SLDC: an open-source workflow for object detection in multi-gigapixel images

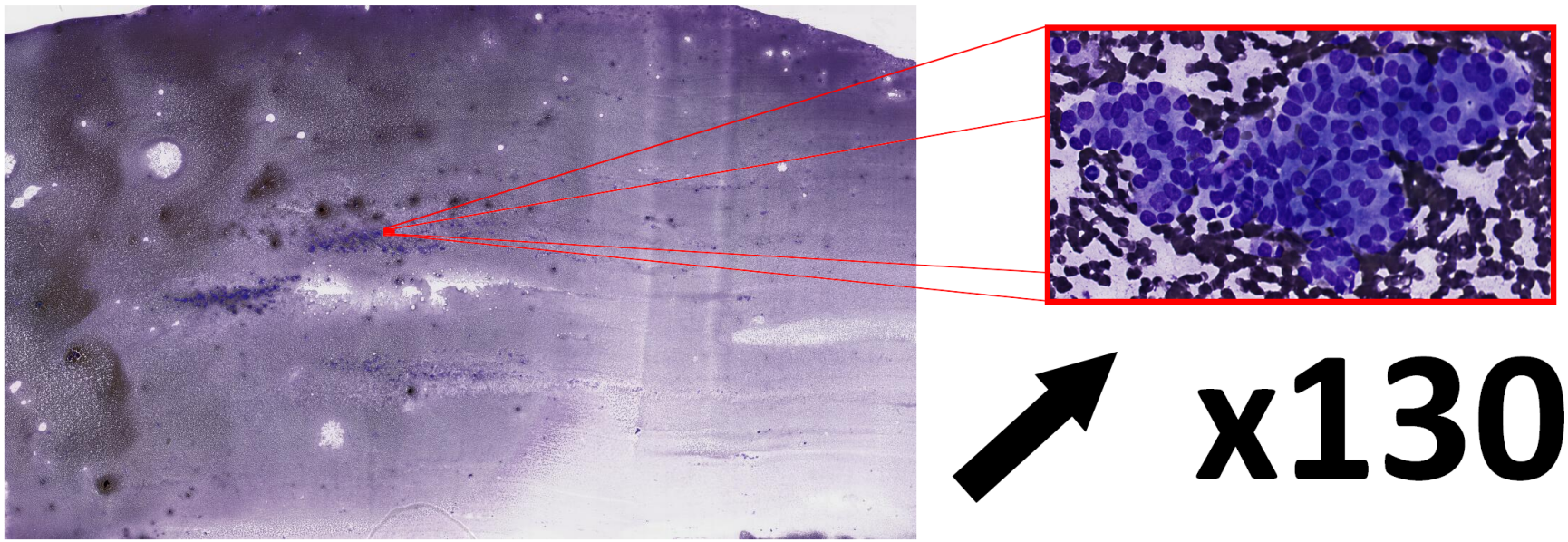
Romain Mormont, Jean-Michel Begon, Renaud Hoyoux, Raphaël Marée

