A CMake-based build and configuration framework

M. Clemencic, P. Mato

CERN - LHCb

CHEP 2012 - New York

Outline

Outline

Building LHCb Software

- Software projects need build tools
  - make, autotools, ant, jam, ...
- HEP software has special requirements
  - reproducibility and control
- CMT: a Configuration Management Tool
  - manages concurrent versions of projects and packages
  - dynamic runtime environment

Building LHCb Software

- Software projects need build tools
  - make, autotools, ant, jam, ...
- HEP software has special requirements
  - reproducibility and control
- CMT: a Configuration Management Tool
  - manages concurrent versions of projects and packages
  - dynamic runtime environment

Building LHCb Software

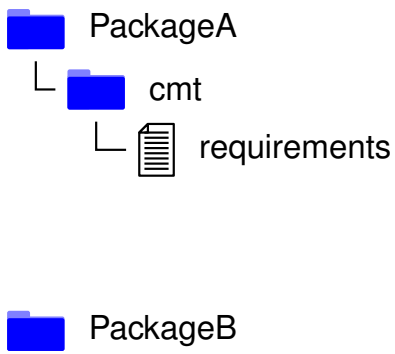
- Software projects need build tools
  - make, autotools, ant, jam, ...
- HEP software has special requirements
  - reproducibility and control
- CMT: a Configuration Management Tool
  - manages concurrent versions of projects and packages
  - dynamic runtime environment

CMT Projects Layout

- Code organized in packages
  - configuration file
    - *use* other packages
    - declare products
    - declare environment
  - sources
  - data files
- Packages grouped in projects
  - configuration file
  - *container* package
  - *policy* package
- Coexisting versions

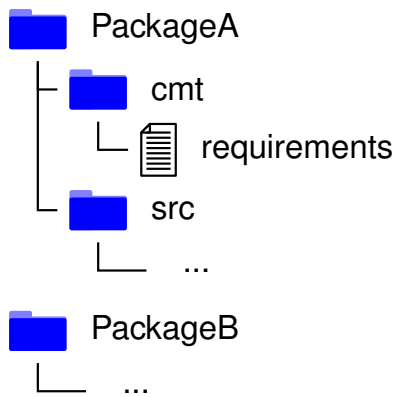
CMT Projects Layout

- Code organized in packages
  - configuration file
    - *use* other packages
    - declare products
    - declare environment
  - sources
  - data files
- Packages grouped in projects
  - configuration file
  - *container* package
  - *policy* package
- Coexisting versions



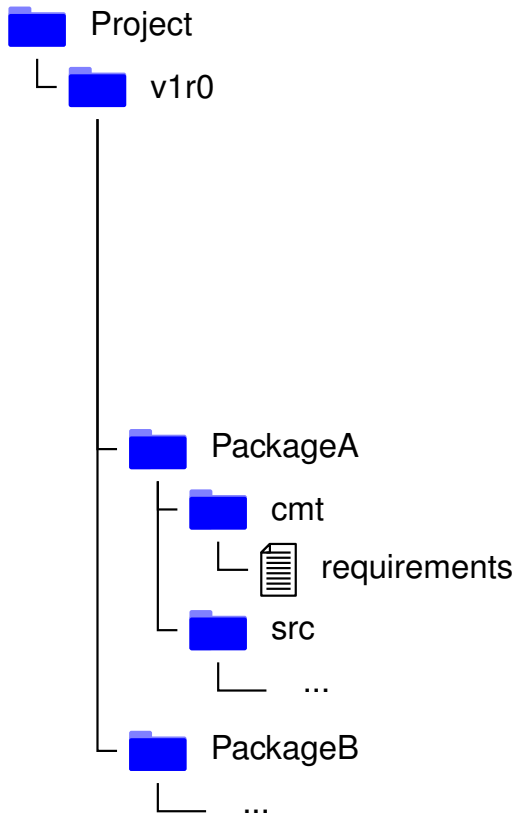
CMT Projects Layout

- Code organized in packages
  - configuration file
    - *use* other packages
    - declare products
    - declare environment
  - sources
  - data files
- Packages grouped in projects
  - configuration file
  - *container* package
  - *policy* package
- Coexisting versions



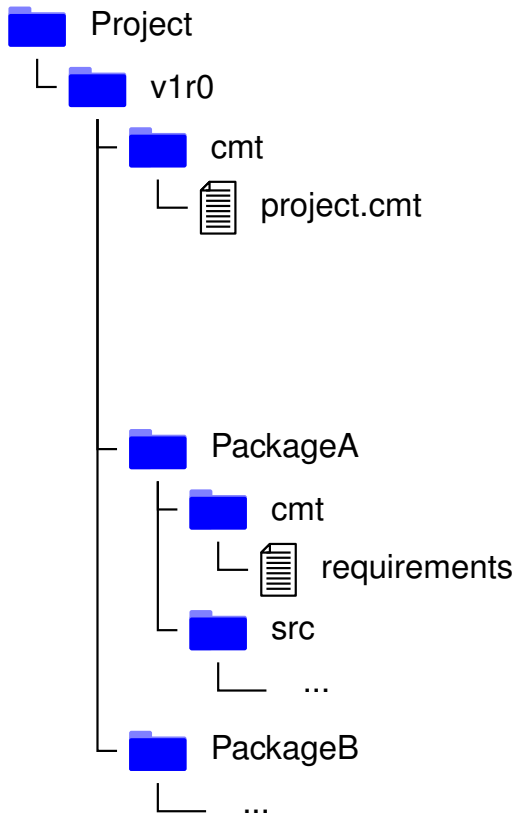
CMT Projects Layout

- Code organized in packages
  - configuration file
    - *use* other packages
    - declare products
    - declare environment
  - sources
  - data files
- Packages grouped in projects
  - configuration file
  - *container* package
  - *policy* package
- Coexisting versions



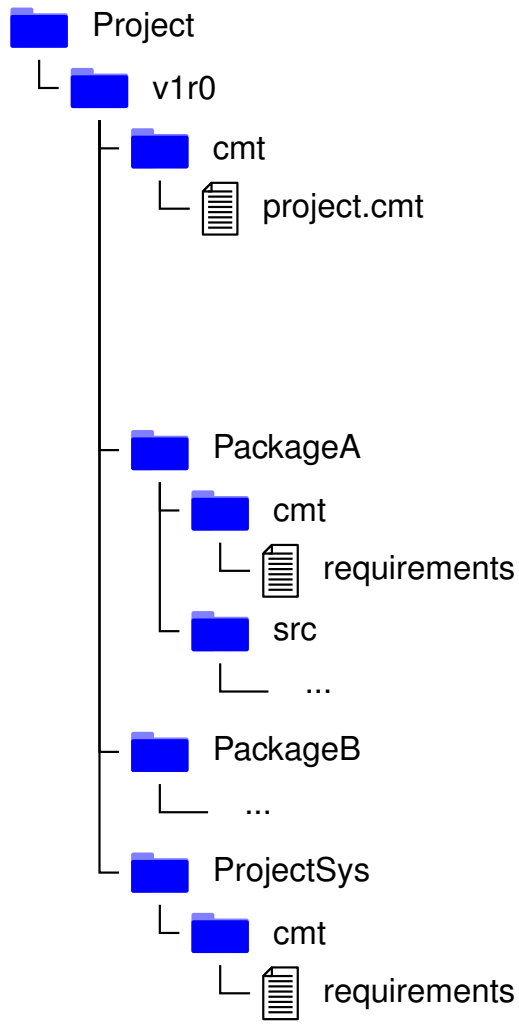
CMT Projects Layout

- Code organized in packages
  - configuration file
    - *use* other packages
    - declare products
    - declare environment
  - sources
  - data files
- Packages grouped in projects
  - configuration file
  - *container* package
  - *policy* package
- Coexisting versions



