Arkitekt

An open-source framework for modern bioimage workflows

Provisional Doctoral Thesis manuscript by

JOHANNES ROOS

Table of Contents

1 Abstract

Bioimage analysis workflows have transformed dramatically over the last years, accelerated by the emergence of deep learning. Indeed, once thought impossible challenges like 3D segmentation of complex microscopy-data or data-driven multidimensional microscopy, also called Smart Microscopy, seem now reachable, and new imaging modalities are breaking records in both resolution and acquisition speed.

This shift brings along a variety of new software platforms and tools, as well as the requirement of dedicated computing resources like GPUs. Plagued by the absence of a common framework for these tools, however, tedious data management, complex orchestration and painful integration of new technologies have become a reality and currently restrict these advanced, distributed bioimage workflows to a limited set of few programming experts.

Additionally, most existing methods are still limited in their real-time capabilities and are usually restricted to off-line analysis workflows, where analysis happens after the acquisition, limiting emerging *smart* workflows, where the analytical result can influence the acquisition.

This PhD thesis introduces a new open-source software framework that acts as a middleman between users and bioimage applications: Arkitekt. Arkitekt allows for the visual and user-friendly design of modern bioimage workflows, orchestrating existing popular bioimage software locally or remotely in a reliable, efficient and in real-time. It interfaces with popular interactive visualization and analysis software, like ImageJ and Napari, but also easily integrates developer scripts and acquisition software.

This thesis is organized in 4 major subsections. After a general introduction of bioimage analysis history and a detailed review of the modern analysis workflows it fully describes the main features of Arkitekt. It then illustrates and validates Arkitekt and its capabilities on representative advanced bioimage workflows and discusses its limits and potentials.

2 Introduction

This introduction serves three purposes. First it gives the reader an understanding of how the current landscape of bioimage analysis and their methods is shaped, how the methods of the past got us here, and how new methods continue to evolve it. Then it elicits the challenges users of the analysis methodologies face, together with the solutions that have been proposed and the challenges they meet and fail to meet. Finally, it explains the reasons to develop a new software platform aiming to overcome these challenges.

2.1 How images shape science

Biology, like every other discipline of science, is fundamentally grounded in the analysis of data. Understanding the mechanisms of life, from the level of ecosystems down to the molecular machinery within cells, relies on our ability to *draw meaningful conclusions* from complex datasets. We *measure* enzyme activity in a biochemical assay, *track* animal behavior in an ecological study, or *visualize* the dynamic interactions of proteins within a living cell.

The process of data analysis in biology is therefore not a mere afterthought or a peripheral aspect of the scientific endeavor. Rather, it is an integral part of the scientific method itself, underpinning each step from the formulation of the hypothesis to the design of experiments, to the interpretation of results, and ultimately the generation of new knowledge. It allows us to discern patterns, identify relationships, and make inferences about the biological phenomena we are studying.

It comes as no surprise then that tied to the evolution and emergence of new analytical methodology, we see groundbreaking biological insights follow: Western blots enabled us to identify the role of the protein tau in Alzheimer's disease (Goedert et al. 1992), while the discovery of the polymerase chain reaction aided in deciphering some forms of hereditary breast cancer. (Wooster et al. 1994)

One field of analysis in biology, however, has played, and continues to play one of the most important roles for advances in scientific discoveries: images.

Images allow us to map features of a biological system into a two- or three-dimensional context. Whereas most biological studies require the extraction and often the alteration of the studied elements from their native context, images can capture the precise spatial

organization and dynamics of the elements in their native environment. They allow us, to our own visual sense to perceive and analyze the *scale* of our biology: may it be from satellite images that monitor blooming patterns of the rainforest, to full body MRIs of a human.

Unparalleled in their significance are images of the intricate scales and the minute organization of our life that remain invisible to our naked eye. They are only revealed when using the tools of **microscopy**. These *bioimages* are integral to whole fields of biology like cellular or molecular biology and facilitate research aiming to understand how the complex cellular machinery works. They help to decipher how well-timed events that occur in the millimeter to sub-micrometer range can determine disease and health, growth and decay.

None of this would be possible without our continued efforts in pushing the limits of microscopy and the analysis of its bioimages, furthering our quest to resolve time and space ever so better. It is through the development of these methods that extend our own senses, that we are able to explore our microscopic world, revealing the fabric of live.

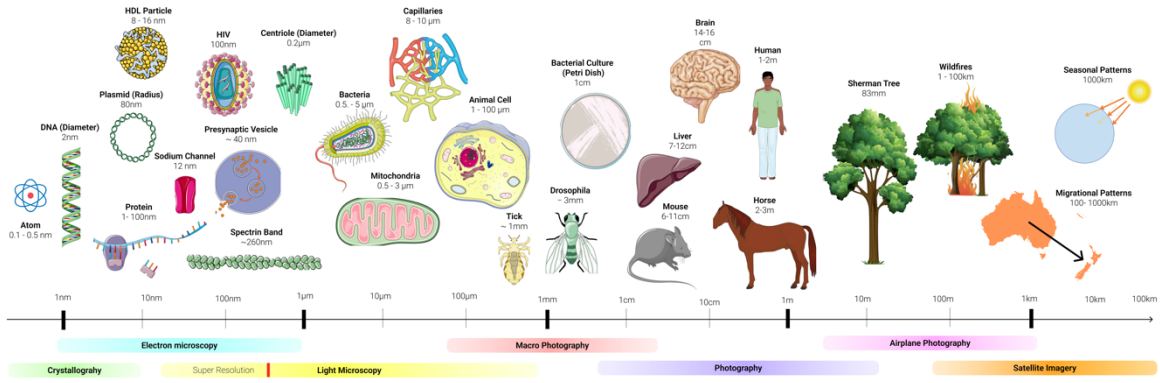

Figure 1 *The scales of biology and sciences and our preferred means of looking at them.*

.

2.2 A brief and highly selective history of bioimage acquisition and analysis

This chapter will focus on some aspects of the history of microscopic bioimage analysis, and how it has continued to evolve over the course of its history. It will delve into how scientists orchestrated their work with the images they produced with their microscopes, and how the analysis methods they developed changed our understanding of what a bioimage can reveal.

This chapter should by no means be considered a full picture of the history of bioimage analysis but aspires more to be a testament and an homage to how the **methods of a selective few**, became the **methods for everyone.**

2.2.1 The beginning of microscopy

The earliest recorded experiments with microscopy date back to the 17th century and to names like Galileo Galilei, who with his '*ochiollino'* (wink) built one of the first compound lights microscope (Strano 2009). Another pioneer, Robert Hooke, in his 1665 work Micrograph, was one of the first to reproduce the images of charcoaled vegetables he saw with his bare eye in minute sketches, depicting probably for the first time the intricate spatial organization of "*cells*", a term he later coined (Hooke 1665)

In similar efforts Antonie van Leeuwenhoek, improved on original designs of the microscope and with this very high magnification microscopes build around the 1670s, applied it to various fields of observational biology like depicted populations of bacteria in water bodies (Egerton 1968). His collected body of work awards him the honorary common title to be the *first* microbiologist.

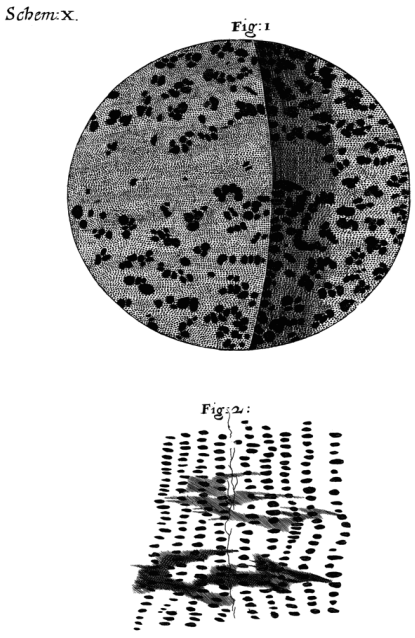

Figure 2 *Leeuwenhoek's first depiction of charcoaled vegetables (adapted from the original publication in1665).*

What unified the analytical process of these early microscopists, is their efforts to depict and describe the image they observed with eyes, trying to relate their size and shape with objects in the real world. Leeuwenhoek in his work even mentioned rudimentary

scales that he engraved in copper and using his beard hair was able to measure establishing the first *morphometry* in microscopy. (Davis 2020)

2.2.2 Staining for human interpretation

Another early milestone in the microscopic analysis of bioimage data was the introduction of advanced staining techniques in the early 1900, that enable scientists to selectively target and label structures and make them visible in their microscopic images. Building on the Golgi staining method, Ramon y Cajal used advanced silver nitrate staining techniques to stain collections of brain tissue, which he then traced and systematically compared. This not only gave evidence for the *neuron doctrine*, which established the neuron as the building block of brain tissue, but also describing neuronal features such as dendrite cell body and axons. (Garcia-Lopez, Garcia-Marin, and Freire 2010)

Staining was one of the first ways scientists were able to selectively highlight specific structures of interest in their samples, providing new interpretability to bioimage data and establishing a technique that set the foundation for the *segmentation* of the bioimage into labeled subregions.

2.2.3 The first bioimages

With the advent of photography in the form *daguerrography* in the late 19 hundreds, it did not take long for pioneering work of scientists to use it in conjunction with microscopy and create so called *micrographs.* (Woodward 1876)

Photomicrography allowed for the capture and preservation of specific microscopic views, which could then be shared, reviewed, and studied in ways that live viewing through a microscope could not permit. These *micrographs* brought a level of objectivity to microscopy. Prior to its invention, scientists had to rely on their drawings and descriptions to document microscopic observations, which were inherently subjective and prone to human errors. Photographic images provided a more accurate and *reliable* record of what was observed under the microscope. These were arguably the first bioimages.

2.2.4 Going beyond light

Trying to overcome the limitations of Light microscopy which is inherently limited in its resolution to a few hundreds of nanometers by the diffraction, the 20th century saw the advent of new modalities in microscopy. The electron microscope invented by Ernst Ruska and Max

Knoll in 1938, allowed scientists to go beyond the diffraction of light to and monitor biology with unprecedented nanometric scales. (Knoll and Ruska 1932).

Their method of sending a beam of electron through a biological sample and measuring the changes in absorption through a oscillograph (Ruska 1987) however, not only ushered in a new era of nanoscale biology, but also had long lasting effects on our understanding of what constitutes an image. Whereas previously images were generated as a capture of a 2D plane at a single moment in time, these techniques broadened our understanding of an image to that of continuously acquired *scan* of a two 2D sample.

Albeit it has become quite normal to think of an MRI, or a picture of an electron microscope as an *image*, this represented a conceptually new idea: An Image, resulted from a *reinterpretation* of data that was acquired in a different context (e.g., a time series of voltage fluctuations in scanning electron microscopy), becomes reinterpreted as a 2D image.

2.2.5 Fluorescence Microscopy

Even though limited by its resolution, light microscopy continued to be a method of choice to monitor our biology, mostly due to its versatility and compatibility with monitoring living organism, something that electron microscopy could not enable due to its fixation protocols. A long and well known phenomena *fluorescence*, furthered the abilities of light microscopy in the 1940s and 50s through pioneering techniques, like the fluorescence microscope pioneered by Ellinger and Hirt (Renz 2013), as well as the first fluorescently labeled antibodies against pneumococci (Coons et al. 1942). These techniques helped create one of the most important methods of modern microscopy: immuno- and later direct (genetically encoded) fluorescence.

Fluorescence provides the microscopist with a wide array of benefits: selectively highlight structures with high contrast even in the most crowded neighborhoods, discerning the expression pattern of proteins over time, just to name two. It also provided the biological foundation for multi-dimensional, *multi-channel* bioimages, where each channel could highlight a different biological feature in the same spatial and temporal context.

2.2.6 Digital Microscopy

The first *digital* image was created in 1957 by Russell Kirsch and his team at the National Bureau of Standards (now the National Institute of Standards and Technology, or NIST). This image was a digital scan of a photograph of Kirsch's 3-month-old son, Walden (Kirsch 1978). The photograph was scanned using a device called a drum scanner, which Kirsch and his team had built. The scanner used a photomultiplier tube to detect the light reflected off the photograph as it was rotated on the drum scanner. This light was then converted into an electrical signal, which was digitized and stored on magnetic tape.

Kirsch's work was groundbreaking because it was the first time an image had been digitally scanned and stored in a computer. This laid the foundation for the development of digital imaging and image processing techniques that have become so ubiquitous in modern bioimage analysis. This development continued throughout the 90s when digital detectors (photomultipliers and CCD cameras) completely transformed microscopy from analog to digital, allowing brightfield and fluorescence microscopy images to be collected instantaneously and with higher and higher sensitivity. In parallel, microscopes became motorized and controllable by computers to perform automated tasks. Flows of digital multidimensional images started to be collected, waiting to be analyzed and interpreted. The digital revolution had started.

2.2.7 Going 3D

Another concept of 1957 would set out to revolutionize the field of microscopy in the coming years: the confocal scanning microscopy (Minsky, 1957). It combined the coherent, focusable light of lasers and simple optical elements such as the pinhole, to dramatically improve on the sectioning abilities in light microscopy.

Whereas previously images were generated as a capture of a 2D plane at a single moment in time, this, and other developments slowly evolved the discipline of microscopy to capture our biological in all its 3 dimensions.

2.2.8 Image Processing

In the years after the digitalization of images, computer aided more and more in the analysis. During the 1960s, the Jet Propulsion Laboratory (JPL) led the development of digital image processing for the purpose of enhancing images of the moon for the Ranger and Lunar Orbiter

programs. The techniques developed here laid the groundwork for many of the digital image processing methods used today.

Processing allowed humans to better make sense of their data, by enhancing human perceivable metrics of the data such as contrast or rendering data in new angles and projections. The coming years saw a burst of new methodologies, and soon techniques like denoising and Fourier image analysis became standards when dealing with bioimages. How ubiquitous signal processing has become is probably best exemplified by the **maximum intensity projection**, a technique developed by Wallis for nuclear imaging (Wallis et al. 1989). It now represents a universal tool in the analysis toolbox of modern bioimage scientist.

Image Processing however was not only limited to making images more human interpretable but could also be used to augment the image computationally. One milestone in this regard was deconvolution microscopy which allowed for the first time to digitally improve the resolution taking into account the image formation process and digitally reassigning diffracted light to its original place. (Sibarita 2005)

2.2.9 Computer Vision

Working with image processing algorithms to establish the detection of edges in images, Larry Roberts in 1965 laid the groundwork for the field of **computer vision** (Roberts 1963). Computer vision gave new interpretability and objective to the bioimage world, as it established a methodology for the computer to automatically detect complex structures and *separate* and *divide* biological structures into interpretable *segments*, without human intervention and bias.

In parallel, another brainchild of Marvin Minsky, the “Perceptron,” (“Perceptrons", Marvin Minsky 1969), provided a mathematical model foundational to modern neural networks and was starting to shape the field of artificial intelligence.

Developing in parallel, these two fields found a beautiful marriage in the 1980s when Yan LeCun applied the first convolution neuronal net to the detection of postal codes (LeCun et al. 1989). His technique would (with due time) revolutionize the modern computer vision, and provide the foundation for modern deep learning algorithms such as Stardist (Weigert et al. 2020) or Cellpose (Stringer et al. 2021), which are now providing fully automated *segmentation* algorithms.

2.2.10 Super Resolution Techniques

The recent years are a perfect illustration of interdisciplinarity in microscopy. With the integration of processing, optics, and chemistry, it became possible to overcome the physical barrier of the diffraction limit, which limits the resolution of an image: any image of an arbitrarily small source of light imaged using a lens-based microscope won't render a point but a point spread function (PSF), usually an Airy pattern, with a central peak of minimal width of ~200–300 nm.

The super-resolution revolution was ushered in by STED (acronym for stimulated emission depletion) (Hell and Wichmann 1994), which both exploited photophysical properties of the fluorophores and the ability of optical elements to minutely shape the lasers' wavefront and therefore its illumination spot, to reduce the effective excitation spot and hence improve the spatial resolution. STED stood as a pure opto-chemical approach to break this diffraction limit but soon other techniques came about that added a variety of computational postprocessing to enhance resolution. SIM (Structured Illumination Microscopy) (Gustafsson 2000) soon utilized specially crafted illumination pattern to enable Moiré`s method, while SMLM (Single Molecule Localization Microscopy) techniques exploited fluorescence properties to stochastically activate a subset of fluorophores and break the diffraction limit by digitally localizing isolated fluorescent molecules (Betzig et al. 2006)(Rust, Bates, and Zhuang 2006) (Sharonov and Hochstrasser 2006).

Accounting for the tremendous impact of these techniques, the combined efforts of these pioneers resulted in the Nobel prize for chemistry in 2014 ("The Nobel Prize in Chemistry 2014"). The continuous development of these methods not only enables us today to structurally monitor our biological data in the nanometer and recently Angstrom range (Reinhardt et al. 2023), but also brings about new ways of tracking and analyzing molecules and their pathway and cascade inside the cell.

2.3 Bioimage Analysis today

Analysis and acquisition methodologies have changed our understanding of biology and let us dive every deeper into the scales of our biology. They pushed the boundaries of what images can portray and how we (and increasingly computers) can use to interpret their content.

This section describes where these methods and others have brought us, and what modern bioimage analysis looks like. It will explore what a bioimage today is in practice, and how most users organize data and metadata. It will then give an overview of the elements of modern bioimage analysis, how we orchestrate these elements and the tools that are used to facilitate them. This section aims to provide a *descriptive* overview of this current state, trying to cover the wide spectrum of bioimage analysis practice as done by (self-described) *non-experts* and *experts* alike.

Given the sparse set of *descriptive* literature on this subject, this section builds on the finding of three large scale community surveys of the bioimage analysis world. A survey performed in 2020 by the Center for Open Bioimage Analysis (COBA), which surveyed 484 scientist spanning various categories of professional familiarity with bioimage analysis in the US (Jamali et al. 2021). A 2021 online survey (Schmidt et al. 2022a) interviewing 204 participants of various analysis expertise with geographic focus on Germany. And finally a survey conducted by Sivagurunathan and colleagues which elaborates and updates the results by Jamali for the year 2022 (Sivagurunathan et al. 2023). It should be noted here that these surveys are subject to their biases in selection (first and foremost the bias of geographic selection to the global west), they however provide a unique *user perspective* on the current landscape.

2.3.1 What is a bioimage?

Before diving into the modern analytical process of images it is important to understand what bioimages today *are*, exploring not only *what* we image but also how we store the image data and its biological context.

2.3.1.1 About Data

Bioimages today constitute a wide range of different data that range from a snapshot of a cell culture acquired on a bright-field microscope, over 3D point scanned volumes on a STED fluorescence microscope, to multidimensional data acquired with a light-sheet fluorescent microscope and the reconstructions of localization events in single molecule localization microscopy.

Given this wide range of analytical data, it is hard to define all-encompassing features of this data. An almost universal denominator of bioimage data today however is that it is *digital*. Another common denominator is the emphasis on the multidimensionality of the data, be it multi-channel 2D images (X, Y, C) or time-series of 3D stacks (X, Y, Z, T).

According to the most recent survey, the most analyzed type of data today is 2D labeled fluorescence cells (Sivagurunathan et al. 2023) and most research data is acquired using confocal fluorescence microscopy (Schmidt et al. 2022a). Interestingly, and showing the dynamic nature of what modern bioimage analysis is applied to, this already represents a shift with regards to the 2021 survey, where brightfield non-fluorescent cells field images represented most data (Jamali et al. 2021).

Even though 2D acquisitions of fluorescent cells are the most common processed datatype, 3D acquisitions are becoming a standard (with the third axis establishing the Z-axis). Time lapse microscopy (2D+T or 3D+T) is also very common to capture the dynamics of biological processes.

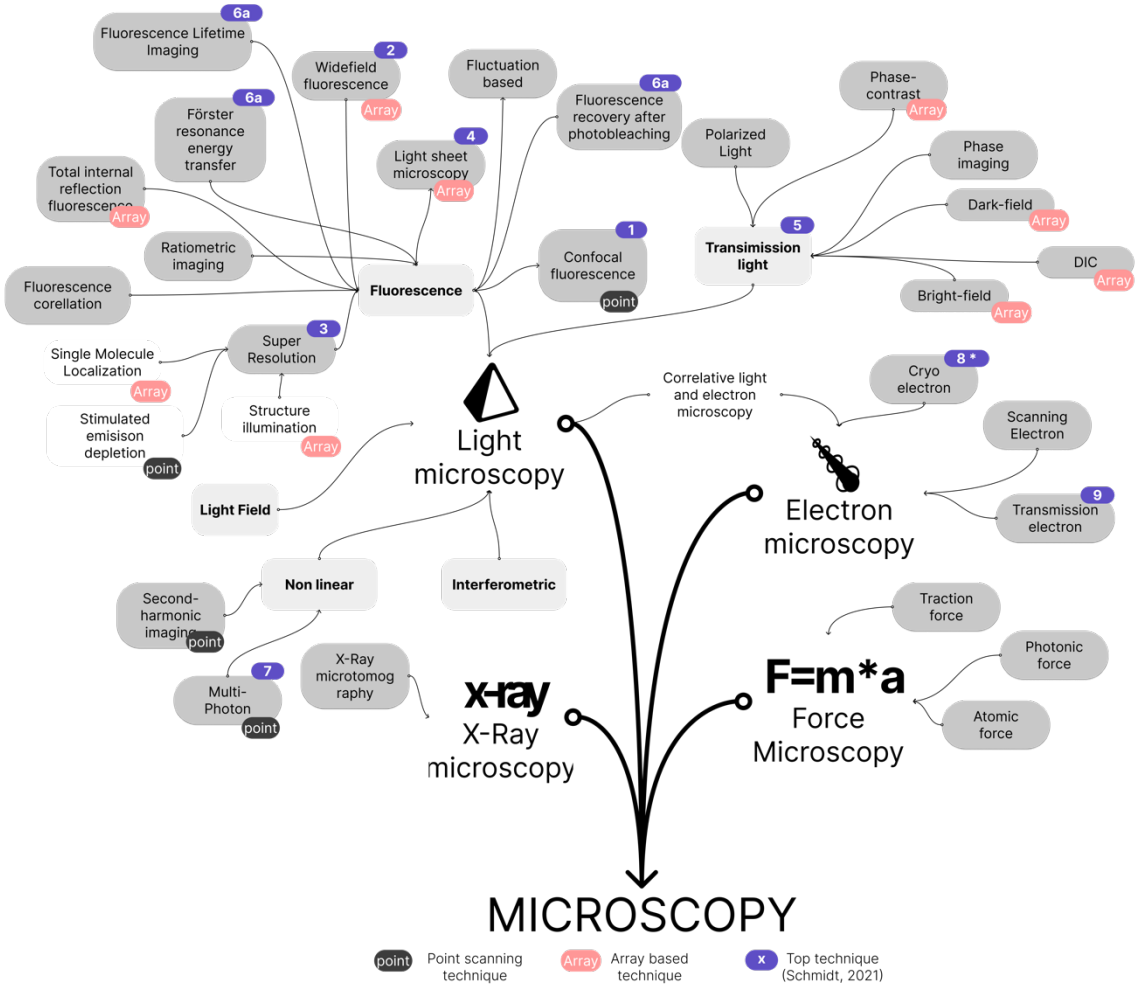

Figure 3 *The techniques of microscopy that produce the bioimage data of today stratified according to the EDAM Bioimaging Ontology (Kalaš et al. 2020), with highlighted "top techniques" according to the 2021 Research Data Infrastructure Questionnaire (item: "Imagine you had to choose ONE Method as the single most important method").*

2.3.1.1.1 Storage

Image storage today is facilitated through image file formats. For most users only apparent through their respective file-ending such as .JPG or .TIF image, file formats describe a computer understandable protocol of how to save the image to organized binary data (a binary *file*) and how to reconstruct the image back from this binary data.

The binary storage of images needs to include a step of discretizing the original signal in computer readable byte-arrays, a process called *rasterization*. While this step varies from file format to file format and a description thereof is out of the scope of this thesis, for the most part this discretization will yield a byte array where each value in the array represents the

intensity at one pixel or voxel in the image. As this byte array then represents the whole (or a subset of the) image in *one linear* sequence, and additional saved parameter, the *shape* determines during reconstructing the image, where to break a line or column.

While *sparse* image formats exist (Potocek et al. 2020), this *raw* byte-array representation of the original image is predominantly *dense,* meaning every original pixel intensity gets assigned a value in the byte array, even when this value is a zero value. As this dense representation of an image can be costly in terms of storage, modern image file formats often use some form of *compression*. Compression of image data is facilitated through algorithms that compress the original raw image either *lossy* or *lossless.* Lossless algorithms exploit symmetries (e.g. a long sequence of zeros) in the original data to compress *all* of the information into the smaller byte array and on decompression will yield an *exact replica* of the original data. Lossy algorithms, on the other hand, use heuristics such as averaging or frequency filtering to compress the image further, losing the exact information stored in the array, but trying to yield an adequate *representation* of the image on reconstruction.

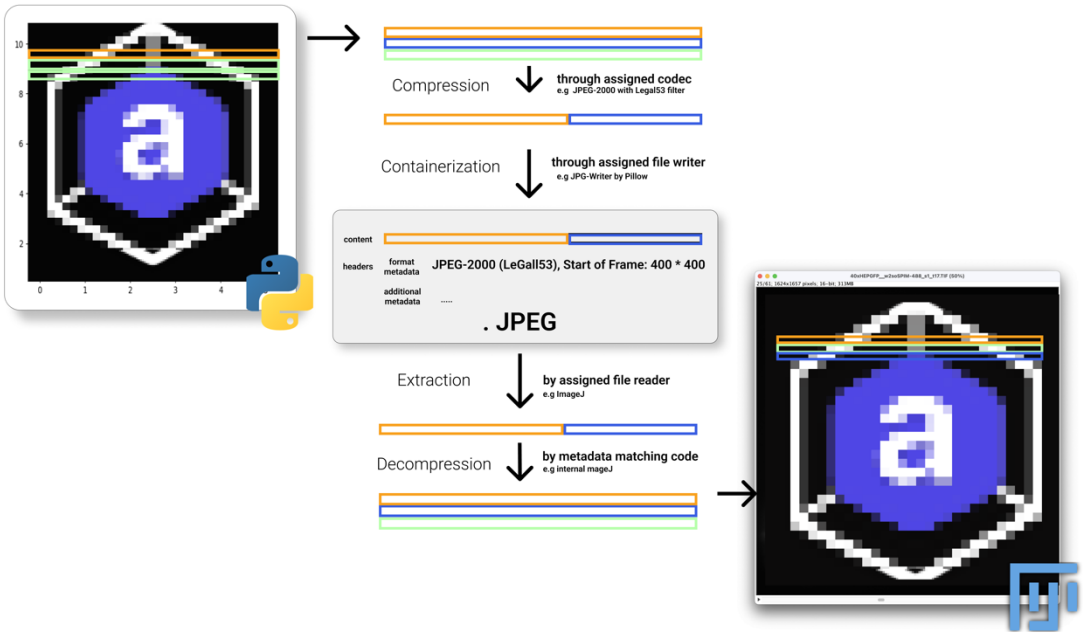

Figure 4 *Image Format protocol on the example of the JPEG Container Format/Protocol, transitioning an in-memory image from Python (here a matplotlib image) through the container format to Java (here FIJI).*

While a big subset of the world's photographic images is stored using lossy compression (using for-example lossy algorithms in the JPEG format), dedicated scientific image formats are designed to use lossless compression, conserving all the original raw information. The

TIFF (Tagged Image File Format) standard is a widely used open image file format for scientific images and uses performant lossless compression. While JPEF and TIFF provide open standards for image storage, most commercial microscopes provide their own dedicated file format for image storage (Nikon's ".nd", or Leicas ".lif").

2.3.1.2 About Metadata

A bioimage today not only includes the raw data of the image, the information at the pixel level, but also an ever-increasing set of information that *describes* the original raw data, its acquisition and processing parameters. This set of information is referred to as meta-data (emphasizing its relationship of describing the data), a term coined by Jack. E Myers in 1969 (Greenberg 2009).

Metadata has always been an integral part of a scientific bioimage, as it puts the image back into the real-world (biological) context. Today, it is widely understood that maintaining metadata is important to ensure the scientific integrity of the data (Schmidt et al. 2022b). Metadata storage and the organization and standardization of the metadata can vary and is highly dependent on the scientific lab practicing it. This section will briefly go into common forms of metadata storage management, as practiced today.

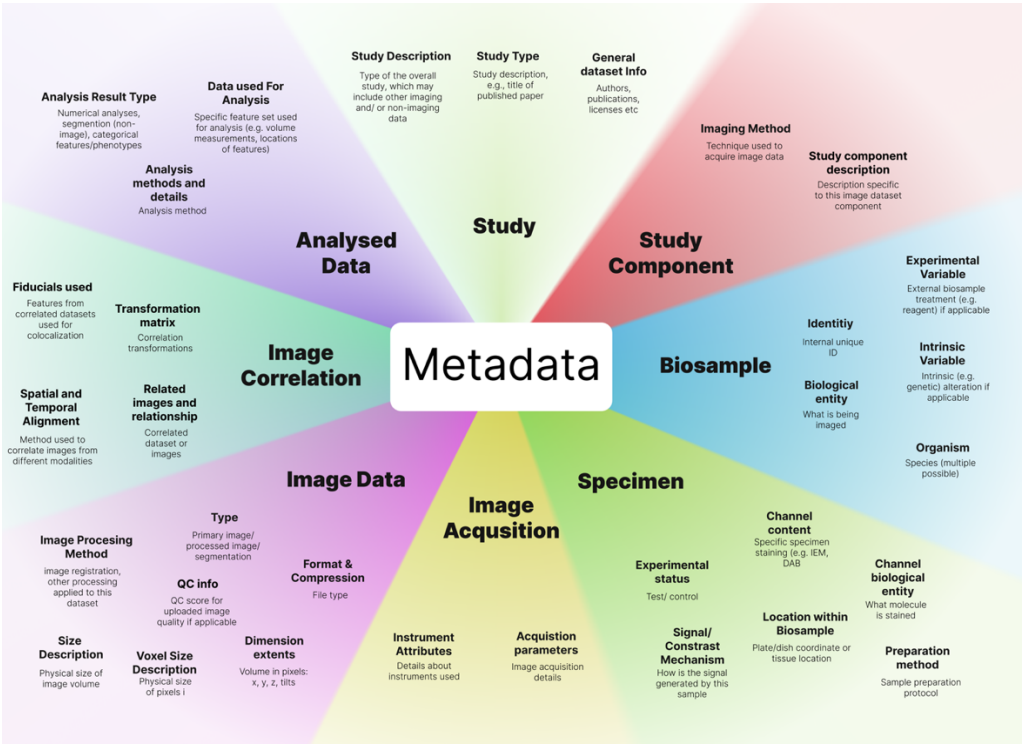

Figure 5 *Common Metadata according to the REMBI classification for recommended metadata in bioimage data (Sarkans et al. 2021).*

2.3.1.2.1 Storage

Metadata storage is facilitated through some orthogonal approaches that will be described here:

2.3.1.2.1.1 Contained

Contained metadata storage is a form of metadata that is inherently linked to the original data and will stay with the original file on standard file operations like copying or moving.

Filenames

The most common form of metadata includes simple filename descriptions like "yesterdays_experiment" or more complex filename schemes like "HEPG2_soSPIM448_T1_S3" (Schmidt et al. 2022b)

Metadata Containers

Metadata containers are dedicated areas inside the container format for the file, that are reserved for the storage of metadata. Examples include descriptive headers in TIFF files, that can include metadata about the original acquisition, or the OME (Open Microscopy Environment) XML headers (Goldberg et al. 2005) that are implemented in a lot of (commercial) file formats for microscopy data storage (e.g. LIF, NDI).

2.3.1.2.1.2 Associated Metadata

Associated Metadata is metadata that lives outside of the context of the data and is only associated by reference with the original file.

Directory

Directory based associated metadata, uses the nested structure of directories, to associate metadata with the files. This strategy is commonly implemented for example through a user generated CSV or Excel File in the same directory that describes properties of the images within the same directory (e.g. which antibodies where used, and in which concentration), and represents the most common additional metadata type (Schmidt et al. 2022a). Directory associated metadata can also include machine generated files like configuration files describing the acquisition parameters in "*ini*", "*json*", "*yaml*" formats.

Database

Associated metadata can also be stored completely apart from the binary data and the directory structure, in a database with a reference to original file. This scenario is often used

when search functionality is desired to restructure, explore and recontextualize the original image data according to its metadata, or when accumulating and archiving a lot of data and metadata. The OMERO-Server Project (Allan et al. 2012) is an important example in this space.

2.3.1.2.2 Schemas

Orthogonal to the types of storage, metadata management can be categorized around its efforts to standardize the metadata and how it allows a *structured retrieval* from the *storage* through a shared mental concept, a *schema.* A schema outlines the organization of the data (for example in a database), indicating how data entities relate to one another and the types of data that can be stored in each field.

2.3.1.2.2.1 No Schema (Schemaless)

At this level, there is no formal structure or standardization applied to the metadata. Metadata may be applied inconsistently or not at all, and there is no formal way to query or analyze it. This approach is often applied in scenarios, where data is considered highly ephemeral, like figuring out experimental conditions or acquisition parameters.

2.3.1.2.2.2 Implicit Schema

At this level, metadata is structured but not well standardized and the schema is not explicitly defined or enforced. For example, users might *agree* to always include certain information (like Timepoint with a T) in a certain format in the filenames, but there's no system in place to *enforce* this or to manage or migrate changes to the schema. Implicit schemas often rely on **key**-**value**-based semantics, where a key (e.g. Timepoint = T) is followed by a value (the actual value = 1). (Schmidt et al. 2022a). This approach heavily relies on user discipline and communication.

2.3.1.2.2.3 Explicit Schema

In an explicit schema, the metadata schema is explicitly defined and enforced. A common example is the OME (Goldberg et al. 2005) standard in microscopy files that defines a strict schema for which information is stored in which part of the header files. An explicit schema can offer powerful querying and analysis capability and is often designed to be machine interpretable. Explicit schemas vary in their scope of applicability: some schemas are established on the lab basis (file naming schemes), the application (application-specific metadata) or are decided and communicated globally (Allan et al. 2012).

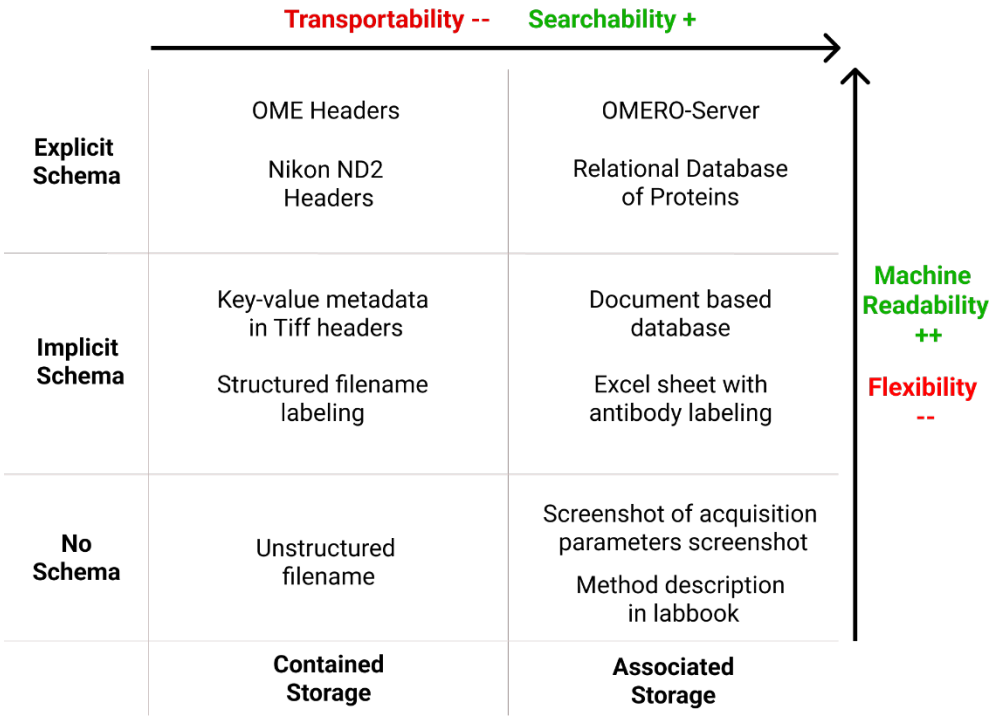

Figure 6 *Schema and Storage with common examples, as well as their implications for transportability (how easy is it to move metadata with data), searchability (how easy is it to explore metadata), machine-readability (how easy is it for a machine to automatically inspect the metadata) and flexibility of information (how easy is it to convey a complex idea in metadata).*

2.3.1.2.3 Ontologies

While schemas provide a structure to data and metadata, they do not establish what a specific relationship or datum inside this structure *means*. An *ontology* brings interpretability to this structure, allowing to establish a shared vocabulary and a high-level representation to model the domain.

This dichotomy is best exemplified by looking at a key-value pair in a filename: “Sample-Cortex_Fluorophore -GFP”. The schema describes the *sample* field and the *antibody* field (with values *Cortex* and *GFP*), while the ontology would assign meaning to the property *Fluorophore* and its value *GFP*. The word *GFP* finds its equivalent in the ontology as the abbreviation for *Green Fluorescent Protein*, a well-known fluorophore. If the order of values were to be switched to “Sample-GFP_ Fluorophore -Cortex”, the filename would still be schematically correct, but ontological non-sensical (as “*Cortex*” is not an *antibody*).

Ontologies just as schemas are often implicit and require human interpretation (e.g. when decerning that *GFP* is indeed referring to *Green Fluorescent Protein* and not *Green Friendly Postman)*. There exists however common efforts to provide a more standardized approach to ontologies: The Protein Data Bank (Berman et al. 2000) allows to map a unique protein identifier in a scheme to their large protein identification databank which can provide

additional information like the protein size, its dominant chains etc. A collection of this and other ontologies are provided European Molecular Biology Laboratory (EMBL), which aims to establish a unified interface for biological ontologies (“Ontology Lookup Service”, EMBL).

As the linkage of schema attributes to these wider ontologies often requires human intervention, efforts to link the metadata more deeply and machine readable to the corresponding ontologies have been made. These developments are commonly put under the umbrella term of “semantic web” (Lan et al. 2022) and rely on formats such as the “Resource Description Framework” (“RDF 1.2 Schema”, W3C), a metadata storage format that stores not only the metadata, but also inline describes the linkage to another web-resource that can take linked attribute, and expand it with additional metadata.

2.3.1.3 Annotations

Falling outside of the classical definition of metadata and data are additional data-structures that make up a modern bioimage, such as a *region of interest* (ROI) or *labels* in a segmentation mask. These annotations neither fall into the category of data and or metadata as they can act both like data when regarded as a subset (a sample) of the data chosen for analysis or processing, but also as metadata when providing information about the data — specifying which portion of the data is best described under a set of metadata.

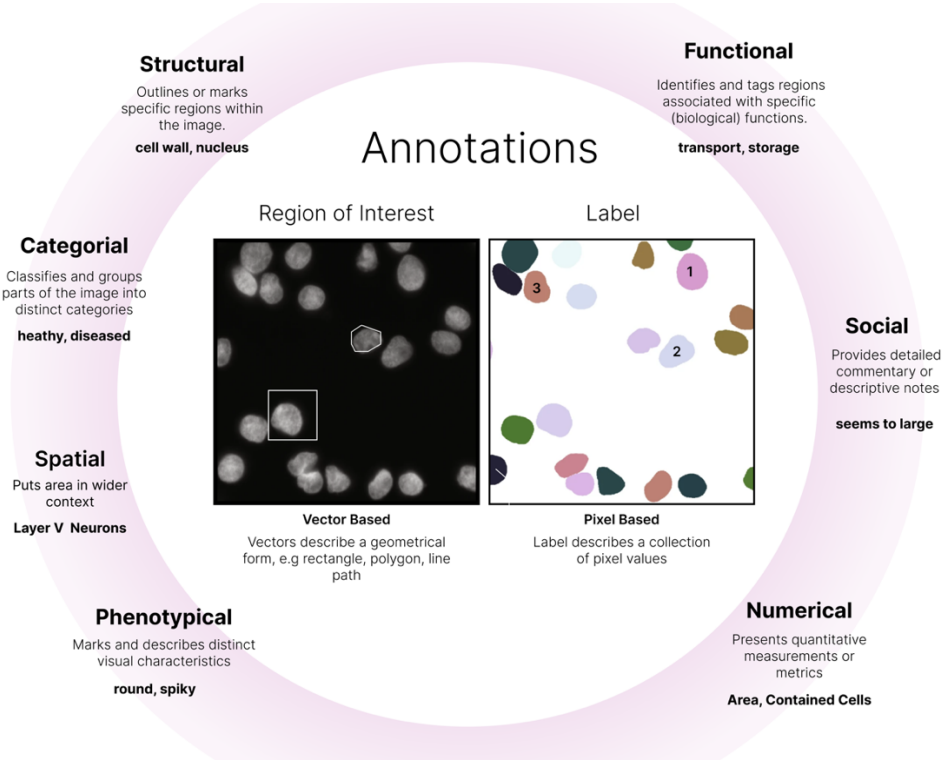

Figure 7 *Annotations: A non-exhaustive list about the annotation types that can be associated with images through labels and regions of interests.*

2.3.2 How do we analyze?

Most, if not all, bioimaging data is subjected to processing and analysis. (Schmidt et al. 2022b). This section will introduce the most common steps in this analysis, and how they are conceptually organized.

2.3.2.1 Steps in the analysis

Dealing with biological image data can be conceptually divided broadly into a few descriptive steps, listed below. These steps are often used to give a *mental model* of scientific analysis as a whole and find usage in training resources and within scientific publications to categorize their analysis. (Kalaš et al. 2019)

- **Acquisition**: This involves capturing images using imaging techniques, such as transmission light microscopy, fluorescence microscopy, confocal microscopy, electron microscopy, but importantly to be considered an analytical method also can also include some steps of *preprocessing* like converting a 2D timeseries of voltage signals into a 2D image.
- **Visualization:** This step includes visualization of the original or processed data to make it *interpretable* for the human. Often this step informs the sequence of steps to be undertaken afterwards.
- **Processing**: This stage includes operations like noise reduction, contrast enhancement, image normalization or deconvolution. The goal of processing is to improve the quality of the acquired images and prepare them for subsequent analysis or visualization.
- **Segmentation:** This step involves algorithms that separate the image into *interpretable* regions: This can include simple algorithms like binarization (*Background – Foreground*), but also more complex routines which divide the image into regions or objects of interest, such as organs, cells, or specific organelles within a cell.
- **Quantitative and Statistical Analysis**: Once objects of interest have been identified and segmented, quantitative measurements can be made. This could include measuring the size, shape, or intensity of objects, counting the number of objects, tracking objects over time, etc. It can also involve statistically analyzing the measurements to draw conclusions. This might involve comparing measurements between different biological conditions, or treatments, over time.

It is important to note that in this mental model, some analytical steps that use the same algorithm can conceptually fall into different categories. E.g. a thresholding algorithm can be used as a processing step when cutting unwanted background out of the original image for further processing or segmentation, but can also serve in a segmentation step to separate a labeled structure from the surrounding signal (e.g. in a SUSHI image where a die is used to stain the extracellular space) (Tønnesen, Inavalli, and Nägerl 2018).

2.3.2.2 Analysis Organization

The analysis of bioimage data is critical and often represents the most complex and time consuming step when dealing with bioimage data (Schmidt et al. 2022b) As it almost always represents multi-step procedure (including aforementioned steps in various combinations) two terms have emerged to describe its organization: *analysis pipeline* or *bioimage workflow*.

Both terms emphasize the transformative aspect of the analysis, as a bioimage or its quantitative metadata (such as a region of interest) is being transformed in various steps to yield an analytical result. As a distinction, the term *pipeline* often is used in scenarios where data moves from an ingest point through the pipeline to become an output, whereas in a *workflow* data can also conditional split in its analysis path or bring back analytical insights onto an earlier step. (Miura et al., n.d.) This distinction however finds incohesive applications in the scientific literature (Paul-Gilloteaux et al. 2021).

Recently the term *bioimage workflow* has gained more traction (Paul-Gilloteaux et al. 2021) as it emphasizes this *non-linearity* of the analysis of biological images: analytical steps often diverge on an *on-channel* basis (e.g. apply filtering only on a foreground channel) or require user-input or iterative interaction with the data.

Workflows are often visualized as a connected graph of nodes, where nodes represent bioimage tasks, that get wired together by their respective input data and output data, providing an easy visual interpretation of the processes and steps involved. Paul-Gilloteaux expands on the idea of nodes, establishing them as *components* which can be part of a larger *collection.* A *component* in their terminology represents a dedicated bioimage algorithm that is part of a software package/tool, which in turn can contain multiple *components* (Paul-Gilloteaux et al. 2021). Their efforts to establish a common workflow ontology now resulted in a web-platform “The BioImage Informatics Index ”, where users can share their and find other workflows based on an established terminology that creates the Edam Bioimaging Ontology (Kalaš et al. 2019).

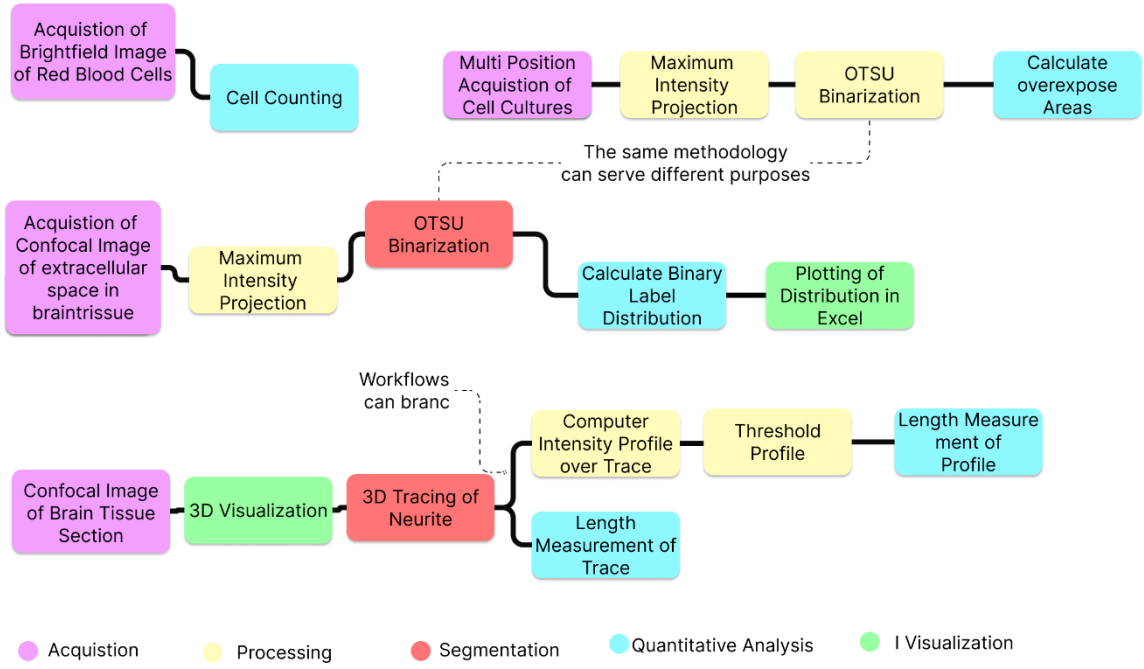

Figure 8 *Exemplary common bioimage workflows color coded by their conceptual steps.*

2.3.2.2.1 Level of Automation

It is to note that bioimage workflows today are by no means an automated process as a vast majority of its analytical steps are only done *partially automated* (Schmidt et al. 2022a). Some biological data analysis does even rely entirely on visual inspection (Schmidt et al. 2022a). Broadly the level of automation of the steps and the workflow can be categorized as follows:

Visual Inspection: This type of analysis step does not require any manipulation of or higher interaction with the data and the analytical insight is gained only by inspecting the sample (e.g. when counting cells under a light microscope).

Manual: Manual analysis encompasses processes that require a high involvement of human interaction with the data, and where automation has no measurable effect on the analysis step. This is found such as image annotation (ROI marking), interactive segmentation (e.g. delineating an axon), and extraction of features of interest.

Partially Automated: In partially automated analysis, automation is becoming more important, however, it often requires a circle of *Inspect and Adjust* following the automated process. Often this is found in settings of image processing; especially when it is combined with human insight to enable a segmentation. An example is image binarization that is performed by manipulating a threshold intensity level.

Fully Automated: In a fully automated setting no human involvement is needed. Often these are basic data management steps (conversion, projection) or on the opposite of the analytical spectrum highly involved computer vision steps where for example neural networks classify cells or tissues.

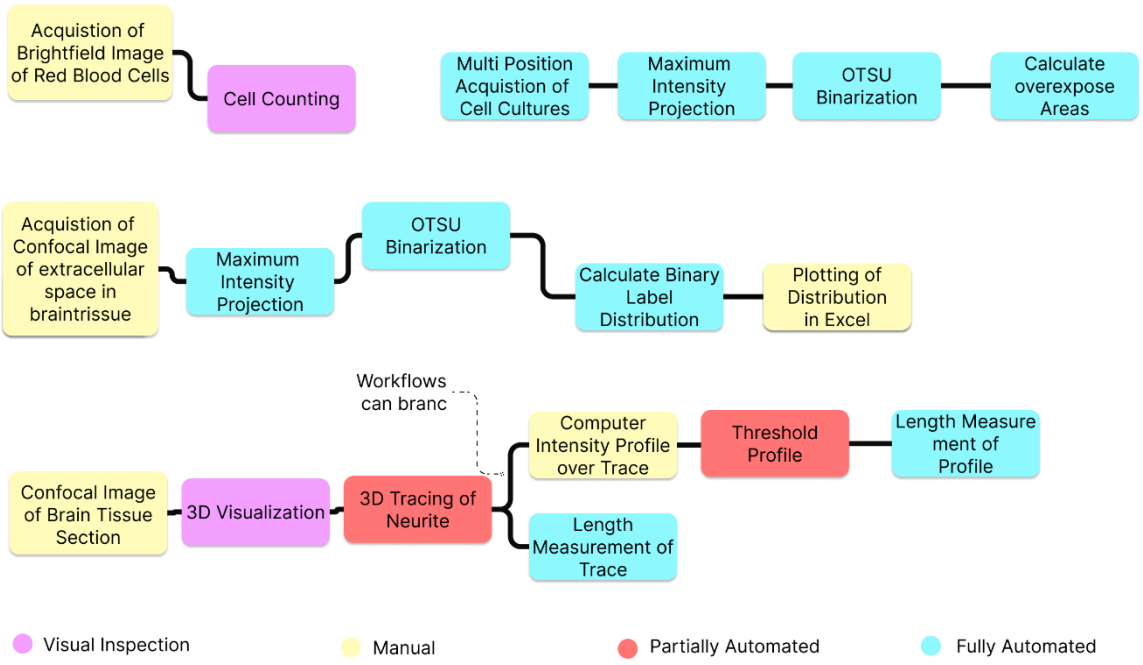

Figure 9 *The previous workflows color coded for the level of automation present at each step.*

2.3.3 The analytical toolbox

Bioimage workflows of today are facilitated by a range of software tools. Following the categories of a conceptual bioimage workflow, this section discusses the tools that are commonly utilized in modern workflows. The selection of tools in this section is based on their appearance in two of the most comprehensive recent surveys on the usage of modern bioimage analysis tools performed in 2020 (Jamali et al. 2021) and 2021 (Schmidt et al. 2022).

It is to note that most tools discussed in this section provide different modules that defy a definite categorization into one of these categories, so their organization should be rather regarded as what their primary *initial* focus has been, or for which step users today primarily choose the software solution.

2.3.3.1 Tools for Acquisition and Preprocessing

Every analysis starts with the acquisition of raw (image) biological data. Tools in this space are generally dedicated acquisition tools and are generally highly interactive and provide feedback to explore the biological sample in real-time. These tools seldom provide advanced processing but often come with a set of basic functionalities such as a projection algorithm, which also puts them often at the start of the analysis pipeline. Workflows that rely solely on Visual Inspection (10% of analysis workflows) also often end here (Sivagurunathan et al. 2023).

2.3.3.1.1 Commercial Microscopy Software

Most biological data is acquired on commercial microscopes (Sivagurunathan et al. 2023) using commercial control software. Beyond purely optical microscopes that are used for example for histology in hospitals, professional scientific microscopes are predominantly sold by four major companies: Nikon, Zeiss, Leica, and Olympus. Each provides its own custom microscope control software. Their software packages are specially catered to microscopy beginners and provide *user-friendly* access to basic functionality.

2.3.3.1.2 General Purpose Software

MetaMorph (Molecular Devices) and LabVIEW (National Instruments) (Bitter, Mohiuddin, and Nawrocki 2006) are the two biggest vendors of commercial non-brand specific microscope control software. MetaMorph provides a dedicated solution for multidimensional acquisition and visualization of microscopic data and is extensible via .NET (Bill, Wagner 2023) powered plugins. It also provides a graphical automation suite called “Journals” that allows for the composition of custom internal functions during an acquisition. LabVIEW on the other hand

is focused on a highly modular approach, and users can make use of its visual programming language “G” to build dedicated user interface fit for their specific microscopy needs. It should be noted here that Molecular Devices has as of June 2023 announced to discontinue their development for MetaMorph and will not issue new licenses.

2.3.3.1.3 Open-source microscopy

Open-source microscopy or Open-microscopy describes tools that allow general non-brand specific microscope control utilizing open-source libraries. (Hohlbein et al. 2022)Micro-Manager, a standalone app that fits as a plugin into the wider ImageJ ecosystem, is probably the most utilized solution in this space. With a variety of new software packages being published in the last years such as Pycro-Manager (Pinkard et al. 2021), python microscope (Susano Pinto et al. 2021) or ImSwitch (Moreno et al. 2021), open-source microscopy is currently experiencing a renaissance and there now exists fully open-source solutions to interact with microscope ranging from Scanning to Light-sheet microscopy. Due to their open nature, these tools generally allow for more low-level hardware access and are often installed in situations where new microscope techniques are pioneered. (Hohlbein et al. 2022)

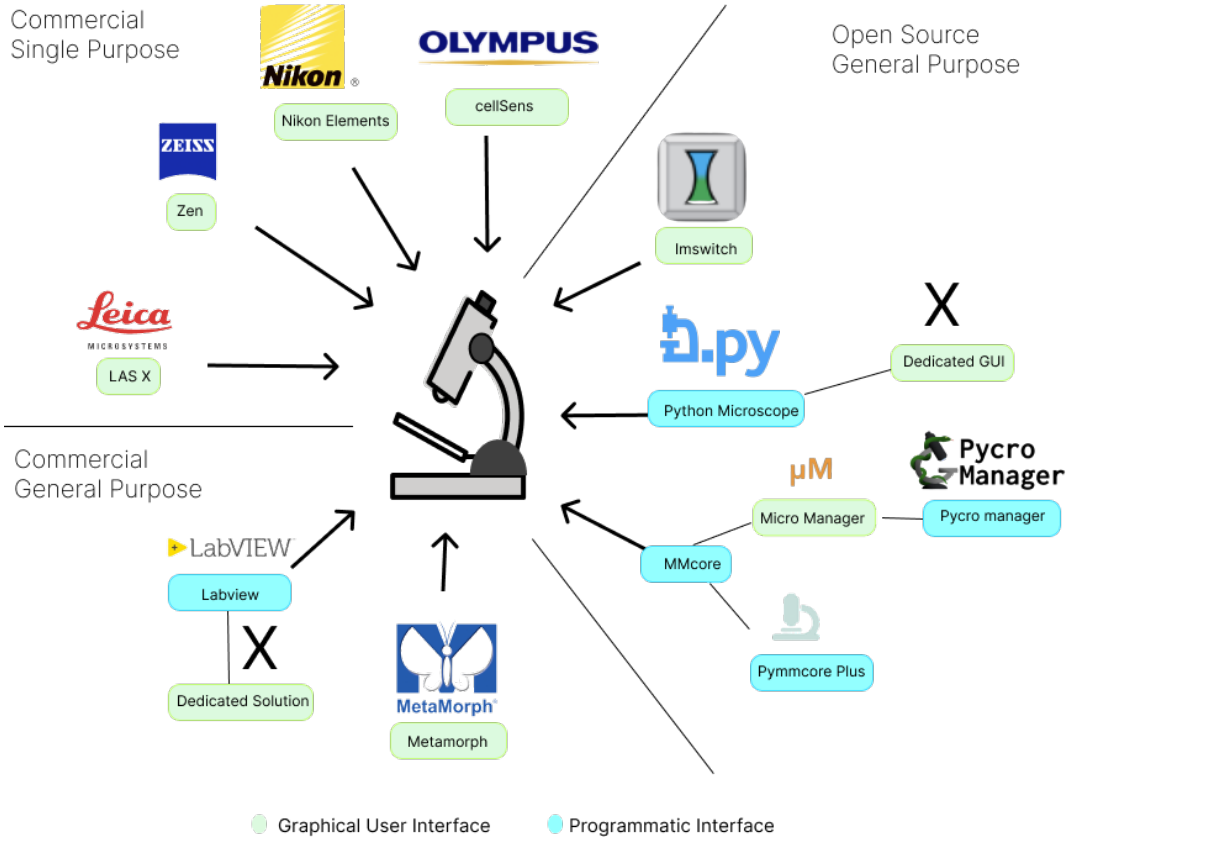

Figure 10 *Popular Microscopy control software labeled according to their license-status as well as their accessibility for non-expert users through graphical user-interfaces.*

2.3.3.2 Tools for Image Analysis

Tools in the image processing category are highly interactive and provide some form of visualization to facilitate the "*Inspect-Adjust*" circle.

2.3.3.2.1 ImageJ/FIJI (GUI)

ImageJ is by far the most utilized software package for image analysis in general and could easily be discussed in the other sections, as it sees a wide application across all bioimaging steps. Developed by Mathews Rasband in 1997 (named NIH Image at this point), it represented one of the first solutions to point and click image analysis and has continued to involve over the past decades. (Schneider, Rasband, and Eliceiri 2012)

ImageJ comes with a graphical user interface that allows easy search and menu access to all registered commands and opens images and their processed results in separate windows to allow for easy comparison between the raw and processed data. A testament to its usability, the core interface of ImageJ has almost not changed in the past 25 years (Schneider, Rasband, and Eliceiri 2012) and most biologists today feel at least somewhat familiar with it. Additionally, it stands as the entry point technology for most image analysis tutorials and courses available (Jamali et al. 2021).

Packaged in its major distribution FIJI, a bundled "*batteries-included*" version of ImageJ with various plugins preinstalled, its citations currently exceed more than 6000 a year. A number that probably grossly misrepresents its actual usage, given that Google Scholar search for ("*ImageJ OR Fiji/ImageJ*") in 2022 alone yields more than 40000 results (retrieved June 2023).

ImageJ's popularity established its status as the *gold-standard* framework for image analysis and it stands as the de-facto standard platform for integrating other image analysis tools. (Schneider, Rasband, and Eliceiri 2012)This is arguable due to its extensive plugin ecosystem and its macro extensions system, providing two major ways of extending its functionality, which fit the needs of modern bioimage scientists:

***Macros*:**
Macros represent sharable pieces of code that are interpreted at runtime in the ImageJ environment and are often used for project specific routine automations. Due to quality-of-life features such as the *Macro Recorder*, the *Macro Code Editor*, and ImageJ's ability to automatically generate user interface elements for macros, they provide a small barrier of entry to let user automate their respective analysis and create new bioimage workflows.

Advanced processing features such as pixel-based transforms are also possible with macros but are often hindered by their interpreted nature yielding very not ideal performance when dealing with large datasets.

***Plugins*:**

Plugins are Java-based modules that can be installed with minimal effort through one of the update sites of ImageJ or through drag-n-drop installation into the plugins folder in the ImageJ directory. They often represent more generic solutions to image analysis (like new algorithms, or dedicated user interface (UI) elements) and can be easily integrated in the wider ImageJ ecosystem.

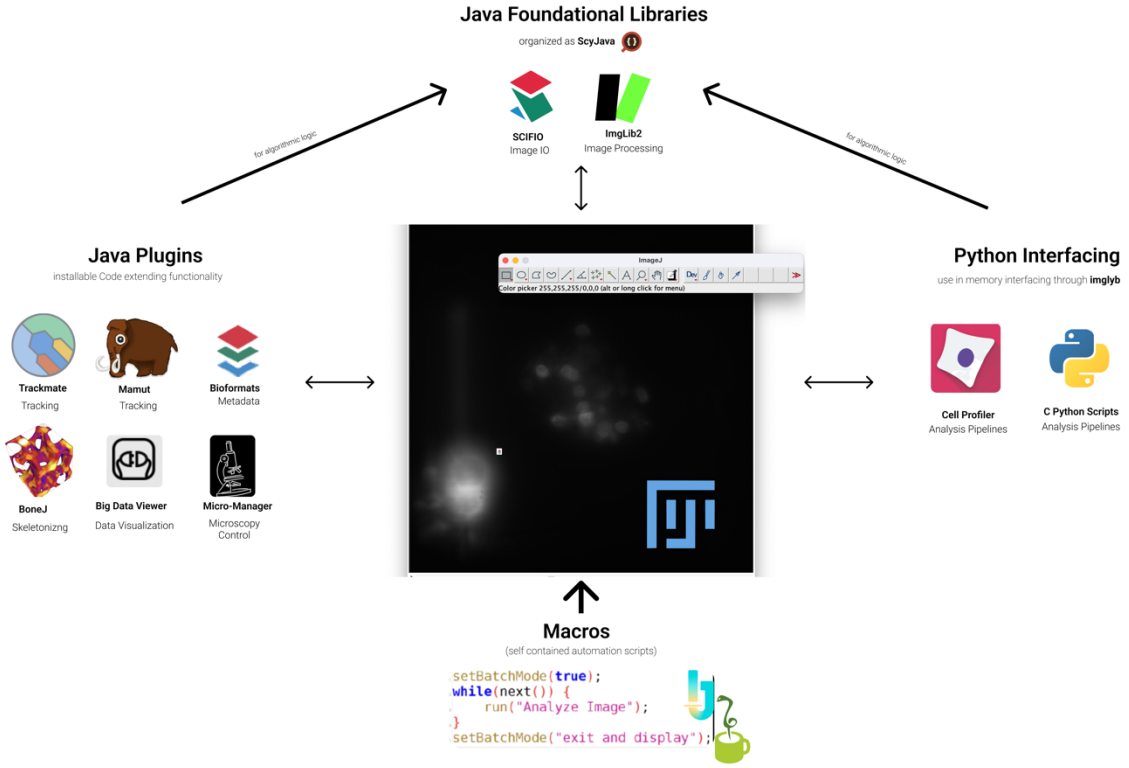

Figure 11 *The ImageJ ecosystem with its major axis of expandability.*

ImageJ stands as one of the most actively maintained projects in open-source image analysis. It provides a foundational framework for other software projects such as Trackmate (Tinevez et al. 2017) or SpineJ (Levet et al. 2020), that offer additional features on top of ImageJ that are not included in the standard distribution such as cell tracking or dendritic spine segmentation. As dedicated Java plugins, they package their own dependencies and are even able to integrate some deep learning algorithms e.g. CellPose (Stringer et al. 2021) into the ImageJ ecosystem.

2.3.3.2.2 Processing Scripts

Ranking as the second most utilized methodology after ImageJ are script-based analysis tools (including the later mentioned analysis and quantitative analysis scripts). Indeed approximately 3 out of 4 participants of the 2023 survey reported using computational libraries and scripts for analyzing their images at least sometimes (Sivagurunathan et al. 2023).

Even though there are solutions of script-based analysis for compiled languages a vast majority of these scripts are written in interpreted languages such as Python (Van Rossum and Drake Jr, 1995) and MATLAB (Inc, 2022). In general usage, MATLAB supersedes Python in the latest survey amongst non-expert-users, while Python is more frequently used in the expert space (Schmidt et al. 2022b).

