Image Processing Basics

Anja Neumann

With material from

Robert Haase, ScaDS.AI

Marcelo Leomil Zoccoler and Till Korten, PoL TU Dresden

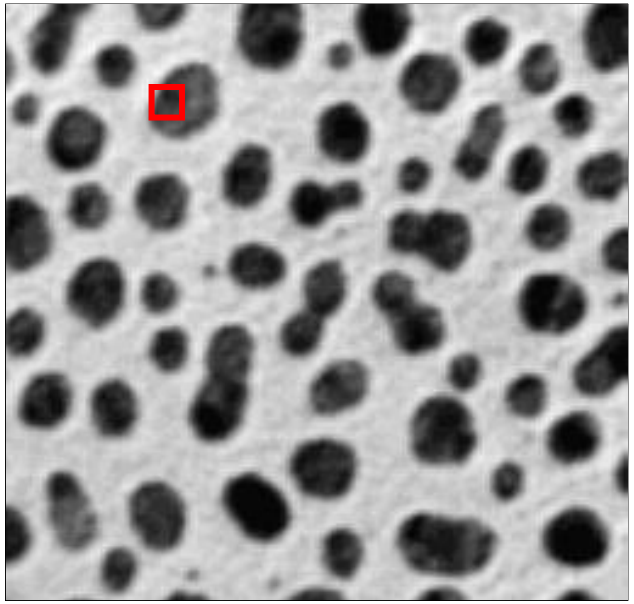
Mauricio Rocha Martins, Norden lab, MPI CBG

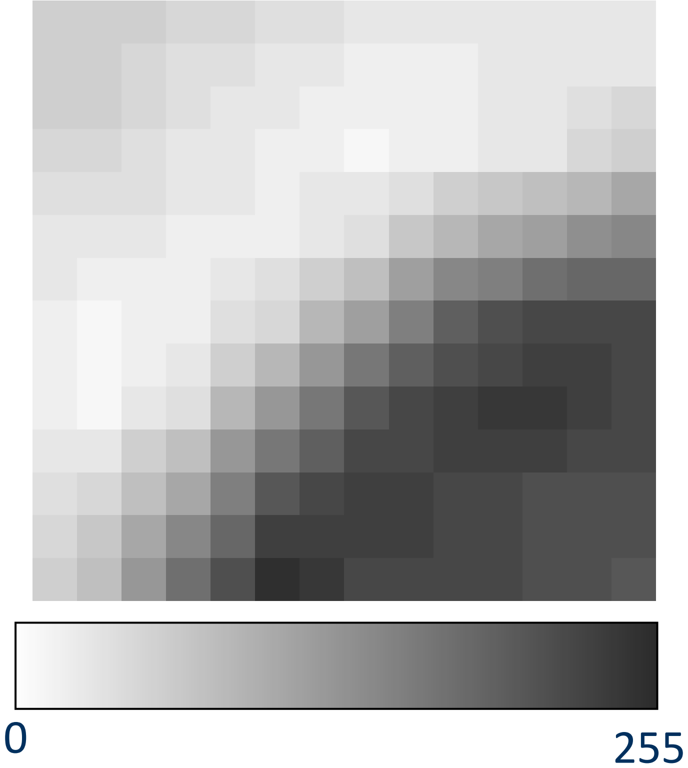
Dominic Waithe, Oxford University

Alex Bird, Dan White, MPI CBG

Images and pixels

- An image is just a matrix of numbers
- Pixel: “picture element”
- The edges between pixels are an artefact of the imaging / digitization. They are not real!

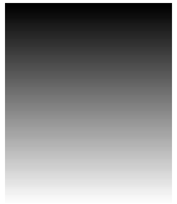


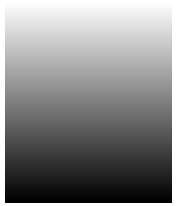


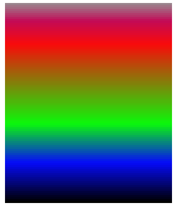
48	48	48	40	40	32	32	24	24	24	24	24	24	24
48	48	40	32	32	24	24	16	16	16	24	24	24	24
48	48	40	32	24	24	16	16	16	16	24	24	32	40
40	40	32	24	24	16	16	8	16	16	24	24	40	48
32	32	32	24	24	16	24	24	32	48	56	64	72	88
24	24	24	16	16	16	24	32	56	72	88	96	112	120
24	16	16	16	24	32	48	64	96	120	128	144	152	152
16	8	16	16	32	40	72	96	128	160	176	184	184	184
16	8	16	24	48	72	104	136	160	176	184	192	192	184
16	8	24	32	72	104	136	168	184	192	200	200	192	184
24	24	48	64	104	136	160	184	184	192	192	192	184	184
32	40	64	88	128	168	184	192	192	184	184	176	176	176
40	56	88	120	152	192	192	192	192	184	184	176	176	176
48	64	104	144	176	208	200	184	184	184	184	176	176	168

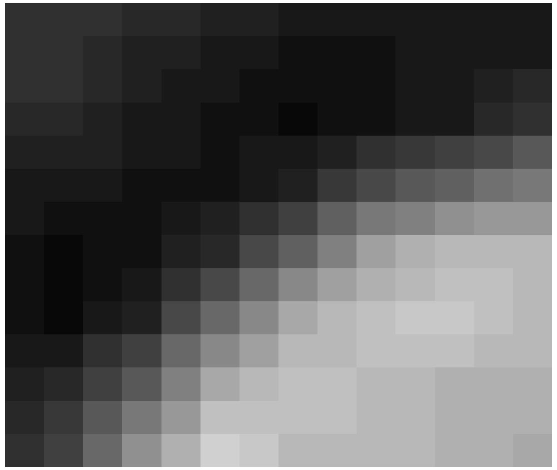
Colormaps / lookup tables

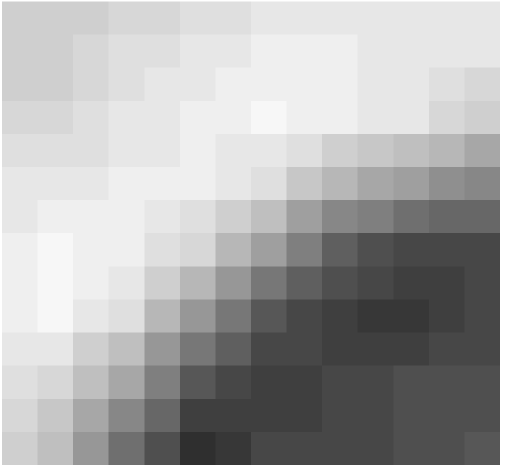
- The lookup table decides how the image is displayed on screen.
- Applying a different lookup table does not change the image. All pixel values stay the same, they just appear differently

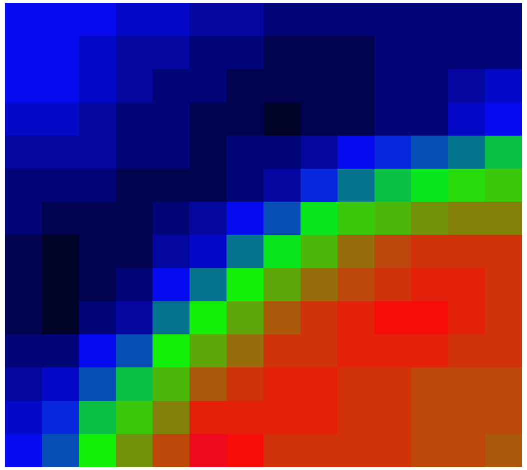
Pixel value	Display color
0 1 2 ... 255	

Pixel value	Display color
0 1 2 ... 255	

Pixel value	Display color
0 1 2 ... 255	

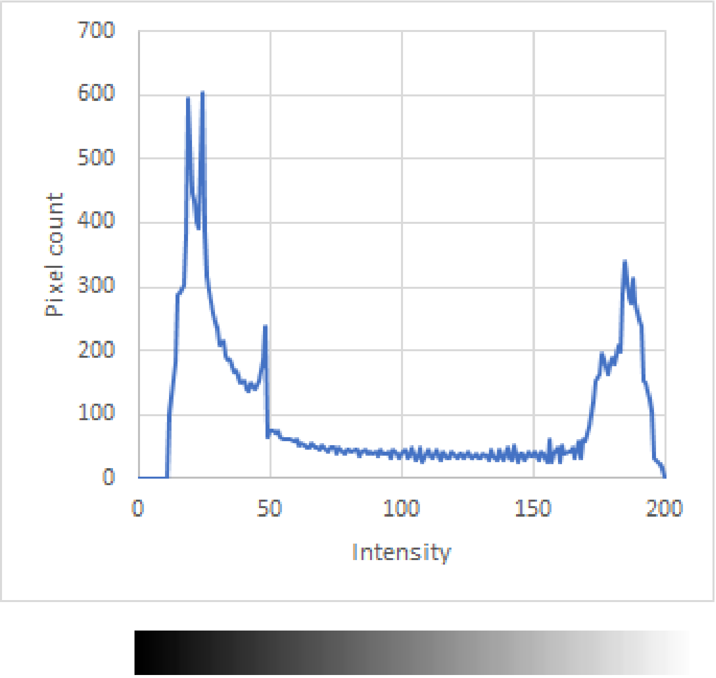


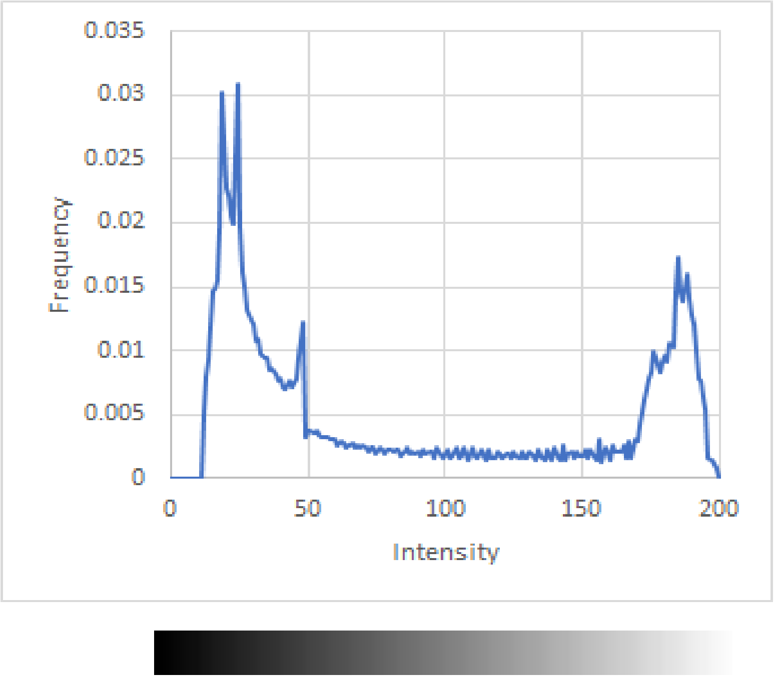




Histograms

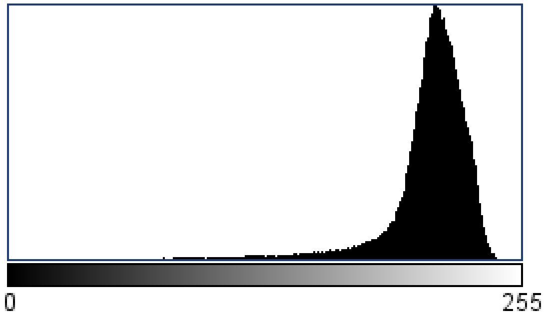
- A histogram shows the probability distribution of pixel intensities.
- The probability of a pixel having a certain grey value can be measured by counting pixels and calculating the frequency of the given intensity.
- Whenever you see a histogram, try to imagine the lookup-table on the X-axis





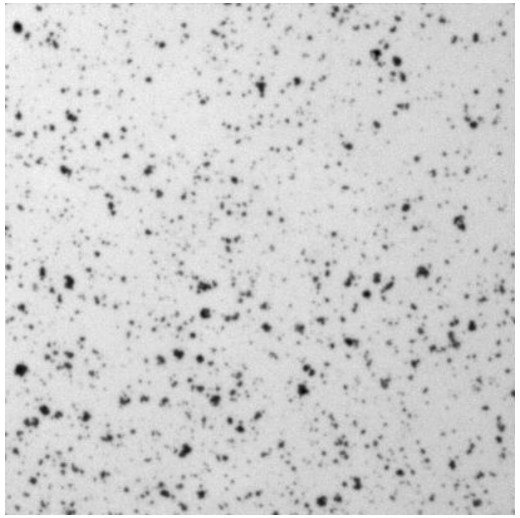
Histograms

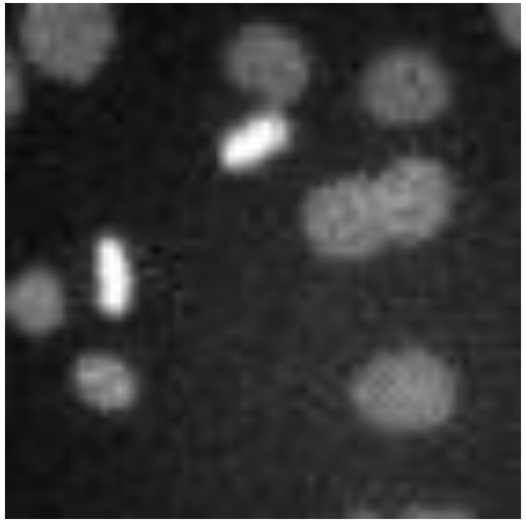
- To which of the three images does this histogram belong to?



N: 165648
Mean: 207.819
StdDev: 25.834
Value: 200

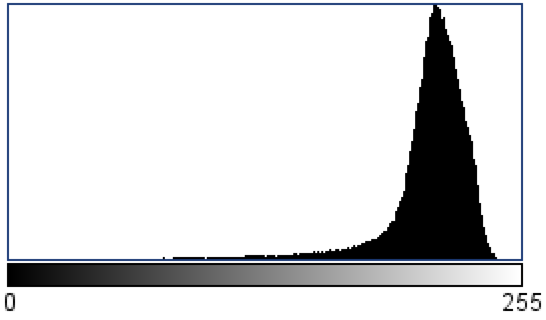
Min: 1
Max: 253
Mode: 212 (5234)
Count: 2219





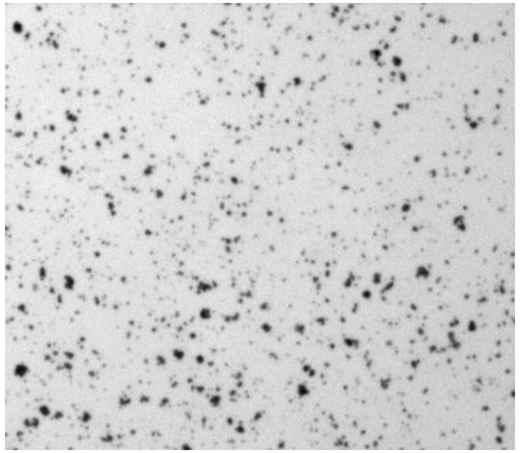
Histograms

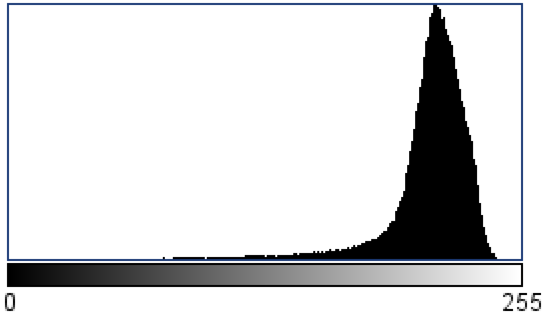
- To which of the three images does this histogram belong to?



