File Formats in Microscopy

(for SW Developers)

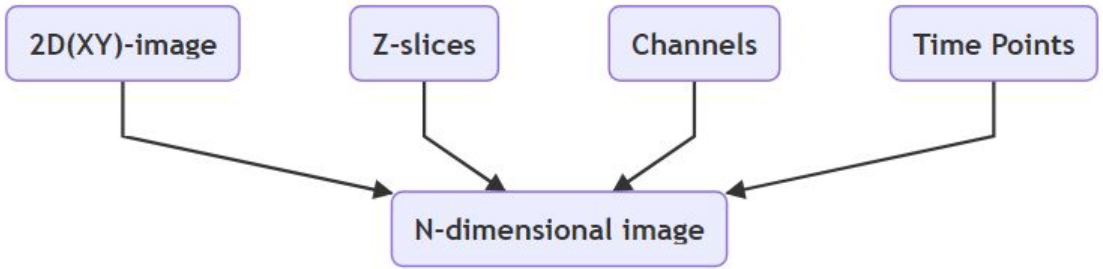
Dr. Sebastian Rhode
Software Architect AI Solutions
Product Center for Software

N-dimensional images

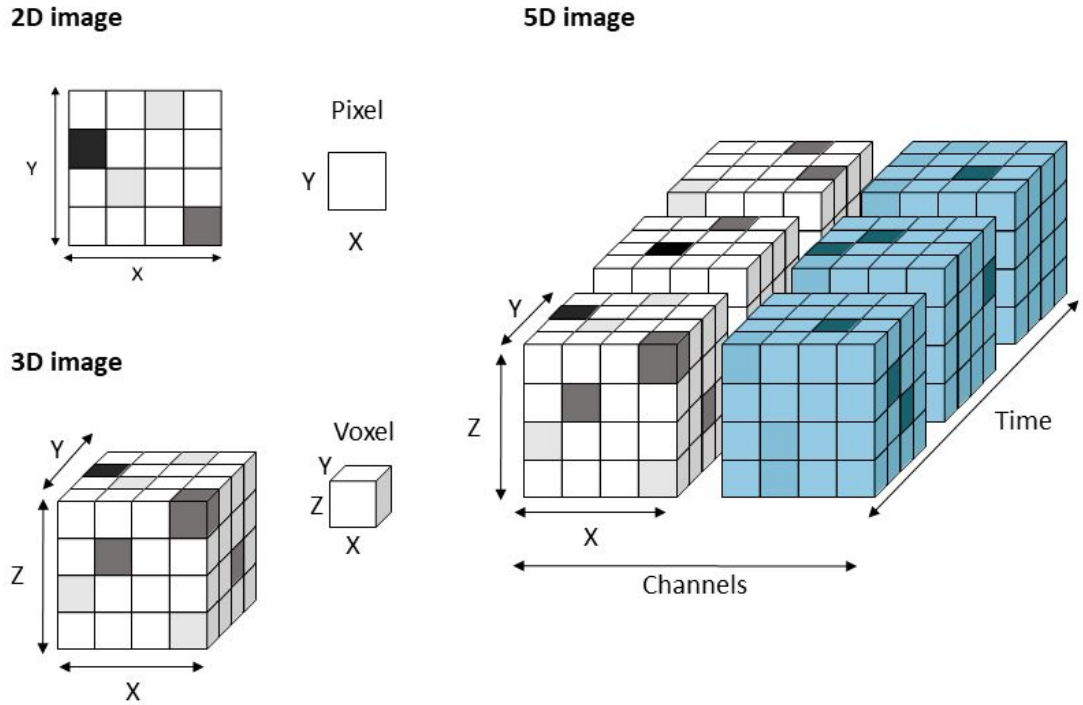
Apart from the X and Y dimensions, visible in the width and height of an image, image data can have additional dimensions.

- Z dimension, providing depth to image data
- channels, showing data recorded with different detectors or detector settings
- time dimensions
- Well plate layout

Viewing multi-dimensional data can be challenging, because a computer monitor can only render a (multi-color) 2D representation.



N-dimensional images

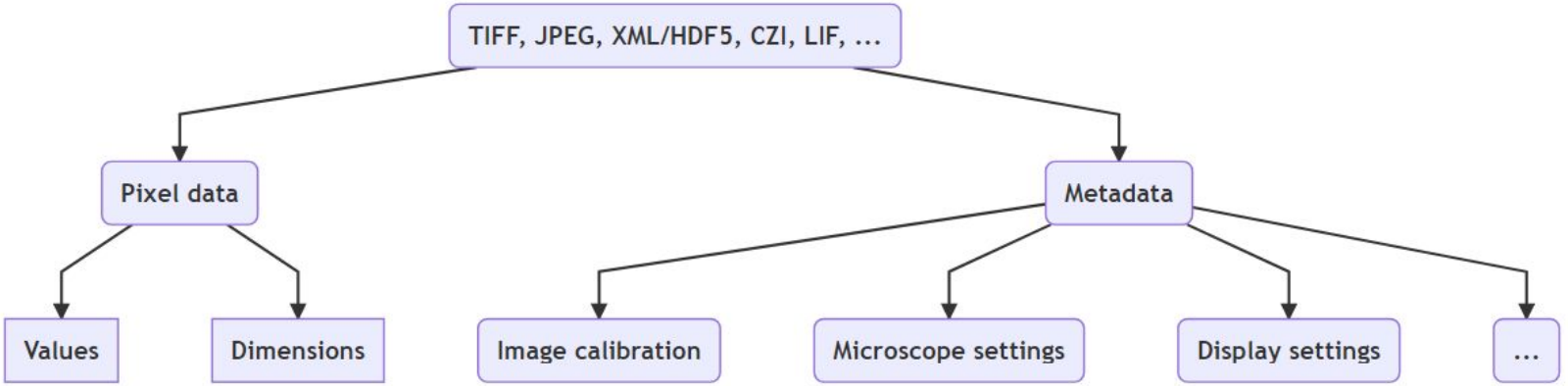


Schematic representation of 2D, 3D, and 5D image data. 2D images are made up of tiny squares called pixels, whereas 3D images are made up of cubes called voxels.

File Formats - Remarks

In general there are numerous ways how to save image data

- Virtually every microscope vendor has their own file format. It is thus very important to understand how to open those files and inspect their content.
- Some software will open only specific image file formats and thus it is sometime necessary to re-save the data.
- During such image file format conversions information can be lost; it is important to be aware of this and avoid such information loss as much as possible.



File Formats – General remarks

- Image files consist of **pixel values** and **metadata**
- Some file formats are suitable for **data to analyze**, others for **data only to display**
- Metadata essential for analysis can be lost by saving in a **non-scientific file format**
- **Compression** can be either **lossless** or **lossy** – with different results
- Original pixel values can be lost through **lossy compression**, **conversion to RGB**, or by **removing dimensions**

File Formats – Working with image formats

- Always keep your original image files, in their original format
- During analysis:
  - Use software designed for scientific data (i.e. not a general photo editor)
  - If you need to save images, use the default file format for the software (e.g. TIFF for ImageJ)
  - **Check everything!** Make sure you can reopen the image after saving, and any pixel values, pixel size information and dimensions are retained exactly.
- Use general-purpose file formats (e.g. PNG, JPEG) for figures, presentations or websites – but not later analysis
- Use TIFF cautiously, because it can be used both as a scientific format suitable for analysis and a general-purpose format suitable only for display
  - OME-TIFF is often a better alternative to TIFF, since it standardizes how the metadata is stored

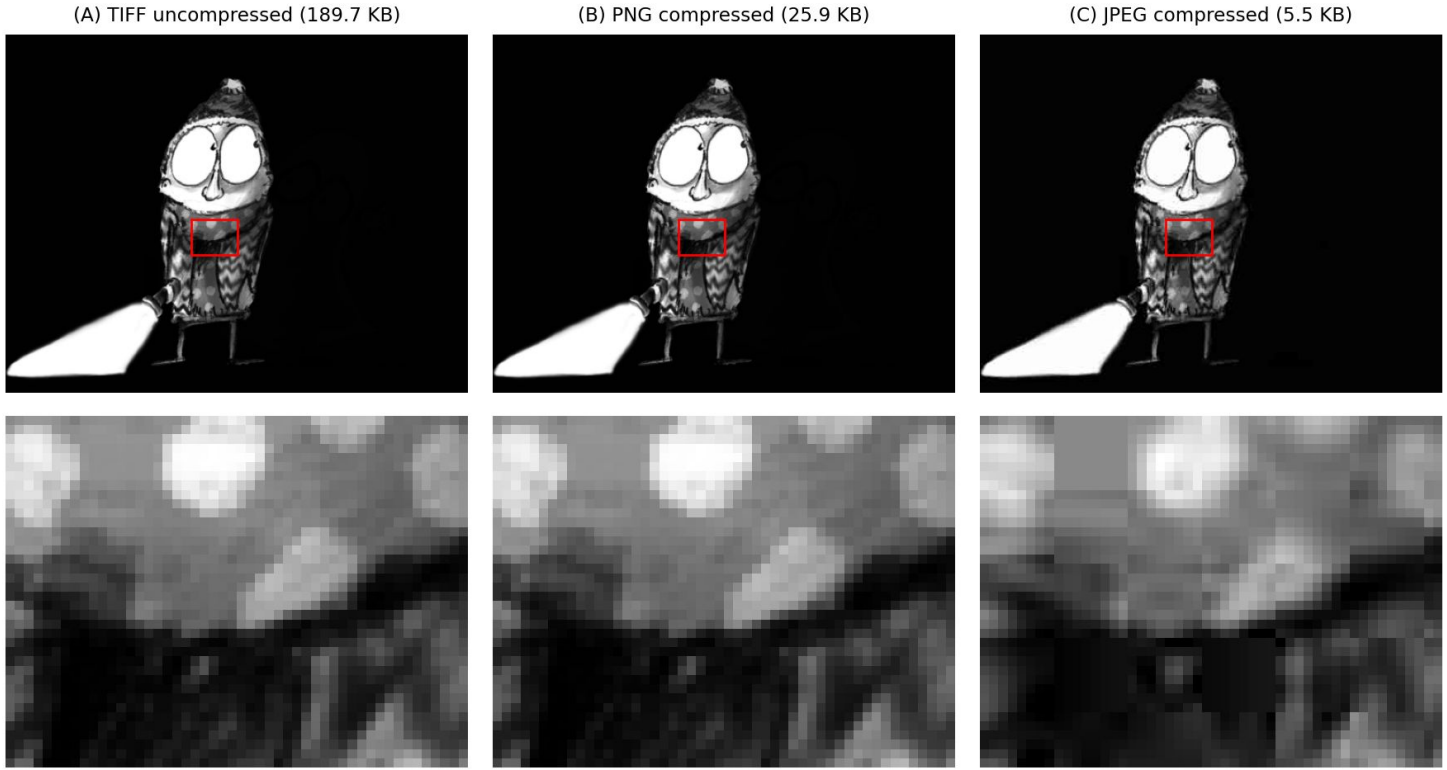
File Formats – Common formats in Microscopy Image Analysis

