Batch analysis and macro development in ImageJ/Fiji:
going beyond the basics

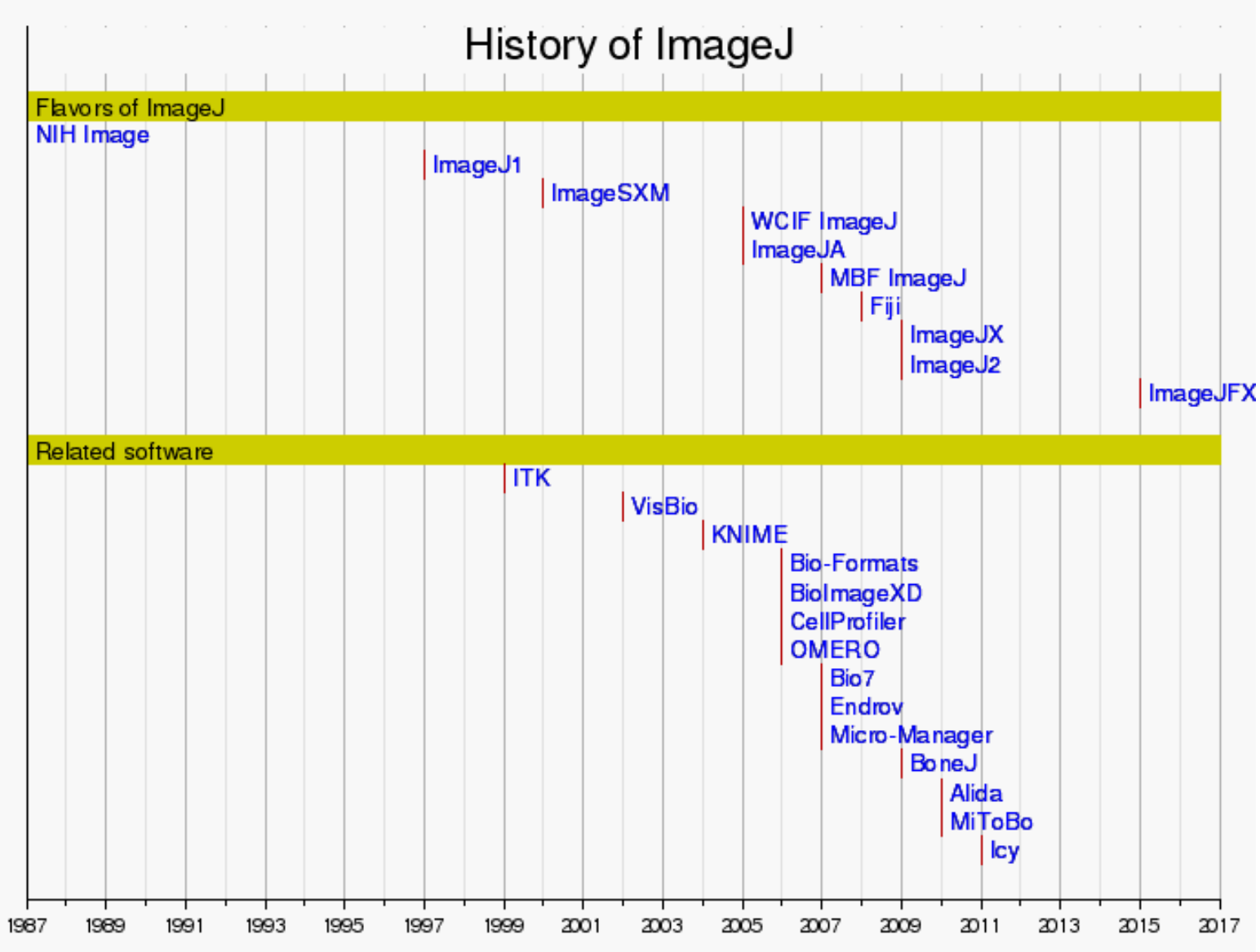
ImageJ2/ImgLib2 scripting

Mafalda Sousa
mafsousa@ibmc.up.pt

I3S, November 2019

Overview

- ImageJ history
- ImageJ2 motivation developement
- ImageJ2 ecosystem
- Basic concepts on object-oriented programming
- ImgLib2 concepts
- SciJava concepts
- Hands-on: Spot detection in multi-channel images



