ScaDS.AI
DRESDEN LEIPZIG

CENTER FOR SCALABLE DATA ANALYTICS AND ARTIFICIAL INTELLIGENCE

Delayed Data Processing

Robert Haase

Funded by

Bundesministerium für Bildung und Forschung

SACHSEN

Diese Maßnahme wird gefördert durch die Bundesregierung aufgrund eines Beschlusses des Deutschen Bundestages. Diese Maßnahme wird mitfinanziert durch Steuermittel auf der Grundlage des von den Abgeordneten des Sächsischen Landtags beschlossenen Haushaltes.

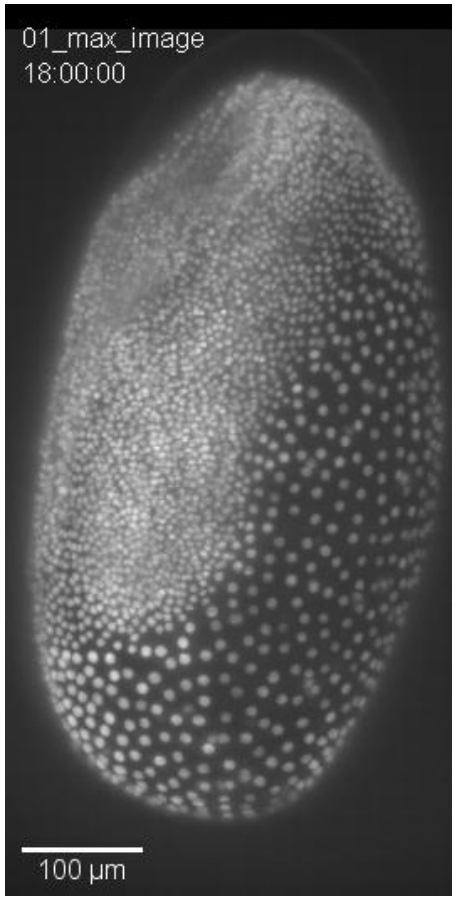
Quiz: Memory constraints

- Assume your computer runs out of memory while processing this image dataset.
  What can you do to avoid this?

1.

2.

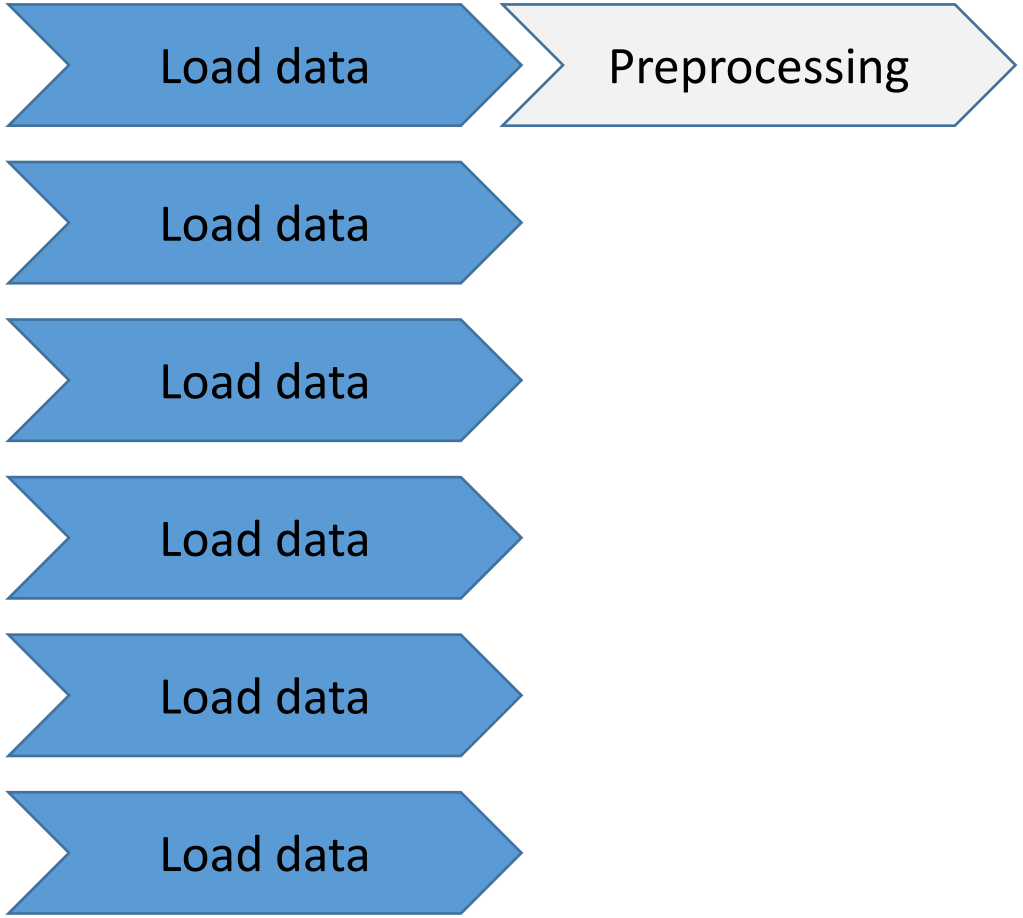
3. Tiled image processing



The full data set is about
1024 (width)
2048 (height)
100 (depth)
3700 (frames)
large

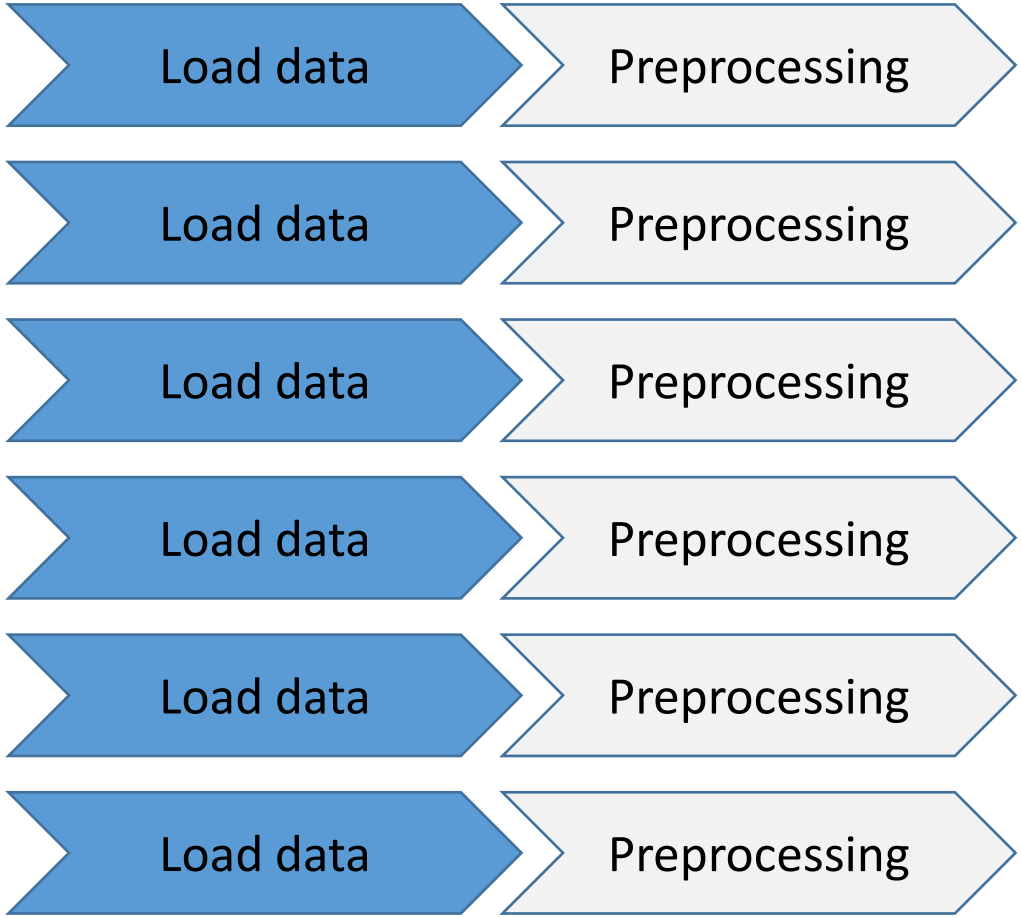
Optimal performance through smart memory management

