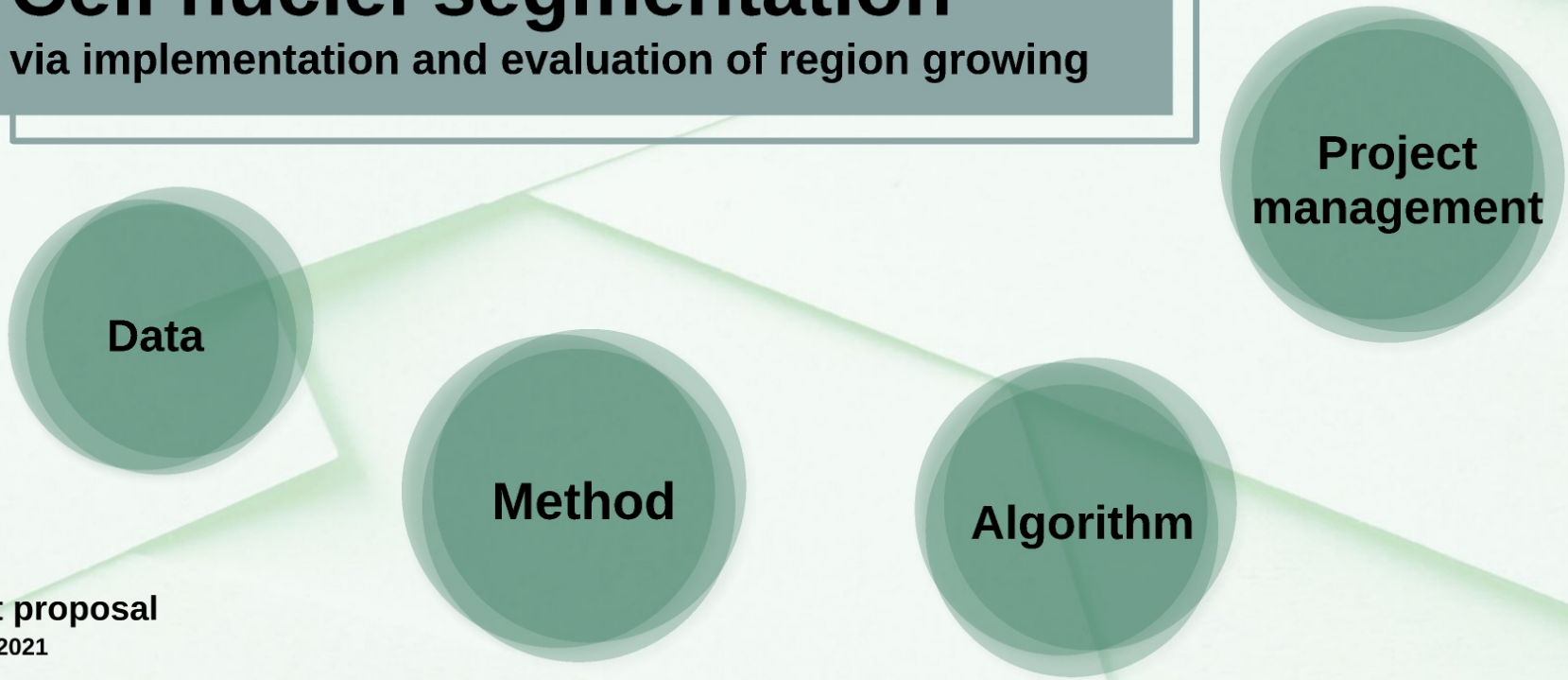
Cell nuclei segmentation

via implementation and evaluation of region growing



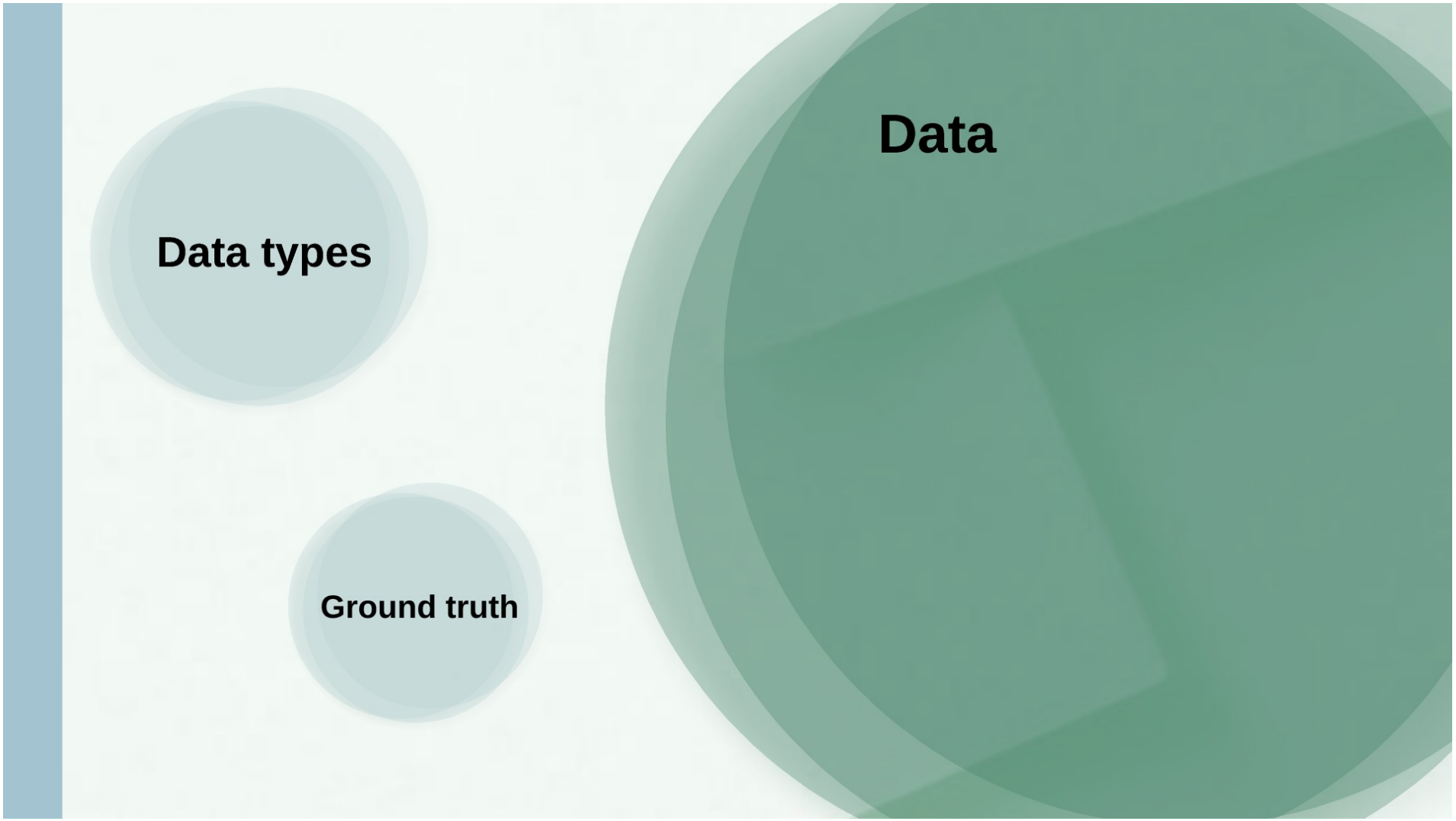
Project proposal

12th May 2021

Data Analysis MoBi SS2021

Supervisor: Dr. Karl Rohr, Christian Ritter; Tutor: Nicholas Peschke

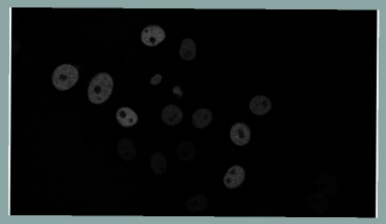
Topic 04: Biomedical image analysis
Group 04: Marie Becker, Ina Jung, Laura Kaschnitz, Johanna Möller


Data
Data types
Ground truth

Data types

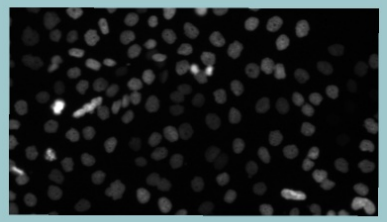
N2DH-GOWT1

- Mouse (*mus musculus*)
- Embryonic Stem Cells
- Established from the epitheloid cervix carcinoma
- Fluorescent chromosomes, core histone 2B tagged with green GFP



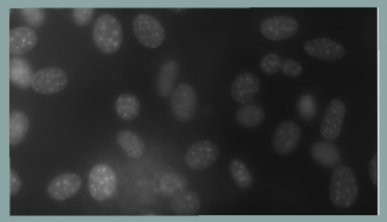
N2DL-HeLa

- Human (*Homo sapiens*)
- Cervix carcinoma
- Established from the epitheloid cervix carcinoma
- Fluorescent chromosomes, core histone 2B tagged with green GFP