1987 NIH IMAGE

1997  **ImageJ1**

2007  **FIJI**

2009  **ImageJ2**

2011  **SciJava**

ImageJ2 motivation development

- Supporting new generation of image data:
  - Life time histograms across a range of spectral emission channels
  - High-throughput screens
  - Phase and frequency
  - Angles and rotations (light sheet microscopy)
- Enable new software collaboration:
  - facilitate ImageJ use as a software library and integration with external software.
- Broadening ImageJ community:
  - A tool for other scientific disciplines (astronomy, computer vision,…)

ImageJ2 ecosystem

SciJava

Independent from image processing
Standard java libraries common to many java external applications

ImageJ2 ecosystem

SciJava


ImageJ
(legacy layer)

Low level components establish image metadata and algorithms patterns built on SciJava and ImgLib2 layers
high level components that includes the UI

ImageJ2 ecosystem

SciJava

ImageJ
(legacy layer)

SCIFIO

Reading, writing and translating between image formats

ImageJ2 ecosystem

SciJava

ImageJ
(legacy layer)

SCIFIO

ImgLib2

All the image processing algorithms, independent of image types, source or organization

ImageJ2 ecosystem

SciJava

ImageJ
(legacy layer)

SCIFIO

ImgLib2

Open for Innovation ® KNIME

CellProfiler™ cell image analysis software

OME

Matlab

R

python™

TensorFlow

ImageJ2 components

ImageJ

Image-specific components

- ImageJ Common
- ImageJ Ops
- ImageJ Updater
- ImageJ Legacy
- SCIFIO

http://imagej.net/ImageJ2

SciJava

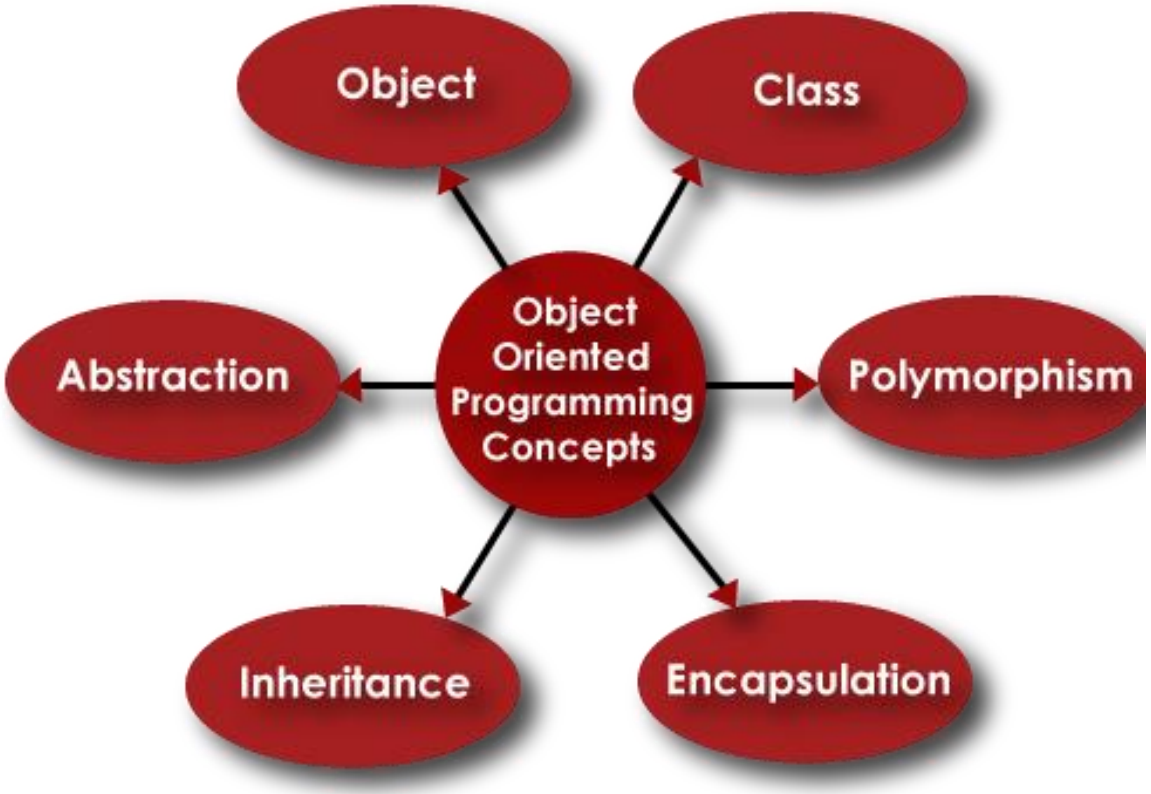
More general than images

- Application container
- Plugin framework
- Module framework
- Display and UI frameworks
- Scripting framework and plugins

