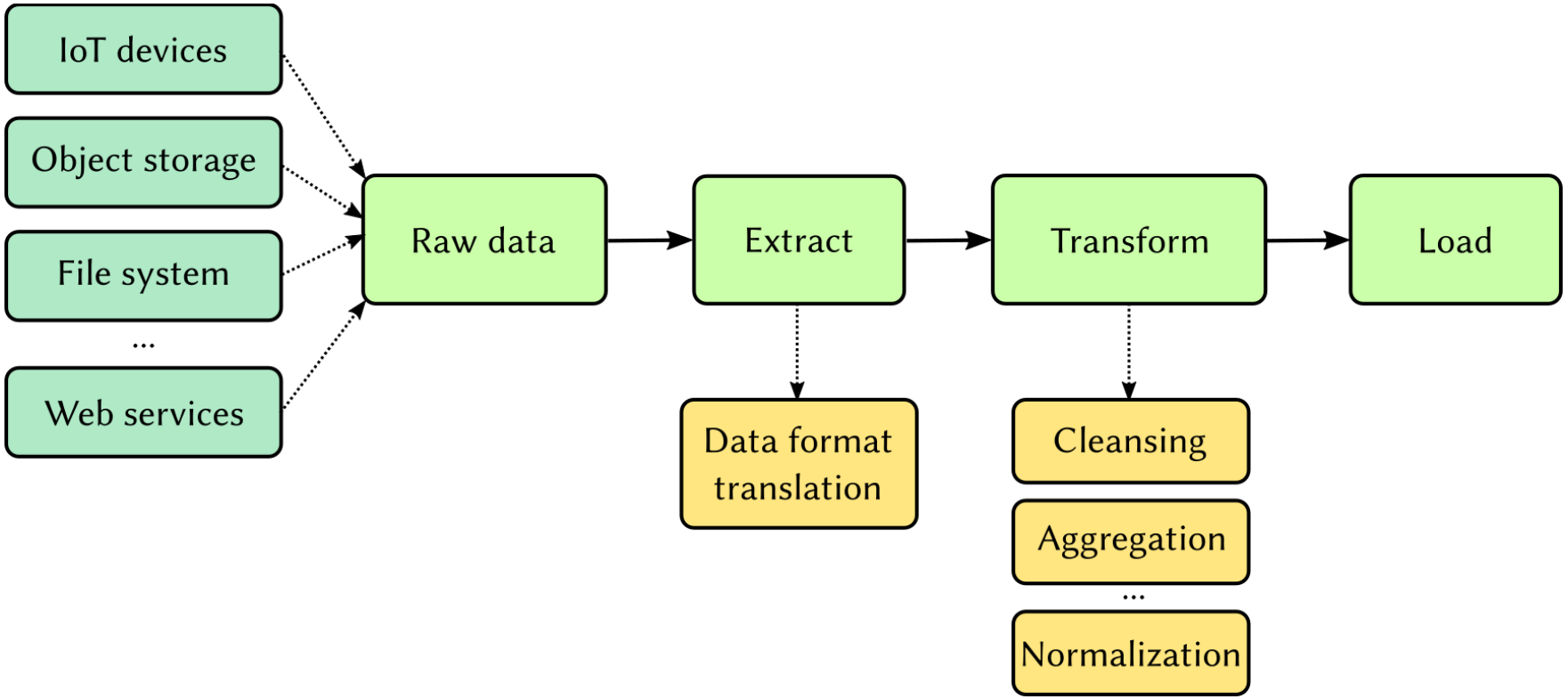
STORAGE DEVELOPER CONFERENCE
SDC 21
BY Developers FOR Developers
Virtual Conference
September 28-29, 2021