While MATLAB does not provide a package manager to install additional packages and comes with a lot of its functionality built in, or distributed by software modules called Addons (“Mattlab Add-Ons", 2023), Pythons popularity is driven by its extensibility, as it boasts a wide ecosystem of scientific libraries that users interact with (Sivagurunathan et al. 2023).

Figure 12 *Python ecosystem of scientific libraries, ordered by core libraries (center) and libraries that are most utilized in the respective image analysis steps.*

2.3.3.3 Tools for Segmentation

Segmentation and its mother field computer vision today probably represent the most active field of continuous change and development in the analytical toolbox. (Meijering 2020)

2.3.3.3.1 Imaris

Imaris is a software solution developed by Bitplane and designed for 3D and 4D image visualization and analysis(“Imaris”, Bitplane). Core features of Imaris are its (proprietary) computers visions algorithms that allow for 3D segmentation of microscopy data and its ability to track movement and changes over time, useful for dynamic processes like cell migration or division. Imaris can quantify various parameters from segmented and tracked structures for statistical analysis, all within an *interactive* user experience. Imaris currently represents the second most utilized analysis software, which could also be partly due to the selection bias of mainly neuroscientists and cell biologists in the original survey (Schmidt et al. 2022b). Due to its high price-tag is often found in imaging facilities where users share a common workspace.

2.3.3.3.2 Ilastik

Ilastik (Berg et al. 2019) is an open-source python-based software with a graphical user interface, designed for the interactive and automated analysis of large and complex biological images. It integrates classic machine learning algorithms, such as random forests, but also dedicated neuronal nets such as Stardist (Weigert et al. 2020), to facilitate image processing steps including pixel classification, object classification, but also quantitative analysis steps such as cell tracking.

2.3.3.3.3 Machine Learning Scripts

An increasing set of Machine Learning applications and scripts provide another core utility in the segmentation of biological data, with an approximately 2-fold increase in the peer-reviewed papers published with the terms ‘deep/machine learning/artificial intelligence’ (Fig S8) during the period of 2020 to 2022 (Sivagurunathan et al. 2023).

With basic understanding of programming becoming an essential skill in a lot of scientific disciplines, exposure to code has become more widespread. Exemplified by the popularity of projects like ZeroCost4DL (von Chamier et al. 2021) interactive and well documented code has become a popular approach to share machine learning scripts and applications with a wider public.

Most of these scripts run within interactive environments like Jupyter Notebooks (Kluyver et al. 2016) which has become the de facto standard to share machine analysis scripts in Python. Tools often bundle code together with plots and tables, making them less "*scary*" for newcomers and entry level programmers. (Meijering 2020)

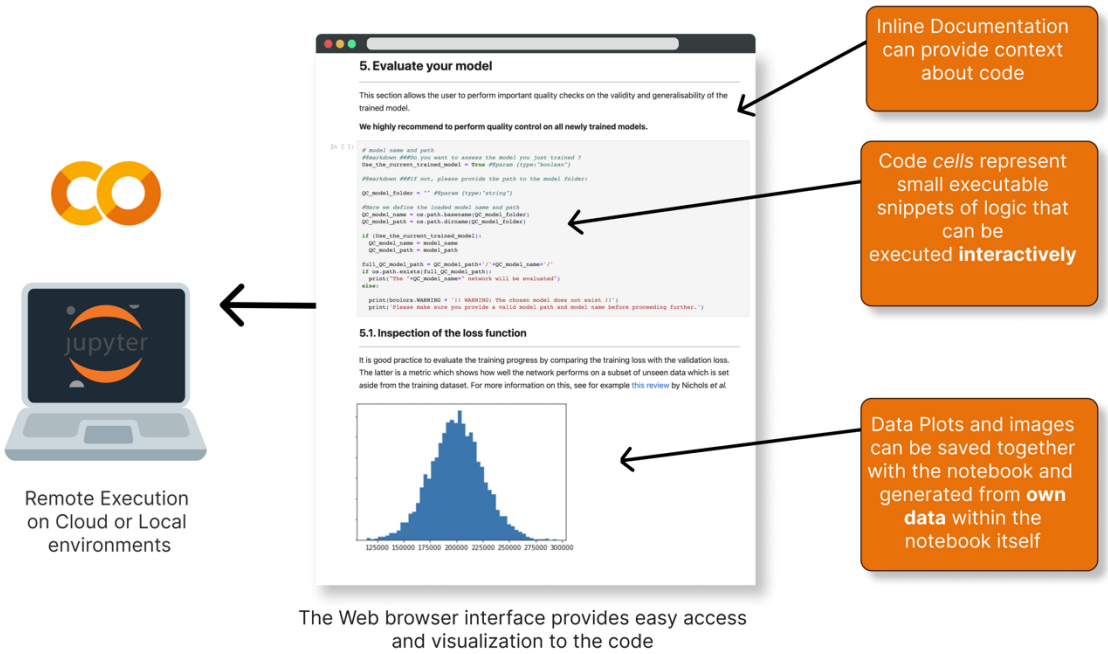

Figure 13 *The Jupyter notebook environment and its core features of interactive code cells that are inline combined with documentation and plots.*

2.3.3.4 Tools for Quantitative Analysis

Tools that are used for quantitative analysis, focus on integrating the information gained from previous steps and extracting statistically interpretable meaning from them. As such, most aforementioned tools provide modules for quantitative analysis.

Dedicated tools in this space often provide features to deal with long traces of tabular data, describing ROIs items or traced intensity profiles. While the analysis of this data happens primarily automatically, some basic functionality to inspect the data and to validate the methodology through plotting is often provided.

2.3.3.4.1 Cell Profiler

CellProfiler (Carpenter et al. 2006) is an open-source software for designing quantitative analysis pipelines of biological images. It comes with a comprehensive graphical user interface to design workflows with a wide range of processing and segmentation algorithms out of the box.

A core feature of Cell profiler, that warrants its categorization as a quantitative analysis tool, is its inclusion of a wide battery of quantitative analysis algorithms to enable the measurement of phenotypes from images automatically. Here its functionality often complements ImageJ processing capabilities, which can be integrated into its pipelines. (Dobson Ellen et al. 2021). It can analyze and provide detailed morphological measurements of individual cells in large image collections, which can be used in a wide range of applications, including profiling genetic perturbations, drug responses, and high-throughput, high-content screening assays (Kamentsky et al. 2011).

2.3.3.4.2 Excel

While often discarded as a tool for statistical analysis and plotting, data sheet oriented software such as Excel (Microsoft Excel, Microsoft Cooperation) is a commonly used tool for quantitative analysis of 1D tabular signals, such as histograms or line plots previously extracted from a bioimage.

2.3.3.4.3 Quantitative Analysis Scripts

Quantitative Analysis Scripts provide the last pillar for script-based analysis. While they are often underpinned by the same software stack of MATLAB and Python, quantitative analysis scripts are also often based on programming languages that focus on gaining and interpreting the statistical insights, such as the R programming language (“R: The R Project” , 2023). Comprehensive software repositories like Bioconductor (Gentleman et al. 2004) in the R ecosystem, provide dedicated solutions like EBImage for image analysis and processing (Oleś et al. 2023).

2.3.3.5 Tools for Visualization

2.3.3.5.1 Napari

Napari (Sofroniew et al. 2022) is an open-source, multi-dimensional image viewer for Python. It has gained traction in the expert bioimage space especially because of its ability to visualize large scale, multi-dimensional images, interactively rendering 3D data with GPU acceleration. (“3rd Napari Survey Report” , Napari Developers, 2022)

A native Python solution, Napari is often integrated into Python-based analysis scripts, where analytical results need to be displayed interactively to inspect. To this end, Napari provides different types of layers for different types of data including images, points, lines, shapes. This

layered approach to data is useful for overlaying annotations or segmentations on top of image data.

A recent addition to the bioimage world, Napari's user group is growing rapidly and having made no appearance in the 2021 survey, is already surveyed as the 10 most utilized software for image analysis. Recently efforts have been made to establish Napari as a wider Python-based analysis hub ("Npe2", 2023). It supports a plugin ecosystem, allowing users to extend its functionality for their specific use cases, and comes with a central repository to make these plugins easily discoverable.

2.3.3.5.2 Web-Based Tools

Web-based visualization software solutions like Viv (Manz et al. 2022), webKnossos (Boergens et al. 2017) or Google's Neuroglancer ("Neuroglancer",2023) provide new means of visualizing datasets in the browser without prior installation. They serve as fast visualization solutions for publicly available data, facilitating a *quick look* on the dataset. (Manz et al. 2022) and are often integrated in larger ecosystems of web-based analysis: Viv provides the visualization for Vitessce (M. S. Keller et al. 2021), a framework for multi-modal and spatially resolve single-cell data, and webKnossos, (the platform) integrates features for the annotation of large scale connectomics data. Almost 30 per cent of users do currently use web-based tools for their analysis (Schmidt et al. 2022b).

2.4 The State of the Art

Having gained an understanding of the modern bioimage analysis and their tools, this section will now introduce some exemplary analysis workflows of the bioimage analysis today, which could potentially shape the landscape of bioimage analysis of tomorrow. These workflows represent state-of-the-art image analysis tools, that exemplify and stand for some wider trends in the image analysis world.

2.4.1 More Data faster

Even though most of today's analysis is done on 2D, our biological systems are mostly 3D. So, it comes as no surprise that we see an undeniable trend to image our biology in 3D. High-resolution 3D imaging was long powered by confocal microscopy. However, its nature of being a scanning technique did not permit to capture the fourth component of a dynamic biological system at highest fidelity: time. Additionally, as it is a highly invasive technique using visible light, it becomes toxic to the living. While spinning disk confocal microscopy (Petráň et al. 1968) allowed partially to overcome the speed constrain, and multi-photon microscopy (Denk, Strickler, and Webb 1990) drastically reduced the toxicity issue in depth, aspects of this problem long persisted.

Today breakthrough technologies like Light-Sheet Microscopy (Huisken et al. 2004)("Method of the Year 2014" 2015), break this time and toxicity limitation, and have allowed the scientific community to witness biological results at ever increasing temporal resolutions and uncover for example cell movement patterns in developing zebrafish (P. J. Keller et al. 2008). As scientist push the frontiers, for example in calcium imaging of neuronal populations (P. J. Keller, Ahrens, and Freeman 2015), when monitoring larger and larger volumes neuronal in less and less time, they are looking more and more towards advanced image processing and visualization techniques to support their analysis.

For example, even the best detectors are physically limited by noise, especially when capturing data fast, with limited number of photons. Denoising the low signal-to-noise ratio data has therefore become a standard in modern analysis workflows (Li et al. 2021), powered predominantly by classical model-based approaches, that, for example, exploit repetitive signal patterns (Kervrann and Boulanger 2006). Recently, also deep learning algorithms such as CARE (Weigert et al. 2018), a supervised general purpose solution for image denoising, or DeepCAD (Li et al. 2021), an unsupervised solution for denoising of calcium imaging data, showed impressive and very promising results.

Now that the size of bioimage data datasets easily exceed the Gigabytes or sometimes even Terabytes, but computer RAM memory remains in the gigabytes, new forms of visualization and data management are being investigated, and find solutions in tools like Napari where *lazy*-loading capabilities have been developed (Yang et al. 2022). A long existing and familiar practice in large-scale map visualizations ("Google Maps", Alphabet), *lazy* loading refers to a programming paradigm where the actual retrieval of a datapoint becomes delayed until the exact moment when it is needed. Often this technique gets combined with *out-of-core* techniques, where only a chunk (i.e., just one timepoint in a 5,000 timepoints long image series) of the data gets loaded into memory and the rest of the data stays *out-of-core*, that is not in RAM but on the hard disk.

The size of the new datasets has led to the integration and the new development of software packages like Zarr (Miles et al. 2020), HDF5 (The HDF Group 2000) and dask (Rocklin 2015), which provide chunked storage and lazy computation for multidimensional data respectively. Accounting for the trend to more data, these new techniques are now shaping to be the foundation of the next-generation image storage formats (Moore et al. 2023).

2.4.2 Automated Analysis

Even though a lot of the analysis is still relying on human annotation and visual inspection (Schmidt et al. 2022b), there is an undeniable trend towards automated and machine-driven analysis. The reasons for this are plentiful, but often quoted arguments mention reproducibility and reduction of human bias. One continued field of innovation in automation, is a field where automation has always played a major part: disciplines that process high data volumes.

High-content screening is one typical example here that finds its application in streamlining drug discovery processes. It employs automated microscopy and analysis to collect quantitative data from a large collection of biological samples, that are then analyzed to gather detailed information about the samples in their experimental condition (Shariff et al. 2010). This can include assessing the effect of the administered drug component on cell morphology or distribution.

Historically, a field of mainly 2D cell culture analysis, more recently we see efforts to bring high-content screening applications to the third dimensions (Beghin et al. 2022) to more closely monitor and screen higher-order biological systems such as spheroids or organoids

("Method of the Year 2017", 2018). Through the assessment of organization of these *in-vitro* 3D cell aggregates scientists hope to more closely study the drug's effects on the histological organizations of organs or cancerous tumors (Krausz et al. 2013), which could in turn help to reduce the 90% failure rate of pre-clinical drugs (Sun et al. 2022).

In order to facilitate the involved analysis pipelines, these applications now heavily rely on deep learning for cell segmentation (Weigert et al. 2020) or for detecting specific cellular events, such as progression through the cell cycle (Kusumoto et al. 2021). They also use and investigate more integrated data-management and visualization solutions, such a CellProfiler Analyst (T. R. Jones et al. 2008).

Faced with massive data loads some of these automated workflows require new forms of parallel computing that bring computation from single computer to multiple machines or even computation clusters (Teodoro et al. 2013).

2.4.3 Smart microscopy

Breaking with the concept of the analysis workflow as a unidirectional stream, from acquisition to insights, a set of new techniques, termed "Smart Microscopy", have recently gained traction, as they enable the loop back of analytical insight onto the ongoing acquisition (Strack 2020) Through them, scientists can overcome physical or experimental tradeoffs in the acquisition process, for example by selectively informing and guiding the microscope to monitor smaller regions or by changing the framerate mid-acquisition.

As one example in this space stands *AutoPilot*, a light-sheet microscopy framework that continuously adapted to measured image quality changes while scanning a Drosophila melanogaster specimen by adjusting illumination angles to the detection path (Royer et al. 2016). Another example, this time in single molecule localization microscopy, could show how smart adjustment to the laser intensity based on the analytically reported molecule density could optimize the acquisition speed and efficiency (Kechkar et al. 2013).

Smart microscopy also enables more biologically informed workflows, that triggered by some biological events can change acquisition parameters to capture the information of interest more closely. This *event-driven* microscopy has seen recent applications for example in event triggered STED microscopy, where it helped to overcome its limitations of temporal resolution, by only selectively monitoring a region of interest, if a calcium event was detected (Alvelid et al. 2022). It also showed its potential when using deep learning algorithms to detect divisions

events in an experimental study of mitochondrial proliferation, facilitating a more *content enriched acquisition* when subsequently adjusting acquisition speed (Mahecic et al. 2022).

These workflows are bringing the analysis ever closer to the microscope, necessitating real-time performances of the algorithms and the capacity to take control over the microscope. They also increasingly interface with robotic devices that can control experimental conditions such as fixation during the acquisition or drug delivery (Almada et al. 2019).

2.5 Challenges of running today's workflows

These exemplary workflows make use of the state-of the art methodology and are pushing the limits of analytical methods. To have a true impact beyond the original experiments they and other methodologies however must be applied and adapted to new problem sets.

This section will now go into the challenges of adapting and orchestrating modern bioimage analysis, its workflows, and the reasons why they are, at this point, still limited to a set of experts.

2.5.1 **Fragmentation** of Tools and Hardware

When inspecting today's image analysis workflows closely one unifying feature becomes evident: they include *multiple* analytical steps that are handled by *dedicated* tools tailored to their solutions. Be it standalone deep learning algorithms for denoising in calcium-imaging workflows, highly specialized 3D visualization tools like Napari, or dedicated acquisition tools in smart-microscopy. The state-of-the-art of the discipline continues to specialize and focus on more single-purpose tools than general solutions, and solutions for specific problems are more often implemented and distributed as separated tools than integrated into a wider solution such as ImageJ.

While ImageJ is still predominantly the go to solution for established and beginner image analysis, this set of new tools are emerging and gaining traction amongst the expert space because they overcome long standing challenges in the ImageJ ecosystem. Solutions like the aforementioned Napari (Sofroniew et al. 2022), but also Cell Profiler (Carpenter et al. 2006) or Ilastik (Berg et al. 2019) provide unique features that are hard to find in ImageJ. Another accelerator for this transition to dedicated expert tools is that machine learning methods are currently the most actively developed subfield of image analysis and are developed in Python rather than Java (Meijering 2020).

Even though there are efforts to bridge some of the technology back into the Java-Space, such as DeepImageJ (Gómez-de-Mariscal et al. 2021) for model *inference*, general adoption of these methods still remains low, and it is questionable if complex state-of the-art AI models will find easy translation especially when relying on Python based pre- and post-processing for training.

Illustrating the potential long-lasting effect of this transition also in the non-expert space is the fact that, both Napari and CellProfiler spike almost the same interest as ImageJ for learning material in the non-expert demographic (Sivagurunathan et al. 2023).

This fragmentation of the tools space is further accentuated by a need for more specialized hardware, such that in modern workflows tools do no longer share the same computational environment, but rather are distributed amongst different hardware: Deep-Learning tasks profit from a powerful GPU, while big-volume processing is parallelizable on a computational cluster.

Both trends render modern bioimage analysis workflows more and more distributed in nature. Instead of consolidating solutions into one general solution, tools become more scattered and users wanting to orchestrate these new tools into bioimage workflows must face the new challenges of dealing with multiple tools.

2.5.2 **Interoperability:** The need for patchwork

In the absence of an overarching ecosystem such as ImageJ to ensure the interoperability of the methods by providing data primitives for images, region-of-interests, tables and more, modern workflows need to find new means of ensuring the interoperability of the participating tools in a workflow.

Expert workflows often ensure this by relying on establishing a complex patchwork of conversion helpers that ensure the compatibility of outputs and inputs in a workflow. This can include for example the conversion of bioimage data into the corresponding file format, that can be read by each participating app, as well as little helper scripts that convert region-of-interest formats (e.g., from Java to Napari).
Even when these scripts are shared, they are not easily translatable by non-experts as these scripts are often specifically tailored to the exact tools and datatypes of the microscopes that were used in the original methods. When transitioning to other datatypes they can cause cryptic bugs for the new user or even more *viciously* can lead to dangerous *silent* conversion errors, where image information is cut due to a faulty data-type conversion introducing bias into the analysis (e.g. when converting a 16bit image to an 8bit image).

Accounting for the need of more interoperable file-formats, open file formats have been developed, and are getting more wildly adapted into the tools. OME-Tiff (Besson et al. 2019)

is probably the most widely used format here and extends the TIFF format with the OME Metadata, to propagate also the complete metadata through the analysis steps.

However, with the previously discussed OME-NGFF (Moore et al. 2023) a new *competing* format for the big-data era has entered this space, potentially furthering the interoperability problematic.

2.5.3 **Orchestration:** The difficulty of bridging applications and devices (in real-time)

The feasibility of an analytical workflow that includes multiple tools, depends on how easy it is to orchestrate and manage the participating steps that are involved in a timely and coordinated manner. This necessitates being able to *move* interoperable data from one tool to another tool efficiently and *schedule* tasks on the right tool at the right time.

In a framework like ImageJ, where tools can share the same memory, this task can be handled through the framework and easily orchestrated through click-based transformations (Process1 -> Menu -> Process2) of the original data or automatically through simple automation scripts. In a fragmented multi-tool environment however, no overarching instance exists that can be used to *schedule* and *move* data, and these management steps must be performed manually or through dedicated automation scrips on the operating system level or, when apps span multiple computer environments, on the network-level.

Both automatic approaches, however, are often out of reach for beginners and even when the potential users are proficient enough, they require the orchestrated tools to be written so that their functionality can be called from the outside, say through a command line interface. For the most part, this orchestration step therefore happens manually and involves *tedious* data management steps of moving data around and manually starting and closing applications.

To minimize the impact of the data management in these involved workflows, modern analytical steps are often done in *batches*. This *batch analysis* however does not only increase the time to have feedback on the biological data and hinders the iteration on finding the right parameters for each analysis step, but inherently conflicts with some modern bioimage workflows as illustrated in the example of Smart Microscopy.

Workflows in this space are no longer unidirectional analysis flows and require the wiring back of analytical insights *during* the acquisition, something that orchestration needs to account for: a datum needs to be processed directly after it being produced in a *stream*-like manner.

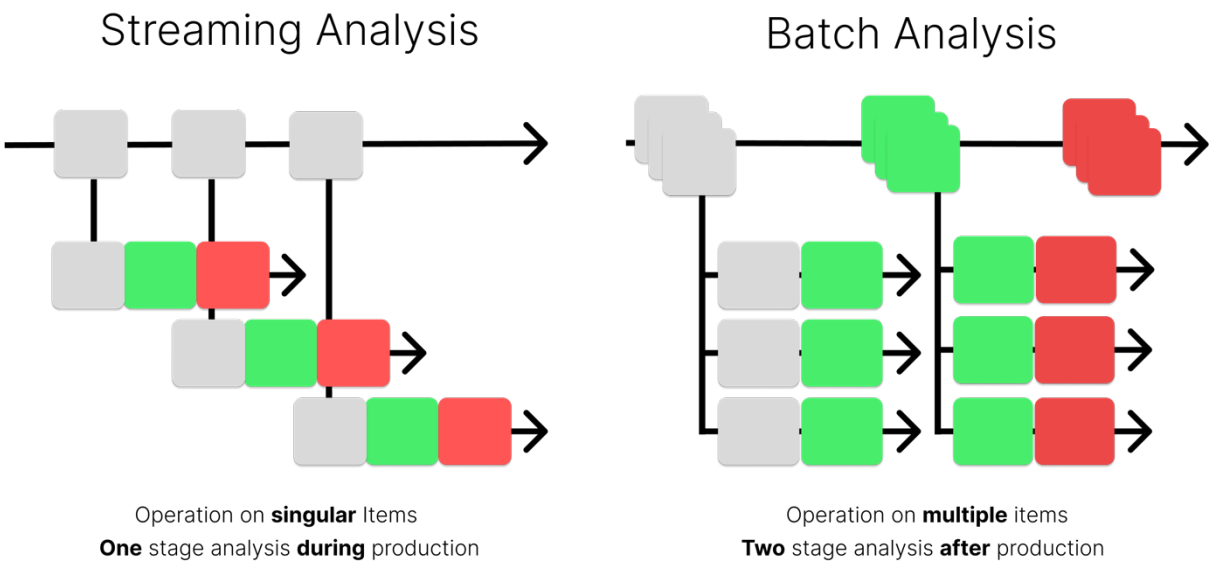

Figure 14 Streaming Analysis vs Batch Analysis

Streams are a more inclusive idea about data than *batches* and take into account the aspect of data production. They can be generally divided into two categories: *unbounded* and *bounded* stream. An *unbounded* stream has no defined start or end as data items are constantly arriving and need to be analyzed: this scenario is the mentioned real-time scenario where data needs to be analyzed as it is being produces by the microscope in an *ongoing* acquisition. A *bounded* stream on the other hand, such as a *finished* acquisition, has a defined start and end and analysis can be run on the complete stream. Batch analysis is only feasible for bounded streams or when unbounded streams become *windowed* (buffering for example 5 items) into a *bounded* stream.

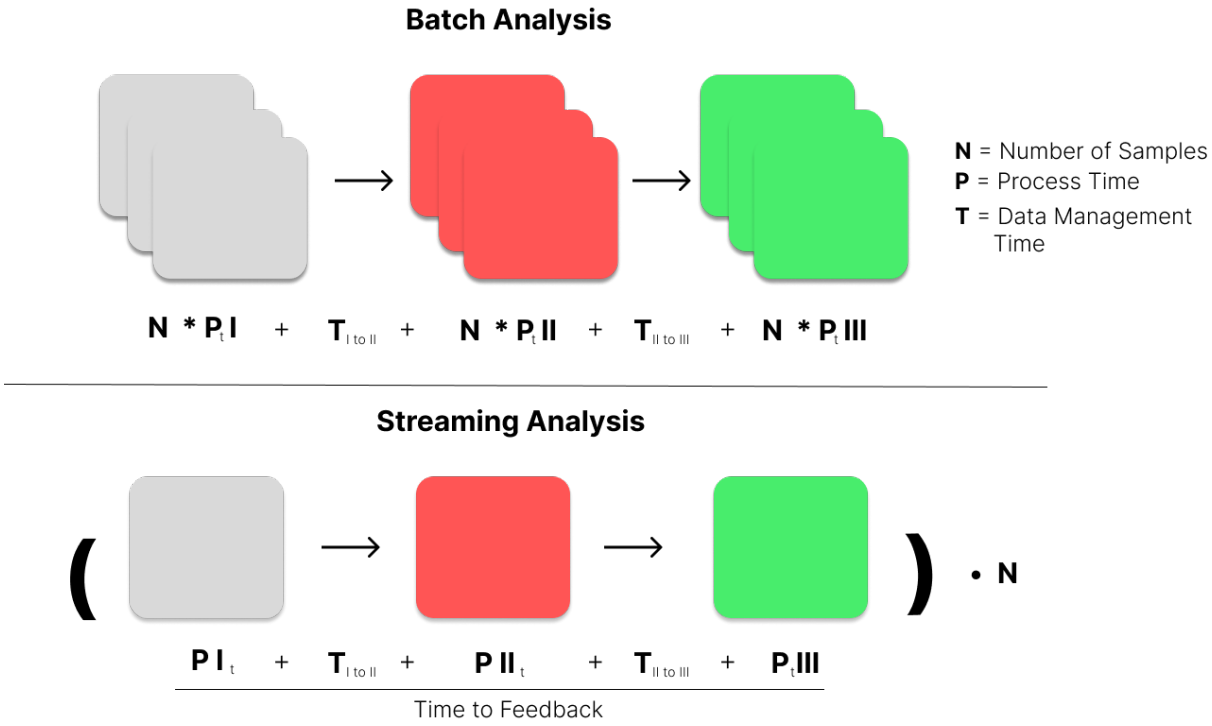

Figure 15 *Streaming analysis vs batch analysis compared to their time to feedback vs their total time needed to process. In batch analysis the time to feedback is equivalent with the total time, in a streaming analysis setting the time to Feedback is reduced as indicated.*

Albeit a common theme in modern big-data analytics is the cloud, where real-time analytics of customer data have become paramount, streaming analysis is rarely adopted in bioimage analysis. This finds its explanation in the fact that the orchestration of real-time tasks comes with additional pitfalls, such as the necessity to handle analytical bottlenecks (a situation where one task is taking considerably longer than the others and causes the analysis flow to perform suboptimal or even to fail).

Modern Smart Workflows in the expert space therefore need to often account for the complexity of integrating real-time information back onto the microscope by tightly integrating all necessary analytical steps on the microscope. They implement custom software *schedulers* in their tool packages that utilize *concurrent programming* patterns account for the management of timely events inside the acquisition workflow. This style of programming relies strongly on threaded and parallel execution and is not part of the standard bioimage analysists toolbox, which currently restricts this type of workflows to advanced programming users (Levet et al. 2021).

Accounting for these limitations, solutions like CyberSco.PY (Chiron et al. 2022) were able to create usable non-programming interfaces to smart workflows by allowing users to graphical embed conditional event paths in the acquisition logic. However, their approach still requires

tightly integrating the methodology in the microscope and falls short when analytical insight should inform decisions on connected robotics, such as illustrated in the discussed NanoJFluidics strategy (Almada et al. 2019). Thus, modern real-time, multi-hardware, multi-tool workflow orchestration is currently reserved for experts and specific applications.

2.5.4 **Usability:** The challenge of implementing GUIs

One core feature that developers can rely on when developing a tool inside ImageJ ecosystems is its integration of simple programming interfaces to easily create dedicated user interfaces, for the non-expert audiences of a tool. This integration enables an easy developer experience for a key contributor to the usability of their tool: an easy-to-use graphical user interface (GUI).

Well-designed interfaces can help to provide simple solutions to complex problems and they represent the number one request of users to tool creators, in order to make their tools more successful (Jamali et al. 2021). This is also a shared believe with developers that would agree that providing GUI would advance the usability of their software (Jamali et al. 2021).

As more and more developers choose to develop outside of this framework however, they need to provide their own solution to the graphical user interface. Even though modern libraries interfacing for example with the Qt Framework (“Qt” , The Qt Company) have made the process of creating a usable UI a lot easier, it remains a challenging task, that is often outside of the experience comfort-zone of the standard bioimage developer (Cardona and Tomancak 2012). Very rarely are developers familiar with concepts of event loops or threaded execution, that are necessary for making responsive and interactive GUIs.

Additionally, ministering to the non-expert group is often an *afterthought* in the development of a new methodology, and developers underestimate the time involvement in maintaining a quality graphical application. In the fast-moving field of science, this type of commitment however is not universally rewarded. This holds especially true in teams that are not primarily focused on method development, but rather on finding new scientific insights. Thus developers typically will not follow the professional software engineering practices necessary for production-grade software. (Levet et al. 2021).

Recently new developer tools like “magicgui” (“Magicgui”) are trying to tackle this gap in easy GUI tools, by auto-generating user interface from code annotations, and help to bridge code

without a lot of UI boilerplate to the users. They remain however tightly specific to the Python and QT Ecosystem and overarching usability facilitating solutions are still lacking.

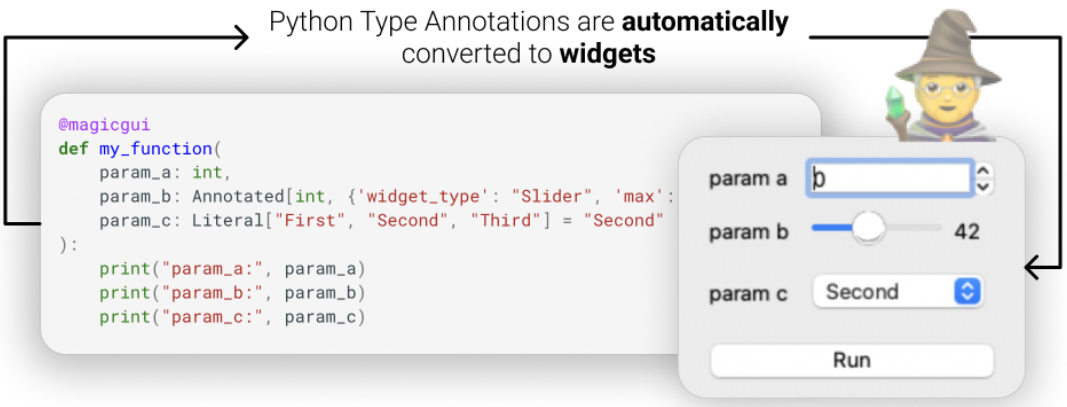

Figure 16 *The magicgui type annotation "magic"*

2.5.5 **Portability**: The difficulty of installing apps.

Another core contingent problem of composing multi-tool, multi-computer workflows is that of having to install the required tools on their respective hardware and *porting* the working solution off the computer of the developer to the new computer of the user. In the absence of a universal plugin framework like ImageJ, which allows users to simple install analytical plugins through drag-n-drop into the filesystem, tool developers need to come up with their own bundling solution to package the newly developed tool and its dependencies into an installable component.

This process, which does not only include bundling the code but potentially the interpreter and external hardware-specific dependencies, represents a long-standing challenge in many programming languages (Lex Fridman 2022). In the ImageJ and Java space this problem was solved with well-established tools for app packaging that bundle source code together with its execution runtime the Java Runtime Environment (Arnold, Gosling, and Holmes 2005).

Python, the major language for new tool development, lacks a cohesive industry standard by design (Lex Fridman 2022) solutions and through its nature as a *glue* language between outside dependencies such as C-bindings for NumPy or Numcodecs, or CUDA for accelerated computing on the GPU, requires high levels of tinkering to package them reliably (“Python Packaging User Guide”, 2023).

This process gets even more complexified as new hardware architectures, such as the new Apple Silicon or RISC-V CPUs find wider application. Even established and well-maintained tools like Napari, struggle with this process currently, and as of June 2023 the bundled stable version of Napari neither works on a new M2 Mac, nor reliably on the GPU of an Ubuntu 22.04 System.

Multiplying the challenge of deploying only one tool in a workflow by the number of included tools in some workflow, the true scope of the problematic becomes evident. These challenges of portability lead to a wider trend where users interested in a new methodology often are lead to multi-month or multi-year journey of adapting the method and sometimes even reproducing the original result (Laine et al. 2021).

To overcome this issue that plagues not only the field of bioimage analysis but stands as a more general problem in software engineering, some tool developers are looking more and more into *containerizing* their applications. Containerized applications represent applications that live in a self-contained and sandboxed space, on the host operating system that behaves like a virtual machine. Contrary to running an application in virtual machines however, they are generally smaller, less resource intensive to maintain and can be more easily managed (da Veiga Leprevost et al. 2017).

Sparked in the early 2010s through Docker (“Docker” 2022), containerized applications have since seen rapid adaption in the software development space and are topping the list of “most important technology” in surveys on StackOverflow for years (“Stack Overflow Developer Survey 2023”, StackOverflow).

In the bioimage space especially the *build-once-deploy-anywhere* paradigm of containers has seen a lot of interest: developing a containerized application includes a build-step, that bundles all dependencies of an application inside the container, freezing them in time. This means that this contained application is almost guaranteed to work on other machines, including also other platforms (e.g. running a container on Mac, Windows and Linux.

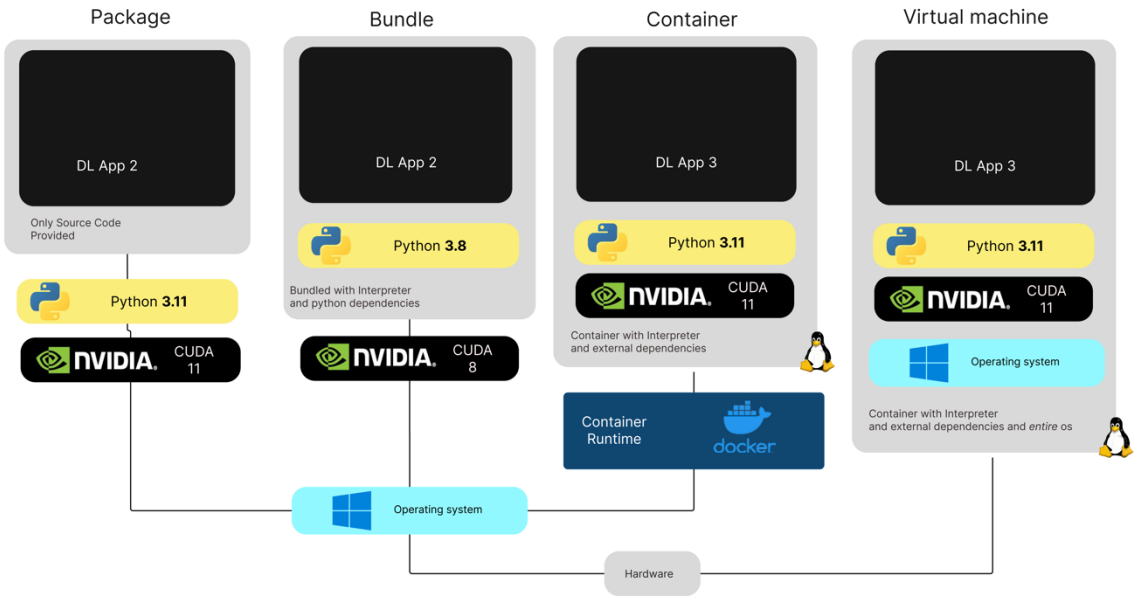

Figure 17 *Comparison of the strategies of installation of a software project, all outside dependencies provide potential sources of interoperability issues as they depend on the adequate environment control of the user.*

There now exists a variety of software projects like Stardist (Weigert et al. 2020) that provide prebuilt containers to facilitate easy installation of their packages, and projects like Bio Containers (da Veiga Leprevost et al. 2017) are trying to further standardize their creation, making them more applicable to the bioimage analysis world.

While this transition to containerized applications will surely have a great impact on the portability of analysis tools, they do not represent a one-size fits all solutions. Highly interactive software such as GUI tools can as of now (June 2023) not be run in containers, and tools that need low-level hardware access, such as acquisition software, are generally not compatible with the sandbox concept of containers.

2.5.6 **Reliability and Sustainability**: The fragility of distributed systems

Reliability (the workflow can perform consistently well) and Sustainability (the workflow will continue to work reliably in the long term) are core features of a scientific analysis workflow, as they ensure the integrity of the methodology and reproducibility in the future. This holds especially true for scenarios that involve automated acquisitions and high throughput of data, such as found in the example of High Content Screening.

To assess for a reliability in any workflow that composes multiple tools, the orchestration and every tool should be considered to represent a *potential point of failure*. If tools share the same

framework, such as if all of them are ImageJ plugins, potential error sources lie mainly within the framework, as most of the underpinning software logic is provided by the framework.

In a distributed setting, however, each of the tool's *underbelly* is contributing separately to the error probability, which can render distributed system less reliable, if the tools themselves are not programmed to account for every eventuality. Given the general instable nature of new methodology, and the illustrated additional problematic of *having to reinvent the wheel* to develop the tools, this renders modern multi-tool workflows inherently less reliable.

In parallel, multi-tool environments also affect the general sustainability of these workflows. Even a minute update to one tool might unexpectedly render the entire pipeline unusable, if it breaks the programming interface that was used to facilitate making these tools interoperable.

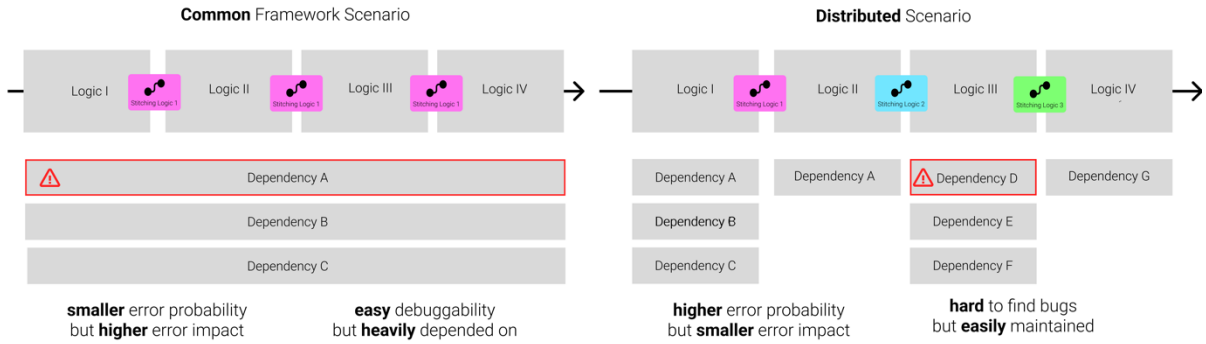

Figure 18 *Reliability and Sustainability Problems in workflows governed by a common framework vs distributed workflows spanning multiple tools. Each box representing a unit of code that can potentially cause an error.*

2.5.7 **Interactivity**: The lack of immediate feedback

As most analytical questions today are only partially automatable, some of the analytical steps require the user to fine-tune the parameters to yield sensible experimental results. The scope of this *Inspect Adjust* circle can expand and span multiple analysis steps: A denoising algorithms parameters may influence a subsequent cell segmentation, that then in turn influences the quantitative logic that measures each cell's volume.

Finetuning these parameters in a unified system such as ImageJ can become trivial, especially when utilizing automated systems like the CLIJ2-Assistent (Haase et al. 2020), that allows for easy propagation of parameter changes through multiple image steps. In a distributed setting

they however necessitate a high level of real-time orchestration and interoperability between the applications, which currently does not exist.

2.5.8 Data Management: Inaccessible, scattered data

Even when it is possible to perfectly pipe the tools together and orchestrate the workflow, a core challenge of modern image analysis is the data management (Schmidt et al. 2022a). Scientific analysis is only translatable to new projects if it can conserve and adapt to the challenges of data handling in these projects. It is not only important for the user to be able to visualize and inspect the data *throughout* the workflow, but equally finding means for exploring and recontextualizing the data once the original workflow has been run. Seldom does a scientific analysis end when the original question that created the analysis workflow has been answered, and data needs to be reprocessed to account for potential biases or to explore new scientific ideas. This exploration is only feasible if data is managed in a safe and standardized way (Wilkinson et al. 2016).

Unfortunately, data management in microscopy today is an immensely challenging task, as in distributed multi-tool workflow environments data gets lost in different folders, or on different hard drives (Tedersoo et al. 2021). Also as microscopy data management does not only include managing binary image files, but also necessitates the inclusion of metadata to further categorize and explain the datasets, today's vast array of organizational metadata storage (filenames, excel sheets,..) render metadata management reportedly still highly challenging, especially for beginners (Sarkans et al. 2021). Especially aggravating to this reality is the fact that many of today's developer tools do not correctly propagate metadata when performing IO operations, i.e. when python script reading an OME-TIFF-file with metadata saves it as a *vanilla* TIFF without any additional metadata (Mitchell et al. 2022).

How widespread the negative impact of the data management problematic is, gets emphasized as scientists quote *No time to search* and *data lost*, as the top two reasons when they decline data sharing, a pillar for good scientific practice (Tedersoo et al. 2021).

Accounting for this reality, the "*The FAIR Guiding Principles for Scientific Data Management and Stewardship*" (Wilkinson et al. 2016) intends to provide guidelines to improve on today's challenges of data management by emphasizing the *Findability*, *Accessibility*, *Interoperability*, and *Reuse* of digital assets. Their proposed principles emphasize machine-actionability, i.e., the capacity of computational systems to find, access, interoperate, and reuse data with none

or minimal human intervention. Hereby they account for the increasing calls to provide and automate the process of feeding data into more central repositories that can be used to train larger machine-learning models or meta-analysis. (Nogare et al. 2023)

Unfortunately, this data sharing to a wider public also still remains highly challenging with only a small percentage of bioimage users having shared data in a public repository (Schmidt et al. 2022a), even though archives like Bioimage Archive (Hartley et al. 2021) provide readily available solutions to publish scientific data in any format. One of the leading causes for the non-adoption, is that researchers are concerned about the security of their shared data and the misuse of the data without their consent (Tenopir et al. 2015).

OMERO-Server (Allan et al. 2012) has been around for almost a decade trying to account for the data management needs and challenges of a lab, and provides a readily available solution that addresses core aspects of FAIR data management. Backed by a relational database for metadata storage and a file-storage solution based on OME-Tiff, it tightly integrates with software like CellProfiler that can extract, transform, and load data back into the server, conserving he metadata in the process.

However, as OMERO-Server needs intensive administration and a trained audience that understands its storage solutions, it is seldomly integrated as a primary data backbone in a bioimage workflow, and for the most part often represents a storage solution for already processed data, limiting its general potential (Schmidt et al. 2022b).

2.5.9 **Provenance**: Ensuring data integrity

A core premise of the scientific method is to ensure transparency in how data was generated and analyzed. This necessitates the bundling of information which transformation were applied to the data at which point, and how parameters of the methodology were set: provenance. Provenance is a crucial piece of scientific information, as it enables to uncover potential biases and error in the methodology. Meta-studies relying on this information could for example show that in the Journal of Cell Biology, 10% of accepted papers contained inappropriate manipulation of image data (Martin and Blatt 2013).

The Method sections often describe their workflows in plain English, simply listing the transactions that happened to the images. They however often omit reporting specific parameters to the used algorithms, as well as the logic of the conversion scripts they

employed for interoperability reasons. While probably mostly accidental omissions, sometime this information is withheld deliberately, to manipulate and deceive (Martin and Blatt 2013).

Recently, efforts in finding and establishing techniques have been spearheaded by organizations like QUAREP (Quality Assessment and Reproducibility for Instruments & Images) (Fallisch 2023) (Boehm et al. 2021) and more wider software solutions like logging data transformations on a blockchain (Jaquet-Chiffelle, Casey, and Bourquenoud 2020) and cryptographical hashing of transactions inside the image ("Overview - C2PA" 2023.) are being developed. However, they lack wider application inside bioimage analysis.

Provenance is not only important for the scientific method, but also required for the accurate integration of processed data into the training set for a machine learning algorithm. Only when transformations can be accounted for, is an accurate classification possible. As image modalities of today's bioimages are so diverse, the lack of protocol and documentation of methodology arguably stands as one of the biggest challenges in deep learning for bioimage data (Meijering 2020).

2.5.10 **Discoverability:** The lack of common repositories

Another core challenge of modern workflows is the discoverability of the software. Tools not only need to be available, but users need to be able to reliably discover them and find their respective implementations. However, while the quantity of tools is constantly increasing, non-expert users stated that they are *overwhelmed* with the variety of new tools being created and call for more 'common repositories" (Sivagurunathan et al. 2023).

Accounting for this need recently projects like the Image.SC Forums are trying to provide an entry point to get users into touch with more image analysis tools and are partnering and advertising open-source projects in dedicated subforums. They now provide a much-needed global discussion space for image analysis. Another community effort the "Bioimage model zoo" (Ouyang et al. 2022) is trying to advertise dedicated deep learning models for image to image translation tasks such as segmentation.

2.6 Proposed Solutions

As illustrated in the last section, modern image analysis workflows seem to be plagued by their increasingly distributed nature and the lack of common framework such as ImageJ to rely on for basic functionality. Each of the mentioned problems has led to the design and implementation of dedicated software tools that were mentioned alongside the corresponding problematic.

However, recently the scientific community has seen more tools and software emerge that try to find overarching solutions to this problem. This section will now introduce some of the generic solutions in this space that try to tackle the problem of modern workflow management in a wider scope, as well as explore dedicated tools that were developed for the bioimage world. This section will also contrast them directly with challenges they aim and fail to solve.

2.6.1 General Purpose Workflow Managers

Lack of workflow orchestration management is not a problem unique to bioimage analysis but found in a variety of scientific and business scenarios. *General Purpose Workflow Managers* are software solutions that emerged to tackle common problems faced in these scenarios and facilitate the easy orchestration of software workflows that span multiple, distributed tools. Generally, they have found their application in highly automatable workflows in bioinformatics or data analytics. Three major platforms, that have seen their application in bioimage analysis, are discussed here.

2.6.1.1 KNIME

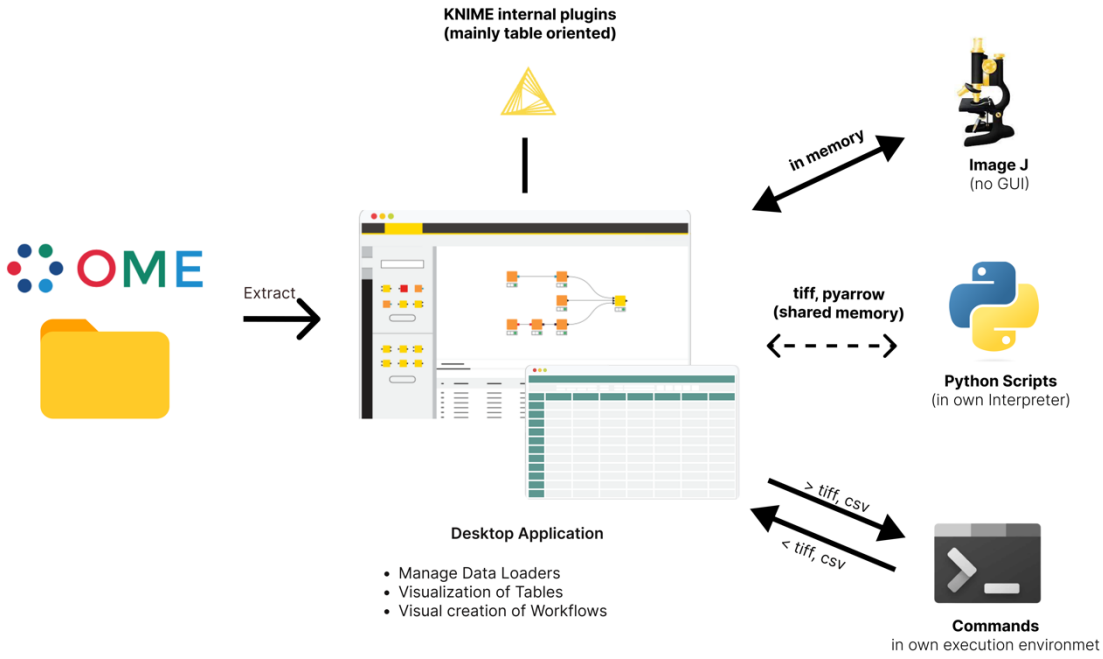

Figure 19 *The KNIME software design*

KNIME (Konstanz Information Miner) (Berthold et al. 2009) is an open-source general-purpose workflow management software, that was created to provide a modular and open data processing platform for the Java ecosystem. It allows for the easy integration of different data loading, processing, analysis, and visualization modules without the focus on any application area. It comes with a general-purpose design, and its visual programming interface has seen a broad application in the data analysis world. KNIME comes with a vast set of built-in modules to handle transformations of tabular data and provides support for transformations

image data, for example through calling ImageJ macros, or running external Python addons or scripts. KNIME is also available as KNIME hub, which is a cloud-based paid version of the KNIME ecosystems, which is not discussed here.

2.6.1.1.1 Benefits

Usability

KNIME allows for the easy design of complex analytical pipeline through its visual programming interface. These workflows can span multiple internal tools such as table readers, table manipulations tools or additional installable plugins such as ImageJ macros. KNIME boasts a comprehensive documentation. (“KNIME Community Hub”) and users can rely on a vast selection of bridged tools for their data analysis need.

2.6.1.1.2 Limitations

Interoperability

Even though KNIME provides support for ImageJ, its interoperability with the wider bioimage ecosystem is highly limited. While it can run ImageJ1 macros, and ImageJ2 plugins, its tight integration does not allow for the usage of external plugins in the pipelines. Other software integrations like being able to run CellProfiler pipelines on the images, remain highly experimental and are currently unmaintained (“Knime-Bridge”, 2015).

Orchestration

KNIME’s workflow are easily designed analytical pipelines that are optimized for larger batch and post-hoc analysis. As such the interface focuses on extract, transform and load operations and provides no comprehensive tools for the orchestration of real-time events during a workflow. This renders KNIME unusable for modern Smart Microscopy workflows.

2.6.1.2 Galaxy

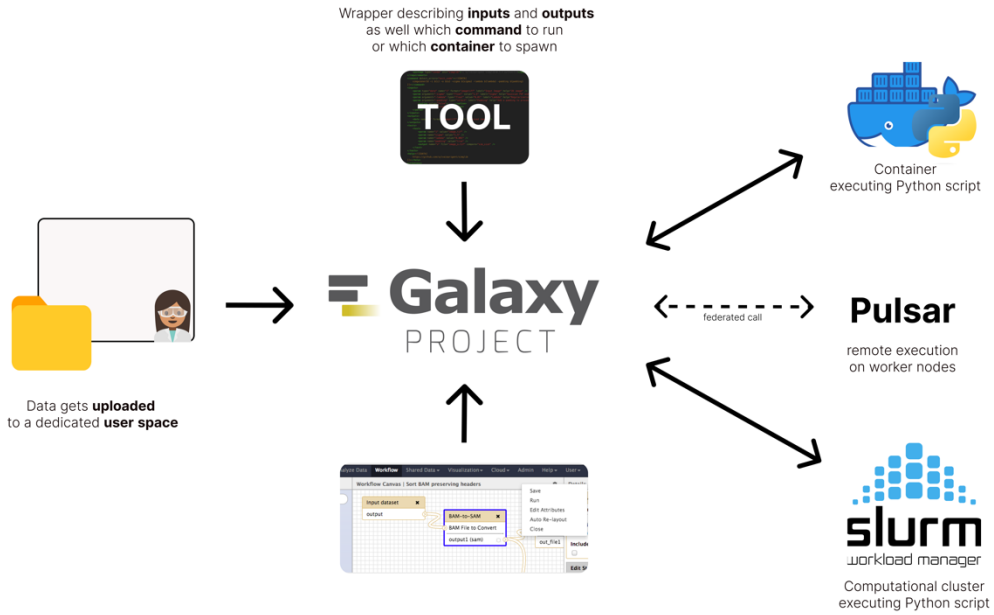

Figure 20 *The Galaxy software design*

Galaxy is an open-source platform that provides a web-based interface for computational research, primarily used in fields such as bioinformatics and genomics. The platform uses a cloud-based model but can also be installed locally, catering to users with varied computational skills. The Galaxy project provides a publicly hosted instance, that allows users to use the projects computational resources, to facilitate their workflows.

Galaxy incorporates a wide array of bioinformatics tools through their abstraction of *Tools* and allows complex data analsis workflows via a visual programming editor. *Tools* can represent a variety of different software (CLI, Python script, Docker Container) that are wrapped with a *Wrapper* file, describing the inputs and outputs of the tool and which command should be executed when invoking this tool in a workflow. When executing a task in a workflow the *Tool* and their respective inputs are send to an executor, that runs the tools on its computing environment. These environments can for example constitute software containers, a connected SLURM Cluster (Yoo, Jette, and Grondona 2003) or remote computers that are addressed through a federated (server-to-server) protocol (“Pulsar”, Galaxy Project). Galaxy also provides a robust and extensible data management system based on files and allows for data import from prominent genomics databases and local storage.

2.6.1.2.1 Benefits

Reproducibility & Portability

Galaxy *Tools* can run on a variety of different technologies such as running directly on the server or on a computational cluster. A common approach however is the containerization of complex software requirements in docker containers that are then executed by the Galaxy server on a connected docker instance. As such these tools can be easily ported from one Galaxy instance to another and analysis workflows yield highly reproducible results.

Provenance:

Galaxy ensures automatic workflow provenance by maintaining comprehensive logs of all analyses step in a workflow, enabling the easy repetition and sharing of workflows. This provenance information is however not linked directly to the processed data and needs to be shared separately.

Data Management:

Galaxy comes with support for file-based data management, organized into user spaces. Files can be inspected on the website with basic image viewing capabilities (not including multidimensional or 3D visualization of data), and maintained with associated metadata, predominantly in form of key-value pairs (“Metadata” , Galaxy Project.).

2.6.1.2.2 Limitations

Usability:

Once set up, the web interface of galaxy is tailored to non-programmers and the visual interface allows for easy creation of workflows. Setting up the galaxy server however requires extensive configuration management and expert knowledge in deploying complex multi service web environments.

Interactivity and Orchestration:

Galaxy analysis workflows are designed for a particular style of command-line based data analysis. It is therefore not designed to support stateful user interaction (e.g. when waiting for user input) nor real-time communication between workflow steps, limiting highly interactive or smart workflows.

2.6.1.3 Nextflow

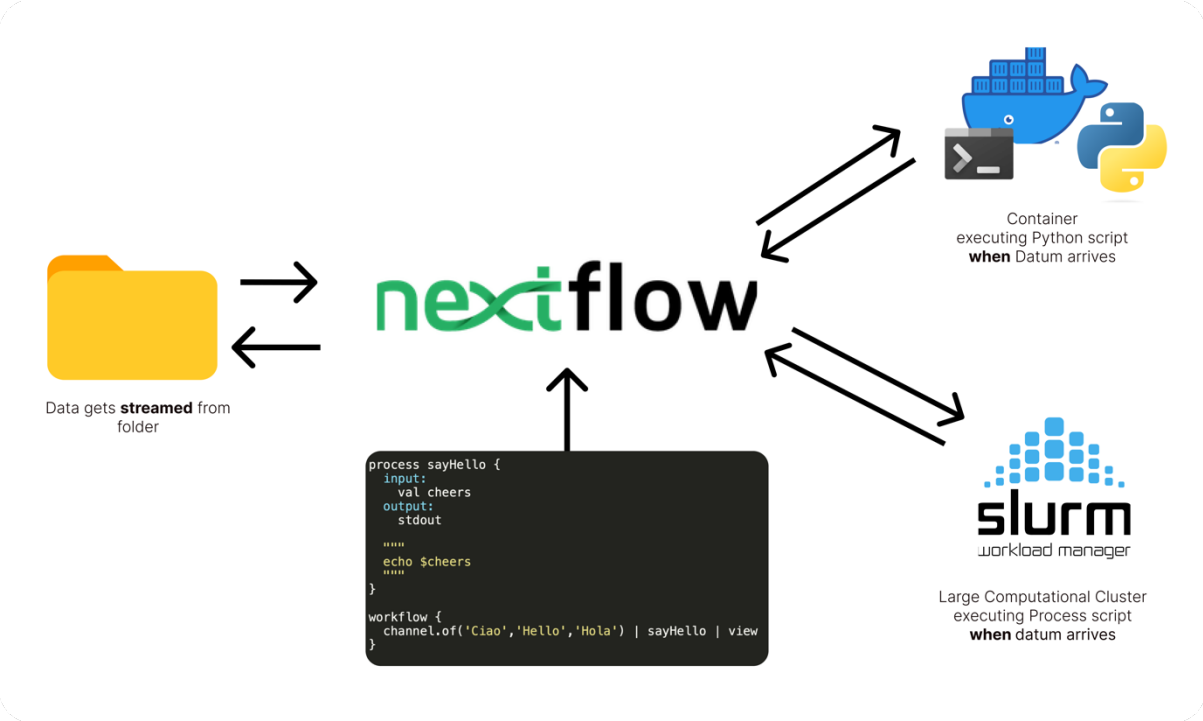

Figure 21 *The Nextflow software design*

Nextflow (Di Tommaso et al. 2017) is an open-source data analysis workflow tool developed by "Seqara Labs". It is designed to handle computational data pipelines in a reproducible and reusable way, irrespective of the infrastructure used to execute them. These pipelines are *scripted* through Nextflow's dedicated workflow language, a domain specific language (DSL), built on Groovy ("Groovy Reference Documentation", Apache Groovy), that uses the dataflow programming paradigm to transform, filter and combine data. This paradigm uses the conceptual abstraction of data as a stream, which moves from one analysis process to another, while these processes run *independently* from another. When workflows are executed, processes can be run through various *executors* that can take the process *script* code and execute them in software containers, or schedule them directly on high performance compute clusters.

2.6.1.3.1 Benefits:

Orchestration

Nextflow's dataflow programming paradigm allows for the design of highly efficient computing pipelines, as tasks can run and process data autonomously and in parallel on their dedicated compute resources, when data arrives, eliminating the need for intense script-based orchestration.

Reproducibility

Nextflow can rely heavily on containerized software for execution, and all execution info including the process code is contained inside the workflow scripts. This design decision renders the whole analysis pipeline highly reproducible.

2.6.1.3.2 Limitations:

Usability

Nextflow does not come provided with a graphical user interface and relies entirely on programming in its own workflow language to create an image analysis pipeline. Users need to be therefore familiar with the scripting language, as well as the command line interfaces for their requested tools to create a workflow.

Interactivity

Similar to Galaxy, Nextflow does not allow for real-time interactive workflows, as tools need to be runnable inside docker containers.

2.6.2 Dedicated Solutions

While the general solutions provide some alleviation to the problem of workflow and data management, their lack of focus on the bioimage analysis space hinders their general adoption. Accounting for this need, more dedicated solutions have been developed and will be discussed in this section.

2.6.2.1 ImJoy

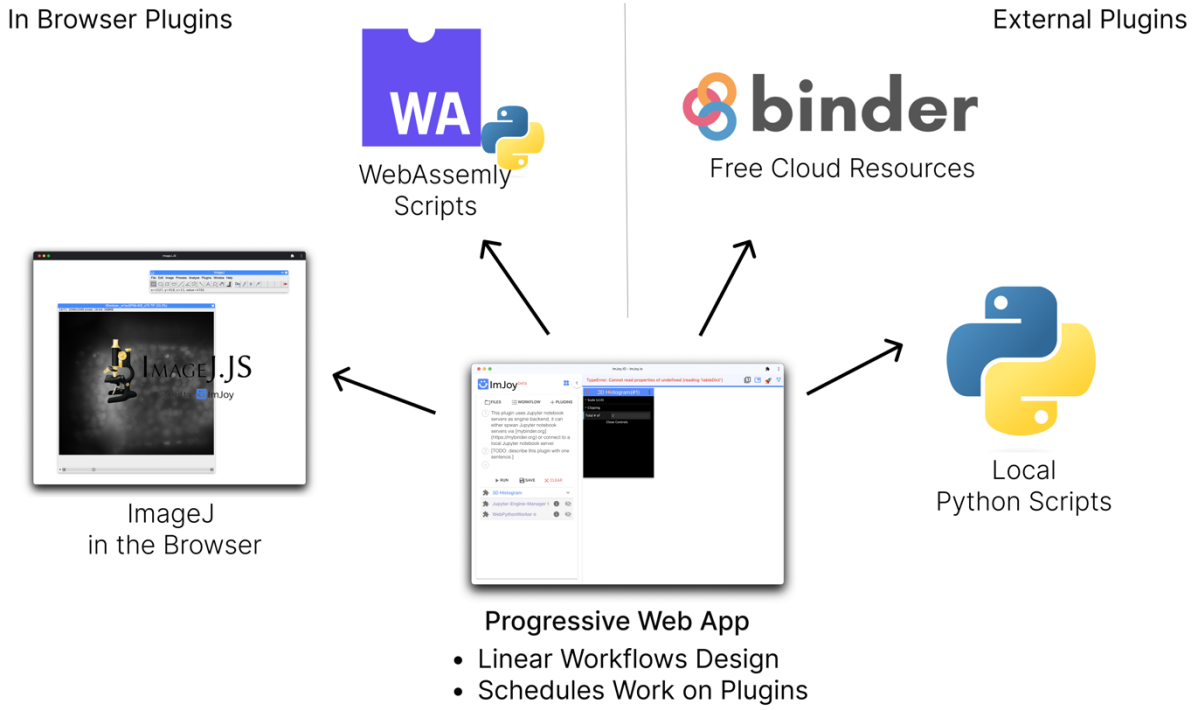

Figure 22 *The ImJoy software design*

ImJoy (Ouyang et al. 2019) is a computational platform originally designed to allow users to easily share, and deploy deep learning-powered application. It comprises a standalone web application that represents a central hub that in turn connects to analysis *plugins*. These *plugins* can represent, for example, Python scripts on a local or remote machine or hosted in the cloud on “Binderhub” (Jupyter et al. 2018), but also code running directly inside the users browser (Rossberg 2019). ImJoy then allows the user to design linear workflows spanning multiple *plugins*, which they can then run on uploaded data. The platform has since grown into a comprehensive platform adding support for plotting (ImJoy graph) or a complete virtualized desktop (ImJoy desktop), which provides browser-based access to common bioimage analysis apps. One of these apps is ImageJ.JS, an ImageJ variant trans-compiled to WebAssembly (Rossberg 2019) and JavaScript.

2.6.2.1.1 Benefits

Usability

ImJoy provided one of the first solutions to easily share deep learning applications in the field of bio-image analysis. Through some example plugins, it was able to show how Deep Learning tasks can be easily handled in the browser and cloud without the need for a software installation.

Interoperability

ImJoy, also comes with support for an extensive list of bioimage applications and can interface with software solutions like KNIME, ImageJ.JS and open platforms such as Bioimage Model Zoo. Most solution can be used fully interactively in the workflows (e.g., for marking region-of-interests). Additionally, it provides an extensible plugin system, that allows the easy integration of developer scripts in its workflows.

2.6.2.1.2 Limitations

Data Management

One of the core limitations of the ImJoy platform for modern workflows is that it does not natively support forms of data or metadata management. Users need to handle their data separately, ensuring the correct propagation of their data, while developers need to define serializable objects that can be passed through the socket connection between the web app and its plugin.

Orchestration

ImJoy ‘s visual workflow system allows for step-by-step pipelines that can transform the data. Non-linear workflows however, that require more complex sorting of the data, are only possible through utilizing the ImJoy programming interface. Outside of its original focus, ImJoy was not designed for user-friendly, real-time analysis, even though its underlying architecture could support it. Plugins like a *pycro-manager* (Pinkard et al. 2021) plugin, that enables streaming images to the WebUI and even controlling the microscope, exist (“Interactive Microscopy Control”) but require a complex setup and are currently limited to be run on the same machine.