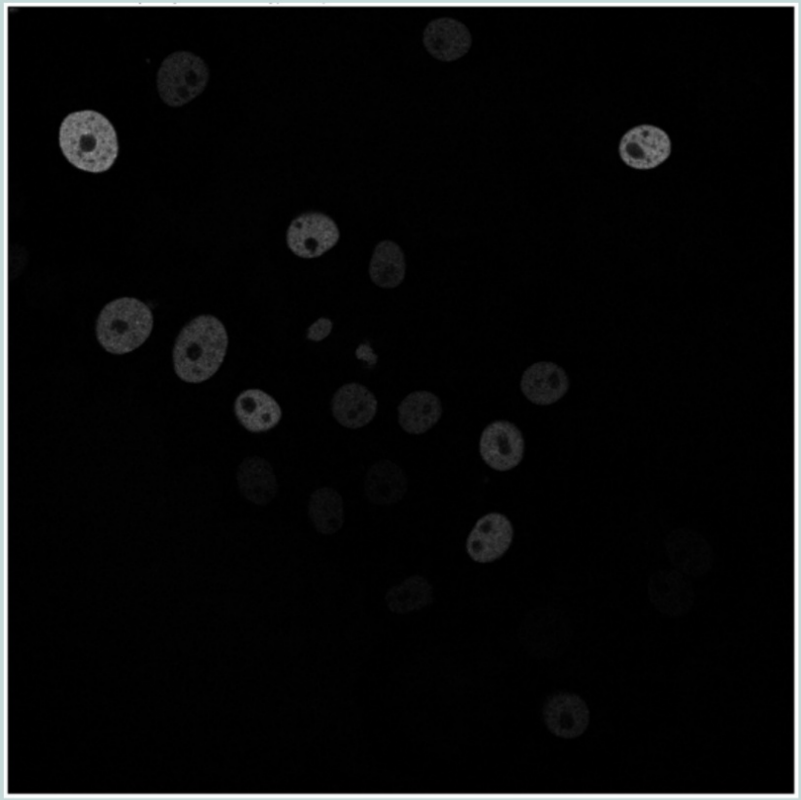


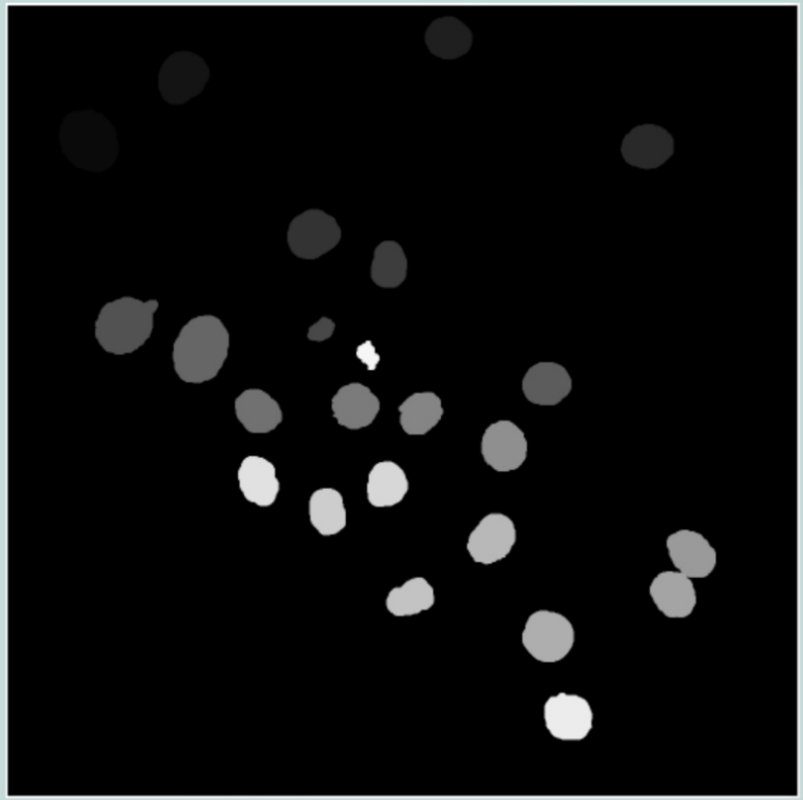
NIH3T3

- Mouse (*mus musculus*)
- Embryo fibroblast
- Contact-inhibited NIH Swiss mouse embryo
- CD-tagged protein, used EGFP