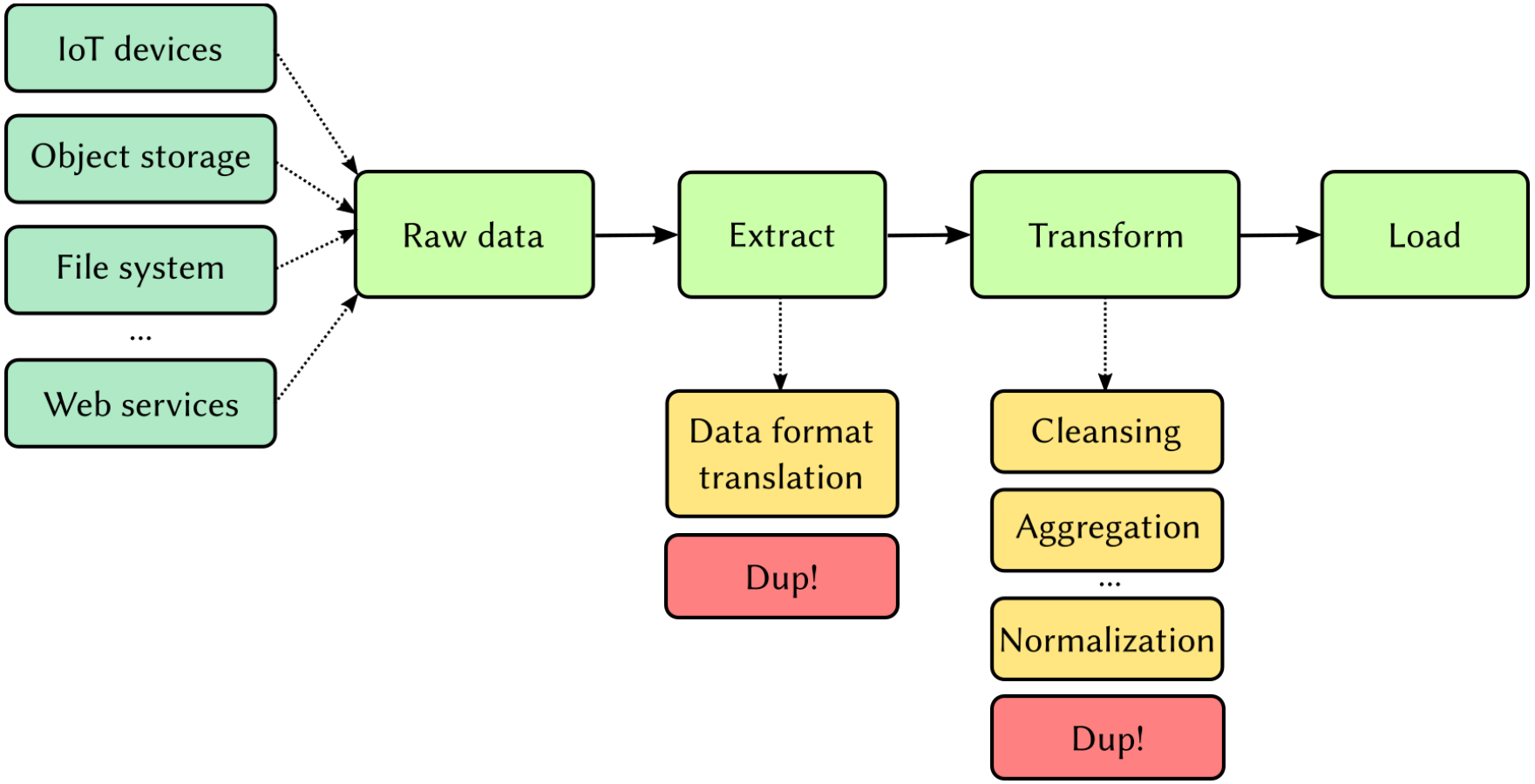
Scientific Data Powered by User-Defined Functions

Presented by Lucas C. Villa Real
lucasvr@br.ibm.com

Traditional data pipeline

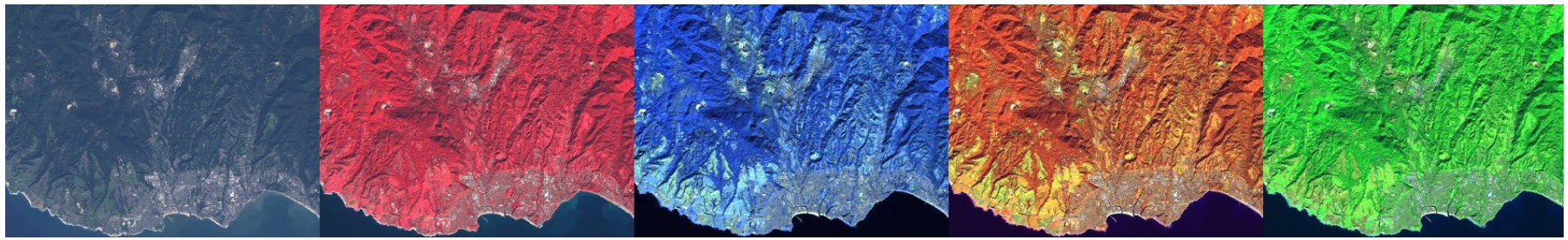


Traditional data pipeline



Many transformations are simple!

```
1 def main()
2      h5 = h5py.File(”landsat.h5”, ”r+”)
3      nir, red = h5[“nir_band”], h5[“red_band”]
4      ndvi = (nir - red) / (nir + red)
5      h5.create_dataset(“ndvi”, nir.shape, nir.dtype, data=ndvi)
```



