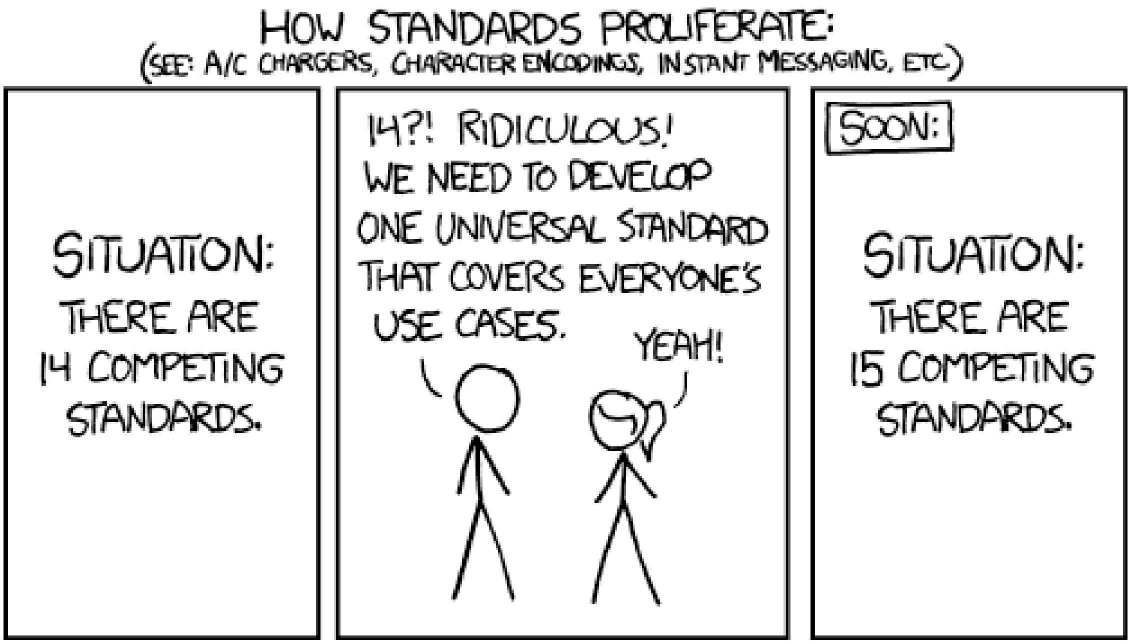
Combining TIFF, HDF5, and Zarr into a Single Image File Format