Format	Extensions	Main Use	Compression	Comment
TIFF	.tif, .tiff	Analysis, display (print)	None, lossless, lossy	Very general image format
OME-TIFF	.ome.tif, .ome.tiff	Analysis, Display (print)	None, lossless, lossy	TIFF, with standardized metadata for microscopy
OME-NGFF	ome.zarr	Analysis, Display	None, lossless, lossy	Emerging format, great for big datasets
PNG	.png	Display (web, print)	Lossless	Small(ish) file sizes without compression artefacts
JPEG	.jpg, .jpeg	Display (web)	Lossy (usually)	Small file sizes, but visible artefacts

File Formats – Pixel values and Compression

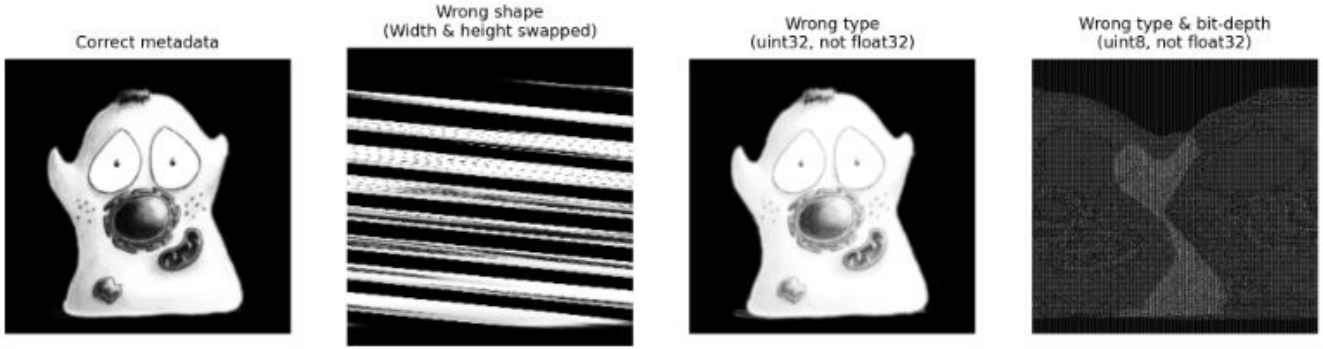
- **No compression**. The bytes representing the pixels are stored directly in the file. This means that the pixel values are preserved exactly, but the file can be quite large.
    - *Examples: TIFF (Uncompressed), ICS/IDS, CZI (uncompressed)*
- **Lossless compression**. The bytes representing the pixels are stored using a compression algorithm that (usually) results in less storage space being required, while making it possible to reconstruct the original values exactly by decompression. Compared to uncompressed data, the file size using lossless compression is generally smaller but reading or writing the file takes longer.
    - *Examples: TIFF (LZW compressed), PNG, BMP, JPEG2000 (lossless), ZSTD*
- **Lossy compression**. The bytes representing the pixels are stored using a compression algorithm that does *not* have to be able to reconstruct the original values exactly. This can result in dramatically smaller file sizes, but at a loss of information in the image – and often visual artefacts.
    - *Examples: TIFF (JPEG compressed), JPEG, JPEG-XR, GIF, JPEG2000 (lossy)*

File Formats – Lossy compression is BAD for image Analysis



File Formats – Core Metadata

- Pixel values are represented as a stream of bits: ones and zeros.
- A few core pieces of metadata *must* be stored alongside these bits to interpret them as an image, such as the dimensions, bit-depth and type.
- If this information is missing – or wrong – then the image usually either cannot be read, or looks strange in some way



File Formats – Additional Metadata

- The most important piece of ‘non-core’ metadata is the pixel size. Unlike core metadata, it is *not* necessary for a file format to preserve the pixel size for an image to be opened and appear ‘correct’.

- This is a problem because if the pixel size is incorrect or missing, it remains possible to make length, area or volume measurements within an image – but these measurements are likely to be wrong.

- **In microscopy, the pixel size is typically represented as a value in µm/pixel. Saving in some file formats (e.g. JPEG and PNG) tends to lose pixel size information.**

- **Others (e.g. TIFF) *might* preserve the pixel size, *might* lose it, or *might* convert it to something else entirely.**

File Formats – Image Acquisition & Image Analysis

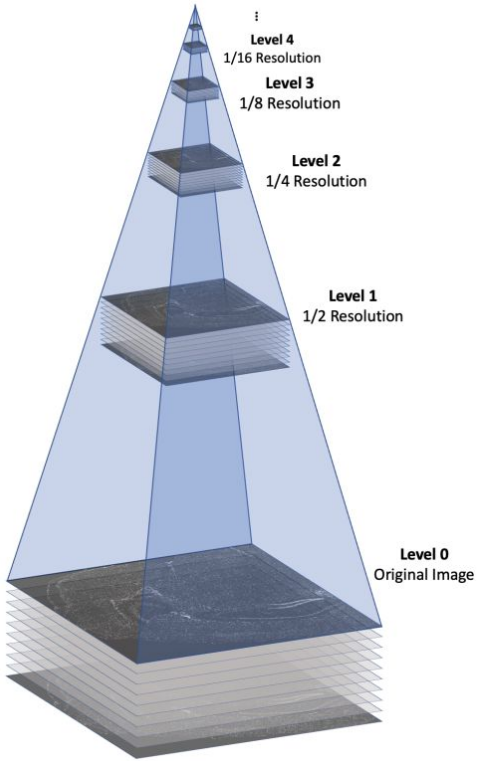
The general rule is: **Use the default file format for your microscope and acquisition software**

- The rationale is that the manufacturer's format ought to preserve all the information and metadata.
- Often, the acquisition software supports saving the data in some other way – but that can be risky. There's a strong chance that at least some metadata could be lost.
- Sometimes, the alternative export might do even worse things – such as convert the data to RGB.
- If in doubt export images as OME-TIFF for further analysis, but not as TIFF, PNG, JPEG or RGB images

Usually there is no need to export CZI in a different format because mots Image Analysis & processing tools can read them directly!

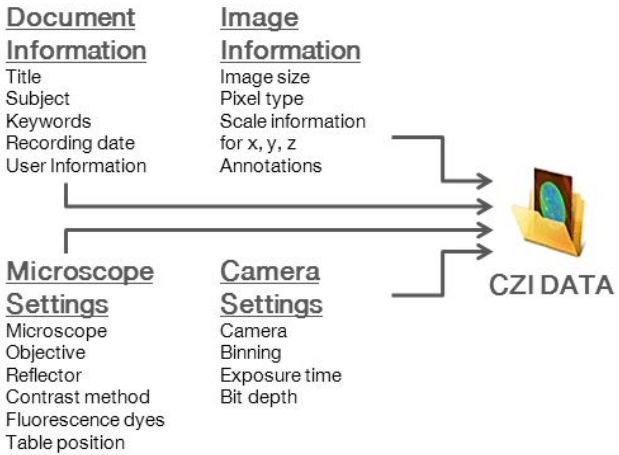
File Formats – Pyramidal Images

- Modern bioimages can be *huge*. It's not uncommon for a single image to contain hundreds of gigabytes of pixel data, making it infeasible to open the full image in one go.
- However, for visualization or analysis we often don't *need* all the pixels at once.
- For example, when viewing an image we only really need to access the pixels that are visible on screen at any moment – at the magnification at which they are being viewed.
- Pyramidal image files help overcome this by using two tricks:
    - They store the image as separate chunks (often called 'tiles', if they are 2D)
    - They store the same image at different resolutions in the same file



- https://www.10xgenomics.com/resources/analysis-guides/performing-3d-nucleus-segmentation-with-cellpose-and-generating-a-feature-cell-matrix

File Formats – Carl Zeiss Image (CZI)



Carl **Z**eiss **I**mage Format:
https://www.zeiss.com/microscopy/en/products/software/zeiss-zen/czi-image-file-format.html

In addition to the image, the following metadata (and more) stored within the file:

- acquisition date, keywords, user comments
- microscope type and settings
- experiment settings
- dimensions types, scaling, pixel types
- used contrasts, objectives, filters
- stage (x,y) and focus (z) positions
- detector parameters such as exposure values, binning, bit depth, pixel dwell time
- automatic or user-defined graphic elements, annotations and display curves
- sample barcode information and label images
- analysis pipelines, results and objects, label images
- audit trail

File Formats – CZI, ZISRAW

- High Level Description: CZI Image Format - High Level Description - Overview (azure.com)
- CZI is the format for image storge
- The fundamental data storage schema is called “**Z**eiss **I**ntegrated **S**oftware **RAW**” data format
- Currently, CZI is the only ZISRAW based implementation
- For all the details please refer to: DS_ZISRAW FileFormat - Overview (azure.com)

File Formats – CZI APIs and Readers

For the CZI exist three main APIs allowing to read and write CZI images

- ZeissImgLib (.NET): to be published as Nuget package soon
- libCZI (C++): ZEISS/libczi
- pylibCZIrw: based on libCZI, available on PyPi as pylibCZIrw (PyPi) and internal repo: RMS_PyPI_pylibCZIrw - Repos (azure.com)
- https://pypi.org/project/aicspylibczi/ - python package from Allen Cell Institute based on libCZI

Existing Readers:

- BioFormats: https://github.com/ome/bioformats/blob/develop/components/formats-gpl/src/loci/formats/in/ZeissCZIReader.java
- Czifile: https://github.com/cgohlke/czifile

