UNIVERSITÄT DES SAARLANDES

dfki ai

Integrating ImageJ/Fiji Image Processing in Chaldene Visual Programming System

Ziwei He

Master Thesis Seminar

March 22, 2024,
Saarbrücken, Germany

Outline

- Background
- Motivation
- Methodology
- Progress Made
- Conclusion
- Next Step

Background - ImageJ/Fiji

ImageJ/Fiji

- **ImageJ**, is a Java-based software widely used for microscopy image processing in material sciences and bioinformatic field.

- **Fiji** is a distribution of ImageJ with plugins which facilitate scientific image analysis.

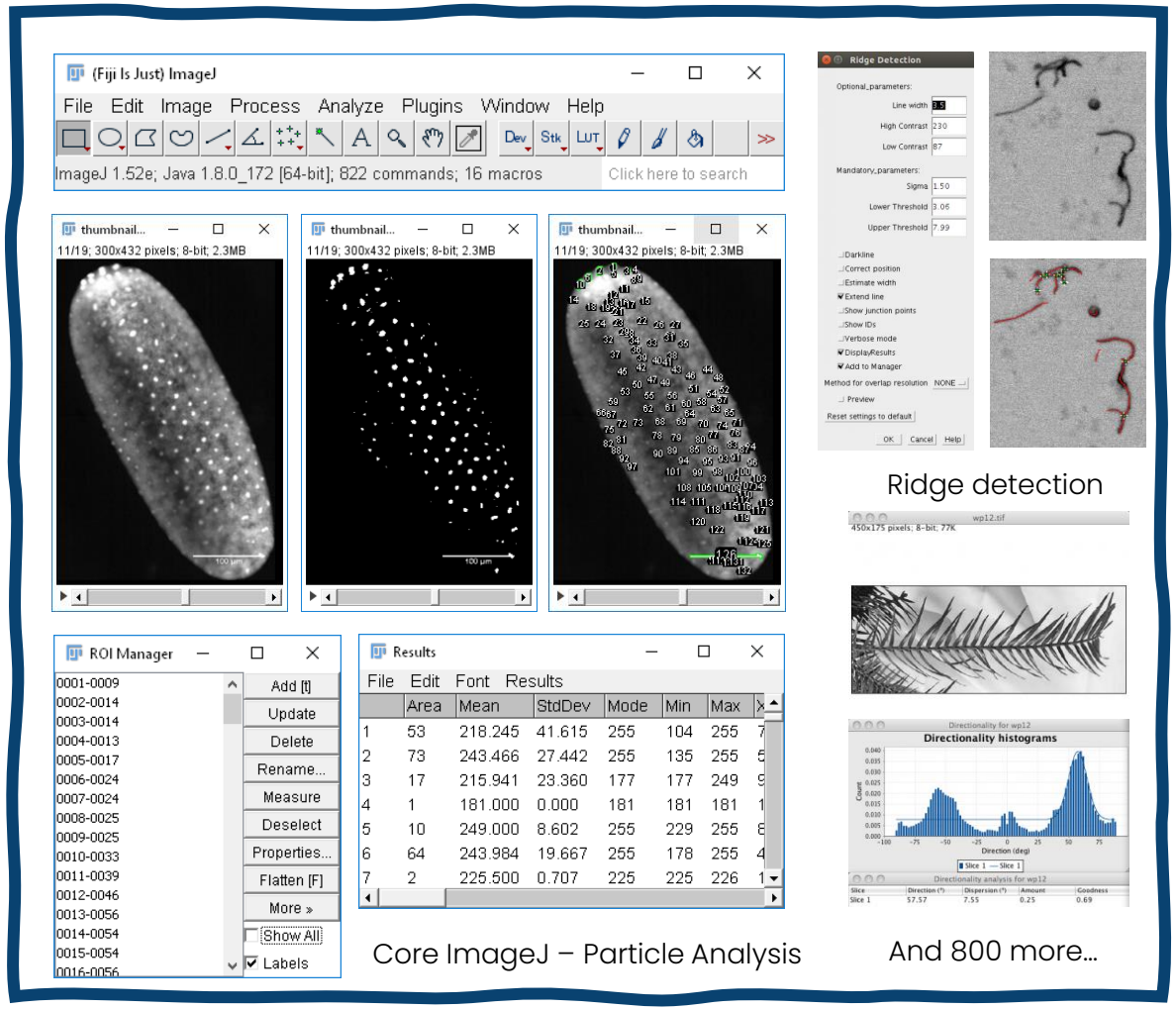


Figure 1 : Core ImageJ/Fiji and other plugins [1]

Background – Chaldene

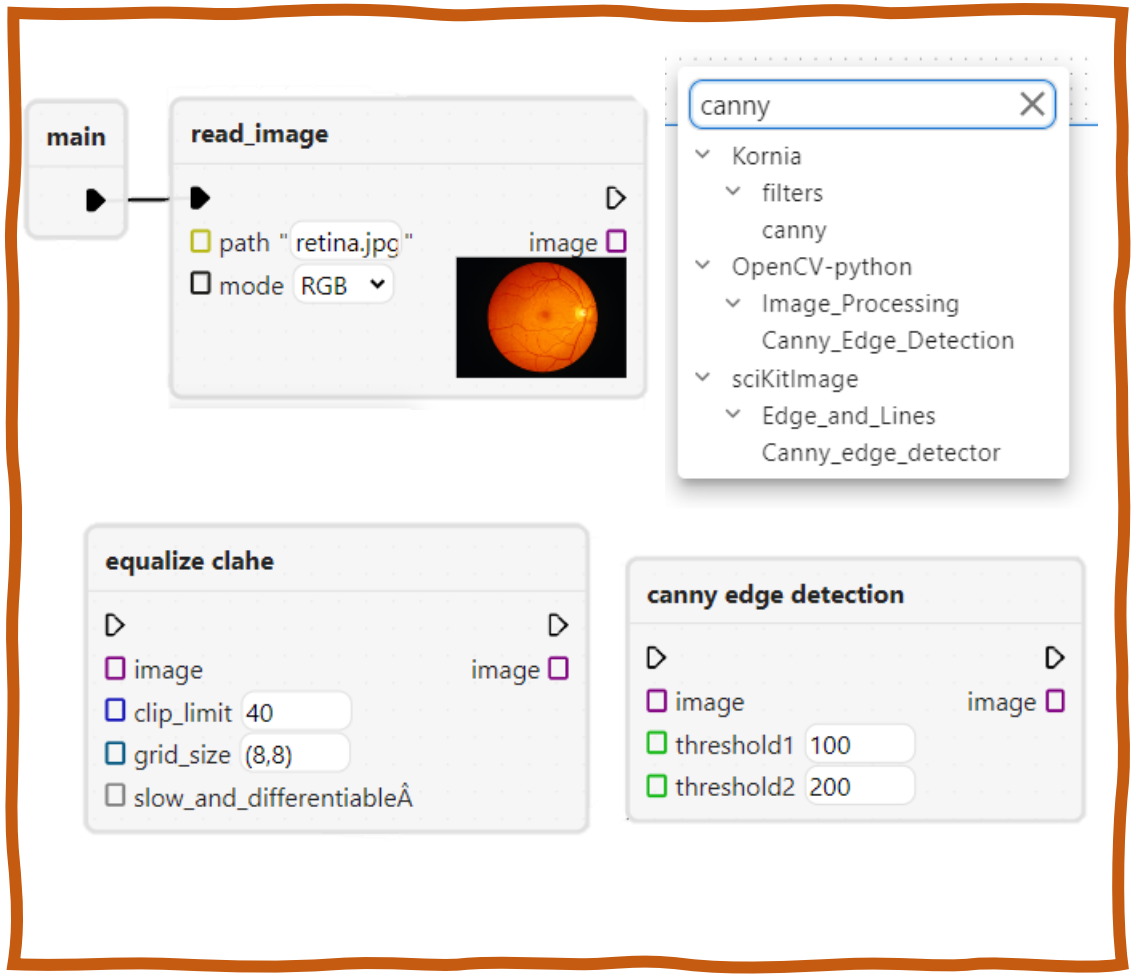
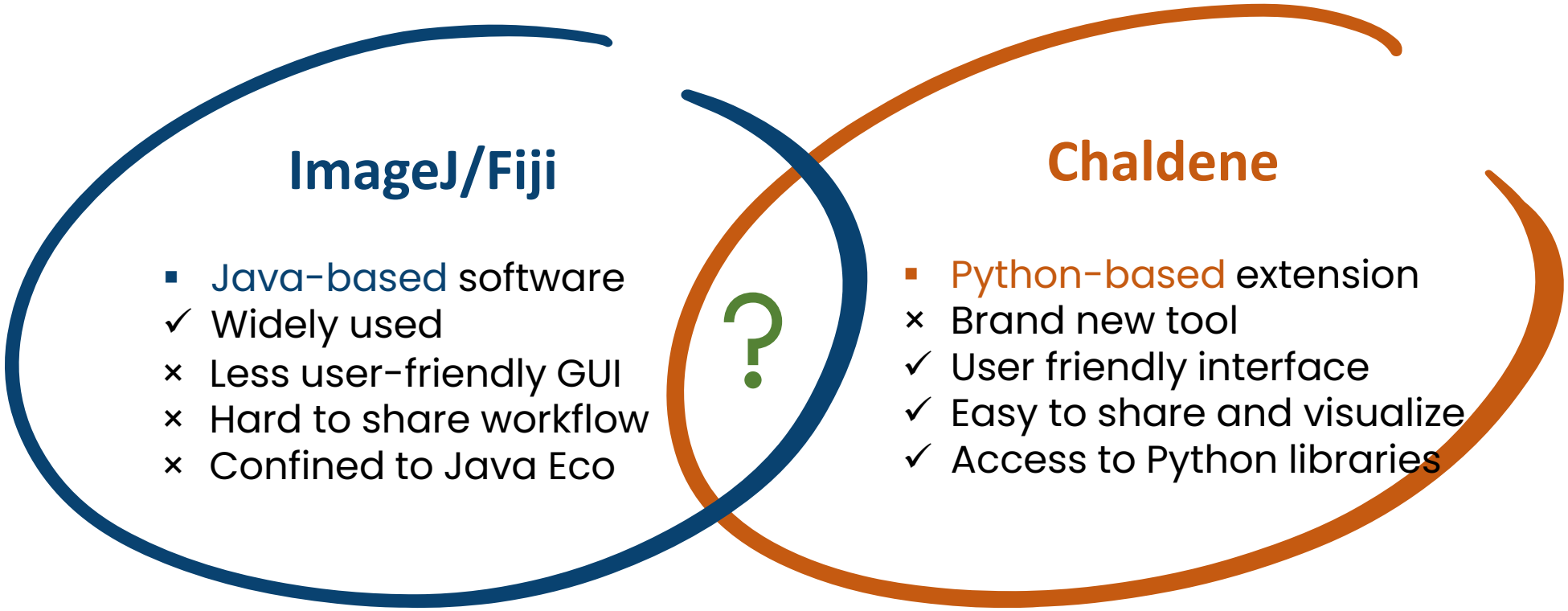