HDF5

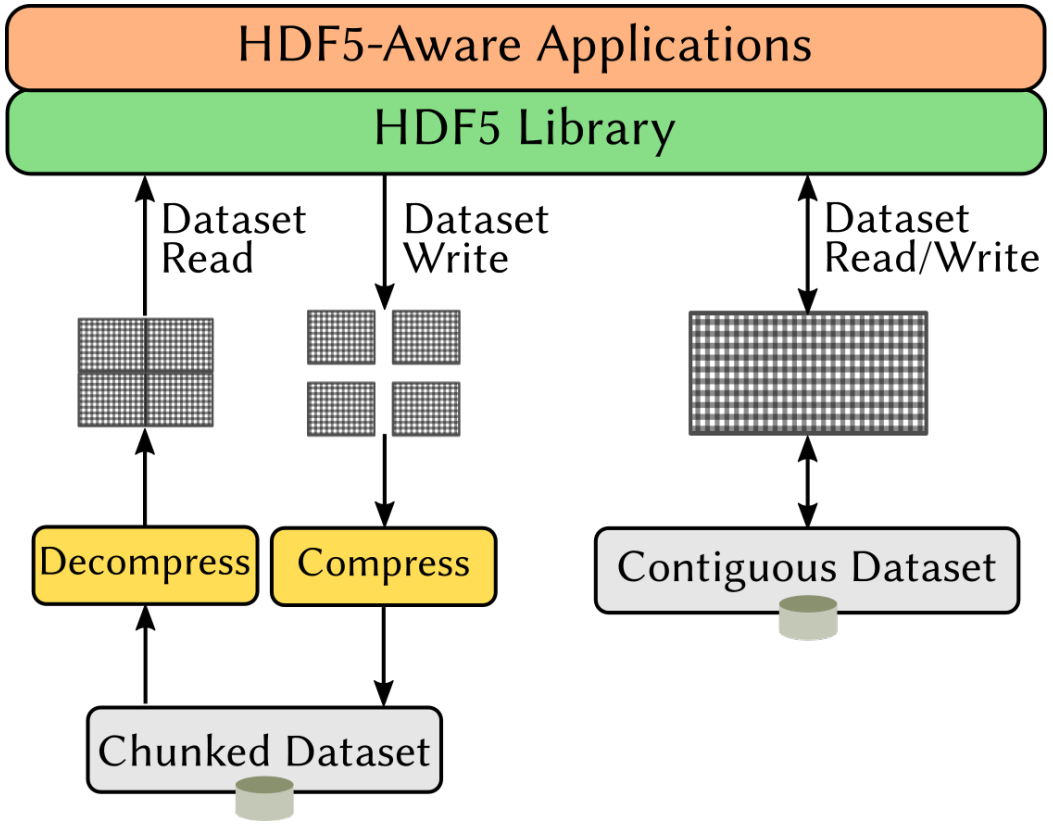
Hierarchical Data Format, version 5

HDF5 in a nutshell

```
1  LandsatMosaic {
2    dimensions:
3      columns = 1440 ;
4      rows = 720 ;
5    variables:
6      short Band1(rows, columns) ;
7        Band1:long_name = "Coastal aerosol" ;
8      short Band2(rows, columns) ;
9        Band2:long_name = "Blue" ;
10     short Band3(rows, columns) ;
11       Band3:long_name = "Green" ;
12     short Band4(rows, columns) ;
13       Band4:long_name = "Red" ;
14     short Band5(rows, columns) ;
15       Band5:long_name = "Near-Infrared (NIR)" ;
16     ...
17     float Band12(rows, columns) ;
18       Band12:long_name = "Normalized Difference
19         Vegetation Index (NDVI)" ;
20     }
21 }
```

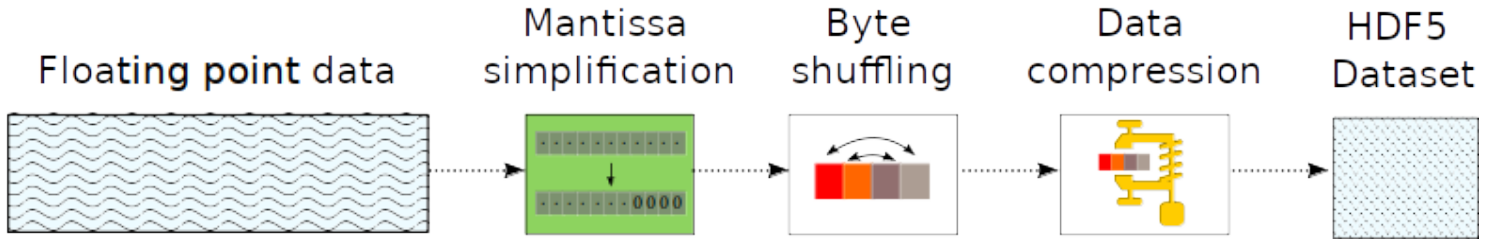
- Hierarchical arrangement of data
- Native and compound data types
- Multidimensional arrays
- Key:value attributes
- Parallel I/O through MPI
- Contiguous & chunked storage layouts

HDF5 in a nutshell



- Hierarchical arrangement of data
- Native and compound data types
- Multidimensional arrays
- Key:value attributes
- Parallel I/O through MPI
- Contiguous & chunked storage layouts

