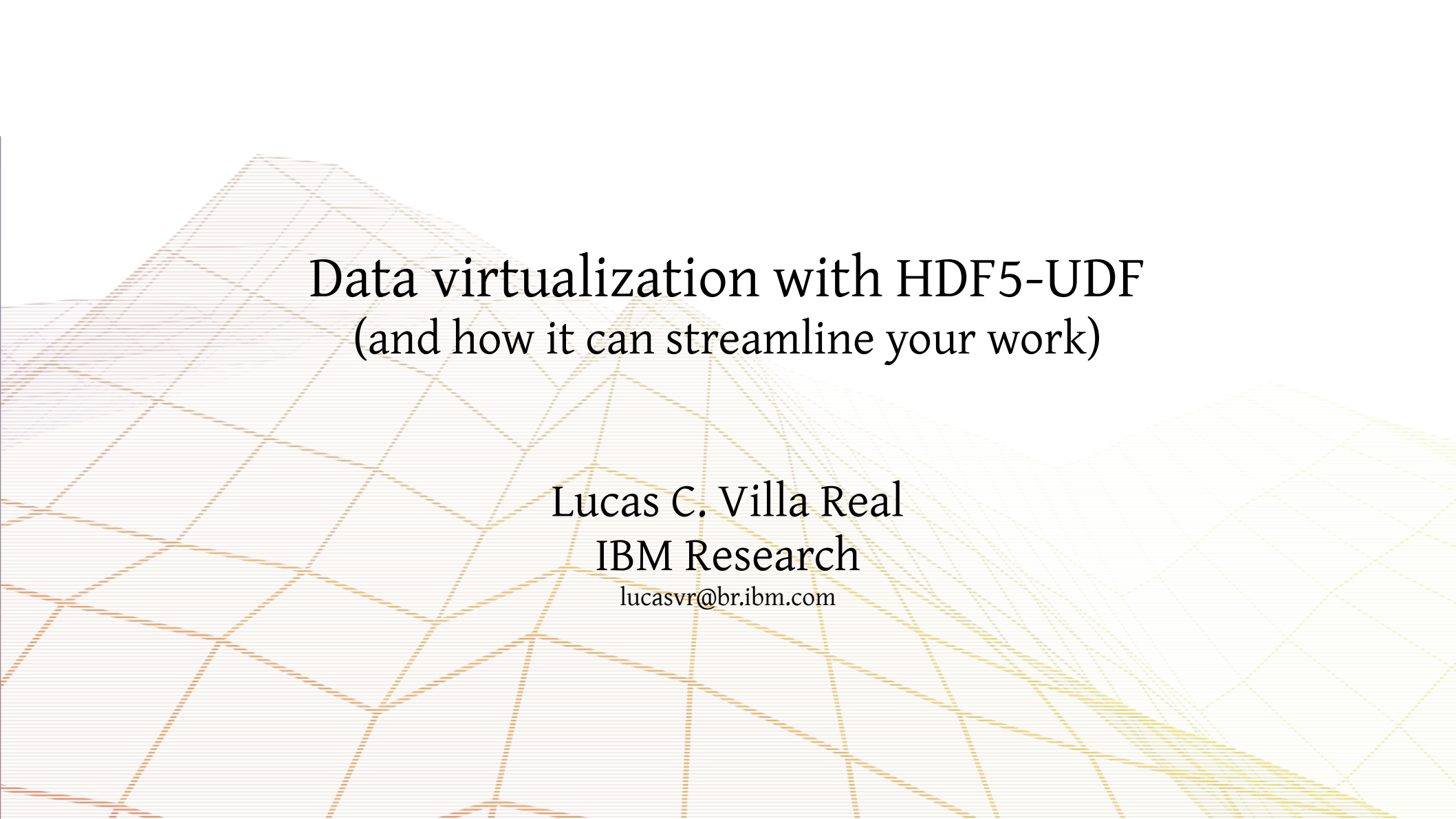


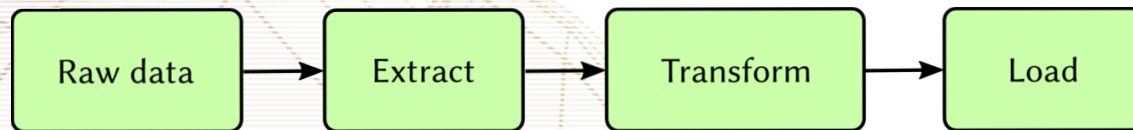


Data virtualization with HDF5-UDF

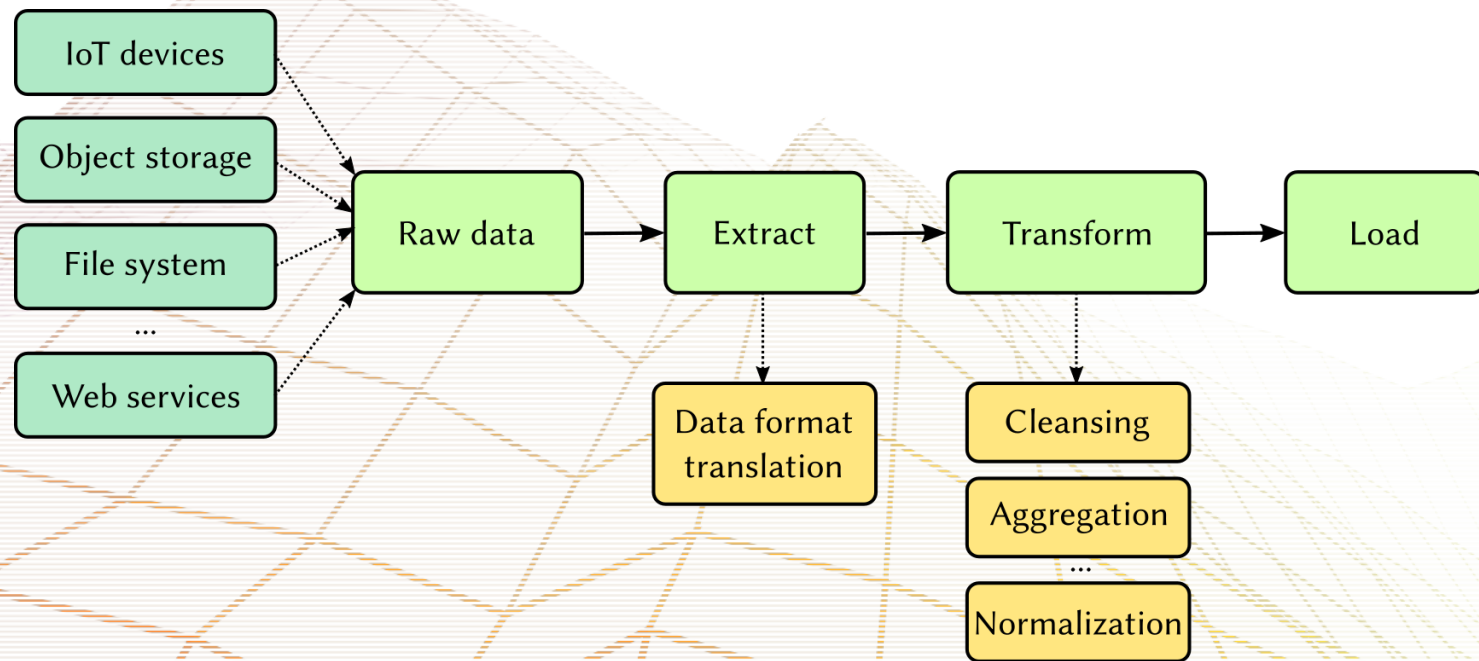
(and how it can streamline your work)