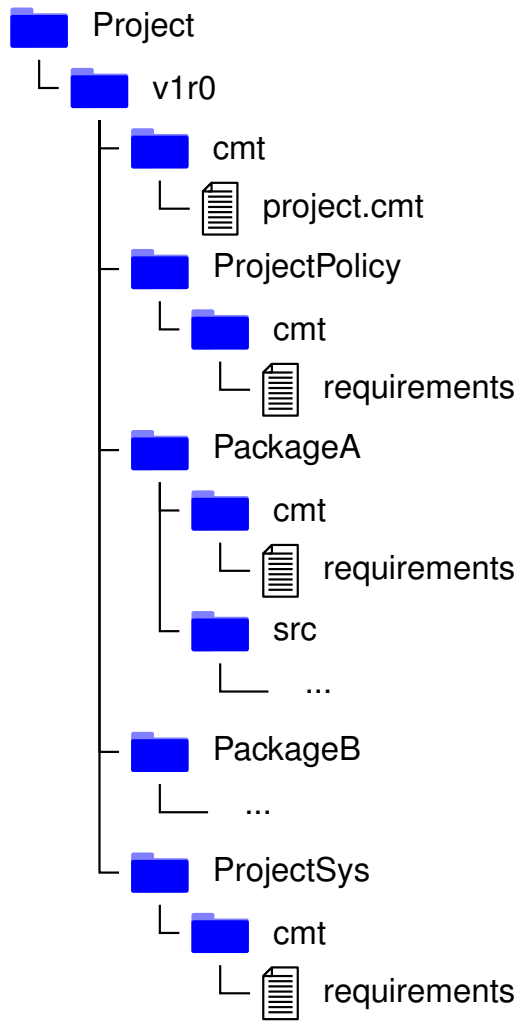
CMT Projects Layout

- Code organized in packages
  - configuration file
    - *use* other packages
    - declare products
    - declare environment
  - sources
  - data files
- Packages grouped in projects
  - configuration file
  - *container* package
  - *policy* package
- Coexisting versions



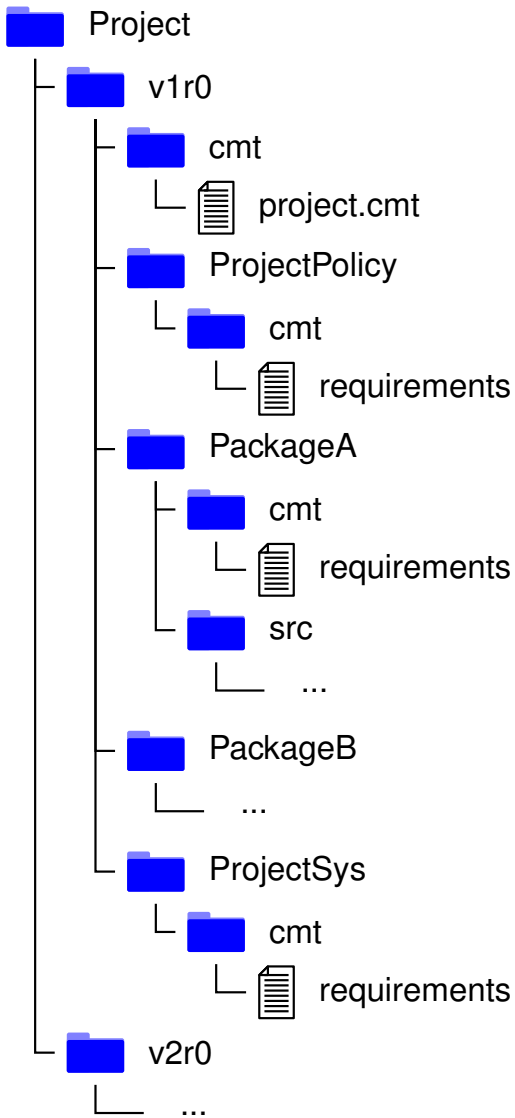
CMT Projects Layout

- Code organized in packages
  - configuration file
    - *use* other packages
    - declare products
    - declare environment
  - sources
  - data files
- Packages grouped in projects
  - configuration file
  - *container* package
  - *policy* package
- Coexisting versions



CMT Projects Layout

- Code organized in packages
  - configuration file
    - *use* other packages
    - declare products
    - declare environment
  - sources
  - data files
- Packages grouped in projects
  - configuration file
  - *container* package
  - *policy* package
- Coexisting versions



LHCb Software & CMT

- Projects Layout
  - enforced by CMT
- Customizations
  - patterns (functions)
  - makefile fragments
- Optimizations
  - improved dependency computation
  - wrapper for parallel build
- Extensions
  - special command to prepare the environment
- Many functionalities not used

LHCb Software & CMT

- Projects Layout
  - enforced by CMT
- Customizations
  - patterns (functions)
  - makefile fragments
- Optimizations
  - improved dependency computation
  - wrapper for parallel build
- Extensions
  - special command to prepare the environment
- Many functionalities not used

LHCb Software & CMT

- Projects Layout
  - enforced by CMT
- Customizations
  - patterns (functions)
  - makefile fragments
- Optimizations
  - improved dependency computation
  - wrapper for parallel build
- Extensions
  - special command to prepare the environment
- Many functionalities not used

LHCb Software & CMT

- Projects Layout
  - enforced by CMT
- Customizations
  - patterns (functions)
  - makefile fragments
- Optimizations
  - improved dependency computation
  - wrapper for parallel build
- Extensions
  - special command to prepare the environment
- Many functionalities not used

LHCb Software & CMT

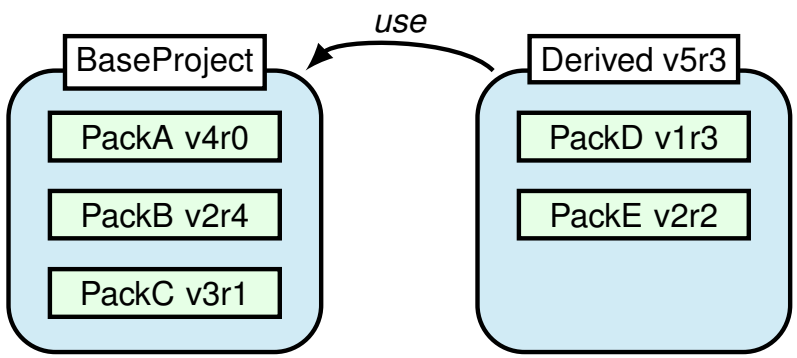
- Projects Layout
  - enforced by CMT
- Customizations
  - patterns (functions)
  - makefile fragments
- Optimizations
  - improved dependency computation
  - wrapper for parallel build
- Extensions
  - special command to prepare the environment
- Many functionalities not used

Overriding Packages

- Special feature of CMT
  - a project can override packages from projects it uses
- Extremely used in LHCb
  - Pick up bugfixes before releases
  - Lightweight development environment

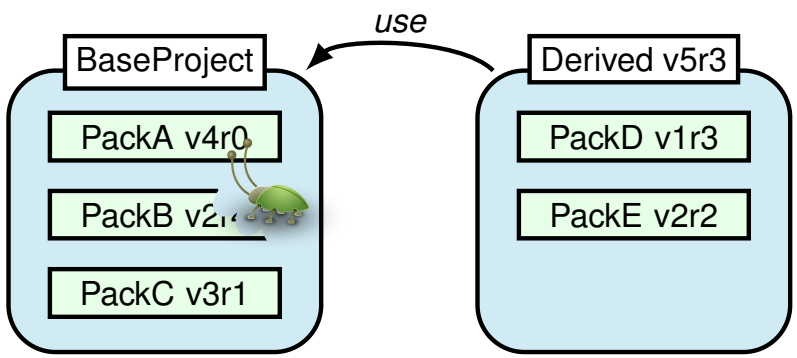
Overriding Packages

- Special feature of CMT
  - a project can override packages from projects it uses
- Extremely used in LHCb
  - Pick up bugfixes before releases
  - Lightweight development environment