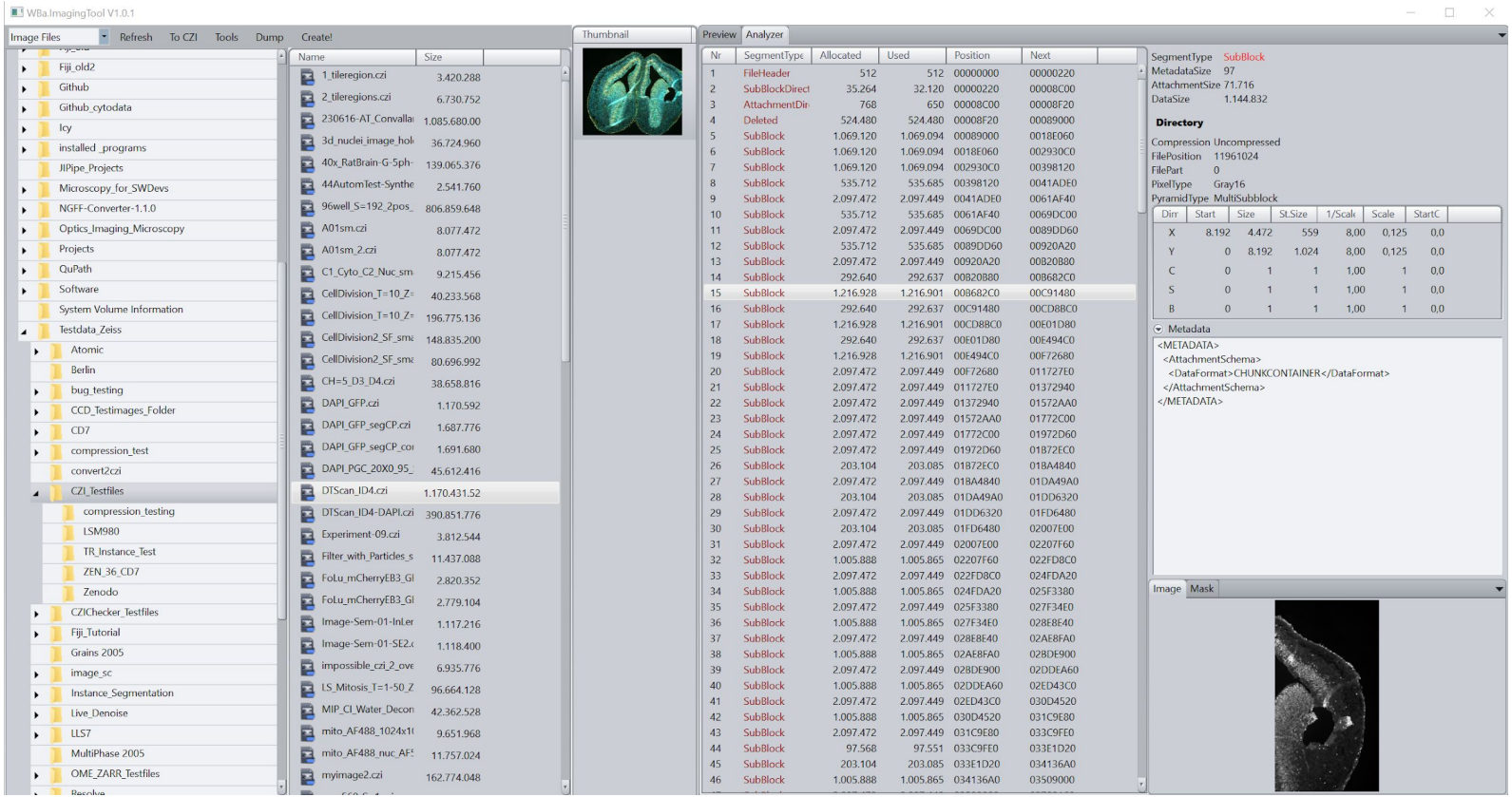
File Formats – CZI Tools

Around the the CZI exist other important tools useful for users and developers

- WBa.ImagingTool.exe (comes with binaries)
- CZICheck – Ceheck the CZI integrity
- CZICompress – CLI to compress CZI image data
- CZIShrink – UI for CZI Compress

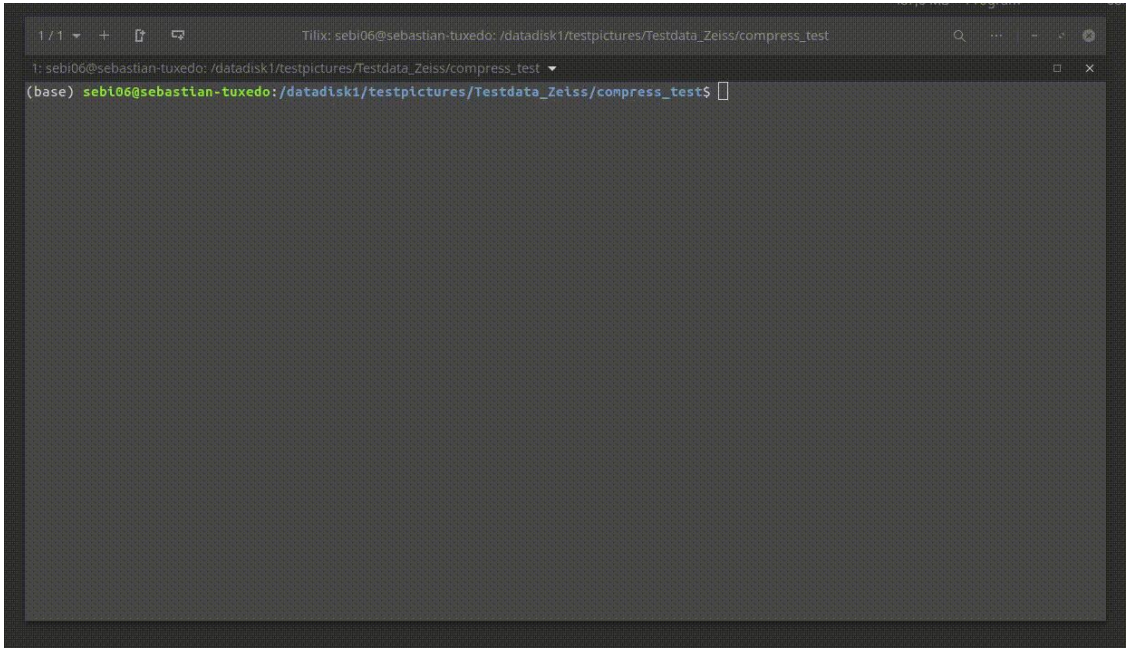
File Formats – WBA.ImagingTool

Inspect subblocks and much more



File Formats – CZICompress

Compress CZI Images Files from the commandline

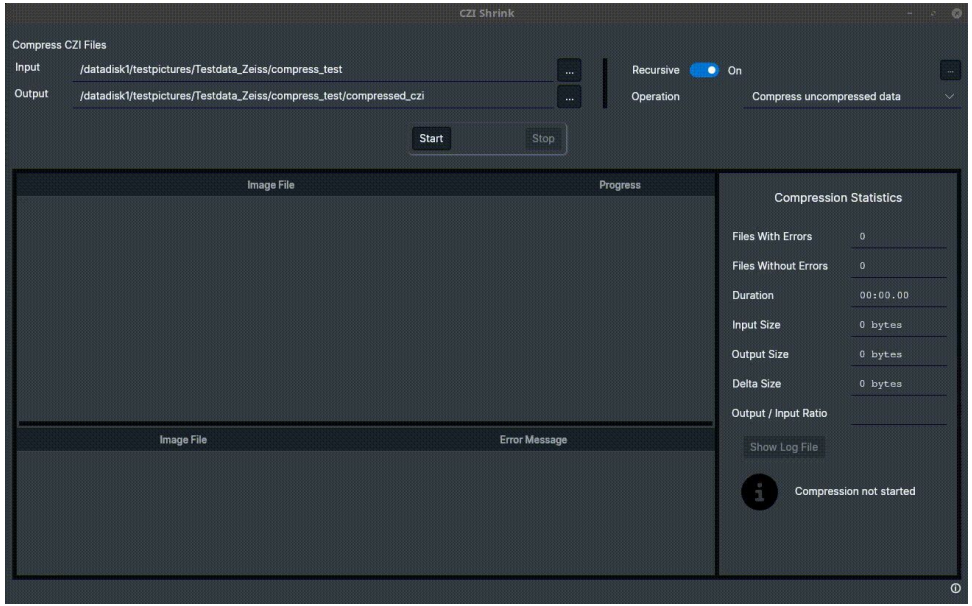


https://github.com/ZEISS/czicompress

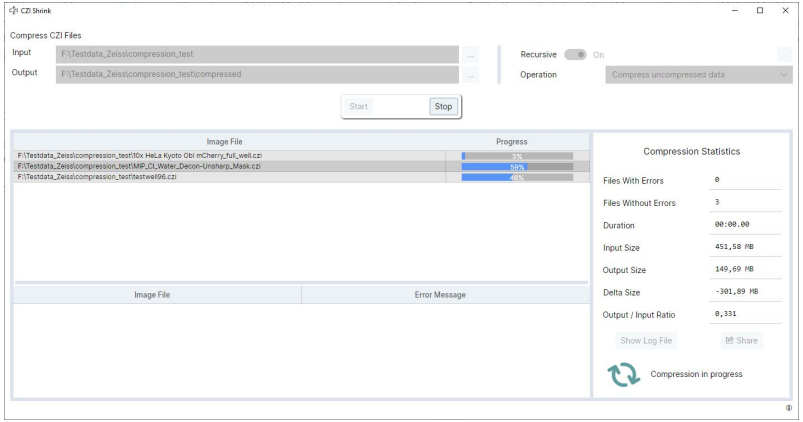
- compress or decompress a single CZI file
- versatile
- scriptable
- run in headless/server environments
- run in cron jobs
- cross-platform (focus on linux-x64 and win-x64)

