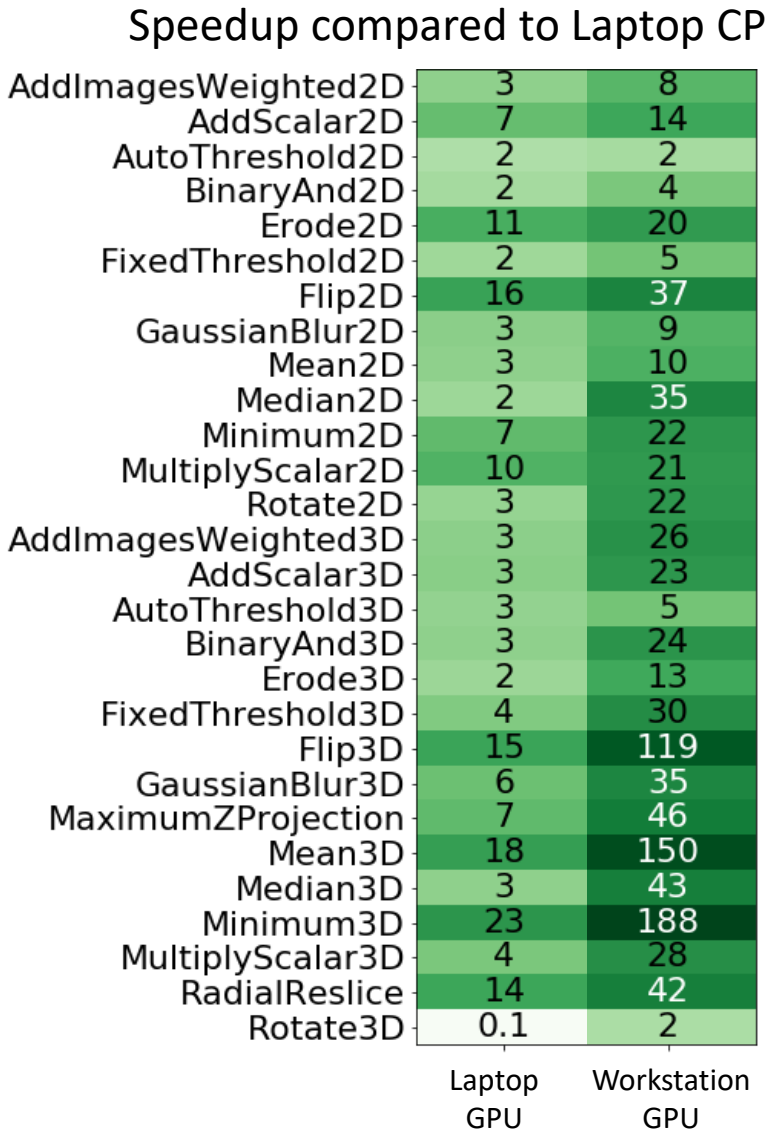
GPU-accelerated image processing

- … depends on operation, image size, parameters, hardware, ….



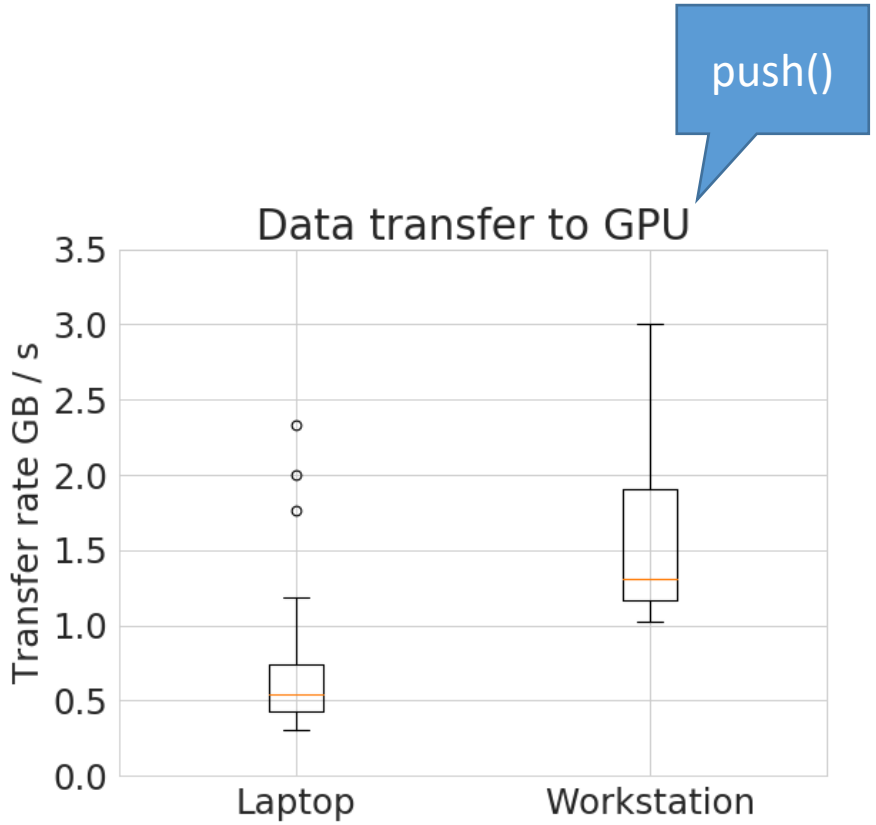
GPU-accelerated image processing

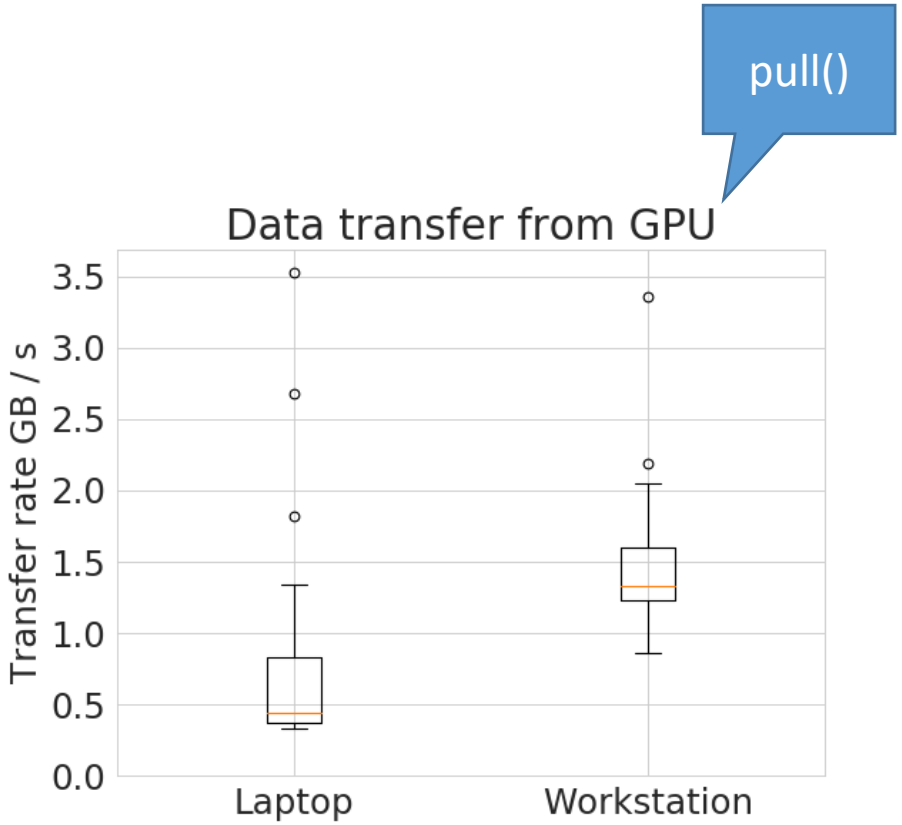
- 8 MB (2D)
- 64 MB (3D)



Data transfer takes time

- BUT: Push() and Pull() take additional time!

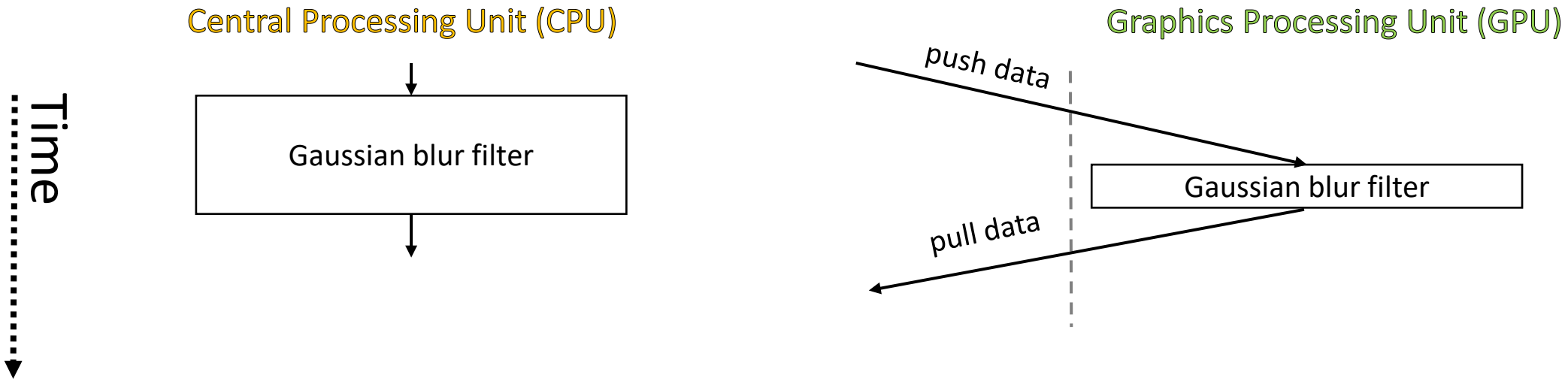




- Rule of thumb: One gigabyte takes one second.

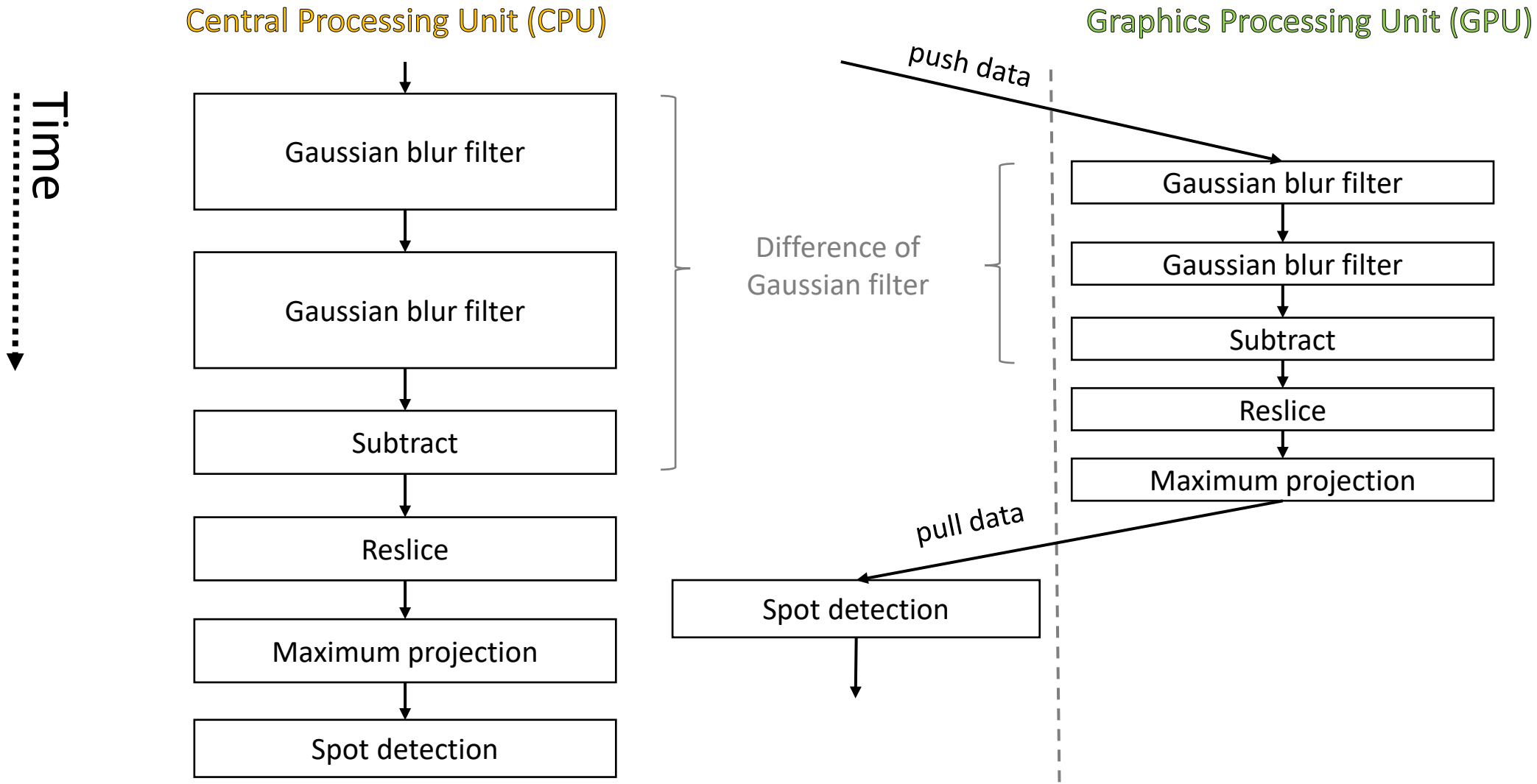
Build workflows consisting of many operations

- GPU acceleration may suffer from data transfer between CPU and GPU



Build workflows consisting of many operations

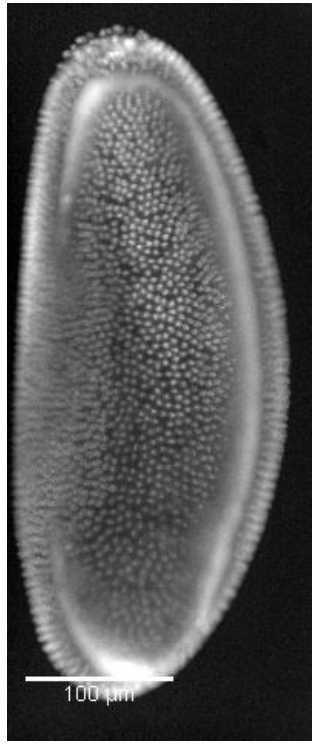
- GPU acceleration may suffer from data transfer between CPU and GPU

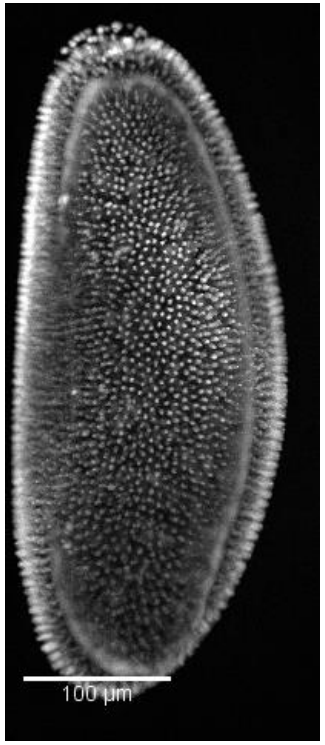


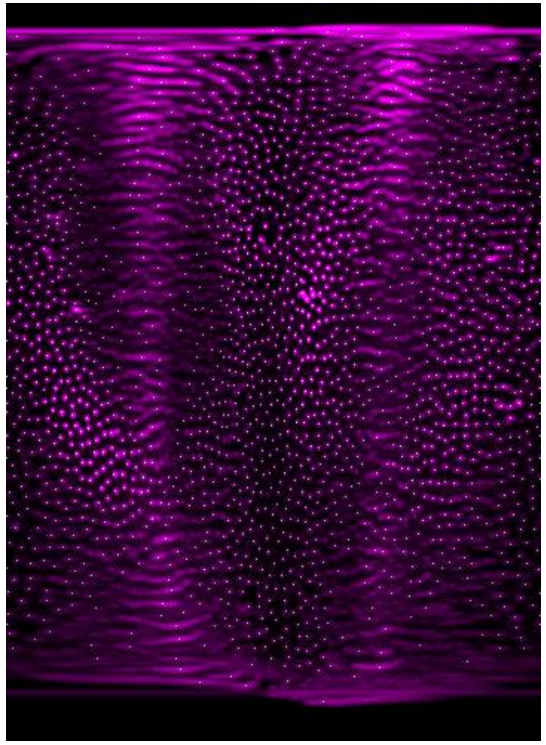
Optimal performance through smart memory management

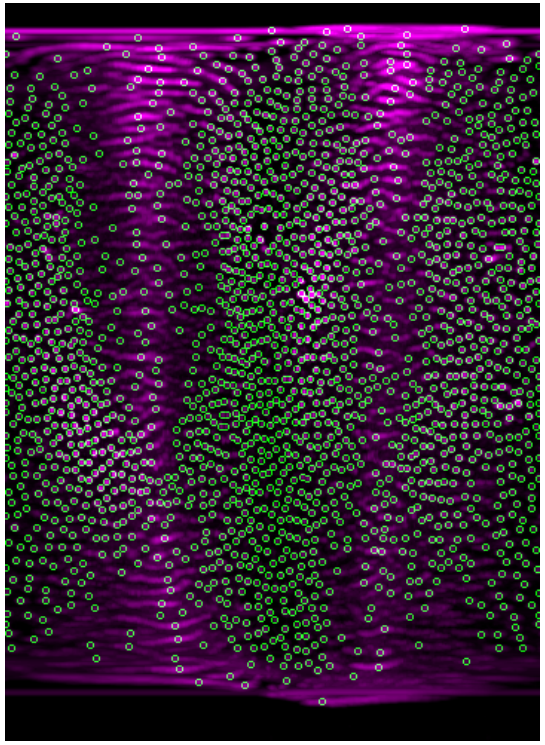
- Example workflow

Load data → Preprocessing → Transformation → Segmentation → Save data









Haase et al Nat Methods (2020)

Optimal performance through smart memory management

- The classical way of dealing with large image stacks…

Load data → Preprocessing

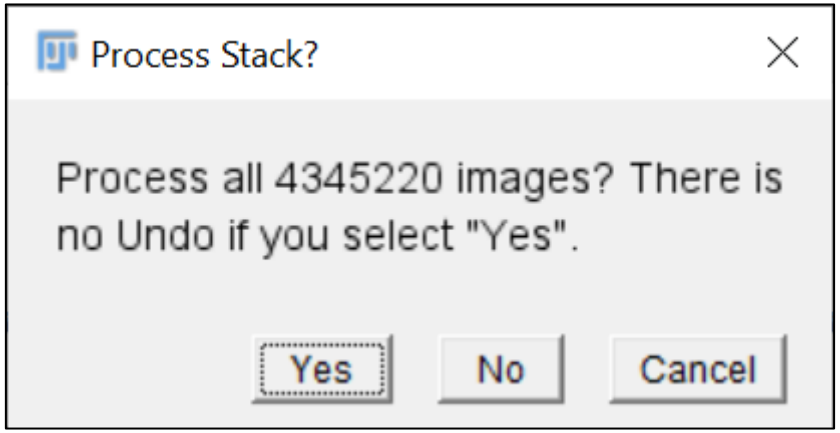
Load data

Load data

Load data

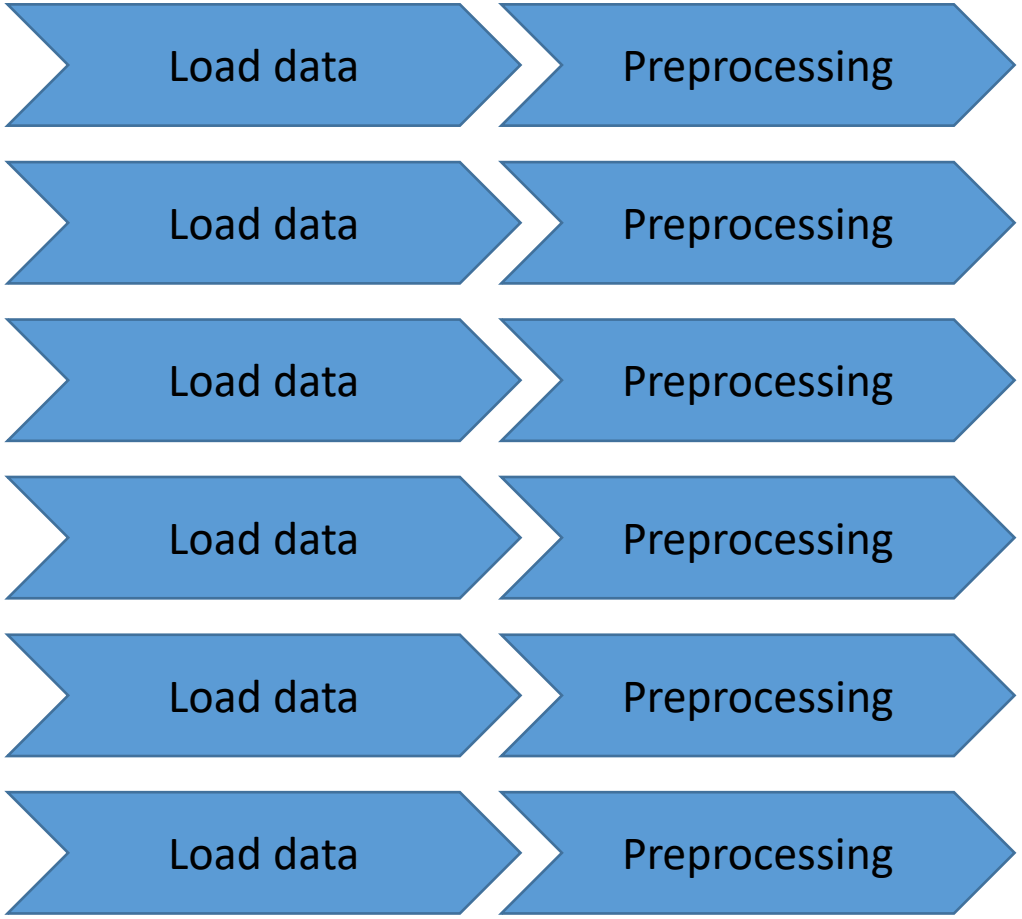
Load data

Load data



Optimal performance through smart memory management

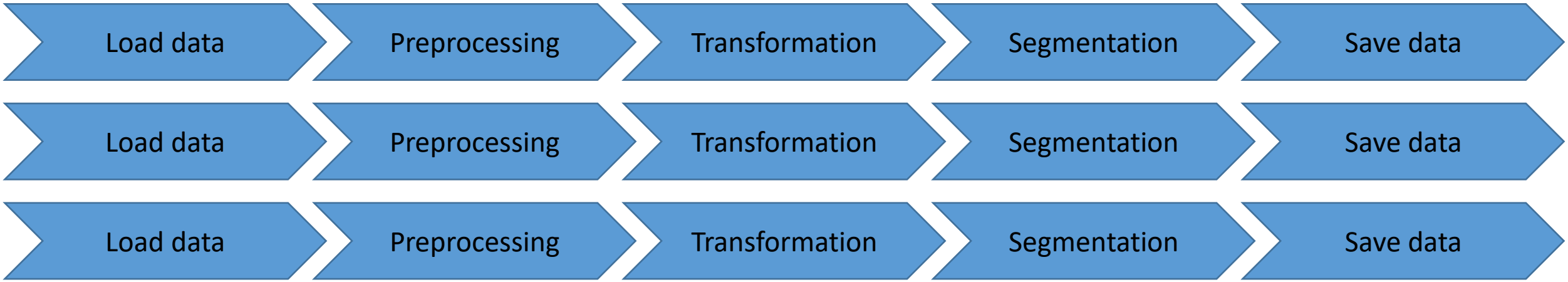
- The classical way of dealing with large image stacks… is suboptimal for memory consumption and performance.



This strategy does not just take long; it also costs a lot of memory!

Optimal performance through smart memory management

- Processing time-point by time-point is more efficient!

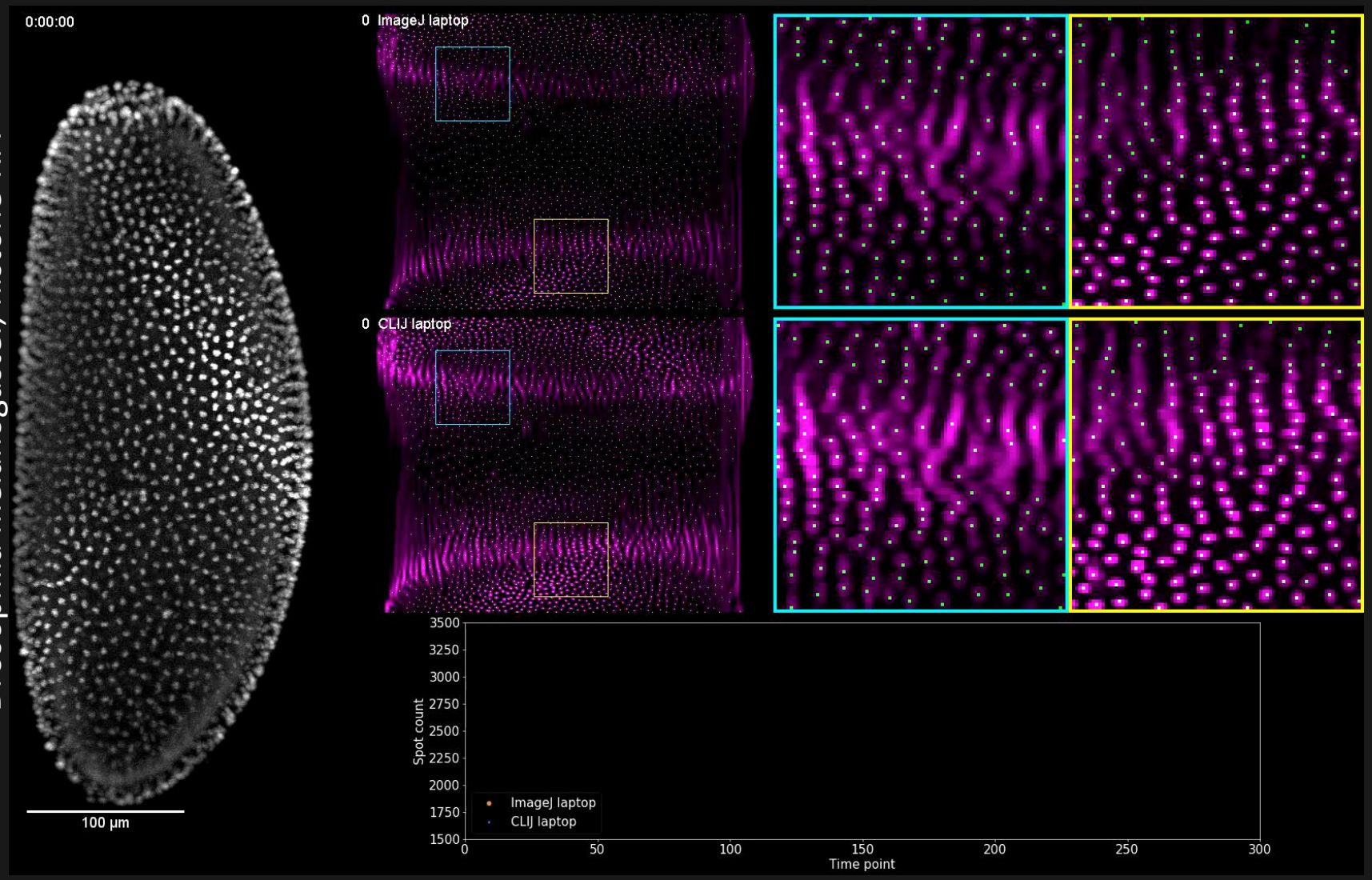


This strategy also works tile-by-tile on large 3D stacks!

Reuse memory!
See exercise 11

Optimal performance through smart memory management

- Counting spots in 300 frames of light sheet data takes:



ImageJ on CPU (laptop)

2:44 h
(31 seconds per frame)

ImageJ using the GPU (laptop)

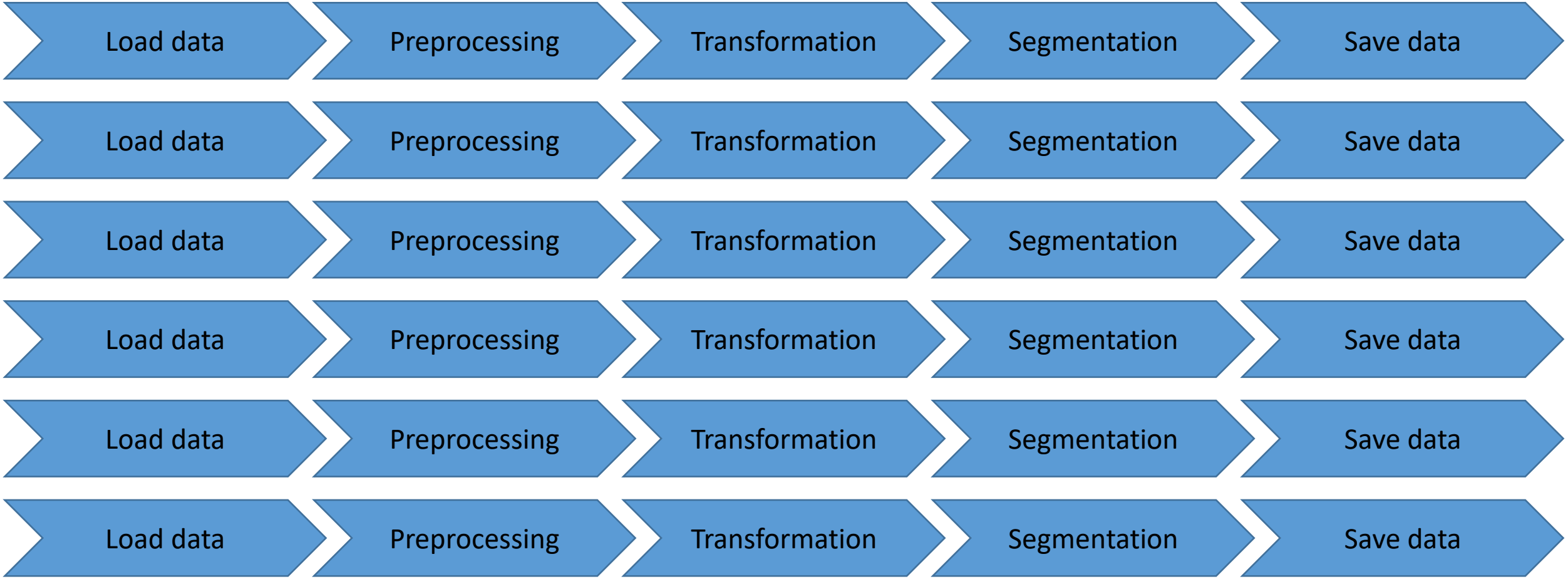
11 min
(1.2 seconds per frame)

ImageJ using a dedicated Nvidia GPU (workstation)

5 min
(0.4 seconds per frame)

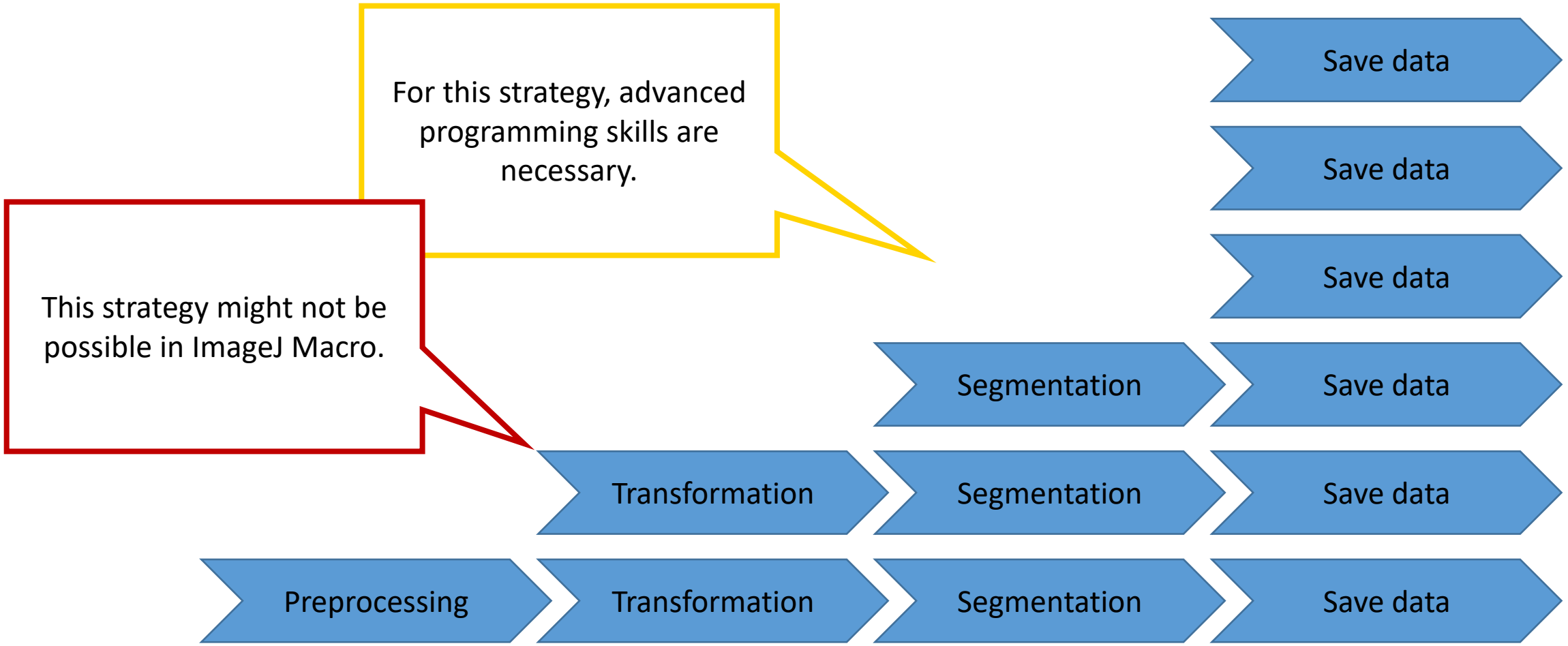
Optimal performance through smart memory management

- Even better: Distribute tasks between parallelized computation systems



Optimal performance through smart memory management

- Even better: Distribute tasks between parallelized computation systems

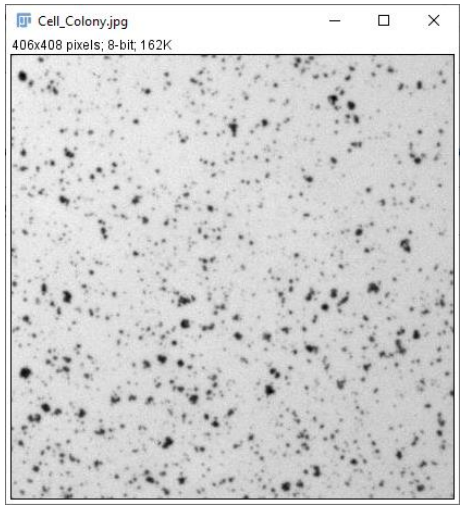


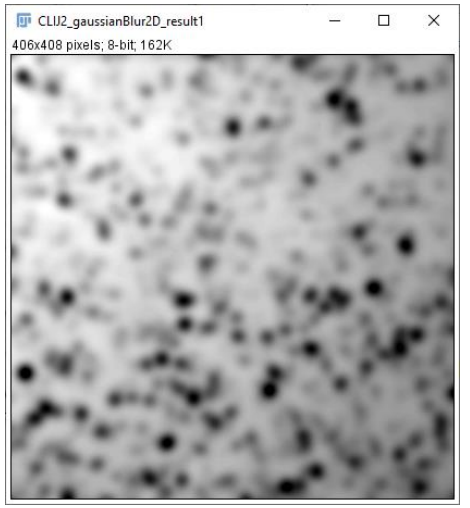
Scripting CLIJ2

CLIJ2: What every ImageJ Macro script must have

- Load data
- Initialize GPU
- Push
- Process images
- Pull
- Cleanup

```
// load data
run("Cell Colony (31K)");

// initialize GPU
run("CLIJ2 Macro Extensions", "cl_device=");
Ext.CLIJ2_clear();

// push image to GPU
input_image = getTitle();
Ext.CLIJ2_push(input_image);

// process image
sigma = 5;
Ext.CLIJ2_gaussianBlur2D(input_image, result_image, sigma, sigma);

// optional: release input data
Ext.CLIJ2_release(input_image);

// pull result back from GPU
Ext.CLIJ2_pull(result_image);

// clean up by the end
Ext.CLIJ2_clear();
```

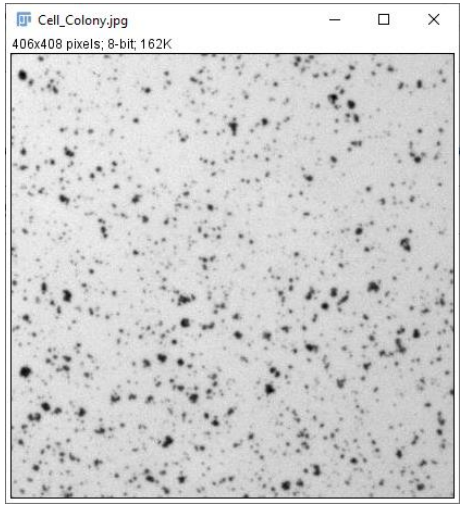


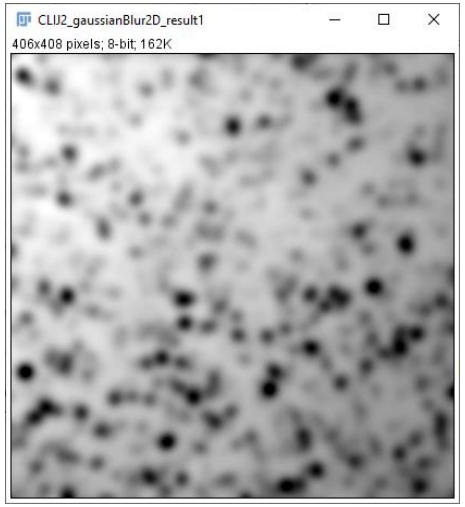


CLIJ2: What every ImageJ Jython script must have

- Load data
- Initialize GPU
- Push
- Process images
- Pull
- Cleanup

```
# Load data
from ij import IJ
imp = IJ.openImage("http://imagej.nih.gov/ij/images/Cell_Colony.jpg")
imp.show()

# initialize GPU
from net.haesleinhuepf.clij2 import CLIJ2;
clij2 = CLIJ2.getInstance()

# push image to GPU
input_image = clij2.push(imp)

# process image
sigma = 5
blurred = clij2.create(input_image)
clij2.gaussianBlur2D(input_image, blurred, sigma, sigma)

# optional: release input data
input_image.close()

# pull result back from GPU
imp = clij2.pull(blurred)
imp.show()

# clean up by the end
clij2.clear()
```

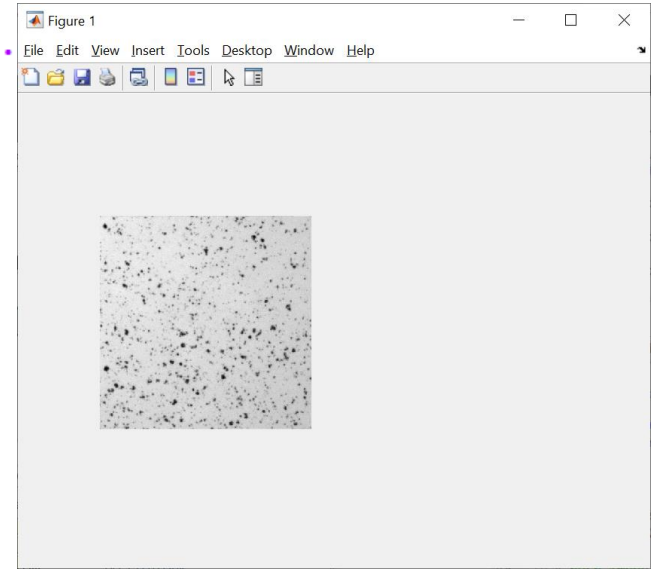


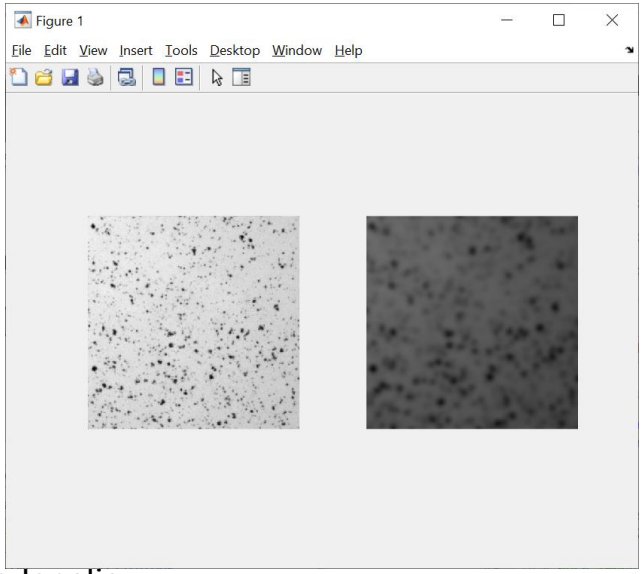


CLIJ2: What every Matlab script must have

- Load data
- Initialize GPU
- Push
- Process images
- Pull
- Cleanup

```
% load data
image = imread("http://imagej.nih.gov/ij/images/Cell_Colony.
subplot(1, 2, 1), imshow(image);

% initialize GPU
clij2 = init_clatlab();

% Push Crop_ResSin1CGFP_016-1.tif to GPU memory
input_image = clij2.pushMat(image);

% process image
sigma = 5;
blurred = clij2.create(input_image);
clij2.gaussianBlur2D(input_image, blurred, sigma, sigma)

% optional: release input data
input_image.close()

% pull result back from GPU
result = clij2.pullMat(blurred);
%subplot(1, 2, 2), imshow(result);

% clean up by the end
clij2.clear()
```