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

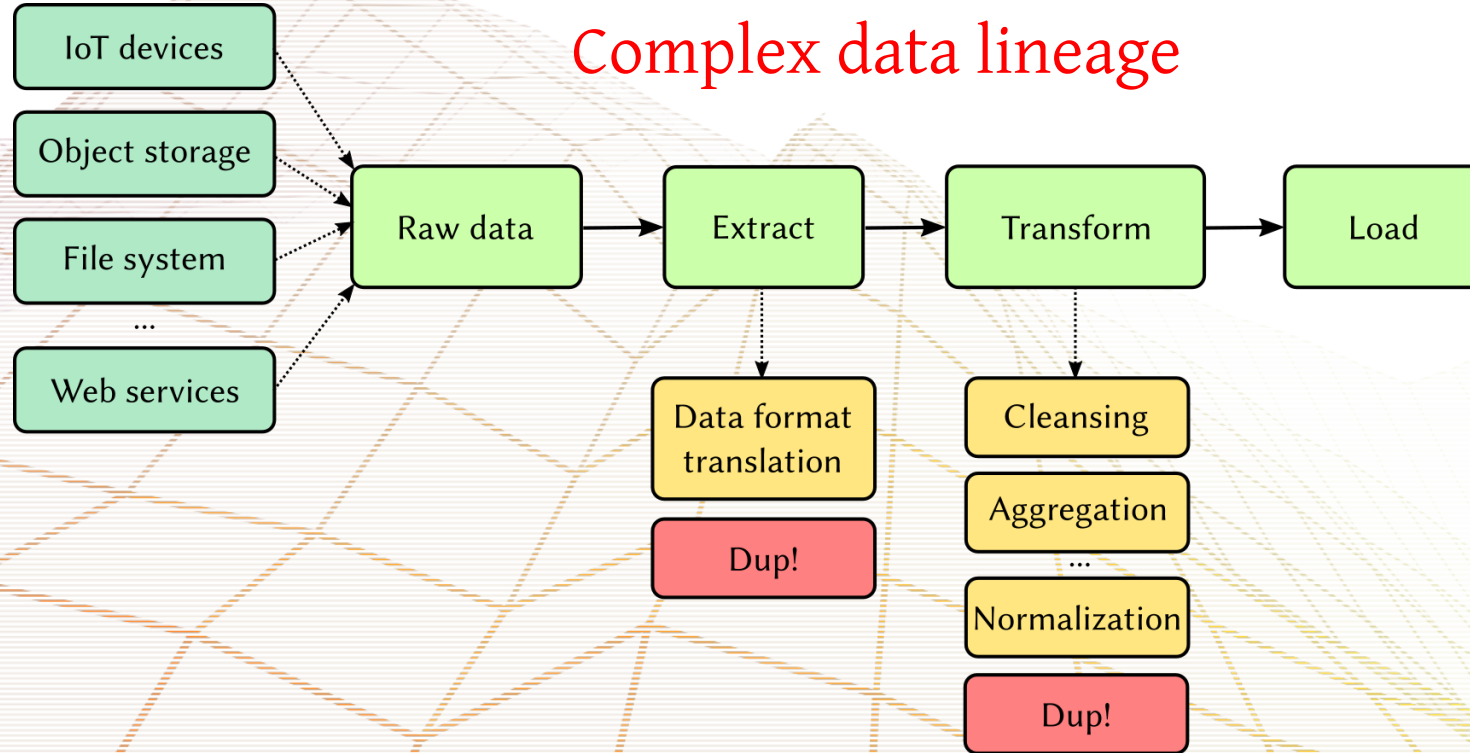
Traditional data pipeline



Traditional data pipeline

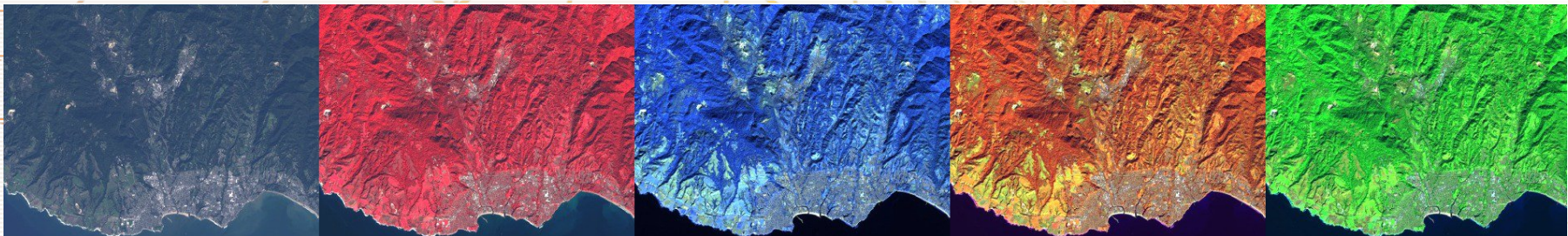


Traditional data pipeline

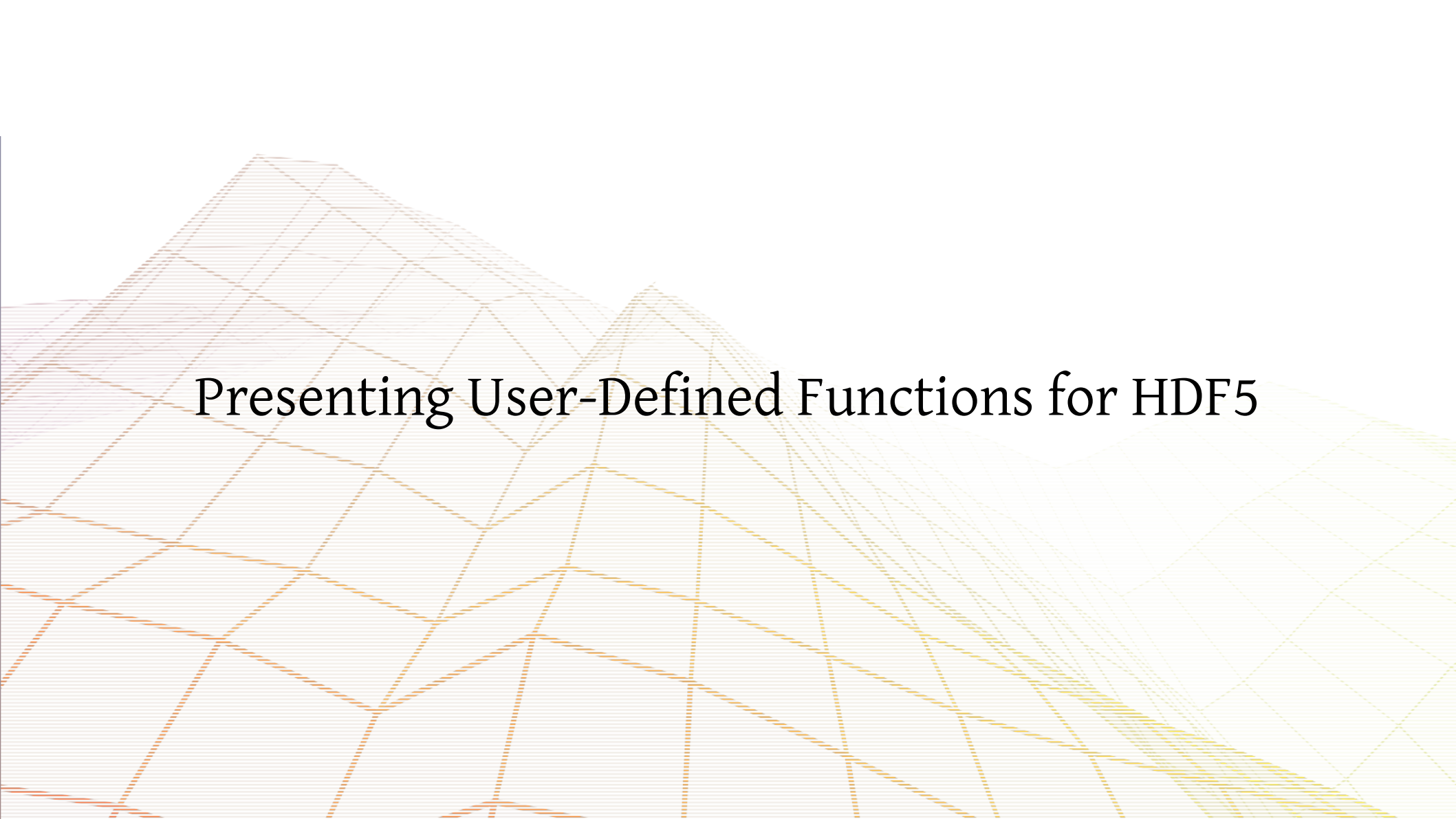


Many transformations are simple!

```
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)
```