File Formats – CZIShrink

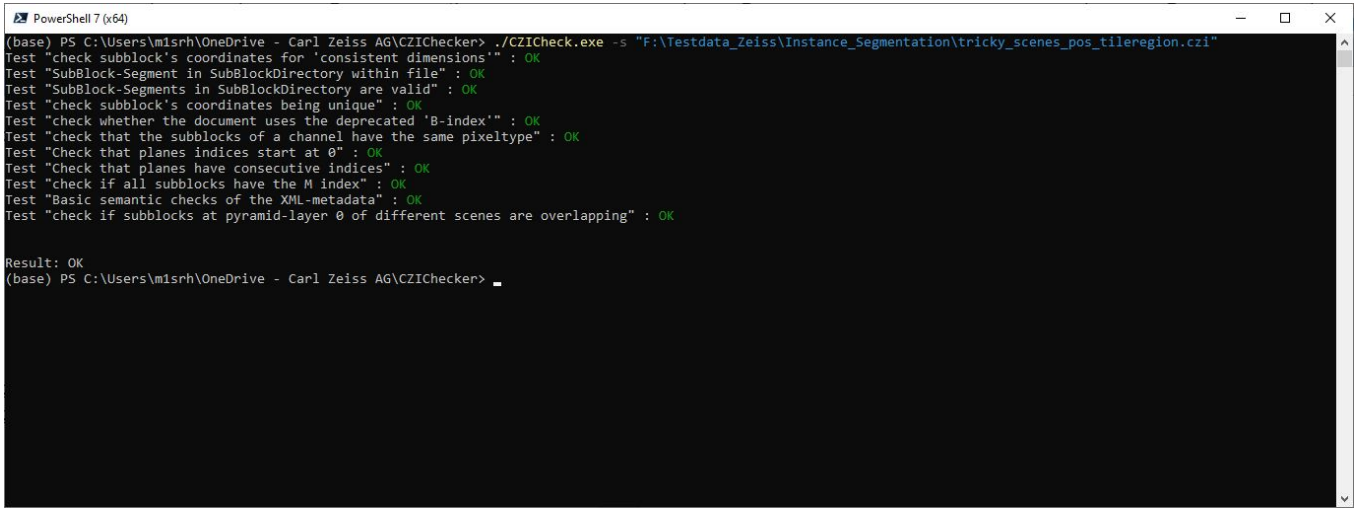
Compress CZI Images Files from a cross-platform UI (not released yet)



- Cross Platform GUI App
- Developed, tested and released on Win-x64 and Linux-x64
- Designed to work with large CZI collections
- Multi-threaded processing
- Strictly non-destructive



File Formats – CZI Tools
Check CZI for internal errors



https://github.com/ZEISS/czicheck

- Check CZI for internal errors
- For Devs: Use it to make sure that newly created CZI (inside project XYZ) pass the check !
- For testers: Use it to verify that the CZI used testing is not causing the issue one tries to track down

File Formats – CZI Dimension Identifiers

Identifier or Letter	Name of dimension	Explanation & Remarks
X	Width	width of the image in X dimension in [pixel]
Y	Height	height of the image in Y dimension in [pixel]
C	Channel	number of channels
Z	Slice	number of z-planes
T	Time	number of time points
M	Mosaic	index of tile for compositing a scene
S	Scene	contiguous regions of interest in a mosaic image
R	Rotation	
I	Illumination	SPIM direction for LightSheet
B	Block	acquisition block
H	Phase	e.g. Airy detector fibers
V	View	e.g. for SPIM

File Formats – CZI TileRegions, Positions and Scenes

Tile Region

- consists out of many individual tiles (which may overlap)
- can have arbitrary shape
- a TileRegion also is tagged as a **scene** (S-Index) and counts as one scene

Tiles

- individual tiles (M-index inside CZI) refers to individual acquisition tiles inside a bigger Tile Region
- overlap can be adjusted inside the experiment
- a tile itself can have multiple channels and slices

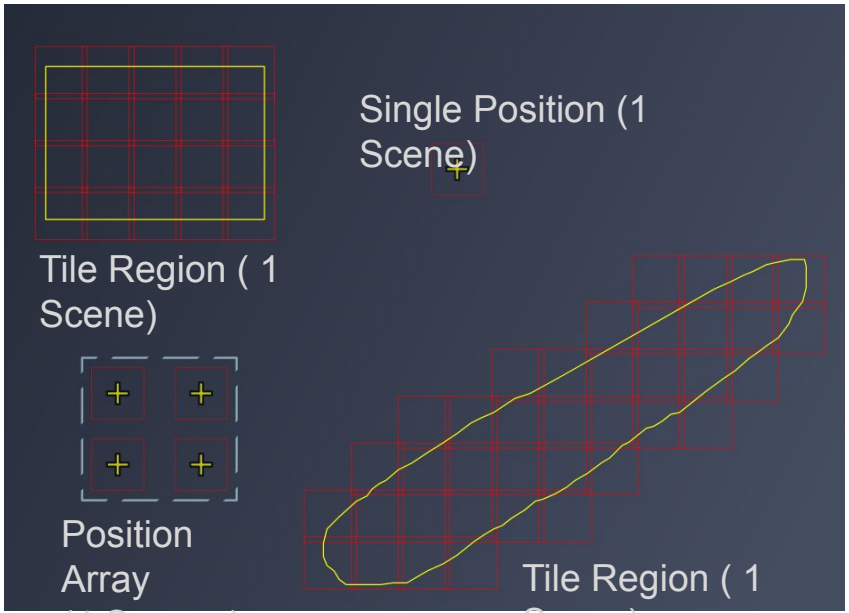
Positions

- single multi-dimensional stack acquired at a specific XYZ position
- a single positions counts as one scene
- position arrays are groups of positions

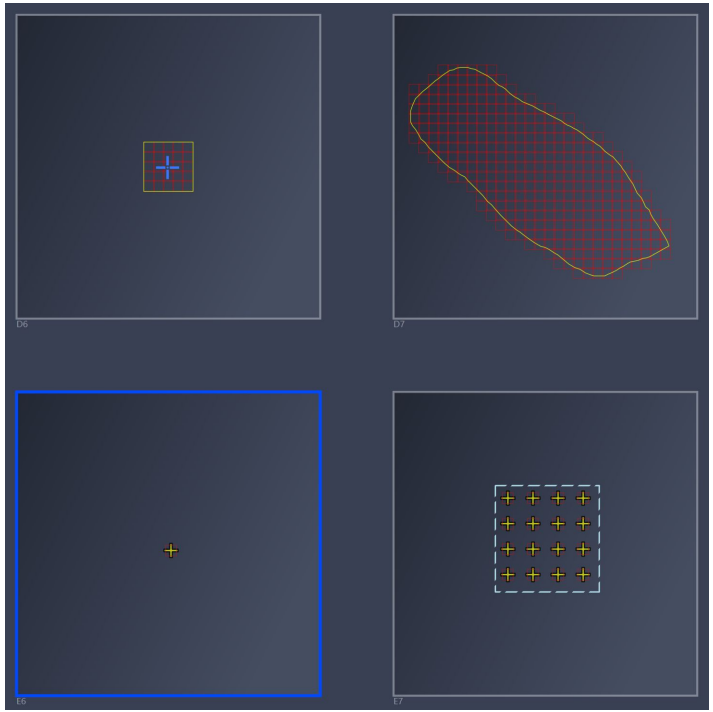
Scenes

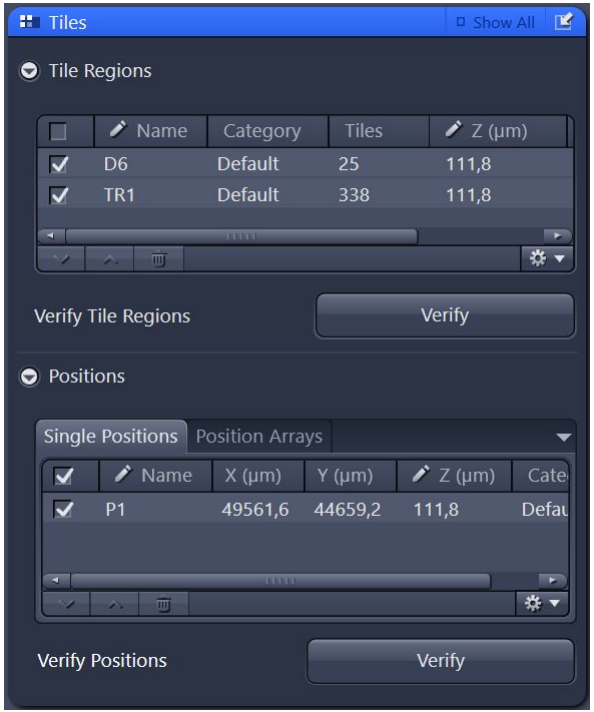
- tile regions or single positions (inside an array) get the tag scene
- scene tag can be used to "show" only specific scenes etc.

File Formats – CZI TileRegions, Positions and Scenes



File Formats – CZI TileRegions, Positions and Scenes

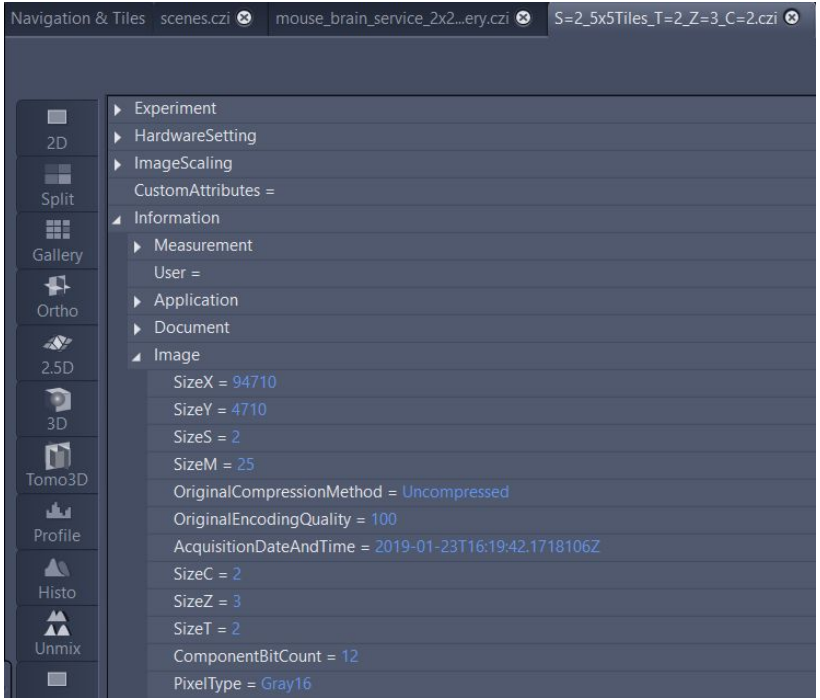




File Formats – CZI Image Bounds

Image Bounds
This term is often used to specify the total size of the dimensions of an CZI. In that context the term **bounding box** is often used. To illustrate the concept, consider the following CZI image:

- 2 Scenes □ **SizeS = 2**
- 2 Time Points □ **SizeT = 2**
- 25 (5x5) Tiles per Tile Region □ **SizeM = 25**
- 3 Slices □ **SizeZ = 3**
- 2 Channels □ **SizeC = 2**
- X (Width) total □ **SizeX = 94170**
- Y (Height) total □ **SizeY = 4170**

