CMAKE
AN INTRODUCTION

Graduiertenkolleg EMS
Robert Jakob

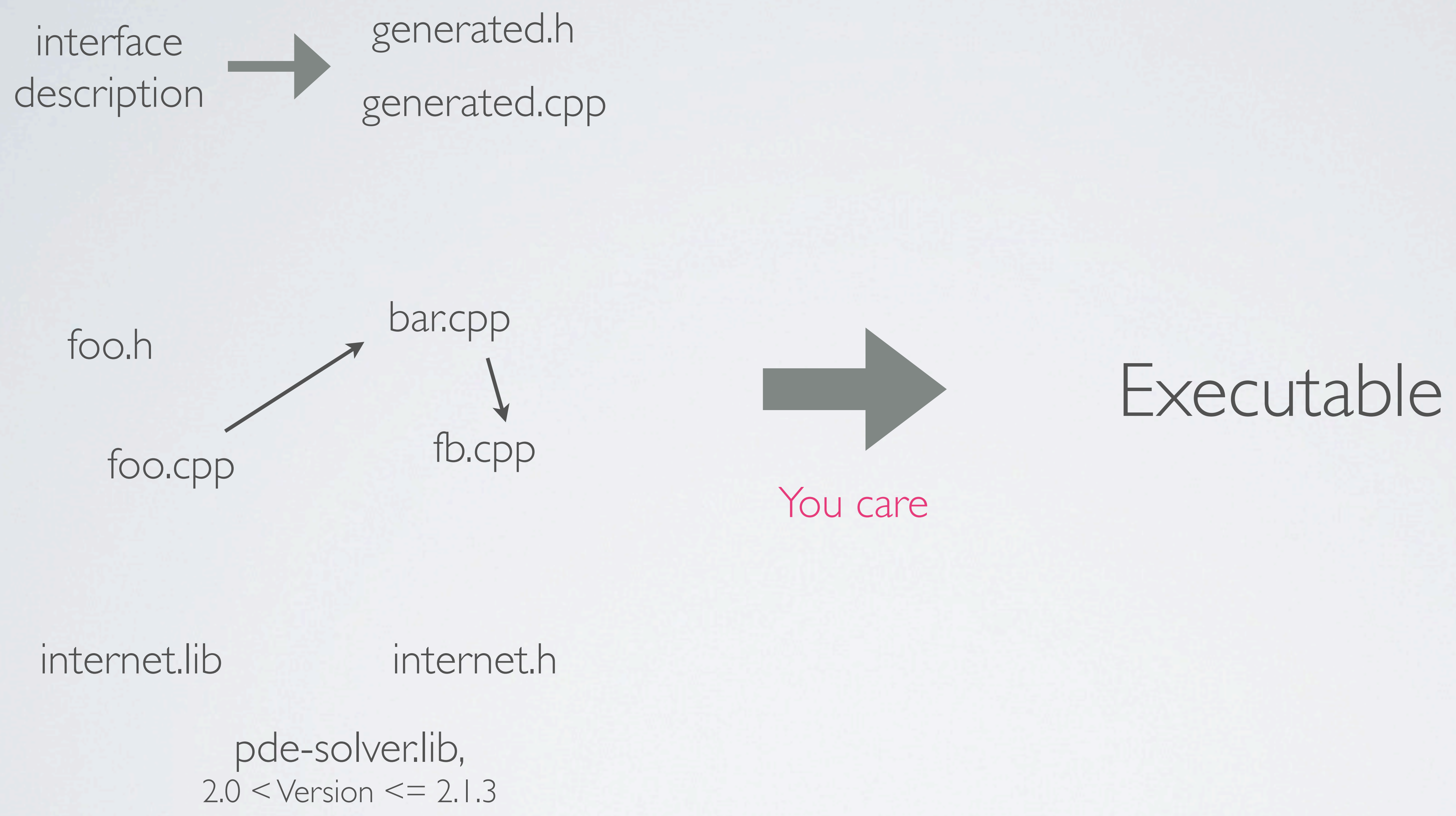
GOAL

Source $\longrightarrow$ Executable

