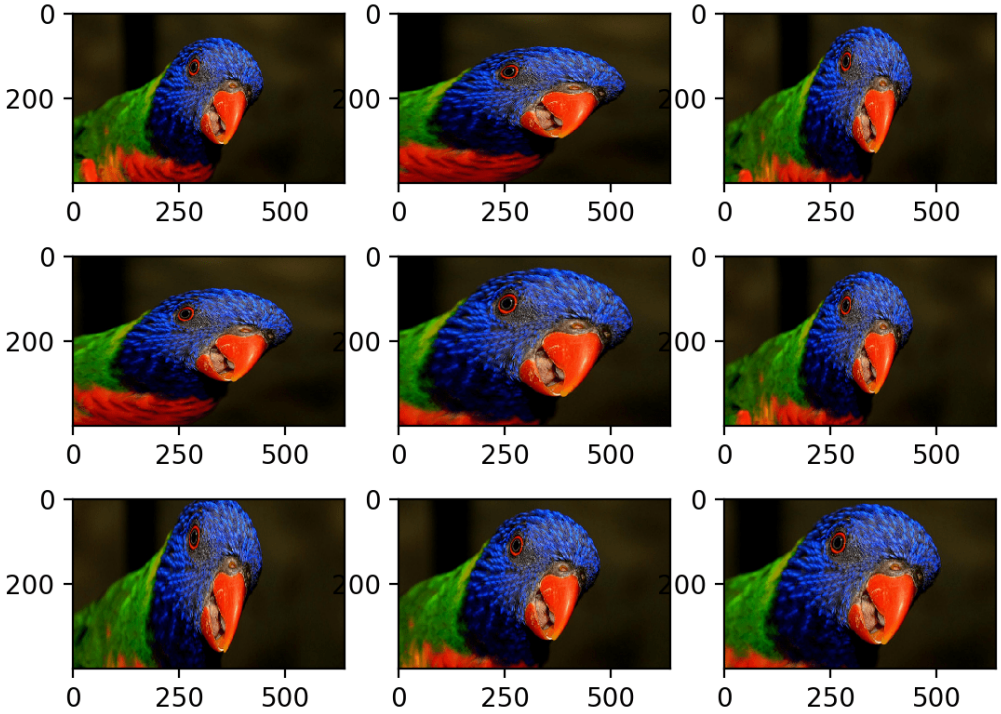
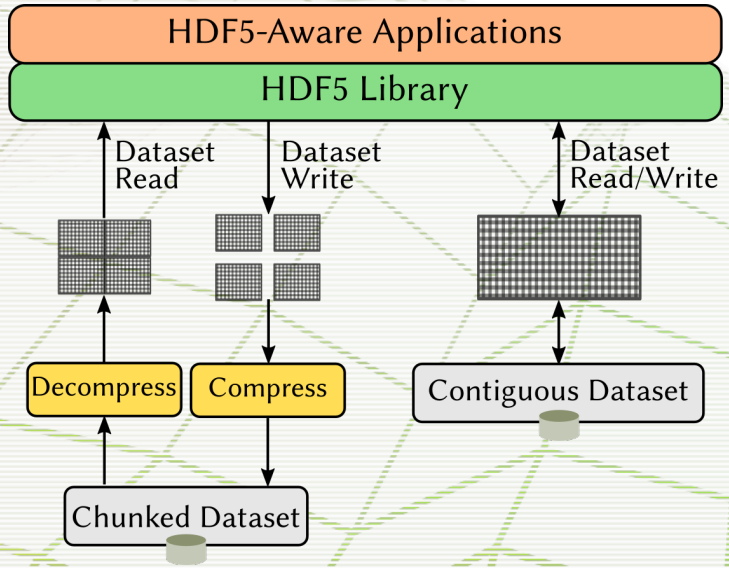
Programmable Datasets with User-Defined Functions (HDF5-UDF)