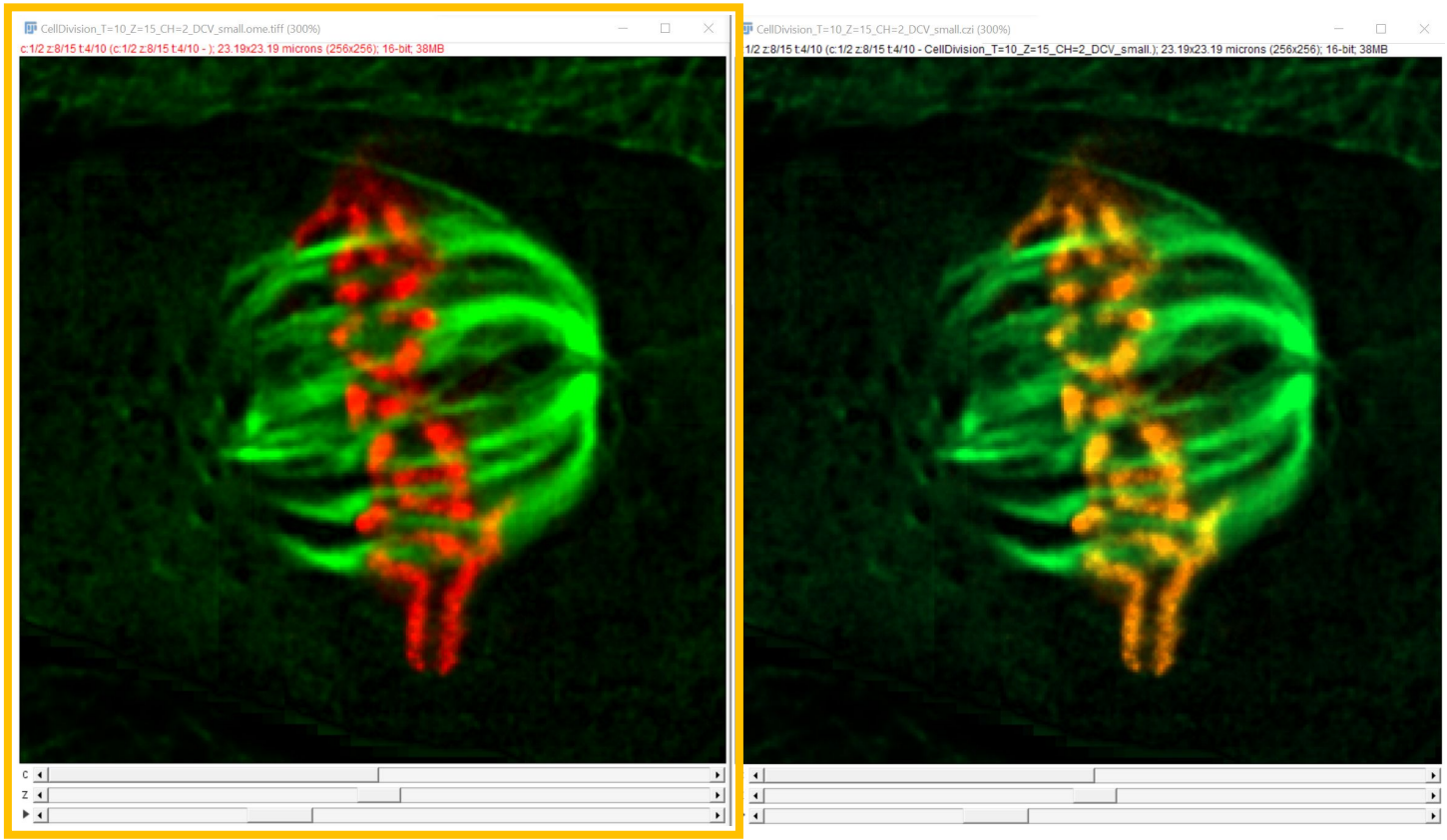


File Formats – CZI Image Bounds

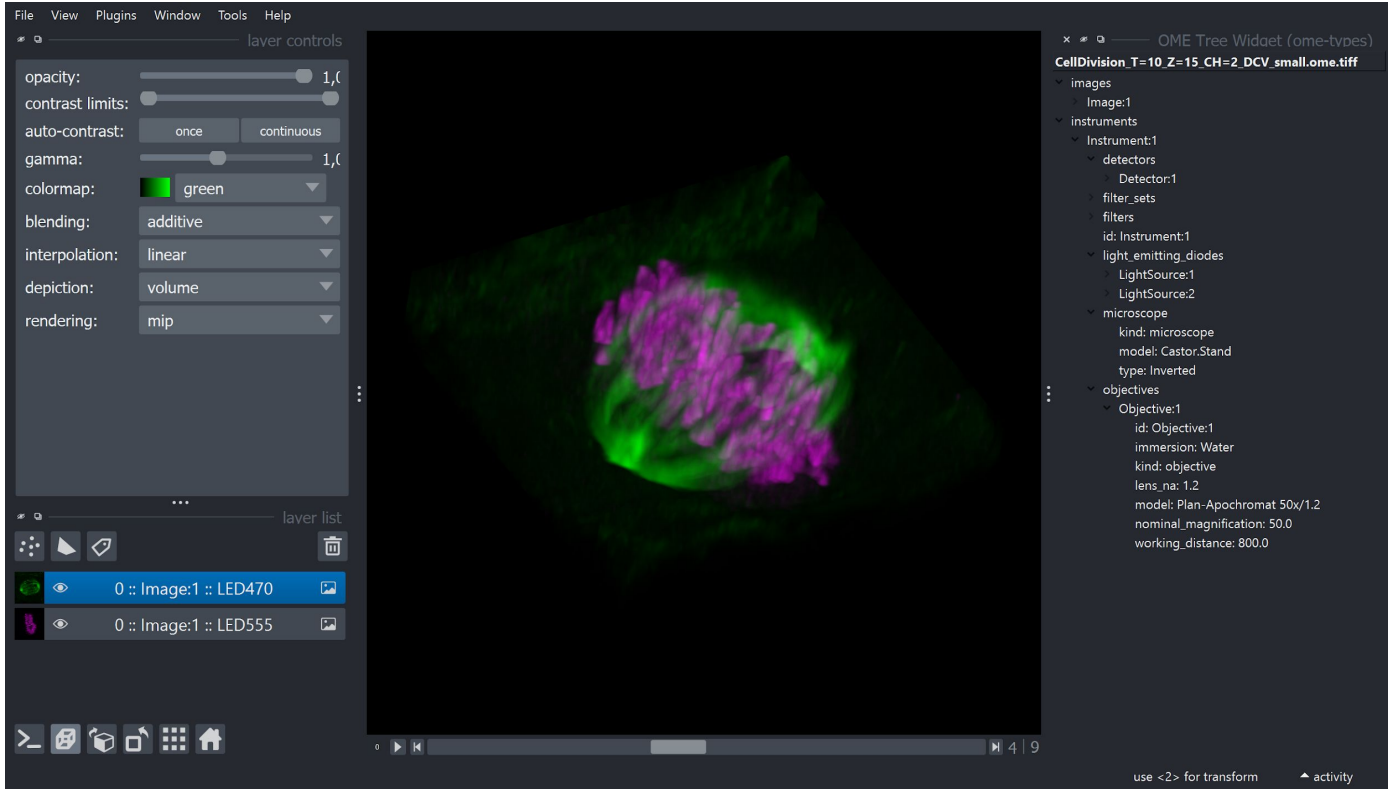
File Formats – OME-TIFF

- **O**pen **M**icroscopy **E**nvironment (OME):
  - https://www.openmicroscopy.org/
  - https://ome-model.readthedocs.io/en/stable/ome-tiff/specification.html#
  - https://github.com/ome/ome-model-documentation/blob/12e87cc5f2c3714a22b06af8163f4a60566dc87a/ome-tiff/specification.rst
- “def-facto” standard format for analyzing microscopy images (but will not really developed any further)
- is supported by many applications and languages (Java, Python, C++, MATLAB, R)
- already “quite old” and has important limitations
  - working with large, multi-dimensional datasets is tricky (or even impossible)
  - additional dimension apart form T-C-Z-Y-X have limited support
- main library to work with is BioFormats: https://www.openmicroscopy.org/bio-formats/
- **ZEN can easily read & write OME-TIFF**

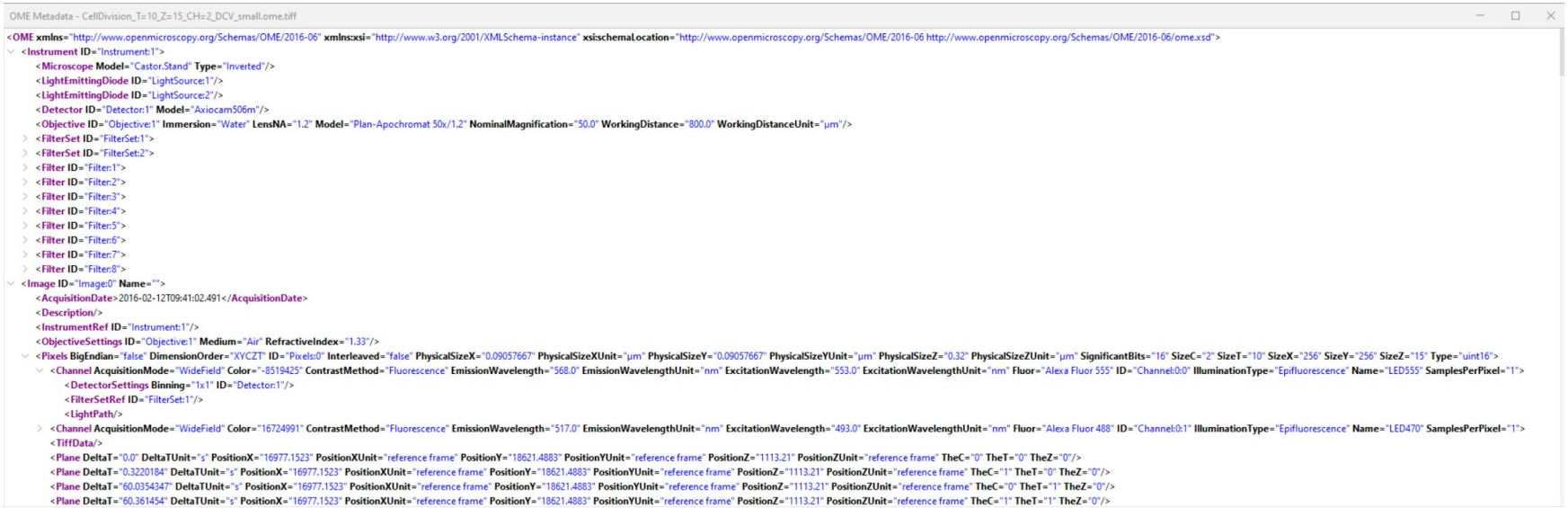
File Formats – OME-TIFF in Fiji (BioFormats)



