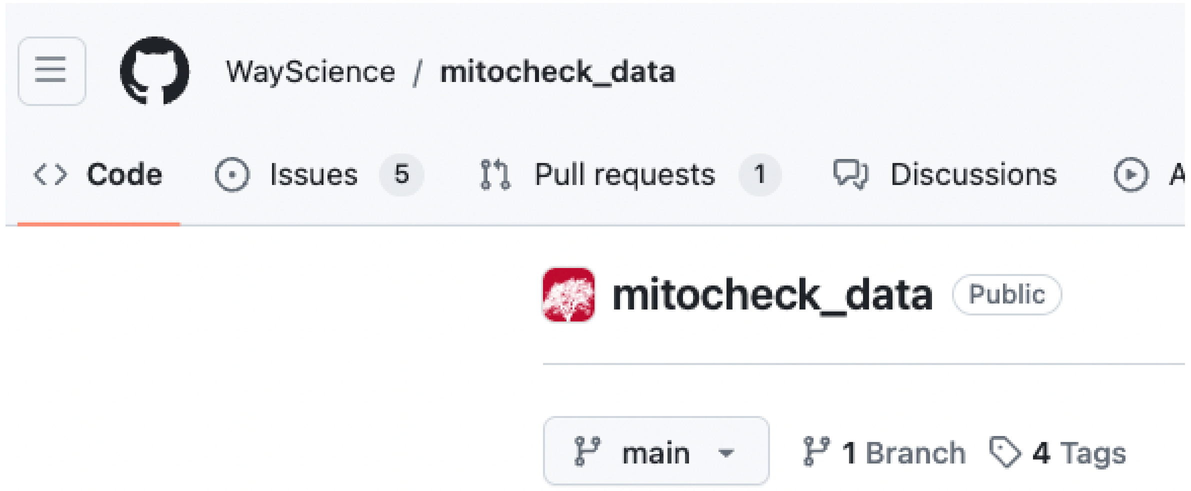
Images and Databases

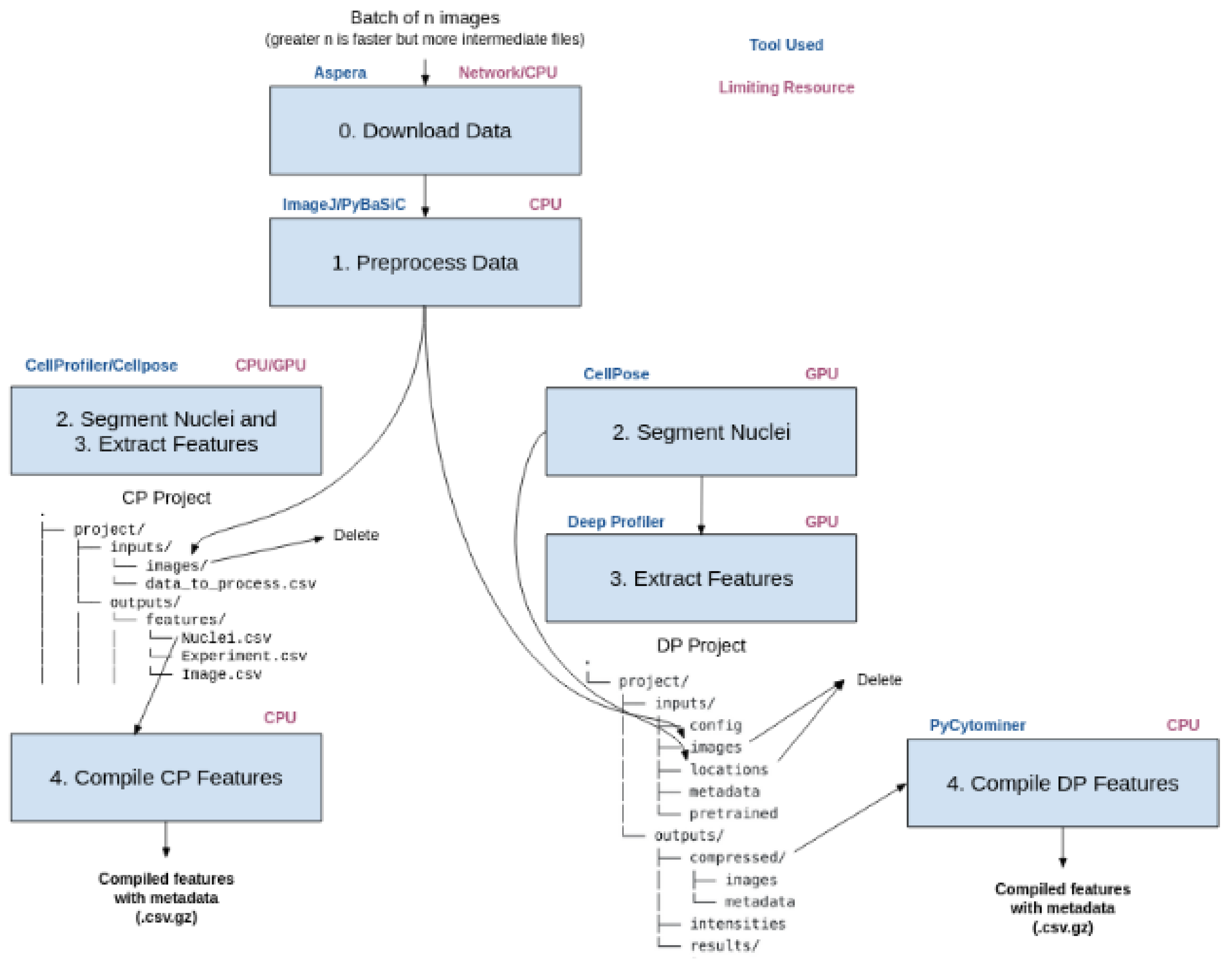
Way Lab - Research in Progress - 2024-06-07