Montefiore Institute, University of Liège, Belgium

7th September 2016

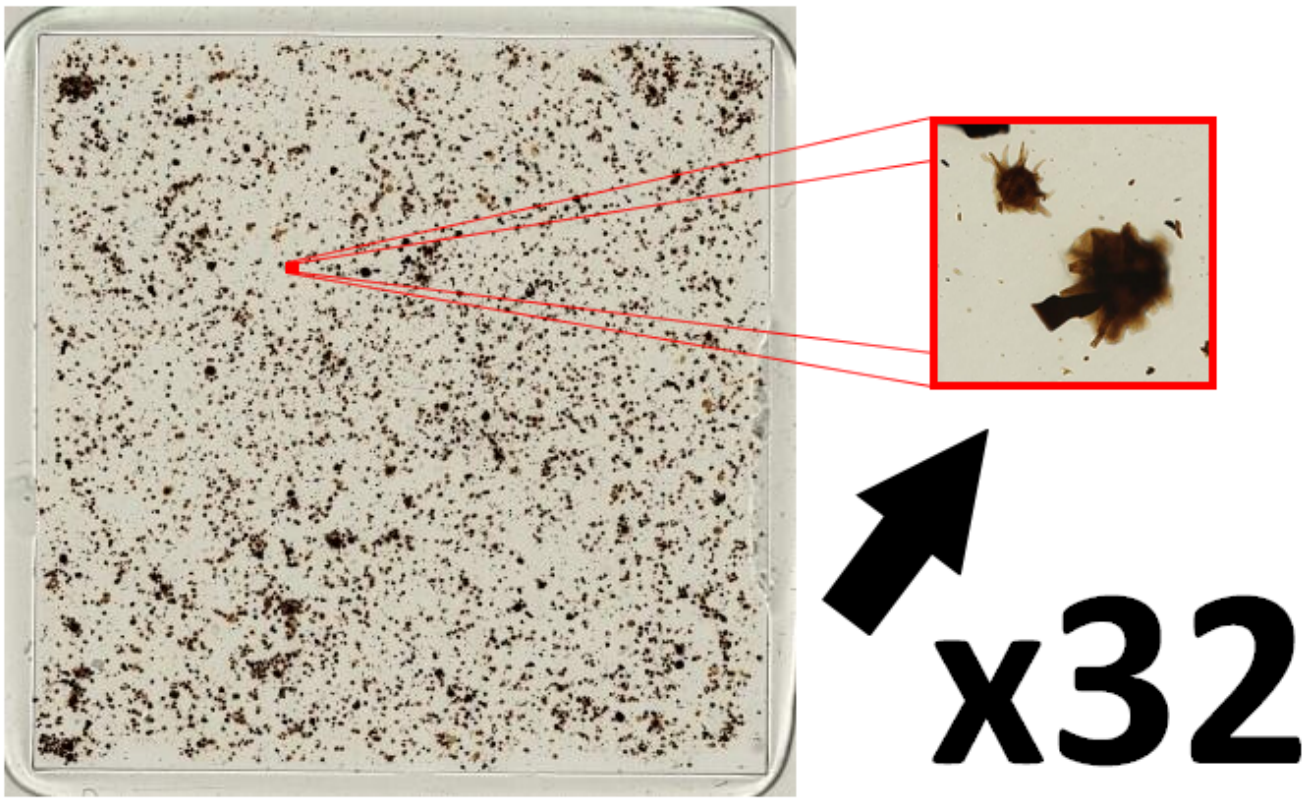
Outline

Context



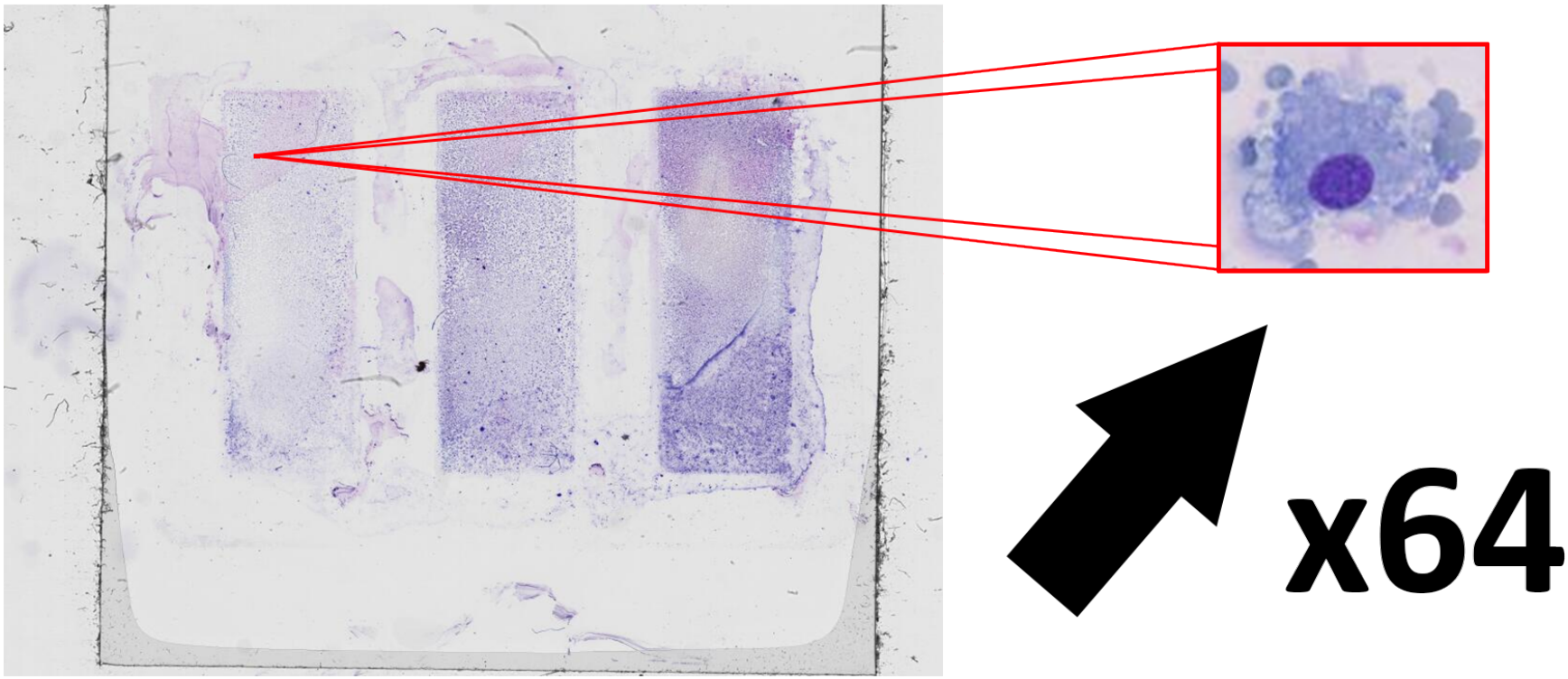
Microscope slide smeared with thyroid cell samples (15 gigapixels).

Context



Microscope slide smeared with core samples (11 gigapixels).

Context



Microscope slide smeared with lung cell samples (3 gigapixels).

Context

- Huge slides usually **analysed manually** !
- Machine learning (ML) and image processing (IP) could be used to assist humans
- Problems of **object detection and classification**

SLDC: framework

SLDC is an **open-source Python framework** created for accelerating development of large image analysis workflows.

How ?

