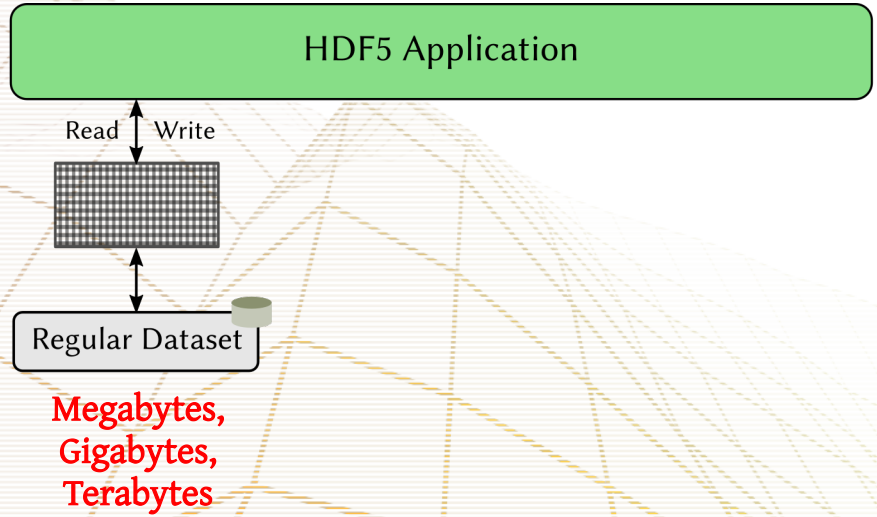
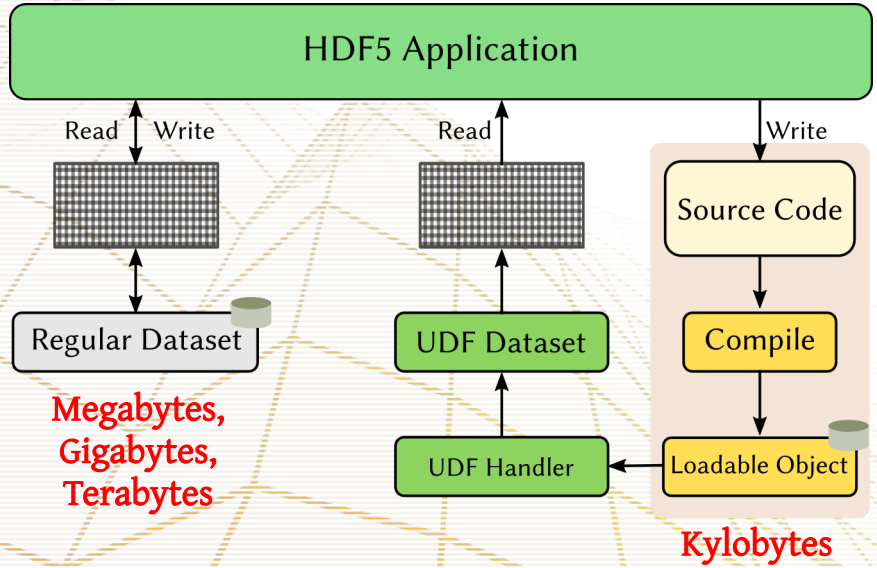
User-Defined Functions for HDF5