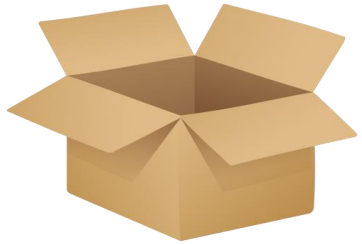
http://imagej.net/SciJava_Common

Object-oriented programming

- **Object** refers to a particular instance of a class where the **object** can be a combination of variables, functions, and data structures.
- A **class** is a collection of methods and variables. It is a blueprint that defines the data and behavior of a type.



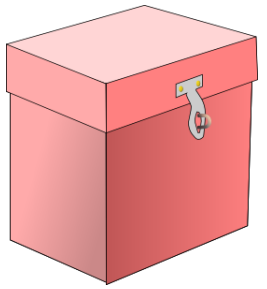
Class inheritance



```
class Box
```

- `open()`
- `close()`
- `putContent(Object content)`
- `getContent()`

- `isOpen()`
- `hasContent()`

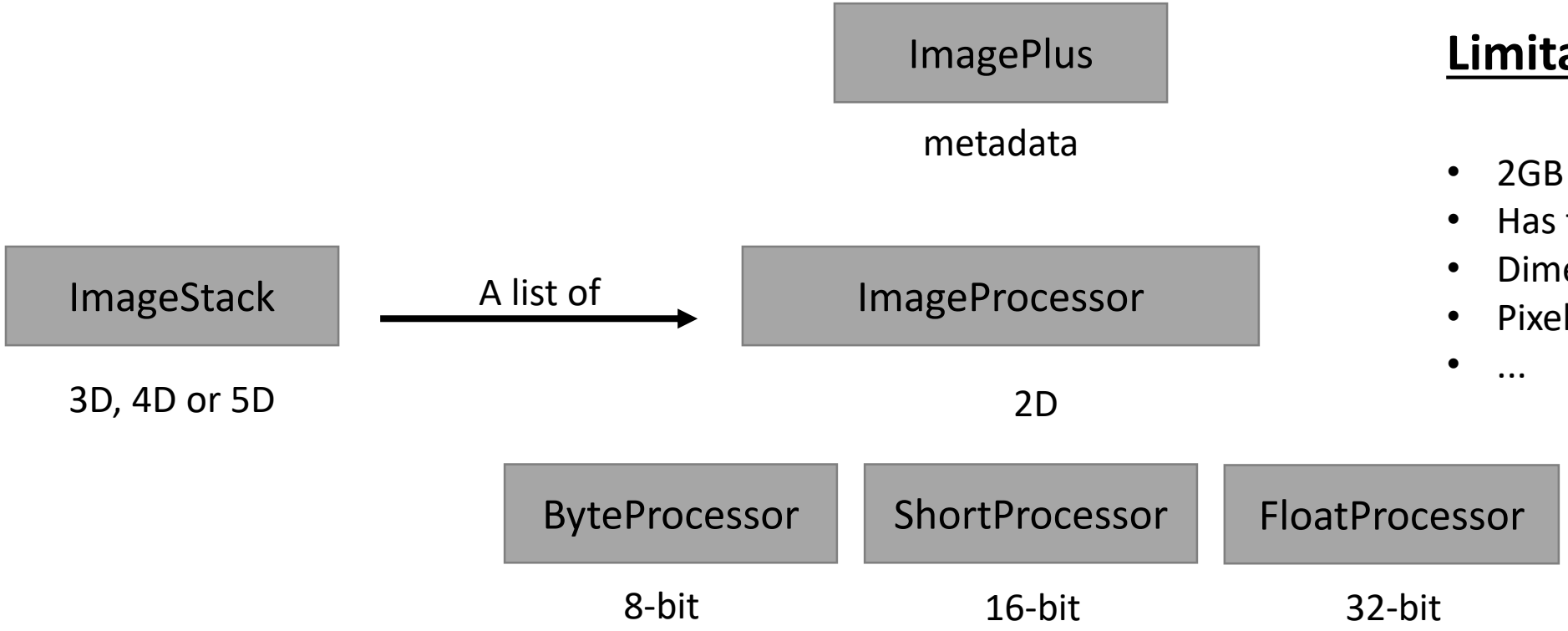


```
class LockableBox extends Box
```

- `lock()`
- `unlock()`

- `isLocked()`

ImageJ1 image objects



Limitations:

- 2GB per plane
- Has to fit into memory
- Dimensionality: XYCZT
- Pixel types: 8-, 16-, 32-bit
- ...

