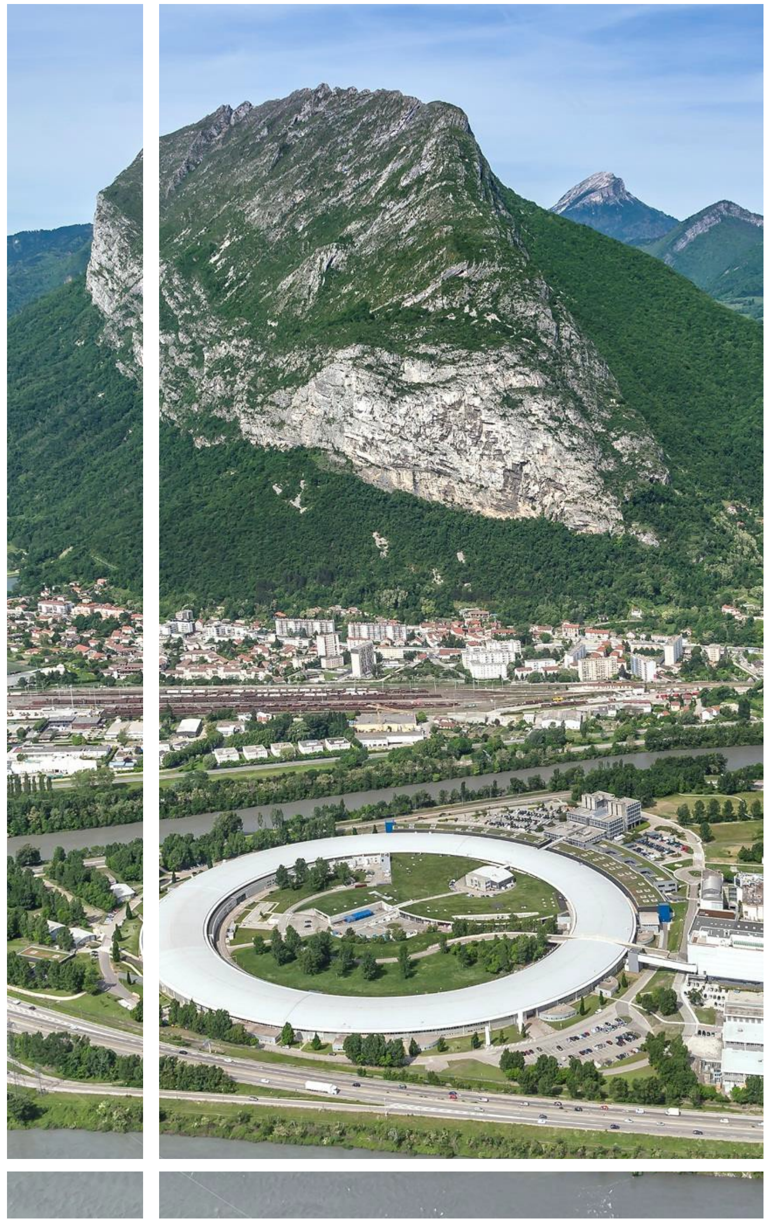
ESRF | The European Synchrotron



Blissdata, unified storage for online/offline acquisition data

Lucas FELIX
ESRF (Beamline Control Unit)

RECAP: ESRF SOFTWARE ECOSYSTEM

Bliss
control

Lima
2D detector acquisition

flint
visualization for desktop

ewoks
automation & data processing

daiquiri
web visualization and control

RECAP: ESRF SOFTWARE ECOSYSTEM

Bliss
control

Lima
2D detector acquisition

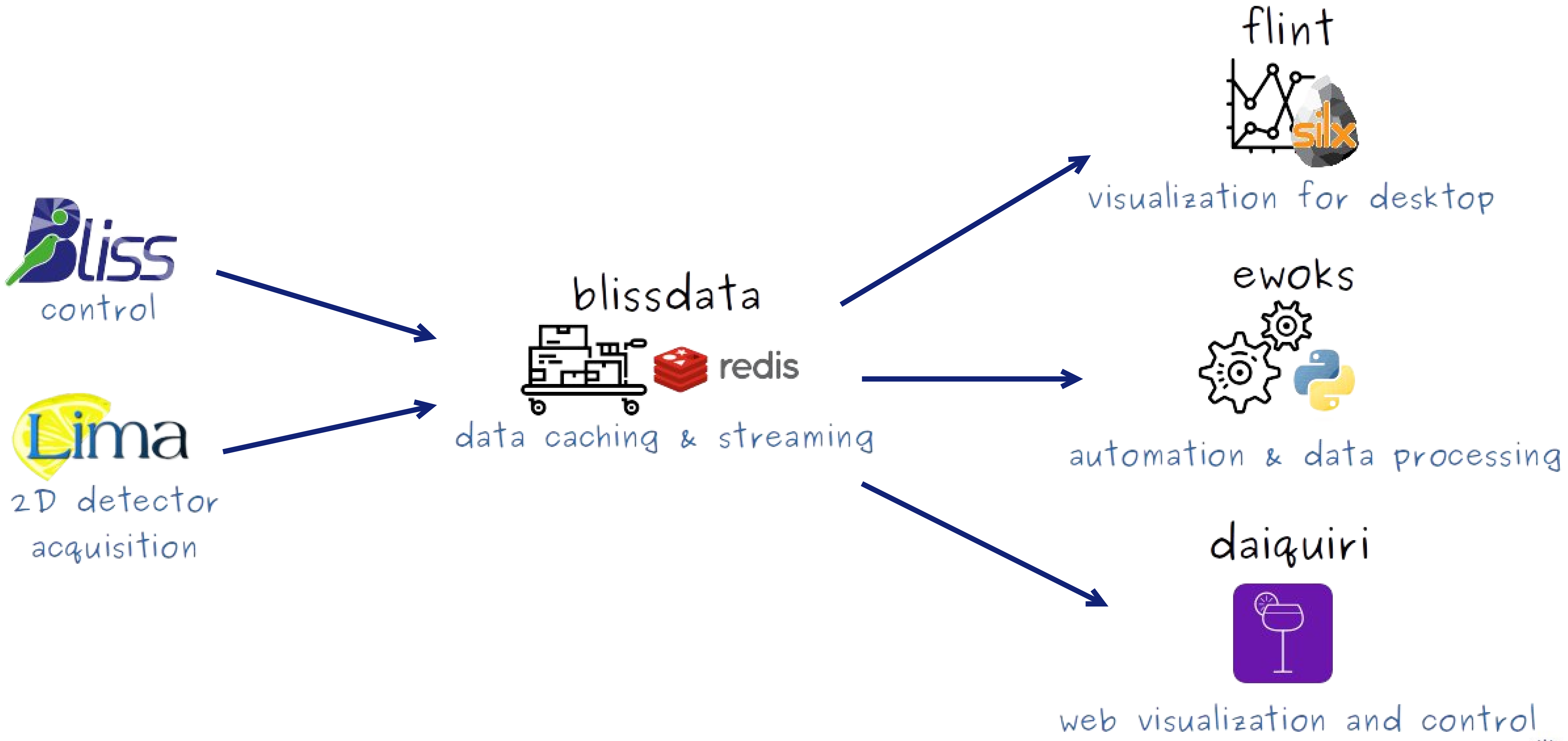
- Where is the data for this detector ?
- Is it available now ?
- Can someone tell me when it is ?
- Did the data move to another place ? (buffer / file)