- The classical way of dealing with large image stacks…



Optimal performance through smart memory management

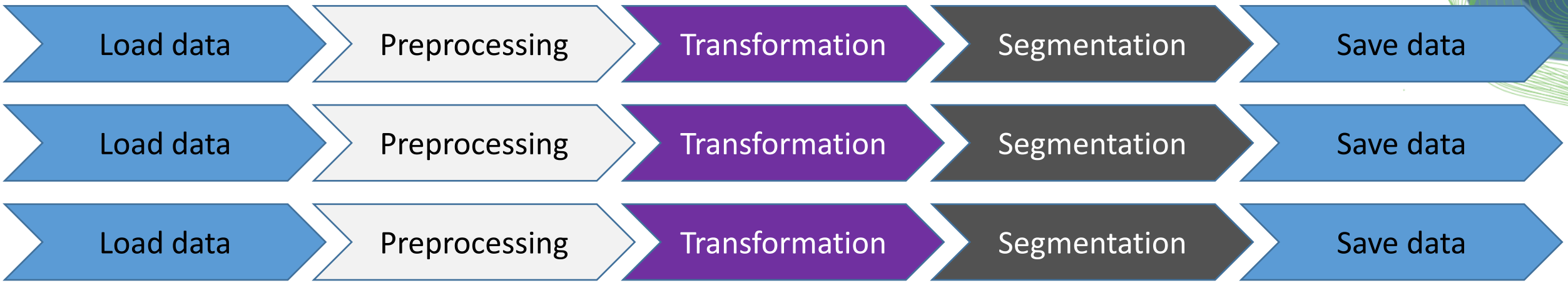
- The classical way of dealing with large image stacks… is suboptimal



This strategy does not just take long; it also costs a lot of memory!

Optimal performance through smart memory management

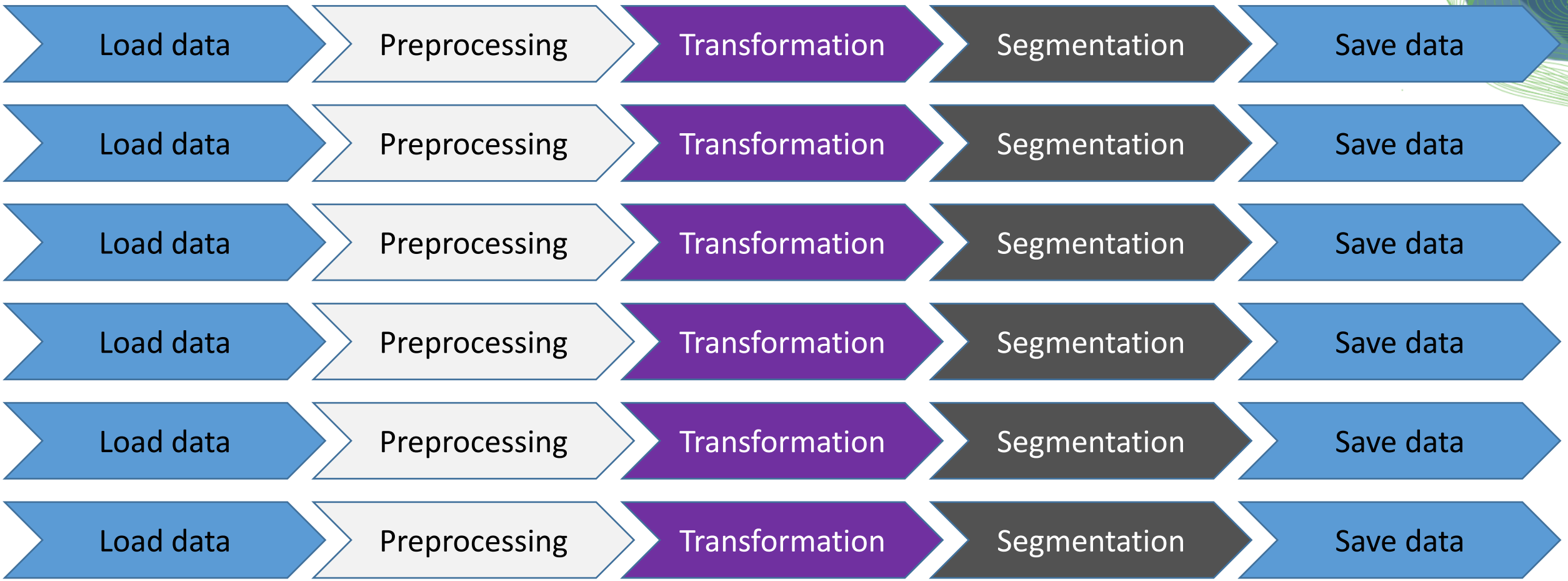
- Processing time-point by time-point is more efficient!



This strategy also works tile-by-tile on large 3D stacks and timelapse data!