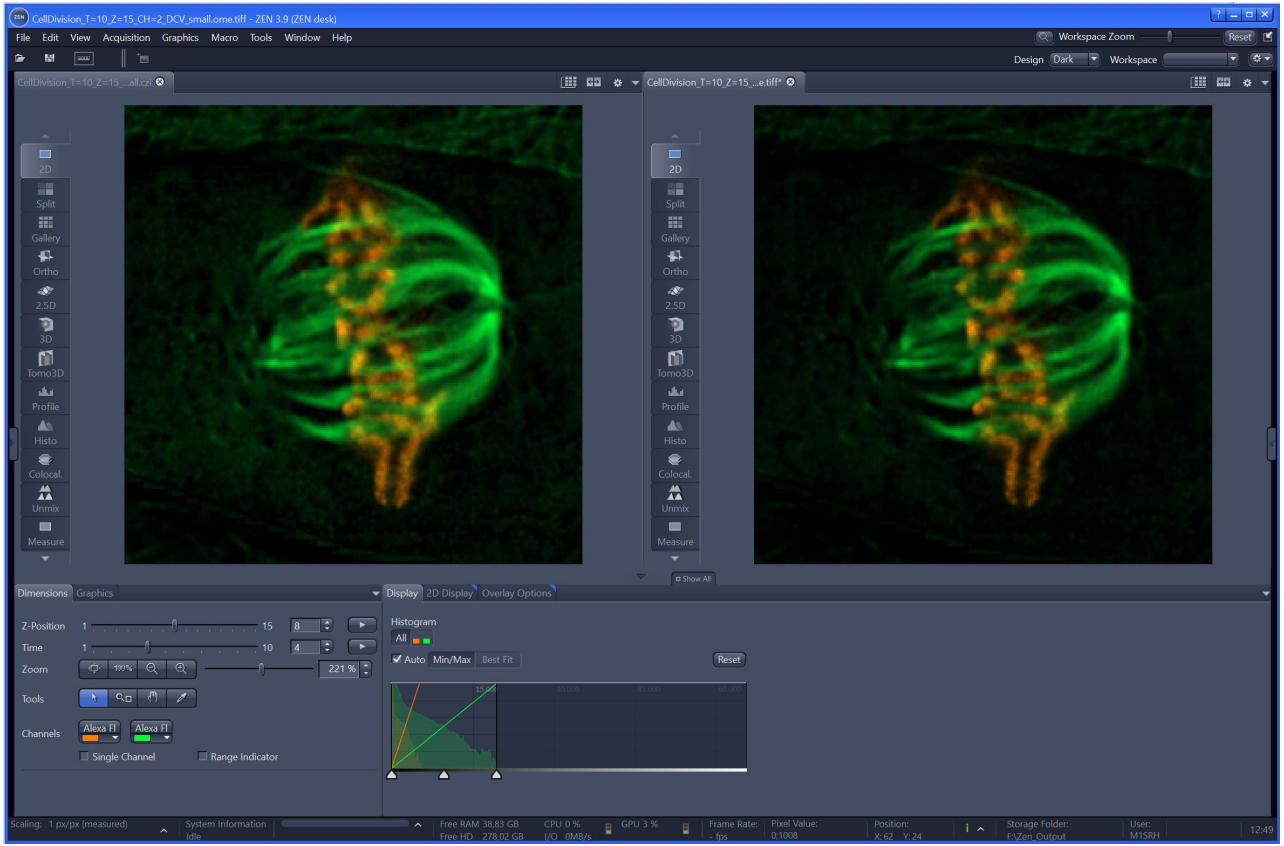
File Formats – OME-TIFF in Napari



File Formats – OME-TIFF in Fiji (BioFormats)



File Formats – OME-TIFF in ZEN



File Formats – OME-ZARR and OME-NGFF
What is ZARR?

Zarr is motivated by the need for a simple, transparent, open, and community-driven format that supports high-throughput distributed I/O on different storage systems.

- Zarr data can be stored in any storage system that can be represented as a key-value store, including most commonly POSIX file systems and cloud object storage but also zip files as well as relational and document databases.

Features:

- Chunk multi-dimensional arrays along any dimension
- Store arrays in memory, on disk, inside a Zip file, on S3, etc.
- Read and write arrays concurrently from multiple threads or processes.
- Organize arrays into hierarchies via annotatable groups

File Formats – OME-ZARR and OME-NGFF

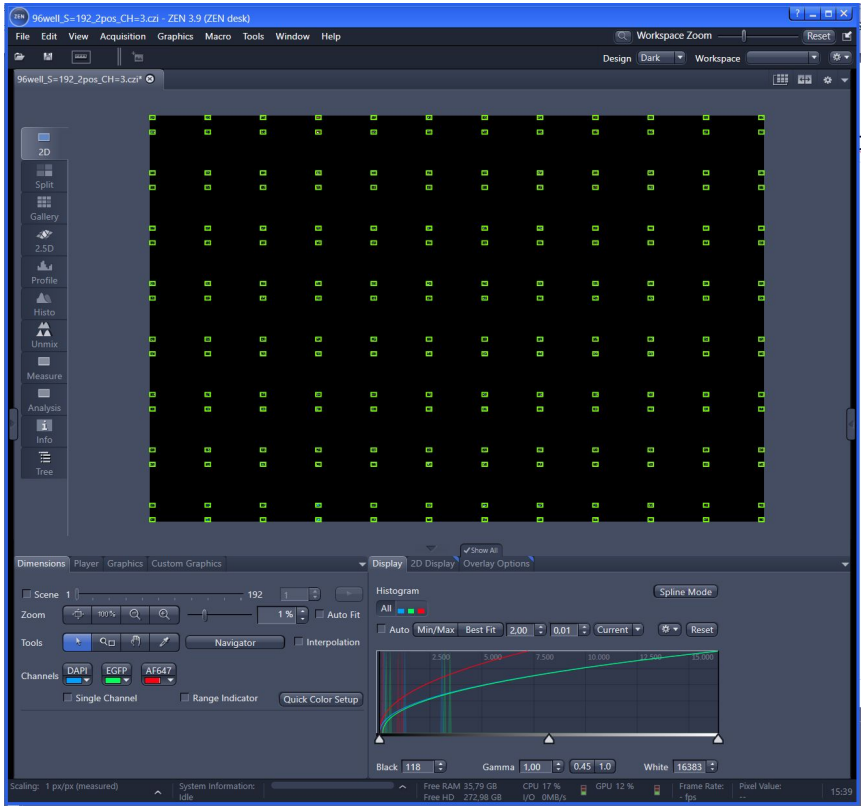
The future of microscopy image file formats (for processing)?

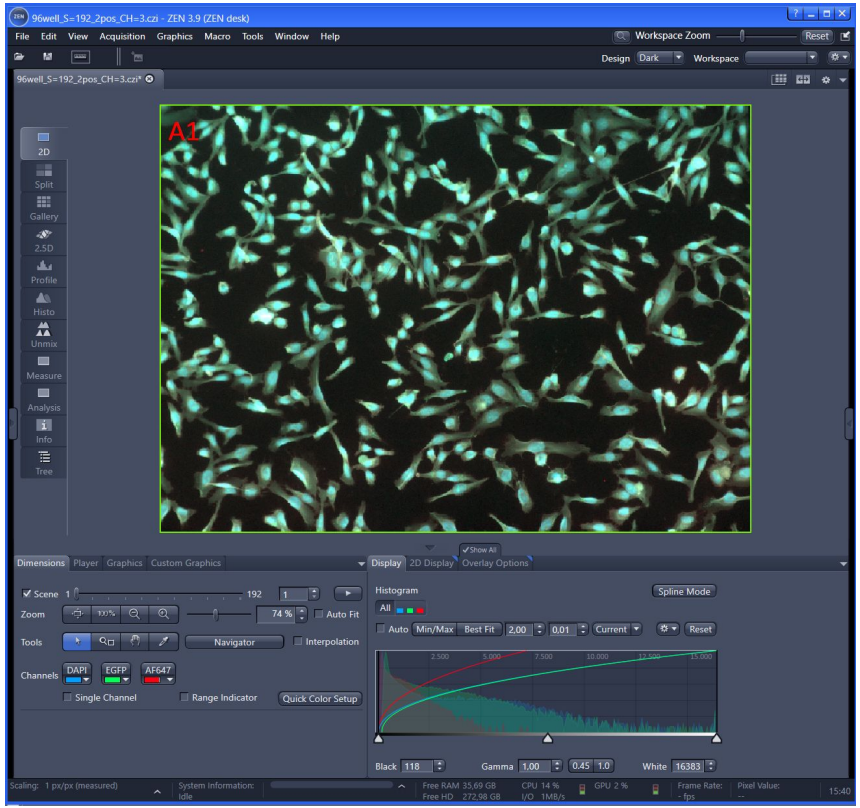
- OME-TIFF combines 2D TIFF format with metadata cast in XML in the TIFF header
- appropriate for many applications, where the plane-based access pattern is appropriate
- N-dimensional formats like HDF5 (“Hierarchical Data Format”) provide much more flexibility and allow chunking across different dimensions chosen by the user
  - depend on fast random access to the entire file not provided by large scalable cloud-based storage
- OEM-ZARR is basically ZARR + OME meta specifications
- specification is called **OME-NGFF** (**OME**-**N**ext **G**eneration **F**ile **F**ormat)

OME-ZARR is a cloud-optimized, chunked format that functions as a common API for both desktop, cluster, and web-based tools as well as national and international repositories.