Figure 2 : Chaldene's Visual Nodes and Search Menu

Chaldene[2]
Visual Programming System

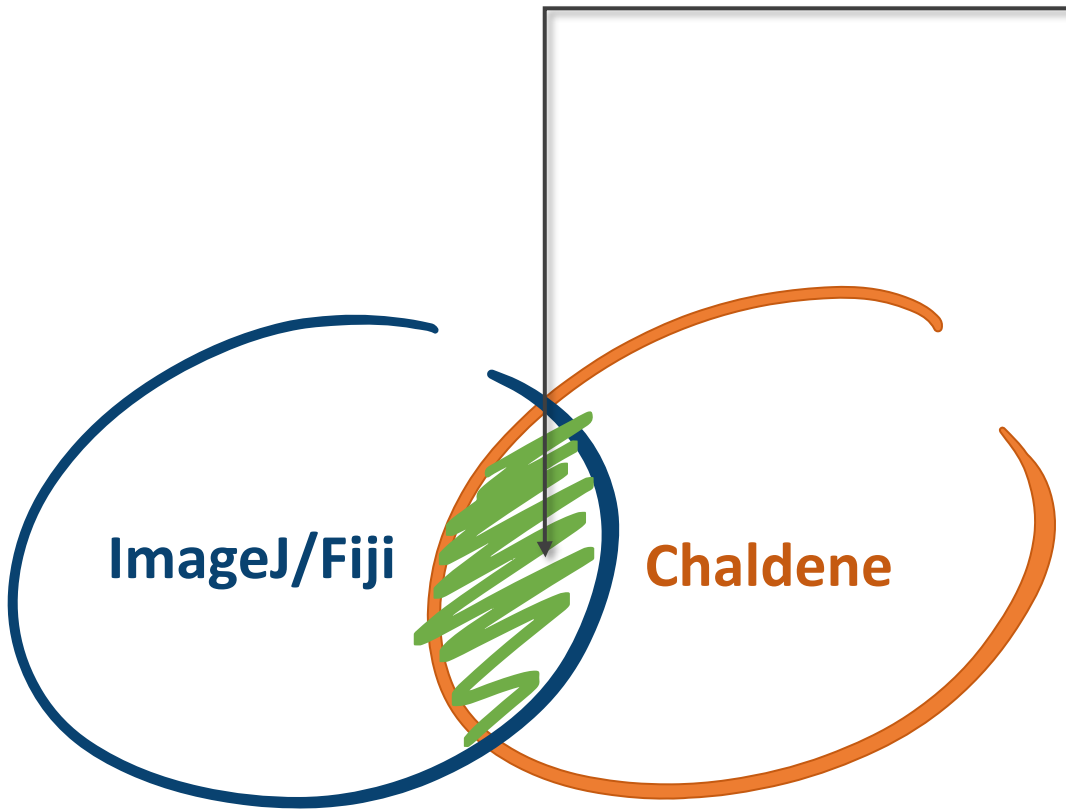
- **Chaldene** is a visual programming extension to JupyterLab that executed based on IPython kernel.
- **Chaldene** is specifically designed to cater to scientists who require scientific image processing and data analysis for their research, but have limited programming experience.

Motivation - Comparison



Integrating ImageJ/Fiji Image Processing in Chaldene Visual Programming System

Motivation – Integration



We want

ImageJ Function + Chaldene System

= Familiar Functions in Good New Tool

= Easy workflow sharing and visualization

= Expand boundary of ImageJ-confined scientists with limited programming skills [3]

Challenges

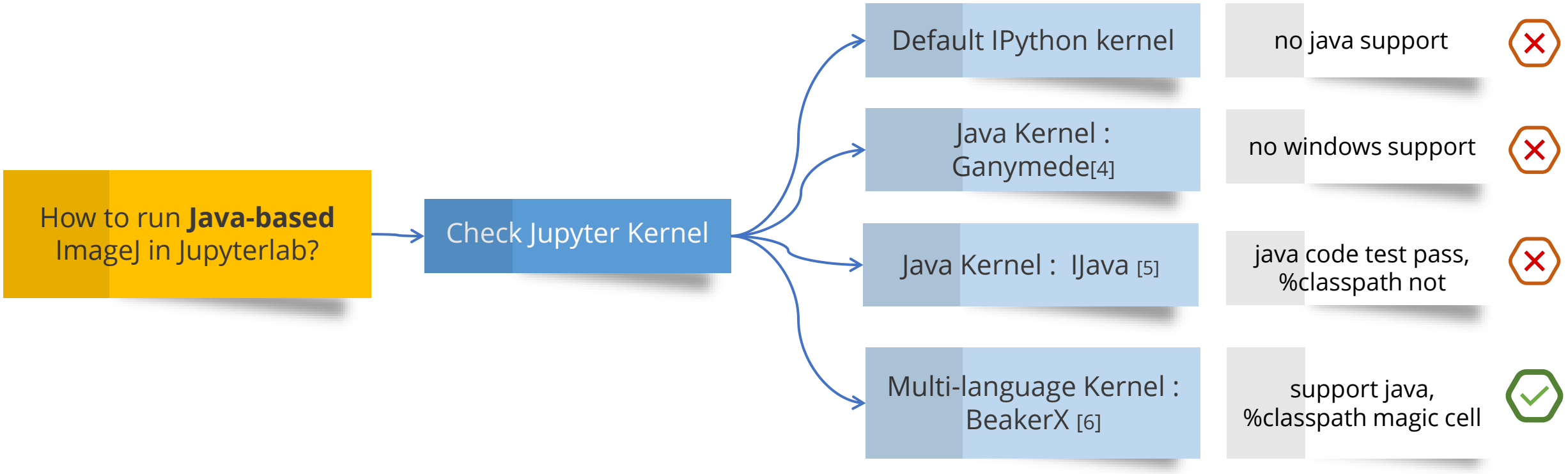
How to run Java-based ImageJ in Jupyterlab?