I don't care

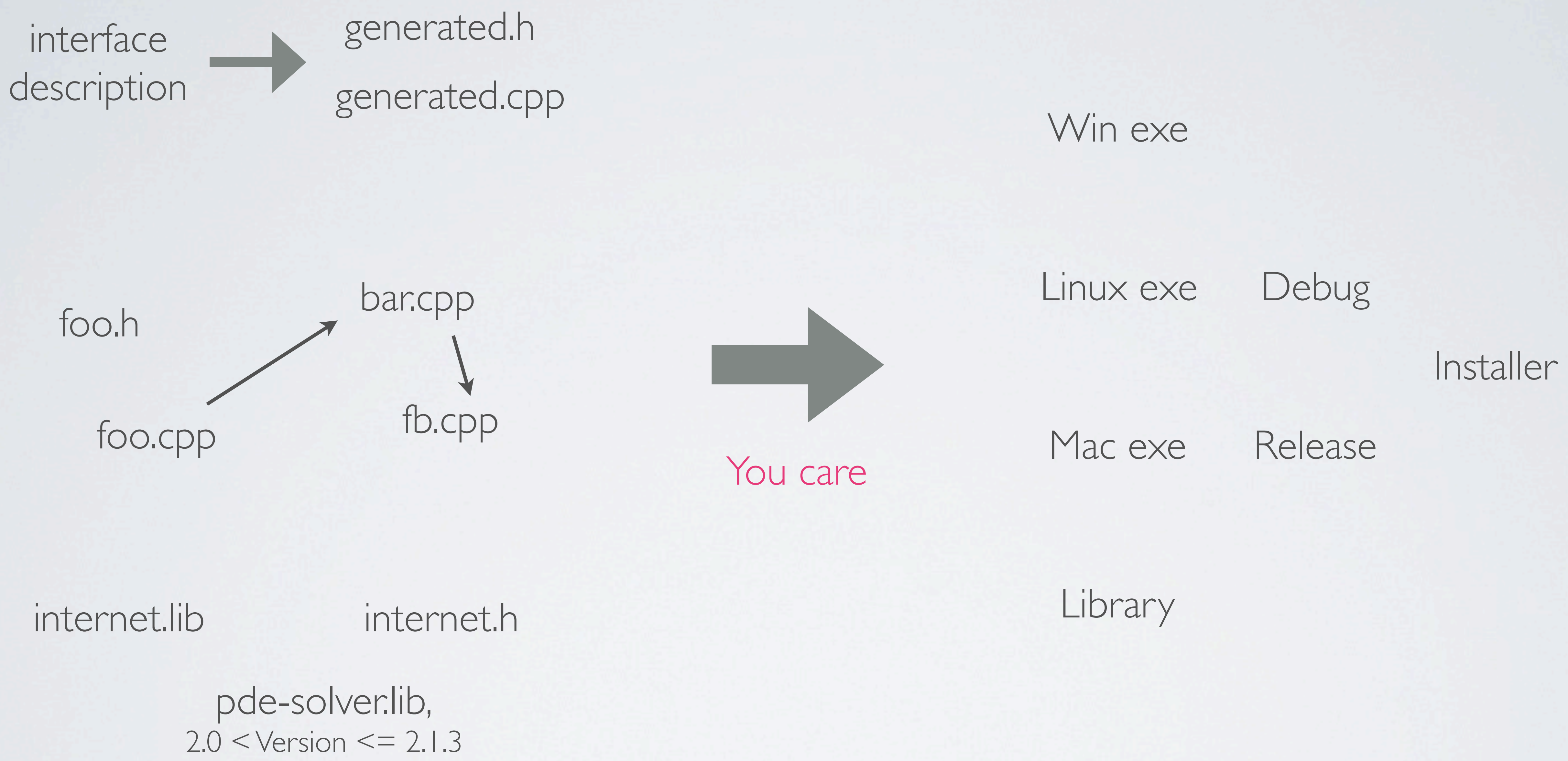
You care

GOAL

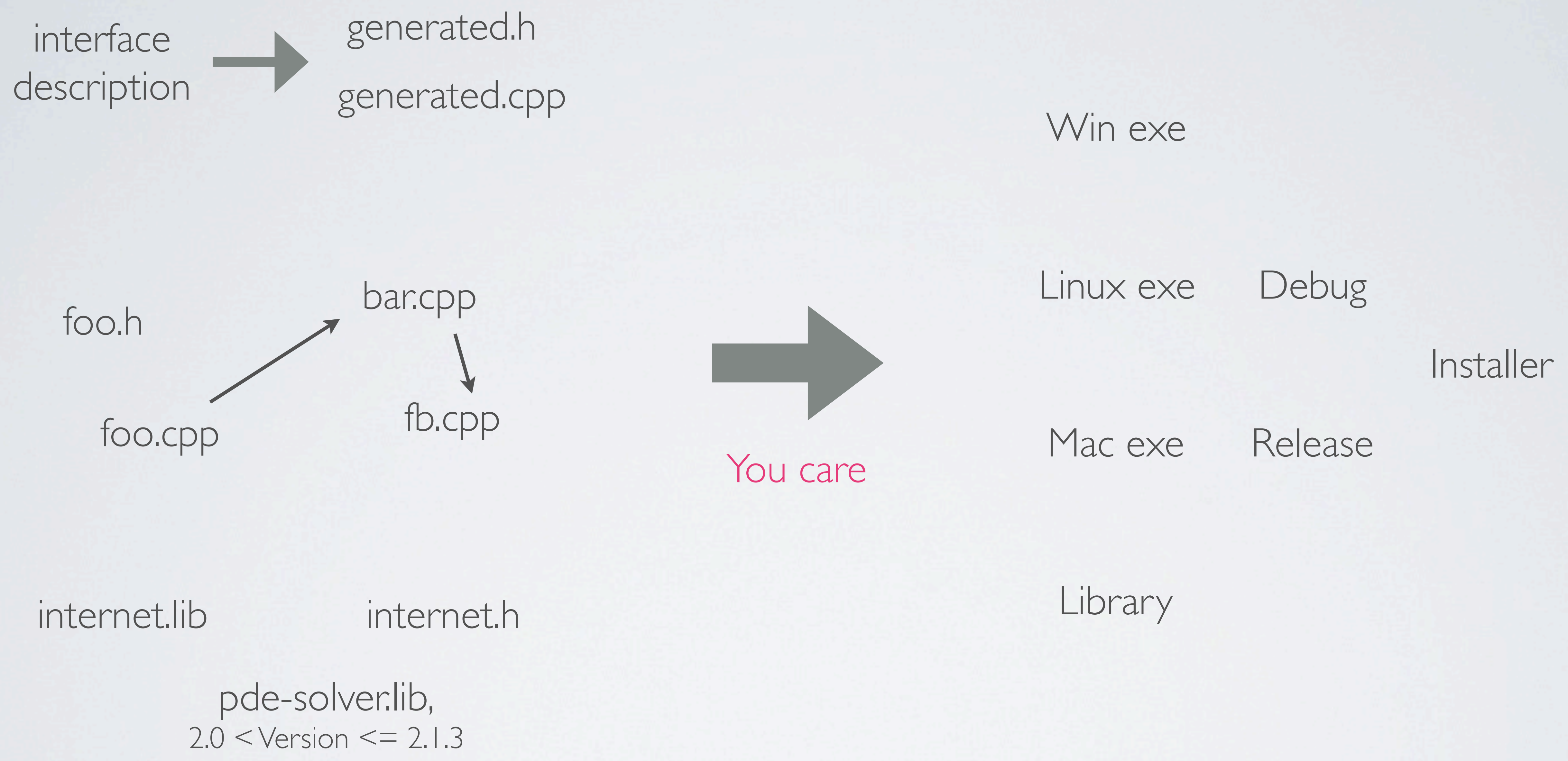


interface description
generated.h
generated.cpp
foo.h
bar.cpp
foo.cpp
fb.cpp
You care
Executable
internet.lib
internet.h
pde-solver.lib,
2.0 < Version <= 2.1.3