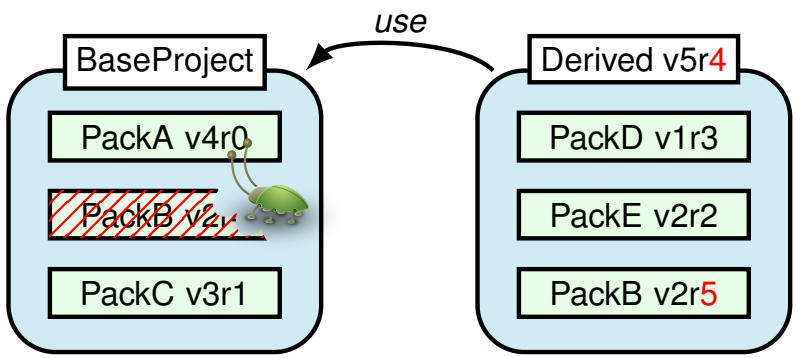
Overriding Packages

- Special feature of CMT
  - a project can override packages from projects it uses
- Extremely used in LHCb
  - Pick up bugfixes before releases
  - Lightweight development environment



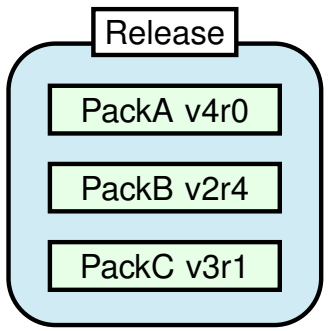
Overriding Packages

- Special feature of CMT
  - a project can override packages from projects it uses
- Extremely used in LHCb
  - Pick up bugfixes before releases
  - Lightweight development environment



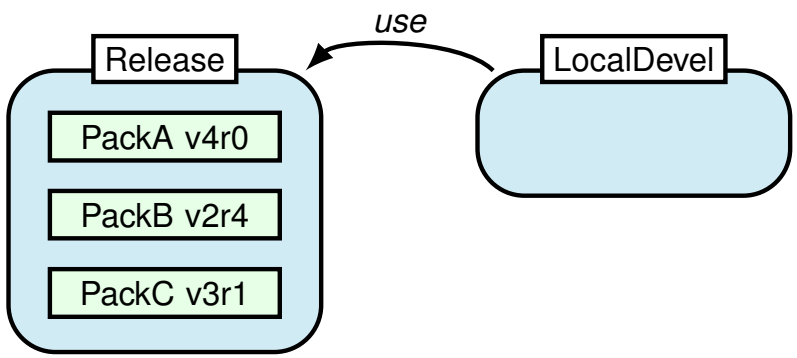
Overriding Packages

- Special feature of CMT
  - a project can override packages from projects it uses
- Extremely used in LHCb
  - Pick up bugfixes before releases
  - Lightweight development environment



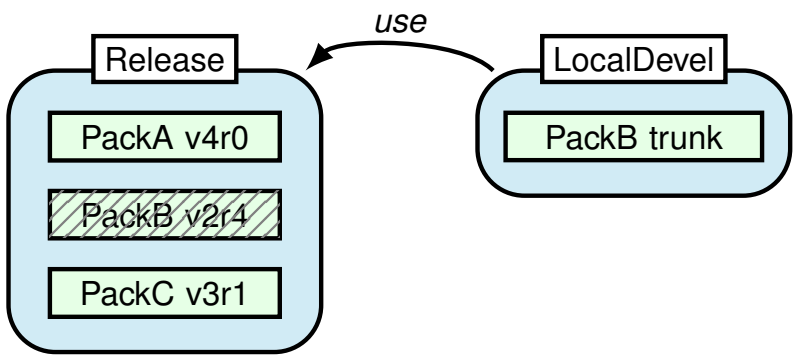
Overriding Packages

- Special feature of CMT
  - a project can override packages from projects it uses
- Extremely used in LHCb
  - Pick up bugfixes before releases
  - Lightweight development environment



Overriding Packages

- Special feature of CMT
  - a project can override packages from projects it uses
- Extremely used in LHCb
  - Pick up bugfixes before releases
  - Lightweight development environment



Why to change?

- CMT has got limitations
  - OK on small projects, but very slow on big projects
  - limited logic of configuration language
- We know what we need
  - we do not use all the features in CMT
  - we extended and customized it to fit better our needs
- New products on the market
- Good time to investigate something new

Why to change?

- CMT has got limitations
  - OK on small projects, but very slow on big projects
  - limited logic of configuration language
- We know what we need
  - we do not use all the features in CMT
  - we extended and customized it to fit better our needs
- New products on the market
- Good time to investigate something new

Why to change?

- CMT has got limitations
  - OK on small projects, but very slow on big projects
  - limited logic of configuration language
- We know what we need
  - we do not use all the features in CMT
  - we extended and customized it to fit better our needs
- New products on the market
- Good time to investigate something new

Why to change?

- CMT has got limitations
  - OK on small projects, but very slow on big projects
  - limited logic of configuration language
- We know what we need
  - we do not use all the features in CMT
  - we extended and customized it to fit better our needs
- New products on the market
- Good time to investigate something new

Outline

Requirements

- Flexibility
  - organize code in projects and packages
  - easily add/remove/move packages
- Override packages
  - derived projects can override packages
- Simplicity
  - minimalistic language (no details)
- Runtime environment
  - easy set-up for any version of any project
- Smooth migration
  - allow for adiabatic adoption of new framework

Requirements

- Flexibility
  - organize code in projects and packages
  - easily add/remove/move packages
- Override packages
  - derived projects can override packages
- Simplicity
  - minimalistic language (no details)
- Runtime environment
  - easy set-up for any version of any project
- Smooth migration
  - allow for adiabatic adoption of new framework

Requirements

- Flexibility
  - organize code in projects and packages
  - easily add/remove/move packages
- Override packages
  - derived projects can override packages
- Simplicity
  - minimalistic language (no details)
- Runtime environment
  - easy set-up for any version of any project
- Smooth migration
  - allow for adiabatic adoption of new framework

Requirements

- Flexibility
  - organize code in projects and packages
  - easily add/remove/move packages
- Override packages
  - derived projects can override packages
- Simplicity
  - minimalistic language (no details)
- Runtime environment
  - easy set-up for any version of any project
- Smooth migration
  - allow for adiabatic adoption of new framework

Requirements

- Flexibility
  - organize code in projects and packages
  - easily add/remove/move packages
- Override packages
  - derived projects can override packages
- Simplicity
  - minimalistic language (no details)
- Runtime environment
  - easy set-up for any version of any project
- Smooth migration
  - allow for adiabatic adoption of new framework

What to use?

- New products with respect to 10 years ago
- Many tools are too specific
  - wrong language (we need C++ and Python)
  - non portable (Unix only)
  - only specific type of projects
- Few are generic and flexible
  - CMake, SCons, . . .
- CMake is powerful and widely used (e.g. KDE)

What to use?

- New products with respect to 10 years ago
- Many tools are too specific
  - wrong language (we need C++ and Python)
  - non portable (Unix only)
  - only specific type of projects
- Few are generic and flexible
  - CMake, SCons, ...
- CMake is powerful and widely used (e.g. KDE)

What to use?