Reliability, Sustainability

As the ImJoy developers switch the established web model of server and client around (ImJoy Webapp acts as server, and Plugins as clients) (“ImJoy RPC”, 2023), they often need to find ways to break out of the security sandbox of modern browsers that heavily restrict communications that can be initiated from a browser app. Since its inception these restrictions have only gotten more and more, which as of 2023 renders the stable ImJoy website unusable for some browsers (tested on Chrome 114.0.5735.106 in June 2023, Windows, Linux). Additionally, as it relies on the browser as a central hub, accidental browser or app closure can terminate the workflow and lead to unpredictable outcomes like continuation of tasks on plugins.

2.6.2.2 BioimageIT

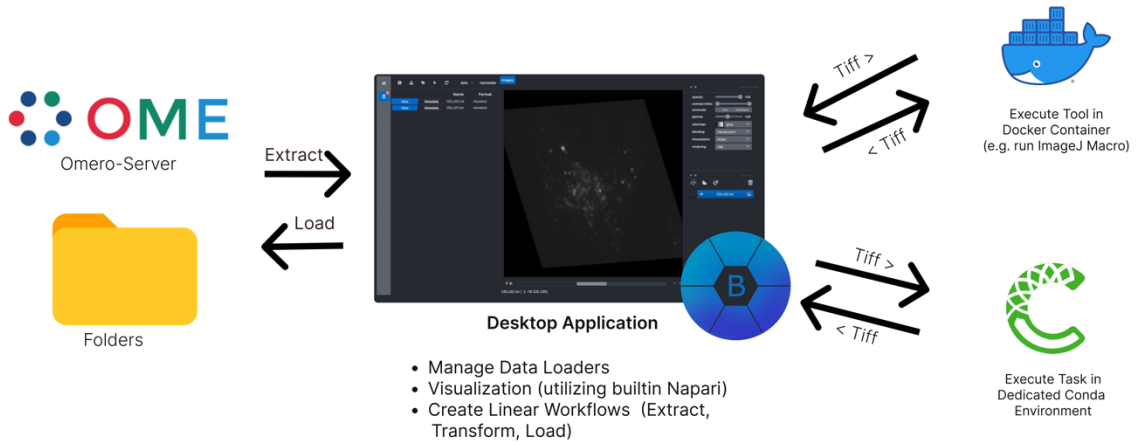

Figure 23 *The BioimageIT software design*

BioimageIT (Prigent et al. 2022) is an analysis and visualization software that runs as a standalone app and connects to a local Docker instance to run analysis software in containerized applications. Users can ingest data from various sources including OMERO-Server as well a local TIFF Files and build linear pipelines transforming the images in their passage through containers or through Python-based plugins, termed “runners” that run in dedicated Anaconda (“Anaconda”, Anaconda Inc.) environments. Containers can be implemented in their desired programming language and define their input and outputs through a dedicated configuration file that is compliant with the Galaxy *Wrapper* format, as well as metadata files, that helps to organize the tools within the BioImageIT interface.

2.6.2.2.1 Benefits

Portability

Through its reliance on containerization of bioimage tasks in Docker containers, BioimageIT can create fully reproducible environments for bioimage tasks, that are frozen in time and are guaranteed to work predictably across different execution environment. Containers are easily installed, akin to ImageJ plugin, through drag and drop.

Data Management and Provenance

BioimageIT includes a dedicated data management solution that provides abstractions for data transformations on a set of data. A data pipeline is contained within an *Experiment*, and then each step of processing will create a container (a *Dataset*) that stores one processing step and its contained data points (*Data*) from raw-data to the end of the pipeline. Additionally, each container can save key-value based metadata attached to the files. A log of the data processing is exportable.

2.6.2.2.2 Limitations

Data management

Although being able to handle and organize data according to the metadata, BioimageIT is restricted to image files as data points and currently does not provide dedicated support for annotations like ROIs or Labels on the Images themselves, which restrict its ability to explore and model more complex relations in the data.

Orchestration

Bioimage IT is not designed for real-time analysis, and analysis tasks are not executed concurrently but rather subsequently. Bioimage ITs execution model of a container task is implemented through putting data into a volume, spinning up the docker container, calling the command line interface of the tool, and then taking the processed data from the volume. This design lends itself to batch analysis, but inherently incurs performance penalties when processing data in real-time, as all three steps can take seconds to perform.

Interactivity

A core limitation of the execution model of docker containers in the Bioimage IT model is that it does not allow for human in the loop type analytical flows, where user interaction on a datum is needed.

2.7 The objective of this thesis

As seen in this introduction, modern bioimage workflows are shaping what the future of bioimage analysis will look like and are already helping to uncover new biological insights.

However, the world of modern bioimage analysis also experiences an increasing set of challenges, that even though manageable by experts, increase the disparity between the expert and non-expert world, restricting the true potential of modern workflows to a limited selection of scientists. While there is an increasing set of solutions to aspects of this overarching problem, there is a need for a more holistic approach that truly democratizes modern bioimage workflows to a wider public.

The objective of this thesis was to therefore explore solutions to the common bioimage problems of today, account for them in a software platform design and lastly implement a sound backbone for the image analysis workflows of today and tomorrow.

To validate the results of the developmental process and test for the applicability of the platform, this thesis will use three exemplary modern bioimage workflows to illustrate its potential and limitations.

3 Design and Implementation

This chapter will now delve into the design and implementation of Arkitekt, the software framework and platform, that was developed in the course of this PhD to address the challenges of modern bioimage analysis. It will first explain the leading design decision of the Arkitekt framework, and then explain and justify the implementation strategies in more detail.

3.1 Leading Concepts

A leading design decision for Arkitekt, is to *not reinvent the wheel* and utilize the features of bioimage analysis tools that already exist. Arkitekt is therefore designed as a sound *backbone* that enables *interconnecting* a variety of apps seamlessly in *workflows* through an *easy interface*, *offloading computation* to these apps and the *hardware* they run on.

Hence, the Arkitekt platform acts as a *middleman* between the users and the bioimage applications and utilizes remote connections to bridge the apps that can be located *anywhere* in the lab or cloud. Arkitekt with its Interface then represents *a simple to use* abstraction layer and *graphical user interface* for the users to remotely do work on connected apps.

Additionality, Arkitekt is meant to act as a *workflow scheduler*, that takes simple to create but powerful *real-time* workflows and schedules them on the ecosystem of apps that it connects to.

Arkitekt was also designed as a multi-user *data hub*, providing central storage for all the *data* and *metadata* needs of modern biology labs, including images, quantitative metadata and annotations. It was also planned to enable *social* features such as comments and notes and *collaborative* data exploration while ensuring means for *provenance* and *data integrity***.**

Lastly, Arkitekt was devised as a *framework*, that *developers* can use to *create*, *run and share* their analysis scripts and software with the wider *community*.

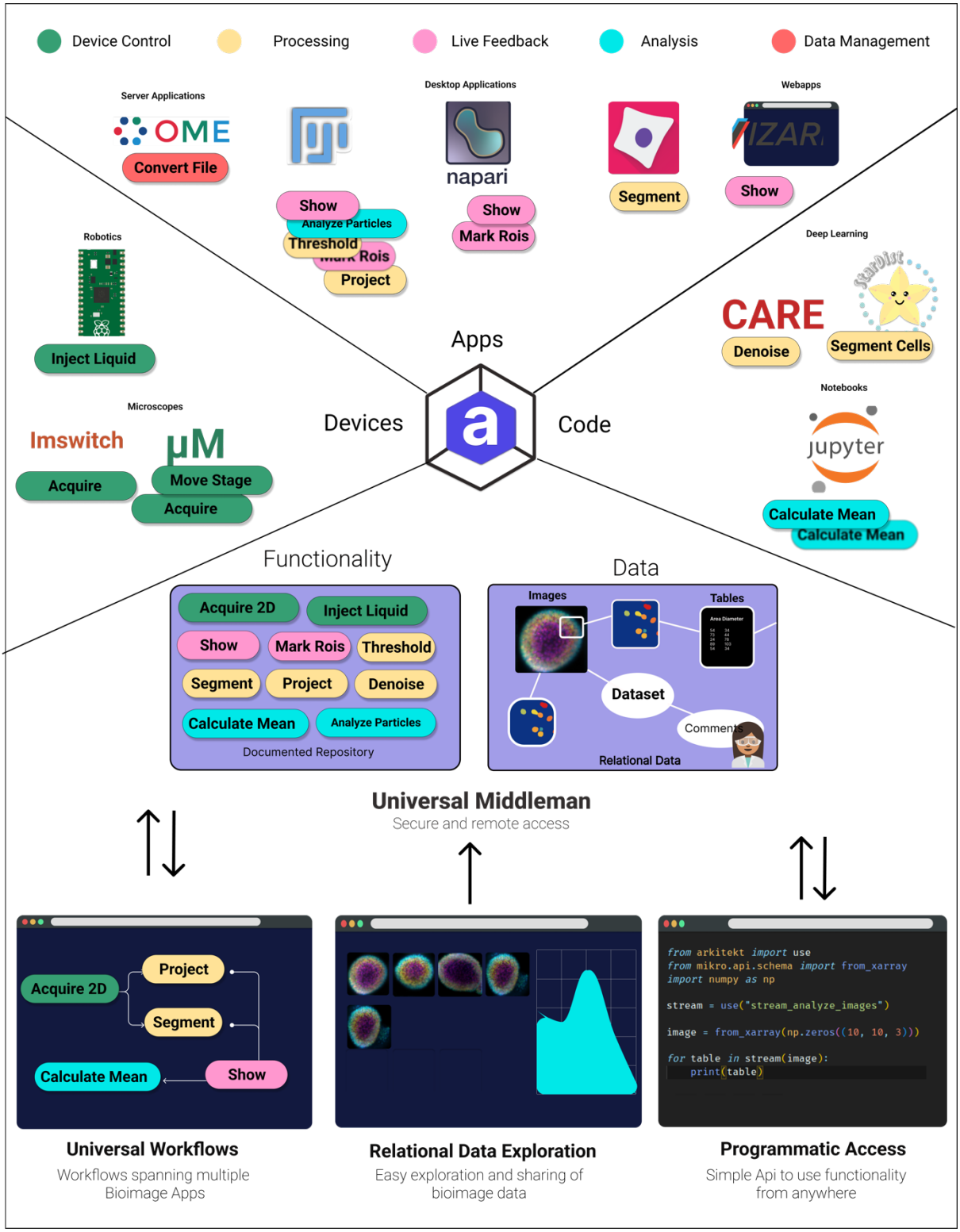

***Figure 24**: The leading design concepts of the Arkitekt framework*

3.2 Concepts in Detail

The following section describes the core design features of Arkitekt more closely and gives a more detailed explanation of their underpinning implementation on the software side.

3.2.1 Arkitekt the Middleman

Arkitekt's core design lets it stand as a *zero-functionality* platform. Zero functionality means that the platform, despite its hundreds of thousands of lines of code, does not offer any functionality of actual image analysis. Rather it stands as providing a secure *foundation* to build applications (tools) with and to provide features to make them interoperable and execute safely and reliably in overarching workflows.

Rather than *imposing* specific implementations of common image analysis procedures as building blocks in workflows through the platform, users of the platform are invited to use the specific implementations that a tool chose to enable on the platform.

Arkitekt itself acts only as a broker and central repository for this functionality, taking care of calling the apps and their functionality in a reliable manner and ensuring the integrity of the call, as well as providing a common interface to all the lab-wide functionality.

To this end, and to ensure the later discussed data features, Arkitekt is implemented as a server system that is installed on a central instance (be it a laptop, desktop computer or cluster of computers) and that advertised itself as an endpoint, that all bioimage applications can connect to. These connections are at the heart of the Arkitekt platform, as they provide a secure way for communicating tasks and data amongst the connected apps and follow protocols of established advanced programming interfaces (APIs) such as GraphQL, S3 and OpenID (more in *On Open, and On Secure*). Following the design principle of a *middleman*, the Arkitekt platform relies entirely on the connected applications to interface with users and indeed does not even provide a non-expert facing web interface itself. All user interactions happen through a dedicated orchestrating and managing app named Orkestrator (see *On Interface*s), which is easily installed on any hardware.

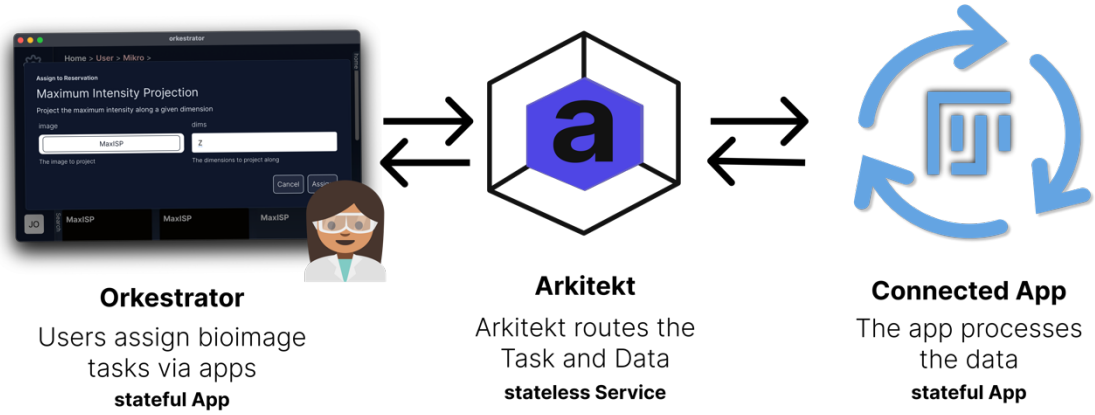

Figure 25 *Arkitekt the middleman: Arkitekt establishes itself as an intermediary between apps, and only forwards requests amongst apps. Apps can choose to assign work to other apps (scheduling apps) and process work for other apps (providing apps), by telling Arkitekt which functionality they provide. Apps never communicate directly but always through Arkitekt, which keeps track of tasks, data and which apps are available for routing and assignment..*

To ensure maintainable software packages and emphasizes separation of concern, Arkitekt is built around a suite of smaller software modules and servers, so called *microservices* (“Microservices”, Martin Fowler 2014), that each is responsible to enable an aspect of the functionality of the Arkitekt platform. These modules can operate completely standalone but work together to enable the whole platform. This was a design choice to enable developers to easily deploy additional plugins (such as new data storage solutions besides *Mikro*) into this ecosystem, but enable also enable them able to pick and choose core functionality like authentication and authorization.

Figure 26 *Arkitekt microservice design stratified amongst the supporting technology layer, its implemented web services, and the API layer that it provides.*

3.2.2 Sensible Abstractions

A core challenge in the design of both the user as well as the programmatic interface of Arkitekt was to find a set of sensible abstractions. These abstractions needed to be both easily understood and memorized by a user, that wants to make use of a simple workflow spanning just two apps on the same computer, but also provide enough flexibility and debuggability for an export user that needs to run a complex analytical workflow spanning microscopes and multiple lab computers.

Another requirement for these abstractions was also to abstract complex implementation details away, like which specific hardware a specific bioimage task is set to be executed on and to establish a more general user-friendly terminology for universal bioimage workflows.

To this end, Arkitekt is developed around a set of abstractions, which closely mimic terminology in everyday language describing workflows akin to the work of the EDAM ontology (Kalaš et al. 2019). They are designed to help users in discovering the functionality they want to use in their workflow, as well as designing them and eventually executing them.

The following section explains these terms in more detail.

3.2.2.1 Nodes

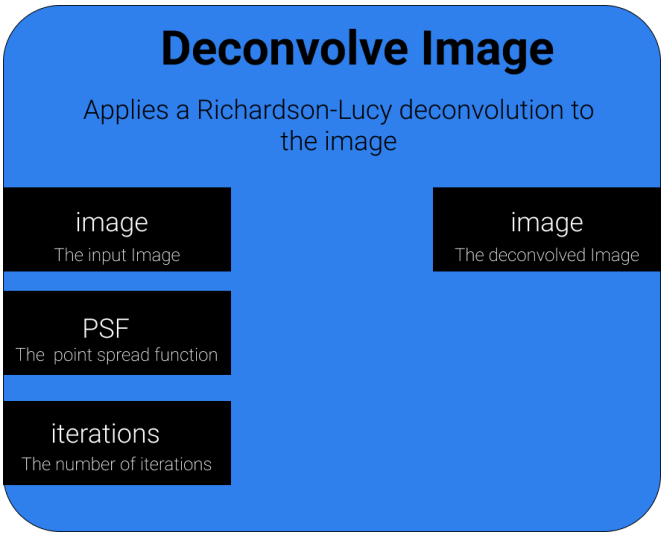

Figure 27 *The Arkitekt node concept. Nodes constitute abstract bioimage tasks, which are defined by their name and a description of their algorithmic processing as well as their Inputs and Outputs (declared as ports on the left and right respectively following the pattern of data type plus label). Arkitekt uses a hashing algorithm to uniquely and universally, identify the node based on this definition.*

Nodes represent the core abstraction of functionality in Arkitekt: following the idea of a node in a workflow and the *component* idea (Paul-Gilloteaux et al. 2021) they represent a *conceptual* bioimage functionality, e.g., *Show an Image*, *Acquire an Image Stack*, *Segment Cells*, *Deconvolve*, etc. and describe a step of data transformation. They are defined through inputs and outputs for each task, as well as a description of its processing. Importantly for the conceptual idea in Arkitekt, nodes are *implementation agnostic* and rather represent a conceptual bioimage task, a *protocol* of transformation.

Apps can then state to provide an *implementation* of this specific functionality: a *Template*. Arkitekt will internally link this *Template* to the *Node* it represents. This abstraction allows multiple *Apps* to provide the same functionality by implementing the same *Node*. For example, both ImageJ and Napari can choose to state that they can visualize images and mark ROIs, which will then internally represent two *Templates* on a given *Node*. By letting users think about *Nodes*, not *Templates* (definition over implementation), the usage and their orchestration in workflows can stay conceptual, universal, and sharable, and only on deployment and execution of workflows their Arkitekt instance will map them to *real world* implementations on the connected Apps.

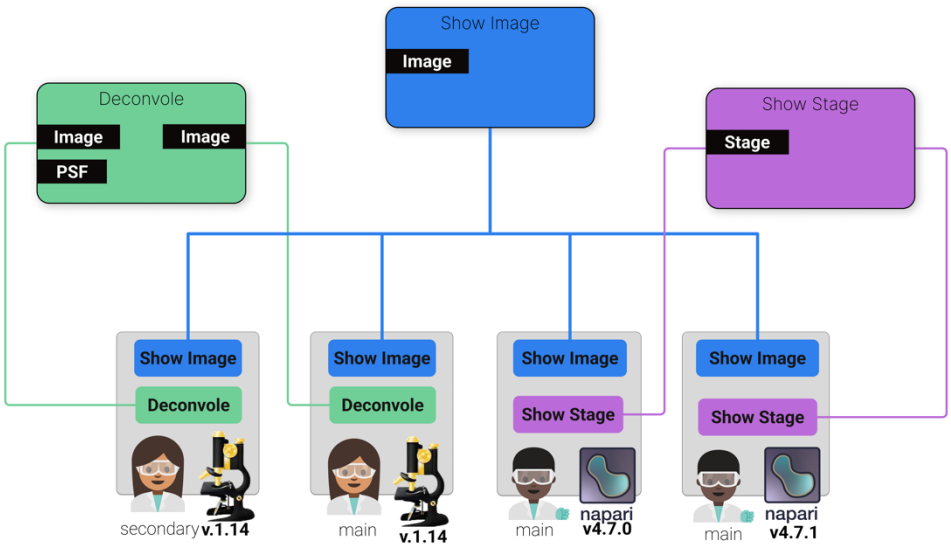

Figure 28 *A visual representation of the Arkitekt Node-Template Abstraction, Nodes represent abstract notions of a bioimage tasks, that are linked to real-world implementations of tasks on connected bioimage apps: Templates. Templates are not only defined uniquely for the implementing application but also for its version number, the person who is using the application and optionally which instance of the app is being used on (e.g., Fiji of Experimenter A on their private laptop and Fiji of Experimenter A on their lab computer). When choosing to do the bioimage tasks the Task request will then be mapped to one or more of the connected templates (more in Reservation and Templates)*

To adjust to the reality of bioimage functionality and how data gets produced or altered, *Nodes* fall into two broad subcategories: *Stream Nodes* and *Functional Nodes*.

A *Functional Node* on task assignment, will execute, give back the results and then stop executing. Examples of such *Nodes* are processing steps like deconvolution that takes two images (image and PSF) and gives back the deconvolved image.

A *Stream Node* on the other hand, can continue executing after giving back the first result, and therefore establish a stream of data. One example of this is an *Acquisition Node* which on invocation starts a microscope time-lapse that continuously produces images or a *Stream Folder Content Node* which watches a folder for changes in files, that then will be automatically uploaded for further processing.

Stream Nodes are at the heart of *streaming* workflows, as they can establish the primary data stream over time (More on this in the *Workflow* section).

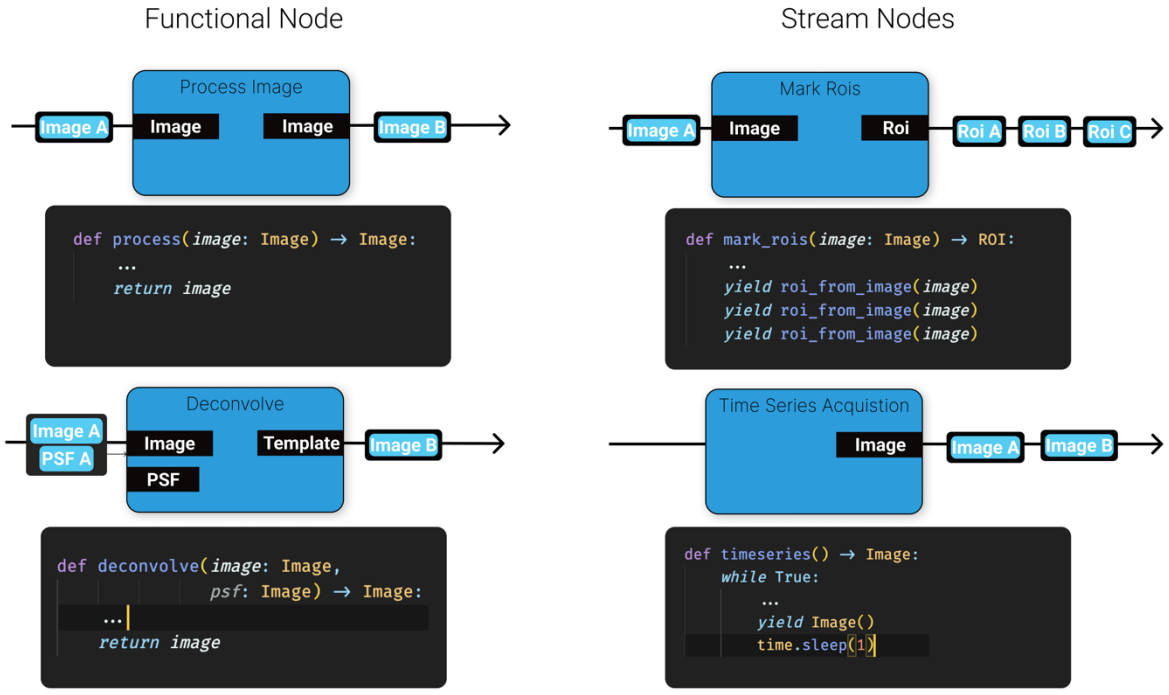

Figure 29 *Nodes closely follow conventions found in functional programming. A Function Node is equivalent to a function in programming: Inputs represent the arguments and Outputs represent the returns values of a function. A Stream Node is equivalent to an iterator that yields the outputs. This resemblance ensures that developers can easily think how they can both provide functionality to the platform (registering a function) and to call functionality of Arkitekt in a familiar way (calling functions).*

By closely staying with the idea of *functions*: Arkitekt's design of *Nodes* and *Templates* enables additional abstractions: *Nodes* can have specific testcases or benchmarks attached, which can then be tested against all of the implementations (e.g. which *Template* is the fasted to run, or which segmentation model provides the best result for a specific dataset).

3.2.2.2 Workflows

Workflows are Arkitekt's abstraction for a bioimage analysis pipeline or workflow. Similar to classic visualizations in the bioimage literature they are visualized as a graph that is

composed of *Nodes* orchestrating multiple bioimage analysis functionality together. They can represent simple batch-like linear analysis workflows (i.e. *Acquire -> Deconvolve -> Segment -> Visualize*) but can also be used to create more diverging workflows based on analytical outcomes and complex real-time workflows such as closed loop microscopy experiments (e.g. *Acquire 2D -> Analyze -> If Analysis Condition=True -> 3D Acquire*)

To facilitate the non-expert design of these workflows they can be designed graphically with the help of the Orkestrator UI. The user selects corresponding *Nodes* for each *Task* that he wants to perform in his analysis and wires them together in order to manipulate and transform the *Data Stream*.
The *Data Stream* is the core abstraction in Arkitekt's implementation of bioimage workflows and represents a pipeline of transformations, combinations (with other *Data Streams*) and filters of data as they happen *over time*.

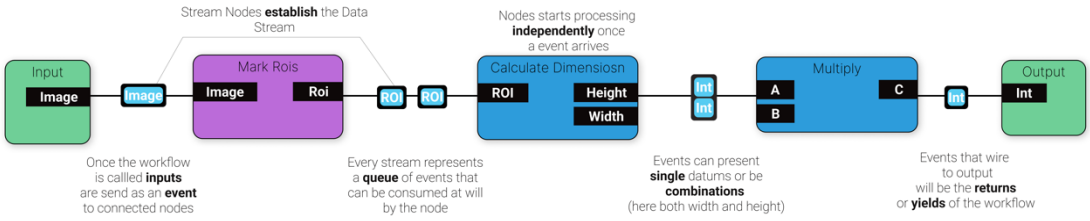

Figure 30 *An Arkitekt workflow that lets a user mark* various *Rectangular ROIs on an image (for example in ImageJ) and calculates the Area of the ROI for each, giving back the output to the user invoking this workflow.*

This approach to pipeline design is borrowing concepts from *dataflow based programming* (Dennis and Misunas 1974) and *reactive programming*, a conceptual style of programming that has gained traction in managing both complex UI interactions and in managing distributed systems (Shibanai and Watanabe 2018). The design and its approach to parallelism is often and probably best explained with the metaphor of *multiple* marbles falling down a marble play ("RxMarbles").

Marbles in this metaphor represent singular events (a datum) such as an image produced by a microscope, or an event detected on a sensor. *Nodes* in the metaphor represent building blocks that the marbles (the data) pass through. These building blocks can for example: alter the marble stream (transform a marble into another marble), filter it (taking marbles off the track), delay the marbles for a moment or combine marbles from different tracks together to form a new stream. Through simple placement of these building blocks even complex time orchestration can be facilitated.

For classic use-cases and workflows, this abstraction is visually equivalent to batch-based pipelines, where images get transformed step by step until the result data is achieved. However, this abstraction also allows to design highly concurrent workflows, where data is processed on the fly by multiple apps independently and recombined to then follow a common path. It can also be used to feedback data back as an input to an earlier task. To enable these advanced forms of real-time orchestration, Arkitekt provides a set of predefined *helper Nodes* derived from reactive programming, to help orchestrate the flow of data in time (e.g. how long to buffer data or when to merge data from two event detectors together).

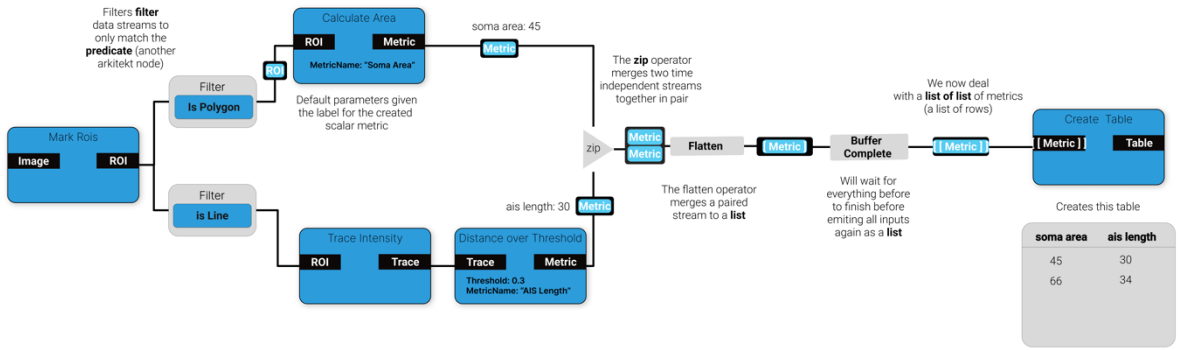

Figure 31 *A highly involved Arkitekt workflow that makes heavy use of the integrated helper Nodes. This workflow was designed to represent an partially automated analysis pipeline where a user on an image of neuronal cells wants to extract both the length of the axon and the corresponding soma are of each cell, by marking the respective instance. Here marked ROIS are filtered through the filter helper node and according to their shape are passed to different analysis functions extracting both length (based on the intensity trace along the line ROI) and area (purely based on ROI Features). Then the streams of ROIs are merged (zip + flatten) and when done with the workflow (buffer complete), the list of rows (corresponding to each cell), are used to create a table.*

Another core feature of *Workflows* in Arkitekt is that they become themselves *Nodes*, as they act on some inputs and yield or return. This not only ensures the integration of sub-workflows in bigger workflows, but has the real-world advantages, of using workflows just as another functionality of a connected *App*. As such, when wanting to *use* a specific *Worfklow*, users will need to *deploy* the workflow on a connected app, which in turn will create a *Template* for this Node. When now calling this *Node*, the *App* will become a *scheduler* for this *Workflow*, making sure to *call* the mentioned *Nodes* and connected applications according to the *Workflow's* blueprint (more in *On Scheduling*).

3.2.2.3 Reservations and Provisions

Workflows (and *Nodes*) in Arkitekt are universal in the sense that they stay fully conceptual (using *Nodes* not *Templates*) and can be easily shared between different Arkitekt instances with a completely different underlying landscape of connected *Apps*. The design of Arkitekt therefore needed to strictly distinguish between *planning* and *execution* of a workflow. Building on the abstractions *Node* and *Template*, Arkitekt therefore introduces abstractions that correspond to the actual *usage* of functionality: *Reservation* and *Provision***.** Building on the middleman paradigm, they represent a *contract of use* between users and Arkitekt on one side as well as Arkitekt and the app on the other side.

When users want to run a bioimage *Task* (like convolution), they first need to *reserve* the functionality. This is analogous to booking a service; they're telling the system that they want to use a specific feature. If they have a preference, they can also specify, where they want the task to be executed (e.g., which connected *App* should do the work), or which collection of *Apps* it should be parallelized upon. *Reservations* in this way represent a contract between the user and the platform: Arkitekt will give the user direct feedbacks and inform if the desired apps become online or disconnected because of a shutdown or failure.

On the other side, all connected apps receive *Provisions* as an implicit contract between the app and Arkitekt. When receiving a provision, the app can decide to spin up a worker (more in section: concurrency), and tell Arkitekt that the provision contract is *active*, or if the app cannot spin up a worker (maybe because of resource constraints or because the user running the app does not currently want to enable this functionality) mark the contract *inactive*. Arkitekt will integrate the information of all linked up *Provisions* for one *Reservation* and then in turn establish the state of the *Reservation*.

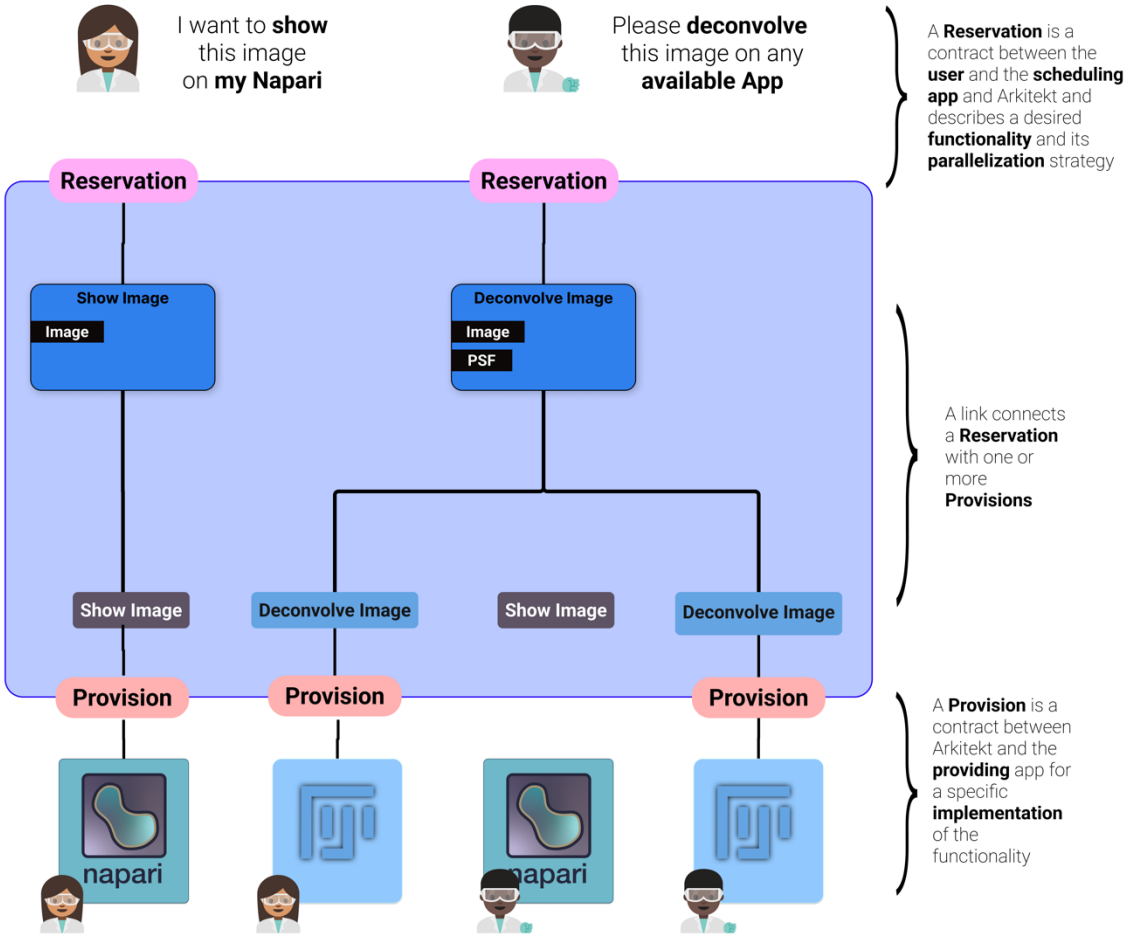

Figure 32 *Experimenter A wants to show an image on their Napari instance, and consequently reserves the functionality just liking it up to one provision of the implementation on their napari. Experimenter B does not care about which app performs their request and has a lot of images to deconvolve, they therefore parallelize their requests on multiple apps (even using ImageJ instances connected under Experimenter As account if they so allow).*

The *Provision* and *Reservation* strategy also applies to workflows, as they are as previously described just *Nodes* that rely on the functionality of other *Nodes* when *executing*. As such, when *reserving* a *Workflow*, the created linked *Provision* of the workflow will *on provision* check its blueprint for *Nodes* and in turn *reserve* them (linking them to its own *Provision*). This process establishes a dependency graph of the workflow and ensures that on *App* failure within the workflow, the information gets propagated to the user wanting to use the workflow.

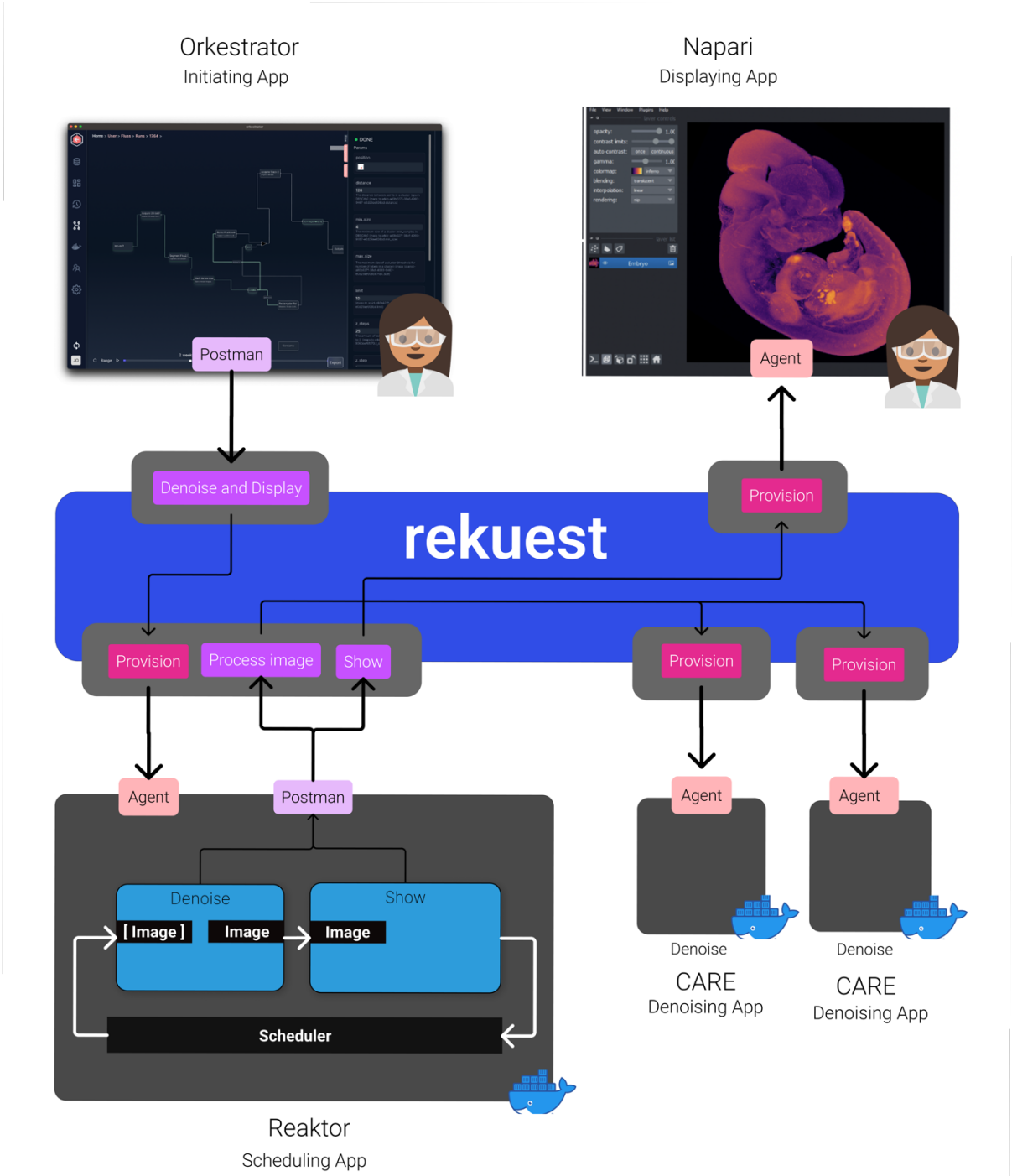

Figure 33 An exemplary dependency graph created through the reservation of a deployed workflow on a scheduling app (here running inside an Arkitekt plugin).

Establishing both *Provisions* and *Reservations* as abstractions on top of the connections between apps, ensures that problems that arise in distributed systems, such as outages in connections or failed workers, can be handled transparently for the user. They closely map to programmatic abstraction inside the Arkitekt design (see on Arkitekt as a distributed system).

3.2.3 Data Management

In addition to establishing itself as a hub for lab-wide functionality, a core design decision for Arkitekt was that it should provide an integrated data-management solution, that was easy to use and automates tedious data organization. Two main design challenges needed therefore to be addressed:

Inter-application Data Management: Build as a distributed system with multiple apps on different computers within the lab, automated workflows necessitate the moving of datasets from one app to another over network connections. This transfer needs to also include transitioning binary data and metadata of images to open formats, that are fit for the information interchange of multiple programming languages.

Central Data Management: Arkitekt needed a solution for central research data management. Data should not only move between applications but be stored in a central place, that ought to provide easy retrieval and management of the data.

To this end, the Arkitekt platform comes provided with a standalone service (*Mikro*) for data management in microscopy, that addresses both scenarios by acting as a *hub* for the apps to *store and retrieve* data from when running in a workflow, storing the binary data centrally in an object storage *MinIO* ("MinIO", Minio 2023) and feeding the metadata into a relation database *PostgreSQL* (Rowe and Stonebraker 1987).

3.2.3.1 Data Model

Mikro comes with a wide range of relational data types that support the data structures of modern image analysis. At its core, it supports bioimages through its model of *Image*, a 5D data container that supports multi-dimensional images (X, Y, Z, Time and Channels). These images can then be associated with metadata through *Mikro's* datatype of *Views*, which allows the selective association of specific coordinates inside the data such as Channel 0, or Timepoint T to metadata. The metadata models of *Mikro* are extensive and closely mimic primitives provided by the OMERO model (Goldberg et al. 2005) providing, models for *Channels*, *Antibodies*, *Objectives*, *Positions* (on *Stages*), *Timepoints* (in *Eras*), *Instruments* and more.

Additionally, its model accounts for annotations such as regions of interest (ROIs), or labels on segmentation masks, as well as generated data tables. All data types are completely relatable amongst them and make up the *Data Graph* of the platform.

The *Mikro* data-management design is adopted to work with multiple-users and beyond standard social features like comments and mentions allows for collaborative access to datasets in real-time such as in interactive online sessions where region-of-interests are marked by multiple parties.

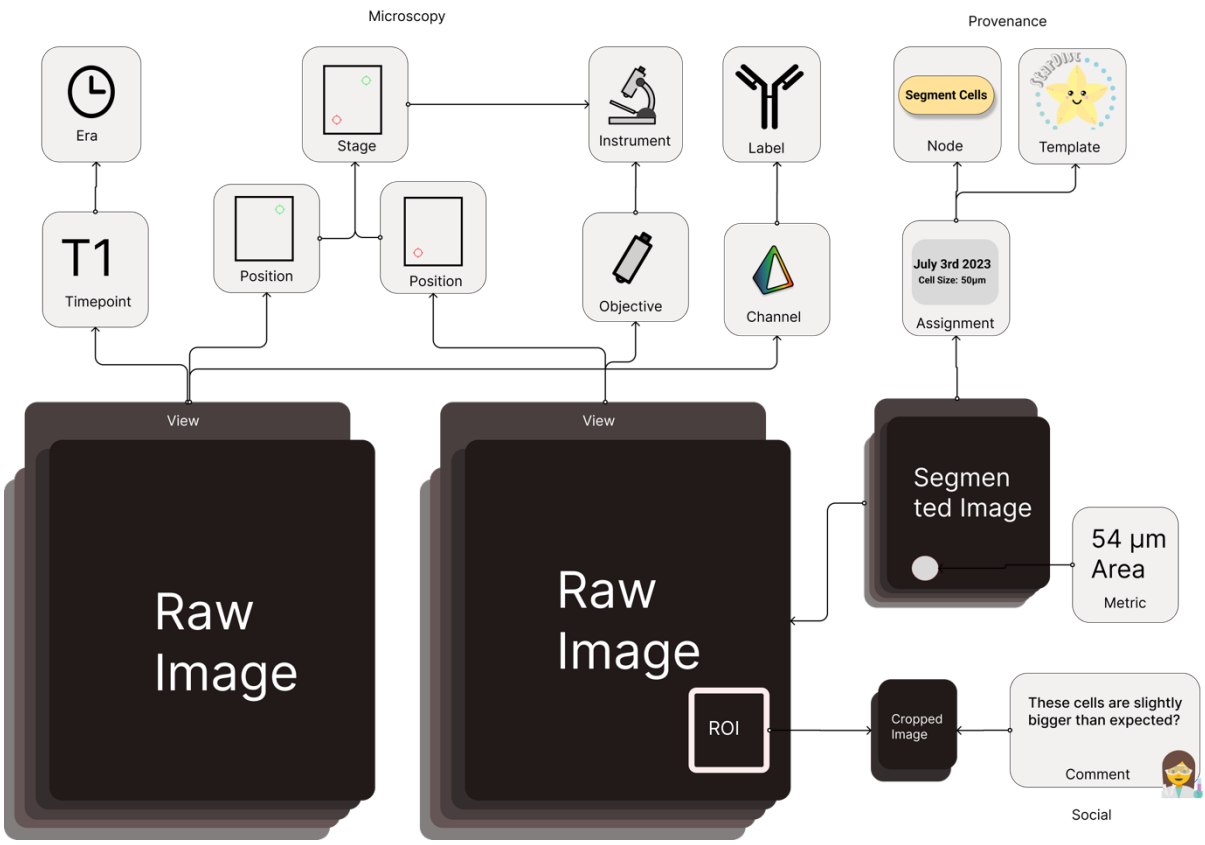

Figure 34 *Organization of the Data Graph within Mikro, (selected models) to highlight the microscopy metadata organization respecting multidimensional (multi-position, multi-timepoint data) metadata through the view model (n-dimensional selection of image stack),*

Once a datum is imported, metadata is organized and maintained exclusively in the relational database. If Datums undergo transformation through *Nodes* (e.g. deconvolution of an Image), a reference of the inputs and outputs is kept and associated with the image (more in On Provenance).

3.2.3.2 Big data functionality

To ensure the fitness of the platform for massive datasets as found in light-sheet microscopy and single molecule localization microscopy, binary data management is relying on the **Zarr** (Miles et al. 2020) standard and is packaged for *lazy data loading* over the network, which means even terabyte datasets that are store on the central storage, can be visualized with little latency remotely. An experimental on-the-fly OME-NGFF (Moore et al. 2021) conversion is also supported. For tabular data, Arkitekt uses the open parquet/pyarrow ("Apache Parquet") standard to again provide lazy-loaded access to massive data sheets.

3.2.3.3 Inter-application Data Management

For *inter-application* data management, Arkitekt uses a pass-by-reference approach, where no data is copied on the assignment of a *Task* but only a reference of the datum passed to and retrieve from the executing apps. This approach was taken to facilitate a lightweight messaging protocol that only passes small message payloads and to ensure that apps have full autonomy on how to access data in their preferred way. Apps can conditionality not retrieve items if they become no longer necessary during execution, or only download the chunk of data they will work on. This also completely separates concerns of the data and task assignment (the *rekuest* and *mikro* microservice are completely standalone).

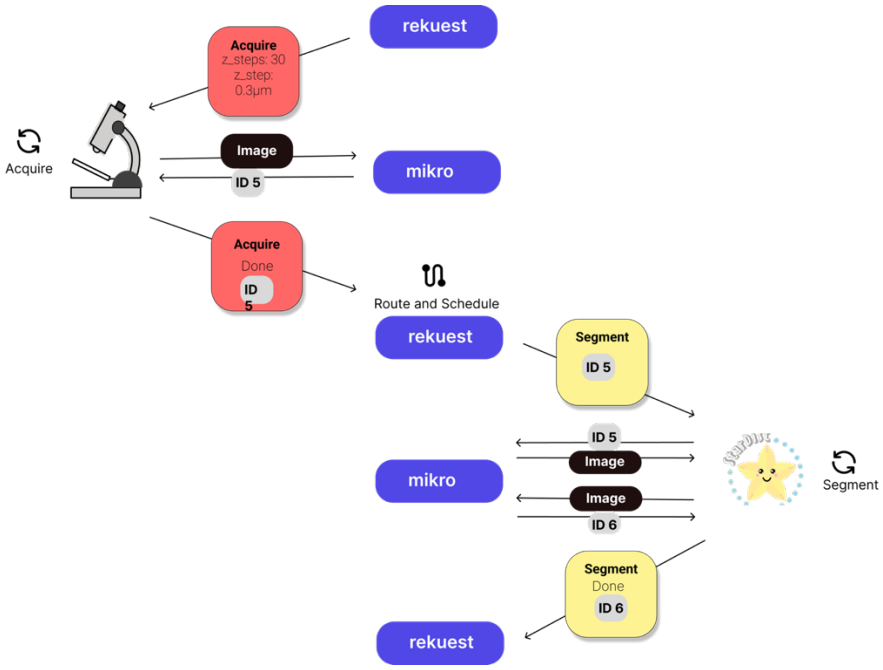

Figure 35 *Data passing by reference inside a workflow. Rekuest sends* Task *requests to the applications (here acquire and Stardist segmentation), including simple parameters such as numbers and strings (JSON serializable types) inside the requests, the microscope then saves the image directly on the Mikro service and retrieves a unique string (an ID) back, that then it passes on again to the rekuest service. Rekuest then sends the message to the scheduler, which routes and schedules a new task, returning it to Rekuest. Rekuest then forwards the serialized request on to the app retrieving the corresponding image back from the servers. This app then processes it, uploads the processed image, and gives the id back to request, continuing the circle Mikro supports single-writer, multiple reader parallelism in operations so the file, can be consumed by many workers simultaneously.*

3.2.3.4 Data Provenance

As Arkitekt was designed around the concept of *Nodes* that *transform* data, it was an early design decision that the data solution attached to the platform should use this information to establish data provenance and integrity.

To that end, the platform establishes a universal provenance mechanism through associating the bioimage *Task* (which becomes the assigned workload to a *Template* on a specific *App*), with the data that is being created throughout its lifetime. Each task on Arkitekt platform is assigned a (globally) unique id on creation. When associating this id with all data that are created during the lifetime of a task, the platform is able to link the original task (and its parameters) with the created data.

This allows the user to retrieve and inspect which parameters were used to create a specific datapoint (e.g. which PSF and how many iterations were used to deconvolve an image). In addition, as executing workflows is equivalent to assigning a *Task* to a node that then in turn assigns *Task* to other nodes, it allows the user to find all data associated with a workflow.

Task identifiers rely on the *UUID4* (Leach, Salz, and Mealling 2005) standard, which renders collisions errors with other task ids neglectable, and provides a way of verifying data integrity outside of the original context (e.g. when storing a workflow run file with all tasks separate to the original data).

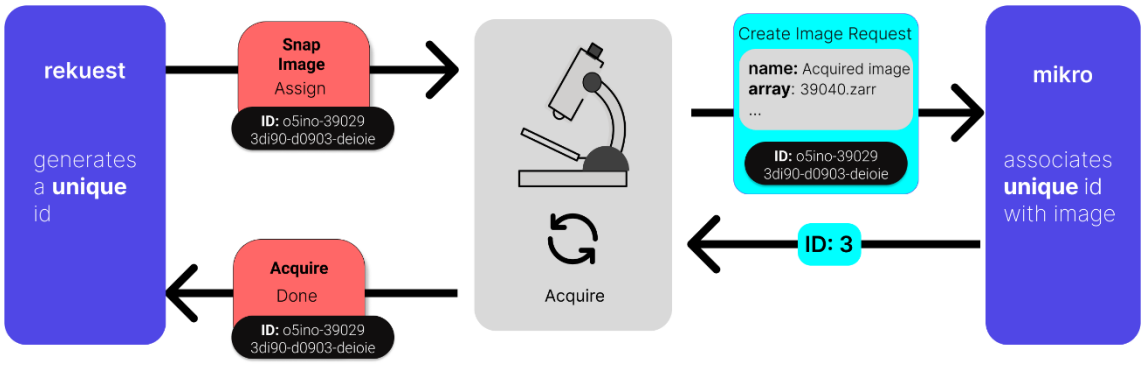

Figure 36 *Provenance. Tasks and their respective parameters are stored and assigned by the rekuest service and passed to the "Acquire"* Template *on "Micro-Manager", the microscope acquires the image, and the original task id is associated with the new image automatically (inferred from the current context of the task). The mikro service now associates the task id with the image. This design ensures low coupling and allows third party services to also include data provenance.*

3.2.4 Easy Interfaces

Arkitekt was designed to provide a unified user-friendly interface for assigning work on apps, managing data on the central storage, and creating and running workflows with apps spanning the entire lab. It was also an important feature for this interface to provide real-time feedback about the current state of workflows and data. However, this interface should not hinder the user from exploring their data in their favorite dedicated tools such as ImageJ and Napari, which themselves should act gateways to interact with the Arkitekt platform.

To this end, Arkitekt comes provided with *Orkestrator*. *Orkestrator* represents the main interface for the Arkitekt platform and can be either accessed via the browser or as a standalone desktop application that auto-discovers Arkitekt servers in the local network. *Orkestrator* allows for most common data management tasks. It comes with basic exploratory visualization capabilities, like visualizing 3D image stacks, and provides the user interface to both schedule work on apps as well as design workflows and installing plugins. As *Orkestrator* is just another Arkitekt enabled *App* it also exposes functionality that can be integrated in an Arkitekt workflow.

This section will briefly illustrate major points of interaction with the Orkestrator interface, explaining its main panes of interaction and which use-case they were adapted for.

3.2.4.1 Dashboard

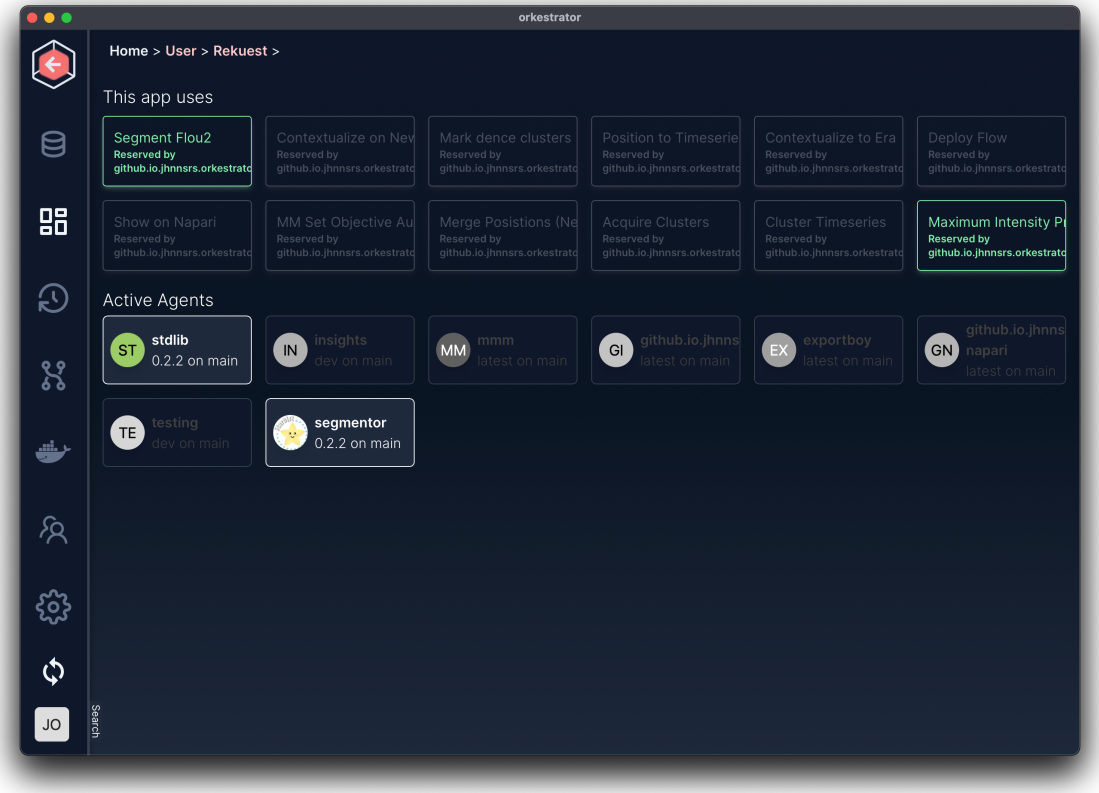

Figure 37 *The Arkitekt Dashboard.*

Building off the abstraction of *Nodes* and *Reservations*, the Dashboard is the Users main entry-point to explore the current state of all applications that are connected to the Arkitekt platform and search for functionality they might want to use. Here *reserved* functionality (including deployed workflows) of the user is monitorable, indicating which apps are currently active or need to be started in order for a workflow to run successfully.

3.2.4.2 Data

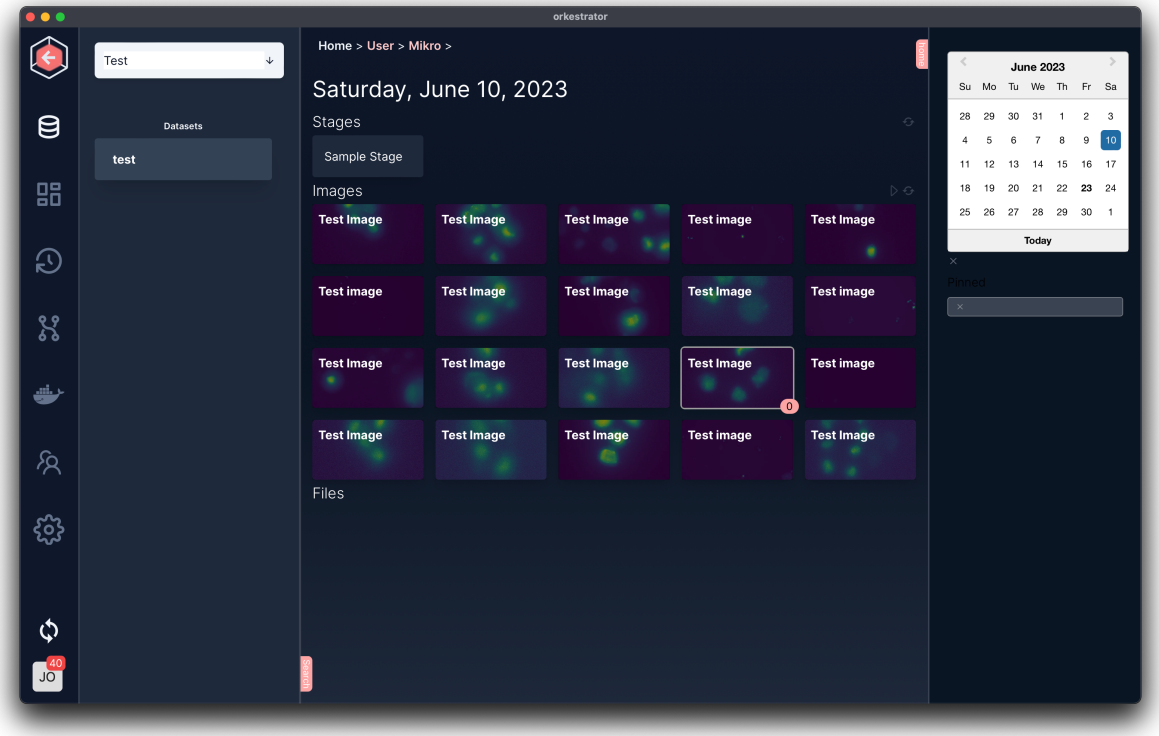

Figure 38 *The Arkitekt Data Pane.*

In the Data Pane, users can explore their data with advanced filtering. helping them to find and explore the relationship between their datapoints. It allows users to search and create arbitrary data links between two data points through drag-n-drops, which can be used to create training dataset (*Contexts*). When selecting a datum in addition to standard functionality such as comment or delete, tasks can be directly assigned by choosing additional functionality that the user *reserved* contextually respecting the inputs of the functionality (e.g. when selecting an image, only reserved nodes will appear that can take an image as an input). To allow the easy integration of new bioimage files, users can always drag and drop files into the data pane.

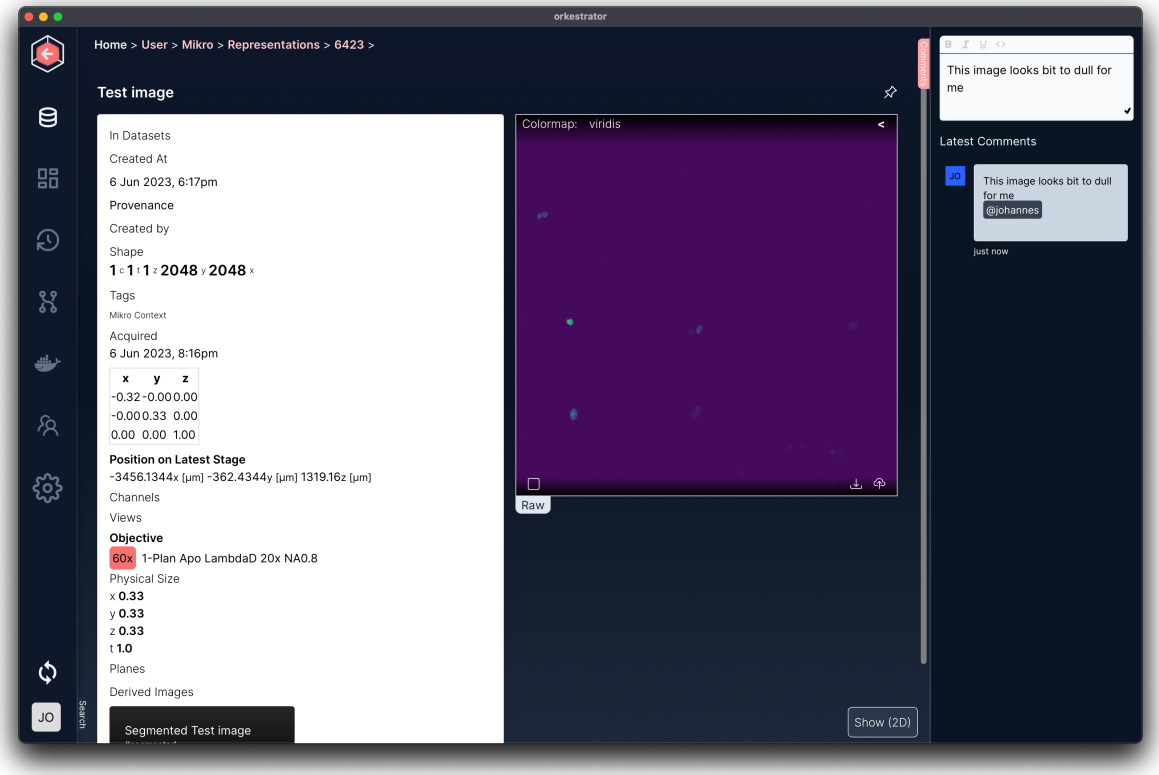

Figure 39 *An Arkitekt detail pane.*

In the detail pane of a datum (here an *Image*), its metadata as well as comments and its relations to other data (here a derived segmented image) can be explored and/or send to connected active *Reservations/Nodes* (lower right). The detail pane also provides some primitive visualization for the datum on the right, streaming actual raw data to the *Orkestrator* interface and rendering it with different color maps.

