CERN openlab

PyXRootD PyPI distribution & new declarative file access API for XRootD client

Krzysztof Jamróg

Supervisors: Michał Simon, Lars Nielsen

What is XRootD?

- High performance data access framework
- Plugin based
- Used by many frameworks e.g. :
    - Caching proxy servers
    - EOS / CTA
    - ROOT / Athena
- Client - server architecture:
    - server - responsible for storage aggregation and providing API to access data from these storages
    - client - used to connect to the server from local machine, contains python bindings (PyXRootD)

PyXRootD PyPI distribution

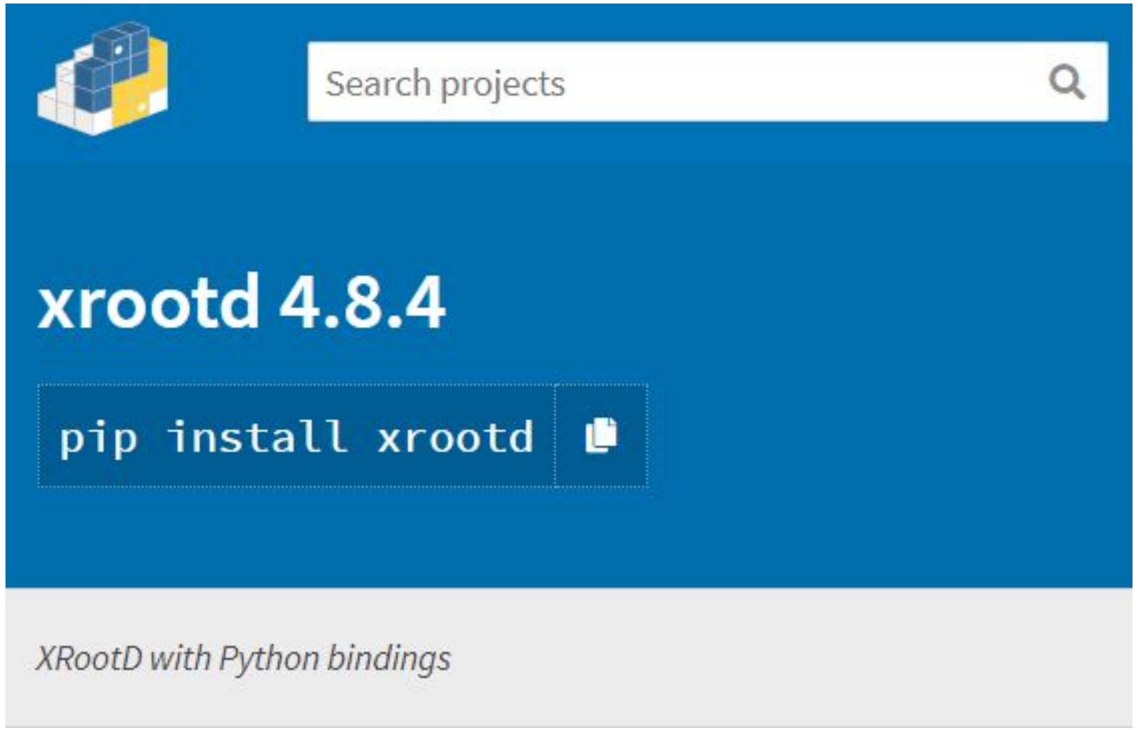
Problems with PyXRootD installation process

- Lack of support for python virtual environments
- Problem with installing specific version
- Installation process different than in case of other python packages

PyPI distribution

Publishing xrootd in Python Package Index

- Easy installation process:
  pip install xrootd
- Versioning
- Possibility to use requirements.txt for installing xrootd
- Publishing integrated with GitLab CI