Optimal performance through smart memory management

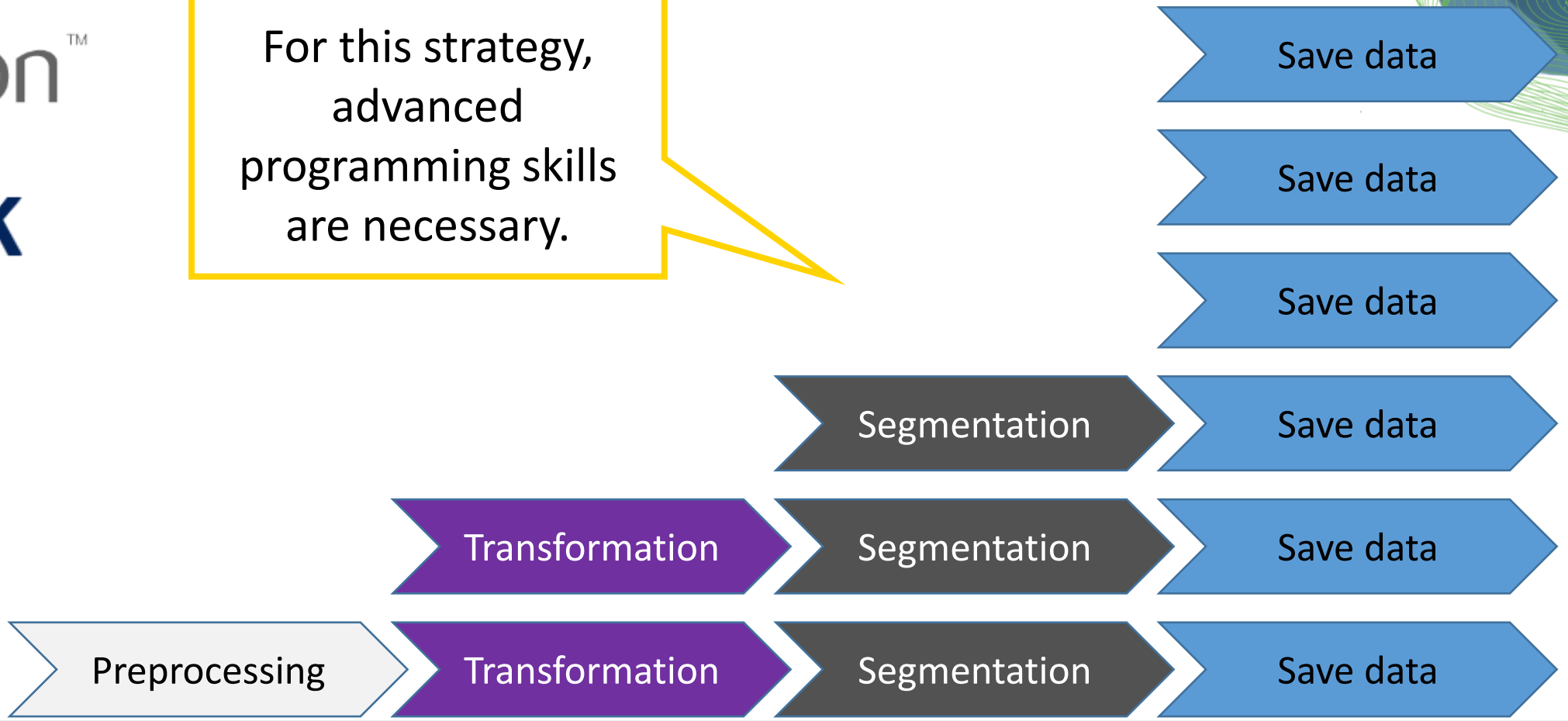
- Even better: Distribute tasks between parallelized computation systems



Optimal performance through smart memory management

- Even better: Distribute tasks between parallelized computation systems

For this strategy, advanced programming skills are necessary.



Delayed processing with Dask

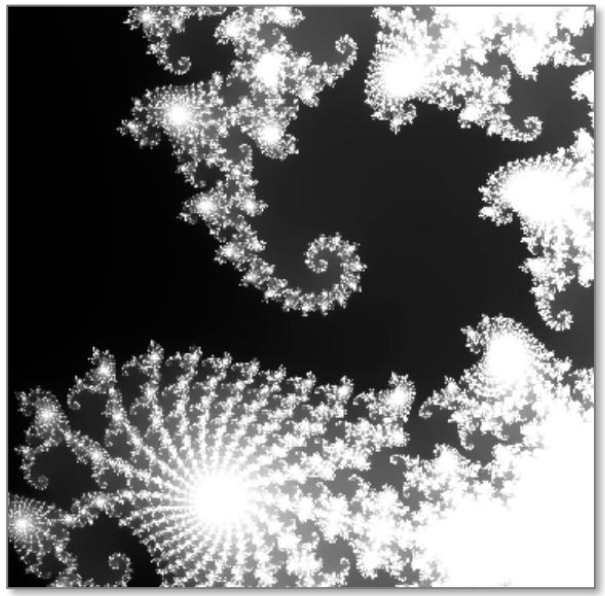
- Not not deal with details behind scheduling data loading and processing, we use Dask `delayed` and Dask `stacks`.

```
import dask
import dask.array as da
```

Decorator

Normal Python function

```
@delayed
def render_level(i):
    from mandelbrot import SCALE0, ZOOM_PER_LEVEL
    scale = SCALE0 / (ZOOM_PER_LEVEL ** i)
    return mandelbrot_array(scale=scale)
```



Delayed processing with Dask

- Not not deal with details behind scheduling data loading and processing, we use Dask `delayed` and Dask `stacks`.

```
levels = [
    da.from_delayed(
        render_level(i),
        shape=(HEIGHT, WIDTH), dtype=DTYPE
    )
    for i in range(number_of_images)
]
stack = da.stack(levels, axis=0)
```

Delayed function calls, NOT executed (yet)

Stack of *delayed* / not yet computed data

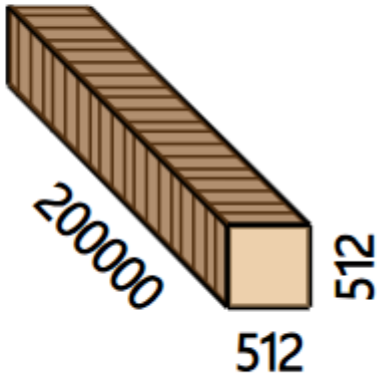
Delayed processing with Dask

- Not not deal with details behind scheduling data loading and processing, we use Dask `delayed` and Dask `stacks`.

```
[20]: stack
```

[20]:

	Array	Chunk
Bytes	97.66 GiB	512.00 kiB
Shape	(200000, 512, 512)	(1, 512, 512)
Dask graph	200000 chunks in 400001 graph layers	
Data type	uint16 numpy.ndarray	



At this point, no image has been computed yet.

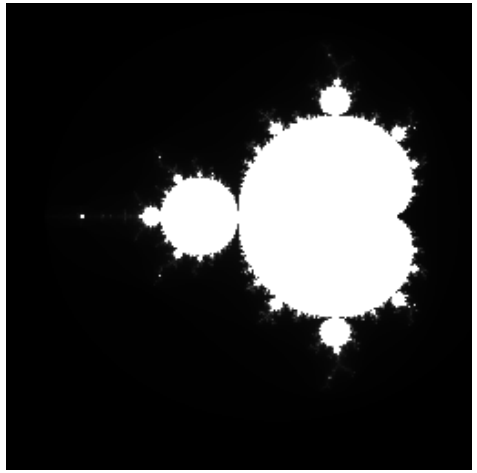
Delayed processing with Dask

- Not not deal with details behind scheduling data loading and processing, we use Dask `delayed` and Dask `stacks`.

```
[22]: stackview.imshow(stack[20000])
```

Computation of one chunk is invoked when we need the data

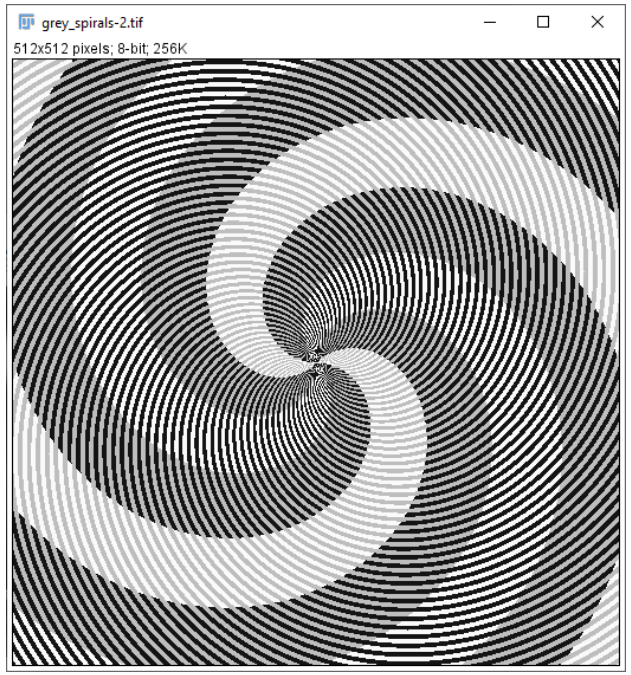
```
[11]: image = np.asarray(stack[0:20000:1000,::2,::2])
      image.shape

[11]: (20, 256, 256)

[17]: stackview.animate(image)
```