flint
visualization for desktop

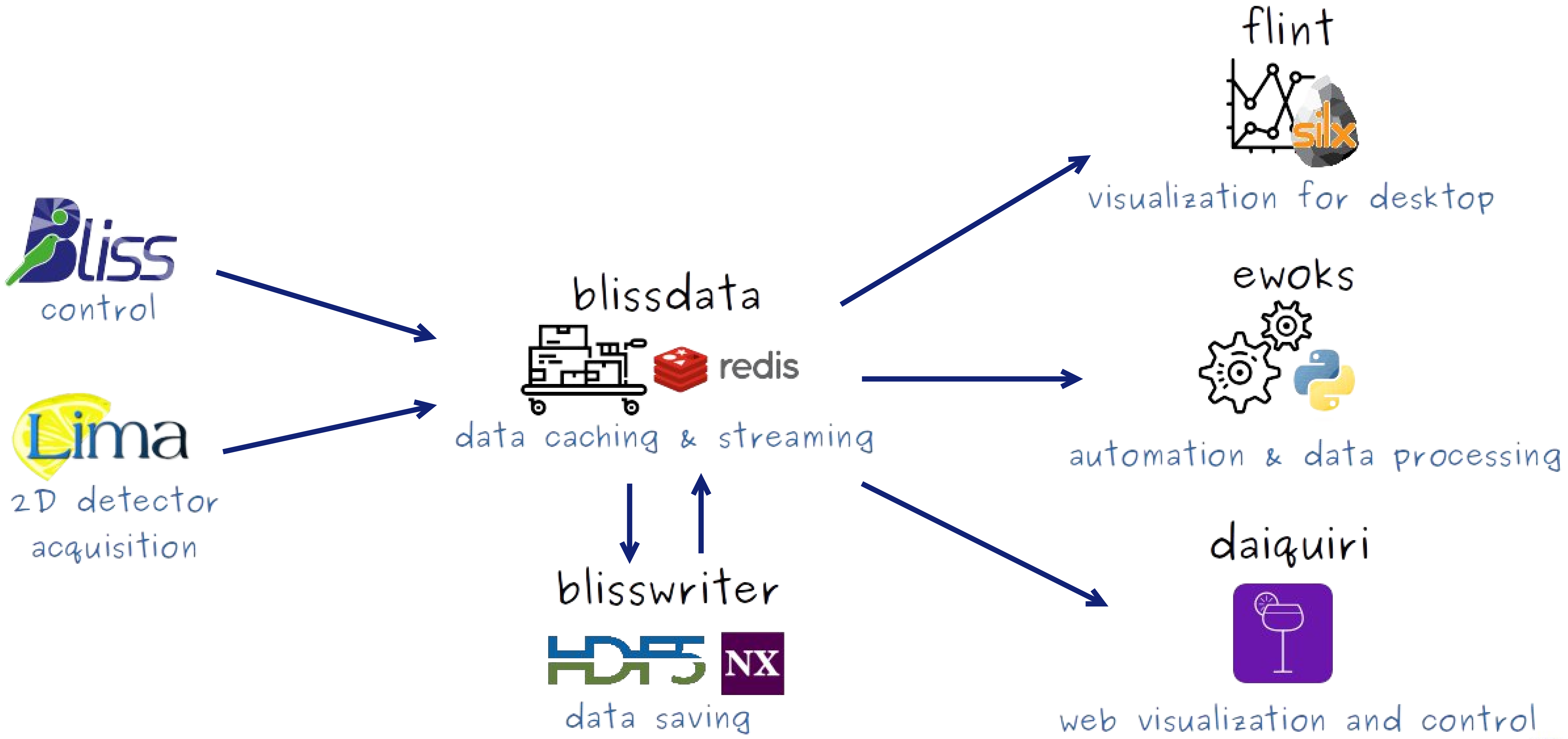
ewoks
automation & data processing

daiquiri
web visualization and control

RECAP: ESRF SOFTWARE ECOSYSTEM



RECAP: ESRF SOFTWARE ECOSYSTEM



BLISSDATA: AN INDEPENDENT PACKAGE

- Blissdata is an independent python package
- Few dependencies
- Available with **Pip** and **Conda**