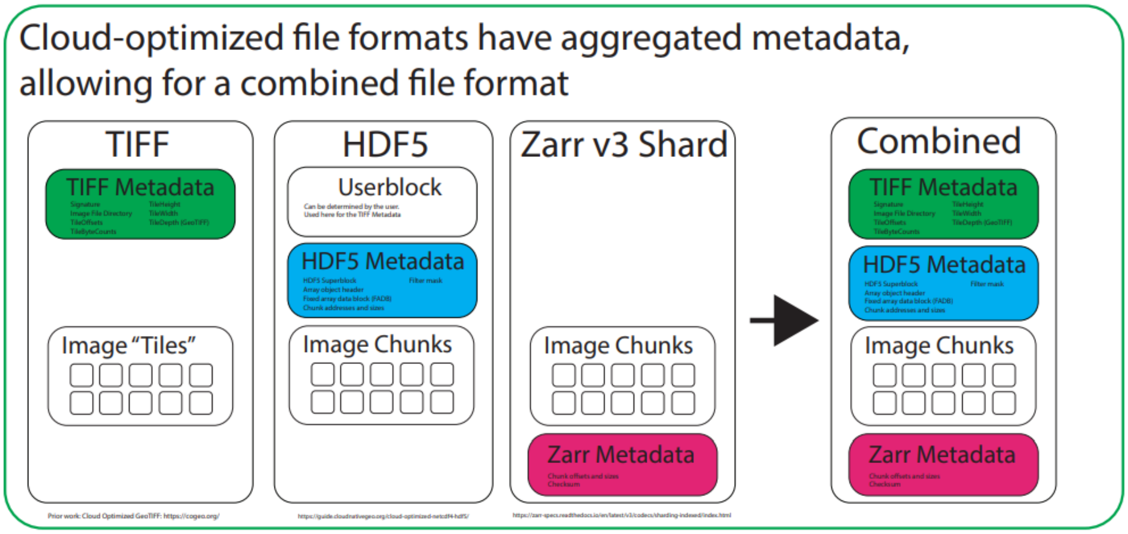
Mark Kittisopikul

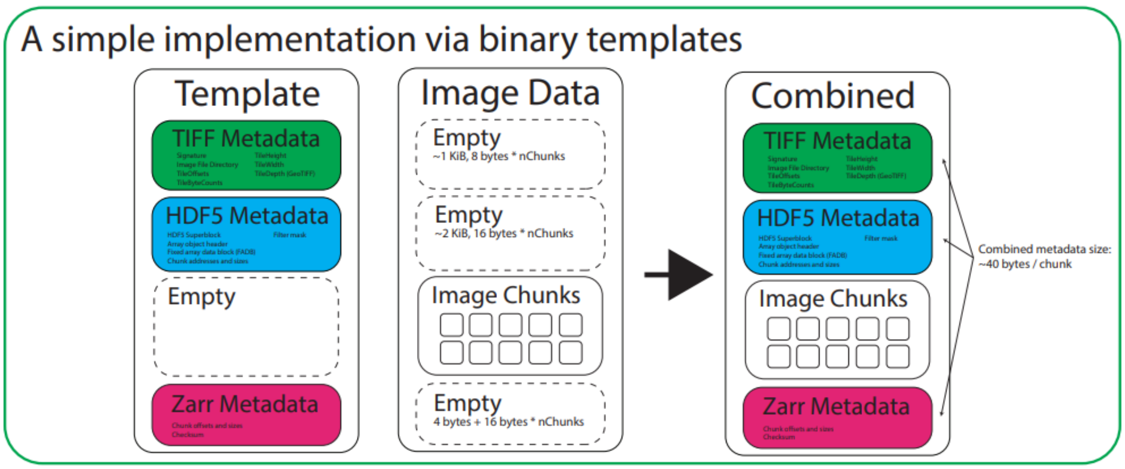
October 3, 2025

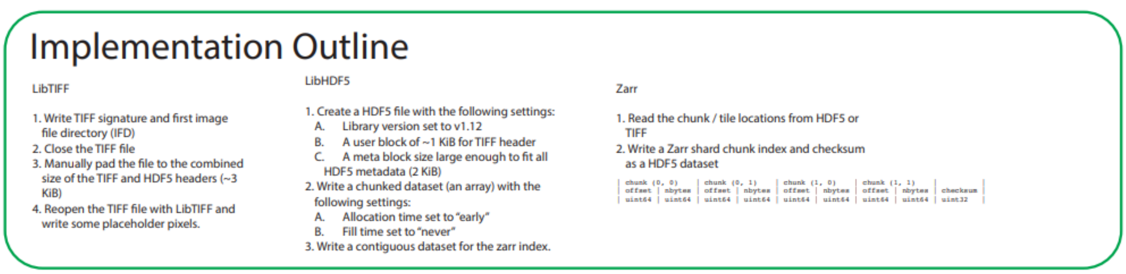
1 Abstract



"Situation: There are 3 competing standards" the last card of a popular XKCD cartoon (#927) might read if applied to recent microscopy bioimaging formats. TIFF, HDF5, and Zarr have all been used to store images as part of popular standards and formats (OME-TIFF, BDV-HDF5, OME-Zarr). The cartoon humorously points out the tendency to create new standards while discounting prior efforts. To combat the proliferation of formats I examine similarities between TIFF, HDF5, and Zarr shard containers. I then exploit them to create a combined data container that is simultaneously a TIFF file, a HDF5 file, and a Zarr version 3 shard without duplicating the image pixel or volumetric voxel data. This combined format is compatible with multiple viewers and image analysis pipelines. Additionally, the techniques involved provide a path to convert between the formats with minimal processing or overhead. In practice, the combined format avoids redundant copies of data while providing great utility to the microscope user. The combined format is a great candidate for a microscope acquisition format as it satisfies both short term needs to view microscope output in traditional viewers while integrating into next generation image analysis pipelines.