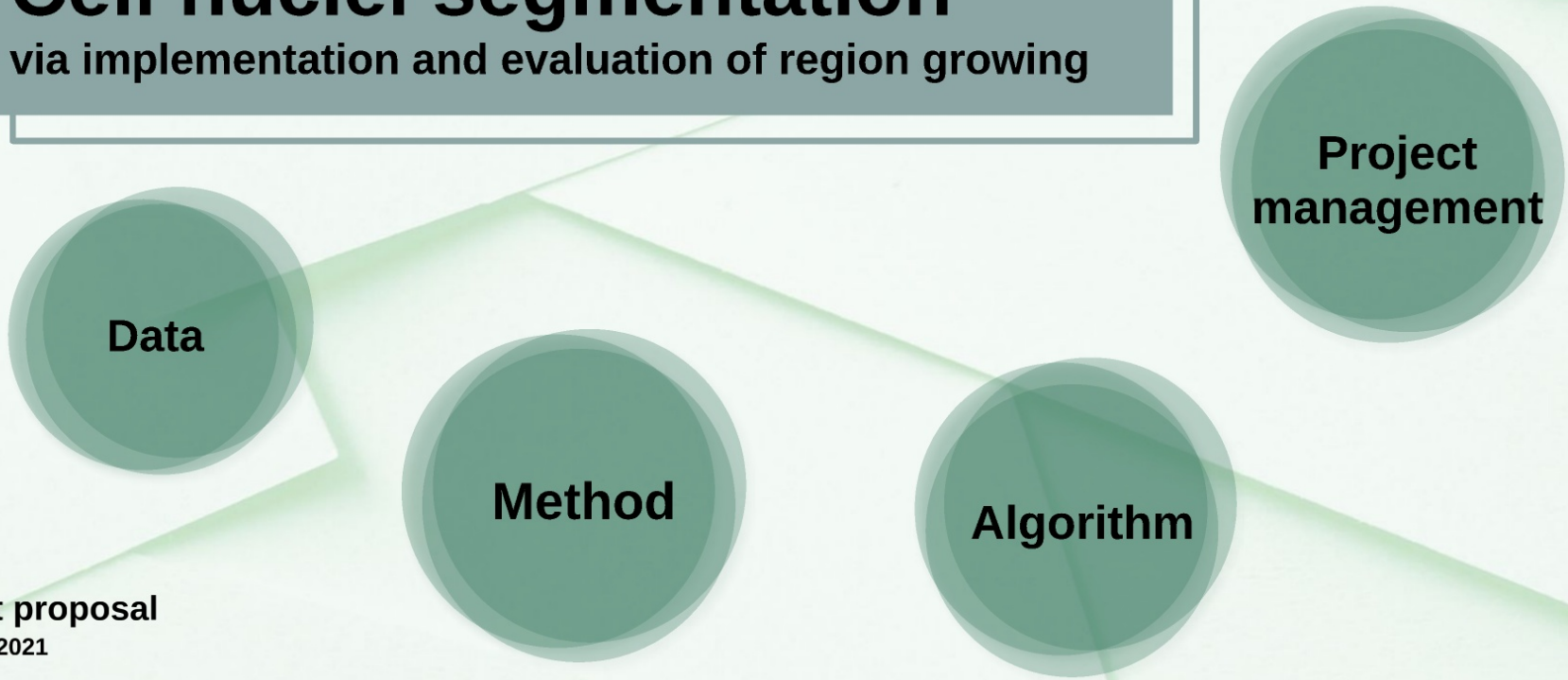
Normal image vs. ground truth image





Cell nuclei segmentation

via implementation and evaluation of region growing



Project proposal

12th May 2021

Data Analysis MoBi SS2021

Supervisor: Dr. Karl Rohr, Christian Ritter; Tutor: Nicholas Peschke

Topic 04: Biomedical image analysis

Group 04: Marie Becker, Ina Jung, Laura Kaschnitz, Johanna Möller

Method

Seeded Region Growing

Unseeded Region Growing

Seeded Region Growing

Basic procedure


Seed
selection
Region
growing
Region
merging

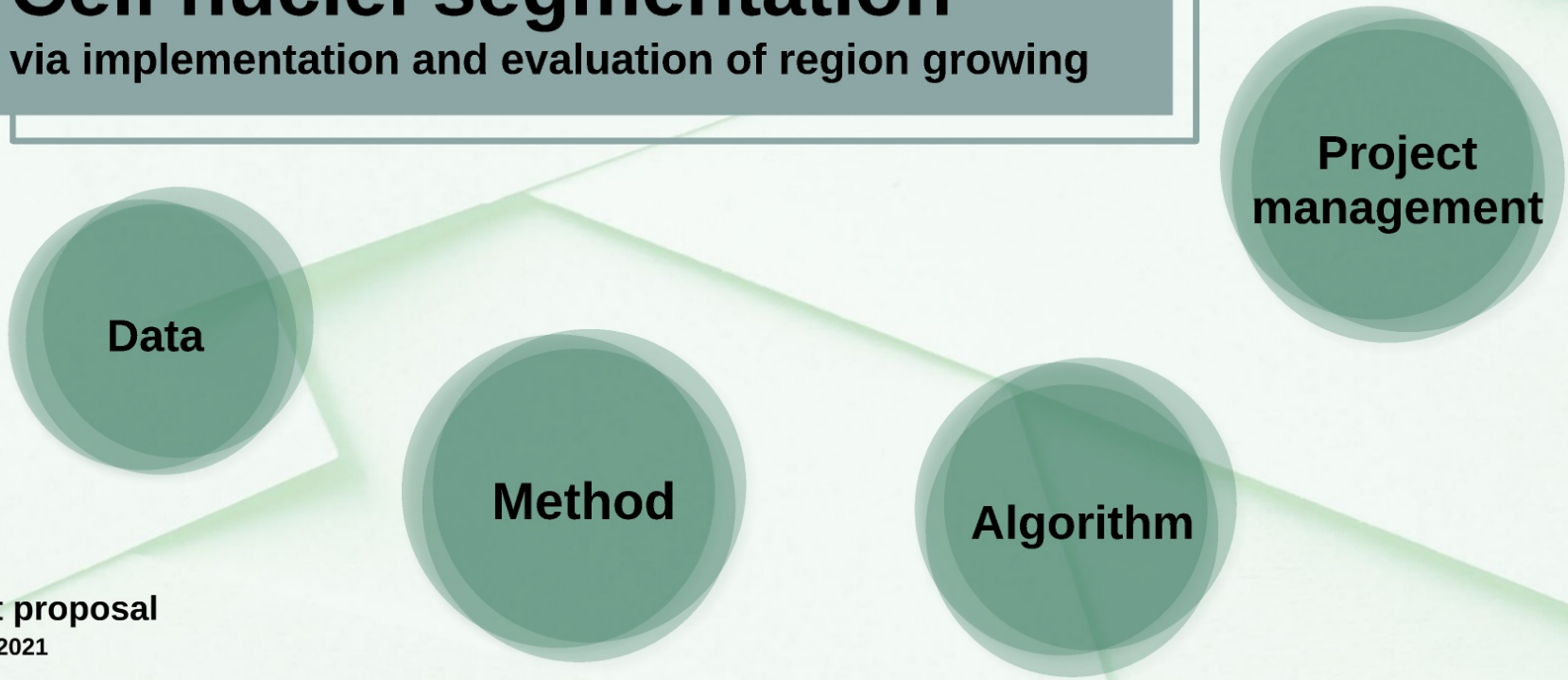
Method

Seeded Region Growing

Unseeded Region Growing

Cell nuclei segmentation

via implementation and evaluation of region growing



Project proposal

12th May 2021

Data Analysis MoBi SS2021

Supervisor: Dr. Karl Rohr, Christian Ritter; Tutor: Nicholas Peschke

Topic 04: Biomedical image analysis

Group 04: Marie Becker, Ina Jung, Laura Kaschnitz, Johanna Möller

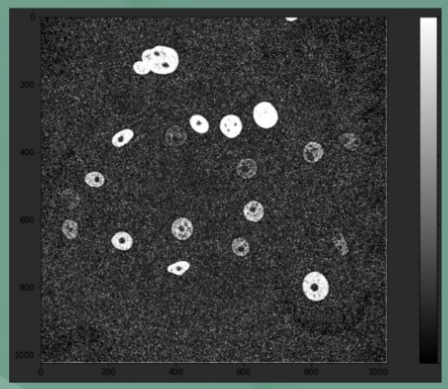
Standard deviation

Euclidean distance

Relative euclidean distance

Automatic seed selection algorithm

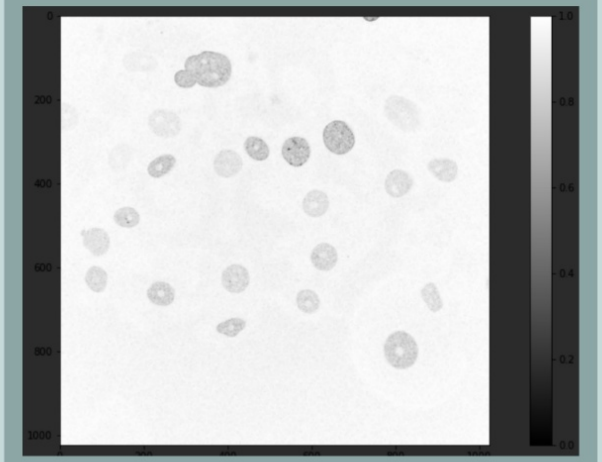
```
def seeds(img, T1, T2): # automatic seed selection algorithm
    result = numpy.zeros(img.shape)
    sd_seeds = standarddeviation(img,3) # standard deviation
    sd_flat = sd_seeds.flatten() # standard deviation as 1D-array
    similarity_seeds = 1 - sd_seeds/max(sd_flat) # calculates similarity of every pixel to its neighbors
    eurel_seeds = euclidean_relative(img, 3) # relative euclidean distance of every pixel to its neighbors
    for p in ndindex(img.shape): # border pixel value is zero
        if similarity_seeds[p] > T1 and eurel_seeds[p] < T2: # compares pixel with threshold
            result[p] = 1 # assigns value 1 to seeds
    return result

seed_final = seeds(img, 0.4, 20)  img: {ndarray: (1024, 1024)}
show_image(seed_final, 15, 8)  seed_final: {ndarray: (1024, 1024)}
```



Standard deviation

```python
def standarddeviation(img, size): # calculates standard deviation of every pixel (image, size of filter mask)
    result = numpy.zeros(img.shape) # create empty array (zeros)
    n = (size - 1) // 2
    for p in ndindex(img.shape): # iterates over every pixel
        neighborhood_sum = 0
        mean = 0
        deviation = 0
        if p[0]-n >= 0 and p[1]-n >= 0 and p[0]+n <= img.shape[0] -1 and p[1]+n <= img.shape[1] -1: # no calculation of border pixels
            for q in ndindex(size, size): # iterates over filter mask (nxn neighborhood)
                i = p[0]-n + q[0]
                j = p[1]-n + q[1]
                neighborhood_sum += img[i,j]
            mean = neighborhood_sum/(size**2) # calculate mean in nxn neighborhood
            for q in ndindex(size, size): # iterates over filter mask (nxn neighborhood)
                i = p[0]-n + q[0]
                j = p[1]-n + q[1]
                deviation += (img[i,j] - mean)**2 # calculate deviation from mean

            result[p] = sqrt(1/(size**2) * deviation) # calculate standard deviation
    return result
sd_test = standarddeviation(img,3)  img: {ndarray: (1024, 1024)}
```

```python
sd_2 = standarddeviation(img,3) # apply method for testing
sd_2 = sd_2/max(sd_2.flatten()) # normes standard deviation
similarity_2 = 1-sd_2 # calculates similarity
show_image(similarity_2, 15, 8)
```

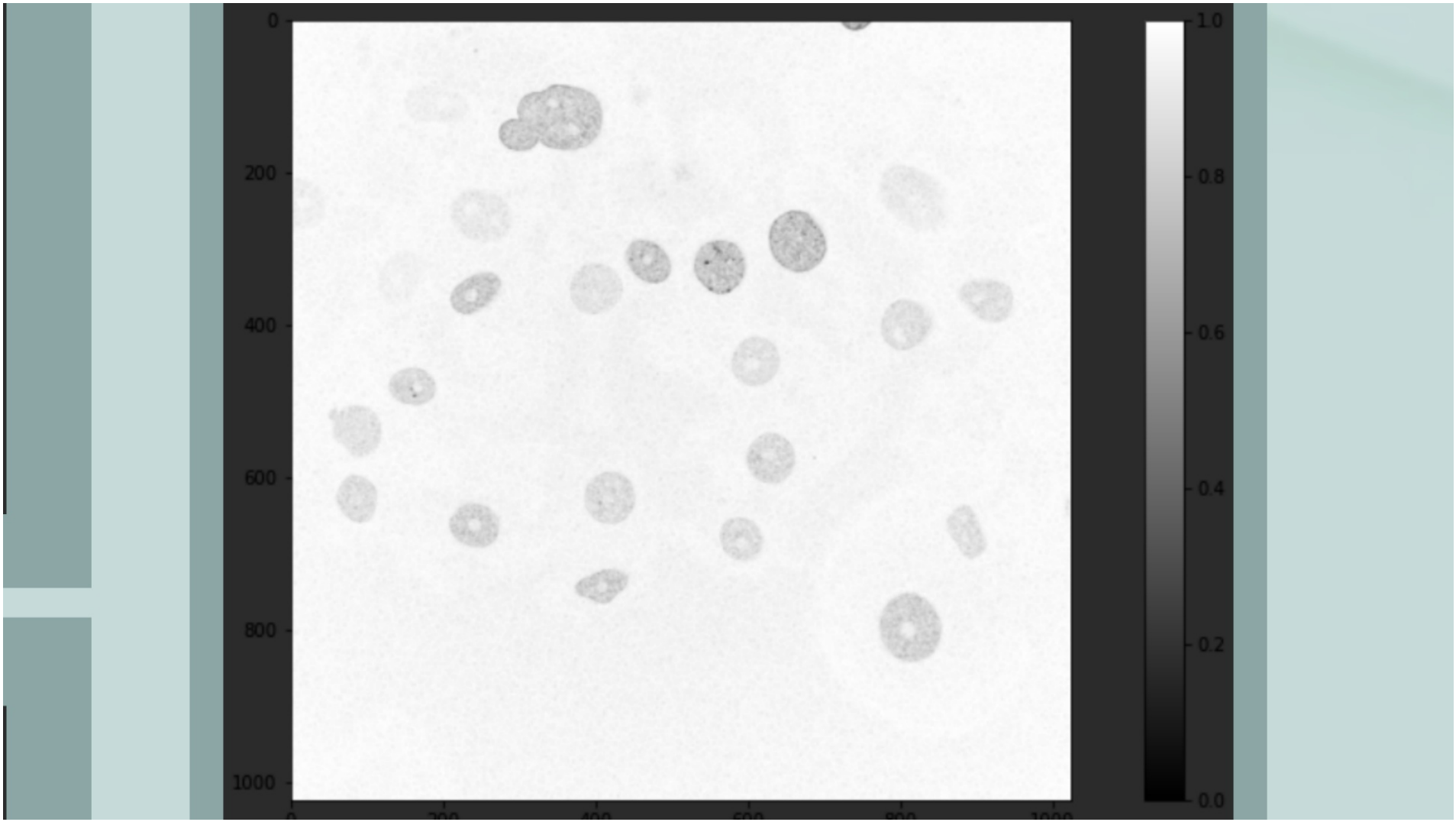


Standard deviation

```
def standarddeviation(img, size): # calculates standard deviation of every pixel (image, size of filter mask)
    result = numpy.zeros(img.shape) # create empty array (zeros)
    n = (size - 1) // 2
    for p in ndindex(img.shape): # iterates over every pixel
        neighborhood_sum = 0
        mean = 0
        deviation = 0
        if p[0]-n >= 0 and p[1]-n >= 0 and p[0]+n <= img.shape[0] -1 and p[1]+n <= img.shape[1] -1: # no calculation of border pixels
            for q in ndindex(size, size): # iterates over filter mask (nxn neighborhood)
                i = p[0]-n + q[0]
                j = p[1]-n + q[1]
                neighborhood_sum += img[i,j]
            mean = neighborhood_sum/(size**2) # calculate mean in nxn neighborhood
            for q in ndindex(size, size): # iterates over filter mask (nxn neighborhood)
                i = p[0]-n + q[0]
                j = p[1]-n + q[1]
                deviation += (img[i,j] - mean)**2 # calculate deviation from mean

            result[p] = sqrt(1/(size**2) * deviation) # calculate standard deviation
    return result
sd_test = standarddeviation(img,3)  img: {ndarray: (1024, 1024)}
```