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

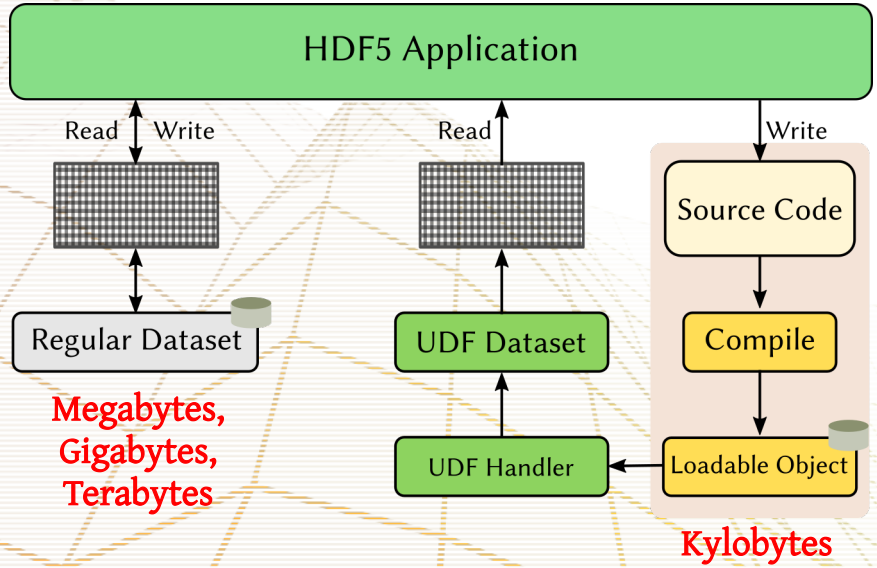
Traditional HDF5 application flow



HDF5 application flow supported by UDFs

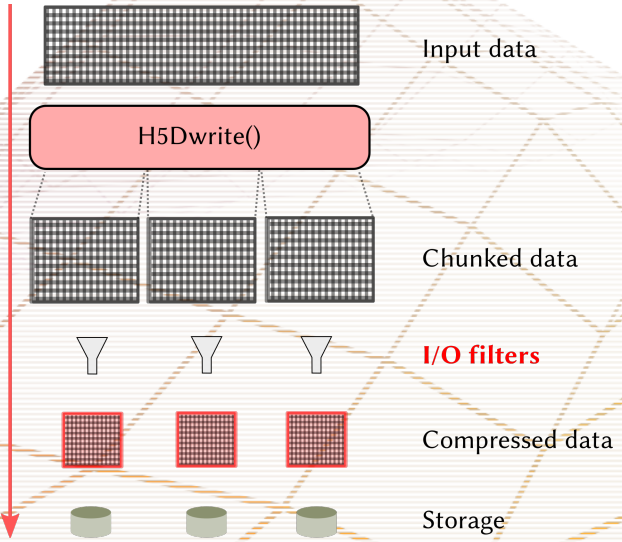


HDF5 application flow supported by UDFs



Datasets are produced on-the-fly

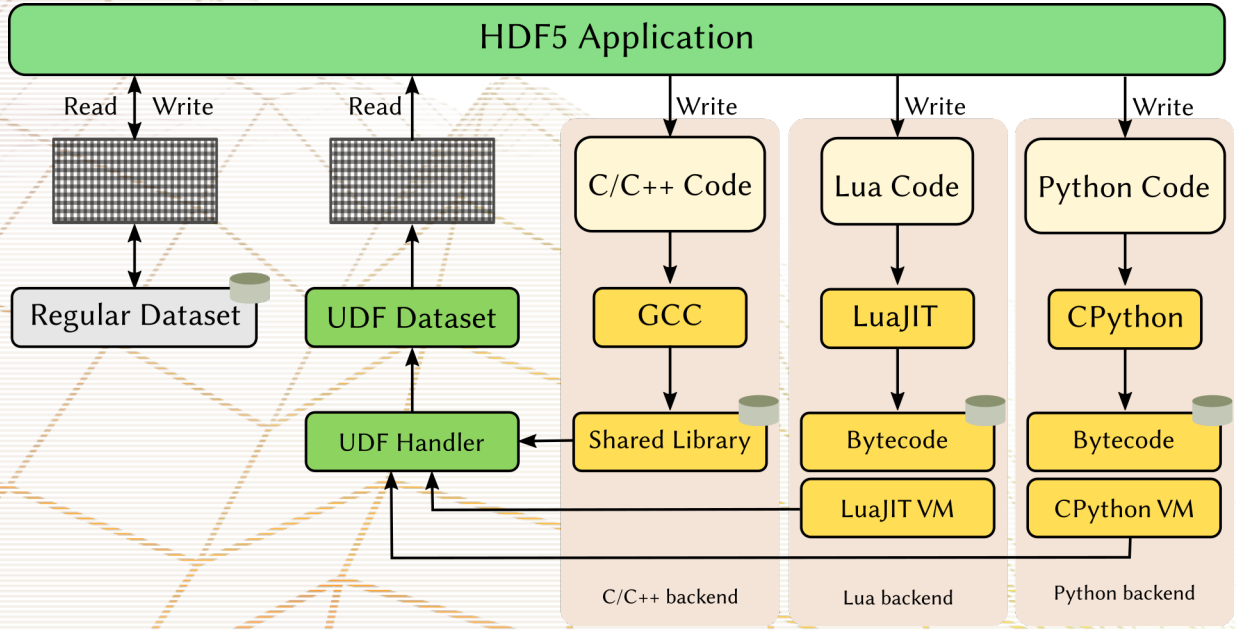
The technology underneath



HDF5 I/O filters

- Loadable plugins: deflate, bzip2, etc
- Operate on chunks of data
- Write path: compresses data
  - UDF: compile and store
- Read path: reads decompressed
  - UDF: execute