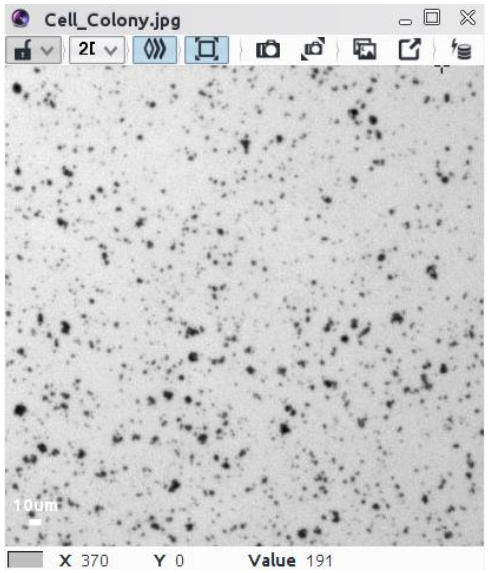
PoL Physics of Life TU Dresden | INSTITUT PASTEUR | CENTER FOR SYSTEMS BIOLOGY DRESDEN | CBG Max Planck Institute of Molecular Cell Biology and Genetics

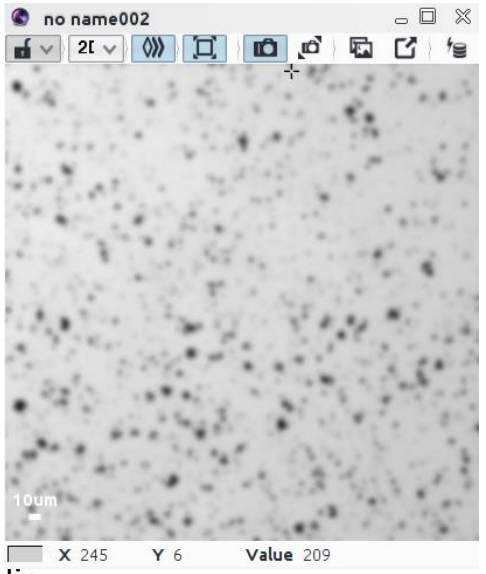
Icy Bioimaging

CLIJ2: What every ICY javascript must have

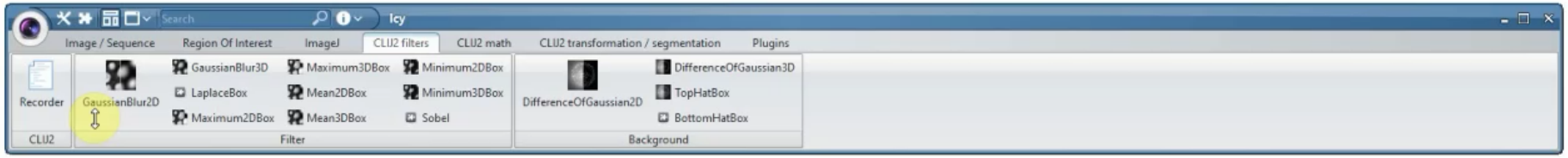
- Load data
- Initialize GPU
- Push
- Process images
- Pull
- Cleanup

```
1   // load data
2   importClass(Packages.icy.main.Icy);
3   importClass(Packages.icy.file.Loader);
4   seq = Loader.loadSequence("Cell_Colony.jpg", 0, true);
5
6   // initialise GPU
7   importClass(net.haesleinhuepf.clicy.CLICY);
8   clij2 = CLICY.getInstance();
9
10  // push data to GPU
11  sequence1 = getSequence();
12  buffer2 = clij2.pushSequence(sequence1);
13
14  // process data
15  buffer3 = clij2.create([406, 408], clij2.Float);
16  clij2.gaussianBlur2D(buffer2, buffer3, 2.0, 2.0);
17
18  // pull data from GPU
19  sequence4 = clij2.pullSequence(buffer3);
20  Icy.addSequence(sequence4);
21
22  // clean up memory
23  clij2.clear();
```





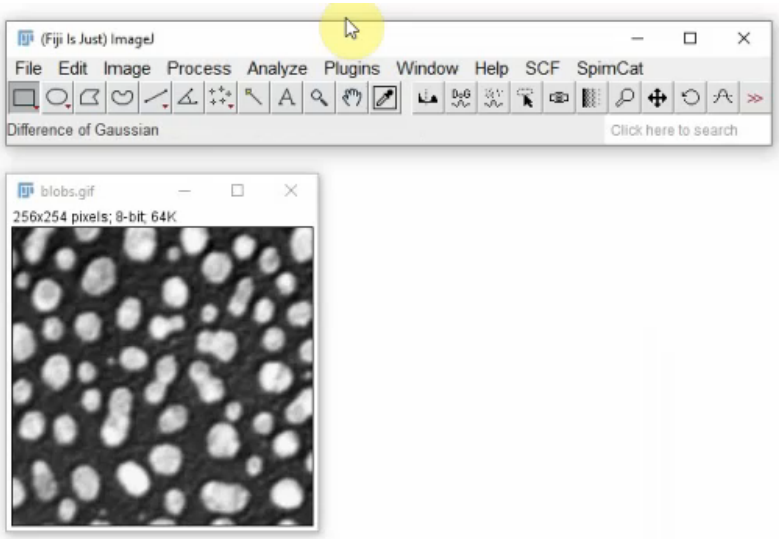
CLIJ2: use ICY javascript recorder!



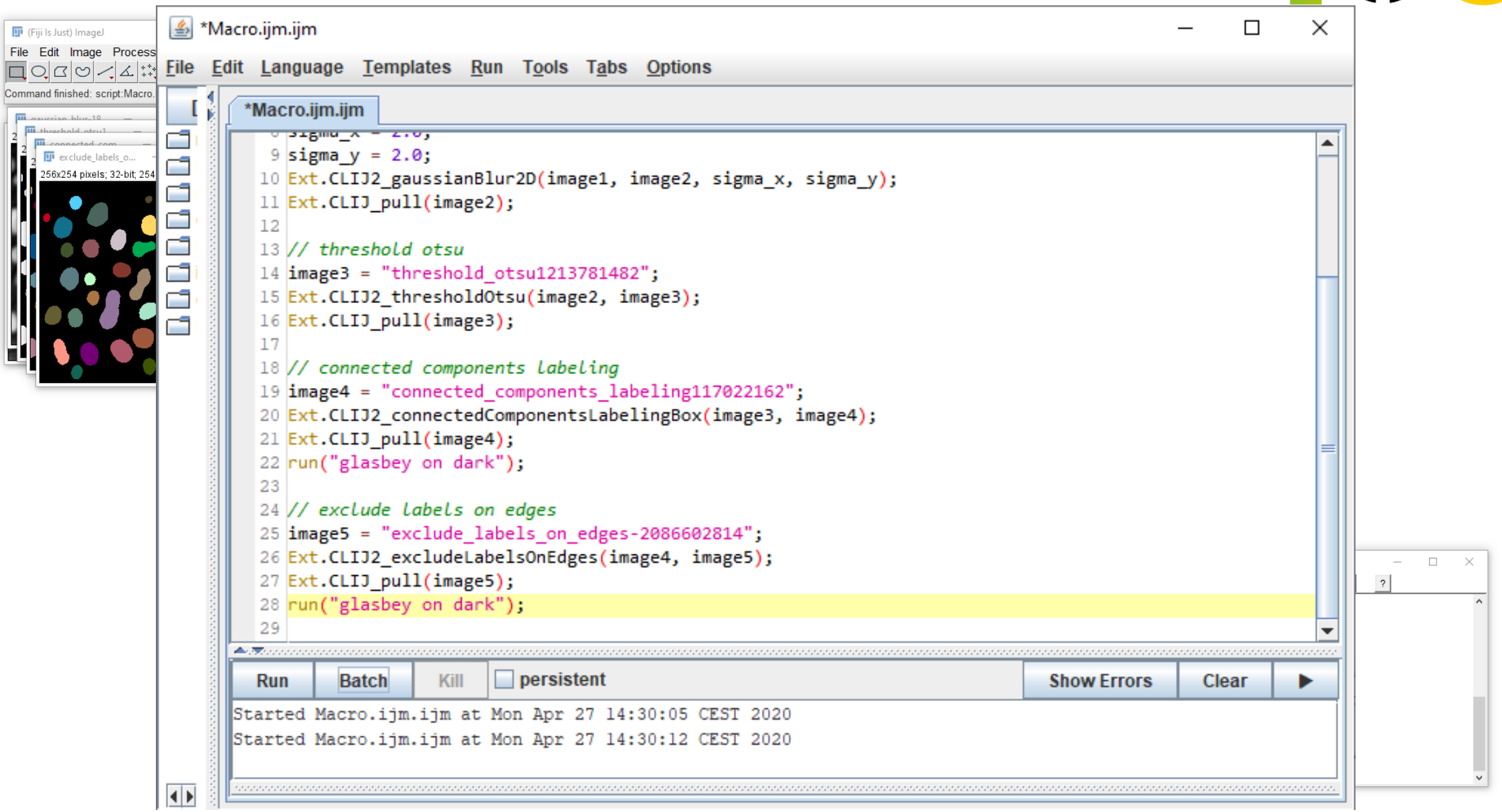
PoL Physics of Life TU Dresden
INSTITUT PASTEUR
CENTER FOR SYSTEMS BIOLOGY DRESDEN
CBG Max Planck Institute of Molecular Cell Biology and Genetics

ImageJ / Fiji

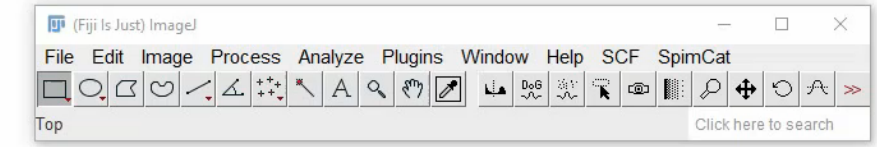
CLIJ2: Macro recording

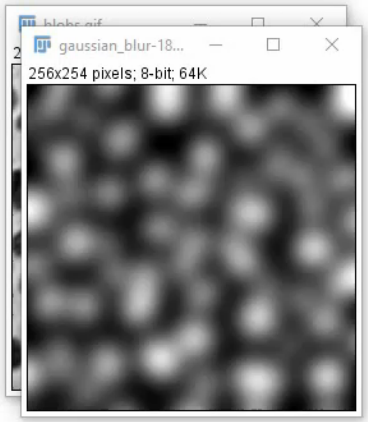


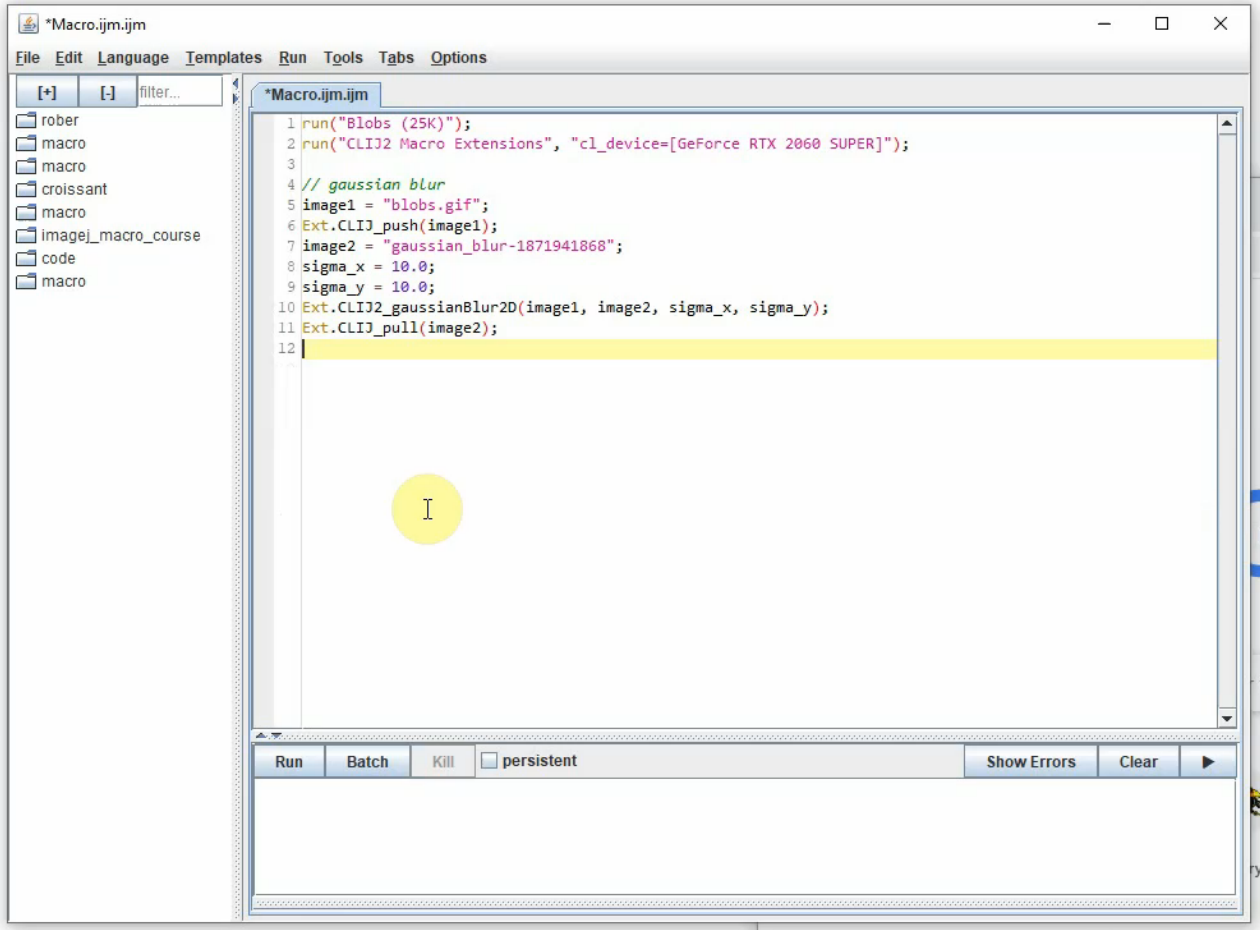
CLIJ2: Macro recording

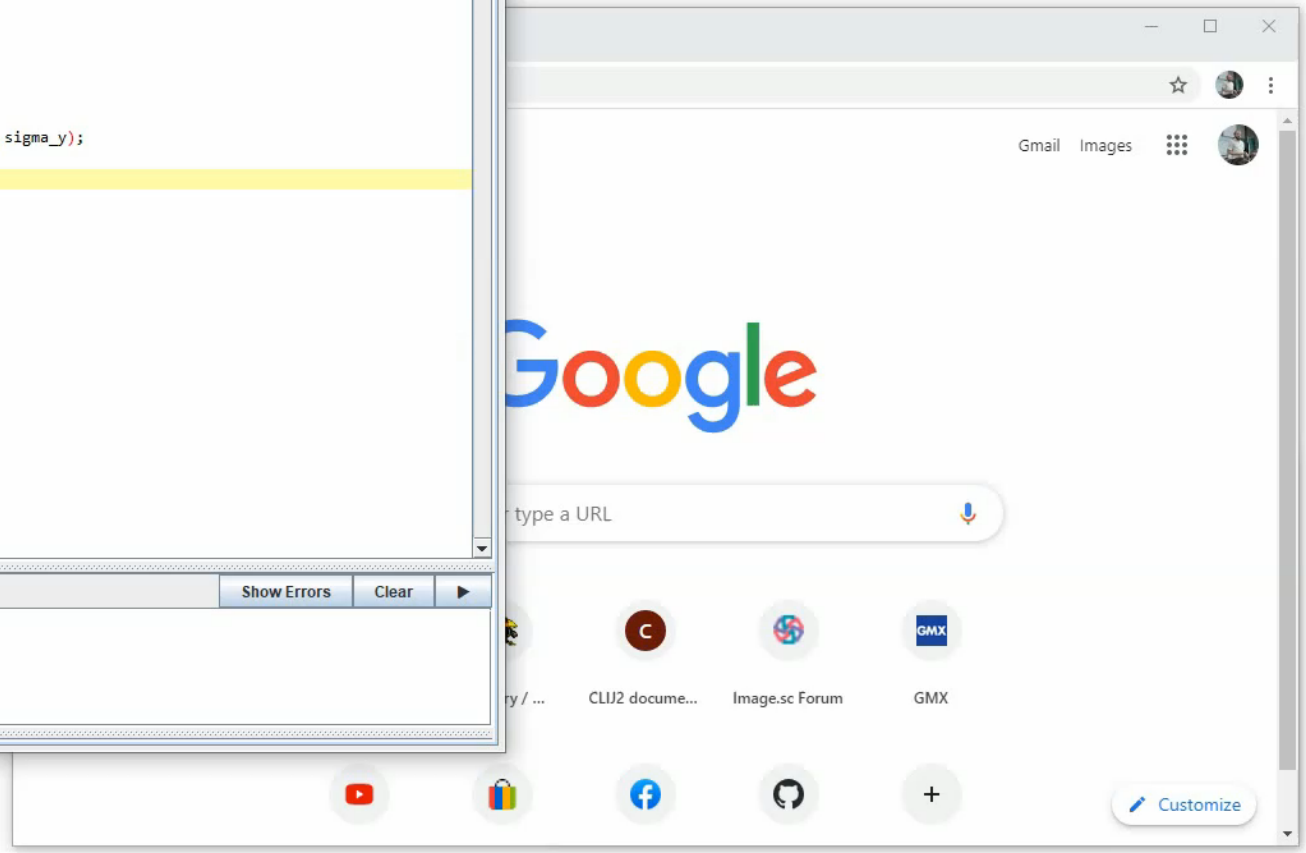


CLIJ2: Macro editing



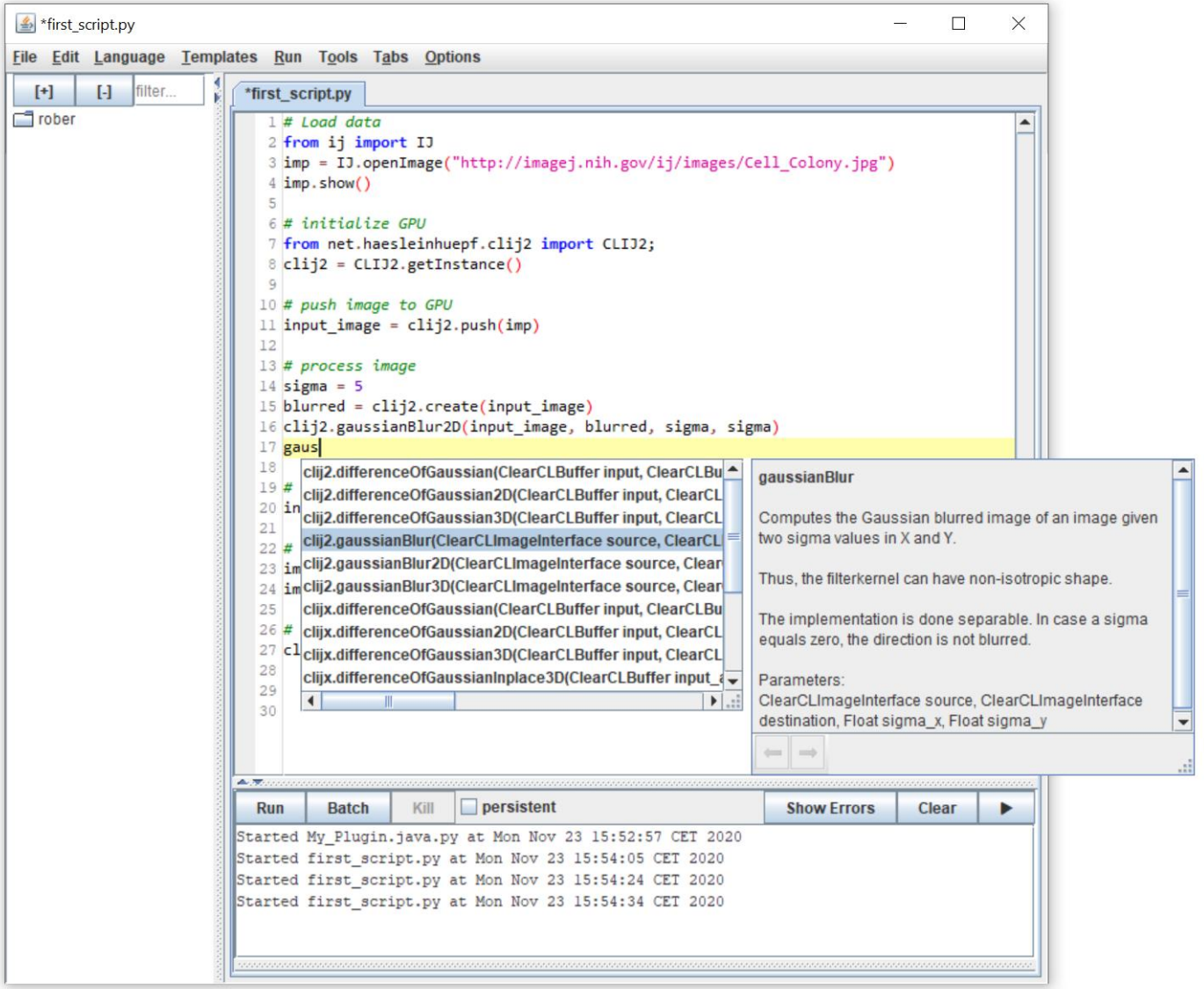






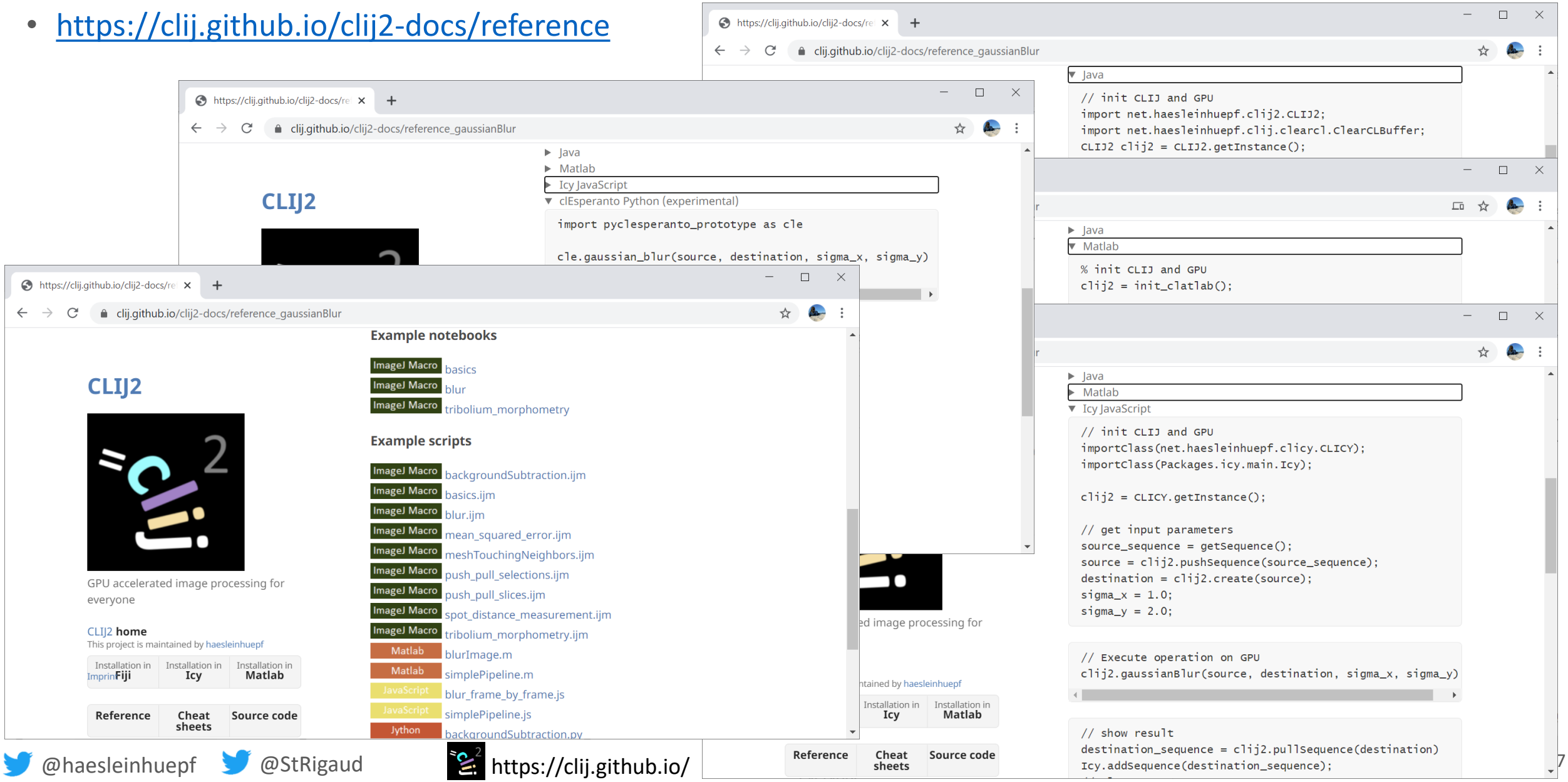
Macro editing

- Auto-completion also works for Jython & friends.
- Start typing what you search e.g. “gauss” and not “clij”.
- Make use of code snippets



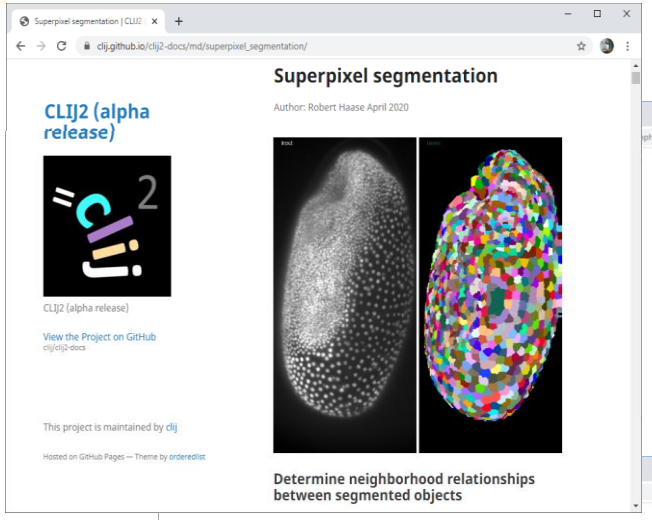
Programming language specific examples

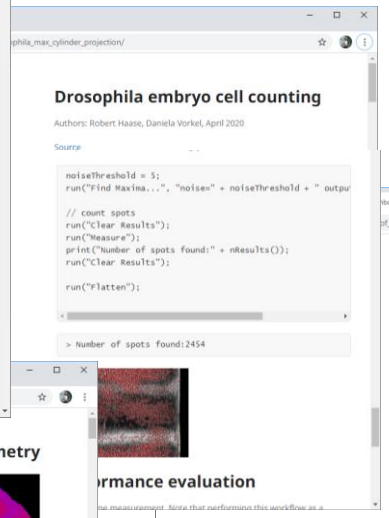
- https://clij.github.io/clij2-docs/reference

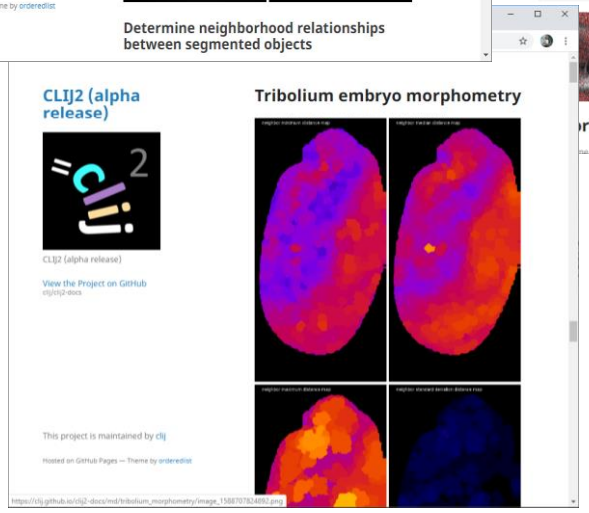


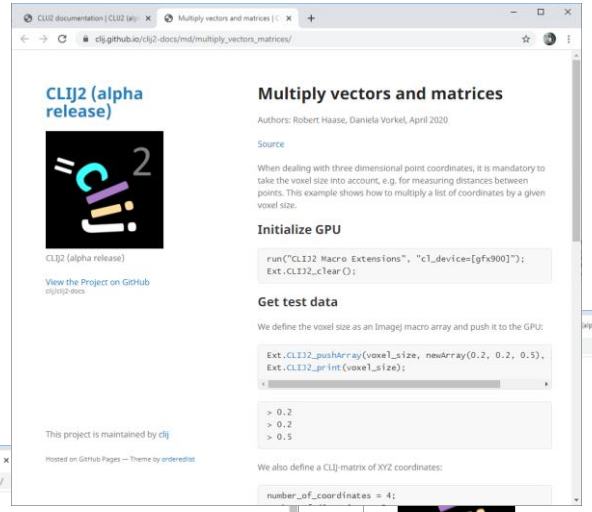
Check out the tutorials

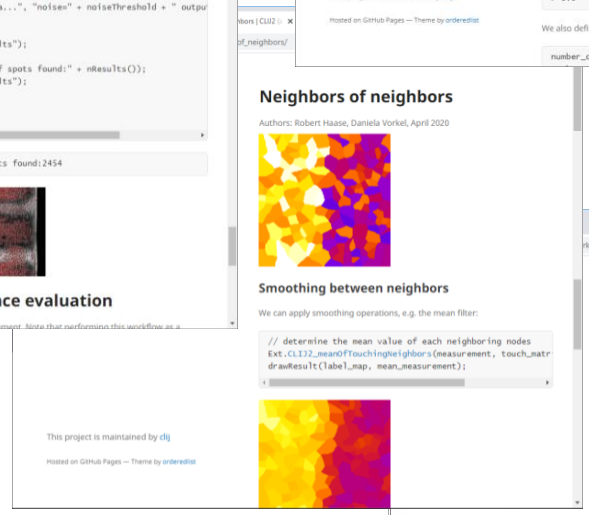
https://clij.github.io/

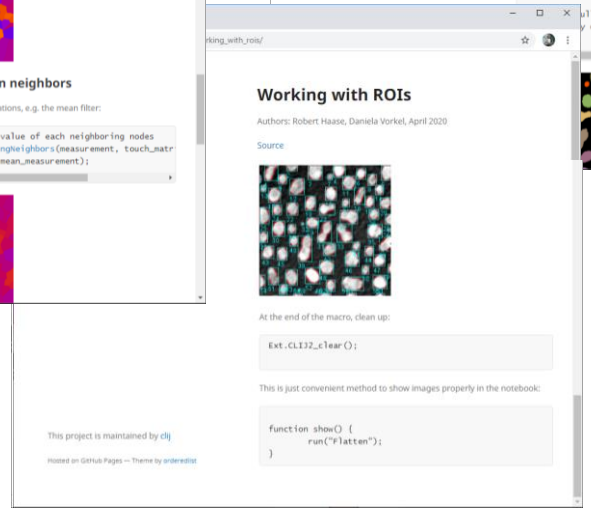


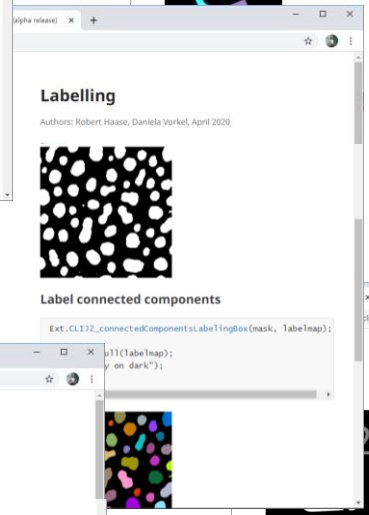


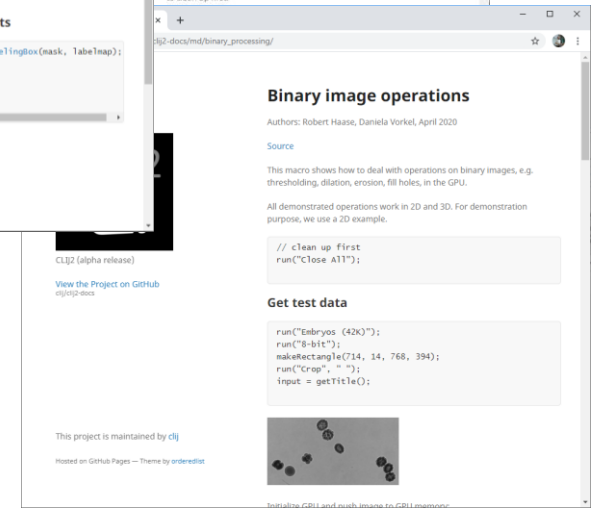










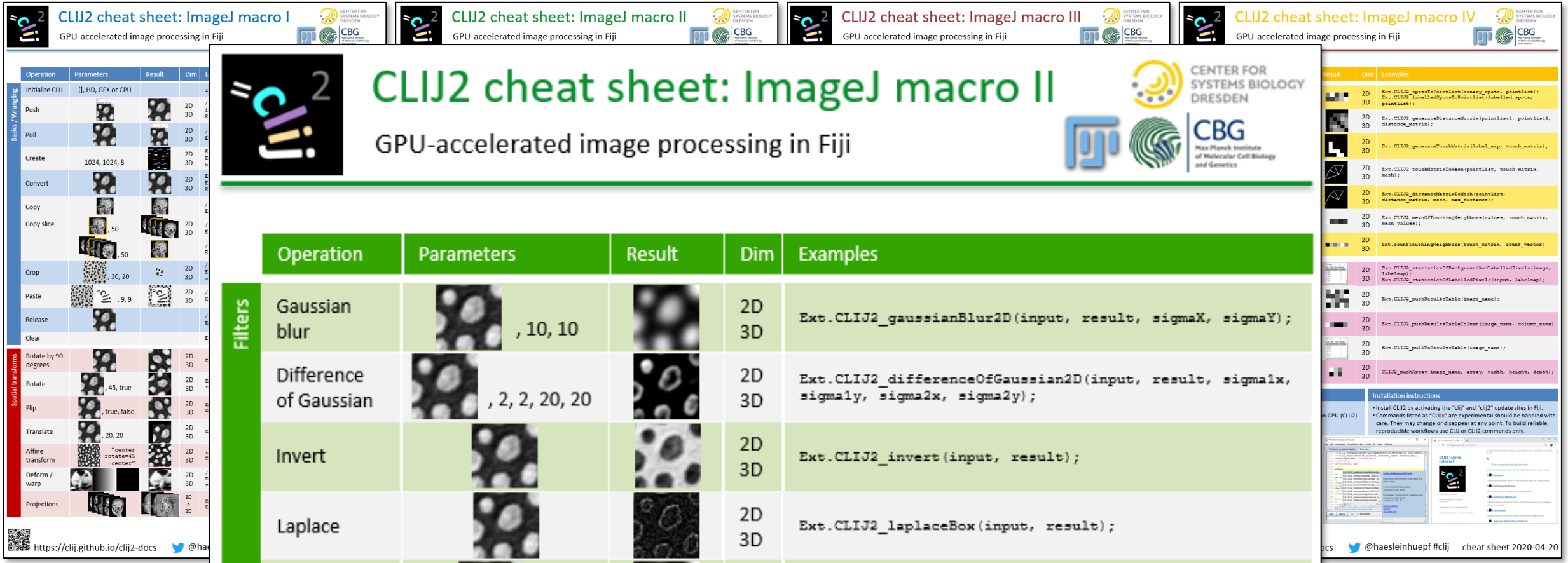


Big thanks to co-author and proof-reader

Dani Vorkel (Myers lab) @happifocus

Cheat sheets

- Cheat sheets show the most important methods with input and output parameters visually.

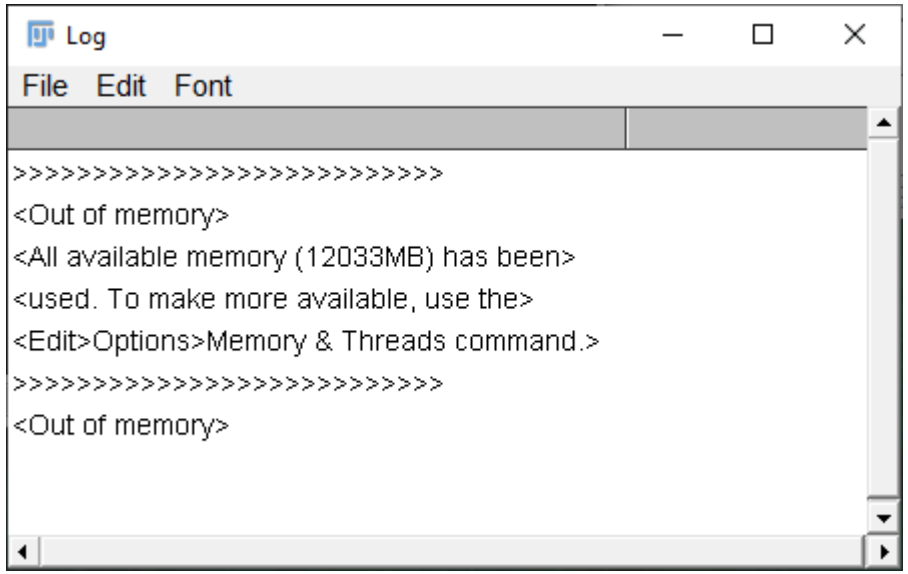


https://clij.github.io/clij2-docs/CLIJ2-cheatsheet_V3.pdf

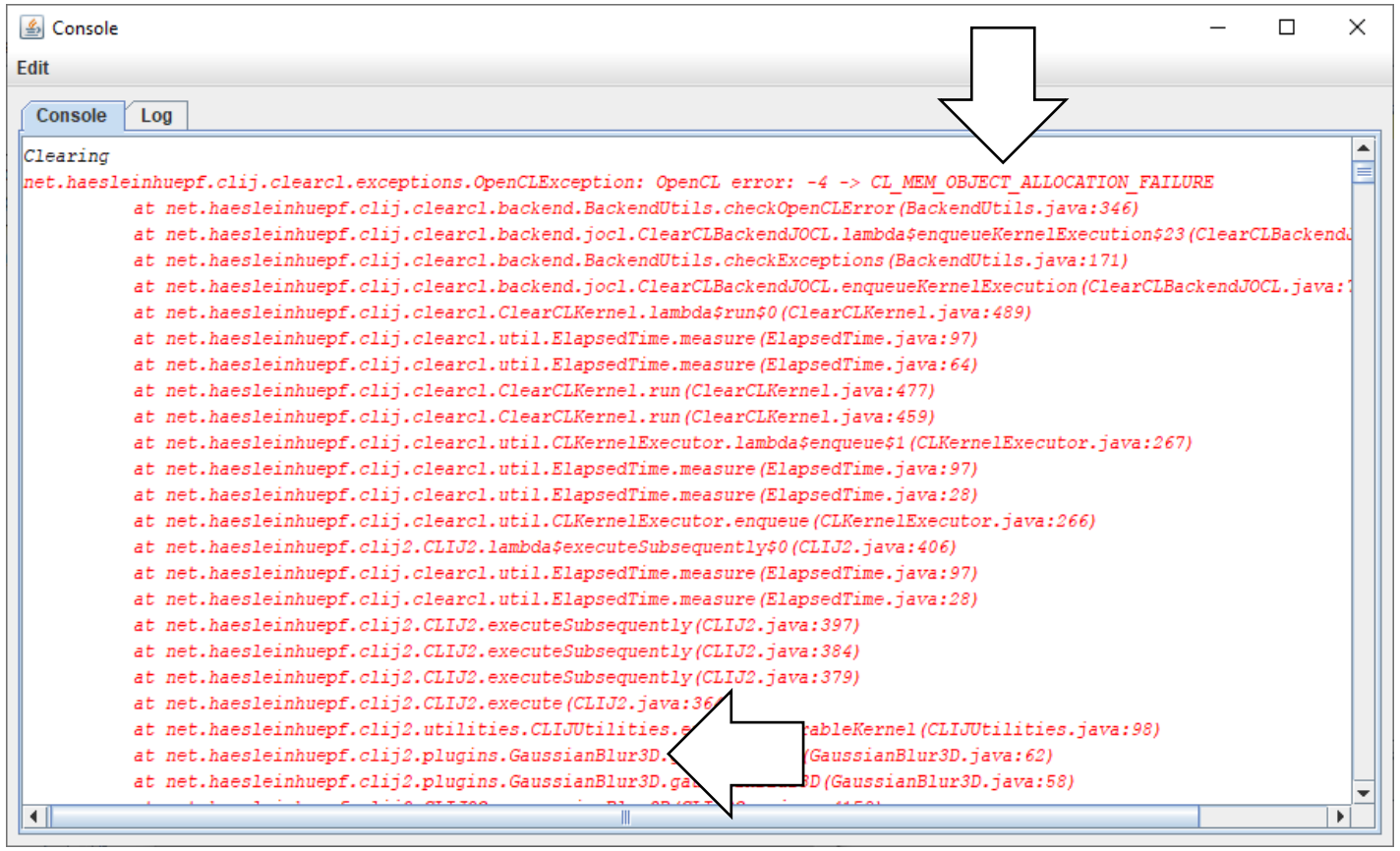
CLIJ2: Out of memory?