Background



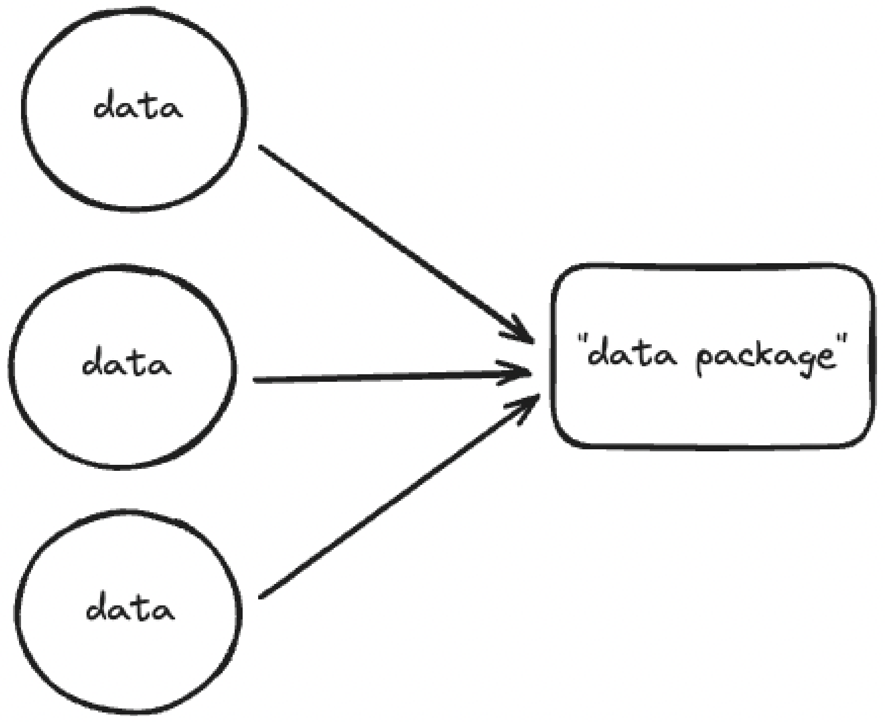
- `mitocheck_data` project

Background



- `idr_stream` project

Background



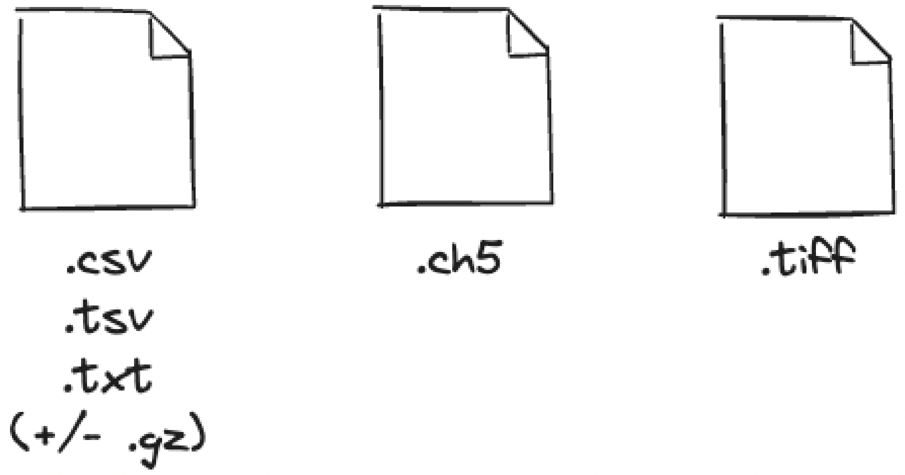
How could we "package" the **`mitocheck_data`** data in such a way to enable development iteration and usefulness to others?

Background

"Data Packaging" Story

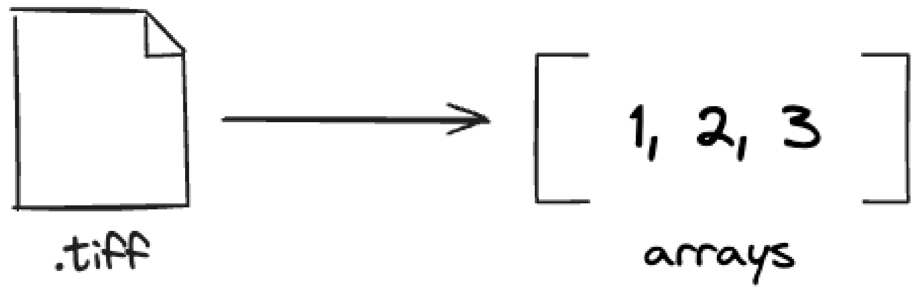
"As a research data participant I need a way to analyze (understand, contextualize, and explore) and implement (engineer solutions which efficiently scale for time and computing resources) the data found here in order to effectively reproduce findings, make new discoveries, and avoid challenging (or perhaps incorrect) translations individually."

Data Files



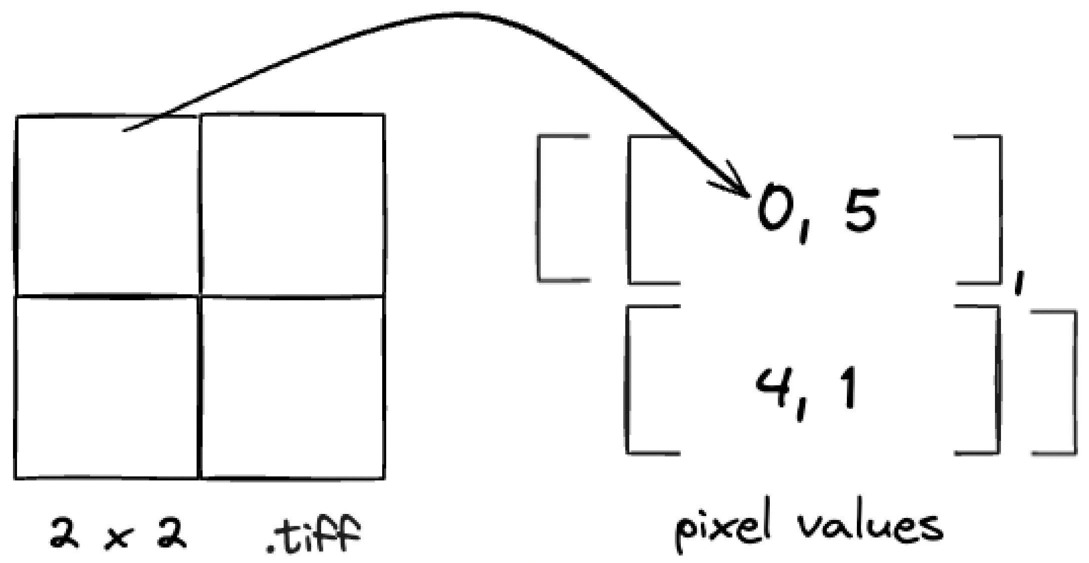
- text file data (CSV's, TSV's, etc)
- ch5 files (microscopy-focused HDF)
- tiff files (tagged image file format)