How to integrate Java-based ImageJ in Python-based Chaldene?

Outline

- Background
- Motivation
- Methodology
- Progress Made
- Conclusion
- Next Step

Methodology – Java Execution in JupyterLab



Progress – ImageJ in JupyterLab via BeakerX

How to run **Java-based** ImageJ in Jupyterlab?

BeakerX

- Multi-language Jupyter Kernel
- Supporting
  - **Java kernel**
  - Python Kernel
  - %classpath

ImageJ Gateway [7]

- Get access to ImageJ API
- Load ImageJ Library from remote Maven repository
- Via *%classpath add mvn net.imagej*

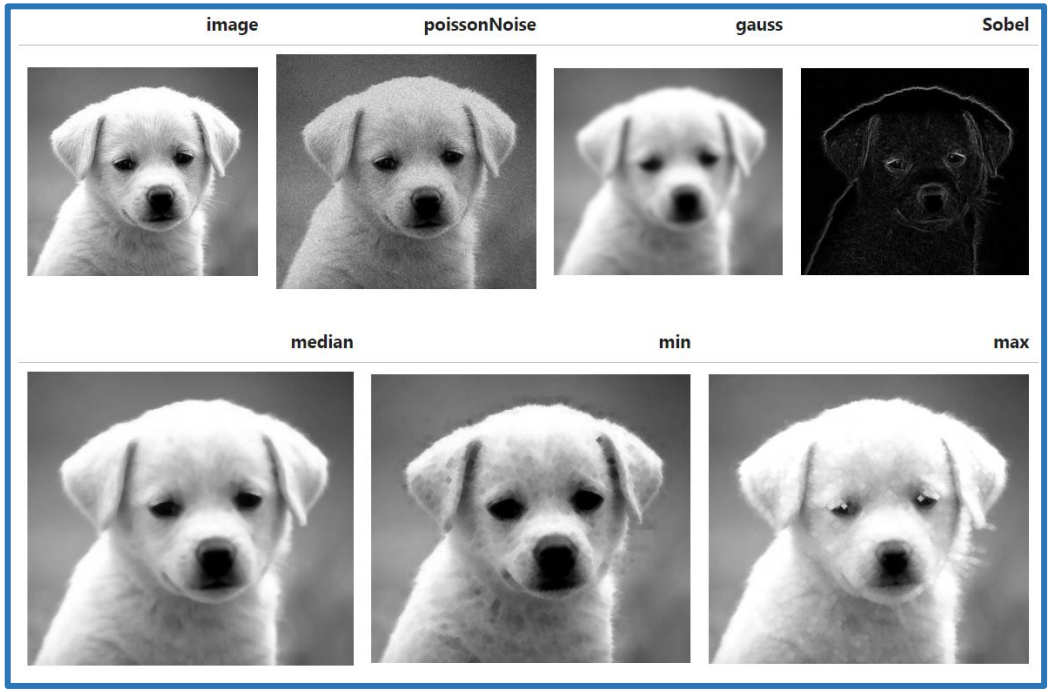


Figure 3 : Image Filtering Result via ImageJ in Jupyterlab

Methodology – Multilanguage Jupyterlab

How to run **Java-based** ImageJ in Jupyterlab?

BeakerX + ImageJ Gateway

- BeakerX supporting
  - Java kernel
  - Python Kernel
  - %classpath
- Get access to ImageJ API via gateway

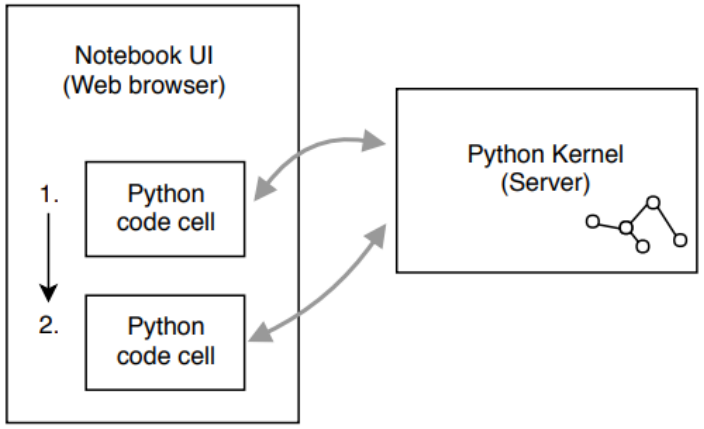
How to integrate **Java-based** ImageJ in **Python-based** Chaldene?

Multi kernels handling Multi languages

- SoS Notebook [8]
- PolyJus Notebook [9]

Methodology – Polyglot Notebook

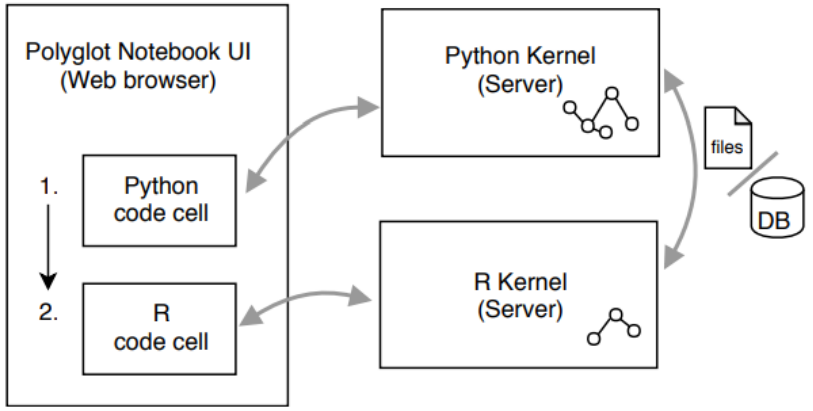
Standard Notebook with Single Kernel (e.g. IPython [8])



Standard Notebook

- Single Jupyter execution kernel
- Supports **one** programming language
- All data are stored in one server

Polyglot Notebook with Separate Kernels (e.g. SoS [7])

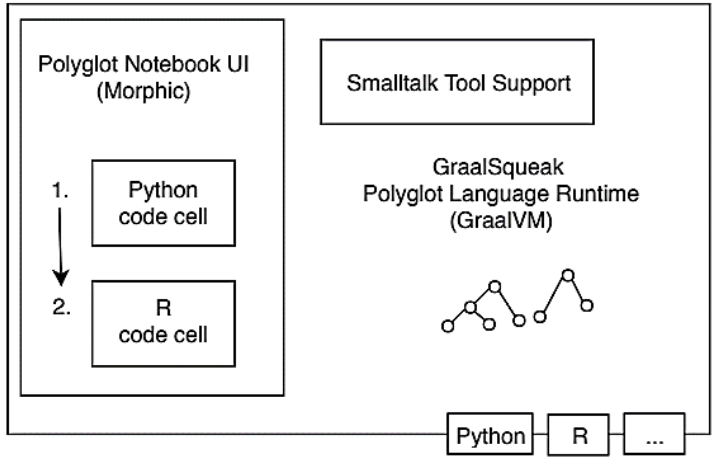


SoS Notebook

- Multiple Jupyter execution kernels
- Supports multi languages
- All data are stored in **additional database**

Changing Kernel would lose all current data

PolyJuS Integrated UI and Polyglot Language Runtime



PolyJus

- One Polyglot kernel based on **GraalVM**
- Supports multi languages
- All data are stored in its **own environment**

Outside Jupyter Environment

Methodology – New Way

Details in Issue 59

How to integrate Java-based ImageJ in Python-based Chaldene?

Multi kernels handling Multi languages

- SoS Notebook [8]
- PolyJus Notebook [9]

How to run Java-based ImageJ in Jupyterlab with **IPython Kernel to be compatible with Chaldene**?