- Two options:
  - Buy more memory. This means, buy a more expensive graphics card.
  - Change your workflow (release memory! Downsample!)

2010:



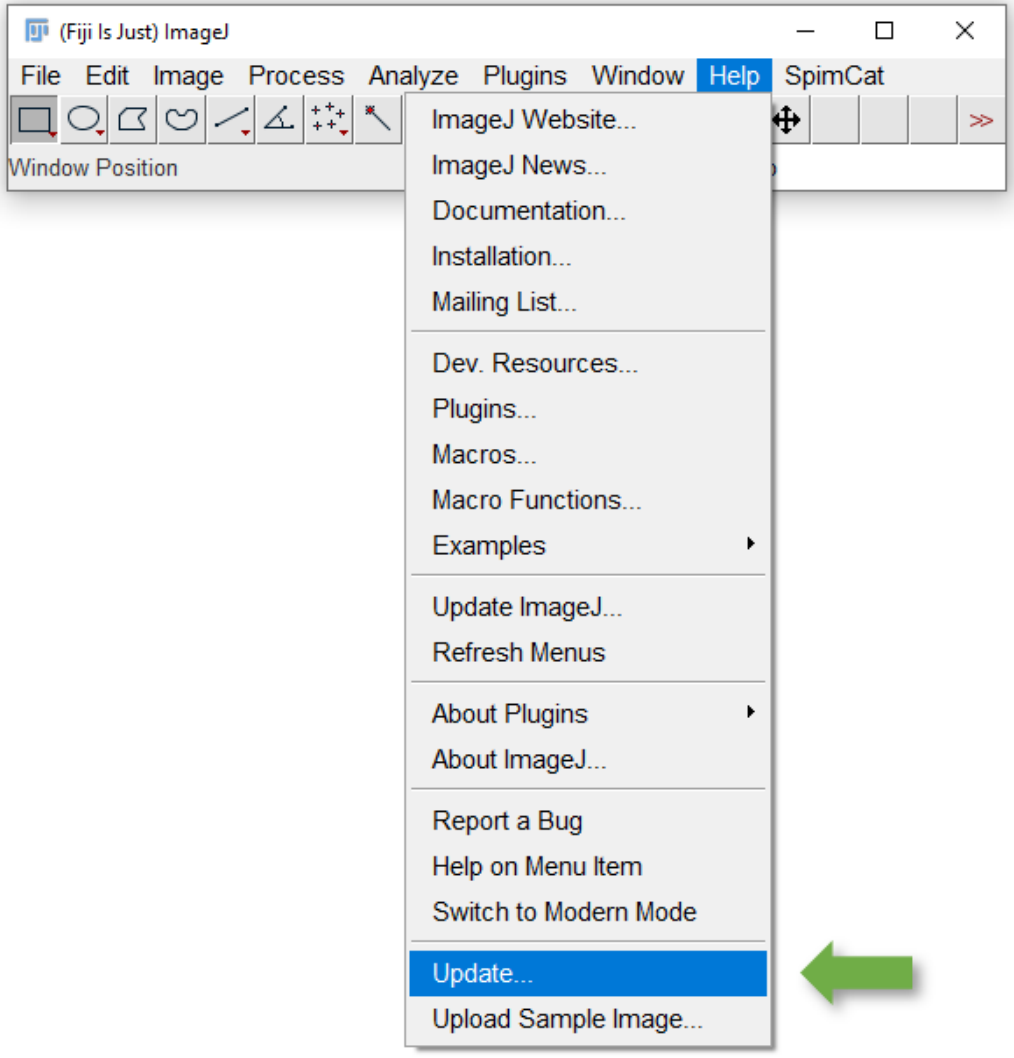
2020:

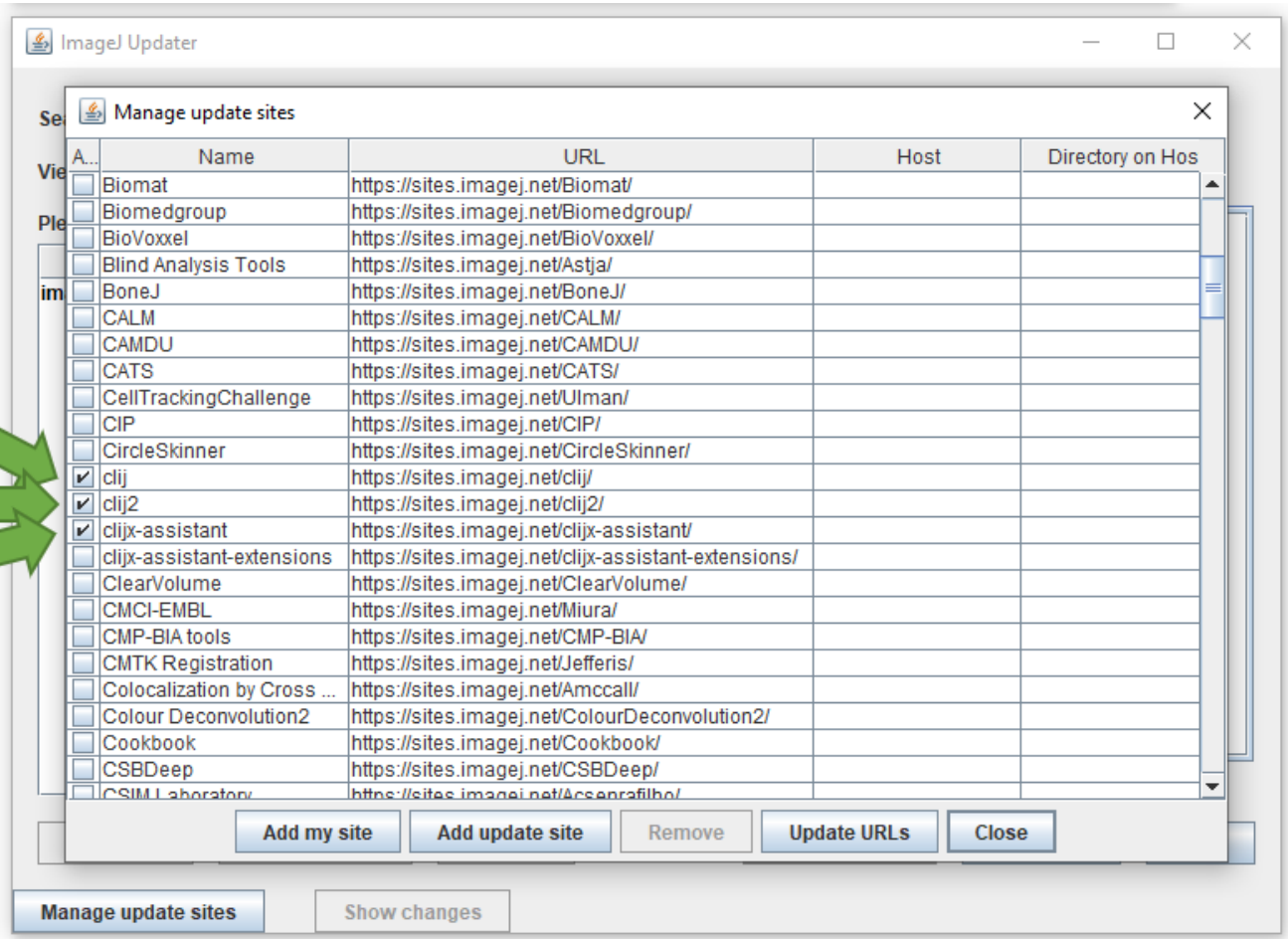


Exercises

Exercise 0: Installing clij, clij2 and the clijx-assistant

- Just activate the CLIJ update sites, in menu Help > Update…



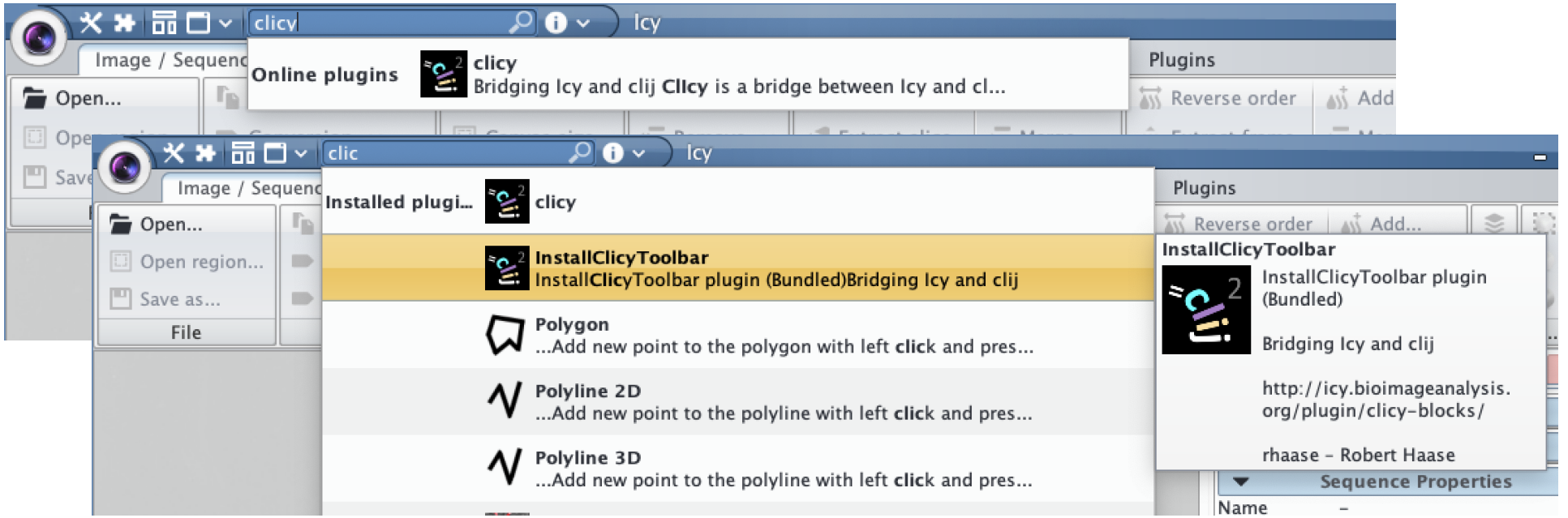


Exercise 0: Installing ICY / CLIcy

Download and unzip ICY from : http://icy.bioimageanalysis.org (Java required)

Search and select “CLIcy” in the search bar to download and Install it

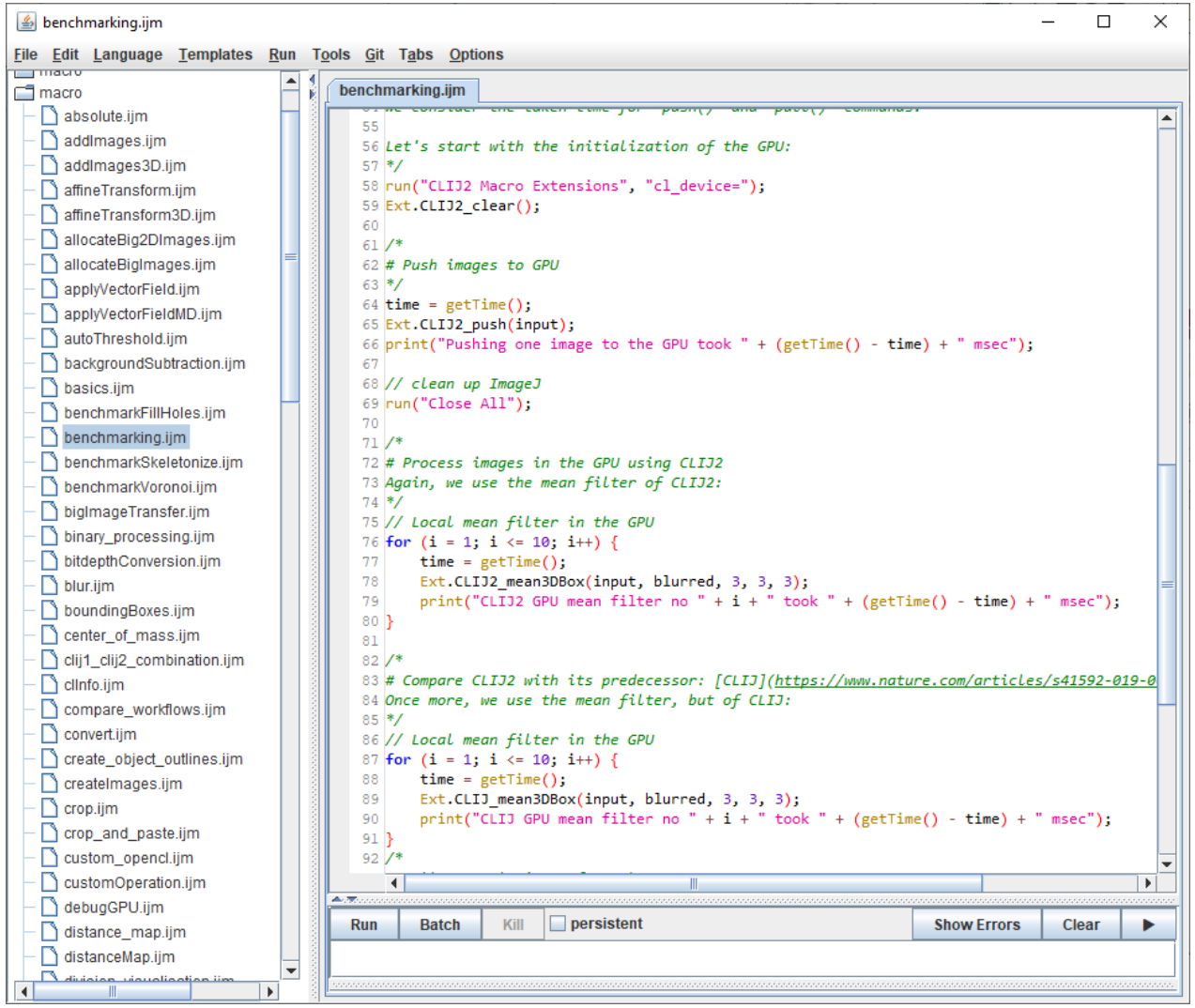
Finally run CLIcy Toolbar Installer (hit: use the search again!)

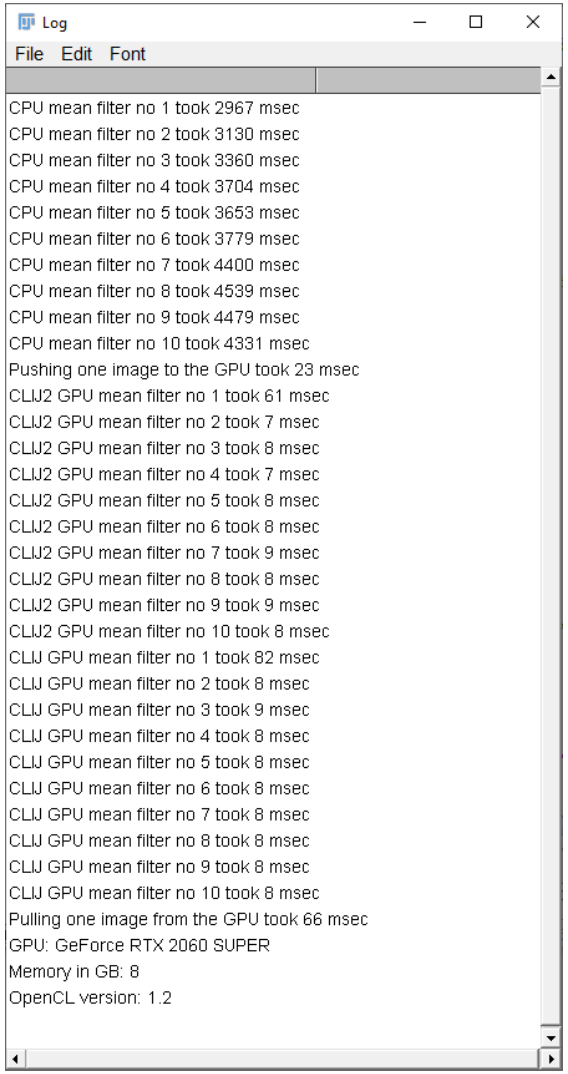


Restart ICY and have fun with the new possibilities!

Exercise 1: Testing if CLIJ2 works

- Get an example macro from http://clij.github.io/clij2-docs/ > Code examples > ImageJ Macro > benchmarking.ijm

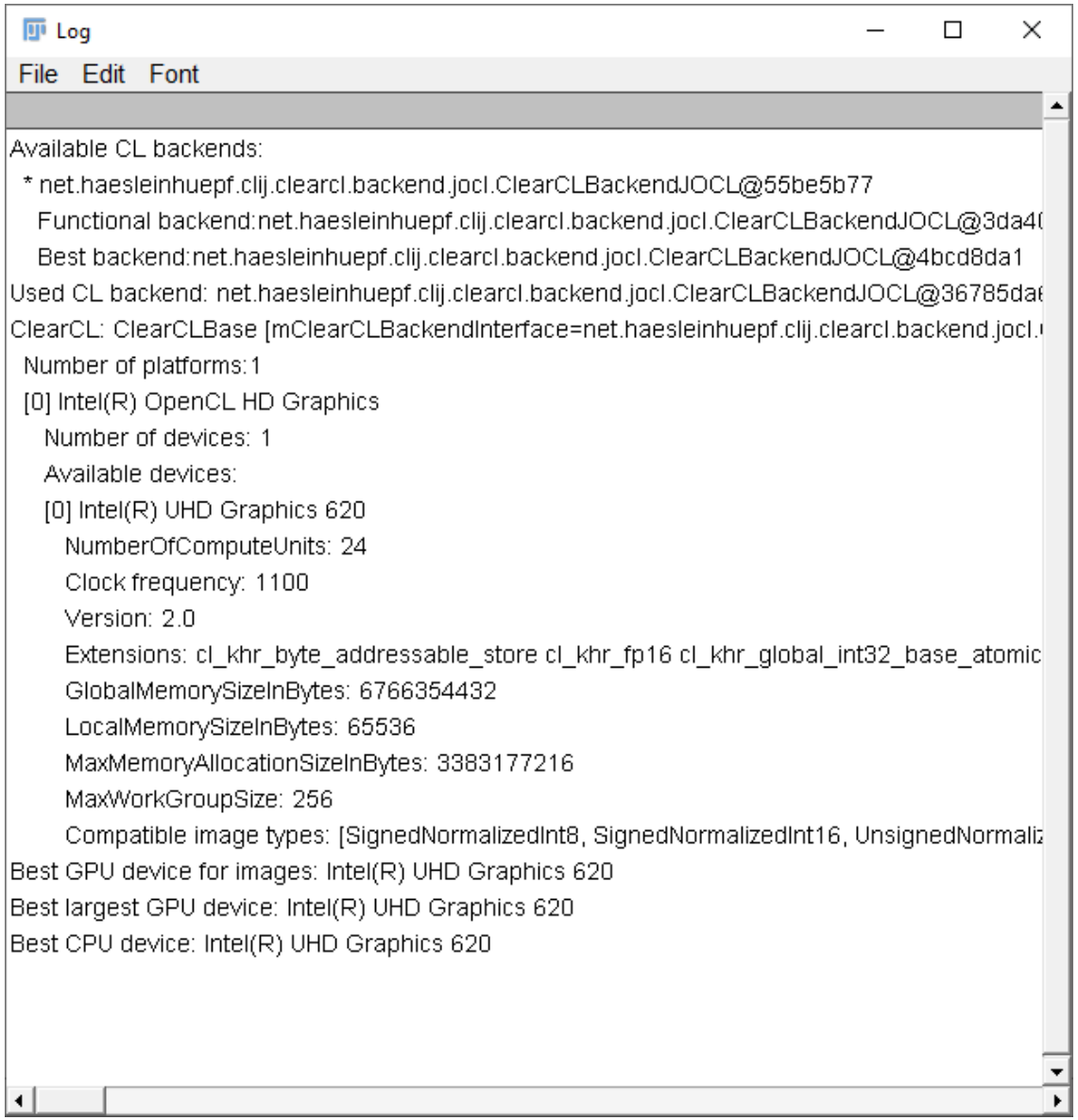




Exercise 2: Study your hardware properties

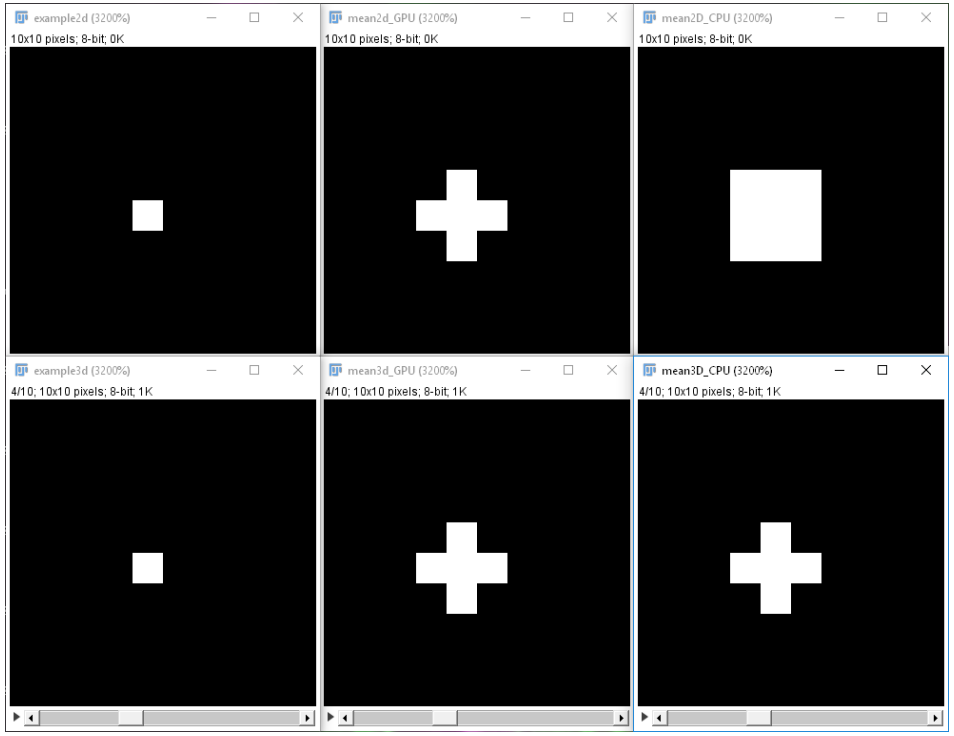
- Execute this script: https://github.com/clij/clij2-docs/blob/master/src/main/macro/clInfo.ijm

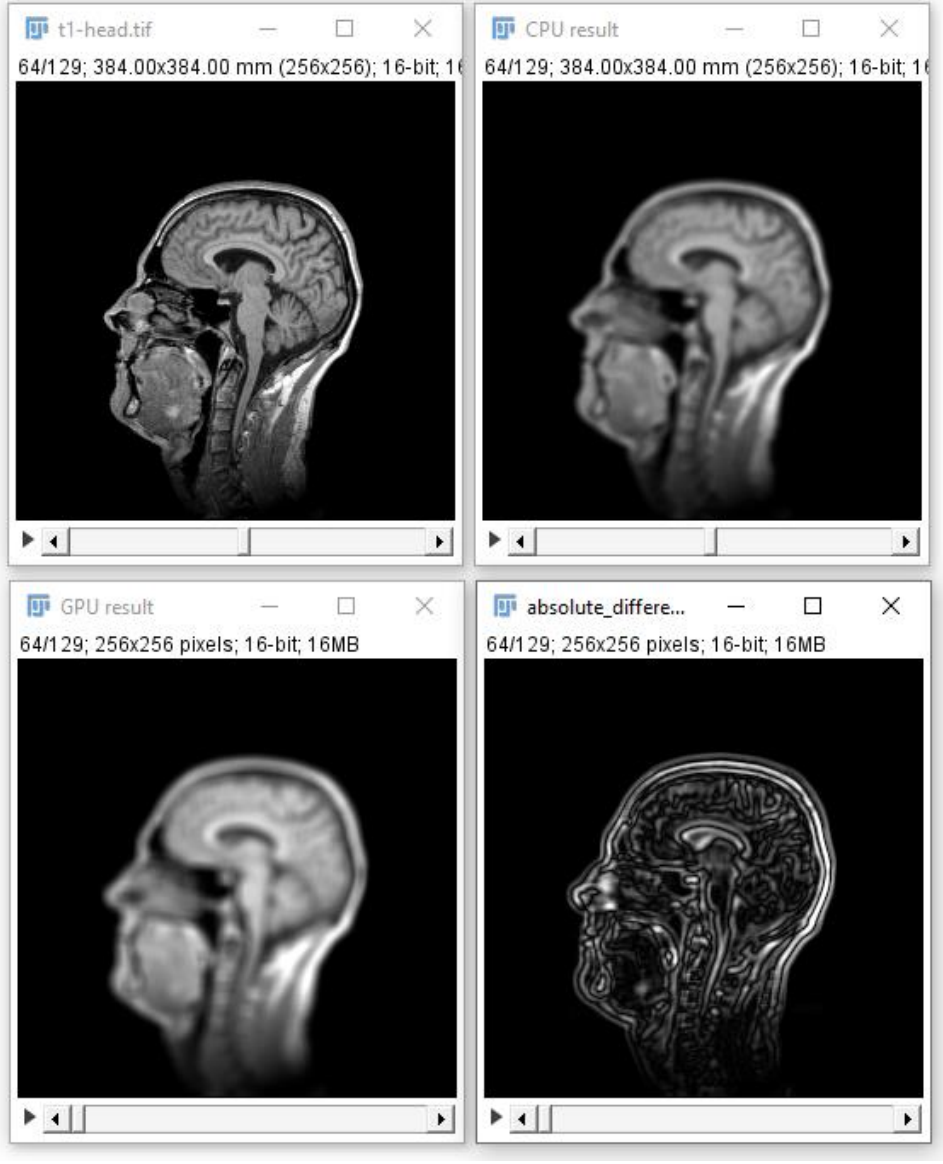
- What GPU is used per default?
- Home much memory does it have available?
- How large can a single image be when being processed in the GPU?
- Assume your image stack is 1024x1024 pixels large and of type 32-bit float, how many planes can it have so that it still fits in GPU memory?



Exercise 3: Quantitative comparison

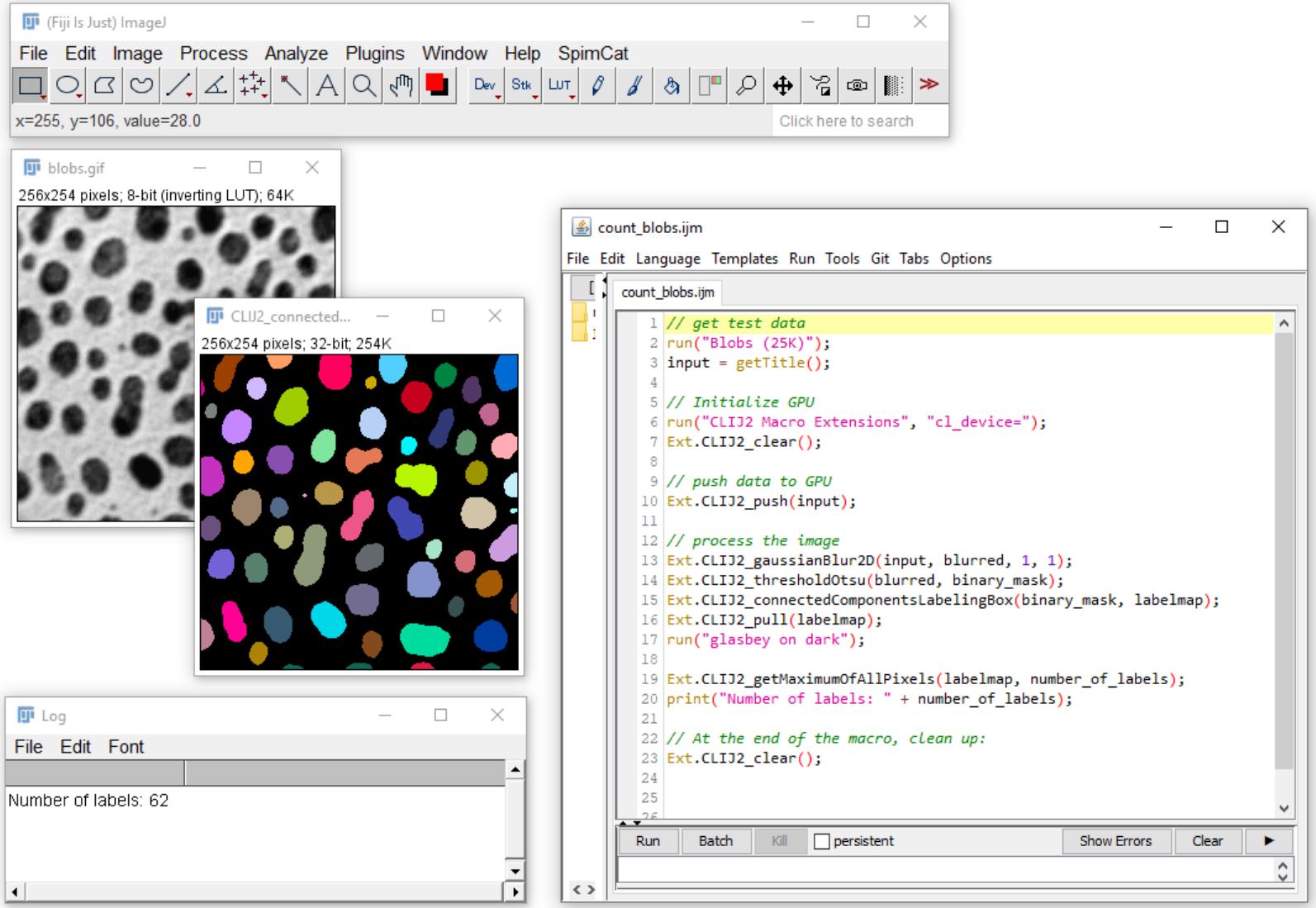
- Compare the mean filters in ImageJ and CLIJ when processing the t1-head example dataset. Compute an image showing the difference or the absolute difference between both results.
- Try mean3DBox and mean3DSphere. Which one is closer to ImageJ's 'Mean 3D' filter?
- Also run mean_detailed_comparison_IJ_CLIJ.ijm. What can you learn from it?





Exercise 4: Segmentation

- Setup a workflow for counting blobs in ImageJ's blobs.gif example image. Use these functions:
  - Gaussian Blur
  - Otsu's threshold method
  - Connected Component Labeling
  - Measure maximum intensity in the resulting label image
- In order to achieve this, it is recommended to use the Macro recorder or auto-completion in the script editor.
- Optional: Write this script in Fijis Jython or Icy JavaScript. If you use Icy, turn on CLIJs recorder and execute the operations from the menu.



PoL Physics of Life TU Dresden

INSTITUT PASTEUR

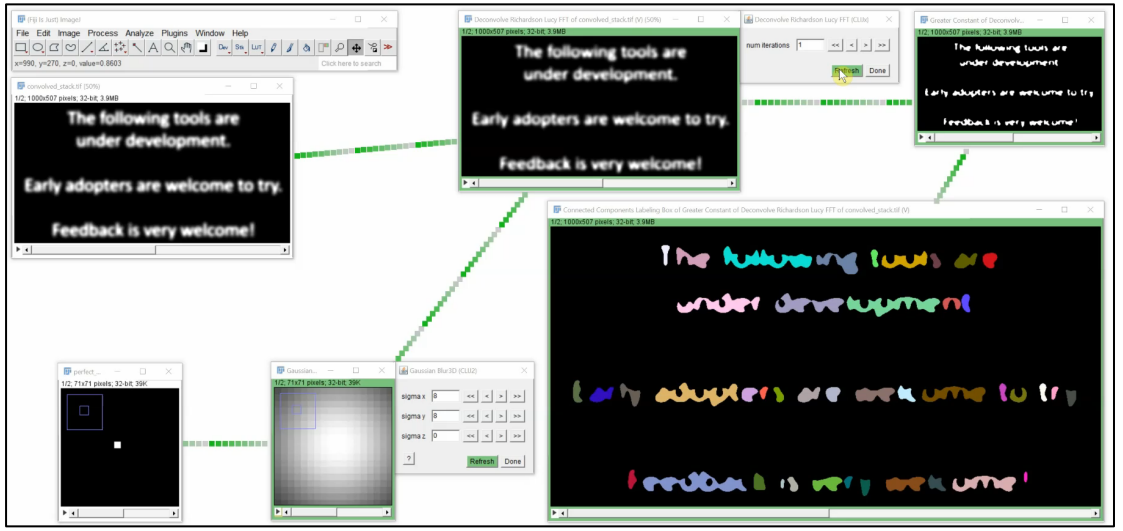
CENTER FOR SYSTEMS BIOLOGY DRESDEN

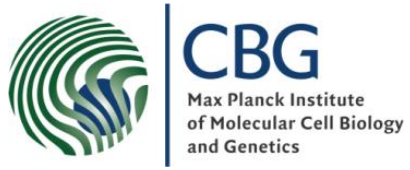
CBG Max Planck Institute of Molecular Cell Biology and Genetics

Pause 15 min

Coming up next:

Workflow design

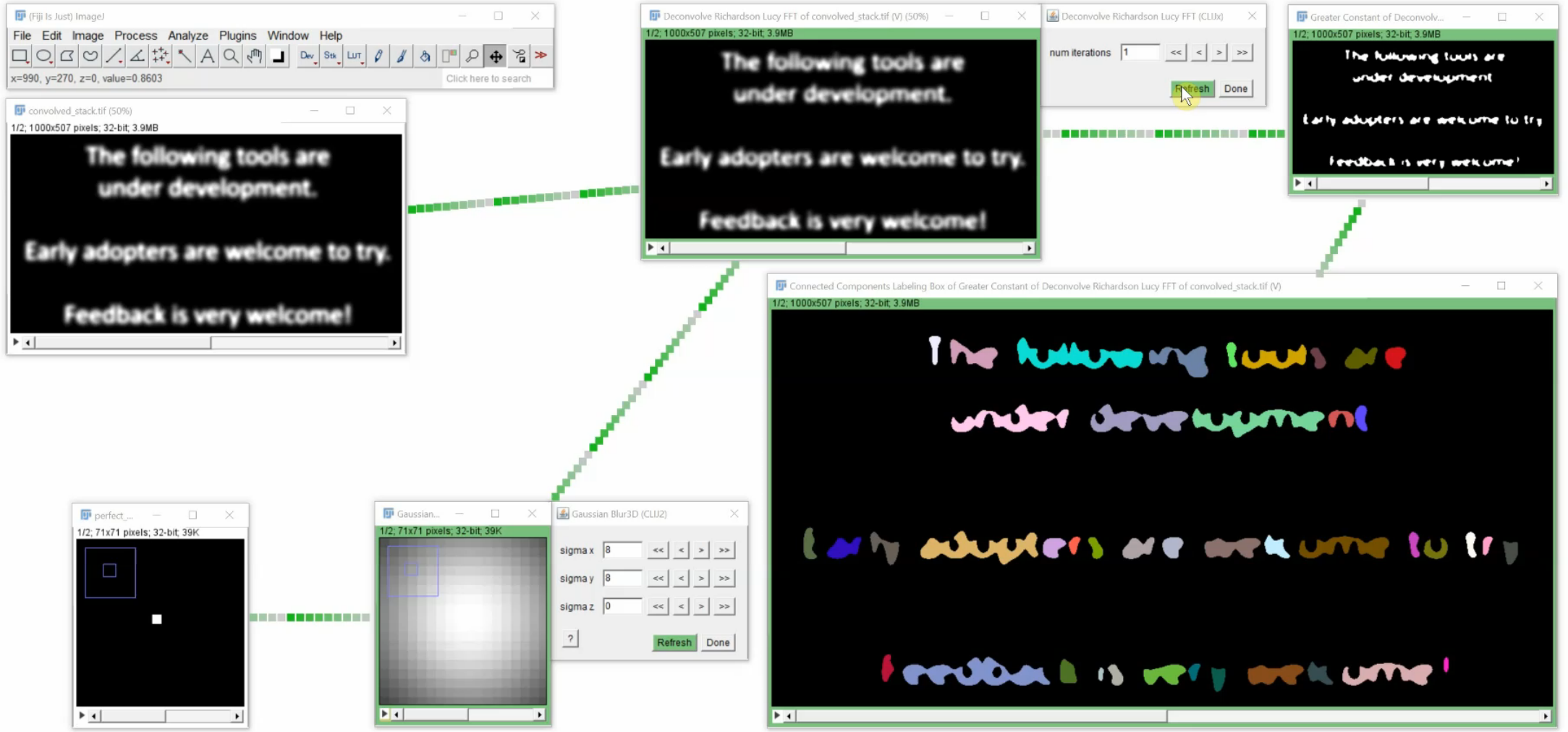




Part II

Interactive Image Data Flow Graph design and code-generation

Image Data Flow Graphs & clEsperanto



Special thanks to Brian Northon! @truenorth_ia

https://en.wikipedia.org/wiki/Release_early,_release_often

Image Data Flow Graphs & clEsperanto

- Use yhe magic tool button and the magic mouse.

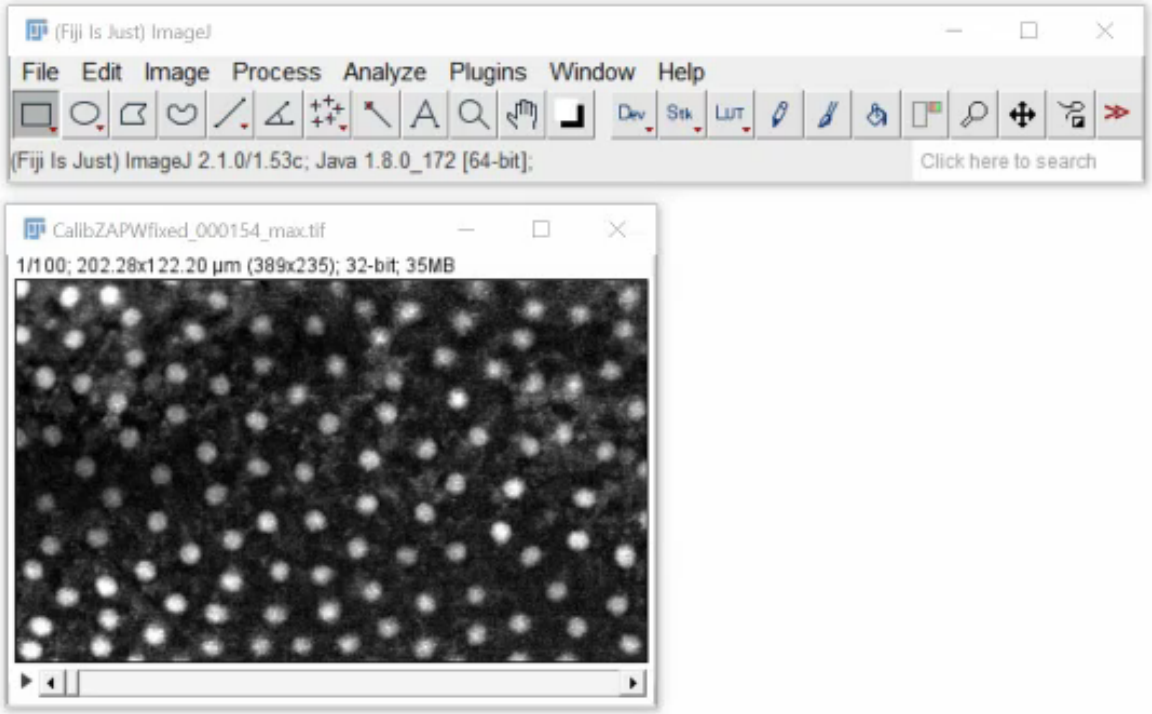
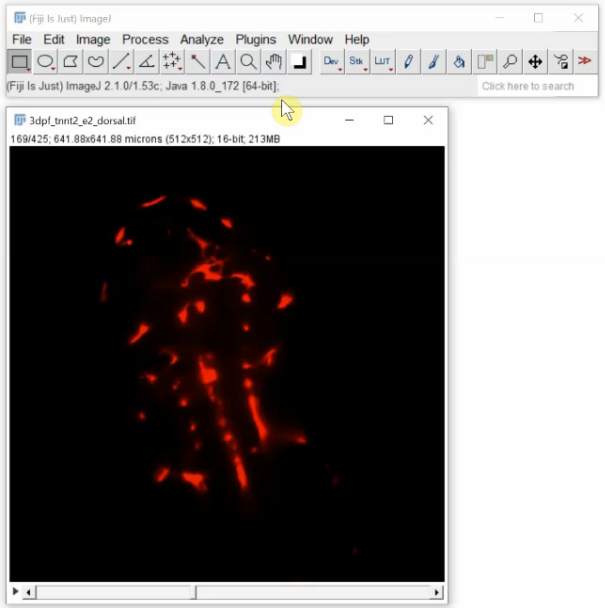
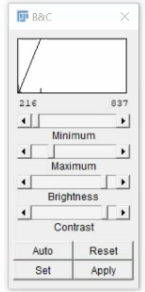
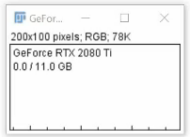


Image Data Flow Graphs & clEsperanto

- Windows are connected and image data flows from the input image downstream.