xrootd 4.8.4

```
pip install xrootd
```

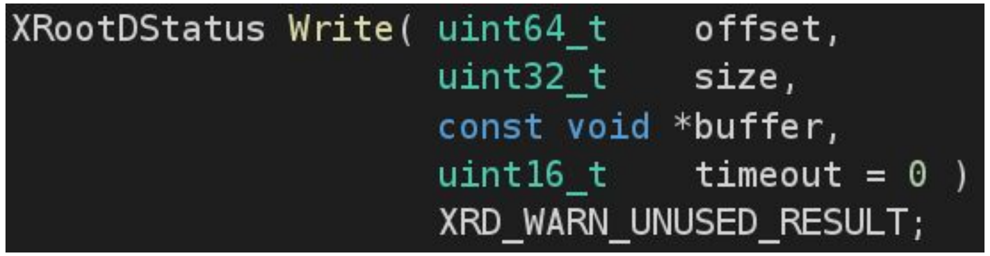
XRootD with Python bindings

New declarative file access API for XRootD client

CERN openlab

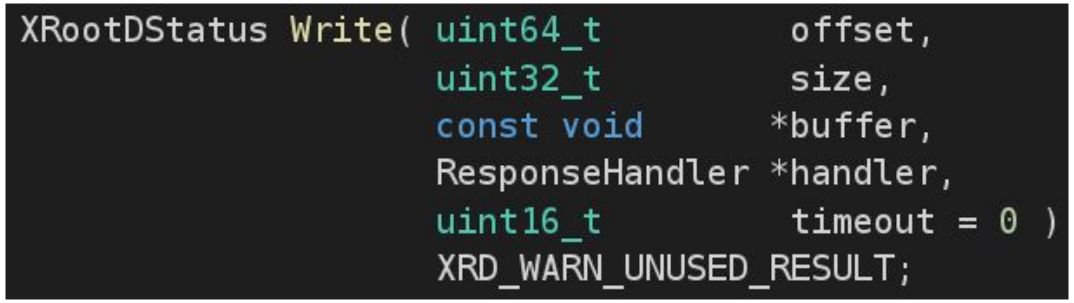
Two versions of file operations

- Synchronous



```
XRootDStatus Write( uint64_t    offset,
                    uint32_t    size,
                    const void *buffer,
                    uint16_t    timeout = 0 )
                    XRD_WARN_UNUSED_RESULT;
```

- Asynchronous



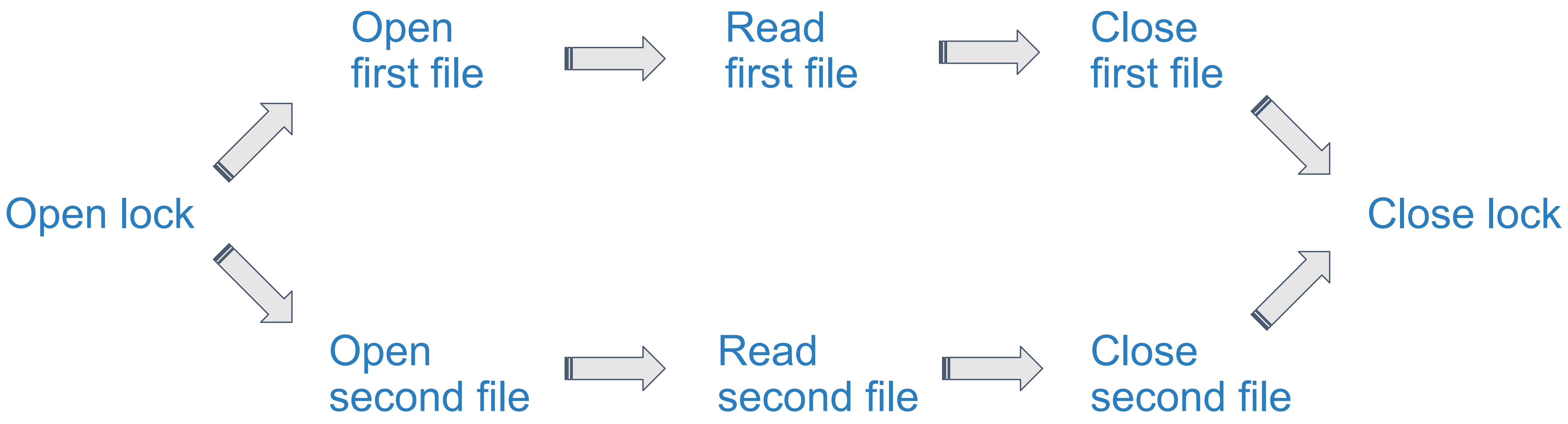
```
XRootDStatus Write( uint64_t         offset,
                    uint32_t         size,
                    const void      *buffer,
                    ResponseHandler *handler,
                    uint16_t         timeout = 0 )
                    XRD_WARN_UNUSED_RESULT;
```