3.2.4.3 Workflow Design

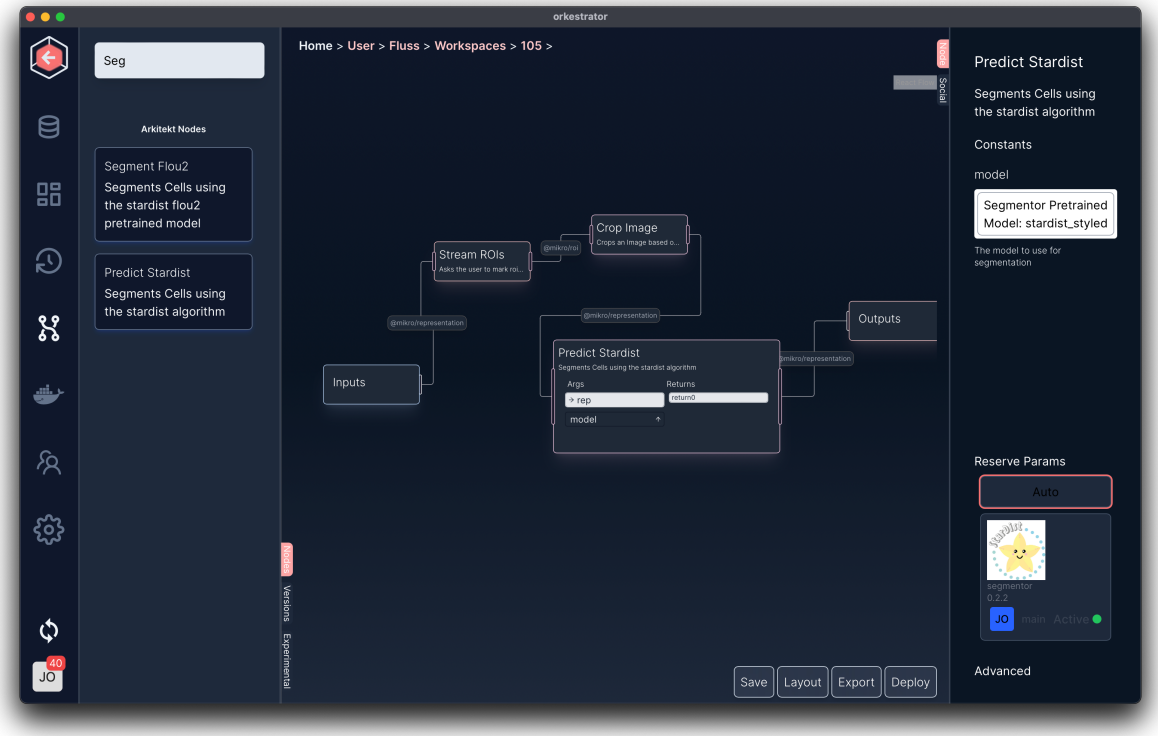

Figure 40 *The Arkitekt workflow design pane*

In the Workflow design pane, users can create *Workflows* utilizing all the available functionality through a visual editor, that allows for drag-and-drop organization and interlinking of *Nodes*. During the design process, users can inspect the *Nodes'* documentation, specify which node parameters will have defaults (upper right) and which will be global parameters of the workflow. They can adjust which *Templates* the node will map to (lower right) and set advanced parameters like the parallelization strategy.

3.2.4.4 Workflow Execution

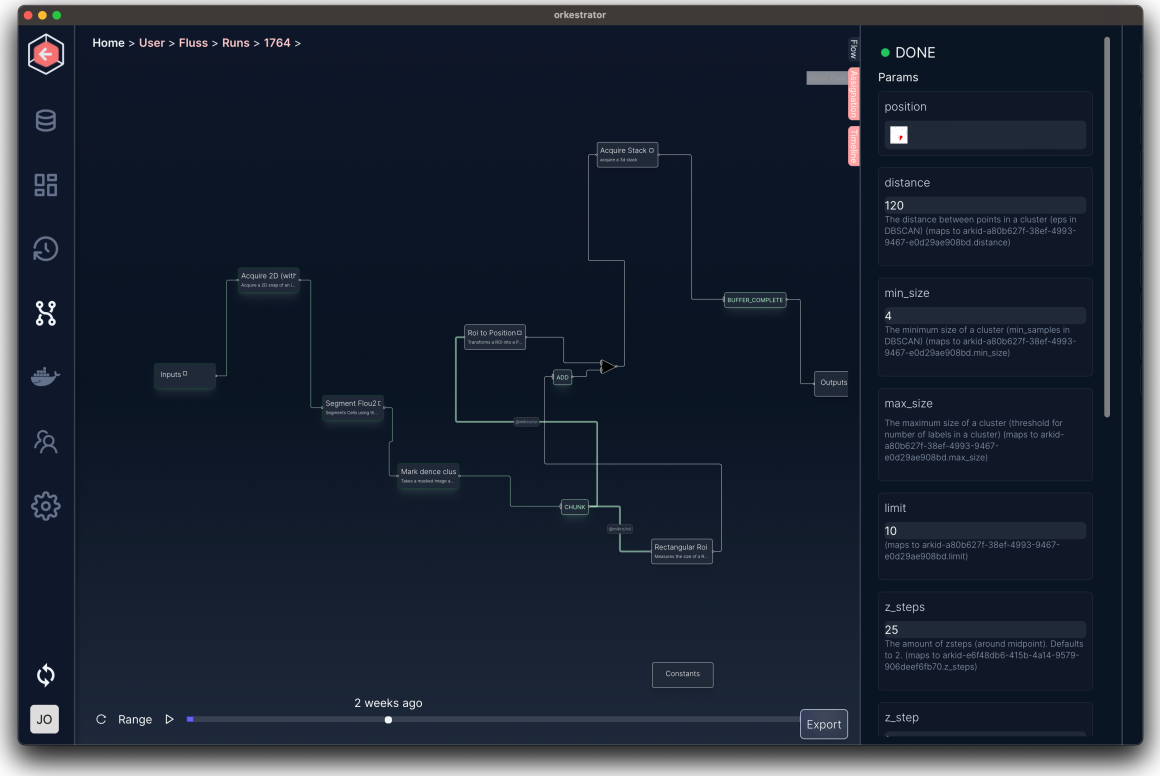

Figure 41 *The Arkitekt workflow run pane*

While a *Workflow* is being run, users can inspect a *Workflow* in the Workflow Run Interface, and asses its current state of progress or error. Data that is being produced during a workflow run, is also directly accessible in real-time, while events (like tasks finishing) are being tracked here. Users can replay a whole workflow, as well as inspect bottlenecks through the waterfall diagram (not displayed here). Workflow runs can also easily be exported and be embedded on any website.

3.2.4.5 Plugins

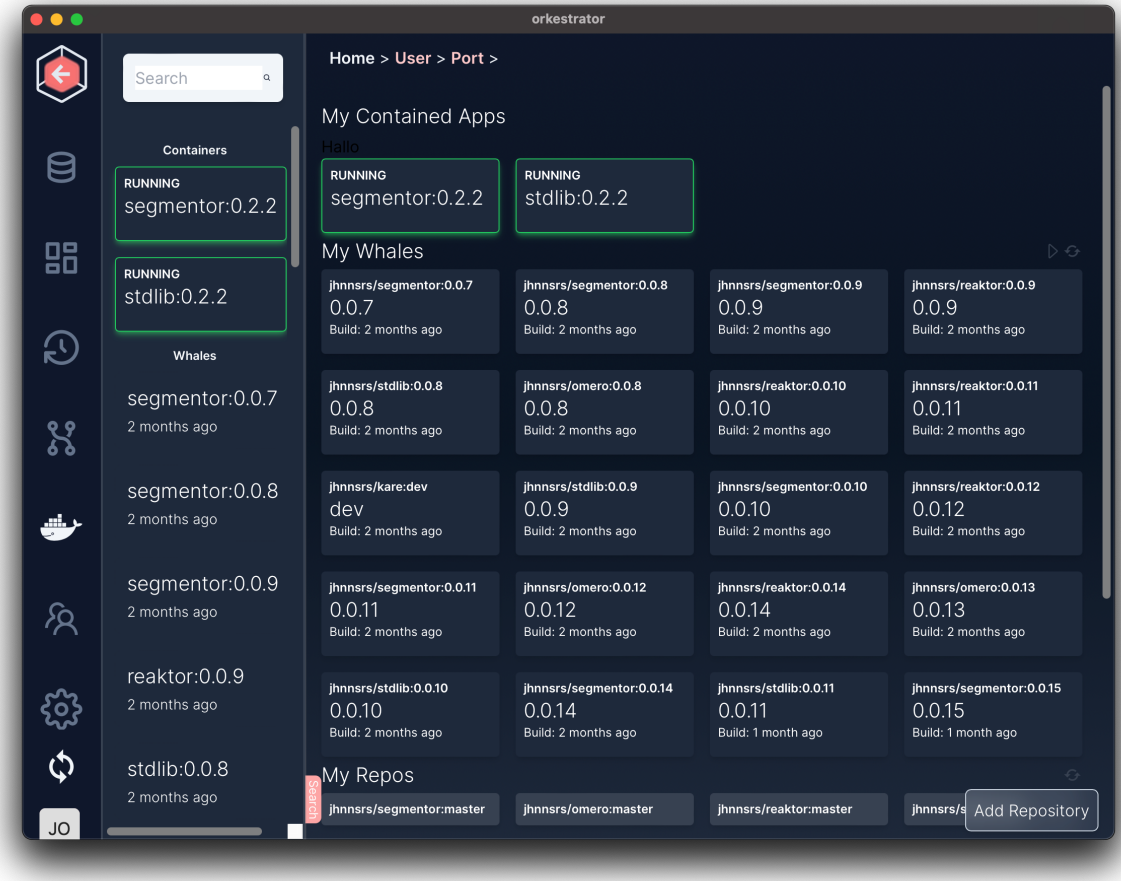

Figure 42 *The Arkitekt Plugin Pane*

Arkitekt comes with an integrated plugin system utilizing docker containers (see on Extensible). Users can use the Plugin Pane to discover and install plugins from GitHub, start and stop them interactively, manage their accessible resources (such as GPUs), as well as inspecting potential errors directly, visualizing for example the command line output.

3.2.5 Task Management

Set to deal to make applications interoperate, that share no common execution environment, may run on different computers and hardware and can potentially be written in different programming languages, Arkitekt needs to perform as a central agency in a *distributed system*. Dealing with such a system, can, when done right, lead to a fault-tolerant and highly performant environment, however, itself introduces a set of potential errors that need to be addressed.

To address these potential errors, Arkitekt adapts features from the Actor Programming Model (Hewitt, Bishop, and Steiger 1973) and treats the participating apps as independent *Actors*, that execute completely autonomously, but can communicate with each other through messages. Arkitekt task management relies here on a principle akin to the real-world task management in a team with remote workers. A supervisor or manager (a *scheduler)*, assigns a task via email to a remote worker *(the actor)* telling them to do the assigned task and give the result back *referencing* the original email/message.

This section will now delve into how Arkitekt handles the communication and execution of tasks on a connected app, and it schedules these tasks when orchestrating a workflow.

3.2.5.1 Task Scheduling

3.2.5.1.1 Communication

When scheduling tasks on another connected app, one aspect is crucial for a successful call: a communication protocol that reliably communicates the task *to* and updates *from* the executing app. This requires a form of *bi-directional* communication between the participating apps.

In the design of Arkitekt, various decentralized protocols (Direct Communication, Peer-to-Peer Networks) for this message delivery were explored but discarded in favor of the Message Broker Design**.** In this centralized system, the *Rekuest* service of the Arkitekt Platform facilitates communication between clients. Clients, such as the *scheduling* app, send messages to *Rekuest* and *Rekuest* in turn routes these messages to the appropriate destinations (other bioimage apps). *Rekuest* only acts as a *broker* and ensures reliable message delivery adhering to routing rules for the messages (Arkitekt's *Provision* and *Reservations*), which can for example route messages to another app if the preferred application fails. Additionally, it also acts as a central *finite state machine*, tracking the

progress of the task during its lifecycles: updates from the bioimage apps are being persisted directly into the database.

Given that now a single point of failure, the broker needs to constitute a highly reliable piece of software. The Rekuest service routing is hence based on an industry standard message broker: RabbitMQ (“RabbitMQ”, RabbitMQ 2023.). The introduced abstractions *Reservations* and *Provisions* directly to worker queue paradigm in RabbitMQ (with *Reservations* representing *Topics*, and Provisions: *Worker Queues*). While applications can choose to directly utilize the RabbitMQ client libraries to listen to messages, in the default setting Arkitekt proxies the protocol through *websocket* connections, ensuring reconnection (see *Reliability*).

This design decision was taken to be in line with Arkitekt’s middleman approach and trying to easily support different programming languages with “dumb” clients that could communicate utilizing a simple protocol over open standards.

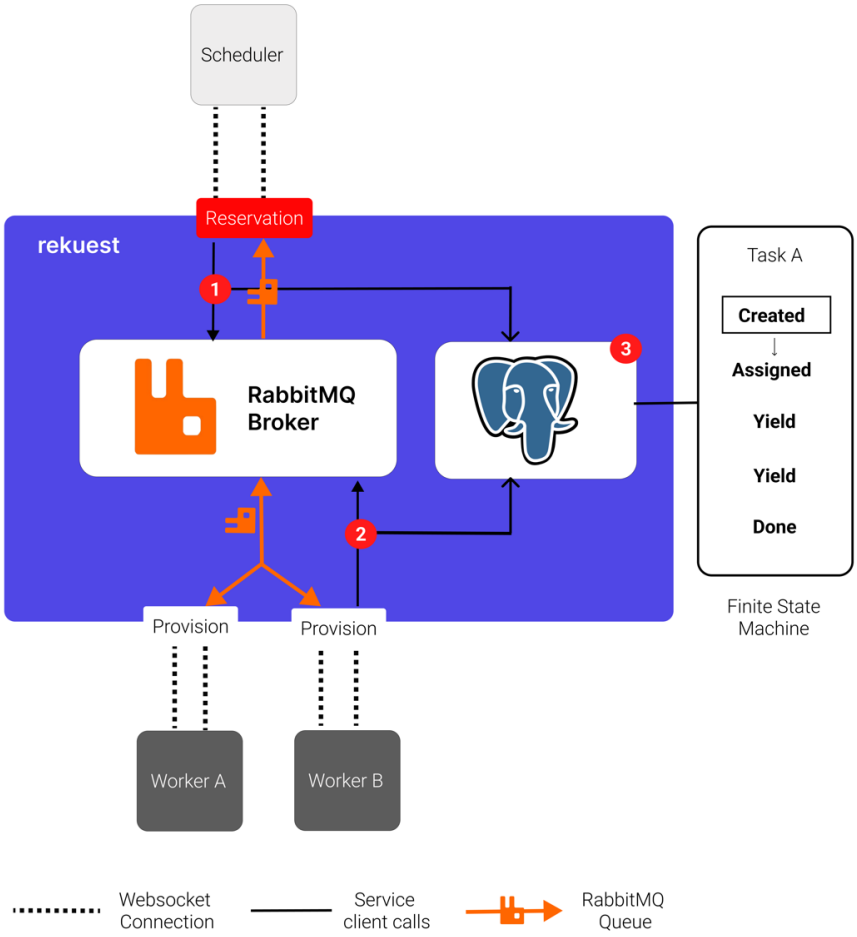

Figure 43 *The rekuest message broker design. Task gets assigned to the Reservation, which forwards the request to the RabbitMQ (1) queue and creates a new task in the database with state “Created”(3), depending on the reservation being marked as a 1 to 1 reservation (mapping only to one app) or a 1 to many reservation (mapping to multiple apps), the task gets routed to dedicated worker, which now publishes updates on the execution of task A to the broker, which routes the message back to the scheduling app and persists the state to database updating (2) the finite state machine.*

3.2.5.1.2 Execution

Once a task is communicated with an app, the app needs to make sure to execute the *Task* and in turn communicate the result back to the original sender. Importantly, to support Arkitekt design decision to fully support *interactive* applications, the execution of a *Task* needed to support two orthogonal types of code execution:

Synchronous Execution: This classic execution model is defined by top-down *sequential* execution of each code element, *blocking* on each request. This is the standard execution model in most scientific scripts.

Asynchronous / Concurrent Execution: Some tasks like marking region-of-interests on an image loaded into a GUI, require user input and therefore will *not terminate* immediately. They require some form of *concurrent* execution as otherwise a blocking call would cause the GUI to become unresponsive. This is often handled through *threading* or *concurrent programming* as the main app needs to stay in control of listening for events (such as a button being pressed).

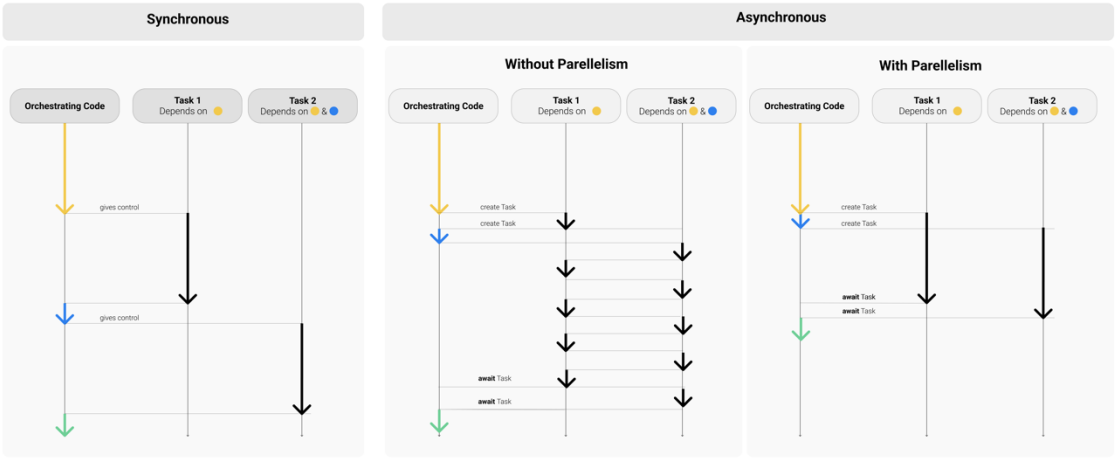

Figure 44 *Synchronous vs Asynchronous Execution, as illustrated through the interaction between orchestrating code (which would receive task from the* Rekuest *service, and or manage the event loop of the graphical user interface), and the execution of two tasks that depend on some code part of the orchestrating code to be run. In synchronous computing the control and lead are given to the corresponding task and only returned on finishing, in asynchronous computing control stays with the orchestrating code and tasks will only be given interruptible access to the computing resources, which can lead to more performant code if tasks can be executed in parallel.*

Additionally, Arkitekt's execution model needed to account for the fact that a wide variety of bioimage tasks need to be cancellable, to ensure full control over the execution (e.g., when cancelling an ongoing acquisition).

To support this dynamic execution policy, Arkitekt adopts design principles from *async/await* programming (Syme, Petricek, and Lomov 2011), wrapping them in its own abstraction where necessary (e.g. when cancellation is not fully supported by the programming language). As *async/await* is not a commonly used abstraction in scientific programming, user-defined functions are automatically wrapped (in thread pools) abstracting this introduced paradigm completely away and ensuring a familiar interface for the programmer.

3.2.5.2 Workflow Orchestration

The scheduling of tasks in a workflow requires not only a *scheduler*, that can handle task assignment, but also an orchestrating entity, an *orchestrator*, that keeps track of the state of the workflow and directs inputs and outputs. Arkitekt here double downs on the *everything is an app* principle and allows for *every app* to become a *scheduler* and *orchestrator* of a workflow, bridging it as a new *template* to the platform. As such, workflow *orchestrators* become *Actors* and follow the same principle of receiving messages with a task with inputs for the workflow and yielding the results as messages. The *scheduling actor* makes sense of the logic of the interconnected nodes of the workflows and calls apps according to the plan, keeps track of the events happening in a workflow and given the current state, makes decisions of calling other apps.

To facilitate this design, the orchestrating apps track all events happing in their workflow in an event loop and record their processing in a *Run*. A *Run* represents a complete record of what currently happens and happened during a workflow execution.
If a data event originating from another node returns or the initial assignment is redirected to another node inside the workflow, the scheduler can decide to schedule/assign the task in two different ways: *remotely* or *locally*.

In *remote assignment,* the Arkitekt platform becomes a middleman for the request, that means the task travels through the network to Arkitekt and then from there on, to another connected app. This type of assignment is the default and takes advantage of Arkitekt features that ensure reliability and automatic parallelism. However, this approach incurs network latencies in the task scheduling.

In *local* assignment, the *scheduling app* can decide to call its *own Nodes* directly, without any network request and no added latency. This type of assignment is useful for scenarios where an Arkitekt workflow is designed to manage *in-memory* transformations of data (e.g.

downloading data from the server, running multiple separate ImageJ plugins in memory, before uploading the data to the central server).

When designing the workflow, this strategy can be chosen on *on-node* basis and users can construct workflows with both types of assignment interchangeably.

3.2.6 Ecosystem integration

A leading design idea of Arkitekt was to integrate with the current bioimage analysis system from the get-go to facilitate an easy inclusion of the software and its functionality. To this end, Arkitekt comes distributed with installable applications that wrap major bioimage software (Napari, ImageJ, Pycro-Manager, Imswitch) as well as a set of plugins that wrap deep learning functionality (Care and Stardist). This section will describe the integrations for the two major software platforms: ImageJ and Napari, and lastly will list some tools that have some functionality bridged to the platform.

3.2.6.1 ImageJ

Arkitekt comes provided with an installable application *MikroJ* that acts as a bridge between the Arkitekt platform and a local *custom* ImageJ installation of the user. This bridge is implemented with the help of the PyImageJ (Rueden et al. 2022) libraries which enable direct interfacing with the ImageJ Programming Interface and shared memory access to ImageJ data primitives from python. While direct ImageJ support through a dedicated ImageJ plugin interface was considered as a design option, this path of integration was chosen to focus on the stability of the client and to easily integrate the local execution of in-memory workflows. With Arkitekt's support for *local* workflow execution, users can easily create workflows that then in turn bridge ImageJ Macros or Plugins in any Arkitekt workflow.

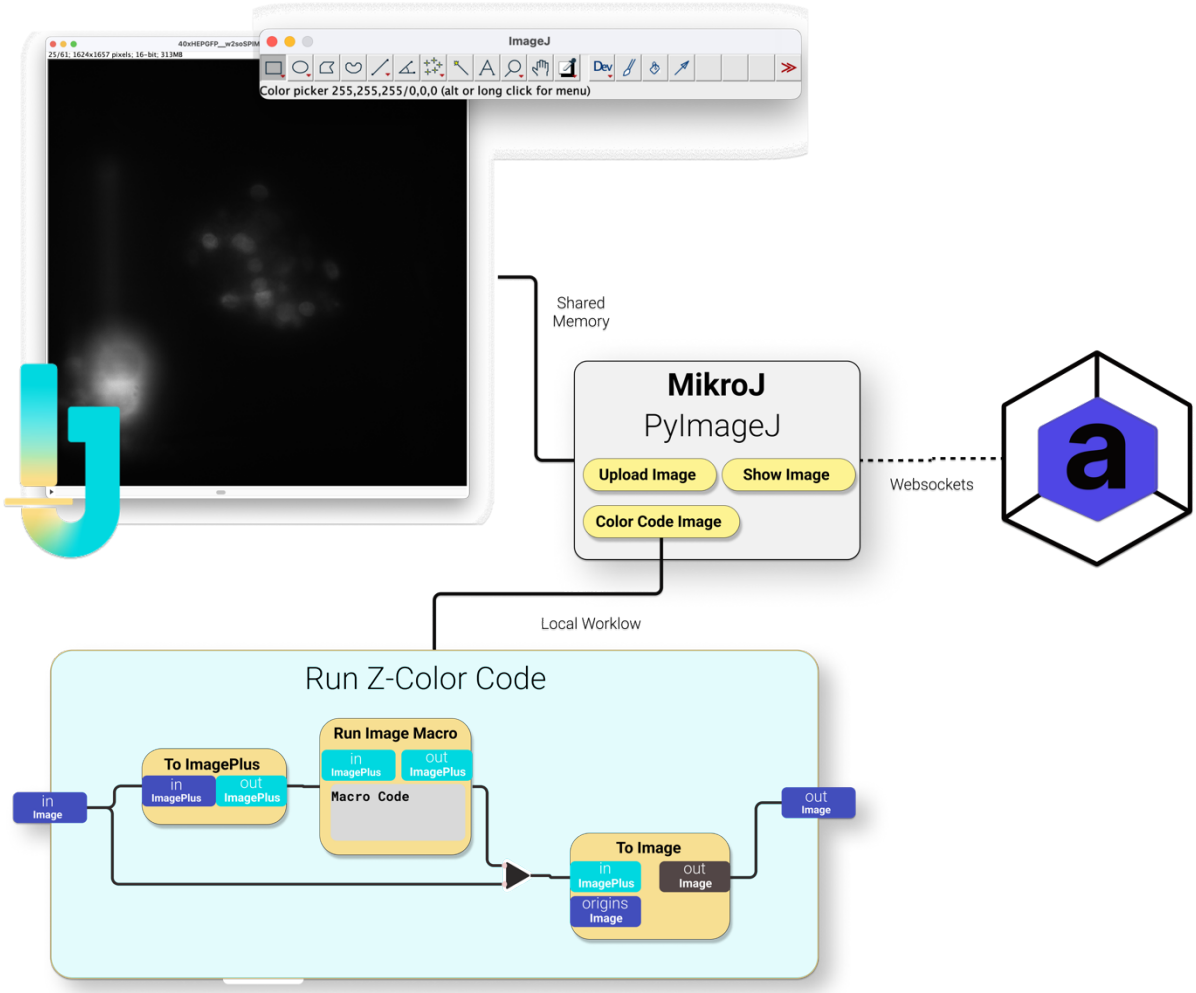

Figure 45 *The conceptual design of the MikroJ wrapper app that integrates with ImageJ. Here a Run-Z-Color Code Node, that runs an ImageJ macro to color code a z-stack, was designed and then deployed through the Arkitekt web interface, and is now registered as a Template in the MikroJ app. This workflow here uses build in functionality of the PyImageJ library to convert the "Image" that is stored on the Arkitekt platform to an ImagePlus primitive (now in ImageJ memory), runs the macro, and upload the resulting ImagePlus again to the Arkitekt platform.*

3.2.6.2 Napari

Arkitekt's software design was heavily inspired by steps taken in the Napari (Sofroniew et al. 2022) community to ensure the easy visualization of large-scale datasets in the terabyte range. As such, it became an early design idea to include Napari as a major visualization solution in the Arkitekt framework, and to support its ability to *lazily load* data from the platform.

Mikro-Napari is a Napari *plugin* that bridges core Napari functionality to the Arkitekt framework, exposing them as *Nodes* in workflows, but also allowing to use Napari directly to explore datasets and dataset relationships in its own GUI. It takes advantage of the Napari technology stack of dask (Rocklin 2015) and Zarr (Miles et al. 2020) to visualize terabyte big images that are lazily *streamed* from the Arkitekt server, mark region-of-interests on the

images *collaboratively* and in real-time (syncing new ROIS directly to other instances of Napari). Additionality it can use the associated metadata of images to recontextualize them in visualizations such as visualizing a multi-position multi-timepoint acquisition in one large “meta”-image, spanning multiple datasets.

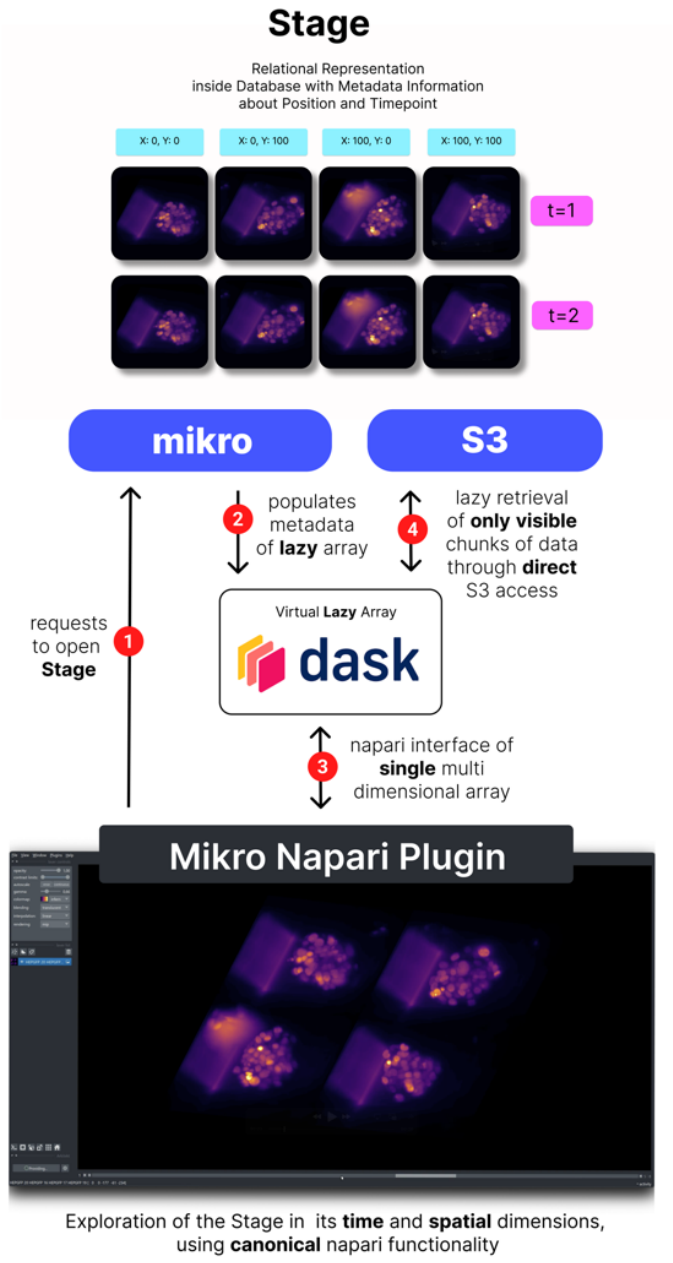

Figure 46 *The mikro-napari plugin. Through the creation of lazy loaded dask arrays that are shaped according to the metadata from the mikro subservice, Napari can directly access terabyte big datasets, recontextualizing images in their original microscopy stage context (respecting the real physical coordinates of the acquired* Image *on the* Stage*).*

Napari comes with easy and accessible programming primitives and a comprehensive API for creating additional UI Elements (Widgets), that can interact deeply with the underlying viewer. Building on this strong foundation, *Mikro-Napari* comes provides with a set of interaction and feedback nodes that can be integrated in Arkitekt workflows to facilitate *Inspect Adjust Circles*.

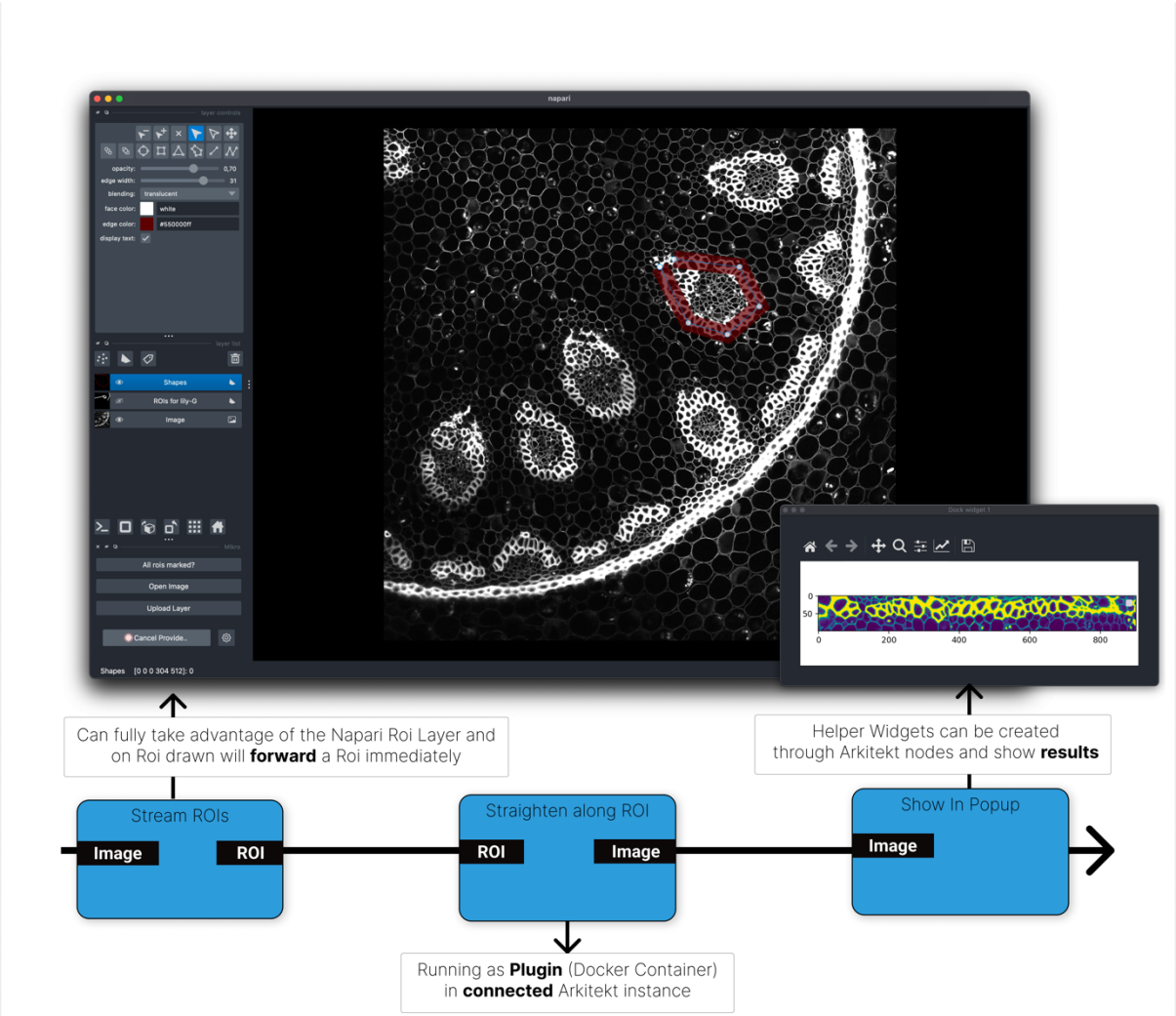

Figure 47 *Example Mikro-Napari UI Nodes as they might be used in an Arkitekt Workflow. The Stream ROI generator hooks into the Napari ROI Layer events and streams path ROIs (here a path along a histological structure). This path ROI can then be utilized inside an Arkitekt Plugin(here to straighten the image along the path) and the result of this straightening is displayed directly to the user by creating a popup inside the napari environment, letting them immediately inspect their result.*

3.2.6.3 Additional Tools

Popular bioimage tools have already been ported to Arkitekt, find their mention here.

Tool	Original Publication	Bridged Application	GitHub Link	Type	Functionality
CARE	(Weigert et al. 2018)	kare	https://github.com/jhnnsrs/kare	Plugin	Denoising
Stardist	(Weigert et al. 2020)	segmentor	https://github.com/jhnnsrs/segmentor	Plugin	Segmentation
Pycro-Manager	(Pinkard et al. 2021)	Mikro-manager	https://github.com/jhnnsrs/mikro-manager	App	Miscroscopy Control
Imswitch	(Moreno et al. 2021)	mikro-imswitch	https://github.com/jhnnsrs/mikro-imswitch	App	Microscopy Control
Napari	(Sofroniew et al. 2022)	mikro-napari	https://githu.com/jhnnsrs/mikro-napari	App	Visualization
PyCudaDecon	-	dekon	https://githu.com/jhnnsrs/dekon	Plugin	Deconvolution
Gucker	-	gucker	https://githu.com/jhnnsrs/gucker	App	File Watcher
Bioformats	(Goldberg et al. 2005)	omero	https://githu.com/jhnnsrs/omero	Plugin	File Conversion
PyImageJ/Fiji	(Rueden et al. 2022)	mikroj	https://githu.com/jhnnsrs/mikroj	App	Processing
Stdlib	-	stdlib	https://githu.com/jhnnsrs/stdlib	Plugin	Processing
Renderfarm	-	renderfarm	https://githu.com/jhnnsrs/renderfarm	Plugin	Video/Image export
Kommunity	-	kommunity	https://githu.com/jhnnsrs/kommunity	Plugin	Notification system

3.2.7 Reliability and Sustainability

3.2.7.1 Reliability

Modern workflows need to run stable and reliably. In a setting with distributed apps that connect to the platform via potentially unstable network connections, reliability becomes a key concern when not adequately addressed.

Arkitekt's design therefore *tries* to account for various potential pitfalls in the communication and recovery from errors that can emerge during *Task* assignments and *Workflow* runs, through strategies of reconnection and rerouting of tasks. These strategies are closely linked to the Arkitekt concepts of *Reservation* and *Provision* and find their explanation below.

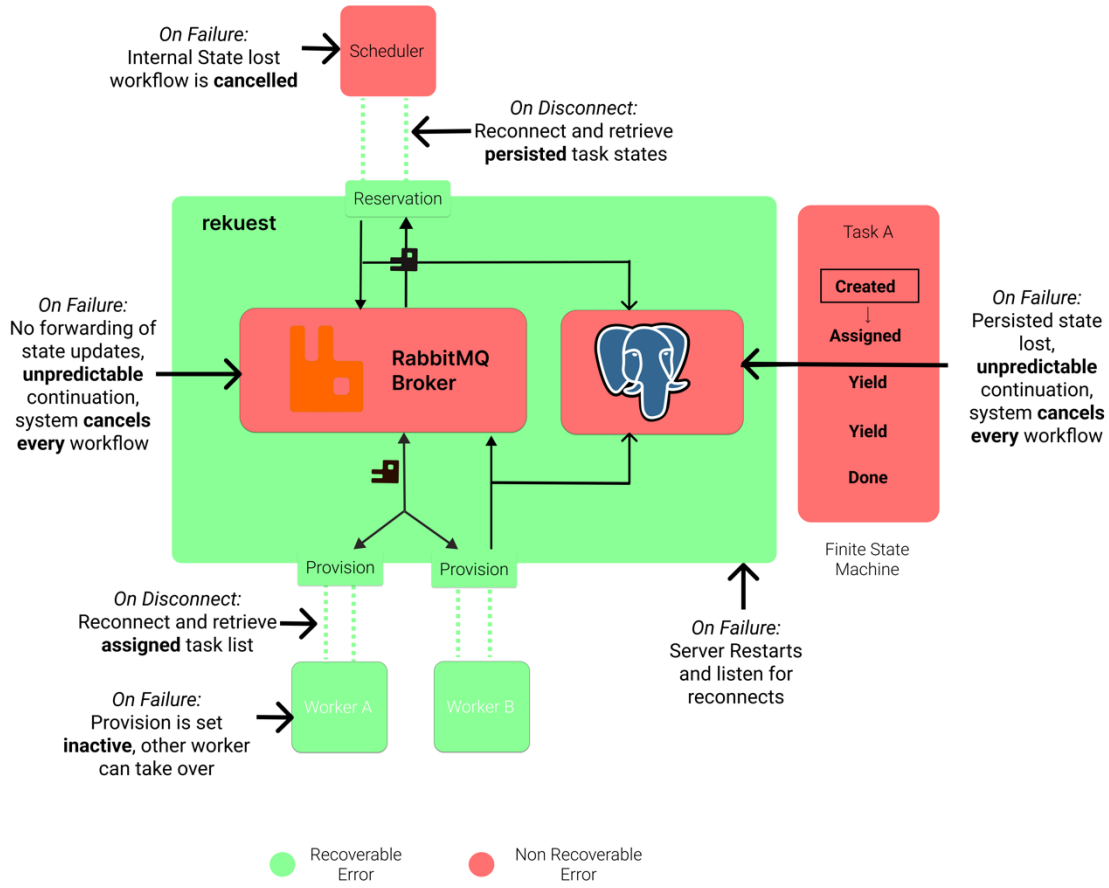

***Figure 48**: Illustration of the various potential points of failure in communication and execution in the Arkitekt system. Failure points in green are recoverable, while failures in system critical parts of the platform are marked red.*

3.2.7.2 Sustainability

Sustainable bioimage workflows rely on an easily maintainable chain of dependencies. While Arkitekt workflows are designed to be universal and allow the easy swap out of an underlying app (e.g. when updating to a newer version or the regression to an older version when said update breaks the workflow), the sustainable *implementation* of any app is inherently reliant on Arkitekt's ecosystem of libraries and this *implementation* can only be as sustainable as they are sustainably designed.

Arkitekt's design therefore emphasizes high modularity and low coupling to ensure the easy maintainability of its software stack. This holds especially true for the client interfaces, which represent the major point of interaction for the developers. Its library stack is comprised of various single purpose libraries, each *unit-tested* for their respective purpose, that can be bundled independently if a specific feature of the platform is not required for this specific application (e.g. when the app does not need to include features of microscopy data, but rather acts on other datatypes).

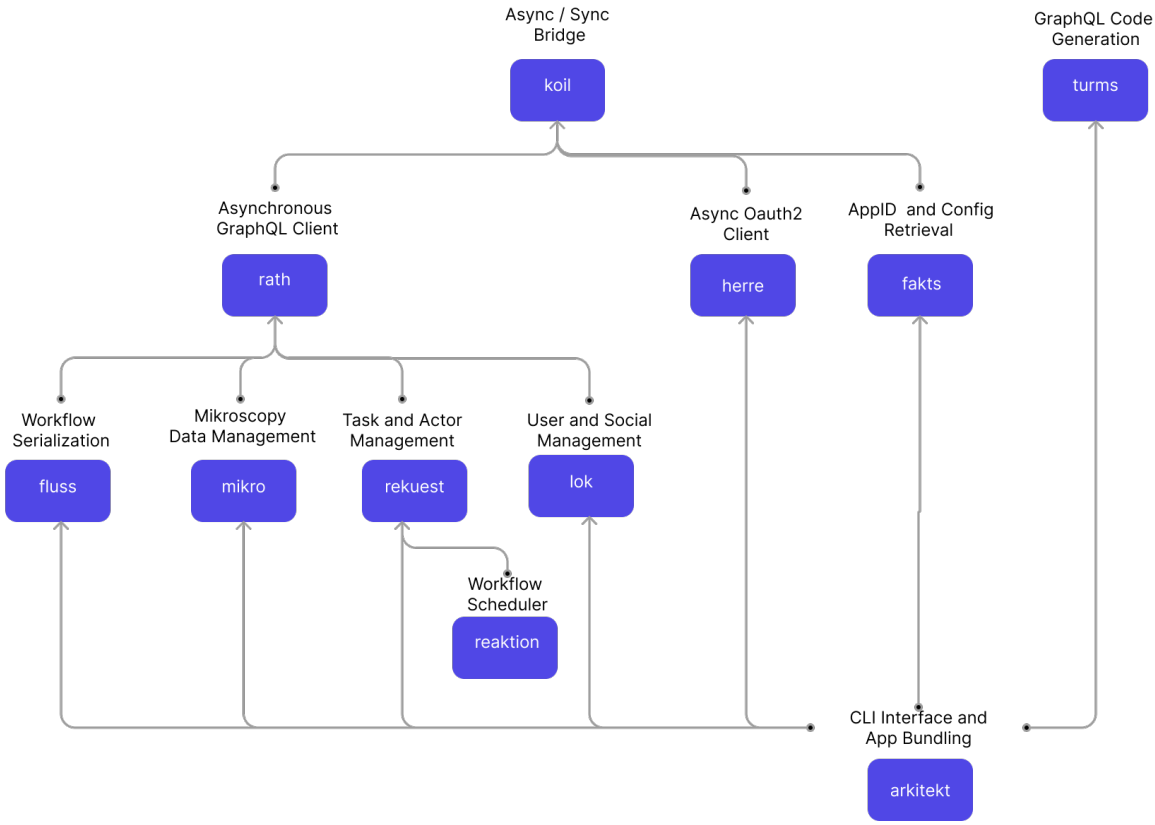

Figure 49 *The Arkitekt python library ecosystem, with their dependency chains. Each of the items represent separate python modules that are hosted on PyPI (with its source-code on GitHub) and can be installed standalone if desired.*

3.2.8 Extensibility

A core benefit of the ImageJ platform is its integration of plugins and macros that allow for the installation and extensions of itself through additional methods and tools in a user-friendly way. Given their wide adoption and respective popularity, a set design goal was to provide similar primitives in the Arkitekt framework.

Arkitekt *workflows* conceptually overlap with macros, as they are mainly used for automation and small scripts). Plugins, providing the actual algorithmic backbone of a workflow, are finding their equivalent in Arkitekt *Apps*. As installation, and especially maintenance of various applications on a computing resource however is a tedious job, Arkitekt needed to provide a plugin system that allows the user-friendly installation and management of *integrated* apps.

Building upon the *everything is an app* paradigm, an Arkitekt *plugin* is nothing more than an application that runs virtualized in the same execution environment as Arkitekt. As such, the developmental approach to a plugin is the same as when developing a standalone Arkitekt application, and developers can use arbitrary outside dependencies such as CUDA in their developmental application. In an additional build step (using for example the *arkitekt cli (see on Developer Experience),* the application will then be bundled with its dependencies into a docker container, and in a second step can be published on an associated git repository.
On the other side, users can then easily point to the publish GitHub Repository and through a one-step installation process, accepting the requested rights of the application (see *on Secure*), install the application in their local Arkitekt instance. Arkitekt will then consider this containerized application just as any other application, and its *Nodes* (exposed functions) are available in workflows, and can be called from any other app. To ensure easy upgrade and maintenance of these *integrated* apps, Arkitekt will regularly check the repository for updates.

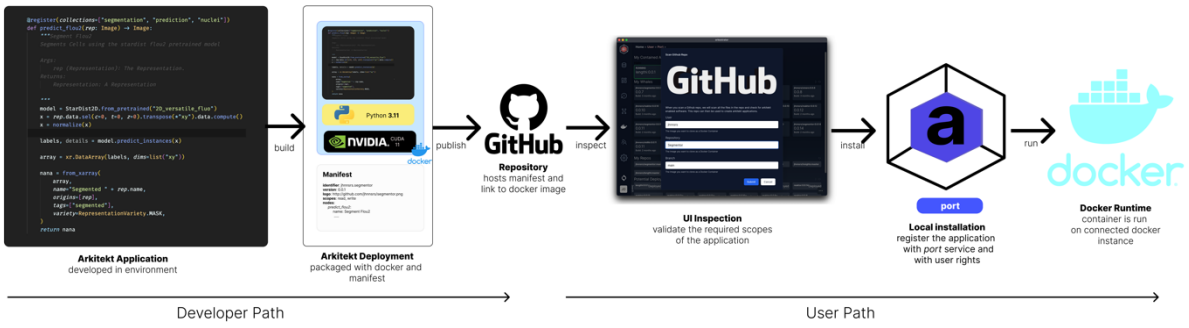

Figure 50 *Arkitekt plugin building and installation process on the example of the included segmentor plugin, illustrating both the developer path to an Arkitekt Plugin as well as the user path of installing said plugin.*

3.2.9 Open Platform

Arkitekt was designed to be open and expandable. Building on the middleman approach, it was an early design decision to establish Arkitekt as a framework, rather than a tool. This entailed for Arkitekt to adapt a *developer first* or *API first* (Garvey 2023)approach. These approaches both describe an approach to software design that seeks to first establish a comprehensive open API design before implementing features that are hidden away from developers (e.g. Product Features that are not accessible from the outside). In developing this API, it was paramount to adopt open standards that are widely used and that fit well for the Arkitekt paradigm of lab-wide task assignment and scientific data exploration.

Arkitekt therefore exposes all its functionality through a set of GraphQL APIs. GraphQL is an API protocol developed by Facebook to facilitate a one-stop-shop and developer-friendly retrieval and manipulation of data with a high level of relationships, such as their friendship graphs (“GraphQL” , GraphQL). It was designed to overcome issues in other API protocols such as REST, where when trying to explore highly relational data, dedicated and separately orchestrated requests to the backend needed to be performed.

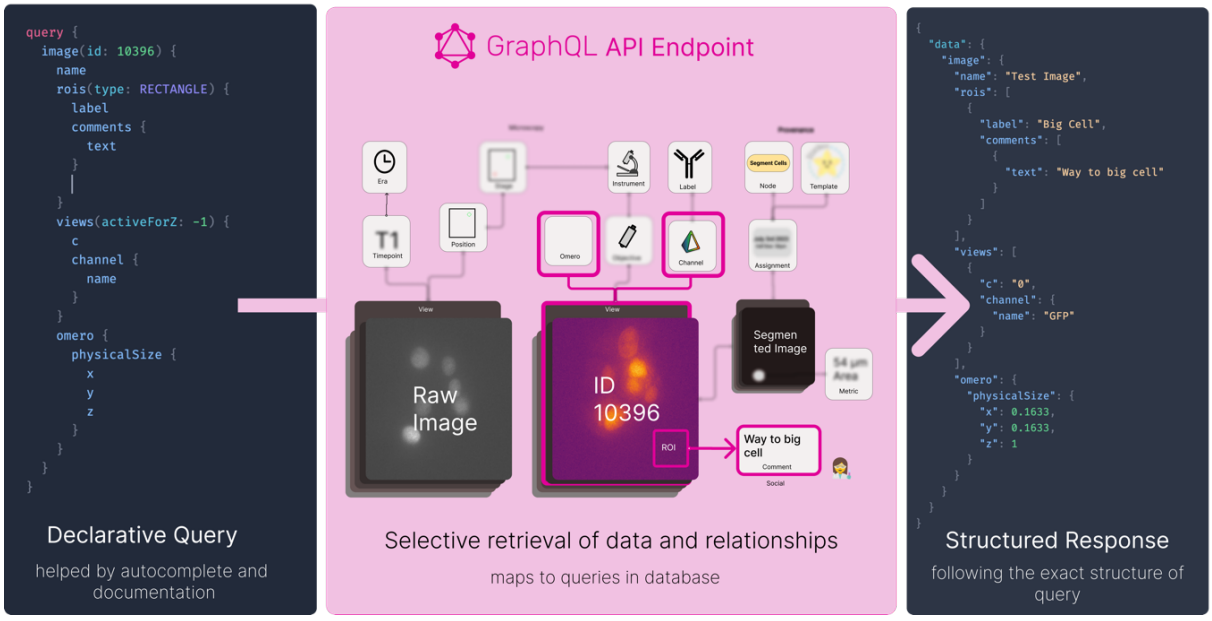

Figure 51 *The Mikro GraphQL advanced programming interface which allows for the structured retrieval of associated metadata and relational information.*

To ensure modularity, each service of the Arkitekt platform exposes its *own* GraphQL API, with API methods and data models that are specific to its use case. Users can explore all of the APIs functionality in an online *playground*, built on GraphQL, that supports the design of a data

query with autocompletion of available properties as well as a full side-by-side documentation of available relationships. As GraphQL provides a *typed* schema of its relationships, users can rely on code generation libraries such a *graphql-codegen* or *turms*, to map their query directly into typed JavaScript/Typescript and Python code and appreciate type-safety at each level of interaction with the platform.

3.2.10 Developer Experience

Given the vast challenges that developers face when developing applications today, Arkitekt is not only focused on making existing tools interoperable but was designed to create a bespoke developer experience, trying to minimize the friction when developing image analysis scripts and bringing them easily from a local setup to a published application.

Arkitekt therefore provides a comprehensive Python and Typescript client, that enables the developer to both *use* functionality of the platform and to *add* to it. The client libraries have been designed to reflect why the respective developer might have chosen the programming language.

3.2.10.1 Python

The Python library in Arkitekt is the most comprehensive client library of the Arkitekt framework. It provides convenient methods to access all the functionality of the platform. Developers can easily call Arkitekt connected apps through a simple *use* call, or as Arkitekt concept of *Nodes* closely resembles functions and generators in Python, adapt their own function to be run on the platform by decorating them with one line of code *(@register*). The client then helps to automatically infer the inputs and outputs from the function and its type annotations, creating an Arkitekt *Template* and *Node* respectively. When running the app, the same client library handles the connection logic (handling authorization, authentication and automatic reconnect) and ensures a seamless transition of the function to the Arkitekt platform.
Once the functionality is connected to Arkitekt, developers and users alike can use highly customizable, autogenerated and user-friendly widgets on the WebUI to interact with the function (similar to the experience provided by Magicgui (“Magicgui”). Importantly they can also directly use their newly provided functionality in Arkitekt Workflows.

Figure 52 *Side by side comparison between a Galaxy Tool Wrapper vs Arkitekt's inline code approach.*

The Arkitekt python client is backed by popular libraries of the scientific stack, to ensure familiar developer access to image data (accessible through *numpy*) and data tables (through *pandas*). Additionality Arkitekt's facilitates common integration patterns of application with the platform in two run modes.

3.2.10.1.1 Script Mode

Script mode was designed for *integrated* development with the Arkitekt platform, and to take a project from early development to a published, easily installable, and reproducible application without extensive programming knowledge. To this end, the Python package comes with a *command line interface* reminiscent of tools like *conda* ("Anaconda", Anaconda Inc.), that can be used to control every stage of app development.

It enables patterns like *hot-module-replacement* (an application (used in a workflow) can be replaced by a new version through a simple save), *scaling* (scaling the application to multiple

instances to automatically parallelize) and *bundling* (packaging the app to a docker container that then can be installed on any other Arkitekt services).

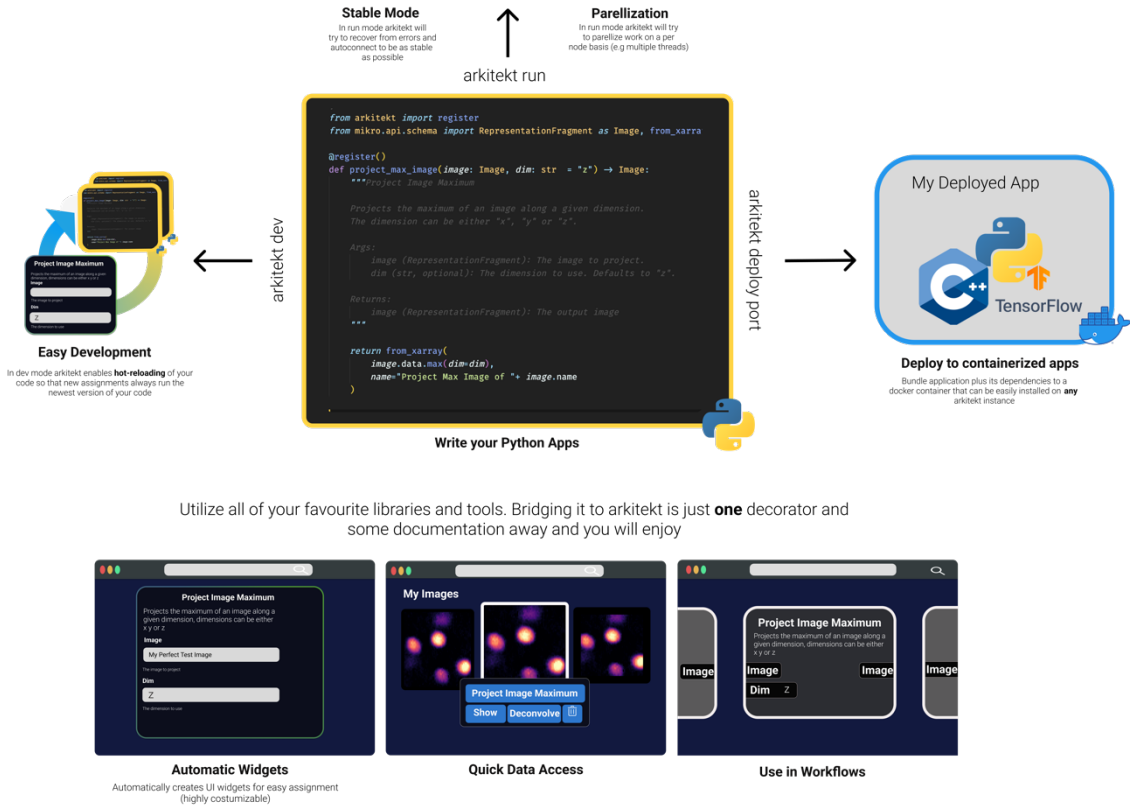

Figure 53 *Arkitekt Script Mode, which allows for the easy development and transition of python function into the arkitekt ecosystem, complete with hot-module replacement, automatic widget creation and deployment.*

3.2.10.1.2 Plugin Mode

In plugin mode, the Arkitekt does not directly control the lifecycle of the application but rather provides easy ways to integrate Arkitekt functionality as a *plugin* in a bigger application. It is the default when an application wants to provide Arkitekt functionality conditionally but would still work separately from it or when it needs other packaging strategies. Example use-cases include graphical user interface applications, such as the *Mikro-Napari* napari plugin.

3.2.10.2 JavaScript/ Typescript

JavaScript (with its typed dialect Typescript) is the most utilized programming language in the world (“Stack Overflow Developer Survey”, StackOverflow). It dominates the web where in conjunction with HTML and CSS, where it is the primary language to design websites. Web frameworks like React (“React”), that are built on top of JavaScript, allow for easier integration of interactive elements in modern websites and have emerged as the standard for modular

web development. As such Arkitekt itself is built around this technology and utilizes React for creating its main web-interface *Orkestrator*.

To facilitate the easy development of additional custom websites that can interact with the Arkitekt platform, and to allow developers to easily create new forms of visualization inside the Arkitekt ecosystem, Arkitekt comes provided with a comprehensive Typescript client that enables webapps to interact with the Arkitekt services and use its functionality easily. Through the "*@jhnnsrs/arkitekt*" package, developers can easily use provided UI elements to connect their webpage to a locally running or remote Arkitekt server. They can then rely on a comprehensive library with interfaces like the python client, to provide functionality to the platform as well as use Arkitekt connected functionality in their webapp.

3.2.11 Security

A core concern in the development of the platform was to provide an interface for *sharing* and *accessing* data, following principles of FAIR (Mitchell et al. 2022) data management. This required not only to open Arkitekt to one user and his connected apps, but establishing Arkitekt as a multi-tenant/multi-user platform that allows fast sharing with users *within* and *outside* of the lab.

With opening the platform to multiple users and potentially the wider web, comes the burden of authenticating users (*Authentication*) as well as managing the respective permissions of the user (*Authorization*). Only through a secure implementation of these security mechanisms users can rest assured that colleagues or unauthenticated attackers do not accidentally, or maliciously, manipulate their data.

Additionally, as Arkitekt was designed to provide a framework to deal with highly unstable environments, such as new analysis scripts, or plugins that are installed from GitHub repository, it needed to ensure that these scripts do not perform actions outside their original intent. (e.g., accidentally deleting data instead of creating new data).
Arkitekt therefore adapts the OAuth2 Standard (Hardt 2012) (in conjunction with OpenID connect), with signed JSON Web Tokens (M. Jones, Bradley, and Sakimura 2015) to ensure Authentication and Authorization on both the *Application* and *User* Level. This adoption means that when a user is requesting to use an app with the current server, the server will ask the user to authenticate itself and then in a second step authorizes the app's request for permissions ("scopes"), ensuring a double layer of protection.

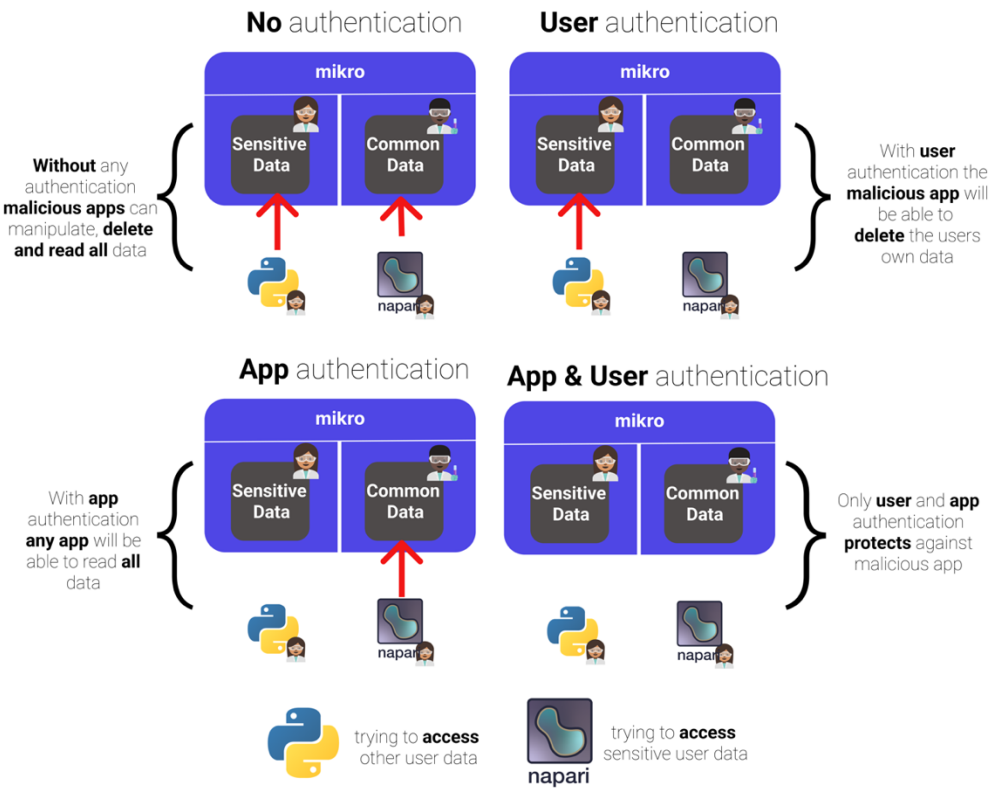

Figure 54 *The problem of trust, illustrating the double authentication mechanisms that Arkitekt introduces. vs unmitigated strategies where apps and users lack an identification mechanism. Here malicious apps try to access restricted sensitive user data or interfere with other user data respectively.*

Arkitekt's security protocol is implemented within the standalone service *lok*, that can be used to provide authentication to other third-party applications, seeking to interface with the platform. Its function is highly interconnected within the platform and its services, as for example the concept of Oauth2 clients provides the core foundation for the Arkitekt concept of *Apps*.

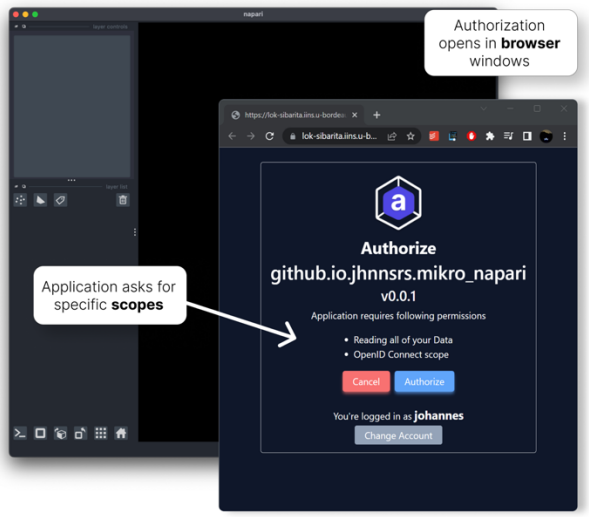

Figure 55 *The Authentication flow exemplified by Napari requesting access to the platform,*

3.2.12 Portability & Installation

Arkitekt was conceptualized to be easily installable by non-experts on a single computer, but also to be adaptably deployed on multiple computers on an institute level or in the cloud. This required Arkitekt to support a variety of underlying hardware and their abstractions (eg, *x86-64Bit* architecture vs *arm64 bit*, *headless* server install vs *GUI installation*).

Arkitekt was hence written as a *cloud-native* microservice platform (“What Is Cloud Native”, Google Cloud 2023) based on containerization software, which yielded an inherently cross-platform software that is especially suited for the cloud and institute deployments. It can easily be deployed and scaled to multiple instances on the cloud through Kubernetes (“Kubernetes Documentation", Kubernetes), the industry standard for *cloud-native* applications.

The non-expert installation, however, stood as a core challenge of the Arkitekt platform, as there is not a big ecosystem of software tools that aid in bridging *cloud native* applications to consumer devices like a Windows Machine or Mac Desktops. To overcome this, Arkitekt comes provided with a small cross platform application that acts as both the installer and admin interface for the Arkitekt platform: *Konstruktor*. *Konstruktor* can be easily installed through its installer provided for all major operating systems (Linux, Mac, Windows) and its main purpose is to thinly wrap *Docker Desktop* for non-expert users.

Docker Desktop (“Docker Desktop” 2023) is an application that enables the local installation and management of containerized applications and is targeted toward developers. With *Konstruktor*, users will be guided through an installation wizard, and once installed can use *Konstruktor* to start, stop and update the platform. Additionally, *Konstruktor* provides a one-stop-shop solution to also easily install other open-source community projects like Jupyterhub (“Project Jupyter” n.d.) on the machine alongside Arkitekt.