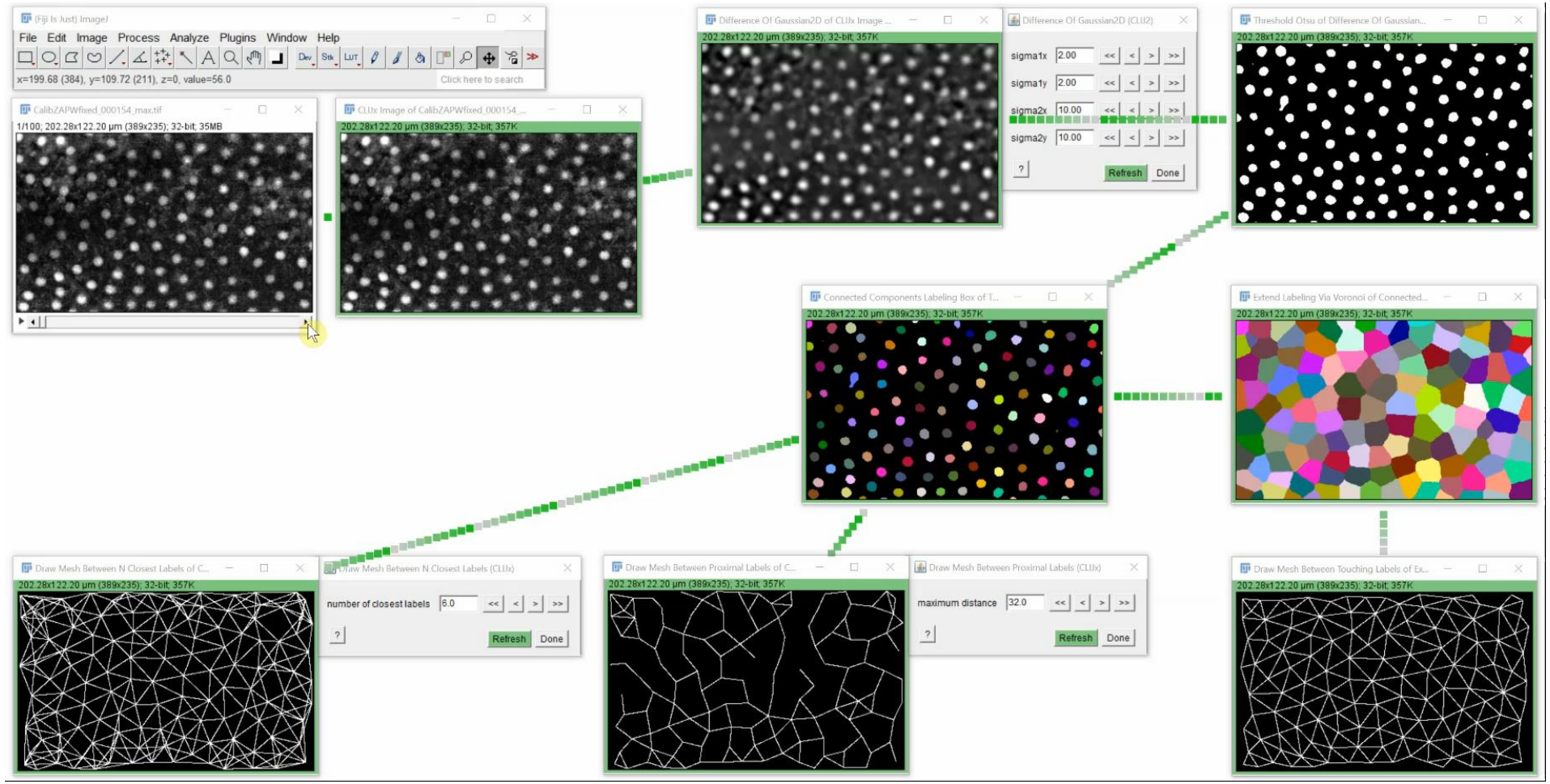
Special thanks to Elisabeth Kugler! @KuglerElisabeth

@haesleinhuepf @StRigaud https://clij.github.io/

Image data source: Elisabeth Kugler; labs of Tim Chico and Paul Armitage, The University of Sheffield (UK)" https://zenodo.org/record/4204839#.X8DCRGj7Q2w

Image Data Flow Graphs & clEsperanto

- Study relationships between operations and compare different strategies



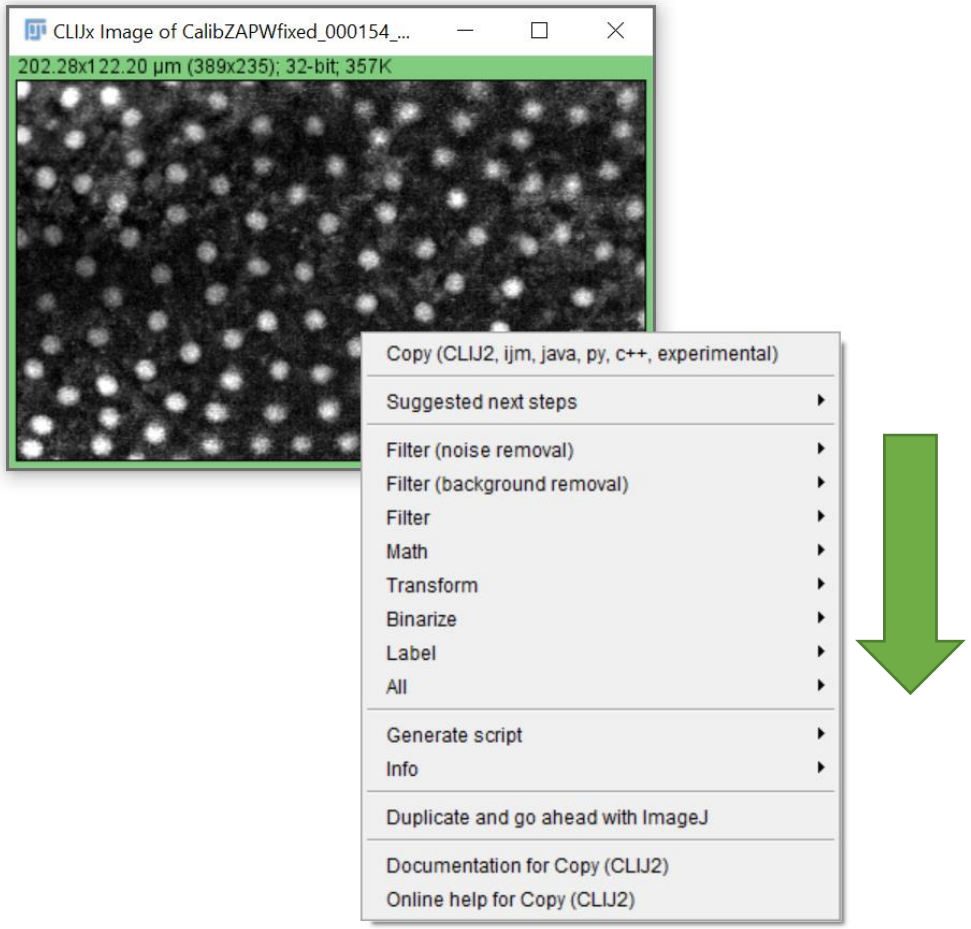
Draw mesh between N closest neighbors

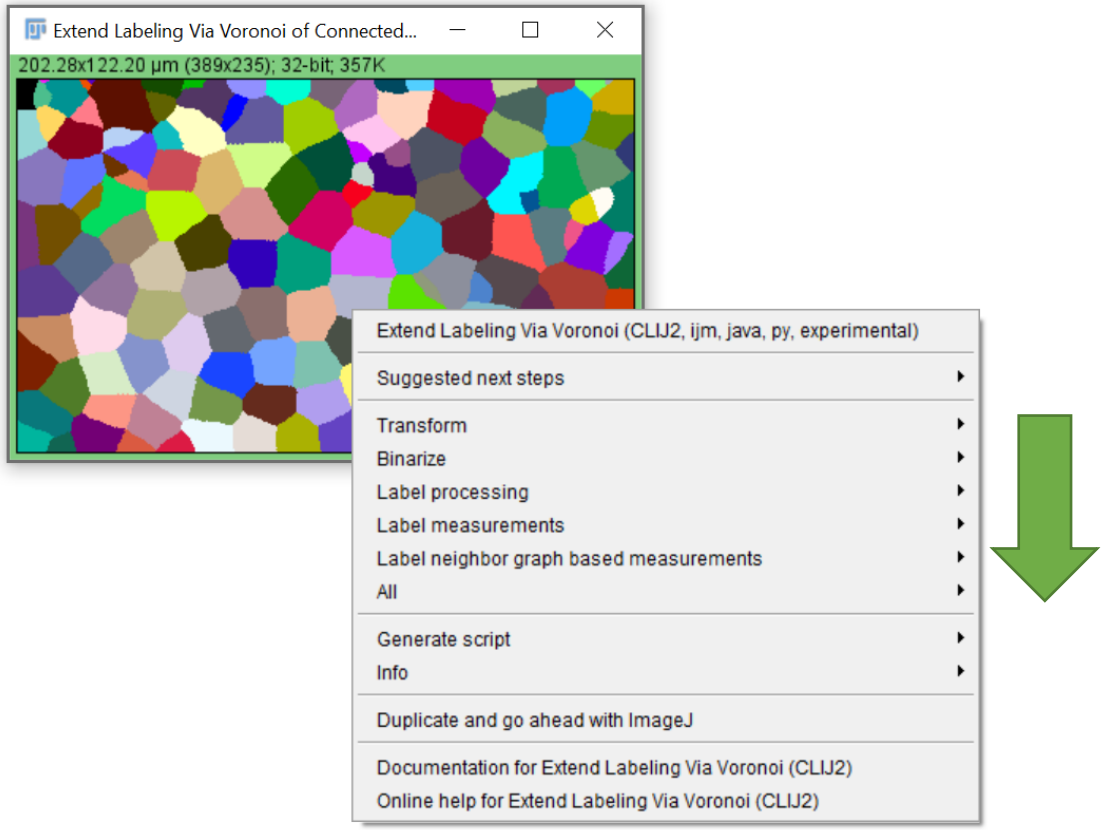
Draw mesh between neighbors closer than d pixels

Draw mesh between touching neighbors

Image Data Flow Graph design

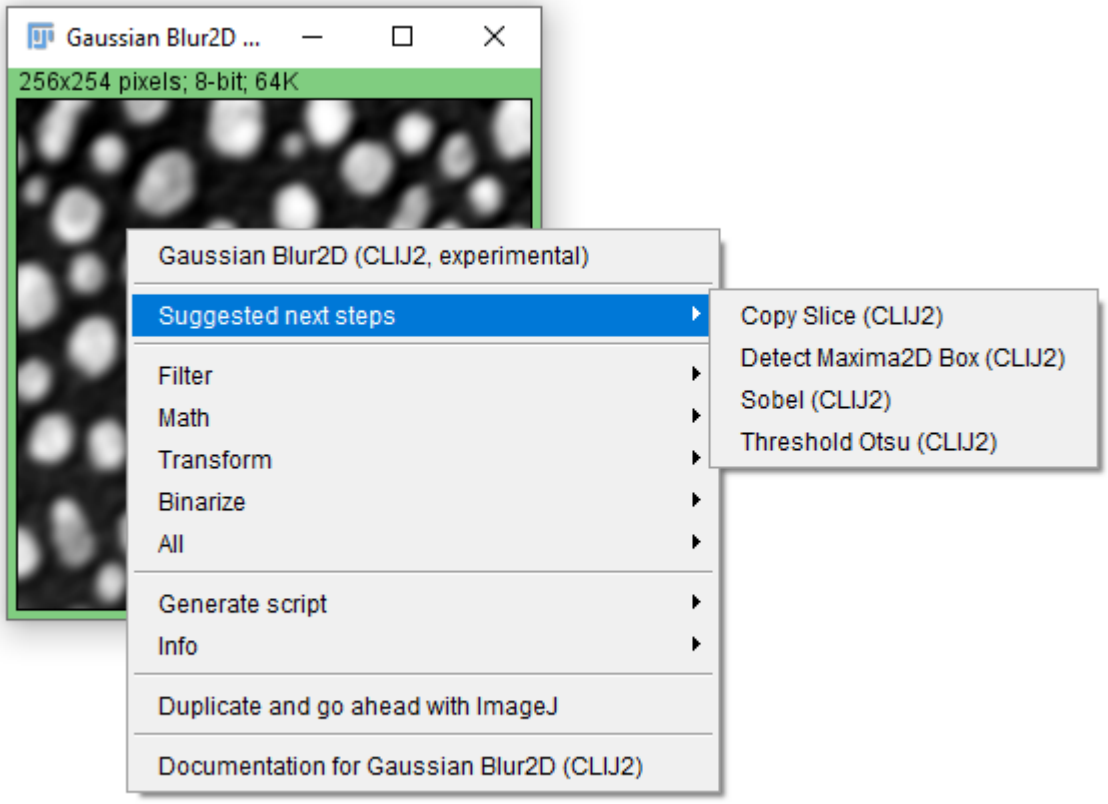
- The menu order is intentional: From preprocessing to analysis

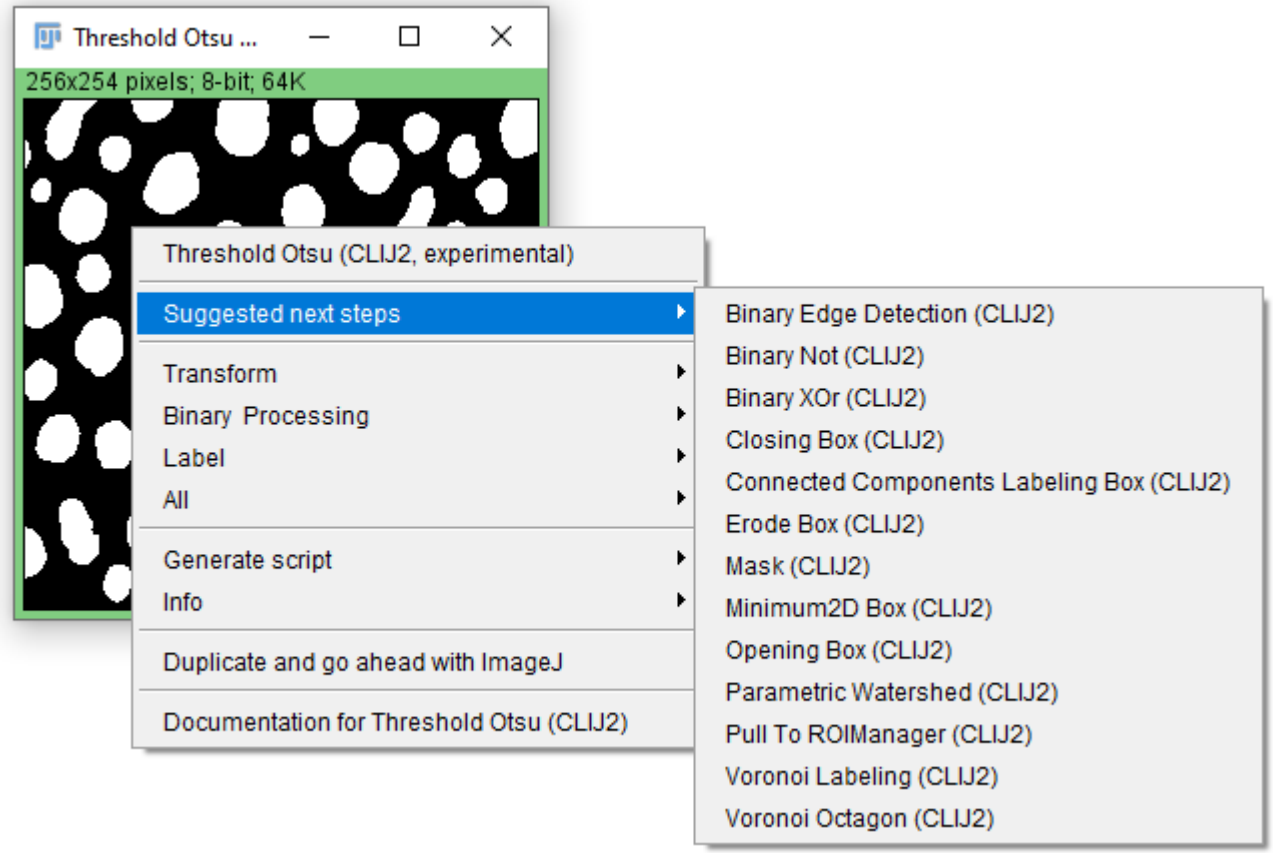




Expert system: Suggestions

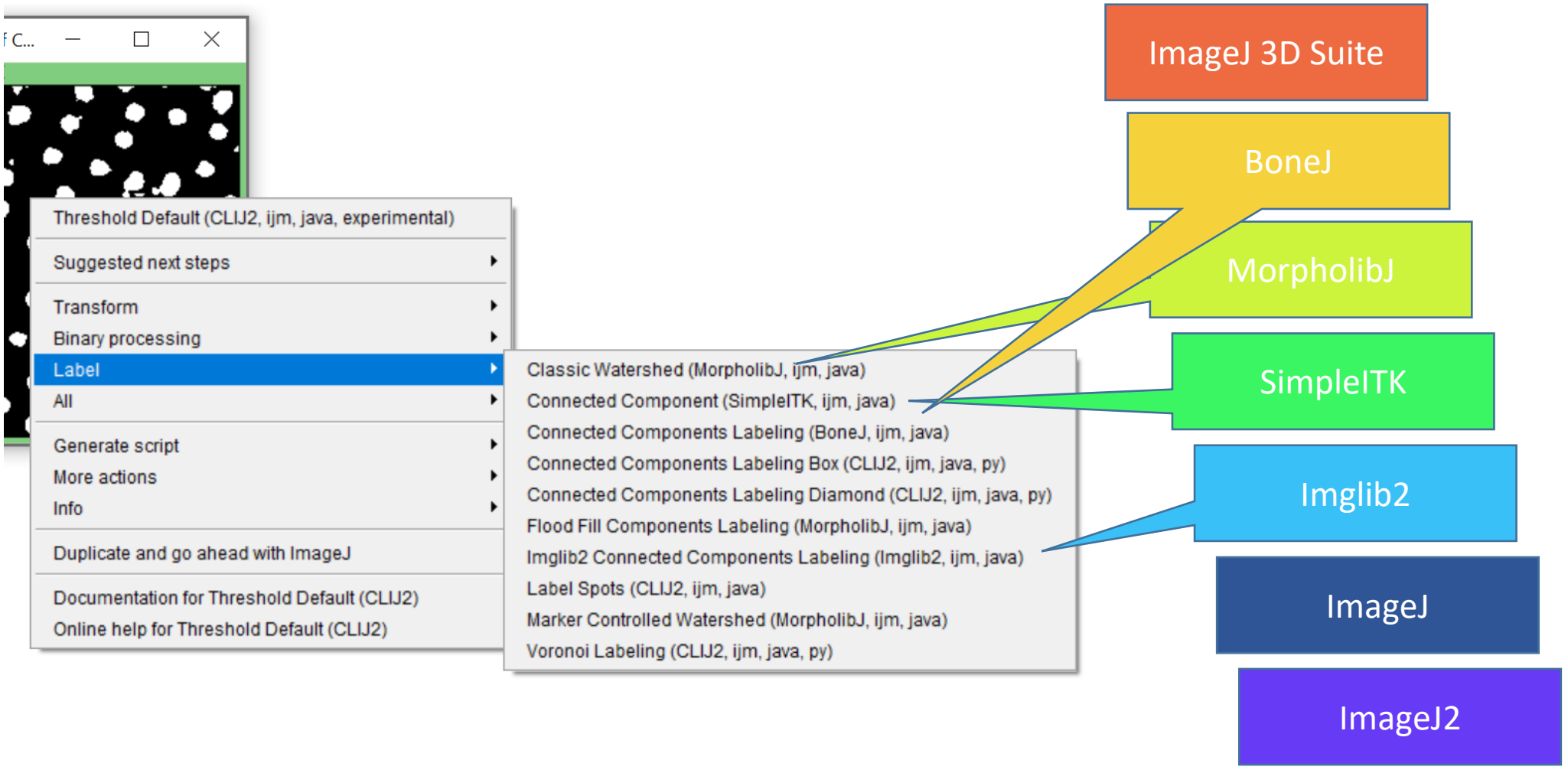
- Explore suggestions!





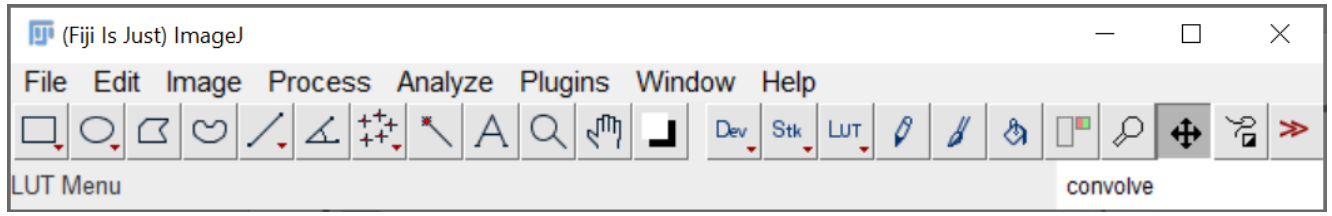
clEsperanto: Extensibility

- Watch out for other image processing libraries.
  - Install: https://clij.github.io/assistant/installation#extensions



Fiji search bar

- In Fijis search bar result, there is a new category: CLIJx-assistant which offers IDFG operations



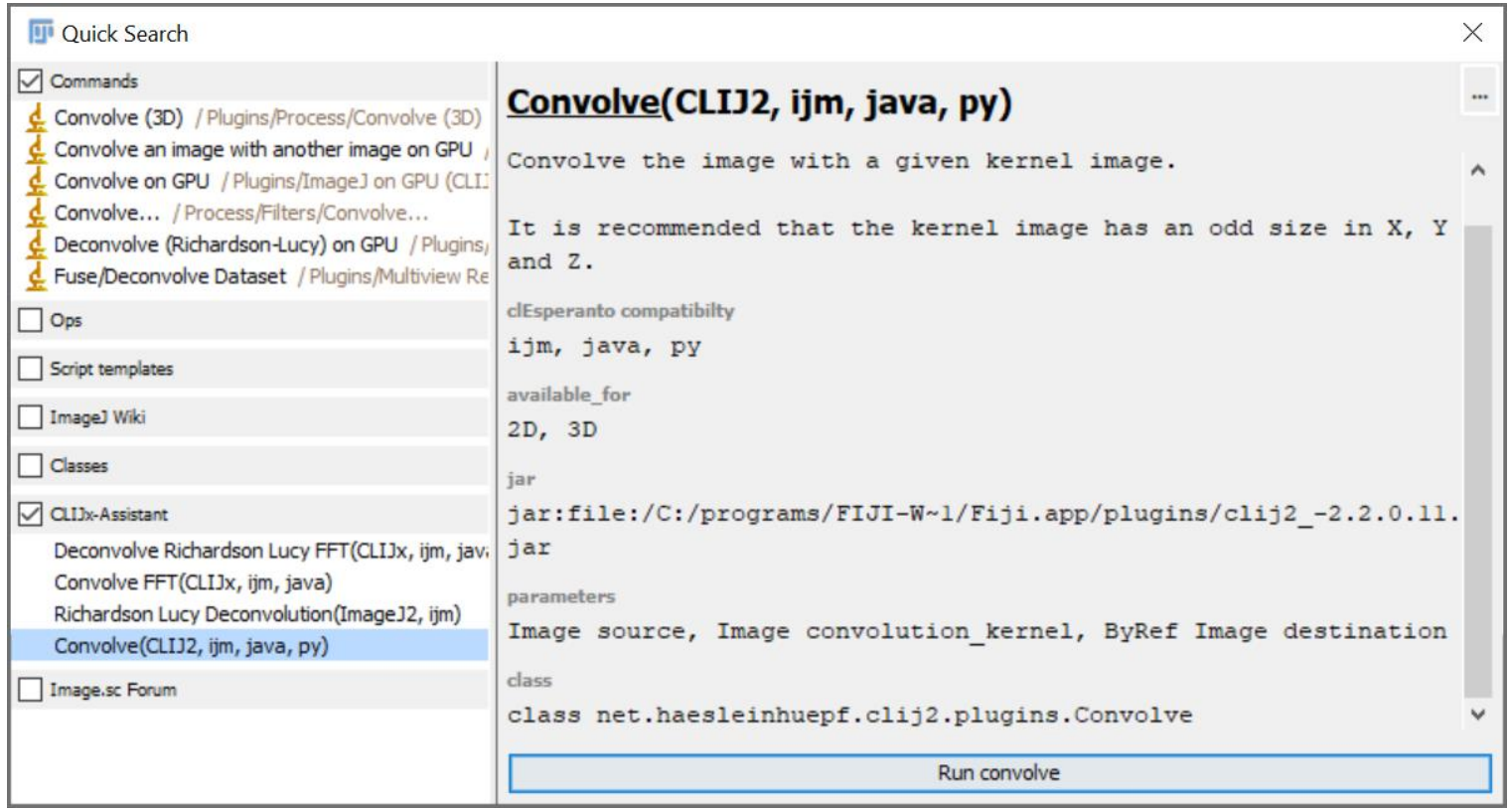
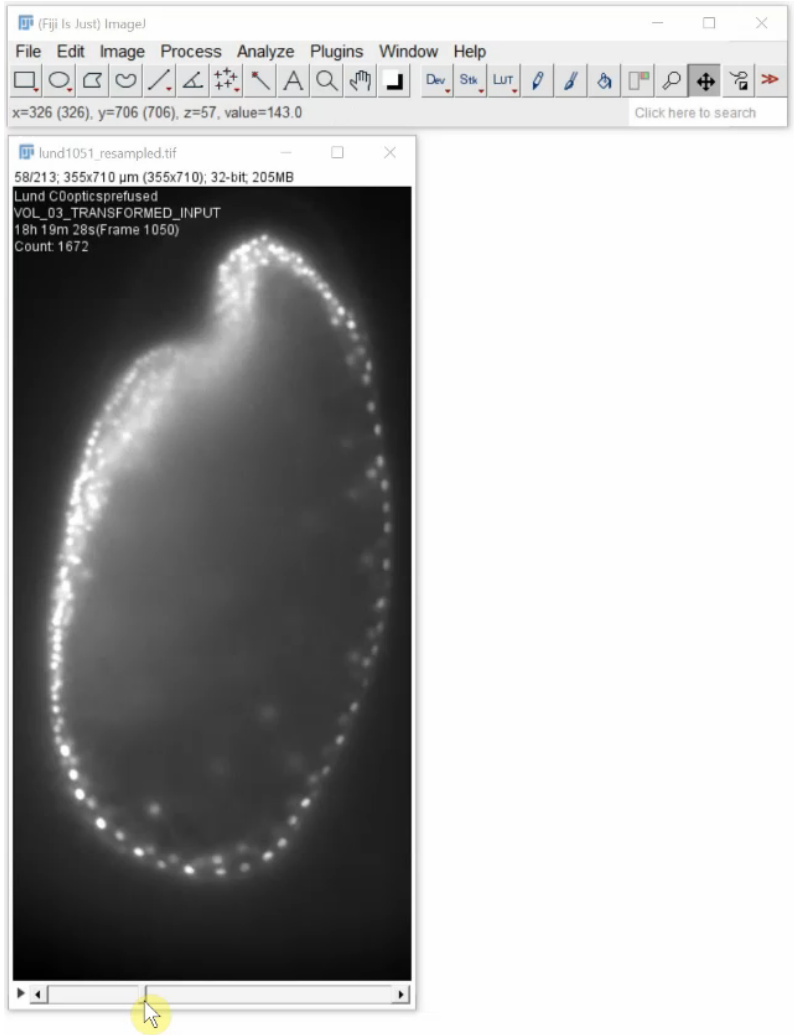
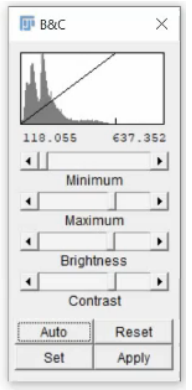


Image Data Flow Graphs

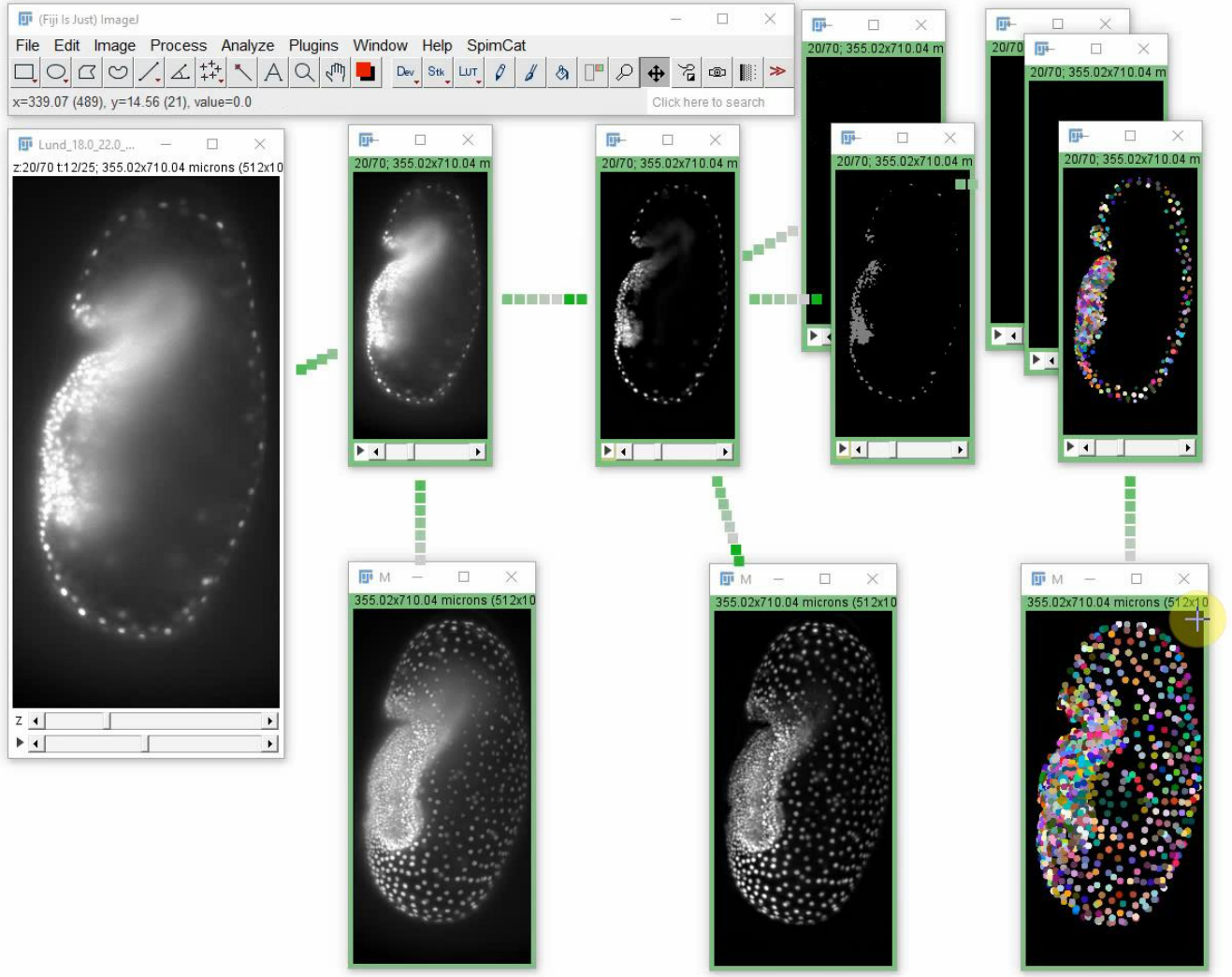
- Don’t let the windows rule your desktop. Rule the windows!





Export to scripts

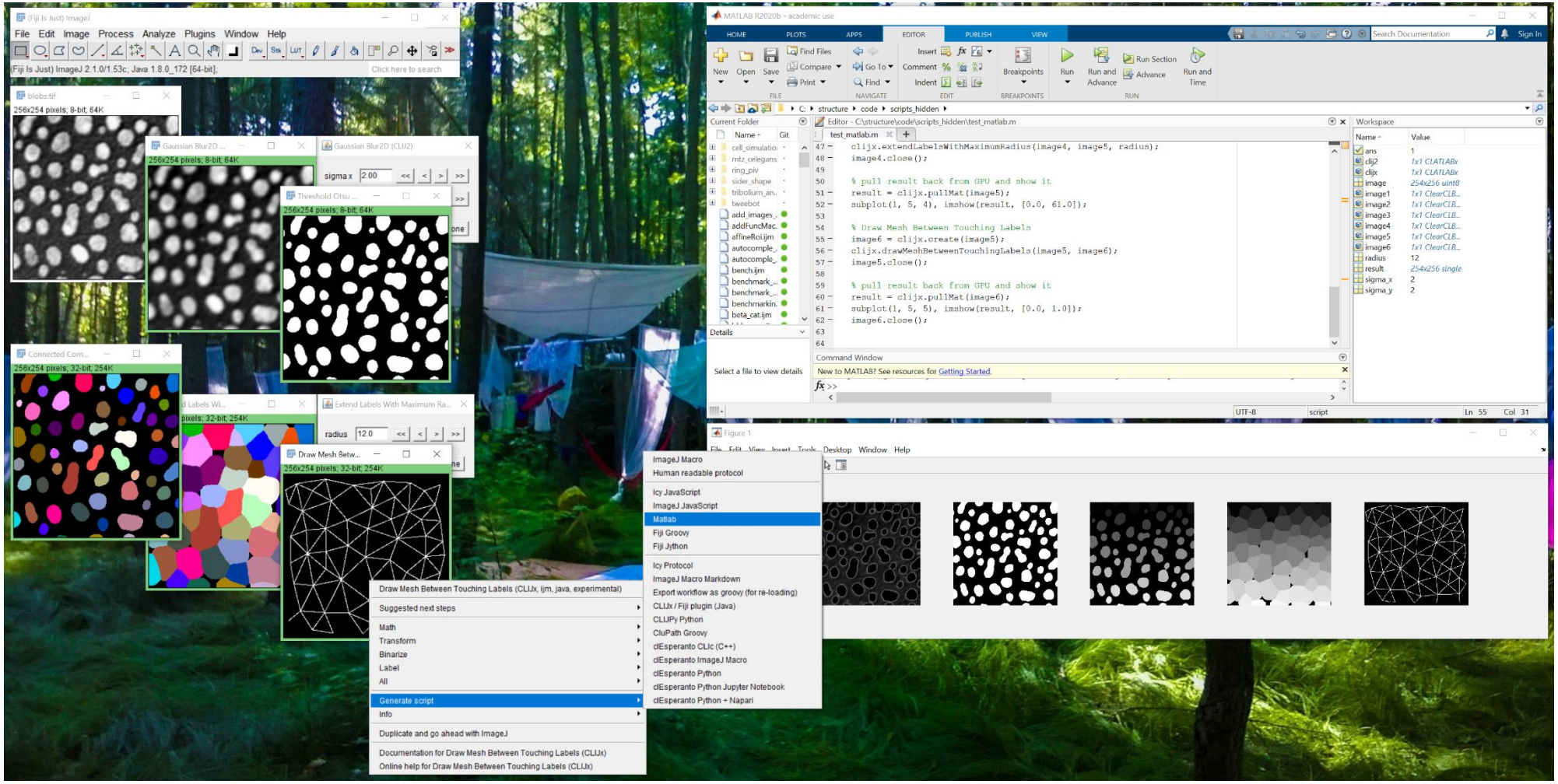
- Export to multiple programming languages*



* Icy Protocols, QuPath Groovy, Python, C++ ready for testing / early adopters

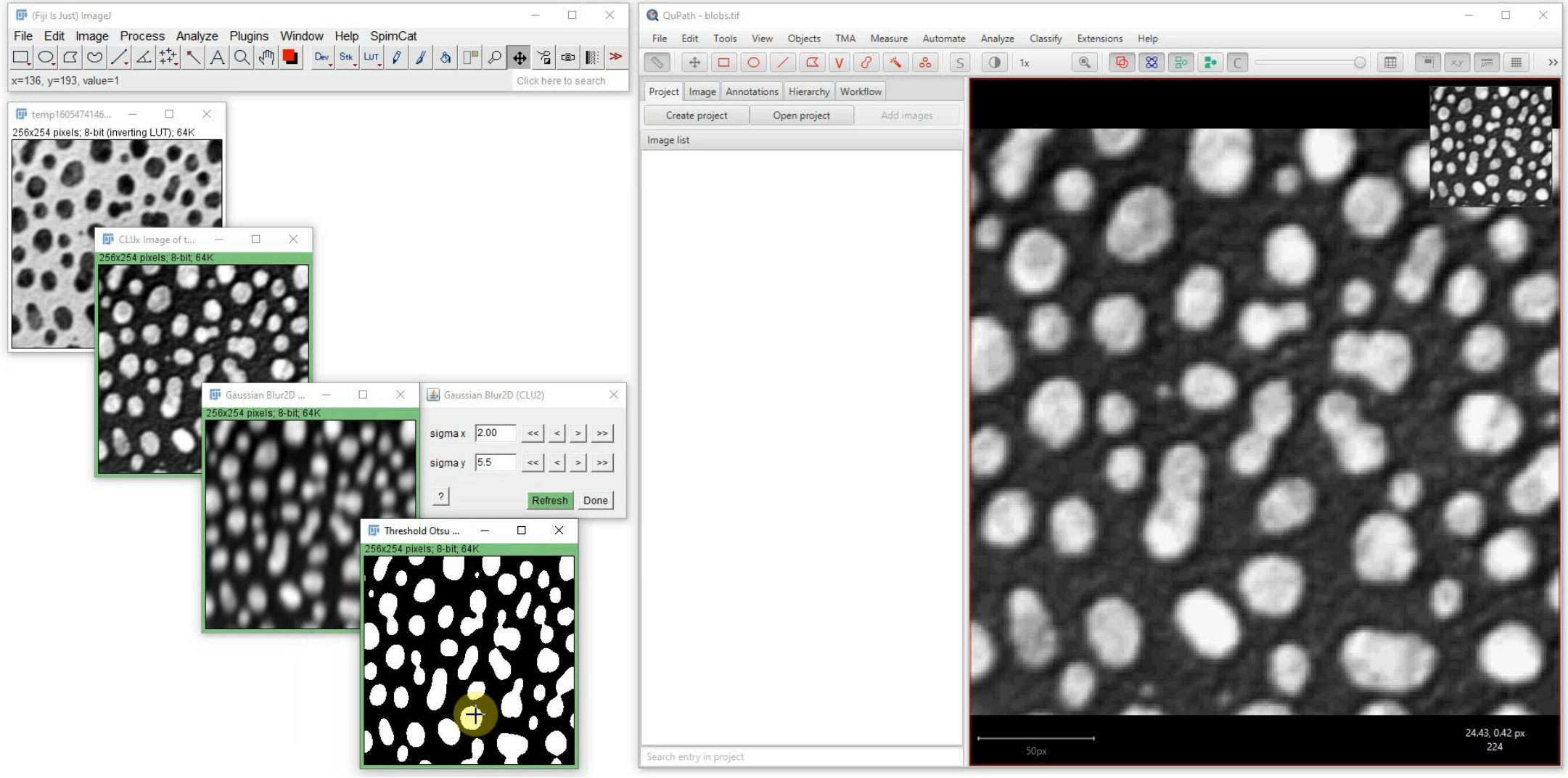
clEsperanto: clatlab

- Export scripts to Matlab
  - Install https://github.com/clij/clatlab or https://github.com/clij/clatlabx