CERN openlab

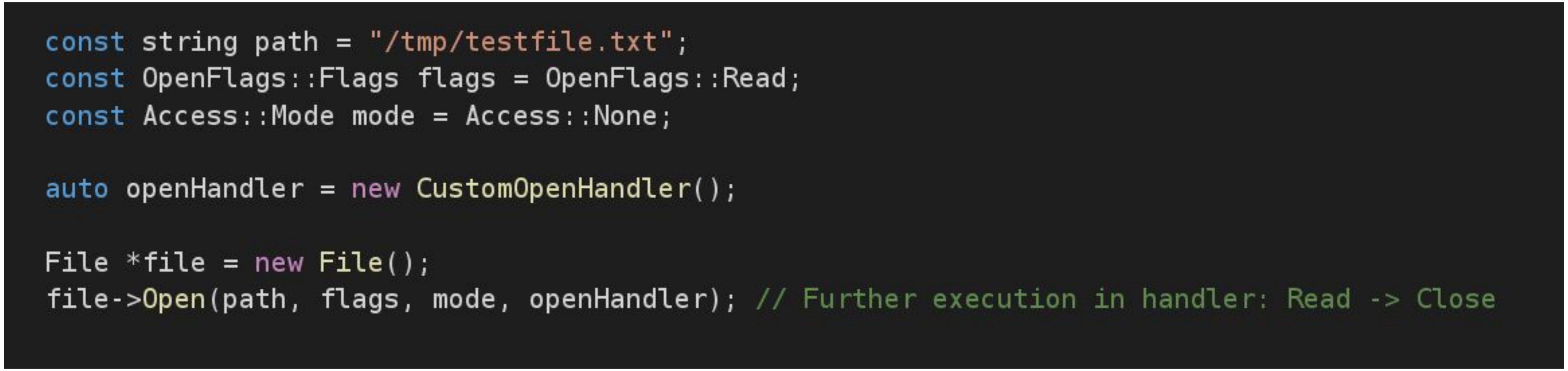
Asynchronous operations

$$\begin{tikzcd}
\text{OpenFile} \arrow[r] & \text{OpenHandler} \arrow[d] & & \\
& \text{ReadFile} \arrow[r] & \text{ReadHandler} \arrow[d] & \\
& & \text{CloseFile} \arrow[r] & \text{CloseHandler}
\end{tikzcd}$$