Implied Data



- arrays from images (for in-memory calculations)

Implied Data

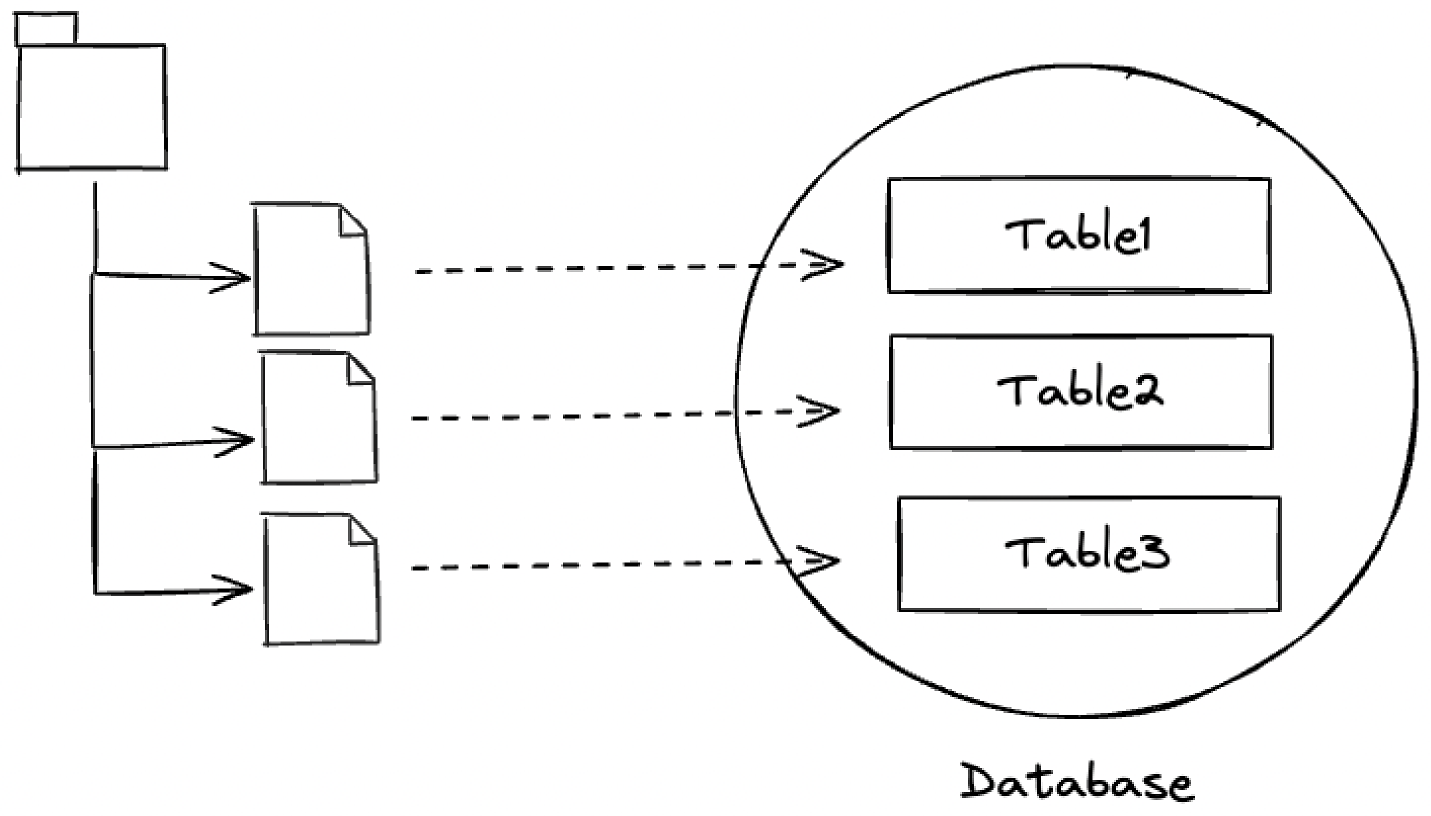


Data package needs

Okay, so we need something that can store and distribute:

- text file tables
- images
- arrays

Files and databases

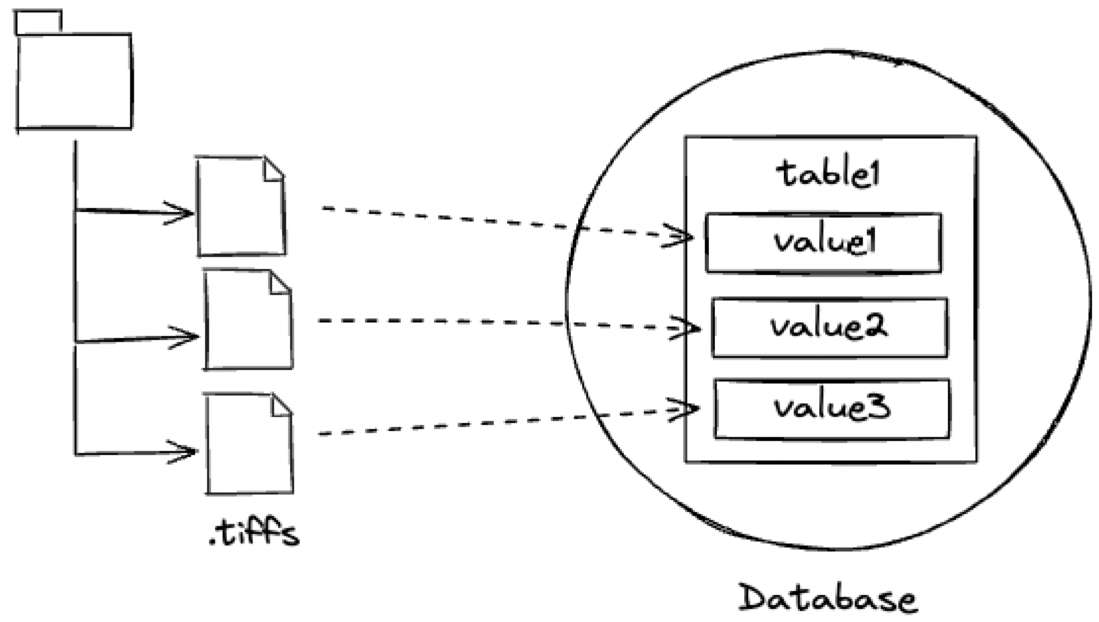


Files and directories follow patterns within databases.

Images and databases

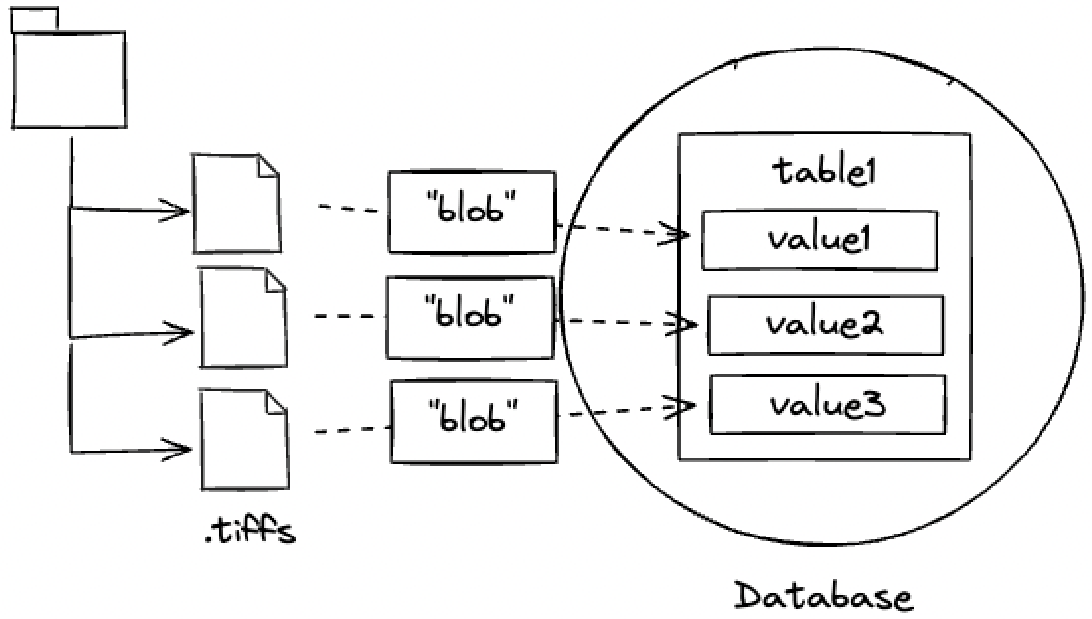
Might be thinking: but images aren't tables!

Images and databases



- Images as *values* within a table.
- The dimensionality is determined by the file.

Images and databases



- When we talk about images this way we can call them "BLOB's" or "objects (object storage)".

Data package needs

Okay, so we need something that can store and distribute:

- files:
  - text file tables
  - images (blobs / objects)
  - arrays
- dimensions:
  - multiple tables
  - multiple values (and dimensions) within tables