- It requires a running **Redis** database (RAM) as its internal data buffer

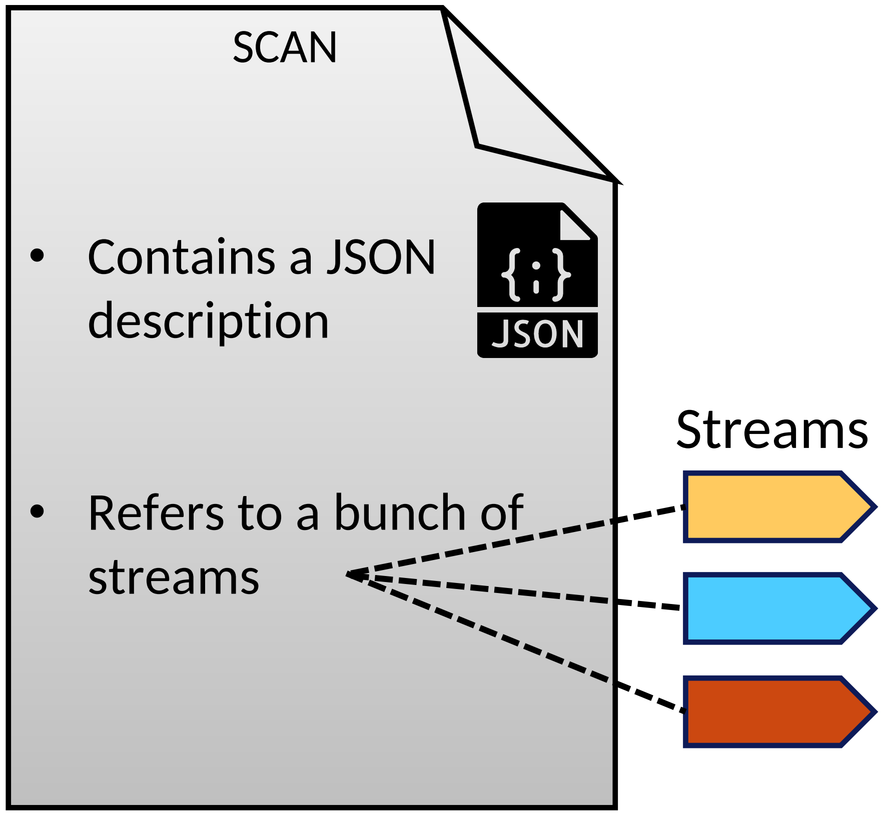
https://pypi.org/project/blissdata/

https://anaconda.org/esrf-bcu/blissdata

BLISSDATA: SCANS AND STREAMS

In **blissdata** everything is built around **Scan** and **Stream**

A **Scan** is like a header on top of a bunch of **streams**, with metadata in JSON format.

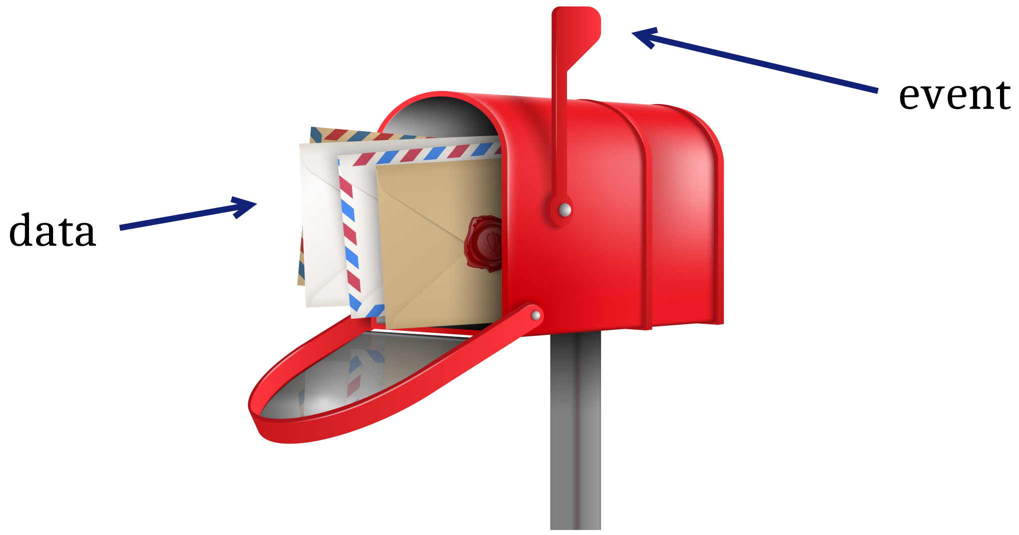


Blissdata getting started:
https://bliss.gitlab-pages.esrf.fr/bliss/master/blissdata/getting_started.html

BLISSDATA: DECOUPLING EVENT AND DATA

Key principle

Inside the streams, data is decoupled from the events.



Events always go through **Redis**, because it is **fast**, **deterministic** and **multiplexable**.

BLISSDATA: INTERNAL/EXTERNAL DATA PATH

Stream’s data, on the other hand, can take different paths.