GOAL


interface description
generated.h
generated.cpp
foo.h
bar.cpp
foo.cpp
fb.cpp
internet.lib
internet.h
pde-solver.lib,
2.0 < Version <= 2.1.3
You care
Win exe
Linux exe
Debug
Installer
Mac exe
Release
Library

HELP! GOAL!


interface description
generated.h
generated.cpp
foo.h
bar.cpp
foo.cpp
fb.cpp
internet.lib
internet.h
pde-solver.lib,
2.0 < Version <= 2.1.3
You care
Win exe
Linux exe
Debug
Installer
Mac exe
Release
Library

SOLUTIONS

- **GNU Build system** (aka Autotools)
  ./configure && make && make install
- **qmake**
  Qt by Nokia‘s build system
- **Scons**
  Python-based build system
- **cmake**
  cross-plattform make system

WHAT IS IT?

CMake is a build-process management tool

- Platform independent
- Supports various output formats
- Dependencies
- Libraries

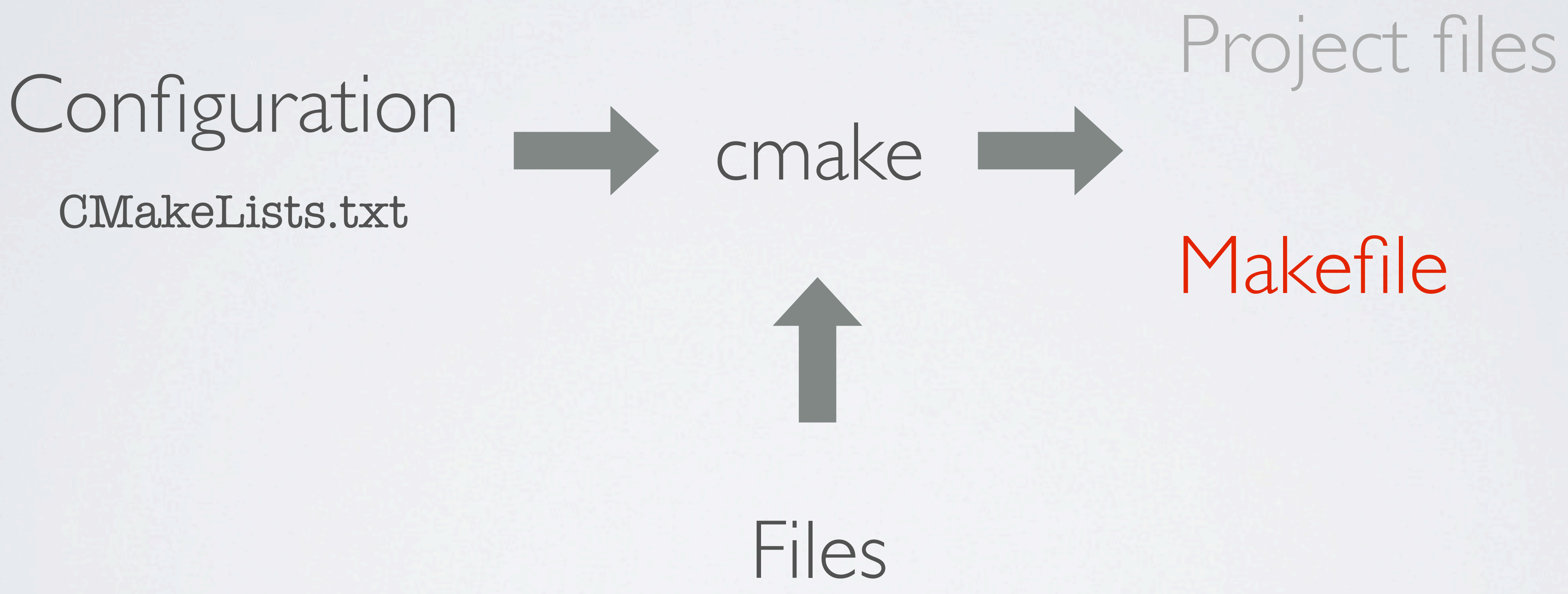
WORKFLOW

$$\begin{tikzcd} \text{Configuration} \arrow[r] & \text{cmake} \arrow[r] & \text{Makefile} \\ \texttt{CMakeLists.txt} & \text{Files} \arrow[u] & \end{tikzcd}$$