PyImageJ

PyImageJ

- Provide wrapper functions that allow us to run ImageJ with Python
- Provide conversion API for ImageJ image <-> Python Image

Methodology – PyImageJ

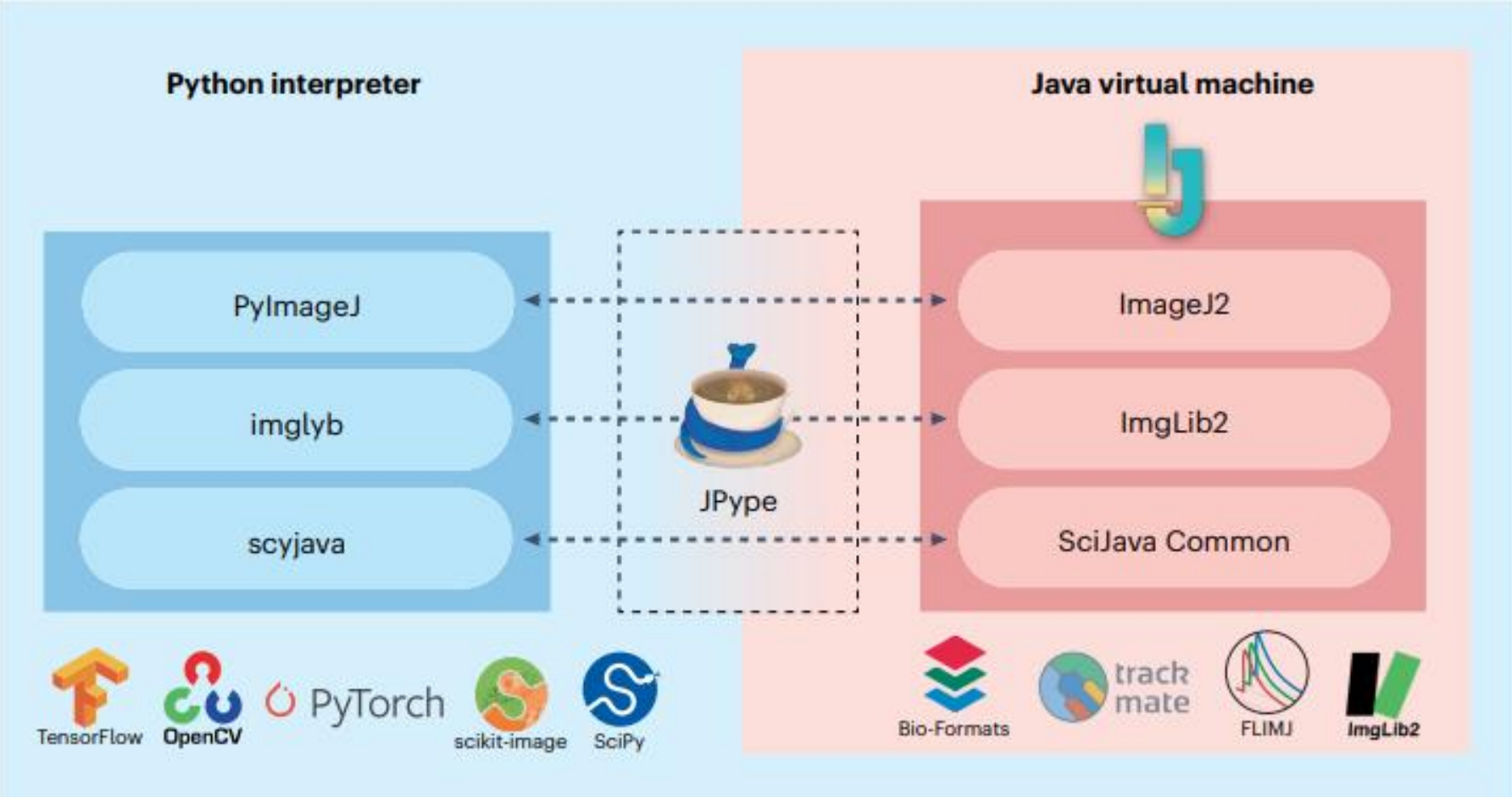


Figure 4 : Architecture of PyImageJ [10]

Progress - Run ImageJ via PyImageJ

PyImageJ_test1.ipynb

Code

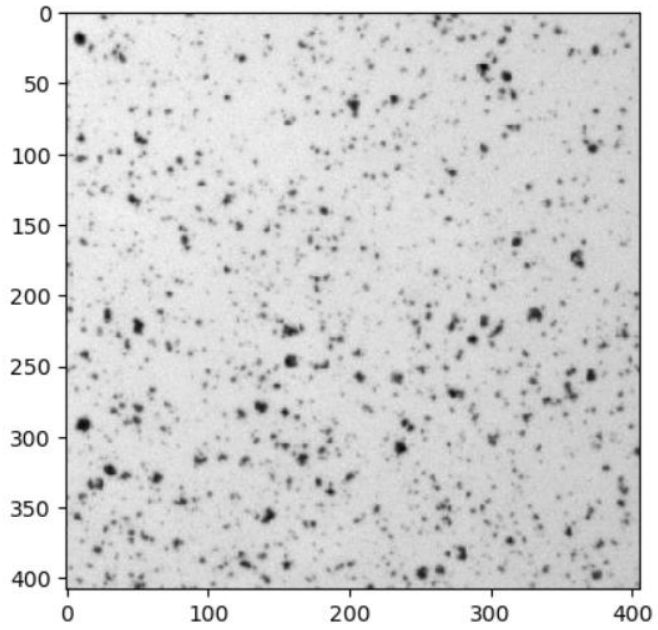
Python (pyimagej)

```
[42]: colony_img= ij.io().open('https://wsr.imagej.net/images/Cell_Colony.jpg')

p_colony_img = ij.py.from_java(colony_img)
ij.py.show(p_colony_img,cmap='gray')
print(f"cell_colony type: {type(colony_img)}")
print(f"xr_colony type: {type(p_colony_img)}")
```

Python Kernel

```
[java.lang.Enum.toString] [INFO] Populating metadata
```




```
cell_colony type: <java class 'net.imagej.DefaultDataset'>
xr_colony type: <class 'xarray.core.dataarray.DataArray'>
```

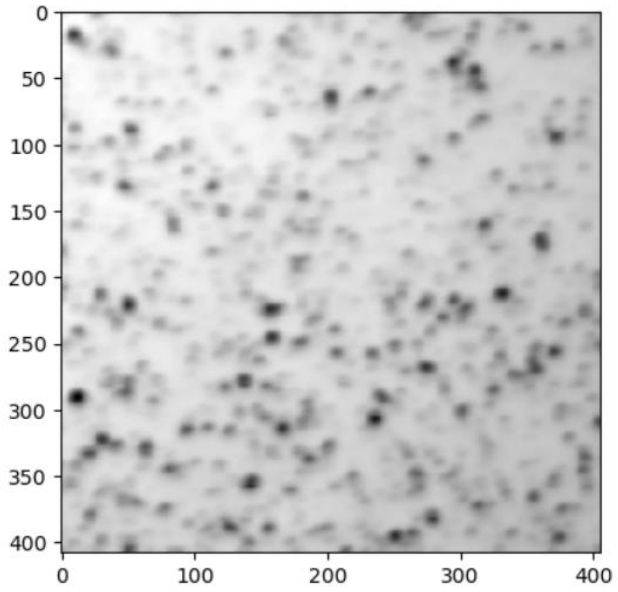
Figure 5: Run ImageJ Functions with Python Kernel

Progress - Run ImageJ via PyImageJ

```
HyperSphereShape = jimport('net.imglib2.algorithm.neighborhood.HyperSphereShape')
radius = HyperSphereShape(2)
```

Import Java Library by scyjava

```
ij.op().filter().mean(result, colony_img, radius)
ij.py.show(result,cmap='gray')
```




```
thresholded = ij.op().run("threshold.otsu", result)
ij.py.show(thresholded,cmap='gray')
```