- New products with respect to 10 years ago
- Many tools are too specific
  - wrong language (we need C++ and Python)
  - non portable (Unix only)
  - only specific type of projects
- Few are generic and flexible
  - CMake, SCons, ...
- CMake is powerful and widely used (e.g. KDE)

What to use?

- New products with respect to 10 years ago
- Many tools are too specific
  - wrong language (we need C++ and Python)
  - non portable (Unix only)
  - only specific type of projects
- Few are generic and flexible
  - CMake, SCons, ...
- CMake is powerful and widely used (e.g. KDE)

Does CMake fit?

- Pros
  - projects and subdirectories
  - very powerful (complete) language
  - library of modules for configuration
  - extensible with functions and macros
  - *properties*
- Cons
  - no support for runtime environment
  - cannot override targets
  - transitivity of libraries, but not of includes

Something just fit, something not, but the language and the features are powerful enough to outweigh the limitations.

Does CMake fit?

- Pros
  - projects and subdirectories
  - very powerful (complete) language
  - library of modules for configuration
  - extensible with functions and macros
  - *properties*
- Cons
  - no support for runtime environment
  - cannot override targets
  - transitivity of libraries, but not of includes

Something just fit, something not, but the language and the features are powerful enough to outweigh the limitations.

Does CMake fit?

- Pros
  - projects and subdirectories
  - very powerful (complete) language
  - library of modules for configuration
  - extensible with functions and macros
  - *properties*
- Cons
  - no support for runtime environment
  - cannot override targets
  - transitivity of libraries, but not of includes

Something just fit, something not, but the language and the features are powerful enough to outweigh the limitations.

Outline

Main Elements

- Projects
  - entry point to the build configuration
  - coordinate the hosted subdirectories
- Subdirectories
  - equivalent to packages in CMT
  - describe the components to build/install
- Toolchains
  - replace the fixed set of external libraries
  - allow special settings (e.g. compiler)
- Properties
  - used to communicate between components
- Exports
  - communicate between projects

Main Elements

- Projects
  - entry point to the build configuration
  - coordinate the hosted subdirectories
- Subdirectories
  - equivalent to packages in CMT
  - describe the components to build/install
- Toolchains
  - replace the fixed set of external libraries
  - allow special settings (e.g. compiler)
- Properties
  - used to communicate between components
- Exports
  - communicate between projects

Main Elements

- Projects
  - entry point to the build configuration
  - coordinate the hosted subdirectories
- Subdirectories
  - equivalent to packages in CMT
  - describe the components to build/install
- Toolchains
  - replace the fixed set of external libraries
  - allow special settings (e.g. compiler)
- Properties
  - used to communicate between components
- Exports
  - communicate between projects

Main Elements

- Projects
  - entry point to the build configuration
  - coordinate the hosted subdirectories
- Subdirectories
  - equivalent to packages in CMT
  - describe the components to build/install
- Toolchains
  - replace the fixed set of external libraries
  - allow special settings (e.g. compiler)
- Properties
  - used to communicate between components
- Exports
  - communicate between projects

Main Elements

- Projects
  - entry point to the build configuration
  - coordinate the hosted subdirectories
- Subdirectories
  - equivalent to packages in CMT
  - describe the components to build/install
- Toolchains
  - replace the fixed set of external libraries
  - allow special settings (e.g. compiler)
- Properties
  - used to communicate between components
- Exports
  - communicate between projects

Outline

Layout

- Main CMake module
  - core of the configuration framework
  - contains all the functions and extensions
- Compile flags module
  - module for compile/link flags and settings
- Toolchains modules
  - define search paths for custom build of external libraries
- Contributed *find* modules
  - `FindX.cmake` modules not provided by standard CMake
- Custom tool for environment manipulation
  - Python script to prepare the environment from simple configuration

Layout

- Main CMake module
  - core of the configuration framework
  - contains all the functions and extensions
- Compile flags module
  - module for compile/link flags and settings
- Toolchains modules
  - define search paths for custom build of external libraries
- Contributed *find* modules
  - `FindX.cmake` modules not provided by standard CMake
- Custom tool for environment manipulation
  - Python script to prepare the environment from simple configuration

Layout

- Main CMake module
  - core of the configuration framework
  - contains all the functions and extensions
- Compile flags module
  - module for compile/link flags and settings
- Toolchains modules
  - define search paths for custom build of external libraries
- Contributed *find* modules
  - `FindX.cmake` modules not provided by standard CMake
- Custom tool for environment manipulation
  - Python script to prepare the environment from simple configuration

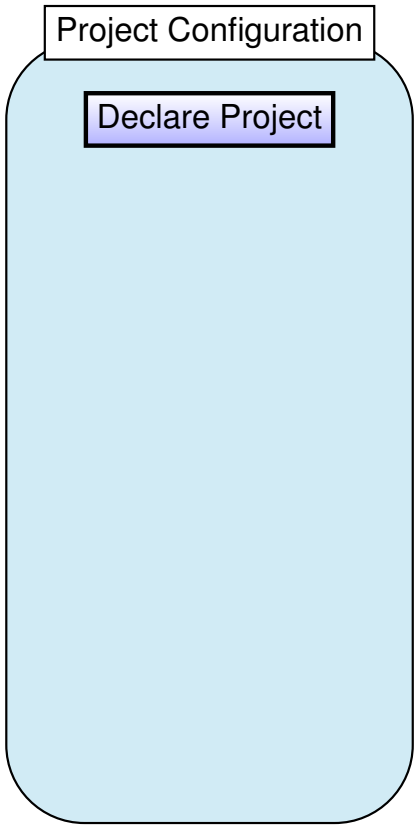
Layout

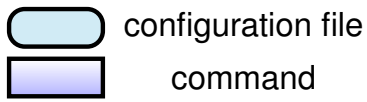
- Main CMake module
  - core of the configuration framework
  - contains all the functions and extensions
- Compile flags module
  - module for compile/link flags and settings
- Toolchains modules
  - define search paths for custom build of external libraries
- Contributed *find* modules
  - `FindX.cmake` modules not provided by standard CMake
- Custom tool for environment manipulation
  - Python script to prepare the environment from simple configuration

Layout

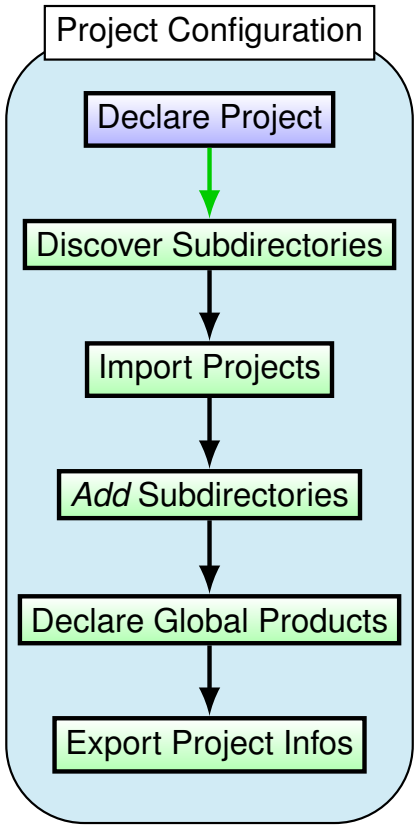
- Main CMake module
  - core of the configuration framework
  - contains all the functions and extensions
- Compile flags module
  - module for compile/link flags and settings
- Toolchains modules
  - define search paths for custom build of external libraries
- Contributed *find* modules
  - `FindX.cmake` modules not provided by standard CMake
- Custom tool for environment manipulation
  - Python script to prepare the environment from simple configuration