WORKFLOW

$$\begin{tikzcd}
\begin{array}{c}\text{Configuration}\\ \texttt{CMakeLists.txt}\end{array} \arrow[r] & \text{cmake} \arrow[r] & \begin{array}{l}\text{Project files}\\ \text{Makefile}\end{array} \\
 & \text{Files} \arrow[u] &
\end{tikzcd}$$

WORKFLOW



Configuration
CMakeLists.txt
cmake
Project files
Makefile
Files

MAKEFILES (IDEA)

- Makefiles execute commands depending on some conditions
- Makefiles consist of targets, dependencies, and commands:
```
target: {dependency}
   {cmd}
```
-
```
foo.exe: foo.c another_target
  compile --input=foo.c --output=foo.exe
```
- make foo.exe
  - last_change (foo.exe) < last_change(foo.c): compile
  - last_change (foo.exe) => last_change(foo.c): nothing

THE BASICS

EXAMPLE PROJECT

```
#include <stdio.h>
#include <string.h>
#include <bzlib.h>
#include "adder.h"

int main(int argc, char* argv[]) {
        int bzError = 0;
        char buffer[51];
        int result;
        FILE *tbz2File = fopen(argv[1], "rb");
        memset(&buffer,0,51);

        BZFILE *bz = BZ2_bzReadOpen(&bzError, tbz2File, 0, 0, 0, 0);
        BZ2_bzRead(&bzError, bz, buffer, 50);

        printf("%50s\n", buffer);

        result = add(buffer);
        printf("Result: %d\n", result);

        fclose(tbz2File);

        return 0;
}
```

EXAMPLE PROJECT

```
#include <stdio.h>
#include <string.h>
#include <bzlib.h>
#include "adder.h"
```

Dependencies

```
int main(int argc, char* argv[]) {
```

Starting point of program

```
        int bzError = 0;
        char buffer[51];
        int result;
        FILE *tbz2File = fopen(argv[1], "rb");
        memset(&buffer,0,51);

        BZFILE *bz = BZ2_bzReadOpen(&bzError, tbz2File, 0, 0, 0, 0);
        BZ2_bzRead(&bzError, bz, buffer, 50);

        printf("%50s\n", buffer);

        result = add(buffer);
        printf("Result: %d\n", result);

        fclose(tbz2File);

        return 0;
}
```

Program code

MANUAL COMPILATION

- Compilation command line:

  `gcc -g -c adder.c` produces `adder.o`

  `gcc -g -c main.c` produces `main.o`

- Linking

  `gcc -g adder.o main.o -lbz2` produces a.out

- You don't want to run all this steps manually

DEPENDENCIES

- Main.c depends on adder.h
- Change adder.h means recompilation of main.c
- And linking of all object files

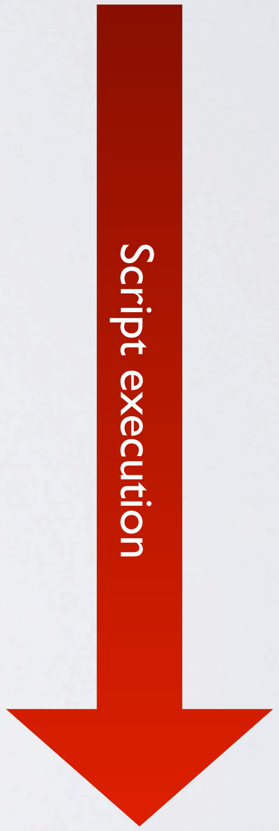
ABOUT DIRECTORIES

- Good directory structure

```
├── build-debug
├── build-release
├── CMakeLists.txt
├── src
│   ├── adder.c
│   ├── adder.h
│   └── main.c
└── tests
    └── test.txt.bz2
```