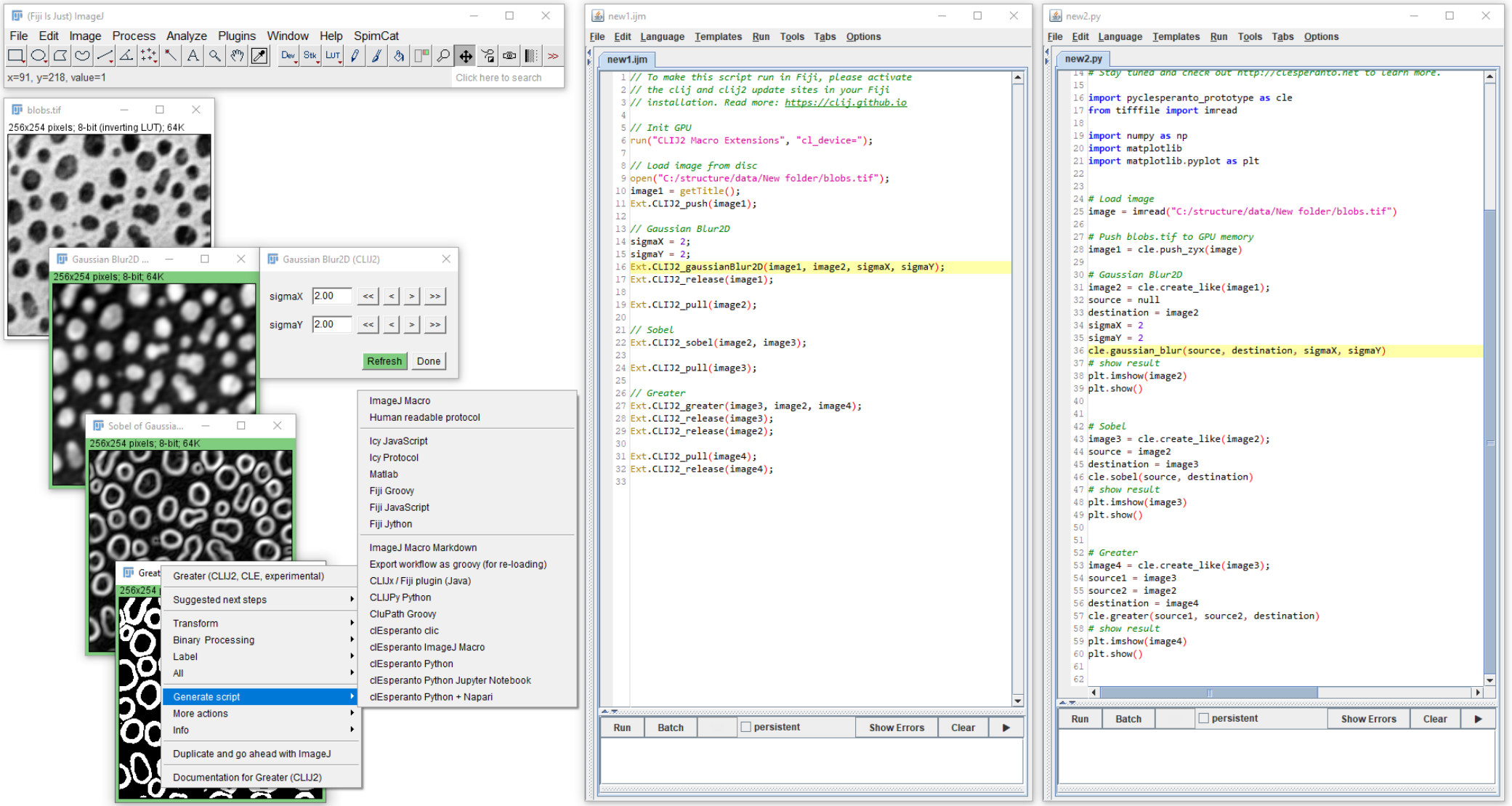
clEsperanto: cluPath

- If your workflow ends in a binary image, you can export a Groovy script for QuPath



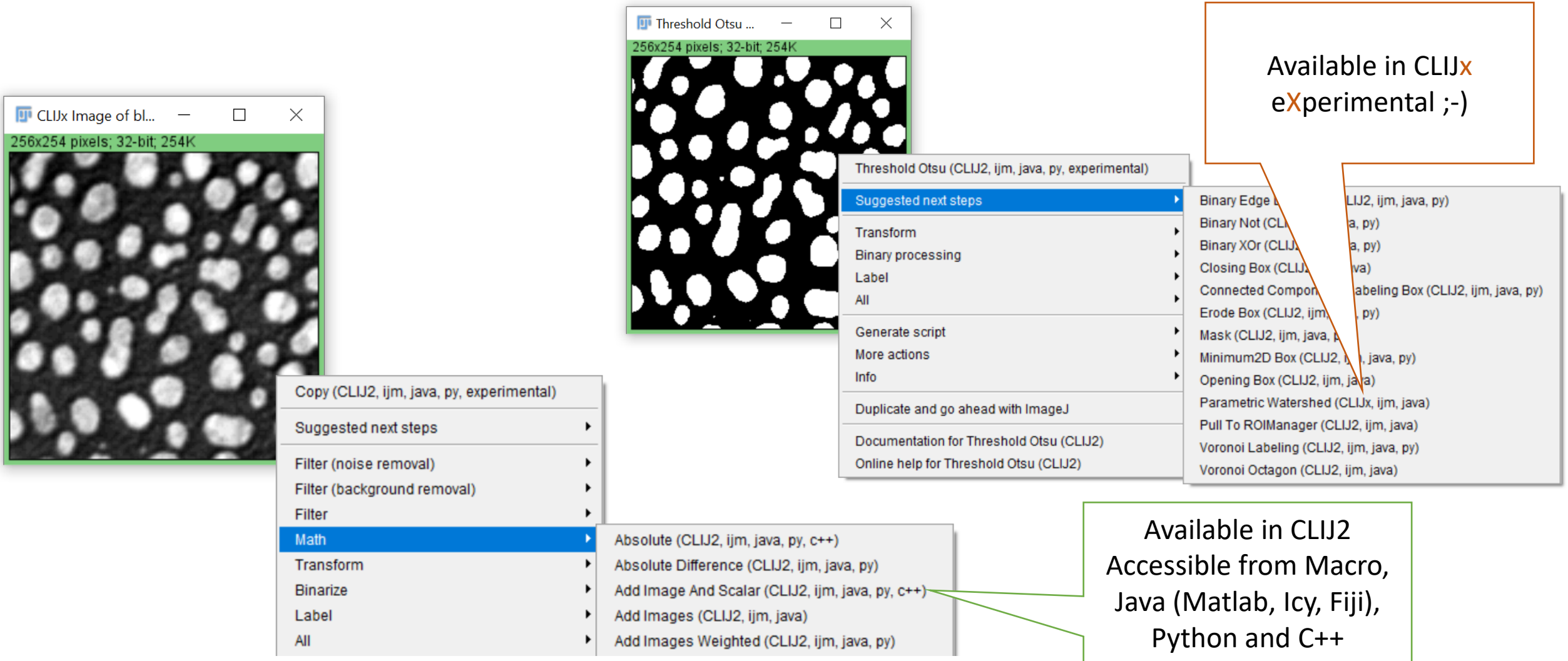
clEsperanto script language comparison

- Generate multiple scripts in multiple languages from a given Image Data Flow Graph



clEsperanto: Compatibility

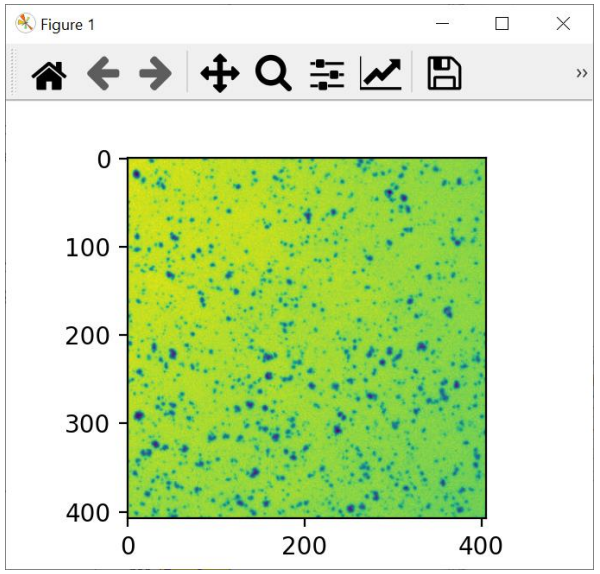
- Some operations are not available in all environments

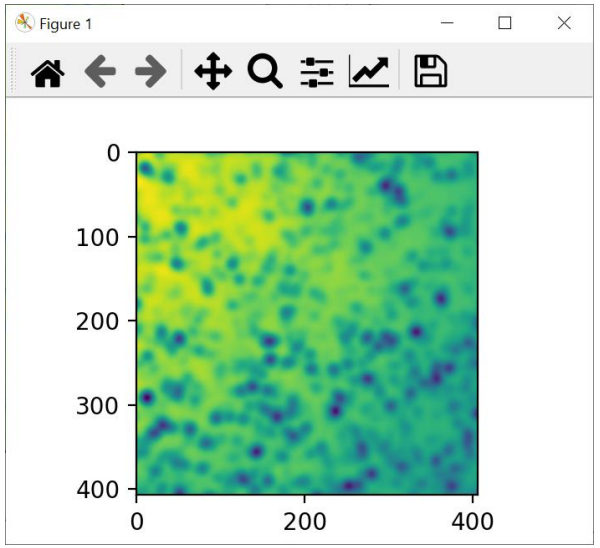


pyclesperanto: What every Python script must have

- Load data
- Initialize GPU
- Push
- Process images
- Pull
- ~~Cleanup~~

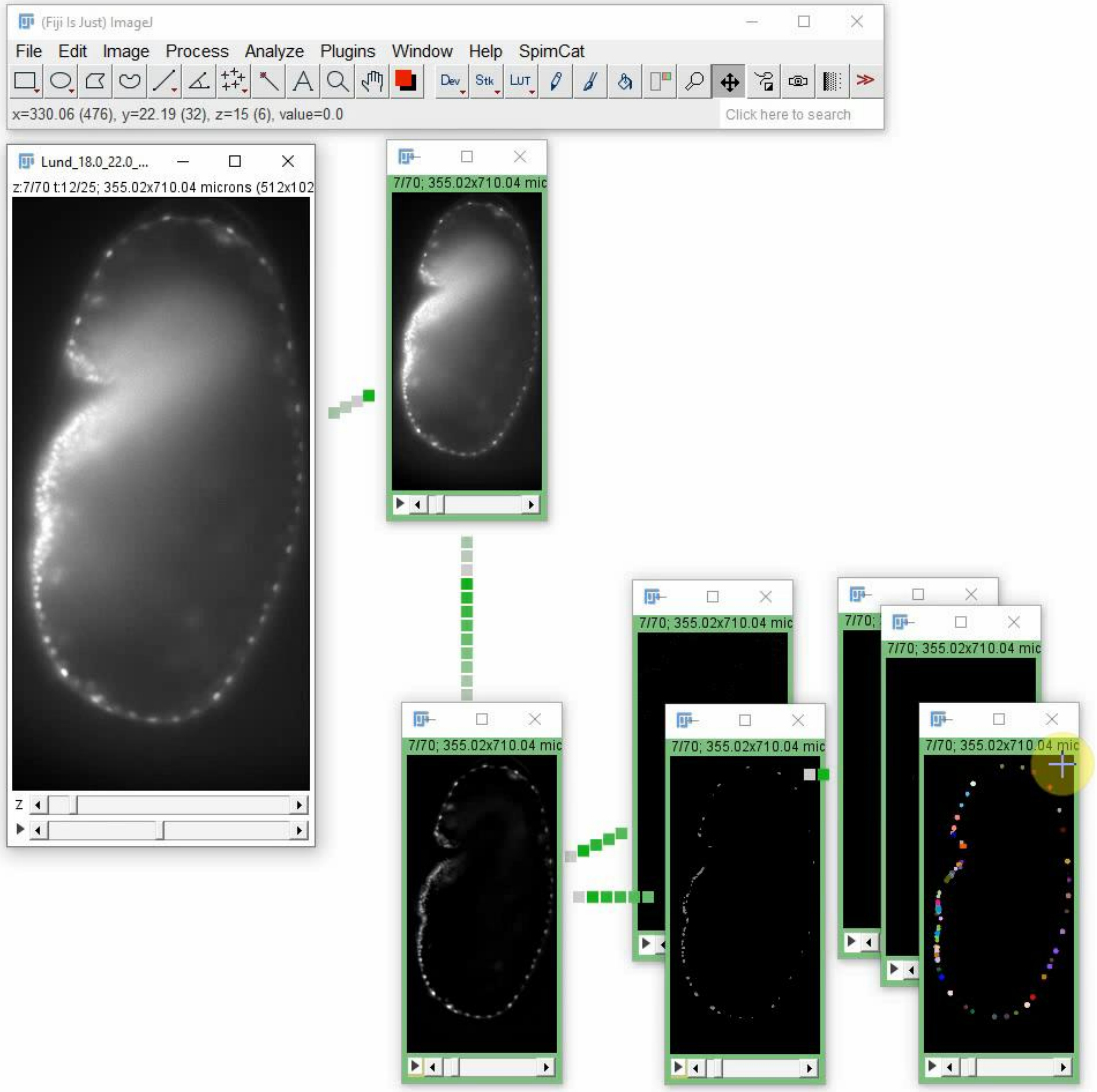
```python
from skimage.io import imread
import matplotlib.pyplot as plt

# Load image
image = imread("https://imagej.nih.gov/ij/images/Cell_Colony.jpg")
plt.imshow(image)
plt.show()

# initialize GPU
import pyclesperanto_prototype as cle
# optional:
cle.select_device("GPU name")

# Push blobs.tif to GPU memory
image1 = cle.push_zyx(image)

# Threshold Otsu
sigma = 5
blurred = cle.create_like(image1)
cle.gaussian_blur(image1, blurred, sigma, sigma)

# pull result back from GPU
result = cle.pull_zyx(blurred)
plt.imshow(result)
plt.show()
```





pyclesperanto + napari script export

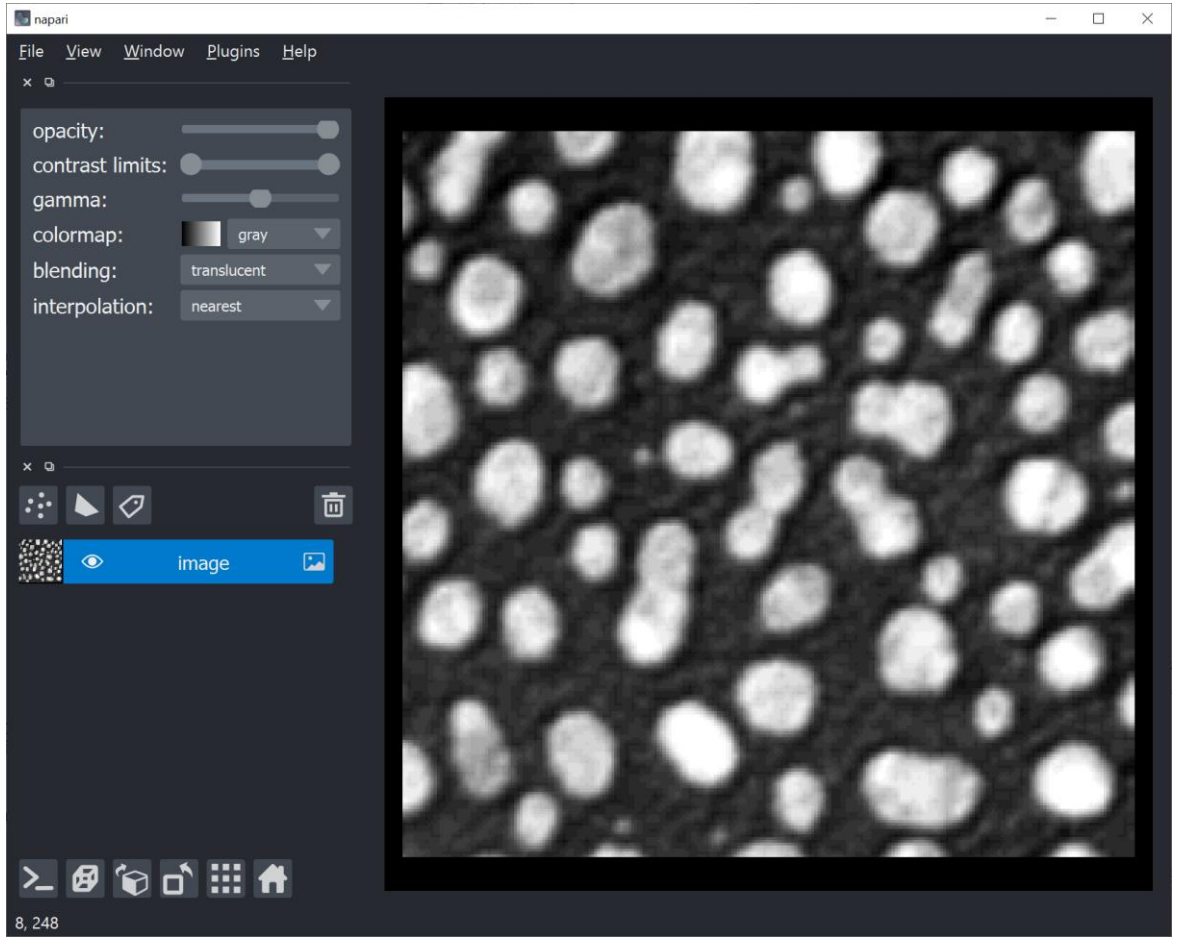
- napari is a multidimensional image viewer in the Python ecosystem



pyclesperanto + napari

- Barebone minimal napari script

```python
# Load image
from tifffile import imread
image = imread("C:/structure/data/blobs.tif")

# Start napari viewer
import napari
with napari.gui_qt():
    viewer = napari.Viewer()

    # show image
    viewer.add_image(image)
```

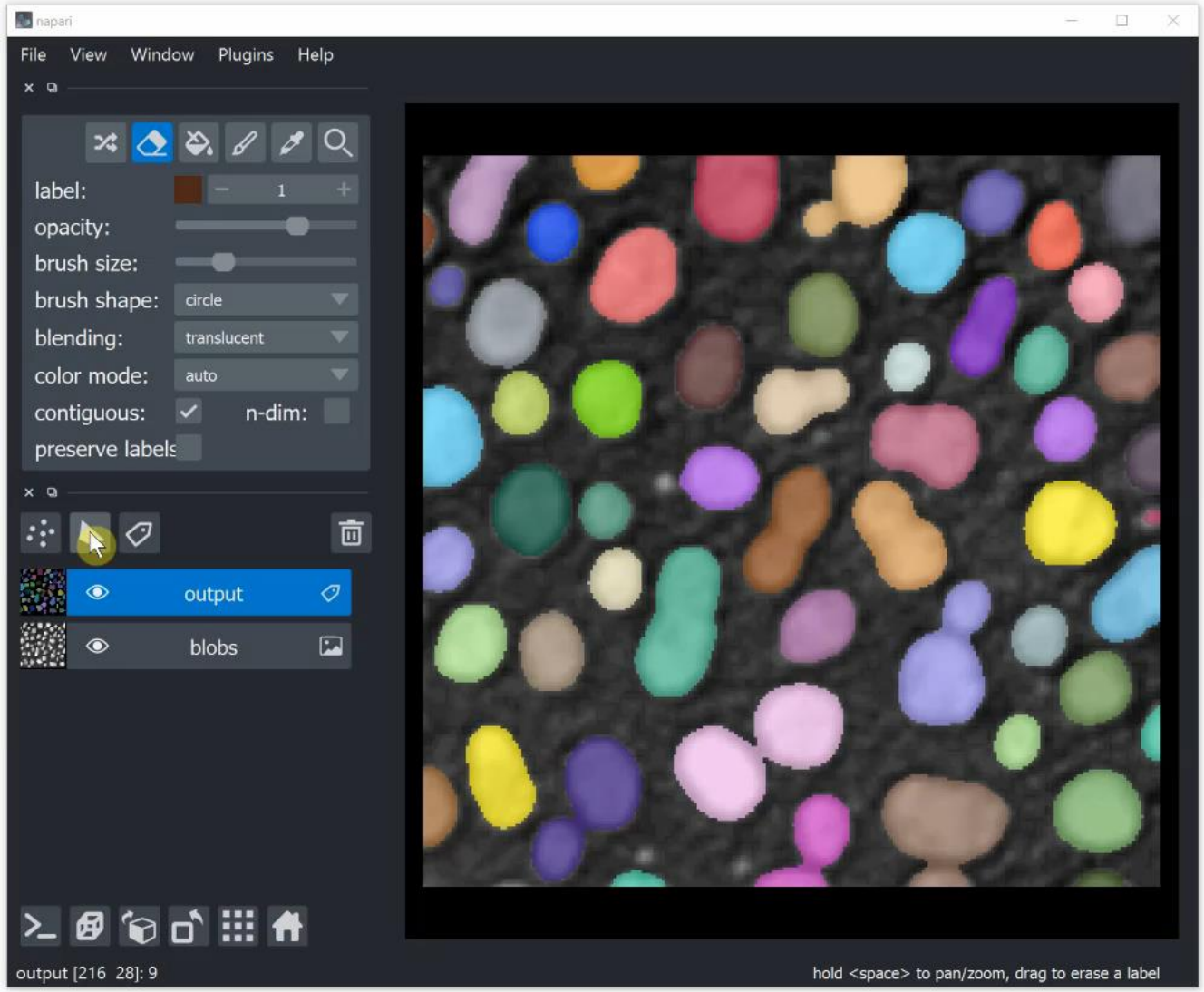


- Call it with
  `ipython --gui=qt napari_script.py`

pyclesperanto + napari

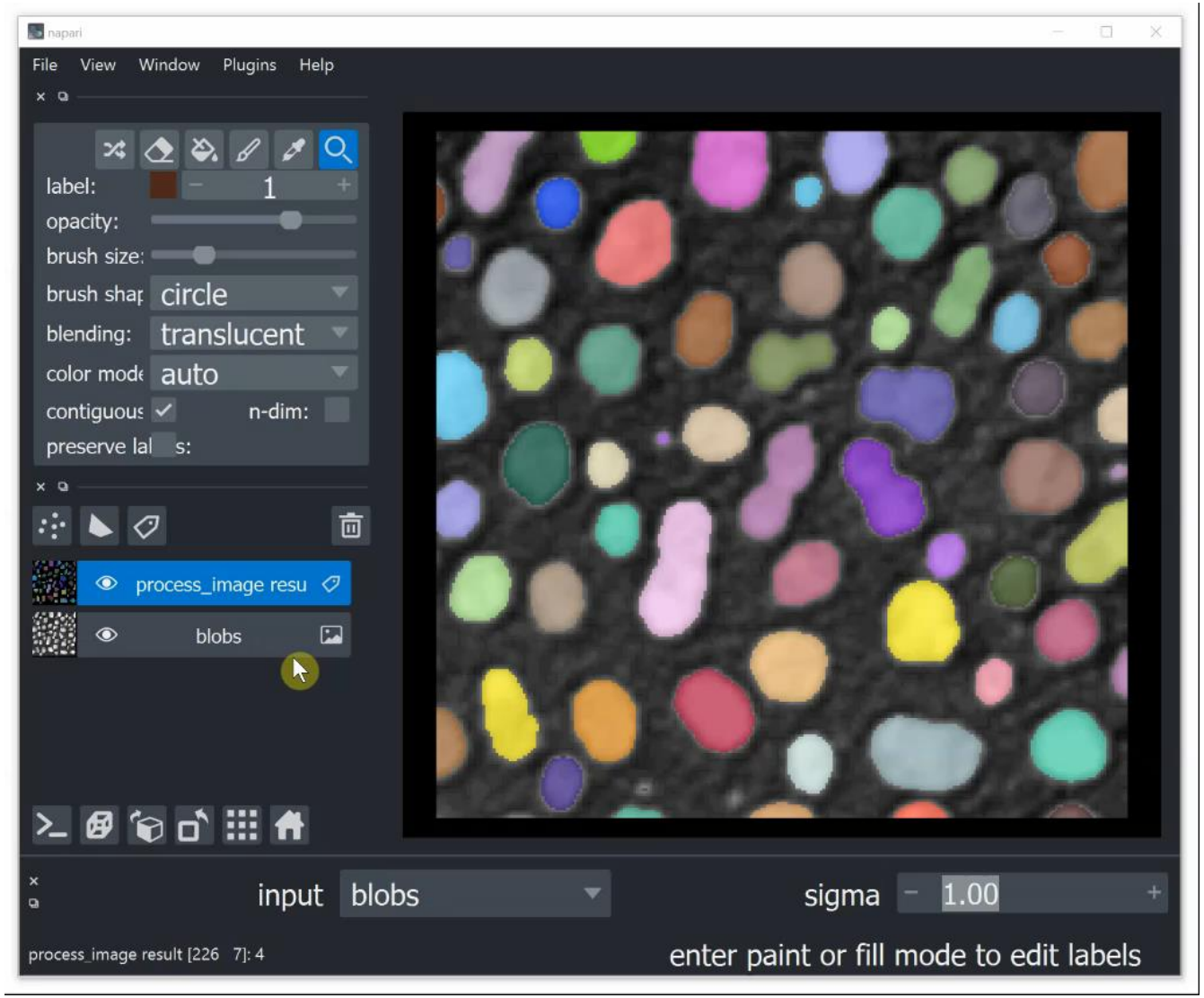
- Minimal napari + clesperanto example

```python
# load data
from skimage.io import imread
image = imread('https://samples.fiji.sc/blobs.png')

# start up napari
import napari
import pyclesperanto_prototype as cle
with napari.gui_qt():
    viewer = napari.Viewer()
    viewer.add_image(image, name='blobs')

    # push image to GPU
    input = cle.push_zyx(image)

    # process the mage
    sigma = 3
    blurred = cle.gaussian_blur(input, sigma_x=sigma, sigma_y=sigma)
    binary = cle.threshold_otsu(blurred)
    labels = cle.connected_components_labeling_box(binary)

    # pull result back
    output = cle.pull_zyx(labels)

    # add it to napari
    viewer.add_labels(output)
```



pyclesperanto + napari + magicgui

```python
# Inspired by
# https://github.com/pr4deepr/pyclesperanto_prototype/blob/master/napari_clij_widget.py
import napari
import pyclesperanto_prototype as cle
from magicgui import magicgui
from napari.layers import Image, Labels

@magicgui(auto_call=True)
def process_image(input: Image, sigma: float = 5) -> Image:
    if input:
        # push image to GPU
        input = cle.push_zyx(input.data)

        # process the mage
        blurred = cle.gaussian_blur(input, sigma_x=sigma, sigma_y=sigma)
        binary = cle.threshold_otsu(blurred)
        labels = cle.connected_components_labeling_box(binary)

        # pull result back
        output = cle.pull_zyx(labels)
        return output

# load data
from skimage.io import imread
image = imread('https://samples.fiji.sc/blobs.png')

# start up napari
with napari.gui_qt():
    viewer = napari.Viewer()
    viewer.add_image(image, name='blobs')

    # generate a Graphical User Interface from the function above magically
    gui = process_image.Gui()
    viewer.window.add_dock_widget(gui)
```



pyclesperanto + napari + dask

- “Lazy” / “Delayed” image processing: Process a frame as soon as the user wants to see it.

```python
def process_image(image):
    import time

    start_time = time.time()

    # push image to GPU memory and show it
    gpu_input = cle.push_zyx(image)
    # print(gpu_input)

    # gaussian blur
    sigma = 2.0
    gpu_blurred = cle.gaussian_blur(gpu_input, sigma_x=sigma, sigma_y=sigma, sigma_z=sigma)

    # detect maxima
    gpu_detected_maxima = cle.detect_maxima_box(gpu_blurred)
```

```python
# adapted from: https://github.com/tlambert03/napari-dask-example/blob/master/dask_napari.ipynb
import dask
import dask.array as da

# create dask stack of lazy image readers
lazy_process_image = dask.delayed(process_image)  # lazy reader
lazy_arrays = [lazy_process_image(timelapse[n]) for n in range(0, timelapse.shape[0])]
dask_arrays = [
    da.from_delayed(lazy_array, shape=timelapse[0].shape, dtype=timelapse[0].dtype)
    for lazy_array in lazy_arrays
]
# Stack into one large dask.array
dask_stack = da.stack(dask_arrays, axis=0)
dask_stack
```

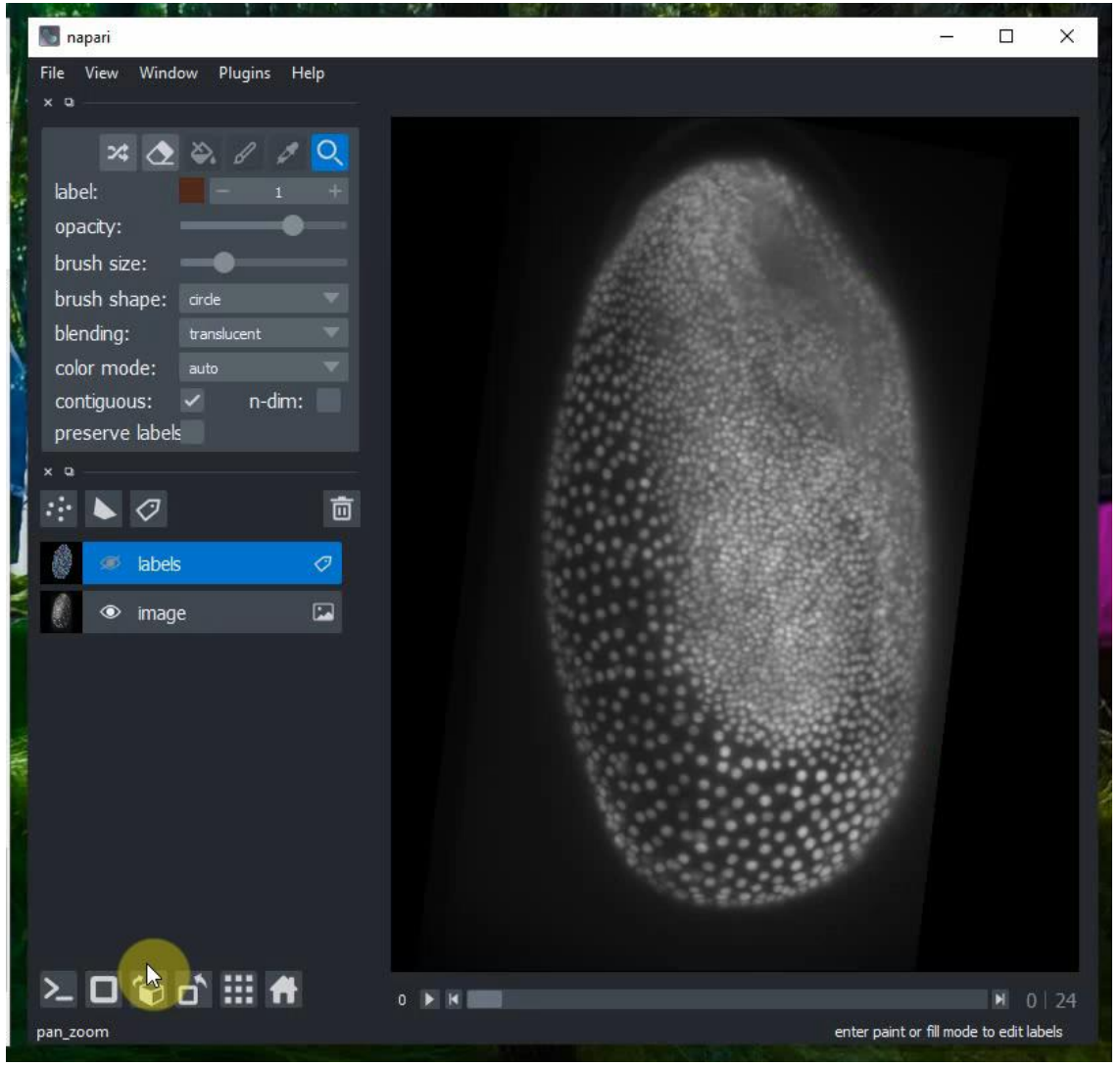
https://github.com/clEsperanto/pyclesperanto_prototype/blob/master/demo/napari_gui/napari_dask.ipynb

pyclesperanto + napari + dask

- “Lazy” / “Delayed” image processing: Process a frame as soon as the user wants to see it.

```python
def process_image(image):
    import time

    start_time = time.time()

    # push image to GPU memory and show it
    gpu_input = cle.push_zyx(image)
    # print(gpu_input)

    # gaussian blur
    sigma = 2.0
    gpu_blurred = cle.gaussian_blur(gpu_input, sigma_x=sigma, sigma_y=sigma, sigma_z=sigma)

    # detect maxima
    gpu_detected_maxima = cle.detect_maxima_box(gpu_blurred)
```

```python
# adapted from: https://github.com/tlambert03/napari-dask-example/blob/master/dask_napari.ipynb
import dask
import dask.array as da

# create dask stack of lazy image readers
lazy_process_image = dask.delayed(process_image)  # lazy reader
lazy_arrays = [lazy_process_image(timelapse[n]) for n in range(0, timelapse.shape[0])]
dask_arrays = [
    da.from_delayed(lazy_array, shape=timelapse[0].shape, dtype=timelapse[0].dtype)
    for lazy_array in lazy_arrays
]
# Stack into one large dask.array
dask_stack = da.stack(dask_arrays, axis=0)
dask_stack
```

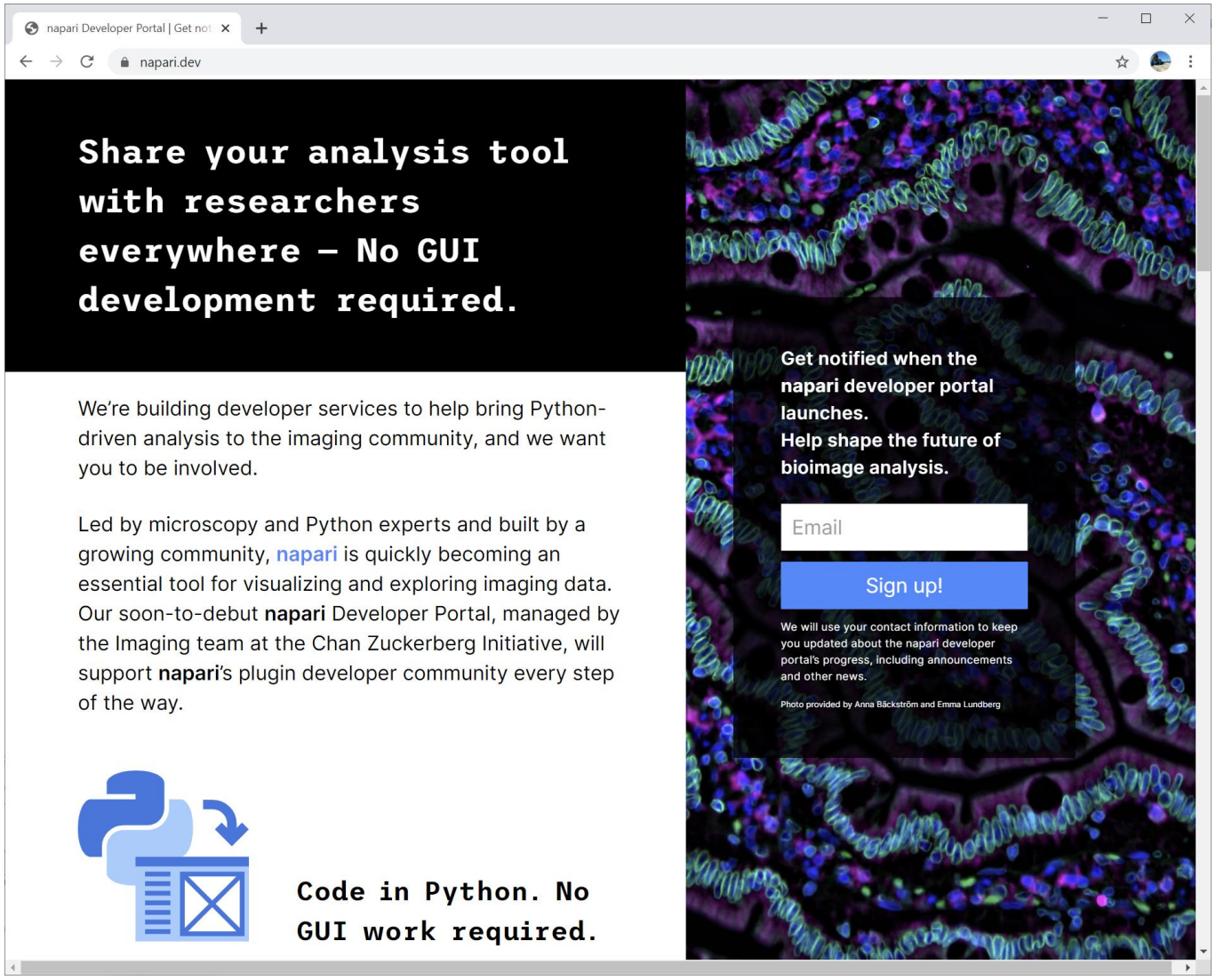


From napari: developer services coming soon!

New!

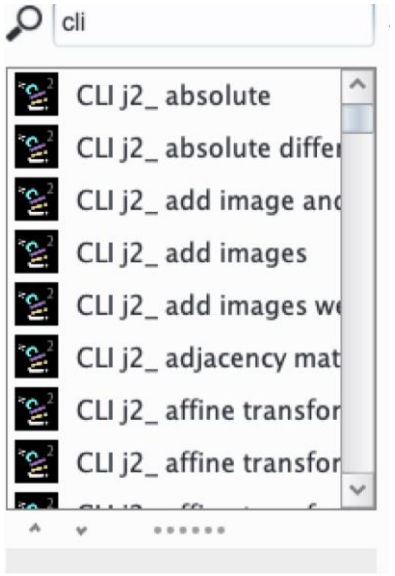
https://www.napari.dev

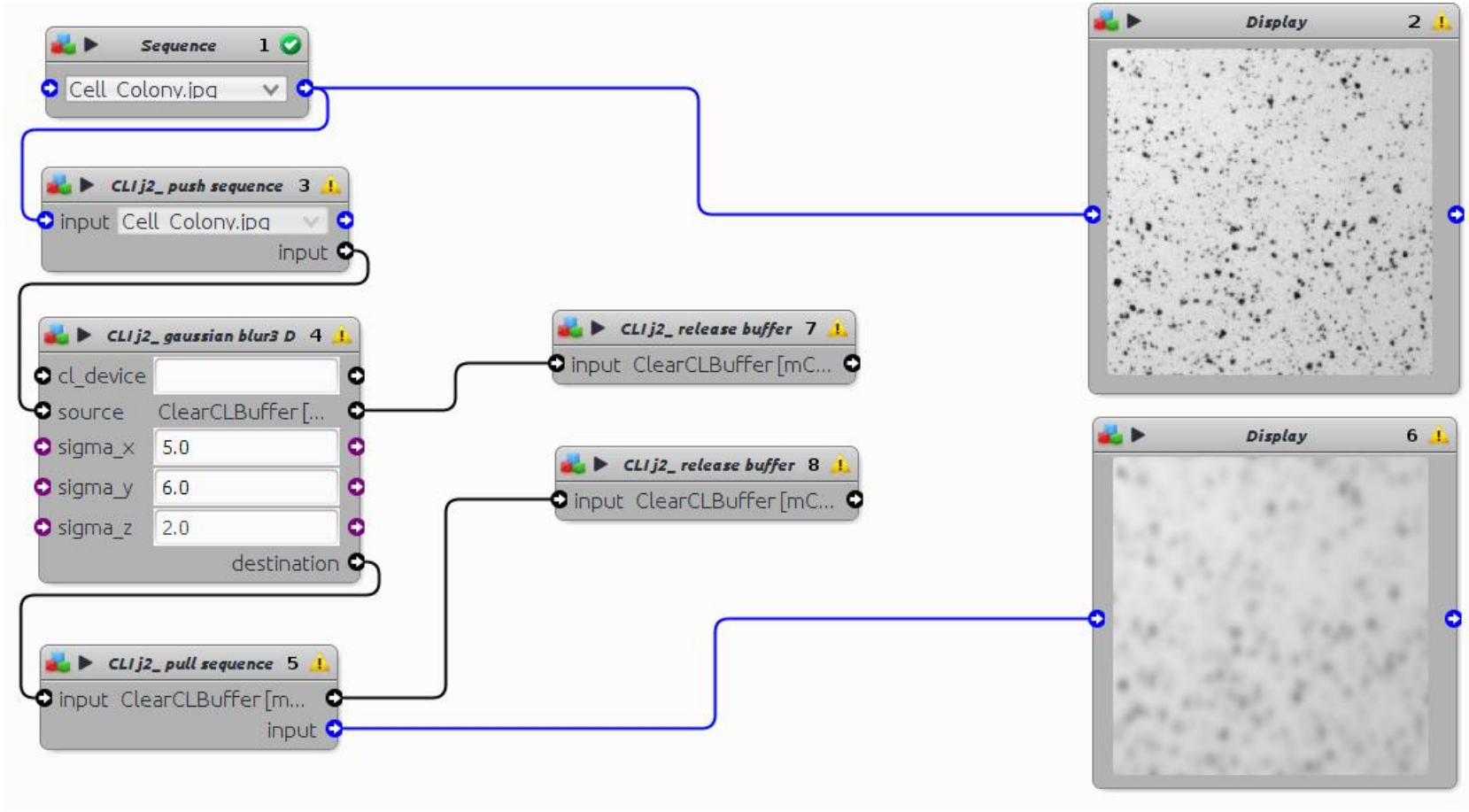
- **Mission**: help developers build their best Python plugin
- Will host services to support the easy **packaging, testing, integration and maintenance** of plugins
- help you **easily distribute** your plugin to users through a plugin store
- **Sign up today** to get early access notification and chat with the developer services team about your plugin!