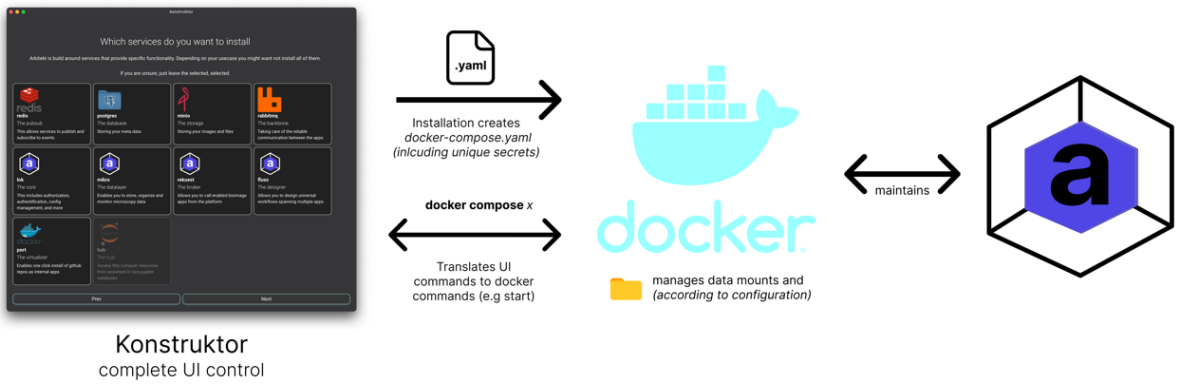

Figure 56 *The Konstruktor installation management software, that abstracts the underlying underbelly of the arkitekt platform. Users will only ever have to interface with the Konstruktor interface to install and maintain the platform.*

3.2.13 Administrability

Arkitekt aims to be installable in different configurations from a single one-laptop development setup to a system that connects apps on a whole institutional level and interfaces with cluster resources. Additionally, it was designed to provide the flexibility to allow for the modular transition of the platform to be hosted outside of the lab (e.g. storing the binary data on an S3 server on the AWS Cloud.).

With a system potentially scattered in multiple different locations, Arkitekt needed to provide a user-friendly, easy and safe way for users and their apps to automatically detect the desired configuration and to seamlessly retrieve the locations of all platform modules, without any level of manual interventions (such as noting down IP-addresses).

Arkitekt's design also needed to account for larger deployments that often require additional fine-grained control over which users can access which resources (e.g., which users will be able to access a SLURM cluster) or which feature set of an application is available (e.g., restricting student-accounts to a teaching version of a larger software package).

To facilitate this administrative top-down configuration of the connected applications, Arkitekt establishes the "*Fakts*" protocol, which aims to automate this process. In this protocol a local *beacon* advertises a configuration endpoint over the network that then all apps can automatically discover and connect to, to retrieve their respective configuration automatically. Facilitating fine grained control, administrative users can then define arbitrary complex routing rules which configuration fits best for the connecting user on the application and the location, and automatically send these parameters to the apps.

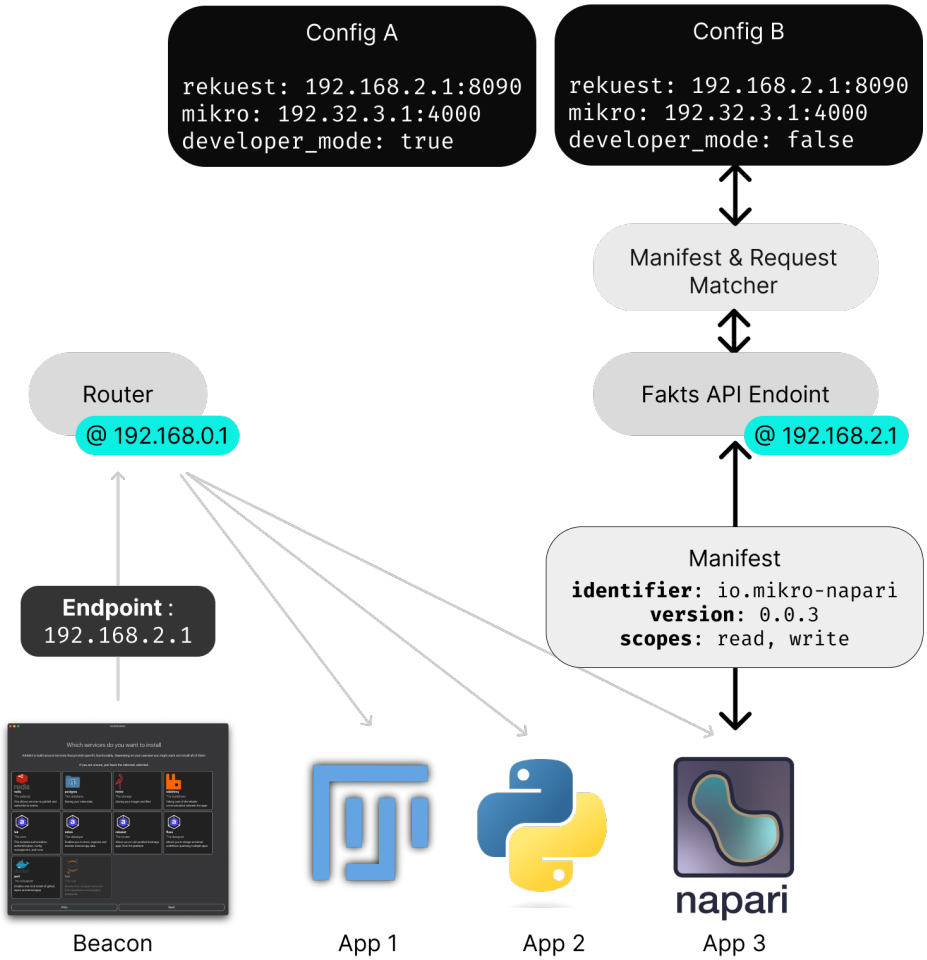

Figure 57 *The Fakts protocol: A beacon advertises the Fakts endpoint to all apps in the network through a UPD broadcast, apps can then choose to connect to the connected endpoint by providing a manifest, will be exchanged for a unique token (if the user allows this app to connect) This token can then in the future be used to retrieve a configuration specific platform feature such as which services the app should connect to but also app specific configuration like if the app should allow advanced features.*

4 Validation

Having thoroughly discussed the implementation details of the Arkitekt platform in the last section, this section now validates this implementation on a selection of modern bioimage workflows.

It will introduce three workflows, representative of the wider trends in the analysis of bioimage data, that have been chosen to validate the implementation of the Arkitekt platform, introducing methods and results inline.

4.1 Workflow I: Bridging the ecosystem:

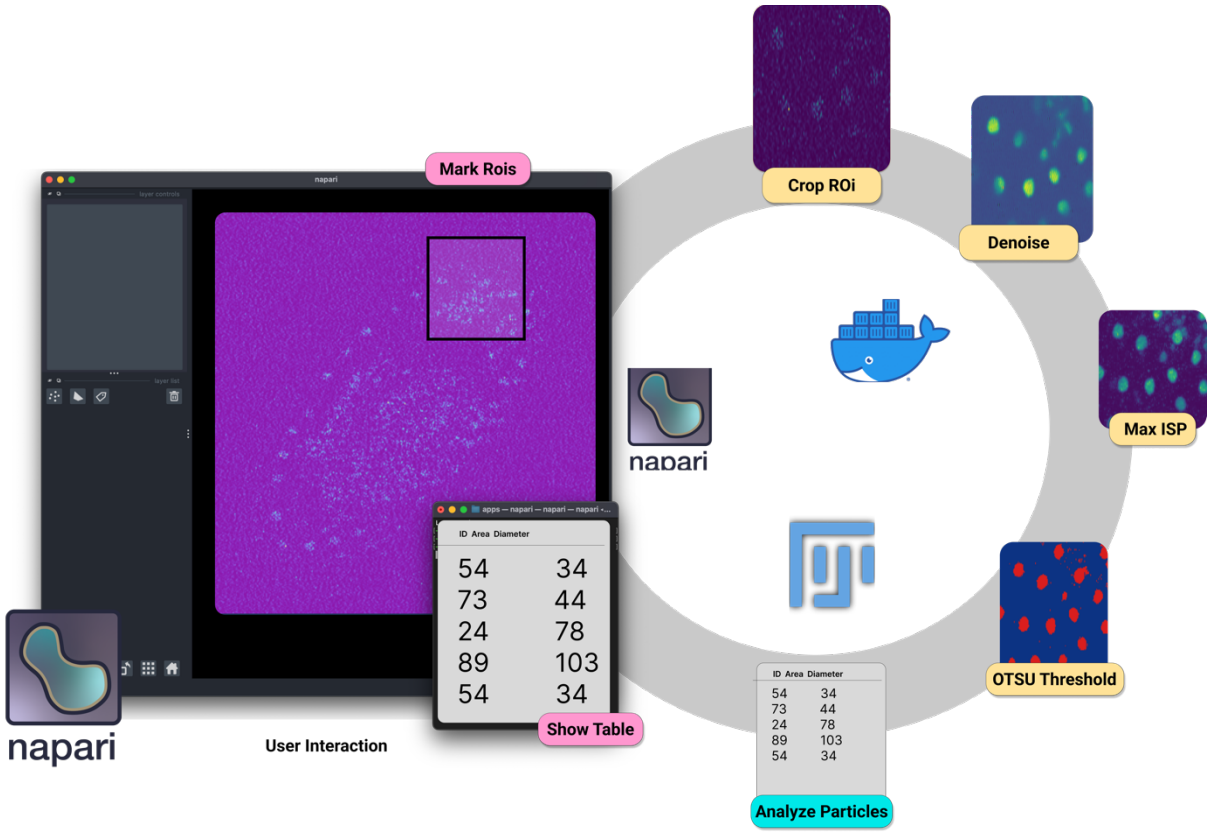

Figure 58 *An exemplary Arkitekt workflow bridging multiple bioimage apps*

In this example, Arkitekt is applied to a representative workflow in the scenario of *More Data Faster*. Here a workflow was designed for the analysis pipeline of a publicly available dataset, *Tribolium* (Weigert et al. 2018). In this workflow, the noisy fluorescence image stack was visualized in Napari, and a user prompted to define region of interests. The marked ROIS where then streamed through the pipeline, cropped from the original image and denoised through the CARE deep-learning-based denoiser. They were then projected and binarized

using OTSU auto-thresholding via python scripts running as Arkitekt plugins, and finally analyzed through the Particle Analyzer of Fiji. The quantifications returned from the Particle Analyzer were then wired back to the user in Napari and visualized on the web-interface. Arkitekt and the orchestrated tools were run on a single Linux desktop computer, but in an additional step the whole environment was transferred to a windows machine.

4.1.1 Methods

4.1.1.1 Installation

Docker Desktop V20.04 was installed on a desktop computer (Windows 10, Nvidia GPU 2080Ti, Intel i9 CPU, 1TB SSD). The Arkitekt installer (*Konstruktor*) was download from its GitHub Repository (https://github.com/jhnnsrs/konstruktor) and started. The platform was installed following its guided installer, choosing a single user setup, with docker virtualization The Arkitekt service was started, and Konstruktor instructed to advertise the Arkitekt installation on the network.

4.1.1.2 Workflow Specific Installation

After installation, both the OMERO conversion plugin (https://github.com/jhnnsrs/omero), the Standard Processing plugin (https://github.com/jhnnsrs/std), as well as the CARE algorithm plugin (https://github.com/jhnnsrs/kare) and the remote workflow scheduling plugin *Reaktor* (https://github.com/jhnnsrs/reaktor) were installed via the "*Plugin Pane*" from their respective GitHub repositories and started as virtualized internal containers managed by the platform. The *MikroJ* app was downloaded from its repository (https://github.com/jhnnsrs/mikroj) and installed and configured to point to a previously installed instance of Fiji/ImageJ (Schindelin et al. 2012). Napari (Sofroniew et al. 2022) 4.17 was installed on the system and the *Mikro-Napari* plugin installed through the interactive code terminal inside the napari environment, via a subprocess call (necessary step as the plugin was not yet available on the napari-plugin hub page). All apps were connected, and the user authenticated with the platform.

4.1.1.3 Data Preparation

The original CARE dataset (Weigert et al. 2018) *Tribolium*, was downloaded from its data repository (https://publications.mpi-cbg.de/publications-sites/7207/) and unzipped. A new *Dataset* was created through the *Orkestrator* UI and the images contained in the dataset were uploaded through drag-and-drop. For the batch conversion of the raw data to the Mikro internal *Image* format, *the Convert File Node* was searched and *reserved* on the Web UI, directly

linking it to the *OMERO Conversion App*. All image files in the dataset were selected using multiselect and converted through drop-drown and *Convert File (Batch)* assignment.

4.1.1.4 Model Preparation

The converted images of the original dataset were inspected through the Arkitekt web-interface, and three corresponding images of low and high signal-to-noise ratio were associated by dragging one over the other, labeling them as “ground-truth” in same *Context* (“CARE training set”) inside the popover *Associate Dialog*. The CARE *App* provided *Node Train CARE model* was reserved and run directly from the drop-down menu of the newly created *Context*, specifying the “ground-truth” relation as the training relation. The progress of the training was inspected on the WebUI.

4.1.1.5 Workflow Design

The workflow as shown below was designed on the Arkitekt design pane, exported, and deployed through the web interface on the *Reaktor* scheduling app, specifying *Segment denoised ROIS* as the new *Node* title. The exported workflow is given in its JSON representation in the annex.

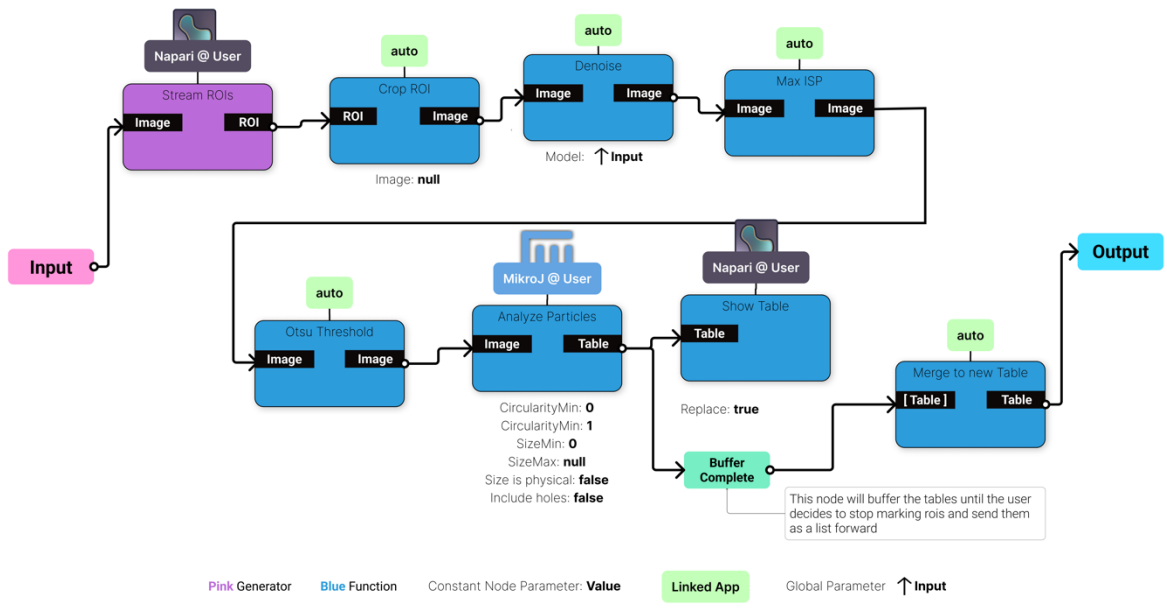

Figure 59 *The constructed workflow*

4.1.1.6 Workflow Run

The deployed workflow *Segment denoised ROIS Node* was reserved through the web interface. *MikroJ* and Napari were started and put into *provide* mode, as the state changes of the

workflow were observed on the web-interface. The workflow was then started on the web-interface, by right clicking on a low signal-to-noise ratio image in the dataset and selecting the *Segment denoised ROIS* assignment. During execution, ROIs were subsequentially marked on the opened ROI layer in Napari, selecting perceived areas of varying cell density. The loop backed results table in Napari was inspected.

4.1.1.7 GraphQL query generation:

The GraphQL Interface Page on the local Mikro instance was visited and the query below was designed in the interactive environment.

```
query {
  table(id: 1) {
    tableOrigins {
      name
      repOrigins(recursive: true, isRoiDerived: true ) {
        name
        roiOrigins(type: RECTANGLE) {
          dimensions {
            height
            width
          }
        }
      }
    }
  }
}
```

Figure 60 *The GraphQL Query that was designed to retrieve all tables their corresponding images and rectangular Rois.*

4.1.1.8 Workflow Portability Preparation

The designed workflow was exported on the web interface and the saved JSON file containing the serialized workflow was transferred to another computer (Ubuntu 22.04, Intel i7, 512 GB SSD). The installation procedure was performed as described the Konstruktor installer build for the Ubuntu environment. Arkitekt, its *plugins*, as well as the data and CARE model were prepared according to their respective sections. The transferred JSON file was imported on the web-interface and saved as a new local workflow. The workflow was then deployed and run in the same way, as described in the respective sections.

4.1.2 Results

4.1.2.1 Arkitekt makes bioimage analysis ecosystem **interoperable.**

This modern offline analysis workflow, demonstrates how Arkitekt allows to combine different bioimage processing, analysis, and visualization functionalities, including a deep-learning restoration algorithm (CARE), all on one single computer, without any expert setup, advanced configuration, or programming scripts. It ensured the full interoperability and transfer of *Images*, *Tables* and *ROIs* between two major bioimage platforms: ImageJ and Napari. No workflow specific interoperability scripts needed to be programmed and the user could use their existing Napari and ImageJ installations.

4.1.2.2 Arkitekt enables **interactive** uninterrupted workflows.

This workflow shows how the asynchronous workflow scheduling employed in Arkitekt, enables highly reactive workflows for interactive *Inspect and Adjust Circles*. Here it illustrated this on the example of a common in-workflow task of marking ROIs in a graphical user interface. Due to the real-time nature of the analysis workflows, users can inspect the analytical result directly in the original software, without having to manually launch and inspect the analytical outcome on the participating apps. Once started, this whole workflow was completely handled within Napari, without any interrupts, such as the necessity to interact with outside apps or even Arkitekt itself. Arkitekt here *immersed* the user in an interactive experience and enabled them to focus on the analytical insight rather than the orchestration.

4.1.2.3 Arkitekt enables **simple** analysis **data** and **metadata management.**

This workflow illustrates how easily Arkitekt passes a variety of different bioimage data types and their associated metadata from one platform to the other. Here Arkitekt kept track of the relationship between original images, marked *ROI*s, cropped images, filtered analysis of the cropped images, as well as all the quantifications associated with these images. This relationship graph could be easily explored on the Arkitekt frontend or exported for further statistical analysis.

Analytical workflows often require revisiting the data or the analysis strategy to uncover potential sources of biases or to check for correlations that weren't thought of in the original workflow. To illustrate this a scenario in the results, a common analytical scenario was devised:

Checking that the region of interests that were marked by the user share similar dimensions and therefore metrics as cell density are interpretable.

As Arkitekt built a linked data graph, respecting the transformations during the workflow execution, it can be revisited to gain new insights on the data. This exploration can happen both on the graphical interface as well as with help of a statistical *GraphQL* query (Figure Panel 2), that describes the traversal through the relationship tree and retrieves the wanted data.

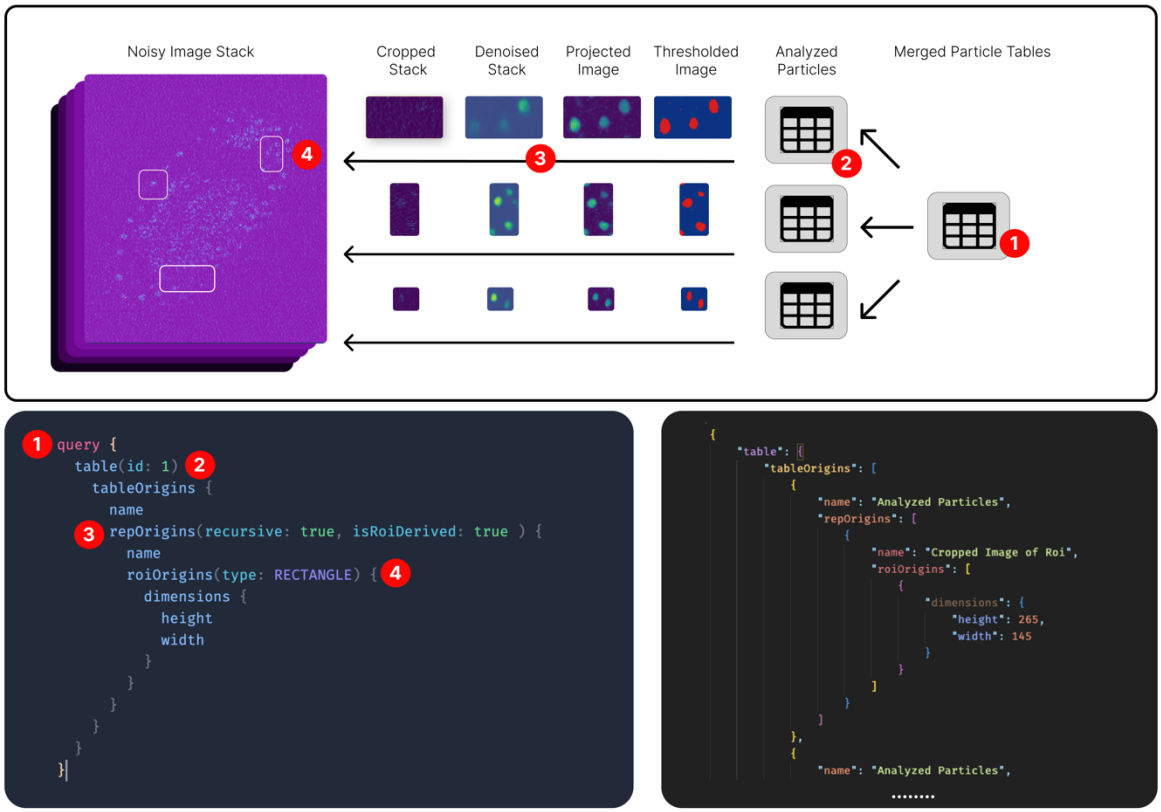

Figure 61 *The programmatic retrieval of the ROI dimension information based on the re-exploration of the data-graph. The GraphQL query as illustrated in the lower left panel was constructed to retrieve the information, in short starting from the original table (queried by its unique ID in the database (1)), the origins of the merged table are explored (2), the linked image origins are then recursively explored and filtered for ROI derived images (3), then for this Image its ROI origins are queried for its dimensions in height and width. Running this query against the mikro service results in the structured response on the lower right.*

4.1.2.4 Arkitekt enables easy **sharing** of **workflows.**

In this example, Arkitekt illustrated its ability to create sharable and universal workflows that can run on varying hardware. Here a workflow was transferred from its original computer environment to a new hardware and software environment, through the export of a universal interpretable workflow description file.

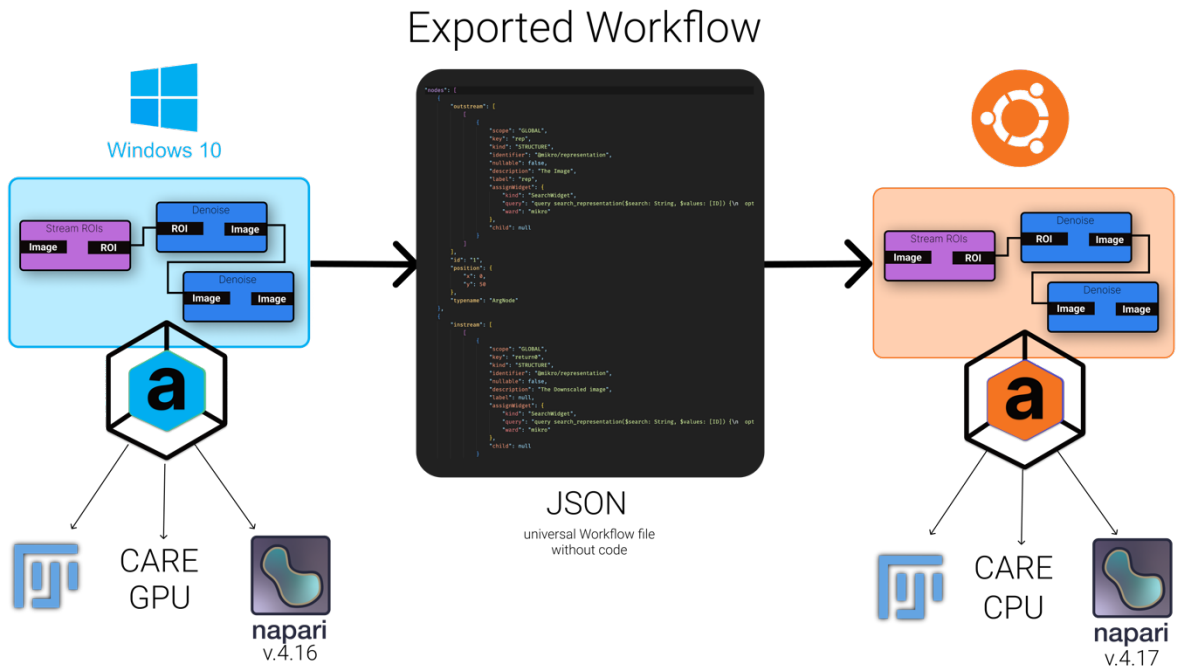

Figure 62 *The Arkitekt Workflow transmission between to different Arkitekt instances. The same workflow runs with GPU support on a Windows Machine, and for the lack of a CUDA enabled GPU, runs without GPU support on the Ubuntu machine.*

4.1.2.5 Arkitekt enables **simplified** deep learning **training** and **inference**

This example illustrates Arkitekt capabilities as a hub for deep learning. Here, beyond providing integrated *inference* on a trained model within an Arkitekt workflow, a neural net was *installed* and *trained* without any advanced configuration or folder-based data management, all within the Arkitekt interface. The installation of the GPU enabled deep-learning app CARE was easily handled through a guided installation process from the plugin interface, by pointing it to a GitHub repository and granting the plugins permissions. Then a training dataset was generated through Arkitekt's interface for selection and interlinking data via drag-n-drop. This interlinking paradigm yields a graph-like training dataset (a *Context*) that can contain *arbitrary* directed relations between any data on the platform such as *Image is ground-truth for Image* or *Roi is ground-truth for Image* .

This *Context* was then assigned in another simple step to the *CARE Training Node*, and GPU accelerated deep learning training commenced, with progress of the training being inspectable live on the Arkitekt interface. The resulting *Model* was then easily added in workflows for inference. Other deep learning models and app are easily supported in a similar fashion to the here described and only need to provide an adapted *Train Node* (selecting their desired relations) as well as a *Predict Node* (relevant Code parts in the annex).

4.2 Workflow II: Real-time Distributed Analysis

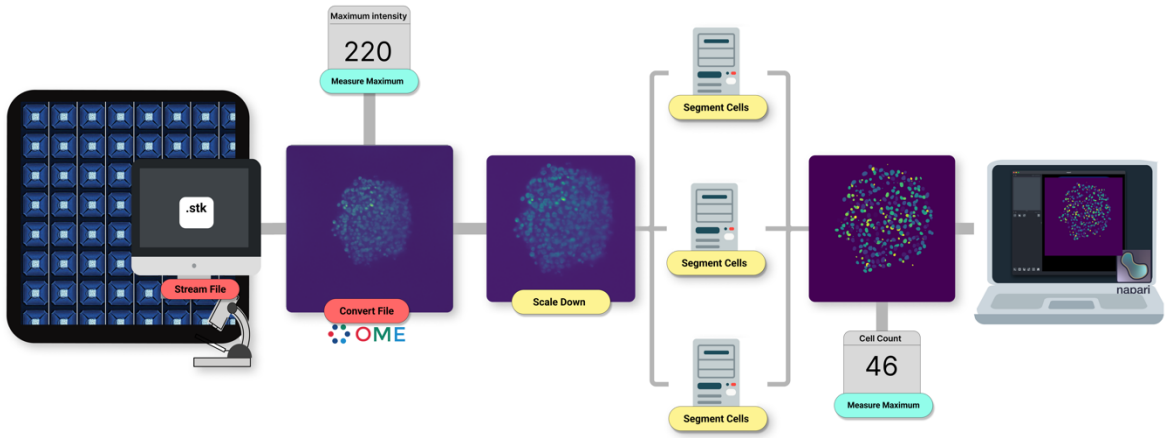

Figure 63 *The streaming analysis workflow, with the employed parallelization amongst 5 Computers (1 acquisition computer, 3 analysis computers and 1 visualization laptop)*

In this second example, Arkitekt's was adapted to orchestrate a fully automated acquisition and analysis workflow, illustrating how its streaming analysis paradigm can ensure reliable real-time analysis. It shows Arkitekt's capability to perform live (i.e. during the acquisition) quantitative monitoring of liver spheroids using soSPIM microscopy controlled by commercial acquisition software ("MetaMorph", Molecular Devices") and a simple analysis pipelines including the popular deep-learning-based Stardist 3D segmentation algorithm (Weigert et al. 2020). This workflow was based on a constant remote monitoring of a user-defined directory, checking for new image files. New images were then sent through an analytical pipeline, spanning file-conversion and metadata retrieval, deep learning-based nuclei segmentation, 3D visualization, and the display of the results (here the number of nuclei and their average volume) on a web dashboard. Powered by Arkitekt's inherent parallelization and multi hardware support, the analysis workload was parallelized amongst 3 GPU workstations to keep up with the acquisition.

4.2.1 Material and Methods:

4.2.1.1 Installation

Docker Desktop V20.04 was installed on a desktop computer (Ubuntu 22.04, Nvidia GPU 2080Ti, Intel i9, 1TB SSD). The Arkitekt installer (*Konstruktor*) was download from its GitHub Repository (https://github.com/jhnnsrs/konstruktor) and started. The platform was installed following its guided installer, choosing a single user setup, with docker virtualization The Arkitekt service was started, and *Konstruktor* instructed to advertise the Arkitekt installation

on the network. The Tailscale virtual private network client was installed on the same machine, added to a shared private virtual network according to their documentation (Tailscale 2023)

4.2.1.2 Sample Preparation:

Jewell Preparation:

JeWell were prepared as thoroughly described in Grenci et al. (Grenci et al. 2022) and Beghin et al (Beghin et al. 2022).

Spheroid Culture

HEP-G2 cells stably expressing H2B-eGFP fusion protein were maintained in DMEM media (11965092, Invitrogen) supplemented with 10% FBS (10082147, Invitrogen), 1% GlutaMAX (35050061, Invitrogen), 1% penicillin-streptomycin (15070063, Invitrogen), and 1% Sodium Pyruvate (11360070, Invitrogen) at 37 °C and 5% CO2. After being trypsinized, the cells were suspended in complete DMEM media supplemented with 20% FBS, 1% GlutaMAX, 1% penicillin-streptomycin, and 1% Sodium Pyruvate. The cell suspension was adjusted to 0.5 × 10^6 cells per ml. Next, 1 ml of cell suspension was placed into a 120 µm opening JeWell plate for 10 min to allow for cell adhesion and to get approximately 80 cells per JeWell. The excess cell suspension was removed, and the plate was gently washed once with DPBS (14190250, Invitrogen). After the wash, 2 ml of complete DMEM media was added to the plate. The cells were then cultured for 3 days at 37°C and 5% CO2.

Fixation

On the third day, the cells were fixed for 20 min in 4% paraformaldehyde (28906, ThermoFisher Scientific) at room temperature.

4.2.1.3 Acquisition preparation

The fixed samples were mounted under a Nikon Ti2 inverted microscope (“ECLIPSE Ti2 Series", Nikon), equipped with a 60X objective (WI 1.27NA, Nikon). The microscope, the motorized stages, and the Multi-Dimensional Acquisition (MDA) process were controlled by MetaMorph software (Molecular Devices). A dedicated home-made plugin was integrated into MetaMorph to allow for the control of the soSPIM beam steering unit and therefore the sample illumination as previously described (Galland et al. 2015) (Beghin et al. 2022). MetaMorph was configured to run a multi-timepoint (30 timepoints every 20min), multi-position (20 selected positions) acquisition of the samples (60 z-steps, 1µm step-size), utilizing a set of *Journals* for automated position correction as previously described (Beghin et al. 2022). The multi-timepoint acquisition was chosen to simulate a timelapse experiment on live samples but

allowing for the assessment of the reproducibility of the analytical pipeline. MetaMorph was instructed to put the resulting TIFF image stacks into a specific folder on the acquisition machine.

4.2.1.4 Workflow specific installation

After installation, the *OMERO* conversion plugin (https://github.com/jhnnsrs/omero), the *Standard Processing* plugin (https://github.com/jhnnsrs/std) the *Stardist* segmentation plugin (https://github.com/jhnnsrs/segmentor) and the remote workflow scheduling plugin *Reaktor* (https://github.com/jhnnsrs/reaktor) were installed via the "*Plugin Pane*" from their respective GitHub repositories and started as virtualized internal containers managed by the platform. *Gucker*, the Arkitekt enabled file watcher app, was installed through its installer (https://github.com/jhnnsrs/gucker) on the microscope computer (Windows 10, Intel-i5, specs) and pointed to the output directory of MetaMorph. *Gucker* was connect to Arkitekt through its GUI and authenticated with the platform.

Napari was installed on a portable desktop computer (MacBook Air M2, 2023) and the *Mikro-Napari* plugin installed as described in the previous workflow. The Notebook was configured to share the same virtual private network as the desktop and microscope computer utilizing Tailscale.

Docker Desktop was installed on two additional GPU-powered computers (both Windows 10, i7 Nvidia GPU 2080TI, 500GB SSD) and two instances of the CARE app were run through the terminal call of *docker run –t jhnnsrs/segmentor arkitekt run easy*, which downloaded and started the plugin, the Fakts connection link in the terminal was followed, and the app authorized on the platform.

4.2.1.5 Data Preparation

A previously trained Stardist3D model (Beghin et al. 2022) was zipped and uploaded through drag and drop on Arkitekt's web interface. The *Node Package Mode",* was reserved and the uploaded zip archive, was converted into an *Model*, *specify*ing the model type as *TensorFlow* and its name as "Stardist soSPIM*"* on assignation.

4.2.1.6 Workflow Preparation

The below workflow was created through the *Orkestrator* workflow interface, setting the uploaded "Stardist soSPIM" model as default for the Stardist prediction node. The *Convert to*

Images Node was given a position tolerance of40µm to merge motion corrected positions together. The workflow was deployed and reserved.

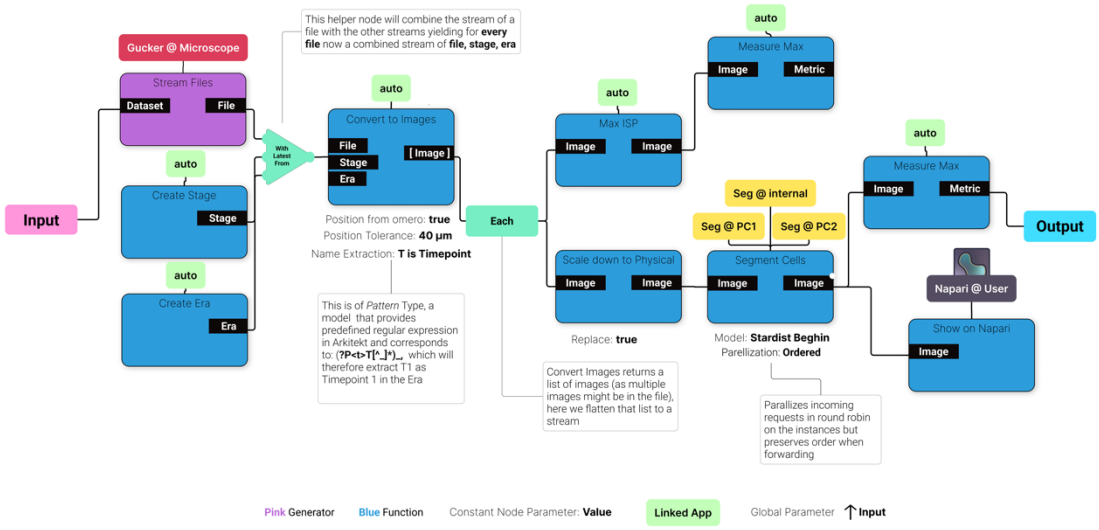

Figure 64 *The designed workflow for real time monitoring.*

4.2.1.7 Workflow Run

The workflow was run, specifying a newly created dataset as the input. The MetaMorph acquisition was started on the microscope computer. During the acquisition, the workflow was monitored both on the Arkitekt web dashboard, back on the acquisition computer, as well as through inspecting the streamed segmentations on Napari, from the remote laptop. Napari was occasionally switched on and off, and moved to different networks, under assessment of the workflow still running.

4.2.1.8 Performance Measurement

For input/output performance measurements in a separate workflow run, the *Maximum Intensity Projection* Node was pointed to a reimplementation of the original functionality running in a standalone app on the Arkitekt platform computer. This implementation utilized the same API calls, but console logged the elapsed time for the IO heavy operation of downloading the image stack to memory, performing the intensity projection, and uploading the image stack again to the platform. All 120 (20 positions, 6 timepoints) measurements were analyzed in a separate Python script, performing normality checks (Shapiro-Wilk test) with SciPy (Virtanen et al. 2020) and subsequently analyzed for mean and standard deviation via NumPy (Harris et al. 2020)

4.2.2 Results

4.2.2.1 Arkitekt enables analytical workflow spanning **multiple** hardware.

Analytical tasks such as acquisition and deep learning have vastly different hardware requirements, that are often not achievable on a single computer. Moreover, as acquisitions are very sensitive when dealing with long-term fast multidimensional acquisitions, it is preferable to avoid doing any compute intensive operations on the acquisition computer, to avoid any risk of crash. Arkitekt here circumvents these problems by offloading specific tasks to the hardware that they were designed for: The acquisition was performed on a standard desktop Windows machine with specific hardware drivers tailored to the microscopy, while the deep learning task could benefit from GPU-acceleration on dedicated hardware, which was readily available in the lab. The task of monitoring was handled by a third remote laptop, which could change location with the experimenter.

4.2.2.2 Arkitekt brings analytical live insights to **commercial** microscopy.

Modern bioimage workflow rely heavily on commercial microscope software to manage the acquisition. These software packages seldom provide a level of feedback or visualization for higher order analytical insights. In this example, Arkitekt while being completely agnostic of the control software (here MetaMorph), circumvents this limitation by streaming files from a shared folder to the platform, offloading the computation to other apps and wiring the results directly back to the acquisition computer or any other remote computer, enabling immediate live analysis and inspection of the results.

4.2.2.3 Arkitekt supports **integration** of variably organized **metadata.**

In this workflow, Arkitekt facilitates the dynamical ingestion and combination two popular formats of metadata into a unified schema in its database:

1. Through the included OMERO conversion, it extracts the *contained* OME XML metadata *which* contains the position of the acquired image on the stage (as X, Y, Z coordinates).
2. *Non-contained* metadata about the specific relative timepoint (T1, T2, T3 etc.) in the experiment were ingested through information provided inside the filename through a regular expression.

Here, Arkitekt's non rigidity in ingesting and combining metadata allowed to capture and save a complete view on the acquired biological data in context.

4.2.2.4 Arkitekt can handle **modern data loads** of large scale bioimage experiments.

In this workflow Arkitekt handles the data throughput of large-scale image data (microscope producing 260MB of image data every 20seconds), with minimal latency overhead. The Stardist segmentation algorithm is *compute* bound and cannot be used as a reliable indicator for the input-output (IO) performance and the platform overhead. The IO heavy Maximum Intensity Projection operation consistently performed a stack projection in around 573ms (+-0.32ms). This time includes loading the 260MB image stack into RAM, performing the operation and reuploading the new image stack to the platform.

4.2.2.5 Arkitekt ensures **reliable and observable** execution of workflows.

This multi-hour, multi-position workflow, demonstrates the general reliability of the platform. During its execution, the workflow was easily monitored on the web interface in real-time, inspecting the transition of datapoints through the analytical pipeline. As the Napari laptop shared the same network than the analysis computer, this monitoring could also happen remotely. As the *Napari Visualization Node* was marked as *non-essential* on workflow creation, its connection errors (forced disconnects), did not affect the overall acquisition.

4.2.2.6 Arkitekt **uncovers** and **eliminates** analytical bottlenecks through parallelization.

This workflow illustrates Arkitekt ability to utilize its built-in multi-worker paradigm to parallelize analytical workloads on multiple machines. In a previous non-parallelized run, the 3D Stardist segmentation task was uncovered as the core bottleneck and limiting factor for the real-time analysis to keep up with the acquisition interval: Segmentation took around 1.5 minutes, while the microscope produced images every 20 seconds. This insight was facilitated entirely through the inspection of autogenerated waterfall diagrams in the Arkitekt run interface.

To eliminate this bottleneck in the illustrated run of the workflow, the node was instructed to parallelize on two additional computers inside the lab, linking up two more connected *templates* representing apps on the additional computers, to the segmentation node. The illustrated waterfall diagram then indicated the compensation of the bottleneck. While in this run this linking up was done prior to the workflow run, new templates/workers can also be linked up *during* the execution, when bottlenecks become evident.

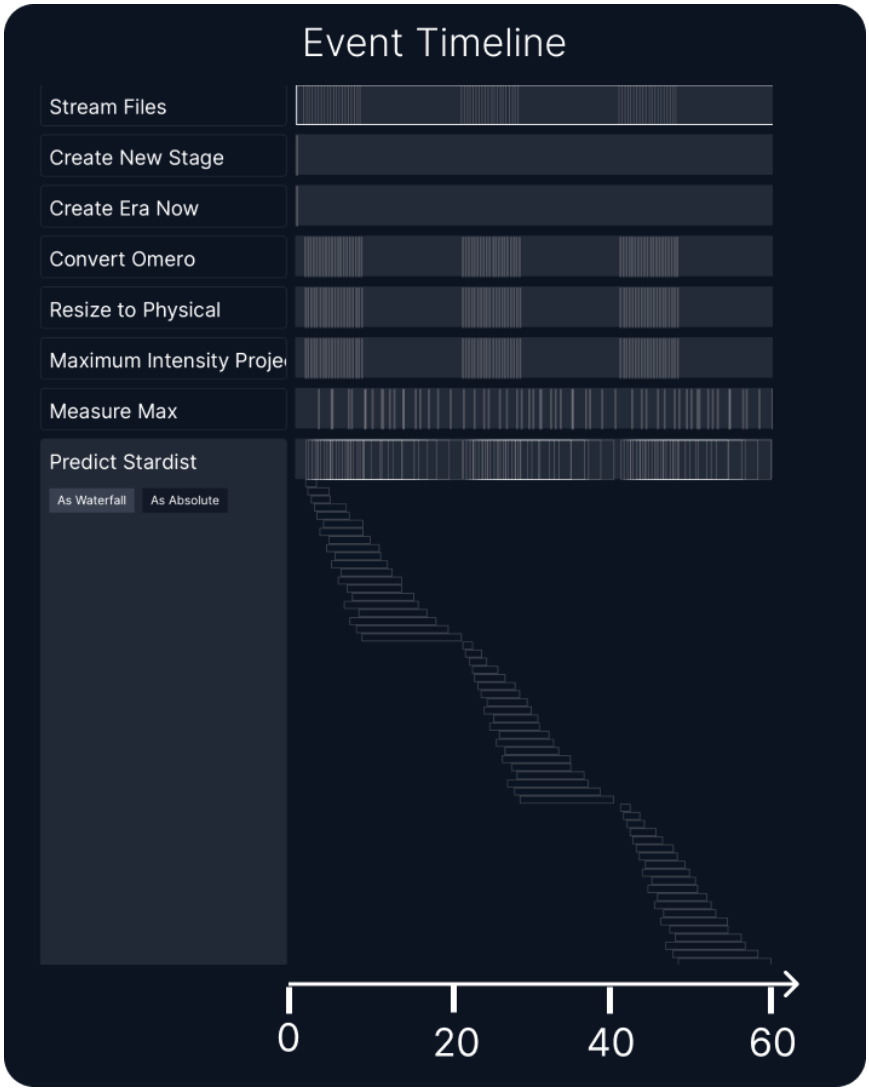

Figure 65 *The Arkitekt Waterfall diagram, as inspectable on the WebUI inside Arkitekt, but trimmed for the first hour for clarity. The Predict Stardist node is unfolded showing all tasks, illustrating the processing time for each assigned task. The processing time corresponds from the time interval of assignation till the task is finished. As the node was parallelized 3fold, bars overlap, however as task length is still increasing, this node needs to be considered as bottleneck for the analysis, which is only compensated by the acquisition pause (processing could continue and catch up).*

4.3 Workflow III: Smart Microscopy

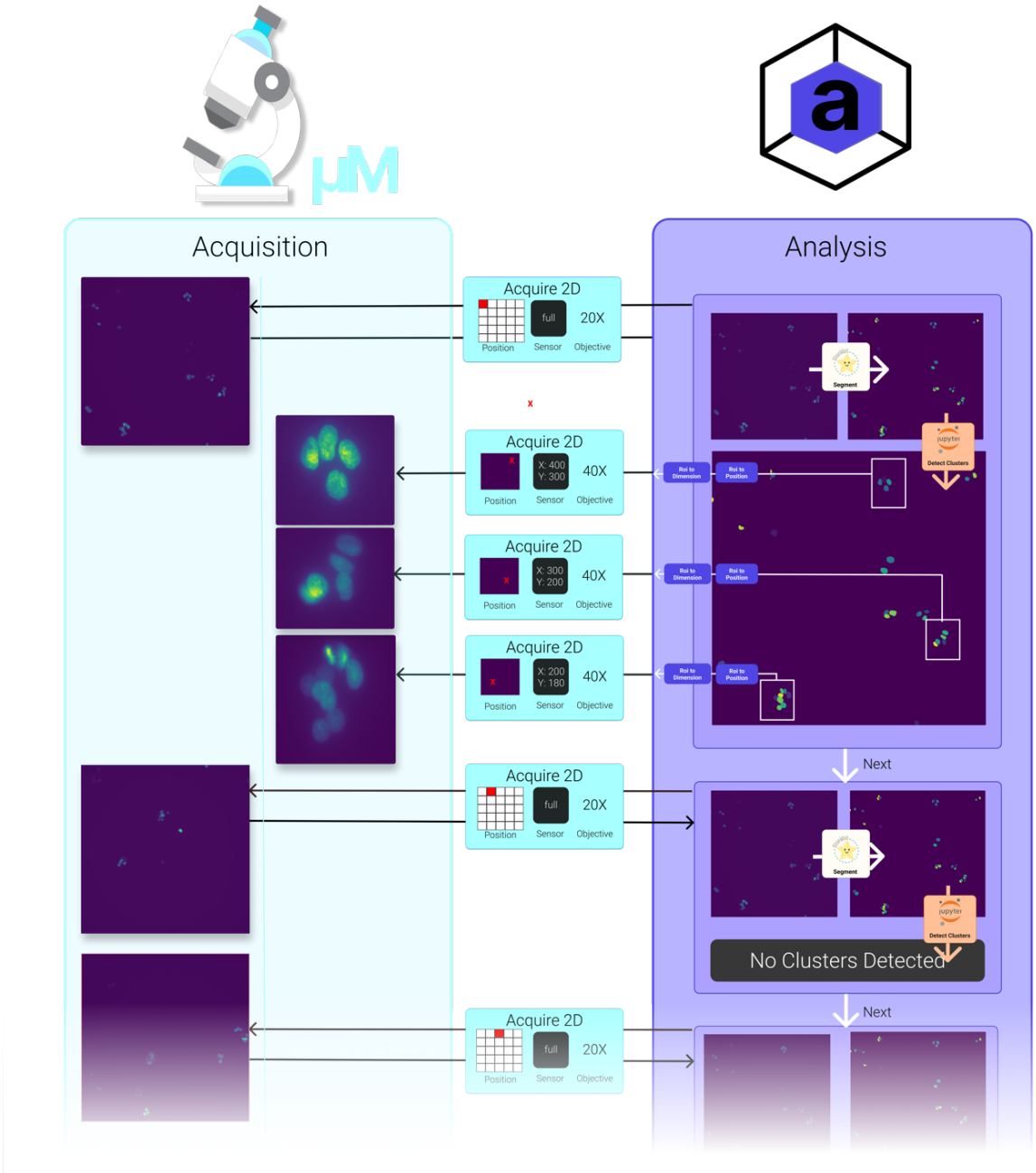

Figure 66 *A conceptual overview of the Smart Microscopy workflow*

To illustrate Arkitekt readiness for Smart Microscopy, a simple but powerful no-code example of a Smart Microscopy workflow was designed to perform the 3D live monitoring of cell clusters with a 40X objective, while scanning a large field of view with a 20X objective. In this workflow, the Micro-Manager open-source software (Edelstein et al. 2014) was used for the multidimensional acquisition. The user interactively set up a grid of positions on large sample area (2.61mm x 2.61mm, corresponding to 4*4 stage positions) containing living fluorescent

cells placed under an inverted microscope, with a 20X magnification objective. All the defined positions were then acquired at 20X magnification every 30 minutes for 24 hours. For every acquired 20X magnification image, nuclei were automatically segmented on a remote computer using Stardist (Weigert et al. 2020) algorithm, and cell clusters computed with DBSCAN algorithms (Ester et al. 1996) with certain properties were identified and marked. When one or more of such identified clusters were detected, the ROIs central coordinates and dimensions were translated to stage coordinates and sent back to the microscope. 3D stacks of these positions (25 planes, 0.5µm step size) were then collected at higher magnification (40X) on a well-adjusted ROI fitting the cell clusters size. After all the positive events were detected, acquired in 3D high-resolution and displayed on the web-interface, the next image was acquired at 20X magnification.

4.3.1 Material Methods

4.3.1.1 Installation

Docker Desktop V20.04 was installed on a desktop computer (Ubuntu 22.04, Nvidia GPU 2080Ti, Intel i9, 1TB SSD). The Arkitekt installer *Konstruktor* was download from its GitHub Repository (https://github.com/jhnnsrs/konstruktor) and started. The platform was installed following its guided installer, choosing a single user setup, with docker virtualization The Arkitekt service was started, and Konstruktor instructed to advertise the Arkitekt installation on the network. The Tailscale virtual private network client was installed on the same machine, and the machine added to the shared private virtual network according to their documentation (Tailscale 2023)

4.3.1.2 Workflow Specific Installation

After installation, the *Standard Processing* plugin (https://github.com/jhnnsrs/std), as well as the *Stardist* segmentation plugin (https://github.com/jhnnsrs/segmentor) and the remote workflow scheduling plugin *Reaktor* (https://github.com/jhnnsrs/reaktor) were installed via the "*Plugin Pane*" respective GitHub repositories and started as virtualized internal containers managed by the platform. *Mikro-Manager* was installed on the microscope computer through the installer hosted on (https://github.com/jhnnsrs/mikro-manager) and pointed to a previously installed instance of Micro-Manager (Edelstein et al. 2014) (nightly version of 30.06.2023)(https://download.micro-manager.org/nightly/2.0/Windows/MMSetup_64bit_2.0.1_20230630 .exe). All apps were connected, and authenticated with the platform.

4.3.1.3 Sample Preparation

Cell Culture

HEP-G2 cells stably expressing H2B-eGFP fusion protein were maintained in DMEM media (11965092, Invitrogen) supplemented with 10% FBS (10082147, Invitrogen), 1% GlutaMAX (35050061, Invitrogen), 1% penicillin-streptomycin (15070063, Invitrogen), and 1% Sodium Pyruvate (11360070, Invitrogen) at 37 °C and 5% CO2.

Preparation

Cells were seeded at 30.000 cell/mL concentration on 18 mm round coverslips the day before the experiment and maintained under culture medium. On the experimental day, the existing media on the 18 mm round coverslips with the HEP-G2 cells was replaced with a fresh batch of Flourobrite DMEM (A1896701, Gibco) media, fortified with 10% FBS and 1% GlutaMAX. During imaging cells were maintained at a constant temperature of 37 degrees Celsius and CO2 levels were maintained at 5%.

4.3.1.4 Microscope Preparation

A Nikon Ti2 inverted microscope, was equipped with a thermal chamber and CO2 airflow. The Micro-Manager installation was instructed to load the corresponding device drivers and configuration groups for the SP2. The separately installed *Mikro-Manager* was configured to point to the correct configuration groups, mapping objectives (20X and 40X) as well as the main stages (X, Y, Z) and auto-focus stage.

On the day of the experiment both 40x (Plan APO 40x/0.95, Nikon) and 20x (Plan API 20x/0.5, Nikon) objectives were installed. Automated pixel size calibrations were performed utilizing sparse cells on the mounted coverslips as guiding stars, through the integrated Micro-manager plugin. In total, 16 positions on the coverslip were selected with the help of the Micro-Manager Grid-Position manager. When the sample was mounted, the offset of the objective with the Perfect Focus System offset was noted. (“T-PFS Perfect Focus Unit", Nikon).

4.3.1.5 Workflow Preparation

The below workflow was created on the Arkitekt web-interface. It represents the logic of the acquisition of a single position and consequent cluster monitoring on this position. The workflow was deployed on the “*Reaktor*” scheduling app, naming the new *Node* “Image Clusters on Position”.

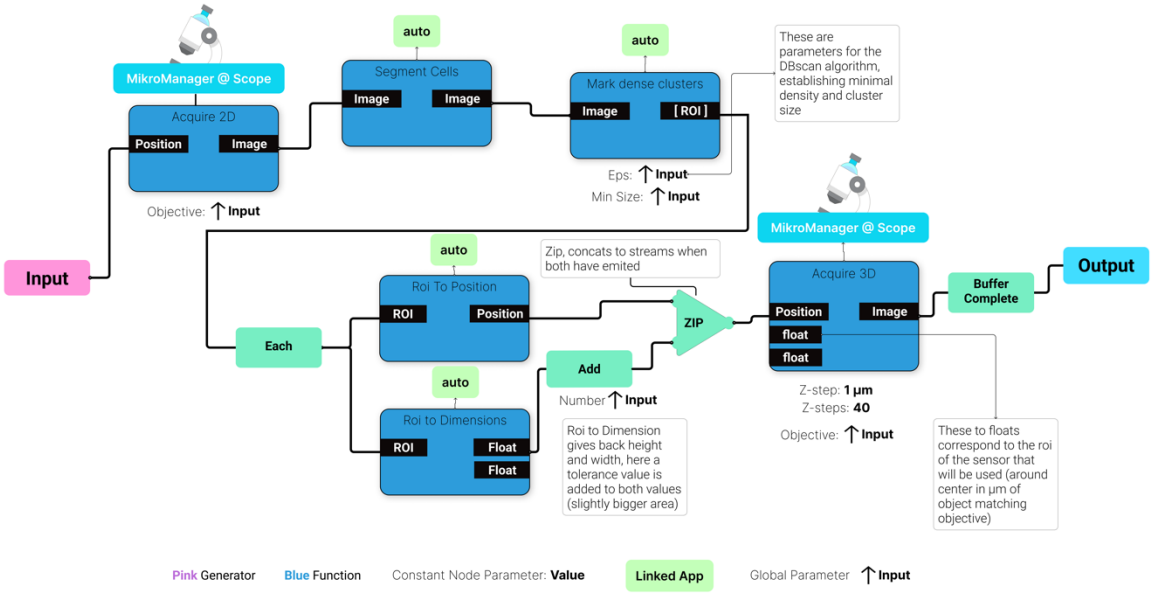

Figure 67 *The single position workflow as designed on the Arkitekt platform. Most properties here are set global to allow for maximum flexibility when using the workflow for testing out potential parameters for DBSCAN density-based cell clustering.*

A second workflow, now representing the wider multi-position acquisition was created, utilizing the just created primary workflow as a *Node*. The workflow was deployed, setting "Stream Multi Position Clusters" as *Node* name.

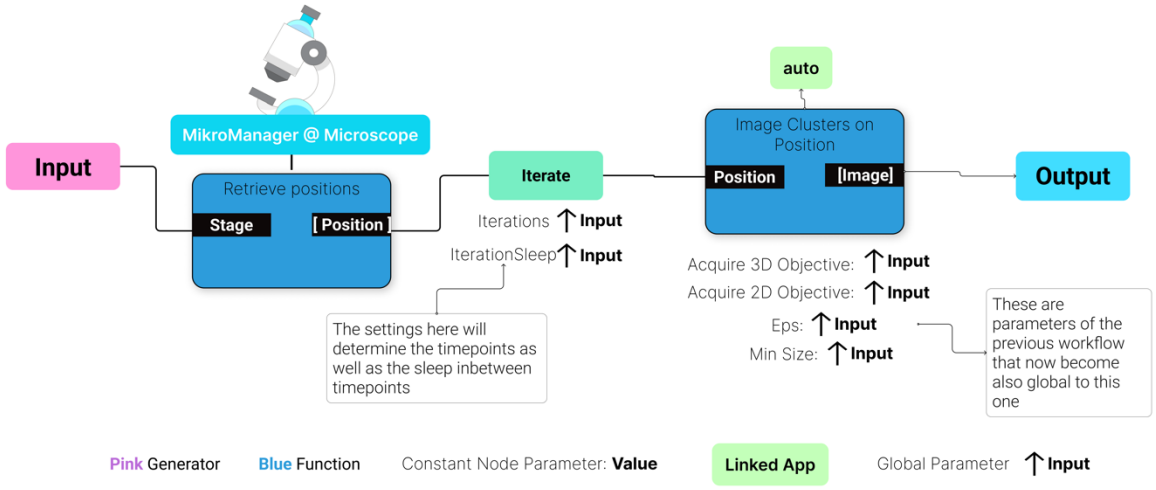

Figure 68 *The overarching workflow, which enables the multi-position, multi-timepoint acquisition of the workflow*

4.3.1.6 Workflow Run

Mikro-Manager was started, connected, and authenticated with the Arkitekt platform. The Mikro-Manager *Set Objective PSF-Offset Node* was *reserved* and run two times, setting each objective's previously noted offset. The workflow *Node Stream Multi Position Clusters* was *reserved* and started on the Arkitekt web interface, providing the following parameters as input: *Stage*: Newly Created Stage, *Acquire 3D Objective*: 40X, *Acquired 2D Objective:* 20X, *Eps*:

100 pixel, *MinSize*: 3, *Iterations*: 24, *IterationSleep*: 20 minutes). The streamed output was monitored on the Arkitekt web interface and on the Stage Detail Pane, which automatically displayed new positions and images.

4.3.2 Results

4.3.2.1 Arkitekt enables **no-code** smart microscopy workflows.

In this exemplary workflows Arkitekt shows its capabilities as a *no-code* smart microscopy platform, providing modular building blocks spanning deep-learning segmentation and cluster algorithm-based event detection. None of the *Nodes* in this example, had to be specifically crafted for this workflow but utilized standard bioimage analysis methods. It is also worth noting that only one *Node* needs replacement to extend the workflow to account for other biological events (e.g., detecting large cells instead of clusters).

4.3.2.2 Arkitekt enables **exploration** of biology at different scales

In this workflow, Arkitekt demonstrates its fitness for the acquisition and exploration of biological samples at different scales. The integration of the microscope data with respective meta-data (e.g. *Position* on *Stage*, or *Image* pixel size) fully models the data graph of the *Mikro* service which can now be used to put all acquired data in full spatio-temporal context. Each 40X data item can be inspected for example for its position on the sample space and the ROI of the cluster that informed the position. Instead of all metadata being barely understandable key-value pairs, Arkitekt allows its exploration through relational linkage.

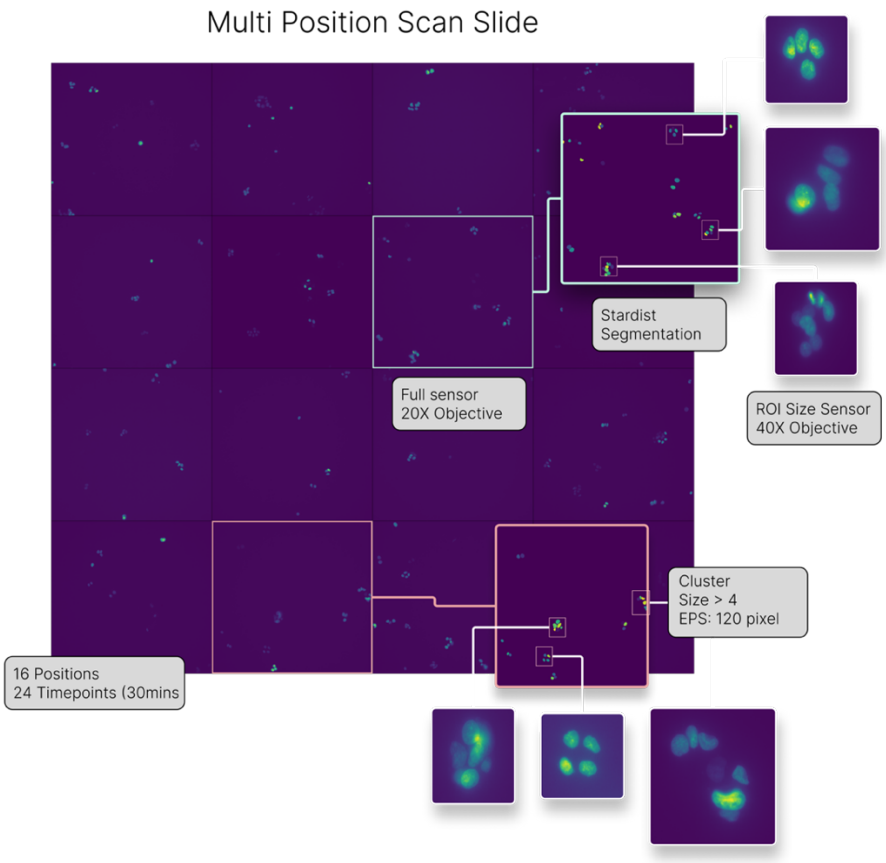

Figure 69 *Arkitekt's constructed data graph, as explorable through links and renders on the web interface.*

The clusters are not only explorable in their spatial context but as Arkitekt kept track of their positions and informed a new acquisition based on the current centroid of the cluster, clusters are easily monitored over time and in 3D.

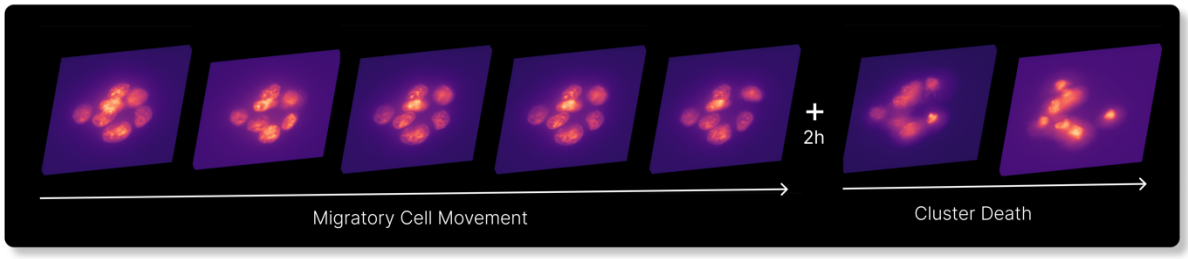

Figure 70 *A exemplary time selection of a tracked cluster visualized using Napari (Plasma colormap) in 3D. This rendering was performed without any postprocessing by using the "Recontextualize Positions" Node, that merges neighboring positions by a given threshold (40µm) together into a new position. That then can be visualized as a timeseries directly in Napari.*

4.3.2.3 Arkitekt interfaces with **open-source** Microscopy

This workflow shows Arkitekt ability to interface with open-source microscopy platforms, such as Micro-Manager. The here demonstrated Mikro-Manager app wraps Micro-manager through Pycro-Manager (Pinkard et al. 2021) on the microscope, necessitating the implementation of only a few convenience methods for the acquisition such as *Acquire 2D*

and *Acquire 3D* (complete code in annex). As Arkitekt takes control of the time orchestration of the workflow, no complex higher-level features of the Micro-Manager platform such as the multidimensional acquisition are needed. Users can however fully rely on interactive functionality inside Micro-Manager to setup positions, explore the coverslip, or set acquisition settings.

