Solving Hierarchal Neuroscience Problems With Parsl

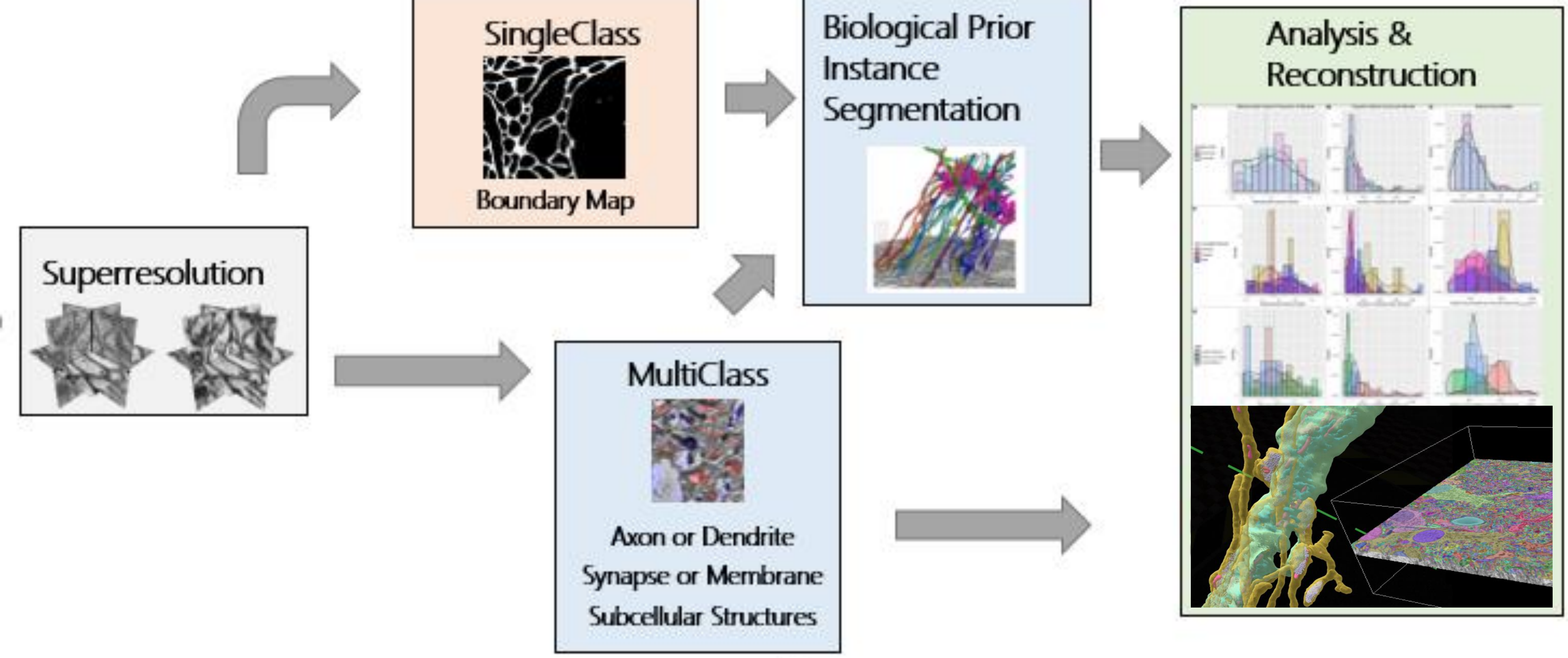
Matthew Madany, madany@ucsd.edu

CRBS
CENTER FOR RESEARCH IN
BIOLOGICAL SYSTEMS

NCMIR
National Center for Microscopy and Imaging Research

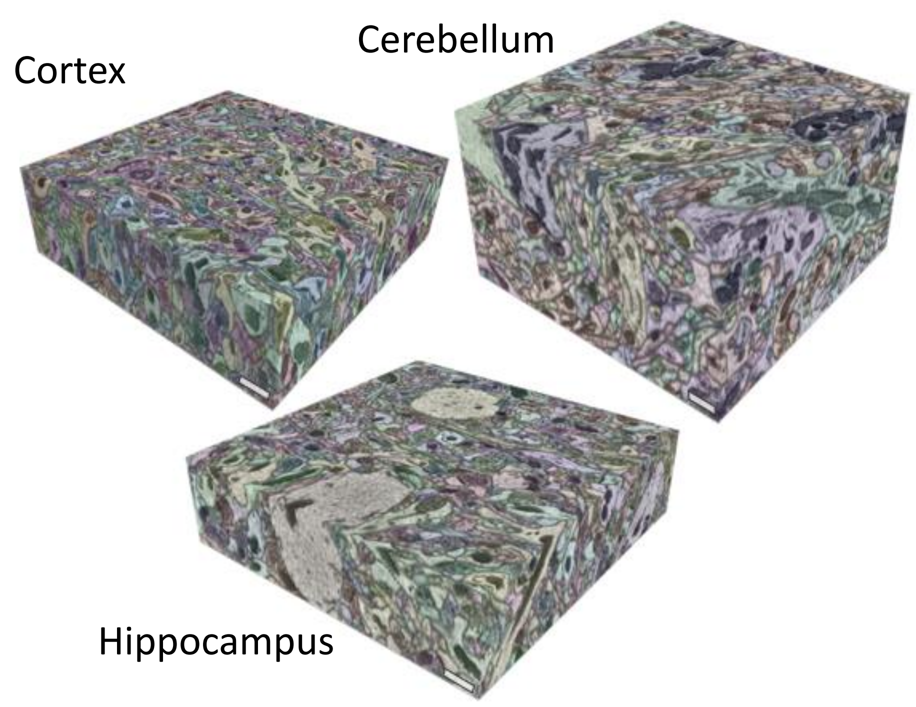
UC San Diego
SCHOOL OF MEDICINE

SDSC
SAN DIEGO