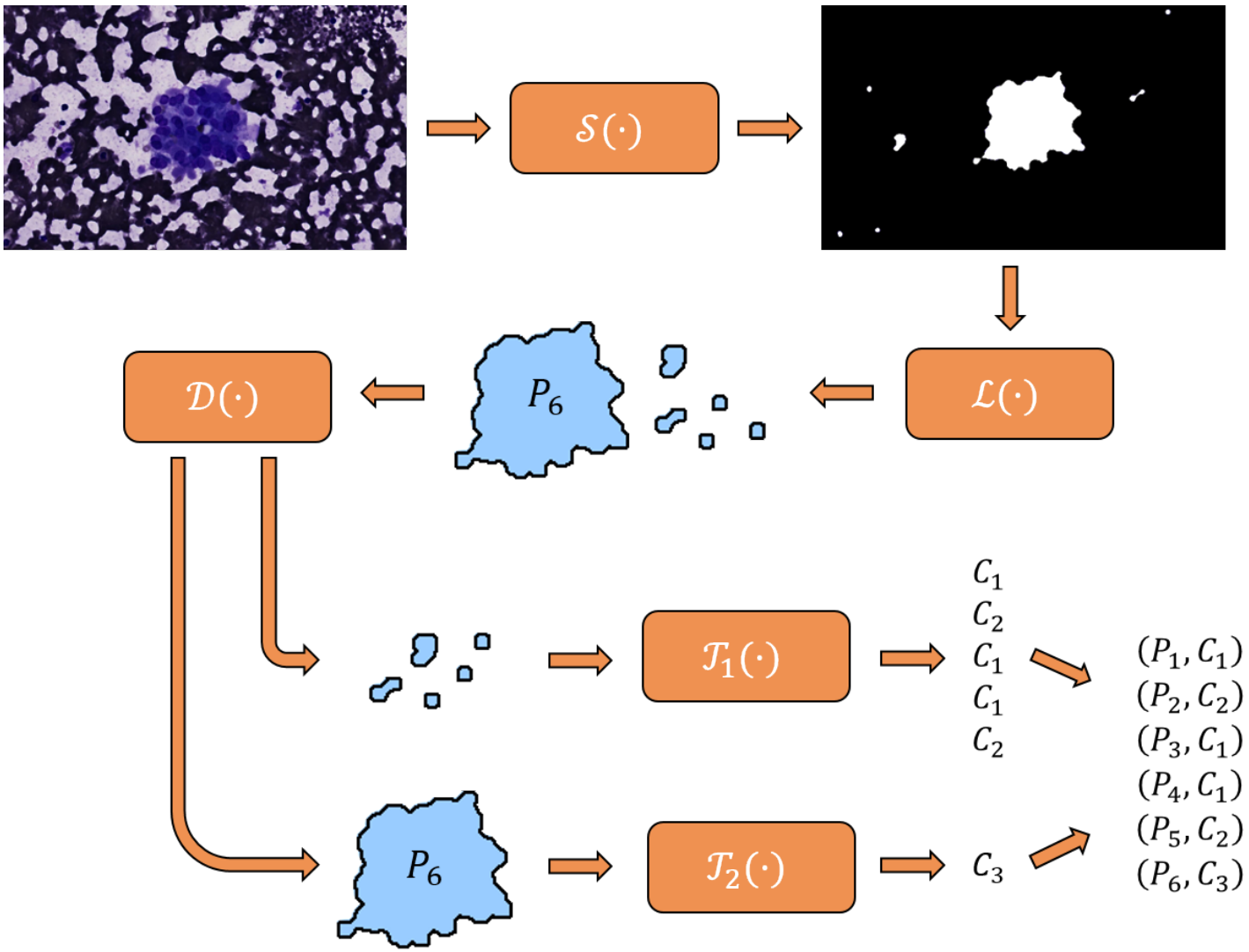
- It encapsulates problem-independent logic (parallelism, memory limitation due to large images handling,. . .)
- It provides a concise way of declaring problem dependant components (segmentation, object classification,. . .)

Where ? On GitHub at https://github.com/waliens/sldc

SLDC: features

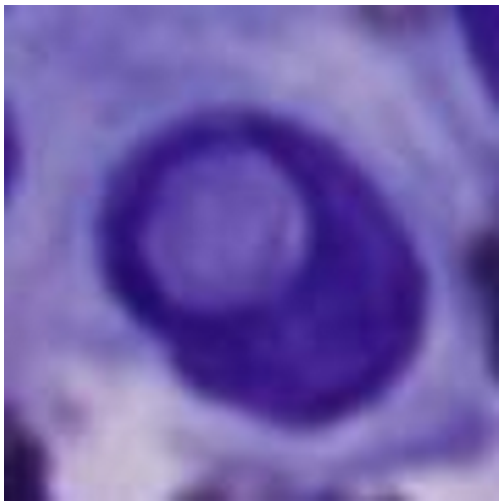
- **Tile-based processing** to avoid loading a full image into memory
- Several level of **parallelism**: tiles, objects, images,...
- A **customizable logging system** providing a rich feedback about the execution
- **Effortless integration** with other Python libraries: scikit-learn (ML), open-cv (IP), PyCuda (GPU),...
- **Builder components** providing an easy way of constructing complex workflows