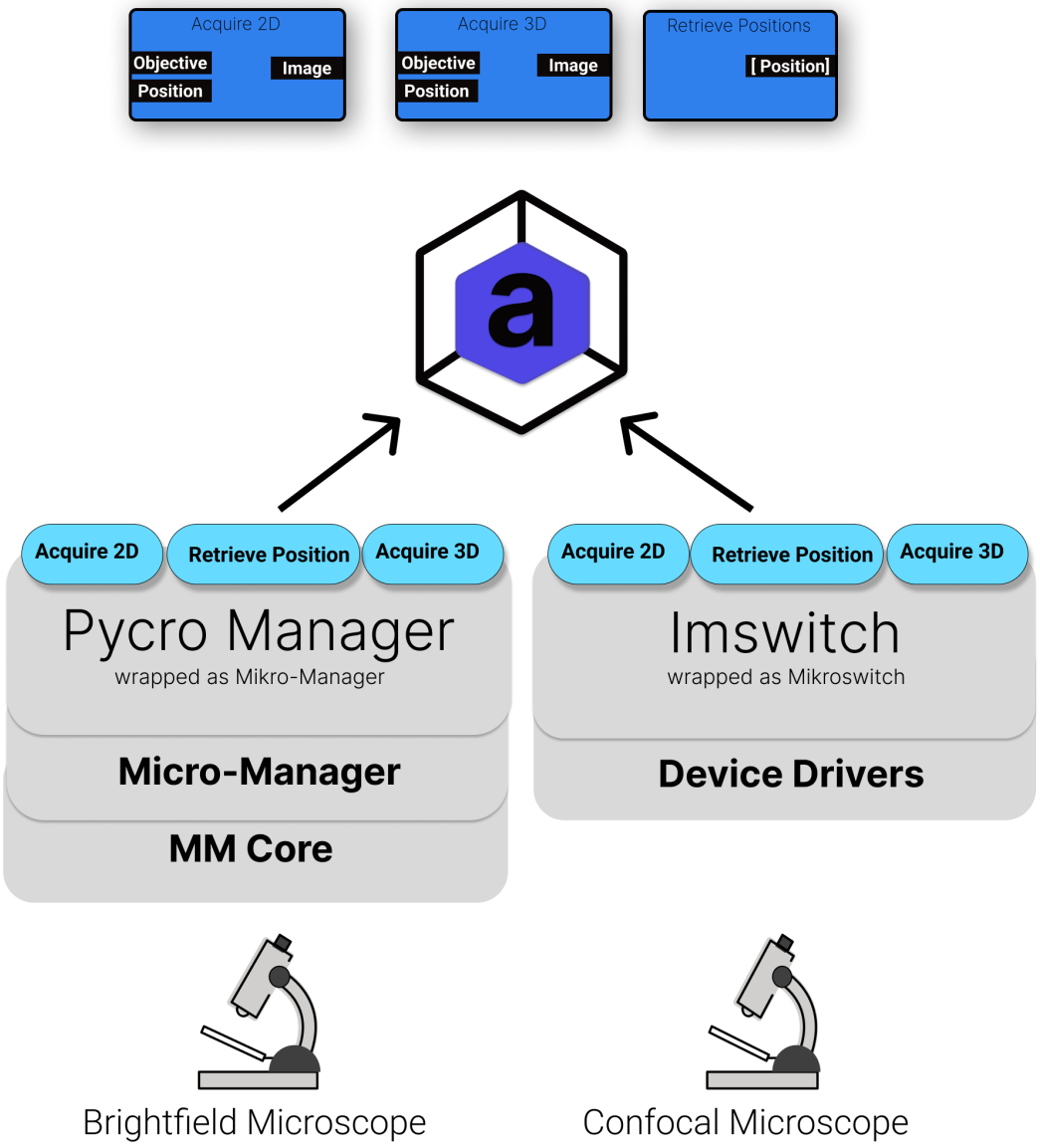

Figure 71 *The protocols used to integrate open-source microscopy software, in this workflow Pycro-manager was used, which talk to the Java interface that in turn tasks to the MMcore interface written in C++ that then controls the microscope hard. For direct python-based access to the microscope, another popular software tool "Imswitch" could have also been used. As either software chooses to expose their respective acquire 2D implementation, this workflow can be run on both software platforms interchangeably.*

4.3.2.4 Arkitekt **reduces** analytical burden.

A core challenge in microscopy workflows that monitor the dynamics in large cell populations is to find the right trade-off between the wanted spatial and temporal resolution of the acquisition. Higher spatial resolution (e.g. through a different objective) conflicts with high temporal resolution as the field of view is now smaller. Here we illustrate how Arkitekt can be used to alleviate this tradeoff. In a comparable scenario to this workflow, that does not use a guided acquisition technique, samples would have to be imaged at a much higher time interval, limiting for example the ability to track cell movement. In addition, such an experiment would occupy a significantly larger storage space, increasing the analytical burden.

Comparing the two approaches, the adaptive strategy excels in a few regards. It saves a tremendous amount of disk-space (2,546 GB vs 50 GB), it drastically saves time (23.8 min FOV vs 8.2 min per entire field of view) and reduces light induces toxicity on the surrounding cells.

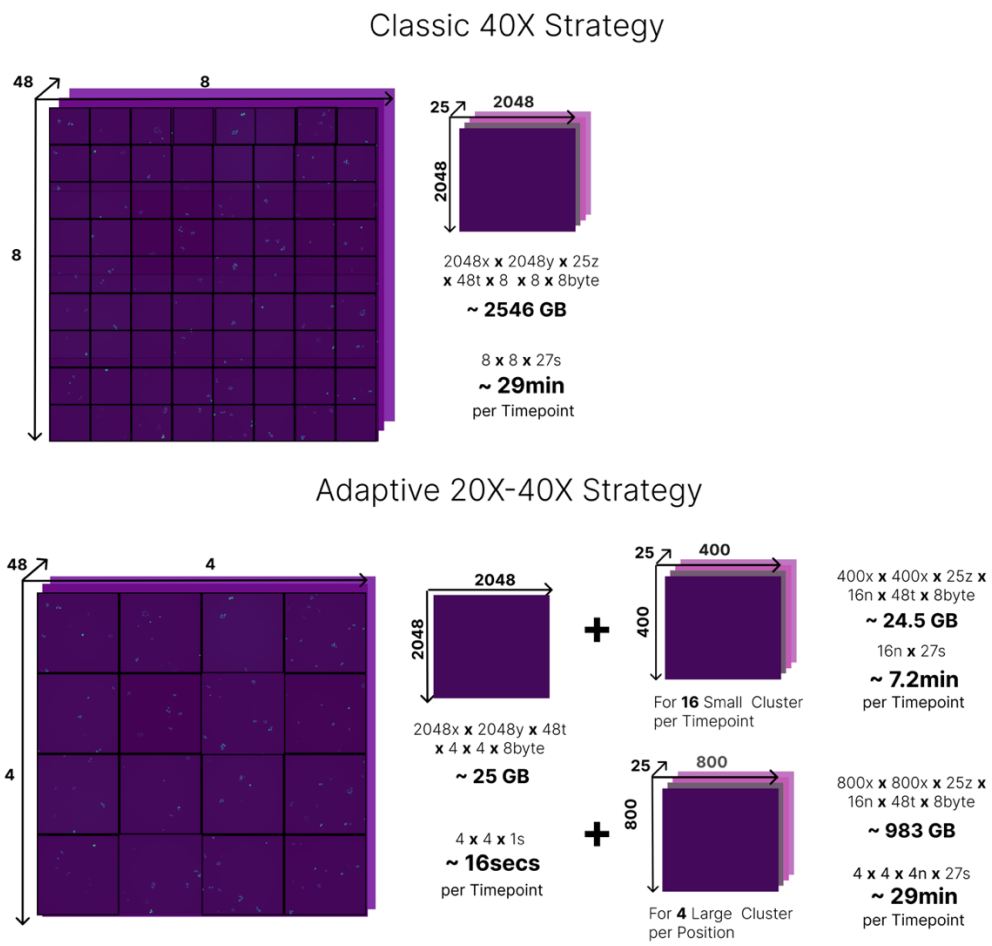

Figure 72 *Comparison of a classic 40X monitoring strategy, were the same field of FOV is being monitored, vs the smart workflow strategy. The smart-workflow strategy panel displays two potential scenarios with high and low cluster density, and their respective time and space burdens.*

5 Discussion

This work established Arkitekt software platform as a possible solution to the overarching challenges of modern bioimage analysis workflows. It showed its capabilities as a highly adaptive open-source framework for modern bioimage analysis in demonstrating its features on three exemplary workflows: combining multiple bioimage apps and scripts interactively, providing real-time distributed analysis in a multi-dimensional microscopy framework, and its applicability as a scheduler for Smart Microscopy.

Arkitekt addresses the concerns and challenges of modern bioimage workflows through a holistic open-source framework approach. As compared to the young body of literature, Arkitekt stands out due to its capability to establish *interoperability* via its computational and data backbone that seamlessly connects multiple bioimage apps and scripts, tailored to the scientist's specific need. It also provides a unique set of features to *orchestrate* these tools *interactively* and in *real-time*, allowing advanced *data flow* and facilitating *human in the loop* analysis. These features are enabled through a set of *user-friendly* abstractions that allows non-experts to easily design parallelizable and universal workflows visually, without having to worry about the underlying hardware.

In accordance with the principles of FAIR (Wilkinson et al. 2016) it employs strategies to address the needs of modern scientific data and metadata management, providing the user with the ability to explore their data-graph *programmatically* and in *usable* interfaces.

Accounting for the challenges of the developers, building our modern bioimage tools, Arkitekt provides easily integrated features to create responsive and *usable* interfaces for their tools. It allows developers of tools to fully rely on the libraries and hardware they deem necessary and gives them full flexibility to build them as *standalone* dedicated apps or integrated plugins a la ImageJ.

The Arkitekt platform is now fully operational and comes with a well packaged set of supporting libraries and an emerging interactive online documentation. Indeed, Arkitekt is already employed outside of its original lab environment, powering the analysis in an integrated solution for light-sheet organoid screening. A standalone microscope and analysis setup, *ORGANOSCREEN*, integrates analysis workflows akin to the *Workflow,* extending them to new biological questions such as cell division event detection. It additionally integrates

phenotypic classification of the acquired data, and feeds the now structured data to an external dedicated big-data analytics database.

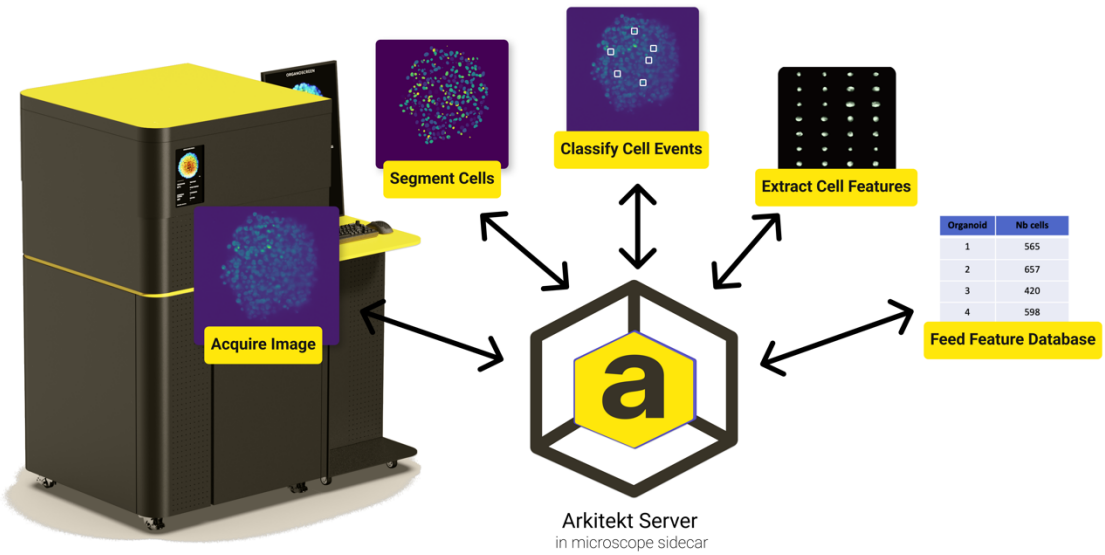

Figure 73 *The ORGANOSCREEN platform, with Arkitekt orchestrating the automated analysis pipeline.*

However, as an early-stage build, and given Arkitekt's development in the scope of a doctoral thesis, this last section will now address some core limitations in the design and implementation of the framework. It will then further shed light on potential future applications and extensions to the Arkitekt platform.

5.1 Limitations

5.1.1 Maintenance

A question that every new software solution especially in the sciences needs to face is that of maintenance. Arkitekt's maintenance is currently hinging on the support of the author of this thesis, which even though committed to providing support, will not be able to stem the maintenance alone. However, given Arkitekt's highly modular design, some of its supporting libraries have already seen community support and acceptance and are openly being developed as open-source projects with little maintenance effort.

5.1.2 Yet another framework

One of the core critique points of the Arkitekt platform is that, even though it tries to aim to provide a unified framework for modern analysis, it could just further the fragmentation of tools in this space by providing *yet* another protocol to make workflows interoperable. It only

serves its purpose, if it or a future implementation sees a wider adoption in the community. This adoption relies on the feasibility of Arkitekt for advanced workflows which in turn relies on the availability of tools that are interoperable with the platform. Even though Arkitekt's set design goal is to make the tool developer experience as enjoyable and flexible as possible, it so far has only seen improvement and iteration through feedback from a selective few inside the lab and our collaborators and could benefit tremendously from a wider integration of opinions that would shape the developer experience.

5.1.3 On programming support

Even though Arkitekt is written with an open API that is reliant on open web standards, which are supported by almost any programming language, client libraries that transfer the data and establish the scheduling in their respective programming paradigm need to be implemented by the respective programming language. This includes the transfer protocol powered by Zarr (Miles et al. 2020) and the dedicated *actor provision model* of Arkitekt. These protocols follow open, documented standards but lack widespread adoption in other programming languages. As such, dedicated client libraries for the Arkitekt platform are currently only developed for Python and JavaScript/Typescript, with plans for making a client in Rust (Matsakis and Klock II 2014). First class programming support for both C++ and Java islacking in Arkitekt's current iteration, limiting widespread deep support for other bioimage analysis software.

5.1.4 On latency

One of the core limitations of Arkitekt and the source of latency in *distributed* workflow execution is the performance of the broker, who is responsible for routing messages to the different apps and needs to handle the most requests per second. Arkitekt uses RabbitMQ for routing, which itself introduces only small latency (~10ms under normal load). However, this routing is protected by the WebSocket API, that in addition to authenticating the request also currently persist the message into the database directly. This increases the cumulative latency in task assignment to the range of 50ms per task, depending on the network connection. For some bioimage workflows, such as in calcium event detection in neuronal populations or molecule tracking in single-molecule localization microscopy, where events need to be detected and reacted upon with millisecond precision, this latency becomes a critical bottleneck and renders the *distributed workflow* scheduling unperforming. Even though *local* workflow execution can circumvent this latency problem, these workflows are then limited to the set of tasks that are implemented in the software. Other lower latency protocols

for remote task assignment are theoretically feasible (e.g. using ZeroMQ (“ZeroMQ”)), but are currently not implemented, establishing this latency as a core limitation for true millisecond real-time performance.

5.2 Perspectives

Arkitekt is not a finished development and is constantly iterated on, aiming to make it adaptable to new scenarios of bioimage analysis. While at the time of writing, this adaptation to the scenarios is discarded in favor of sustainability improvements such as bug-fixing and documentation, a few perspectives for the platform feel worthwhile exploring.

5.2.1 New Containerization Opportunities

Just like docker revolutionized the containerization of applications in the last decade, one technology is poised to provide a similar paradigm shift in this decade. WebAssembly (Rossberg 2019) originally emerged out of the need to provide efficient computation to the browser beyond JavaScript, and stands as an intermediary programming language that other languages can compile to. It enables sandboxed execution environments but also lower-level hardware access and is shaping to become a new solution for containerized applications. As such, it does not suffer from some container hurdlers, like *cold starts*, that limited the older technologies applicability to real-time workflows. Integration of the Web Assembly standard could therefore push Arkitekt to become a more reproducible and performant software solution, enabling containerized and reproducible environment, even when interfacing with microscopes.

5.2.2 Graphical Database management

Arkitekt currently employs the relational PostgreSQL (Rowe and Stonebraker 1987) database for its metadata storage. This provides an industry standard and reliable storage solution. However, as this metadata is considerably highly relational, a call for other database systems like graph databases could be justified. These database types, implemented for example in Neo4J (“Neo4j Graph Database"), can alleviate performance bottlenecks when highly relational data queries are executed against the database. Indeed, they were a core alternative consideration to the relation database type deployed in Arkitekt, but decided against as the developing community has not settled on a universal querying interface. Should these database types however find more widespread adoption, they would clearly warrant integration and could help uncover new highly relational insights in the connected network of microscopy data.

5.2.3 Extending to other Domains

This first iteration of Arkitekt provides data types to work with microscopy data. However, following the modular design decision of Arkitekt, only the *Mikro* service of the platform was specifically developed for microscopy data. The rest of the platform, including task assignment, workflow scheduling, authentication and authorization, app bundling, are standalone modules that can provide the power of real-time workflows to other data settings. One potential application could be the world of behavioral experiments that encounter a similar problem to the setting of Smart Microscopy: In closed loop experiments, analytical insights of the animals behavioral need to be fed back onto the ongoing experiment. A broad software arsenal in this space like Bonsai (Lopes et al. 2015) and more recently Autopilot (Saunders and Wehr 2019) have already shown the merit of applying reactive and distributed workflows in these settings, and can also integrate deep learning algorithms i.e. for pose estimation (Mathis et al. 2018). Joining efforts in finding a unifiable solution to both *worlds* could spark new experimental ideas, such as combining *behavior*-driven (e.g., through pose estimation) and *neuronal-event*-driven (e.g., through calcium imaging in vivo) closed loop experiments.

5.2.4 The Supergraph

Arkitekt in its current iteration relies on multiple API Endpoints per module that can be queried separately through their respective client. As discussed, this was a deliberate design decision to let developers separate concerns and develop apps that specifically focus on their given problem set. However, the exploration of scientific data is inherently highly relational and with the trends towards more multi-modal data, and given the call for more comprehensive and automated ontologies, Arkitekt could benefit from a more universal outward facing data API. One potential direction this could take is the *Supergraph* approach pioneered in projects like Wundergraph (“Wundergraph”, 2023) and Apollo Supergraph (“The Supergraph: A New Way to Think about GraphQL”, Apollo Team). This approach uses a federated protocol that merges multiple GraphQL Schemas and endpoints together in one unified accessible schema, that can then be used to retrieve highly related information or information storage in ontologies in one go, using the similar accessible Syntax of a GraphQL.

Figure 74 *The GraphQL Supergraph with two different services (here Mikro and the fictional Behavioral service). Both services can stand and develop completely standalone but can be merged to a unified and inspectable service.*

5.2.5 Large Language Model Integration

ChatGPT (“ChatGPT”, OpenAI) has dramatically reshaped our perceptions of what modern *Natural Language Processing* (NLP) networks are capable of. Recently, it and other *Large Language Models* have gained even more traction with their new ability to yield executable code, which can establish a *Language-Machine Interface* without any programming expertise. Recent discourse highlights the potential benefits of integrating intelligent agents more closely with microscopes to enable sophisticated interactions via language (Carpenter, Cimini, and Eliceiri 2023). However, despite their advanced capabilities, these models are not yet adept at understanding complex multi-layer systems programming, as employed in microscope control software. They excel at integrating with high-level interfaces, like NumPy

or Pandas, and when automating routine tasks. Arkitekt *Nodes*, with their well-documented interfaces to tools, could serve as a crucial interaction layer for *Large Language Models* to interface with hardware, providing an orthogonal additional way of automation to workflows, and enabling *conversational* microscopy.

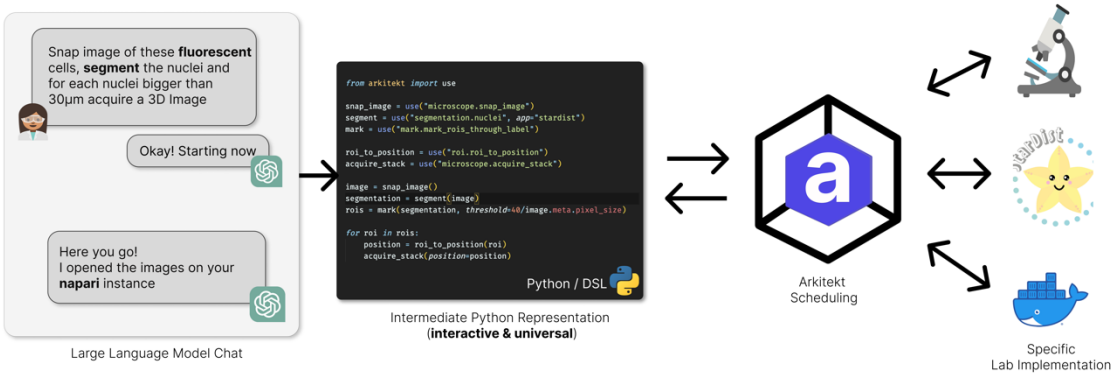

Figure 75 *A conceptual integration of large language models into the Arkitekt ecosystem.*

5.3 Concluding Remarks

The disciplines of modern microscopy and bioimage analysis have never been more exciting and are continuing to shape our understanding of what images of our biology can convey. Their methods and workflows allow us to delve ever deeper into the scales of our biology, uncovering new insights left and right, disobeying long-thought laws on the way.

However, in this flourishing field, the gap between developers and end-users is ever increasing, limiting the tools' full ability to advance the field they were designed to serve: biology.

Integrative solutions like ImJoy (Ouyang et al. 2019), BioImageIT (Prigent et al. 2022) and potentially Arkitekt will help in democratizing these tools, bringing them to wider audiences. Conceivably being disregarded as convenience methods, they serve a common goal: to further the collaborative nature of science and to bridge today's methods of a selective few, to methodologies for everyone.

6 Bibliography

"3rd Napari Survey Report (2022)." n.d. Google Docs. Accessed June 19, 2023. https://docs.google.com/document/d/1Ai8VCvMV-klw-Ws9N9dVbXQWZeIYoLqbCB1FZZ4GiK0/edit?usp=drivesdk&usp=embed_facebook.

Allan, Chris, Jean-Marie Burel, Josh Moore, Colin Blackburn, Melissa Linkert, Scott Loynton, Donald MacDonald, et al. 2012. "OMERO: Flexible, Model-Driven Data Management for Experimental Biology." *Nature Methods* 9 (3): 245–53. https://doi.org/10.1038/nmeth.1896.

Almada, Pedro, Pedro M. Pereira, Siân Culley, Ghislaine Caillol, Fanny Boroni-Rueda, Christina L. Dix, Guillaume Charras, et al. 2019. "Automating Multimodal Microscopy with NanoJ-Fluidics." *Nature Communications* 10 (1): 1223. https://doi.org/10.1038/s41467-019-09231-9.

Alvelid, Jonatan, Martina Damenti, Chiara Sgattoni, and Ilaria Testa. 2022. "Event-Triggered STED Imaging." *Nature Methods* 19 (10): 1268–75. https://doi.org/10.1038/s41592-022-01588-y.

"An Overview of Packaging for Python — Python Packaging User Guide." n.d. Accessed July 19, 2023. https://packaging.python.org/en/latest/overview/.

"Anaconda | The World's Most Popular Data Science Platform." n.d. Anaconda. Accessed July 19, 2023. https://www.anaconda.com.

"Apache Parquet." n.d. Accessed July 13, 2023. https://parquet.apache.org/.

Arnold, Ken, James Gosling, and David Holmes. 2005. *The Java Programming Language*. Addison Wesley Professional.

Beghin, Anne, Gianluca Grenci, Geetika Sahni, Su Guo, Harini Rajendiran, Tom Delaire, Saburnisha Binte Mohamad Raffi, et al. 2022. "Automated High-Speed 3D Imaging of Organoid Cultures with Multi-Scale Phenotypic Quantification." *Nature Methods* 19 (7): 881–92. https://doi.org/10.1038/s41592-022-01508-0.

Berg, Stuart, Dominik Kutra, Thorben Kroeger, Christoph N. Straehle, Bernhard X. Kausler, Carsten Haubold, Martin Schiegg, et al. 2019. "Ilastik: Interactive Machine Learning for (Bio)Image Analysis." *Nature Methods* 16 (12): 1226–32. https://doi.org/10.1038/s41592-019-0582-9.

Berman, H. M., J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne. 2000. "The Protein Data Bank." *Nucleic Acids Research* 28 (1): 235–42. https://doi.org/10.1093/nar/28.1.235.

Berthold, Michael R., Nicolas Cebron, Fabian Dill, Thomas R. Gabriel, Tobias Kötter, Thorsten Meinl, Peter Ohl, Kilian Thiel, and Bernd Wiswedel. 2009. "KNIME - the Konstanz Information Miner: Version 2.0 and Beyond." *SIGKDD Explor. Newsl.* 11 (1): 26–31. https://doi.org/10.1145/1656274.1656280.

Besson, Sébastien, Roger Leigh, Melissa Linkert, Chris Allan, Jean-Marie Burel, Mark Carroll, David Gault, et al. 2019. "Bringing Open Data to Whole Slide Imaging." In *Digital Pathology*, edited by Constantino Carlos Reyes-Aldasoro, Andrew Janowczyk, Mitko Veta, Peter Bankhead, and Korsuk Sirinukunwattana, 3–10. Lecture Notes in Computer Science. Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-23937-4_1.

Betzig, Eric, George H. Patterson, Rachid Sougrat, O. Wolf Lindwasser, Scott Olenych, Juan S. Bonifacino, Michael W. Davidson, Jennifer Lippincott-Schwartz, and Harald F. Hess. 2006. "Imaging Intracellular Fluorescent Proteins at Nanometer Resolution." *Science* 313 (5793): 1642–45. https://doi.org/10.1126/science.1127344.

BillWagner. n.d. ."NET Documentation." Accessed July 24, 2023. https://learn.microsoft.com/en-us/dotnet/.

Bitter, Rick, Taqi Mohiuddin, and Matt Nawrocki. 2006. *LabVIEW: Advanced Programming Techniques*. Crc Press.

Boehm, Ulrike, Glyn Nelson, Claire M. Brown, Steve Bagley, Peter Bajcsy, Johanna Bischof, Aurelien Dauphin, et al. 2021. "QUAREP-LiMi: A Community Endeavor to Advance Quality Assessment and Reproducibility in Light Microscopy." *Nature Methods* 18 (12): 1423–26. https://doi.org/10.1038/s41592-021-01162-y.

Boergens, Kevin M., Manuel Berning, Tom Bocklisch, Dominic Bräunlein, Florian Drawitsch, Johannes Frohnhofen, Tom Herold, et al. 2017. "WebKnossos: Efficient Online 3D Data Annotation for Connectomics." *Nature Methods* 14 (7): 691–94. https://doi.org/10.1038/nmeth.4331.

Cardona, Albert, and Pavel Tomancak. 2012. "Current Challenges in Open-Source Bioimage Informatics." *Nature Methods* 9 (7): 661–65. https://doi.org/10.1038/nmeth.2082.

Carpenter, Anne E., Beth A. Cimini, and Kevin W. Eliceiri. 2023. "Smart Microscopes of the Future." *Nature Methods* 20 (7): 962–64. https://doi.org/10.1038/s41592-023-01912-0.

Carpenter, Anne E., Thouis R. Jones, Michael R. Lamprecht, Colin Clarke, In Han Kang, Ola Friman, David A. Guertin, et al. 2006. "CellProfiler: Image Analysis Software for Identifying and Quantifying Cell Phenotypes." *Genome Biology* 7 (10): R100. https://doi.org/10.1186/gb-2006-7-10-r100.

Chamier, Lucas von, Romain F. Laine, Johanna Jukkala, Christoph Spahn, Daniel Krentzel, Elias Nehme, Martina Lerche, et al. 2021. "Democratising Deep Learning for Microscopy with ZeroCostDL4Mic." *Nature Communications* 12 (1): 2276. https://doi.org/10.1038/s41467-021-22518-0.

"ChatGPT." n.d. Accessed July 22, 2023. https://chat.openai.com.

Chiron, Lionel, Matthias Le Bec, Céline Cordier, Sylvain Pouzet, Dimitrije Milunov, Alvaro Banderas, Jean-Marc Di Meglio, Benoit Sorre, and Pascal Hersen. 2022. "CyberSco.Py an Open-Source Software for Event-Based, Conditional Microscopy." *Scientific Reports* 12 (1): 11579. https://doi.org/10.1038/s41598-022-15207-5.

Coons, Albert H., Hugh J. Creech, R. Norman Jones, and Ernst Berliner. 1942. "The Demonstration of Pneumococcal Antigen in Tissues by the Use of Fluorescent Antibody1." *The Journal of Immunology* 45 (3): 159–70. https://doi.org/10.4049/jimmunol.45.3.159.

Davis, Ian M. 2020. "Antoni van Leeuwenhoek and Measuring the Invisible: The Context of 16th and 17th Century Micrometry." *Studies in History and Philosophy of Science Part A* 83 (October): 75–85. https://doi.org/10.1016/j.shpsa.2020.03.004.

Denk, W., J. H. Strickler, and W. W. Webb. 1990. "Two-Photon Laser Scanning Fluorescence Microscopy." *Science (New York, N.Y.)* 248 (4951): 73–76. https://doi.org/10.1126/science.2321027.

Dennis, Jack B., and David P. Misunas. 1974. "A Preliminary Architecture for a Basic Data-Flow Processor." In *Proceedings of the 2nd Annual Symposium on Computer Architecture*, 126–32. ISCA '75. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/642089.642111.

Di Tommaso, Paolo, Maria Chatzou, Evan W. Floden, Pablo Prieto Barja, Emilio Palumbo, and Cedric Notredame. 2017. "Nextflow Enables Reproducible Computational Workflows." *Nature Biotechnology* 35 (4): 316–19. https://doi.org/10.1038/nbt.3820.

Dobson Ellen, T.A., Beth Cimini, H. Klemm Anna, Carolina Wählby, E. Carpenter Anne, and W Eliceiri Kevin. 2021. "ImageJ and CellProfiler: Complements in Open Source Bioimage Analysis." *Current Protocols* 1 (5): e89. https://doi.org/10.1002/cpz1.89.

"Docker: Accelerated, Containerized Application Development." 2022. May 10, 2022. https://www.docker.com/.

"Docker Desktop." 2023. Docker Documentation. July 24, 2023. https://docs.docker.com/desktop/.

"ECLIPSE Ti2 Series." n.d. Nikon Instruments Inc. Accessed July 23, 2023. https://www.microscope.healthcare.nikon.com/products/inverted-microscopes/eclipse-ti2-series.

Edelstein, Arthur D, Mark A. Tsuchida, Nenad Amodaj, Henry Pinkard, Ronald D. Vale, and Nico Stuurman. 2014. "Advanced Methods of Microscope Control Using MManager Software." *Journal of Biological Methods* 1 (2): e10. https://doi.org/10.14440/jbm.2014.36.

Egerton, Frank N. 1968. "Leeuwenhoek as a Founder of Animal Demography." *Journal of the History of Biology* 1 (1): 1–22. https://doi.org/10.1007/BF00149773.

Ester, Martin, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise." In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, 226–31. KDD'96. Portland, Oregon: AAAI Press.

Fallisch, Arne. 2023. "Community-Developed Checklists for Publishing Images and Image Analysis." *QUAREP* (blog). February 15, 2023. https://quarep.org/community-developed-checklists-for-publishing-images-and-image-analysis/.

Galland, Remi, Gianluca Grenci, Ajay Aravind, Virgile Viasnoff, Vincent Studer, and Jean-Baptiste Sibarita. 2015. "3D High- and Super-Resolution Imaging Using Single-Objective SPIM." *Nature Methods* 12 (7): 641–44. https://doi.org/10.1038/nmeth.3402.

Garcia-Lopez, Pablo, Virginia Garcia-Marin, and Miguel Freire. 2010. "The Histological Slides and Drawings of Cajal." *Frontiers in Neuroanatomy* 4 (March): 9. https://doi.org/10.3389/neuro.05.009.2010.

Garvey, Dave. 2023. "What Is API-First?" Tyk API Management. March 14, 2023. https://tyk.io/blog/res-what-is-api-first/.

Gentleman, Robert C., Vincent J. Carey, Douglas M. Bates, Ben Bolstad, Marcel Dettling, Sandrine Dudoit, Byron Ellis, et al. 2004. "Bioconductor: Open Software Development for Computational Biology and Bioinformatics." *Genome Biology* 5 (10): R80. https://doi.org/10.1186/gb-2004-5-10-r80.

"Get and Manage Add-Ons - MATLAB & Simulink - MathWorks France." n.d. Accessed July 20, 2023. https://fr.mathworks.com/help/matlab/matlab_env/get-add-ons.html.

Goedert, M., M. G. Spillantini, N. J. Cairns, and R. A. Crowther. 1992. "Tau Proteins of Alzheimer Paired Helical Filaments: Abnormal Phosphorylation of All Six Brain Isoforms." *Neuron* 8 (1): 159–68. https://doi.org/10.1016/0896-6273(92)90117-v.

Goldberg, Ilya G., Chris Allan, Jean-Marie Burel, Doug Creager, Andrea Falconi, Harry Hochheiser, Josiah Johnston, Jeff Mellen, Peter K. Sorger, and Jason R. Swedlow. 2005. "The Open Microscopy Environment (OME) Data Model and XML File: Open Tools for Informatics and Quantitative Analysis in Biological Imaging." *Genome Biology* 6 (5): R47. https://doi.org/10.1186/gb-2005-6-5-r47.

Gómez-de-Mariscal, Estibaliz, Carlos García-López-de-Haro, Wei Ouyang, Laurène Donati, Emma Lundberg, Michael Unser, Arrate Muñoz-Barrutia, and Daniel Sage. 2021. "DeepImageJ: A User-Friendly Environment to Run Deep Learning Models in ImageJ." *Nature Methods* 18 (10): 1192–95. https://doi.org/10.1038/s41592-021-01262-9.

"Google Maps." n.d. Google Maps. Accessed July 24, 2023. https://www.google.com/maps.

"GraphQL | A Query Language for Your API." n.d. Accessed July 25, 2023. https://graphql.org/.

Greenberg, Jane. 2009. "Understanding Metadata and Metadata Schemes." *Cataloging & Classification Quarterly*, October. https://doi.org/10.1300/J104v40n03_02.

Grenci, Gianluca, Florian Dilasser, Saburnisha Binte Mohamad Raffi, Marion Marchand, Mona Suryana, Geetika Sahni, Virgile Viasnoff, and Anne Beghin. 2022. "A High-Throughput Platform for Culture and 3D Imaging of Organoids." *Journal of Visualized Experiments: JoVE*, no. 188 (October). https://doi.org/10.3791/64405.

Gustafsson, M. G. L. 2000. “Surpassing the Lateral Resolution Limit by a Factor of Two Using Structured Illumination Microscopy.” *Journal of Microscopy* 198 (2): 82–87. https://doi.org/10.1046/j.1365-2818.2000.00710.x.

Haase, Robert, Akanksha Jain, Stéphane Rigaud, Daniela Vorkel, Pradeep Rajasekhar, Theresa Suckert, Talley J. Lambert, et al. 2020. “Interactive Design of GPU-Accelerated Image Data Flow Graphs and Cross-Platform Deployment Using Multi-Lingual Code Generation.” bioRxiv. https://doi.org/10.1101/2020.11.19.386565.

Hardt, Dick. 2012. “The OAuth 2.0 Authorization Framework.” Request for Comments RFC 6749. Internet Engineering Task Force. https://doi.org/10.17487/RFC6749.

Harris, Charles R., K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, et al. 2020. “Array Programming with NumPy.” *Nature* 585 (7825): 357–62. https://doi.org/10.1038/s41586-020-2649-2.

Hartley, Matthew, Gerard Kleywegt, Ardan Patwardhan, Ugis Sarkans, Jason R. Swedlow, and Alvis Brazma. 2021. “The BioImage Archive - Home of Life-Sciences Microscopy Data.” bioRxiv. https://doi.org/10.1101/2021.12.17.473169.

Hell, S. W., and J. Wichmann. 1994. “Breaking the Diffraction Resolution Limit by Stimulated Emission: Stimulated-Emission-Depletion Fluorescence Microscopy.” *Optics Letters* 19 (11): 780–82. https://doi.org/10.1364/ol.19.000780.

Hewitt, Carl, Peter Bishop, and Richard Steiger. 1973. “A Universal Modular ACTOR Formalism for Artificial Intelligence.” In *Proceedings of the 3rd International Joint Conference on Artificial Intelligence*, 235–45. IJCAI’73. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Hohlbein, Johannes, Benedict Diederich, Barbora Marsikova, Emmanuel G. Reynaud, Séamus Holden, Wiebke Jahr, Robert Haase, and Kirti Prakash. 2022. “Open Microscopy in the Life Sciences: Quo Vadis?” *Nature Methods* 19 (9): 1020–25. https://doi.org/10.1038/s41592-022-01602-3.

Hooke, Robert (1635-1703) Auteur du texte. 1665. *Micrographia or Some Physiological Descriptions of Minutes Bodies Made by Magnifying Glasses : With Observations and Inquiries Thereupon ([Reprod.]) / by R. Hooke,...* https://gallica.bnf.fr/ark:/12148/bpt6k98770v.

Huisken, Jan, Jim Swoger, Filippo Del Bene, Joachim Wittbrodt, and Ernst H. K. Stelzer. 2004. “Optical Sectioning Deep Inside Live Embryos by Selective Plane Illumination Microscopy.” *Science* 305 (5686): 1007–9. https://doi.org/10.1126/science.1100035.

“Imaris for Cell Biologists - Imaris.” n.d. Oxford Instruments. Accessed July 24, 2023. https://imaris.oxinst.com/products/imaris-for-cell-biologists.

“ImJoy RPC.” (2020) 2023. Python. ImJoy Team. https://github.com/imjoy-team/imjoy-rpc.

Inc, The MathWorks. 2022. “MATLAB Version: 9.13.0 (R2022b).” Natick, Massachusetts, United States: The MathWorks Inc. https://www.mathworks.com.

“Interactive Microscopy Control with ImJoy — Pycro-Manager Documentation.” n.d. Accessed June 14, 2023. https://pycro-manager.readthedocs.io/en/latest/application_notebooks/pycro_manager_imjoy_tutorial.html.

Jamali, Nasim, Ellen T. A. Dobson, Kevin W. Eliceiri, Anne E. Carpenter, and Beth A. Cimini. 2021. “2020 BioImage Analysis Survey: Community Experiences and Needs for the Future.” *Biological Imaging* 1: e4. https://doi.org/10.1017/S2633903X21000039.

Jaquet-Chiffelle, David-Olivier, Eoghan Casey, and Jonathan Bourquenoud. 2020. “Tamperproof Timestamped Provenance Ledger Using Blockchain Technology.” *Forensic Science International: Digital Investigation* 33 (June): 300977. https://doi.org/10.1016/j.fsidi.2020.300977.

Jones, M., J. Bradley, and N. Sakimura. 2015. “JSON Web Token (JWT).” RFC7519. RFC Editor. https://doi.org/10.17487/RFC7519.

Jones, Thouis R., In Han Kang, Douglas B. Wheeler, Robert A. Lindquist, Adam Papallo, David M. Sabatini, Polina Golland, and Anne E. Carpenter. 2008. “CellProfiler Analyst: Data

Exploration and Analysis Software for Complex Image-Based Screens." *BMC Bioinformatics* 9 (1): 482. https://doi.org/10.1186/1471-2105-9-482.
Jupyter, Project, Matthias Bussonnier, Jessica Forde, Jeremy Freeman, Brian Granger, Tim Head, Chris Holdgraf, et al. 2018. "Binder 2.0 - Reproducible, Interactive, Sharable Environments for Science at Scale." *Proceedings of the 17th Python in Science Conference*, 113–20. https://doi.org/10.25080/Majora-4af1f417-011.
Kalaš, Matúš, Laure Plantard, Nataša Sladoje, Joakim Lindblad, Moritz Alexander Kirschmann, Martin Jones, Anatole Chessel, et al. 2019. "<p>EDAM-Bioimaging: The Ontology of Bioimage Informatics Operations, Topics, Data, and Formats (2019 Update)</P>." *F1000Research* 8 (February). https://doi.org/10.7490/f1000research.1116432.1.
Kamentsky, Lee, Thouis R. Jones, Adam Fraser, Mark-Anthony Bray, David J. Logan, Katherine L. Madden, Vebjorn Ljosa, Curtis Rueden, Kevin W. Eliceiri, and Anne E. Carpenter. 2011. "Improved Structure, Function and Compatibility for CellProfiler: Modular High-Throughput Image Analysis Software." *Bioinformatics (Oxford, England)* 27 (8): 1179–80. https://doi.org/10.1093/bioinformatics/btr095.
Kechkar, Adel, Deepak Nair, Mike Heilemann, Daniel Choquet, and Jean-Baptiste Sibarita. 2013. "Real-Time Analysis and Visualization for Single-Molecule Based Super-Resolution Microscopy." *PLoS ONE* 8 (4): e62918. https://doi.org/10.1371/journal.pone.0062918.
Keller, Mark S., Ilan Gold, Chuck McCallum, Trevor Manz, Peter V. Kharchenko, and Nils Gehlenborg. 2021. "Vitessce: A Framework for Integrative Visualization of Multi-Modal and Spatially-Resolved Single-Cell Data." OSF Preprints. https://doi.org/10.31219/osf.io/y8thv.
Keller, Philipp J., Misha B. Ahrens, and Jeremy Freeman. 2015. "Light-Sheet Imaging for Systems Neuroscience." *Nature Methods* 12 (1): 27–29. https://doi.org/10.1038/nmeth.3214.
Keller, Philipp J., Annette D. Schmidt, Joachim Wittbrodt, and Ernst H.K. Stelzer. 2008. "Reconstruction of Zebrafish Early Embryonic Development by Scanned Light Sheet Microscopy." *Science* 322 (5904): 1065–69. https://doi.org/10.1126/science.1162493.
Kervrann, Charles, and Jérôme Boulanger. 2006. "Optimal Spatial Adaptation for Patch-Based Image Denoising." *IEEE Transactions on Image Processing: A Publication of the IEEE Signal Processing Society* 15 (10): 2866–78. https://doi.org/10.1109/tip.2006.877529.
Kirsch, Russel A. 1978. "Algorithms for Image Analysis of Wood Pulp Fibers." NBS IR 78-1442. 0 ed. Gaithersburg, MD: National Bureau of Standards. https://doi.org/10.6028/NBS.IR.78-1442.
Kluyver, Thomas, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, et al. 2016. "Jupyter Notebooks – a Publishing Format for Reproducible Computational Workflows." In *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, edited by F. Loizides and B. Schmidt, 87–90. IOS Press.
"KNIME Community Hub." n.d. KNIME Community Hub. Accessed July 19, 2023. https://hub.knime.com/.
"Knime-Bridge." (2015) 2015. Java. CellProfiler. https://github.com/CellProfiler/knime-bridge.
Knoll, M., and E. Ruska. 1932. "Beitrag Zur Geometrischen Elektronenoptik. I." *Annalen Der Physik* 404 (5): 607–40. https://doi.org/10.1002/andp.19324040506.
Krausz, Eberhard, Ronald de Hoogt, Emmanuel Gustin, Frans Cornelissen, Thierry Grand-Perret, Lut Janssen, Nele Vloemans, et al. 2013. "Translation of a Tumor Microenvironment Mimicking 3D Tumor Growth Co-Culture Assay Platform to High-Content Screening." *Journal of Biomolecular Screening* 18 (1): 54–66. https://doi.org/10.1177/1087057112456874.

"Kubernetes Documentation." n.d. Kubernetes. Accessed July 25, 2023. https://kubernetes.io/docs/home/.

Kusumoto, Dai, Tomohisa Seki, Hiromune Sawada, Akira Kunitomi, Toshiomi Katsuki, Mai Kimura, Shogo Ito, et al. 2021. "Anti-Senescent Drug Screening by Deep Learning-Based Morphology Senescence Scoring." *Nature Communications* 12 (1): 257. https://doi.org/10.1038/s41467-020-20213-0.

Laine, Romain F., Ignacio Arganda-Carreras, Ricardo Henriques, and Guillaume Jacquemet. 2021. "Avoiding a Replication Crisis in Deep-Learning-Based Bioimage Analysis." *Nature Methods* 18 (10): 1136–44. https://doi.org/10.1038/s41592-021-01284-3.

Lan, Gongjin, Ting Liu, Xu Wang, Xueli Pan, and Zhisheng Huang. 2022. "A Semantic Web Technology Index." *Scientific Reports* 12 (1): 3672. https://doi.org/10.1038/s41598-022-07615-4.

Leach, Paul J., Rich Salz, and Michael H. Mealling. 2005. "A Universally Unique IDentifier (UUID) URN Namespace." Request for Comments RFC 4122. Internet Engineering Task Force. https://doi.org/10.17487/RFC4122.

LeCun, Yann, Bernhard Boser, John S. Denker, Donnie Henderson, Richard E. Howard, Wayne Hubbard, and Lawrence D. Jackel. 1989. "Backpropagation Applied to Handwritten Zip Code Recognition." *Neural Computation* 1 (4): 541–51.

Levet, Florian, Anne E. Carpenter, Kevin W. Eliceiri, Anna Kreshuk, Peter Bankhead, and Robert Haase. 2021. "Developing Open-Source Software for Bioimage Analysis: Opportunities and Challenges." *F1000Research* 10 (April): 302. https://doi.org/10.12688/f1000research.52531.1.

Levet, Florian, Jan Tønnesen, U. Valentin Nägerl, and Jean-Baptiste Sibarita. 2020. "SpineJ: A Software Tool for Quantitative Analysis of Nanoscale Spine Morphology." *Methods*, Progress in Super-resolution Fluorescence Microscopy, 174 (March): 49–55. https://doi.org/10.1016/j.ymeth.2020.01.020.

Lex Fridman, dir. 2022. *Guido van Rossum: Python and the Future of Programming | Lex Fridman Podcast #341*. https://www.youtube.com/watch?v=-DVyjdw4t9I.

Li, Xinyang, Guoxun Zhang, Jiamin Wu, Yuanlong Zhang, Zhifeng Zhao, Xing Lin, Hui Qiao, et al. 2021. "Reinforcing Neuron Extraction and Spike Inference in Calcium Imaging Using Deep Self-Supervised Denoising." *Nature Methods* 18 (11): 1395–1400. https://doi.org/10.1038/s41592-021-01225-0.

"Logiciel tableur Microsoft Excel | Microsoft 365." n.d. Accessed July 20, 2023. https://www.microsoft.com/fr-fr/microsoft-365/excel.

Lopes, Gonçalo, Niccolò Bonacchi, João Frazão, Joana P. Neto, Bassam V. Atallah, Sofia Soares, Luís Moreira, et al. 2015. "Bonsai: An Event-Based Framework for Processing and Controlling Data Streams." *Frontiers in Neuroinformatics* 9. https://www.frontiersin.org/articles/10.3389/fninf.2015.00007.

"Magicgui." n.d. Accessed July 17, 2023. https://pyapp-kit.github.io/magicgui/.

Mahecic, Dora, Willi L. Stepp, Chen Zhang, Juliette Griffié, Martin Weigert, and Suliana Manley. 2022. "Event-Driven Acquisition for Content-Enriched Microscopy." *Nature Methods* 19 (10): 1262–67. https://doi.org/10.1038/s41592-022-01589-x.

Manz, Trevor, Ilan Gold, Nathan Heath Patterson, Chuck McCallum, Mark S. Keller, Bruce W. Herr, Katy Börner, Jeffrey M. Spraggins, and Nils Gehlenborg. 2022. "Viv: Multiscale Visualization of High-Resolution Multiplexed Bioimaging Data on the Web." *Nature Methods* 19 (5): 515–16. https://doi.org/10.1038/s41592-022-01482-7.

Martin, Cathie, and Mike Blatt. 2013. "Manipulation and Misconduct in the Handling of Image Data." *The Plant Cell* 25 (9): 3147–48. https://doi.org/10.1105/tpc.113.250980.

Mathis, Alexander, Pranav Mamidanna, Kevin M. Cury, Taiga Abe, Venkatesh N. Murthy, Mackenzie Weygandt Mathis, and Matthias Bethge. 2018. "DeepLabCut: Markerless Pose Estimation of User-Defined Body Parts with Deep Learning." *Nature Neuroscience* 21 (9): 1281–89. https://doi.org/10.1038/s41593-018-0209-y.

Matsakis, Nicholas D, and Felix S Klock II. 2014. "The Rust Language." In *ACM SIGAda Ada Letters*, 34:103–4. ACM.

Meijering, Erik. 2020. “A Bird’s-Eye View of Deep Learning in Bioimage Analysis.” *Computational and Structural Biotechnology Journal* 18 (January): 2312–25. https://doi.org/10.1016/j.csbj.2020.08.003.
“Metadata.” n.d. Accessed July 19, 2023. https://galaxyproject.org/support/metadata/.
“MetaMorph, Microscope Imaging, Microscopy Analysis Software | Molecular Devices.” n.d. Accessed July 23, 2023. https://www.moleculardevices.com/products/cellular-imaging-systems/acquisition-and-analysis-software/metamorph-microscopy.
“Method of the Year 2014.” 2015. *Nature Methods* 12 (1): 1–1. https://doi.org/10.1038/nmeth.3251.
“Method of the Year 2017: Organoids.” 2018. *Nature Methods* 15 (1): 1–1. https://doi.org/10.1038/nmeth.4575.
“Microservices.” n.d. Martinfowler.Com. Accessed July 25, 2023. https://martinfowler.com/articles/microservices.html.
Miles, Alistair, John Kirkham, Martin Durant, James Bourbeau, Tarik Onalan, Joe Hamman, Zain Patel, et al. 2020. “Zarr-Developers/Zarr-Python: V2.4.0.” Zenodo. https://doi.org/10.5281/zenodo.3773450.
“MinIO | High Performance, Kubernetes Native Object Storage.” n.d. Accessed July 13, 2023. https://min.io/.
Mitchell, Sonia Natalie, Andrew Lahiff, Nathan Cummings, Jonathan Hollocombe, Bram Boskamp, Ryan Field, Dennis Reddyhoff, et al. 2022. “FAIR Data Pipeline: Provenance-Driven Data Management for Traceable Scientific Workflows.” *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 380 (2233): 20210300. https://doi.org/10.1098/rsta.2021.0300.
Miura, Kota, Sébastien Tosi, Christoph Möhl, Chong Zhang, Perrine Paul-Gilloteaux, Ulrike Schulze, Simon F Norrelykke, Christian Tischer, and Thomas Pengo. n.d. “Bioimage Analysis Tools.”
Moore, Josh, Chris Allan, Sébastien Besson, Jean-Marie Burel, Erin Diel, David Gault, Kevin Kozlowski, et al. 2021. “OME-NGFF: A next-Generation File Format for Expanding Bioimaging Data-Access Strategies.” *Nature Methods* 18 (12): 1496–98. https://doi.org/10.1038/s41592-021-01326-w.
Moore, Josh, Daniela Basurto-Lozada, Sébastien Besson, John Bogovic, Jordão Bragantini, Eva M. Brown, Jean-Marie Burel, et al. 2023. “OME-Zarr: A Cloud-Optimized Bioimaging File Format with International Community Support.” bioRxiv. https://doi.org/10.1101/2023.02.17.528834.
Moreno, Xavier Casas, Staffan Al-Kadhimi, Jonatan Alvelid, Andreas Bodén, and Ilaria Testa. 2021. “ImSwitch: Generalizing Microscope Control in Python.” *Journal of Open Source Software* 6 (64): 3394. https://doi.org/10.21105/joss.03394.
“Neo4j Graph Database & Analytics – The Leader in Graph Databases.” n.d. Graph Database & Analytics. Accessed July 22, 2023. https://neo4j.com/.
“Neuroglancer.” (2016) 2023. TypeScript. Google. https://github.com/google/neuroglancer.
Nogare, Damian Dalle, Matthew Hartley, Joran Deschamps, Jan Ellenberg, and Florian Jug. 2023. “Using AI in Bioimage Analysis to Elevate the Rate of Scientific Discovery as a Community.” *Nature Methods* 20 (7): 973–75. https://doi.org/10.1038/s41592-023-01929-5.
“Npe2.” (2021) 2023. Python. napari. https://github.com/napari/npe2.
Oleś, Andrzej, Gregoire Pau, Mike Smith, Oleg Sklyar, Wolfgang Huber, with contributions from Joseph Barry, and Philip A. Marais. 2023. “EBImage: Image Processing and Analysis Toolbox for R.” Bioconductor version: Release (3.17). https://doi.org/10.18129/B9.bioc.EBImage.
“Ontology Lookup Service < EMBL-EBI.” n.d. Accessed July 13, 2023. https://www.ebi.ac.uk/ols/index.
Ouyang, Wei, Fynn Beuttenmueller, Estibaliz Gómez-de-Mariscal, Constantin Pape, Tom Burke, Carlos Garcia-López-de-Haro, Craig Russell, et al. 2022. “BioImage Model Zoo:

A Community-Driven Resource for Accessible Deep Learning in BioImage Analysis." bioRxiv. https://doi.org/10.1101/2022.06.07.495102.
Ouyang, Wei, Florian Mueller, Martin Hjelmare, Emma Lundberg, and Christophe Zimmer. 2019. "ImJoy: An Open-Source Computational Platform for the Deep Learning Era." *Nature Methods* 16 (12): 1199–1200. https://doi.org/10.1038/s41592-019-0627-0.
"Overview - C2PA." n.d. Accessed July 22, 2023. https://c2pa.org/.
Paul-Gilloteaux, Perrine, Sébastien Tosi, Jean-Karim Hériché, Alban Gaignard, Hervé Ménager, Raphaël Marée, Volker Baecker, et al. 2021. "Bioimage Analysis Workflows: Community Resources to Navigate through a Complex Ecosystem." *F1000Research* 10 (April): 320. https://doi.org/10.12688/f1000research.52569.1.
"Perceptrons." n.d. MIT Press. Accessed June 25, 2023. https://mitpress.mit.edu/9780262630221/perceptrons/.
"PERFECT FOCUS UNIT T-PFS Perfect Focus Unit Instructions." n.d.
Petráň, Mojmír, Milan Hadravský, M. David Egger, and Robert Galambos. 1968. "Tandem-Scanning Reflected-Light Microscope*." *JOSA* 58 (5): 661–64. https://doi.org/10.1364/JOSA.58.000661.
Pinkard, Henry, Nico Stuurman, Ivan E. Ivanov, Nicholas M. Anthony, Wei Ouyang, Bin Li, Bin Yang, et al. 2021. "Pycro-Manager: Open-Source Software for Customized and Reproducible Microscope Control." *Nature Methods* 18 (3): 226–28. https://doi.org/10.1038/s41592-021-01087-6.
Potocek, Pavel, Patrick Trampert, Maurice Peemen, Remco Schoenmakers, and Tim Dahmen. 2020. "Sparse Scanning Electron Microscopy Data Acquisition and Deep Neural Networks for Automated Segmentation in Connectomics." *Microscopy and Microanalysis* 26 (April): 1–10. https://doi.org/10.1017/S1431927620001361.
Prigent, Sylvain, Cesar Augusto Valades-Cruz, Ludovic Leconte, Léo Maury, Jean Salamero, and Charles Kervrann. 2022. "BioImageIT: Open-Source Framework for Integration of Image Data Management with Analysis." *Nature Methods* 19 (11): 1328–30. https://doi.org/10.1038/s41592-022-01642-9.
"Project Jupyter." n.d. Accessed July 25, 2023. https://jupyter.org.
"Qt | Tools for Each Stage of Software Development Lifecycle." n.d. Accessed June 21, 2023. https://www.qt.io.
"R: The R Project for Statistical Computing." n.d. Accessed July 22, 2023. https://www.r-project.org/.
"RabbitMQ: Easy to Use, Flexible Messaging and Streaming — RabbitMQ." n.d. Accessed July 25, 2023. https://www.rabbitmq.com/.
"RDF 1.2 Schema." n.d. Accessed July 24, 2023. https://www.w3.org/TR/rdf12-schema/.
"React." n.d. Accessed July 25, 2023. https://react.dev/.
Reinhardt, Susanne C. M., Luciano A. Masullo, Isabelle Baudrexel, Philipp R. Steen, Rafal Kowalewski, Alexandra S. Eklund, Sebastian Strauss, et al. 2023. "Ångström-Resolution Fluorescence Microscopy." *Nature* 617 (7962): 711–16. https://doi.org/10.1038/s41586-023-05925-9.
Renz, Malte. 2013. "Fluorescence Microscopy—A Historical and Technical Perspective." *Cytometry Part A* 83 (9): 767–79. https://doi.org/10.1002/cyto.a.22295.
Roberts, Lawrence G. 1963. "Machine Perception of Three-Dimensional Solids." Thesis, Massachusetts Institute of Technology. https://dspace.mit.edu/handle/1721.1/11589.
Rocklin, Matthew. 2015. "Dask: Parallel Computation with Blocked Algorithms and Task Scheduling." In *Proceedings of the 14th Python in Science Conference*. Citeseer.
Rossberg, Andreas. 2019. "WebAssembly Core Specification." W3C. https://www.w3.org/TR/wasm-core-1/.
Rowe, Lawrence A., and Michael R. Stonebraker. 1987. "The POSTGRES Data Model:" Fort Belvoir, VA: Defense Technical Information Center. https://doi.org/10.21236/ADA184251.

Royer, Loïc A., William C. Lemon, Raghav K. Chhetri, Yinan Wan, Michael Coleman, Eugene W. Myers, and Philipp J. Keller. 2016. “Adaptive Light-Sheet Microscopy for Long-Term, High-Resolution Imaging in Living Organisms.” *Nature Biotechnology* 34 (12): 1267–78. https://doi.org/10.1038/nbt.3708.

Rueden, Curtis T., Mark C. Hiner, Edward L. Evans, Michael A. Pinkert, Alice M. Lucas, Anne E. Carpenter, Beth A. Cimini, and Kevin W. Eliceiri. 2022. “PyImageJ: A Library for Integrating ImageJ and Python.” *Nature Methods* 19 (11): 1326–27. https://doi.org/10.1038/s41592-022-01655-4.

Ruska, Ernst. 1987. “The Development of the Electron Microscope and of Electron Microscopy.” *Bioscience Reports* 7 (8): 607–29. https://doi.org/10.1007/BF01127674.

Rust, Michael J., Mark Bates, and Xiaowei Zhuang. 2006. “Sub-Diffraction-Limit Imaging by Stochastic Optical Reconstruction Microscopy (STORM).” *Nature Methods* 3 (10): 793–96. https://doi.org/10.1038/nmeth929.

“RxMarbles: Interactive Diagrams of Rx Observables.” n.d. Accessed July 13, 2023. https://rxmarbles.com/.

Sarkans, Ugis, Wah Chiu, Lucy Collinson, Michele C. Darrow, Jan Ellenberg, David Grunwald, Jean-Karim Hériché, et al. 2021. “REMBI: Recommended Metadata for Biological Images—Enabling Reuse of Microscopy Data in Biology.” *Nature Methods* 18 (12): 1418–22. https://doi.org/10.1038/s41592-021-01166-8.

Saunders, Jonny L., and Michael Wehr. 2019. “Autopilot: Automating Behavioral Experiments with Lots of Raspberry Pis.” bioRxiv. https://doi.org/10.1101/807693.

Schindelin, Johannes, Ignacio Arganda-Carreras, Erwin Frise, Verena Kaynig, Mark Longair, Tobias Pietzsch, Stephan Preibisch, et al. 2012. “Fiji: An Open-Source Platform for Biological-Image Analysis.” *Nature Methods* 9 (7): 676–82. https://doi.org/10.1038/nmeth.2019.

Schmidt, Christian, Janina Hanne, Josh Moore, Christian Meesters, Elisa Ferrando-May, and Stefanie Weidtkamp-Peters. 2022a. “Research Data Management for Bioimaging: The 2021 NFDI4BIOIMAGE Community Survey - Extended Data 1 - Supplementary Information and Supplementary Figures,” September. https://doi.org/10.5281/zenodo.7082514.

———. 2022b. “Research Data Management for Bioimaging: The 2021 NFDI4BIOIMAGE Community Survey.” *F1000Research* 11 (September): 638. https://doi.org/10.12688/f1000research.121714.2.

Schneider, Caroline A., Wayne S. Rasband, and Kevin W. Eliceiri. 2012. “NIH Image to ImageJ: 25 Years of Image Analysis.” *Nature Methods* 9 (7): 671–75. https://doi.org/10.1038/nmeth.2089.

Shariff, Aabid, Joshua Kangas, Luis Pedro Coelho, Shannon Quinn, and Robert F. Murphy. 2010. “Automated Image Analysis for High-Content Screening and Analysis.” *SLAS Discovery*, High-Content Screening, Imaging, & Data Analysis, 15 (7): 726–34. https://doi.org/10.1177/1087057110370894.

Sharonov, Alexey, and Robin M. Hochstrasser. 2006. “Wide-Field Subdiffraction Imaging by Accumulated Binding of Diffusing Probes.” *Proceedings of the National Academy of Sciences* 103 (50): 18911–16. https://doi.org/10.1073/pnas.0609643104.

Shibanai, Kazuhiro, and Takuo Watanabe. 2018. “Distributed Functional Reactive Programming on Actor-Based Runtime.” In *Proceedings of the 8th ACM SIGPLAN International Workshop on Programming Based on Actors, Agents, and Decentralized Control*, 13–22. AGERE 2018. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3281366.3281370.

Sibarita, Jean-Baptiste. 2005. “Deconvolution Microscopy.” In *Microscopy Techniques: -/-*, edited by Jens Rietdorf, 201–43. Advances in Biochemical Engineering. Berlin, Heidelberg: Springer. https://doi.org/10.1007/b102215.