N: 165648
Mean: 207.819
StdDev: 25.834
Value: 200

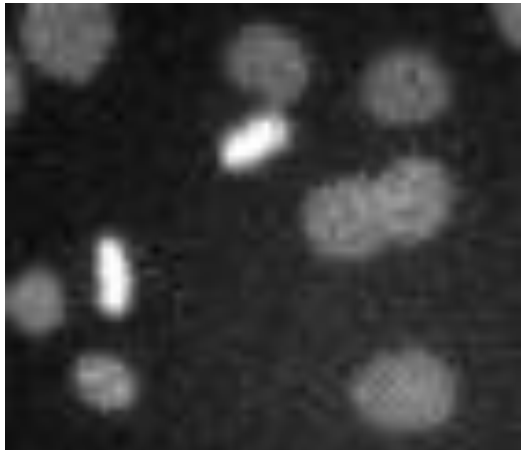
Min: 1
Max: 253
Mode: 212 (5234)
Count: 2219

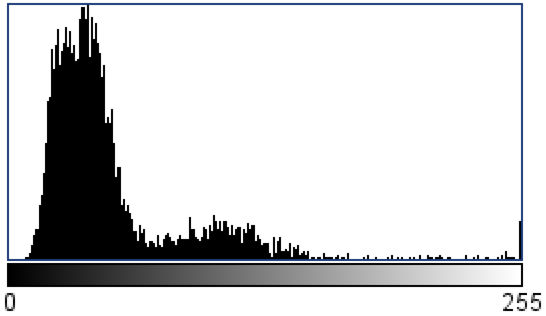




N: 165648
Mean: 207.819
StdDev: 25.834
Value: 200

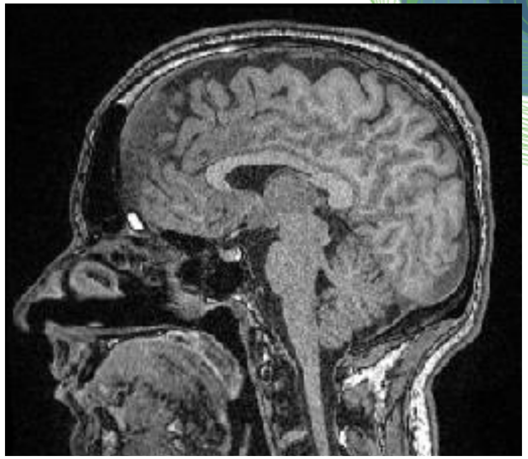
Min: 1
Max: 253
Mode: 212 (5234)
Count: 2219

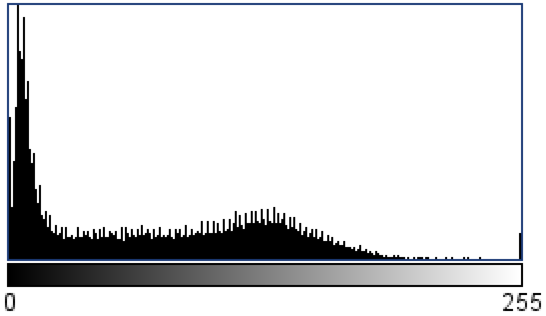




N: 4900
Mean: 51.060
StdDev: 36.426
Value: 64

Min: 8
Max: 255
Mode: 39 (125)
Count: 9





N: 65536
Mean: 70.929
StdDev: 59.567
Value: 59

Min: 0
Max: 255
Mode: 4 (2352)
Count: 239

Image Filtering

With material from

Robert Haase, Marcelo Leomil Zoccoler and Till Korten, PoL, TU Dresden

Filters

- An image processing filter is an operation on an image.
- It takes an image and produces a new image out of it.
- Filters change pixel values.

- There is no "best" filter. Which filter fits your needs, depends on the context.
- Filters do not do magic. They can not make things visible which are not in the image.

- Application examples
  - Noise-reduction
  - Artefact-removal
  - Contrast enhancement
  - Correct uneven illumination

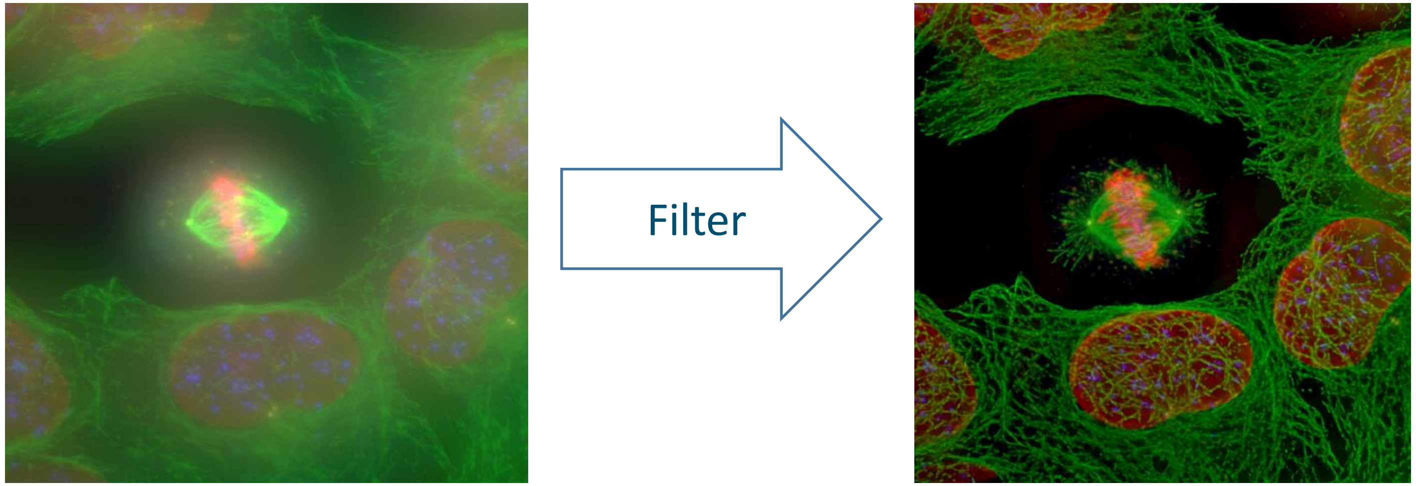
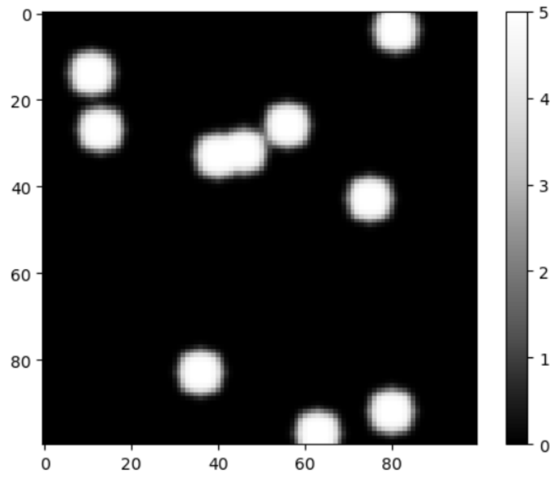


Image source: Alex Bird / Dan White MPI CBG

Effects harming image quality

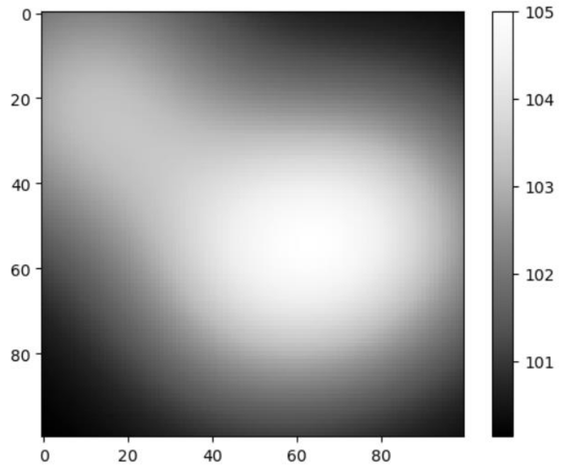
- Image formation (simulated)

“nuclei”



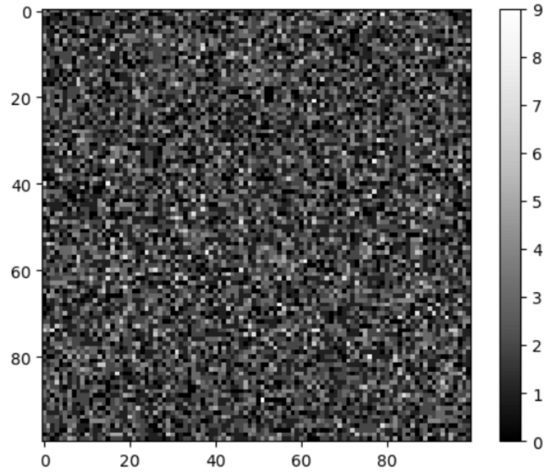
- Aberrations, defocus
- Motion blur

“background”



- Light from objects behind and in front of the scene (out-of-focus light)
- Dirt on the object slide
- Camera offset

“noise”

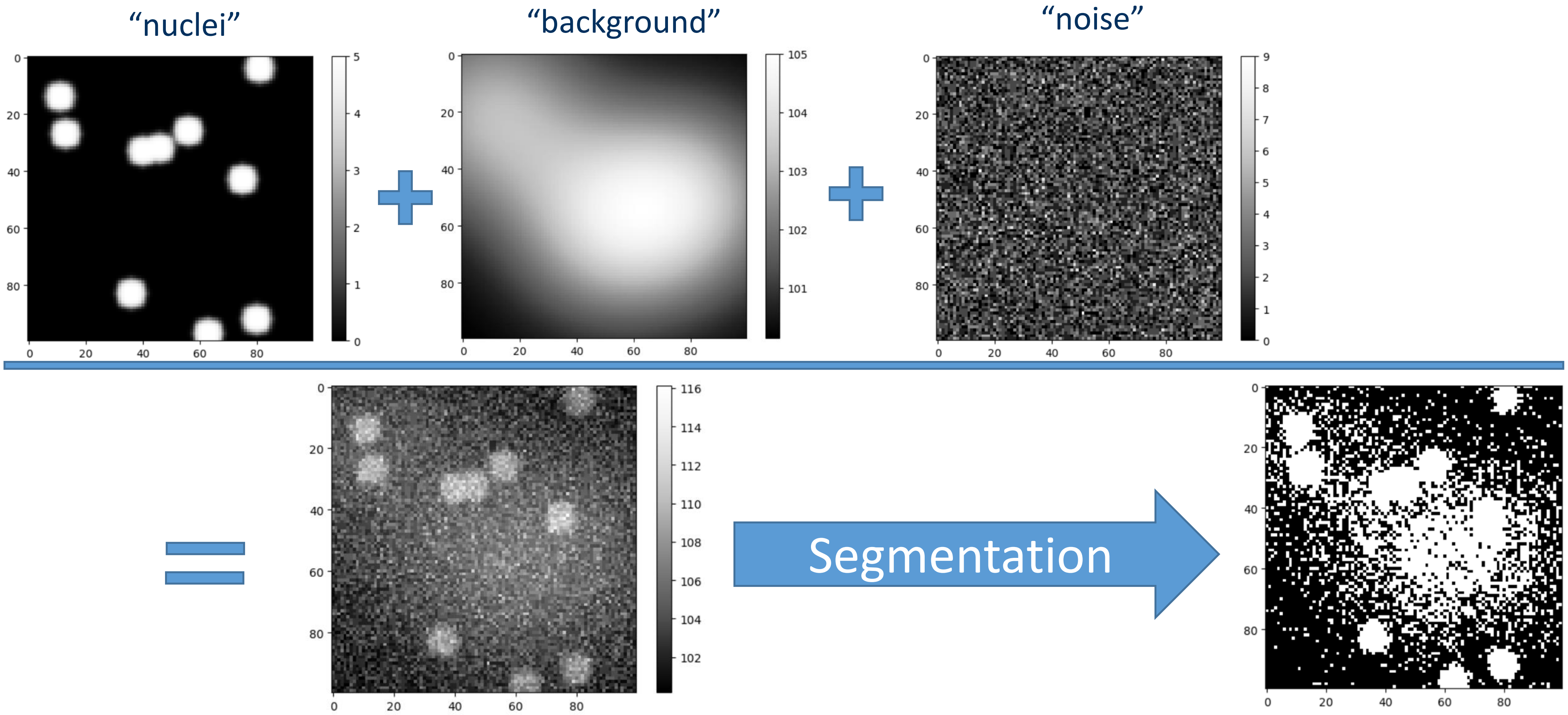


- Shot noise (arriving photons)
- Dark noise (electrons made from photons)
- Read-out-noise (electronics)

https://github.com/BiAPoL/Bio-image_Analysis_with_Python/blob/49a787514a367829c3e0e1832f6cc533e96d549f/03_image_processing/simulated_dataset.ipynb

Effects harming image quality

- Image formation (simulated)



https://github.com/BiAPoL/Bio-image_Analysis_with_Python/blob/49a787514a367829c3e0e1832f6cc533e96d549f/03_image_processing/simulated_dataset.ipynb