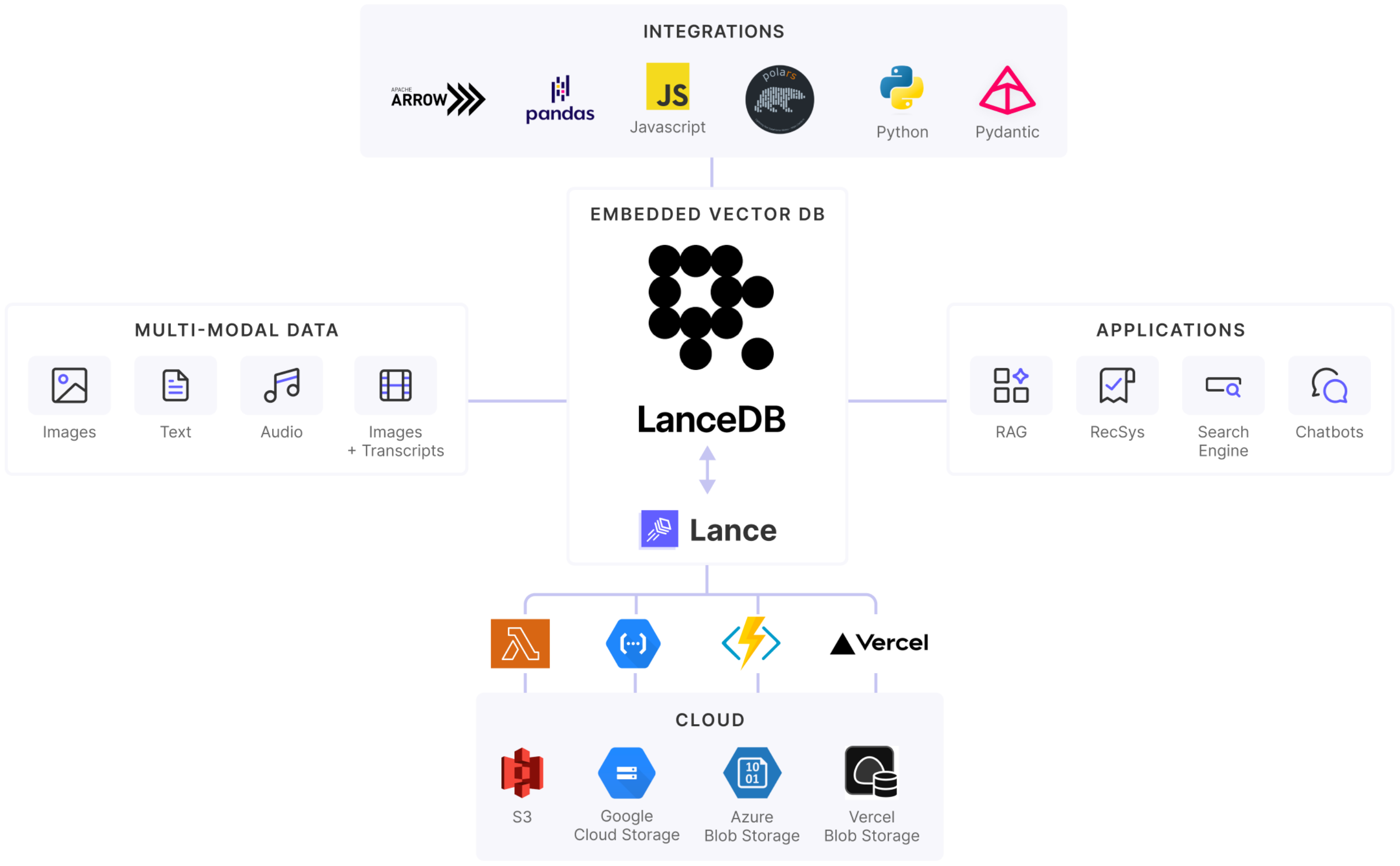
LanceDB

LanceDB

> *LanceDB is an open-source vector database for AI that's designed to store, manage, query and retrieve embeddings on large-scale multi-modal data. The core of LanceDB is written in Rust 🦀 and is built on top of Lance, an open-source columnar data format designed for performant ML workloads and fast random access.*

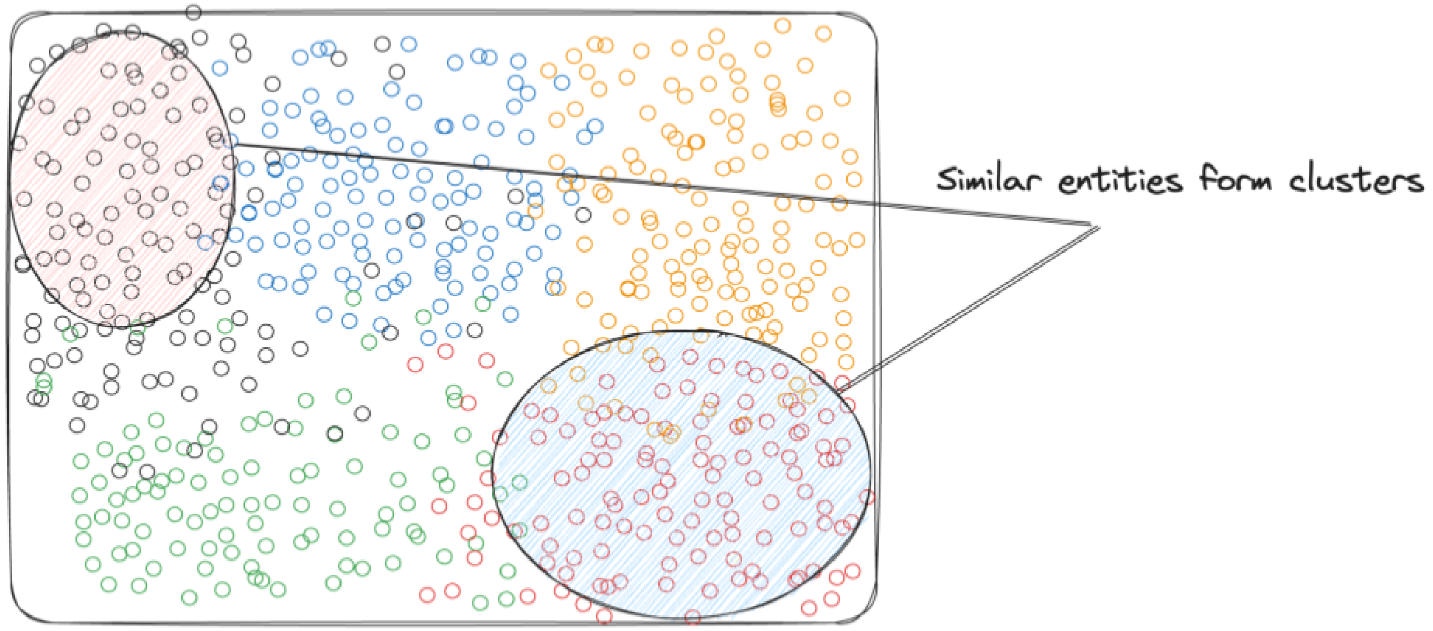
- Source: https://lancedb.github.io/lancedb/