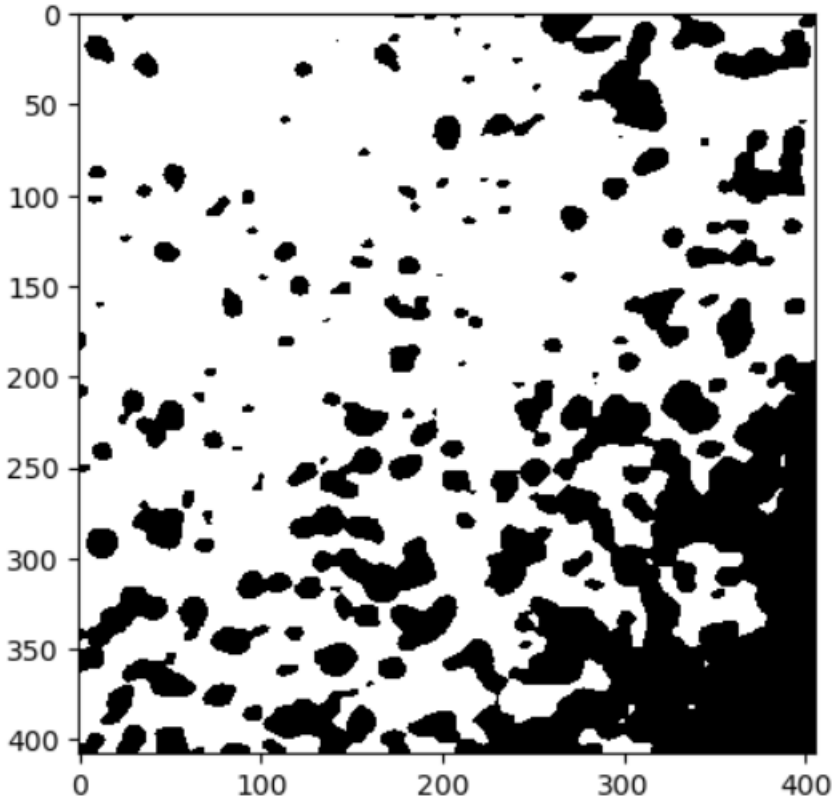


Figure 6: PyImageJ supports most of ImageJ plugins in headless mode

Progress - Run ImageJ via PyImageJ

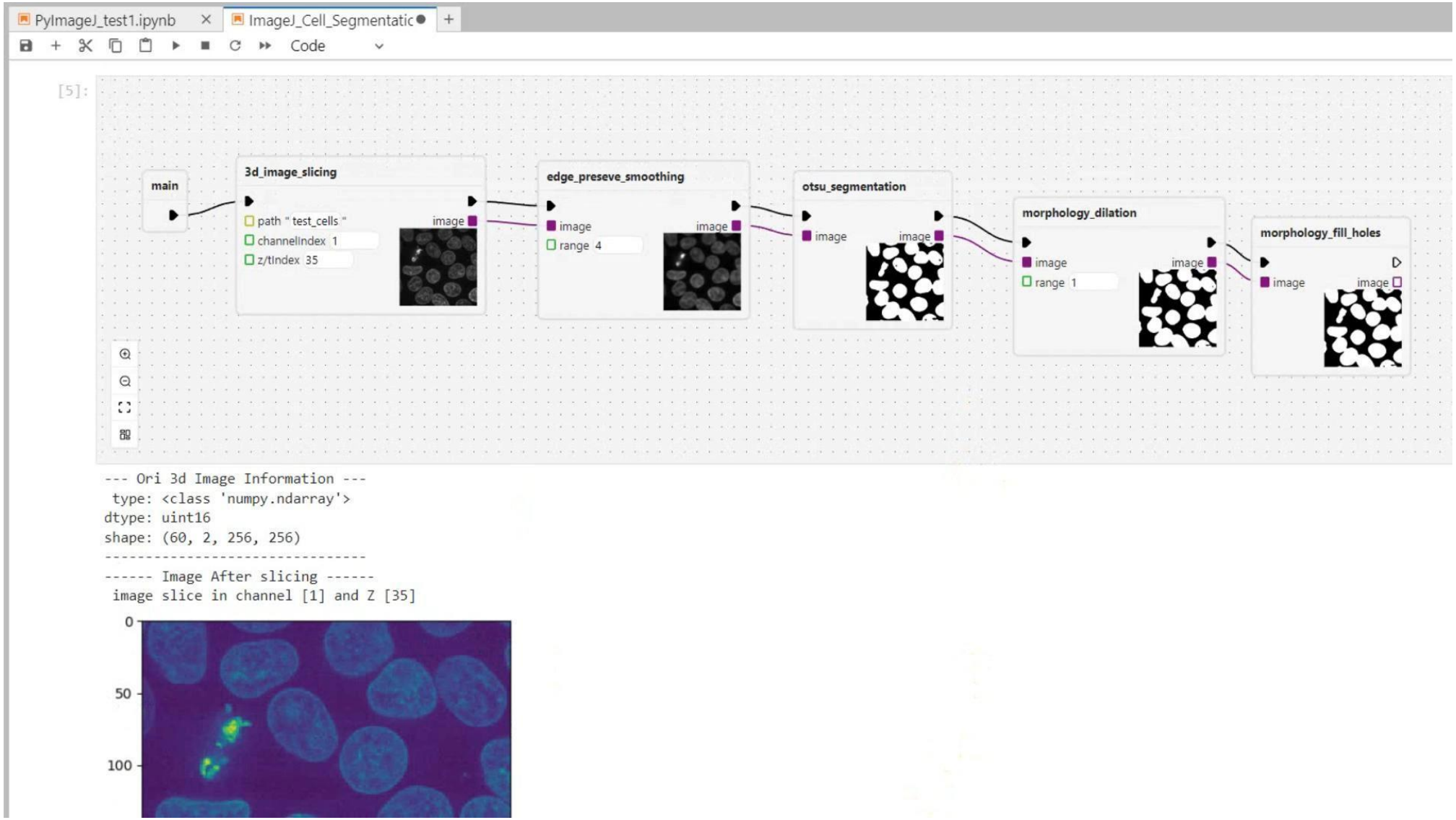
Java -> Python direct image conversions

Convert between Java and Python Image

Input type	Input dimensions/shape	Hints	Method	Output type	Output dimensions/shape
`net.imagej.Dataset`	`('X', 'Y', 'Channel', 'Time')` / `(250, 100, 3, 15)`	*Not supported*	`to_xarray()`	`xarray.DataArray`	`('t', 'row', 'col', 'ch')` / `(15, 100, 250, 3)`
`net.imagej.ImgPlus`	`('X', 'Y', 'Channel', 'Time')` / `(250, 100, 3, 15)`	*Not supported*	`to_xarray()`	`xarray.DataArray`	`('t', 'row', 'col', 'ch')` / `(15, 100, 250, 3)`
`net.imglib2.img.Img`	No `dims` attribute / `(250, 100, 3, 15)`	*Not supported*	`to_xarray()`	`xarray.DataArray`	`('dim_0', 'dim_1', 'dim_2', 'dim_3')` / `(15, 3, 100, 250)`
`ij.ImagePlus`	`('X', 'Y', 'C', 'T')` / `(250, 100, 3, 15)`	*Not supported*	`to_xarray()`	`xarray.DataArray`	`('t' ,'row', 'col', 'ch')` / `(15, 100, 250, 3)`

Figure 7: Conversion API for ImageJ image <--> Python Image[10]