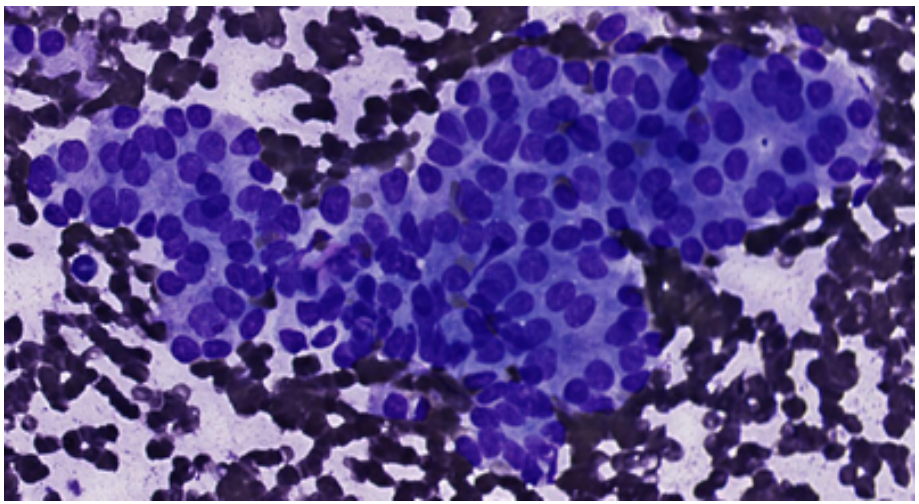
SLDC: how it works



SLDC at work

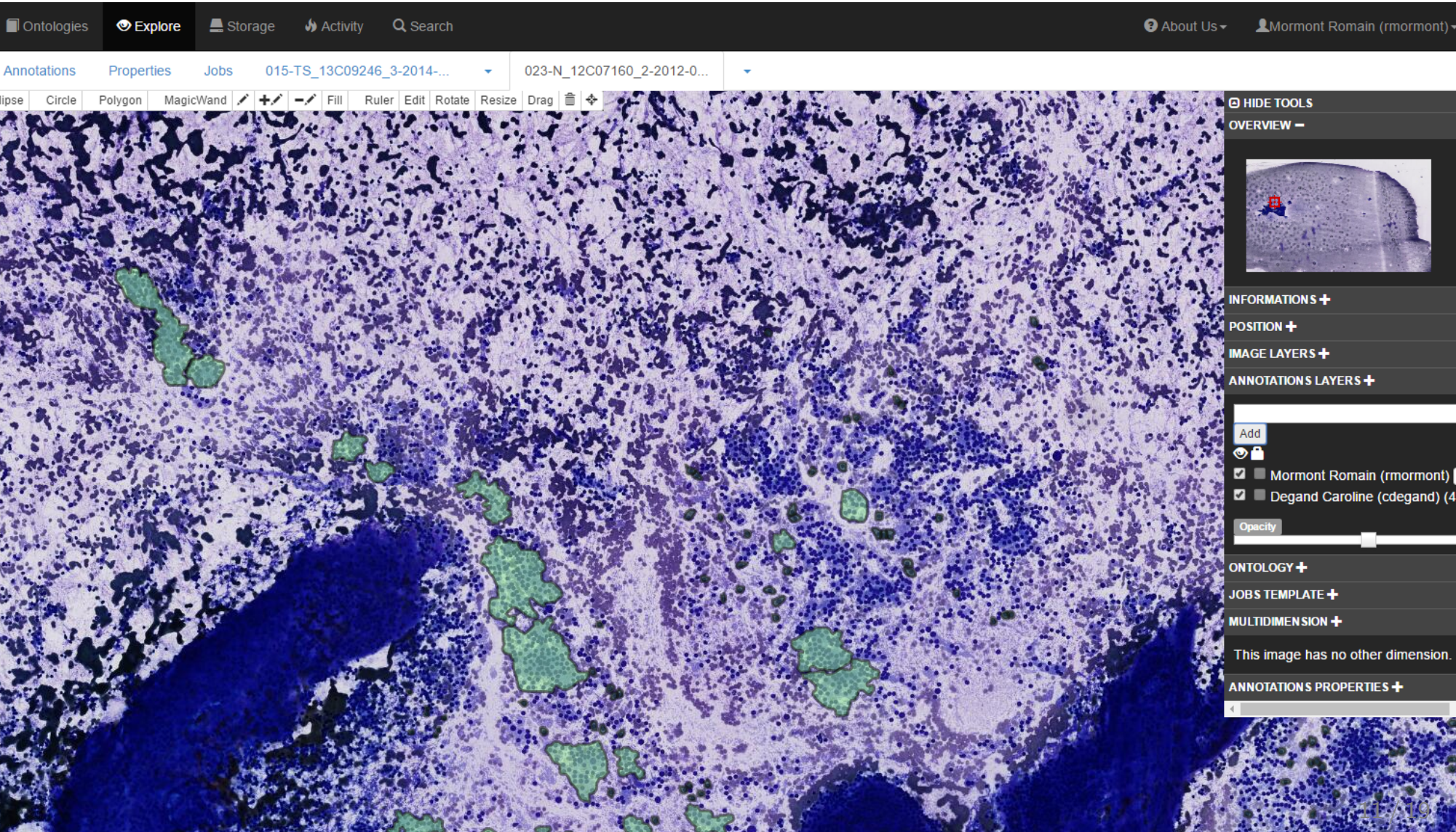
Aim: detect **cells with inclusion** and **proliferative architectural patterns**





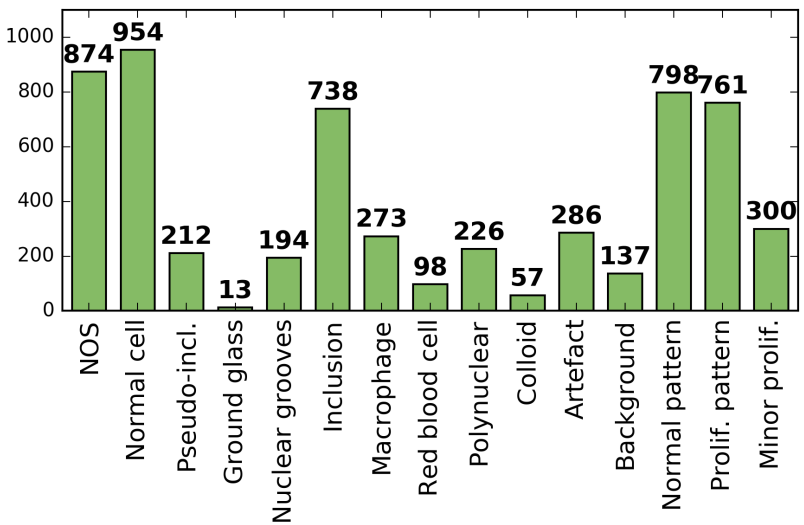
SLDC at work: Cytomine

cytomine is a web-based environment enabling collaborative multi-gigapixel images analysis. (Marée & al., Bioinformatics; 2016)