Lucas Villa Real
IBM Research
lucasvr@br.ibm.com

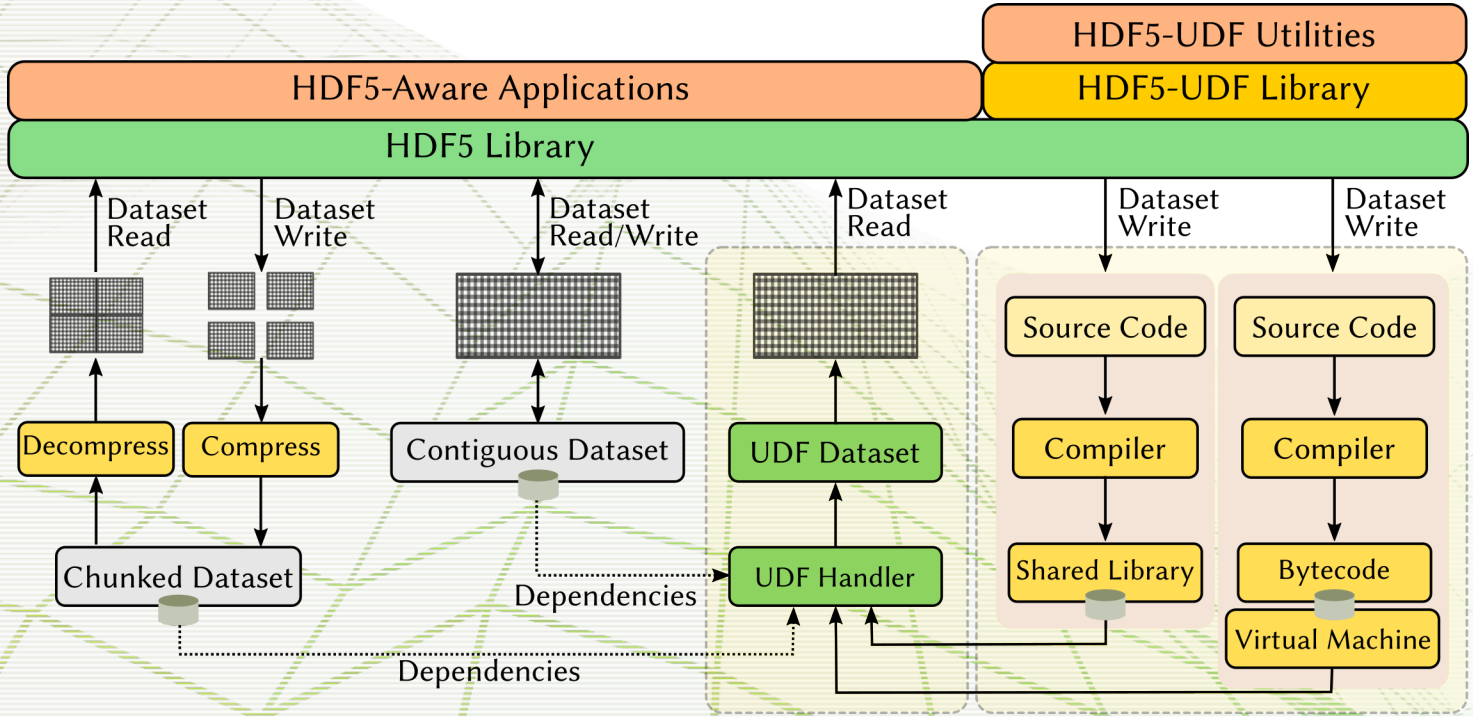
Data augmentation → storage bloat



Traditional HDF5 data flow



Enhanced data flow with User-Defined Functions



Support for popular programming languages

```python
def dynamic_dataset()
    a, b, c = lib.getData("A"), lib.getData("B"), lib.getData("C")

    for i in range(lib.getDims("A")[0] * lib.getDims("A")[1]):
        c[i] = a[i] + b[i]
```

Python

```cpp
extern "C" void dynamic_dataset() {
    auto a = lib.getData<float>("A");
    auto b = lib.getData<float>("B");
    auto c = lib.getData<float>("C");

    for (auto i=0, i < lib.getDims("A")[0] * lib.getDims("A")[1]; ++i)
        c[i] = a[i] + b[i];
}
```