Image filtering

- We need to remove the noise to help the computer *interpreting* the image

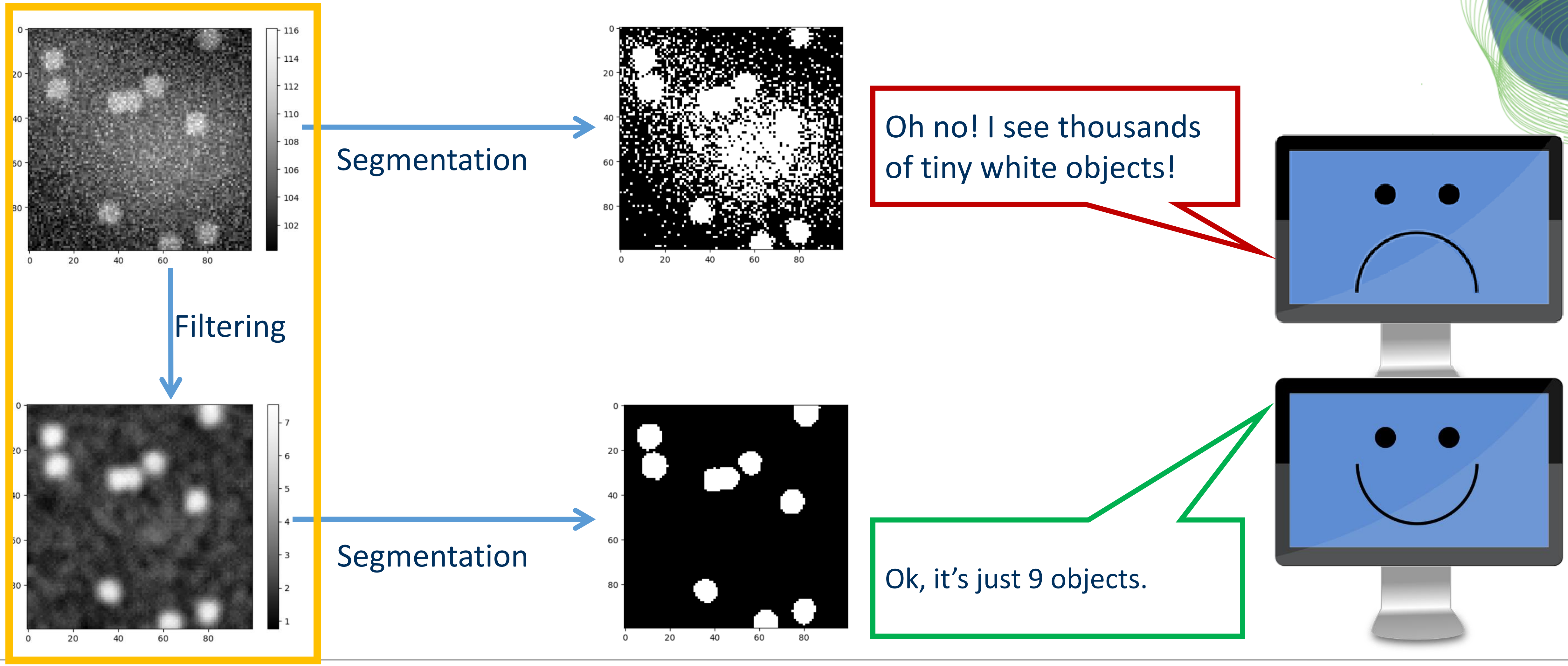
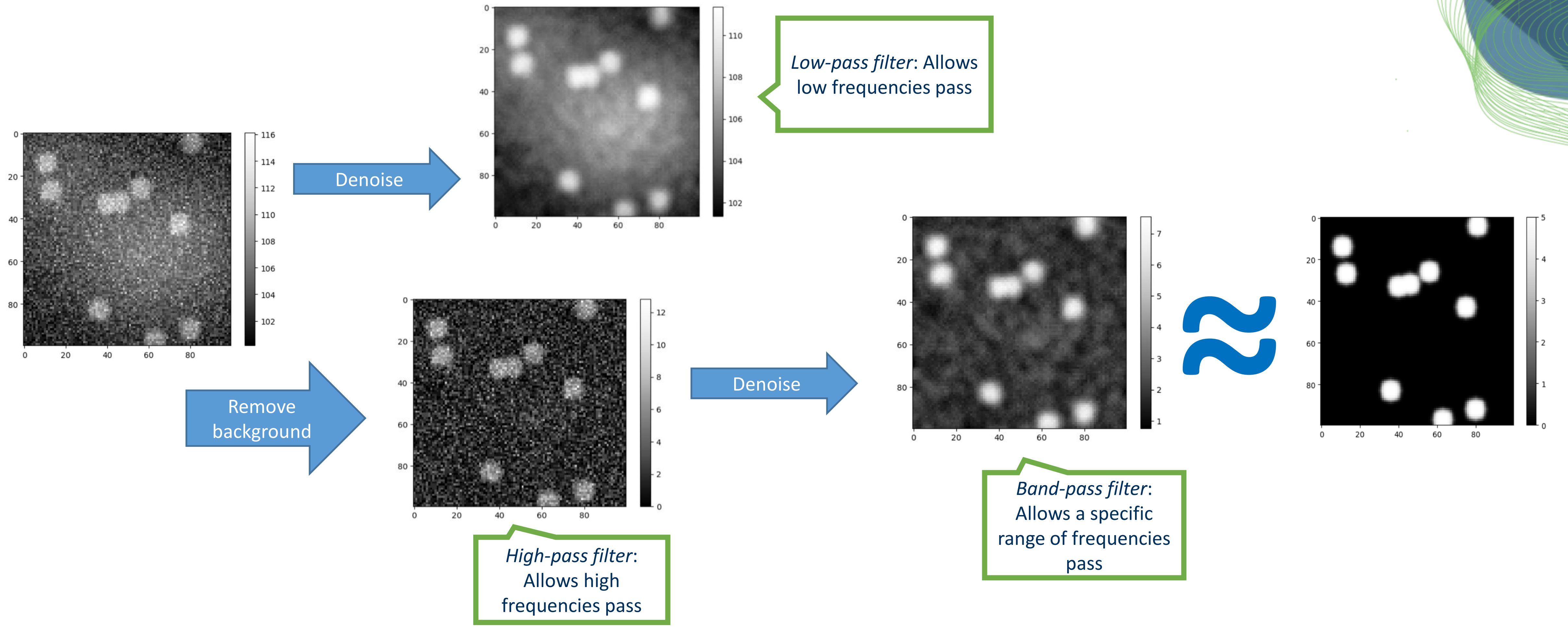


Image filtering

- Attempt to invert / “undo” processes disturbing image quality

Image filtering

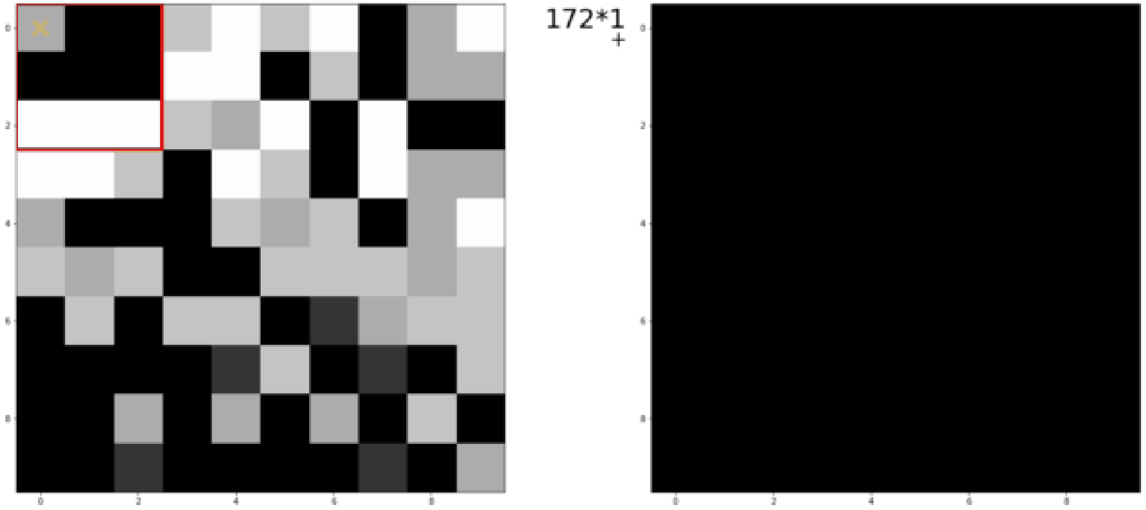
- Attempt to invert / “undo” processes disturbing image quality



https://github.com/BiAPoL/Bio-image_Analysis_with_Python/blob/49a787514a367829c3e0e1832f6cc533e96d549f/03_image_processing/simulated_dataset.ipynb

Linear Filters

- *Linear filters* replace each pixel value with a weighted linear combination of surrounding pixels
- Filter *kernels* are matrices describing a linear filter
- This multiplication of surrounding pixels according to a matrix is called *convolution*



Animation source: Dominic Waithe, Oxford University
https://github.com/dwaithe/generalMacros/tree/master/convolution_ani

Mean filter, 3x3 kernel

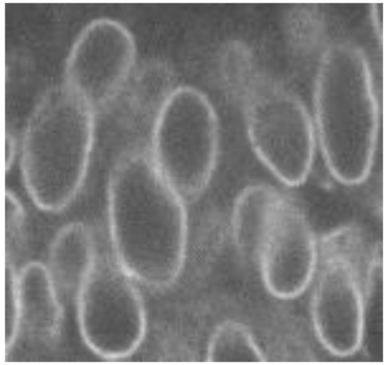
$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$
$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$
$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$

Linear filters

- Terminology:
  - “We convolve an image with a kernel.”
  - Convolution operator: *

Examples

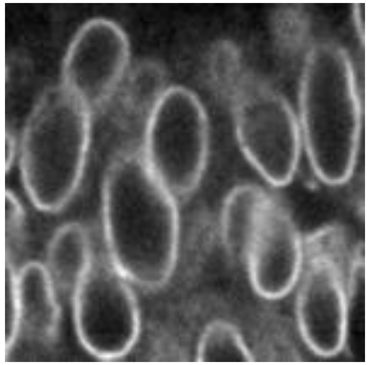
- Mean
- Gaussian blur
- Sobel-operator
- Laplace-filter

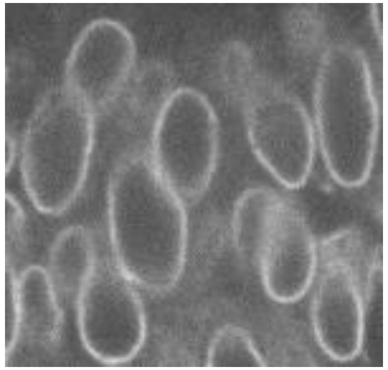


*

1	1	1
1	8	1
1	1	1

=

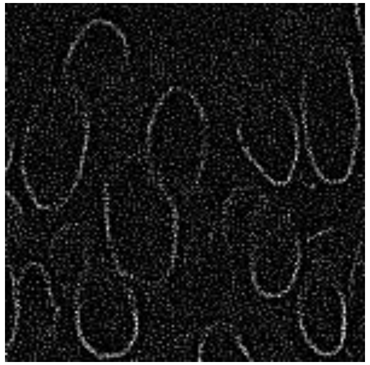




*

0	-1	0
-1	4	-1
0	-1	0

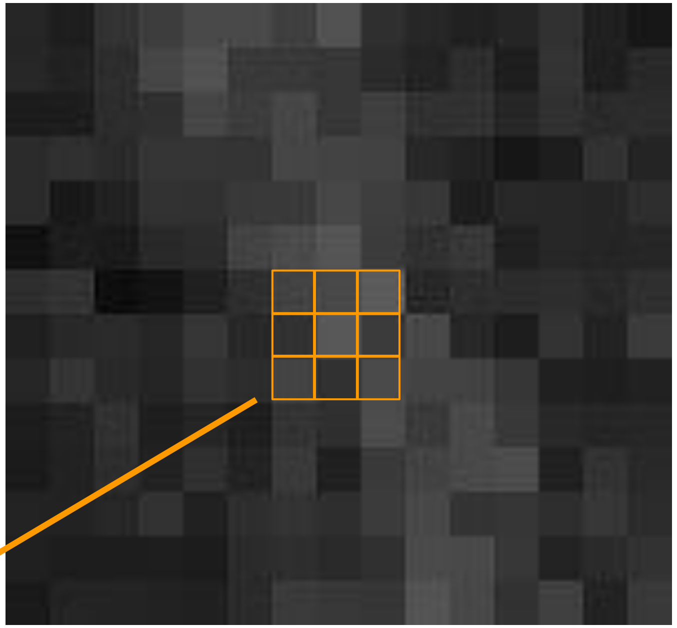
=



Nonlinear Filters

- Non linear filters also replace pixel value inside as rolling window but using a non-linear function.

- Examples: order statistics filters
  - Min
  - Median
  - Max
  - Variance
  - Standard deviation



75	85	60
67	73	91
50	88	59

[50 59 60 67 73 75 85 88 91]

Min Median Max

Noise removal

- Gaussian filter
- Median filter (computationally expensive)

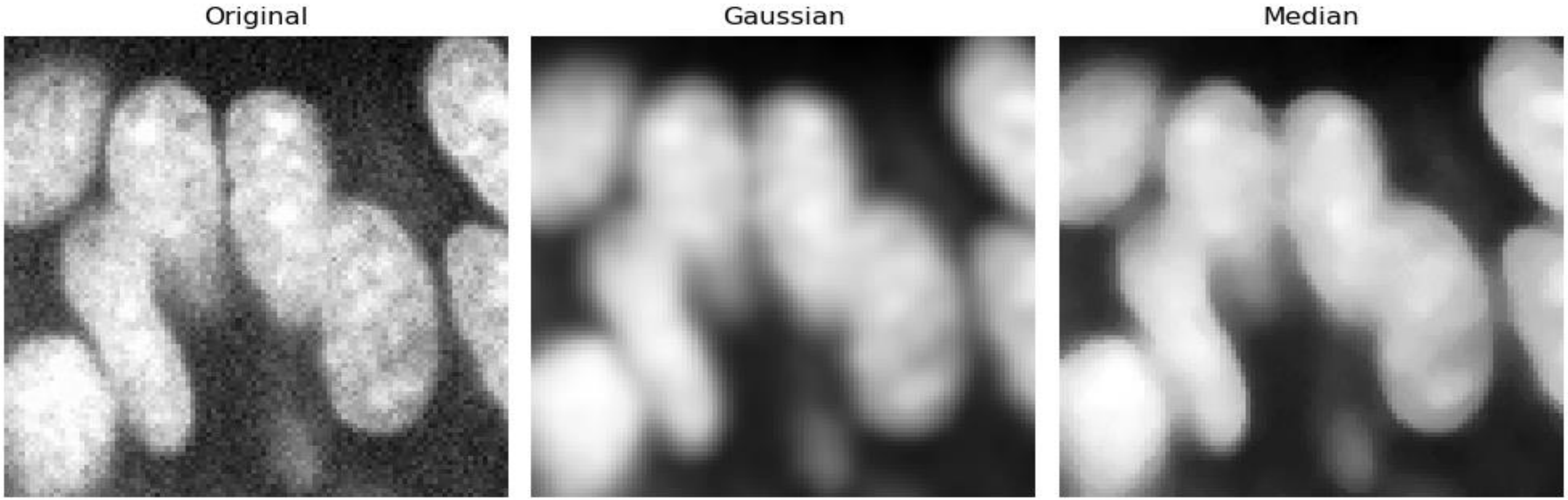
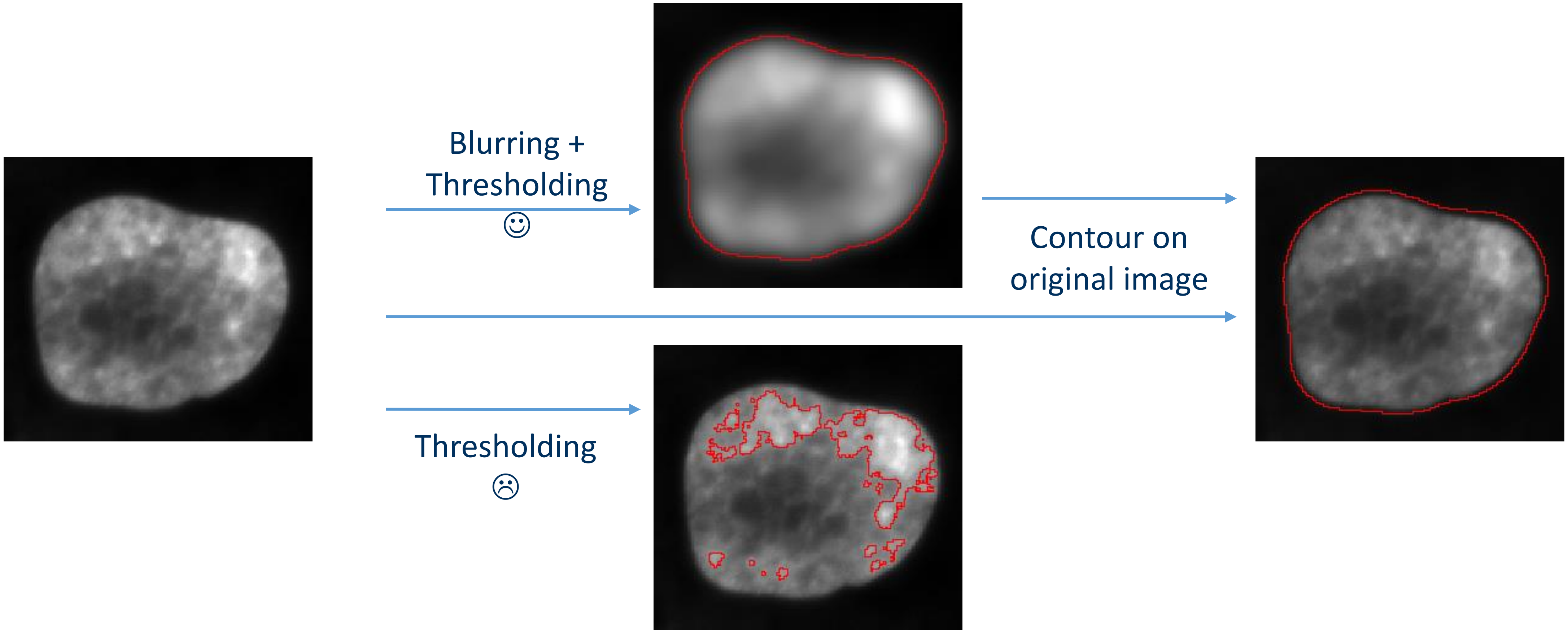