SUPERCOMPUTER CENTER



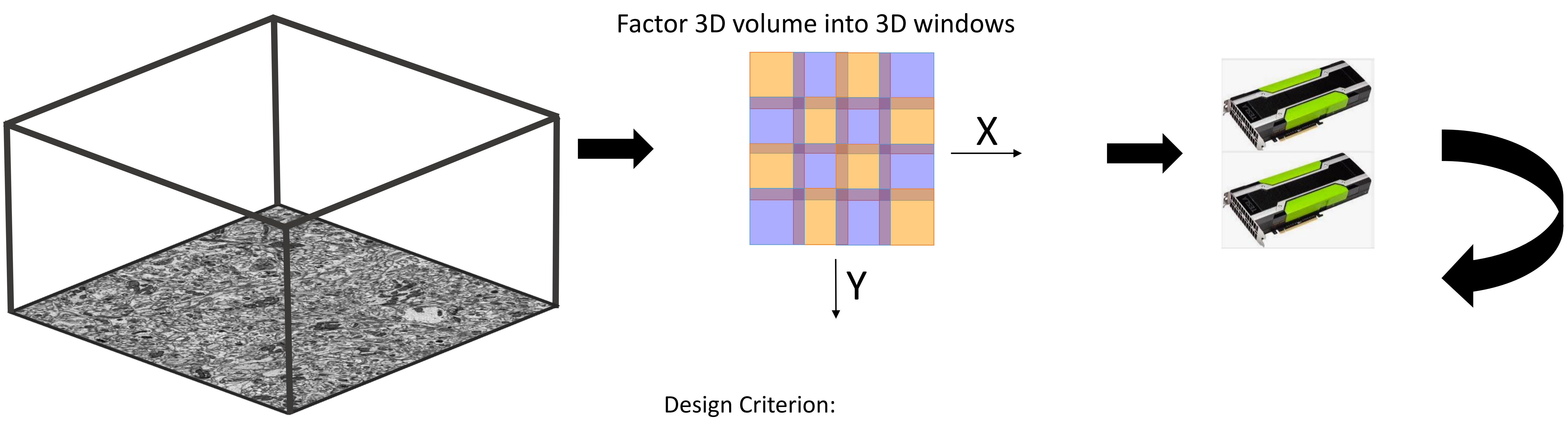
Volume Electron Microscopy A.I. Applications

Labeled Volume Data from Brain Images



Challenges in Application Design

- Defining A parallelization scheme For efficient GPU and CPU utility
- Defining a data structure scheme that is scalable and portable