UDF architecture at a glance



API for UDF writers

Entry point: dynamic_dataset()

lib.getData(“DatasetName”) → pointer to data

lib.getDims(“DatasetName”) → array

ex: [1024, 768]

lib.getType(“DatasetName”) → string

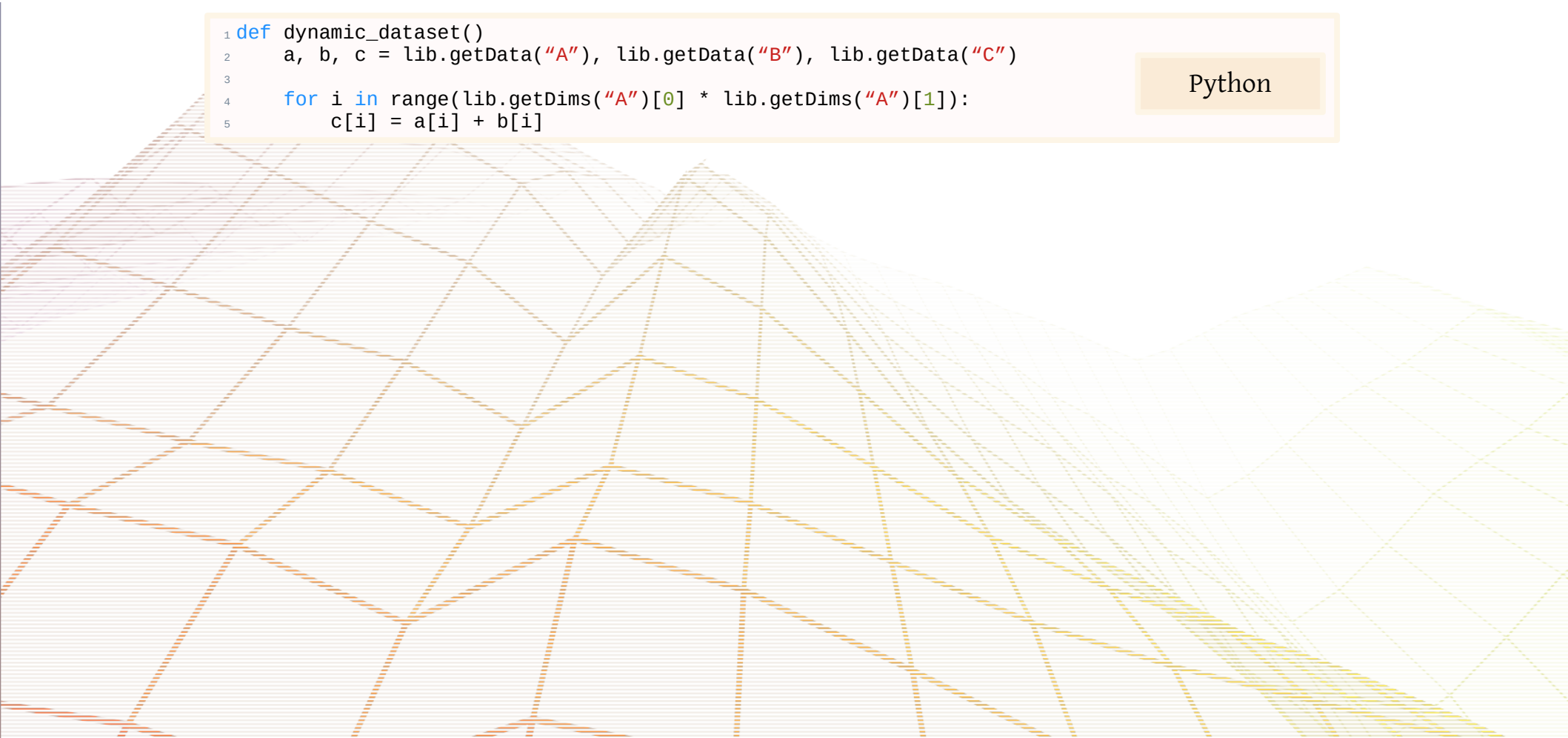
int16, int32, int64,
uint16, uint32, uint64,
float, double, string, compound

lib.setString(compound.member, “Value”)

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
```

Skipping the command line: HDF5-UDF API