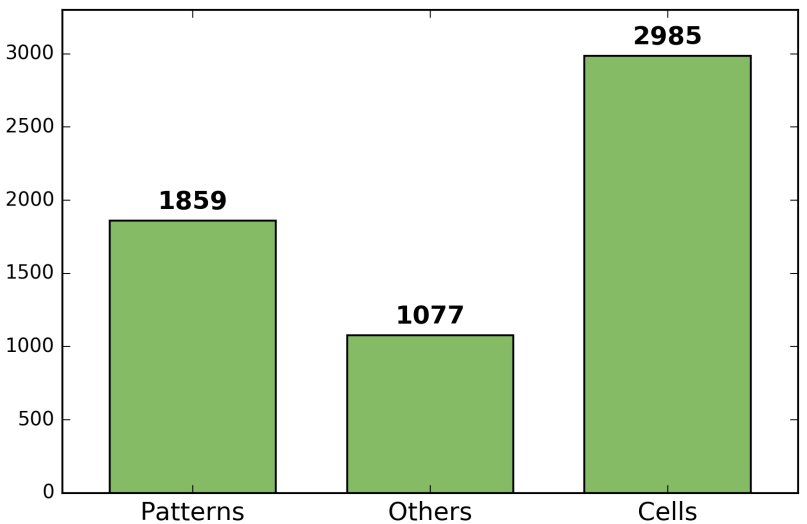


SLDC at work: data

- **84 images** with size ranging from 4 to 18 gigapixels
- **68 annotated images**
- **5921 labelled annotations** made by cytopathologists[1]



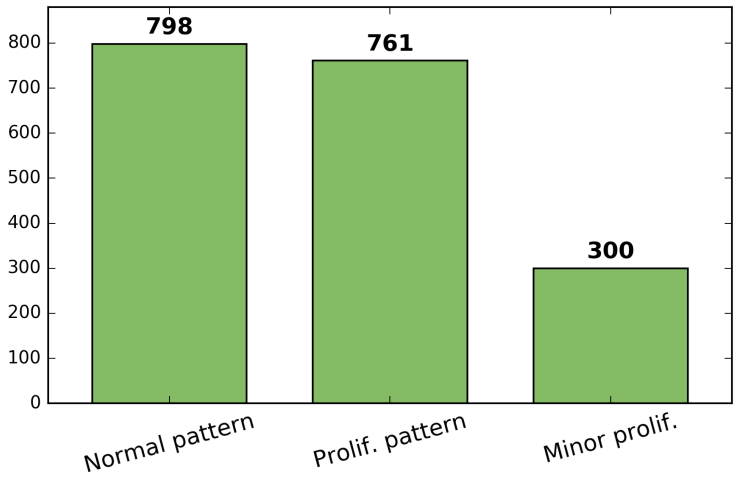
(a) Annot. per term



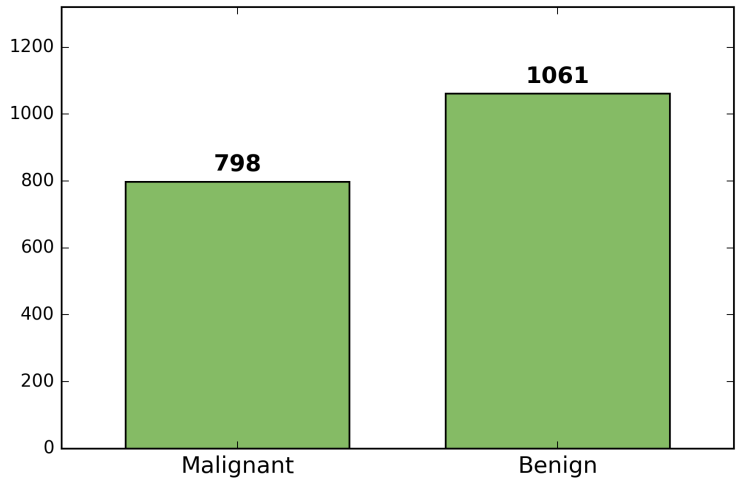
(b) Annot. per group

[1]Team of Pr. Isabelle Salmon, Department of Pathology, Faculty of Medecine, ULB