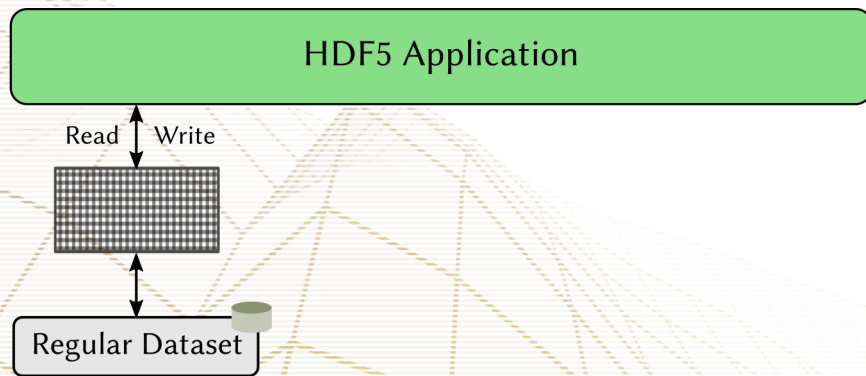


Credit: <https://twitter.com/jstnbraaten/status/995043633798660096>

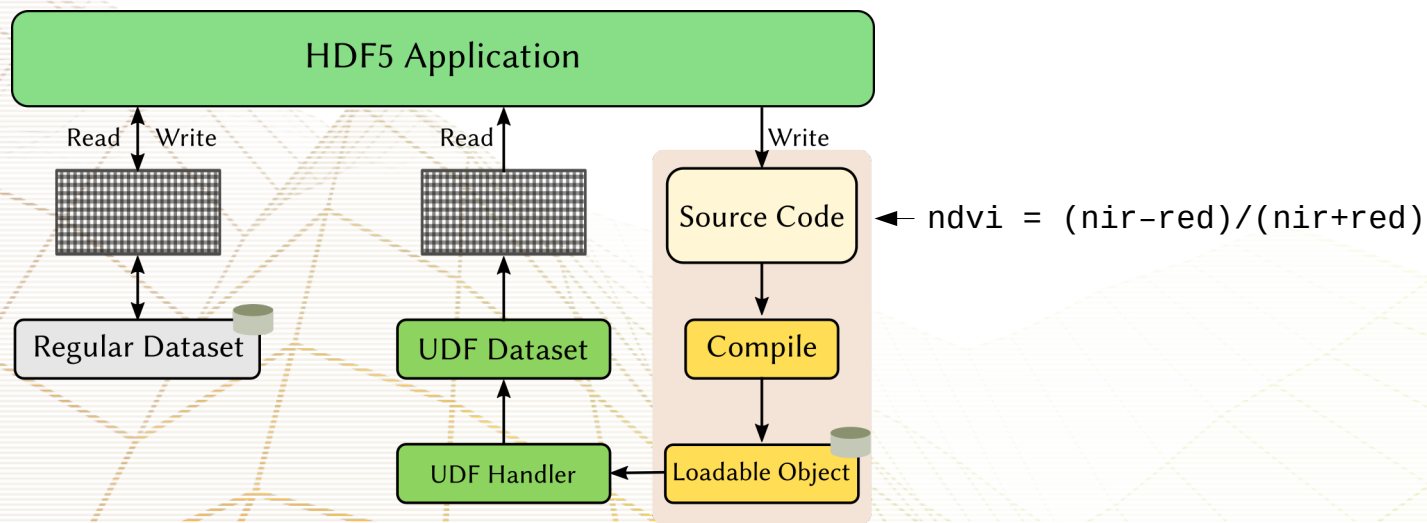


Presenting User-Defined Functions for HDF5

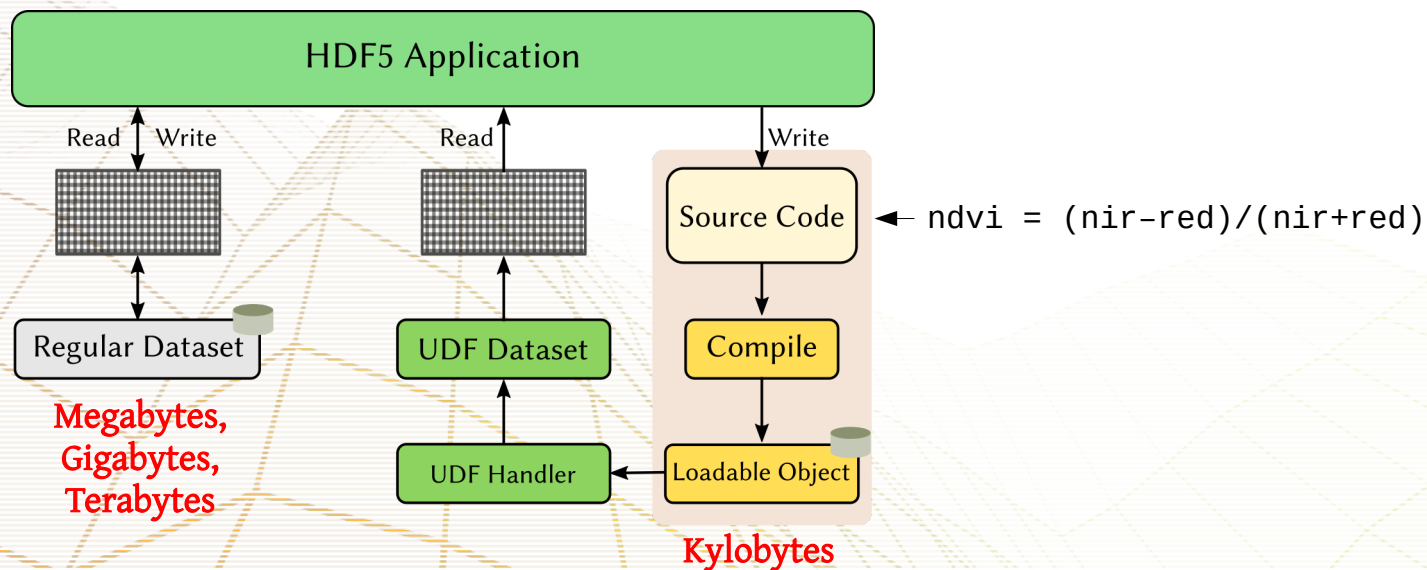
Traditional HDF5 application flow



HDF5 application flow supported by UDFs

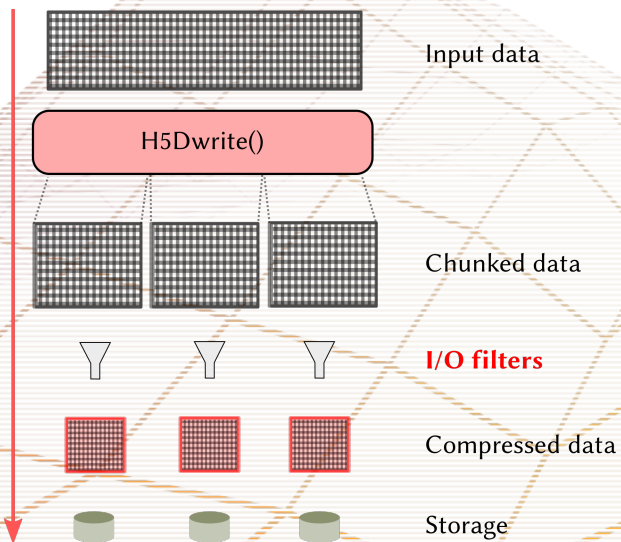


HDF5 application flow supported by UDFs



Data transformation is performed **when needed**

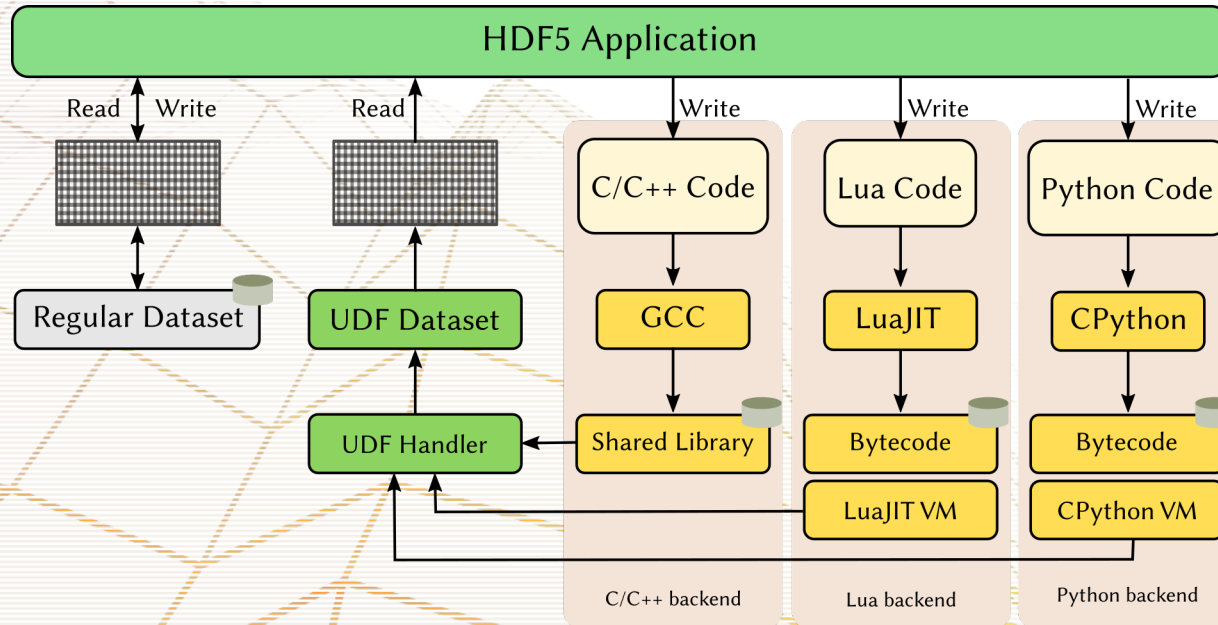
The technology underneath



HDF5 I/O filters

- Loadable plugins: deflate, bzip2, etc
- Operate on chunks of data
- Write path: compresses data
- Read path: reads decompressed

The big picture, as of HDF5-UDF 1.2



Three languages, one API

Python: `def dynamic_dataset()`
C/C++: `void dynamic_dataset()`
Lua: `function dynamic_dataset()`

`lib.getData("DatasetName")` → `pointer` to data (FFI)

`lib.getDims("DatasetName")` → `array`
`[1024, 768]`

`lib.getType("DatasetName")` → `string`
`int16, int32, int64, uint16, uint32, uint64, float, double, string, compound`

`lib.setString(compound.member, "Value")`

Python

```
1 def dynamic_dataset()  
2     a, b, c = lib.getData("A"), lib.getData("B"), lib.getData("C")  
3     size = lib.getDims("A")[0] * lib.getDims("A")[1]  
4  
5     for i in range(size):  
6         c[i] = a[i] + b[i]
```

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