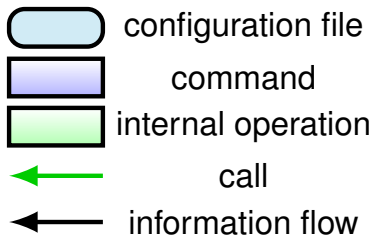
Configuration Files



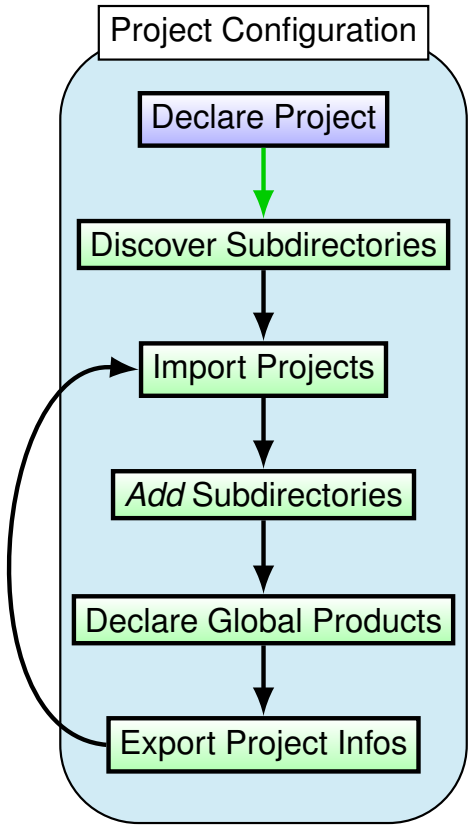


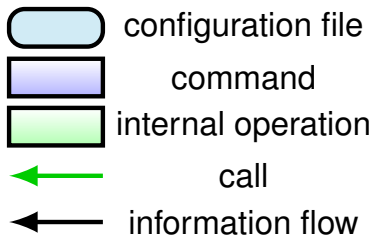
Configuration Files



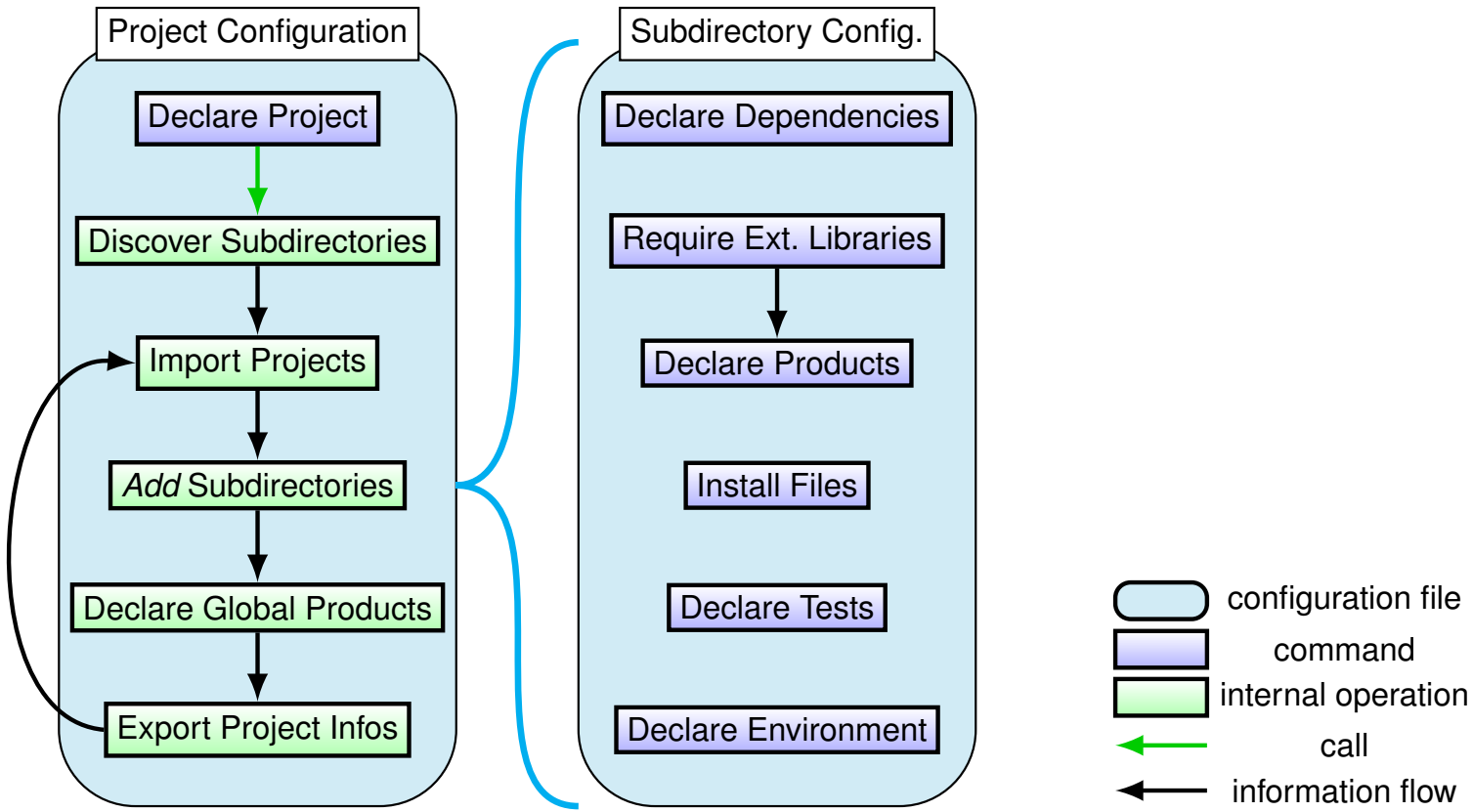


Configuration Files

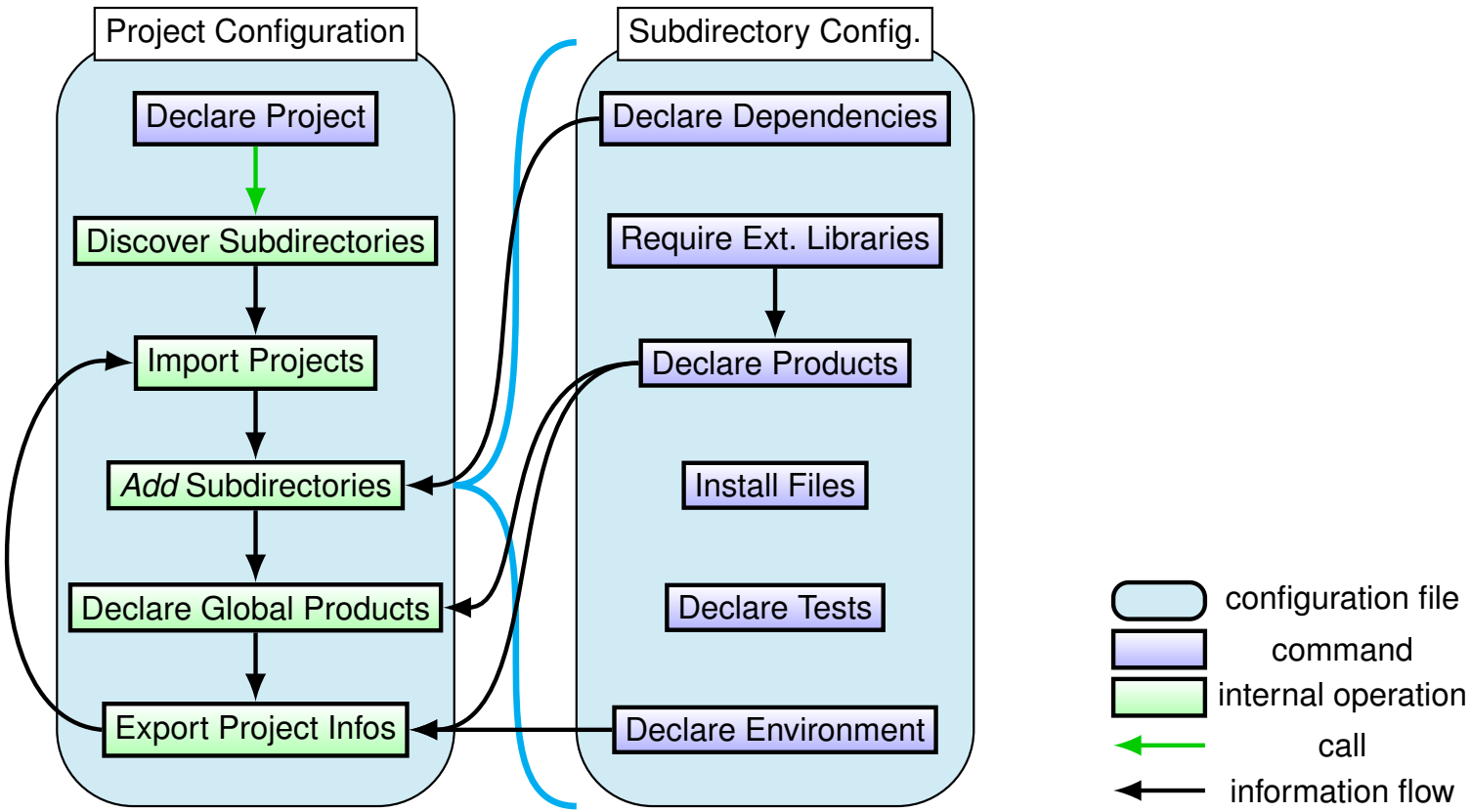




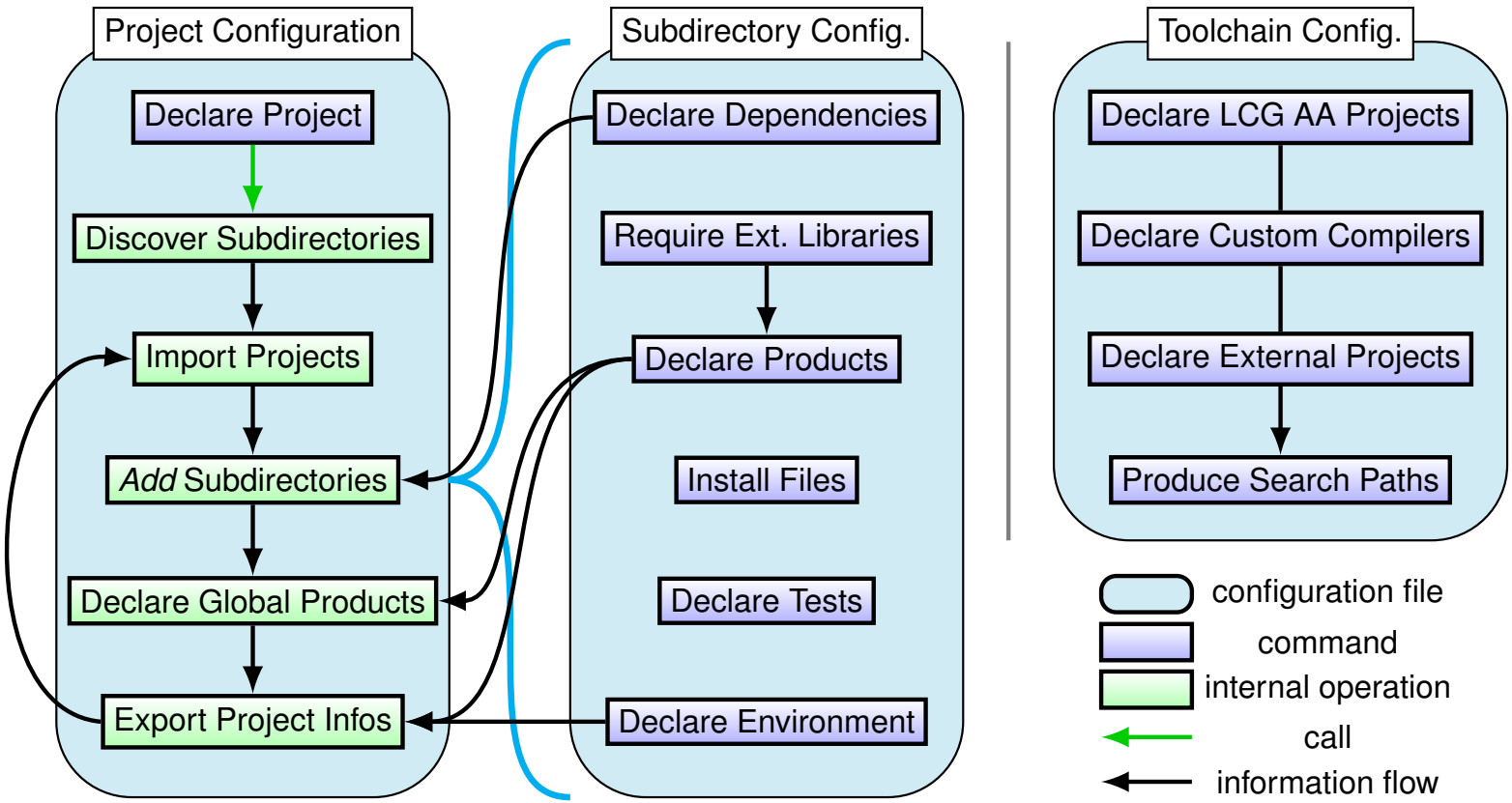
Configuration Files



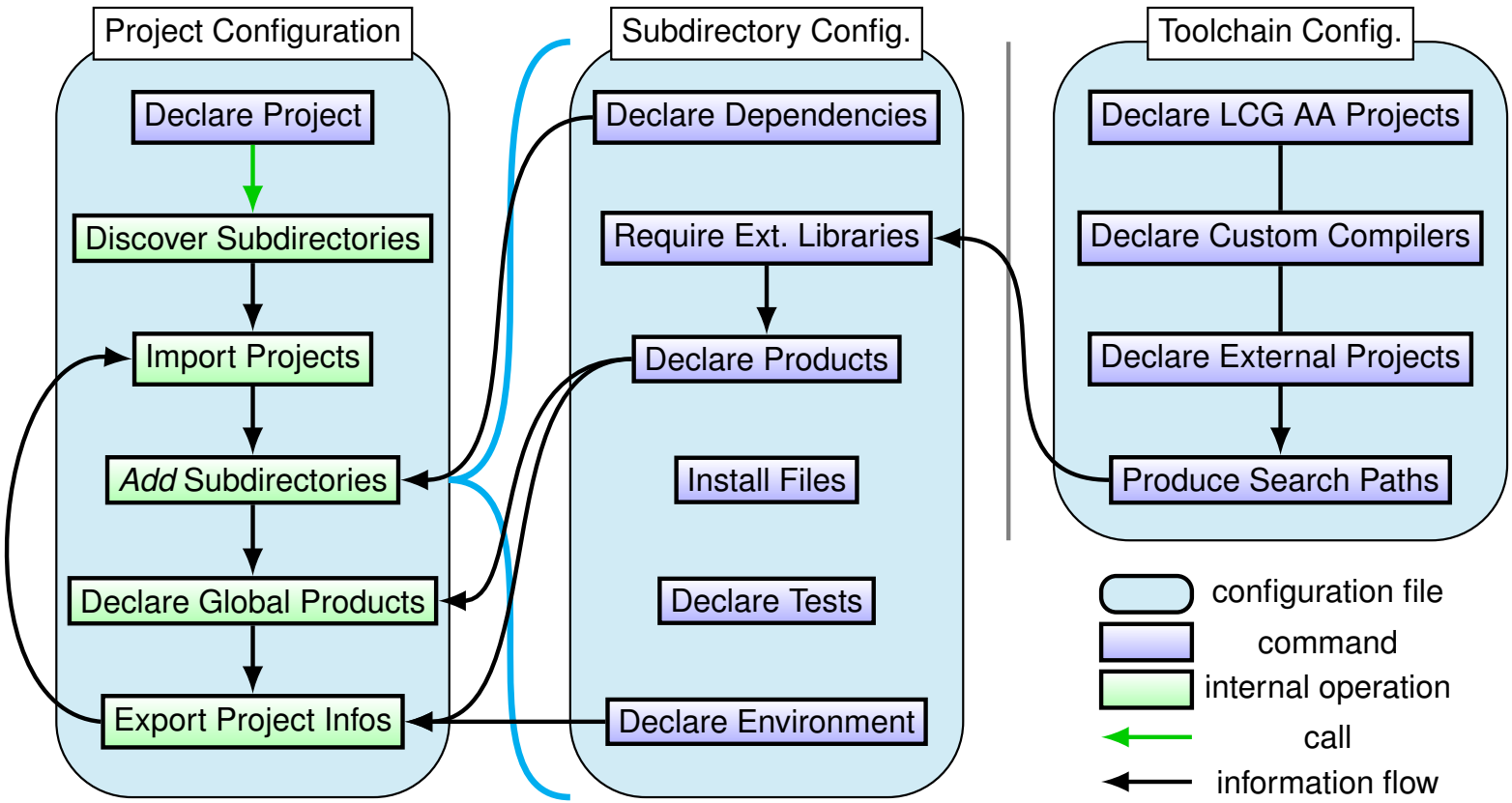
Configuration Files



Configuration Files



Configuration Files



From CMT to CMake

- Compatibility
  - produce same filesystem hierarchy as CMT
  - preserve CMT configuration files
- Translation tool
  - Python script to analyze and translate configurations
  - used only for the first translation
- Migration plan
  - Validate framework (Summer)
  - Migrate project by project (LHC shutdown)
  - Phase out CMT (still used in old versions)

From CMT to CMake

- Compatibility
  - produce same filesystem hierarchy as CMT
  - preserve CMT configuration files
- Translation tool
  - Python script to analyze and translate configurations
  - used only for the first translation
- Migration plan
  - Validate framework (Summer)
  - Migrate project by project (LHC shutdown)
  - Phase out CMT (still used in old versions)

From CMT to CMake

- Compatibility
  - produce same filesystem hierarchy as CMT
  - preserve CMT configuration files
- Translation tool
  - Python script to analyze and translate configurations
  - used only for the first translation
- Migration plan
  - Validate framework (Summer)
  - Migrate project by project (LHC shutdown)
  - Phase out CMT (still used in old versions)

From CMT to CMake (2)

```
# ======================================
package GaudiUtils
version v4r0
# ======= structure ====================
branches GaudiUtils src cmt doc
# ======= dependencies =================
use GaudiKernel *
use ROOT        * LCG_Interfaces
use AIDA        * LCG_Interfaces -no_auto_imports
use Boost       * LCG_Interfaces -no_auto_imports
use Reflex      * LCG_Interfaces -no_auto_imports
use uuid        * LCG_Interfaces -no_auto_imports
use XercesC     * LCG_Interfaces -no_auto_imports
# ======= own includes =================
apply_pattern install_more_includes more=GaudiUtils
# ======= constituents =================
library GaudiUtilsLib Lib/*.cpp \