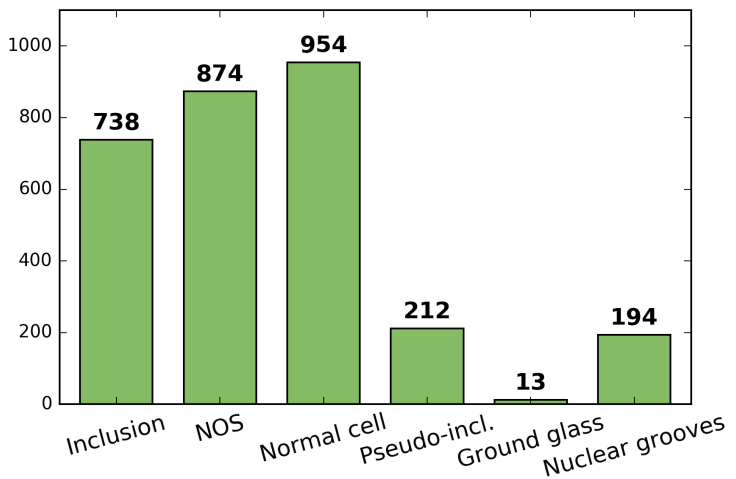
SLDC at work: data



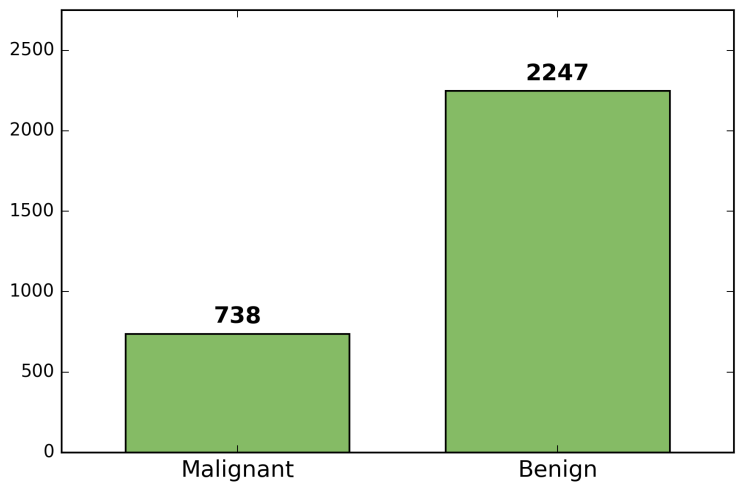
(c) Pattern annot. per term



(d) Pattern annot. per group

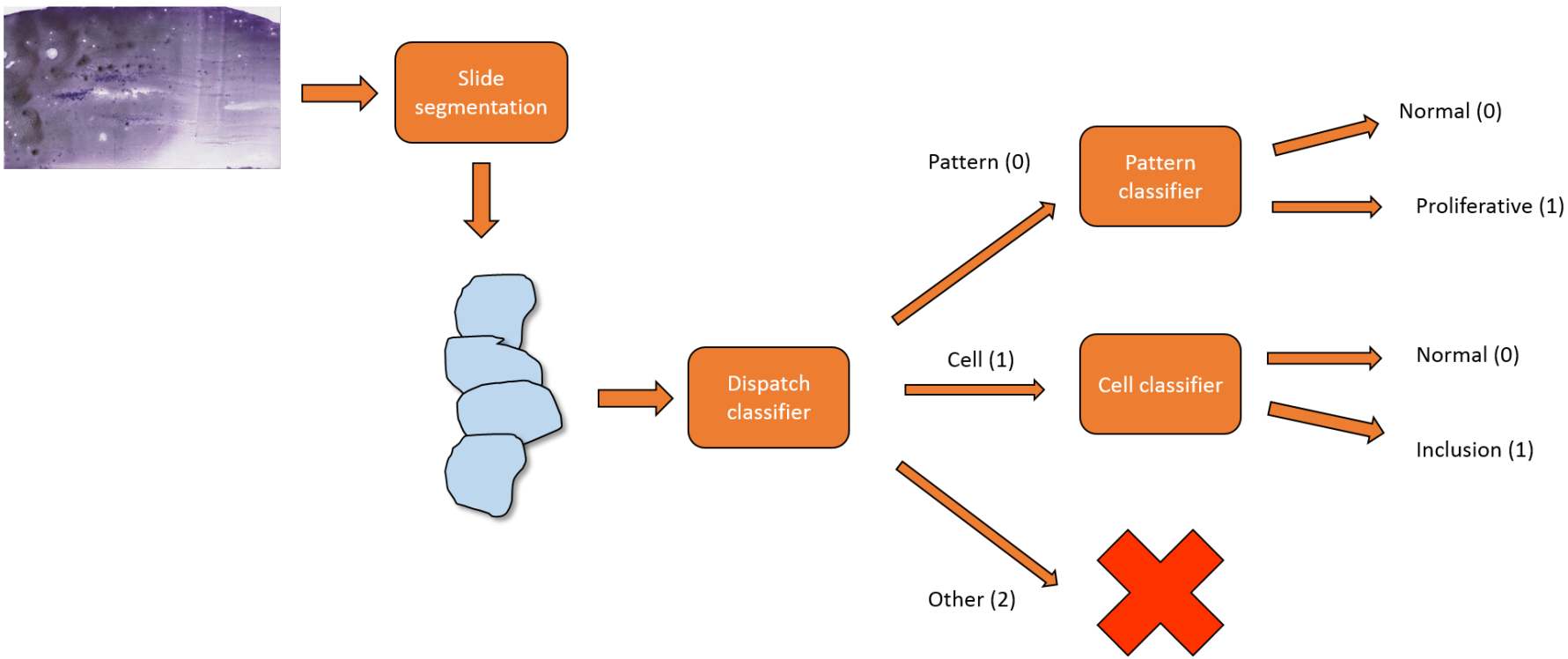


(e) Cell annot. per term

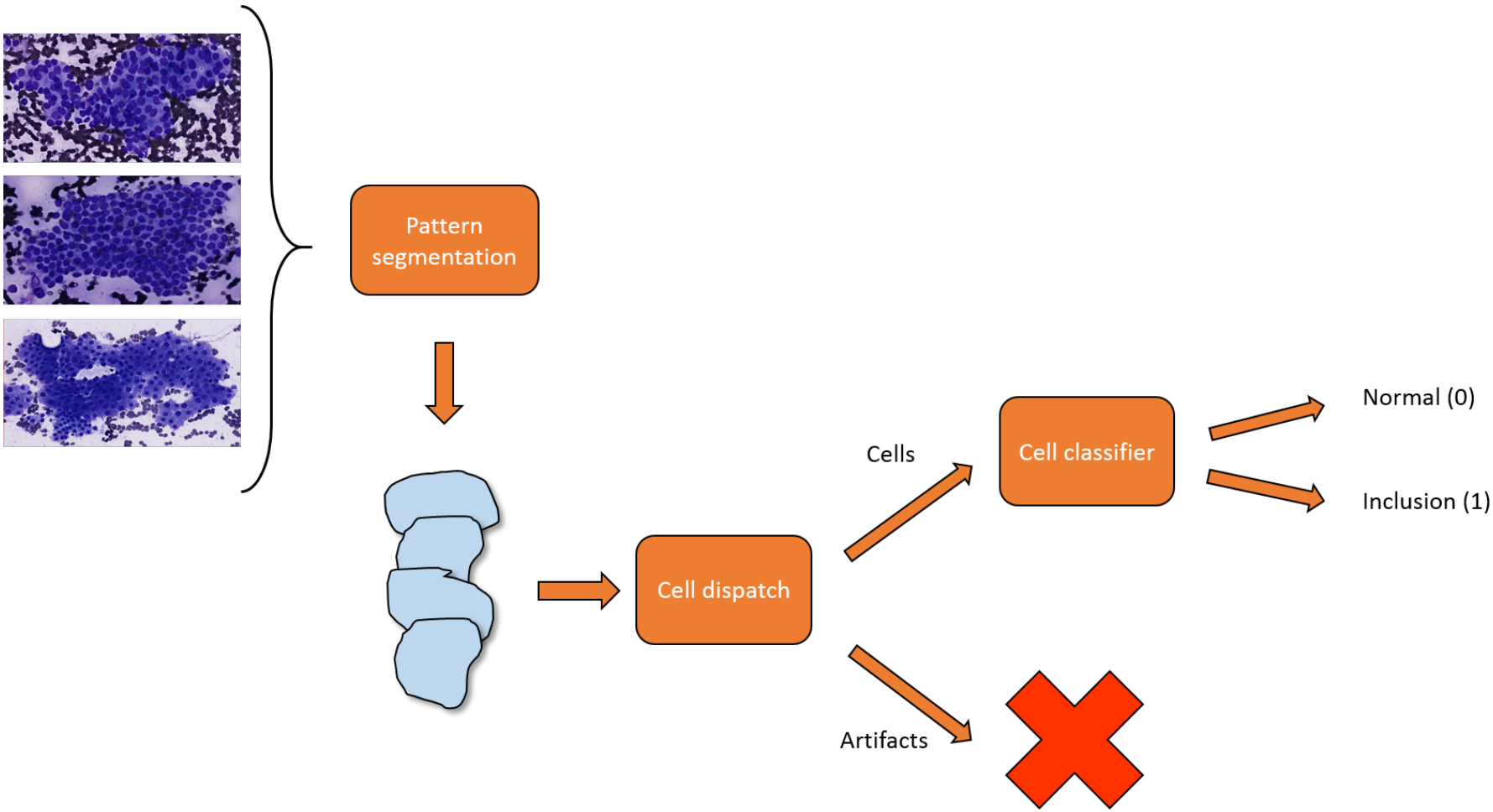


(f) Cell annot. per group

SLDC at work: workflow



SLDC at work: workflow (cont'd)



SLDC at work: workflow (cont'd)

