Cross-platform deployment of a complex C++ computational software with GUI and Python API

Ammar Nejati, Mikhail Svechnikov, Joachim Wuttke
Jülich Centre for Neutron Science (JCNS) at Heinz Maier-Leibnitz Zentrum (MLZ), Garching, Germany

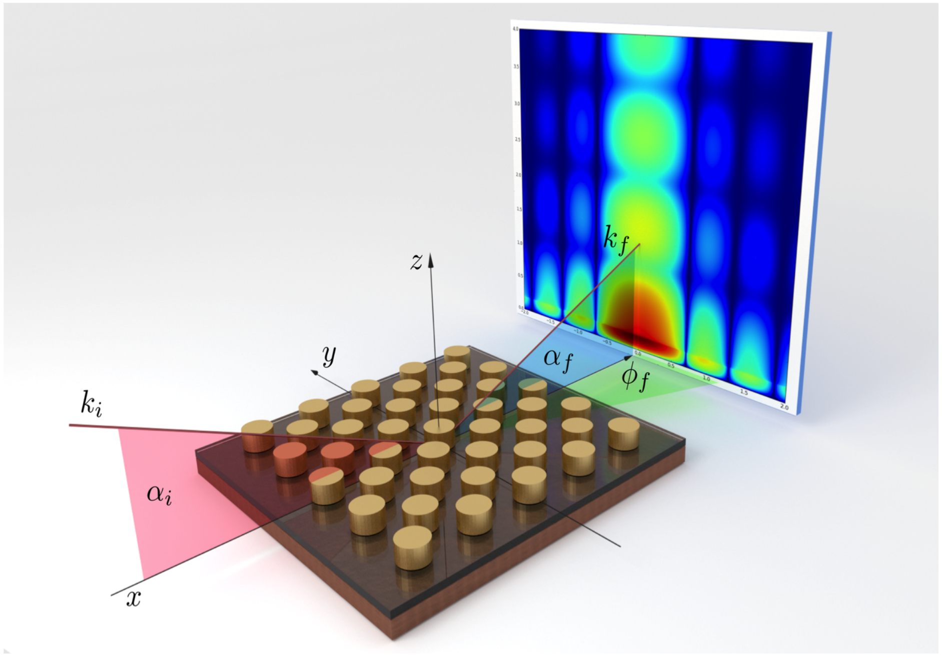
deRSE2024
March 2024, Julius-Maximilians-Universität Würzburg

BornAgain

open-source cross-platform software to simulate and fit GISAS and reflectometry

Performance:
C/C++ (low-level implementation of core)

Scripting interface:
Python (high-level interface) for
Multiple versions of Python



Graphical user interface (GUI):
Qt Framework (C++)

Multiple platforms:
Linux, MacOS, MS-Windows

Multiple packaging/installation methods:
Rootless, GUI-based, headless, self-contained

Possible to build from source

Pospelov et al, J Appl Cryst 53 (2020)
https://doi.org/10.1107/S1600576719016789

Dependencies and Linking

- **Intricate chain of dependencies**

  Computational libraries, GUI libraries and plug-ins (Qt), inter-operation with interpreters (Python)