CMAKELISTS.TXT

```
project(mygitness)
cmake_minimum_required(VERSION 2.6)

add_definitions(-Wall)

include_directories(${CMAKE_CURRENT_BINARY_DIR})

set(SOURCE
  src/main.c
  src/adder.c)

add_executable(cmakeexample ${SOURCE})

find_package (BZip2)
include_directories(${BZIP_INCLUDE_DIRS})

target_link_libraries (cmakeexample
      ${BZIP2_LIBRARIES})
```



CMAKELISTS.TXT

```
project(mygitness)
cmake_minimum_required(VERSION 2.6)

add_definitions(-Wall)

include_directories(${CMAKE_CURRENT_BINARY_DIR})

set(SOURCE
  src/main.c
  src/adder.c)

add_executable(cmakeexample ${SOURCE})

find_package (BZip2)
include_directories(${BZIP_INCLUDE_DIRS})

target_link_libraries (cmakeexample
        ${BZIP2_LIBRARIES})
```

Preamble

Source file definitions

Defining targets

Libraries to link to

COMMANDS

- Basic syntax
  `command(args...)`
- Project definition
  `project (name [CXX] [C] [JAVA])`
- Setting a variable
  `set(VARIABLE 2)`
- Using a variable
  `${VARIABLE}`

FLOW CONTROL

- Conditionals

```
if (FOO)
  # comments
else (FOO)
  # comments
endif (FOO)
```

- If, else, and endif need argument! (may be empty)
- FOO is true if it is 1,ON,YES,TRUE,Y

CONDITIONAL

- if(var)
- if(NOT var)
- if(var AND var)
- if(var OR var)
- if(DEFINED var)
- if(EXISTS filename)
- if(EXISTS dirname)
- if(n1 IS_NEWER_THAN n2)
- if(var MATCHES regex)
- if(1 LESS 3)
- if(FOO STRLESS BAR)

LOOPS / MESSAGES

```
set(SRC adder.c main.c)

message(„Printing all source files:“)

if(NOT DEFINED SRC)
  message (FATAL_ERROR „No sources defined“)
endif ()

foreach (file ${SRC})
   message(${file})
endforeach ()

message(„Done printing all source files“)
```

- There is also a while loop

TARGETS

- Defining a new target of type executable
  `add_executable(foo.exe ${SRC})`

- Defining a new target of type library
  ```
  add_library(foo STATIC foo1.c foo2.c)
  add_library(foo SHARED foo1.c foo2.c)
  ```

- Defining an arbitrary target
  `add_custom_target(...)`

INCLUDE DIRECTORIES

- Add additional include directories
  `include_directories(INCLUDE_DIR)`
- Add the output build directory (e.g. generated files in Qt)
  `include_directories(${CMAKE_CURRENT_BINARY_DIR})`
- Can be called multiple times and appends to the include dirs.

LIBRARIES

- Linking to libraries is simple
  `target_link_libraries(foo path_to_lib1 path_to_lib2)`

- How to get the path to the library?

FINDING LIBRARIES

- Looking for the TCL Library

```
find_library (TCL_LIBRARY
   NAMES tcl tcl84 tcl83 tcl82 tcl80
   PATHS /usr/lib /usr/local/lib)

if (TCL_LIBRARY)
  target_link_library(fooexe ${TCL_LIBRARY})
endif ()
```