Classification is performed based on the detected object's crop image using **random subwindows and extremely randomized trees**[2].

Cell with inclusion vs. normal cells:

Accuracy:	0.8523
Precision:	0.6310
Recall:	0.4930

	Normal	Inclusion
Normal	881	62
Inclusion	109	106

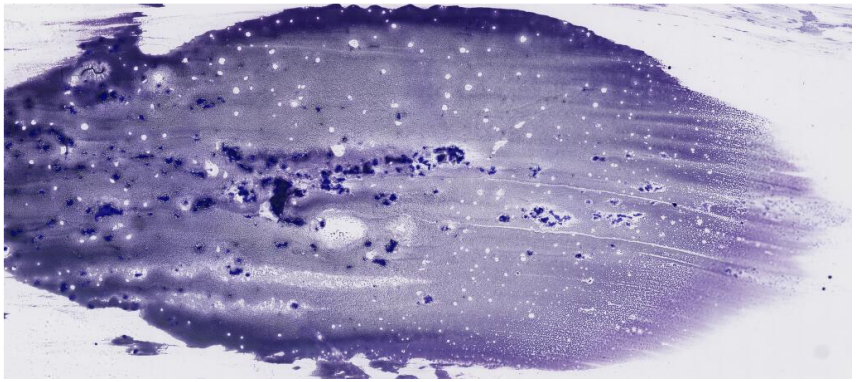
Proliferative vs. normal patterns:

Accuracy:	0.8625
Precision:	0.8363
Recall:	0.9493

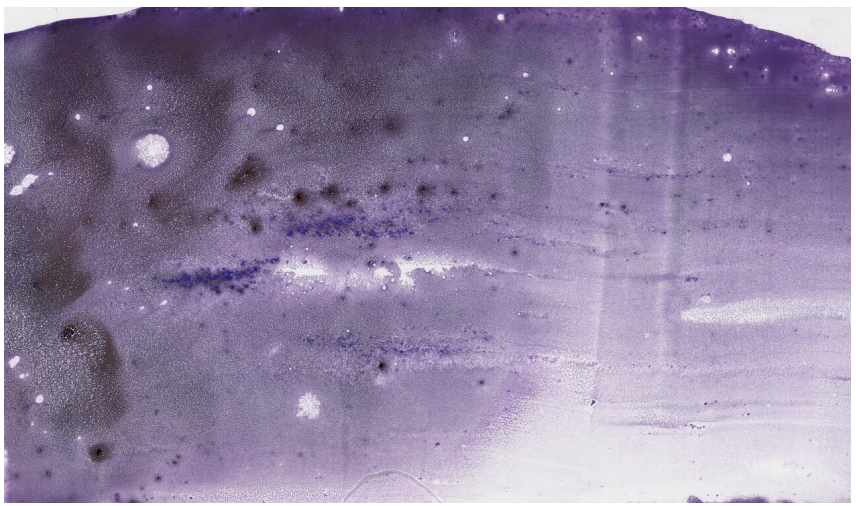
	Normal	Prolif.
Normal	158	55
Prolif.	15	281