File Formats – OME-ZARR and OME-NGFF

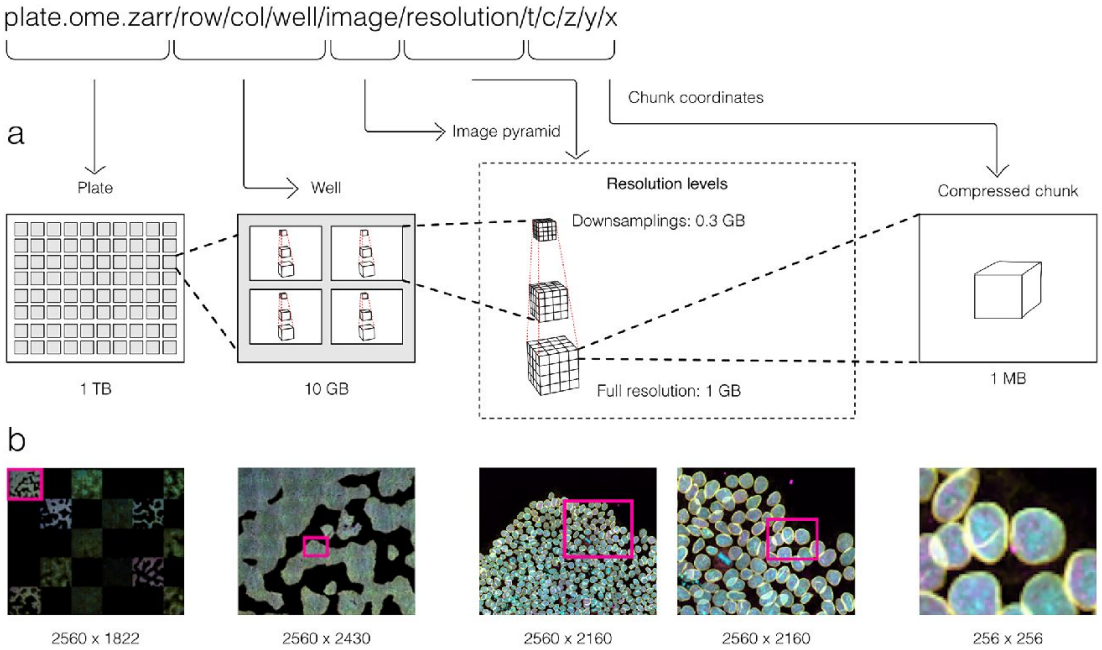
Application example: Annotated hierarchies for Plate data





File Formats – OME-ZARR and OME-NGFF

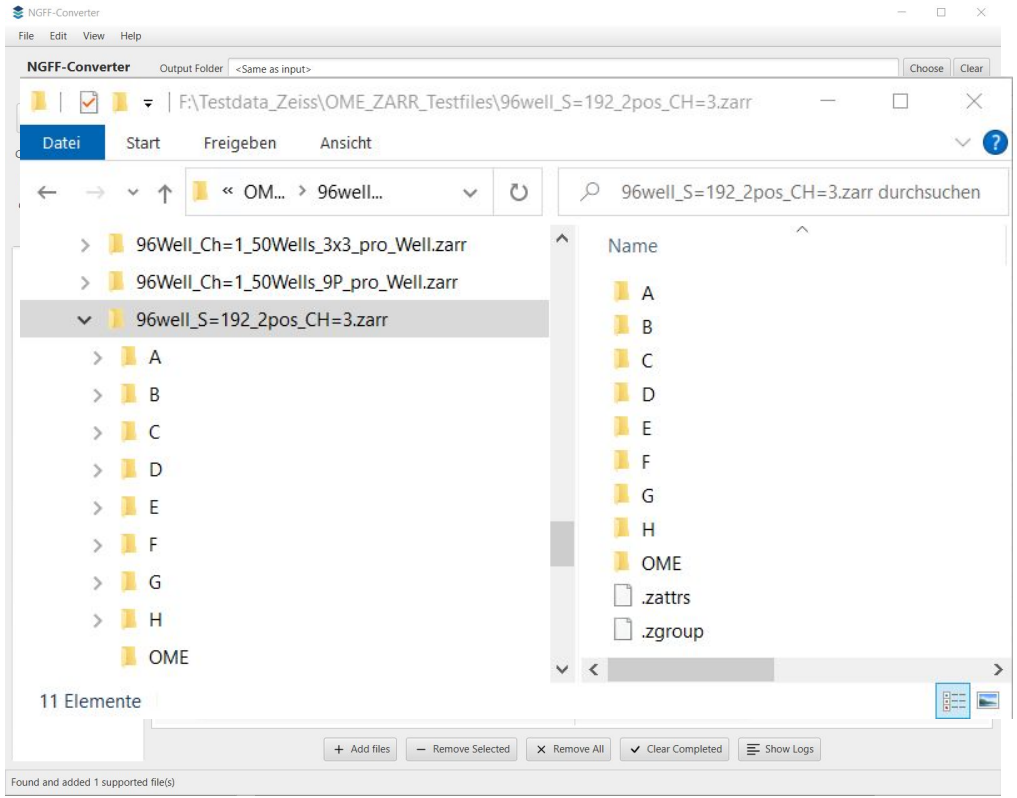
Application example: Annotated hierarchies for Plate data

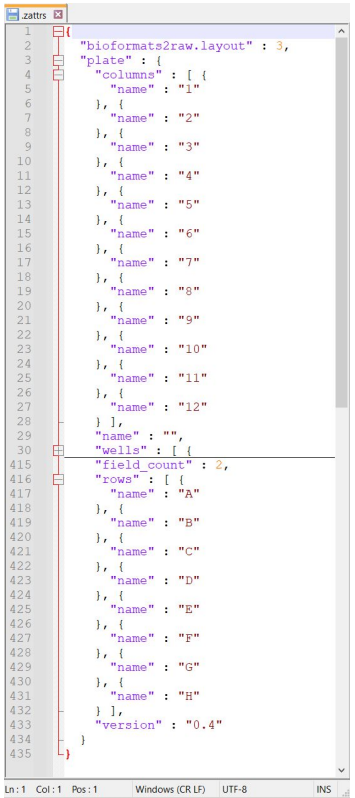


- top-level directory can be 1TB plate with > 100000 pixel in XY
- lowest level directory has chunks of N-dimensional data of 1MB
- Amount of pixel at different zoom levels to be loaded by the client is manageable
- Chunks can be compressed
- Metadata can be attached to each group and array separately in web-readable JSON files