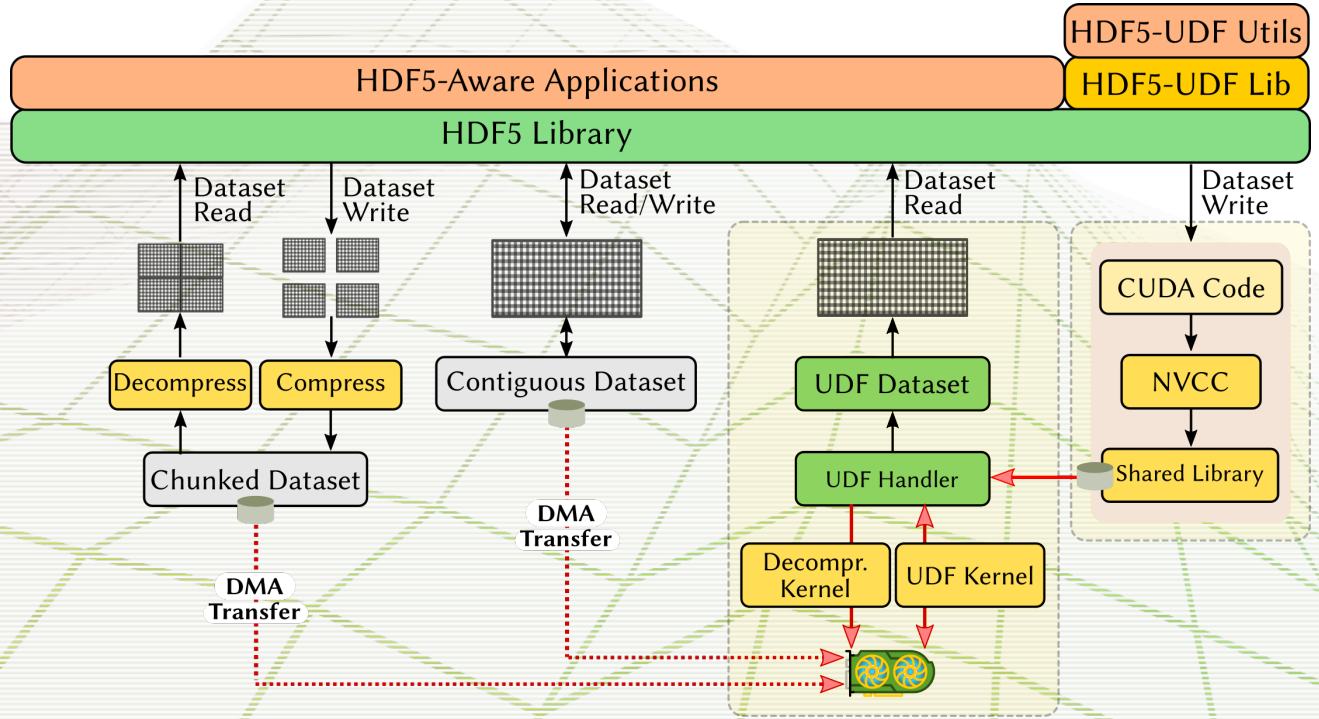
C++

```lua
function dynamic_dataset()
    local a = lib.getData("A")
    local b = lib.getData("B")
    local c = lib.getData("C")

    for i=1, lib.getDims("A")[1] * lib.getDims("A")[2] do
        c[i] = a[i] + b[i]
    end
end
```

Lua

“HPC datasets” with CUDA UDFs




```
__global__ void
kernel(int *red, int *nir, float *ndvi, size_t n)
{
  int idx = blockIdx.x * blockDim.x + threadIdx.x;
  if (i < n)
    ndvi[i] = (nir[i]-red[i]) / (nir[i]+red[i]);
}

extern “C” void dynamic_dataset()
{
  // Output dataset
  auto ndvi = lib.getData<float>(“NDVI”);
  auto dims = lib.getDims(“NDVI”);
  auto n = dims[0] * dims[1];

  // Input datasets
  auto red = lib.getData<int>(“Red”);
  auto nir = lib.getData<int>(“NIR”);

  // Configure and launch the kernel
  int block_dim = 1024;
  int grid_dim = ceil((float)(n*sizeof(int))/block_dim);

  kernel<<<grid_dim,block_dim>>>(red, nir, ndvi, n);
}
```

Attaching UDFs to HDF5 files, command line

```
$ hdf5-udf file.h5 udf.{py,cpp,cu,lua} DatasetName:Dimensions:DataType
```

Example:

```
$ hdf5-udf file.h5 udf.py xray_negative_logarithm:512x512:float
```

Attaching UDFs to HDF5 files, Python

```
$ pip install PyHDF5-UDF
```

```python
from hdf5_udf import UserDefinedFunction

with UserDefinedFunction(hdf5_file='file.h5', udf_file='udf.py') as udf:
    udf.push_dataset({'name': name, 'datatype': dtype, 'resolution': [x,y,z]})
    udf.compile()
    udf.store()
```

UDF metadata

```
UDF dataset header = {
 “backend”: “CPython”,
 “bytecode_size”: 879,
 “input_datasets”: [“xray”],
 “output_dataset”: “xray_negative_logarithm”,
 “output_datatype”: “float”,
 “output_resolution”: [512, 512],
 “signature”: {
   “email”: “lucasvr@br.ibm.com”,
   “name”: “Lucas C. Villa Real”,
   “public_key”: “17ryiejfF2SNlT+MCIaPLb7bKqo2FTX/PIiCDrh45Fc=”
 },
 ...
}
```