```
udf_context *libudf_init(const char *hdf5_file, const char *udf_file);

bool libudf_set_option(const char *option, const char *value, udf_context *ctx);

bool libudf_push_dataset(const char *description, udf_context *ctx);

bool libudf_compile(udf_context *ctx);

bool libudf_store(char *metadata, size_t *size, udf_context *ctx);

bool libudf_get_metadata(const char *dataset, char *metadata, size_t *size, udf_context *ctx);

size_t libudf_get_error(char *buf, size_t size, udf_context *ctx);

void libudf_destroy(udf_context *ctx);
```

Skipping C programming: Python bindings

```
# pip install PyHDF5-UDF
```

```
from hdf5_udf import UserDefinedFunction

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

UDF metadata

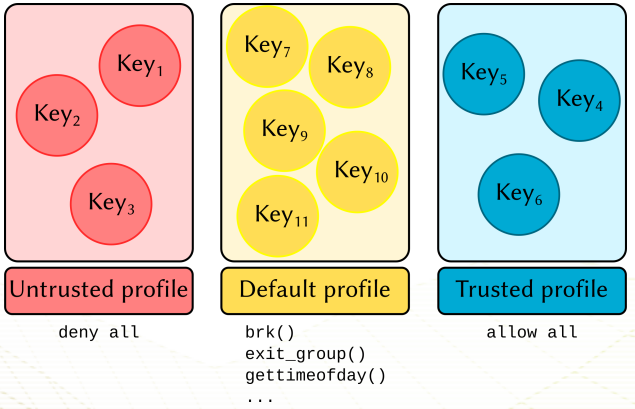
```
UDF dataset header = {
 “backend”: “CPython”,
 “bytecode_size”: 879,
 “input_datasets”: [“A”, “B”],
 “output_dataset”: “C”,
 “output_datatype”: “float”,
 “output_resolution”: [1024, 768],
 “signature”: {
   “email”: “lucasvr@PageFault”,
   “name”: “Lucas C. Villa Real”,
   “public_key”: “17ryiejfF2SNlT+MCIaPLb7bKqo2FTX/PIiCDrh45Fc=”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

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

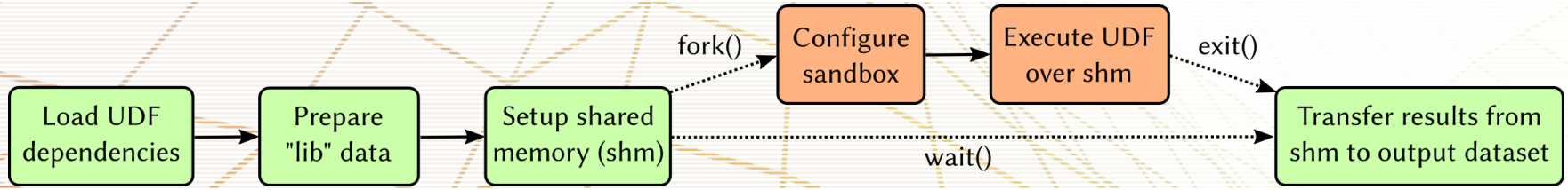
- Newly imported public keys are unstrusted
- UDFs are always trusted when signed and executed on the same machine
- Default profile allows access to Python modules and basic system libraries (e.g., SSL, math)