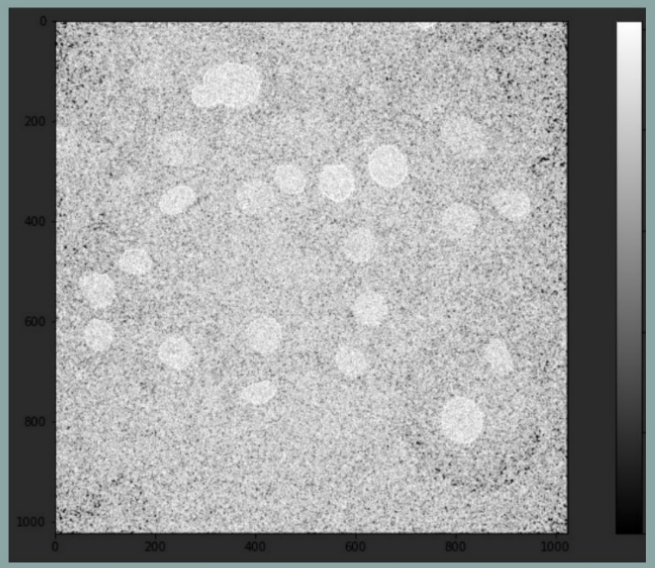
```
            deviation += (img[i,j] - mean)**2 # calculate deviation from mean

        result[p] = sqrt(1/(size**2) * deviation) # calculate standard deviation
    return result
sd_test = standarddeviation(img,3)  img: {ndarray: (1024, 1024)}
```

```
sd_2 = standarddeviation(img,3) # apply method for testing
sd_2 = sd_2/max(sd_2.flatten()) # normes standard deviation
similarity_2 = 1-sd_2 # calculates similarity
show_image(similarity_2, 15, 8)
```


0
200
400
600
800
1000
1.0
0.8
0.6
0.4
0.2
0.0

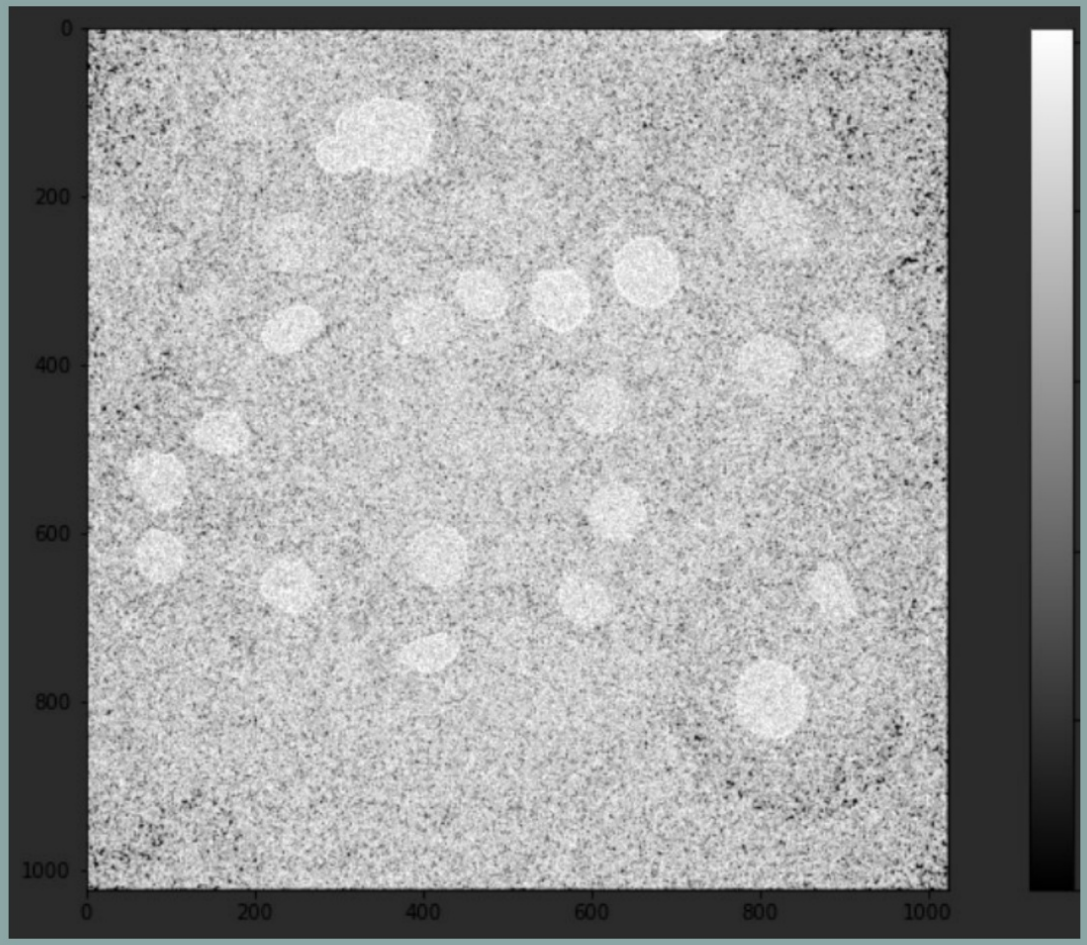
Euclidean distance

```python
def euclidean_n(img, size): # calculates maximum euclidean distance for every pixel in nxn neighborhood
    result = numpy.zeros(img.shape) # create empty array (zeros)
    n = (size - 1) // 2
    for p in ndindex(img.shape): # iterates over every pixel using a touple
        neighborhood_distance = [] # creates a list for distances
        if p[0]-n >= 0 and p[1]-n >= 0 and p[0]+n <= img.shape[0] -1 and p[1]+n <= img.shape[1] -1: # no calculation of border pixels
            for q in ndindex(size, size): # iterates over nxn neighborhood
                i = p[0]-n + q[0]
                j = p[1]-n + q[1]
                neighborhood_distance.append((img[p]-img[i,j])) # adds euclidean distance to list
            result[p] = max(neighborhood_distance) # chooses maximum distance
    return result
```



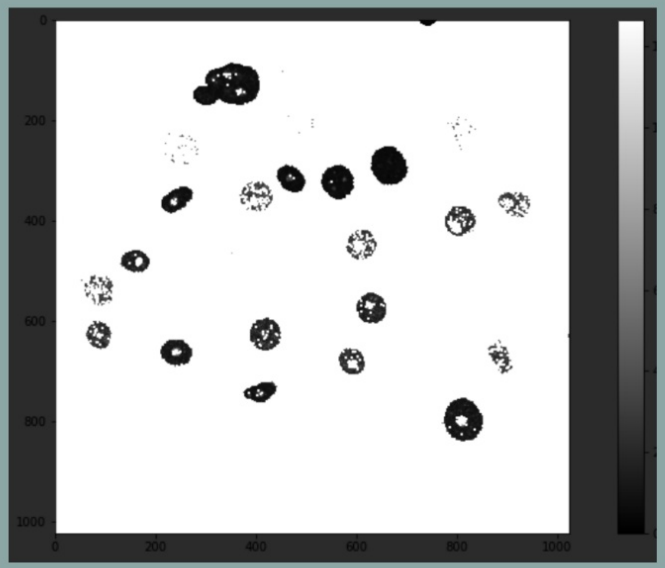
```python
def euclidean_n(img, size): # calculates maximum euclidean distance for every pixel in nxn neighborhood
    result = numpy.zeros(img.shape) # create empty array (zeros)
    n = (size - 1) // 2
    for p in ndindex(img.shape): # iterates over every pixel using a touple
        neighborhood_distance = [] # creates a list for distances
        if p[0]-n >= 0 and p[1]-n >= 0 and p[0]+n <= img.shape[0] -1 and p[1]+n <= img.shape[1] -1: # no calculation of border pixels
            for q in ndindex(size, size): # iterates over nxn neighborhood
                i = p[0]-n + q[0]
                j = p[1]-n + q[1]
                neighborhood_distance.append((img[p]-img[i,j])) # adds euclidean distance to list
            result[p] = max(neighborhood_distance) # chooses maximum distance
    return result
```