Image source: Mauricio Rocha Martins (Norden/Myers lab, MPI CBG)

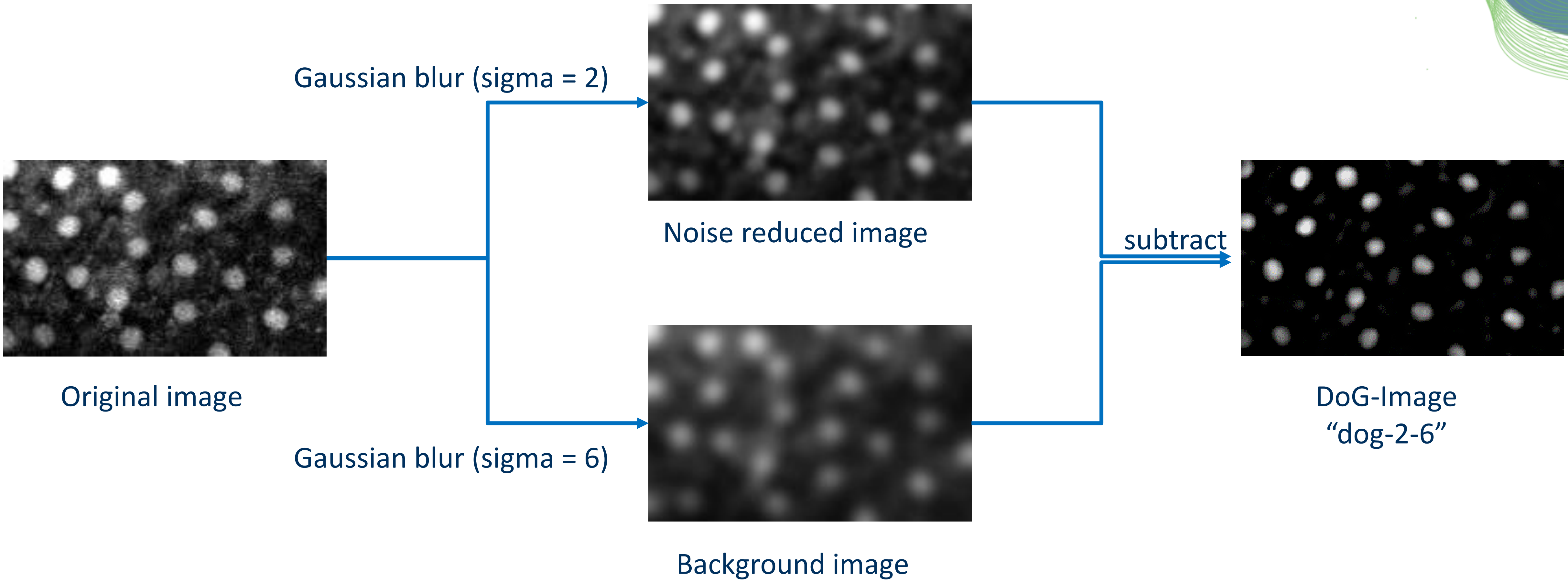
Filtering for improving thresholding results

- In case thresholding algorithms outline the wrong structure, blurring in advance may help.
- However: **Do not** continue processing the blurred image, continue with the original!



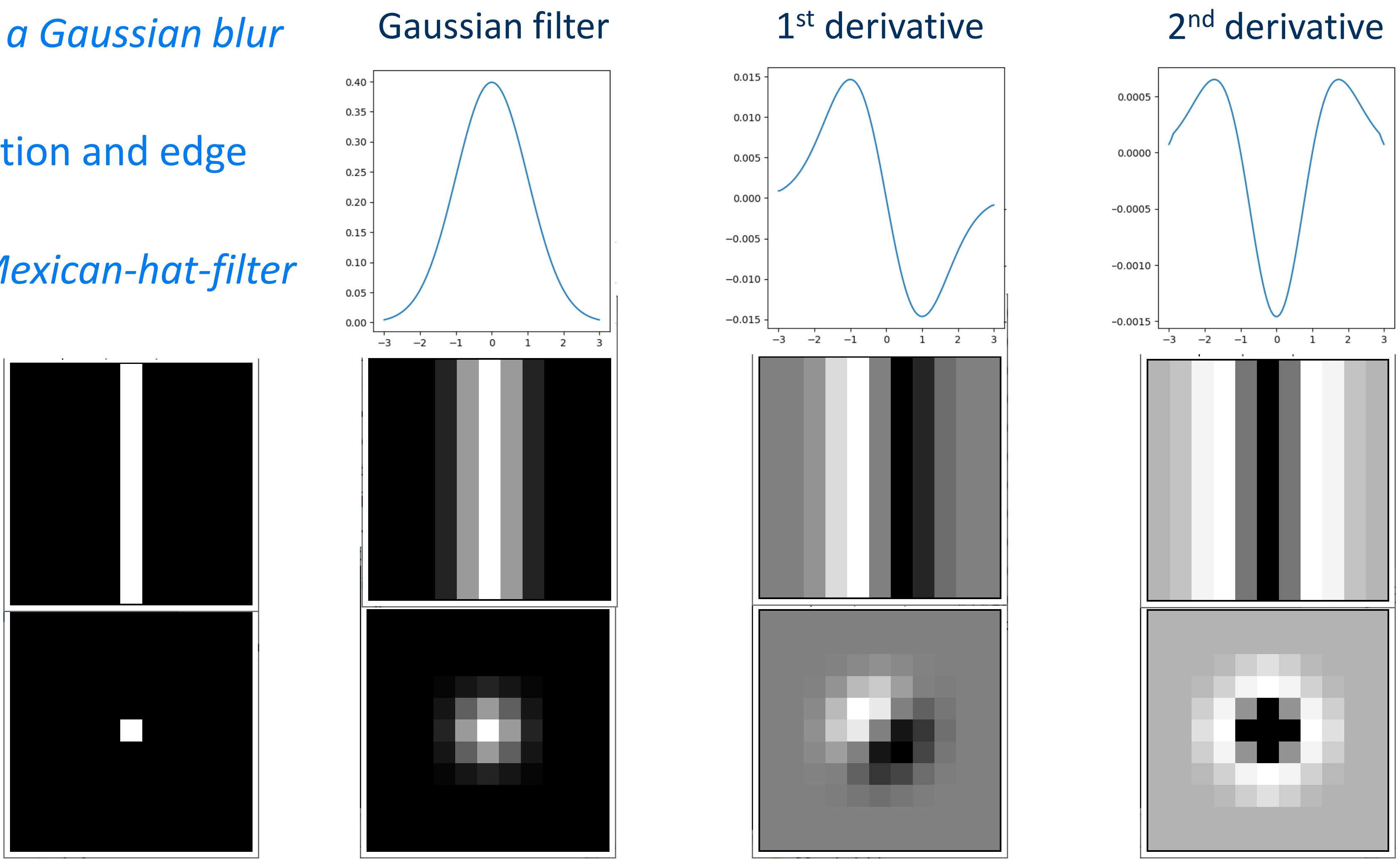
Difference-of-Gaussian (DoG)

- Improve image in order to detect bright objects.
- Band-pass filter



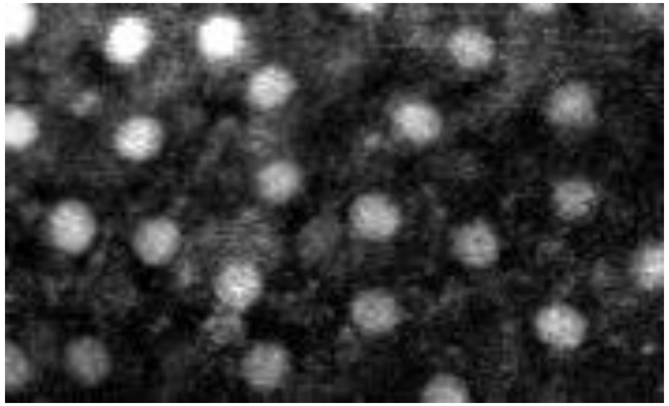
Laplace-filter

- *Second derivative of a Gaussian blur filter*
- Used for edge-detection and edge enhancement
- Also known as the *Mexican-hat-filter*



Laplacian-of-Gaussian (LoG)