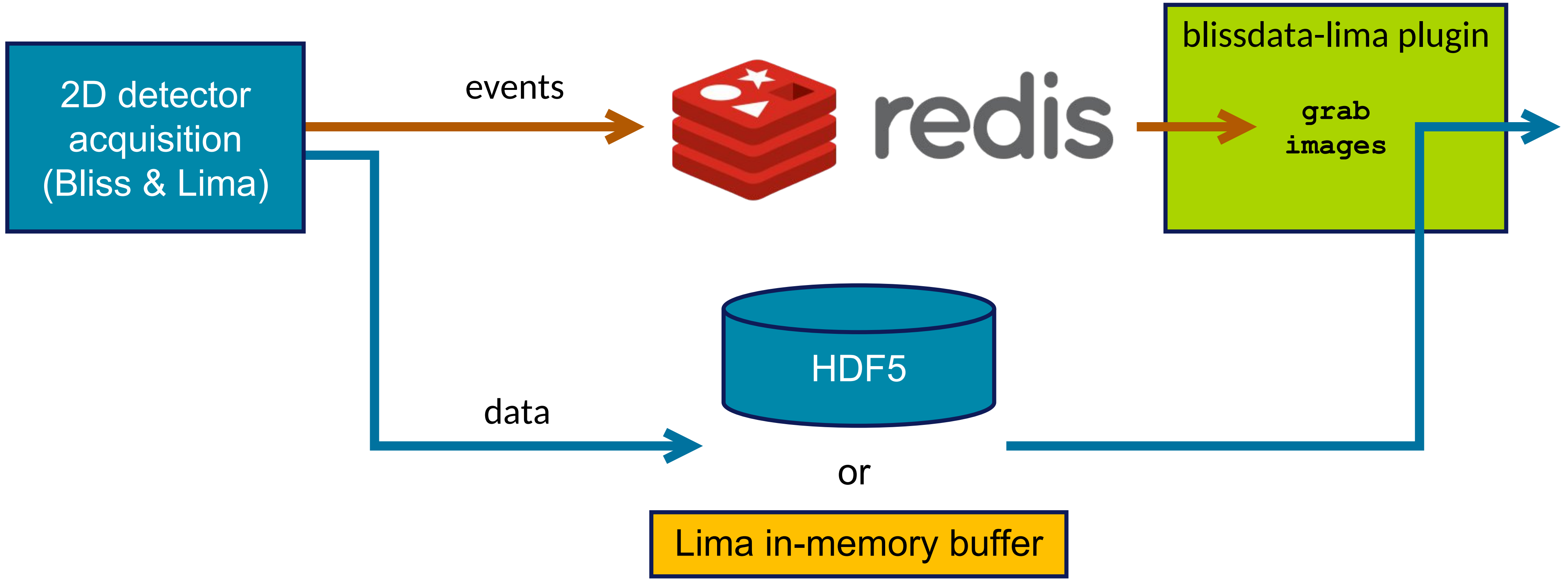
Simple case:

It is small enough to fit into **Redis**, simply put the data inside events.

Others *(Data is too large, we store it somewhere else, …)*:

Make a **plugin** to grab the data when events arrive.

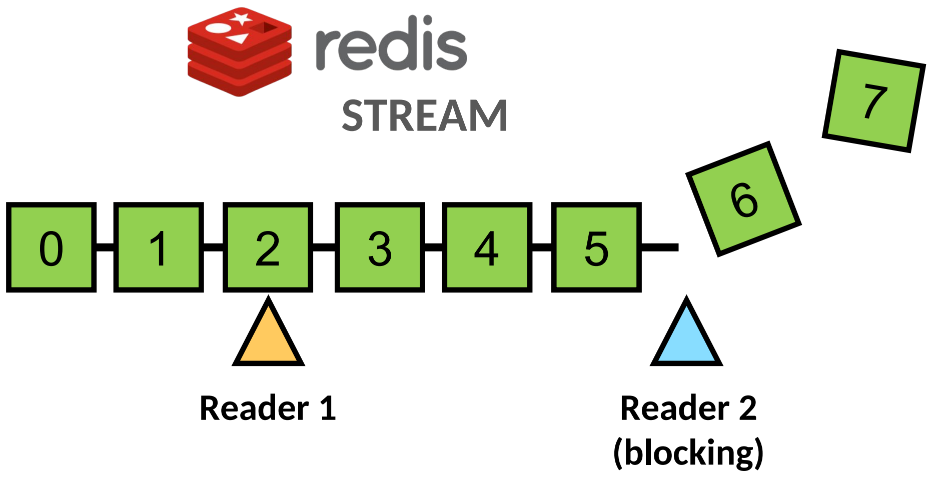
EXAMPLE: EXTERNAL IMAGE PLUGIN



PARENTHESIS ON REDIS PERSISTENCY

Redis streams are persisted, there is no need to keep listening constantly.

You can read previous events as an array, or perform blocking read to wait for next ones.



A reader can read:

- at any time
- at any speed
- any number of streams at once

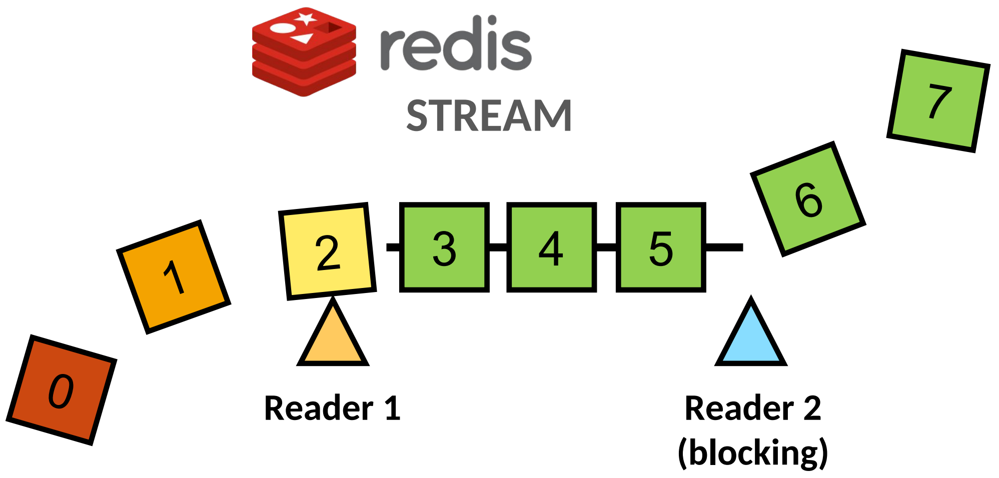
⇨

Redis streams alleviate the need of multitasking by design, which greatly **reduce code complexity**.

PARENTHESIS ON REDIS PERSISTENCY

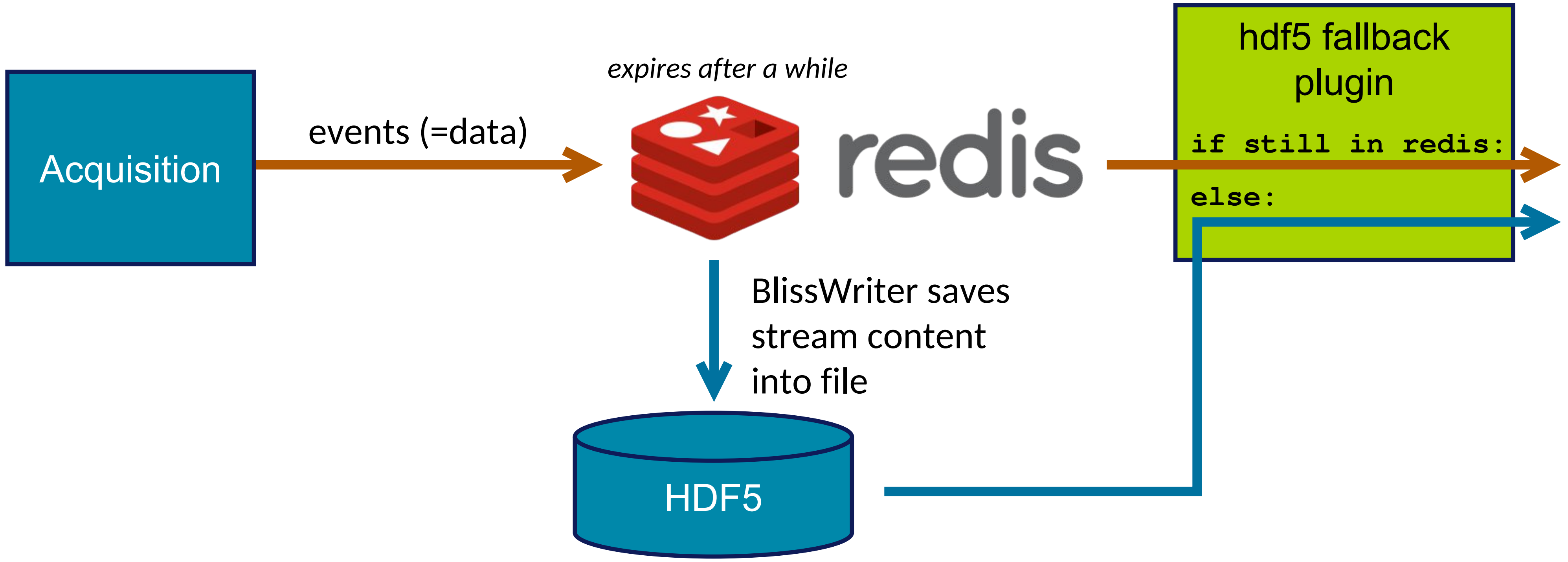
A daemon process called **memory tracker** keeps freeing space inside Redis.
This may cause readers to fail when reading old events.

Stream content is gradually **discarded** to release memory inside Redis.



Index 2 is **expired**, reader 1 will **fail** to read !