der pixels

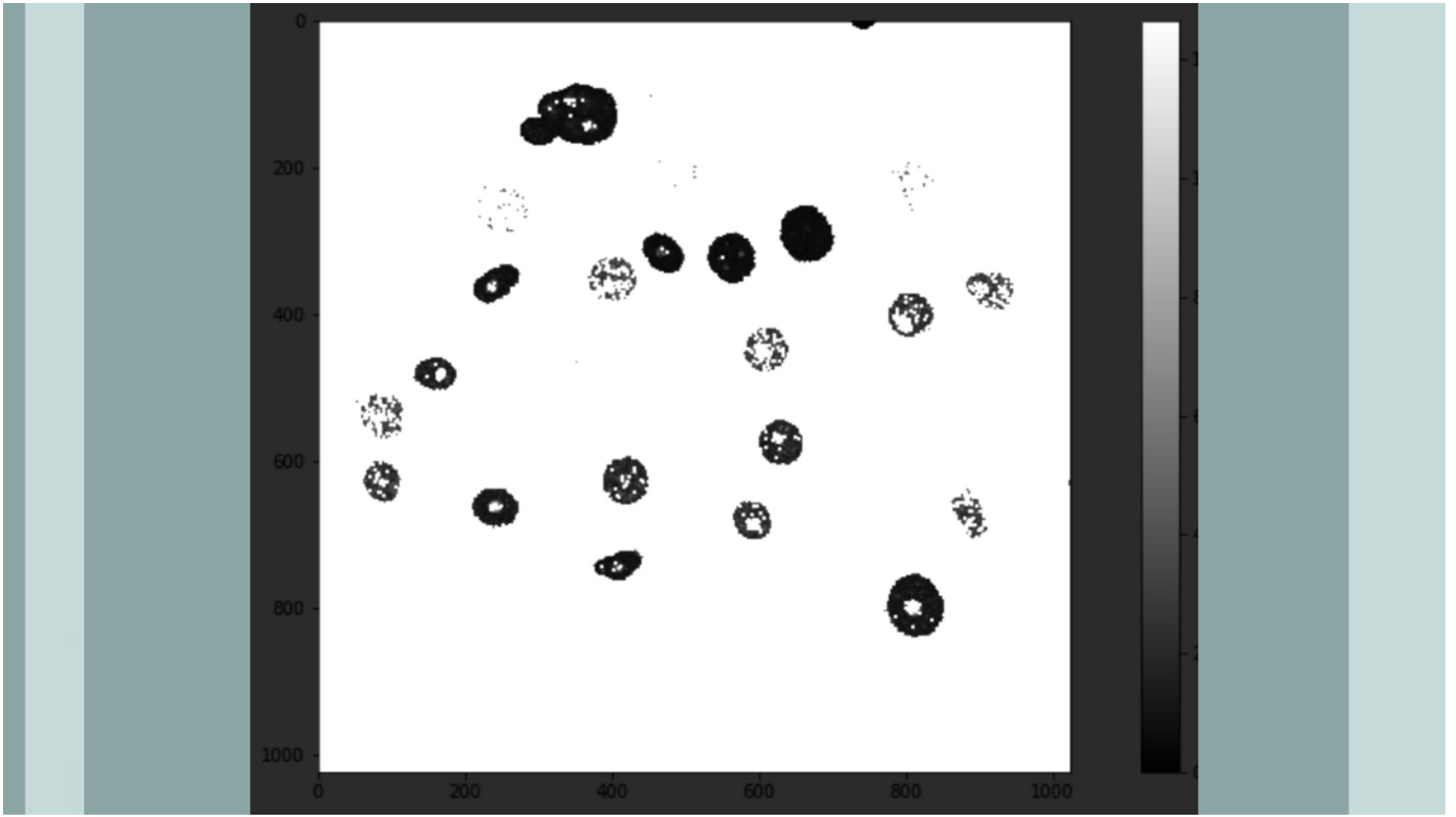


Relative euclidean distance

```python
def euclidean_relative(img, size): # calculates maximum relative euclidean distance for every pixel in nxn neighborhood
    result = numpy.zeros(img.shape) # create empty array (zeros)
    n = (size - 1) // 2
    for p in ndindex(img.shape): # iterates over every pixel using a touple
        neighborhood_distance = [] # creates a list for distances
        if p[0]-n >= 0 and p[1]-n >= 0 and p[0]+n <= img.shape[0] -1 and p[1]+n <= img.shape[1] -1: # no calculation of border pixels
            for q in ndindex(size, size): # iterates over nxn neighborhood
                i = p[0]-n + q[0]
                j = p[1]-n + q[1]
                neighborhood_distance.append((img[p]-img[i,j])/img[p]) # adds relative euclidean distance to list

            result[p] = max(neighborhood_distance) # chooses maximum distance
    return result
```

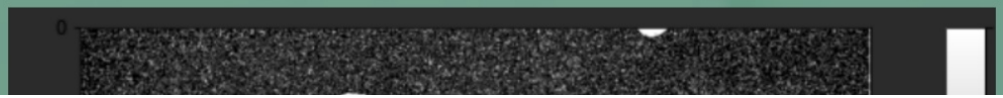


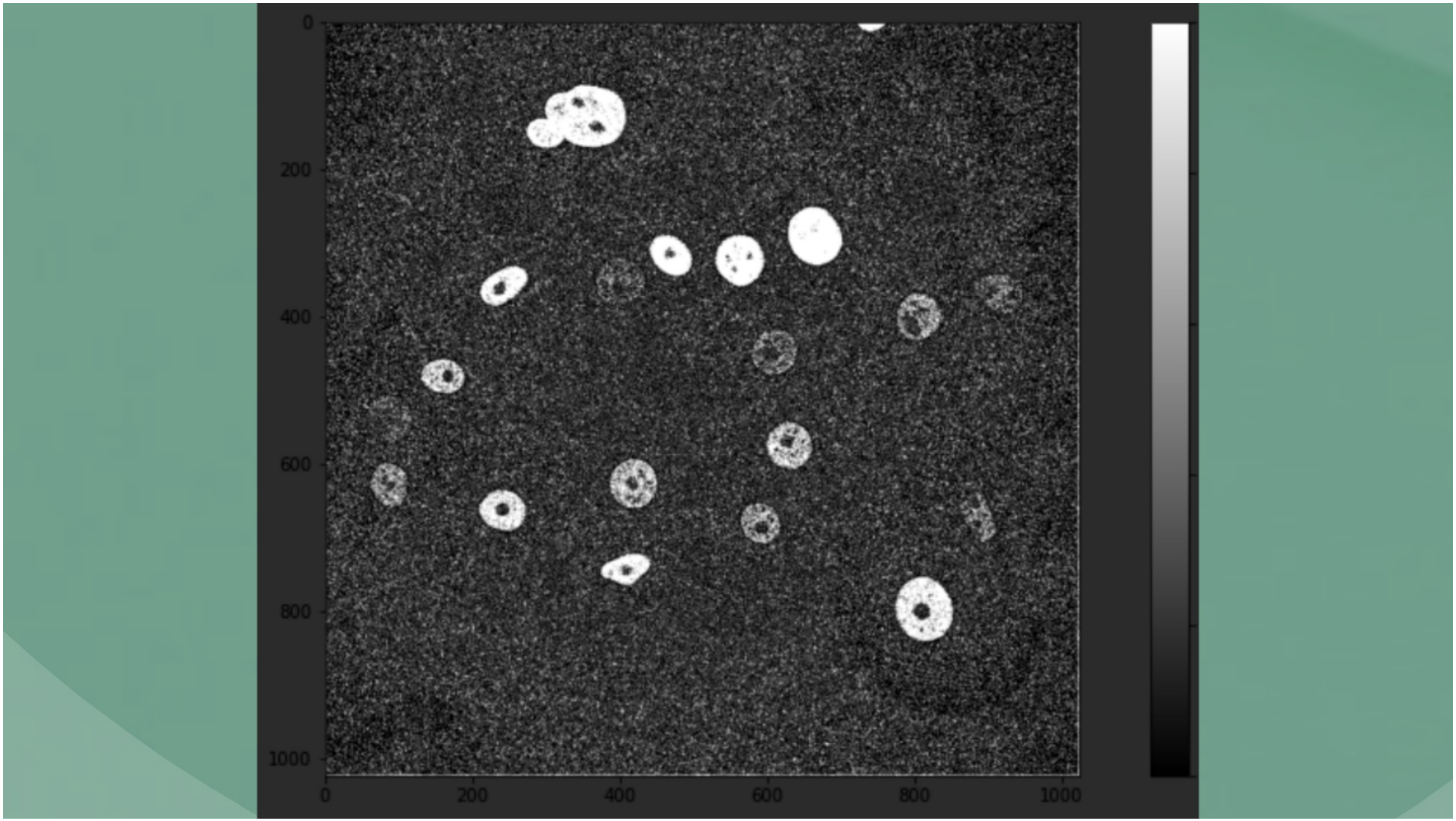
```
def euclidean_relative(img, size): # calculates maximum relative euclidean distance for every pixel in nxn neighborhood
    result = numpy.zeros(img.shape) # create empty array (zeros)
    n = (size - 1) // 2
    for p in ndindex(img.shape): # iterates over every pixel using a touple
        neighborhood_distance = [] # creates a list for distances
        if p[0]-n >= 0 and p[1]-n >= 0 and p[0]+n <= img.shape[0] -1 and p[1]+n <= img.shape[1] -1: # no calculation of border pixels
            for q in ndindex(size, size): # iterates over nxn neighborhood
                i = p[0]-n + q[0]
                j = p[1]-n + q[1]
                neighborhood_distance.append((img[p]-img[i,j])/img[p]) # adds relative euclidean distance to list

            result[p] = max(neighborhood_distance) # chooses maximum distance
    return result
```