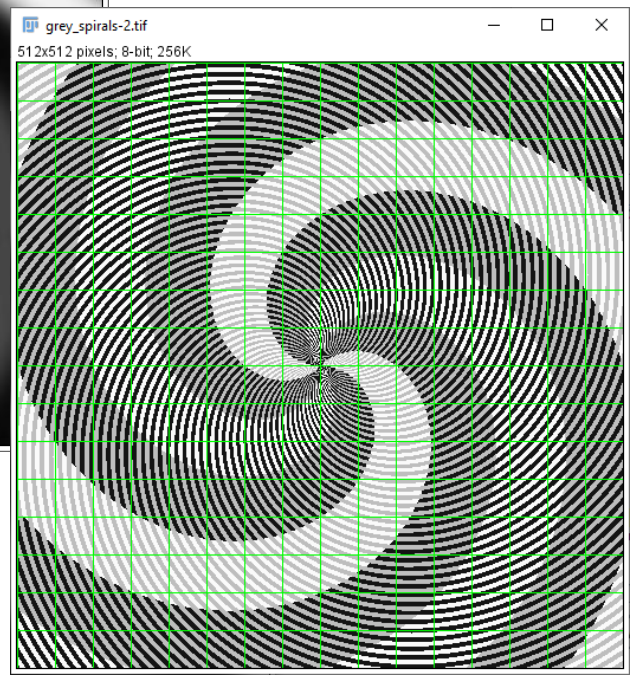
Also works with multiple chunks

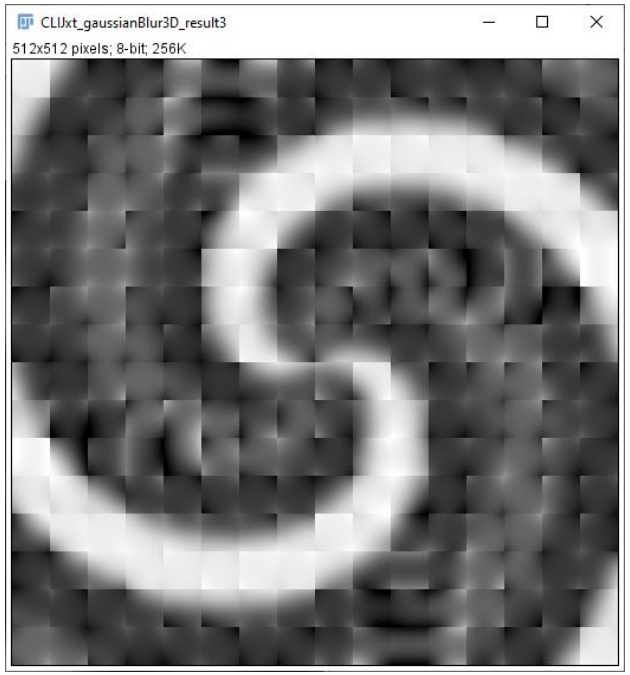
Tiling

- The last perimeter against processing big image data



If the image is too large for the computer memory, image processing as a whole is *not possible.*





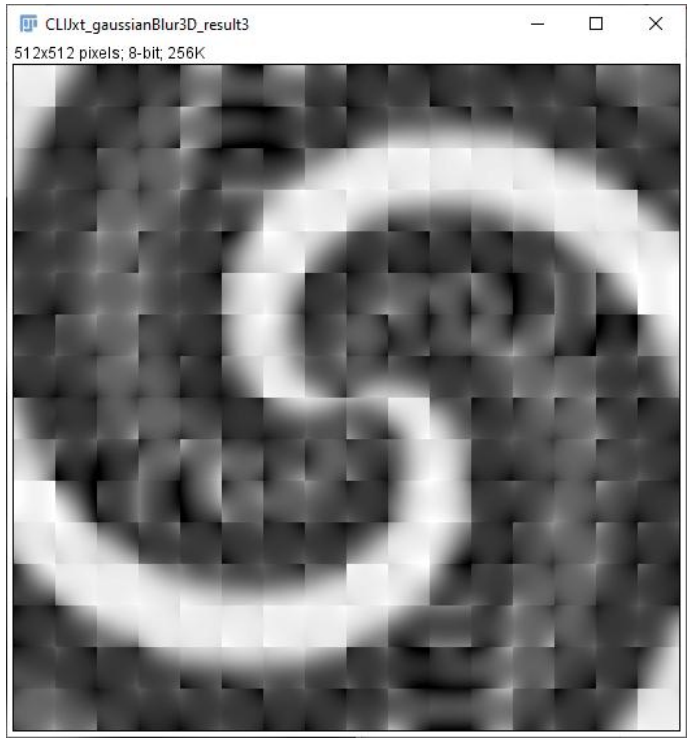
Processing tile-by-tile poses new challenges

Tiling

- Example: Gaussian blur (sigma = 20)
- Solution: Process with overlapping tiles (size + margin)