Python

```
1 def dynamic_dataset()  
2     a, b, c = lib.getData("A"), lib.getData("B"), lib.getData("C")  
3     size = lib.getDims("A")[0] * lib.getDims("A")[1]  
4  
5     for i in range(size):  
6         c[i] = a[i] + b[i]
```

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

```
UDF dataset header = {  
    "backend": "CPython",  
    "bytecode_size": 879,  
    "input_datasets": ["A", "B"],  
    "output_dataset": "C",  
    "output_datatype": "float",  
    "output_resolution": [1024, 768]  
}
```

Python

```
1 def dynamic_dataset()  
2     a, b, c = lib.getData("A"), lib.getData("B"), lib.getData("C")  
3     size = lib.getDims("A")[0] * lib.getDims("A")[1]  
4  
5     for i in range(size):  
6         c[i] = a[i] + b[i]
```

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

```
UDF dataset header = {  
    "backend": "CPython",  
    "bytecode_size": 879,  
    "input_datasets": ["A", "B"],  
    "output_dataset": "C",  
    "output_datatype": "float",  
    "output_resolution": [1024, 768]  
}
```

C/C++

```
1 extern "C" void dynamic_dataset() {  
2     auto a = lib.getData<float>("A");  
3     auto b = lib.getData<float>("B");  
4     auto c = lib.getData<float>("C");  
5     auto size = lib.getDims("A")[0] * lib.getDims("A")[1];  
6  
7     for (auto i=0, i<size; ++i)  
8         c[i] = a[i] + b[i];  
9 }
```

```
# hdf5-udf file.h5 udf.cpp
```