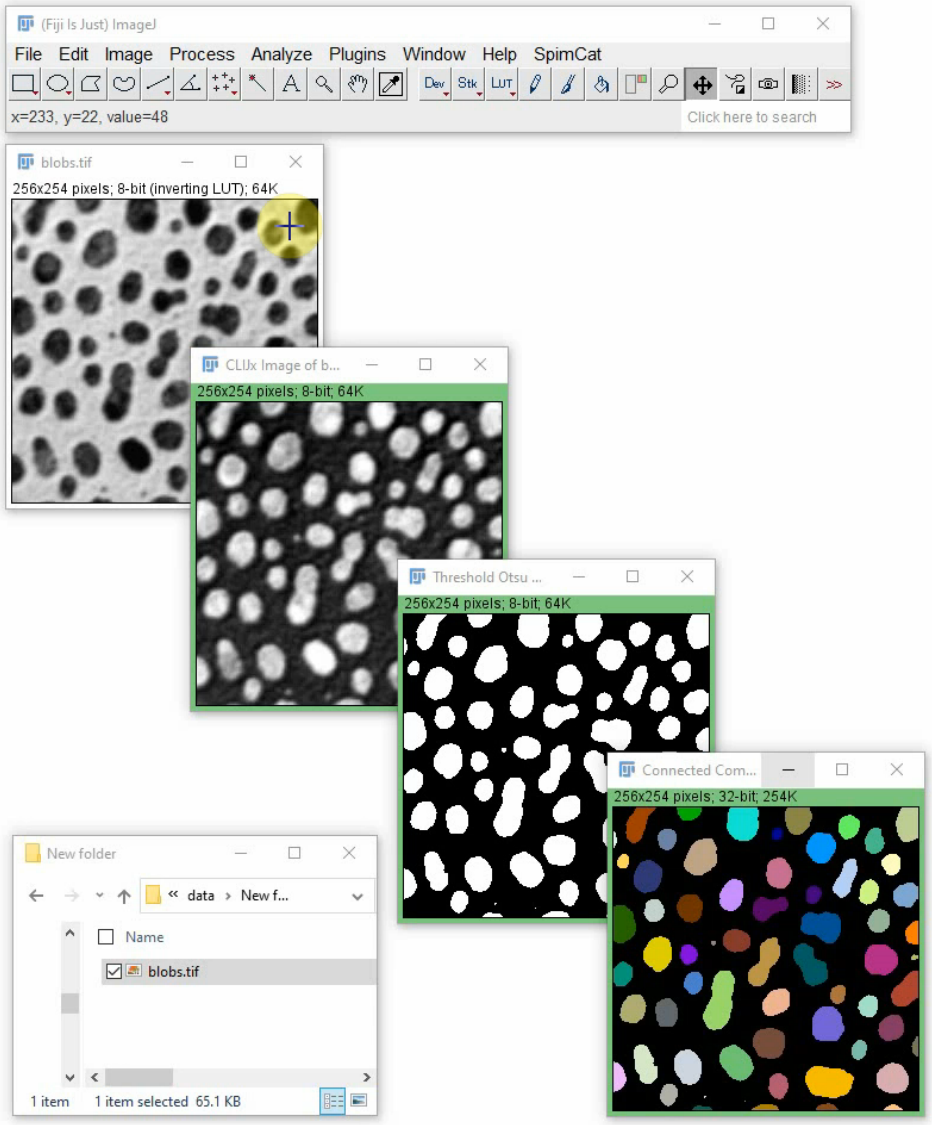
Icy Protocol generator

CLIcy in ICY protocols





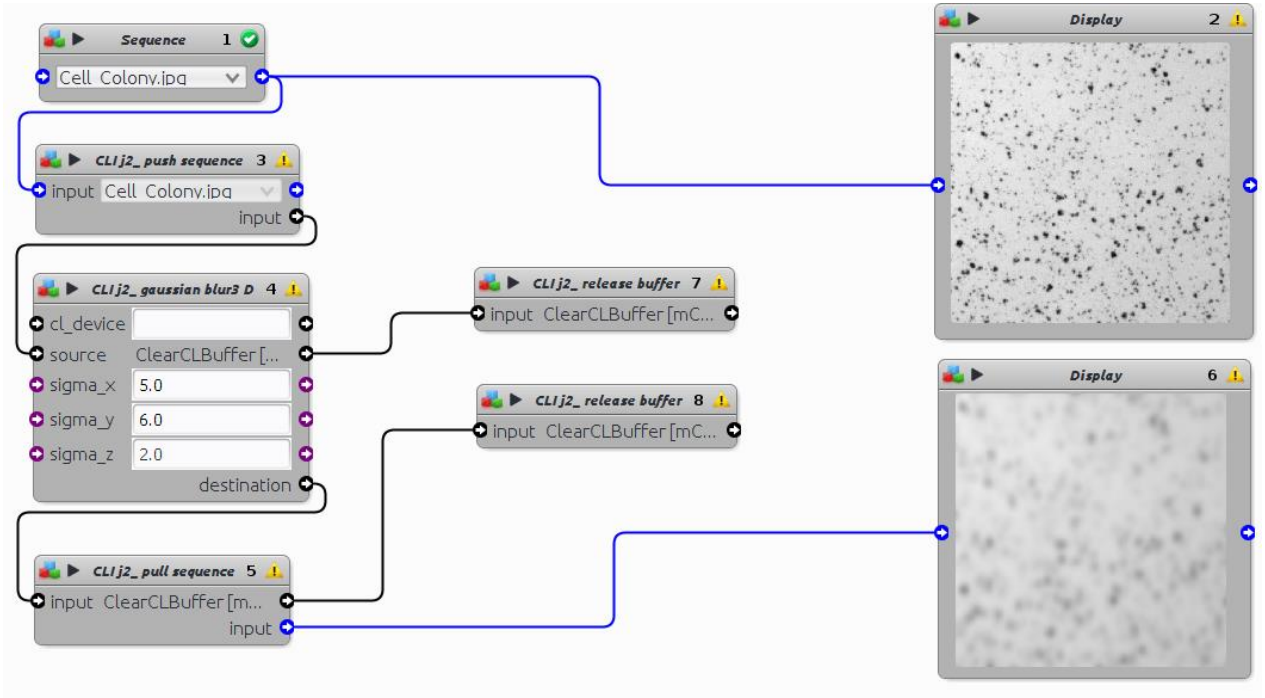
Export ICY Protocols from Fiji



- https://clij.github.io/assistant/installation#icy

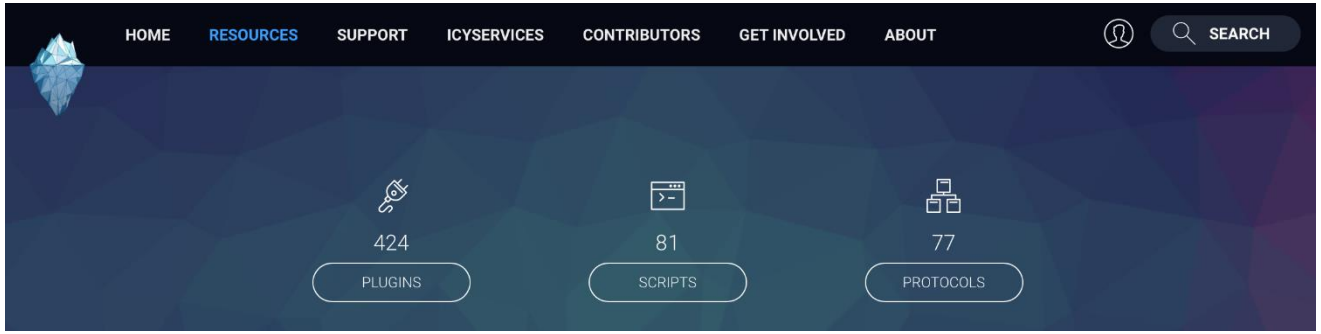
CLIcy in ICY protocols

- Protocols can be easily shared, and published in journals for reproducibility
- You can also upload them on the ICY website for reference
- You can then also find other example of protocols there!

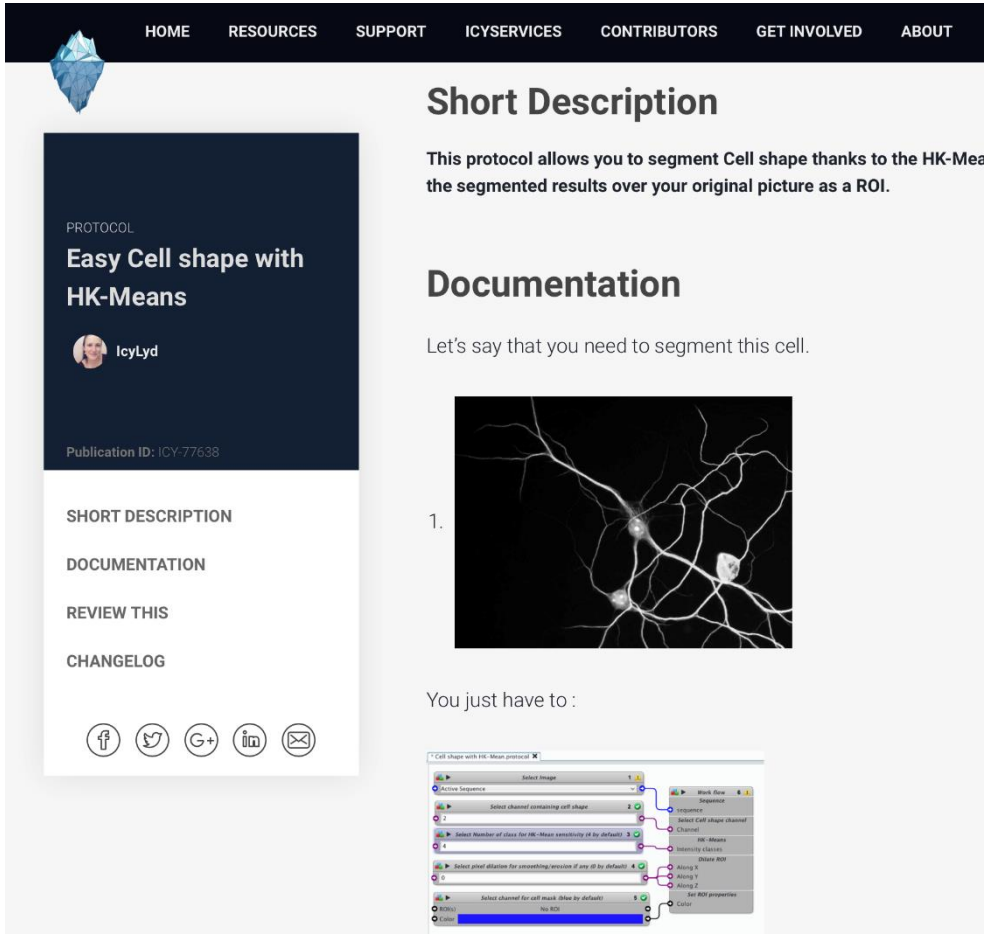


CLIcy in ICY protocols

http://icy.bioimageanalysis.org



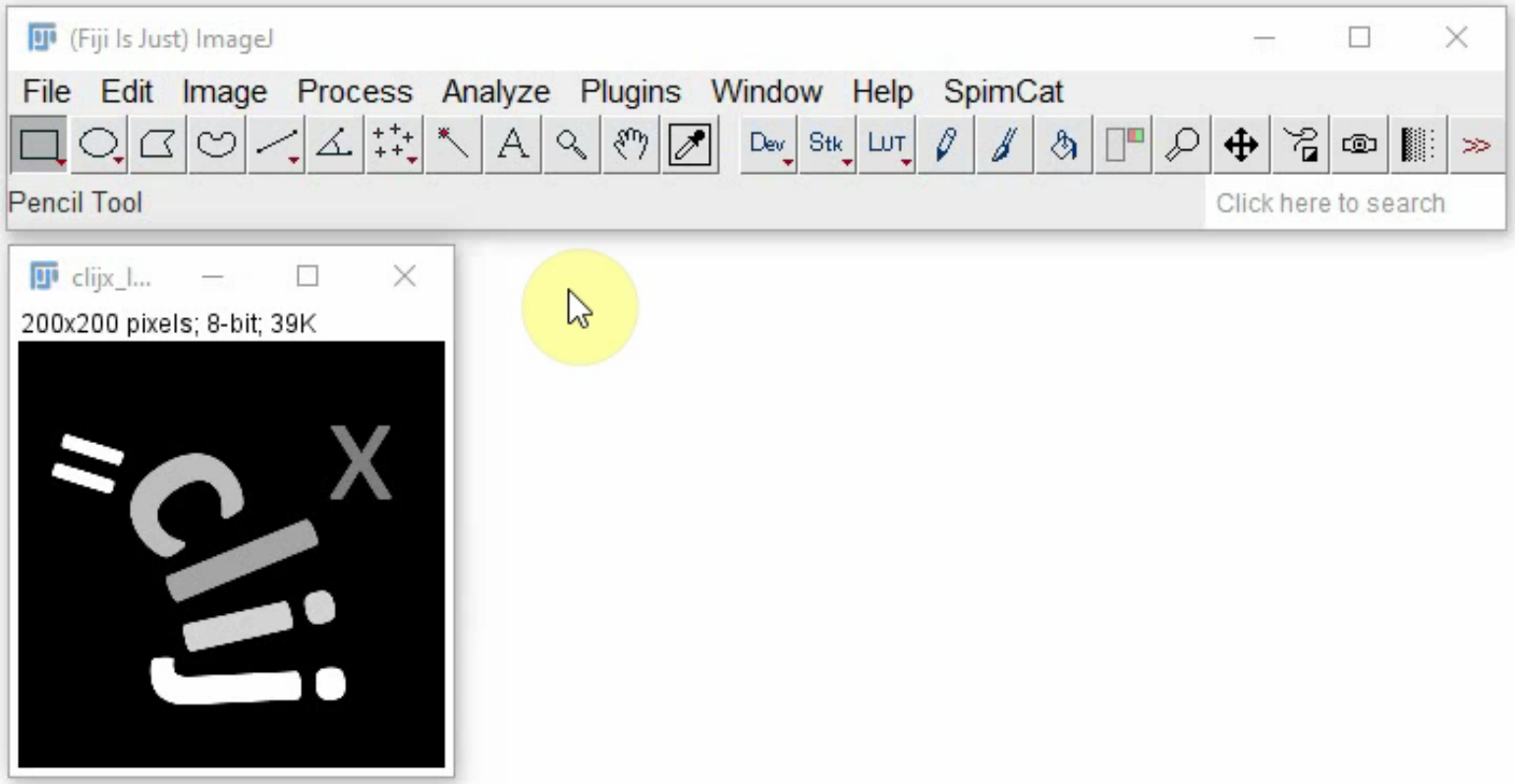
- Protocols can be easily shared, and published in journals for reproducibility
- You can also upload them on the ICY website for reference
- You can then also find other example of protocols there!



Fiji Plugin generator

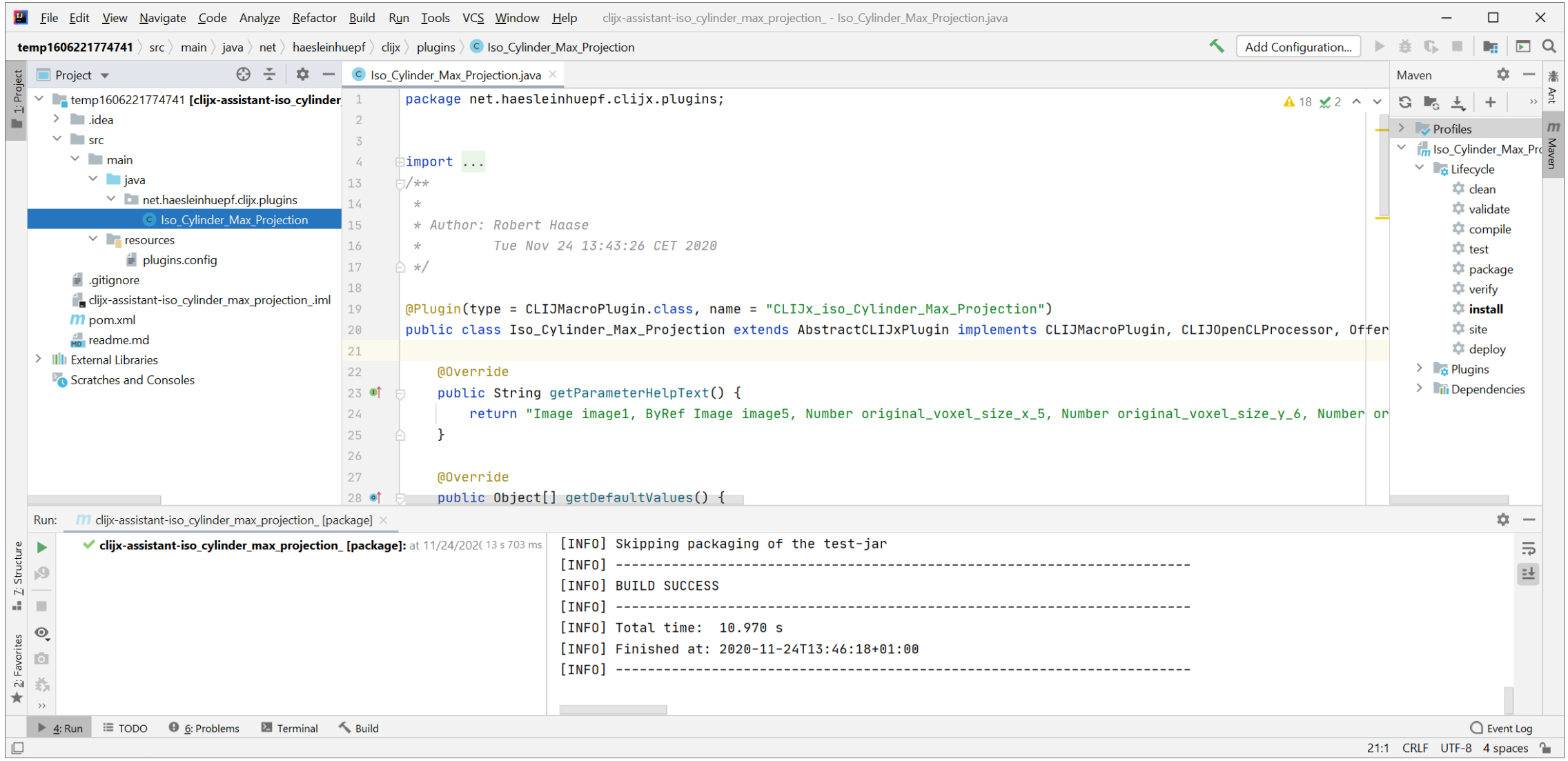
Fiji plugin generator

- Generate Fiji plugins using Maven https://clij.github.io/assistant/installation#maven



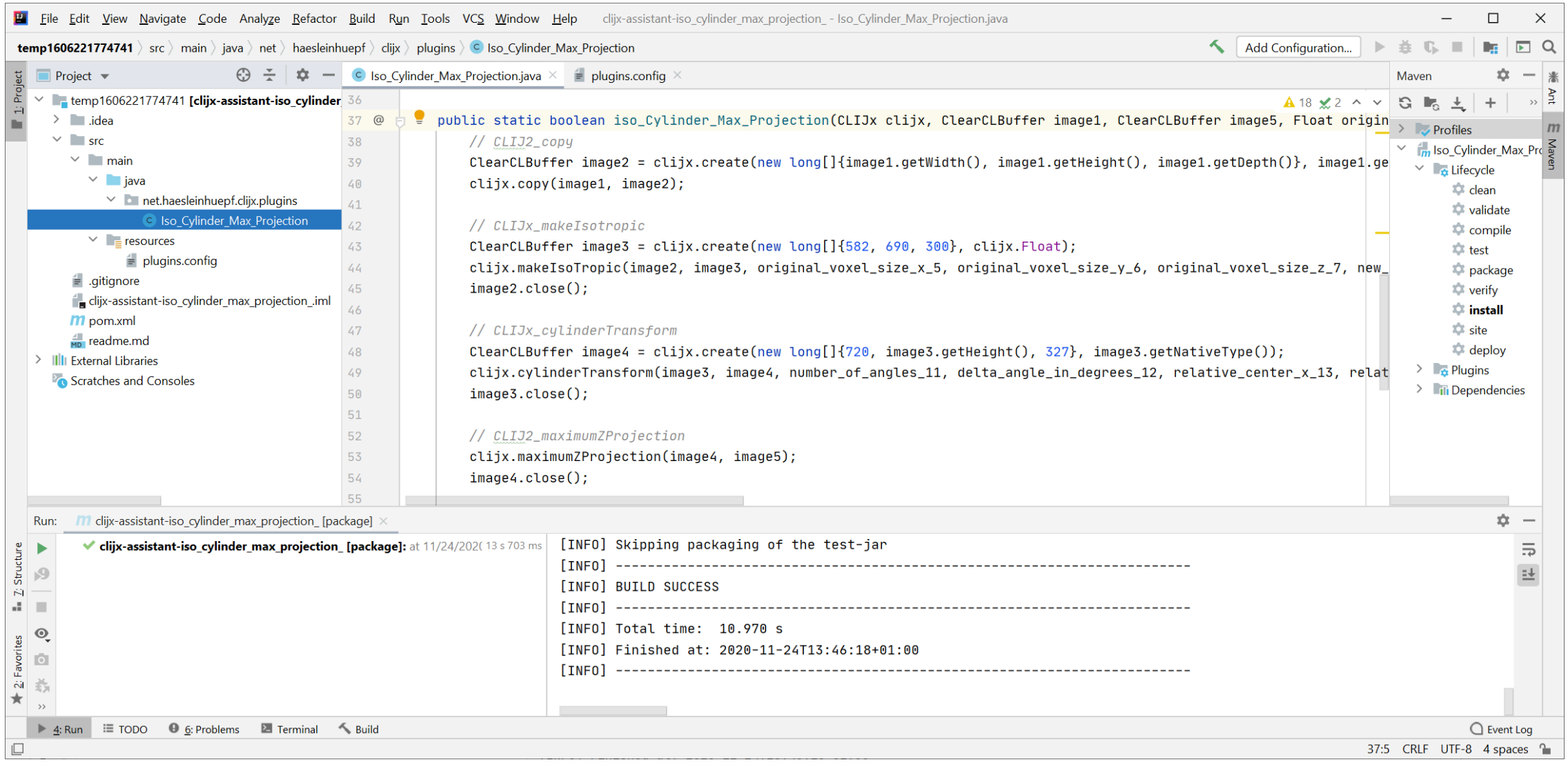
Fiji plugin generator

- You can explore the generated folder: The only Java file contains your algorithm



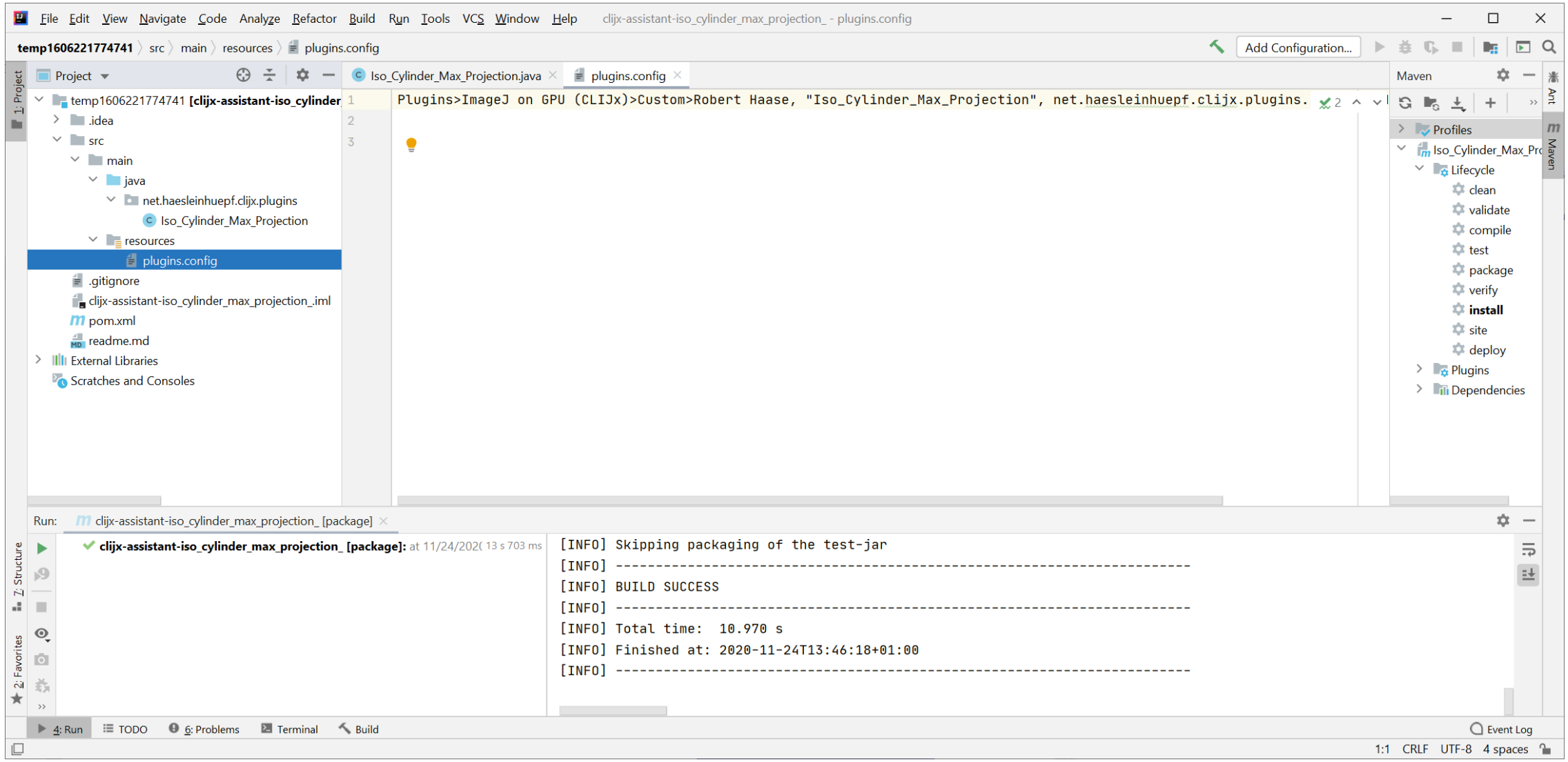
Fiji plugin generator

- You can explore the generated folder: The only Java file contains your algorithm



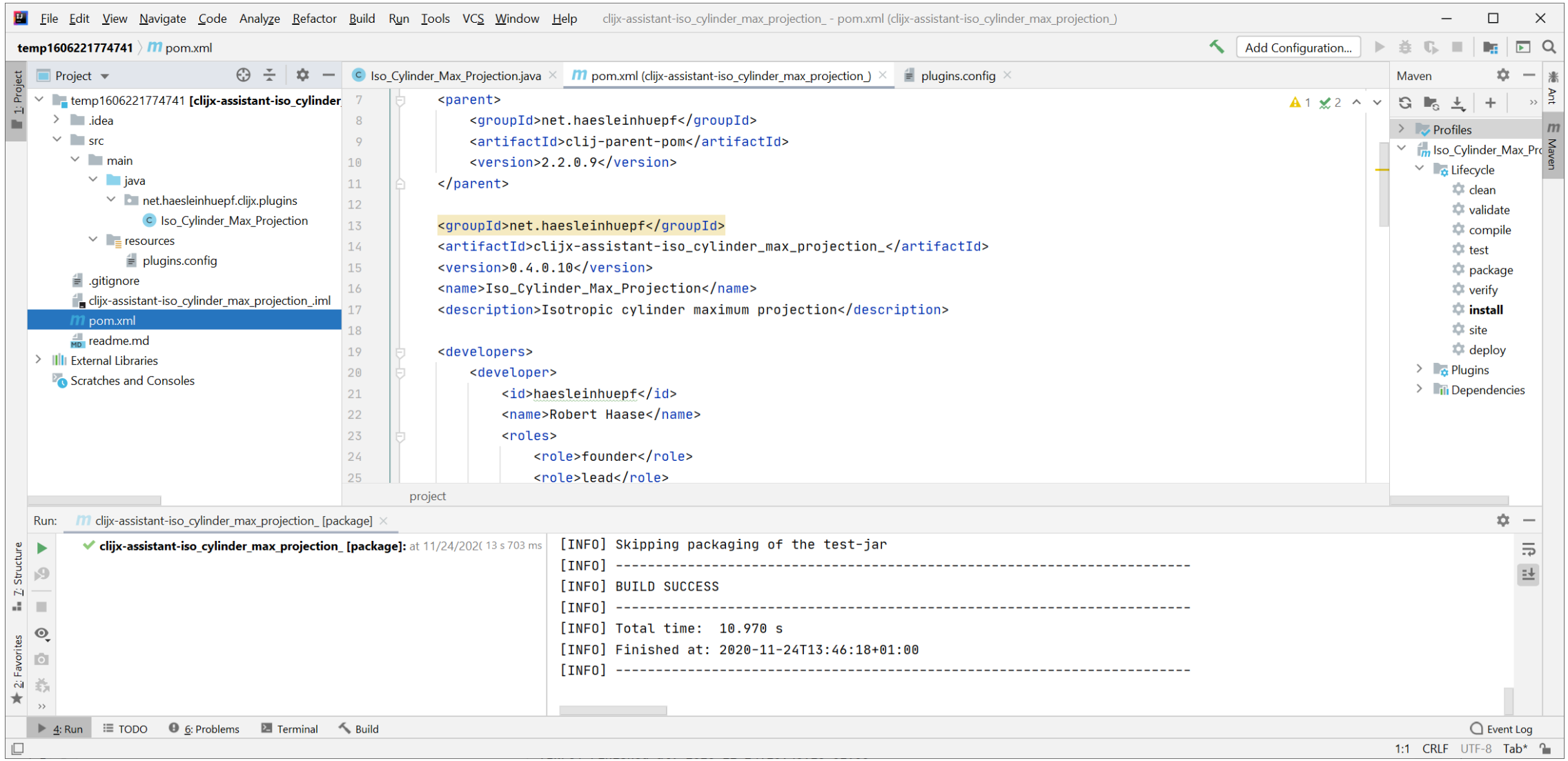
Fiji plugin generator

- The plugins.config file contains the menu path where your plugin can be found in ImageJ



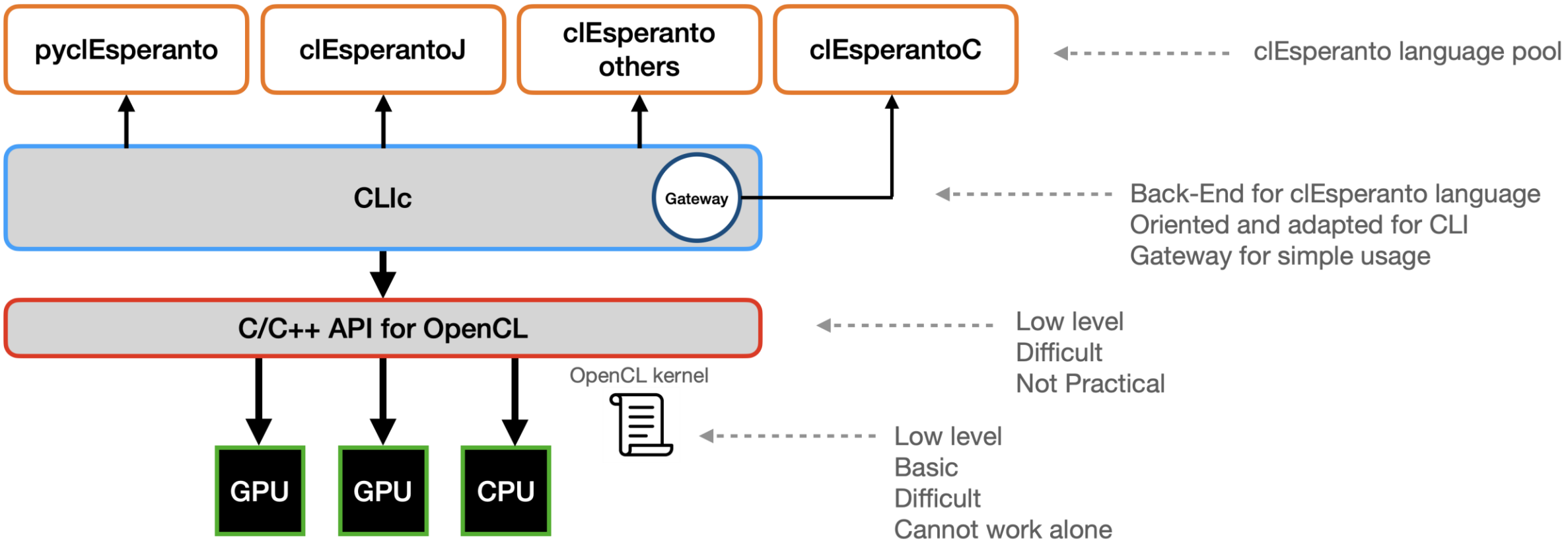
Fiji plugin generator

- The pom.xml file contains project information



CLIc: One language to (try to) rule them all

CLIc: One language to (try to) rule them all



CLIc: Back-end API and clEsperantoC

API / Back-End

Kernel Classes

gaussian filter

mean filter

maximum project

Object Classes

buffer

scalar

image3d

image2d

Manager Classes

command queue

device

gpu

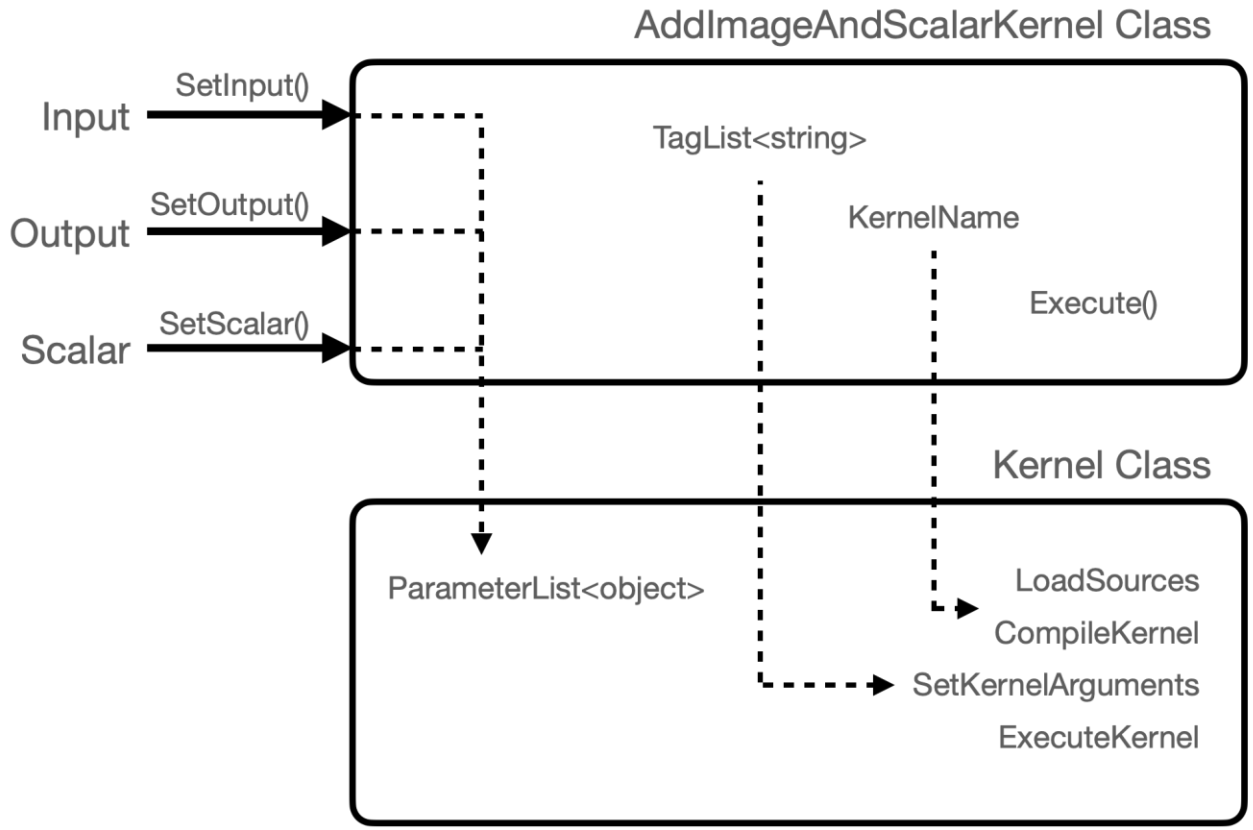
context

CLE

clEsperantoC script

```
// Initialise GPU information.
cle::GPU gpu;
cle::CLE cle(gpu);

// Push image
Image<unsigned char> raw_img (raw, 256, 254, "unsigned char");
cle::Buffer img1 = cle.Push<unsigned char>(raw_img);

// Create buffer
std::array<unsigned int, 2> dimensions = {256, 254};
cle::Buffer img2 = cle.Create<unsigned char>(dimensions.data(),
                                             "unsigned char");

// Smaller Or Equal Constant
cle.smaller_or_equal_constant(img1, img2, 110);

// Pull output into container
Image<unsigned char> res = gpu.Pull<unsigned char>(img2);
```

CLIc: Kernel class design



- Clear and simple Set methods name
- Manage parameter name and order
  - Need to correspond to .cl implementation
- Kernel class manage dimensionality (2d/3d)
- Provide an Execute() method
  - Allow mini-pipeline of kernels
  - Allow complexe operations

- Manage input size and type
- Compilation of kernel and program
- Connect parameter to kernel
- Set operation to GPU queue
- Do the hard lifting behind the curtains

User visibility through the gateway:

```
cle.AddImageAndScalar(Buffer _input, Buffer _output, float _scalar);
```

CLIc: Kernel class design

header

```
namespace cle
{

class AddImageAndScalarKernel : public Kernel
{
private:

    void DefineDimensionality();

public:
    AddImageAndScalarKernel(GPU& gpu) : Kernel(gpu)
    {
        kernelName = "add_image_and_scalar";  // kernel name
        tagList = {"src", "dst", "scalar"};   // parameter tag/order
    }

    void SetInput(Object&);
    void SetOutput(Object&);
    void SetScalar(float);
    void Execute();

    ~AddImageAndScalarKernel() = default;
};
```

source

```
void AddImageAndScalarKernel::SetInput(Object& x)
{
    this->AddObject(&x, "src");
}

void AddImageAndScalarKernel::SetOutput(Object& x)
{
    this->AddObject(&x, "dst");
}

void AddImageAndScalarKernel::SetScalar(float x)
{
    Float* val = new Float(x);   // convert float into generic object
    this->AddObject(val, "scalar");
}

void AddImageAndScalarKernel::Execute()
{
    DefineDimensionality();      // manage 2d or 3d data
    CompileKernel();             // compile .cl code into a gpu program
    AddArgumentsToKernel();      // link parameters to the compiled kernel
    DefineRangeKernel();         // set execution range
}
```

clEsperantoC: What every C++ script must have

Initialise GPU

```
// Initialise GPU information.
cle::GPU gpu;
cle::CLE cle(gpu);
```

clEsperantoC: What every C++ script must have

Initialise GPU

```
// Initialise GPU information.
cle::GPU gpu;
cle::CLE cle(gpu);
```

Use CLIjx-Assistant script generator!
Or trust the auto-completion of your C++ IDE

```
cle.
// p
Imag
// V
floa
for
{
}
if (
{
```

- Absolute — void cle::CLE::Absolute(cle::Buffer &…
- AddImageAndScalar
- Create
- GetGPU
- MaximumOfAllPixels
- MaximumXProjection
- MaximumYProjection
- MaximumZProjection
- Mean2DSphere
- Pull
- Push
- SmallerOrEqualConstant

```
// Create buffer
std::array<unsigned
cle::Buffer img2 =
```

pull

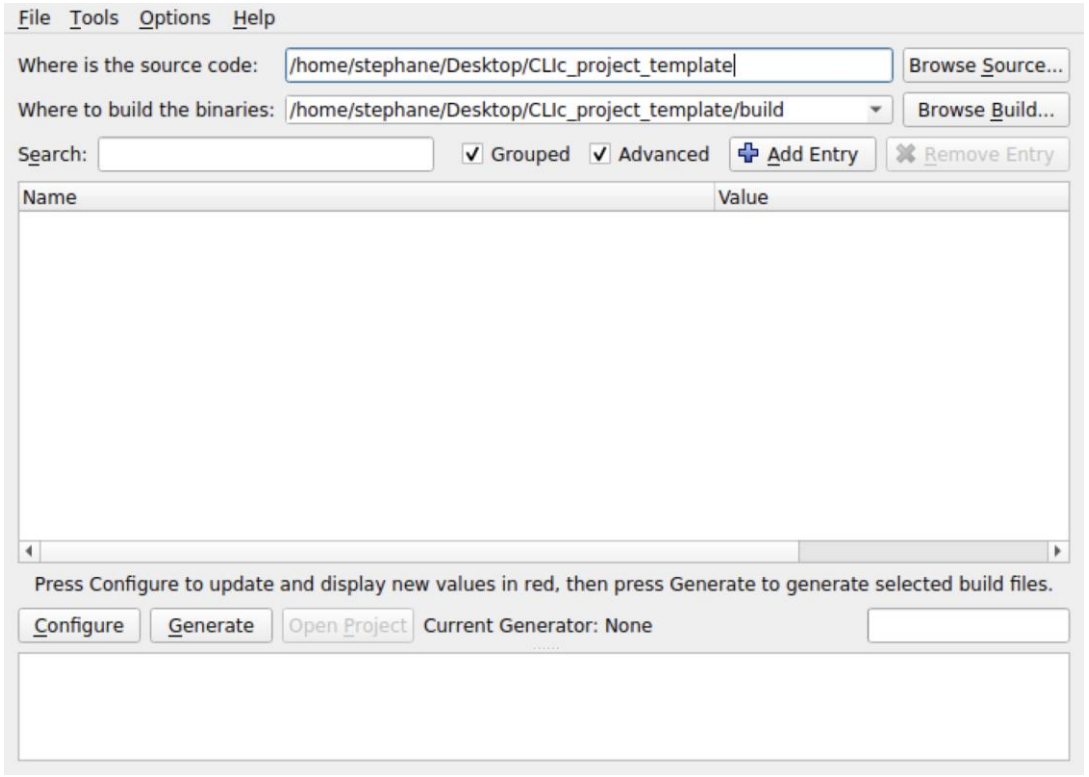
```
Image<unsigned char> res = gpu.Pull<unsigned char>(img2);
```

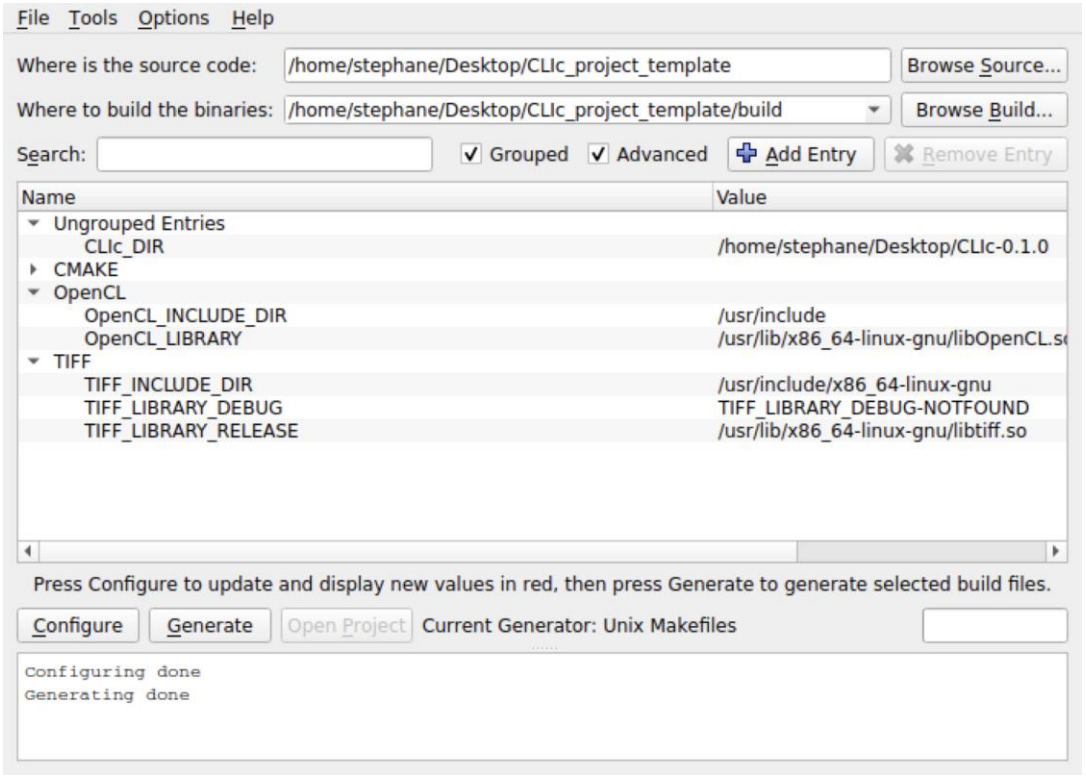
Practical links
Code test example : https://github.com/clEsperanto/CLIc_prototype/test
Pre-made C++ empty project : https://github.com/StRigaud/CLIc_project_template

CLIc: Installation and Compilation

<u>Requirement:</u> OpenCL, LibTiff, Cmake

And let CMake do all the hard work of project creation and configuration!