LanceDB



- Source: https://lancedb.github.io/lancedb/

LanceDB

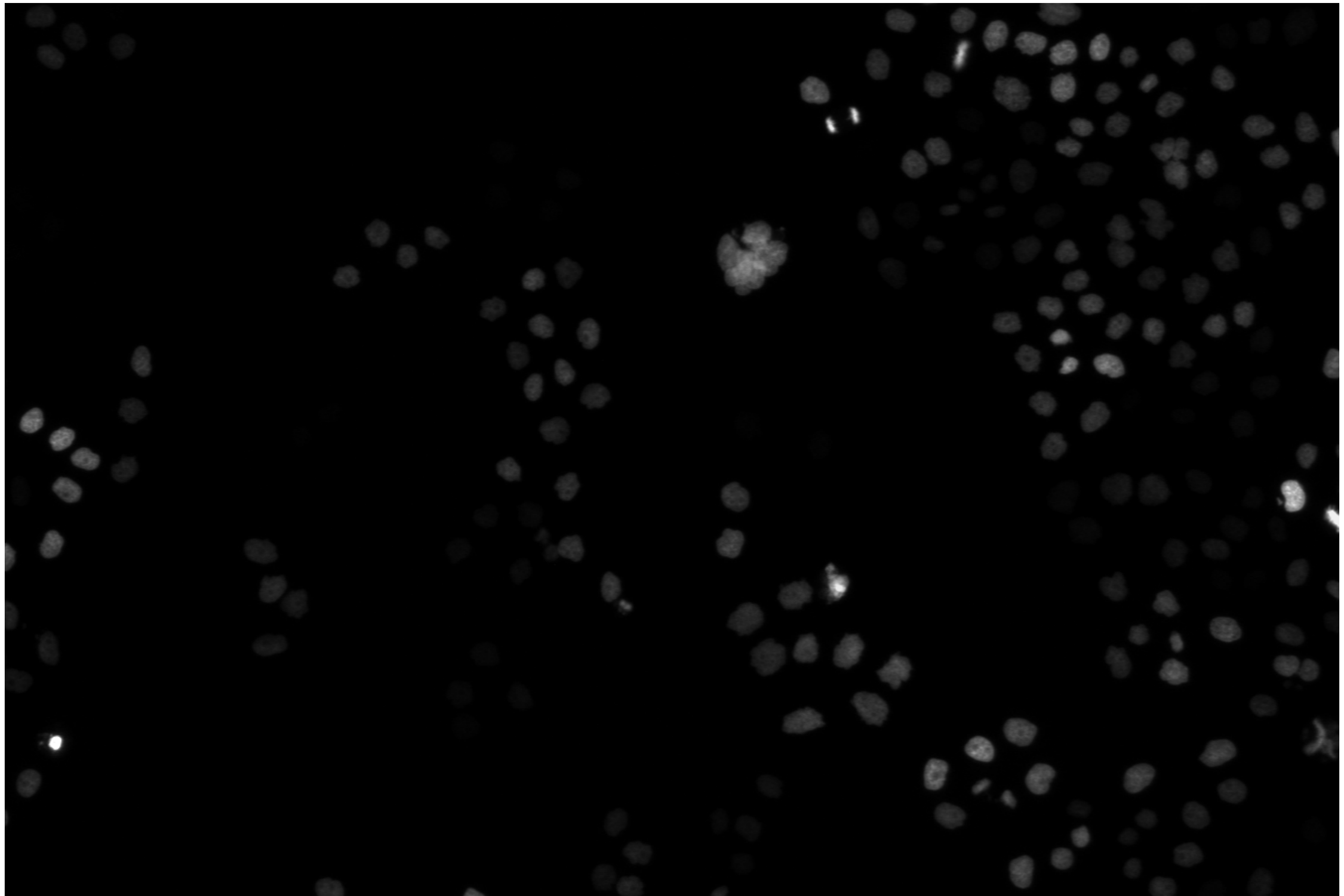


- LanceDB is purpose-built with embeddings and vector search in mind.
- Source: https://lancedb.github.io/lancedb/concepts/vector_search/

```
In [5]: import pathlib

        # images from:
        # mitocheck_data: https://github.com/WayScience/mitocheck_data
        # Image Data Resource (IDR): idr0013(screenA)

        # show some images in an image dir
        image_dir = "mitocheck_example_images"

        # create a list of images using glob on the dir
        images = list(pathlib.Path(image_dir).glob("*"))

        images
```

```
Out[5]: [PosixPath('mitocheck_example_images/LT0001_02.LT0001_02_26_46
        _IC.tif'),
         PosixPath('mitocheck_example_images/LT0001_02.LT0001_02_15_43
        _IC.tif')]
```

In [6]:
```
from IPython.display import display
from PIL import Image

# display the image
display(Image.open(images[0]))
```



In [7]:
```python
from IPython.display import display
from PIL import Image

# display the image
display(Image.open(images[1]))
```

In [8]:

```python
# file size in bytes
print(images[0])
print(images[0].stat().st_size)
```

```
mitocheck_example_images/LT0001_02.LT0001_02_26_46_IC.tif
1376512
```

```
In [9]: # show first few bytes as byte string
        with open(images[0], "rb") as f:
            print(f.read(10))
```

```
b'II*\x00\x08\x00\x00\x00\x0e\x00'
```

```
In [10]: # show some metadata associated with the image
         !tifffile mitocheck_example_images/LT0001_02.LT0001_02_15_43_IC.tif
```

```
Reading TIFF header: 0.000340 s
Reading image data: 0.000260 s
Generating report:   0.001179 s

TiffFile 'LT0001_02.LT0001_02_15_43_IC.tif'  1344.25 KiB  shape
d

TiffPageSeries 0  1024x1344  uint8  YX  shaped  1 Pages  @256

TiffPage 0 @8  1024x1344  uint8  minisblack memmappable  shaped

TiffTag 256 ImageWidth @10 LONG @18 = 1344
TiffTag 257 ImageLength @22 LONG @30 = 1024
TiffTag 258 BitsPerSample @34 SHORT @42 = 8
TiffTag 259 Compression @46 SHORT @54 = NONE
TiffTag 262 PhotometricInterpretation @58 SHORT @66 = MINISBLAC
K
TiffTag 270 ImageDescription @70 ASCII[24] @182 = {"shape": [10
24, 1344]}
TiffTag 273 StripOffsets @82 LONG @90 = (256,)
TiffTag 277 SamplesPerPixel @94 SHORT @102 = 1
TiffTag 278 RowsPerStrip @106 LONG @114 = 1024
TiffTag 279 StripByteCounts @118 LONG @126 = (1376256,)
TiffTag 282 XResolution @130 RATIONAL @222 = (1, 1)
TiffTag 283 YResolution @142 RATIONAL @230 = (1, 1)
TiffTag 296 ResolutionUnit @154 SHORT @162 = NONE
TiffTag 305 Software @166 ASCII[12] @238 = tifffile.py
```

```
In [11]: import skimage

         # read the image as an array
         array = skimage.io.imread(images[0])
         print(array.shape)
         print(array)
```

```
(1024, 1344)
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 1 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
```

```
In [12]: import pandas as pd

         # create a pandas dataframe with image paths
         df = pd.DataFrame({"path": images})

         df
```

```
Out[12]:
                                                 path
   0   mitocheck_example_images/LT0001_02.LT0001_02_2...
   1   mitocheck_example_images/LT0001_02.LT0001_02_1...
```