Lua

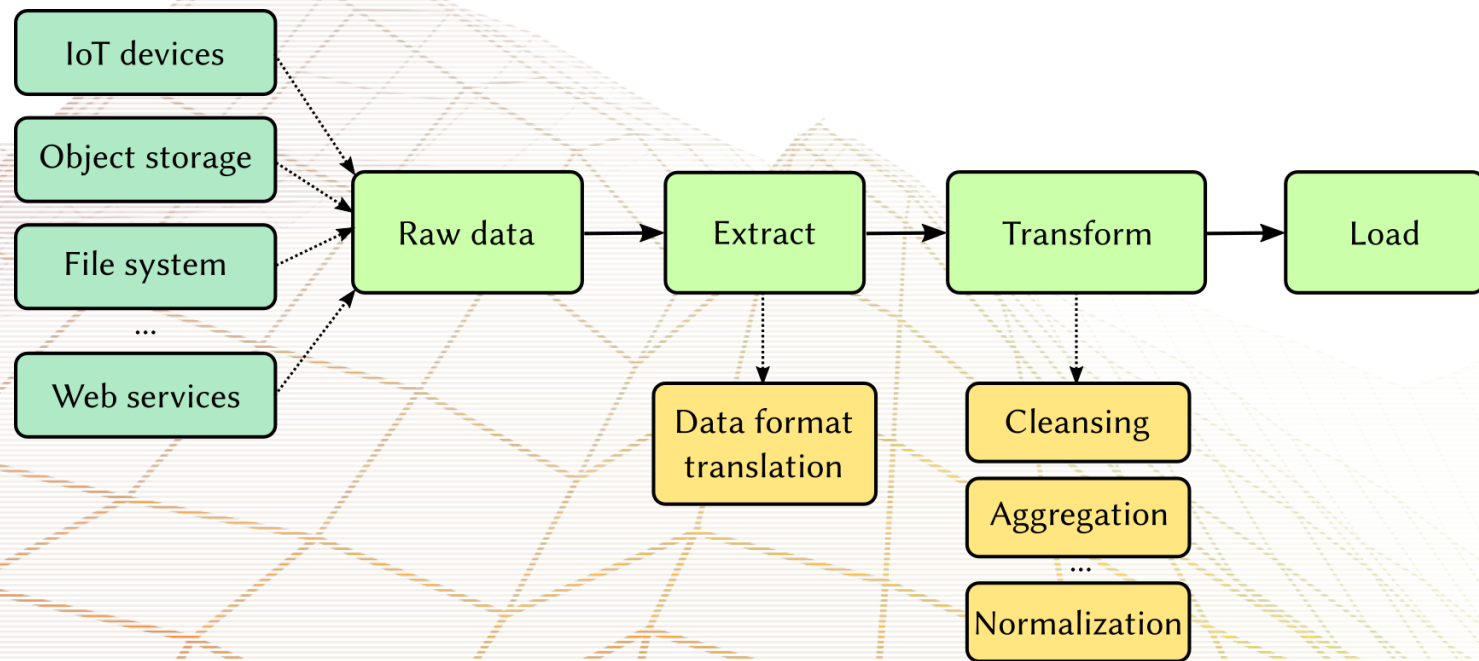
```
1 function dynamic_dataset()  
2     local a = lib.getData("A")  
3     local b = lib.getData("B")  
4     local c = lib.getData("C")  
5     local size = lib.getDims("A")[1] * lib.getDims("A")[2]  
6  
7     for i=1, size do  
8         c[i] = a[i] + b[i]  
9     end  
10 end
```

```
# hdf5-udf file.h5 udf.lua
```

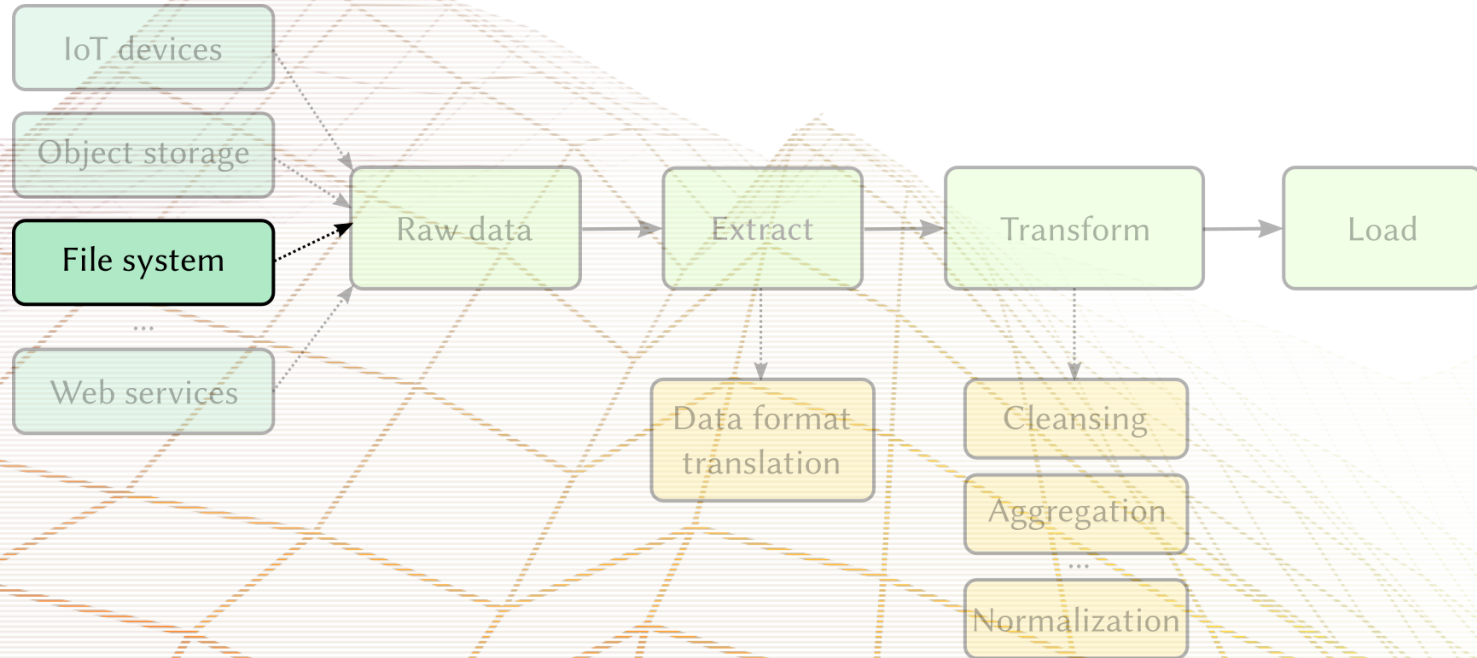


Back to our original example!