        -import=AIDA -import=Boost -no_static
apply_pattern linker_library library=GaudiUtilsLib
# ======= constituents =================
library GaudiUtils component/*.cpp \
        -import=Boost -import=Reflex \
        -import=uuid -import=XercesC -no_static
apply_pattern component_library library=GaudiUtils
# ======= local settings ===============
private
  macro_append ROOT_linkopts " -lHist -lXMLIO "
  macro_append Boost_linkopts  " $(Boost_linkopts_date_time) "
end_private
```

From CMT to CMake (2)

```
# ========================================
package GaudiUtils
version v4r0
# ======= structure ====================
branches GaudiUtils src cmt doc
# ======= dependencies =================
use GaudiKernel *
use ROOT         * LCG_Interfaces
use AIDA         * LCG_Interfaces -no_auto_imports
use Boost        * LCG_Interfaces -no_auto_imports
use Reflex       * LCG_Interfaces -no_auto_imports
use uuid         * LCG_Interfaces -no_auto_imports
use XercesC      * LCG_Interfaces -no_auto_imports
# ======= own includes =================
apply_pattern install_more_includes more=GaudiUtils
# ======= constituents =================
library GaudiUtilsLib Lib/*.cpp \
        -import=AIDA -import=Boost -no_static
apply_pattern linker_library library=GaudiUtilsLib
# ======= constituents =================
library GaudiUtils component/*.cpp \
        -import=Boost -import=Reflex \
        -import=uuid -import=XercesC -no_static
apply_pattern component_library library=GaudiUtils
# ======= local settings ===============
private
  macro_append ROOT_linkopts " -lHist -lXMLIO "
  macro_append Boost_linkopts  " $(Boost_linkopts_date_time) "
end_private
```

```
# ============================
gaudi_subdir(GaudiUtils v4r0)
```

From CMT to CMake (2)

```
# ========================================
package GaudiUtils
version v4r0
# ======= structure ======================
branches GaudiUtils src cmt doc
# ======= dependencies ==================
use GaudiKernel *
use ROOT          * LCG_Interfaces
use AIDA          * LCG_Interfaces -no_auto_imports
use Boost         * LCG_Interfaces -no_auto_imports
use Reflex        * LCG_Interfaces -no_auto_imports
use uuid          * LCG_Interfaces -no_auto_imports
use XercesC       * LCG_Interfaces -no_auto_imports