Progress - ImageJ Workflow in Chaldene



Video 1: ImageJ Cell Segmentation Workflow [10] integrated in Chaldene System

Details in Issue 68

Motivation – Auto Generation for ImageJ Op

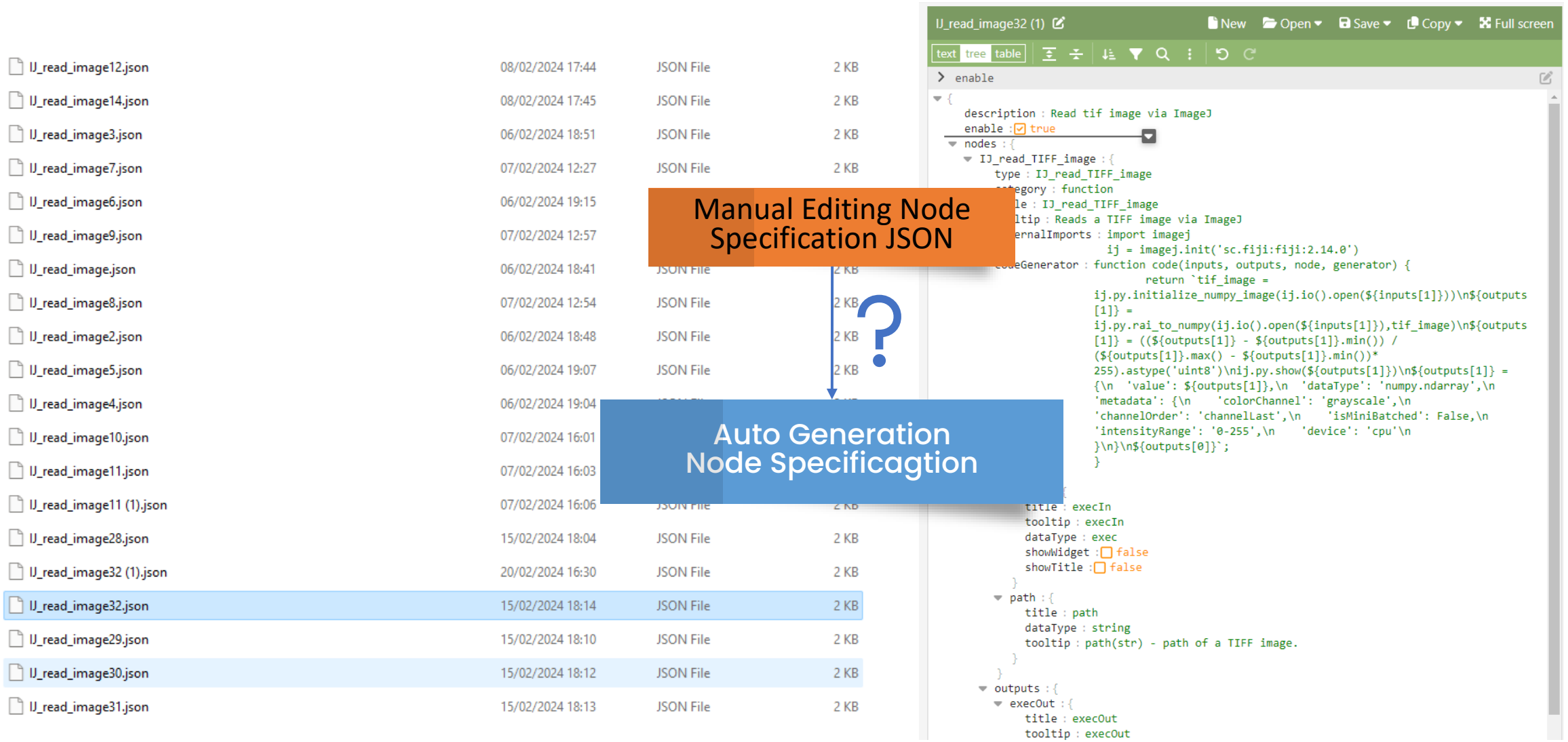
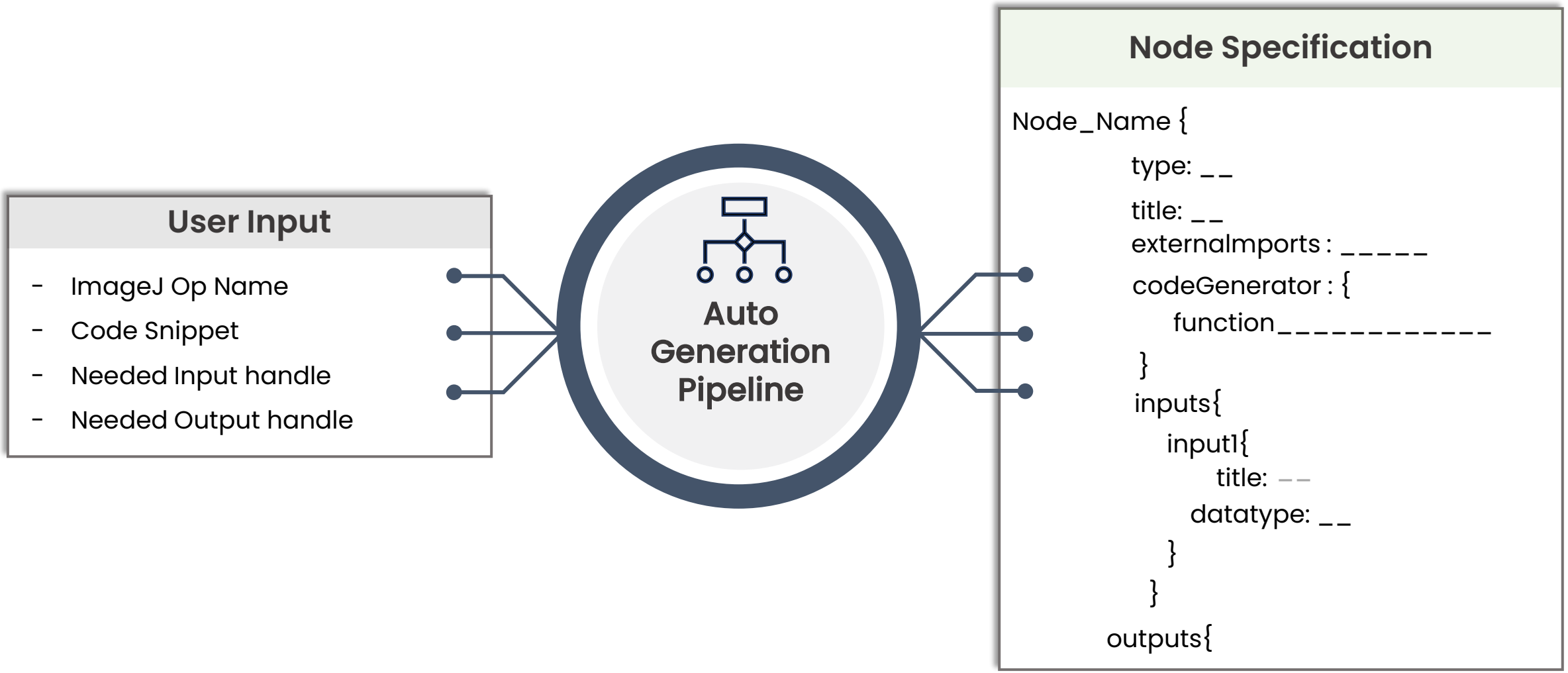
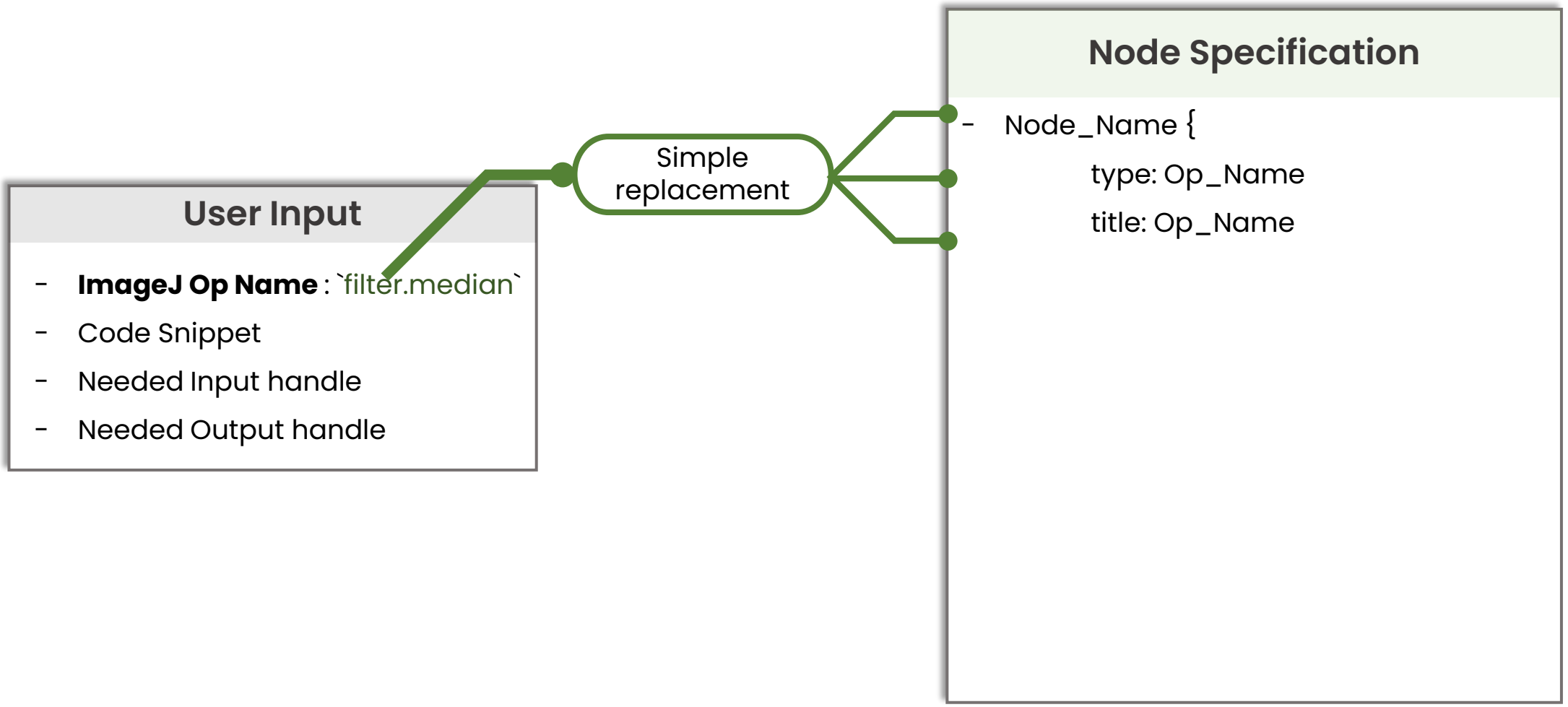