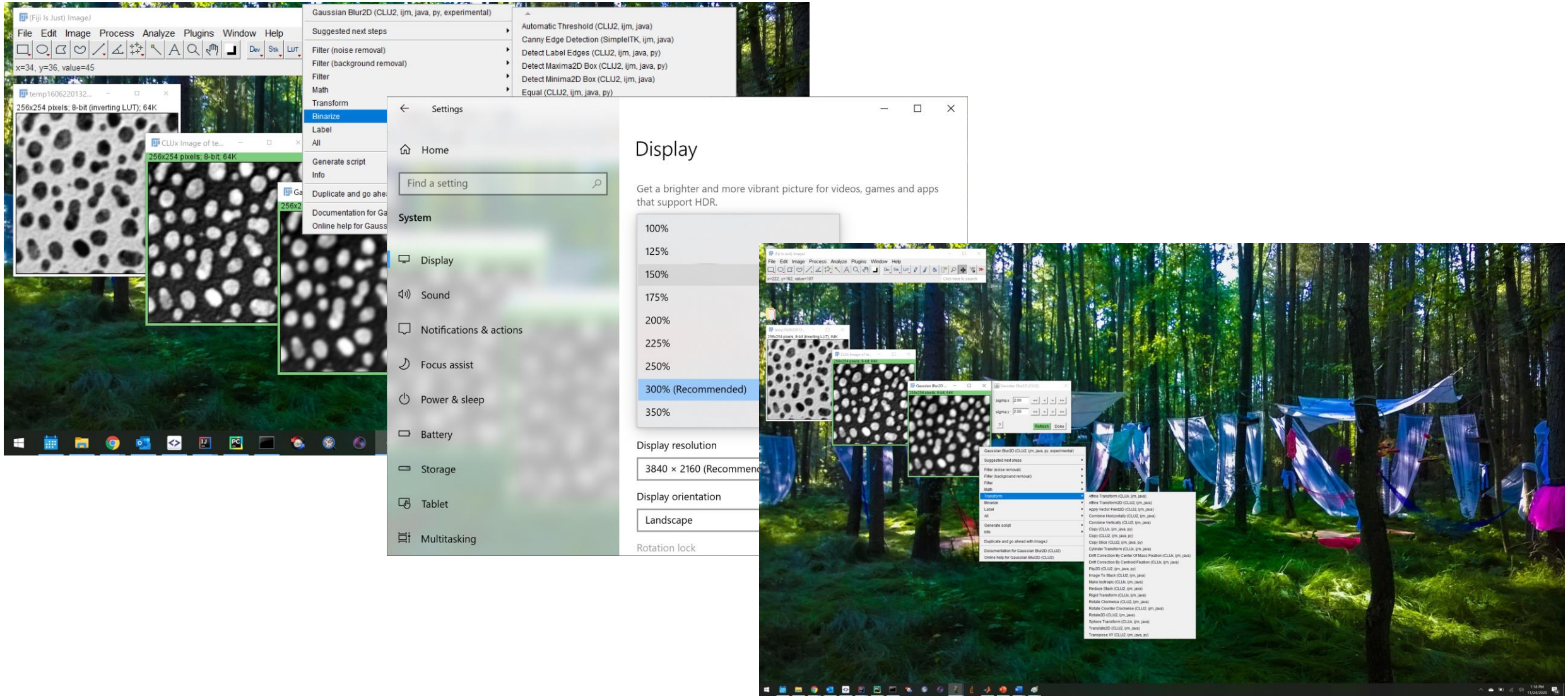


Guideline and troubleshooting here → https://github.com/clEsperanto/CLIc_prototype

Troubleshooting

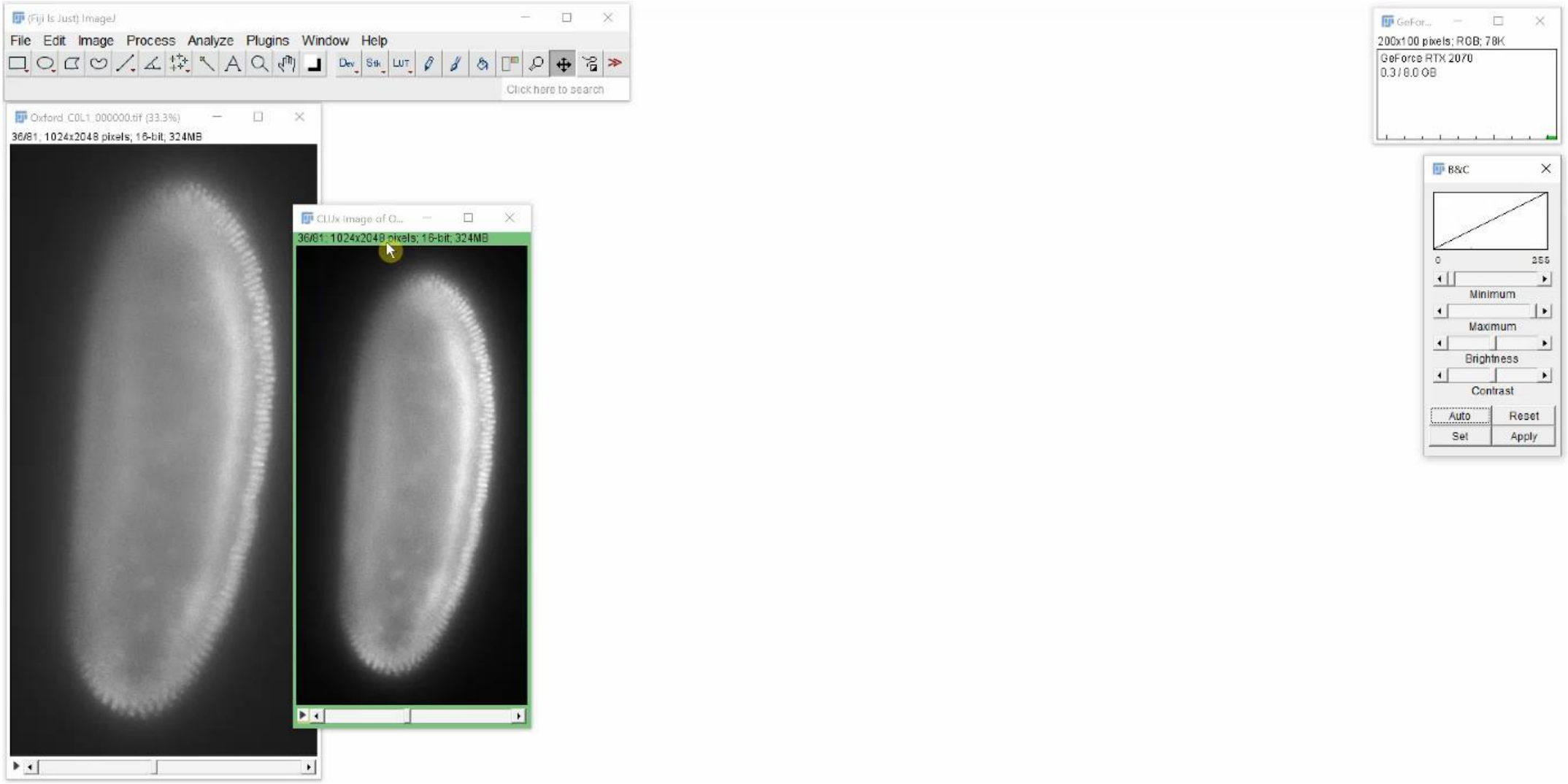
Hints & troubleshooting

- Windows screen full of windows? Check display settings!



Hints & troubleshooting

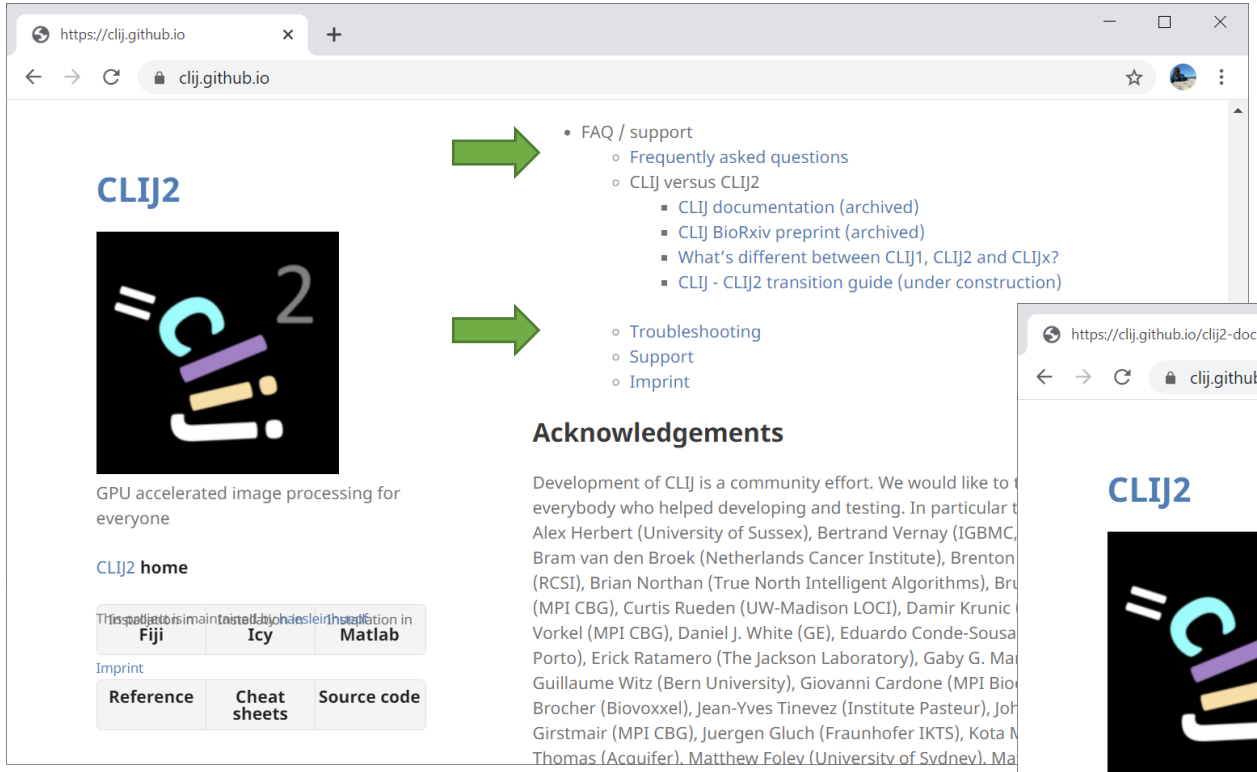
- Out of memory? Wrap a chain of operations in a Fiji plugin!

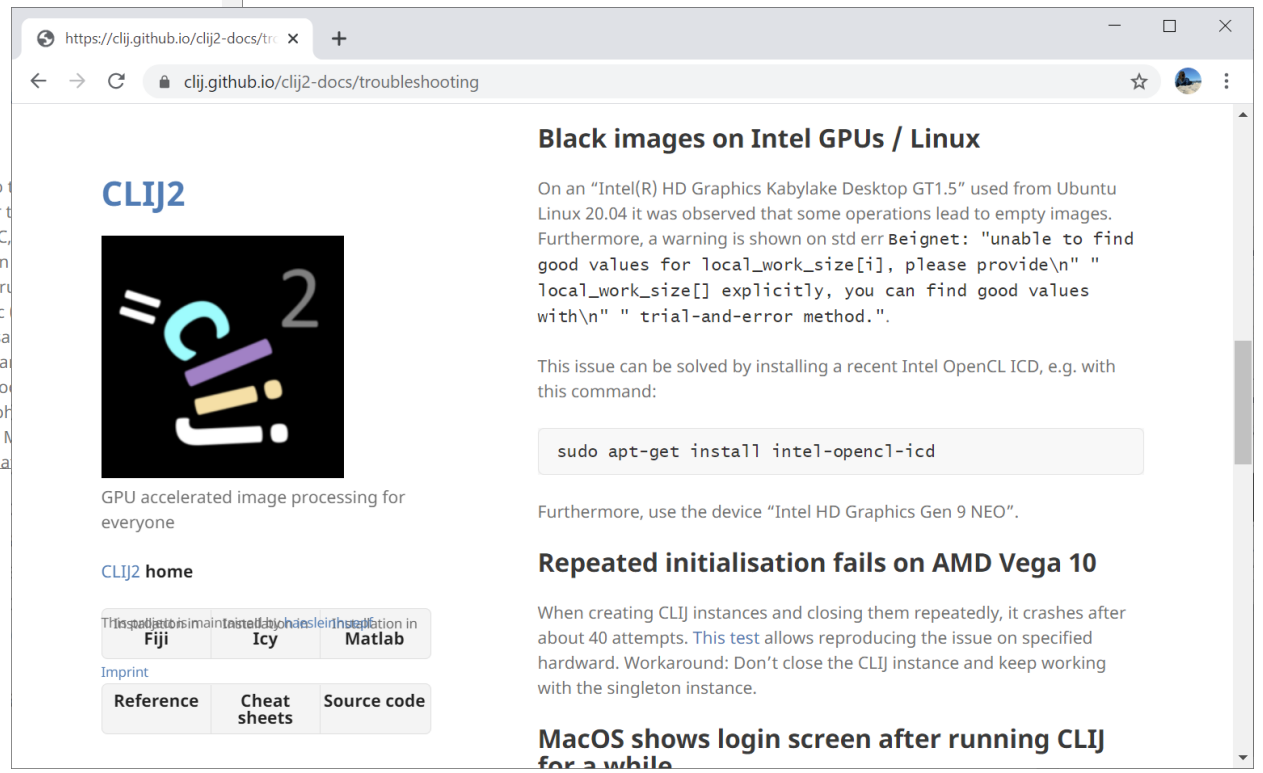


Trouble shooting

- Check the website!

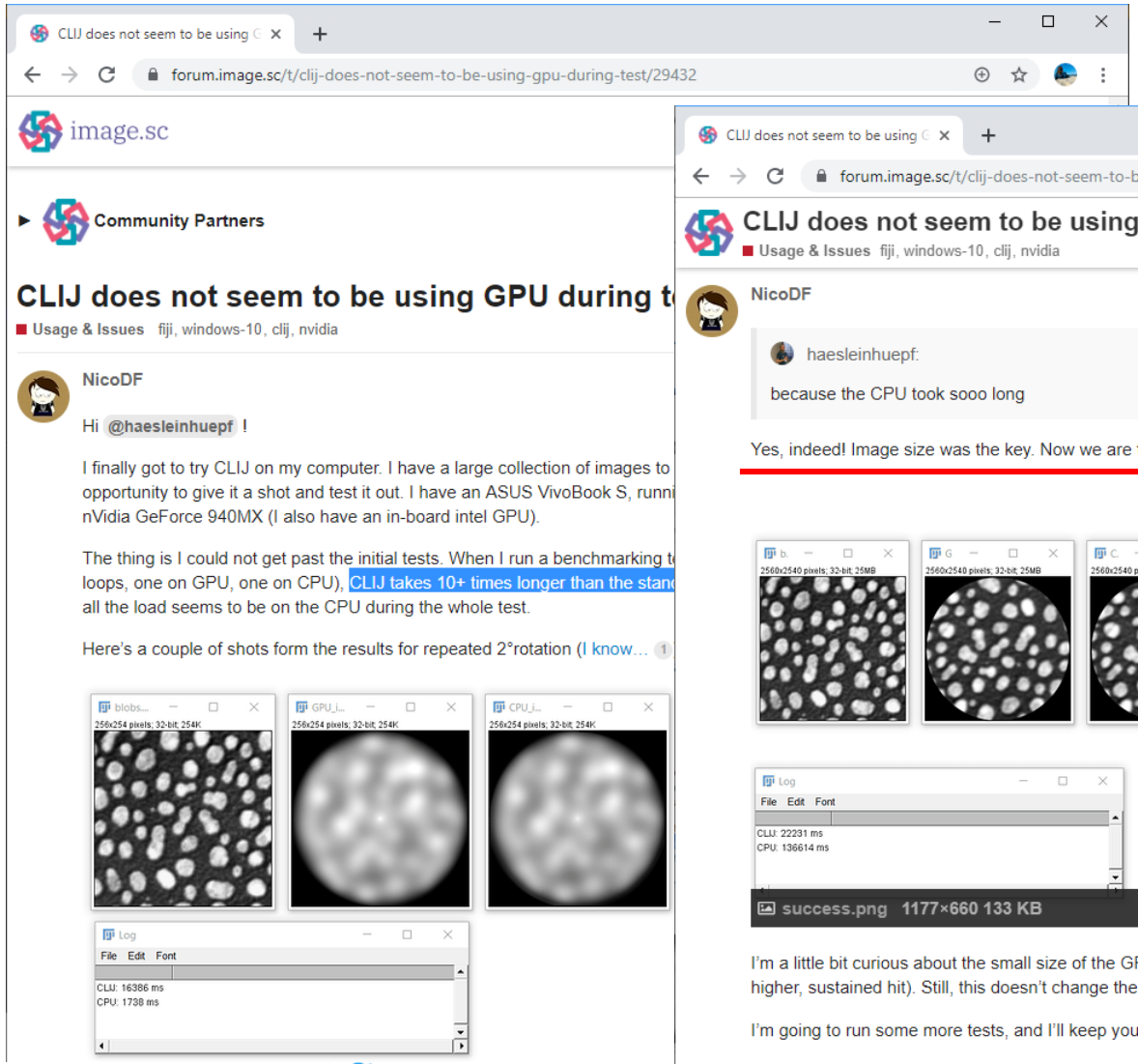
https://clij.github.io/clij2-docs/troubleshooting

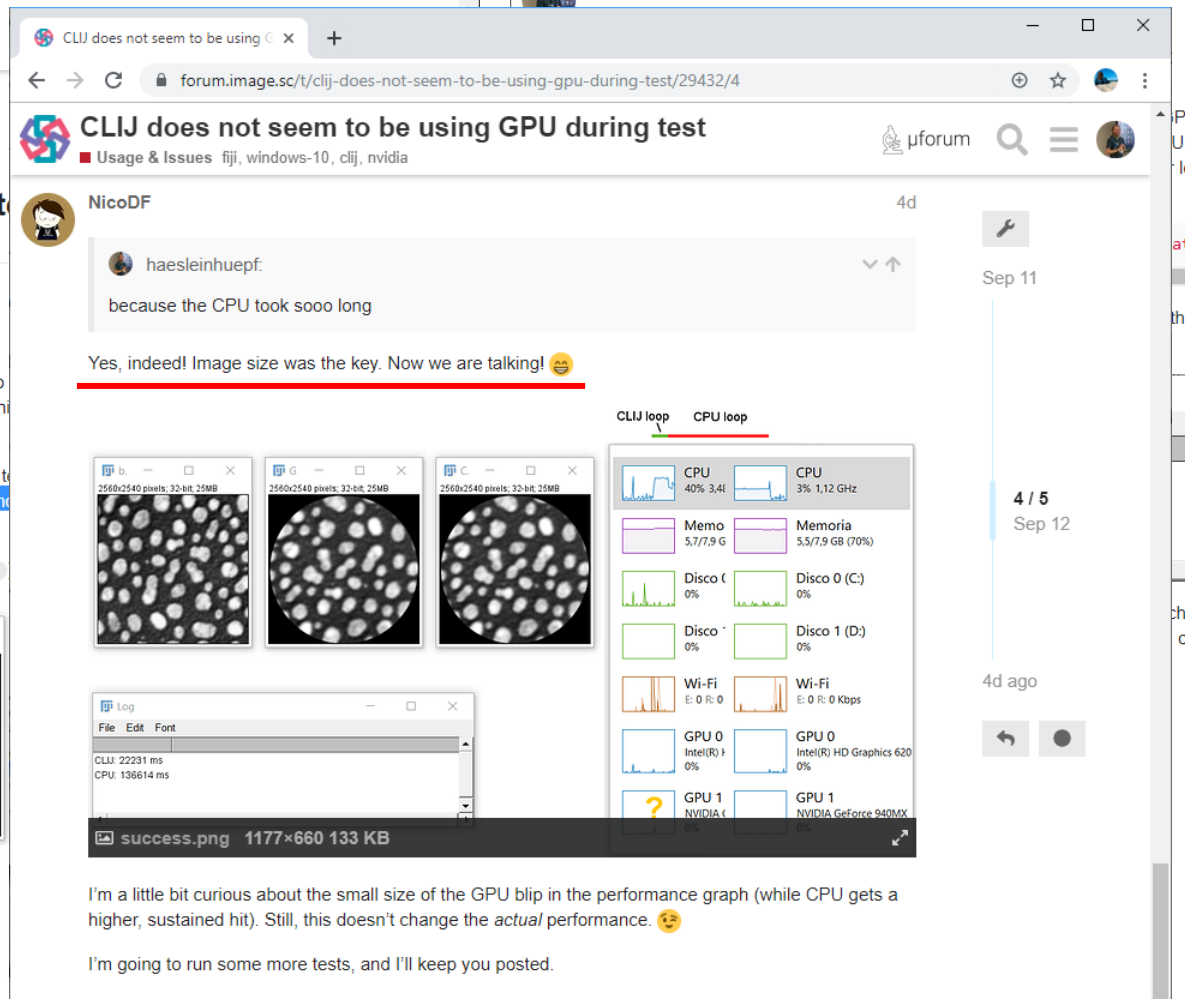




Support: http://image.sc

- Ask the online!





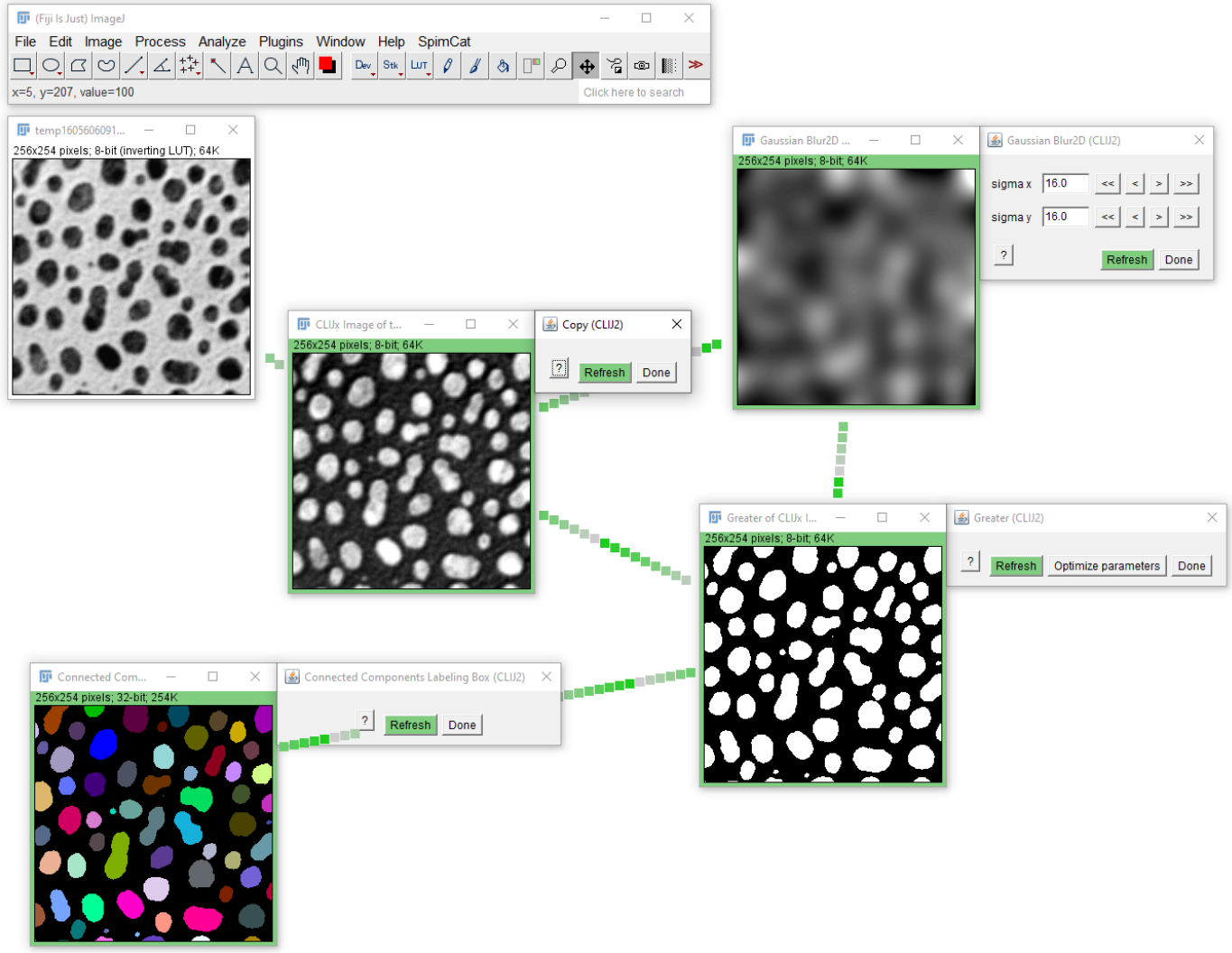
@haesleinhuepf @StRigaud

elia

Exercises

Exercise 5: Interactive workflow design

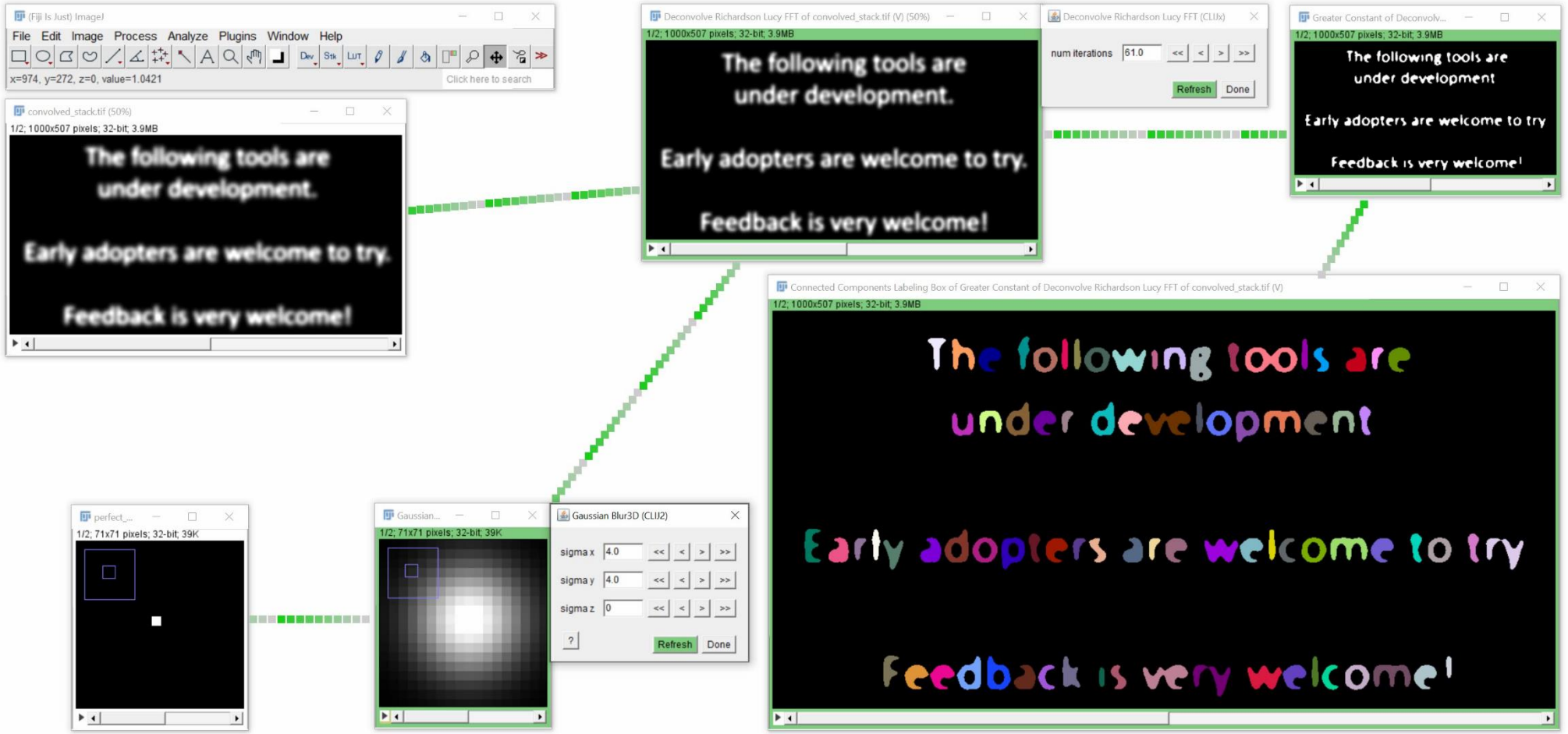
- Design a workflow for segmenting blobs.gif
- Export the workflow as ImageJ Macro script for and Fiji Jython.
- Optional: Export the workflow as Icy Javascript, as Icy protocol and for QuPath as groovy script. Feel free to generate a Fiji plugin.
- Furthermore, if you only used operations, which are "py" compatbile, export a clEsperanto-based Jupyter notebook and a Python script that uses Napari.



https://github.com/clEsperanto/i2k2020_tutorial_clij_clesperanto

Exercie 6: Deconvolution + Segmentation

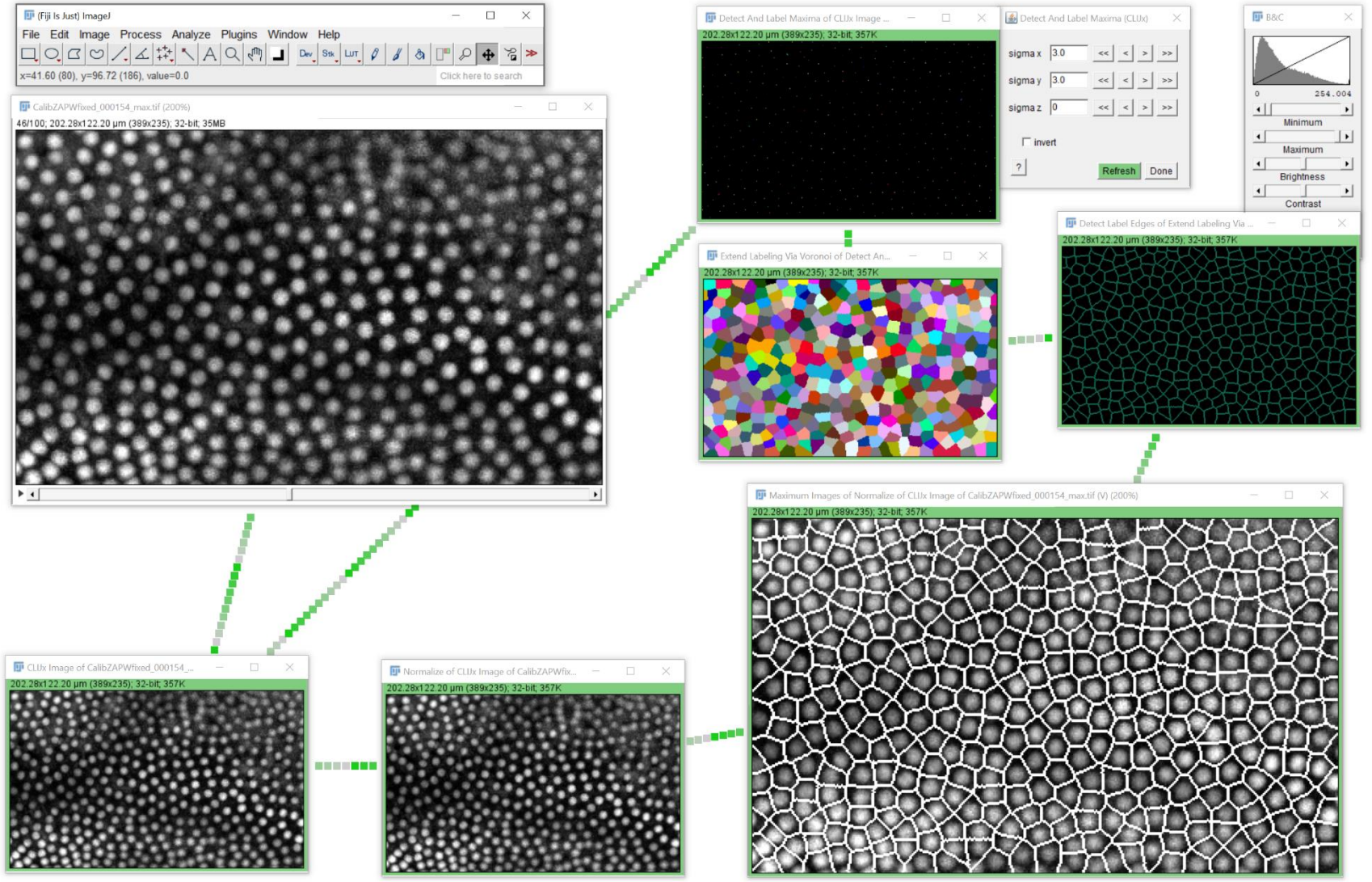
- Reproduce the deconvolution + segmentation workflow from the beginning of this session



Exercise 7: Segmentation in time lapse data

- Setup a workflow for estimating cell borders between nuclei

- Export an ImageJ Macro and run it.
- Note: This exercise will teach you how to deal with errors in generated macro code and how to build in a for loop to process frames individually.



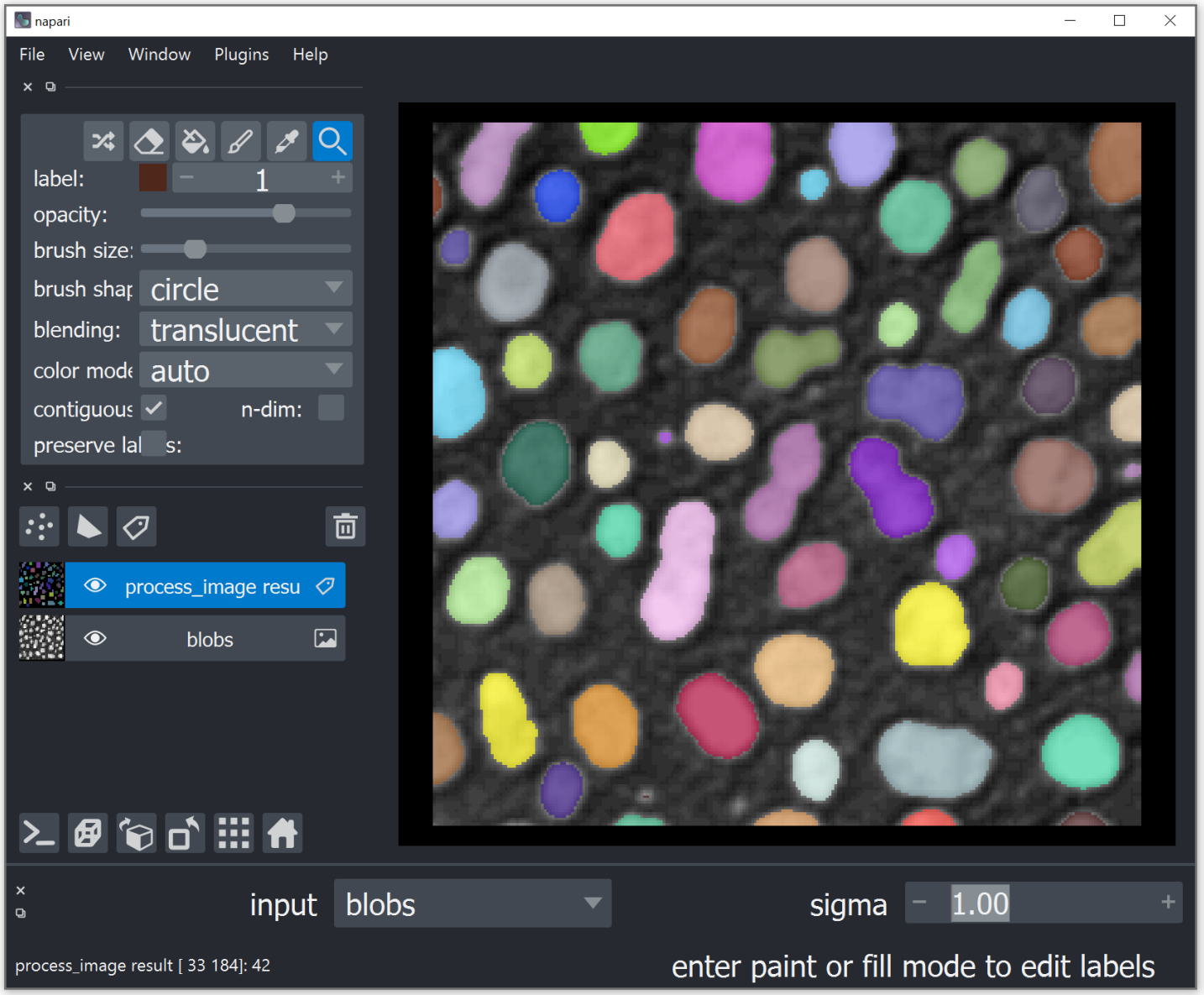
Exercise 8: Napari + magicgui

- After installing python, also install magicgui:

```
pip install magicgui
```

- Check out the minimal napari example and the reference of pycleasperanto.