EXAMPLE: FILE FALLBACK PLUGIN



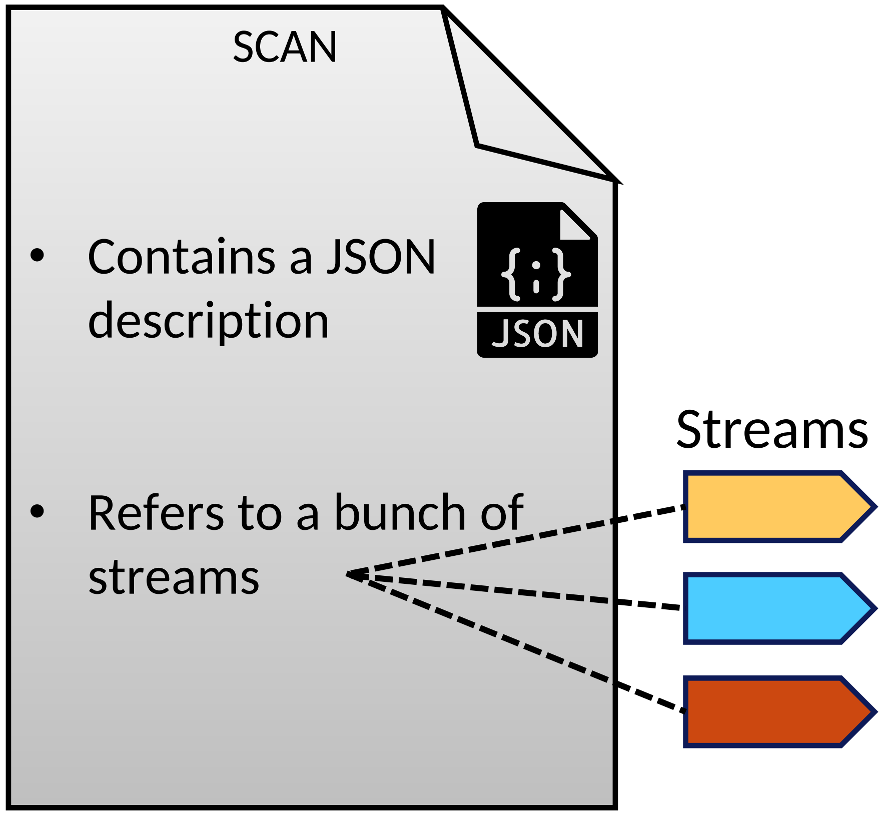
BLISSDATA: PLUGINS

redis
HDF5
Lima
TANGO
• • •

- Unify heterogeneous storage under a common API
- Built-in persistence, can be extended by fallback plugins
- Rely on Redis for stream multiplexing (no multitasking required)
- Redis data rate is not the limit
- Gain flexibility by decoupling users from the data back-ends

TAKING A LOOK BACK AT THE SCANS

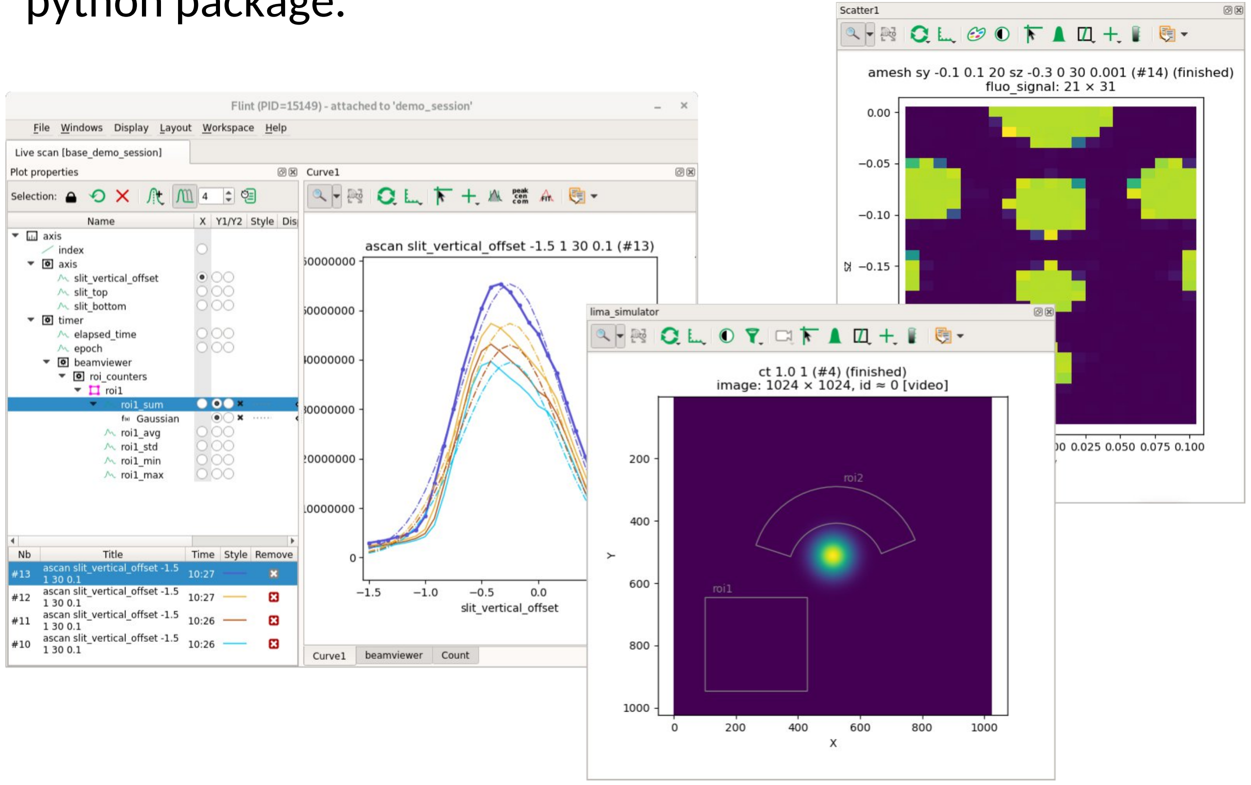
- We know how to transport data with streams
- What makes it interpretable by others tools ?