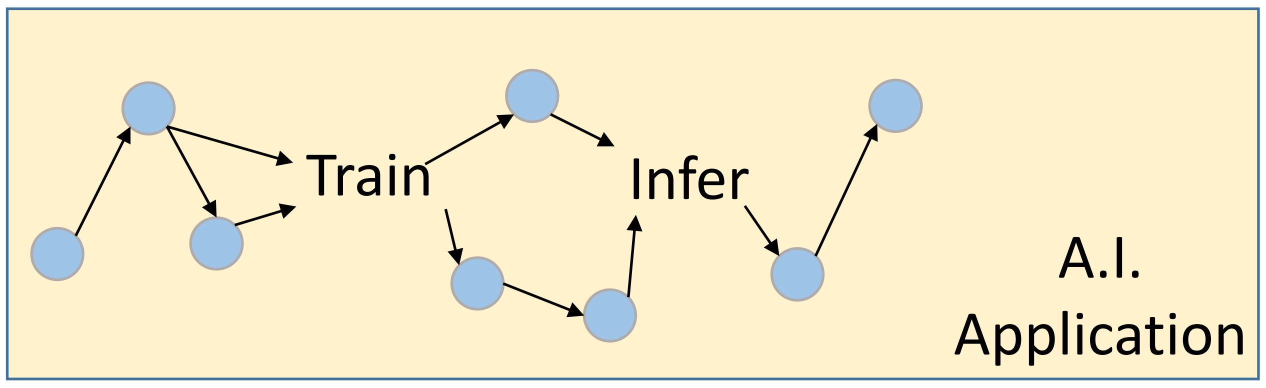
Design Criterion:
3D volume has to be factored into windows (tensors) compatible to GPU specs, while this process must also compress and store data compatible to storage location, (balancing shared filesystem capacity, node storage capacity, compression, inodes)

Scalable Volume ML/GPU Applications must therefore define:
Workers to import and format data, workers to port this data through GPUs, workers to compile outputs back to original volume while dealing with complex multidimensional processing pipelines that wrap around simplified machine-learning experiments.

Solution: Combine a parallel execution engine with a concurrent data structure scheme

Parsl

Build tensor processing applications in parsl apps which can port into any kind of local, cloud, or supercomputer application



Zarr

Define data concurrency and compression schemes that are compatible with the parsl app and A.I. workflow

```
data.n5/tensors/source/0/0/0/0
```