- **Static vs. dynamic linking**

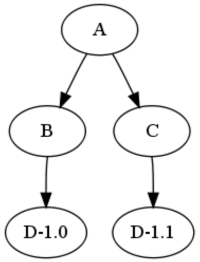
  — *static linking:*

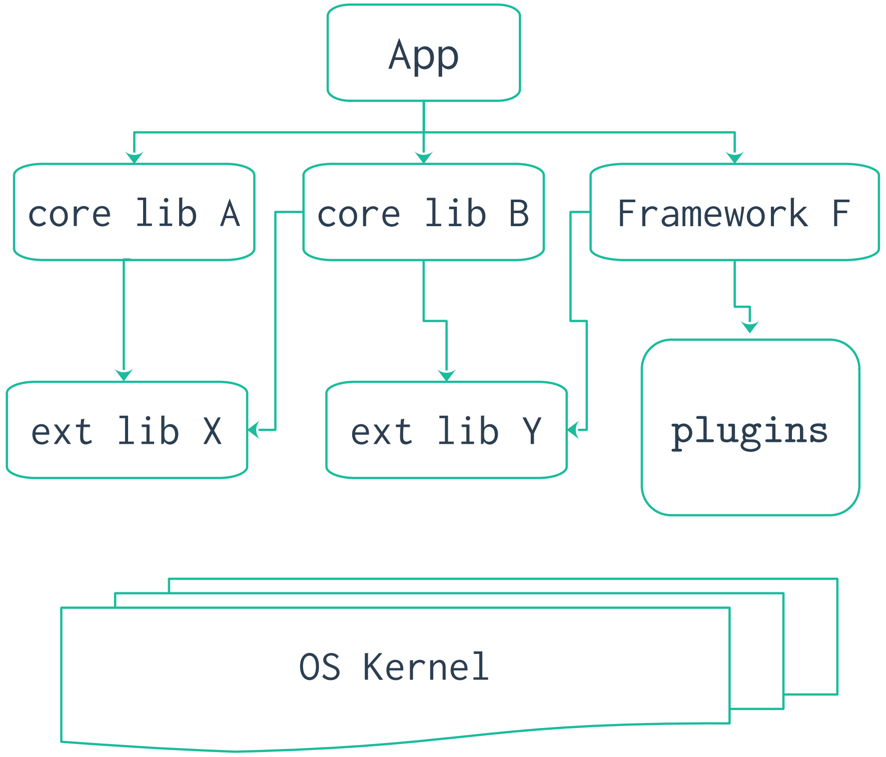
  Combining various pieces of code (object files) and data into the final executable

  — *dynamic linking:*

  Linking the final executable to "shared" libraries to be loaded into memory at runtime

- **"Infinite tree"**
- **"Dependency hell"**
- **"More is different"**





Dynamic Linking on Different Platforms

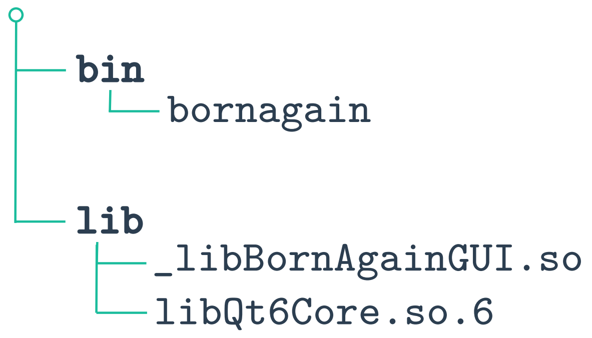
- **Linux**
  * Shared libraries (ELF format, `.so`)
  * Dynamic linker: `ld.so`
  * Versioning based on distinct file names or symbolic links; e.g. `libA.so.1.0.2`
  * Libraries located based on predefined search paths (see `man ld.so`)
    — Directories specified in `DT_RPATH` (applied to searches for all children in the dependency tree)
    — Environment variable `LD_LIBRARY_PATH`
    — Directories specified in the `DT_RUNPATH` (does not apply to the children in dependency tree)
    — Cache file `/etc/ld.so.cache` (see `man ldconfig`)
    — Default paths `/lib` and `/usr/lib`

Dynamic Linking on Different Platforms

- Linux

```
> readelf -d bornagain

Dynamic section at offset 0x16cd8 contains 34 entries:
  Tag        Type                         Name/Value
 0x0000000000000001 (NEEDED)   Shared library: [_libBornAgainGUI.so]
 0x0000000000000001 (NEEDED)   Shared library: [libboost_program_options.so.1.0]
 0x0000000000000001 (NEEDED)   Shared library: [libQt6Core.so.6]
 0x0000000000000001 (NEEDED)   Shared library: [libstdc++.so.6]
 0x0000000000000001 (NEEDED)   Shared library: [libgcc_s.so.1]
 0x0000000000000001 (NEEDED)   Shared library: [libc.so.6]
 0x000000000000000f (RPATH)    Library rpath:  [$ORIGIN:$ORIGIN/../lib]
```



Dynamic Linking on Different Platforms

- **MacOS**

  * Shared libraries (Mach-O format, .dylib or `.so`)

  * Dynamic linker: `dyld.so`

  * Versioning based on distinct file names or symbolic links and "install names"; e.g. `libA.so.1.0.2`;
  employs `install_name_tool`

  — Libraries located via their full path (not file name); e.g. `/usr/lib/libA.dylib` (see `man dyld.so`)

  — Relative paths use 3 path prefixes, e.g. `@prefix/../lib/libA.dylib`

  1. `@executable_path/`: directory of the main executable for the process
  2. `@loader_path/`: directory of the binary containing the load command
  3. `@rpath/`: substituted with each path in the runpath list until a dylib is found;
     run-paths are stored in `LC_RPATH` attributes of the dependency chain leading to the current library