Optimal margin size depends on algorithm and its parameters

Margin: 0 pixels

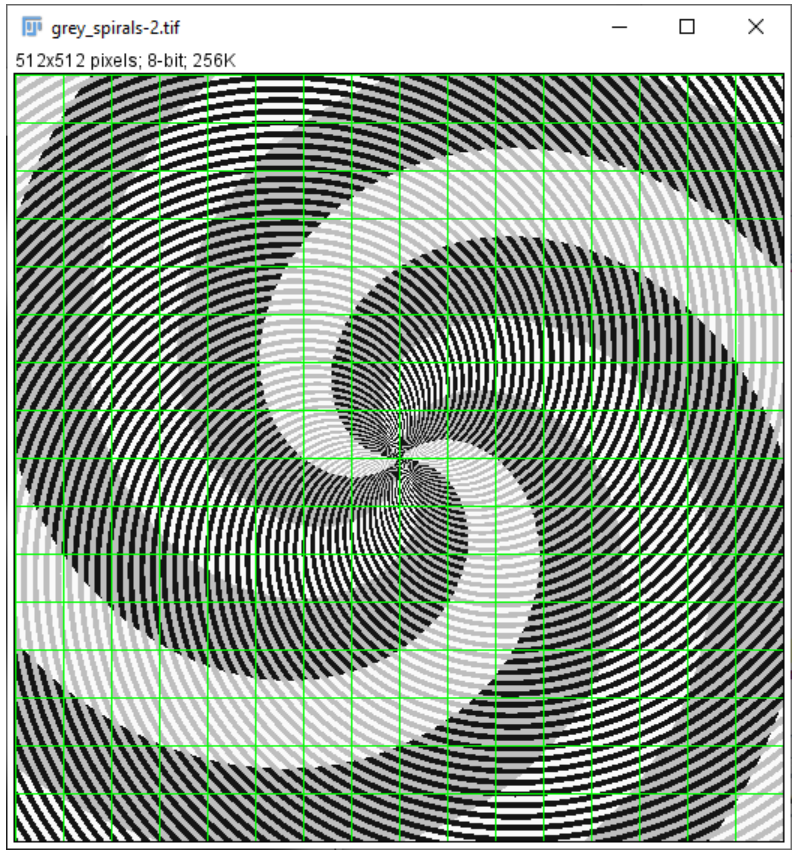


Margin: 10 pixels

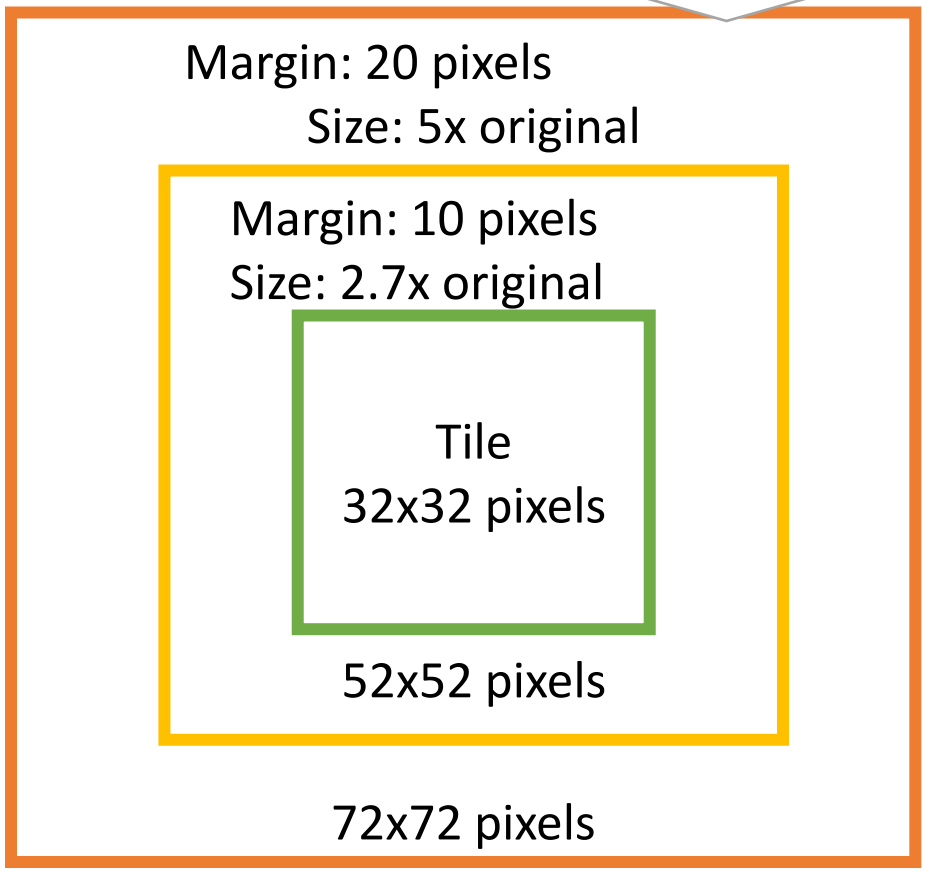
Margin: 20 pixels

Tiling

- Example: Gaussian blur (sigma = 20 pixels)
- Solution: Process with overlapping tiles (size + margin)



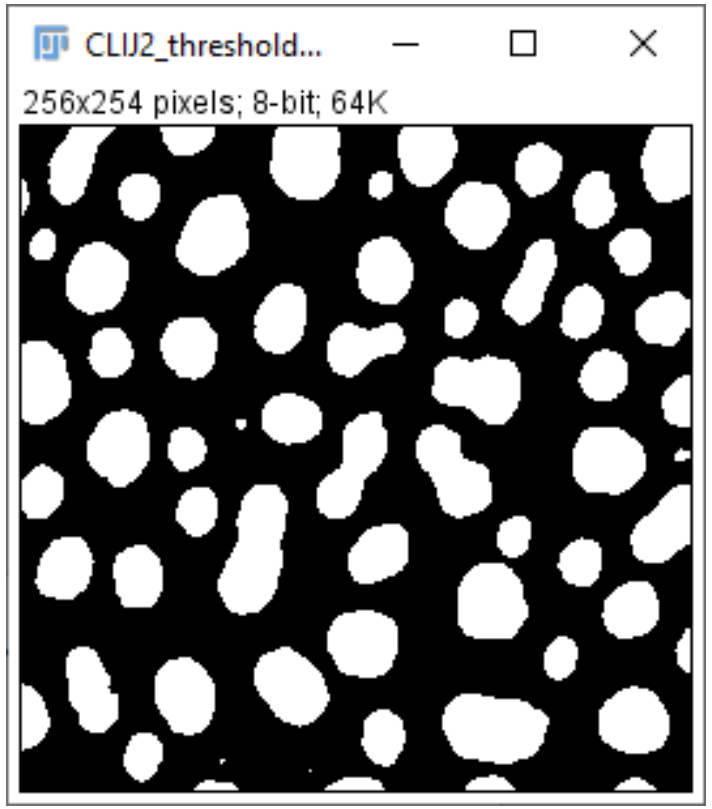
Computation time depends on tile size and margin width