```
# 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
```

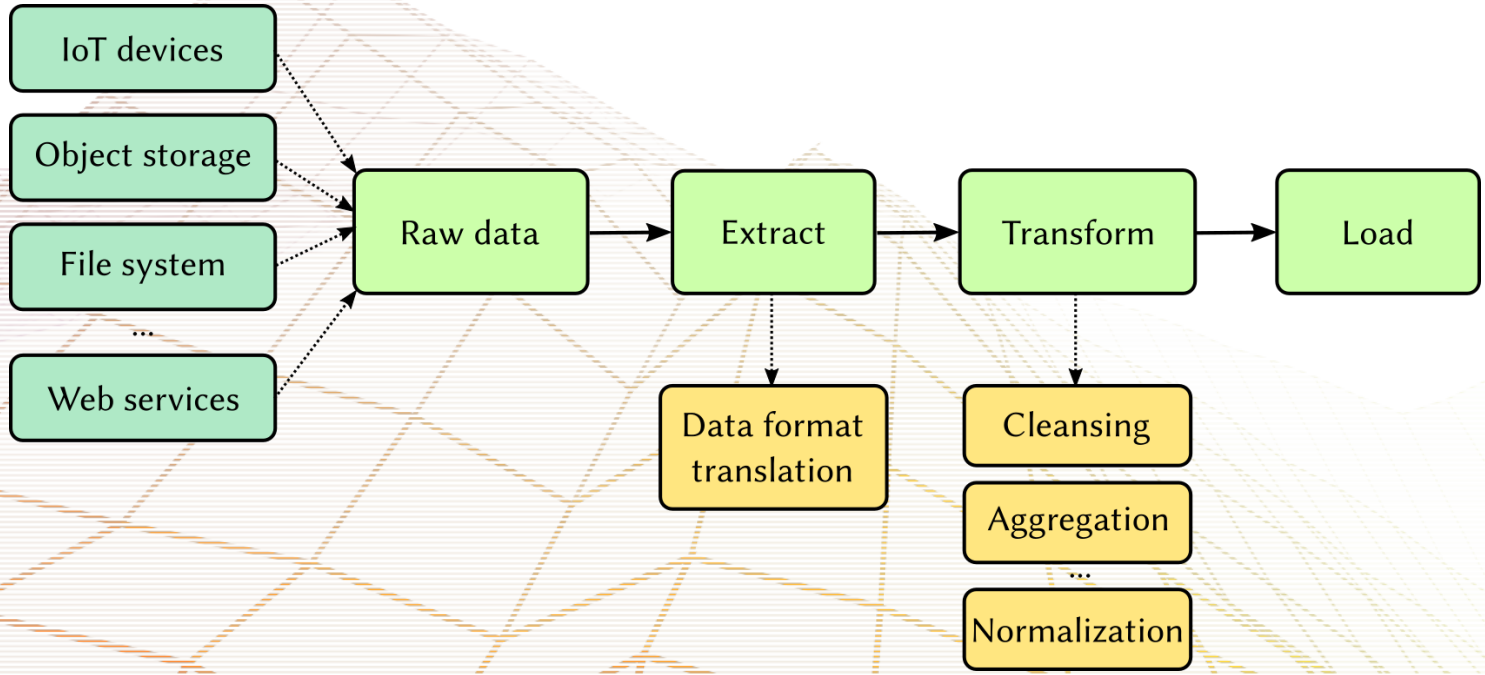
Sandboxed environment

UDFs execute in a sandbox

- The UDF process is terminated if it violates the profile rules
- Linux: Seccomp + Ptrace
- Windows, macOS: N/A

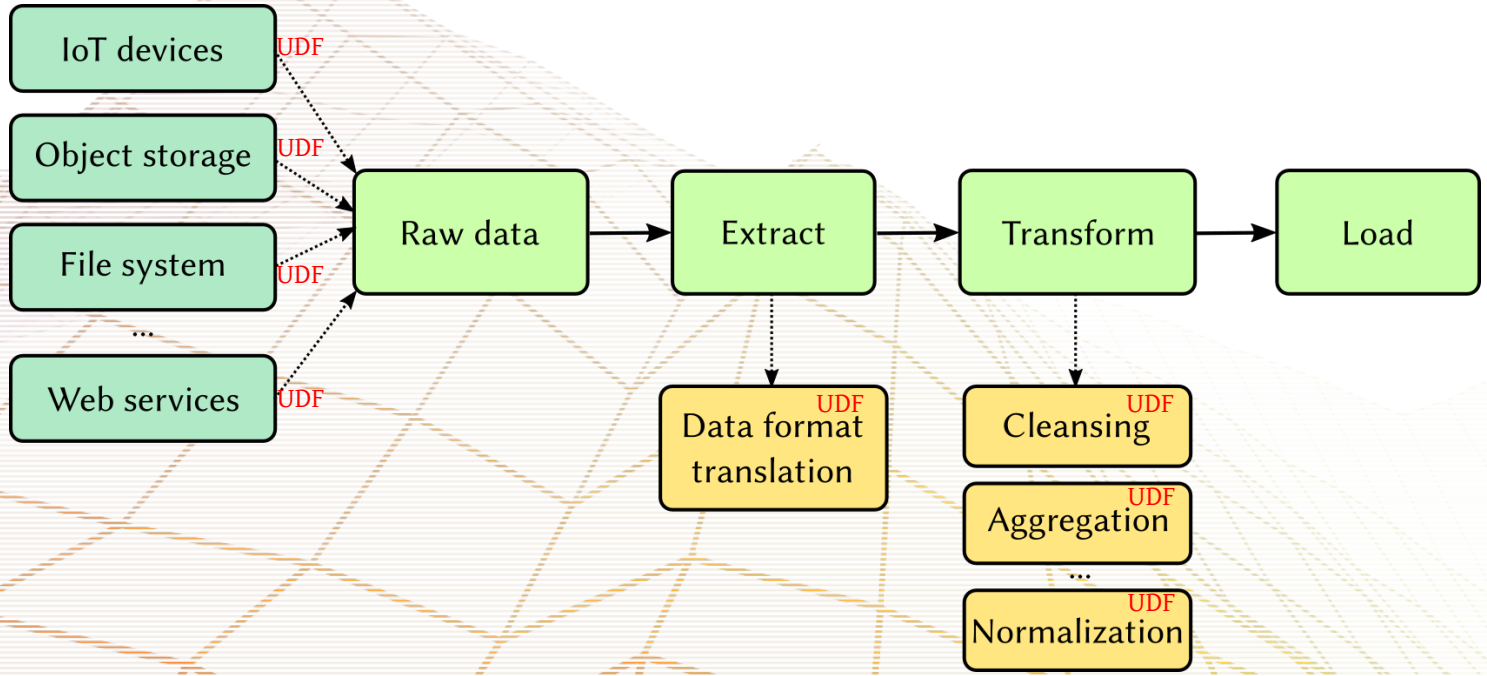


Use case in ETL pipelines



Data is produced on all stages of the pipeline
Sometimes *just in case* someone will need it

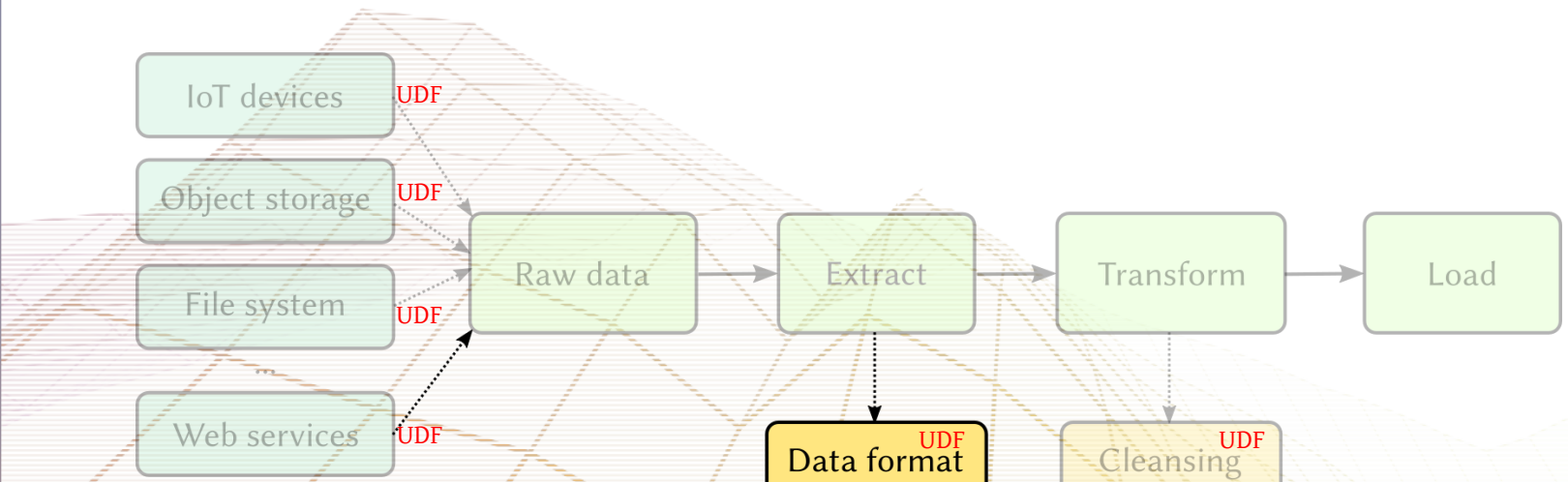
Use case in ETL pipelines



Data is produced <u>when needed</u>

Saves storage and leads to faster data transfers over the network

Use case in ETL pipelines




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

```python
def dynamic_dataset():
    input = lib.getData('aggregated')
    output = lib.getData('normalized')
    n = lib.getDims('normalized')
    max_val = max([x for x in input[0:n]])
    output[0:n] = [x/max_val for x in input[0:n]]
```

Have fun with HDF5-UDF!

Try it out at:
https://github.com/lucasvr/hdf5-udf

Check the examples at:
https://github.com/lucasvr/user-defined-functions

User-Defined Functions for HDF5

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