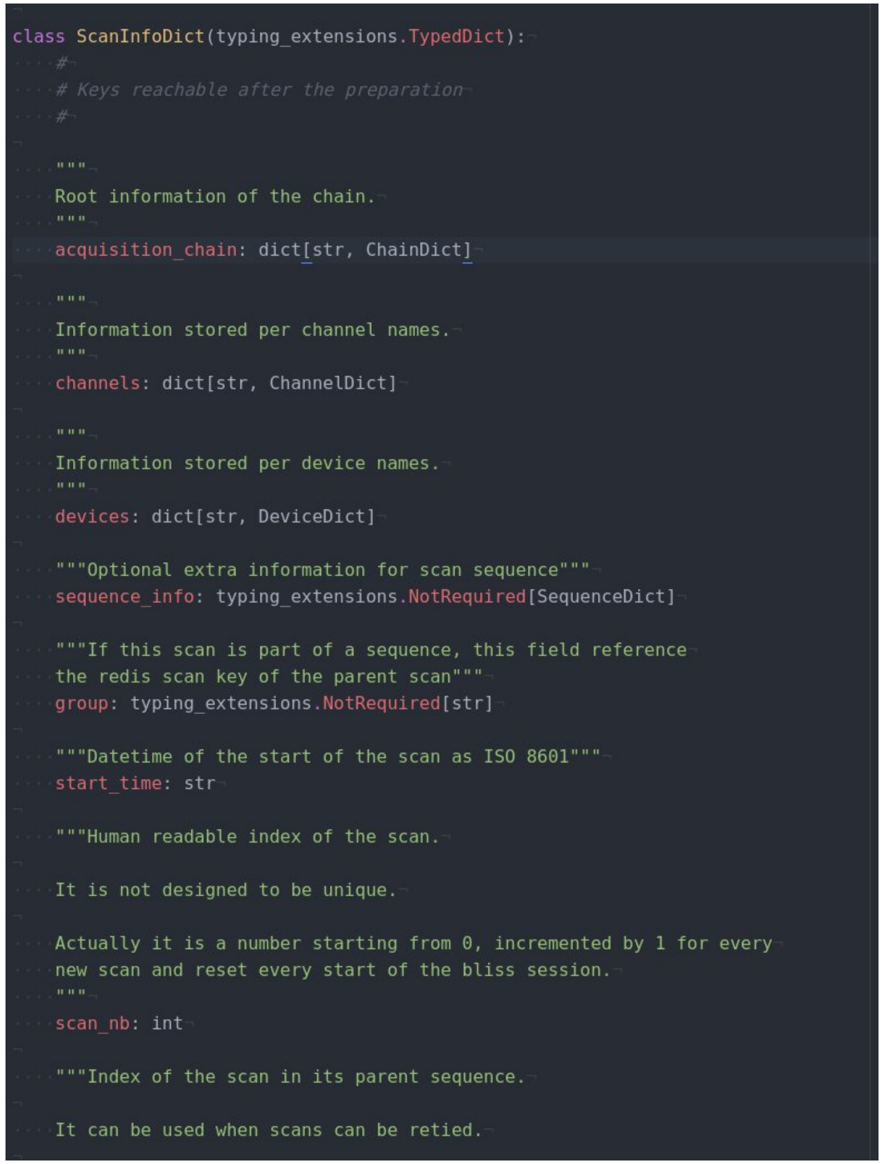
BLISSDATA: JSON FORMAT

Tools like **Flint** or **Blisswriter** requires specific information in the **JSON header** so they know what streams do represent and what to do with them.

JSON schemas are available in **/blissdata/schemas** of the python package.



Flint (PID=15149) - attached to 'demo_session'
File Windows Display Layout Workspace Help
Live scan [base_demo_session]
Plot properties
Curve1
ascan slit_vertical_offset -1.5 1 30 0.1 (#13)
slit_vertical_offset
Scatter1
amesh sy -0.1 0.1 20 sz -0.3 0 30 0.001 (#14) (finished)
fluo_signal: 21 × 31
lima_simulator
ct 1.0 1 (#4) (finished)
image: 1024 × 1024, id ≈ 0 [video]
roi1
roi2
Curve1 beamviewer Count

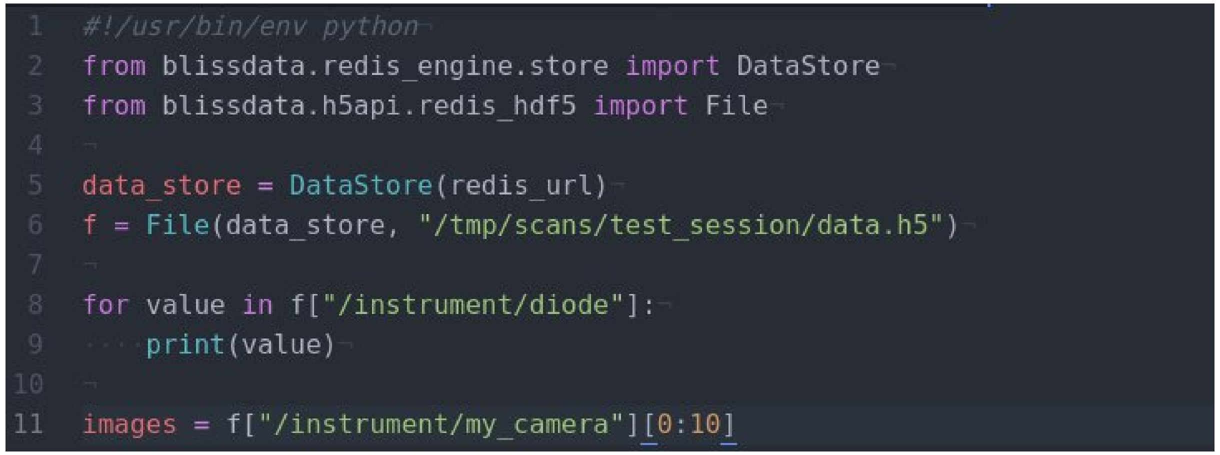


```
class ScanInfoDict(typing_extensions.TypedDict):
    #
    # Keys reachable after the preparation
    #

    """
    Root information of the chain.
    """
    acquisition_chain: dict[str, ChainDict]

    """
    Information stored per channel names.
    """
    channels: dict[str, ChannelDict]

    """
    Information stored per device names.
    """
    devices: dict[str, DeviceDict]

    """Optional extra information for scan sequence"""
    sequence_info: typing_extensions.NotRequired[SequenceDict]

    """If this scan is part of a sequence, this field reference
    the redis scan key of the parent scan"""
    group: typing_extensions.NotRequired[str]

    """Datetime of the start of the scan as ISO 8601"""
    start_time: str

    """Human readable index of the scan.

    It is not designed to be unique.

    Actually it is a number starting from 0, incremented by 1 for every
    new scan and reset every start of the bliss session.
    """
    scan_nb: int

    """Index of the scan in its parent sequence.

    It can be used when scans can be retied.
```

BLISSDATA: H5PY-LIKE API (CURRENTLY IN DEVELOPMENT)

In each scan, we embed its future HDF5 layout.