In [13]:
```
# add the filesize bytes as a column to the dataframe
df["filesize_bytes"] = df.apply(lambda row: row["path"].stat().st_size,
df
```

Out[13]:

	path	filesize_bytes
0	mitocheck_example_images/LT0001_02.LT0001_02_2...	1376512
1	mitocheck_example_images/LT0001_02.LT0001_02_1...	1376512

In [14]:

```python
def read_image_bytes(image_path):
    with open(image_path, "rb") as f:
        return f.read()


# read the image as a bytearray object and store in dataframe
df["image_bytes"] = df.apply(lambda row: read_image_bytes(row["path"]),
df
```

Out[14]:

	path	filesize_bytes	
0	mitocheck_example_images/LT0001_02.LT0001_02_2...	1376512	b'II*\x00\x0
1	mitocheck_example_images/LT0001_02.LT0001_02_1...	1376512	b'II*\x00\x0

```
In [15]: from io import BytesIO

         import skimage

         # read the bytearray from the dataframe as an array in new column
         df["image_array"] = df.apply(
             lambda row: skimage.io.imread(BytesIO(row["image_bytes"])), axis=1
         )
         df
```

```
Out[15]:
                                                   path   filesize_bytes
   0   mitocheck_example_images/LT0001_02.LT0001_02_2...        1376512   b'II*\x00\x0
   1   mitocheck_example_images/LT0001_02.LT0001_02_1...        1376512   b'II*\x00\x0
```

```
In [17]: # show images within the dataframe output
         ImageDataFrame(df[["path", "filesize_bytes", "image_bytes"]])
```

	path	filesize_bytes	image
0	mitocheck_example_images/LT0001_02.LT0001_02_26_46_IC.tif	1376512	
1	mitocheck_example_images/LT0001_02.LT0001_02_15_43_IC.tif	1376512	

```
In [18]:
# try to write to parquet
df.to_parquet("mitocheck_example_images.parquet")
```

```
---------------------------------------------------------------------------
ArrowInvalid                              Traceback (most recent call last)
Cell In[18], line 1
----> 1 df.to_parquet("mitocheck_example_images.parquet")

File ~/Library/Caches/pypoetry/virtualenvs/2024-06-07-images-and-databases-jUUoCL3p-py3.11/lib/python3.11/site-packages/pandas/util/_decorators.py:333, in deprecate_nonkeyword_arguments.<locals>.decorate.<locals>.wrapper(*args, **kwargs)
    327 if len(args) > num_allow_args:
    328     warnings.warn(
    329         msg.format(arguments=_format_argument_list(allow_args)),
    330         FutureWarning,
    331         stacklevel=find_stack_level(),
    332     )
--> 333 return func(*args, **kwargs)

File ~/Library/Caches/pypoetry/virtualenvs/2024-06-07-images-and-databases-jUUoCL3p-py3.11/lib/python3.11/site-packages/pandas/core/frame.py:3113, in DataFrame.to_parquet(self, path, engine, compression, index, partition_cols, storage_options, **kwargs)
   3032 """
   3033 Write a DataFrame to the binary parquet format.
```

```
In [19]: # show columns and types
         df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2 entries, 0 to 1
Data columns (total 4 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   path            2 non-null      object
 1   filesize_bytes  2 non-null      int64 
 2   image_bytes     2 non-null      object
 3   image_array     2 non-null      object
dtypes: int64(1), object(3)
memory usage: 196.0+ bytes
```

```
In [20]: # show the type of a single path value
         type(df["path"].iloc[0])

Out[20]: pathlib.PosixPath
```

In [21]:
```python
import pyarrow as pa

# update the paths to be strings, then try conversion again
df["path"] = df.apply(lambda row: str(row["path"]), axis=1)
pa.Table.from_pandas(df)
```

```
---------------------------------------------------------------------------
ArrowInvalid                              Traceback (most recent call last)
Cell In[21], line 5
      3 # update the paths to be strings, then try conversion again
      4 df["path"] = df.apply(lambda row: str(row["path"]), axis=1)
----> 5 pa.Table.from_pandas(df)

File ~/Library/Caches/pypoetry/virtualenvs/2024-06-07-images-and-databases-jUUoCL3p-py3.11/lib/python3.11/site-packages/pyarrow/table.pxi:3874, in pyarrow.lib.Table.from_pandas()

File ~/Library/Caches/pypoetry/virtualenvs/2024-06-07-images-and-databases-jUUoCL3p-py3.11/lib/python3.11/site-packages/pyarrow/pandas_compat.py:611, in dataframe_to_arrays(df, schema, preserve_index, nthreads, columns, safe)
    606     return (isinstance(arr, np.ndarray) and
    607             arr.flags.contiguous and
    608             issubclass(arr.dtype.type, np.integer))
    610 if nthreads == 1:
--> 611     arrays = [convert_column(c, f)
```

```
In [22]: # show the type of a single image_array value
         type(df["image_array"].iloc[0])

Out[22]: numpy.ndarray
```

```
In [23]: import awkward as ak

         # use awkward array to interpret the nested arrays from dict records
         awk_arr = ak.Array(df.to_dict(orient="records"))
         awk_arr
```

```
Out[23]: [{path: 'mitocheck_example_images/LT0001_02.LT0001_02_26_46_IC.
         tif', ...},
          {path: 'mitocheck_example_images/LT0001_02.LT0001_02_15_43_IC.
         tif', ...}]
         ---------------------------------------------------------------
         -----------
         type: 2 * {
             path: string,
             filesize_bytes: int64,
             image_bytes: bytes,
             image_array: var * var * int64
         }
```

```
In [24]: from pyarrow import parquet

         # write a parquet table from the awkward array
         parquet.write_table(
             table=ak.to_arrow_table(awk_arr), where="mitocheck_example_images.p
         )
         # show that we have a file
         pathlib.Path("mitocheck_example_images.parquet").is_file()

Out[24]: True
```

```
In [25]: # read the file as a dataframe
         df = pd.read_parquet(path="mitocheck_example_images.parquet")
         # show that it's the same
         ImageDataFrame(df[["path", "filesize_bytes", "image_bytes"]])
```

	path	filesize_bytes	image
0	mitocheck_example_images/LT0001_02.LT0001_02_26_46_IC.tif	1376512	
1	mitocheck_example_images/LT0001_02.LT0001_02_15_43_IC.tif	1376512	