Data File Dataset Data Chunks

Building Scalable volumetric A.I. applications becomes much easier

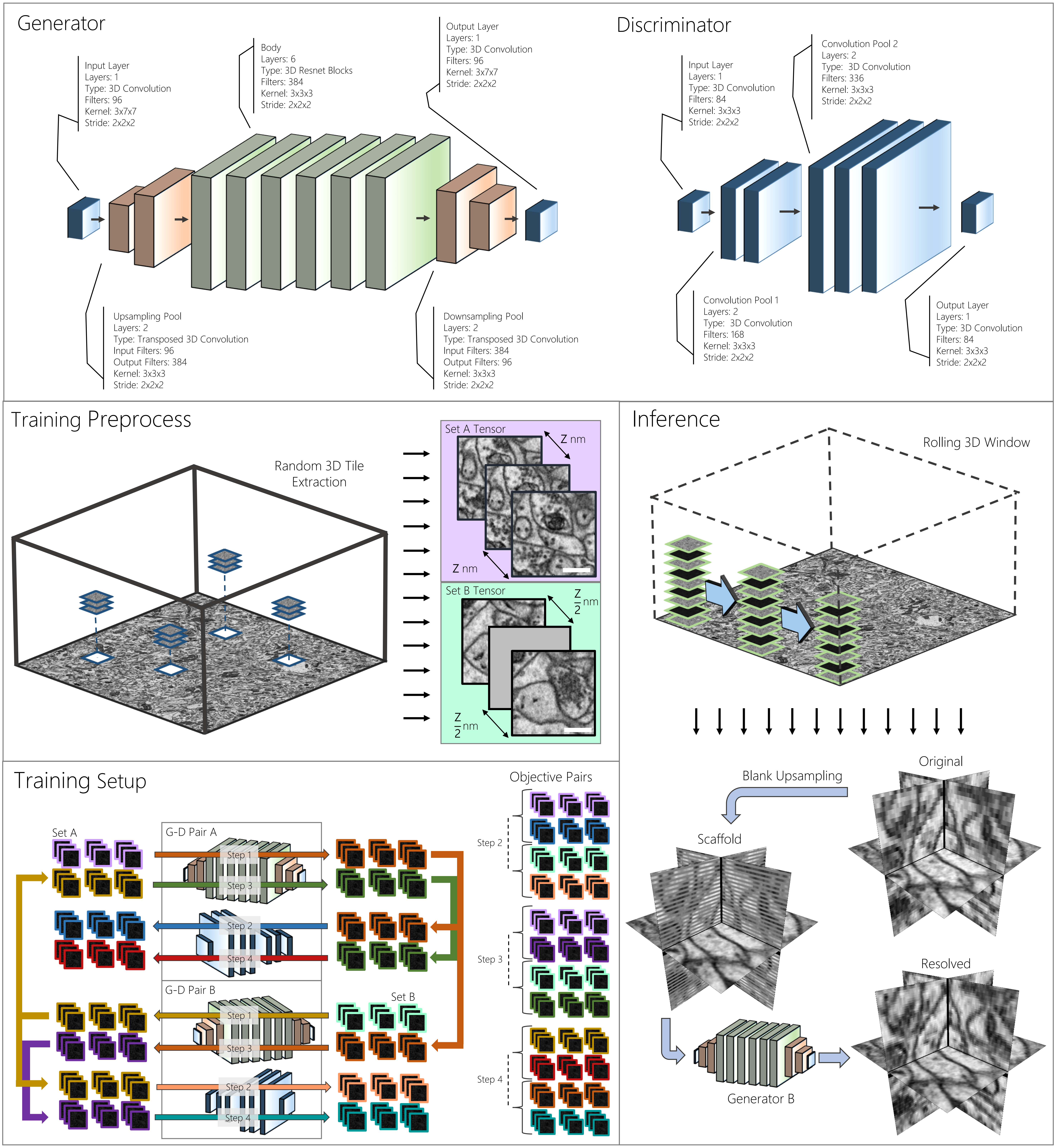
```
from torch.utils.data import DataLoader
import z5py, parsl
#Define Configuration (local, local-ssh-cluster, slurm, lsf, kubernetes)

class N5Dataset(DataLoader):
    def load(self, image_data):
        # Use z5py to define concurracy and compression dataset
        # z5py.create_dataset('image',shape=image shape,chunks=(1,512,512),dtype='uint8', compression options)
        # define parsl applications
        @python_app
        def factor(iteration):
            # Large image data -> tensor field -> single tensor
```

Data achieves very high dimensionalities and efficiencies
Dataset x Batch x Channels x Z-Depth x X-Dimension x Y-Dimension x Tensor-iteration