0
200
400
600
800
1000
0
200
400
600
800
1000

selection algorithm

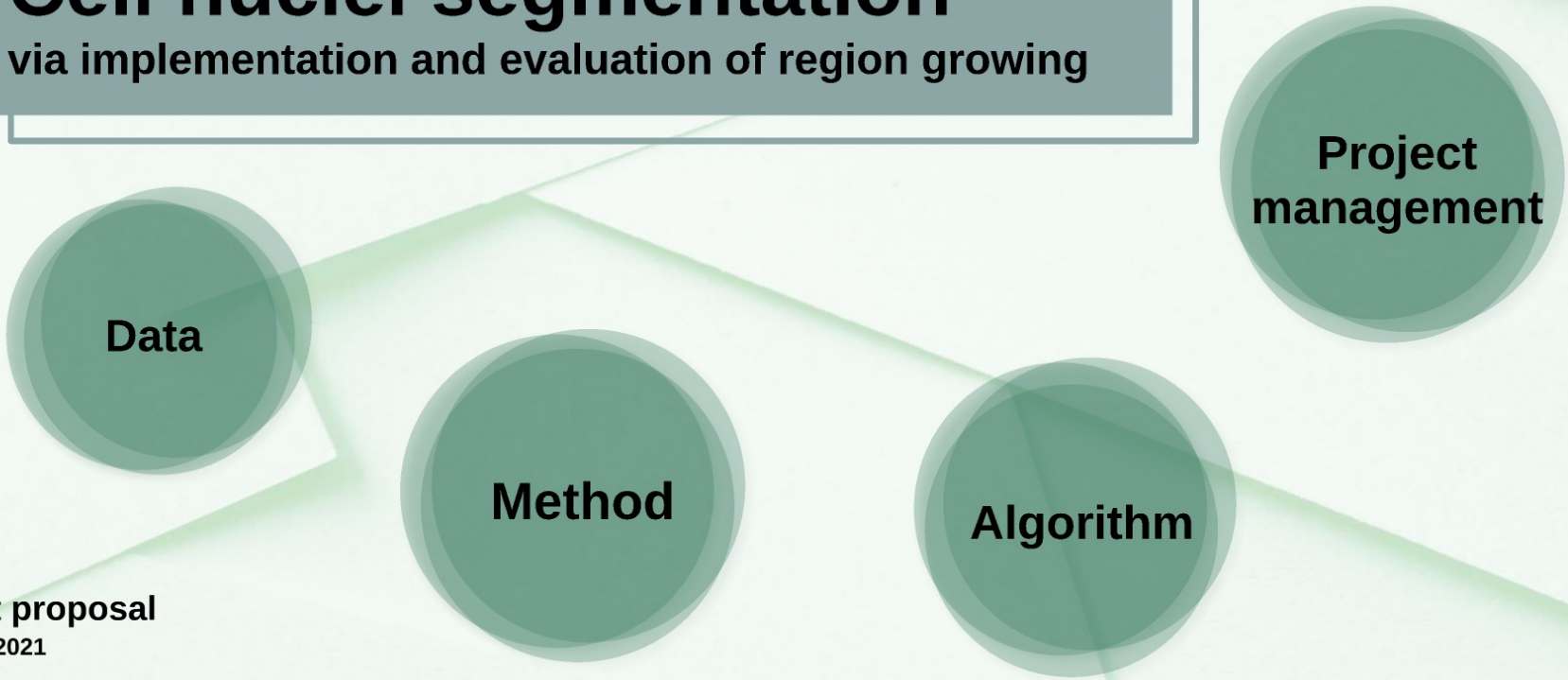
```python
def seeds(img, T1, T2): # automatic seed selection algorithm
    result = numpy.zeros(img.shape)
    sd_seeds = standarddeviation(img,3) # standard deviation
    sd_flat = sd_seeds.flatten() # standard deviation as 1D-array
    similarity_seeds = 1 - sd_seeds/max(sd_flat) # calculates similarity of every pixel to its neighbors
    eurel_seeds = euclidean_relative(img, 3) # relative euclidean distance of every pixel to its neighbors
    for p in ndindex(img.shape): # border pixel value is zero
        if similarity_seeds[p] > T1 and eurel_seeds[p] < T2: # compares pixel with threshold
            result[p] = 1 # assigns value 1 to seeds
    return result

seed_final = seeds(img, 0.4, 20)  img: {ndarray: (1024, 1024)}
show_image(seed_final, 15, 8)  seed_final: {ndarray: (1024, 1024)}
```




0
200
400
600
800
1000
0
200
400
600
800
1000

Cell nuclei segmentation

via implementation and evaluation of region growing



Project proposal

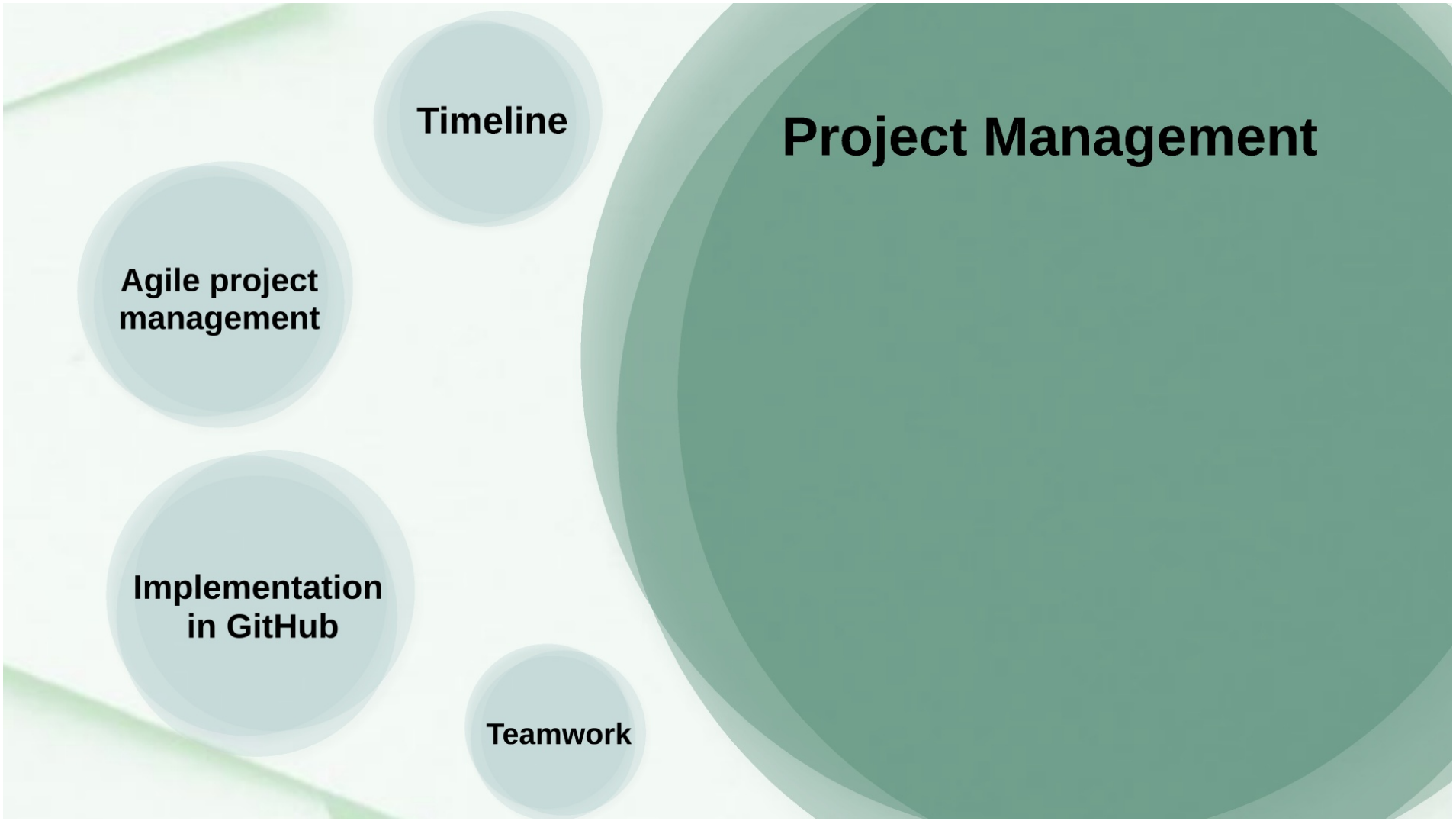
12th May 2021

Data Analysis MoBi SS2021

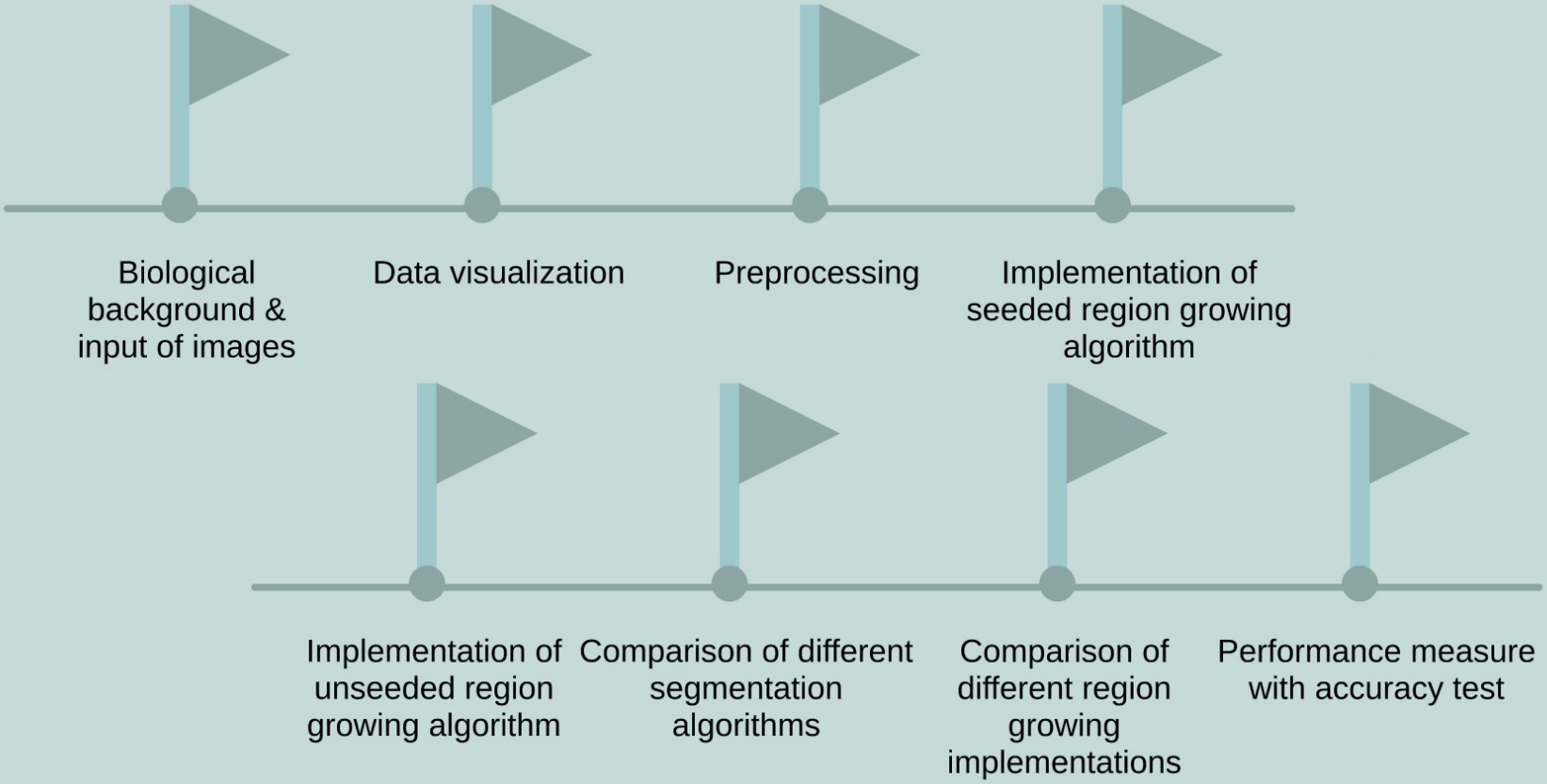
Supervisor: Dr. Karl Rohr, Christian Ritter; Tutor: Nicholas Peschke