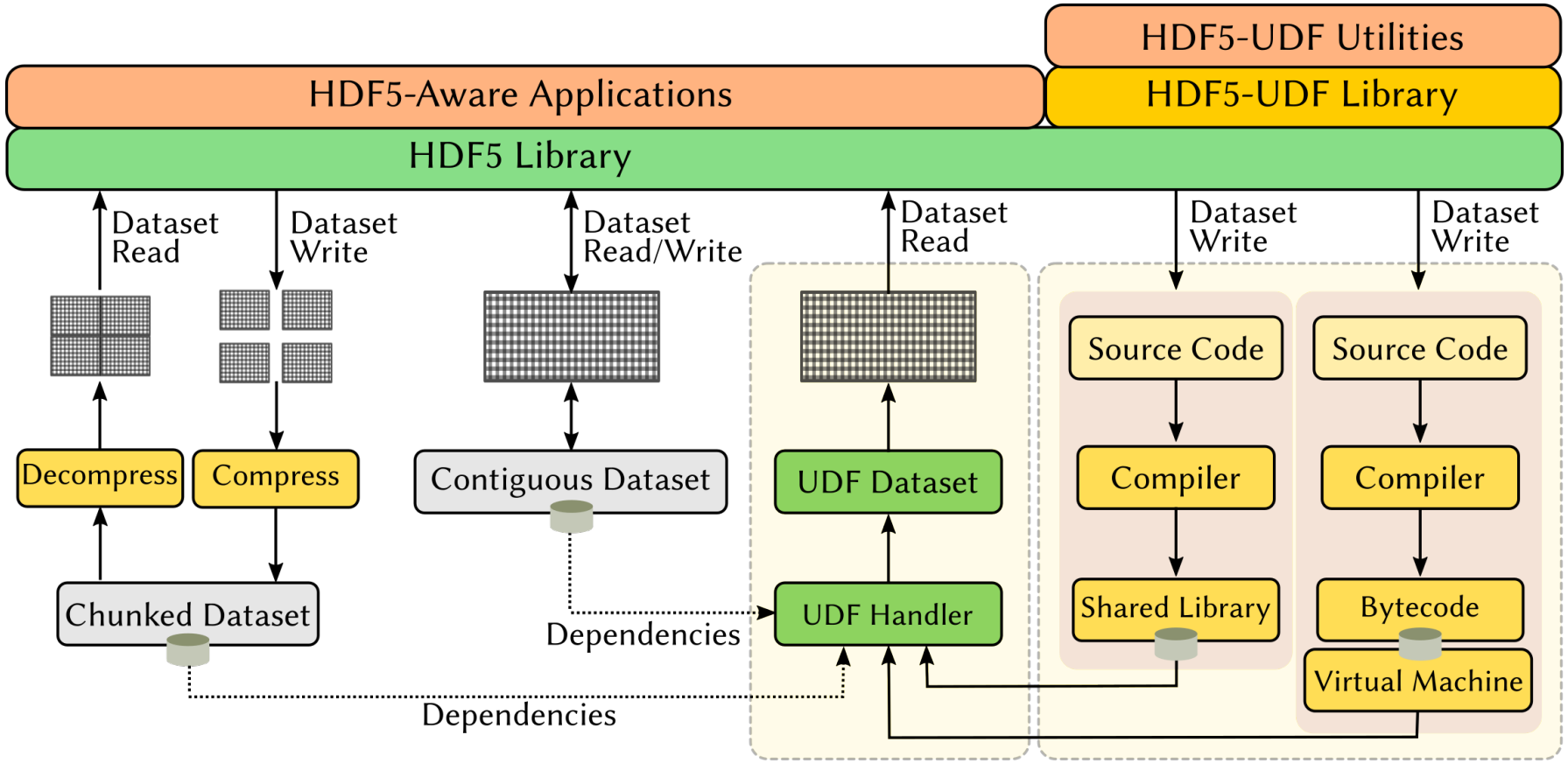
HDF5 I/O Filters



User-Defined Functions for HDF5

Handling data redundancy differently

UDF architecture at a glance



API for UDF writers

```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

API for UDF writers

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

C++

API for UDF writers

```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 < sizelib.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

API for UDF writers

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

C++

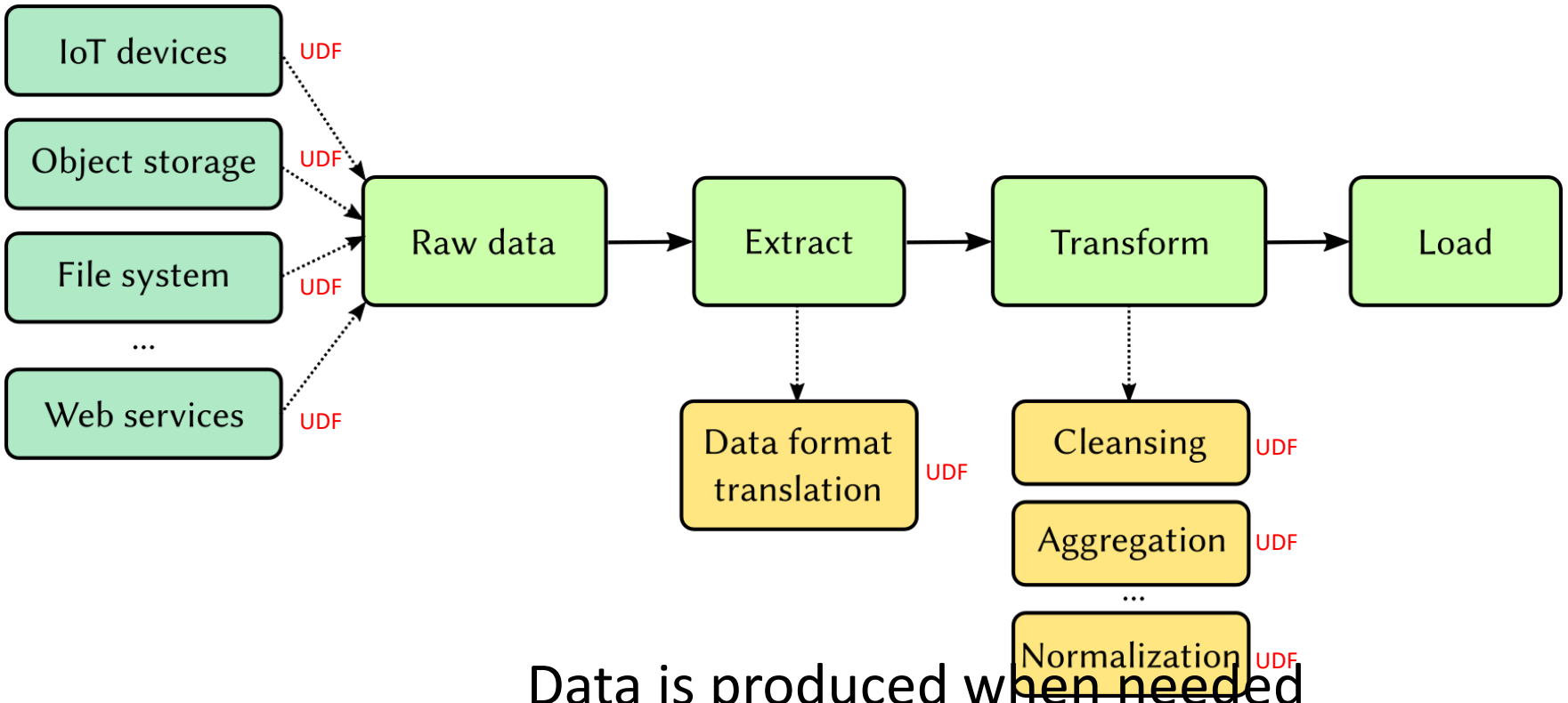
```lua
function dynamic_dataset()
    local a = lib.getData("A")
    local b = lib.getData("B")
    local c = lib.getData("C")

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

Lua