Laplace filter



*

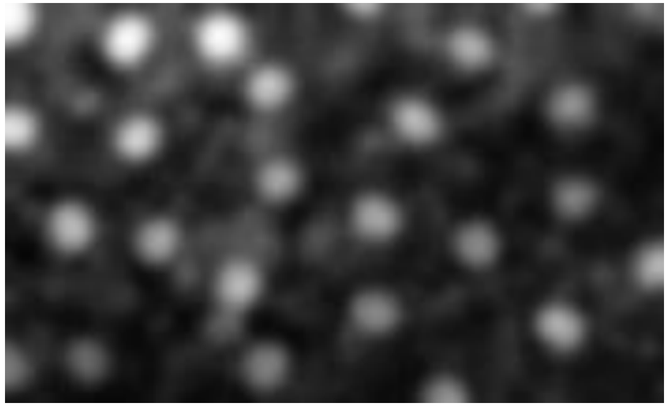
0	-1	0
-1	4	-1
0	-1	0

=

Laplace filtered image

Gaussian filter

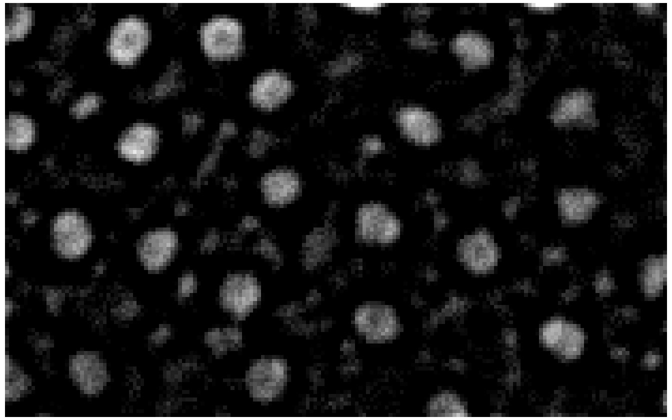
Laplacian of Gaussian filter



*

0	-1	0
-1	4	-1
0	-1	0

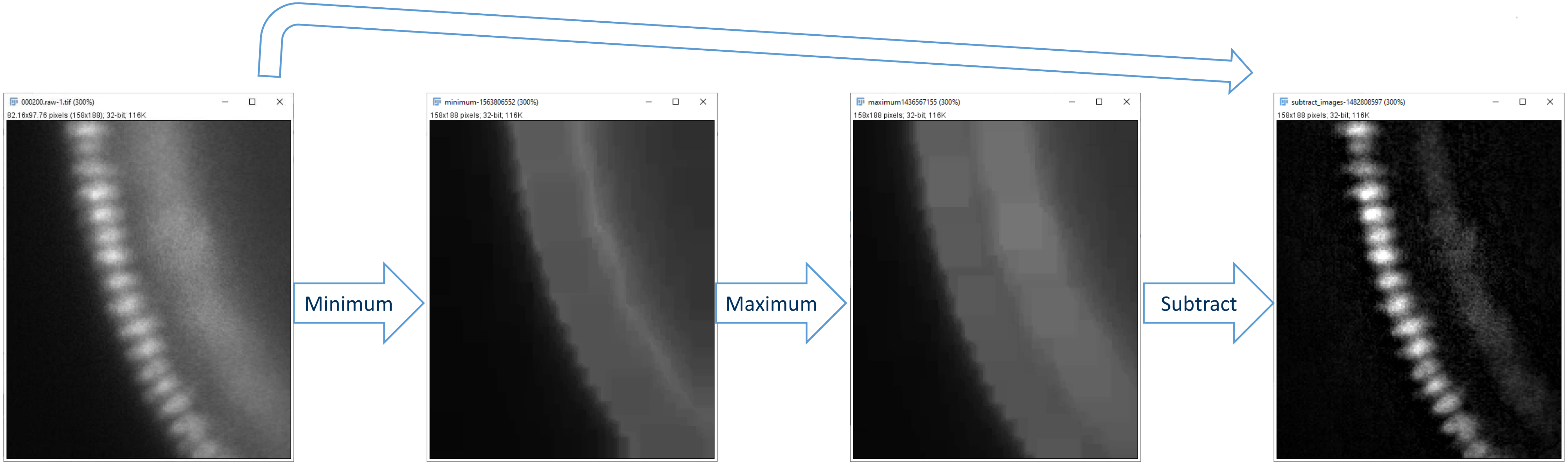
=



LoG image

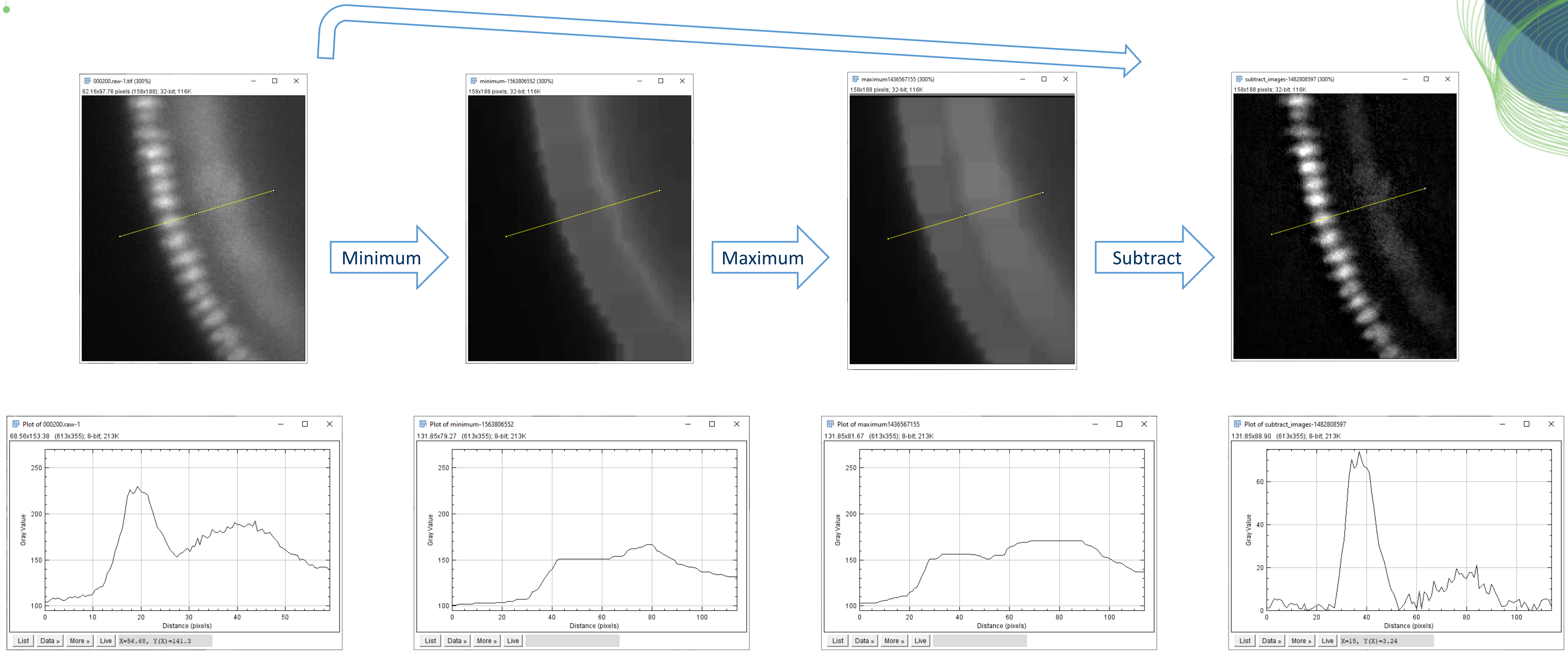
Top-hat filter

- Background subtraction



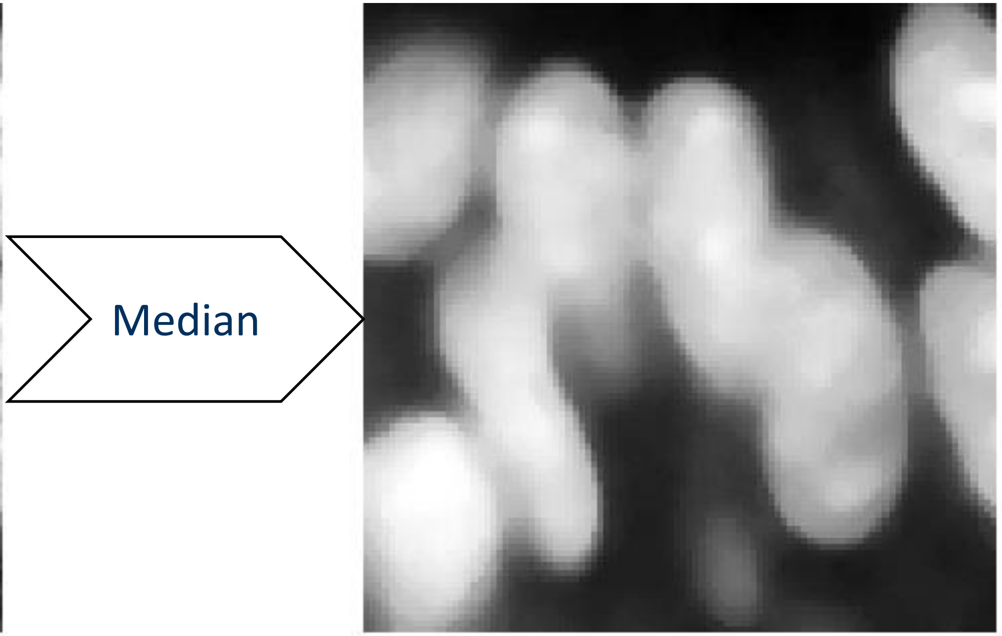
Top-hat filter

- Background subtraction



Quiz: Noise removal

- The median filter is a …

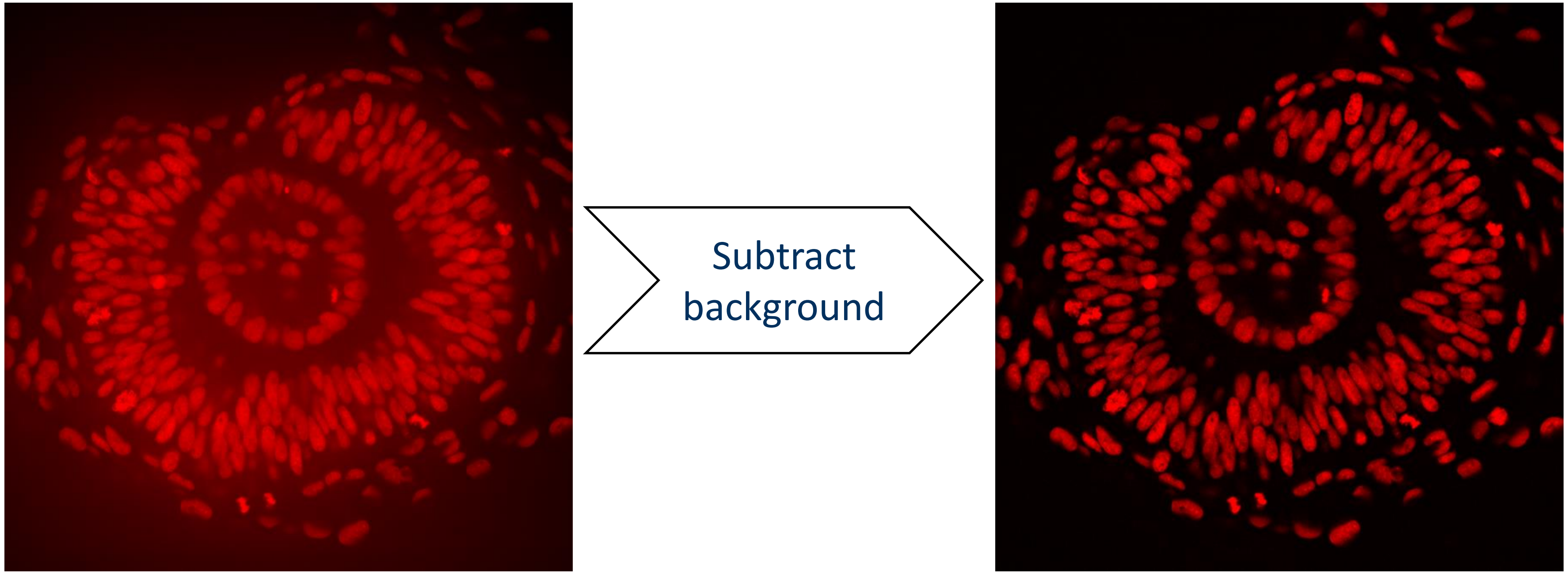


Linear filter

Non-linear filter

Background removal

- Removing background from an image is a … ?



Low-pass filter

High-pass filter

Image source: Mauricio Rocha Martins (Norden/Myers lab, MPI CBG)

Image Processing: Morphological Operations

With material from

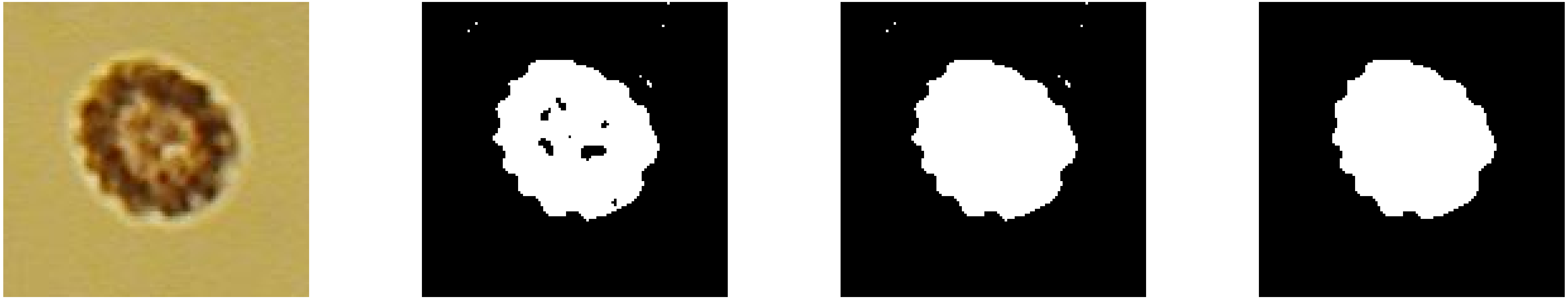
Robert Haase,

Marcelo Leomil Zoccoler, Physic of Life, TU Dresden

Refining masks

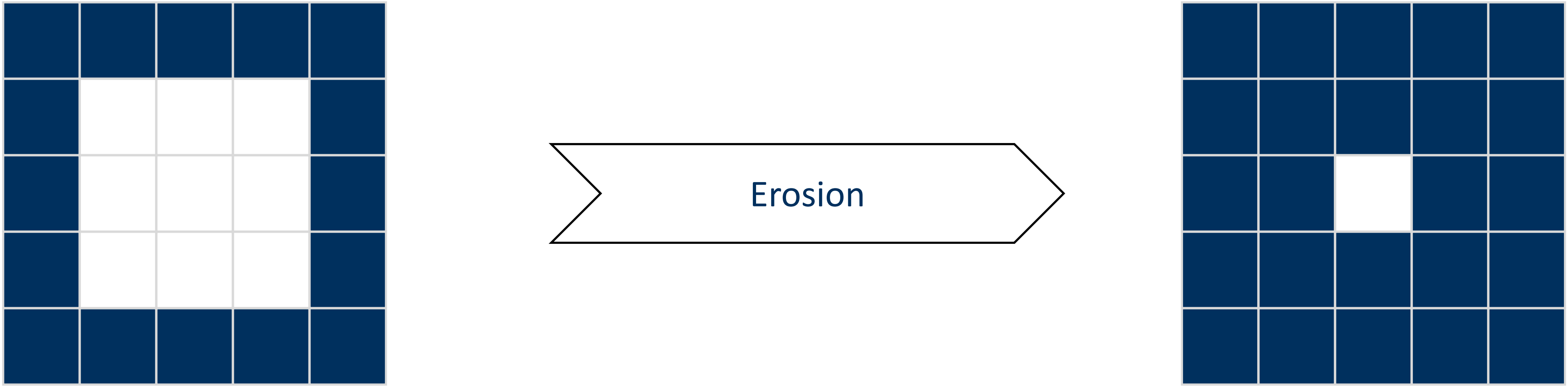
- Binary mask images may not be perfect immediately after thresholding.
- There are ways of refining them

Thresholding → Closing → Opening



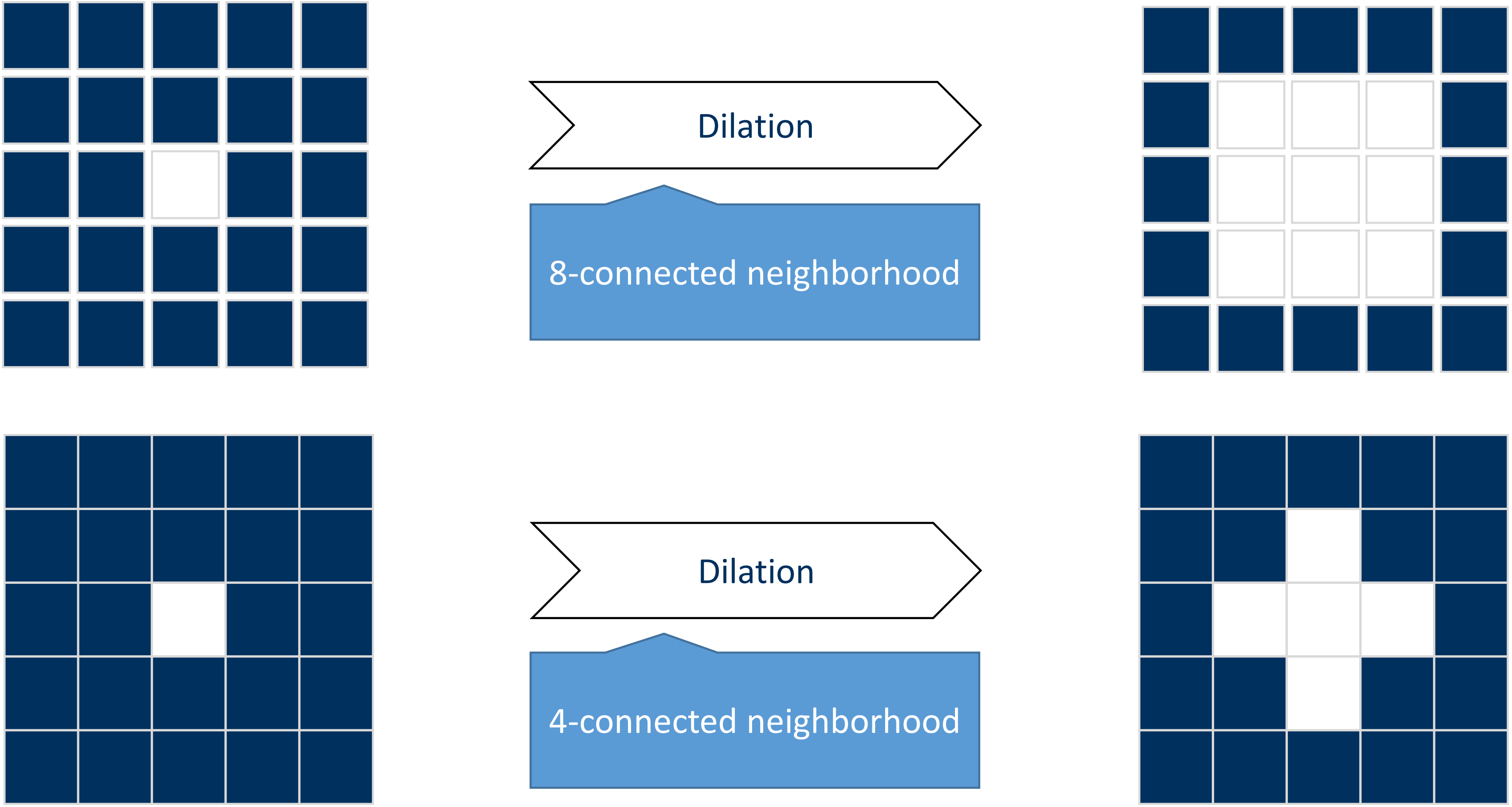
Erosion

- Erosion: Every pixel with at least one black neighbor becomes black.