Sivagurunathan, Suganya, Stefania Marcotti, Carl J. Nelson, Martin L. Jones, David J. Barry, Thomas J. A. Slater, Kevin W. Eliceiri, and Beth A. Cimini. 2023. “Bridging Imaging

Users to Imaging Analysis - A Community Survey." bioRxiv. https://doi.org/10.1101/2023.06.05.543701.
Sofroniew, Nicholas, Talley Lambert, Kira Evans, Juan Nunez-Iglesias, Grzegorz Bokota, Philip Winston, Gonzalo Peña-Castellanos, et al. 2022. "Napari: A Multi-Dimensional Image Viewer for Python." Zenodo. https://doi.org/10.5281/zenodo.7276432.
"Stack Overflow Developer Survey 2023." n.d. Stack Overflow. Accessed July 24, 2023. https://survey.stackoverflow.co/2023/?utm_source=social-share&utm_medium=social&utm_campaign=dev-survey-2023.
Strack, Rita. 2020. "Smarter Microscopes." *Nature Methods* 17 (1): 23–23. https://doi.org/10.1038/s41592-019-0708-0.
Strano, Giorgio. 2009. "Galileo's Telescope: History, Scientific Analysis, and Replicated Observations." *Experimental Astronomy* 25 (1): 17–31. https://doi.org/10.1007/s10686-009-9142-0.
Stringer, Carsen, Tim Wang, Michalis Michaelos, and Marius Pachitariu. 2021. "Cellpose: A Generalist Algorithm for Cellular Segmentation." *Nature Methods* 18 (1): 100–106. https://doi.org/10.1038/s41592-020-01018-x.
Sun, Duxin, Wei Gao, Hongxiang Hu, and Simon Zhou. 2022. "Why 90% of Clinical Drug Development Fails and How to Improve It?" *Acta Pharmaceutica Sinica. B* 12 (7): 3049–62. https://doi.org/10.1016/j.apsb.2022.02.002.
Susano Pinto, David Miguel, Mick A. Phillips, Nicholas Hall, Julio Mateos-Langerak, Danail Stoychev, Tiago Susano Pinto, Martin J. Booth, Ilan Davis, and Ian M. Dobbie. 2021. "Python-Microscope – a New Open-Source Python Library for the Control of Microscopes." *Journal of Cell Science* 134 (19): jcs258955. https://doi.org/10.1242/jcs.258955.
Syme, Don, Tomas Petricek, and Dmitry Lomov. 2011. "The F# Asynchronous Programming Model." In *Practical Aspects of Declarative Languages*, edited by Ricardo Rocha and John Launchbury, 175–89. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer. https://doi.org/10.1007/978-3-642-18378-2_15.
Tailscale. 2023. "Tailscale Quickstart." Tailscale. May 4, 2023. https://tailscale.com/kb/1017/install/.
Tedersoo, Leho, Rainer Küngas, Ester Oras, Kajar Köster, Helen Eenmaa, Äli Leijen, Margus Pedaste, et al. 2021. "Data Sharing Practices and Data Availability upon Request Differ across Scientific Disciplines." *Scientific Data* 8 (1): 192. https://doi.org/10.1038/s41597-021-00981-0.
Tenopir, Carol, Elizabeth D. Dalton, Suzie Allard, Mike Frame, Ivanka Pjesivac, Ben Birch, Danielle Pollock, and Kristina Dorsett. 2015. "Changes in Data Sharing and Data Reuse Practices and Perceptions among Scientists Worldwide." *PLOS ONE* 10 (8): e0134826. https://doi.org/10.1371/journal.pone.0134826.
Teodoro, George, Tony Pan, Tahsin M. Kurc, Jun Kong, Lee A.D. Cooper, Norbert Podhorszki, Scott Klasky, and Joel H. Saltz. 2013. "High-Throughput Analysis of Large Microscopy Image Datasets on CPU-GPU Cluster Platforms." In *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, 103–14. https://doi.org/10.1109/IPDPS.2013.11.
"The Apache Groovy Programming Language - Groovy Reference Documentation." n.d. Accessed July 24, 2023. https://www.groovy-lang.org/single-page-documentation.html.
The HDF Group. 2000. "Hierarchical Data Format Version 5." 2000. http://www.hdfgroup.org/HDF5.
"The Nobel Prize in Chemistry 2014." n.d. NobelPrize.Org. Accessed July 22, 2023. https://www.nobelprize.org/prizes/chemistry/2014/summary/.
"The Supergraph: A New Way to Think about GraphQL." n.d. Apollo GraphQL Blog. Accessed July 22, 2023. https://www.apollographql.com/blog/announcement/backend/the-supergraph-a-new-way-to-think-about-graphql/.

Tinevez, Jean-Yves, Nick Perry, Johannes Schindelin, Genevieve M. Hoopes, Gregory D. Reynolds, Emmanuel Laplantine, Sebastian Y. Bednarek, Spencer L. Shorte, and Kevin W. Eliceiri. 2017. “TrackMate: An Open and Extensible Platform for Single-Particle Tracking.” *Methods*, Image Processing for Biologists, 115 (February): 80–90. https://doi.org/10.1016/j.ymeth.2016.09.016.

Tønnesen, Jan, V. V. G. Krishna Inavalli, and U. Valentin Nägerl. 2018. “Super-Resolution Imaging of the Extracellular Space in Living Brain Tissue.” *Cell* 172 (5): 1108-1121.e15. https://doi.org/10.1016/j.cell.2018.02.007.

Van Rossum, Guido, and Fred L Drake Jr. 1995. *Python Reference Manual*. Centrum voor Wiskunde en Informatica Amsterdam.

Veiga Leprevost, Felipe da, Björn A Grüning, Saulo Alves Aflitos, Hannes L Röst, Julian Uszkoreit, Harald Barsnes, Marc Vaudel, et al. 2017. “BioContainers: An Open-Source and Community-Driven Framework for Software Standardization.” *Bioinformatics* 33 (16): 2580–82. https://doi.org/10.1093/bioinformatics/btx192.

Virtanen, Pauli, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, et al. 2020. “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python.” *Nature Methods* 17: 261–72. https://doi.org/10.1038/s41592-019-0686-2.

Wallis, J.W., T.R. Miller, C.A. Lerner, and E.C. Kleerup. 1989. “Three-Dimensional Display in Nuclear Medicine.” *IEEE Transactions on Medical Imaging* 8 (4): 297–230. https://doi.org/10.1109/42.41482.

Weigert, Martin, Uwe Schmidt, Tobias Boothe, Andreas Müller, Alexandr Dibrov, Akanksha Jain, Benjamin Wilhelm, et al. 2018. “Content-Aware Image Restoration: Pushing the Limits of Fluorescence Microscopy.” *Nature Methods* 15 (12): 1090–97. https://doi.org/10.1038/s41592-018-0216-7.

Weigert, Martin, Uwe Schmidt, Robert Haase, Ko Sugawara, and Gene Myers. 2020. “Star-Convex Polyhedra for 3D Object Detection and Segmentation in Microscopy.” In *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 3655–62. Snowmass Village, CO, USA: IEEE. https://doi.org/10.1109/WACV45572.2020.9093435.

“Welcome to Pulsar’s Documentation! — Pulsar 0.15.0.Dev0 Documentation.” n.d. Accessed July 25, 2023. https://pulsar.readthedocs.io/en/latest/.

“What Is Cloud Native.” n.d. Google Cloud. Accessed July 25, 2023. https://cloud.google.com/learn/what-is-cloud-native.

Wilkinson, Mark D., Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, et al. 2016. “The FAIR Guiding Principles for Scientific Data Management and Stewardship.” *Scientific Data* 3 (1): 160018. https://doi.org/10.1038/sdata.2016.18.

Woodward, J. J. 1876. “The Application of Photography to Micrometry, with Special Reference to the Micrometry of Blood in Criminal Cases.” *The Monthly Microscopical Journal* 16 (3): 144–54. https://doi.org/10.1111/j.1365-2818.1876.tb01816.x.

Wooster, Richard, Susan L. Neuhausen, Jonathan Mangion, Yvette Quirk, Deborah Ford, Nadine Collins, Kim Nguyen, et al. 1994. “Localization of a Breast Cancer Susceptibility Gene, BRCA2, to Chromosome 13q12-13.” *Science* 265 (5181): 2088–90. https://doi.org/10.1126/science.8091231.

“Wundergraph/Wundergraph.” (2022) 2023. TypeScript. WunderGraph. https://github.com/wundergraph/wundergraph.

Yang, Bin, Merlin Lange, Alfred Millett-Sikking, Xiang Zhao, Jordão Bragantini, Shruthi VijayKumar, Mason Kamb, et al. 2022. “DaXi—High-Resolution, Large Imaging Volume and Multi-View Single-Objective Light-Sheet Microscopy.” *Nature Methods* 19 (4): 461–69. https://doi.org/10.1038/s41592-022-01417-2.

Yoo, Andy B., Morris A. Jette, and Mark Grondona. 2003. “SLURM: Simple Linux Utility for Resource Management.” In *Job Scheduling Strategies for Parallel Processing*, edited by Dror Feitelson, Larry Rudolph, and Uwe Schwiegelshohn, 44–60. Lecture Notes in

Computer Science. Berlin, Heidelberg: Springer. https://doi.org/10.1007/10968987_3.
“ZeroMQ.” n.d. Accessed July 22, 2023. https://zeromq.org/.

7 Annex

This annex includes two peer-reviewed articles that were published in the course of this thesis and that I had the opportunity to co-author. They constitute the culmination of efforts in side-projects that were part of my PhD but did not fit under the thematic umbrella of the Arkitekt project.

Additionally, a second section includes exemplary code snippets, which found their mention during this thesis.

Imaging dendritic spines in the hippocampus of a living mouse by 3D-stimulated emission depletion microscopy

Stéphane Bancelin[a,*] **Luc Mercier**[a] **Johannes Roos**[a] **Mohamed Belkadi**[a] **Thomas Pfeiffer**[a] **Sun Kwang Kim**[b] **and U. Valentin Nägerl**[a,*]

[a]University of Bordeaux, CNRS, Interdisciplinary Institute for Neuroscience, IINS, UMR 5297, Bordeaux, France

[b]Kyung Hee University, Graduate School, Department of Science in Korean Medicine, Seoul, Republic of Korea

Abstract

Significance: Stimulated emission depletion (STED) microscopy has been used to address a wide range of neurobiological questions in optically well-accessible samples, such as cell culture or brain slices. However, the application of STED to deeply embedded structures in the brain of living animals remains technically challenging.

Aim: In previous work, we established chronic STED imaging in the hippocampus *in vivo* but the gain in spatial resolution was restricted to the lateral plane. In our study, we report on extending the gain in STED resolution into the optical axis to visualize dendritic spines in the hippocampus *in vivo*.

Approach: Our approach is based on a spatial light modulator to shape the focal STED light intensity in all three dimensions and a conically shaped window that is compatible with an objective that has a long working distance and a high numerical aperture. We corrected distortions of the laser wavefront to optimize the shape of the bottle beam of the STED laser.

Results: We show how the new window design improves the STED point spread function and the spatial resolution using nanobeads. We then demonstrate the beneficial effects for 3D-STED microscopy of dendritic spines, visualized with an unprecedented level of detail in the hippocampus of a living mouse.

Conclusions: We present a methodology to improve the axial resolution for STED microscopy in the deeply embedded hippocampus *in vivo*, facilitating longitudinal studies of neuroanatomical plasticity at the nanoscale in a wide range of (patho-)physiological contexts.

 [DOI: 10.1117/1.NPh.10.4.044402]

Keywords: 3D-stimulated emission depletion; *in vivo* imaging; hippocampal window; point spread function optimization; super-resolution microscopy.

Paper 23006SSR received Feb. 7, 2023; revised manuscript received Mar. 31, 2023; accepted for publication Apr. 14, 2023; published online May 17, 2023.

1 Introduction

The hippocampus is a deeply embedded brain region, which plays a critical role in encoding new memories. In the hippocampus, as elsewhere in the mammalian brain, pyramidal neurons receive most of their excitatory synaptic input at dendritic spines, which are small protrusions in the postsynaptic membrane that house the postsynaptic signaling machinery including glutamate receptors. Structural and functional plasticity of dendritic spines is a fundamental neurobiological process that underlies all higher brain functions, such as memory, thought, and action,[1,2] whereas spine dysfunction is closely associated with neuropsychiatric and neurodegenerative disorders, such as autism and Alzheimer's disease.[3]

*Address all correspondence to Stéphane Bancelin, stephane.bancelin@cnrs.fr; U. Valentin Nägerl, valentin.nagerl@u-bordeaux.fr

Two-photon fluorescence microscopy provides high depth penetration and optical sectioning in turbid media,[4] making it the standard technique for imaging in acute brain slices[5,6] and intact brains.[7–9] Over the last 20 years, it has transformed our understanding of the structure and function of dendritic spines in mouse brain.[10]

Until now, most *in vivo* studies of dendritic spines have been limited to superficial layers of the cortex (somatosensory, motor, visual cortex),[11–13] mainly because of the challenge to optically reach more deeply embedded structures. For example, the hippocampus is more than 1 mm below the cortical surface of the mouse brain. Only a few groups have ventured into imaging hippocampal spines *in vivo*, relying either on surgical resection of the overlaying cortex[14,15] or a microendoscope with a gradient-index lens.[16]

However, 2-photon microscopy is a diffraction-limited approach, which offers at best a spatial resolution of around 350 nm laterally and 1 μm axially, falling substantially short of visualizing several key neuro-anatomical structures and spaces, including spine necks, axon shafts, astroglial processes, and synaptic clefts, whose sizes can range well below 100 nm. As a consequence, there is a great need to develop and improve super-resolution imaging techniques that could be applied to deeply embedded regions, such as the hippocampus. Among the various super-resolution methods, stimulated emission depletion (STED) microscopy[17] is currently the only one that has been successfully applied *in vivo*,[18–22] notably in the hippocampus.[23]

STED microscopy is based on laser-scanning microscopy (such as confocal or 2-photon microscopy), where a Gaussian laser beam is focused to a small spot generating the fluorescence signal used to construct the image. In addition, there is a second laser (the STED laser), which exerts the opposite effect, namely it de-excites molecules by the process of stimulated emission. By spatially shaping the focal STED intensity like a donut,[17] it is possible to suppress the spontaneous fluorescence in the peripheral region of the focal spot, narrowing down the effective point spread function (PSF) in the *XY* plane by up to an order of magnitude.[24] By delivering STED light also above and below the focal region (a profile referred to as "bottle beam"), it becomes possible to constrict the fluorescence along the optical axis as well.[25,26]

In STED microscopy, spatial resolution and signal-to-noise ratio (SNR) of the images crucially depend on the quality of the PSF of the STED beam.[27] Yet, maintaining a high-quality PSF inside light scattering brain tissue poses several challenges. This problem is particularly evident in the context of *in vivo* imaging, where a variety of biological and mechanical constrains stand in the way of ensuring sufficiently good optical conditions for STED imaging. Notably, long working distance water immersion objectives typically used for *in vivo* imaging are not optimal for STED. Indeed, the effective STED resolution scales nonlinearly with the numerical aperture (NA) of the objective lens.[28] Hence, oil or glycerin immersion lens are typically preferred, offering high NA up to 1.49 at the expense of limited working distance. In addition to gain optical access to the brain, most studies rely on implanting a cranial window[29] in which a small part of the skull is replaced by a coverslip. Consequently, the mechanical stability of the cranial window attachment as well as of the brain itself are crucial, since any vibrations stemming from muscle contractions, pulmonary breathing, and blood pulsations can produce motion artefacts and diminish image quality.

The bottle beam PSF used for the axial gain in resolution is particularly sensitive to optical aberrations and misconfigurations in the beam path. Notably, the modified cranial window used to image the hippocampus[15,24] reduced the effective NA, which prevented the use of a bottle beam profile for 3D-STED in our previous study based on a cylindrically shaped "hippocampal window."[23] In this paper, we propose the use of a conically shaped window, specifically designed to maintain the bottle beam profile, while minimizing the size of the surgical resection of the overlaying cortex.

2 Material and Methods

2.1 *2-Photon STED Microscope*

Imaging was performed using a custom-built upright laser-scanning fluorescence microscope, as previously described.[23,27,30] In brief, the 2-photon excitation beam (100 fs, 80 MHz, 900 nm) was provided by a femtosecond titanium:sapphire laser (Tsunami, Spectra-Physics), pumped by

a high power continous wave diode pumped solid state laser (Millennia EV 15, Spectra-Physics), sent through a Pockels cell (302 RM, Conoptics) to control the excitation power. The STED beam (592 nm, 700 ps, 80 MHz) was provided by another pulsed laser (Katana 06 HP, NKT Photonics) whose power was adjustable using a half-wave plate and a polarization beam splitter. Both lasers were synchronized and temporally overlapped using commercial electronics ("Lock-to-clock," Model 3930 and 3931, Spectra-Physics).

The STED beam was passed via a spatial light modulator (SLM) (3D module, Abberior Instruments) to modulate the wavefront in a way that a donut or bottle beam intensity distribution of the STED light is achieved in the focal plane. Half and quarter-wave plates ($\lambda/2$ and $\lambda/4$) were used to produce a left-handed circular polarization at the entrance pupil of the objective. Both 2-photon excitation and STED beams were combined using a long-pass dichroic mirror (DCSPXRUV—T700, AHF). Appropriate lens combinations were used to conjugate the SLM on a telecentric scanner (Yanus IV, TILL Photonics), which was then imaged on the back focal plane of the objective (CFI Apo NIR 60× W, NA 1.0, Nikon) mounted on a z-focusing piezo actuator (Pifoc 725.2CD, Physik Instrumente). This objective provided a working distance of 2.8 mm, sufficient to bridge the physical distance between the surface of the brain and the deeply embedded hippocampus, while still offering a relatively high NA conducive for high-resolution imaging.

The epifluorescence signal was descanned, separated from the incident beams using a long-pass dichroic mirror (580 DCXRUV, AHF) and detected by an avalanche photodiode (SPCM-AQRH-14-FC, Excelitas) with appropriate notch (594S-25, Semrock) and bandpass filters (680SP-25, 520-50, Semrock) along the emission path. Signal detection and hardware control were performed via a data acquisition card (PCIe-6259, National Instruments) and the Imspector software (Abberior Instruments).

To visualize and prealign the donut or bottle PSFs, a pellicle beam splitter (BP145B1, Thorlabs) was flipped into the beam path to detect the signal reflected by gold beads (150 nm Gold nanospheres, Sigma Aldrich) using a photomultiplier tube (MD963, Excelitas). In the following, 2D-STED, z-STED, and 3D-STED will refer to images acquired using a pure donut, a pure bottle beam or a combination of the two beams, respectively. Optical resolution was assessed by imaging fluorescent beads (yellow–green fluorescent beads, 40 or 170 nm in diameter, Invitrogen) immobilized on glass slides.

2.2 *Animal Experimentation*

We used adult female and male transgenic mice (Thy1 $-$ $H^{tg/+}$, 3 to 12 months old) where a subset of hippocampal neurons was fluorescently labeled with YFP.[31] Heterozygous mice were used with sparse yet robust cytosolic labeling well adapted for high contrast superresolution imaging. The mice were group-housed by gender at a 12/12 h light/dark cycles. All procedures were in accordance with the Directive 2010/63/EU of the European Parliament and approved by the Ethics Committee of Bordeaux under agreement number 8899.

2.3 *Hippocampal Window Implantation*

Chronic hippocampal windows were implanted as described previously[14,15,23,32] to provide optical access to the *Stratum oriens* and *Stratum pyramidale* of the CA1 region of the hippocampus. In brief, mice were anesthetized with isoflurane (2%) and received intraperitoneal injections of analgesic (buprenorphine, 0.05 mg/kg) and anti-inflammatory drugs (dexamethasone, 0.2 mg/kg) to minimize brain swelling during the surgical procedure. The mouse scalp was shaved in the surgical region, and the mouse was placed into a stereotaxic frame with a heating pad. Lidocaine was locally applied prior to the removal of the skin and periosteum above the skull. A 3-mm-diameter craniotomy was performed above the right or left hemisphere using a dental drill (anteroposterior −2.2 mm; mediolateral +1.8 mm). The dura was carefully removed using fine forceps before aspiring the somatosensory cortex above the hippocampus using a vacuum pump connect to 29 G blunt needle. The overlying alveus was carefully peeled away to expose the surface of the hippocampus. A custom-made metal tube sealed with a coverslip (#1) on the bottom side (both 3 mm in diameter) was inserted into the craniotomy and tightly

fixed to the skull with acrylic glue. Since our objective lens does not have a correction collar, this coverslip offers a good compromise between the thick #1.5 H coverslip, necessitating strong correction of spherical aberration with the SLM, and the thin #0 coverslip, which can be too fragile for cranial window implantation. Once in place, the hippocampal window was fixed using ultraviolet light curable dental cement.

2.4 In Vivo *Imaging*

After the surgery, mice received analgesics for 2 days (buprenorphine, 0.05 mg/kg, intraperitoneal injection) and be allowed to recover for at least 4 weeks before starting imaging sessions. During these sessions, mice were anesthetized under 4% isoflurane prior to be transferred to a custom-made 3D printed tiltable frame, based on ear bars and nose fixations, incorporating a mask delivering 1.5% to 2% isoflurane at 0.2 L/ min O_2. The eyes were protected with ointment (bepanthen) and body temperature was maintained using a heating pad with anal probe. Imaging of CA1 pyramidal neurons was performed at 10 to 30 μm depth to avoid scare tissue at the surface while limiting optical aberrations steaming from the sample.[19,27] Typical image size was 20×20 μm^2 in XY with a pixel size of 20 nm, z stacks typically extended over 4 μm with a z-step size of 100 nm. Images were acquired with a 20 μs pixel dwell time, whereas excitation and STED laser powers were in the range of 10 to 20 mW after the objective lens. With these acquisition settings, no signs of phototoxicity were visible.

3 PSF Computation

The PSF of the STED beam was calculated using vectorial diffraction theory by Richard and Wolf,[33,34] which makes it possible to calculate the electromagnetic field ($\mathbf{E}$) in an arbitrary point P close to the focal region of a high NA (NA $\geq$ 0.7) objective, based on the Debye integral[35,36]

$$\mathbf{E}(P) = -\frac{ikf}{2\pi}\iint_{\Omega}\frac{A(\mathbf{s})}{s_z}e^{ik\mathbf{s}.\mathbf{R}}\mathrm{d}s_x\,\mathrm{d}s_y, \tag{1}$$

where k is the wavenumber, f the objective focal length, Ω the solid angle of the exit pupil from the focal spot, $\mathbf{s}$ the unit vector along each ray from the objective pupil to the focal volume, $A(\mathbf{s})$ the complex amplitude of the incident laser beam after the objective and $\mathbf{R}$ the position vector of point $\mathrm{P}(x, y, z)$.

Considering the geometry depicted in Fig. 1(a), the diffraction integral can be expressed, in spherical coordinates, as[36,37]

$$\mathbf{E}(x,y,z) = C\int_0^{\alpha}\int_0^{2\pi}B(\theta,\varphi)\mathbf{P}(\theta,\varphi)e^{i(M(\theta,\varphi)+\Phi(\theta,\varphi))}e^{k_0 n_1(x\cos\varphi\sin\theta_1+y\sin\varphi\sin\theta_1)}$$
$$e^{ik_0 n_3 z\cos\theta_3}e^{ik_0(n_3 d\cos\theta_3 - n_1(t+d)\cos\theta_1)}\sin\theta\mathrm{d}\theta\,\mathrm{d}\varphi, \tag{2}$$

where C is a constant, k_0 the wavenumber in the vacuum, $\alpha = \arcsin(\mathrm{NA}/n)$ the marginal ray angle, n_l and θ_l the refractive indices and incident angle in the (1) immersion media, (2) the coverslip, and (3) the sample, respectively, d the depth in the sample, t the thickness of the coverslip, $B(\theta,\varphi)$ the amplitude profile of the incident beam, $\mathbf{P}$ the polarization state of the electromagnetic field in the focal region, $M(\rho,\varphi)$ the phase profile of the input beam, corresponding in our case to the phase mask used to shape the STED beam, and $\Phi(\theta,\varphi)$ the wavefront distortion with respect to the Gaussian reference sphere, which describes the optical aberrations in the system. Note that the second and third exponential terms correspond to the aberrations induced along the optical path through the coverslip and the biological sample.[37]

Although passing through an aplanatic objective, the incident plane wave transforms into a spherical wave converging to the focal point. Therefore, assuming a Gaussian profile of the input beam, the amplitude distribution after the objective can be expressed as

$$B(\theta,\varphi) = B_0 e^{-\frac{\rho^2}{w^2}}\sqrt{\cos\theta}, \tag{3}$$

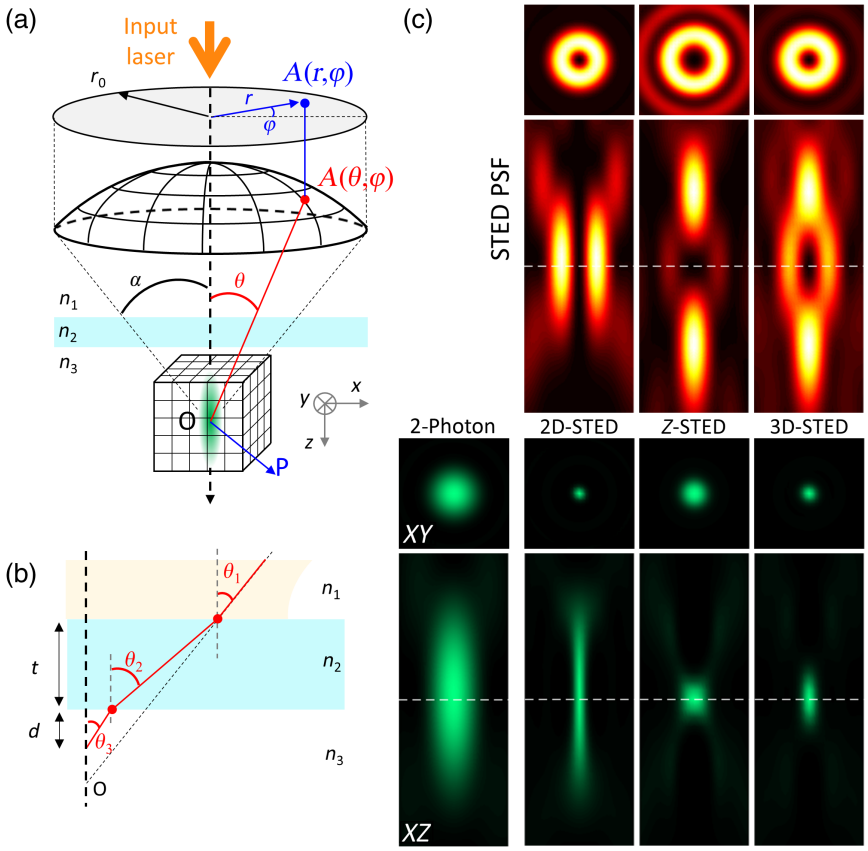

Fig. 1 (a) Schematic representation of the propagation of a light wave focused by a high NA objective used to calculate the PSF in the vicinity of the focus. (b) Refraction angles within the coverslip. Due to the refractive index mismatch, each interface decreases transmission and induces spherical aberrations. (c) STED beam (top, fire LUT) and effective fluorescence (bottom, green LUT) PSFs in XY plane (square panels, image size 1×1 μm^2) and XZ plane (rectangle panels, image size 1×2.5 μm^2), for the different configurations used in this paper (2-photon only, 2D-STED, z-STED and 3D-STED). The dashed line indicates the focus position.

where B_0 is a constant, w the beam waist, $\rho = f \sin\theta$ the cylindrical coordinate on the exit pupil of the objective lens, and $\sqrt{\cos\theta}$ the apodization term ensuring energy conservation while the beam pass through the objective. In addition, the objective transforms the input left-handed circular polarization $\mathbf{P}_0(\theta,\varphi) = (1, i, 0)$, classically used in STED microscopy, through tight focalization, which can be described as

$$\mathbf{P}(\theta,\varphi) = \mathcal{R}_\varphi^{-1}[\mathcal{P}^{(3)}]^{-1}\mathcal{I}^{(2)}\mathcal{P}^{(1)}\mathcal{L}_{\theta_1}\mathcal{R}_\varphi\mathbf{P}_0, \tag{4}$$

where $\mathcal{R}_\varphi$ is the rotation matrix around z axis, $\mathcal{L}_\theta$ describe the change in electric field as it passes through the objective, $\mathcal{P}^{(i)}$ corresponds to the coordinate system rotation in medium l, and $\mathcal{I}^{(2)}$ is the matrix describing the effect of the coverslip medium, considered as a stratified medium of two interfaces

$$\mathcal{R}_\varphi = \begin{pmatrix} \cos\varphi & \sin\varphi & 0 \\ -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad \mathcal{L}_\theta = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix}, \tag{5}$$

$$\mathcal{P}^{(l)} = \begin{pmatrix} \cos\theta_l & 0 & -\sin\theta_l \\ 0 & 1 & 0 \\ \sin\theta_l & 0 & \cos\theta_l \end{pmatrix}, \quad \mathcal{I}^{(2)} = \begin{pmatrix} T_p^{(2)} & 0 & 0 \\ 0 & T_s^{(2)} & 0 \\ 0 & 0 & T_p^{(2)} \end{pmatrix}, \tag{6}$$

where $T_{s,p}^{(2)}$ is the transmission coefficient in the coverslip (see[37] for complete derivation).

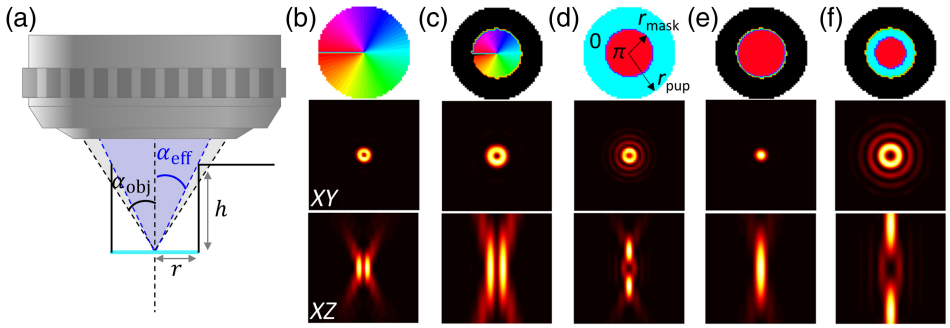

Fig. 2 (a) Schematic of the hippocampal cranial window and its effect on the focused beam, notably the clipping of the outer optic rays. Numerical simulation of the STED beam XY (middle panels) and XZ (bottom panels) profiles calculated for the different phase masks (top panels) used in STED microscopy: Donut (b) effective donut in presence of hippocampal window (c) bottle beam (d) effective bottle beam in presence of hippocampal window and (e) same with adjusted phase mask radius ($r_{mask} = 0.33 r_{pup}$) enabling the retrieval of the (f) bottle profile. Image size: $5 \times 5\ \mu m^2$.

Finally, in the case of left-handed circular polarization

$$\mathbf{P}(\theta,\varphi) = \begin{pmatrix} T_p^{(2)} \cos\theta_3 \cos^2\varphi + T_s^{(2)} \sin^2\varphi \\ T_p^{(2)} \cos\theta_3 \cos\varphi \sin\varphi - T_s^{(2)} \cos\varphi \sin\varphi \\ -T_p^{(2)} \sin\theta_3 \cos\varphi \end{pmatrix} + i \begin{pmatrix} T_p^{(2)} \cos\theta_3 \sin\varphi \cos\varphi - T_s^{(2)} \sin\varphi \cos\varphi \\ T_p^{(2)} \cos\theta_3 \sin^2\varphi + T_s^{(2)} \cos^2\varphi \\ -T_p^{(2)} \sin\theta_3 \cos\varphi \end{pmatrix}. \tag{7}$$

In the context of STED microscopy, the PSF of the STED beam is spatially shaped [Fig. 1(c)—top profiles] using specific phase masks $M(\theta,\varphi)$, that can be expressed as

$$M(\theta,\varphi) = \begin{cases} 0 & \text{No phase mask} - \text{Gaussian beam,} \\ \varphi & \text{Vortex phase mask} - \text{Donut beam,} \\ \begin{cases} \pi & \text{for } \theta \le \theta_M \\ 0 & \text{for } \theta_M \le \theta \le \alpha \end{cases} & \text{Ring phase mask} - \text{Bottle beam,} \end{cases} \tag{8}$$

where $\theta_M = a \sin(\frac{r_{mask}}{r_{pupil}} \sin\alpha)$ is the angle between the optical axis and the ray passing through the edge of the π-phase ring of the phase mask of radius r_{mask} on the objective output pupil of radius r_{pupil} [Fig. 2(d)—top panel].

The focal intensity can be calculated as the squared modulus of the electric field

$$I = |\mathbf{E}|^2 = |E_x|^2 + |E_y|^2 + |E_z|^2. \tag{9}$$

Finally, the effective PSF [Fig. 1(c)—bottom profiles] is calculated as[38]

$$I_{eff}(x,y,z) = I_{2P} e^{-\frac{\ln 2 I_{STED}}{I_{sat}}}, \tag{10}$$

where I_{exc} and I_{STED} are the excitation and STED beams, respectively, and I_{sat} is the saturation intensity, which describes the de-excitation rate of the molecules by the STED beam.

4 Results and Discussion

4.1 *Impact of the Cranial Window*

The chronic hippocampal window used here was originally developed to image pyramidal neurons in the hippocampus by 2-photon microscopy[14,15,32] using a 0.8 NA objective. In this case, the specific geometry of the window, a metal cylinder sealed with a coverslip, limited the angle of the marginal rays transmitted through the window [Fig. 2(a)]. Indeed, to reach the hippocampus surface, the implanted cylinder and holder has a height (h) of 2.23 mm and an inner diameter (r) of 2.6 mm, which corresponds to a maximum opening angle of 30.2 deg and hence an effective NA of 0.67. Although such an NA can be acceptable for 2-photon imaging, albeit at the expense of reduced spatial resolution (notably axial resolution because of extended excitation PSF), it is prohibitive for STED microscopy. Indeed, beyond the NA limitation, the elimination of the marginal rays by the window design has a dramatic effect on the STED-PSF, rendering it useless, even counterproductive, for improving the STED axial resolution.

To further investigate this effect on the STED PSF and the resulting effective fluorescence PSF, we modified the calculation to consider the effect of the cranial window. We introduced an additional amplitude mask $T(\rho,\varphi)$ in Eq. (2), which models the additional aperture stop at the entrance of the window leading to the clipping of the outer rays after the objectives. The diffraction integral can be expressed as

$$\mathbf{E}(x,y,z) = C\int_0^{\alpha}\int_0^{2\pi} T(\theta,\varphi)B(\theta,\varphi)\mathbf{P}(\theta,\varphi)e^{i(M(\theta,\varphi)+\Phi(\theta,\varphi))}e^{k_0 n_1(x\cos\varphi\sin\theta_1+y\sin\varphi\sin\theta_1)}$$
$$e^{ik_0 n_3 z\cos\theta_3}e^{ik_0(n_3 d\cos\theta_3-n_1(t+d)\cos\theta_1)}\sin\theta\,\mathrm{d}\theta\,\mathrm{d}\varphi, \tag{11}$$

with

$$T(\theta) = \begin{cases} 1 & \text{for } \theta \le \theta_c \quad \text{transmitted rays,} \\ 0 & \text{for } \theta_c \le \theta \le \alpha \quad \text{blocked region,} \end{cases} \tag{12}$$

where θ_c is the angle between the optical axis and the marginal ray of the cranial window on the output pupil of the objective.

Figures 2(b)–2(f) displays the results of these simulations. In Figs. 2(b) and 2(c), looking at the donut beam, the effect of the reduced NA is easily visible through a clear elongation of the profile. Yet, the presence of the cranial window does not change the geometry of the phase mask (it remains a vortex) and hence the symmetry of the PSF. Therefore, even if suboptimal, this configuration still permits superresolution imaging. In Fig. 2(d), in contrast, the bottle beam is dramatically degraded in the presence of the cranial window. In Fig. 2(e), in the case of the ring phase mask, the outer rays (with 0 phase—blue area in the phase mask) are not passing through the hippocampal window, whereas the inner rays (with π phase—red area in the phase mask) remain unaffected. This prevents destructive interference to happen in the focus and hence the formation of the central zero that is required for suppressing the fluorescence in the periphery of the fluorescence PSF while leaving the central region unaffected. In Fig. 2(e), note that by simply adjusting the ring radius on the phase mask, one could effectively retrieve a correct bottle beam profile. Yet, this does not solve the issue of the elongated shape [see panel (d) and (f)] due to the limited effective NA, which results in decreased axial resolution.

4.2 *New Cranial Window Design and Experimental Validation*

To retrieve a usable bottle beam and to achieve a substantial STED gain in spatial resolution in all three dimensions, we designed a new cranial window, or hippocampal porthole. Although increasing the radius of the conical window is possible [Fig. 3(a)], retrieving the full NA would require implanting a cylinder with a diameter of 5.1 mm into in the mouse brain, which is prohibitive in terms of the amount of cortical volume that would have to be surgically removed (about 45 mm^3 of cortex). Instead, we chose to modify the geometry of the window. Using a conical shape [Fig. 3(b), see Supplementary Fig. S1 (3D drawing)] makes it possible to benefit

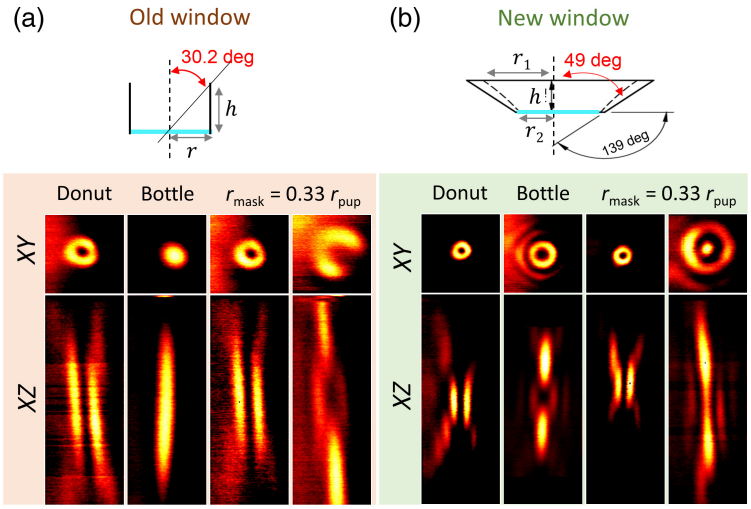

Fig. 3 Schematic of the (a) old and (b) new hippocampal windows (top panels). The conical shape makes it possible to use the full NA of the objective while minimizing the cortical volume that needs to be removed. STED beam PSFs (bottom panels) in XY (2×2 μm^2) and XZ (2×8 μm^2) planes experimentally measured, using gold nanoparticles attached to the coverslip, and imaged through the new (top panel) and old (bottom panel) cranial window designs, demonstrating the recovery of the appropriate bottle beam shape. In the right panels, the radius of the phase mask has been adjusted using the SLM.

from the full nominal NA of the objective, while minimizing the amount of tissue that needs to be resected to expose the hippocampus, reducing it to about 14 mm^3, which is less than one third.

We first validated this cranial window geometry *ex vivo*, by placing gold nanoparticles on a poly-L-lysine coated coverslip, the same that we had used in the cranial window for *in vivo* imaging, and imaged them either through the cylinder (old window) or conical (new window) porthole without implantation on the head of the animal. This illustrates experimentally the impact of the cranial window design on the PSF of the STED beam.

Figure 3 clearly illustrates the effect of the two different cranial window designs on the STED PSF. Beyond the reduction of the NA, the old cylindrical window seriously degrades the shape of the bottle beam, obliterating the central intensity minimum, which is a must for STED microscopy. By contrast, the new conical cranial window design permits the generation of improved donut and bottle beam shapes. With the new design, the NA is limited by the optical design of the objective, and not the geometry of the optical access.

4.3 In Vivo *3D-STED Imaging in the Hippocampus*

To visualize the gain in resolution in live conditions, fluorescence beads (diameter: 170 μm) were attached to the coverslip using poly-L-lysine prior to be grafted into the animal. Imaging the fluorescent beads through the old and new window designs makes it possible to quantify and compare the gain in resolution between them. Figure 4 displays images in XY and XZ direction of the fluorescent beads visualized through the cranial window. Note that the look-up table is adjusted between images for better visualization.

Table 1 reports the axial and lateral resolution obtained experimentally using the two different windows together with the theoretical resolution obtained from numerical simulations. In this table, the spatial resolution is estimated as the full-width at half maximum (FWHM) of the PSF. Beyond the resolution, we quantified the maximum number of counts on the detector as a measure of image brightness (and thus SNR), which is strongly affected by the window geometry. Comparing our simulations with the experimental results, we normalized the number of counts in the simulated image with the value obtained from 2-photon images acquired with the new cranial window (which is expected to yield the highest number of counts).

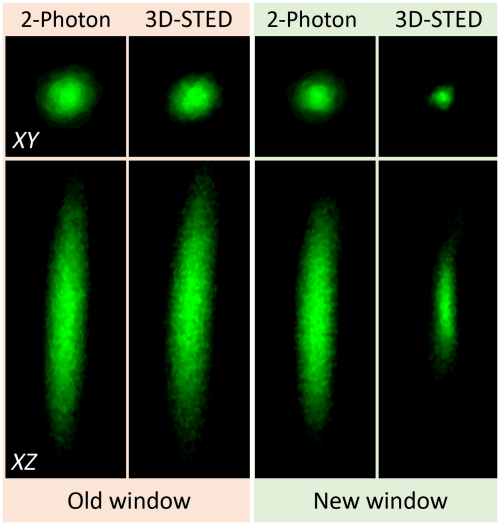

Fig. 4 Effective fluorescence PSF in 2-photon and 3D-STED obtained by imaging the fluorescent nanoparticles attached below the coverslip in the old cylindrical and new conical cranial window implanted above the hippocampus of an adult mice. *XY* image size: 1.5×1.5 μm^2 and *XZ* image size 1.5×4 μm^2.

Table 1 Spatial resolution (estimated as FWHM of the PSF) and maximum number of counts obtained in 2-photon and STED with the two different cranial window geometries. Mean ± SD from 10 fluorescent beads in 2 different samples prepared from the same batch.

		XY resolution (nm)		*Z* resolution (nm)		I_{max} (No. of counts)	
		2-photon	STED	2-photon	STED	2-photon	STED
Cylindrical	Simu	472	456	2831	2788	60	17
window	Exp	470 ± 10	430 ± 20	2800 ± 150	2500 ± 500	50 ± 20	20 ± 15
Conical	Simu	344	81	1244	285	158	138
window	Exp	350 ± 8	80 ± 10	1200 ± 100	310 ± 40	158 ± 8	110 ± 10

The 2-photon PSFs are slightly improved by the new cylindrical window, yielding a modest but clear improvement in spatial resolution. In contrast, the 3D-STED performance is greatly affected by the type of window design. With the old cylindrical window, the effective PSF is very similar to the 2-photon PSF but with a strongly reduced signal, as expected from the absence of a zero in the bottle beam profile, diminishing the excitation of molecules without yielding of a gain in spatial resolution. In contrast, the new conical window permits a significant constriction of the fluorescent spot, while largely preserving the signal level.

Having established this proof of principle, we validated this modified hippocampal window on biological samples by imaging fluorescently labeled neurons in living transgenic mice. Figure 5(a) shows a segment of dendrite in the *Stratum radiatum* of the CA1 region in the hippocampus of a living mouse. Notably, the fine morphological features, including the hallmark cup-like shapes of spine heads and the ultrathin neck regions, connecting the spine head with the dendrite, can be appreciated with unprecedentedly high image quality in an *in vivo* setting. Figures 5(b) and 5(c) show a volume rendering of the same segment of dendrite qualitatively illustrating the gain in resolution and anatomical fidelity that can be achieved by our improved approach.

Finally, we also quantified neck diameters of the dendritic spines using a semiautomatic software,[39] which was specifically designed for morphometric analysis of superresolution images of dendritic spines. The results are summarized in Table 2 and are consistent with the published literature based on electron microscopy or STED imaging in brain slices.[40–42]

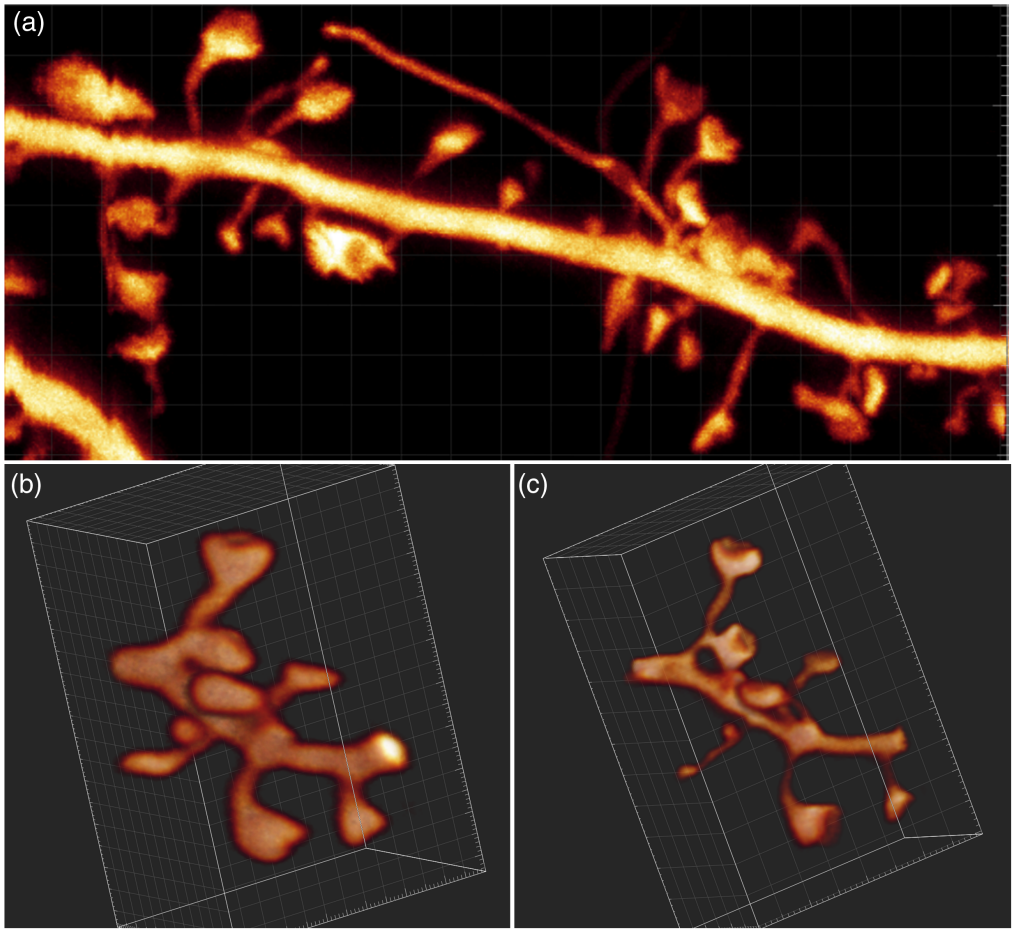

Fig. 5 (a) Image of a YFP-labeled segment dendrite in the *S. radiatum* of a Thy1 – $H^{tg/+}$ mouse, lying about 30 μm below the surgically created surface. Image size 5×11 μm^2. (b), (c) 3D rendering of the same segment of dendrite obtained with 2-photon and 3D-STED imaging, respectively. Image size $5 \times 3.2 \times 3.2$ μm^3. Panels (b) and (c) are still images from videos, 2-photon (Video 1) and STED (Video 2) (Video 1, MP4, 18 MB [URL: https://doi.org/10.1117/1.NPh.10.4.044402.s1]), (Video 2, MP4, 21 MB [URL: https://doi.org/10.1117/1.NPh.10.4.044402.s2]).

Table 2 Average spine morphological parameters measured by 2-photon and 3D-STED microscopy. Mean ± SD from 23 spines collected from 3 mice.

	Spine neck with (μm)			
	Lateral	Axial	Spine neck length (μm)	Spine head volume (μm^3)
2-photon	0.45 ± 0.05	1.5 ± 0.4	0.7 ± 0.3	0.4 ± 0.2
3D-STED	0.17 ± 0.03	0.39 ± 0.08	0.8 ± 0.3	0.1 ± 0.1

5 Conclusion

In this paper, we propose and validate a modified hippocampal window design, which makes full use of the NA of a long-working distance objective. Although this new design by itself already increases the spatial resolution and optical sectioning of 2-photon microscopy, it is essential for STED microscopy. Notably, it can preserve the bottle beam shape needed for 3D-STED microscopy. We illustrate the benefit of this new cranial window design by visualizing dendritic spines, greatly improving STED image quality, rendering it comparable to the state of the art in brain slice preparations. Combined with state-of-the-art adaptive optics approaches,[20,43]

this hippocampal window design will pave the way for 3D nanoscale imaging deep within the hippocampus of live mouse.

Our new approach improves the achievable spatial resolution for nanoscale imaging of neuroanatomical structures and compartments (e.g., the extracellular space between brain cells[44,45]), enabling longitudinal investigations into how their dynamics may underpin the ability of neurons and their networks to adapt themselves to ever-changing environmental conditions in health and disease.

Disclosures

The authors declare no conflict of interest.

Acknowledgments

This project received funding from the European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie action (Grant No. 794492), the Fonds AXA pour la Recherche to SB, the Doctoral School for Health and Life Sciences of the University of Bordeaux to JR, the European Research Council (ERC-SyG ENSEMBLE) (Grant No. 951294), Human Frontiers Science Program (Grant No. RGP0036/2020), ERA-NET NEURON (Grant No. ANR-17-NEU3-0005), the Féderation pour la recherche sur le cerveau (FRC), and Agence Nationale de la Recherche (Grant No. ANR-17-CE37-0011) to UVN. We thank the animal facility at IINS for their support.

Code, Data, and Availability

All datasets and codes are available from the corresponding authors upon reasonable request.

References

1. C. Sala and M. Segal, "Dendritic Spines: the locus of structural and functional plasticity," *Physiol. Rev.* **94**(1), 141–188 (2014).
2. R. Yuste and T. Bonhoeffer, "Genesis of dendritic spines: insights from ultrastructural and imaging studies," *Nat. Rev. Neurosci.* **5**(1), 24–34 (2004).
3. T. C. Südhof, "Neuroligins and neurexins link synaptic function to cognitive disease," *Nature* **455**(7215), 903–911 (2008).
4. K. Svoboda and R. Yasuda, "Principles of two-photon excitation microscopy and its applications to neuroscience," *Neuron* **50**(6), 823–839 (2006).
5. Z. Mainen et al., "Two-photon imaging in living brain slices," *Methods* **18**(2), 231–239 (1999).
6. G. C. Ellis-Davies, "Two-photon microscopy for chemical neuroscience," *ACS Chem. Neurosci.* **2**(4), 185–197 (2011).
7. J. N. Kerr and W. Denk, "Imaging *in vivo*: watching the brain in action," *Nat. Rev. Neurosci.* **9**(3), 195–205 (2008).
8. M. J. Levene et al., "*In vivo* multiphoton microscopy of deep brain tissue," *J. Neurophysiol.* **91**(4), 1908–1912 (2004).
9. A. L. A. Mascaro, L. Sacconi, and F. S. Pavone, "Multi-photon nanosurgery in live brain," *Front. Neuroenergetics* **2**, 21 (2010).
10. A. Holtmaat and K. Svoboda, "Experience-dependent structural synaptic plasticity in the mammalian brain," *Nat. Rev. Neurosci.* **10**(9), 647–658 (2009).
11. J. T. Trachtenberg et al., "Long-term *in vivo* imaging of experience-dependent synaptic plasticity in adult cortex," *Nature* **420**(6917), 788–794 (2002).
12. S. K. Kim, K. Eto, and J. Nabekura, "Synaptic structure and function in the mouse somatosensory cortex during chronic pain: *in vivo* two-photon imaging," *Neural Plast.* **2012**, 640259 (2012).

13. S. El-Boustani et al., "Locally coordinated synaptic plasticity of visual cortex neurons *in vivo*," *Science* **360**(6395), 1349–1354 (2018).
14. A. Mizrahi et al., "High-resolution *in vivo* imaging of hippocampal dendrites and spines," *J. Neurosci.* **24**(13), 3147–3151 (2004).
15. L. Gu et al., "Long-term *in vivo* imaging of dendritic spines in the hippocampus reveals structural plasticity," *J. Neurosci.* **34**(42), 13948–13953 (2014).
16. A. Attardo, J. E. Fitzgerald, and M. J. Schnitzer, "Impermanence of dendritic spines in live adult CA1 hippocampus," *Nature* **523**(7562), 592–596 (2015).
17. S. W. Hell, "Far-field optical nanoscopy," *Science* **316**(5828), 1153–1158 (2007).
18. S. Berning et al., "Nanoscopy in a living mouse brain," *Science* **335**(6068), 551 (2012).
19. H. Steffens, W. Wegner, and K. I. Willig, "*In vivo* STED microscopy: a roadmap to nanoscale imaging in the living mouse," *Methods* **174**, 42–48 (2020).
20. M. G. M. Velasco et al., "3D super-resolution deep-tissue imaging in living mice," *Optica* **8**(4), 442–450 (2021).
21. K. I. Willig, "*In vivo* super-resolution of the brain–how to visualize the hidden nanoplasticity," *iScience* **25**, 104961 (2022).
22. M. Fuhrmann et al., "Super-resolution microscopy opens new doors to life at the nanoscale," *J. Neurosci.* **42**(45), 8488–8497 (2022).
23. T. Pfeiffer et al., "Chronic 2P-STED imaging reveals high turnover of dendritic spines in the hippocampus *in vivo*," *eLife* **7**, e34700 (2018).
24. C. V. Middendorff et al., "Isotropic 3D nanoscopy based on single emitter switching," *Opt. Express* **16**(25), 20774–20788 (2008).
25. D. Wildanger et al., "A compact STED microscope providing 3D nanoscale resolution," *J. Microsc.* **236**(1), 35–43 (2009).
26. M. O. Lenz et al., "3-D stimulated emission depletion microscopy with programmable aberration correction," *J. Biophotonics* **7**(1–2), 29–36 (2014).
27. S. Bancelin et al., "Aberration correction in stimulated emission depletion microscopy to increase imaging depth in living brain tissue," *Neurophotonics* **8**(3), 035001 (2021).
28. J. Heine et al., "Three dimensional live-cell sted microscopy at increased depth using a water immersion objective," *Rev. Sci. Instrum.* **89**(5), 053701 (2018).
29. S. W. Cramer et al., "Through the looking glass: a review of cranial window technology for optical access to the brain," *J. Neurosci. Methods* **354**, 109100 (2021).
30. P. Bethge et al., "Two-photon excitation STED microscopy in two colors in acute brain slices," *Biophys. J.* **104**(4), 778–785 (2013).
31. G. Feng et al., "Imaging neuronal subsets in transgenic mice expressing multiple spectral variants of GFP," *Neuron* **28**(1), 41–51 (2000).
32. D. A. Dombeck et al., "Functional imaging of hippocampal place cells at cellular resolution during virtual navigation," *Nat. Neurosci.* **13**(11), 1433–1440 (2010).
33. E. Wolf, "Electromagnetic diffraction in optical systems–I. An integral representation of the image field," *Proc. R. Soc. Lond. Ser. A. Math. Phys. Sci.* **253**(1274), 349–357 (1959).
34. B. Richards and E. Wolf, "Electromagnetic diffraction in optical systems-II. Structure of the image field in an aplanatic system," *Proc. R. Soc. Lond. Ser. A. Math. Phys. Sci.* **253**(1274), 358–379 (1959).
35. M. Gu, *Advanced Optical imaging Theory*, Vol. 75, Springer Science & Business Media (2000).
36. P. Török et al., "Electromagnetic diffraction of light focused through a planar interface between materials of mismatched refractive indices: an integral representation," *JOSA A* **12**(2), 325–332 (1995).
37. P. Török and P. Varga, "Electromagnetic diffraction of light focused through a stratified medium," *Appl. Opt.* **36**(11), 2305–2312 (1997).
38. K. Willig et al., "STED microscopy resolves nanoparticle assemblies," *New J. Phys.* **8**(6), 106 (2006).
39. F. Levet et al., "SpineJ: a software tool for quantitative analysis of nanoscale spine morphology," *Methods* **174**, 49–55 (2020).

40. K. Harris and J. Stevens, "Dendritic spines of CA 1 pyramidal cells in the rat hippocampus: serial electron microscopy with reference to their biophysical characteristics," *J. Neurosci.* **9**(8), 2982–2997 (2018).
41. J. I. Arellano et al., "Ultrastructure of dendritic spines: correlation between synaptic and spine morphologies," *Front. Neurosci.* **1**(1), 131–143 (2007).
42. J. Tønnesen et al., "Spine neck plasticity regulates compartmentalization of synapses," *Nat. Neurosci.* **17**(5), 678–685 (2014).
43. T. J. Gould et al., "Adaptive optics enables 3D STED microscopy in aberrating specimens," *Opt. Express* **20**(19), 20998–21009 (2012).
44. J. Tønnesen, V. K. Inavalli, and U. V. Nägerl, "Super-resolution imaging of the extracellular space in living brain tissue," *Cell* **172**(5), 1108–1121.e15 (2018).
45. S. Hrabetova et al., "Unveiling the extracellular space of the brain: from super-resolved microstructure to *in vivo* function," *J. Neurosci.* **38**(44), 9355–9363 (2018).

U. Valentin Nägerl is a professor of neuroscience and bioimaging at the University of Bordeaux, where he leads a research group and is in charge of the masters program in biophotonics and neurotechnology. He studied physics and pre-clinical medicine in Göttingen, got his PhD in neuroscience from UCLA in 2000, and worked as a postdoc at the Max Planck Institute of Neurobiology. His research focuses on the development and application of super-resolution imaging techniques to study the structure and function of the brain's microarchitecture including its extracellular spaces.

Biographies of the other authors are not available.

HardwareX 13 (2023) e00400

Contents lists available at ScienceDirect

HardwareX

journal homepage: www.elsevier.com/locate/ohx

An open-source microscopy framework for simultaneous control of image acquisition, reconstruction, and analysis

Xavier Casas Moreno [a,*], Mariline Mendes Silva [a], Johannes Roos [b], Francesca Pennacchietti [a], Nils Norlin [c], Ilaria Testa [a]

[a] Science for Life Laboratory, Department of Applied Physics, KTH Royal Institute of Technology, 171 65 Stockholm Sweden
[b] Interdisciplinary Institute for Neuroscience, CNRS UMR 5297, 33000 Bordeaux, France
[c] Department of Experimental Medical Science, Lund University Bioimaging Centre (LBIC), 221 00 Lund University, Sweden

ARTICLE INFO

Article history:
Received 31 October 2022
Received in revised form 26 January 2023
Accepted 2 February 2023

Keywords:
RESOLFT
Automation
Software

ABSTRACT

We present a computational framework to simultaneously perform image acquisition, reconstruction, and analysis in the context of open-source microscopy automation. The setup features multiple computer units intersecting software with hardware devices and achieves automation using python scripts. In practice, script files are executed in the acquisition computer and can perform any experiment by modifying the state of the hardware devices and accessing experimental data. The presented framework achieves concurrency by using multiple instances of ImSwitch and napari working simultaneously. ImSwitch is a flexible and modular open-source software package for microscope control, and napari is a multidimensional image viewer for scientific image analysis.

The presented framework implements a system based on file watching, where multiple units monitor a filesystem that acts as the synchronization primitive. The proposed solution is valid for any microscope setup, supporting various biological applications. The only necessary element is a shared filesystem, common in any standard laboratory, even in resource-constrained settings. The file watcher functionality in Python can be easily integrated into other python-based software.

We demonstrate the proposed solution by performing tiling experiments using the molecular nanoscale live imaging with sectioning ability (MoNaLISA) microscope, a high-throughput super-resolution microscope based on reversible saturable optical fluorescence transitions (RESOLFT).

* Corresponding author.
E-mail address: xavier.casas@outlook.com (X. Casas Moreno).
@@xaviercasasm (X. Casas Moreno)

https://doi.org/10.1016/j.ohx.2023.e00400
2468-0672/

Specifications table

Hardware name	Sync-Scope
Subject area	Biological sciences
Hardware type	Imaging tools
Closest commercial analog	MetaMorph, Molecular Devices
Open source license	GNU General Public License (GPL)
Cost of hardware	2.000–4.000 EUR
Source file repository	https://zenodo.org/record/7561142

Hardware in context

Microscopy is a powerful tool for cell biology. Still, it becomes costly and time-consuming when pushed towards high throughput, such as when imaging extended areas for a prolonged time. Therefore, microscopy automation is an increasing area of interest in biological imaging [1] and high-throughput screening [2,3] applications. However, automation imposes multiple challenges from the hardware and software integration side, particularly regarding computational efficiency and workload distribution. Most solutions often rely on commercial software and hardware, making it difficult to adapt to different microscopy modalities and extend support to new applications.

Open-source software and hardware solutions for microscopy automation are relatively new, and most either rely on commercial microscopes [4,5] or are tailored to a specific technique [6–10]. Different general solutions have been proposed, such as Pycromanager [11], AutoScanJ [12], and MicroMator [13]. They interface with μManager [14], a well-established open-source software package with extensive driver support, distributed as an ImageJ [15] plugin. However, μManager presents limitations as the complexity of the microscopy modalities increases. Some examples are controlling multiple cameras independently, synchronizing multiple hardware devices, performing simultaneous image acquisition and reconstruction, or controlling specialized devices such as spatial light modulators or point detectors. Python-based software alternatives have been developed [16–19], making debugging and contributing more convenient in contrast to Java.

Sample-adaptive automation methods for image acquisition have been recently proposed [20,21], in which experiments and image analysis are combined to achieve fast, directed imaging by adapting to events in the sample of interest. Furthermore, microscopy automation in high-throughput imaging has been implemented using a custom-made low-cost microscope [22]. Imaging extended sample areas by tiling has been previously achieved through microscope automation in light-sheet [23,24] and STED [25] microscopy.

Several microscopy modalities require a reconstruction algorithm to turn the acquired data into final images. Techniques such as single-molecule localization microscopy (SMLM) [26–28], parallelized RESOLFT [29–32], structured illumination microscopy (SIM) [33,34], super-resolution optical fluctuation imaging (SOFI) [35], universal live-cell super-resolution microscopy (SRRF) [36], among others, require reconstructions algorithms to output a high-resolution image from the experimentally-acquired series of raw images. Furthermore, light field microscopy (LFM) [37], Fourier ptychography microscopy (FPM) [38], and lens-free on-chip microscopy [39] require computational algorithms to generate microscopy images by gathering information from different sources, such as the angle of the illumination light. However, the reconstruction procedure is often performed sequentially after all image data is acquired. This can be a limiting factor if the users need a rapid answer to adapt imaging schemes, optimize sample preparation or perform time-consuming experiments. Multiple efforts have been placed to accelerate the reconstruction process using a Graphics Processing Unit (GPU) [40–43]. Nevertheless, these techniques would greatly benefit from distributing computational resources and scheduling the experimental tasks in real time.

Image analysis of data in real-time is a promising direction, with solutions like ImJoy [44], BioImageIT [45], and cell profiler [46]. They implement several algorithms and plugins, which can, in principle, be applied to data acquired directly from the microscope. However, combining these pipelines with the entire experimental acquisition scheme, for example, using their output as active feedback on the acquisition, reminds a challenge. Therefore, a general open-source framework that integrates image acquisition, reconstruction, visualization, and data analysis and flexibly adapts to different microscopy modalities is still missing.

Here we present a framework to simultaneously perform image acquisition, reconstruction, and analysis using ImSwitch [19] and napari [47]. On the one hand, ImSwitch is an open-source software solution for controlling microscopes, adaptable to different levels of complexity. It includes image acquisition and reconstruction modules, where the experimental images can be visualized in real-time. On the other hand, napari is a community-driven image viewer with multiple plugins for image analysis and is widely used in the microscopy field. Our solution is based on the simple concept of file watching, where different units monitor a filesystem searching for new files to process and add the arriving items into a queue.

The solution doesn't require implementing any client–server-based approach, which relieves the burden of using a centralized server in terms of complexity, security, and dependency on other software packages. We provide a scripting engine in python that enables microscopy automation of any experiment designed by the user. The experiments are orchestrated remotely by creating and editing the scripts and adding them to the acquisition unit queue. However, the list of microscope commands is executed directly in the acquisition unit instead of running independent hardware orders through function calls over a network, which would limit the experiment's time resolution. Security is not a concern since the computer units are assumed to be in the same laboratory or institute, or remotely shared with specific and known users. The proposed framework can even be implemented without an internet connection, either by using multiple computers or running all the instances on the same machine, just by having a local or remote filesystem.

We provide the design files containing the main functionality of the file watcher as well as its integration with napari and ImSwitch. Furthermore, it can be easily implemented in other python-based software [16–18] both for acquisition and reconstruction. For example, the user can develop their own python scripts for image reconstruction and add them to the workflow by including the file watcher model.

The imaging data and metadata are fetched from ImSwitch and saved into either Zarr or HDF5 files, and the reconstructed images are then saved in both OME-Zarr and Tiff. OME-Zarr is an implementation of the Zarr format using the Open Microscopy Environment (OME) specifications [48]. Zarr can store chunks of data in a directory tree, which is highly beneficial for access times in a shared filesystem. The chunk retrieval time was compared to HDF5 and Tiff, and proven to be less sensitive to data location. This is the main reason we used Zarr for our experiments when using different physical computers. We have implemented a napari plugin that orchestrates the ImSwitch experiments and displays the images as layers. The plugin displays the metadata of each file to enable experimental reproducibility.