Topic 04: Biomedical image analysis

Group 04: Marie Becker, Ina Jung, Laura Kaschnitz, Johanna Möller

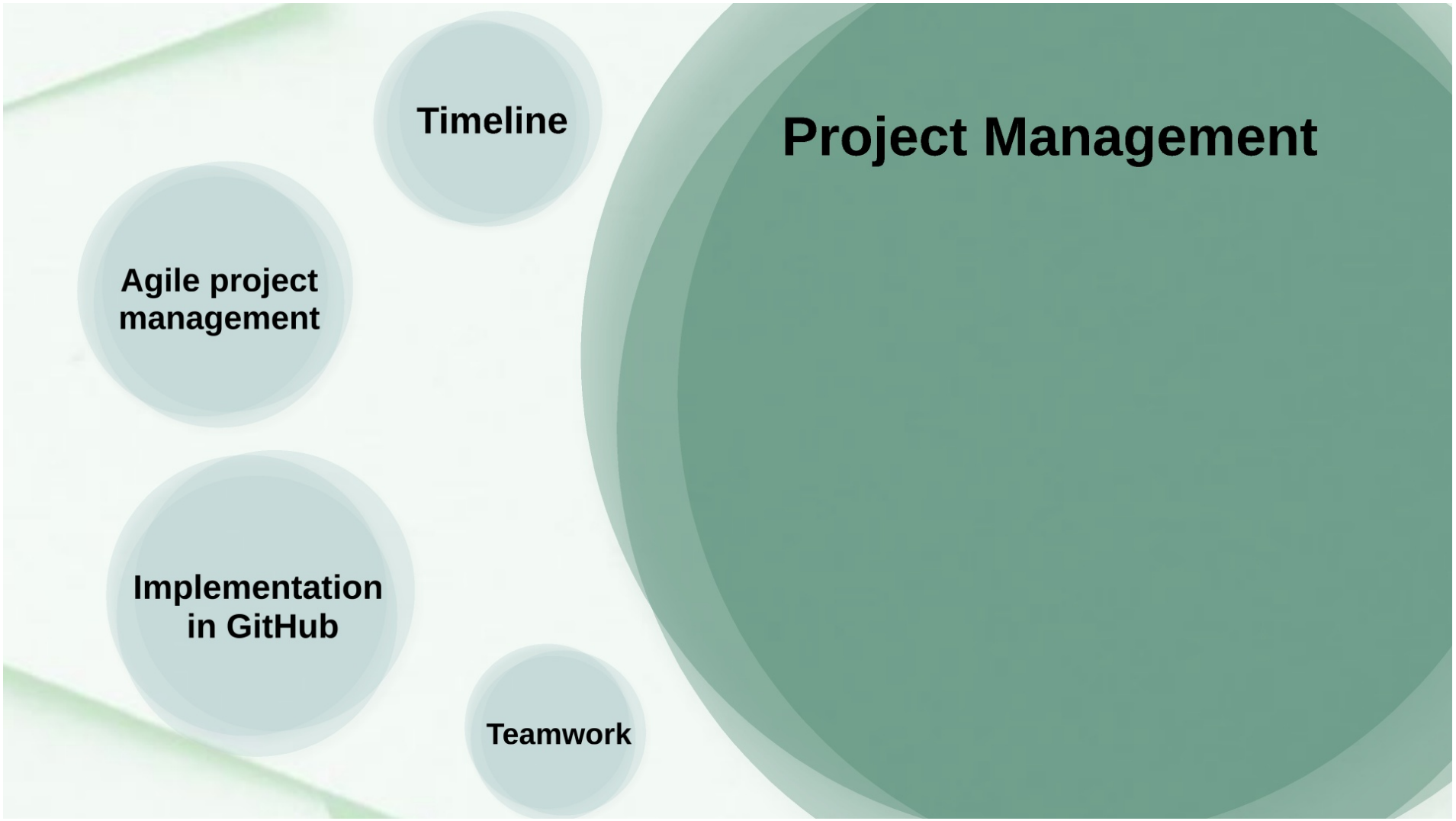

Timeline
Project Management
Agile project management
Implementation in GitHub
Teamwork

Milestones


Biological background & input of images
Data visualization
Preprocessing
Implementation of seeded region growing algorithm
Implementation of unseeded region growing algorithm
Comparison of different segmentation algorithms
Comparison of different region growing implementations
Performance measure with accuracy test