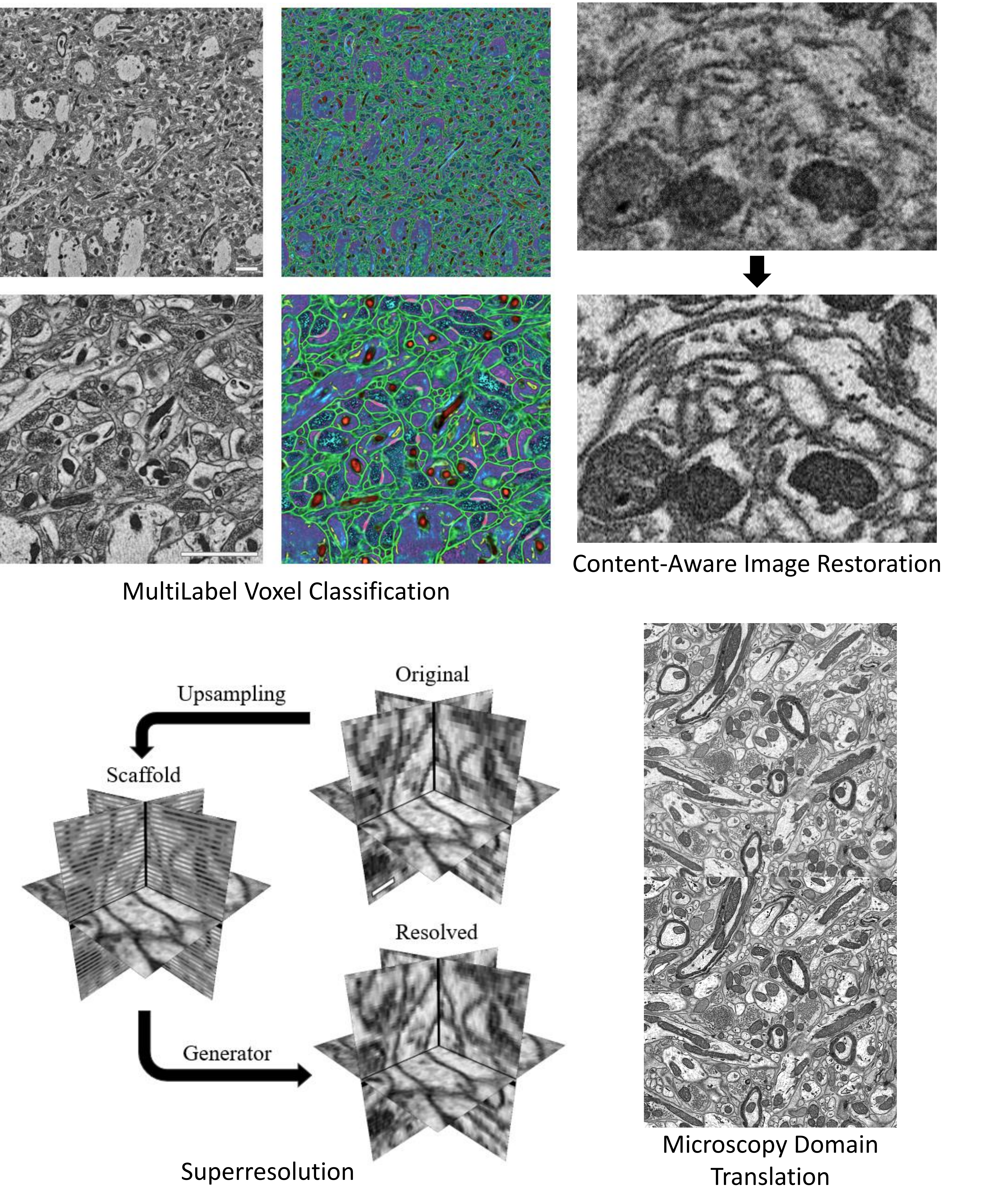
Configuration options passed to Parsl and z5py will naturally load-balance these needs:
- Efficiency in GPU/CPU processing
- Wrangling large data into deep learning frameworks
- Easy design of high-dimensionality processing pipelines
- Data compression cost and benefit
- Porting of applications to heterogonous node definitions

Altogether enable the testing of applications at scale that we wouldn't normally even consider building in the first-place due to parallelization challenges.