ImgLib2 design goals

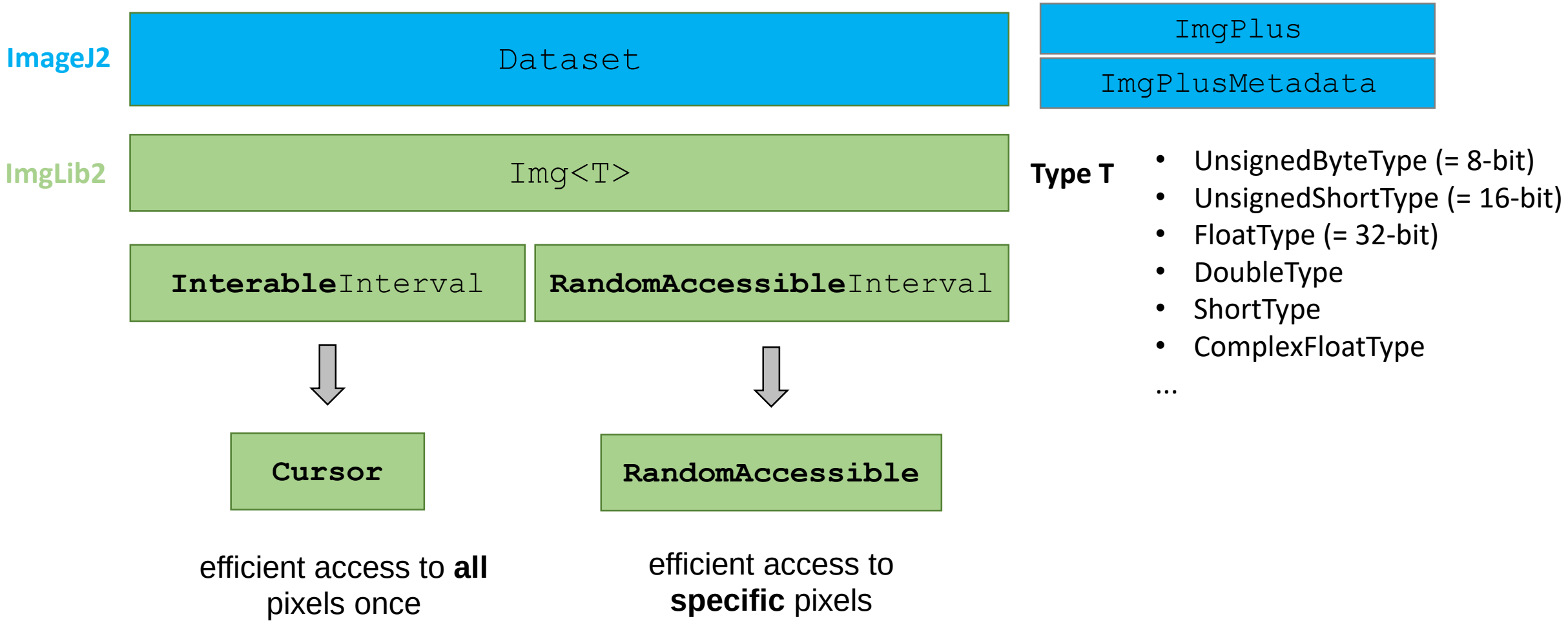
- Re-usability, avoid code duplication
- Separate algorithm development from data management
- High-level programming interface
- High performance
- Extensibility (adding algorithms, pixel type, storage strategies)
- Adaptability (to existing data structures)

As complicated as possible

ImgLib2 concepts

- **Accessibles**:
  - are images
  - Coordinates: integers or real-value types
  - Boundaries: bounded or infinite
  - Type of access: Random access or iterable
- **Accessors**:
  - Provide access to pixel values and coordinates
- **Types**:
  - describe algebraic properties of families of concrete types
  - E.g. Comparable types or NumericTypes support basic arithmetic operations (+,-,*,/).

