A CMake toolkit for migrating to clang's module system

Raphael Isemann

ROOT
Data Analysis Framework
https://root.cern

▶ Replacing the old textual `#include` model for handling headers.

▶ Many advantages, most notably **faster compilation** times due to precompiled headers.

▶ Works with current C/C++ code!

▶ Requires some extra configuration code.
- Configuration happens via modulemap files.

▶ Still some bugs in the clang implementation.

▶ **System libraries are not modularized.**

- ▶ System libraries...
  - ● are in read-only locations.
  - ● come in different versions.
  - ● are sometimes patched by the OS distribution, administrator or user.
  - ● and also need module configurations!
- ▶ Which means you have to modularize this in your build script for all versions.

- ▶ A Python project for automatic module setup!
- ▶ Comes with modulemaps for many different system libraries.
- ▶ Checks on the fly which modulemap works for your current system setup.
- ▶ Places the right modulemaps in these paths with clang's virtual file system.

Using ClangAutoModules in CMake

- ▶ Standalone CMake wrapper containing all files.
- ▶ To add it to your CMake project:

```
file(DOWNLOAD https://github.com/Teemperor/ClangAutoModules/releases/[...]/ClangModules.cmake
            ${CMAKE_BINARY_DIR}/ClangModules.cmake ...)
include(${CMAKE_BINARY_DIR}/ClangModules.cmake)
```

- ▶ Queries script with current CMake settings.
- ▶ Configures current CMake project with the results from the script.

▶ Wrapper script around clang (like scan-build)

▶ `$ CC=clang_modules CXX=clang_modules++ cmake ...`

▶ Automatically configures system modules and invokes clang with the needed flags.

▶ Still requires that the code is inside a CMake project at the moment.

▶ Optional fallback to non-module compilation.

github.com/Teemperor/ClangAutoModules

(goo.gl/M923kR)

▶ For more features, examples and API documentation.
▶ Create issues for other libraries we should support!

And thanks for your attention!