  — List of paths in `DYLD_LIBRARY_PATH` and `DYLD_FRAMEWORK_PATH`

Dynamic Linking on Different Platforms

- *MacOS*

```
> otool -l MacOS/bornagain

MacOS/bornagain:
Load command 18
          cmd LC_LOAD_DYLIB
         name @rpath/libcerf.2.dylib
      current version 2.0.0
compatibility version 2.0.0
Load command 21
          cmd LC_LOAD_DYLIB
         name @rpath/_libBornAgainSample.so
Load command 45
          cmd LC_RPATH
         path @loader_path/../Library
Load command 46
          cmd LC_RPATH
         path @loader_path/../Frameworks/
```

```
App Contents
 ├── MacOS
 │    └── bornagain
 ├── Library
 │    ├── libcerf.2.dylib
 │    └── libgsl.25.dylib
 ├── Frameworks
 │    └── Qt
 │         └── QtGui.framework
 │              └── QtGui
 └── PlugIns
```

Dynamic Linking on Different Platforms

- **Windows**

  * Shared libraries (`.dll` extension) and import libraries (`.lib` file) with PE format

  * DLL Loader (`LoadLibrary` in the Windows API)

  * Versioning system based on a manifest file (XML) embedded within the library DLL

  * Libraries are located based on predefined search paths

  (see `https://learn.microsoft.com/en-us/windows/win32/dlls/dynamic-link-library-search-order`)

— Folder from which the application loaded

— System folders, `%SystemRoot%\system32\System32` or `%SystemRoot%\SysWow64`

— Current folder

— Directories listed in the `PATH` environment variable

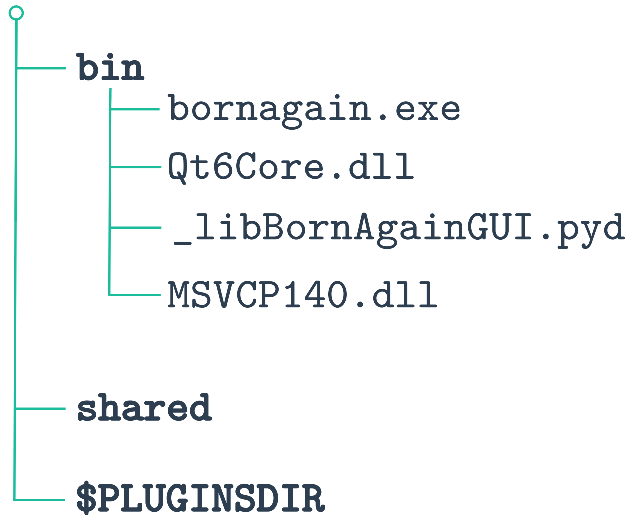
Dynamic Linking on Different Platforms

- **Windows**

```
> DUMPBIN /DEPENDENTS bornagain.exe

Dump of file bornagain.exe

File Type: EXECUTABLE IMAGE
  Image has the following dependencies:
    _libBornAgainBase.pyd
    boost_program_options-vc142-mt-x64-1_77.dll
    Qt6Core.dll
    MSVCP140.dll
    VCRUNTIME140.dll
    VCRUNTIME140_1.dll
    api-ms-win-crt-runtime-l1-1-0.dll
    api-ms-win-crt-math-l1-1-0.dll
    api-ms-win-crt-utility-l1-1-0.dll
    KERNEL32.dll
```



Generating a High-Level Python Interface

- **SWIG (Simplified Wrapper and Interface Generator)**

  Automatically generates Python wrapper code for C/C++ libraries, based on parsing C/C++ header files

  The generated code, along with the original C/C++ code, is compiled to create shared library which can be imported in Python

- **pybind11** or **nanobind**

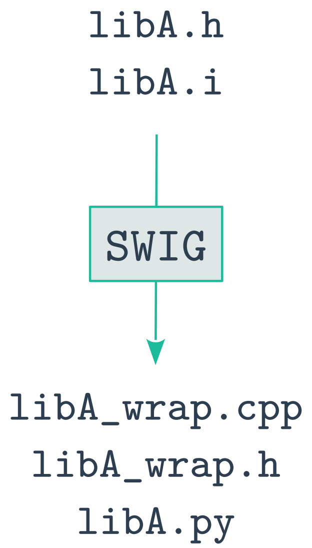
- **Cython + Python setuptools**

  Superset of Python that allows Python-like code with C-like performance

  The compiled Cython code generates CPython extension modules

- **F2PY**

  NumPy tool that automatically generates Python interfaces for Fortran 77 or 95 code