UDFs are always signed with the user’s private key

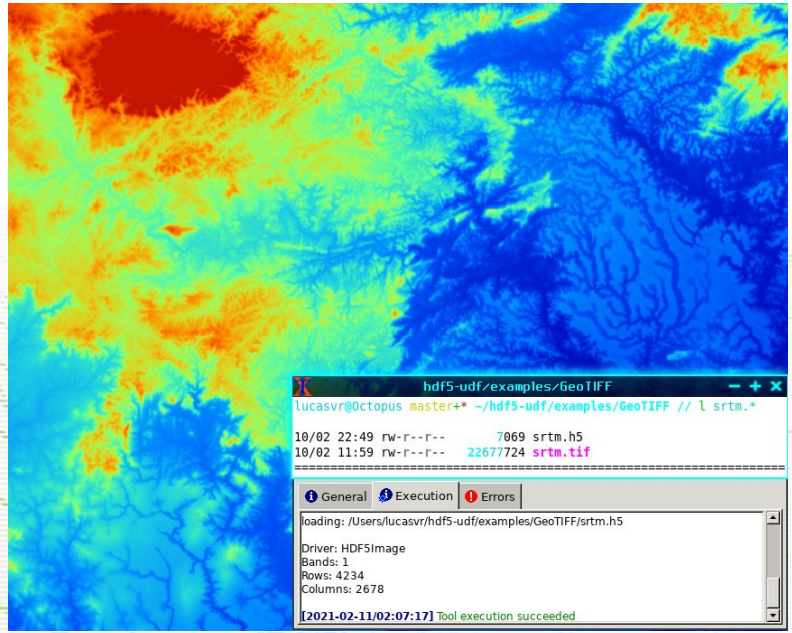
Use case: data virtualization

GeoTIFF to HDF5

```
def dynamic_dataset():
    from tifffile import TiffFile
    input = TiffFile('srtm.tif').pages[0].asarray().flatten()
    output = lib.getData('SRTM')
    output[0:input.shape[0]] = input[:]
```

CSV to HDF5

```
def dynamic_dataset():
    udf_data = lib.getData('IoT_Sensor')
    udf_dims = lib.getDims('IoT_Sensor')
    with open('sensor.csv') as f:
        for i, line in enumerate(f.readlines()[1:]):
            entries = line.split(',')
            udf_data[i].timestamp = int(entries[0])
            udf_data[i].value = float(entries[1])
            lib.setString(udf_data[i].event, entries[2].encode('utf-8')
```



Use case: data virtualization + reprojection

```
def dynamic_dataset():
    import rioxarray as rxr
    input = rxr.open_rasterio('srtm-utm.tif')
    output = lib.getData('LatLonDataset')
    dims = lib.getDims('LatLonDataset')

    reprojected = input.rio.reproject('EPSG:4326').data.flatten()
    output[0 : dims[1]*dims[2]] = reprojected
```