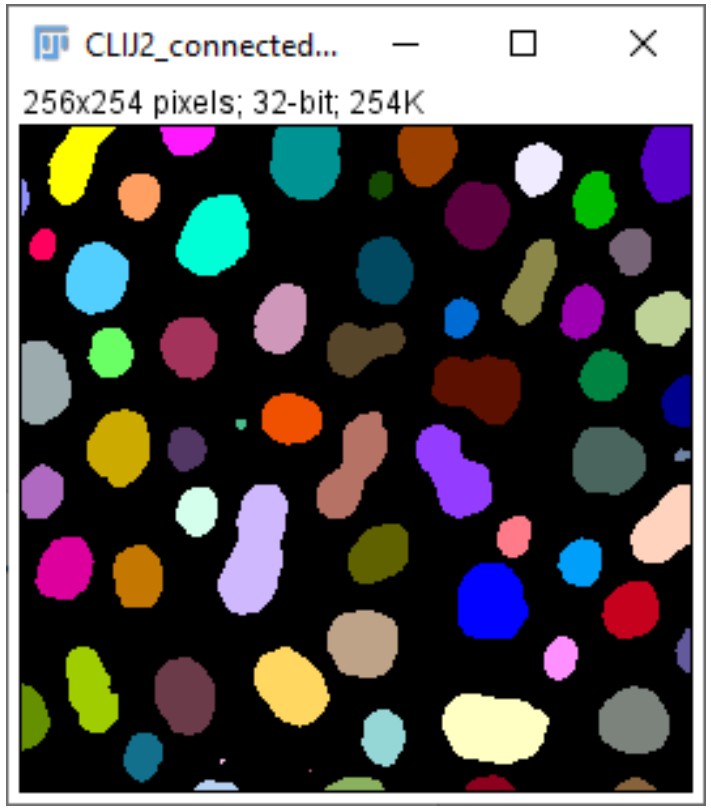


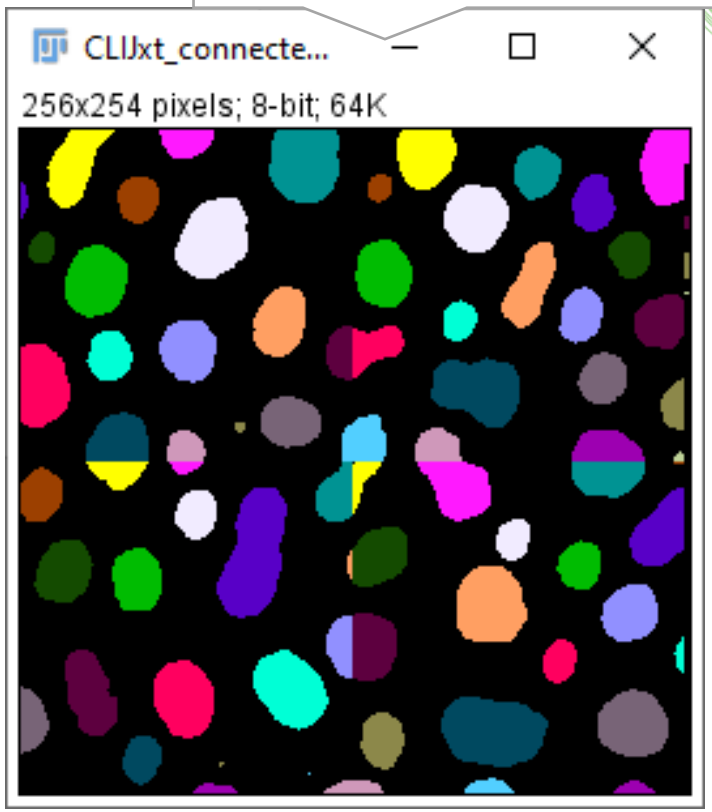
Tiling

- Some algorithms are hard to solve by processing tiles
- Example: Connected component analysis

Checking which labels touch and combine them is feasible.







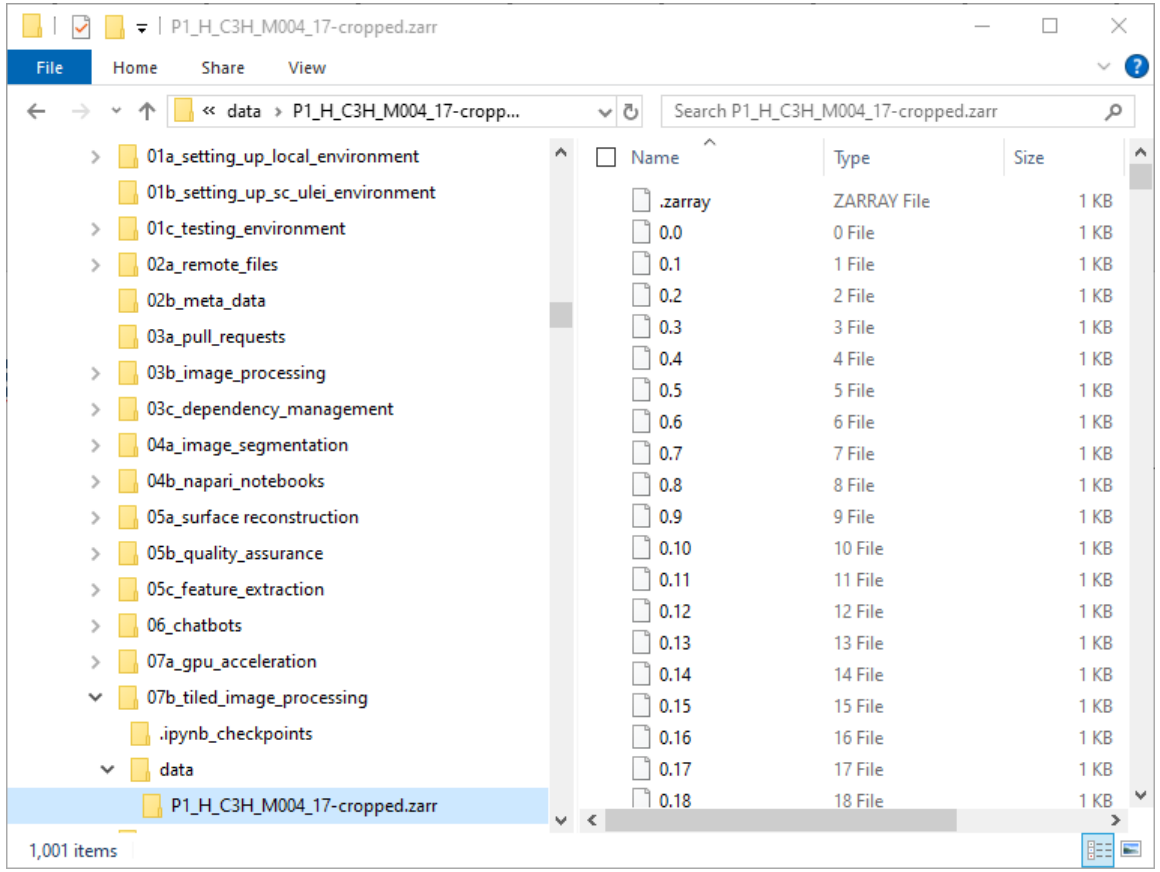
Tiling

- Some algorithms are hard to solve by processing tiles
- Example: Connected component analysis

There are algorithms for that, but hardly available tools.

Tiled image processing in Python

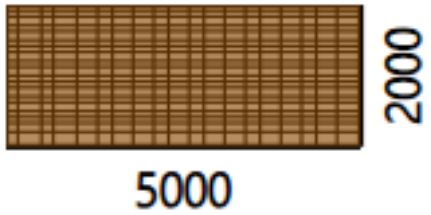
- Key: tiled file formats, for parallel, distributed, lazy loading



After executing this, no pixel has been read yet.

```
zarr_image = da.from_zarr(zarr_filename)
zarr_image
```

	Array	Chunk
Bytes	9.54 MiB	9.77 kiB
Shape	(2000, 5000)	(100, 100)
Dask graph	1000 chunks in 2 graph layers	
Data type	uint8 numpy.ndarray	



Tiled image processing in Python

- Delayed processing

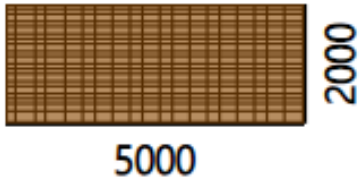
After executing this, no pixel has been read or processed yet.

```
[5]: tile_map = da.map_blocks(count_nuclei, zarr_image)

     tile_map
```

```
Processing image of size (0, 0)
(1, 1)
Processing image of size (1, 1)
(1, 1)
```

[5]:

	Array	Chunk
Bytes	76.29 MiB	78.12 kiB
Shape	(2000, 5000)	(100, 100)
Dask graph	1000 chunks in 3 graph layers	
Data type	float64 numpy.ndarray	



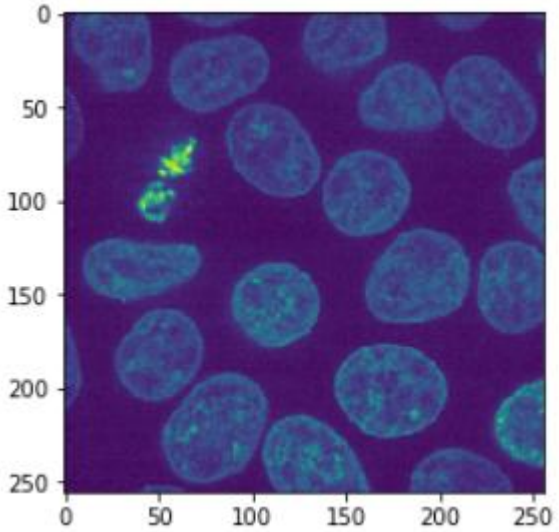
After that, results are available

```
result = tile_map.compute()
```

```
Processing image of size (100, 100)
Processing image of size (100, 100)
Processing image of size (100, 100)
Processing image of size (100, 100)
Processing image of size (100, 100)
Processing image of size (100, 100)
Processing image of size (100, 100)
```