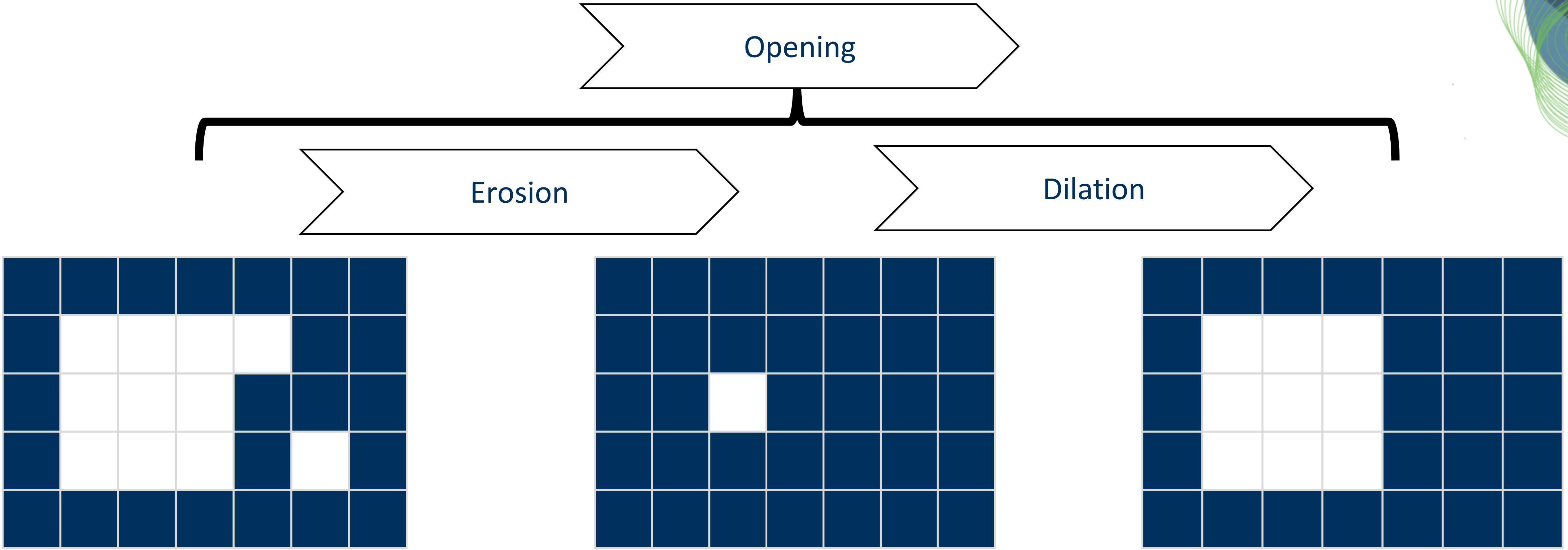
Dilation

- Dilation: Every pixel with at least one white neighbor becomes white.



Opening

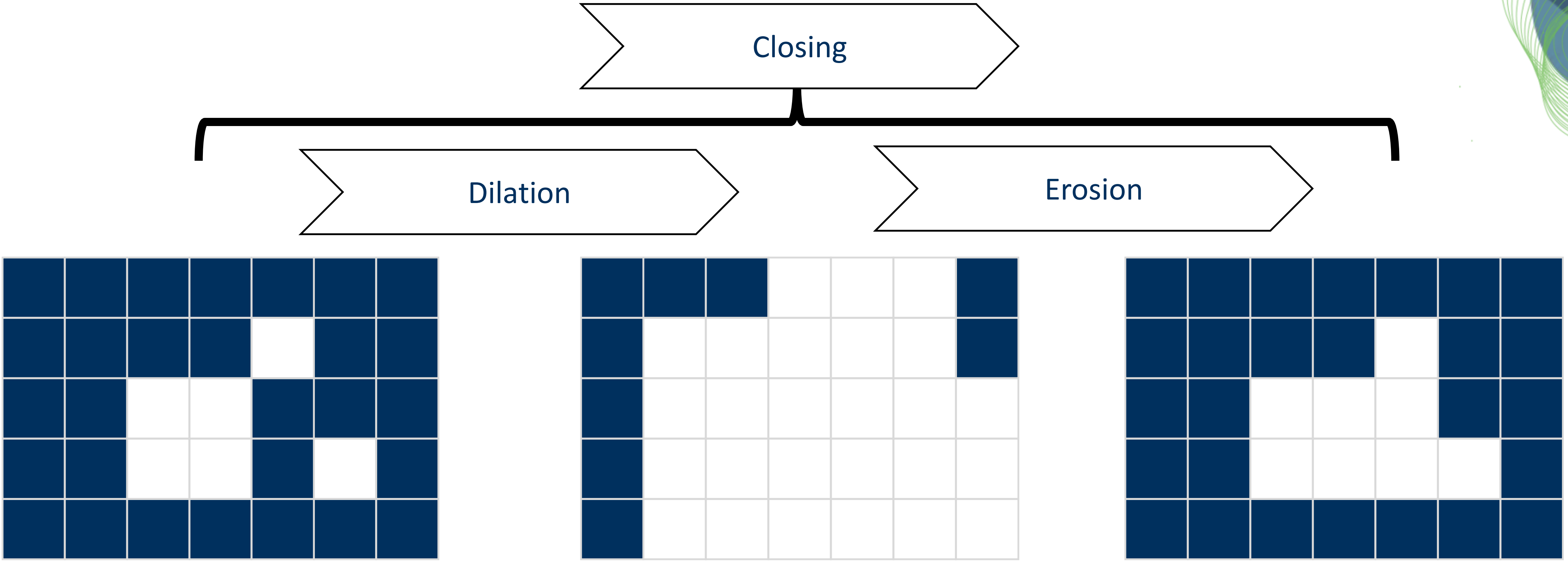
- Erosion and dilation combined allow correcting outlines.



- It can separate white (high intensity) structures that are weakly connected
- It may erase small white structures

Closing

- Erosion and dilation combined allow correcting outlines.



- It can connect white (high intensity) structures that are nearby
- It may close small holes inside structures

Image Processing in Python

With material from

Robert Haase, Marcelo Leomil Zoccoler, Physics of Life, TU Dresden

Working with images in python

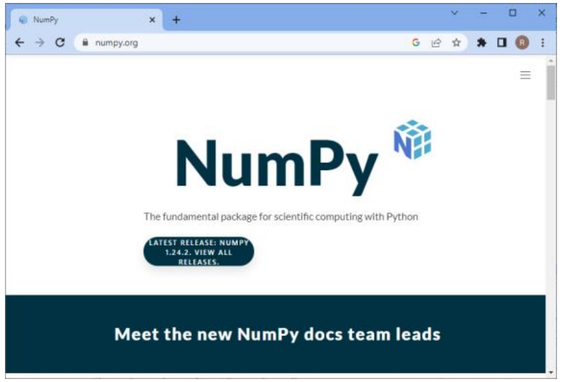
- ## Open images

```
from skimage.io import imread

image = imread("blobs.tif")
```

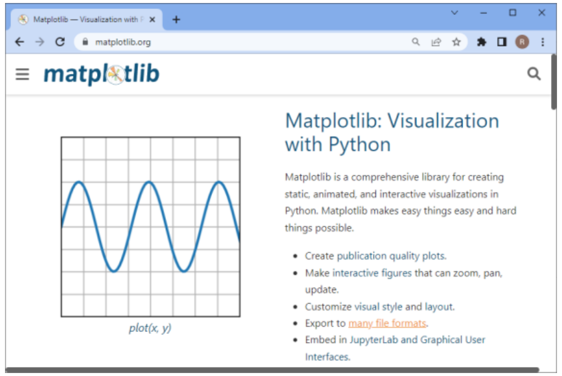
```
image
```

```
array([[ 40,  32,  24, ..., 216, 200, 200],
       [ 56,  40,  24, ..., 232, 216, 216],
       [ 64,  48,  24, ..., 240, 232, 232],
       ...,
       [ 72,  80,  80, ...,  48,  48,  48],
       [ 80,  80,  80, ...,  48,  48,  48],
       [ 96,  88,  80, ...,  48,  48,  48]], dtype=uint8)
```

Images are *just* multi-dimensional arrays or "arrays of arrays".



https://numpy.org/



https://matplotlib.org/

Working with images in python

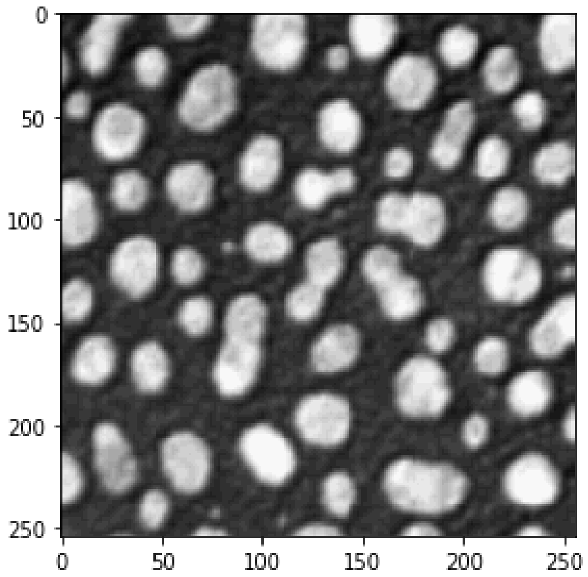
Open images

```python
from skimage.io import imread

image = imread("blobs.tif")
```

Visualize images

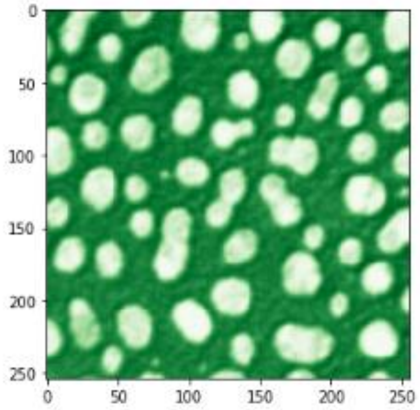
```python
from skimage.io import imshow

imshow(image)
```

```
<matplotlib.image.AxesImage at 0x245e7
```



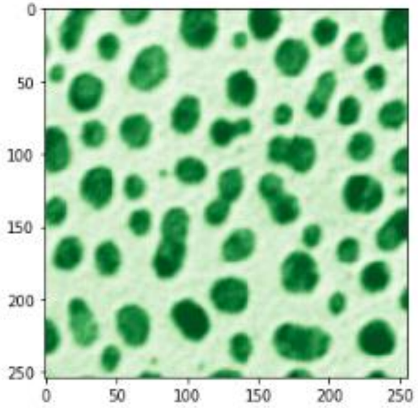
```python
imshow(image, cmap="Greens_r")
```

```
<matplotlib.image.AxesImage at 0:
```



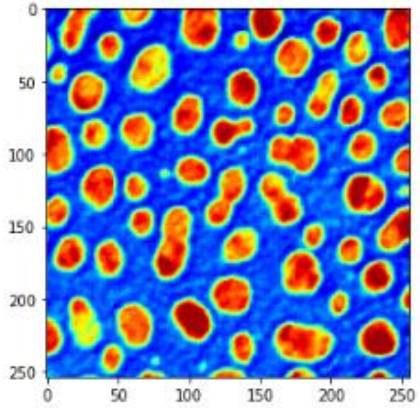
```python
imshow(image, cmap="Greens")
```

```
<matplotlib.image.AxesImage at (
```



```python
imshow(image, cmap="jet")
```

```
<matplotlib.image.AxesImage at
```



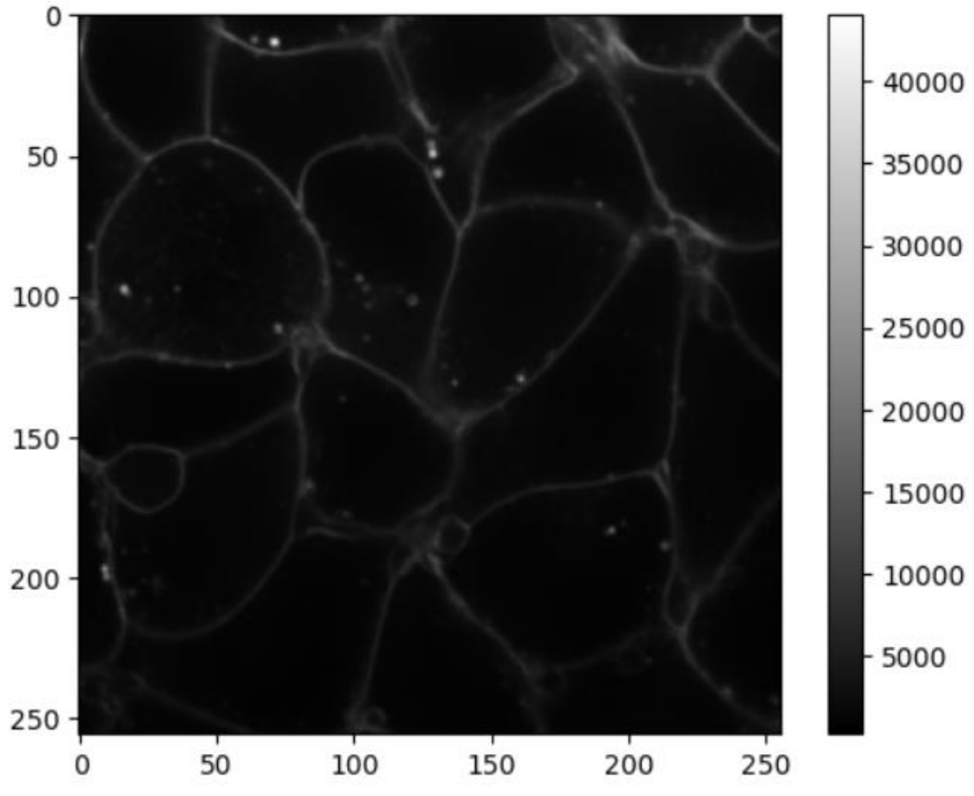
This does not modify the image data. The images are just shown with different colors representing the same values.