PREDEFINED MODULES

- ALSA
- Armadillo
- ASPELL
- AVIFile
- BISON
- BLAS
- Boost
- Bullet
- **BZip2**
- CABLE
- Coin3D
- CUDA
- Cups
- CURL
- Curses
- CVS
- CxxTest
- Cygwin
- Dart
- DCMTK
- DevIL
- Doxygen
- EXPAT
- FLEX
- FLTK2
- FLTK
- Freetype
- GCCXML
- GDAL
- Gettext
- GIF
- Git
- GLU
- GLUT
- Gnuplot
- GnuTLS
- GTest
- GTK2
- GTK
- HDF5
- HSPELL
- HTMLHelp
- ImageMagick
- ITK
- Jasper
- Java
- JNI
- JPEG
- KDE3
- KDE4
- LAPACK
- LATEX
- LibArchive
- LibXml2
- LibXslt
- Lua50
- Lua51
- Matlab
- MFC
- Motif
- MPEG2
- MPEG
- MPI
- OpenAL
- OpenGL
- OpenMP
- OpenSceneGraph
- OpenSSL
- OpenThreads
- osgAnimation
- osg
- osgDB
- osg_functions
- osgFX
- osgGA
- osgIntrospection
- osgManipulator
- osgParticle
- osgProducer
- osgShadow
- osgSim
- osgTerrain
- osgText
- osgUtil
- osgViewer
- osgVolume
- osgWidget
- PackageHandleStandardArgs
- PackageMessage
- Perl
- PerlLibs
- PHP4
- PhysFS
- Pike
- PkgConfig
- PNG
- PostgreSQL
- Producer
- Protobuf
- PythonInterp
- PythonLibs
- QJSON
- Qt3
- Qt4
- Qt
- QuickTime
- RTI
- Ruby
- SDL
- SDL_image
- SDL_mixer
- SDL_net
- SDL_sound
- SDL_ttf
- SelfPackers
- Squish
- Subversion
- SWIG
- TCL
- Tclsh
- TclStub
- Threads
- TIFF
- UnixCommands
- VTK
- Wget
- Wish
- wxWidgets
- wxWindows
- X11
- XMLRPC
- ZLIB

USE PREDEFINED MODULES

- Predefined „find“-modules search for the libraries and define variables

```
#  BZIP2_FOUND - system has BZip2
#  BZIP2_INCLUDE_DIR - the BZip2 include directory
#  BZIP2_LIBRARIES - Link these to use BZip2
#  BZIP2_NEED_PREFIX - this is set if the functions are prefixed with

BZ2_find_package (BZip2)
include_directories(${BZIP_INCLUDE_DIRS})

target_link_libraries (cmakeexample
     ${BZIP2_LIBRARIES})
```

CMAKELISTS.TXT

```
project(mygitness)
cmake_minimum_required(VERSION 2.6)

add_definitions(-Wall)

include_directories(${CMAKE_CURRENT_BINARY_DIR})

set(SOURCE
  src/main.c
  src/adder.c)

add_executable(cmakeexample ${SOURCE})

find_package (BZip2)
include_directories(${BZIP_INCLUDE_DIRS})

target_link_libraries (cmakeexample
        ${BZIP2_LIBRARIES})
```

BUILD PROCESS

```
. $ cd build-debug/
./build-debug $ cmake ../
-- The C compiler identification is GNU
-- The CXX compiler identification is GNU
-- Check for working C compiler: /usr/bin/gcc
-- Check for working C compiler: /usr/bin/gcc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Found BZip2: /usr/lib/libbz2.so
-- Looking for BZ2_bzCompressInit in /usr/lib/libbz2.so
-- Looking for BZ2_bzCompressInit in /usr/lib/libbz2.so - found
-- Configuring done
-- Generating done
-- Build files have been written to: /home/jakobro/projects/cmake-example-app/build-debug
./build-debug $ make
Scanning dependencies of target cmakeexample
[ 50%] Building C object CMakeFiles/cmakeexample.dir/src/main.c.o
[100%] Building C object CMakeFiles/cmakeexample.dir/src/adder.c.o
Linking C executable cmakeexample
[100%] Built target cmakeexample
```

BUILD PROCESS

```
./build-debug $ change ../src/main.c
./build-debug $ make
Scanning dependencies of target cmakeexample
[ 50%] Building C object CMakeFiles/cmakeexample.dir/src/main.c.o
Linking C executable cmakeexample
[100%] Built target cmakeexample
```

BUILD PROCESS

- When do we have to call cmake again?
  - Normally, no call to cmake necessary
  - Not even if we change something inside

```
./build-debug $ change ../CMakeLists.txt
./build-debug $ make
-- Configuring done
-- Generating done
-- Build files have been written to: /home/jakobro/projects/cmake-example-app/build-debug
[ 50%] Building C object CMakeFiles/cmakeexample.dir/src/main.c.o
[100%] Building C object CMakeFiles/cmakeexample.dir/src/adder.c.o
Linking C executable cmakeexample
[100%] Built target cmakeexample
```