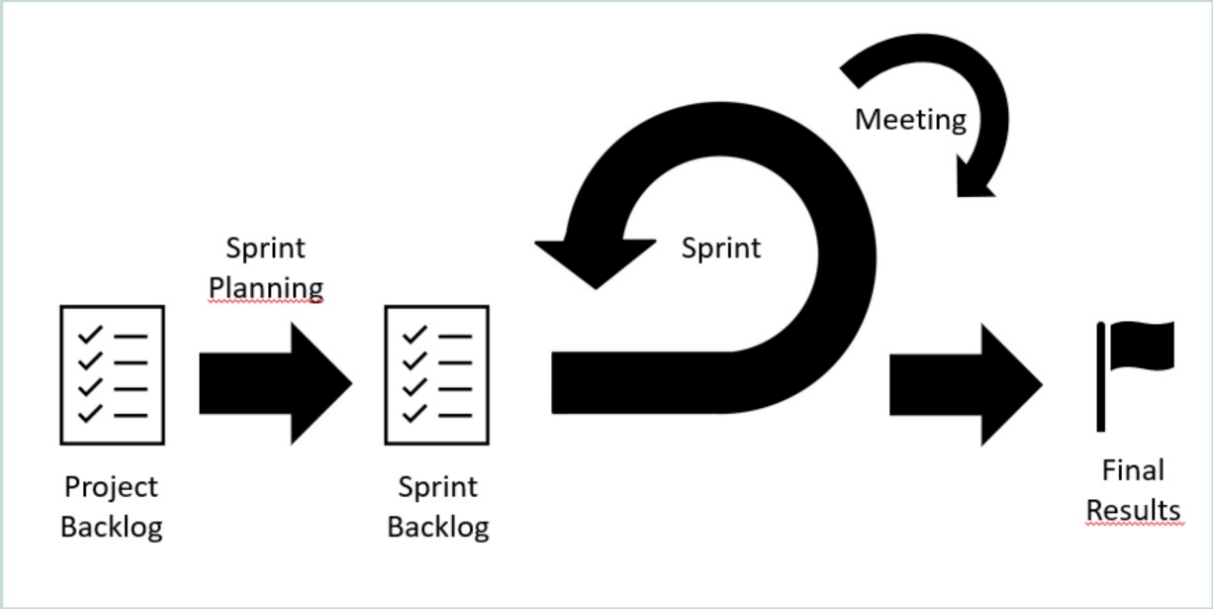
Calendar

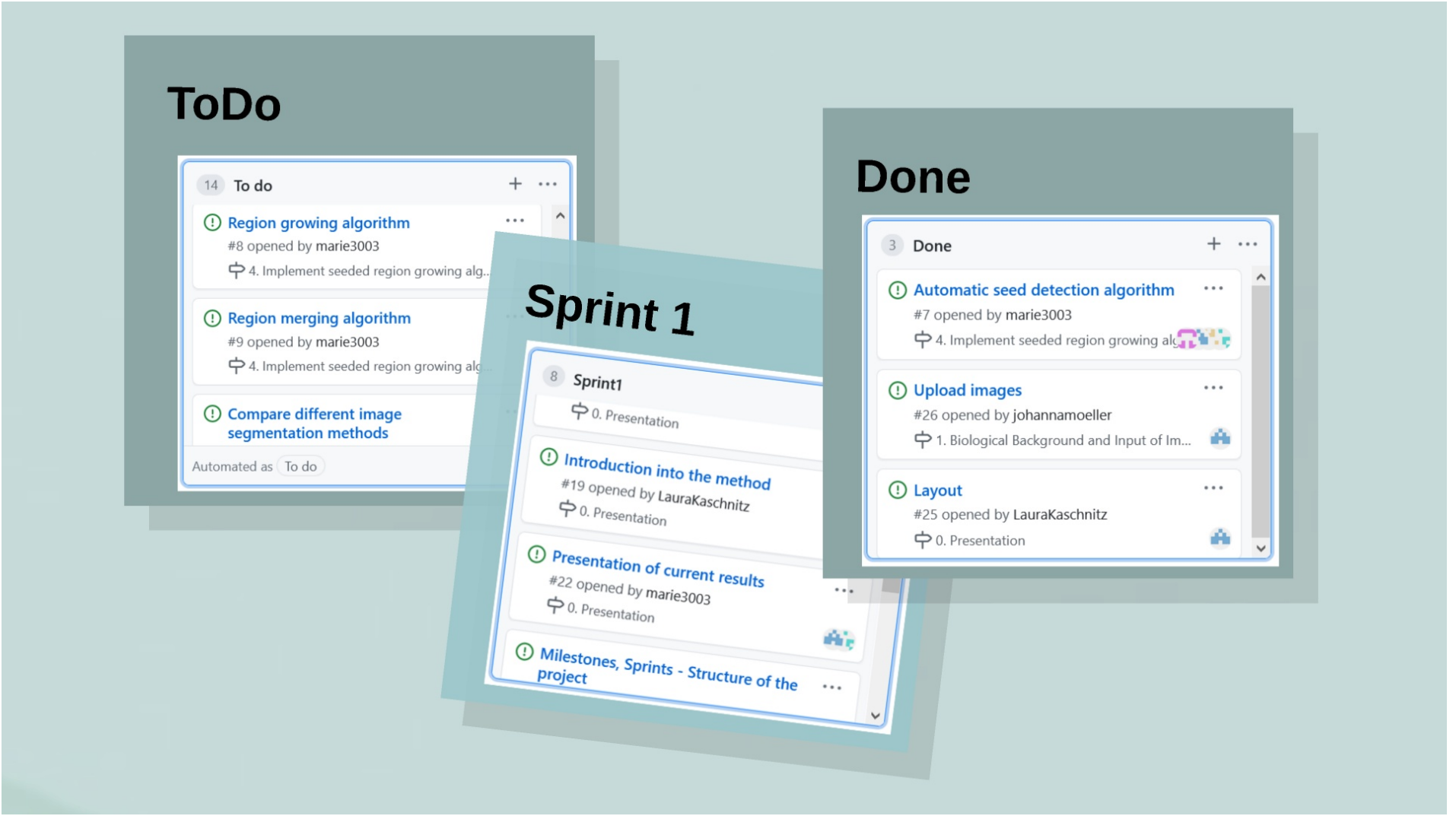
Project timetable		
Sprint 1	**Sprint 2**	**Sprint 3**
• Biological background & input of images • Data visualization		


Timeline
Project Management
Agile project management
Implementation in GitHub
Teamwork

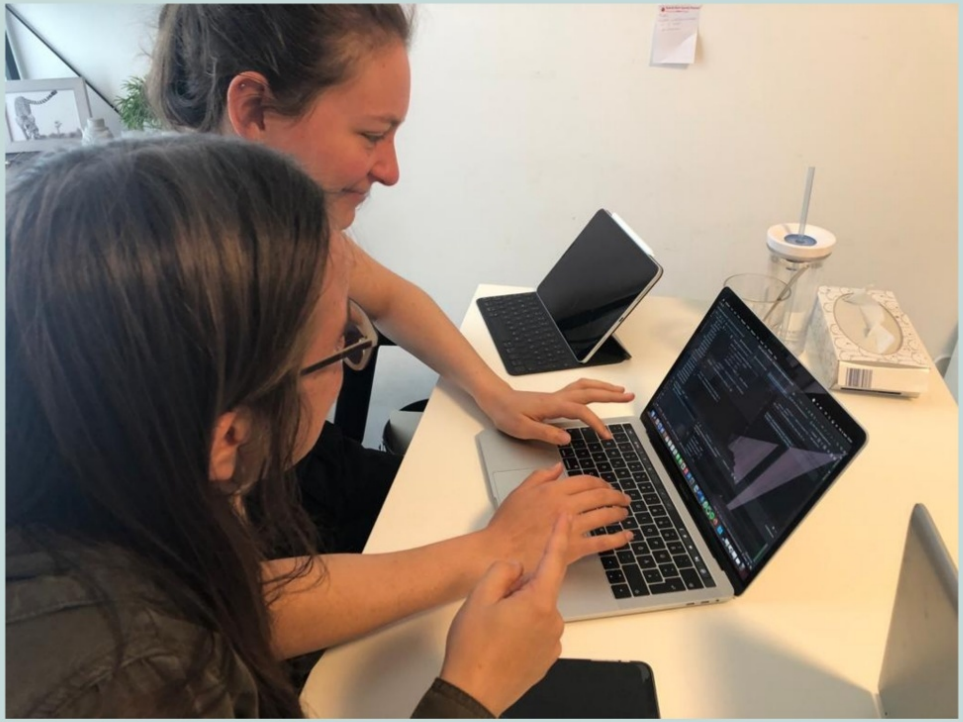
Agile Project Management

Scrum



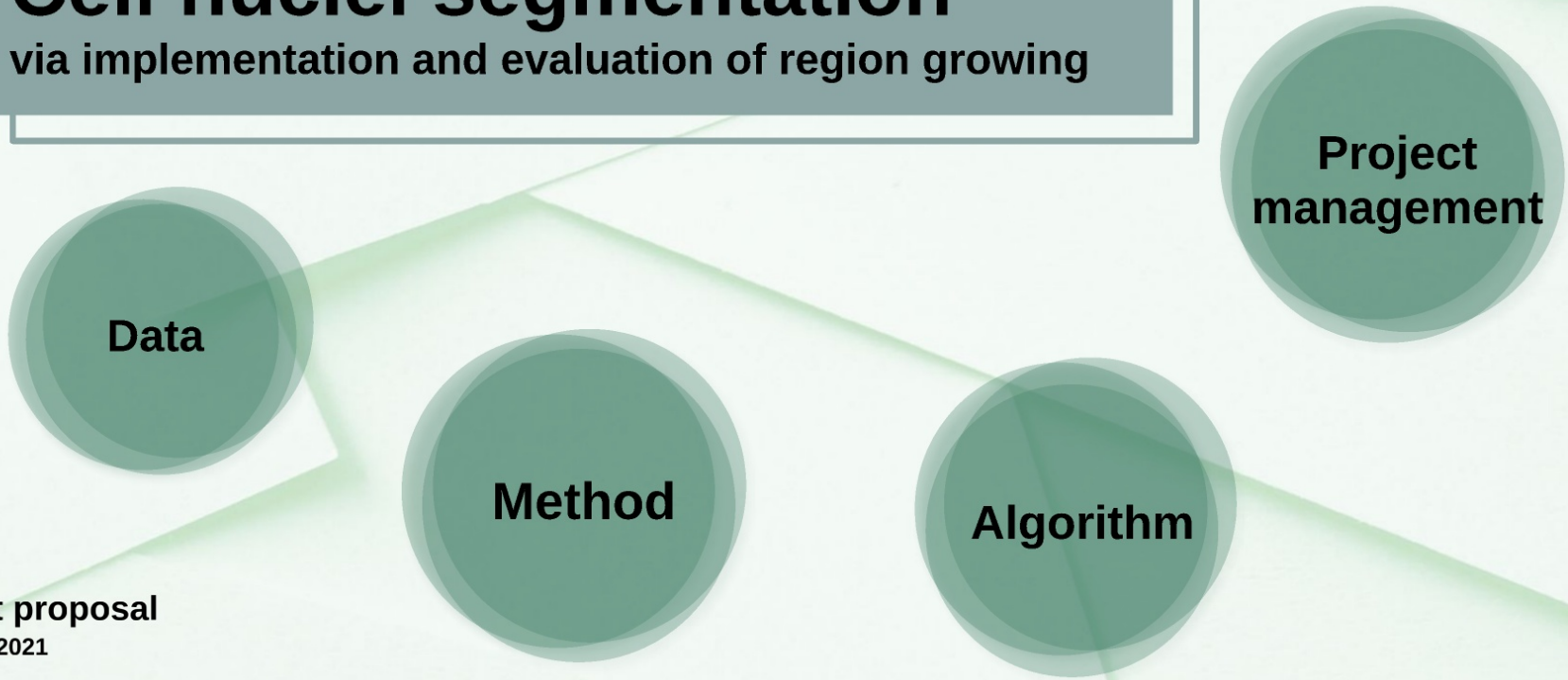

ToDo
14 To do
Region growing algorithm
#8 opened by marie3003
4. Implement seeded region growing alg...
Region merging algorithm
#9 opened by marie3003
4. Implement seeded region growing alg...
Compare different image segmentation methods
Automated as To do
Sprint 1
8 Sprint1
0. Presentation
Introduction into the method
#19 opened by LauraKaschnitz
0. Presentation
Presentation of current results
#22 opened by marie3003
0. Presentation
Milestones, Sprints - Structure of the project
Done
3 Done
Automatic seed detection algorithm
#7 opened by marie3003
4. Implement seeded region growing alg
Upload images
#26 opened by johannamoeller
1. Biological Background and Input of Im...
Layout
#25 opened by LauraKaschnitz
0. Presentation

Pair programming



Cell nuclei segmentation

via implementation and evaluation of region growing



Project proposal

12th May 2021

Data Analysis MoBi SS2021

Supervisor: Dr. Karl Rohr, Christian Ritter; Tutor: Nicholas Peschke

Topic 04: Biomedical image analysis
Group 04: Marie Becker, Ina Jung, Laura Kaschnitz, Johanna Möller