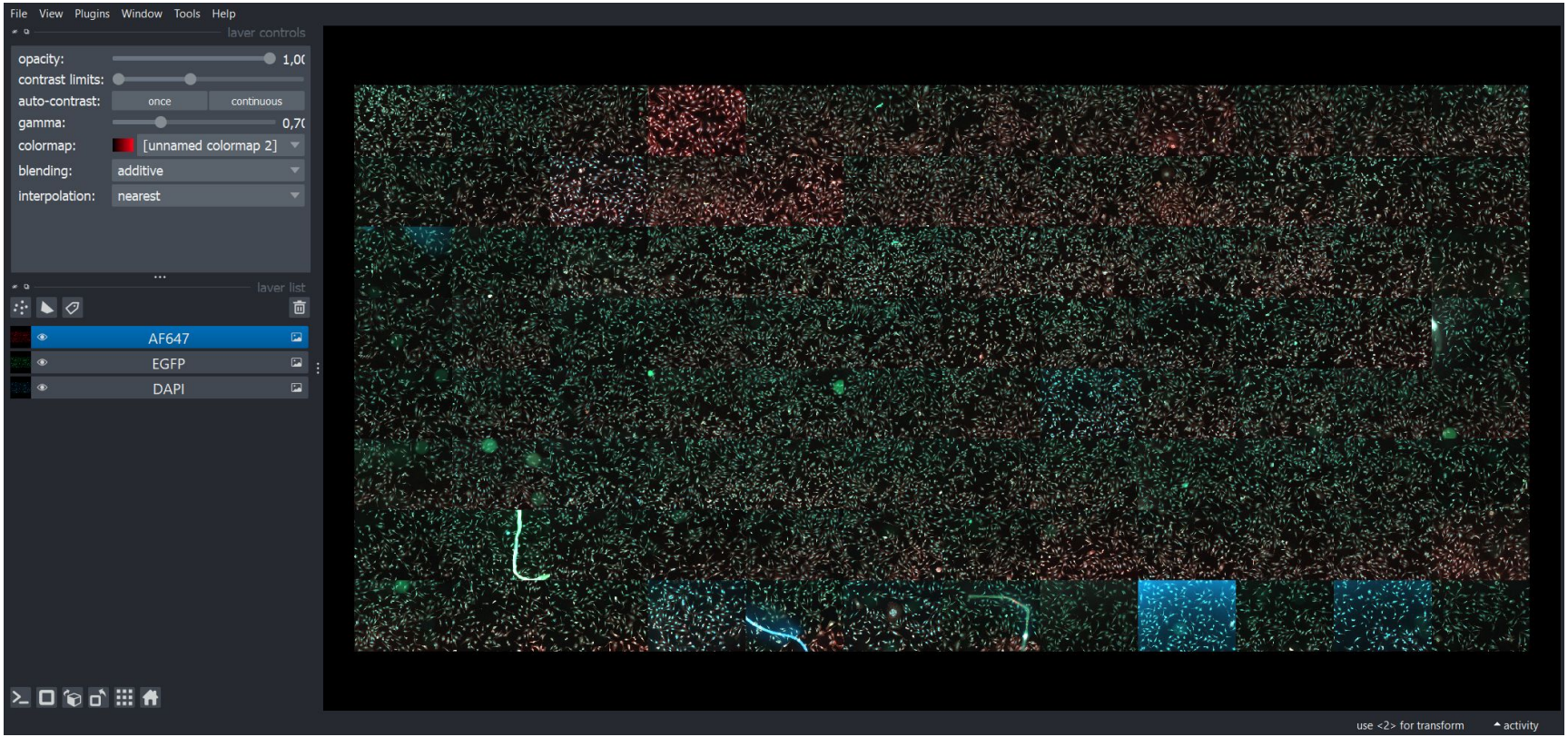
File Formats – OME-ZARR and OME-NGFF

Application Example: Annotated hierachies for Plate data



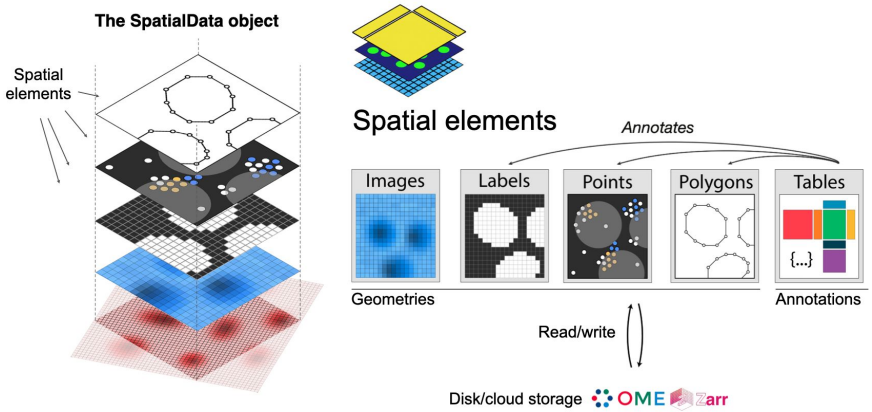


File Formats – OME-ZARR and OME-NGFF
Application Example: Visualize in Napari

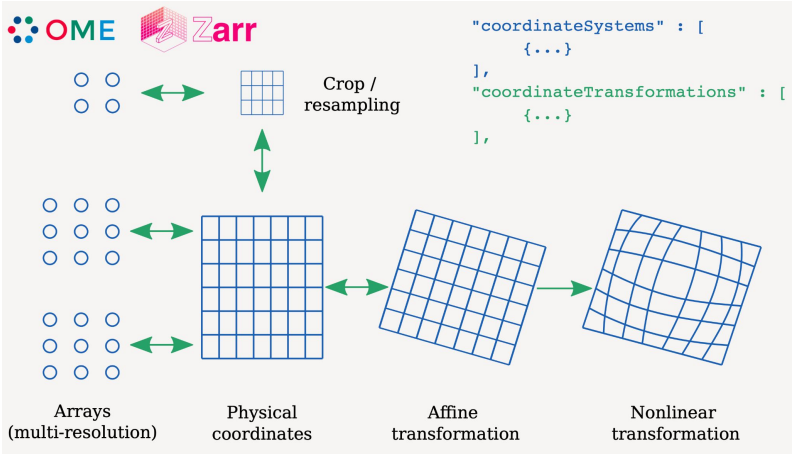


File Formats – OME-ZARR and OME-NGFF
Additional Spatial Data and Tables

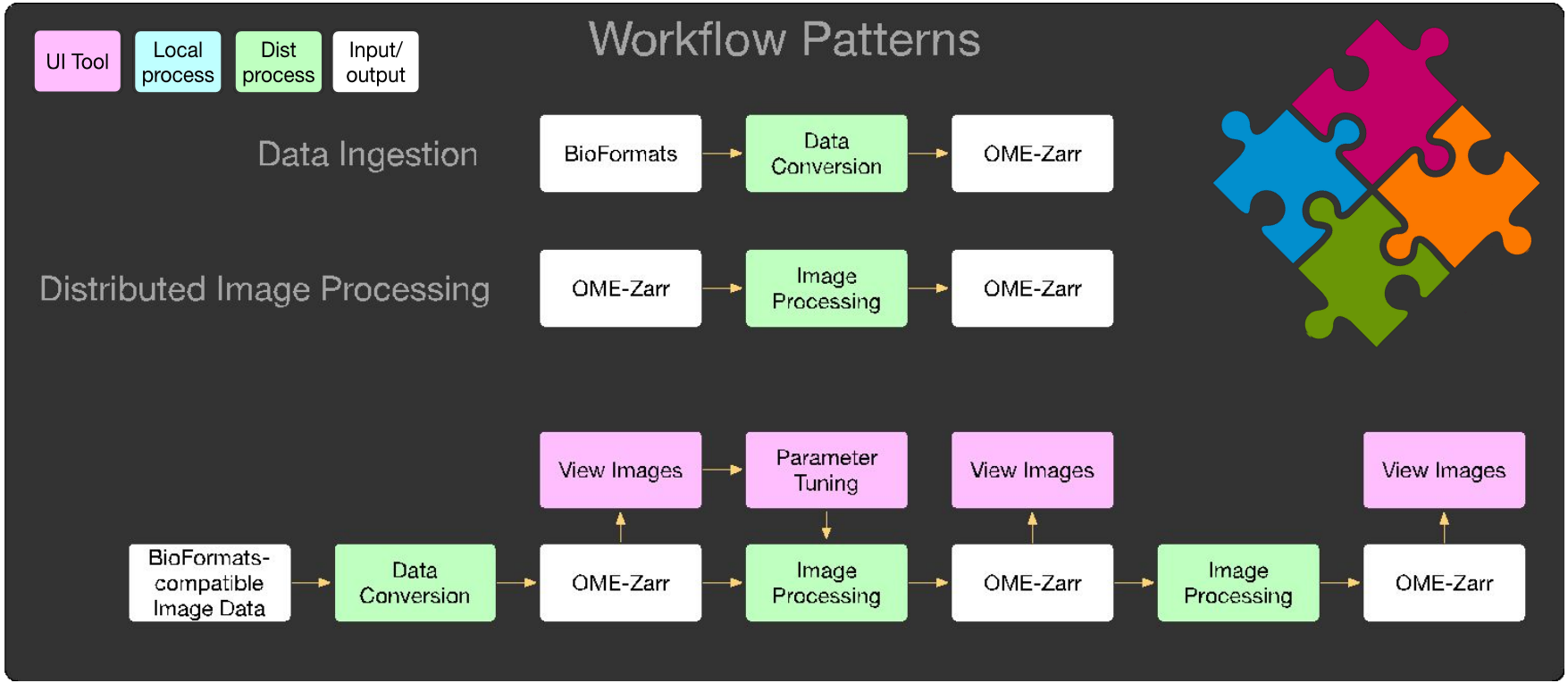
Tables (i.e. SpatialData)



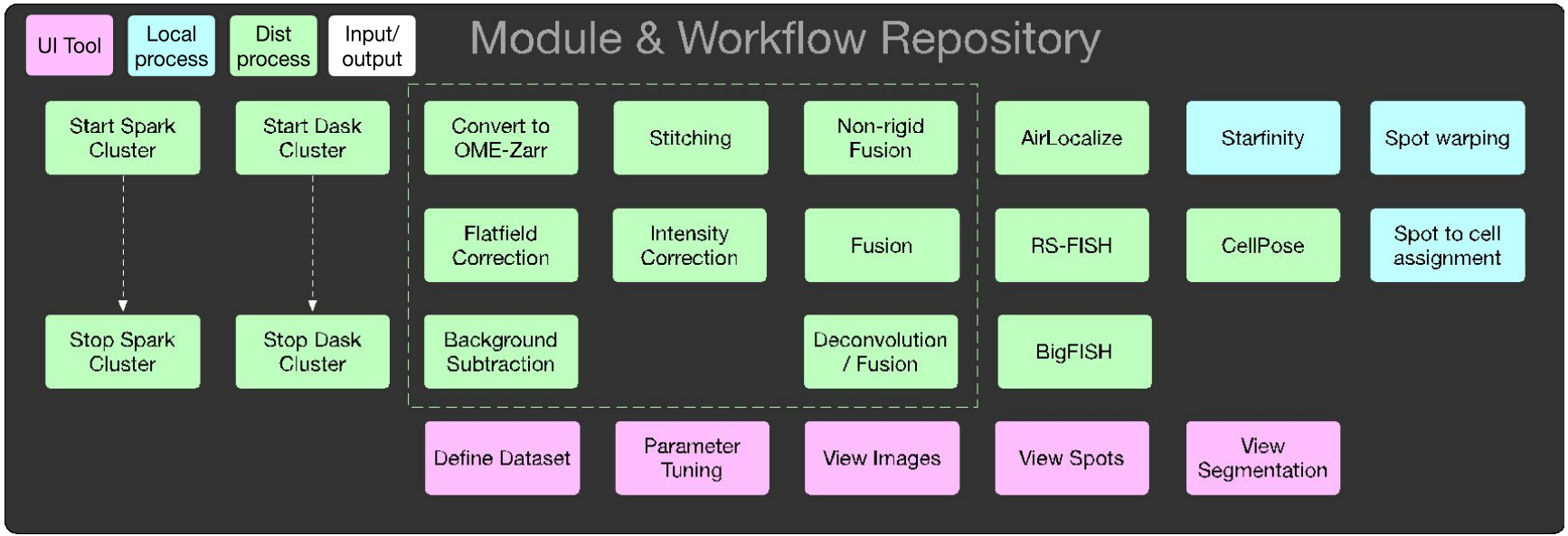
Transforms



OME-ZARR – How is it used already



OME-ZARR – How is it used already



- Reusable modules and subworkflows
- Each module is encapsulated into a BioContainer (https://biocontainers.pro/)
- Module repository: https://github.com/JaneliaSciComp/nextflow-modules

Kudos & Links

- https://www.zeiss.com/microscopy/en/applications.html
- https://www.ibiology.org/online-biology-courses/microscopy-series/
- https://www.youtube.com/watch?v=d8Tqoo0S6gc
- https://github.com/CamachoDejay
- https://www.tuwien.at/phy/iap/biophysics/team/gerhard-schuetz
- https://www.microscopyu.com/tutorials
- https://petebankhead.gitbooks.io/imagej-intro/content/
- https://downloads.micron.ox.ac.uk/lectures/micron_course_2020/lecture-20-introduction-bioimage-analysis.pdf
- https://www.edx.org/course/image-processing-and-analysis-for-life-scientists
- http://www.imaging-git.com/olympus-website-bioimage-data-analysis
- https://bioimagebook.github.io/
- https://www.biodip.de/w/images/6/67/2011-10-Basics_of_Imaging_Processing_Course.pdf
- https://github.com/dwaithe/ONBI_image_analysis
- https://github.com/quiwitz/Python_image_processing
- https://neubias.github.io/training-resources/
- https://www.scribbr.com/methodology/reliability-vs-validity/
- https://www.scribbr.com/methodology/reproducibility-repeatability-replicability/
- https://protocolsmethods.springernature.com/posts/intelligent-microscopy-work-smarter-not-harder
- https://www.nature.com/articles/s41592-022-01589-x?utm_campaign=related_content&utm_source=MTHDS&utm_medium=communities
- https://www.gu.se/en/core-facilities/centre-for-cellular-imaging/smart-microscopy
- https://www.nature.com/articles/s41592-023-01912-0
- https://www.nature.com/articles/s41592-019-0708-0
- https://www.eurobioimaging.eu/service/feedback-microscopy-FDBKM
- https://neubias.github.io/training-resources/multidimensional_image_basics/index.html
- https://forum.image.sc/t/outcomes-of-the-next-generation-bioimage-analysis-workflows-hackathon/88733
- https://link.springer.com/article/10.1007/s00418-023-02209-1
- https://doi.org/10.1038/s41592-022-01482-7
- https://doi.org/10.48550/arXiv.2101.09153
- https://doi.org/10.1101/2021.03.31.437929
- https://doi.org/10.1038/s41592-021-01326-w
- https://biocontainers.pro
- https://github.com/JaneliaSciComp/nextflow-modules
- https://doi.org/10.1016/j.cell.2021.11.024