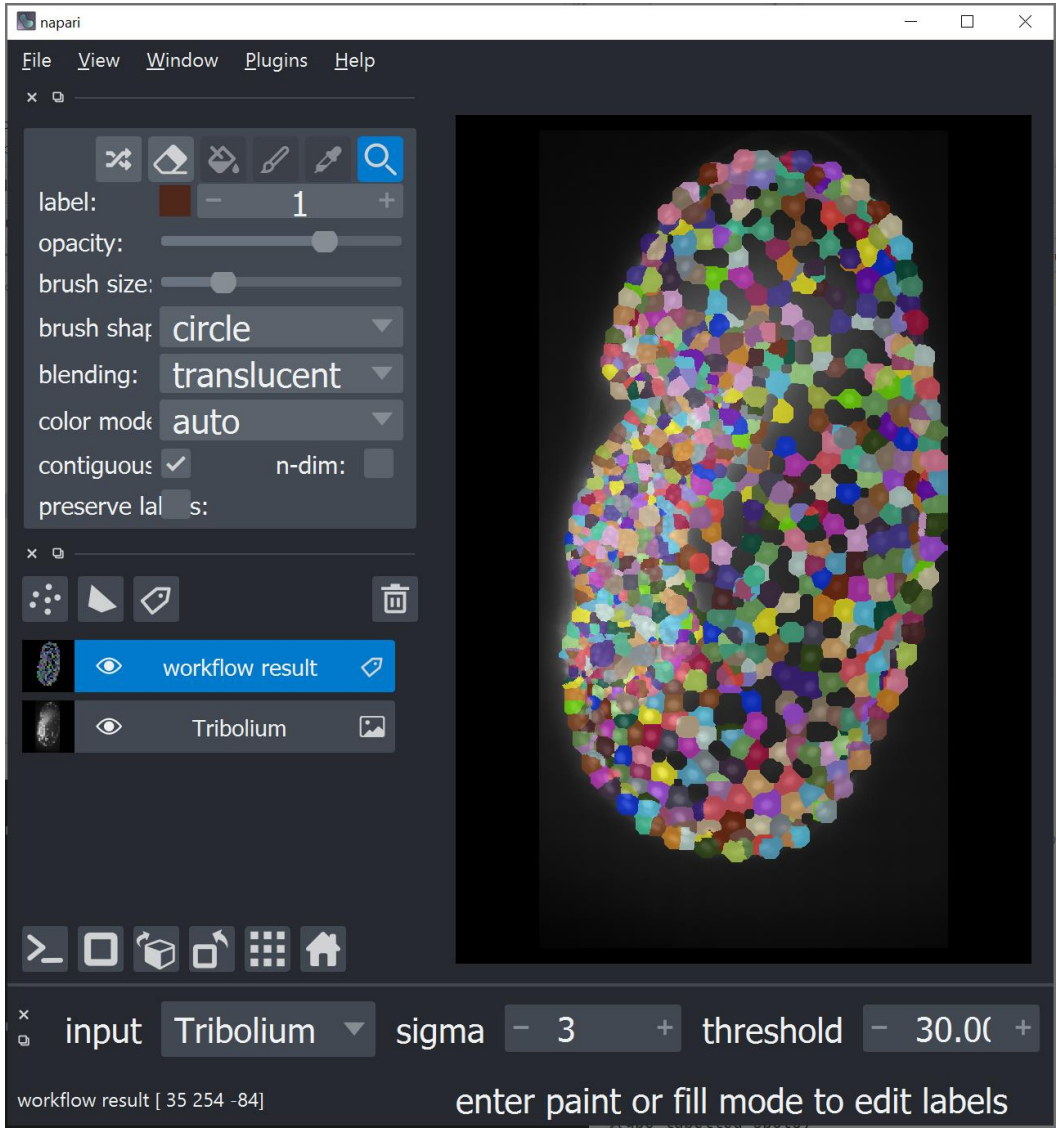
- Write an interactive user interface for changing parameters while segmenting / labeling blobs.



Exercise 9: Tribolium Morphometry in Python

- Design a workflow that can segment cells starting from nuclei center positions in this Tribolium embryo

- Check out the ImageJ macro tutorial and rewrite the workflow in Python https://clij.github.io/clij2-docs/md/tribolium_morphometry/



Exercise 10: Export to cluPath

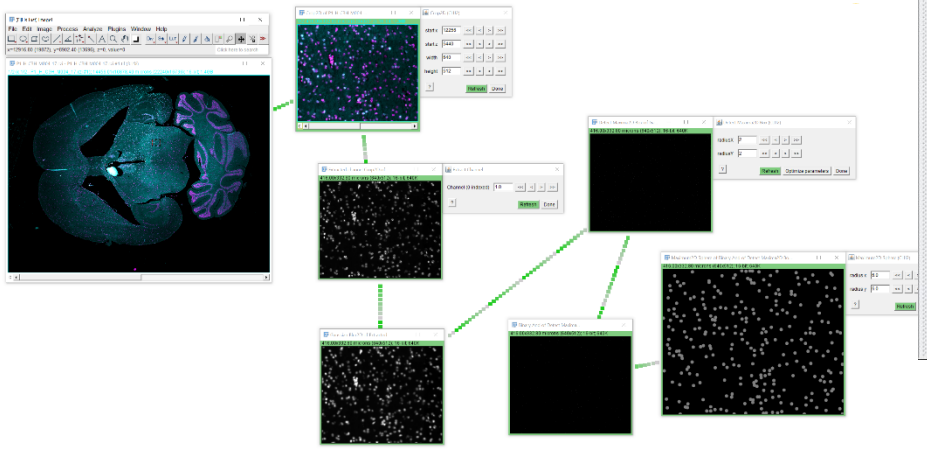
Setup a workflow for detecting nuclei in P1_H_C3H_M004_17.czi - P1_H_C3H_M004_17.czi #1.tif from

https://zenodo.org/record/4276076#.X7_7cM1KguU

(Courtesy of Theresa Suckert, OncoRay, University Hospital Carl Gustav Carus, TU Dresden, license CC-BY 4.0)

Use operations such as

- Crop 2D
- Extract Channel
- Detect Maxima 2D?
- Threshold Otsu 2D?
- Binary Weka Pixel Classifier?

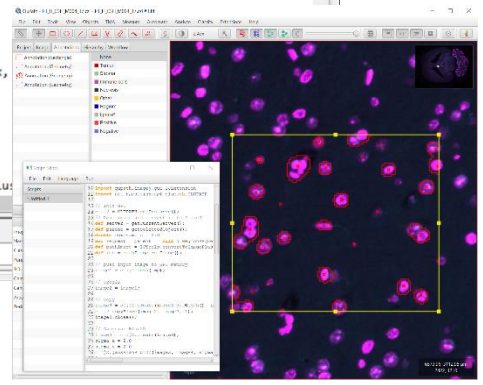


clupath_script_generated.groovy

```
import qupath.imagej.gui.IJExtension
import net.haesleinhuepf.clupath.CLUPATH

// Init GPU
clij2 = CLUPATH.getInstance();
// Read image and convert it to ImageJ
def server = getCurrentServer();
def parent = getSelectedObject();
double downsample = 2.0
def request = parent == null ? RegionRequest.createInstance(server, downsample) :
def pathImage = IJTools.convertToImagePlus(server, request);
def imp = pathImage.getImage();

// push input image to GPU memory
image1 = clij2.push(imp);

// Crop2D
image2 = clij2.create(new longnew long[]{}{1392, 672}, clij2.UnsignedShort);
destination = null
start_x = 12336
start_z = 5712
width = 1392
height = 672
clij2.crop2D(image1, destination, start_x, start_z, width, height);
image1.close();

// Copy
image3 = clij2.create(image2);
clij2.copy(image2, image3);
image2.close();

// Gaussian Blur2D
image4 = clij2.create(image3);
sigma_x = 2.0
sigma_y = 2.0
clij2.gaussianBlur2D(image3, image4, sigma_x, sigma_y);
image3.close();

// Detect Maxima2D Box
image5 = clij2.create(image4);
radiusX = 2.0
radiusY = 2.0
clij2.detectMaxima2DBox(image4, image5, radiusX, radiusY);
```

clupath_script_modified.groovy

```
import qupath.imagej.gui.IJExtension
import net.haesleinhuepf.clupath.CLUPATH

// Init GPU
clij2 = CLUPATH.getInstance();
// Read image and convert it to ImageJ
def server = getCurrentServer();
def parent = getSelectedObject();
double downsample = 2.0
def request = parent == null ? RegionRequest.createInstance(server, downsample) : Regior
def pathImage = IJTools.convertToImagePlus(server, request);
def imp = pathImage.getImage();

// push input image to GPU memory
image1 = clij2.push(imp);

// Crop2D
image2 = image1;

// Copy
image3 = clij2.create(image2.getWidth(), image2.getHeight());
clij2.copySlice(image2, image3, 1);
image2.close();

// Gaussian Blur2D
image4 = clij2.create(image3);
sigma_x = 2.0
sigma_y = 2.0
clij2.gaussianBlur2D(image3, image4, sigma_x,
image3.close();

// Detect Maxima2D Box
image5 = clij2.create(image4);
radiusX = 2.0
radiusY = 2.0
clij2.detectMaxima2DBox(image4, image5, radius
```



Exercise 11: Reuse memory

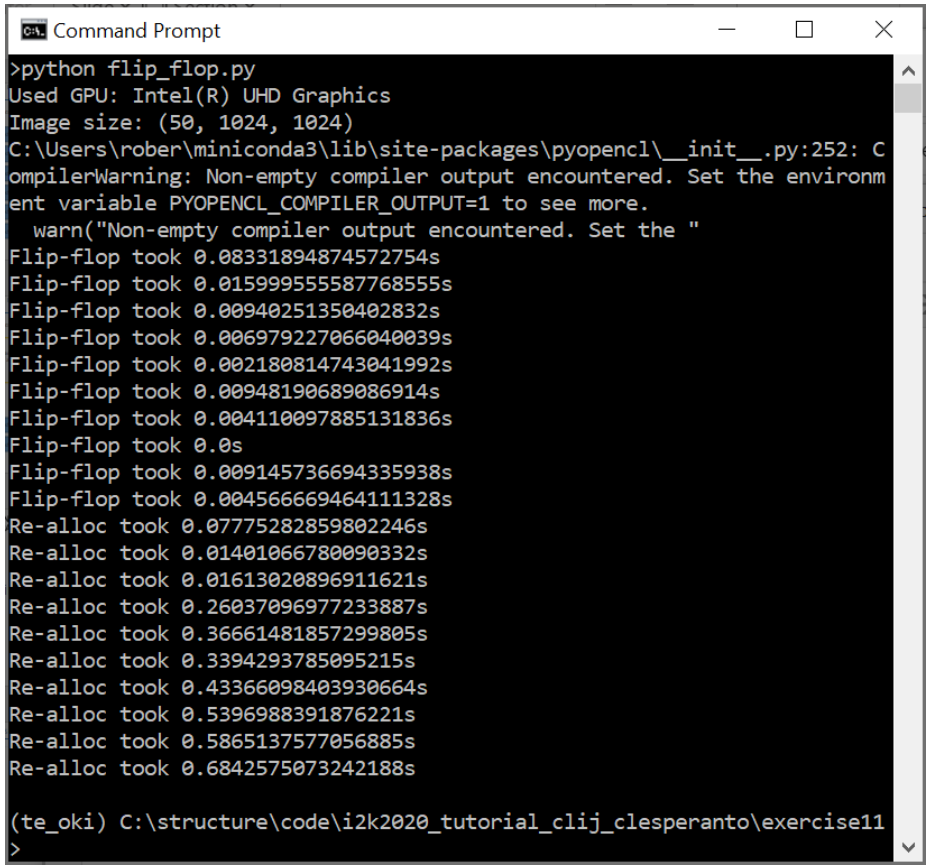
- Compare reusing memory and re-allocating memory. Execute some operations in a loop and measure processing time.

- Reuse memory

```
for i in range(0, num_iterations):
    cle.maximum_sphere(flip, flop, 10, 10, 0)
    cle.minimum_sphere(flop, flip, 10, 10, 0)
```

- Re-allocate memory

```
for i in range(0, num_iterations):
    flop = cle.maximum_sphere(flip, radius_x=10, radius_y=10, radius_z=0)
    flip = cle.minimum_sphere(flop, radius_x=10, radius_y=10, radius_z=0)
```



Exercise 12: CLIc script compile and run

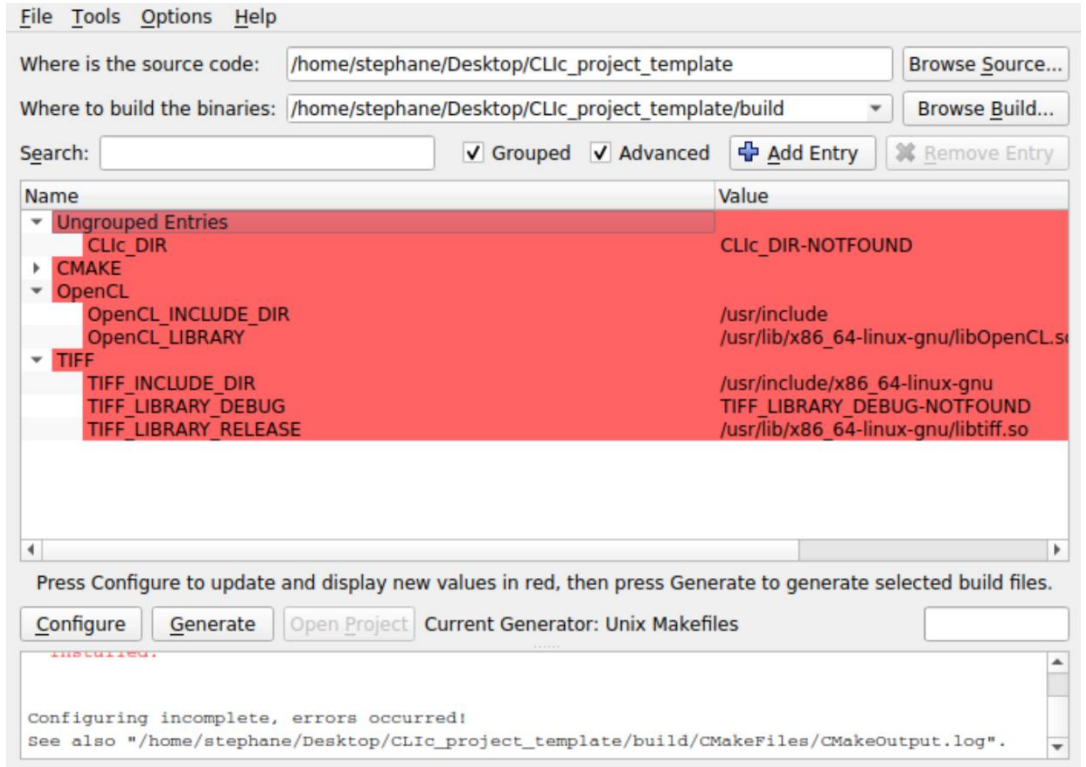
Let’s try to configure and compile your first clEsperantoC script code!

Download the CLIc API and the CLIc template project:

https://github.com/clEsperanto/CLIc_prototype

https://github.com/StRigaud/CLIc_project_template

- Use CMake to configure and generate a C++ project
- Compile the empty CLIc template project
- Add a 2D Gaussian Blur filter



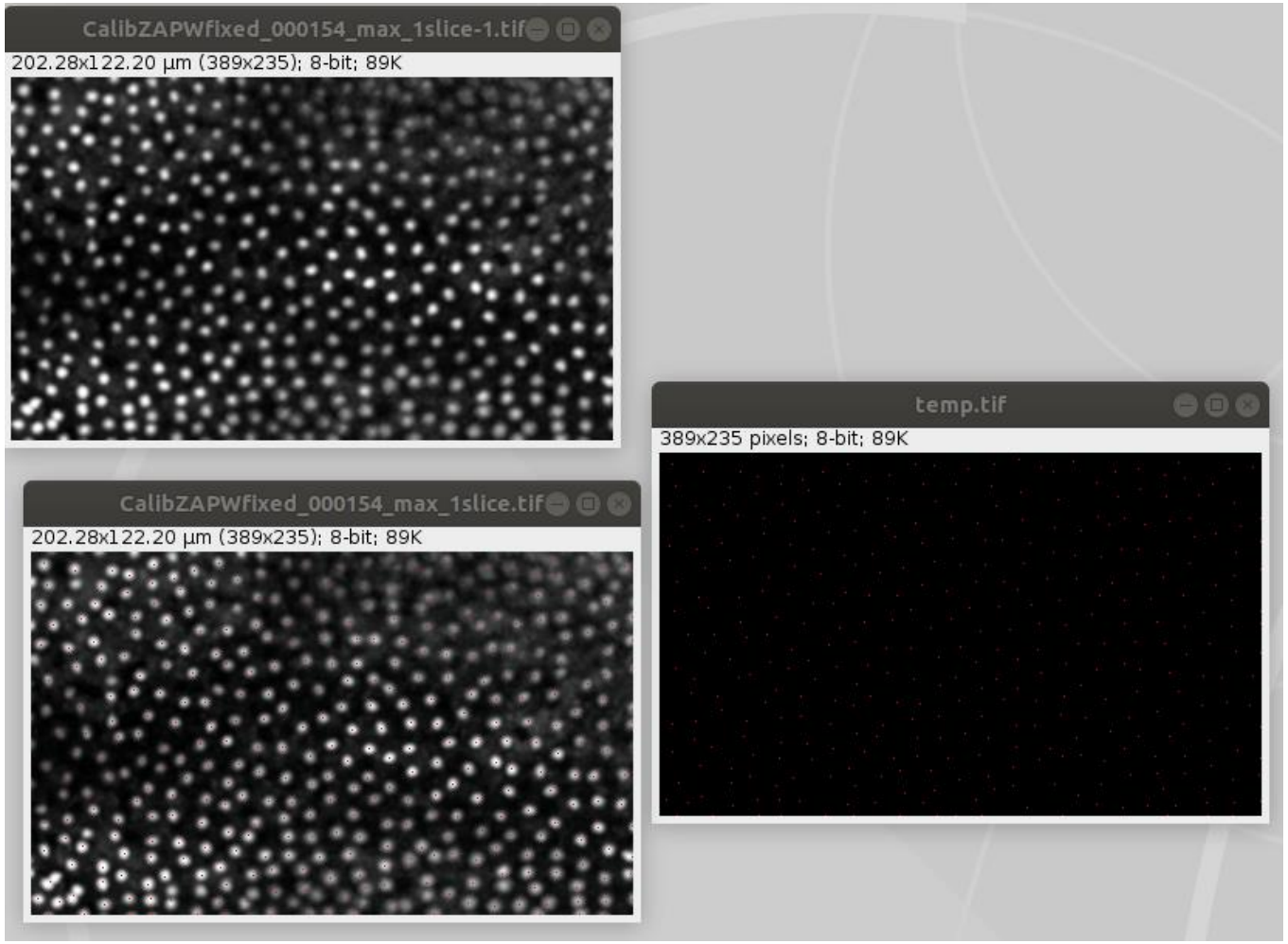
Exercise 13: Count nuclei with CLIc

Why make it simple when you can make it complicate!

Reusing the CLIc project from Execise 12, make a CLIc script to count the number of cell in an image

- Build a C++ workflow
- Manage input and output buffer
- Use CLIjx-Assistant for help/starter

(we should be around 362 nuclei)



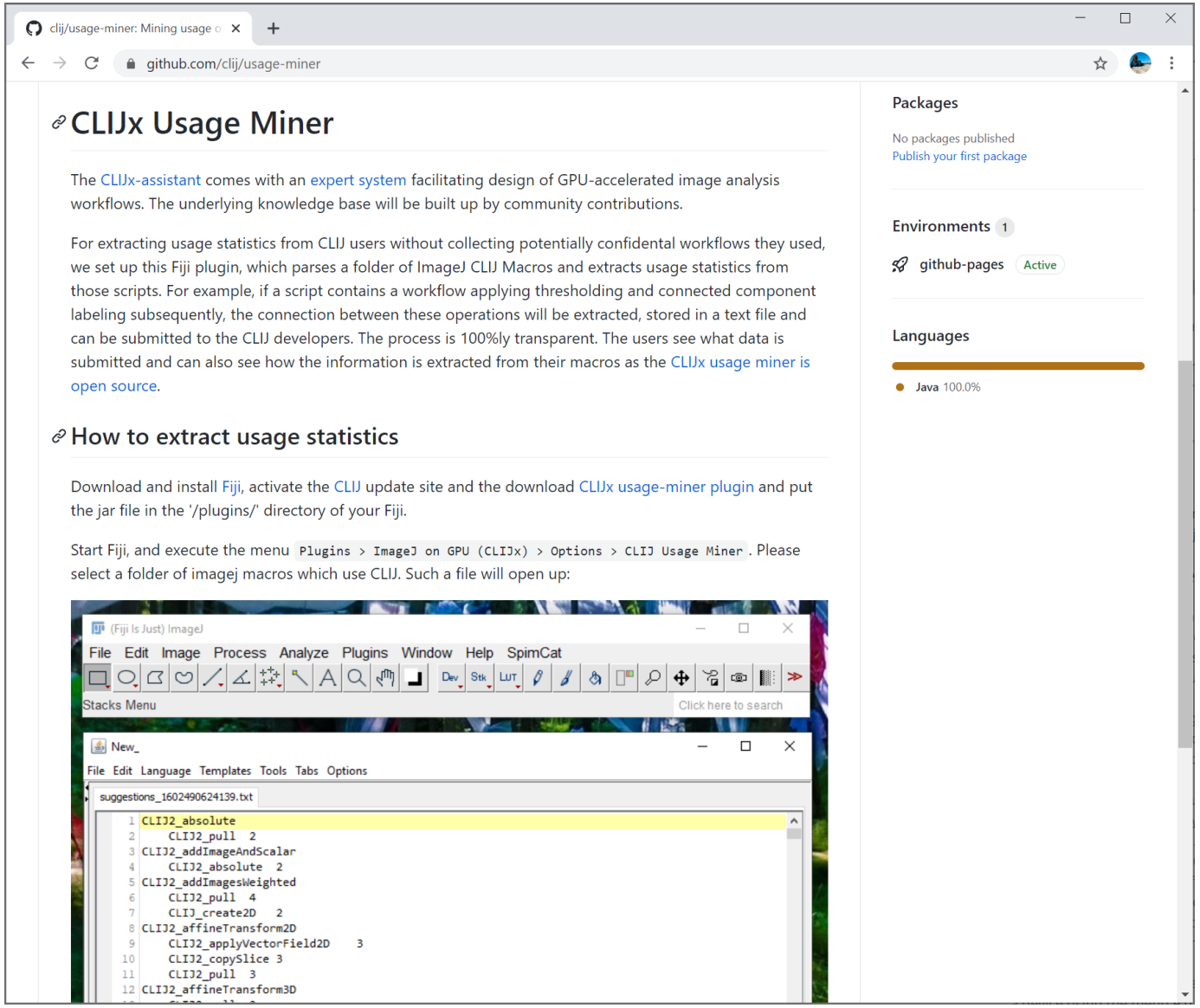
Exercise 14: User miner

- Homework: After you implemented some ImageJ Macros at home, let us analyse your command usage statistics. Visit the usage-miner website to learn how.

https://github.com/clij/usage-miner

- With your contribution, we can make suggestions better!

- Thanks ☺



Exercise 15: clEsperantoJ: Camel or Snake?

- Please try CamelCase snake_case coding using clEsperantoJ in Fiji Java/Jython/Groovy.
- After the session: Go to the CLIJ2 hut in gather.town and place your vote!

```
#
import net.clesperanto.javaprototype.Camel as cle
from ij import IJ

maximum_distance = 72.0

# Load image from disc
imp = IJ.openImage("C:/structure/data/blobs.tif")
image1 = cle.push(imp);
image2 = cle.create(image1)
image3 = cle.create(image2.getDimensions(), cle.Float )
image4 = cle.create(image3)

# image processing
cle.thresholdOtsu(image1, image2)
cle.connectedComponentsLabelingBox(image2, image3)
cle.drawMeshBetweenProximalLabels(image3, image4, maximum_distance)

# show result
cle.pull(image4).show();

# cleanup
cle.clear()
```

```
#
import net.clesperanto.javaprototype.Snake as cle
from ij import IJ

maximum_distance = 72.0

# Load image from disc
imp = IJ.openImage("C:/structure/data/blobs.tif")
image1 = cle.push(imp);
image2 = cle.create(image1)
image3 = cle.create(image2.getDimensions(), cle.Float )
image4 = cle.create(image3)

# image processing
cle.threshold_otsu(image1, image2)
cle.connected_components_labeling_box(image2, image3)
cle.draw_mesh_between_proximal_labels(image3, image4, maximum_distance)
```


PoL
Physics of Life
TU Dresden

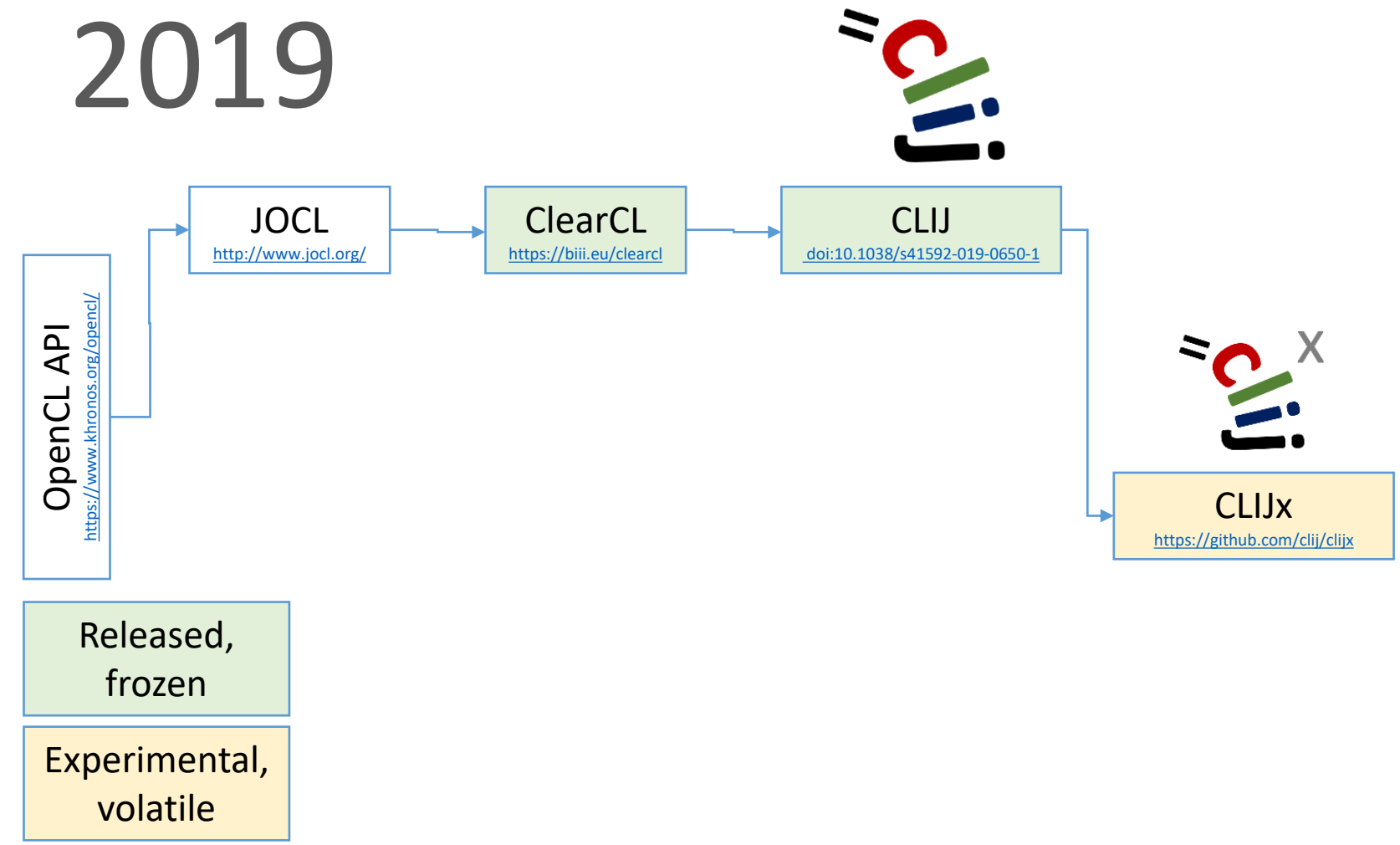
INSTITUT
PASTEUR

CENTER FOR
SYSTEMS BIOLOGY
DRESDEN

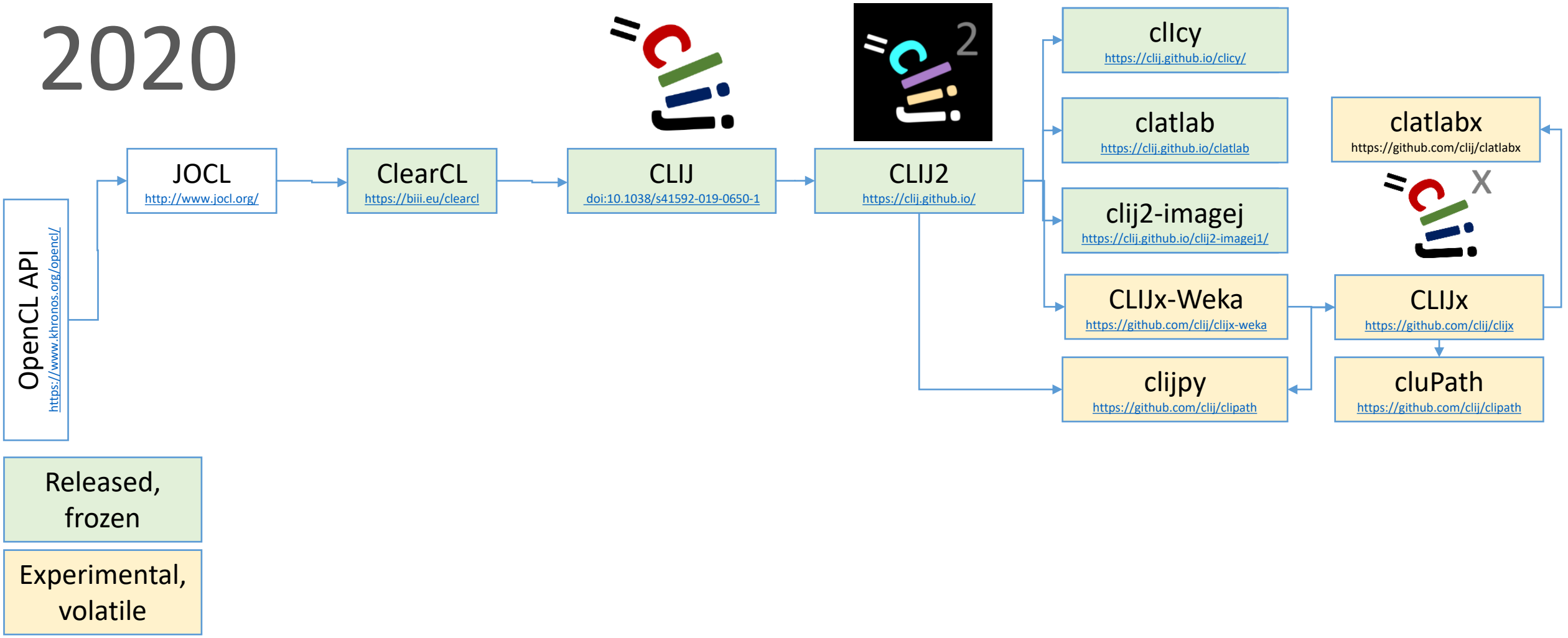
CBG
Max Planck Institute
of Molecular Cell Biology
and Genetics

Summary

clEsperanto lineage tree



clEsperanto lineage tree



clEsperanto lineage tree

2021?

OpenCL API
https://www.khronos.org/opencl/

JOCL
http://www.jocl.org/

ClearCL
https://biii.eu/clearcl

CLIJ
doi:10.1038/s41592-019-0650-1

CLIJ2
https://clij.github.io/

clIcy
https://clij.github.io/clicy/

clatlab
https://clij.github.io/clatlab

clij2-imagej
https://clij.github.io/clij2-imagej1/

CLIJx-Weka
https://github.com/clij/clijx-weka

clijpy
https://github.com/clij/clipath

clatlabx
https://github.com/clij/clatlabx

CLIJx
https://github.com/clij/clijx

cluPath
https://github.com/clij/clipath

clEsperantoJ
?

pyopencl
https://documen.tician.de/pyopencl

pyclesperanto
https://github.com/clEsperanto/pyclesperanto_prototype

CLIc
https://github.com/clEsperanto/CLIc_prototype

cle._

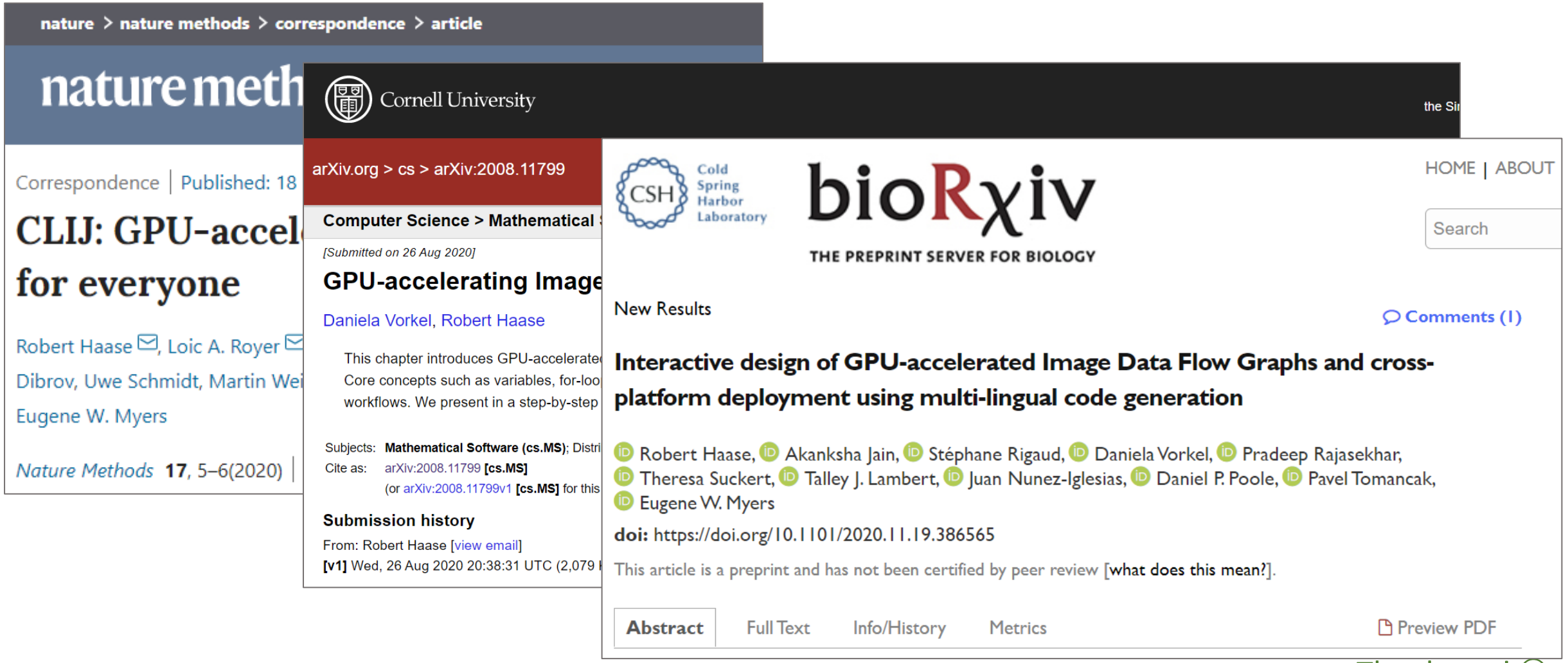
Released, frozen

Experimental, volatile

No longer supported

Last but not least

- If you work with CLIJ and friends, please cite the paper(s). It'll be hard to apply for grants otherwise.



Thank you! ☺

Acknowledements


Akanksha Jain (Treutlein lab) @jain_akanksha_


Dani Vorkel (Myers lab) @happifocus


Theresa Suckert (Krause lab, Oncoray)


Pradeep Rajasekhar (Poole lab, Monash U) @pr4deepr


Talley J. Lambert (HMS) @TalleyJLambert


Juan Nunez-Iglesias (Monash) @jnuneziglesias


Pavel Tomancak (CSBD / MPI-CBG) @PavelTomancak


Gene Myers (CSBD / MPI-CBG) @TheGeneMyers


Jean-Yves Tinevez (Institut Pasteur) @jytinevez


Florian Jug (CSBD / MPI-CBG) @florianjug


Szabolcs Horvát (Modes lab) @szhorvat


Johannes Girstmair (Tomancak lab) @jogirstmair


Brian Northon @truenorth_ia


Matthias Arzt (Myers/Jug lab)

Alex Herbert (University of Sussex)
Alexandr Dibrov (MPI CBG)
Bertrand Vernay (IGBMC, Strasbourg)
Bert Nitzsche (PoL TU Dresden)
Bradley Lowekamp (NIAID Washington)
Bram van den Broek (Netherlands Cancer Institute)
Brenton Cavanagh (RCSI)
Bruno C. Vellutini (MPI CBG)
Carl D. Modes (CSBD/MPI-CBG)
Curtis Rueden (UW-Madison LOCI)
Damir Krunic (DKFZ)
Daniel J. White (GE)
Deborah Schmidt (CSBD/MPI CBG)
Douglas P. Shepard (Univ Arizona)
Eduardo Conde-Sousa (University of Porto)
Erick Ratamero (The Jackson Laboratory)
Gaby G. Martins (IGC)
Gayathri Nadar (MPI CBG)
Guillaume Witz (Bern University)
Giovanni Cardone (MPI Biochem)
Irene Seijo Barandiaran (MPI CBG Dresden)
Jan Brocher (Biovoxxel)
Jean-Yves Tinevez (Institut Pasteur)
Johannes Girstmair (MPI CBG)
Juergen Gluch (Fraunhofer IKTS)
Kisha Sivanathan (Harvard Medical School)
Kota Miura
Laurent Thomas (Acquifer)
Lior Pytowski (University of Oxford)
Loic A. Royer (CZ Biohub)
Marion Louveaux (Institut Pasteur Paris)
Martin Weigert (EPFL Lausanne)
Matthew Foley (University of Sydney)
Nico Stuurman (UCSF)
Nicola Maghelli (MPI CBG)
Nicolas Brouilly (IBDM Marseille)
Nik Cordes (Los Alamos National Laboratory)
Noreen Walker (MPI CBG Dresden)
Ofra Golani (Weizmann Institute of Science)
Patrick Dummer
Peter Haub
Pete Bankhead (University of Edinburgh)
Peter Steinbach (HZDR Dresden)
Pit Kludig
Rita Fernandes (University of Porto)
Romain Guiet (BIOP-EPFL)
Ruth Whelan-Jeans
Sebastian Munck (VIB Leuven)
Siân Culley (MRC LMCB)
Stein Rørvik
Stéphane Dallongeville (Institut Pasteur)
Tanner Fadero (UNC-Chapel Hill)
Thomas Irmer (Zeiss)
Tobias Pietzsch (MPI-CBG)
Uwe Schmidt (MPI CBG)
Wilson Adams (VU Biophotonics)

https://fiji.sc

https://napari.dev

https://image.sc

http://eubias.org/NEUBIAS/

MPI CBG Core Facilities

- Advanced Imaging Facility
- Light Microscopy Facility
- Scientific Computing
- IT Department

Funding:

R.H. was supported by the German Federal Ministry of Research and Education (BMBF) under the code 031L0044 (Sysbio II) and by the Deutsche Forschungsgemeinschaft under Germany's Excellence Strategy – EXC2068 - Cluster of Excellence Physics of Life of TU Dresden.

@haesleinhuepf @StRigaud

https://clij.github.io/

#I2K2020, virtually in Janelia

Summary

- GPU-acceleration works best with
  - workflows with many steps between push and pull
  - processing many images
  - on dedicated GPUs.
- IDFGs work best if you
  - browse suggestions
  - move the windows
  - use Fiji search!