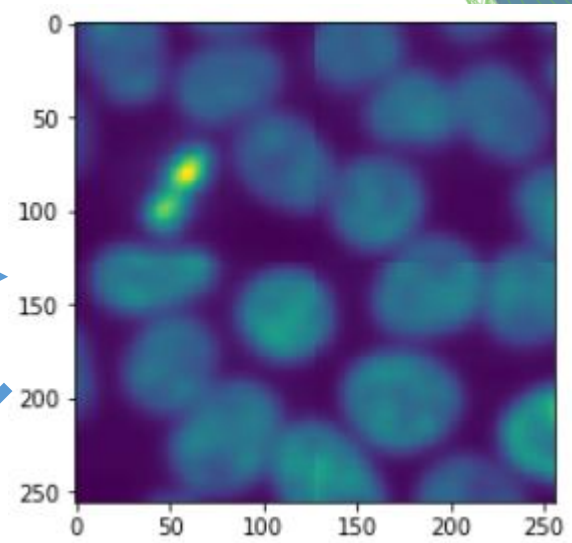
Tiling with/out overlap

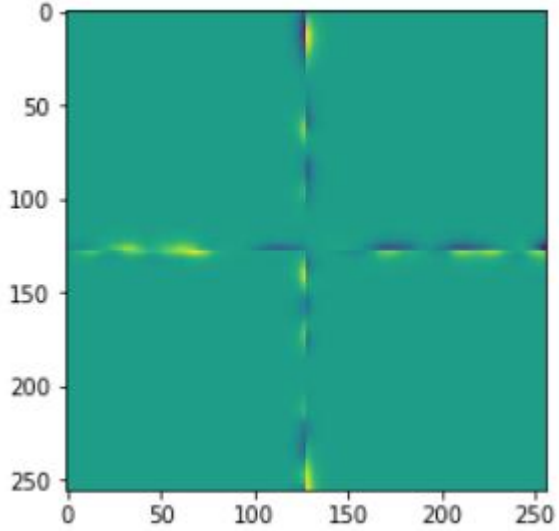
- Processing of images in tiles: artifacts at tile borders




```
tile_map = da.map_blocks(procedure, tiles)
```

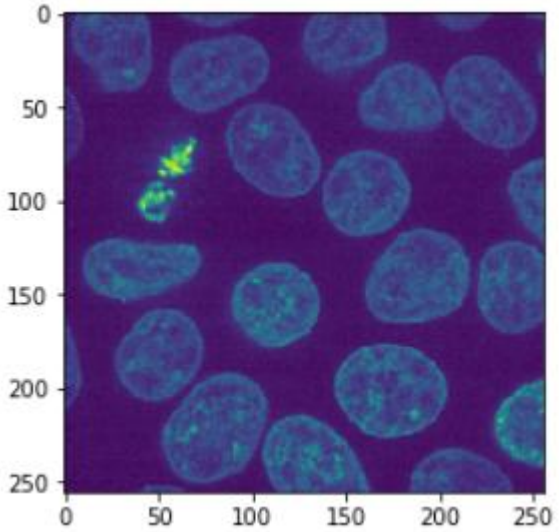
```
result = tile_map.compute()
```





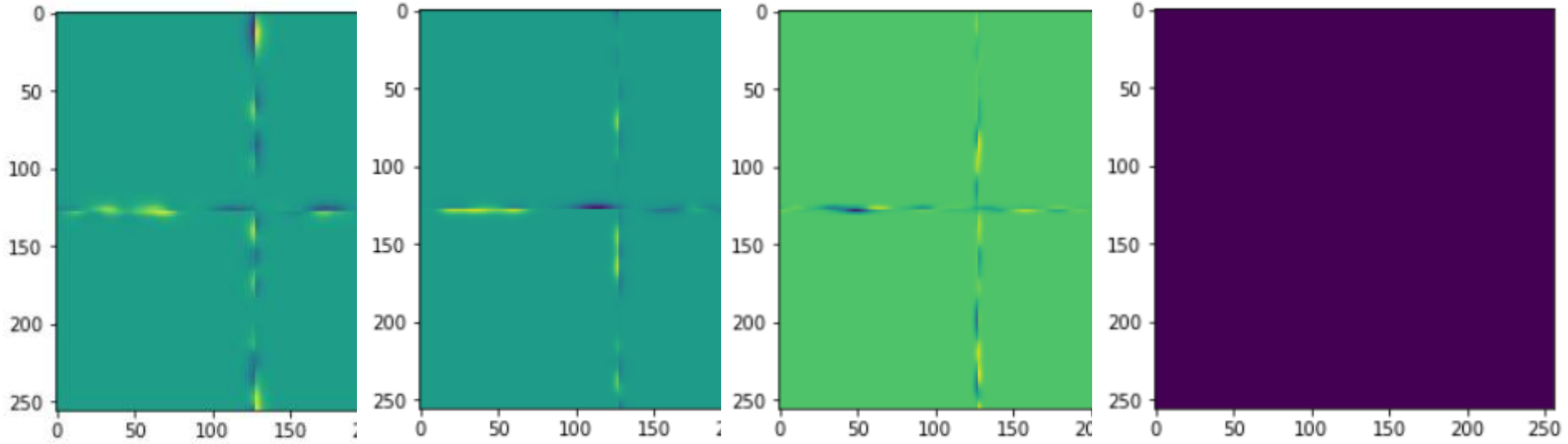
Tiling with/out overlap

- Processing of images in tiles: artifacts at tile borders



```
overlap_width = 1
tile_map = da.map_overlap(procedure, tiles, depth=overlap_width)
```

```
result = tile_map.compute()
```



```
sum difference 1.5288188631 sum difference 2.0981679908 sum difference -0.0057617038 sum difference 0.0
```