[2] Marée et al., Pattern Recognition Letters ; 2016

SLDC at work: results



Size:	131072 × 57856
Objects found:	**20046**
Cells found:	18966
Patterns found:	1080
Time (1st pass):	**7 min 30 sec**
Time (2nd pass):	1 h 10 min



Size:	163840 × 95744
Objects found:	**79063**
Cells found:	72740
Patterns found:	6323
Time (1st pass):	**18 min 20 sec**
Time (2nd pass):	4 h 50 min

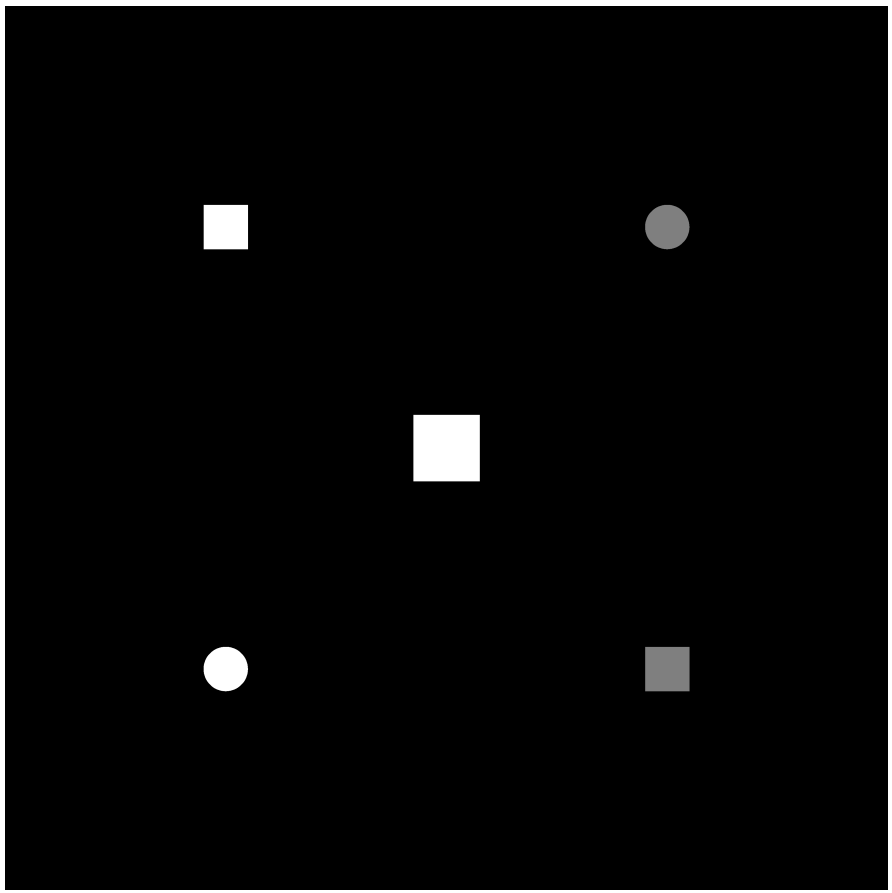
Future works

???

Thank you for your attention !
Any question ?

SLDC: toy example

The aim is to detect circles in the following image. As a bonus, we want to know their center color.



SLDC: toy example (cont'd)

```python
# Defining a segmenter
class CustomSegementer(Segmenter):
    """All non-black pixels are in an object of interest"""
    def segment(self, image):
        return (image > 0).astype(np.uint8)

# Defining a dispatching rule
class CircleRule(DispatchingRule):
    """A rule which matches circle polygons"""
    def evaluate_batch(self, image, polygons):
        return [circularity(p) > 0.85 for p in polygons]

# Defining a polygon classifier
class ColorClassifier(PolygonClassifier):
    """
    A classifier which returns the color (greyscale)
    of the center pixel of the object
    """
    def predict_batch(self, image, polygons):
        classes = [center_pxl_color(image, p) for p in polygons]
        probas = [1.0] * len(polygons)
        return classes, probas
```

SLDC: toy example (cont'd)

```
# Build the workflow
builder = WorkflowBuilder()
builder.set_n_jobs(100)
builder.set_segmenter(CustomSegementer())
builder.add_classifier(CircleRule(), ColorClassifier(), disp_label="circle")
workflow = builder.get()

# Process an image
results = workflow.process(image)

# Go through the detected objects
for polygon, dispatch, label, proba in results:
  print "Detected polygon {}".format(polygon)
  print "Dispatched by '{}'".format(dispatch)
  print "Predicted class {}".format(label)
  print "Probability {}".format(proba)
  print ""
```

SLDC: toy example (cont'd)

```
Detected polygon POLYGON ((...))
Dispatched by 'circle'
Predicted class 128
Probability 1.0

Detected polygon POLYGON ((...))
Dispatched by 'circle'
Predicted class 255
Probability 1.0
```