```
# hdf5-udf file.h5 udf.py C:1024x768:float
```


Going back to our original problem…

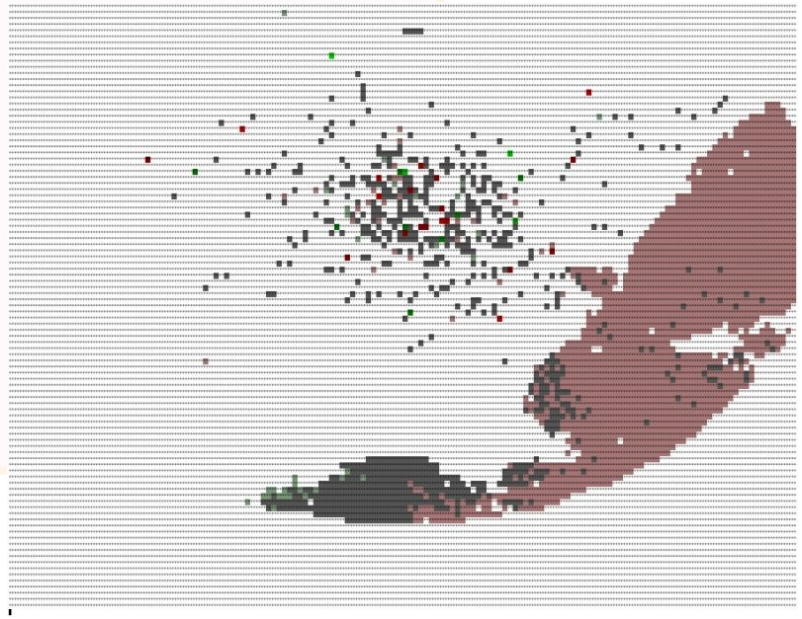
Traditional data pipeline



Data is produced ~~when needed~~

Retrieval of data from web services

```
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
```



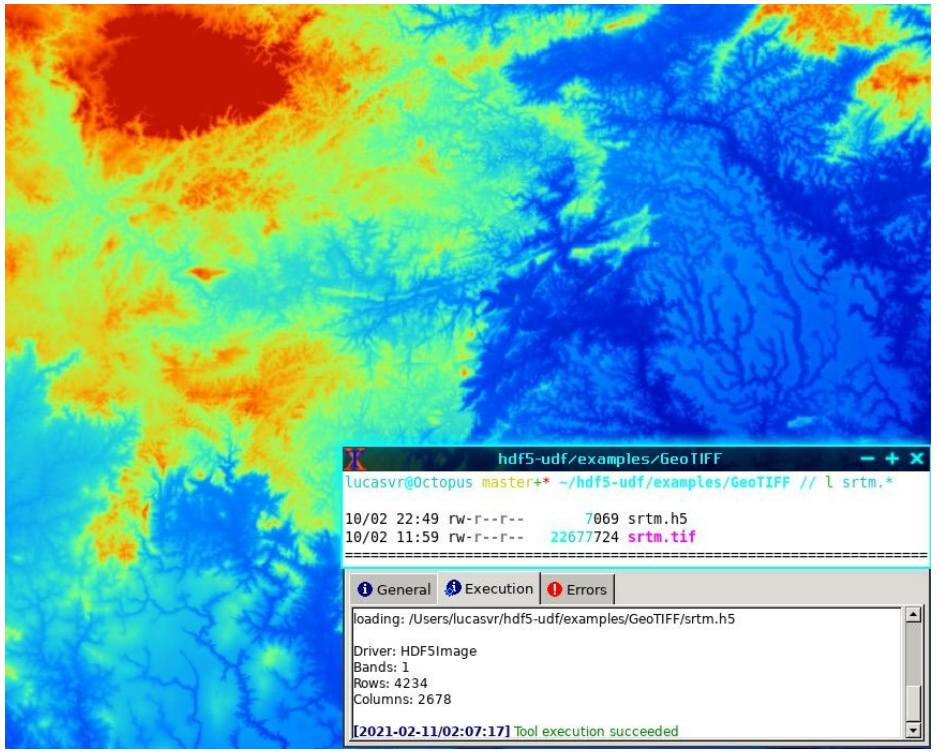
Data format translation

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[:]
```