BUILD PROCESS

- cmake has an internal cache (build-debug/CMakeCache.txt)
- If changing cached variables, makefile is not recreated!
- Solution:

```
./build-debug $ make rebuild_cache
Running CMake to regenerate build system...
-- Configuring done
-- Generating done
-- Build files have been written to: build-debug
```

PROBLEMS

- If you want to see what cmake really does
  cmake --debug-output
- If you want to see the commands make runs
  make VERBOSE=1

ERROR SOLUTION CHAIN

- Error when running make
  - Try: make clean && make
  - make rebuild_cache
  - Try: rm -R build-debug/
  - Try: mkdir build-debug && cmake ../
- Error when running cmake
  - cmake --debug-output ../
  - cmake --trace ../ (This will get you lots of output)

ADVANCED STUFF

SUBCONFIGS

- If you have submodules and want them to have an extra config

```
├── build-debug
├── build-release
├── CMakeLists.txt      // toplevel config
├── src
│   ├── adder.c
│   ├── adder.h
│   └── main.c
└── mymathmodule
    ├── CMakeLists.txt     // subconfig
    └── math.cpp
```

SUBCONFIGS

- Two possibilities:
  - Subconfig creates its own executable/library which is used by toplevel config
  - Only describes source and header files and toplevel adds them to its build process

TOPLEVEL CONFIGURATION

```
   set(SOURCE
  ${CMAKE_CURRENT_SOURCE_DIR}/main.cpp
 )

add_subdirectory("${PROJECT_SOURCE_DIR}/mymathmodule")

add_executable(fooexec ${SOURCE} ${HEADERS})
```

SUBCONFIG

```
set(SOURCE
  ${SOURCE}
  ${CMAKE_CURRENT_SOURCE_DIR}/file1.cpp
  ${CMAKE_CURRENT_SOURCE_DIR}/file2.cpp
  PARENT_SCOPE
)
set(HEADERS
  ${HEADERS}
  ${CMAKE_CURRENT_SOURCE_DIR}/file1.hpp
  ${CMAKE_CURRENT_SOURCE_DIR}/file2.hpp
  PARENT_SCOPE
)
```

DEBUG/RELEASE BUILDS

- Either give the cmake process a variable:
  - `cmake -DCMAKE_BUILD_TYPE=Debug`
  - `cmake -DCMAKE_BUILD_TYPE=Release`
- or specify it in the config

```
SET(CMAKE_BUILD_TYPE Debug)
```

OPTIONS

- User-definable options
  - building optional parts of the application
  - using special math library
- Shows up in GUI

```
option(BUILD_SPECIAL_PART „Build special part“ OFF)

$ cmake -DBUILD_SPECIAL_PART=ON
```

CONFIGURE FILE

- Preprocessor definitions from cmake to Code?

```
#ifdef BUILD_SPECIAL_PART
  ...
#endif
```

CONFIGURE FILE

- Copy file from in_file to out_file and replace all variables with their values:
```
configure_file(„{$PROJECT_SOURCE_DIR}/configure.h.in“
               „{$PROJECT_BINARY_DIR}/configure.h“)
```
- Configure.h.in:
```
#cmakedefine BUILD_SPECIAL_PART
```
- Configure.h:
```
#define BUILD_SPECIAL_PART   or   /* #define BUILD_SPECIAL_PART */
```
- Access to values of variables
```
@VARNAME@
```

BEYOND CMAKE

- CPack
  Installer creation
- CTest
  Large test framework
- LaTeX
  http://www.cmake.org/Wiki/CMake_FAQ#How_do_I_use_CMake_to_build_LaTeX_documents.3F

REFERENCES

- Martin and Hoffmann: Mastering CMake
  (Available in our library)
- CMake useful variables
  http://www.cmake.org/Wiki/CMake_Useful_Variables
- FAQ
  http://www.cmake.org/Wiki/CMake_FAQ
- The CMake documentation
  http://www.cmake.org/cmake/help/documentation.html

Questions ?