# ======= own includes ==================
apply_pattern install_more_includes more=GaudiUtils
# ======= constituents ==================
library GaudiUtilsLib Lib/*.cpp \
        -import=AIDA -import=Boost -no_static
apply_pattern linker_library library=GaudiUtilsLib
# ======= constituents ==================
library GaudiUtils component/*.cpp \
        -import=Boost -import=Reflex \
        -import=uuid -import=XercesC -no_static
apply_pattern component_library library=GaudiUtils
# ======= local settings ================
private
  macro_append ROOT_linkopts " -lHist -lXMLIO "
  macro_append Boost_linkopts  " $(Boost_linkopts_date_time) "
end_private
```

```
# ============================
gaudi_subdir(GaudiUtils v4r0)

# ======= dependencies ================
depends_on_subdirs(GaudiKernel)
```

From CMT to CMake (2)

```
# ========================================
package GaudiUtils
version v4r0
# ======= structure ======================
branches GaudiUtils src cmt doc
# ======= dependencies ===================
use GaudiKernel *
use ROOT        * LCG_Interfaces
use AIDA        * LCG_Interfaces -no_auto_imports
use Boost       * LCG_Interfaces -no_auto_imports
use Reflex      * LCG_Interfaces -no_auto_imports
use uuid        * LCG_Interfaces -no_auto_imports
use XercesC     * LCG_Interfaces -no_auto_imports
# ======= own includes ===================
apply_pattern install_more_includes more=GaudiUtils
# ======= constituents ===================
library GaudiUtilsLib Lib/*.cpp \
        -import=AIDA -import=Boost -no_static
apply_pattern linker_library library=GaudiUtilsLib
# ======= constituents ===================
library GaudiUtils component/*.cpp \
        -import=Boost -import=Reflex \
        -import=uuid -import=XercesC -no_static
apply_pattern component_library library=GaudiUtils
# ======= local settings ================
private
  macro_append ROOT_linkopts " -lHist -lXMLIO "
  macro_append Boost_linkopts  " $(Boost_linkopts_date_time) "
end_private
```

```
# ============================
gaudi_subdir(GaudiUtils v4r0)

# ======= dependencies ==================
depends_on_subdirs(GaudiKernel)

find_package(ROOT COMPONENTS RIO Hist XMLIO)
find_package(AIDA)
find_package(Boost COMPONENTS date_time)
find_package(uuid)
find_package(XercesC)
```