Traditional data pipeline

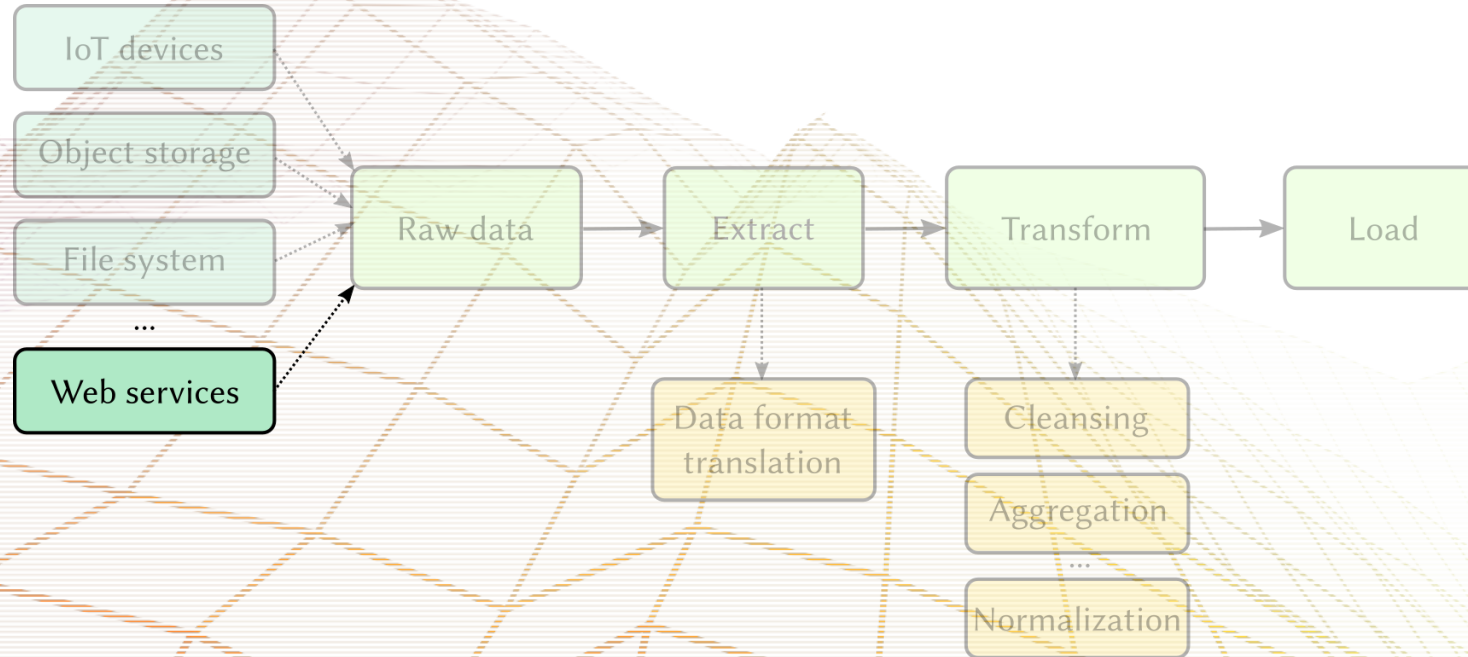


Traditional data pipeline



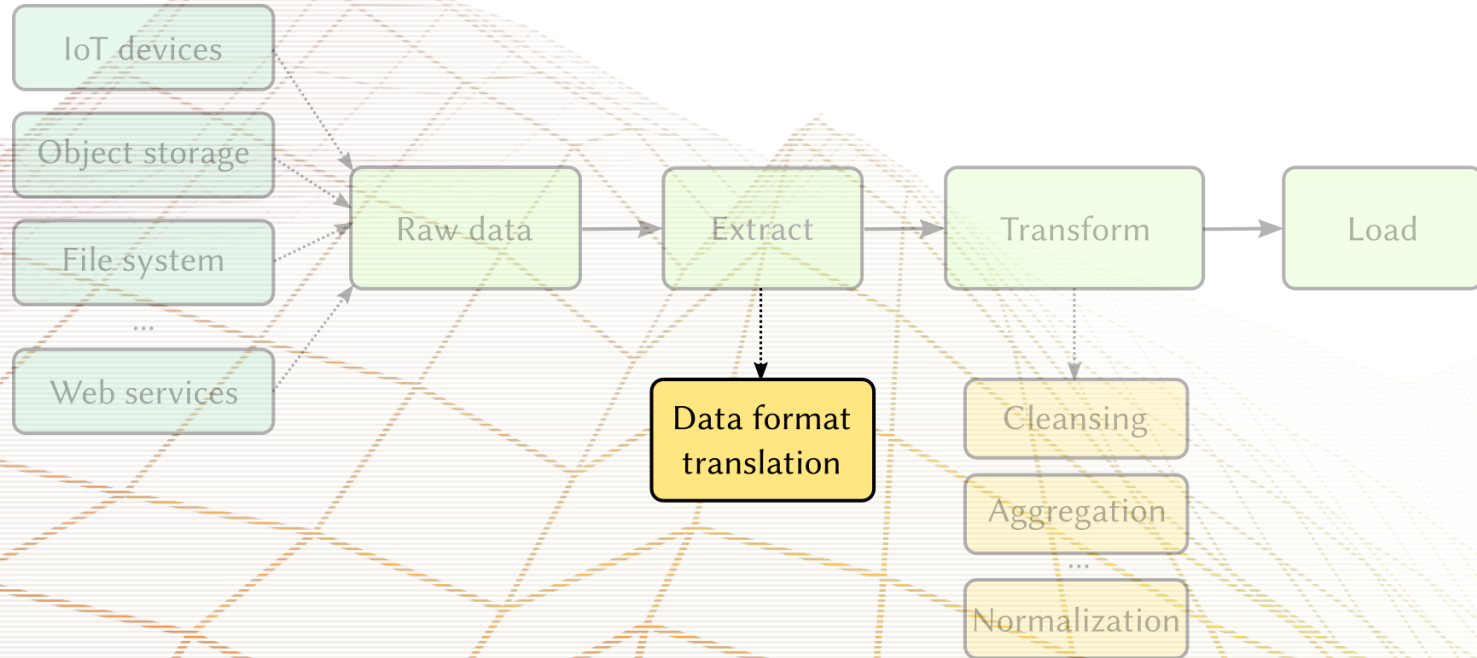
<this slide has an associated live demo>

Traditional data pipeline



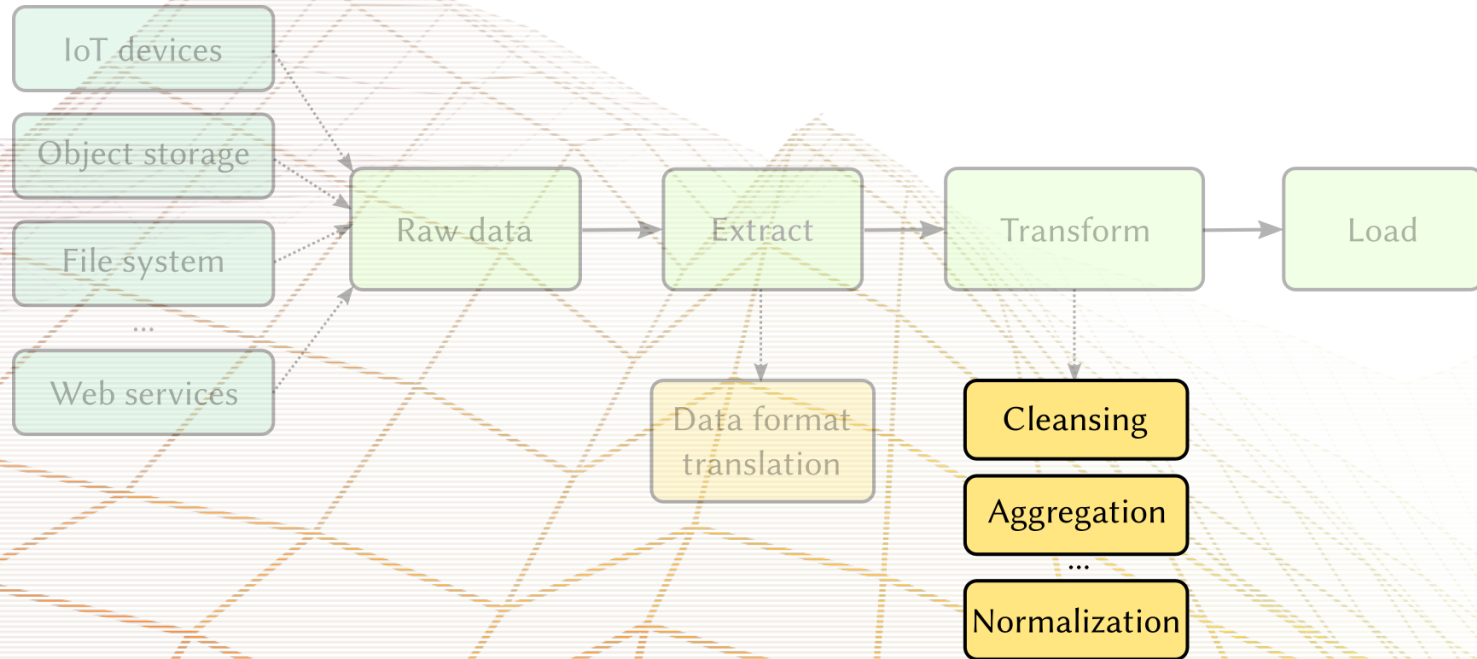
<this slide has an associated live demo>