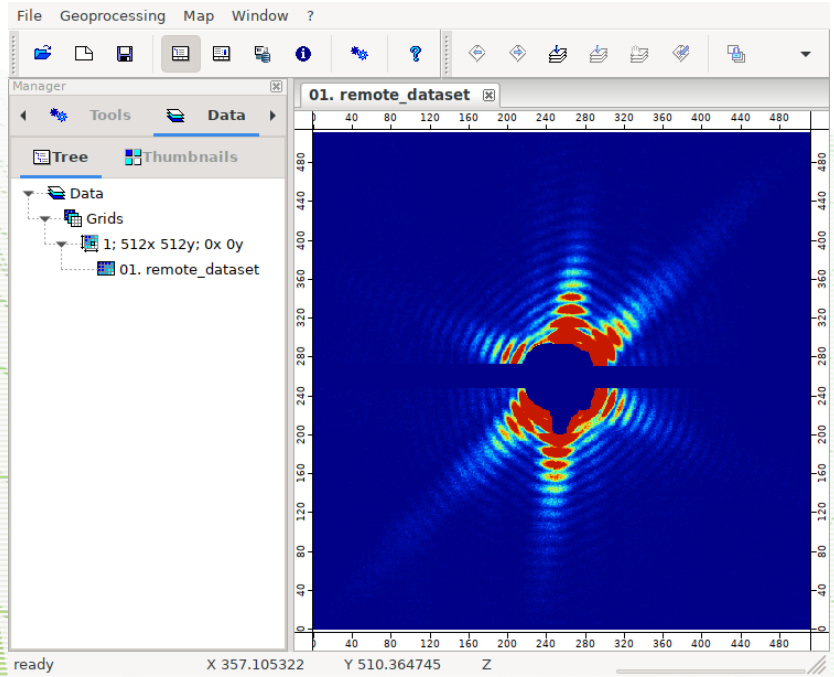
Use case: virtualization of remote data

CXIDB
Coherent X-ray Imaging Data Bank

```
def dynamic_dataset():
    import h5py
    import requests
    from io import BytesIO

    udf_data = lib.getData('remote_dataset')
    udf_dims = lib.getData('remote_dataset')
    size = udf_dims[0] * udf_dims[1] * udf_dims[2]

    bio = BytesIO(requests.get(url='https://cxidb.org/data/1/cxidb-1.h5').content)
    with h5py.File(bio, 'r') as f:
        udf_data[0:size] = f['real'][:,:,:].flatten()
```



Use case: accessing data behind web services

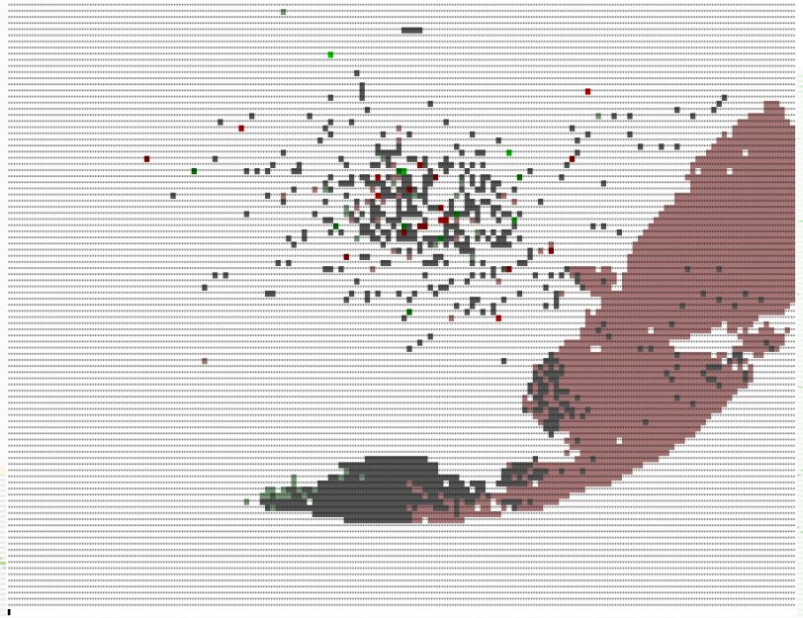
```python
from owslib.wms import WebMapService
import png

def dynamic_dataset():
    # tord: Chicago O'Hare International Airport, bvel: L3 Base Radial Velocity
    wms = WebMapService("https://opengeo.ncep.noaa.gov/geoserver/tord/ows?service=wms", version="1.3.0")
    bbox = wms["tord_bvel"].boundingBoxWGS84

    rdims = lib.getDims("RadialVelocity")
    img = wms.getmap(layers=["tord_bvel"], srs="EPSG:4316", bbox=bbox, size=rdims, format="image/png").read()

    # Create a look-up table to map (r,g,b) to a color index
    lut = {}
    velocity = lib.getData("RadialVelocity")
    rows = png.Reader(bytes=img).read()[2]
    for i, row in enumerate(rows):
        rgba = list(zip(*[row[i::4] for i in range(4)]))
        for j, (r,g,b,a) in enumerate(rgba):
            if not (r,g,b) in lut:
                velocity[i*len(rgba) + j] = lut[(r,g,b)] = len(lut)
            else:
                velocity[i*len(rgba) + j] = lut[(r,g,b)]

    # Export palette
    palette = lib.getData("Palette")
    reverse_lut = sorted([(v,k) for k,v in lut.items()], key=lambda kv: kv[0])
    for (i, rgb) in reverse_lut:
        palette[i*3 : i*3+3] = rgb
```



Welcome back to the era of KBs

	Contiguous layout		Chunked layout	
	1000 x 1000	16000 x 16000	1000 x 1000	16000 x 16000
Traditional dataset	3.8 MB	976 MB	624 KB	155 MB
UDF (C++)	3.9 KB		3.9 KB	
UDF (LuaJIT)	2.2 KB		2.2 KB	
UDF (CPython)	6.2 KB		6.2 KB	

Userbase not restricted to a single OS

Please visit us at
https://github.com/lucasvr/hdf5-udf

Programmable Datasets with User-Defined Functions (HDF5-UDF)

Lucas Villa Real
IBM Research
lucasvr@br.ibm.com