Figure 8: Thirty times of manual editing for one Visual Node Specification JSON file

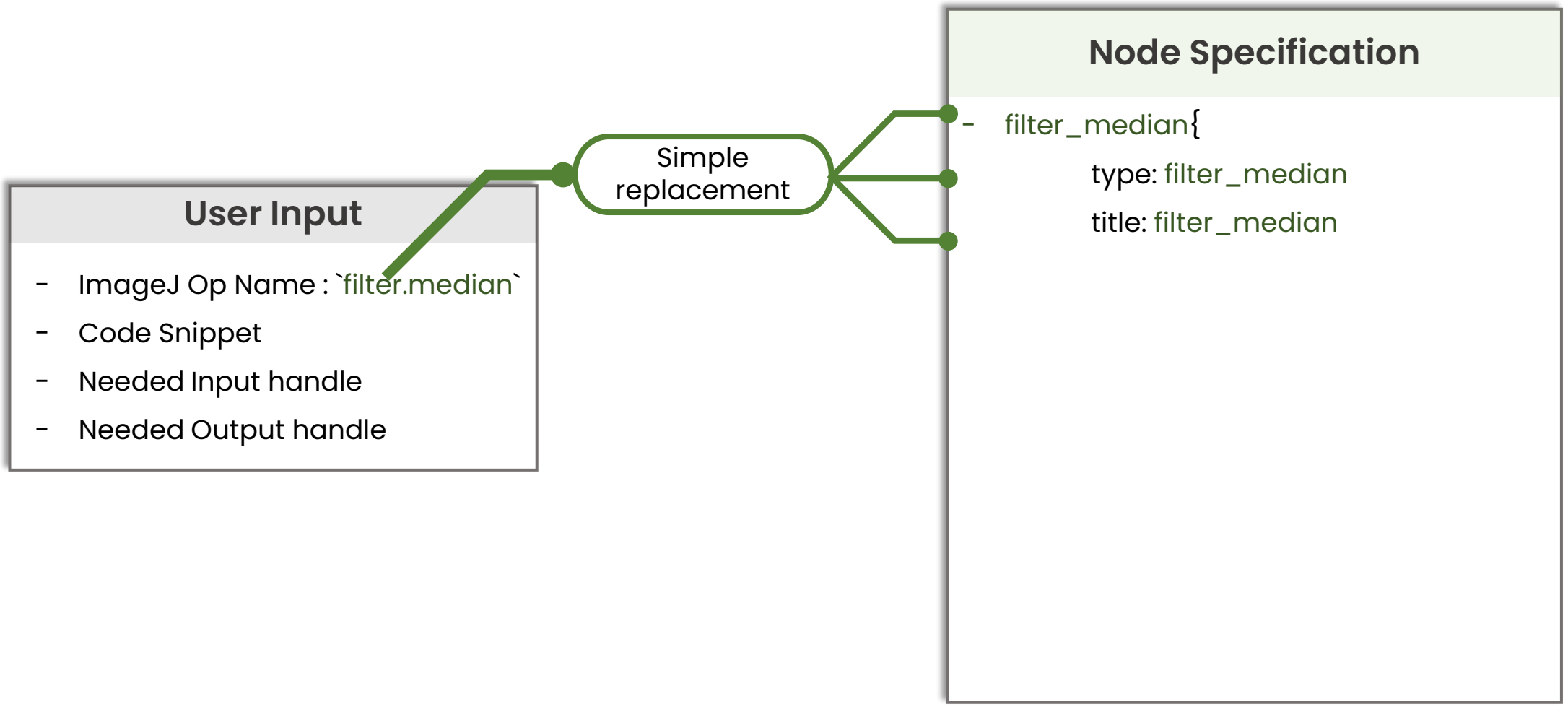
Methodology – Auto Generation for ImageJ



Methodology – Pipeline 1 Update Op name



Methodology - Pipeline 1 Update Op name



Methodology - Pipeline 2 Extract Imports

User Input

- ImageJ Op Name
- **Code Snippet**
- Needed Input handle
- Needed Output handle

Sample Code Snippet

```
- import imagej
- import scyjava as sj
- ij = imagej.init('sc.fiji:fiji:2.14.0')
- result_image =
  ij.op().run("create.img",jimage
- result_image= ij.op().run(Op_name,
  ij.py.jargs(result_image,
  process_image,
  HyperSphereShape(range),None))
```

AST code analysis

Node Specification

```
- filter_median{
      type: filter_median
      title: filter_median
      externalImports :
```

Methodology - Pipeline 2 Extract Imports

User Input

- ImageJ Op Name
- **Code Snippet**
- Needed Input handle
- Needed Output handle

Sample Code Snippet

- result_image = ij.op().run("create.img",jimage
- result_image= ij.op().run(Op_name, ij.py.jargs(result_image, process_image, HyperSphereShape(range),None))

AST code analysis

Node Specification

- filter_median{

  type: filter_median

  title: filter_median

  externalImports :
  - import imagej
  - import scyjava as sj
  - ij = imagej.init('sc.fiji:fiji:2.14.0')

Methodology - Pipeline 3 Format codeGenerator

User Input

- ImageJ Op Name
- **Code Snippet**
- **Needed Input handle**
- **Needed Output handle**

Sample Code Snippet

- result_image = ij.op().run("create.img",jimage
- result_image= ij.op().run(Op_name, ij.py.jargs(result_image, process_image, HyperSphereShape(range),None))

Specific code formatting

Node Specification

- filter_median{

  type: filter_median

  title: filter_median

  externalImports :
  - import imagej
  - import scyjava as sj
  - ij = imagej.init('sc.fiji:fiji:2.14.0')

  codeGenerator :

Methodology - Pipeline 4 Update Input/Output

User Input

- **ImageJ Op Name**
- **Code Snippet**
- Needed Input handle
- Needed Output handle

Static Code Snippet

```
- import imagej
- import scyjava as sj
- ij = imagej.init('sc.fiji:fiji:2.14.0')
- result_image =
  ij.op().run("create.img",jimage
- result_image= ij.op().run(Op_name,
  ij.py.jargs(result_image,
  process_image,
  HyperSphereShape(range),None))
```

ImageJ API: OpInfo[12]

AST code analysis

Op Matching

Node Specification

```
- filter_median{
      type: filter_median
      title: filter_median
      externalImports :
        - import imagej
        - import scyjava as sj
        - ij = imagej.init('sc.fiji:fiji:2.14.0')
      codeGenerator :
function codeGenerator(inputs,outputs,code){
return `
${output1} = ___(${input1},${input2})/n`
}
```

Methodology - Pipeline 4 Update Input/Output

User Input

- **ImageJ Op Name**
- **Code Snippet**
- Needed Input handle
- Needed Output handle

Static Code Snippet

```
- import imagej
- import scyjava as sj
- ij = imagej.init('sc.fiji:fiji:2.14.0')
- result_image =
  ij.op().run("create.img",jimage
- result_image= ij.op().run(Op_name,
  ij.py.jargs(result_image,
  process_image,
  HyperSphereShape(range),None))
```

ImageJ API: OpInfo[12]