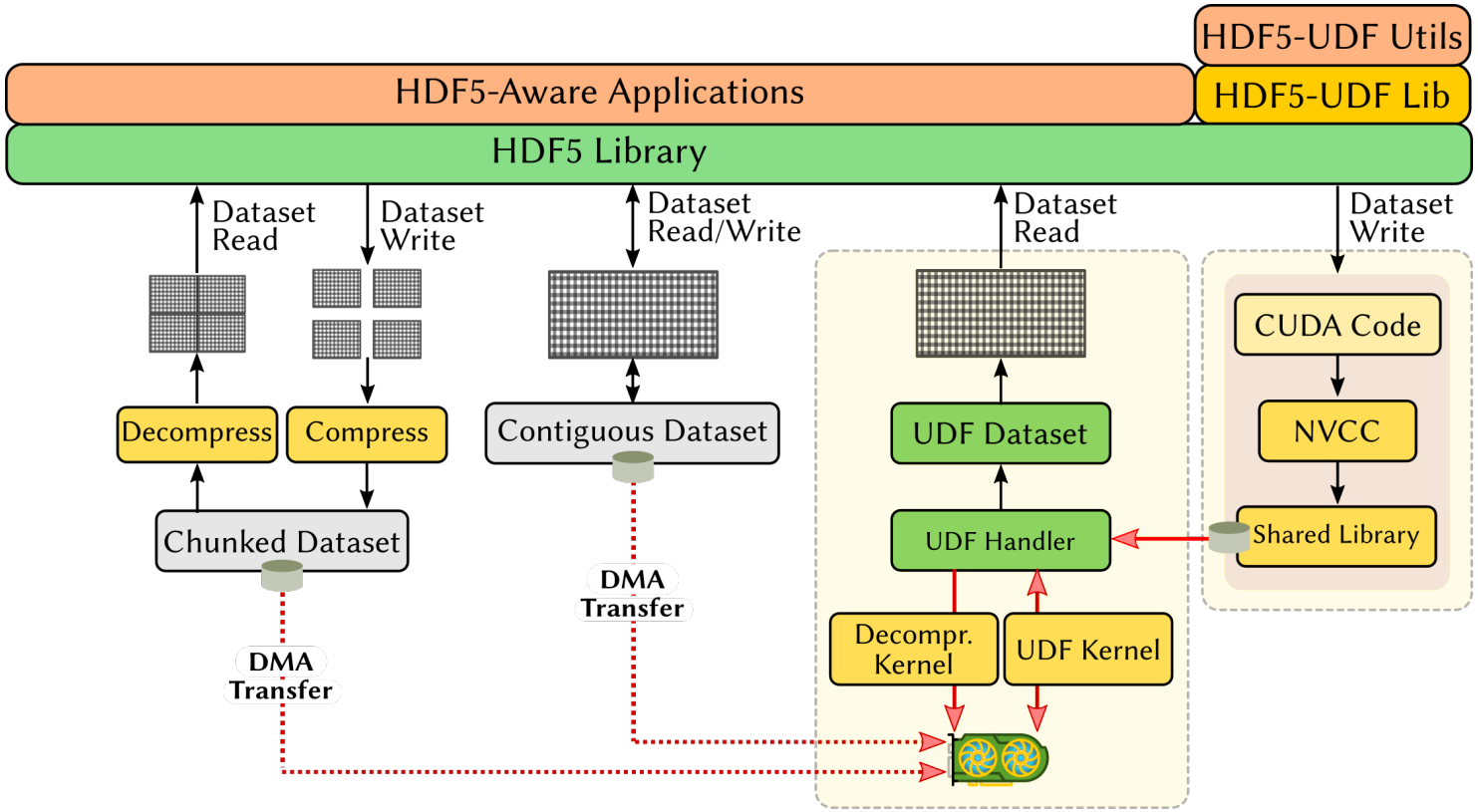
CSV to HDF5

```
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')
```



UDFs and computational storage

UDFs + hardware accelerators




```
__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);
}
```


HDF5-UDF under the hoods

On-disk format: bytecode + metadata

```
UDF dataset header = {
 "backend": "CPython",
 "bytecode_size": 879,
 "input_datasets": ["A", "B"],
 "output_dataset": "C",
 "output_datatype": "float",
 "output_resolution": [1024, 768],
 "signature": {
   "email": "lucasvr@PageFault",
   "name": "Lucas C. Villa Real",
   "public_key": "17ryiejfF2SNlT+MCIaPLb7bKqo2FTX/PIiCDrh45Fc="
 },
 ...
}
```

UDFs are always signed with the user's private key

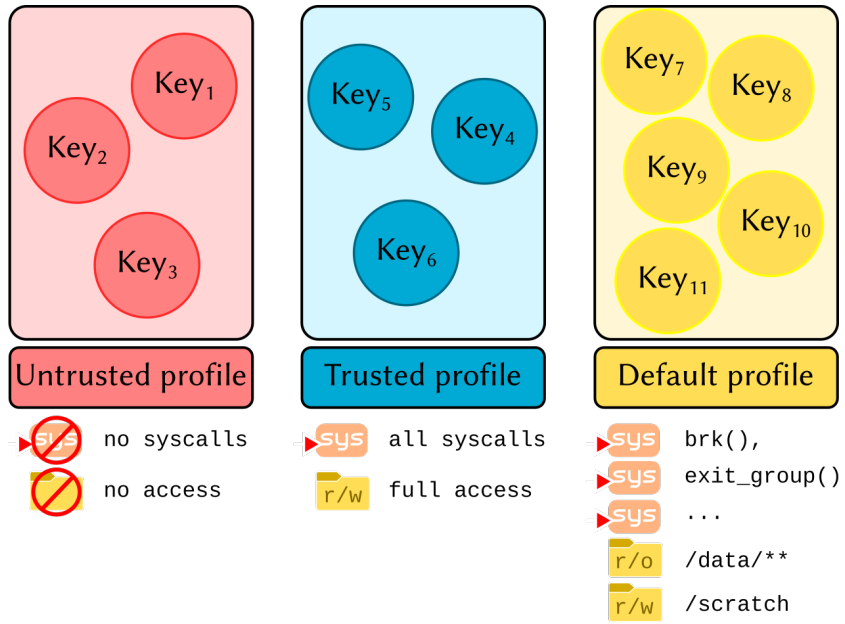
Security profiles

Define syscalls and filesystem access rules

- Directory-based, rules written in JSON
- Simply move .pub files to associate them with a different profile