1.1 1. Write the Combined File Format

[1]:
```python
import header_formats
```

[2]:
```python
import os
import shutil
import numpy as np
def run_demo():
    # create a template file
```

```python
    header_formats.tiff_hdf5_zarr("test.tiff.hdf5.zarr", (256,256), "uint16",␣
↪chunks=(128,128))
    header_formats.create_zarr3("demo/test.zarr")

    # extract header and footer from template
    header, footer = header_formats.read_header_footer("test.tiff.hdf5.zarr")

    # create chunks
    A = np.full((128, 128), 0, dtype="uint16")
    B = np.full((128, 128), 2**14-2, dtype="uint16")
    C = np.full((128, 128), 2*2**14-2, dtype="uint16")
    D = np.full((128, 128), 3*2**14-2, dtype="uint16")

    os.makedirs("demo/test.zarr/c/0/")

    # write header, tiles and footer to demo file
    with open("demo/demo.hdf5.zarr.tiff", "wb") as f:
        f.write(header)
        f.write(A)
        f.write(B)
        f.write(C)
        f.write(D)
        f.write(footer)

    # copy one file to many files with different file extensions
    # TIFF
    shutil.copyfile("demo/demo.hdf5.zarr.tiff", "demo/demo.tiff")
    # HDF5
    shutil.copyfile("demo/demo.hdf5.zarr.tiff", "demo/demo.h5")
    # Zarr v3 shard
    shutil.copyfile("demo/demo.hdf5.zarr.tiff", "demo/test.zarr/c/0/0")
```

[3]:
```python
run_demo()
```

```
MissingRequired: TIFF directory is missing required "TileOffsets" field.
MissingRequired: TIFF directory is missing required "TileOffsets" field.
```

1.2 2. Check the HDF5 and TIFF File Structure

[4]:
```
!pixi run h5ls -va demo/demo.hdf5.zarr.tiff
```

```
Opened "demo/demo.hdf5.zarr.tiff" with sec2 driver.
data                         Dataset {256/256, 256/256}
    Location:  1:195
    Links:     1
    Modified:  2025-10-03 17:39:17 EDT
    Chunks:    {128, 128} 32768 bytes
    Storage:   131072 logical bytes, 131072 allocated bytes, 100.00% utilization
```

```
    Type:      native unsigned short
    Address: 2048
           Flags    Bytes     Address          Logical Offset
        ========== ======== ========== ==============================
        0x00000000    32768       2048 [0, 0, 0]
        0x00000000    32768      34816 [0, 128, 0]
        0x00000000    32768      67584 [128, 0, 0]
        0x00000000    32768     100352 [128, 128, 0]
zarrindex                Dataset {68/68}
    Location:  1:479
    Links:     1
    Storage:   68 logical bytes, 68 allocated bytes, 100.00% utilization
    Type:      native unsigned char
    Address:   133120
```

[5]:
```
!pixi run tiffinfo -s demo/demo.hdf5.zarr.tiff
```

```
=== TIFF directory 0 ===vironment
TIFF Directory at offset 0x86 (134)
  Image Width: 256 Image Length: 256
  Tile Width: 128 Tile Length: 128
  Bits/Sample: 16
  Sample Format: unsigned integer
  Compression Scheme: None
  Photometric Interpretation: min-is-black
  Orientation: row 0 top, col 0 lhs
  Planar Configuration: single image plane
  4 Tiles:
      0: [     3072,     32768]
      1: [    35840,     32768]
      2: [    68608,     32768]
      3: [   101376,     32768]
```

1.3 3. Read the Data using h5py, libtiff, and tensorstore

[6]:
```python
import h5py
with h5py.File("demo/demo.hdf5.zarr.tiff") as h5f:
    h5data = h5f["data"][:]

h5data
```

[6]:
```
array([[    0,     0,     0, …, 16382, 16382, 16382],
       [    0,     0,     0, …, 16382, 16382, 16382],
       [    0,     0,     0, …, 16382, 16382, 16382],
       …,
       [32766, 32766, 32766, …, 49150, 49150, 49150],
       [32766, 32766, 32766, …, 49150, 49150, 49150],
```

```
       [32766, 32766, 32766, …, 49150, 49150, 49150]],
      shape=(256, 256), dtype=uint16)
```