Increasing the microscope throughput is a common challenge in super-resolution microscopy. In SMLM, studies have shown how illuminating the sample with uniform illumination [49,50] increases the field of view up to 200 $\times$ 200 μm^2. Using parallelized illumination and specialized optics [30,51] in RESOLFT, up to 130 $\times$ 130 μm^2 was reached. These approaches can be combined with tiling to extend the throughput further. We apply the proposed framework to tiling super-resolution images using the molecular nanoscale live imaging with sectioning ability (MoNaLISA) [31], a microscope based on the concept of reversible saturable optical fluorescence transitions (RESOLFT) [52–54]. The microscope setup aims at parallelizing the illumination using patterned light to extend the field of view (FOV) and achieve faster recordings. Our approach provides a general framework that can be applied to a variety of microscopy techniques and experiments to increase the throughput not only in acquisition but also in reconstruction and analysis.

We have focused on parallelizing the acquisition and reconstruction tasks by distributing the computational workload in different units, further increasing the throughput of the technique. We have performed two experiments that encompass tiling and timelapse imaging to extend the microscope throughput further. We imaged actin cytoskeleton and mitochondria in human epithelial cells and other cells of increased morphological complexity, such as neurons. The proposed solution extends the recording to multiple cells and can be applied to whole sample screening in theory. As proof of principle, we recorded a FOV of 160x160μm^2 from 5x5 tiles of 38x38μm^2. The experiment was performed using a Python script executed in ImSwitch, which follows two steps: (1) widefield-based user registration of the focus in each tile, and (2) automatic RESOLFT imaging. The registration was needed to compensate for movement and tilt of the sample respect to the stage. We added overlap between the tiles so that the reconstructed images could be aligned in a post-processing step, reducing the constrain of stage precision in (x, y).

Hardware description

We present a file-based synchronization framework to simultaneously perform image acquisition, reconstruction, and analysis in microscopy applications. Fig. 1 exemplifies a typical microscope automation procedure using the proposed framework, where three units are synchronized to perform high-throughput imaging of a neuronal cell by tiling. One cycle consists of (1) imaging a FOV (i.e., tile) using a microscope and the acquisition unit, which will generate the raw data; (2) reconstructing the raw data into a final image of the tile; and (3) visualizing the image as a napari layer and post-processing. The cycle is concurrently performed multiple times. Essentially, the units will run independently (see Fig. 1 timeline) to extend the FOV by repeating the cycle for each tile. All the tiles will then be stitched to build the final image, thus expanding the microscope's throughput. The execution is handled using python scripts, which contain all the commands to perform the experiments, acquire data in each tile, move the stage between each tile, and collect the user parameters from the GUI.

It is important to remark that the presented approach is not limited to a specific number of units, and multiple units can run on the same physical computer. Each implementation will depend on two factors: the performance and specifications of the computers and the requirements of the experimental acquisition and data reconstruction applications (CPU usage, RAM occupied, and time of processing). For example, in microscopy applications where the recorded images are large, the reconstruction process might allocate a significant portion of the RAM and result in computer freezing, compromising the experiments. In this case, using separate computers for each unit is beneficial to maximize the resources. Then, the acquisition will not be affected by delays in the reconstruction unit.

If the resources of the computer allow, the framework can run on the same computer. This can be implemented either with the same unit for acquisition, reconstruction, and analysis using multithreading or by distributing the three tasks to

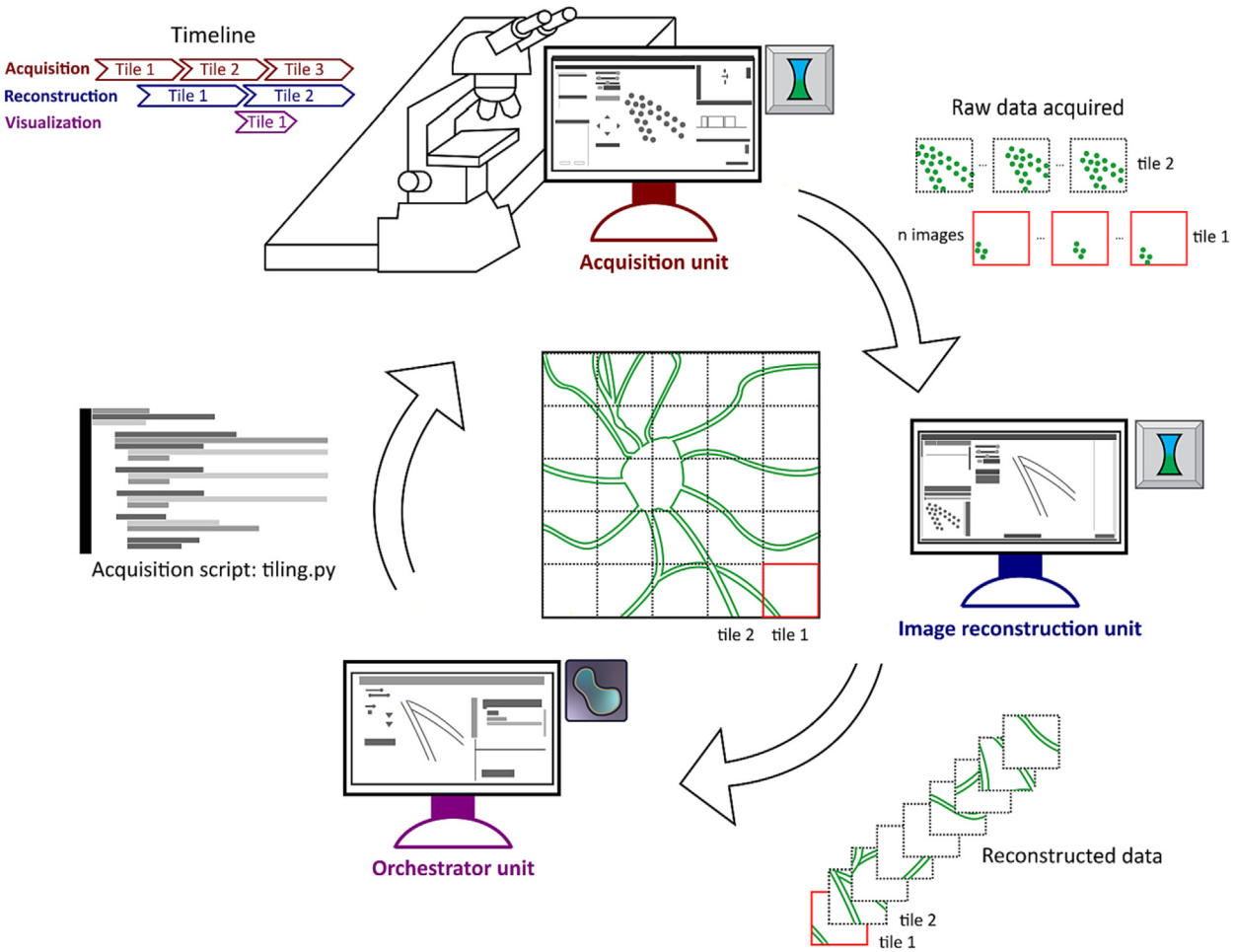

Fig. 1. General framework for simultaneous experiment acquisition, image reconstruction, and visualization using multiple computational units. The microscope experiments are performed in the acquisition unit by executing python scripts, which are created and distributed by the orchestrator unit. The user specifies the acquisition parameters in the scripts, such as number of tiles and laser powers. The recorded data is saved on disk (raw data), which acts as a queue for the image reconstruction unit. The image reconstruction unit turns the acquired raw data into tiles in a parallelized manner (see timeline in the top left corner) and can be visualized and post-processed in the orchestrator unit. Repeating this cycle gives an image of a neuron with an increased FOV.

different units. We chose the latter approach because it offers two benefits with respect to the first. Firstly, it is compatible with multiple software for data acquisition and image reconstruction. For example, the framework can use ImSwitch for acquisition and a Python script for reconstruction using the File Watcher python file provided (Design Files). Secondly, multiple microscopes (acquisition units) can delegate to the same unit for image reconstruction, which can be beneficial in microscope facilities and laboratories.

Each of the presented units serves a different task: acquisition, reconstruction, and orchestration. Firstly, the acquisition unit features an ImSwitch instance with the control module – *imcontrol*. This unit will perform experiments on the microscope by controlling hardware devices and storing the imaging data (raw data) and experimental metadata in disk or RAM. Each experiment is represented by a python script file and is executed in the scripting module of ImSwitch – *imscripting*. The scripts have access to the custom-designed and exported Application Programming Interface (API), which can be easily extended to include new functionality. In practice, scripts can automate the microscope by changing the state of any hardware device and accessing the resulting data. Secondly, the reconstruction unit features an ImSwitch instance with the reconstruction module – *imreconstruct*. This module is designed for those microscopy applications that require signal-processing algorithms to generate the final images from the raw data (e.g., n frames are reconstructed into one image). Finally, the orchestrator instance features a napari instance with the *napari-file-watcher* plugin. The plugin provides basic script editing functionality and displays the new reconstructed images as napari layers for further analysis. The user performs experiments using the script editing module in the orchestrator graphical user interface (GUI) and visualizes the incoming data as image layers.

The benefit of using napari for image analysis is that the reconstructed data can be post-processed by one of the available plugins in the napari hub (for example, image segmentation). Users can also develop plugins and contribute to the broader community. Image reconstruction is separated from visualization to cover those applications where raw data require specialized scripts for reconstruction that are not distributed as napari plugins.

The proposed solution can be extended to other experimental applications by simply designing different scripts, for example, performing a 3D stack or timelapse imaging. A complete list of API functions for scripting is available in the *readthedocs* documentation of ImSwitch, as well as instructions on adding new functionality (https://imswitch.readthedocs.io/en/stable/). It also contains a detailed description of the experimental metadata of ImSwitch and how to load it, the GUI components, and information on how to expand ImSwitch to multiple microscopy modalities and hardware devices. Furthermore, the framework can be generalized to other low-cost computing devices, such as Raspberry Pi, Arduino, and Jetson Nano. The only requirement is a shared file system and a python software package for image reconstruction and file watching.

The software pipeline is described in Fig. 2. The user interacts with the orchestrator unit, which is implemented as a napari plugin (*napari-file-watcher*) and contains two widgets (Fig. 2a). The *ImSwitch scripting* widget implements an editor that the user can use to define experiments and adjust imaging parameters. Once the script is finalized, it is added to the shared filesystem (FS). The second widget, the *File watcher,* displays the experimental results as napari layers (already reconstructed). The user can post-process the images using existing napari plugins or develop other processing pipelines on top.

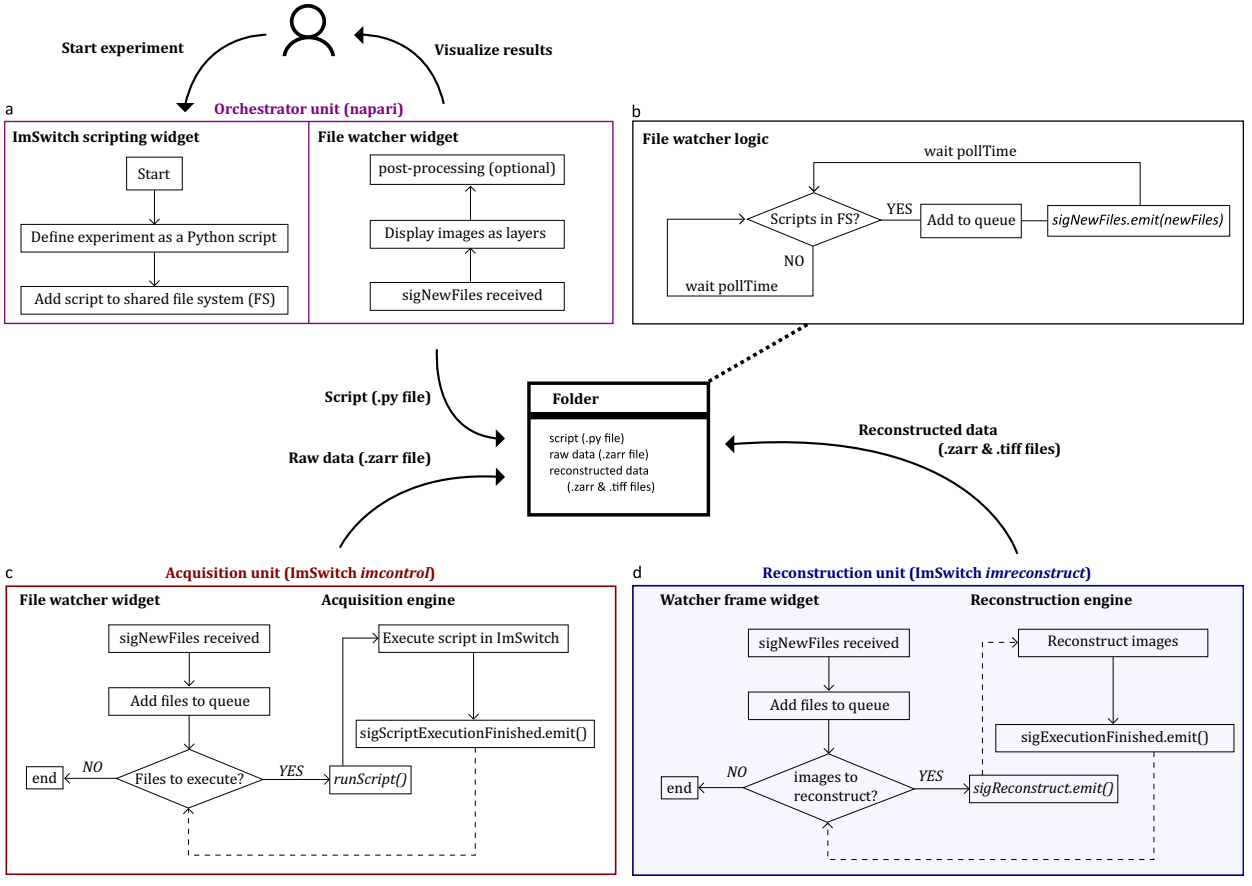

Fig. 2. Software pipeline using multiple units. a. The orchestrator is an instance of napari with the *napari-file-watcher* plugin. The user interacts with the orchestrator unit by defining experiments as python scripts, which are then added to the filesystem using the ImSwitch scripting widget. Once the reconstructed images are received, they are displayed as napari layers available for post-processing. b. The logic for each file watcher in Python. It periodically monitors a filesystem (FS), adds the incoming files to a queue and sends them using the *sigNewFiles* signal. c. The acquisition unit is an instance of *ImSwitch*. The file watcher widget displays and tracks the new files in each folder. Whenever new files are in the FS, it adds them to a queue and executes them sequentially. After every execution, the signal *sigScriptExecutionFinished* is called. d. The reconstruction unit is also an instance of ImSwitch, featuring the *imreconstruct* module. The workflow is similar to the acquisition, with the difference that the files are raw images that are reconstructed with an image processing algorithm. The reconstruction function is not called directly but through the *sigReconstruct* signal instead. This choice is due to the architecture of the *imreconstruct* module. Solid lines represent direct connections, and dotted lines are calls using python signals.

Each unit implements the file watcher thread (Fig. 2b) to monitor a folder periodically (pollTime) until new files arrive. Then, it adds the files to a queue and sends them using the signal *sigNewFiles*. This signal is connected to the main engine in each unit so that files are processed. We provide the file for the file watcher thread so that it can be used by other python software in any of the units.

The acquisition and reconstruction units are similar (Fig. 2c-d). Each of them implements a file watcher thread that monitors a folder selected by the user and then adds the files to a queue. Each file is processed sequentially, and a signal is emitted when each execution is finalized. The acquisition unit deals with python scripts that execute the experiments (*runScript*) and emits the signal *sigScriptExecutionFinished*. The reconstruction handles raw images that are reconstructed, followed by emitting the signal *sigExecutionFinished*.

The user can select between saving the raw images in Zarr or HDF5 files in the acquisition unit. The metadata is also included as an attribute (*ImSwitchData)* and contains all the experimental details needed for reproducibility. The reconstruction unit saves the data in both TIFF and Zarr files.

The file watcher functionality is implemented in the model layer of ImSwitch, and the orchestrator as an independent plugin distributed in the napari hub. The widgets can be easily applied to other python software packages and monitor any file extension added to the initialization function, providing generalization to multiple applications. A logger file is written into the folder once the experiment is finalized. It contains information about the computer name and starting date, all the files processed, and the time between arrival and end of execution of each item. Therefore, the performance of each computer can be easily evaluated for different reconstruction algorithms and experimental applications using resource monitor software, such as the Resource Manager in Windows.

Design files summary

Design file name	File type	Open source license	Location of the file
Computer framework: Fig. 1.eps	Figure	Creative Commons Attribution 4.0 International	Zenodo: https://zenodo.org/record/7561142
Automation pipeline: Fig. 2.eps	Figure	Creative Commons Attribution 4.0 International	Zenodo
Timelapse tiling script: tiling.py	Python script	Creative Commons Attribution 4.0 International	Zenodo
Selective tiling script incl. registration: timelapse.py	Python script	Creative Commons Attribution 4.0 International	Zenodo
Napari-file-watcher plugin: napari-file-watcher.zip	Python software	GNU General Public License (GPL)	Zenodo and https://www.napari-hub.org/plugins/napari-file-watcher release v0.1.1
File watcher model: FileWatcher.py	Python software	GNU General Public License (GPL)	Zenodo
File watcher ImSwitch acquisition widget: WatcherWidget (ImSwitch imcontrol).zip	Python software	GNU General Public License (GPL)	Zenodo
File watcher ImSwitch reconstruction widget: WatcherFrame (ImSwitch imreconstruct).zip	Python software	GNU General Public License (GPL)	Zenodo
Tiling widget: TilingWidget (ImSwitch imcontrol).zip	Python software	GNU General Public License (GPL)	Zenodo
ImSwitch version (ImSwitch-2.0.0.zip)	Python software	GNU General Public License (GPL)	Zenodo and https://github.com/kasasxav/ImSwitch release v2.0.0

Bill of materials summary

Designator	Component	Number	Cost per unit - currency	Total cost - currency	Source of materials	Material type
Acquisition unit	Dell Precision 5820 MT	1	2.500€	2.500€	Dell	
Reconstruction unit	HP Workstation Z2 Mini G3	1	1.500€	1.500€	Hewlett-Packard	

Build instructions

The requirements to implement the framework using different computers are a microscope with a workstation (acquisition) and a second computer to perform the reconstructions in real-time. If the acquisition computer is powerful enough, both units can be executed in the same machine as well. The orchestrator can often be included in the acquisition or reconstruction computers because it requires a minimum workload. The user can implement real-time image analysis pipelines on top of the visualized images using napari plugins in the orchestrator unit.

ImSwitch must be installed in both the acquisition and reconstruction units and napari with the *napari-file-watcher* plugin in the orchestrator unit. This section provides a detailed description of the installation procedure and how to set up the framework for experimental orchestration. We recommend that the user or developer reads the ImSwitch online documentation to check for device compatibility.

ImSwitch installation and configuration

We have released a new version of ImSwitch (v2.0.0) that supports the features presented in this article, which is included in the Python Package Index (PyPI). Therefore, the only necessary steps in order to install ImSwitch are to install conda and run the following commands:

```
conda create -n imswitch-env
conda activate imswitch-env
pip install imswitch
imswitch
```

After installing ImSwitch, a configuration folder is generated in "*Documents*" > "*ImSwitchConfig*". A standard procedure is to add the setup file containing the hardware device list and software widgets to display in "*imcontrol_setups*" folder (see more in the documentation). Upon installation, there is already a list of example files, and upon initialization, the user can choose one of the available options. After ImSwitch is initialized, the setup file to load can be changed in "*Tools*" > "*Pick hardware setup...*".

The widget with the functionality for the file watcher needs to be included in the "*availableWidgets*" by the end of the setup file as "*Watcher*". Fig. 3 shows the ImSwitch GUI in the acquisition unit for the microscope that we employed in the experimental part of this work (see Validation and Characterization).

The file "*config*"> "*modules*" contains a list of the GUI modules to be displayed. We recommend only having "*imcontrol*" and "*imscripting*" in the acquisition unit and "*imreconstruct*" in the reconstruction unit.

Fig. 4 shows the reconstruction unit ImSwitch GUI. The reconstruction module is implemented for parallelized RESOLFT strategies. However, different modules can be implemented and added to ImSwitch. It is also possible to use other software reconstruction alternatives and implement the File Watcher widget, similar to how it is developed in the napari plugin.

Napari installation

Napari is available in PyPI as well. Therefore, to install napari:

```
conda create -n napari-env
conda activate napari-env
pip install napari
napari
```

After napari is installed, the plugin "*napari-file-watcher*" needs to be installed. The plugin can be installed from "*Plugins*"> "*Install/Uninstall Plugins...*". The GUI is shown in Fig. 5.

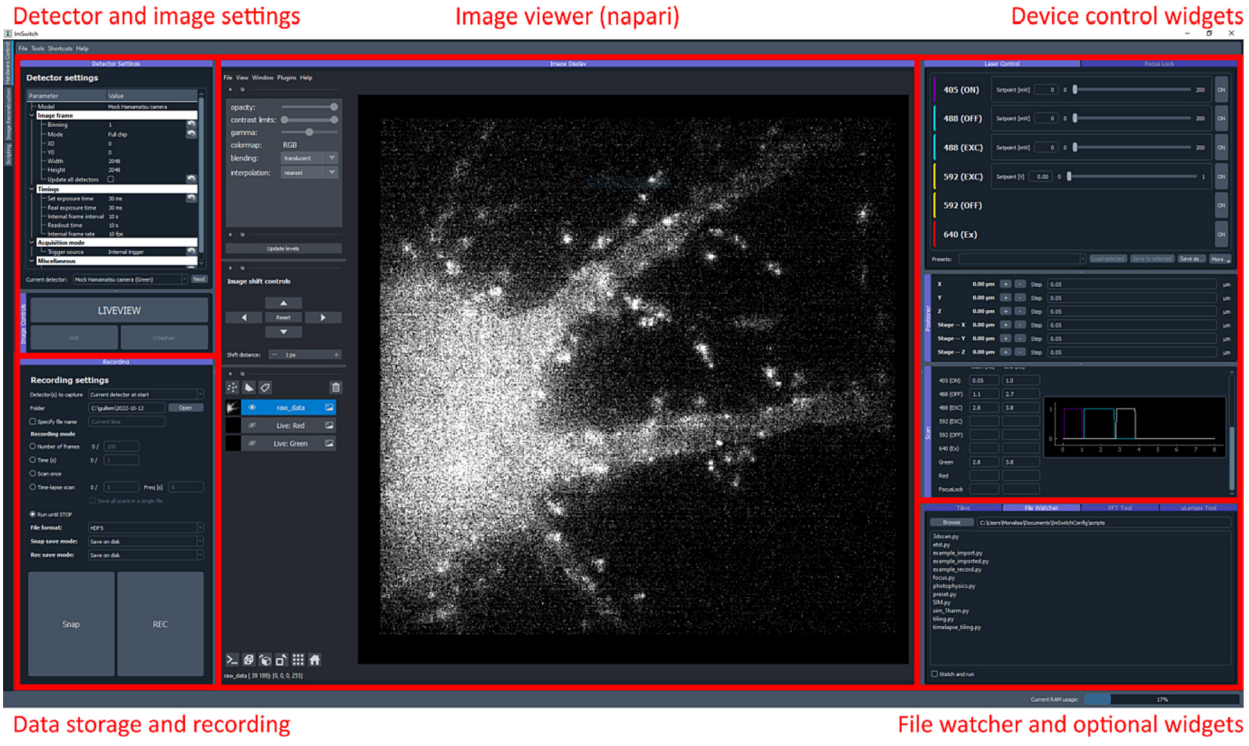

Fig. 3. ImSwitch GUI, acquisition unit. The GUI implements different widgets for the control of devices (“*Device control widgets*”) and experimental acquisition (“*Detector and image settings*”). The “*Data storage and recording*” module saves data into disk or memory. The file watcher and other specialized widgets are implemented in “*File watcher and optional widgets*”.

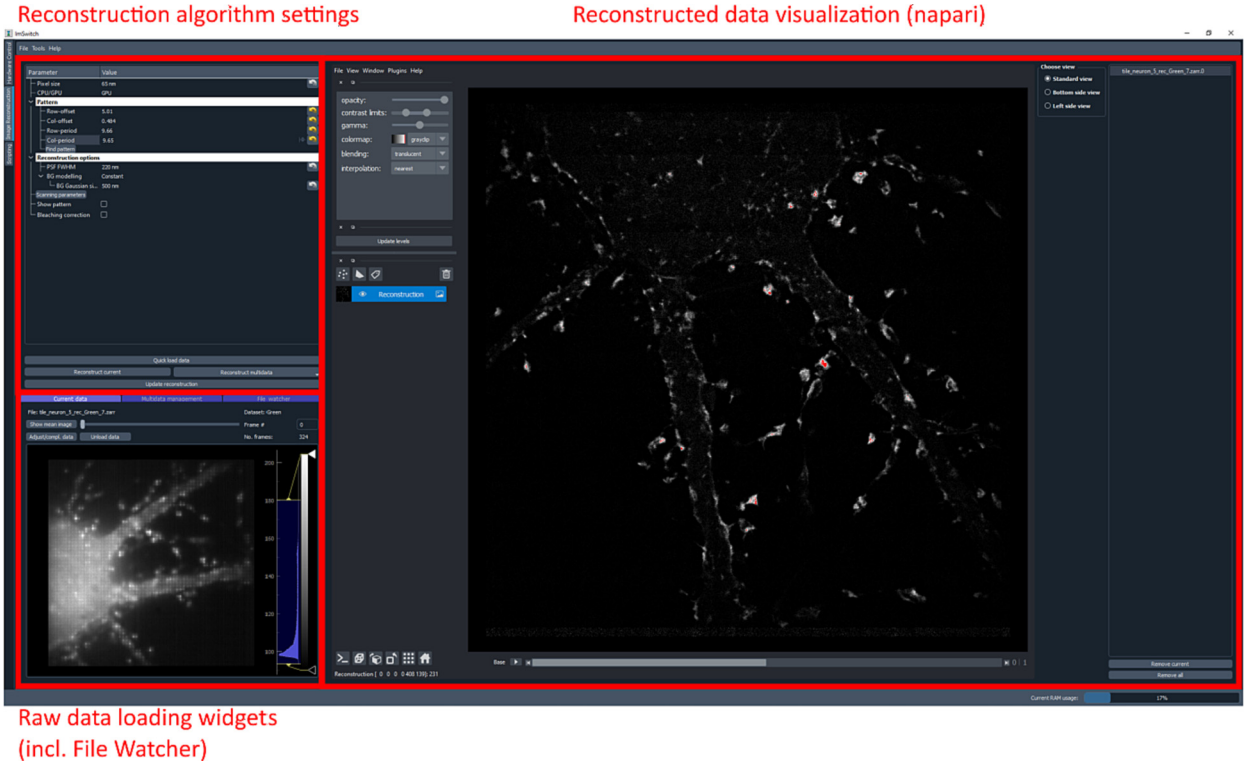

Fig. 4. ImSwitch GUI, reconstruction unit. The raw data is displayed in the “*Raw data loading widgets*” section, and the reconstructed data is in the “*Reconstructed data visualization*” module. The reconstruction algorithm parameters can be set in “*Reconstruction algorithm settings*”.

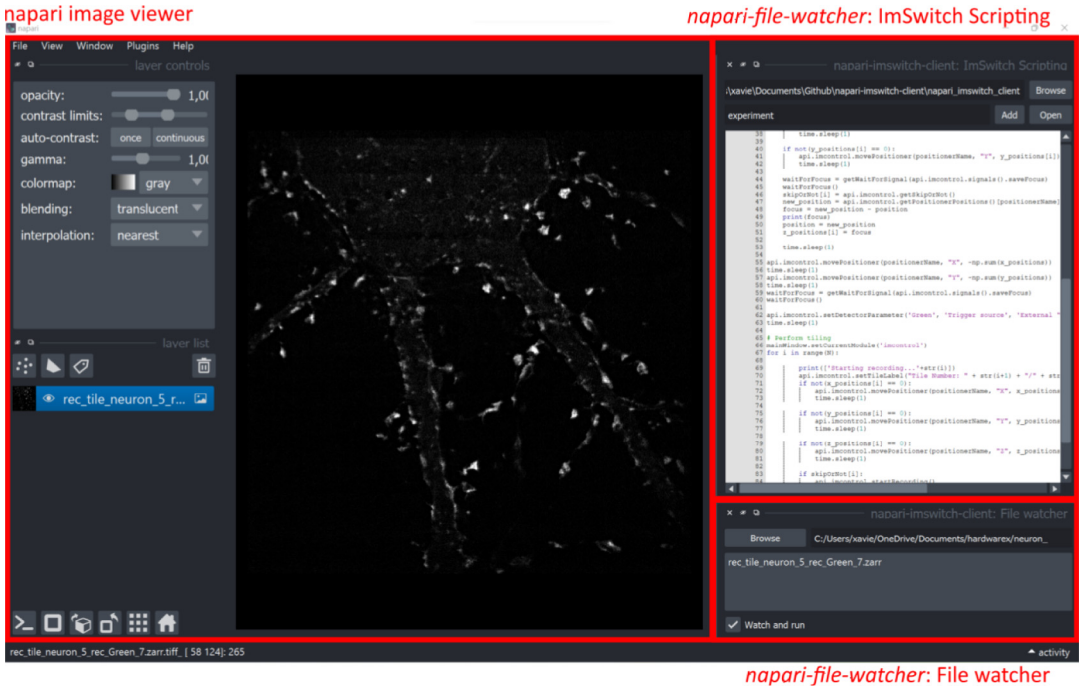

Fig. 5. Napari GUI and the *napari-file-watcher* plugin, orchestrator unit. The plugin contains two widgets: "ImSwitch Scripting" for editing and creating execution scripts and the "File Watcher" waits for new images to be displayed.

Operation instructions

Before starting the experiments, all units must be synchronized by setting up the file watchers. The user selects the filesystem folder, and folders for scripting and reconstruction are then automatically created. The following steps should be followed to set up the experimental framework:

1. In the acquisition unit, the scripting folder can be selected by clicking on the "*Browse*" button and then check the "*Watch and run*" box.
2. The best practice is to have a disk in the acquisition unit that can be shared over the network. This can be easily done in Windows by opening File Explorer and right-clicking on the disk of interest, and then "*Properties*" > "*Sharing*" > "*Advanced Sharing*" > "*Permissions*". The user used in the reconstruction unit should be added here. However, other shared storage options are also possible, as long as both the acquisition and reconstruction units have writing access to the filesystem.
3. In the acquisition unit, in the recording settings widget (bottom left of the GUI), the user can select the folder where the experimental data will be saved, which will be a folder in the shared disk. Also, select Zarr as the file extension.
4. In the reconstruction unit, in the bottom-left corner and the widget "*File Watcher*," the data folder should be selected, and then check the "*Watch and run*" box.
5. The reconstructed data will be saved in a folder named "*rec*" inside the data folder. Select that folder in the file watcher widget of the "*napari-file-watcher*" plugin, and then press "*Watch and run*."
6. Finally, select the scripting folder (same as in 1.) from the scripting widget in the "*napari-file-watcher*" plugin, write scripts or open existing ones, and then start the experiment by pressing "*Add*".

The script can access any API exported function, and new functions can be easily added using the @APIExport decorator in any controller. A list of the accessible functions is shown on the documentation page. Fig. 6 shows an example script for performing timelapse imaging of N = 10 lapses in the (x,y,z) coordinates. Each of the lapses is a scan-based experiment that saves the raw data on disk.

The scripting engine is in charge of executing the files in the acquisition unit (*imscripting*). It also has a GUI, which we display in Fig. 7. It provides basic script editing functionality and the possibility to run and stop experiments. The GUI can be helpful first to test, debug and implement the experimental scripts locally before using the full framework. After the scripts are implemented, the orchestrator unit will be used to adapt the scripts further and send them to the acquisition unit.

```python
import time
import numpy as np

mainWindow.setCurrentModule('imcontrol')

N = 10
positionerName = "Stage"
x = 100
y = 150
z = 0.5

api.imcontrol.movePositioner(positionerName, "X", x)
api.imcontrol.movePositioner(positionerName, "Y", y)
api.imcontrol.movePositioner(positionerName, "Z", z)

for i in range(N):
    api.imcontrol.startRecording()
    waitForRecordingToEnd = getWaitForSignal(api.imcontrol.signals().recordingEnded)
    waitForRecordingToEnd()
    time.sleep(1)
```

1. Show hardware control module
2. Define script parameters:
number of lapses (N)
stage name (positionerName)
position coordinates (x, y, z)
3. Move stage to a (x,y,z) position
4. Perform timelapse imaging (scanning) and data storage

Fig. 6. Timelapse ImSwitch automation script. N = 10 lapses are executed in the (x,y,z) stage position coordinates. The ImSwitch API is exported and accessible with the "*api.imcontrol*" call.

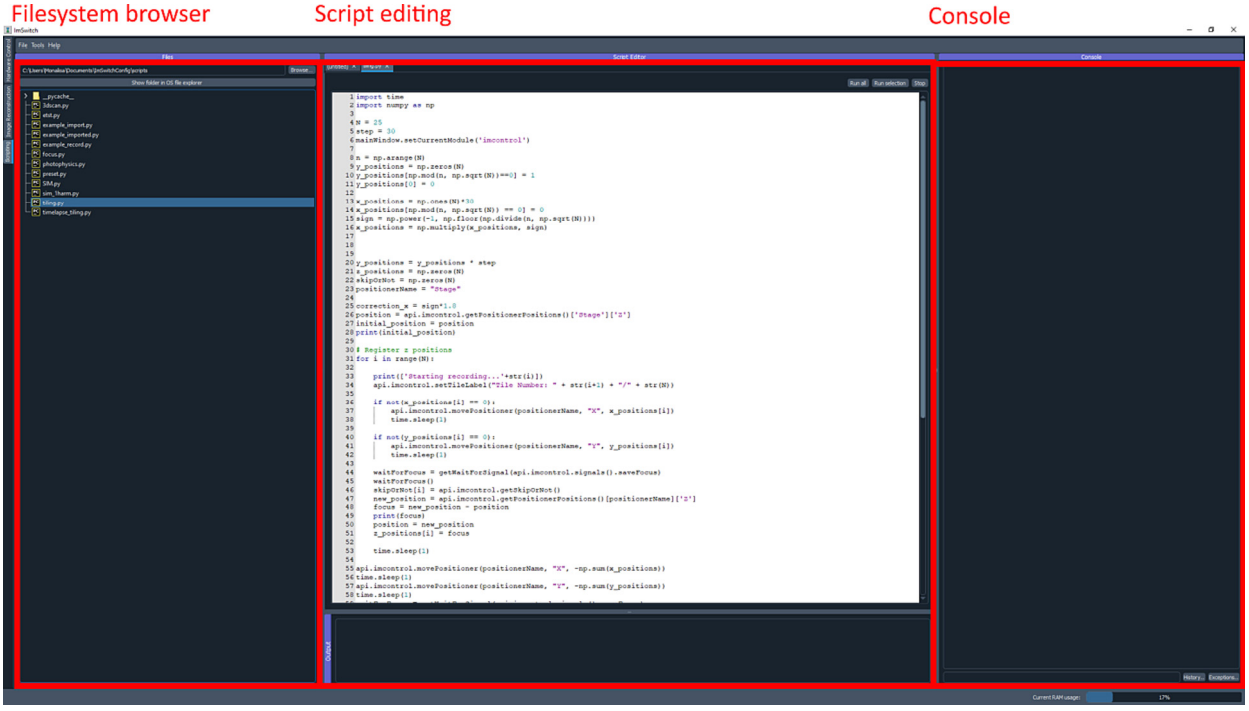

Fig. 7. ImSwitch GUI in acquisition unit, *imscripting* module for editing and executing scripts. The scripts can be loaded in "*Filesystem browser*", edited, and executed in the "*Script editing*" section, and a "*Console*" is implemented for debugging purposes.

Validation and characterization

In order to validate and characterize the proposed framework, we have used the MoNaLISA [31] microscope. MoNaLISA is a parallelized RESOLFT method that offers a large FOV (up to 50x50μm^2) with an increased speed (0.3–1.3 Hz). We chose two use cases related to timelapse and tiling images of extended regions for different biological structures: mitochondria and actin in U2OS and neuronal cells. The main focus is to automate the acquisition using scripts and increase the microscope throughput by distributing the acquisition and reconstruction workload using different devices.

Observing mitochondrial dynamics with super-resolution microscopy

MoNaLISA has been previously employed to study mitochondria dynamics in U2OS cells [31,32,55]. Mitochondria is a highly dynamic organelle and demands an equally fast imaging strategy to follow its movements over the cell. Therefore, throughput in this biological application means (1) the possibility of acquiring multiple frames with a minimum delay caused by the computational acquisition and reconstruction and (2) observing the dynamics in multiple cells to acquire statistical relevance. Previously, to keep the temporal resolution, experiments were first performed, and the data were reconstructed and analyzed offline. However, there will be a delay until the user receives substantial feedback, such as whether the selected cells presented relevant dynamics. Visualization of experimental data in real-time is also crucial in the case of drug treatment when the administration of a drug needs to happen at a specific moment during the acquisition. Our approach can be beneficial to perform adaptive imaging, adjusting imaging parameters based on the analysis of the data [21].

The use case presented involves observing cell dynamics with super-resolution details, such as visualizing the outer membrane compartment in an extended region by performing a cyclic time-lapse experiment. In particular, extending the FOV can be obtained using a motorized stage that moves from tile to tile performing one acquisition at each position. If this procedure is repeated (here ten times), the cell dynamics can be observed in the entire extended area. In order to maintain the stability of the focus during the movement, a focus lock strategy is implemented following previously-published instructions [25]. The focus lock keeps the focus automatically over time by using the cover glass reflection and successfully adjusts to the 2x2 tiling movement.

The experimental design and results are shown in Fig. 8. A script contains the experimental instructions to be sent to the microscope. In this case, we perform a MoNaLISA scan in every tile and save the files on the hard drive. In between scans, the motorized stage moves to the following area. We use the digital outputs of a data analog card (*NIDAQ PCI 6371*) to synchronize hardware devices (e.g., lasers and stage axis) with the camera (externally triggered) using a pulse scheme optimized for RESOLFT imaging. The script can successfully wait for the end of the scan by using exported signals in the *imscripting* module of ImSwitch.

By combining MoNaLISA with the presented multiunit framework, higher throughput can be achieved by distributing the reconstruction workload (limited mainly by the RAM usage) and improving the computation time while keeping the lateral and temporal resolution introduced by the technique.

Selective tiling in neurons and U2OS cells with a user-driven approach

As a second application, we extended the FOV of the MoNaLISA microscope by applying the proposed framework in tiling experiments to achieve higher throughput. We imaged extended sample regions of neurons and U2OS cells. Neurons are highly polarized cells that communicate with each other via specialized sites called synapses, which occur along neuritic processes such as axons and dendrites. These processes grow from the cell body and expand over large areas with high morphological and functional complexity relevant to synaptic organization and transmission [51]. Therefore, tiling in super-resolution microscopy is a method that can highly benefit neuronal imaging by massively expanding the FOV [25].

We developed a tiling script that interacts with the user in order to perform selective tiling. The script consists of two steps. Firstly, user-driven registration of focus and tiles is performed, with the possibility of annotating the tiles that can be skipped because they do not contain biological information. Secondly, the microscope performs automatic tiling of the selected tiles, performing a MoNaLISA scan in each position (Fig. 9a). The stage moves automatically between the tiles, and the script interacts with the user through a dedicated widget (*TilingWidget*).

The MoNaLISA optical setup used has a field of view of 38 $\times$ 38 μm^2 and collects the raw data by scanning the sample with piezoelectric actuators (*3-axis NanoMax* stage from Thorlabs). To move in between fields of view, we employed the stepper motors of the stage controlled through USB and integrated into ImSwitch using the *BSC203Manager* (which calls the python library *thorlabs-apt-devices*), and a 14 % overlap between tiles. The raw data consist of a stack of 324 frames and the acquisition time is 8 ms per frame (2.6 s per image), and we added an extra time of 1 s after each stage movement to prevent from drift. Because of the tilt between the sample and the stage, the z position was slightly different for each tile, which is why we added a widefield-based registration. This process could be further improved by characterizing the tilt or automatically finding the focus in each tile. We used the Fiji plugin MosaicJ for stitching the images [56].

We performed 5x5 tiles (Fig. 9b) to extend the FOV from 38 $\times$ 38 μm^2 (Fig. 9c) to 160 $\times$ 160 μm^2 (Fig. 9d-e) while keeping the super-resolution features introduced by the microscope (Fig. 9f-h).

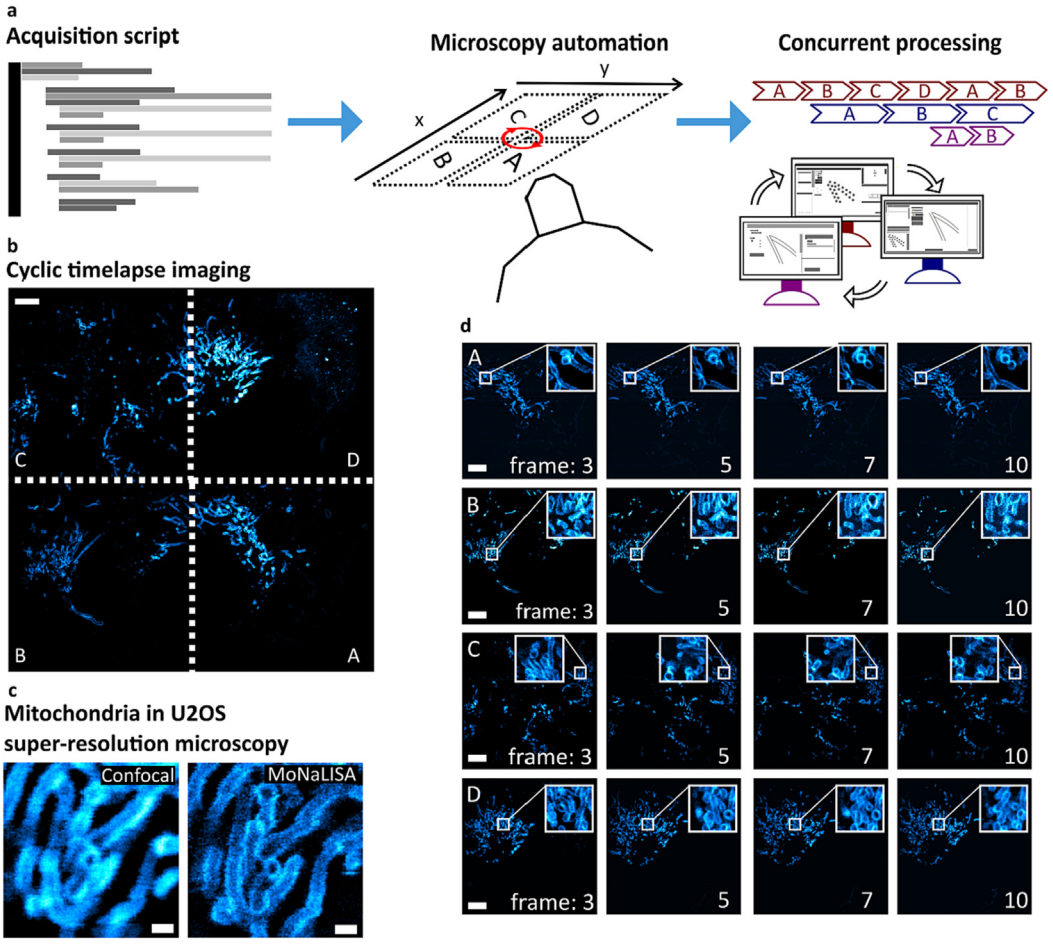

Fig. 8. Cyclic time-lapse imaging of mitochondria in U2OS cells in a 2x2 (A, B, C, D) tile array. **a** The acquisition script (timelapse-tiling.py) drives the microscopy automation acquisition by calling ImSwitch functions. Concurrent processing enables subsequent acquisition and reconstruction of the image tiles. **b** Cyclic timelapse images are performed by imaging each of the tiles and sequentially repeating the procedure. Scale bar 5 μm. **c-d** The super-resolution features can be observed during a prolonged time (10 super-resolved frames) for each tile independently in U2OS cells expressing OMP25-rsEGFP2. A zoom-in visualizes the dynamics of selected areas in both confocal and MoNaLISA modalities. Scale bars 0.5 μm and 5 μm, respectively.

Furthermore, we extracted the reconstruction time of the remote unit from the logger files. The logger files contain information about the processing computer and the time of execution of each file. We displayed the reconstruction time for 200 acquisitions in Fig. 9i. The total reconstruction time ranged between 25 and 45 s when using an external unit and 7–10 s if the unit ran on the same computer as the acquisition machine.

Evaluation of acquisition and reconstruction latency and resources

The system can be employed either by using multiple computers or running ImSwitch instances within the same machine. In order to evaluate whether multiple computers are necessary, the workload of each process needs to be assessed for each application. In Table 1. we characterize the occupied resources of the acquisition and reconstruction units in terms of RAM, CPU usage, and computation time for the proposed use cases. We have done so by using the Resource Monitor in Windows for the RAM and CPU usage and the *datetime* python library directly in ImSwitch to precisely compute the timings.

Results show that CPU usage is always low and close to idle (control) since our experiments use digital triggering from an external data acquisition card. Also, the raw data is directly saved on a hard drive. CPU usage would be higher in the case of microscopy modalities that require software triggering or real-time image processing. In our use cases, the RAM is the limiting factor, occupying > 30 GB in total (especially in the reconstruction unit). Therefore, using a separate computer for the reconstruction unit is beneficial for computers with less available memory. In our proof of concept, the reconstruction time would then increase from 7 to 10 s to 30–50 s. This could be further improved using higher-speed networks or higher-performance devices for computation or optimizing the Zarr data saving (e.g., chunk size).

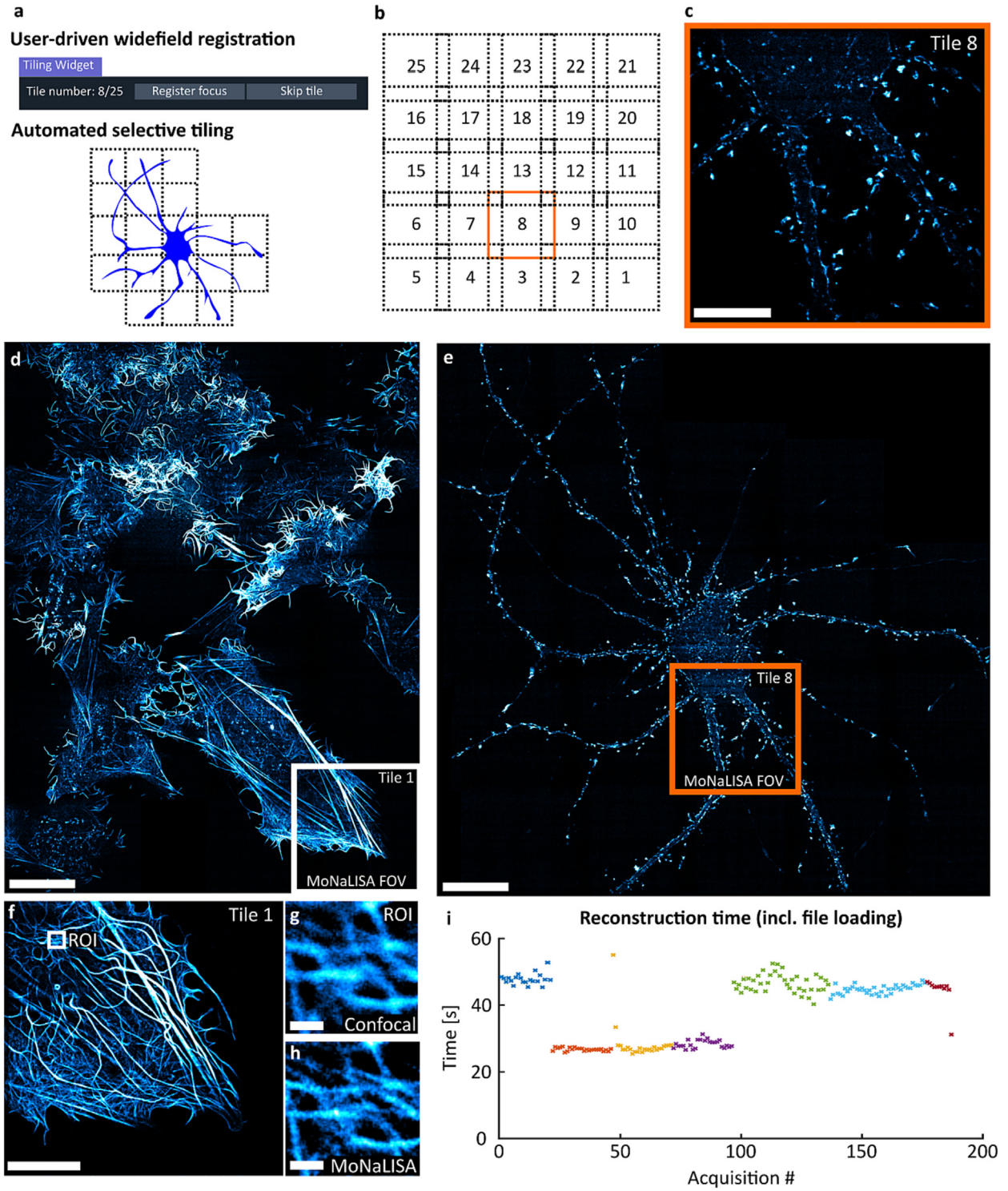

Fig. 9. Selective tiling for extending the FOV to 160x160μm^2 (25x25 tiles). **a** The focus positions are registered by the user using a widefield laser and an ImSwitch widget. The user can skip the tile, thus adapting to the image content. The microscope then performs selective tiling by imaging only the selected areas. **b-c** Schematic of the tiling scheme, including 14 % overlap, and visualization of a single tile (38x38μm^2). **d-e** Imaging the actin cytoskeleton of U2OS cells and primary hippocampal neurons expressing actinChromobody-rsEGFP2. Scale bar 20 μm. **f-h** zooming in Tile 1 in the U2OS cells (d), and comparison between confocal and MoNaLISA. Scale bars 5 μm and 0.5 μm, respectively. **i** Reconstruction time of 200 executions using a remote instance. Each color represents a different experiment, and each data point is a single image acquisition (or tile). The values were extracted from the logger files for all the experiments performed using the framework (200 acquisitions).

Table 1
Study of the computation resources needed for acquisition and reconstruction.

	Idle (control)	Acquisition	Reconstruction from disk
RAM (total 64 GB)	6 GB	8.6 GB	15–23 GB
CPU usage	0 %	8–13 %	6 %
Time	–	2 s MoNaLISA image 35–55 MB/s writing time of experimental data.	0.4–0.5 s reconstruction algorithm 7–10 s local unit (incl. file loading) 30–50 s remote unit (incl. file loading)

The reconstruction time is always more extensive than the acquisition time. Therefore, the user performing image reconstruction using the GUI manually would impose an evident bottleneck. This is why, previously, the reconstruction was usually performed after the entire acquisition had finished. By distributing the resources and using multiple units for acquisition and reconstruction, the tasks can be performed simultaneously without compromising the temporal resolution of the microscope.

By characterizing the requirements of each microscope application, this framework could be extended to low-cost computing devices for both acquisition and reconstruction, such as Raspberry Pi and Jetson Nano.

To conclude, we have designed and developed a new framework for simultaneously performing acquisition, reconstruction, and visualization for microscope automation. The solution is implemented using the ImSwitch and napari software packages and is easily generalizable to other software. We validated the performance by applying it in the context of super-resolution microscopy and the distribution of computation resources to achieve higher throughput. We performed tiling and timelapse experiments to observe mitochondria and actin in U2OS cells and actin in primary neurons.

Ethics statements

Primary cultures of hippocampal neurons were prepared from rat embryos. All procedures were performed in accordance with the animal welfare guidelines of Karolinska Institutet and were approved by Stockholm North Ethical Evaluation Board for Animal Research.

Funding

This work was supported by the Chan Zuckerberg Initiative (napari Plugin Accelerator program, project “Integrating Technology to Improve Real-Time Microscopy”), the European Union (Horizon 2020 program, project IMAGEOMICS 964016), and Vinnova (2020-04702 Imaging-omics).

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] N. Scherf, J. Huisken, The smart and gentle microscope, Nat. Biotechnol. 33 (2015) 815–818.
[2] X. Yan et al, High-content imaging-based pooled CRISPR screens in mammalian cells, J. Cell Biol. 220 (2021) e202008158.
[3] S.K. Jones et al, Massively parallel kinetic profiling of natural and engineered CRISPR nucleases, Nat. Biotechnol. 39 (2021) 84–93.
[4] M.H.A. Schmitz, D.W. Gerlich, Automated Live Microscopy to Study Mitotic Gene Function in Fluorescent Reporter Cell Lines. in Mitosis: Methods and Protocols (ed. McAinsh, A. D.) 113–134 (Humana Press, 2009). doi:10.1007/978-1-60327-993-2_7.
[5] Y. Cai et al, Experimental and computational framework for a dynamic protein atlas of human cell division, Nature 561 (2018) 411–415.
[6] B. Li, C. Wu, M. Wang, K. Charan, C. Xu, An adaptive excitation source for high-speed multiphoton microscopy, Nat. Methods 17 (2020) 163–166.
[7] J. Dreier et al, Smart scanning for low-illumination and fast RESOLFT nanoscopy in vivo, Nat. Commun. 10 (2019) 556.
[8] J. Heine et al, Adaptive-illumination STED nanoscopy, Proc. Natl. Acad. Sci. 114 (2017) 9797–9802.
[9] H. Pinkard et al, Learned adaptive multiphoton illumination microscopy for large-scale immune response imaging, Nat. Commun. 12 (2021) 1916.
[10] C. Conrad et al, Micropilot: automation of fluorescence microscopy–based imaging for systems biology, Nat. Methods 8 (2011) 246–249.
[11] H. Pinkard et al, Pycro-Manager: open-source software for customized and reproducible microscope control, Nat. Methods 18 (2021) 226–228.
[12] S. Tosi et al, AutoScanJ: A Suite of ImageJ Scripts for Intelligent Microscopy, Front. Bioinformatics 1 (2021) 627626.
[13] Z.R. Fox et al, Enabling reactive microscopy with MicroMator, Nat. Commun. 13 (2022) 2199.
[14] A. Edelstein, N. Amodaj, K. Hoover, R. Vale, N. Stuurman, Computer Control of Microscopes Using μManager. Current Protocols in Molecular Biology 92, 14.20.1-14.20.17 (2010).
[15] C.A. Schneider, W.S. Rasband, K.W. Eliceiri, NIH Image to ImageJ: 25 years of image analysis, Nat. Methods 9 (2012) 671–675.
[16] D.M. Susano Pinto et al, Python-Microscope – a new open-source Python library for the control of microscopes, J. Cell Sci. 134 (2021) jcs258955.
[17] A.E.S. Barentine et al, PYME: an integrated platform for high-throughput nanoscopy, Biophys. J. 121 (2022) 137a.
[18] M.N. Alsamsam, A. Kopūstas, M. Jurevičiūtė, M. Tutkus, The miEye: Bench-top super-resolution microscope with cost-effective equipment, HardwareX 12 (2022).
[19] X. Casas Moreno, S. Al-Kadhimi, J. Alvelid, A. Bodén, I. Testa, ImSwitch: generalizing microscope control in Python, J. Open Source Softw. 6 (2021) 3394.
[20] J. Alvelid, M. Damenti, C. Sgattoni, I. Testa, Event-triggered STED imaging, Nat. Methods 19 (2022) 1268–1275.
[21] D. Mahecic et al, Event-driven acquisition for content-enriched microscopy, Nat. Methods 19 (2022) 1262–1267.

[22] H. Li, H. Soto-Montoya, M. Voisin, L.F. Valenzuela, M. Prakash, Octopi: Open configurable high-throughput imaging platform for infectious disease diagnosis in the field. bioRxiv 684423 (2019) doi: 10.1101/684423.
[23] L. Gao, Extend the field of view of selective plan illumination microscopy by tiling the excitation light sheet, Opt. Express 23 (2015) 6102–6111.
[24] S.K. Chow et al, Automated microscopy system for mosaic acquisition and processing, J. Microsc. 222 (2006) 76–84.
[25] J. Alvelid, I. Testa, Stable stimulated emission depletion imaging of extended sample regions, J. Phys. D Appl. Phys. 53 (2019) 024001.
[26] E. Betzig et al, Imaging intracellular fluorescent proteins at nanometer resolution, Science 313 (2006) 1642–1645.
[27] S.T. Hess, T.P.K. Girirajan, M.D. Mason, Ultra-high resolution imaging by fluorescence photoactivation localization microscopy, Biophys. J. 91 (2006) 4258–4272.
[28] M.J. Rust, M. Bates, X. Zhuang, Sub-diffraction-limit imaging by stochastic optical reconstruction microscopy (STORM), Nat. Methods 3 (2006) 793–796.
[29] A. Chmyrov et al, Nanoscopy with more than 100,000 'doughnuts', Nat. Methods 10 (2013) 737–740.
[30] A. Chmyrov et al, Achromatic light patterning and improved image reconstruction for parallelized RESOLFT nanoscopy, Sci. Rep. 7 (2017) 44619.
[31] L.A. Masullo et al, Enhanced photon collection enables four dimensional fluorescence nanoscopy of living systems, Nat. Commun. 9 (2018) 3281.
[32] A. Bodén et al, Volumetric live cell imaging with three-dimensional parallelized RESOLFT microscopy, Nat. Biotechnol. 39 (2021) 609–618.
[33] M.G.L. Gustafsson, Surpassing the lateral resolution limit by a factor of two using structured illumination microscopy, J. Microsc. 198 (2000) 82–87.
[34] R. Heintzmann, T.M. Jovin, C. Cremer, Saturated patterned excitation microscopy—a concept for optical resolution improvement, J. Opt. Soc. Am. A 19 (2002) 1599–1609.
[35] T. Dertinger, R. Colyer, G. Iyer, S. Weiss, J. Enderlein, Fast, background-free, 3D super-resolution optical fluctuation imaging (SOFI), Proc. Natl. Acad. Sci. 106 (2009) 22287–22292.
[36] S. Culley, K.L. Tosheva, P. Matos Pereira, R. Henriques, SRRF: Universal live-cell super-resolution microscopy, Int. J. Biochem. Cell Biol. 101 (2018) 74–79.
[37] M. Levoy, R. Ng, A. Adams, M. Footer, M. Horowitz, Light field microscopy, ACM Trans. Graph. 25 (2006) 924–934.
[38] G. Zheng, R. Horstmeyer, C. Yang, Wide-field, high-resolution Fourier ptychographic microscopy, Nat. Photonics 7 (2013) 739–745.
[39] A. Greenbaum, et al. Wide-field computational imaging of pathology slides using lens-free on-chip microscopy. Sci. Transl. Med. 6, 267ra175-267ra175 (2014).
[40] J. Ries, SMAP: a modular super-resolution microscopy analysis platform for SMLM data, Nat. Methods 17 (2020) 870–872.
[41] A. Przybylski, B. Thiel, J. Keller-Findeisen, B. Stock, M. Bates, Gpufit: an open-source toolkit for GPU-accelerated curve fitting, Sci. Rep. 7 (2017) 15722.
[42] K.J.A. Martens, A.N. Bader, S. Baas, B. Rieger, J. Hohlbein, Phasor based single-molecule localization microscopy in 3D (pSMLM-3D): an algorithm for MHz localization rates using standard CPUs, J. Chem. Phys. 148 (2018) 123311.
[43] H. Gong, W. Guo, M.A.A. Neil, GPU-accelerated real-time reconstruction in Python of three-dimensional datasets from structured illumination microscopy with hexagonal patterns, Philos. Trans. R. Soc. A Math. Phys. Eng. Sci. 379 (2021) 20200162.
[44] W. Ouyang, F. Mueller, M. Hjelmare, E. Lundberg, C. Zimmer, ImJoy: an open-source computational platform for the deep learning era, Nat. Methods 16 (2019) 1199–1200.
[45] S. Prigent, et al., BioImageIT: Open-source framework for integration of image data-management with analysis. bioRxiv 2021.12.09.471919 (2021) doi: 10.1101/2021.12.09.471919.
[46] A.E. Carpenter et al, Cell Profiler: image analysis software for identifying and quantifying cell phenotypes, Genome Biol. 7 (2006) R100.
[47] napari contributors (2019). napari: a multi-dimensional image viewer for python.
[48] J. Moore et al, OME-NGFF: a next-generation file format for expanding bioimaging data-access strategies, Nat. Methods 18 (2021) 1496–1498.
[49] K.M. Douglass, C. Sieben, A. Archetti, A. Lambert, S. Manley, Super-resolution imaging of multiple cells by optimized flat-field epi-illumination, Nat. Photonics 10 (2016) 705–708.
[50] A. Mau, K. Friedl, C. Leterrier, N. Bourg, S. Lévêque-Fort, Fast widefield scan provides tunable and uniform illumination optimizing super-resolution microscopy on large fields, Nat. Commun. 12 (2021) 3077.
[51] X. Casas Moreno et al, Multi-foci parallelised RESOLFT nanoscopy in an extended field-of-view, *J. Microsc.* n/a (2022).
[52] M. Hofmann, C. Eggeling, S. Jakobs, S.W. Hell, Breaking the diffraction barrier in fluorescence microscopy at low light intensities by using reversibly photoswitchable proteins. Proc. Natl. Acad. Sci. 102, 17565–17569 (2005).
[53] T. Grotjohann et al, Diffraction-unlimited all-optical imaging and writing with a photochromic GFP, Nature 478 (2011) 204–208.
[54] I. Testa et al, Nanoscopy of living brain slices with low light levels, Neuron 75 (2012) 992–1000.
[55] M. Damenti, G. Coceano, F. Pennacchietti, A. Bodén, I. Testa, STED and parallelized RESOLFT optical nanoscopy of the tubular endoplasmic reticulum and its mitochondrial contacts in neuronal cells, Neurobiol. Dis. 155 (2021) 105361.
[56] P. Thévenaz, M. Unser, User-friendly semiautomated assembly of accurate image mosaics in microscopy, Microsc. Res. Tech. 70 (2007) 135–146.

Ilaria Testa, Associate Professor KTH, SciLifeLab campus, Royal Institute of Technology, Department of Applied Physics. **Research experience: fluorescence microscopy, optical nanoscopy or super resolution light microscopy, STED/RESOLFT, single molecule, neuronal bioimaging.** I performed my PhD between 2006 and 2009 at the University of Genoa (Italy) in the group of Professor Alberto Diaspro working on the use of Photoswitchable Fluorescent Proteins in two photon microscopy for functional analysis in living cells. Between 2009-2014 I worked as a Postdoc Researcher at the Department of NanoBiophotonics directed by Professor Stefan Hell at the Max Planck Institute for Biophysical Chemistry in Göttingen (Germany). During this time I actively designed and developed several nanoscopes that implement two different approaches for achieving super-resolution imaging: (1) the first set-up was based on stochastic switching of single molecule between a fluorescent and a non-fluorescent state (GSDIM/PALM/STORM) with multicolour ability based on ratiometric detection (2) the second set-up showed the practical accomplishment of RESOLFT, a target switching technique based on reversible long lived molecular transition. Within an interdisciplinary team of biologists and physicists we succeeded to demonstrate RESOLFT in living cells and even tissues. Since 2015 I'm conducting my independent research at the Science for Life Laboratory as Faculty at the Royal Institute of Technology (KTH, Stockholm, Sweden). The goal of my group is to develop the novel paradigms made available by super-resolution microscopy to address contemporary challenges in biophysics and neuroscience. To achieve this goal we push forward the quantitative aspect of live cell imaging by setting-up and applying different concepts of super-resolution microscopy based on target switching such as automated microscopy. We developed new nanoscopes named MoNaLISA and smart RESOLFT and etSTED including the software ImSwitch. Together these technologies allow the precise identification of populations of neuronal proteins and organelles depending on their localization, abundance and dynamics inside their native environment.