From CMT to CMake (2)

```
# ========================================
package GaudiUtils
version v4r0
# ======= structure ======================
branches GaudiUtils src cmt doc
# ======= dependencies ===================
use GaudiKernel *
use ROOT        * LCG_Interfaces
use AIDA        * LCG_Interfaces -no_auto_imports
use Boost       * LCG_Interfaces -no_auto_imports
use Reflex      * LCG_Interfaces -no_auto_imports
use uuid        * LCG_Interfaces -no_auto_imports
use XercesC     * LCG_Interfaces -no_auto_imports
# ======= own includes ==================
apply_pattern install_more_includes more=GaudiUtils
# ======= constituents ==================
library GaudiUtilsLib Lib/*.cpp \
        -import=AIDA -import=Boost -no_static
apply_pattern linker_library library=GaudiUtilsLib
# ======= constituents ==================
library GaudiUtils component/*.cpp \
        -import=Boost -import=Reflex \
        -import=uuid -import=XercesC -no_static
apply_pattern component_library library=GaudiUtils
# ======= local settings ================
private
  macro_append ROOT_linkopts " -lHist -lXMLIO "
  macro_append Boost_linkopts  " $(Boost_linkopts_date_time) "
end_private
```

```
# ============================
gaudi_subdir(GaudiUtils v4r0)

# ======= dependencies ==================
depends_on_subdirs(GaudiKernel)

find_package(ROOT COMPONENTS RIO Hist XMLIO)
find_package(AIDA)
find_package(Boost COMPONENTS date_time)
find_package(uuid)
find_package(XercesC)

# ======= libraries =======
gaudi_add_library(GaudiUtilsLib Lib/*.cpp
                  LINK_LIBRARIES GaudiKernel Boost ROOT
                  INCLUDE_DIRS AIDA Boost ROOT
                  PUBLIC_HEADERS GaudiUtils)
```

From CMT to CMake (2)

```
# ========================================
package GaudiUtils
version v4r0
# ======= structure ======================
branches GaudiUtils src cmt doc
# ======= dependencies ===================
use GaudiKernel *
use ROOT        * LCG_Interfaces
use AIDA        * LCG_Interfaces -no_auto_imports
use Boost       * LCG_Interfaces -no_auto_imports
use Reflex      * LCG_Interfaces -no_auto_imports
use uuid        * LCG_Interfaces -no_auto_imports
use XercesC     * LCG_Interfaces -no_auto_imports
# ======= own includes ===================
apply_pattern install_more_includes more=GaudiUtils
# ======= constituents ===================
library GaudiUtilsLib Lib/*.cpp \
        -import=AIDA -import=Boost -no_static
apply_pattern linker_library library=GaudiUtilsLib
# ======= constituents ===================
library GaudiUtils component/*.cpp \
        -import=Boost -import=Reflex \
        -import=uuid -import=XercesC -no_static
apply_pattern component_library library=GaudiUtils
# ======= local settings =================
private
  macro_append ROOT_linkopts " -lHist -lXMLIO "
  macro_append Boost_linkopts  " $(Boost_linkopts_date_time) "
end_private
```

```
# ============================
gaudi_subdir(GaudiUtils v4r0)

# ======= dependencies ==================
depends_on_subdirs(GaudiKernel)

find_package(ROOT COMPONENTS RIO Hist XMLIO)
find_package(AIDA)
find_package(Boost COMPONENTS date_time)
find_package(uuid)
find_package(XercesC)

# ======= libraries =======
gaudi_add_library(GaudiUtilsLib Lib/*.cpp
                  LINK_LIBRARIES GaudiKernel Boost ROOT
                  INCLUDE_DIRS AIDA Boost ROOT
                  PUBLIC_HEADERS GaudiUtils)
gaudi_add_module(GaudiUtils component/*.cpp
                 LINK_LIBRARIES GaudiUtilsLib uuid XercesC
                 INCLUDE_DIRS uuid XercesC)
```

From CMT to CMake (2)

```
# ======================================
package GaudiUtils
version v4r0
# ======= structure ====================
branches GaudiUtils src cmt doc
# ======= dependencies =================
use GaudiKernel *
use ROOT        * LCG_Interfaces
use AIDA        * LCG_Interfaces -no_auto_imports
use Boost       * LCG_Interfaces -no_auto_imports
use Reflex      * LCG_Interfaces -no_auto_imports
use uuid        * LCG_Interfaces -no_auto_imports
use XercesC     * LCG_Interfaces -no_auto_imports
# ======= own includes =================
apply_pattern install_more_includes more=GaudiUtils
# ======= constituents =================
library GaudiUtilsLib Lib/*.cpp \
        -import=AIDA -import=Boost -no_static
apply_pattern linker_library library=GaudiUtilsLib
# ======= constituents =================
library GaudiUtils component/*.cpp \
        -import=Boost -import=Reflex \
        -import=uuid -import=XercesC -no_static
apply_pattern component_library library=GaudiUtils
# ======= local settings ===============
private
  macro_append ROOT_linkopts " -lHist -lXMLIO "
  macro_append Boost_linkopts  " $(Boost_linkopts_date_time) "
end_private
```

```
# ============================
gaudi_subdir(GaudiUtils v4r0)

# ======= dependencies =================
depends_on_subdirs(GaudiKernel)

find_package(ROOT COMPONENTS RIO Hist XMLIO)
find_package(AIDA)
find_package(Boost COMPONENTS date_time)
find_package(uuid)
find_package(XercesC)

# ======= libraries =======
gaudi_add_library(GaudiUtilsLib Lib/*.cpp
                  LINK_LIBRARIES GaudiKernel Boost ROOT
                  INCLUDE_DIRS AIDA Boost ROOT
                  PUBLIC_HEADERS GaudiUtils)
gaudi_add_module(GaudiUtils component/*.cpp
                   LINK_LIBRARIES GaudiUtilsLib uuid XercesC
                   INCLUDE_DIRS uuid XercesC)
```

Outline

Conclusions

- CMT is a valid product, but with limits
- CMake is not meant to address our use case. . .
- . . . but it is powerful enough to be adapted
- Developed a CMake-based build framework
  - can replace CMT in LHCb use
  - better performance
  - will be adopted by after some more validation

Conclusions

- CMT is a valid product, but with limits
- CMake is not meant to address our use case. . .
- . . . but it is powerful enough to be adapted
- Developed a CMake-based build framework
  - can replace CMT in LHCb use
  - better performance
  - will be adopted by after some more validation

Conclusions

- CMT is a valid product, but with limits
- CMake is not meant to address our use case...
- ... but it is powerful enough to be adapted
- Developed a CMake-based build framework
  - can replace CMT in LHCb use
  - better performance
  - will be adopted by after some more validation

Conclusions

- CMT is a valid product, but with limits
- CMake is not meant to address our use case...
- ... but it is powerful enough to be adapted
- Developed a CMake-based build framework
  - can replace CMT in LHCb use
  - better performance
  - will be adopted by after some more validation