Traditional data pipeline



<this slide has an associated live demo>

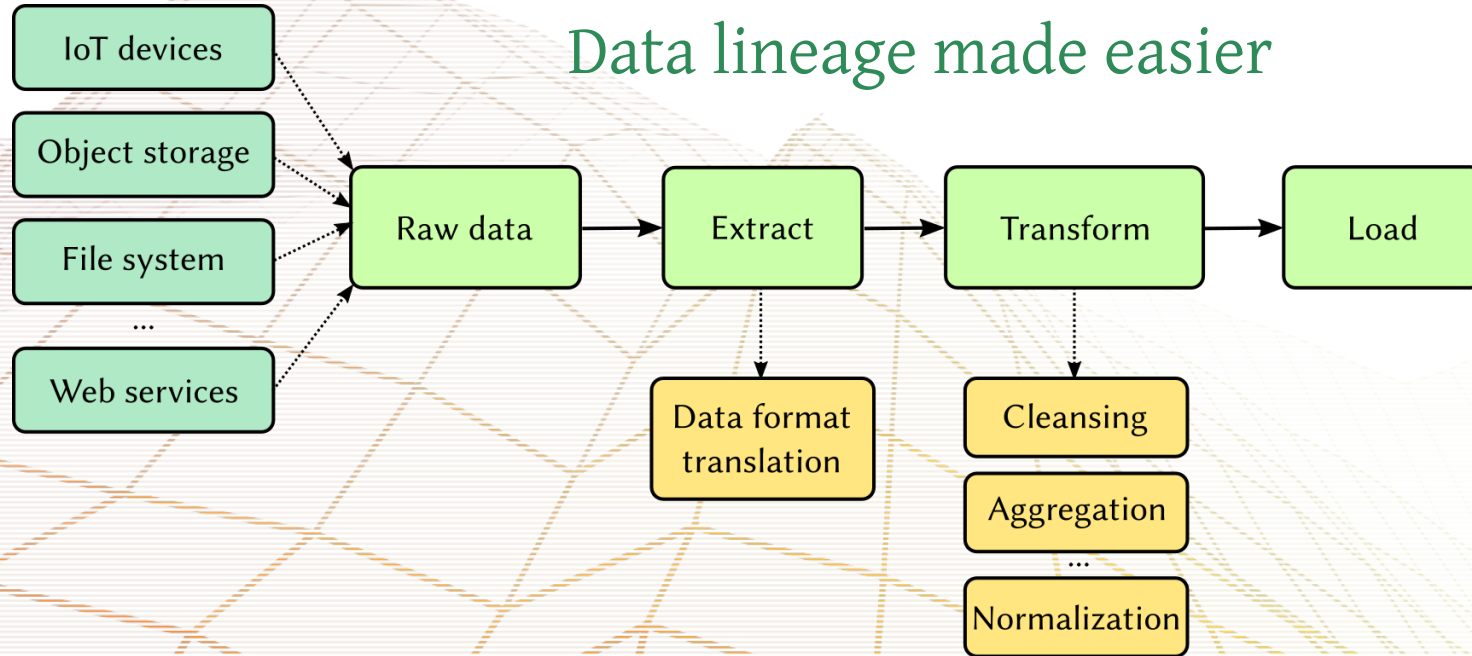
Traditional data pipeline



<this slide has an associated live demo>

Traditional data pipeline

Data lineage made easier





Security Considerations

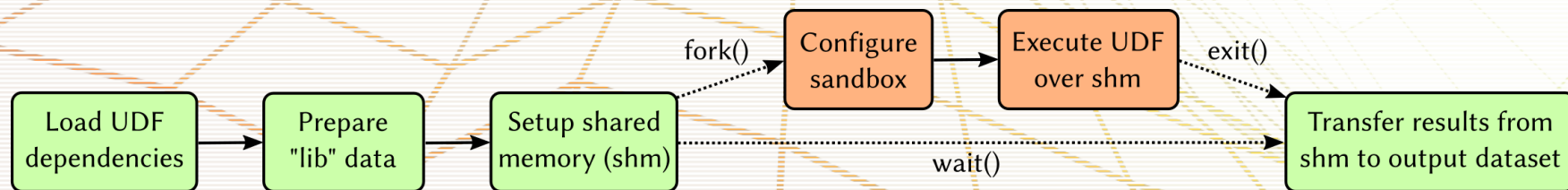
Would you trust third-party UDFs?

Seccomp

- Defines a list of system calls an UDF can execute
- Possible to specify valid syscall arguments (strings not supported)
- Kills the UDF if it violates the rules

Syscall-Intercept Library

- Intercepts allowed system calls
- Enables filtering of string-based arguments



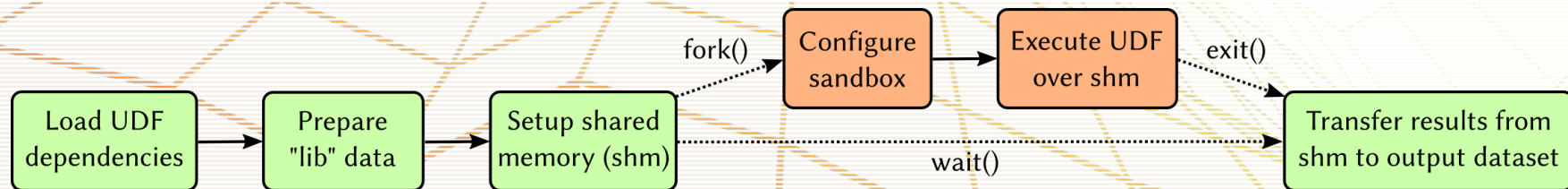
Would you trust third-party UDFs?

Seccomp: **Linux only**

- Defines a list of system calls an UDF can execute
- Possible to specify valid syscall arguments (strings not supported)
- Kills the UDF if it violates the rules