ImgLib2 Concepts



ImgLib2 tutorials

- Learn more at:

https://nbviewer.jupyter.org/github/imagej/tutorials/blob/master/notebooks/3-Advanced-Usage/2-ImgLib2-in-Detail.ipynb

https://imagej.net/ImgLib2_Examples#Example_2_-_How_to_use_Cursor.2C_RandomAccess_and_Type

SciJava concepts

Services:

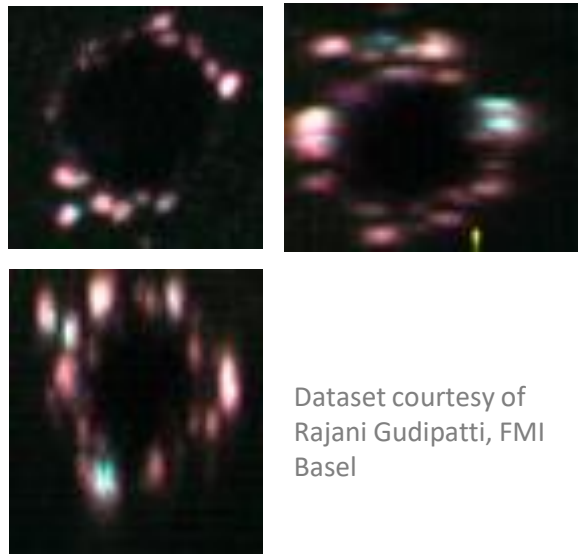
https://imagej.net/SciJava_Common#Services

- `IOService`
- `UIService`
- `ScriptService`
- `MenuService`
- `OpService`
- …

API documentation:

https://javadoc.scijava.org/SciJava/

Hands on

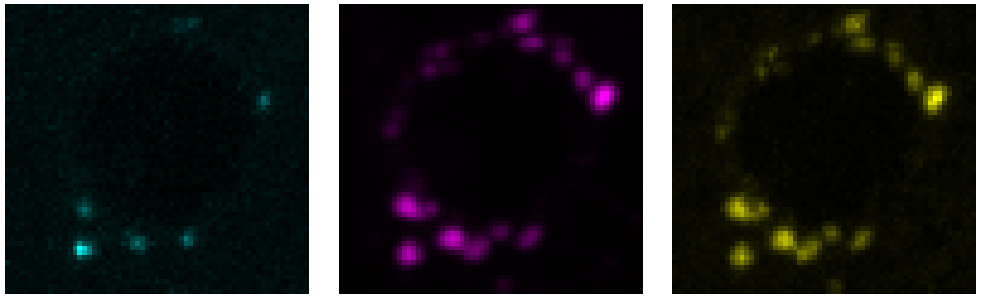


Dataset courtesy of Rajani Gudipatti, FMI Basel

3D spot detection in a 3-channel stack

DataSet:

Oocytes from C. elegans, stained with 3 different markers



Task:

For each channel, **detect foci** and report their **location**

Step-by-step

Step 1: Open one image

Step 2: Get axis (i.e. z and channel) metadata

Step 3: Extract single channel

Step 4: Apply Difference-of-Gaussians (DoG) filter

Step 5: Detect local maxima

Step 6: Add maxima coordinates to results table

Step 7: Repeat steps 3-6 for the other two channels

(Step 8: Plot the resulting coordinates)

Source code

3D spot detection in a 3-channel stack

https://github.com/mafsousa/Training-I3s-MacrosFiji-2019-hands-on

➢ Start by Task1, task 2… until solution

Technical Point

DefaultDataSet format

```
image = io.open(path)
image.class
image.getName()
image.dimensions(long[] dimensions)
image.getChannels()
```

Technical point

ImageJ Ops

- `Shift Alt L`: the Ops Finder

- Pluggins > Utilities > Find Ops…

```
dog =  ops.run("filter.dog", image, largeSigma, smallSigma)
```

Technical point

Detecting local maxima using ImgLib2 LocalExtrema

Find pixels that are extrema in their local neighborhood.

Static<P,T> List<P> = findLocalExtrema(RandomAccessibleInterval<T> source, LocalExtrema.LocalNeighborhoodCheck<P,T> localNeighborhoodCheck)

Returns a list of local maximum, similar to FindMaxima

image

MaximumCkeck(threshold)

Determine whether a pixel is a local extremum, that is, if its value is greater than or equal to a specified minimum allowed value (threshold), and no pixel in the neighborhood has a greater value.

Technical point

Populating the Results Table

ImageJ1 ResultsTable

```
#@ ResultsTable rt

rt.incrementCounter()          # Increments the measurement counter by one.
rt.addValue("Column", 1.2)     # Adds a value to the end of the given column.
rt.show("Results")
```