More complex example

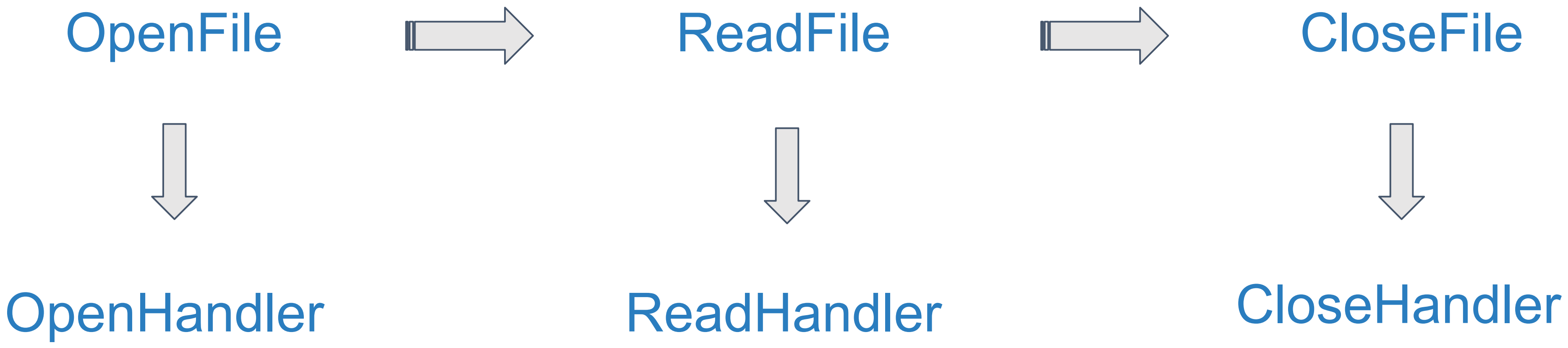


Asynchronous operations usage example

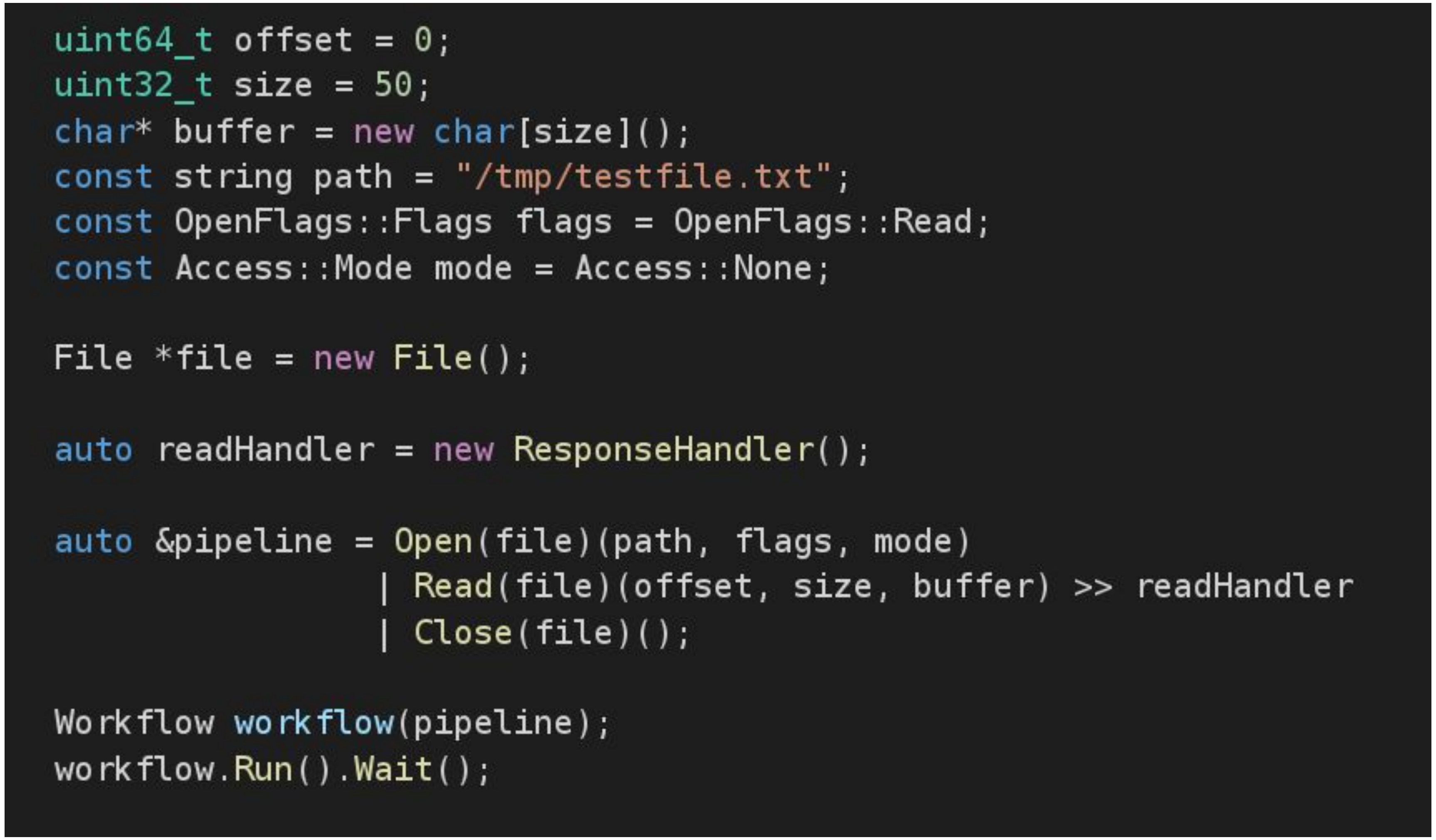


```
const string path = "/tmp/testfile.txt";
const OpenFlags::Flags flags = OpenFlags::Read;
const Access::Mode mode = Access::None;

auto openHandler = new CustomOpenHandler();

File *file = new File();
file->Open(path, flags, mode, openHandler); // Further execution in handler: Read -> Close
```

Idea: workflow mechanism



Workflow example



```cpp
uint64_t offset = 0;
uint32_t size = 50;
char* buffer = new char[size]();
const string path = "/tmp/testfile.txt";
const OpenFlags::Flags flags = OpenFlags::Read;
const Access::Mode mode = Access::None;

File *file = new File();

auto readHandler = new ResponseHandler();

auto &pipeline = Open(file)(path, flags, mode)
                | Read(file)(offset, size, buffer) >> readHandler
                | Close(file)();

Workflow workflow(pipeline);
workflow.Run().Wait();
```

Workflows - elements of implementation

- Clear semantics of composed operations
- Possibility to pass arguments between operations
- Possibility to apply it to all file operations
- Error handling
- Compile time checking of workflow declaration
- Tests

Future work

- Applying workflow mechanism to file system operations
- Adding more tests

QUESTIONS?

krzysiek.jamrog@gmail.com