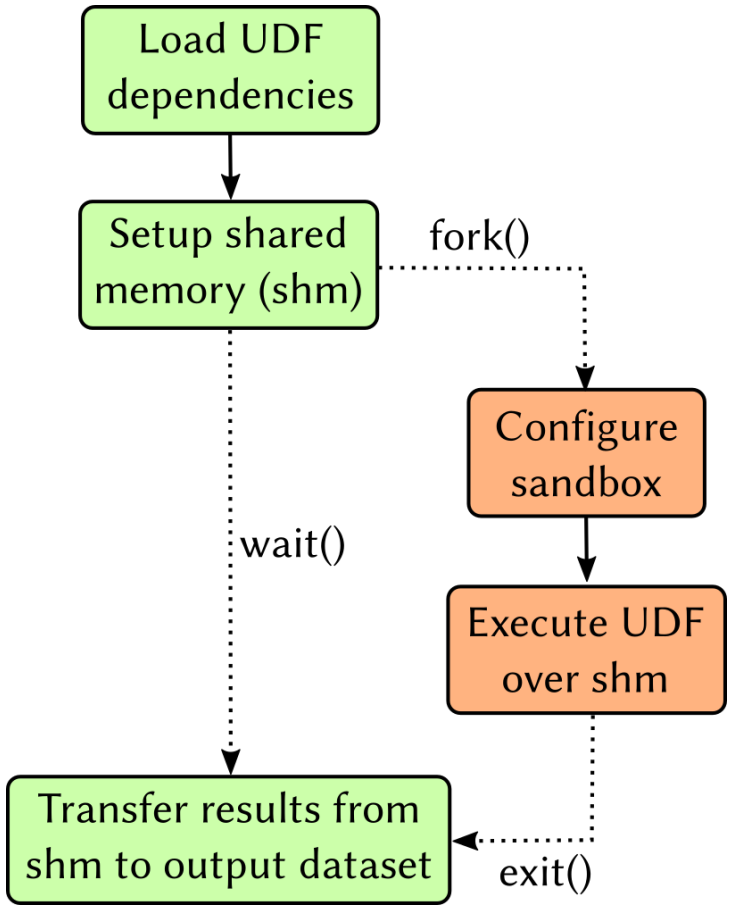
Sane defaults

- Newly imported public keys are unstrusted
- UDFs are always trusted when signed and executed on the same machine
- Default profile allows access to Python modules and basic system libraries




```
# ls ~/.config/hdf5_udf
allow  default  deny  lucasvr.priv  lucasvr.pub

# ls ~/.config/hdf5_udf/deny
deny.json  john@doe.pub

# mv ~/.config/hdf5_udf/deny/john@doe.pub \
     ~/.config/hdf5_udf/allow
```


Sandboxed execution of UDFs



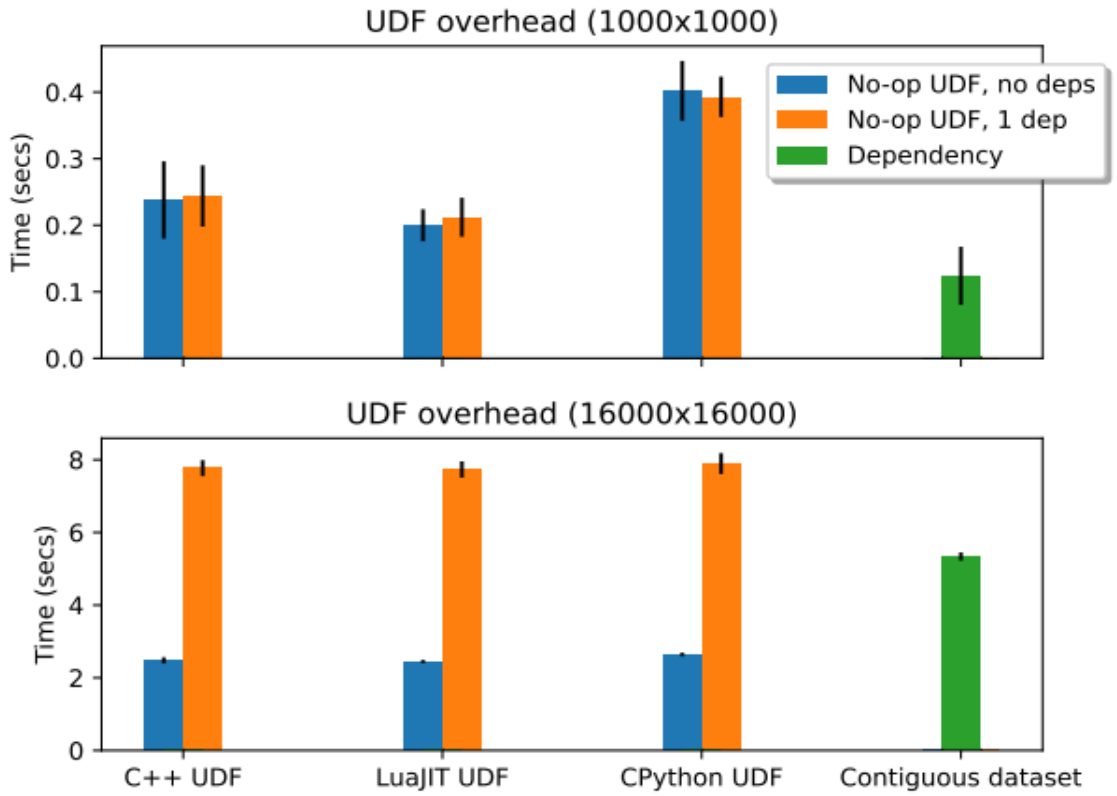
- The UDF process is terminated if it violates access rules
- Linux: Seccomp + Ptrace (eBPF coming soon)
- Windows: no process creation with fork() semantics, needs a different approach
- macOS: Ptrace only partially implemented – Apple's Endpoint Security is an alternative