[7]:
```python
from libtiff import TIFF
tif = TIFF.open("demo/demo.hdf5.zarr.tiff", "r")
tiff_data = tif.read_image()
tif.close()
tiff_data
```

[7]:
```
array([[    0,     0,     0, …, 16382, 16382, 16382],
       [    0,     0,     0, …, 16382, 16382, 16382],
       [    0,     0,     0, …, 16382, 16382, 16382],
       …,
       [32766, 32766, 32766, …, 49150, 49150, 49150],
       [32766, 32766, 32766, …, 49150, 49150, 49150],
       [32766, 32766, 32766, …, 49150, 49150, 49150]],
      shape=(256, 256), dtype=uint16)
```

[8]:
```python
import tensorstore as ts
ts.open({
    "driver": "zarr3",
    "kvstore": {
       "driver": "file",
       "path": "demo/test.zarr/"
    },
}).result().read().result()
```

[8]:
```
array([[    0,     0,     0, …, 16382, 16382, 16382],
       [    0,     0,     0, …, 16382, 16382, 16382],
       [    0,     0,     0, …, 16382, 16382, 16382],
       …,
       [32766, 32766, 32766, …, 49150, 49150, 49150],
       [32766, 32766, 32766, …, 49150, 49150, 49150],
       [32766, 32766, 32766, …, 49150, 49150, 49150]],
      shape=(256, 256), dtype=uint16)
```

[9]:
```python
import h5py
with h5py.File("demo/demo.hdf5.zarr.tiff") as h5f:
    h5data = h5f["data"][:]
    h5zarrindex = h5f["zarrindex"][:]
    offsets_and_bytes = header_formats.
 ↪get_hdf5_chunk_offsets_and_bytes(h5f["data"])

h5zarrindex[0:-4].view(np.uint64)
offsets_and_bytes
```

[9]:
```
array([[  3072,  32768],
       [ 35840,  32768],
       [ 68608,  32768],
       [101376,  32768]], dtype=uint64)
```

1.4 4. Rewrite the data using h5py

[10]:
```python
import h5py
import shutil
with h5py.File("demo/demo.hdf5.zarr.tiff", "r+") as h5f:
    h5f["data"][:128,:128] = 1
    h5f["data"][:128,128:] = 2
    h5f["data"][128:,:128] = 3
    h5f["data"][128:,128:] = 4

# copy to the zarr shard, consider a symlink on Linux systems
shutil.copyfile("demo/demo.hdf5.zarr.tiff", "demo/test.zarr/c/0/0")
```

[10]:
```
'demo/test.zarr/c/0/0'
```

1.5 5. Read the updated data using h5py, libtiff, and tensorstore

[11]:
```python
with h5py.File("demo/demo.hdf5.zarr.tiff") as h5f:
    print(h5f["data"][:])
```

```
[[1 1 1 … 2 2 2]
 [1 1 1 … 2 2 2]
 [1 1 1 … 2 2 2]
 …
 [3 3 3 … 4 4 4]
 [3 3 3 … 4 4 4]
 [3 3 3 … 4 4 4]]
```

[12]:
```python
tif = TIFF.open("demo/demo.hdf5.zarr.tiff", "r")
print(tif.read_image())
tif.close()
```

```
[[1 1 1 … 2 2 2]
 [1 1 1 … 2 2 2]
 [1 1 1 … 2 2 2]
 …
 [3 3 3 … 4 4 4]
 [3 3 3 … 4 4 4]
 [3 3 3 … 4 4 4]]
```

[13]:
```python
ts.open({
    "driver": "zarr3",
    "kvstore": {
```

```
        "driver": "file",
        "path": "demo/test.zarr/"
    },
}).result().read().result()
```

[13]:
```
array([[1, 1, 1, …, 2, 2, 2],
       [1, 1, 1, …, 2, 2, 2],
       [1, 1, 1, …, 2, 2, 2],
       …,
       [3, 3, 3, …, 4, 4, 4],
       [3, 3, 3, …, 4, 4, 4],
       [3, 3, 3, …, 4, 4, 4]], shape=(256, 256), dtype=uint16)
```