AST code analysis

Op Matching

Node Specification

```
- filter_median{
      type: filter_median
      title: filter_median
      externalImports :
        - import imagej
        - import scyjava as sj
        - ij = imagej.init('sc.fiji:fiji:2.14.0')
      codeGenerator :
function codeGenerator(inputs,outputs,code){
return `
${output1} = ___(${input1},${input2})/n`
}
      inputs :{
        input_image{
            datatype: image
```

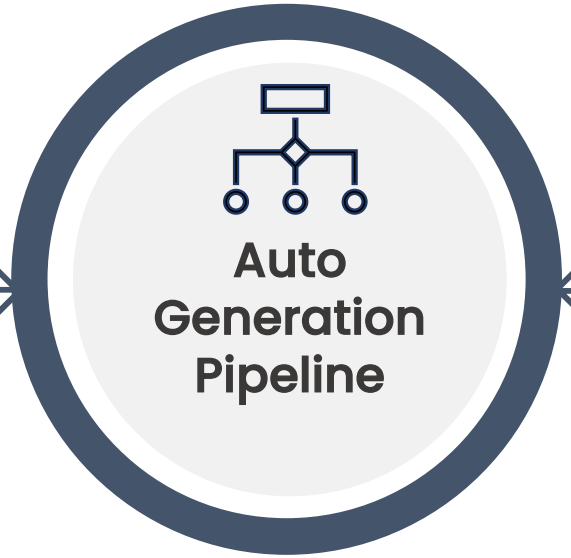
Methodology - Auto Generation for ImageJ

UNIVERSITÄT DES SAARLANDES

dfki ai

User Input

- ImageJ Op Name
- Code Snippet
- Needed Input handle
- Needed Output handle



Node Specification

```
- filter_median{
        type: filter_median
        title: filter_median
        externalImports :
          - import imagej
          - import scyjava as sj
          - ij = imagej.init('sc.fiji:fiji:2.14.0')
        codeGenerator :
function codeGenerator(inputs,outputs,code){
return `
${output1} =  _ _ _(${input1},${input2})/n`
}
        inputs :{
          input_image{
              datatype: image
```

Progress – Live Generation

ImageJ Op Node Auto-Generation

Details in Issue 72

Outline

- Background
- Motivation
- Methodology
- Progress Made
- Conclusion
- Next Step

Conclusion

We want

ImageJ Function + Chaldene System

= Familiar Functions in Good New Tool

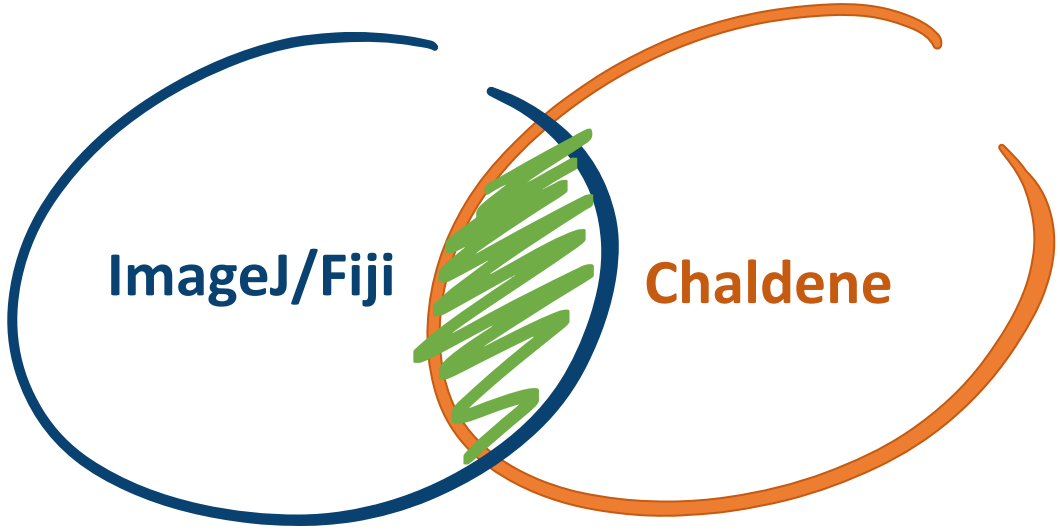
= Easy workflow sharing and visualization

= Expand boundary of ImageJ-confined scientists with limited programming skills [3]

Challenges

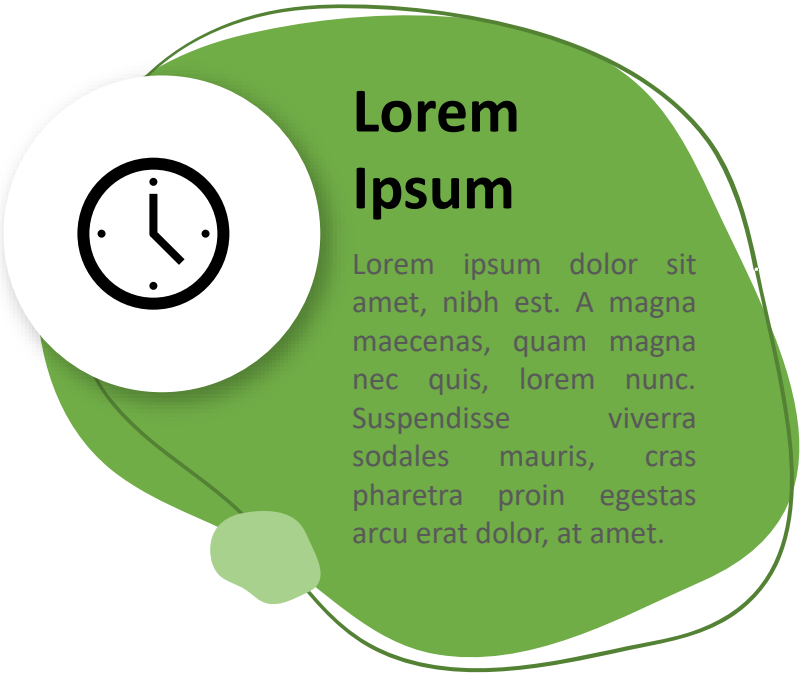
How to run Java-based ImageJ in Jupyterlab?

How to integrate ImageJ functionalities in Chaldene project?



Next Step

- Enhancement of Auto-Op-generation Pipeline
- Wrap up generation pipeline into a special Chaldene Node
- Add ImageJ image types support for Knowledge Graph in Chaldene Project



References

[1] Haase, R. Core ImageJ and other plugins. ImageJ API-beating: ImgLib2, ImageJ2 and the Big-Data Viewer. Training School, Luxembourg. 2019.

[2] Chen, Fei, et al. "Chaldene: Towards Visual Programming Image Processing in Jupyter Notebooks." 2022 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC). IEEE, 2022.

[3] Martins, Gabriel G., et al. "Highlights from the 2016-2020 NEUBIAS training schools for Bioimage Analysts: a success story and key asset for analysts and life scientists." F1000Research 10, 2021.

[4] "*Ganymede Java Kernel*". GitHub, owned by Allen Ball, 14.03.2024. https://github.com/allen-ball/ganymede

[5] "*IJava Java Kernel*". GitHub, owned by SpencerPark, 14.03.2024. https://github.com/SpencerPark/IJava

[6] "BeakerX Multi-language Kernel". GitHub, owned by twosigma, 14.03.2024. https://github.com/twosigma/beakerx

[7] "ImageJ Gateway". GitHub, owned by ImageJ, 14.03.2024. https://github.com/imagej/tutorials/blob/master/notebooks/1-Using-ImageJ/1-Fundamentals.ipynb

[8] Peng, Bo, et al. "SoS Notebook: an interactive multi-language data analysis environment." Bioinformatics 34.21 (2018): 3768-3770.

[9] Niephaus, Fabio, et al. "PolyJuS: a Squeak/Smalltalk-based polyglot notebook system for the GraalVM." Companion Proceedings of the 3rd International Conference on the Art, Science, and Engineering of Programming. 2019.

[10] Rueden, Curtis T., et al. "PyImageJ: A library for integrating ImageJ and Python." Nature methods 19.11 (2022): 1326-1327.

[11] "*Classic Segmentation Workflow with ImageJ2*." PyImageJ.14.03.2024. https://py.imagej.net/en/latest/Classic-Segmentation.html#segmentation-workflow-with-imagej2

[12] Figge, Marc, Ruman Gerst, and Zoltan Cseresnyes. "JIPipe: Visual batch processing for ImageJ." 2022.

Thank you very much!
Any questions?