Wheel: Python Binary Package

- **Wheel (PEP 427)**

  * A ZIP-archive with a specific file name:

  ```
  {distribution}-{version}(-{build tag})?-{python tag}-{abi tag}-{platform tag}.whl
  ```

  * Can be installed via standard package installers (like `pip`) or simply unpacking into `site-packages` with via ‘unzip’ tool

  * Created via respective `pip` command

  * *Platform Wheel*: depends on the Python Standard Library and additional platform-specific dependencies

```
├── setup.py
├── setup.cfg
├── pyproject.toml
└── src
    └── bornagain
        ├── __init__.py
        └── lib
            ├── __init__.py
            ├── libA.py
            └── _libA.so
```

Support for Multiple Versions of Python

- **PyEnv** `<https://github.com/pyenv/pyenv>`

  Python version management tool that enables users to manage multiple Python versions and environments on the same system

- **Conda**

  * Anaconda `<https://www.anaconda.com>`

  Comprehensive Python distribution with multiple versions of Python plus a collection of pre-installed packages

  Provides the `conda` package manager for managing environments and dependencies

  * Miniconda `<https://docs.anaconda.com/free/miniconda>`

  Lightweight version of Anaconda with fewer pre-installed packages

Different Installers for Each Platform

- **Linux**

  Standard packages (`.deb`, `.rpm` extension)

  Self-extracting installer (`.sh` extension)

- **MacOS**

  Disk images (`.dmg` extension)

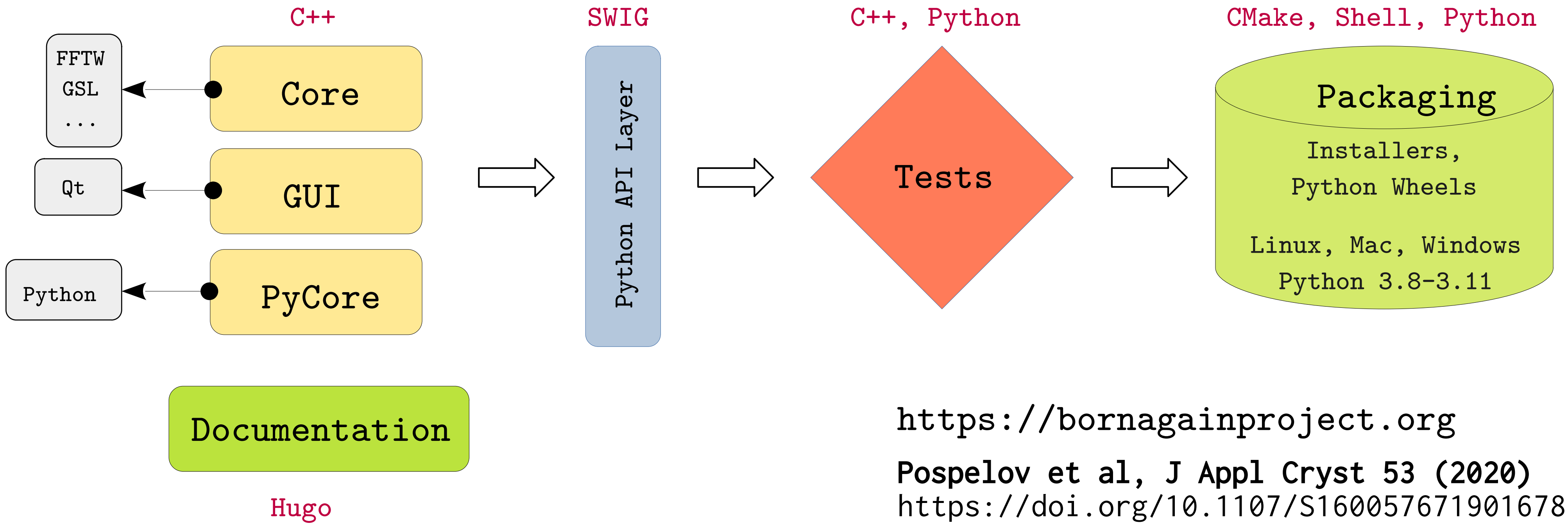
- **Windows**

  Windows installer produced by NSIS or Qt Installer Framework (`.exe` extension)

∴ *Rootless*, *GUI-based*, *headless* and *self-contained* installers for *all* platforms

A View of BornAgain Build System

GitLab + CMake



https://bornagainproject.org

Pospelov et al, J Appl Cryst 53 (2020)
https://doi.org/10.1107/S1600576719016789

Thanks for your attention

Q U E
S T I
O N S
?