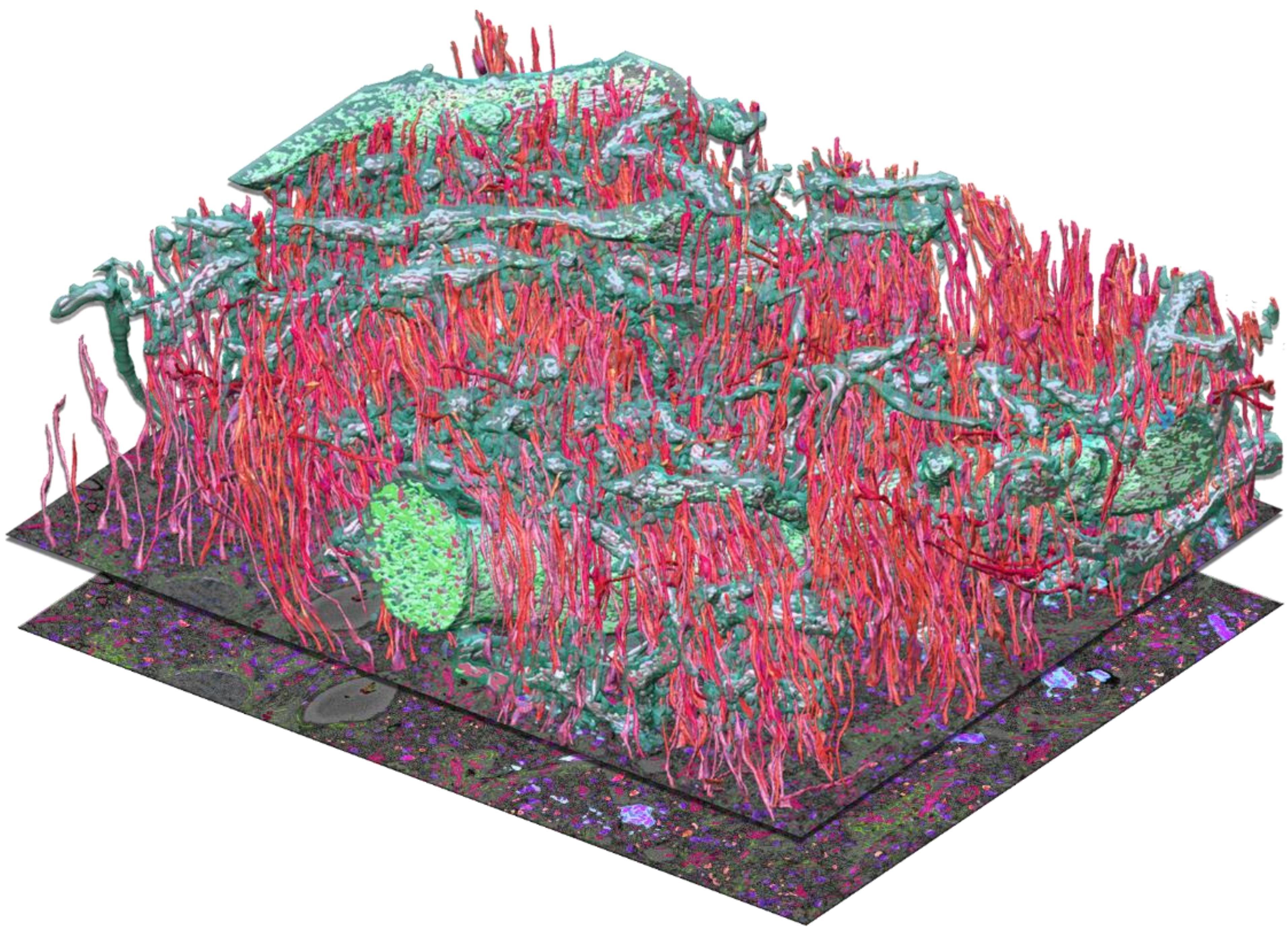
Electron Microscopy Superresolution

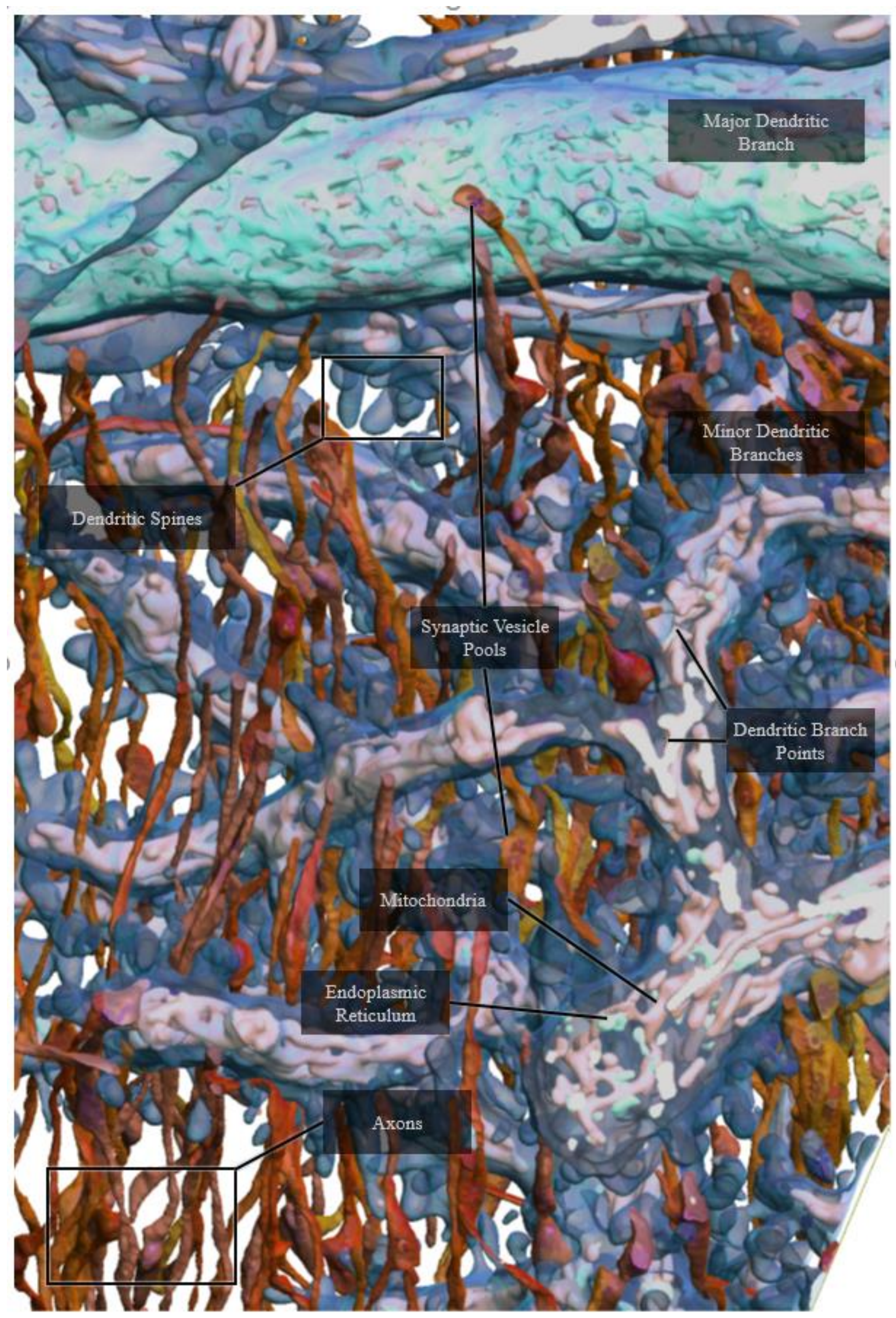


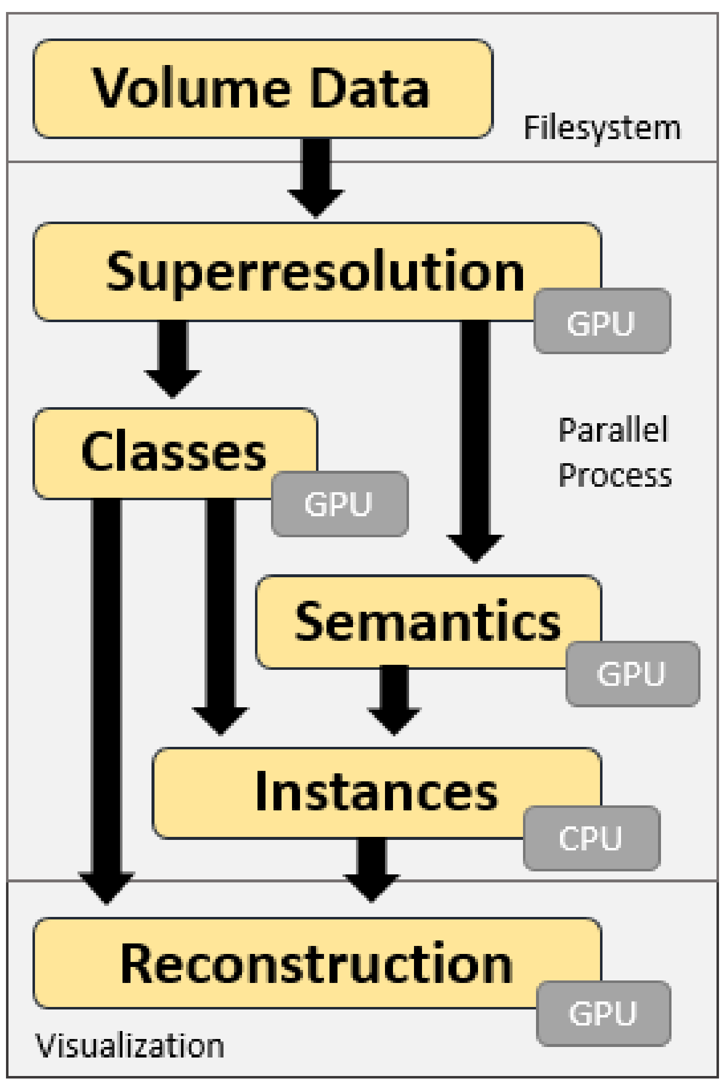
Examples of Applications