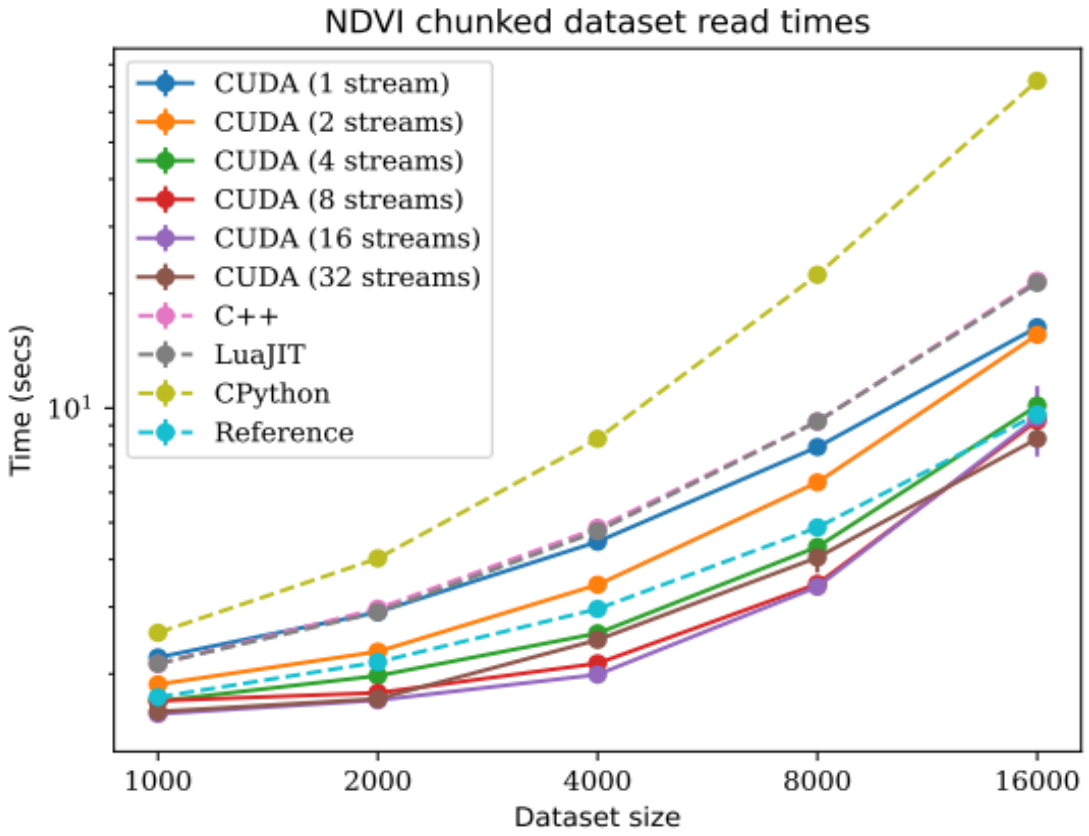
Performance numbers

Dataset size comparison

	Contiguous layout		Chunked layout	
	1000x1000	**16000x16000**	**1000x1000**	**16000x16000**
Reference	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	

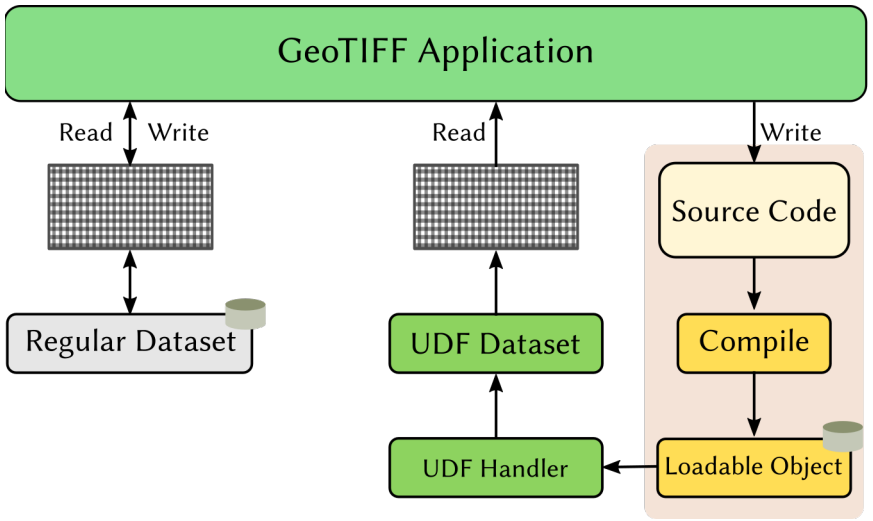
Overhead of programming language runtimes





Moving forward…

Support for other file formats?



TIFF / GeoTIFF

- Writer: create standalone utility
- Reader:
- – Dynamically register a TIFF Tag field
- – Declare supported data types
- – Implement a read callback
- – May need to resort to LD_PRELOAD

STORAGE DEVELOPER CONFERENCE
SDC 21
BY Developers FOR Developers
Virtual Conference
September 28-29, 2021

Scientific Data Powered by User-Defined Functions

Presented by Lucas C. Villa Real
https://github.com/lucasvr/hdf5-udf

Please take a moment to rate this session.

Your feedback is important to us.