Brightness, contrast, display-range

- After loading data, make sure you can see the structure you're interested in

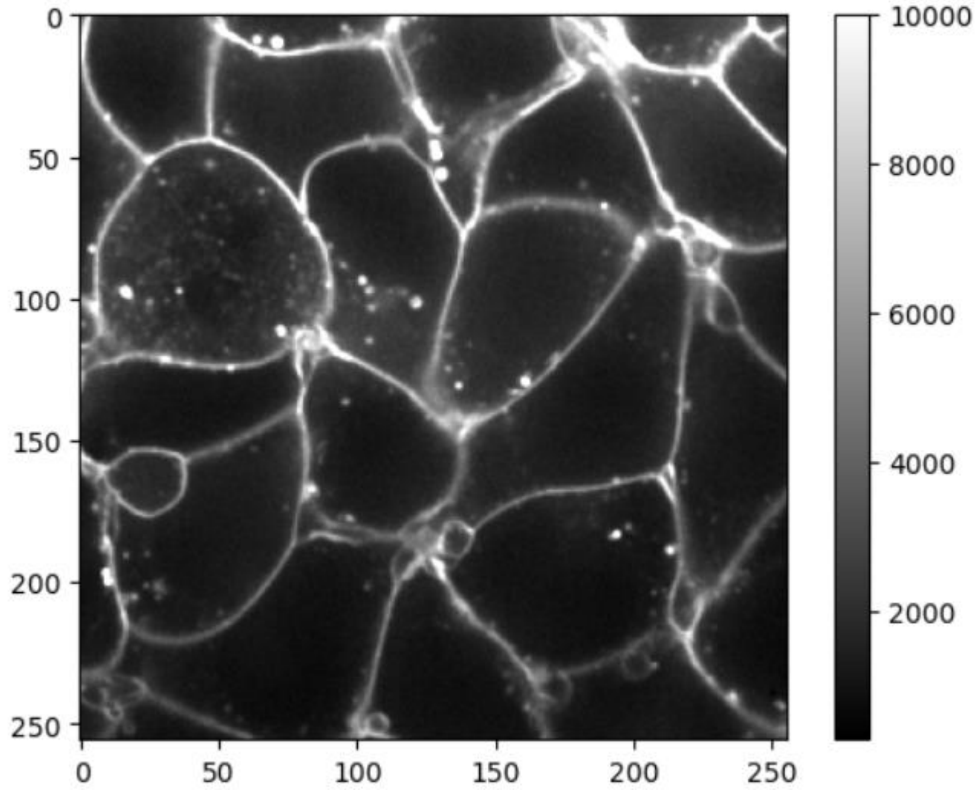
```
plt.imshow(image, cmap='gray')
plt.colorbar()
```

<matplotlib.colorbar.Colorbar at 0x14f22cf71f0>




```
plt.imshow(image, cmap='gray', vmax=10000)
plt.colorbar()
```

<matplotlib.colorbar.Colorbar at 0x14f22d70310>

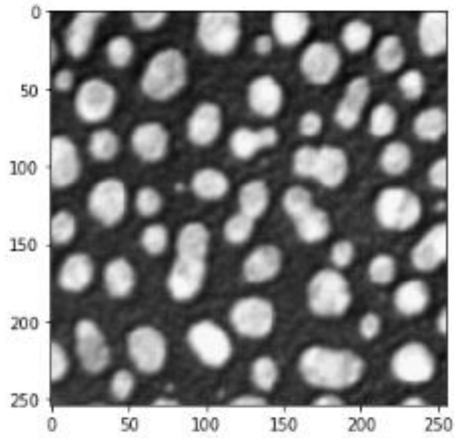


Cropping and resampling images

- Indexing and cropping *numpy-arrays* works like with python arrays.

```
imshow(image)
```

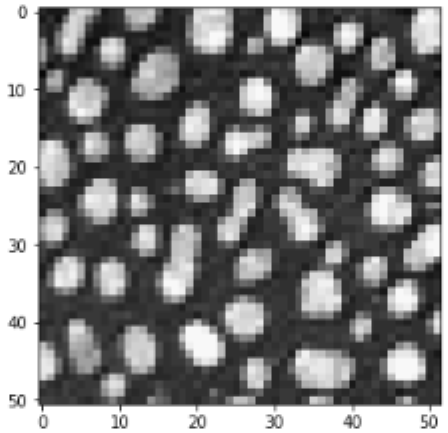
<matplotlib.image.AxesImage at 6



Original image

```
sampled_image = image[::5, ::5]

imshow(sampled_image)
```

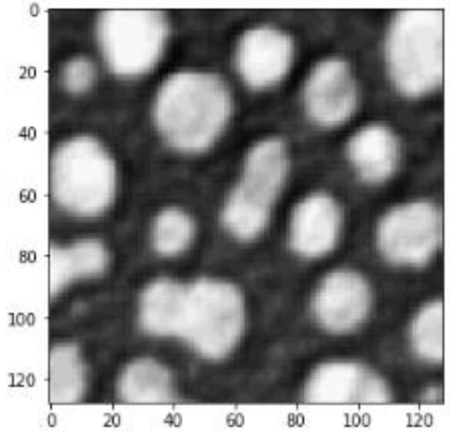
<matplotlib.image.AxesImage at 0x



Sub-sampled image

```
cropped_image2 = image[0:128, 128:]

imshow(cropped_image2)
```

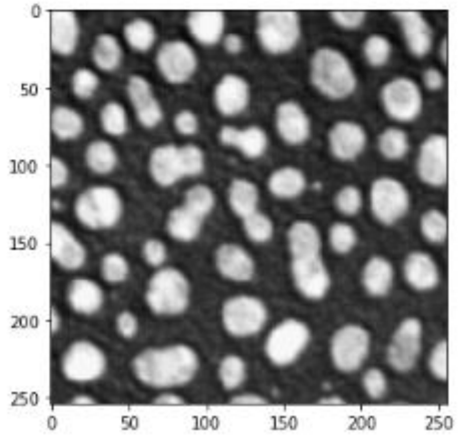
<matplotlib.image.AxesImage at 0x29e



Cropped image

```
flipped_image = image[::, ::-1]

imshow(flipped_image)
```

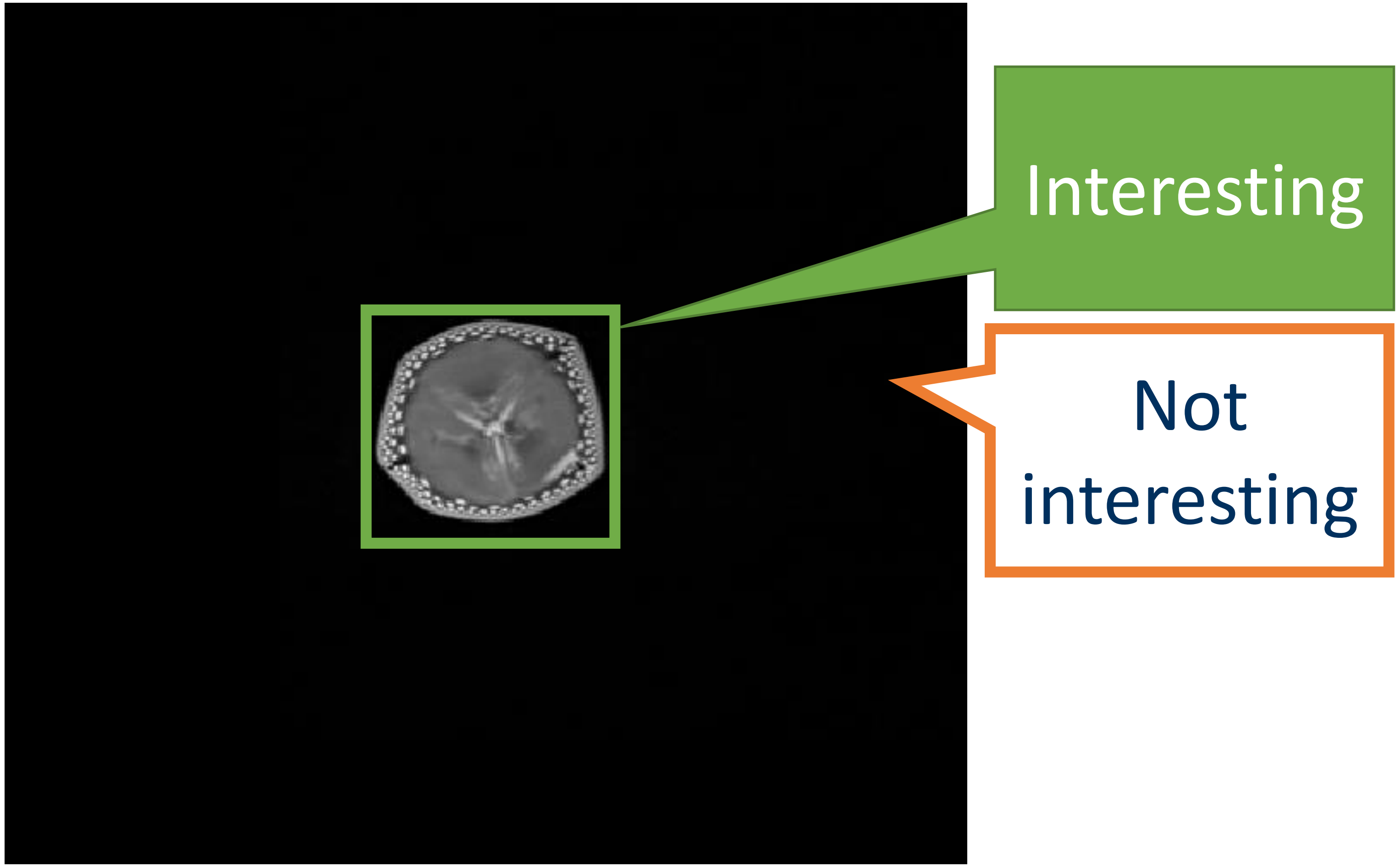
<matplotlib.image.AxesImage at 0x



Flipped image

Cropping and resampling images

- Crop out the region you're interested in



In this case
you can spare
8/9 compute time for
following processing steps

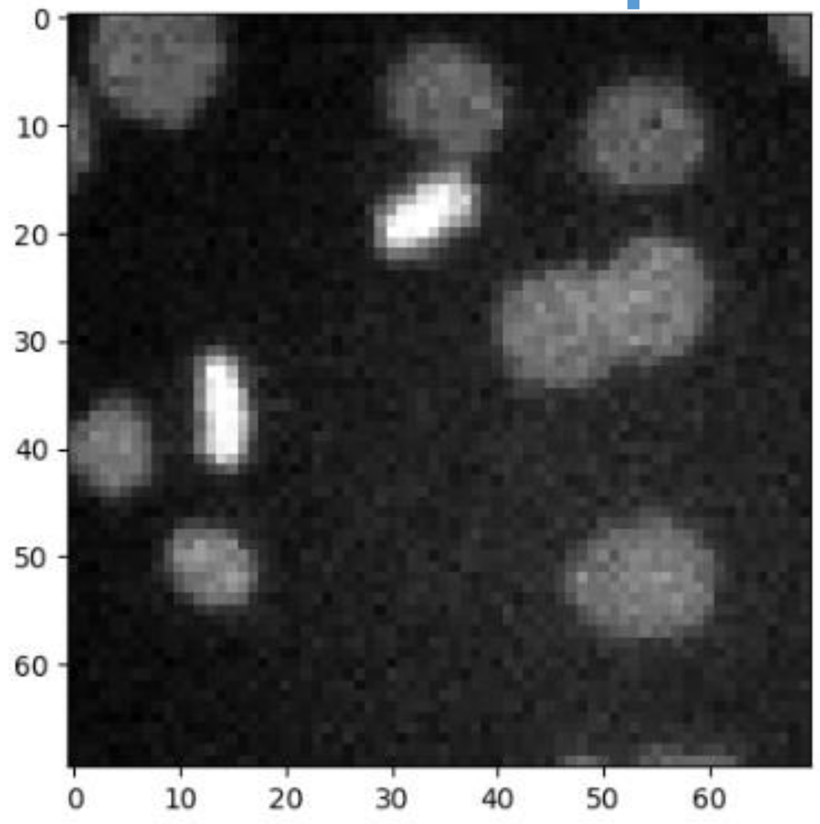
Image data source: Nasreddin Abolmaali, TU Dresden

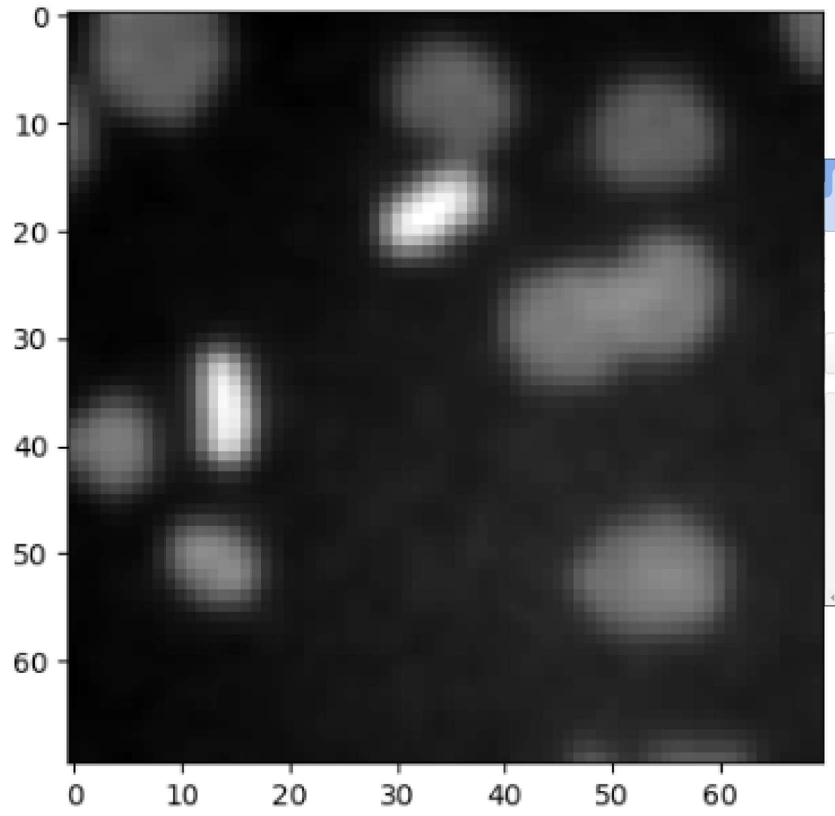
Filters

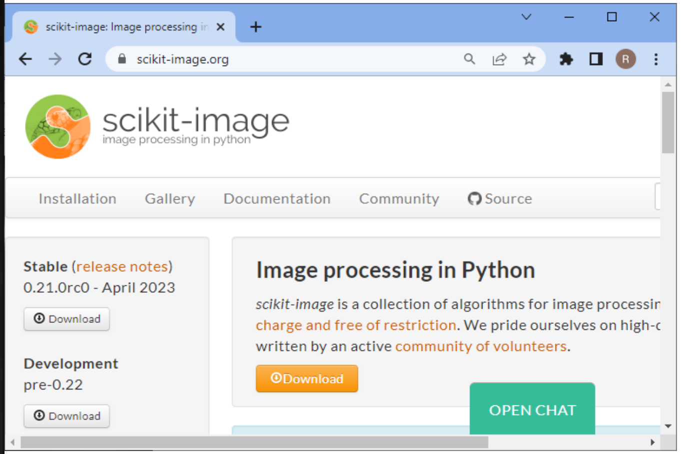
... are just functions

```
denoised_gaussian = filters.gaussian(image3, sigma=1)

plt.imshow(denoised_gaussian, cmap='gray')
```

<matplotlib.image.AxesImage at 0x283aab3ba90>







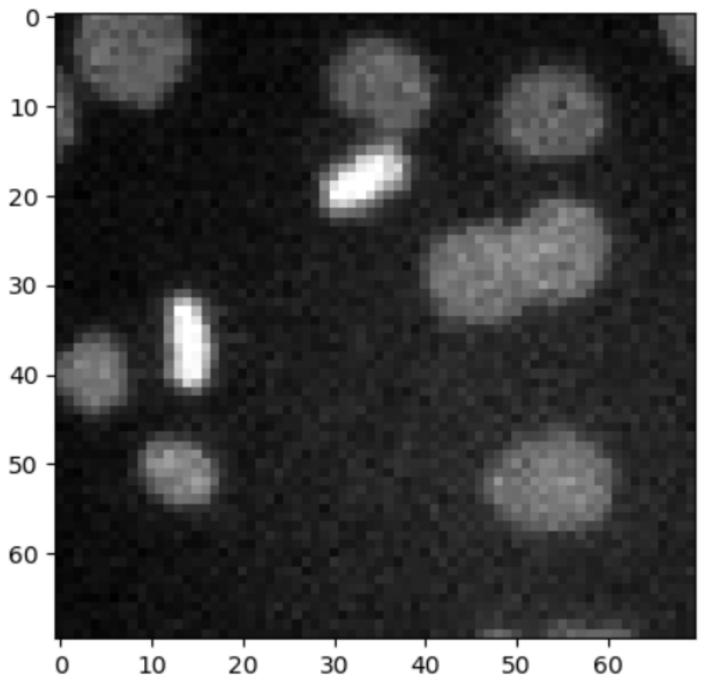
https://scikit-image.org/

Filters

- Use every opportunity and play with filter parameters to get an idea what they do.