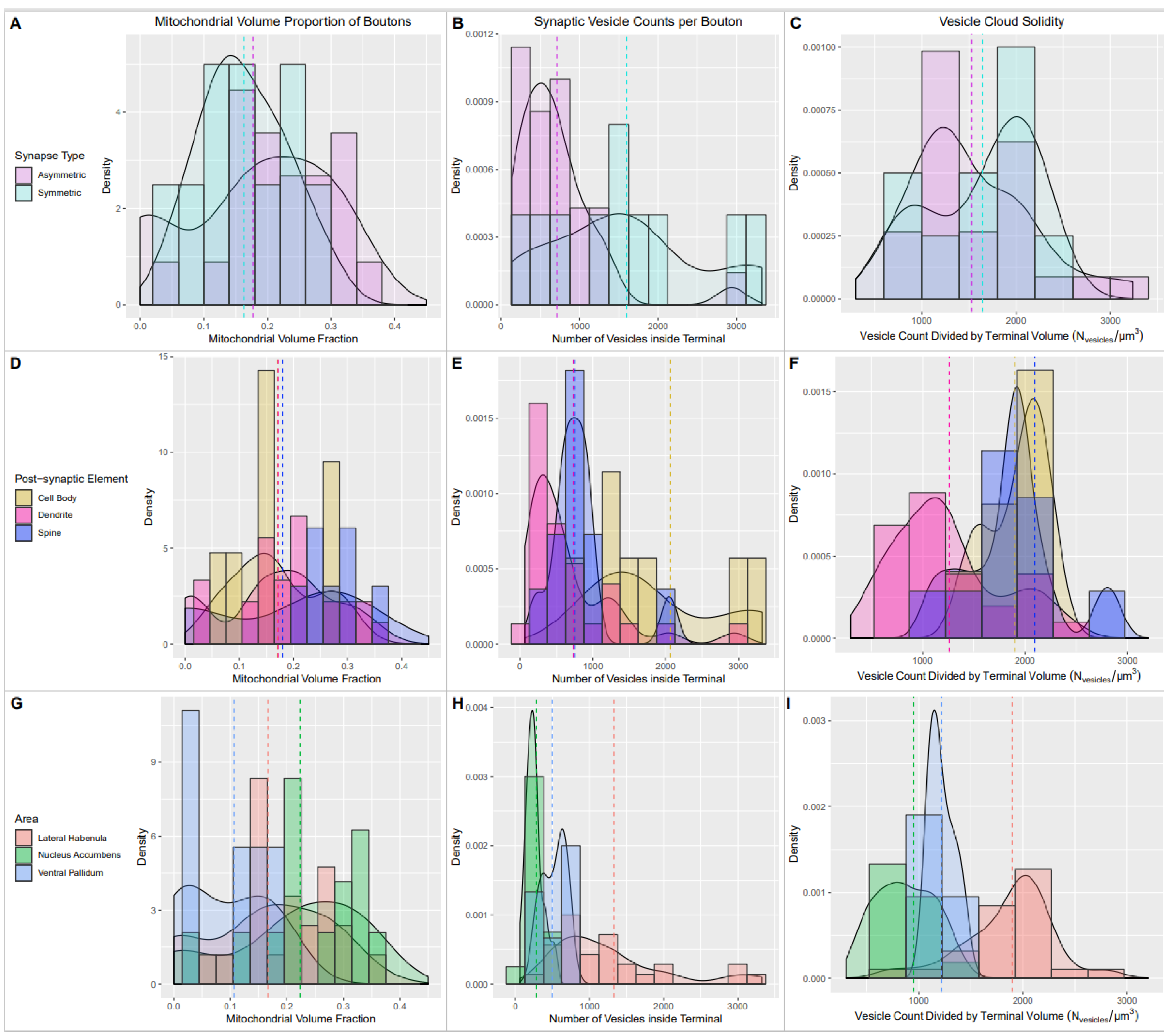


Examples of Reconstructions, Analyses, and Composable Workflows









Performance

CPU / Mem

GPU

kubernetes

PRP PACIFIC RESEARCH PLATFORM

CRBS
CENTER FOR RESEARCH IN
BIOLOGICAL SYSTEMS

NCMIR
National Center for Microscopy and Imaging Research

UC San Diego
SCHOOL OF MEDICINE

SDSC SAN DIEGO
SUPERCOMPUTER CENTER