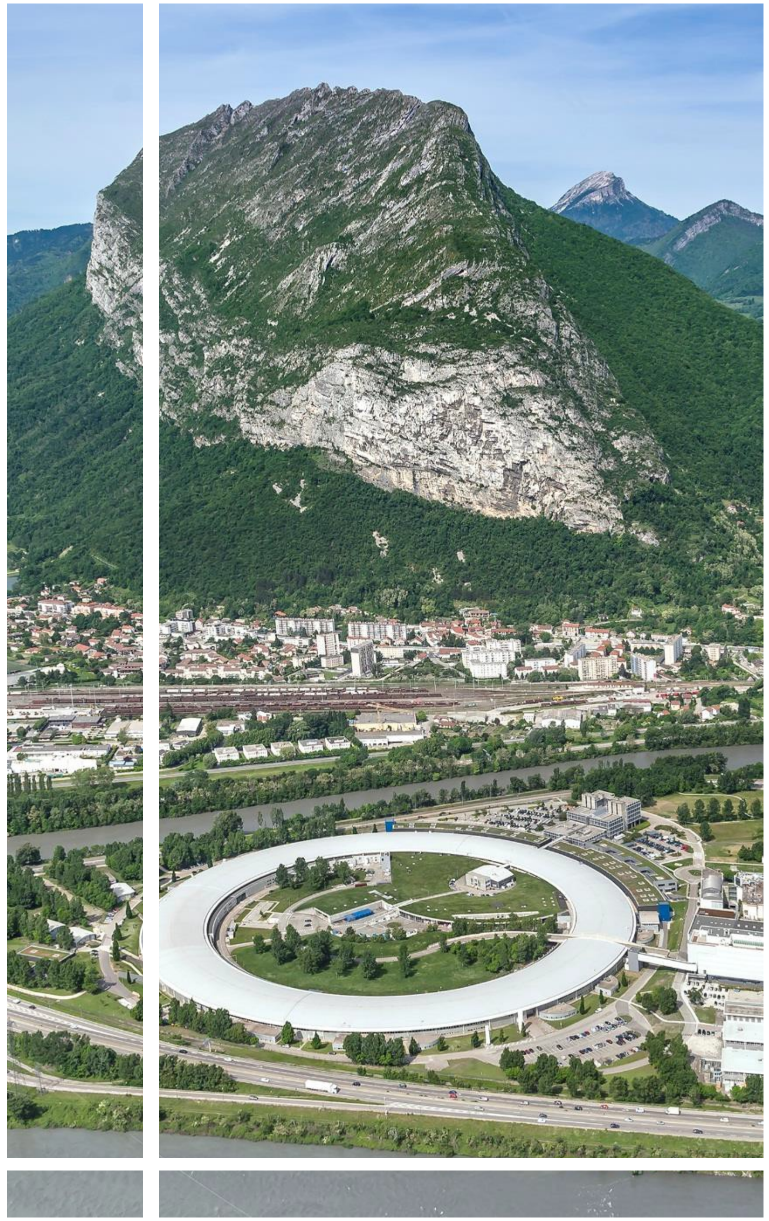
Blissdata would then be able to expose scans with the same API as **h5py** from the very beginning of the scan.



```python
#!/usr/bin/env python
from blissdata.redis_engine.store import DataStore
from blissdata.h5api.redis_hdf5 import File

data_store = DataStore(redis_url)
f = File(data_store, "/tmp/scans/test_session/data.h5")

for value in f["/instrument/diode"]:
    print(value)

images = f["/instrument/my_camera"][0:10]
```

```python
scan.info["mapping"] = (
    {
        "NX_class": "NXentry",
    },
    {
        "instrument": (
            {
                "NX_class": "NXinstrument",
            },
            {
                "name": ({"short_name": "id00"}, RAW, "esrf-id00a"),
                "diode": (
                    {"NX_class": "NXdetector"},
                    {"data": ({}, STREAM, "simulation_diode_sampling_controller:diode")},
                ),
                "elapsed_time": (
                    {"NX_class": "NXpositioner"},
                    {"value": ({"units": "s"}, STREAM, "timer:elapsed_time")},
                ),
                "epoch": (
                    {"NX_class": "NXpositioner"},
                    {"value": ({"units": "s"}, STREAM, "timer:epoch")},
                ),
            },
        ),
        "measurement": (
            {},
            {
                "diode": ({}, STREAM, "simulation_diode_sampling_controller:diode"),
                "elapsed_time": ({"units": "s"}, STREAM, "timer:elapsed_time"),
                "epoch": ({"units": "s"}, STREAM, "timer:epoch"),
            },
        ),
    },
)
```

CONCLUSION



Blissdata

- ↪ Unify the way software access data:
  - from heterogeneous sources
  - during and after acquisition
- ↪ Built-in intermediate buffer and persistence
- ↪ Plugins can further extend transfer speed and persistence
- ↪ Flexible architecture (we could swap the way we distribute 2D data without users noticing)
- ↪ Enable early h5py-like access to post-process the same way as files

Questions ?