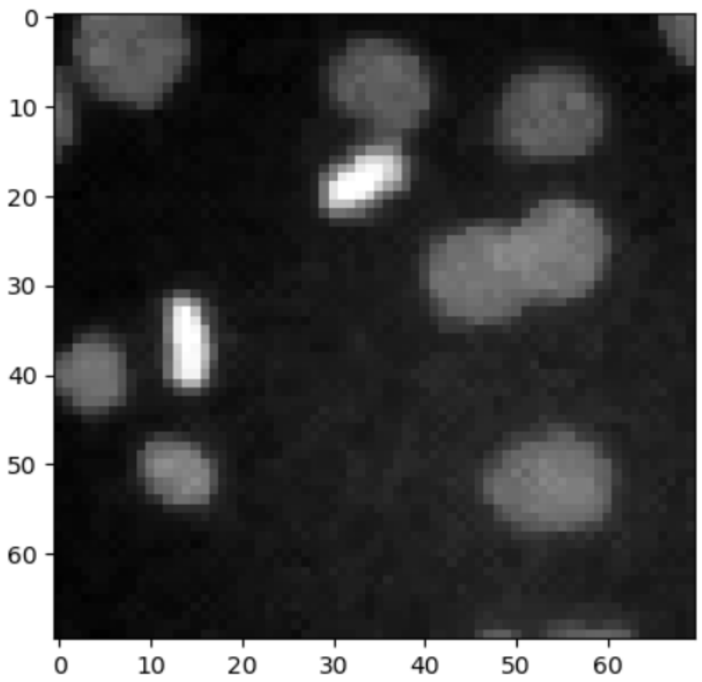
```
plt.imshow(image3, cmap='gray')
```

<matplotlib.image.AxesImage at 0x1d86893b6d0>



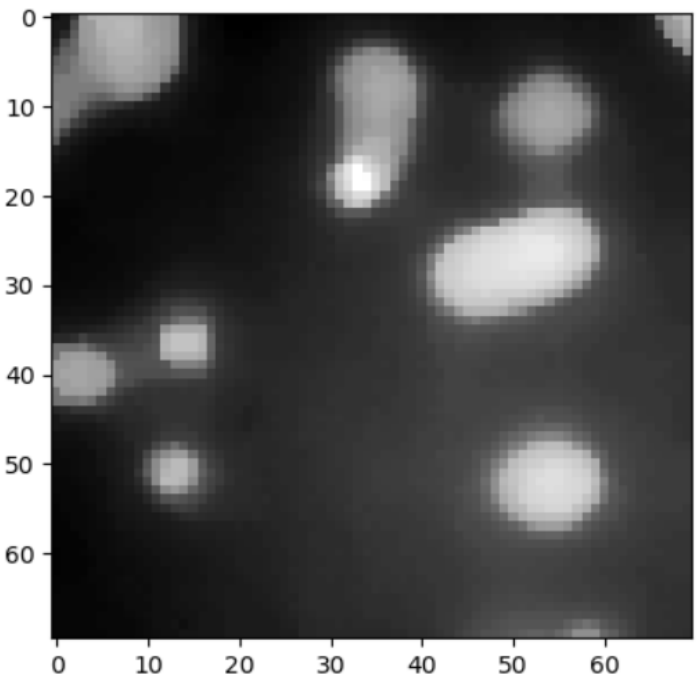
```
denoised_median = filters.median(image3, morphology.disk(1))

plt.imshow(denoised_median, cmap='gray')
```

<matplotlib.image.AxesImage at 0x1d868a189d0>



```
denoised_median2 = filters.median(image3, morphology.disk(5))

plt.imshow(denoised_median2, cmap='gray')
```

<matplotlib.image.AxesImage at 0x1d868ca7af0>

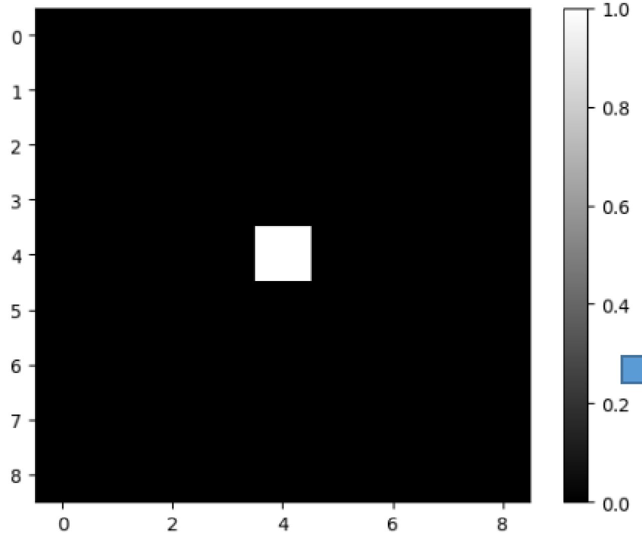


Filters

... may be custom functions

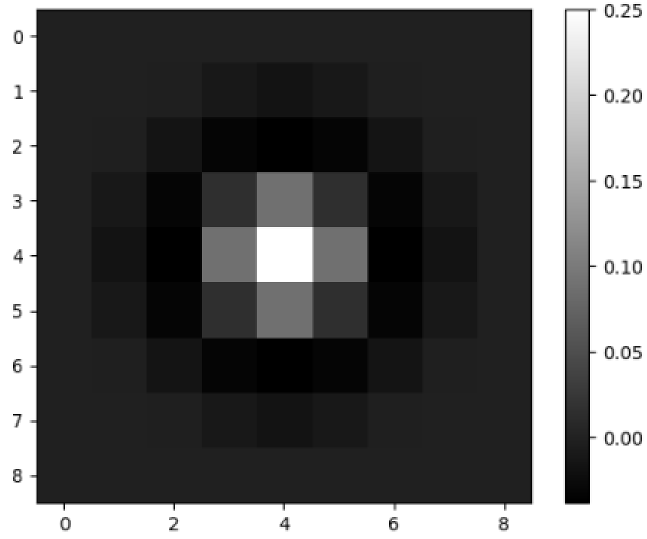
Recommendation: Apply custom filters to super simple images to see if they do the right thing.

```python
def laplacian_of_gaussian(image, sigma):
    """
    Applies a Gaussian kernel to an image and the Laplacian afterwards.
    """

    # blur the image using a Gaussian kernel
    intermediate_result = filters.gaussian(image, sigma)

    # apply the mexican hat filter (Laplacian)
    result = filters.laplace(intermediate_result)

    return result
```




```python
log_image1 = laplacian_of_gaussian(image2, sigma=1)

plt.imshow(log_image1, cmap='gray')
plt.colorbar()
```

```
<matplotlib.colorbar.Colorbar at 0x283a9679430>
```



Binarization / Thresholding

- Turn images into binary images
  (very basic form of segmentation)
- When using scikit-image, `threshold_` functions typically return a threshold you need to apply yourself.

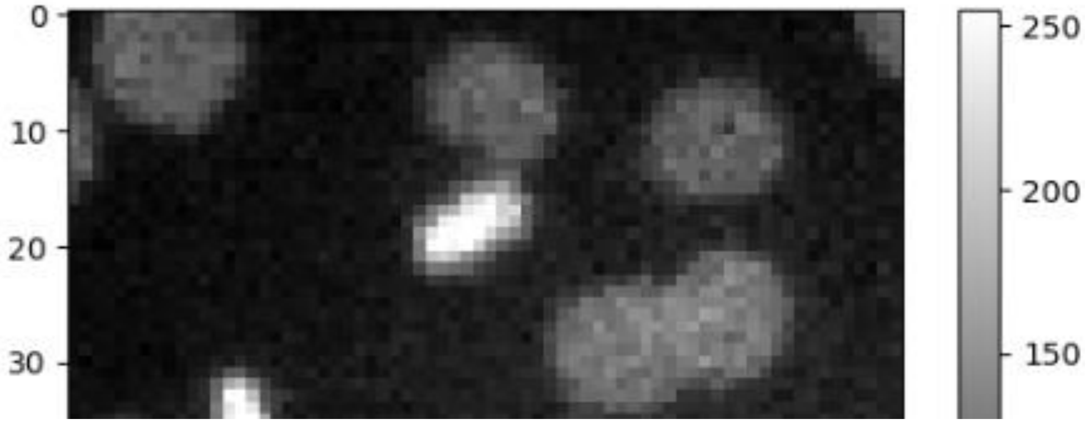
```python
from skimage.filters import threshold_otsu
```

```python
threshold = threshold_otsu(image_nuclei)
threshold
```

```
77
```

```python
image_otsu_binary = image_nuclei > threshold

plt.imshow(image_otsu_binary, cmap='gray')
plt.colorbar()
```

```
<matplotlib.colorbar.Colorbar at 0x1c285b4f550>
```



Morphological operations

- To *morph* objects in binary images

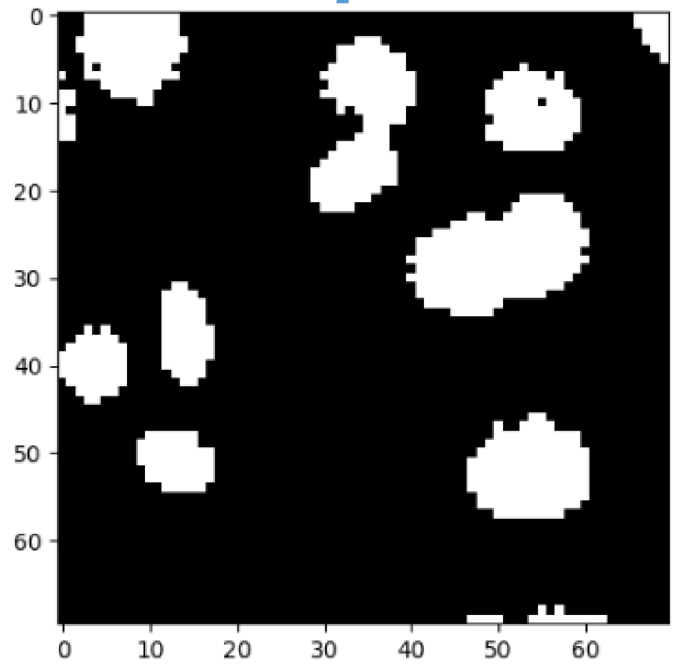
```
from skimage import morphology
```

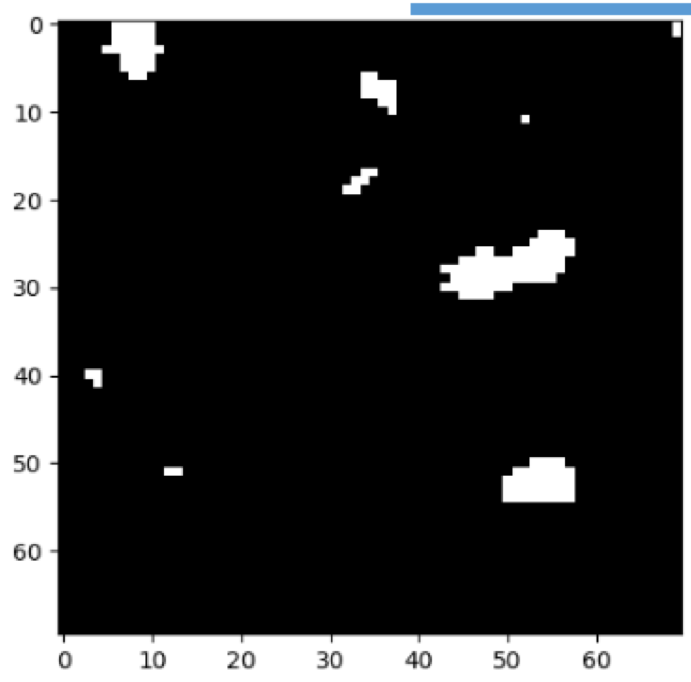
```
eroded = morphology.binary_erosion(image_binary, disk)
plt.imshow(eroded, cmap='gray')
```

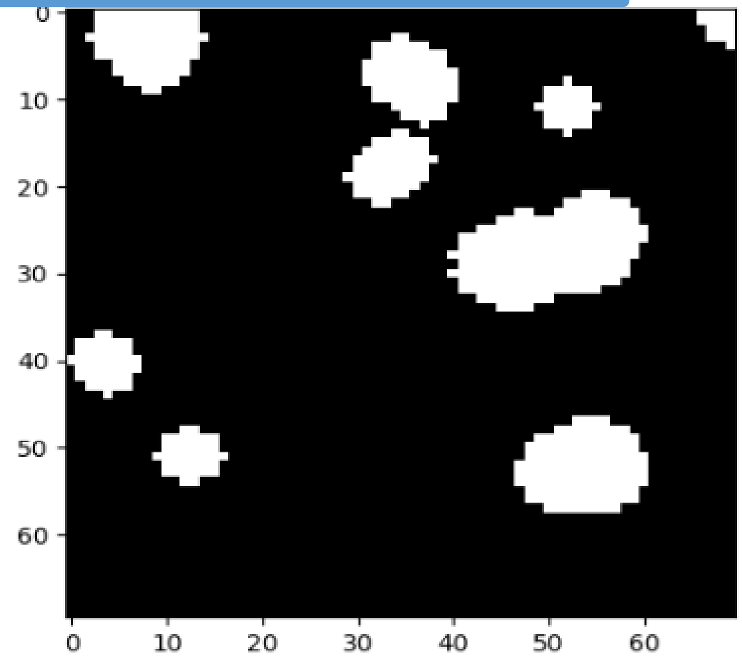
<matplotlib.image.AxesImage at 0x15288661520>

```
eroded_dilated = morphology.binary_dilation(eroded, disk)
plt.imshow(eroded_dilated, cmap='gray')
```

<matplotlib.image.AxesImage at 0x1528893ffd0>







Summary

- Image basics
  - Pixel size, colormaps
  - Image histogram
- Image Filtering
- Morphological Operations

- Python libraries
  - Matplotlib
  - Scikit-image

Coming up next

- Image Segmentation
  - Connected component analysis
  - Voronoi-Otsu-Labeling
- Surface reconstruction