Syscall-Intercept Library: **Glibc, x86-64 only**

- Intercepts allowed system calls
- Enables filtering of string-based arguments



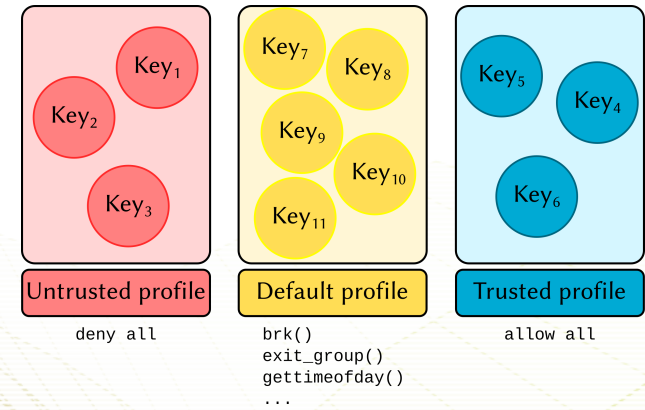
Ongoing effort: security profiles

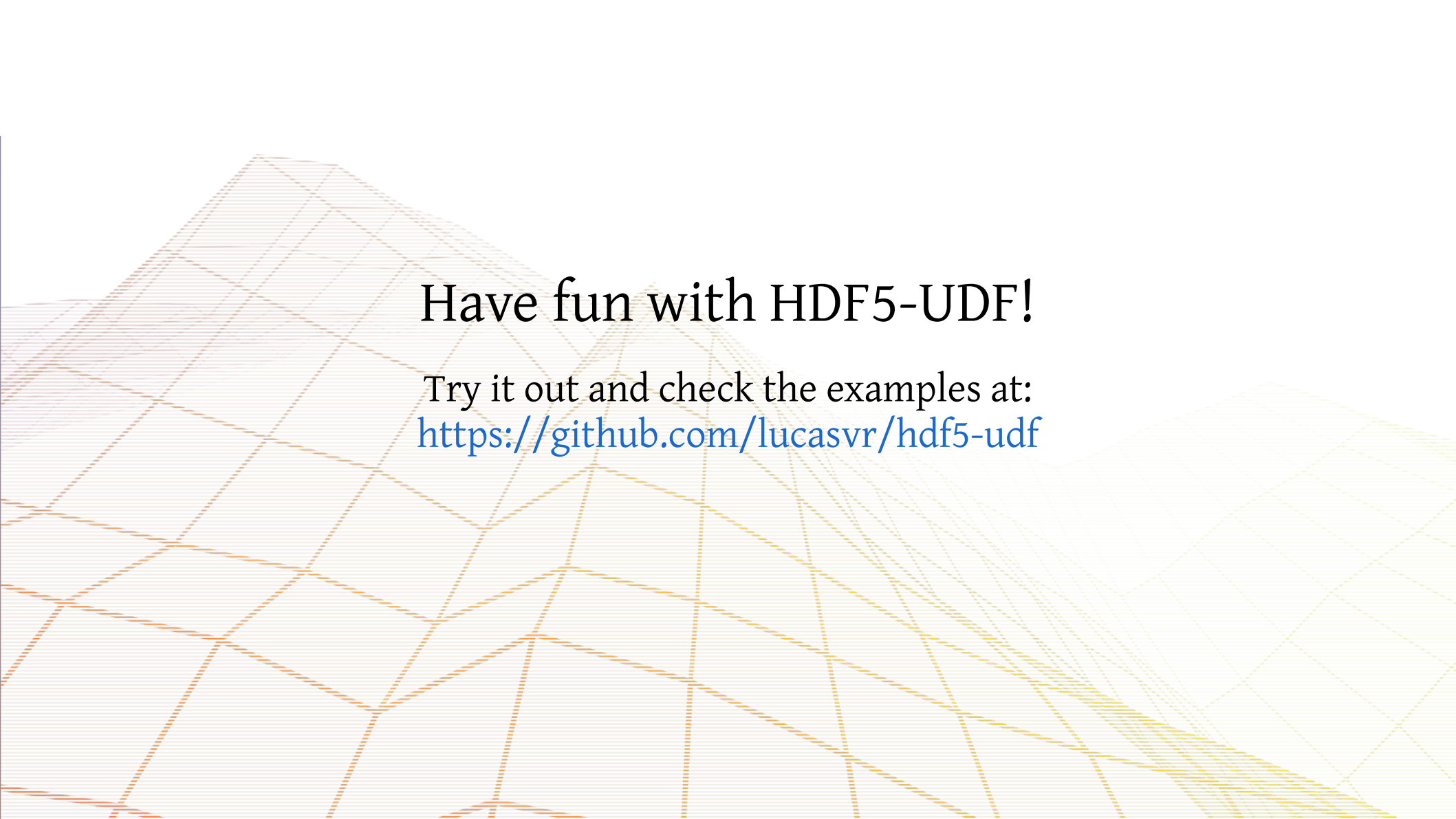
Writing UDFs

- User signs UDF with their private key
- Public key is shipped as metadata

Reading UDFs

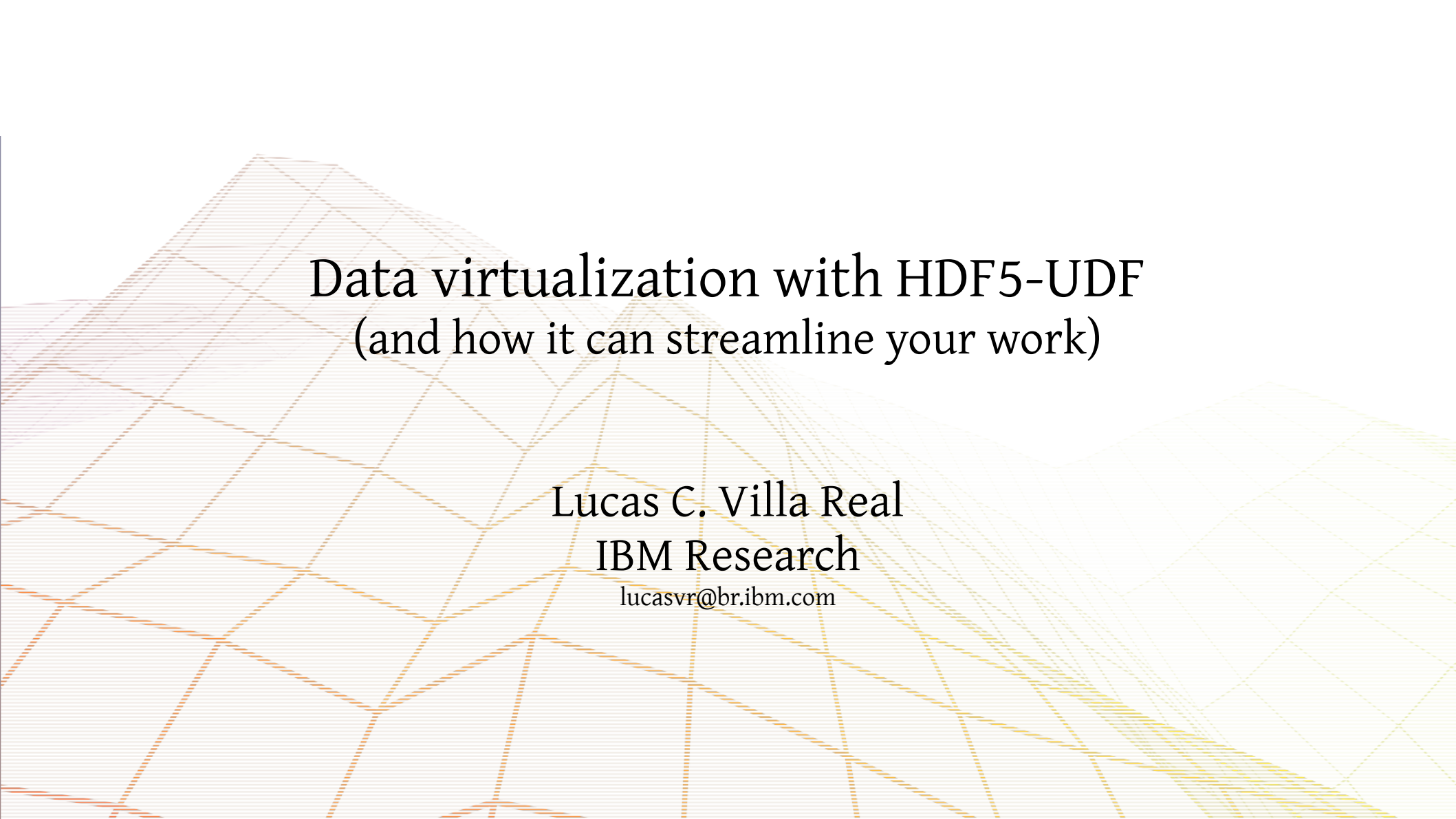
- Public keys are associated with security profiles
- Each profile has a configurable list of allowed system calls





Have fun with HDF5-UDF!

Try it out and check the examples at:
<https://github.com/lucasvr/hdf5-udf>



Data virtualization with HDF5-UDF

(and how it can streamline your work)

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