In [26]:
```python
import shutil

import lancedb
from pyarrow import parquet

# remove any earlier work
shutil.rmtree("mitocheck_example_images.lance")

# specify a dir where the lancedb database may go and create lancedb cl
lancedb_dir = pathlib.Path("mitocheck_example_images.lance")
ldb = lancedb.connect(lancedb_dir)

# create a lancedb table from the parquet data
ldb.create_table(
    data=parquet.read_table("mitocheck_example_images.parquet"),
    name="mitocheck_example_images",
)
```

```
/Users/dabu5788/Library/Caches/pypoetry/virtualenvs/2024-06-07-
images-and-databases-jUUoCL3p-py3.11/lib/python3.11/site-packag
es/tqdm/auto.py:21: TqdmWarning: IProgress not found. Please up
date jupyter and ipywidgets. See https://ipywidgets.readthedoc
s.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm

---------------------------------------------------------------
------------
ArrowNotImplementedError                  Traceback (most recen
t call last)
Cell In[26], line 14
```

```
In [36]: # show the parquet file schema
         parquet.read_table("mitocheck_example_images.parquet").schema
```

```
Out[36]:   path: extension<awkward<AwkwardArrowType>> not null
           filesize_bytes: extension<awkward<AwkwardArrowType>> not null
           image_bytes: extension<awkward<AwkwardArrowType>> not null
           image_array: extension<awkward<AwkwardArrowType>> not null
           -- schema metadata --
           ak:parameters: '[{"optiontype_fields": []}, {"record_is_scala
           r": false}, ' + 22
```

```
In [27]: import lancedb
         from pyarrow import parquet

         # specify a dir where the lancedb database may go and create lancedb cl
         lancedb_dir = pathlib.Path("mitocheck_example_images.lance")
         ldb = lancedb.connect(lancedb_dir)

         # create the table from pandas via parquet file
         ldb.create_table(
             data=pd.read_parquet("mitocheck_example_images.parquet"),
             name="mitocheck_example_images",
             mode="overwrite",
         )
```

```
[2024-06-07T17:17:33Z WARN  lance::dataset] No existing dataset
at /Users/dabu5788/Documents/work/set-presentations/2024-06-07-
images-and-databases/src/notebooks/mitocheck_example_images.lan
ce/mitocheck_example_images.lance, it will be created
```

```
Out[27]: LanceTable(connection=LanceDBConnection(/Users/dabu5788/Docume
         nts/work/set-presentations/2024-06-07-images-and-databases/sr
         c/notebooks/mitocheck_example_images.lance), name="mitocheck_e
         xample_images")
```

```
In [28]: # read from lancedb to pandas
         ldb.open_table("mitocheck_example_images").to_pandas()
```

```
Out[28]:
                    path  filesize_bytes          image_bytes        image_array
   0  mitocheck_exampl...         1376512  b'II*\x00\x08\x0...  [[0, 0, 0, 0, 0,...
   1  mitocheck_exampl...         1376512  b'II*\x00\x08\x0...  [[0, 0, 0, 0, 0,...
```

```
In [29]: # show that the dataframes are equal
         pd.testing.assert_frame_equal(
             pd.read_parquet("mitocheck_example_images.parquet"),
             ldb.open_table("mitocheck_example_images").to_pandas(),
         )
```

```
In [30]: %%timeit

         # time pd read_parquet
         pd.read_parquet(path="mitocheck_example_images.parquet")
```

```
24.9 ms ± 654 µs per loop (mean ± std. dev. of 7 runs, 10 loops
each)
```

```
In [31]: %%timeit

         # time pyarrow parquet read
         parquet.read_table(source="mitocheck_example_images.parquet")
```

```
23.4 ms ± 454 µs per loop (mean ± std. dev. of 7 runs, 10 loops
each)
```

```
In [32]: %%timeit

         # time lancedb read
         ldb.open_table("mitocheck_example_images").to_pandas()
```

```
8.13 ms ± 293 µs per loop (mean ± std. dev. of 7 runs, 100 loop
s each)
```

```
In [33]: # show how to add data to an existing table
         ldb.open_table("mitocheck_example_images").add(
             pd.read_parquet(filename := "mitocheck_example_images.parquet")
         )
         ldb.open_table("mitocheck_example_images").to_pandas()
```

Out[33]:

	path	filesize_bytes	image_bytes	image_array
0	mitocheck_exampl...	1376512	b'II*\x00\x08\x0...	[[0, 0, 0, 0, 0,...
1	mitocheck_exampl...	1376512	b'II*\x00\x08\x0...	[[0, 0, 0, 0, 0,...
2	mitocheck_exampl...	1376512	b'II*\x00\x08\x0...	[[0, 0, 0, 0, 0,...
3	mitocheck_exampl...	1376512	b'II*\x00\x08\x0...	[[0, 0, 0, 0, 0,...

```
In [34]: # show version of the table
         ldb.open_table("mitocheck_example_images").version

Out[34]: 3
```

```
In [35]: # show a change to original version
         ldb.open_table("mitocheck_example_images").checkout(version=1)
         ldb.open_table("mitocheck_example_images").to_pandas()
```

Out[35]:

	path	filesize_bytes	image_bytes	image_array
0	mitocheck_exampl...	1376512	b'II*\x00\x08\x0...	[[0, 0, 0, 0, 0,...
1	mitocheck_exampl...	1376512	b'II*\x00\x08\x0...	[[0, 0, 0, 0, 0,...
2	mitocheck_exampl...	1376512	b'II*\x00\x08\x0...	[[0, 0, 0, 0, 0,...
3	mitocheck_exampl...	1376512	b'II*\x00\x08\x0...	[[0, 0, 0, 0, 0,...

Reflections

- Images can be treated as values through objects in a database table.
- LanceDB seems like a good option for storing multiple tables together as a "package".
- LanceDB integrates well with Parquet, Arrow, and Pandas.
- LanceDB feels fast!

Thank you for attending!

Questions / comments?