ScaDS.AI
DRESDEN LEIPZIG
CENTER FOR SCALABLE DATA ANALYTICS
AND ARTIFICIAL INTELLIGENCE

Exercises

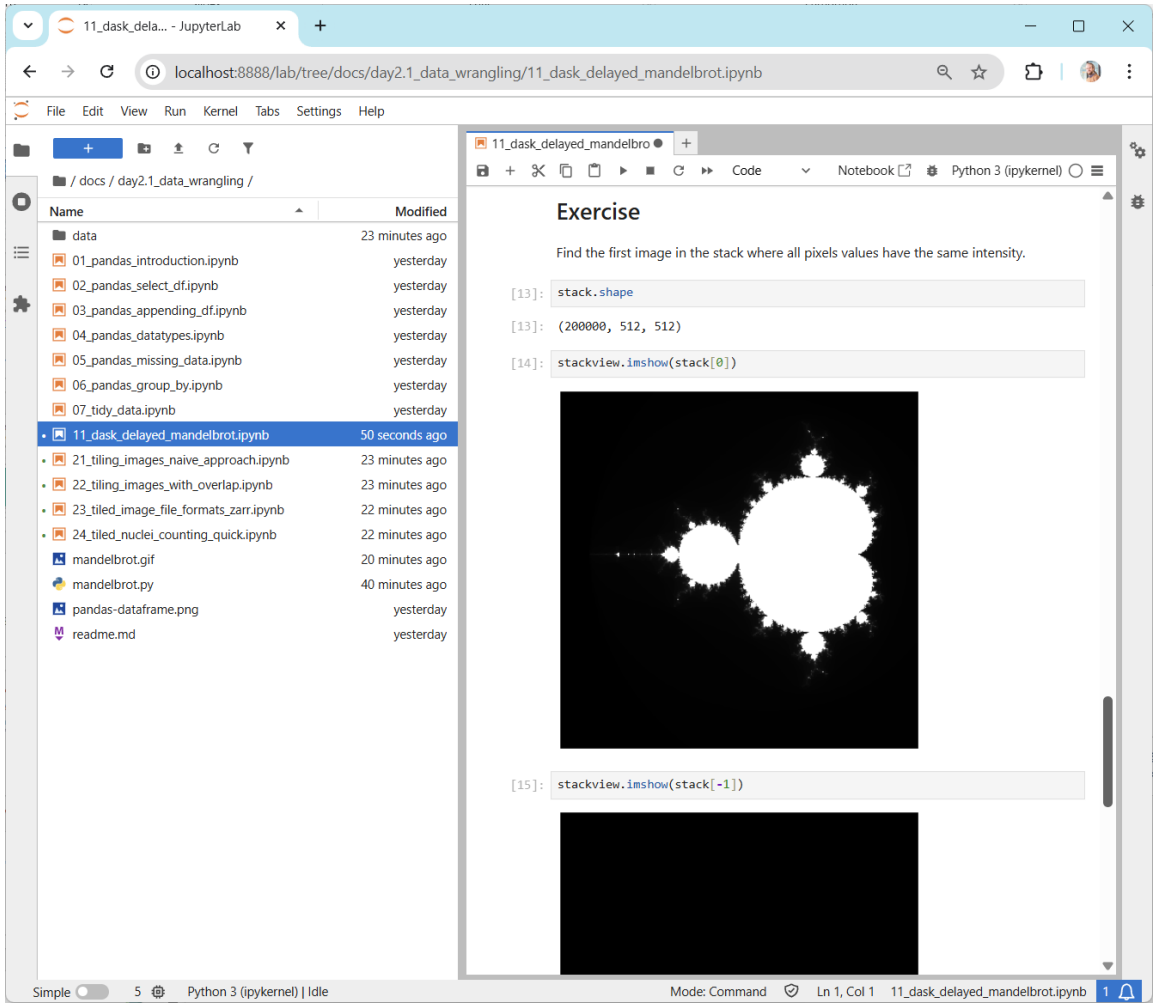
Robert Haase

GEFÖRDERT VOM
Bundesministerium
für Bildung
und Forschung

SACHSEN
Diese Maßnahme wird gefördert durch die Bundesregierung aufgrund eines Beschlusses des Deutschen Bundestages.
Diese Maßnahme wird mitfinanziert durch Steuermittel auf der Grundlage des von den Abgeordneten des Sächsischen Landtags beschlossenen Haushaltes.

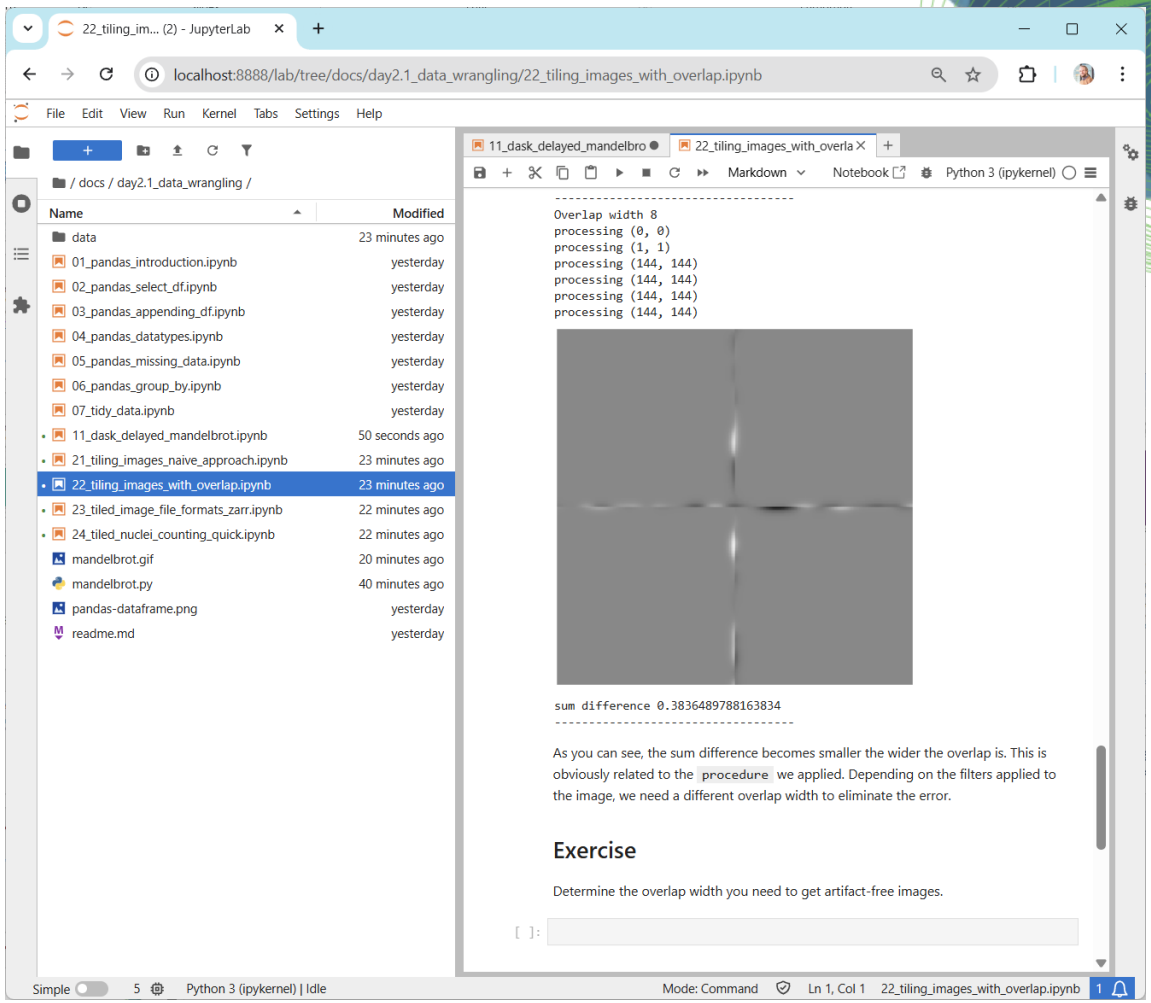
Exercise: Delayed processing

- Determine the first image index when all pixels have the same intensity (approximately).



Exercise: Tiled image processing

- Apply background-removal to an image in tiles. Determine the overlap width that‘s necessary to have artifact-free results.



Exercise: Counting Nuclei in tiles

- After segmenting and counting nuclei in tiles, compare the result to processing the whole image. Try out strategies for reducing the error.